

# Optimizing and Parallelizing Phylogenetic Likelihood Calculations

Alexandros Stamatakis

ERA chair – Institute of Computer Science  
Fondation for Research and Technology Hellas, Greece

Associate Group Leader – Heidelberg Institute for Theoretical  
Studies, Germany

Professor – Institute for theoretical Informatics, Karlsruhe Institute  
of Technology, Germany

[www.biocomp.gr](http://www.biocomp.gr) (Crete lab)

[www.exelixis-lab.org](http://www.exelixis-lab.org) (Heidelberg lab)

# The Biodiversity Computing Group

HOME

NEWS

PUBLICATIONS

TALKS

SOFTWARE

PEOPLE

PROJECT

TEACHING

CONTACT

## The Biodiversity Computing Group

Advancing computational Biodiversity Research in Crete, Greece, and beyond.



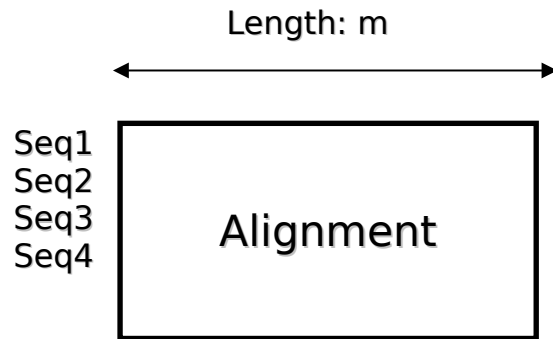
**Funded by  
the European Union**

We are hiring:  
[stamatak@ics.forth.gr](mailto:stamatak@ics.forth.gr)

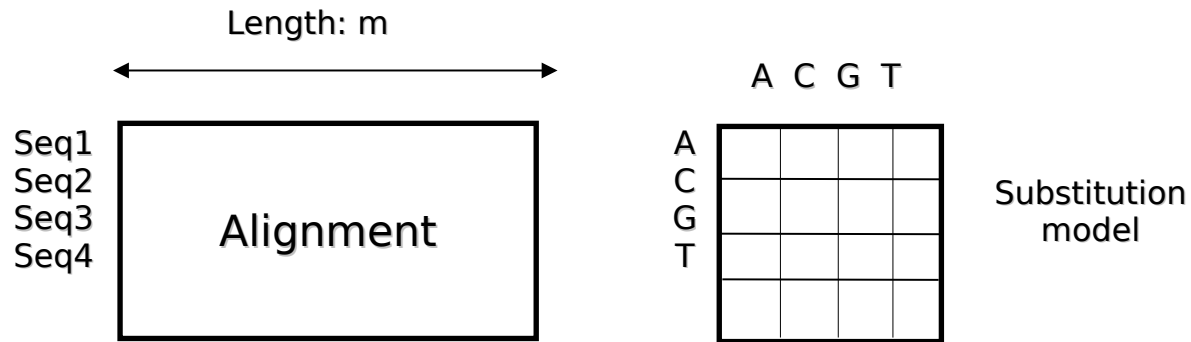
# Outline

- **Maximum Likelihood (Recap)**
- Sequential Optimization
- Parallelization
- Parallel I/O
- Numerical Nightmares
- Energy Efficiency

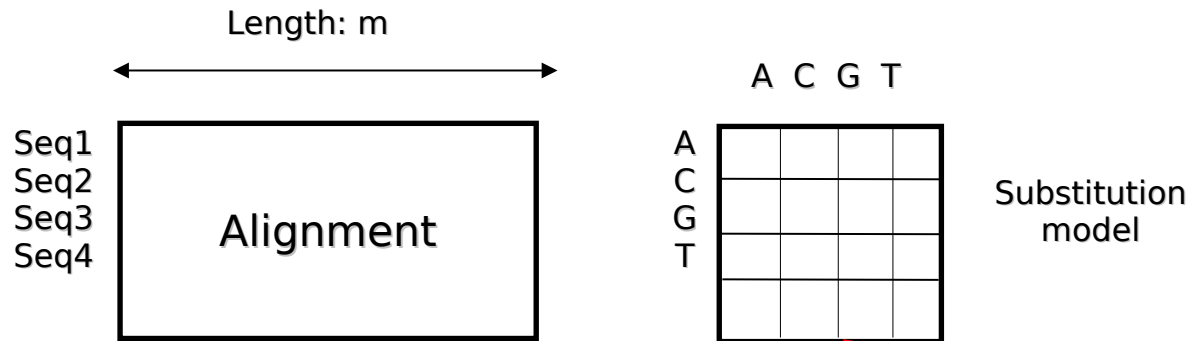
# Maximum Likelihood



# Maximum Likelihood

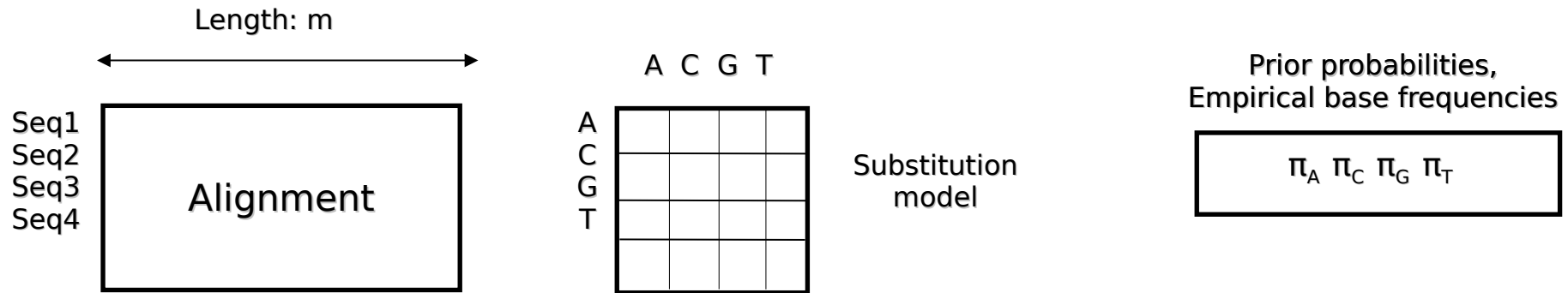


# Maximum Likelihood

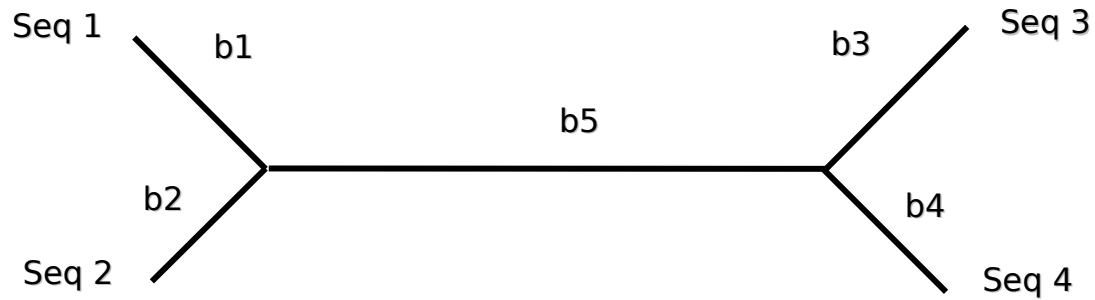
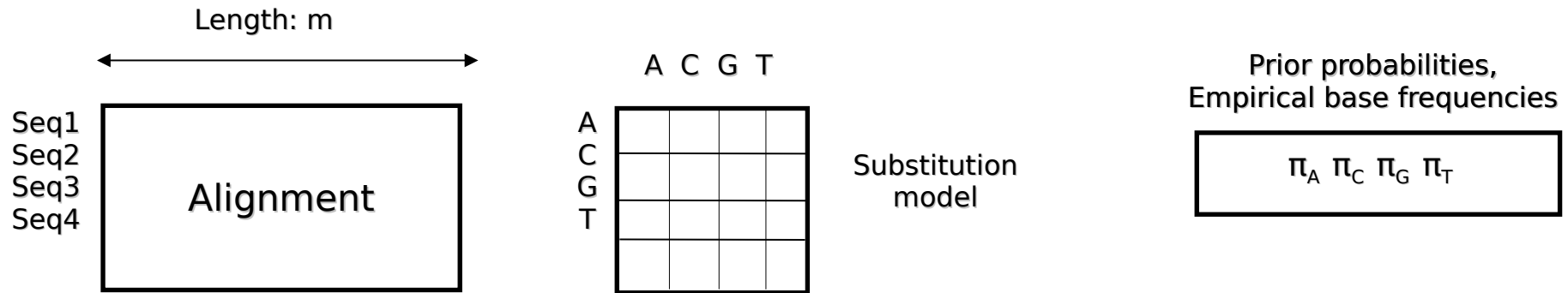


Commonly denoted as  $Q$  matrix:  
transition rates for time  $dt$ , for time  
 $t$ :  $P(t) = e^{Qt}$

# Maximum Likelihood

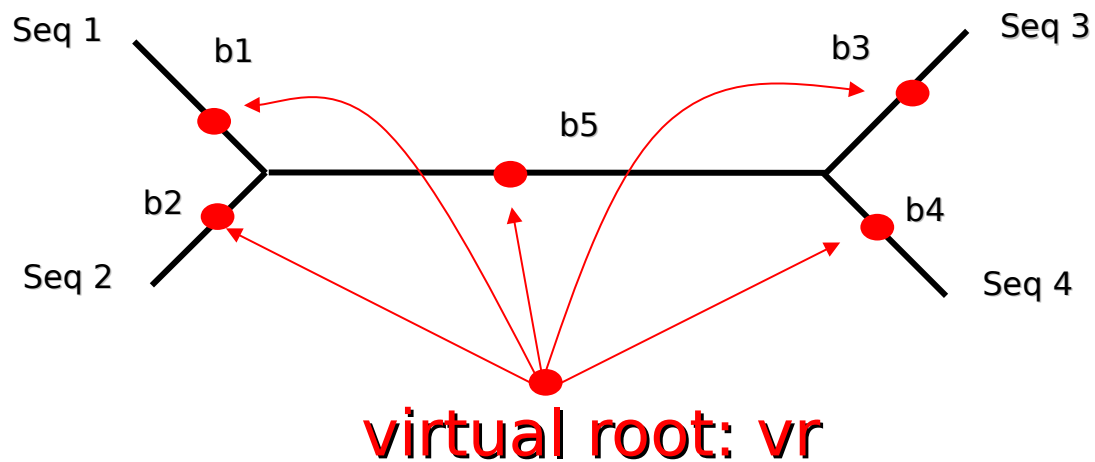
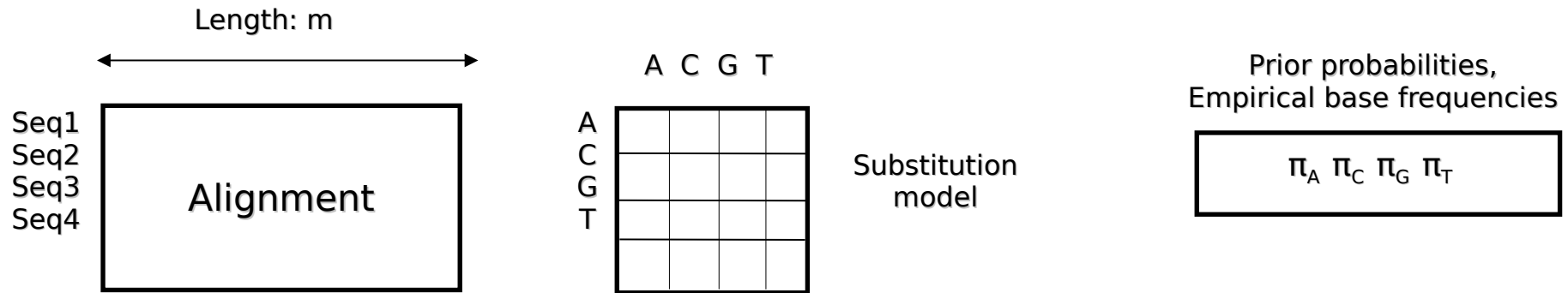


# Maximum Likelihood

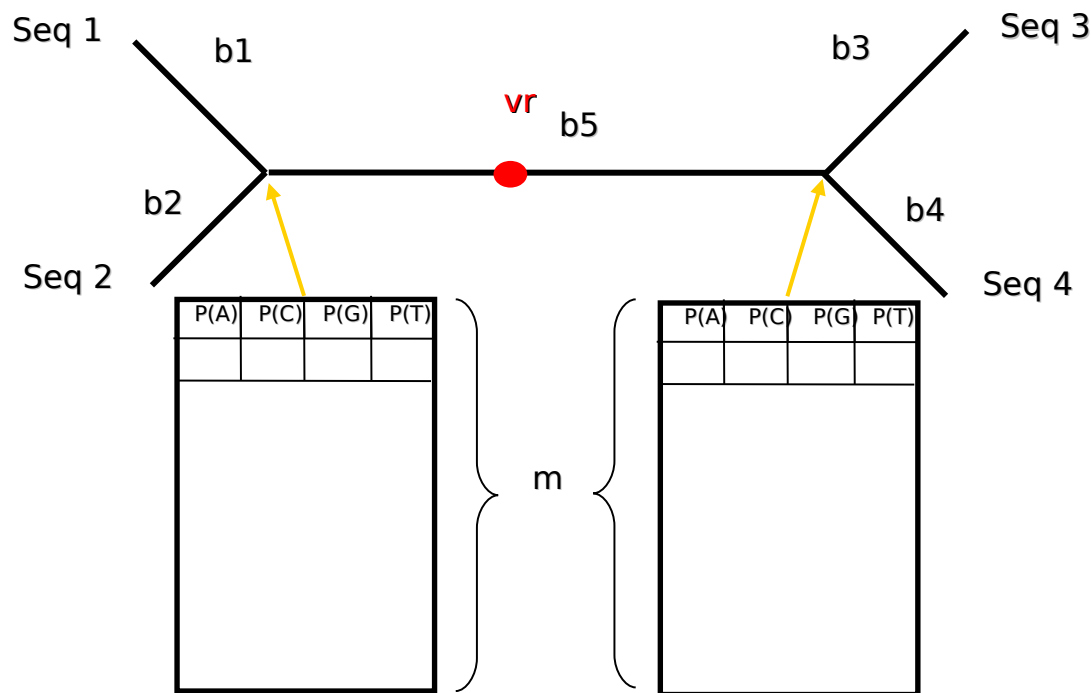
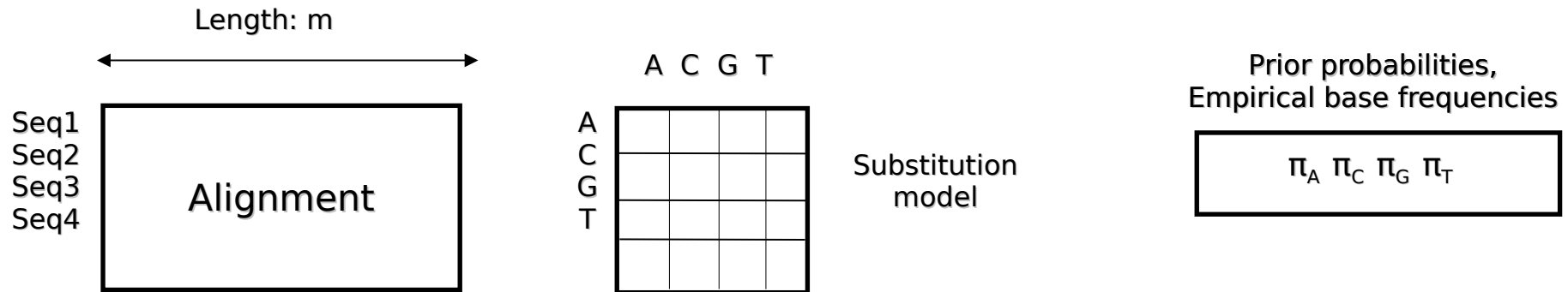




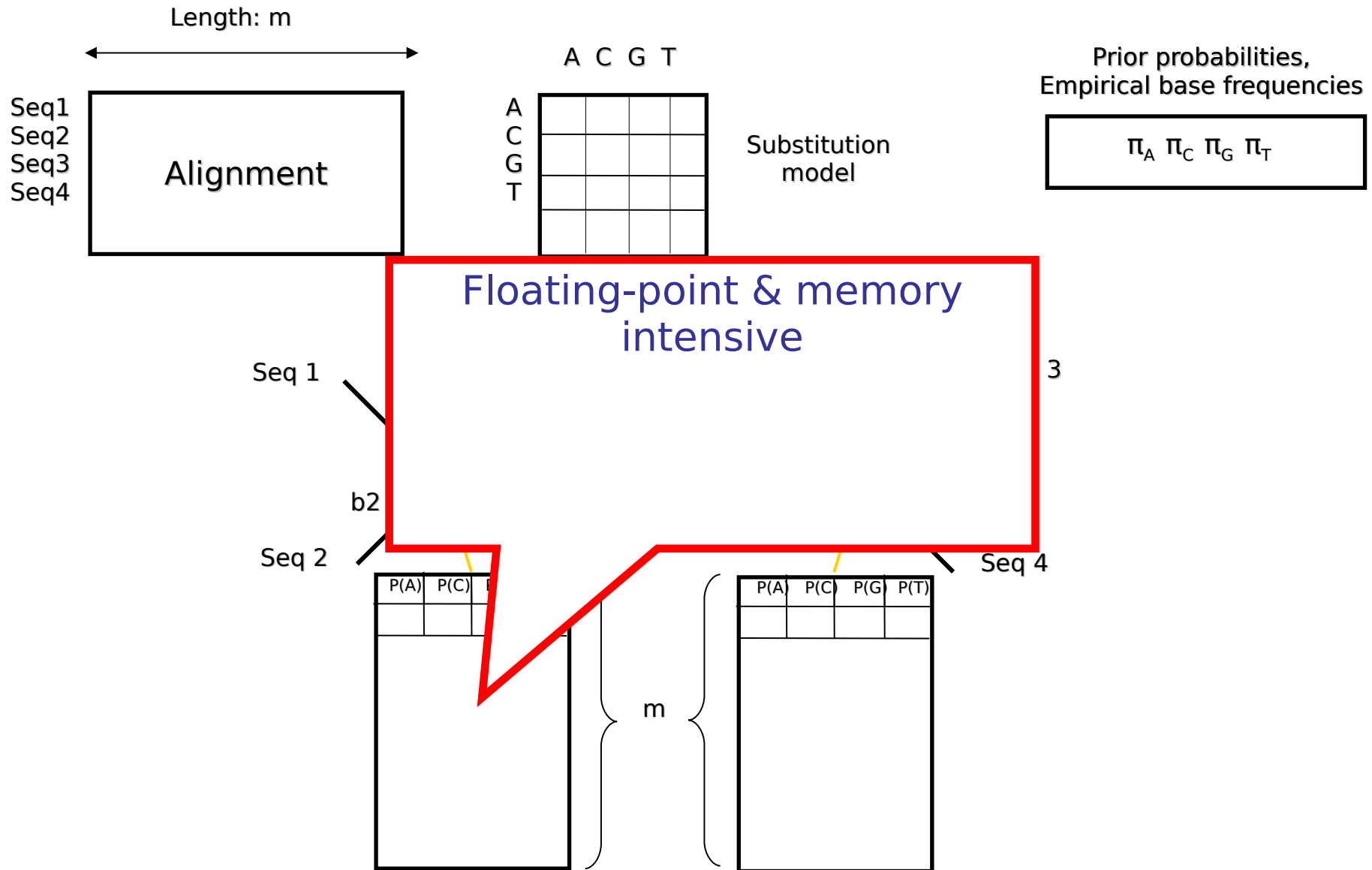
# Maximum Likelihood



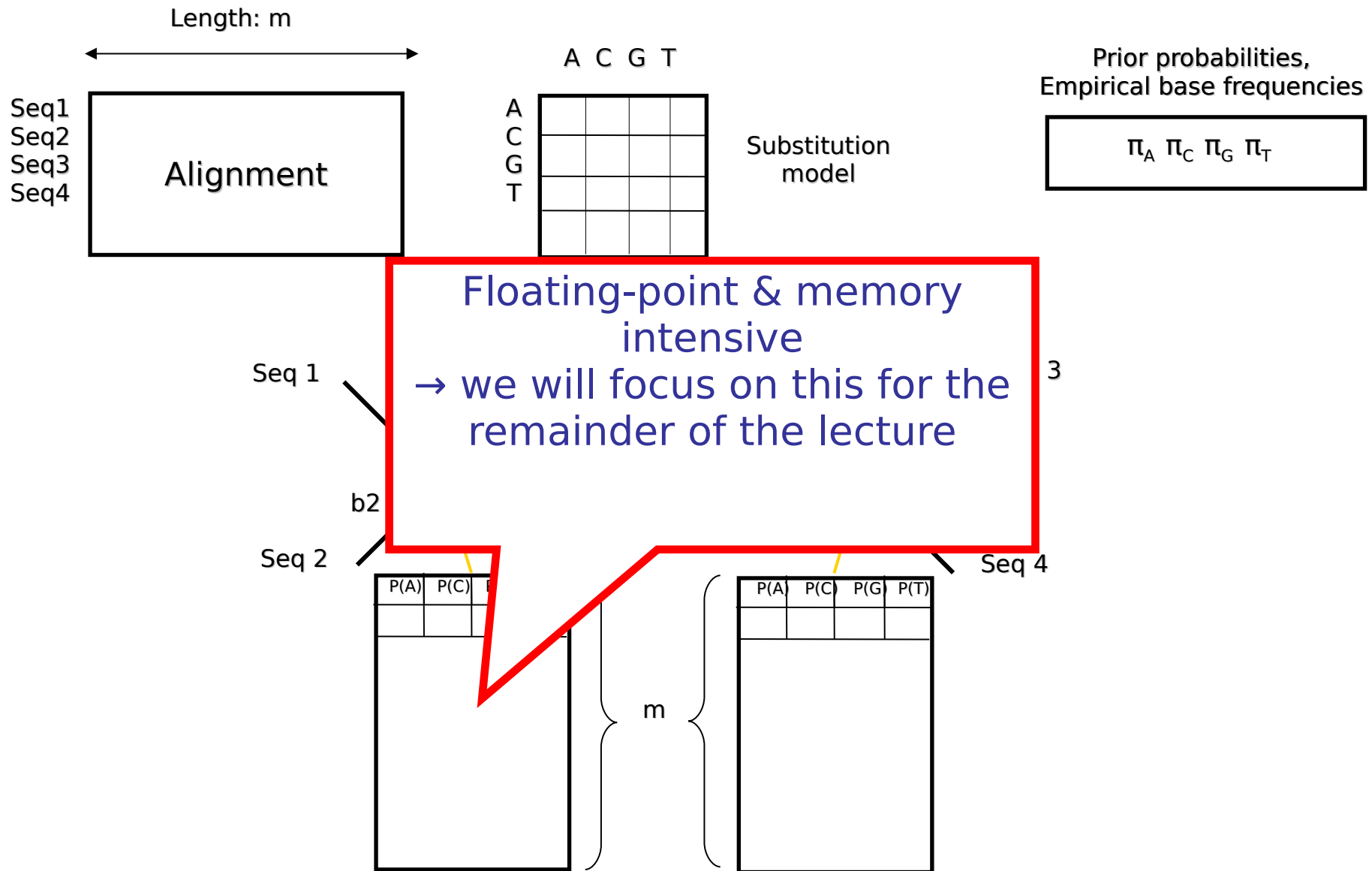
# Maximum Likelihood



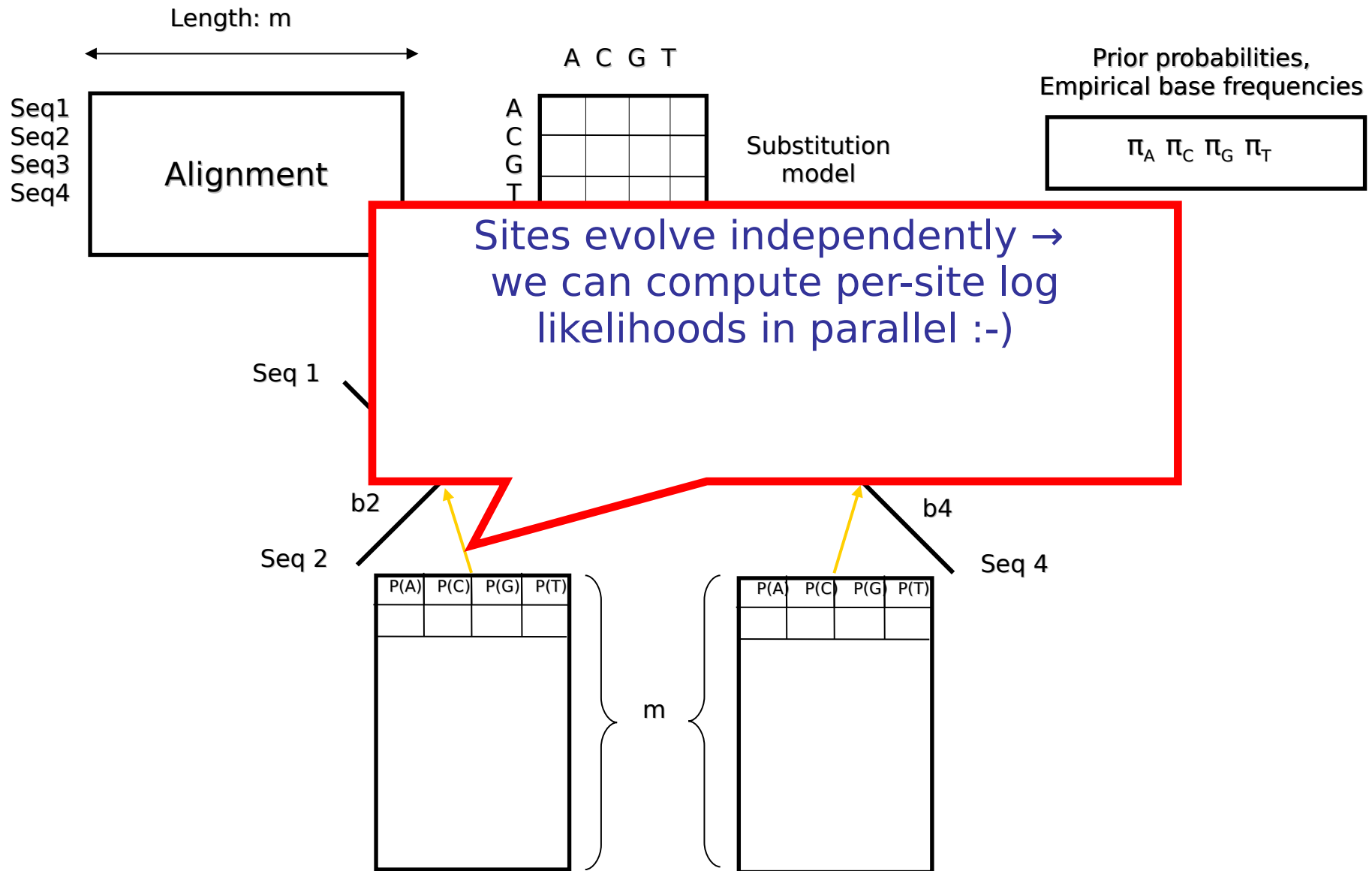
# Maximum Likelihood



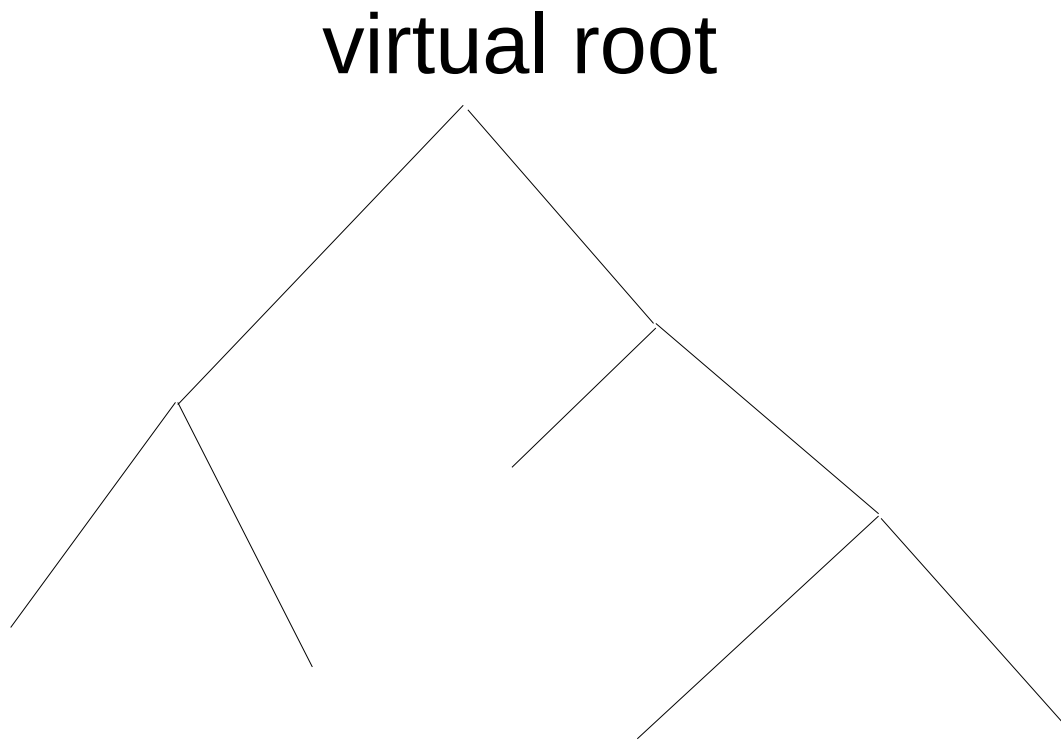
# Maximum Likelihood



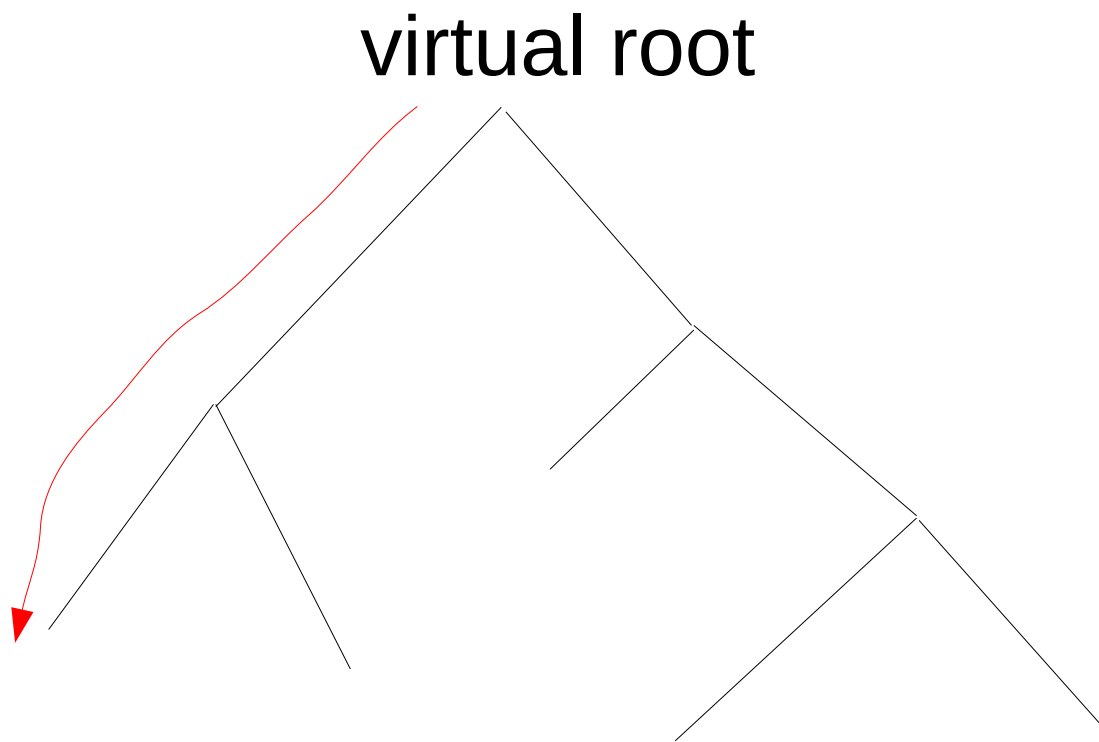
# Maximum Likelihood



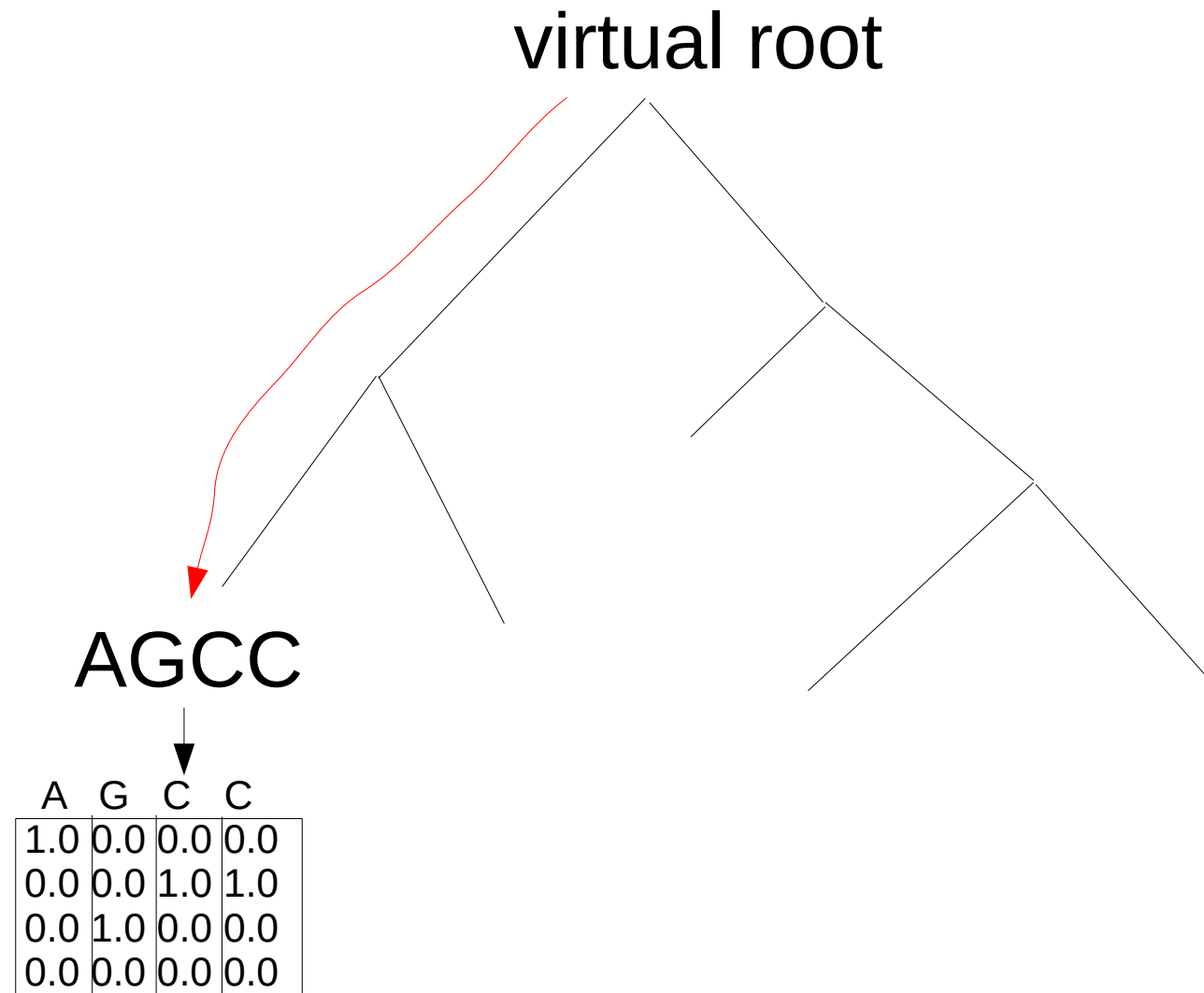
# Post-order Traversal



# Post-order Traversal



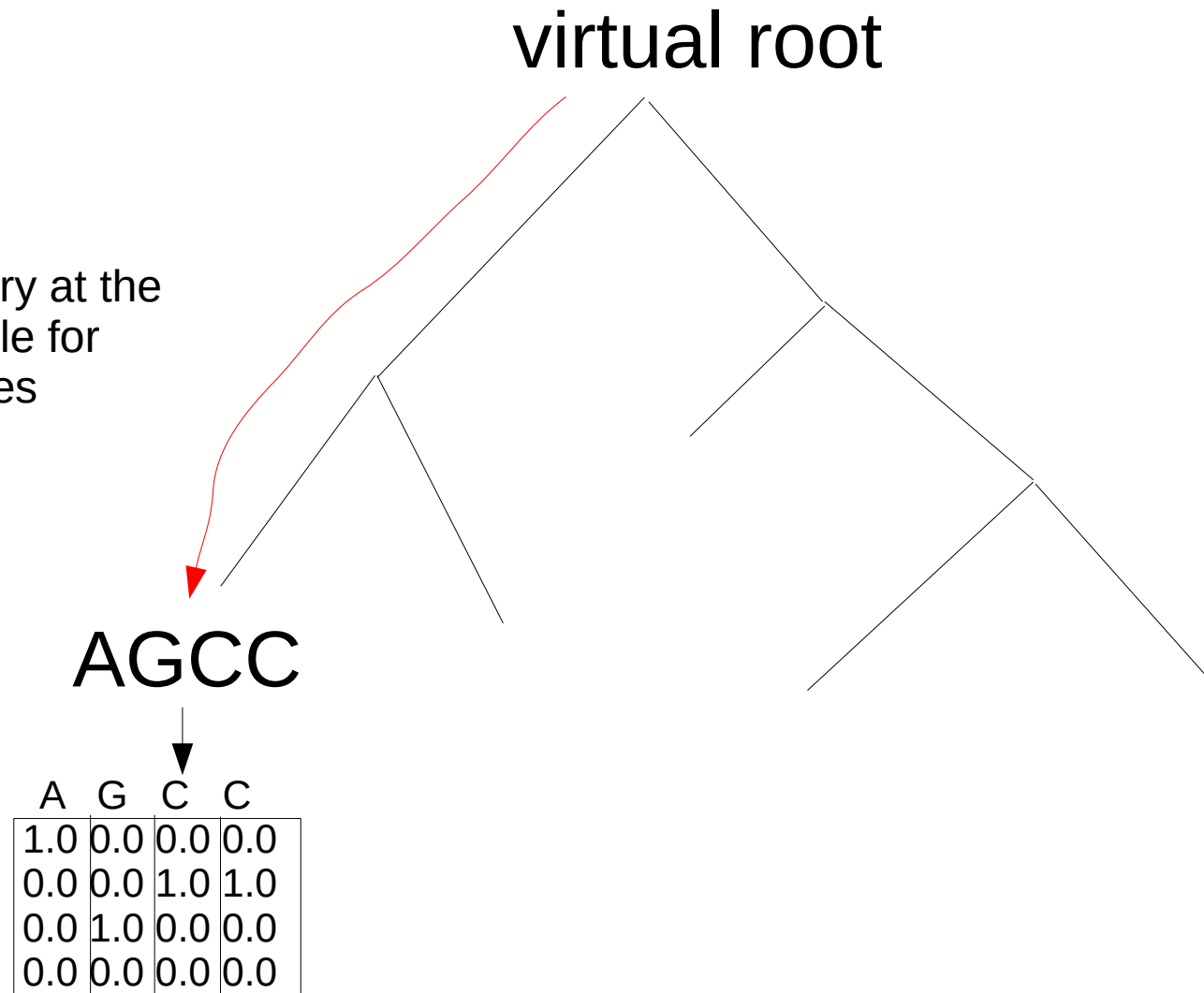
# Post-order Traversal



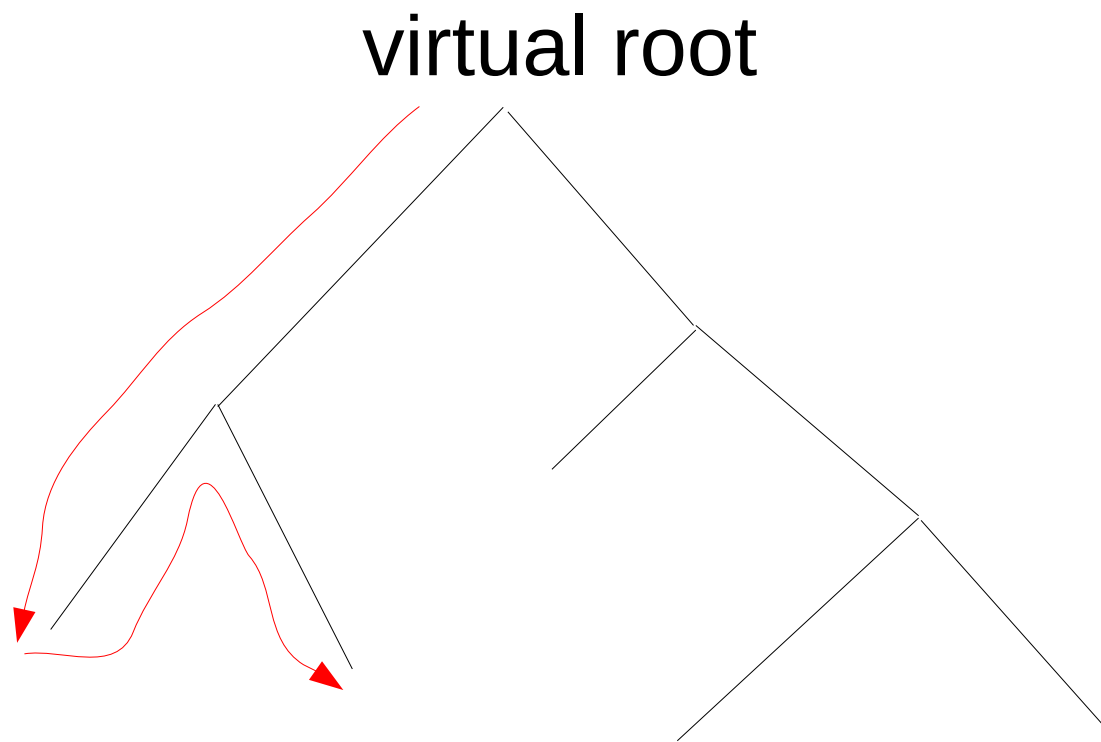


# Post-order Traversal

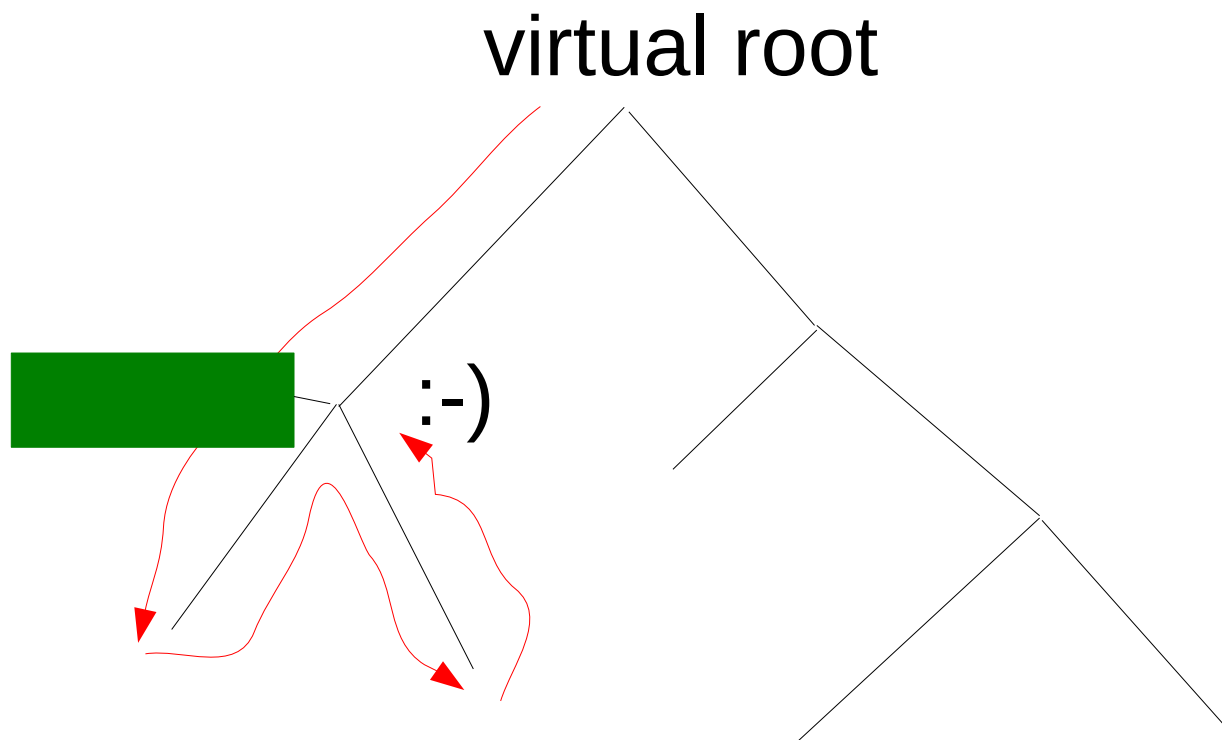
We can save memory at the tips via a lookup table for these constant values



# Post-order Traversal

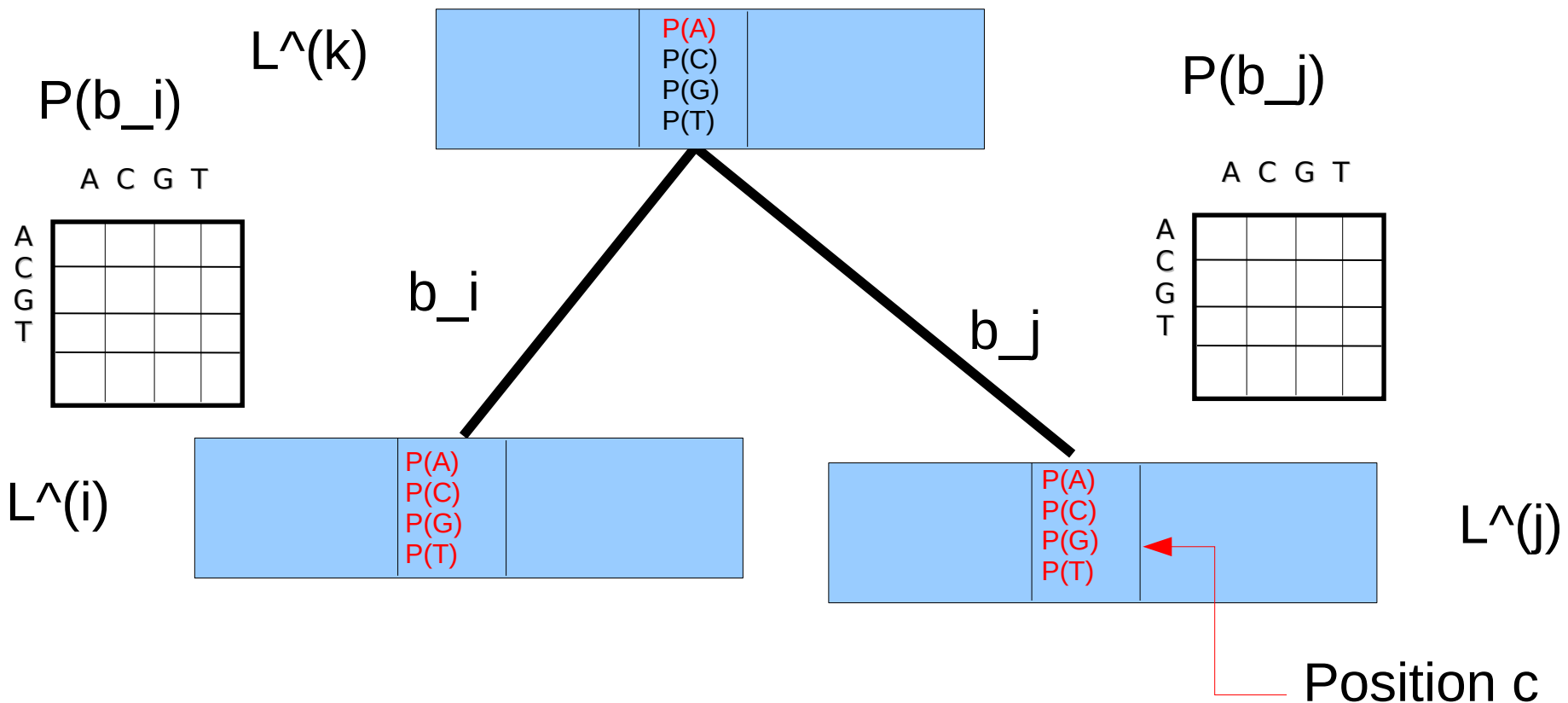


# Post-order Traversal



# What happens when we compute this inner vector?

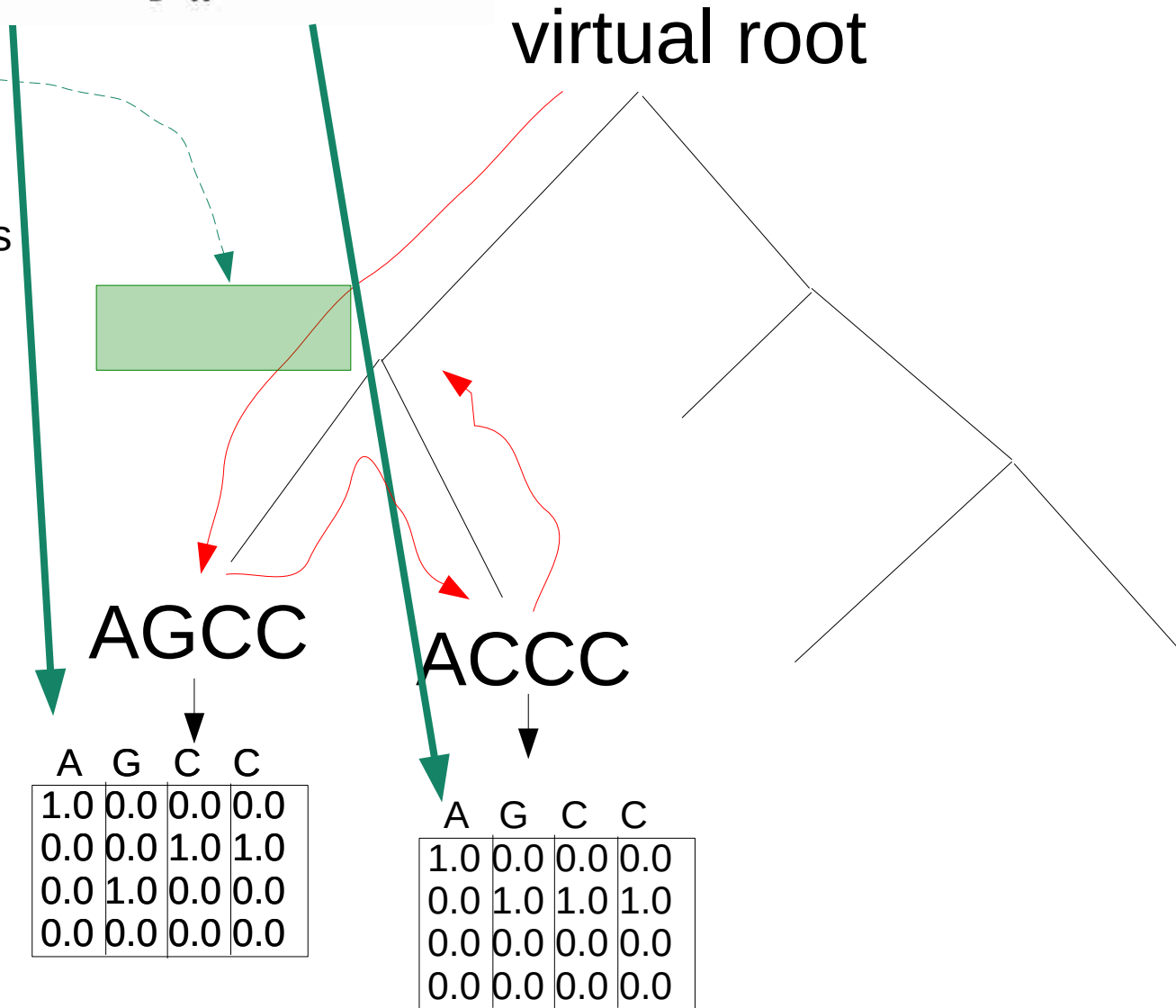
$$\vec{L}_A^{(k)}(c) = \left( \sum_{S=A}^T P_{AS}(b_i) \vec{L}_S^{(i)}(c) \right) \left( \sum_{S=A}^T P_{AS}(b_j) \vec{L}_S^{(j)}(c) \right)$$



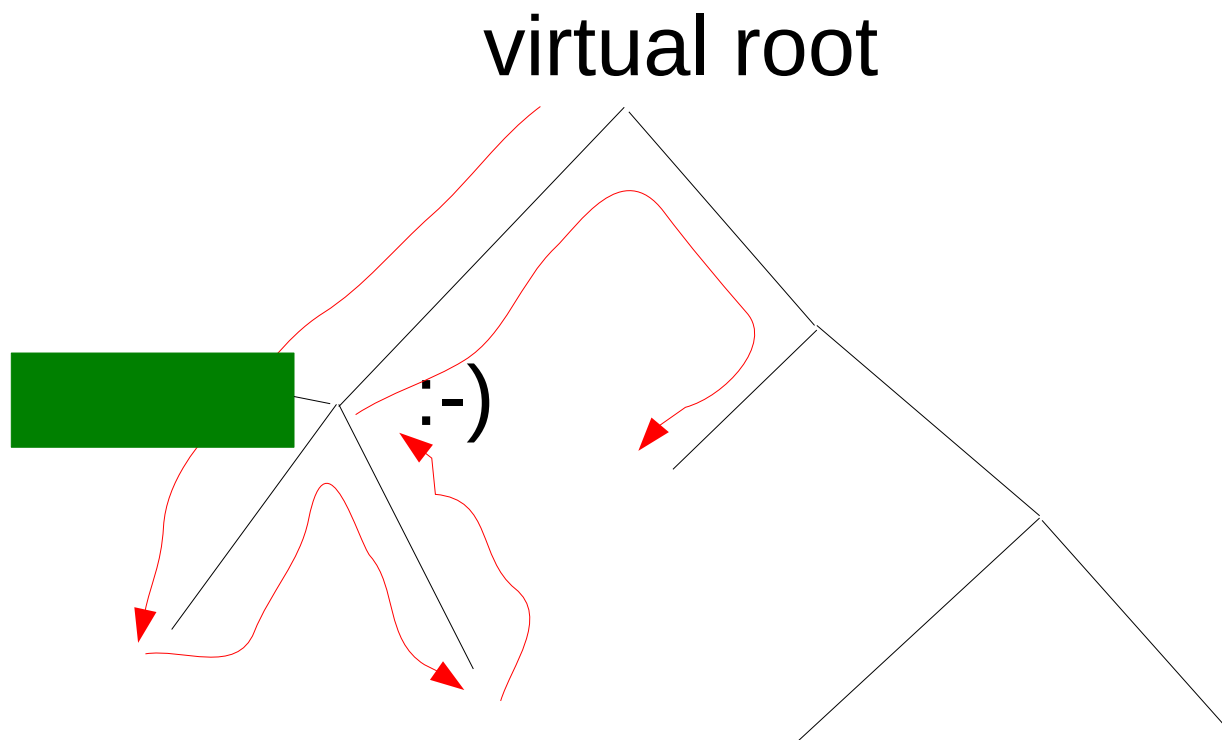
# Post-order Traversal

$$\vec{L}_A^{(k)}(c) = \left( \sum_{S=A}^T P_{AS}(b_i) \vec{L}_S^{(i)}(c) \right) \left( \sum_{S=A}^T P_{AS}(b_j) \vec{L}_S^{(j)}(c) \right)$$

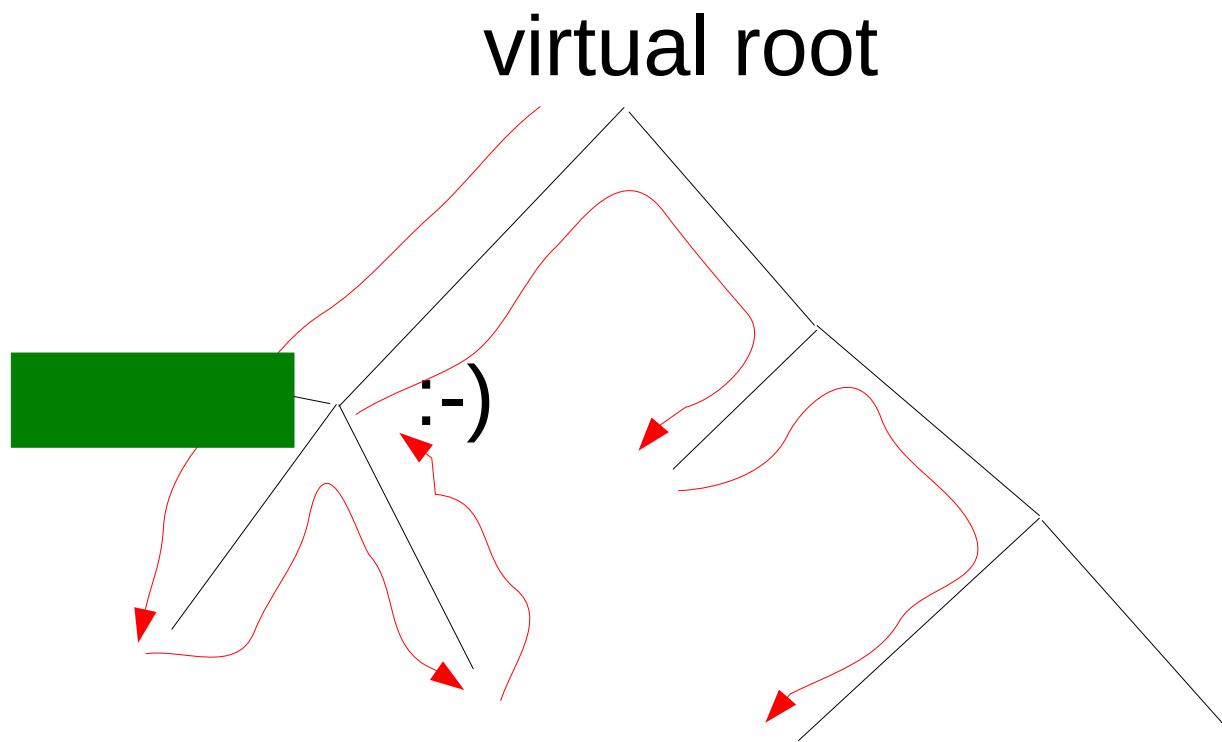
Simplify equation  
as most summands  
are 0 !



