WWV 2013

Deconstructing Complex Distributed Platforms

A Report from the Trenches

François Taïani

Joint work with

at&t

- Matti Hiltunen
- Rick Schlichting

LANCASTER UNIVERSITY

- Shen Lin
- Tom Ormerod
- Linden Ball

Distributed Software Today

Standards

JSON
JavaScript Object Notation

External developers

facebook

twitter

bit.ly

External services

Large, complex, multi-party

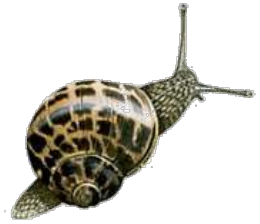
Geosocial app, est. 2009

10M Users

Middleware

Why deconstruct software?
(in particular distributed)

- Needed for **diagnosis / modifications**
  - ➔ Harden software (FT, replication)
  - ➔ To address non-functional issues (e.g. performance)



- Assembled from **third party** components
  - ➔ Unfamiliar software
  - ➔ Frankenstein effect

Why is it difficult?

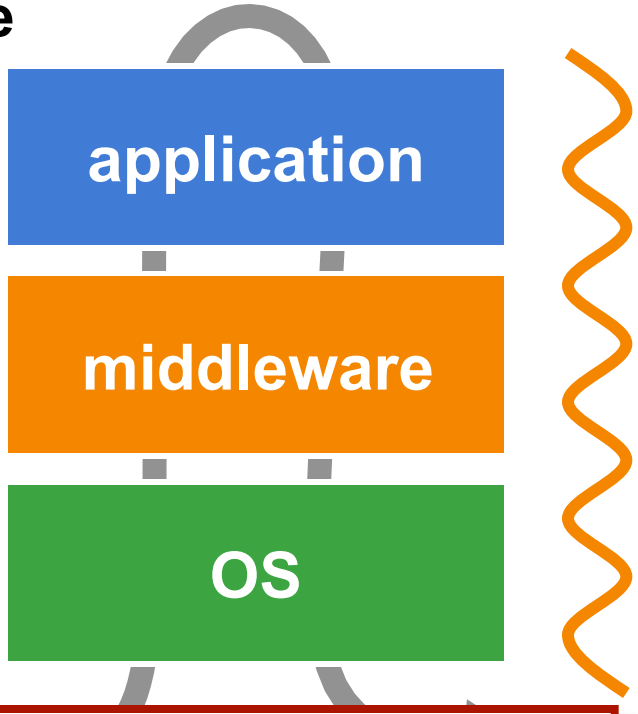
- Software easier to construct than to **understand**
- Particularly true for **emergent dynamic behaviours**
  - ➔ e.g. **dependability** and **performance**
  - ➔ arise from interactions within/ without target system
- **Dynamic** observation data difficult to make sense of
  - ➔ **large** size (up to 1,000,000 invocations for a few sec)
  - ➔ likely to involves **poorly understood** parts

Example 1: Replication

- Replicating COTS applications: **non-determinism**
  - ➔ Observe mutex activities
  - ➔ Link mutexes to **request life cycle**

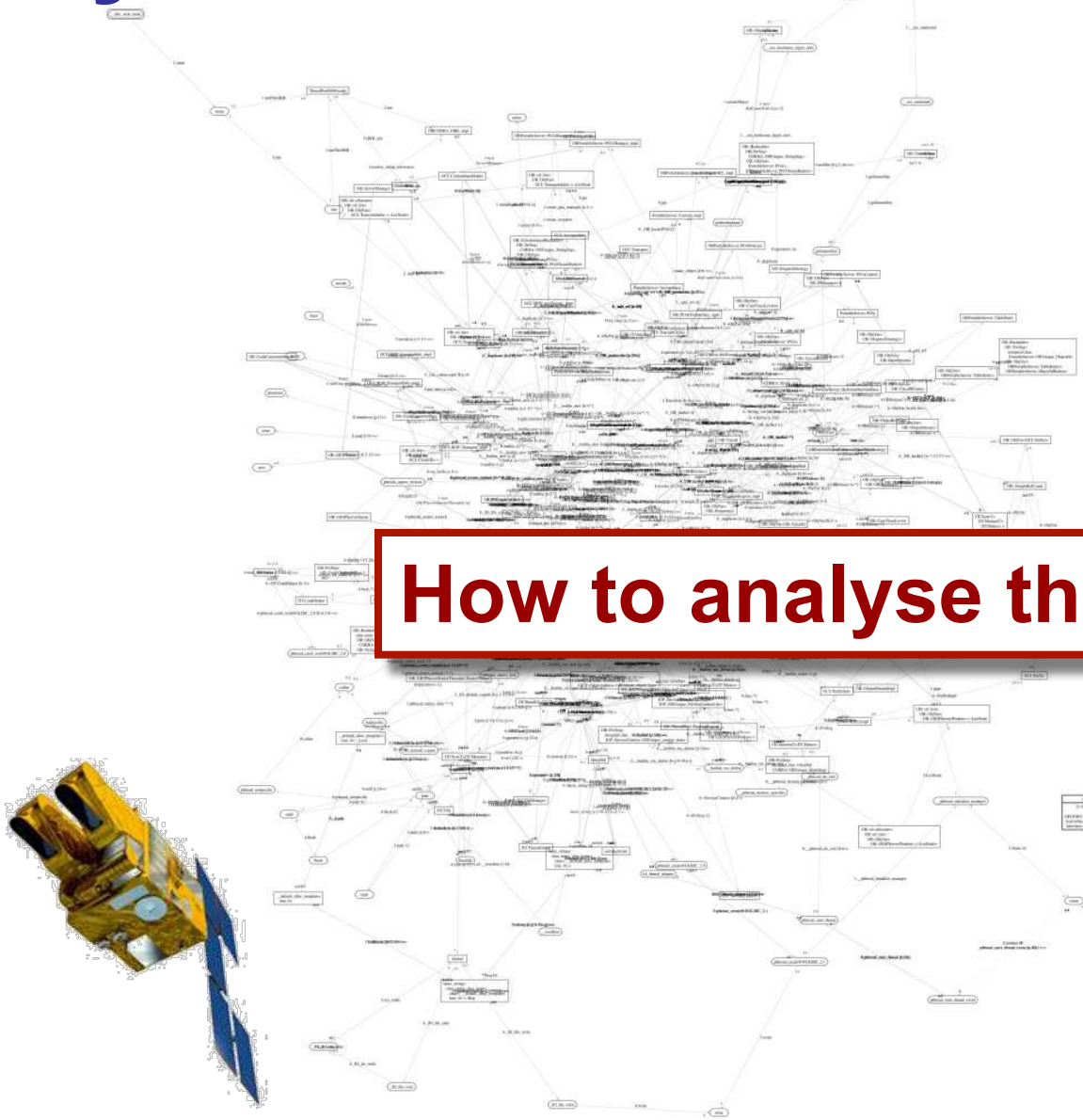
middleware services
(RPC, pub/sub, …)

syscalls, syslibs
(synchronisation,
memory managment...)



Approach: dynamic reverse engineering

Dynamic Reverse Engineering



ORBacus
1 Request

one request,
2065 individual invocations,
over 50 C-functions and 140
C++ classes.

How to analyse this?

Example 2: Profiling

- **Globus** (Argonne): ref. implementation for Grid
  - ➔ Grid Computing + Web Services

- Transition to WS stack (Version 3.9.x, 2005)
  - ➔ within a short time (a few months)
  - ➔ large, complex, collaborative (IBM, Apache,...) ➚ reuse

- **But … poor performances**
  - ➔ Up to **30s** to **create** a simple distributed object (counter)
  - ➔ Up to **2s** for a roundtrip remote **add** operation

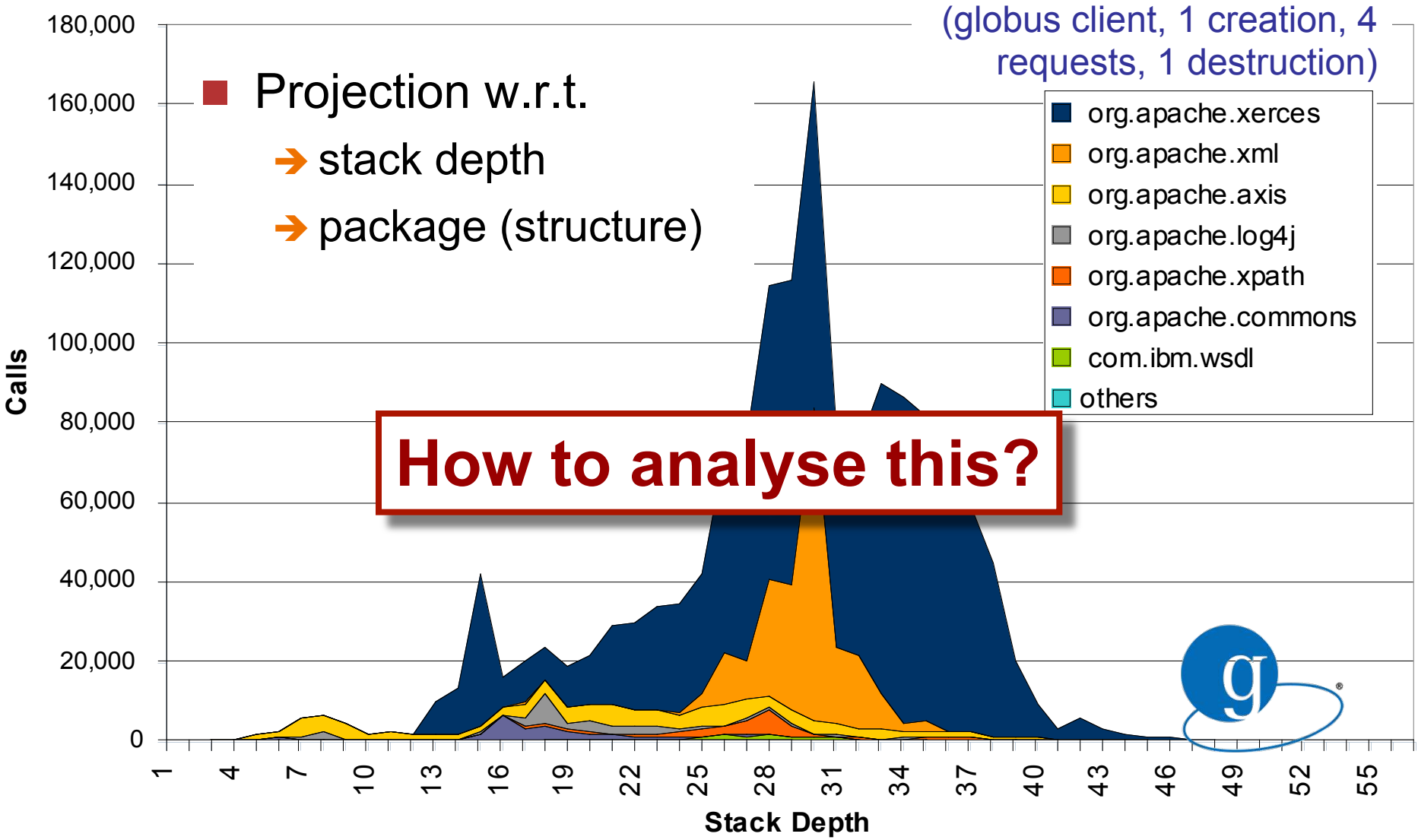
Profiling

- **Large piece of** software (3.9.x)
  - ➔ 123,839 lines in Java (w/oreused libraries)

- **Many libraries** layered
  - ➔ XML, WSDL (Descr. Lang), WSRF (Resource Fwork)
  - ➔ Axis (SOAP), Xerces (XML Parsing), com.ibm.wsdl

