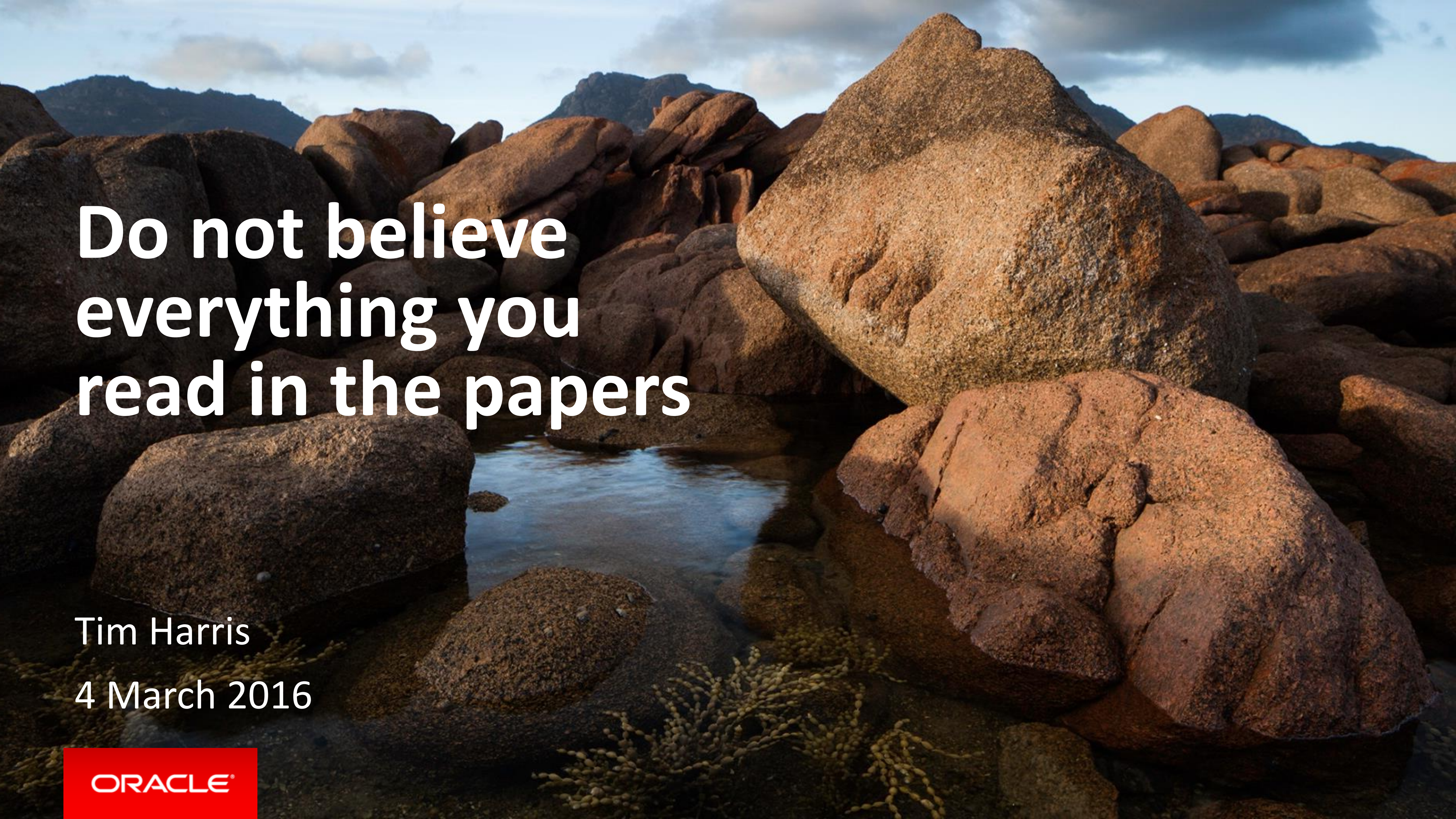
ORACLE®



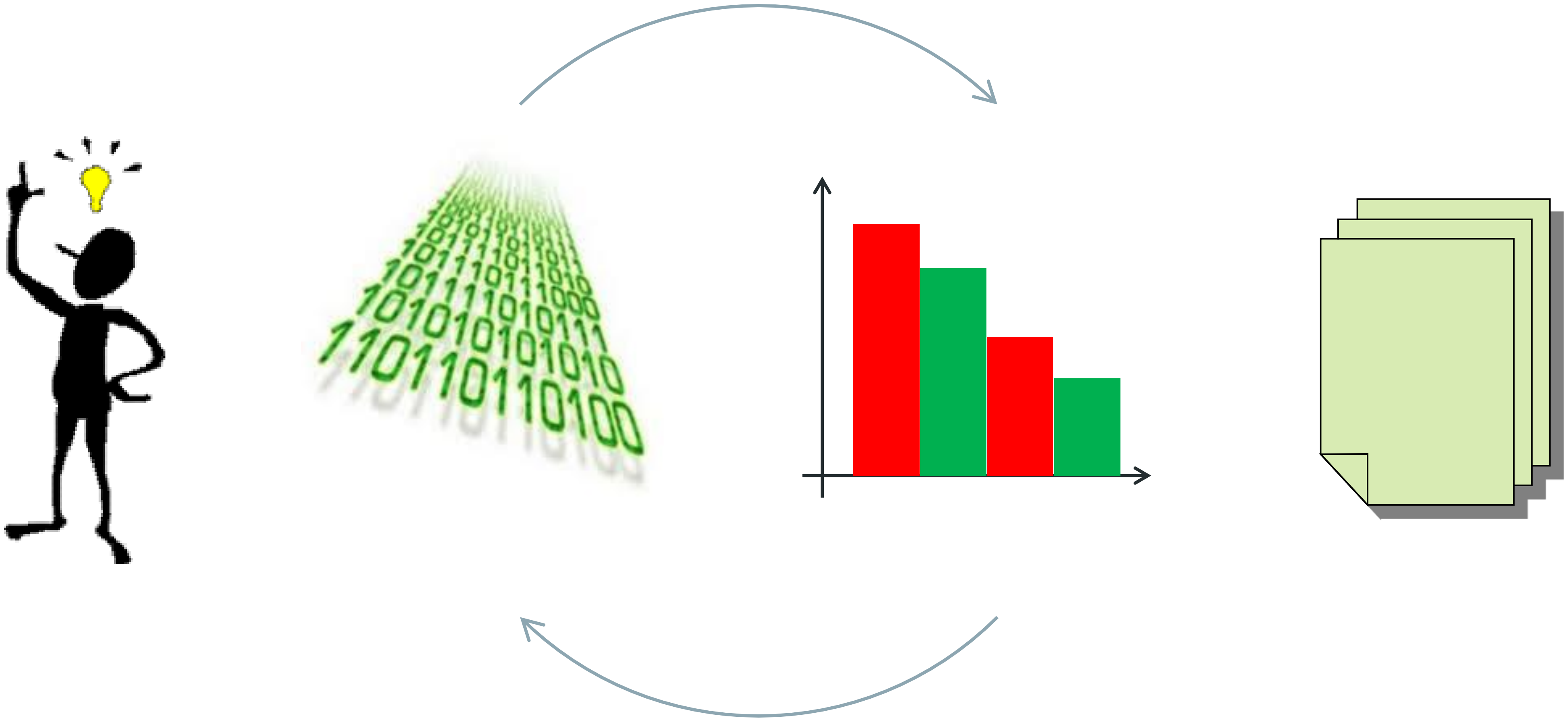
Do not believe everything you read in the papers

Tim Harris

4 March 2016

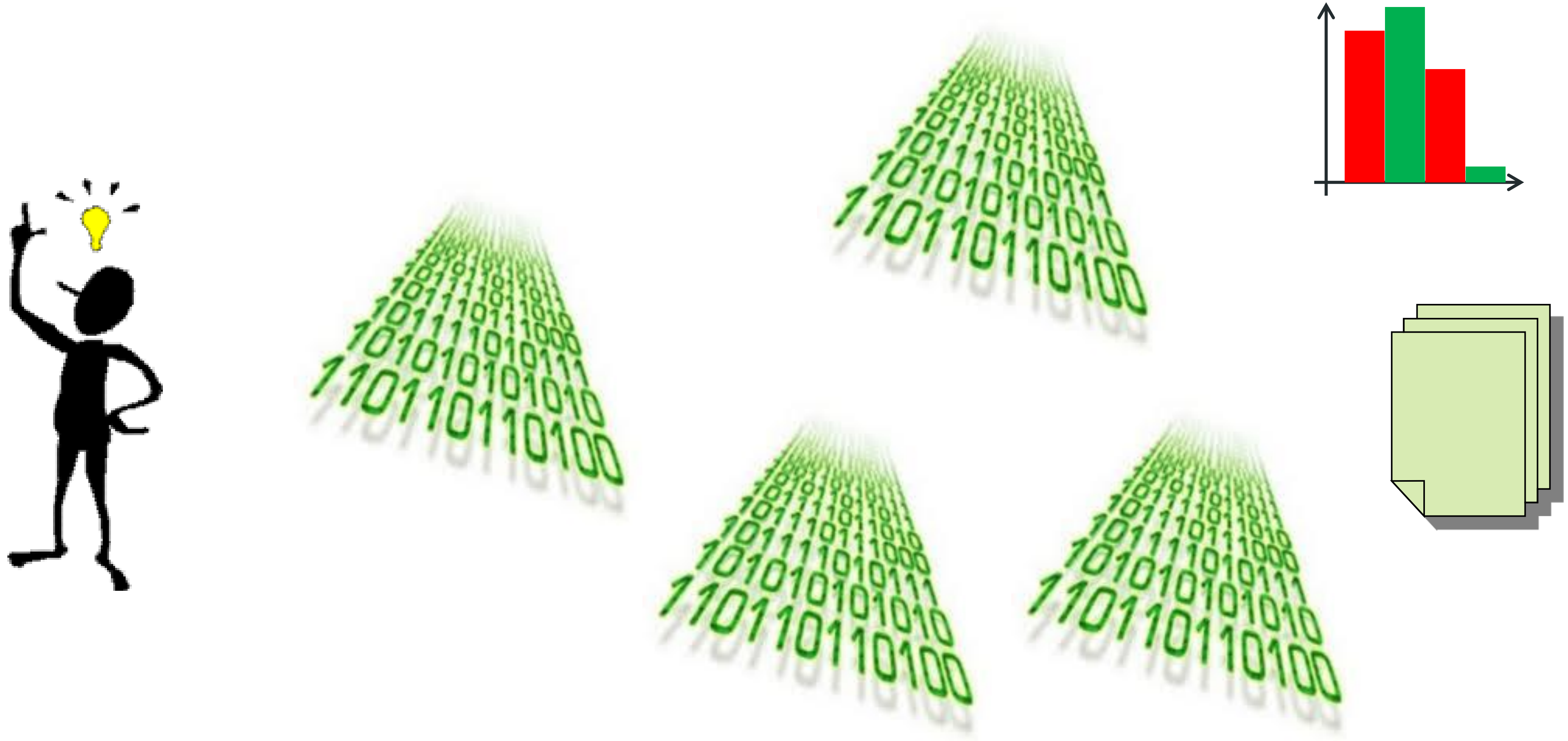
The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Good intentions, bad clip art



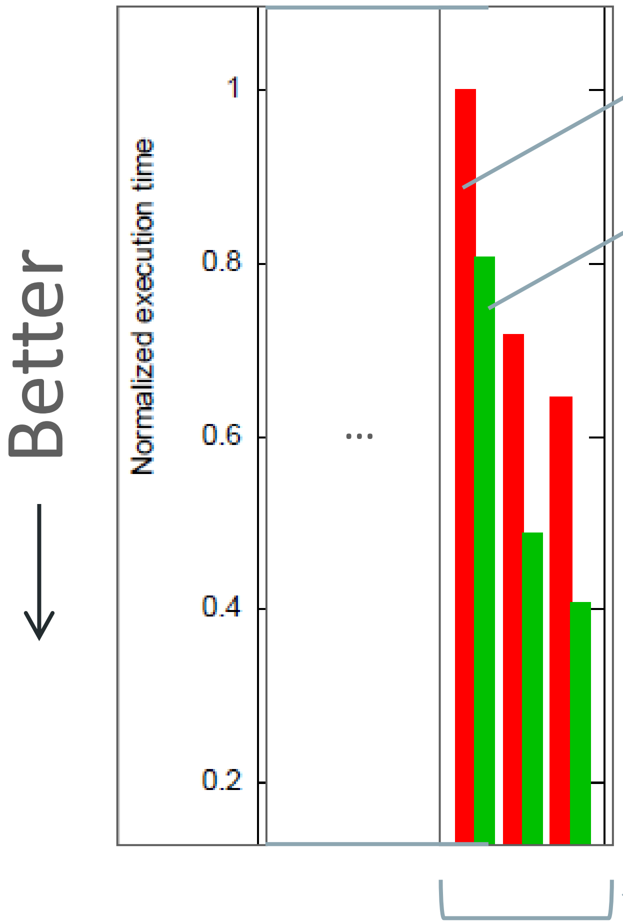
DEADLINE * DEADLINE * DEADLINE * DEA

Good intentions, bad clip art



DEADLINE * DEADLINE * DEADLINE * DEA

An example from my recent work



Previous system

New system

Algorithm running with 18/36/72 threads

What I want to compare

the performance using our C++ runtime system from Java (via an optimizing compiler with a lightweight native function interface)

with

the performance using standard Java fork-join.

What I am actually comparing

Differences in thread placement

Differences in page sizes

Differences in memory placement

Differences in GC activity

Changes in low-level code quality

Changes in work distribution granularity

…

This talk is about

- Making experimental work more methodical
- Some of the “usual suspects” when understanding performance
- Presenting results

This talk is about

- Making experimental work more methodical
- Some of the “usual suspects” when understanding performance
- Presenting results
- Caveats
  - I am mainly talking about work on shared-memory algorithms and data structures
  - Some of these observations may apply elsewhere, but I am sure the war stories differ

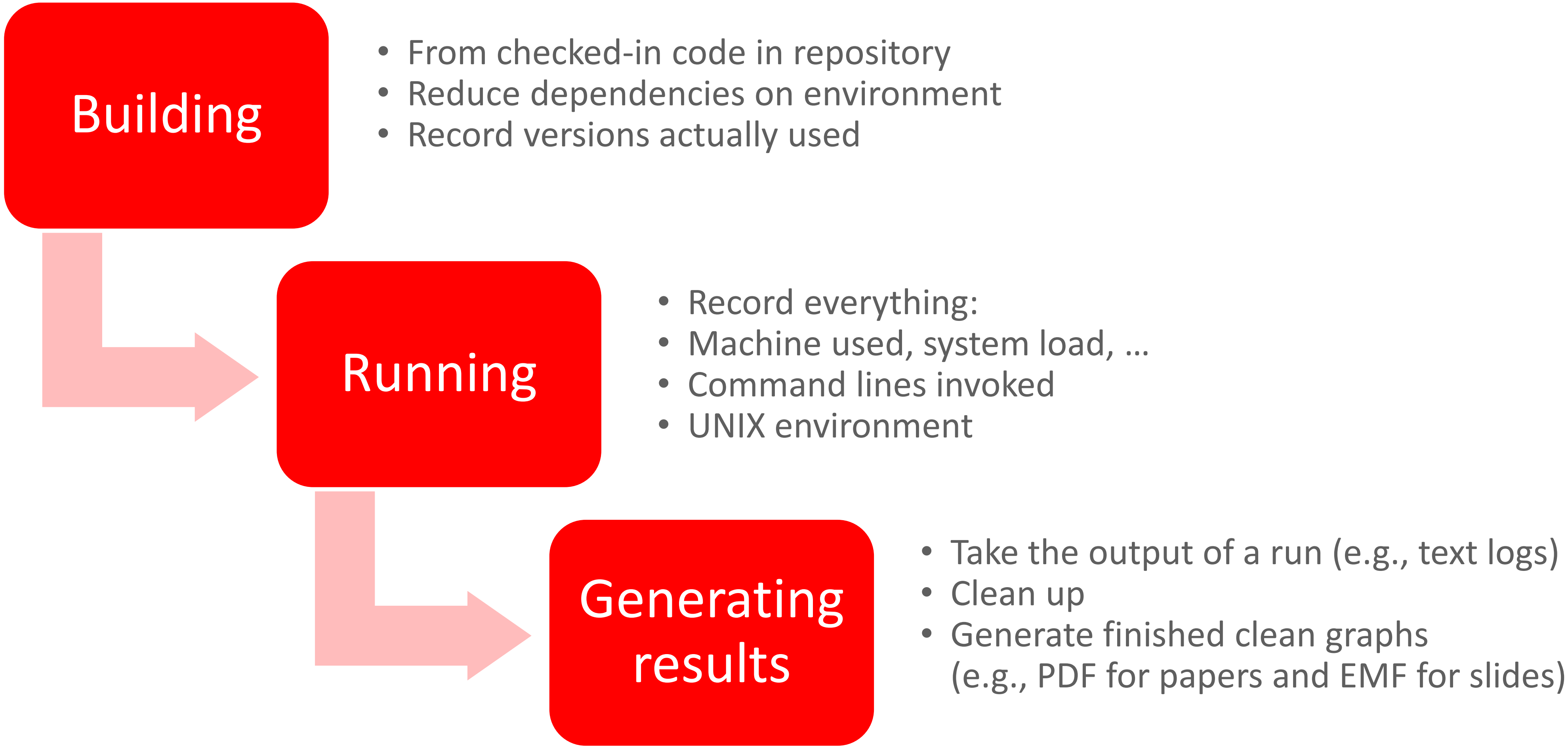
This talk is about

- Making experimental work more methodical
- Some of the “usual suspects” when understanding performance
- Presenting results
- Caveats
  - I am mainly talking about work on shared-memory algorithms and data structures
  - Some of these observations may apply elsewhere, but I am sure the war stories differ
- There are a lot of other elements to consider
  - Experimental design
  - Statistical analysis of results

Overview

1. Script everything, derive results from measurements
2. Plan how to present results before starting work
3. Understand simple cases first

Script everything, record everything



Building

- From checked-in code in repository
- Reduce dependencies on environment
- Record versions actually used

Running

- Record everything:
- Machine used, system load, …
- Command lines invoked
- UNIX environment

