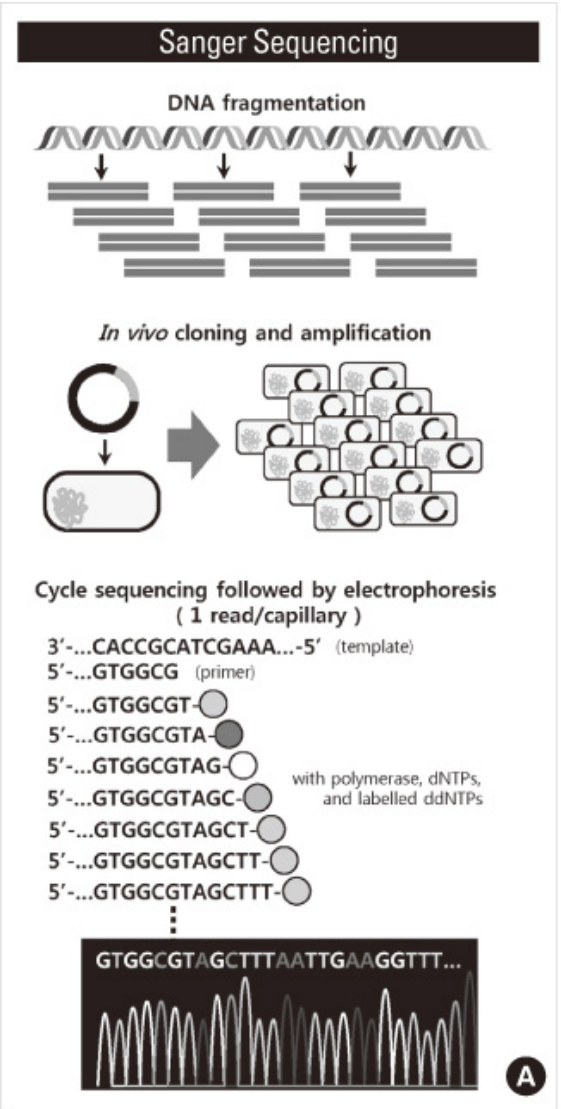
Wstęp do biologii obliczeniowej

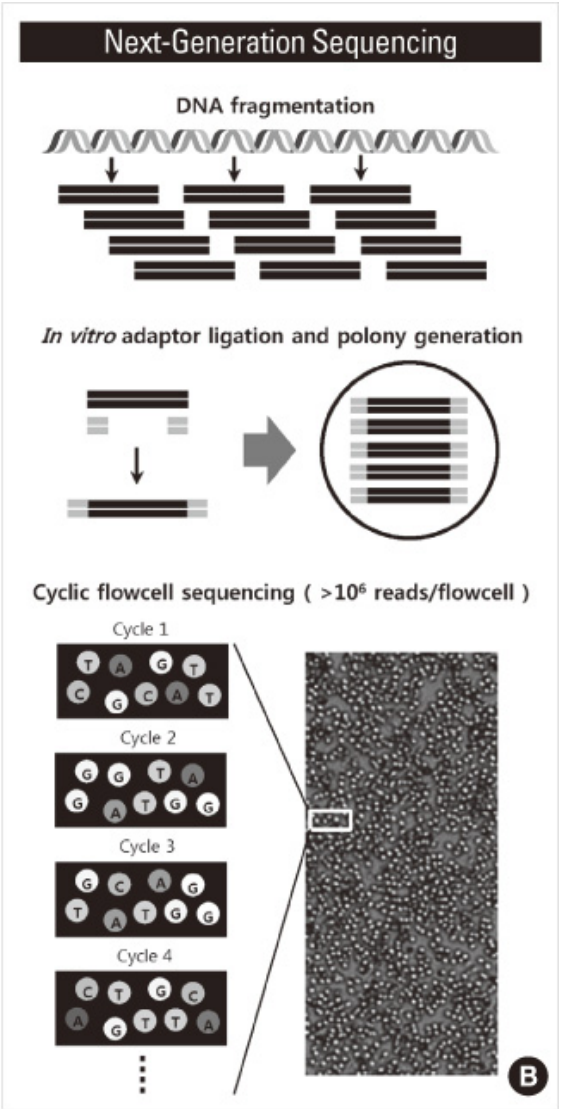
Wykład 12 -
Mapowanie krótkich odczytów DNA do genomu

Bartek Wilczyński

29 V 2018

Next Generation Sequencing





- NGS gives millions of short reads (30-200bp) instead of 1 longer read (up to few kb)
  - Desk-size devices,
  - costly chemistry (in 1000$ range for ~1TB of data)
  - error rates ~0.0001

Single molecule sequencing

- Oxford nanopore MinION on the ISS (Aug 2016)

- Single molecule sequencing is in the prototype phase – gives even longer reads (up to 100kb), but with large error rate (~10%)
- Small devices for single use are promised to cost below 1000$

How to map a short sequence to the genome?

