DevOps: Concerning Developers ...

Petr Tůma

Department of
Distributed and
Dependable
Systems

FACULTY OF MATHEMATICS AND PHYSICS
CHARLES UNIVERSITY IN PRAGUE

... How It All Started

We have this nice workshop about DevOps ...

in some panic - What exactly is DevOps ?

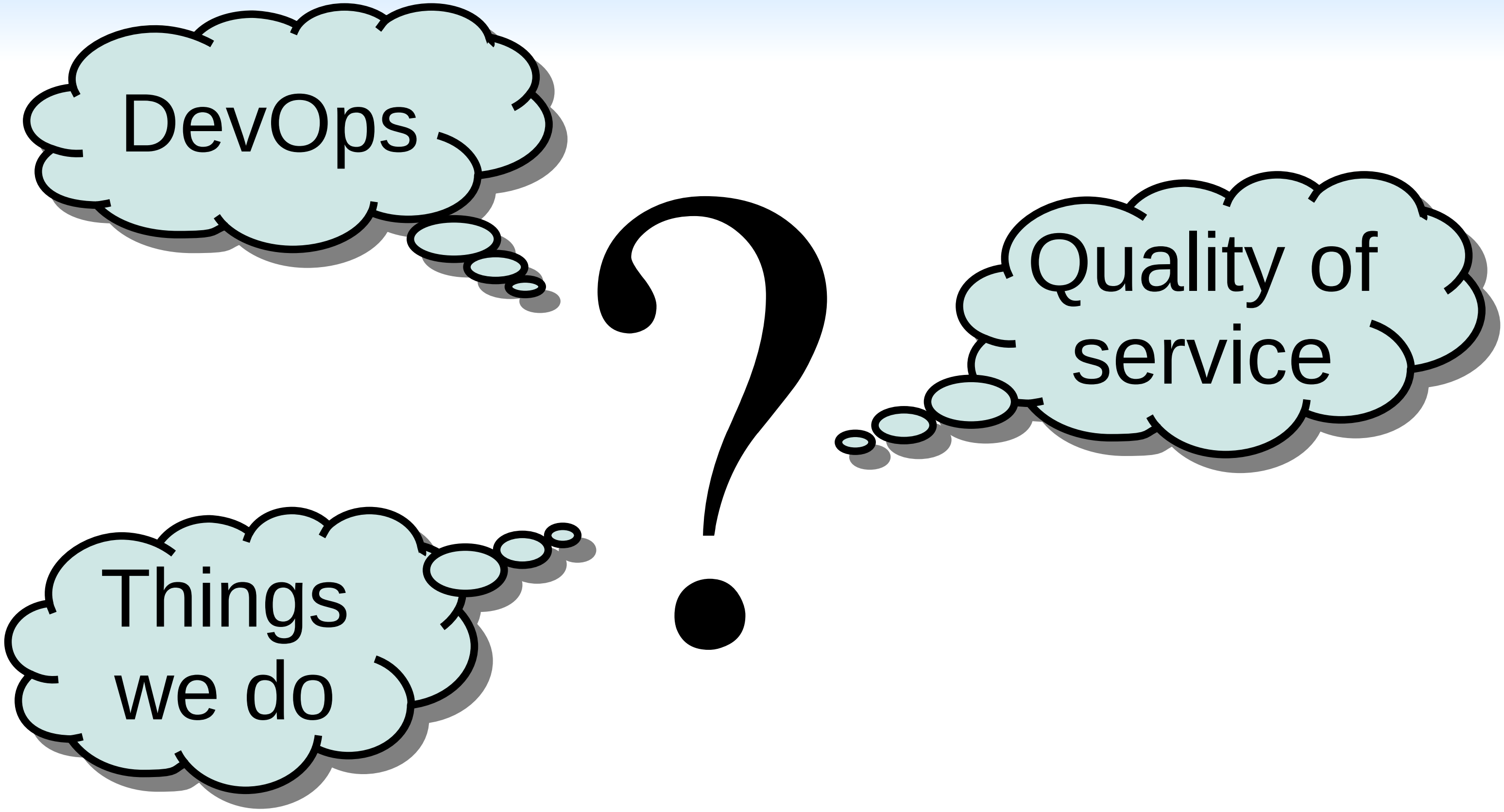
We have this nice whitepaper about DevOps ...

reading - I think we do Dev but not Ops.

I know. And that is where you miss all the fun !

* The words could have been entirely different :-)

Department of
Distributed and
Dependable
Systems
D3S



DevOps

Quality of service

Things we do

?

DevOps for Developers

see how we can reach the actual programmers

Why Developers ?

This could be an **entirely wrong idea !**

- Interaction is about **teams**
- Programmers are **not** team **interface**
- Quality of service management is more for **architects** and **performance engineers**

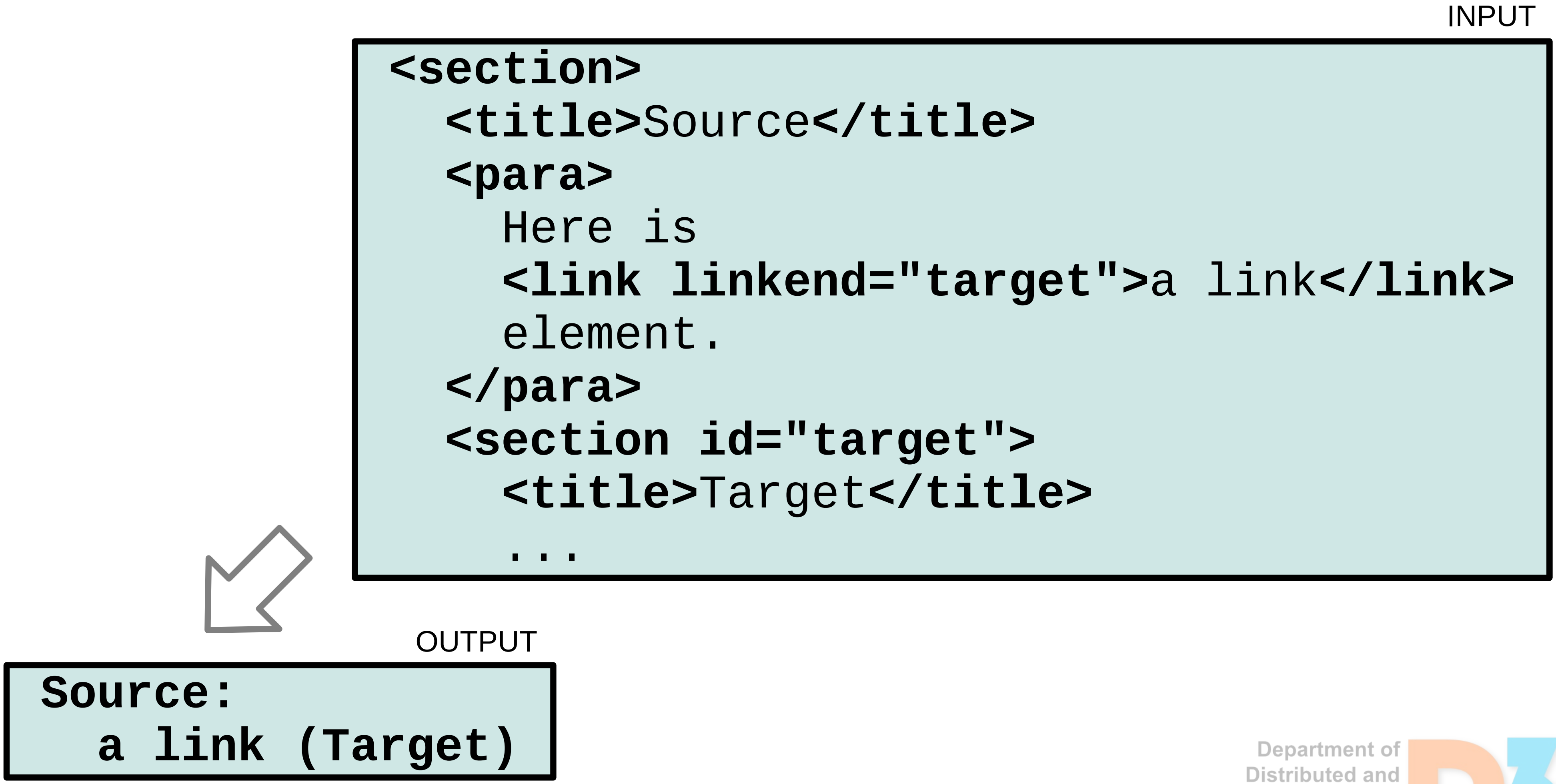
But perhaps there is something to it

- It is hard to imagine building **good system** from **crap code**
- Agile experts vs mindless drones ?
- Much harm can be done at code level ...

Just One Way To Code This ?

Assignment

- Read an XML document
- Display a table of references



INPUT

```
<section>
  <title>Source</title>
  <para>
    Here is
    <link linkend="target">a link</link>
    element.
  </para>
  <section id="target">
    <title>Target</title>
    ...
```

OUTPUT

```
Source:
  a link (Target)
```

Just One Way To Code This ?

