The Impact of Web Service Integration on Grid Performance

François Taïani
Lancaster University

Matti Hiltunen & Rick Schlichting
AT&T Labs - Research

HPDC-14, *The 14th IEEE International Symposium on High Performance Distributed Computing*, Research Triangle Park, NC, USA, July 24-27, 2005

Context & Motivation

- New Globus version (3.9.x / 4.0.x): convergence
  - ➔ **Grid Computing:** federating resources (OGSA)
  - ➔ **Web Services:** integrating services (WSRF)
- Web Services and their associated technologies (SOAP, XML, WSDL) are reputed **inefficient**
  - ➔ What is the **performance impact** on Globus?
- Globus has grown into a **large, complex, collaborative** middleware (IBM, Apache,...)
  - ➔ How to **extract meaningful** profiling data?

- How to **profile** a **complex** piece of software?
- What does it tell us about **Globus**?

Chosen Approach

- 2 steps:
  1. **Black box profiling:** minimal interferences. Coarse results.
  2. **Sample based profiling:** less accurate but more detailed.
- We focused on the **connectivity** of the **WSRF** implementation of GT4-Java:
  - ➔ Low level "**plumbing**". No high level service involved
  - ➔ Motivation: profile the **founding bricks** of the Globus platform
- Experimental set-up:
  - ➔ **Standalone SMP** server running 4 Intel Xeon @ 1.6GHz
  - ➔ **No network cost** involved!
  - ➔ **Avoids context switching** overhead!
  - ➔ Globus **3.9.4** used (last GT4 alpha release, released Dec.04)

Outline

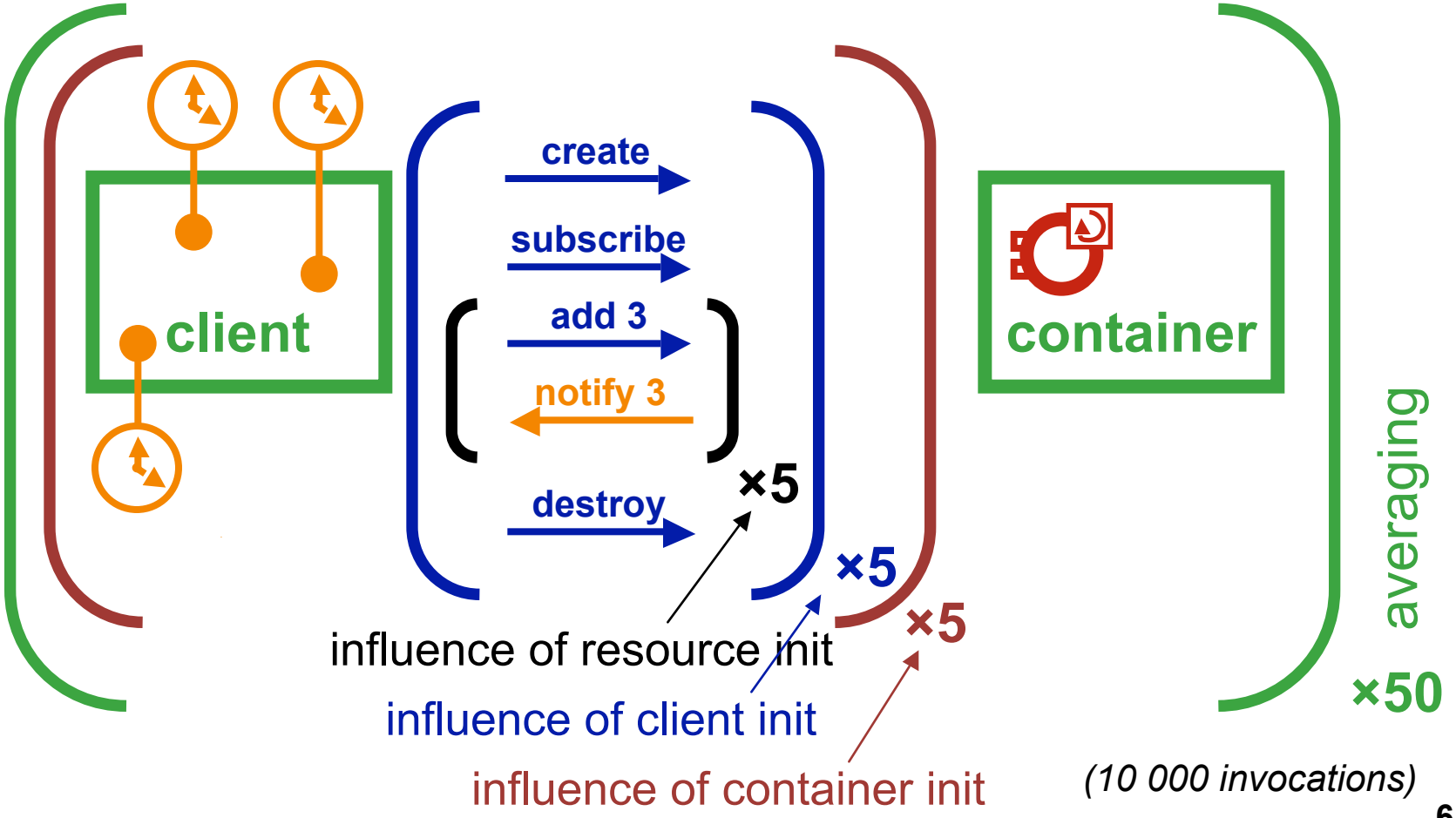
- Introduction: Motivation and Approach
- Black Box Profiling: Set-Up and Results
- Sample Based Profiling: Approach and Results
- Conclusion

Outline

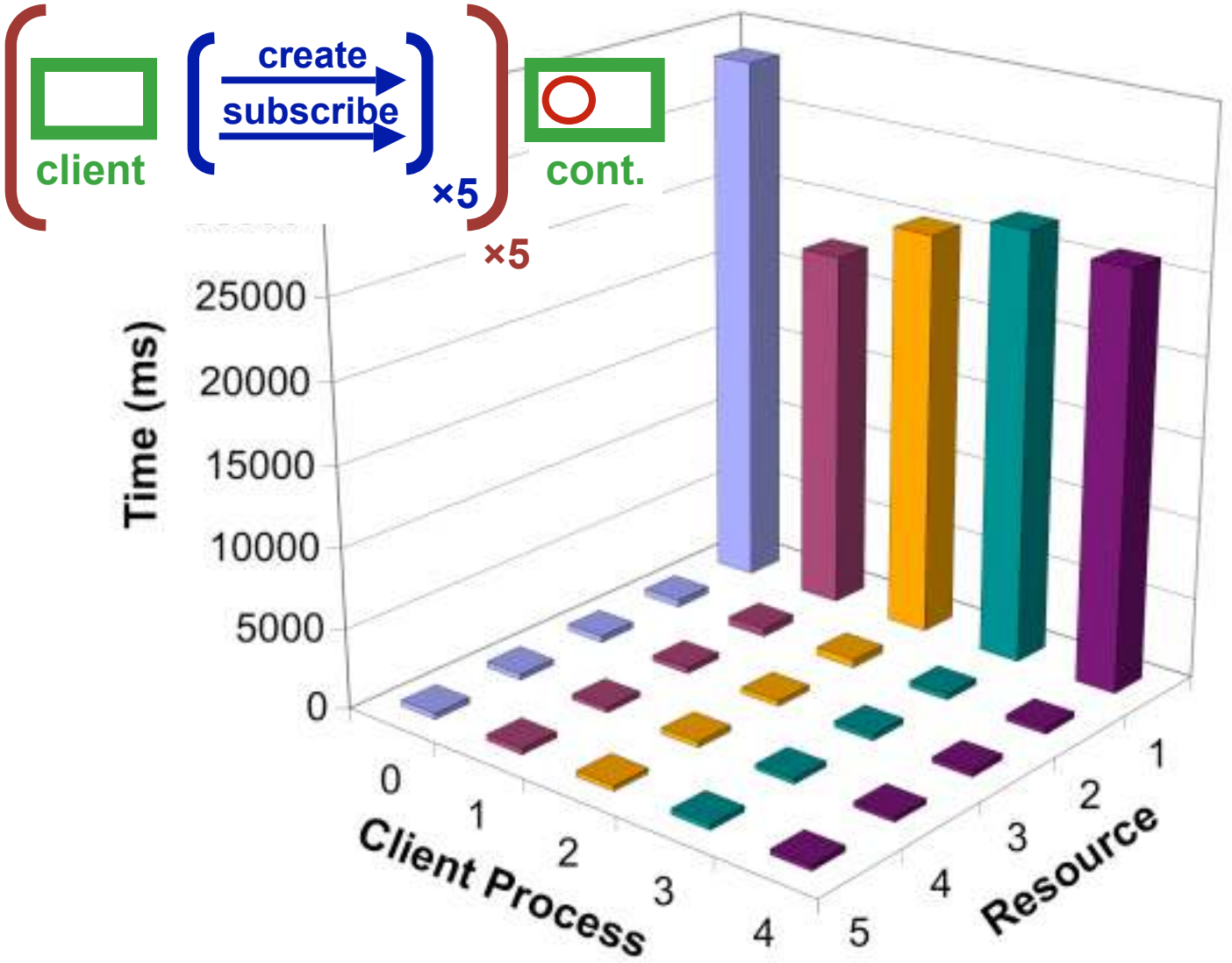
- Introduction: Motivation and Approach
- Black Box Profiling: Set-Up and Results
- Sample Based Profiling: Approach and Results
- Conclusion

