

Algorithms for analysis of deep sequencing data

Main Source:

4761 Computational Genomics

Itsik Pe'er, Dept of CS, Columbia Univ,
Spring 2010

Also:

M. Brudno: Introduction to High
Throughput Sequencing



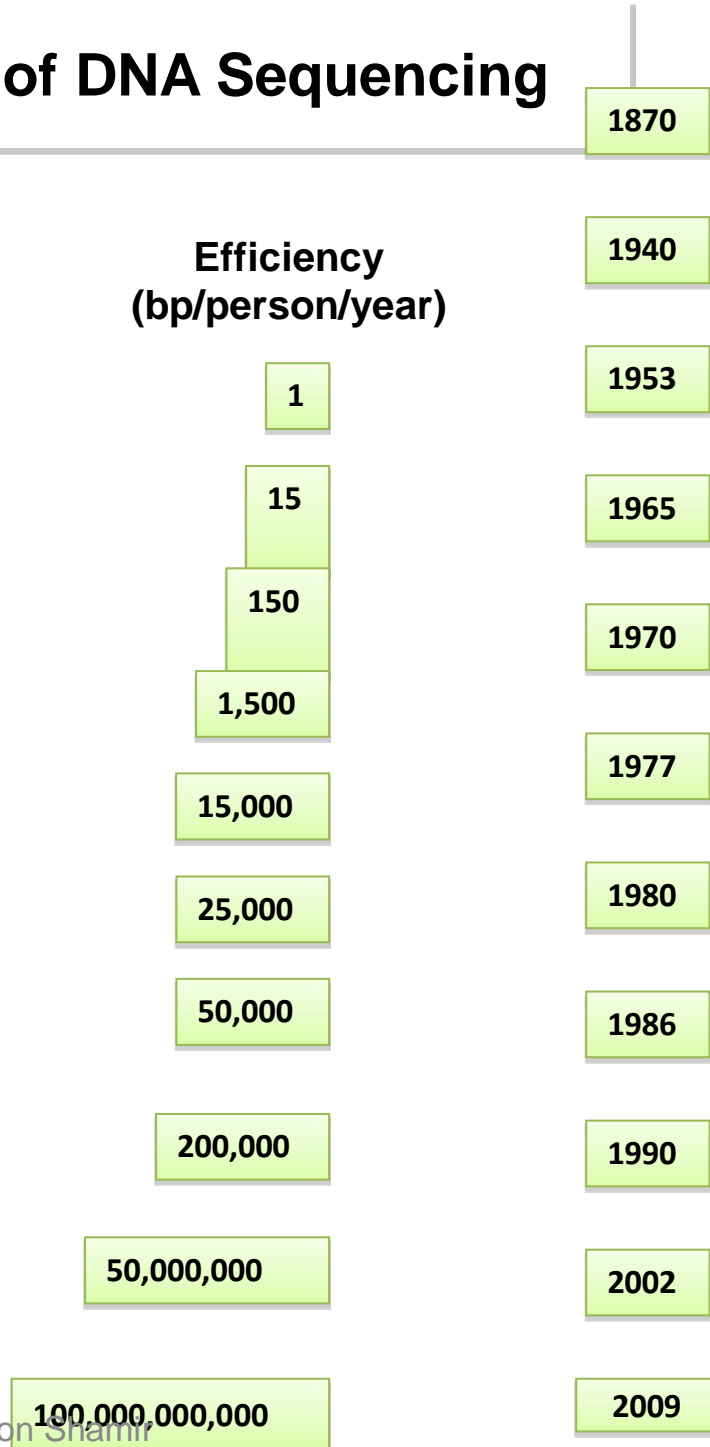
Outline

- Intro to deep sequencing technology
 - The numbers
 - Technology
- Mapping
 - basic techniques
 - MAQ algorithm
 - Bowtie algorithm
- Assembly
 - de Bruijn graph algorithms



History of DNA Sequencing

Efficiency
(bp/person/year)



1870 Miescher: Discovers DNA

1940 Avery: Proposes DNA as 'Genetic Material'

Watson & Crick: Double Helix Structure of DNA

1953 Holley: Sequences Yeast tRNA^{Ala}

1965 Wu: Sequences λ Cohesive End DNA

1970 Sanger: Dideoxy Chain Termination
Gilbert: Chemical Degradation

1977 Messing: M13 Cloning

1980 Hood et al.: Partial Automation

- 1986
- Cycle Sequencing
 - Improved Sequencing Enzymes
 - Improved Fluorescent Detection Schemes

- 2002
- Next Generation Sequencing
 - Improved enzymes and chemistry
 - New image processing
- 2009

The era of personal genomics (2010 - ?)

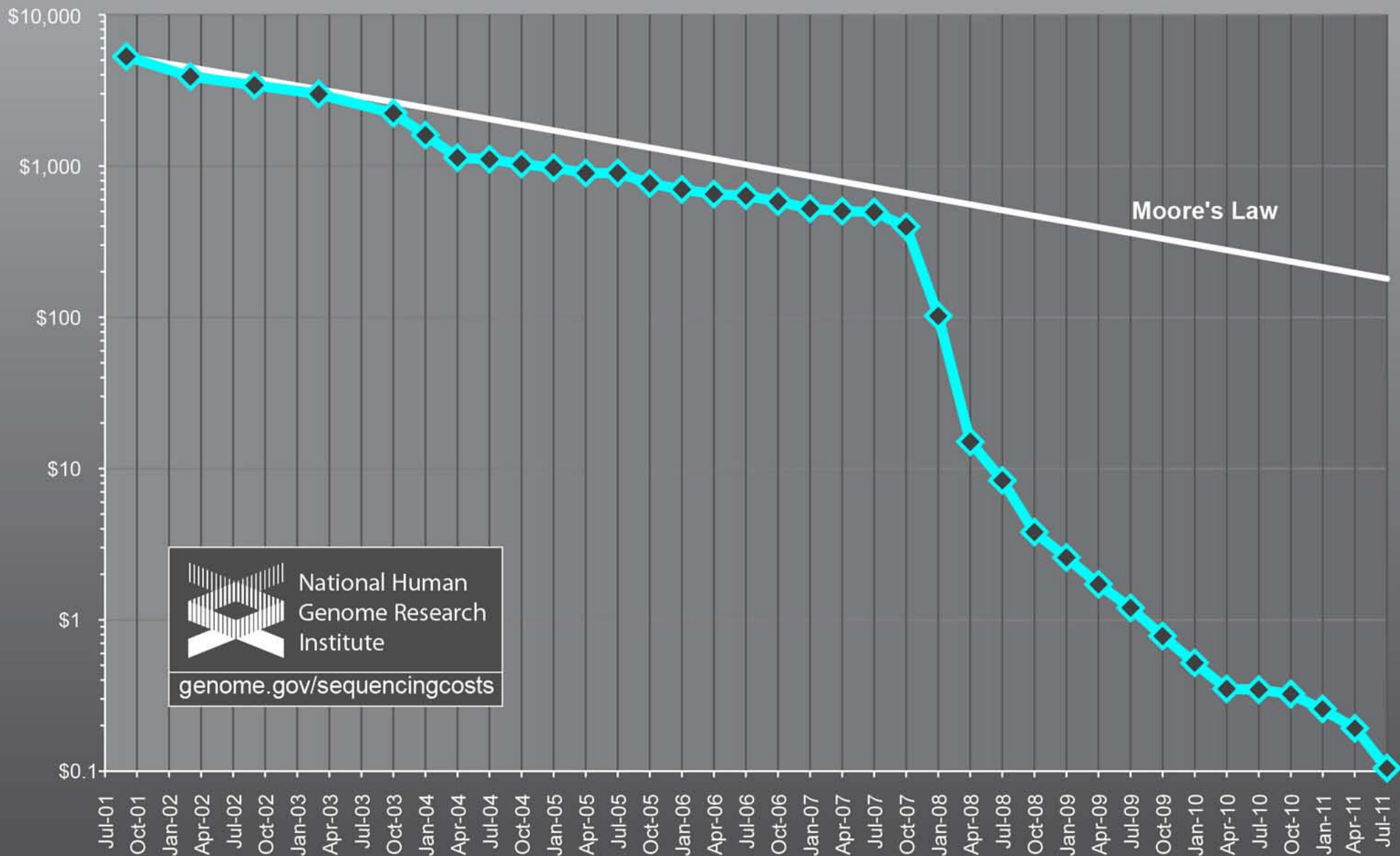
- 2003: Human genome project completed – cost **\$3 billion**
- 2003: J. Craig Venter foundation announced a prize of \$500,000 to be awarded to a group that could sequence a human genome for \$1000.
- 2005-2006: NIH announces awards of grants totaling ~\$32M for the development of sequencing technologies. X Prize foundation creates the Archon X Prize for Genomics: \$10M to a registered group that can build a device and use it to sequence 100 human genomes within 10 days or less with an accuracy of no more than one error in every 100,000 bases sequenced for no more than \$10,000 per genome.
Numerous challenges
Bioscience
- **Sep.-Oct.** personal genomics company 23andme offers *genotyping* for \$100.

September 21, 2010

Illumina Offers \$10K Human Genome for Research; Sees 'All-In' \$1,000 Genome in Three to Five Years

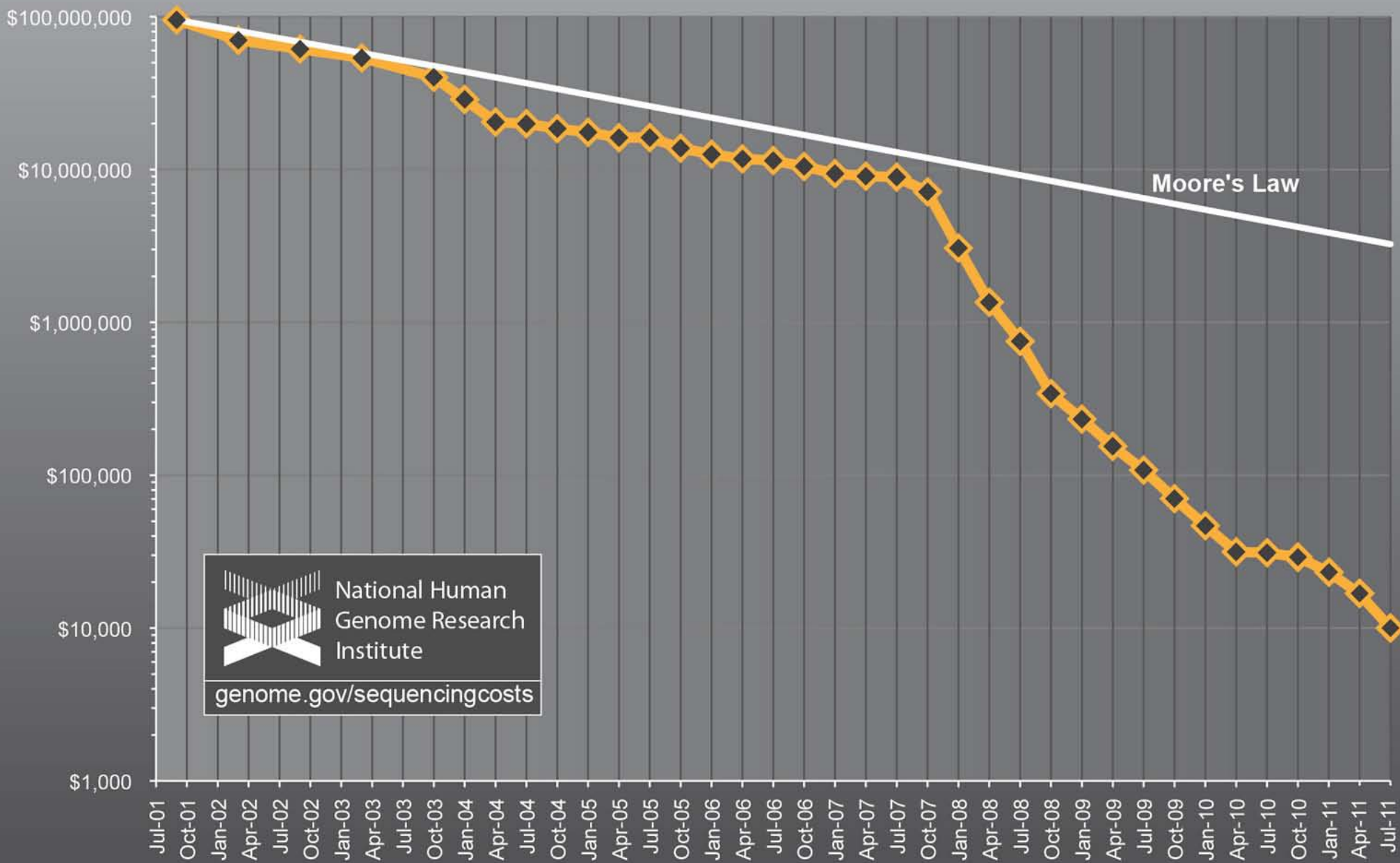
Under its Individual Genome Sequencing service, which is targeted at consumers and doctors, the company has sequenced the genomes of 24 individuals to date, among them eight with clinical conditions.

