Wide-SIMD Parallelization of Streaming Dataflow, with Applications to Bioinformatics

Jeremy Buhler

For CSE 591

NSF Awards CNS-1500173, CNS-1763503

Take-Home Message

- **Biological sequence analysis** is a source of high-impact computational problems
- Using SIMD parallel computing for these problems requires dealing with **irregularity**
- **MERCATOR** is an ongoing research effort to make irregular application development on SIMD platforms easier.

Who Am I?

- I study how to accelerate high-impact bioinformatics problems.
- One way to do this is via parallelization on modern architectures (FPGAs, GPUs, …)

- Along the way, many interesting CS questions…
  - Streaming computation [FCCM'07, JVSP'07,M&M'09]
  - Systolic array design [FPL'09,FCCM'10,ASAP'10]
  - Deadlock avoidance [SPAA'10,PPoPP'12,DFM'13,JPDC'17]
  - SIMD mapping [ISPDC'14,DFM'15,HPCS'17]

Talk Overview

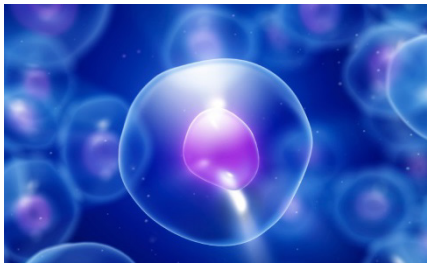
- Problems: DNA comparison and read mapping
- Algorithmic approach – Why SIMD?
- MERCATOR overview and performance
- Research challenges

Molecular Biology is Fundamental

- **Genetic basis** of disease and disease risk
- **Systems biology** – what are your cells doing?
- Studying **natural history** and evolution
- **Engineering** cells' behavior for medicine, industry, agriculture

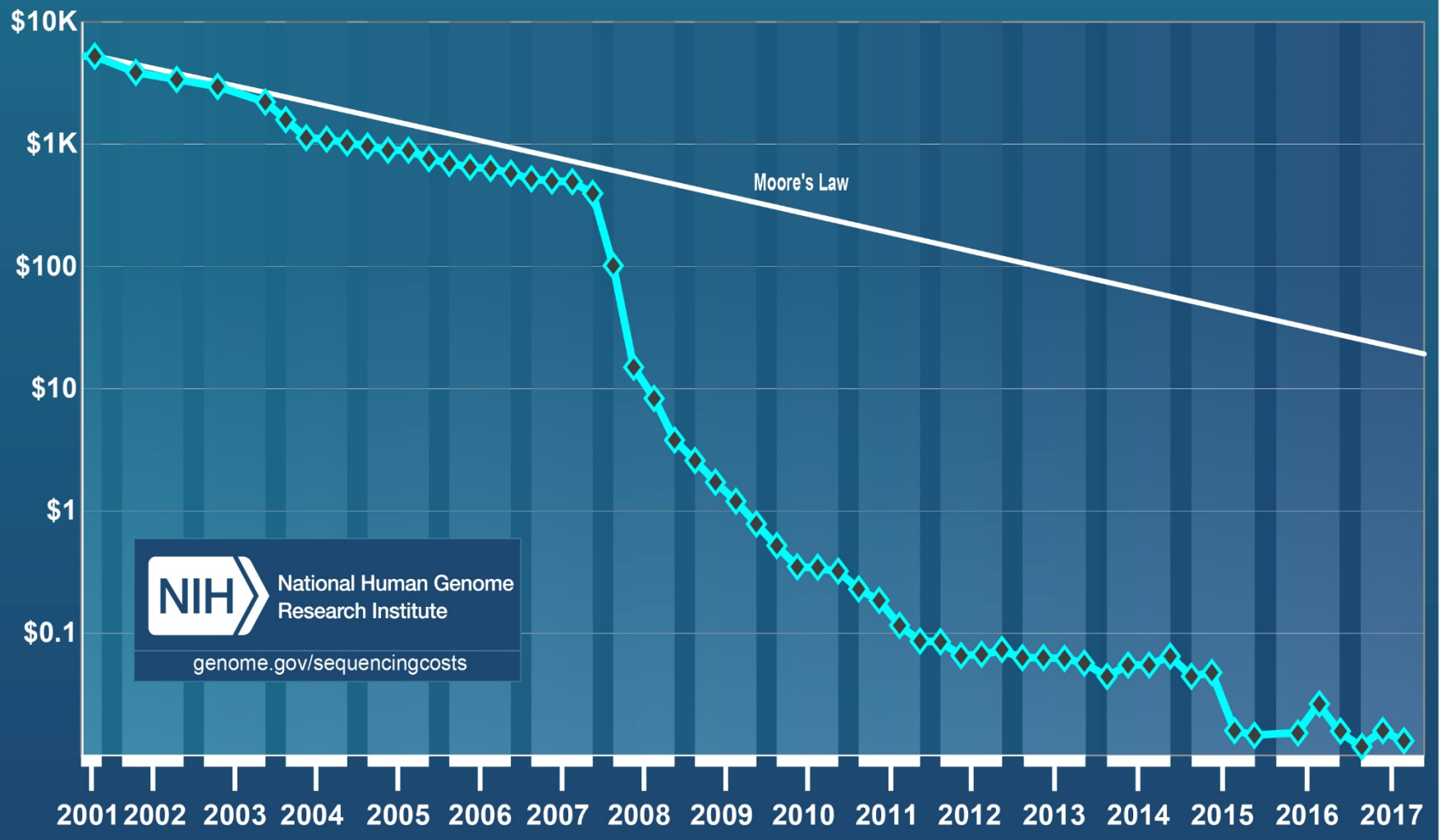
The First Step: DNA Sequencing

- Sequencing can tell us what is in a genome…
- … but also the basis of experiments to probe gene expression, protein binding, chromosome conformation, epigenetic marks, polymorphism, copy number variants …



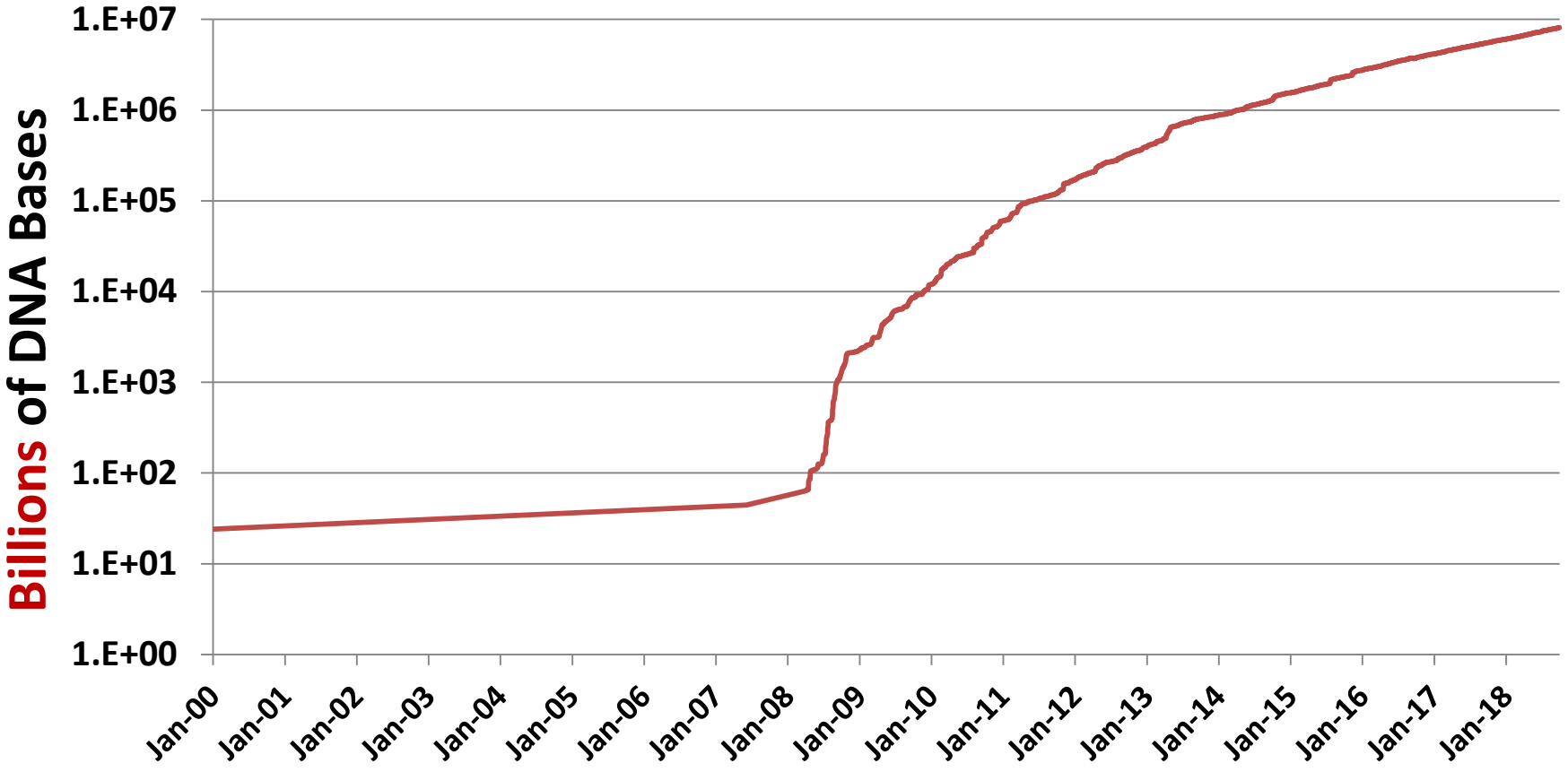
…acaggatagtaccgataccat
cacccggataggacctatgag
ggacacaggacttatggcattt…

Cost per Raw Megabase of DNA Sequence



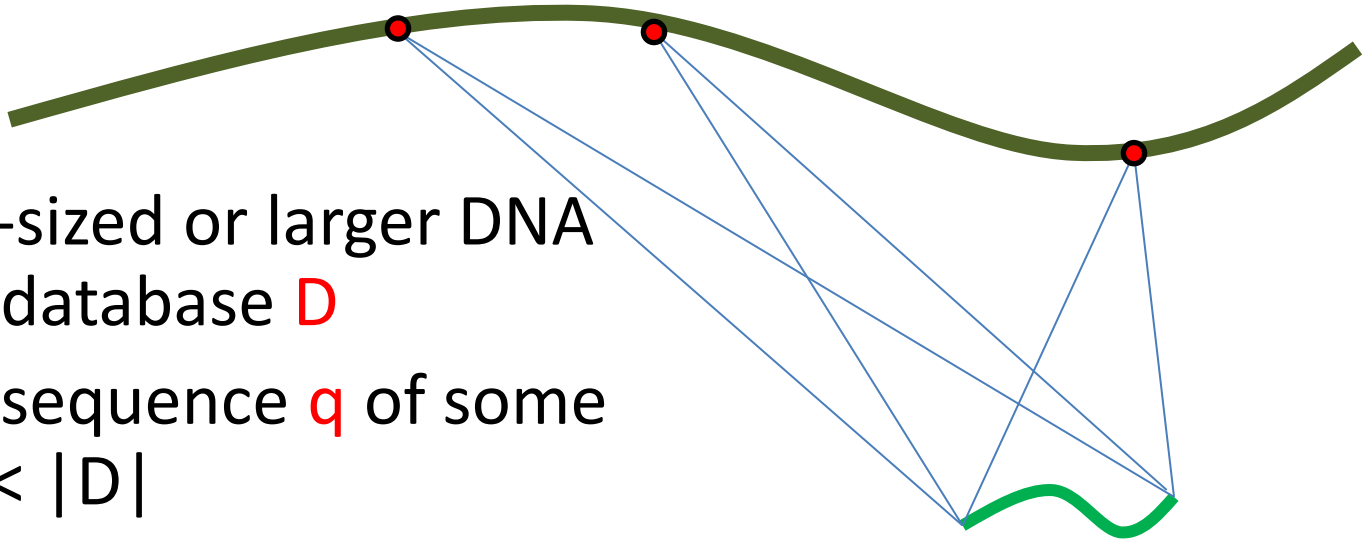
NIH Sequence Read Archive Size

(Open Access Only)



http://www.ncbi.nlm.nih.gov/Traces/sra/

Problem: Classical Similarity Search



- Given
  - a genome-sized or larger DNA sequence database D
  - a "query" sequence q of some length L << |D|

- *Does q appear in D with at most k differences*, and if so, where?