- We frequently sequence DNA originating from a genome closely related to a known one (e.g. human patient samples, bacteria, viruses, etc)
- Even though they are closely related, they are not identical (remember, mutations?)
- Sequence reads are short (30-100), genomes are long (up to 10^10)
- Obviously we need faster methods than Dynamic programming

Text searching algorithms

- Exact searching (Knuth-Morris-Pratt, Boyer-Moore) : not applicable
- Many reads and one genome – we would like to index the genome to be able to process the reads quickly
- We need to take errors and variants into account, but hopefully not too many of them in a single read
- We should consider text indexes (Suffix trees, suffix arrays and Burrows-Wheeler transform)

Suffix tree

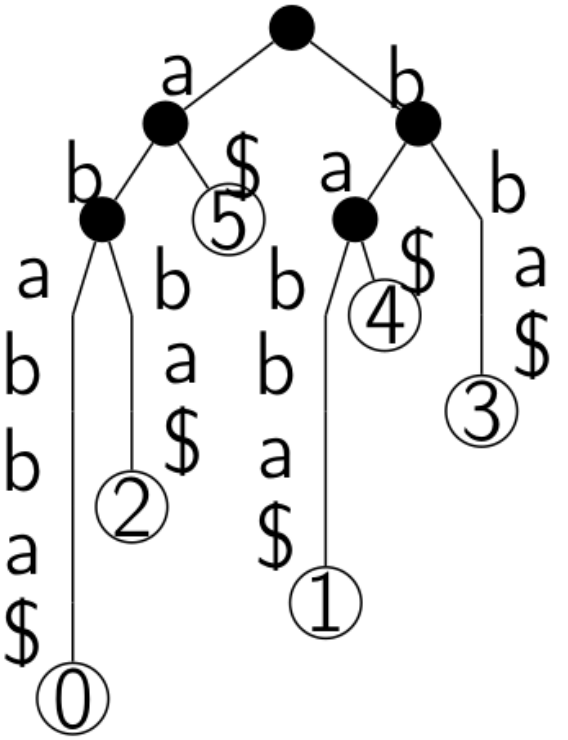
Suffix tree

- every edge is labelled with a text substring
- labels from consecutive edges on pathes from root to leafs constitute suffixes
- each suffix is represented in this way and corresponding leaf is labelled with its position in the text
- labels of sibling edges begin with different symbols

Index size: $\geq 10 \cdot |T|$ bytes

Matching time: $\mathcal{O}(|P| + |occurences|)$

Suffix tree for text ababba



Suffix array

Suffix array contains starting positions of lexicographically ordered suffixes

position	suffix	SA entry
0	ababba$	SA[0]=0
2	abba$	SA[1]=2
5	a$	SA[2]=5
1	babba$	SA[3]=1
4	ba$	SA[4]=4
3	bba$	SA[5]=3
6	$	SA[6]=6

Suffix array for text ababba

Index size: $4 \cdot |T|$ bytes

Matching time: $\mathcal{O}(|P| \cdot \log |T| + |occurences|)$

with additional LCP table

Index size: $5 \cdot |T| - 8 \cdot |T|$ bytes

Matching time: $\mathcal{O}(|P| + \log |T| + |occurences|)$

Burrows-Wheeler transform

Burrows-Wheeler transform
contains symbols predecessing
lexicographically ordered suffixes.

$$\mathrm{BWT}[i] = T[\mathrm{SA}[i] - 1]$$

Burrows-Wheeler transform for
text ababba

position	suffix	BWT entry
0	ababba$	BWT[0]=$
2	abba$ab	BWT[1]=b
5	a$ababb	BWT[2]=b
1	babba$a	BWT[3]=a
4	ba$abab	BWT[4]=b
3	bba$aba	BWT[5]=a
6	$ababba	BWT[6]=a

Last-to-first mapping

Cyclic shifts
of text ababba$

i	F L	SA[i]	LF[i]
0	ababba$	0	6
1	abba$ab	2	3
2	a$ababb	5	4
3	babba$a	1	0
4	ba$abab	4	5
5	bba$aba	3	1
6	$ababba	6	2

Last-to-first mapping

$LF(i)$ is the position in column F of the i-th symbol of column L.

Observation

$$SA[i] = SA[LF(i)] + 1$$

Corollary

$$SA[i] = SA[LF^k(i)] + k$$

Computing last-to-first mapping

Cyclic shifts
of word ababba$

i	F	L
0	ababba$	
1	abba$ab	
2	a$ababb	
3	babba$a	
4	ba$abab	
5	bba$aba	
6	$ababba	

Observation

Occurences of symbol x in columns F and L are ordered accordingly.

Proof

The order is determined by suffixes following occurences of x.

$C(x)$ number of occurences of symbols lexicographically smaller than x in T

$Occ(x, i)$ number of occurences of symbol x in $BWT[0 : i]$

Observation

$$LF(i) = C(BWT[i]) + Occ(BWT[i], i)$$

Extracting text

Structure for extracting text

- Burrows-Wheeler transform of T
- array C
- regularly sampled values of arrays $Occ(x, \_)$
- array with regularly sampled values of SA^{-1}

Algorithm

```
1:  function EXTRACT(begin, end)
2:      p ← cache[⌈end / CacheEvery_text⌉]        ▷ Get the closest cached position after end
3:      dist ← end − end mod CacheEvery_text
4:      while dist > 0 do                          ▷ LF-map to the end position
5:          p ← LF(p)
6:      end while
7:      dist ← end − begin
8:      result = ϵ
9:      while dist > 0 do                          ▷ LF-map and extract next begin − end characters
10:         result = BWT[p] + result               ▷ Prepend current character to the result
11:         p ← LF(p)
12:     end while
13:     return result
14: end function
```

