

# Sailfish: Rapid Alignment-free Quantification of Isoforms from RNA-Seq Reads

Rob Patro

Joint work with Carl Kingsford (CMU) and Steve Mount (UMD)

Why study gene expression?

A genome tells us a lot about an organism

small & large mutations can effect phenotype

a lot of work tying such mutations to e.g. disease

but ... the picture is still incomplete:

DNA mostly static

Same genome  $\Rightarrow$  same phenotype

Different env. / condition / tissue effect gene expression

Genome effects itself in complicated ways we can't always predict

Usually interested in protein abundance

# Proteins are the workhorses of the cell

They perform most cellular functions

Often, different protein levels  $\Rightarrow$  different function

*but . . . measuring proteins directly is difficult*

# “central dogma” of molecular biology

DNA  $\Rightarrow$  (m)RNA  $\Rightarrow$  Protein  
transcription translation

Usually interested in protein abundance

Proteins are the workhorses of the cell

They perform most cellular functions

Often, different protein levels  $\Rightarrow$  different function

*but . . . measuring proteins directly is difficult*

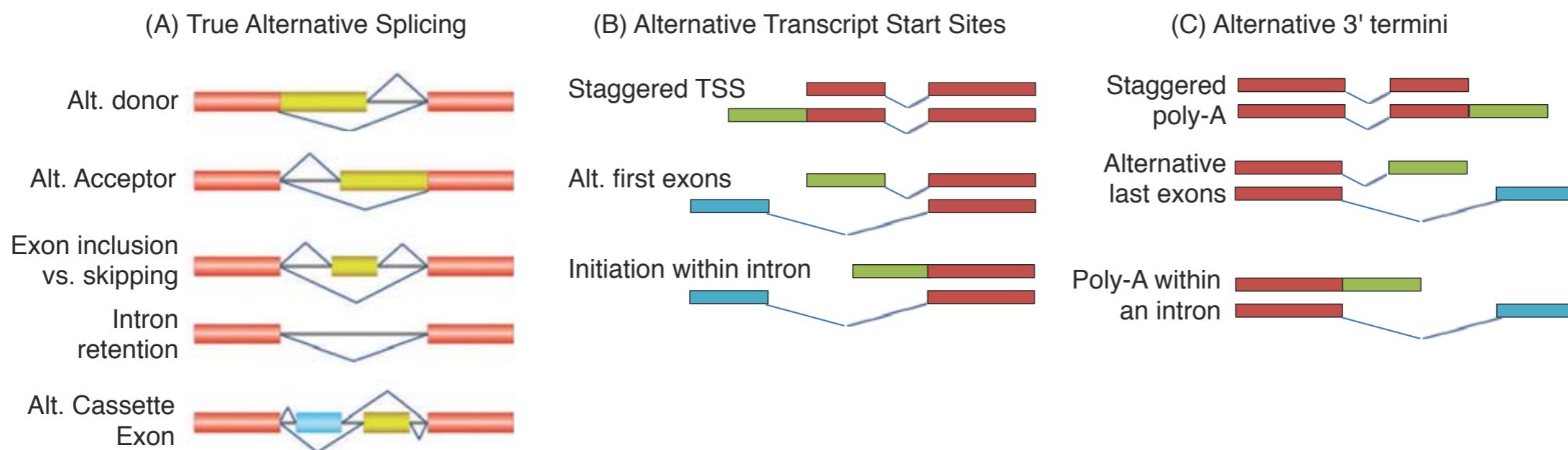
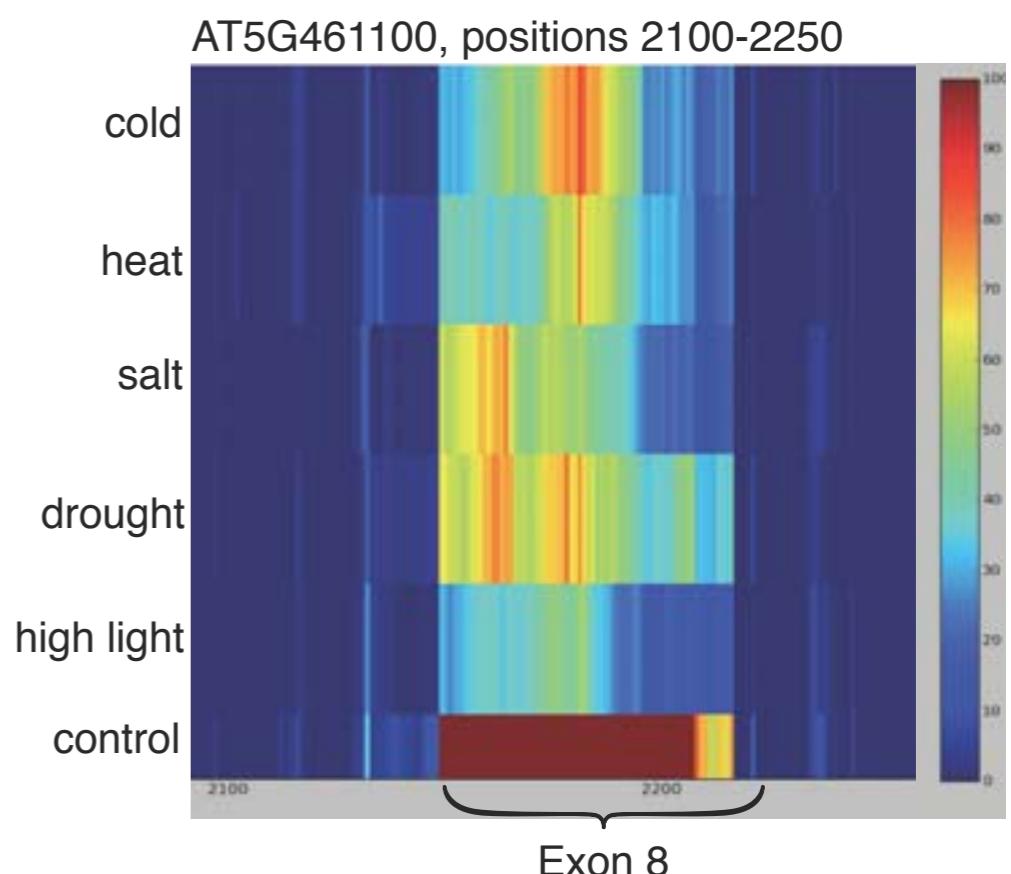
“central dogma” of molecular biology



RNA abundance can tell us a lot about protein abundance

# Alternative Splicing & Isoform Expression

- Sub-sequences of expressed genes can be sampled via RNA-seq (sequencing transcripts)
- Sequencing gives you short (35-300bp length “reads”)
- One gene ⇒ many different variants (called isoforms)



# The Isoform Expression Estimation Problem

- RNA-Seq now standard for gene and isoform expression estimation.
- A main use for transcriptome sequencing is estimating gene and isoform abundance.
- This leads to the following computational problem:

**Given:**

- Collection of RNA-Seq reads
- A set of known transcript sequences

**Estimate:**

- The relative abundance of each transcript

# The Standard Paradigm

- **Map** reads to transcripts using, e.g., Bowtie, BWA, etc.
  - Hundreds of millions of “patterns” in a large “text”
  - Inexact multi-pattern search
  - Tells us where a read *could* have come from
- **Shuffle** ambiguously mapped reads around, usually with the goal of uniform coverage.
  - If a read could have come from many places, we need to assign one
  - Under random sampling, transcript should have ~ uniform “coverage”

# The Standard Paradigm

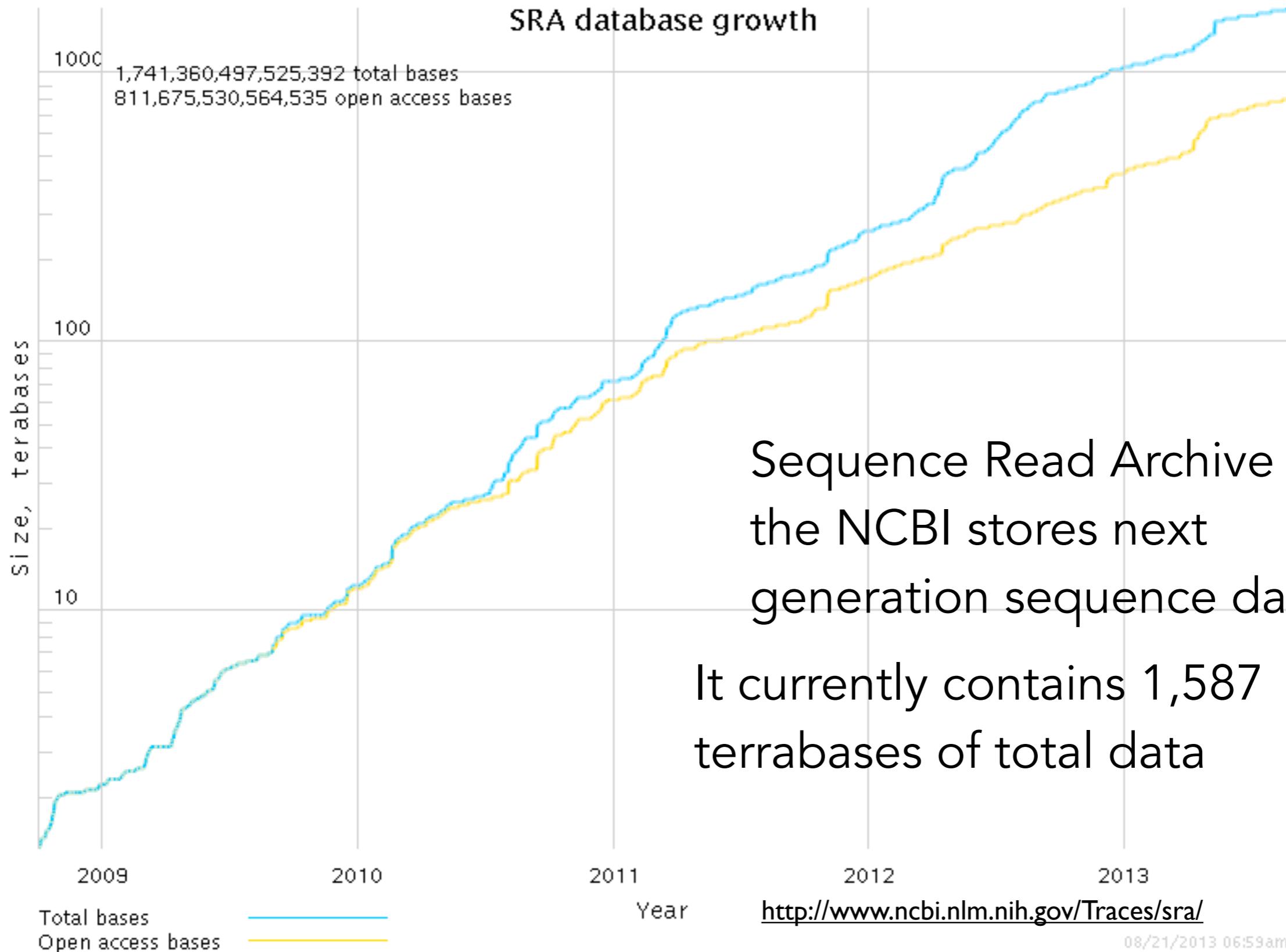
## Given assignment of reads:

- Estimate abundance via Reads Mapped Per Kilobase Per Million Reads (RPKM) [Mortazavi et al., 2008] or FPKM [Trapnell et al., 2010]
- Main problem with this approach: mapping and “shuffling” step can be very computationally intensive.