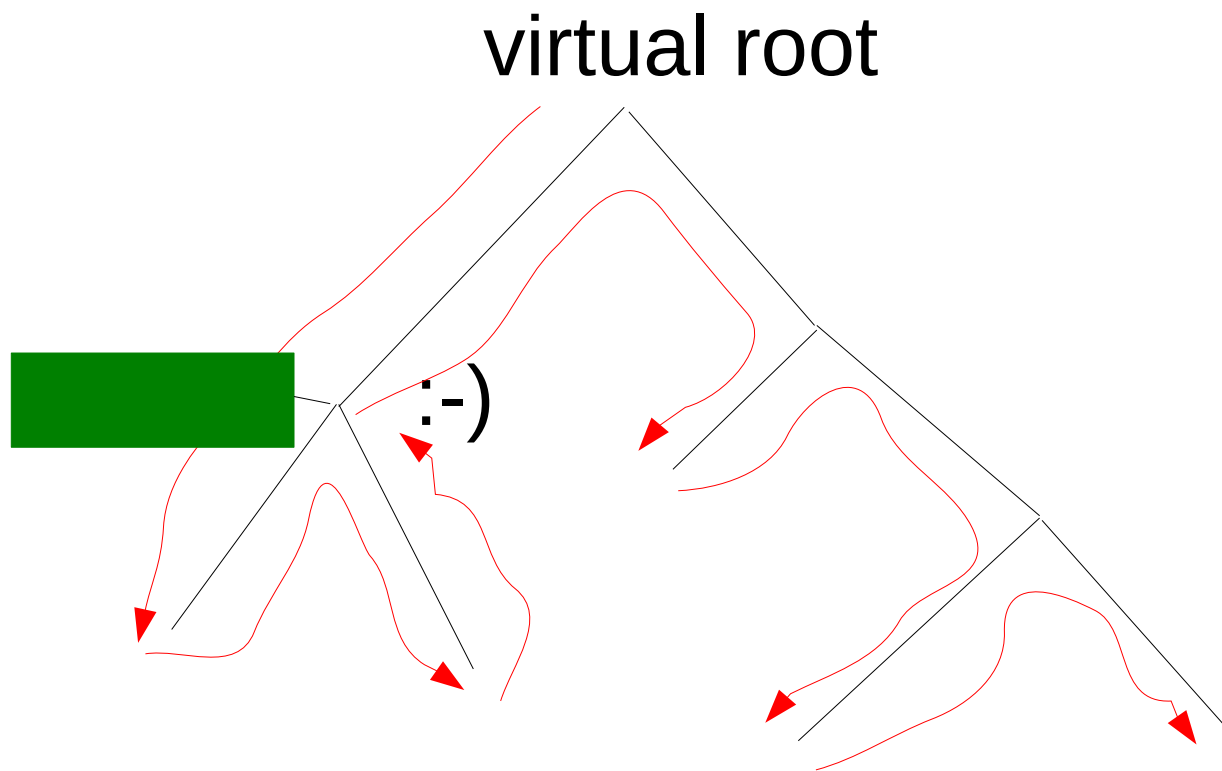
# Post-order Traversal



# Post-order Traversal

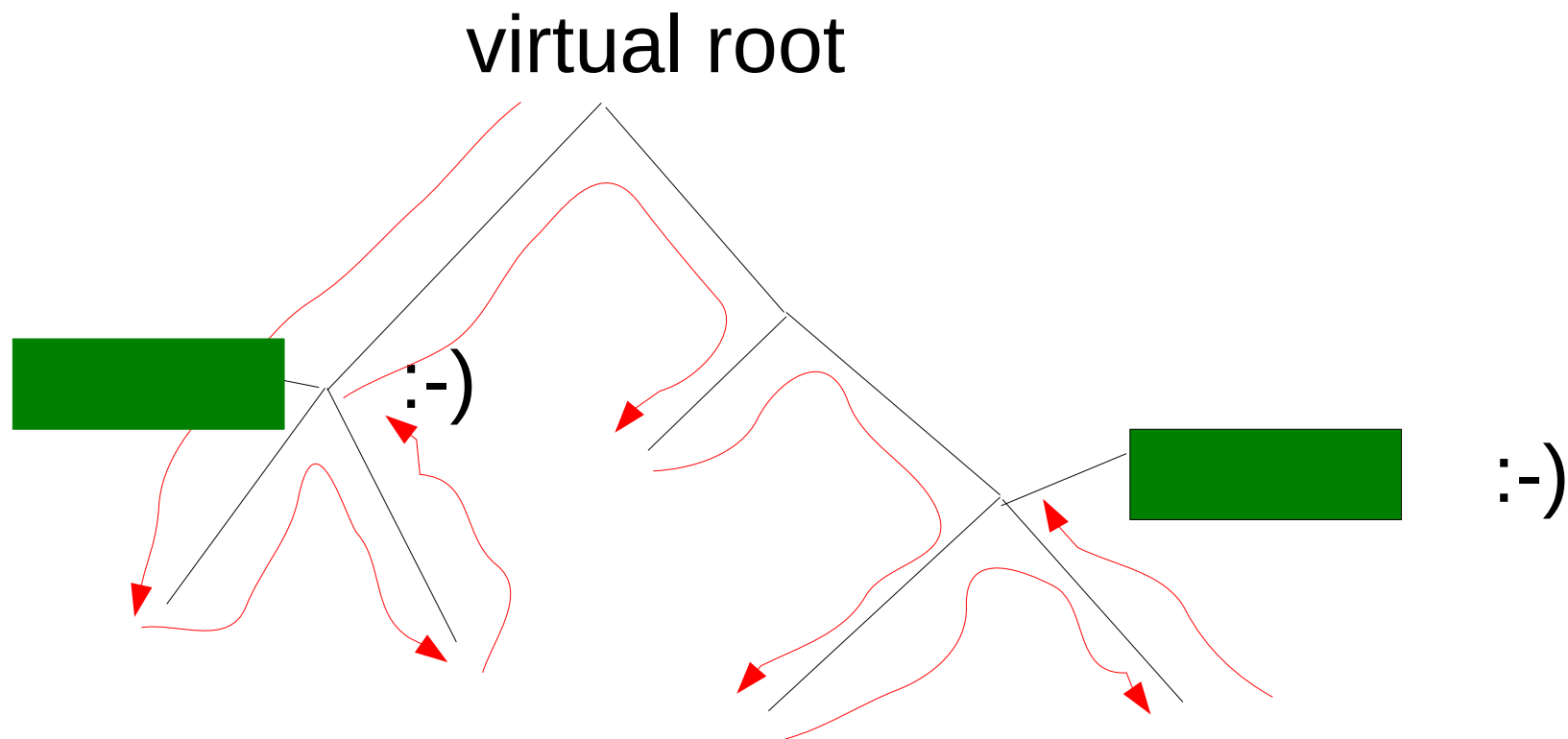


# Post-order Traversal

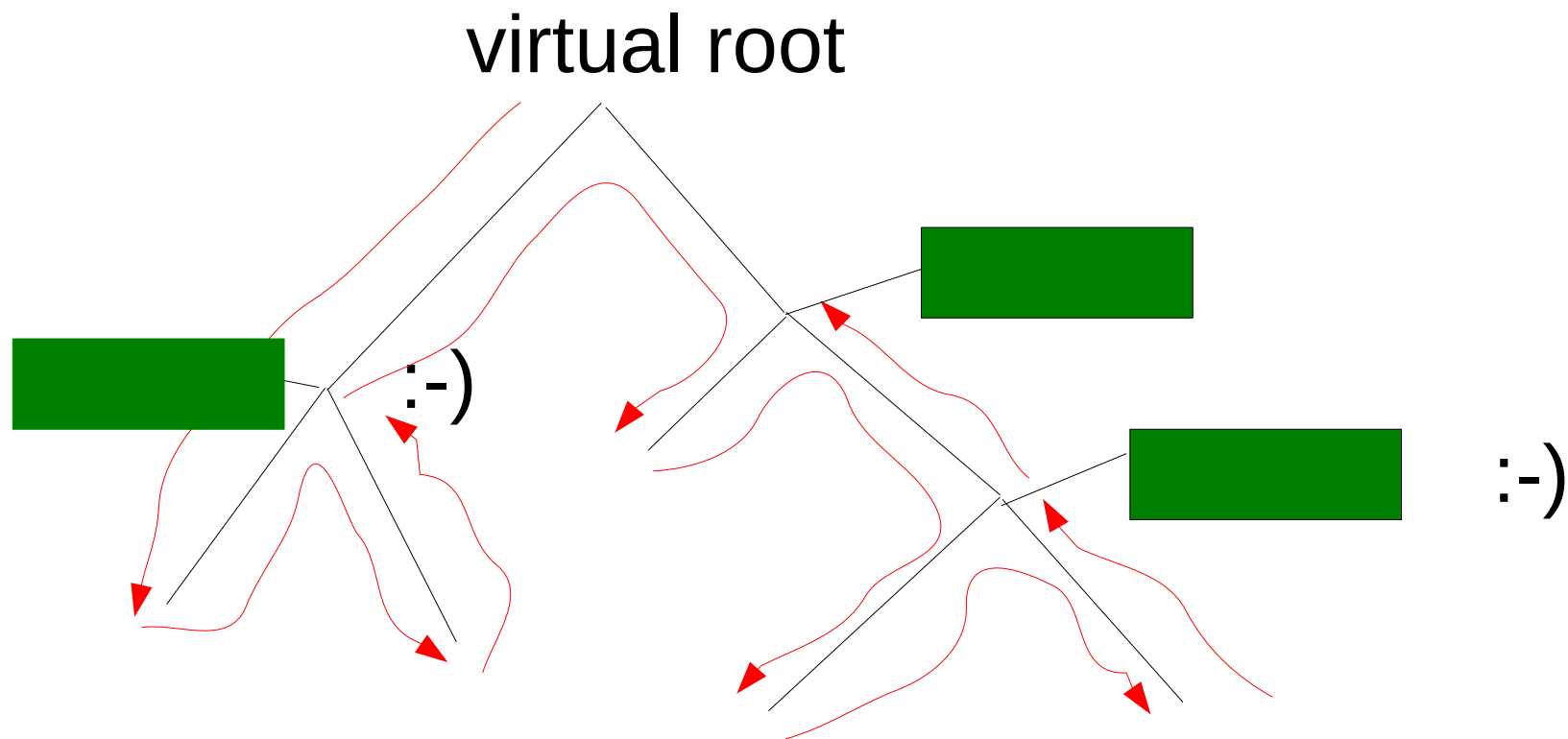




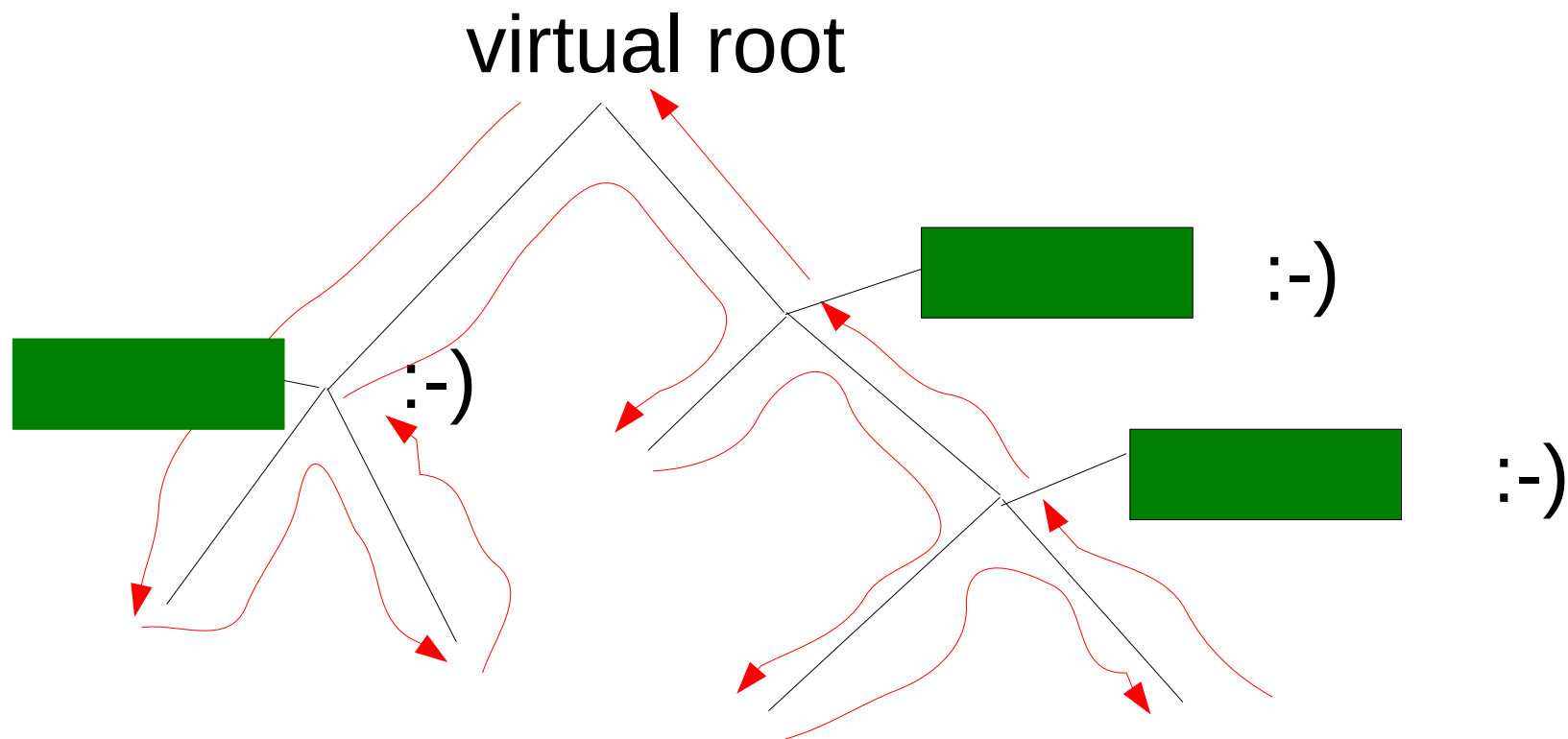
# Post-order Traversal



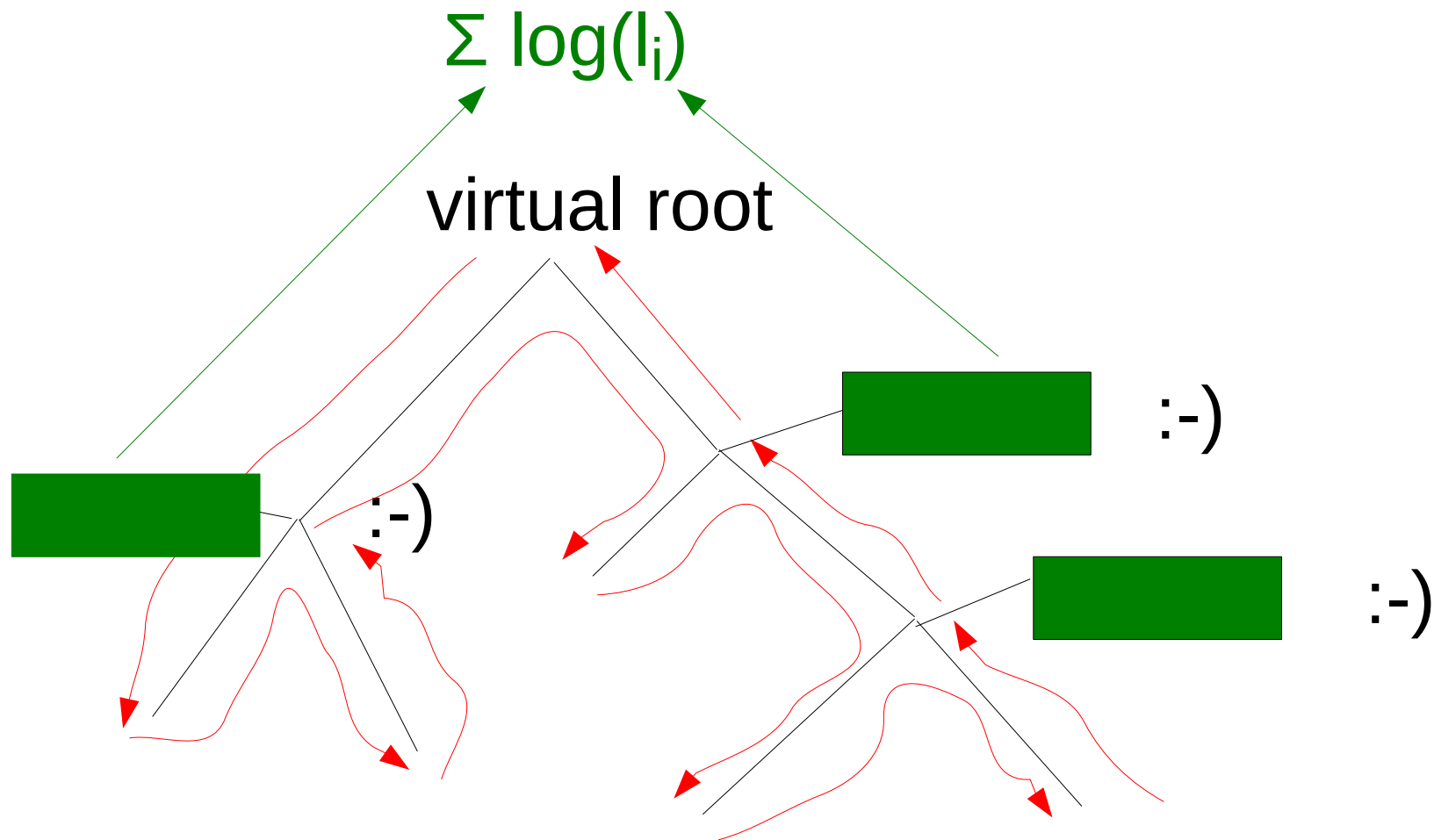
# Post-order Traversal



# Post-order Traversal



# Post-order Traversal



# Basic Operations

## Maximum Likelihood

- Compute Conditional Likelihood Array at an inner node
- Compute Likelihood at Virtual Root
- Optimize a Branch Length for a given Branch
- Optimize all Branch Lengths
- Optimize other Model Parameters

# Basic Operations

## Maximum Likelihood

- **Compute** Conditional Likelihood Array at an inner node
- **Compute** Likelihood at Virtual Root
- **Optimize** a Branch Length for a given Branch
- **Optimize** all Branch Lengths
- **Optimize** other Model Parameters



Why are Bayesians  
lazy people?

# Basic Operations

## Maximum Likelihood

- Compute Conditional Likelihood Array at an inner node
- Compute Likelihood at Virtual Root
- Optimize a Branch Length for a given Branch
- Optimize all Branch Lengths
- Optimize other Model Parameters

Bayesian programs only  
require two operations so it is  
easy

# Basic Operations

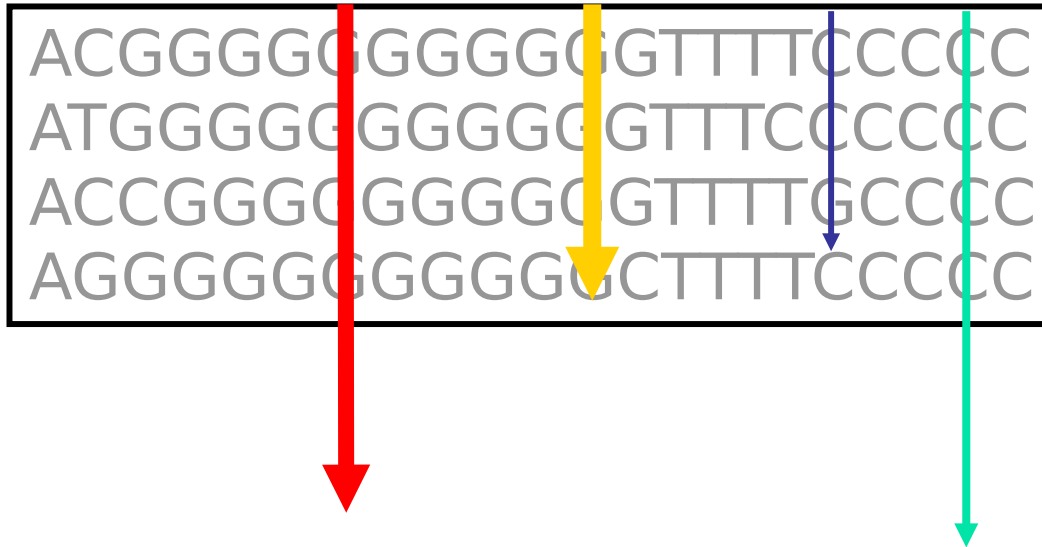
## Maximum Likelihood

- Compute Conditional Likelihood Array at an inner node
- Compute Likelihood at Virtual Root
- Optimize a Branch Length for a given Branch
- Optimize all Branch Lengths
- Optimize other Model Parameters

But they need to design  
efficient proposal mechanisms  
& get the Hastings correction  
right

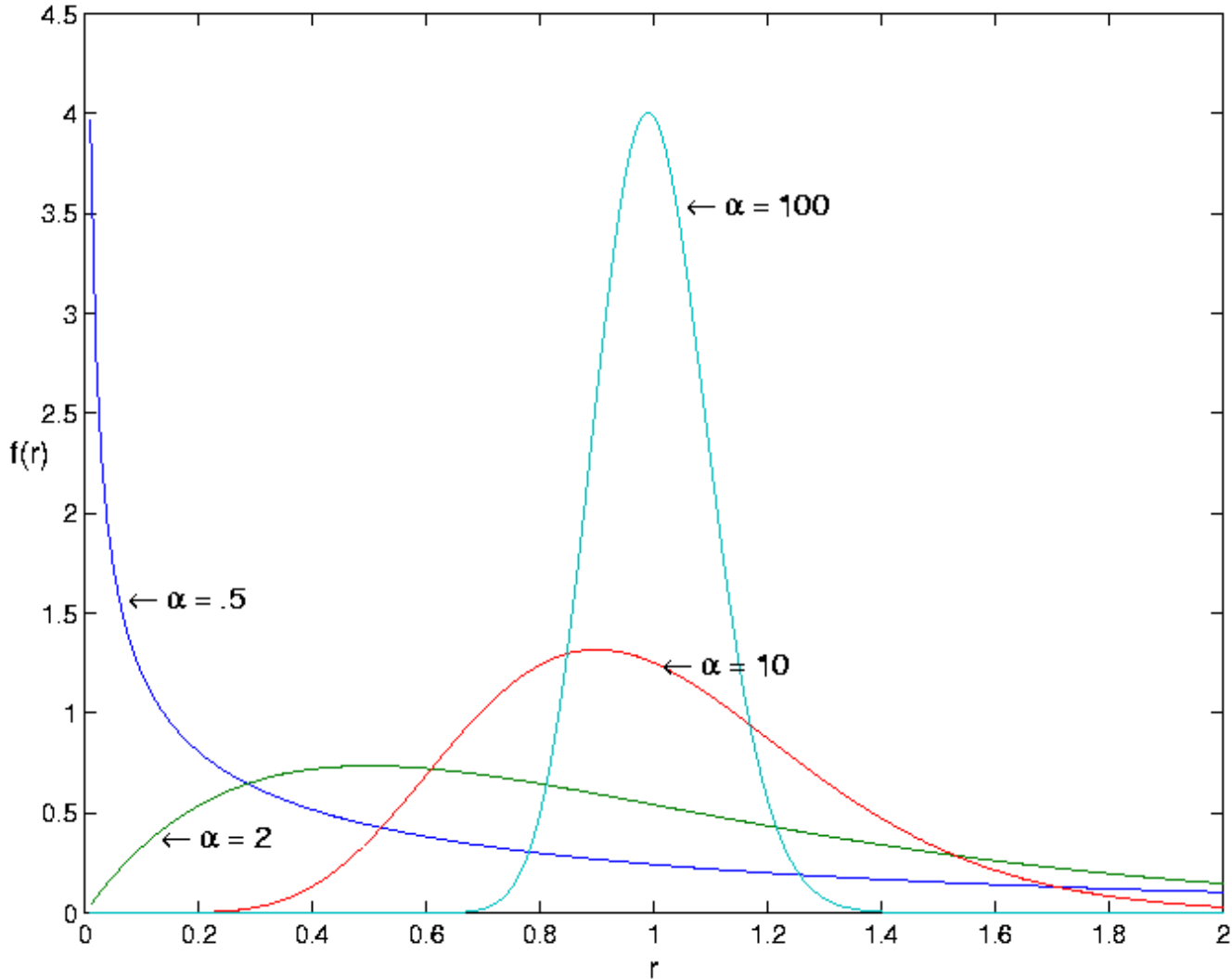


# Rate Heterogeneity among Sites



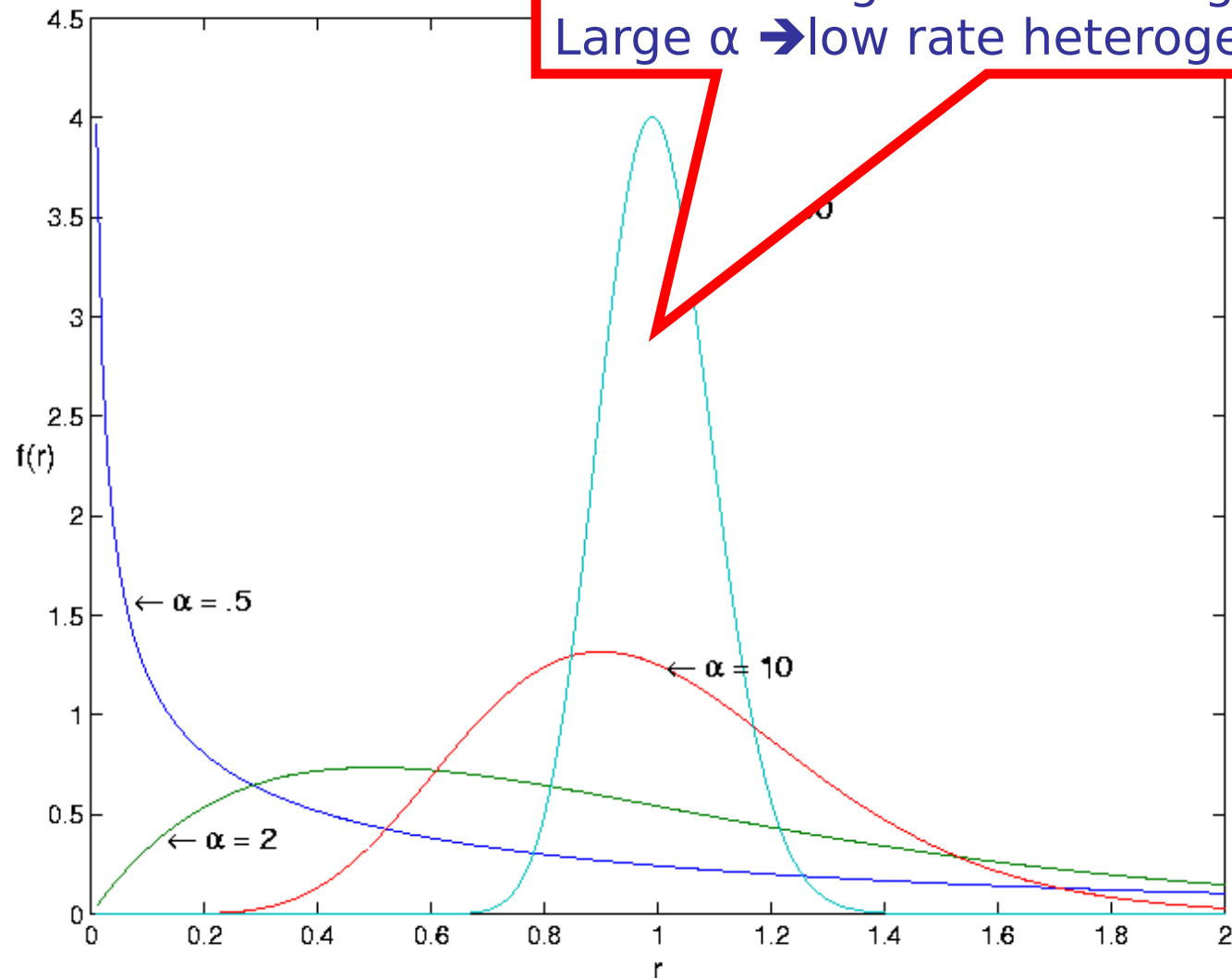
- Biological phenomenon: different sites/columns evolve at distinct speeds
- Need to extend our model

# $\Gamma$ -Distribution

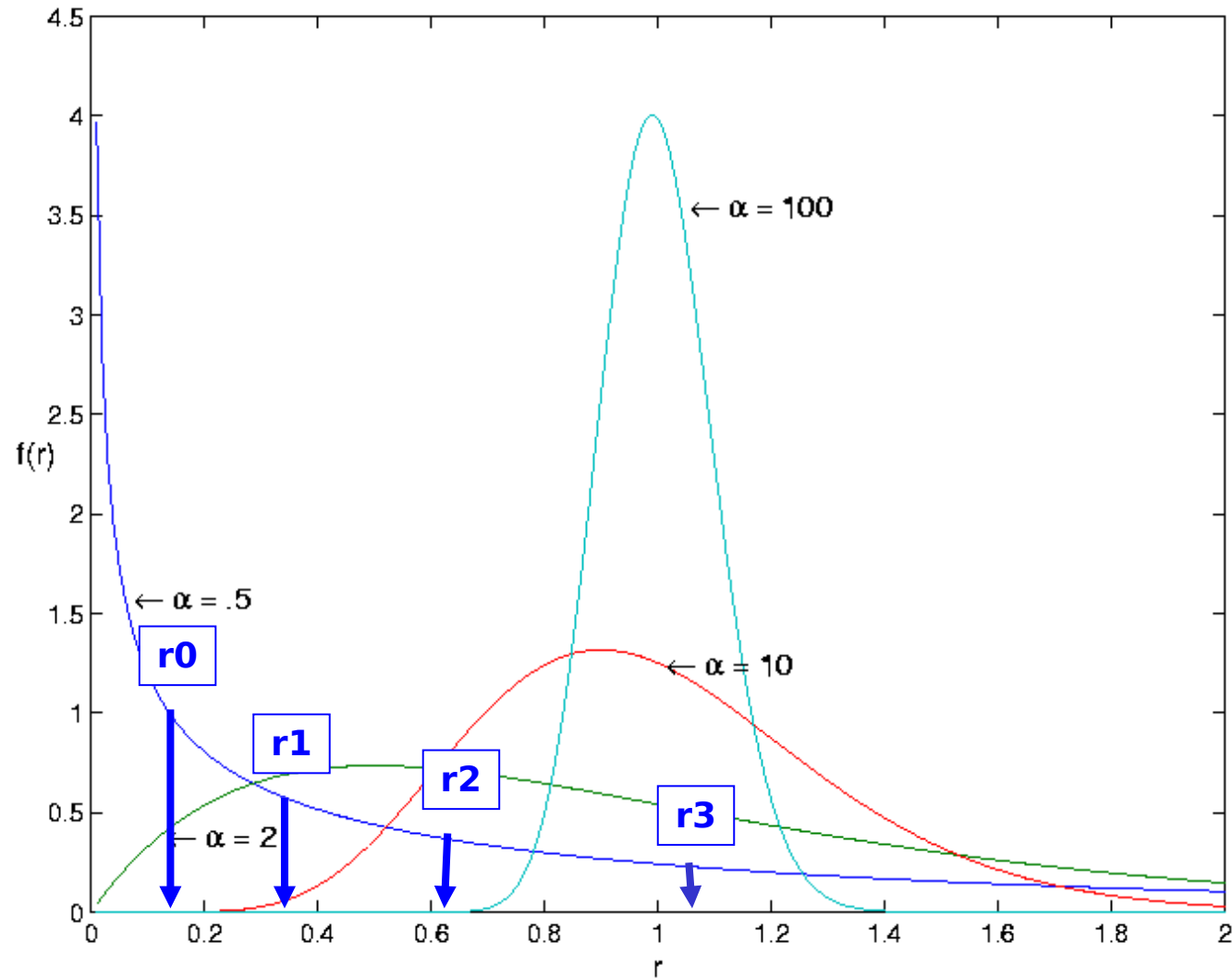


# $\Gamma$ -Distribution

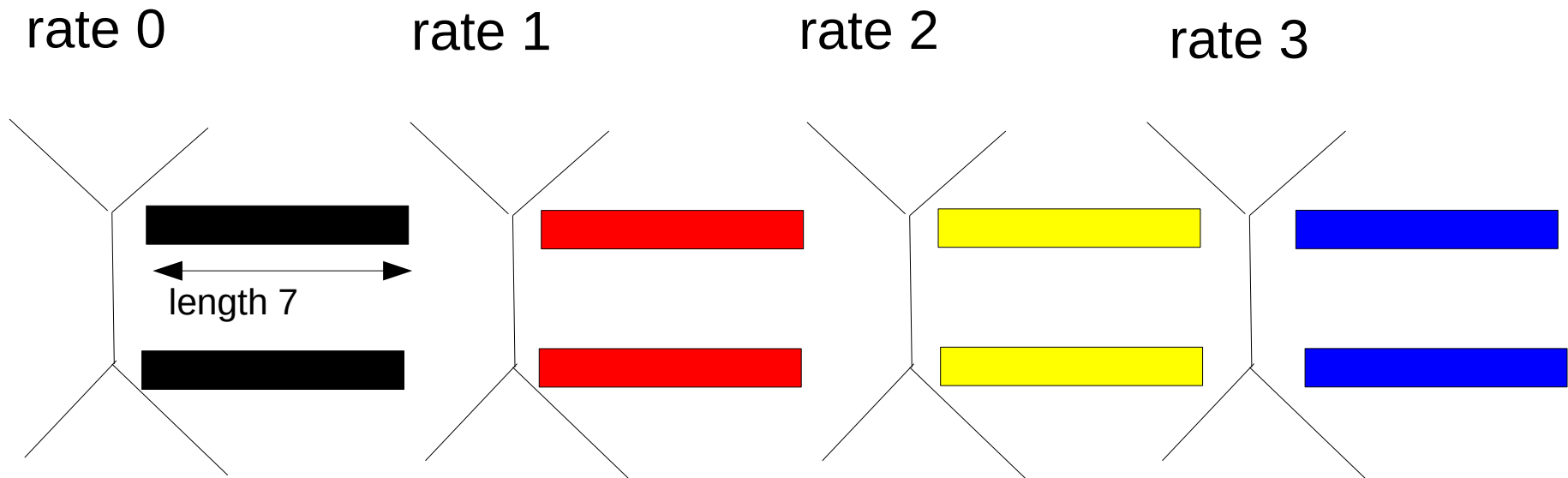
Small  $\alpha \rightarrow$  high rate heterogeneity  
Large  $\alpha \rightarrow$  low rate heterogeneity



# Discrete $\Gamma$ -Distribution



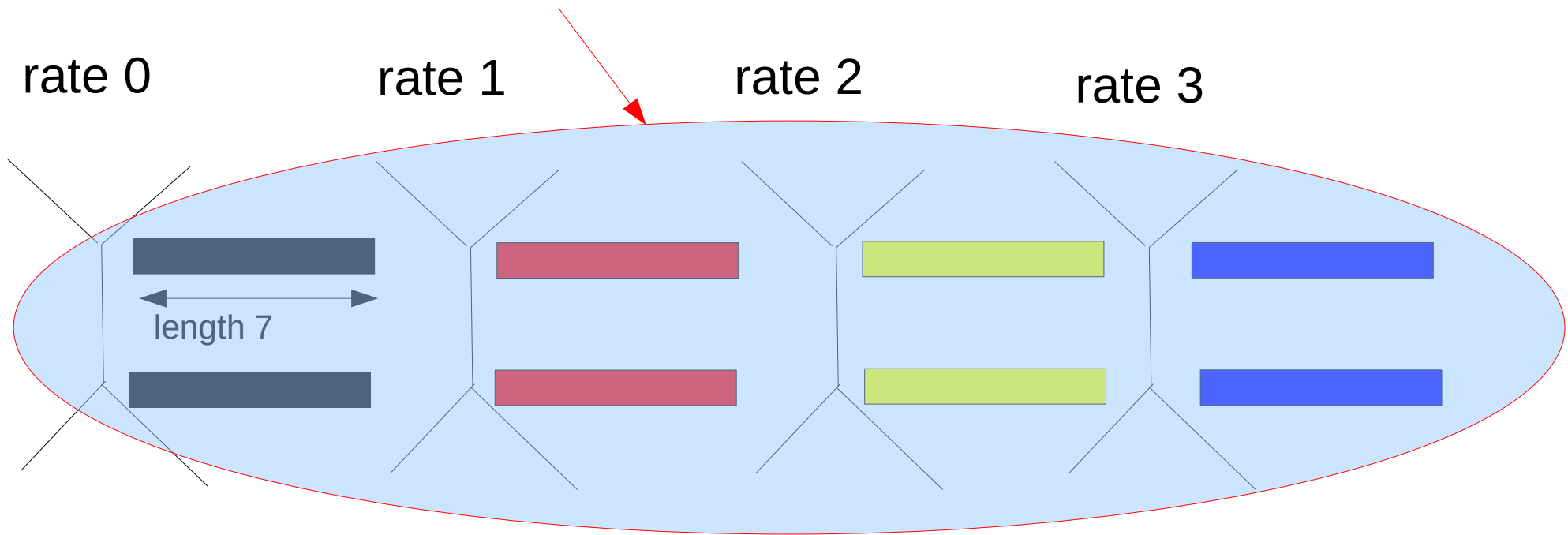
# An Abstract View of $\Gamma$



$$\text{LnL} = \log(L0 * \frac{1}{4}) + \log(L1 * \frac{1}{4}) + \log(L2 * \frac{1}{4}) + \log(L3 * \frac{1}{4})$$

# An Abstract View of $\Gamma$

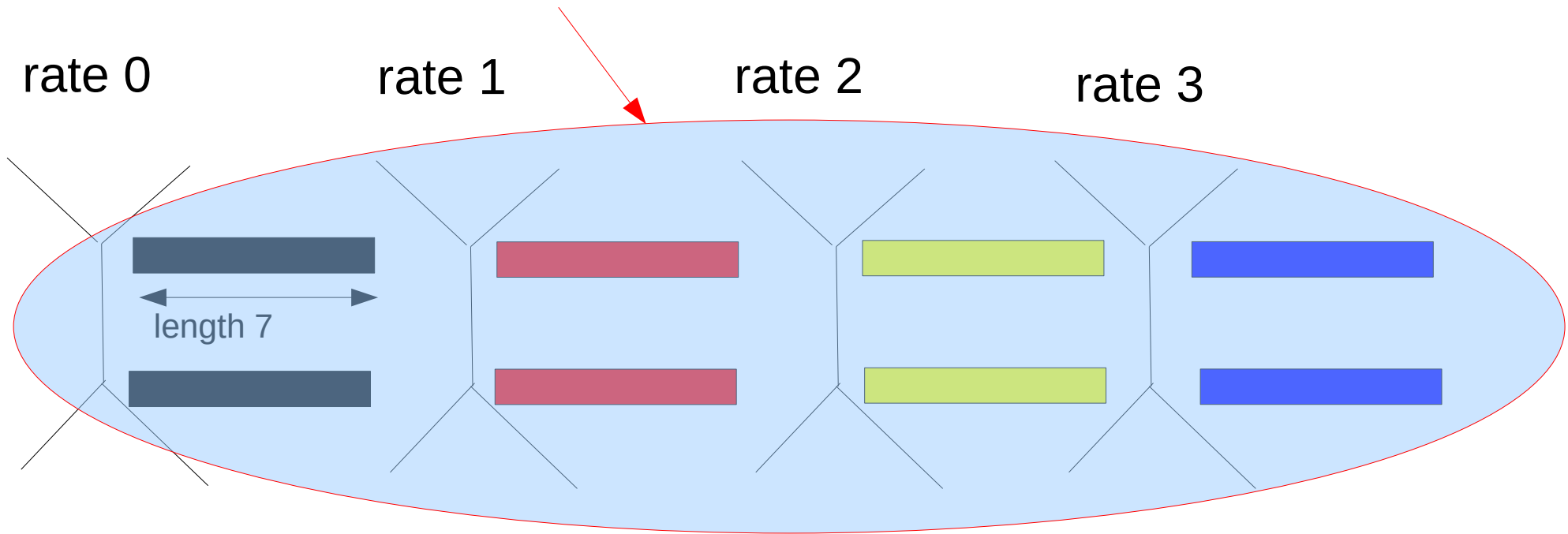
4 times higher memory consumption



$$\text{LnL} = \log(L0 * \frac{1}{4}) + \log(L1 * \frac{1}{4}) + \log(L2 * \frac{1}{4}) + \log(L3 * \frac{1}{4})$$

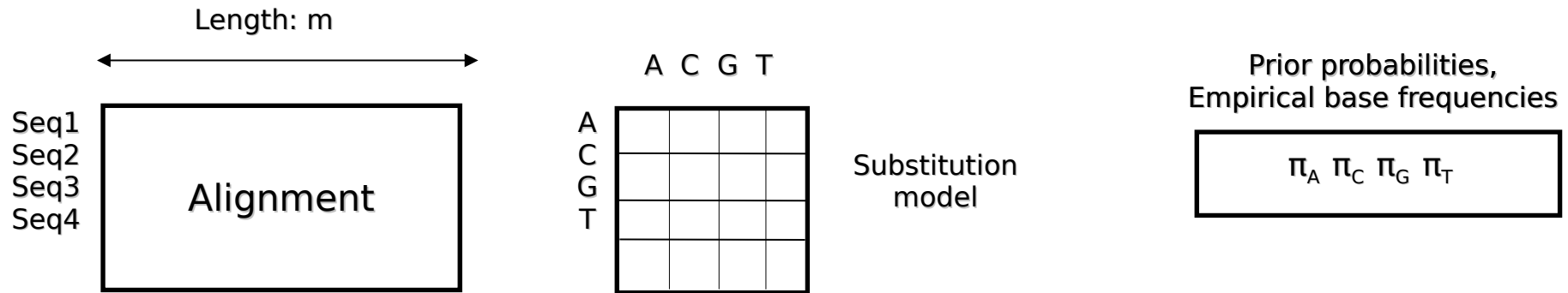
# An Abstract View of $\Gamma$

4 times more FLOPs

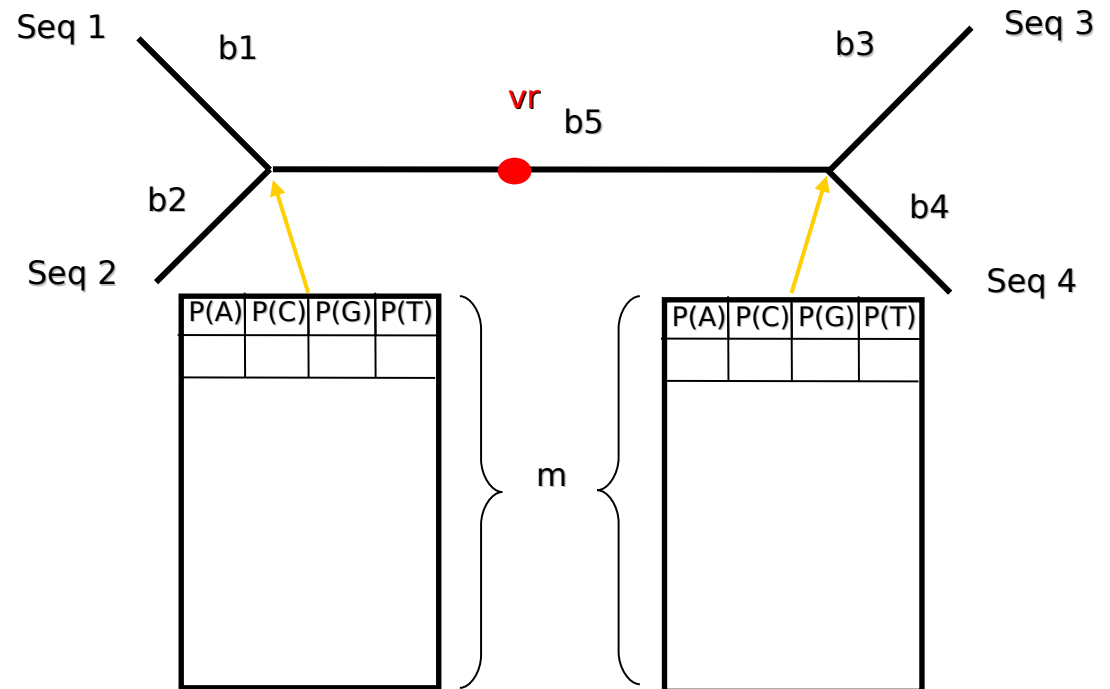


$$\text{LnL} = \log(\text{L0} * \frac{1}{4}) + \log(\text{L1} * \frac{1}{4}) + \log(\text{L2} * \frac{1}{4}) + \log(\text{L3} * \frac{1}{4})$$

# Memory Requirements

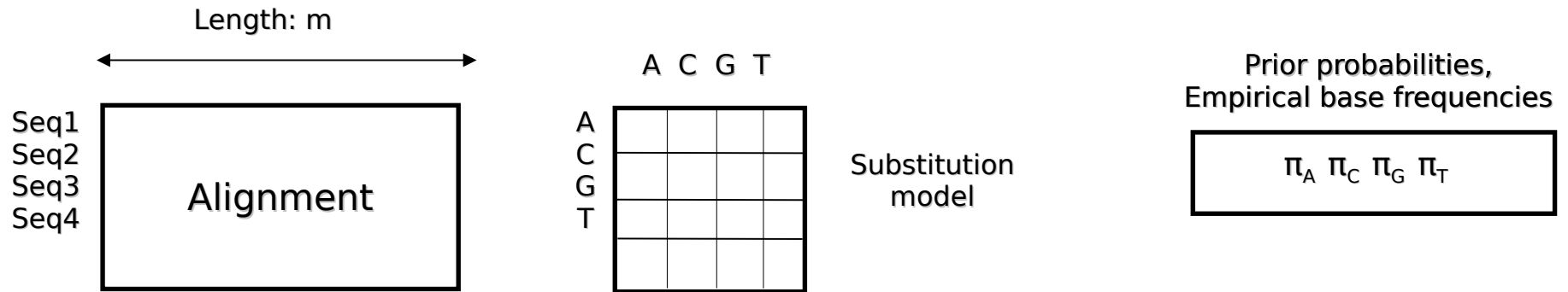


Memory Consumption:  
What's the accumulated size of all conditional likelihood vectors in our tree?





