CosmOpen: Dynamic reverse engineering on a budget

François Taïani[1], Marc-Olivier Killijian[2], Jean-Charles Fabre[2]
[1]Lancaster University, UK
[2]LAAS-CNRS, France
(f.taiani@lancaster.ac.uk)

LANCASTER UNIVERSITY

InfoLab21

LAAS-CNRS

Let us inquire diligently

The best and safest method of philosophizing seems to be, first to ***inquire diligently into the properties of things****, and to establish those properties by* ***experiences*** *and then to proceed more slowly to hypotheses for the explanation of them.*

Isaac Newton

Lancaster Middleware Group

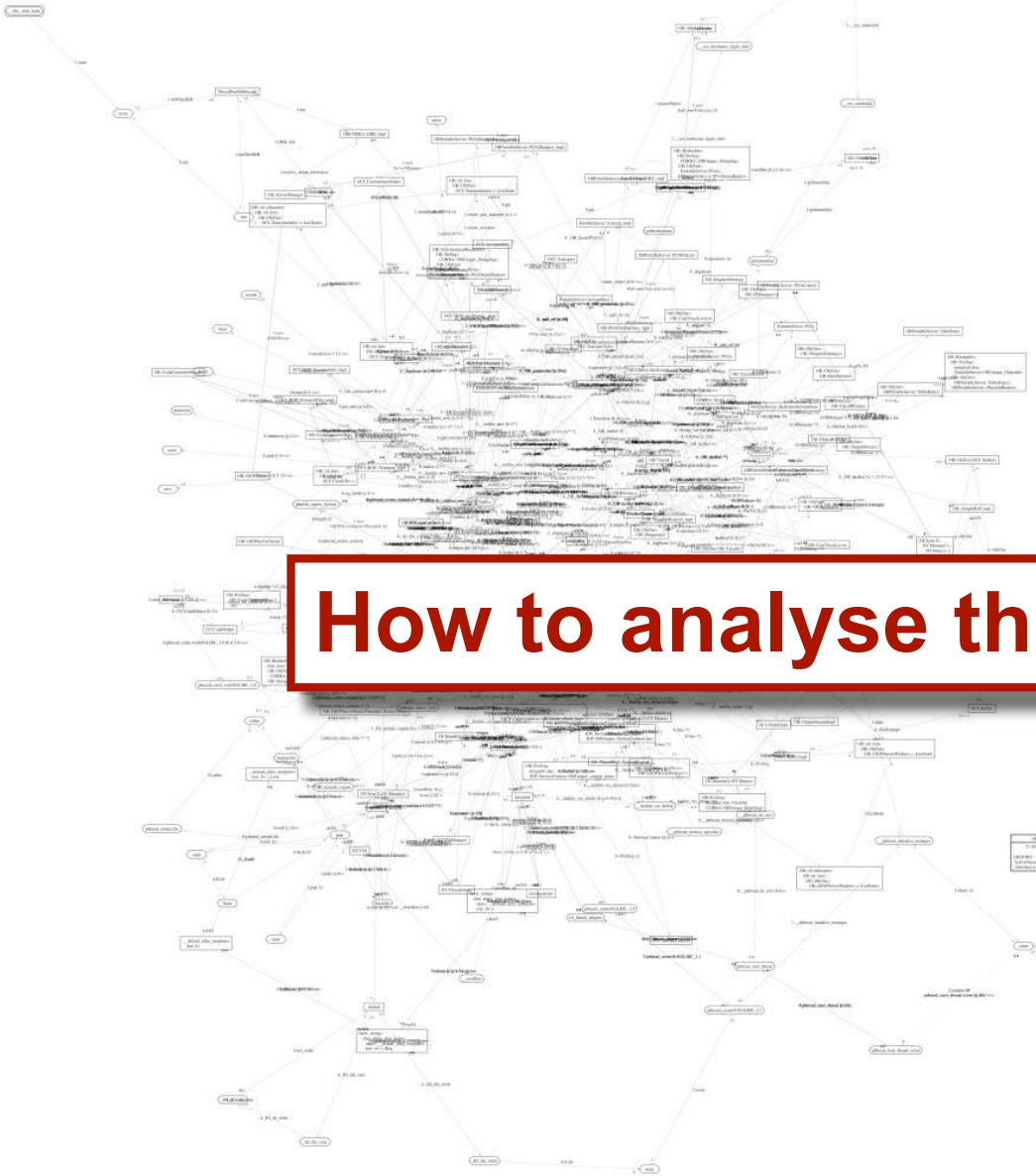
- Facilitate development of **new distributed systems**
  - Abstractions?
  - Mechanisms?
  - Exotic environments: MANETs, WSN, routers, …

