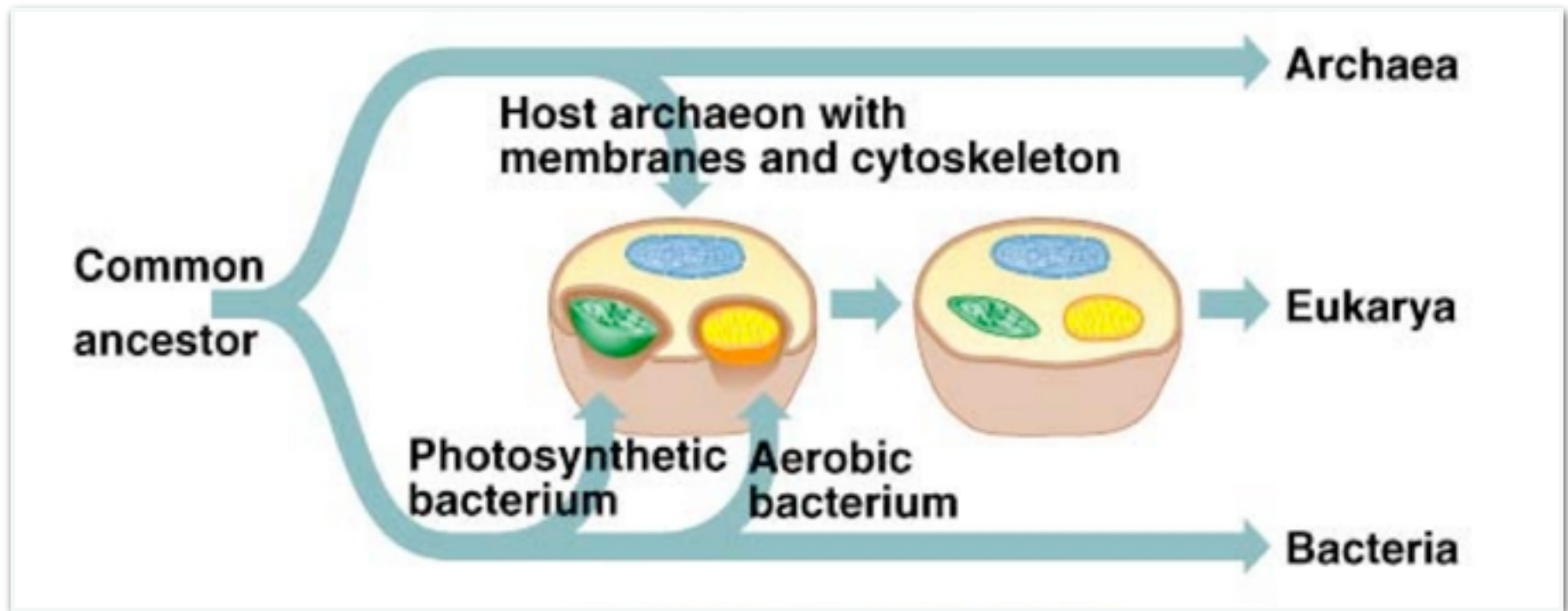


# Molekulare Evolution & Bioinformatik

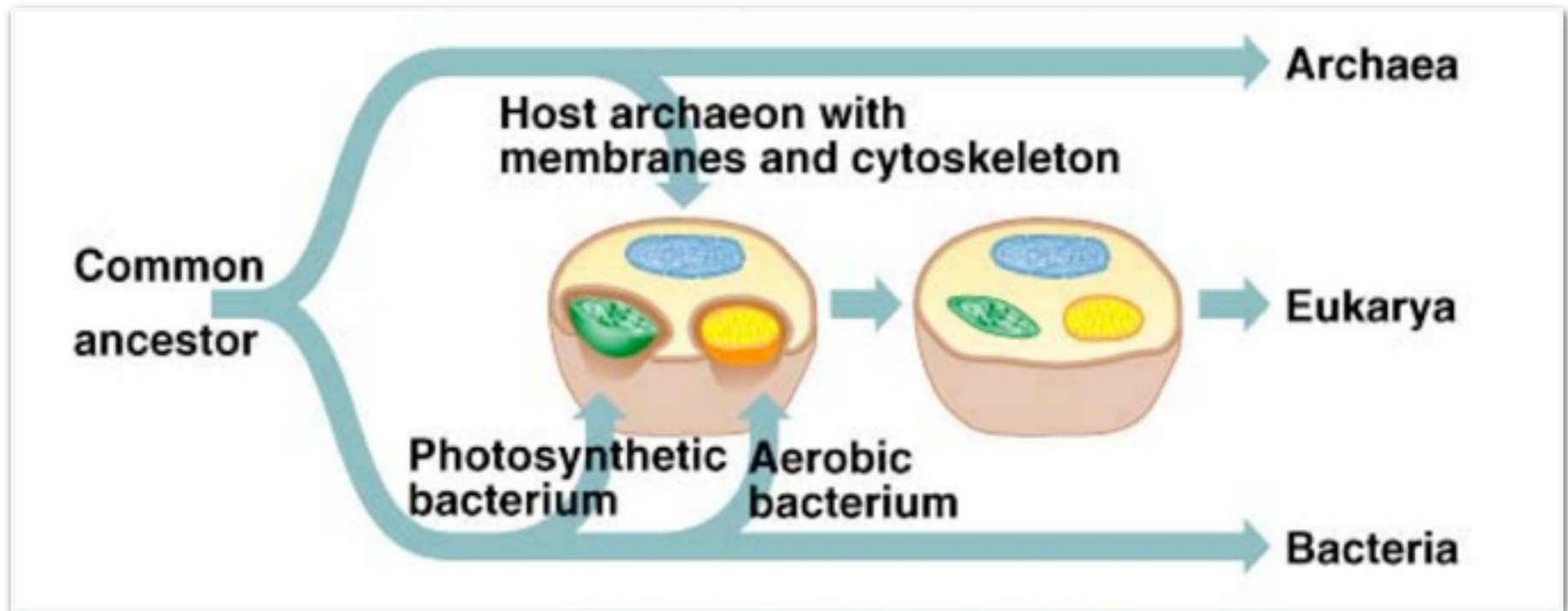
## Part1: Whole Genome Shotgun analyses



**The proposed evolutionary relationships of contemporary living organisms**

# Molekulare Evolution & Bioinformatik

## Part1: Whole Genome Shotgun analyses

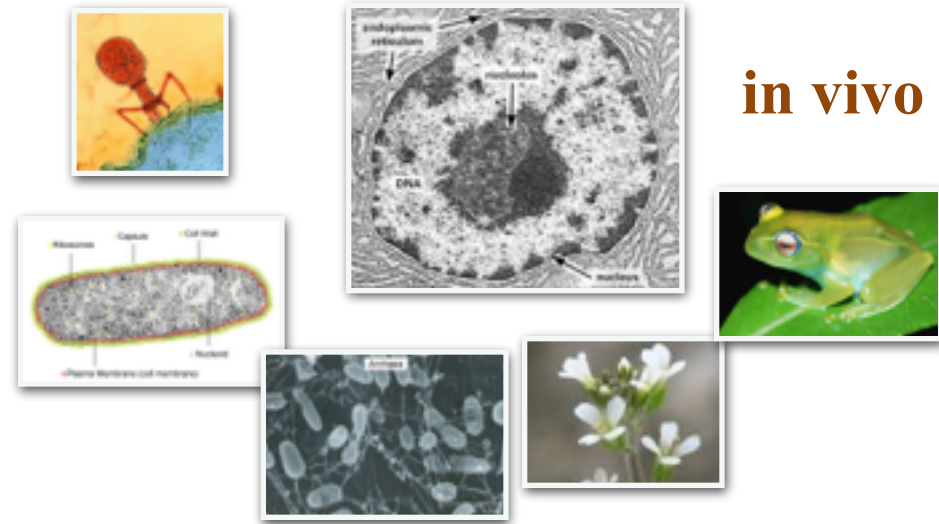


**The proposed evolutionary relationships of contemporary living organisms**

What about viruses?

We encounter DNA in three forms

# We encounter DNA in three forms

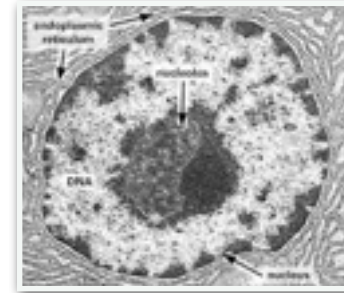


**in vivo**

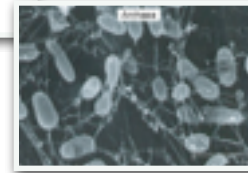
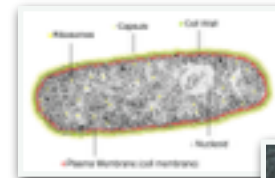
Archaea: [http://teachoceanscience.net/teaching\\_resources/education\\_modules/marine\\_bacteria/learn\\_about/](http://teachoceanscience.net/teaching_resources/education_modules/marine_bacteria/learn_about/)  
Virus: [jonlieffmd.com](http://jonlieffmd.com)  
Bacterium: <http://dtc.pima.edu>  
Arabidopsis: <http://de.wikipedia.org/wiki/Acker-Schmalwand>  
Frog: <http://de.wikipedia.org/wiki/Froschlurche>

# We encounter DNA in three forms

**in vitro**



**in vivo**



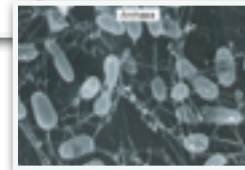
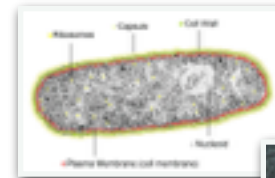
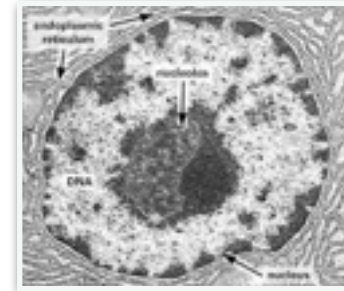
Archaea: [http://teachoceanscience.net/teaching\\_resources/education\\_modules/marine\\_bacteria/learn\\_about/](http://teachoceanscience.net/teaching_resources/education_modules/marine_bacteria/learn_about/)  
Virus: [jonlieffmd.com](http://jonlieffmd.com)  
Bacterium: <http://dtc.pima.edu>  
Arabidopsis: <http://de.wikipedia.org/wiki/Acker-Schmalwand>  
Frog: <http://de.wikipedia.org/wiki/Froschlurche>

# We encounter DNA in three forms

**in vitro**



**in vivo**



**in silico**

```
@Clagr-170543-2741/1
CAGAGAATAAATTCATCTTCGCCAGCTACAAGTAGCTTTGAARTGGACTGGAATGGAGAAAGGGGATCATCTC
AACTTCTGGAAGAGGGCCGACAGCTGGTCTADAAAGGCCDCTGAAGAGAGTCCGACAGACTCTAGTGAAGT
GCAGTTTACTTATTTAACACGCTTTGTTTTTGTAAACAAAGACGAGTACAGGCAGGAGGAAGTACGGG
TATACAGAGCCGATACCACTGGAGCT
+
?A????B?DDA<DBDDGAGC/GIHAH/H/IEFIIIHIIHFHIIII>HI?HHHDF-DFEGEIFHHIE7IIH
IIHIIHFHIIIEHIIHBHHHHHGGIHHIHFHG;IEGGHH=FGHGGEGEHHDGED?G@FAGICFCG4GE?)GE
GEGG@HG?CEEEFCE;E(8FFC<GGECHA'GFG8E,6C?CGFFAGGC;GEFFFG?E*GEAGEHHESHECGGGE
C;ACECAGGGCEGEG?GEEEC(E;EG*
@Clagr-170541-2741/1
TATTTAAGAATAAGATAATAAATATATTTAAGAATAGTGAATCTATTAATAAATTATTATAGAATAAATAA
TTCATTTCTATATCTTAATAAAGTACTTACTTAGTATTATCTTTATTAATTTATATAATAAGGAAGATATTA
TAGTTAAAGAAATATGTCATAGTGAAGGCATAAGCGATGAGCTAATATGGCTATCAAGCTCTAAACAGCTAT
GTGATAACATAAAGCGATGTTCTAATGG
+
?<???B?B0DD<@DDGGGGFFHIFIFIHIIIGIHHIHIHIIHHIGIHHECHGCICIEHH=IFHHF58II
IHHCIHIIII@HFDFIHFHIIHIEHIFGHIHF@HHGFHGEHFFHDGGGGHFFEGGHGGEGG?GGGF*=GDG
GGFGD6EGGEC;?GGGCFEEEE)GG+GECG<G?GHAEG(FG;GG*FEC;GFE<FEGFEAG3DFACFEEEE;CE
G.EEEGE?CGCGC;EGCAGFGGGECEGG
```

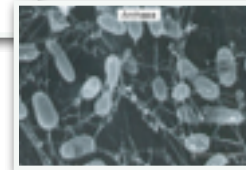
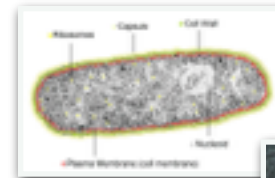
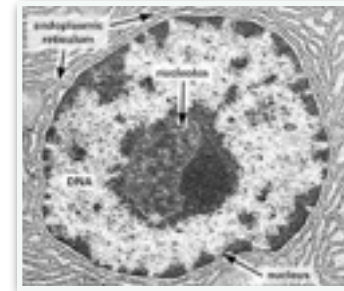
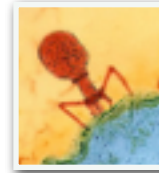
- Archaea: [http://teachoceanscience.net/teaching\\_resources/education\\_modules/marine\\_bacteria/learn\\_about/](http://teachoceanscience.net/teaching_resources/education_modules/marine_bacteria/learn_about/)
- Virus: [jonlieffmd.com](http://jonlieffmd.com)
- Bacterium: <http://drc.pima.edu>
- Arabidopsis: <http://de.wikipedia.org/wiki/Acker-Schmalwand>
- Frog: <http://de.wikipedia.org/wiki/Froschlurche>

# Genome Sequencing

**in vitro**



**in vivo**



**in silico**

```
@Clagr-170543-2741/1
CAGAGAATAAATTCATCTTCGCCAGCTACAAGTAGCTTTGAARTGGACTGGAATGGAGAAAGGGGATCATCTC
AACTTCTGGAAGAGGGCCGACAGCTGGTCTADAAAGGCCDCTGAAGAGAGTCCGACAGACTCTAGTGAAGT
GCAGTTTACTTATTTAACACGCTTTGTTTTTGTAAACAAAGACGAGTACAGGCAGGAGGAAGTACGGG
TATACAGAGCCGATACCCTGGAGCT
+
?A????B?DDA<DBDDGAGC/GIHAH/H/IEFIIIHIIHFIHIIII>HI?HHHDF-DFEGEIFHHIE7IIH
IIHIIHFIHIIHIIHBBHHHHHGIHIIHFHG;IEGGHH=FGHGGEGEHHDGEB?G@FAGICFCG4GE?)GE
GEGG@HG?CEEEFCE;E(8FFC<GGECHA'GFG8E,6C?CGFFAGGC;GEFFFG?E*GEAGEHHE6MECGGGE
C;ACECAGGGCEGEG?GEEEC(E;EG*
@Clagr-170541-2741/1
TATTTTAAAGATAAGATAATAAATATATTTAAGAATAGTGAATCTATTAATAAATTATTATAGAATAAATAA
TTCATTTCTATATCTTAATAAAGTACTTACTTAGTATTATCTTTATTAATTTATATAATAAGGAAGATATTA
TAGTTAAAGAAATATGTCATAGTGAAGGCATAAGCGATGAGCTAATATGGCTATCAAGCTCTAAACAGCTAT
GTGATAACATAAAGCGATGTTCTAATGG
+
?<????B?BDDDD<@DDGGGGFFHIFIFIHIIIGIHHIHIHIIHIIHIGIHHECHGCICIEHH=IFHHF58II
IHHCIHIIII@HFDFFIHIFHIIHIEHIFGHIHF@HHGFHGEHFFHDGGGGHFFEGGHGGEGG?GGGF*=GDG
GGFGD6EGGEC;?GGGGCFEEEE)GG+GECG<G?GHAEG(FG;GG*FEC;GFE<FEGFEAG3DFACFEEEE;CE
G.EEEGE?CGCGC;EGCAGFGGGCEGG
```

- Archaea: [http://teachoceanscience.net/teaching\\_resources/education\\_modules/marine\\_bacteria/learn\\_about/](http://teachoceanscience.net/teaching_resources/education_modules/marine_bacteria/learn_about/)
- Virus: [jonlieffmd.com](http://jonlieffmd.com)
- Bacterium: <http://dtc.pima.edu>
- Arabidopsis: <http://de.wikipedia.org/wiki/Acker-Schmalwand>
- Frog: <http://de.wikipedia.org/wiki/Froschlurche>

# How big a problem is the data generation for the sequencing of entire genomes?



← Human Chr 8: 146,000,000 bp →

Method	Approach	Real-time	Read-length	Bp per run	# of runs for 10x coverage
<i>Sanger (ABI 3730xl)</i>	<i>Sequencing by synthesis</i>	<i>No</i>	<i>700 - 1000 bp</i>	<i>0.77 Mb</i>	<i>2,000</i>
<i>454/Roche<sup>2</sup></i>	<i>Sequencing by synthesis</i>	<i>Yes</i>	<i>700 - 1000 bp</i>	<i>700 Mb</i>	<i>2</i>
<i>Illumina MiSeq<sup>3</sup></i>	<i>Sequencing by synthesis</i>	<i>Yes</i>	<i>300 bp</i>	<i>15 Gb</i>	<i>~0.1</i>
<i>Illumina HiSeq<sup>3</sup></i>	<i>Sequencing by synthesis</i>	<i>Yes</i>	<i>150 bp</i>	<i>1000 Gb</i>	<i>~0.02</i>

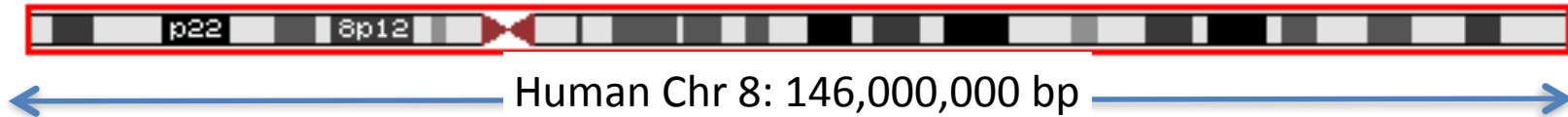
<sup>1</sup> <http://www6.appliedbiosystems.com/products/abi3730xlspeccs.cfm>

<sup>2</sup> <http://454.com/products/gs-flx-system/>

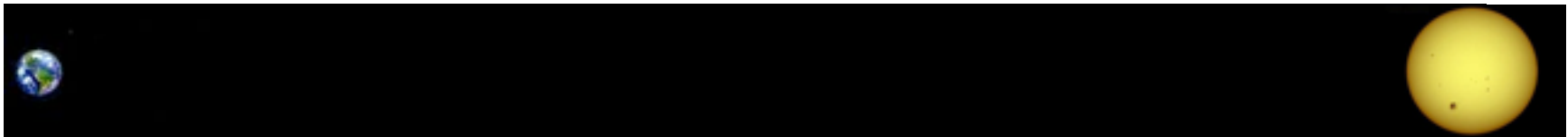
<sup>3</sup> <http://www.illumina.com/systems/sequencing.ilmn>



# How big a problem is the data generation for sequencing of entire genomes?



Method	Approach	Real-time	Read-length	Bp per run	# of runs for 10x coverage
<i>Illumina</i>	<i>Sequencing by synthesis</i>	<i>Yes</i>	<i>125 bp</i>	<i>1000 Mb</i>	<i>~0.02</i>

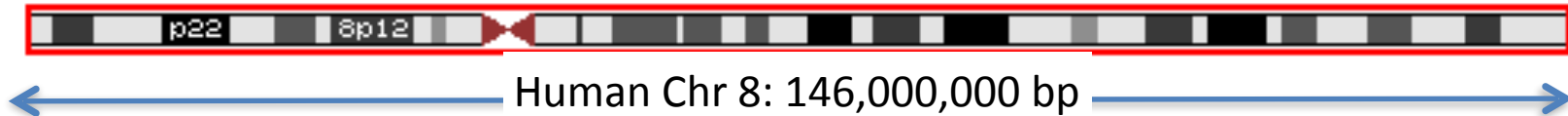


150,000,000 Km

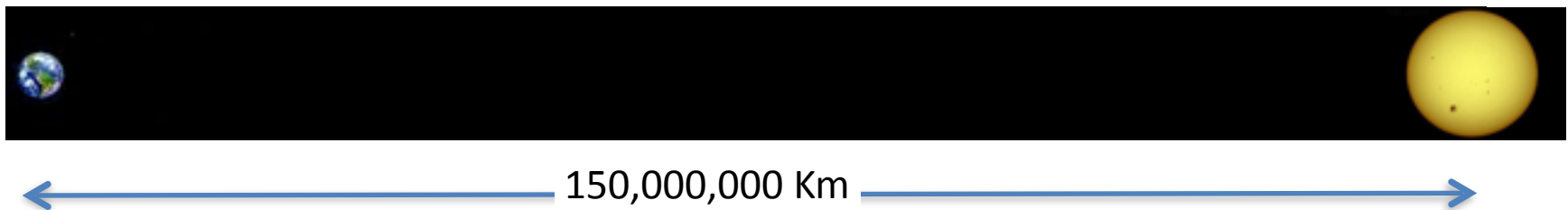


Oberursel -> Siegen: 123 km

# How big a problem is the data generation for sequencing of entire genomes?



Method	Approach	Real-time	Read-length	Bp per run	# of runs for 10x coverage
<i>Illumina</i>	<i>Sequencing by synthesis</i>	<i>Yes</i>	<i>125 bp</i>	<i>1000 Mb</i>	<i>~0.02</i>



In fact, the problem is at least 2 orders of magnitude larger since:

- \* The entire human genome consists of approx. 3.2 Billion base pairs
- \* 1-fold coverage is not sufficient. Typically at least 10 x coverage\* should be achieved. Thus, we need to sequence 32 Billion base pairs.

\*ca 80x required for short read sequencer

For the better part of my presentations we will look at DNA either as lines or as text strings...

## **Strategies to sequence long DNA molecules: Shotgun Sequencing**



# Strategies to sequence long DNA molecules: Shotgun Sequencing



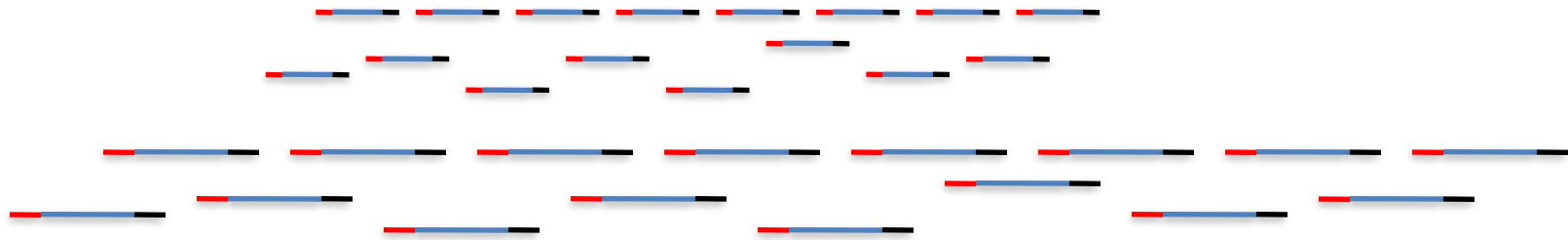
1. Randomly break template DNA into pieces

# Strategies to sequence long DNA molecules: Shotgun Sequencing



1. Randomly break template DNA into pieces

# Strategies to sequence long DNA molecules: Shotgun Sequencing



1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends and size select

## The Template:

5' -...CTGATCTATGCTCGCACT...- 3'

3' -...GACTAGATACGAGCGTGA...- 5'

# Sanger Sequencing in a Nutshell (Sequencing by synthesis)

## Step1: Template amplification

single template molecule



Polymerase  
Chain  
Reaction  
~35 cycles

Millions of identical template molecules

## Step2: Cycle sequencing

DNA-Polymerase

Primer for starting the synthesis

Desoxinucleotides:

dATP, dCTP, dTTP, dGTP

Di-Desoxinucleotides (Dye-Terminators)

ddATP  ddCTP 

ddTTP  ddGTP 

3' -...GACTAGATACGAGCGTGA...- 5' (template)  
5' -...CTGAT →→→ (primer)

Repeat cycle of primer  
annealing,  
polymerization and  
strand separation  $n$   
times

...CTGATCTAT 

...CTGATCTATGCTC 

## The Template:

5' -...CTGATCTATGCTCGCACT...- 3'

3' -...GACTAGATACGAGCGTGA...- 5'

# Sanger Sequencing in a Nutshell (Sequencing by synthesis)

## Step1: Template amplification

single template molecule



Polymerase  
Chain  
Reaction  
~35 cycles

Millions of identical template molecules

## Step2: Cycle sequencing

DNA-Polymerase

Primer for starting the synthesis

Desoxinucleotides:

dATP, dCTP, dTTP, dGTP

Di-Desoxinucleotides (Dye-Terminators)

ddATP (green), ddCTP (blue)

ddTTP (red), ddGTP (black)

3' -...GACTAGATACGAGCGTGA...- 5' (template)  
5' -...CTGAT →→→ (primer)

Repeat cycle of primer  
annealing,  
polymerization and  
strand separation  $n$   
times

...CTGATC (blue)  
...CTGATCT (red)  
...CTGATCTA (green)  
...CTGATCTAT (black)  
...CTGATCTATG (blue)  
...CTGATCTATGC (red)  
...CTGATCTATGCT (blue)  
...CTGATCTATGCTC (black)  
...CTGATCTATGCTCG (black)



## The Template:

5' -...CTGATCTATGCTCGCACT...-3'  
3' -...GACTAGATACGAGCGTGA...-5'

# Sanger Sequencing in a Nutshell (Sequencing by synthesis)

## Step1: Template amplification

single template molecule



Polymerase  
Chain  
Reaction  
~35 cycles

Millions of identical template molecules

## Step2: Cycle sequencing

DNA-Polymerase

Primer for starting the synthesis

Desoxinucleotides:




dATP, dCTP, dTTP, dGTP

Di-Desoxinucleotides (Dye-Terminators)

ddATP  ddCTP   
ddTTP  ddGTP 

3' -...GACTAGATACGAGCGTGA...-5' (template)  
5' -...CTGAT →→→→ (primer)

Repeat cycle of primer  
annealing,  
polymerization and  
strand separation  $n$   
times

...CTGATC   
...CTGATCT   
...CTGATCTA   
...CTGATCTAT   
...CTGATCTATG   
...CTGATCTATGC   
...CTGATCTATGCT   
...CTGATCTATGCTC   
...CTGATCTATGCTCG

Step 3: Size separation via electrophoresis  
and detection of fluorescence markers

## The Template:

5' -...CTGATCTATGCTCGCACT...-3'  
3' -...GACTAGATACGAGCGTGA...-5'

# Sanger Sequencing in a Nutshell (Sequencing by synthesis)

## Step1: Template amplification

single template molecule



Polymerase  
Chain  
Reaction  
~35 cycles

Millions of identical template molecules

## Step2: Cycle sequencing

DNA-Polymerase

Primer for starting the synthesis

Desoxinucleotides:

dATP, dCTP, dTTP, dGTP

Di-Desoxinucleotides (Dye-Terminators)

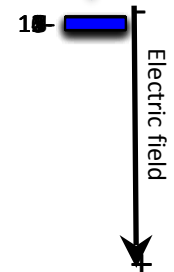
ddATP (green), ddCTP (blue),  
ddTTP (red), ddGTP (black)

3' -...GACTAGATACGAGCGTGA...-5' (template)  
5' -...CTGAT →→→→ (primer)

Repeat cycle of primer  
annealing,  
polymerization and  
strand separation  $n$   
times

...CTGATC (blue)  
...CTGATCT (red)  
...CTGATCTA (green)  
...CTGATCTAT (black)  
...CTGATCTATG (blue)  
...CTGATCTATGC (red)  
...CTGATCTATGCT (black)  
...CTGATCTATGCTC (blue)  
...CTGATCTATGCTCG (black)

Step 3: Size separation via electrophoresis  
and detection of fluorescence markers



## The Template:

5' -...CTGATCTATGCTCGCACT...- 3'  
3' -...GACTAGATACGAGCGTGA...- 5'

# Sanger Sequencing in a Nutshell (Sequencing by synthesis)

## Step1: Template amplification

single template molecule



Polymerase  
Chain  
Reaction  
~35 cycles

Millions of identical template molecules

## Step2: Cycle sequencing

DNA-Polymerase

Primer for starting the synthesis

Desoxinucleotides:

dATP, dCTP, dTTP, dGTP

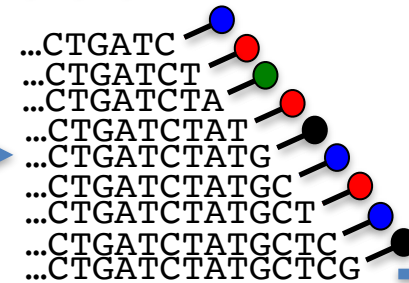
Di-Desoxinucleotides (Dye-Terminators)

ddATP (green dot), ddCTP (blue dot)

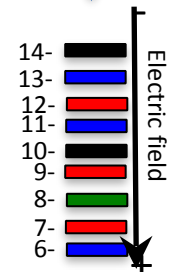
ddTTP (red dot), ddGTP (black dot)

3' -...GACTAGATACGAGCGTGA...- 5' (template)  
5' -...CTGAT →→→ (primer)

Repeat cycle of primer  
annealing,  
polymerization and  
strand separation *n*  
times



Step 3: Size separation via electrophoresis  
and detection of fluorescence markers



## The Template:

5' -...CTGATCTATGCTCGCACT...- 3'  
3' -...GACTAGATACGAGCGTGA...- 5'

# Sanger Sequencing in a Nutshell (Sequencing by synthesis)

## Step1: Template amplification

single template molecule



Polymerase  
Chain  
Reaction  
~35 cycles

Millions of identical template molecules

## Step2: Cycle sequencing

DNA-Polymerase

Primer for starting the synthesis

Desoxinucleotides:

dATP, dCTP, dTTP, dGTP

Di-Desoxinucleotides (Dye-Terminators)

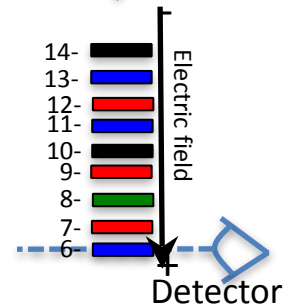
ddATP (green dot), ddCTP (blue dot)  
ddTTP (red dot), ddGTP (black dot)

3' -...GACTAGATACGAGCGTGA...- 5' (template)  
5' -...CTGAT →→→ (primer)

Repeat cycle of primer  
annealing,  
polymerization and  
strand separation *n*  
times



Step 3: Size separation via electrophoresis  
and detection of fluorescence markers



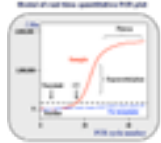
## The Template:

5' -...CTGATCTATGCTCGCACT...-3'  
3' -...GACTAGATACGAGCGTGA...-5'

# Sanger Sequencing in a Nutshell (Sequencing by synthesis)

## Step1: Template amplification

single template molecule



Polymerase  
Chain  
Reaction  
~35 cycles

Millions of identical template molecules

## Step2: Cycle sequencing

DNA-Polymerase

Primer for starting the synthesis

Desoxinucleotides:

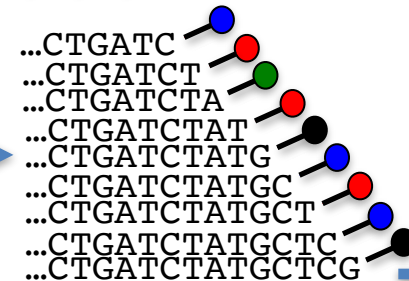
dATP, dCTP, dTTP, dGTP

Di-Desoxinucleotides (Dye-Terminators)

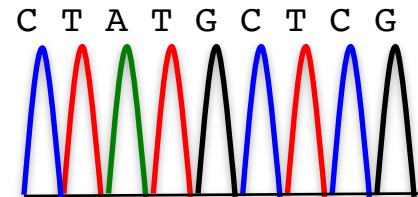
ddATP (green), ddCTP (blue),  
ddTTP (red), ddGTP (black)

3' -...GACTAGATACGAGCGTGA...-5' (template)  
5' -...CTGAT →→→→ (primer)

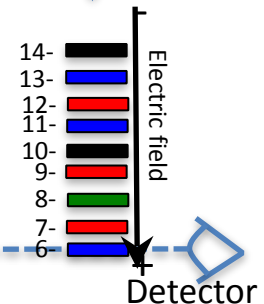
Repeat cycle of primer  
annealing,  
polymerization and  
strand separation *n*  
times



Step 3: Size separation via electrophoresis  
and detection of fluorescence markers



Base calling



Detector

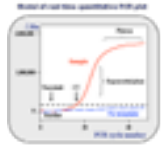
## The Template:

5' -...CTGATCTATGCTCGCACT...-3'  
3' -...GACTAGATACGAGCGTGA...-5'

# Sanger Sequencing in a Nutshell (Sequencing by synthesis)

## Step1: Template amplification

single template molecule



Polymerase  
Chain  
Reaction  
~35 cycles

Millions of identical template molecules

## Step2: Cycle sequencing

DNA-Polymerase

Primer for starting the synthesis

Desoxinucleotides:

dATP, dCTP, dTTP, dGTP

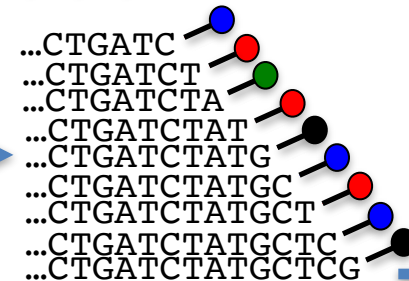
Di-Desoxinucleotides (Dye-Terminators)

ddATP (green), ddCTP (blue)

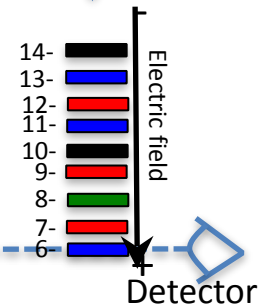
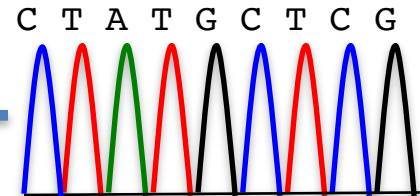
ddTTP (red), ddGTP (black)

3' -...GACTAGATACGAGCGTGA...-5' (template)  
5' -...CTGAT →→→→ (primer)

Repeat cycle of primer  
annealing,  
polymerization and  
strand separation  $n$   
times

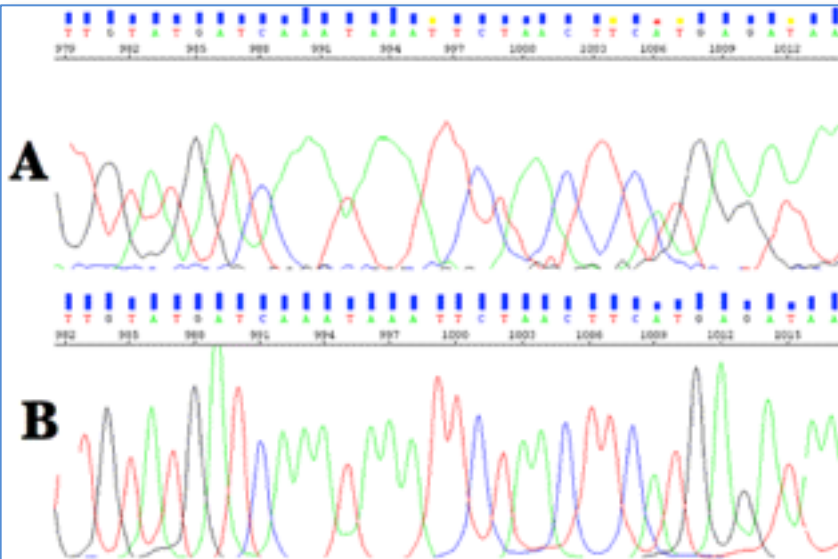


Step 3: Size separation via electrophoresis  
and detection of fluorescence markers



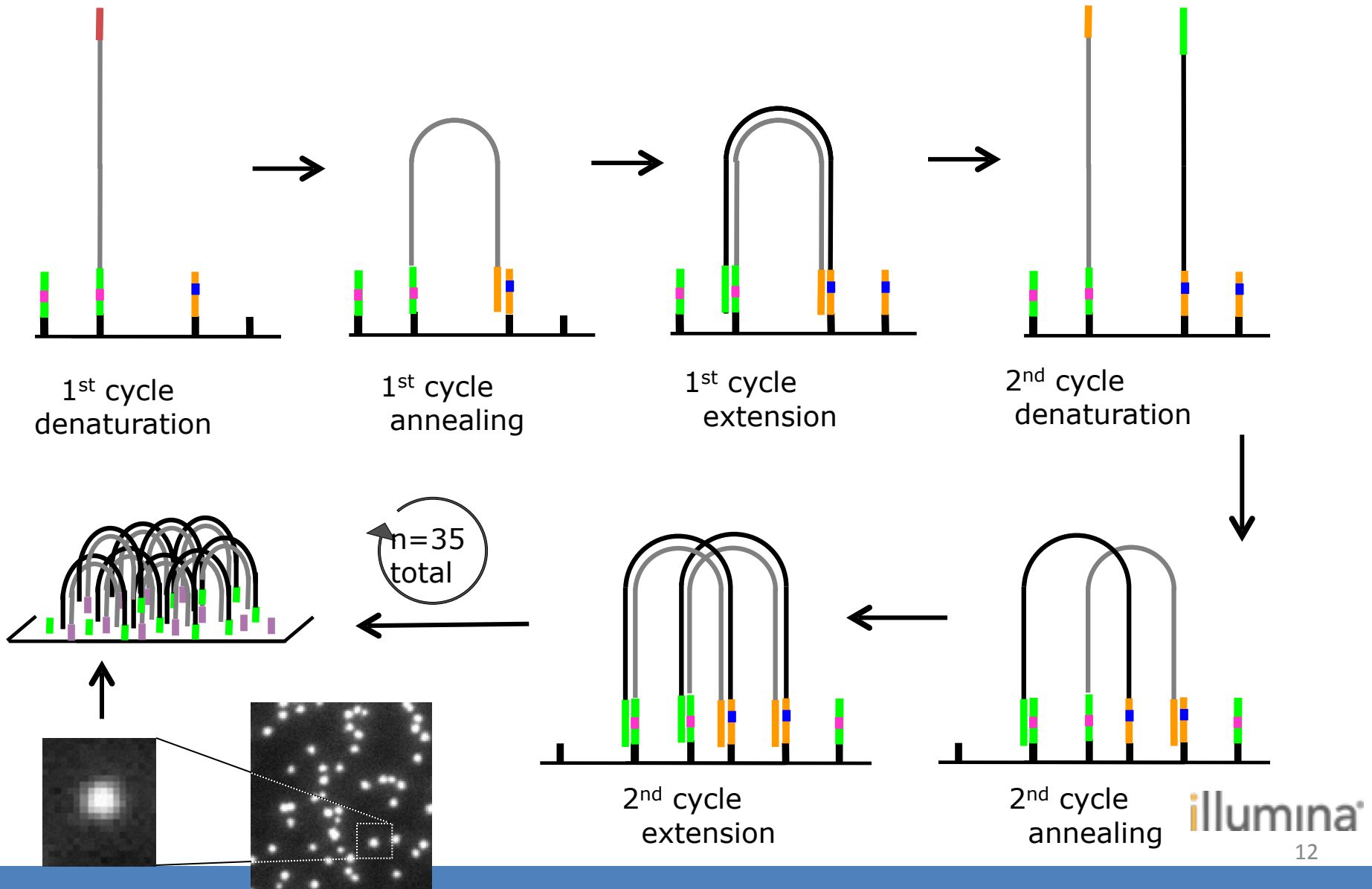
Base calling

Detector

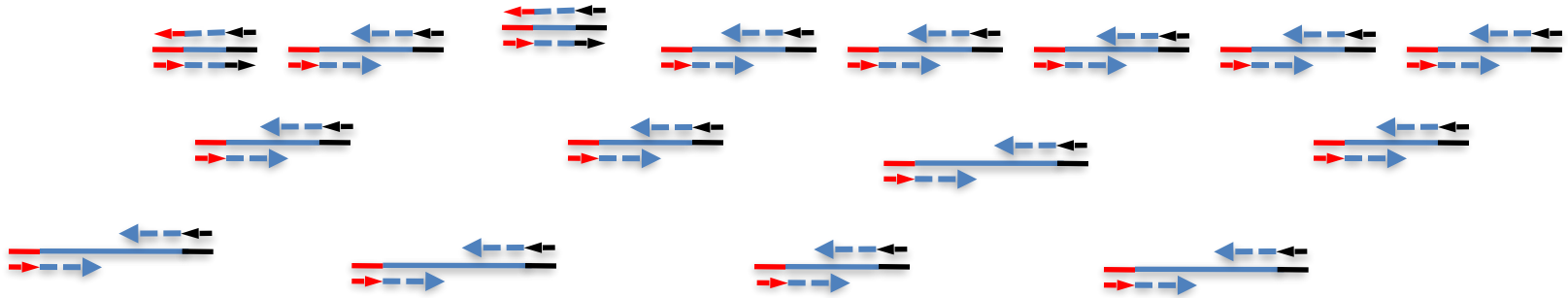


Example for a chromatogram

# Cluster Generation: Amplification



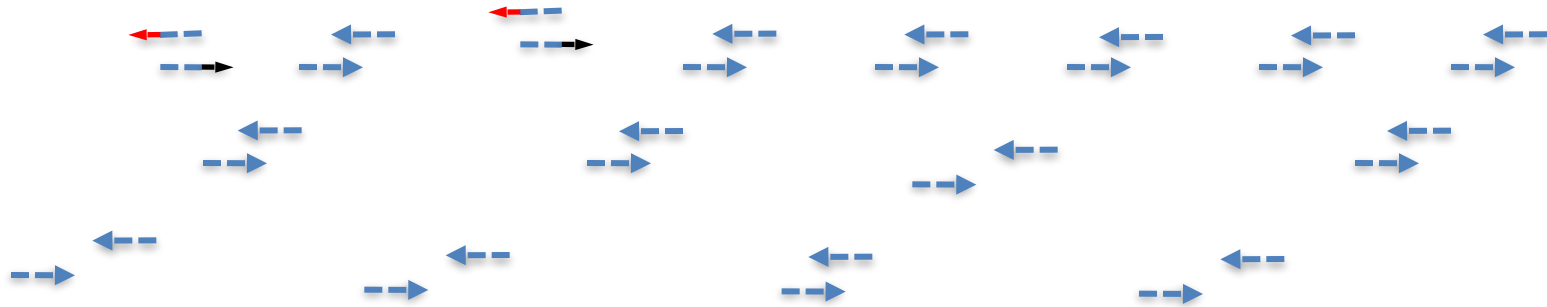
# Strategies to sequence long DNA molecules: Shotgun Sequencing



1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments



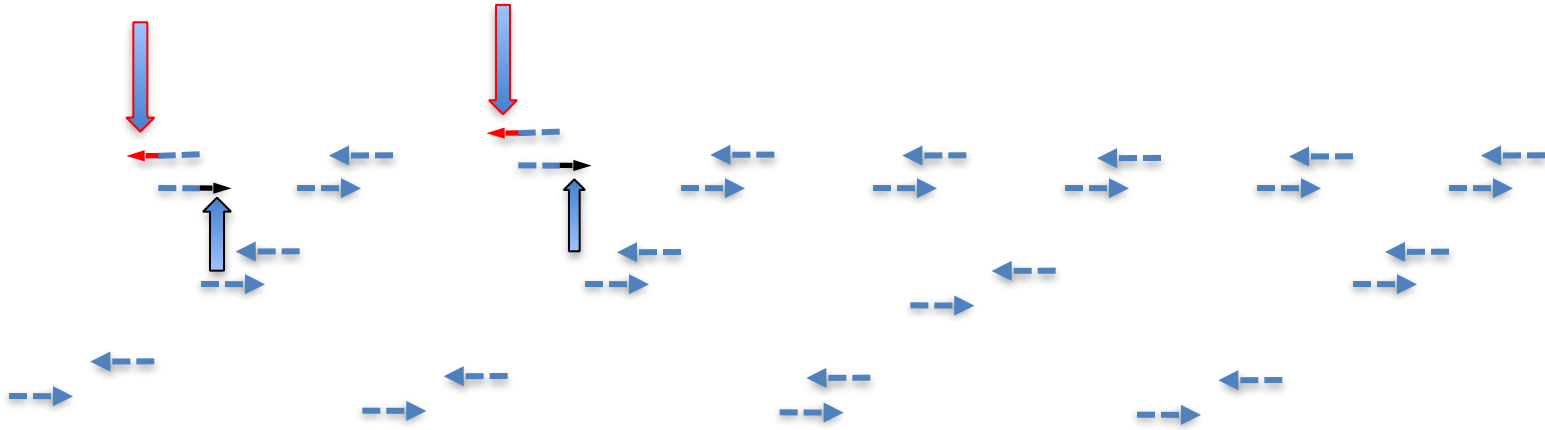
# Strategies to sequence long DNA molecules: Shotgun Sequencing



1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments

# Strategies to sequence long DNA molecules: Shotgun Sequencing

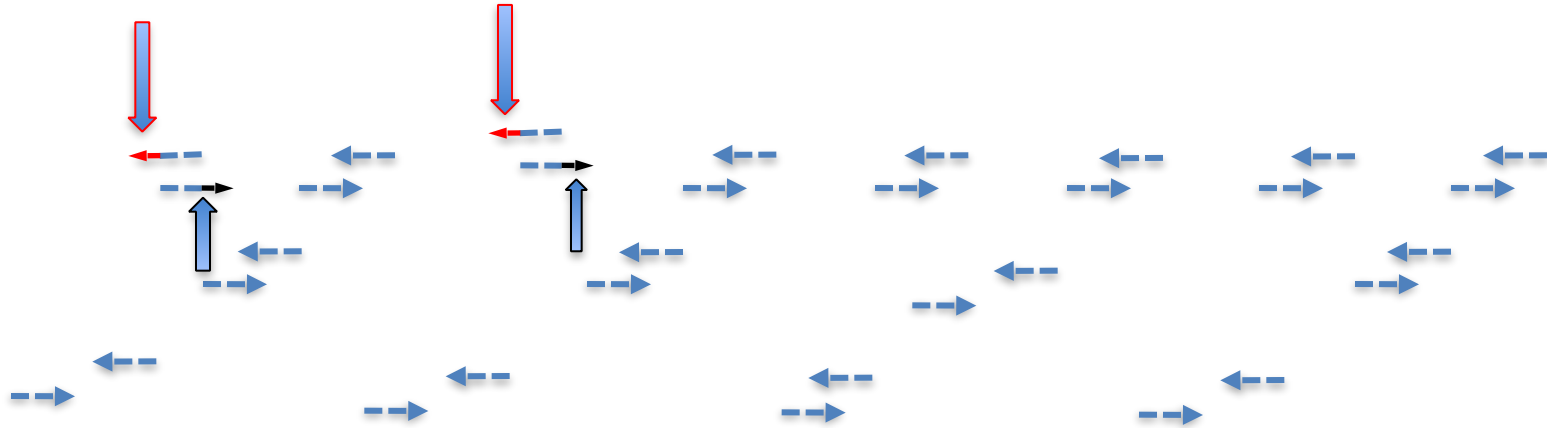
Sometimes adapter sequences remain!



1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments

# Strategies to sequence long DNA molecules: Shotgun Sequencing

Sometimes adapter sequences remain!



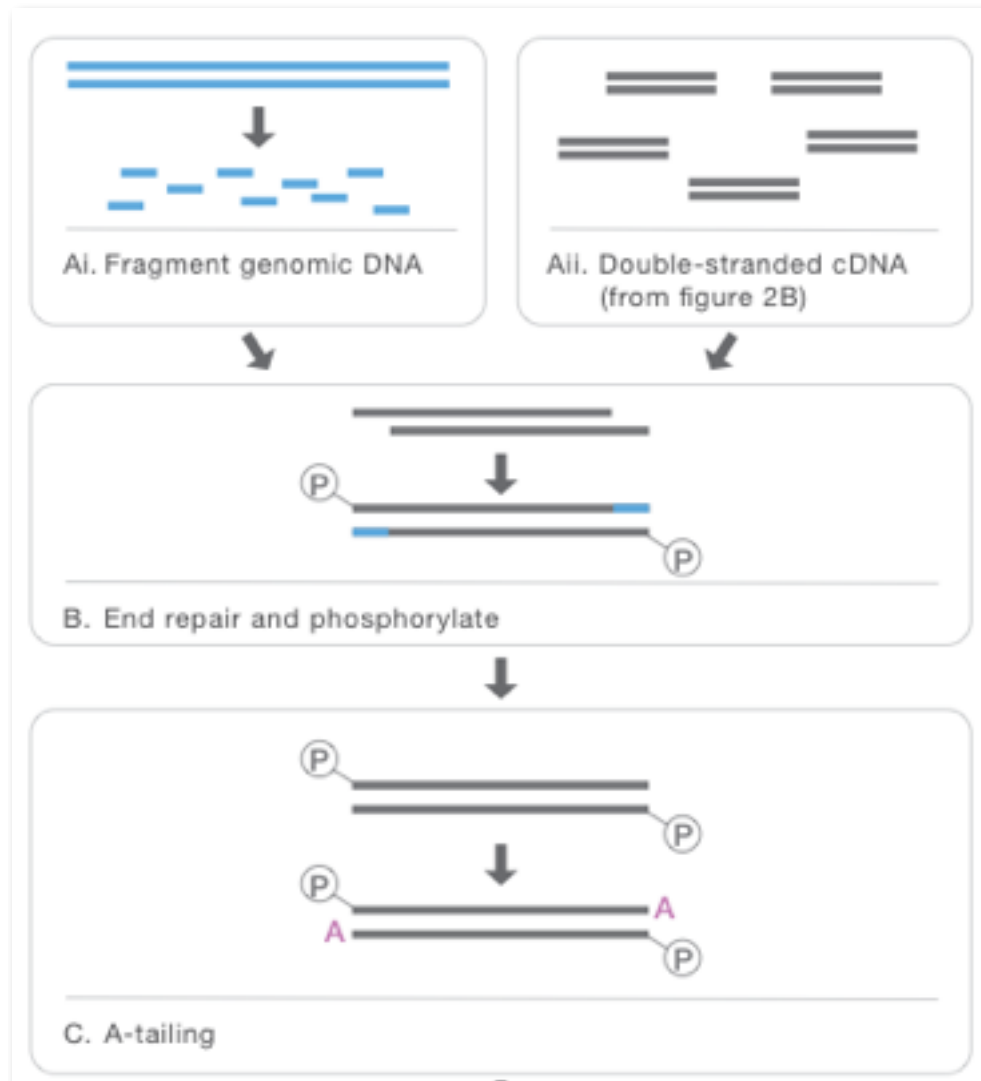
1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments

**Identifying these sequences is simple when we ignore the complexity of the search**

**The problem is, what sequence(s) are we looking for?**

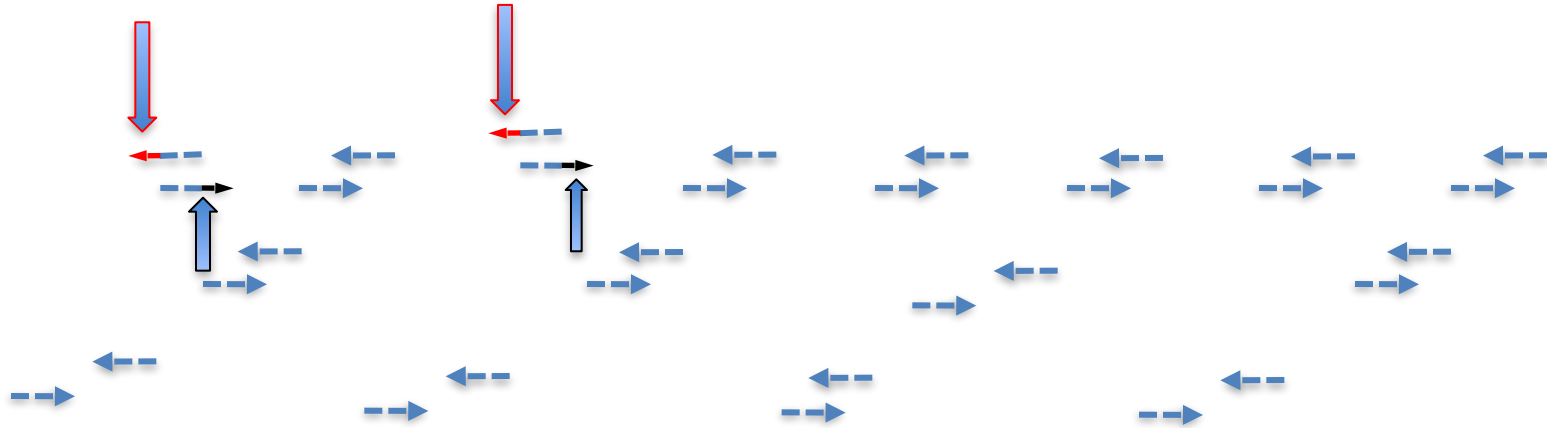
# Illumina sequence library generation

## Part 1. Template preparation



# Strategies to sequence long DNA molecules: Shotgun Sequencing

Sometimes adapter sequences remain!



