Main Seminar
Hot Topics in Bioinformatics

Alexandros Stamatakis

Now

- **How to give a scientific presentation**
- How to write a technical report/scientific paper

Scientific Presentations

- This is my personal view, based on over 120 talks I have given
- At CS conferences: typically 25 minutes + 5 minutes questions
- Timing
  - Practice your talk beforehand
  - Usually 1.5-2 minutes per slide
  - Keep in mind, when presenting one usually talks faster than during the rehearsal
  - Use SW tools for timing, e.g., sliding time bars
  - Use a **spell-checker** for the slides!

The audience

- Giving talks gets easier and easier the more talks you give
- Try to imagine what kind of audience you might be expecting
- What may or may they not know about the problem at hand?
- What terms & acronyms can be assumed to be standard knowledge?
- What terms & acronyms are too subject-specific?
- A talk about the same subject will be very different if you are talking to
  - Theoretical computer scientists
  - HPC engineers
  - Bioinformaticians
  - Evolutionary Biologists
- You will have to explain different things/concepts in more detail!

Slide layout

- Keep slides simple
- Use a **spell checker** for the slides
- Decide if you want to use American or British English
- Avoid numbering of type 10/50, 11/50 → audience will think: "... another 39 boring slides to go"
- Reduce text to the absolute minimum!
- Avoid busy slides including text & graphs & tables

  → if you need a complex slide use layers, i.e., first show text, then add graph, then add a table

  → direct the attention of your audience
- Avoid tables if possible → use intuitive graphical representations

Structure

- Provide an outline for your talk

  → re-use it at the beginning of each section

- Structure
  - State & Motivate the problem
    - Why is it interesting?
    - Why is it important?
  - Own contribution: very brief
    - What did you do/What did you contribute?
    - This is often very fuzzy, I have attended many talks where it was not clear for a long time what the contribution of the authors actually was
    - Throughout the talk, make it very clear:

      (i) what is prior knowledge

      (ii) what you did contribute

Structure

- Outline
- Intro & Motivation
- Own Contribution
- Abstract, but more detailed problem description

  → omit unnecessary details
- Describe your solution/contribution
- Experimental Results

    → experimental setup

    → HW & datasets used

    → results (if possible no tables, intuitive graphs)

    → comparison with competing approaches
- Conclusions & Future Work
- Acknowledgements: funding agencies, people who have helped you

Reporting Results

- e.g., parallel speedups

  → don't show execution time over processor plots

  → show speedups, much easier to interpret
- Showing graphs

  → label the *x* and *y* axis!

  → before discussing the graphs, say what the *x* and *y* represent!

  → don't show more than one graph per slide!

Acronyms

- I can't stand it when people use acronyms in texts without introducing them
- There are very few acronyms that are known to everybody
  - MPI is one of these …
  - But only at a HPC conference
- Don't use acronyms in slides
- People can not remember their meaning in such a short time

An example presentation

- Work on barriers
- Presented at IEEE Cluster in 2010
- I didn't have much time to prepare the slides
- By far not perfect
- I have commented the slides
  - Things I liked
  - Things I didn't like

Assessment of Barrier Implementations for Fine-Grain Parallel Regions on Current Multi-core Architectures

The title

The Authors
Who is giving the talk?

Simon A. Berger and **Alexandros Stamatakis**

The Exelixis Lab
Bioinformatics Unit
Department of Computer Science
Technical University of Munich

stamatak@cs.tum.edu
http://wwwkramer.in.tum.de/exelixis

Affiliation and
Contact details

Outline

- **Motivation**
- Type of parallel regions
- Barriers
- Test Applications
- Results
- Conclusions

Motivation

- Little is known about efficient barrier implementations on multi-cores using Pthreads and OpenMP
- Need for assessment on current multi-core architectures
- Focus on applications with large number of fine-grain parallel regions

→ applications where barrier performance is an issue

- Background: Bioinformatics application for reconstruction of evolutionary trees from DNA data

There's a bit too much text
On this slide + it's badly formatted

Questions

Sentence too long!

- Does barrier efficiency depend on
  - Specific multi-core architecture?
  - Memory and Cache utilization and access behavior of the application at hand?
- Best barrier implementation for Pthreads?
- Pthreads implementation required for non-expert users!

That's actually the motivation for Using Pthreads

Goals

Sentence maybe too long!
Not well-formatted

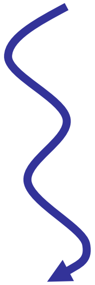
- Devise efficient barrier implementation
- Provide for barrier with deterministic reduction implementation
  - Reductions a + b + c + d must always be conducted in the same order, e.g.:
    - (a + b) + (c + d)
    - A reduction on the same numerical values must yield exactly the same result!
  - Not necessarily the case with OpenMP and MPI

Outline

- Motivation
- **Type of parallel regions**
- Barriers
- Test Applications
- Results
- Conclusions

Thread Sync:
Fork-Join Model

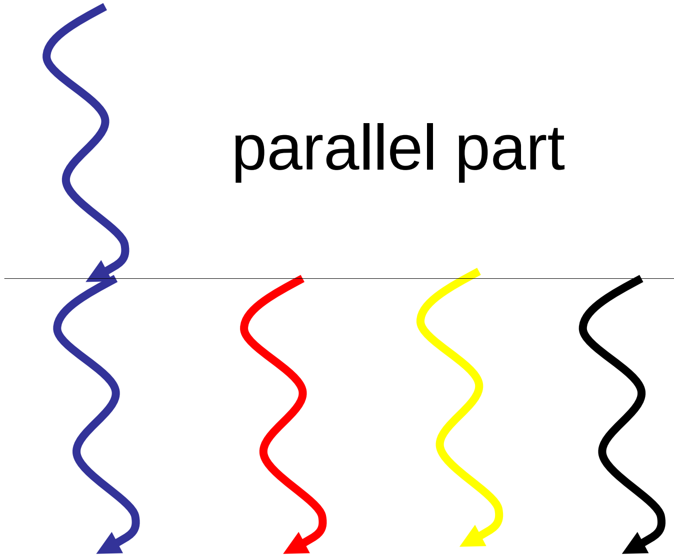
Master thread



Sequential part

The followings slides are rather nice
We build up what we need step by step

Thread Sync:
Fork-Join Model

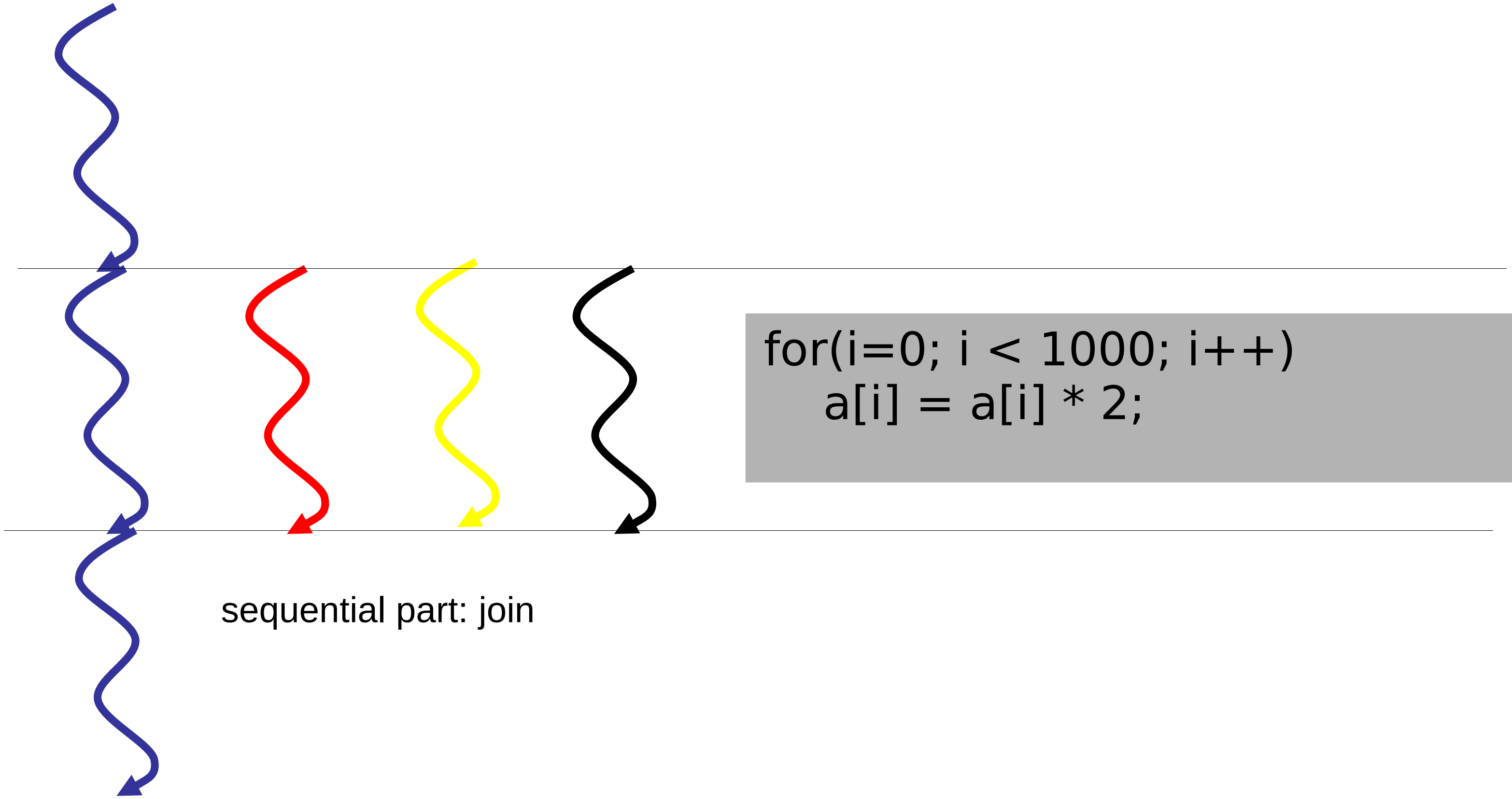


parallel part

Maybe add this in the next slide

```
for(i=0; i < 1000; i++)
    a[i] = a[i] * 2;
```

Thread Sync:
Fork-Join Model

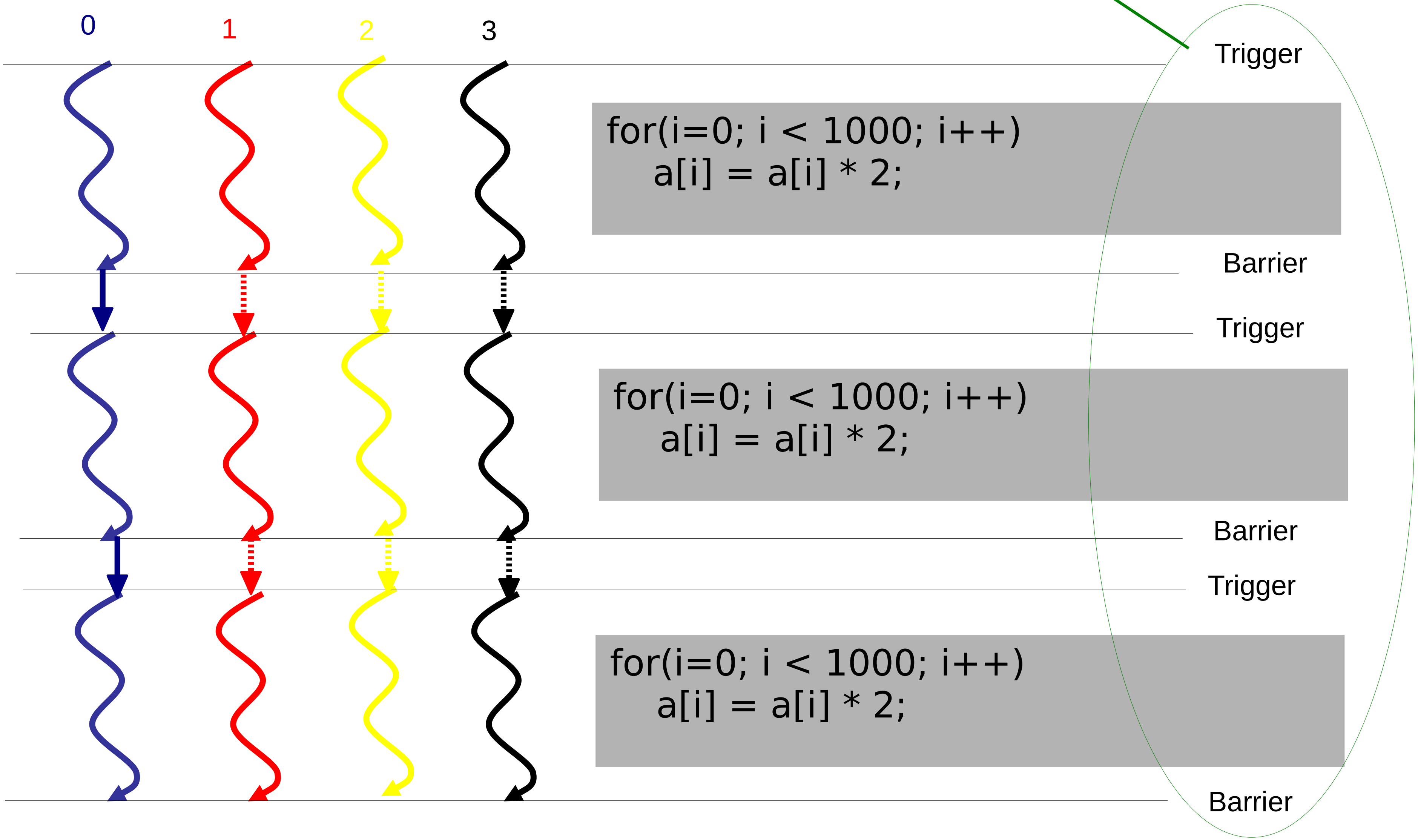


```
for(i=0; i < 1000; i++)
    a[i] = a[i] * 2;
```