Typical Parameters

- Database D has size $10^9 – 10^{10}$ bases
- Query q has size $10^2 – 10^4$ bases
- # differences k is 5-25% of |q| (bases added, deleted, changed)

Tools for Similarity Search

- **BLAST** [Altschul et al. 1990, 1996]

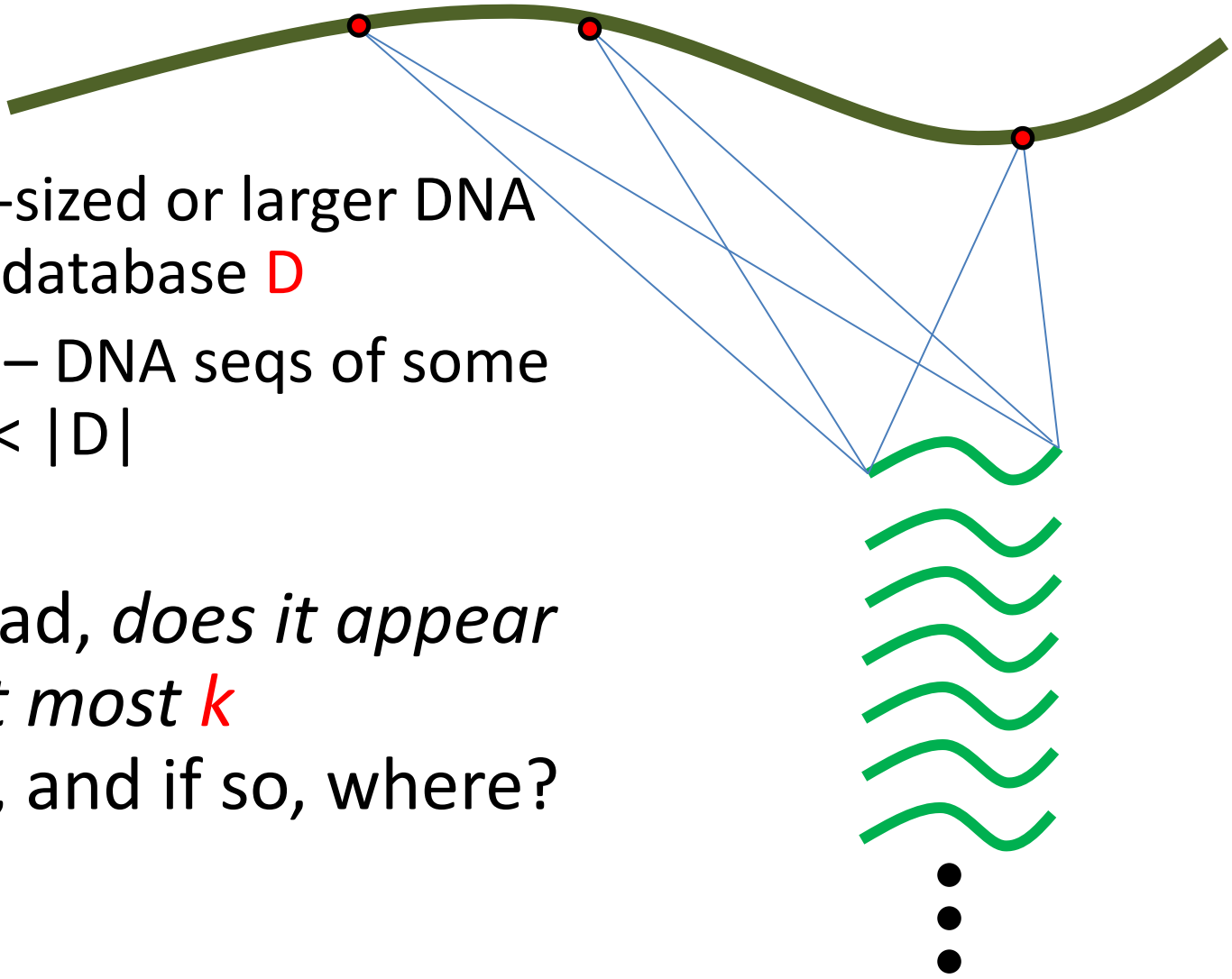
- **BLAT** [Kent 2002]

These tools use variants of same basic search algorithm.

Problem: Short-Read Mapping

- Given
  - a genome-sized or larger DNA sequence database D
  - N "reads" – DNA seqs of some length L << |D|

- For each read, *does it appear in D with at most k differences*, and if so, where?



Typical Parameters

- Database D has size $10^9 – 10^{10}$ bases
- Number of reads N is $10^6 – 10^8$
- Length L is 75-150 (may vary among reads)
- # differences k is 0-3 (added, deleted, changed)

Tools for Mapping

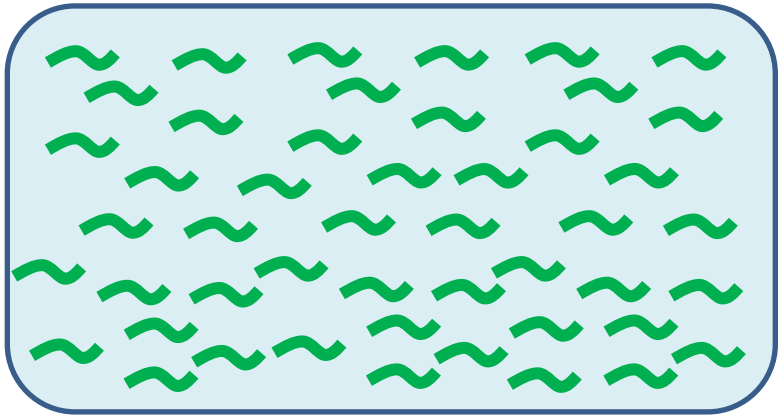
- **Bowtie** [Langmead et al. 2009, 2012]
- **BWA** [Li & Durbin 2009, 2010]
- **SOAP2** [Li et al. 2009]

All these tools use variants of same basic search algorithm.

Why Short-Read Mapping?

- Some experimental procedure selects a subset of everything in the database
- Reads are sampled from this subset by your sequencing machine
- Mapping tells you which parts of database are present in your sample

Problem: Alignment-Free Organism ID



- Given
  - a metagenome-sized DNA sequence database D
  - N microbial genomes – DNA seqs of some length L << |D|

- For each genome, *do (some of) its sequences appear in D?*

Alignment-Free Techniques

- **Min-Hash Sketching** – convert a seq to a small sample (m ~ 1000) of hash values

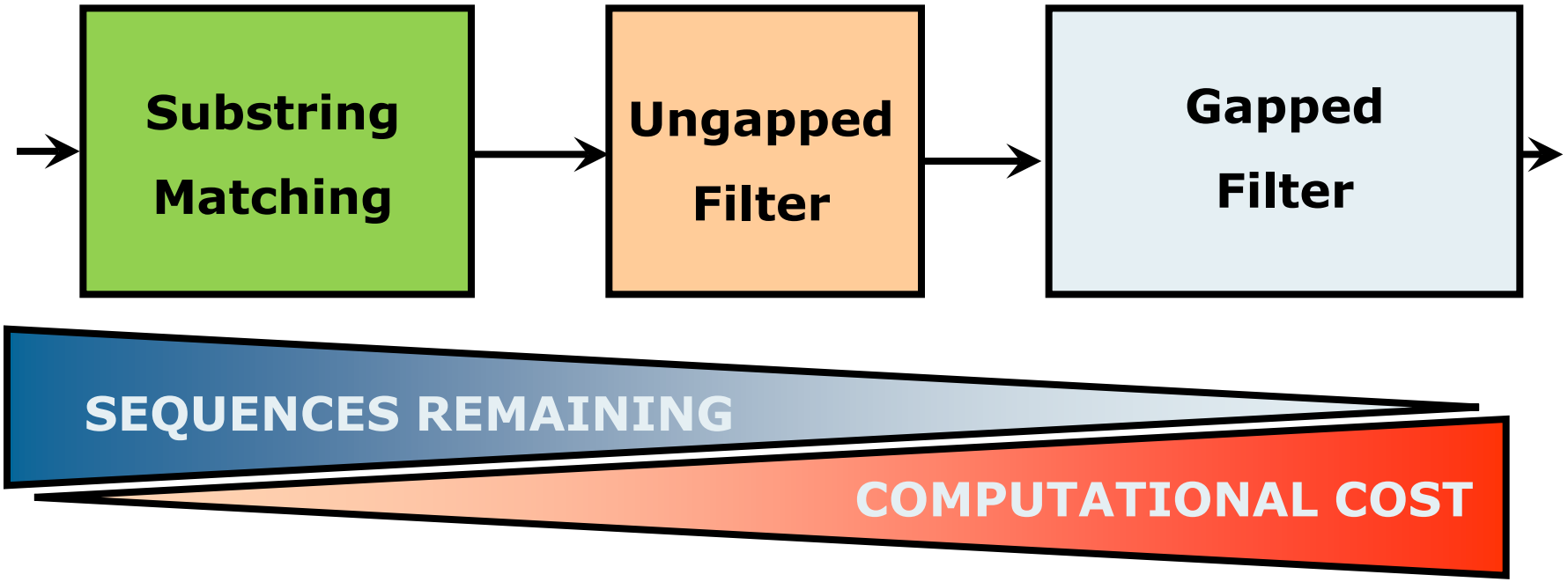
- **Approximate Containment**: how much of (the sketch for) a genome overlaps (the sketch for) a metagenome?

- MASH (Ondov et al. 2016)
- SourMASH (Brown et al. 2016)

Talk Overview

- Problems: DNA alignment and read mapping
- Algorithmic approach – Why SIMD?
- MERCATOR overview and performance
- Research challenges

How BLAST Works



BLAST operates as a pipeline of computational stages.

Stages of BLAST

- Stage 1: identify *potential match locations* between q, D
- Stage 2: keep only those locations that look somewhat promising
- Stage 3: keep only those locations that actually yield high-similarity alignments

Generating Possible Matches

- Every place where some 11-mer from q matches an 11-mer from D *exactly* is a candidate.

  accagatacatagcactcgctacgtcagatgggtaca

  gttaagtcagatgggtagactcaggatgacagtggaca

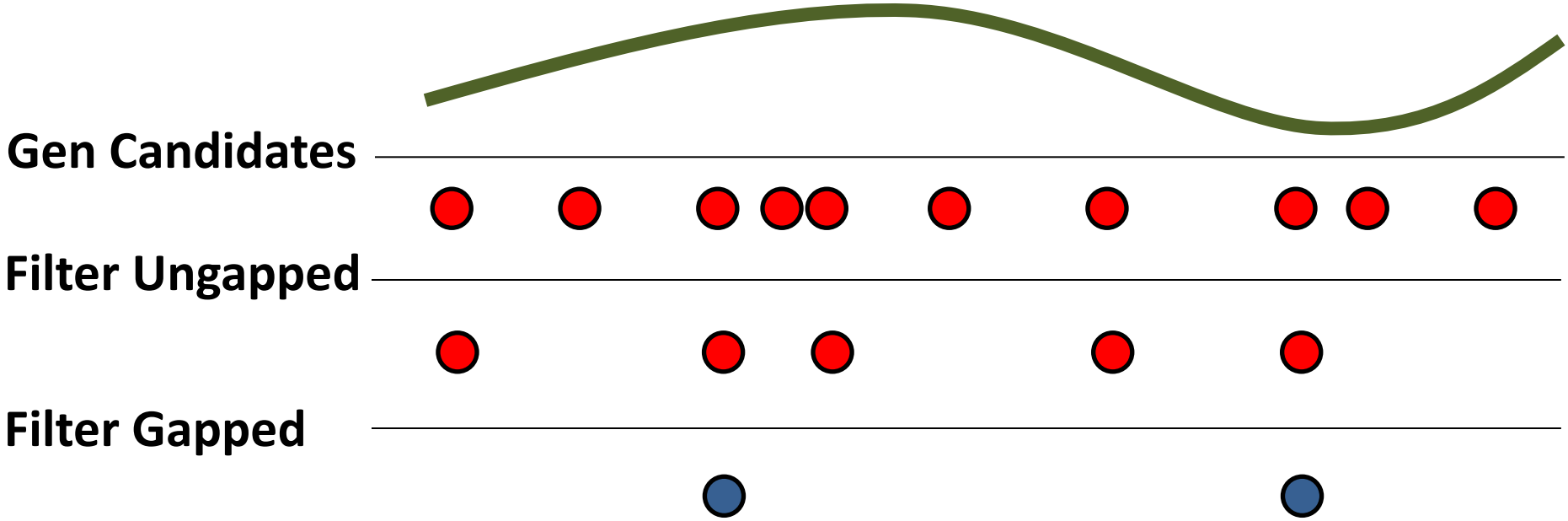
- Can rapidly find all such matching locations using hash table of 11-mers in sequence q

Filtering Candidates

- Uses explicit edit distance computation between q, part of D (Smith-Waterman algo)
- Expensive dynamic programming!
- “Easy” version (substitutions only), followed by hard version (add/delete chars allowed)

BLAST Parallelization

- Can generate candidates in parallel at each DB location, then filter them in parallel.