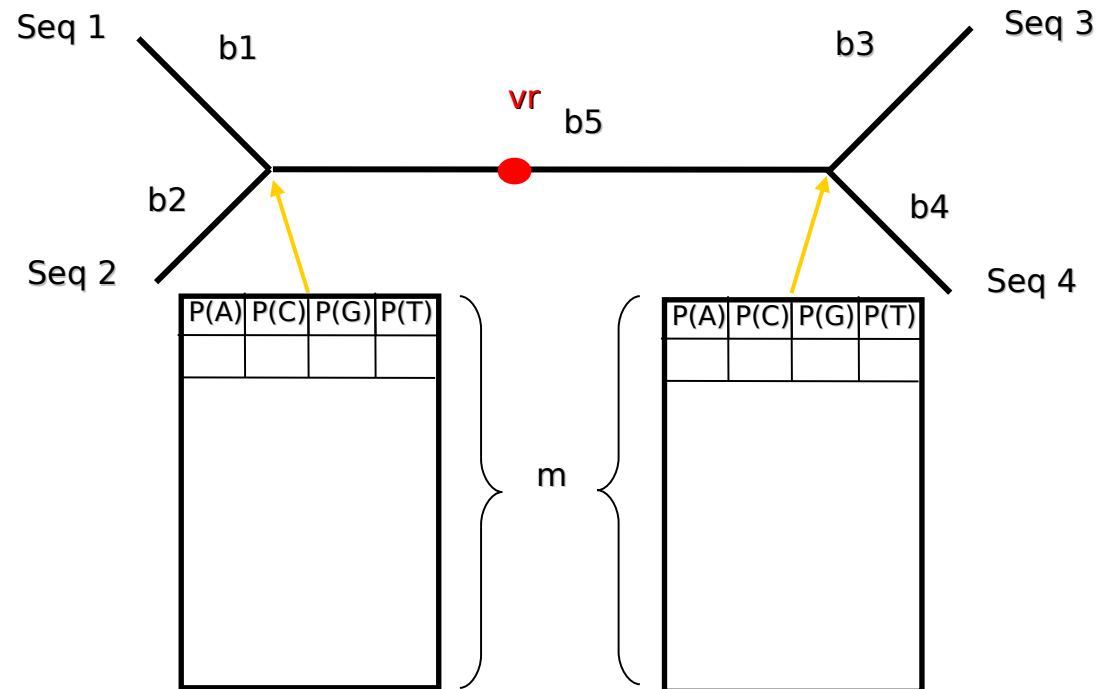
# Memory Requirements



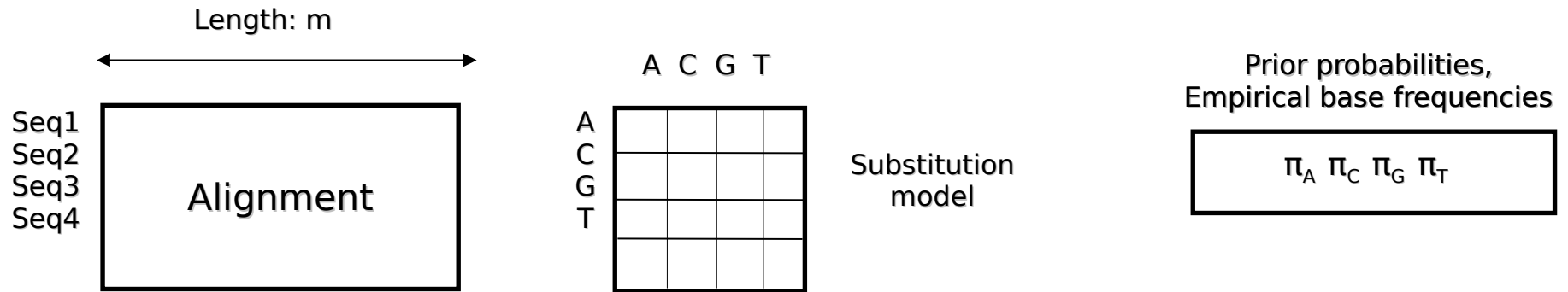
Memory Consumption:  
 $(n-2)$

↑

# of CLVs



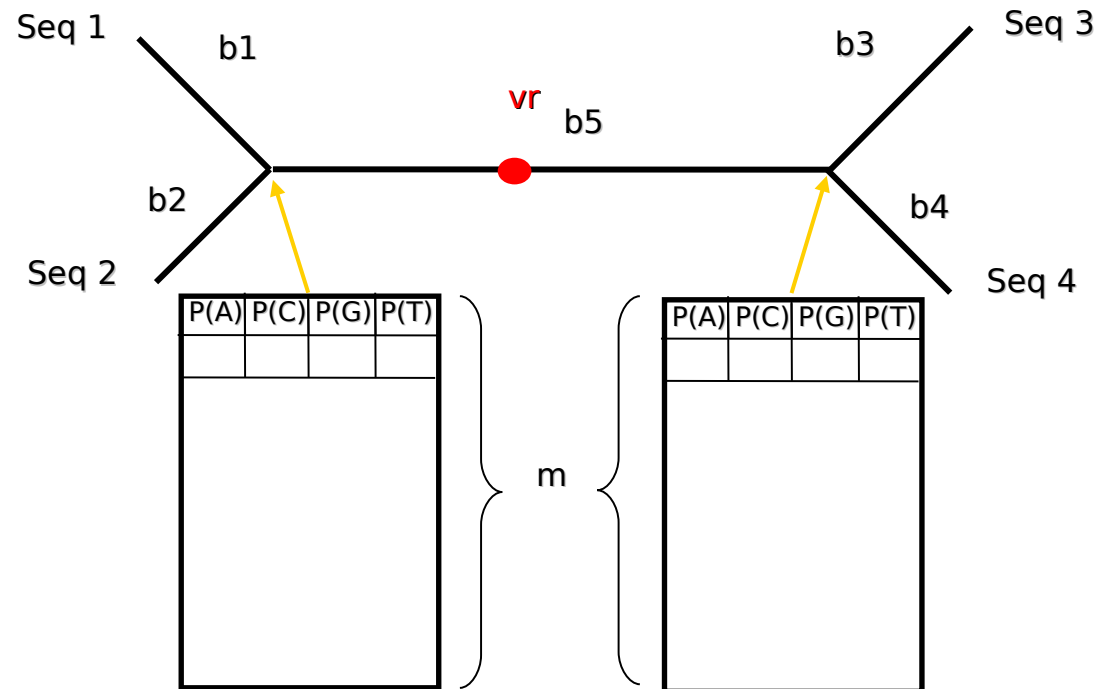
# Memory Requirements



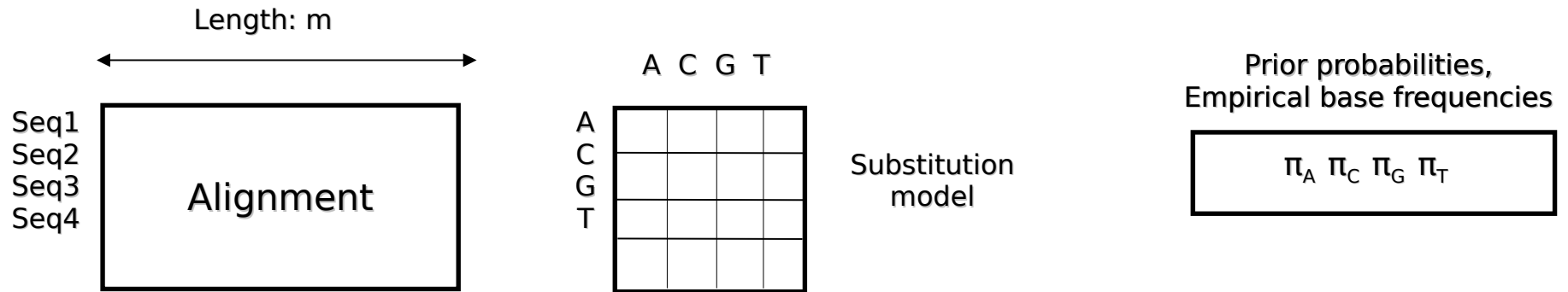
Memory Consumption:  
 $(n-2) * m$

↑

# of entries per CLV

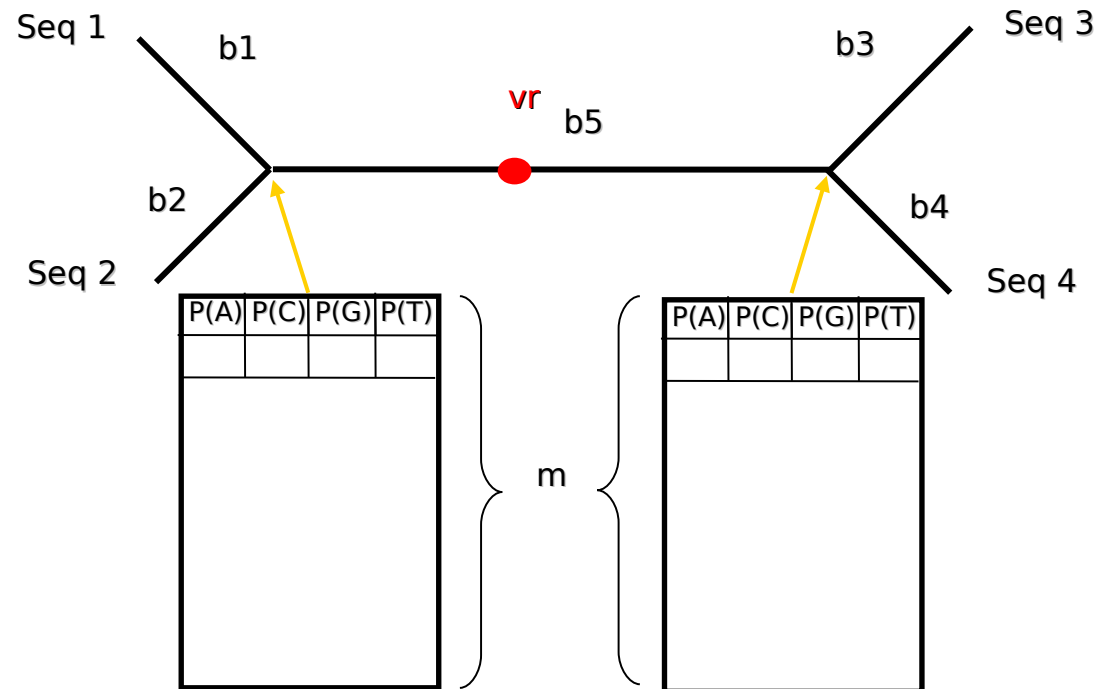


# Memory Requirements

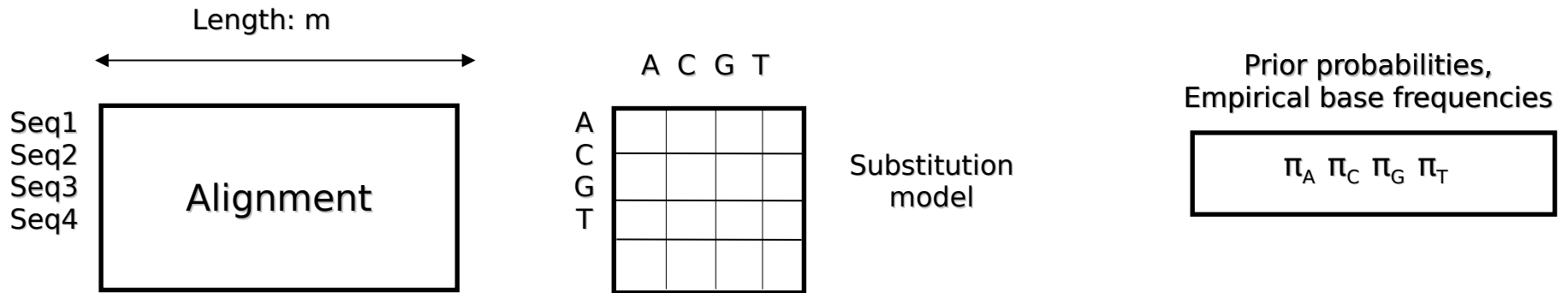


Memory Consumption:  
 $(n-2) * m * 4$

# DNA states

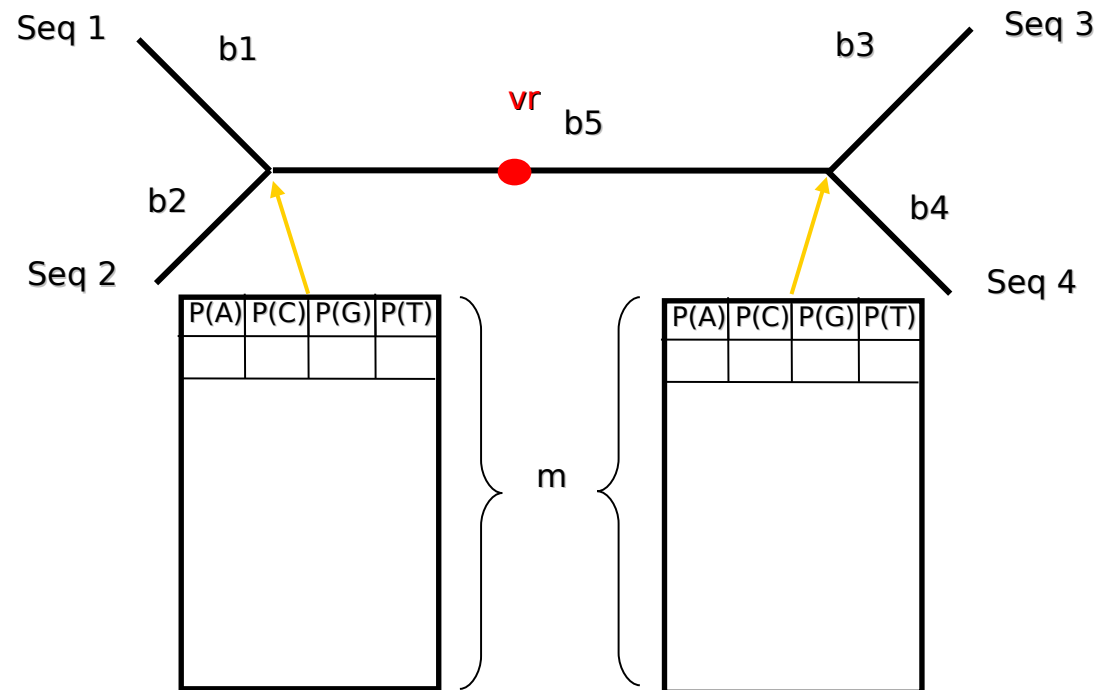


# Memory Requirements

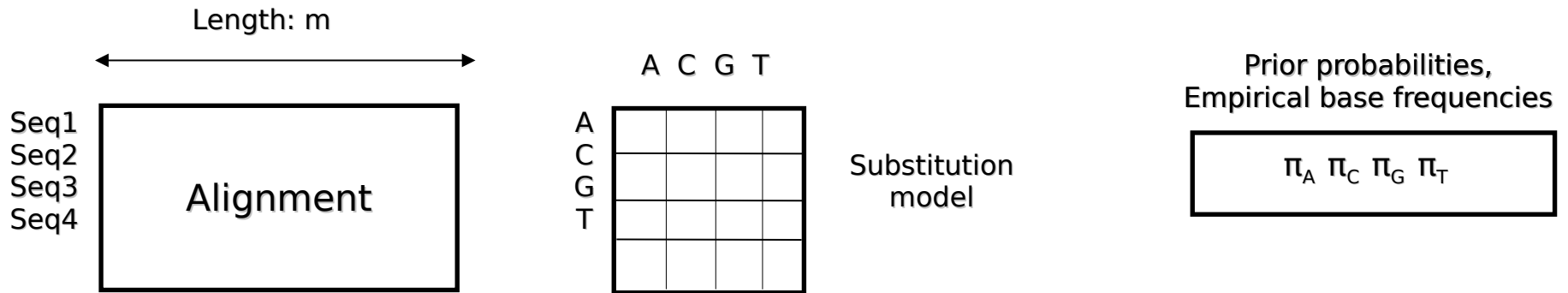


Memory Consumption:  
 $(n-2) * m * 4 * 1$

Rate heterogeneity:  
no rate heterogeneity here

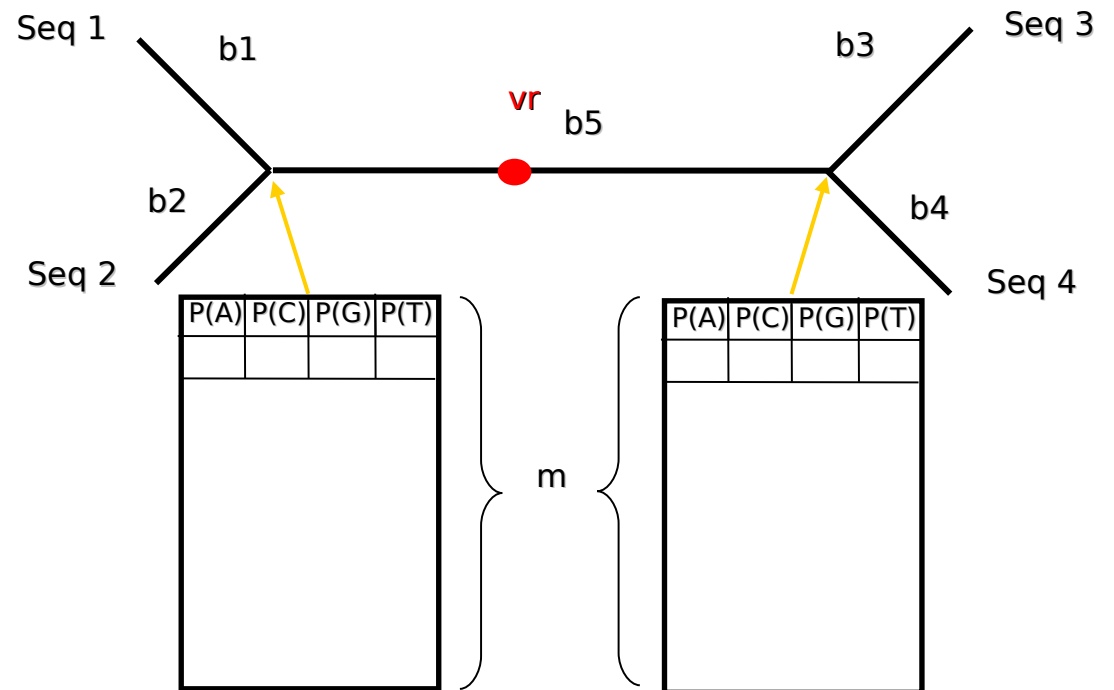


# Memory Requirements



Memory Consumption:  
 $(n-2) * m * 4 * 1 * 8 \text{ bytes}$

double  
precision  
arithmetics



# Memory Requirements for lazy people

Length:  $m$

Seq1  
Seq2  
Seq3  
Seq4

A C G T

A

Prior probabilities, Empirical base frequencies

$\pi_G \pi_T$

3 Seq 3

b4 Seq 4

P(T)

Memory Consumption:  $(n-2) * m * 4 * 8$  bytes

### RAxML memory requirements

Since datasets are getting larger here is a formula to estimate RAxML memory requirements: Given an alignment of  $n$  taxa and  $m$  distinct patterns the memory consumption is approximately:

- $MEM(AA+GAMMA) = (n-2) * m * (80 * 8)$  bytes
- $MEM(AA+CAT) = (n-2) * m * (20 * 8)$  bytes
- $MEM(DNA+GAMMA) = (n-2) * m * (16 * 8)$  bytes
- $MEM(DNA+CAT) = (n-2) * m * (4 * 8)$  bytes

To convert bytes to MB or GB you can use this [on-line converter](#)

You may also use the on-line calculator below:

taxa (n):  pattern (m):

DNA+GAMMA calc size!

Required size:

$(n-2) * m * (x * 8)$  bytes = MEM

# Phylogenetics: Memory Challenge

- Memory Footprints are becoming huge
  - **2011: 190GB** (Ziheng Yang)
  - **2014: 1TB**  $\approx$  140 insect transcriptomes

"Whole-genome analyses resolve early branches in the tree of life of modern birds". *Science*, 46(6215):1320-1331, 2014
  - **2017: 7-8 TB** for  $\approx$  1600 insect transcriptomes
  - **2019: 9TB** for 350 bird genomes and **500,000** core hours for just computing 1 single ML tree

# Phylogenetics: Memory Challenge

- Solutions
  - Algorithmic means & data structures
  - Supercomputers
  - Just don't infer trees on such supermatrices
    - Use gene tree ↔ species tree reconciliation methods?
      - e.g., `ASTRAL` or ML tools developed in my lab (`GeneRax` & `SpeciesRax`)
      - but inference of gene trees has higher difficulty → gene tree uncertainty
    - Filter out relevant sites from alignment beforehand?

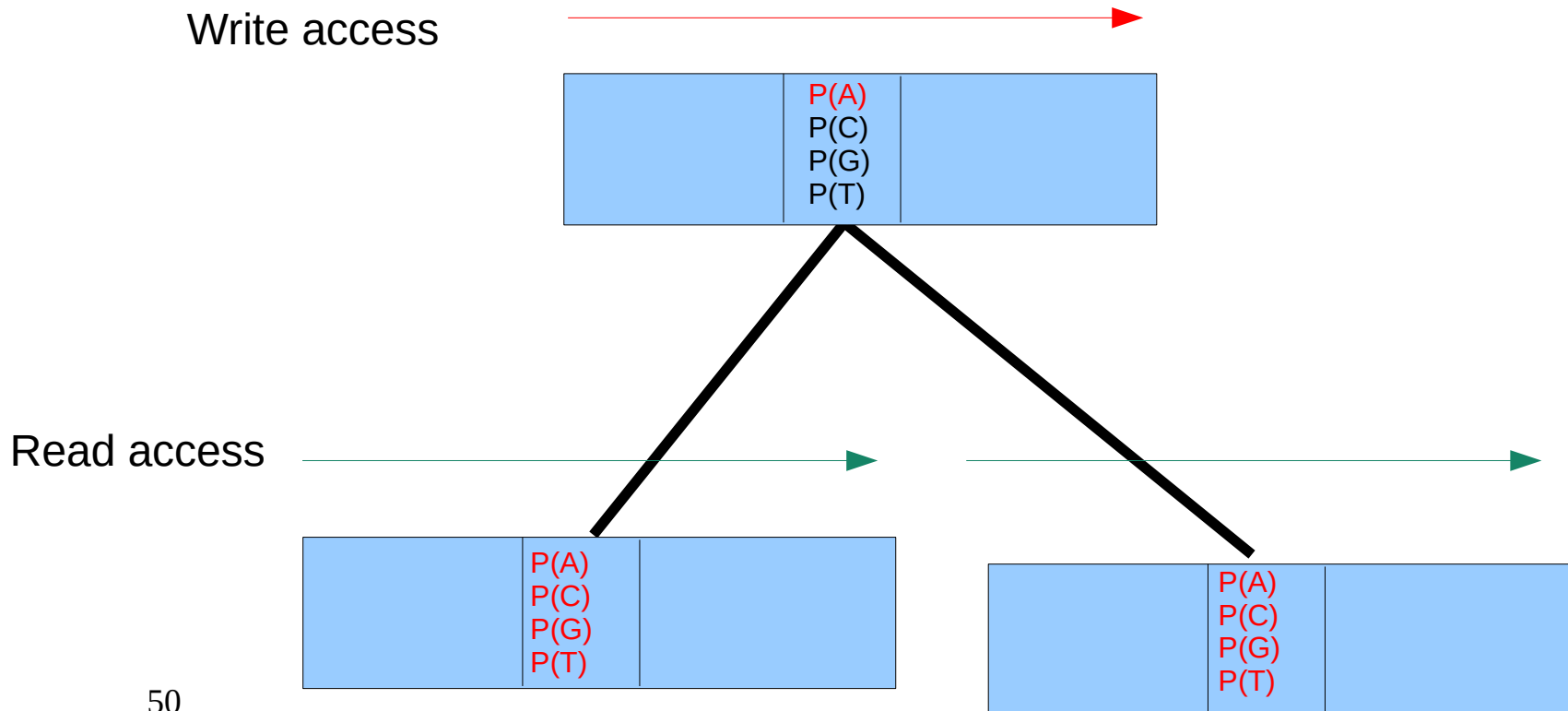


# Outline

- Maximum Likelihood (Recap)
- **Sequential Optimization**
- Parallelization
- Parallel I/O
- Numerical Nightmares
- Energy Efficiency

# Use Vector Instructions

- 128 and 256 bit vector instructions worked well
  - Part of production level tools
- 512 bit vector instructions → not so well (**zero speedup**)
  - Likelihood calculations are memory bandwidth bound
  - We are moving along linearly among three conditional likelihood vectors
  - but don't do so many computations per vector entry



# An Example

- We can also use vector instructions for parsimony calculations
- My open source parsimony code <https://github.com/stamatak/Parsimonator-1.0.2>
- On my laptop

```
./parsimonator -p 12345 -s 125.phy -n X1
```

```
Parsimony tree [0] with length 193639 computed in 3.074347 seconds
```

```
./parsimonator-SSE3 -p 12345 -s 125.phy -n X2
```

```
Parsimony tree [0] with length 193639 computed in 1.576415 seconds
```

```
./parsimonator-AVX -p 12345 -s 125.phy -n X3
```

```
Parsimony tree [0] with length 193639 computed in 1.312245 seconds
```

# An Example

- We can also use vector instructions for parsimony calculations
- My open source parsimony code <https://github.com/stamatak/Parsimonator-1.0.2>
- On my laptop

```
./parsimonator -p 12345 -s 125.phy -n X1
```

```
Parsimony tree [0] with length 193639 computed in 3.074347 seconds
```

```
./parsimonator-SSE3 -p 12345 -s 125.phy -n X2
```

```
Parsimony tree [0] with length 193639 computed in 1.576415 seconds
```

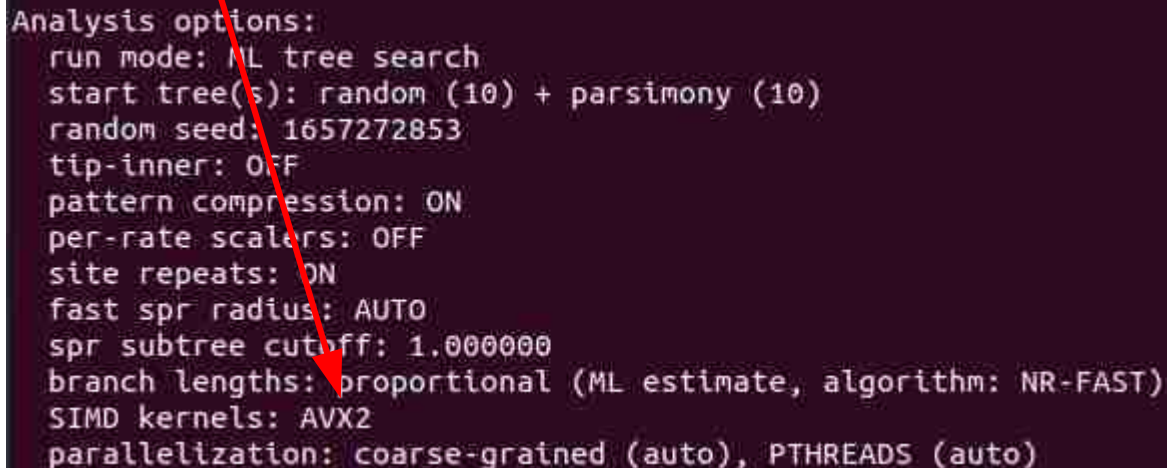
```
./parsimonator-AVX -p 12345 -s 125.phy -n X3
```

```
Parsimony tree [0] with length 193639 computed in 1.312245 seconds
```

Why don't we get a good speedup for AVX (256 bit vectors)?

# User friendly Vector Instructions: RAxML-NG

- Will automatically chose the best available vector instruction set

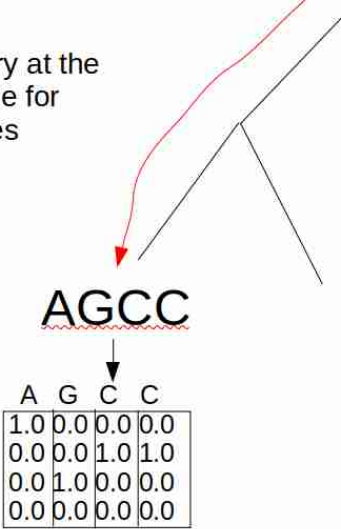


```
Analysis options:  
run mode: ML tree search  
start tree(s): random (10) + parsimony (10)  
random seed: 1657272853  
tip-inner: OFF  
pattern compression: ON  
per-rate scalars: OFF  
site repeats: ON  
fast spr radius: AUTO  
spr subtree cutoff: 1.000000  
branch lengths: proportional (ML estimate, algorithm: NR-FAST)  
SIMD kernels: AVX2  
parallelization: coarse-grained (auto), PTHREADS (auto)
```

# Optimizations we have already seen

## Tip vector lookup

We can save memory at the tips via a lookup table for these constant values

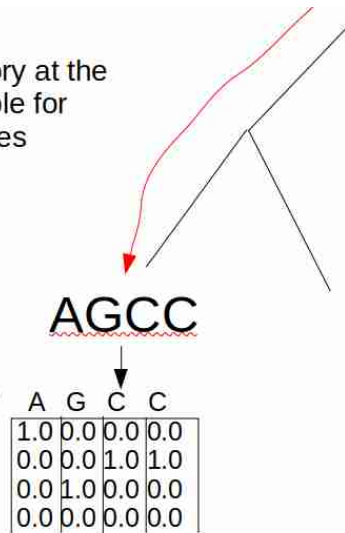


17

# Optimizations we have already seen

## Tip vector lookup

We can save memory at the tips via a lookup table for these constant values

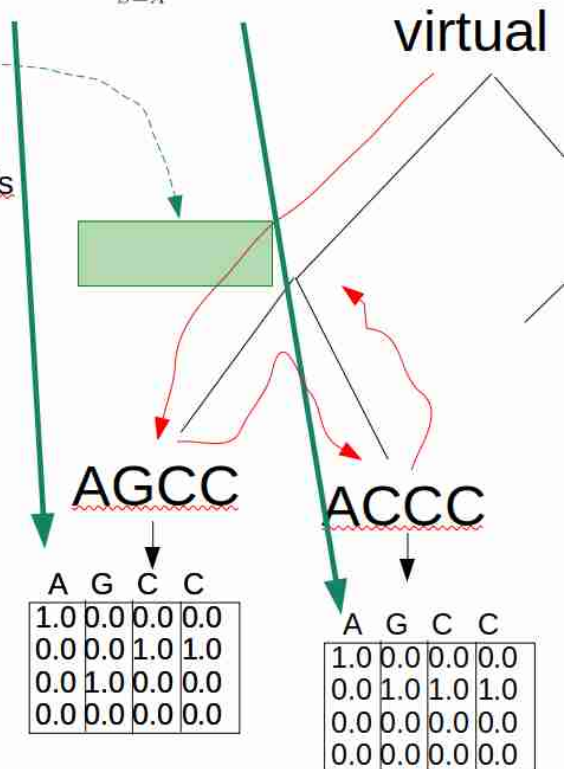


17

## Optimizations for special cases

$$\vec{L}_A^{(k)}(c) = \left( \sum_{S=A}^T P_{AS}(b_i) \vec{L}_S^{(i)}(c) \right) \left( \sum_{S=A}^T P_{AS}(b_j) \vec{L}_S^{(j)}(c) \right)$$

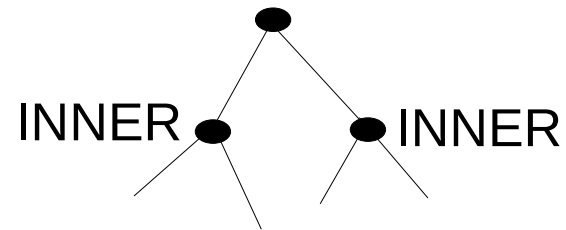
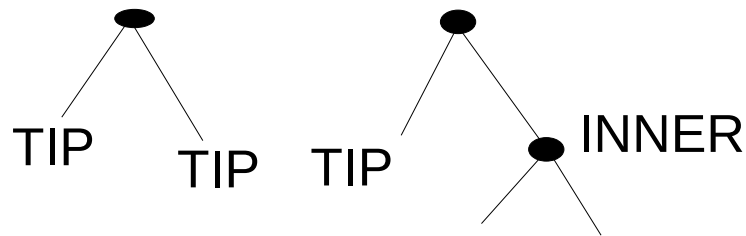
Simplify equation as most summands are 0!



21

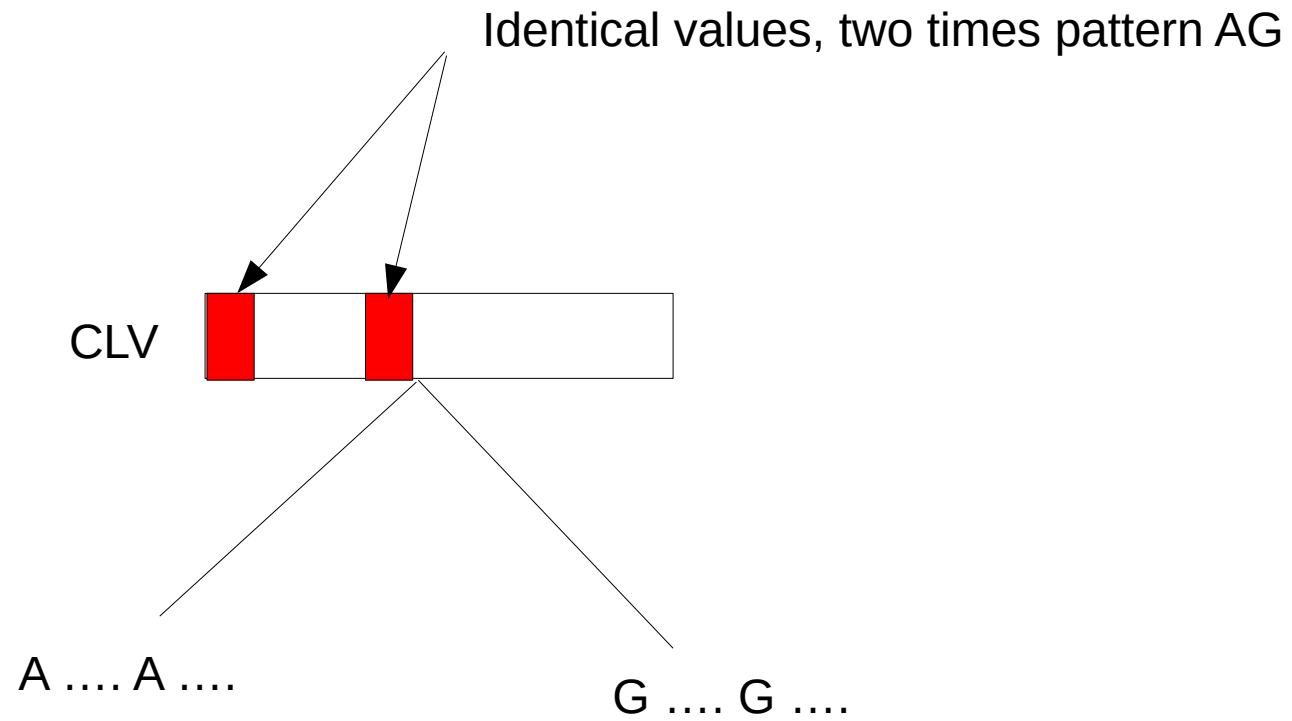
# Standard Optimizations

- Dedicated implementations for computing CLVs & Tip Vector lookups
- To be found in all modern tools: RAxML-NG, IQ-Tree, MrBayes, etc. etc.



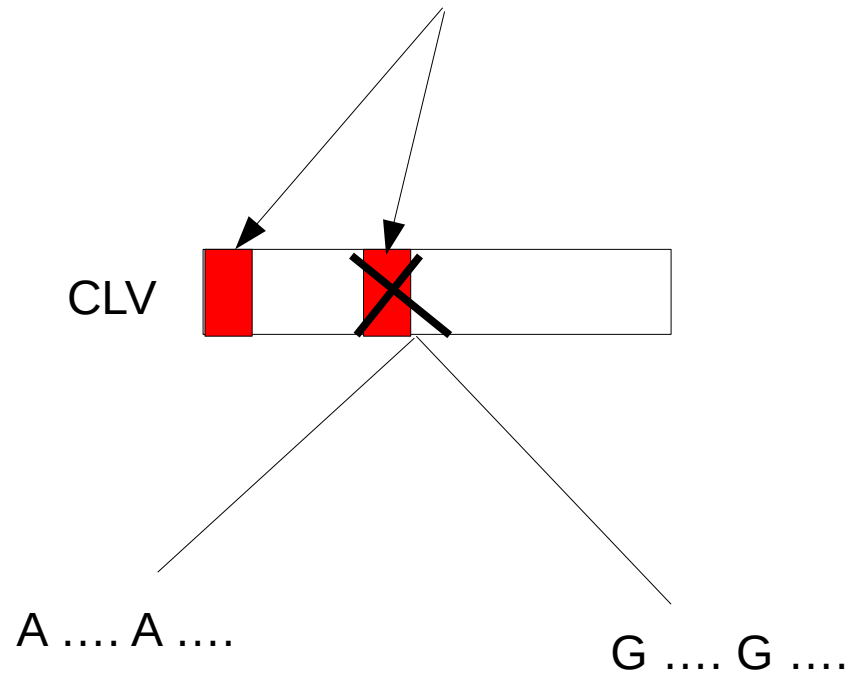


# Repeating Patterns



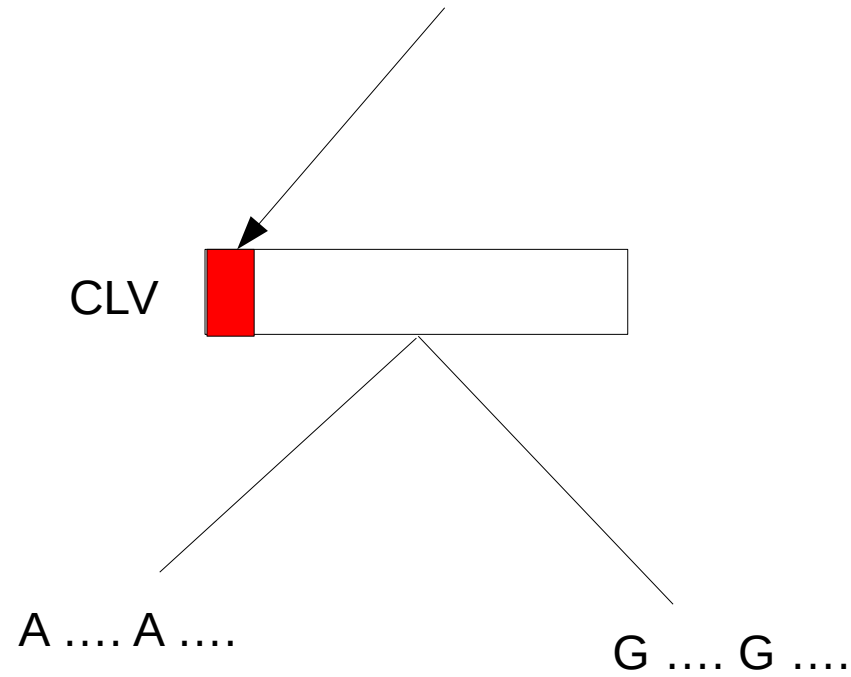
# Repeating Patterns

Detect identical patterns and omit second computation



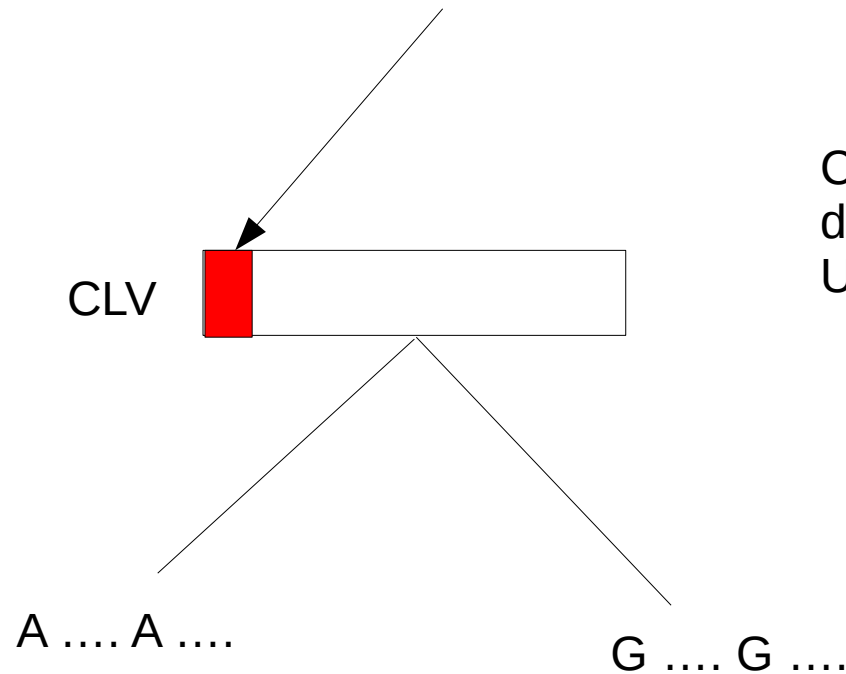
# Repeating Patterns

Also, shorten CLV → less memory required



# Repeating Patterns

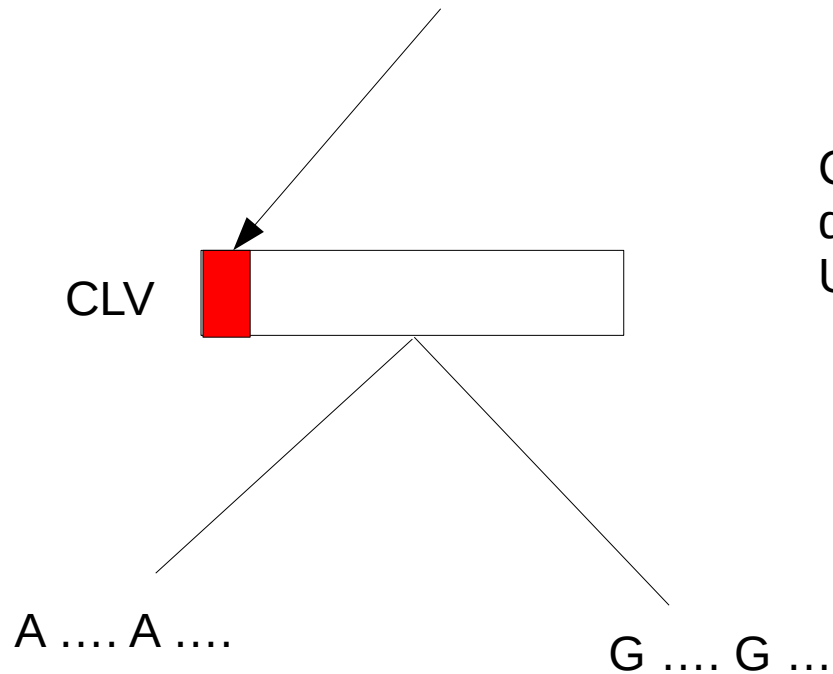
Also, shorten CLV → less memory required



Challenge: Efficient data structure to detect & store repeats  
Up to 10-fold run-time improvements

# Repeating Patterns

Also, shorten CLV → less memory required



Challenge: Efficient data structure to detect & store repeats  
Up to 10-fold run-time improvements

RAxML-NG

```
Analysis options:  
run mode: ML tree search  
start tree(s): random (10) + parsimony (10)  
random seed: 1657272853  
tip-inner: OFF  
pattern compression: ON  
per-rate scalars: OFF  
site repeats: ON  
fast spr radius: AUTO  
spr subtree cutoff: 1.000000  
branch lengths: proportional (ML estimate, algorithm: NR-FAST)  
SIMD kernels: AVX2  
parallelization: coarse-grained (auto), PTHREADS (auto)
```

# Repeating Patterns

- Implemented in RAxML-NG production code
- **But**, totally messes up parallelization
  - highly variable times to compute per-site likelihoods

JOURNAL ARTICLE

## Efficient Detection of Repeating Sites to Accelerate Phylogenetic Likelihood Calculations

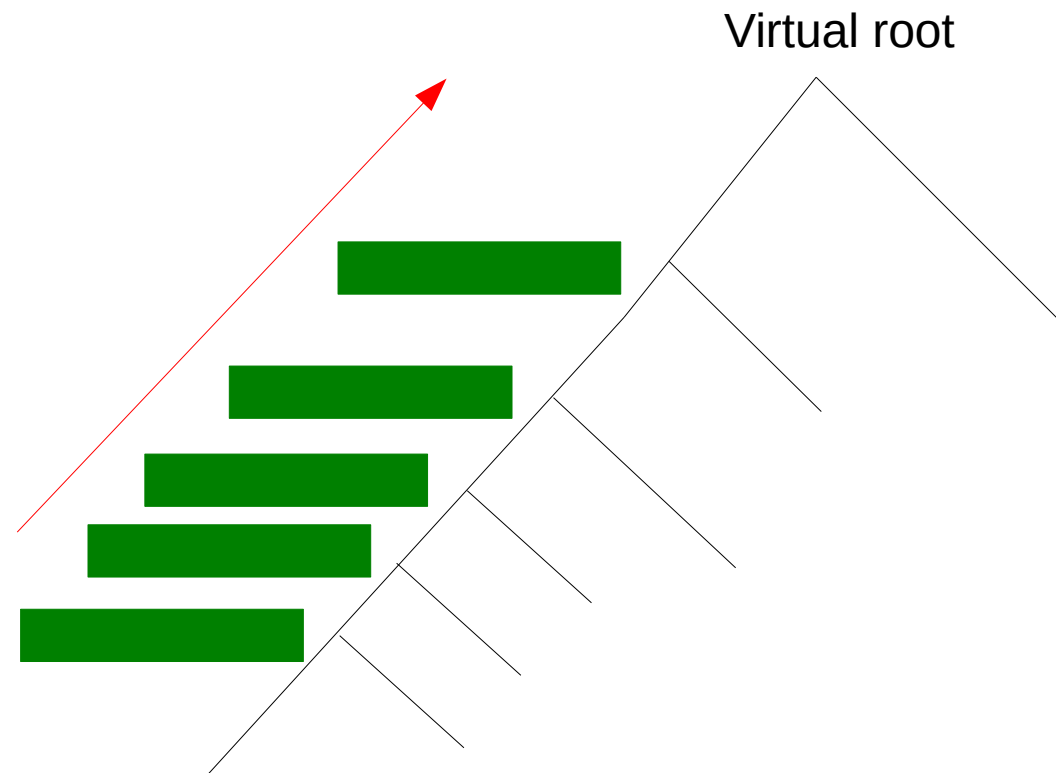
K. Kobert, A. Stamatakis, T. Flouri  [Author Notes](#)

*Systematic Biology*, Volume 66, Issue 2, March 2017, Pages 205–217,

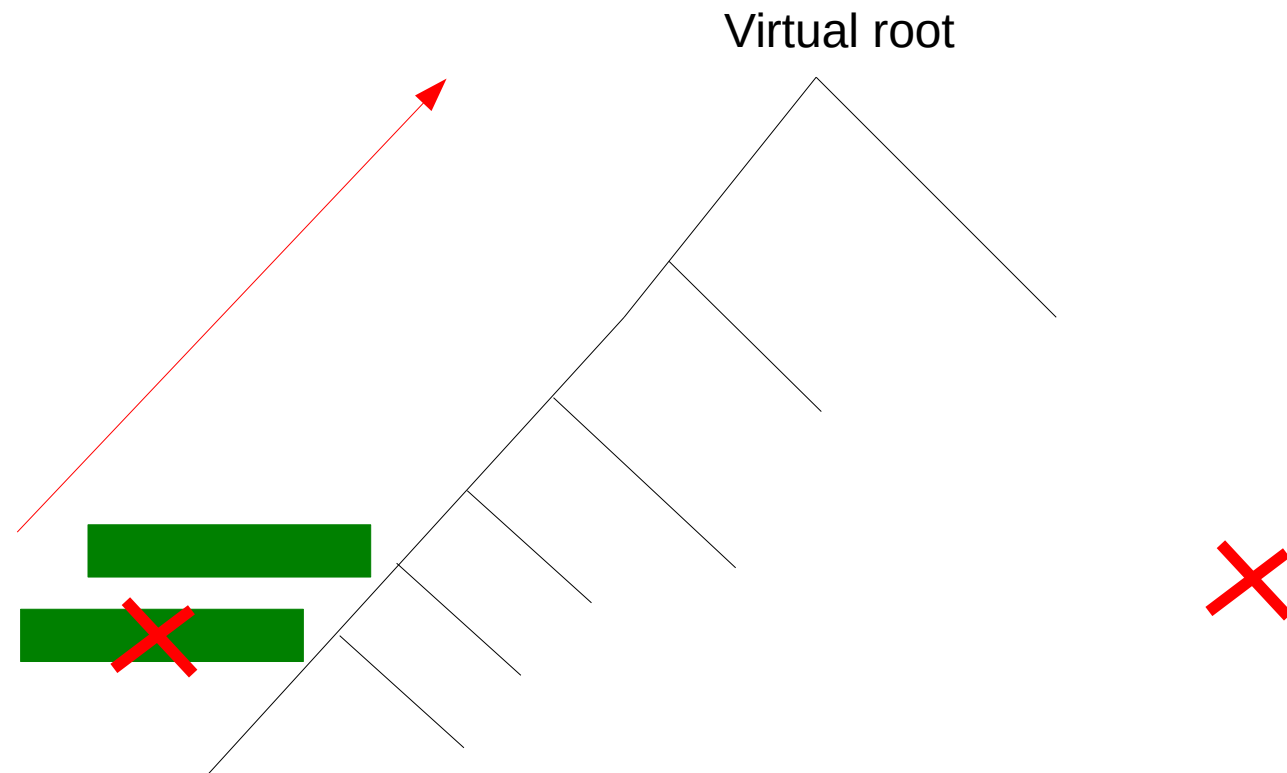
<https://doi.org/10.1093/sysbio/syw075>

**Published:** 29 August 2016 **Article history** ▼

# Saving Memory on Fixed Trees

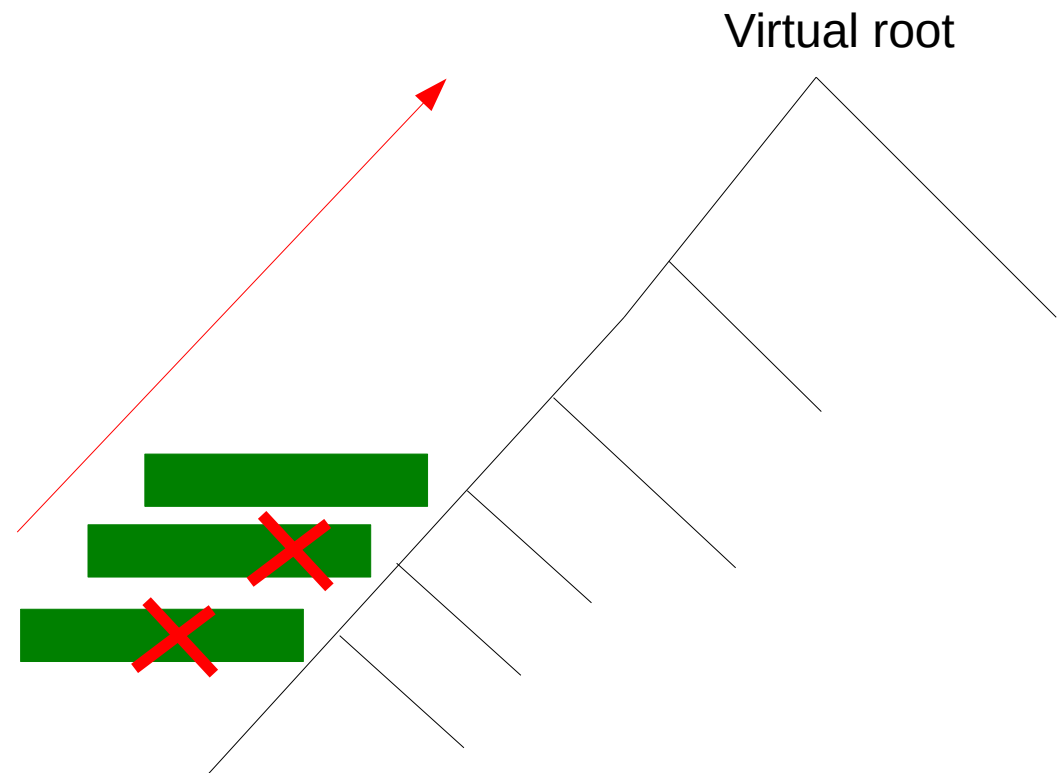


# Saving Memory on Fixed Trees

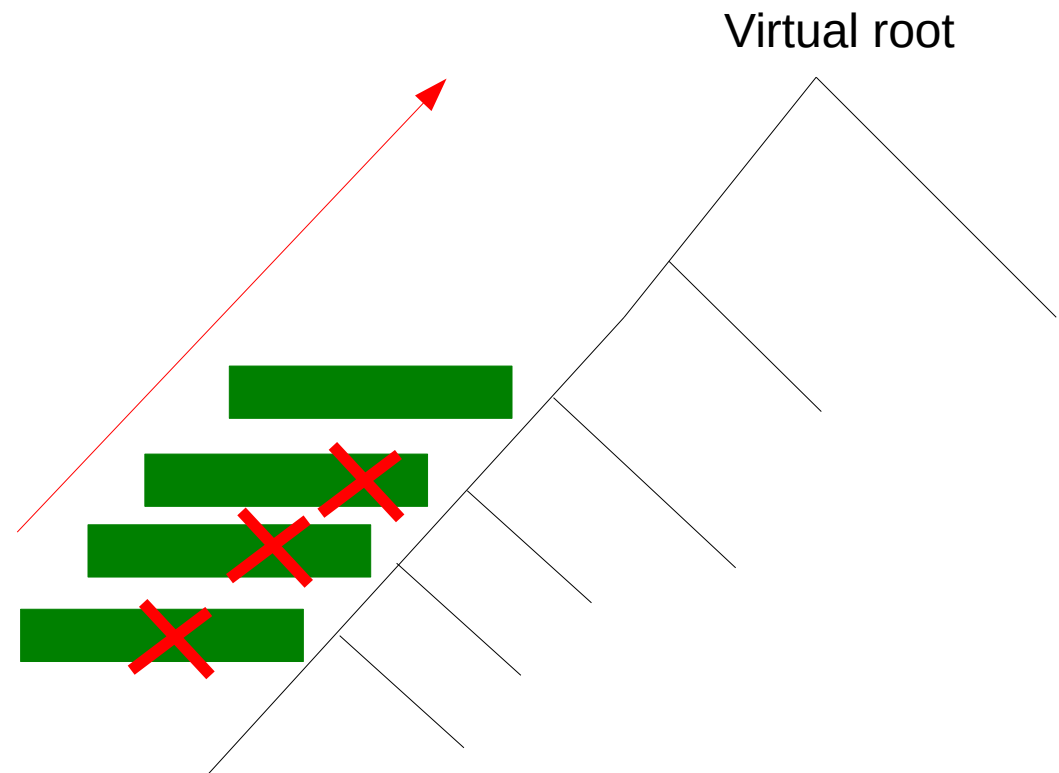




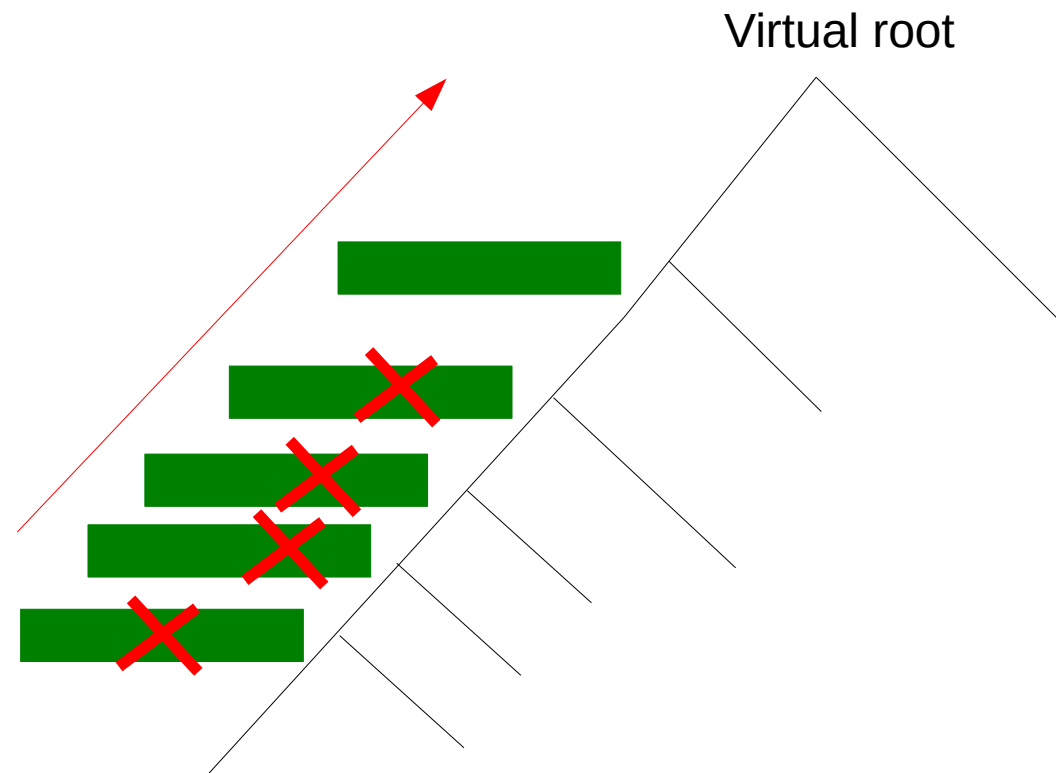
# Saving Memory on Fixed Trees



# Saving Memory on Fixed Trees



# Saving Memory on Fixed Trees



# Can we also do this on changing trees?

- Trade memory for additional re-computations
- A cache-like replacement strategy
- We need to store at least  $\log(n) + 2$  conditional likelihood vectors to compute the likelihood on any unrooted binary tree with  $n$  tips

## TRADING MEMORY FOR RUNNING TIME IN PHYLOGENETIC LIKELIHOOD COMPUTATIONS

Fernando Izquierdo-Carrasco<sup>1</sup>, Julien Gagneur<sup>2</sup> and Alexandros Stamatakis<sup>1</sup>

<sup>1</sup>The Exelixis Lab, Scientific Computing Group, Heidelberg Institute for Theoretical Studies, Schloss-Wolfsbrunnengweg 35, D-69118 Heidelberg, Germany

<sup>2</sup>European Molecular Biology Laboratory, Meyerhofstr. 1, 69117 Heidelberg, Germany  
fernando.izquierdo@h-its.org, julien.gagneur@embl.de, alexandros.stamatakis@h-its.org

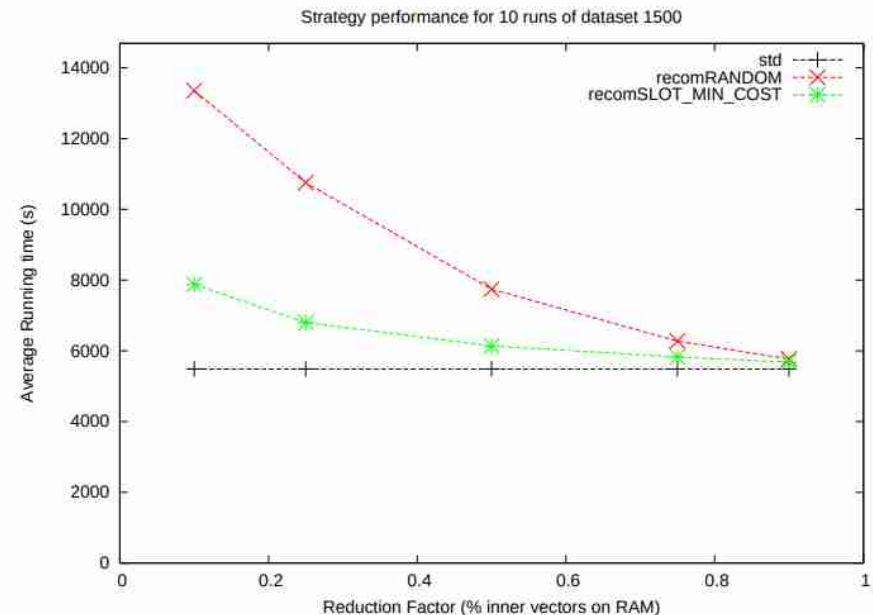
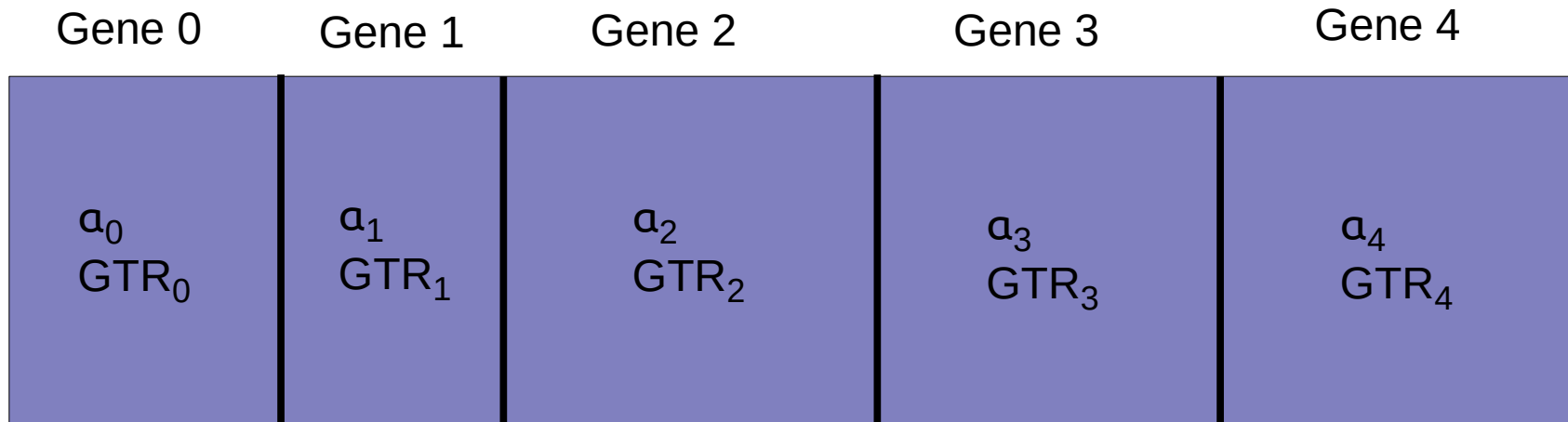


Figure 3: Different replacement strategies. The dataset was run with RAM allocations of 10%, 25%, 50%, 75%, and 90%, of the total required memory for storing all probability vectors. Run times are averaged across 10 searches with distinct starting trees.