Black-Box Profiling: Approach

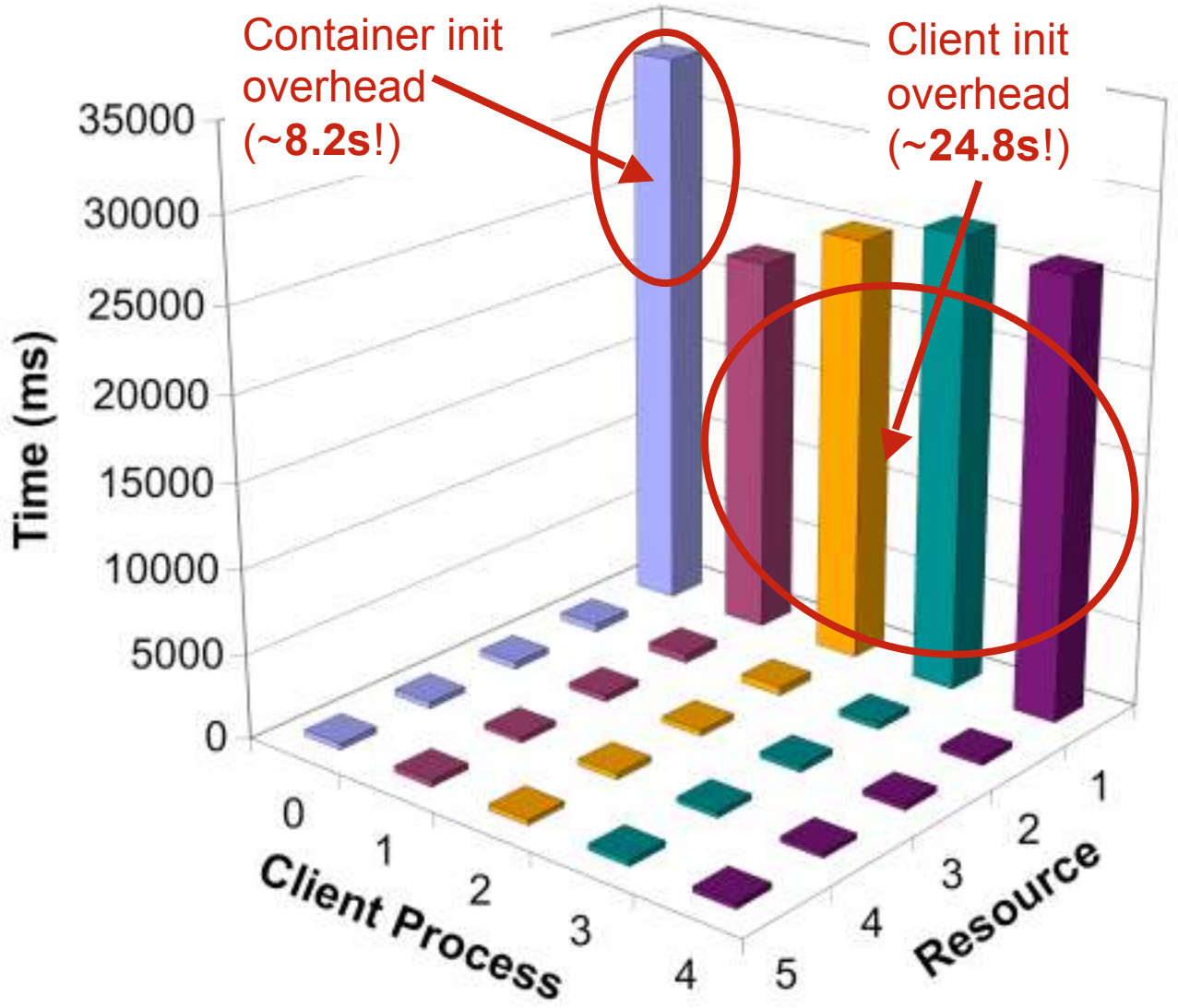
- Black Box Approach: Measure **externally** visible latencies
  - ➔ **Many** different situations to be considered!