Cost per Megabase of DNA Sequence



 National Human
Genome Research
Institute
genome.gov/sequencingcosts

Cost per Genome

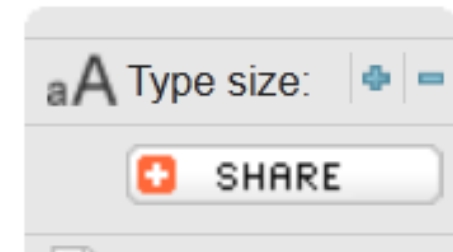


 National Human
Genome Research
Institute
genome.gov/sequencingcosts

Comparison of Illumina, Complete Genomics for Human WGS Shows No Single Platform Tells it All

December 20, 2011

By Julia Karow



According to Reid, sequencing the same genome on two platforms as a method to validate variants is "a reasonable strategy" but "may not be the best choice if resources are limited and doing so would reduce study sample size."

Taken together, the similarities between the two platforms are "quite strong," and the results "very similar," said Stanford's Clark. He added that Illumina's and Complete's human whole-genome sequencing services are "very competitive" in terms of price, about \$3,000 to \$4,000 per genome. "If you had to pick one, you are better off with whichever gives you the best deal and the best turnaround time," he said.

Lam et al, Nature Biotechnology AOP 18 Dec 2011⁷

How do we read DNA?



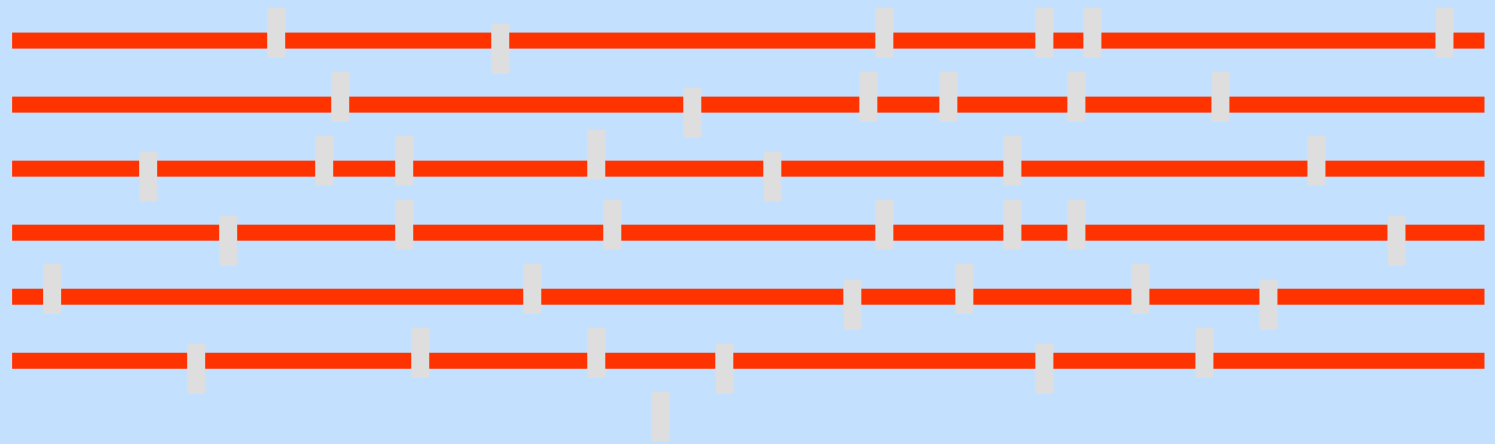
How do we read DNA?

- We replicate it

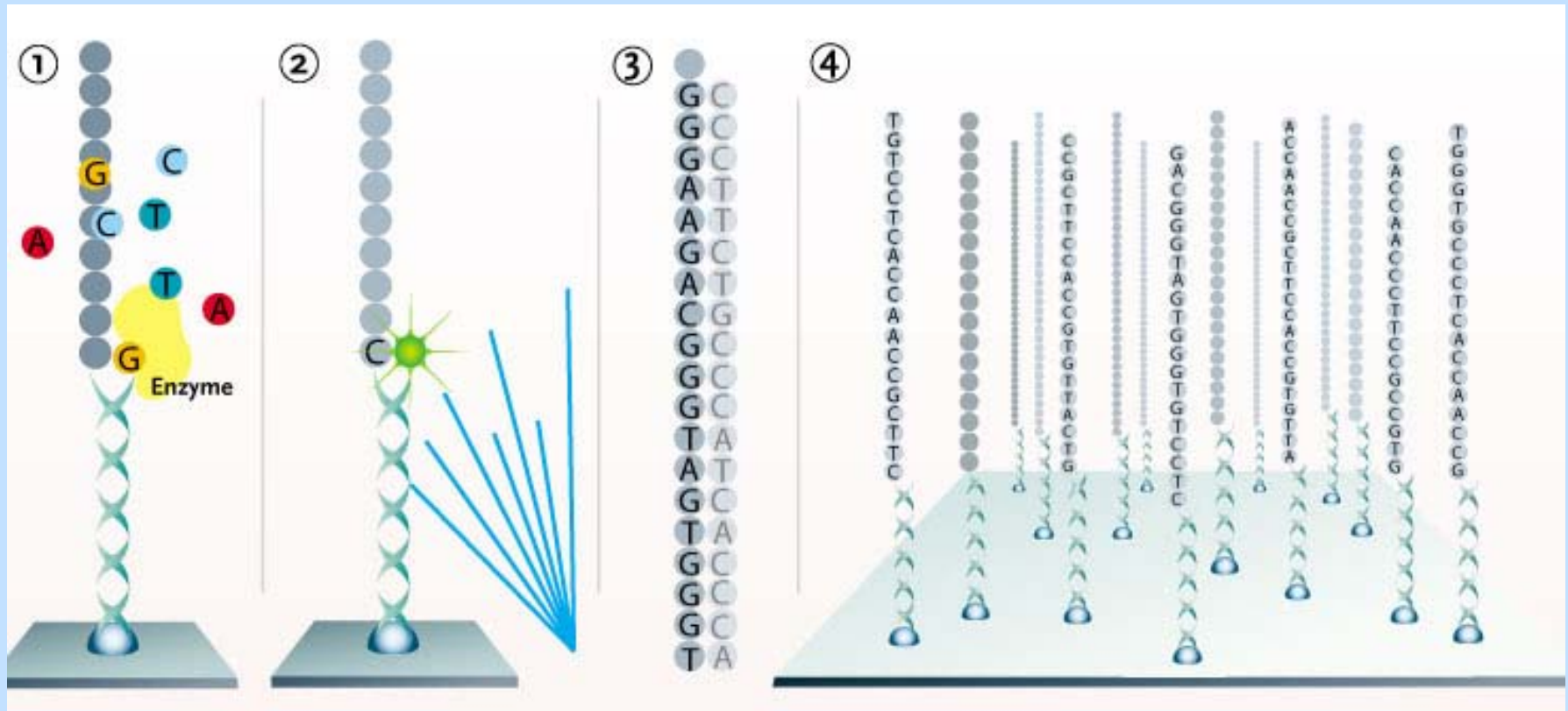


How do we read DNA?

- We replicate it
- We shred it

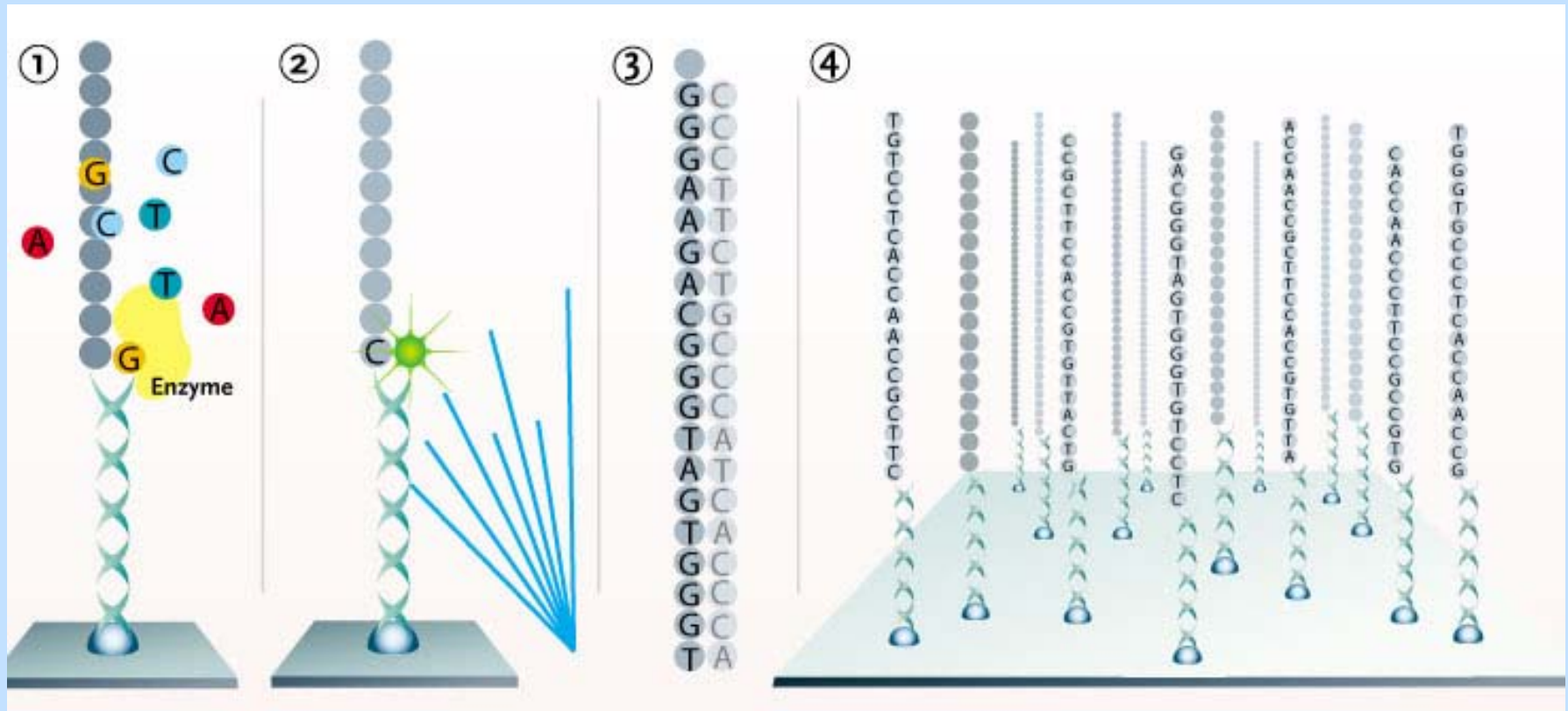


Reading short DNA



- Use replication machinery with colored bases
 - Take pictures of massively parallel reaction
- 10 million reads of 30 per day & \$1000

Reading short DNA



- Use replication machinery with colored bases
- Take pictures of massively parallel reaction

100 million reads of 100 per day & \$1000

Example

The genome is:

TTATGGTCGGTGAGTGTGACTGGTGTTGTCTAA

The reads are:

GGTCGGTGAG

TGAGTGTGAC

TGGTGTTGTC

TGACTGGTTT

AATGGTCGGT

GAGTGTGACT

AAAAAAAAAA

Example

The genome is:

TTATGGTCGGTGAGTG TGACTGGTGTTGTCTAA
 | | | | | | | | | |
 GGTCGGTGAG

The reads are:

GGTCGGTGAG
TGAGTGTGAC
TGGTGTTGTC
TGACTGGTTT
AATGGTCGGT
GAGTGTGACT
AAAAAAAAAA

Example

The genome is:

TTATGGTCGGTGAGTGTGACTGGTGTTGTCCTAA
 | | | | | | | | | |
 TGAGTGTGAC

The reads are:

GGTCGGTGAG
TGAGTGTGAC
TGGTGTTGTC
TGACTGGTTT
AATGGTCGGT
GAGTGTGACT
AAAAAAAAAA

Example

The genome is:

TTATGGTCGGTGAGTGTGACTGGTGTTGTCTAA

|||||

AATGGTCGGT

The reads are:

GGTCGGTGAG

TGAGTGTGAC

TGGTGTTGTC

TGACTGGTTT

AATGGTCGGT

GAGTGTGACT

AAAAAAAAAA

Example

The genome is:

TTATGGTCGGTGAGTGTGACTGGTGTTGTCTAA
GGTCGGTGAG

TGAGTGTGAC

TGGTGTTGTC

TGACTGGTTT

AATGGTCGGT

GAGTGTGACT

AAAAAAAAAA

NGS algorithmics

1. Mapping and the MAQ algorithm

Short read Mapping Problem

- Align reads to genome
- **Input:**
Many reads: l -long strings S_1, \dots, S_m
Approximate reference genome: string R
- **Output:**
 x_1, \dots, x_m along R where reads match, resp.
- **Complications:**
 - Errors, Differences, Repetitive regions



Solutions

- Naive :
 - For each S_i
 - For each position $p=l, \dots, |R|$
 - Try matching S_i to the substring $R[p-l+1, \dots, p]$
- Complexity:
 $O(lm|R|)$ exact or inexact matching



Solutions (2)

- Less Naive:
 - For each S_i
 - Match S_i to R using KMP
- Complexity:
 $O(m(l + |R|)) = O(ml + m|R|)$ exact matching



Solutions (3)

- Suffix tree approach:
 - Build suffix tree for R
 - For each S_i
 - Find matches of S_i to R by tree traversal from the root
- Time Complexity: $O(lm + |R|)$ exact matching
- Space Complexity: $O(|R| \log |R|)$ vs $|R| \log |\Sigma|$ for the text
- Can store Human Genome text in 750M bytes (6G bits) but, need ~64G bytes for the tree
 - large constants, hard to implement



Solutions (4)

- Preprocessing:
 - Create hash-table H
 - For each position $p=l, \dots, |R|$
 - Hash (key= $R[p-l+1, \dots, p]$, value= p) in H
- For each S_i report $H(S_i)$
- Complexity:
 $O(lm + |R|l)$
- Problems:
Only exact matching, memory $O(|R|l)$



Improvements

- Pack strings into bit vectors
 - Need only $2l$ bits for each read
 - Hash set to fit in memory
- Partition the genome to $|R|/|Mem|$ chunks

TGAGTGTGAC
11100010111011100001

Mapping Issues

Goal: Mapping, Alignment, and Calling

- Problems:
 - Millions of reads to analyze
 - Repetitive regions: non-unique mapping
 - Mutations/errors: imperfect match, incorrect mapping, diploid samples
- Solutions (to be expanded later) :
 - Hash while allowing mismatches
 - Map as best as you can + report probability of error
 - Combine data from multiple reads for calling



Parameters

- Assume:
 - 10^7 reads of length 50-100 per run
 - Possibly paired-ends
 - Genome of length $3 \cdot 10^9$
 - 2Gb memory



Outline of approach

1. Hash the **reads**, then scan the **reference**
2. Find the *reference* match with the lowest mismatch
3. Assign each alignment the probability it is not true
4. Use mapped reads to call nucleotide with variant scores



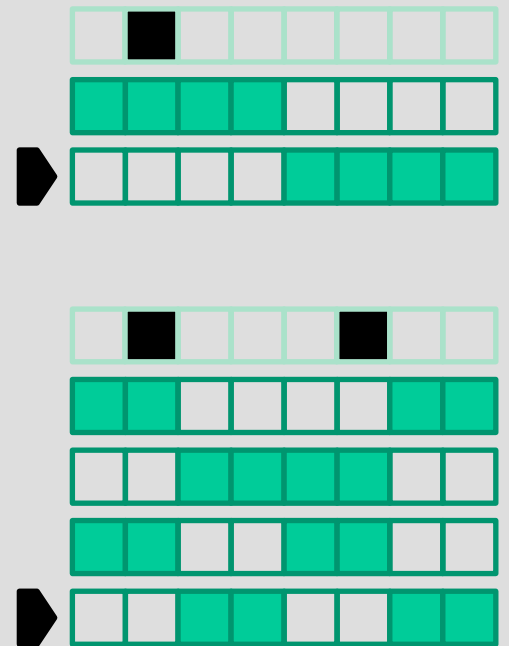
Read alignment

1. **Indexing:** Hash the first n bits iteratively into 2 masked tables.
2. Check reference against hash
3. If hit:
 - score it (#mismatches)
 - keep the best

6 templates find 100% of 2mm and 57% 3mm.

20 templates find 100% of 3mm and 64% 4mm.

Gapped Seeds - Similar ideas improve accuracy of homology searches like BLAST



MAQ algorithm

Mapping and Assembly with Quality

1. Index the first 28 bases of each read with templates 1,2; generate H1, H2
2. Scan the reference: for each position and each orientation, hash with template 1,2; if hit - extend and score the complete read; keep per read 2 best scored hits and no. of 0/1/2 mismatch hits only
3. Repeat with templates 3,4, then 5,6

~Complexity: Time: 1. $O(ml)$ 2. $O(|R|l)$

Space: 1. $O(ml)$ 2. $O(ml+|R|)$ total, but only $O(ml)$ in cache



Mapping qualities



$$Q_s = -10 \log_{10} \Pr(\text{read is wrongly mapped})$$

“Phred-scaled quality”

e.g. $Q_s=30$ if 1 in 1000 prob of incorrect mapping of the read

Assume sequencing errors are independent along read

$p(z|x,u)$: Pr. read z coming from position u in reference x

If Z mapped to u has 2 mismatches with Phred base qualities 10, 20 then $p(z|x,u)=10^{-(20+10)/10} = 0.001$



Mapping qualities (2)

x = reference seq; z = read seq;

u = mapped position; *Assume uniform prior $p(u/x)$*

$$\begin{aligned} p_s(u | x, z) &= \frac{p(z | x, u) p(u)}{p(x, z)} = \frac{p(z | x, u) p(u)}{\sum_{v=1}^{|R|-l+1} p(z | x, v) p(v)} = \\ &= \frac{p(z | x, u)}{\sum_{v=1}^{|R|-l+1} p(z | x, v)} \end{aligned}$$

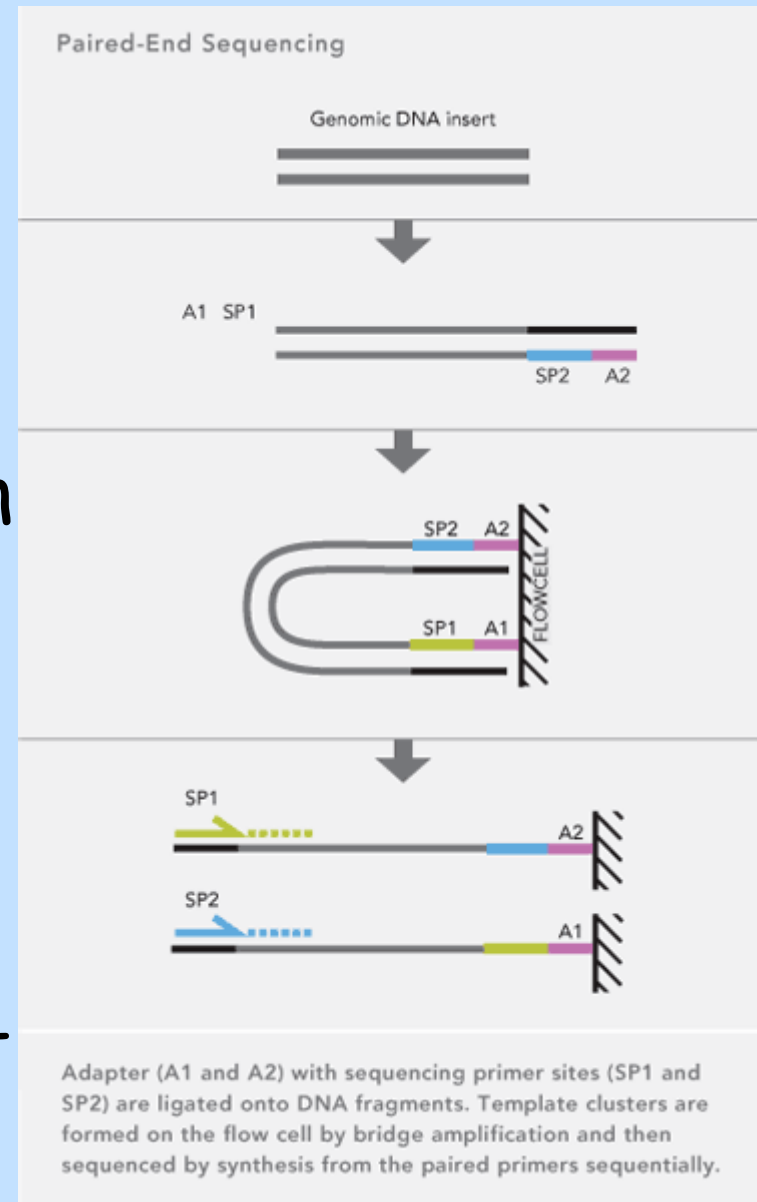
$p(z|x,u)$ = product of mismatch error probabilities
 Σ estimated from best hit, 2nd best, all other hits

$$Q_s(u | x, z) = -10 \log_{10}(1 - p_s(u | x, z))$$



Paired-end reads

- Index "mate" reads s_1, s_2 simultaneously
- Hash reference
- For mates that hit uniquely & consistently (right orientation distance): $Q_p = Q_{s_1} + Q_{s_2}$
- For mates that hit in multiple places: (Q_{s_1}, Q_{s_2})
- For mapped s_1 , but no hit to s_2 , find gapped alignment (Smith-Waterman) to detect short



Summary

- Indexing speeds up alignment in DB-search; mapping
- Storage and cache considerations are paramount!

Bibliography:

Li, Ruan and Durbin

Mapping short DNA sequencing reads and calling variants using mapping quality scores, *Genome Research*, 2008

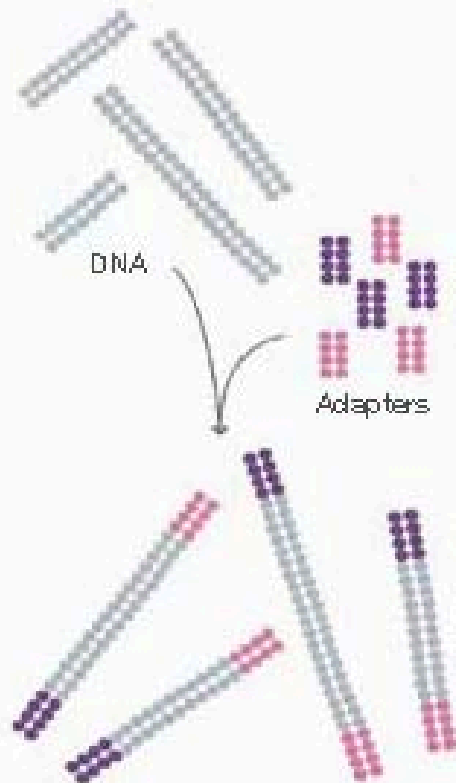


More on the NGS technology



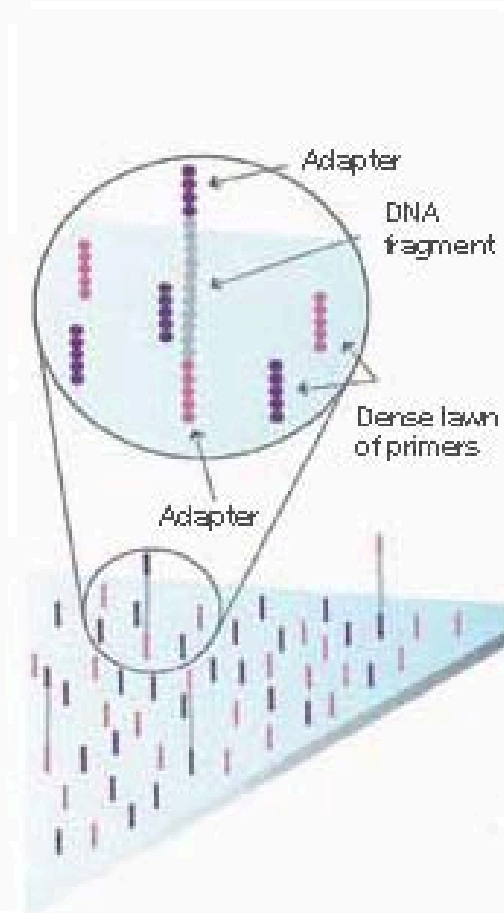
Illumina NGS technique

Figure 2: Prepare Genomic DNA Sample



Randomly fragment genomic DNA and ligate adapters to both ends of the fragments.