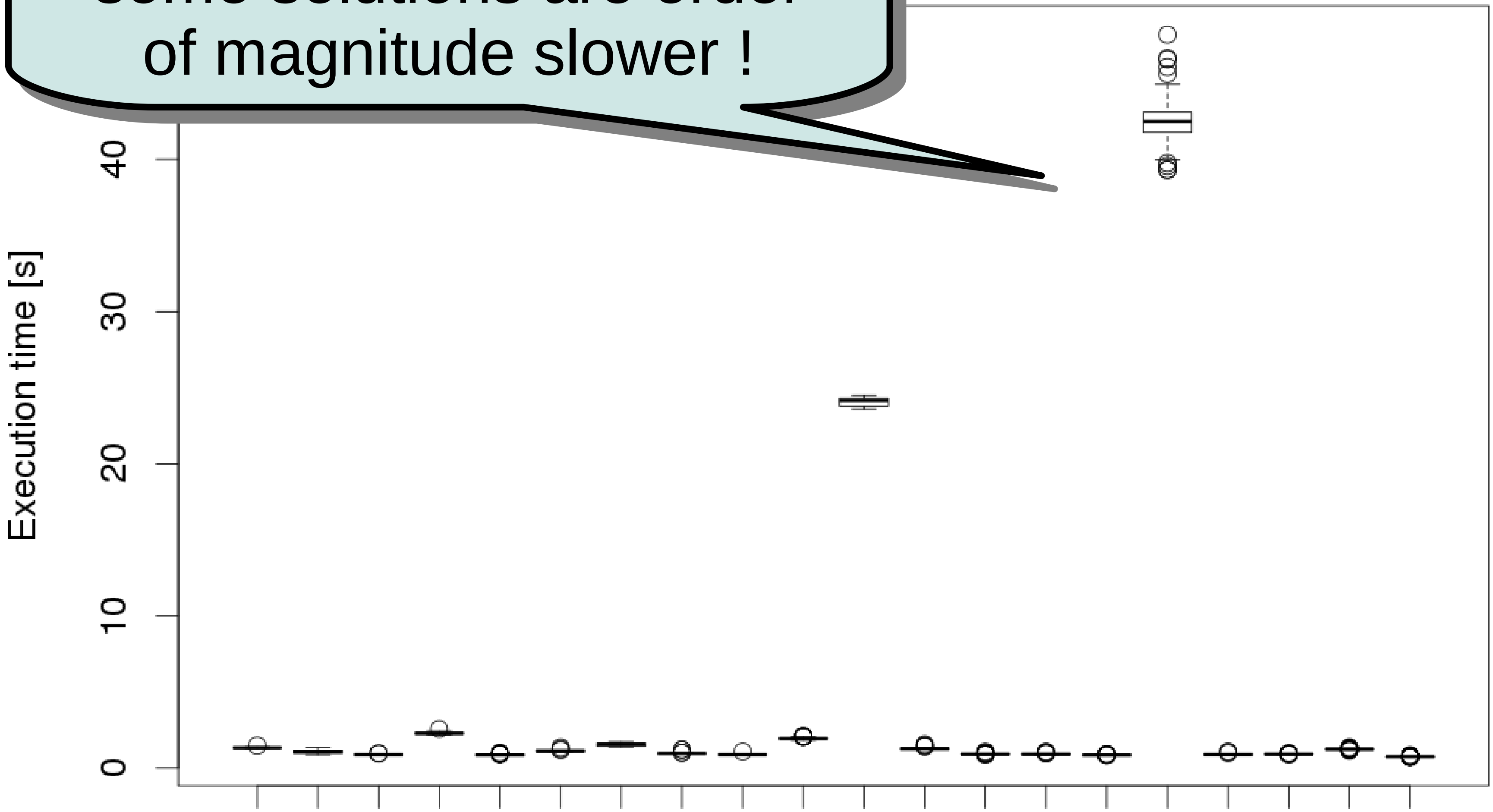
Assignment

- Read an XML document
- Display a table of references

Considerations

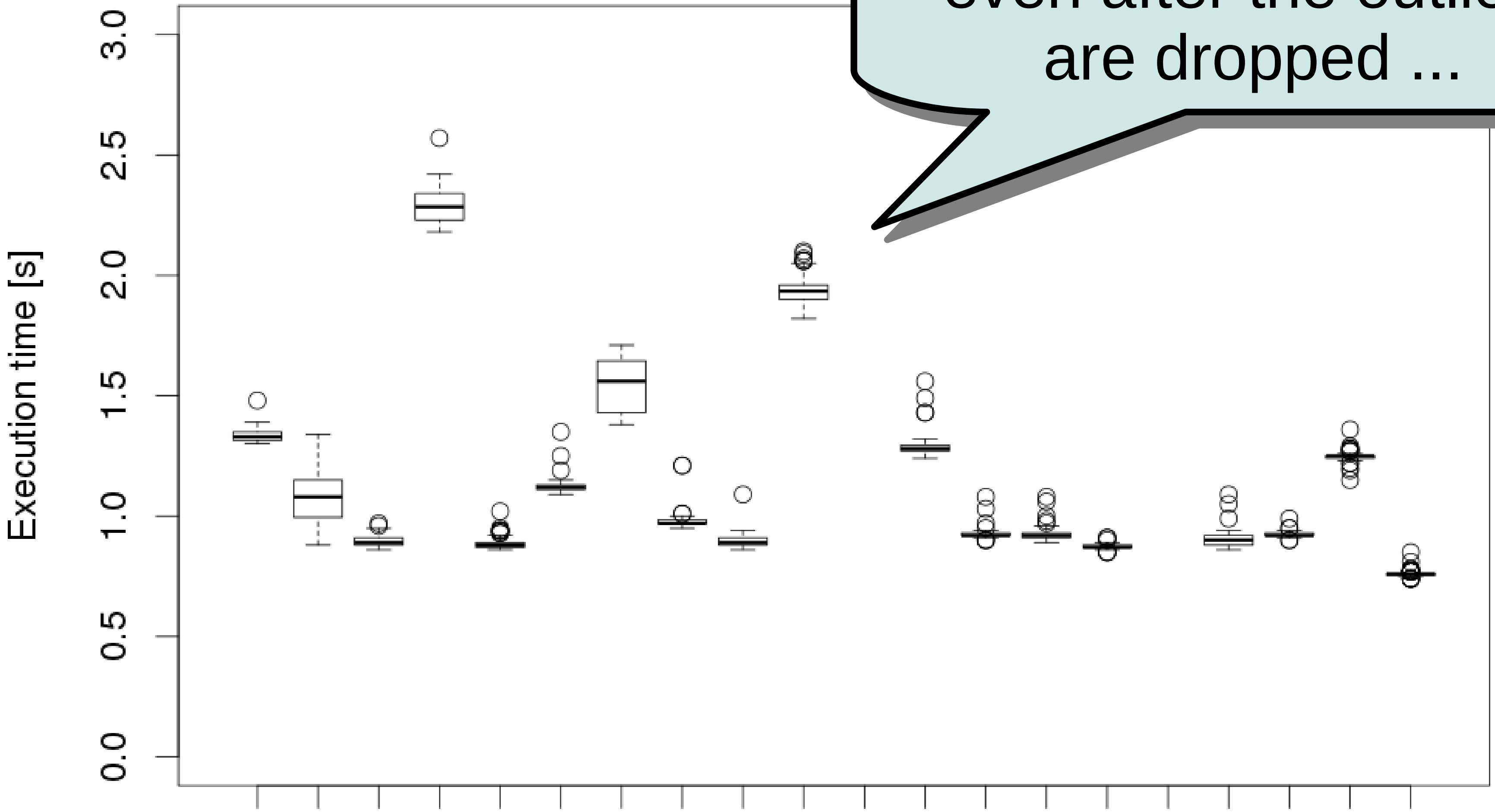
- Students on advanced programming course
- Library for manipulating XML data provided
- No complex data structures required
  - Sequence of sections with references
  - Mapping from identifiers to sections
- Basic timing information provided
- Example inputs provided

Just One Way To Code This ?



On an 8 MB XML document some solutions are order of magnitude slower !

Just One Way To Code This ?



A lot of variance remains even after the outliers are dropped ...

Department of Distributed and Dependable Systems

Survey After Coding

Code

- Mostly but not always functionally correct
- Complexity anywhere from $O(n)$ to $O(n^3)$

Attitude

- Complexity mostly but not always judged correctly
- Execution time almost never guessed correctly
- Memory consumption considered irrelevant
- Input size in megabytes considered too small to deserve optimizing

Many of the mistakes are easily corrected by **runtime observations**

Historical Excursion

how we worked on middleware performance evaluation and what we learned about supporting developers

Middleware Performance

1991

- CORBA 1.0 specification released
- Pricing eventually from free to thousands of $ per runtime

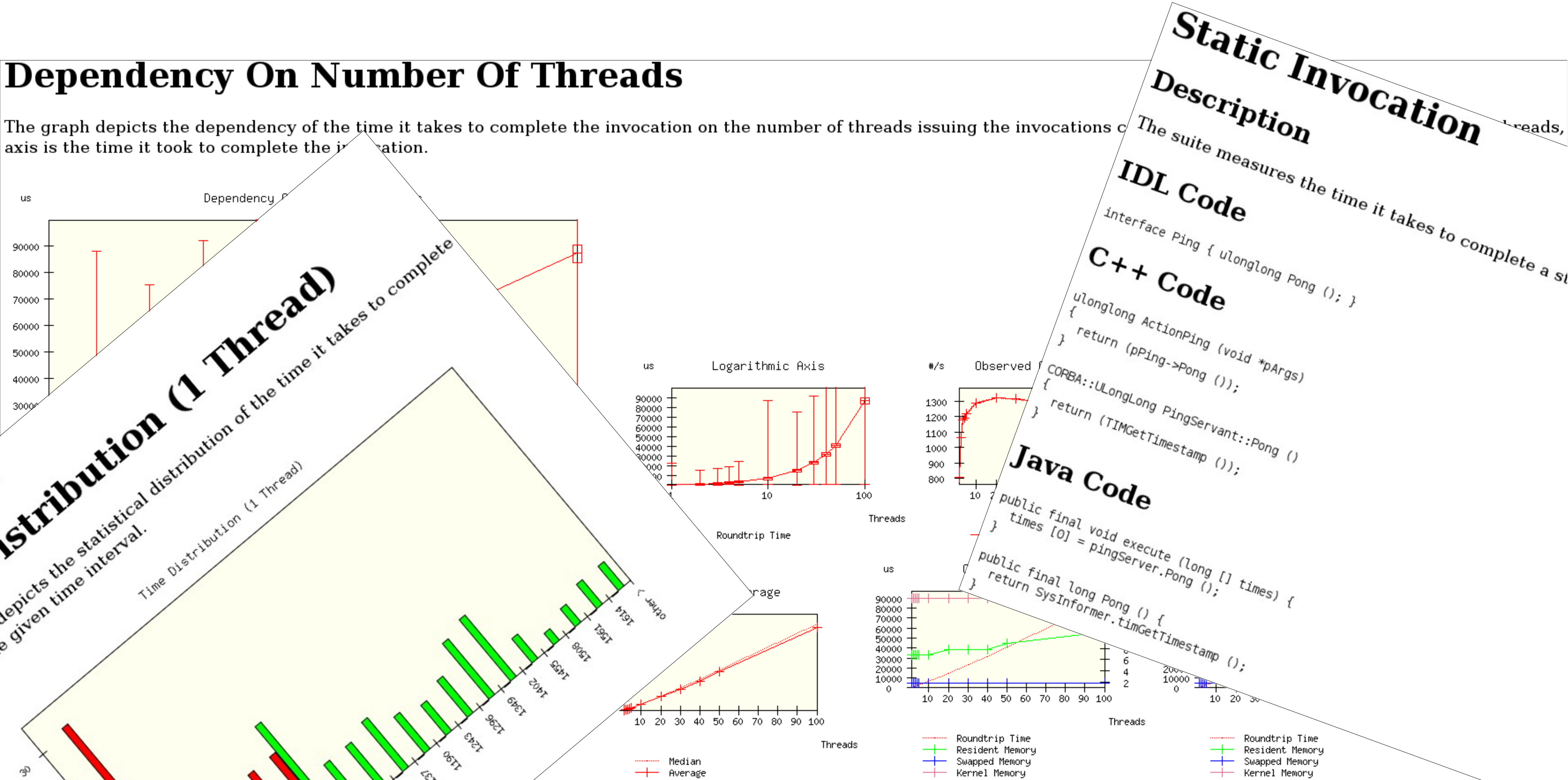
How to examine performance ?