For example:

Program	Reference	Time
RSEM	Li et al., BMC Bioinformatics, 2011	days or hours
eXpress	Roberts et al., Nature Methods, 2012	days or hours
Cufflinks	Trapnell et al., Nature Biotech, 2010	days or hours

# Big Genomic Data



# Why Speed is Important

- RNA-Seq data collection will take days or weeks, but is massively parallel.
- Why is it important to estimate expression with low computational resources?
  1. Try many parameters, bias-correction techniques, filterings to gain confidence in estimates
  2. Apply to hundreds of experimental conditions
  3. Personalized medicine starting to use RNA-seq as a diagnostic technique.
  4. Start to think of the RNA-seq estimation step as an easy building block in a larger pipeline.
  5. Kant's categorical imperative: if everyone didn't care about speed, everything would be slow.

# Main idea behind Sailfish

Read mapping is unnecessary:

Replace **inexact** pattern search with **exact** sub-pattern **counting**

Exact sub-pattern is a k-mer (substring of length k)

**ATTCGACAGTAGCCATGACTGG**  
\_\_\_\_\_ ... \_\_\_\_\_

String of length N contains  $N-k+1$  k-mers

We know all meaningful sub-patterns ahead of time

If a k-mer doesn't appear in any transcript, it won't affect quantification

## The Standard Paradigm

Pre-process transcripts  
(e.g. build BWT)

Align reads to transcripts

Shuffle / allocate reads

Compute abundance

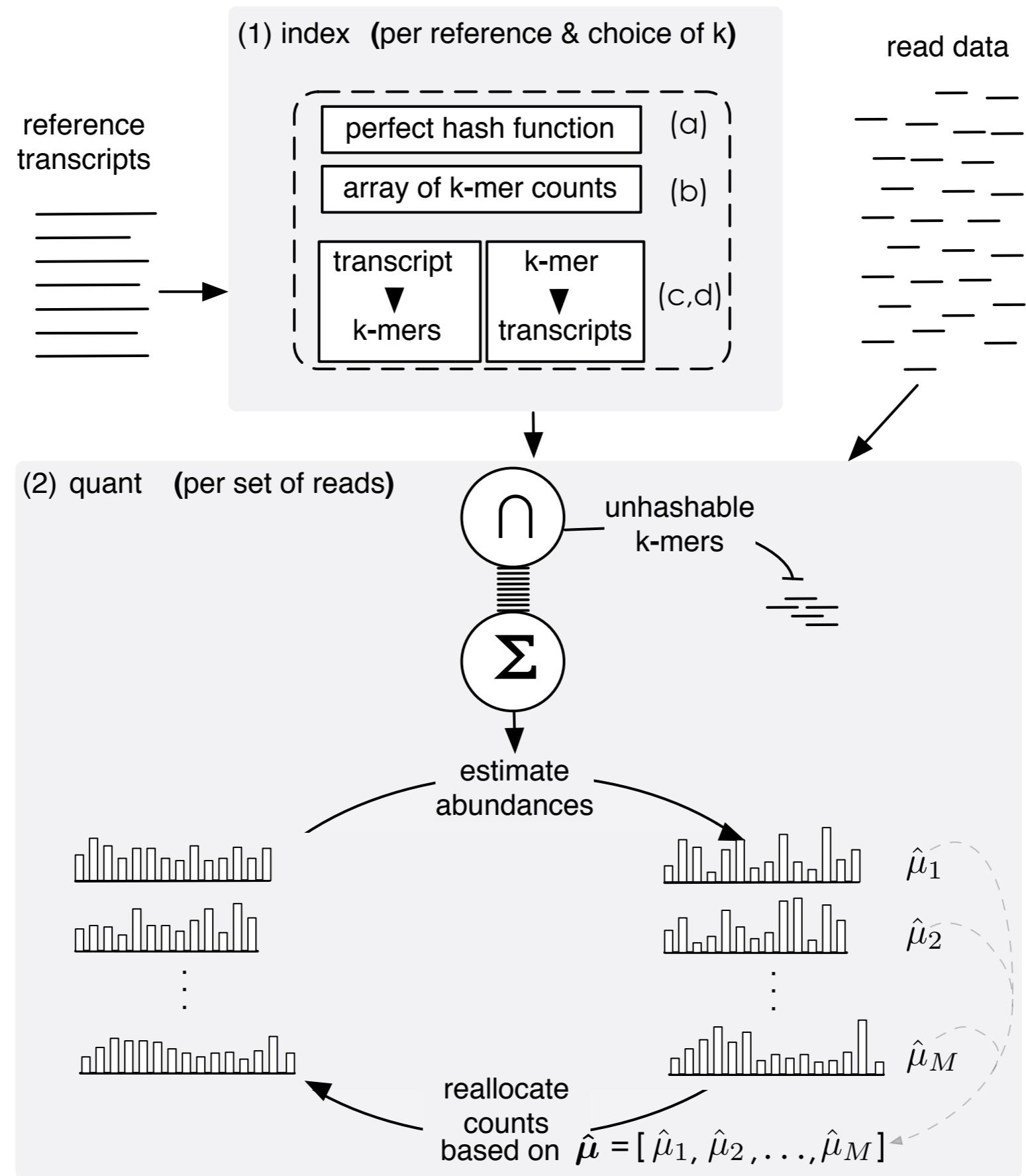
## Sailfish (Lightweight) Paradigm

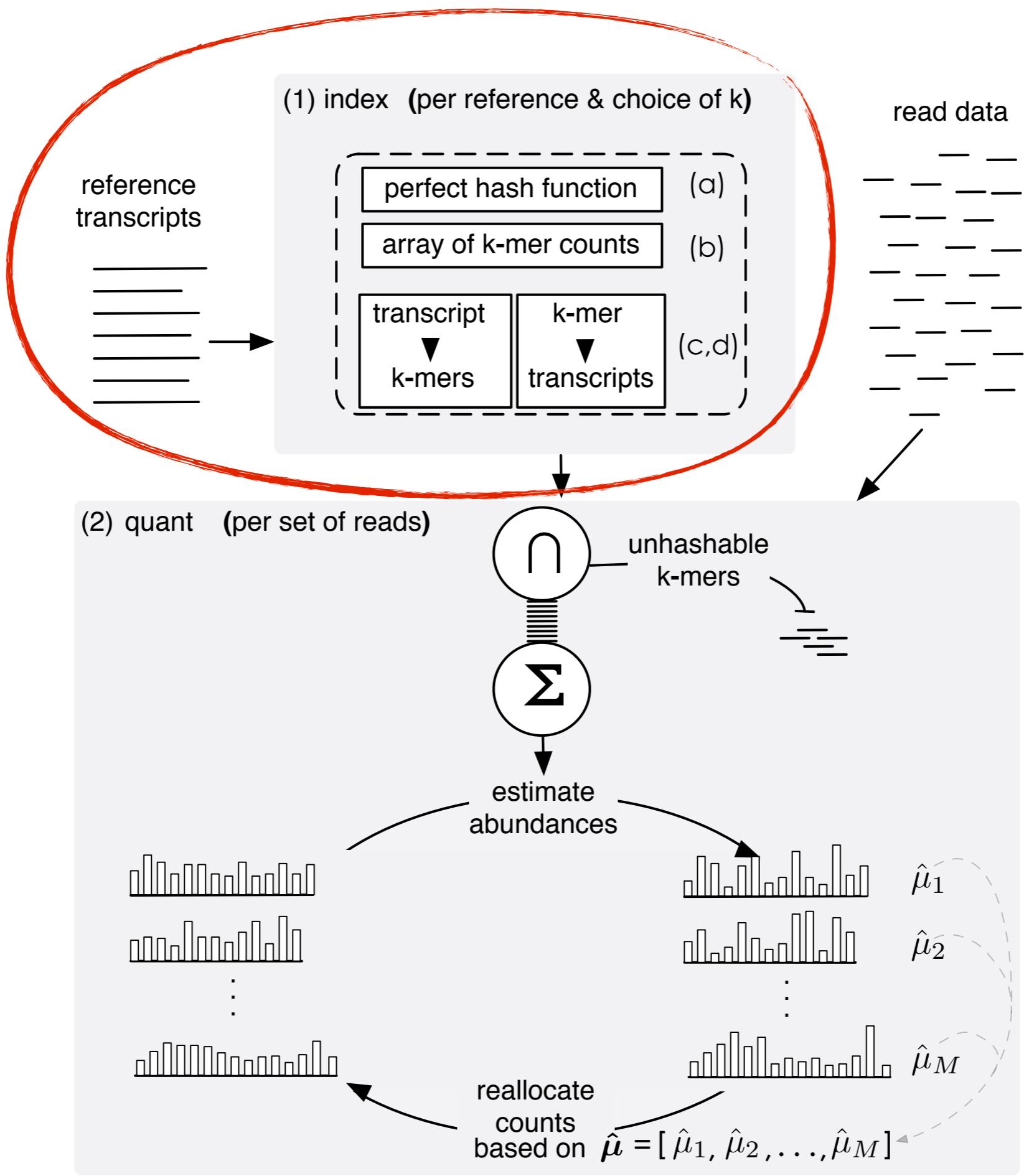
Pre-process transcripts  
(e.g. build k-mer index)

Count k-mers in reads

Shuffle / allocate k-mers

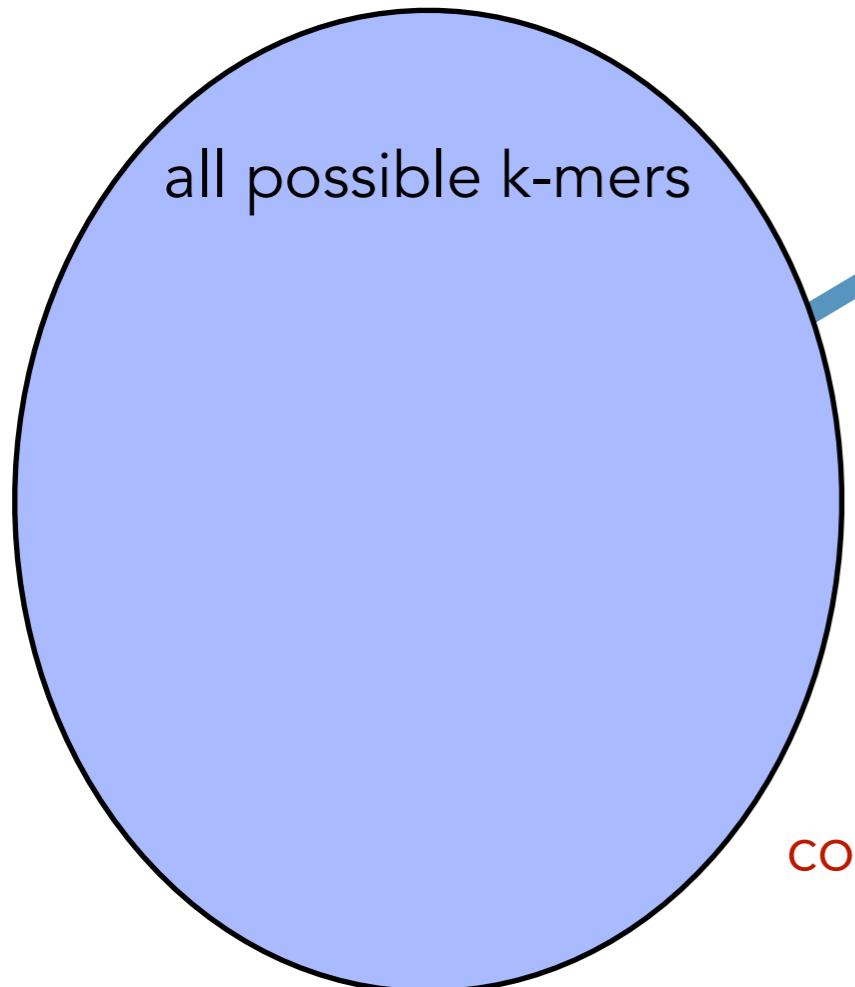
Compute abundance





# Construction of a Perfect Hash Transcript Index

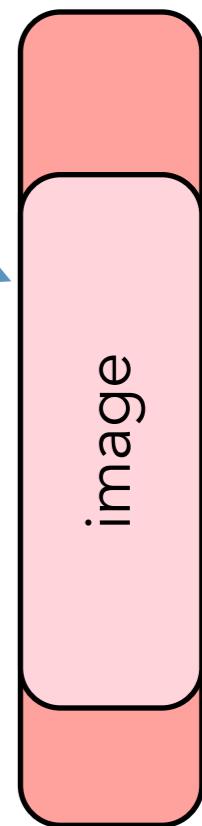
Domain (e.g. kmers)