What About Read Mapping?

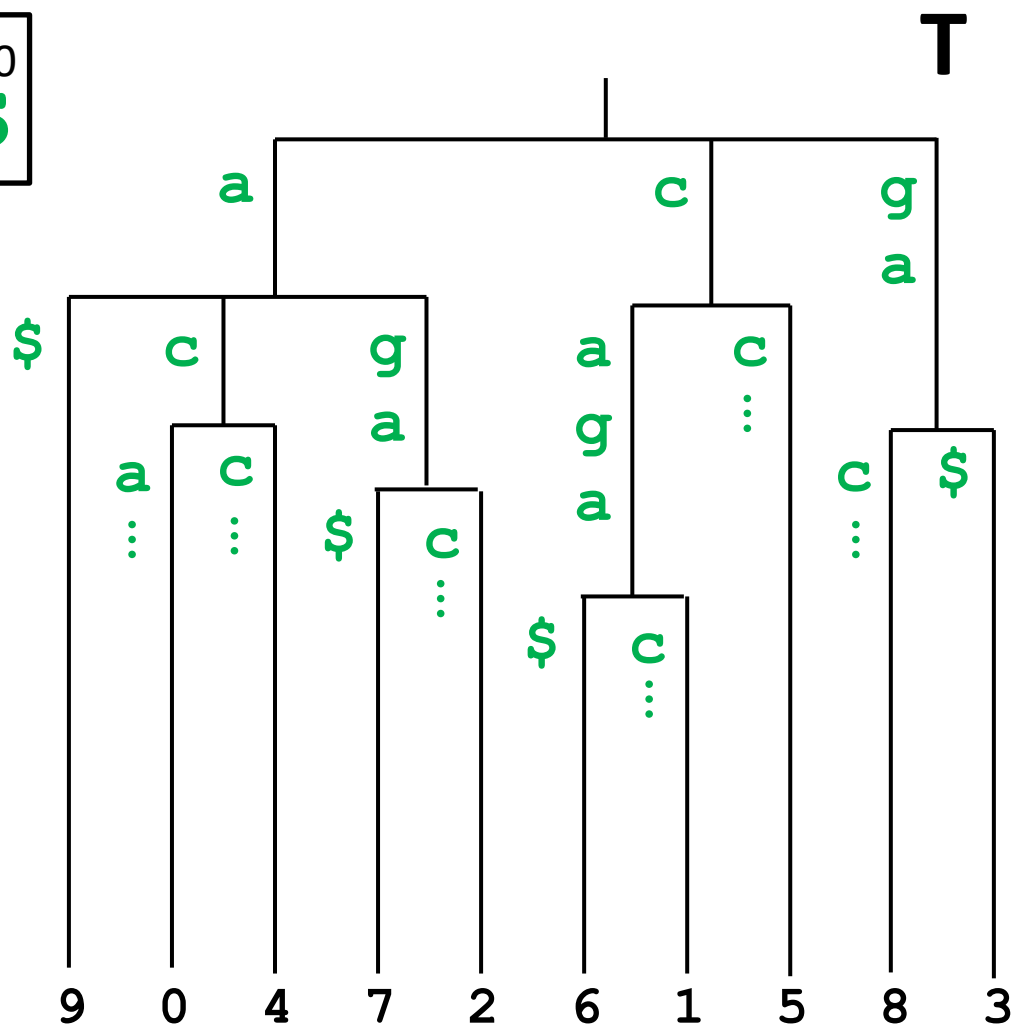
- Uses an index (virtual suffix tree) of database
- Matching involves tracing a path down index tree for each read
- (must try several paths if differences are allowed)
- *Can do in parallel for many reads at once!*

Suffix Tree Example

D = 0 1 2 3 4 5 6 7 8 9 10
acagaccaga$

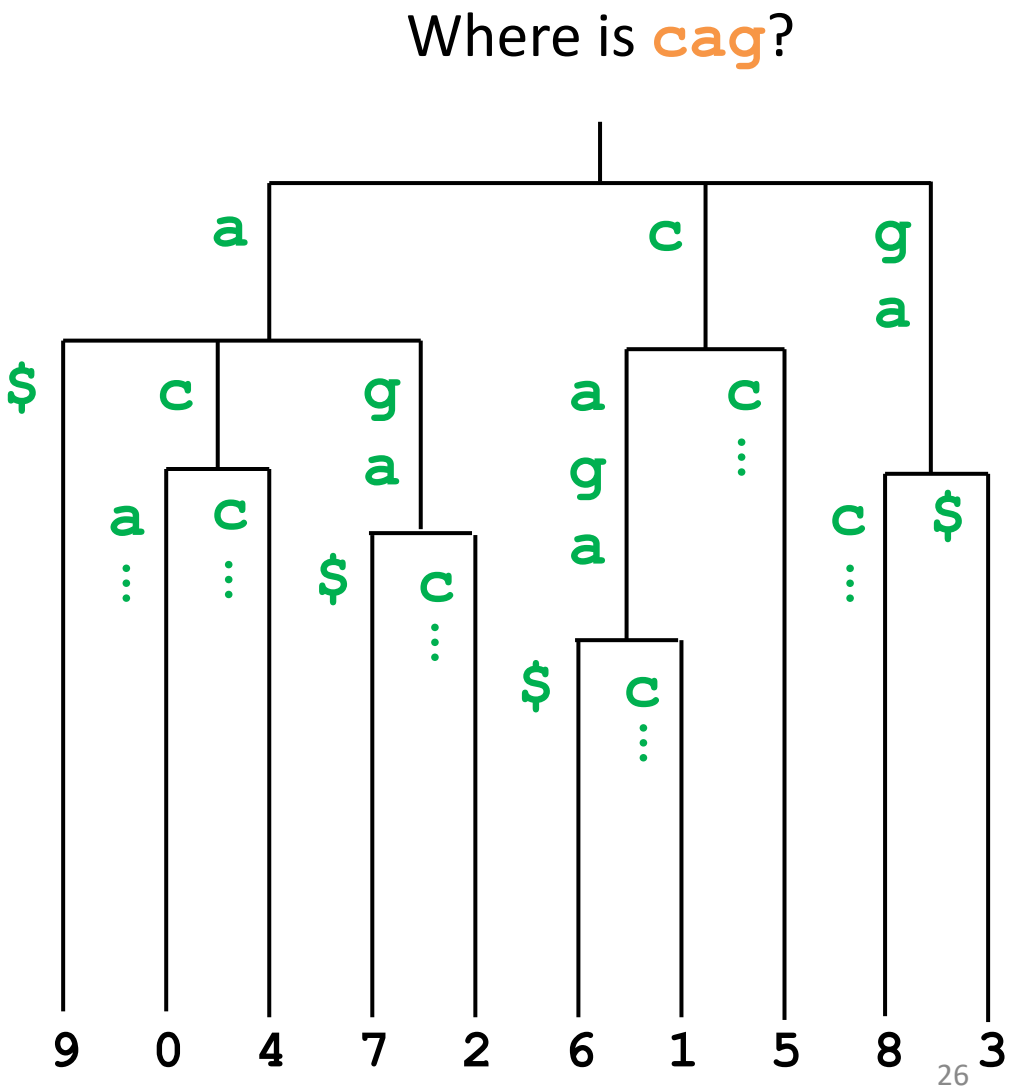
A

10	$
9	a$
0	aca…
4	acc…
7	aga$
2	agac…
6	caga$
1	cagac…
5	cca…
8	ga$
3	gac…



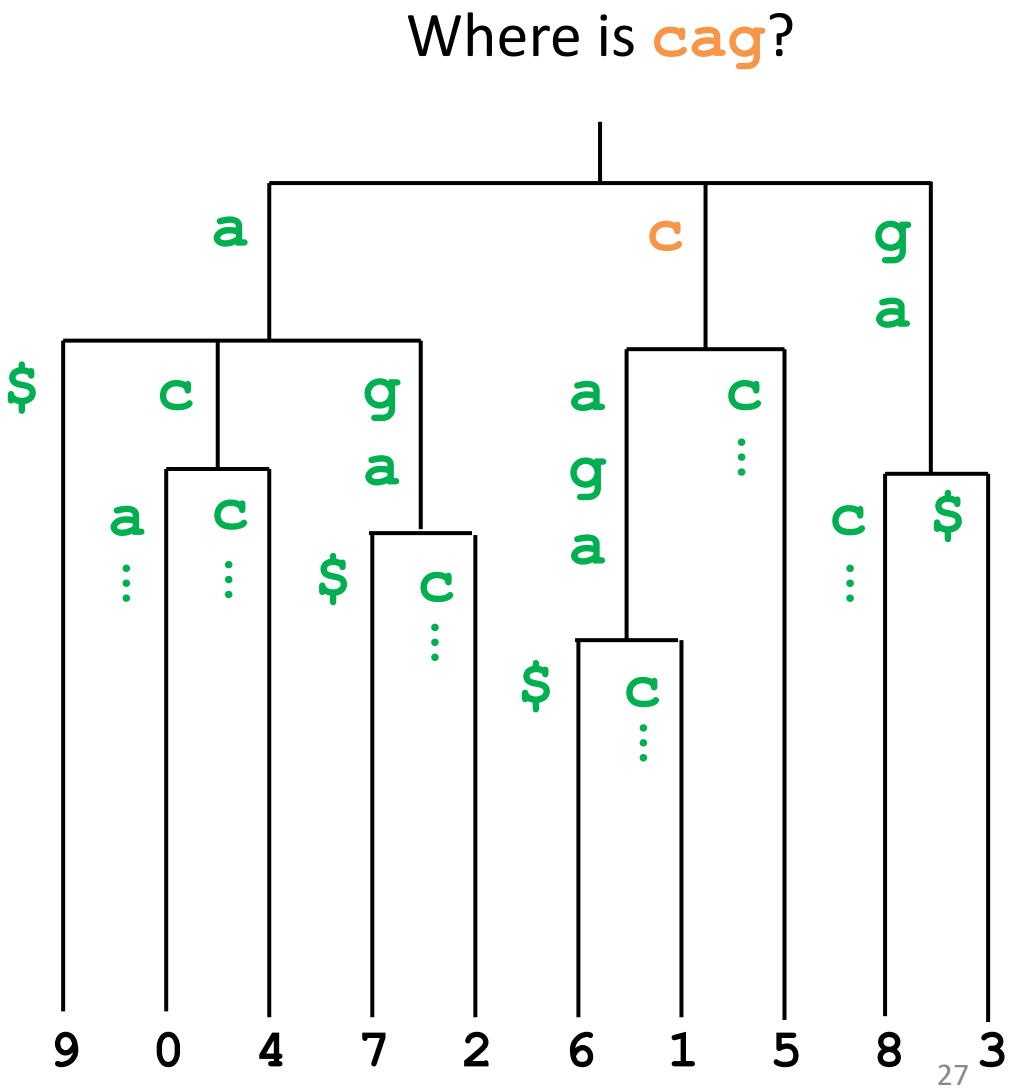
Rapid Matching vs Suffix Tree

- Can find all matches to a read in D in time proportional to read length L.