- Eventual application workloads not very clear
- Features few and clearly defined
  - Measure each feature in isolation
  - Measure reasonable combinations
  - Report measurement results
    - Graphs
    - Anomalies

Middleware Performance

Results

- Completely automated evaluation environment
- Reports hundreds of pages per platform



Dependency On Number Of Threads

The graph depicts the dependency of the time it takes to complete the invocation on the number of threads issuing the invocations c… axis is the time it took to complete the i… ation.

…stribution (1 Thread)

…depicts the statistical distribution of the time it takes to complete … e given time interval.

Static Invocation

Description

The suite measures the time it takes to complete a s…

IDL Code

```
interface Ping { ulonglong Pong (); }
```

C++ Code

```
ulonglong ActionPing (void *pArgs)
{
  return (pPing->Pong ());
}
CORBA::ULongLong PingServant::Pong ()
{
  return (TIMGetTimestamp ());
}
```

Java Code

```
public final void execute (long [] times) {
  times [0] = pingServer.Pong ();
}
public final long Pong () {
  return SysInformer.timGetTimestamp ();
}
```

Lessons Learned

Full Automation Achievable

but sometimes extremely tricky

Everybody Loves Graphs

for the first five minutes

Persisting Issues

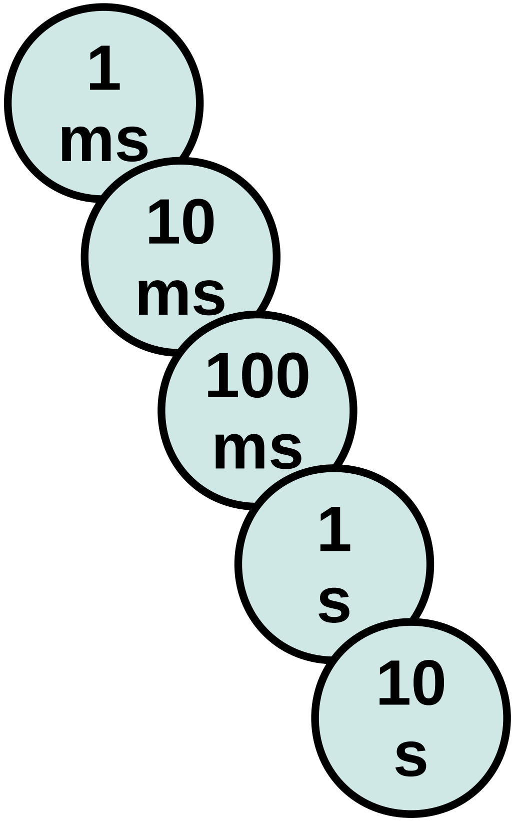
Results **difficult to interpret** correctly

- Lack of feel for actual numbers
- Some conclusions cross graphs
- Eventually requires looking at sources
  - Workload plus application plus platform
  - Developer only wrote application

Recall student experiment

Significant expenses in terms of **time**

- Measurement time does not scale
  - Workloads and configurations
  - Large basic constants
- Developer time is even worse
  - Notifications rather than results
  - False alarms very irritating



Where We Have Moved Since

1. Performance Specifications
2. Performance Unit Testing
3. Performance Documentation

Performance Specifications

if a developer specifies performance requirements we can ***save time*** *by only measuring relevant data and* ***target reporting*** *at specific requirements*

Perf Spec Wish List

Appropriate **granularity**

- Methods, classes, perhaps modules
- Not about end user visible transactions
  - Absolute timing rarely available
  - Timing depends on workload

Suitable for **vaguely defined constraints**

- “X should now be faster than before”
- “X should not be (much) slower than Y”
- “X should scale for practical range of inputs”

Working with **real measurements**

- Noise and other artifacts
- Platform portability

SPL Specifications

Formal language for performance specifications

“on input sizes of 1000 and 5000, NewFunction should be at least 10% faster than OldFunction”

Quantifiers over finite sets

Performance as random variable

$$\forall s \in \{1000, 5000\}:$$
$$\mathrm{NewFunction}\ (s) \leq 0.9 \times \mathrm{OldFunction}\ (s)$$