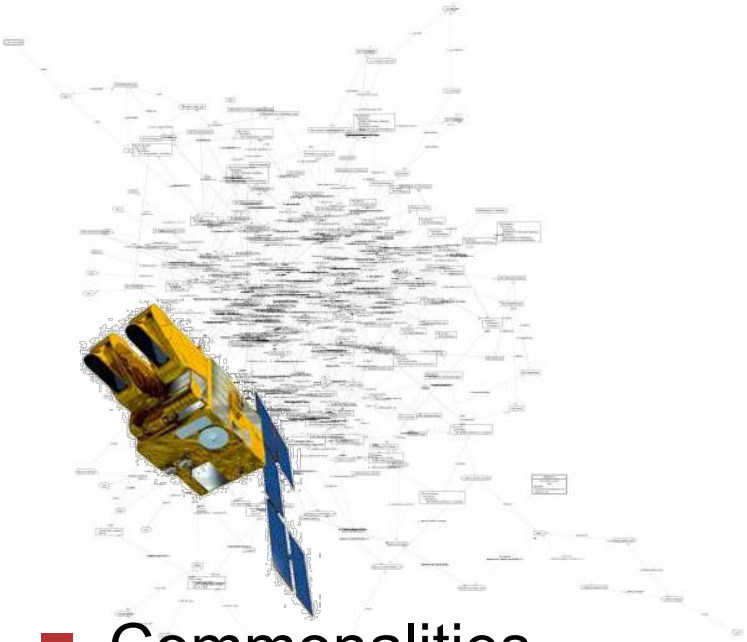
- Java: **exhaustive tracing** (outside the JVM libs)
  - ➔ client : **1,544,734** local method call (sic)
  - ➔ server : **6,466,652** local method calls (sic) [+time out]

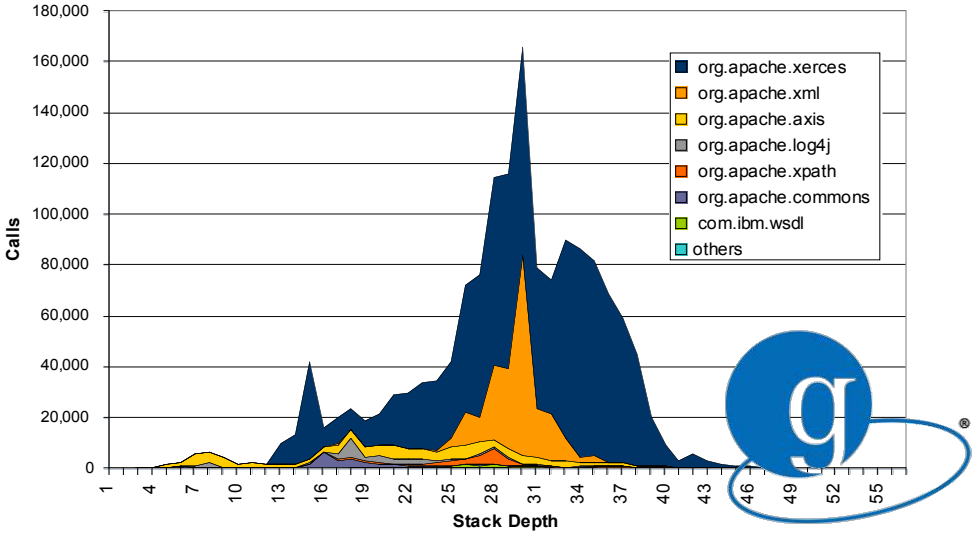
Profiling



The Impact of Web Service Integration on Grid Performance. Taïani, Hiltunen, Schlichting, HPDC-14, 2005

The Deconstruction Problem





- Commonalities
  - ➔ Large scale, software intensive, composite
  - ➔ Unfamiliar
- Our general philosophy
  - ➔ Pragmatic, semi-automatic, interactive

Outline

- Motivation

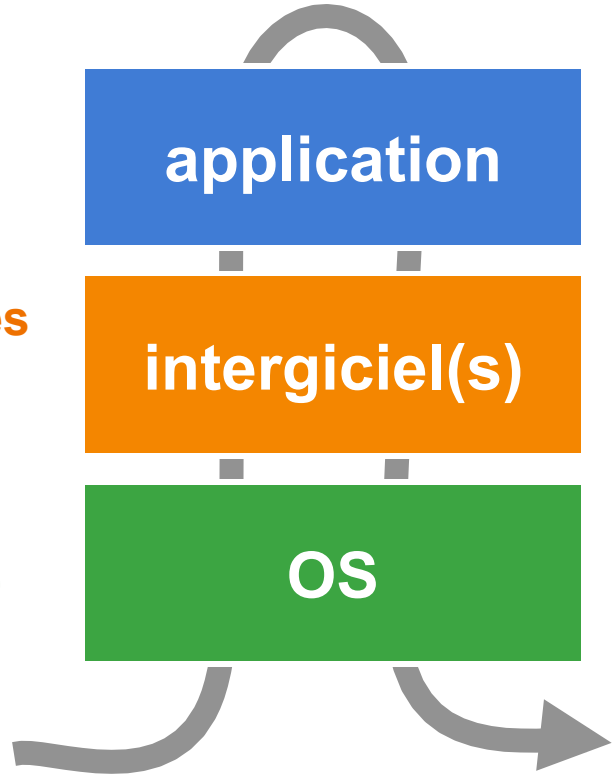
- Example 1:
  - ➔Analysing **locking behaviour** in industrial ORB
- Example 2:
  - ➔Visualizing **performance anomalies** in Grid MW
- Some concluding words

Back to Example 1

- Original goal
  - ➔ Control non-determinism in industry grade middleware
  - ➔ Observation of mutex activities
  - ➔ Link to request life cycle

middleware services
(RPC, pub/sub, …)

syscalls, syslibs
(synchronisation,
memory managment...)



Back to Example 1

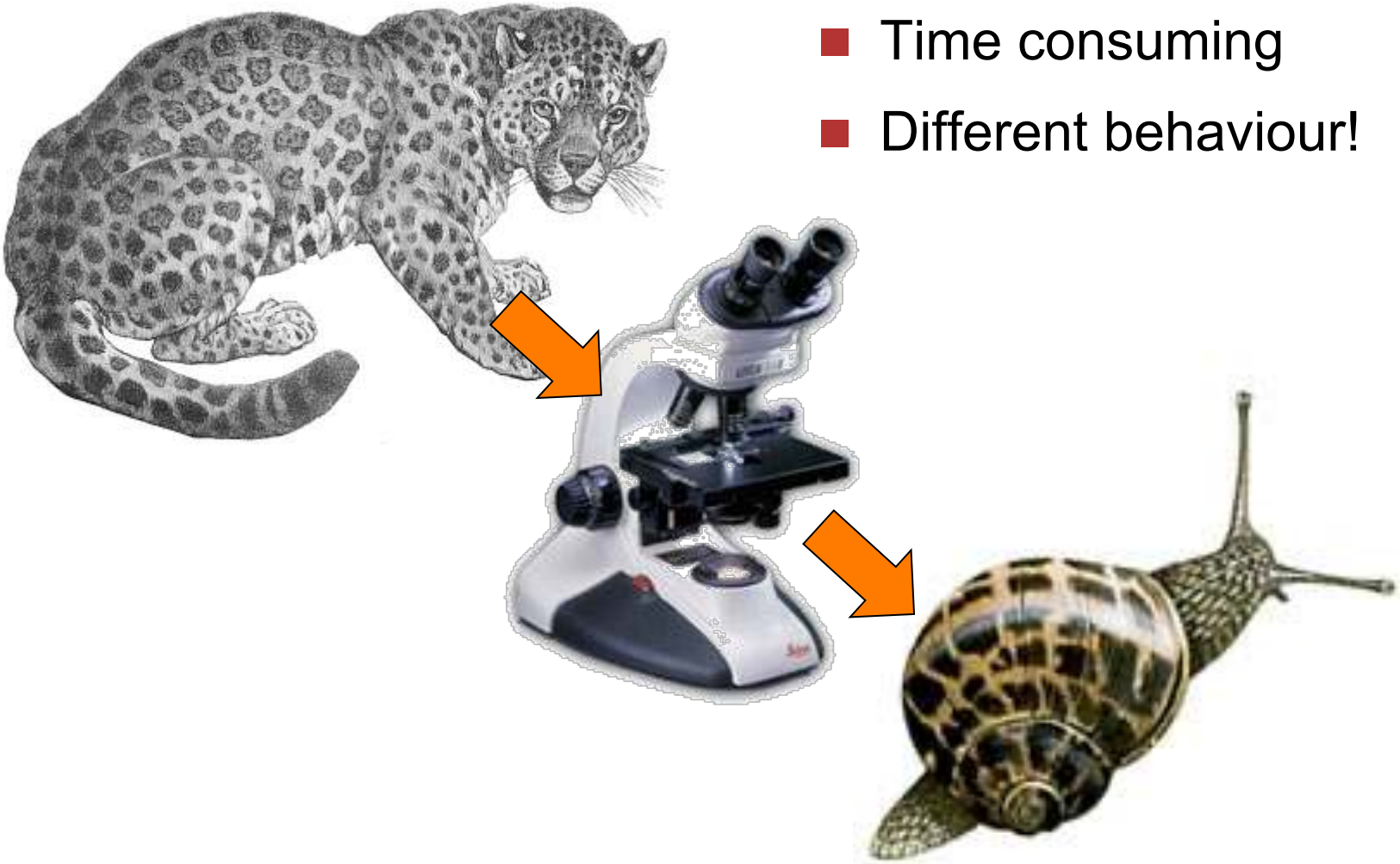
- Original goal
  - ➔ Control non-determinism in industry grade middleware
  - ➔ Observation of mutex activities
  - ➔ Link to request life cycle

- Complex & layered system
- Hard to observe and analyse

  Problem 1: Observation cost

  Problem 2: Cross-layer entangling

Problem 1: Observation costs



- Time consuming
- Different behaviour!

Problem 2: Cross-layer entangling

dummy1

dummy2

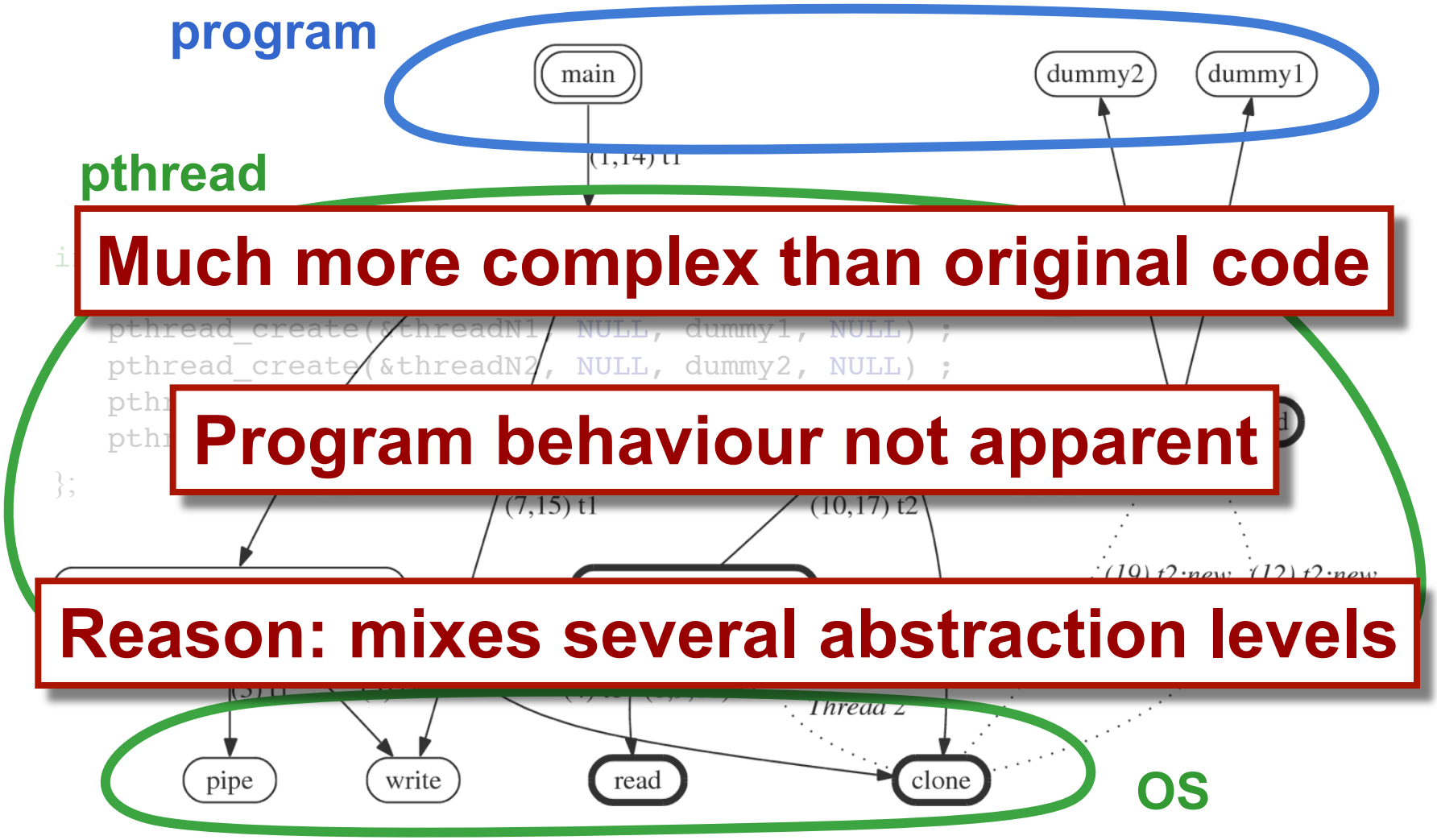
```
int main () {
   pthread_t threadN1, threadN2 ;
   pthread_create(&threadN1, NULL, dummy1, NULL) ;
   pthread_create(&threadN2, NULL, dummy2, NULL) ;

};
```

Can we reconstruct main's behaviour?