Backward searching

Structure for backward searching

- Burrows-Wheeler transform of T
- array C
- regularly sampled values of arrays $Occ(x, \_)$

Algorithm

1: **function** FIND($Q_{1..m}$)
2: $sp \leftarrow C(Q_m)$
3: $ep \leftarrow C(Q_m + 1) - 1$
4: **for** $i \leftarrow m - 1, 1$ **do**
5: $sp = C(Q_i) + Occ(Q_i, sp - 1) + 1$
6: $ep = C(Q_i) + Occ(Q_i, ep)$
7: **if** $ep > sp$ **then**
8: break ▷ No matches, jump out
9: **end if**
10: **end for**
11: **return** (sp, ep) ▷ The opaque result is just a range in the BWT array
12: **end function**

Suffix indexes

Suffix tree suffixes = paths from root to leaves

- index size: $\geq 10 \cdot |genome|$ bytes
- exact mapping time: $\mathcal{O}(|read| + |occurences|)$

Suffix array lexicographic order on suffixes

- index size: $\geq 4 \cdot |genome|$ bytes
- exact mapping time: $\mathcal{O}(|read| \cdot \log |genome| + |occurences|)$

FM index self-index based on Burrows-Wheeler transform

- index size: $< 1 \cdot |genome|$ bytes (including sequence!)
- exact mapping time: 2-1000$\times$ slower than suffix arrays

Operations in Ferragina-Manzini index

$\text{Find}(Q) \rightarrow R$ searches for all occurrences of sequence Q and returns an opaque result R that can be used with other operations.

$\text{FindSuffixes}(Q_{1..m}) \rightarrow R_{1..m}$ works just like Find, but returns results for each suffix of Q so that R_i is the result of searching for $Q_{i..m}$.

$\text{FindContinue}(Q_{1..m}, R_{old}, f) \rightarrow R_{new}$ just like Find searches for all occurrences of $Q_{1..m}$, but takes advantage of an earlier result R_{old}, assumed to be obtained by searching for $Q_{f..m}$, and returns a new result R_{new}.

$\text{Count}(R) \rightarrow k$ returns the number of occurrences k represented by R.

$\text{Locate}(R) \rightarrow l_{1..k}$ returns locations of occurrences represented by R.

$\text{Extract}(b, l) \rightarrow S$ retrieves a subsequence of the reference sequence T: $S = T[b..b + l - 1]$.

Bowtie (Langmead et al. '09)

Seed – high-quality part of the read (default: first 28bp)

Policy

Search for read occurences in the genome with

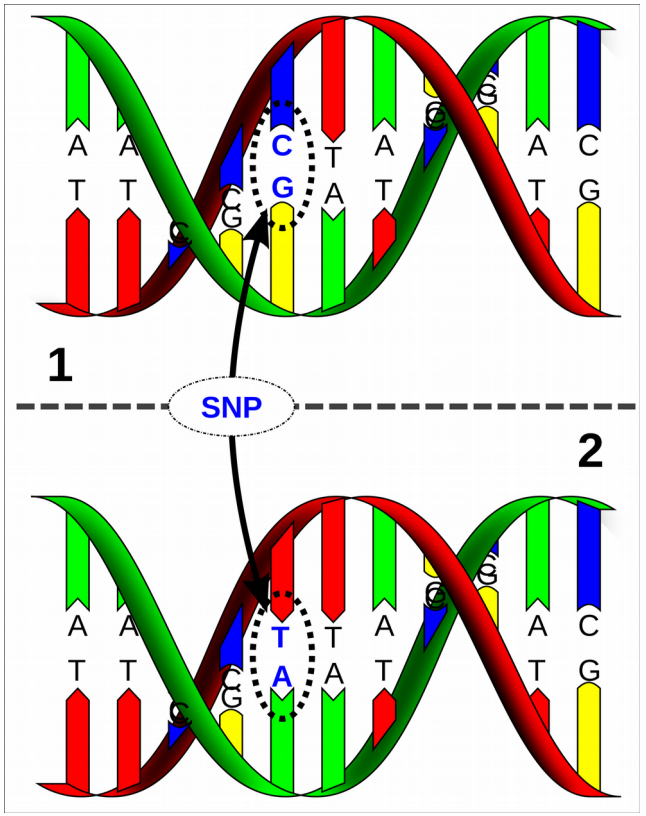
- limited number of errors in the seed (default: first 28bp),
- limited sum of quality values of mismatched positions in the whole read.