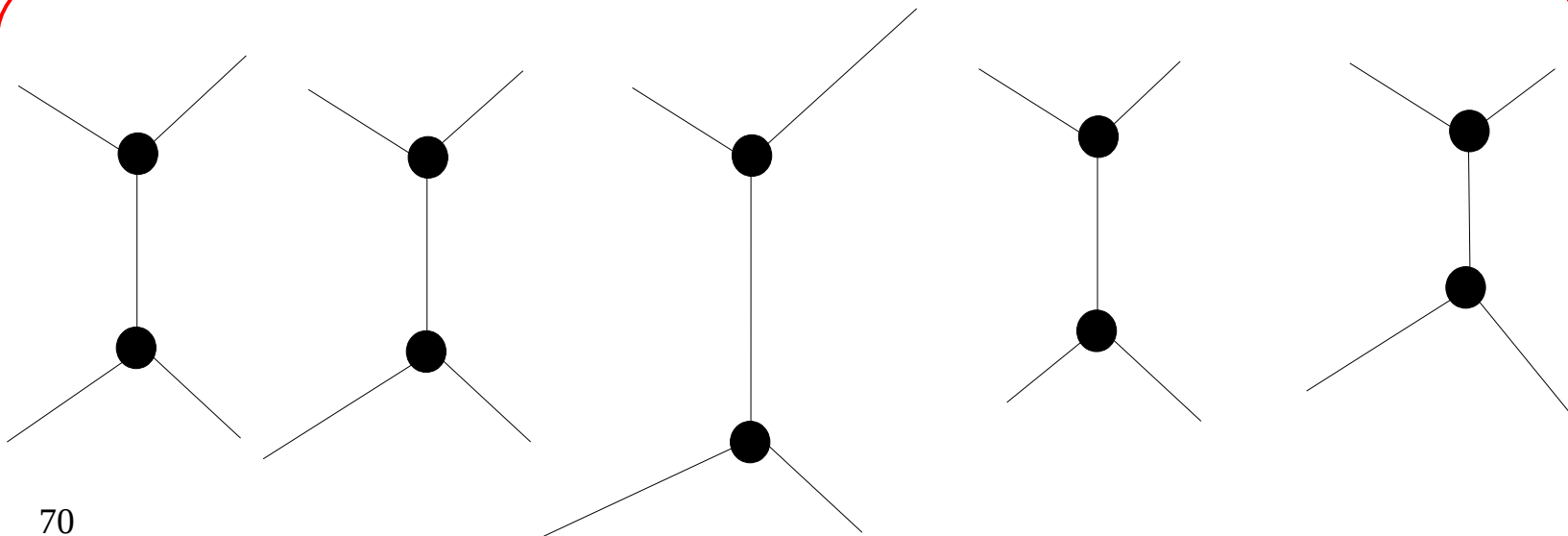
# The Real World

- Partitioned genomic datasets
- That's the kind of dataset type that real users analyze

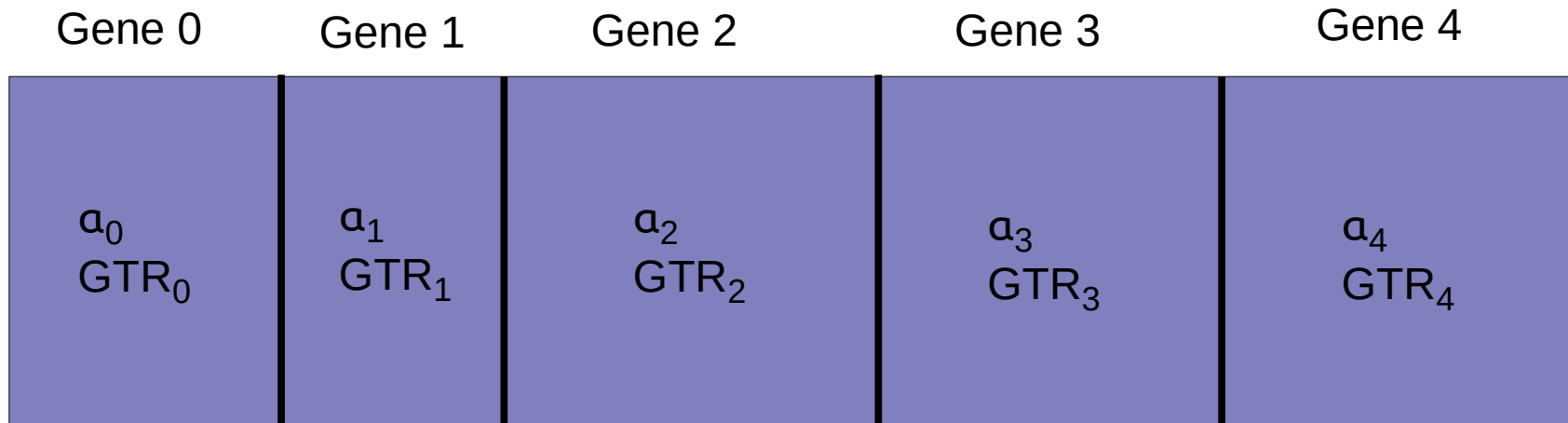
# Partitioned datasets



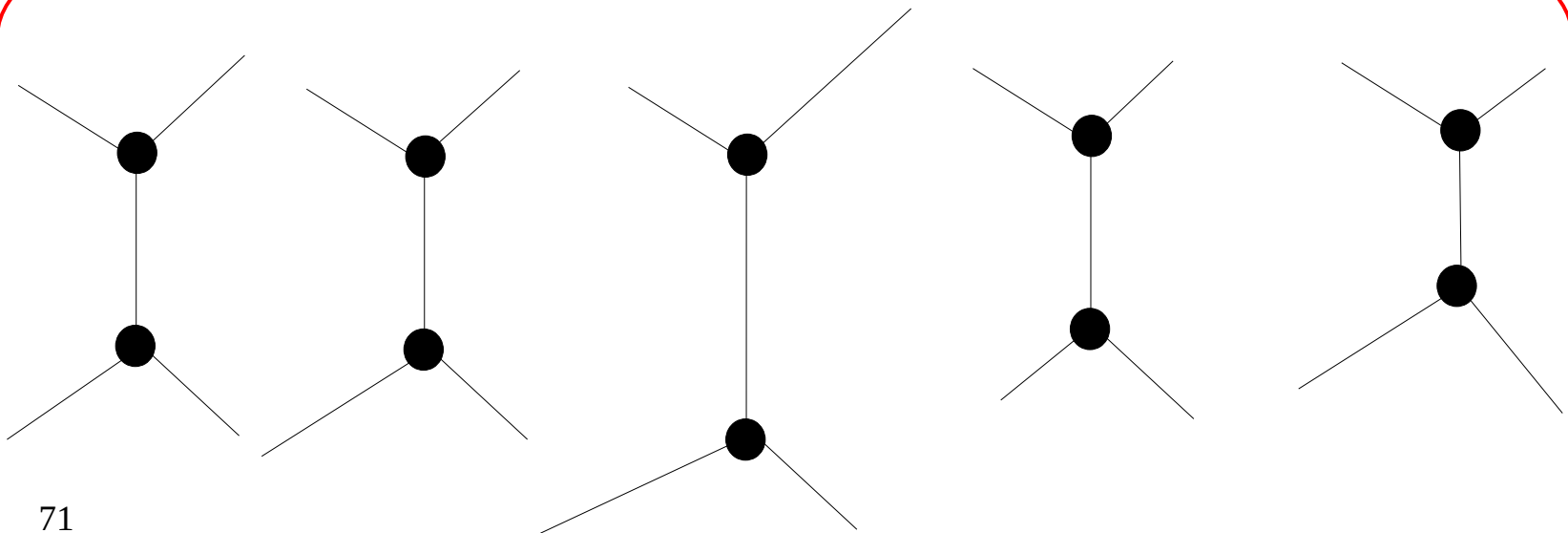
Same underlying tree topology!



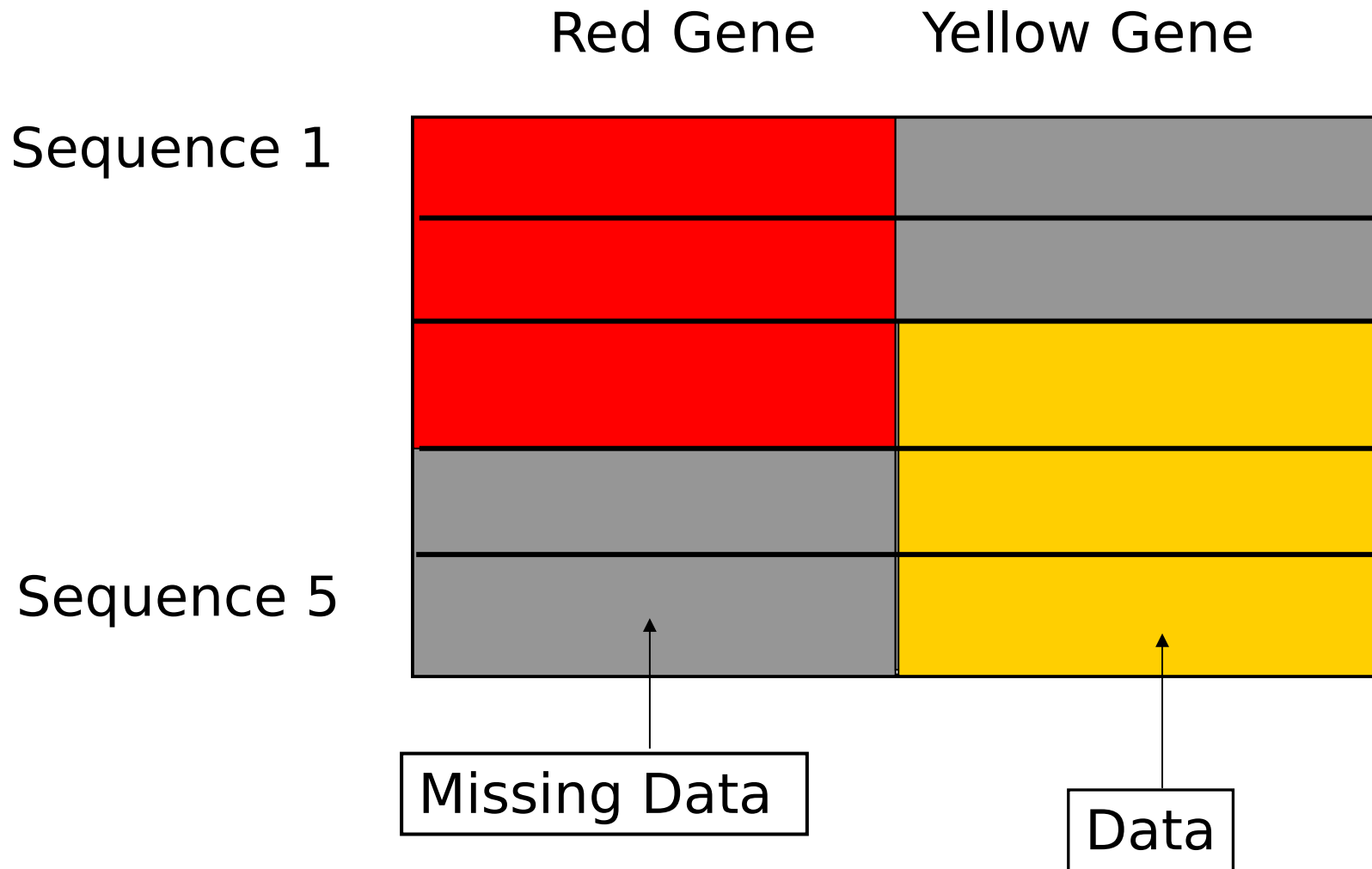
# Partitioned datasets



But with independent per-partition  
branch length estimate!  
→ In this case, terraces might occur!

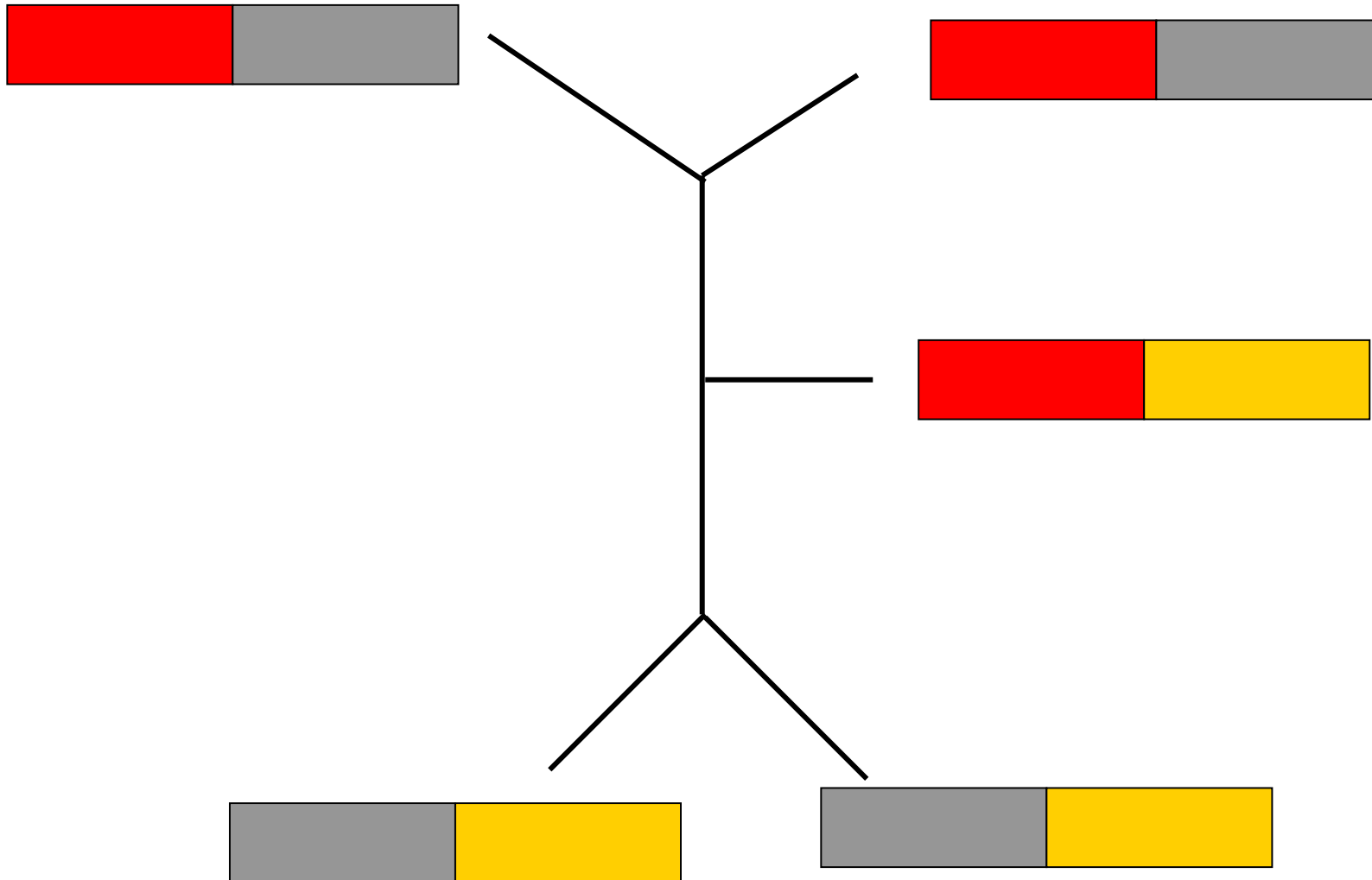


# Partitioned Data Example

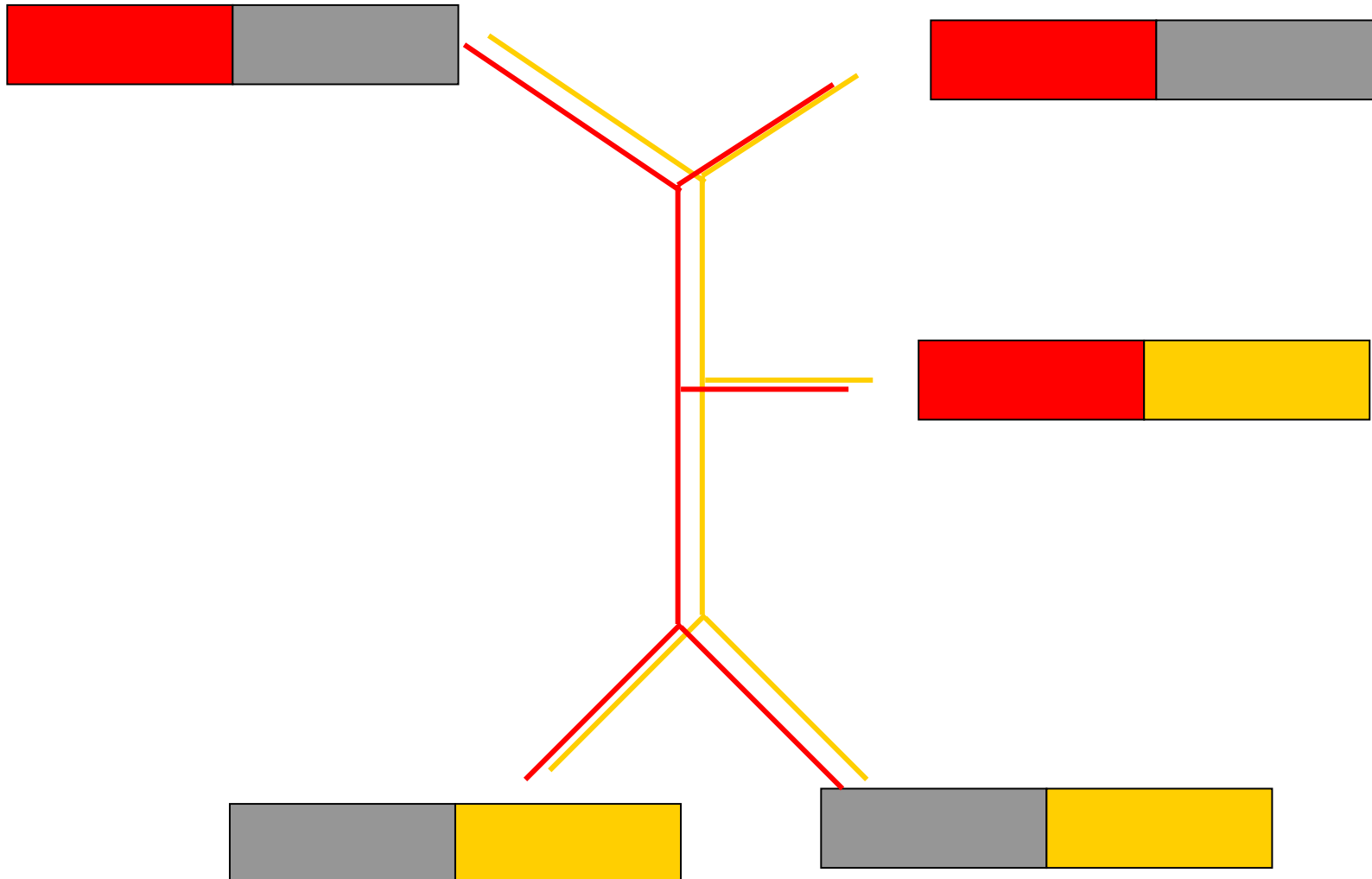




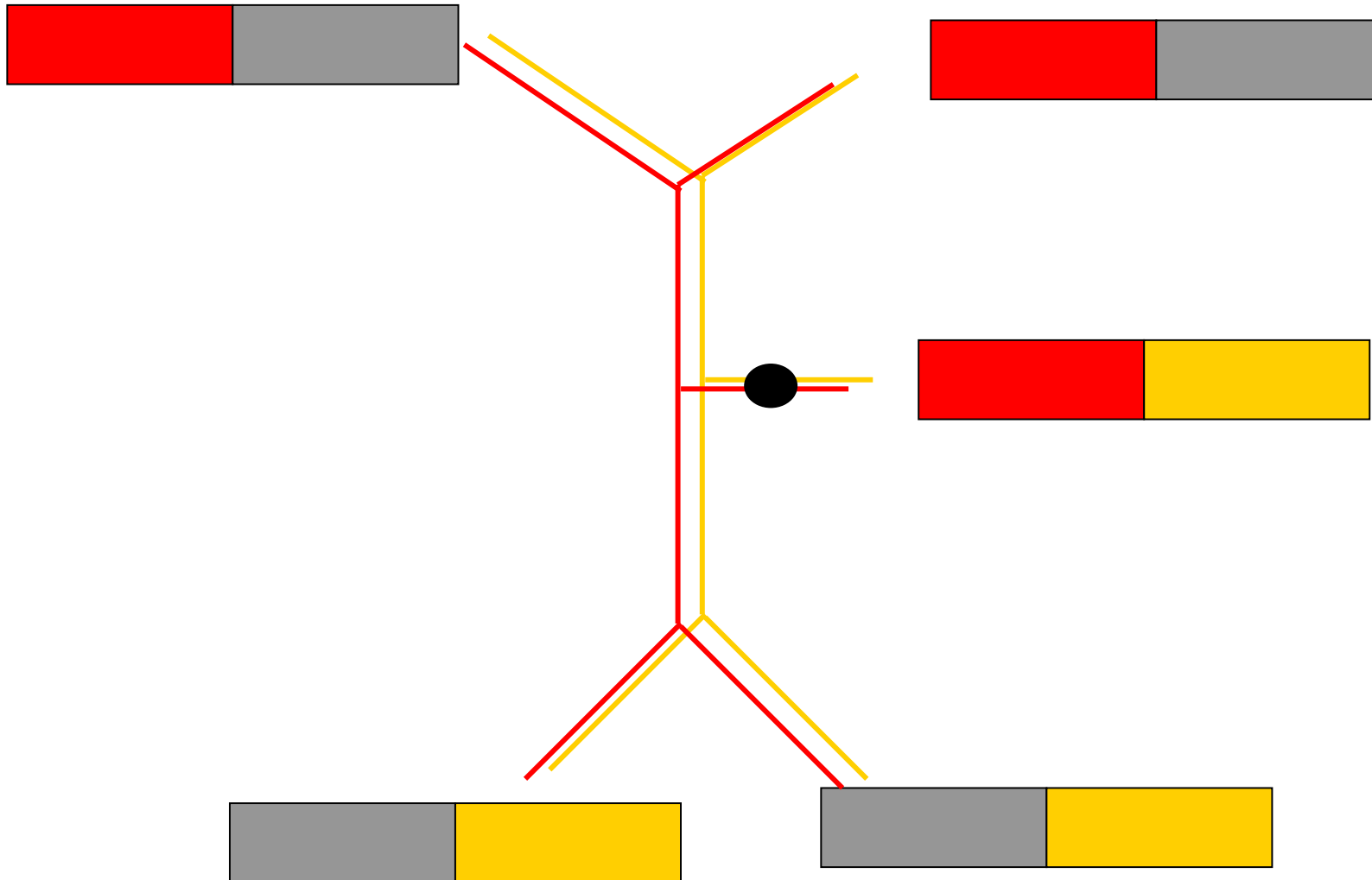
# Partitioned Data: Calculating the Likelihood



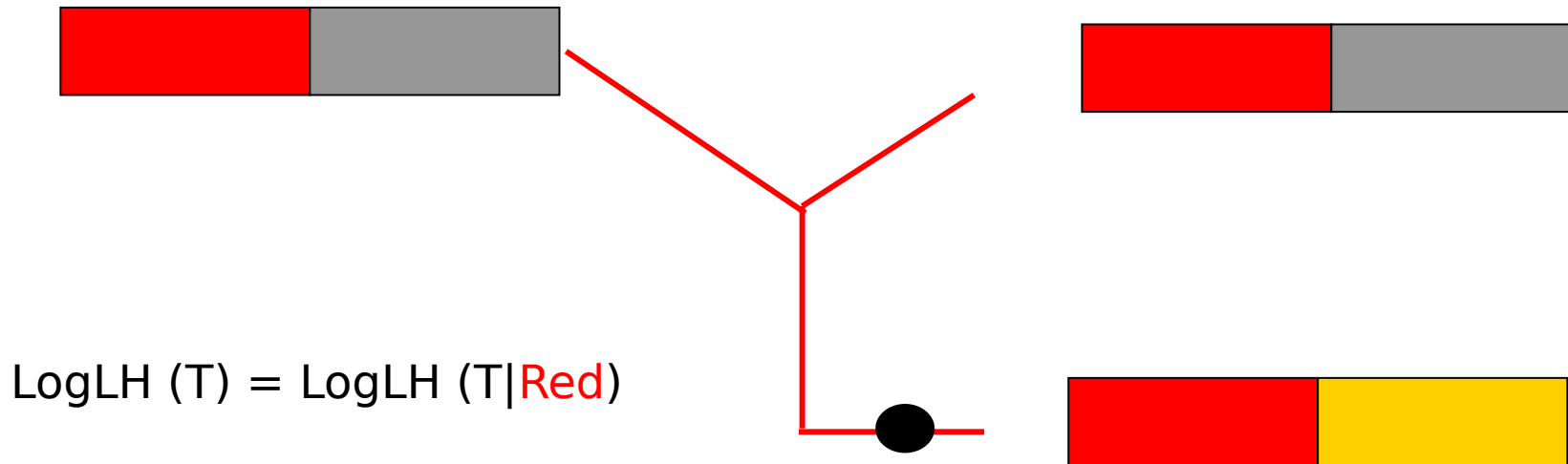
# Partitioned Data: Calculating the Likelihood



# Partitioned Data: Calculating the Likelihood

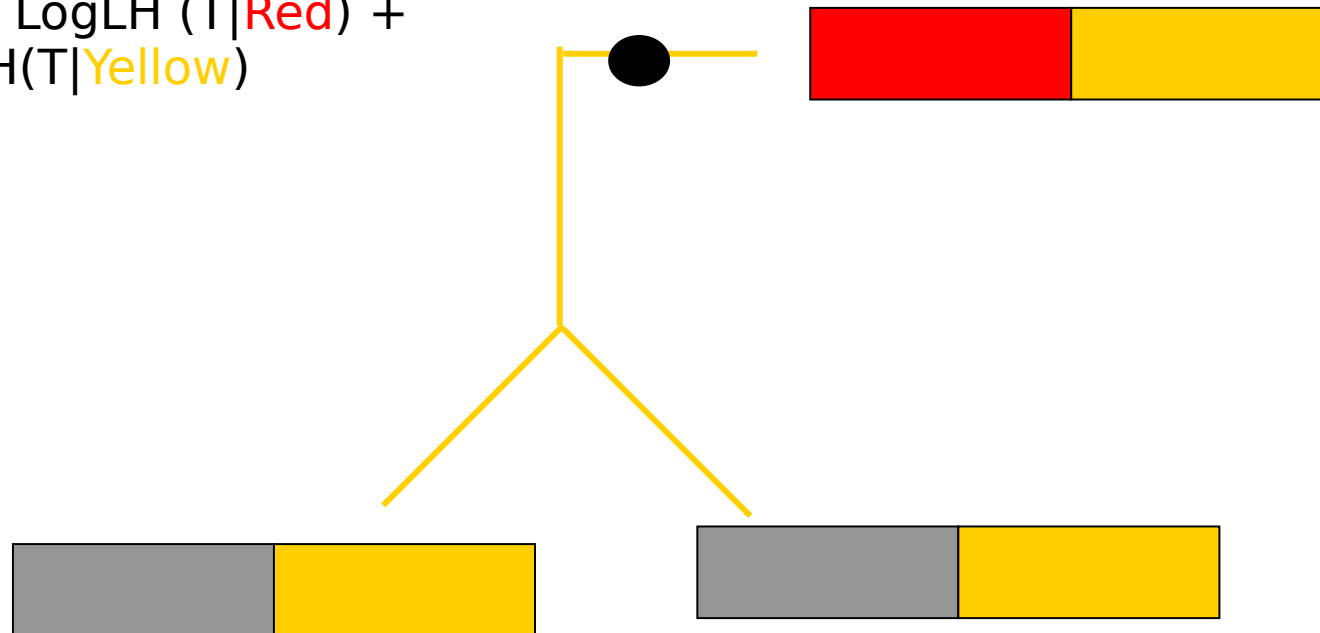


# Partitioned Data: Calculating the Likelihood

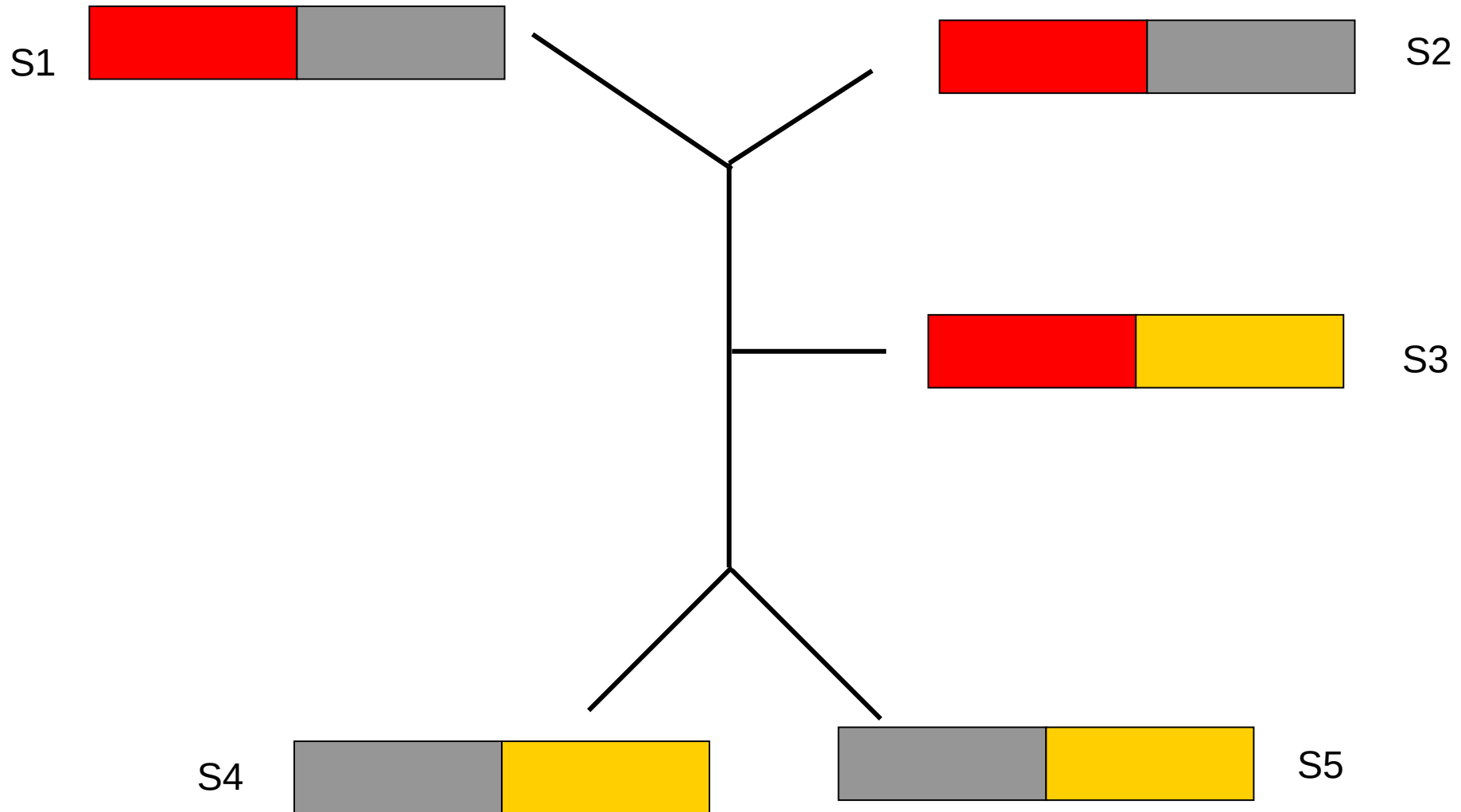


# Partitioned Data: Calculating the Likelihood

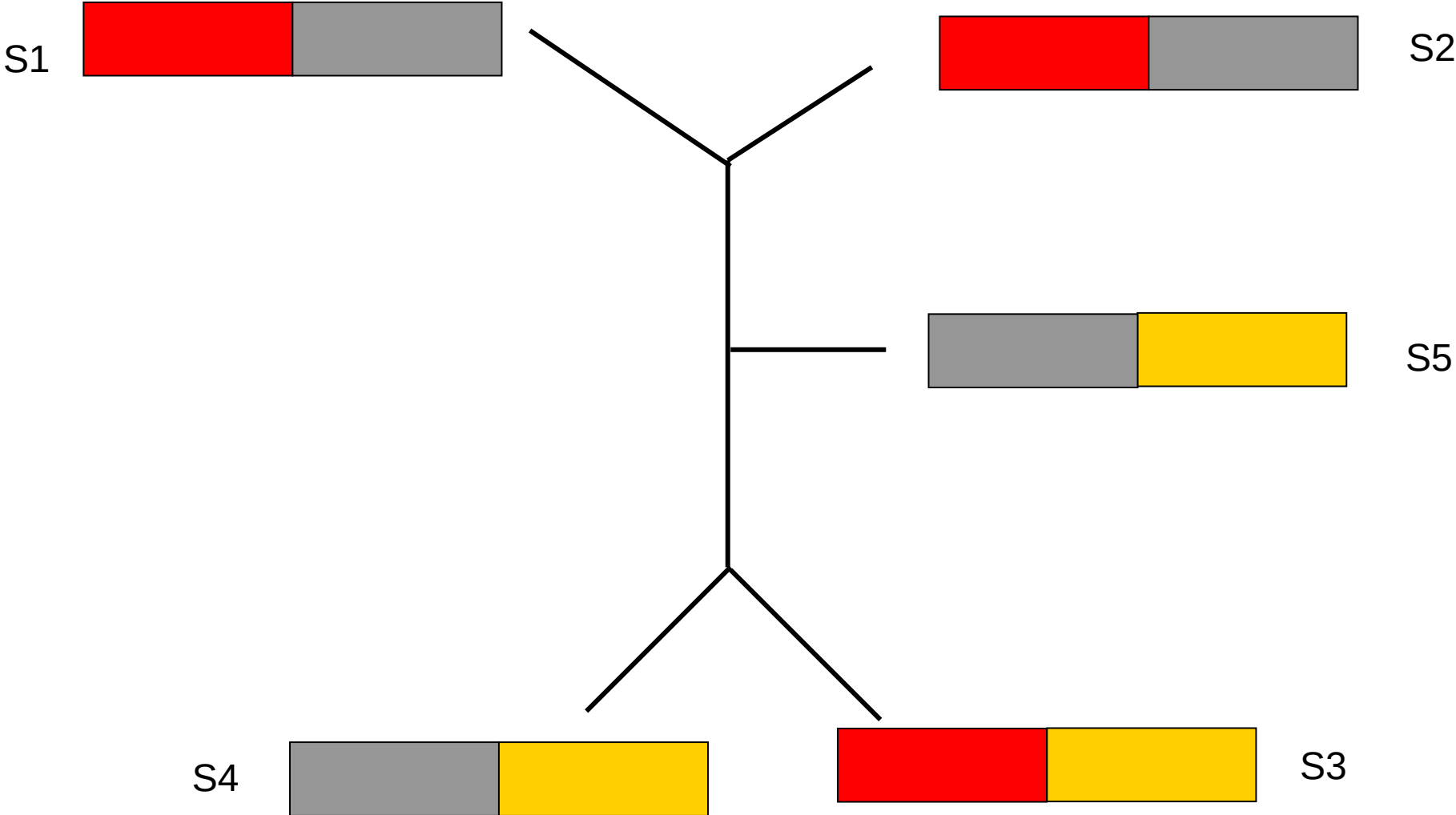
$$\text{LogLH}(T) = \text{LogLH}(T|\text{Red}) + \text{LogLH}(T|\text{Yellow})$$



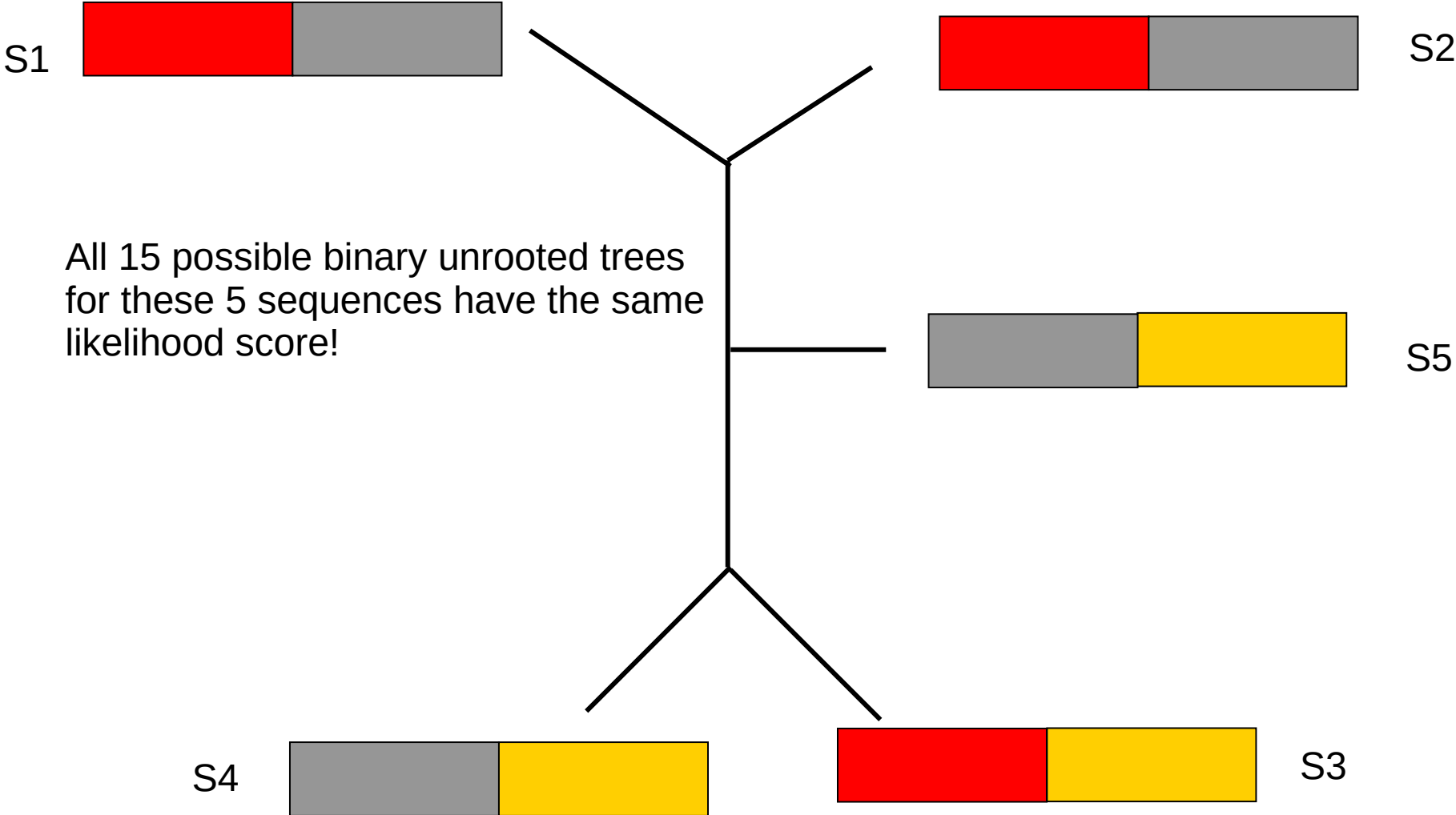
# Partitioned Data: We calculated the Likelihood on this tree



# What's the likelihood of this topologically different tree now?

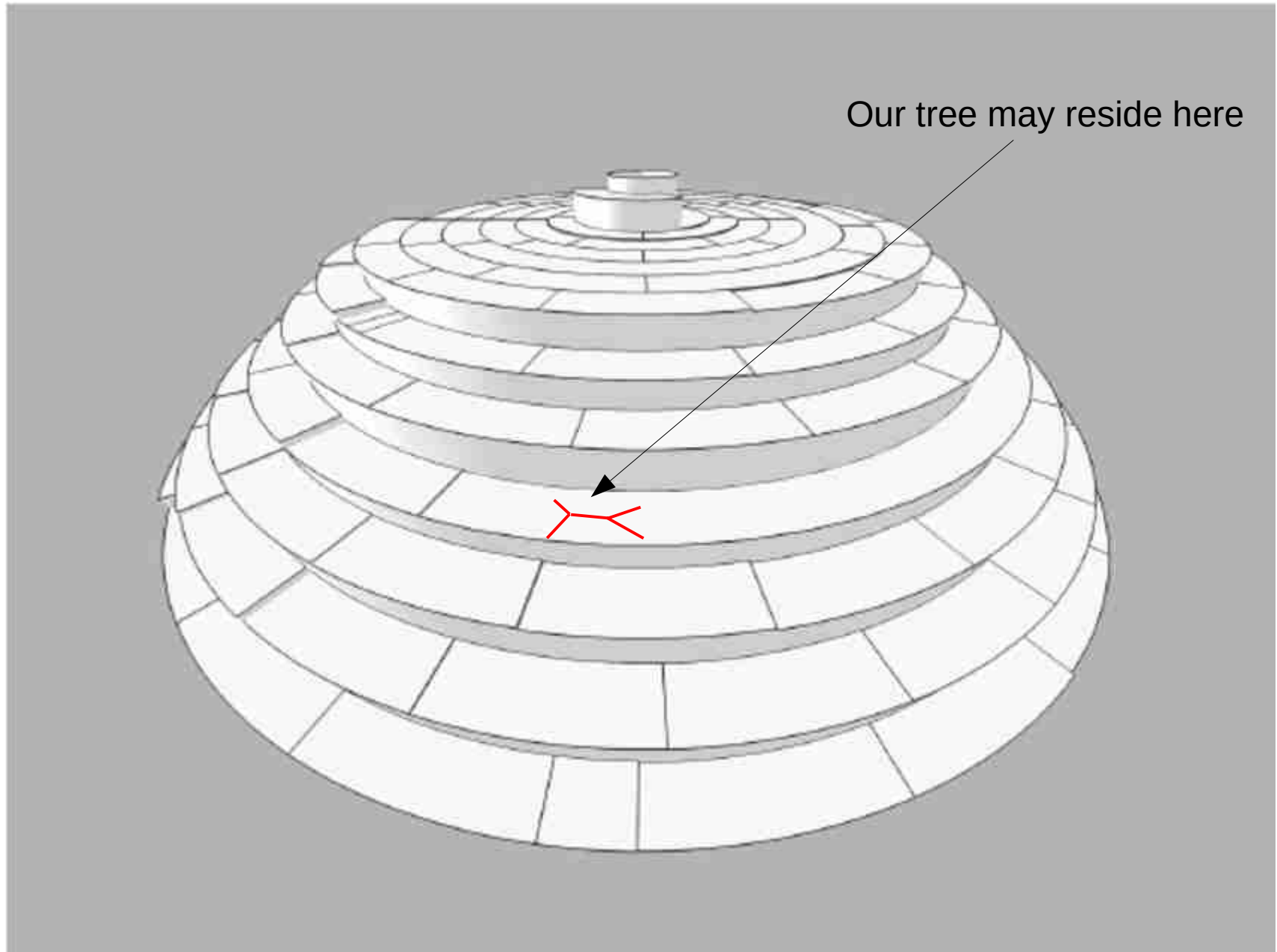


# What's the likelihood of this topologically different tree now?



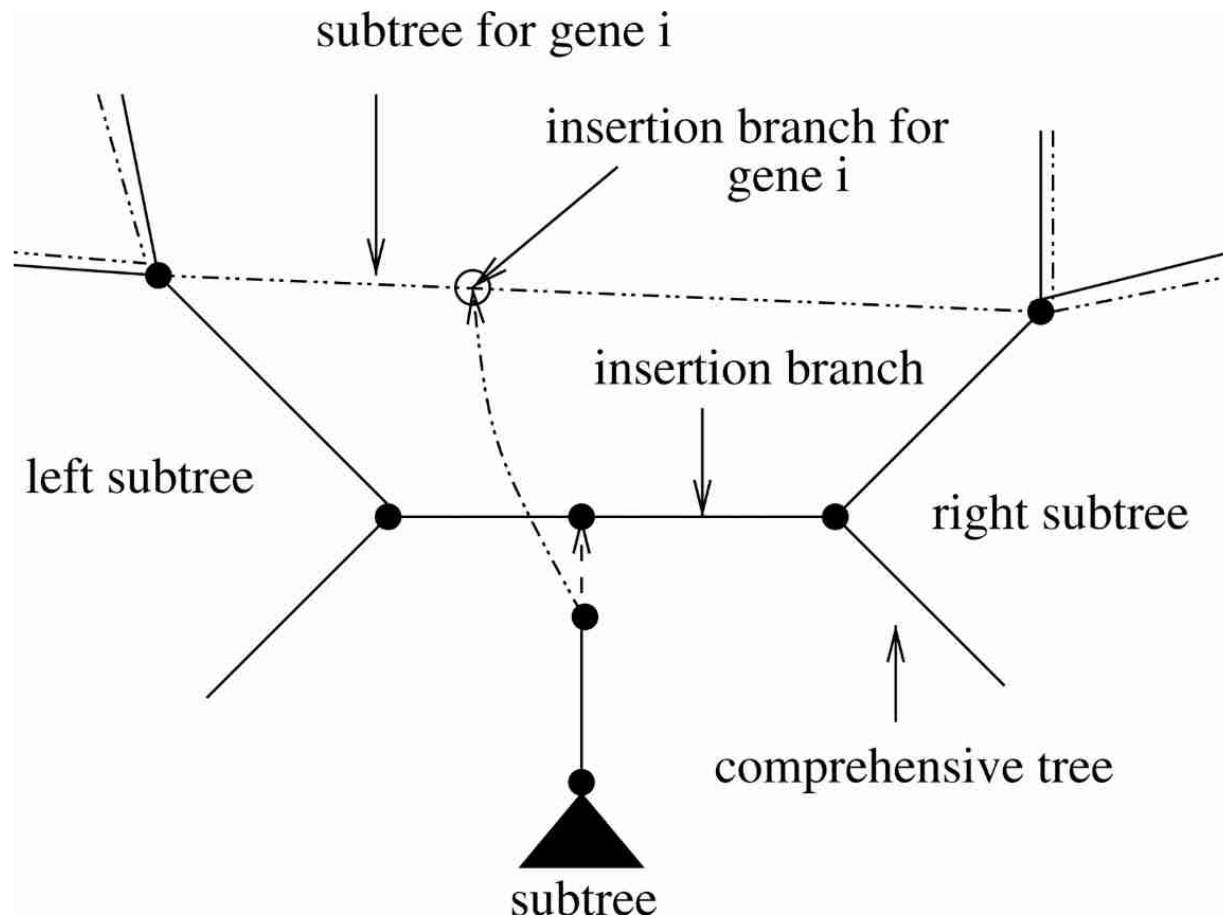


# A terrace in tree space



# Using Terraces to accelerate Likelihood Calculations

- Back in **2010** .... looking at SPR moves



# Implicit use of Terraces

JOURNAL ARTICLE

## Time and memory efficient likelihood-based tree searches on phylogenomic alignments with missing data

Alexandros Stamatakis , Nikolaos Alachiotis

*Bioinformatics*, Volume 26, Issue 12, June 2010, Pages i132–i139,

<https://doi.org/10.1093/bioinformatics/btq205>

**Published:** 01 June 2010

**Table 1.** Speedups of mesh-based likelihood approach versus standard approach

Dataset	Model optimization	Fast SPR	Slow SPR
d59_8	1.30	2.04	1.59
d94_1487	5.56	16.69	4.41
d126_34	1.34	1.79	1.80
d404_11	3.05	4.91	3.51
d2177_68	11.24	16.08	10.26
d37831_6	3.86	5.36	3.99

# Take home message

- Sometimes don't be such an engineer!
- **2011**

 REPORT



## Terraces in Phylogenetic Tree Space

MICHAEL J. SANDERSON, MICHELLE M. MCMAHON, AND MIKE STEEL [Authors Info & Affiliations](#)

SCIENCE • 16 Jun 2011 • Vol 333, Issue 6041 • pp. 445-450 • DOI: 10.1126/science.1206357

 248  71



 CHECK ACCESS

# Terraces

- Essentially we have an identifiability problem here!
  - Different parameter values (tree topologies) yield exactly the same analytical likelihood score!
- Trees and datasets exhibiting terraces are more frequent in published empirical studies than one might think!