Range (e.g. [0, m|D|])

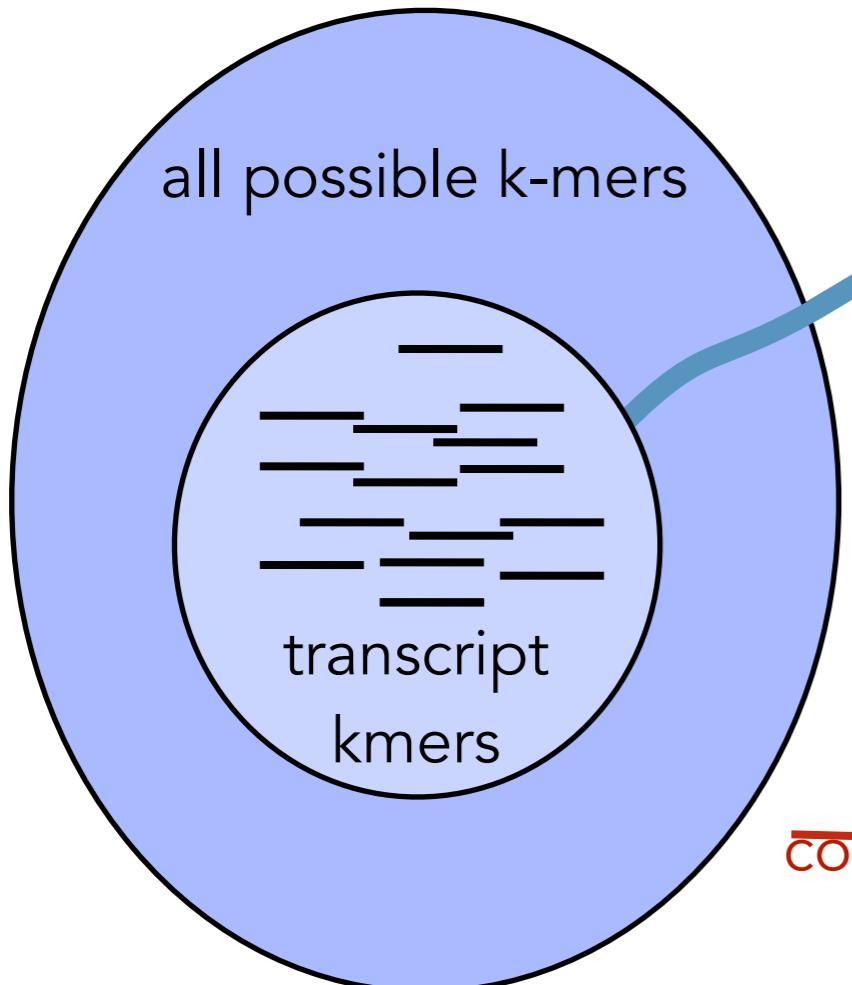
Hash  
maps elements from the domain  
into the range

collisions - different keys with same value



# Construction of a Perfect Hash Transcript Index

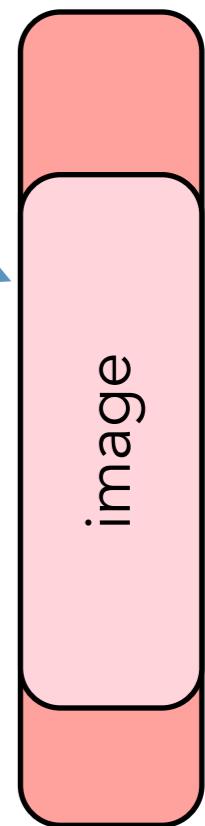
Domain (e.g. kmers)



Range (e.g. [0, m|D|])

Perfect Hash

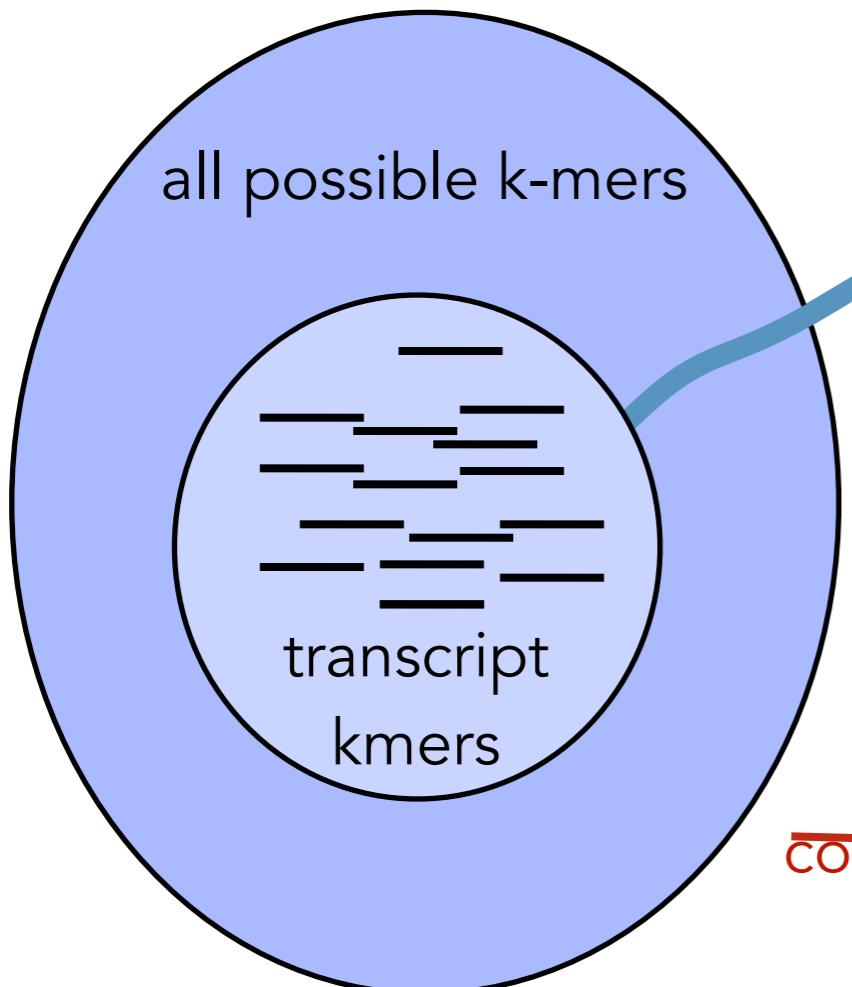
maps elements from the domain  
into the range



~~collisions - different keys with same value~~

# Construction of a Perfect Hash Transcript Index

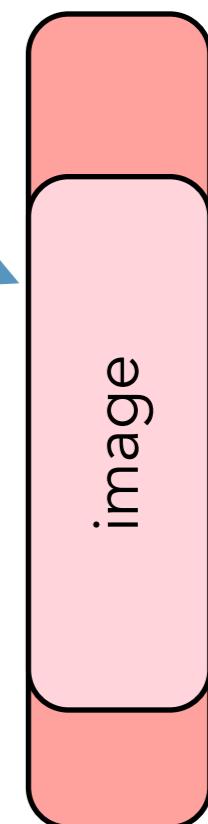
Domain (e.g. kmers)



Range (e.g.  $[0, m|D|]$ )

Minimal  
Perfect Hash

maps elements from the domain  
into the range

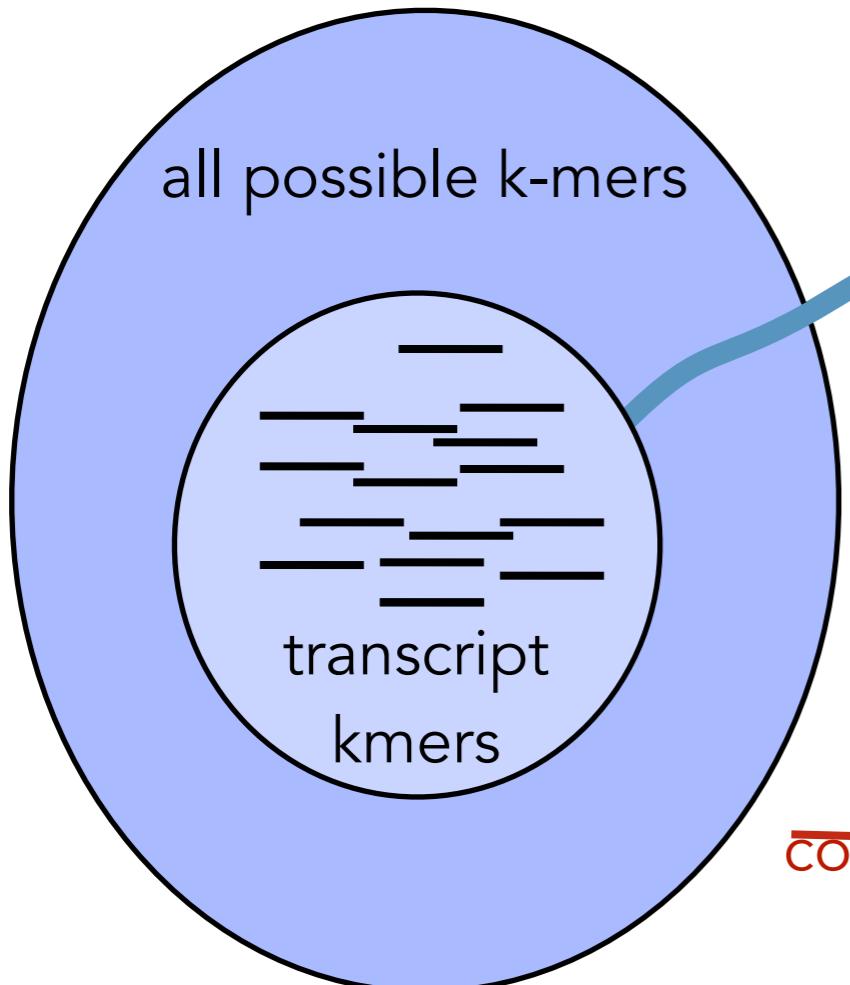


~~collisions - different keys with same value~~

maps keys to consecutive integers in  $[0, |D|-1]$

# Construction of a Perfect Hash Transcript Index

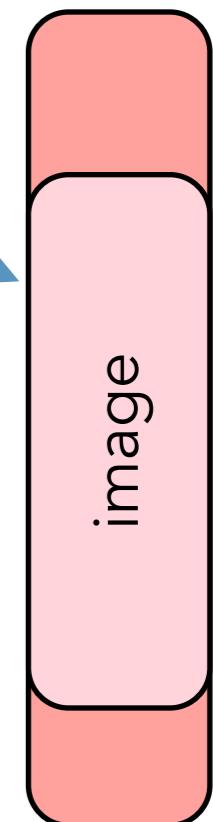
Domain (e.g. kmers)



Range (e.g.  $[0, m|D|]$ )