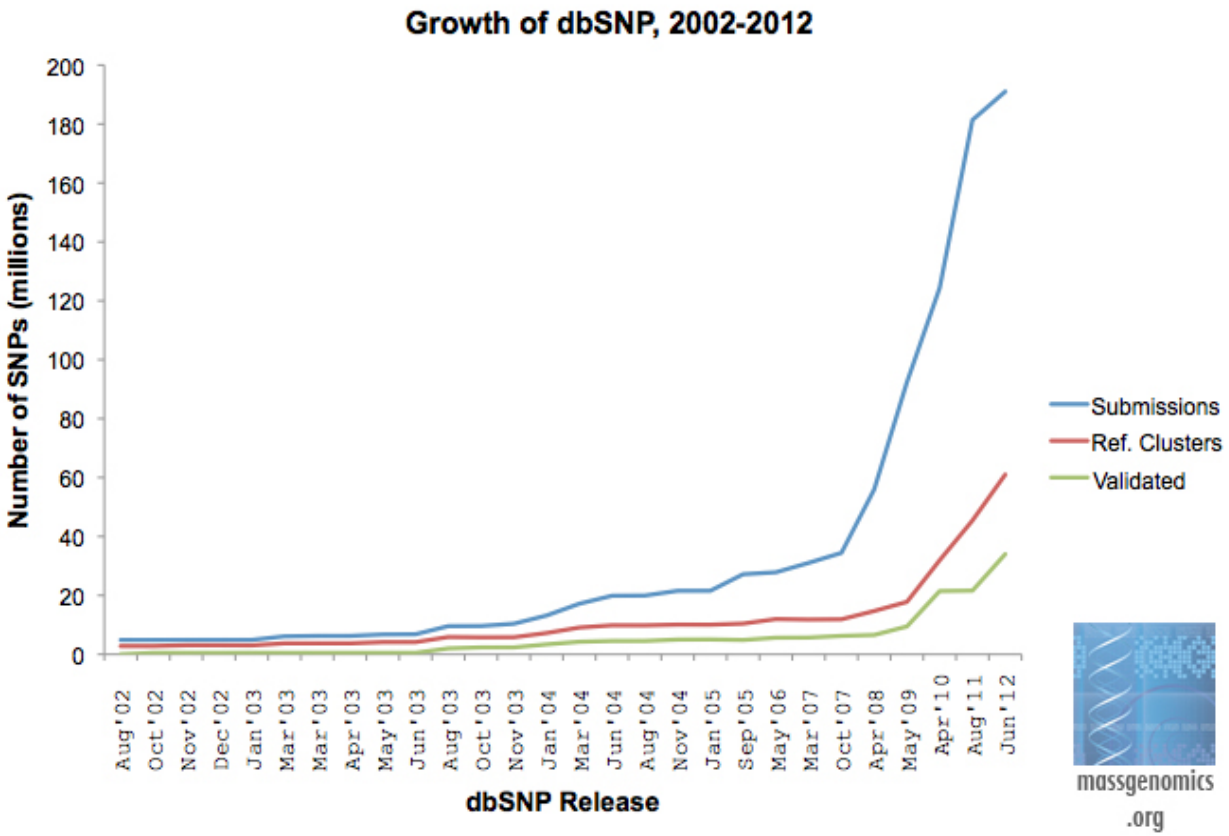
Algorithm

- Genome index is searched with k-neighborhood of the seed of a read.
- Located occurences are extended to whole read mappings and the quality criterion is checked.

Something about SNPs

- Single nucleotide polymorhism (SNP) a position in the genome where a natural variation in population occurs