# Outline

- Maximum Likelihood (Recap)
- Sequential Optimization
- **Parallelization**
- Parallel I/O
- Numerical Nightmares
- Energy Efficiency

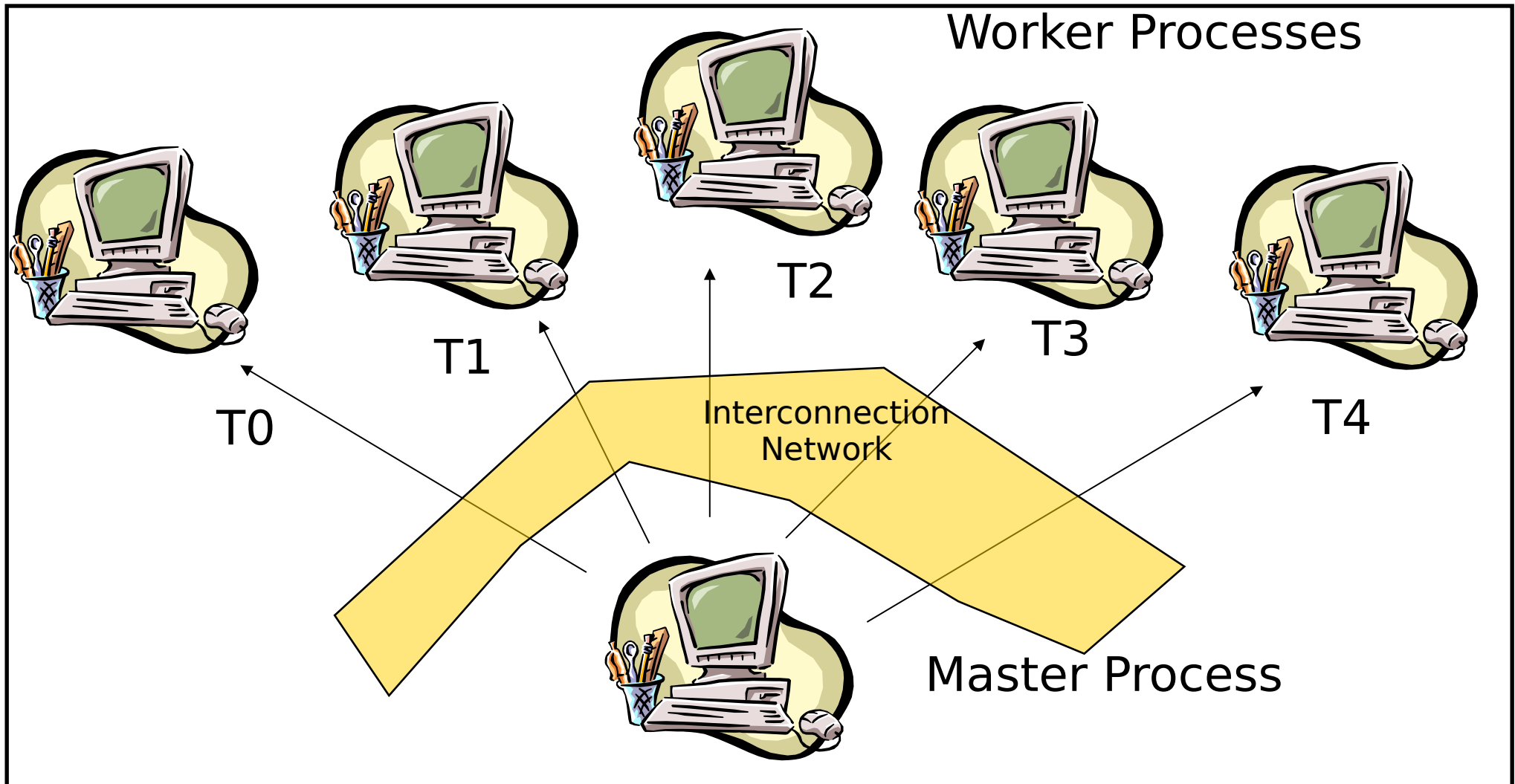
# Levels of Parallelism

Embarrassing Parallelism

MPI, Internet, Cloud

# Coarse-Grained Parallelism in RAxML-NG

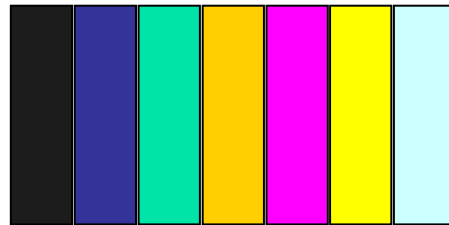
## PC-CLUSTER



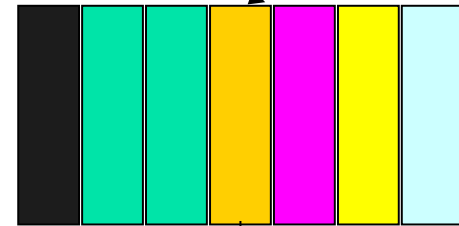
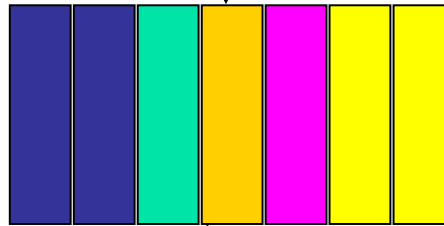
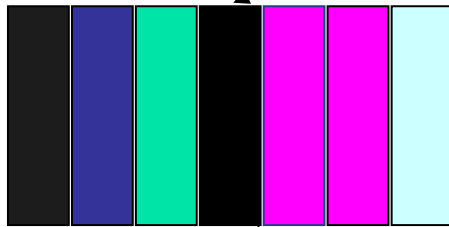


# Bootstrapping

Original Alignment



Perturbation



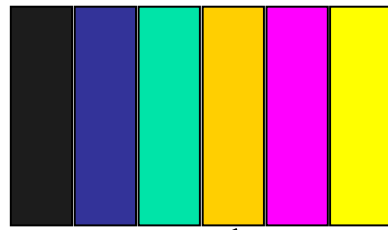
compute tree

compute tree

compute tree

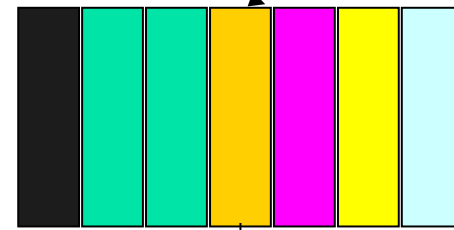
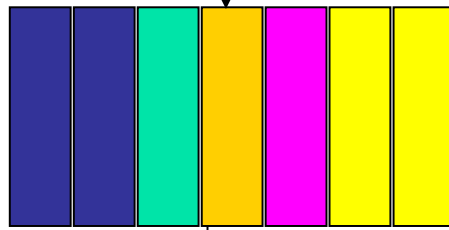
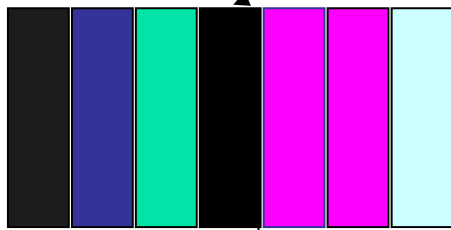
# Bootstrapping

Original Alignment



This needs to be done 100-1,000 times!  
Embarrassingly parallel problem!

Perturbation

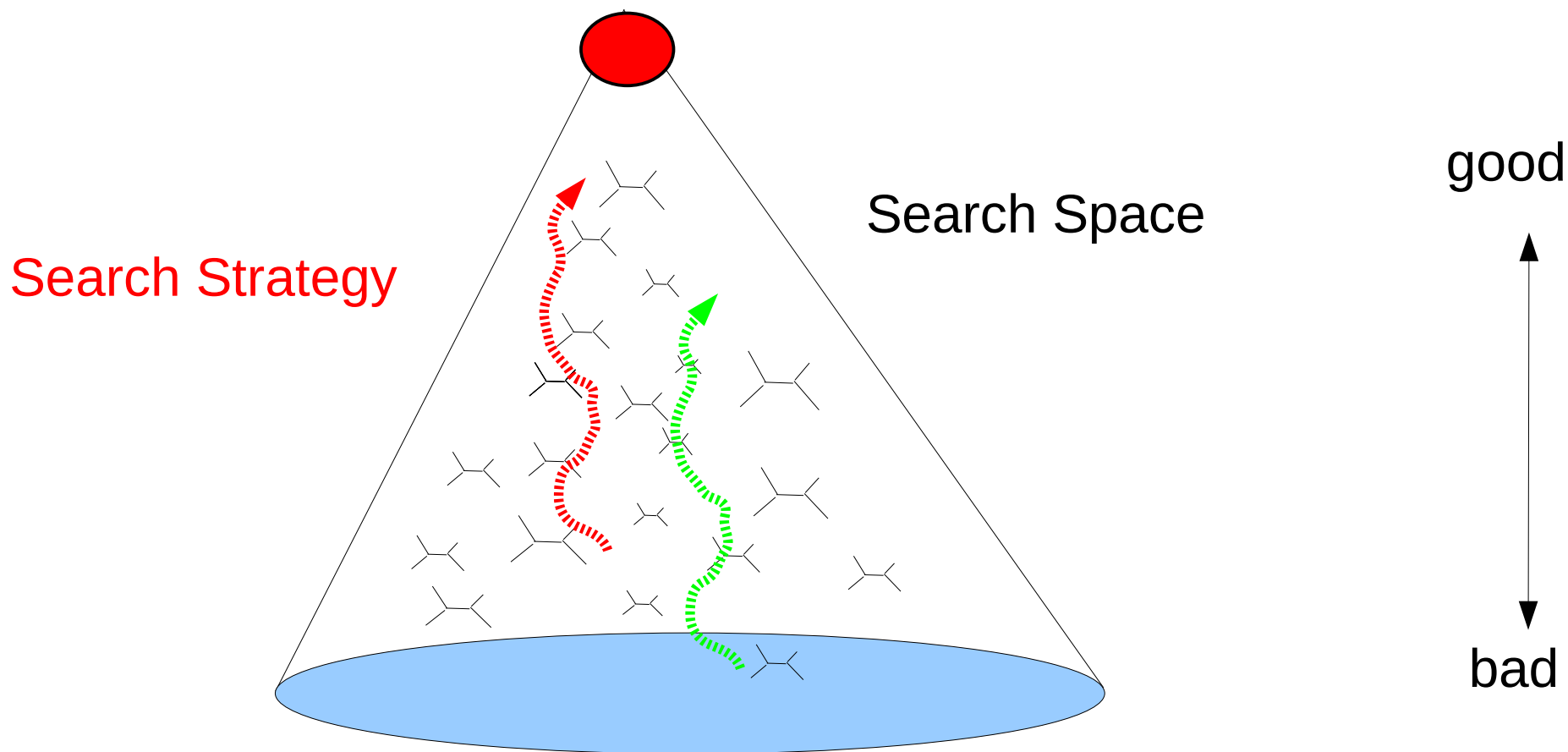


compute tree

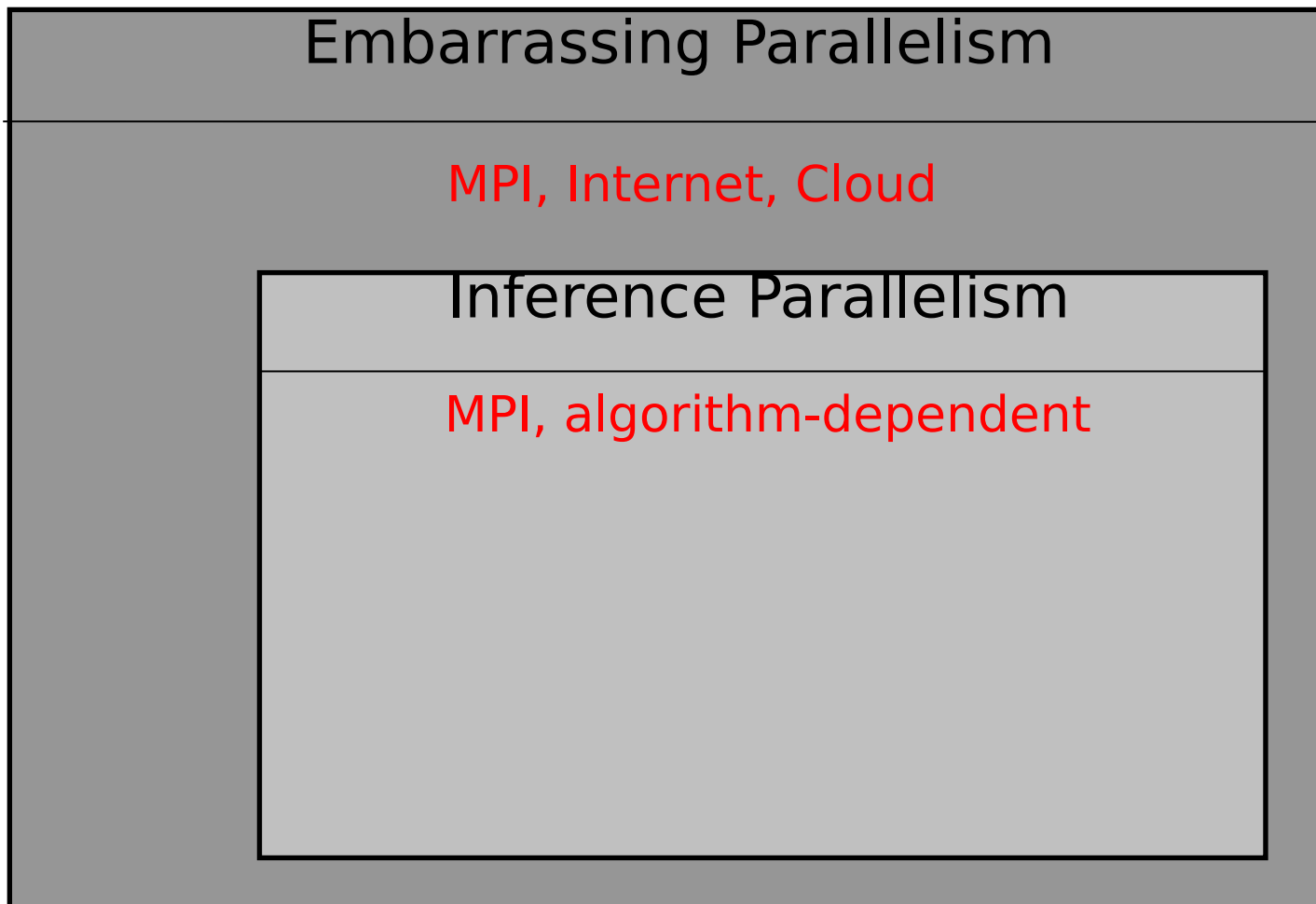
compute tree

compute tree

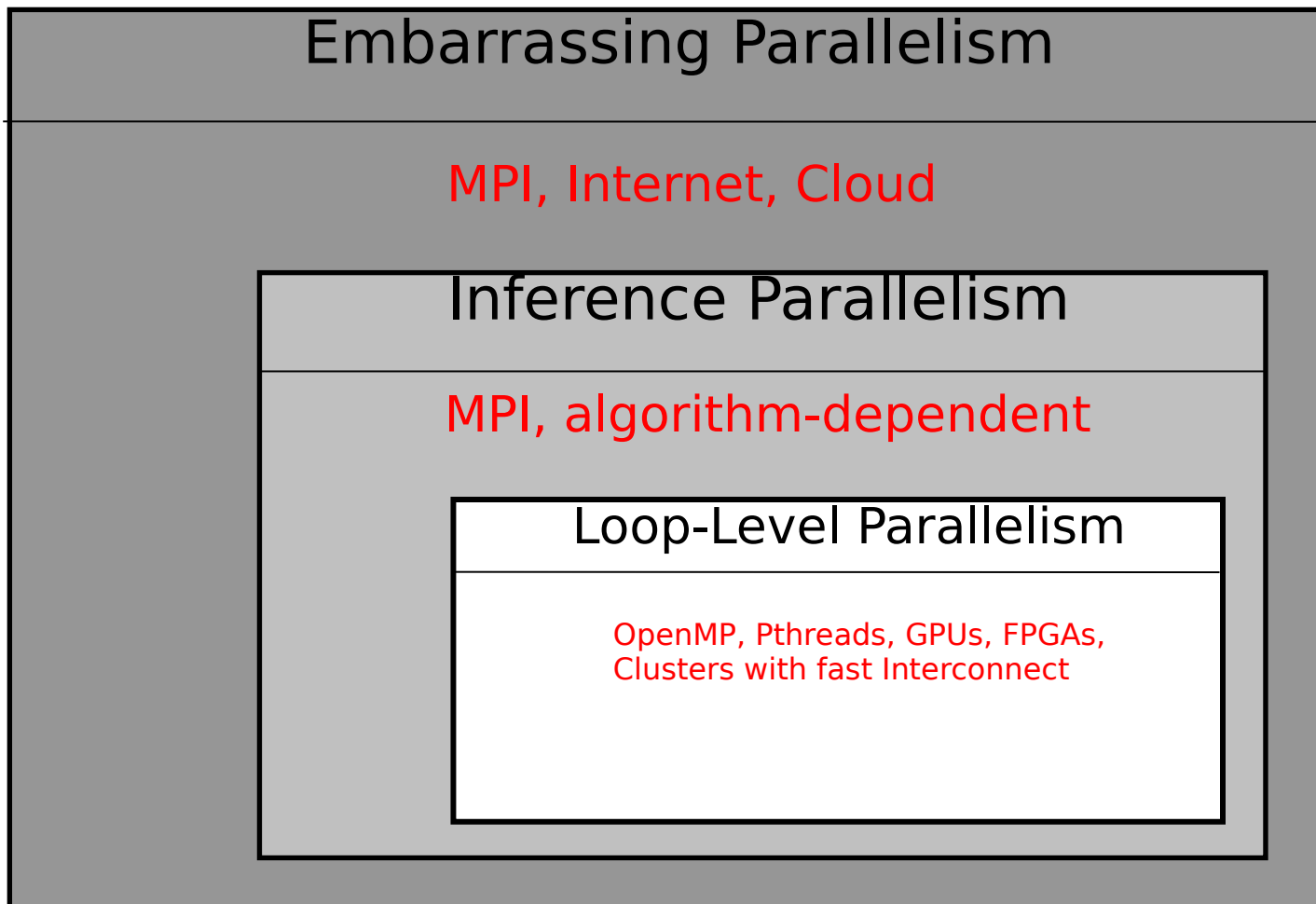
# Search Strategies ML Analyses



# Levels of Parallelism



# Levels of Parallelism



# Loop Level Parallelism

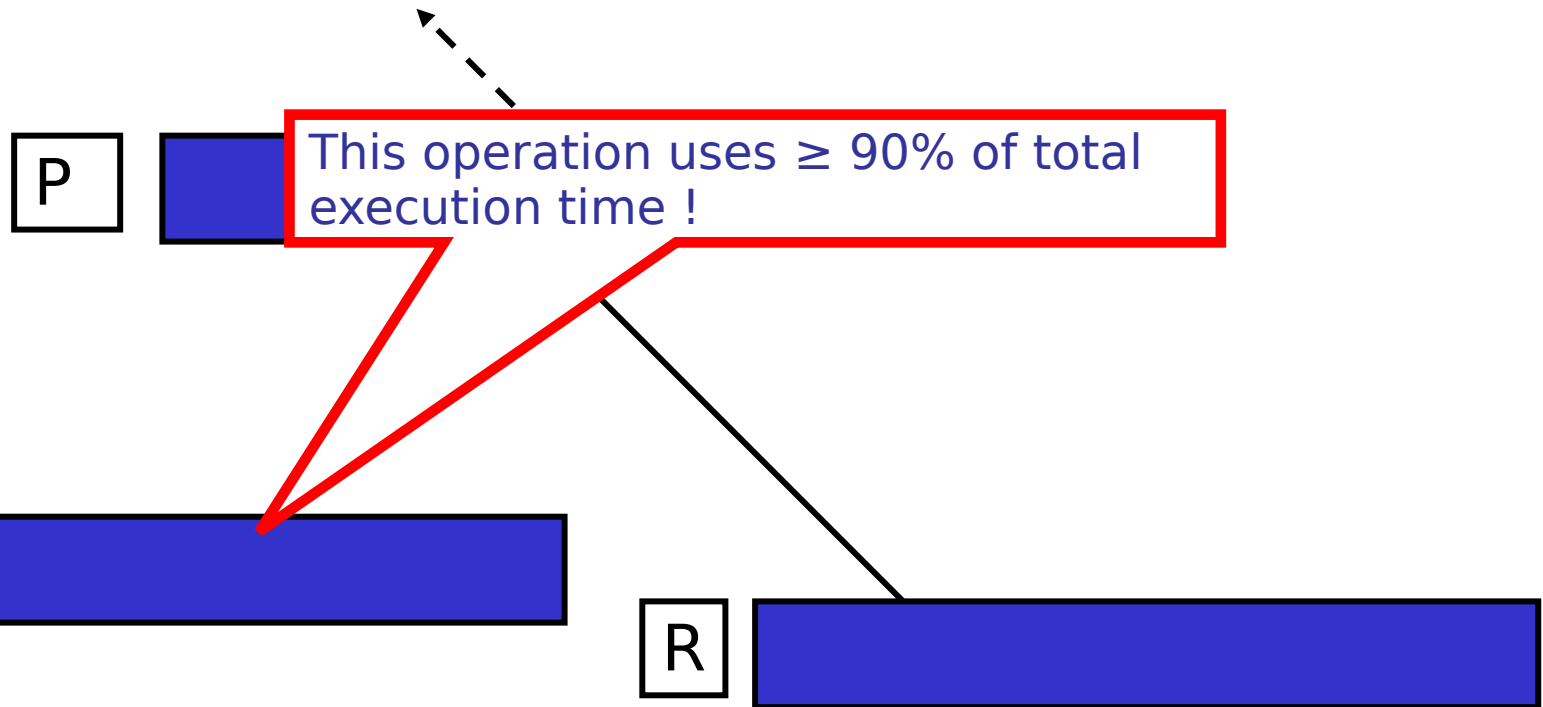
virtual root



$$P[i] = f(Q[i], R[i])$$

# Loop Level Parallelism

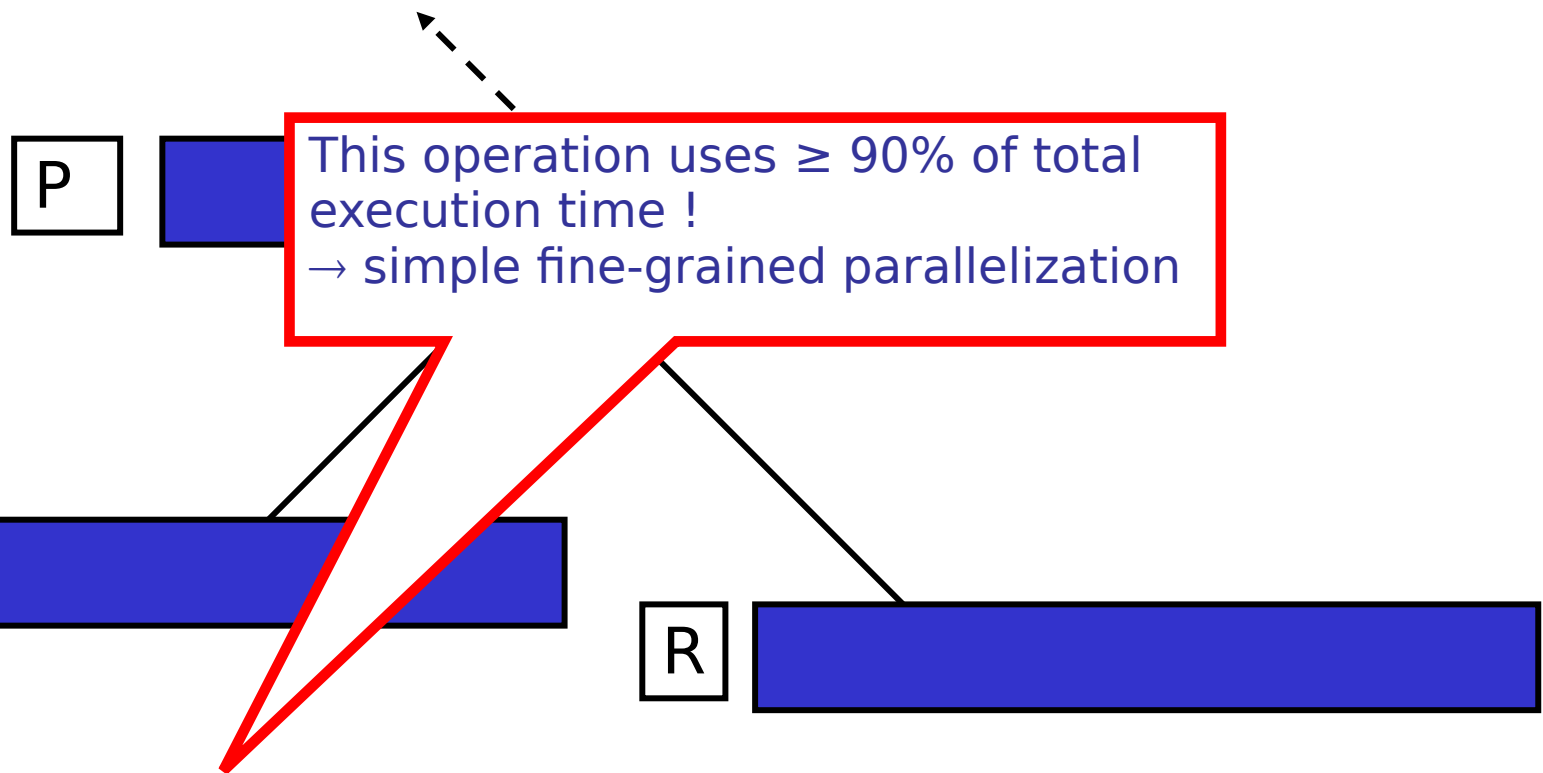
virtual root



$$P[i] = f(Q[i], R[i])$$

# Loop Level Parallelism

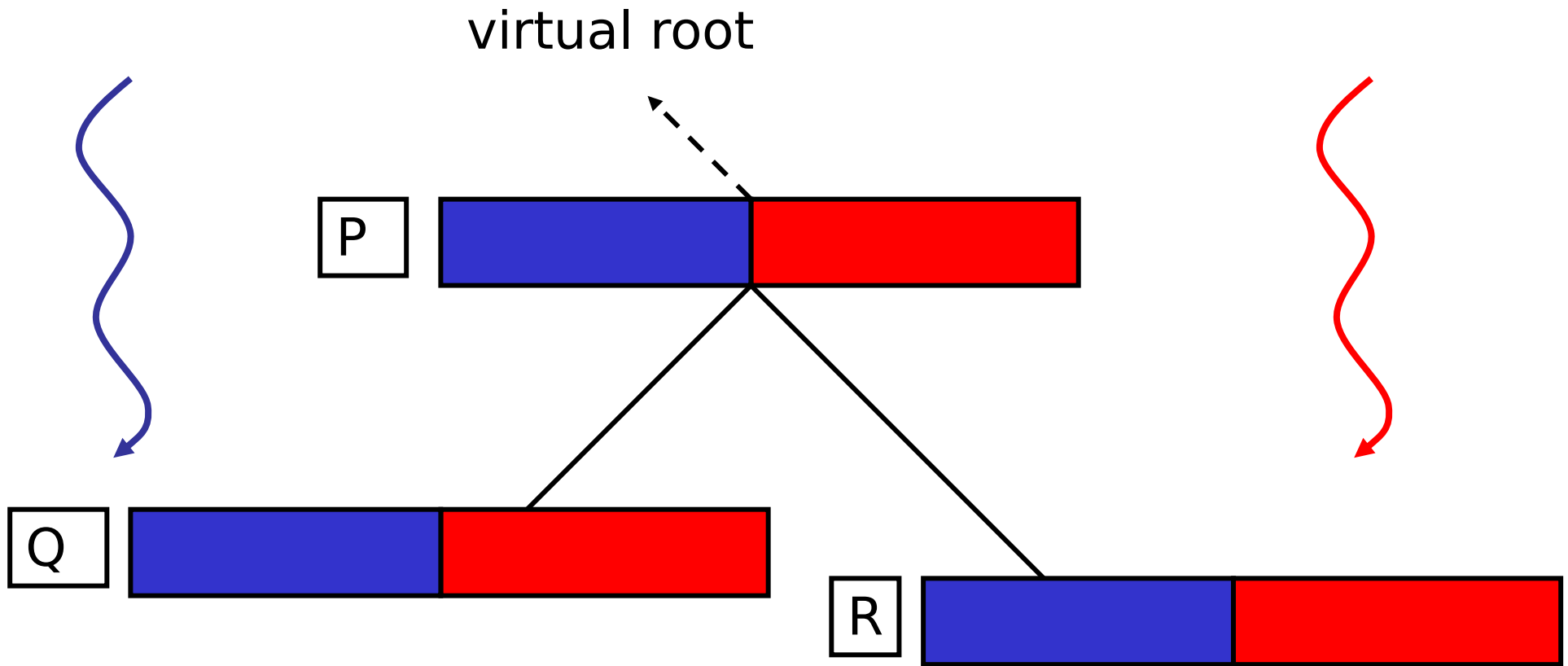
virtual root



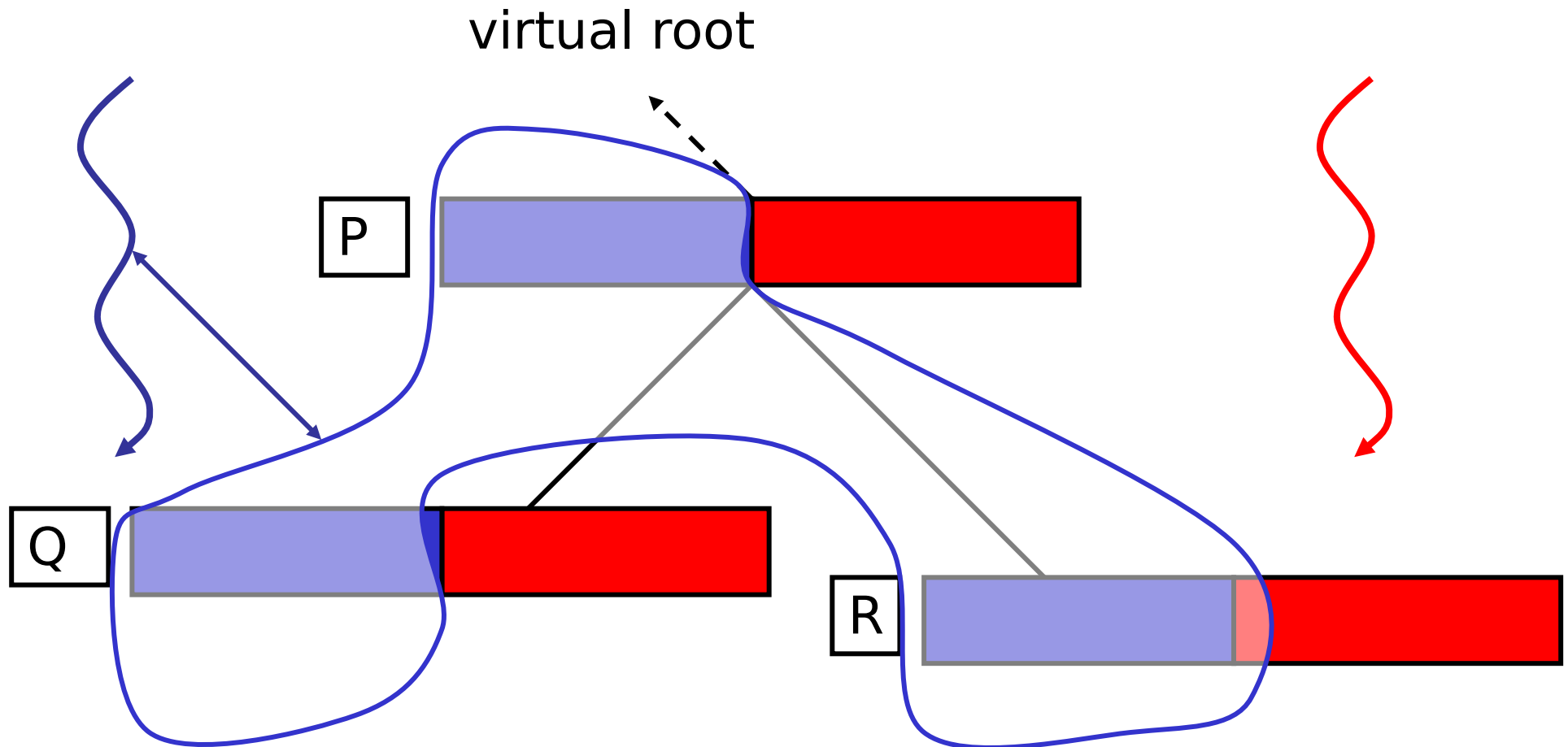
$$P[i] = f(Q[i], R[i])$$



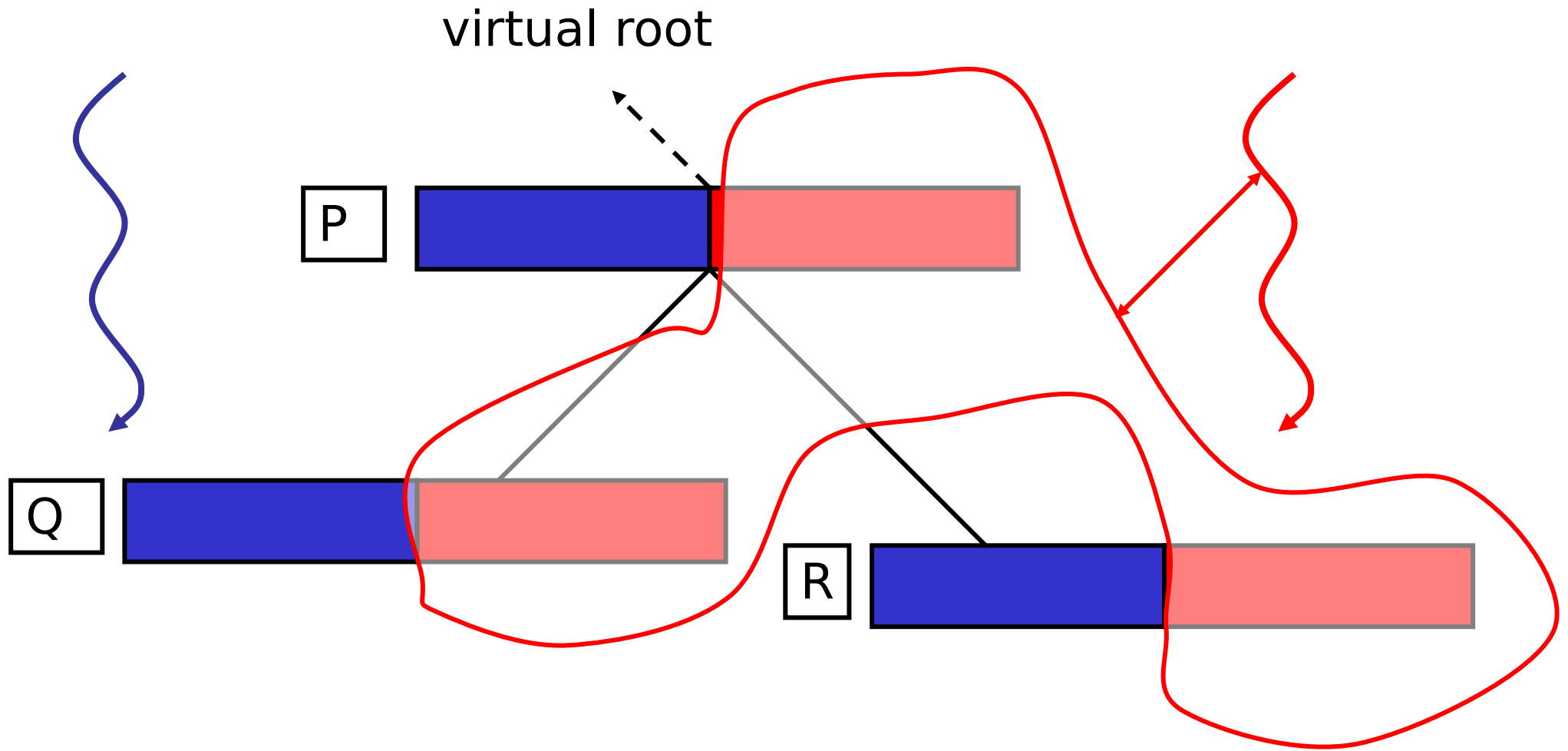
# Loop Level Parallelism



# Loop Level Parallelism



# Loop Level Parallelism



# OpenMP parallelization

```
for(i = 0; i < m; i++)  
    P[i] = f(Q[i],R[i]);
```

# OpenMP parallelization

```
for(i = 0; i < m; i++)  
    P[i] = f(Q[i],R[i]);
```

Iterations  $i$  and  $i+1$  can be computed independently of each other!

# OpenMP parallelization

```
for(i = 0; i < m; i++)  
    P[i] = f(Q[i],R[i]);
```

Iterations  $i$  and  $i+1$  can be computed independently of each other → parallelize with OpenMP

```
#pragma parallel for  
for(i = 0; i < m; i++)  
    P[i] = f(Q[i],R[i]);
```

# OpenMP parallelization

```
some other sequential code
#pragma parallel for
for(i = 0; i < 100; i++)
    P[i] = f(Q[i],R[i]);
some other sequential code
```

# OpenMP parallelization

some other **sequential** code

---

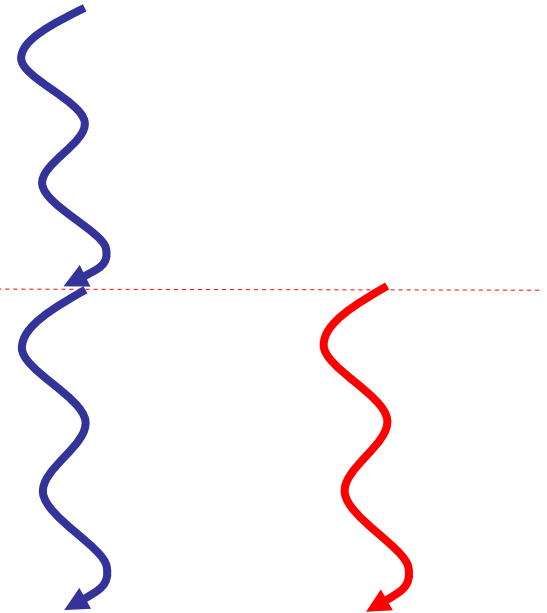
```
#pragma parallel for  
for(i = 0; i < 100; i++)  
    P[i] = f(Q[i],R[i]);  
some other sequential code
```





# OpenMP parallelization

```
some other sequential code  
-----  
#pragma parallel for  
for(i = 0; i < 100; i++)  
    P[i] = f(Q[i],R[i]);  
some other sequential code
```

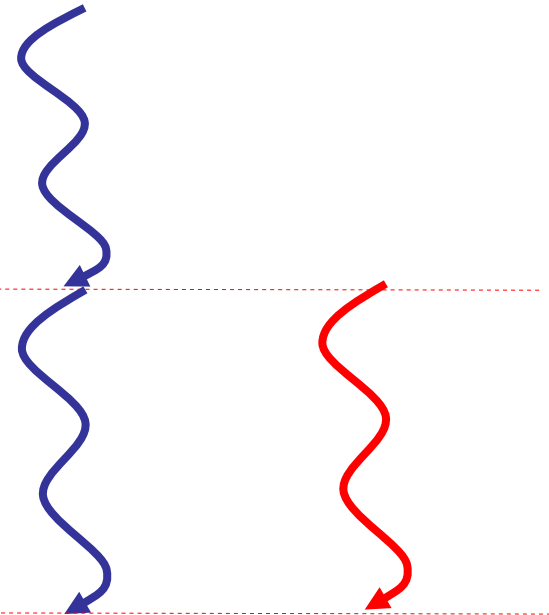


# OpenMP parallelization

some other **sequential** code

```
#pragma parallel for  
for(i = 0; i < 100; i++)  
    P[i] = f(Q[i],R[i]);
```

some other **sequential** code

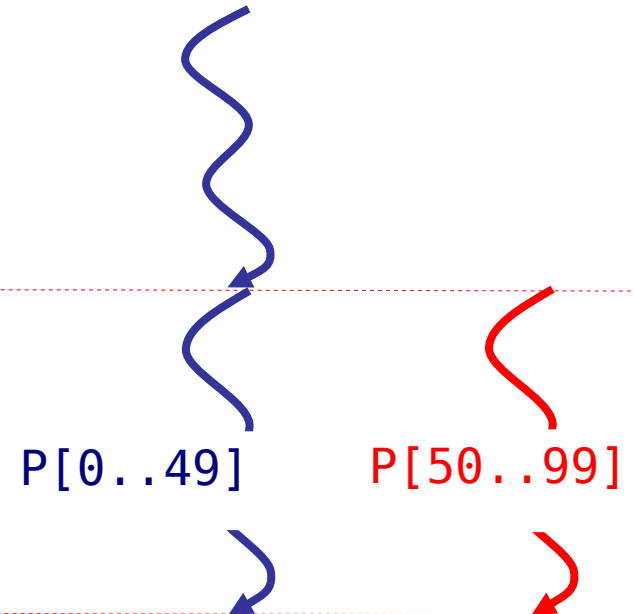


# OpenMP parallelization

some other **sequential** code

```
#pragma parallel for  
for(i = 0; i < 100; i++)  
    P[i] = f(Q[i],R[i]);
```

some other **sequential** code

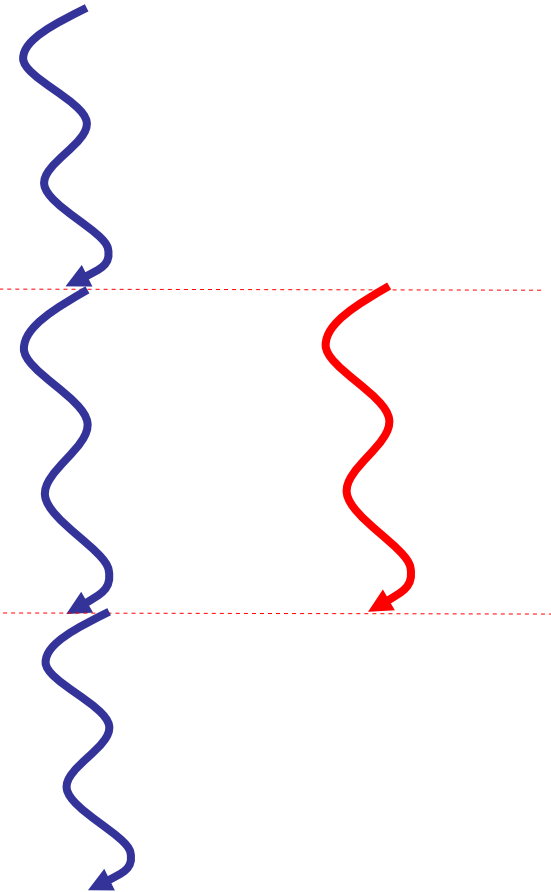


# OpenMP parallelization

some other **sequential** code

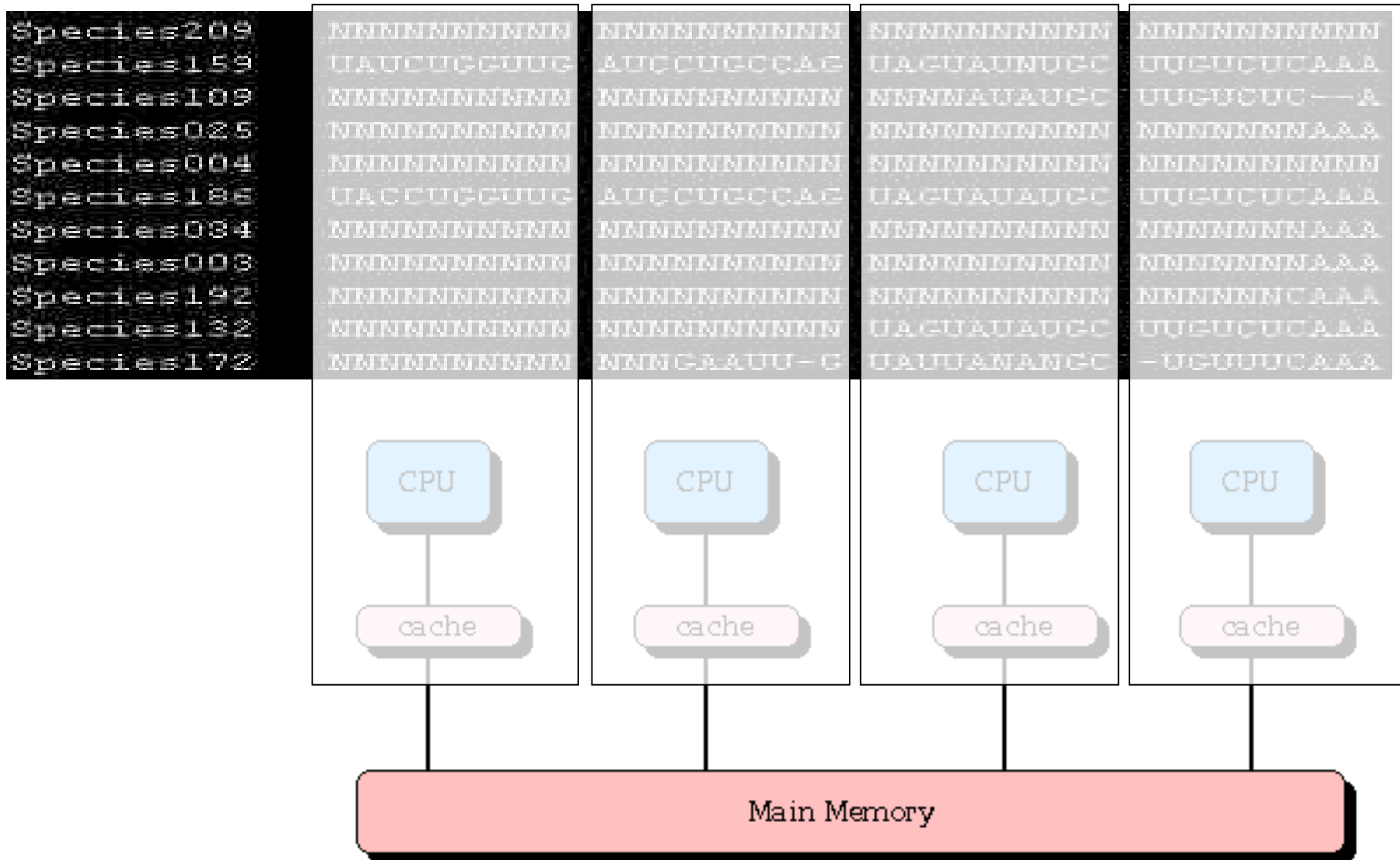
```
#pragma parallel for  
for(i = 0; i < 100; i++)  
    P[i] = f(Q[i],R[i]);
```

some other **sequential** code



# Loop-Level Parallelism

→ allows to use more cache memory capacity



# OpenMP parallelization

some other **sequential** code

```
#pragma parallel for  
for(i = 0; i < 100; i++)  
    P[i] = f(Q[i],R[i]);
```

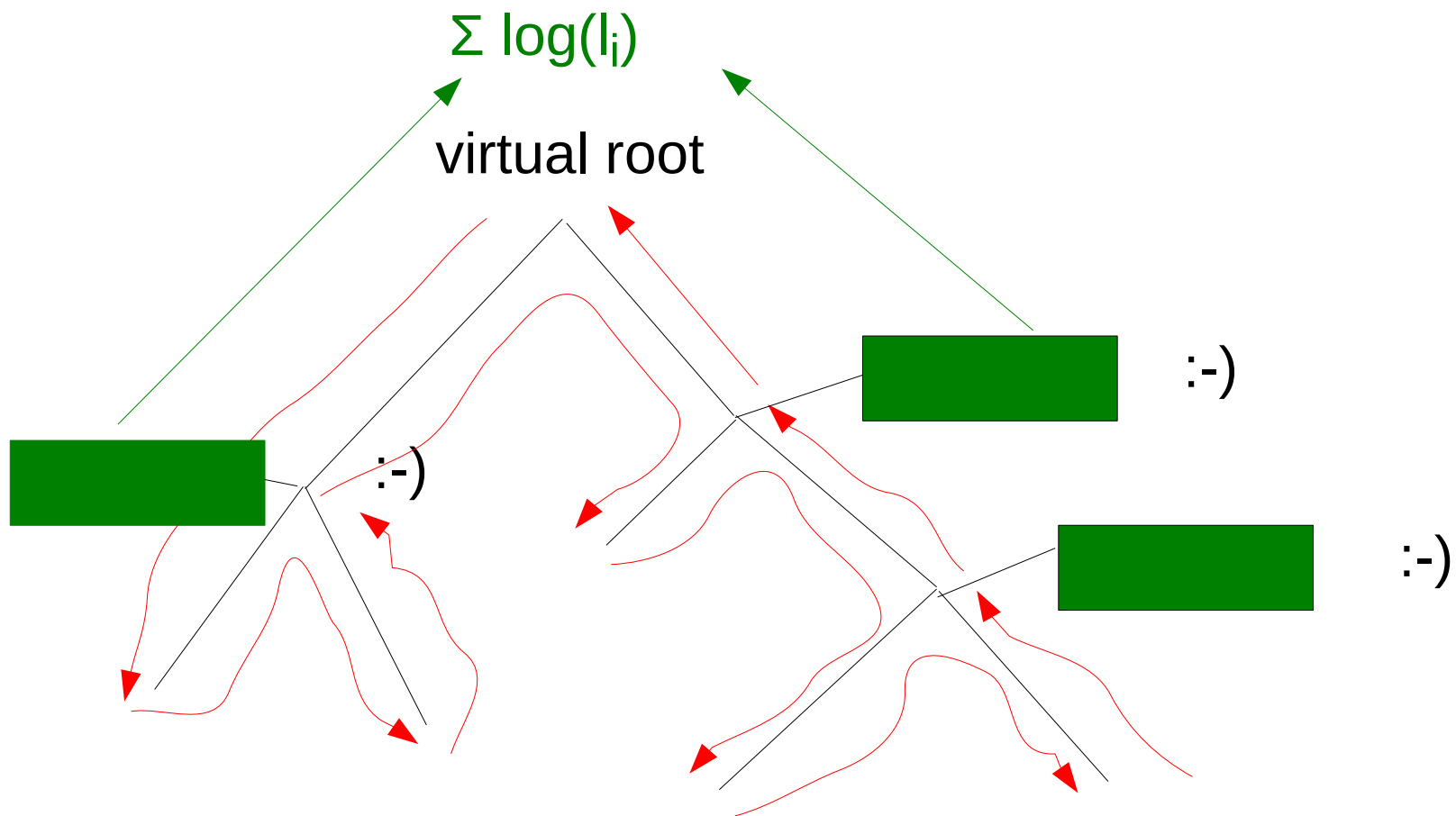
some other **sequential** code

Why is this not the best way  
to parallelize likelihood calculations?