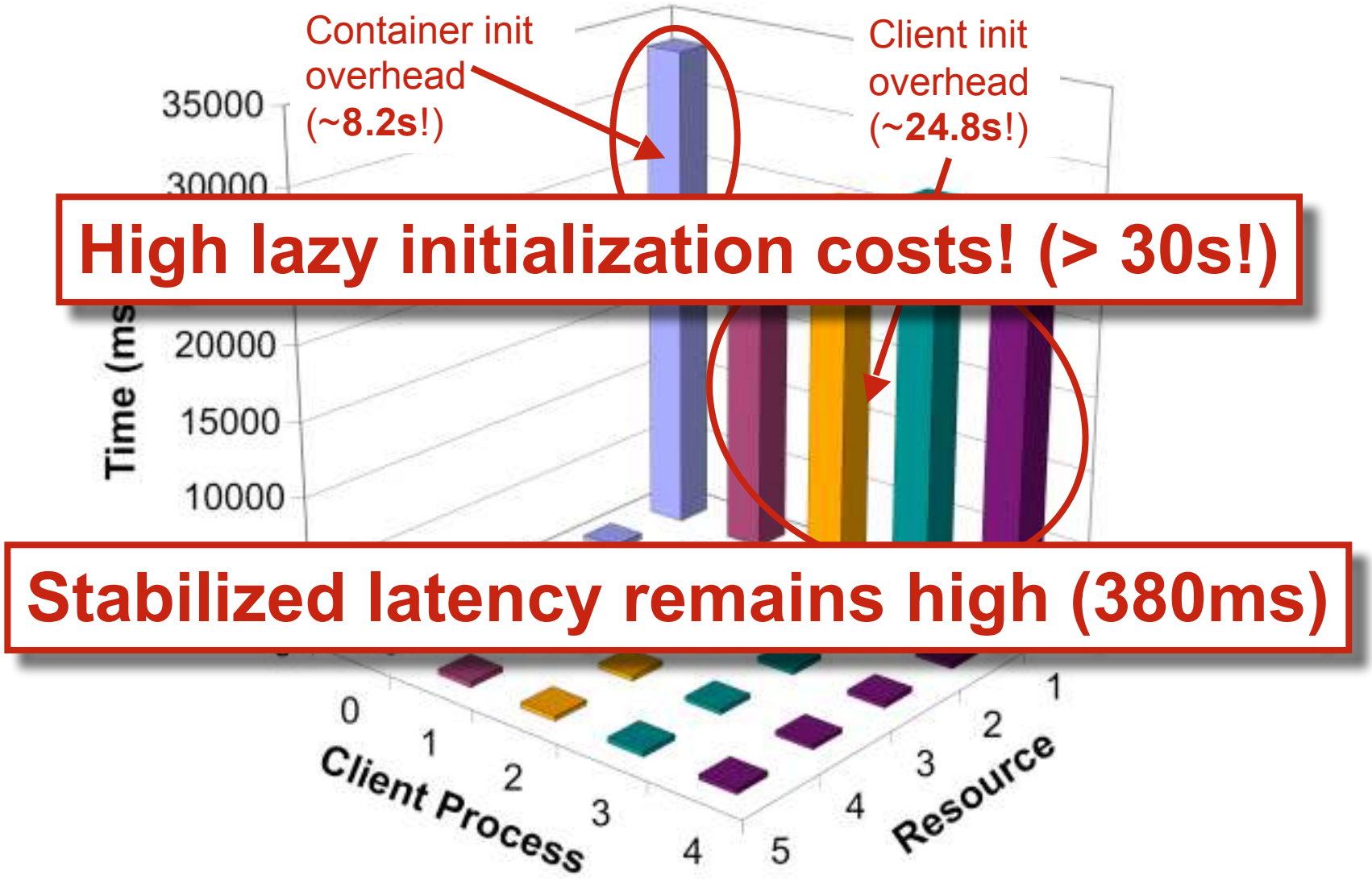
Resource Set-Up



Resource Set-Up



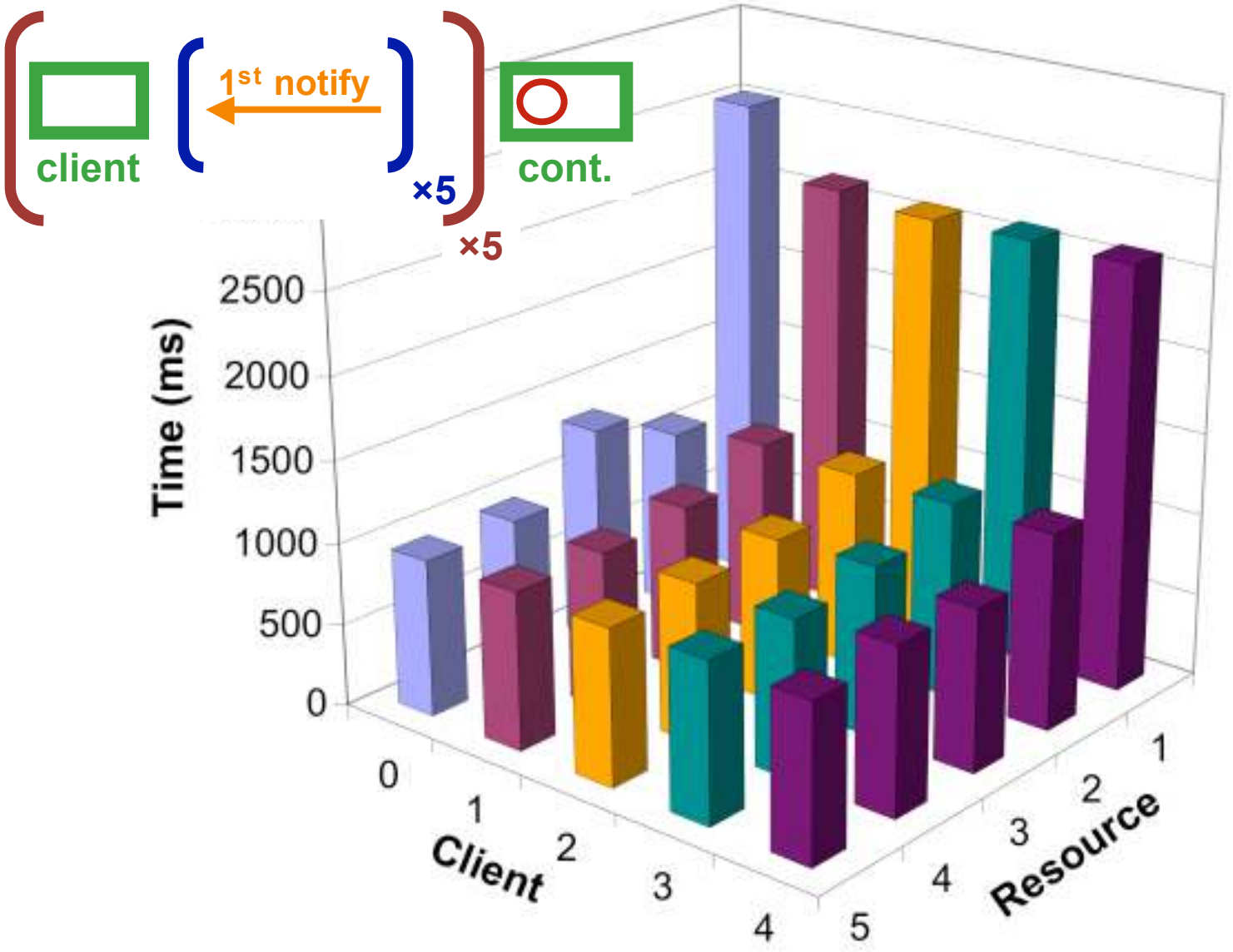
Resource Set-Up



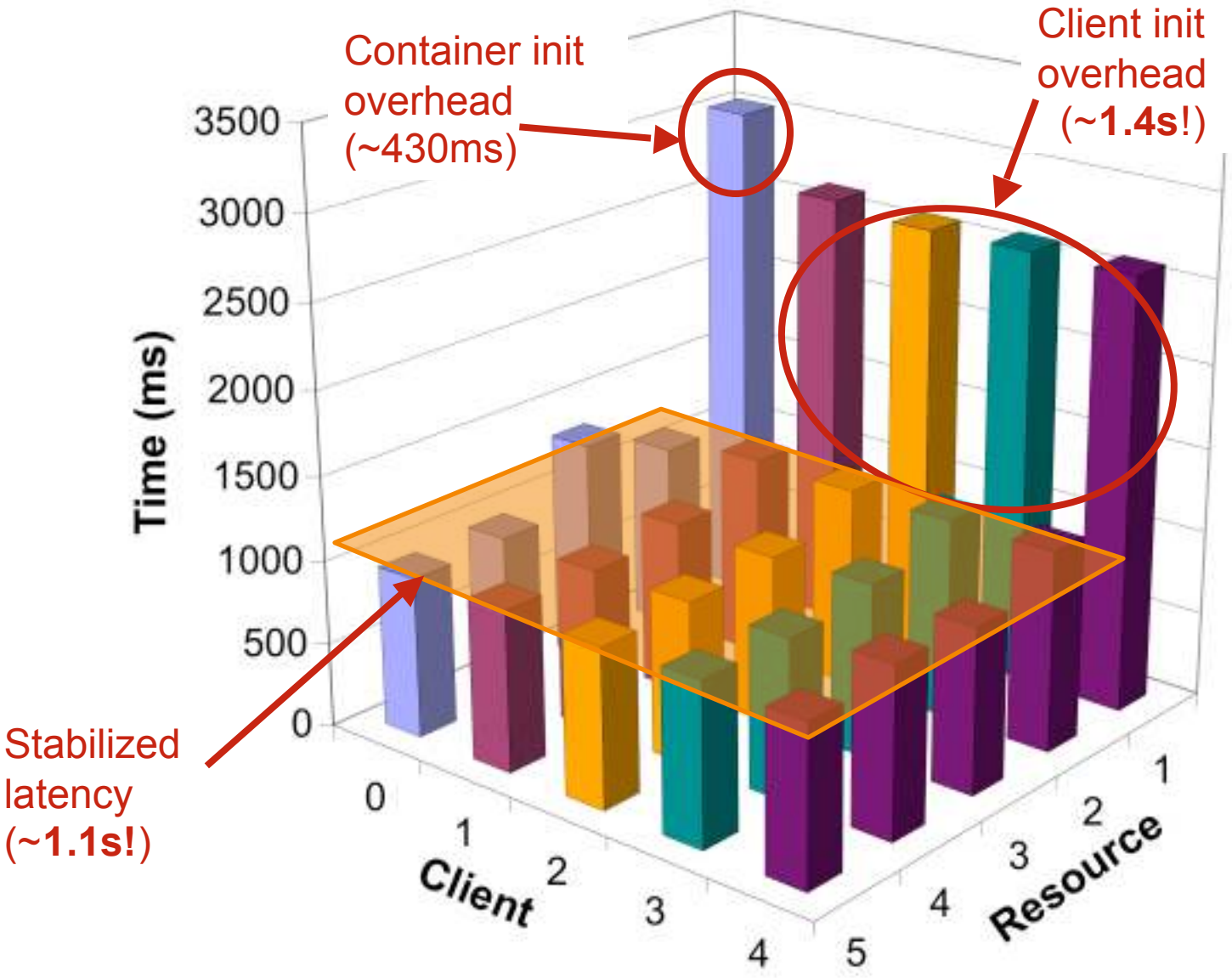
High lazy initialization costs! (> 30s!)