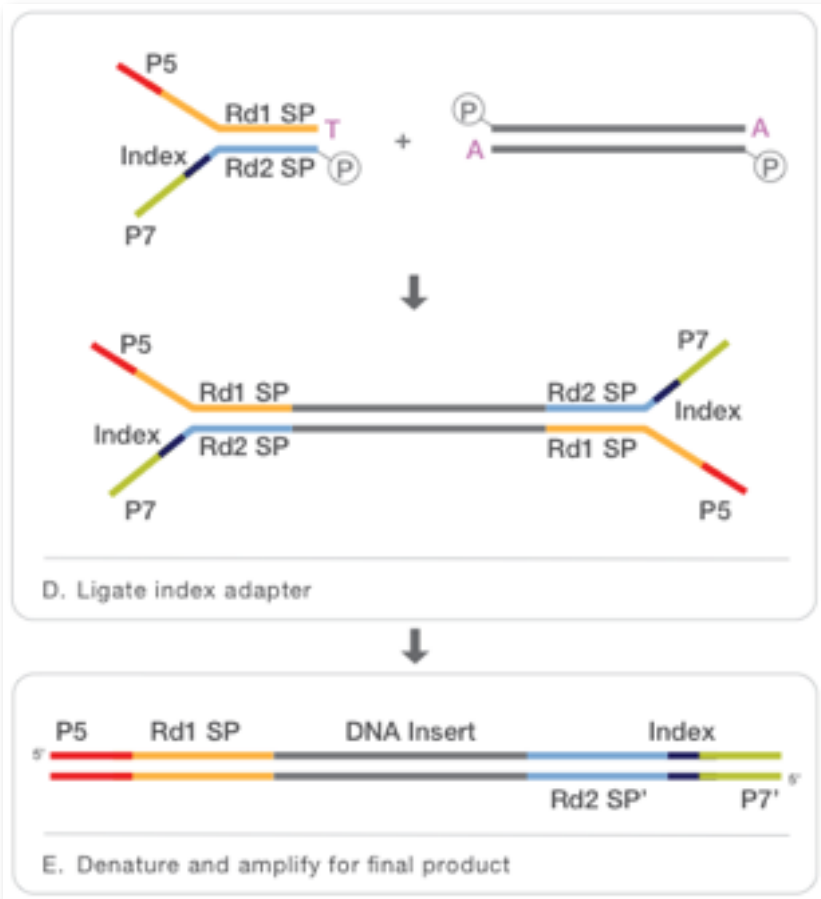
1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments

**Identifying these sequences is simple when we ignore the complexity of the search**

**The problem is, what sequence(s) are we looking for?**

# Illumina sequence library generation

## Part 2. Adapter ligation



### Multiplexing Adapters

5' P-GATCGGAAGAGCACACGTCT

5' ACACTCTTTCCCTACACGACGCTCTTCCGATCT

### Multiplexing PCR Primer 1.0

5' AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCT

### Multiplexing PCR Primer 2.0

5' GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT

### PCR Primer Index 1

5' CAAGCAGAAGACGGCATAACGAGATCGTGATGTGACTGGAGTTC

### PCR Primer Index 2

5' CAAGCAGAAGACGGCATAACGAGATACATCGGTGACTGGAGTTC

### Multiplexing Read 1 Sequencing Primer

5' ACACTCTTTCCCTACACGACGCTCTTCCGATCT

### Multiplexing Index Read Sequencing Primer

5' GATCGGAAGAGCACACGTCTGAACTCCAGTCAC

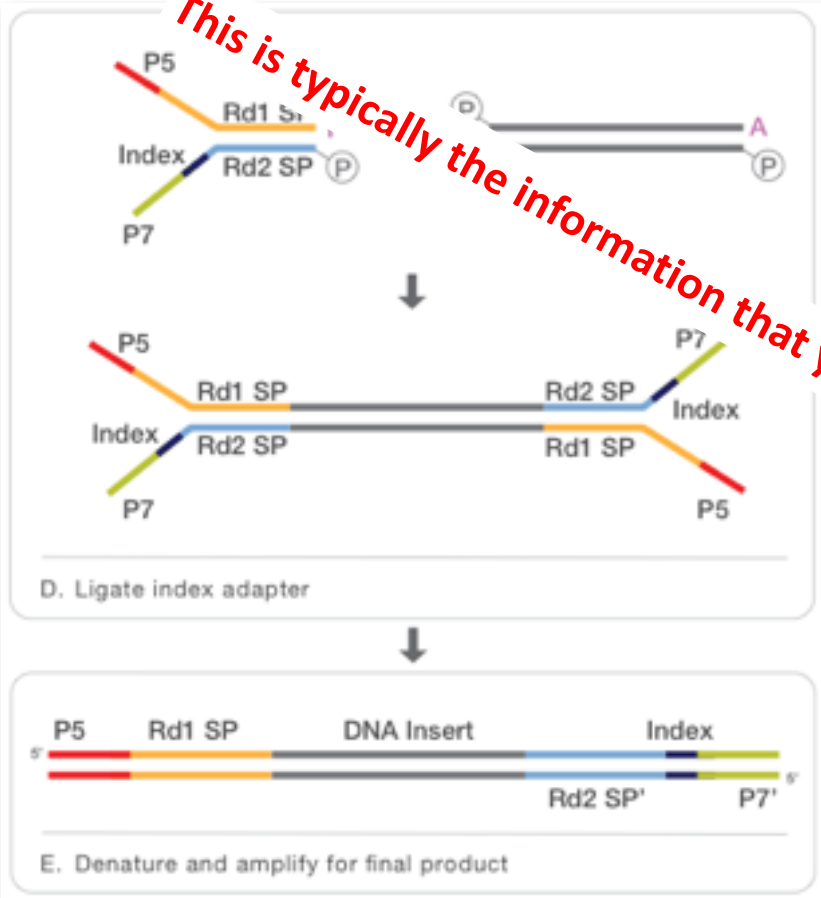
### Multiplexing Read 2 Sequencing Primer

5' GTGACGGAGTTCAGACGTGTGCTCTTCCGATCT

# Illumina sequence library generation

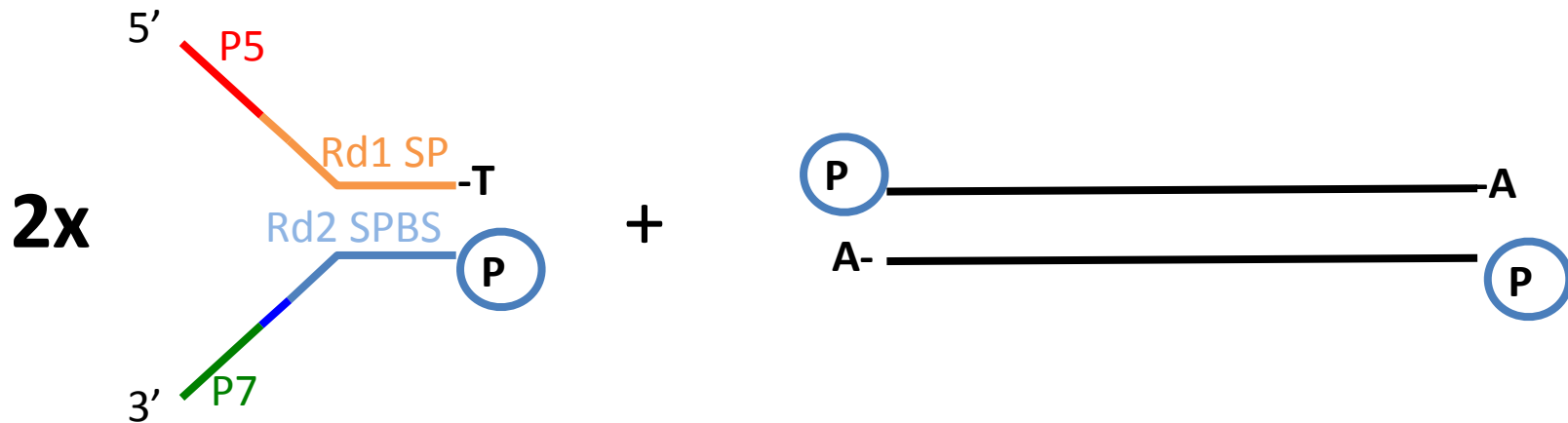
## Part 2. Adapter ligation

*This is typically the information that you get to plan your post-processing of sequence reads...*



- Read 1 Sequencing Adapters**
- 5' P-...GACACACGTCT
- 5' ACACTC...GACGCTCTCCGATCT
- Multiplexing PCR Primer 1.0**
- 5' AATGATACGGCGACCA...CACTCTTCCCTACACGACGCTCTCCGATCT
- Multiplexing PCR Primer 2.0**
- 5' GTGACTGGAGTTCAGACGTGTGCTC...
- PCR Primer Index 1**
- 5' CAAGCAGAAGACGGCATAACGAGATCGTGATGTGAC
- PCR Primer Index 2**
- 5' CAAGCAGAAGACGGCATAACGAGATACATCGGTGACTGGAGTTC
- Multiplexing Read 1 Sequencing Primer**
- 5' ACACTCTTCCCTACACGACGCTCTCCGATCT
- Multiplexing Index Read Sequencing Primer**
- 5' GATCGGAAGAGCACACGTCTGAACTCCAGTCAC
- Multiplexing Read 2 Sequencing Primer**
- 5' GTGACGGAGTTCAGACGTGTGCTCTCCGATCT

# Illumina sequence library generation: Taking a closer look



Multiplex Adapter P5

5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

Multiplexing PCR Primer 1.0

5' **AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCT**

Multiplex Adapter P7

5' **P-GATCGGAAGAGCACACGTCT**

Multiplex PCR Primer 2.0 (Reverse complement\*)

**AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC**

PCR Primer Index 1 (RC\*)

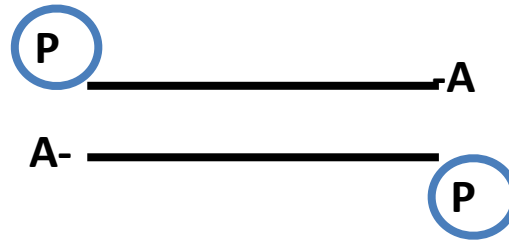
**GAACTCCAGTCACATCACGATCTCGTATGCCGTCTTCTGCTTG**

\*Relative to Illumina documentation



# Illumina sequence library generation: Taking a closer look

## Step 1. Ligation of the Multiplexing Adapters



5' **AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCT** 5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

**P-GATCGGAAGAGCACACGTCT**  
5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**  
**GAACTCCAGTCACATCAGATCTCGTATGCCGTCTTCTGCTTG**

# Illumina sequence library generation: Taking a closer look

## Step 1. Ligation of the Multiplexing Adapters



5' **AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCT**

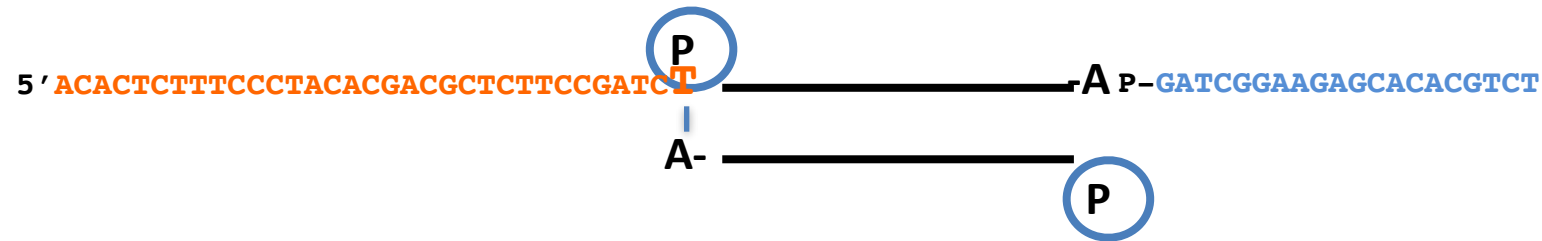
**P-GATCGGAAGAGCACACGTCT**

5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**

**GAACTCCAGTCACATCAGATCTCGTATGCCGTCTTCTGCTTG**

# Illumina sequence library generation: Taking a closer look

## Step 1. Ligation of the Multiplexing Adapters

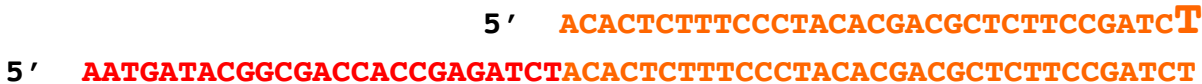


5' **AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCT**

5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**  
**GAACTCCAGTCACATCACGATCTCGTATGCCGTCTTCTGCTTG**

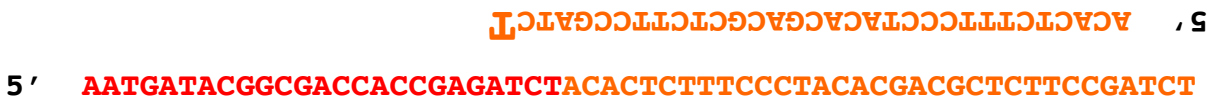
# Illumina sequence library generation: Taking a closer look

## Step 1. Ligation of the Multiplexing Adapters



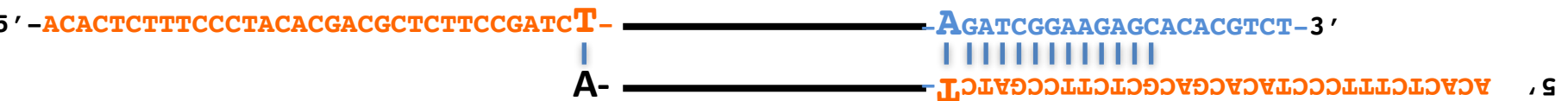
# Illumina sequence library generation: Taking a closer look

## Step 1. Ligation of the Multiplexing Adapters



# Illumina sequence library generation: Taking a closer look

## Step 1. Ligation of the Multiplexing Adapters



5' **AATGATACGGCGACCACCGAGATCTACACTCTTTCCTACACGACGCTCTTCCGATCT**

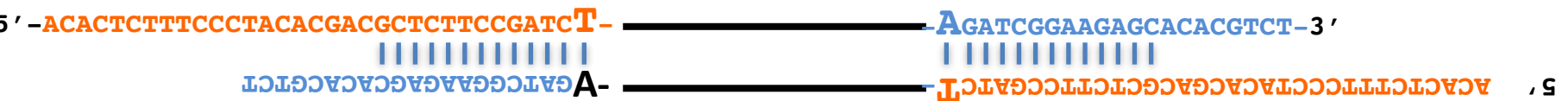
**GATCGGAAGAGCACGCTCTTCCGATCT**

5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**

**GAACTCCAGTCACATCACGATCTCGTATGCCGTCTTCTGCTTG**

# Illumina sequence library generation: Taking a closer look

## Step 1. Ligation of the Multiplexing Adapters



5' **AATGATACGGCGACCACCGAGATCTACACTCTTTCCTACACGACGCTCTTCCGATCT**

5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**  
**GAACTCCAGTCACATCACGATCTCGTATGCCGTCTTCTGCTTG**

# Illumina sequence library generation: Taking a closer look

## Step 2. PCR starting from Multiplex PCR Primer 2.0

5' -**ACACTCTTTCCTACACGACGCTCTTCCGATCT**--**AGATCGGAAGAGCACACGTCT**-3'

5' **AATGATACGGCGACCACCGAGATCTACACTCTTTCCTACACGACGCTCTTCCGATCT**

**GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**

**GAATCCAGTCACATCACGATCTCGTATGCCGTCTTCTGCTTG**



# Illumina sequence library generation: Taking a closer look

## Step 2. PCR starting from Multiplex PCR Primer 2.0

5' - **ACACTCTTCCCTACACGACGCTCTTCCGATCT** -  - **AGATCGGAAGAGCACACGTCT** - 3'

5' **AATGATACGGCGACCACCGAGATCTACACTCTTCCCTACACGACGCTCTTCCGATCT**

**GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**

**GAATCCAGTCACATCACGATCTCGTATGCCGTCTTCTGCTTG**

# Illumina sequence library generation: Taking a closer look

## Step 2. PCR starting from Multiplex PCR Primer 2.0

5' - **ACACTCTTCCCTACACGACGCTCTTCCGATCT** -  - **AGATCGGAAGAGCACACGTCT** - 3' /  
3' - **AGATCGGAAGAGCGGTCTGTAGTAGGAAGAGTGT** -  **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**

5' **AATGATACGGCGACCACCGAGATCTACACTCTTCCCTACACGACGCTCTTCCGATCT**

**GAACTCCAGTCACATCACGATCTCGTATGCCGTCTTCTGCTTG**

# Illumina sequence library generation: Taking a closer look

## Step 2b: PCR starting from Multiplexing PCR Primer 1.0

5' - **ACACTCTTTCCCTACACGACGCTCTTCCGATCT** - **AGATCGGAAGAGCACACGTCT** - 3'  
3' - **AGATCGGAAGAGGAGCGTCTGTTAGGGAAGAAGTGT** - **GTGACTGGAGTTCAGACGTTGTTCTTCCGATCT** - 5'

**AATGATACGGCGACCACCGAGATCT** **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

**GAACTCCAGTCACATCACGATCTCGTATGCCGTCTTCTGCTTG**

# Illumina sequence library generation: Taking a closer look

## Step 2b: PCR starting from Multiplexing PCR Primer 1.0

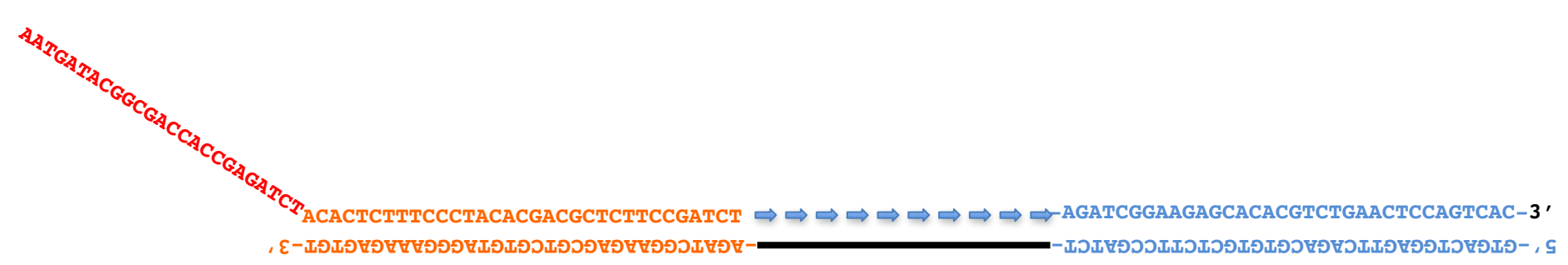
5' -GTGACTGGAGTTCAAGACGTTGTTGCTCTTCCGATCT-3' (blue)  
-AGATCGGAAGAAGAGCCTCGTGTAGGGAAAGAAGTGT-3' (orange)

AATGATACGGCGACCACCGAGATCT (red)  
ACACTCTTTCCCTACACGACGCTCTTCCGATCT (orange)

GAACTCCAGTCACATCACGATCTCGTATGCCGTCTTCTGCTTG (green)

# Illumina sequence library generation: Taking a closer look

## Step 2b: PCR starting from Multiplexing PCR Primer 1.0



GAACTCCAGTCACATCACGATCTCGTATGCCGTCTTCTGCTTG

# Illumina sequence library generation: Taking a closer look

## Step 3. PCR starting from PCR Primer Index 1

AATGATACGGCGACCCGGATCT

ACACTCTTTCCTACACGACGCTCTTCCGATCT ————— AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC-3'  
AGATCGGAAGAGCGCTTCGTAGAGAAAGAAGTGT-3' ————— GTGACTGGAGTTCAGACAGTGTGCTCTCCGATCT-5'

5' CAAGCAGAAGACGGCATACGAGATCGTGATGTGACTGGAGTTC

# Illumina sequence library generation: Taking a closer look

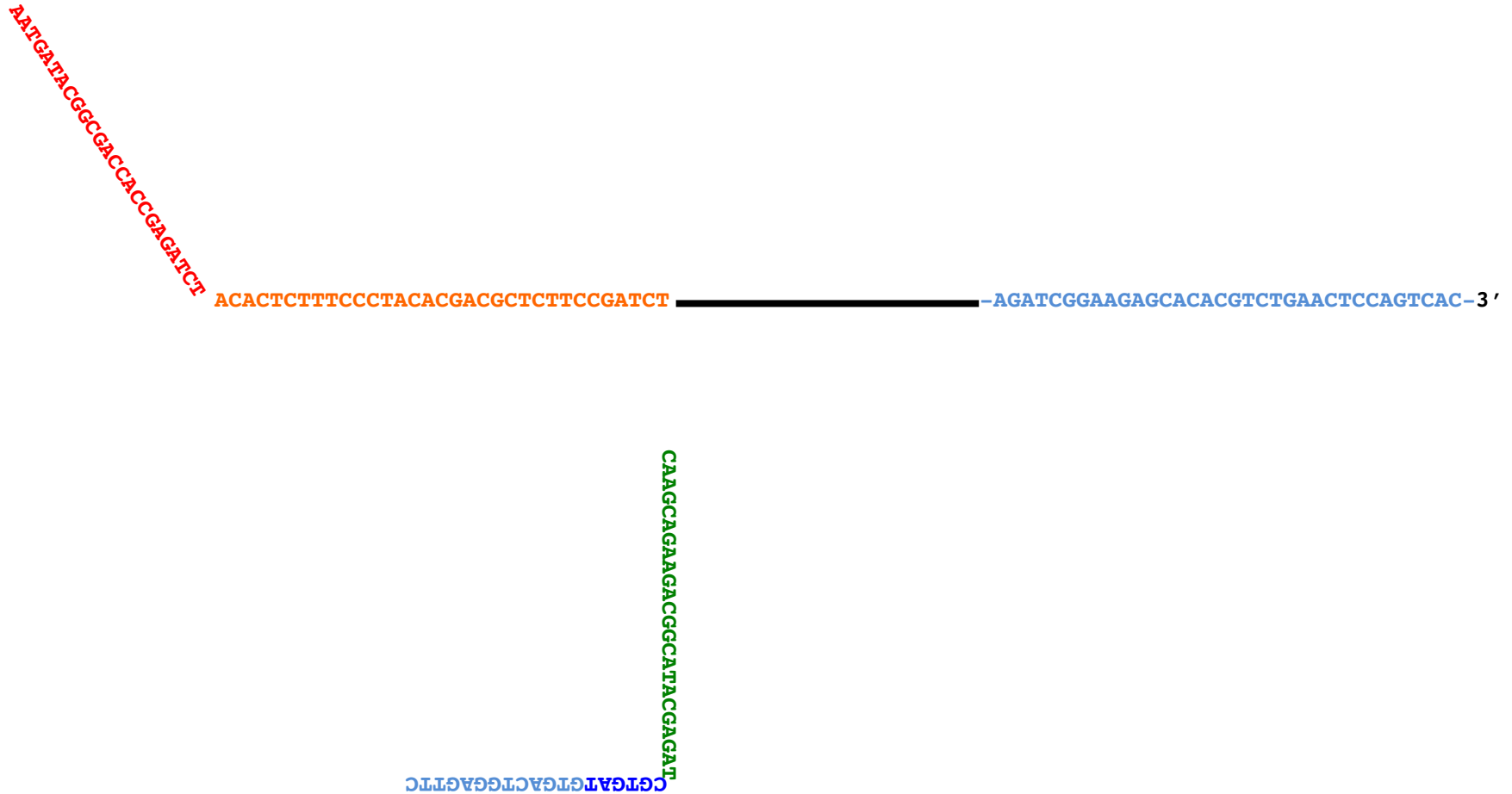
## Step 3. PCR starting from PCR Primer Index 1

ATGATACGGCGACCACCGAGTCT  
ACACTCTTTCCCTACACGACGCTCTTCCGATCT — AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC—3'

5' CAAGCAGAAGACGGCATACGAGATCGTGATGTGACTGGAGTTC

# Illumina sequence library generation: Taking a closer look

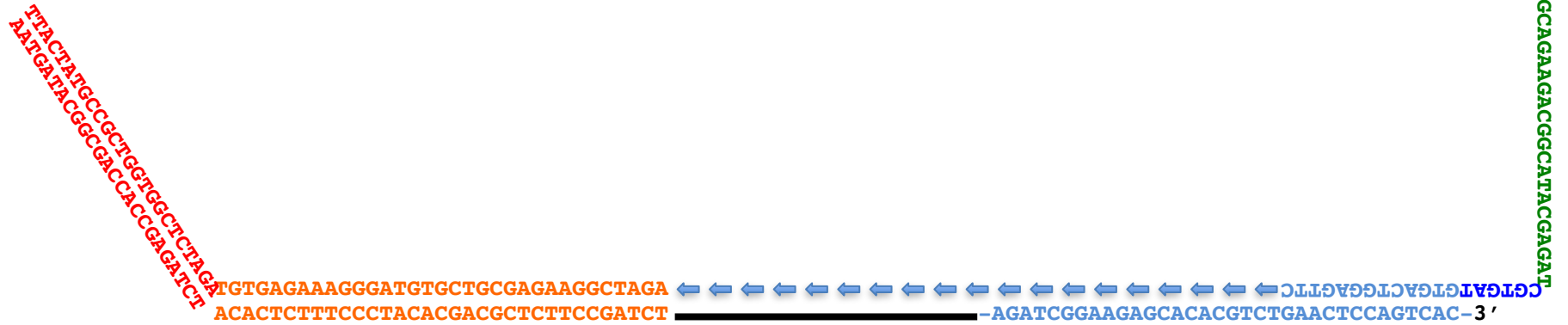
## Step 3. PCR starting from PCR Primer Index 1





# Illumina sequence library generation: Taking a closer look

## Step 3. PCR starting from PCR Primer Index 1



# Illumina sequence library generation: Taking a closer look

## Step 4. Completion of the construct

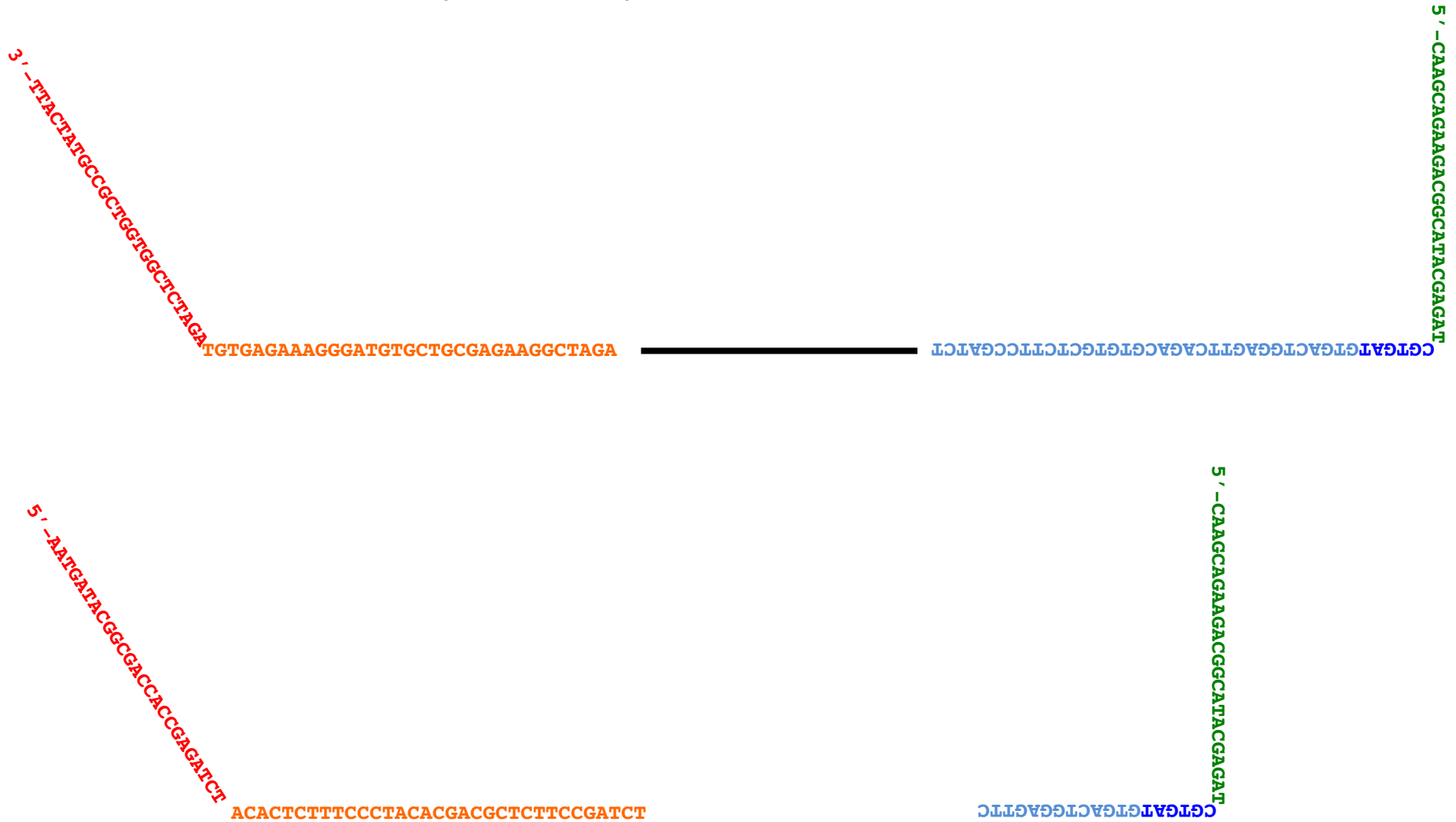


Multiplexing PCR Primer 1.0

PCR Primer Index 1

# Illumina sequence library generation: Taking a closer look

## Step 4. Completion of the construct

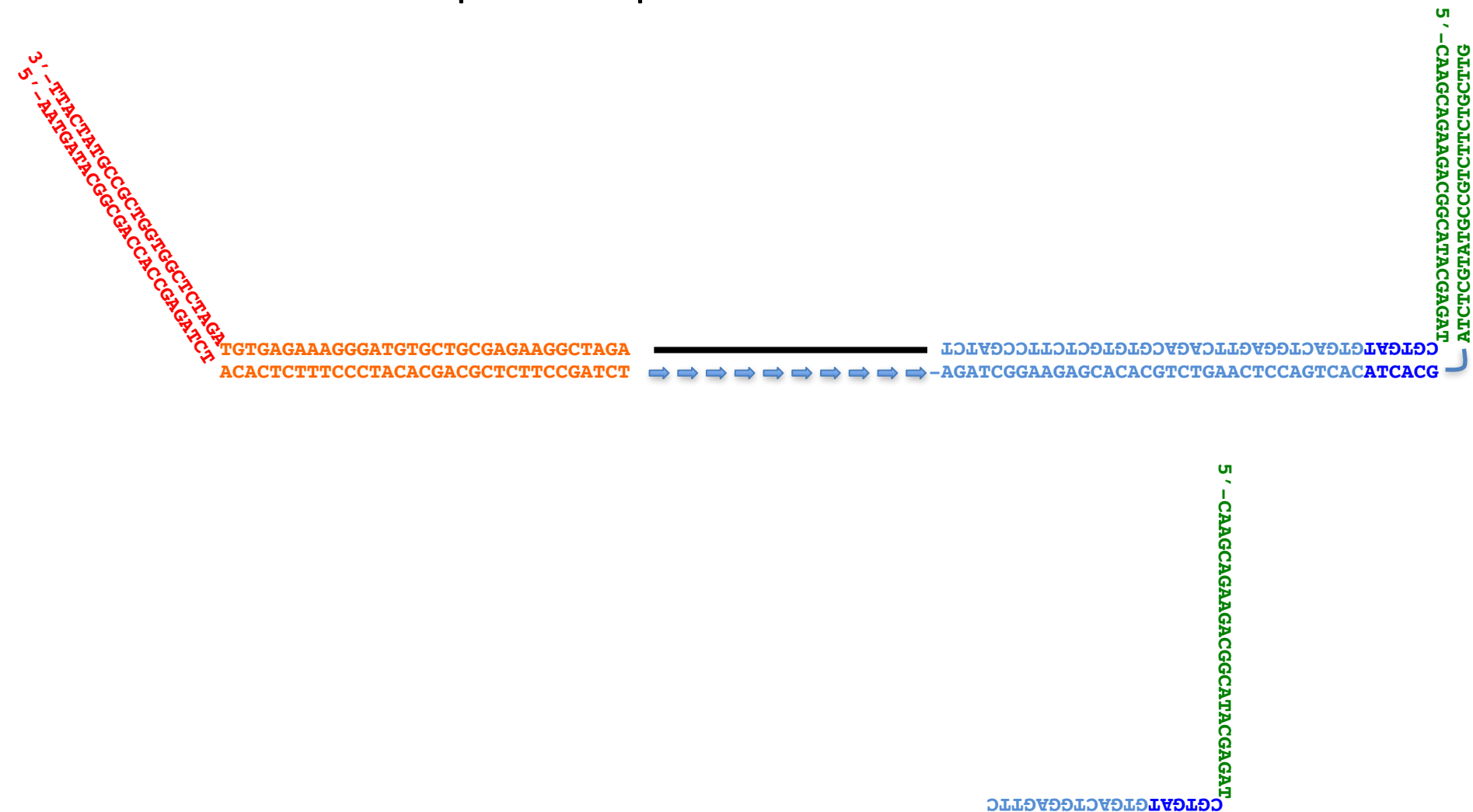


Multiplexing PCR Primer 1.0

PCR Primer Index 1

# Illumina sequence library generation: Taking a closer look

## Step 4. Completion of the construct



Multiplexing PCR Primer 1.0

PCR Primer Index 1

# Illumina sequence library generation: Taking a closer look

## Step 5. Amplify the construct with the two PCR primers



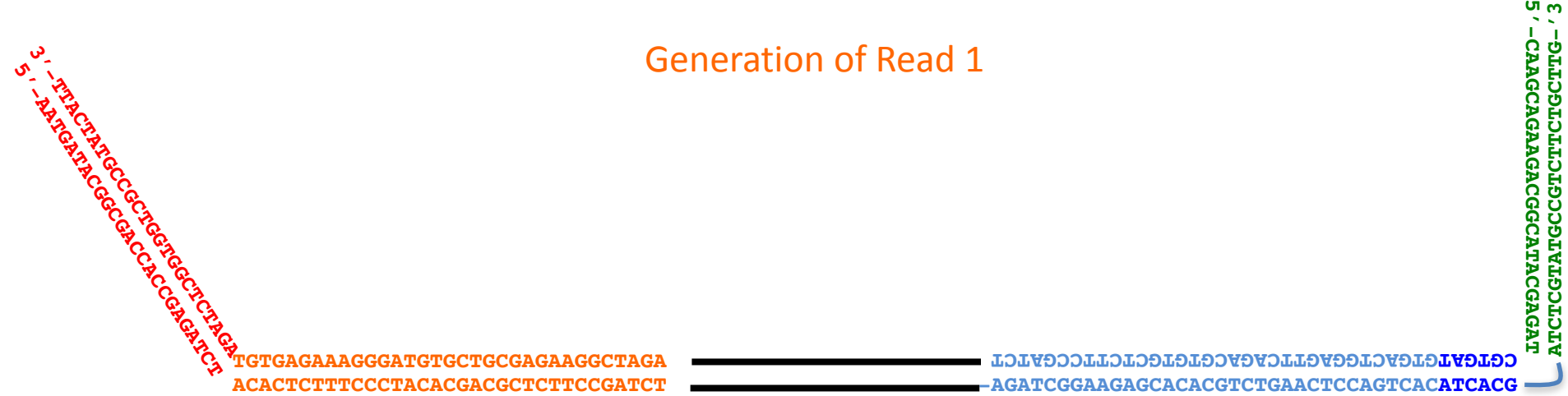
Multiplexing PCR Primer 1.0

PCR Primer Index 1

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index

## Generation of Read 1



## Multiplexing Read 1 Sequencing Primer

5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

## Multiplexing Index Read Sequencing Primer

5' **GATCGGAAGAGCACACGTCTGAACTCCAGTCAC**

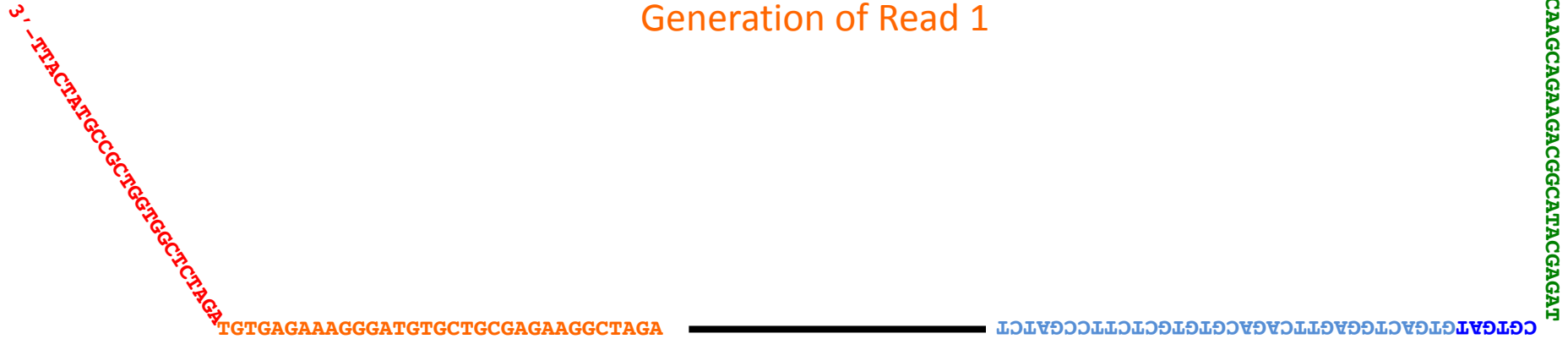
## Multiplexing Read 2 Sequencing Primer

5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index

## Generation of Read 1



## Multiplexing Read 1 Sequencing Primer

5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

## Multiplexing Index Read Sequencing Primer

5' **GATCGGAAGAGCACACGTCTGAACTCCAGTCAC**

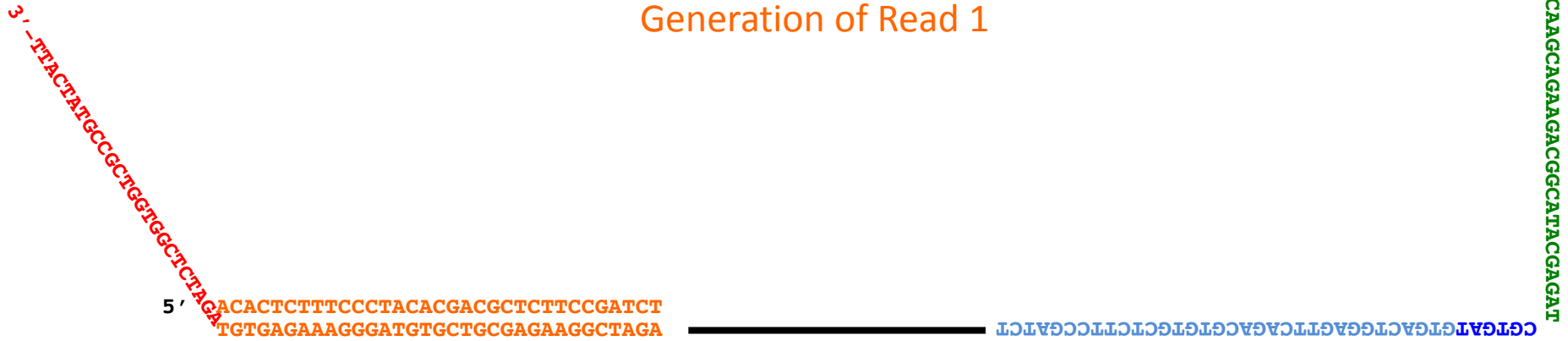
## Multiplexing Read 2 Sequencing Primer

5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index

## Generation of Read 1



Multiplexing Read 1 Sequencing Primer

Multiplexing Index Read Sequencing Primer

5' GATCGGAAGAGCACACGTCTGAACTCCAGTCAC

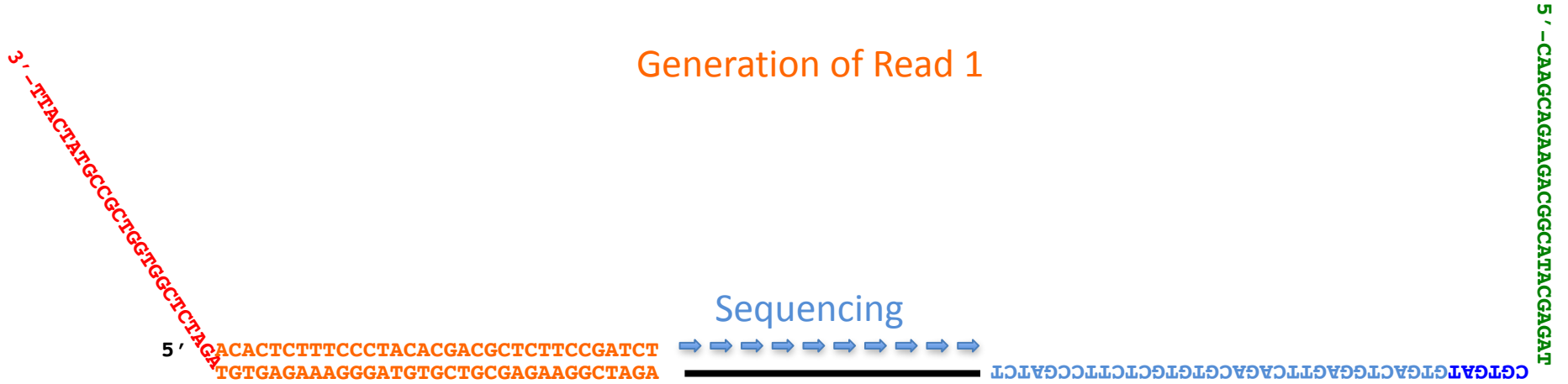
Multiplexing Read 2 Sequencing Primer

5' GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT



# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index



Multiplexing Read 1 Sequencing Primer

Multiplexing Index Read Sequencing Primer

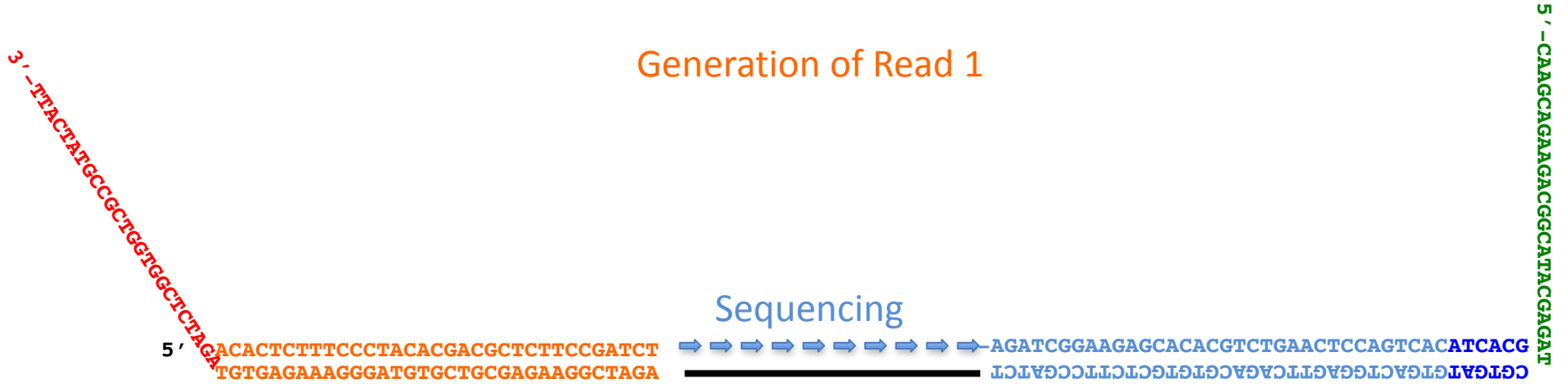
5' GATCGGAAGAGCACACGTCTGAACTCCAGTCAC

Multiplexing Read 2 Sequencing Primer

5' GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index



Multiplexing Read 1 Sequencing Primer

Multiplexing Index Read Sequencing Primer

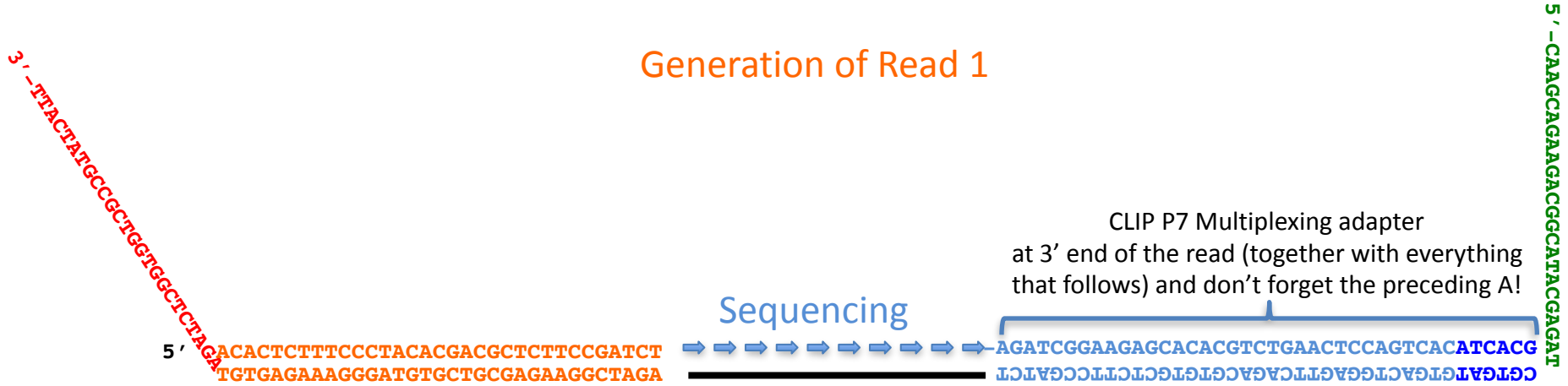
5' GATCGGAAGAGCACACGTCTGAACTCCAGTCAC

Multiplexing Read 2 Sequencing Primer

5' GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index



Multiplexing Read 1 Sequencing Primer

Multiplexing Index Read Sequencing Primer

5' GATCGGAAGAGCACACGTCTGAACTCCAGTCAC

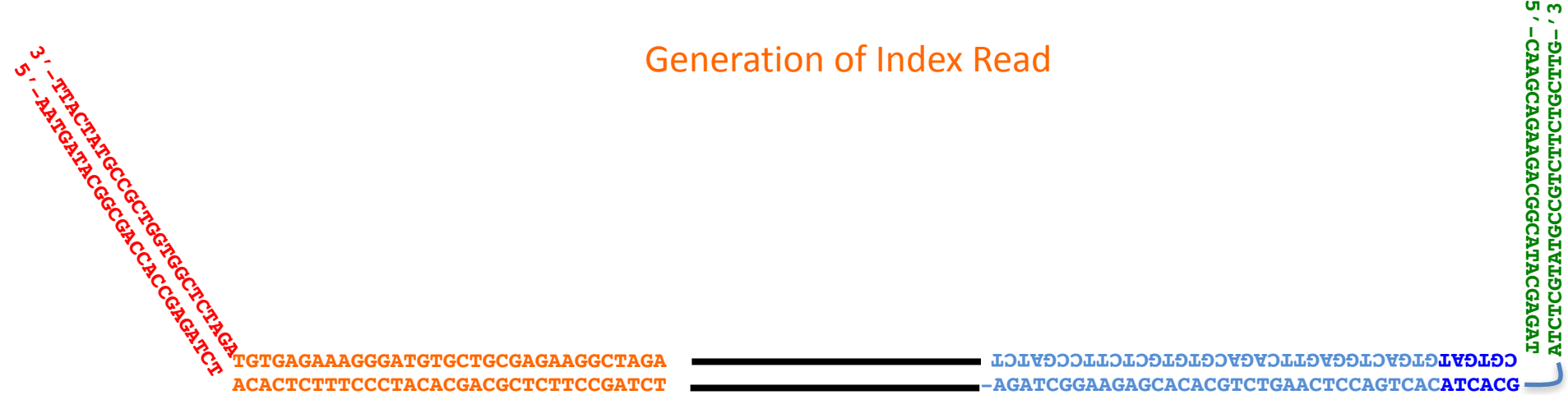
Multiplexing Read 2 Sequencing Primer

5' GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index

## Generation of Index Read



## Multiplexing Read 1 Sequencing Primer

5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

## Multiplexing Index Read Sequencing Primer

5' **GATCGGAAGAGCACACGTCTGAACTCCAGTCAC**

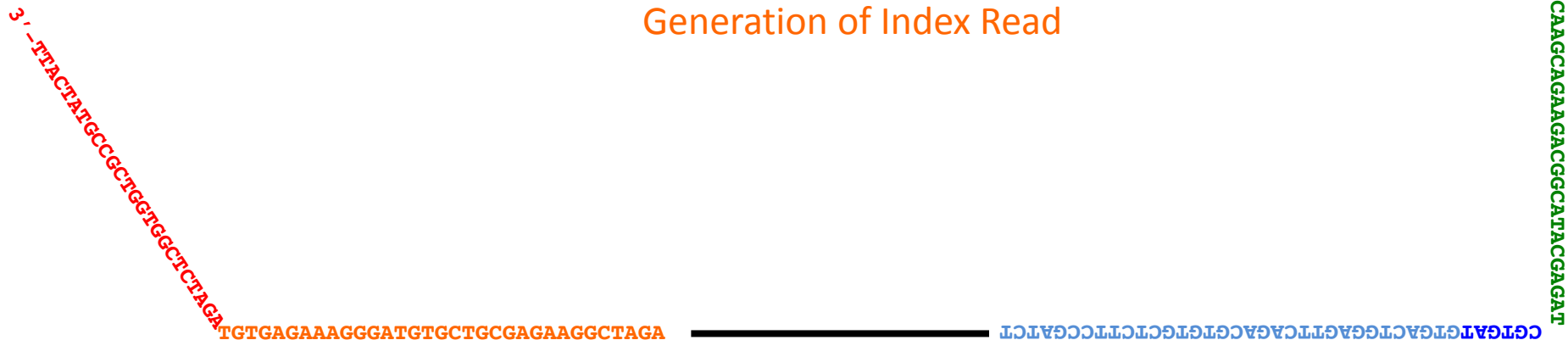
## Multiplexing Read 2 Sequencing Primer

5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index

## Generation of Index Read



## Multiplexing Read 1 Sequencing Primer

5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

## Multiplexing Index Read Sequencing Primer

5' **GATCGGAAGAGCACACGTCTGAACTCCAGTCAC**

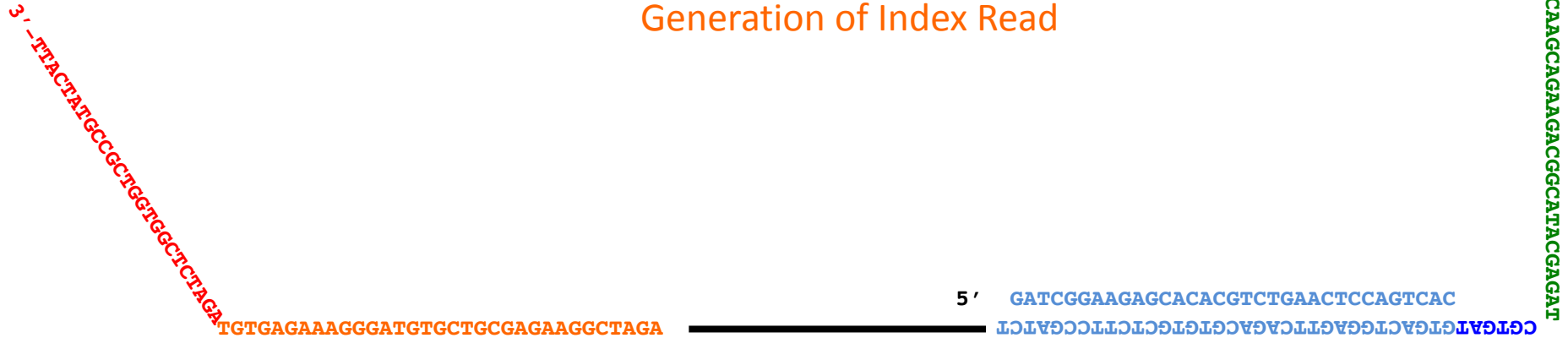
## Multiplexing Read 2 Sequencing Primer

5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index

## Generation of Index Read



## Multiplexing Read 1 Sequencing Primer

5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

## Multiplexing Index Read Sequencing Primer

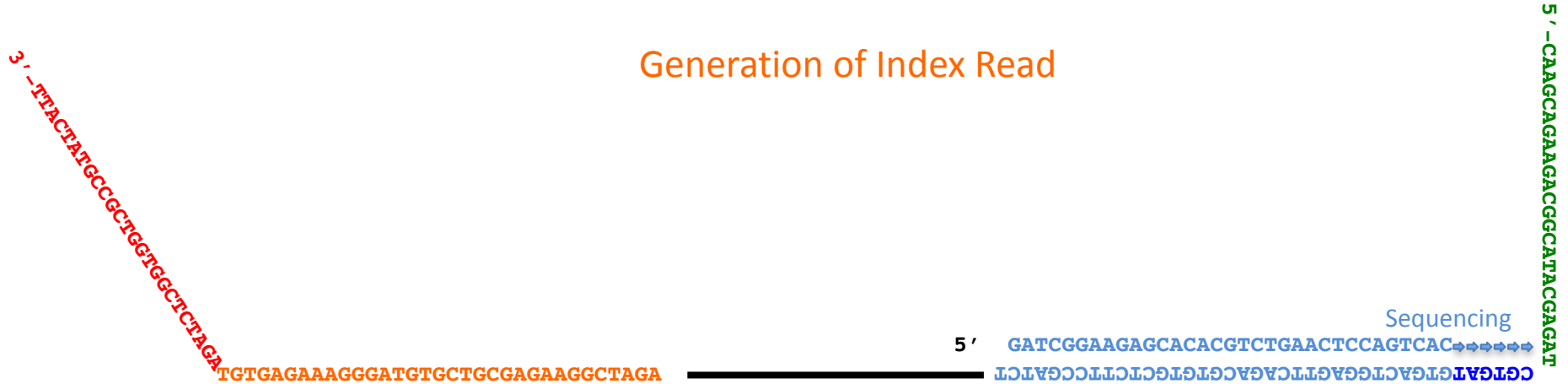
## Multiplexing Read 2 Sequencing Primer

5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index

## Generation of Index Read



## Multiplexing Read 1 Sequencing Primer

5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

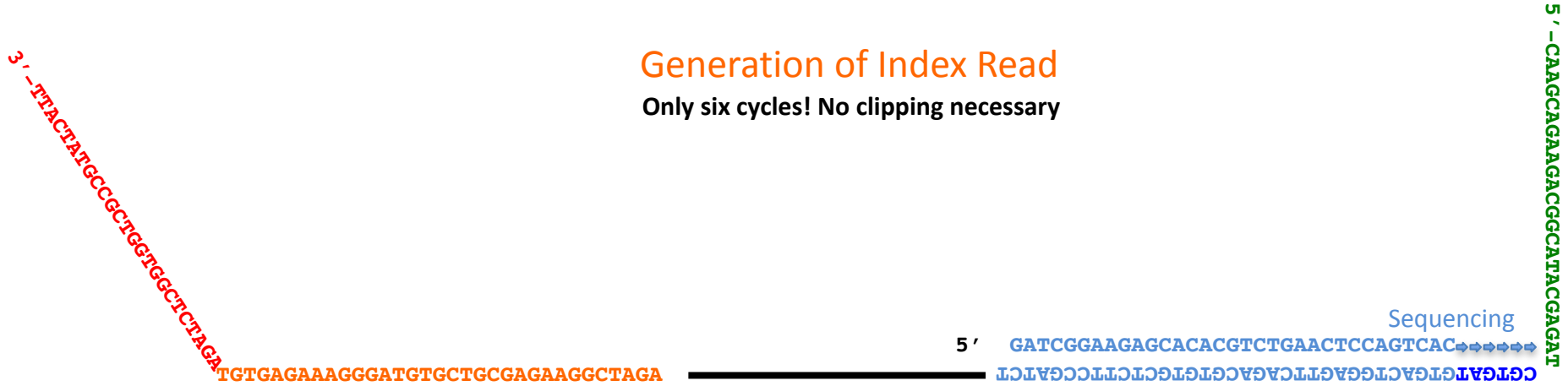
## Multiplexing Index Read Sequencing Primer

## Multiplexing Read 2 Sequencing Primer

5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index



## Generation of Index Read

Only six cycles! No clipping necessary

## Multiplexing Read 1 Sequencing Primer

5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

## Multiplexing Index Read Sequencing Primer

## Multiplexing Read 2 Sequencing Primer

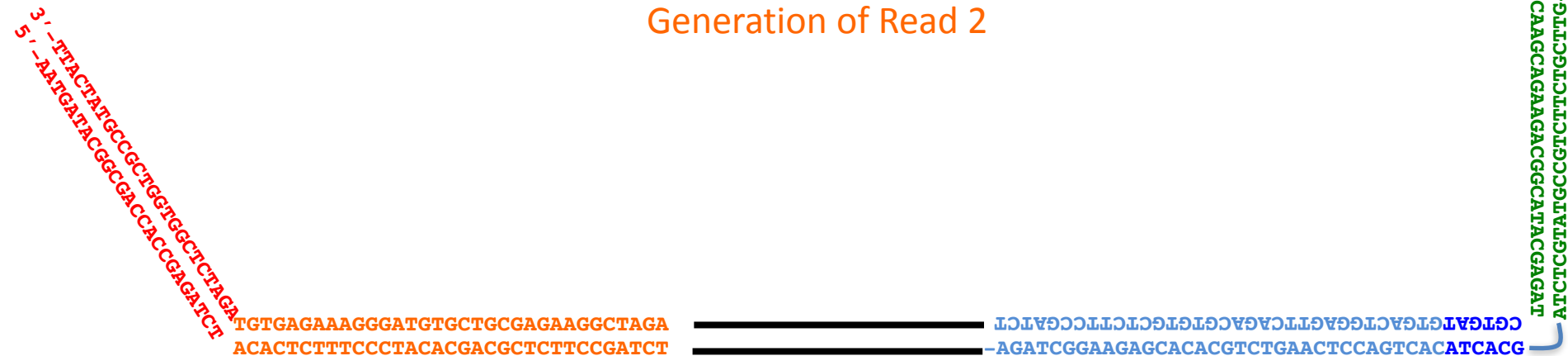
5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**



# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index

## Generation of Read 2



## Multiplexing Read 1 Sequencing Primer

5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

## Multiplexing Index Read Sequencing Primer

5' **GATCGGAAGAGCACACGTCTGAACTCCAGTCAC**

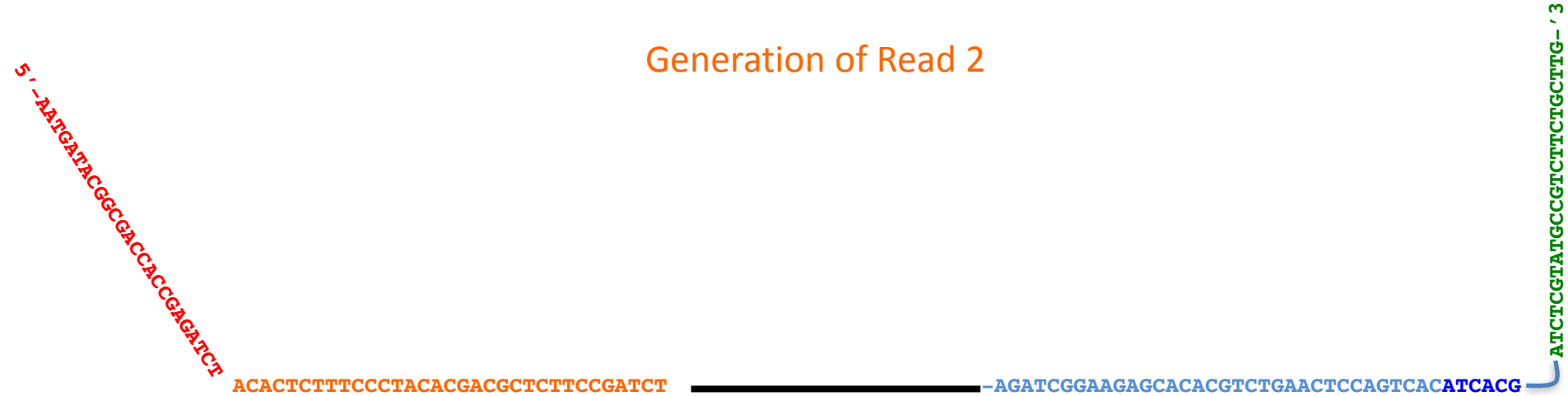
## Multiplexing Read 2 Sequencing Primer