OS

observa by tracing

Problem 2: Cross-layer entangling



Much more complex than original code

Program behaviour not apparent

Reason: mixes several abstraction levels

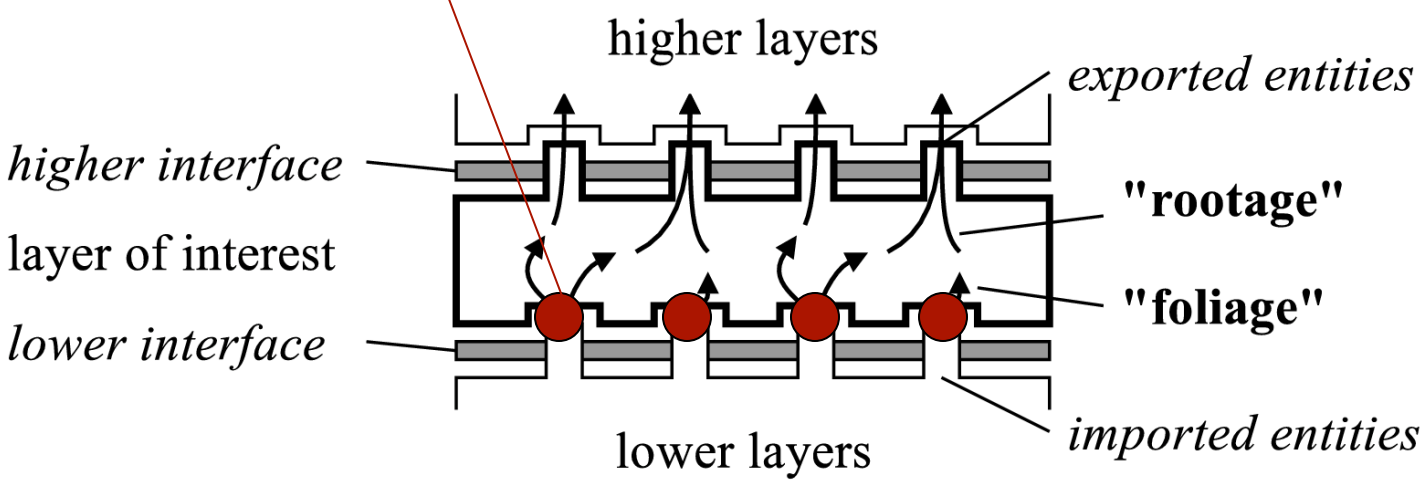
Approach: “interferometry”

- **Limit** observation costs: **more for less**
  - ➔ capture stack traces (more)
  - ➔ limit observation points to component boundaries (less)
- **Filter** out unneeded data: “interferometry”
  - ➔ graph manipulation script language
  - ➔ reusable filters

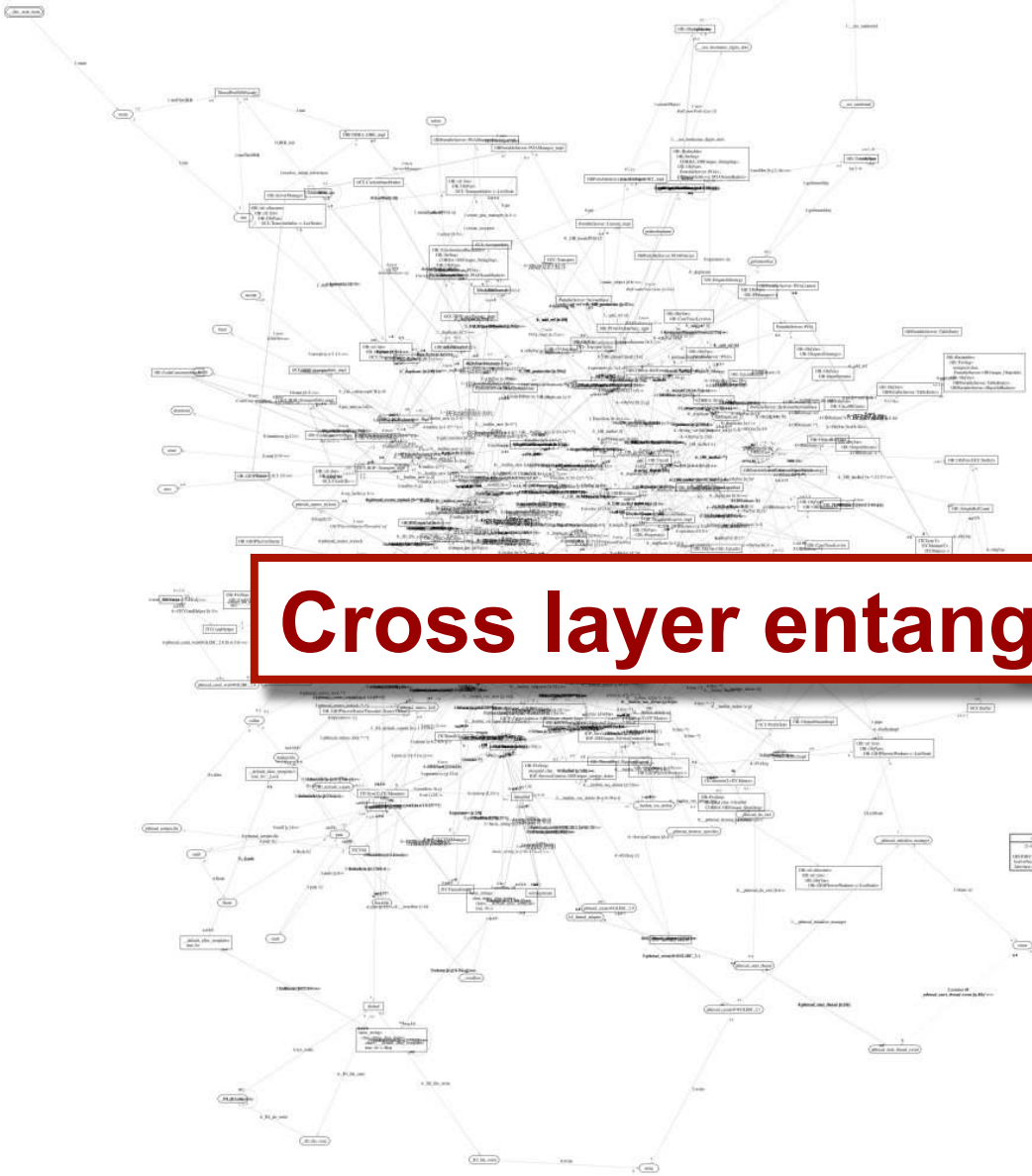
vs.

Tackling Observation Costs

```
#0  0x405c8f20 in send () from /lib/libpthread.so.0
#1  0x4015804a in omni::tcpConnection::Send ()
#2  0x4011364c in omni::giopStream::sendChunk ()
[..]
#8  0x400bdb0a in omniAsyncWorker::run ()
#9  0x405abbdf in omni_thread_wrapper ()
#10 0x405c2bf0 in pthread_start_thread ()
#11 0x405c2c6f in pthread_start_thread_event ()
```



Result from Observation



ORBacus
1 Request

one request,
2065 individual invocations,
over 50 C-functions and 140 C++ classes.

Cross layer entangling

Tackling Entangling

A number of graph **operators**

Interactive graph scripting language

- **selection**

  `put ::pthread_create* G CREATE`

- **recursive extension**

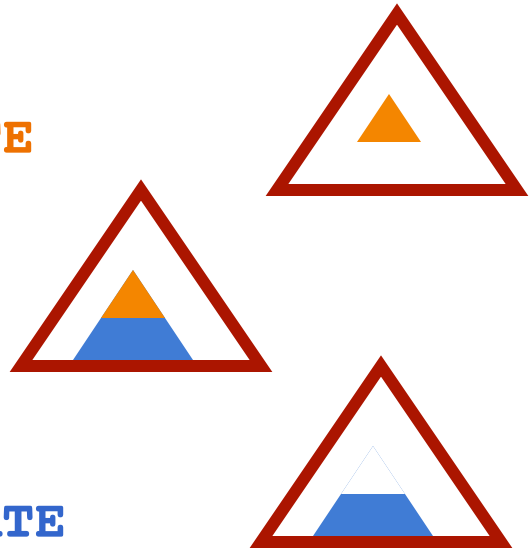
  `forward CREATE G`

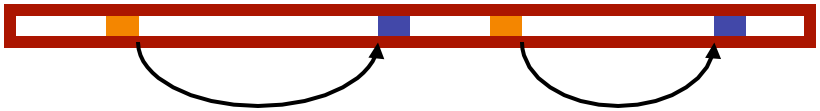
- **boolean algebra**

  `remove ::pthread_create* CREATE`

- **temporal operations**

  `fuse ::pthread_create* ::pthread_start_thread* G`





Disentangling Script

```
// [remove pthread]
put ::recv'* GlobalGraph R
put ::send'* GlobalGraph R
put Hello_impl::say_hello'* GlobalGraph R
put ::accept'* GlobalGraph R
backward R GlobalGraph
put ::lsf_thread_adapter'* ABS
put JTC* ABS
put OCI::* ABS
put ::__libc_start_main't1-0 ABS
put ::run't1-9 ABS
put OB::ORBControl::initializeRootPOA't1-11 ABS
put ORPortableServer::POAManagerFactory_impl::create_poa_manager't1-12 POA
```