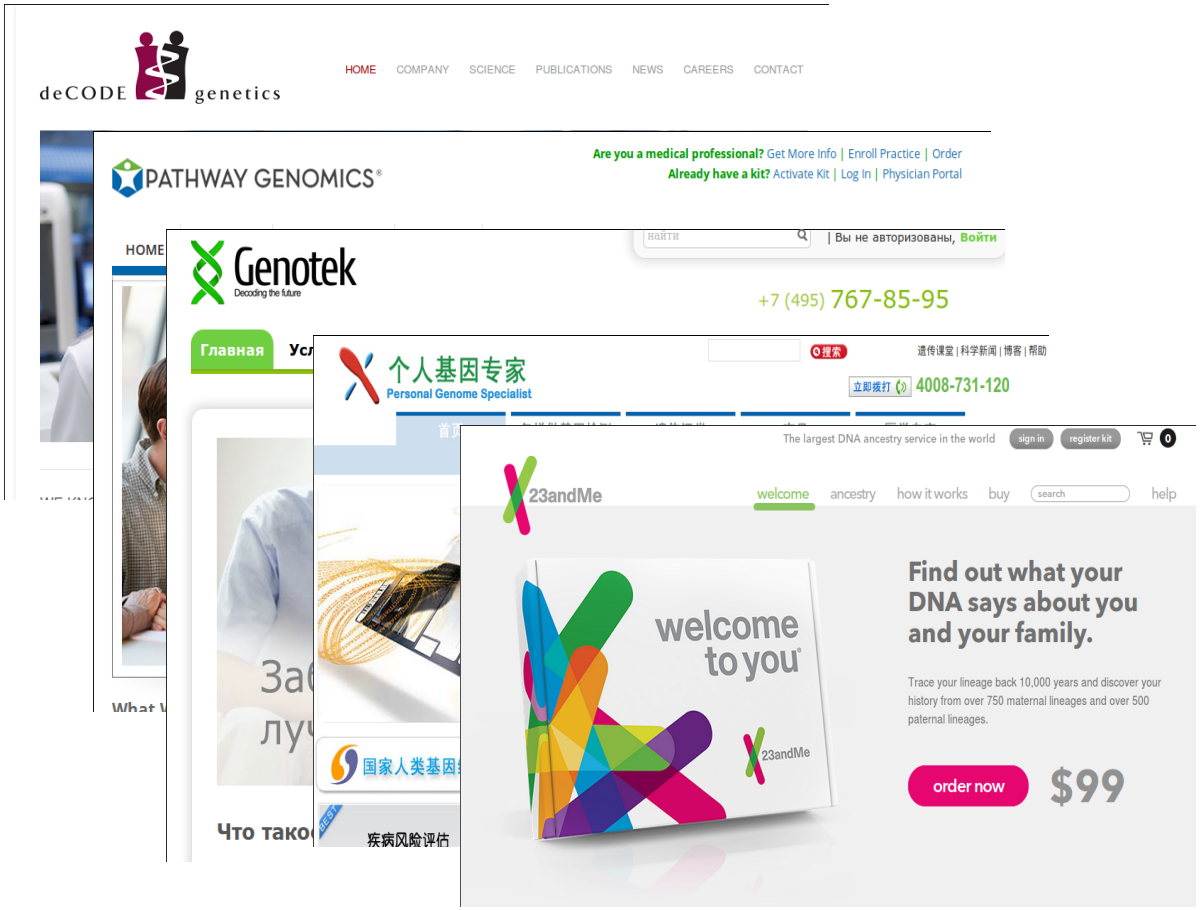




massgenomics.org

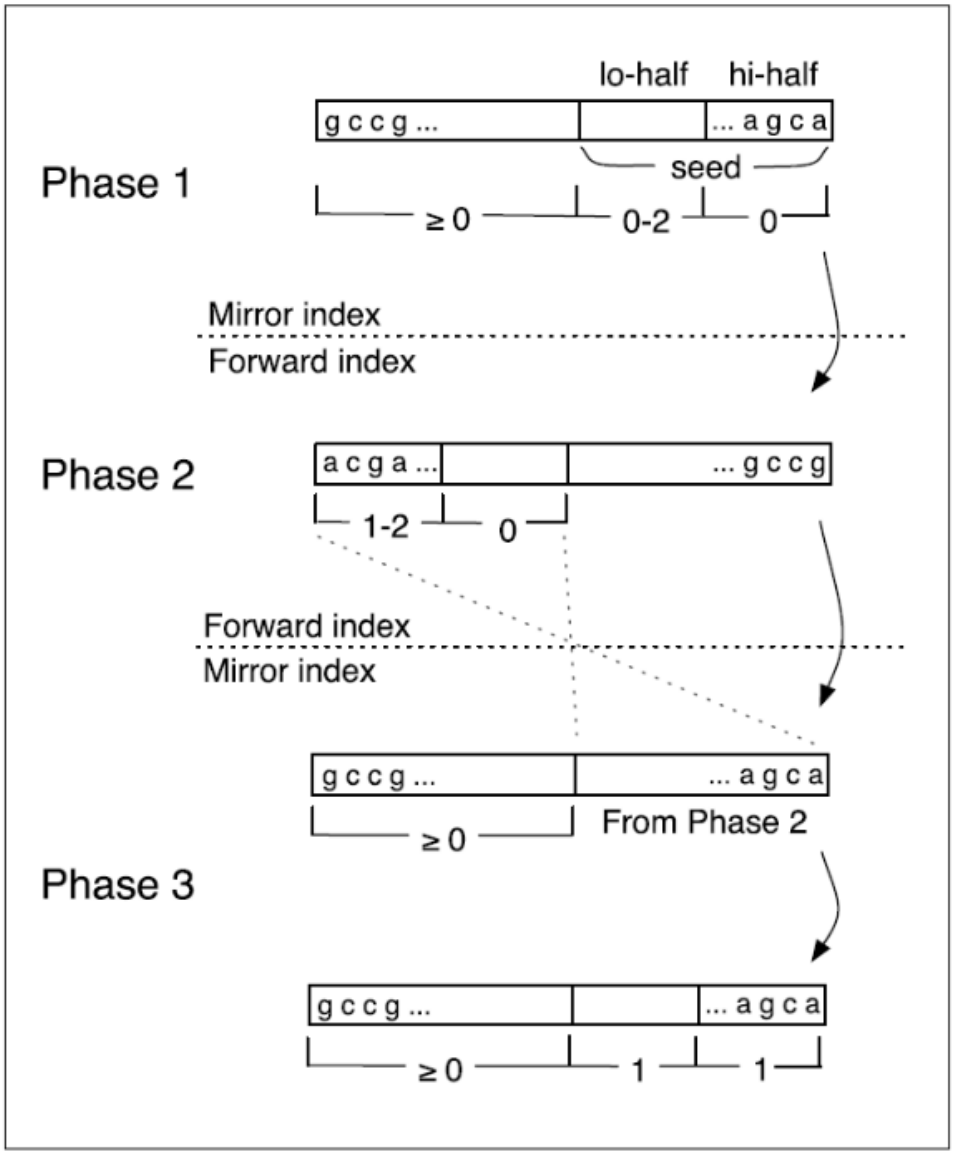
Genotyping vs. Sequencing

- Many commercial services offer genotyping (usually not sequencing) for very low prices
- Some of this information might be important if you are sick
- Most of the information provided by such companies is pure noise and correlative data
- Data security is a big issue



Bowtie – avoiding excessive backtracking

- $k \leq 3$
- Double indexing:
  FM-index is build for a genome sequence (*forward* index) and for a reverse sequence (*mirror* index).