5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index

## Generation of Read 2



### Multiplexing Read 1 Sequencing Primer

5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

### Multiplexing Index Read Sequencing Primer

5' **GATCGGAAGAGCACACGTCTGAACTCCAGTCAC**

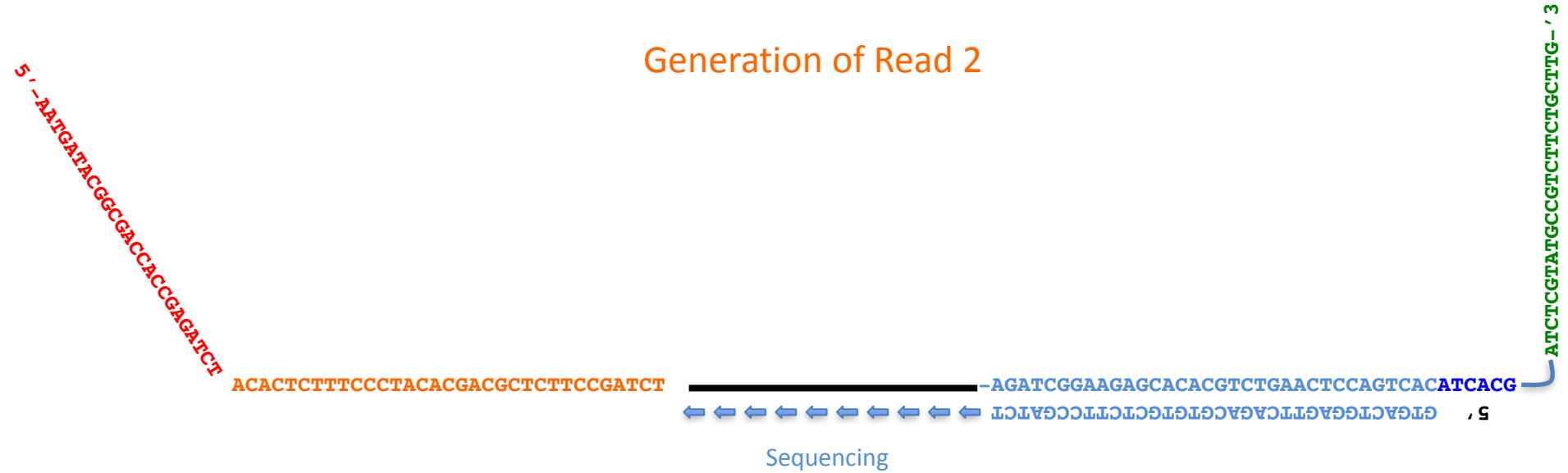
### Multiplexing Read 2 Sequencing Primer

5' **GTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT**

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index

## Generation of Read 2



Multiplexing Read 1 Sequencing Primer

5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

Multiplexing Index Read Sequencing Primer

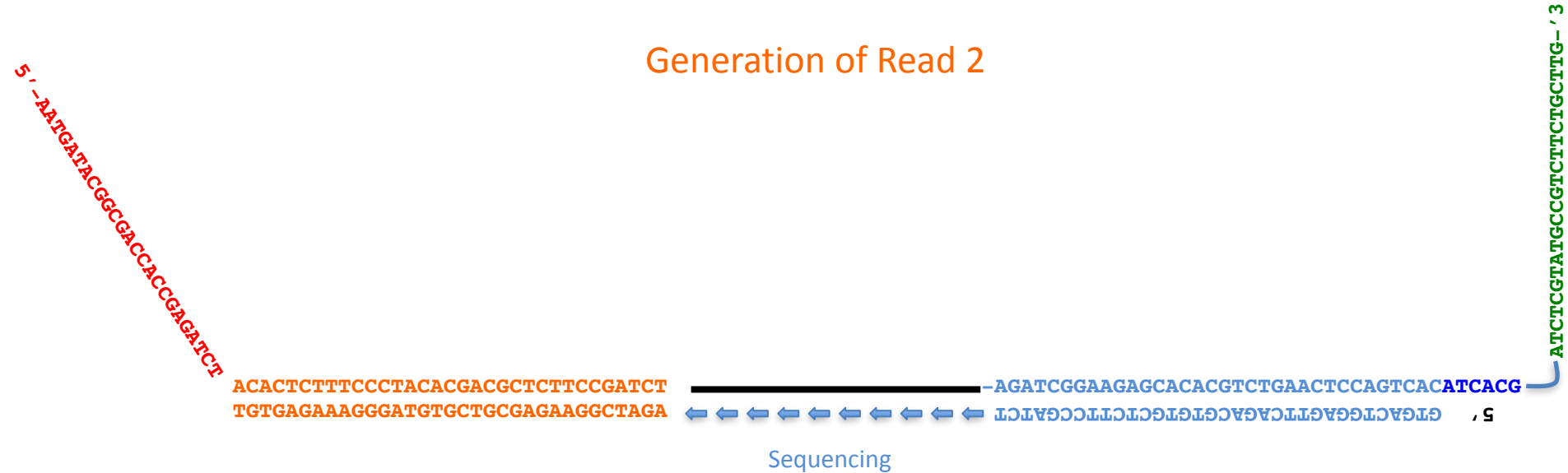
5' **GATCGGAAGAGCACACGTCTGAACTCCAGTCAC**

Multiplexing Read 2 Sequencing Primer

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index

## Generation of Read 2



### Multiplexing Read 1 Sequencing Primer

5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

### Multiplexing Index Read Sequencing Primer

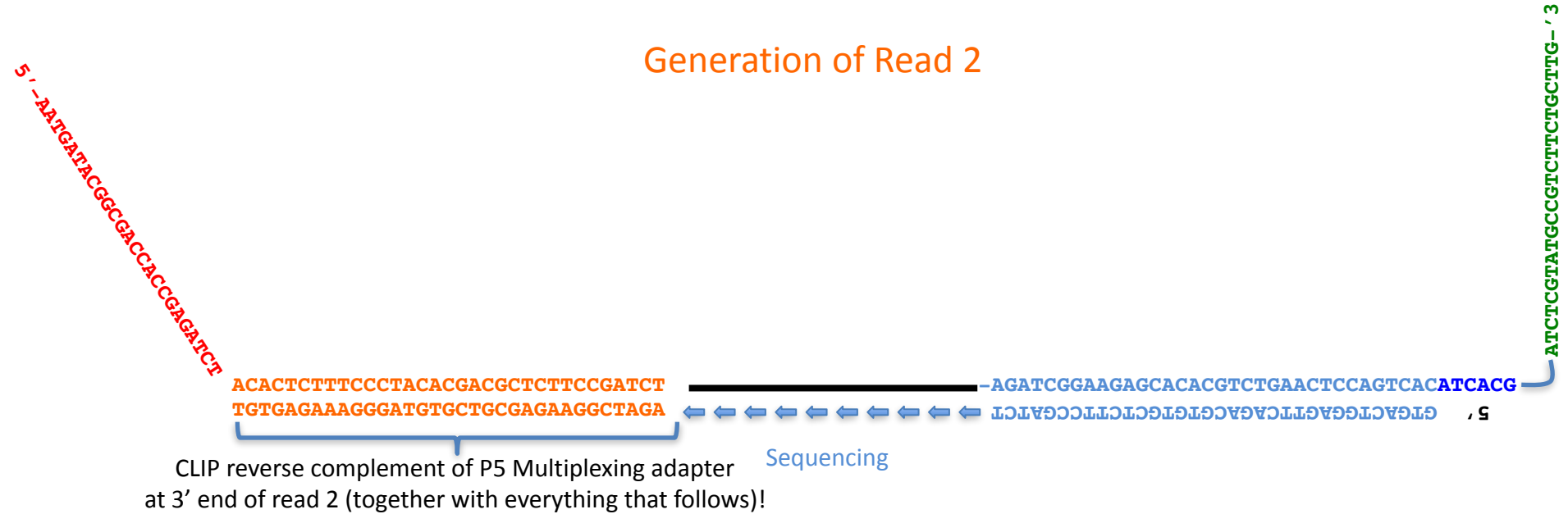
5' **GATCGGAAGAGCACACGTCTGAACTCCAGTCAC**

### Multiplexing Read 2 Sequencing Primer

# Illumina sequence library sequencing: Taking a closer look

Three sequencing primers are used to generate paired end reads and the index

## Generation of Read 2



### Multiplexing Read 1 Sequencing Primer

5' **ACACTCTTTCCCTACACGACGCTCTTCCGATCT**

### Multiplexing Index Read Sequencing Primer

5' **GATCGGAAGAGCACACGTCTGAACTCCAGTCAC**

### Multiplexing Read 2 Sequencing Primer

# There are many different kinds of libraries\*

## ▶ Single read libraries:

- Unidirectional Sequencing
- Single Read Flowcells ONLY
- Counting applications: ChIP or low coverage resequencing projects



## ▶ Paired end libraries:

- Uni- OR Bi-directional (paired reads)
- Paired End Flowcells; Single: Unidirectional only
- Most applications, #1 whole genome shotgun assembly
- Tailor insert size and distribution per project:
  - Tight size distribution – Assembly, structural rearrangement detection
  - Wide distribution libraries - Resequencing, high coverage

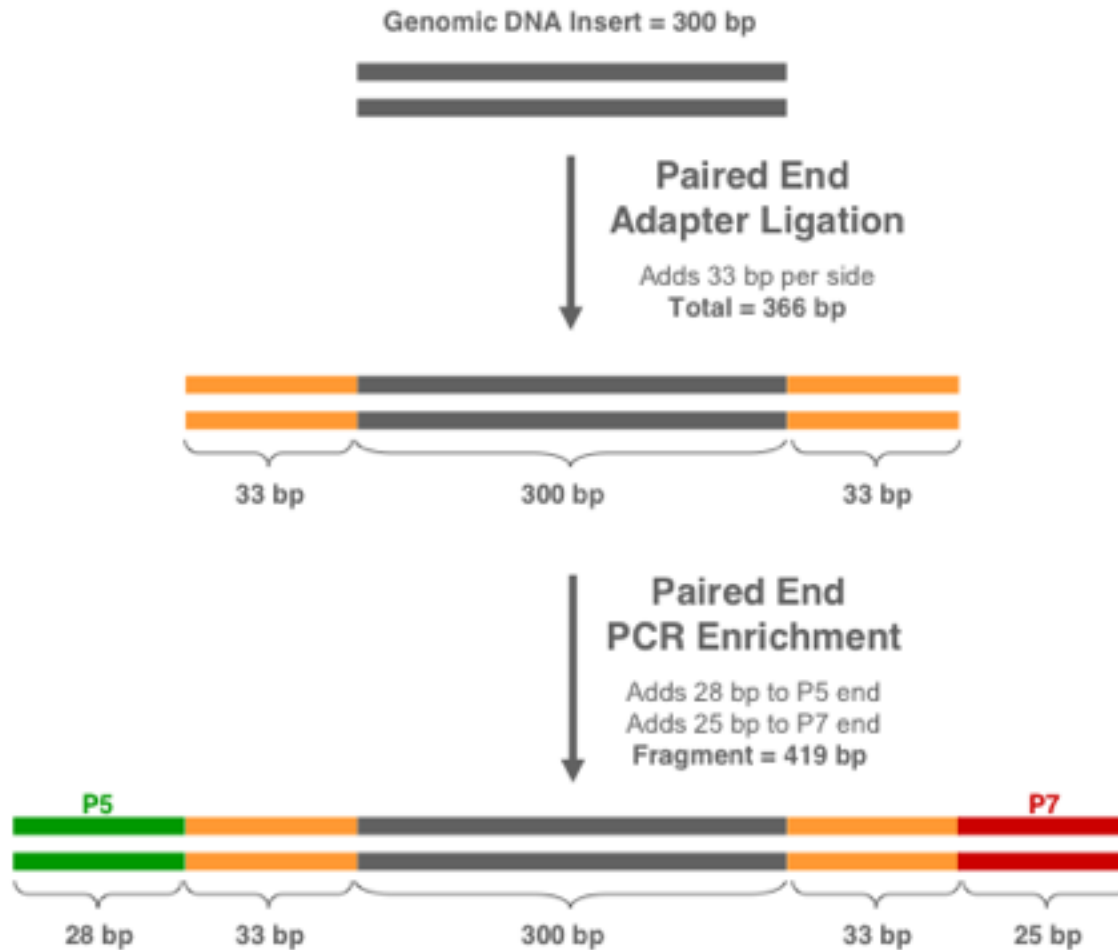


## ▶ Multiplex Paired End (aka Indexing or Barcoding)

- Uni- OR Bi-directional
- Allows multiple libraries per lane
- 12 Index tags available x 8 lanes = 96 libraries per flowcell

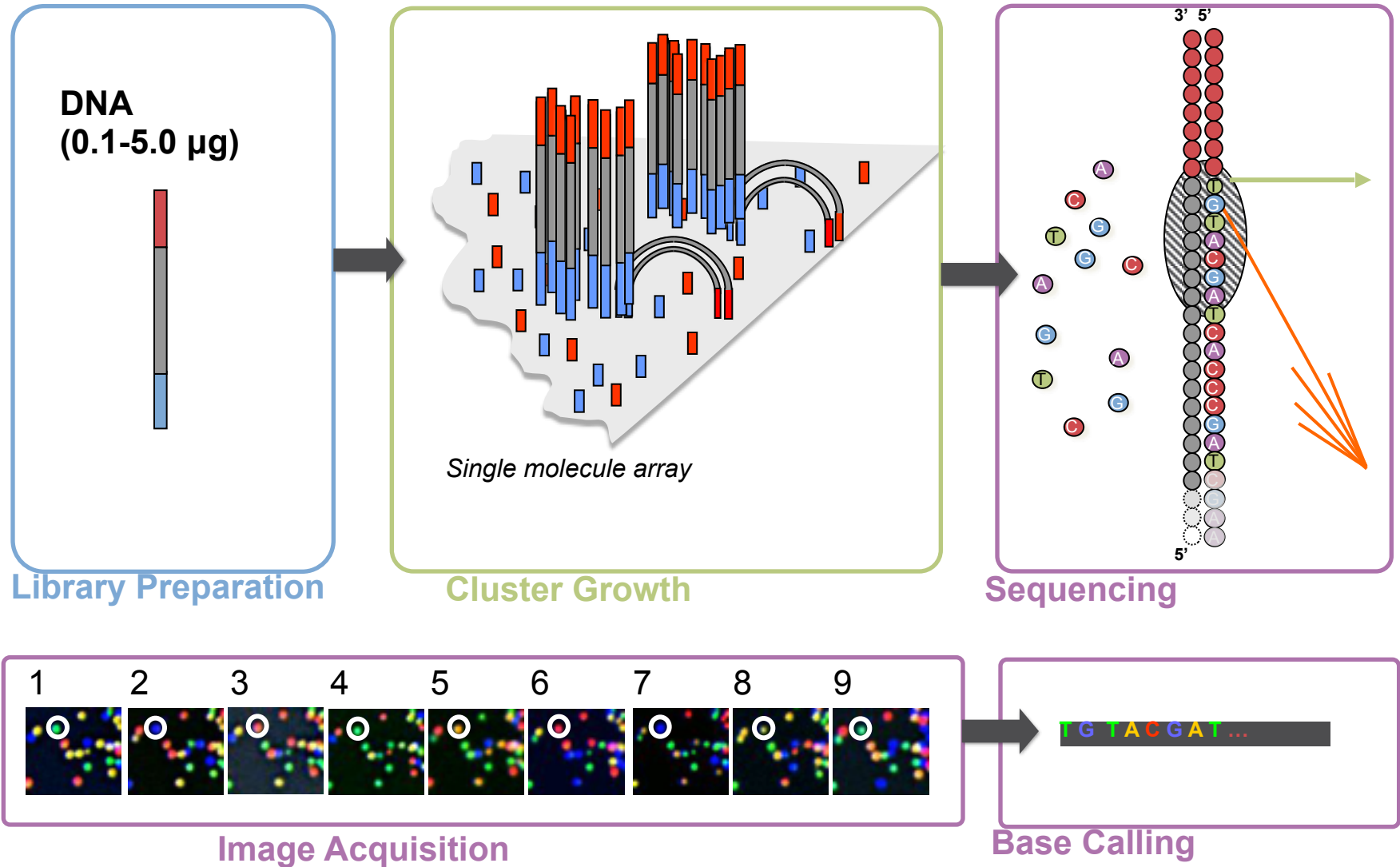


The insert size is not the same as the library fragment size\*



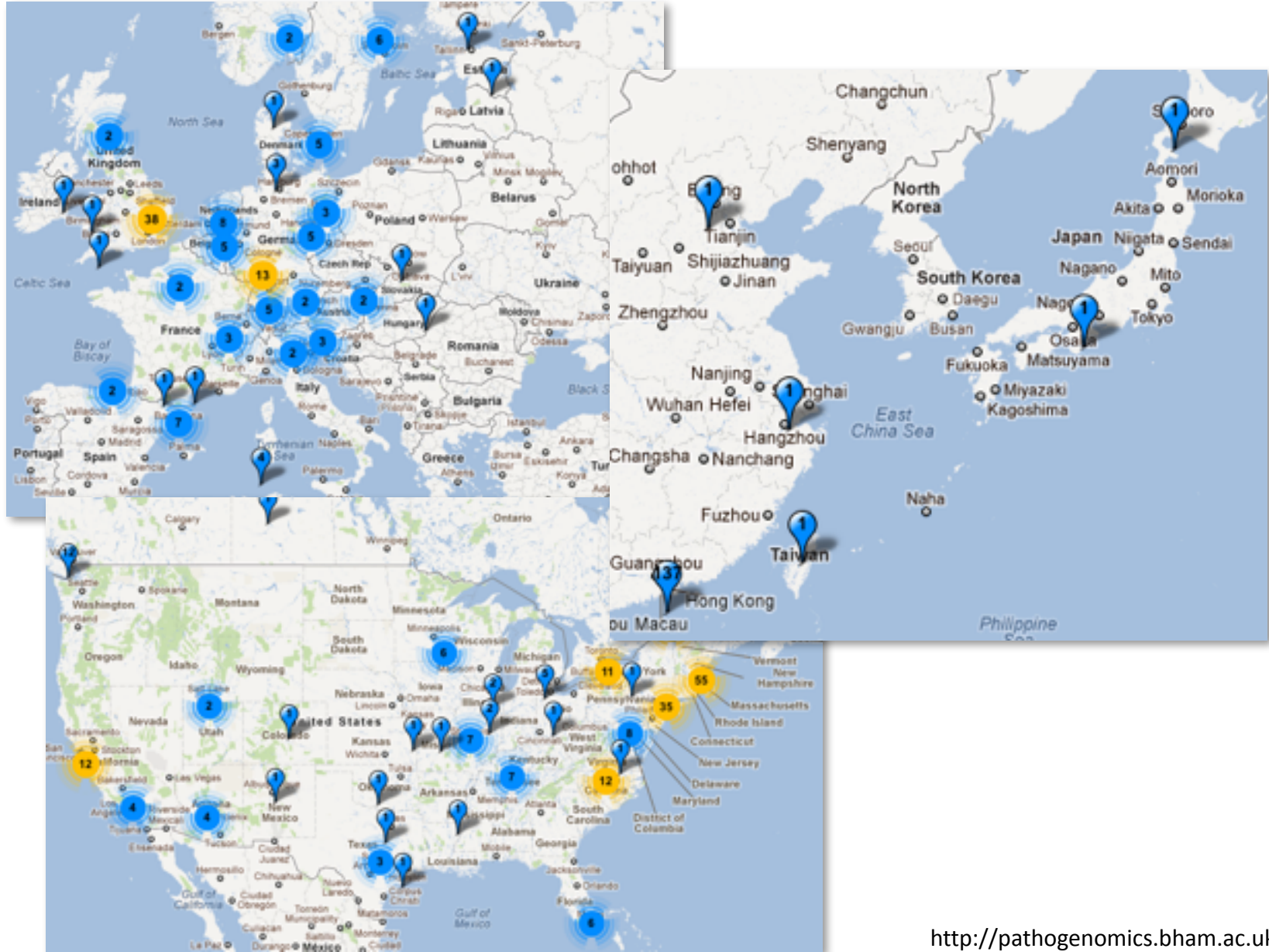
\*Make sure you know what the people in the lab have selected for!

# Illumina Sequencing Technology Overview



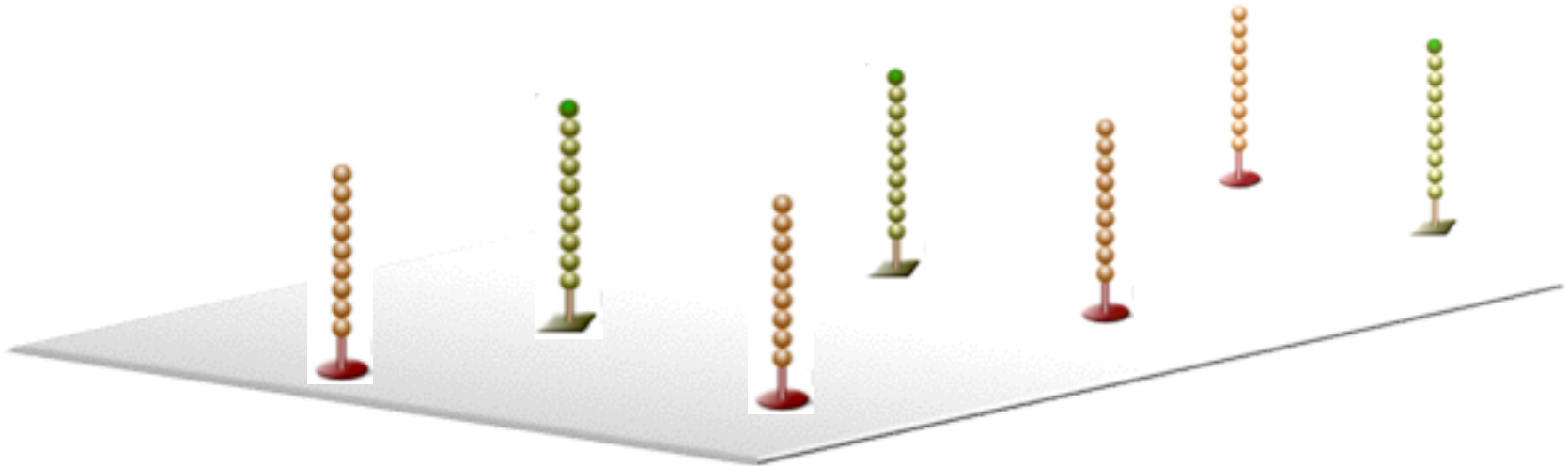


# Worldwide distribution of Illumina HighSeq machines



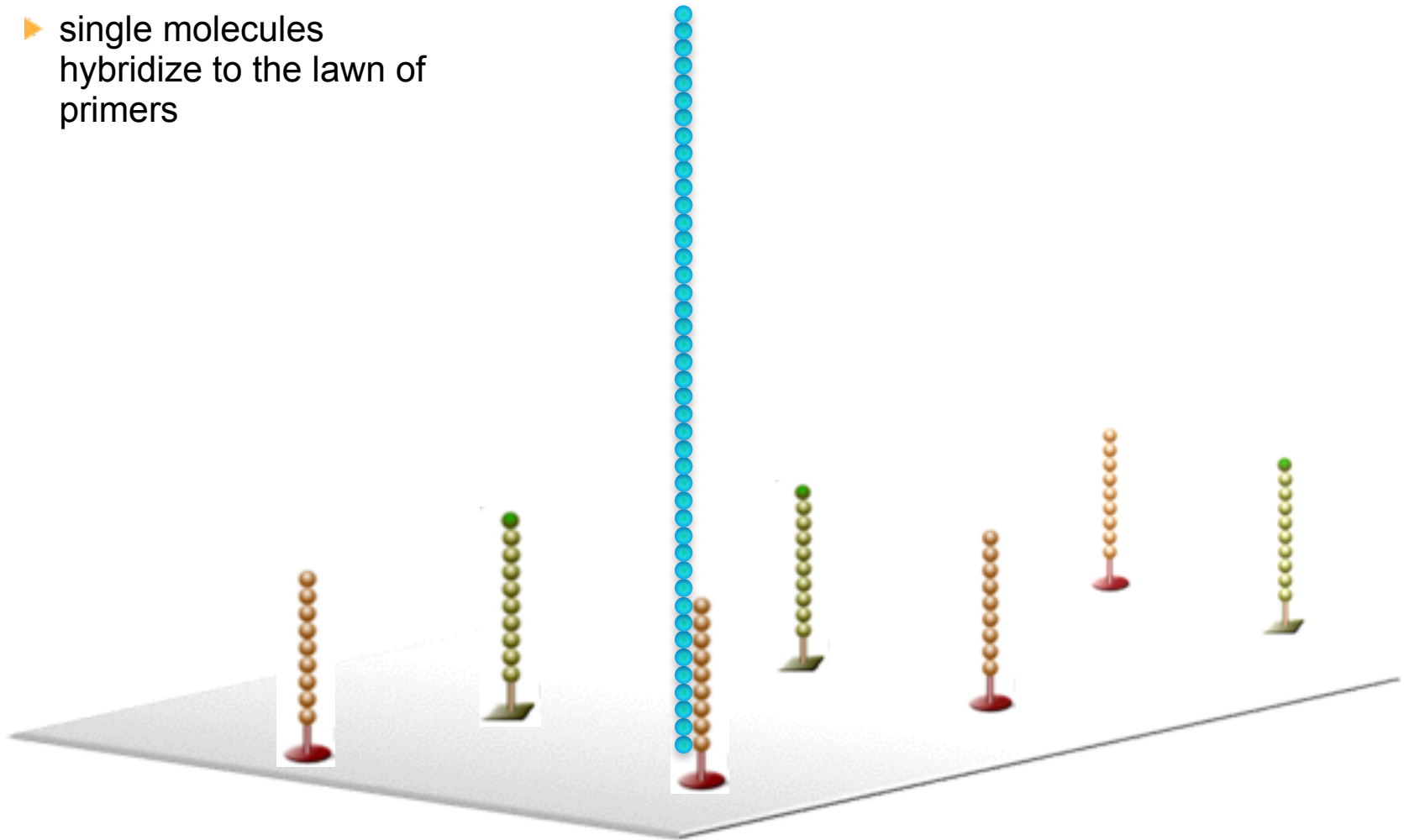
# Template hybridization and extension

- ▶ single molecules hybridize to the lawn of primers

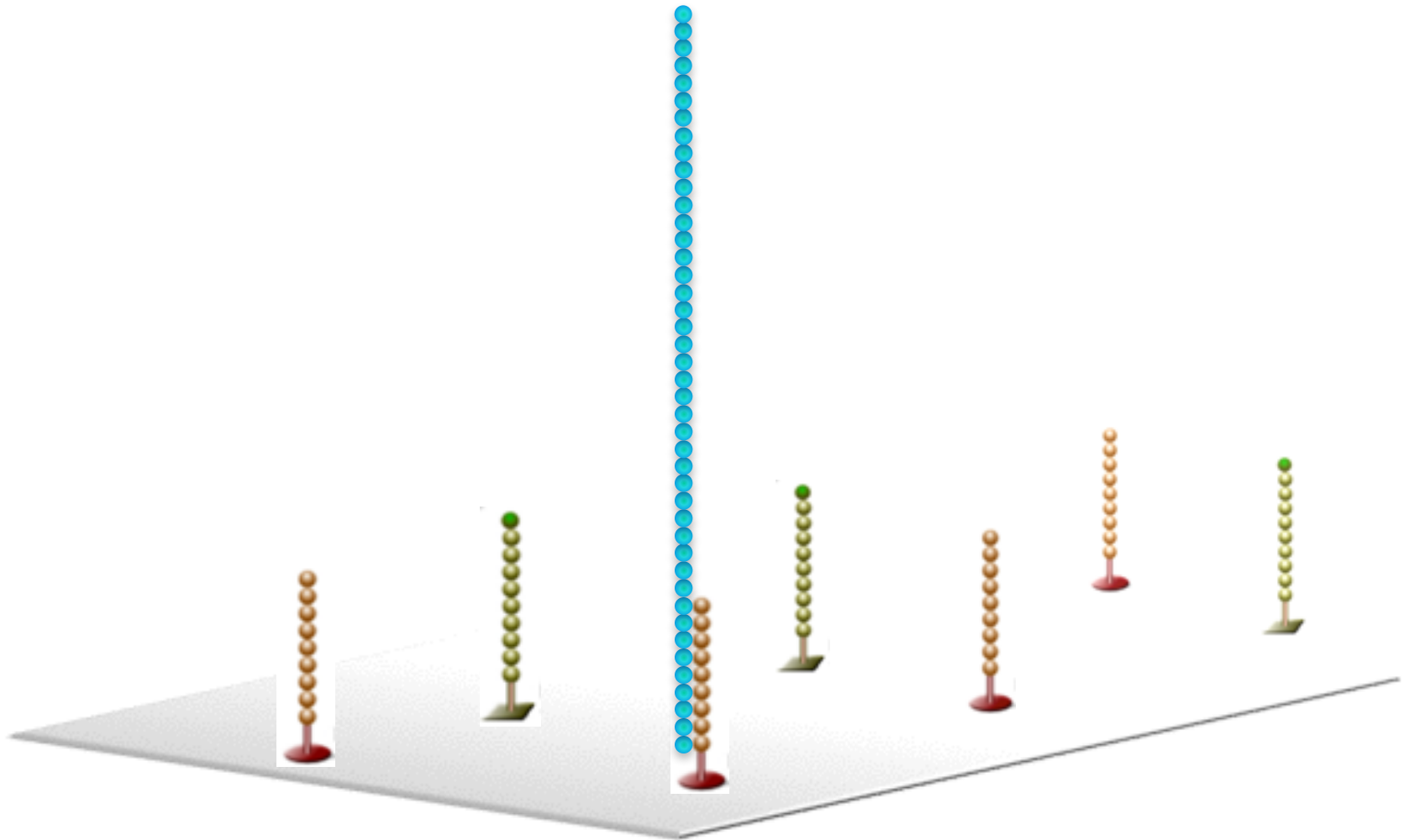


# Template hybridization and extension

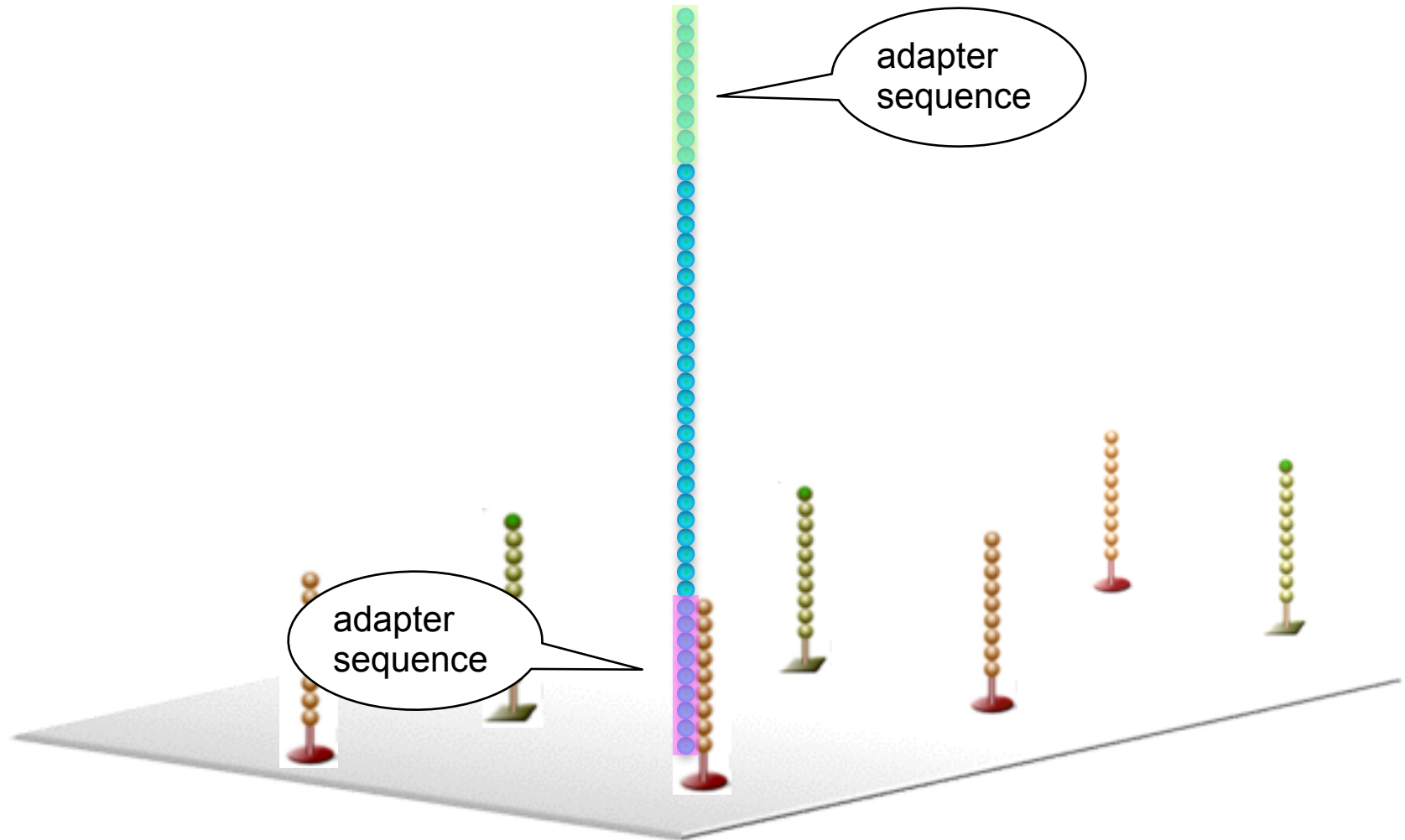
- ▶ single molecules hybridize to the lawn of primers



# Template hybridization and extension

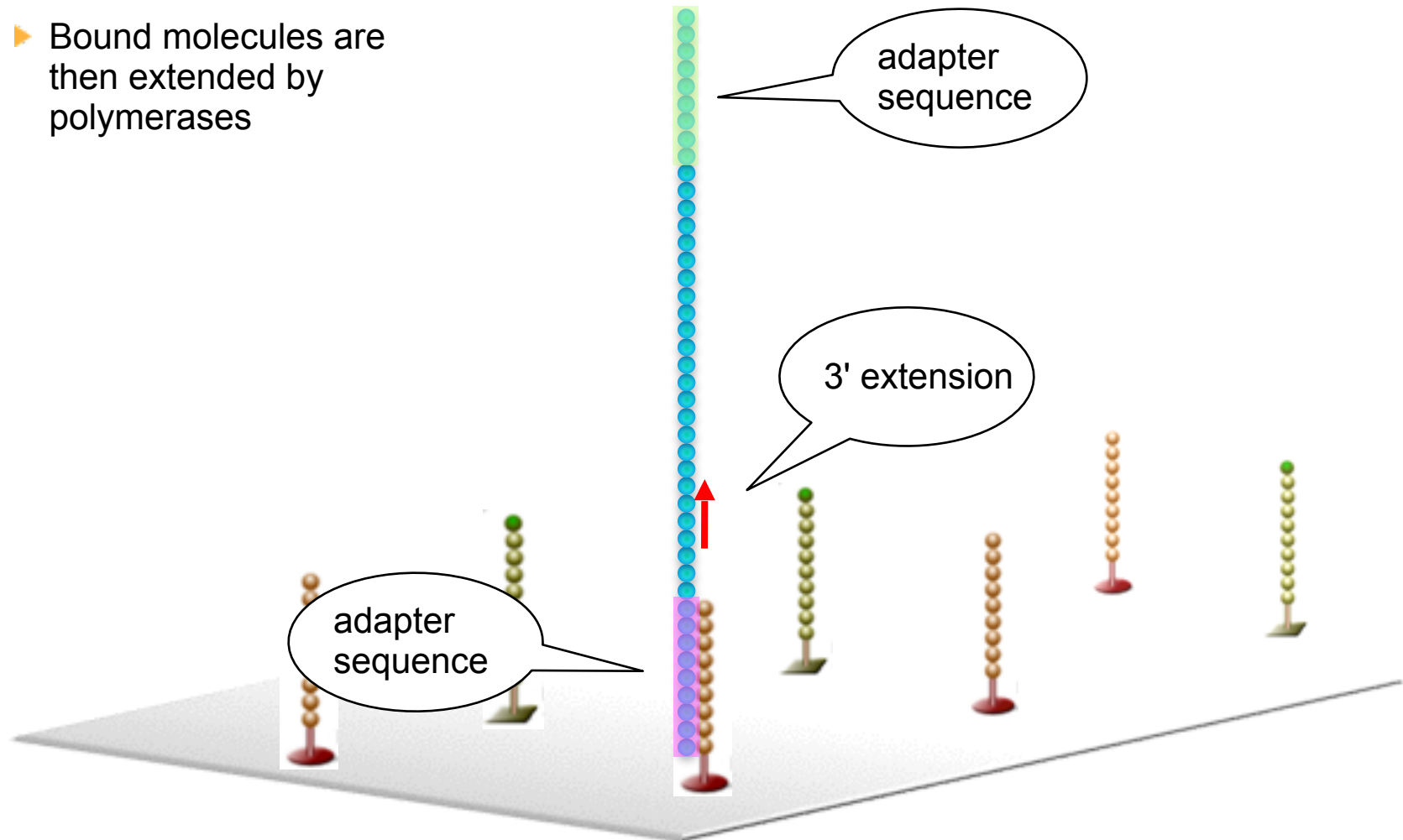


# Template hybridization and extension



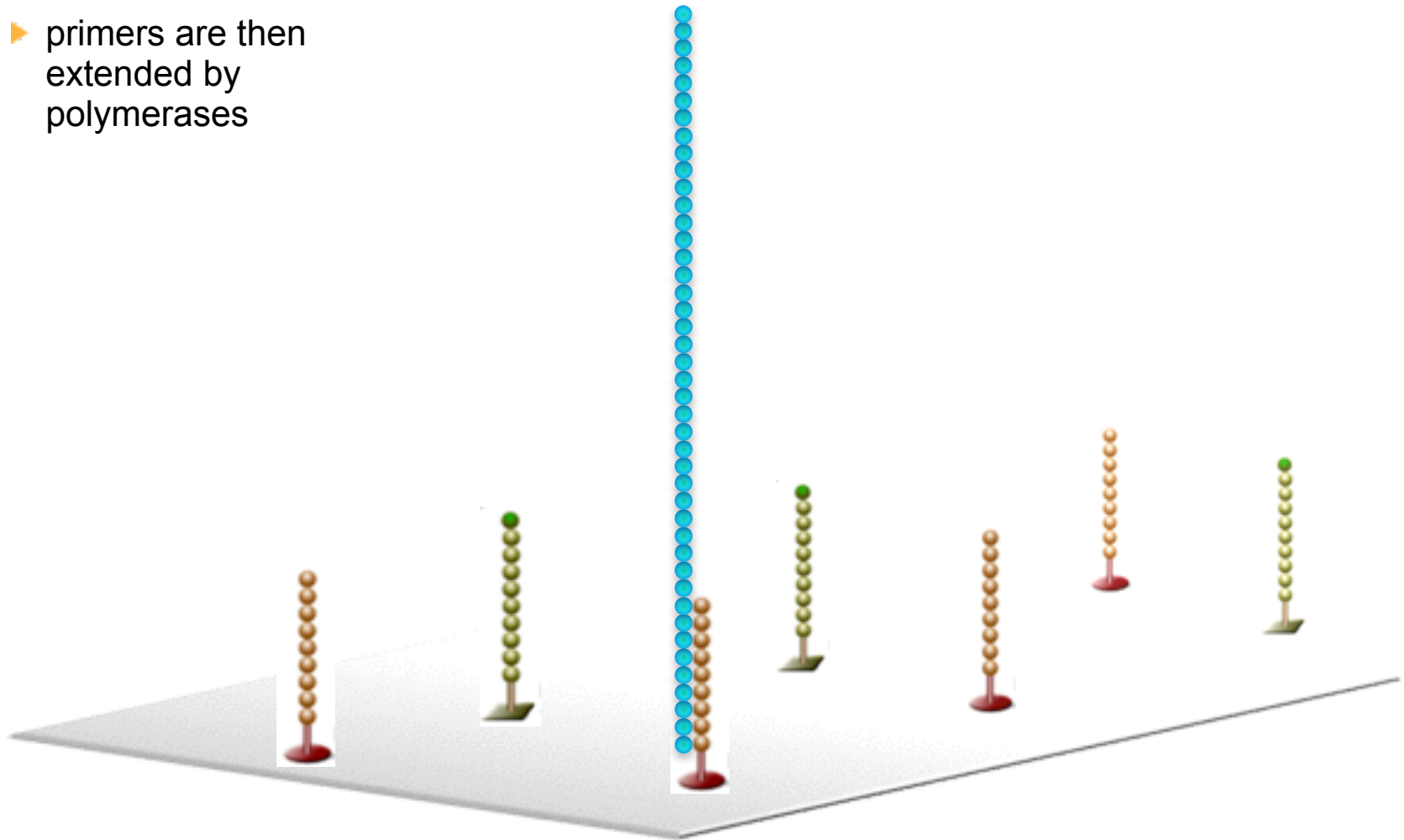
# Template hybridization and extension

- ▶ Bound molecules are then extended by polymerases



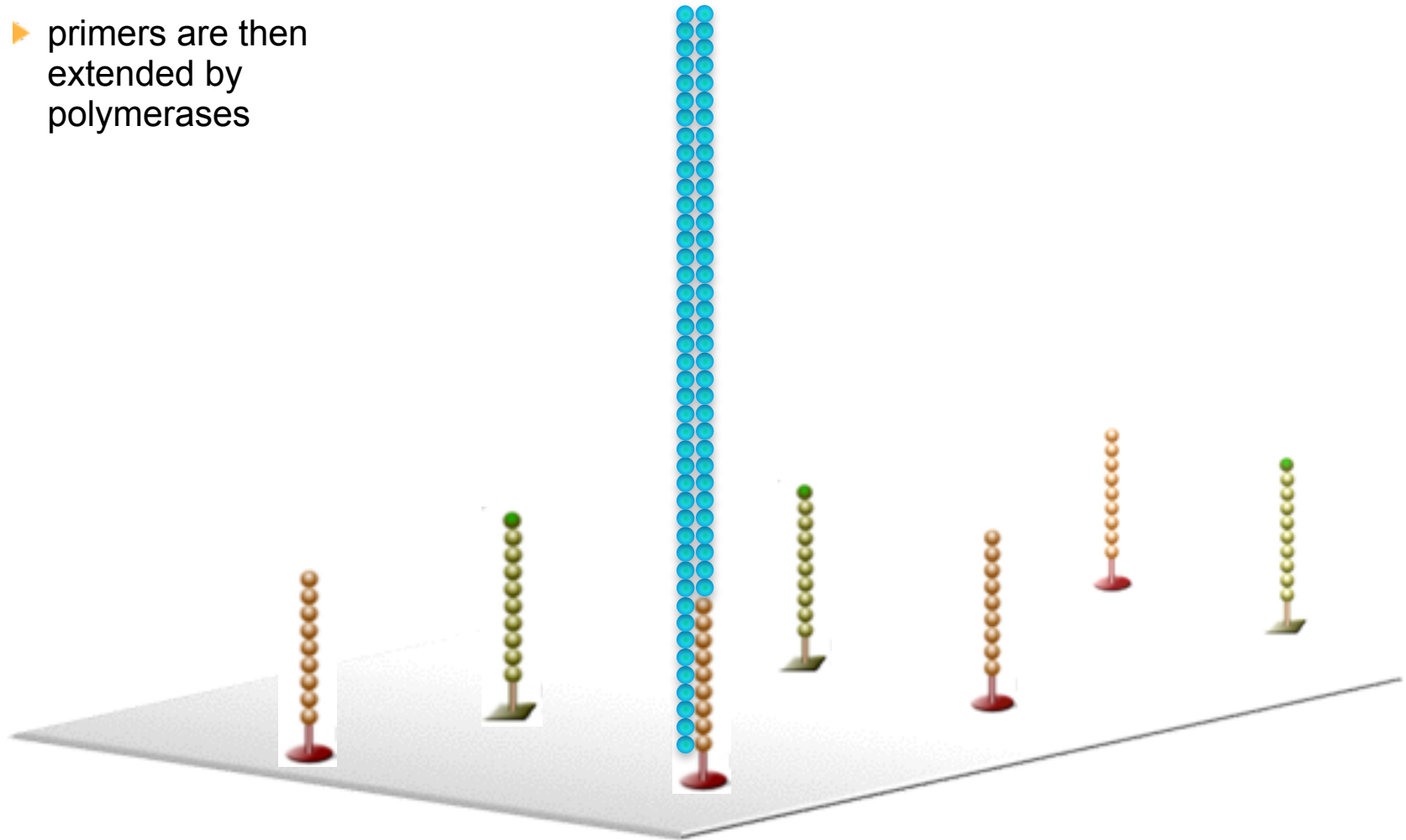
# Template hybridization and extension

- ▶ primers are then extended by polymerases



# Template hybridization and extension

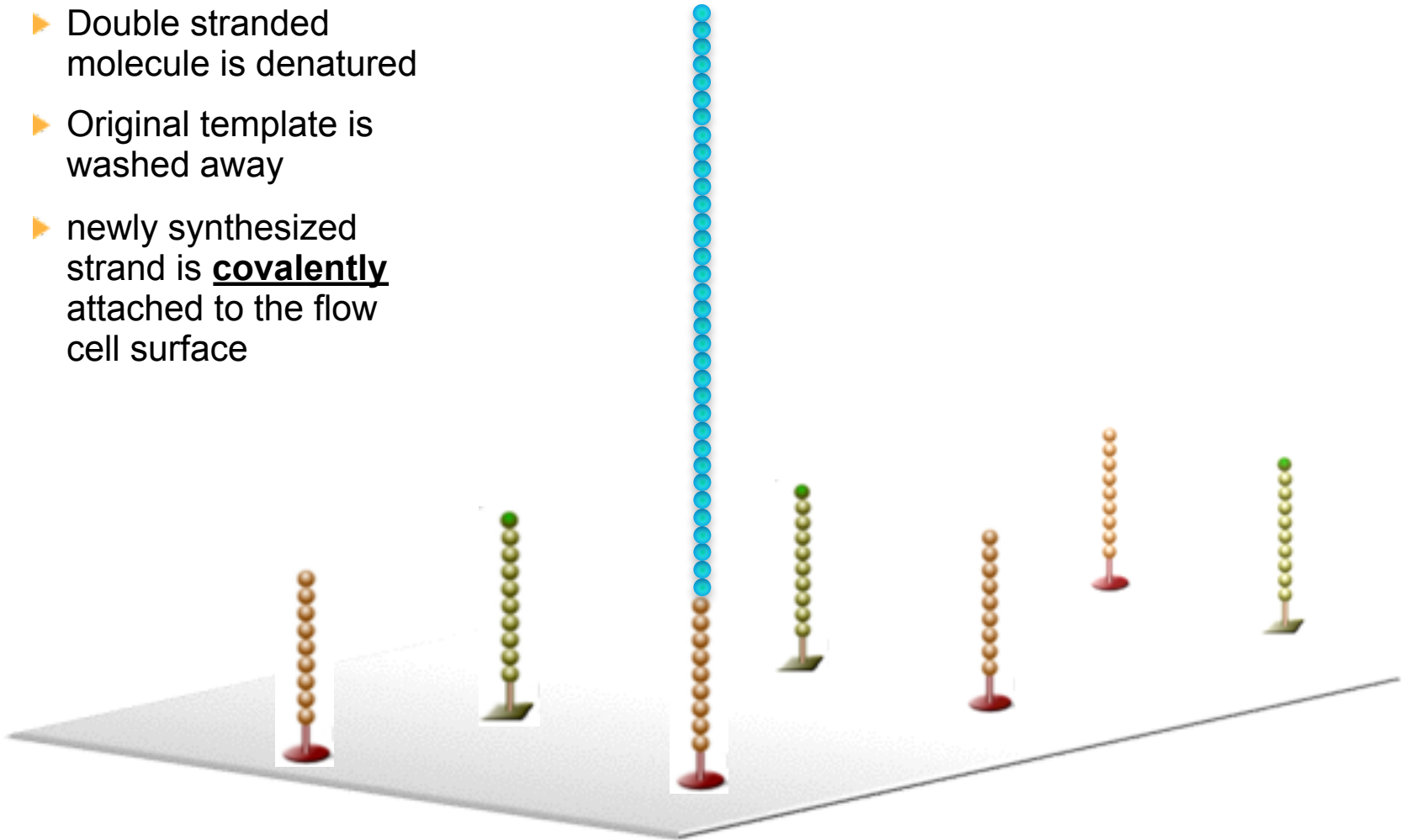
- ▶ primers are then extended by polymerases





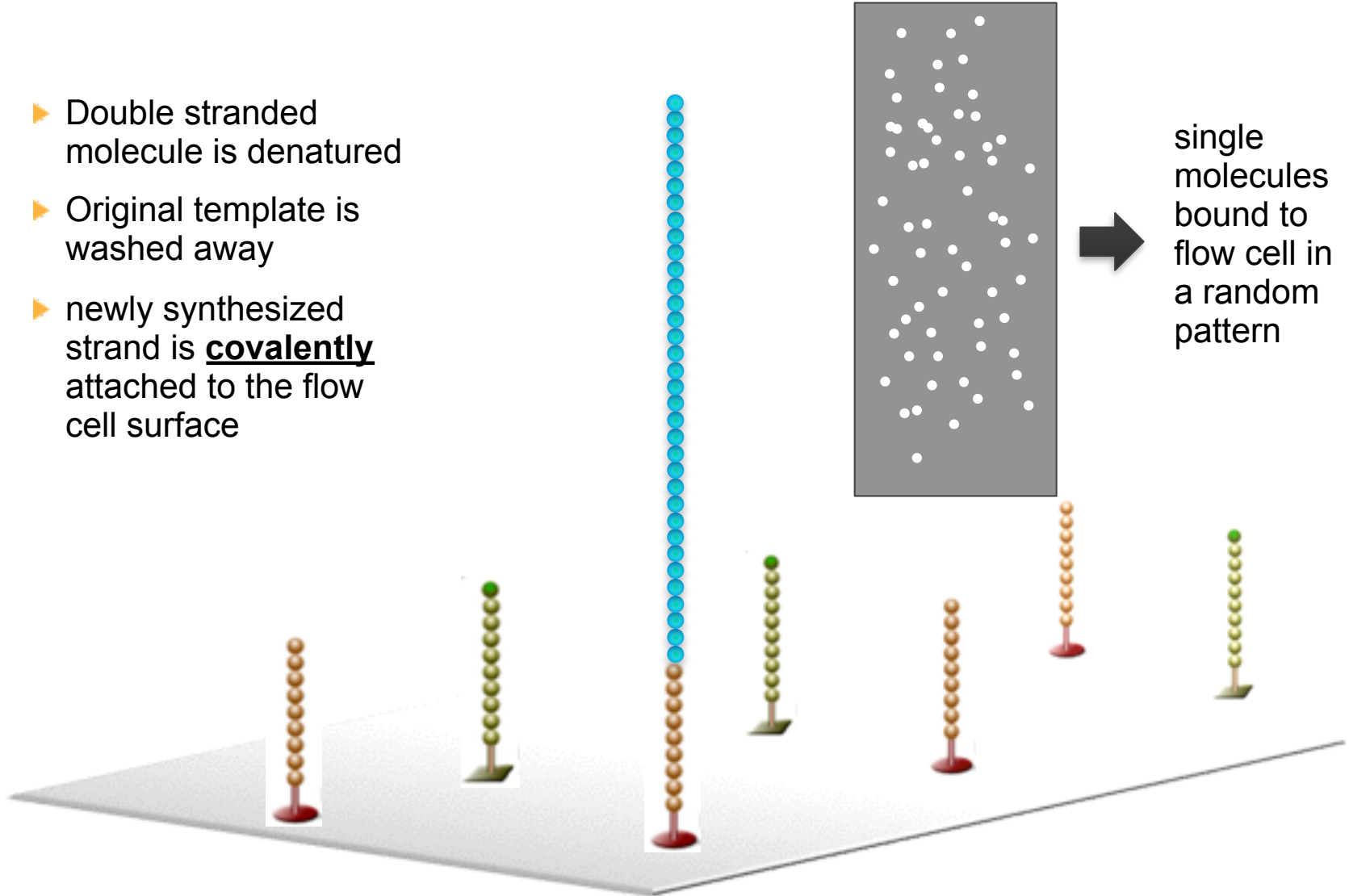
## Removal of original strand

- ▶ Double stranded molecule is denatured
- ▶ Original template is washed away
- ▶ newly synthesized strand is **covalently** attached to the flow cell surface

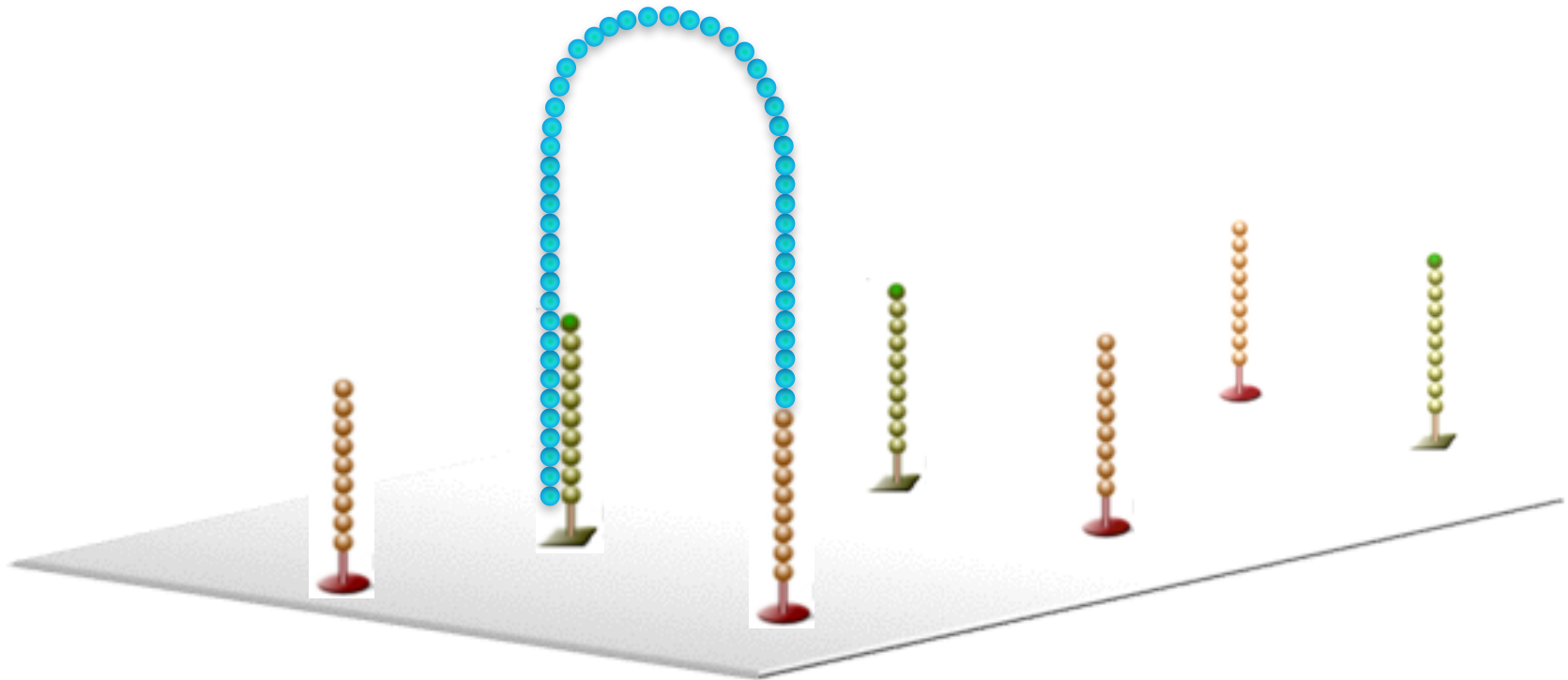


## Removal of original strand

- ▶ Double stranded molecule is denatured
- ▶ Original template is washed away
- ▶ newly synthesized strand is **covalently** attached to the flow cell surface