## Minimal Perfect Hash

maps elements from the domain  
into the range



~~collisions - different keys with same value~~

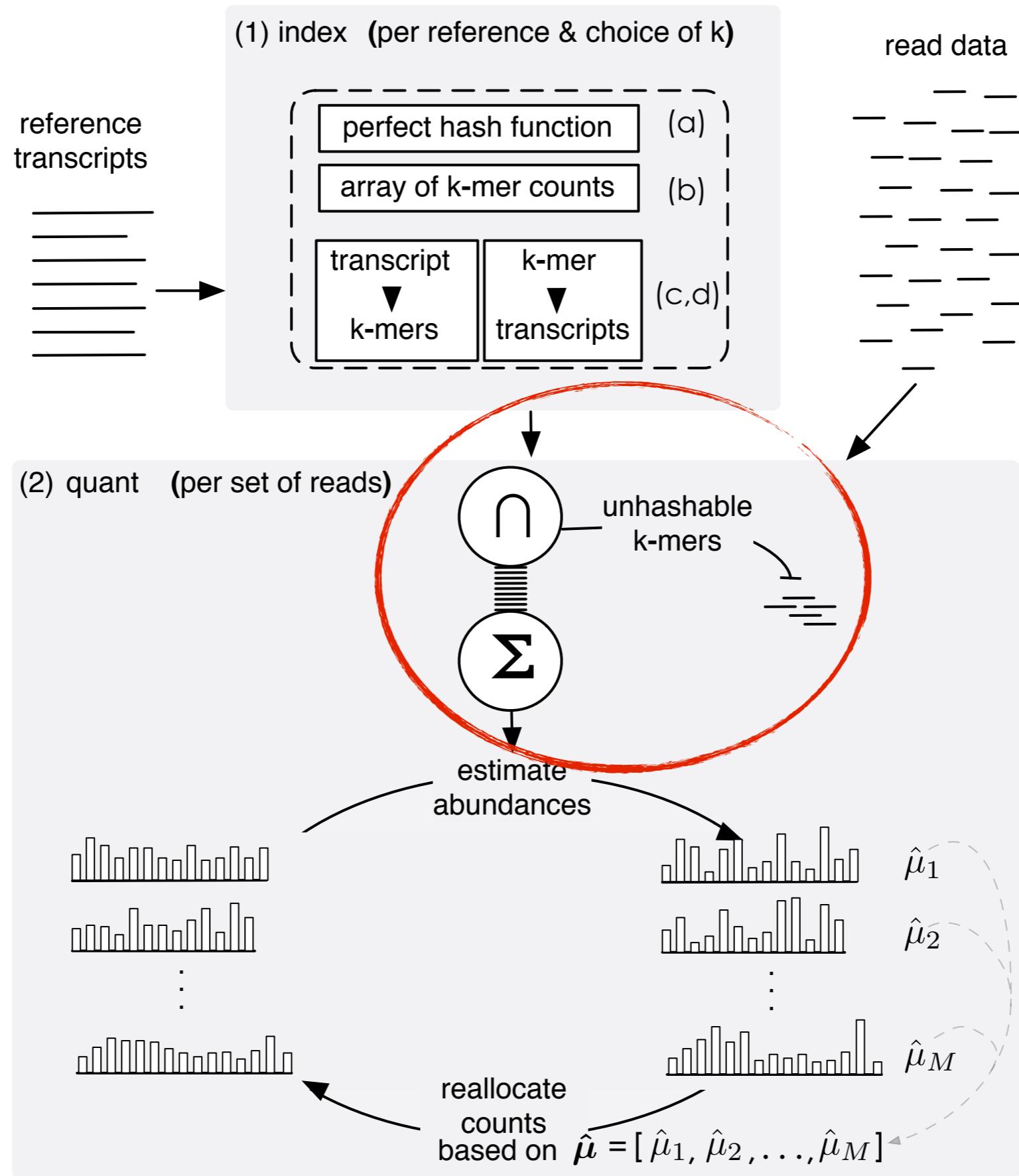
maps keys to consecutive integers in  $[0, |D|-1]$

BZD (Botelho et al.) minimal perfect hash algorithm to construct a compact function  $f(kmer)$  that maps each transcript kmer to an integer in  $[0, |D|-1]$ .

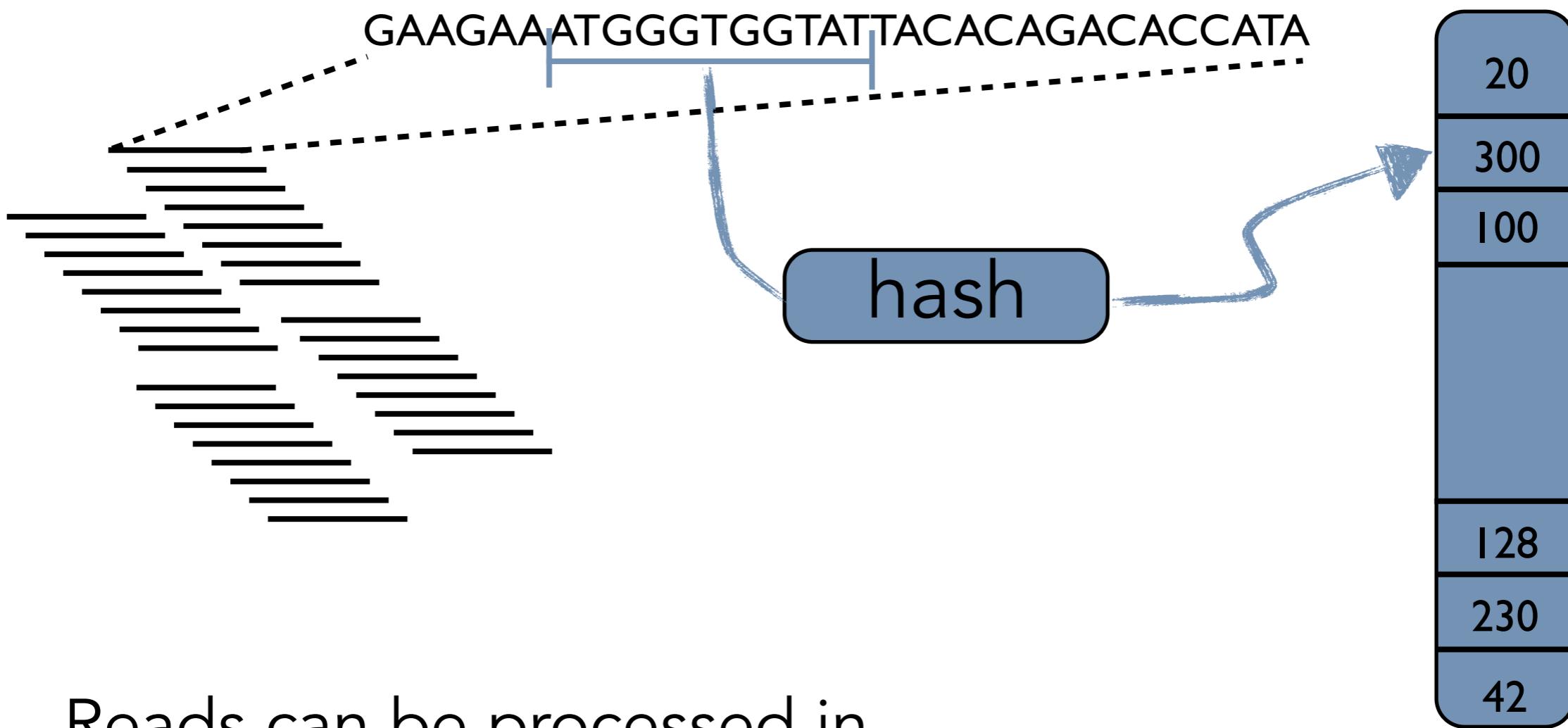
# Benefits of Minimal Perfect Hashing

Memory → Since we know all keys ahead of time, can construct a compact (low-overhead) hash

sorted array	Jellyfish hash (Marçais & K, 2011)	MPH
~0.6 µs/key	~0.35 µs/key	~0.1 µs/key



Parallelism →



Reads can be processed in parallel, use of CAS ensures efficient lock-free count updates

Array of atomics (CAS)

## K-mers are robust to errors

Transcript:

ATCAGACTTACACATGGAGGGACTAGCAGATG

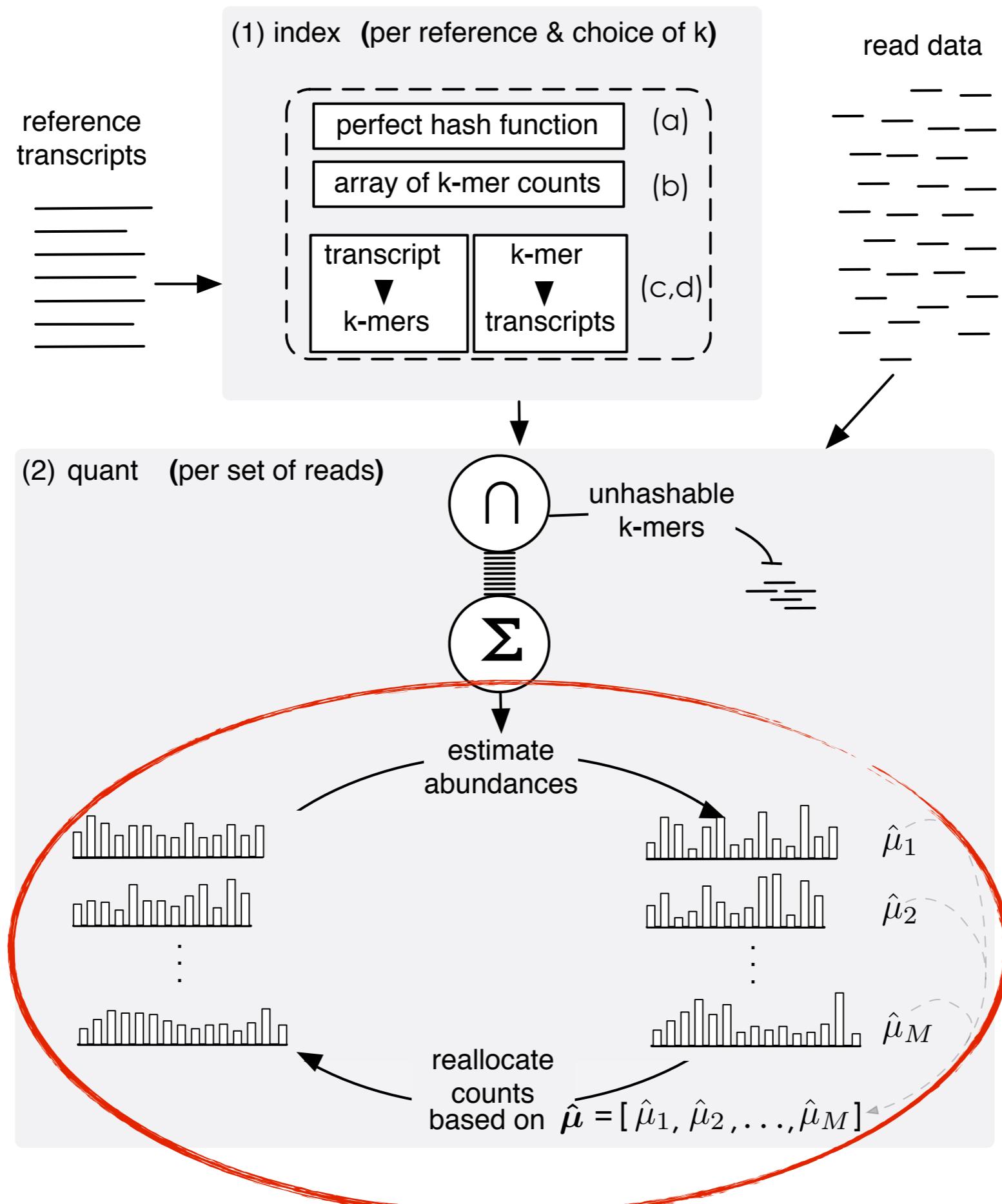
Read:

ACGCATGGAGGGACTAGCAA

.....