sequential part: join

Should have added this on the next slide

Thread Sync: Barrier-based

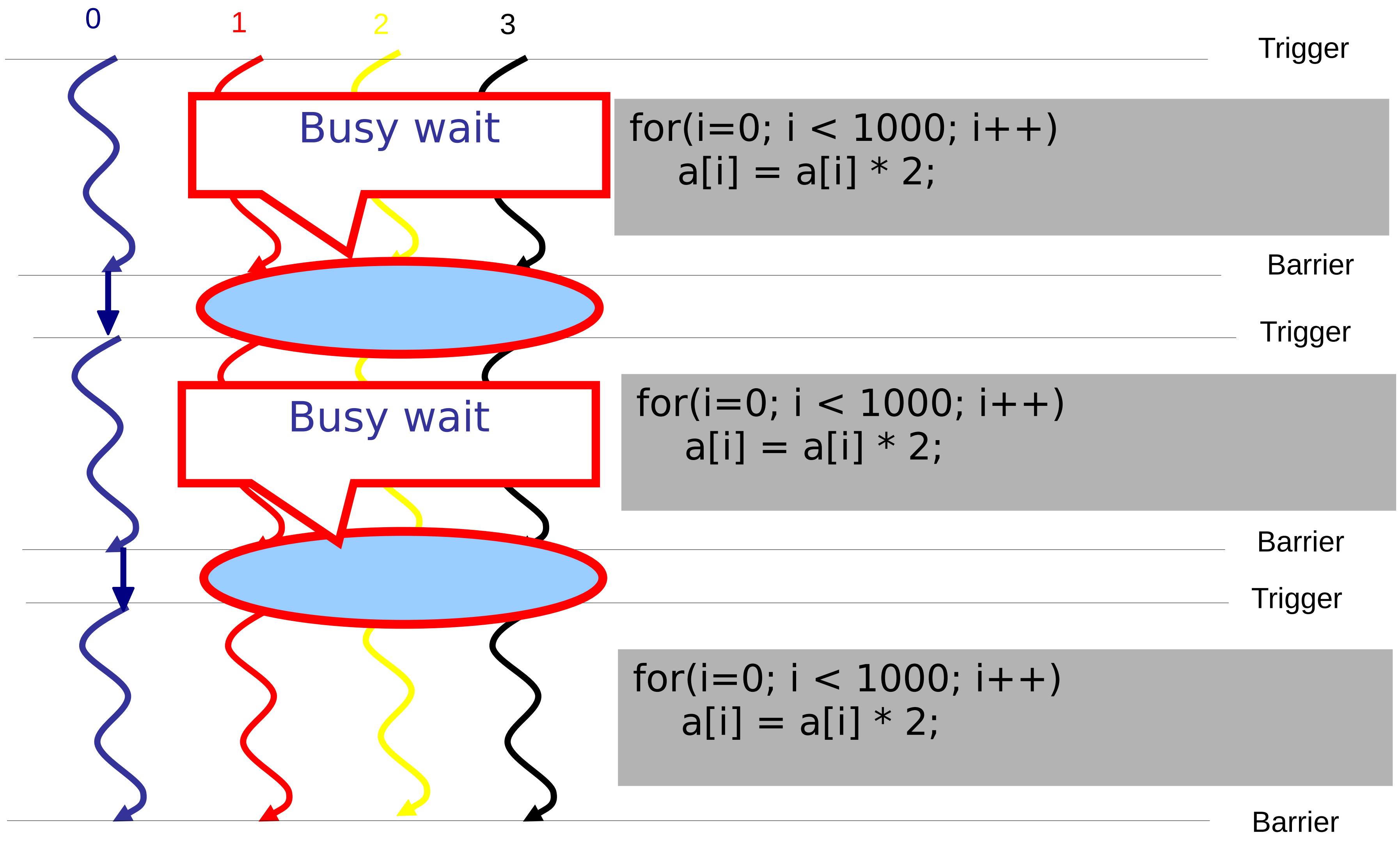


```
for(i=0; i < 1000; i++)
    a[i] = a[i] * 2;
```

```
for(i=0; i < 1000; i++)
    a[i] = a[i] * 2;
```

```
for(i=0; i < 1000; i++)
    a[i] = a[i] * 2;
```

Thread Sync: Barrier-based

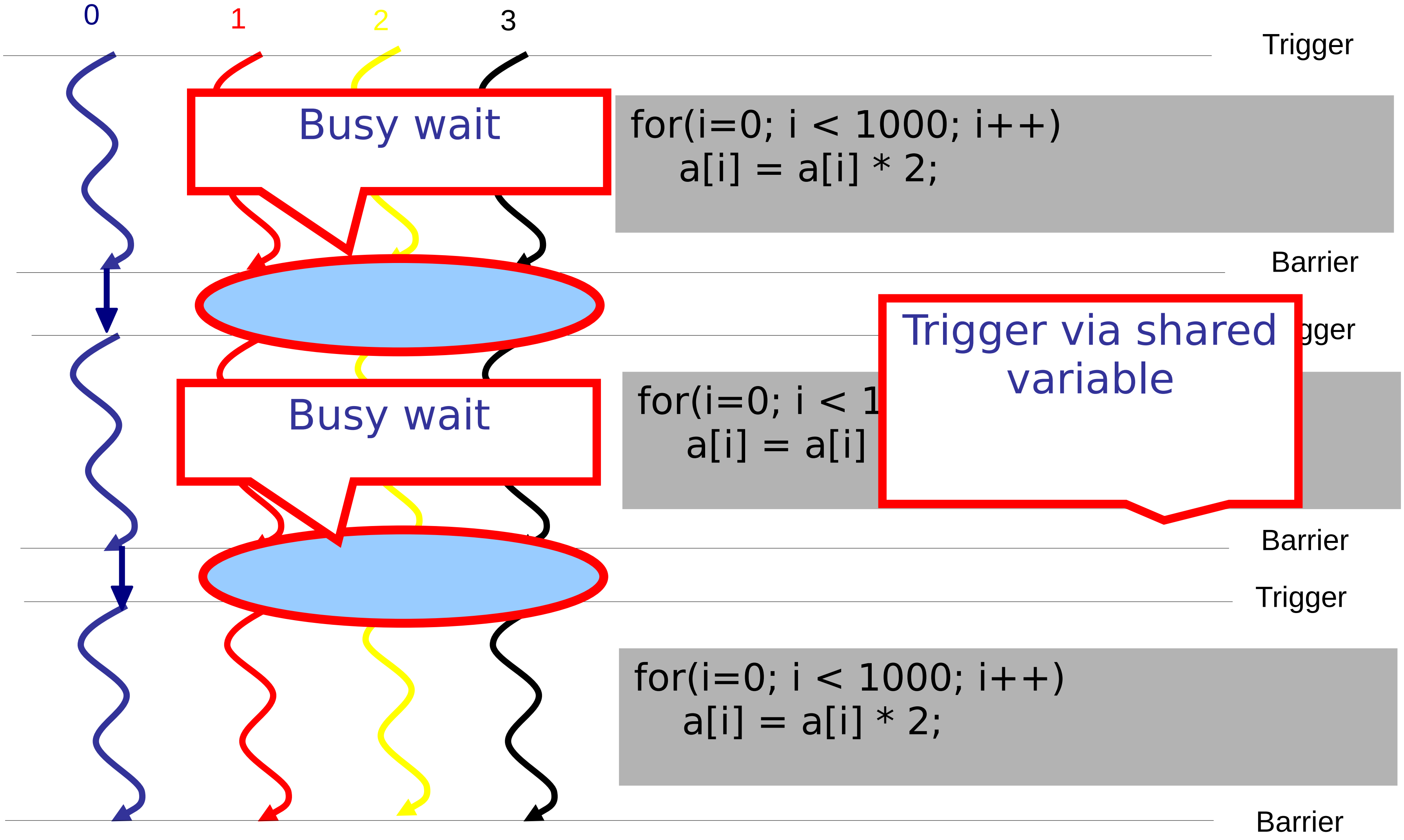


```
for(i=0; i < 1000; i++)
    a[i] = a[i] * 2;
```

```
for(i=0; i < 1000; i++)
    a[i] = a[i] * 2;
```

```
for(i=0; i < 1000; i++)
    a[i] = a[i] * 2;
```

Thread Sync: Barrier-based



```
for(i=0; i < 1000; i++)
    a[i] = a[i] * 2;
```

Busy wait

```
for(i=0; i < 1
    a[i] = a[i]
```

Busy wait

Trigger via shared variable

```
for(i=0; i < 1000; i++)
    a[i] = a[i] * 2;
```

Outline

Here I need to say that now I am Introducing the barriers we tested!

- Motivation
- Type of parallel regions
- **Barriers**
- Test Applications
- Results
- Conclusions

Master-Threads

Pasted in, badly formatted C code
Don't do this ;-)
Use pseudocode!

```
volatile int jobCycle = 0;

void masterBarrier(int tid, int n)
{
   jobCycle = !jobCycle;
   executeWork(tid, n);
   masterSync(tid, n);
}
```

Worker-Thread

```
void workerThread(int tid, int n)
{
  int mycycle = 0;
  while(1)
   {
     while(myCycle == jobCycle);
     myCycle = jobCycle;
     executeWork(tid, n);
     workerSync(tid, n);
   }
}
```

Lock-Free

```
void workerSync(int tid, int n)
{
   barrierBuffer[tid] = 1;
}

void masterSync(int tid, int n)
{
   int i, sum;
   do
      {
        for(i = 1, sum = 1; i < n; i++)
           sum += barrierBuffer[i];
      }
   while(sum < n);
}
```

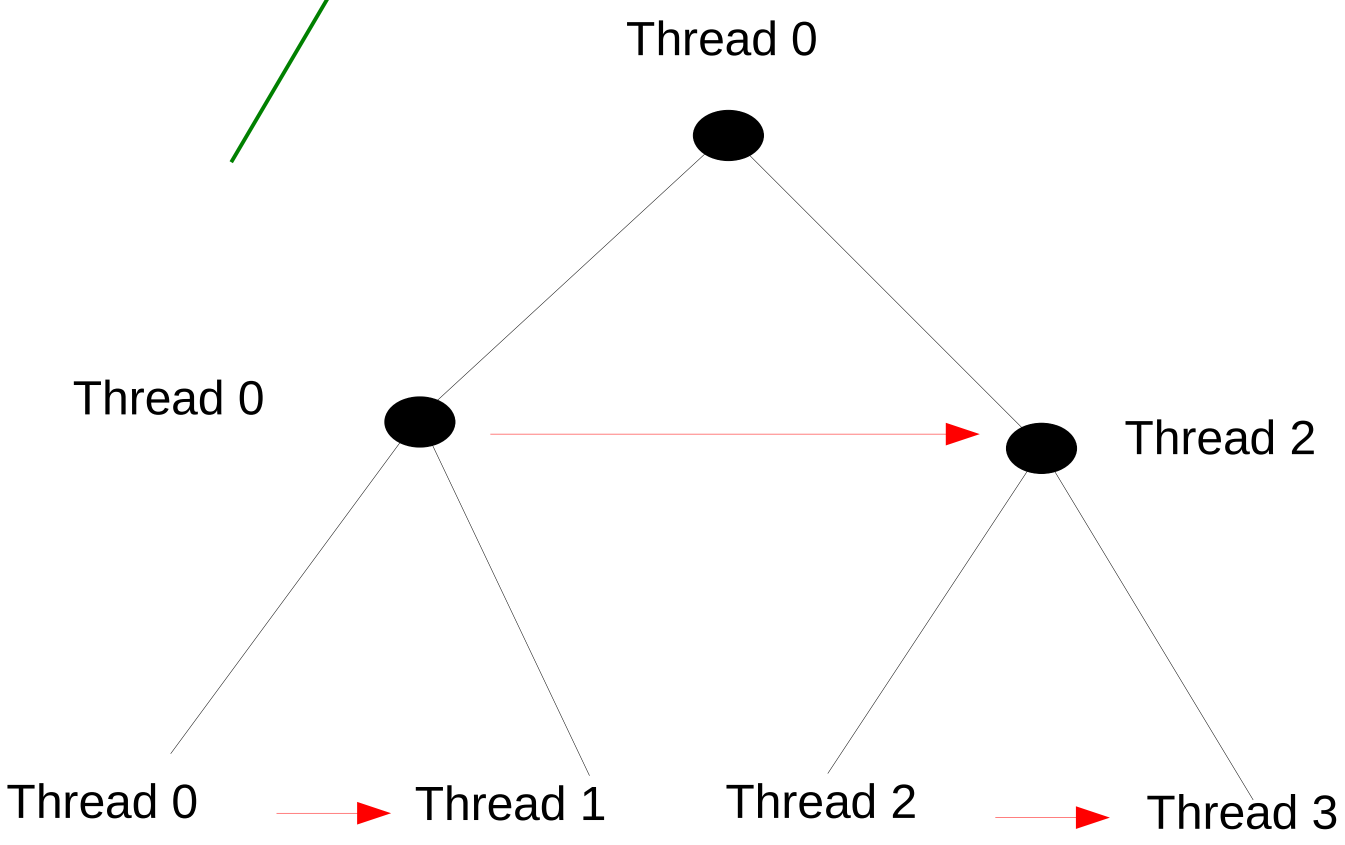
Lock-Free Padded

```
void workerSync(int tid, int n)
{
   barrierBuffer[tid * padding] = 1;
}

void masterSync(int tid, int n)
{
   int i, sum;
   do
      {
        for(i = 1, sum = 1; i < n; i++)
           sum += barrierBuffer[i * padding];
      }
   while(sum < n);
}
```