Pattern-based, reusable

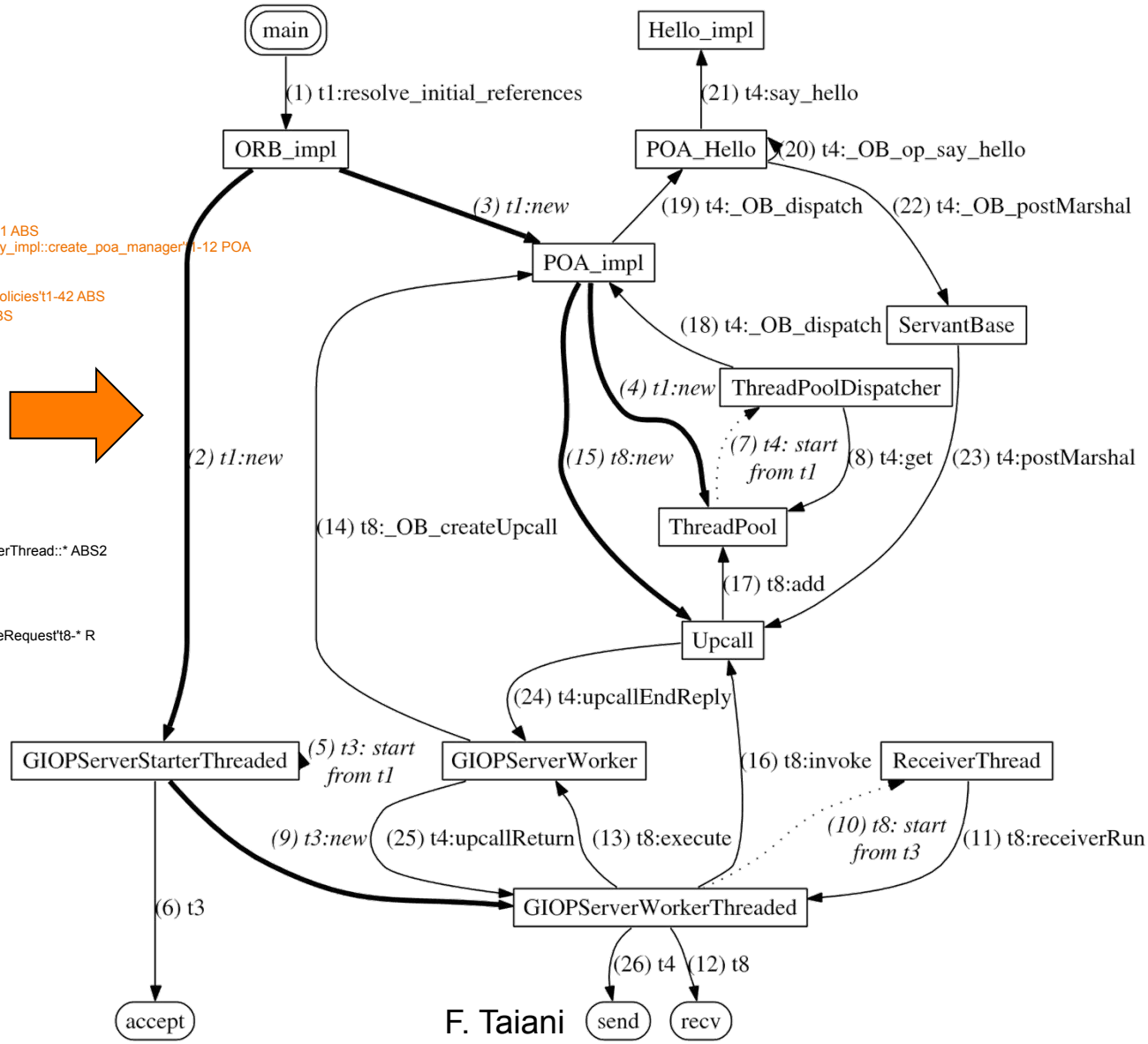
```
es't1-42 ABS
abstract ABS R
abstract ABS GlobalGraph
remove ::accept't3-882 R
absPatern ::recv't8-828 R
absPatern ::recv't8-1309 R
absPatern ::recv't8-1908 R
put OB::Upcall::Upcall't8-1083 U
backward U GlobalGraph
add U R
put OB::ThreadPool::add't8-1224 TP
put OB::ThreadPool::get't4-1265 TP
backward TP GlobalGraph
add TP R
put OB::GIOPServerStarterThreaded::StarterThread::* ABS2
put OB::DispatchThreadPool_impl::* ABS2
put OB::DispatchRequest_impl::* ABS2
put OB::POAOAInterface_impl::* ABS2
abstract ABS2 R
absPatern OB::GIOPServerWorker::executeRequest't8-* R
```

2065 individual invocations, over 50 C-functions and 140 C++ classes.

CosmOpen: Dynamic reverse-engineering on a budget (journal version). Taïani, Killijian, Fabre, Software: Practice and Experience (SPE), John Wiley & Sons, Ltd. 39(18)

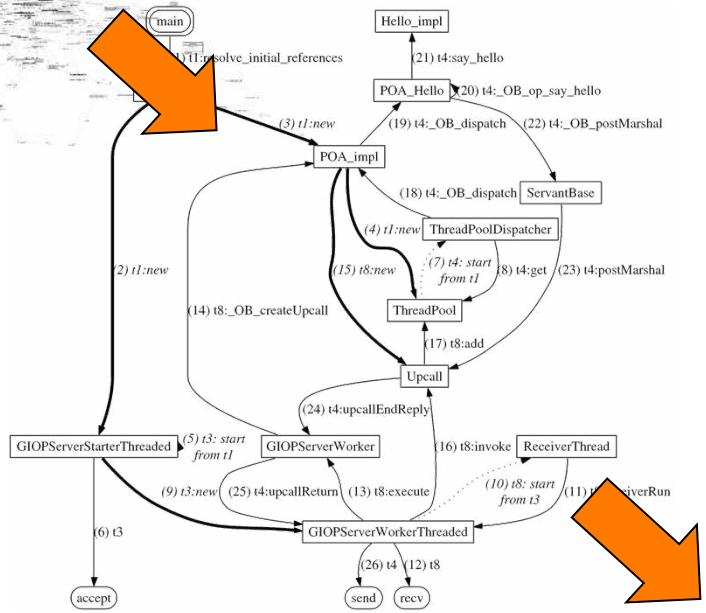
Result

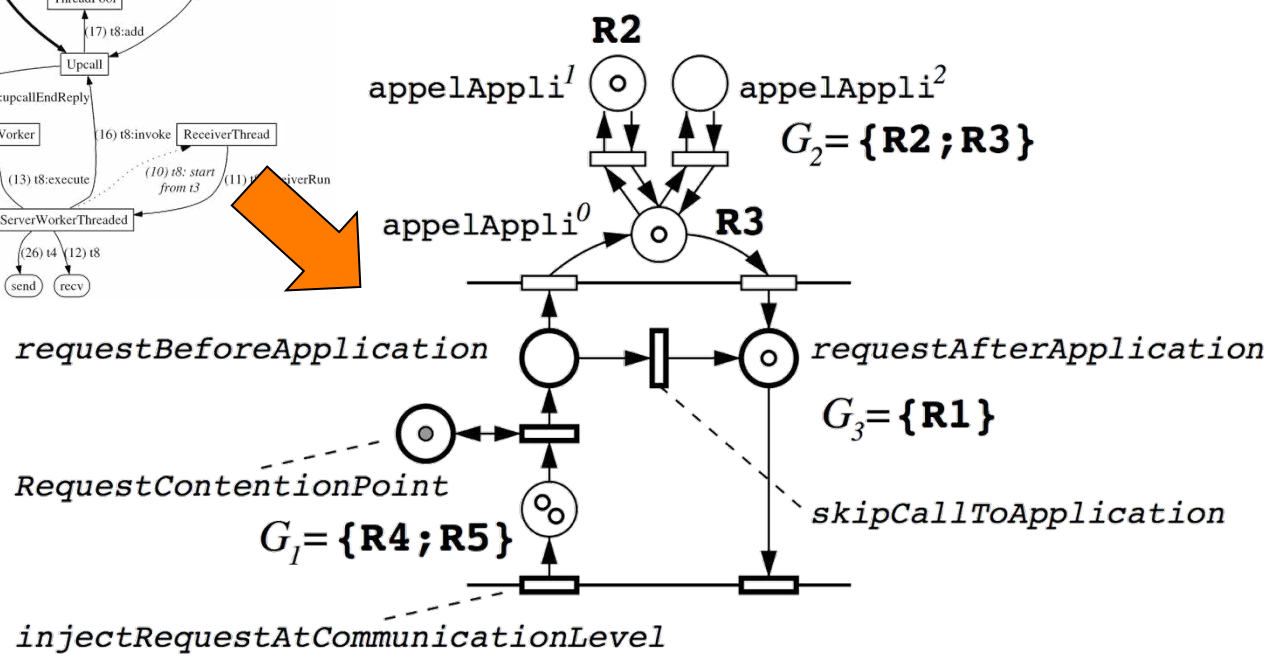
```
// [remove pthread]
put ::recv'* GlobalGraph R
put ::send'* GlobalGraph R
put Hello_impl::say_hello'* GlobalGraph R
put ::accept'* GlobalGraph R
backward R GlobalGraph
put ::lsf_thread_adapter'* ABS
put JTC* ABS
put OCI::* ABS
put ::__libc_start_main't1-0 ABS
put ::run't1-9 ABS
put OB::ORBControl::initializeRootPOA't1-11 ABS
put OBPortableServer::POAManagerFactory_impl::create_poa_manager't1-12 POA
forwN 3 POA R
add POA ABS
put OBPortableServer::POAPolicies::POAPolicies't1-42 ABS
put OB::DispatchStrategyFactory_impl::* ABS
abstract ABS R
abstract ABS GlobalGraph
remove ::accept't3-882 R
absPatern ::recv't8-828 R
absPatern ::recv't8-1309 R
absPatern ::recv't8-1908 R
put OB::Upcall::Upcall't8-1083 U
backward U GlobalGraph
add U R
put OB::ThreadPool::add't8-1224 TP
put OB::ThreadPool::get't4-1265 TP
backward TP GlobalGraph
add TP R
put OB::GIOPServerStarterThreaded::StarterThread::* ABS2
put OB::DispatchThreadPool_impl::* ABS2
put OB::DispatchRequest_impl::* ABS2
put OB::POAOAInterface_impl::* ABS2
abstract ABS2 R
absPatern OB::GIOPServerWorker::executeRequest't8-* R
```



Use

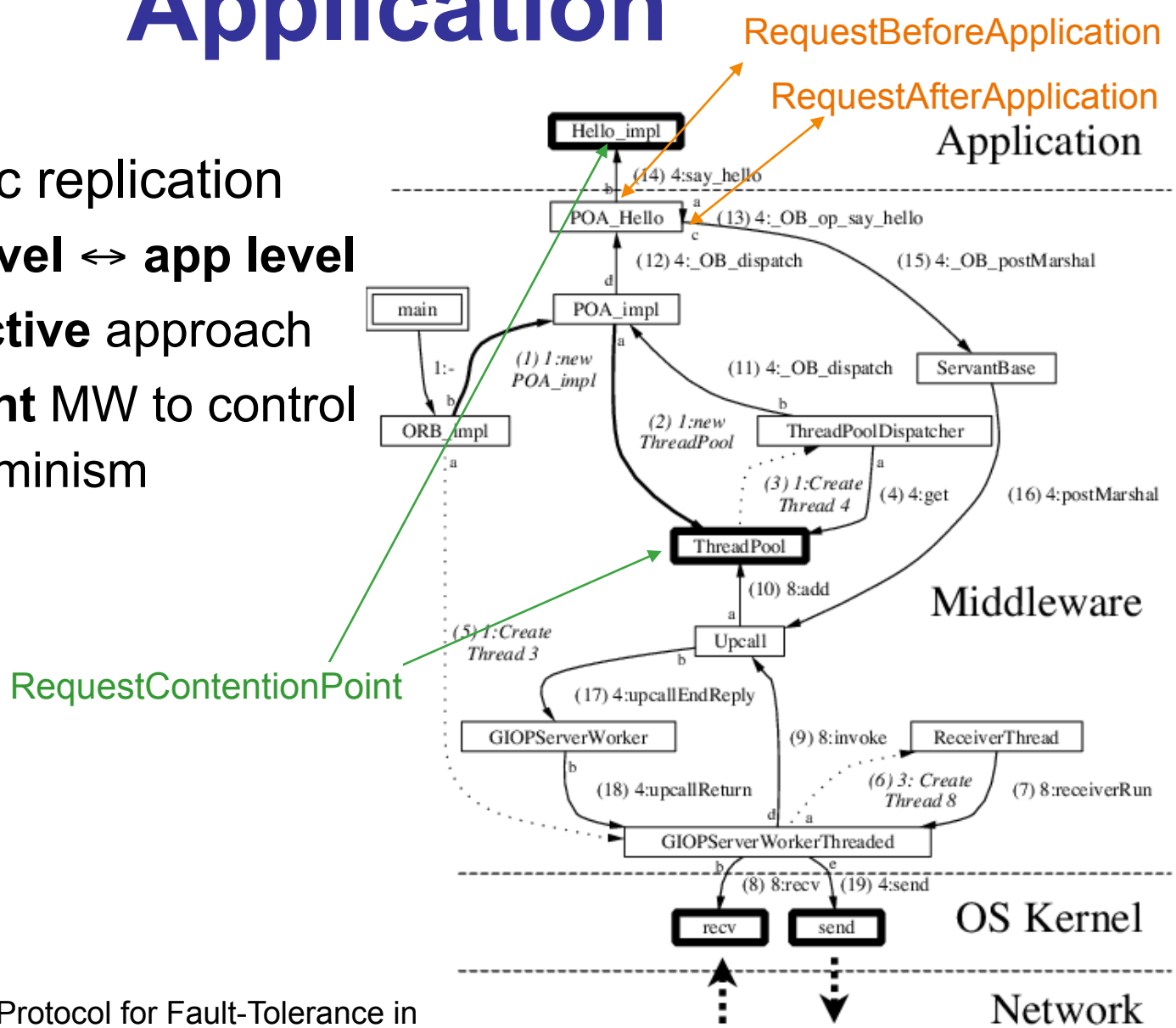
- Modelling
- Analysis





Application

- Deterministic replication
  - ➔ link **OS level** ↔ **app level**
  - ➔ use **reflective** approach
  - ➔ **instrument** MW to control non-determinism