Figure 3: Attach DNA to Surface

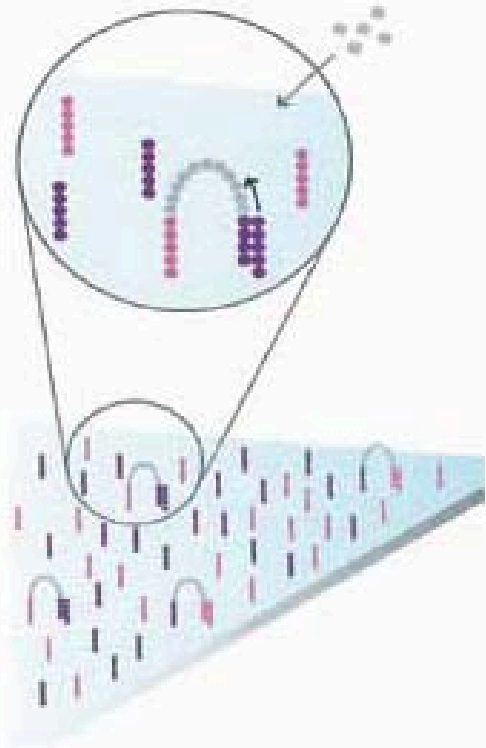


Bind single-stranded fragments randomly to the inside surface of the flow cell channels.

Technology Spotlight: Illumina® Sequencing 2010

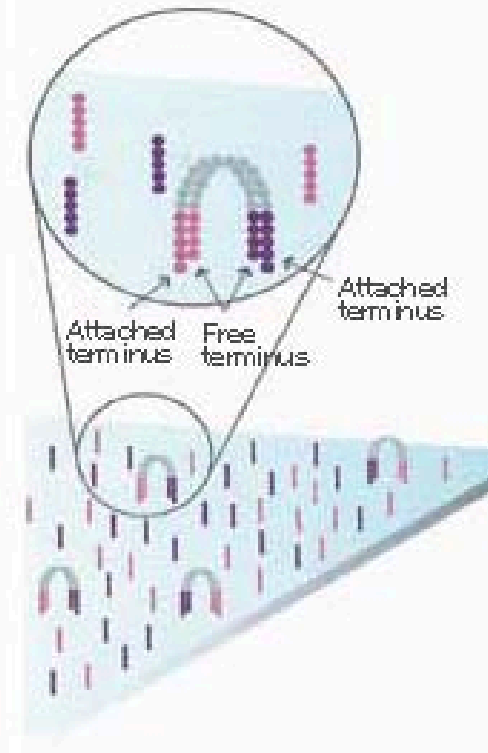
<http://www.illumina.com/support/literature.ilmn>

Figure 4: Bridge Amplification



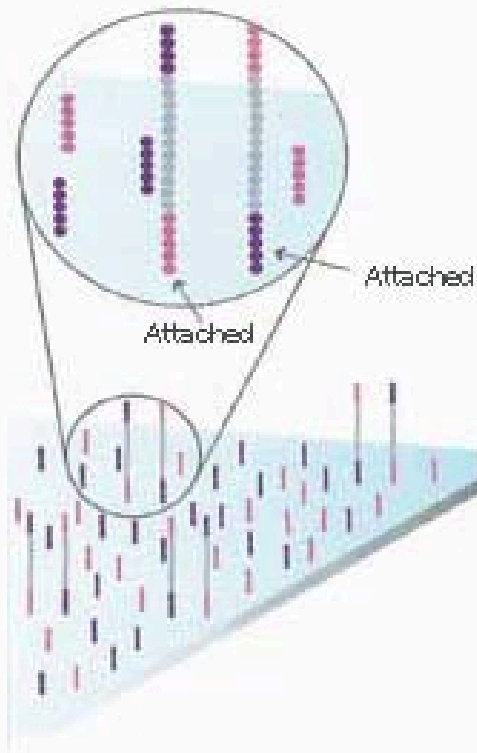
Add unlabeled nucleotides and enzyme to initiate solid-phase bridge amplification.

Figure 5: Fragments Become Double Stranded



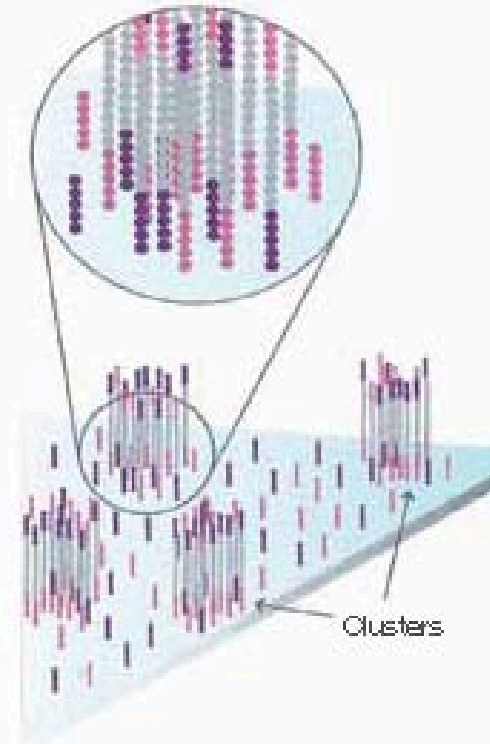
The enzyme incorporates nucleotides to build double-stranded bridges on the solid-phase substrate.

Figure 6: Denature the Double-Stranded Molecules



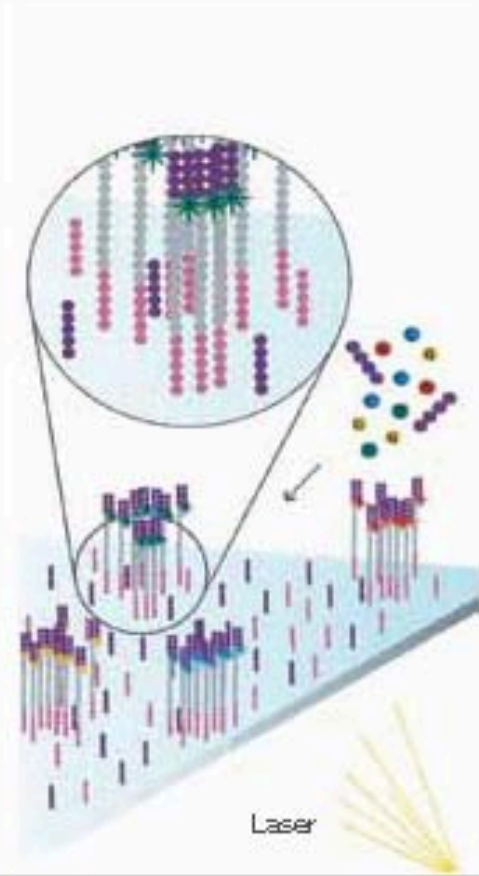
Denaturation leaves single-stranded templates anchored to the substrate.

Figure 7: Complete Amplification



Several million dense clusters of double-stranded DNA are generated in each channel of the flow cell.

Figure 8: Determine First Base



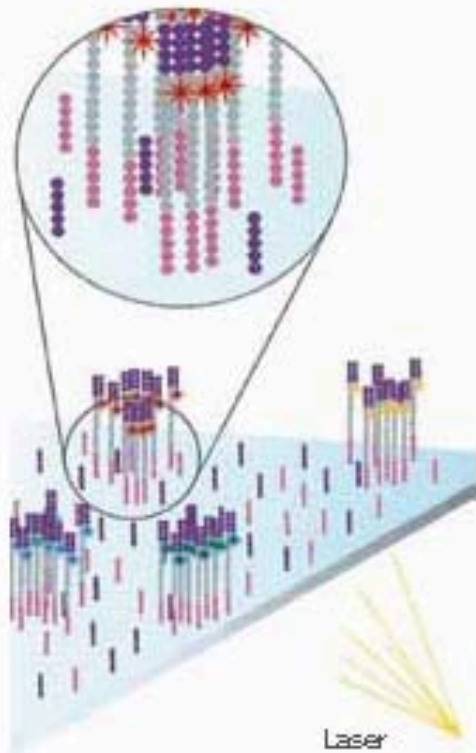
The first sequencing cycle begins by adding four labeled reversible terminators, primers, and DNA polymerase.

Figure 9: Image First Base



After laser excitation, the emitted fluorescence from each cluster is captured and the first base is identified.

Figure 10: Determine Second Base



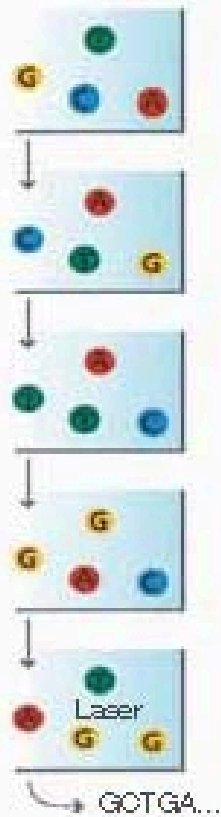
The next cycle repeats the incorporation of four labeled reversible terminators, primers, and DNA polymerase.

Figure 11: Image Second Chemistry Cycle



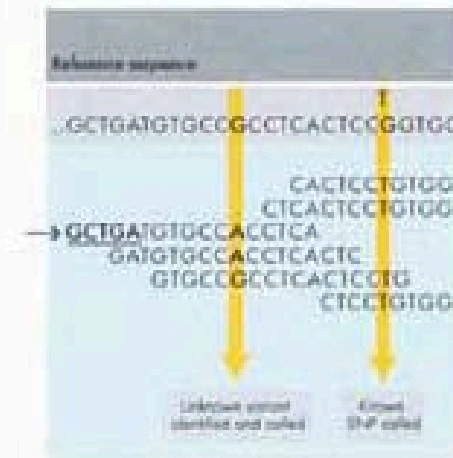
After laser excitation, the image is captured as before, and the identity of the second base is recorded.

Figure 12: Sequencing Over Multiple Chemistry Cycles



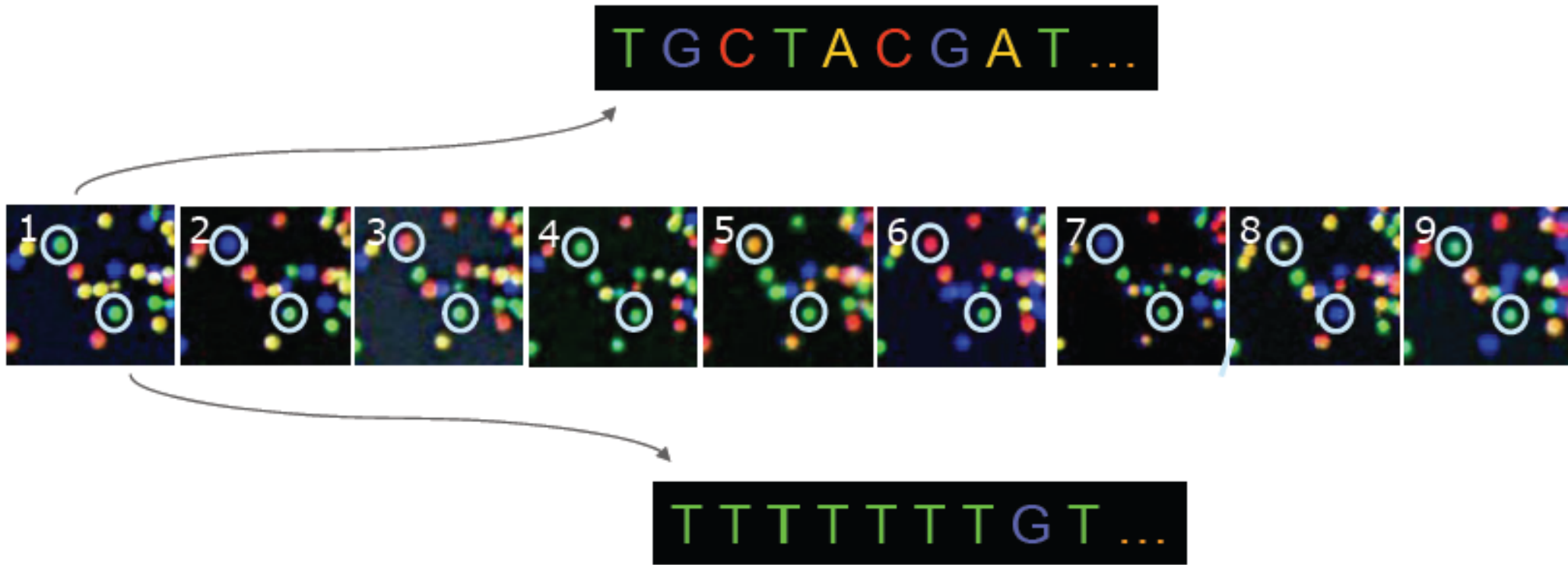
The sequencing cycles are repeated to determine the sequence of bases in a fragment, one base at a time.