A read with errors still has many “good” k-mers

Only k-mers overlapping errors are discarded / mis-counted



# Kmer Allocation to Transcripts

kmers in reads and their counts

31

87

10

71

30

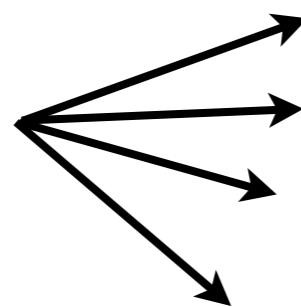
16

•

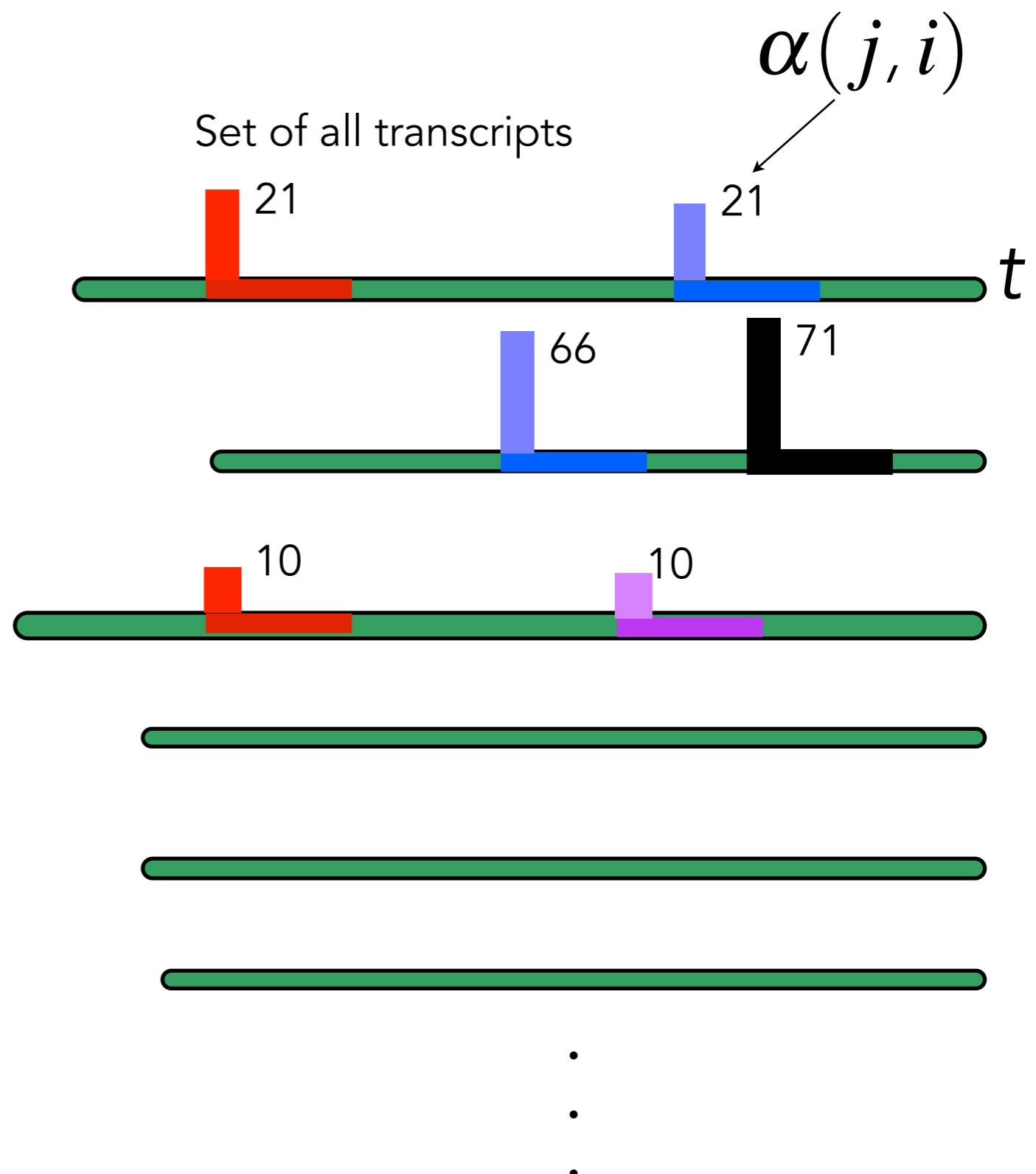
•

•

Goal: distribute kmer counts across transcripts so that each transcript is covered uniformly as possible (maybe at 0)

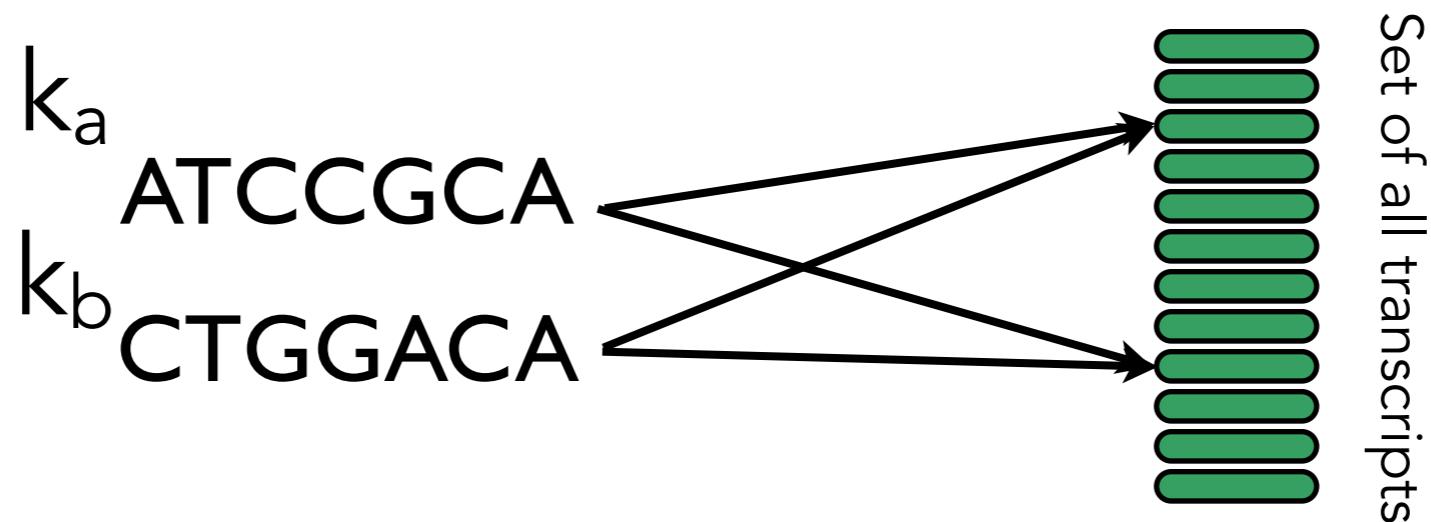


Set of all transcripts



## Elimination of Redundant Information

If kmers  $k_a$  and  $k_b$  always occur in the same transcripts at the same rate,  
then keeping track of them separately is redundant



Changes to the optimization:

$$[\mathbf{s}_i] = \mathbf{s}_j \quad \text{kmers}(\mathcal{T}) \quad \chi(\mathbf{s}_j) = \chi(\mathbf{s}_i)$$

Replace kmer & count with those of its equivalence class:

$$([\ ]) = \sum [\ ]_R (( ))$$

## Many Kmers are Redundant in this Way

### Example:

- In the protein-coding Human transcriptome of 104,770 transcripts & a set of 150M, 76bp paired-end reads...
- **72,627,992** unique 20-mers that appear can be collapsed into just **468,616** equivalence classes.

## Savings in Memory are Substantial

- Collapsing redundant kmers takes memory usage from ~60GB on the human transcript set to ~6GB.
- This enables the computation to be carried out on a modern laptop.

### Bonus:

- Since we're keeping track of fewer variables (kmers), the algorithm also becomes faster!
- Each iteration goes from  $\approx 15\text{s}$  to 1s.

# Expectation Maximization for Quantification

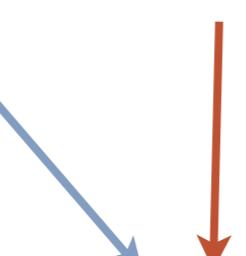
## E-Step

$$\alpha(j, i) = \frac{\hat{\mu}_i T([s_j])}{\sum_{t \supseteq [s_j]} \hat{\mu}_t}$$

# Expectation Maximization for Quantification

## E-Step

Allocation of **kmer** to **transcript**

$$\alpha(j, i) = \frac{\hat{\mu}_i T([s_j])}{\sum_{t \supseteq [s_j]} \hat{\mu}_t}$$


# Expectation Maximization for Quantification

## E-Step

Allocation of **kmer** to **transcript**    Normalized mean transcript coverage

$$\alpha(j, i) = \frac{\hat{\mu}_i T([s_j])}{\sum_{t \supseteq [s_j]} \hat{\mu}_t}$$

# Expectation Maximization for Quantification

## E-Step

Allocation of **kmer** to **transcript**      Normalized mean transcript coverage

$$\alpha(j, i) = \frac{\hat{\mu}_i T([s_j])}{\sum_{t \supseteq [s_j]} \hat{\mu}_t}$$

The diagram illustrates the components of the E-step formula. A blue arrow points to the index  $j$  in the term  $\alpha(j, i)$ . A red arrow points to the index  $i$  in the same term. A purple arrow points to the term  $T([s_j])$ , which is labeled "Normalized mean transcript coverage". A green arrow points to the denominator  $\hat{\mu}_t$ , which is labeled "Count of this k-mer class".

# Expectation Maximization for Quantification

## E-Step

$$\alpha(j, i) = \frac{\hat{\mu}_i T([s_j])}{\sum_{t \supseteq [s_j]} \hat{\mu}_t}$$

## M-Step

$$\mu_i = \frac{\sum_{[s_j] \subseteq t_i} \alpha(j, i)}{\hat{l}_i}.$$

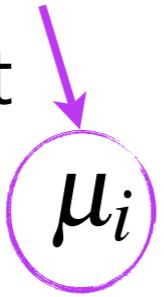
# Expectation Maximization for Quantification

## E-Step

$$\alpha(j, i) = \frac{\hat{\mu}_i T([s_j])}{\sum_{t \supseteq [s_j]} \hat{\mu}_t}$$

## M-Step

Estimated mean coverage

of transcript 

$$\hat{\mu}_i = \frac{\sum_{[s_j] \subseteq t_i} \alpha(j, i)}{\hat{l}_i}.$$

# Expectation Maximization for Quantification

## E-Step

$$\alpha(j, i) = \frac{\hat{\mu}_i T([s_j])}{\sum_{t \supseteq [s_j]} \hat{\mu}_t}$$

## M-Step

Estimated mean coverage  
of transcript

$$\hat{\mu}_i = \frac{\sum_{[s_j] \subseteq t_i} \alpha(j, i)}{\hat{l}_i}.$$

Allocation of k-mer /  
transcript

# Expectation Maximization for Quantification

## E-Step

$$\alpha(j, i) = \frac{\hat{\mu}_i T([s_j])}{\sum_{t \supseteq [s_j]} \hat{\mu}_t}$$