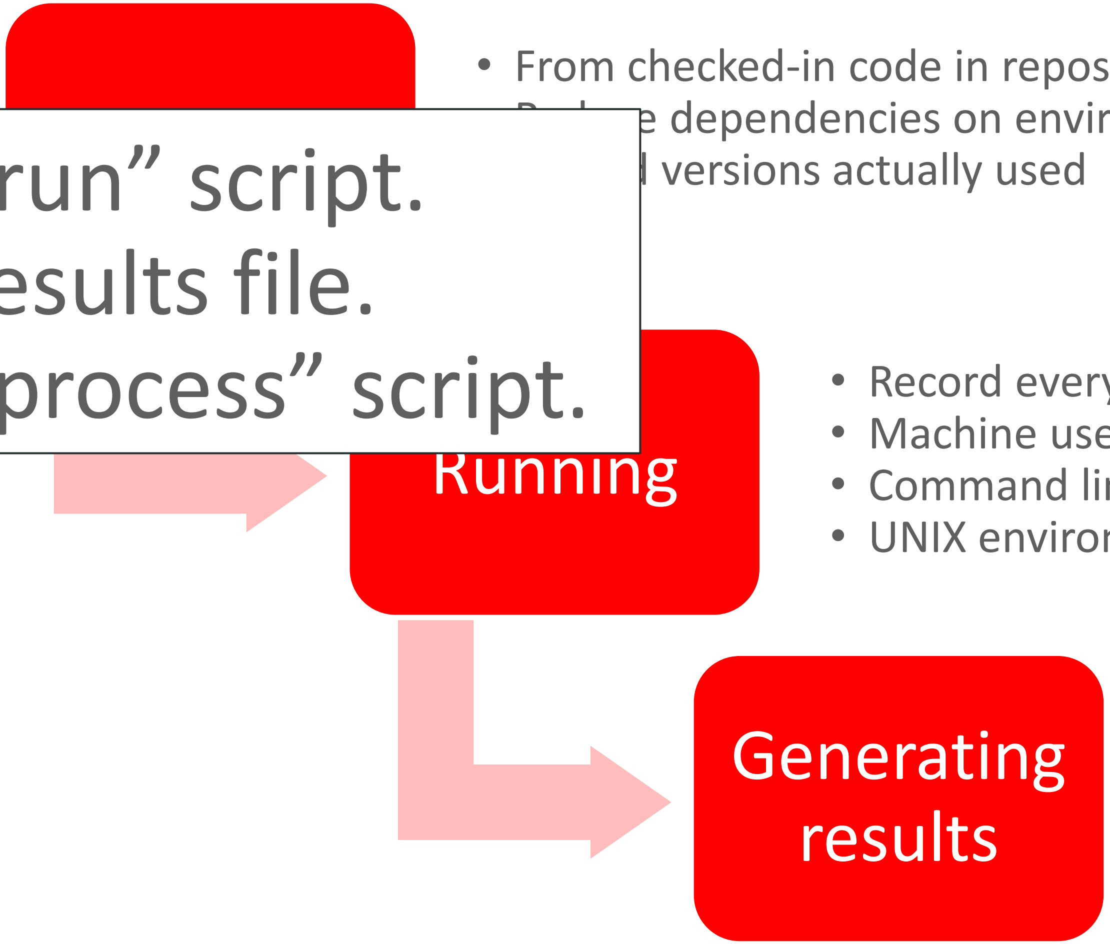
Generating results

- Take the output of a run (e.g., text logs)
- Clean up
- Generate finished clean graphs (e.g., PDF for papers and EMF for slides)

Script everything, record everything

One “run” script.
One results file.
One “process” script.

- From checked-in code in repository
- e dependencies on environment
- versions actually used



- Record everything:
- Machine used, system load, …
- Command lines invoked
- UNIX environment

- Take the output of a run (e.g., text logs)
- Clean up
- Generate finished clean graphs (e.g., PDF for papers and EMF for slides)

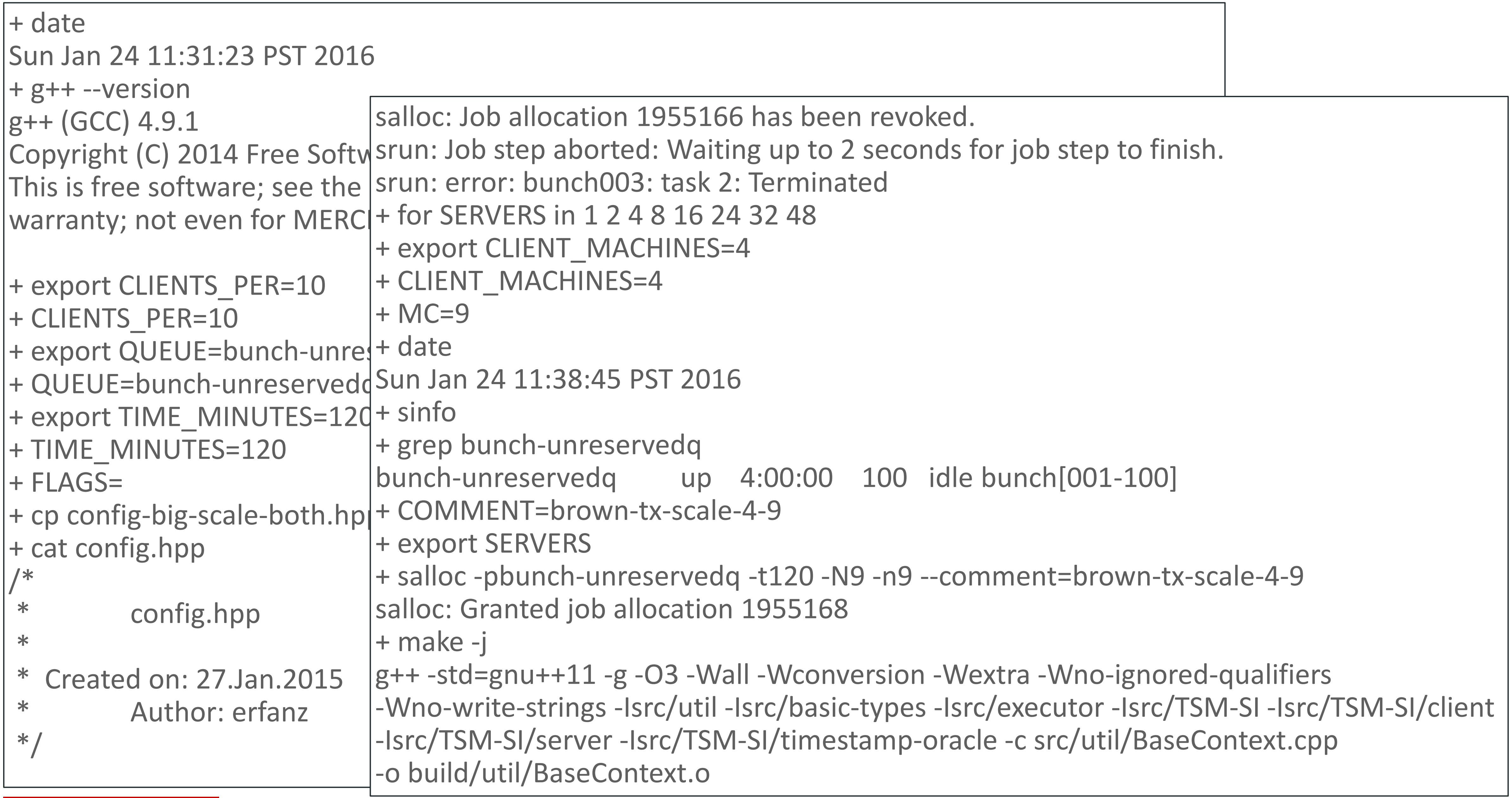
ORACLE

```
+ date
Sun Jan 24 11:31:23 PST 2016
+ g++ --version
g++ (GCC) 4.9.1
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

+ export CLIENTS_PER=10
+ CLIENTS_PER=10
+ export QUEUE=bunch-unreservedq
+ QUEUE=bunch-unreservedq
+ export TIME_MINUTES=120
+ TIME_MINUTES=120
+ FLAGS=
+ cp config-big-scale-both.hpp config.hpp
+ cat config.hpp
/*
 *          config.hpp
 *
 *  Created on: 27.Jan.2015
 *          Author: erfanz
 */
```

run (e.g., text logs)

lean graphs
s and EMF for slides)

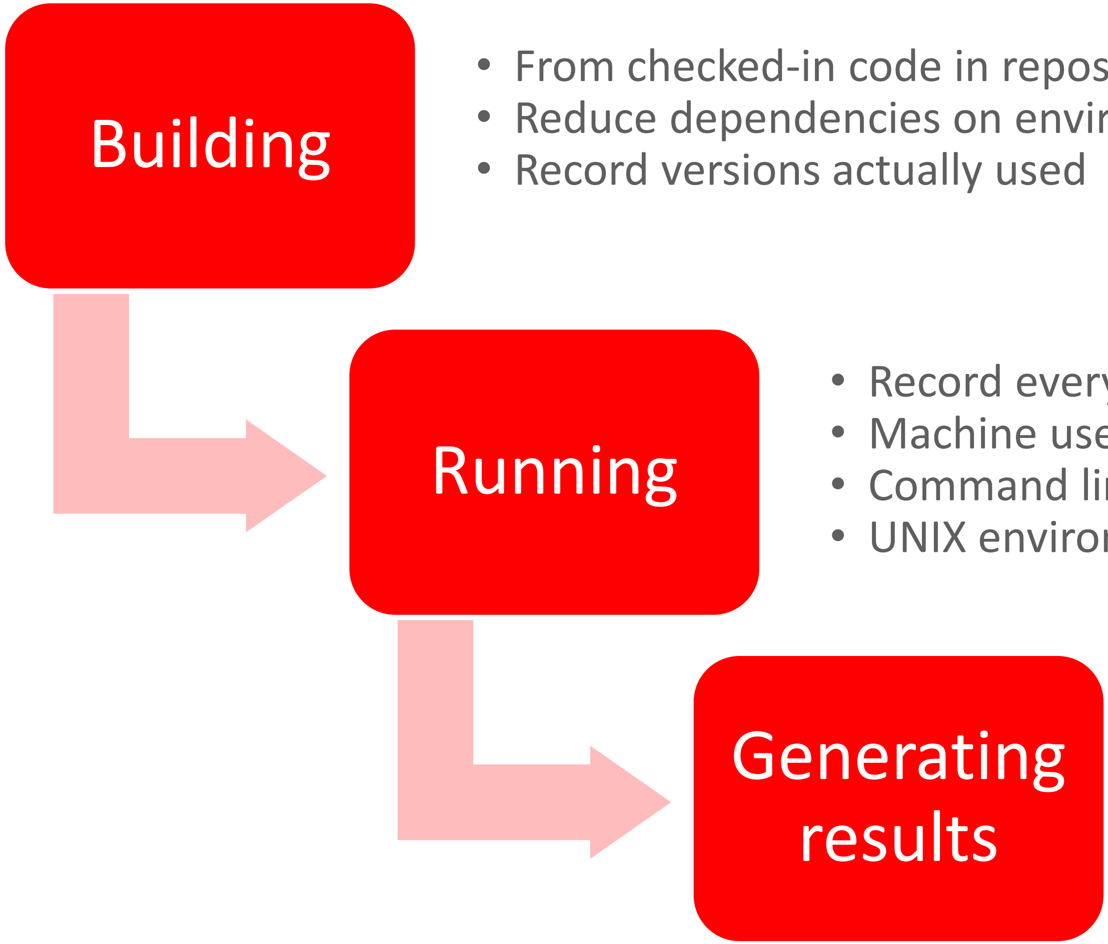


```
+ date
Sun Jan 24 11:31:23 PST 2016
+ g++ --version
g++ (GCC) 4.9.1
Copyright (C) 2014 Free Softw
This is free software; see the
warranty; not even for MERC

+ export CLIENTS_PER=10
+ CLIENTS_PER=10
+ export QUEUE=bunch-unres
+ QUEUE=bunch-unreservedq
+ export TIME_MINUTES=120
+ TIME_MINUTES=120
+ FLAGS=
+ cp config-big-scale-both.hpp
+ cat config.hpp
/*
 *          config.hpp
 *
 *  Created on: 27.Jan.2015
 *          Author: erfanz
 */
```

```
salloc: Job allocation 1955166 has been revoked.
srun: Job step aborted: Waiting up to 2 seconds for job step to finish.
srun: error: bunch003: task 2: Terminated
+ for SERVERS in 1 2 4 8 16 24 32 48
+ export CLIENT_MACHINES=4
+ CLIENT_MACHINES=4
+ MC=9
+ date
Sun Jan 24 11:38:45 PST 2016
+ sinfo
+ grep bunch-unreservedq
bunch-unreservedq      up    4:00:00    100   idle bunch[001-100]
+ COMMENT=brown-tx-scale-4-9
+ export SERVERS
+ salloc -pbunch-unreservedq -t120 -N9 -n9 --comment=brown-tx-scale-4-9
salloc: Granted job allocation 1955168
+ make -j
g++ -std=gnu++11 -g -O3 -Wall -Wconversion -Wextra -Wno-ignored-qualifiers
-Wno-write-strings -Isrc/util -Isrc/basic-types -Isrc/executor -Isrc/TSM-SI -Isrc/TSM-SI/client
-Isrc/TSM-SI/server -Isrc/TSM-SI/timestamp-oracle -c src/util/BaseContext.cpp
-o build/util/BaseContext.o
```

Script everything, record everything



Building

- From checked-in code in repository
- Reduce dependencies on environment
- Record versions actually used

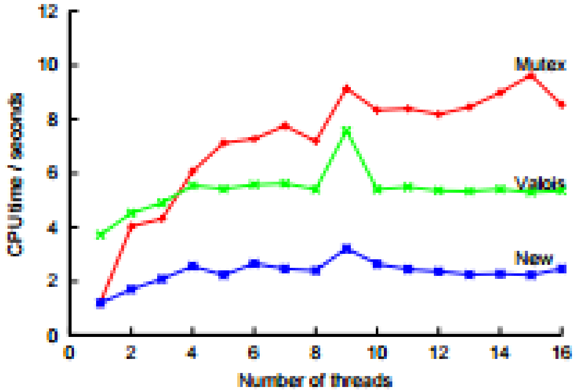
Running

- Record everything:
- Machine used, system load, …
- Command lines invoked
- UNIX environment

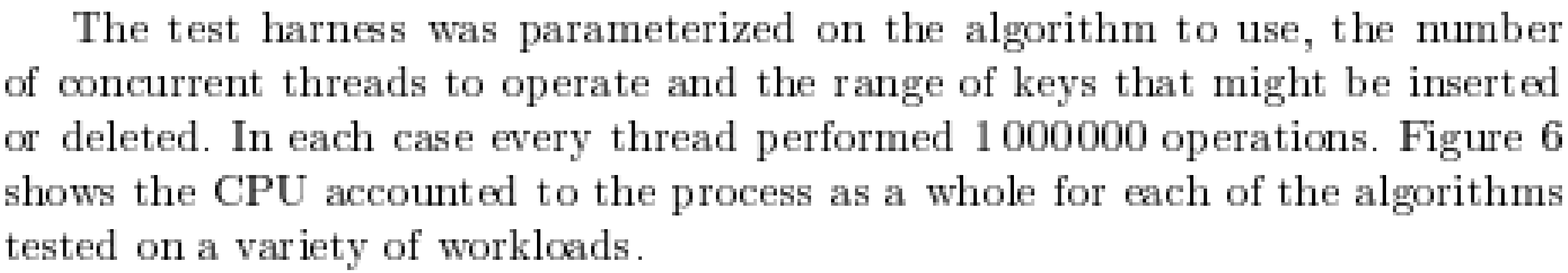
Generating results

- Take the output of a run (e.g., text logs)
- Clean up
- Generate finished clean graphs
  (e.g., PDF for papers and EMF for slides)

Starting and stopping work



(*b*) Non-reference-counted algorithms with keys 0 . . . 255.

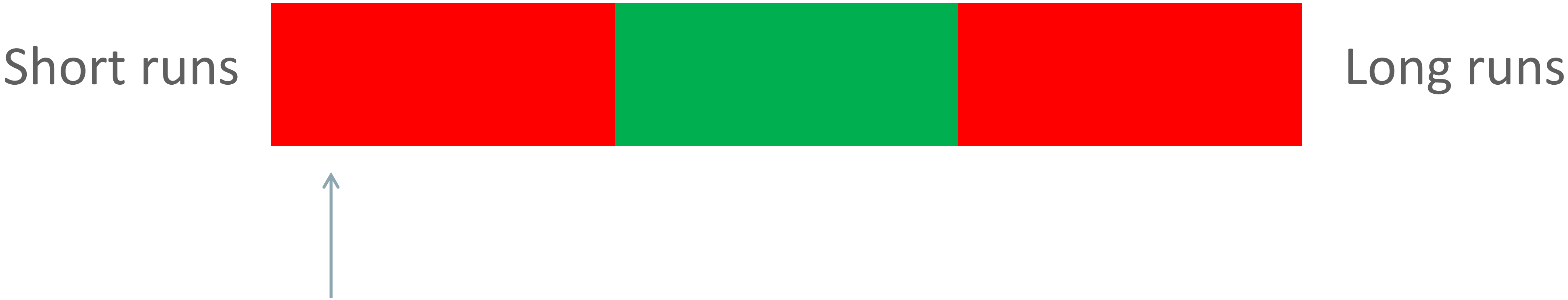


The test harness was parameterized on the algorithm to use, the number of concurrent threads to operate and the range of keys that might be inserted or deleted. In each case every thread performed 1 000 000 operations. Figure 6 shows the CPU accounted to the process as a whole for each of the algorithms tested on a variety of workloads.

“A pragmatic implementation of non-blocking linked lists”, Tim Harris, DISC 2001

Starting and stopping work

- How much work to do?



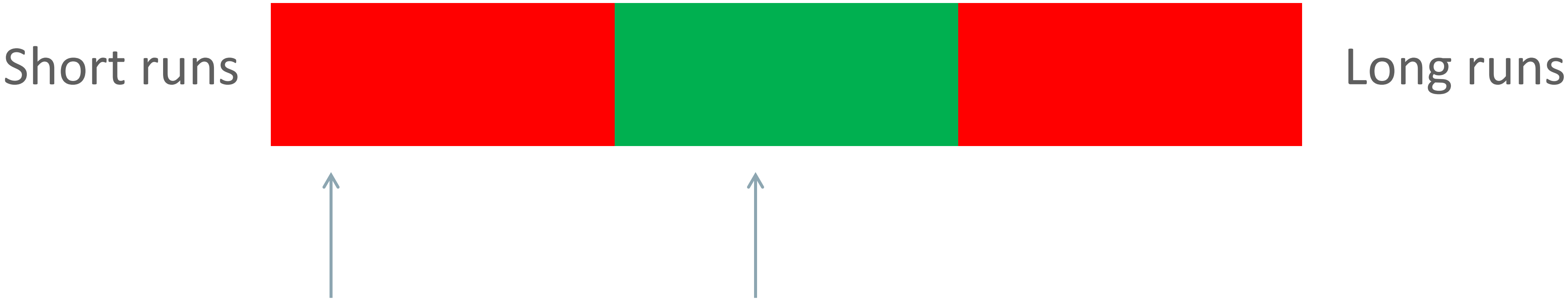
Short runs

Long runs

Too little: results dominated by start-up effects. Normalized metrics vary as you vary the duration.

Starting and stopping work

- How much work to do?