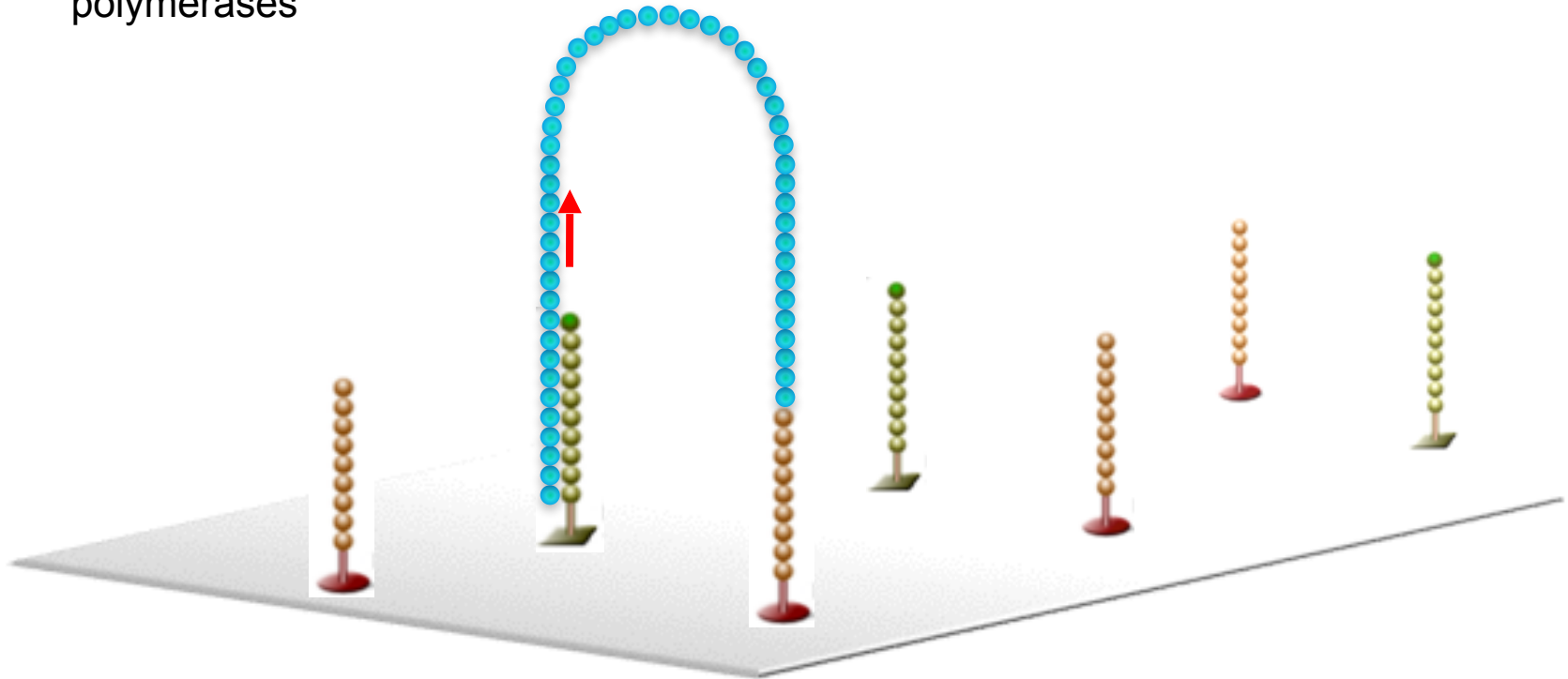


# Bridging over



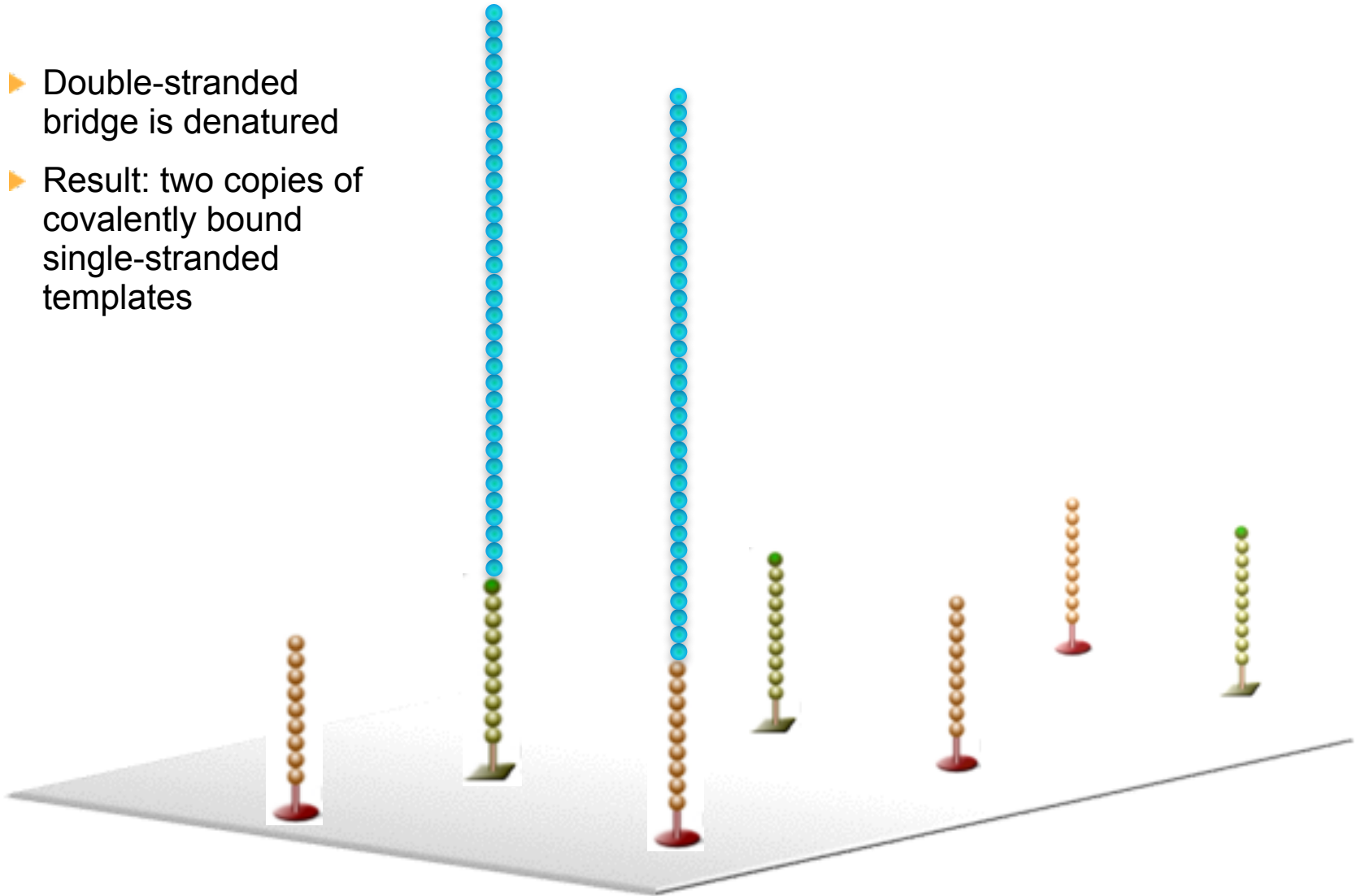
# Bridging over

- ▶ Single-strand flips over to hybridize to adjacent oligos to form a bridge
- Hybridized primer is extended by polymerases

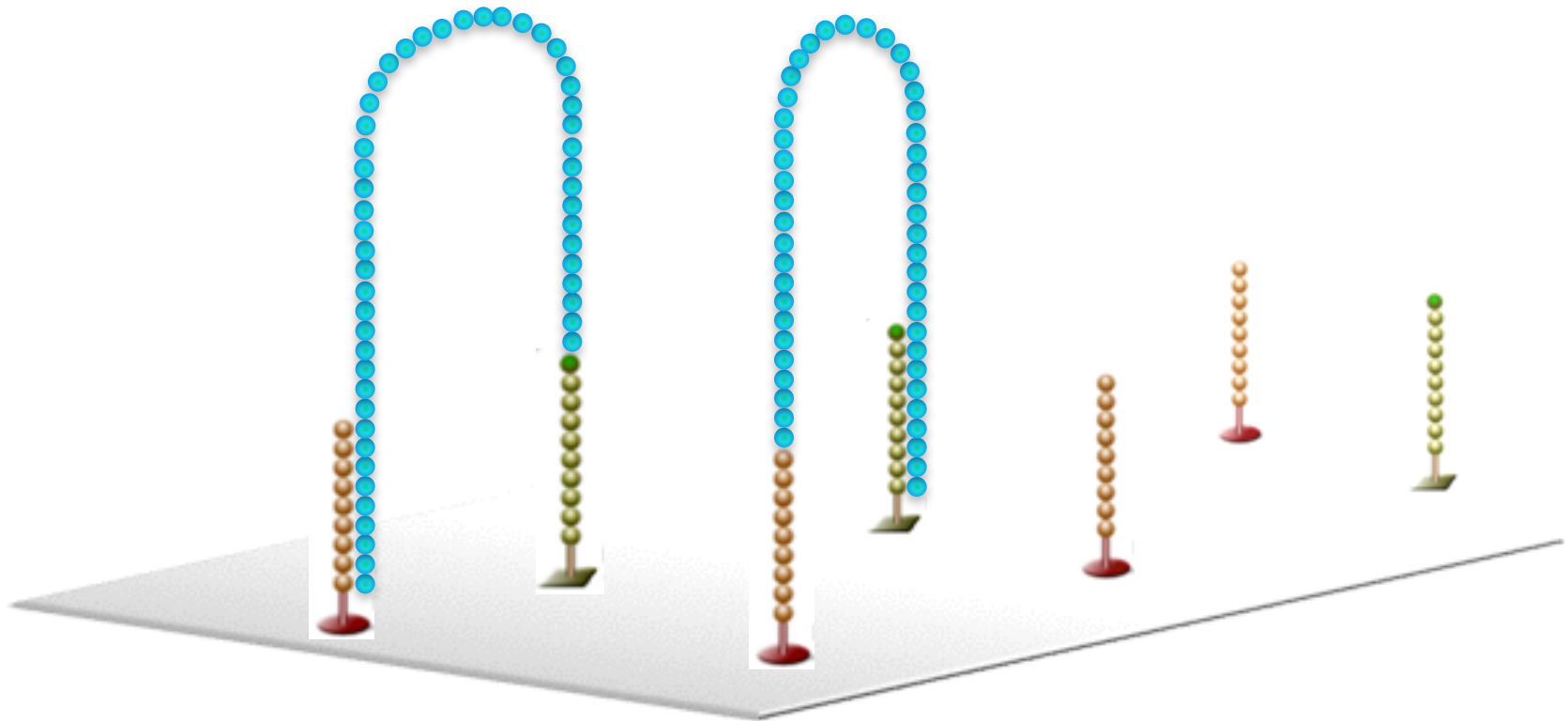


# Denaturation

- ▶ Double-stranded bridge is denatured
- ▶ Result: two copies of covalently bound single-stranded templates

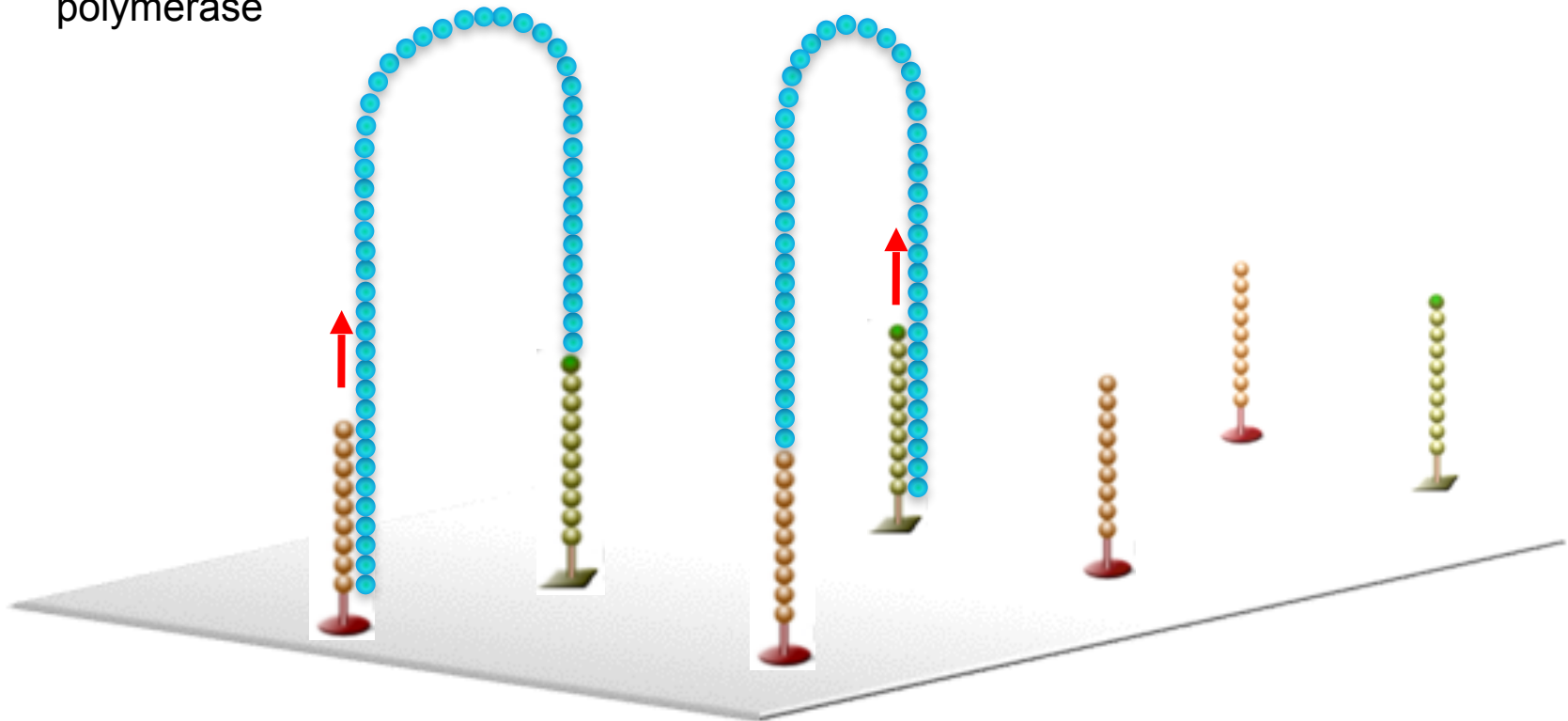


# Bridging over of templates



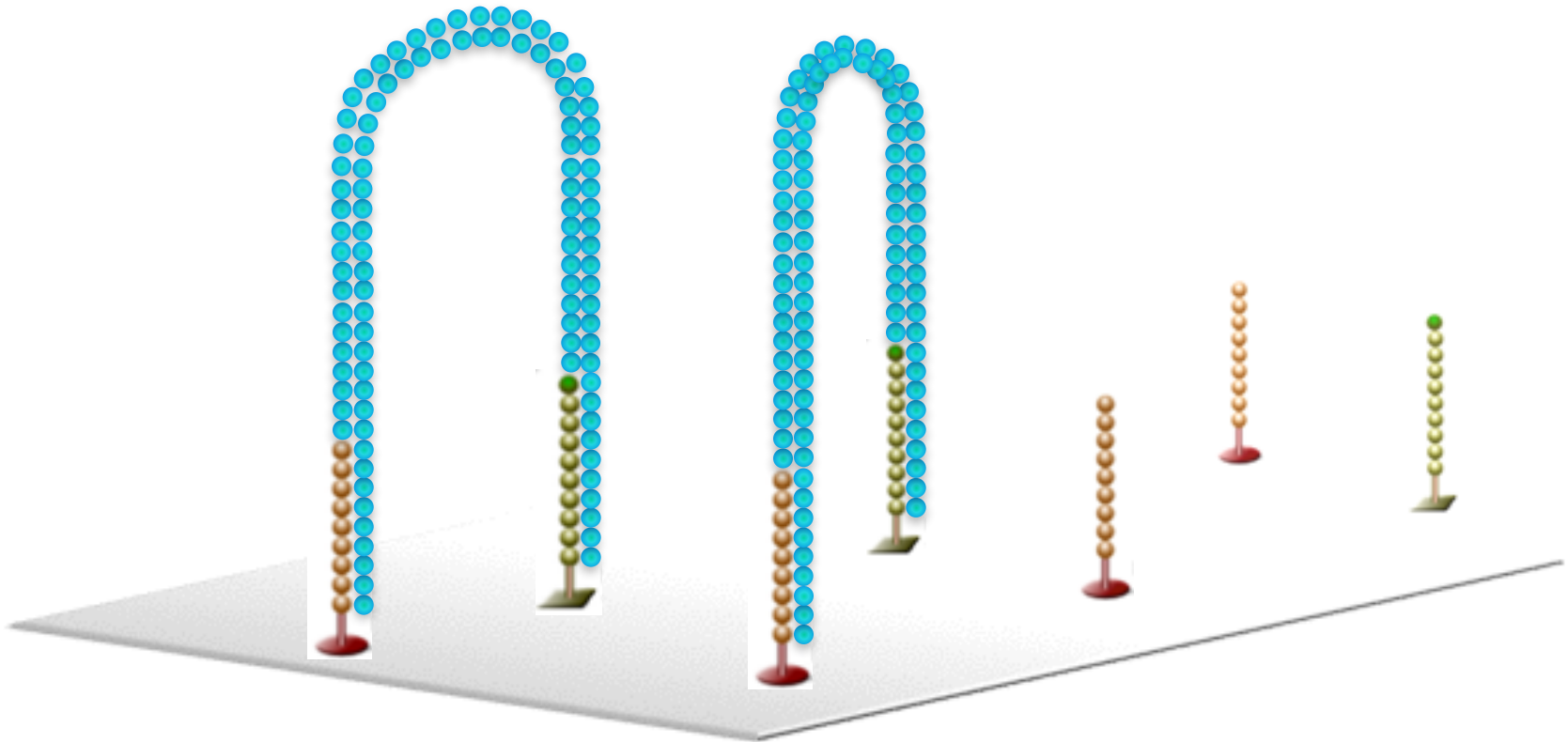
# Bridging over of templates

- ▶ Single-strands flip over to hybridize to adjacent oligos to form bridges
- Hybridized primer is extended by polymerase



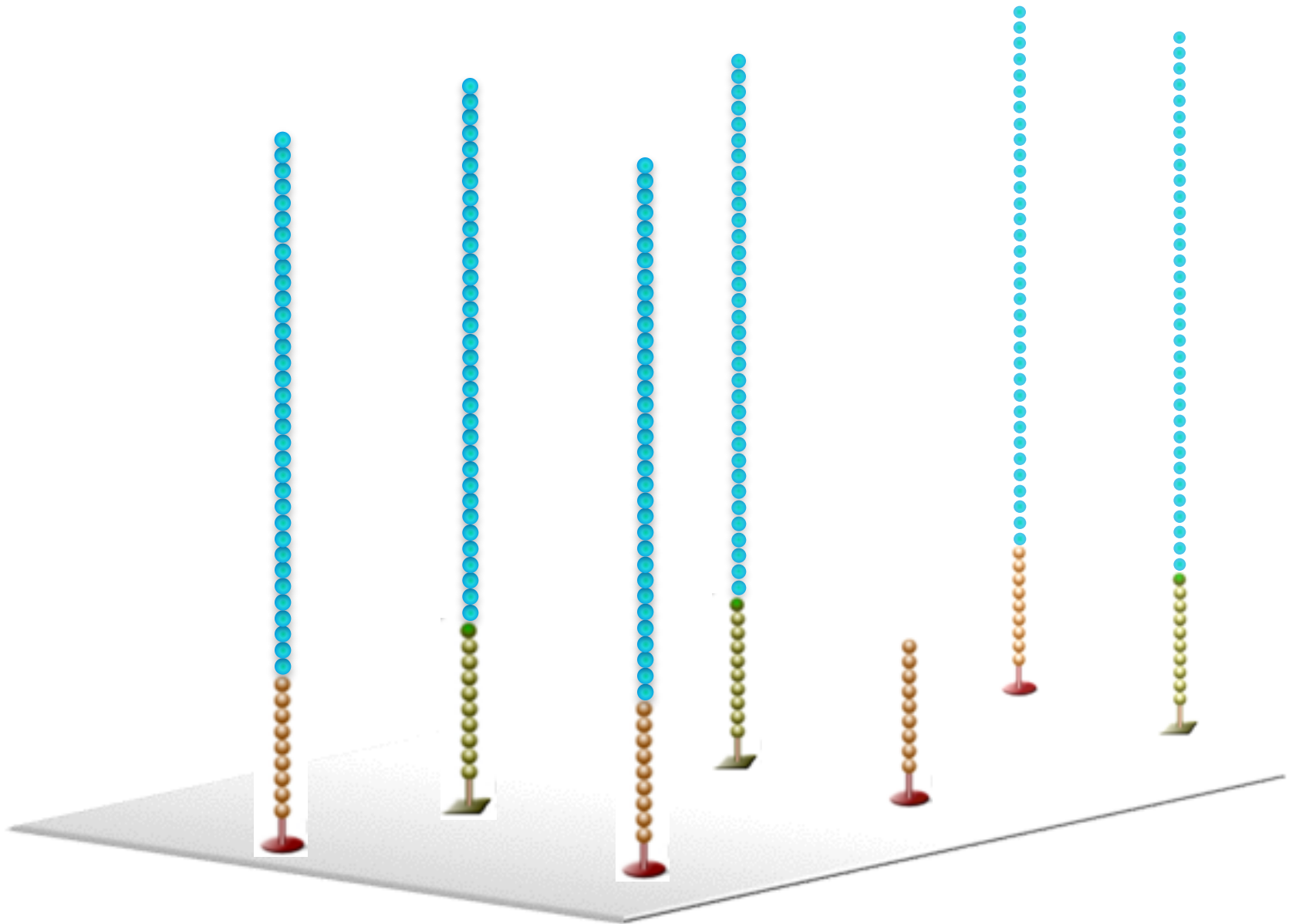
# Amplification

- ▶ Bridge amplification cycle repeated until multiple bridges are formed across the entire flow cell



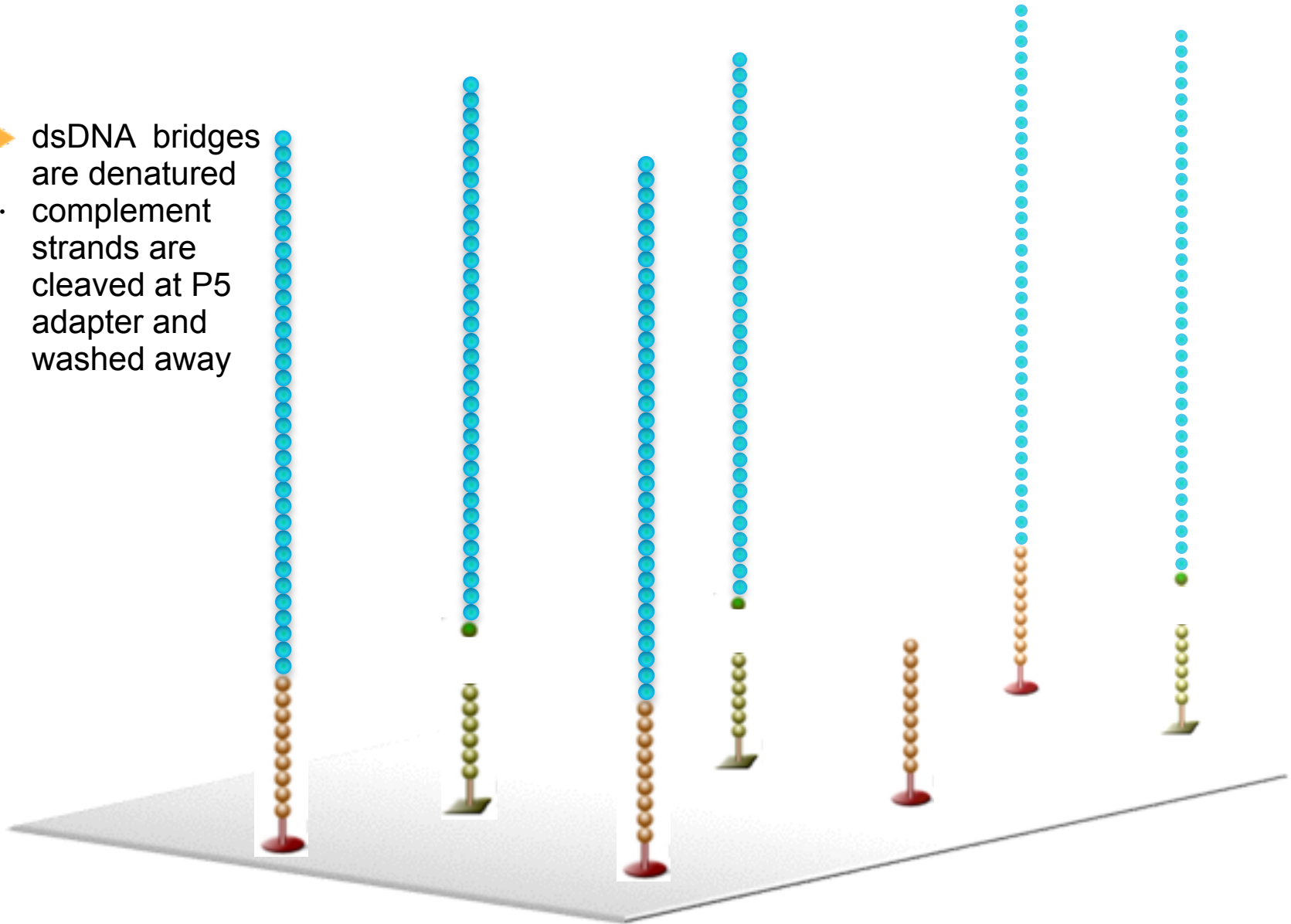


# Linearization



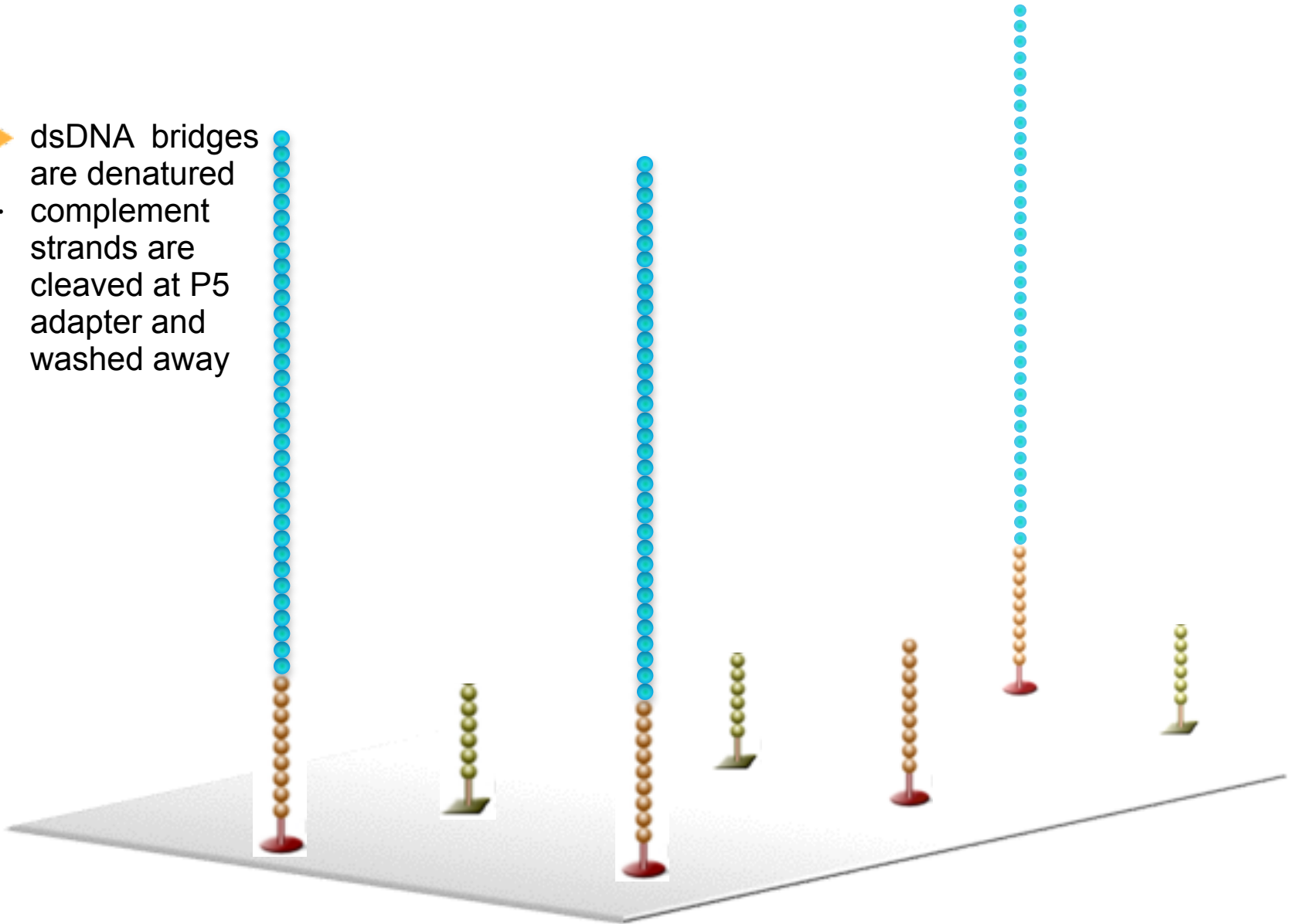
# Linearization

- ▶ dsDNA bridges are denatured
- complement strands are cleaved at P5 adapter and washed away



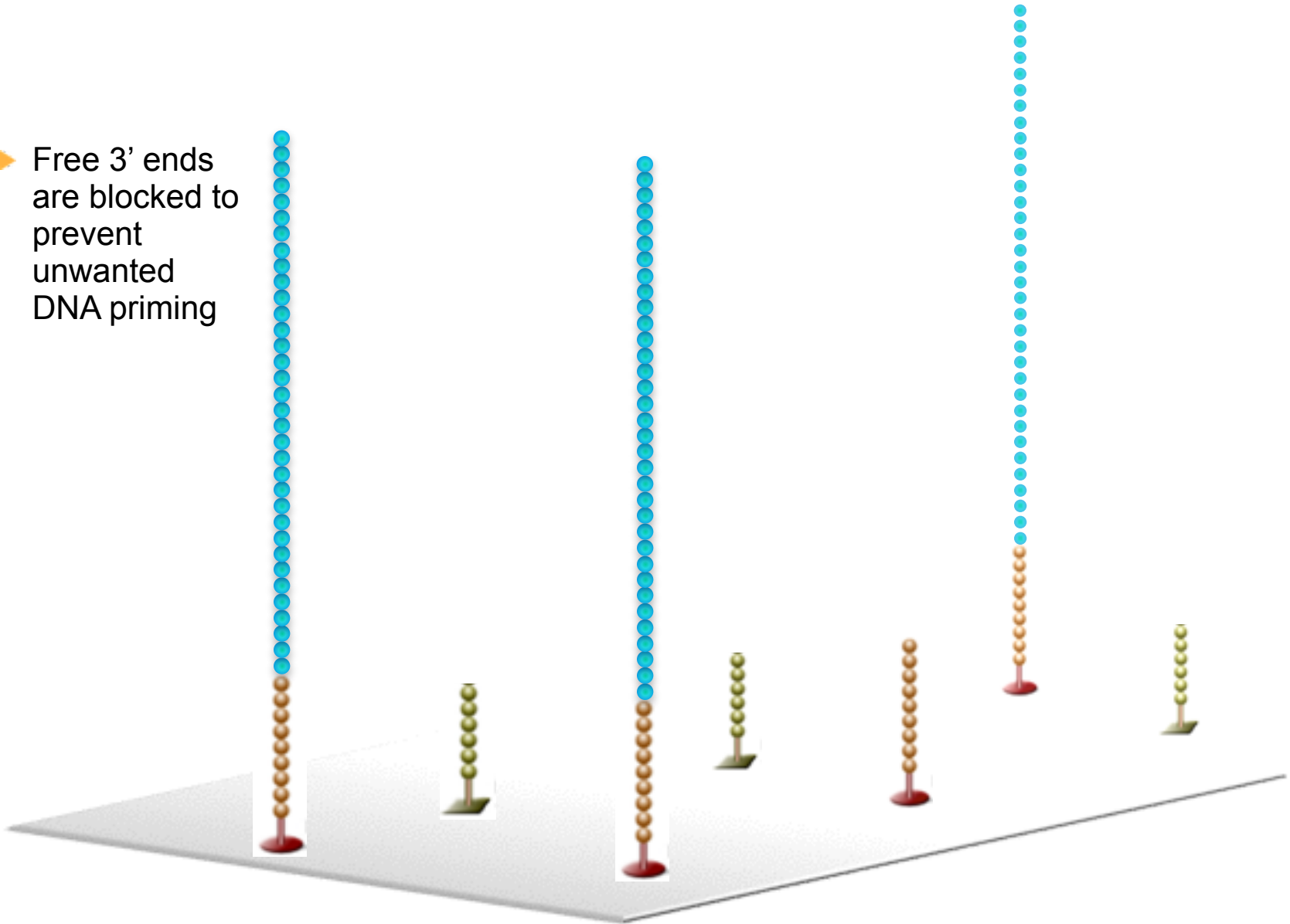
# Linearization

- ▶ dsDNA bridges are denatured
- complement strands are cleaved at P5 adapter and washed away



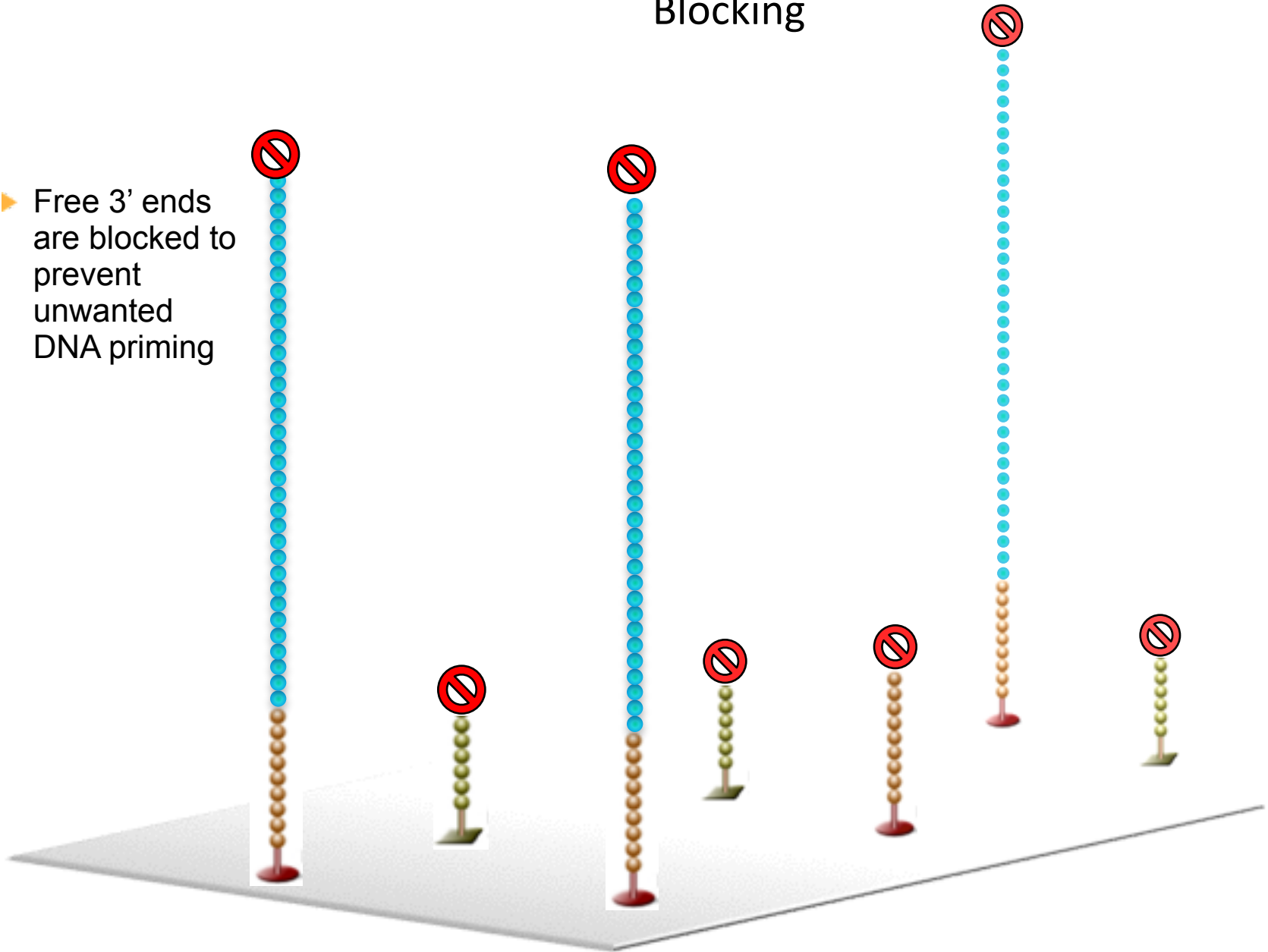
# Blocking

- ▶ Free 3' ends are blocked to prevent unwanted DNA priming



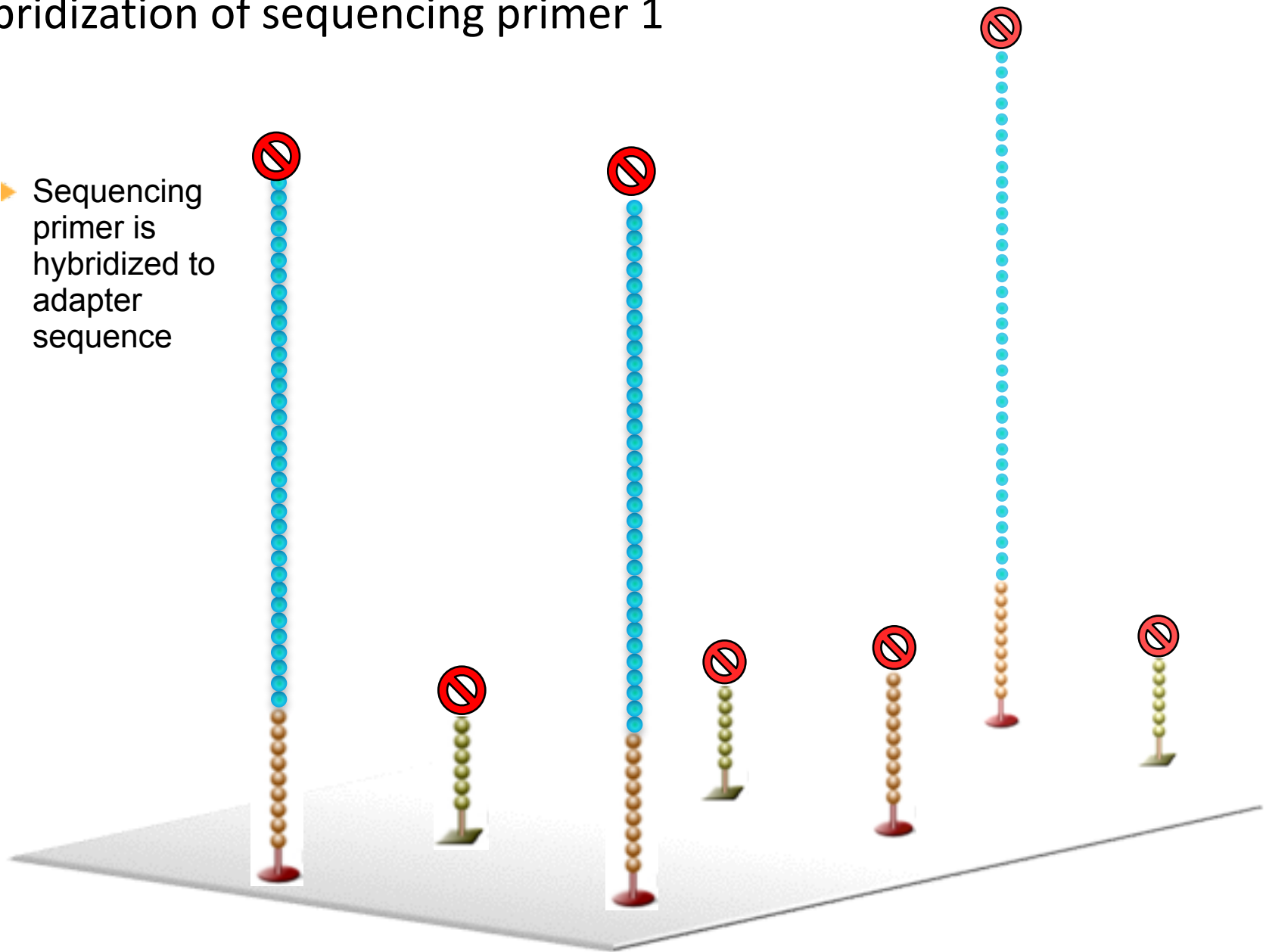
# Blocking

▶ Free 3' ends are blocked to prevent unwanted DNA priming



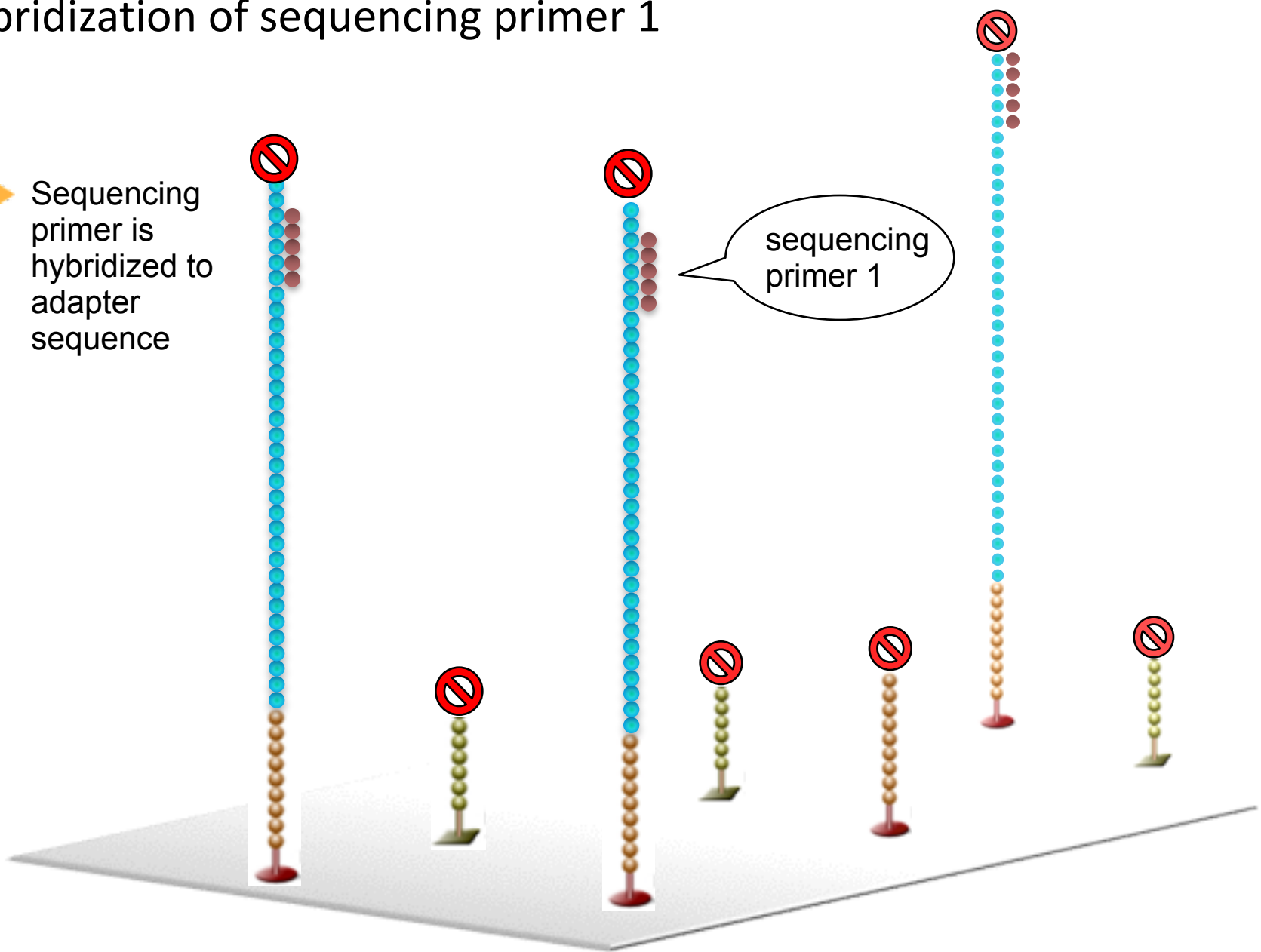
# Hybridization of sequencing primer 1

▶ Sequencing primer is hybridized to adapter sequence



# Hybridization of sequencing primer 1

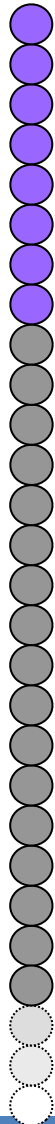
▶ Sequencing primer is hybridized to adapter sequence



# Sequencing By Synthesis (SBS)



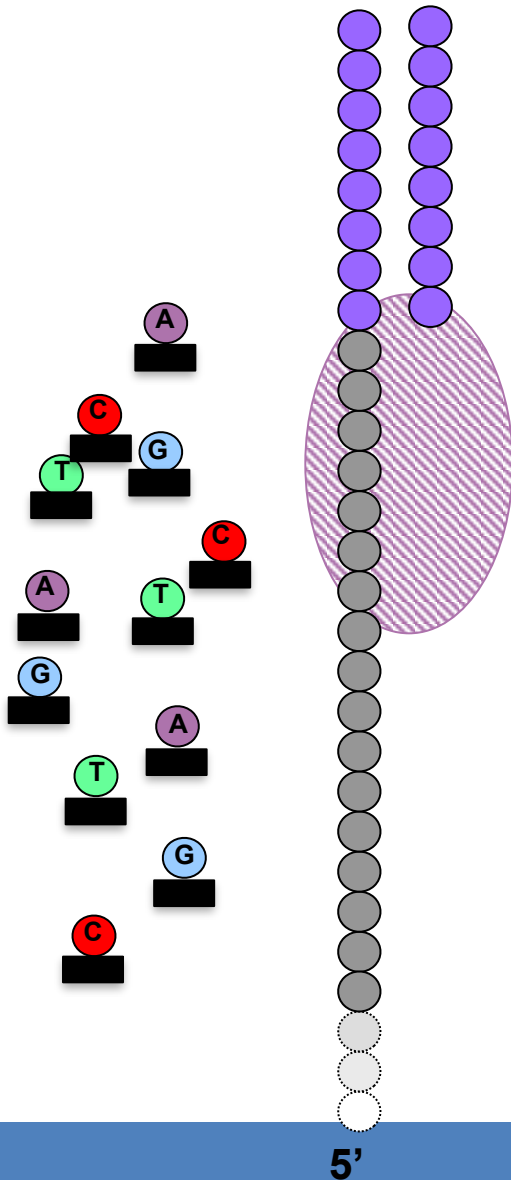
# Sequencing By Synthesis (SBS)



5'

# Sequencing By Synthesis (SBS)

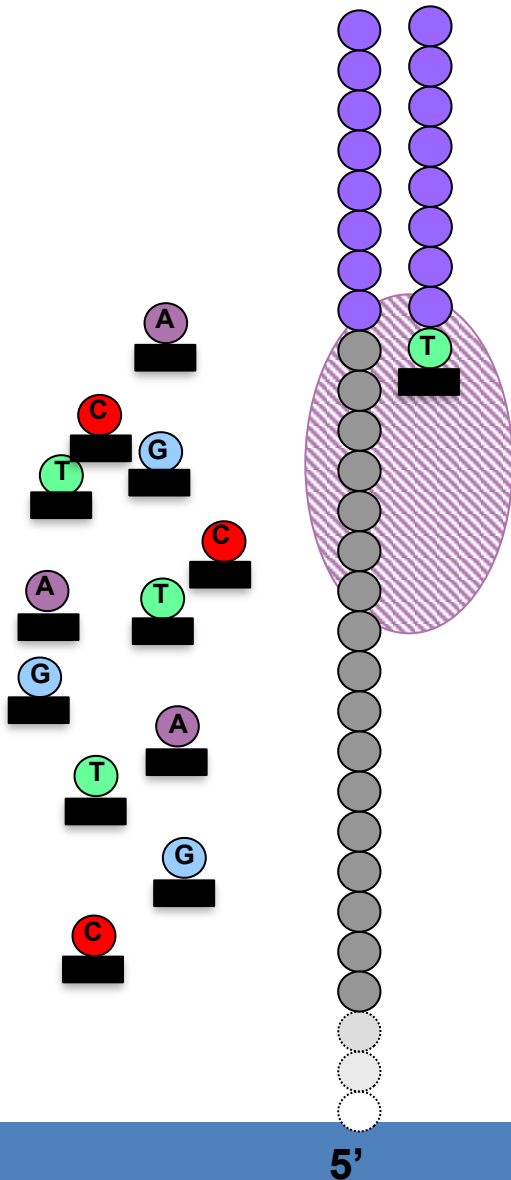
**Cycle 1: Add sequencing reagents (All 4 labeled nucleotides in 1 reaction)**



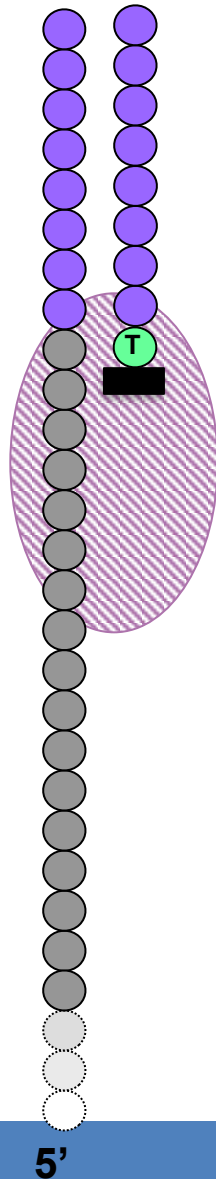
# Sequencing By Synthesis (SBS)

**Cycle 1: Add sequencing reagents (All 4 labeled nucleotides in 1 reaction)**

First base incorporated (reversible dye terminator)



# Sequencing By Synthesis (SBS)

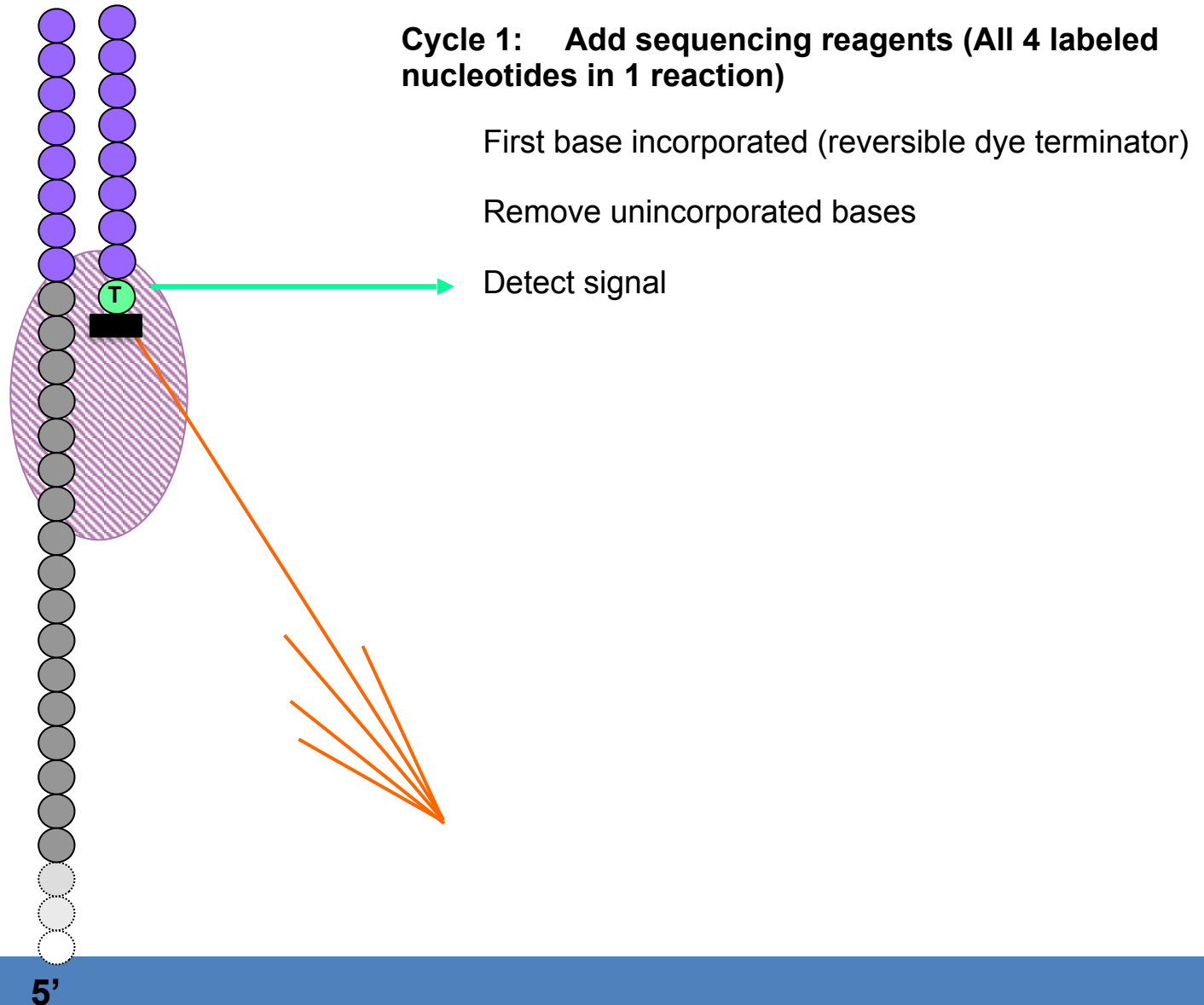


**Cycle 1: Add sequencing reagents (All 4 labeled nucleotides in 1 reaction)**

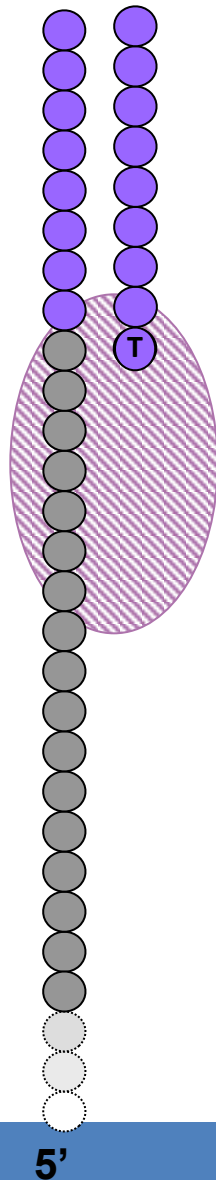
First base incorporated (reversible dye terminator)

Remove unincorporated bases

# Sequencing By Synthesis (SBS)



# Sequencing By Synthesis (SBS)



**Cycle 1: Add sequencing reagents (All 4 labeled nucleotides in 1 reaction)**

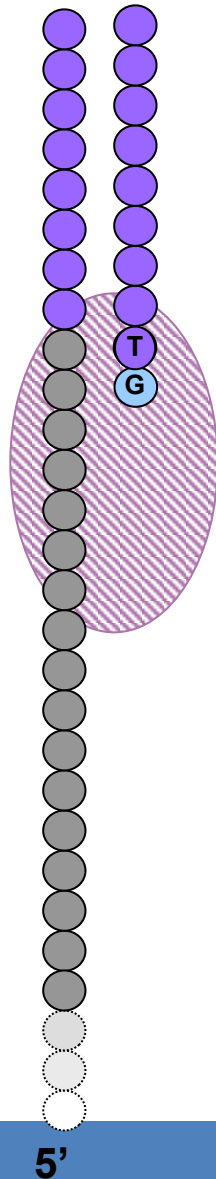
First base incorporated (reversible dye terminator)

Remove unincorporated bases

Detect signal

Unprotect/remove dye

# Sequencing By Synthesis (SBS)



**Cycle 1: Add sequencing reagents (All 4 labeled nucleotides in 1 reaction)**

First base incorporated (reversible dye terminator)

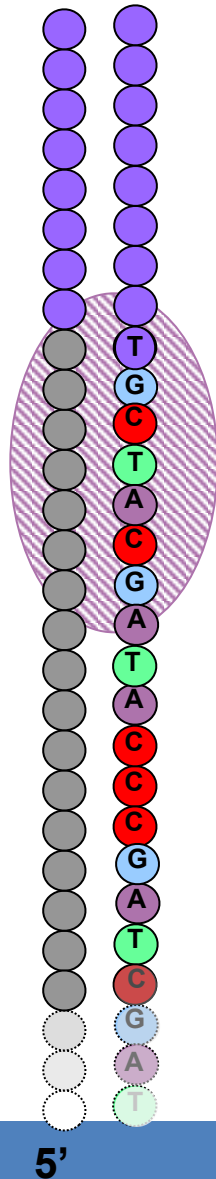
Remove unincorporated bases

Detect signal

Unprotect/remove dye

**Cycle 2-n: Add sequencing reagents and repeat**

# Sequencing By Synthesis (SBS)



**Cycle 1: Add sequencing reagents (All 4 labeled nucleotides in 1 reaction)**

First base incorporated (reversible dye terminator)

Remove unincorporated bases

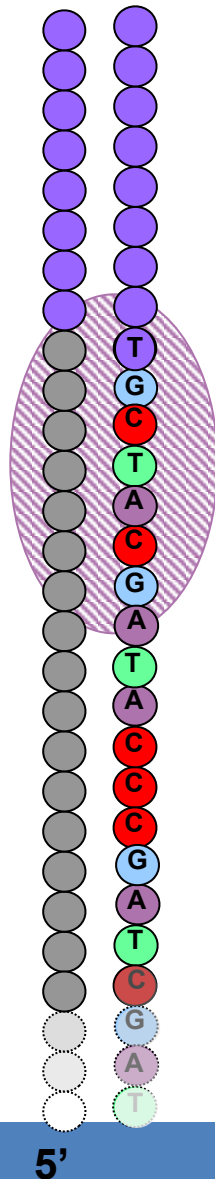
Detect signal

Unprotect/remove dye

**Cycle 2-n: Add sequencing reagents and repeat**



# Sequencing By Synthesis (SBS)



**Cycle 1: Add sequencing reagents (All 4 labeled nucleotides in 1 reaction)**

First base incorporated (reversible dye terminator)