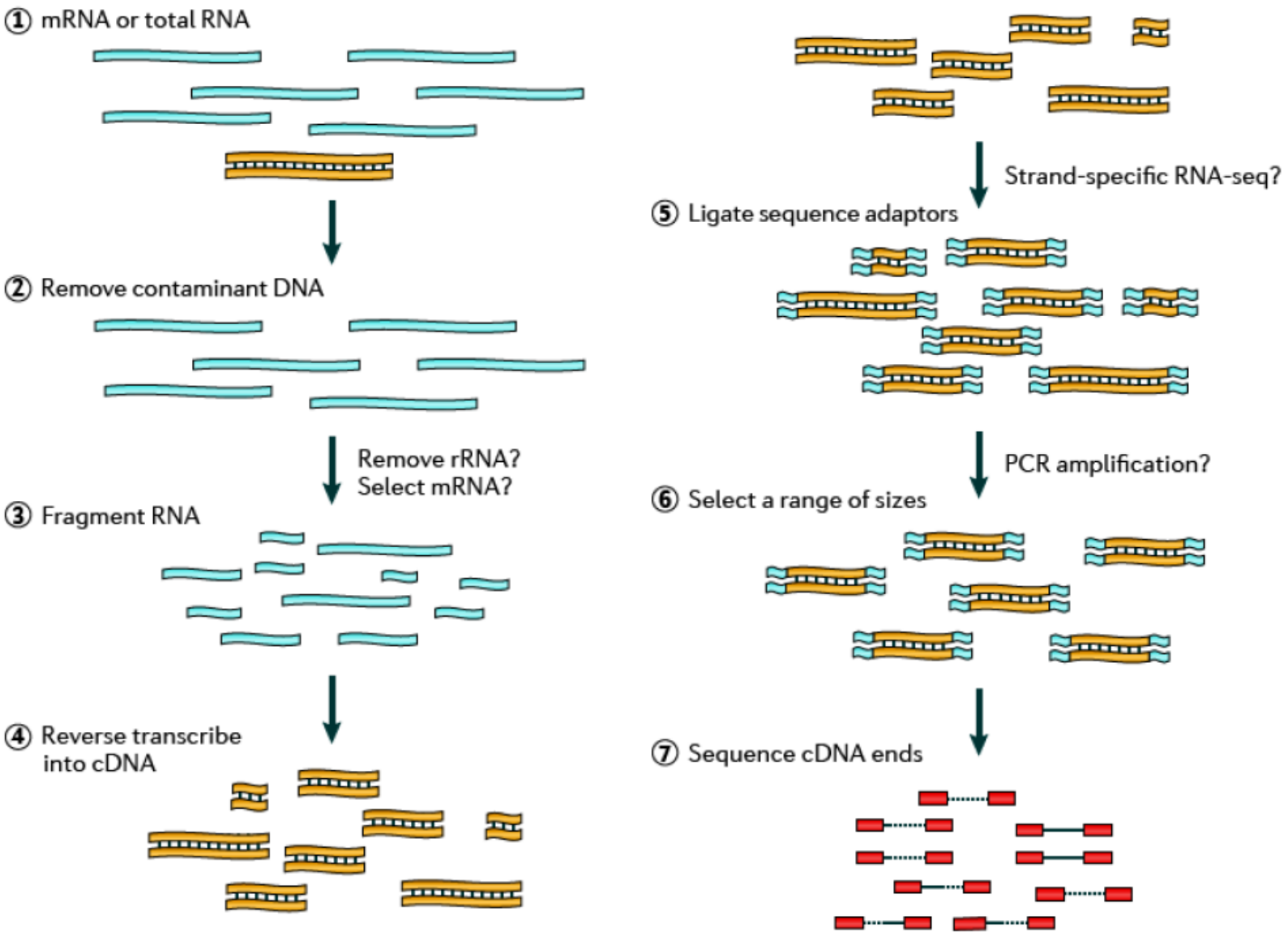
BWT mapping summary

- Effective tools are used in short read mapping using BWT and FMI
- Index can be linear in genome size and match finding with small (<3) number of mismatches is feasible
- Large number of mismatches works against these methods

Even faster read mapping?