## M-Step

Estimated mean coverage  
of transcript

$$\hat{\mu}_i = \frac{\sum_{[s_j] \subseteq t_i} \alpha(j, i)}{\hat{l}_i}$$

Transcript length

Allocation of k-mer /  
transcript

# Expectation Maximization for Quantification

E-Step

$$\alpha(j, i) = \frac{\hat{\mu}_i T([s_j])}{\sum_{t \supseteq [s_j]} \hat{\mu}_t}$$

M-Step

$$\mu_i = \frac{\sum_{[s_j] \subseteq t_i} \alpha(j, i)}{\hat{l}_i}.$$

# Expectation Maximization for Quantification

E-Step

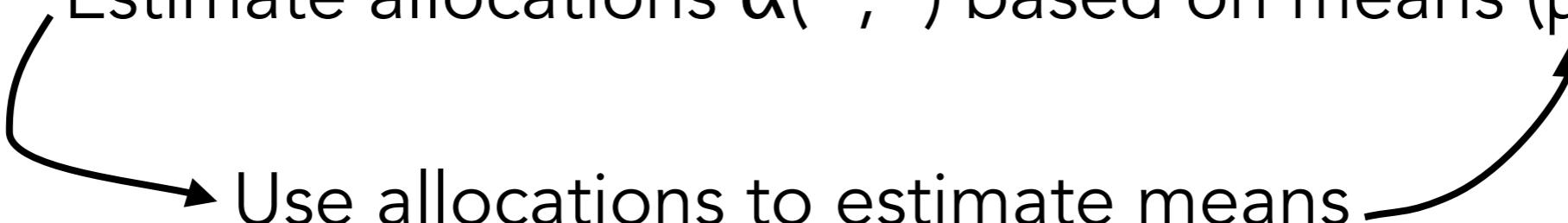
$$\alpha(j, i) = \frac{\hat{\mu}_i T([s_j])}{\sum_{t \supseteq [s_j]} \hat{\mu}_t}$$

M-Step

$$\mu_i = \frac{\sum_{[s_j] \subseteq t_i} \alpha(j, i)}{\hat{l}_i}.$$

**Iterative optimization (EM):**

Estimate allocations  $\alpha(\cdot, \cdot)$  based on means ( $\mu_t$ )



## Two Step EM for Improved Speed

- Pachter (2011) shows that under the assumption that reads are drawn from transcripts in proportion to its abundance, these EM-type procedures will converge to the true abundance.
- Actually use (Varadhan & Roland, 2008) two-step EM procedure to speed up convergence.
  - Idea: compute a couple of steps of EM to estimate a “gradient” between solutions and use that to take bigger steps when warranted.
  - Allows us to do the equivalent of a thousand EM steps in the time it takes for a few tens of EM steps.

Finally, we can estimate abundance

Different measures of abundance

Transcripts Per Million (TPM)

$$\text{TPM}_i = 10^6 \hat{\mu}_i$$

Reads Per Kilobase per Million mapped reads (RPKM)

$$\text{RPKM}_i = \frac{\frac{C_i}{l_i/10^3}}{\frac{N}{10^6}} = \frac{10^9 C_i}{N} \approx \frac{10^9 \mu_i}{N} \quad \text{where, } N = \sum_{[s_i]} T([s_i])$$

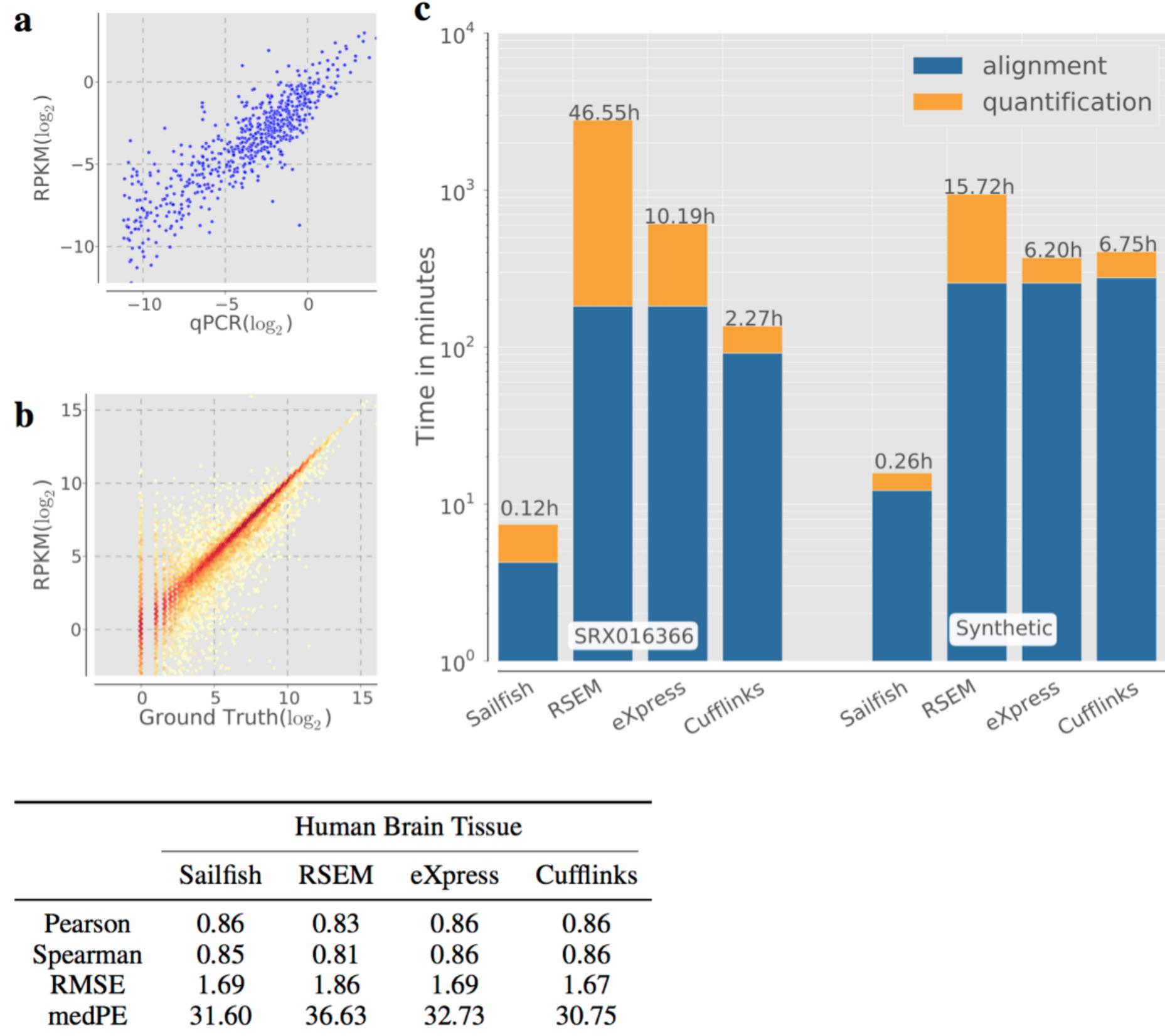
# Benefits of the Sailfish Approach

- Quantification without mapping
- Avoids error correction (b/c bad kmers tossed; frugal data usage)
- Massively parallel (exploits many-core machines and scales well by operating on small atomic units at a time.)
- Spends a bit of memory to gain time (uses  $\approx 8\text{gb}$  for entire human transcriptome; 256gb, 32-core machines now \$7.5k)

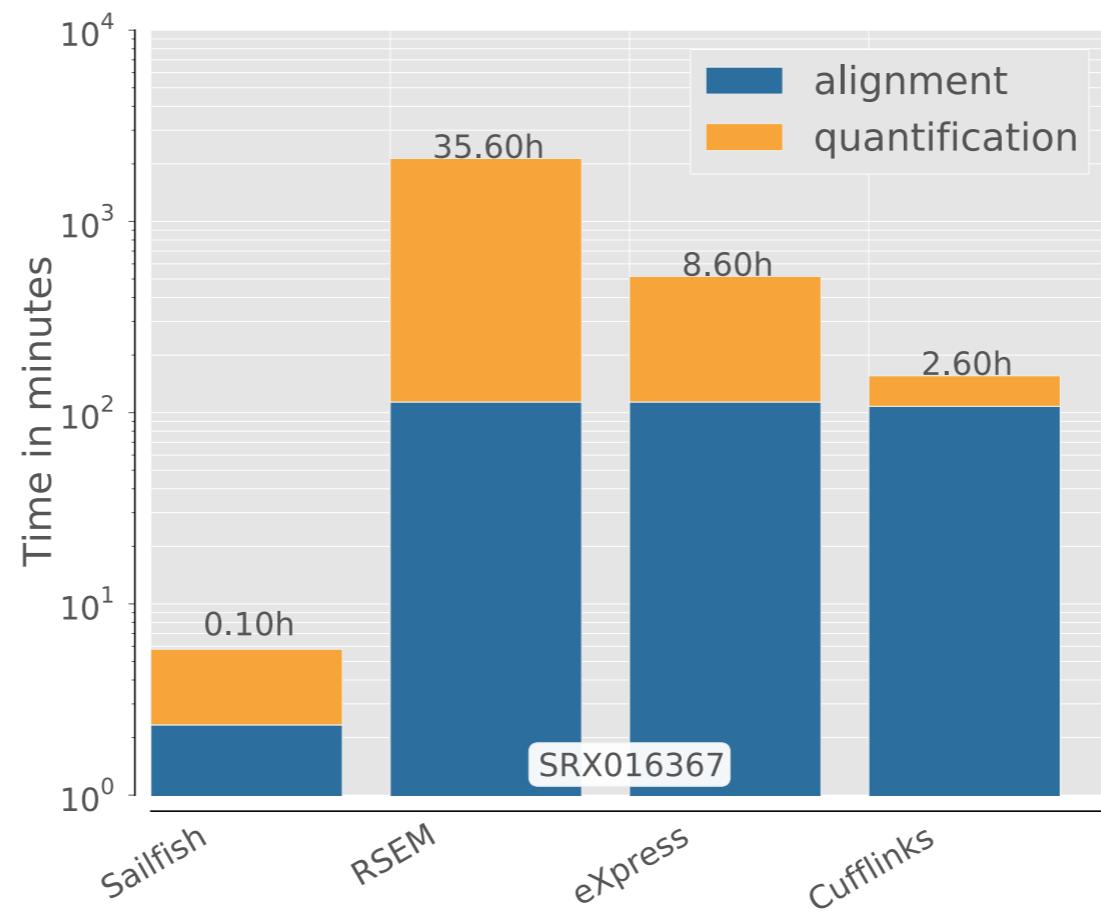
## “Lightweight Algorithms”

- a. “Simpler” algorithms (fast better than best)
- b. Frugal use of data (use only the units of data necessary)
- c. Use many cores
- d. Use “lots” of memory (trade memory for time; memory now cheap)

# Performance on Human Brain Tissue



# Performance on Universal Human Reference Tissue



---

	Sailfish	RSEM	eXpress	Cufflinks
Pearson	0.87	0.85	0.87	0.87
Spearman	0.88	0.85	0.88	0.88
RMSE	1.64	1.81	1.65	1.68
medPE	29.95	34.77	31.03	27.33