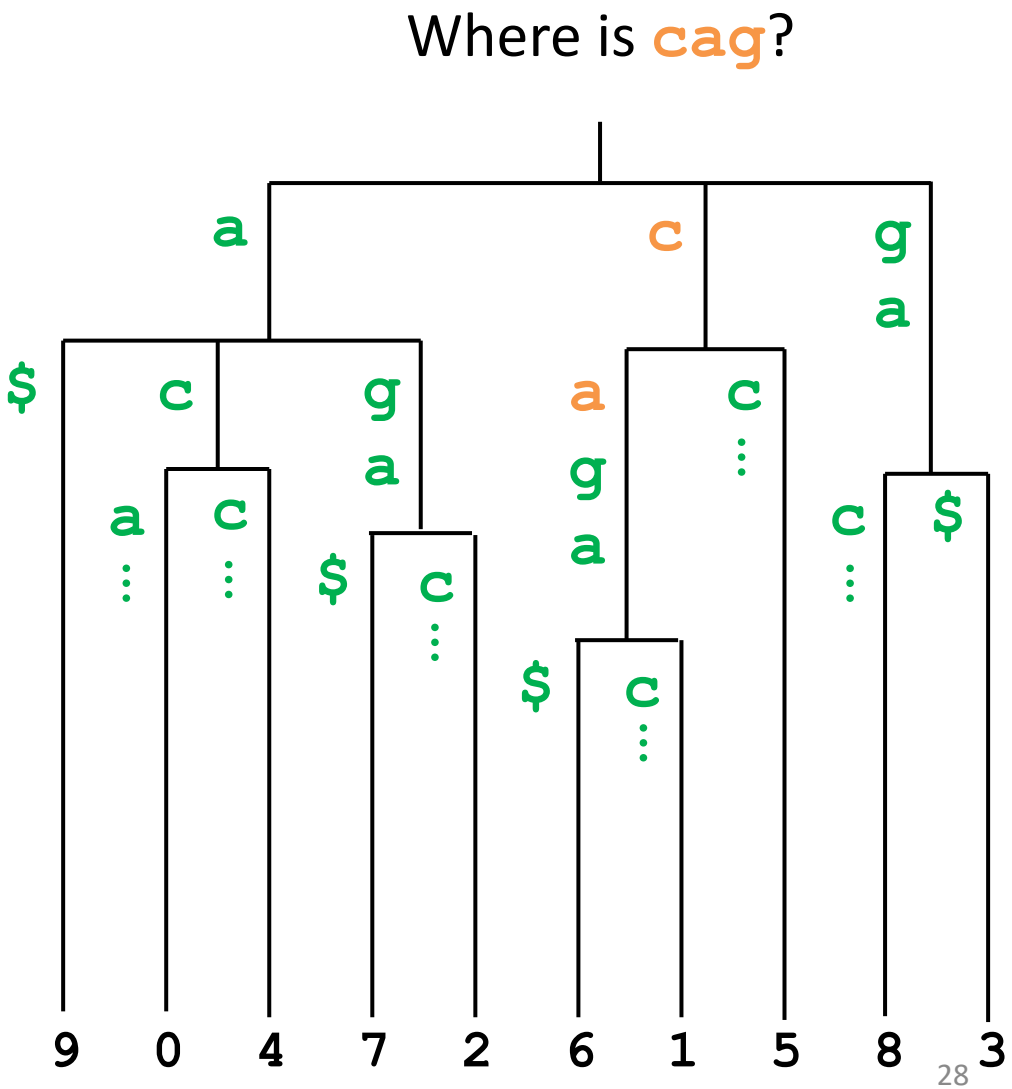
Rapid Matching vs Suffix Tree

- Can find all matches to a read in D in time proportional to read length L.



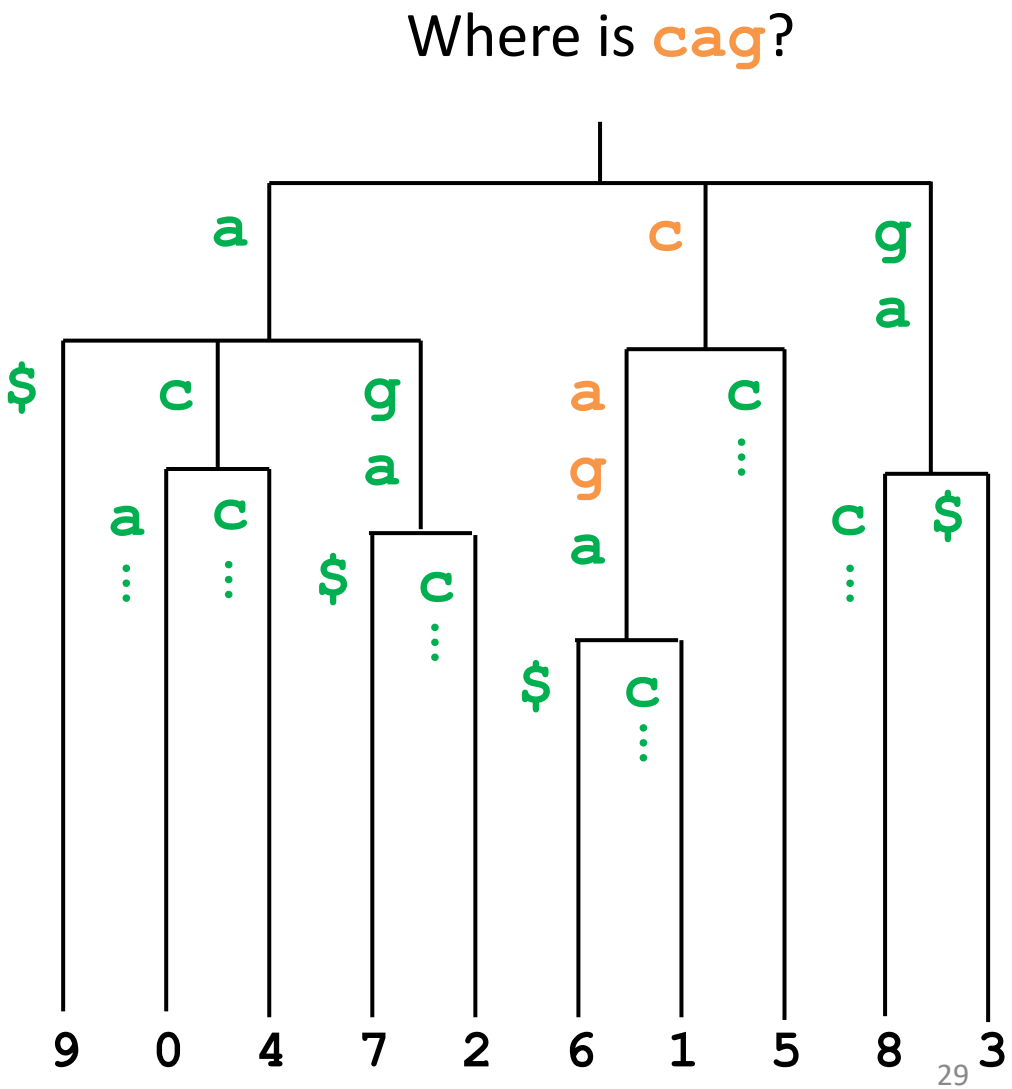
Rapid Matching vs Suffix Tree

- Can find all matches to a read in D in time proportional to read length L.



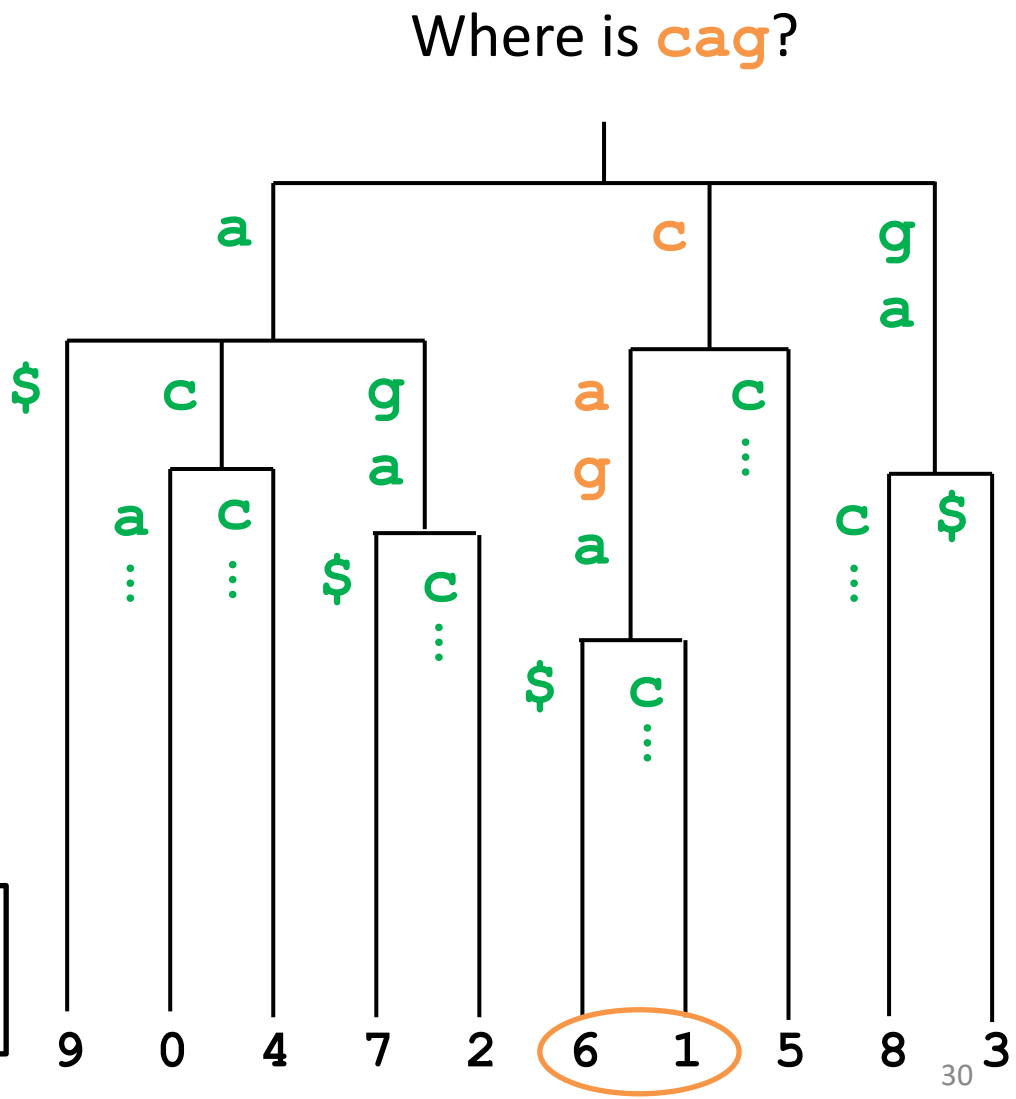
Rapid Matching vs Suffix Tree

- Can find all matches to a read in D in time proportional to read length L.



Rapid Matching vs Suffix Tree

- Can find all matches to a read in D in time proportional to read length L.



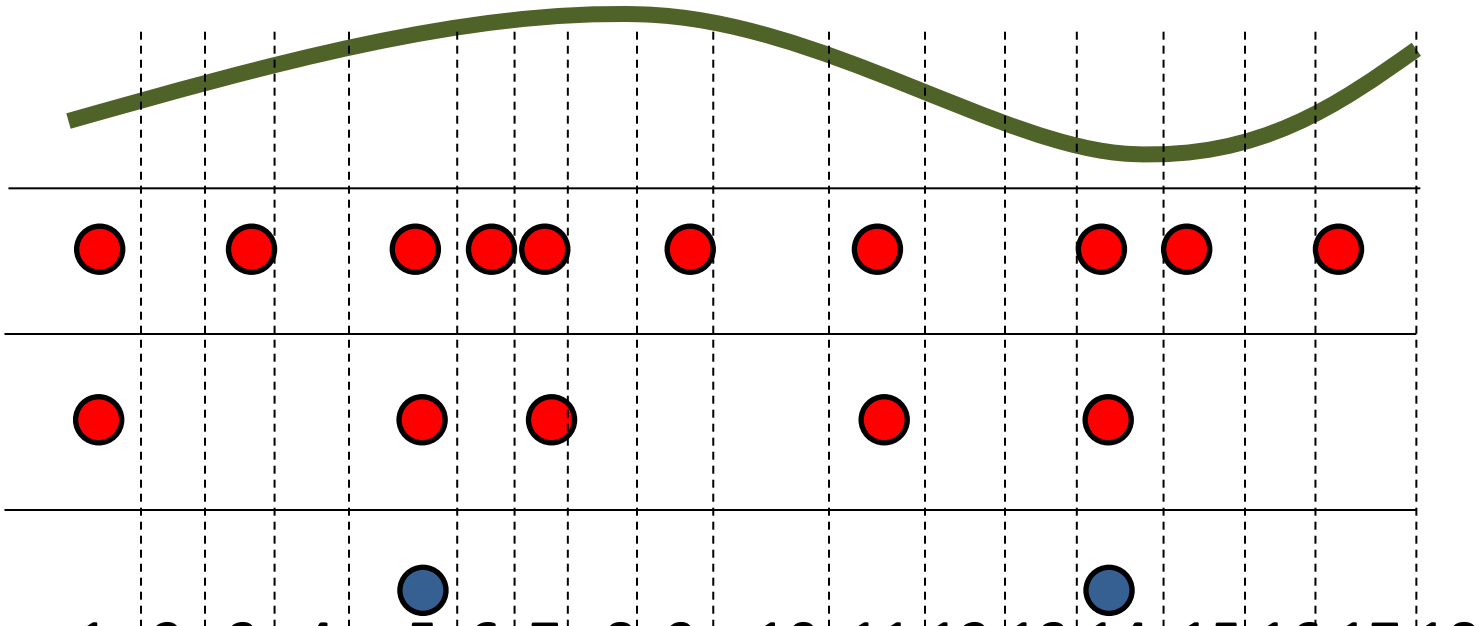
D = acagaccaga$ (0 1 2 3 4 5 6 7 8 9 10)

Extension to Inexact Matching

- To permit matches with k substitutions, try multiple paths, but charge for each mismatch.
- To permit matches with k differences, we do dynamic programming to compute edit distance of read against each path in tree.
- Descent stops for a read when we hit bottom of tree or find that path requires > k differences.