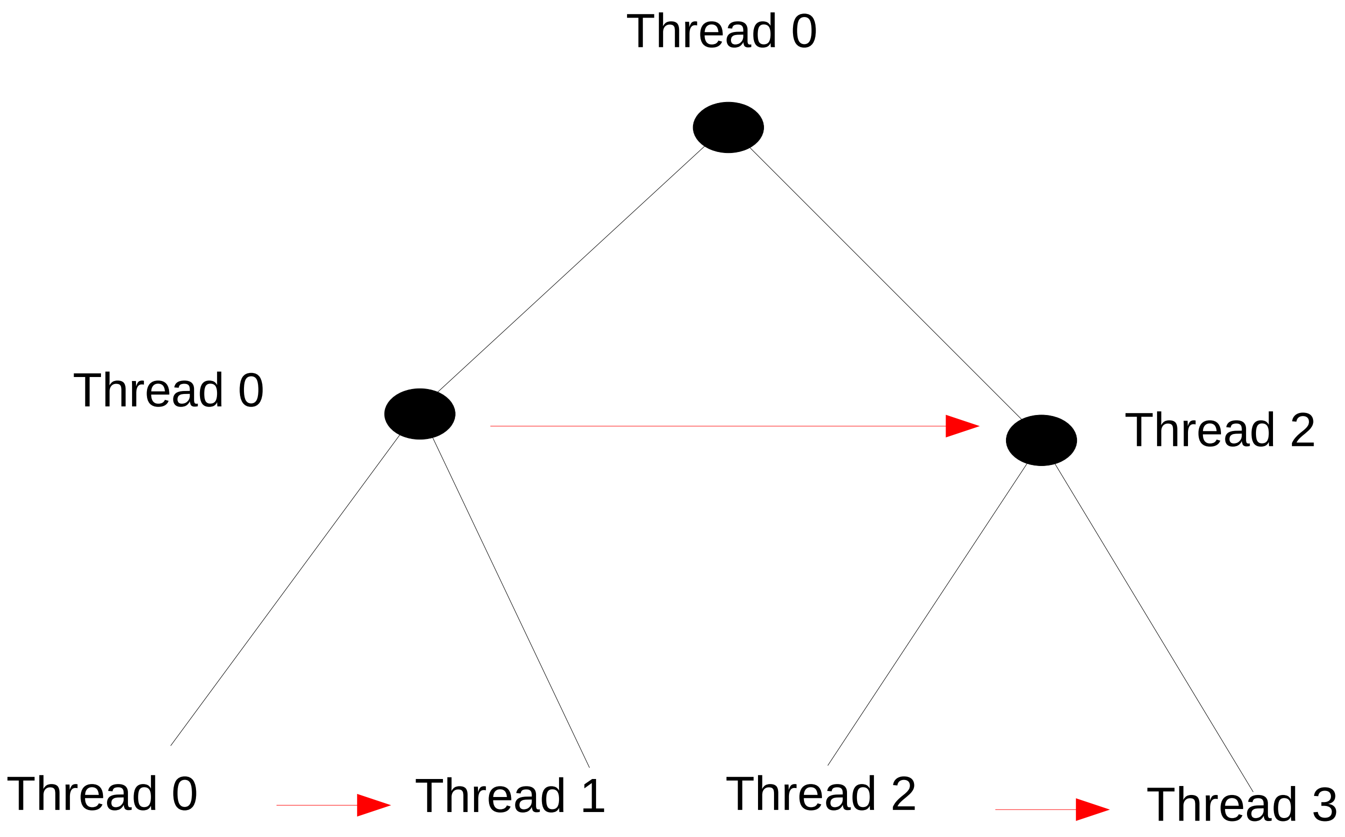
I should have provided an overview of the barrier implementations we Tested at the beginning of this part!

Recursive Lock-Free

That's a slide I like
One could have animated it though!



Recursive Lock-Free Padded



Intrinsic Atomic Increment

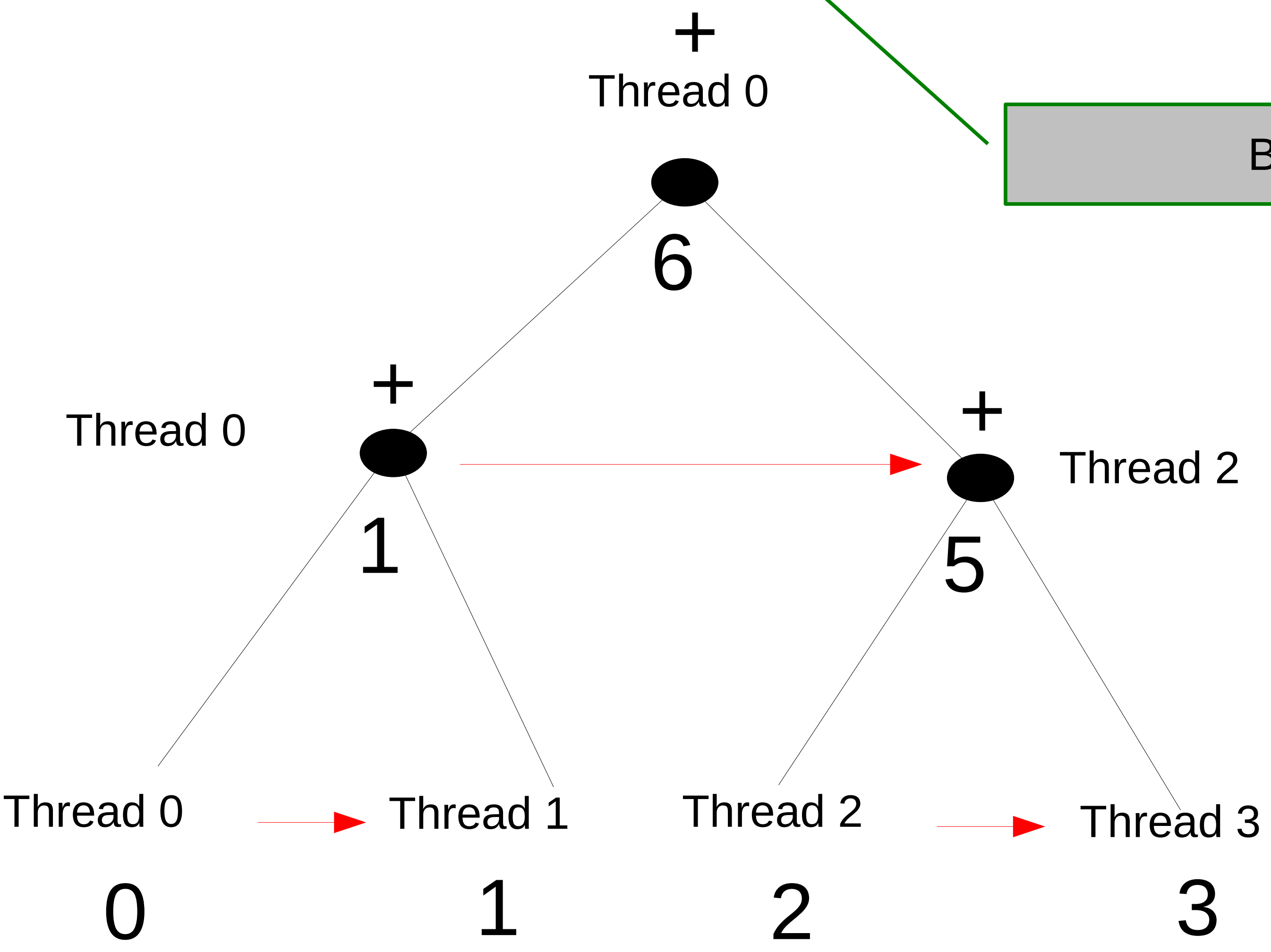
```
volatile int counter = 0;

void workerSync(int tid, int n)
{
    __sync_fetch_and_add(&counter, 1);
}

void masterSync(int tid, int n)
{
    int workers = n - 1;

    while(counter != workers);
    counter = 0;
}
```

Lock-Based

```
volatile int counter = 0;

void workerSync(int tid, int n)
{
        pthread_mutex_lock(&mutexCounter);
        counter++;
        pthread_mutex_unlock(&mutexCounter);
}

void masterSync(int tid, int n)
{
        int workers = n - 1;

        while(counter != workers);
        counter = 0;
}
```

Reduction Flavors

- Recursive “classic reduction”



Badly formatted!

Flat Reduction

- Threads store partial sums in shared array
- Master conducts reduction alone after barrier
- Tested with and without SSE3 vectorization
- Assumes that only one or two simple reductions are computed, e.g., reduction function is -,*,+

Too much text
I could have added a graphical representation here as well

Outline

- Motivation
- Type of parallel regions
- Barriers
- **Test Applications**
- Results
- Conclusions

Test Applications

I don't say what I mean by workload here!

- Synthetic Benchmarks
  - Without workload
  - With workload
- Real Benchmark
  - RAxML Bioinformatics application

Synthetic Benchmarks

Badly formatted!

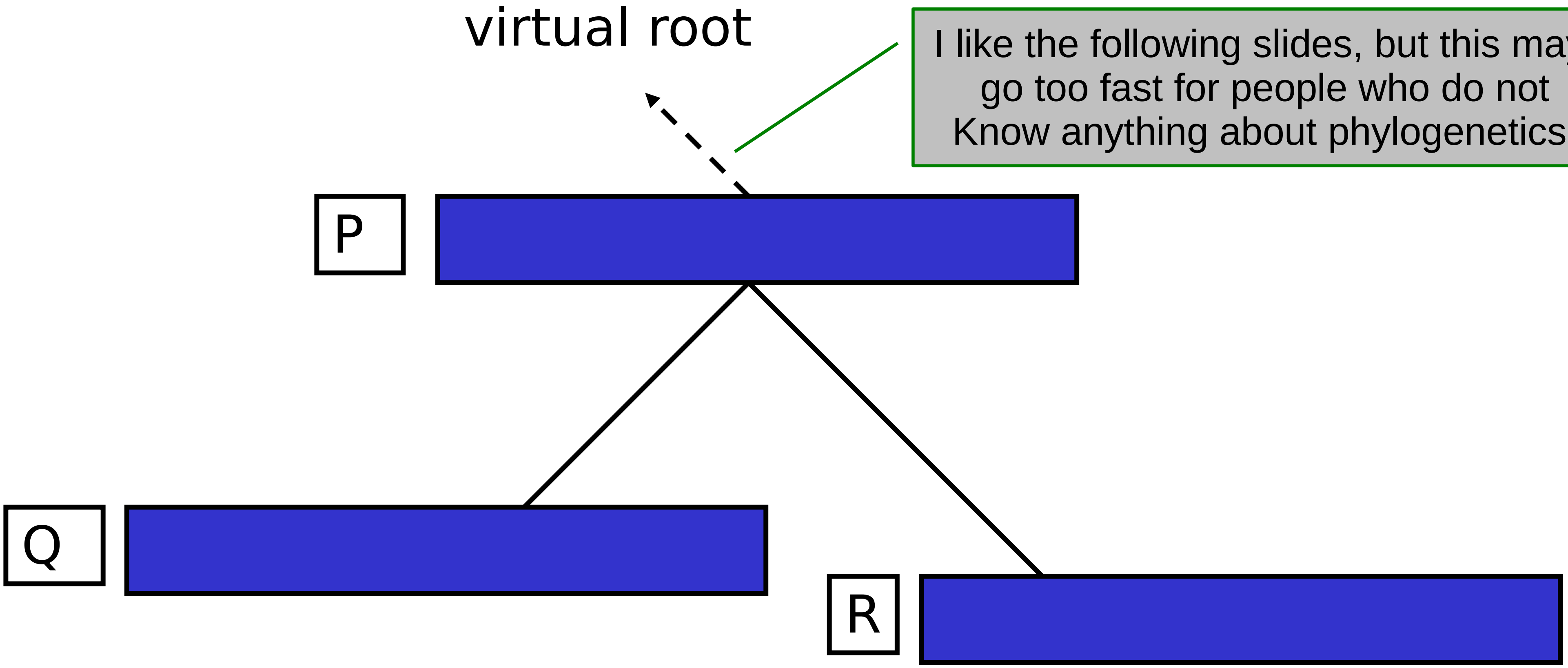
- With workload
  - 3 arrays v1, v2, v3 of length *M*
  - compute “v3[i] = v1[i] * v2[i]” *N* times with intermittent barriers
- Without workload
  - Set *M* to zero :-)
- Variables *N* and *M* are set at compile time
- Use of mmap() to control array allocation
- Cache utilization controlled by *M*

Real Workload

- Pthreads parallelization of RAxML
- RAxML: widely used tool for inference of evolutionary trees from DNA data
- Fine-grain production-level Pthreads parallelization
- Floating-point and memory intensive
- Considered subset of the phylogenetic likelihood function:
  - requires largest amount of sync
  - Two reductions on 1st and 2nd derivative of the likelihood (Newton-Raphson procedure)