Stabilized latency remains high (380ms)

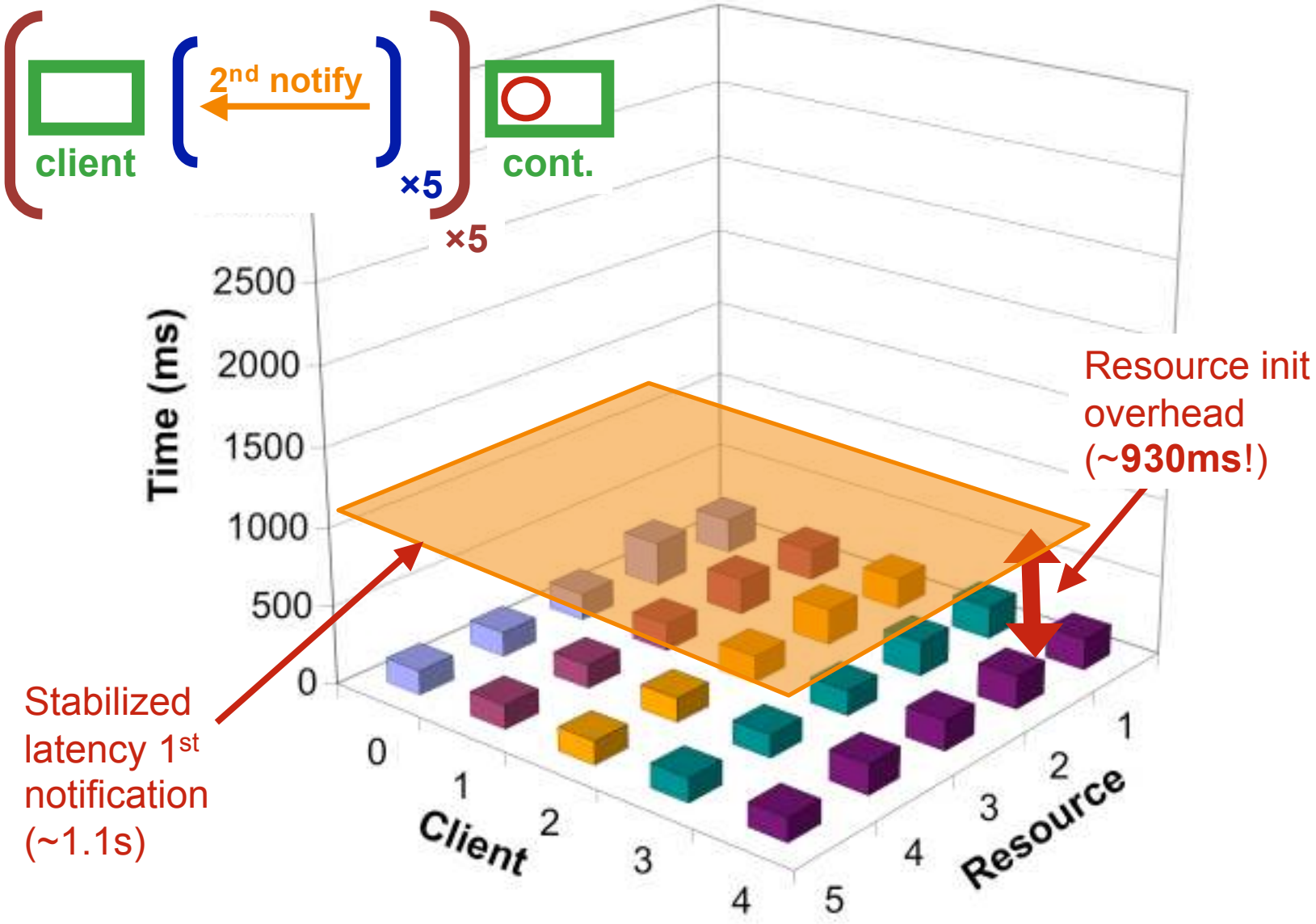
First Notification



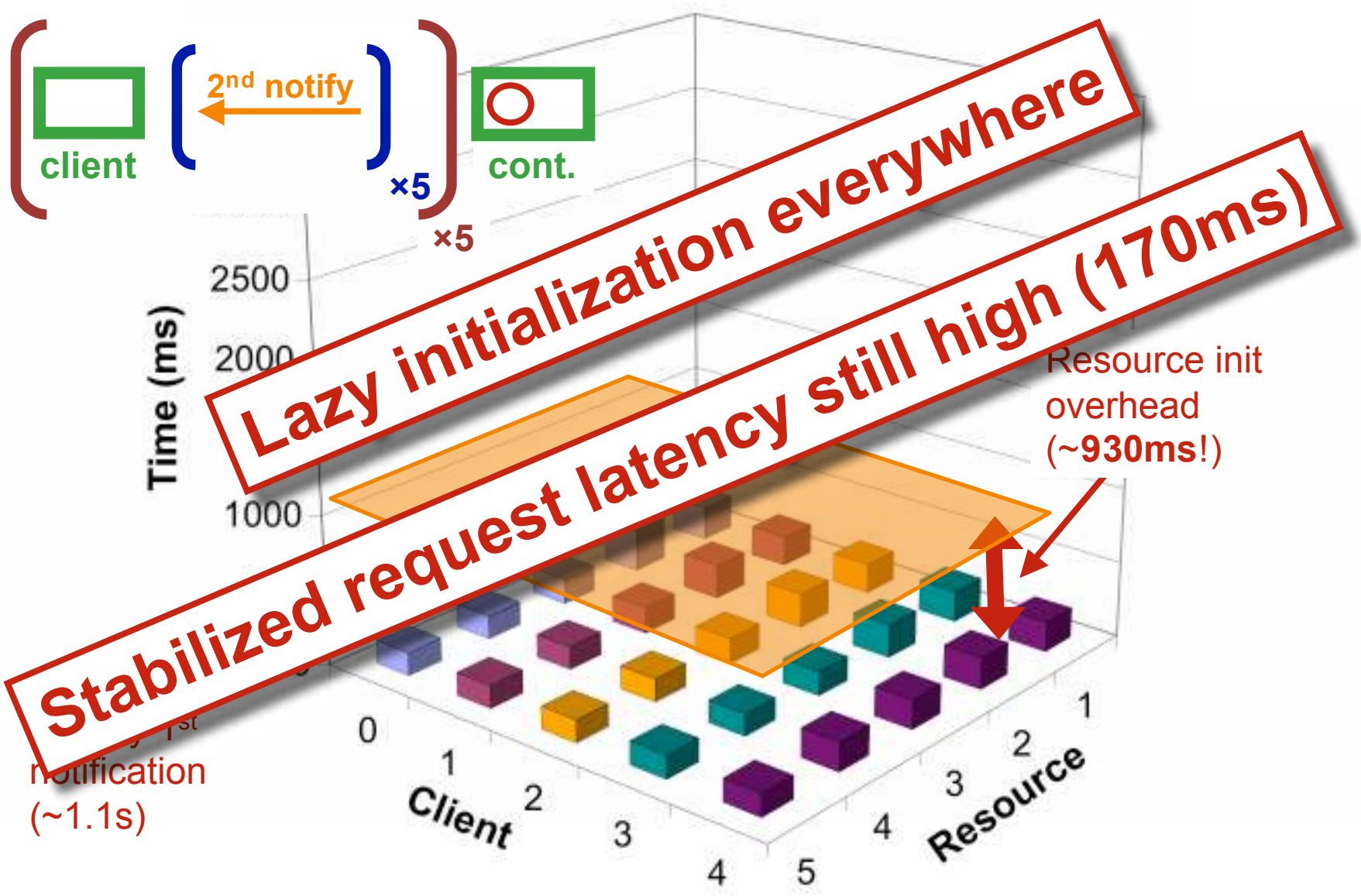
First Notification



Second Notification



Second Notification



Outline

- Introduction: Motivation and Approach
- Black Box Profiling: Set-Up and Results
- Sample Based Profiling: Approach and Results
- Conclusion