Remove unincorporated bases

Detect signal

Unprotect/remove dye

**Cycle 2-n: Add sequencing reagents and repeat**

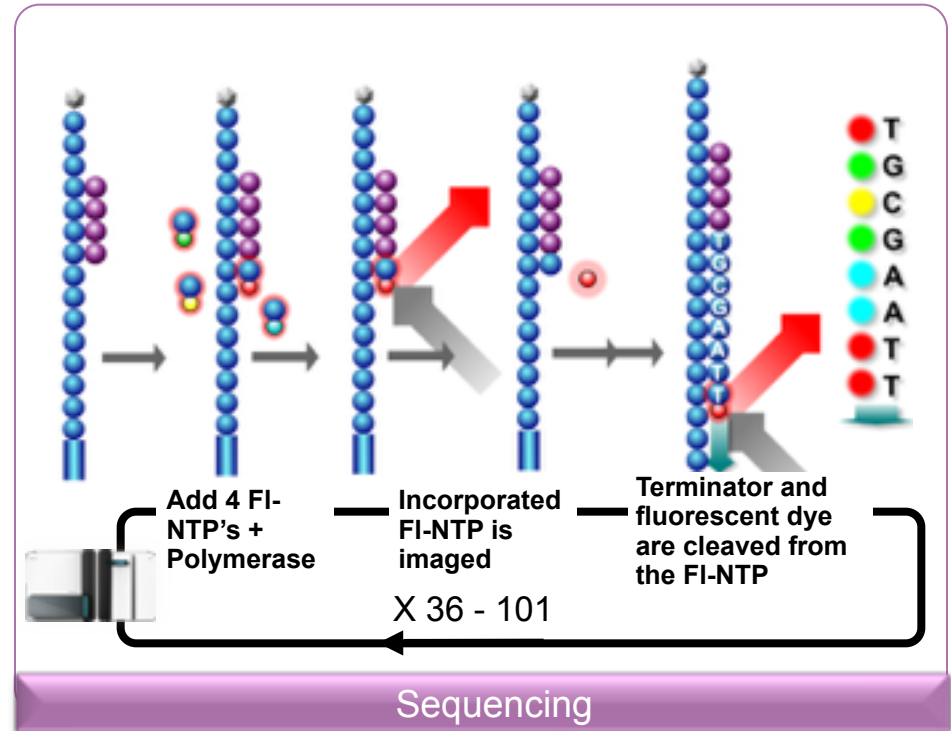
## Key points

- All four labelled nucleotides in one reaction
- Reversible dye terminator
- Base-by-base sequencing
- Real-time sequencing

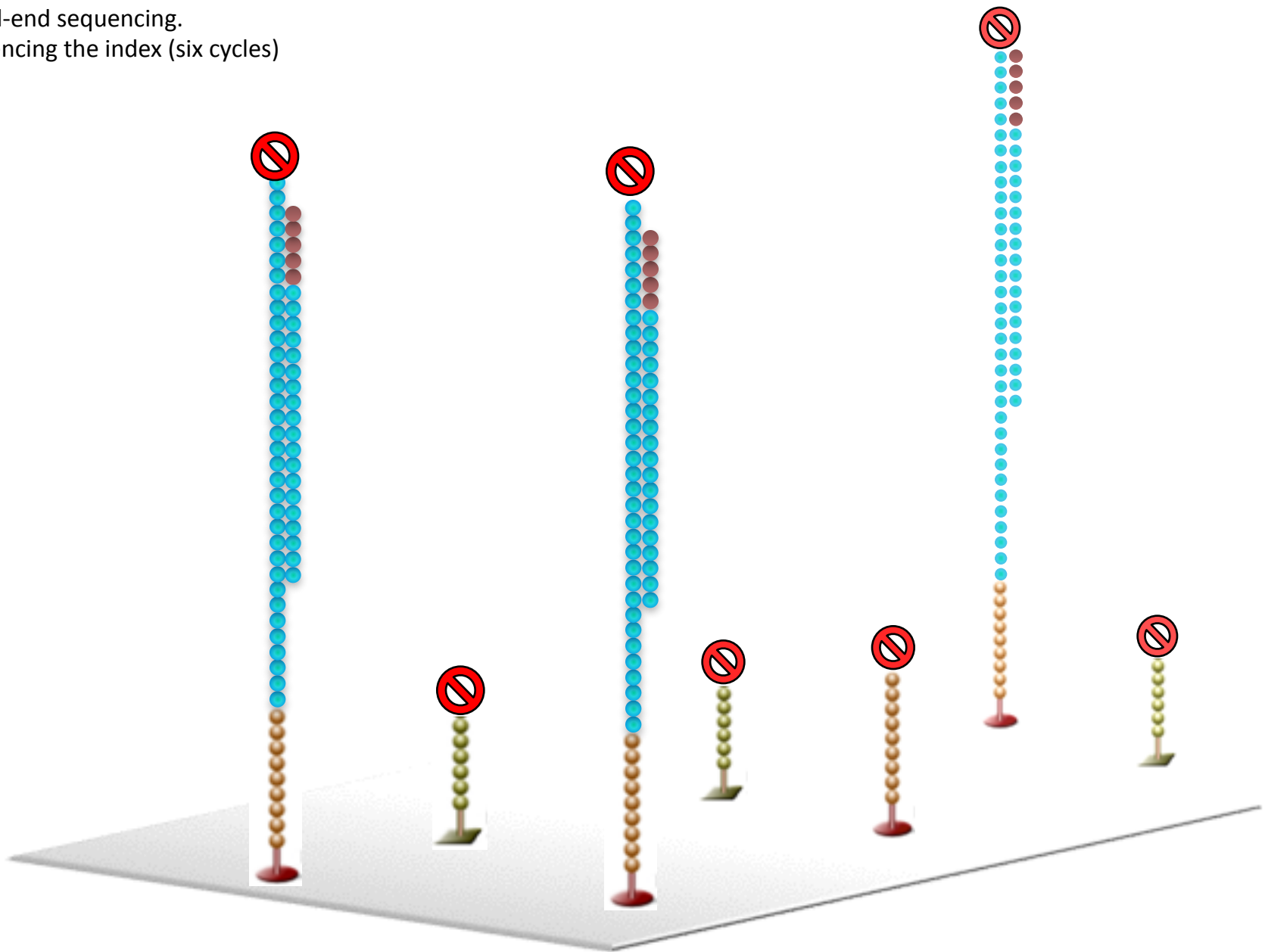
# Sequencing

- ▶ chemistry:
  - All 4 labeled nucleotides in 1 reaction
  - Reversible dye terminators

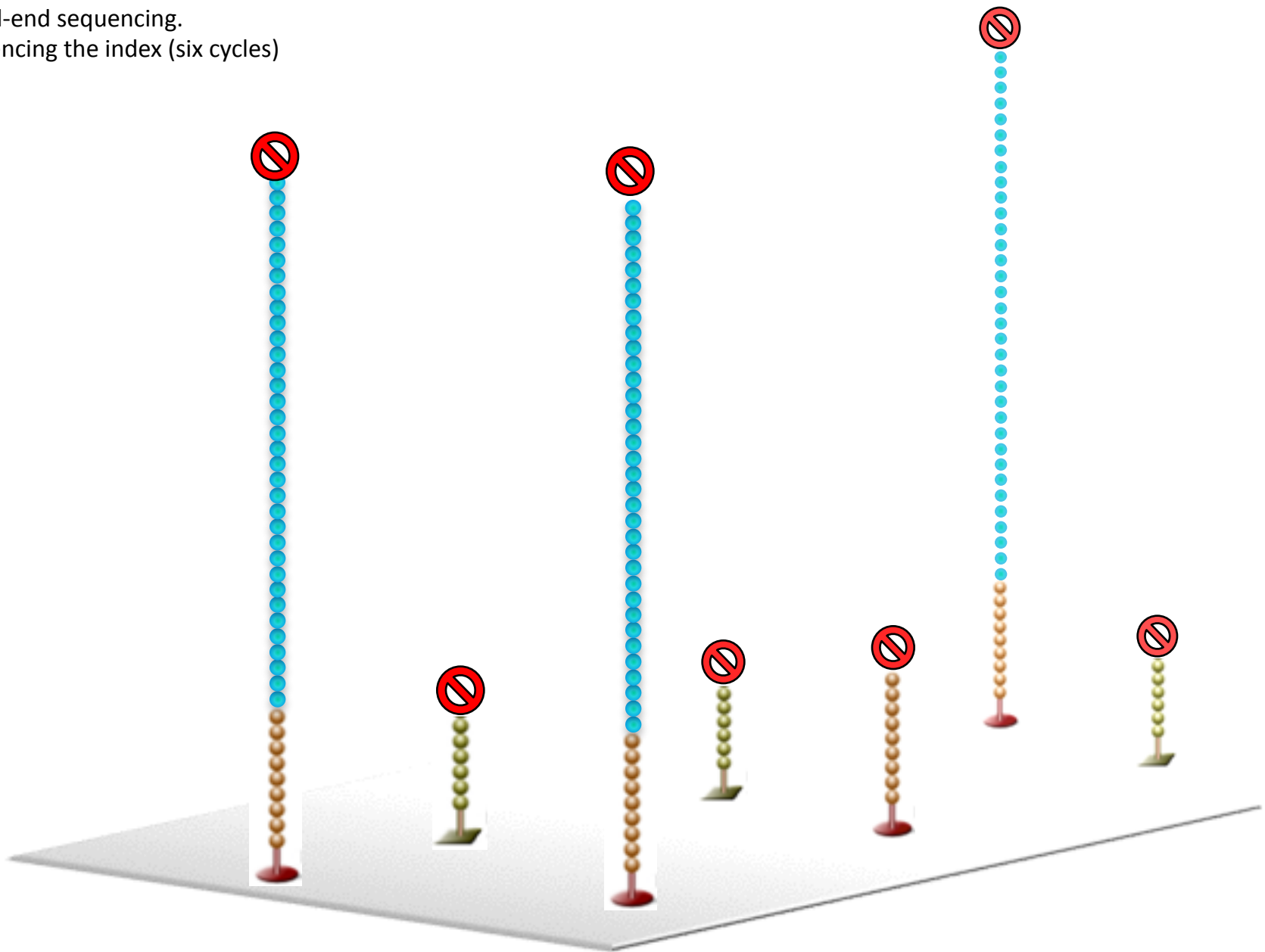
- ▶ 3-step cycles:
  - Incorporate fluorescent nucleotide
  - Image tiles
  - Cleave terminator and fluor



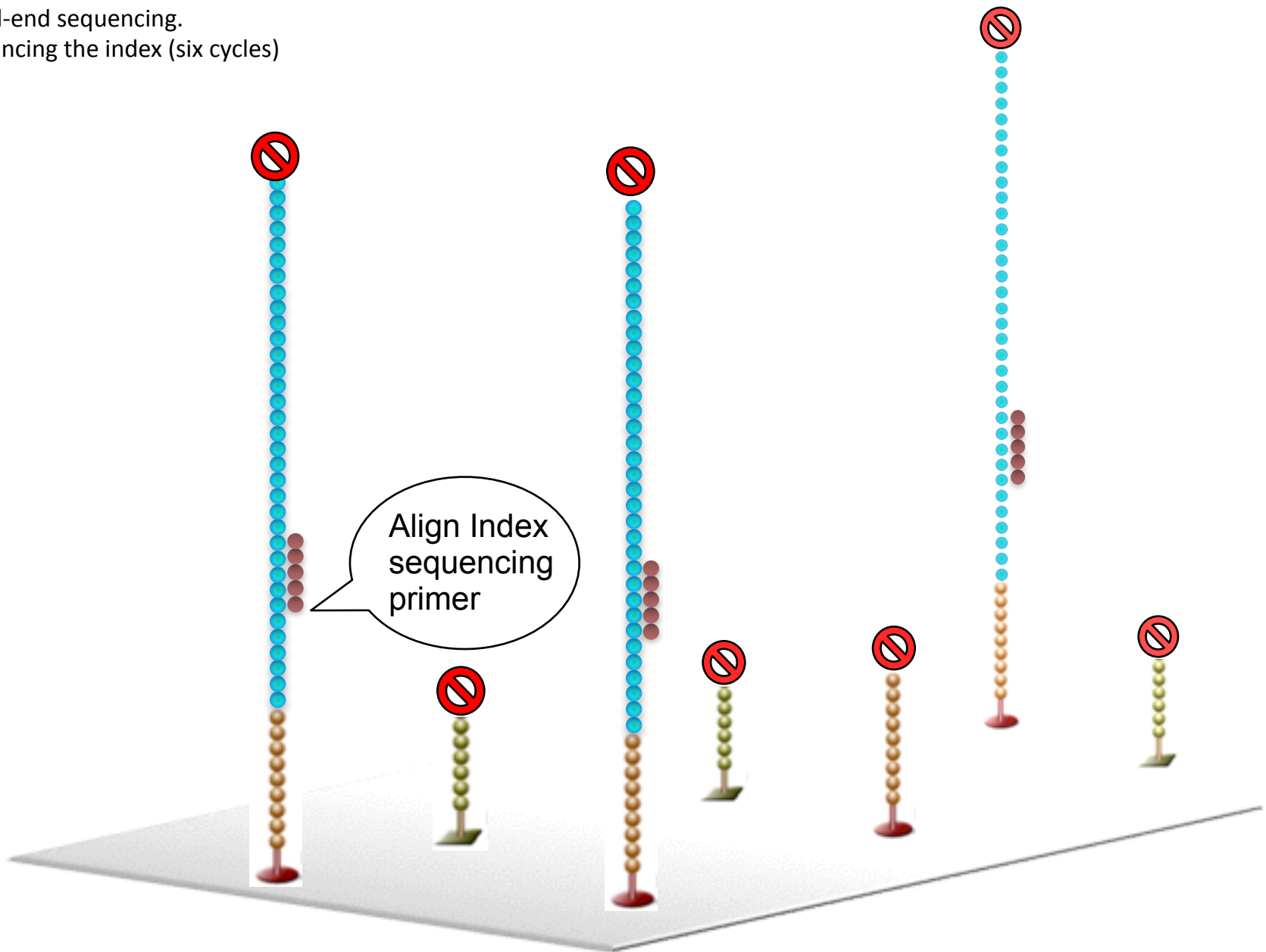
Paired-end sequencing.  
Sequencing the index (six cycles)



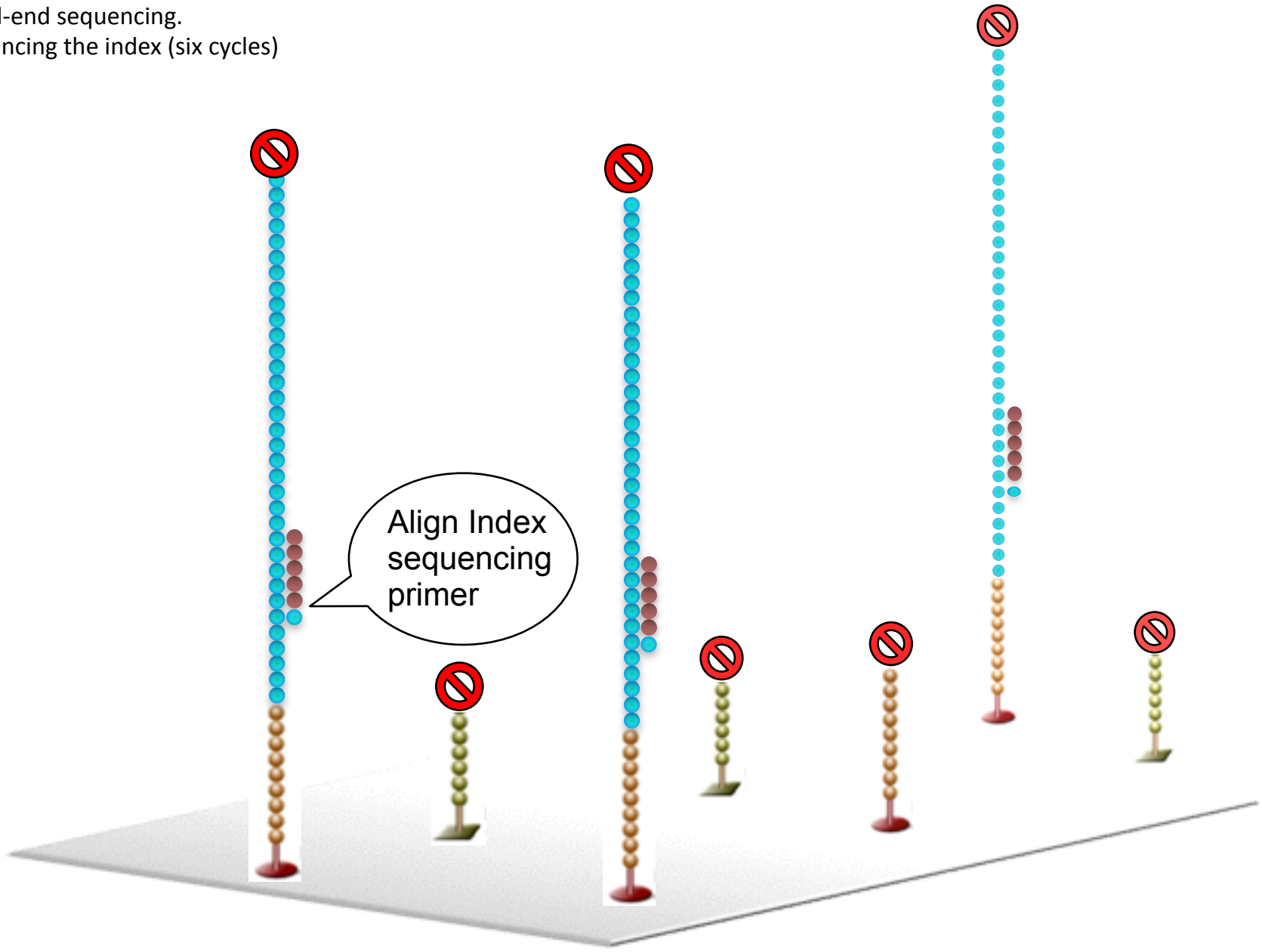
Paired-end sequencing.  
Sequencing the index (six cycles)



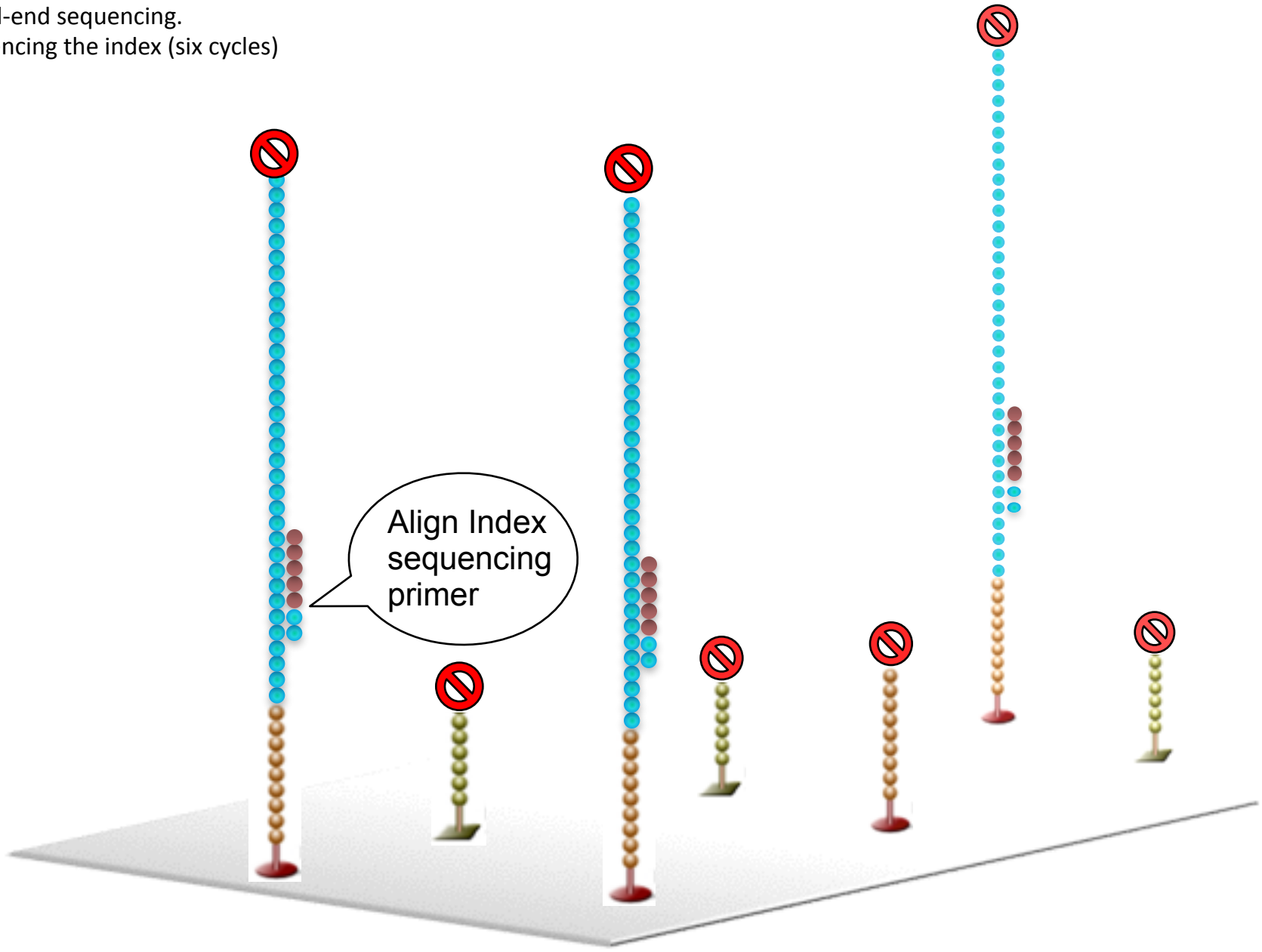
Paired-end sequencing.  
Sequencing the index (six cycles)



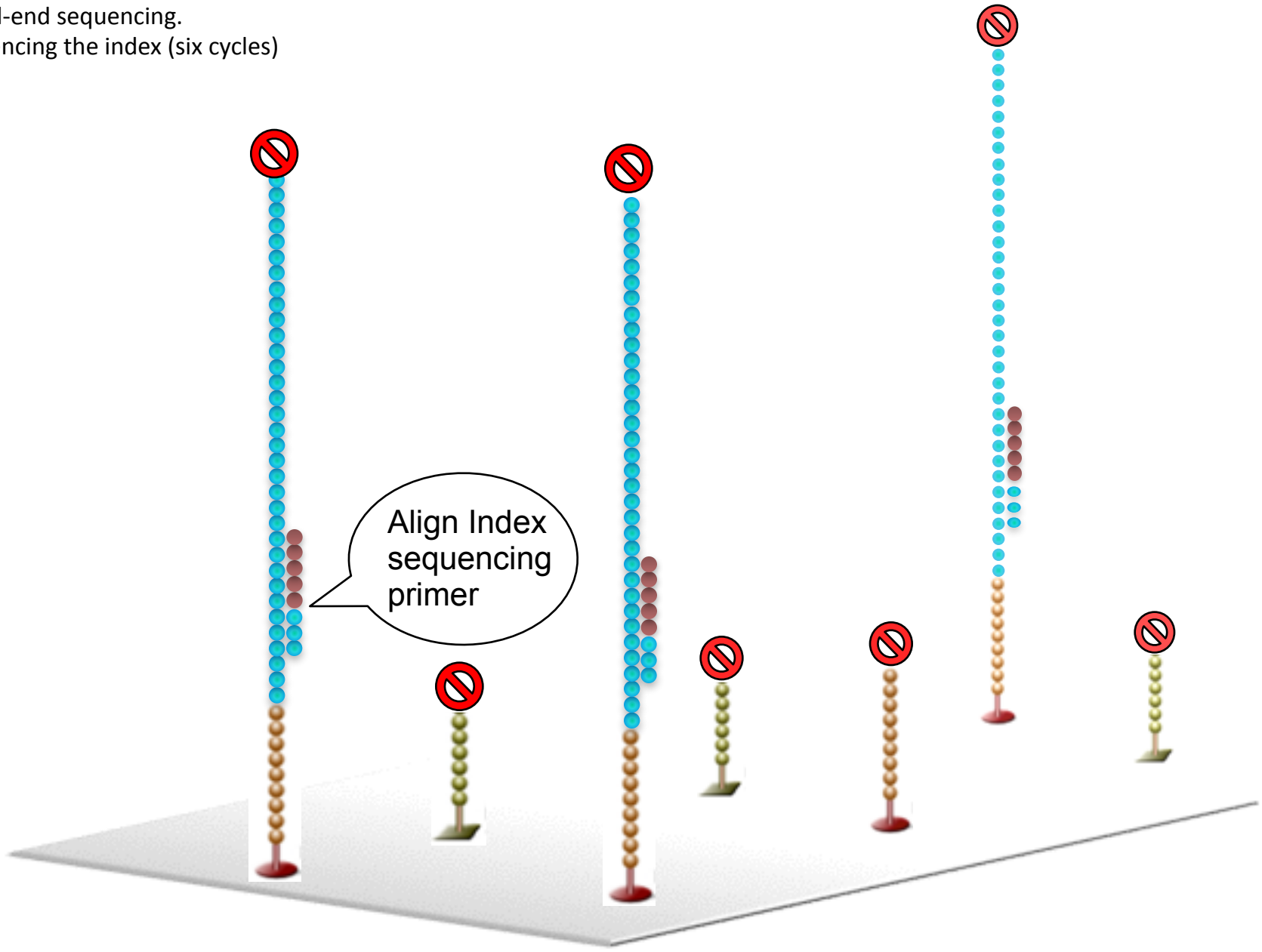
Paired-end sequencing.  
Sequencing the index (six cycles)



Paired-end sequencing.  
Sequencing the index (six cycles)

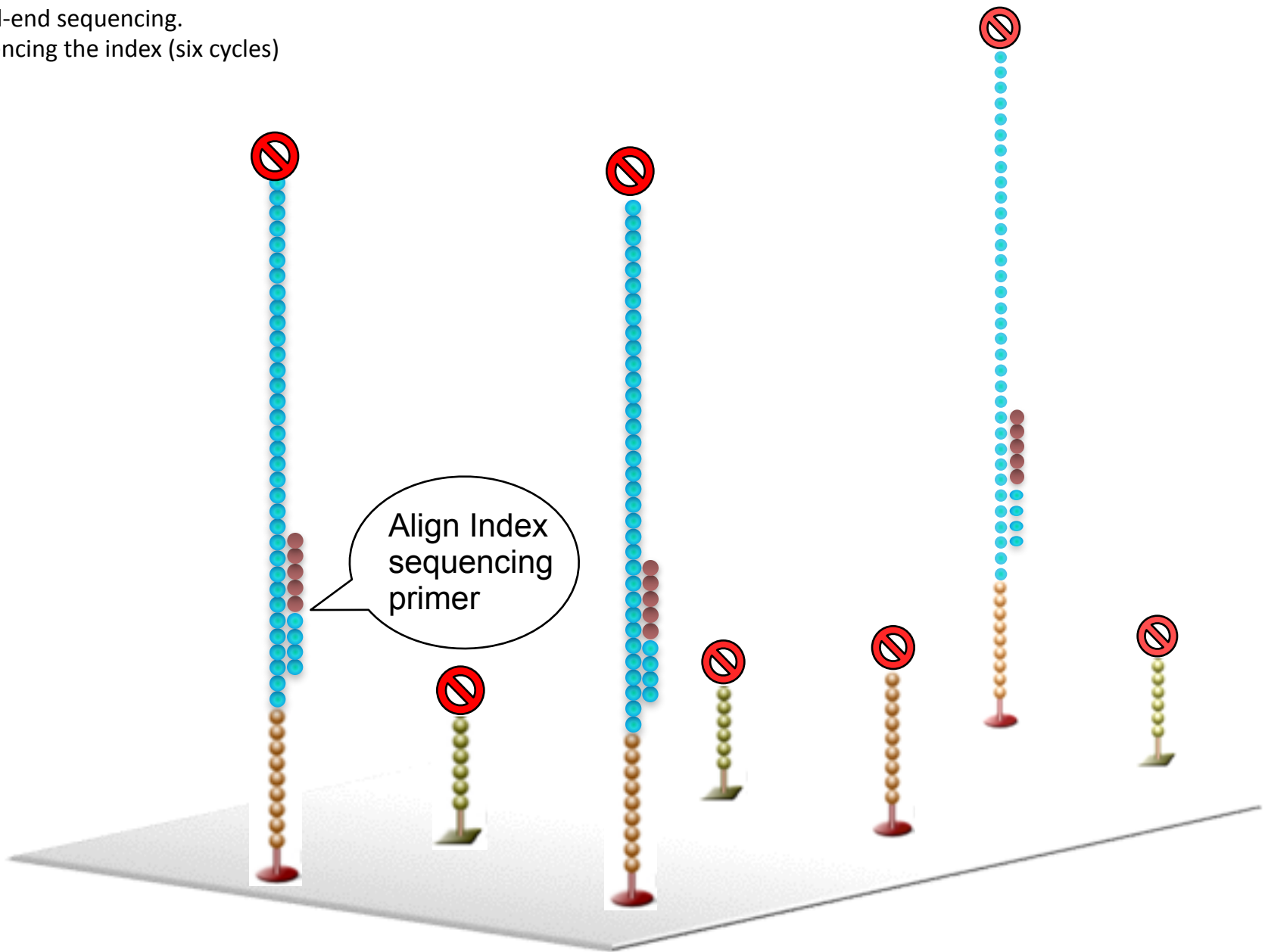


Paired-end sequencing.  
Sequencing the index (six cycles)

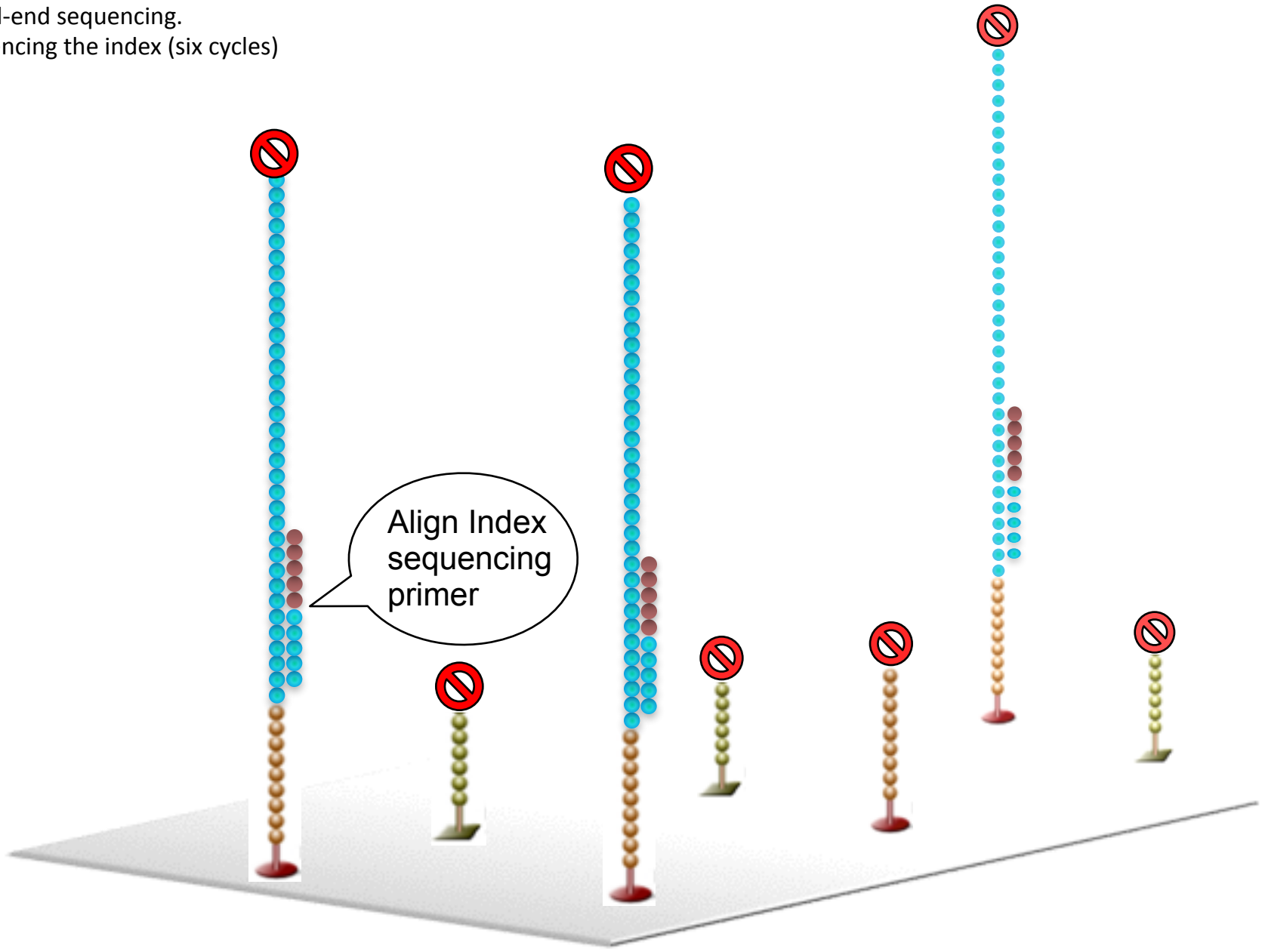




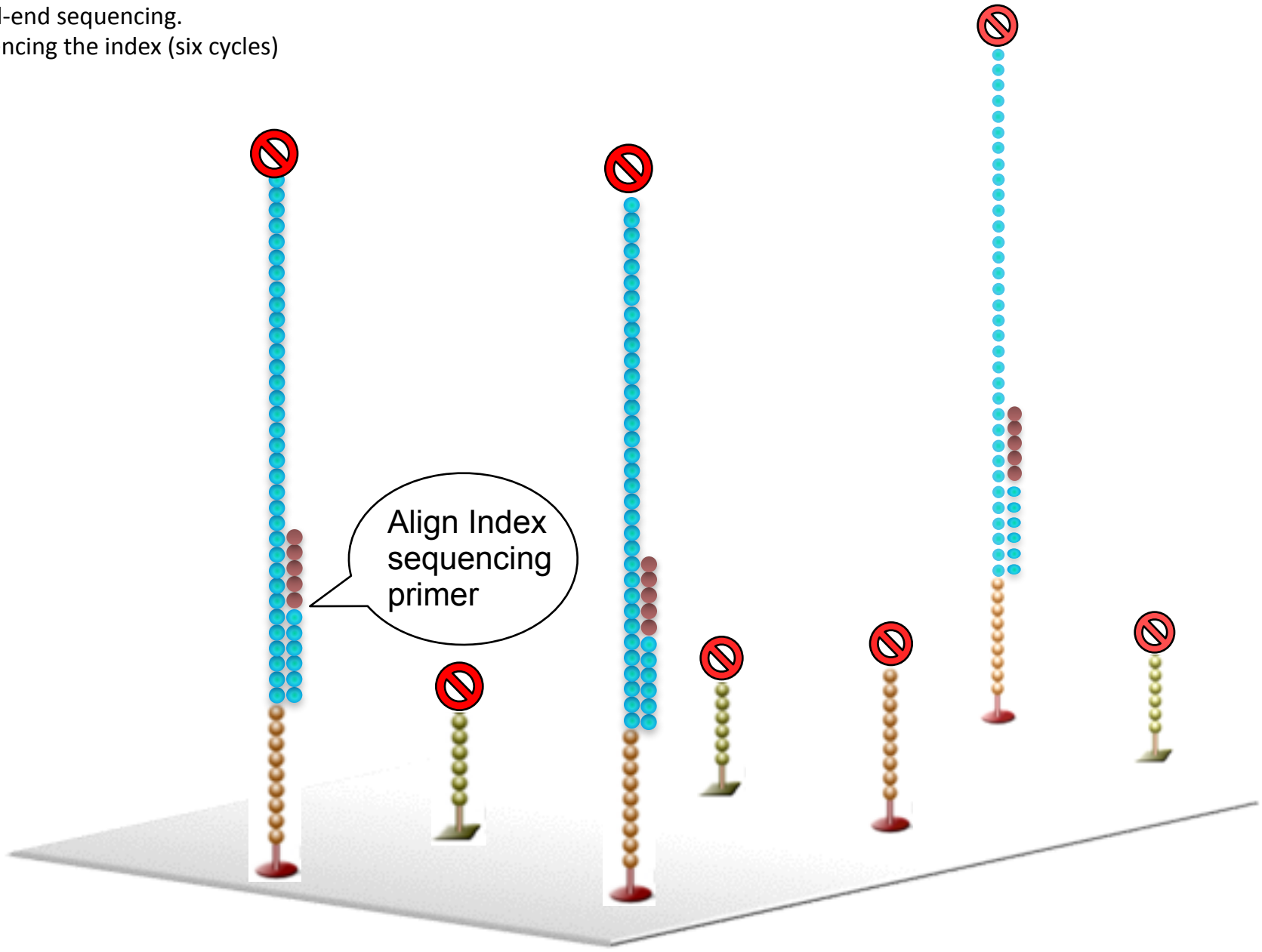
Paired-end sequencing.  
Sequencing the index (six cycles)



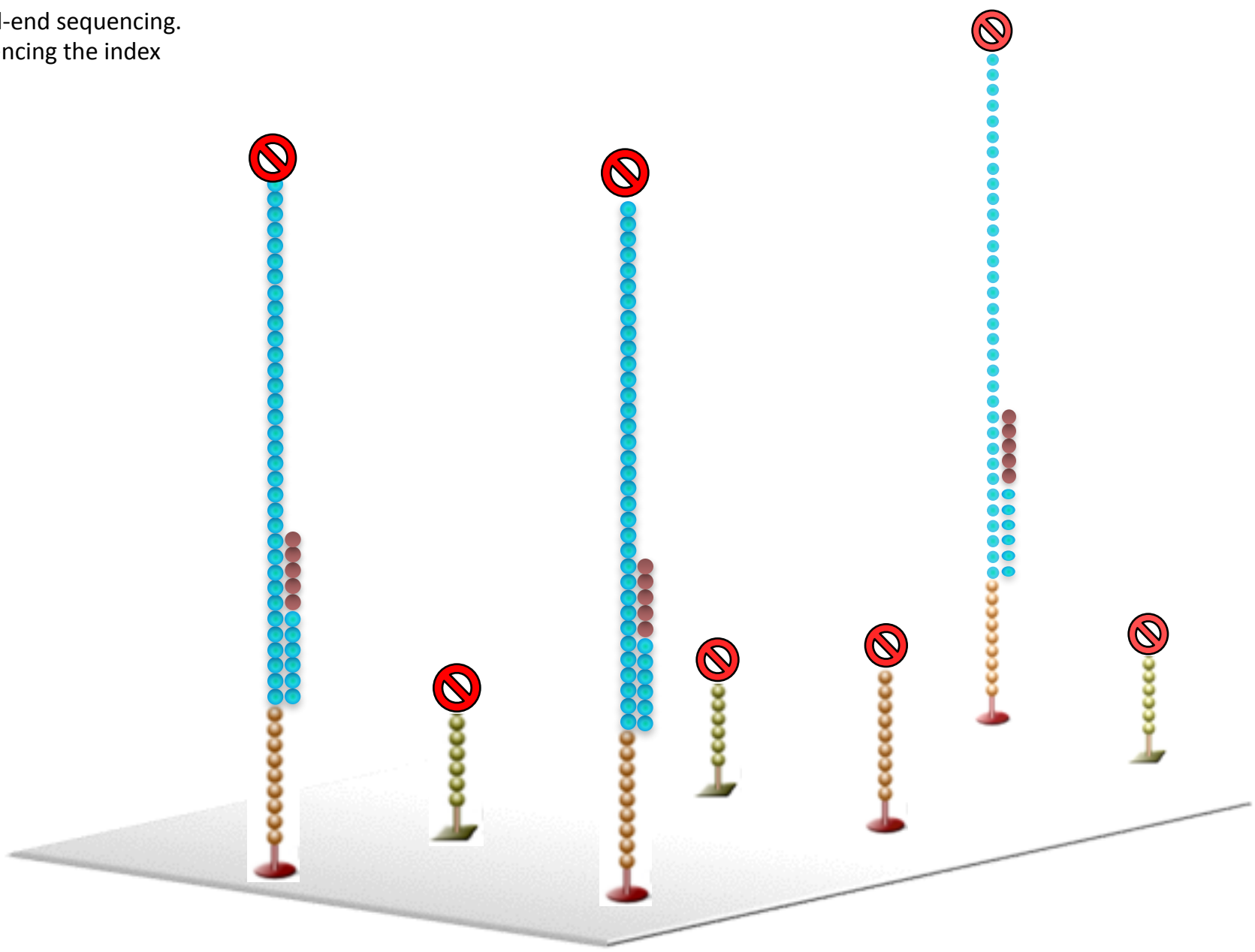
Paired-end sequencing.  
Sequencing the index (six cycles)



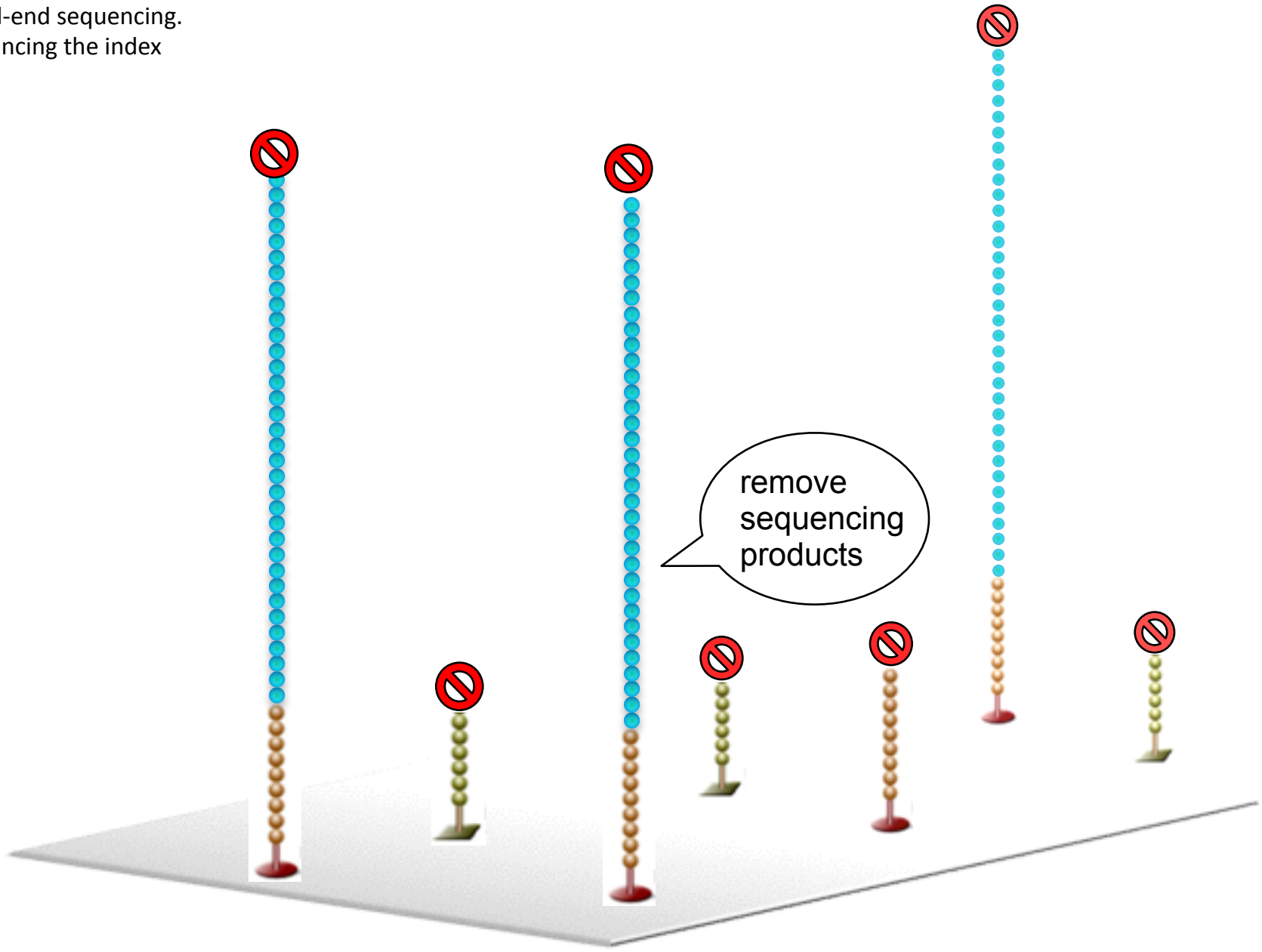
Paired-end sequencing.  
Sequencing the index (six cycles)



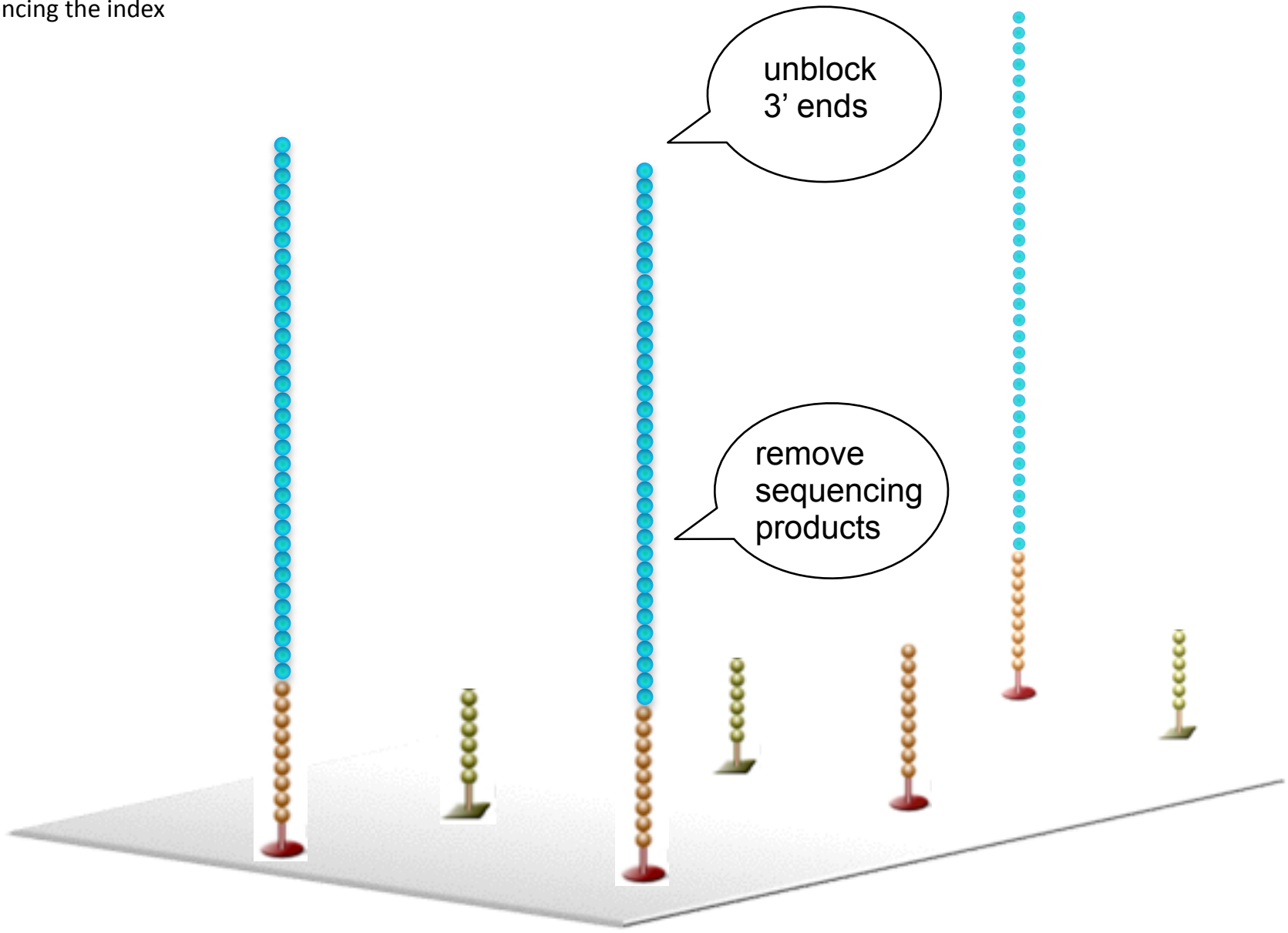
Paired-end sequencing.  
Sequencing the index



Paired-end sequencing.  
Sequencing the index

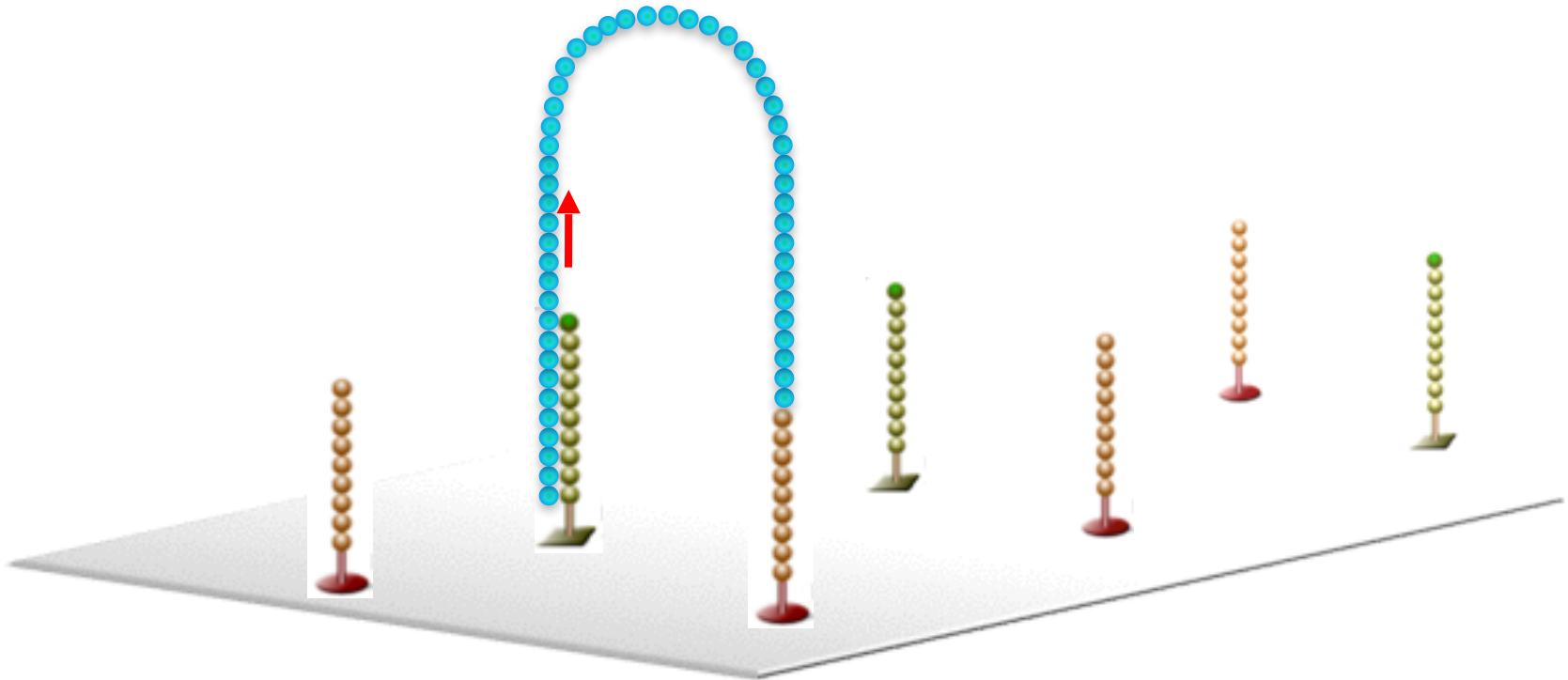


Paired-end sequencing.  
Sequencing the index



# Paired-end sequencing, re-synthesis of 2<sup>nd</sup> strand

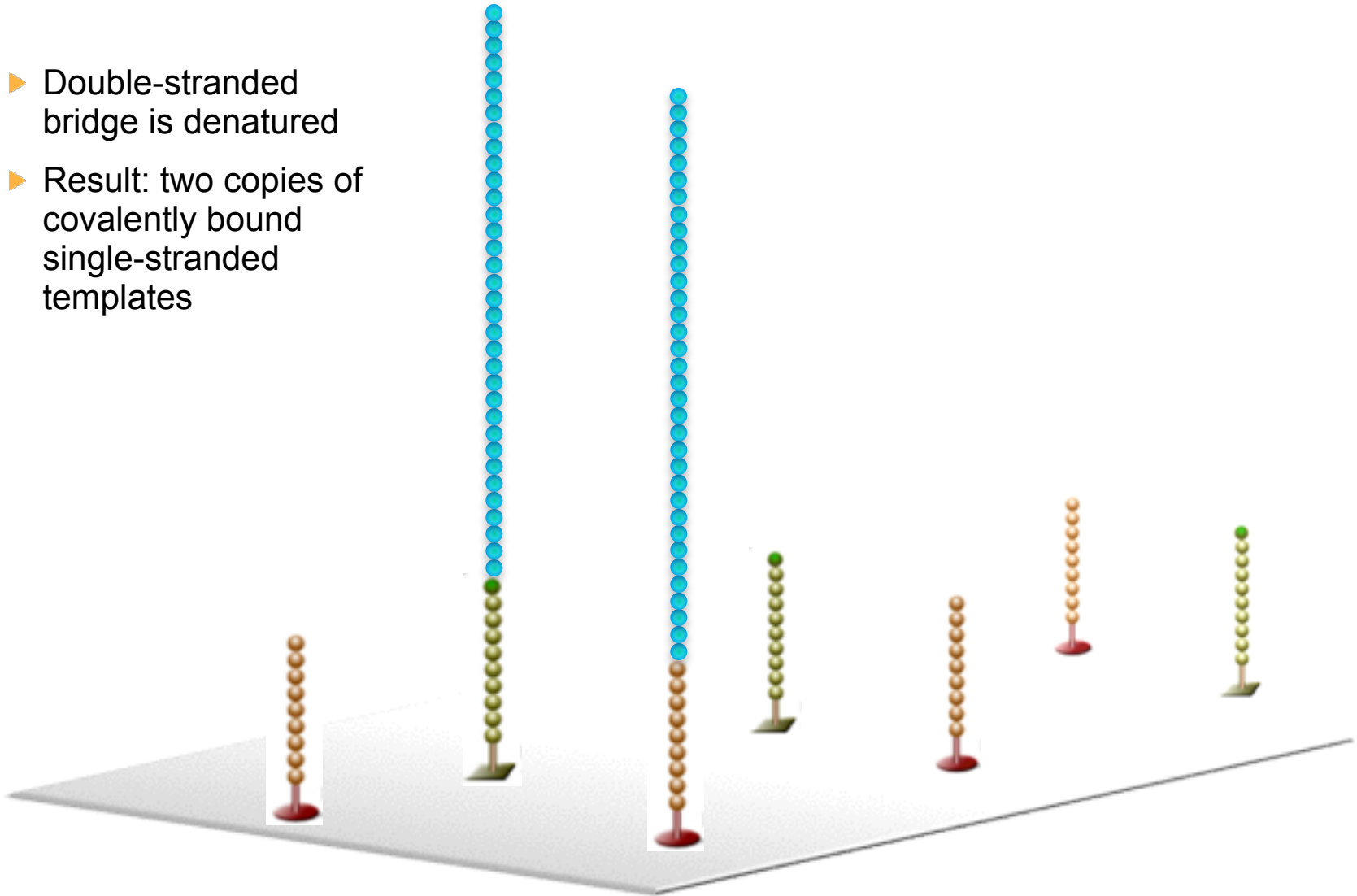
- ▶ Bridge formation and 3' extension



\*Note, that this phrase is typically used in the context of cDNA synthesis!