Too much & partially unnecessary text

Loop Level Parallelism

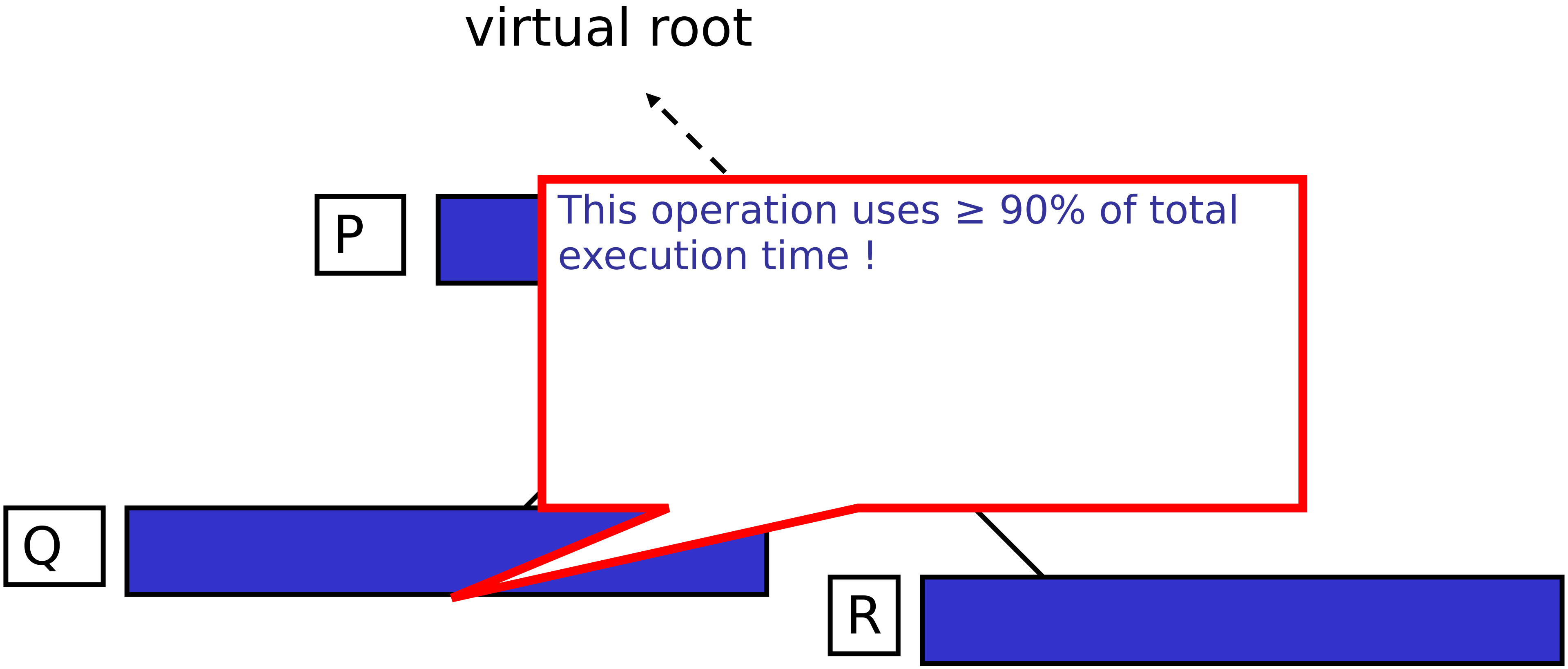


I like the following slides, but this may go too fast for people who do not Know anything about phylogenetics!

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

This should pop up on a second slide

Loop Level Parallelism



virtual root

P

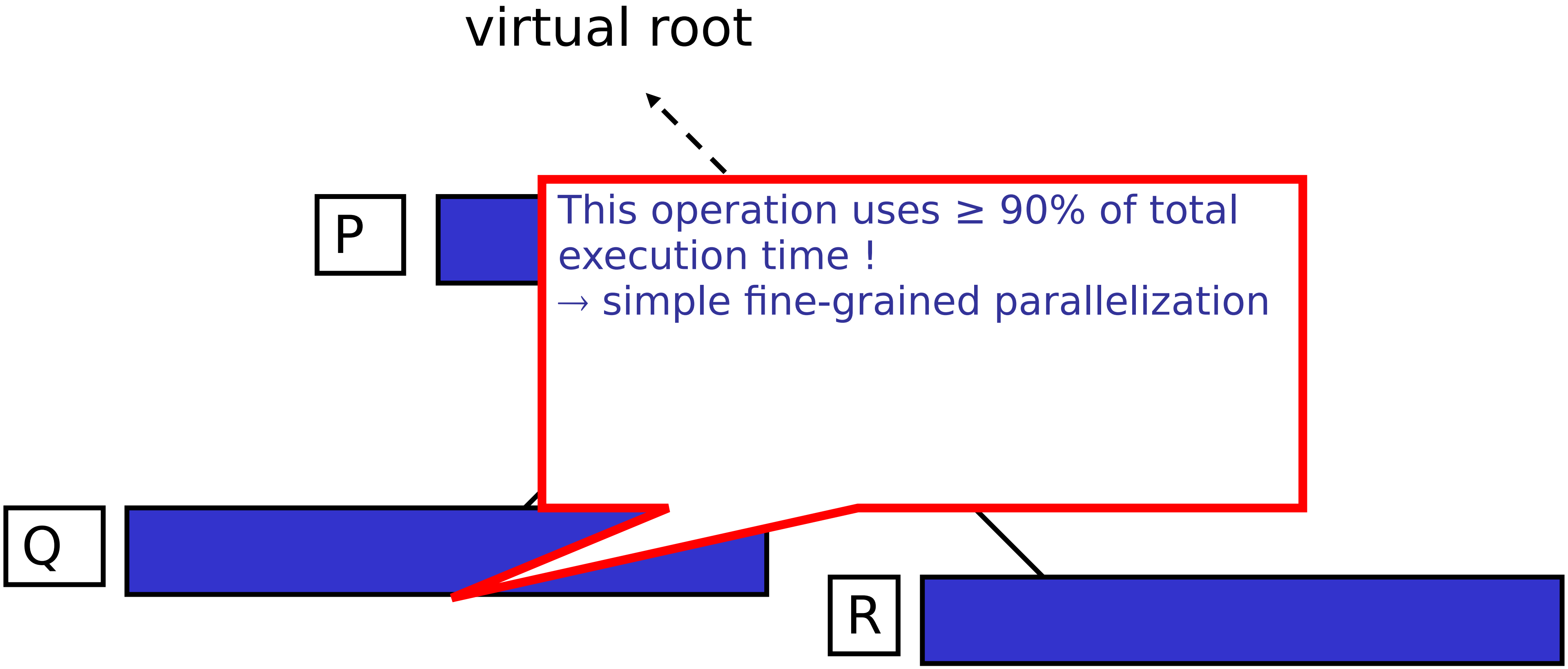
Q

R

This operation uses ≥ 90% of total execution time !

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

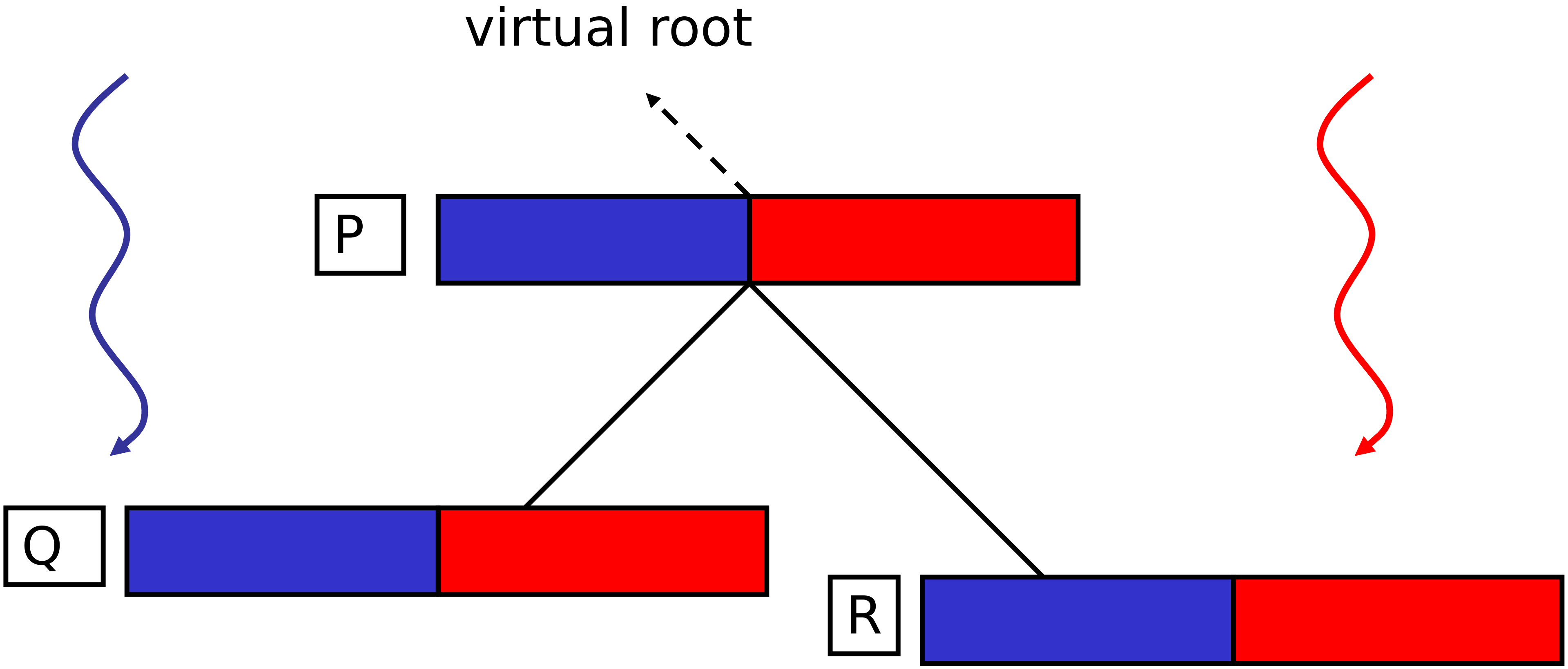
Loop Level Parallelism



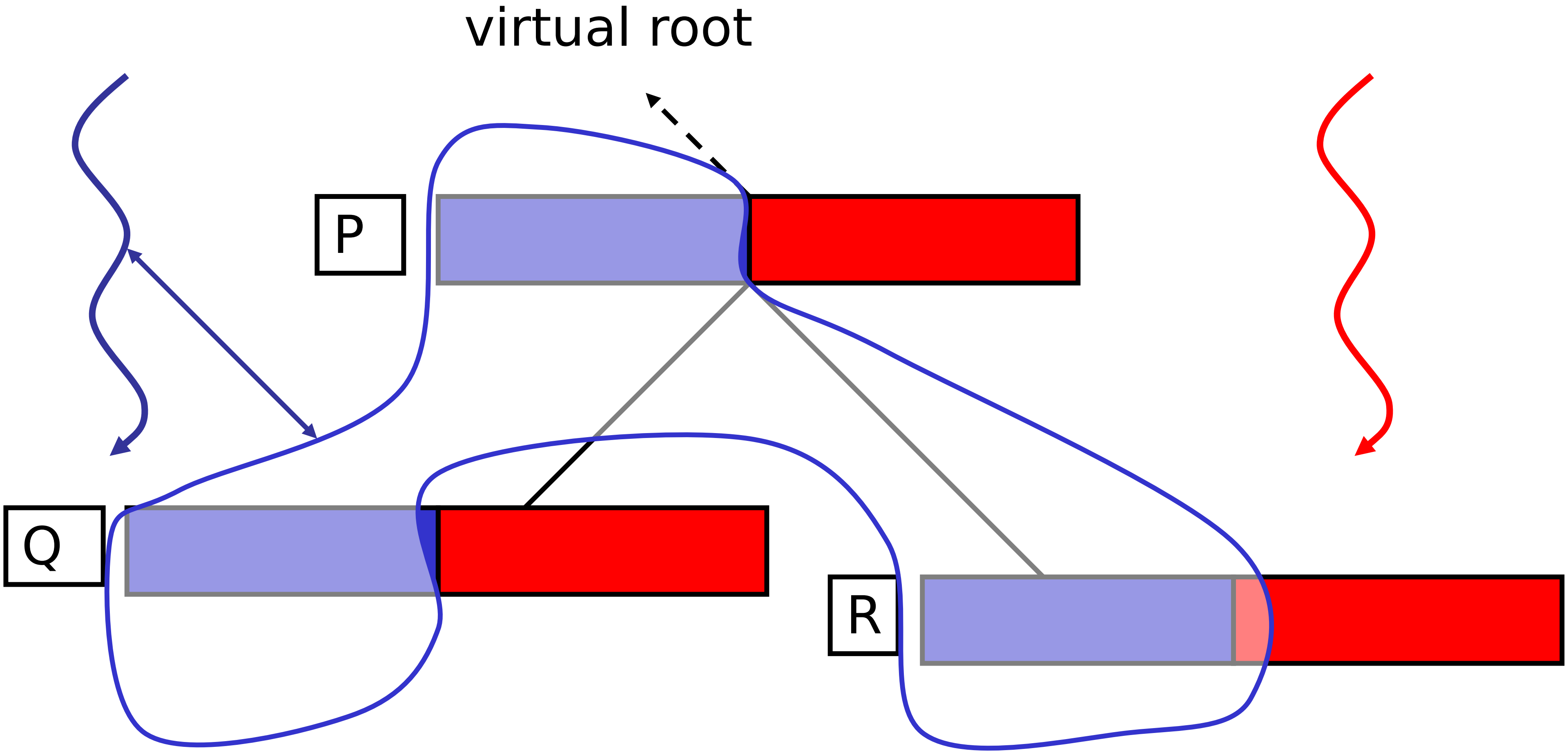
This operation uses ≥ 90% of total execution time !
→ simple fine-grained parallelization