---

93M reads, each 35bp long

## Simulated Data?

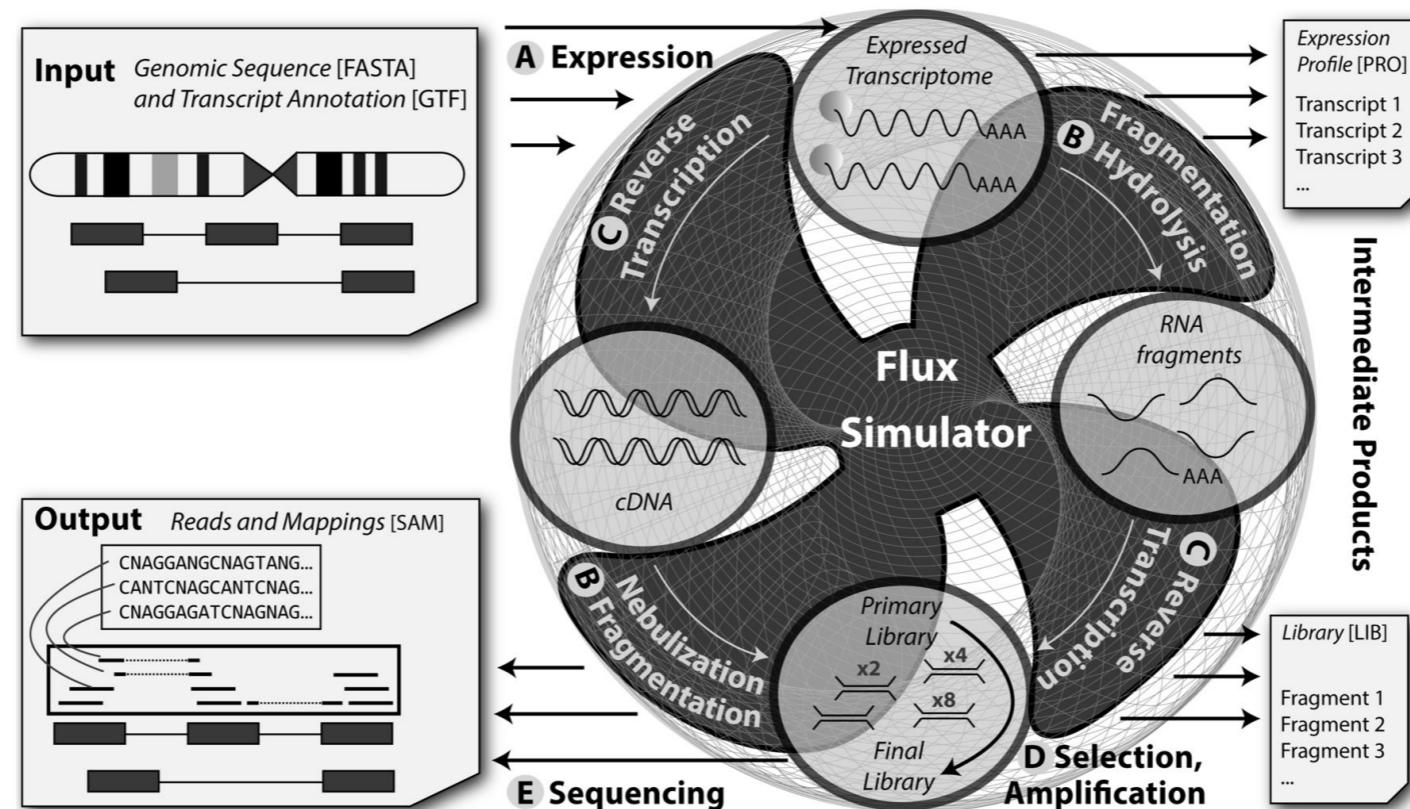
- qRT-PCR data is great, but it may be too easy:
  - ▶ Relatively few genes to compare against
  - ▶ Transcript quantification aggregated to the gene level
- Most methods have been validated on synthetic data as well:
  - ▶ From simulation, get a ground truth abundance
  - ▶ Can compare transcript-level quantification
  - ▶ Can compare effects of different experimental variables (e.g. read length, # of reads, paired-end) on quantification
- Many previous approaches (RSEM, eXpress) roll their own read simulators.
  - ▶ They assume their generative graphical model when producing sequences – is this begging the question?

# Flux Simulator (Griebel et al. NAR, 2012)

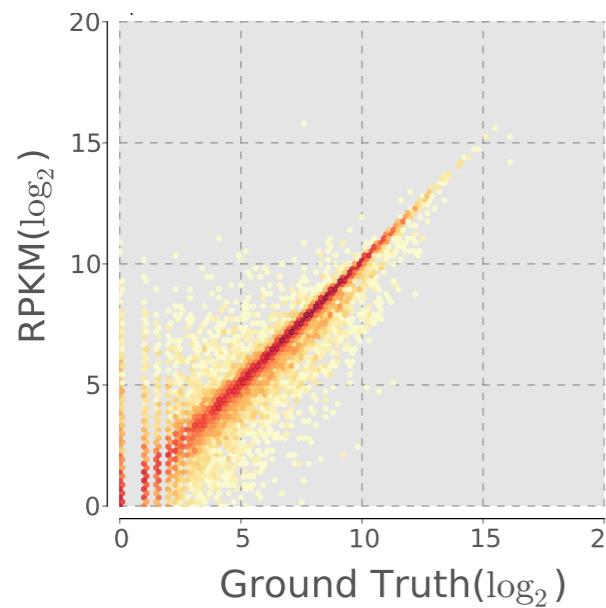
Not based on the specific generative model of any RNA-seq estimation method

In-depth comparison against multiple real datasets

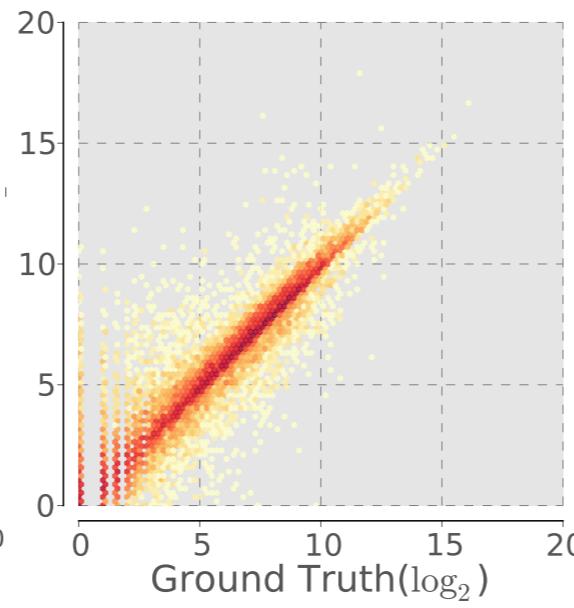
Can control many various experimental variables: fragmentation, selection/amplification, sequencing



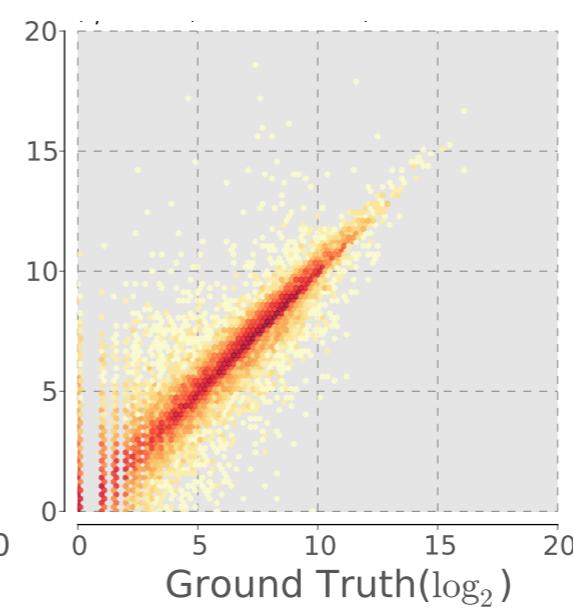
Sailfish



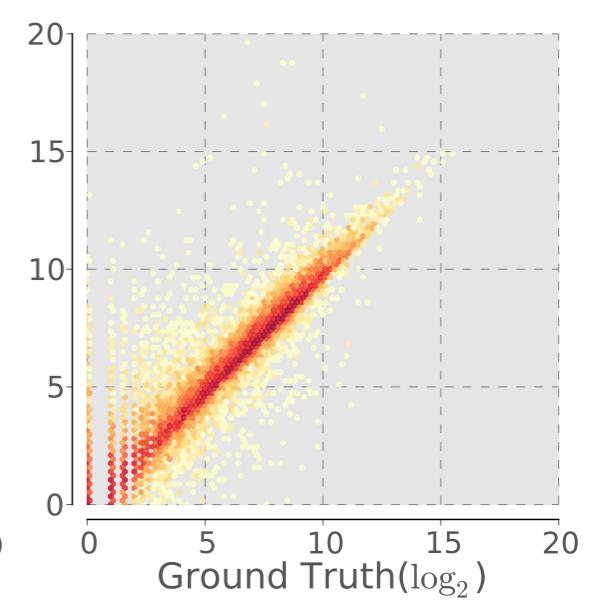
RSEM



eXpress



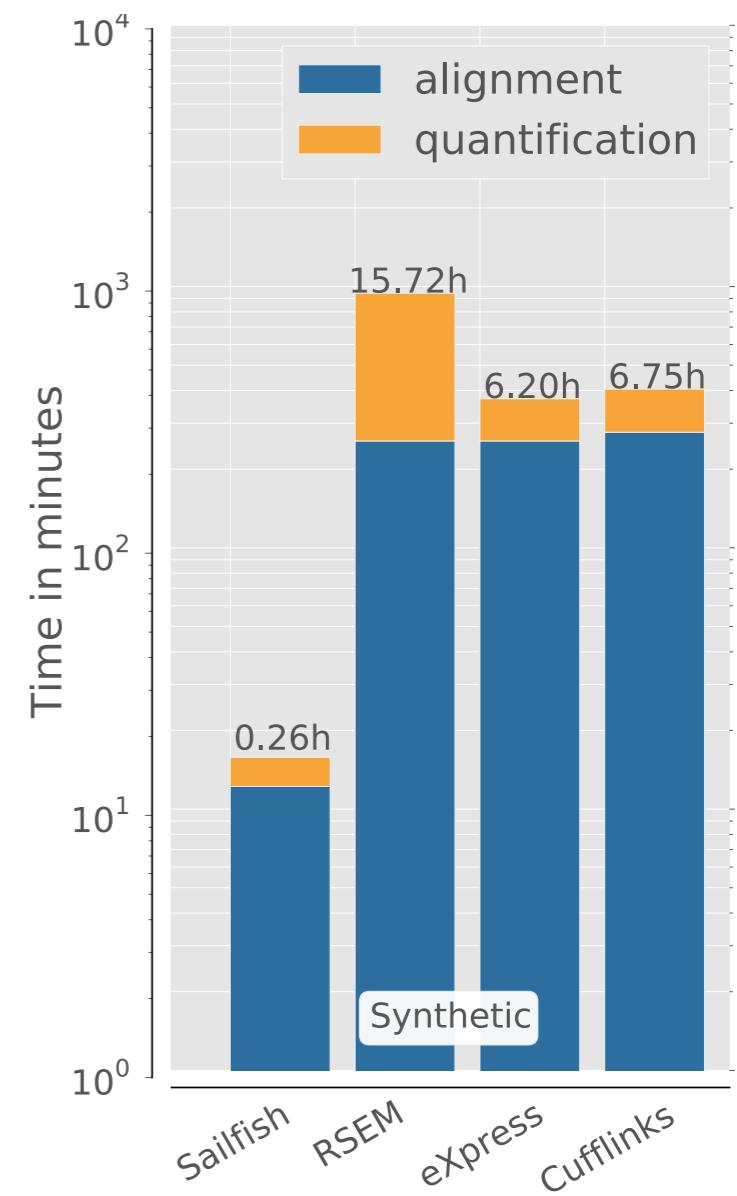
Cufflinks



Pearson  
Spearman

	Sailfish	RSEM	eXpress	Cufflinks
Pearson	0.92	0.92	0.90	0.91
Spearman	0.94	0.93	0.92	0.93