# Denaturation

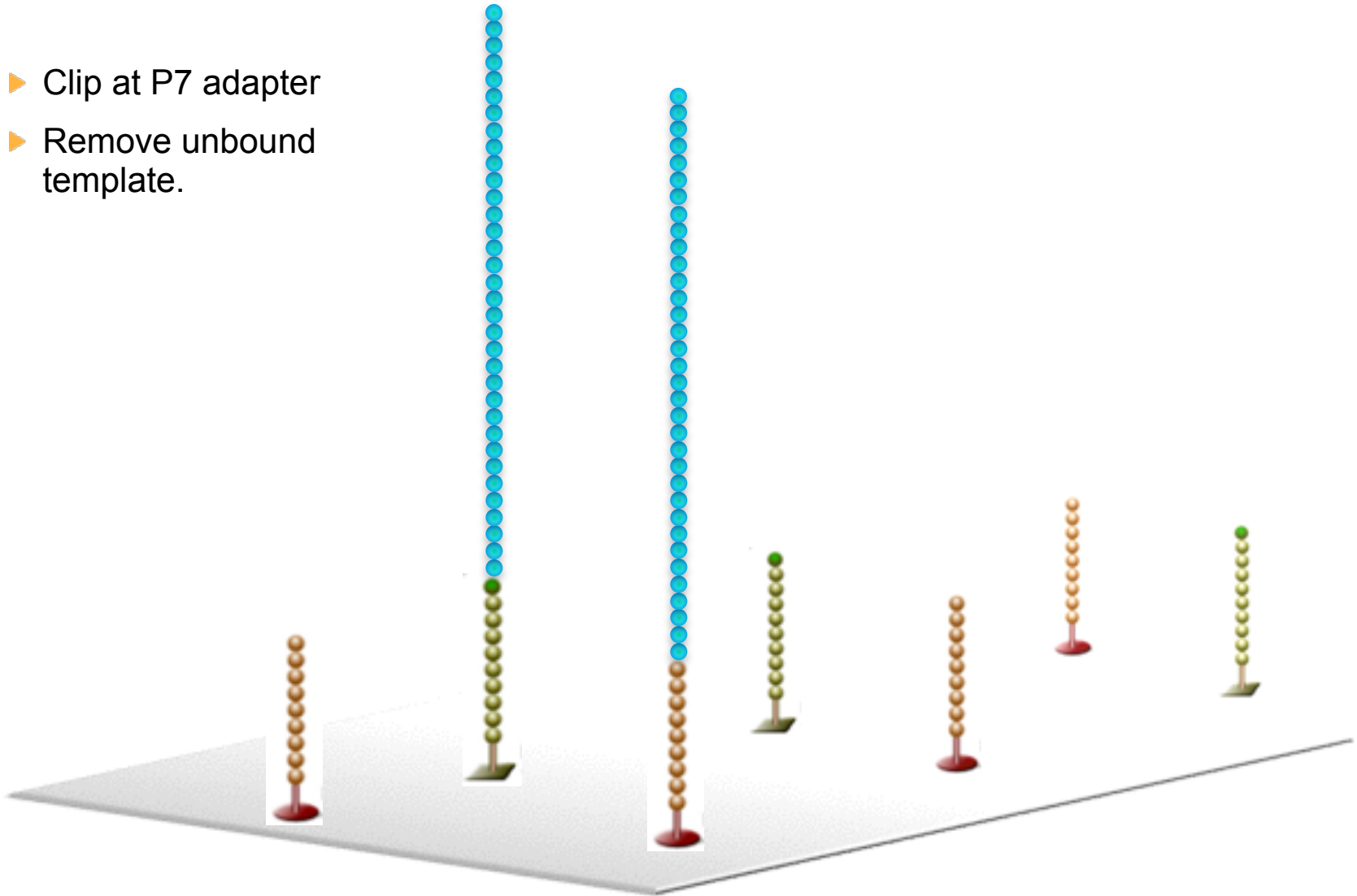
- ▶ Double-stranded bridge is denatured
- ▶ Result: two copies of covalently bound single-stranded templates





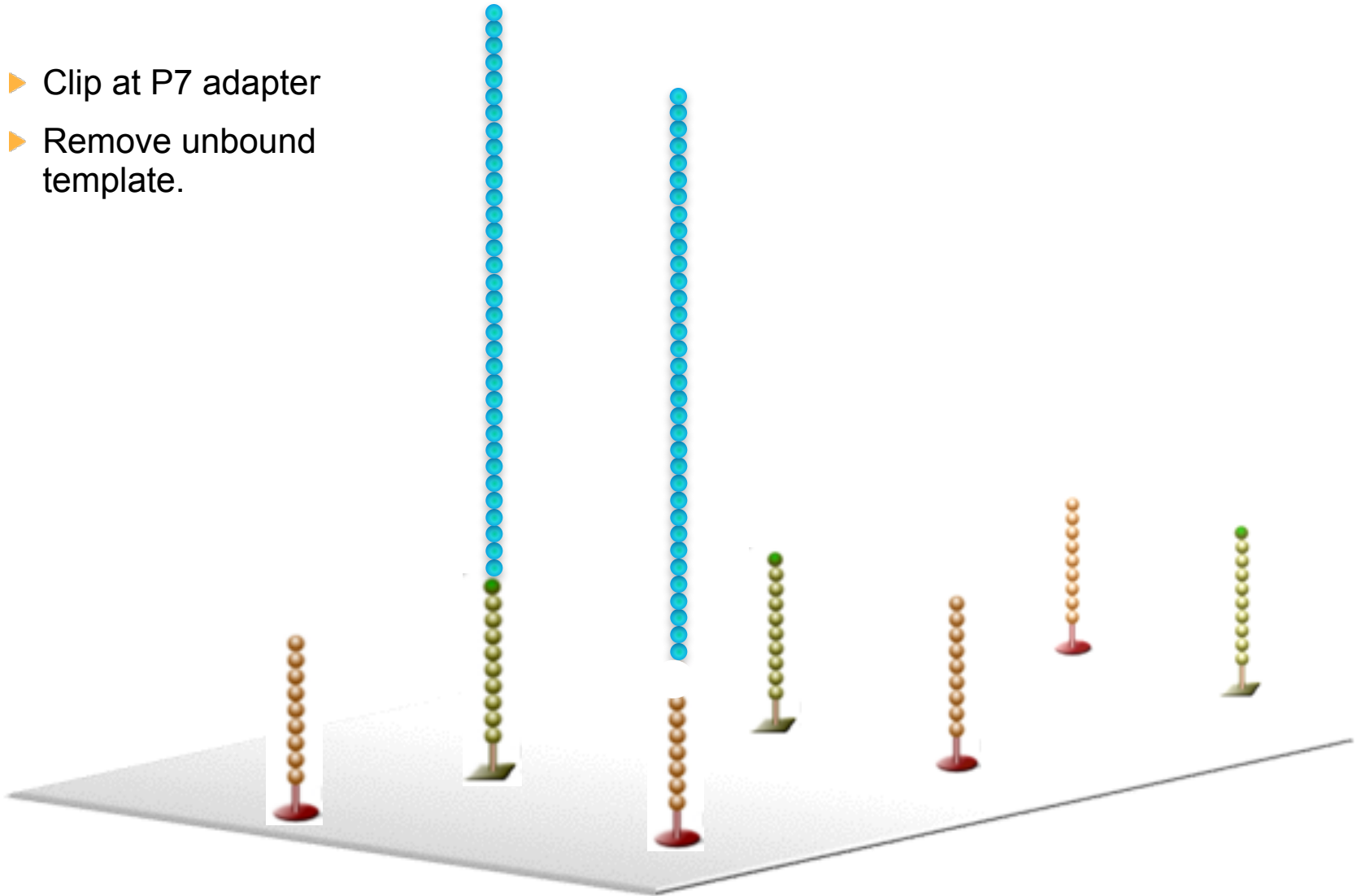
# Cleavage and removal of first strand

- ▶ Clip at P7 adapter
- ▶ Remove unbound template.



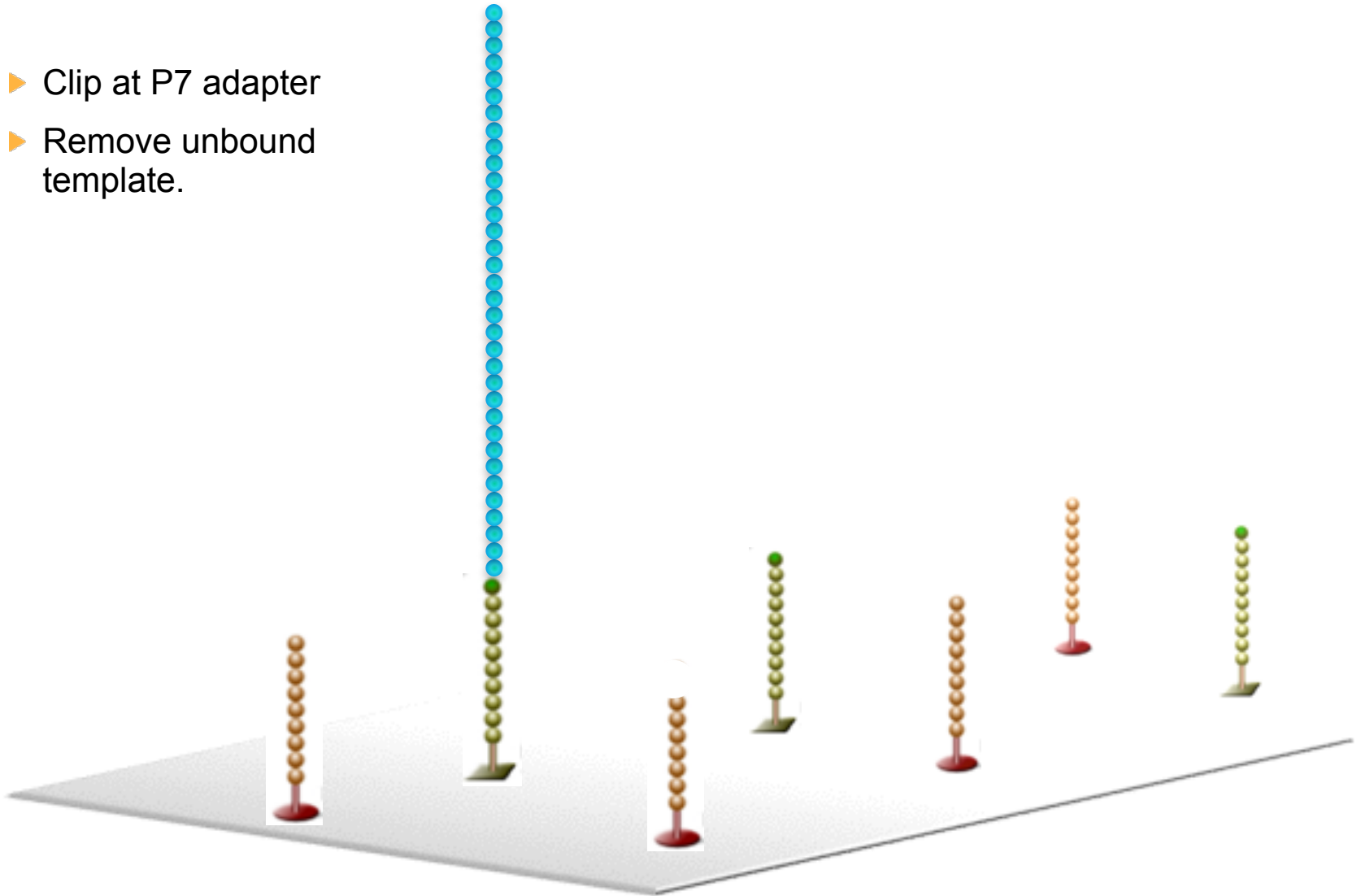
# Cleavage and removal of first strand

- ▶ Clip at P7 adapter
- ▶ Remove unbound template.



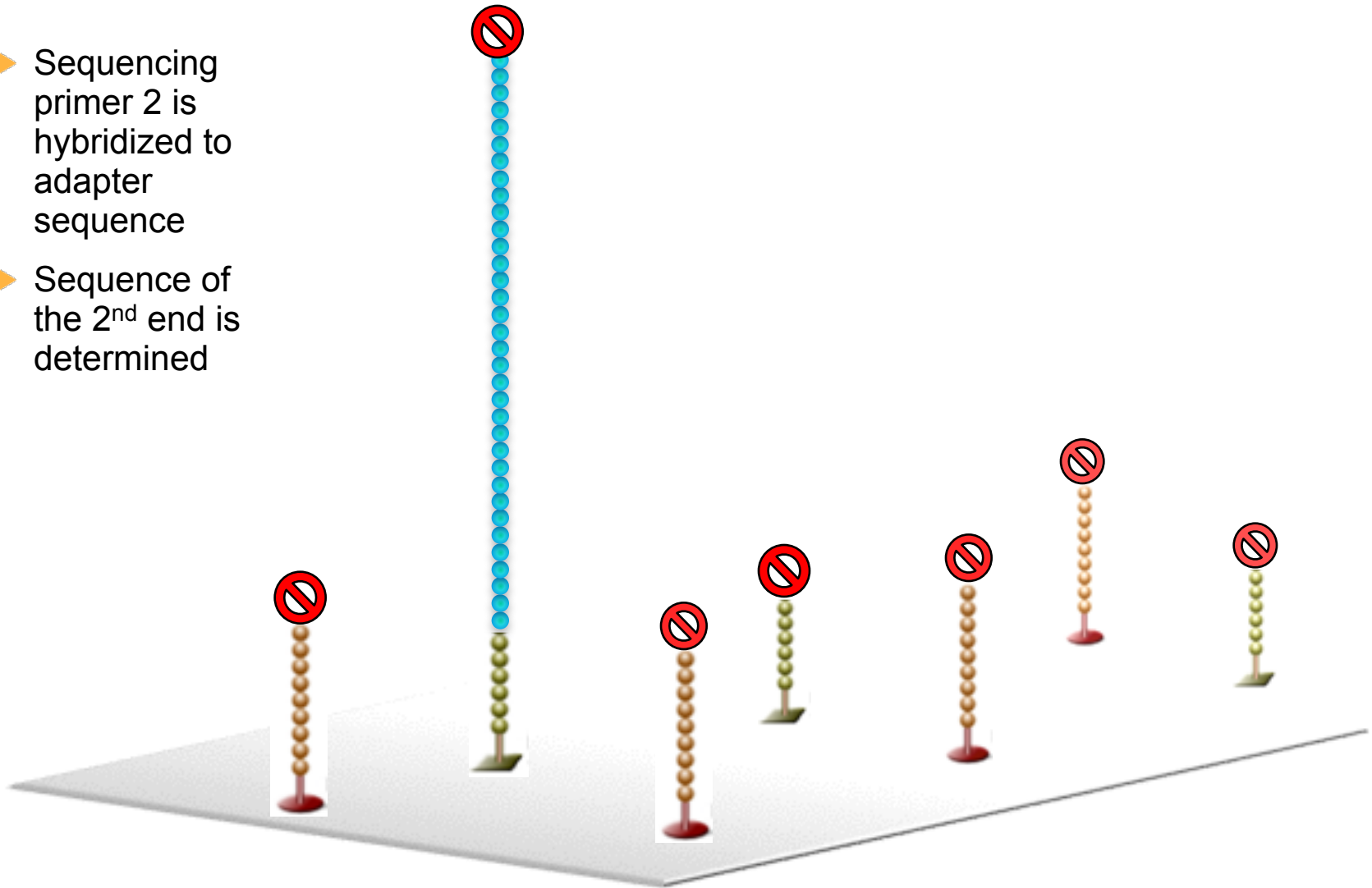
# Cleavage and removal of first strand

- ▶ Clip at P7 adapter
- ▶ Remove unbound template.



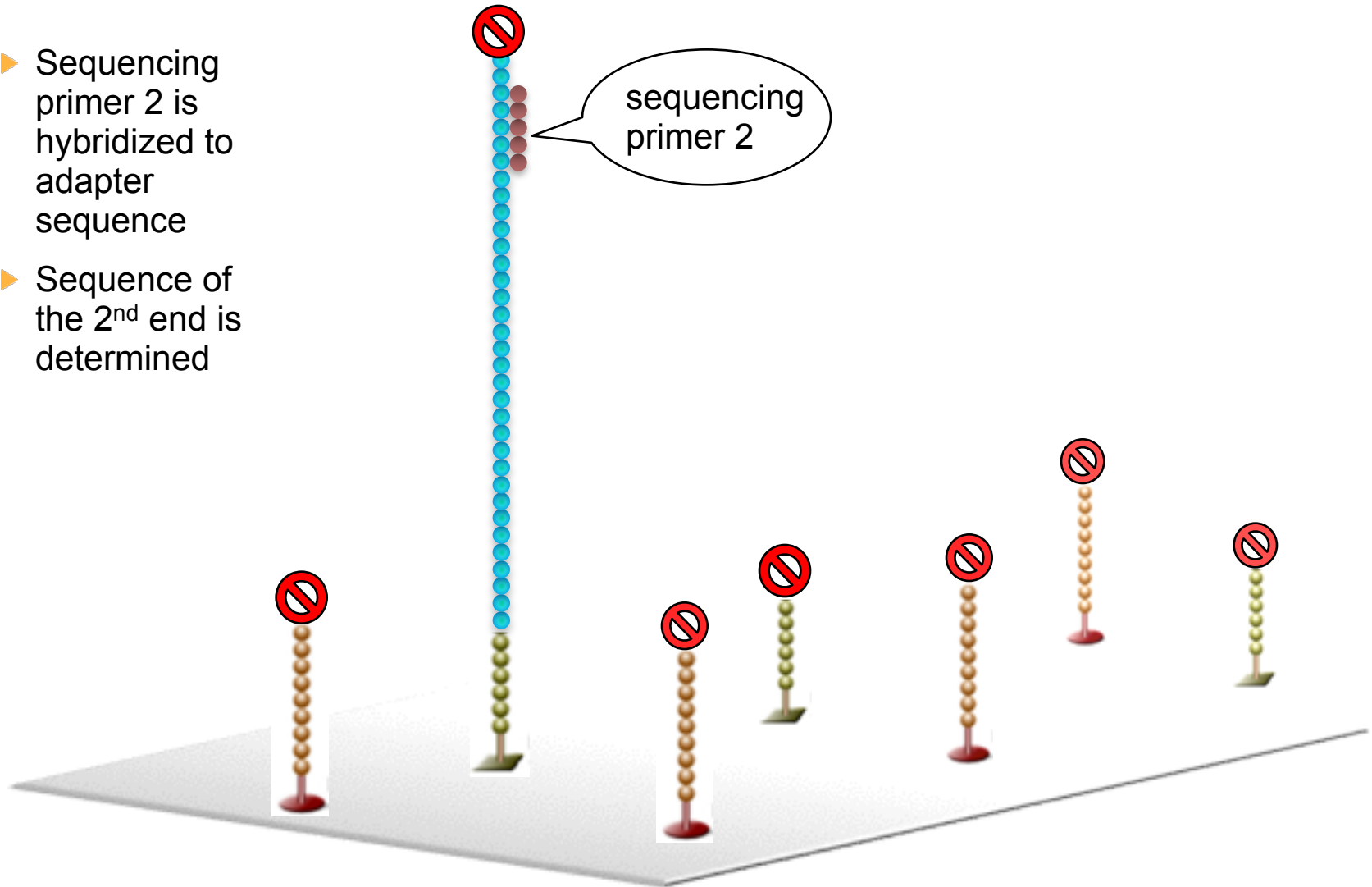
# Hybridization of sequencing primer 2

- ▶ Sequencing primer 2 is hybridized to adapter sequence
- ▶ Sequence of the 2<sup>nd</sup> end is determined



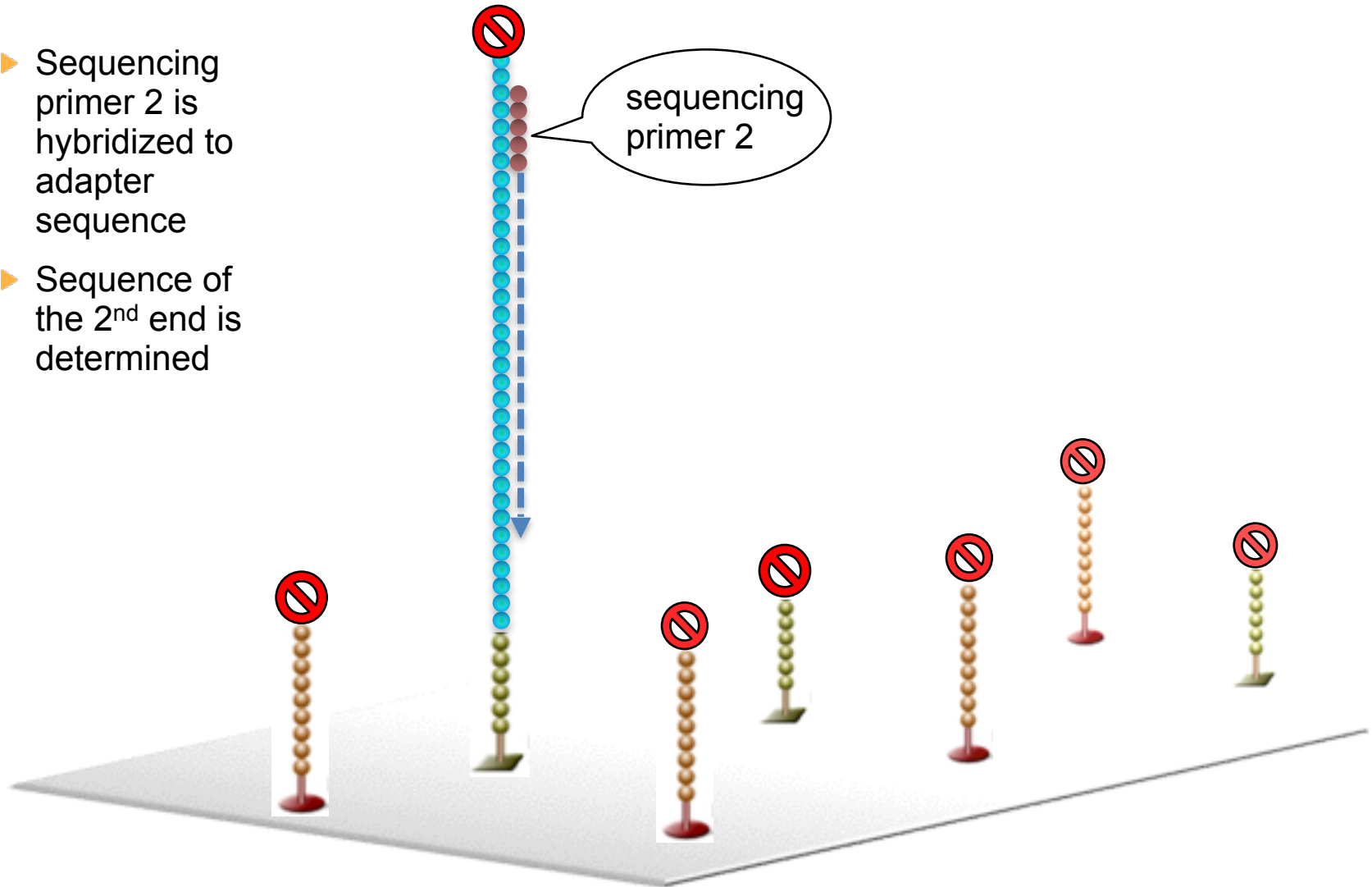
# Hybridization of sequencing primer 2

- ▶ Sequencing primer 2 is hybridized to adapter sequence
- ▶ Sequence of the 2<sup>nd</sup> end is determined

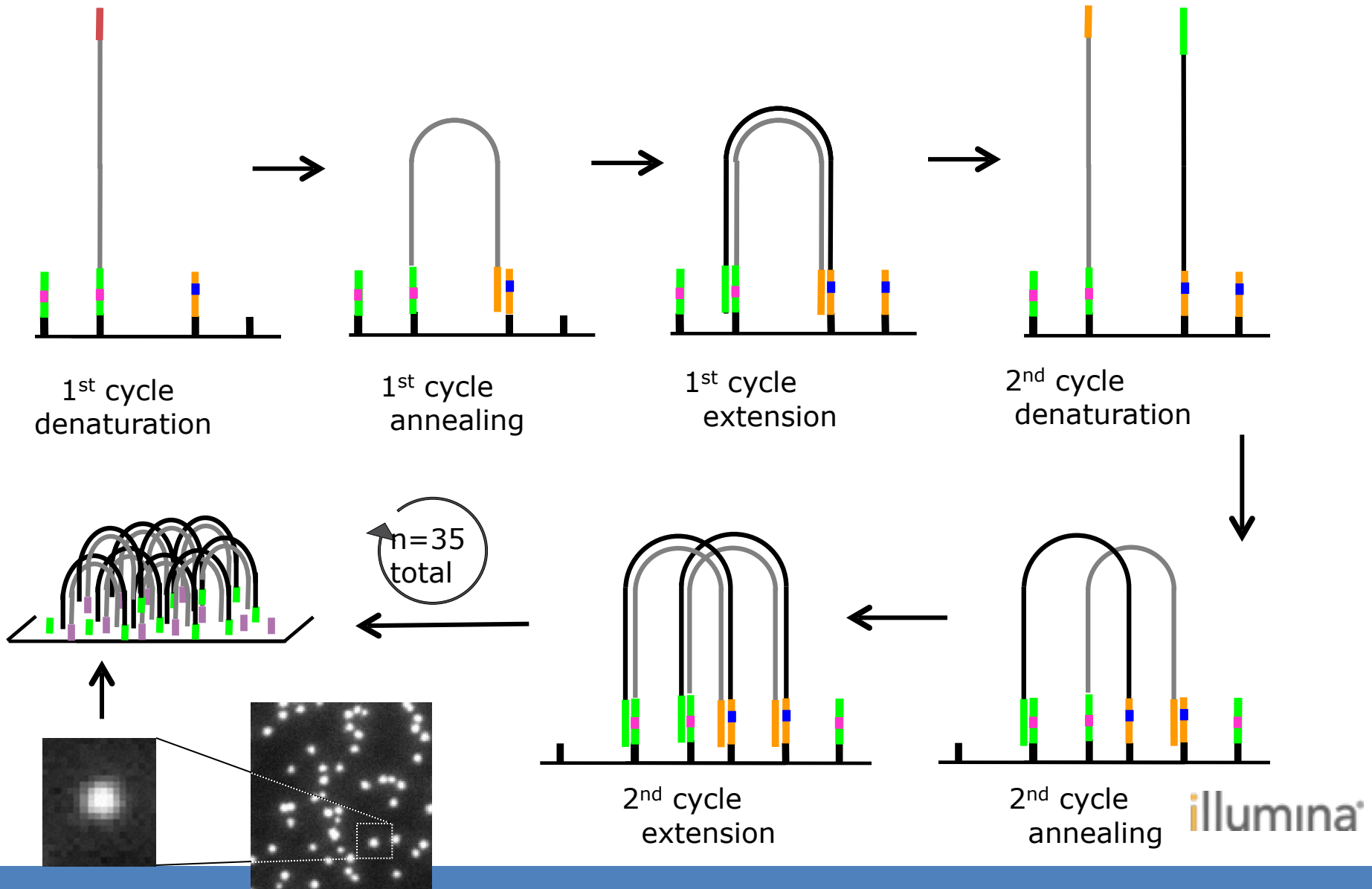


# Hybridization of sequencing primer 2

- ▶ Sequencing primer 2 is hybridized to adapter sequence
- ▶ Sequence of the 2<sup>nd</sup> end is determined



# Cluster Generation: Amplification



The methods described so far average the signal over millions of copies of the same sequence. Why is this problematic?



The methods described so far average the signal over millions of copies of the same sequence. Why is this problematic?

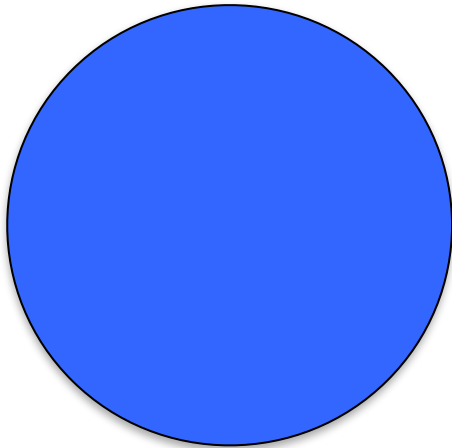
- Errors during PCR amplification render copies not 100% identical. Especially errors at an early stage of the PCR can mimic heterozygous positions.

The methods described so far average the signal over millions of copies of the same sequence. Why is this problematic?

- Errors during PCR amplification render copies not 100% identical. Especially errors at an early stage of the PCR can mimic heterozygous positions.
- Not every copy of a pool of millions of sequences will incorporate a base in each cycle. With increasing numbers of cycles the length heterogeneity of the already sequenced fraction will increase and the sequencing will get **out of phase**.

The methods described so far average the signal over millions of copies of the same sequence. Why is this problematic?

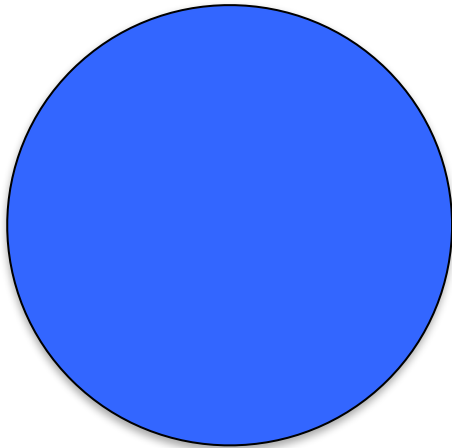
- Errors during PCR amplification render copies not 100% identical. Especially errors at an early stage of the PCR can mimic heterozygous positions.
- Not every copy of a pool of millions of sequences will incorporate a base in each cycle. With increasing numbers of cycles the length heterogeneity of the already sequenced fraction will increase and the sequencing will get **out of phase**.



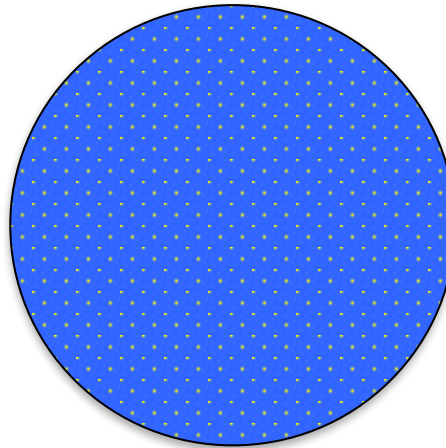
Template: AGACTATTTA  
TCT

The methods described so far average the signal over millions of copies of the same sequence. Why is this problematic?

- Errors during PCR amplification render copies not 100% identical. Especially errors at an early stage of the PCR can mimic heterozygous positions.
- Not every copy of a pool of millions of sequences will incorporate a base in each cycle. With increasing numbers of cycles the length heterogeneity of the already sequenced fraction will increase and the sequencing will get **out of phase**.



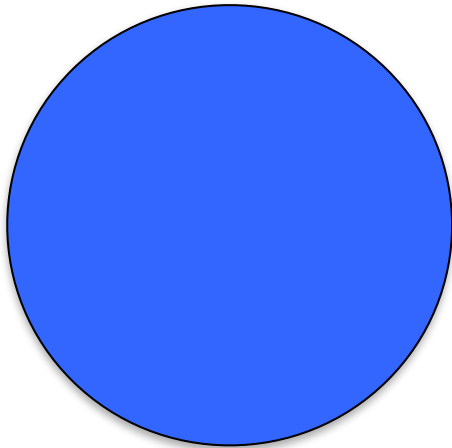
Template: AGACTATTTA  
TCT



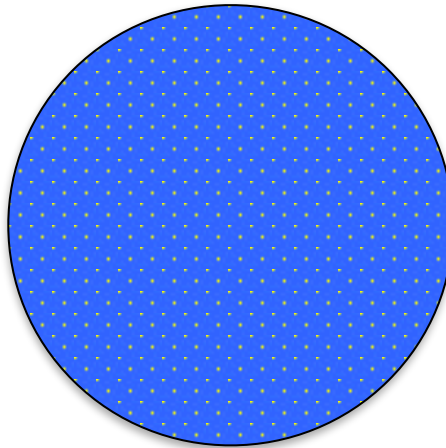
Template: AGACTATTTA  
(9x) TCTGAT  
Template: AGACTATTTA  
(1x) TCTGA

The methods described so far average the signal over millions of copies of the same sequence. Why is this problematic?

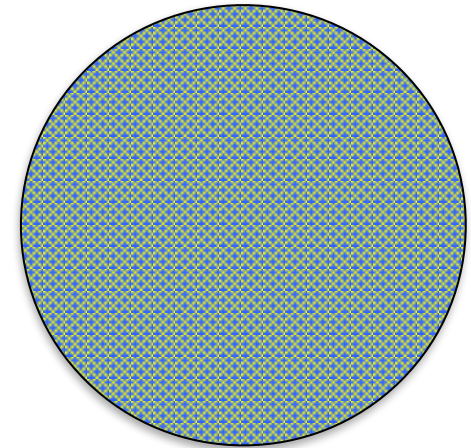
- Errors during PCR amplification render copies not 100% identical. Especially errors at an early stage of the PCR can mimic heterozygous positions.
- Not every copy of a pool of millions of sequences will incorporate a base in each cycle. With increasing numbers of cycles the length heterogeneity of the already sequenced fraction will increase and the sequencing will get **out of phase**.



Template: AGACTATTTA  
TCT

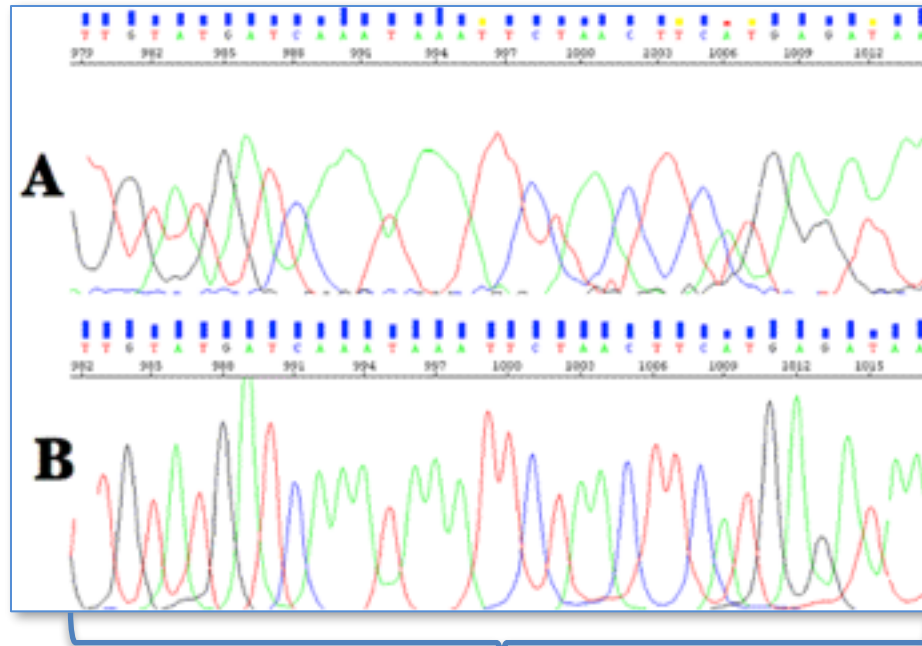


Template: AGACTATTTA  
(9x) TCTGAT  
Template: AGACTATTTA  
(1x) TCTGA



Template: AGACTATTTA  
(5x) TCTGATAAA  
Template: AGACTATTTA  
(5x) TCTGATAA

# Base quality values $Q$ (Sanger Sequence reads as example)



## Quality Parameters

- Peak Spacing (7)
- Uncalled/Called Ratio (7)
- Uncalled/Called Ratio (3)
- Peak Resolution

$$Q = -10 \log_{10}(P_e)^*$$

\* $P_e$ : empirical error probability

# Dealing with sequences means dealing with sequence formats

@Clagr-170543-2741/1

CAGAGAATAAATTCAATCTTCGCCAGCTACAAGTAGCTTTGAAATGGACTGGAATGGAGAAAGGGGATCATCTC  
AAACTTCTGGAAGAAGGCCGACAGCTGGTCTACAAAGGCCCTCTGAAGAAGAGTCCGACAGACTCTAGTGAAGT  
GCACGTTTACTTATTTAACACGCTTTGTTTTTTGTAAACAAAAGACGAGTAACAGGCAGGAGGAACACTACGGG  
TATACAAGAAGCCGATACCACTGGAGCT

+

?A????B?DDA<DBDDGAGC/GIHAH/IEFIIHIIHHFIHIIIII>HI?HHHDF-DFEGEIFHHIE7IIH  
IIHIIHHFHIIHIIHHBHHHHHHGIHHIHFHG;IEGGHH=FGEHGEGEHHDGEB?G@FAGICFCG4GE?>GE  
GEGCG@HG?CEEEFCE;E(8FFC<GGEGHA'GFG8E,6C?CGFFAGGC;GEFFFG?E\*GEAGEHHE6HECGGGE  
C;ACECAGGGCEGEG?GEEEC(E;EG\*

@Clagr-170541-2741/1

TATTTTAAGAATAAGATAATAAATATATTTAAGAATAGTGAATCTATTAATAAATTATTATAGAATAAATAA  
TTCATTTCTATATCTTAATAATAAGTACTTACTTAGTATTATCTTTATTAATTTATATAATAAGGAAGATATTA  
TAGTTAAAAGAATATGTCATAGTGAAGGCATAAGCGATGAAGCTAATATGGCTATGAAGCTCTAAACAGCTAT  
GTGATAACATAAAGCGATGTTCTAATGG

+

?<???B?B@DDD<@DDGGGGFFHIFIFIHHHIIIGIHHIHIHIIHIIHHIGIHHECHGCICIEHH=IFHHF58II  
IHHCIHHIIII@HFDFFIHFHIIHIEHIFGHIHF@HHGFHGEHHFHDDGGGGHHFEGGHGEGG?GGGFE\*=GDG  
GGFGC6EGGEC;?GGGCFEEEE)GG+GECG<G?GHAEG(FG;GG\*FEC;GFE<FEGFEAG3DFACFEED;CE  
G,EEEGE?CGCGC;EGCAGFGGGECEGG

# Dealing with sequences means dealing with sequence formats

## Sequence ID

@Clagr-170543-2741/1

```
CAGAGAAATAAATTCATCTTCGCCAGCTACAAGTAGCTTTGAAATGGACTGGAATGGAGAAAGGGGATCATCTC  
AAACTTCTGGAAGAAGGCCGACAGCTGGTCTACAAAGGCCCTCTGAGAAGAGTCCGACAGACTCTAGTGAAGT  
GCACGTTTACTTATTTAACACGCTTTGTTTTTGTAAACAAAAGACGAGTAACAGGCAGGAGGAACACTACGGG  
TATACAAGAAGCCGATACCACTGGAGCT
```

+

```
?A????B?DDA<DBDDGAGC/GIHAHIH/IEFIIHIIHHFIHIIIII>HI?HHHDF-DFEGEIFHHIE7IIH  
IIHIIHHFHIIIEHIIHBBHHHHHHGIHHIHFHG;IEGGHH=FGEHGEGEHHHDGEB?G@FAGICFCG4GE?>GE  
GEGCG@HG?CEEEFCE;E(8FFC<GGEGHA'GFG8E,6C?CGFFAGGC;GEFFFG?E*GEAGEHHE6HECGGGE  
C:ACECAGGGCEGGEG?GFEFC(E;EG*
```

@Clagr-170541-2741/1

```
TATTTTAAAGAAATAAGATAAATAAATAATATATTTAAGAATAGTGAATCTATTAATAAAATTATTATAGAATAAAAAT  
TTCATTTCTATATCTTAATAATAAGTACTTACTTAGTATTATCTTTATTAATTTATATAATAAGGAAGATATTA  
TAGTTAAAAGAATATGTCATAGTGAAGGCATAAGCGATGAAGCTAATATGGCTATGAAGCTCTAAACAGCTAT  
GTGATAACATAAAGCGATGTTCTAATGG
```

+

```
?????B?B@DDD<@DDGGGGGFFHIFIFIHHHIIIGIHHIHIHIIHIIHHIGIHHECHGCICIEHH=IFHHF58II  
IHHCIHHIIII@HFDFFIHFHIIHIEHIFGHIHF@HHGFHGEHHFHDDGGGGHHFEGGHGEGG?GGGFE*=GDG  
GGFGC6EGGEC;?GGGGCFEEEE)GG+GECG<G?GHAEG(FG;GG*FEC;GFE<FEGFEAG3DFACFEED;CE  
G,EEEGE?CGCGC;EGCAGFGGGECEGG
```



# Dealing with sequences means dealing with sequence formats

## Sequence ID

@Clagr-170543-2741/1

```
CAGAGAAATAAATTCATCTTCGCCAGCTACAAGTAGCTTTGAAATGGACTGGAATGGAGAAAGGGGATCATCTC  
AAACTTCTGGAAGAAGGCCGACAGCTGGTCTACAAAGGCCCTCTGAAGAAGAGTCCGACAGACTCTAGTGAAGT  
GCACGTTTACTTATTTAACACGCTTTGTTTTTGTAAACAAAAGACGAGTAACAGGCAGGAGGAACACTACGGG  
TATACAAGAAGCCGATACCACTGGAGCT
```

```
+  
?A????B?DDA<DBDDGAGC/GIHAHIH/IEFIIIHIIHHFIIHIIII>HI?HHHDF-DFEGEIFHHIE7IIH  
IIHIIHHFHIIIEHIIHBBHHHHHGIHHIHFHG;IEGGHH=FGEHGEGEHHDGEB?G@FAGICFCG4GE?>GE  
GEGCG@HG?CEEEFCE;E(8FFC<GGEGHA'GFG8E,6C?CGFFAGGC;GFFFFG?E*GEAGEHHE6HECGGGE  
C:ACECAGGGCEGGEG?GFEFC(E;EG*
```

} Sequence

@Clagr-170541-2741/1

```
TATTTTAAAGAAATAAGATAAATAAATAATATATTTAAGAATAGTGAATCTATTAATAAAATTATTATAGAATAAAAAT  
TTCATTTCTATATCTTAATAATAAGTACTTACTTAGTATTATCTTTATTAATTTATATAATAAGGAAGATATTA  
TAGTTAAAAGAATATGTCATAGTGAAGGCATAAGCGATGAAGCTAATATGGCTATGAAGCTCTAAACAGCTAT  
GTGATAACATAAAGCGATGTTCTAATGG
```

```
+  
?<??>B?B@DDD<@DDGGGGGFFHIFIFIHHHIIIGIHHIHIHIIIIHHHIGIHHECHGCICIEHH=IFHHF58II  
IHHCIHHIIII@HFDFFIHFIIHIEHIFGHIHF@HHGFHGEHHFHDDGGGGHHFEGGHGEGG?GGGFE*=-GDG  
GGFGC6EGGEC;?GGGGCFEEEE)GG+GECG<G?GHAEG(FG;GG*FEC;GFE<FEGFEAG3DFACFEED;CE  
G,EEEGE?CGCGC;EGCAGFGGGECEGG
```

# Dealing with sequences means dealing with sequence formats

## Sequence ID

@Clagr-170543-2741/1

```
CAGAGAAATAAATTCATCTTCGCCAGCTACAAGTAGCTTTGAAATGGACTGGAATGGAGAAAGGGGATCATCTC  
AAACTTCTGGAAGAAGGCCGACAGCTGGTCTACAAAGGCCCTCTGAGAAGAGTCCGACAGACTCTAGTGAAGT  
GCACGTTTACTTATTTAACACGCTTTGTTTTTTGTAAACAAAAGACGAGTAACAGGCAGGAGGAACACTACGGG  
TATACAAGAAGCCGATACCACTGGAGCT
```

} Sequence

```
+  
?A????B?DDA<DBDDGAGC/GIHAHIH/IEFIIHIIHHFIHIIIII>HI?HHHDF-DFEGEIFHHIE7IIH  
IIHIIHHFHIIIEHIIHBBHHHHHHGIHHIHFHG;IEGGHH=FGEHGEGEHHHDGEB?G@FAGICFCG4GE?>GE  
GEGCG@HG?CEEEFCE;E(8FFC<GGEGHA'GFG8E,6C?CGFFAGGC;GEFFFG?E*GEAGEHHE6HECGGGE  
C:ACEFCAGGGCEGGEG?GFEFC(E;EG*
```

@Clagr-170541-2741/1

```
TATTTTAAAGAAATAAGATAAATAAATAATATATTTAAGAATAGTGAATCTATTAATAAAATTATTATAGAATAAAAAT  
TTCATTTCTATATCTTAATAATAAGTACTTACTTAGTATTATCTTTATTAATTTATATAATAAGGAAGATATTA  
TAGTTAAAAGAATATGTCATAGTGAAGGCATAAGCGATGAAGCTAATATGGCTATGAAGCTCTAAACAGCTAT  
GTGATAACATAAAGCGATGTTCTAATGG
```

```
+  
?<??B?B@DDD<@DDGGGGFFHIFIFIHIIHIGIHHIHIHIIHIIHHIGIHHHECHGCICIEHH=IFHHF58II  
IHHCIHHIIII@HFDFFIHFIIHIEHIFGHIHF@HHGFHGEHHFHDDGGGGHHFEGGHGEGG?GGGFE*=-GDG  
GGFGC6EGGEC;?GGGCFEEEE)GG+GECG<G?GHAEG(FG;GG*FEC;GFE<FEGFEAG3DFACFEeee;CE  
G,EEEGE?CGCGC;EGCAGFGGGECEGG
```

Separator

# Dealing with sequences means dealing with sequence formats

## Sequence ID

@Clagr-170543-2741/1

CAGAGAAATAAATTCATCTTCGCCAGCTACAAGTAGCTTTGAAATGGACTGGAATGGAGAAAGGGGATCATCTC  
AAACTTCTGGAAGAAGGCCGACAGCTGGTCTACAAAGGCCCTCTGAGAAGAGTCCGACAGACTCTAGTGAAGT  
GCACGTTTACTTATTTAACACGCTTTGTTTTTTGTAAACAAAAGACGAGTAACAGGCAGGAGGAACACTACGGG  
TATACAAGAAGCCGATACCACTGGAGCT

} Sequence

+  
?A????B?DDA<DBDDGAGC/GIHAHIH/IEFIIHIIHHFIHIIIII>HI?HHHDF-DFEGEIFHHIE7IIH  
IIHIIHHFHIIIEHIIHBBHHHHHGIHHIHFHG;IEGGHH=FGEHGEGEHHDGEB?G@FAGICFCG4GE?>GE  
GEGCG@HG?CEEEFCE;E(8FFC<GGEGHA'GFG8E,6C?CGFFAGGC;GEFFFG?E\*GEAGEHHE6HECGGGE  
C:ACECAGGGCEGGEG?GFEFC(E;EG\*

} Sequence Quality String

@Clagr-170541-2741/1

TATTTTAAAGAAATAAGATAAATAAATAATATATTTAAGAATAGTGAATCTATTAATAAAATTATTATAGAATAAAAAT  
TTCATTTCTATATCTTAATAATAAGTACTTACTTAGTATTATCTTTATTAATTTATATAATAAGGAAGATATTA  
TAGTTAAAAGAATATGTCATAGTGAAGGCATAAGCGATGAAGCTAATATGGCTATGAAGCTCTAAACAGCTAT  
GTGATAACATAAAGCGATGTTCTAATGG

+  
?<??>B?B@DDD<@DDGGGGFFHIFIFIHHHIIIGIHHIHIHIIIIHHHIGIHHECHGCICIEHH=IFHHF58II  
IHHCIHHIIII@HFDFFIHFHIIHIEHIFGHIHF@HHGFHGEHHFHDDGGGGHHFEGGHGEGG?GGGFE\*=-GDG  
GGFGC6EGGEC;?GGGCFEEEE)GG+GECG<G?GHAEG(FG;GG\*FEC;GFE<FEGFEAG3DFACFEED;CE  
G,EEEGE?CGCGC;EGCAGFGGGECEGG

Separator



# The file format conversion is a typical problem in bioinformatics analyses and in some instances not reversible

```
@Clagr-170543-2741/1
CAGAGAATAAATTCAATCTTCGCCAGCTACAAGTAGCTTTGAAATGGAC
TGGAATGGAGAAAGGGGATCATCTCAAACCTCTGGAAGAAGGCCGACAG
CTGGTCTACAAAGGCCCTCTGAAGAAGAGTCCGACAGACTCTAGTGAAG
TGCACGTTTACTTATTTAACACGCTTTGTTTTTTGTAAAACAAAAGAC
GAGTAACAGGCAGGAGGAACACTACGGGTATACAAGAAGCCGATACCCTG
GAGCT
+
?A????B?DDA<DBDDGAGC/GIHAHIH/
IEFIIIIHHHFIHIIIII>HI?HHHDF/
DFEGEIFHHIE7IIHIIHHHFIIEHIIHBBHHHHHHGIHHIHFHG;I
EGGHH=FGEHGGEGEHHDGEB?G@FAGICFCG4GE?>GEGEGCG@HG?
CEEEFCE;E(8FFC<GGEGHA'GFG8E.6C?CGFFAGGC:GEFFFG?
E*GEAGEHHE6HECGGEC;ACECAGGGCEGEG?GEEEC(E:EG*
@Clagr-170541-2741/1
TATTTTAAGAATAAGATAATAAATATATTTAAAGAATAGTGAATCTAT
TAAAAATTATTATAGAATAAAAAATTTCAATTTCTATATCTTAATAATA
GTACTTACTTAGTATTATCTTTATTAATTTATATAATAAGGAAGATATT
ATAGTTAAAAGAATATGTCATAGTGAAGGCATAAGCGATGAAGCTAATA
TGGCTATGAAGCTCTAAAACAGCTATGTGATAACATAAAGCGATGTTCT
AATGG
+
?<????B?
B@DDD<@DDGGGGFFHIFIFIHIIIGIHHIHHIHHIHHIHHIHHIHHI
GCICIEHH=IFHHF58IIHHCIHHIIII@HFDFFIHIFHIIHIEHIFGH
IHF@HHGFHGEHFFHDGGGGHHFEGGHGEGG?
GGGFE*=GDGGGFGC6EGGEC:?GGGGCFEEEE)GG+GECG<G?
GHAEG(FG:GG*FEC:GFE<FEGFEAG3DFACFEEEE:CEG.EEEGE?
CGCGC:EGCAGFGGGECEGG
```

```
>Clagr-170543-2741/1
CAGAGAATAAATTCAATCTTCGCCAGCTACAAGTAGCTTTGAAATGGACTGGAATG
GAGAAAGGGGATCATCTCAAACCTCTGGAAGAAGGCCGACAGCTGGTCTACAAAGG
CCCTCTGAAGAAGAGTCCGACAGACTCTAGTGAAGTGCACGTTTACTTATTTAACC
ACGCTTTGTTTTTTGTAAAACAAAAGACGAGTAACAGGCAGGAGGAACACTACGGGTA
TACAAGAAGCCGATACCCTGAGCT
>Clagr-170541-2741/1
TATTTTAAGAATAAGATAATAAATATATTTAAAGAATAGTGAATCTATTAATAAATA
TTATTATAGAATAAAAAATTTCAATTTCTATATCTTAATAATAAGTACTTACTTAGTA
TTATCTTTATTAATTTATATAATAAGGAAGATATTATAGTTAAAAGAATATGTCAT
AGTGAAGGCATAAGCGATGAAGCTAATATGGCTATGAAGCTCTAAAACAGCTATGT
GATAACATAAAGCGATGTTCTAATGG
```

Solution exists...

[http://hannonlab.cshl.edu/fastx\\_toolkit/](http://hannonlab.cshl.edu/fastx_toolkit/)

<http://molbiol-tools.ca/Convert.htm>

but sometimes are hard to use...

Other formats

Nexus

Paup

.doc

txt

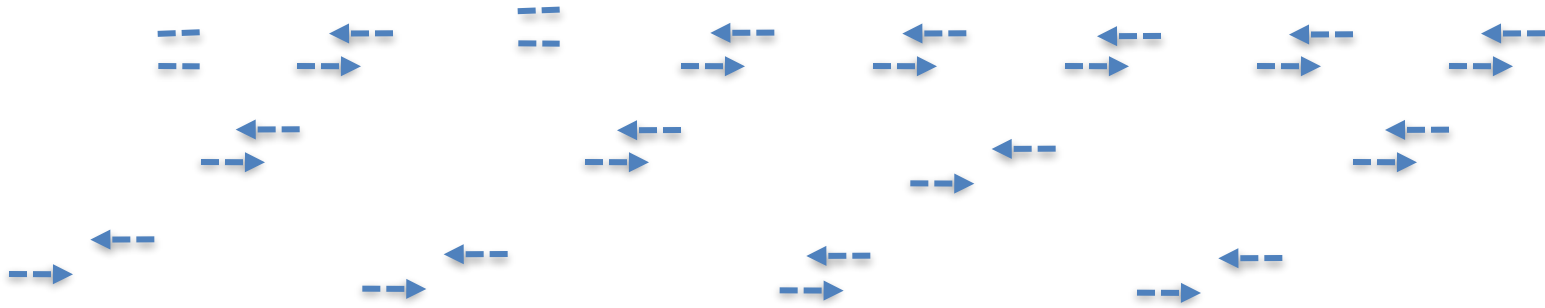
...



# Introduction into Text Processing with PERL

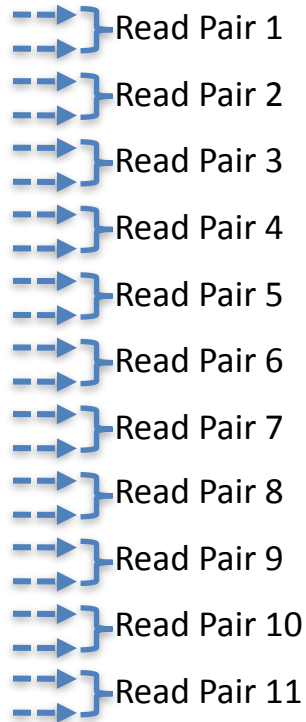
<http://seqanswers.com/forums/index.php>

# Strategies to sequence long DNA molecules: Shotgun Sequencing



1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments
4. Identify and remove adapter part from the sequence reads

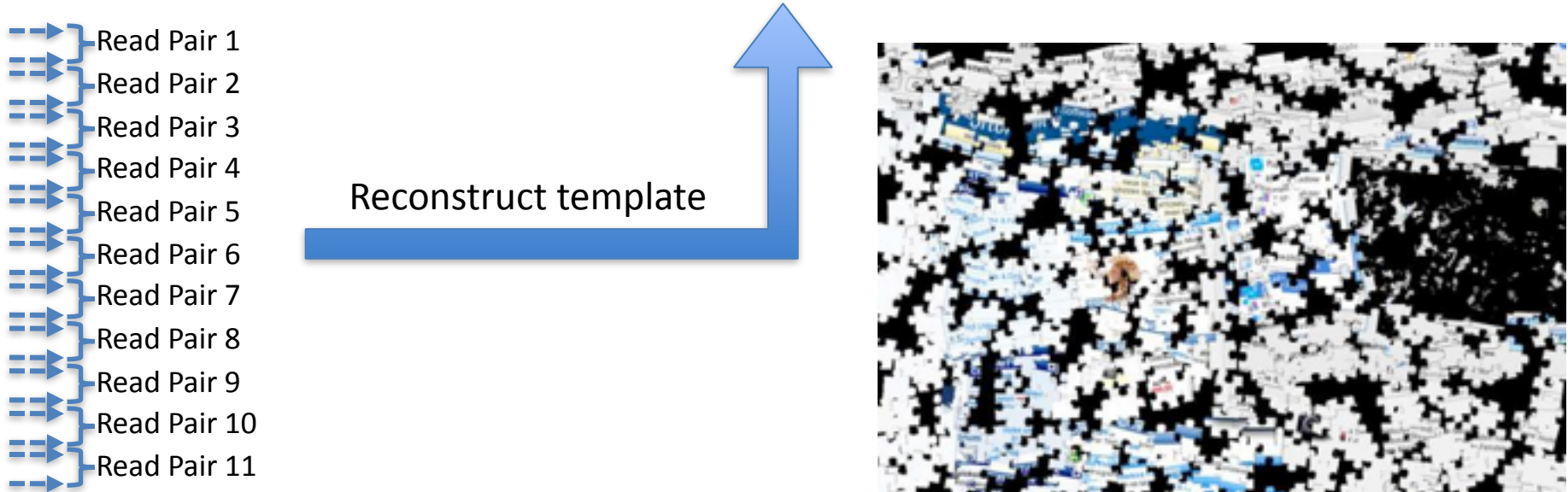
# Strategies to sequence long DNA molecules: Shotgun Sequencing



1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments
4. Identify and remove adapter part from the determined sequences



# Strategies to sequence long DNA molecules: Shotgun Sequencing



1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments
4. Identify and remove adapter part from the determined sequences
5. Reconstruct template sequence from the sequence reads

## **Assembly:**

A hierarchical data structure that maps the sequence data to a reconstruction of the target. It groups reads into contigs and contigs into scaffolds. Contigs provide a multiple sequence alignment of reads plus the consensus sequence. The scaffolds (sometimes called supercontigs) define the contig order and orientation and the sizes of the gaps between contigs.

# Why are we here?

- We want to solve problems automatically that are either too time consuming or too complex to solve them manually, or that occur so often that we want to have a standardized<sup>1</sup> solution.





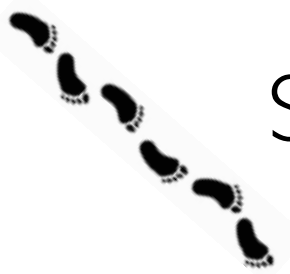
A little sad example from the past...

- The task was: “In how many positions do humans and chimpanzees differ in their ZFX gene?”
- The solution was: “Print out the alignment, get equipped with a set of markers and start counting...”.




A little sad example from the past...

- The task was: “In how many positions do humans and chimpanzees differ in their ZFX gene?”
- The solution was: “Print out the alignment, get equipped with a set of markers and start counting...”.
- Unfortunately, the alignment was about 100,000 bp in length :(



# Some general and obvious things to consider



- What is my problem? The more precise you can formulate it the better!
- What is my problem? The more abstract you can formulate it the better!
- How can I formulate the problem solution procedure, i.e. the algorithm?
- What does my input look like? (Are you sure?)
- How should my output look like?
- What can go wrong and how do I capture errors?



# Perl, one solution to your problems...



Perl was created by Larry Wall.