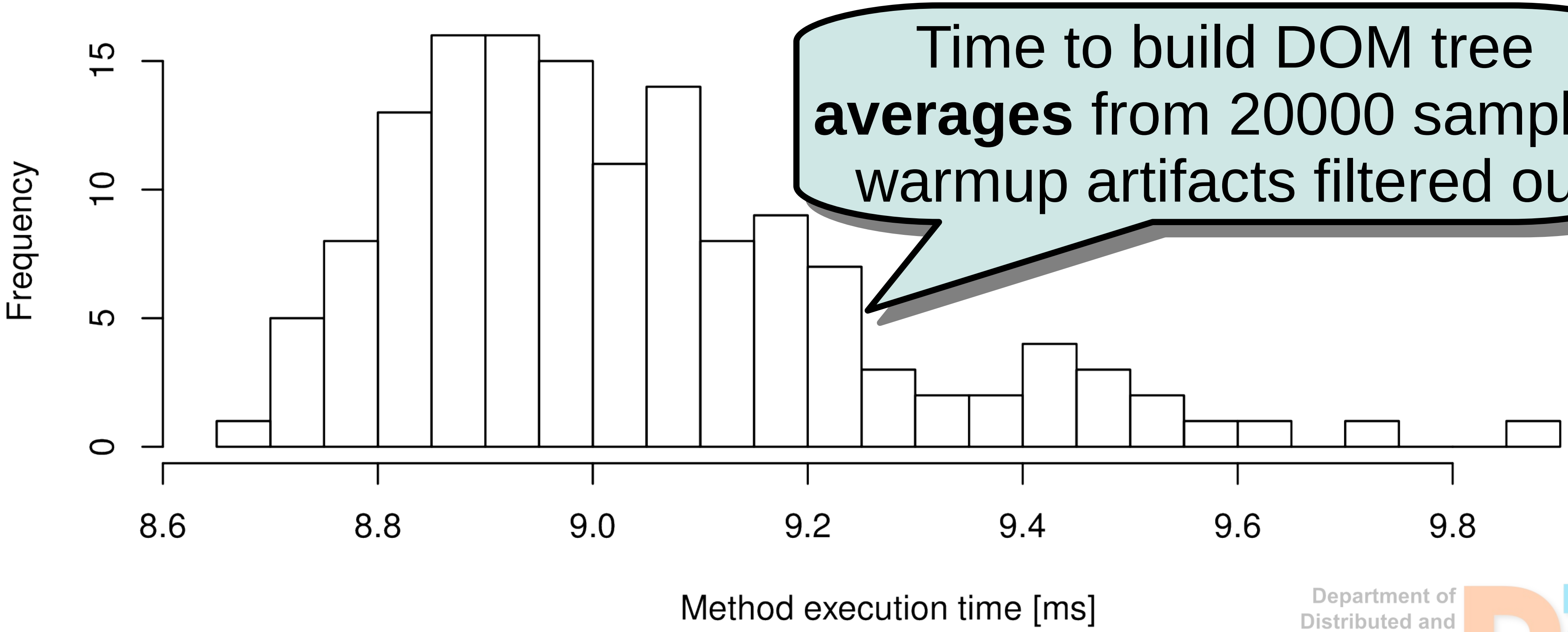
Comparison is hypothesis testing

Performance transformation function

Comparison Sensitivity

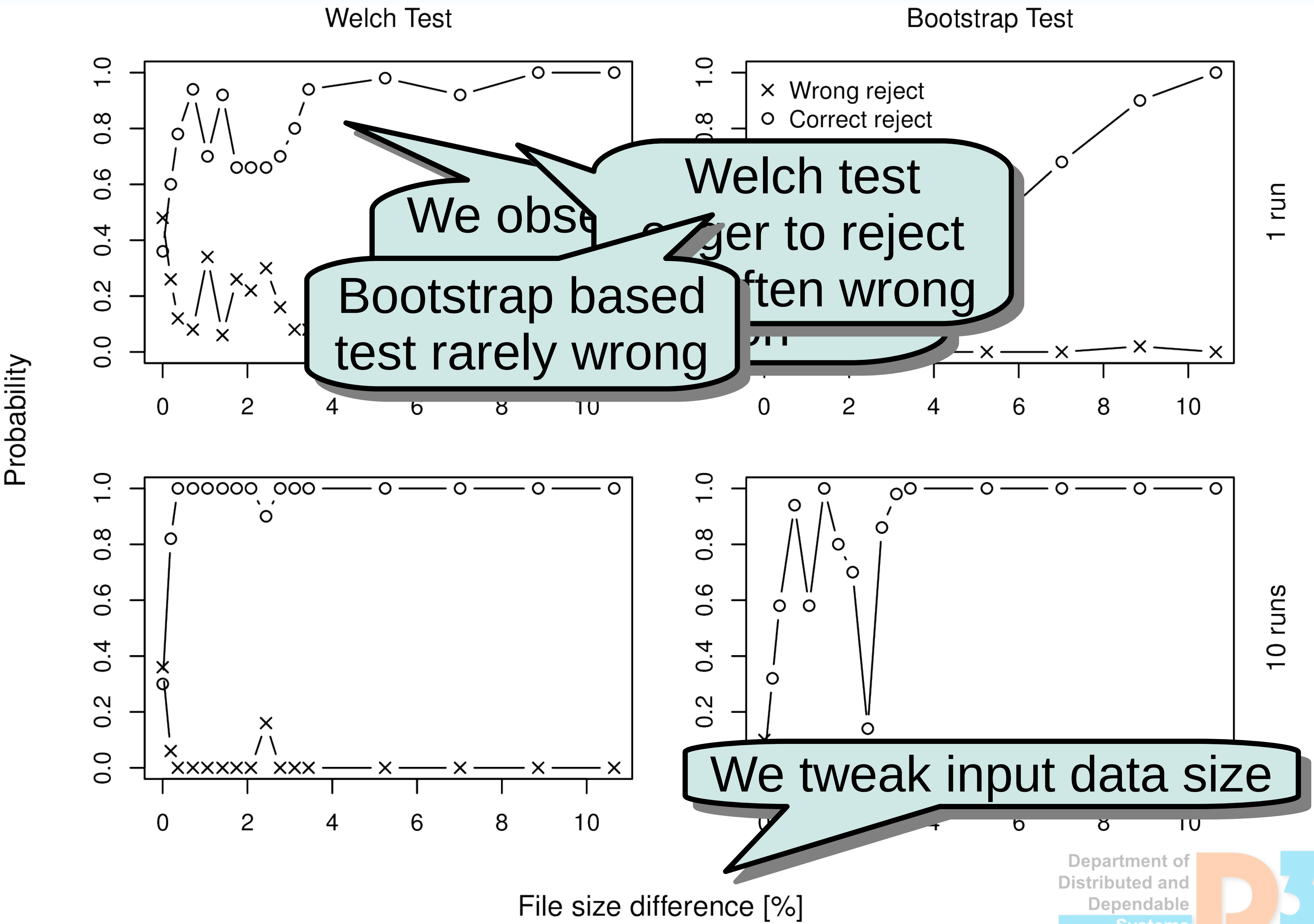
High sensitivity expected

- Sensitivity to changes above **1%** seems **reasonable** to ask
- But changes around **5%** are easily **random fluctuations**



Time to build DOM tree **averages** from 20000 samples warmup artifacts filtered out

Bootstrap Procedure



We obse

Welch test
ger to reject
ften wrong
n

Bootstrap based test rarely wrong

We tweak input data size

@ Runtime: What For ?

Dynamic applications

- More integrated runtime performance adaptation
- Many interesting applications but not discussed here

Performance assertions

- For important conditions difficult to estimate otherwise
- Just having debug dump can be useful

Interactive performance information

- Evaluating developer supplied formula in runtime context
- Developer can fire off queries while programming

@ Runtime: Open Issues

Overhead

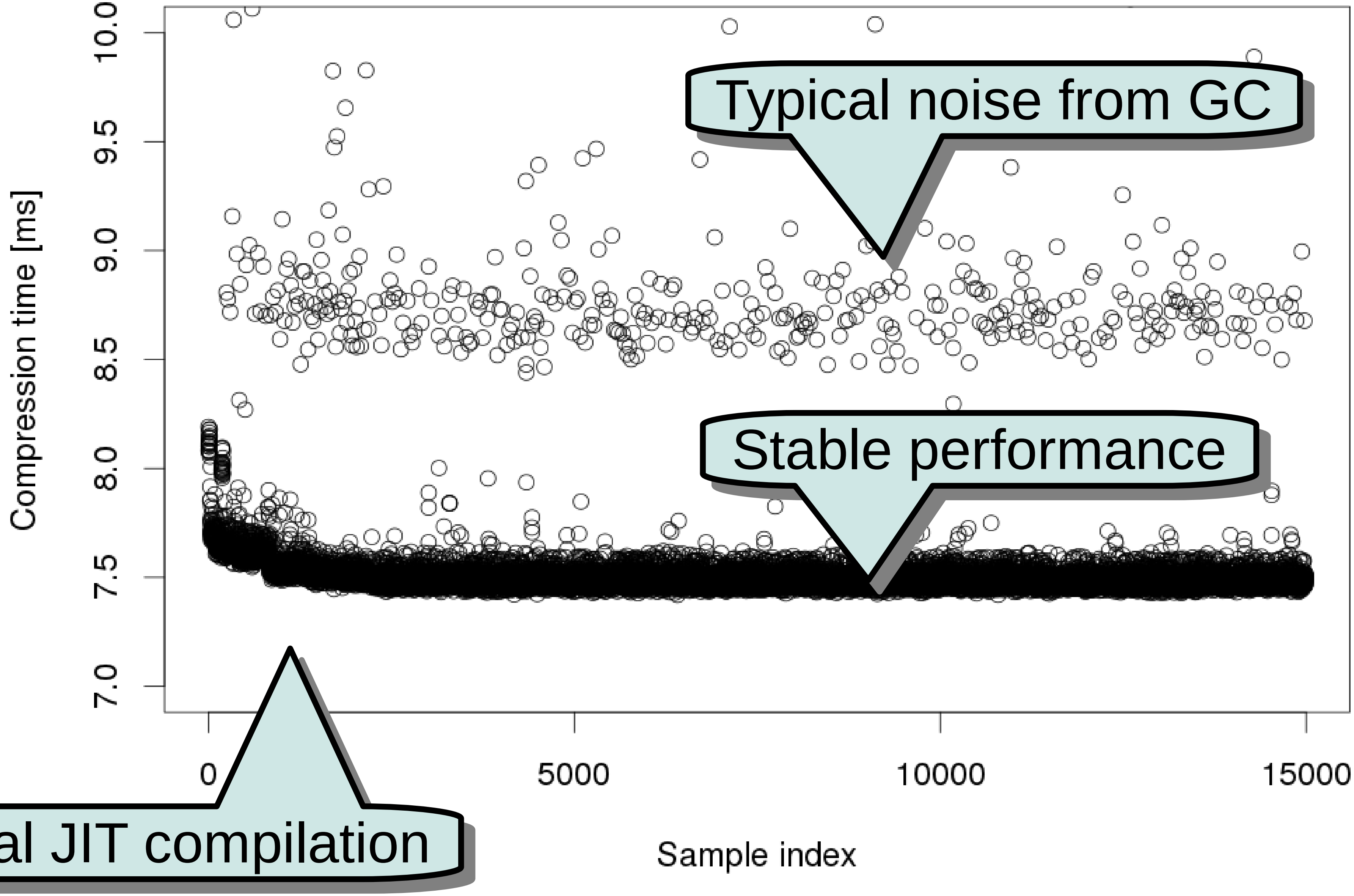
- Measuring everything clearly too expensive
- But what is acceptable ?
  - Pitting **actual** overhead against **visions** of benefits
  - Remember moving from HTTP to HTTPS ?
- Can we predict overhead ?

Location naming

- Source code names terribly static
- Call sites better but not by much
  - Virtual dispatch complications
  - Shallow call sites more useful than deep ones
- No good way to refer to sessions or instances
  - Testing for instance identity creates more overhead
  - Session is not even a language concept