Parallel Alignment is a SIMD Computation

- We process every BLAST starting loc / every read through *same* filtering computation
- ***S**ingle **I**nstruction stream, **M**ultiple **D**ata items*



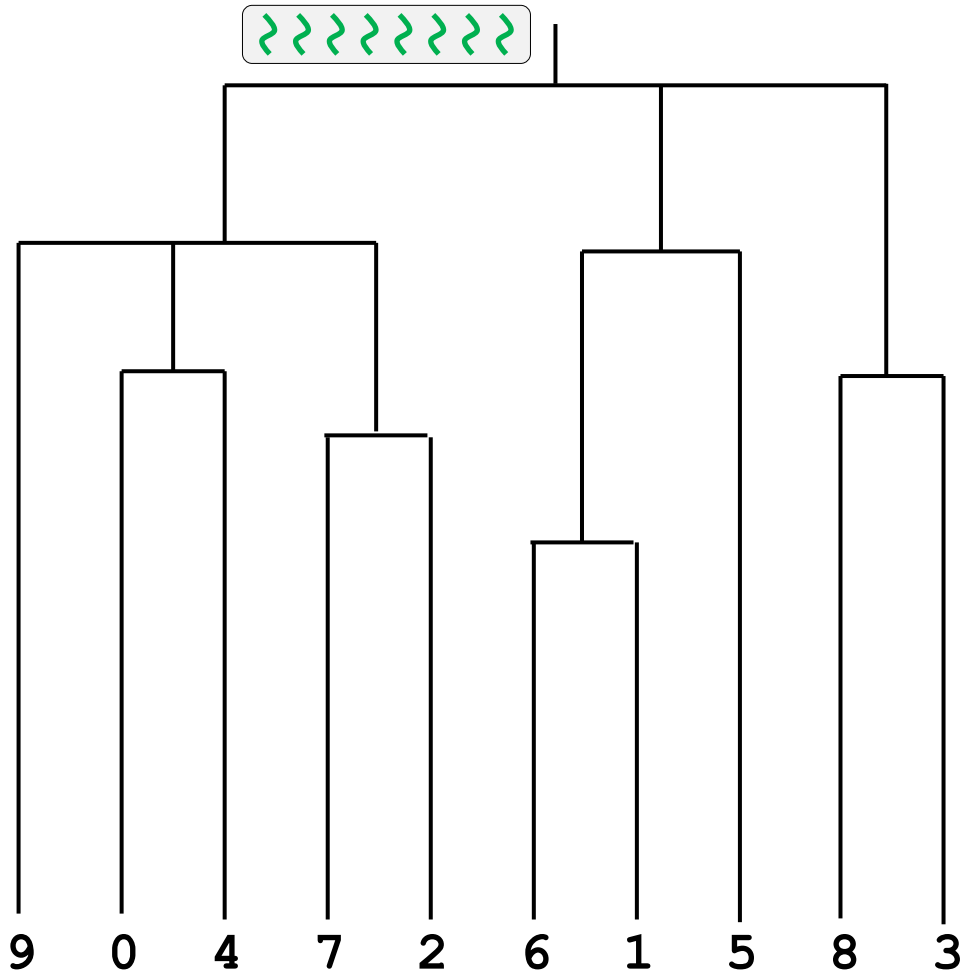
SIMD Targets

- Our work: **NVIDIA GPUs**
  (32 SIMD lanes x 4+ threads x 2-64 cores)

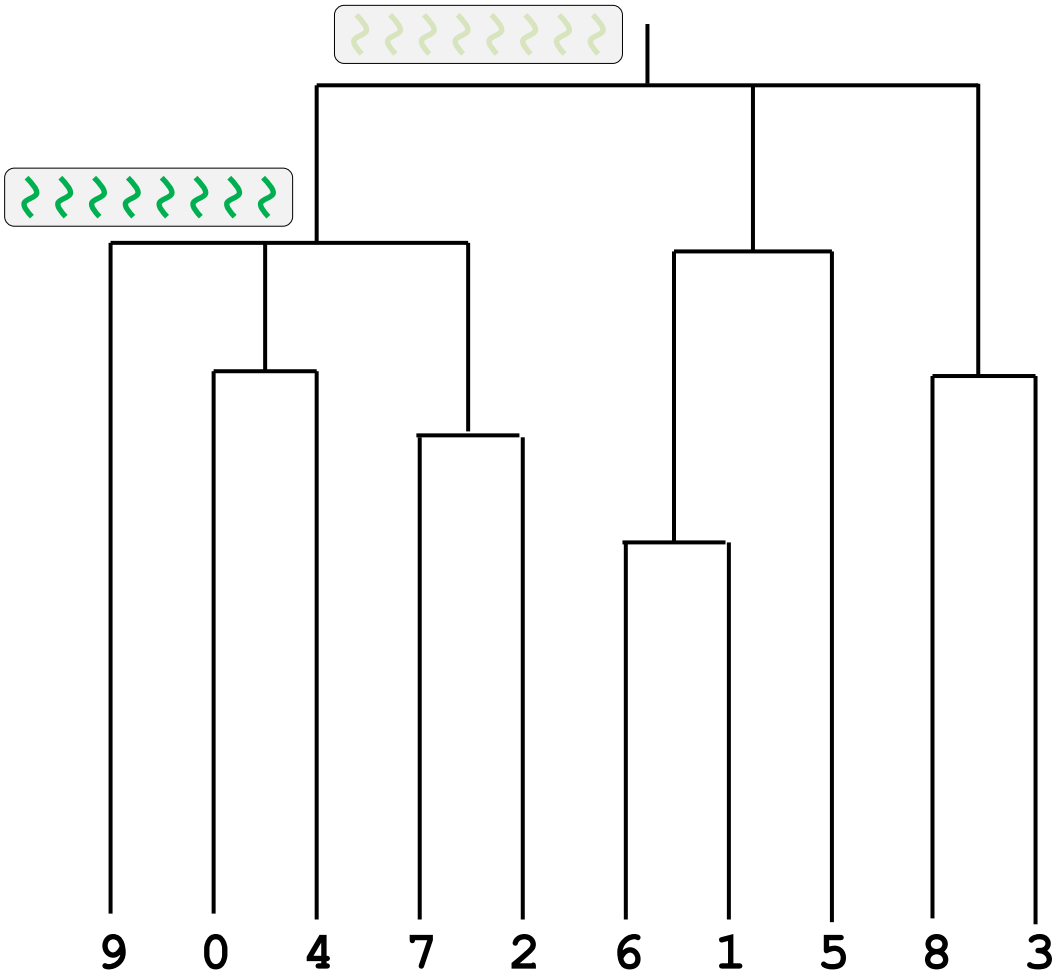
- Other possibilities: any multicore with wide vector instructions (Intel Xeon, AMD, ARM, …)

- ~All modern processors have wide SIMD!

Batched Traversal (Short Reads)