A Multi-Level Meta-Object Protocol for Fault-Tolerance in Complex Architectures. Taïani, Fabre, Killijian, DSN'2005

Outline

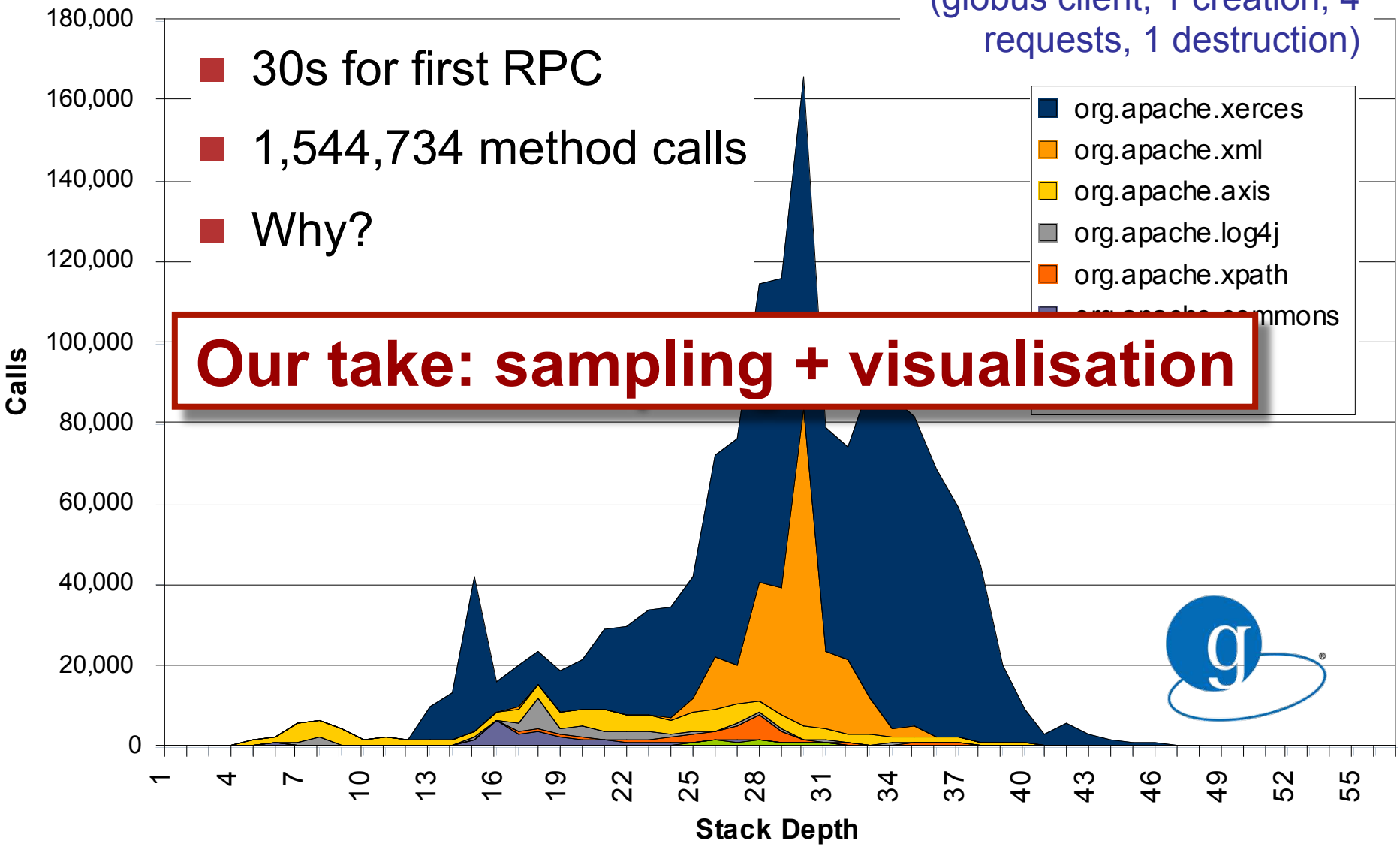
- Example 1:
  - ➔Analysing **locking behaviour** in industrial ORB
- Example 2:
  - ➔Visualizing **performance anomalies** in Grid MW
- Some concluding words

Example 2 : Profiling

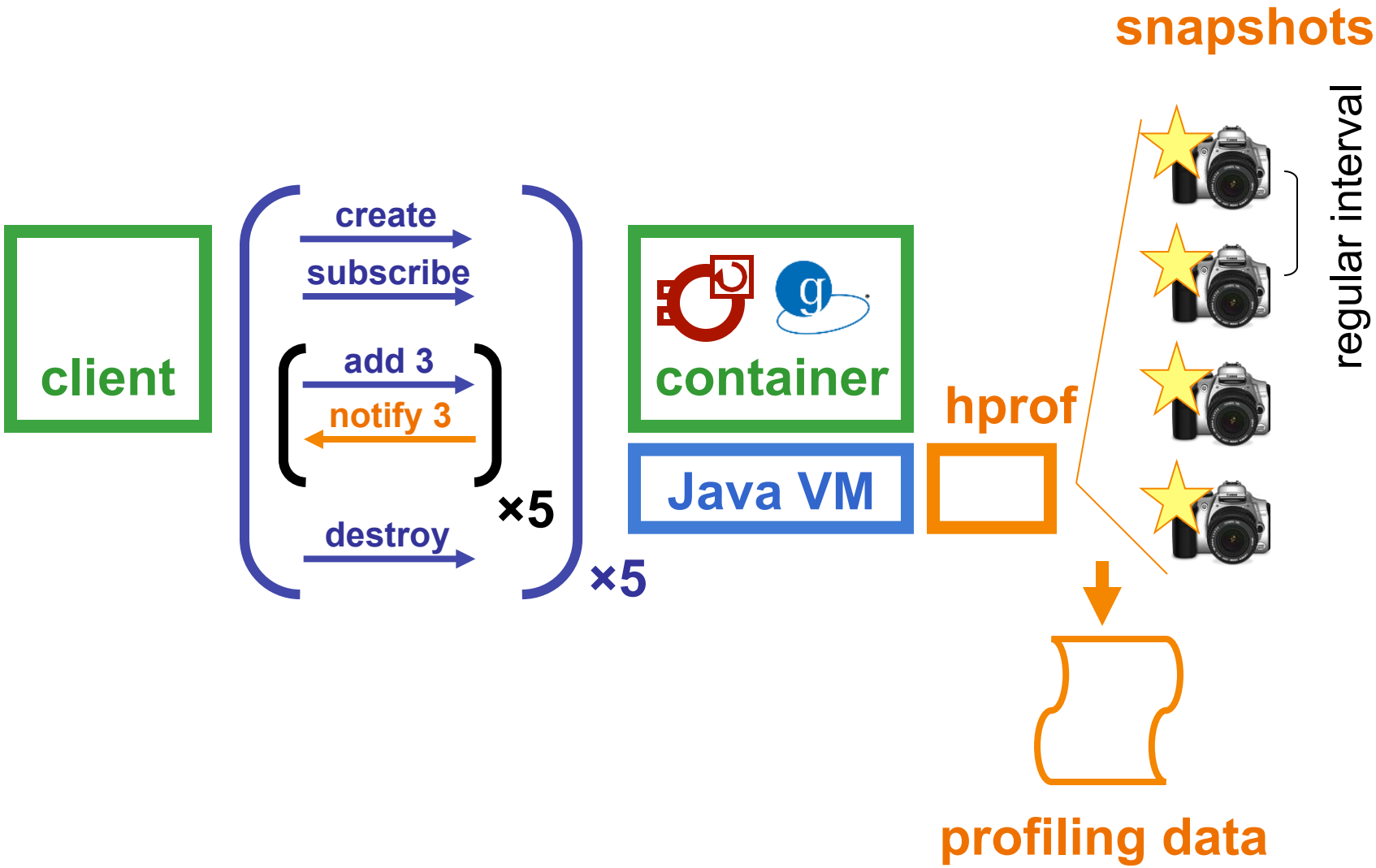
(globus client, 1 creation, 4 requests, 1 destruction)

- 30s for first RPC
- 1,544,734 method calls
- Why?

Our take: sampling + visualisation



Sample-based profiling



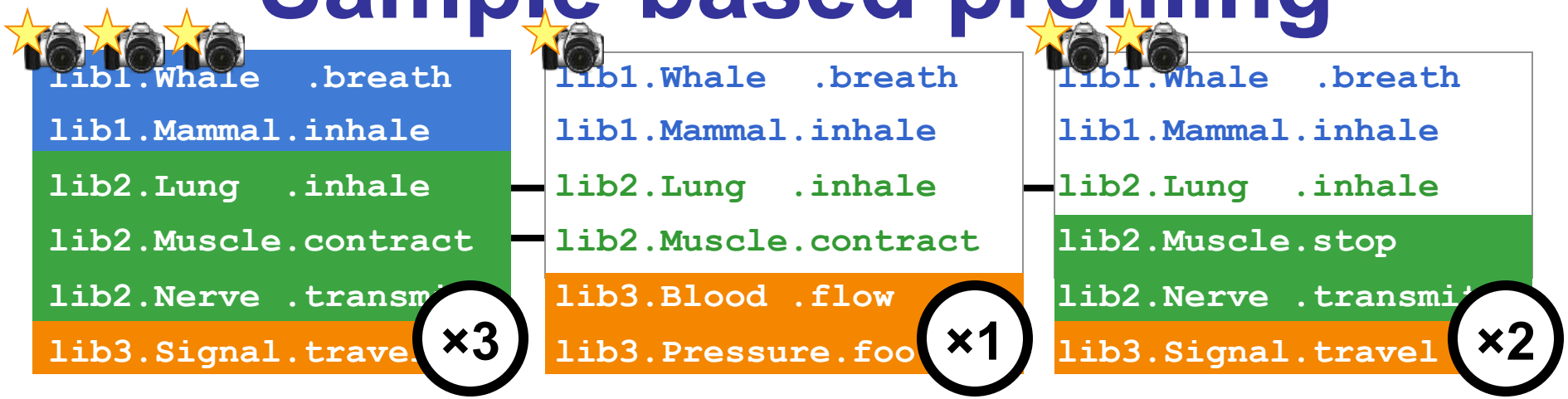
Sample-based profiling

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

```
lib3.Blood .flow
lib3.Pressure.foo
```

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

Sample-based profiling



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

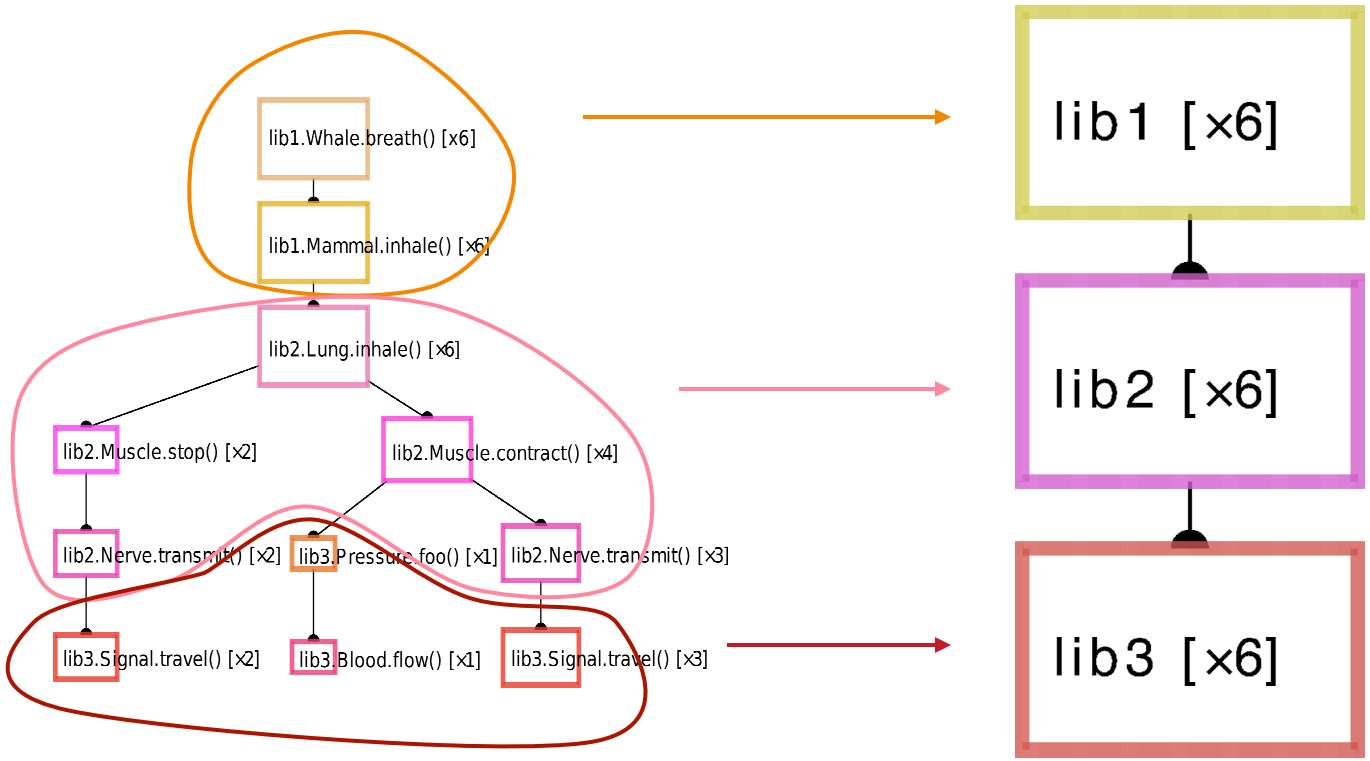
- Problem: **Data explosion remains.** On Globus :
  - ➔ 55550 method invocations
  - ➔ 1861 methods
  - ➔ 724 classes
  - ➔ 182 Java packages.
  - ➔ 32 threads