@ Runtime: Prototype Experiments

Baseline compress benchmark from SPEC jvm 2008



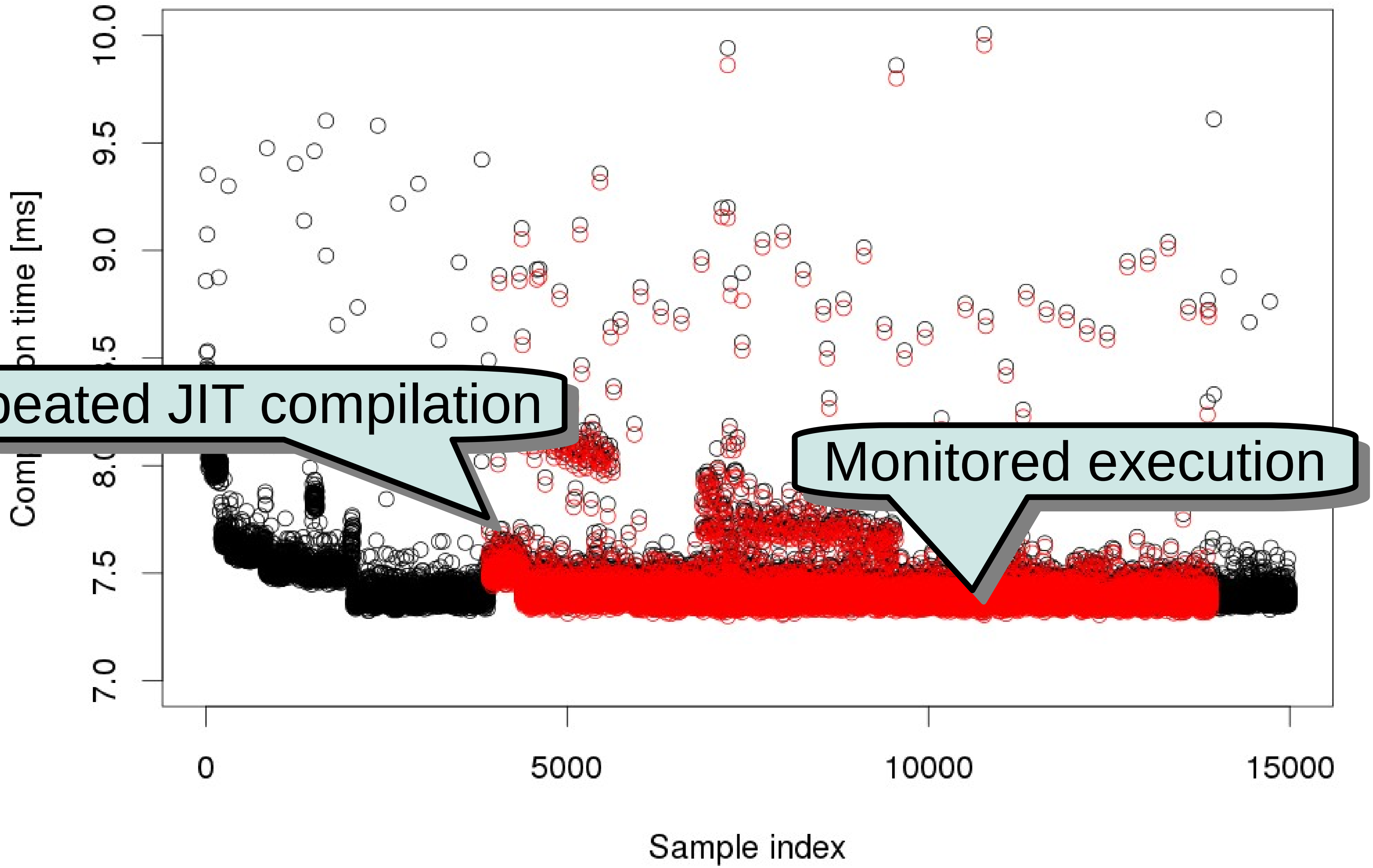
Typical noise from GC

Stable performance

Initial JIT compilation

@ Runtime: Prototype Experiments

Monitoring of Harness.compress ( ) turned on and off

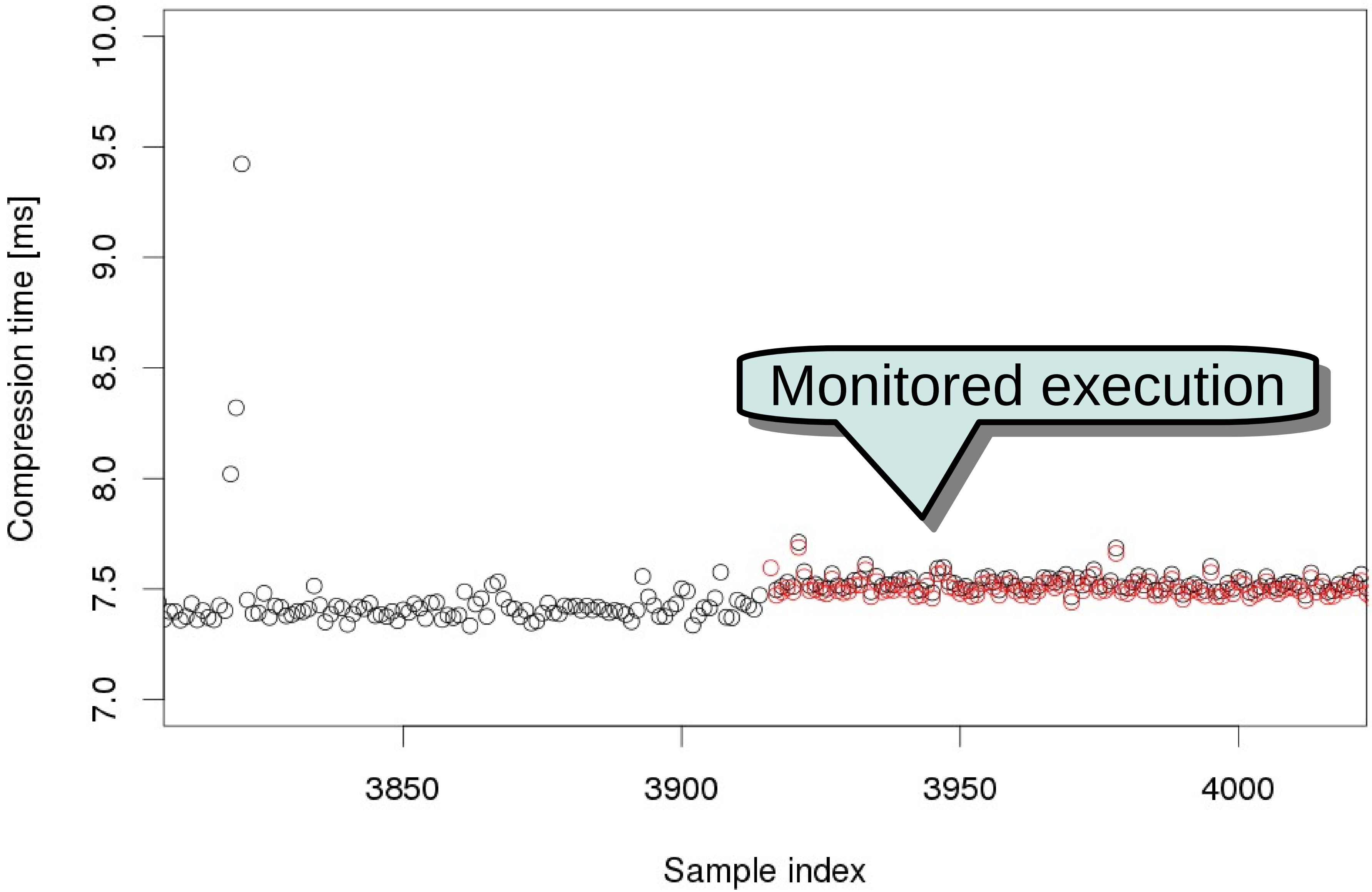


Repeated JIT compilation

Monitored execution

@ Runtime: Prototype Experiments

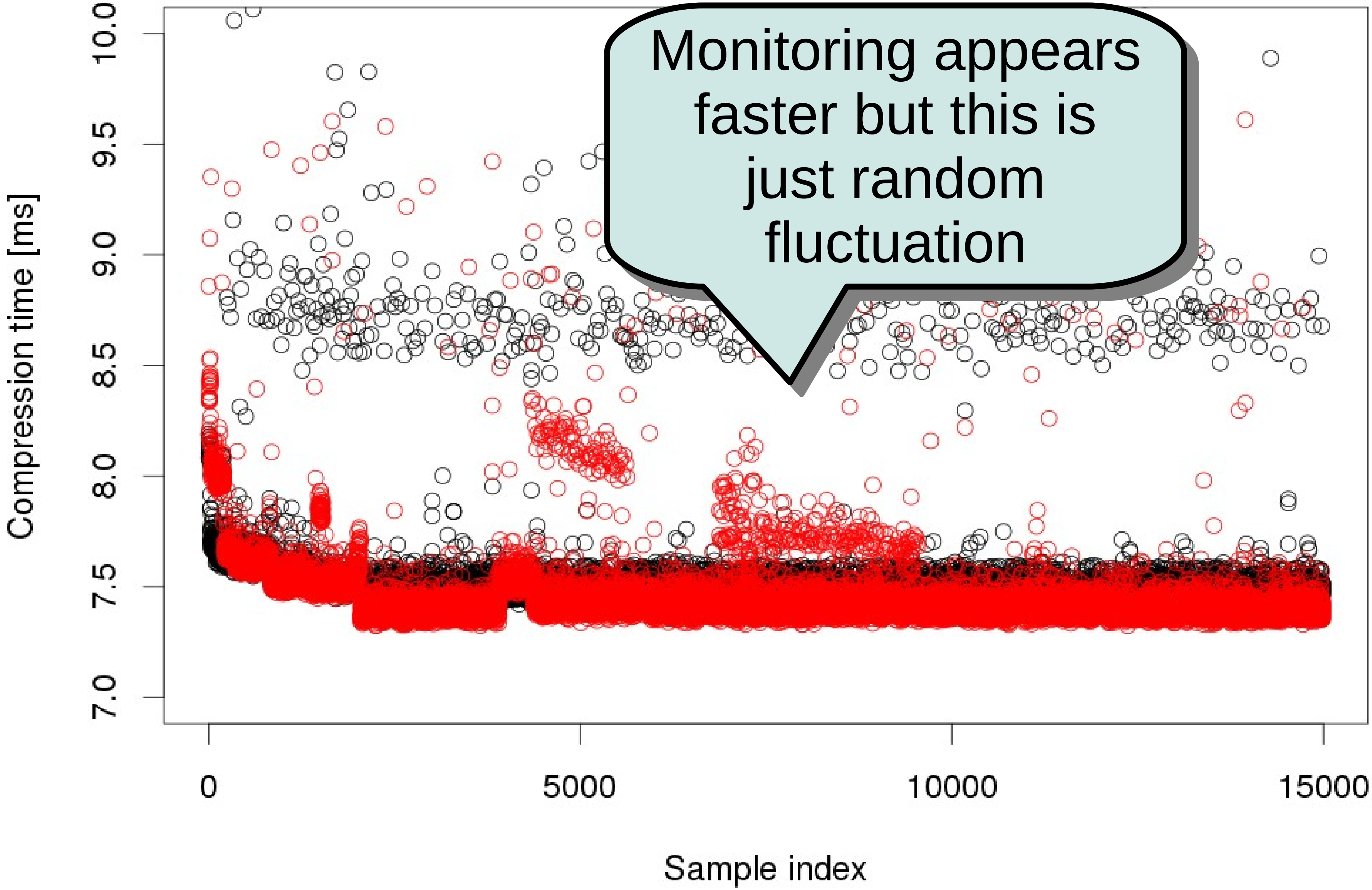
Zooming in to when monitoring is turned on