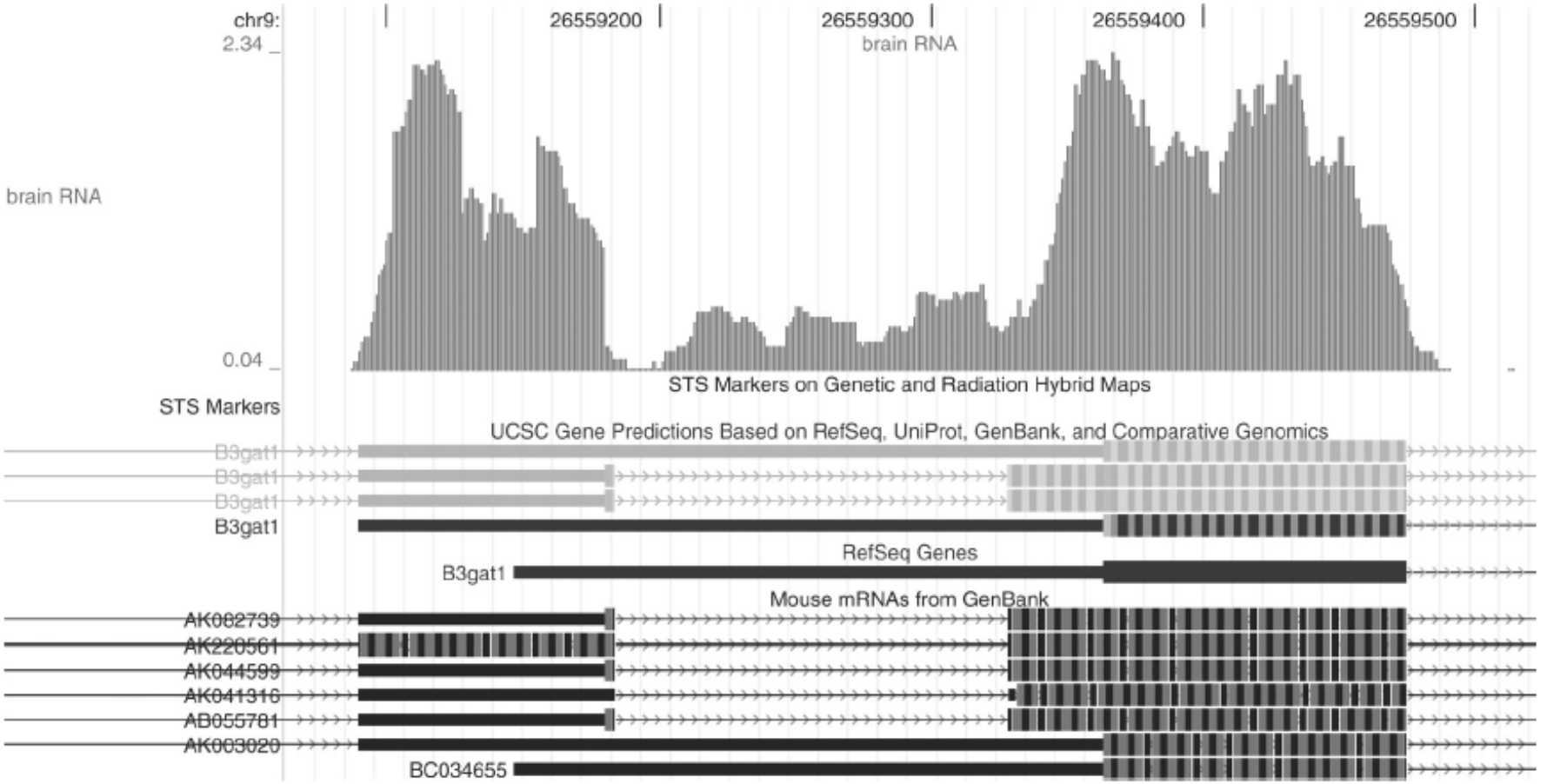
- Sometimes we can agree to a worse mapping efficiency (some random reads not mapped) if it increases the speed of overall mapping
- This is in particular true in cases where we want to count reads rather than identify the variants
- One such case is mRNA expression profiling, when we are interested in relative abundances of fragments of the reference sequence

RNA-seq data preparation



J. A. Martin and Z. Wang *Next-generation transcriptome assembly*. Nature Reviews 2011.

RNAseq Reads mapped to the genome



STAR – ultrafast read mapping (Dobin et al. 2012)