(read his [forward to the book "Learning Perl"](#))

Perl = Practical Extraction and Report Language

Perl is a scripting language

Perl was originally developed for text processing

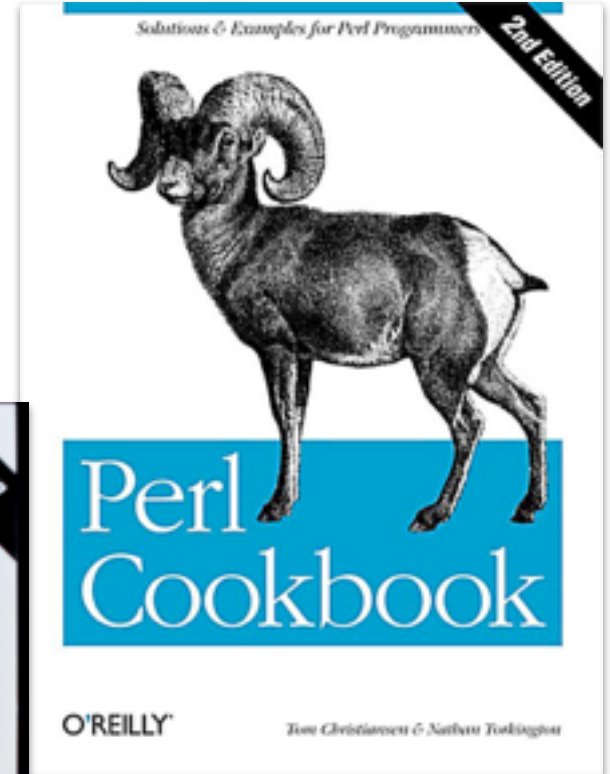
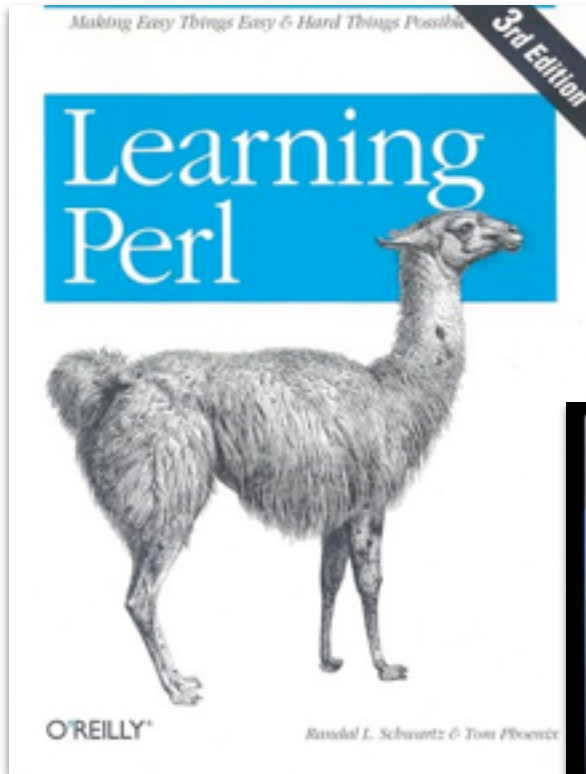


# Why Perl ?

- Open Source project
- Perl is a cross-platform programming language
- Perl is a very popular programming language, especially for bioinformatics
- Perl is strong in text manipulation
- Perl can easily handle files and directories
- Perl can easily run other programs



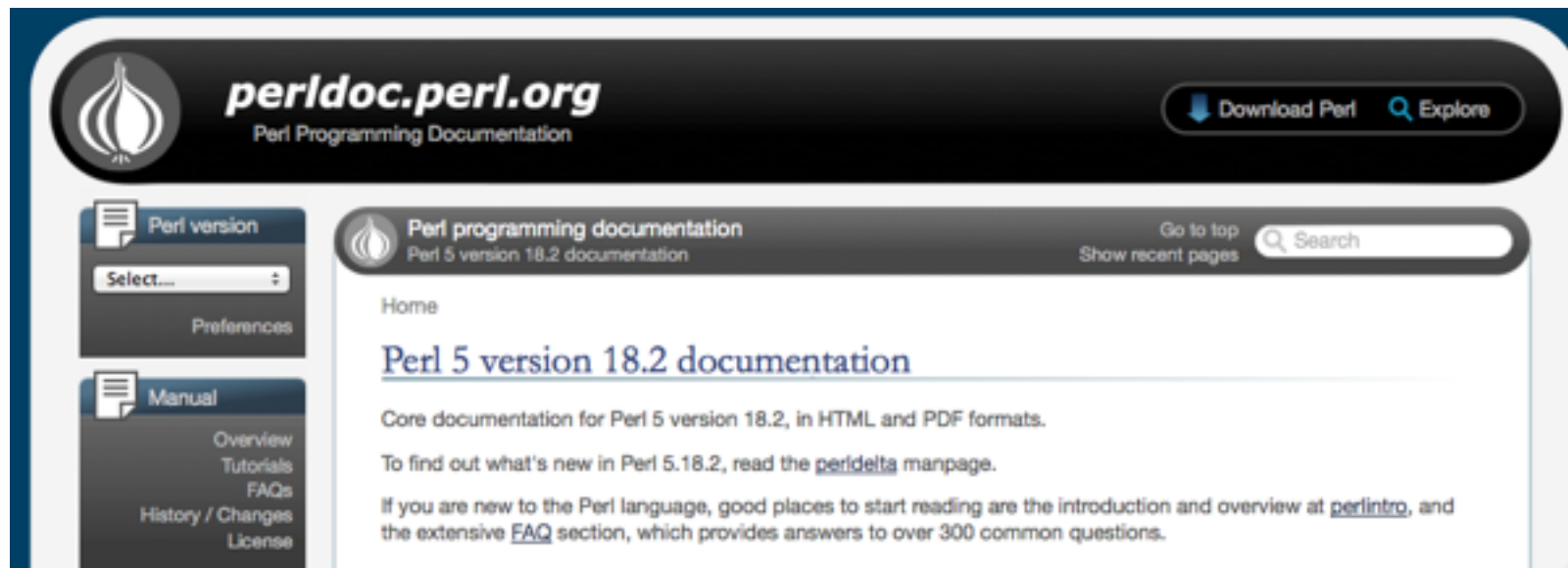
# Literature about Perl



# Documentation of perl functions

A good place to start is the list of all basic Perl functions in the Perl documentation site:

<http://perldoc.perl.org/>



The screenshot shows the Perl documentation website. The header features the Perl logo and the text "perldoc.perl.org Perl Programming Documentation". Navigation links include "Download Perl" and "Explore". A sidebar on the left contains "Perl version" (with a "Select..." dropdown) and "Manual" (with links for Overview, Tutorials, FAQs, History / Changes, and License). The main content area displays "Perl programming documentation Perl 5 version 18.2 documentation" and includes a search bar. The page title is "Perl 5 version 18.2 documentation", and the content describes the core documentation for Perl 5 version 18.2, available in HTML and PDF formats. It also provides links to the [perldelta](#) manpage for updates and [perlintro](#) for new users, along with an extensive [FAQ](#) section.

# Setting the stage

## A very simple Perl script

The shebang points to the interpreter located at `/usr/bin/perl`

```
#!/usr/bin/perl
# The following line just prints the string
#'Hello world!'
print "Hello world!";
exit;
```

just a comment

The *print* function outputs some information to the terminal screen

'*exit*' denotes the termination of the script (optional)

A *Perl statement* must end with a semicolon

# Your very first Perl script

Now it is (almost) your turn

```
#!/usr/bin/perl -w
use strict;
# The following line just prints the string
#'Hello world!'
print "Hello world!";
exit;
```

- Write this script in a text editor
- Save it under `~/Desktop/perl_course/scripts/hello.pl`
- Execute this script by visiting the directory and typing `./hello.pl`

# Your very first Perl script

Now it is (almost) your turn


```
#!/usr/bin/perl -w
use strict;
# The following line just prints the string
#'Hello world!'
print "Hello world!";
exit;
```

- Write this script in a text editor
- Save it under `~/Desktop/perl_course/scripts/hello.pl`
- Execute this script by visiting the directory and typing `./hello.pl`

But there are some further things to consider...

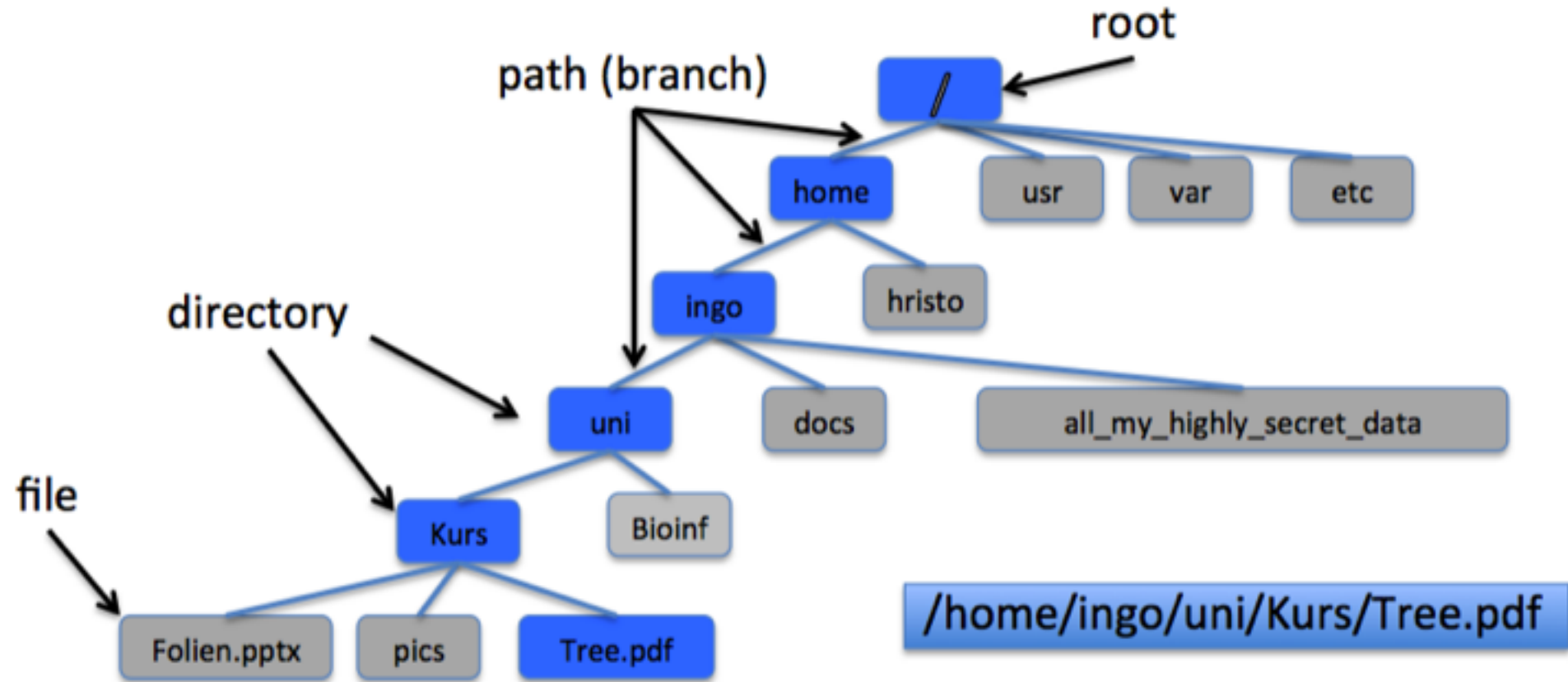
# Your very first Perl script

Traditionally, Perl scripts are run from a command line interface  
Start one by clicking *Applications* in the top menu bar -> System Tools -> Terminal

A screenshot of a terminal window. The title bar at the top reads "scripts — bash — 80x24". The terminal content shows a shell prompt "bastian@phix:~/Desktop/perl\_course/scripts\$" with a cursor at the end. The prompt is colored green for the user, red for the host, and black for the path and shell character.

```
scripts — bash — 80x24
bastian@phix:~/Desktop/perl_course/scripts$
```

# The directory structure of the linux operating system



# Using the command line (shell) in linux

First let's go to the correct directory:

`pwd:` - shows you the current path

`cd` - change to the home directory '~' from wherever you are

`cd ~/Desktop/perl_course` - change directory to the perl\_course directory

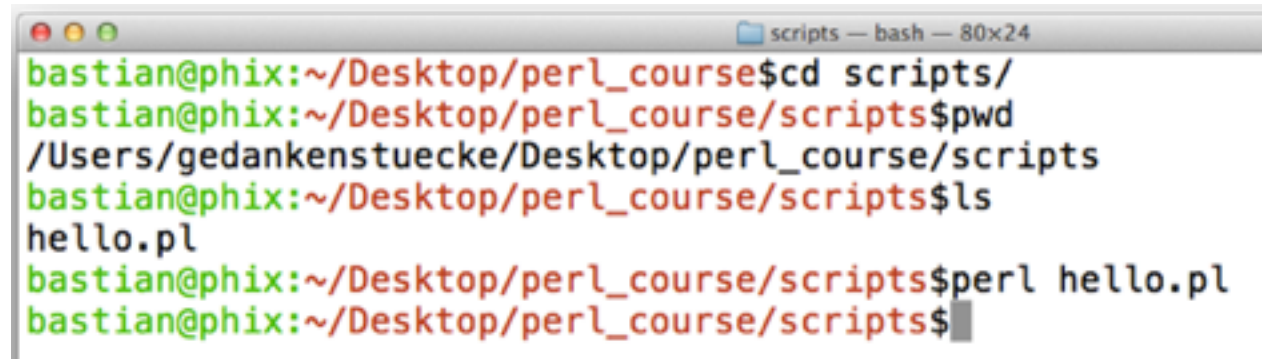
`cd scripts` - change directory to the scripts directory

`ls` - list all the files in the directory (you should see your script here)

`chmod a+x hello.pl` tells your operating system that hello.pl is an executable

Running the Perl script

`./hello.pl`

A terminal window titled "scripts — bash — 80x24" showing a series of commands and their outputs. The user is in the directory ~/Desktop/perl\_course/scripts. The commands and outputs are: 

```
bastian@phix:~/Desktop/perl_course$ cd scripts/  
bastian@phix:~/Desktop/perl_course/scripts$ pwd  
/Users/gedankenstuecke/Desktop/perl_course/scripts  
bastian@phix:~/Desktop/perl_course/scripts$ ls  
hello.pl  
bastian@phix:~/Desktop/perl_course/scripts$ perl hello.pl  
bastian@phix:~/Desktop/perl_course/scripts$
```



# Using the command line (shell) in linux

Common useful commands in the shell (command line)<sup>1</sup>:

<code>mkdir my_dir</code>	make a new directory called 'my_dir'
<code>cd my_dir</code>	change to the sub-directory 'my_dir'
<code>cd ..</code>	move one directory up
<code>ls</code>	list files
<code>man dir</code>	get help on a particular command, here 'mkdir'
<code>&lt;TAB&gt;</code>	(hopefully) auto-complete an input
<code>&lt;up/down&gt;</code>	go to previous/next command
<code>&lt;Ctrl&gt;-c</code>	Emergency exit to interrupt a process
<code>which perl</code>	asks where your perl interpreter is located on this system.
<code>chmod a+x hello.pl</code>	tells your operating system that hello.pl is an executable

<sup>1</sup> For further explanations about using commands in the shell see our tutorial

# Two possible ways to return information

- Print to standard out (the screen)

- We have seen this already:

- `print “my information for user”;`

- For printing advanced formats, such as rounded numbers, see the `printf` function in perl.



# Two possible ways to return information

- Print to a file on the hard drive



- requires opening a file handle

- this is new, looks complicated, but is not

```
open (OUT, ">myoutputfile.txt");
```

```
print OUT "my information that should be stored on disk";
```

```
print OUT "some more information";
```

```
close OUT;
```

- If the file 'myoutputfile.txt' is not yet existing, it will be generated on the fly.

- If the file 'myoutputfile.txt' is already existing, its content will be completely overwritten! You can tell perl to append the information by using '>>' instead of '>'.

# How to advance from here



- Perl scripting becomes more interesting when we start doing the following things...
  - store data in variables within the script
  - modifying data within the script
  - loading data from various sources and various formats
  - interact dynamically with the user
  - **modifying data in files**
  - **and outputting data to the screen or to a file**

# Link to slides and exercises



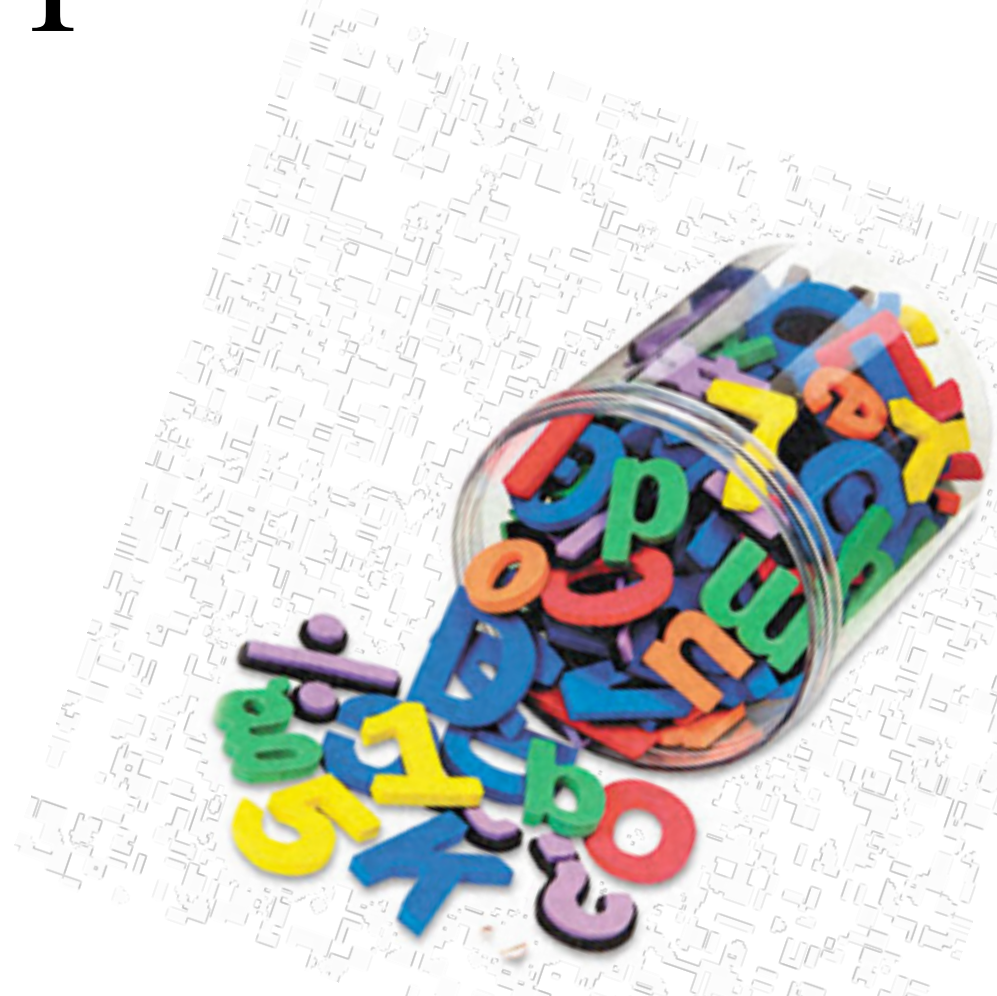
- <http://fsbioinf.biologie.uni-frankfurt.de/asa2014>
  - user: asa
  - password: asa2014
- Please concentrate on the first set of Tasks
  - Task 1.1 - Linux basics
  - Task 1.2 - Perl basics
  - Task 1.3 - My first Perl script - Printing to standard out

# Your very first Perl script

Now it is really your turn

```
#!/usr/bin/perl -w
use strict;
# The following line just prints the string #'Hello world!'
print "Hello world!";
# The following line opens a Filehandle returning an error
# if this is not possible (you can take 'die' literally)
open (OUT, ">hello.out") or die "could not open outfile\n";
print OUT "Hello world!";
close OUT;
exit;
```

# 1. Data type: Scalar



# Scalar variables: Numbers & Strings

Perl allows the storage of scalar values in a variable starting with a

\$

followed by the name of the variable.

`$firstvariable`



# Variables - always *use strict*!

Always include the line:

```
use strict;
```

as the first line of every script after the [shebang](#).

- “Strict” mode forces you to declare all variables by [my](#)
- This will help you avoid very annoying bugs, such as spelling mistakes in the names of variables.

```
my $varname = 1;  
$varName++;
```

Warning:

Global symbol "\$varName" requires explicit package name at ... line ...

# Scalar variables: Numbers & Strings

Perl allows the storage of scalar values in a variable starting with a

\$

followed by the name of the variable.

**\$firstvariable**

In principle you are free to use any variable name you can imagine but there are few guidelines

- First time you introduce a variable you **have to** declare it using ‘**my**’
- avoid using numbers as names (perl uses e.g. \$1, \$2,... for its own purpose)
- don’t use \$\_ or \$\$ (for the same reason as above)
- don’t use too complex names, e.g. \$hghCVEgdiU, as you are prone to misspell them later in the script...
- it might be a good idea to use names that are somehow related to the information stored in this variable, e.g. \$input

# Variables

Scalar variables can store scalar values.

Variable **declaration**

```
my $priority;
```

**Numerical** assignment

```
$priority = 1;
```

**String** assignment

```
$priority = 'high';
```

Note: Assignments are evaluated from **right to left**

Multiple variable **declaration**

```
my $a, $b;
```

**Copy** the value of variable `$priority` to `$a`

```
$a = $priority;
```

← Assignment from right to left

Note: Here we make a **copy** of `$priority` in `$a`.

# Scalar variables hold numerical values!

A scalar can be a number.

3

-20

3.14152965

1.3e4 (=  $1.3 \times 10^4 = 1,300$ )

6.35e-14 (=  $6.35 \times 10^{-14}$ )

You assign a numerical value to a variable simply by using the following syntax:

```
$variable = 1;
```

```
$othervar = 47;
```

```
$thirdvar = 1.777;
```

```
$fourthvar = 1e-17;
```

# Scalar variables can **ALSO** hold string values<sup>1</sup>!

Strings are anything that we typically consider as letters, words, or sentences. In biology DNA or protein sequences are among the most commonly used strings.

You assign a string to a variable simply by using the following syntax:

```
$strvar = 'Hello World'; # Holds Hello World  
$otherstrvar = 'AGAACTCCATG'; # A DNA sequence  
$thirdstrvar = 'MCGKRRWT'; # A protein sequence  
$fourthstrvar = '$strvar\t\n\s'; # holds string '$strvar\t\n\s'
```

Just a comment!

Anything within single quotes will be taken literally!

<sup>1</sup> Note, when working with strings Perl does typically not check for whether you are operating on strings or numerical values! It simply interprets the variable as string or number according to the context.

# Scalar variables can **ALSO** hold string values!

You can assign a string to a variable also by using double quotes:

```
$strvar = "Hello World"; # Holds 'Hello World'  
$otherstrvar = "AGAACTCCATG"; #A DNA sequence  
$thirdstrvar = "MCGKRRWT"; # A protein sequence  
$fourthstrvar = "$strvar how are you?"; # holds 'Hello World how  
# are you?'
```

Perl will try to interpolate anything within double quotes!!

Backslash is an “escape” character that gives the next character a special meaning:

Construct	Meaning
<code>\n</code>	Newline
<code>\t</code>	Tab
<code>\\</code>	Backslash
<code>\"</code>	Double quote

# The first approach to user interaction

- We can assign values to variables by hard-coding the information into the script

```
my $output = "Hello World\n";
```

- We can ask the user to dynamically enter information via the command line

```
my $output = <STDIN>;
```

**'chomp'** removes the newline character '\n' from a string (remember, you have to hit enter to complete your user input on the command line)!

```
#!/usr/bin/perl -w
use strict;
my $message = "Please enter your name\n";
print $message;
my $user = <STDIN>;
chomp $user;
print "Hello $user, how are you?\n";
exit;
```

# Using operators to work with variables

An operator takes some values (operands), operates on them, and produces a new value.

Numerical operators can be used to do math:  $+$   $-$   $*$   $/$   $\%$

```
$var1 = 2;
```

```
$var2 = 3;
```

```
$var3 = $var1 + $var1; # $var3 holds the value 4
```

```
$var3 = $var3 - $var2; # $var3 holds the value 1
```

```
$var4 = $var1 / $var2; # $var4 holds the value 2/3
```

```
$var2 = $var2 + 1; Increment $var2 by 1
```

```
$var2 = $var2++; # Increment $var2 by 1
```

```
$var2 += 1; # Increment $var2 by 1
```

```
$var5 = $var1**2; # $var5 holds now the value 4
```

```
$var6 = ($var1+$var1)%2; # $var6 holds now 4%2, i.e. 0
```

This is all the same!





# Using operators to work with variables

String Operators can be used to:

- Concatenate strings using ‘.’
- Replicate strings using ‘x’

```
$var1 = ‘I am hungry!’;
```

```
$var2 = ‘Give me something to eat!’;
```

```
print “$var1 $var2”; # Obvious, right?
```

```
$var3 = $var1 . $var2; # Holds ‘I am hungry!Give me...’
```

```
$var3 = $var1 . ‘ ‘ . $var2; # Holds ‘I am hungry! Give me...’
```

```
$var4 = $var1x3; # Holds ‘I am hungry!I am hungry!I am hungry!’
```

```
print $var1 . ‘ ‘ . $var2; # same result as line 3 in the example
```

```
print (($var1 . ‘ ‘)x3); # prints I am hungry! I am hungry! I am hungry! ’
```

Please note the last white space!!



# String or number?

Perl decides the type of a value depending on its **context** but it is **HIGHLY** advisable to use **variables only in the correct context!!**

**(9+5).'a'**

**14.'a'**

**'14'.'a'**

**'14a'**

**(9x2) +1**

**('9'x2) +1**

**'99' +1**

**99+1**

**100**



These things can  
cause serious trouble

**Warning:** When you use **parentheses** in print make sure to put one pair of parantheses around the **WHOLE** expression:

```
print (9+5).'a'; # wrong
```

```
print ((9+5).'a'); # right
```

You will know that you have such a problem if you see this warning:

**print (...) interpreted as function at ex1.pl line 3.**

# Assigning Values to Variables

For example:

	\$a	\$b
my \$a = 1;	1	undef
my \$b = \$a;	1	1
\$b = \$b + 1;	1	2
\$b++;	1	3
\$a--;	0	3

# Uninitialised variables

Uninitialised variables (before assignment) receive a special value:

`undef`

If uninitialised variables are used a warning is issued:

```
my $a;
```

```
print($a+3);
```

Use of uninitialised value in addition (+)

3

```
print("a is :$a:");
```

Use of uninitialised value in concatenation (.) or string

a is ::

# The *length* function

The *length* function returns the length of a string:

```
my $str = "hi you";  
print length($str);
```

6

Actually *print* is also a function so you could write:

```
print(length($str));
```

6

# The *split* function

The *split* function splits a string at the specified character:

```
my $str = "hi you";  
my ($first, $second) = split //, $str; # splits the string at each white space  
print "First word is '$first', second word is '$second'\n";
```

First word is 'hi', second word is 'you'

Note, the for  $n$  split characters in the string *split* will return a list of  $n+1$  strings!

# The *substr* function

The *substr* function extracts a substring out of a string.

It receives 3 arguments: `substr(EXPR,OFFSET,LENGTH)`

Note: `OFFSET` count starts from 0.

For example:

```
my $str = "university";
```

```
my $sub = substr($str, 3, 5);
```

`$sub` is now "versi", and `$str` remains unchanged.

Also note : You can use variables as the offset and length parameters.

The *substr* function can do a lot more, Google it and you will see...

# Filehandles: Reading from and writing into files

- We already have learned how to write into files

```
open (OUT, ">myoutputfile.txt") or die "could not open\n";  
print OUT "$mytext\n";  
close OUT;
```

- The syntax for reading from files is similar

```
open (IN, "myinputfile.txt") or die "could not open file for  
reading\n";  
my $firstline = <IN>;  
chomp $firstline;  
close IN;  
print "First line is $firstline\n";
```



# Exercises 2



- Task set 2
  - Task 2.1 - Printing to a file
  - Task 2.2 - Reading from Standard In (the command line)
  - Task 2.3 - Reading from a file
  - Task 2.4 - Accessing parts of strings: split and substr

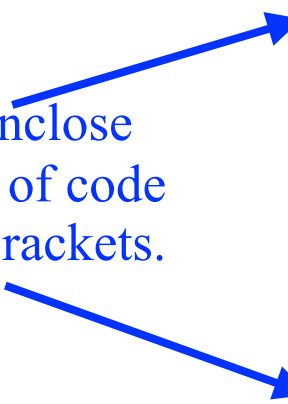
# Adding structure to the code

```
#!/usr/bin/perl -w
use strict;
print "Please give me a filename\n";
my $filename = <STDIN>;
chomp $filename;
open (IN, "$filename") or die "could not find $filename";
my $firstline = <IN>;
close IN;
my ($firstword) = split //, $firstline;
print "First word of first line in $filename is $firstword";
```

# Adding structure to the code

```
#!/usr/bin/perl -w
use strict;
print "Please give me a filename\n";
my $filename = <STDIN>;
chomp $filename;
{
  open (IN, "$filename") or die "could not find $filename";
  my $firstline = <IN>;
  close IN;
  my ($firstword) = split / /, $firstline;
  print "First word of first line in $filename is $firstword";
}
```

You can enclose  
any block of code  
by curly brackets.



# Adding structure to the code

## Scope of variables

Any variable declared with 'my' is valid only

- within in the code block it has been declared in!
- and in any code block nested within the block it has been declared in.

```
#!/usr/bin/perl -w
use strict;
print "Please give me a filename\n";
my $filename = <STDIN>;
chomp $filename;
{
    open (IN, "$filename") or die "could not find $filename";
    my $firstline = <IN>;
    close IN;
    my ($firstword) = split //, $firstline;
    print "First word of first line in $filename is $firstword";
}
```

To enhance readability of your script you can use indentation to make blocks standing out in the code. Perl will ignore this layout!

# Adding structure to the code

## Scope of variables

```
#!/usr/bin/perl -w
use strict;
print "Please give me a filename\n";
my $filename = <STDIN>;
chomp $filename;
{
    open (IN, "$filename") or die "could not find $filename";
    my $firstline = <IN>;
    close IN;
    my ($firstword) = split //, $firstline;
    print "First word of first line in $filename is $firstword";
}
print "$firstword\n";
```

OK

NOT OK

- Any variable declared with 'my' is valid only
- within in the code block it has been declared in!
  - and in any code block nested within the block it has been declared in.

Global symbol "\$firstword" requires explicit package name at ./hello.pl line 13. Execution of ./hello.pl aborted due to compilation errors.

# Adding structure to the code: Conditional statements *if* and *else*

You can enclose  
any block of code  
by curly brackets  
and execute it only  
IF a given  
conditional  
statement is true →

```
#!/usr/bin/perl -w
use strict;
print "Please give me a filename\n";
my $filename = <STDIN>;
chomp $filename;
if ("thisstatement is true"){
    open (IN, "$filename") or die "could not find $filename";
    my $firstline = <IN>;
    close IN;
    my ($firstword) = split //, $firstline;
    print "First word of first line in $filename is $firstword";
}
```

# Adding structure to the code

You can enclose any block of code by curly brackets and execute it only IF a given conditional statement is true ELSE you can do something different.

```
#!/usr/bin/perl -w
use strict;
print "Please give me a filename\n";
my $filename = <STDIN>;
chomp $filename;
if ("thisstatement is true"){
    open (IN, "$filename") or die "could not find $filename";
    my $firstline = <IN>;
    close IN;
    my ($firstword) = split //, $firstline;
    print "First word of first line in $filename is $firstword";
}
else {
    print "Condition was not met. I will exit\n";
    exit;
}
```

# Adding structure to the code

The expression `if (-e "$filename")` tests if the specified file exists in the current directory. It returns `'TRUE'` if this is the case, otherwise `'FALSE'`.

```
#!/usr/bin/perl -w
use strict;
print "Please give me a filename\n";
my $filename = <STDIN>;
chomp $filename;
if (-e "$filename"){
    open (IN, "$filename") or die "could not find $filename";
    my $firstline = <IN>;
    close IN;
    my ($firstword) = split / /, $firstline;
    print "First word of first line in $filename is $firstword";
}
else {
    print "I could not find the file $filename. I will exit\n";
    exit;
}
```

Note, to enhance readability of your script you can use an offset to make blocks standing out in the code. However, Perl will ignore this layout!



# True or False?

- True:
  - `1 # 1` is always TRUE
  - `1 == 1 #` this comparison is also true. Note, you need two equal signs, otherwise it is an assignment!
  - `1 < 2 #` Also true
  - `$stringvar eq $stringvar #` you compare whether two variables contain the same string. Here of course true as you compare the variable with itself
  - `length('test') < 5 #` True, as 'test' holds only 4 characters.
  - `defined $anyvar #` True, if \$anyvar holds a value

# True or False?

- FALSE:
  - `0 # 0` is always FALSE
  - `1 == 2` # Of course false. Note, you need two equal signs, otherwise it is an assignment!
  - `2 < 1` # Also FALSE
  - `$stringvar ne $stringvar` # you compare whether two variables contain different strings. Here of course FALSE as you compare the variable with itself
  - `length('test') > 5` # FALSE, as 'test' holds only 4 characters.
  - `defined $anyvar` # FALSE if \$anyvar holds no value

# True or False?

- The ‘!’ (Not) character turns TRUE into FALSE and vice versa.
  - !0 # ‘Not FALSE’ is TRUE
  - !1 # ‘Not TRUE’ is FALSE
  - !(defined \$anyvar) # True, if \$anyvar is NOT defined
  - !(-e “\$filename”) # True if a file with this name does not exist.
  - etc...

# Conditional statements in the code

## General structure

```
if (condition1) {  
    some code block1;  
}  
elseif (condition2) {  
    some alternative codeblock2;  
}  
elseif (condition3) {  
    some alternative codeblock3;  
}  
else {  
    last possible codeblock;  
}
```

# Conditional statements in the code

## Combining conditions with 'and' or 'or'

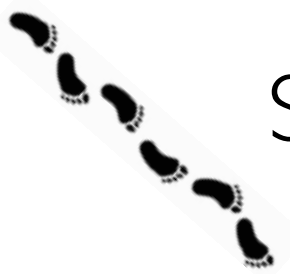
```
if (condition1a and condition1b) {  
    some code block1;  
}  
elseif (condition2a or condition2b) {  
    some alternative codeblock2;  
}  
elseif (condition3) {  
    some alternative codeblock3;  
}  
else {  
    last possible codeblock;  
}
```

# Adding complexity to the code: Loops


Let's now focus on this code block! Reading in only one line from a file is not really satisfying

```
#!/usr/bin/perl -w
use strict;
print "Please give me a filename\n";
my $filename = <STDIN>;
chomp $filename;
if (-e "$filename"){
    open (IN, "$filename") or die "could not find $filename";
    my $firstline = <IN>;
    close IN;
    my ($firstword) = split //, $firstline;
    print "First word of first line in $filename is $firstword\n";
}
else {
    print "I could not find the file $filename. I will exit\n";
    exit;
}
```

So far, we have executed each line of code zero or one time. Loops facilitate the repeated execution of code blocks.



# Some general and obvious things to consider



- What is my problem? The more precise you can formulate it the better!
- What is my problem? The more abstract you can formulate it the better!
- How can I formulate the problem solution procedure, i.e. the algorithm?
- What does my input look like? (Are you sure?)
- How should my output look like?
- What can go wrong and how do I capture errors?

# *while* loops

A while loop executes a code block as long as the conditional statement is TRUE!

```
if (-e "$filename"){  
    open (IN, "$filename") or die "could not find $filename";  
→ while (my $firstline = <IN>) {  
    my ($firstword) = split //, $firstline;  
    print "First word of line in $filename is $firstword\n";  
→ }  
}
```

Remember, <IN> retrieves a line from a filehandle. If issued repeatedly you will walk line by line through the text. If the end of the text is reached, <IN> will return FALSE!



# *for loops*

In *for* loops you can specify the number of iterations!

However, the variable assignment has now to be moved into the loop!

```
if (-e "$filename"){  
    open (IN, "$filename") or die "could not find $filename";  
→ for (my $i = 0; $i < 100; $i++) {  
    → my $firstline = <IN>;  
    my ($firstword) = split //, $firstline;  
    print "First word of line $i in $filename is $firstword\n";  
→ }  
}
```

# *for loops*

initialisation  
of the run  
index \$i

```
if (-e "$filename"){  
    open (IN, "$filename") or die "could not find $filename";  
    → for (my $i = 0; $i < 100; $i++) {  
        → my $firstline = <IN>;  
        my ($firstword) = split //, $firstline;  
        print "First word of line $i in $filename is $firstword\n";  
    }  
}
```

In *for* loops you  
can specify the  
number of  
iterations!

However, the  
variable  
assignment has  
now to be moved  
into the loop!

# *for loops*

initialisation  
of the run  
index \$i

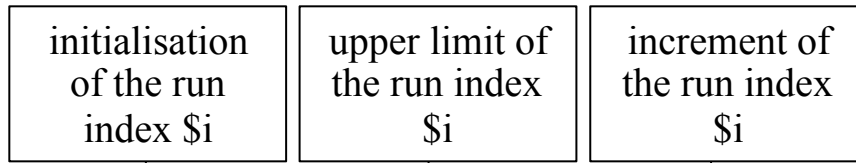
upper limit of  
the run index  
\$i

```
if (-e "$filename") {  
    open (IN, "$filename") or die "could not find $filename";  
    → for (my $i = 0; $i < 100; $i++) {  
        → my $firstline = <IN>;  
        my ($firstword) = split //, $firstline;  
        print "First word of line $i in $filename is $firstword\n";  
    }  
}
```

In *for* loops you  
can specify the  
number of  
iterations!

However, the  
variable  
assignment has  
now to be moved  
into the loop!

# *for loops*

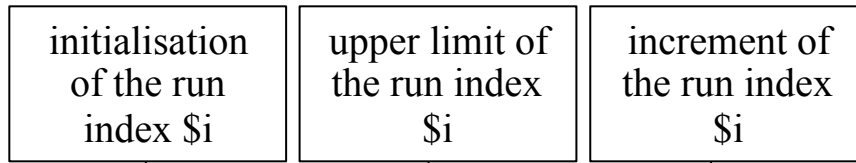


```
if (-e "$filename") {  
    open (IN, "$filename") or die "could not find $filename";  
    → for (my $i = 0; $i < 100; $i++) {  
        → my $firstline = <IN>;  
        my ($firstword) = split //, $firstline;  
        print "First word of line $i in $filename is $firstword\n";  
    }  
}
```

In *for* loops you can specify the number of iterations!

However, the variable assignment has now to be moved into the loop!

# *for loops*



```
if (-e "$filename") {  
    open (IN, "$filename") or die "could not find $filename";  
    → for (my $i = 0; $i < 100; $i++) {  
        → my $firstline = <IN>;  
        my ($firstword) = split //, $firstline;  
        print "First word of line $i in $filename is $firstword\n";  
    → }  
}
```

In *for* loops you can specify the number of iterations!

However, the variable assignment has now to be moved into the loop!

Our for loop runs now exactly 100 times. Thus, we can never be caught in an infinite loop!

Note, if you increment the index \$i by 2 each time, you will pass only 50 times through the loop!

# Revisiting the *split* function

The *split* function splits a string at the specified character:

```
my $str = "hi you how are you?";  
my ($first, $second) = split //, $str; # splits the string at each white space  
print "First word is '$first', second word is '$second'\n";
```

First word is 'hi', second word is 'you', but how about the rest?

We need another type of variables that can hold a list of scalar values: ARRAYS

# Lists and arrays

A **list** is an ordered set of scalar values:

```
(3,2,1,"fred")
```

An **array** is a variable that holds a list:

```
my @arr = (3,2,1,"fred");
```

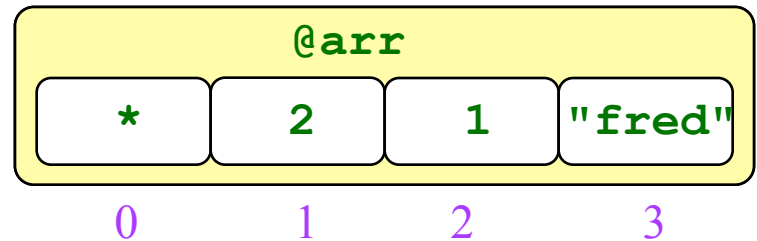
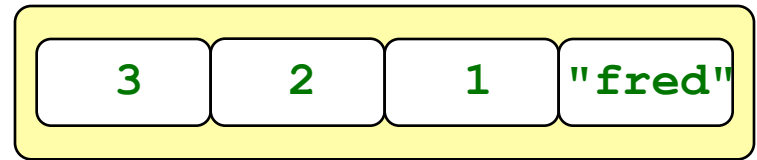
```
print @arr;      3 2 | fred
```

You can access an individual **array element**:

```
print $arr[1];   2
```

```
$arr[0] = "*";
```

```
print @arr;      * 2 | fred
```



# Manipulating arrays – push & pop

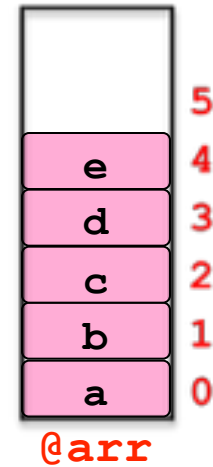




# Manipulating arrays – push & pop

```
my @arr = ('a','b','c','d','e');
```

```
print @arr;      abcde
```



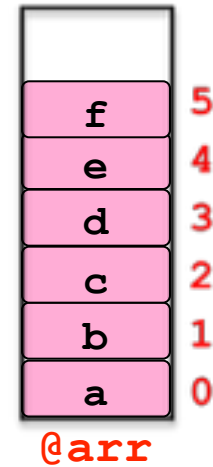
# Manipulating arrays – push & pop

```
my @arr = ('a','b','c','d','e');
```

```
print @arr;      abcde
```

```
push(@arr,'f');
```

```
print @arr;      abcdef
```



# Manipulating arrays – push & pop

```
my @arr = ('a','b','c','d','e');
```

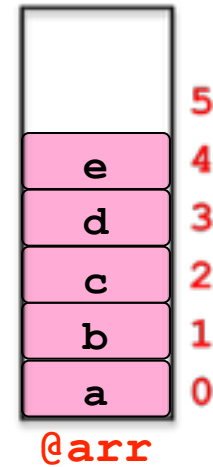
```
print @arr;      abcde
```

```
push(@arr,'f');
```

```
print @arr;      abcdef
```

-----

```
my @arr = ('a','b','c','d','e');
```



# Manipulating arrays – push & pop

```
my @arr = ('a','b','c','d','e');
```

```
print @arr;      abcde
```

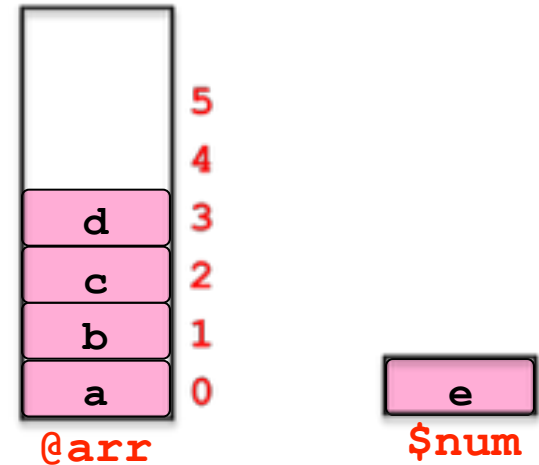
```
push(@arr,'f');
```

```
print @arr;      abcdef
```

-----

```
my @arr = ('a','b','c','d','e');
```

```
my $num = pop(@arr);
```



# Manipulating arrays – push & pop

```
my @arr = ('a','b','c','d','e');
```

```
print @arr;      abcde
```

```
push(@arr,'f');
```

```
print @arr;      abcdef
```

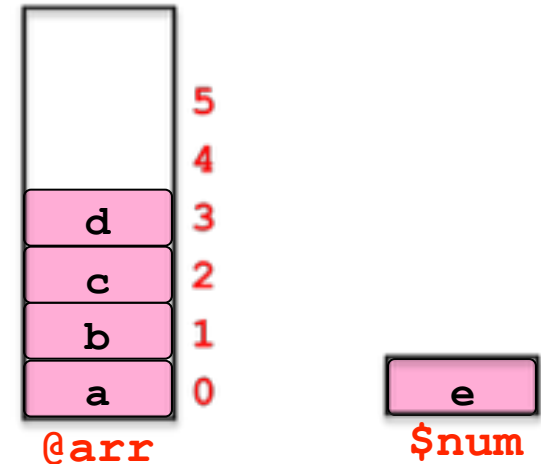
-----

```
my @arr = ('a','b','c','d','e');
```

```
my $num = pop(@arr);
```

```
print $num;      e
```

```
print @arr;      abcd
```

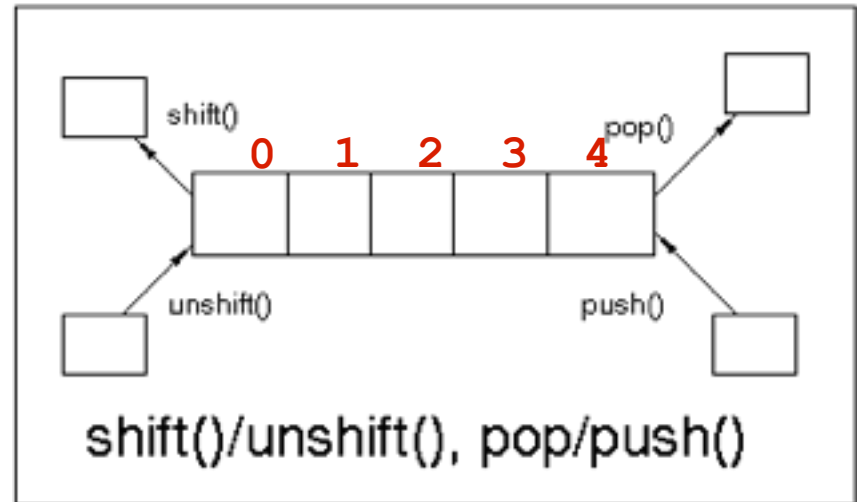


# shift & unshift

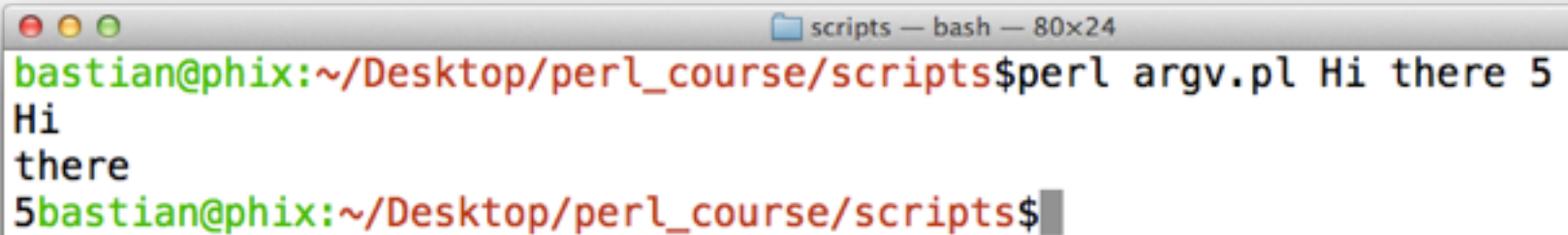
```
my @arr = ('a','b','c');  
print @arr;          abc  
unshift(@arr,'?');  
print @arr;          ?abc
```

-----

```
my @arr = ('a','b','c');  
my $num = shift(@arr);  
print $num;          a  
print @arr;          bc
```



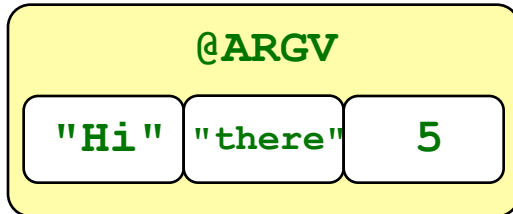
@ARGV



A terminal window titled "scripts — bash — 80x24" showing a command execution. The prompt is "bastian@phix:~/Desktop/perl\_course/scripts\$". The command "perl argv.pl Hi there 5" is entered. The output is "Hi" on the first line and "there" on the second line. The prompt "5bastian@phix:~/Desktop/perl\_course/scripts\$" is shown on the third line, with a cursor at the end.

```
bastian@phix:~/Desktop/perl_course/scripts$ perl argv.pl Hi there 5
Hi
there
5bastian@phix:~/Desktop/perl_course/scripts$
```

# @ARGV

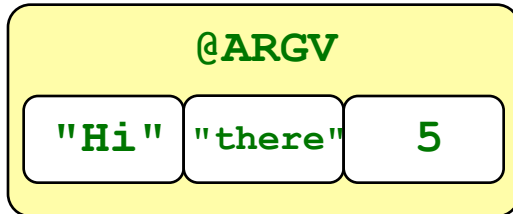


```
scripts — bash — 80x24
bastian@phix:~/Desktop/perl_course/scripts$ perl argv.pl Hi there 5
Hi
there
5bastian@phix:~/Desktop/perl_course/scripts$
```



# @ARGV

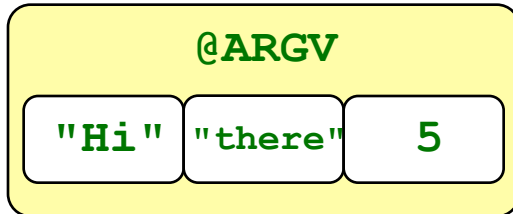
It is possible to pass arguments to Perl from the command line. These Command-line arguments are stored in an **array created automatically** named **@ARGV**:



```
scripts — bash — 80x24
bastian@phix:~/Desktop/perl_course/scripts$ perl argv.pl Hi there 5
Hi
there
5bastian@phix:~/Desktop/perl_course/scripts$
```

# @ARGV

It is possible to pass arguments to Perl from the command line. These Command-line arguments are stored in an **array created automatically** named **@ARGV**:



```
scripts — bash — 80x24
bastian@phix:~/Desktop/perl_course/scripts$ perl argv.pl Hi there 5
Hi
there
5bastian@phix:~/Desktop/perl_course/scripts$
```

# @ARGV

It is possible to pass arguments to Perl from the command line. These Command-line arguments are stored in an **array created automatically** named **@ARGV**:

Consider the following example script: print\_input.pl

```
#!/usr/bin/perl -w
use strict;
my $joinedArr = join("\n",@ARGV);
print $joinedArr;
print $ARGV[0] . "\n";
```

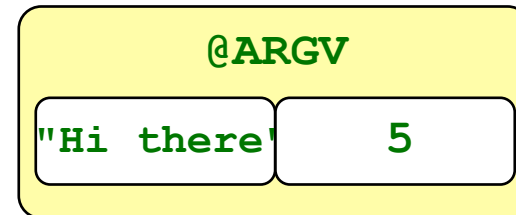
<ingo> print\_input.pl "Hi there" 5

Hi there

5

Hi there

<ingo>





# Introduction into Text Processing & Data Analysis with PERL - Day 2

Hashes, pattern matching, sub-routines

# Assigning values to variables

- `my $stringVar = 'test';`
- `my $numVar = 7;`
- `my @anyArr = (1, 8, 'tedious');`
- `my @anyArr2 = ($stringVar, $numVar, @anyArr);`
- `my $firstEntry = shift(@anyArr2);`
- `my $lastEntry = pop(@anyArr2);`
- `push @anyArr, "new entry at the end";`
- `unshift @anyArr, "new entry at the beginning";`
- `my @splitArr = split //, $anystring;`
- `my $anyString = join " ", @splitArr;`

# Hashes

(associative arrays)



# Variable types in PERL

## Scalar

**\$number**  
-3.54

**\$string**  
"hi\n"

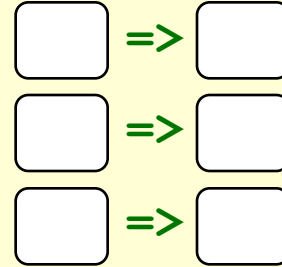
## Array

**@array**



## Hash

**%hash**



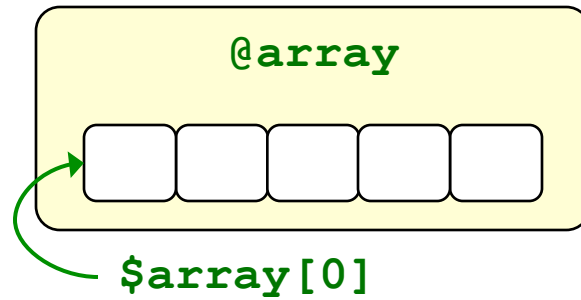
# Variable types in PERL

## Scalar

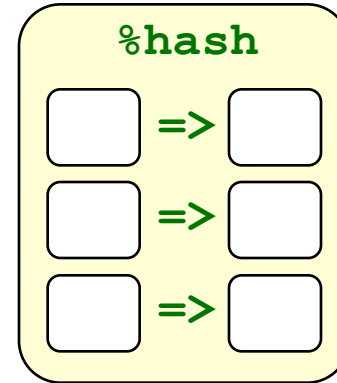
**\$number**  
-3.54

**\$string**  
"hi\n"

## Array



## Hash





# Variable types in PERL

## Scalar

`$number`  
`-3.54`

`$string`  
`"hi\n"`

## Array

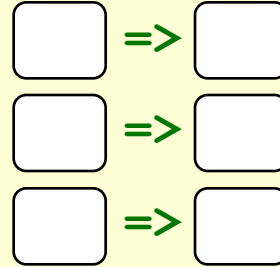
`@array`



`$array[0]`

## Hash

`%hash`



`$hash{key}`

# Hash Motivation

Let's say we want to create a phone book ...

**Enter a name that will be added to the phone book:**

**Dudi**

**Enter a phone number:**

**6409245**

**Enter a name that will be added to the phone book:**

**Dudu**

**Enter a phone number:**

**6407693**

Hash – an associative array

# Hash – an associative array

An *associative array* of the phone book suggested in the first slide  
(we will see a more elaborated version later on):

# Hash – an associative array

An **associative array** of the phone book suggested in the first slide (we will see a more elaborated version later on):

- **# Declare. Note, a hash variable always starts with a ‘%’**  
**my %phoneBook;**

# Hash – an associative array

An *associative array* of the phone book suggested in the first slide (we will see a more elaborated version later on):

- **# Declare. Note, a hash variable always starts with a ‘%’**

```
my %phoneBook;
```

- **# Initialize**

```
%phoneBook = ("Dudi"=>9245, "Dudu"=>7693);
```

# Hash – an associative array

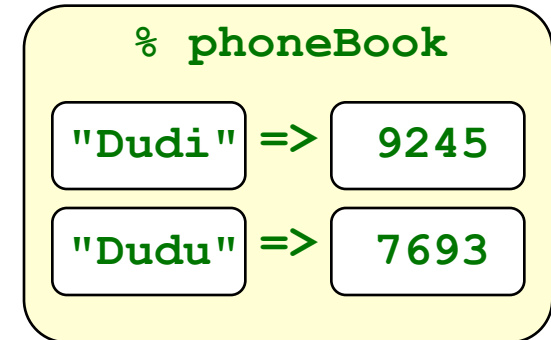
An *associative array* of the phone book suggested in the first slide (we will see a more elaborated version later on):

- **# Declare. Note, a hash variable always starts with a ‘%’**

```
my %phoneBook;
```

- **# Initialize**

```
%phoneBook = ("Dudi"=>9245, "Dudu"=>7693);
```



# Hash – an associative array

An **associative array** of the phone book suggested in the first slide (we will see a more elaborated version later on):

- **# Declare. Note, a hash variable always starts with a ‘%’**

```
my %phoneBook;
```

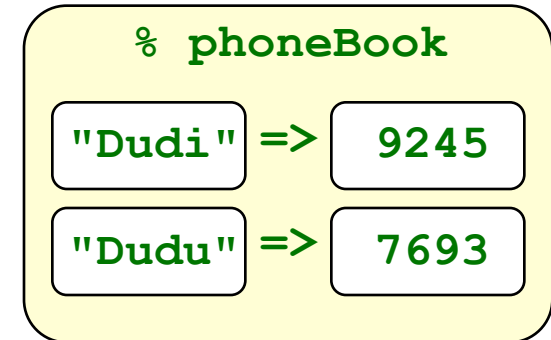
- **# Initialize**

```
%phoneBook = ("Dudi"=>9245, "Dudu"=>7693);
```

- **# Update**

```
$phoneBook{"Dudi"} = 7777;
```

```
$phoneBook{"Dudu"} = 4711;
```





# Hash – an associative array

An **associative array** of the phone book suggested in the first slide (we will see a more elaborated version later on):

- **# Declare. Note, a hash variable always starts with a ‘%’**

```
my %phoneBook;
```

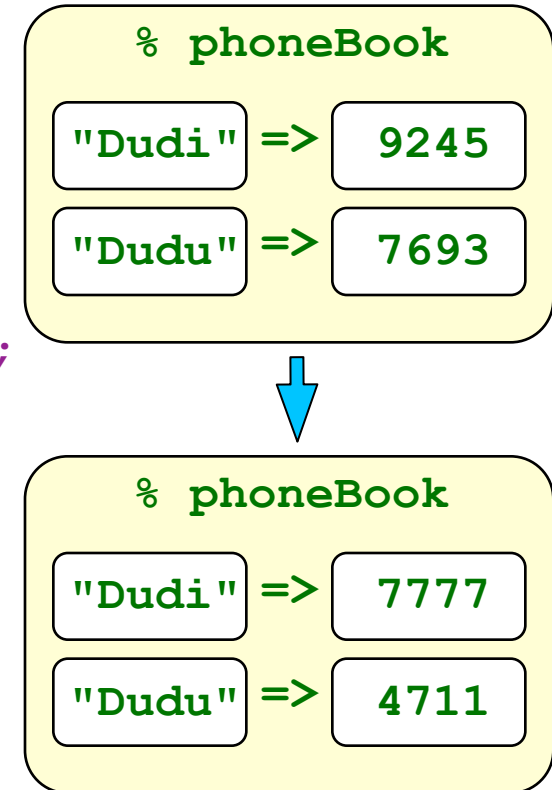
- **# Initialize**

```
%phoneBook = ("Dudi"=>9245, "Dudu"=>7693);
```

- **# Update**

```
$phoneBook{"Dudi"} = 7777;
```

```
$phoneBook{"Dudu"} = 4711;
```



# Hash – an associative array

An **associative array** of the phone book suggested in the first slide (we will see a more elaborated version later on):

- **# Declare. Note, a hash variable always starts with a ‘%’**

```
my %phoneBook;
```

- **# Initialize**

```
%phoneBook = ("Dudi"=>9245, "Dudu"=>7693);
```

- **# Update**

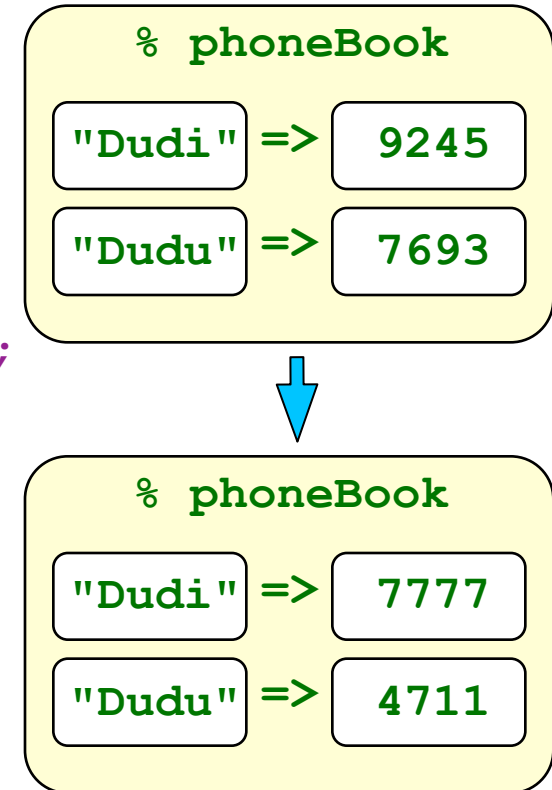
```
$phoneBook{"Dudi"} = 7777;
```

```
$phoneBook{"Dudu"} = 4711;
```

- **# Fetching the value**

```
print $phoneBook{"Dudi"};
```

**9245**



# Hash – an associative array

An **associative array** of the phone book suggested in the first slide (we will see a more elaborated version later on):

- **# Declare. Note, a hash variable always starts with a ‘%’**

```
my %phoneBook;
```

- **# Initialize**

```
%phoneBook = ("Dudi"=>9245, "Dudu"=>7693);
```

- **# Update**

```
$phoneBook{"Dudi"} = 7777;
```

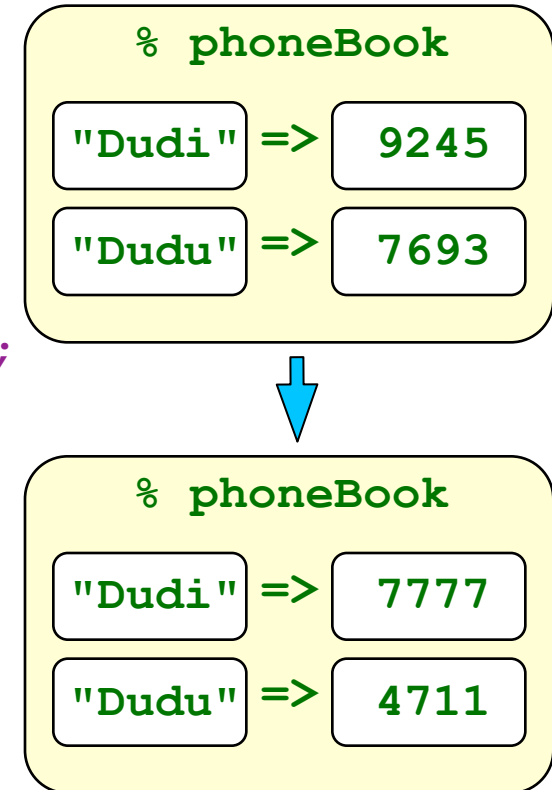
```
$phoneBook{"Dudu"} = 4711;
```

- **# Fetching the value**

```
print $phoneBook{"Dudi"};
```

9245

Note the curly braces!