Monitored execution

@ Runtime: Prototype Experiments

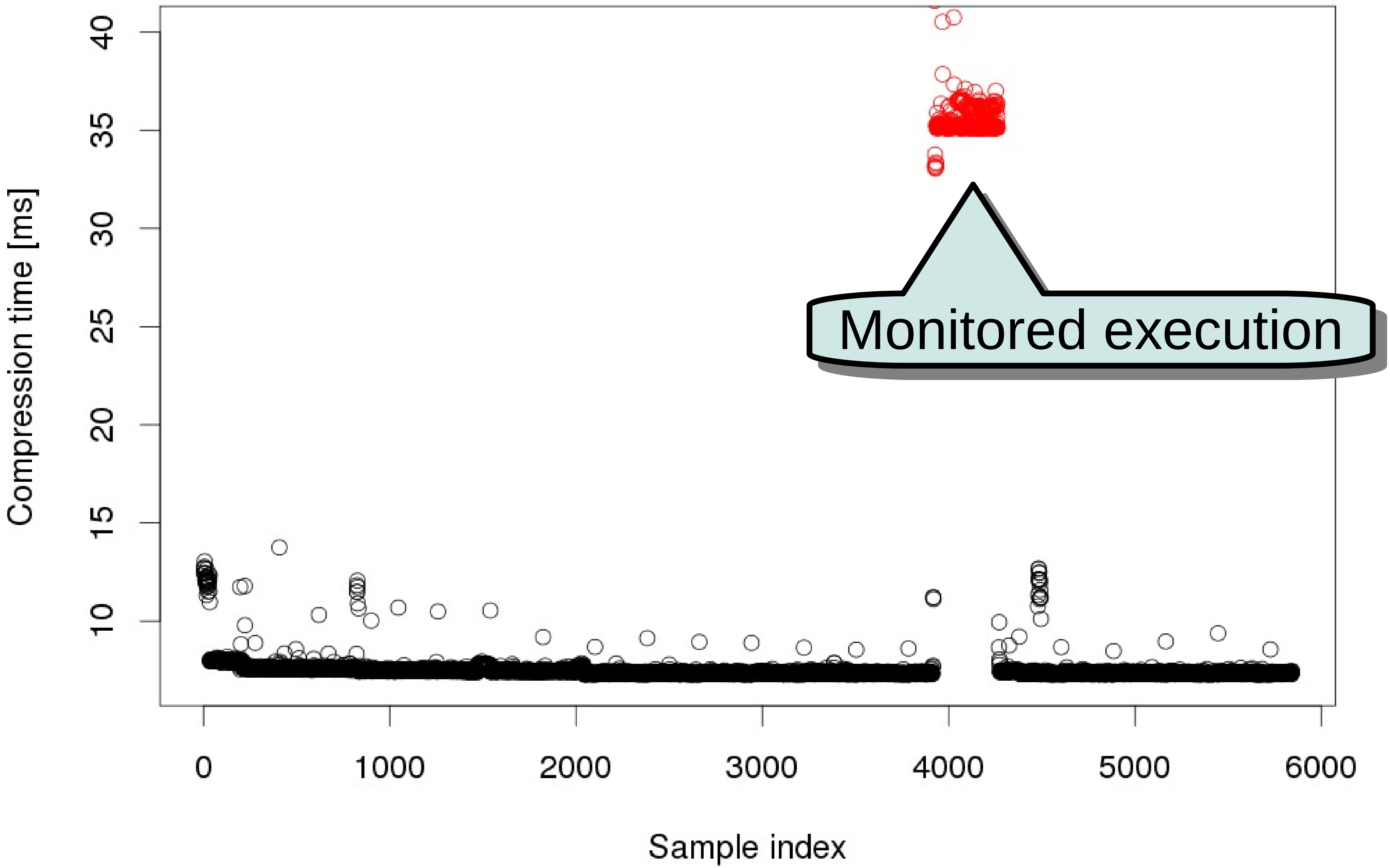
Baseline vs monitoring



Monitoring appears faster but this is just random fluctuation

@ Runtime: Prototype Experiments

Monitoring of HashMap.get ( ) turned on and off



Monitored execution

Performance Unit Testing

if a unit test can test performance we can ***save time*** *in execution and evaluation by focusing on specific issue and collect results related to particular* ***code*** *and* ***author***

Perf Test Wish List

Construction same as functional unit test

- Setup, execution, validation, cleanup
- Robust execution
  - Measurement handled by framework
  - Avoid common implementation mistakes
- Validation against performance specification
  - Also documents contracts and assumptions

Do people
still make them ?

Executed during **commit**

- Automated selection of tests
- Regulated measurement volume

Reasonably **portable**

Implementation Mistakes

```
public static void main (String [] args) {
  LOMap<I,I> map = new LOMap<I,I> ();
  for (int i = 0 ; i < 30000 ; i++)
    map.put(i, i);
  AList<I> toRemove = new AList<I> ();
  for (int i = size ; i < 60000 ; i++)
    toRemove.add(i);

  long start = System.currentTimeMillis ();
  for (Integer cur : toRemove)
    map.remove(cur);
  long stop = System.currentTimeMillis ();
  System.out.println (stop - start);
}
```

Multiple similar tests in Apache Commons JIRA

SPL Unit Testing

Framework for performance unit testing

```
void saxBuilderTest (SPL spl, String file) {
  byte [] data = Files.readAllBytes (file);
  IStream is = new BAIStream (data);
  SAXBuilder sax = new SAXBuilder ();
  Document xml = null;

  while (spl.needsMore ()) {
    is.reset ();
    spl.start ();
    xml = sax.build (is);
    spl.end ();
  }
}
```

Test setup

Measurement loop

Loop setup

Measured code

SPL Unit Testing

Validation separate from test execution

Implementation version

```
m1 := org.jdom.SAXBuilder.build@6a49ef6
m2 := org.jdom.SAXBuilder.build@4e27535
w := saxBuilderTest
```

Workload implementation

```
for f in { "tiny.xml", "big.xml" }
m1 [w](f) >= m2 [w](f)
```

Workload parameters

Test condition

JDOM Case Study

Most **conditions** very **simple**

- "I have now made X faster"
- "I hope I have not made X slower"
- "I have coded X assuming A is faster than B"

Workload rarely available

Some developer **assumptions** were **wrong**

- In our case about 10%
- Not clear whose fault
  - Impossible to reconstruct conditions exactly
  - Platform development terribly fast these days

@ Runtime: What For ?

Getting **real workloads**

- True workloads difficult to predict
  - What is the typical data structure size ?
  - What is the typical concurrency pattern ?
  - How much does this change with context ?
- Specialization offers optimization opportunities
  - Libraries coded assuming general workload
  - Is one-element ArrayList better than one-element TreeList ?

Getting **real background interference**

- Measuring performance in unit tests is like evaluating driving performance without traffic

@ Runtime: Open Issues

Recording real workload

- Basic overhead already discussed
- Recording complete workload not practical
  - Data size issues
  - Privacy issues
- From workload **generator** code to workload **sizing** code
  - Requires extra coding
  - Not always clear what data aspects matter

Understanding real interference

- Too many possible sources: Data locking ? Cache sharing ? Thermal budget ? Disk fragmentation ?
- Not clear what indicators to observe and report

@ Runtime: Open Issues

Evaluating test conditions

Performance likely to change

- With every **restart**
  even when nothing else changes
- With every **deployment**
  because platforms are not exactly equivalent
- With every **code change**
  even when the change appears unrelated

How to distinguish **incidental** and **essential** changes ?