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

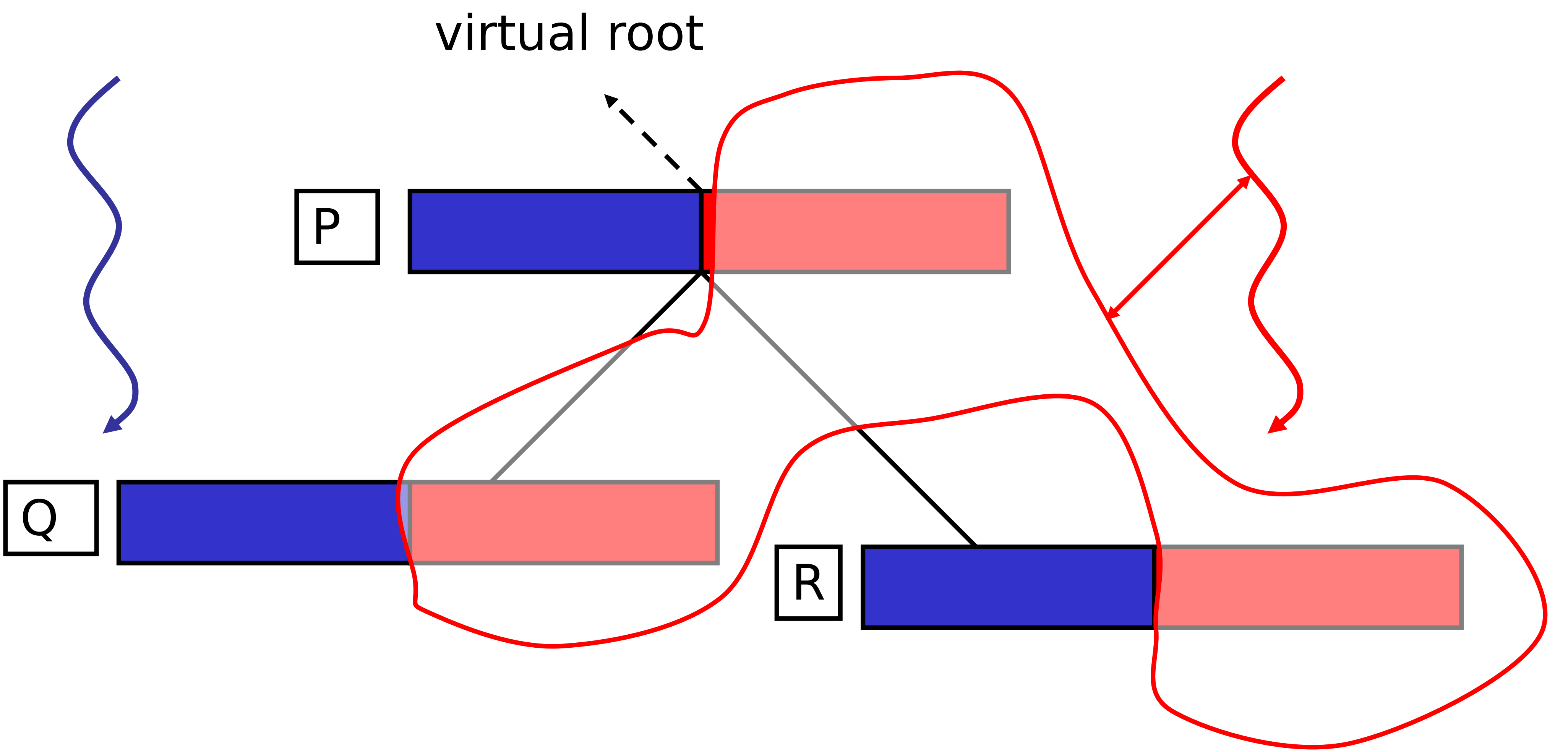
Loop Level Parallelism



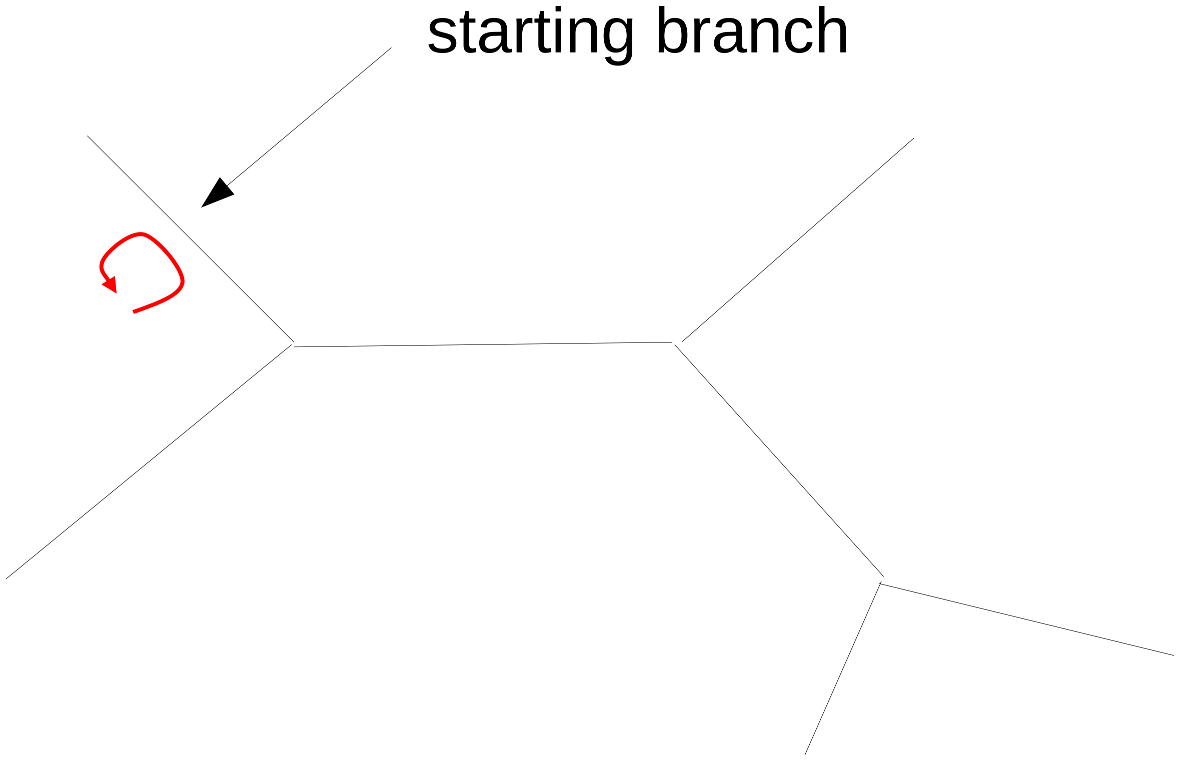
Loop Level Parallelism



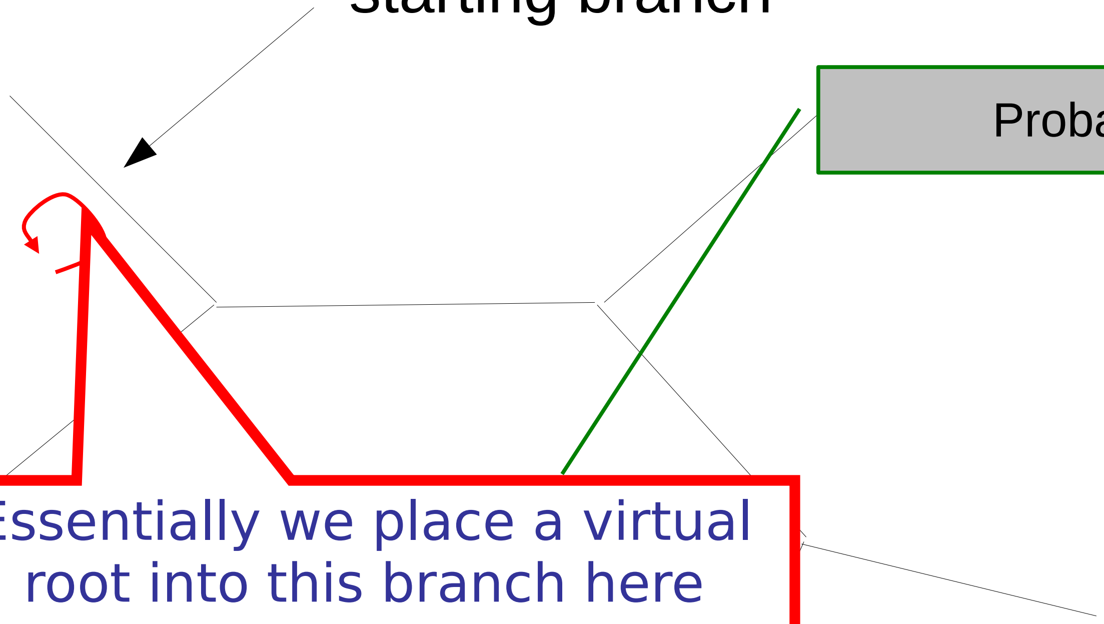
Loop Level Parallelism



Branch Length Optimization



Branch Length Optimization

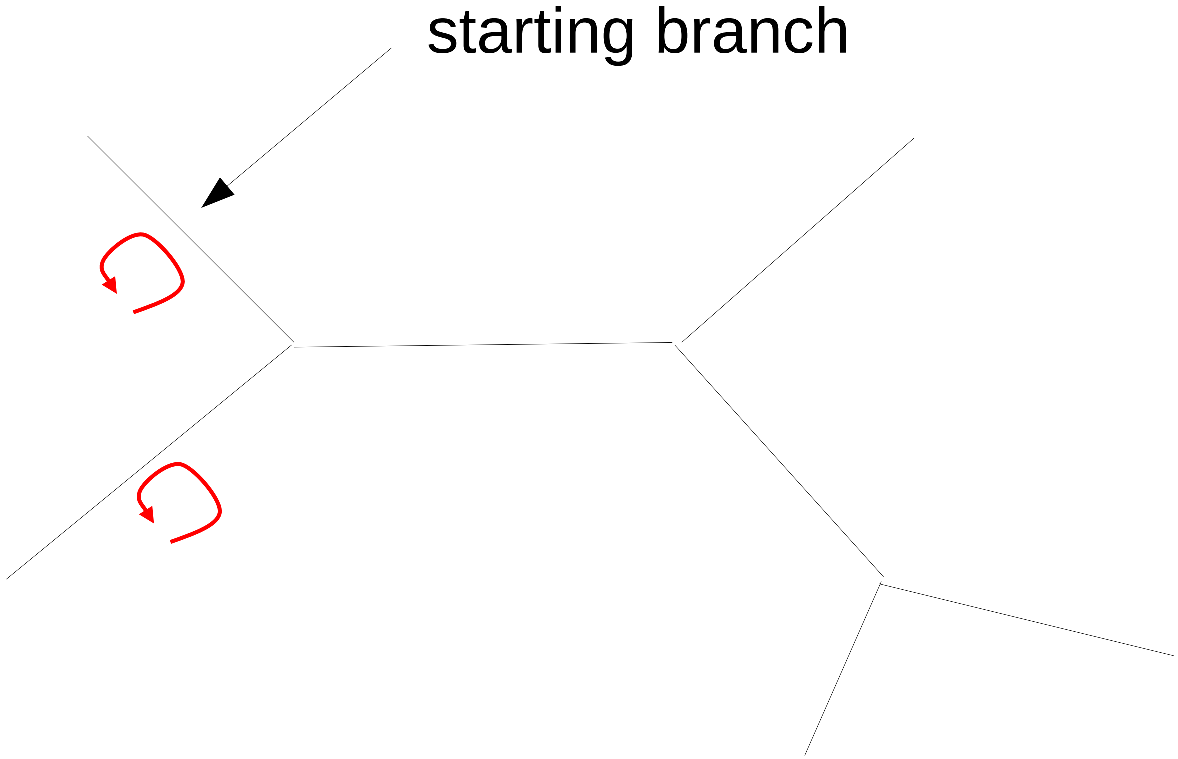


starting branch

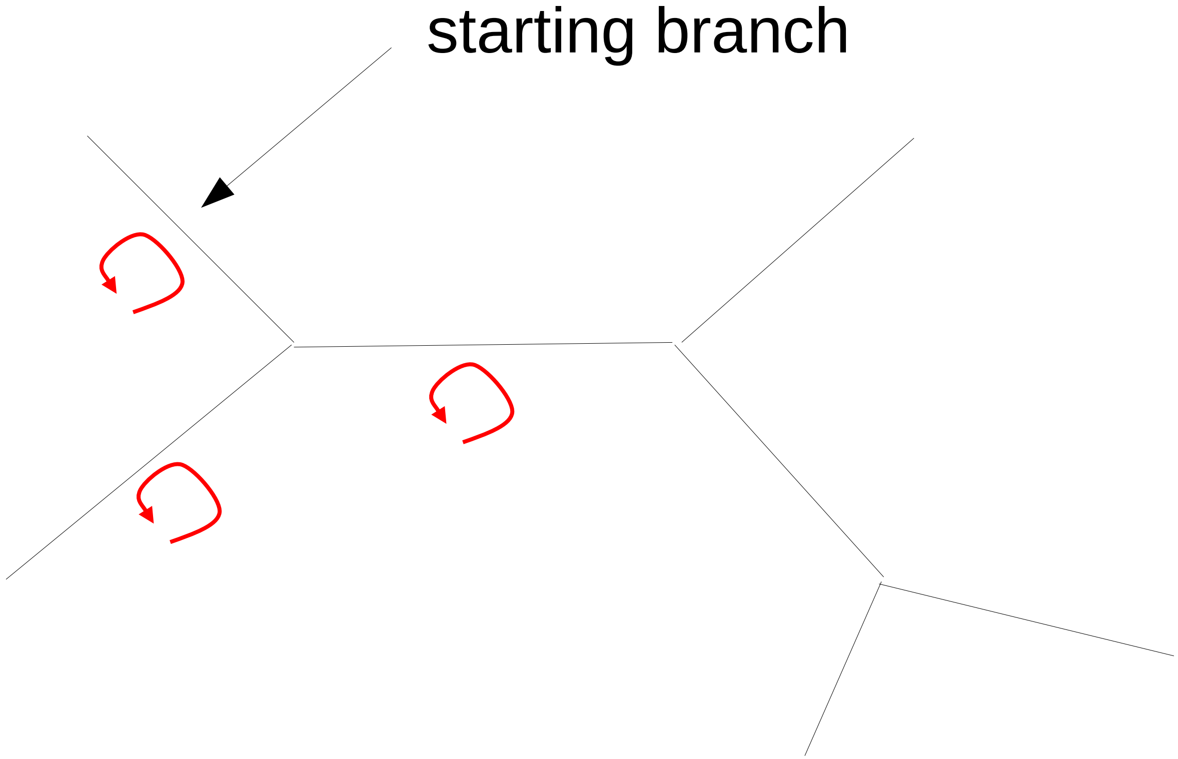
Probably not necessary!