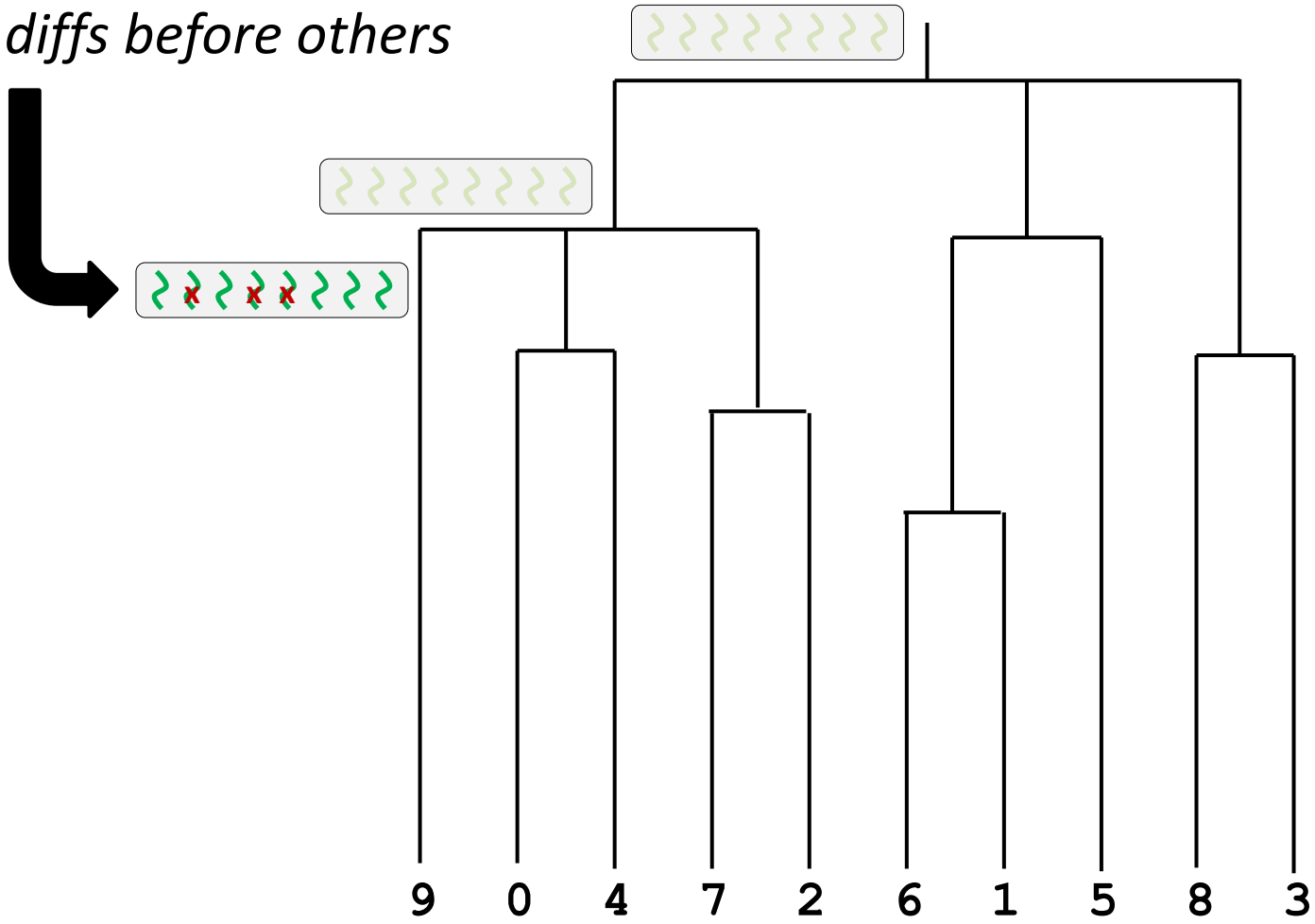


Batched Traversal

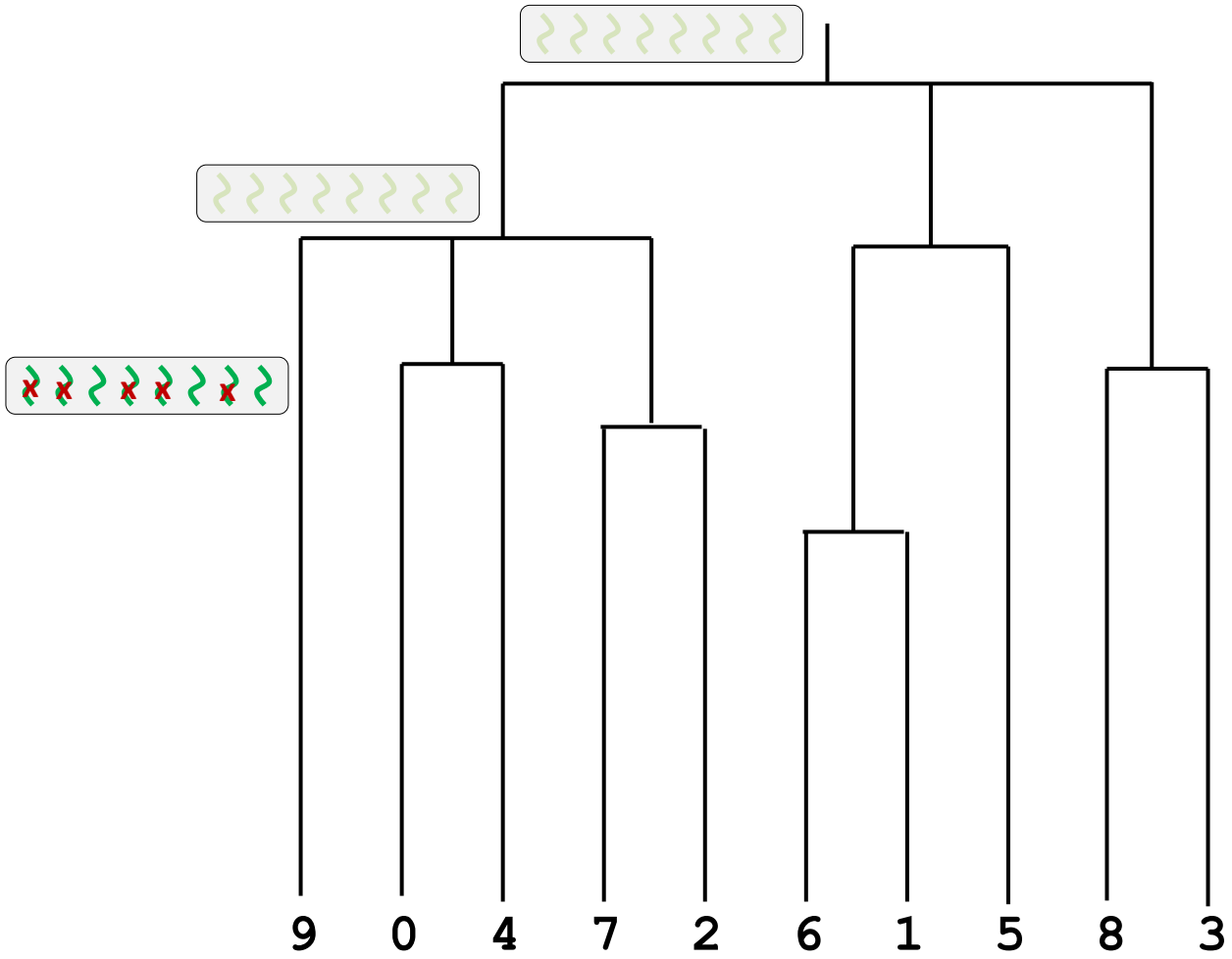


Batched Traversal

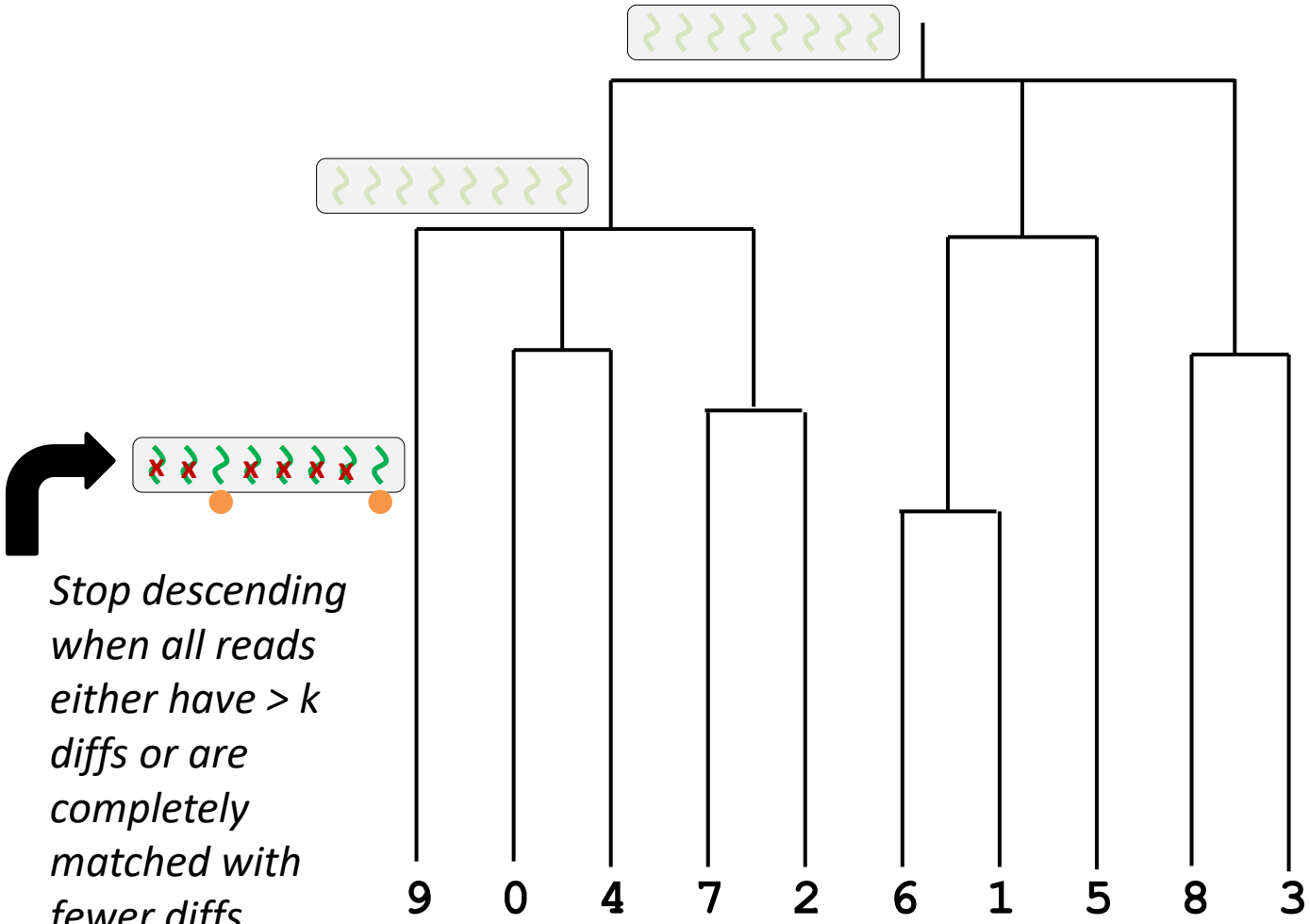
Some reads may accumulate > k diffs before others



Batched Traversal

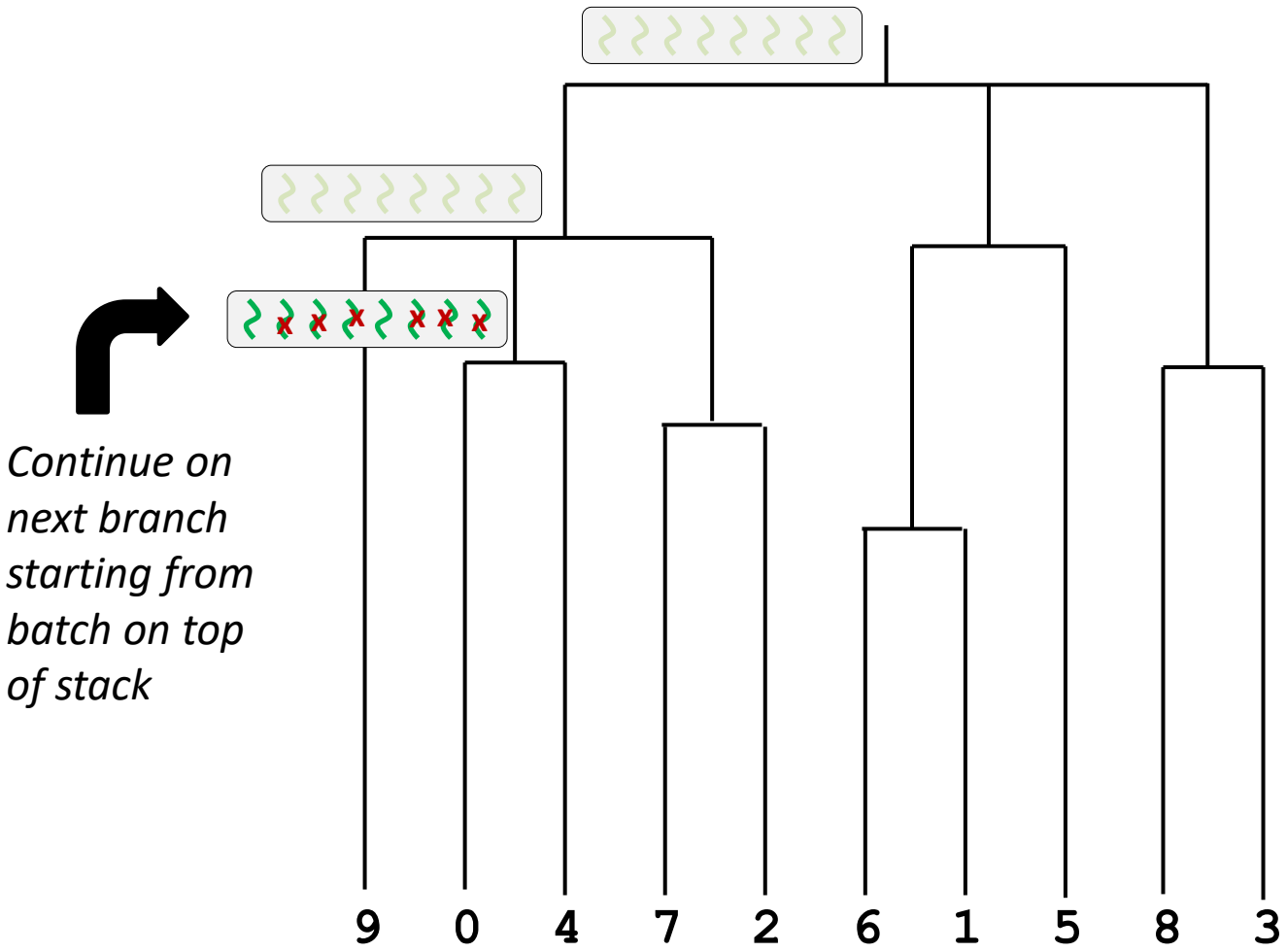


Batched Traversal

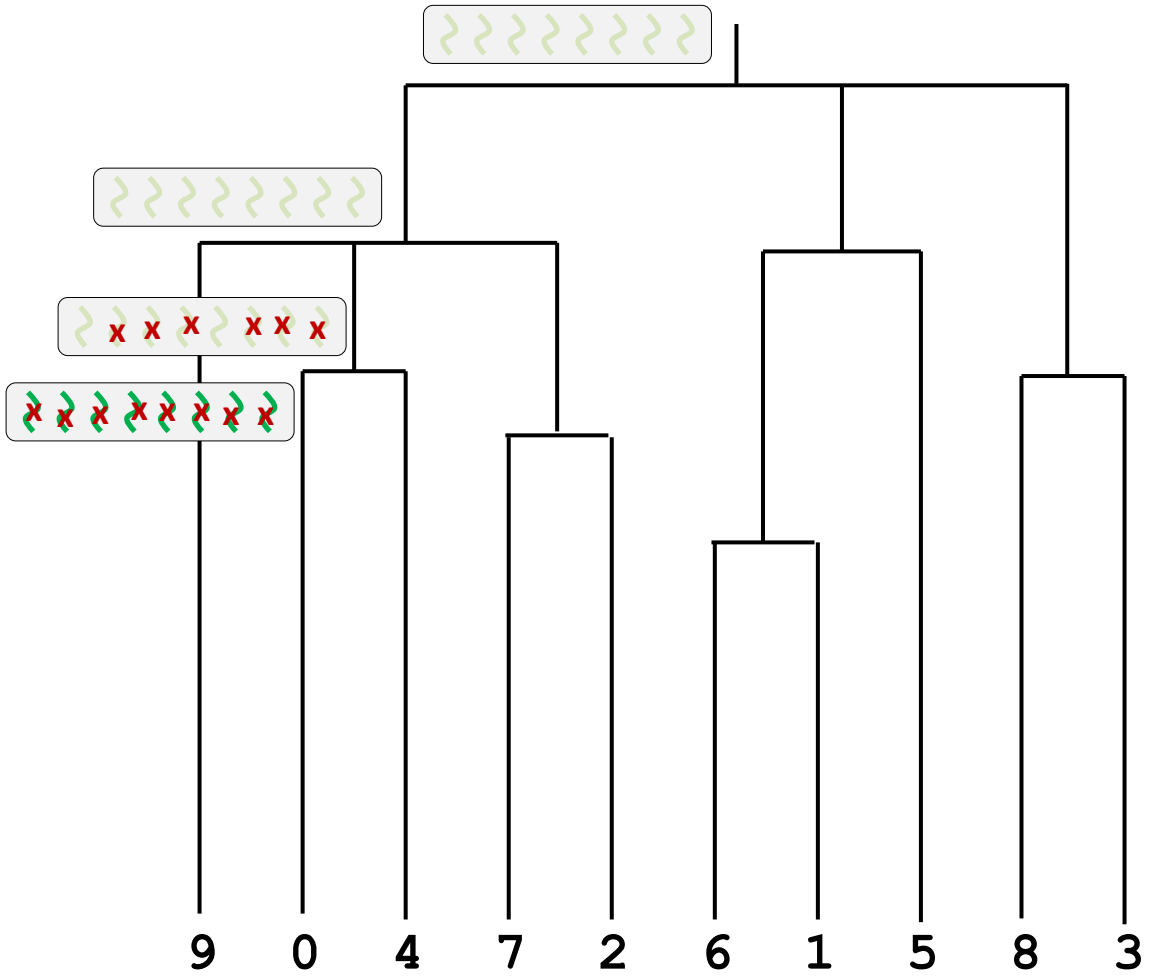


Stop descending when all reads either have > k diffs or are completely matched with fewer diffs