Short runs

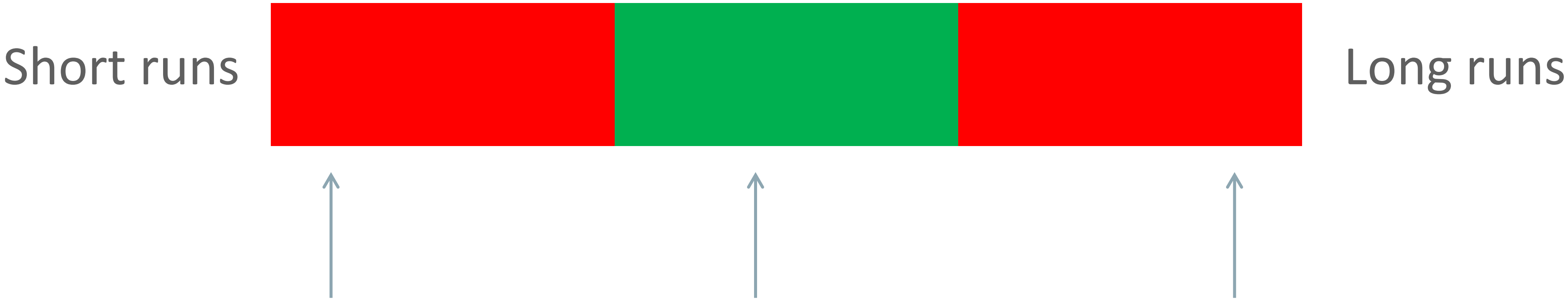
Long runs

Too little: results dominated by start-up effects. Normalized metrics vary as you vary the duration.

OK: results not sensitive to the exact choice of settings. Confirm this: double / halve duration with no change.

Starting and stopping work

- How much work to do?



Short runs

Long runs

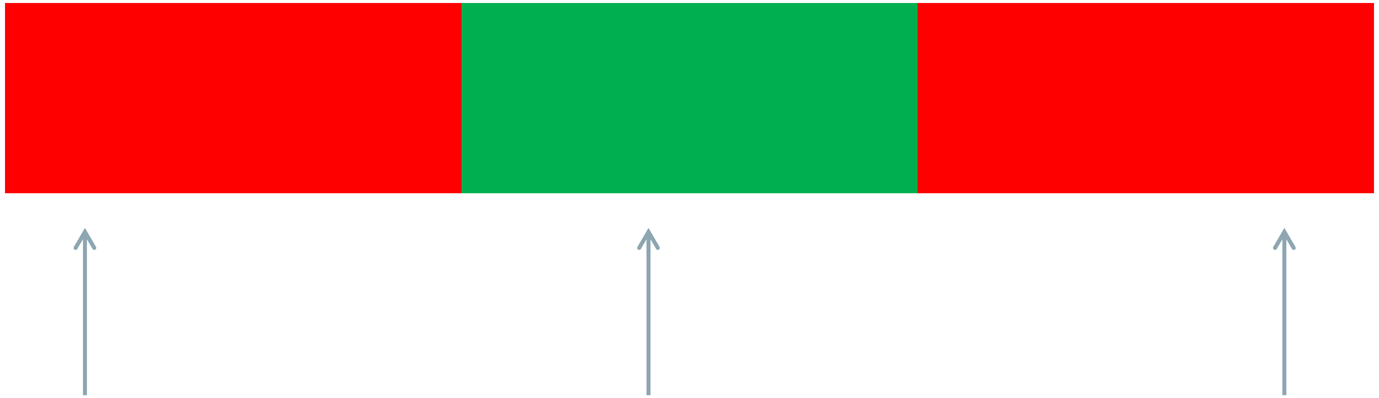
Too little: results dominated by start-up effects. Normalized metrics vary as you vary the duration.

OK: results not sensitive to the exact choice of settings. Confirm this: double / halve duration with no change.

Too much??

Starting and stopping work

- How much work to do?



Short runs

Long runs

Too little: results dominated by start-up effects. Normalized metrics vary as you vary the duration.

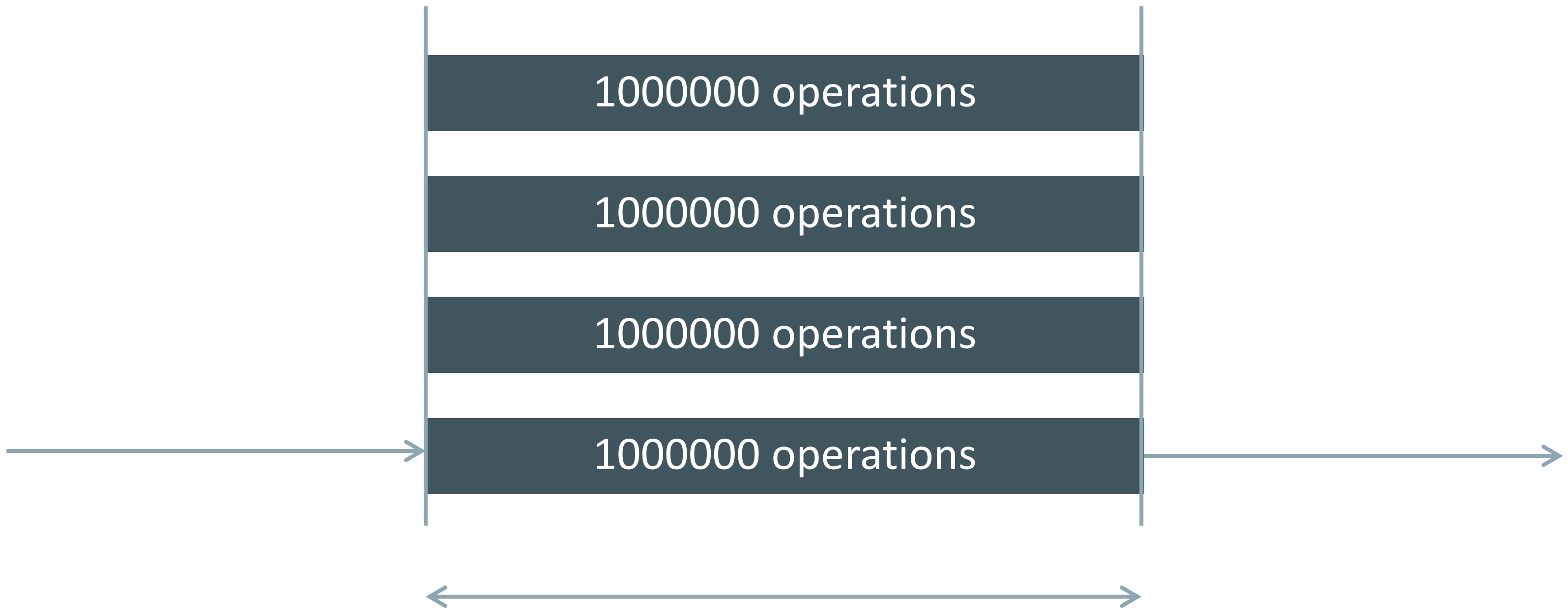
OK: results not sensitive to the exact choice of settings. Confirm this: double / halve duration with no change.

Too much??

Deters experimentation if turnaround time is long (e.g. >> overnight)

Harder to separate resource re-use policy from the rest of the expt.

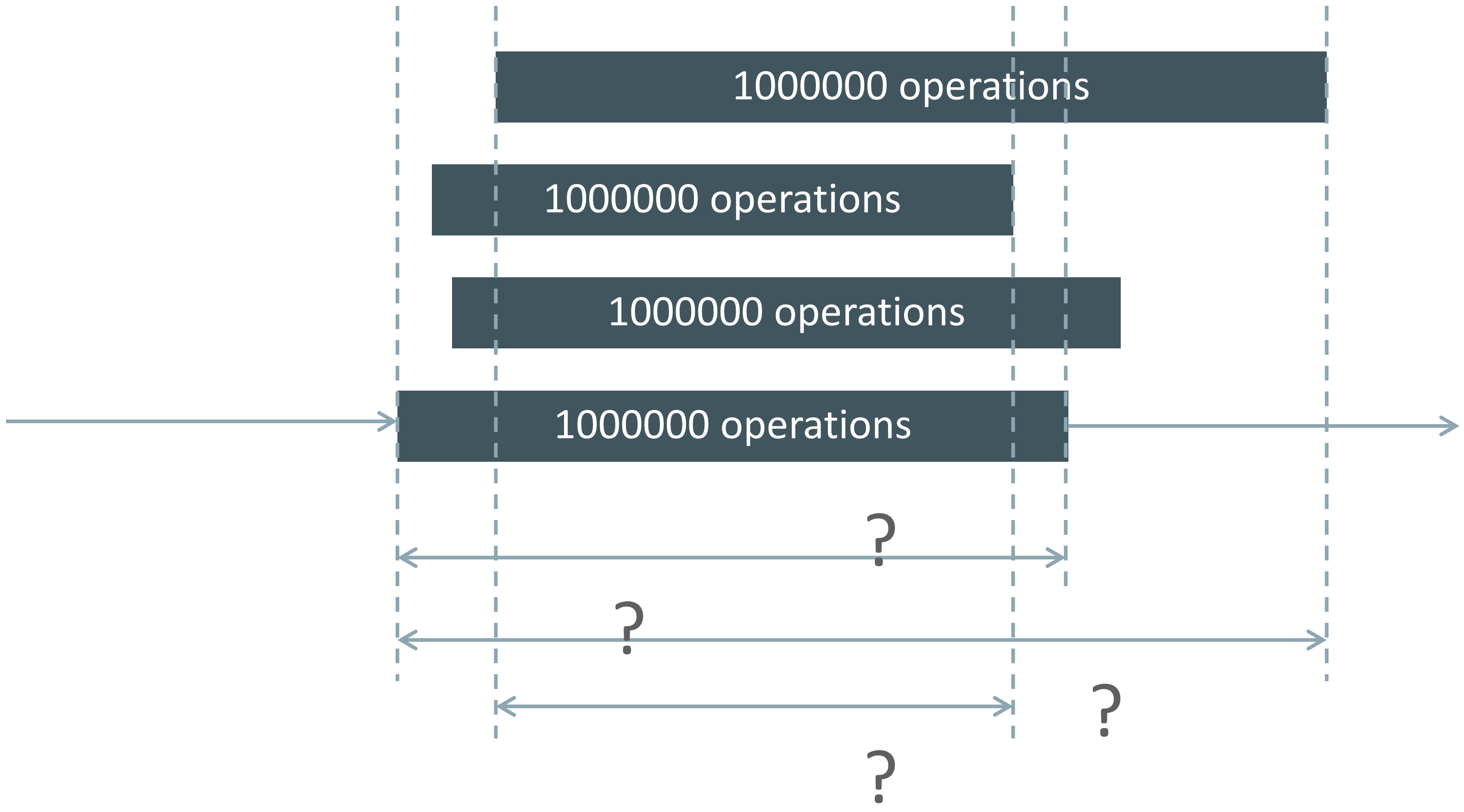
Starting and stopping work… what we imagine:



Measure duration = 2s

Throughput = 4M / 2s = 2M ops / s

Starting and stopping work… what we get:

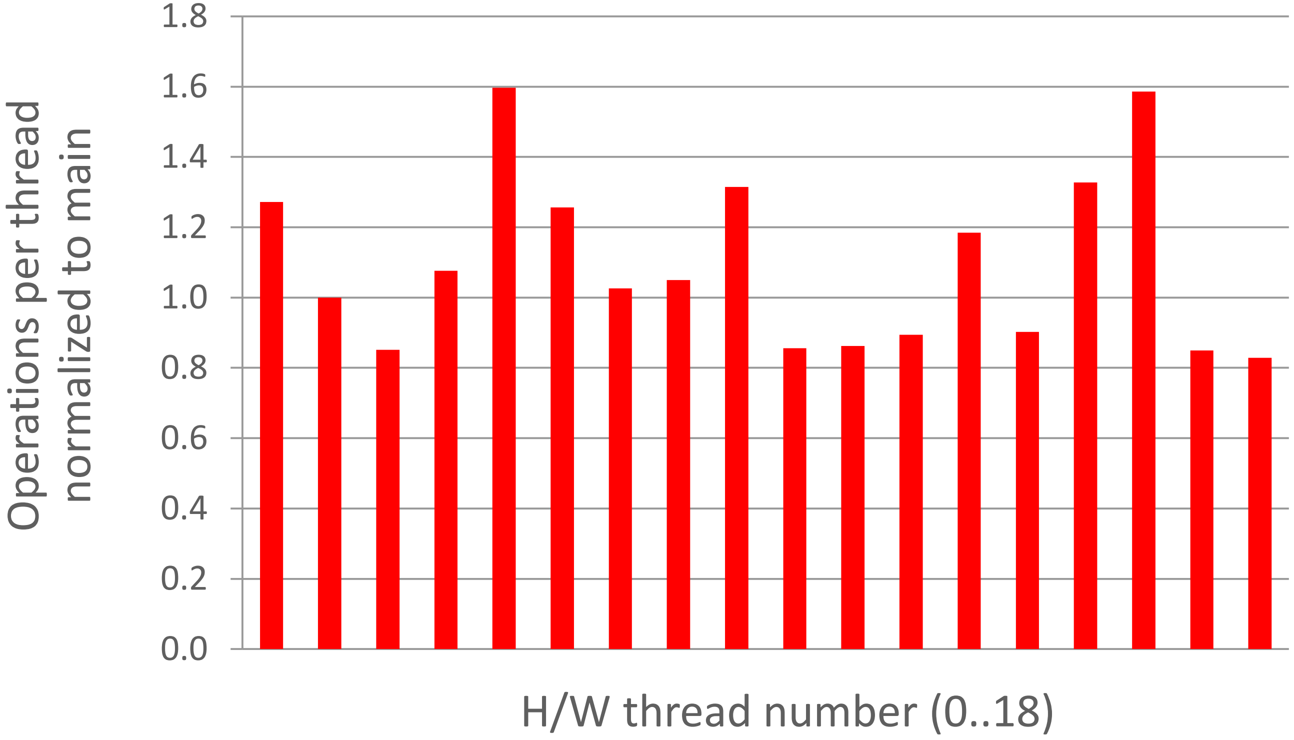


Constant load

- Fixed number of threads active
  - E.g., data structure micro-benchmarks
  - Look at how the structure under test behaves under varying loads
- Keep all threads active throughout experiment. Typically:
  - Create threads
  - Perform warm-up work in each thread
  - Barrier
  - Actual measurement interval
  - Main thread signals request to exit to others
- Investigate and report differences in actual work completed by threads

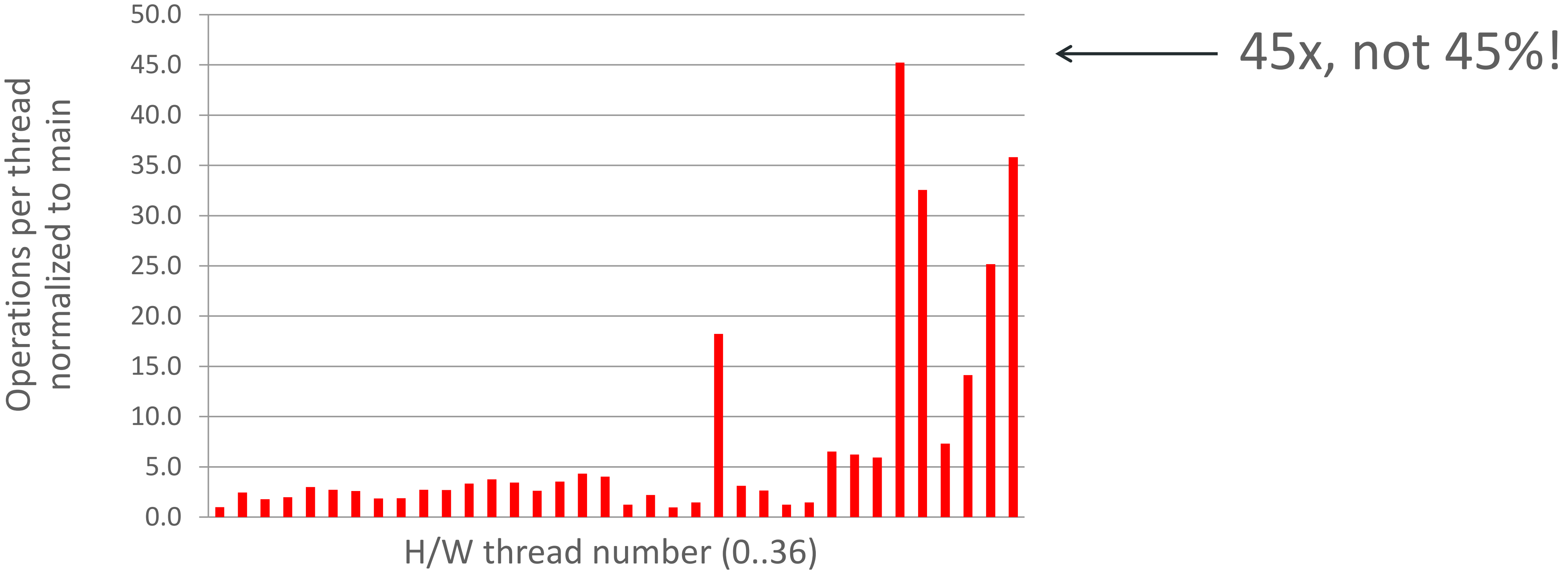
Constant load unfairness: simple test-and-test-and-set lock

- Main thread runs a constant number of iterations, signals others to stop
- 2-socket Haswell, threads pinned sequentially to cores in 1 socket



Constant load unfairness: simple test-and-test-and-set lock

- Main thread runs a constant number of iterations, signals others to stop
- 2-socket Haswell, threads pinned sequentially to cores in both sockets

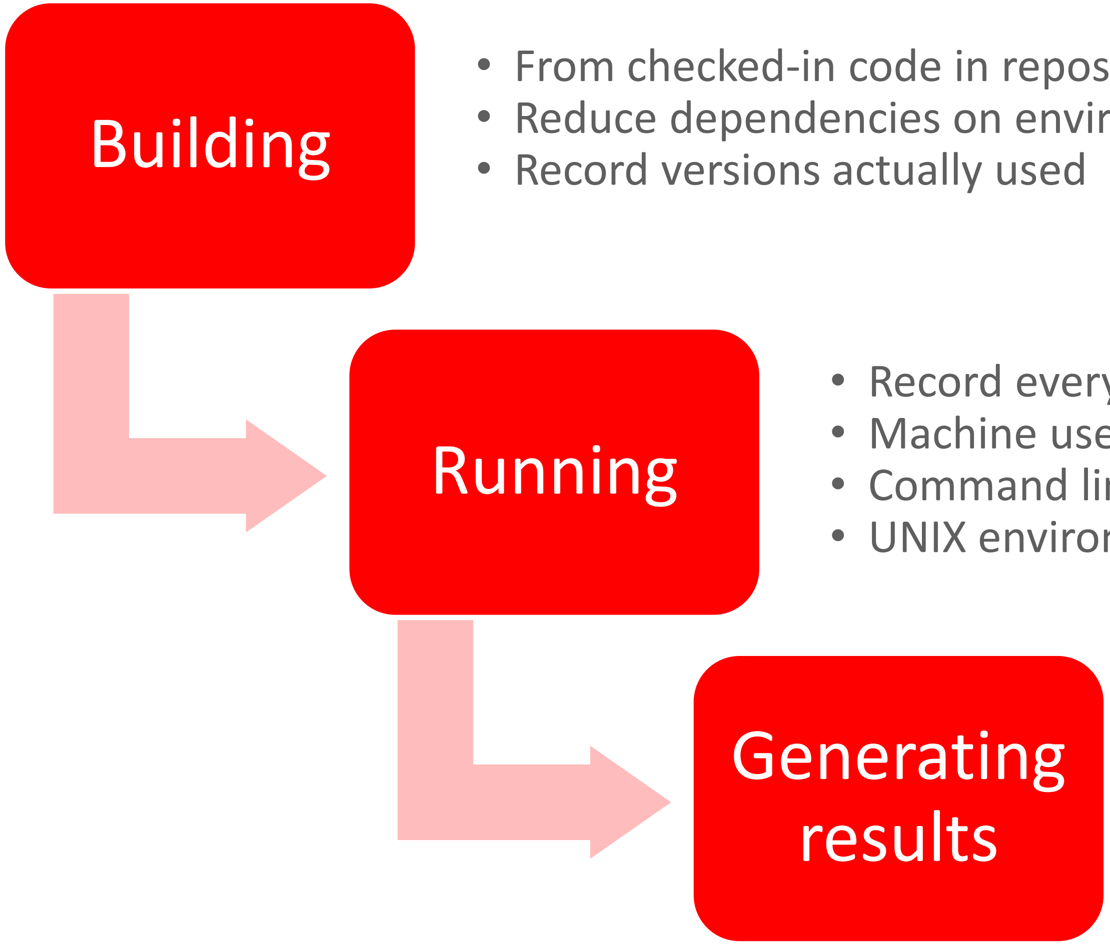


45x, not 45%!