Essentially we place a virtual root into this branch here

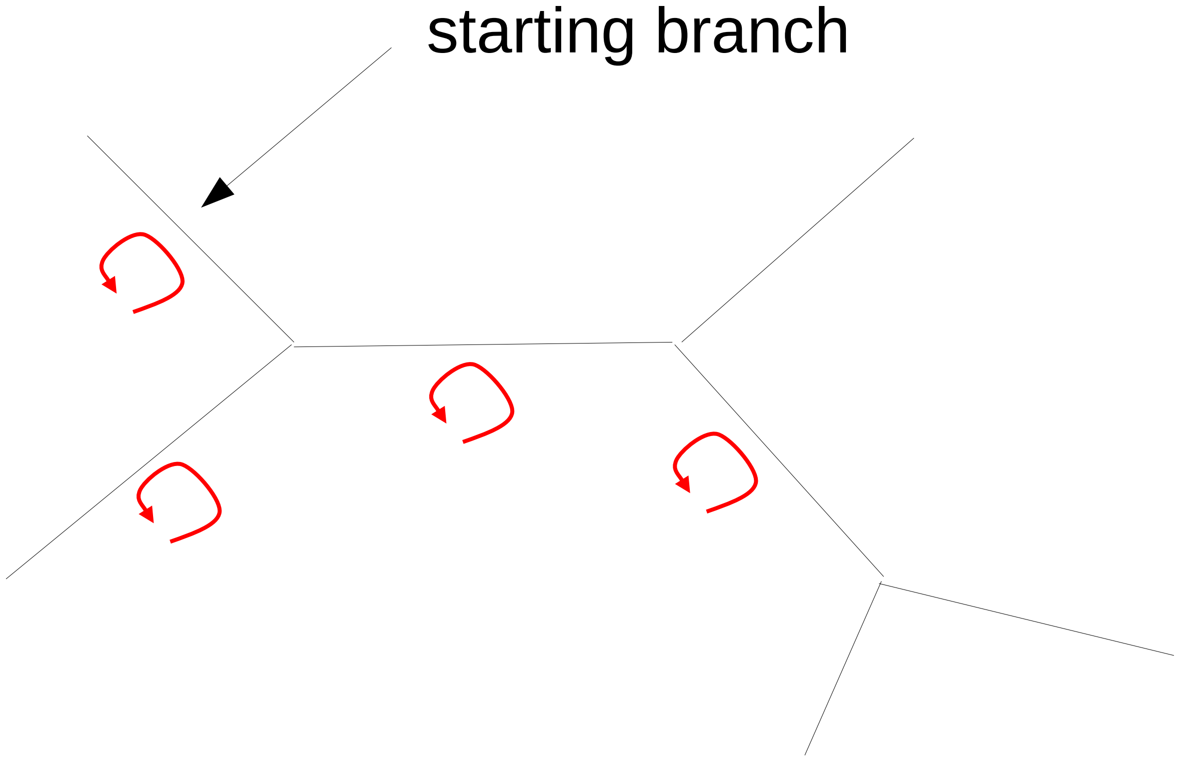
Branch Length Optimization



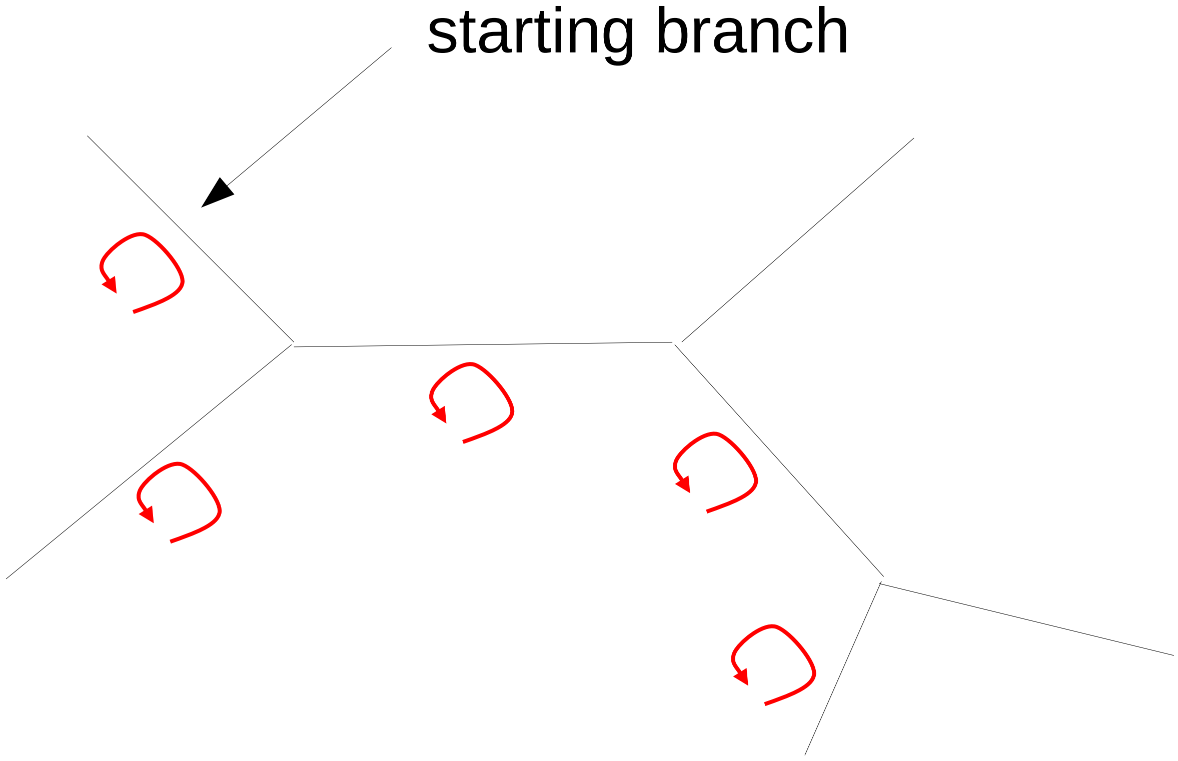
Branch Length Optimization



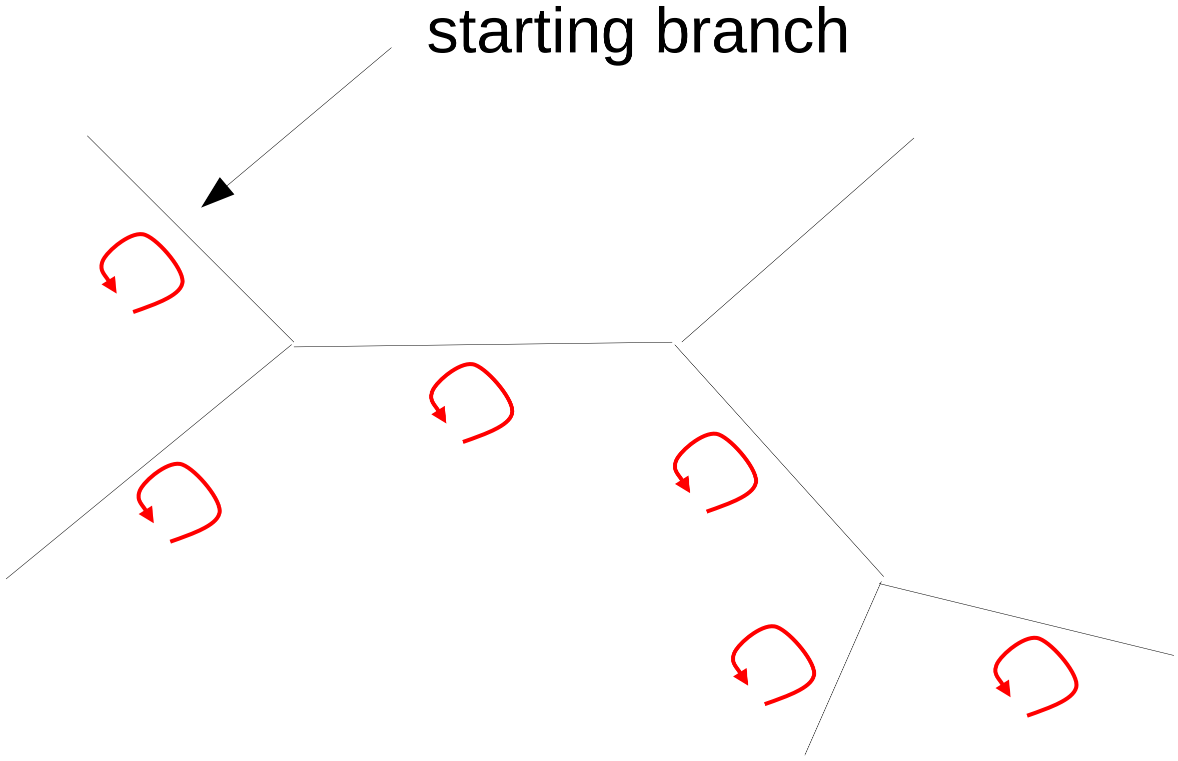
Branch Length Optimization



Branch Length Optimization

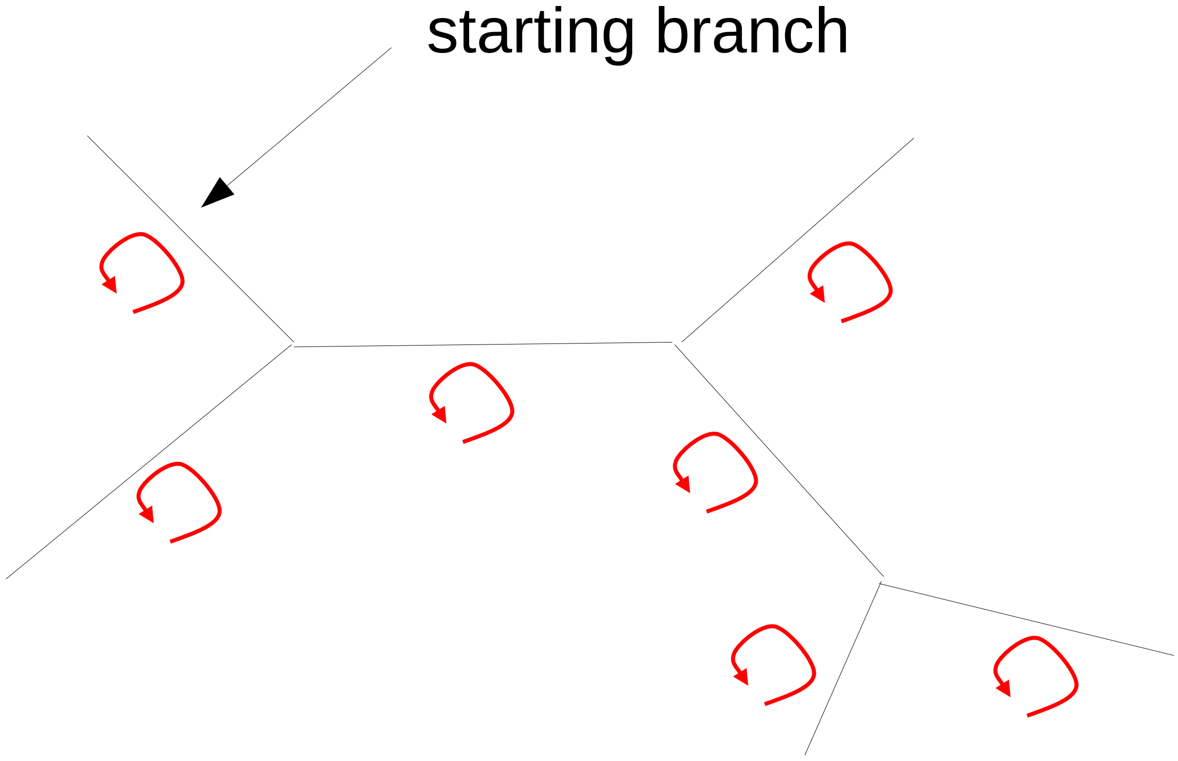


Branch Length Optimization

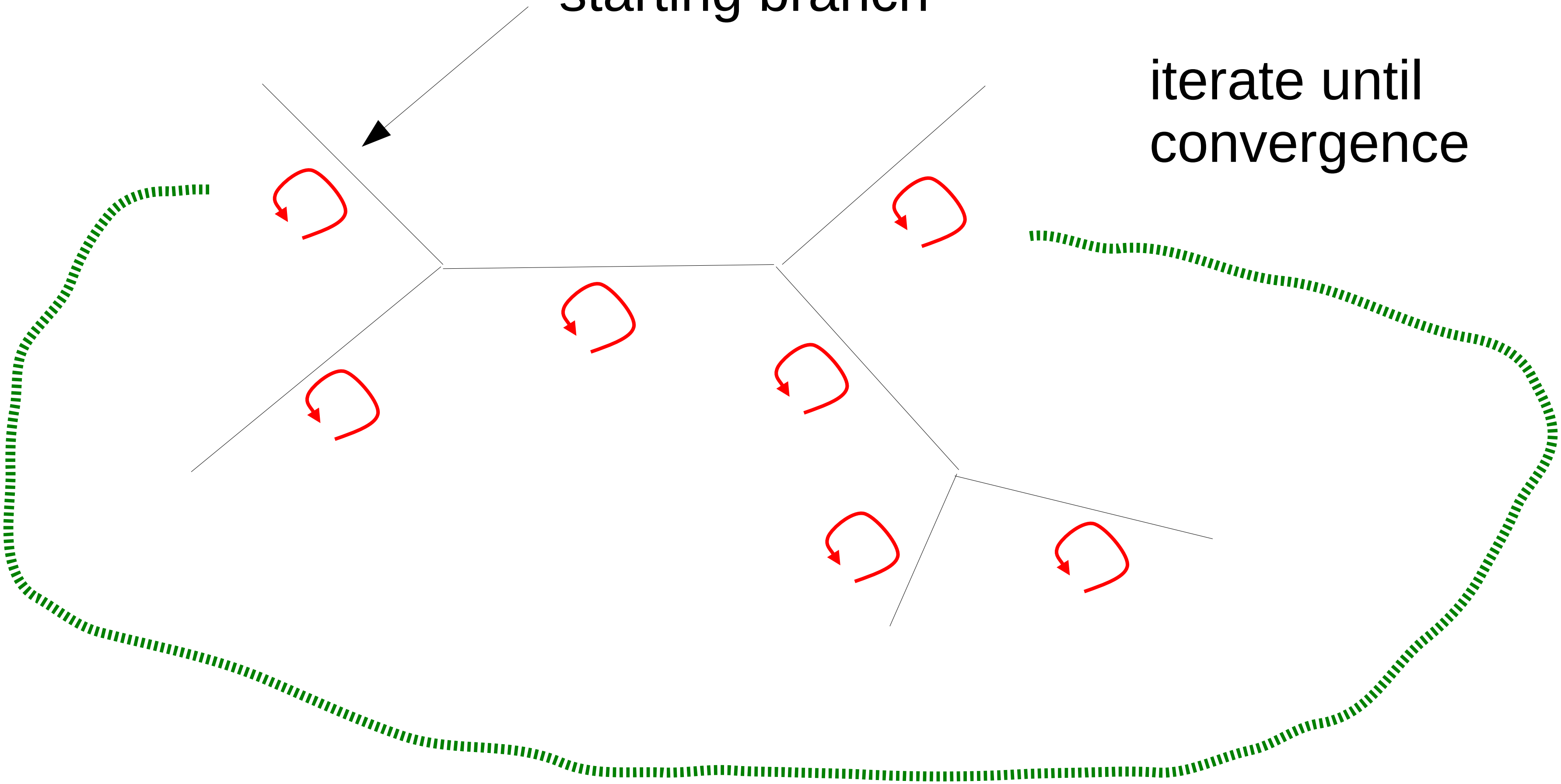


starting branch

Branch Length Optimization



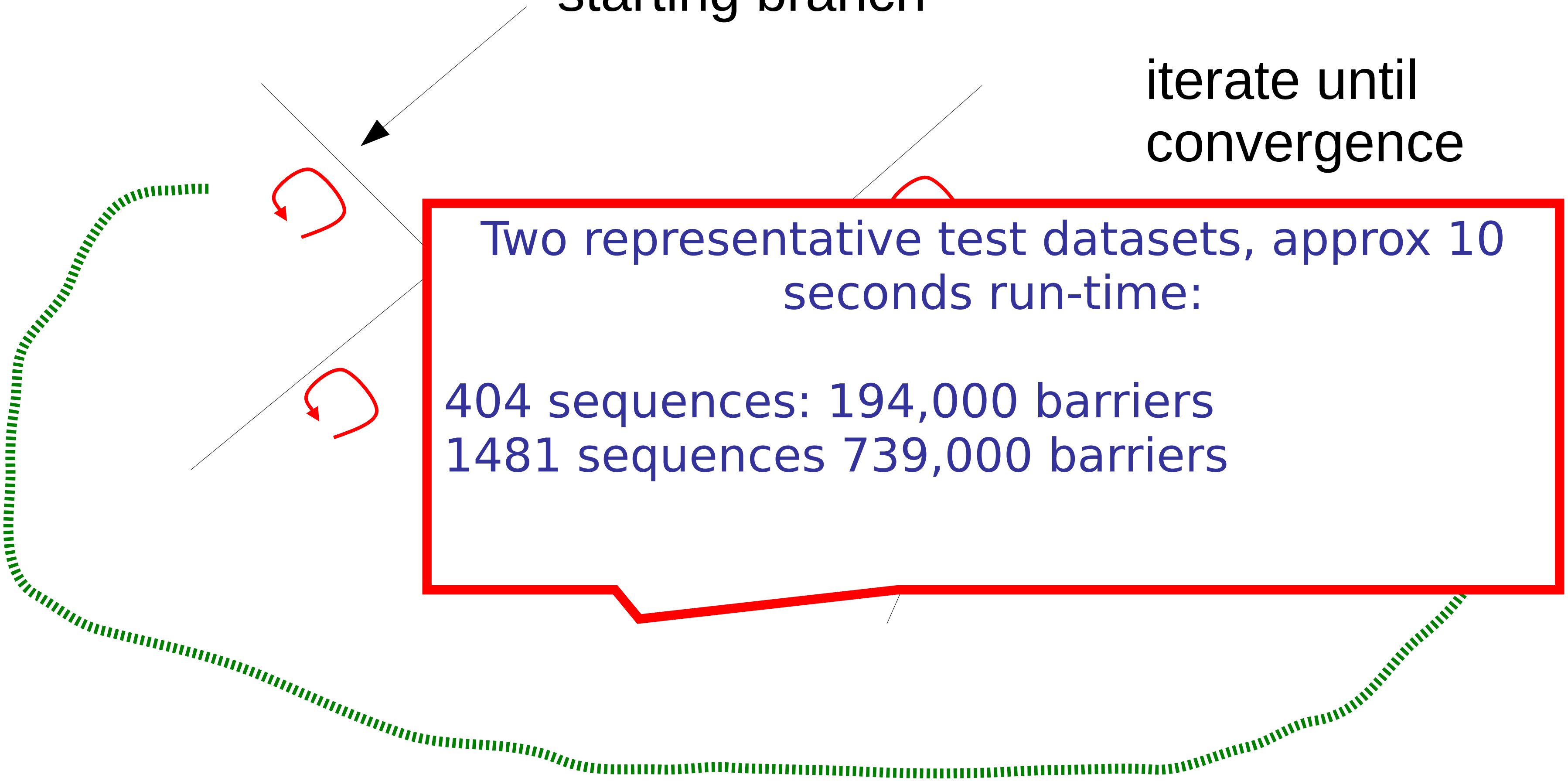
Branch Length Optimization



starting branch

iterate until convergence

Branch Length Optimization



starting branch

iterate until convergence

Two representative test datasets, approx 10 seconds run-time:

404 sequences: 194,000 barriers
1481 sequences 739,000 barriers

Outline

- Motivation
- Type of parallel regions
- Barriers
- Test Applications
- **Results**
- Conclusions

Test Platforms

- 2-core Intel Core2 Duo
- 4-core Intel Core 2 Quad
- 4-core Intel Core i5
- 8-core Intel Nehalem
- 16-core AMD Barcelona
- 32-core AMD Sun x4600
- 24-core AMD Sun x4440