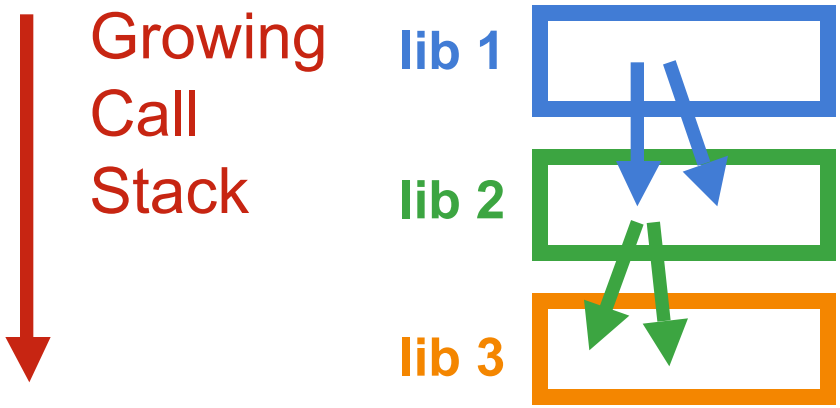
Sample Based Profiling: Introduction

- **Goal:** *relate* observed latencies to Globus internal structure
- **Profiling** data obtained through **sampling**
  (SUN hprof basic profiler)
  - ➔ JVM periodically **stopped**. **Stack** of active thread is **captured**.
  - ➔ Result : A set of **weighted stack traces**. Weight = measures how often the stack was observed.
- **Visualization**:
  Set of weight stacks = **multi-dimensional object**
  - ➔ *Time* (represented by weights)
  - ➔ *Threads*: each trace belongs to a thread
  - ➔ *Control flow* (represented by stacks, reflects use relationships)
  - ➔ *Code Structure* (package organization, class hierarchy, *etc.*)

Program visualization

- Problem studied for quite a long time now.
- **Projection** (aggregation / collapsing) required
- *Many* possibility.
  - ➔ Our goal: related profiling to software structure
  - ➔ Our choice: **package aggregation + stack depth**

```
lib1.Wale   .breath
lib1.Mammal.inhale
lib2.Lung   .inhale
lib2.Muscle.contract
lib2.Nerve  .transmit
lib3.Signal.travel
```



- Tracing calls reveals the software structure.

Sample Based Profiling: Example

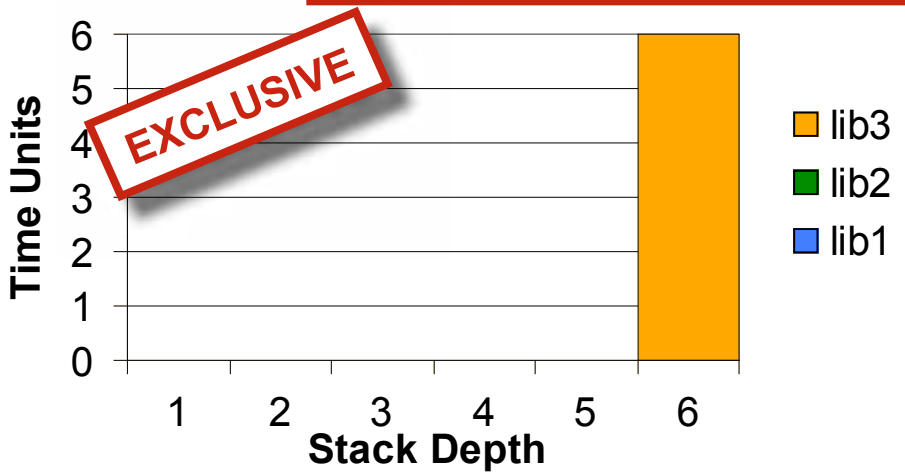
```
lib1.Wale   .breath
lib1.Mammal.inhale
lib2.Lung   .inhale
lib2.Muscle.contract
lib2.Nerve  .transmi
lib3.Signal.trave
```
×3

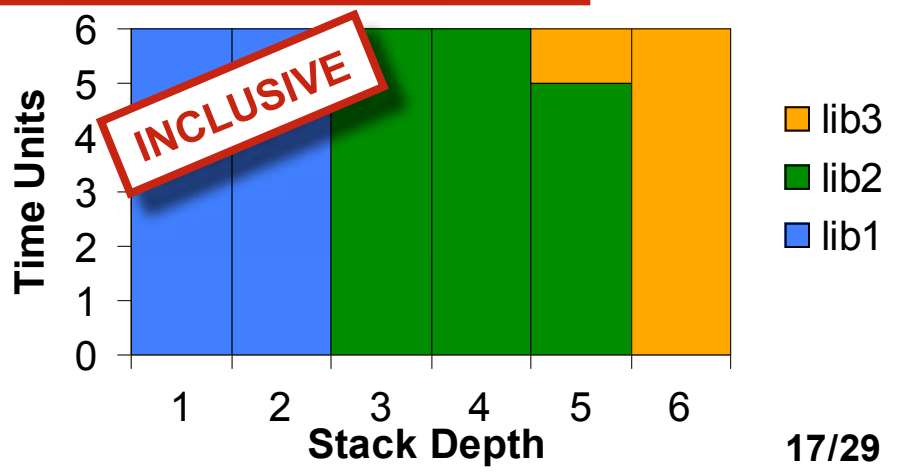
```
lib3.Blood  .flow
lib3.Pressure.foo
```
×1

```
lib2.Muscle.stop
lib2.Nerve  .transmit
lib3.Signal.travel
```
×2

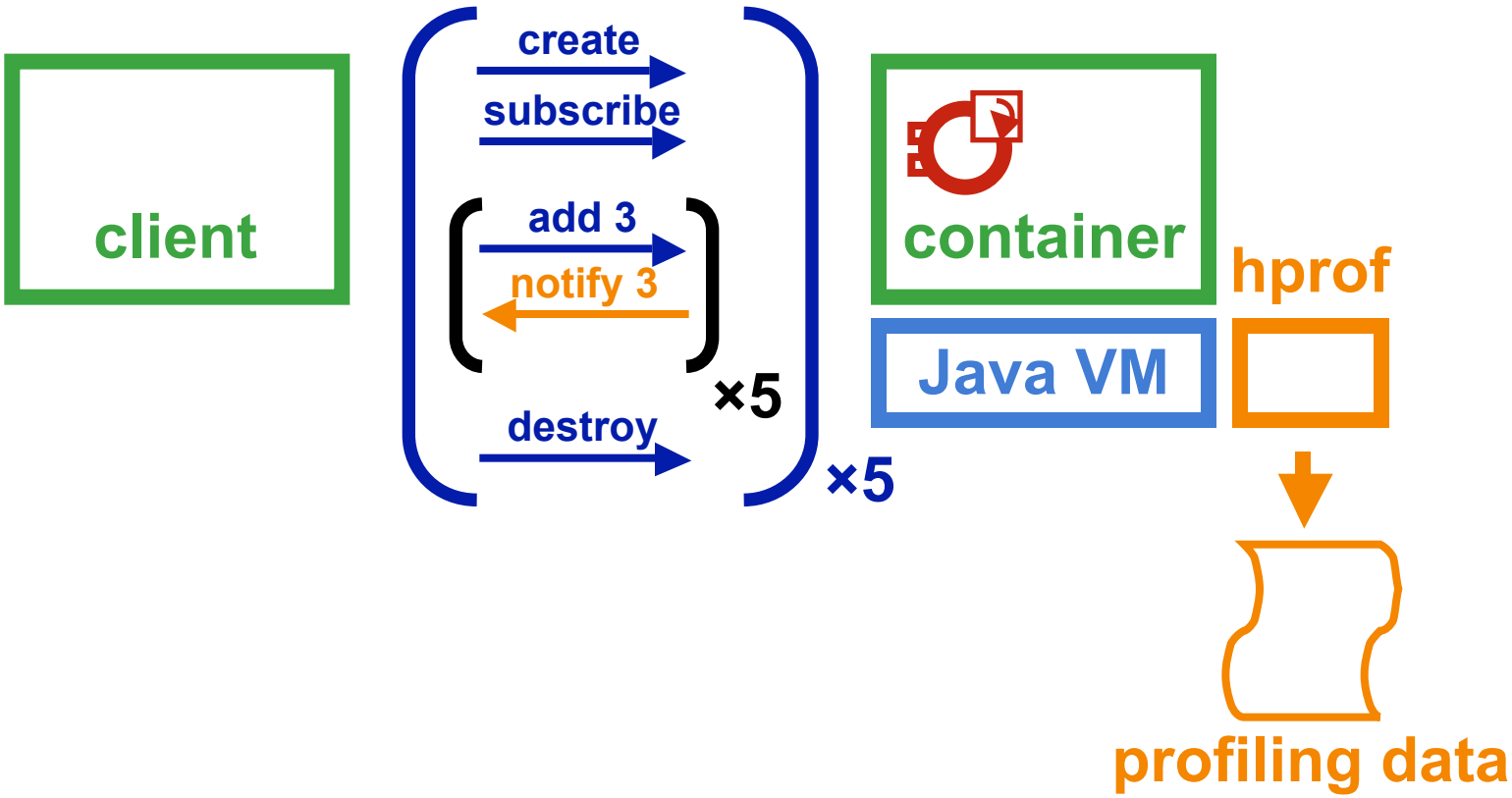
Sampling yields a set of weighted *stack traces (weight reflects* time *spent)*