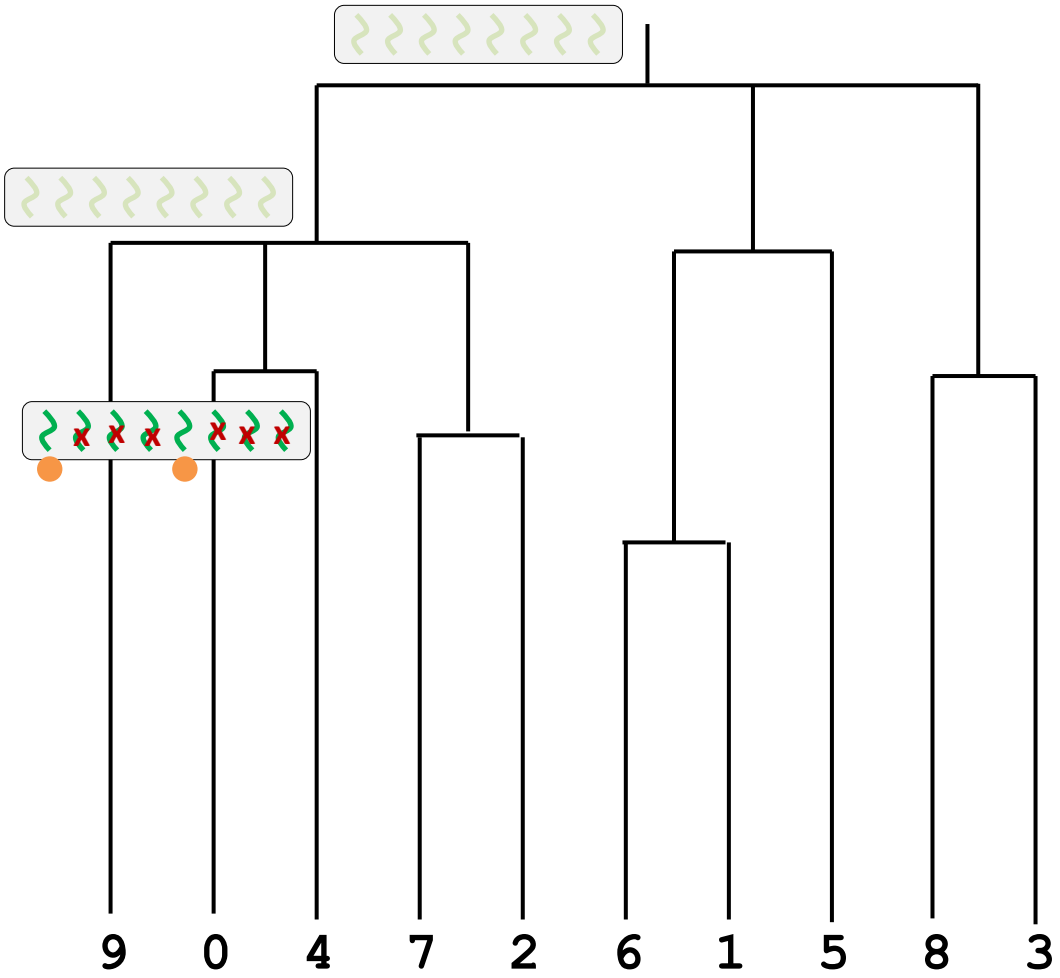
Batched Traversal



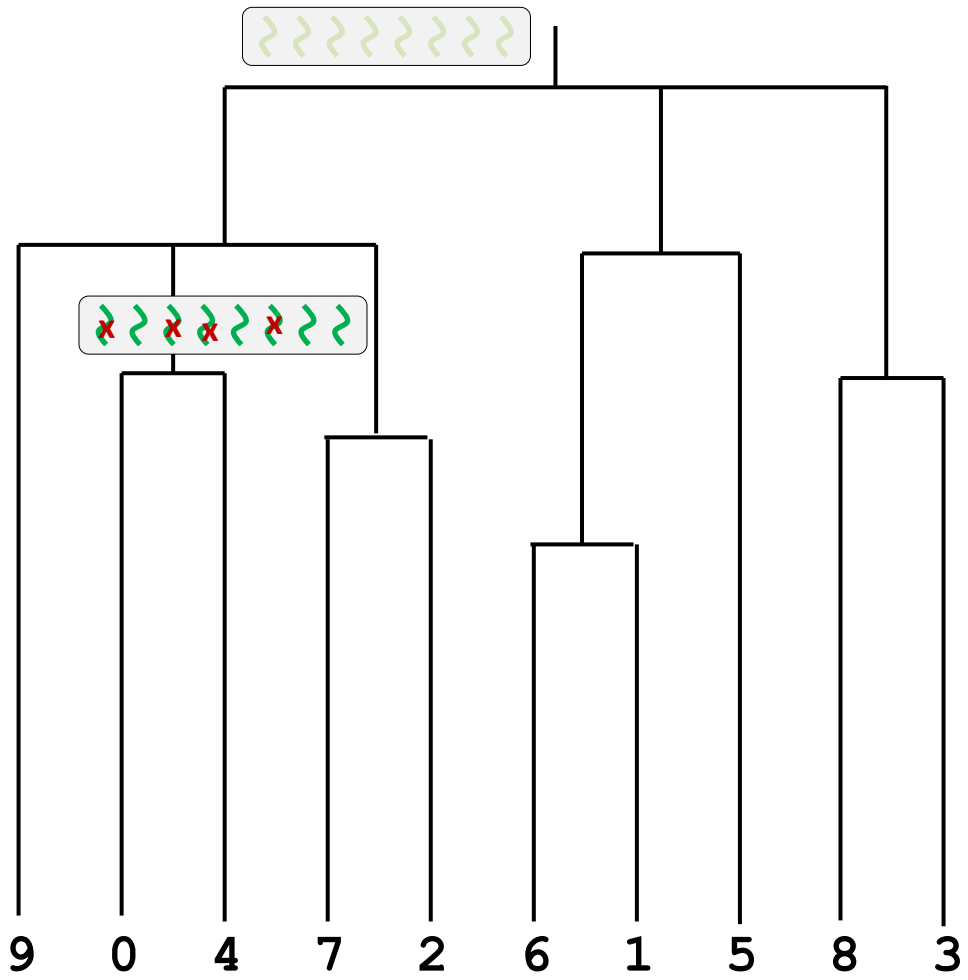
Batched Traversal



Batched Traversal

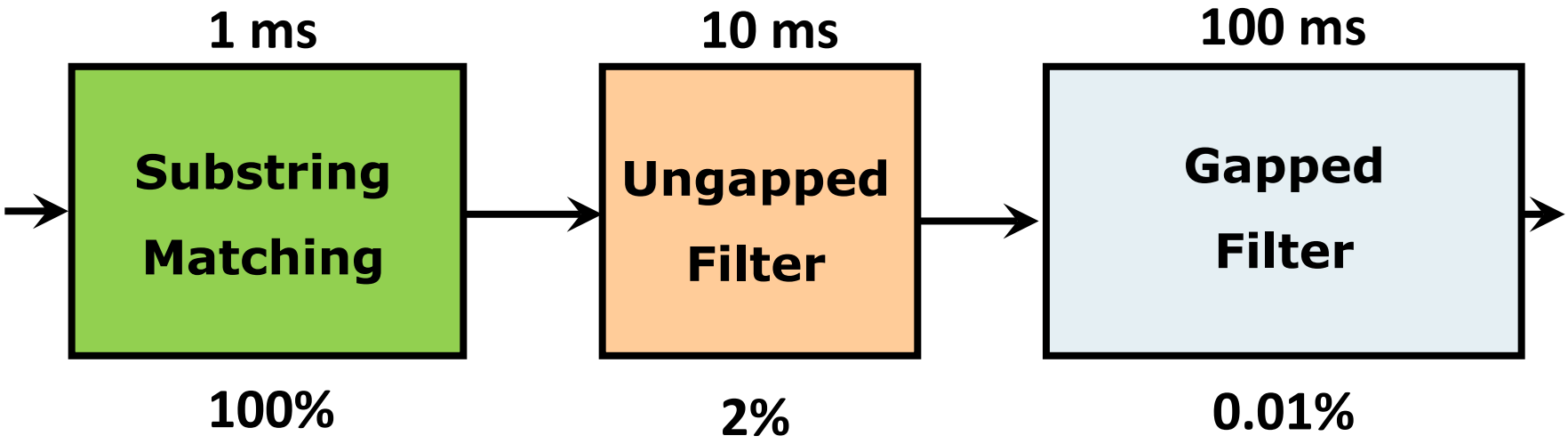


Batched Traversal



Performance?

- Each stage of BLAST costs more but processes less input.



- 98% of threads idle for 110/111 ms
- 1.99% of threads idle for 100/111 ms
- **SIMD EFFICIENCY**: 1.1%

Irregular Computations

- DNA alignment is an irregular computation: different inputs (*i.e. DB locations, reads*) require different amounts of work to process.

- Antithesis of, e.g., linear algebra calculations that are easily vectorized

- *Irregular computations are highly inefficient if naively implemented on SIMD processors.*

The Key Problem

- **How can we *efficiently* map irregular computations onto a SIMD architecture?**

Talk Overview

- Problems: DNA alignment and read mapping
- Algorithmic approach – Why SIMD?
- MERCATOR overview and performance
- Research challenges

Pause for MERCATOR demo

MERCATOR Paradigm

- Application processes a long **stream** of inputs
- **Application graph** consists of nodes (computations), edges (data transfer)
- Data flows through graph of computations
- **Irregularity**: paths differ per input, each input to a node generates 0, 1, or multiple outputs

Handling Irregularity

- Each edge between nodes has a **queue**
- MERCATOR queues inputs to a node until there are enough to fill all its SIMD lanes
- Node is only fired when it has “full ensemble” of inputs in all lanes.