The list looks ugly! We need the details here.
I should have sub-divided it
into Intel and AMD systems!

Synthetic without workload

This is something you should never do ;-)
→ represent numerical results graphically!

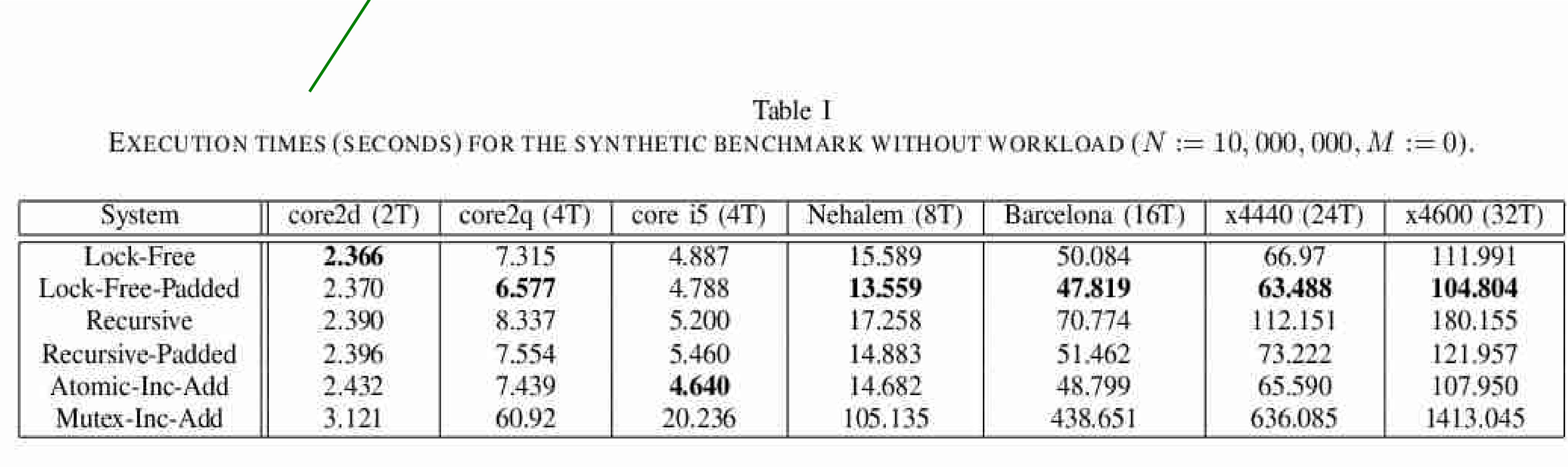


Table I

EXECUTION TIMES (SECONDS) FOR THE SYNTHETIC BENCHMARK WITHOUT WORKLOAD ($N := 10,000,000, M := 0$).

System	core2d (2T)	core2q (4T)	core i5 (4T)	Nehalem (8T)	Barcelona (16T)	x4440 (24T)	x4600 (32T)
Lock-Free	**2.366**	7.315	4.887	15.589	50.084	66.97	111.991
Lock-Free-Padded	2.370	**6.577**	4.788	**13.559**	**47.819**	**63.488**	**104.804**
Recursive	2.390	8.337	5.200	17.258	70.774	112.151	180.155
Recursive-Padded	2.396	7.554	5.460	14.883	51.462	73.222	121.957
Atomic-Inc-Add	2.432	7.439	**4.640**	14.682	48.799	65.590	107.950
Mutex-Inc-Add	3.121	60.92	20.236	105.135	438.651	636.085	1413.045

Synthetic with workload

This is something you shouldn't do ;-)

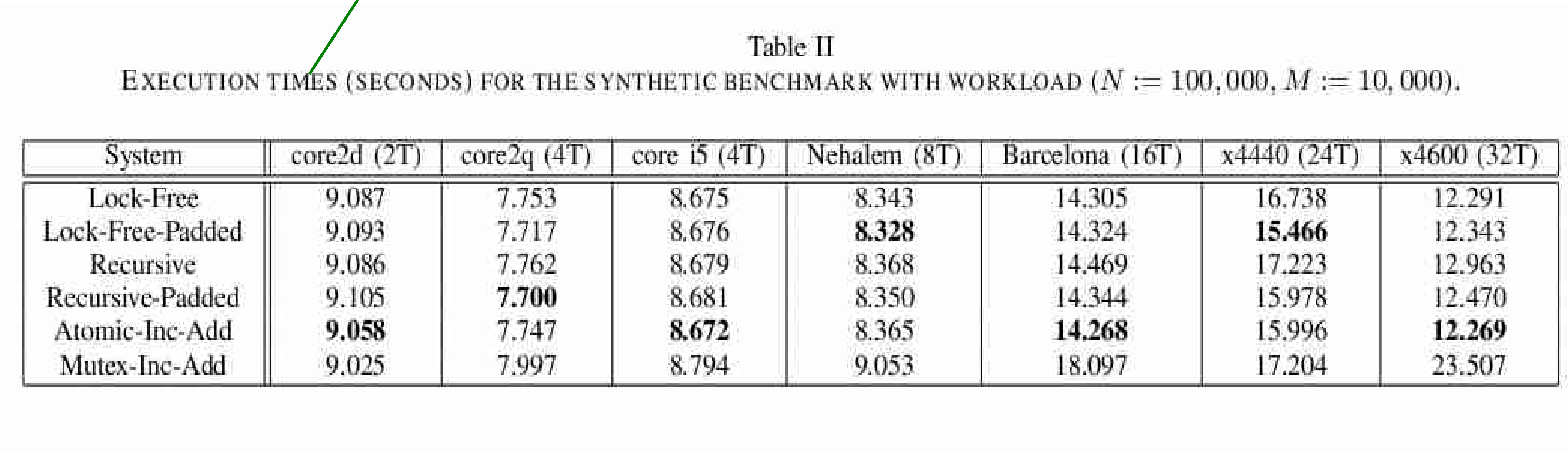


Table II

EXECUTION TIMES (SECONDS) FOR THE SYNTHETIC BENCHMARK WITH WORKLOAD ($N := 100,000$, $M := 10,000$).