Figure 13: Align Data



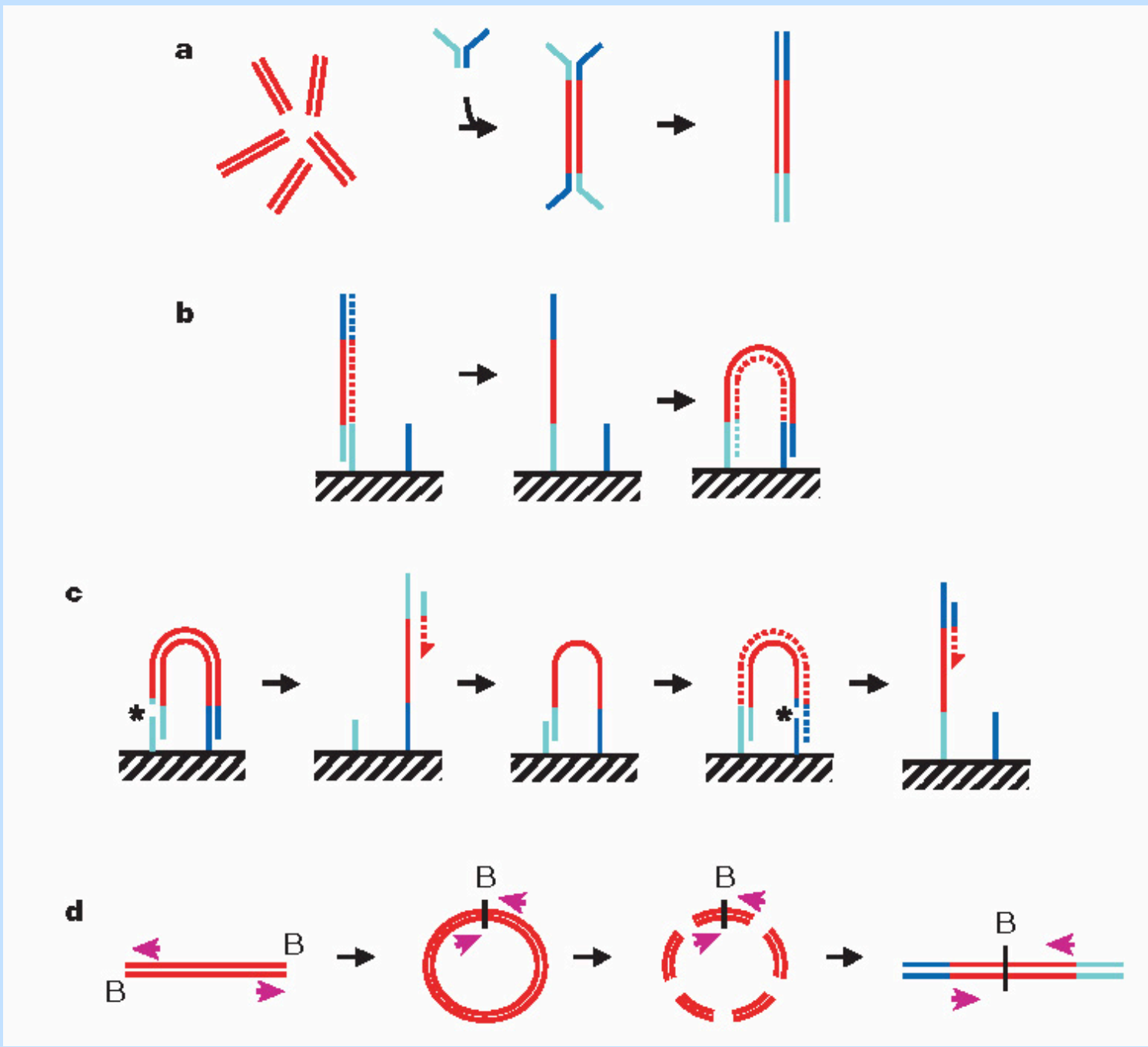
The data are aligned and compared to a reference, and sequencing differences are identified.

Base calling from raw data



The identity of each base of a cluster is read off from sequential images





Bentley et al. Accurate whole human genome sequencing using reversible terminator chemistry, Nature 2008
 CG © Ron Shamir

Technologies

- 454/Roche Genome Sequencer (GS)
 - Since 2004
 - FLX Titanium platform: 1 million reads, over 400 bp long
- Solexa/Illumina Genome Analyzer (GA)
 - Since 2006
 - GAIIx platform: 200 million reads, 75–100 bp long
- Applied Biosystems (ABI) Sequencing by Oligo Ligation and Detection (SOLiD)
 - Since 2007
 - 400 million reads, 50bp long



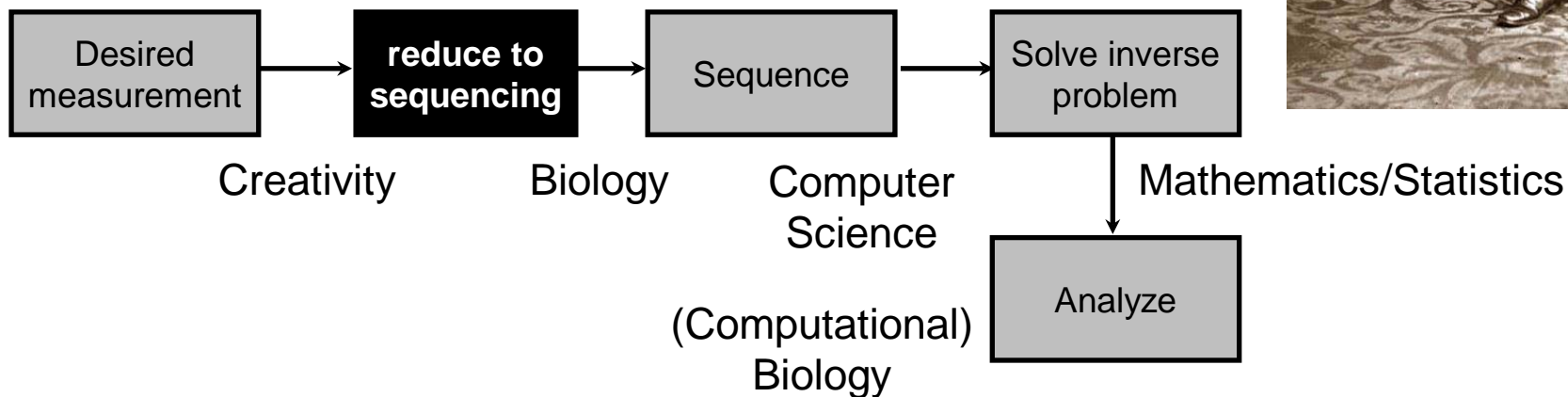
→ yield per run: 0.4 to 20 Gpb ($\frac{1}{4}$ 6 human genomes)!

NGS Technologies (2010)

Human Genome	6GB == 6000 MB		
Req'd Coverage	6	12	30
	3730	454	Illumina
bp/read	600	400	2X75
reads/run	96	500,000	100,000.000
bp/run	57,600	0.5 GB	15 GB
# runs req'd	625,000	144	12
runs/day	2	1	0.1
Machine days/human genome	312,500 (856 years)	144	120
Cost/run	\$48	\$6,800	\$9,300
Total cost	\$15,000,000	\$979,200	\$111,600

Sequence census methods

- High throughput **DNA sequencers are also broadly utilized as bean counters** for “sequence census” methods.
- The **vast majority** of sequencing experiments are used for *-seq protocols:



- Assays include: **ChIP-Seq, RNA-Seq, methyl-Seq, GRO-Seq, Clip-Seq, BS-Seq, FRT-Seq, TraDI-Seq, Hi-C, ...**

New *-seq technologies

each comes equipped with its own inverse problem

- Variant detection: **Seq, Exome-Seq, BIC-Seq**
- Protein DNA binding: **ChIP-Seq**
- Transcriptome: **RNA-Seq, FRT-Seq**
- Ribosomal profiling: **GRO-Seq, Ribosomal profiling**
- Replication timing: **Repli-Seq**
- DNase I hypersensitivity sites: **DNase-Seq**
- Transposon mapping: **TraDIS**
- Chromatin conformation: **Hi-C, ChIA-Pet**
- Methylation: **methyl-Seq, BS-Seq, RRBS, MeDIP-Seq, MBD-Seq**
- Protein RNA binding: **Rip-Seq, Clip-Seq**
- RNA secondary structure: **PARS, Frag-Seq, Shape-Seq**
- Metagenomics: **Metagenome sequencing, metatranscriptome sequencing**

NGS algorithmics

2. Efficient indexing of sequences

Bowtie and the Burrows-Wheeler Transform



NGS Parameters

- Assume:
 - 10^7 reads of length 50-100 per run
 - Possibly paired-ends
 - Genome of length $3 \cdot 10^9$
 - 2Gb memory



Mapping Reads

The genome is:

TTATGGTCGGTGAGTGTGACTGGTGTTGTCTAA

The reads are:

GGTCGGTGAG

TGAGTGTGAC

TGGTGTTGTC

TGACTGGTTT

AATGGTCGGT

GAGTGTGACT

AAAAAAAAAA

Indexing Reads

m l -long reads

1.	GGTCGGTGAG
2.	TGAGTGTGAC
3.	TGGTGTTGTC
4.	TGACTGGTTT
5.	AATGGTCGGT
6.	GAGTGTGACT
7.	AAAAAAAAAA

AAAAA	7	7
AATGG	5	
GAGTG	6	
GGTCG	1	
GGTTT	4	
GTGAC	2	
GTGAG	1	
TCGGT	5	
TGACT	4	6
TGAGT	2	
TGGTG	3	
TTGTC	3	

Memory [bits]:
 $m \times 2l$
 +
 $m \times (2l/2 + \log m)$

G -long genome

TTATGGTCGGTGAGTGTGACTGGTGTTGTCTAA

Indexing the Genome

G-long genome

TTATGGTCGGTGAGTGTGACTGGTGTTGTCTAA

m l-long reads

- GGTCGGTGAG
- TGAGTGTGAC
- TGGTGTTGTC
- TGACTGGTTT
- AATGGTCGGT
- GAGTGTGACT
- AAAAAAAAAA

ACTGG	19	GTGAC	16	TGGTC	4
AGTGT	13	GTGAG	10	TGTTG	24
ATGGT	3	GTGTG	14	TGTGA	15
CGGTG	8	GTGTT	23	TTATG	1
CTGGT	20	GTTGT	25	TTGTC	26
GACTG	18	TATGG	2		
GAGTG	12	TCGGT	7		
GGTCG	5	TCTAA	29		
GGTGA	9	TGACT	17		
GGTGT	22	TGAGT	11		
GTCGG	6	TGGTG	21		
GTCTA	28	TGTCT	27		

Memory [bits]:
 $G \times (2l/2 + \log G)$

Efficient Indexing: Burrows-Wheeler Transform

the_next_text_that_i_index.