Visualising Dynamic Data

- Strategies: extract **salient features**
  - ➔ collapse recurring patterns (e.g. Jinsight)
  - ➔ remove and focus (e.g. CosmOpen)
  - ➔ data-mining (e.g. PCA in Xu et. al, SOSP'09)
- Our strategy
  - ➔ use **structural information** in dynamic data
    e.g. org.apache.axis.utils.ClassUtils.forName()
  - ➔ vary **'local abstraction'** level at which data is shown
- Contributions
  - ➔ application of structural compaction to **dynamic data**
  - ➔ algorithm for **localised structural compaction**
  - ➔ prototype and **explorative user study**

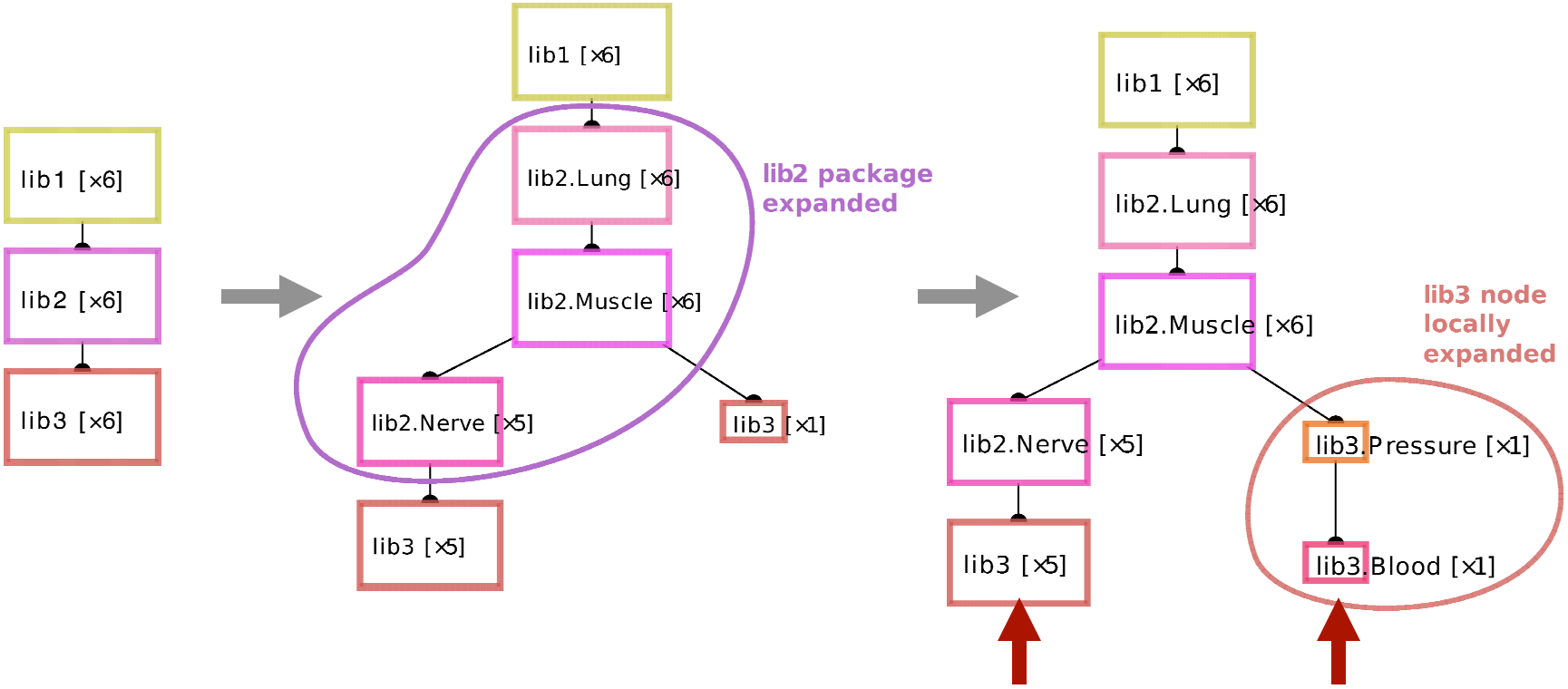
Compaction

- Only highest level packages visible



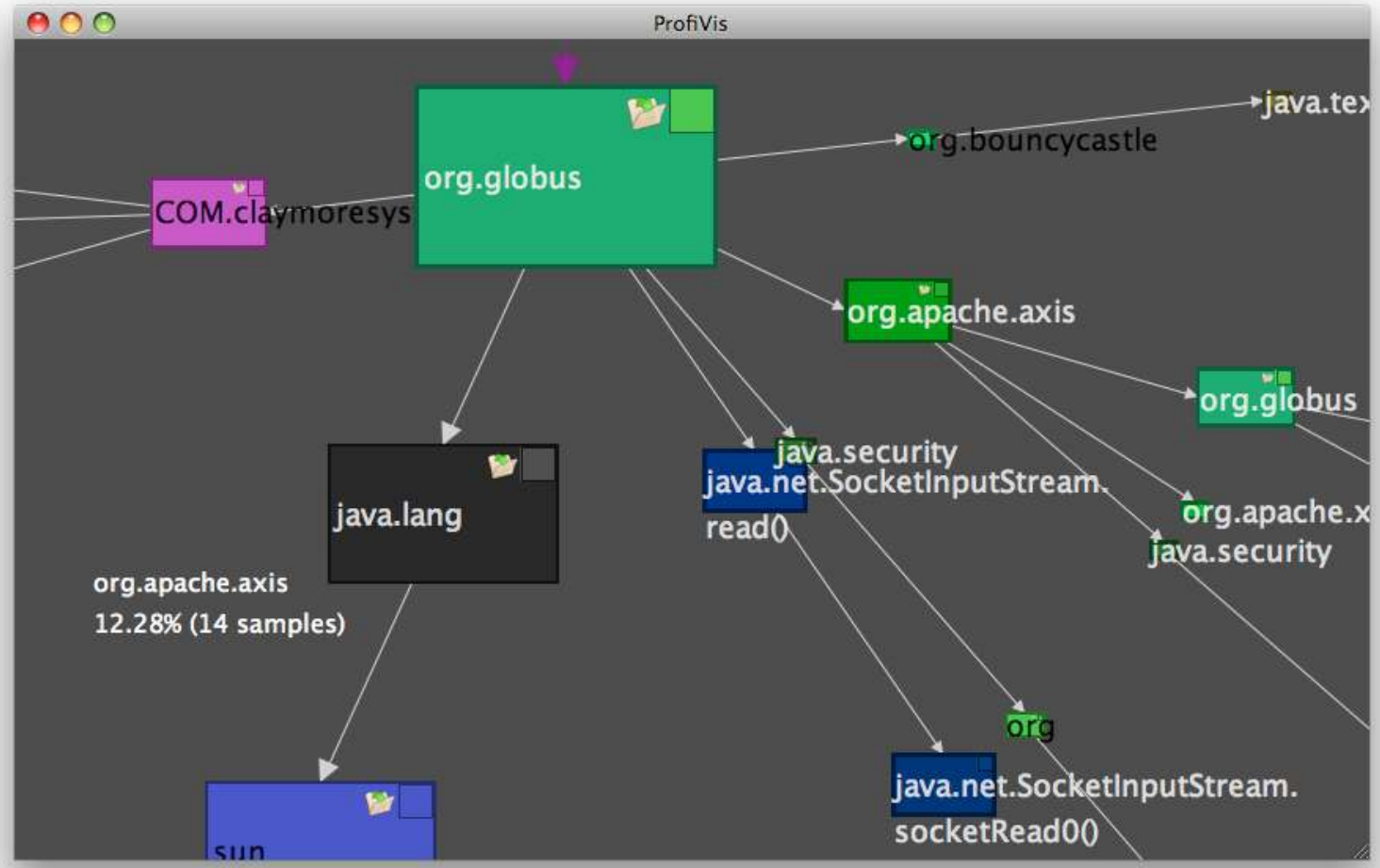
Progressive exploration

- Different levels of compaction in different parts of graph
  - ➔ including for the same package (here lib3)



different compaction levels for same package

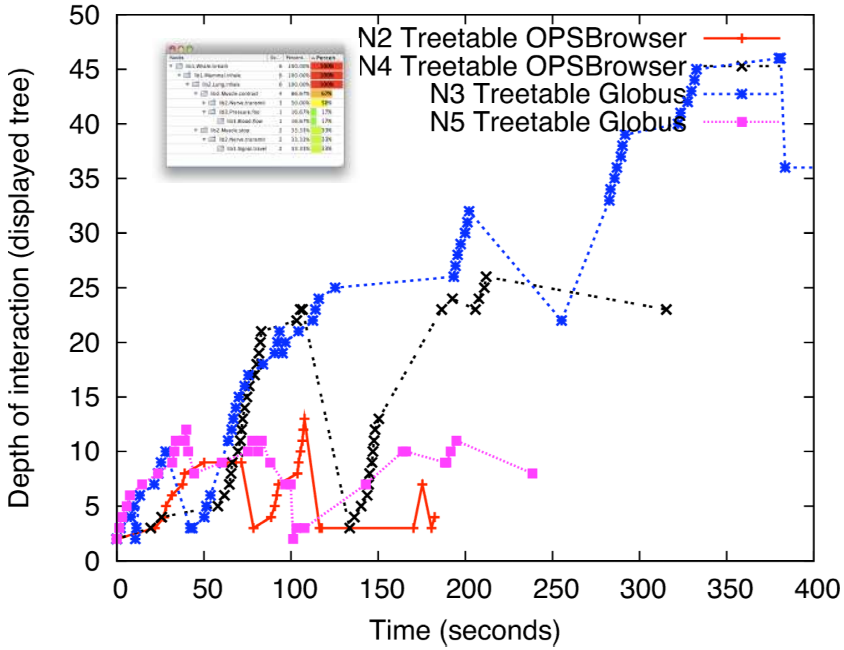
Demo

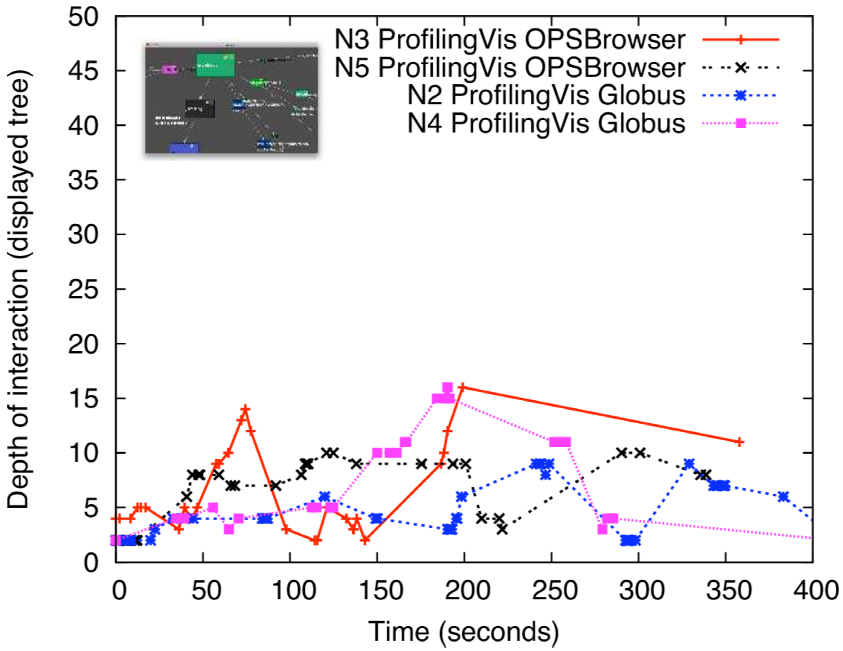


Towards anomaly comprehension: Using structural compression to navigate profiling call-trees S. Lin, F. Taiani, T. C. Ormerod, L. J. Ball, ACM SoftVis'2010
http://ftaiani.ouvaton.org/7-software/profvis.html