System	core2d (2T)	core2q (4T)	core i5 (4T)	Nehalem (8T)	Barcelona (16T)	x4440 (24T)	x4600 (32T)
Lock-Free	9.087	7.753	8.675	8.343	14.305	16.738	12.291
Lock-Free-Padded	9.093	7.717	8.676	**8.328**	14.324	**15.466**	12.343
Recursive	9.086	7.762	8.679	8.368	14.469	17.223	12.963
Recursive-Padded	9.105	**7.700**	8.681	8.350	14.344	15.978	12.470
Atomic-Inc-Add	**9.058**	7.747	**8.672**	8.365	**14.268**	15.996	**12.269**
Mutex-Inc-Add	9.025	7.997	8.794	9.053	18.097	17.204	23.507

RAxML 404 sequences

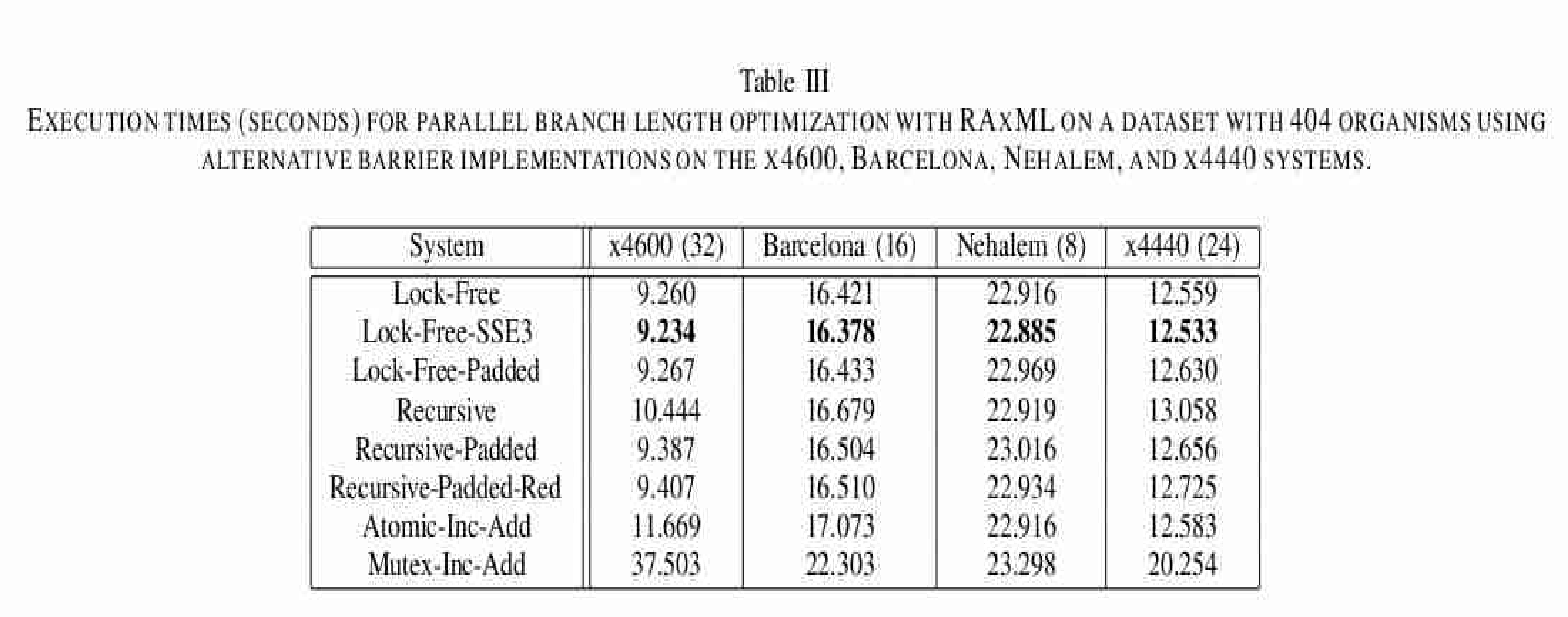


Table III

EXECUTION TIMES (SECONDS) FOR PARALLEL BRANCH LENGTH OPTIMIZATION WITH RAXML ON A DATASET WITH 404 ORGANISMS USING ALTERNATIVE BARRIER IMPLEMENTATIONS ON THE X4600, BARCELONA, NEHALEM, AND X4440 SYSTEMS.

System	x4600 (32)	Barcelona (16)	Nehalem (8)	x4440 (24)
Lock-Free	9.260	16.421	22.916	12.559
Lock-Free-SSE3	**9.234**	**16.378**	**22.885**	**12.533**
Lock-Free-Padded	9.267	16.433	22.969	12.630
Recursive	10.444	16.679	22.919	13.058
Recursive-Padded	9.387	16.504	23.016	12.656
Recursive-Padded-Red	9.407	16.510	22.934	12.725
Atomic-Inc-Add	11.669	17.073	22.916	12.583
Mutex-Inc-Add	37.503	22.303	23.298	20.254

RAxML 1481 Sequences

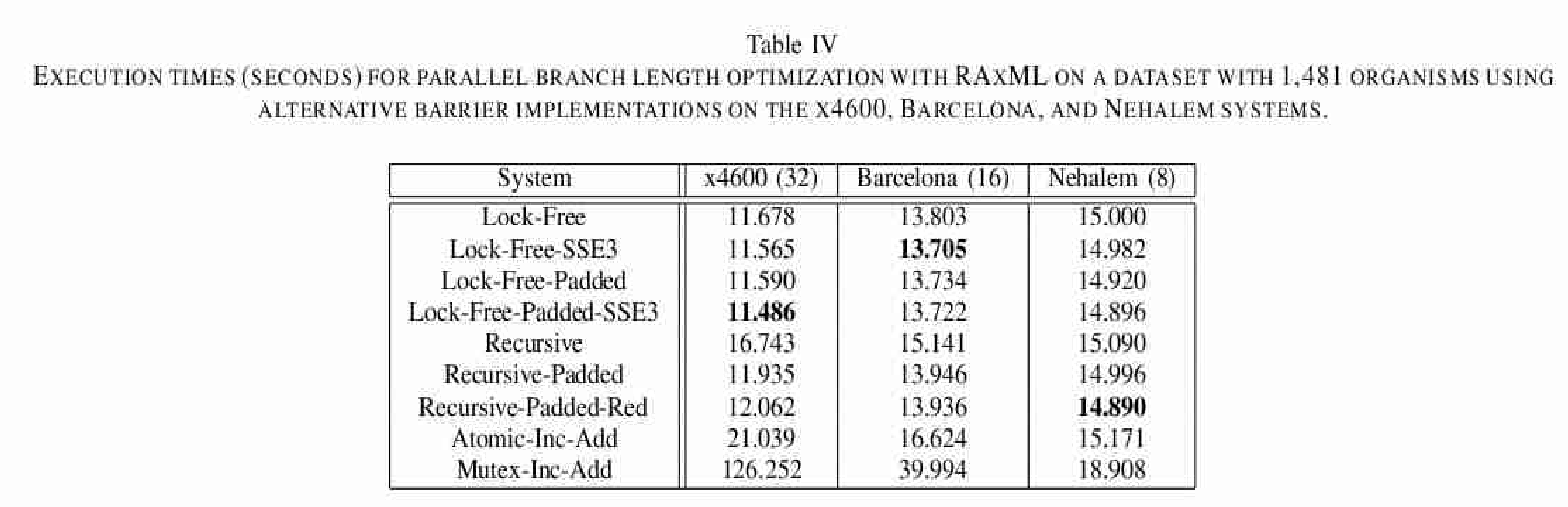


Table IV

EXECUTION TIMES (SECONDS) FOR PARALLEL BRANCH LENGTH OPTIMIZATION WITH RAxML ON A DATASET WITH 1,481 ORGANISMS USING ALTERNATIVE BARRIER IMPLEMENTATIONS ON THE X4600, BARCELONA, AND NEHALEM SYSTEMS.

System	x4600 (32)	Barcelona (16)	Nehalem (8)
Lock-Free	11.678	13.803	15.000
Lock-Free-SSE3	11.565	**13.705**	14.982
Lock-Free-Padded	11.590	13.734	14.920
Lock-Free-Padded-SSE3	**11.486**	13.722	14.896
Recursive	16.743	15.141	15.090
Recursive-Padded	11.935	13.946	14.996
Recursive-Padded-Red	12.062	13.936	**14.890**
Atomic-Inc-Add	21.039	16.624	15.171
Mutex-Inc-Add	126.252	39.994	18.908

Outline

- Motivation
- Type of parallel regions
- Barriers
- Test Applications
- Results
- **Conclusions**

Conclusions

- Intrinsic atomic increment does not yield optimal performance
- Performance of different barrier flavors depends on cache utilization of application
- Lock-free barriers and SSE3-based flat reductions appear to work best across all platforms

Again: too much text!

Acknowledgments

Simon Berger, TUM

Denis Krompaß, TUM

Nikos Alachiotis, TUM

Stephen Smith, Brown/TUM

Nick Pattengale, Sandia

Michael Ott, TUM

Andre Aberer,
TUM

Fernando Izquierdo,
TUM

Christian v. Mering & Manuel Stark
University of Zürich

Wayne Pfeiffer, SDSC

Thank you for your Attention !

Zakros, Crete, Greece, September 2008

Summary

- This was an “okay” presentation, but
- Too much text
- Sometimes lacking a clearer structure!
- Could have included more pictures
- Result section

→ copied & pasted tables from paper

→ don't do this

Today

- How to give a scientific presentation
- **How to write a technical report/scientific paper**

General structure

- Meaningful (catchy?) Title, catchy title, for instance:

  “Short tree, long tree, right tree, wrong tree: new acquisition bias corrections for inferring SNP phylogenies”

- Authors and affiliations
- Abstract
  - Motivation
  - Problem Statement
  - Own Contribution
  - Results
- Introduction and Motivation

General structure

- Related work (can also be moved further to the end)
- Own contribution, e.g.,
  - Algorithm
  - Parallelization
  - Model
- Results
  - Experimental setup
    - How were test datasets generated?
    - What kind of HW was used?
    - How can the experiments be reproduced?
      → reproducible science
      → make SW and datasets available for download!
      → Archive the data for 10 years!
      → Archive data when the paper is accepted for publication!

General structure

- Results
  - Experimental Setup
    - Datasets used
    - HW platforms used
    - Compilers used
  - Results
    - Graphs, speedup plots, tables
    - Comparison to competing SW, algorithms etc
- Conclusion and Future Work
- Acknowledgements
  - Funding agencies
  - Colleagues who helped

Biology Papers

- The structure is a bit different
- Abstract
- Introduction
- Materials & Methods (e.g., field work, sequencing, bioinformatics analysis, etc.)
- Results
- Discussion ← this is longer and more important than in CS, because results are typically more fuzzy
- Acknowledgements

Writing papers

- Be exact & precise
- Be exact & precise
- Be exact & precise
- Omit unnecessary information
- Don't be wordy, be concise
- Use **short sentences**!
- Avoid colloquial expressions!
- Avoid qualitative words: much, little, good, few
- Quantify things!
- Don't say “in most cases our code performed well” → “in 65% of the cases our code showed an accuracy improvement exceeding 5% over ...”

Writing papers

- In engineering-style papers
- Always provide a rationale for design choices!

  Instead of “We use an array representation with binary search for storing and retrieving elements.” → “We use an array representation with binary search for storing and retrieving elements, because binary search trees performed worse for the problem at hand.”
- In English: Sentences are generally much shorter
- Don't show that you are educated (as sometimes done in German) by writing long sentences using elaborate vocabulary
- **Keep Things Simple!**

Writing papers

- If your English is generally mediocre don't build in “100 Dollar words” you looked up in the dictionary → this just sounds ridiculous
- Occasional language jokes are allowed, e.g., using phrases such as “Based on the prolegomena”
- Use a spell checker
- Use a spell checker
- Use a spell checker

Writing papers

- In Latex add a ~ after . in the middle of a phrase
- Properly introduce acronyms
- Say what acronyms mean the first time you use them
- Use/introduce acronyms consistently to make the text shorter
- E.g., multiple sequence alignment → MSA
- If you use a long term frequently “phylogenetic placement of query sequences” find a shorter one “henceforth, denoted as 'query placement' “

Writing papers

- Don't use qualitative terms like “very, highly, significantly (in the non-statistical sense), much, good, bad” quantify everything as much as you can
- I personally don't like the passive form, that is, write “We implemented a cool software” instead of “a cool software has been implemented”
- “cool” shouldn't be used of course
- This also makes it much clearer what your own contribution was and what has be done by others/what is prior knowledge & work

Writing papers

- Before handing in your reports/papers use an academic “writing checker” that catches the most common mistakes!
- Academic-Writing-Check: https://github.com/devd/Academic-Writing-Check
- Make sure you know what you are doing with Latex!
- **Please read these pages!**
  - http://www.ece.ucdavis.edu/~jowens/commonerrors.html
  - http://www.cs.columbia.edu/~hgs/etc/writing-bugs.html
  - http://www.you-can-teach-writing.com/grammar_websites.html
- Attend a scientific writing course → it's worth it!

Google Scholar

- Attention when importing .bib entries from Google Scholar
- Don't just copy & paste
- You need to check the entries
  - All data available?
  - Correct Journal/Conference Abbreviations
  - Entries missing?
- I will check the reports for correct bibliography data!