(Not) Handling Complexity

Everybody wants to test complexity

$$\forall s \in (1 \,..\, 1000000):$$

$$\text{Tree.get}\ (s) \leq \log\ (\text{List.get}\ (s))$$

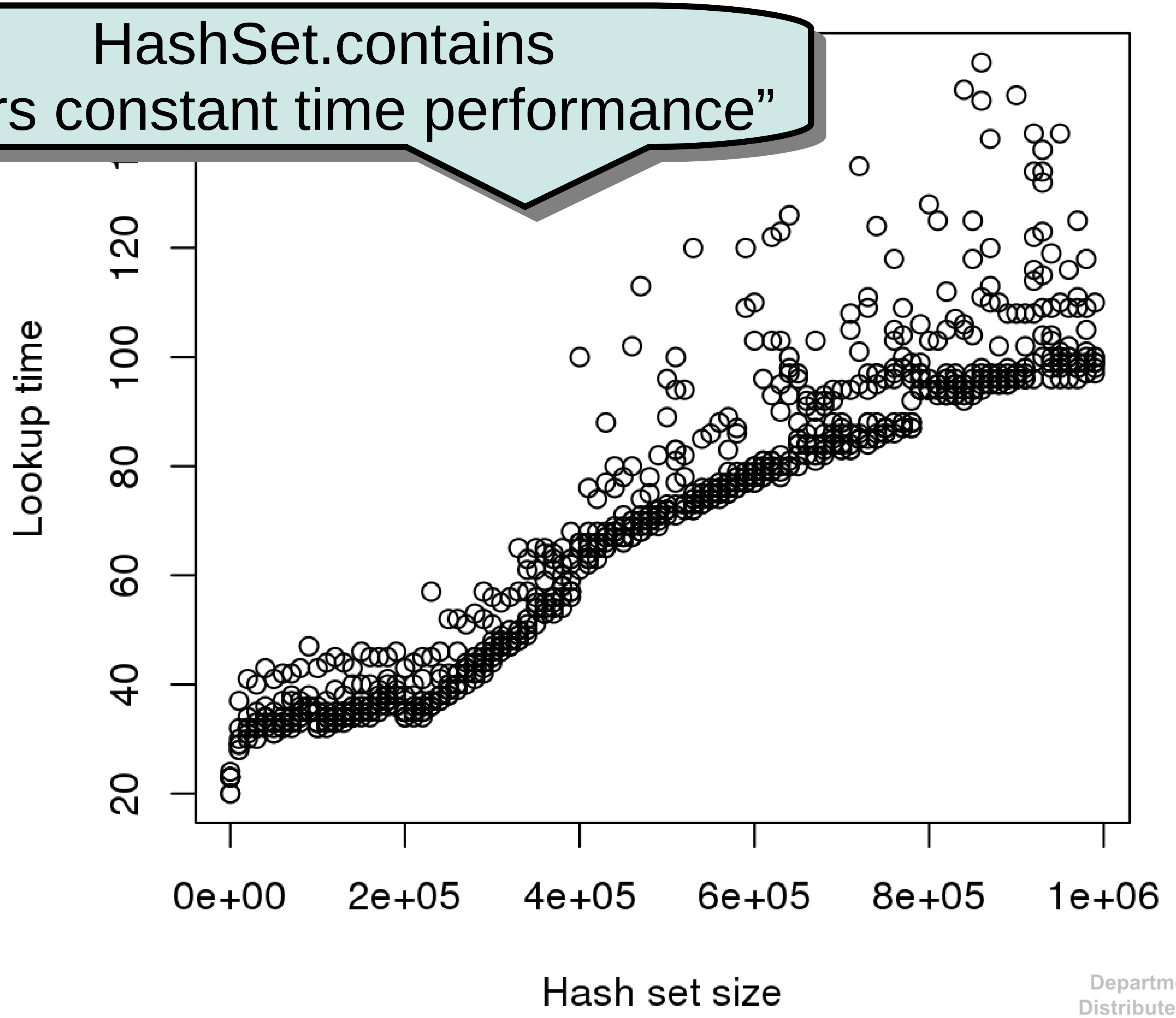
Complexity is useful for **algorithms**

We are dealing with **systems**

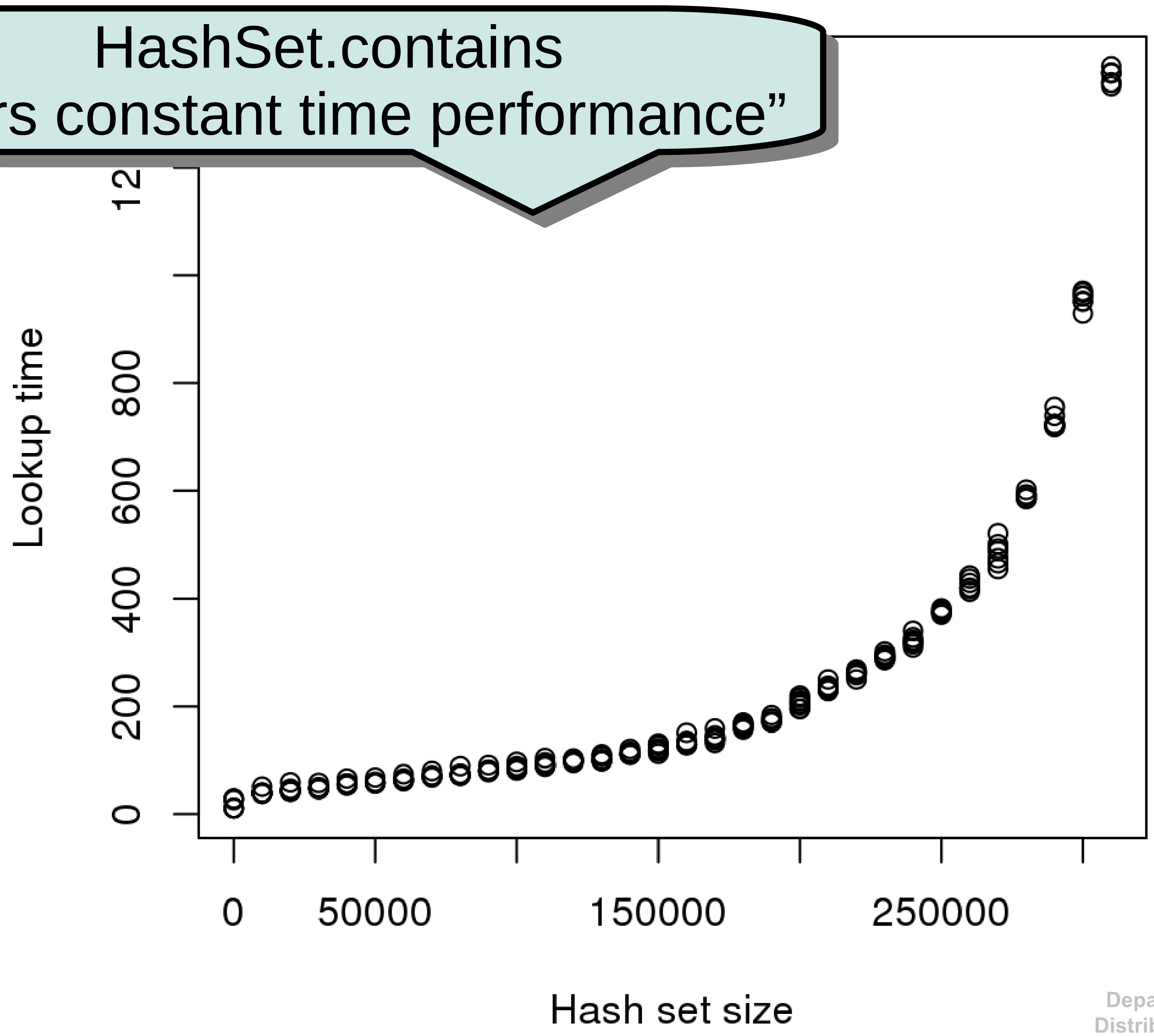
Can we write this for number of servers ?

(Not) Handling Complexity

HashSet.contains
“offers constant time performance”



(Not) Handling Complexity



HashSet.contains
“offers constant time performance”

(Not) Handling Complexity

Change list from Apache Commons Collections 4.0

- 17 issues that explicitly mention “performance”
  - 1 code style change that happened to make things faster
  - 1 optimization to replace inefficient iterator use
  - 1 optimization to introduce boolean shortcut
  - 1 specialized tree merge algorithm
  - **13 fixes** of excessive complexity

```
Collection intersect (Collection one,
                      Collection two) {
  for (Object o : one) {
    if (two.contains (o)) {
      ...
```

How do we fix this ?

Issue tracker mentions **excessive execution time**

Performance Documentation

if program documentation can describe performance we can perhaps ***prevent implementation mistakes***
and provide ***relevant*** *measurements*

Perf Doc Wish List

Generated almost **automatically**

- We have most pieces ready
  - Workload from unit tests
  - Measurements from unit tests
  - Execution infrastructure from unit tests
  - Scaling dimensions from performance specifications
- We need workload description

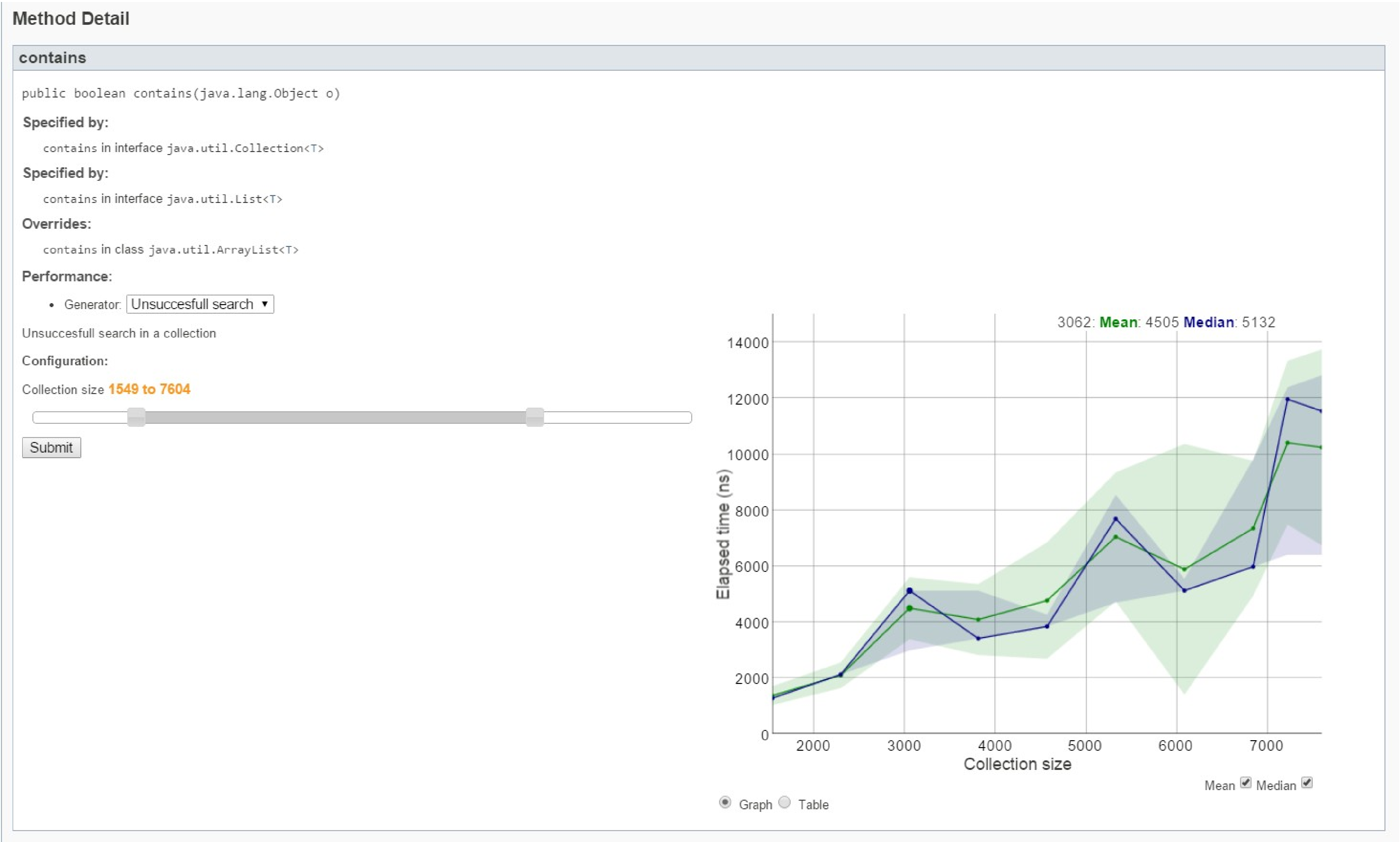
Generated **on demand**

- When particular documentation viewed

Avoid **misleading** information

- Performance in general is not composable
- Performance is not just timing

SPL Documentation



Method Detail

contains

```
public boolean contains(java.lang.Object o)
```

Specified by:

`contains` in interface `java.util.Collection<T>`

Specified by:

`contains` in interface `java.util.List<T>`

Overrides:

`contains` in class `java.util.ArrayList<T>`

Performance:

- Generator: Unsuccesfull search ▾

Unsuccesfull search in a collection

Configuration:

Collection size **1549 to 7604**

@ Runtime: What For ?