Burrows-Wheeler Transform: Cyclic shifts

the_next_text_that_i_index.
he_next_text_that_i_index.t
e_next_text_that_i_index.th
_next_text_that_i_index.the
next_text_that_i_index.the_
ext_text_that_i_index.the_n
xt_text_that_i_index.the_ne
t_text_that_i_index.the_nex
_text_that_i_index.the_next
text_that_i_index.the_next_

Burrows-Wheeler Transform: Cyclic shifts

the_next_text_that_i_index.
he_next_text_that_i_index.t
e_next_text_that_i_index.th
_next_text_that_i_index.the
next_text_that_i_index.the_
ext_text_that_i_index.the_n
xt_text_that_i_index.the_ne
t_text_that_i_index.the_nex
_text_that_i_index.the_next
text_that_i_index.the_next_
ext_that_i_index.the_next_t
xt_that_i_index.the_next_te
t_that_i_index.the_next_tex
_that_i_index.the_next_text

that_i_index.the_next_text_
hat_i_index.the_next_text_t
at_i_index.the_next_text_th
t_i_index.the_next_text_tha
_i_index.the_next_text_that
i_index.the_next_text_that_
_index.the_next_text_that_i
index.the_next_text_that_i_
ndex.the_next_text_that_i_i
dex.the_next_text_that_i_in
ex.the_next_text_that_i_ind
x.the_next_text_that_i_inde
.the_next_text_that_i_index

Burrows-Wheeler Transform: Cyclic shifts sorted

.the_next_text_that_i_index
_i_index.the_next_text_that
_index.the_next_text_that_i
_next_text_that_i_index.the
_text_that_i_index.the_next
_that_i_index.the_next_text
at_i_index.the_next_text_th
dex.the_next_text_that_i_in
e_next_text_that_i_index.th
ex.the_next_text_that_i_ind
ext_text_that_i_index.the_n
ext_that_i_index.the_next_t
hat_i_index.the_next_text_t
he_next_text_that_i_index.t

i_index.the_next_text_that_
index.the_next_text_that_i_
ndex.the_next_text_that_i_i
next_text_that_i_index.the_
t_i_index.the_next_text_tha
t_text_that_i_index.the_nex
t_that_i_index.the_next_tex
text_that_i_index.the_next_
that_i_index.the_next_text_
the_next_text_that_i_index.
x.the_next_text_that_i_inde
xt_text_that_i_index.the_ne
xt_that_i_index.the_next_te

Inverting BWT

- How many “e” do we have?
- Can you recover the 1st column?

x
t
i
e
t
t
h
n
h
d
n
t
t
t
t
-
-
i
-
a
x
x
-
-
·
e
e
e

Inverting BWT

- How many “xt” do we have?
- Can you recover the first two columns?
- Which of the **x**'s is followed by a “t”?
- Which of the “xt”-s follows an “e”?
- Which “ext” is in “text”?

.
-
-
-
-
-
-
a
d
e
e
e
e
h
h
i
i
n
n
t
t
t
t
t
t
t
t
x
x
x

.....

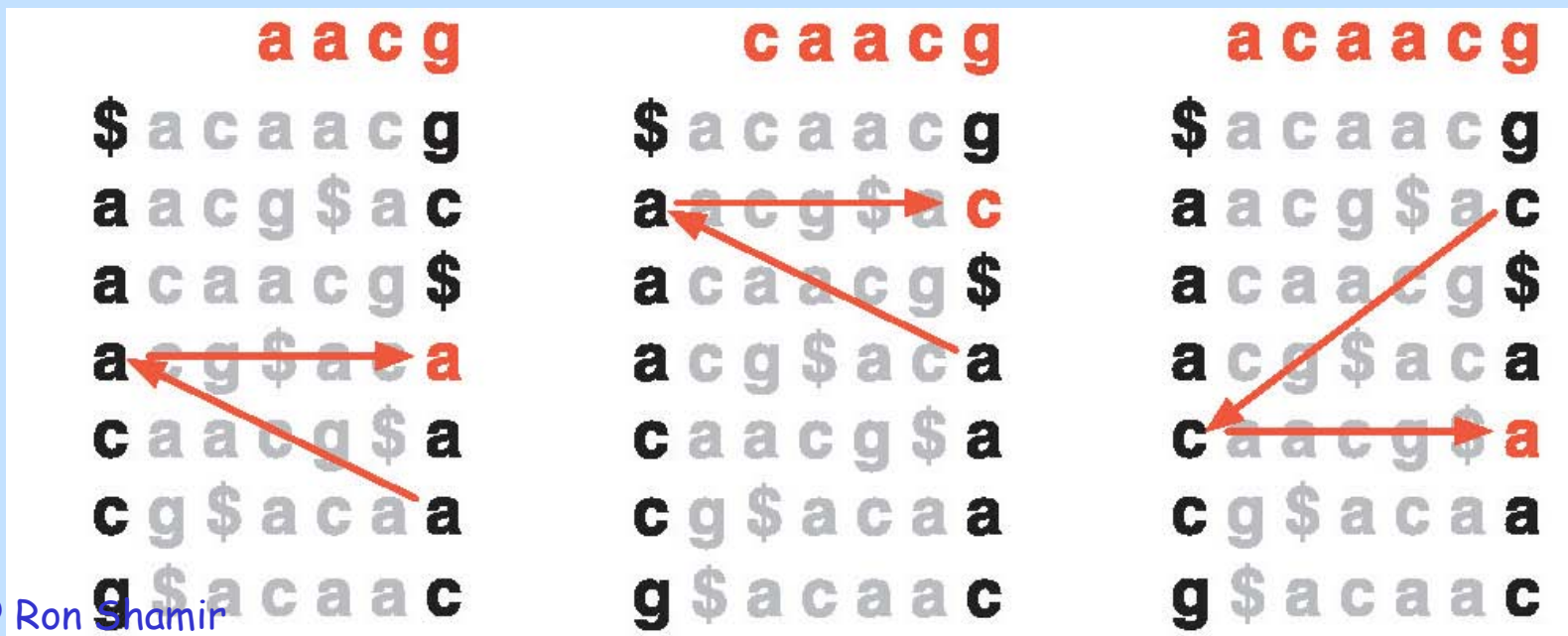
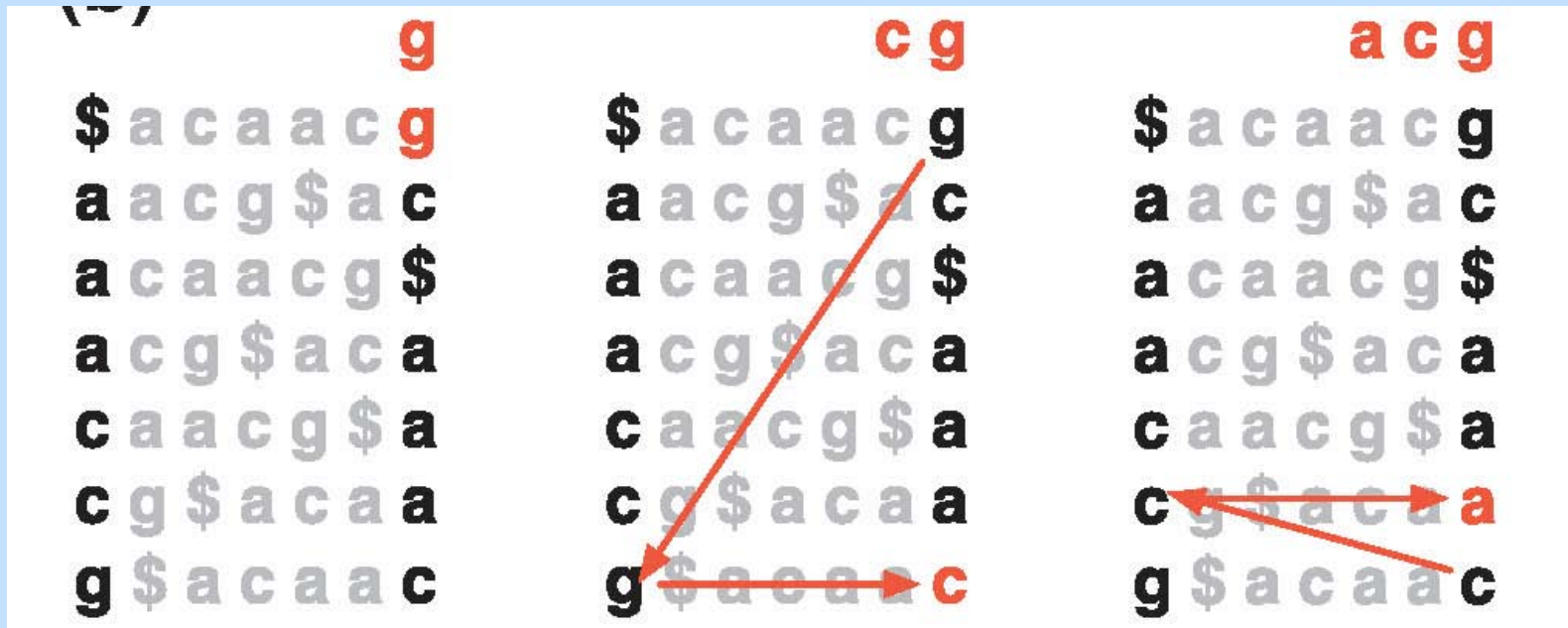
x
t
i
e
t
t
h
n
h
d
n
t
t
t
-
i
-
a
x
x
-
-
.
e
e
e

Key lemmas

- In the i -th row of M , $L(i)$ precedes $F(i)$ in the original text: $T = \dots L(i)F(i) \dots$
 - The i -th occurrence of char X in L corresponds to the same text character as the i -th occurrence of X in F
- “last-first (LF) mapping”



Inverting BWT



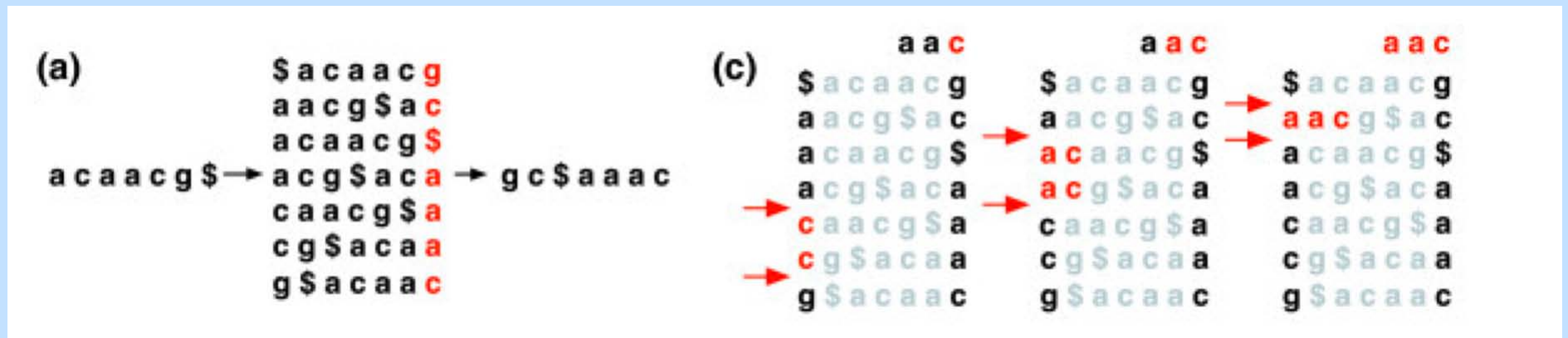
Searching: find all occurrences of **ext**

.the_next_text_that_i_index
_i_index.the_next_text_that
_index.the_next_text_that_i
_next_text_that_i_index.the
_text_that_i_index.the_next
_that_i_index.the_next_text
at_i_index.the_next_text_th
dex.the_next_text_that_i_in
e_next_text_that_i_index.th
ex.the_next_text_that_i_ind
ext_text_that_i_index.the_n
ext_that_i_index.the_next_t
hat_i_index.the_next_text_t
he_next_text_that_i_index.t

i_index.the_next_text_that_
index.the_next_text_that_i_
ndex.the_next_text_that_i_i
next text that i index.the
t_i_index.the_next_text_tha
t_text_that_i_index.the_nex
t_that_i_index.the_next_tex
text_that_i_index.the_next_
that_i_index.the_next_text_
the_next_text_that_i_index.
x.the_next_text_that_i_inde
xt_text_that_i_index.the_ne
xt_that_i_index.the_next_te

Out of all x-s in the last col, those in the interval of t are 2nd and 3rd → should appear 2nd and 3rd in the interval of x (if at all)

Searching the Index



- Search last nucleotide and expand backwards
- Maintain interval of possible matches