Constant work

- Fixed amount of work to perform
  - Share it among a set of threads – e.g., OpenMP parallel loop
  - Aim to use threads to complete the work more quickly
  - Measure from when the work is started until when it is all complete
- Show results for
  - Strong scaling: same amount of work as you vary the number of threads
  - Weak scaling: increase the work proportional to the threads
- Investigate and report differences in
  - Load imbalance (do threads finish early?)
  - Actual amount of work completed by threads (do some threads work faster?)

Script everything, record everything



Building

- From checked-in code in repository
- Reduce dependencies on environment
- Record versions actually used

Running

- Record everything:
- Machine used, system load, …
- Command lines invoked
- UNIX environment

Generating results

- Take the output of a run (e.g., text logs)
- Clean up
- Generate finished clean graphs (e.g., PDF for papers and EMF for slides)

Generating results

General principle: derive results from numbers you measure, not from numbers you configure

Generating results

General principle: derive results from numbers you measure, not from numbers you configure

Configuration setting written in incorrect file

Code that reads the setting is buggy

System overrides the settings (e.g., thread pinning)

Environment variable set incorrectly (“GOMP_PROC_BIND”)

Setting is invalid and ignored at runtime

Generating results

“Bind threads 1 per socket”	Have each thread report where it is running
“Run for 10s”	Record time at start & end
“Use 50% reads”	Measured #reads/#ops
“Distribute memory across the machine”	Actual locations and page sizes used

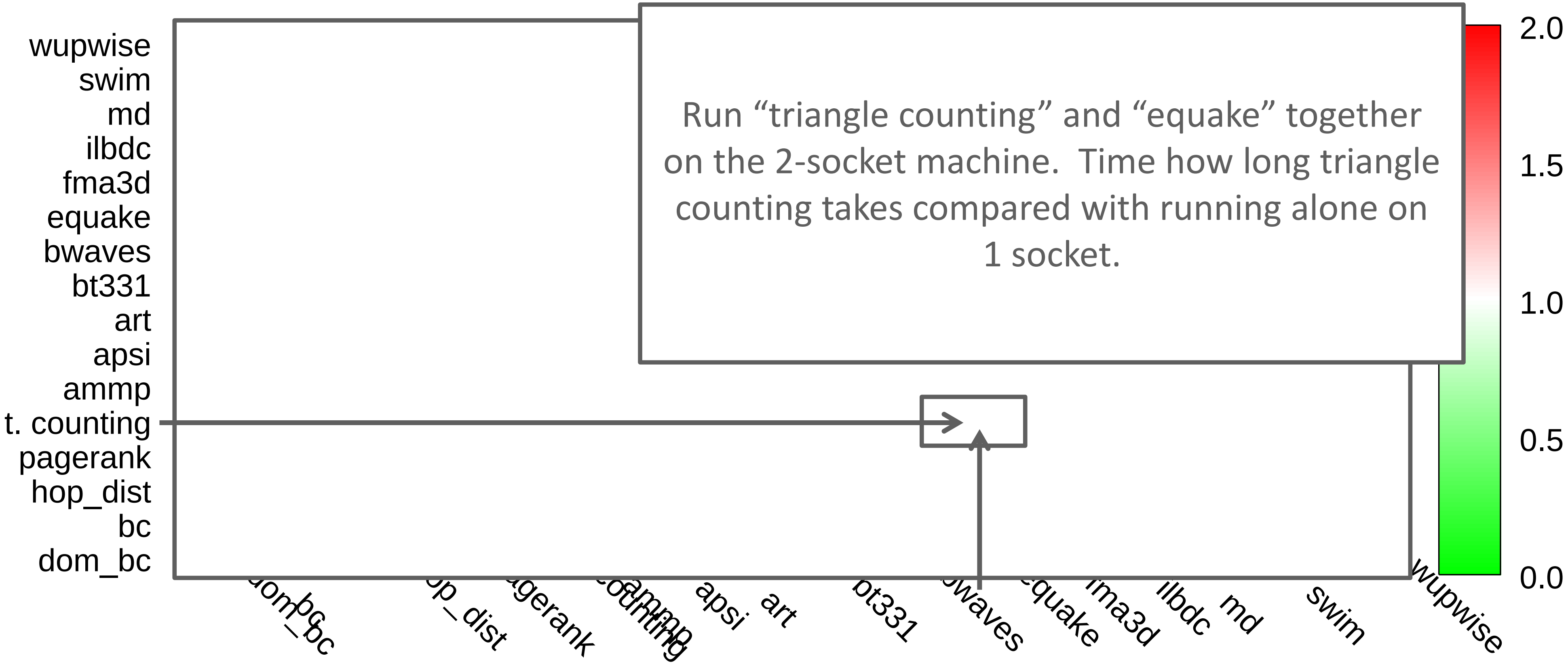
Overview

1. Script everything, derive results from measurements
2. Plan how to present results before starting work
3. Understand simple cases first

Plan how to present results before starting work

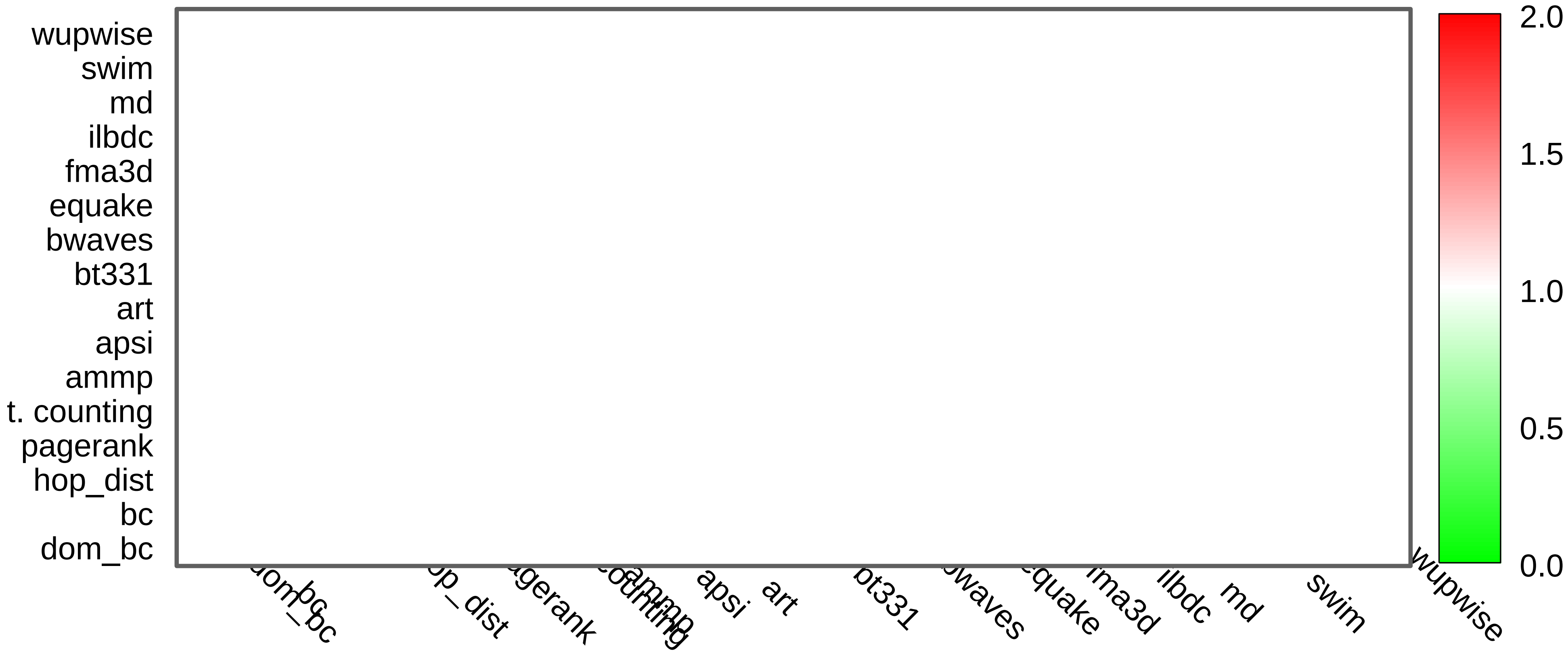
- Why?
  - Make sure you can illustrate the problem you are solving and you know the questions you want to see answered
    - How bad are things now?
    - How much scope exists for improvement?
  - Time to practice explaining the format of the results to other people
  - Time to notice and resolve difficulties running experiments
  - Coding/tweaking/experimenting will expand to fill the time available
    - Let them!

Running pairs of workloads together on a 2-socket machine

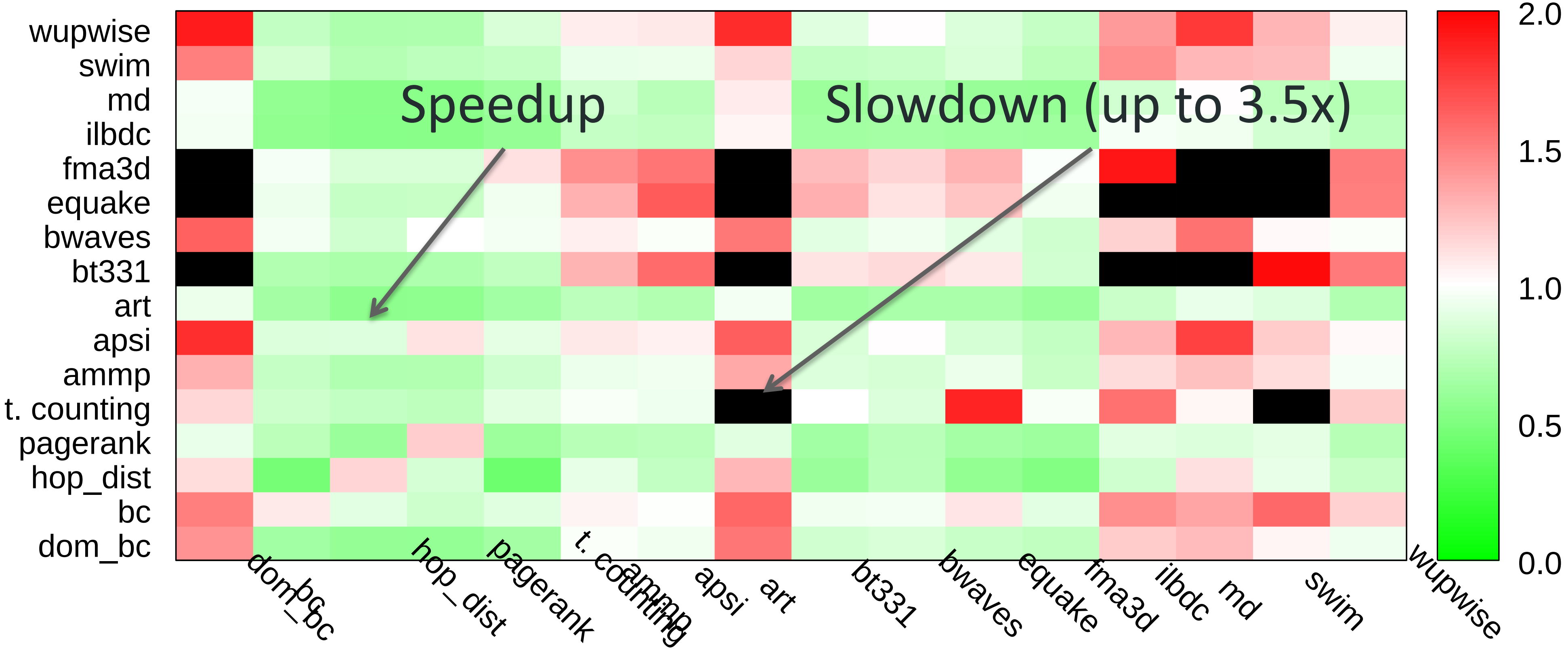


Run “triangle counting” and “equake” together on the 2-socket machine. Time how long triangle counting takes compared with running alone on 1 socket.

Running pairs of workloads together on a 2-socket machine



Running pairs of workloads together on a 2-socket machine



Speedup

Slowdown (up to 3.5x)

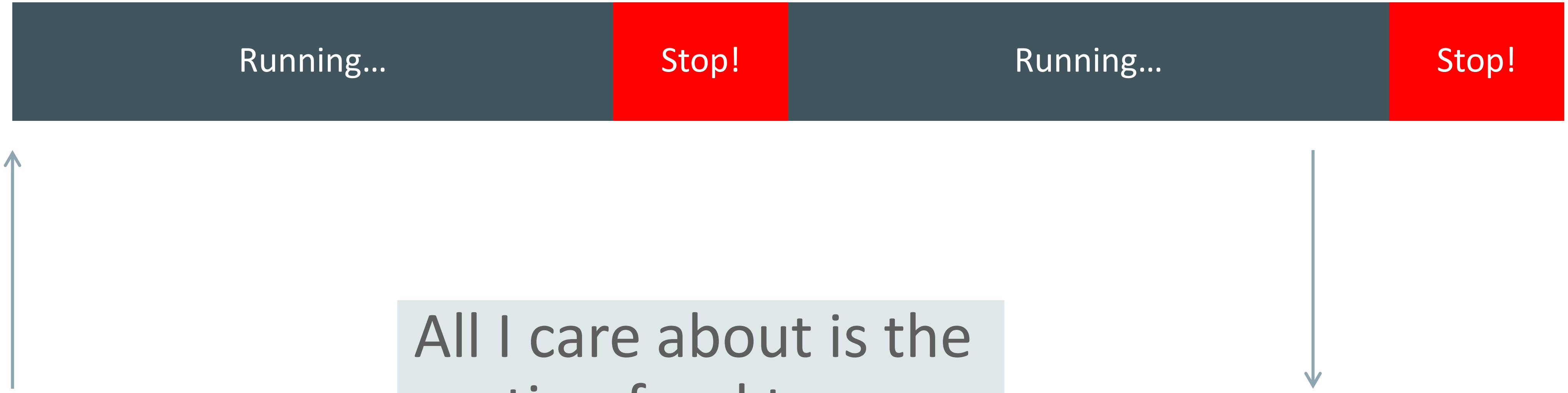
Why does this format work?

- Easy to explain what a good result is like and what a bad result is like
- A neutral result is “quiet”
  - All the squares are white
  - No need to understand what the workloads actually do
- Captures trade-offs
  - Results here often come in pairs
  - Green with red
  - We will see both of them together
- “Dashboard” while doing the work

Trade-offs

- Parallel stop-the-world garbage collector
- Suppose it takes 5% of execution time on average
- Do you care?

Trade-offs



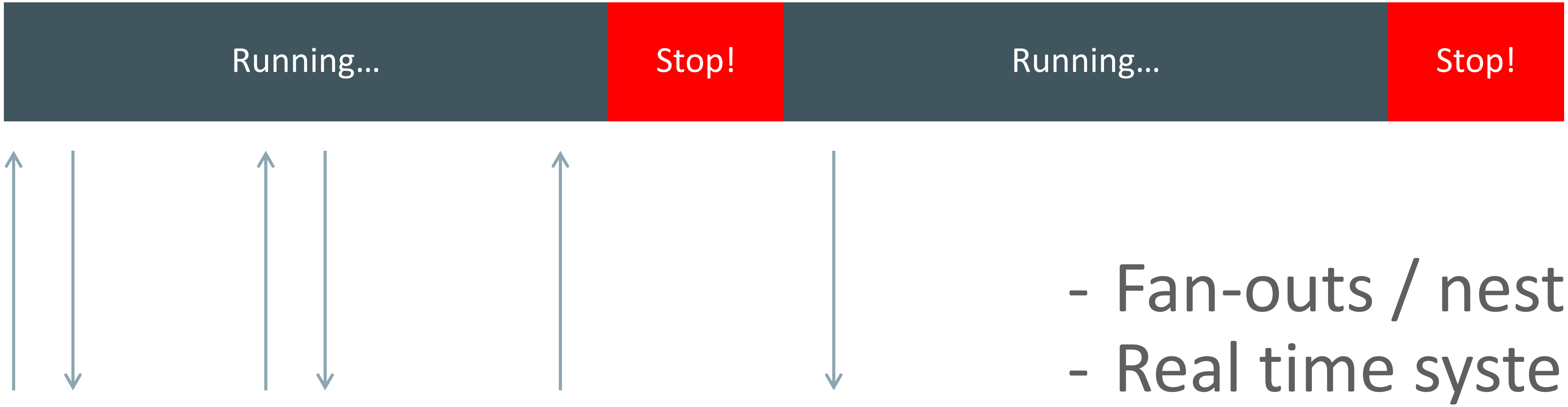
All I care about is the ratio of red to grey

Submit request

Get response

Trade-offs

Now I do care that unlucky requests are delayed



Running… Stop! Running… Stop!