- Aggregates invocations of the same library.
- Chart w.r.t. position in call stack.

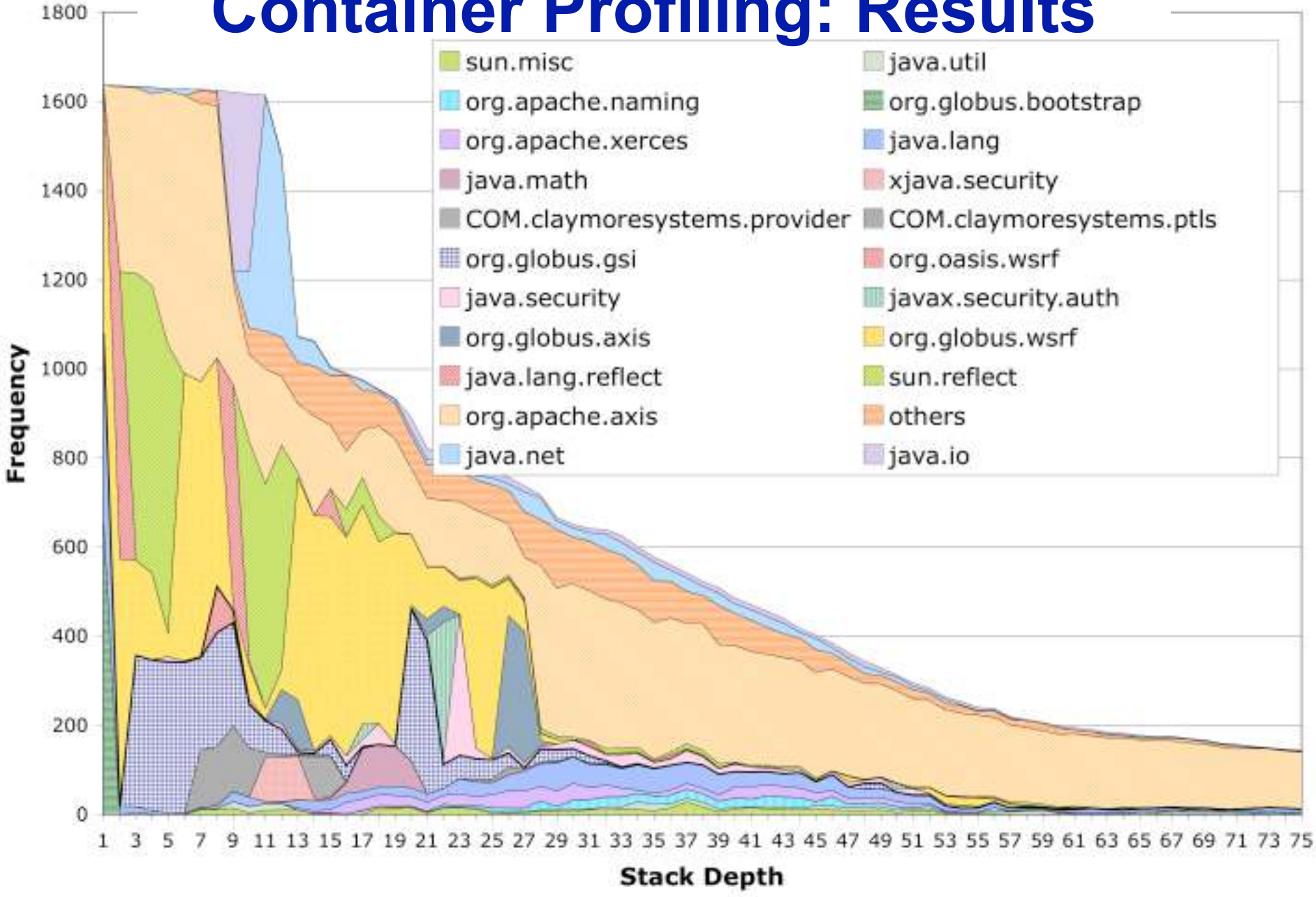




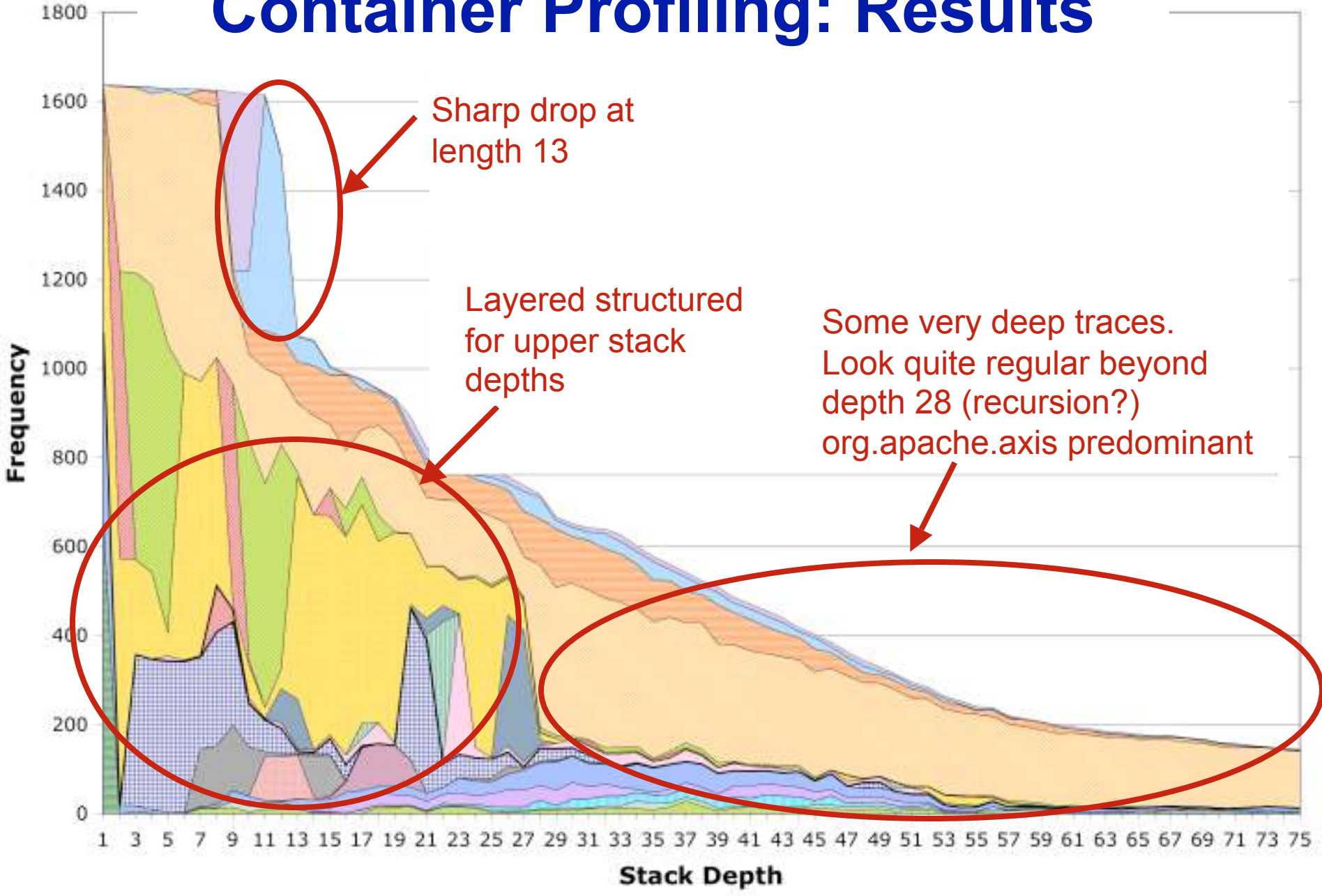
Experimental Set-Up



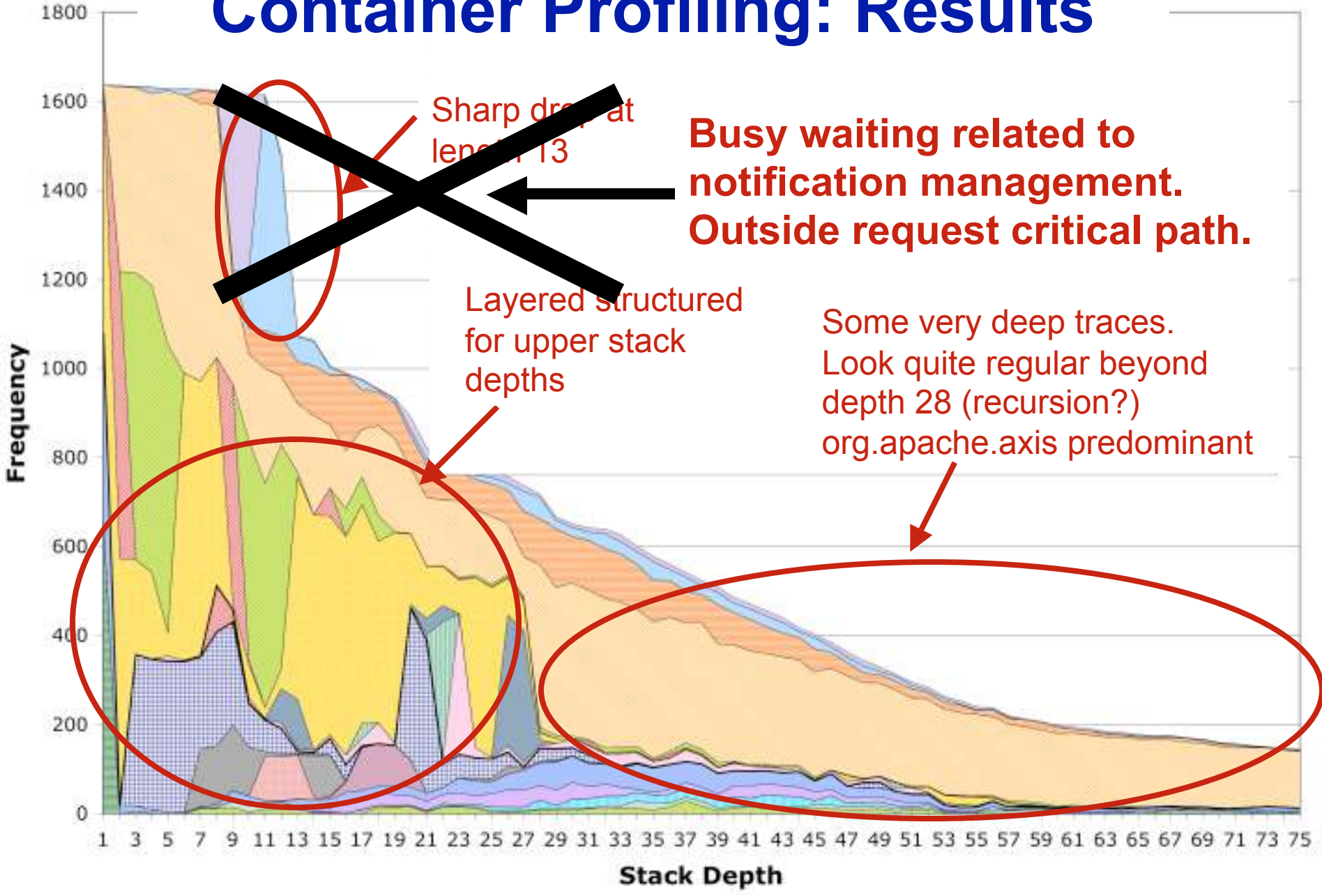
Container Profiling: Results



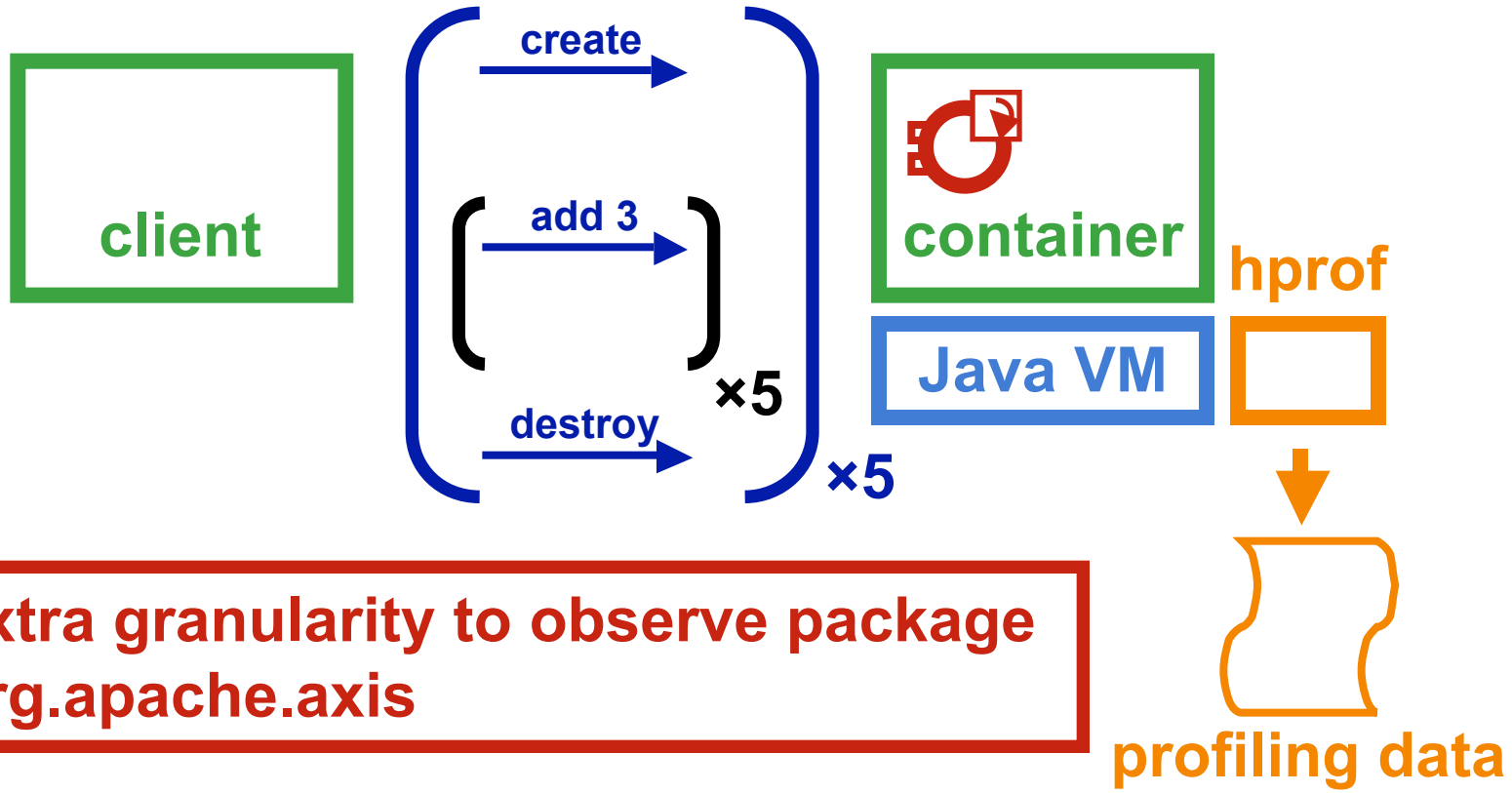
Container Profiling: Results



Container Profiling: Results

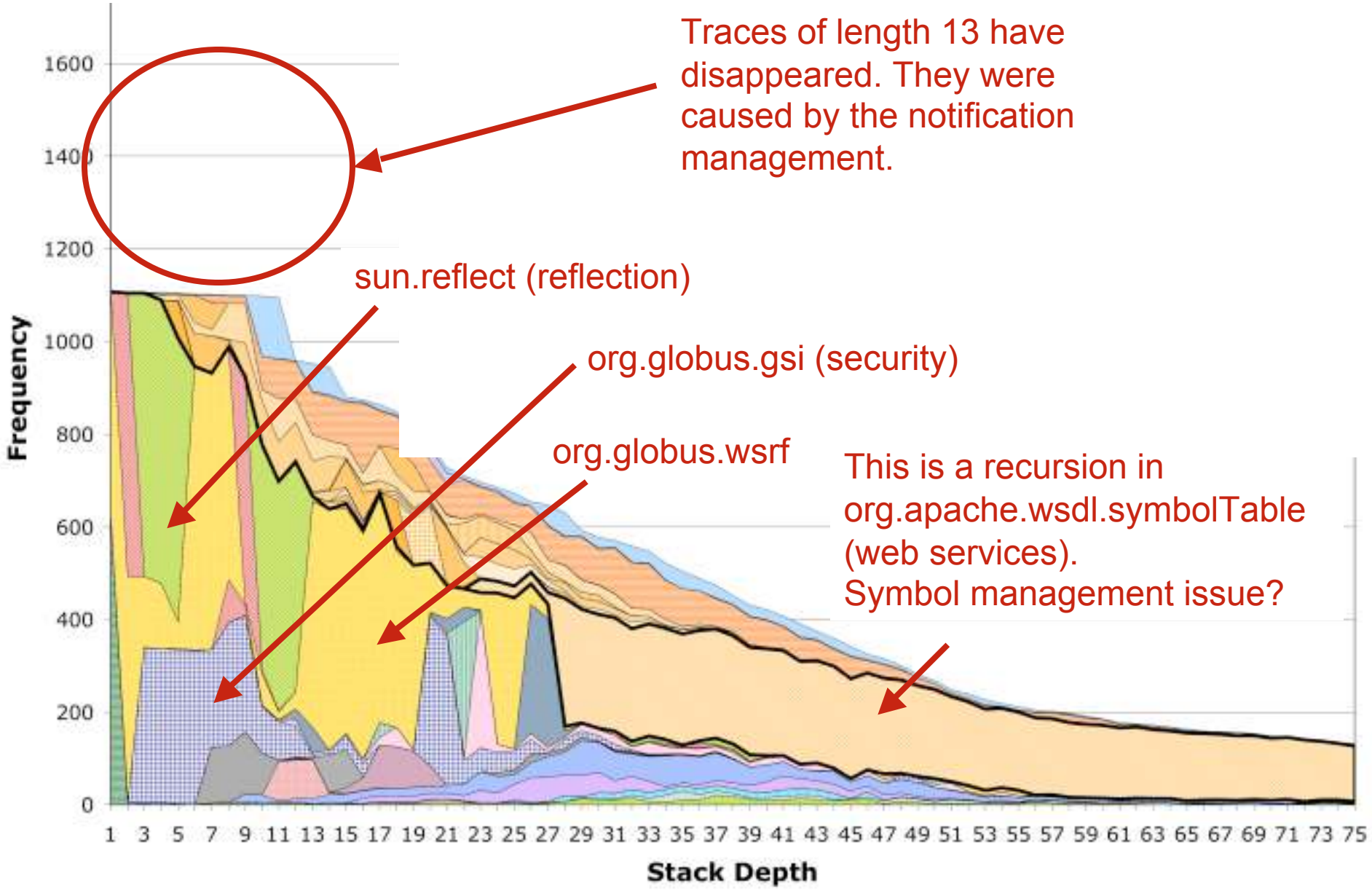


New Experimental Set-Up



+ extra granularity to observe package org.apache.axis

New Results


Traces of length 13 have disappeared. They were caused by the notification management.
sun.reflect (reflection)
org.globus.gsi (security)
org.globus.wsrf
This is a recursion in org.apache.wsdl.symbolTable (web services).
Symbol management issue?
Frequency
0
200
400
600
800
1000
1200
1400
1600
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75
Stack Depth

Profiling Breakdown

- Abstracts away **low level** packages (java.*, etc.)
- **Sample breakdown** among **"higher level"** packages:

Package Name	*Samples*	*%*
➔ org.apache.axis.wsdl	231	21%
➔ org.apache.axis.encoding	66	6%
➔ org.apache.axis (others)	113	10%
➔ org.globus.gsi	249	23%
➔ org.globus.wsrf	49	4%
➔ cryptix.provider.rsa	82	7%
➔ org.apache.xerces	78	7%
➔ others	237	21%

Profiling Breakdown

- Abstracts away **low level** packages (java.*, etc.)