Looking at **production performance**

- Developers can see **exact** performance of any code in executing application

Knowing production performance

- Corrects misconceptions about workload
- Provides performance awareness
- Perhaps makes developers think about performance **in the right places**

How much hindsight is in the advice to “avoid premature optimization” ?

@ Runtime: Open Issues

Does it **scale** ?

- Imagine cloud application
  - Are measurements from different instances replaceable ?
  - How much overhead will occasional measurement incur ?

Can we **make enough sense** of real measurements ?

- Observation effects with short times
- Workload characterization missing
- Times include interference
  - Nice to see real behavior
  - No hints on what is going on

How long do measurements stay **valid** ?

Beyond Timing

What about memory usage ?

Memory usage has **multiple aspects**

- Total occupation obviously essential
- Access patterns important for caches
- Temporary allocations related to garbage collection

Most aspects **difficult to observe**

- Total occupation only per process
- Access patterns indirectly through miss rate counters
- Temporary allocations on stack and heap look the same

Mostly at **wrong level of granularity** for developer

- What eactly is memory usage of a **function** ?

Temporary Allocations

Experiment to see if temporary allocations matter

- Workload that **allocates** and **reads** an array
- Independent **array size** and **read count**

Allocated size

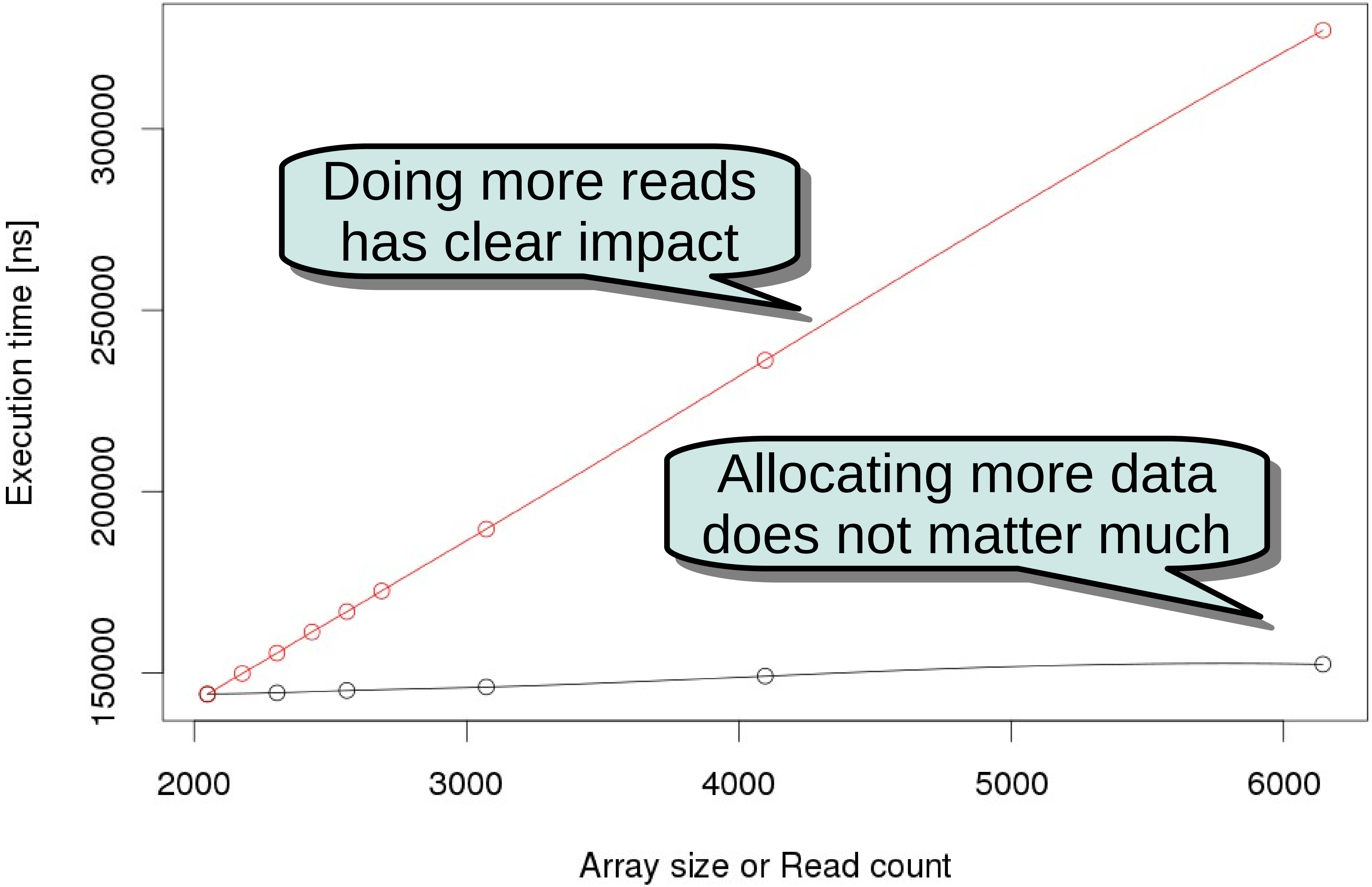
```
public static void work () {
  int [] data = new int [arr_len];
  data [arr_len - 1] = rnd.nextInt ();
  for (i = 0 ; i < wlk_len ; i++)
    data [i] = rnd.nextInt ();

  for (step = 0 ; step < arr_rds ; step++)
    sum += data [rnd.nextInt (wlk_len)];
}
```

Performed work

Temporary Allocations

Measuring performance in isolation

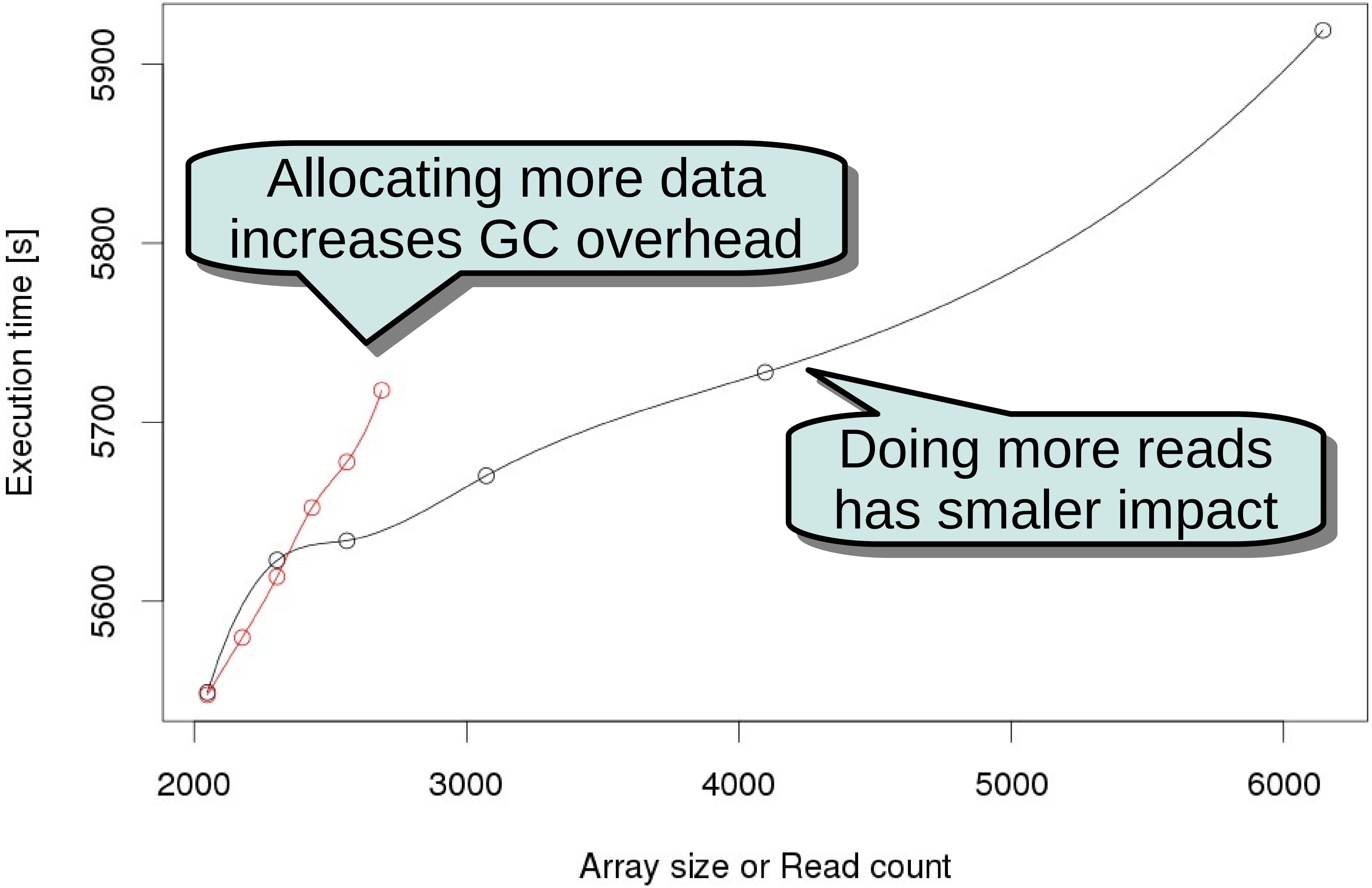


Doing more reads has clear impact

Allocating more data does not matter much

Temporary Allocations

Measuring performance in larger application



Allocating more data increases GC overhead

Doing more reads has smaler impact

To Summarize

Many potential benefits ...

- Continuous feedback on performance
- Validating performance assumptions
- Measurements with real interference
- Programming for real workloads

... and many challenges !

- Managing overhead and stability
- Navigation in runtime structures
- Understanding measurements
- Appropriate granularity

Thank You

SIXTH FRAMEWORK PROGRAMME

SEVENTH FRAMEWORK PROGRAMME

GAČR CZECH SCIENCE FOUNDATION

Much of this talk originated from the long-time work of my research colleagues, which I gratefully acknowledge. The errors, alas, are mine.

Department of Distributed and Dependable Systems D3S