# Thread Synchronizations in parallel RAxML-NG

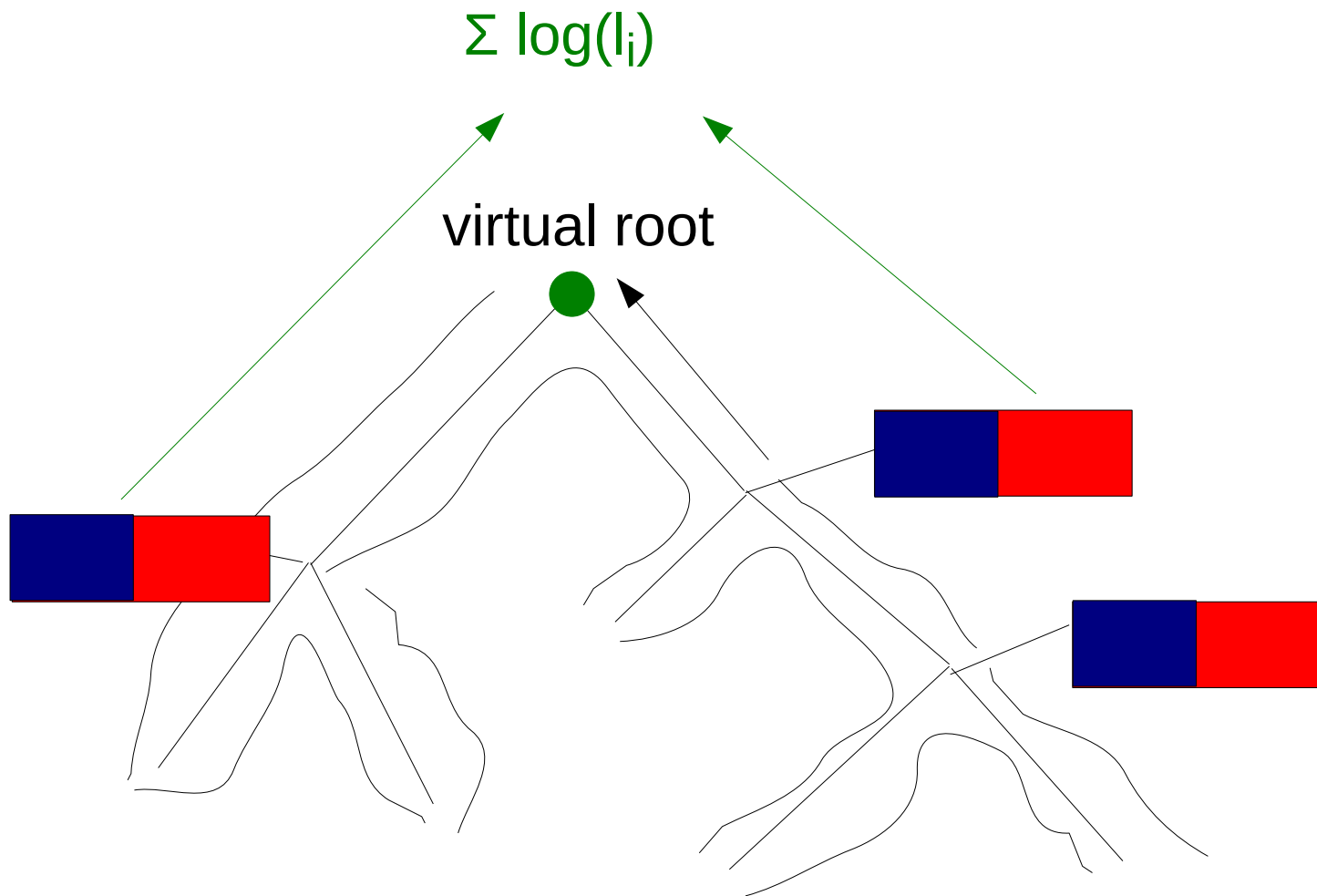
- Computing the likelihood of a single tree concurrently on many cores:
  - Snapshot: Just **10 seconds** of runtime using 16 cores/threads
  - 400 taxa, 7000 sites: **194,000** syncs
  - 1500 taxa, 1200 sites: **739,000** syncs

# Post-order Traversal

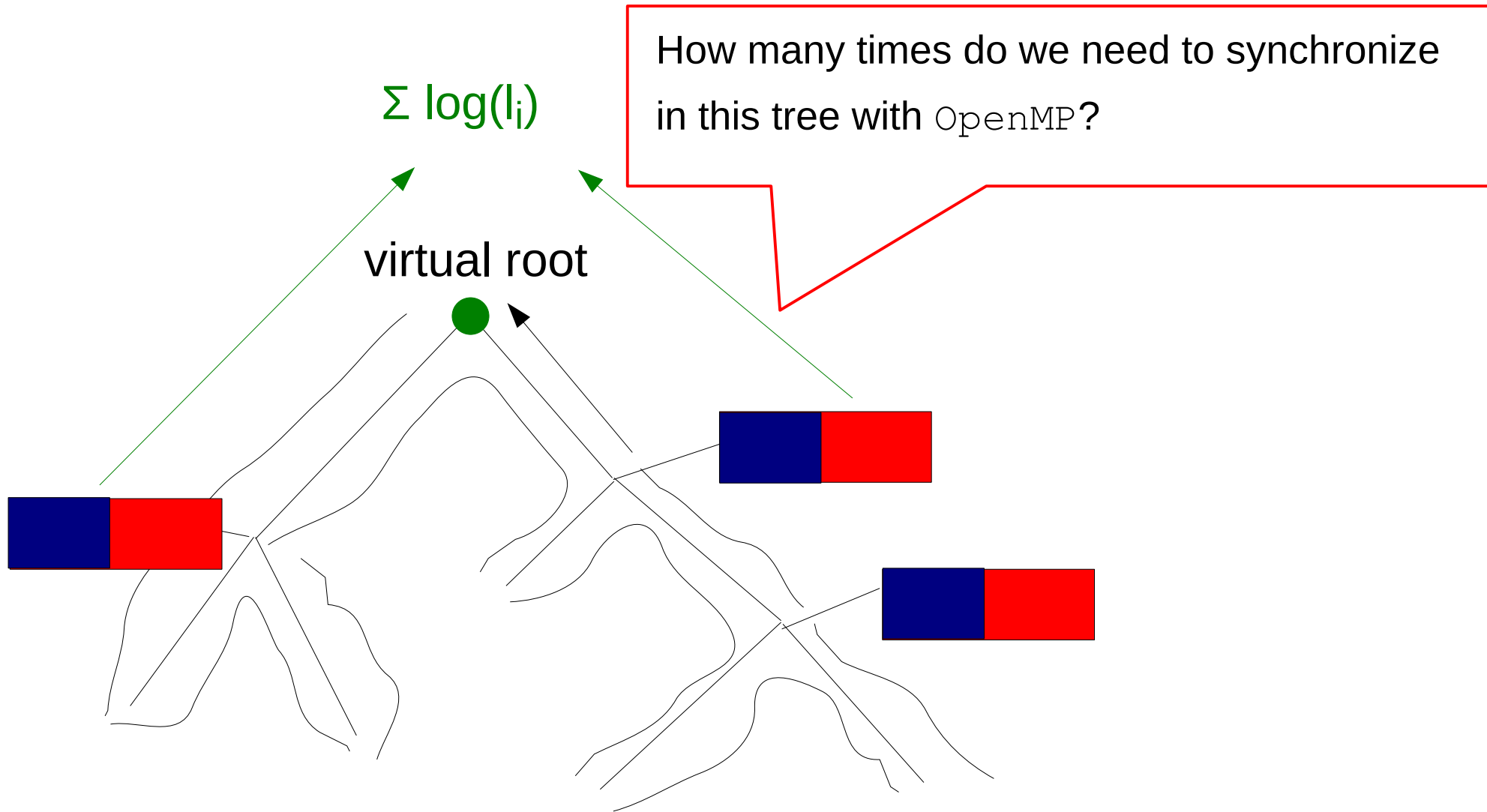




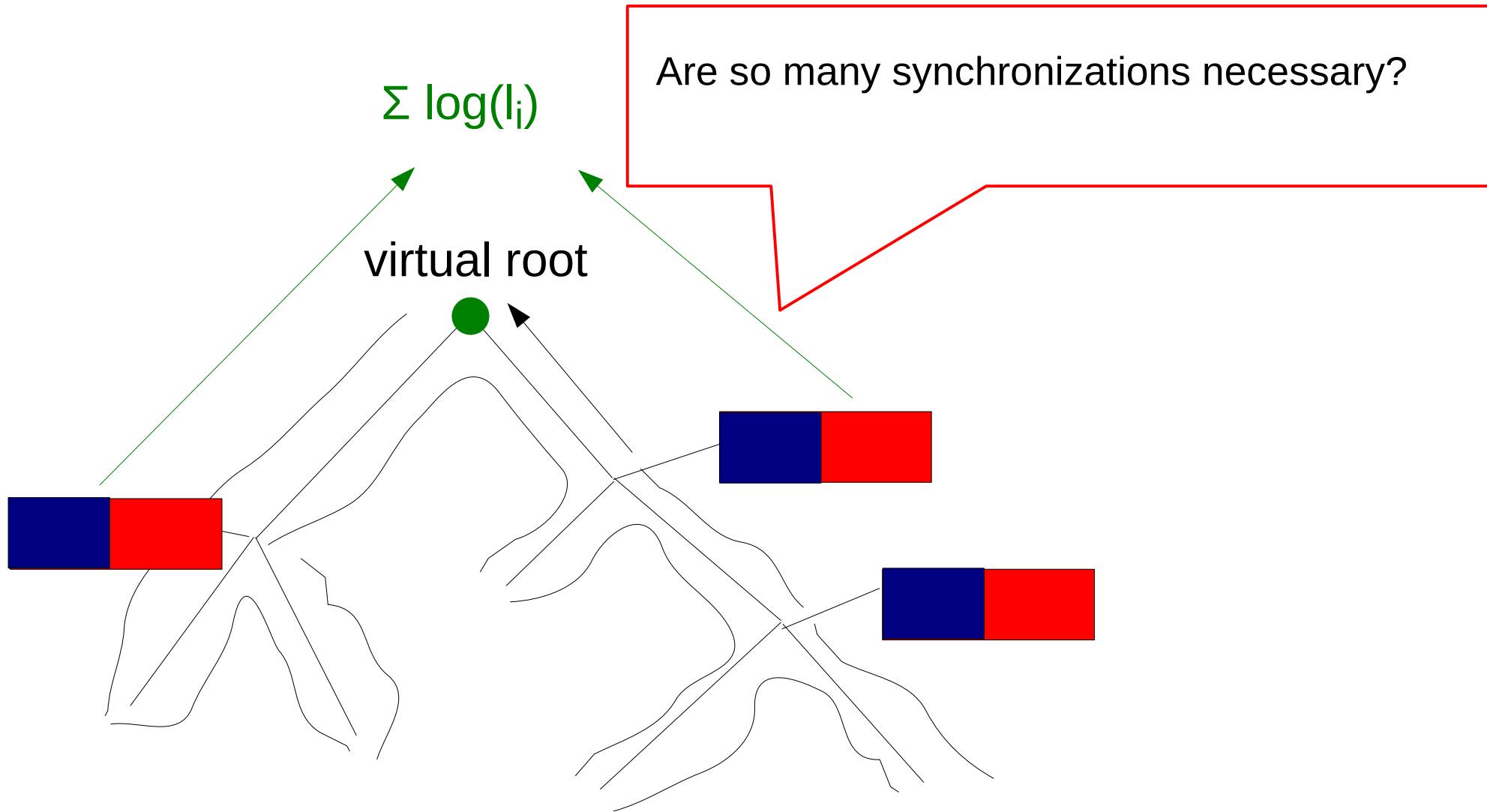
# Parallel Post-order Traversal



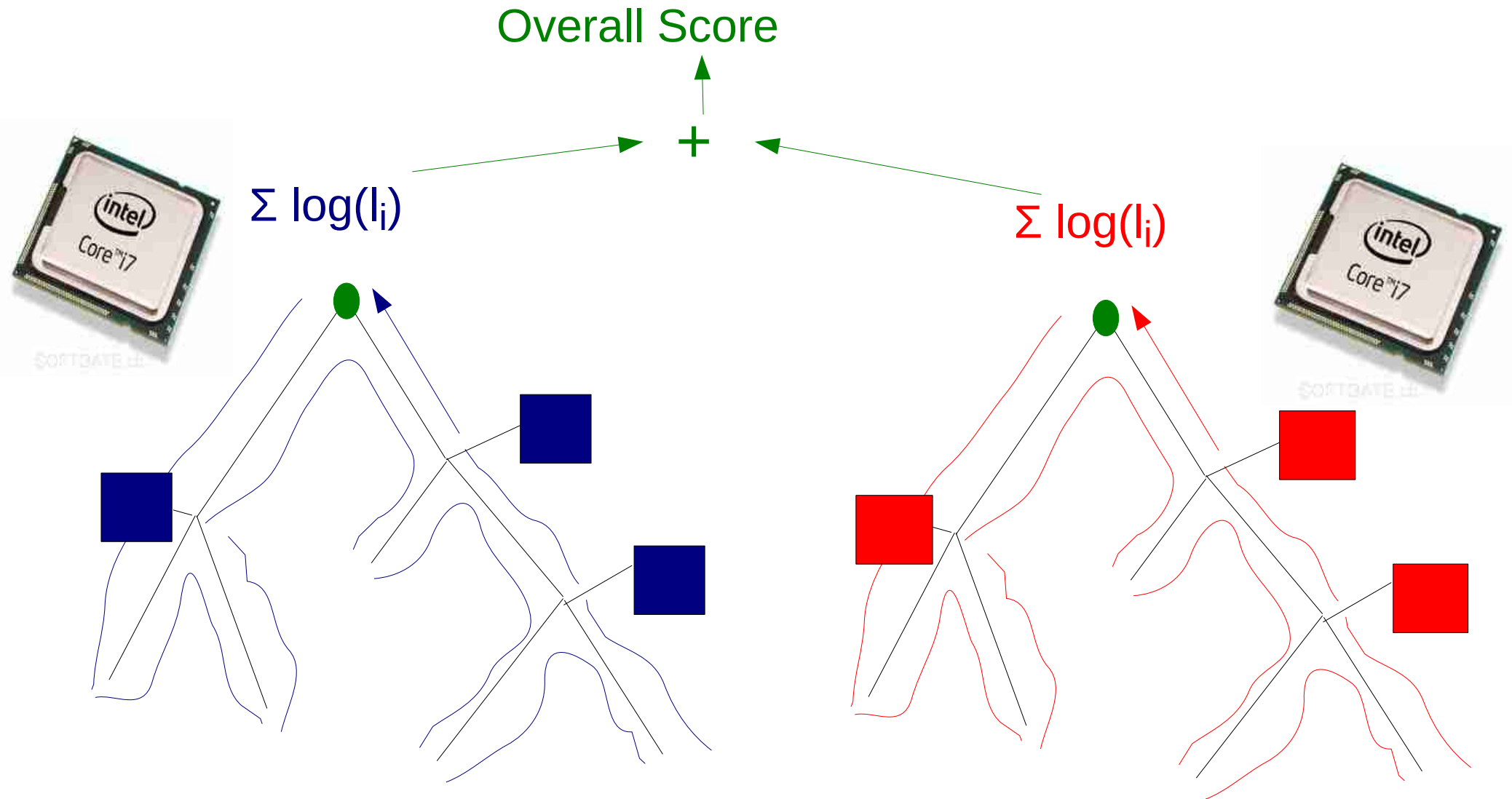
# Parallel Post-order Traversal



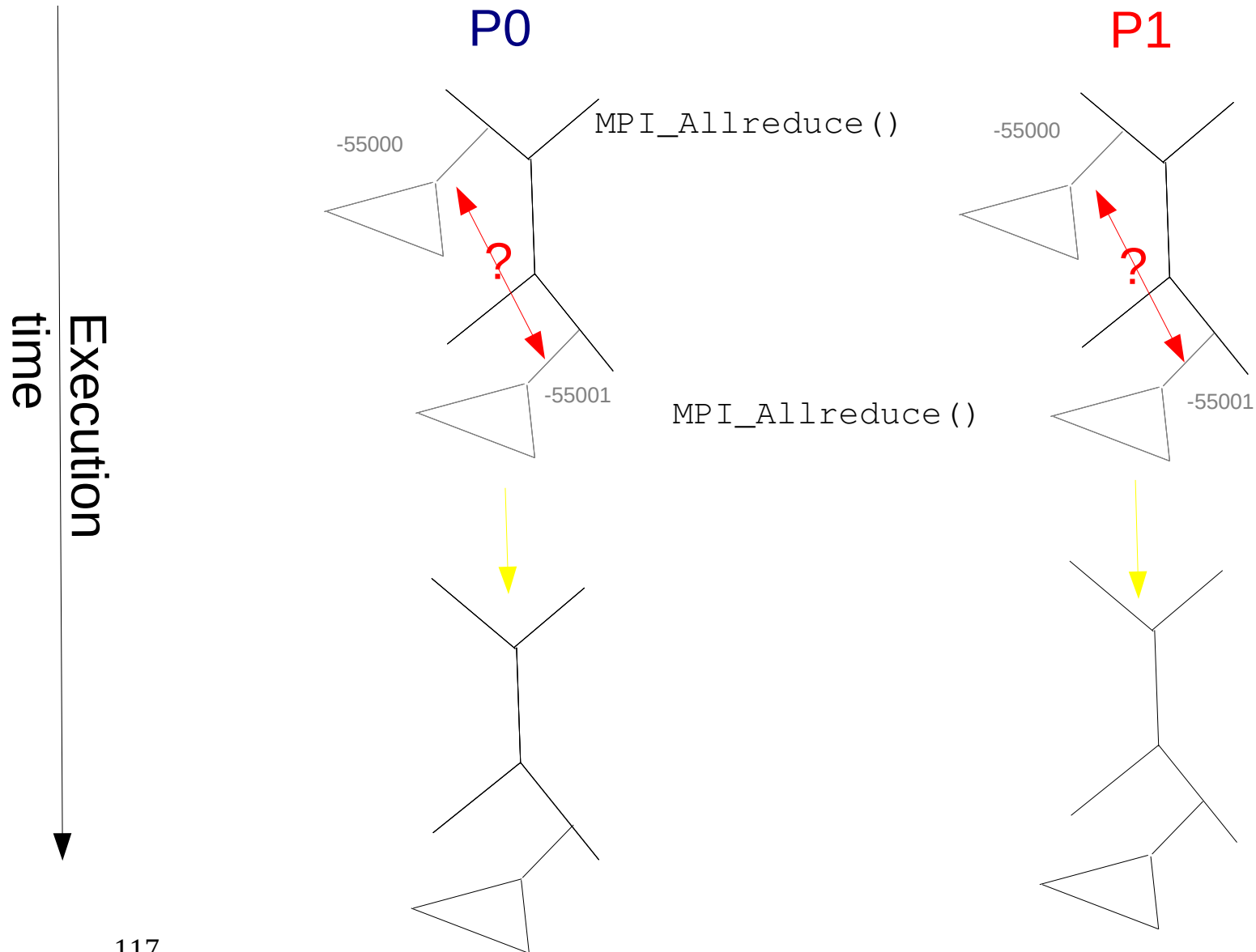
# Parallel Post-order Traversal



# Parallel Post-order Traversal



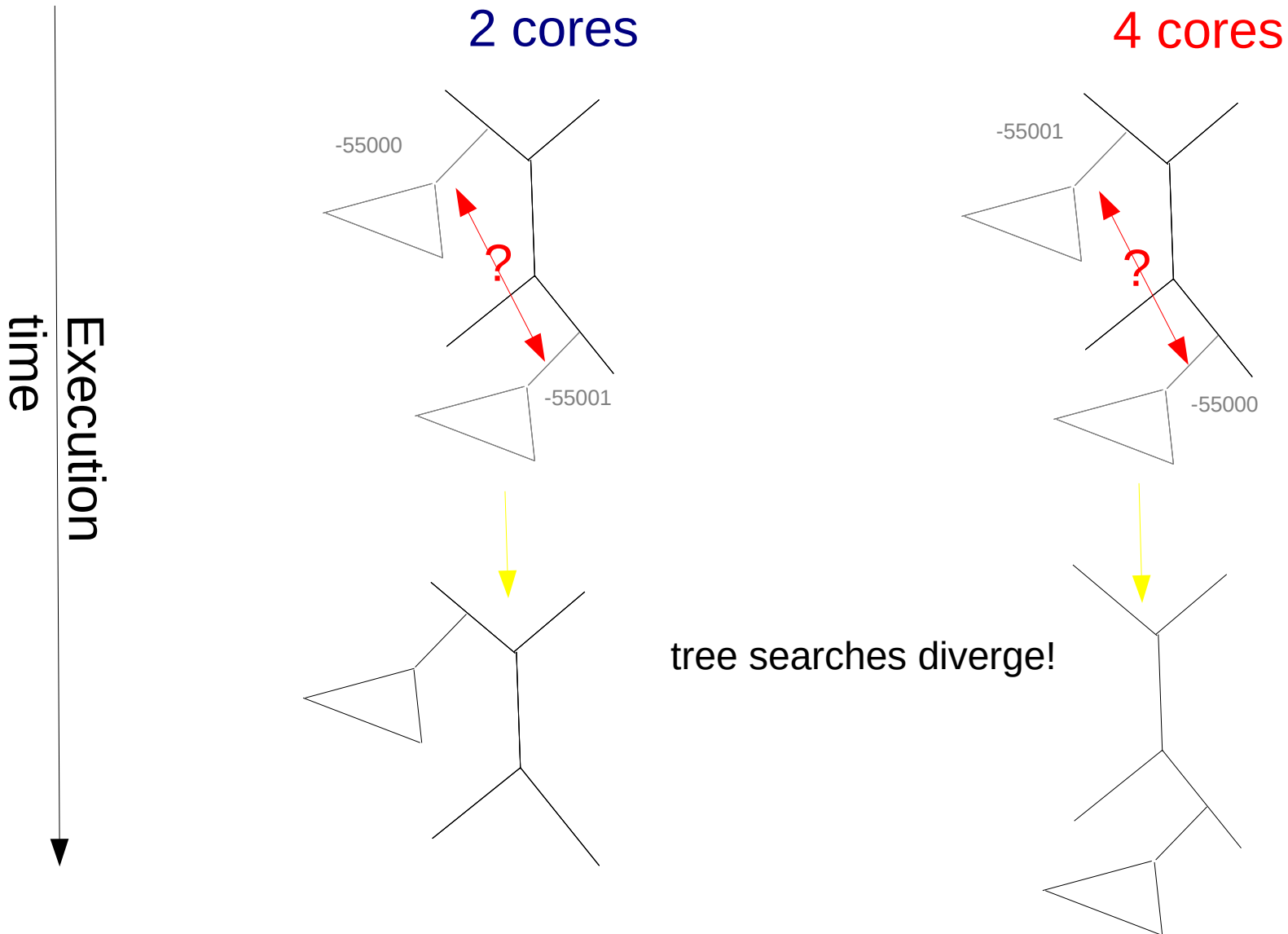
# Current MPI parallelization



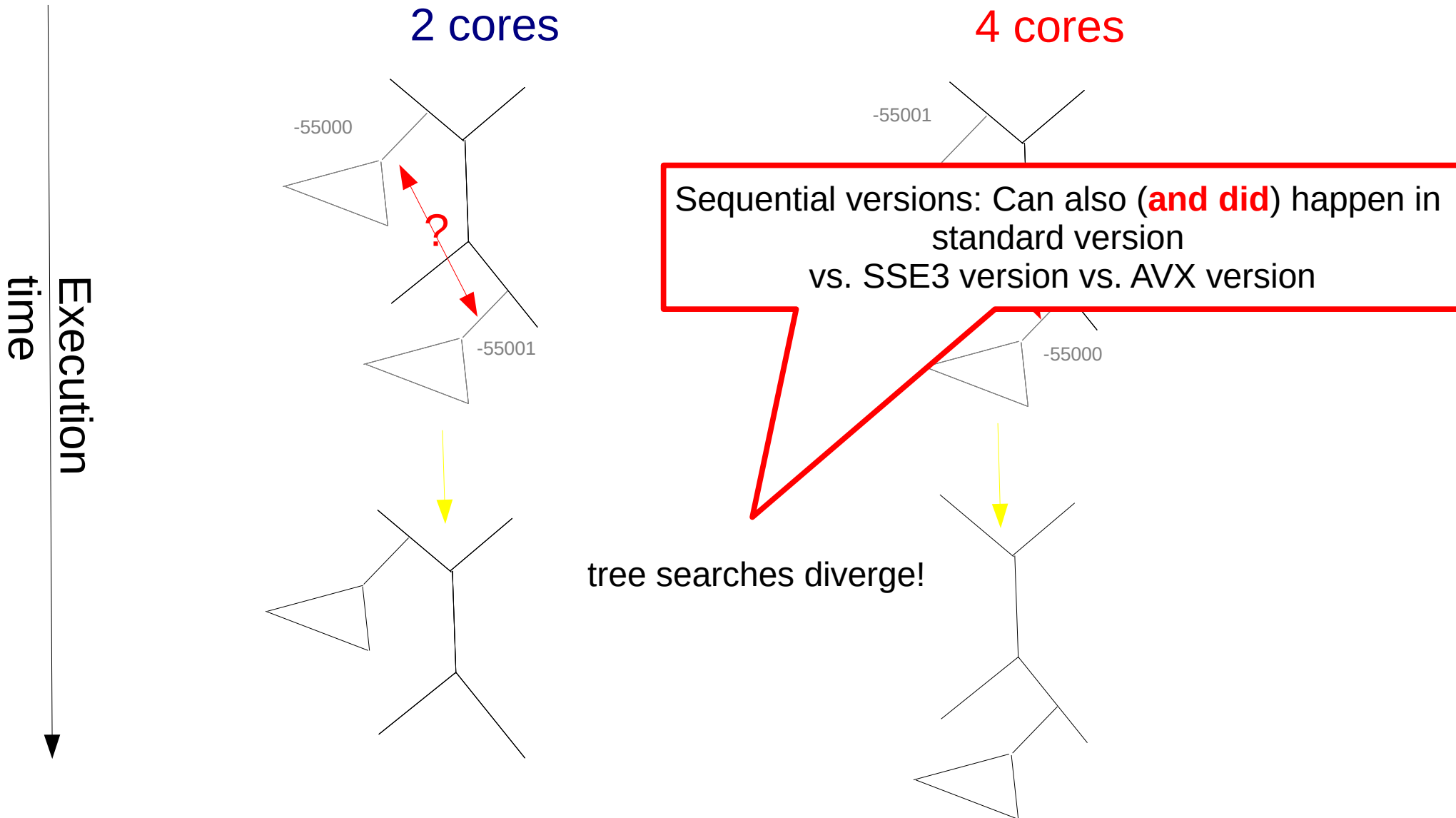
# MPI\_Allreduce ()

- **Reproducibility:** Ideally we want to get bit-wise identical results regardless of the number of cores we use → **not the case**

# Why? → distinct round off error propagation



# Why? → distinct round off error propagation

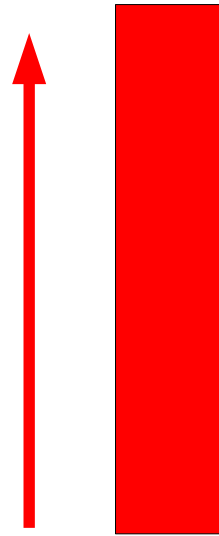




# MPI\_Allreduce ()

- **Reproducibility:** Ideally we want to get bit-wise identical results regardless of the number of cores we use
- For this we need a reproducible `MPI_Allreduce ()`
- **Christoph Stelz** "Core-Count Independent Reproducible Reduce", Bachelor thesis, Institute of Theoretical Computer Science, Karlsruhe Institute of Technology, Germany, April 2022.
- Of course there is a performance trade-off → still needs to be assessed

# Scalability depends on dataset Shapes!



Bad scalability

10,000 bp  
55,000 taxa

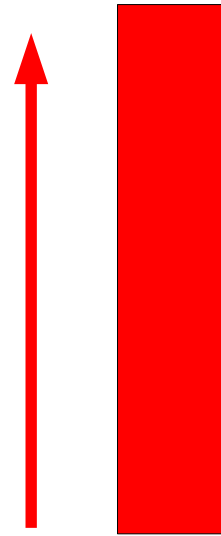
Orangutan	A A C G T T T T -
Gorilla	A A G G T T T - -
Chimp	A - G G T T T T -
Homo Sapiens	A G G A T T T T T



Good scalability

1,000,000 bp  
100 taxa

# Scalability depends on dataset Shapes!



Bad scalability

10,000 bp  
55,000 taxa

Rule of thumb:  
 $\geq 1000$  DNA sites per  
core for good scalability

Orangutan	A A C G T T T T -
Gorilla	A A G G T T T - -
Chimp	A - G G T T T T -
Homo Sapiens	A G G A T T T T T




Good scalability

1,000,000 bp  
100 taxa

# User friendly Parallelism: RAxML-NG

```
Analysis options:  
run mode: ML tree search  
start tree(s): random (10) + parsimony (10)  
random seed: 1657272853  
tip-inner: OFF  
pattern compression: ON  
per-rate scalars: OFF  
site repeats: ON  
fast spr radius: AUTO  
spr subtree cutoff: 1.000000  
branch lengths: proportional (ML estimate, algorithm: NR-FAST)  
SIMD kernels: AVX2  
parallelization: coarse-grained (auto), PTHREADS (auto)
```



- Will automatically chose the best parallelization strategy depending on alignment (MSA) length

# Likelihood Parallelization

## Load Balance

- It's not that easy for partitioned datasets
  - “The Multi-Processor Scheduling Problem in Phylogenetics”, 2012
  - “The divisible load balance problem and its application to phylogenetic inference”, 2014
- It's not that easy if the computation cost for the likelihood of a site varies among sites
  - “The divisible load balance problem with shared cost and its application to phylogenetic inference”, 2015
  - “A novel heuristic for data distribution in massively parallel phylogenetic inference using site repeats”, 2018
  - “Data Distribution for Phylogenetic Inference with Site Repeats via Judicious Hypergraph Partitioning”, 2019

# Likelihood Parallelization

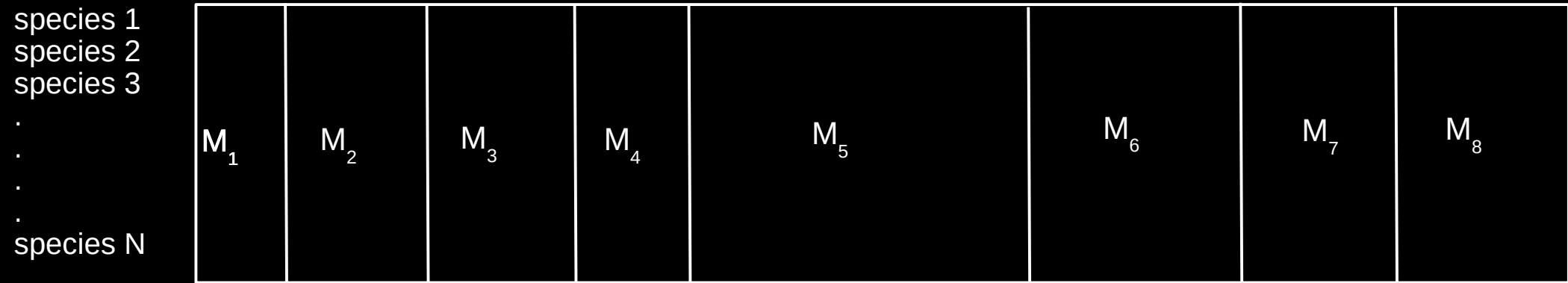
## Load Balancing

Essentially solved via approximation algorithm with very tight bound – not RAxML-specific

- It's not that easy for partitioned datasets
  - “The Multi-Processor Scheduling Problem in Phylogenetics”, 2012
  - “The divisible load balance problem and its application to phylogenetic inference”, 2014
- It's not that easy if the computation cost for the likelihood of a site varies among sites
  - “The divisible load balance problem with shared cost and its application to phylogenetic inference”, 2015
  - “A novel heuristic for data distribution in massively parallel phylogenetic inference using site repeats”, 2018
  - “Data Distribution for Phylogenetic Inference with Site Repeats via Judicious Hypergraph Partitioning”, 2019

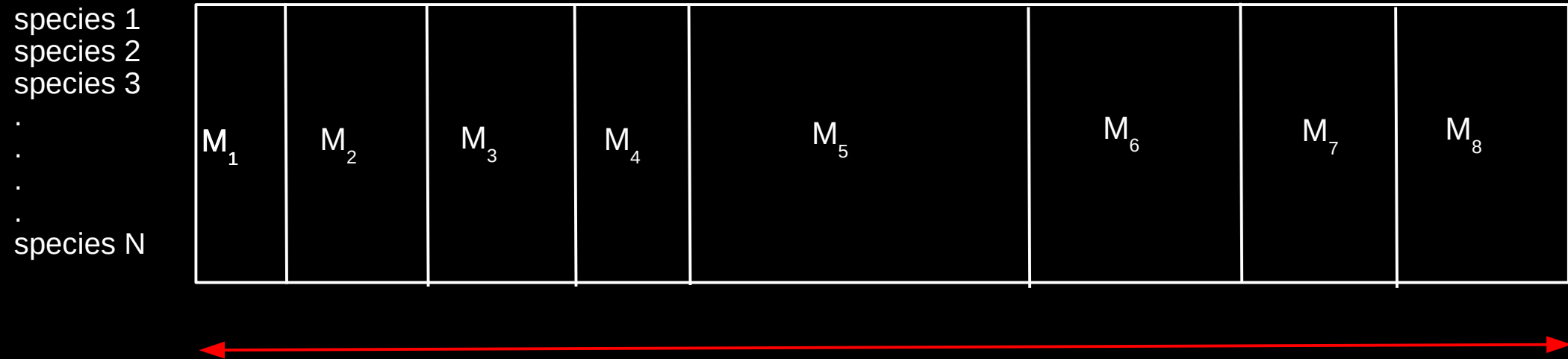
3 failed attempts!

# Load Balance



We parallelize over alignment sites/columns

# Load Balance



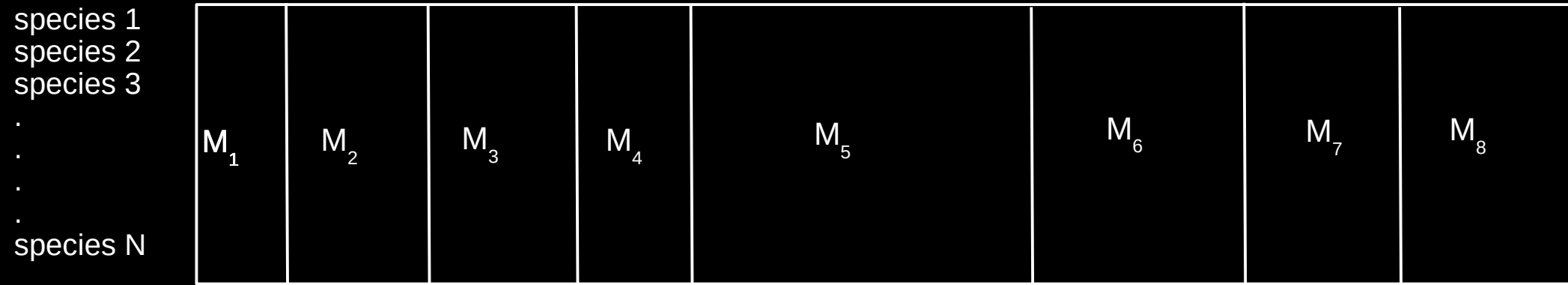
We parallelize over alignment sites/columns

Key assumptions:

- sites can be computed independently!
- all sites have the same computational cost!

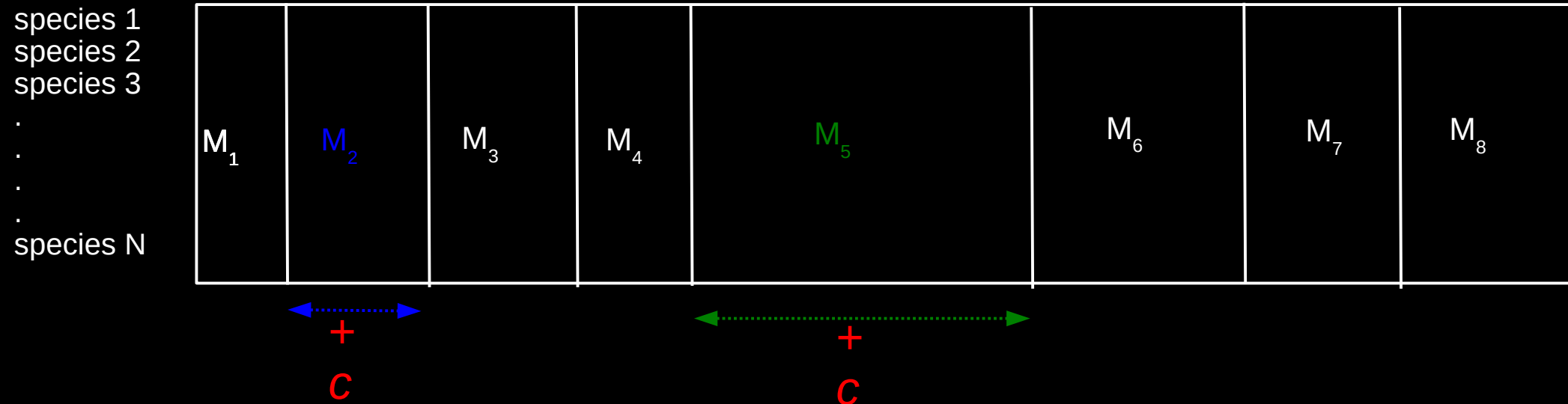


# Load Balance



→ Every gene evolves according to an independent model  $M_i$

# Load Balance



- Every gene evolves according to an independent model  $M_i$
- Computing time per model is proportional to the number of sites
- But, every partition/gene has a constant 'start-up' time  $C$
- $C$  is the time for calculating  $P(t)=e^{Q_i t}$

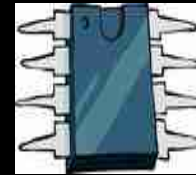
# Load Balance

- Optimization problem
  - Distribute sites to  $p$  processors such that:
    1. All processors have the same #sites
    2. The number of accumulated constant calculations  $C$  (i.e., # of partitions) per processor is minimized!
  - Sites of a partition with model  $M_i$  can be distributed across several processors → in such a case the cost  $C$  for model  $M_i$  has to be payed at every processor

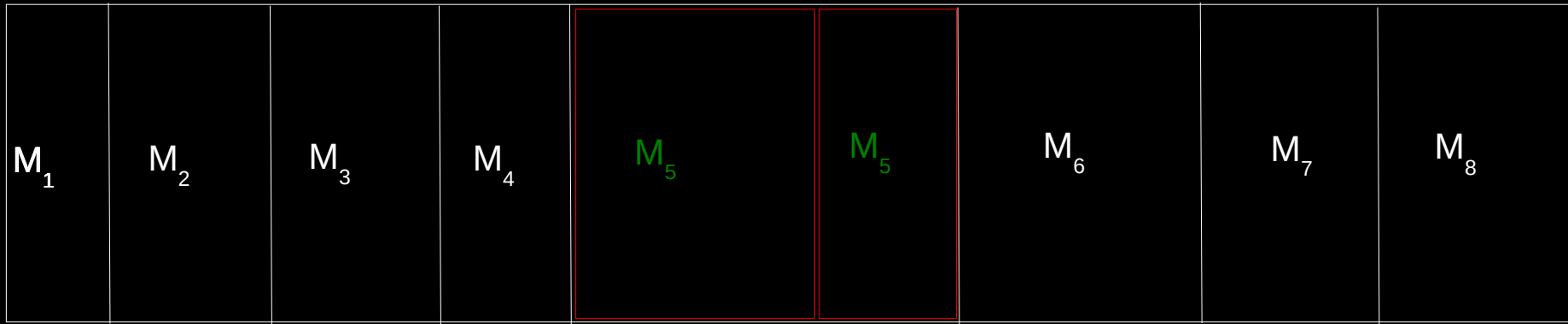
# Load Balance

CPU 0

CPU 1



species 1  
species 2  
species 3  
.  
.  
.  
species N



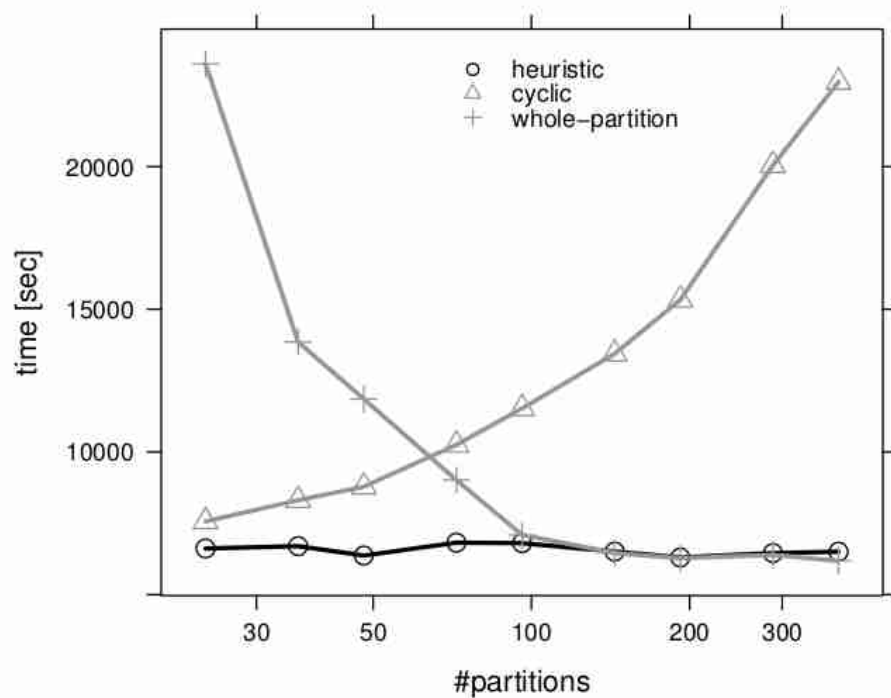
+  
C

+  
C

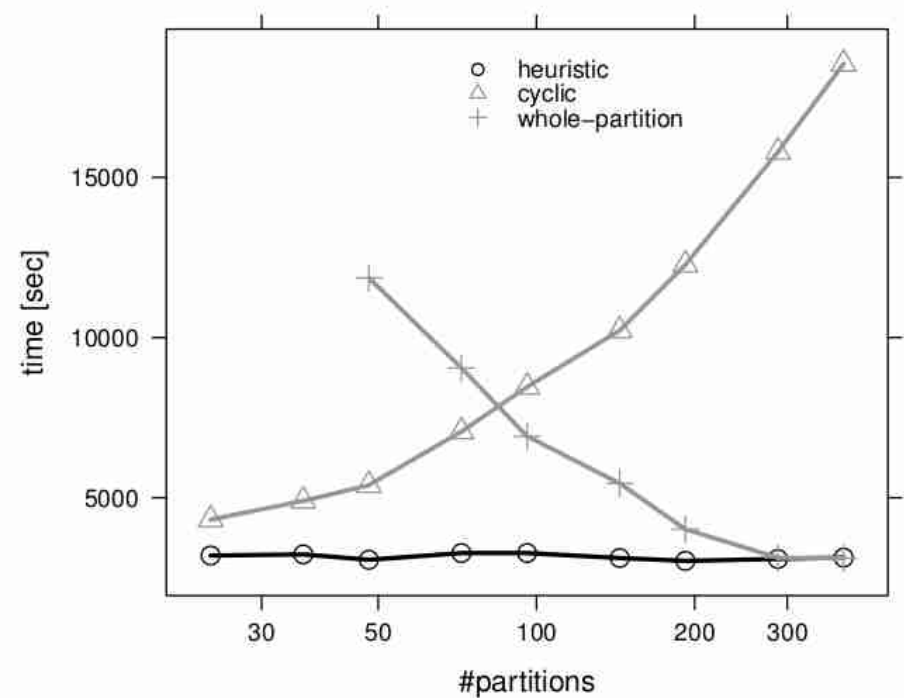
# Solution

- Finding the optimal solution is NP-hard
- Approximation algorithm that it is only **one off** → we proved that CPUs will do at most one more 'start-up' calculation with cost **C** than for the optimal solution
- Using this algorithm actually improves performance
  - theory meets practice

# Results for ExaML



(a) Runtimes on 24 cores.



(b) Runtimes on 48 cores.

**Fig. 4.** Runtime comparison for ExaML employing algorithm `LOADBALANCE`, the cyclic data distribution scheme, or the whole-partition data distribution scheme.

# Advertisement Section

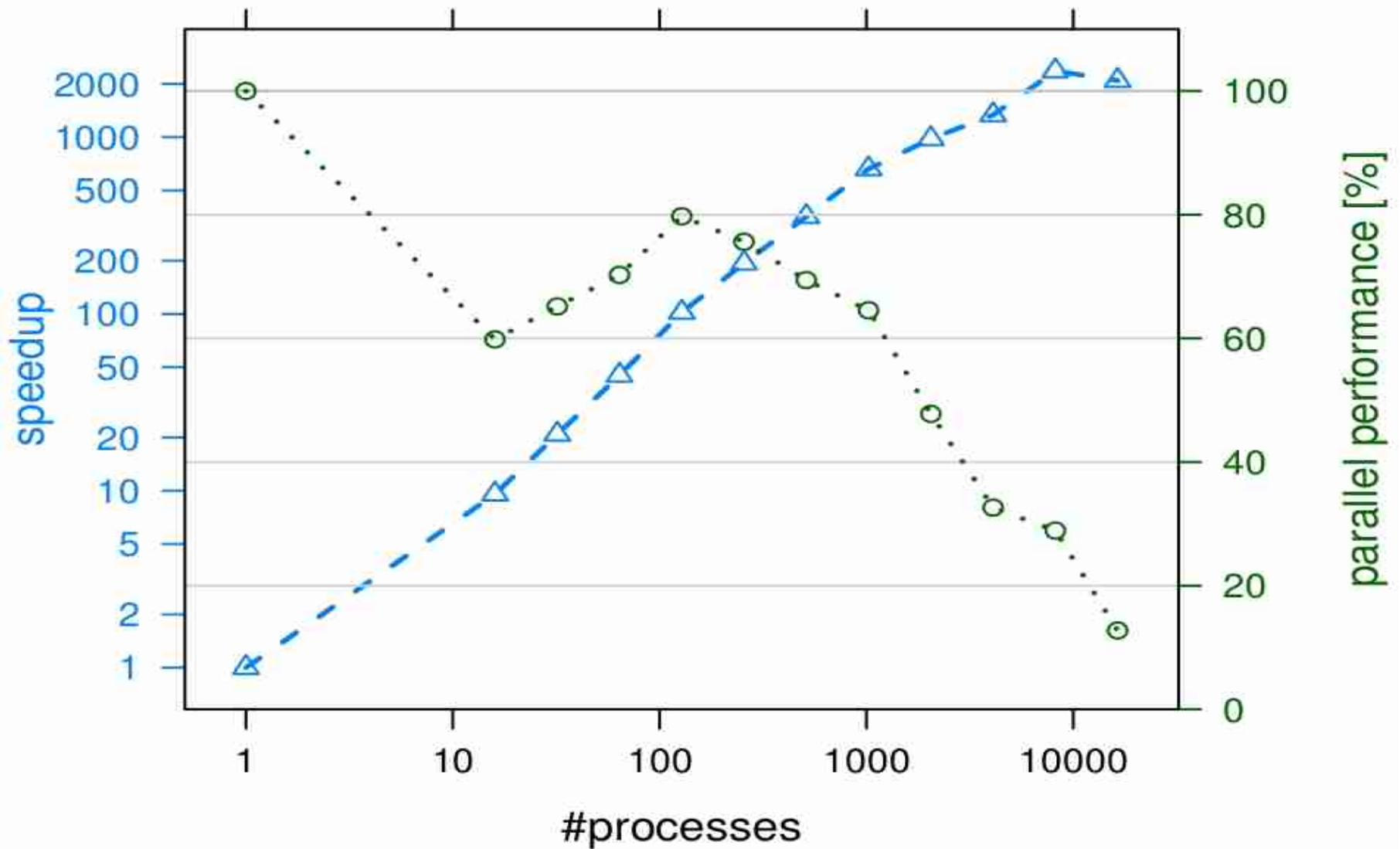
Some of our tools ....

# ExaBayes

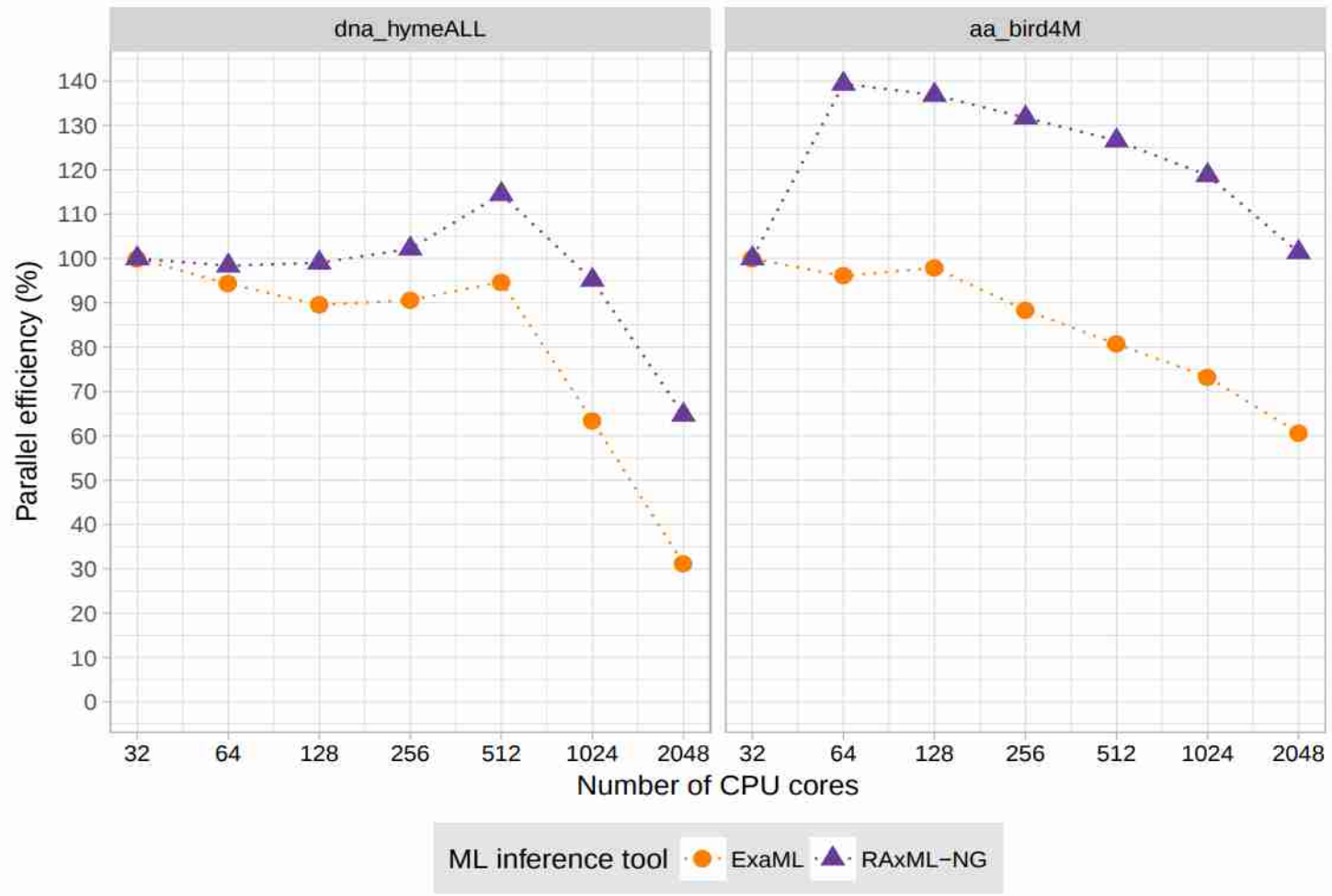
- **Similar to MrBayes**
  - Speed
    - On DNA datasets it's approx. 30% faster
  - Convergence/quality
- Better parallel scalability
- Executed on dataset with 200 taxa & 100,000,000 sites on 32,000 cores of the Munich Supercomputer



# ExaBayes Scalability

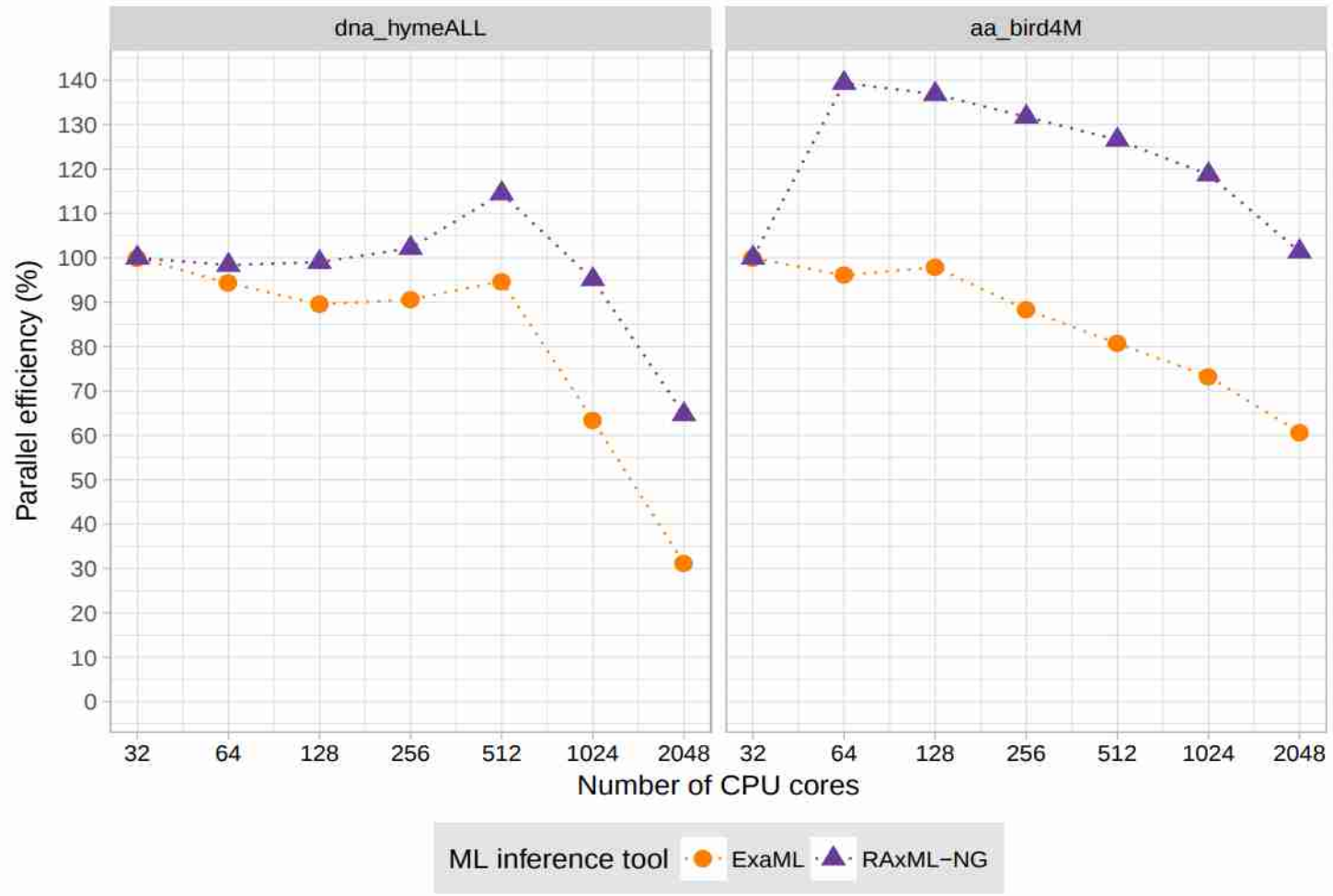


# RAxML-NG

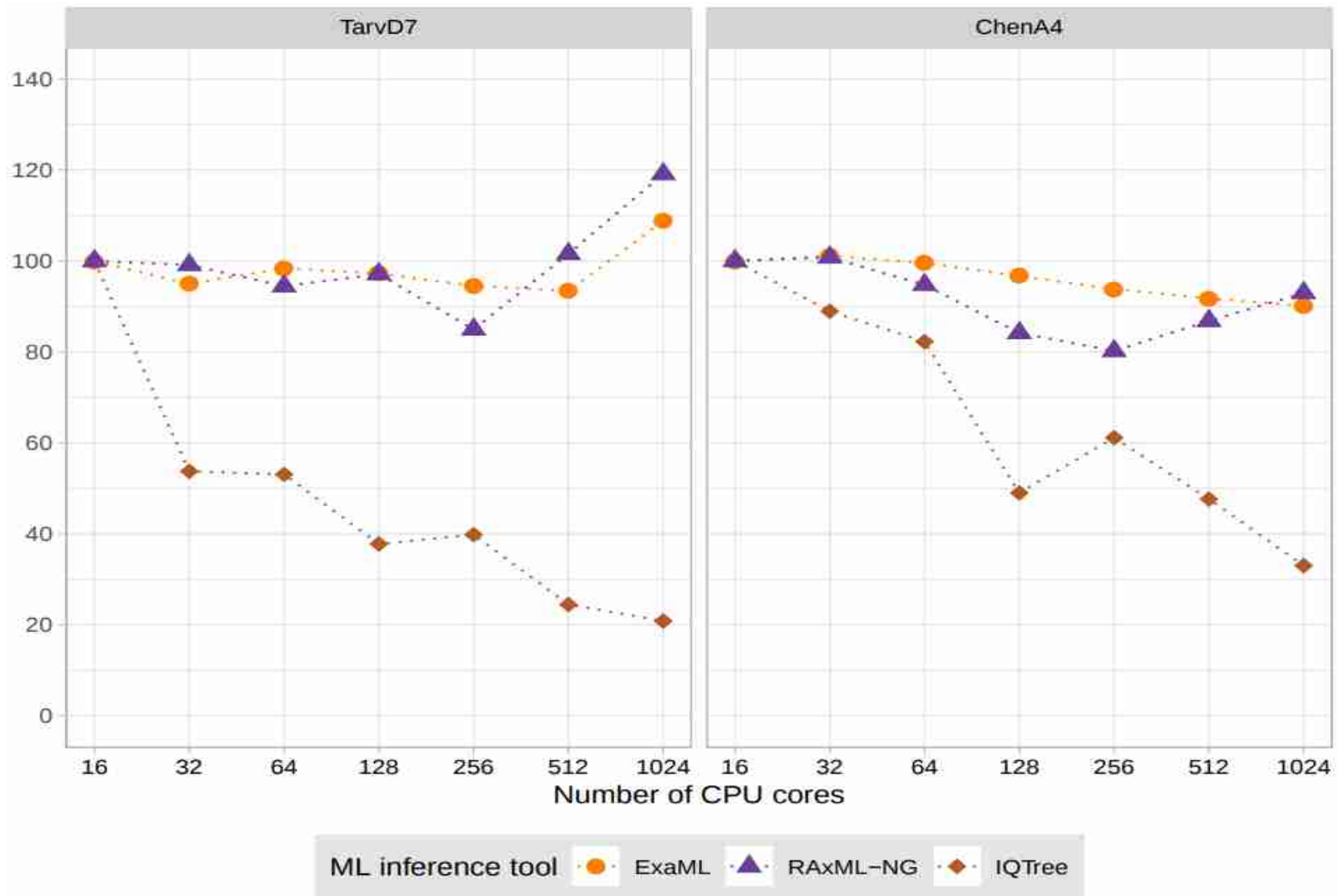


# RAxML-NG

What is suspicious about these plots?