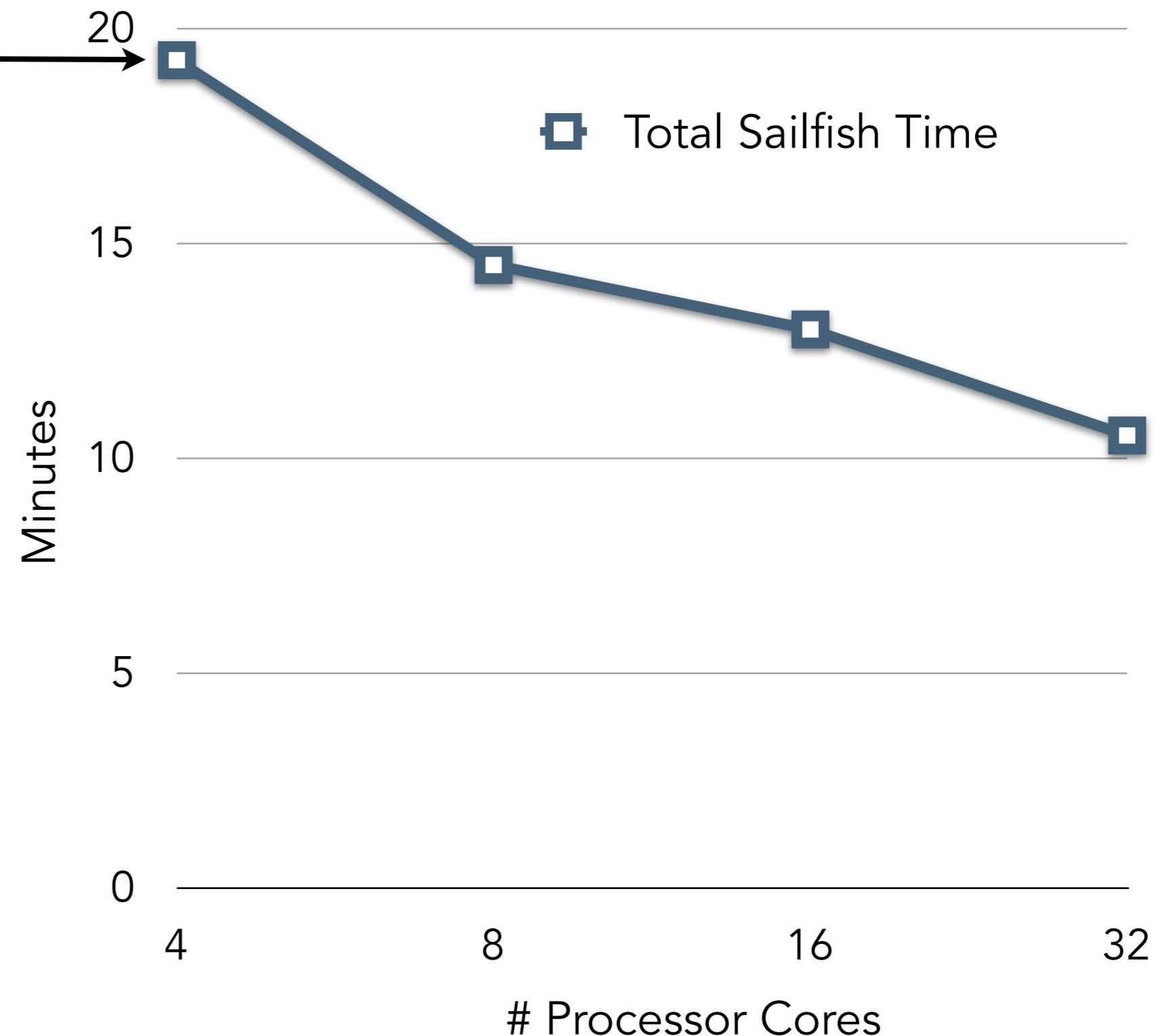
Simulated Data  
75M 76bp x2 paired-end reads



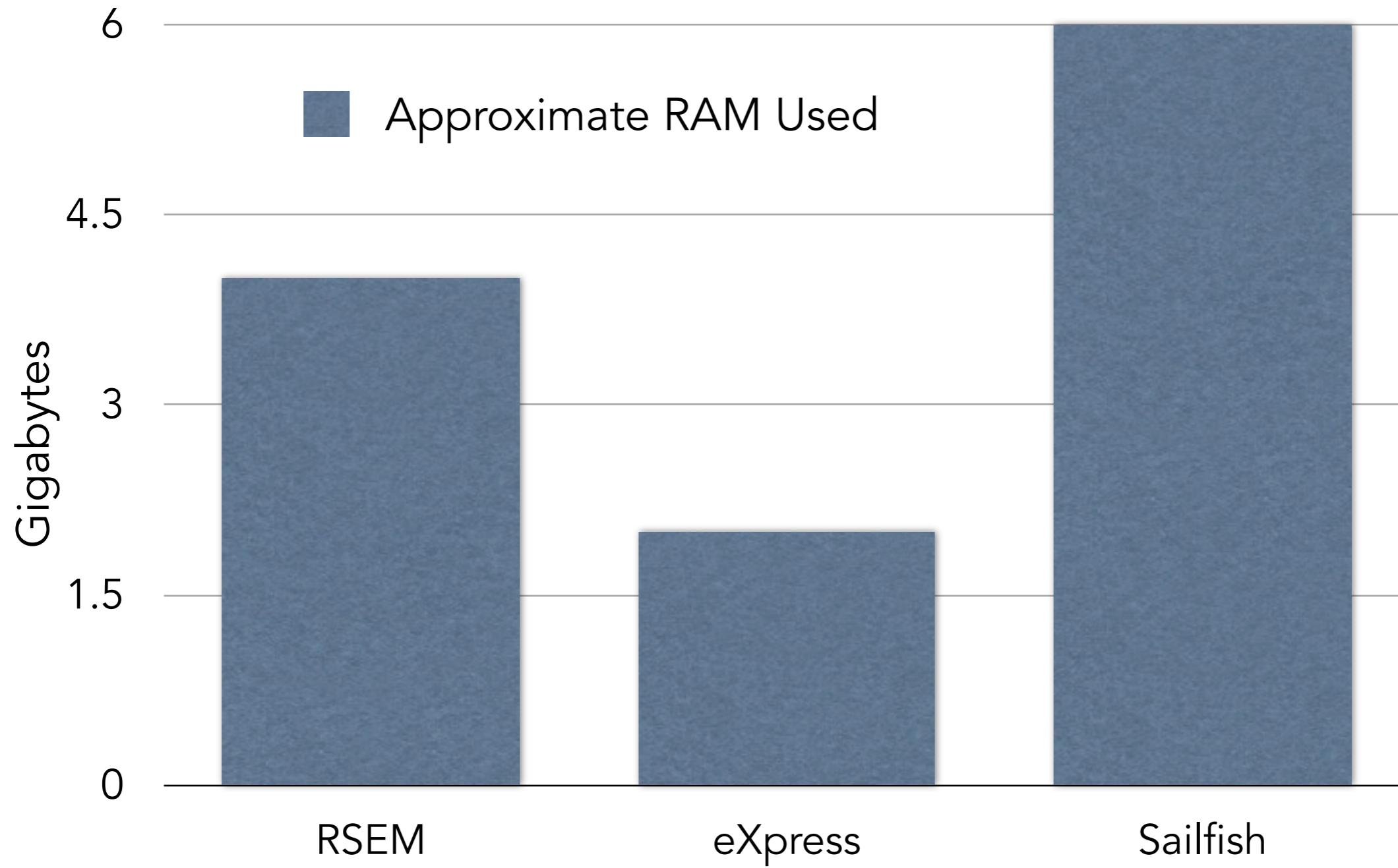
# Use of Multicore Architecture

Even using only 4 cores, Sailfish is about 25 times faster than RSEM using 32 cores.

Even using only 4 cores, and **not counting mapping time**, Sailfish is 7.7 times faster than RSEM



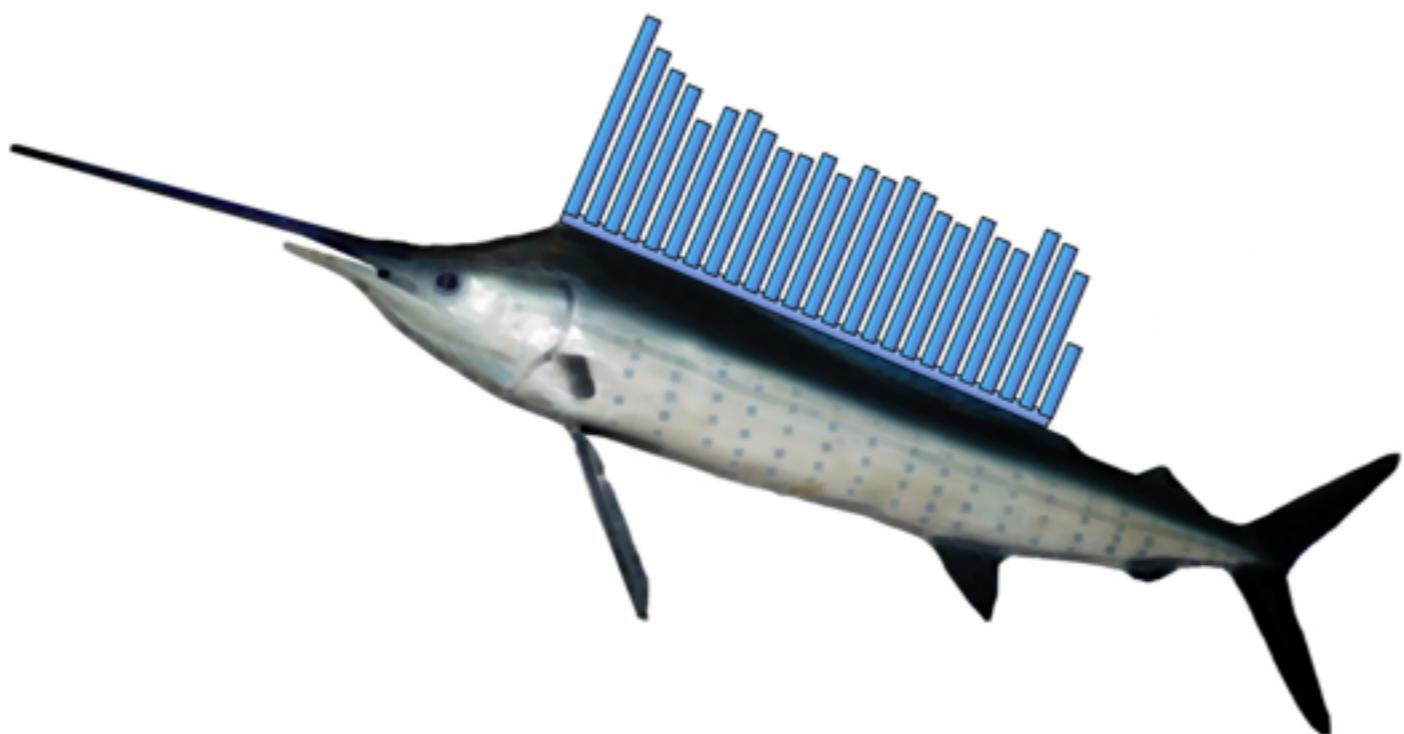
# Memory Usage



92,542,365 reads; universal human reference tissue.

# Sailfish

- Sailfish is a far faster approach for quantifying the abundance of known transcripts.
- A good example of a “lightweight” algorithm whose design is matched to both modern hardware configurations (multicore, large memory) and modern data sizes (big).
- Sailfish is open source, available at <http://www.cs.cmu.edu/~ckingsf/software/sailfish> (It's written in C++11.)



[arXiv:1308.3700](https://arxiv.org/abs/1308.3700)