Illustration of Queues

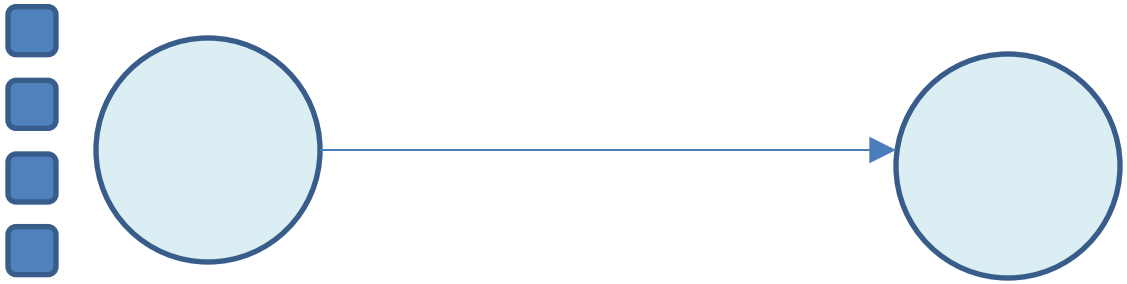


Illustration of Queues

Illustration of Queues

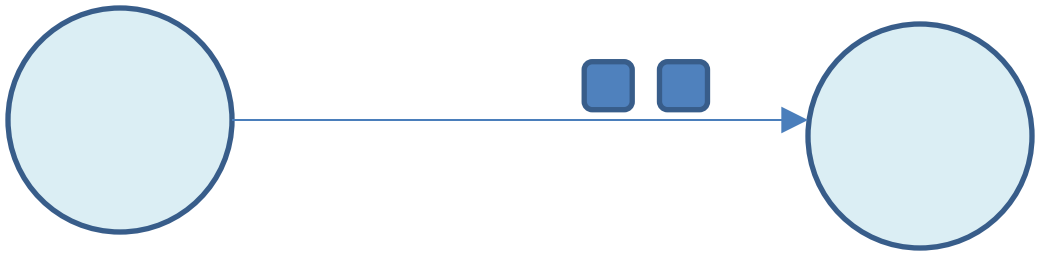


Illustration of Queues

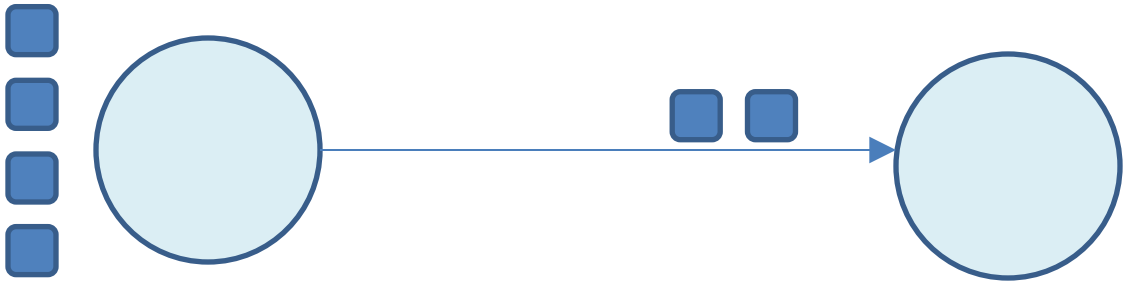


Illustration of Queues

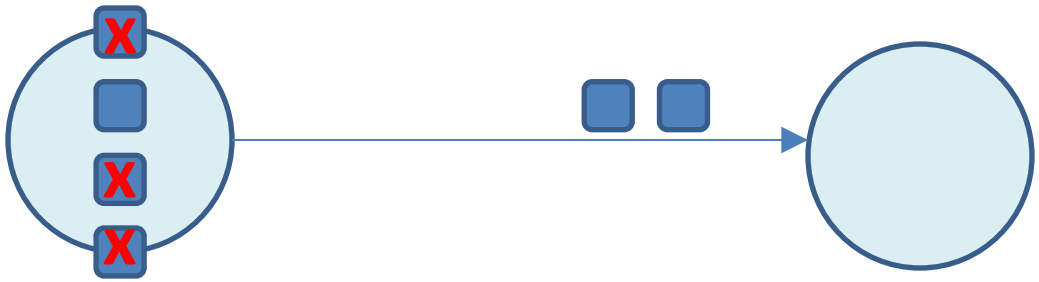


Illustration of Queues

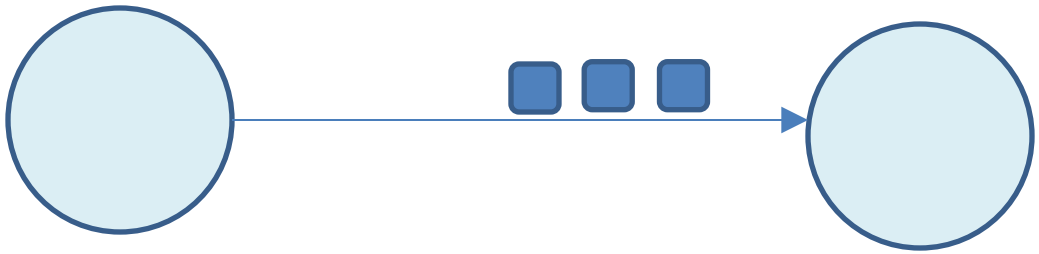


Illustration of Queues

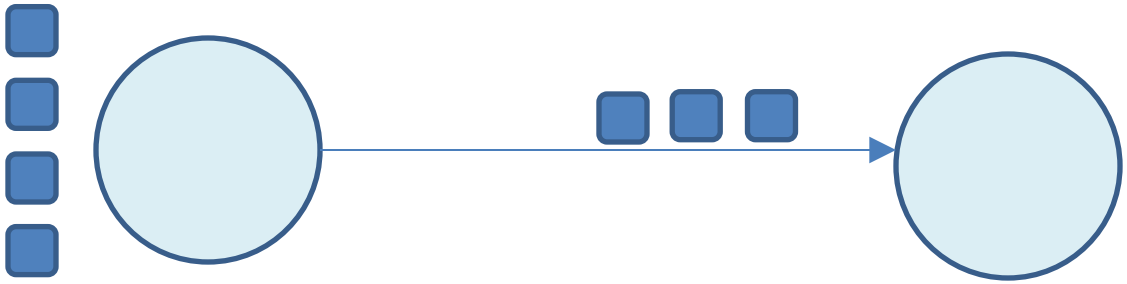


Illustration of Queues

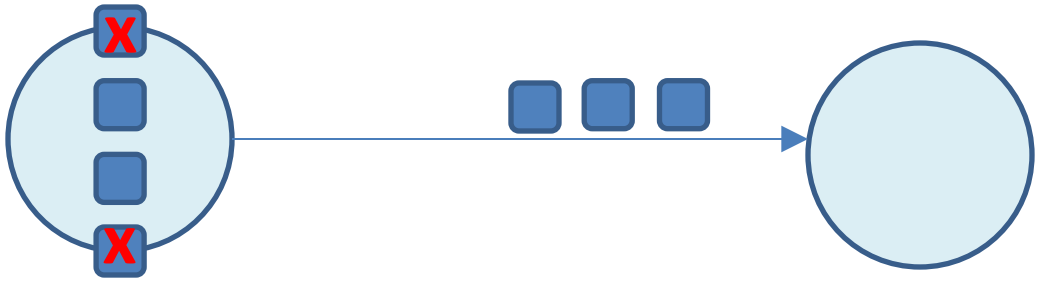


Illustration of Queues

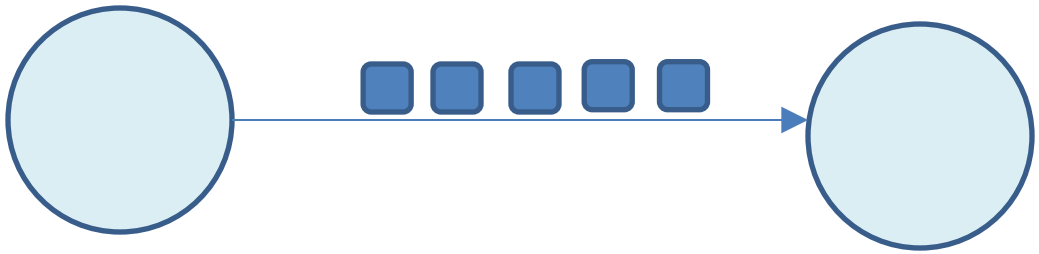


Illustration of Queues

A Few Complications

- **Shared Code** – two or more nodes may do same thing (e.g. Viola-Jones)
- **Overhead** – queueing isn't free
- **Asynchrony** – must use multiple processors, each with multiple SIMD lanes
- **Ordering** – are inputs processed "in order"?

Exploiting Shared Code

- "Module type" ←→ CUDA code
- Multiple nodes with same function have same module type
- *We execute all nodes of a given module type in parallel!*
- [Requires pulling data from each node's queue concurrently]

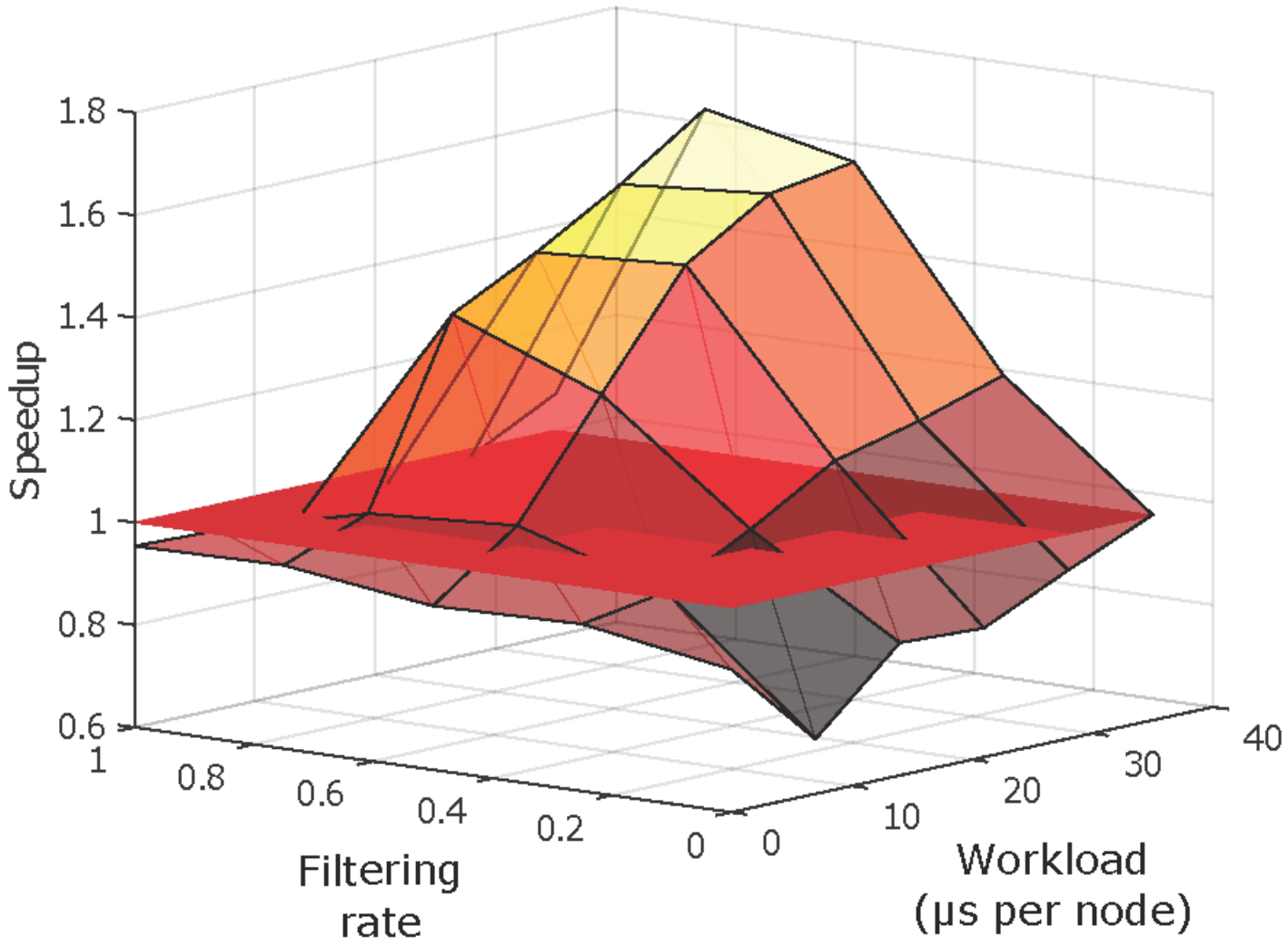
Minimizing Overhead

- Queue manipulation is itself parallelized
- *Easy case*: “read next k inputs from queue into threads 1..k.”
- *More fun*: “read k total inputs from all queues combined into threads 1..k, and remember which queue each input came from.”

Sneaky Tricks

- Parallel scan
- Branch-free binary search
- Parallel output compaction
- *[exploits, maintains input ordering]*

Results of Synthetic Trial



Dealing with Asynchrony

- Shared input / output buffers
- Output order with multiple processors?
- *[Need **stream-synchronized signaling**]*
- Associative (and commutative?) reductions

Applications with Cycles

- App graph can have *back edges*
- **Issue***:* deadlock prevention
- [*topology restrictions, queueing policy*]
- **Order preservation?**

Optimization Opportunities

- Parameter tuning (queue sizes, **scheduler**, …)
- Latency-sensitive applications *vs* occupancy
- Fusing nodes to elide queueing (at what cost to occupancy?)

Want to Play?

- *https://github.com/jdbuhler/mercator*
- MERCATOR will be a testing ground for SIMD-aware irregular streaming computation
- Many interesting problems still to be solved!
- → **thesis topics**