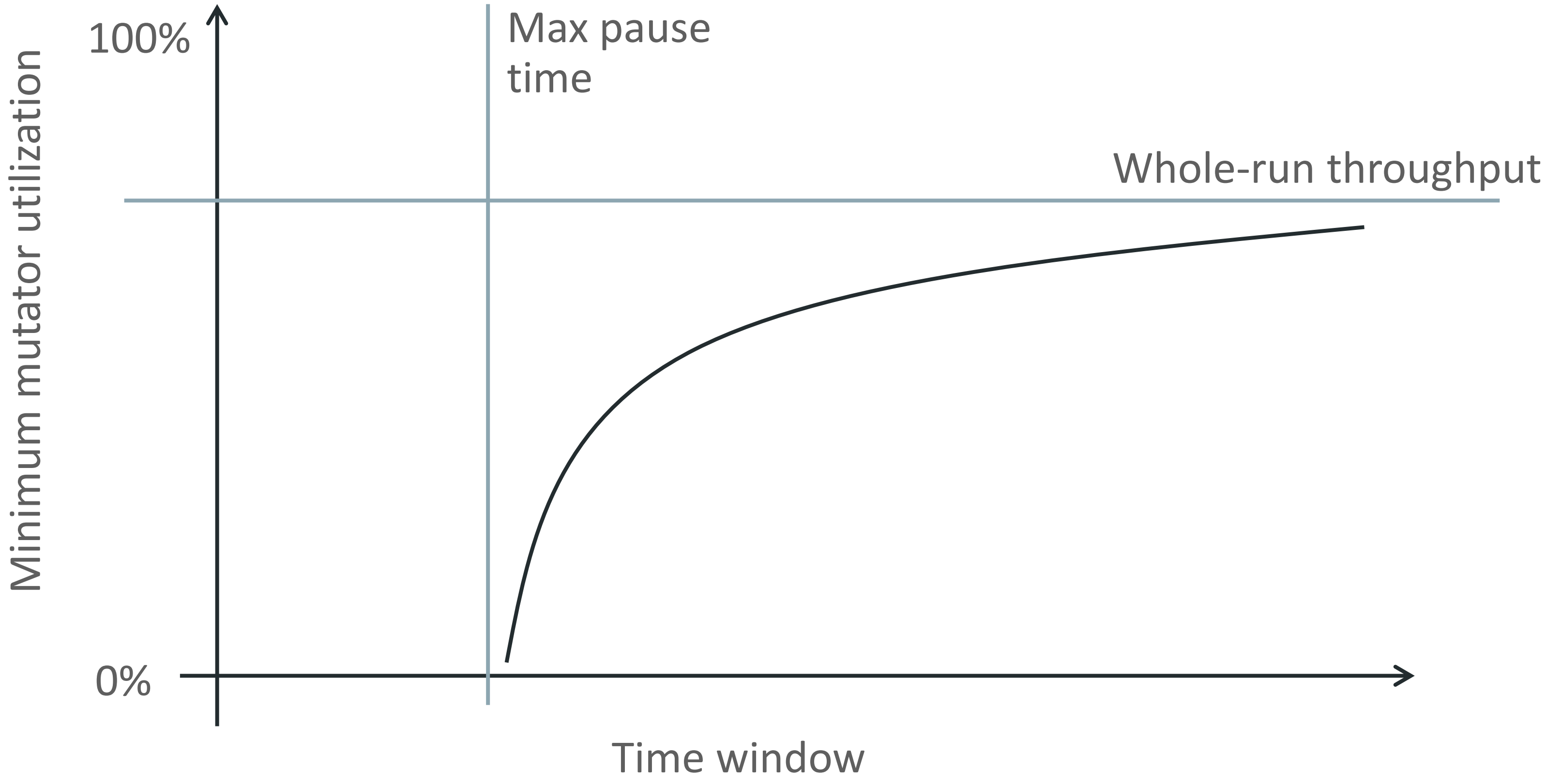
Req & response

Req & response

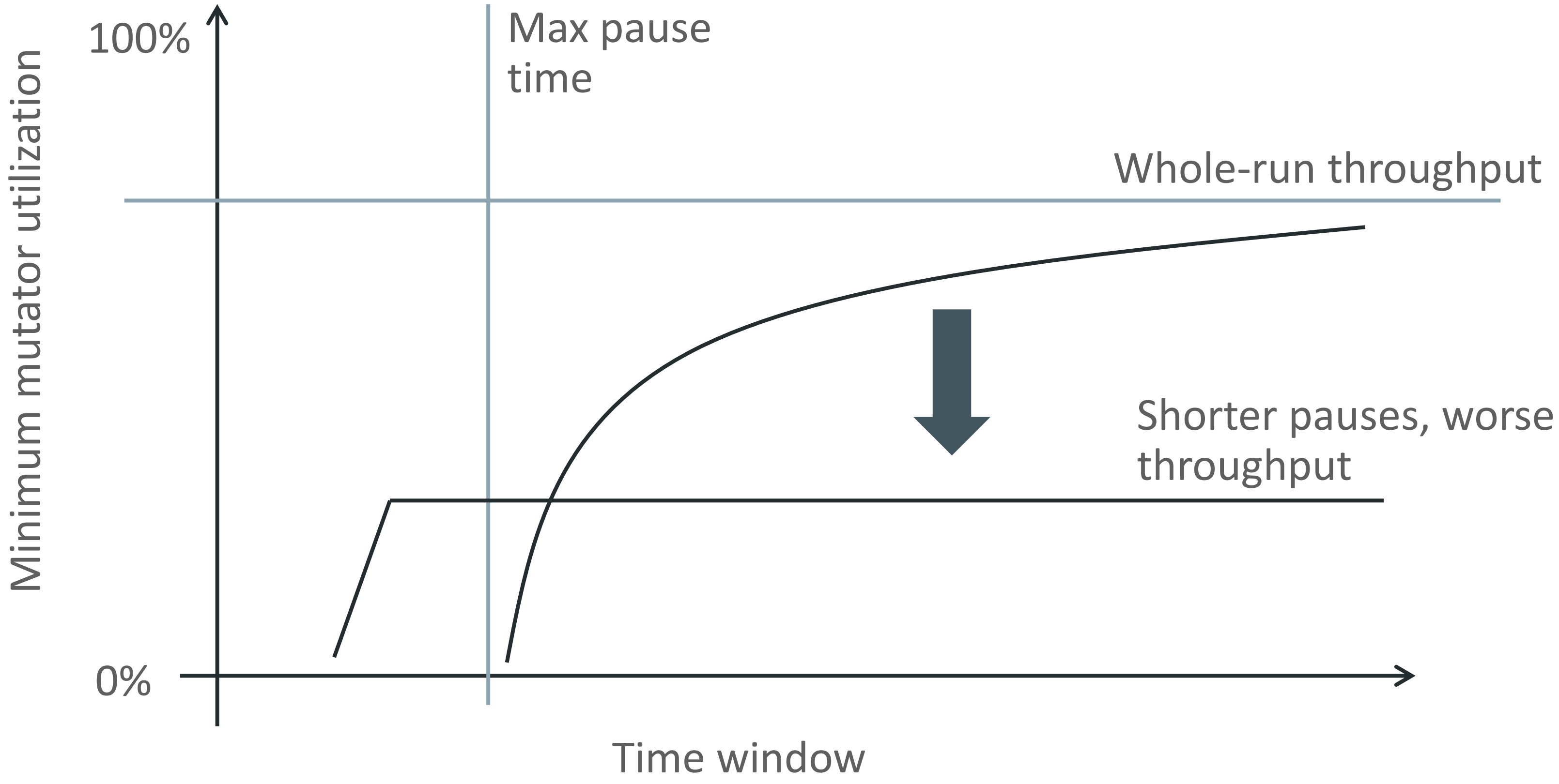
Req & . . . . . response

- Fan-outs / nesting
- Real time systems
- Low-latency trading

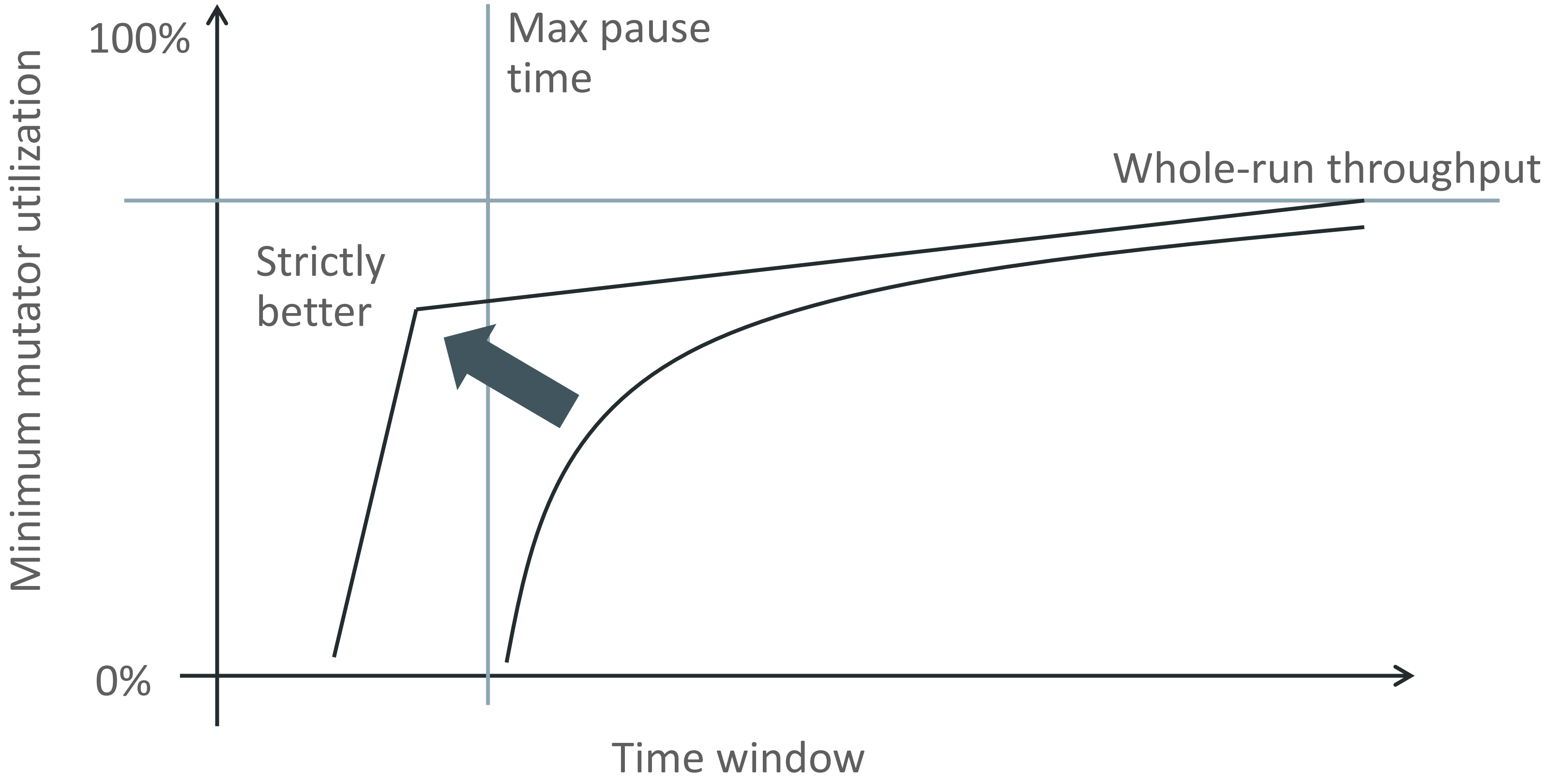
Minimum mutator utilization



Minimum mutator utilization



Minimum mutator utilization



Summary

- Make formats easy to explain, e.g.:
  - Ideal behaviour is a horizontal line
  - Ideal behaviour is a blank heat map
- Make numbers easy to read off
  - What does a y-intercept mean?
  - What does a x-intercept mean?
  - Is anything hidden where lines are clumped together?
- Show and expect to see trade-offs

Overview

1. Script everything, derive results from measurements
2. Plan how to present results before starting work
3. Understand simple cases first

Understand simple cases first

- Why? Almost without exception:
  - There are bugs in the test harness
  - There are bugs in the data processing scripts (grep, cut-n-paste, …)
  - There are unexpected factors influencing the results

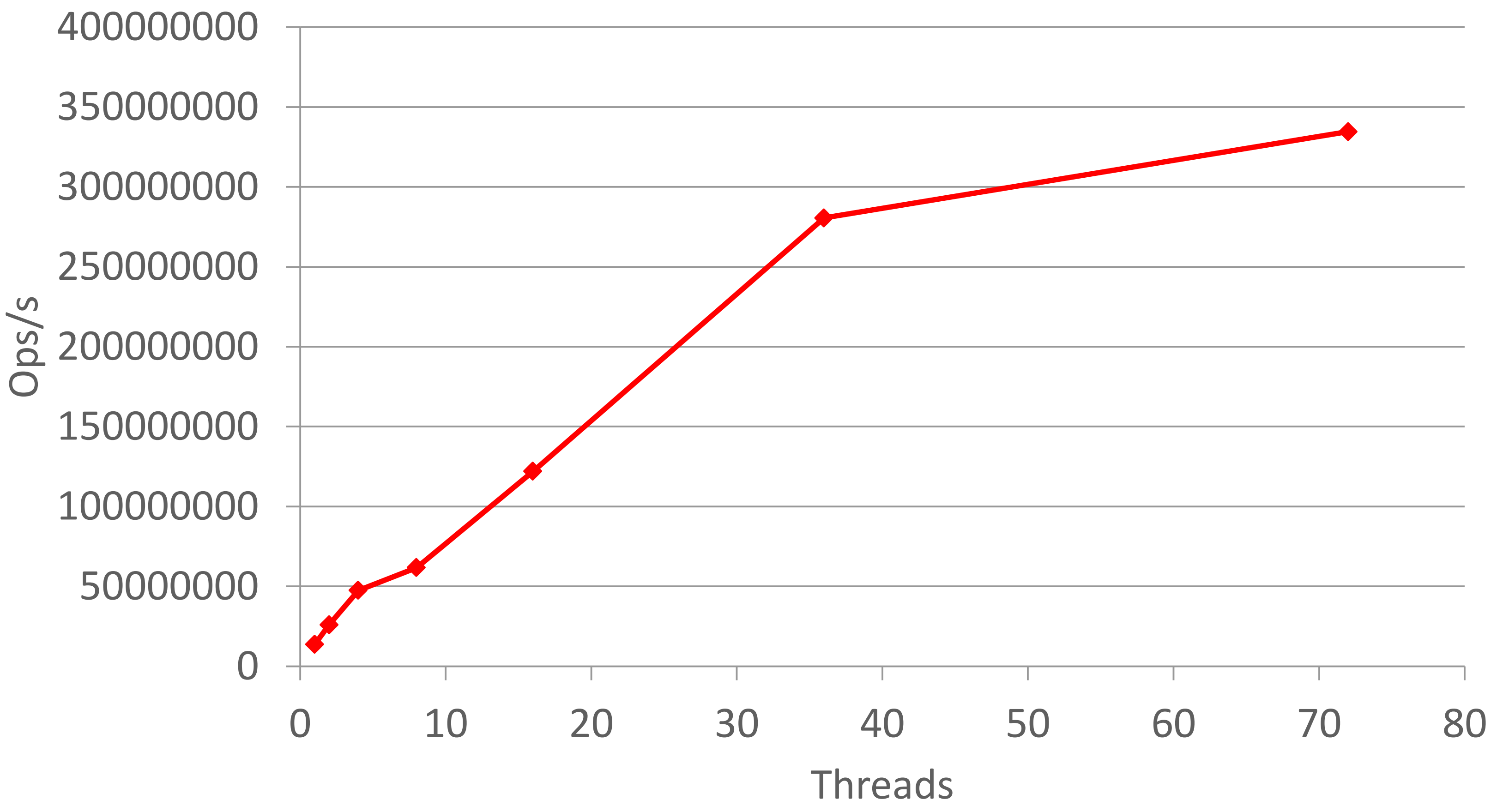
Understand simple cases first

- Why? Almost without exception:
  - There are bugs in the test harness
  - There are bugs in the data processing scripts (grep, cut-n-paste, …)
  - There are unexpected factors influencing the results
- Before paying any attention to actual results, try to identify simple test cases that should have known behavior
  - (Even if you do not care about them, or they are contrived)
  - Do they behave as expected?
  - Can you completely explain them? (“Memory system effects” is not an answer)
  - Add them to regression tests, and watch for them breaking

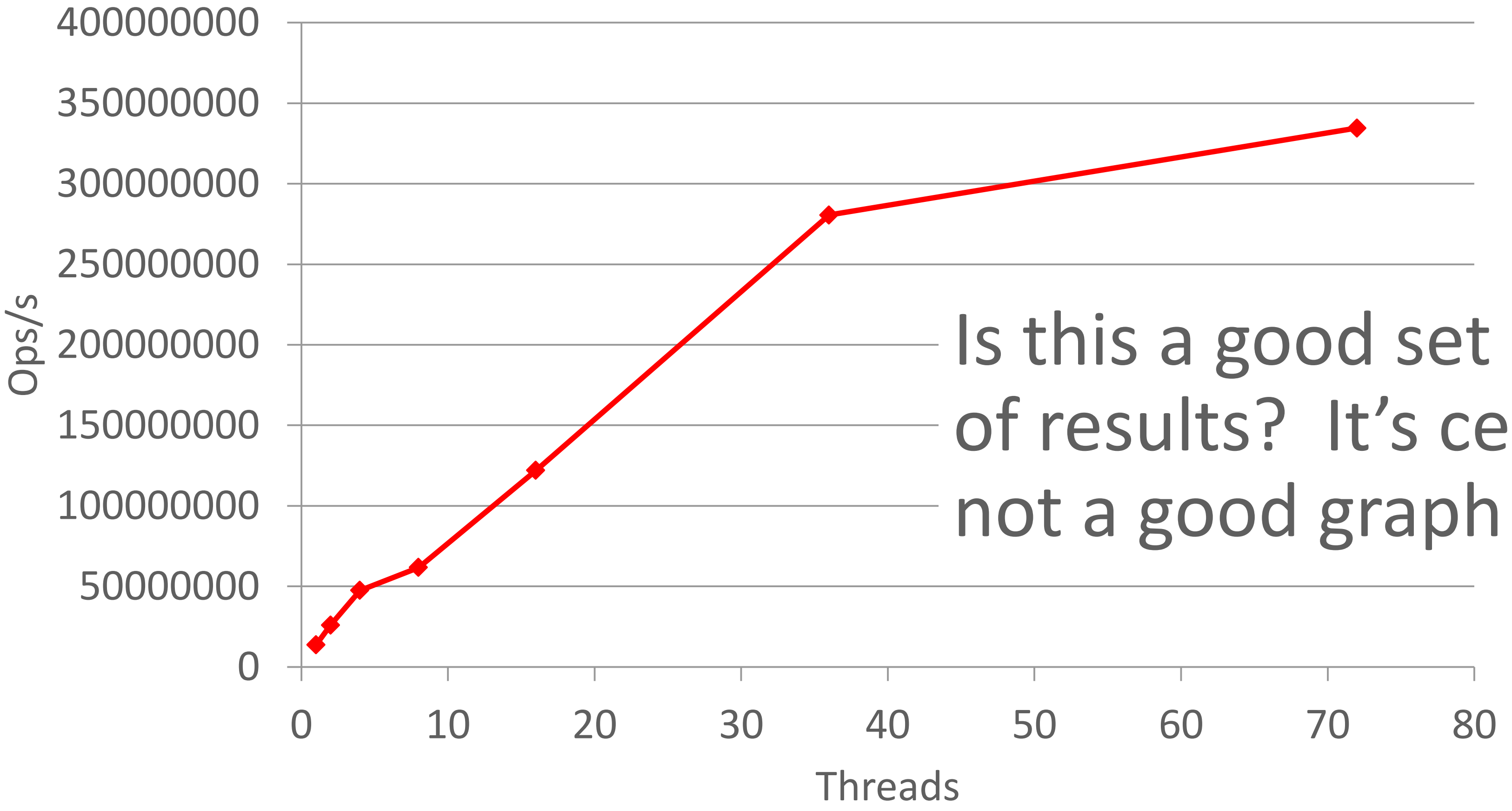
Basic checks to make

- Should the workload be 100% user mode?
  - Confirm this with “top”
  - Check that “strace” is quiet (no system call activity)
- Where are the threads running?
- Where is the memory they access located?
- What do profiling tools show?
  - Can you use with optimized builds? If not, check impact of disabling optimization
  - If you have long-running use cases, does the profile actually match them?
  - Look at 1-thread workloads – as expected?
  - Increase thread count and look for trends