# Hash – an associative array

**Note, modifying an existing value, and adding a new key=>value pair have the same syntax!**

**# modifying an existing entry**

**\$phoneBook{"Dudi"} = 7766;** (modifying an existing value)

**# adding a key=>value pair**

**\$phoneBook{"Viri"} = "z";** (adding a new key-value pair)

**# Delete a key=>value pair**

**delete(\$phoneBook{"Viri"});**

**# You can ask whether a certain key exists in a hash:**

**if (exists \$phoneBook{"Viri"} )...**

**# You can ask whether a certain value has been defined in a hash:**

**if (defined \$phoneBook{"Viri"})...**

**# Reset the hash (to an empty one):**

**%phoneBook = ();**

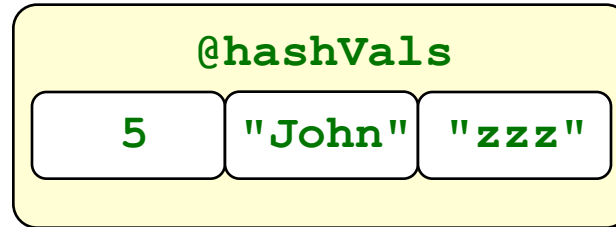
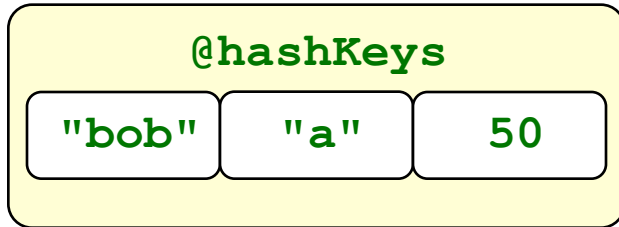
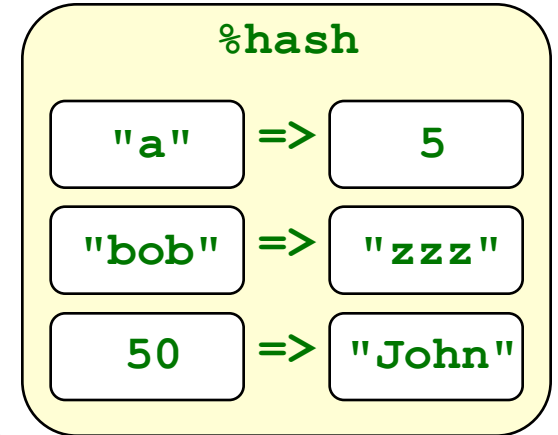
# Iterating over hash elements

# It is possible to get a list of all the keys in %hash

```
my @hashKeys = keys(%hash);
```

# Similarly you can get an array of the values in %hash

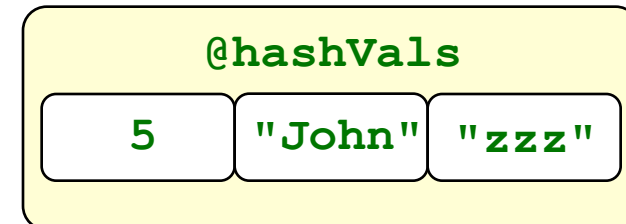
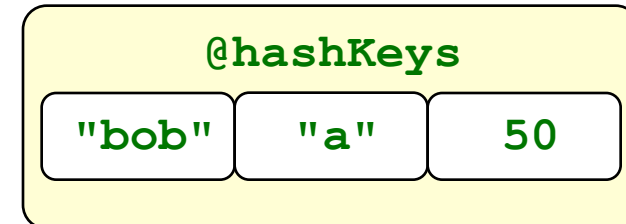
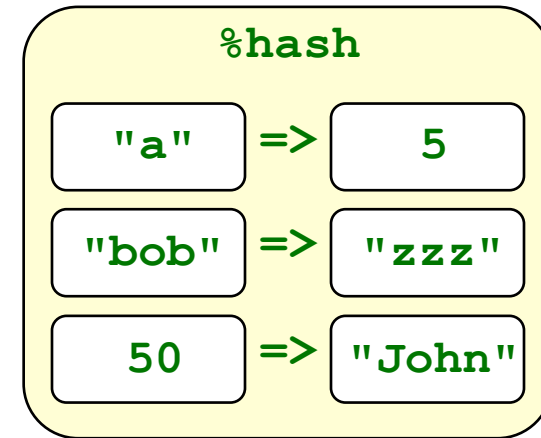
```
my @hashVals = values(%hash);
```



# Iterating over hash elements

```
my @hashKeys = keys(%hash);  
for (my $i=0; $i < @hashKeys; $i++) {  
    print "The key is $hashKeys[$i]\n";  
    print "The value is $hash{$hashKeys[$i]}\n";  
}
```

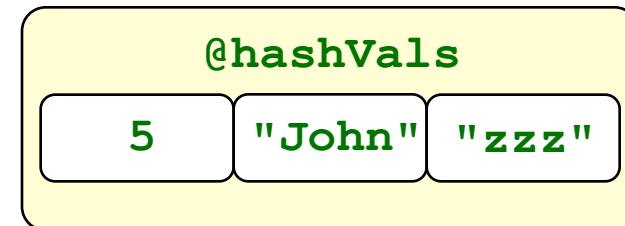
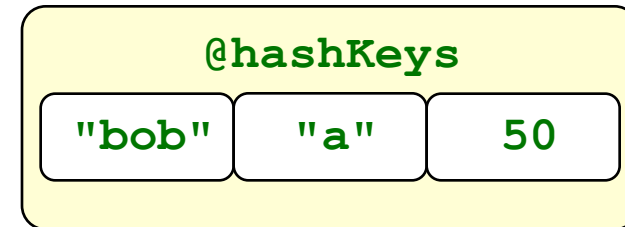
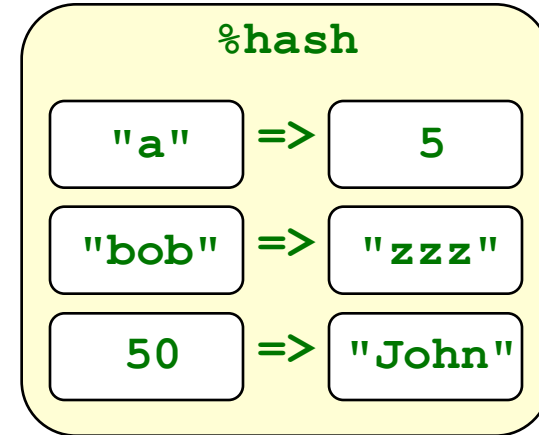
```
The key is bob  
The value is zzz  
The key is a  
The value is 5  
The key is 50  
The value is John
```



# Iterating over hash elements

**Note:** The elements are given in an **arbitrary order**,  
so if you want a certain order use `sort`:

```
my @hashKeys = keys(%hash);  
my @sortedHK = sort(@hashKeys);  
  
for (my $i=0; $i < @sortedHK; $i++) {  
    print "The key is $sortedHK[$i]\n";  
    print "The value is $hash{$sortedHK[$i]}\n";  
}
```



# Pattern matching





# Pattern matching

We often want to find a certain piece of information within the file, for example:

# Pattern matching

We often want to find a certain piece of information within the file, for example:

1. Extract GI numbers or accessions from Fasta

```
>gi|16127995|ref|NP_414542.1| thr operon ...  
>gi|145698229|ref|YP_001165309.1| hypothetical ...  
>gi|90111153|ref|NP_415149.4| citrate ...
```

# Pattern matching

We often want to find a certain piece of information within the file, for example:

1. Extract GI numbers or accessions from Fasta

```
>gi|16127995|ref|NP_414542.1| thr operon ...  
>gi|145698229|ref|YP_001165309.1| hypothetical ...  
>gi|90111153|ref|NP_415149.4| citrate ...
```

```
CDS 1542..2033  
CDS complement(3844..5180)
```

# Pattern matching

We often want to find a certain piece of information within the file, for example:

1. Extract GI numbers or accessions from Fasta

```
>gi|16127995|ref|NP_414542.1| thr operon ...  
>gi|145698229|ref|YP_001165309.1| hypothetical ...  
>gi|90111153|ref|NP_415149.4| citrate ...
```

2. Extract the coordinates of all open reading frames from the annotation of a genome

```
CDS 1542..2033  
CDS complement(3844..5180)
```

```
Sequences producing significant alignments:  
ref|NT_039621.4|Mm15_39661_34 Mus musculus chromosome 15 genomic... 186 1e-45  
ref|NT_039353.4|Mm6_39393_34 Mus musculus chromosome 6 genomic c... 38 0.71  
ref|NT_039477.4|Mm9_39517_34 Mus musculus chromosome 9 genomic c... 36 2.8
```

# Pattern matching

We often want to find a certain piece of information within the file, for example:

1. Extract GI numbers or accessions from Fasta

```
>gi|16127995|ref|NP_414542.1| thr operon ...  
>gi|145698229|ref|YP_001165309.1| hypothetical ...  
>gi|90111153|ref|NP_415149.4| citrate ...
```

2. Extract the coordinates of all open reading frames from the annotation of a genome

```
CDS 1542..2033  
CDS complement(3844..5180)
```

```
Sequences producing significant alignments:
ref|NT_039621.4|Mm15_39661_34 Mus musculus chromosome 15 genomic... 186 1e-45
ref|NT_039353.4|Mm6_39393_34 Mus musculus chromosome 6 genomic c... 38 0.71
ref|NT_039477.4|Mm9_39517_34 Mus musculus chromosome 9 genomic c... 36 2.8
```

All these examples are patterns in the text.

# Pattern matching

# Pattern matching

Finding a **sub-string** (match) **somewhere** in a string:

# Pattern matching

Finding a **sub-string** (match) **somewhere** in a string:



# Pattern matching

Finding a **sub-string** (match) **somewhere** in a string:

**if (\$line =~ m/he/)** ... remember to use slash (/) and not back-slash

# Pattern matching

Finding a **sub-string** (match) **somewhere** in a string:

**if (\$line =~ m/he/)** ... remember to use slash (/) and not back-slash

Will be true for "hello" and for "the cat" but not for "good bye" or "Hercules".

# Pattern matching

Finding a **sub-string** (match) **somewhere** in a string:

**if (\$line =~ m/he/)** ... remember to use slash (/) and not back-slash

Will be true for “**hello**” and for “**the cat**” but not for “**good bye**” or “**Hercules**”.

You can **ignore case** of letters by adding an “i” after the pattern:

# Pattern matching

Finding a **sub-string** (match) **somewhere** in a string:

```
if ($line =~ m/he/) ...
```

 remember to use slash (/) and not back-slash

Will be true for “**hello**” and for “**the cat**” but not for “**good bye**” or “**Hercules**”.

You can **ignore case** of letters by adding an “**i**” after the pattern:

```
m/he/i
```

# Pattern matching

Finding a **sub-string** (match) **somewhere** in a string:

**if (\$line =~ m/he/)** ... remember to use slash (/) and not back-slash

Will be true for “**hello**” and for “**the cat**” but not for “**good bye**” or “**Hercules**”.

You can **ignore case** of letters by adding an “i” after the pattern:

**m/he/i**

(matches for “**the**”, “**Hello**”, “**Hercules**” and “**hEHD**”)

# Enhancing pattern matching using regular expressions

## Single-character patterns

# Enhancing pattern matching using regular expressions

## Single-character patterns

`m/./` Matches *any character* (except “\n”)

# Enhancing pattern matching using regular expressions

## Single-character patterns

`m/./` Matches any character (except “\n”)



# Enhancing pattern matching using regular expressions

## Single-character patterns

`m/./` Matches any character (except “\n”)

You can also match one of a group of characters:

# Enhancing pattern matching using regular expressions

## Single-character patterns

`m/./` Matches **any character** (except “\n”)

You can also match **one of a group** of characters:

`m/[atcg]/` Matches “a” or “t” or “c” or “g”

`m/[a-d]/` Matches “a” though “d” (a, b, c or d)

# Enhancing pattern matching using regular expressions

## Single-character patterns

`m/./` Matches **any character** (except “\n”)

You can also match **one of a group** of characters:

<code>m/[atcg]/</code>	Matches “a” or “t” or “c” or “g”
<code>m/[a-d]/</code>	Matches “a” through “d” (a, b, c or d)
<code>m/[a-zA-Z]/</code>	Matches any letter
<code>m/[a-zA-Z0-9]/</code>	Matches any letter or digit
<code>m/[a-zA-Z0-9_]/</code>	Matches any letter or digit or an underscore

# Enhancing pattern matching using regular expressions

## Single-character patterns

`m/./` Matches **any character** (except “\n”)

You can also match **one of a group** of characters:

<code>m/[atcg]/</code>	Matches “a” or “t” or “c” or “g”
<code>m/[a-d]/</code>	Matches “a” though “d” (a, b, c or d)
<code>m/[a-zA-Z]/</code>	Matches any letter
<code>m/[a-zA-Z0-9]/</code>	Matches any letter or digit
<code>m/[a-zA-Z0-9_]/</code>	Matches any letter or digit or an underscore
<code>m/[^[atcg]/</code>	Matches any character <b>except</b> “a” or “t” or “c” or “g”

# Single-character patterns

For example:

```
if ($line =~ m/TATAA[AT]/)
```

Will this be true for?

**TATTAA** ✗

**TATAATA** ✓

**CTATAATAGCTAGGCGCATG** ✓

# Single-character patterns

Perl provides predefined **character classes**:

`\d` a digit (same as: `[0-9]`)

`\w` a “word” character (same as: `[a-zA-Z0-9_]`)

`\s` a space character (same as: `[\t\n\r\f]`)

And their **negatives**:

`\D` anything but a digit

`\W` anything but a word char

`\S` anything but a space char

For example:

```
if ($line =~ m/class\.ex\d\.S/)
```

```
class.ex3.1.pl
```

```
class.ex3.
```

```
my class.ex8.(old)
```

# Single-character patterns

Perl provides predefined **character classes**:

`\d` a digit (same as: `[0-9]`)

`\w` a “word” character (same as: `[a-zA-Z0-9_]`)

`\s` a space character (same as: `[\t\n\r\f]`)

And their **negatives**:

`\D` anything but a digit

`\W` anything but a word char

`\S` anything but a space char

For example:

```
if ($line =~ m/class\d\S/)
```

`class.ex3.1.pl`



`class.ex3.`



`my class.ex8.(old)`



# Repetitive patterns



# Repetitive patterns

? means zero or one repetitions of what's before it:

`m/ab?c/` Matches “ac” or “abc”

# Repetitive patterns

? means zero or one repetitions of what's before it:

`m/ab?c/` Matches “ac” or “abc”

# Repetitive patterns

? means zero or one repetitions of what's before it:

**m/ab?c/** Matches “ac” or “abc”

+ means one or more repetitions of what's before it:

**m/ab+c/** Matches “abc” ; “abbbbc” but not “ac”

# Repetitive patterns

? means zero or one repetitions of what's before it:

**m/ab?c/** Matches “ac” or “abc”

+ means one or more repetitions of what's before it:

**m/ab+c/** Matches “abc” ; “abbbbc” but not “ac”

# Repetitive patterns

? means zero or one repetitions of what's before it:

**m/ab?c/** Matches “ac” or “abc”

+ means one or more repetitions of what's before it:

**m/ab+c/** Matches “abc” ; “abbbbc” but not “ac”

A pattern followed by \* means zero or more repetitions of that pattern:

**m/ab\*c/** Matches “abc” ; “ac” ; “abbbbc”

# Repetitive patterns

? means zero or one repetitions of what's before it:

**m/ab?c/** Matches “ac” or “abc”

+ means one or more repetitions of what's before it:

**m/ab+c/** Matches “abc” ; “abbbbc” but not “ac”

A pattern followed by \* means zero or more repetitions of that pattern:

**m/ab\*c/** Matches “abc” ; “ac” ; “abbbbc”

# Repetitive patterns

? means zero or one repetitions of what's before it:

**m/ab?c/** Matches “ac” or “abc”

+ means one or more repetitions of what's before it:

**m/ab+c/** Matches “abc” ; “abbbbc” but not “ac”

A pattern followed by \* means zero or more repetitions of that pattern:

**m/ab\*c/** Matches “abc” ; “ac” ; “abbbbc”

Generally – use { } for a certain number of repetitions, or a range:

**m/ab{3}c/** Matches “abbbbc”

**m/ab{3,6}c/** Matches “a”, 3-6 times “b” and then “c”

# Repetitive patterns

? means zero or one repetitions of what's before it:

**m/ab?c/** Matches “ac” or “abc”

+ means one or more repetitions of what's before it:

**m/ab+c/** Matches “abc” ; “abbbbc” but not “ac”

A pattern followed by \* means zero or more repetitions of that pattern:

**m/ab\*c/** Matches “abc” ; “ac” ; “abbbbc”

Generally – use { } for a certain number of repetitions, or a range:

**m/ab{3}c/** Matches “abbbbc”

**m/ab{3,6}c/** Matches “a”, 3-6 times “b” and then “c”

**m/ab{3,}c/** Matches “a”, “b” 3 times or more and then “c”



# Repetitive patterns

? means zero or one repetitions of what's before it:

**m/ab?c/** Matches “ac” or “abc”

+ means one or more repetitions of what's before it:

**m/ab+c/** Matches “abc” ; “abbbc” but not “ac”

A pattern followed by \* means zero or more repetitions of that pattern:

**m/ab\*c/** Matches “abc” ; “ac” ; “abbbc”

Generally – use { } for a certain number of repetitions, or a range:

**m/ab{3}c/** Matches “abbbc”

**m/ab{3,6}c/** Matches “a”, 3-6 times “b” and then “c”

**m/ab{3,}c/** Matches “a”, “b” 3 times or more and then “c”

# Repetitive patterns

? means zero or one repetitions of what's before it:

`m/ab?c/` Matches “ac” or “abc”

+ means one or more repetitions of what's before it:

`m/ab+c/` Matches “abc” ; “abbbbc” but not “ac”

A pattern followed by \* means zero or more repetitions of that pattern:

`m/ab*c/` Matches “abc” ; “ac” ; “abbbbc”

Generally – use { } for a certain number of repetitions, or a range:

`m/ab{3}c/` Matches “abbbc”

`m/ab{3,6}c/` Matches “a”, 3-6 times “b” and then “c”

`m/ab{3,}c/` Matches “a”, “b” 3 times or more and then “c”

Use parentheses to mark more than one character for repetition:

`m/h(e1)*lo/` Matches “hello” ; “hlo” ; “helello”

# Repetitive patterns

For example:

```
if ($line =~ m/TATAA[AT][ATCG]{2,4}ATG/)
```

Will this be true for?

**TATAAAGAATG**



**ACTATAATAAAAATG**



**TATAATGATGTATAATATG**



# Example code

Consider the following code:

```
print "please enter a line...\n";
my $line = <STDIN>;
chomp($line);

if ($line =~ m/-?\d+/) {
    print "This line seems to contain a number...\n";
}
else {
    print "This is certainly not a number...\n";
}
```

## Example code

```
my $filename = "numbers.txt";
open(my $in, "$filename") or die "cannot open $filename $!";
my $line = <$in>;
while (defined $line) {
    chomp $line;
    if ($line =~ m/-?\d+/) {
        print "This line: '$line' seems to contain a number...\n";
    }
    else {
        print "This '$line' is certainly not a number...\n";
    }
    $line = <$in>;
}
```

# Substitute one pattern with another

Replacing a sub string (substitute):

```
$line = "the cat on the tree";
```

```
$line =~ s/he/hat/;
```

`$line` will be turned to “that cat on the tree”

To Replace all occurrences of a sub string add a “g” (for “globally”):

```
$line = "the cat on the tree";
```

```
$line =~ s/he/hat/g;
```

`$line` will be turned to “that cat on that tree”

## Enforce line start/end

To force the pattern to be at the beginning of the string add a “^”:

`m/^>/` Matches only strings that begin with a “>”

“\$” forces the end of string:

`m/.pl$/` Matches only strings that end with a “.pl”

**And together:**

`m/^\s*$/` Matches empty lines and all lines that contains only space characters.

Some examples



## Some examples

`m/\d+ (\.\d+)?/` Matches numbers that may contain a decimal point:

## Some examples

`m/\d+ (\.\d+)?/` Matches numbers that may contain a decimal point:  
“10”; “3.0”; “4.75” ...

## Some examples

`m/\d+(\.\d+)?/` Matches numbers that may contain a decimal point:  
“10”; “3.0”; “4.75” ...

`m/^NM_\d+/` Matches Genbank RefSeq accessions like “NM\_079608”

## Some examples

`m/\d+(\.\d+)?/` Matches numbers that may contain a decimal point:  
“10”; “3.0”; “4.75” ...

`m/^NM_\d+/` Matches Genbank RefSeq accessions like “NM\_079608”

# Extracting part of a pattern using special variables \$1, \$2, \$3...

We can extract parts of the pattern by parentheses:

```
$line = "1.35";  
if ($line =~ m/(\d+)\.(\d+)/ ) {  
    print "$1\n";    # 1  
    print "$2\n";    # 35  
}
```

## Extracting part of a pattern

We can extract parts of the string that matched parts of the pattern that are marked by parentheses:

```
my $line = "    CDS          87..1109";
if ($line =~ m/CDS\s+(\d+)\.\.(\d+)/ ) {
    print "regexp:$1,$2\n";           # regexp:87,1109
    my $start = $1;
    my $end = $2;
}
```

## Finding a pattern in an input file

Usually, we want to scan all lines of a file, and find lines with a specific pattern. E.g.:

```
my ($start,$end);
foreach $line (@lines) {
    if ($line =~ m/CDS\s+(\d+)\.\.(\d+)/ ) {
        $start = $1; $end = $2;
        ...
        ...
    }
}
```

# Extracting part of a pattern

We can extract parts of the string that matched parts of the pattern that are marked by parentheses. Suppose we want to match

both `$line = " CDS complement(4815..5888)";`

and `$line = " CDS 6087..8109";`

```
if ($line =~ m/CDS\s+(\complement\()?( (\d+)\.\.(\d+) )\)?/ )
{
    print "regexp:$1,$2,$3,$4.\n";
    $start = $3; $end = $4;
}
```

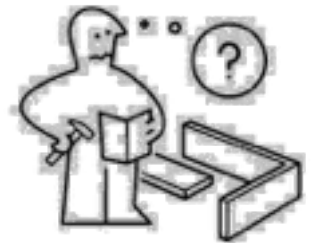
•When `$line = " CDS complement(4815..5888)";`

`regexp:complement(,4815..5888,4815,5888.`

•When `$line = " CDS 6087..8109";`

**Use of uninitialized value in concatenation...**

`regexp: ,6087..8109,6087,8109.`





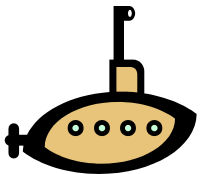


# Functions

A **function** is a portion of code that performs a specific task when called.

Functions we've met:

<code>\$newStr = substr (\$str,1,4);</code>	Takes a string and returns a sub-string
<code>@arr = split (/\t/, \$line);</code>	Splits a string into an array
<code>push (@arr, \$num);</code>	Pushes a scalar to the end of an array



# Functions

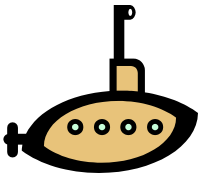
A **function** is a portion of code that performs a specific task when called.

Functions can have **arguments** and can **return values**:

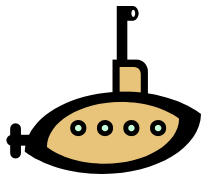
```
$start = substr ($str,1,4);
```

Return value:  
This function returns a string

Arguments:  
(STRING, OFFSET,  
LENGTH)



# Subroutines



A **subroutine** is a user-defined function.

```
sub SUB_NAME {  
    # Do something  
    ...  
}
```

Subroutines can be placed *anywhere* in the code, but are usually stacked *together* at the *beginning* or the *end* of the script.

```
sub printHello {  
    print "Hello World!\n";  
}  
sub bark {  
    print "Woof-woof\n";  
}  
sub reverseComplement {  
    my ($seq)=@_  
    $seq =~ tr/ACGTacgt/  
TGCAtgca/  
    $revSeq = reverse ($seq);  
    return $revSeq;  
}
```

# Subroutines

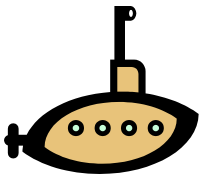
To *invoke* (execute) a subroutine we call it by its name with its arguments:

```
SUB_NAME (ARGUMENTS) ;
```

For example:

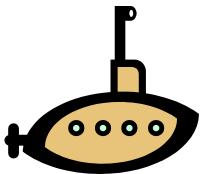
```
bark () ;  
    Woof-woof
```

```
my $seq = "GCAGTG" ;  
my $rev = reverseComplement ($seq) ;  
print $rev ;  
    CACTGC
```

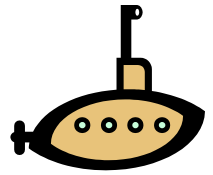


# Why use subroutines?

- Code in a subroutine is reusable as it has a defined input and returns a defined output.  
For example: a subroutine to produce the reverse-complement of a DNA sequence
- A subroutine can provide a general solution for different situations.  
For example: read a FASTA file
- Encapsulation: A well defined task can be outsourced in a subroutine, making the main script simpler and easier to read and understand.



# Why use subroutines? - Example



```
# Get the file name
```

```
my $filename = <STDIN>;  
chomp $filename;
```

```
# Read fasta sequence from file
```

```
open (my $in, "<", $filename) or die "Can't open file: '$filename' $!";
```

```
my $line = <$in>;
```

```
my $seq;
```

```
while (defined $line) {
```

```
    chomp $line;
```

```
    if ($line =~ m/^>/)
```

```
    {
```

```
        $line = <$in>;
```

```
    }
```

```
    else {
```

```
        $seq = $seq.$line;
```

```
        $line = <$in>;
```

```
    }
```

```
}
```

```
close ($in);
```

```
# Reverse complement the sequence
```

```
$seq =~ tr/ACGTacgt/TGCAtgca/;
```

```
$revSeq = reverse ($seq);
```

```
# Print the reverse complement in fasta format
```

```
my $i = 0;
```

```
while ((($i+1) * 70 < length ($revSeq)) {
```

```
    my $fastaLine = substr($revSeq, $i * 70, 70) .
```

```
    print $fastaLine."\n";
```

```
    $i++;
```

```
}
```

```
$fastaLine = substr($revSeq, $i * 70);
```

```
print $fastaLine."\n"
```

Much better than this



```
>gi|229577210|ref|NM_001743.4| Homo sapiens calmodulin 2 (CALM2), mRNA  
ATGGCTGACCAACTGACTGAAGAGCAGATTGCAGAATCAAAGAAGCTTTTTCACTATTTGACAAAGATG  
GTGATGGAACATAACAACAAAGGAATTGGGAACGTAAATGAGATCTCTTGGGCAGAATCCACAGAAGC  
AGAGTTACAGGACATGATTAATGAAGTAGATGCTGATGGTAAATGGCACAATTGACTTCCCTGAATTTCTG  
ACAATGATGGCAAGAAAAATGAAAGACACAGACAGTGAAGAAGAAATAGAGAAGCATTCCGTGTGTTT  
ATAAGGATGGCAATGGCTATATTAGTGTGTCAGAACTTCGCCATGTGATGACAAACCTGGAGAGAAGTT  
AACAGATGAAGAAGTTGATGAAATGATCAGGGAAGCAGATATTTGATGGTGTGTTCAAGTAACTATGAA  
GAGTTTGTACAAATGATGACAGCAAAGTGA
```

# Why use subroutines? - Example

```
my filename = $ARGV[0];
```

```
# Read fasta sequence from file
```

```
$seq = readFastaFile($fileName),
```

A general solution: works with different files

```
# Reverse complement the sequence
```

```
$revSeq = reverseComplement($seq),
```

Can be invoked from many points in the code

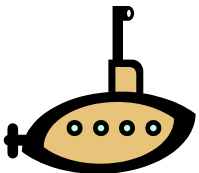
```
# Print the reverse complement in fasta format
```

```
printFasta($revSeq);
```

```
# Subroutines definition...
```

```
.....
```

And the program is beautiful

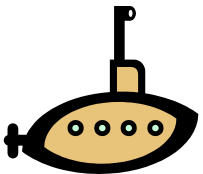




# Subroutine example

```
my $bart4today = "I do not have diplomatic immunity";  
bartFunc($bart4today ,100);
```

```
sub bartFunc {  
    my ($string, $times) = @_;  
    print $string x $times;  
}
```

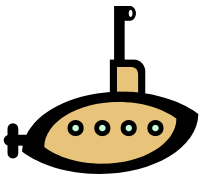


# Subroutine example

```
my $bart4today = "I do not have diplomatic immunity";  
bartFunc($bart4today ,100);
```

```
sub bartFunc {  
    my ($string, $times) = @_;  
    print $string x $times;  
}
```

We pass arguments to the subroutine



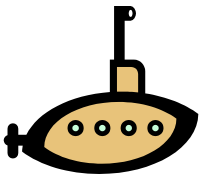
# Subroutine example

```
my $bart4today = "I do not have diplomatic immunity";  
bartFunc($bart4today ,100);
```

```
sub bartFunc {  
    my ($string, $times) = @_  
    print $string x $times;  
}
```

We pass arguments to the subroutine

Inside the subroutine block they are saved in the special array @\_



# Subroutine example

```
my $bart4today = "I do not have diplomatic immunity";  
bartFunc($bart4today ,100);
```

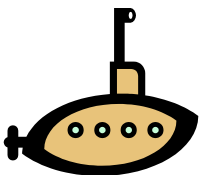
```
sub bartFunc {  
    my ($string, $times) = @_  
    print $string x $times;  
}
```

We pass arguments to the subroutine

Inside the subroutine block they are saved in the special array @\_  
\_

```
I do not have diplomatic immunity  
I do not have diplomatic immunity  
I do not have diplomatic immunity  
I do not have diplomatic immunity
```

...



# Subroutine example

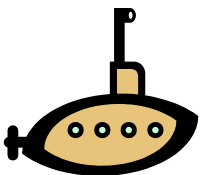
```
my $bart4today = "I do not have diplomatic immunity";  
bartFunc($bart4today ,100);
```

```
sub bartFunc {  
    my ($string, $times) = @_  
    print $string x $times;  
}
```

We pass arguments to the subroutine

Inside the subroutine block they are saved in the special array `@_`

```
I do not have diplomatic immunity  
I do not have diplomatic  
I do not have diplomatic  
I do not have diplomatic  
...
```



# Return value

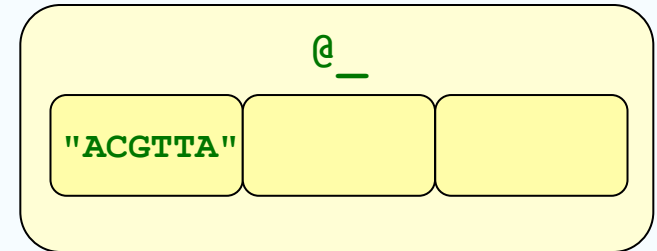
Returning return values:

```
$reversed = reverseComplement("ACGTTA");
```

\$reversed

"TAACGT"

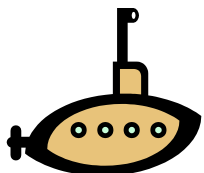
```
sub reverseComplement {  
  my ($seq) = @_;  
  $seq =~ tr/ACGT/TGCA/;  
  my $revSeq = reverse $seq;  
  return $revSeq;  
}
```



\$seq "TGCAAT"

\$revSeq "TAACGT"

The **return** statement ends the execution of the subroutine and returns a value



# Return value

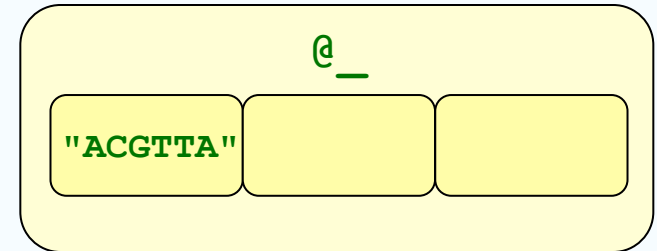
Returning **return** values:

```
$reversed = reverseComplement("ACGTTA");
```

`$reversed`

"TAACGT"

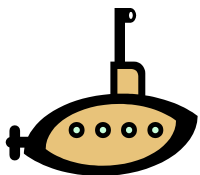
```
sub reverseComplement {  
  my ($seq) = @_;  
  $seq =~ tr/ACGT/TGCA/;  
  my $revSeq = reverse $seq;  
  return $revSeq;  
  print "I am the walrus!"  
}
```



`$seq` "TGCAAT"

`$revSeq` "TAACGT"

Anything after the **return** statement **will be ignored**



# Return list

```
my ($firstChar, $lastChar) = firstLastChar("Yellow");
```

```
print "First char: $firstChar, last one: $lastChar.\n";
```

```
    First char: Y, last one: w.
```

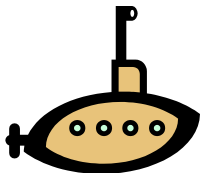
```
sub firstLastChar{
```

```
    my ($string) = @_;
```

```
    $string =~ m/^(.)*(.)$/;
```

```
    return ($1,$2);
```

```
}
```





# Return list

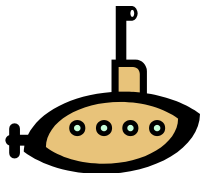
```
my ($firstChar, $lastChar) = firstLastChar("Yellow");
```

We pass an argument

```
print "First char: $firstChar, last one: $lastChar.\n";
```

```
    First char: Y, last one: w.
```

```
sub firstLastChar{  
    my ($string) = @_;  
    $string =~ m/^(.)*(.)$/;  
    return ($1,$2);  
}
```



# Return list

```
my ($firstChar, $lastChar) = firstLastChar("Yellow");
```

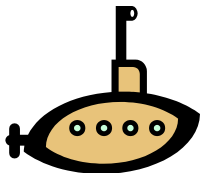
The return value is a list of two elements

We pass an argument

```
print "First char: $firstChar, last one: $lastChar.\n";
```

```
First char: Y, last one: w.
```

```
sub firstLastChar{  
    my ($string) = @_;  
    $string =~ m/^(.)*(.)$/;  
    return ($1,$2);  
}
```



# Return list

➔ `my ($firstChar, $lastChar) = firstLastChar("Yellow");`

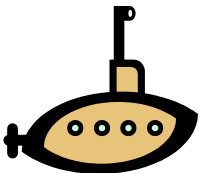
The return value is a list of two elements

We pass an argument

```
print "First char: $firstChar, last one: $lastChar.\n";
```

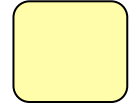
```
    First char: Y, last one: w.
```

```
sub firstLastChar{  
    my ($string) = @_;  
    $string =~ m/^(.)*(.)$/;  
    return ($1,$2);  
}
```

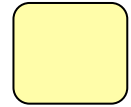


# Return list

\$firstChar



\$lastChar



 `my ($firstChar, $lastChar) = firstLastChar("Yellow");`

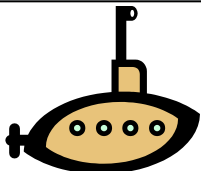
The return value is a list of two elements

We pass an argument

```
print "First char: $firstChar, last one: $lastChar.\n";
```

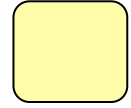
```
First char: Y, last one: w.
```

```
sub firstLastChar{  
    my ($string) = @_;  
    $string =~ m/^(.)*(.)$/;  
    return ($1,$2);  
}
```

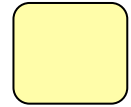


# Return list

\$firstChar



\$lastChar



 `my ($firstChar, $lastChar) = firstLastChar("Yellow");`

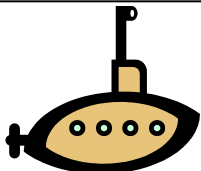
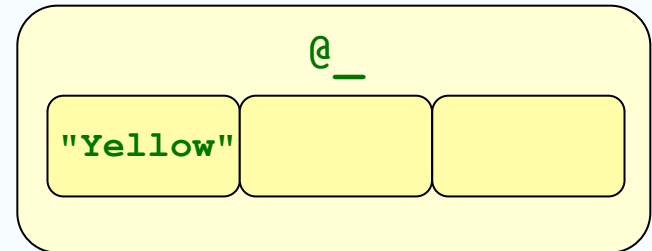
The return value is a list of two elements

We pass an argument

```
print "First char: $firstChar, last one: $lastChar.\n";
```

**First char: Y, last one: w.**

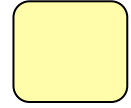
```
sub firstLastChar{  
    my ($string) = @_;  
    $string =~ m/^(.)*(.)$/;  
    return ($1,$2);  
}
```



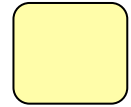
# Return list

```
my ($firstChar, $lastChar) = firstLastChar("Yellow");
```

\$firstChar



\$lastChar



The return value is a list of two elements

We pass an argument

```
print "First char: $firstChar, last one: $lastChar.\n";
```

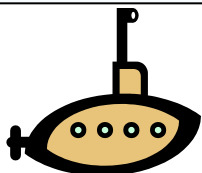
First char: Y, last one: w.

```
sub firstLastChar{
```

```
→ my ($string) = @_;  
  $string =~ m/^(.)*(.)/;  
  return ($1,$2);  
}
```

@\_

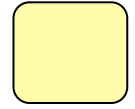
"Yellow"



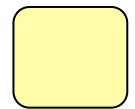
# Return list

```
my ($firstChar, $lastChar) = firstLastChar("Yellow");
```

\$firstChar



\$lastChar



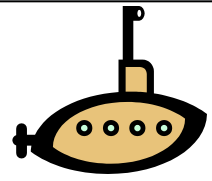
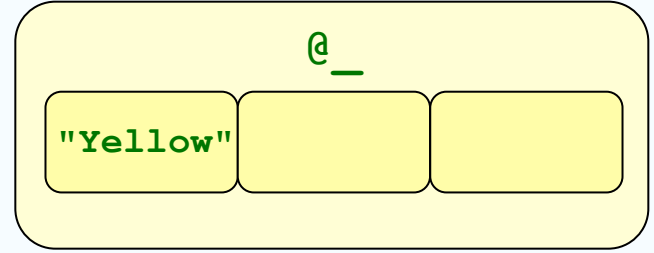
The return value is a list of two elements

We pass an argument

```
print "First char: $firstChar, last one: $lastChar.\n";
```

First char: Y, last one: w.

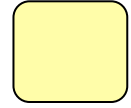
```
sub firstLastChar{  
    my ($string) = @_;  
    $string =~ m/^(.)*(.)$/;  
    return ($1,$2);  
}
```



# Return list

```
my ($firstChar, $lastChar) = firstLastChar("Yellow");
```

\$firstChar



\$lastChar



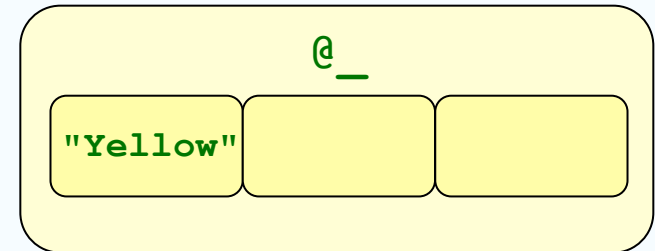
The return value is a list of two elements

We pass an argument

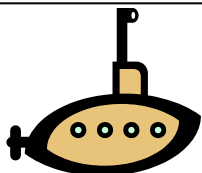
```
print "First char: $firstChar, last one: $lastChar.\n";
```

First char: Y, last one: w.

```
sub firstLastChar{  
    my ($string) = @_;  
    $string =~ m/^(.)*(.)$/;  
    return ($1,$2);  
}
```



\$string "Yellow"

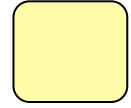




# Return list

```
my ($firstChar, $lastChar) = firstLastChar("Yellow");
```

\$firstChar



\$lastChar



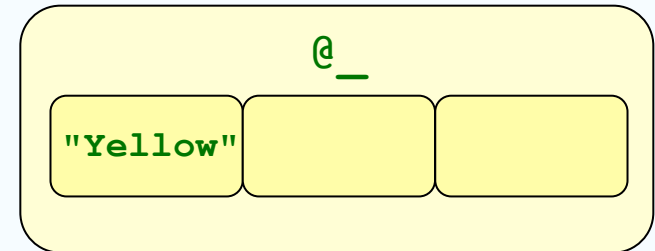
The return value is a list of two elements

We pass an argument

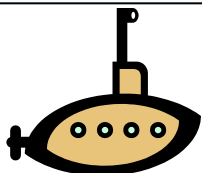
```
print "First char: $firstChar, last one: $lastChar.\n";
```

First char: Y, last one: w.

```
sub firstLastChar{  
  my ($string) = @_;  
  $string =~ m/^(.)*(.)$/;  
  return ($1,$2);  
}
```



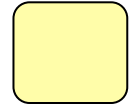
\$string "Yellow"



# Return list

```
my ($firstChar, $lastChar) = firstLastChar("Yellow");
```

\$firstChar



\$lastChar



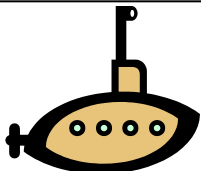
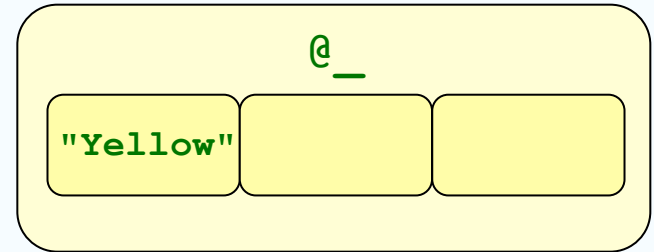
The return value is a list of two elements

We pass an argument

```
print "First char: $firstChar, last one: $lastChar.\n";
```

First char: Y, last one: w.

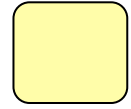
```
sub firstLastChar{  
  my ($string) = @_;  
  $string =~ m/^(.)*(.)$/;  
  return ($1,$2);  
}
```



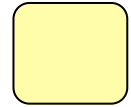
# Return list

```
my ($firstChar, $lastChar) = firstLastChar("Yellow");
```

\$firstChar



\$lastChar



The return value is a list of two elements

We pass an argument

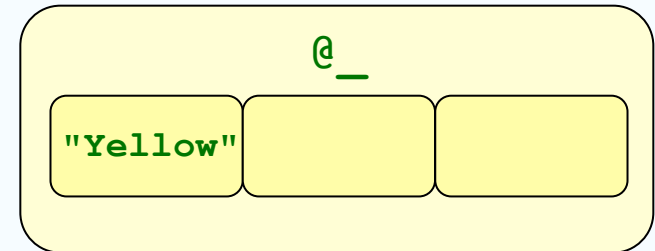
```
print "First char: $firstChar, last one: $lastChar.\n";
```

First char: Y, last one: w.

```
sub firstLastChar{  
  my ($string) = @_;  
  $string =~ m/^(.)*(.)$/;  
  return ($1,$2);  
}
```



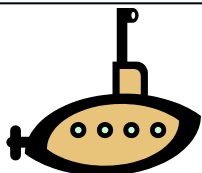
The subroutine returns a list of two elements.



\$string "Yellow"

\$1 "Y"

\$2 "w"



# Return list

```
my ($firstChar, $lastChar) = firstLastChar("Yellow");
```

\$firstChar

"Y"

\$lastChar

"w"

The return value is a list of two elements

We pass an argument

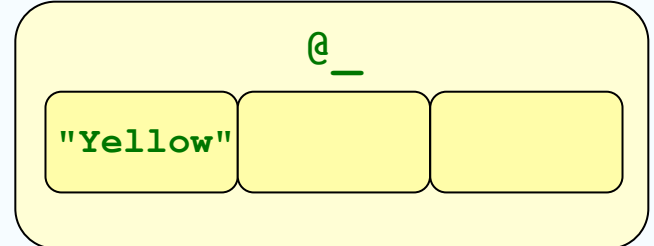
```
print "First char: $firstChar, last one: $lastChar.\n";
```

First char: Y, last one: w.

```
sub firstLastChar{  
  my ($string) = @_;  
  $string =~ m/^(.)*(.)$/;  
  return ($1,$2);  
}
```



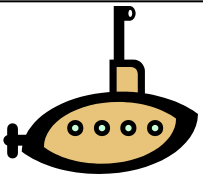
The subroutine returns a list of two elements.



\$string "Yellow"

\$1 "Y"

\$2 "w"

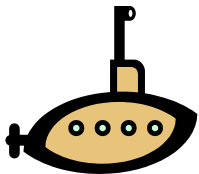


# Variable scope

When a variable is defined using `my` inside a subroutine:

- It does not conflict with a variable by the same name outside the subroutine
- Its existence is limited to the scope of the subroutine

```
sub printHello {  
    my ($name) = @_;  
    print "Hello $name\n";  
}  
  
my $name = "Liko";  
printHello("Heftziba");  
print "Bye $name\n";
```



# Variable scope

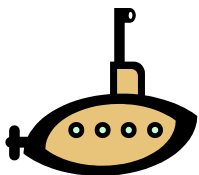
When a variable is defined using `my` inside a subroutine:

- It does not conflict with a variable by the same name outside the subroutine
- Its existence is limited to the scope of the subroutine

```
sub printHello {  
    my ($name) = @_;  
    print "Hello $name\n";  
}
```

```
my $name = "Liko";  
printHello("Heftziba");  
print "Bye $name\n";
```

**Hello Heftziba**



# Variable scope

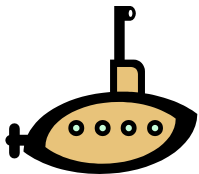
When a variable is defined using `my` inside a subroutine:

- It does not conflict with a variable by the same name outside the subroutine
- Its existence is limited to the scope of the subroutine

```
sub printHello {  
    my ($name) = @_;  
    print "Hello $name\n";  
}
```

```
my $name = "Liko";  
printHello("Heftziba");  
print "Bye $name\n";
```

```
Hello Heftziba  
Bye Liko
```

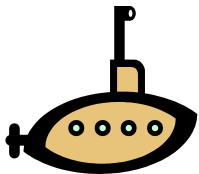


# Variable scope

When a variable is defined using `my` outside a subroutine:

- It is accessible inside the subroutine

```
my $text = "Hello World!\n"  
sub printHello {  
    print $text;  
}  
printHello();
```





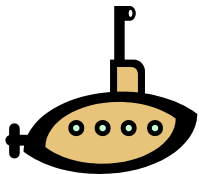
# Variable scope

When a variable is defined using `my` outside a subroutine:

- It is accessible inside the subroutine

```
my $text = "Hello World!\n"  
sub printHello {  
    print $text;  
}  
printHello();
```

**Hello World!**



# Sort revision

We learned the default sort, which is lexicographic:

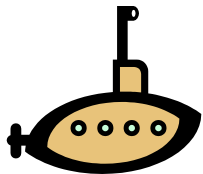
```
my @arr = (8,3,45,8.5);  
my @sorted = sort(@arr);  
print "@sorted";
```

3 45 8 8.5

To sort by a different order rule we need to give a [comparison subroutine](#) – a subroutine that compares two scalars and says which comes first

```
sort COMPARE_SUB (@array);
```

no comma here



# Sorting numbers

```
sort COMPARE_SUB (LIST);
```

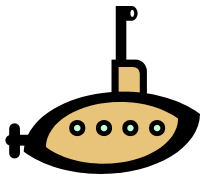
**COMPARE\_SUB** is a **subroutine** that compares two special scalars: **\$a** and **\$b** which are any two elements from the list of items to be compared.

The subroutine determines which comes first (by returning **1**, **0** or **-1**). For example:

```
sub compareNumber {  
    if ($a > $b)      {return 1;}  
    elsif ($a == $b) {return 0;}  
    else              {return -1;}  
}
```

```
my @sorted = sort compareNumber (8,3,45,8.5);  
print "@sorted\n";
```

3 8 8.5 45



no comma here

# The operator `<=>`

The `<=>` operator does exactly that – it returns 1 for “greater than”, 0 for “equal” and -1 for “less than”:

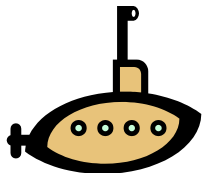
```
sub compareNumber {  
    return $a <=> $b;  
}  
print sort compareNumber (8,3,45,8.5);
```

For easier use, you can use a [temporary subroutine definition](#) in the same line:

```
print sort {return $a<=>$b;} (8,3,45,8.5);
```

or just:

```
print sort {$a<=>$b;} (8,3,45,8.5);
```

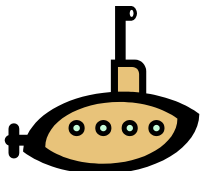


# @\_ Passing variables

What happens if we want to pass an array to a subroutine?

```
my $text = "Hello";  
my @array = (1,3,5,8,13);  
  
sub fooBar {  
    my ($sub_text,@sub_array) = @_;  
    print $sub_text."\n";  
    print @sub_array;  
}  
  
fooBar($text,@array);
```

```
Hello  
135813
```

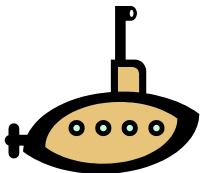


## @\_ Passing variables

What happens if we want to pass an array to a subroutine?

```
my $text = "Hello";  
my @array = (1,3,5,8,13);  
  
sub fooBar {  
    my (@sub_array,$sub_text) = @_;  
    print $sub_text."\n";  
    print @sub_array;  
}  
  
fooBar (@array,$text);
```

135813Hello



# @\_ Passing variables

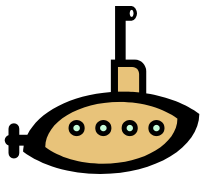
And if we want to pass multiple arrays?

```
my @array_one = ("a", "b", "c", "d");  
my @array_two = (1, 3, 5, 8, 13);
```

```
sub fooBar {  
    my (@sub_array_one, @sub_array_two) = @_;  
    print @sub_array_one;  
    print @sub_array_two;  
}
```

```
fooBar(@array_one, @array_two);
```

**abcd135813**



# References

A **reference** to a variable is a **scalar** value that “points” to another variable.

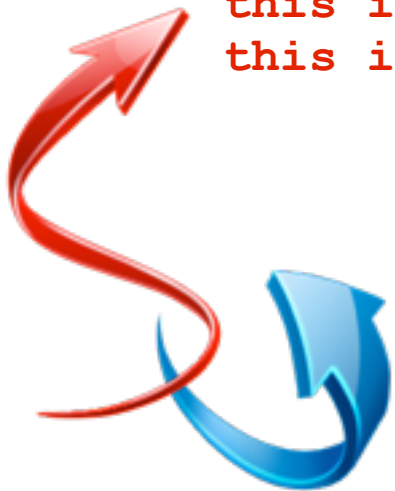
`\@array` and `\%hash` return a **reference** to the array/hash **itself**.

```
my @array = ("this", "is", "an", "array");  
print join(" ", @array) . "\n";
```

```
my $array_ref = \@array;  
print "this is the reference to the array: " . $array_ref;
```

this is an array

this is the reference to the array: ARRAY(0x7fbd5082add8)





# References

A **reference** to a variable is a **scalar** value that “points” to another variable.

`\@array` and `\%hash` return a **reference** to the array/hash **itself**.

To access the variables content you will have to **dereference** it.

```
my @array = ("this", "is", "an", "array");  
my $array_ref = \@array;  
print "this is a reference to the array: ".$array_ref."\n";  
print "it's content is: ". join(" ", @{$array_ref});
```

```
this is a reference to the array: ARRAY(0x7fd17902add8)  
it's content is: this is an array
```



# Passing variables by reference

If we want to pass arrays or hashes to a subroutine, we should pass a reference:

Passing array references:

```
subRoutine (\@arr);
```

Dereferencing arrays:

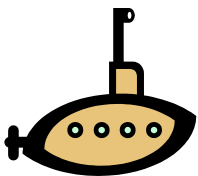
```
sub subRoutine {  
    my ($arrRef) = @_;  
    my @arr = @{$arrRef};  
    ...  
}
```

Passing hash references:

```
subRoutine (\%hash);
```

Dereferencing hashes:

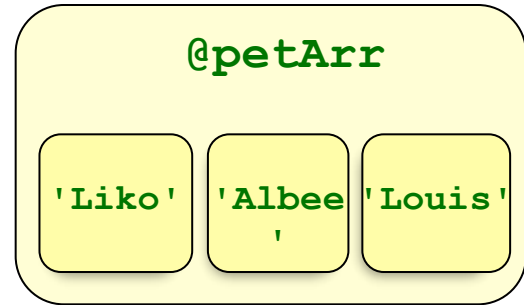
```
sub subRoutine {  
    my ($hashRef) = @_;  
    my %hash = %{$hashRef};  
    ...  
}
```



# Passing variables by reference

Passing *references*:

```
my @petArr = ('Liko', 'Albee', 'Louis');  
printPets (\@petArr);
```

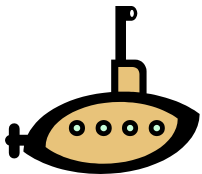


We create a reference to the array

```
sub printPets {  
  my ($petRef) = @_;  
  foreach my $pet (@{$petRef}) {  
    print "Good $pet\n";  
  }  
}
```

De-reference of `$petRef`

```
Good Liko  
Good Albee  
Good Louis
```



# Returning variables by reference

Similarly, to return a hash use a reference:

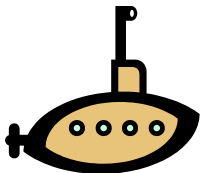
```
sub getDetails {  
    my %details;  
    $details{"name"} = <STDIN>;  
    $details{"address"} = <STDIN>;  
    ...  
    return \%details;  
}
```

```
my $detailsRef = getDetails();
```

In this case the hash continues to exist outside the subroutine!

To dereference use:

```
my %detailHash = %{$detailsRef}
```



# Exercises

1. Write a script that reads the text of “On the Origin of Species” and gives you the word frequency for each word used in it. Print out:
  1. The number of unique words (ignore upper/lowercase) used in the text
  2. the word/frequency combination in order of decreasing frequency
2. Modify the script so that it will not count words if they are shorter than a user-defined threshold and contain lowercase characters (e.g. if the threshold is  $\leq 3$  “DNA” should be counted, “The” should not).

# Exercises II

Optional: The file `books_us_english_1800_1899.csv` contains word counts found in english books published between 1800 & 1899, taken from *Google ngrams*. The first row gives the word, the second the year of publication, the third the number the word is found in that year.

Edit your script from the last exercise so that it creates the word frequencies for all words found in that century. Use this data to see which words are over- or under-represented in “On the Origin of Species”

The whole file is ~3.5 GB in size, so don't read it all at once

<b>Darwinism</b>	<b>1865</b>	<b>6</b>
<b>Darwinism</b>	<b>1866</b>	<b>3</b>
<b>Darwinism</b>	<b>1867</b>	<b>2</b>
<b>Darwinism</b>	<b>1868</b>	<b>6</b>
<b>Darwinism</b>	<b>1869</b>	<b>22</b>
<b>Darwinism</b>	<b>1870</b>	<b>71</b>
<b>Darwinism</b>	<b>1871</b>	<b>136</b>
<b>Darwinism</b>	<b>1872</b>	<b>195</b>
<b>Darwinism</b>	<b>1873</b>	<b>142</b>
<b>Darwinism</b>	<b>1874</b>	<b>319</b>
<b>Darwinism</b>	<b>1875</b>	<b>156</b>
<b>Darwinism</b>	<b>1876</b>	<b>308</b>