Evaluation: User Study

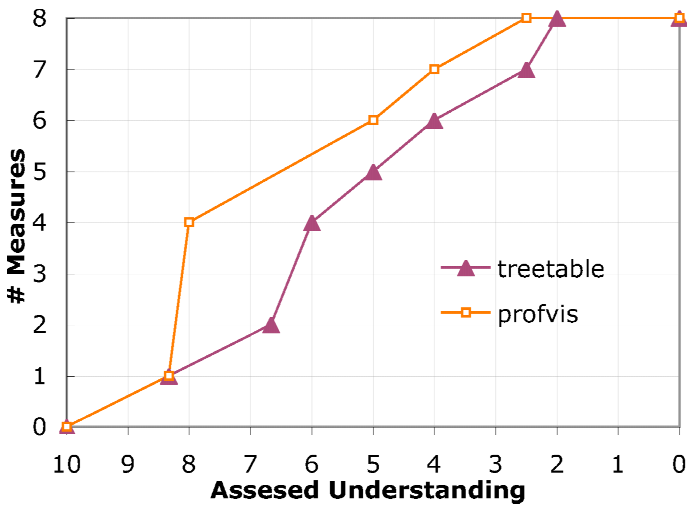
- Visual presentation high impact on exploration pattern
  - ➔ users go deep w/ TreeTable, tend to hover w/ ProfVis

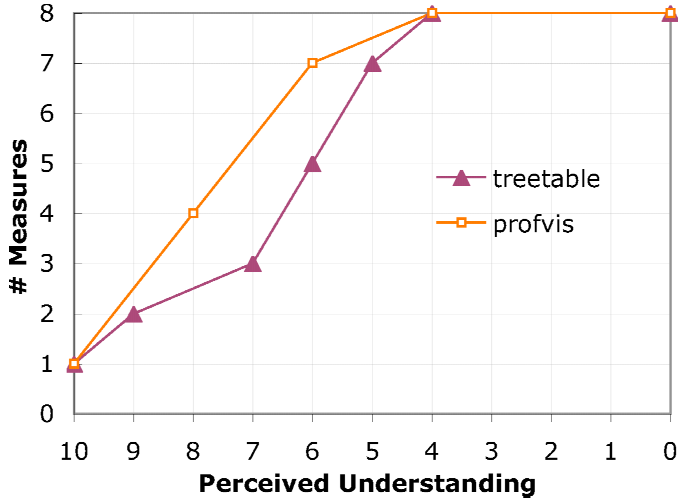




Evaluation: User Study

- Understanding
  - ➔ slight advantage for Provis
- But
  - ➔ very small study!
  - ➔ informally, users preferred TreeTable
- Questions raised
  - ➔ acceptability
  - ➔ ergonomics
  - ➔ design combination





Conclusion

- **Distributed Systems**
  - ➔ Increased reused → layered systems
  - ➔ Frankenstein effect : abstraction leaks

- **Pragmatic** approaches to dynamic reverse engineering
  - ➔ addresses complex layered software
  - ➔ aim: help user construct mental models of system

- **Future questions**
  - ➔ To which extent can this be automated (slicing? SA?)
  - ➔ Useful for verification?

Thank you

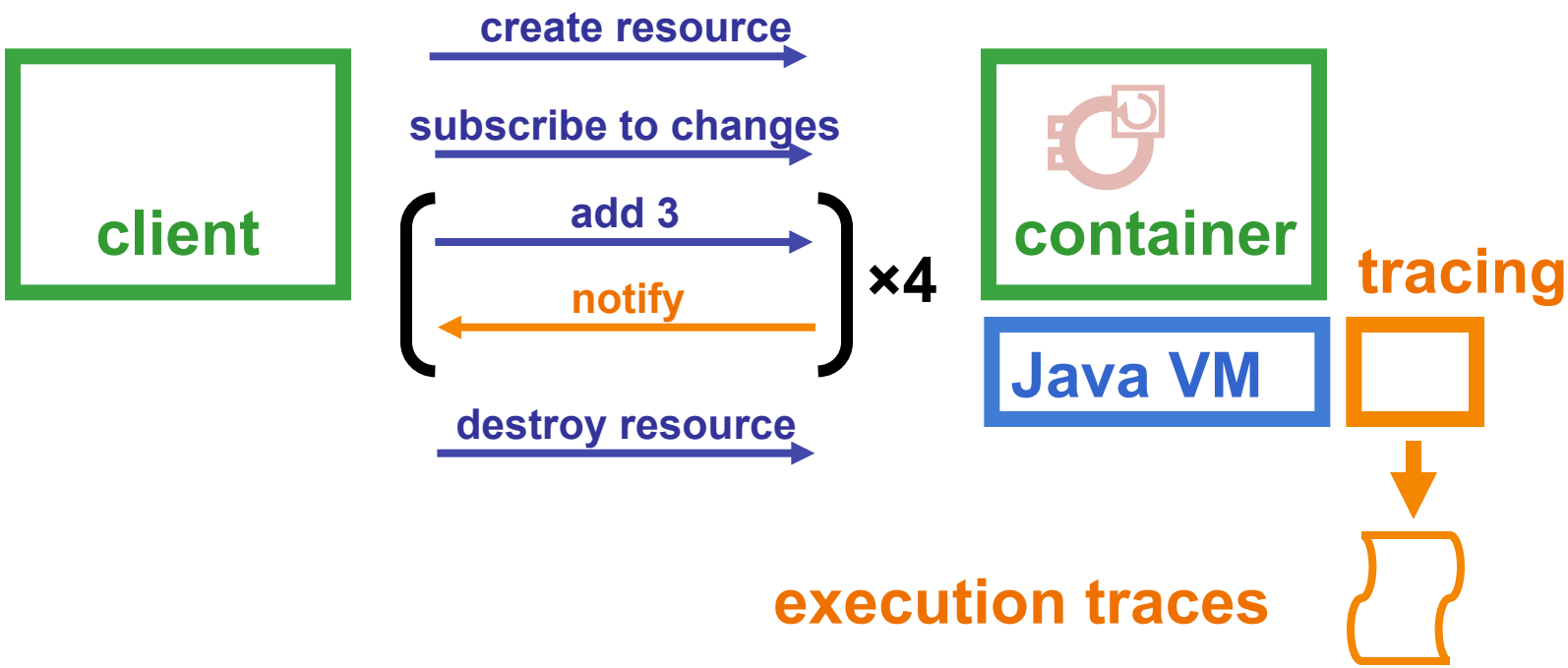
(Some) references

- Taïani F, Killijian MO, Fabre JC, CosmOpen: Dynamic reverse-engineering on a budget (journal version), Software: Practice and Experience, Wiley, 39(18): (Dec. 2009) pp. 1467-1514 (48p.)
- Taïani F, Hiltunen M, Schlichting R, The Impact of Web Service Integration on Grid Performance, The 14th IEEE Int. Symp. on High Performance Dist. Comp. (HPDC-14), 2005, pp.14-23 (10 p.)
- Chen YFR, Gansner ER, and Koutsofios E. A C++ data model supporting reachability analysis and dead code detection. SIGSOFT Softw. Eng. Notes 1997; 22(6):414-431
- Jerding DF, Stasko JT, and Ball T. Visualizing interactions in program executions. Proc. of the 19th Int. Conf. on Software Engineering (ICSE'97), ACM Press, 1997; 360-370
- Walker RJ, Murphy GC, Freeman-Benson B, Wright D, Swanson D, Isaak J. Visualizing Dynamic Software System Information through High-Level Models. OOPSLA'98, ACM Press, 1998; 271-283

(Some) references (cont)

- Systa T, Koskimies K, Mueller H. Shimba – an environment for reverse engineering Java software systems. Software Practice and Experience (SPE), 2001; 31(4):371-394. DOI: 10.1002/spe.386
- Wolf F, Mohr B, Dongarra J, Moore S: Efficient Pattern Search in Large Traces through Successive Refinement. Proc. of the 2004 European Conf. on Parallel Computing (Euro-Par), Lecture Notes in Computer Science, Springer, 2004; 3149:47-54.
- Arnold DC, Ahn DH, de Supinski BR, Lee GL, Miller BP, Schulz M. Stack Trace Analysis for Large Scale Debugging. Proc. of the IEEE Int. Parallel & Distributed Processing Symposium (IPDPS 2007), Long Beach, CA, 2007. DOI: 10.1109/IPDPS.2007.370254

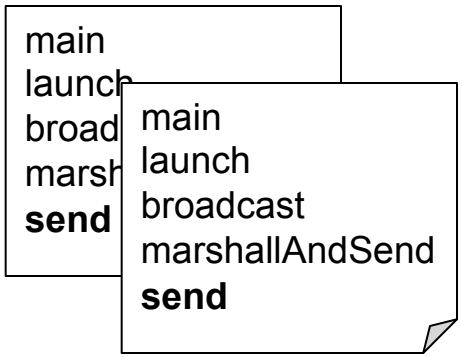
Methodology I + First Results



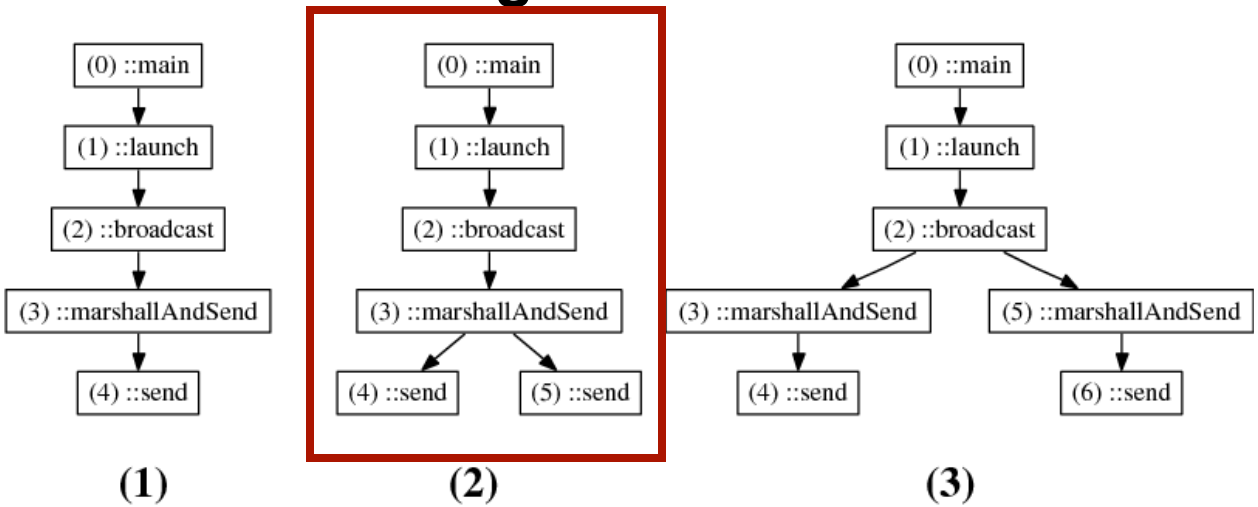
- First attempt: tracing everything (outside the JVM libs)
  - ➔ client : **1,544,734** local method call (sic)
  - ➔ server : **6,466,652** local method calls (sic) [+time out]
- **How to visualize such results?**