# RAxML-NG VS. IQ-Tree



# RAXML-NG **VS.** IQ-Tree

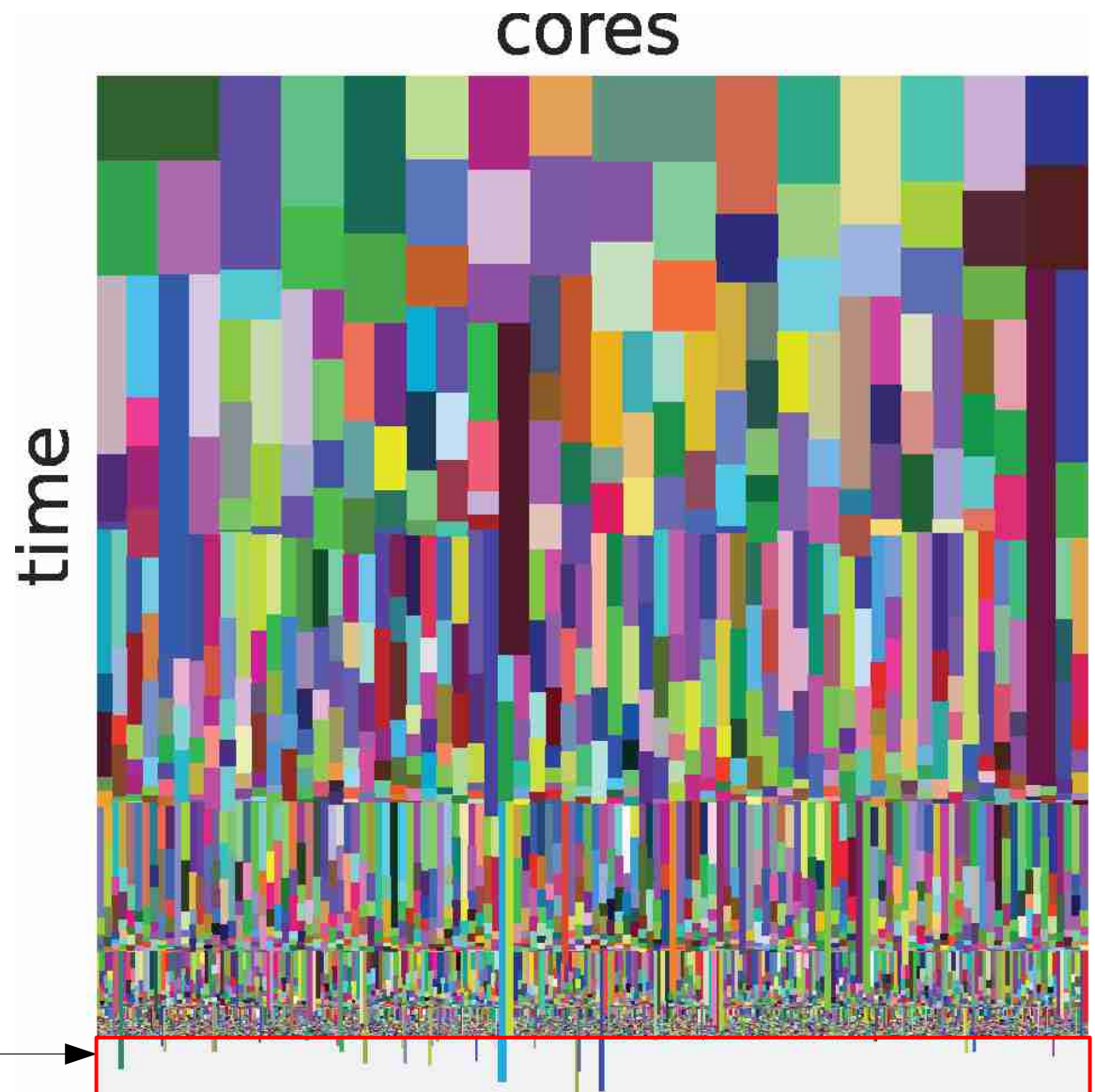
- RAXML - NG found best scoring tree most often (19 out of 21 datasets)
- 1.3 – 4.5 times faster
- Parallel efficiency of up to 125% !!!!!
- RAXML - NG is generally faster & returns higher scoring trees on taxo-rich MSAs, IQ-Tree results exhibit lower variance (if you do multiple tree searches)
- For MSAs with strong phylogenetic signal, IQ-Tree may require fewer searches than RAXML - NG
- RAXML - NG: A fast, scalable, and user-friendly tool for maximum likelihood phylogenetic inference, *Bioinformatics*, May 2019
- By the way, the IQ-Tree guys are our friends :-)

# Inferring Gene Trees

- Often, we want to infer gene trees on thousands of genes to generate input for so-called gene tree/species tree reconciliation methods → input for `ASTRAL`, `SpeciesRax`, or `GeneRax`
- How do we efficiently orchestrate such computations on a cluster?

# ParGenes Tool

- ParGenes : a tool for massively parallel model selection and phylogenetic tree inference on thousands of genes *Bioinformatics* 2019
- A classic scheduling problem!



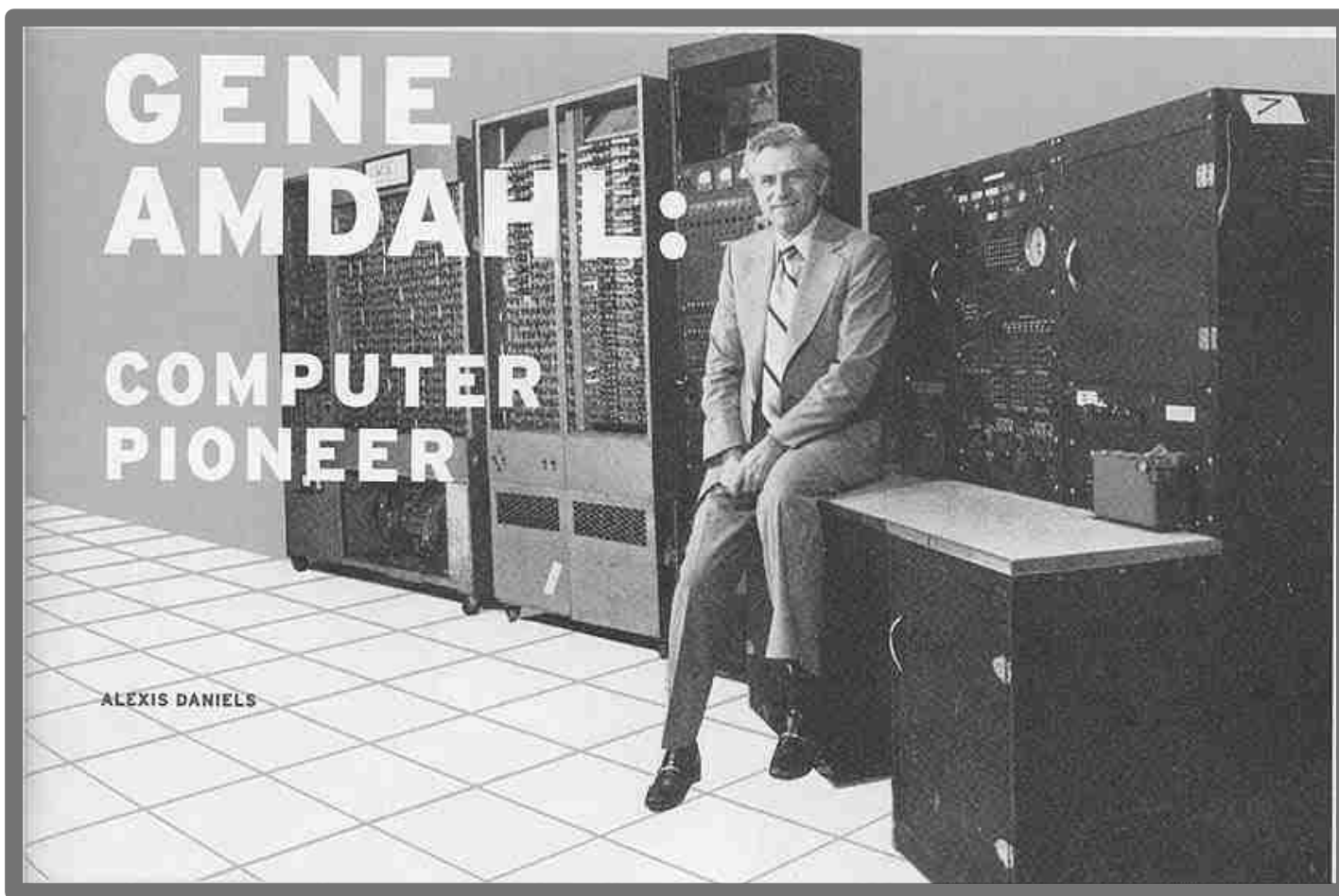
Cores not fully utilized

# Outline

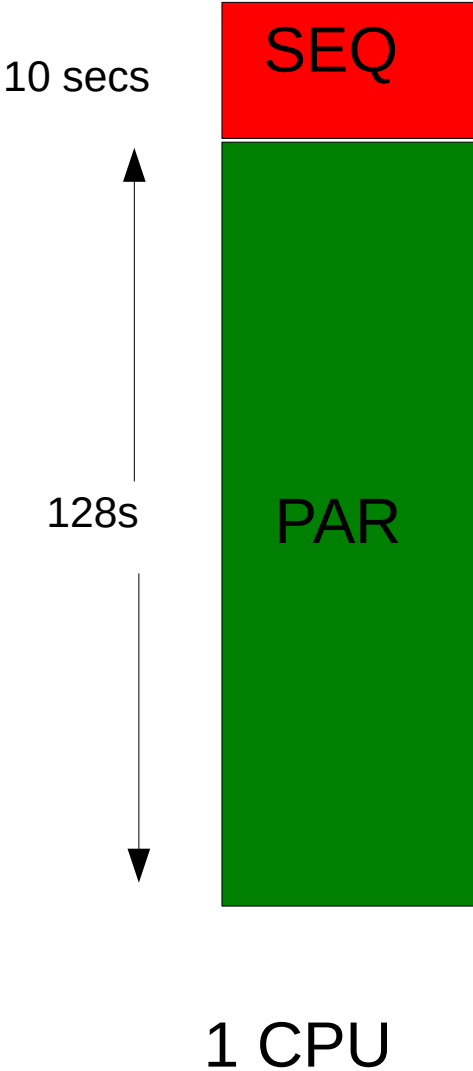
- Maximum Likelihood (Recap)
- Sequential Optimization
- Parallelization
- **Parallel I/O**
- Numerical Nightmares
- Energy Efficiency



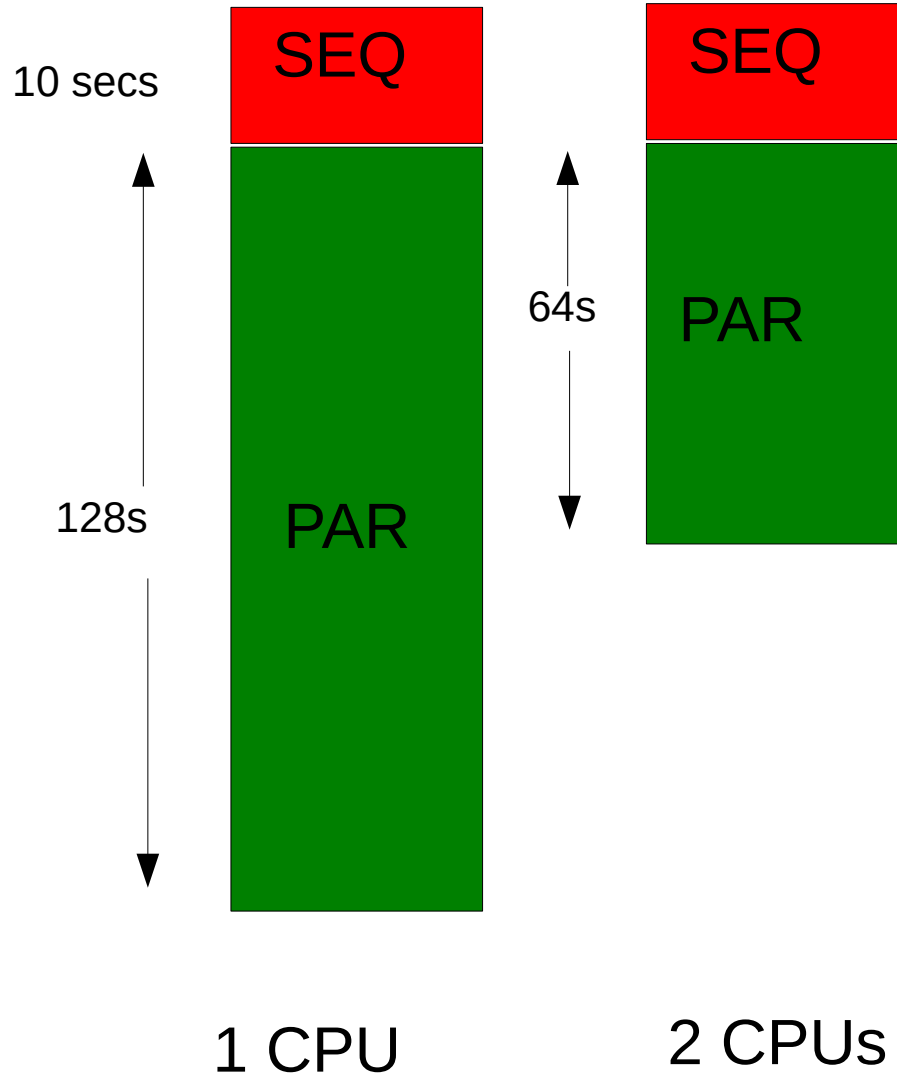
# Gene Amdahl



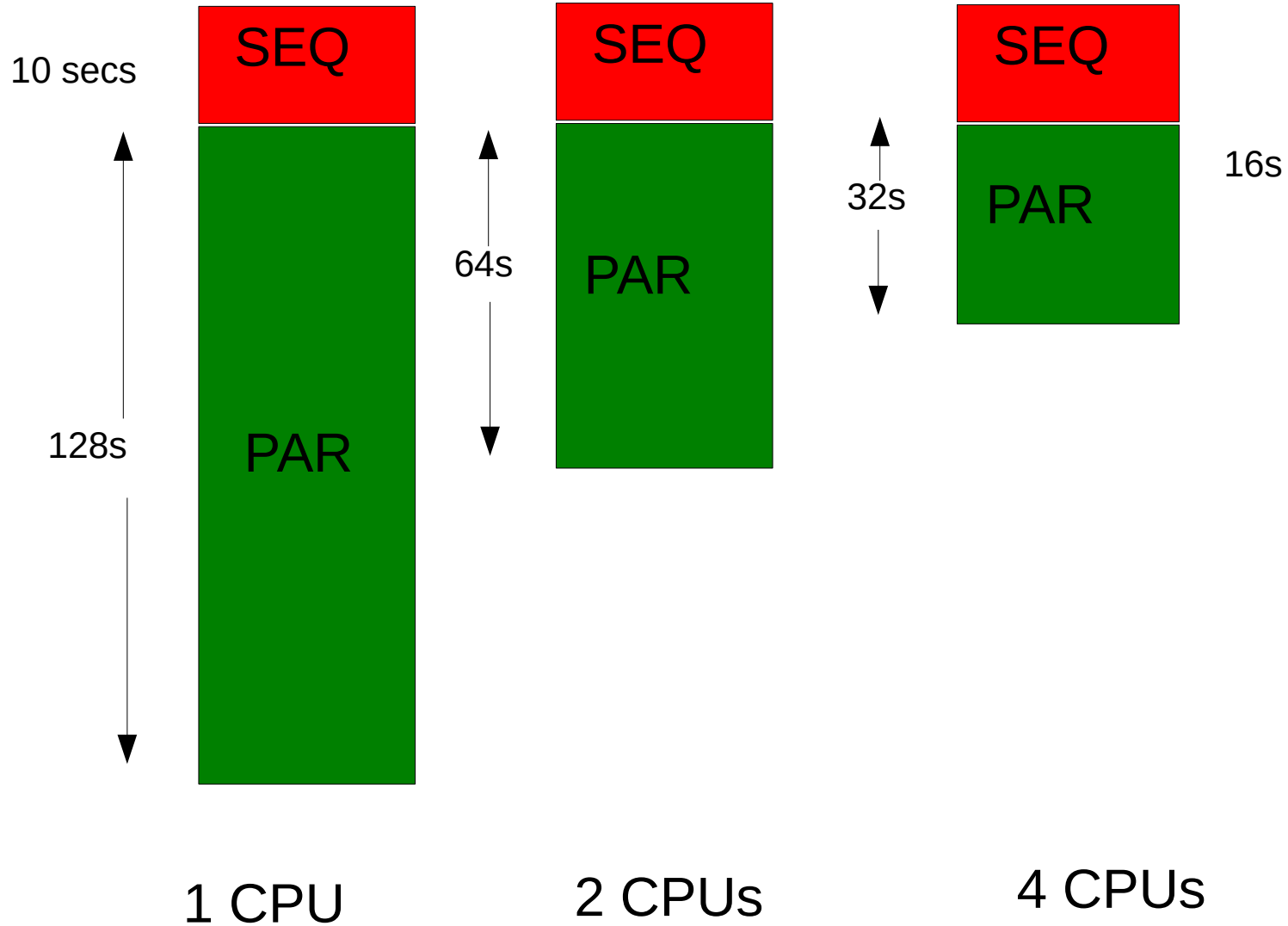
# Amdahl's Law



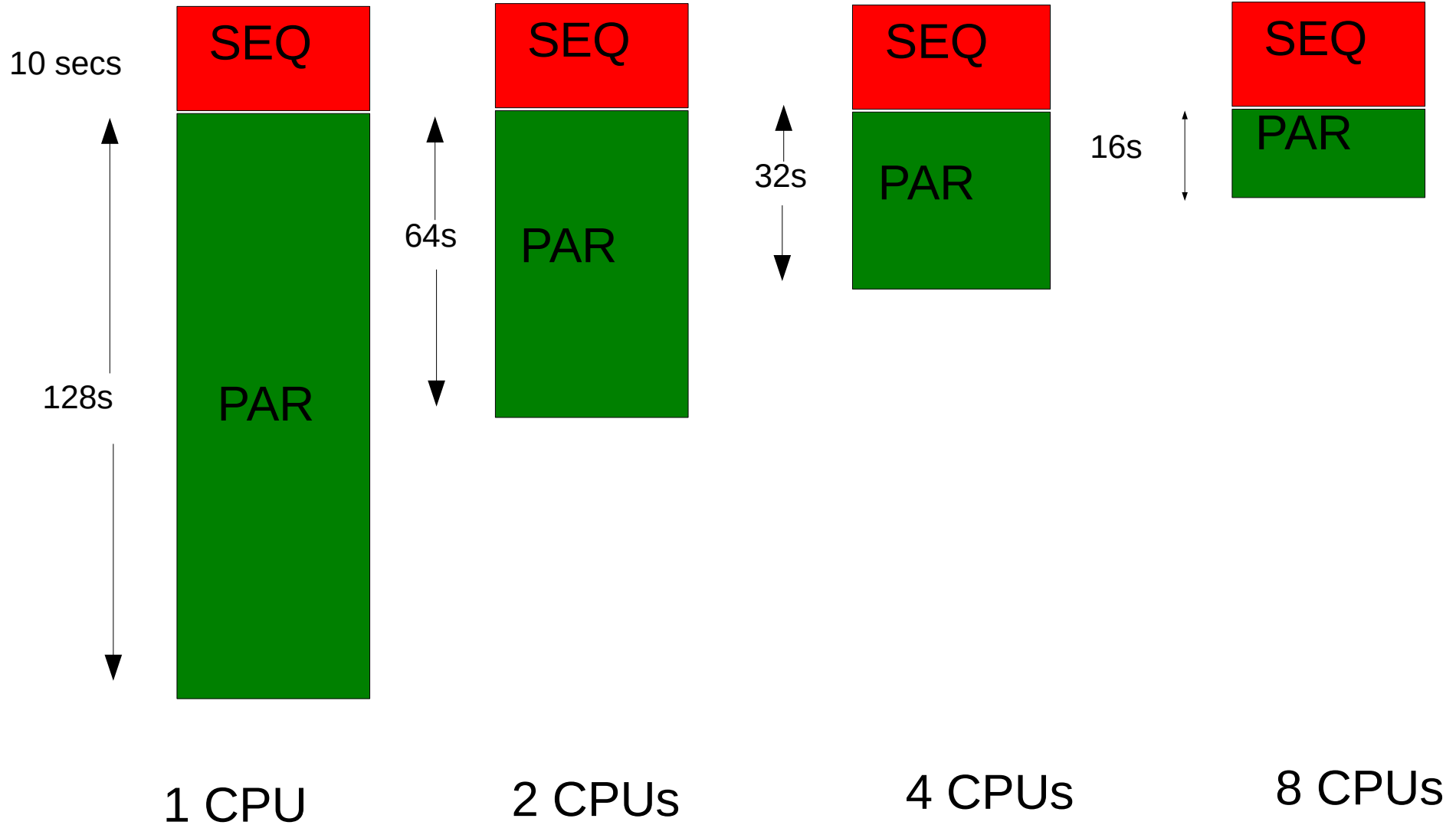
# Amdahl's Law



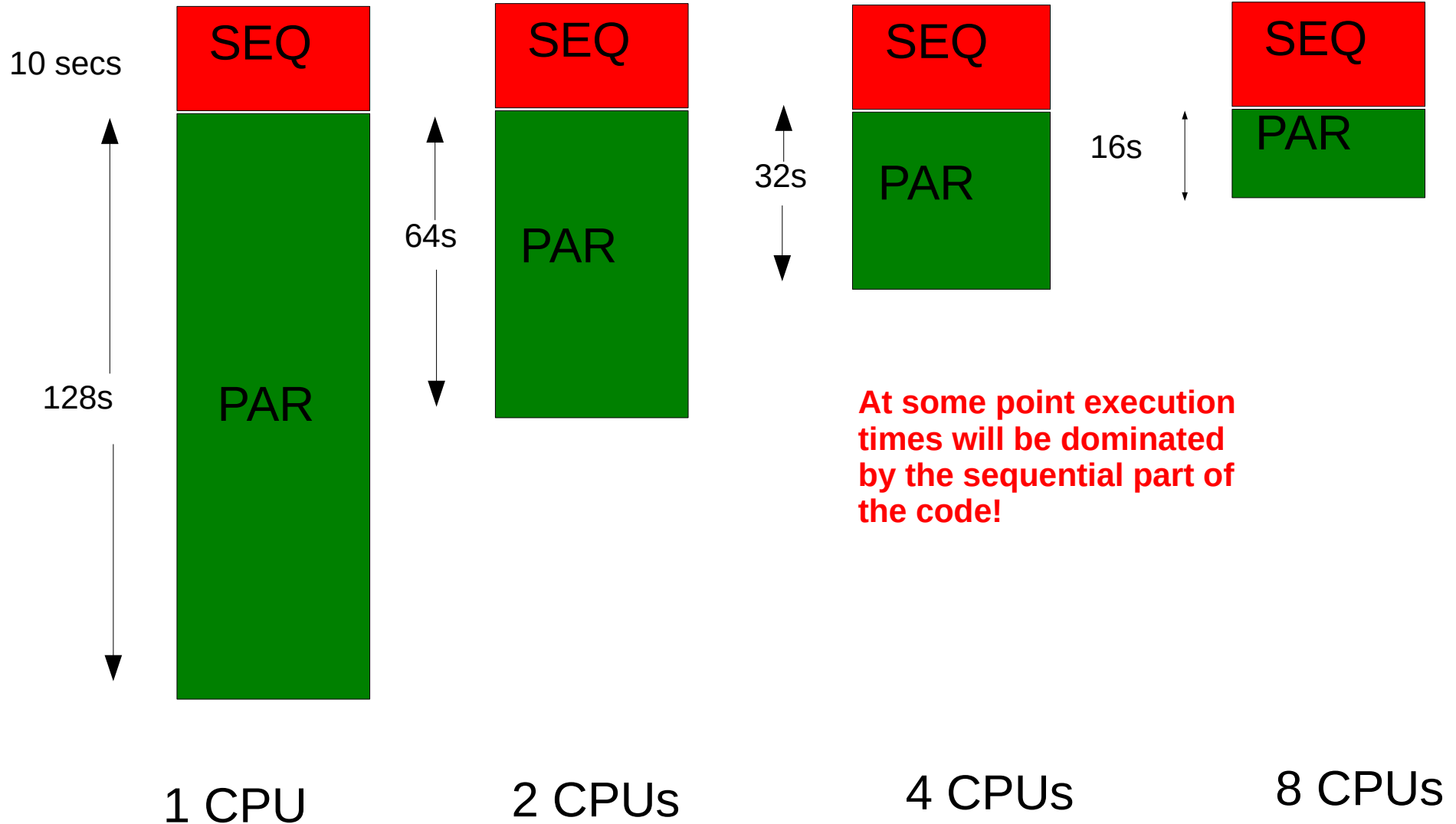
# Amdahl's Law



# Amdahl's Law

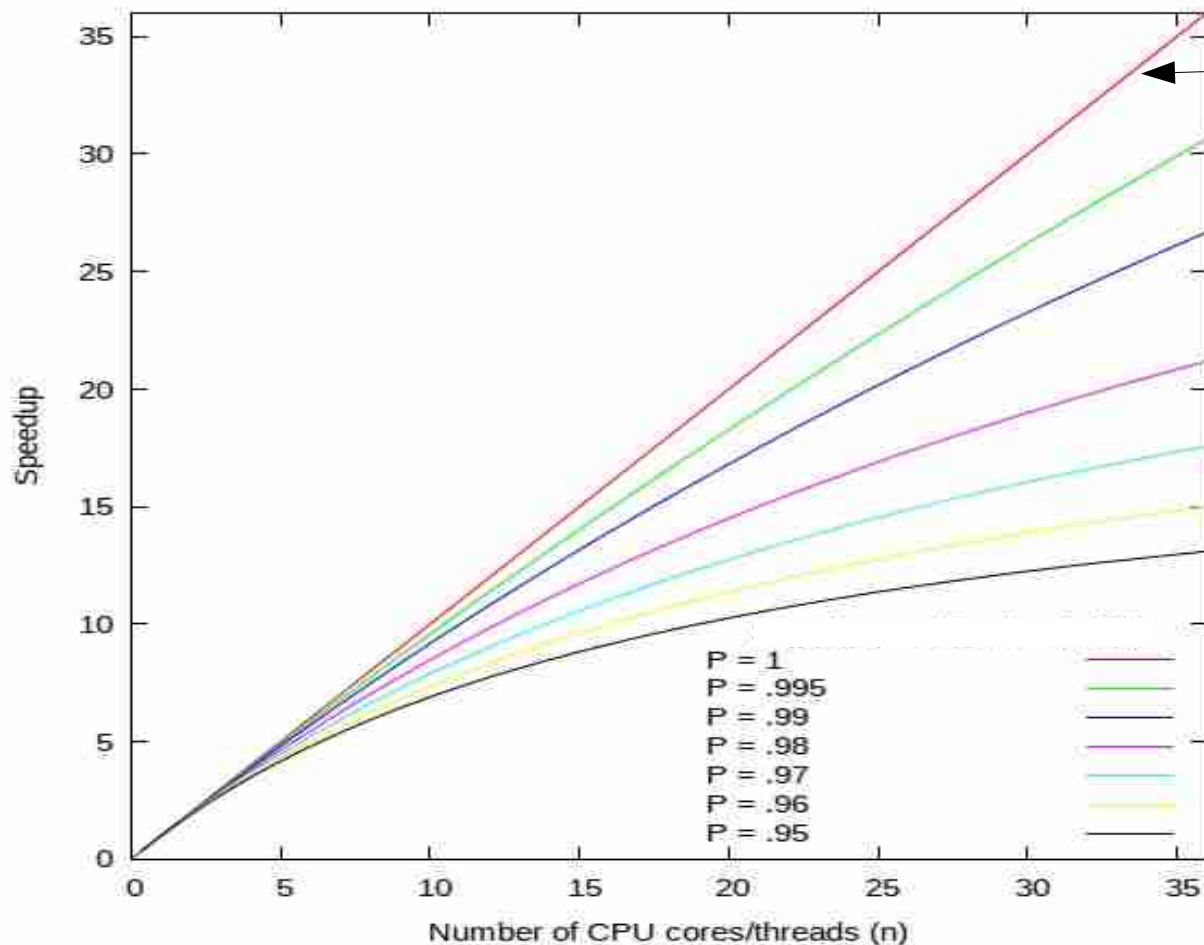


# Amdahl's Law



# Amdahl's law

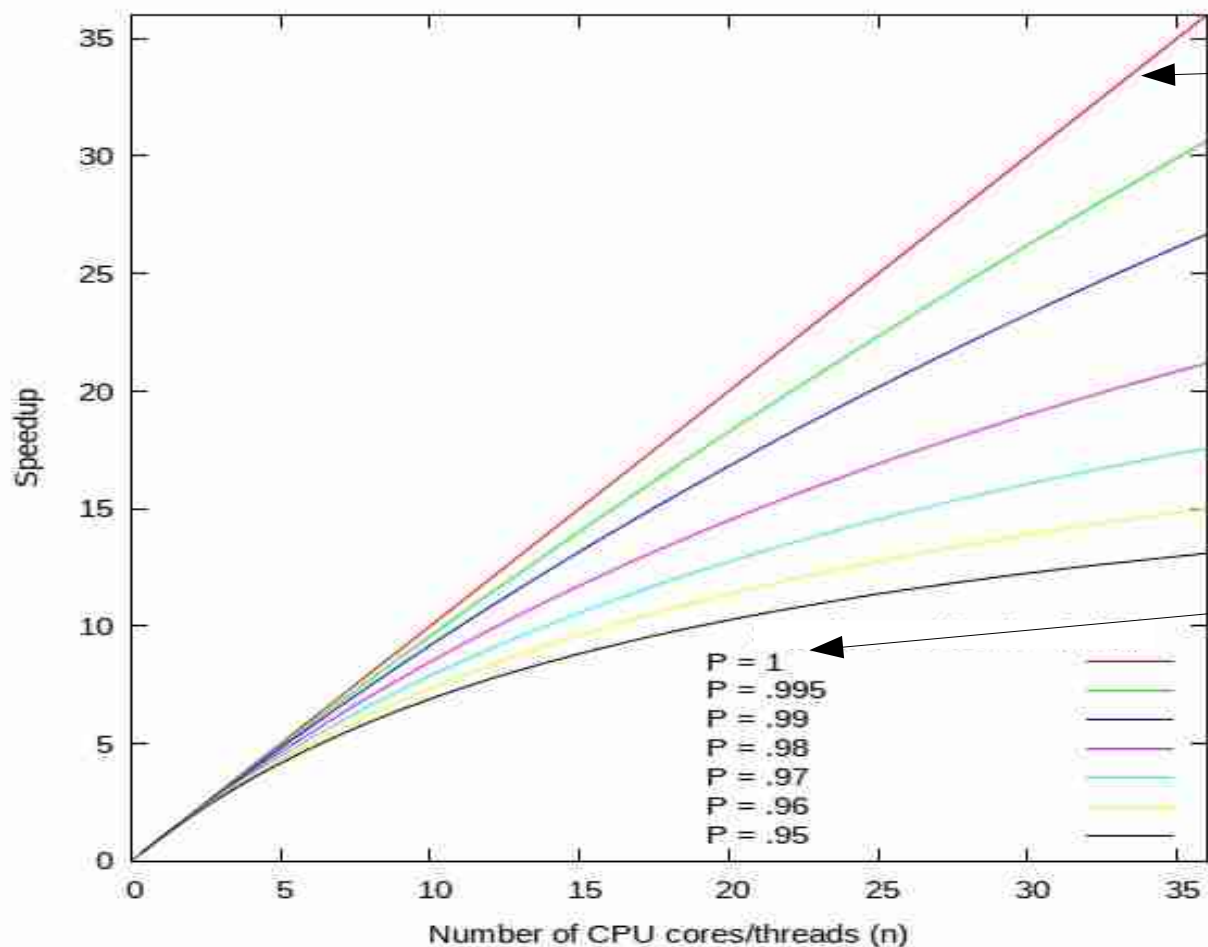
Amdahl's Law at various parallel fractions  $P$



Linear speedup:  $n$  times faster with  $n$  cores!

# Amdahl's law

Amdahl's Law at various parallel fractions  $P$



Linear speedup:  $n$  times faster with  $n$  cores!

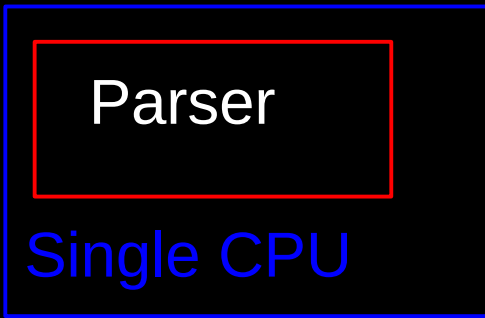
Fraction of code that can be optimally parallelized



# I/O in ExaBayes



T1 ACGT  
T2 ACC-  
T3 ACGG  
T4 AAGC



01010101010101  
01010000001101  
010101010101

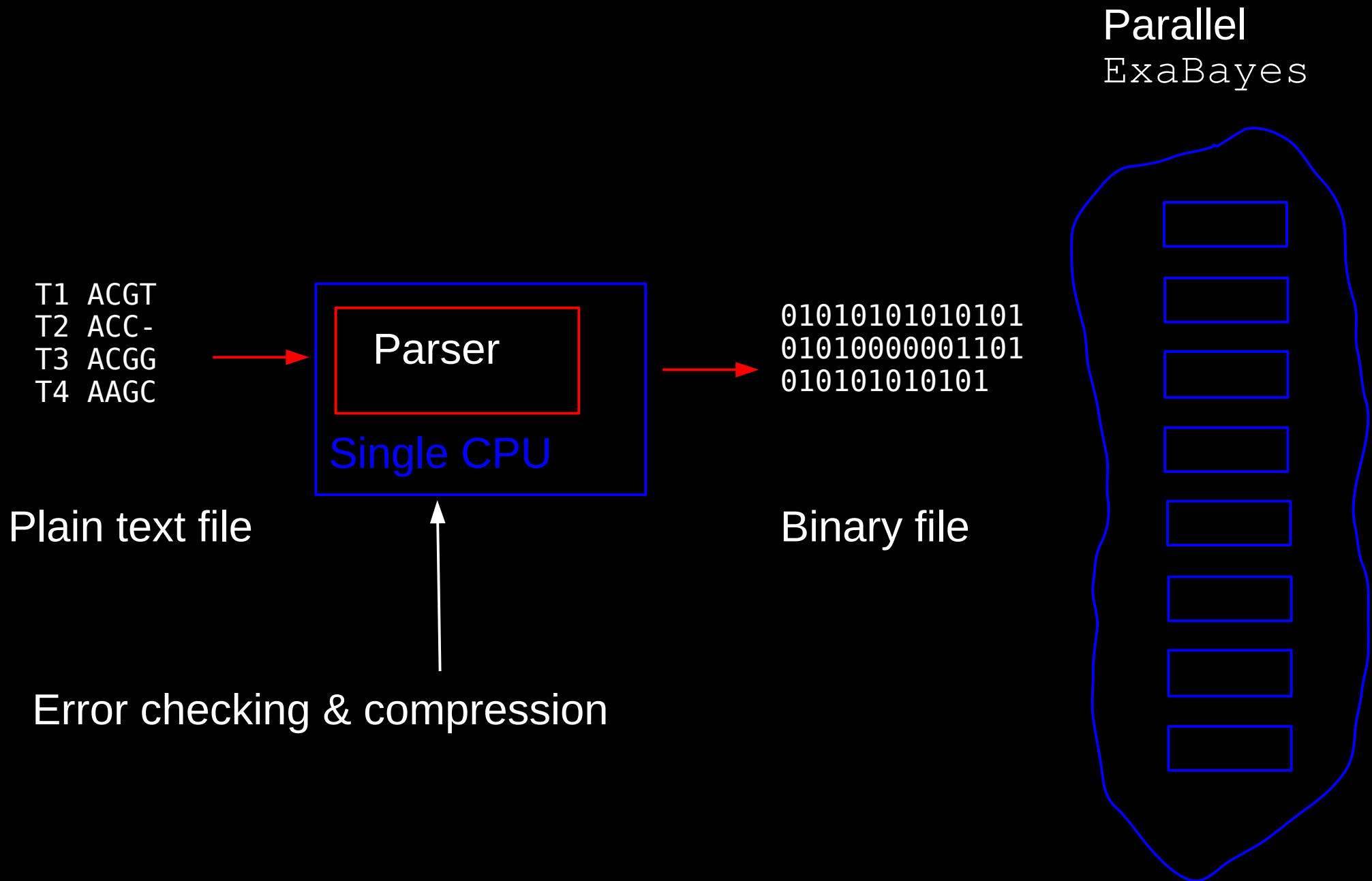
Plain text file

Binary file



Error checking & compression

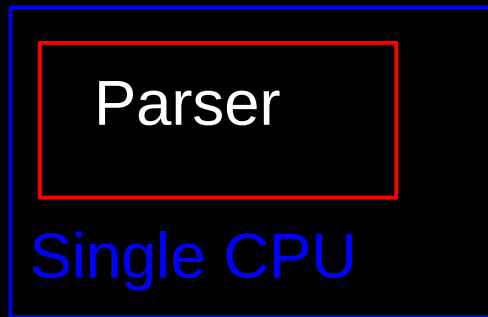
# I/O in ExaBayes



# I/O in ExaBayes

Parallel  
ExaBayes

T1 ACGT  
T2 ACC-  
T3 ACGG  
T4 AAGC

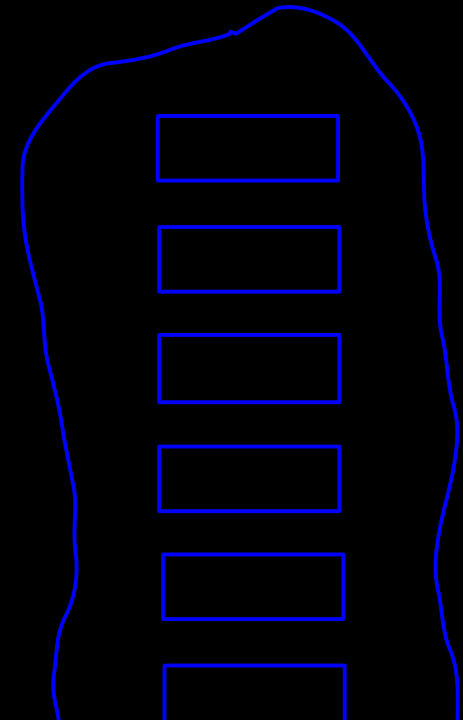


01010101010101  
01010000001101  
010101010101

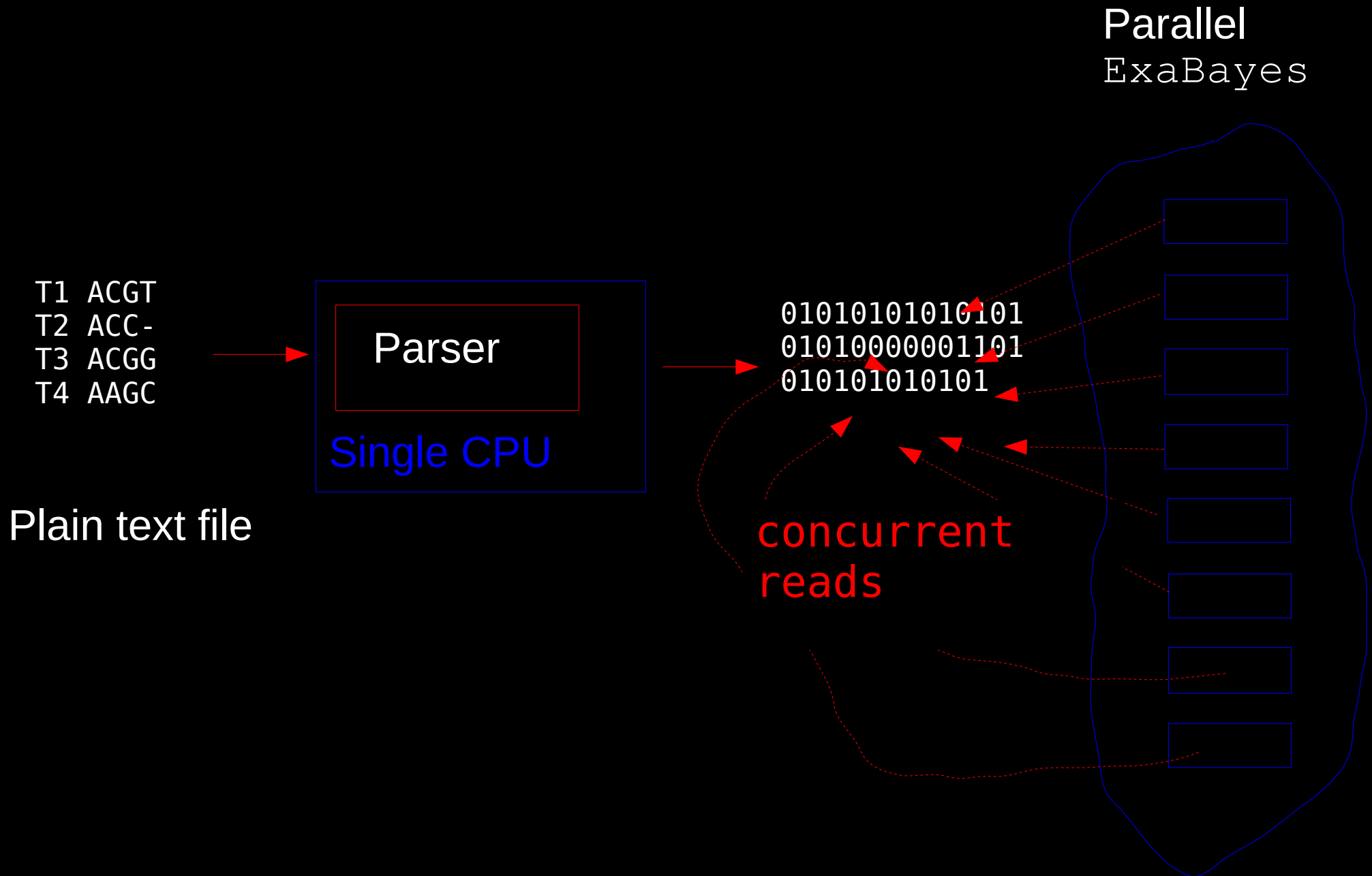
Plain text file

Binary file

Error checking & compression



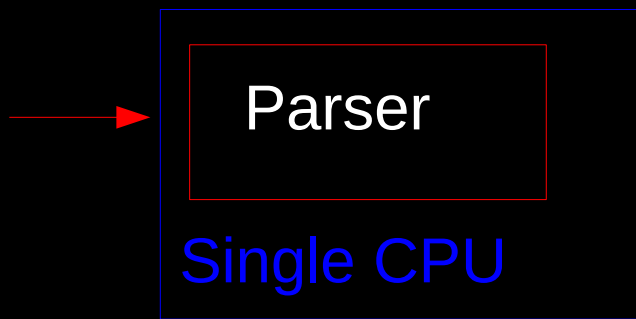
# I/O in ExaBayes



# I/O in ExaBayes

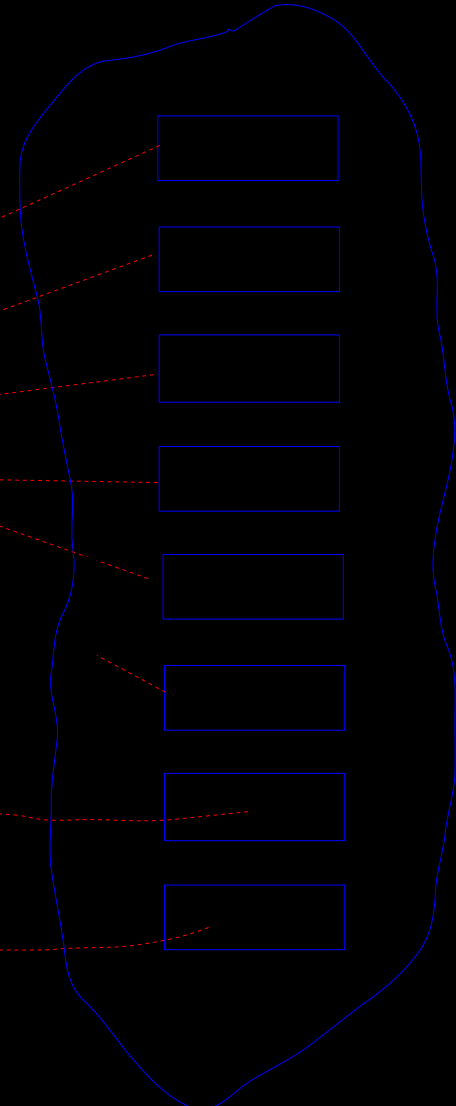
Parallel  
ExaBayes

T1 ACGT  
T2 ACC-  
T3 ACGG  
T4 AAGC



01010101010101  
01010000001101  
010101010101

concurrent  
reads



Plain text file

Start-up reduction from 15 to  
below 1 minute!

# I/O in ExaBayes

Parallel  
ExaBayes

T1 ACGT  
T2 ACC-  
T3 ACGG  
T4 AAGC

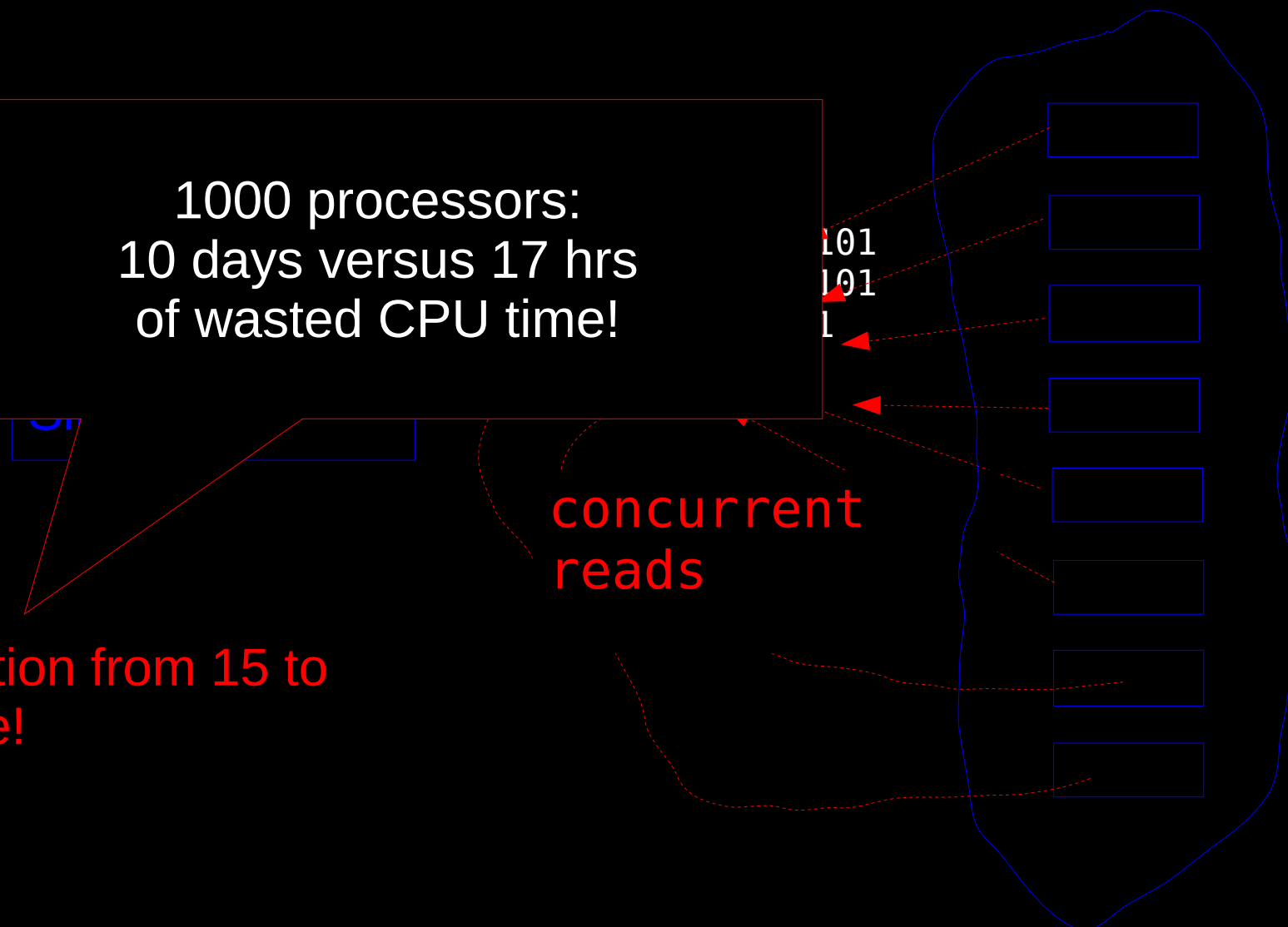
1000 processors:  
10 days versus 17 hrs  
of wasted CPU time!