Synchrobench, Fraser skip-list, 100 % read only, 2-socket Xeon

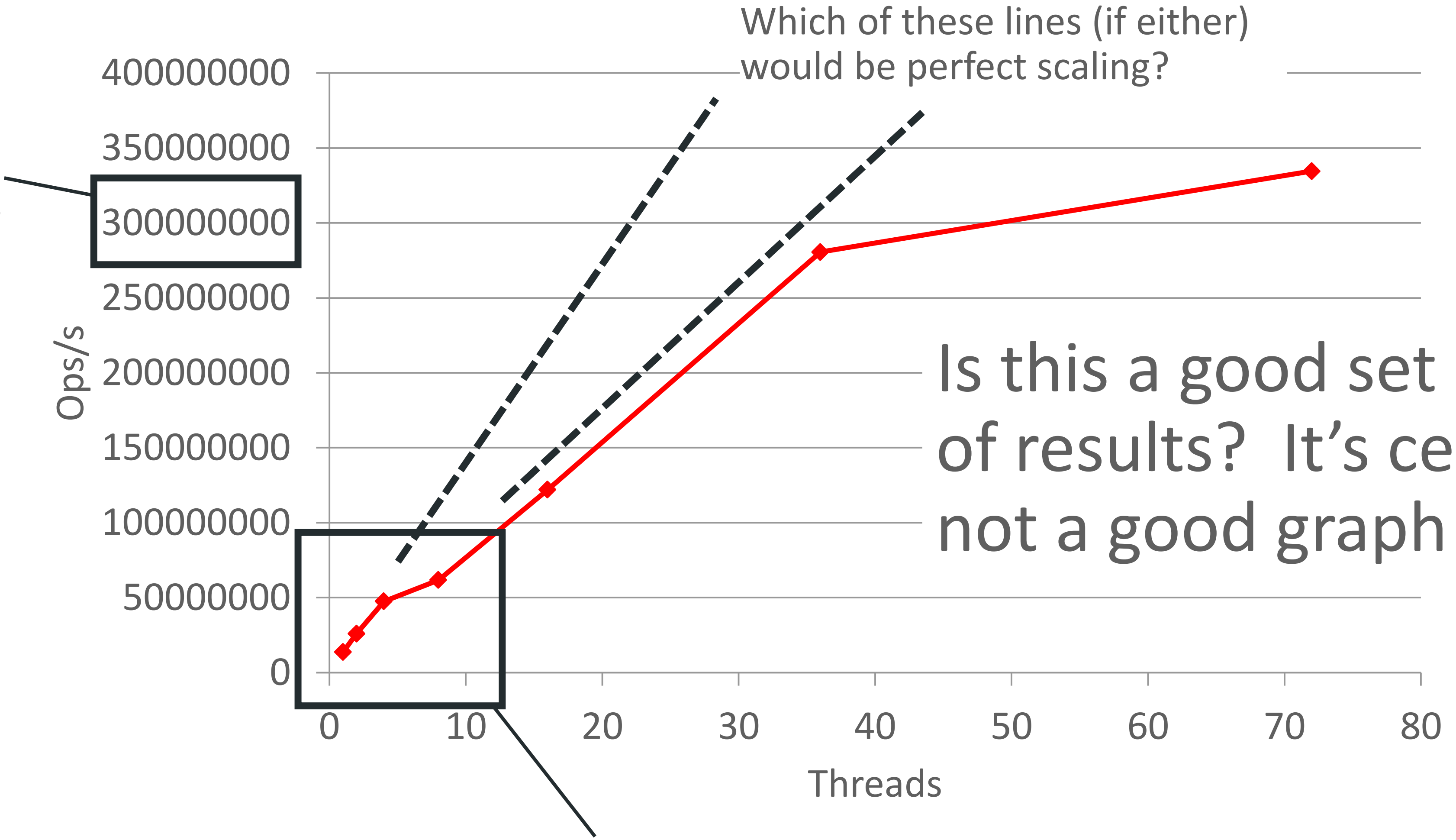


Synchrobench, Fraser skip-list, 100 % read only, 2-socket Xeon



Is this a good set of results? It’s certainly not a good graph

Synchrobench, Fraser skip-list, 100 % read only, 2-socket Xeon



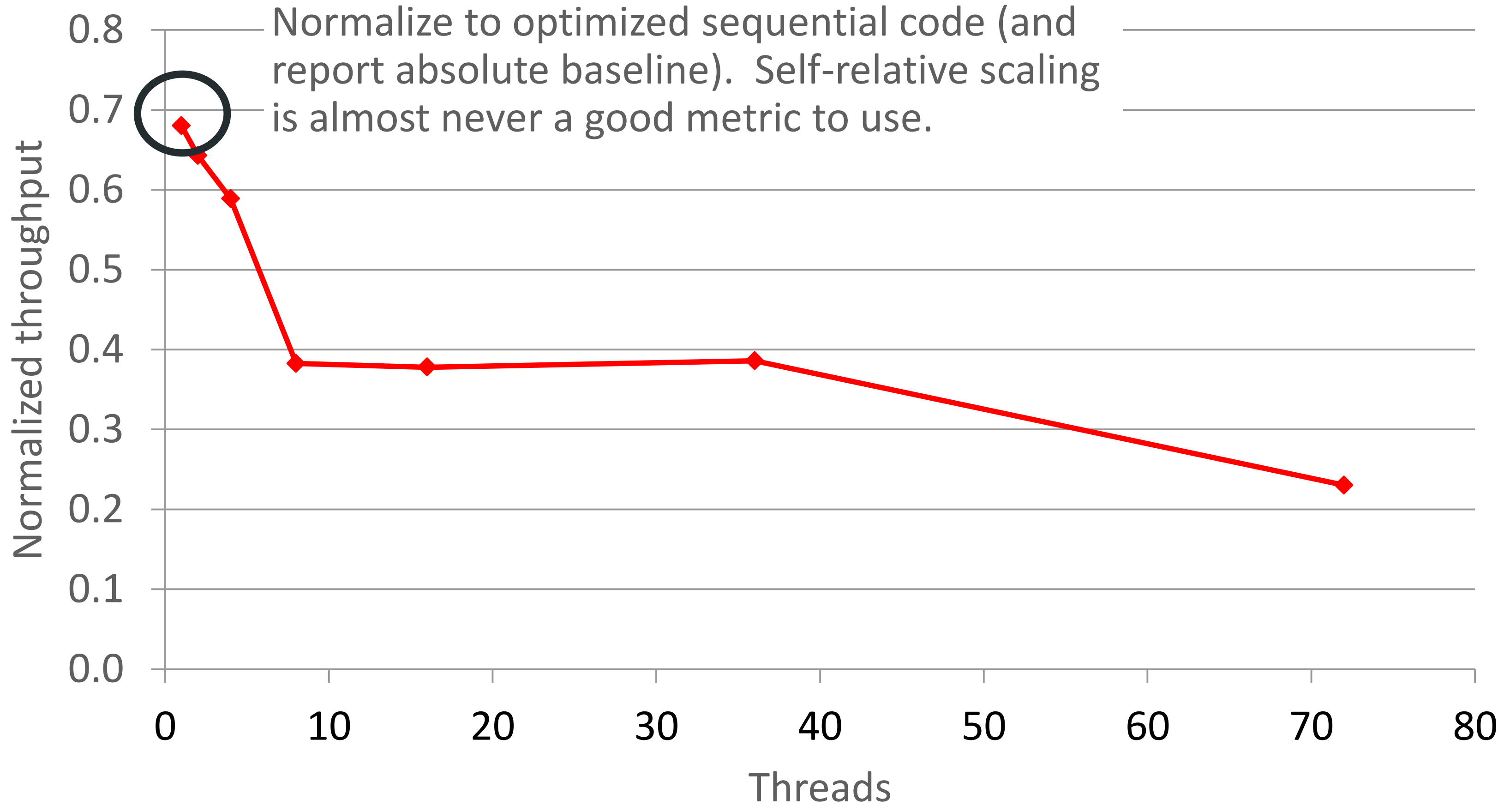
Which of these lines (if either) would be perfect scaling?

Ugly numbers. Is this good performance or poor?

Is this a good set of results? It's certainly not a good graph

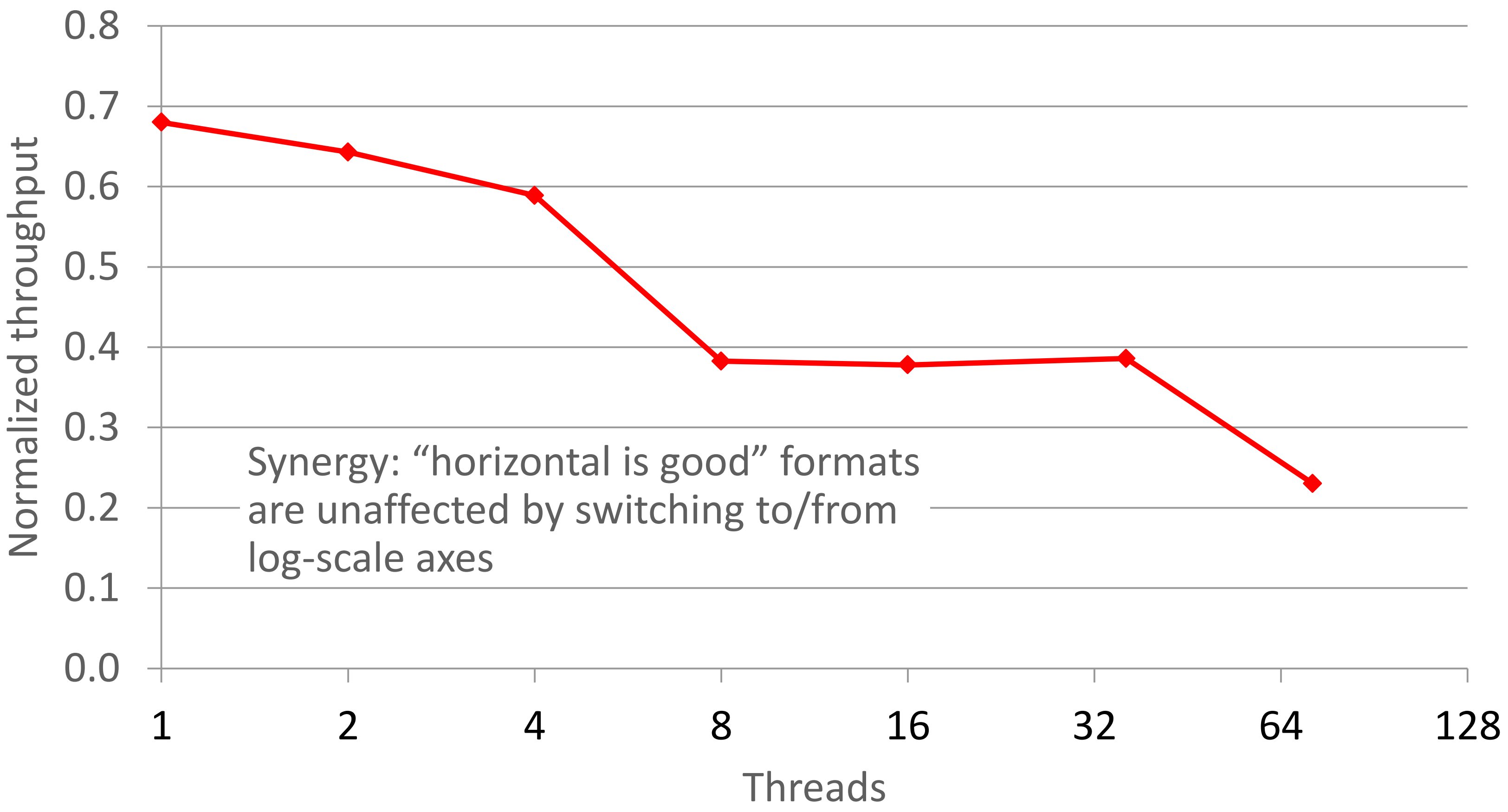
Most of the data is buried down here

Synchrobench, Fraser skip-list, 100 % read only, 2-socket Xeon



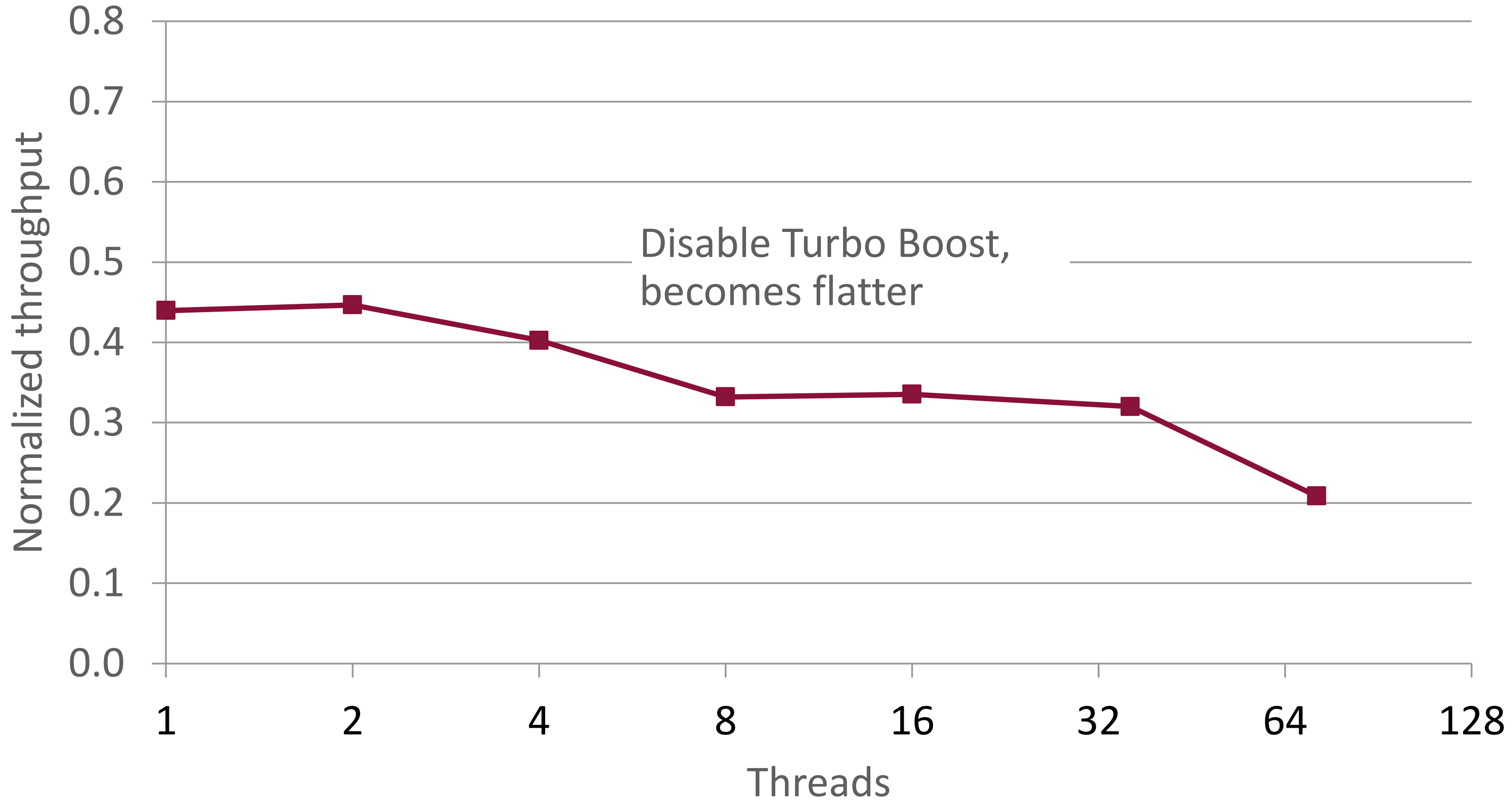
Normalize to optimized sequential code (and report absolute baseline). Self-relative scaling is almost never a good metric to use.