- **Sample breakdown** among **"higher level"** packages:

Package Name	*Samples*	*%*
➔ **org.apache.axis.wsdl**	**231**	**21%**
➔ org.apache.axis.encoding	66	6%
➔ org.apache.axis (others)	113	10%
➔ org.globus.gsi	249	23%
➔ org.globus.wsrf	49	4%
➔ cryptix.provider.rsa	82	7%
➔ org.apache.xerces	78	7%
➔ others	237	21%

Symbol management issue?

Profiling Breakdown

- Abstracts away **low level** packages (java.*, etc.)
- **Sample breakdown** among **"higher level"** packages:

Package Name	*Samples*	*%*
➔ **org.apache.axis.wsdl**	**231**	**21%**
➔ **org.apache.axis.encoding**	**66**	**6%**
➔ **org.apache.axis (others)**	**113**	**10%**
➔ org.globus.gsi	249	23%
➔ org.globus.wsrf	49	4%
➔ cryptix.provider.rsa	82	7%
➔ **org.apache.xerces**	**78**	**7%**
➔ others	237	21%

SOAP + XML: 44%

Profiling Breakdown

- Abstracts away **low level** packages (java.*, etc.)
- **Sample breakdown** among **"higher level"** packages:

Package Name	*Samples*	*%*
➔ org.apache.axis.wsdl	231	21%
➔ org.apache.axis.encoding	66	6%
➔ org.apache.axis (others)	113	10%
➔ **org.globus.gsi**	**249**	**23%**
➔ org.globus.wsrf	49	4%
➔ **cryptix.provider.rsa**	**82**	**7%**
➔ org.apache.xerces	78	7%
➔ others	237	21%

Security / Cryptography: 30%

(Temporary) Conclusion on Globus

- **Globus**:
  - **Lazy optimisation**: very high latency on first invocation of operations (up to 30s to set up a resource on a new container!)
  - **Stabilized latencies** still high: ~ 160ms for a round trip request (with authentication turned on)
- No clear culprit. A mix of factors: **WSDL, SOAP, security**
- Is **lazy optimisation** a **problem**? **Yes and No.**
  - Brand new version. 3.9.4 numbers already better than 3.9.2!
  - Containers not supposed to be started frequently
  - Globus services are there to manage very long running jobs. A few seconds does not really matter.
  - But points at some applications for which Globus (in its present form) would be clearly ill chosen

Conclusion & Outlook On Approach

- Use of **simple** and **well known** profiling **techniques**
- Visualisation was adapted to **scale up** to the **complexity** of a software like Globus
- The **diagrams** we used don't contain all the answers:
  - ➔ They can be best seen as **maps** to guide further steps
  - ➔ **Different kinds** of projection actually useful
- Interesting complexity related problems:
  - ➔ Which is the best **"semantically relevant"** level to project profiling traces? Too low: no meaning. Too high: no details.
  - ➔ Can we leverage the **"middleware" nature** of Globus to obtain **finer profiling data** with the same lightweight tools?

The End

(Thank you)