Exact or Inexact Match Using BWT



88, 88
177, 177
266, 266
396, 396

g

1, 30
104, 124
184, 184
278, 290

t

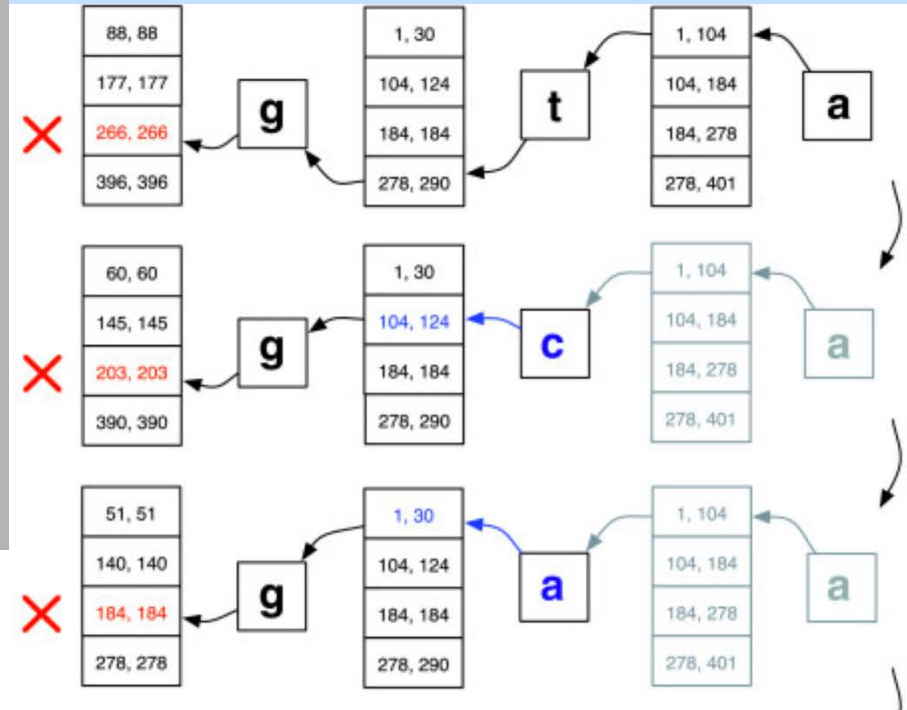
1, 104
104, 184
184, 278
278, 401

a

Position after BWT is not a position along original sequence!

Matching Using BWT: Details

- To compute intervals fast:
 - Slice blocks of $\log G$ rows
 - Maintain #a's, c's, g's, t's up to each block
- Position \neq Position after BWT along sequence



Suffix Array Representation

1 the_next_text_that_i_index.
2 he_next_text_that_i_index.t
3 e_next_text_that_i_index.th
4 _next_text_that_i_index.the
5 next_text_that_i_index.the_
6 ext_text_that_i_index.the_n
7 xt_text_that_i_index.the_ne
8 t_text_that_i_index.the_nex
9 _text_that_i_index.the_next
10 text_that_i_index.the_next_
11 ext_that_i_index.the_next_t
12 xt_that_i_index.the_next_te
13 t_that_i_index.the_next_tex
14 that_i_index.the_next_text_

15 that_i_index.the_next_text_
16 hat_i_index.the_next_text_t
17 at_i_index.the_next_text_th
18 t_i_index.the_next_text_tha
19 _i_index.the_next_text_that
20 i_index.the_next_text_that_
21 _index.the_next_text_that_i
22 index.the_next_text_that_i_
23 ndex.the_next_text_that_i_i
24 dex.the_next_text_that_i_in
25 ex.the_next_text_that_i_ind
26 x.the_next_text_that_i_inde
27 .the_next_text_that_i_index

Suffix Array: Sorted Running Index

27.the_next_text_that_i_index
19_i_index.the_next_text_that
21_index.the_next_text_that_i
4_next_text_that_i_index.the
9_text_that_i_index.the_next
14_that_i_index.the_next_text
17at_i_index.the_next_text_th
24dex.the_next_text_that_i_in
3e_next_text_that_i_index.th
25ex.the_next_text_that_i_ind
6ext_text_that_i_index.the_n
11ext_that_i_index.the_next_t
16hat_i_index.the_next_text_t
2he_next_text_that_i_index.t

20i_index.the_next_text_that
22index.the_next_text_that_i
23ndex.the_next_text_that_i_i
5next_text_that_i_index.the
18t_i_index.the_next_text_tha
8t_text_that_i_index.the_nex
13t_that_i_index.the_next_tex
10text_that_i_index.the_next
15that_i_index.the_next_text
1the_next_text_that_i_index.
26x.the_next_text_that_i_inde
7xt_text_that_i_index.the_ne
12xt_that_i_index.the_next_te

Another view point on finding the position

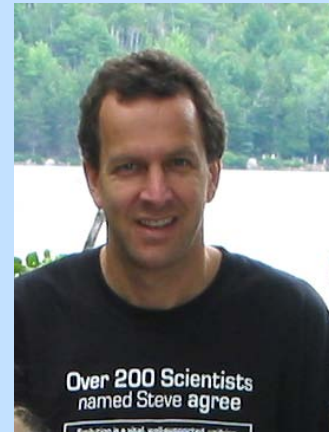
? .the_next_text_that_i_index
? _i_index.the_next_text_that
? _index.the_next_text_that_i
? _next_text_that_i_index.the
9 ← text_that_i_index.the_next
? _that_i_index.the_next_text
? at_i_index.the_next_text_th
? dex.the_next_text_that_i_in
? e_next_text_that_i_index.th
25 ex.the_next_text_that_i_ind
? ext_text_that_i_index.the_n
★ ? ext_that_i_index.the_next_t
? hat_i_index.the_next_text_t
? he_next_text_that_i_index.t

20 i_index.the_next_text_that
? index.the_next_text_that_i
? ndex.the_next_text_that_i_i
? next_text_that_i_index.the_
? t_i_index.the_next_text_tha
8 t_text_that_i_index.the_nex
? t_that_i_index.the_next_tex
? text_that_i_index.the_next
? that_i_index.the_next_text_
? the_next_text_that_i_index.
26 x.the_next_text_that_i_inde
? xt_text_that_i_index.the_ne
? xt_that_i_index.the_next_te

★ = 11

Sources

- Langmead, Trapnell, Pop and Salzberg., Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, *Genome Biology* '09
- Ferragina & Manzini, Opportunistic data structures with applications, *FOCS* '00
- Li & Durbin, Fast and accurate short read alignment with Burrows-Wheeler transform, *Bioinformatics*, '09
- Li et al., SOAP2: an improved ultrafast tool for short read alignment, *Bioinformatics*, '09



Burrows-Wheeler Transform

the not-so-gory details

Sources

Ferragina and Manzini, Opportunistic data structures with applications, FOCS 00

Langmead et al. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, Genome Biology 09



Suffix Arrays

- $T[1..u]$ text
 - Suffix array A of T : lexicographically ordered suffixes of T represented by pointers to their starting points
 - $T=acabc \rightarrow A=[3,1,4,2,5]$
 - Requires $4u$ bytes in practice
 - Can one compress suffix arrays?
- 1 acabc
2 cabc
3 abc
4 bc
5 c



Burrows-Wheeler Transform

Forward BWT: $T \rightarrow L$

- a. $T \rightarrow T\#$ (# unique smallest char)
- b. Form conceptual matrix M : all cyclic shifts of $T\#$, sorted lexicographically
- c. Transformed text L : the last column of T . $L = \text{BWT}(T)$

Notation: F : first column in M

Close connection to suffix arrays!



Key lemmas

- In the i -th row of M , $L(i)$ precedes $F(i)$ in the original text: $T = \dots L(i)F(i) \dots$
 - The i -th occurrence of char X in L corresponds to the same text character as the i -th occurrence of X in F
- “last-first (LF) mapping”



Backward BWT (1)

goal: $L \rightarrow T$

Σ - alphabet, lex ordered. # - unique end char.

Compute the array $C[1, \dots, |\Sigma|]$:

$C(c)$ is the total no. of chars $\{\#, 1, \dots, c-1\}$ in T

$Occ(c, r)$ = no. of occurrences of c in BWT up to but not including the element at index r

Alg Stepleft(r):

1. Return $C[BWT[r]]+1+Occ[BWT[r],r]$



Backward BWT (2)

goal: $L \rightarrow T$

- a. $r \leftarrow 1; T \leftarrow ""$
- b. While $BWT[r] \neq \$$ do
- c. $T \leftarrow$ prepend $BWT[r]$ to r
- d. $r \leftarrow \text{stepleft}[r]$
- e. Return T



Alg Exactmatch ($P[1,p]$)

goal: Find the interval $[sp,ep]$ of matrix rows that begin with the query $P[1,p]$

1. $C \leftarrow P[p]; sp \leftarrow C[c]+1; ep \leftarrow C[c+1]+1;$
 $i \leftarrow p-1$
2. While $sp < ep$ and $i \geq 1$ do
3. $c \leftarrow P[i]$
4. $sp \leftarrow C[c] + Occ(c,sp) + 1$
5. $ep \leftarrow C[c] + Occ(c,ep) + 1$
6. $i \leftarrow i-1$
7. Return sp, ep



Computing $Occ(c,r)$

1. Precompute $Occ(c,r)$ naively for each c,r

-

Time $O(|\Sigma|u)$, but space $O(|\Sigma|u \log u)$, for text of length u

2. Precompute $Occ(c,r)$ only for $r=j*p$

When $Occ(c,r)$ is needed - if r not available go back to the previous multiple of p and add

Time $O(|\Sigma|u)$ to preprocess, $O(p)$ to compute

Space $O((|\Sigma|u \log u)/p)$



Computing the Text location of an exact match

- Problem: Exactmatch P gives row(s) in M that begin with the query - need to find their offset - location in the text
- Solution:
 - mark some rows with pre-calculated offsets.
 - In search, if row is not marked, do Stepleft k times until finding a marked row with offset o ; report $o+k$
- Time/space tradeoff



Burrows & Wheeler

- David Wheeler (1927-2004)
 - Educated in Math, Cambridge
 - First PhD in the world in CS (51)
 - Invented the subroutine
 - Cambridge prof. active in cryptography
- Michael Burrows (~1963-)
 - PhD Cambridge
 - Worked in DEC, Microsoft, now Google
 - Co-developed the AltaVista search engine
- Burrows M, Wheeler D (1994), *A block sorting lossless **data compression** algorithm*, Technical Report 124, Digital Equipment Corporation



NGS algorithmics

3. Sequence assembly



What if we have no reference?