Synchrobench, Fraser skip-list, 100 % read only, 2-socket Xeon



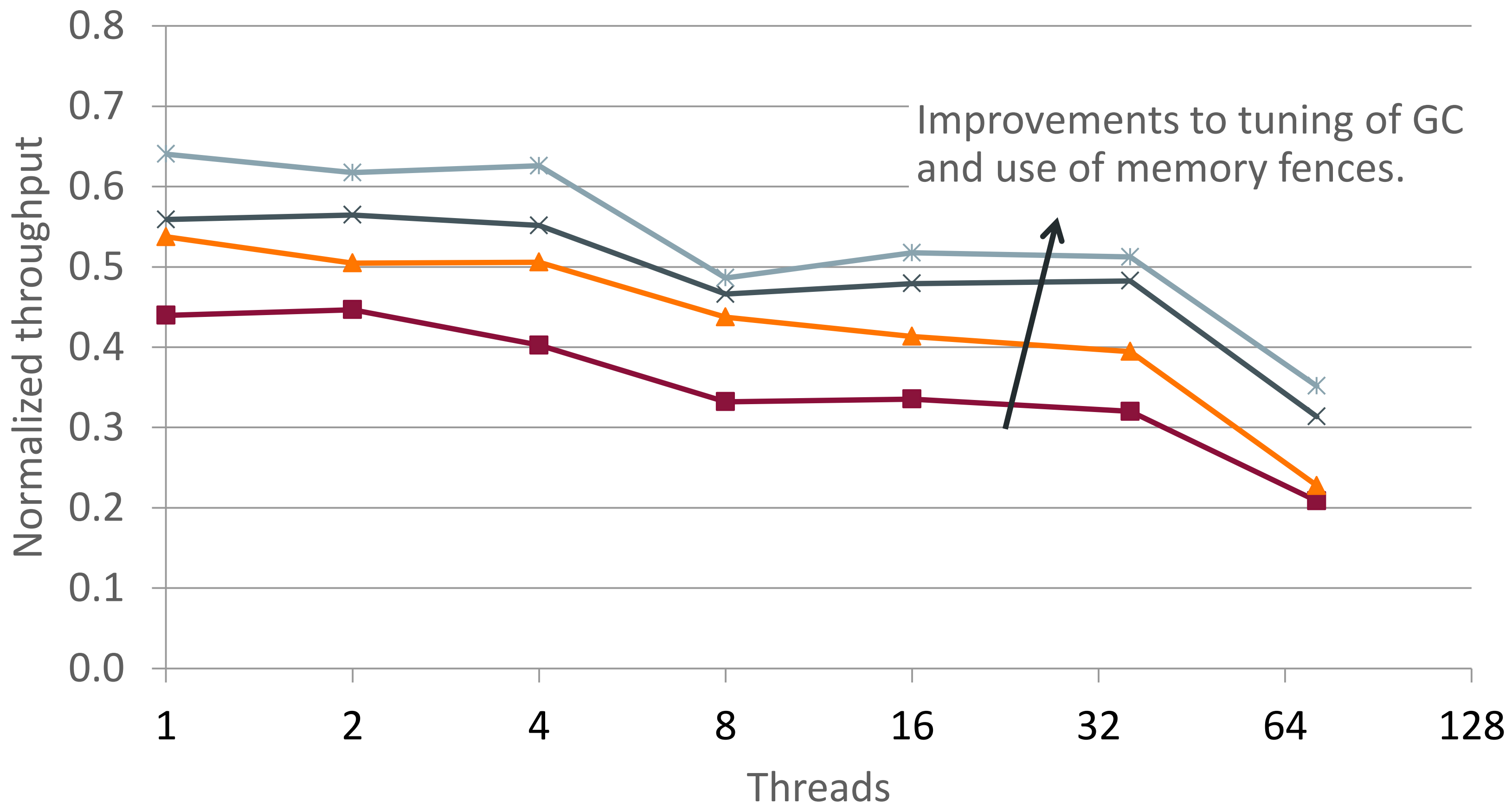
Synergy: “horizontal is good” formats are unaffected by switching to/from log-scale axes

Synchrobench, Fraser skip-list, 100 % read only, 2-socket Xeon



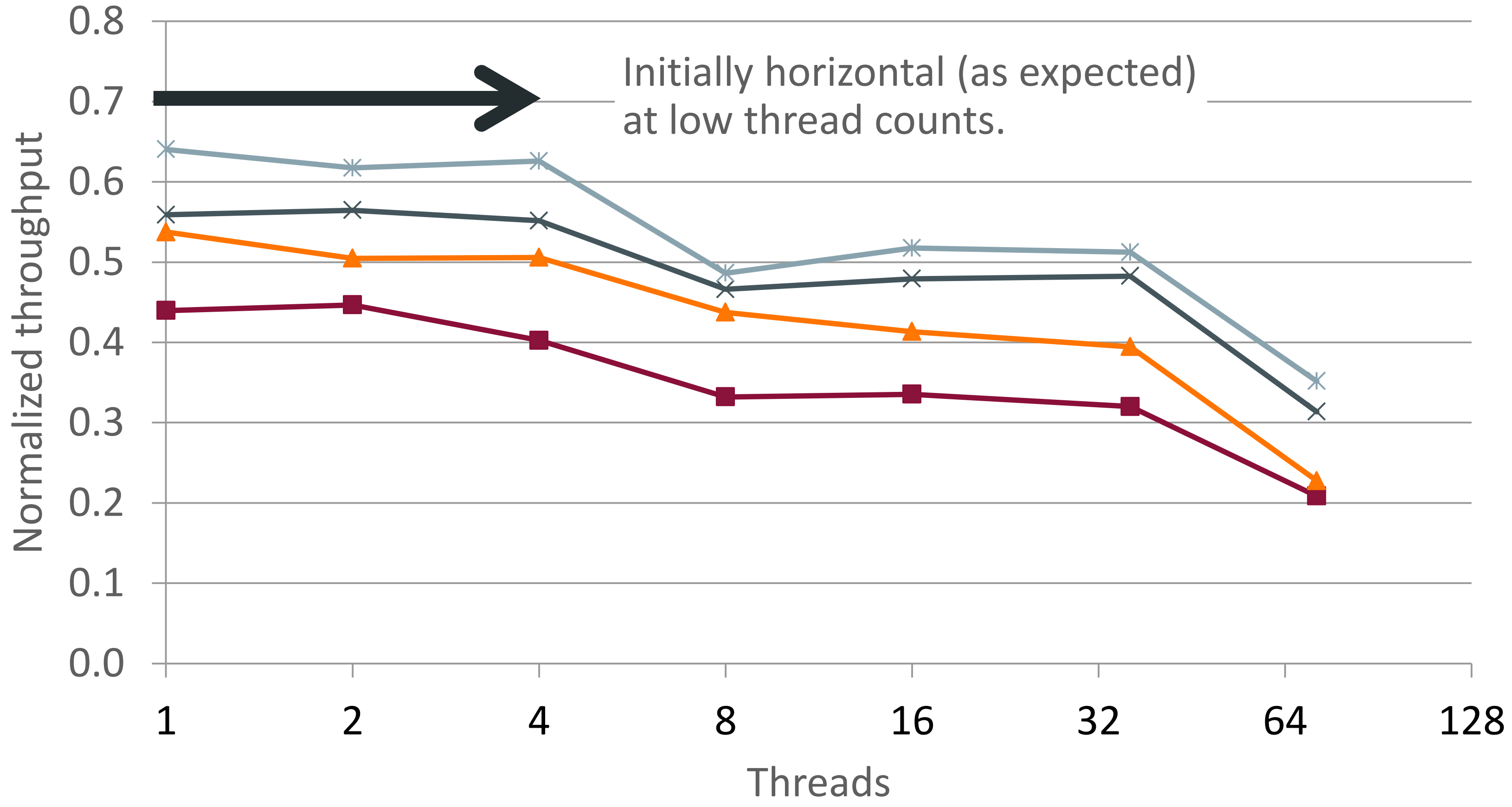
Disable Turbo Boost, becomes flatter

Synchrobench, Fraser skip-list, 100 % read only, 2-socket Xeon



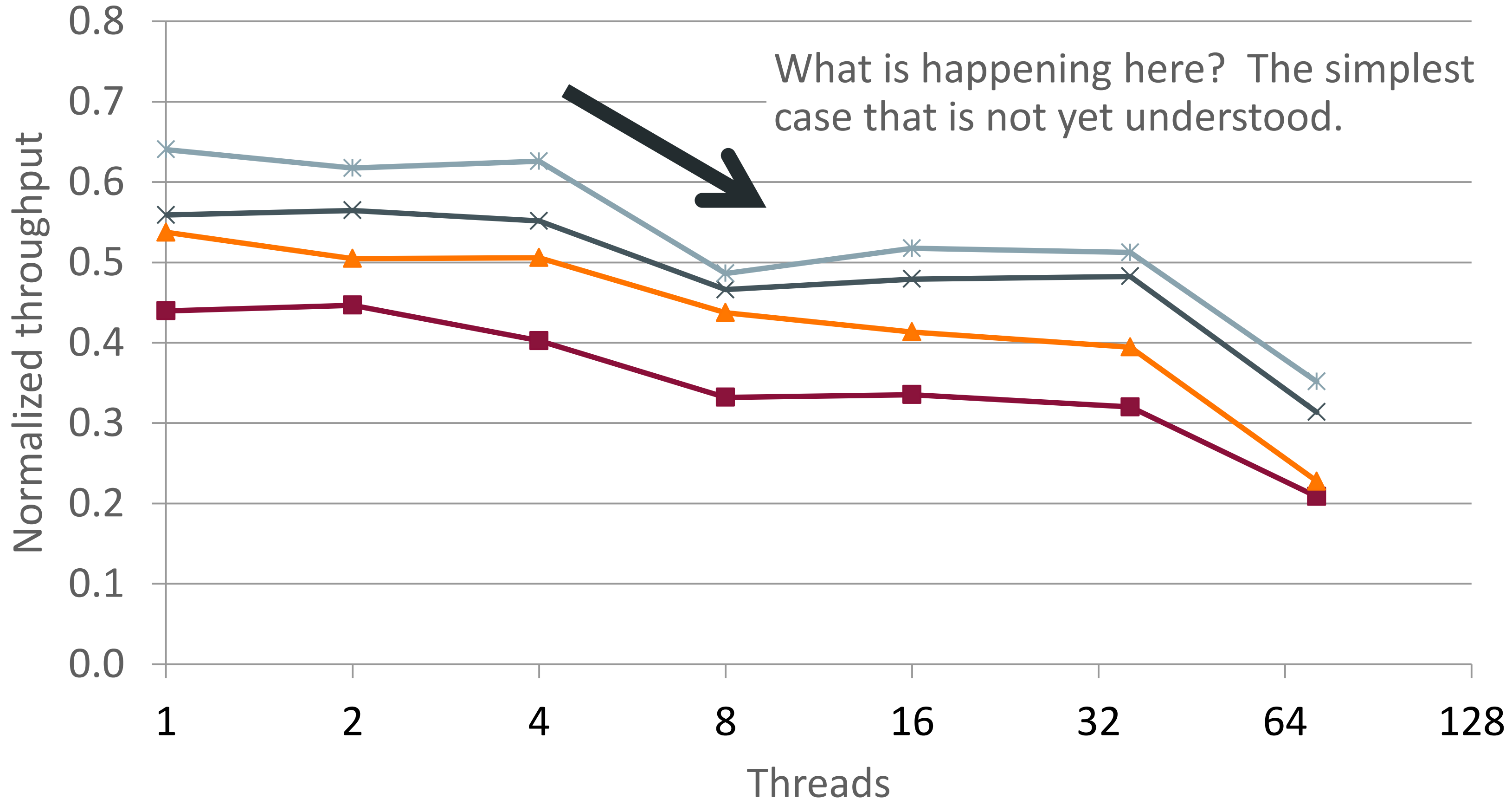
Improvements to tuning of GC and use of memory fences.

Synchrobench, Fraser skip-list, 100 % read only, 2-socket Xeon



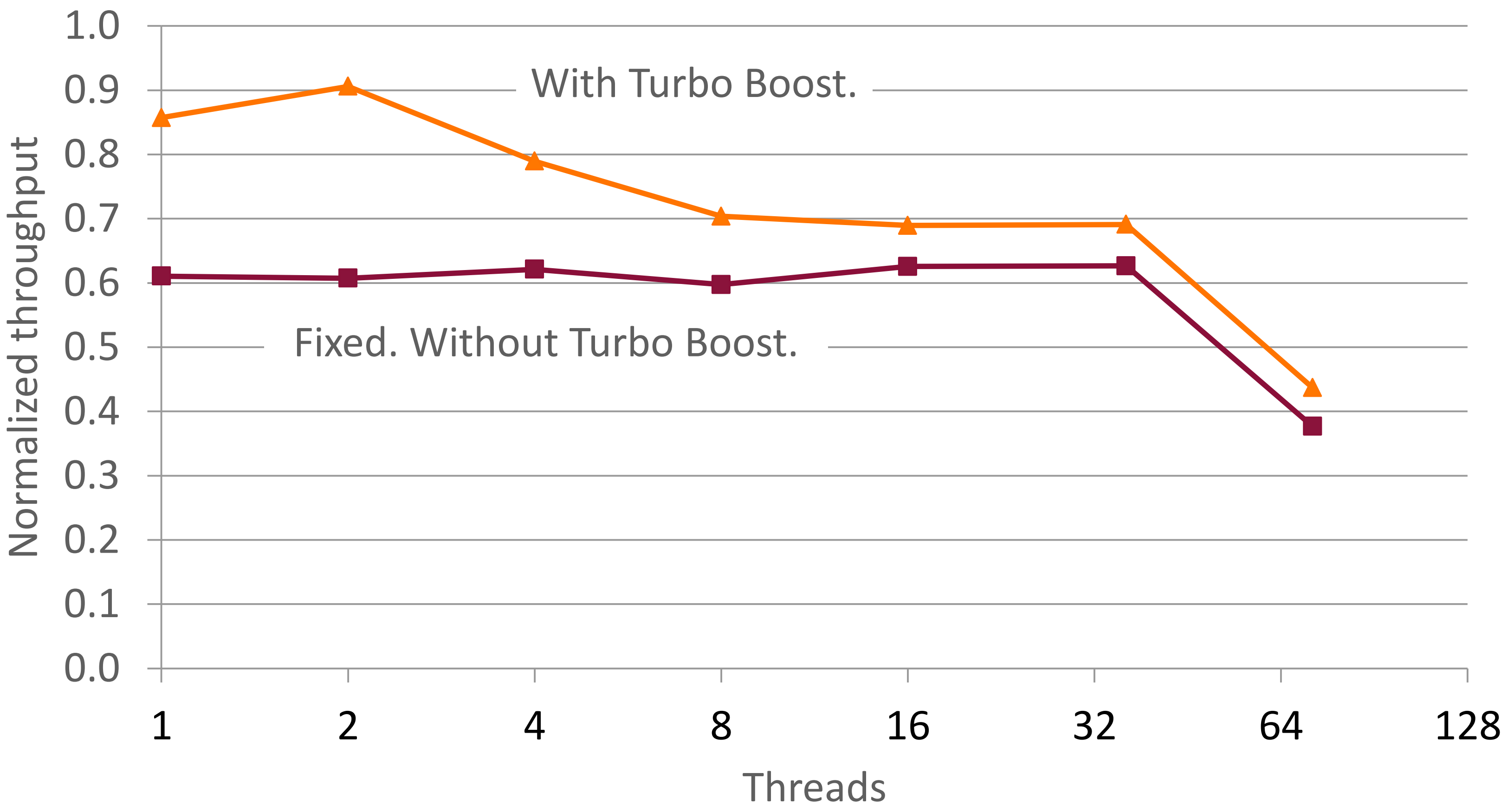
Initially horizontal (as expected) at low thread counts.

Synchrobench, Fraser skip-list, 100 % read only, 2-socket Xeon



What is happening here? The simplest case that is not yet understood.

(It was a stray process still running on the machine)