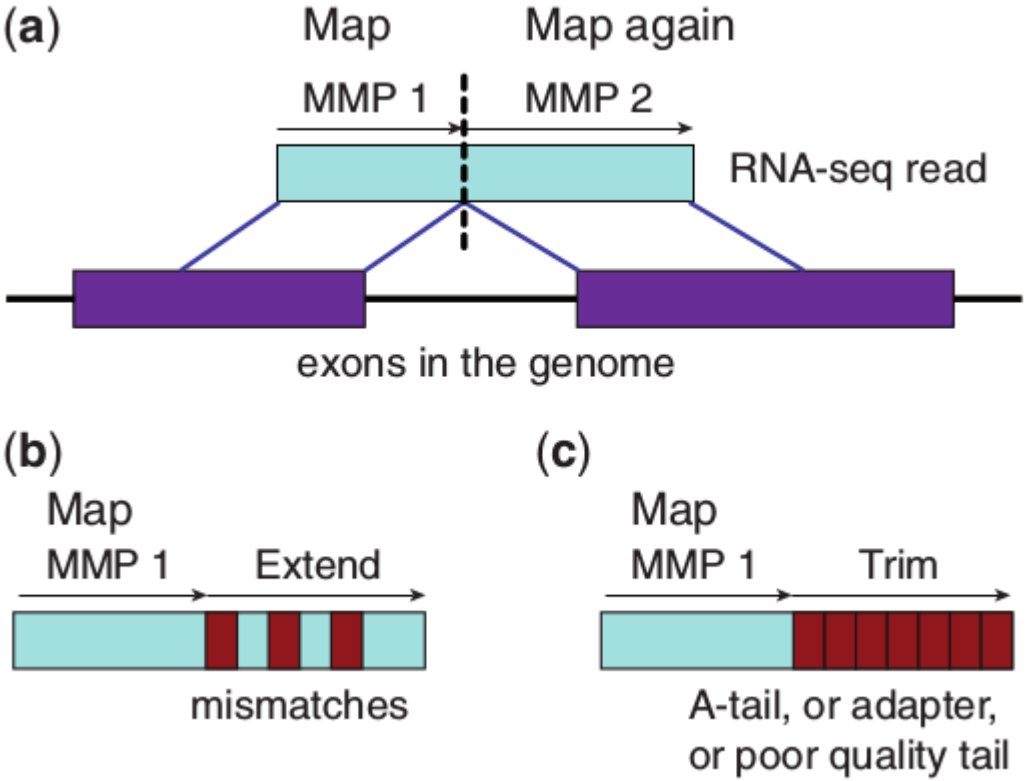
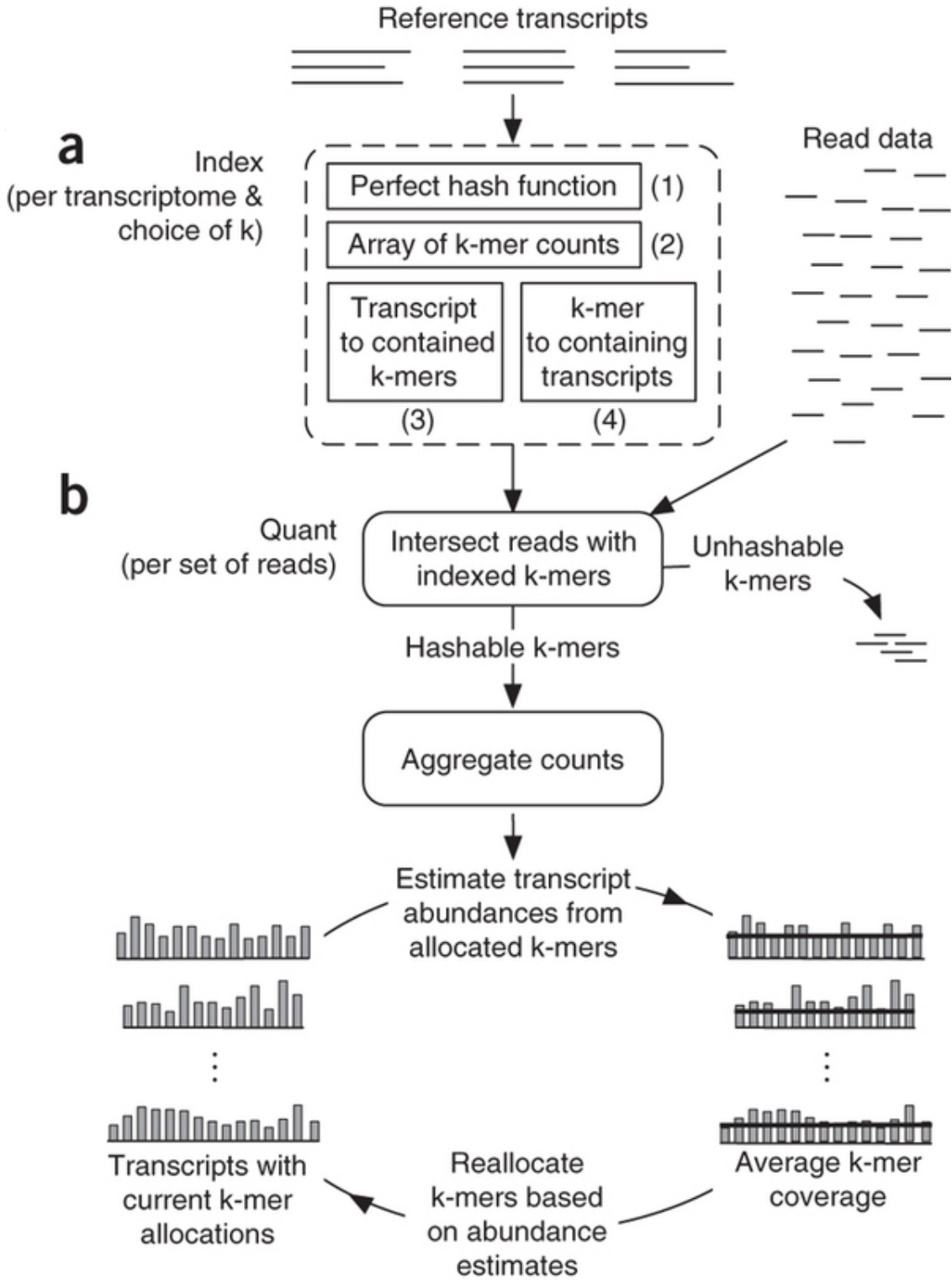


Table 1. Mapping speed and RAM benchmarks on the experimental RNA-seq dataset

Aligner	Mapping speed: million read pairs/hour		Peak physical RAM, GB	
	6 threads	12 threads	6 threads	12 threads
STAR	309.2	549.9	27.0	28.4
STAR sparse	227.6	423.1	15.6	16.0
TopHat2	8.0	10.1	4.1	11.3
RUM	5.1	7.6	26.9	53.8
MapSplice	3.0	3.1	3.3	3.3
GSNAP	1.8	2.8	25.9	27.0

Alignment free RN quantitation

- Sailfish method (Patro et al. 2014)
- We can simply count unique k-mers in the reads and use only those to quantify transcripts
- 25x speed improvement, without much loss in accuracy



Kallisto - even faster quantitation

- Kallisto method (Bray et al. 2015)
- Introducing a graph of overlapping k-mers for the different transcripts as an index
- Better implementation gives another 10x speed improvement