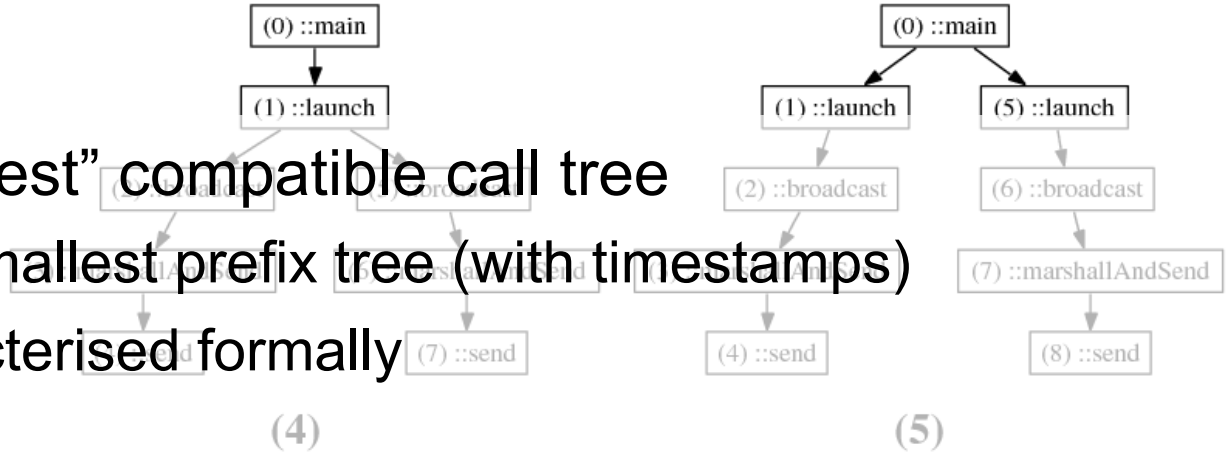
Reconstructing a call tree

- Problem: stack traces are **ambiguous**



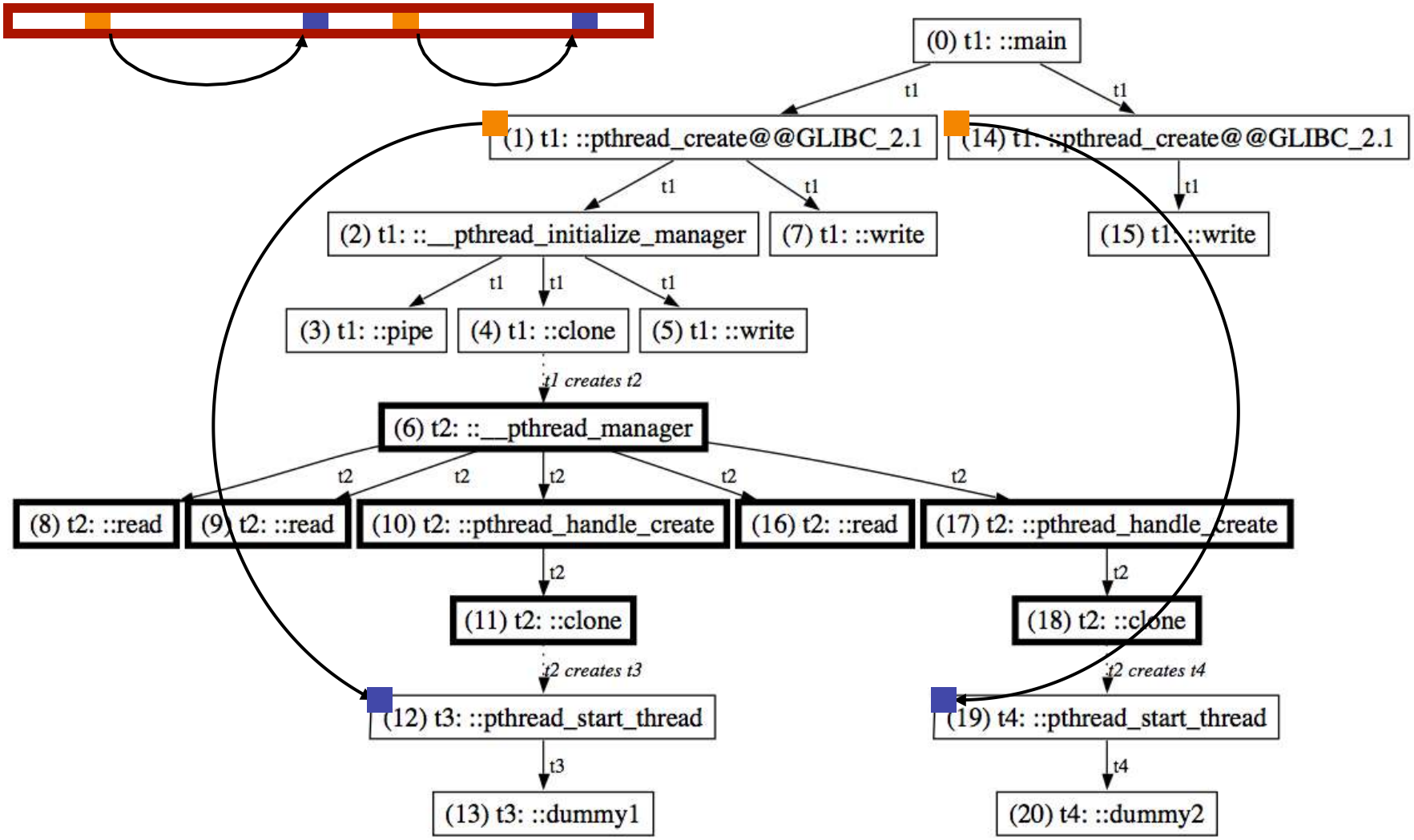
2 identical traces





- Choice: “smallest” compatible call tree
  - a variant of smallest prefix tree (with timestamps)
  - can be characterised formally

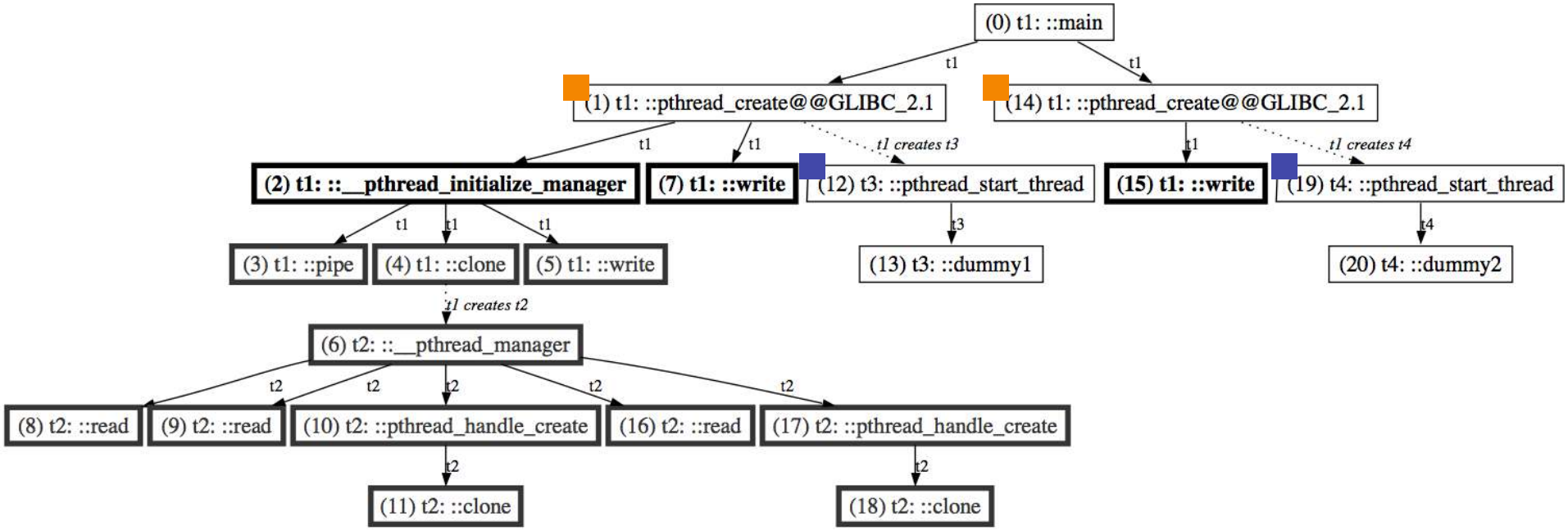
Revisiting pthread



`fuse ::pthread_create* ::pthread_start_thread* G`

Revisiting pthread

- ‘Reconstructing’ data-flow from temporal succession



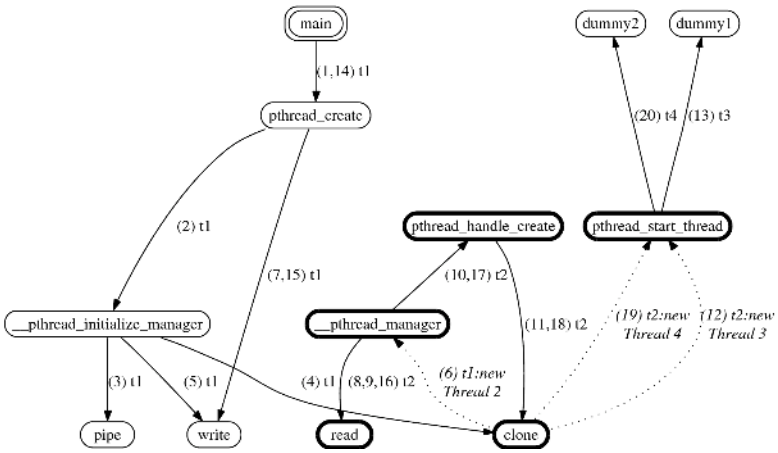

```
fuse ::pthread_create* ::pthread_start_thread* G
```

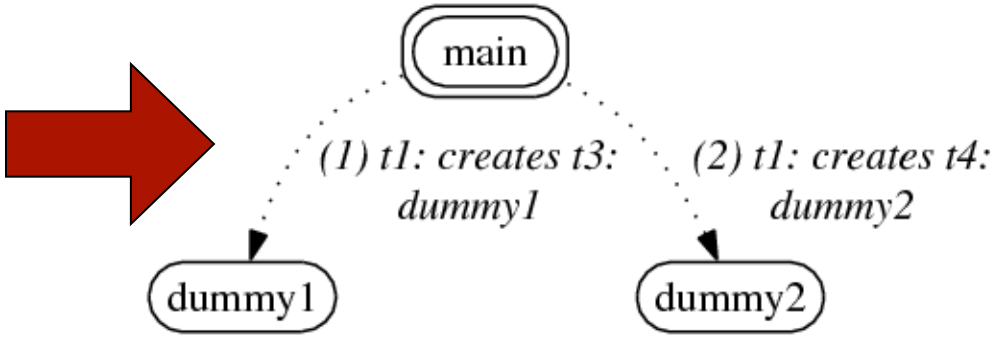
Revisiting pthread

```
fuse     ::pthread_create* ::pthread_start_thread* G
put      ::pthread_create* G CREATE
forwN    1 CREATE G
```

Generic script: applies to any pthread code

```
exclude CREATE G
absPatern ::pthread_create* G
absPatern ::pthread_start_thread* G
```





Case study

- 3 C/C++ industry-grade CORBA products
  - **ORBacus**, **omniORB**, **TAO**
- Set up
  - **~ 60 breakpoints** (locks, memory, threading, callbacks)
  - one ping-pong request
  - 1GHz Pentium III, Linux kernel 2.4, gdb 5.1-1
- Observation **overheads** (Orbacus)
  - non-instrumented: < 1s
  - fully instrumented: 1h 2m 11s
  - lock tracing disabled during initialisation: 4m 53s

Stack capture speedup

```
#0  0x405c8f20 in send () from /lib/libpthread.so.0
#1  0x4015804a in omni::tcpConnection::Send ()
#2  0x4011364c in omni::giopStream::sendChunk ()
[..]
#8  0x400bdb0a in omniAsyncWorker::run ()
#9  0x405abbdf in omni_thread_wrapper ()
#10 0x405c2bf0 in pthread_start_thread ()
#11 0x405c2c6f in pthread_start_thread_event ()
```

ORB	threads	traces	frames	invocations	invocations/traces
ORBACUS 4.1	8	658	9178	2066	3.13
OMNIORB 4	7	1828	16807	3088	1.68
TAO 1.2.1	6	512	11260	1352	2.64

breakpoint activation = cost

reconstructed tree = added value

ratio = speedup

(with lock tracing disabled during initialisation)