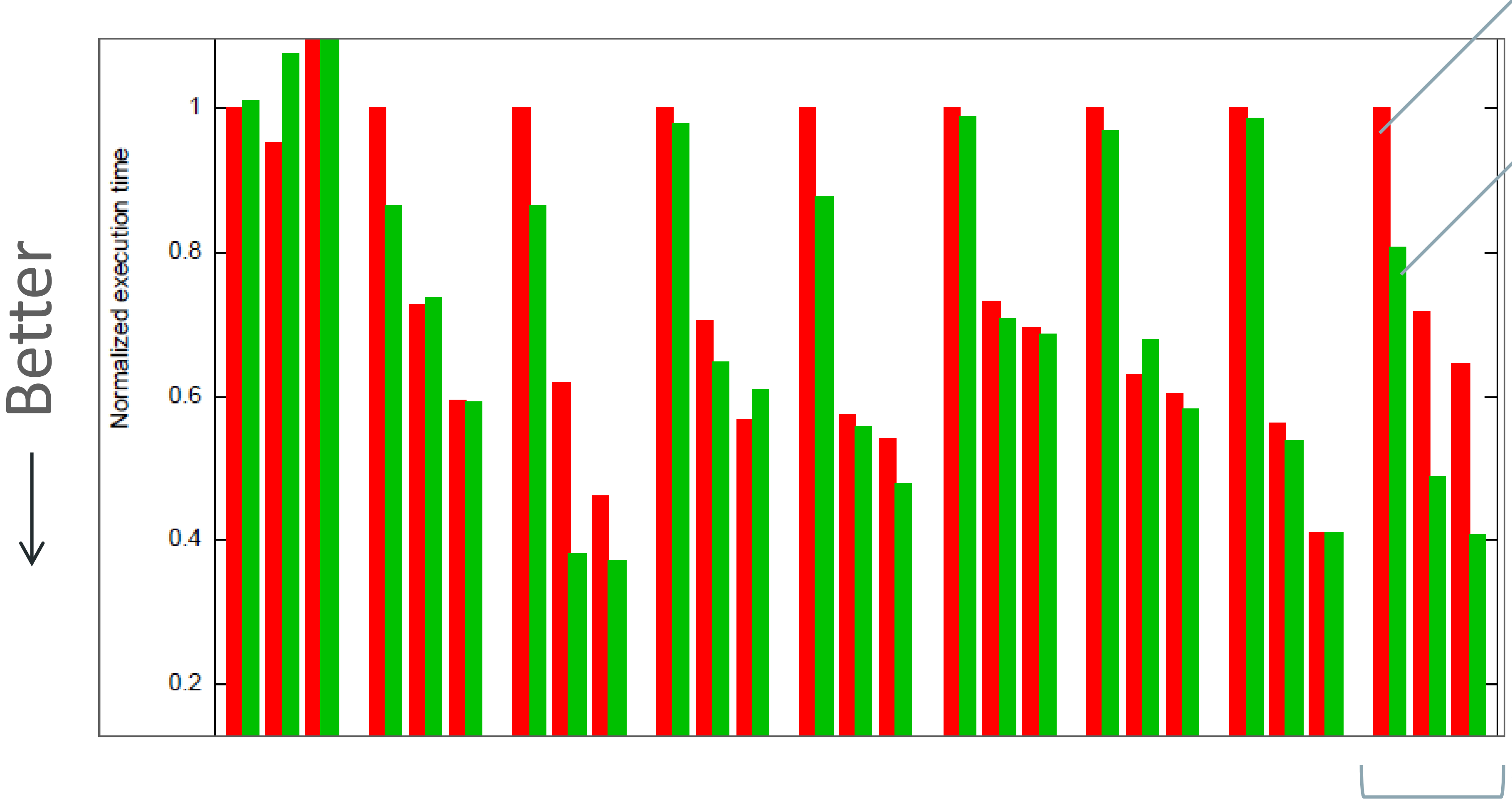
Overview

1 Script everything, derive results from measurements

2 Plan how to present results before starting work

3 Understand simple cases first

An example from my recent work

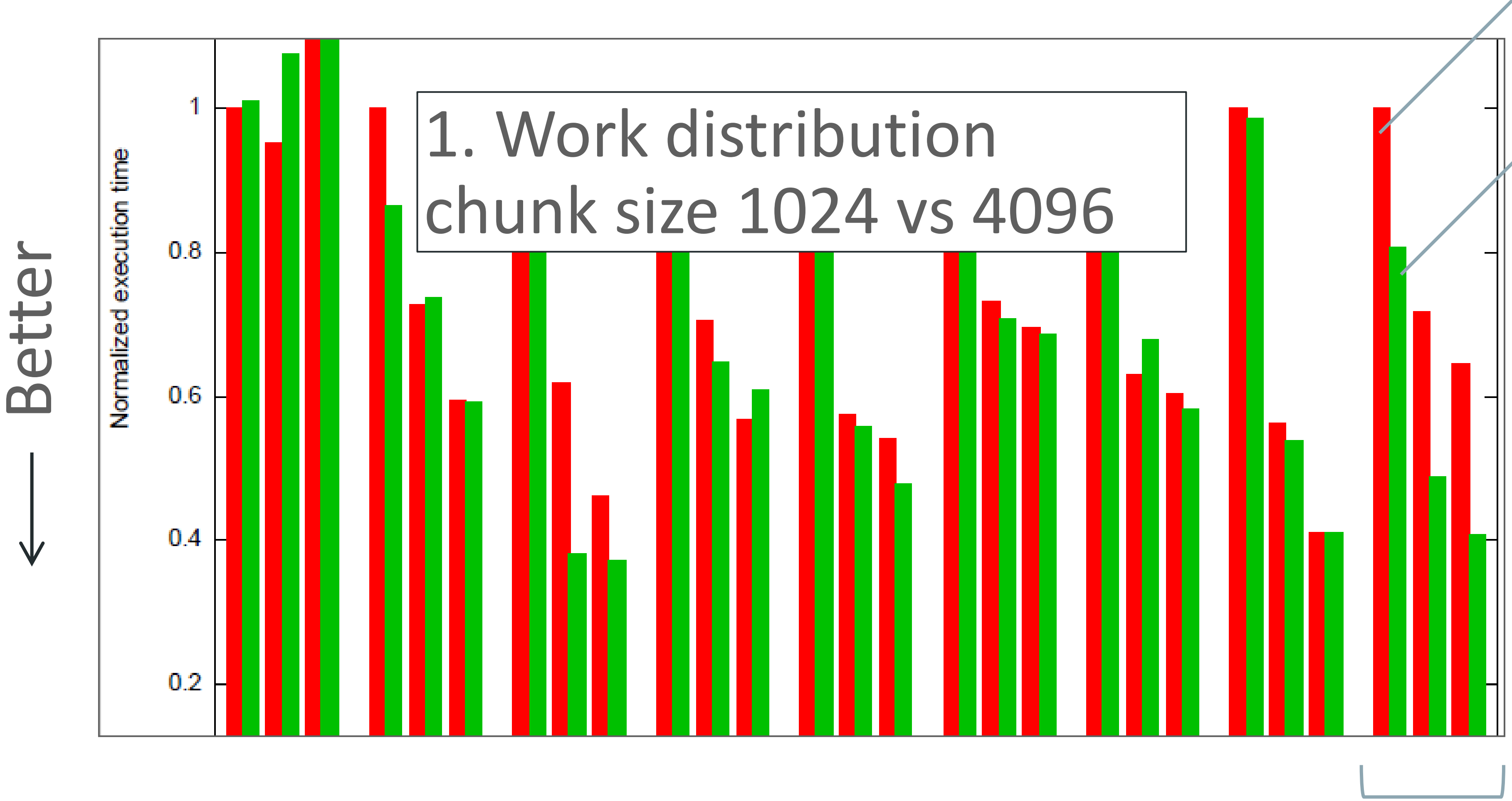


Previous system

New system

Algorithm running with 18/36/72 threads

An example from my recent work



1. Work distribution chunk size 1024 vs 4096

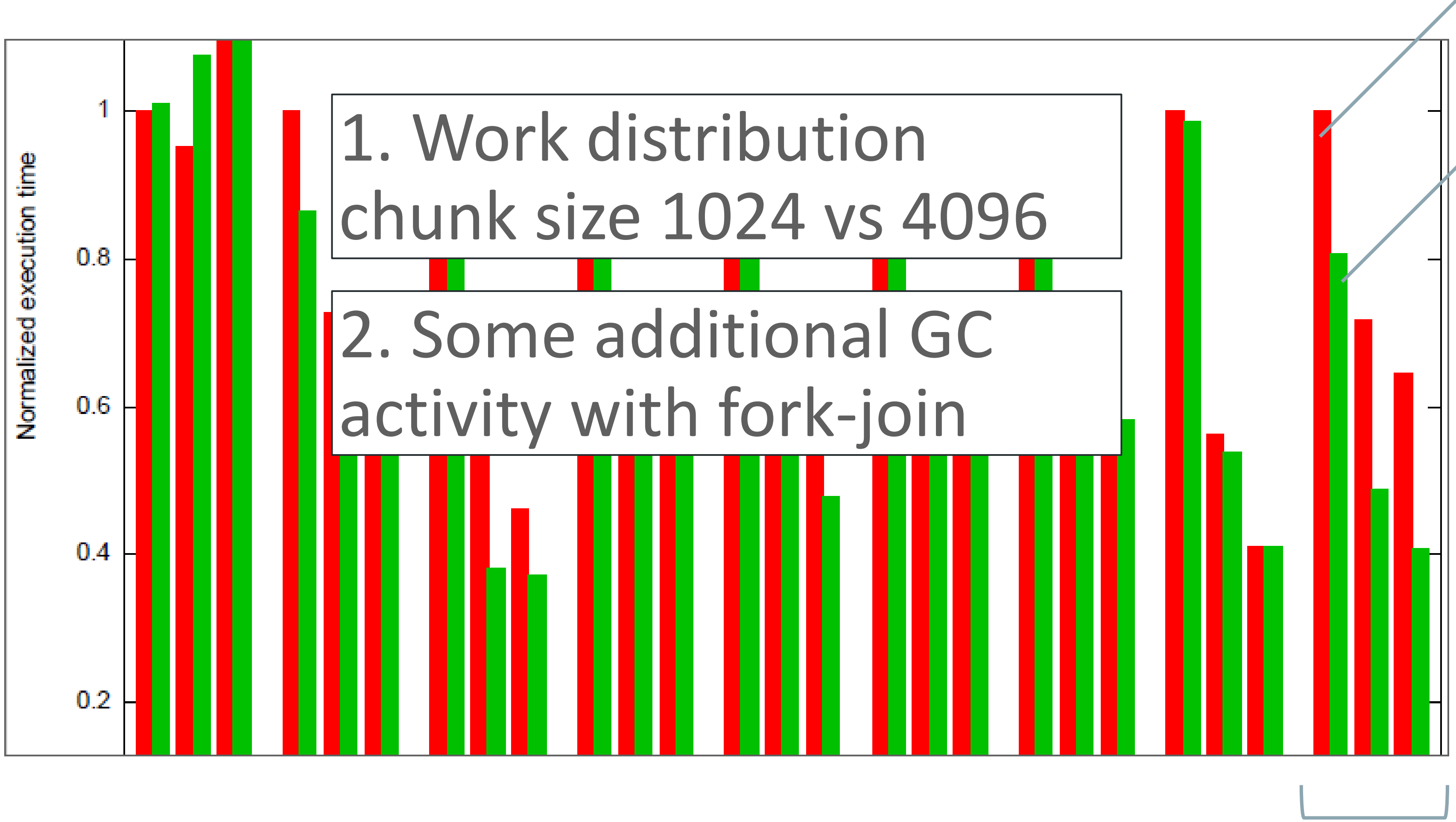
Previous system 🟥

New system 🟩

Algorithm running with 18/36/72 threads

An example from my recent work

Better ↓



1. Work distribution chunk size 1024 vs 4096

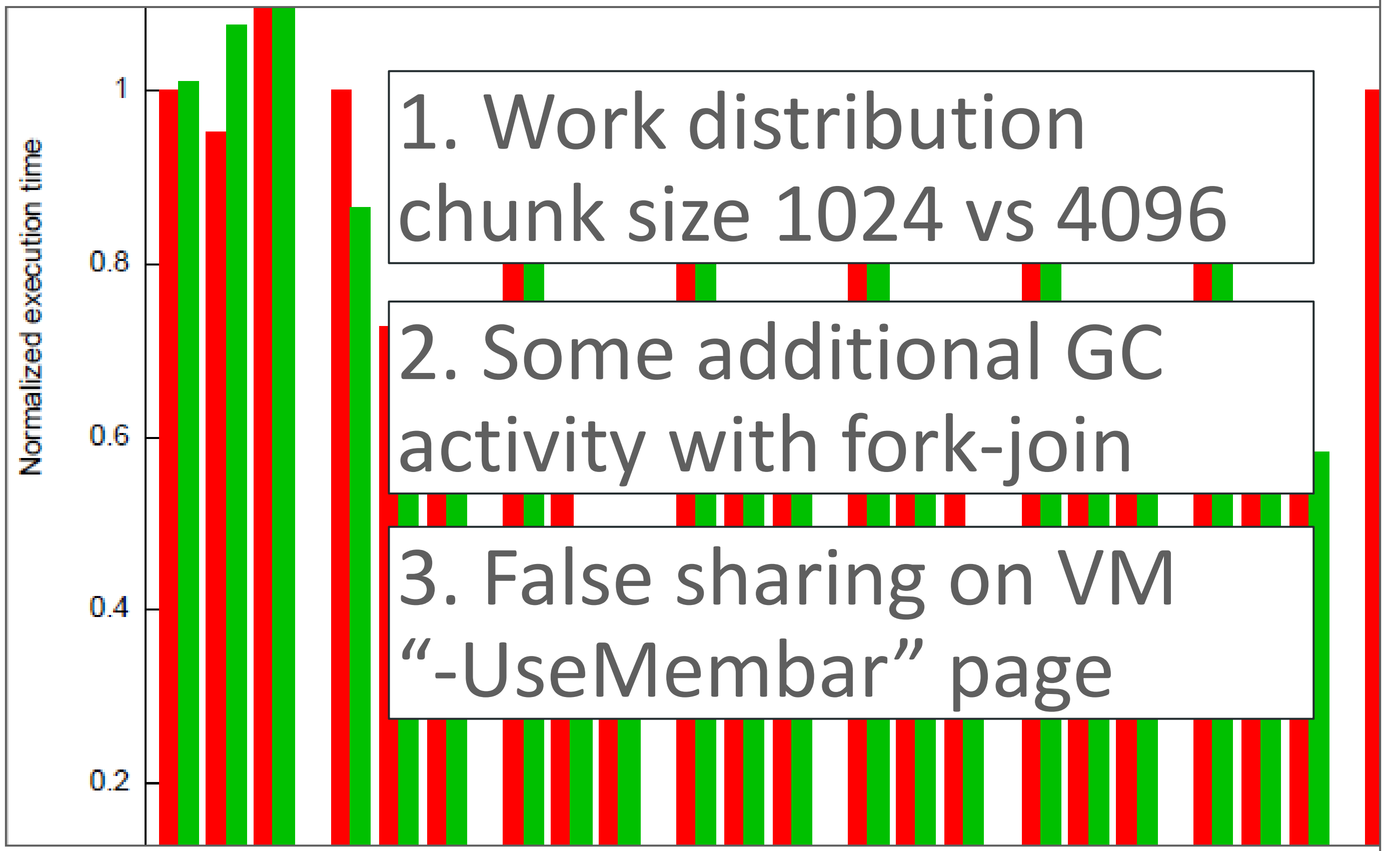
2. Some additional GC activity with fork-join

Previous system

New system

Algorithm running with 18/36/72 threads

An example from my recent work



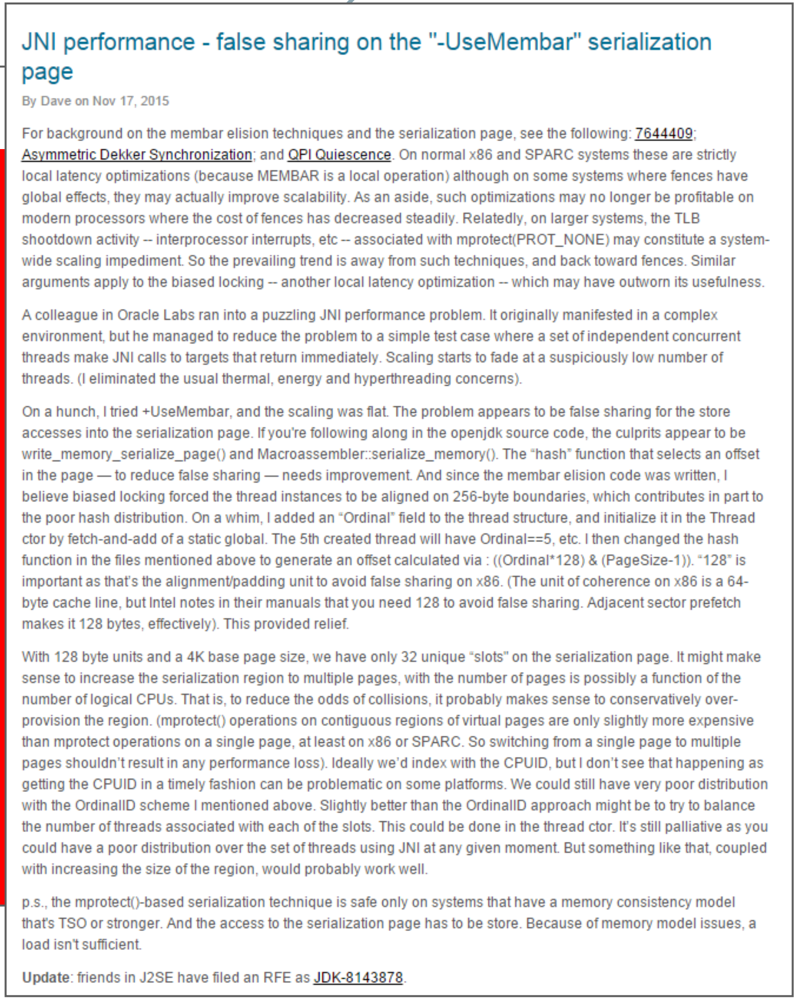
Better ↓

1. Work distribution chunk size 1024 vs 4096

2. Some additional GC activity with fork-join

3. False sharing on VM “-UseMembar” page

Previous system



JNI performance - false sharing on the "-UseMembar" serialization page

By Dave on Nov 17, 2015

For background on the membar elision techniques and the serialization page, see the following: 7644409; Asymmetric Dekker Synchronization; and QPI Quiescence. On normal x86 and SPARC systems these are strictly local latency optimizations (because MEMBAR is a local operation) although on some systems where fences have global effects, they may actually improve scalability. As an aside, such optimizations may no longer be profitable on modern processors where the cost of fences has decreased steadily. Relatedly, on larger systems, the TLB shootdown activity -- interprocessor interrupts, etc -- associated with mprotect(PROT_NONE) may constitute a system-wide scaling impediment. So the prevailing trend is away from such techniques, and back toward fences. Similar arguments apply to the biased locking -- another local latency optimization -- which may have outworn its usefulness.

A colleague in Oracle Labs ran into a puzzling JNI performance problem. It originally manifested in a complex environment, but he managed to reduce the problem to a simple test case where a set of independent concurrent threads make JNI calls to targets that return immediately. Scaling starts to fade at a suspiciously low number of threads. (I eliminated the usual thermal, energy and hyperthreading concerns).

On a hunch, I tried +UseMembar, and the scaling was flat. The problem appears to be false sharing for the store accesses into the serialization page. If you're following along in the openjdk source code, the culprits appear to be write_memory_serialize_page() and Macroassembler::serialize_memory(). The “hash” function that selects an offset in the page — to reduce false sharing — needs improvement. And since the membar elision code was written, I believe biased locking forced the thread instances to be aligned on 256-byte boundaries, which contributes in part to the poor hash distribution. On a whim, I added an “Ordinal” field to the thread structure, and initialize it in the Thread ctor by fetch-and-add of a static global. The 5th created thread will have Ordinal==5, etc. I then changed the hash function in the files mentioned above to generate an offset calculated via : ((Ordinal*128) & (PageSize-1)). “128” is important as that’s the alignment/padding unit to avoid false sharing on x86. (The unit of coherence on x86 is a 64-byte cache line, but Intel notes in their manuals that you need 128 to avoid false sharing. Adjacent sector prefetch makes it 128 bytes, effectively). This provided relief.

With 128 byte units and a 4K base page size, we have only 32 unique “slots" on the serialization page. It might make sense to increase the serialization region to multiple pages, with the number of pages is possibly a function of the number of logical CPUs. That is, to reduce the odds of collisions, it probably makes sense to conservatively over-provision the region. (mprotect() operations on contiguous regions of virtual pages are only slightly more expensive than mprotect operations on a single page, at least on x86 or SPARC. So switching from a single page to multiple pages shouldn’t result in any performance loss). Ideally we’d index with the CPUID, but I don’t see that happening as getting the CPUID in a timely fashion can be problematic on some platforms. We could still have very poor distribution with the OrdinalID scheme I mentioned above. Slightly better than the OrdinalID approach might be to try to balance the number of threads associated with each of the slots. This could be done in the thread ctor. It’s still palliative as you could have a poor distribution over the set of threads using JNI at any given moment. But something like that, coupled with increasing the size of the region, would probably work well.

p.s., the mprotect()-based serialization technique is safe only on systems that have a memory consistency model that's TSO or stronger. And the access to the serialization page has to be store. Because of memory model issues, a load isn't sufficient.

Update: friends in J2SE have filed an RFE as JDK-8143878.

Future work

- Three aspects to this talk:
  - Working practices to try to make sure there is time to understand results
  - Formats for presenting results to help understand them
  - Recurring problems from this particular area of research

Future work

- Three aspects to this talk:
  - Working practices to try to make sure there is time to understand results
  - Formats for presenting results to help understand them
  - Recurring problems from this particular area of research
- I would like to have more common infrastructure for running experiments
  - Help run experiments consistently
  - Same allocator, same thread placement, …
  - Use raw output logs as part of artefact evaluation processes
  - By using it, help convince others that experiments are run well

Further reading

- Books
  - Huff & Geis – “How to Lie with Statistics”
  - Jain – “The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling”
  - Tufte – “The Visual Display of Quantitative Information”
- Papers and articles
  - Bailey – “Twelve Ways to Fool the Masses”
  - Fleming & Wallace – “How not to lie with statistics: the correct way to summarize benchmark results”
  - Heiser – “Systems Benchmarking Crimes”
  - Hoefler & Belli – “Scientific Benchmarking of Parallel Computing Systems”

Integrated Cloud

Applications & Platform Services

ORACLE®