TTATGGTCGGTGAGTGTGACTGGTGTTGTCTAA

GGTCGGTGAG

TGAGTGTGAC

TGGTGTTGTC

TGACTGGTTT

AATGGTCGGT

GAGTGTGACT

Sequence Assembly

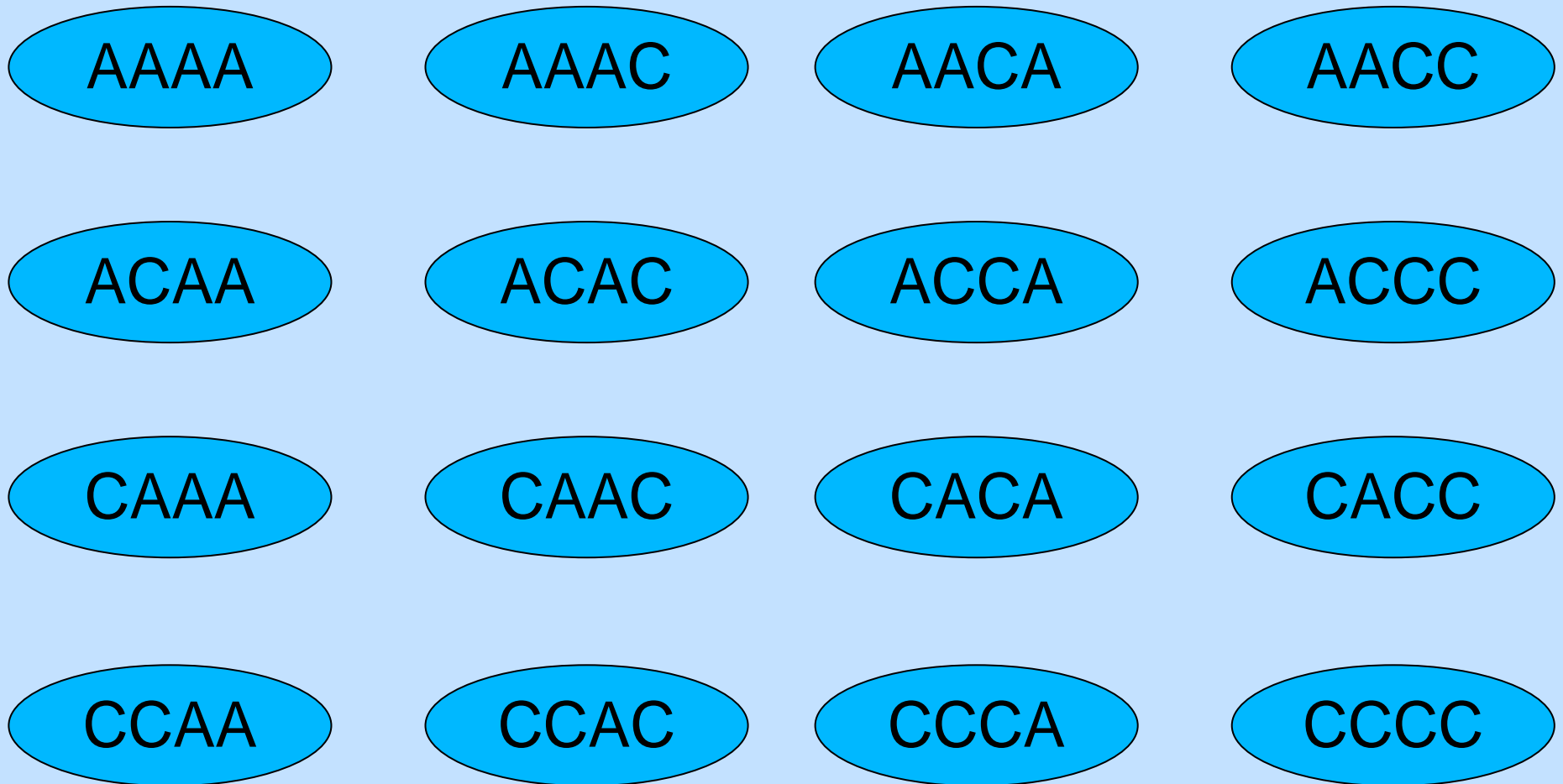
- **Input:**

GGTCGGTGAG
TGAGTGTGAC
TGGTGTTGTC
TGACTGGTTT
AATGGTCGGT
GAGTGTGACT
AAAAAAAAAA

- **Output:**

ATATGGTCGGTGAGTGTGACTGGTGTTGTCCTAA

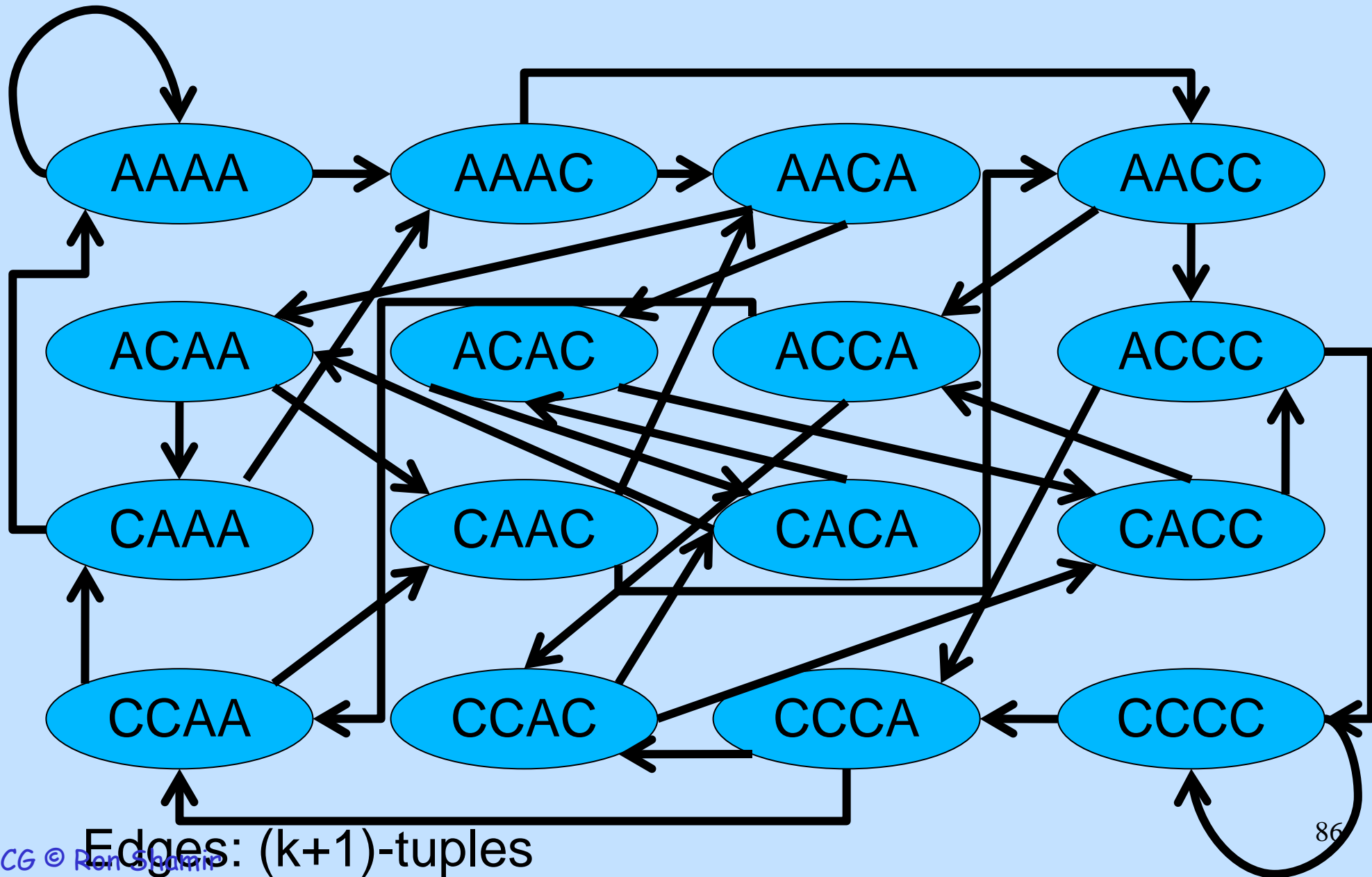
Idea: de Bruin Graph



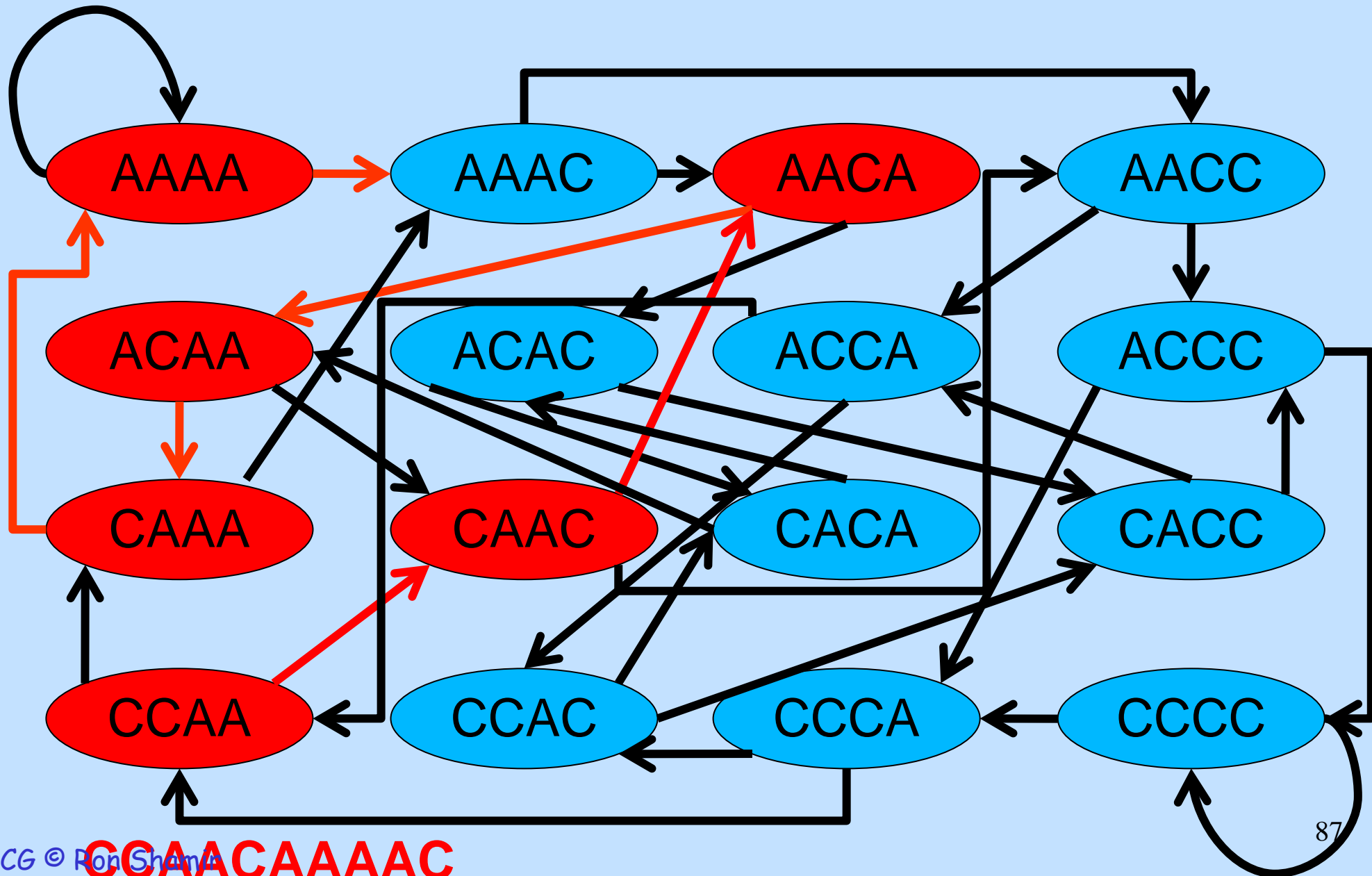
Nodes: k-tuples



Idea: de Bruin Graph



Sequence \Leftrightarrow Path in Graph



CGAACA AAAAC



Assembly Using de-Bruijn Graphs

Input:

GGTCGGTGAG

TGAGTGTGAC

TGGTGTTGTC

1. Turn reads to paths

GGTC → GTCG → TCGG → CGGT → GGTG → GTGA → TGAG

TGAG → GAGT → AGTG → GTGT → TGTG → GTGA → TGAC

TGGT → GGTG → GTGT → TGTT → GTTG → TTGT → TGTC

Assembly Using de-Bruijn Graphs

GGTC → GTCG → TCGG → CGGT → GGTG → GTGA → TGAG

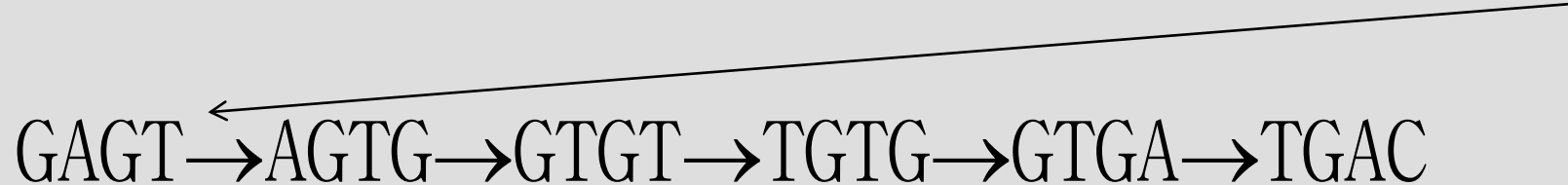
TGAG → GAGT → AGTG → GTGT → TGTG → GTGA → TGAC

1. Turn reads to paths

2. Merge paths

GGTC → GTCG → TCGG → CGGT → GGTG → GTGA → TGAG

GAGT ← AGTG → GTGT → TGTG → GTGA → TGAC



Assembly Using de-Bruijn Graphs

GGTC → GTCG → TCGG → CGGT → GGTG → GTGA → TGAG

GAGT ← AGTG → GTGT → TGTG → GTGA → TGAC

1. Turn reads to paths
2. Merge paths
3. Resolve error “bubbles”
4. Resolve cycles (repeats)

Bibliography:

- Zerbino & Birney, Velvet:
Algorithms for De Novo Short
Read Assembly Using De Bruijn
Graphs, *Genome Research* 2008