Plain text file

Start-up reduction from 15 to  
below 1 minute!

concurrent  
reads

101  
101  
1

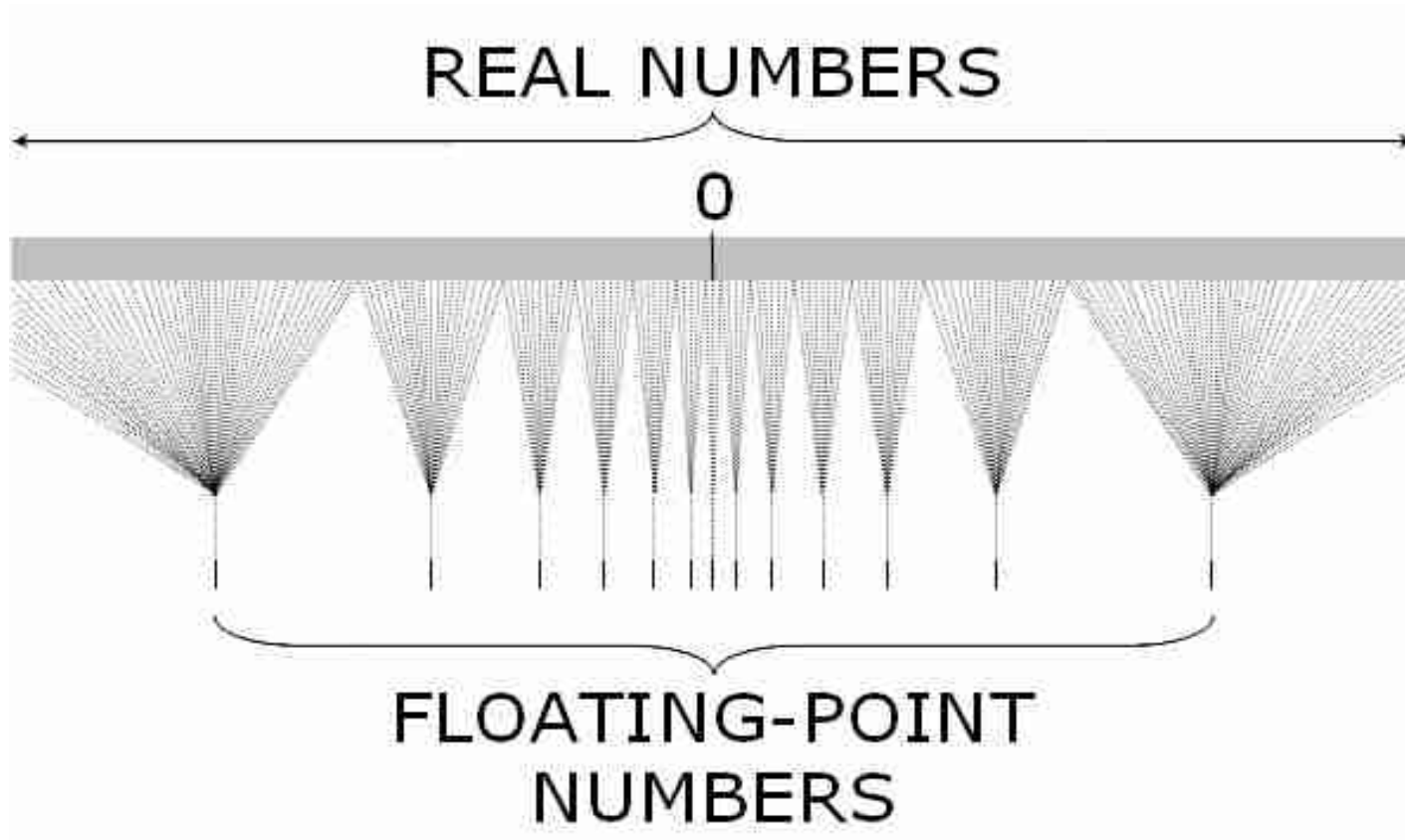


# Outline

- Maximum Likelihood (Recap)
- Sequential Optimization
- Parallelization
- Parallel I/O
- **Numerical Nightmares**
- Energy Efficiency

# Floating Point Numbers

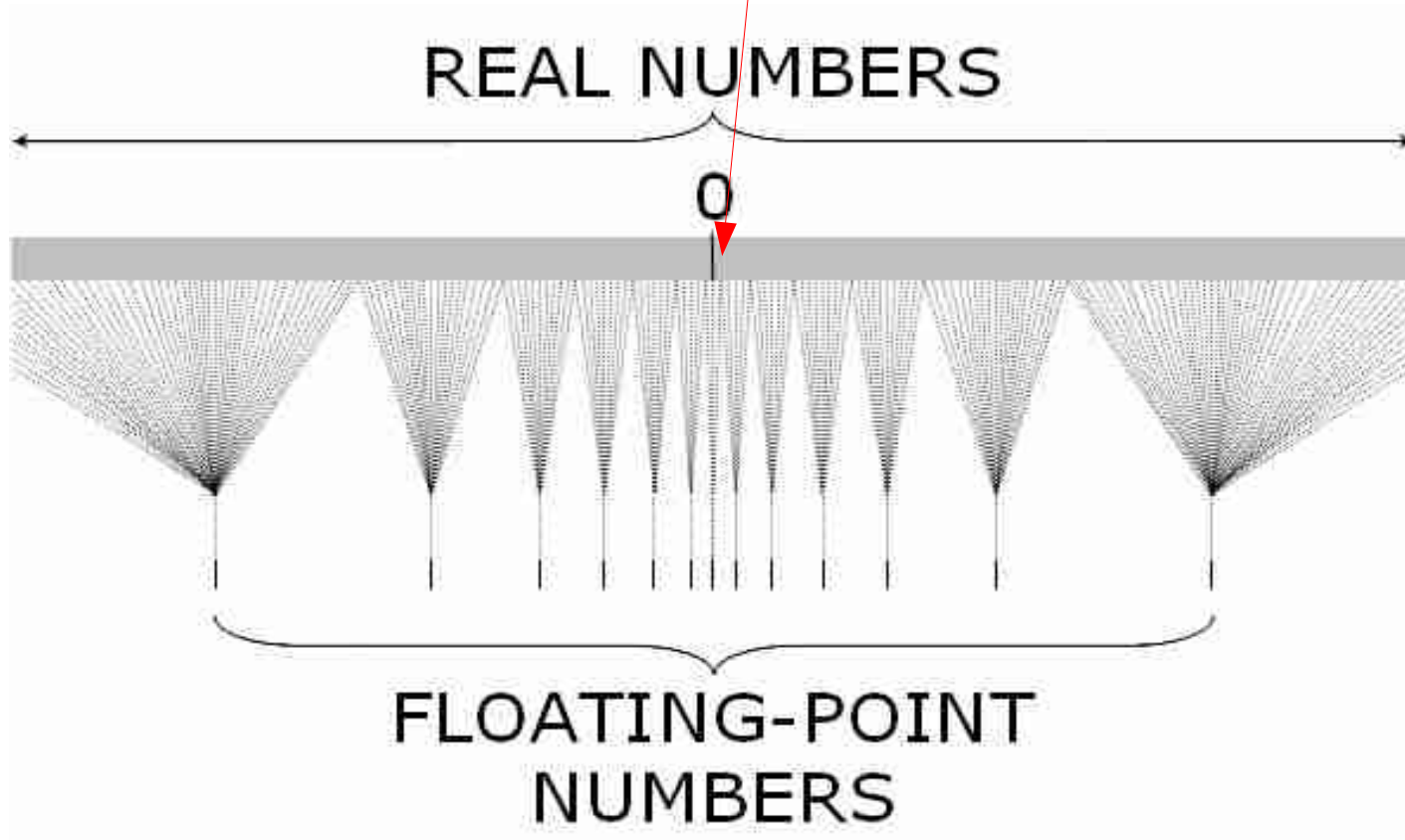
- Machine numbers are an imperfect mapping of the infinite real numbers to a finite number of machine values!



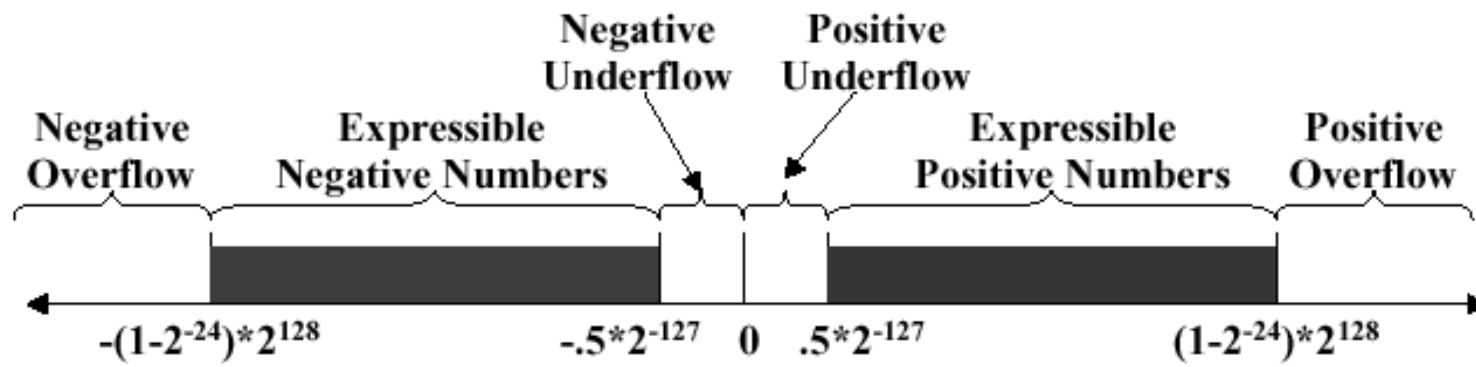


# Numerical Underflow

Conditional likelihood values become so small that they can not be represented on a computer any more → underflow !!!!



# Overflow & Underflow

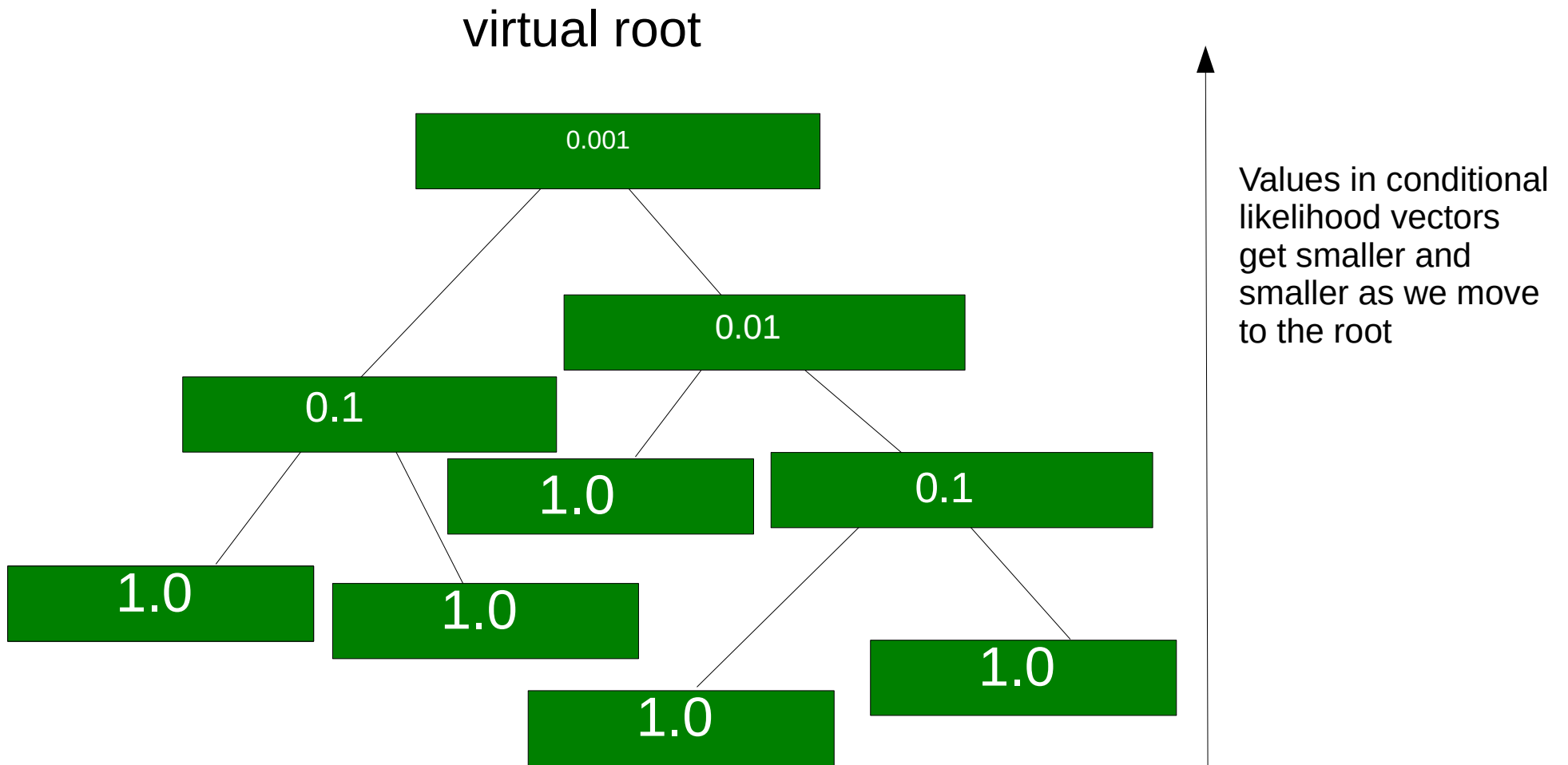


IEEE 754 standard for 32-bit floating point numbers

1 bit     sign  
8 bits    exponent  
23 bits   significand

# Post-order Traversal

*preventing underflow*

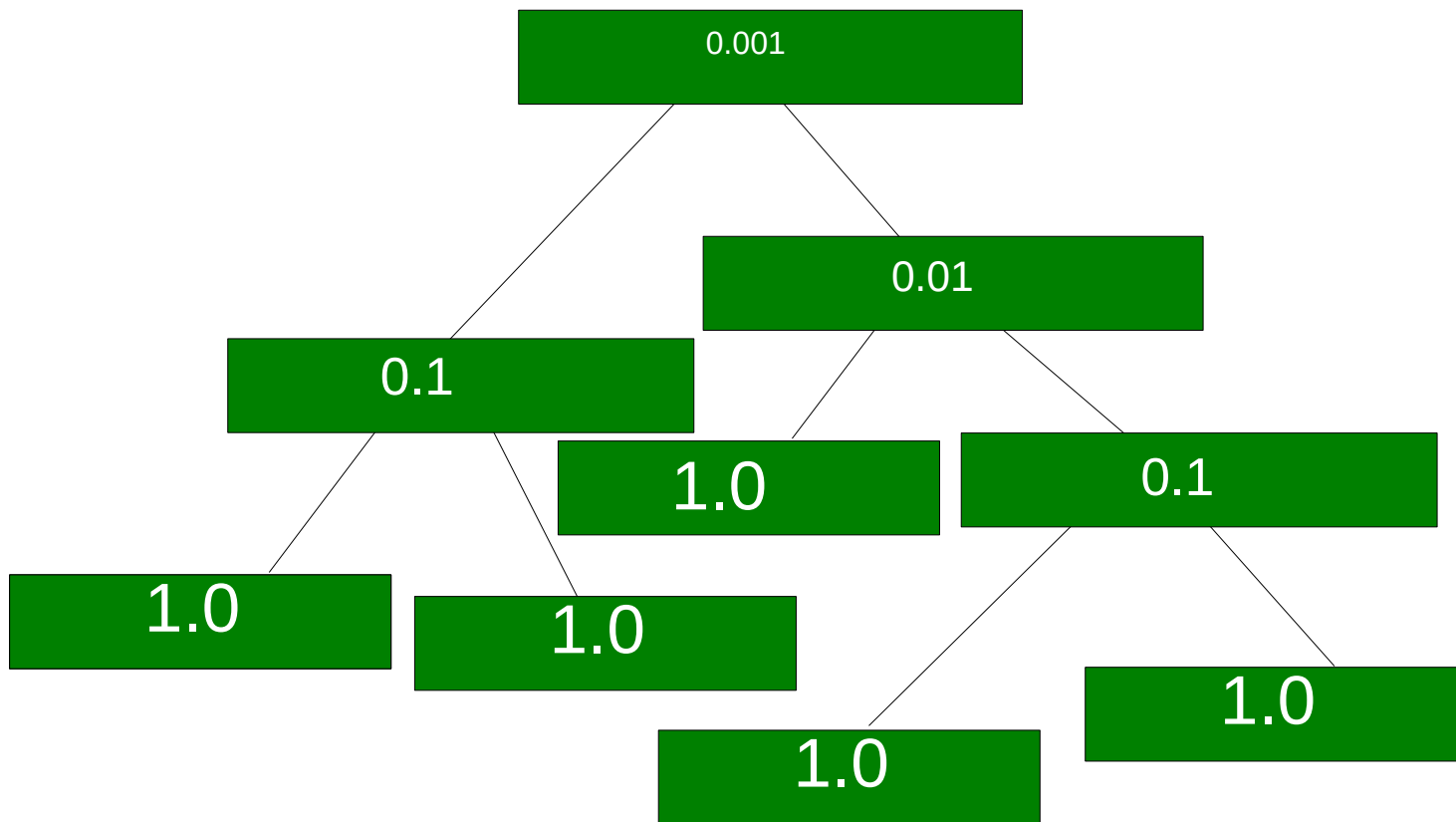


# Post-order Traversal

*preventing underflow*

We need to apply numerical scaling techniques to avoid underflow!

virtual root



Values in conditional likelihood vectors get smaller and smaller as we move to the root → this needs to be handled!

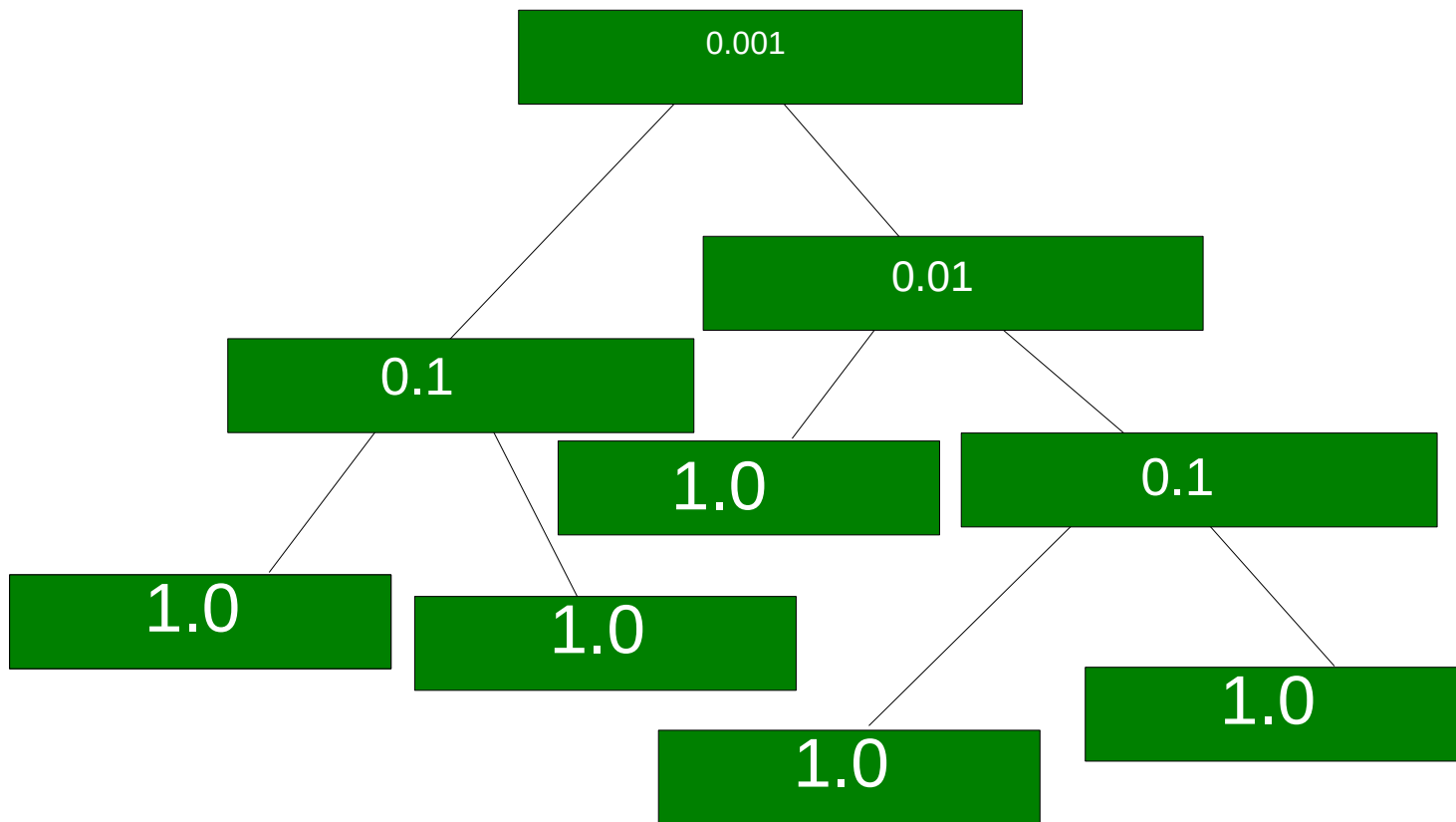
# Post-order Traversal

*preventing underflow*

Typical approach:

- 1) Check if values are too small
- 2) If so multiply with some large number
- 3) Undo those scaling multiplications (somehow) in the end for likelihood this undoing is easy

virtual root



Values in conditional likelihood vectors get smaller and smaller as we move to the root → this needs to be handled!

# What went wrong?

- For DNA models without rate heterogeneity this scaling approach worked fine
  - check if all **4** conditional likelihoods at a given CLV and site are smaller than a minimum & multiply with large number
- For DNA models with rate heterogeneity this doesn't work
  - jointly checking that all **16** conditional likelihoods for the **4** typical discrete rates are smaller than a minimum doesn't work
  - the spread of the values is too large because of the distinct rate categories
  - scale individually per rate category
  - higher computational cost

> [BMC Bioinformatics](#). 2011 Dec 13;12:470. doi: 10.1186/1471-2105-12-470.

## Algorithms, data structures, and numerics for likelihood-based phylogenetic inference of huge trees

Fernando Izquierdo-Carrasco <sup>1</sup>, Stephen A Smith, Alexandros Stamatakis

Affiliations + expand

PMID: 22165866 PMCID: [PMC3267785](#) DOI: 10.1186/1471-2105-12-470



[Free PMC article](#)

# Single Precision?

- We know that likelihood calculations are compute- and memory-intensive
- So why not use single-precision (32 bit) instead of double precision (64 bit) floating point values?
- Numerics for Maximum Likelihood break down
- 10-fold increase in scaling multiplications when using single precision

Accuracy and Performance of Single versus Double Precision  
Arithmetics for Maximum Likelihood Phylogeny  
Reconstruction

[Simon A. Berger](#) & [Alexandros Stamatakis](#)

Conference paper

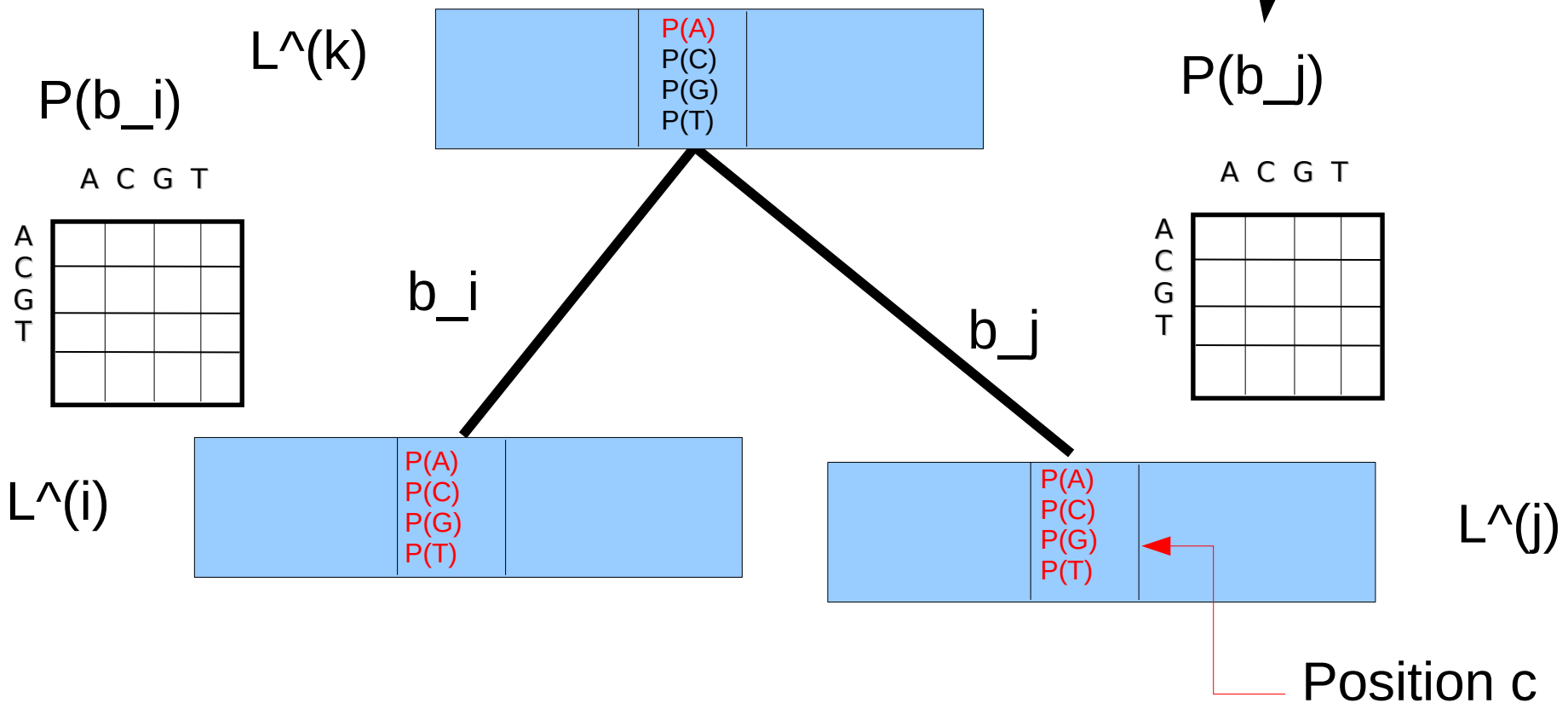
958 Accesses | 4 Citations

Part of the [Lecture Notes in Computer Science](#) book series (LNTCS, volume 6068)

# Felsenstein pruning

$P(t) = e^{Qt}$  is numerically not easy

$$\vec{L}_A^{(k)}(c) = \left( \sum_{S=A}^T P_{AS}(b_i) \vec{L}_S^{(i)}(c) \right) \left( \sum_{S=A}^T P_{AS}(b_j) \vec{L}_S^{(j)}(c) \right)$$





# Felsenstein pruning

$P(t) = e^{Qt}$  is numerically not easy

SIAM REVIEW © Society for Industrial and Applied Mathematics  
 Vol. 20, No. 4, October 1978 0036-1445/78/2004-0031\$01.00/0

**NINETEEN DUBIOUS WAYS TO COMPUTE  
THE EXPONENTIAL OF A MATRIX\***

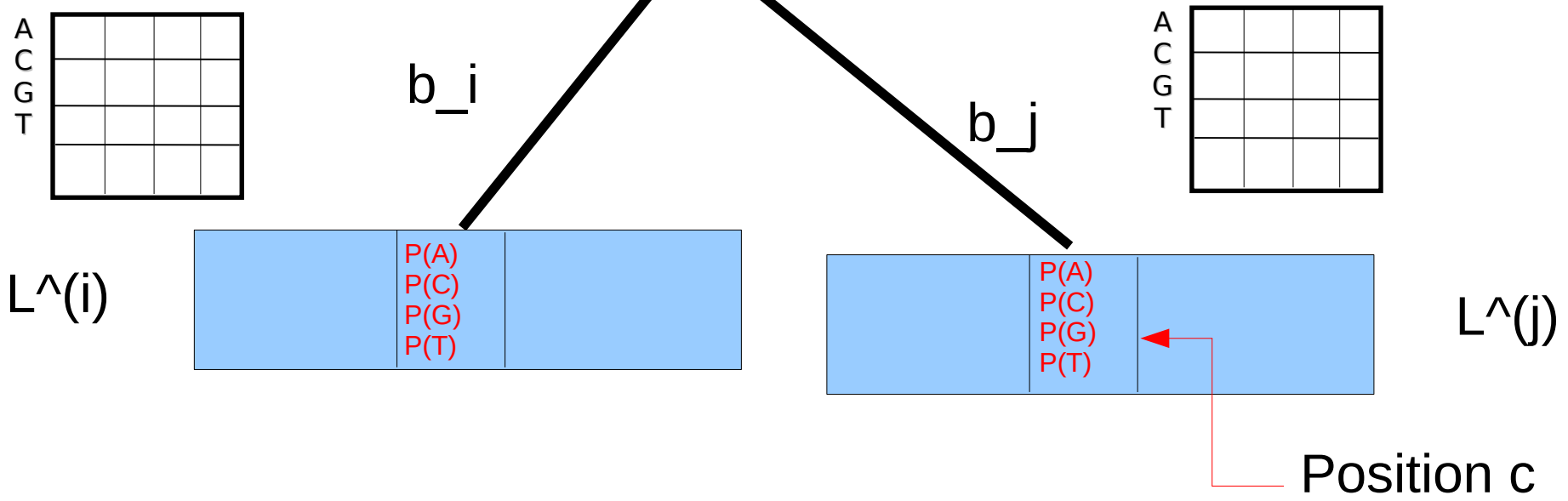
CLEVE MOLER† AND CHARLES VAN LOAN‡

**Abstract.** In principle, the exponential of a matrix could be computed in many ways. Methods involving approximation theory, differential equations, the matrix eigenvalues, and the matrix characteristic polynomial have been proposed. In practice, consideration of computational stability and efficiency indicates that some of the methods are preferable to others, but that none are completely satisfactory.

$\bar{L}_S^{(j)}(e)$

j)

G T



# What went wrong?

- In `RAxML` we used the matrix exponential function from the book - *Numerical Recipes in C*
- Especially the Intel `icc` compiler tended to be very aggressive when trying to optimize this function
  - numerical breakdown
- Solution

```
eigen.o : eigen.c $(GLOBAL_DEPS)
          $(CC) -c -o eigen.o eigen.c
```

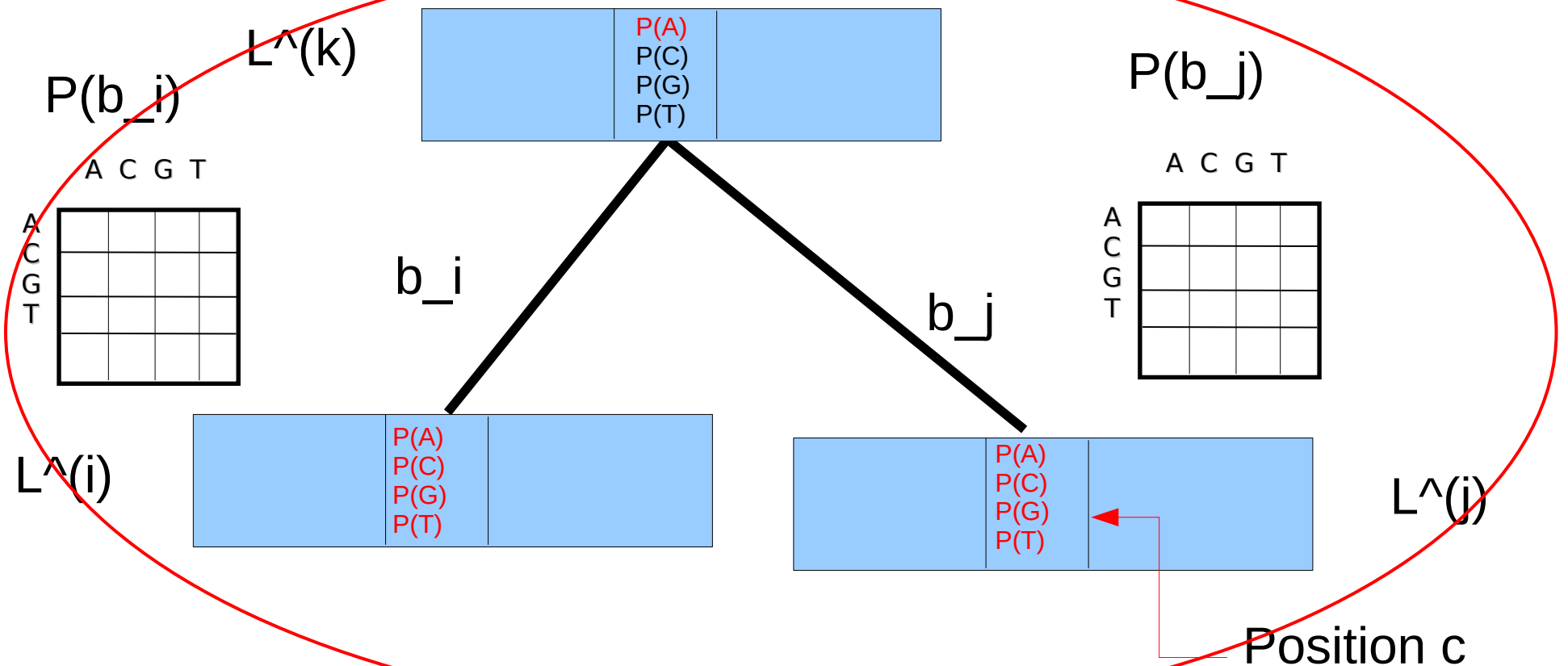
Compile eigenvector decomposition function **without** optimization flags

Consider that you only want to compute this triplet of conditional likelihood vectors of fixed length  $n$ .

$L^{(i)}$ ,  $L^{(j)}$ ,  $P(b_i)$ ,  $P(b_j)$  are given as input and you just compute  $L^{(k)}$  as output of a micro-benchmark.

What do you expect the run-times to be if you just provide different input vectors  $L^{(i)}$ ,  $L^{(j)}$  but all of length  $n$ ?

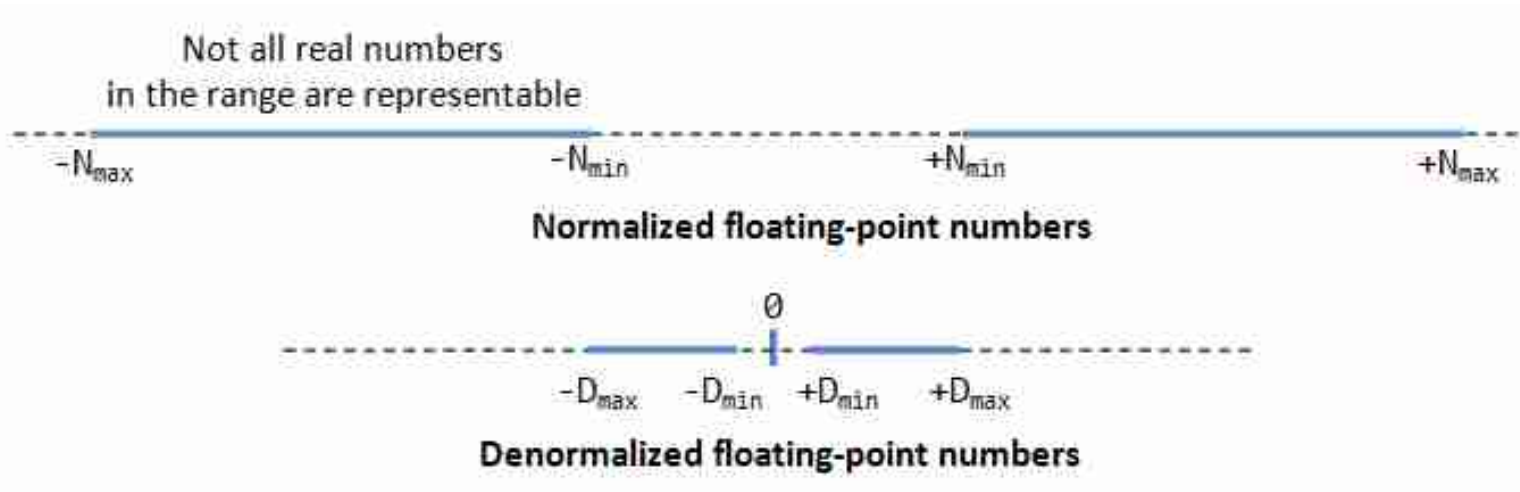
$$\vec{L}_A^{(k)}(c) = \left( \sum_{S=A}^T P_{AS}(b_i) \vec{L}_S^{(i)}(c) \right) \left( \sum_{S=A}^T P_{AS}(b_j) \vec{L}_S^{(j)}(c) \right)$$



# What went wrong?

- When developing phylogenetic placement methods, we observed some inexplicable run time deviations for exactly this operation of about 50%
- It didn't make any sense since we executed  $n$  times the exact same arithmetic operations, just on different input data
  - until we learned about de-normalized floating point values

# Denormalized Floating Point Numbers



Intended to allow for gradual underflow to zero

When de-normalized values are encountered, the processing cost inside the CPU for multiplications and additions is increased.

- the runtimes are input-data dependent !
- Problem with reproducibility of run time performance benchmarks

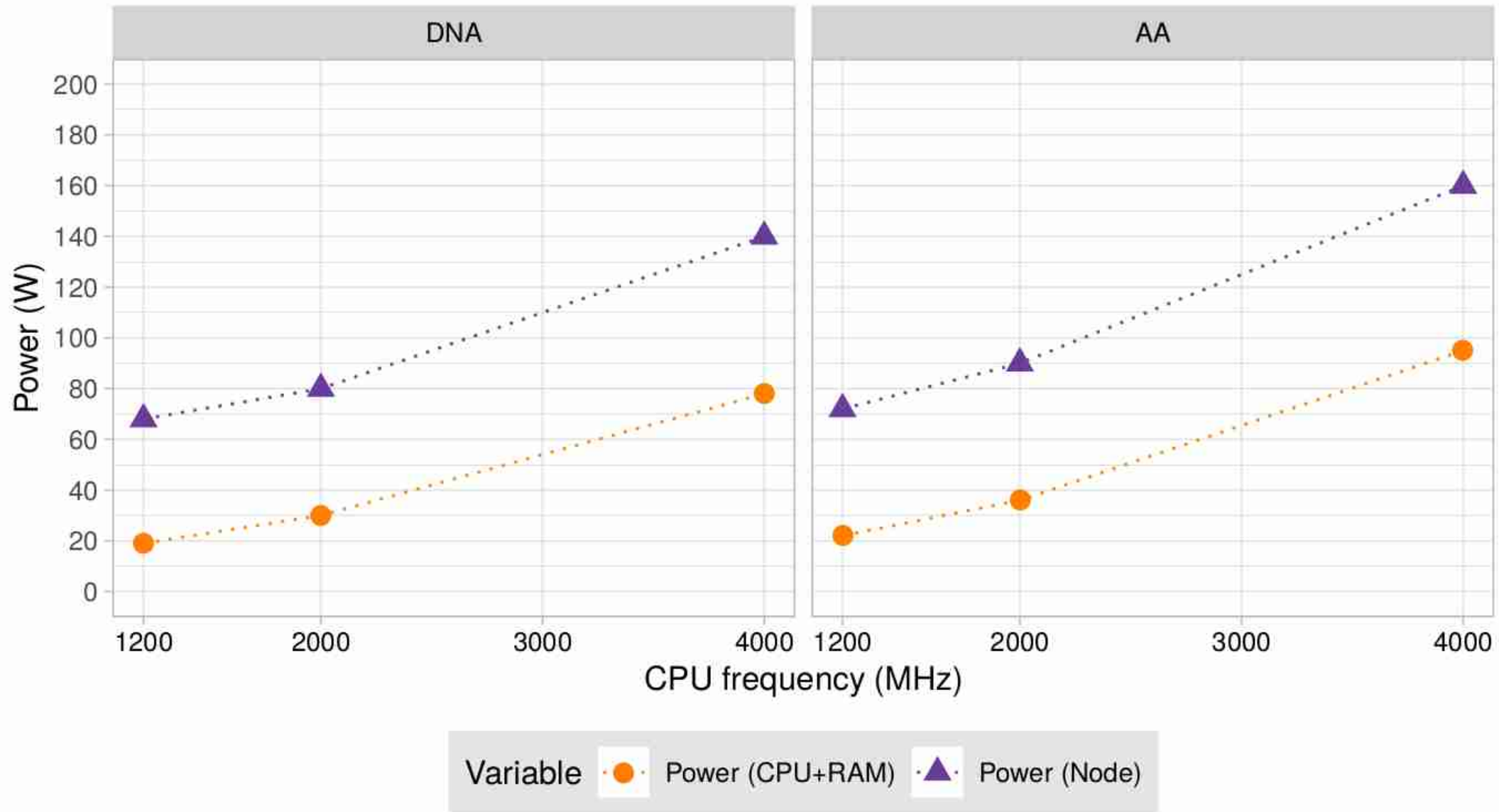
# Denormalized Numbers

- De-normalized floating point numbers and their impact on run-times and performance benchmark
  - J. Björndalen, O. Anshus: “Trusting floating point benchmarks-are your benchmarks really data-independent?” Applied Parallel Computing. State of the art in Scientific Computing 2010; pp 178-188, Springer.
  - Alexandre F. Tenca, Kyung-Nam Han, David Tran: “Performance Impact of Using Denormalized Numbers in Basic Floating-point Operations” IEEE, Forty-First Asilomar Conference on Signals, Systems and Computers, 2007.
- The concrete example with Conditional Likelihood Vector computations that yielded highly diverging run times due to de-normalized floating point numbers can be found here  
<https://github.com/stamatak/denormalizedFloatingPointNumbers>

# Outline

- Maximum Likelihood (Recap)
- Sequential Optimization
- Parallelization
- Parallel I/O
- Numerical Nightmares
- **Energy Efficiency**

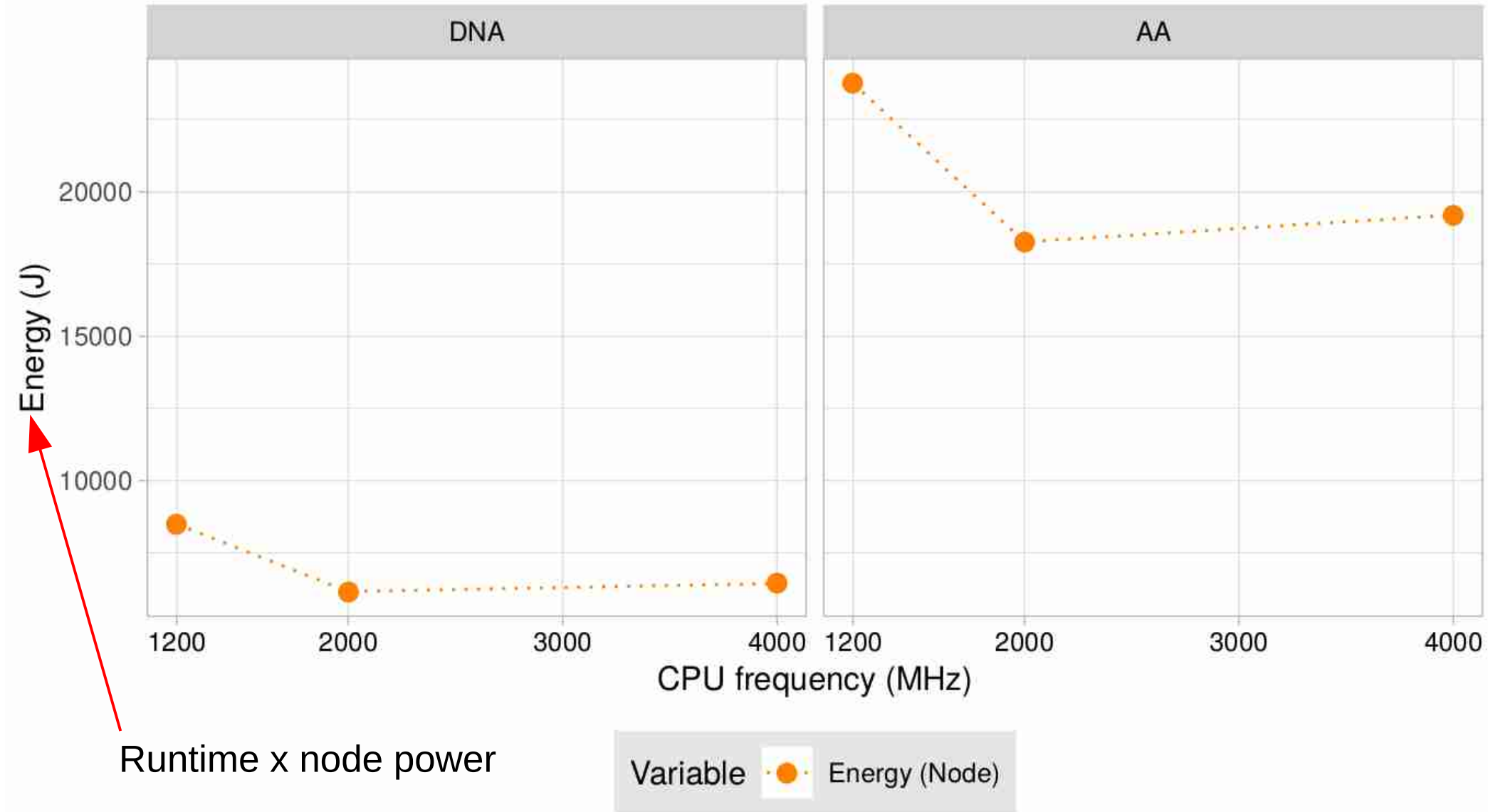
# Phylogenetic Inference



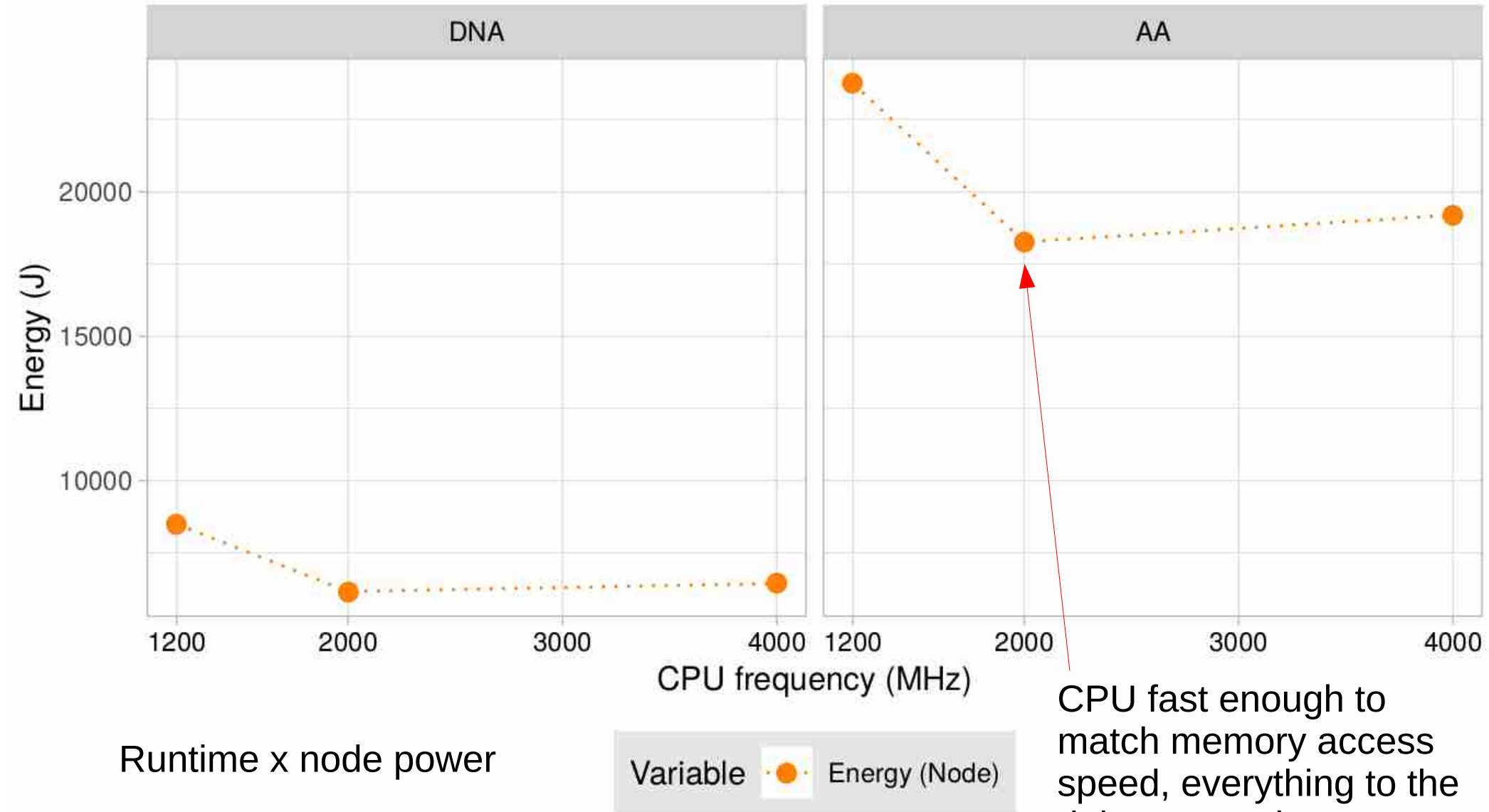
CPU frequency correlates well with power for RA<sub>x</sub>ML-NG



# Phylogenetic Inference

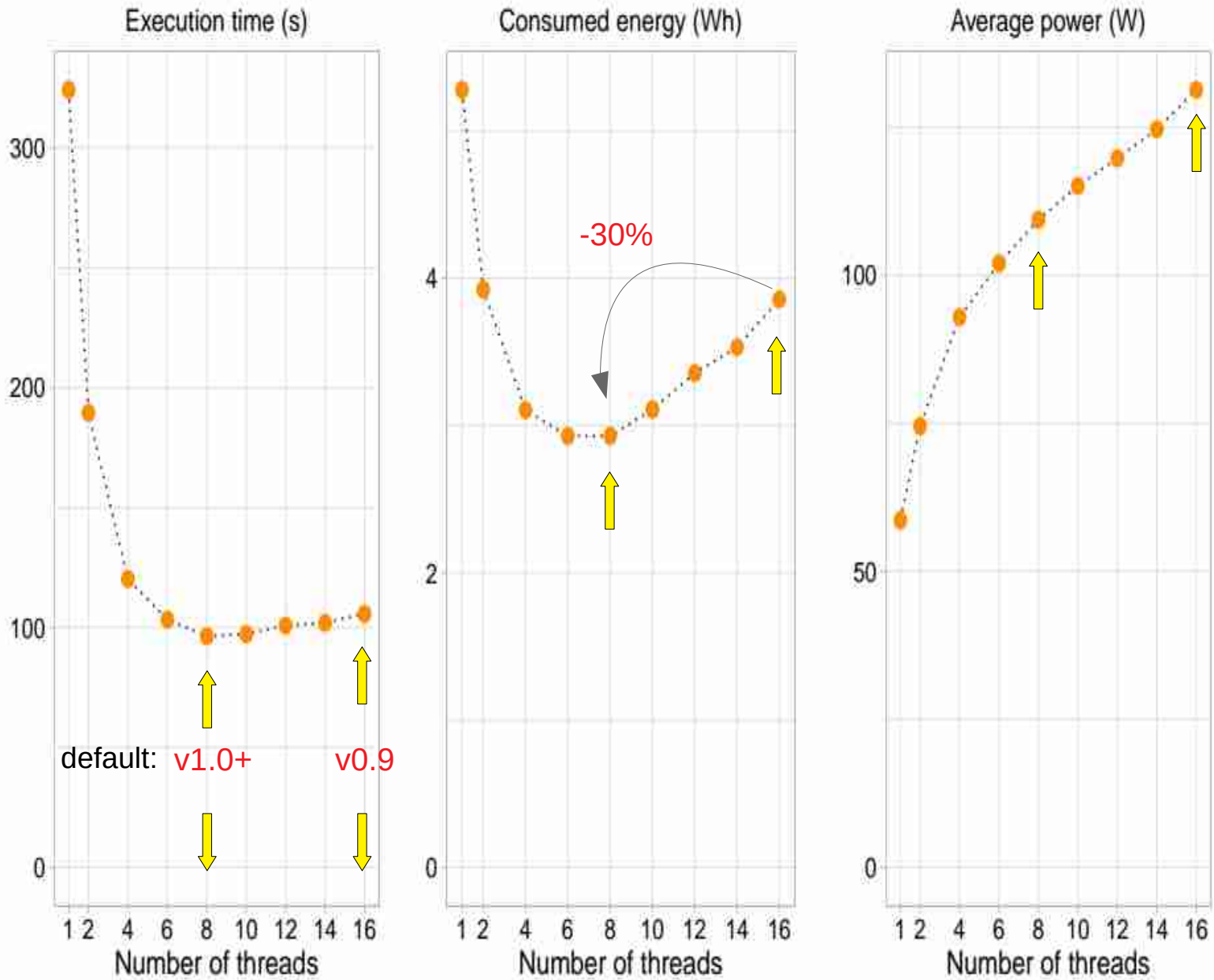


# Phylogenetic Inference



# RAxML-NG v0.9 vs. v1.0

## energy saving



# Thank you for your Attention !