Lancaster Middleware Group

- Facilitate development of **new distributed systems**
  - Abstractions?
  - Mechanisms?
  - Exotic environments: MANETs, WSN, routers, …

- Flip-side: study of **existing middleware**
  - Emergent structures?
  - Development practices?
  - How to ease refactoring / dissemination?

Example: one CORBA request



ORBacus Request Processing

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

How to analyse this?

Outline

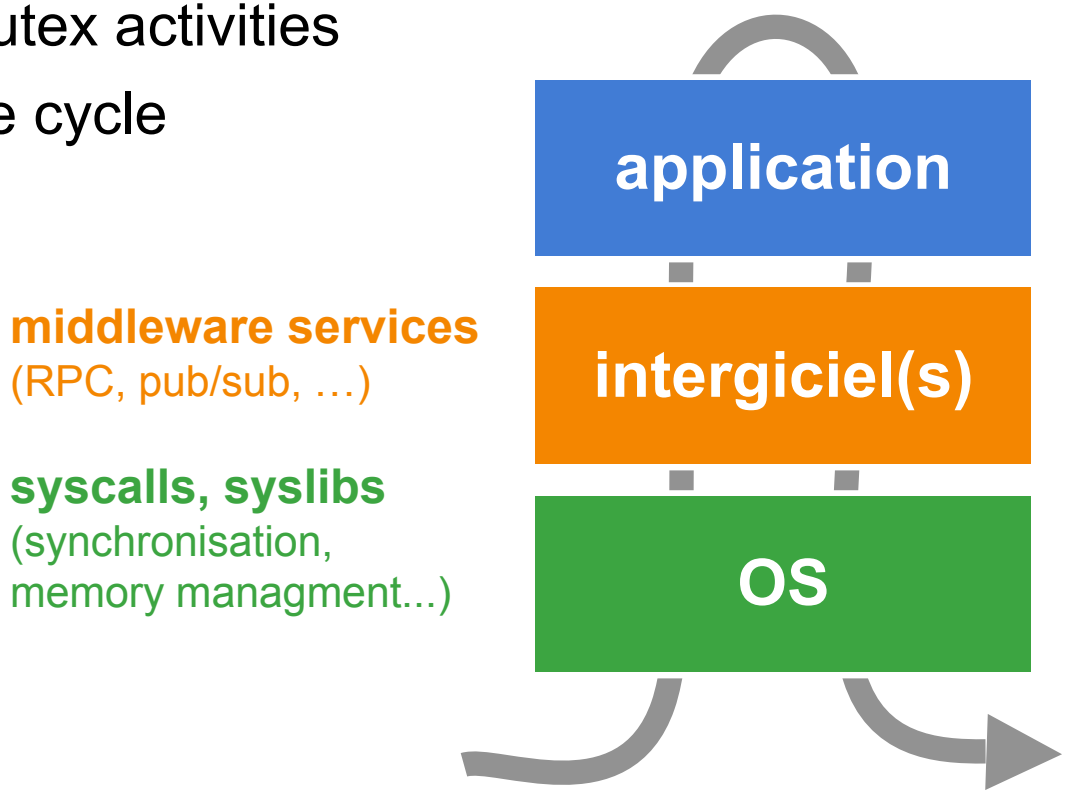
- **Why** reverse engineering middleware?
- Our approach: software "**interferometry**"
- Does it work? **A case study**

Outline

- **Why** reverse engineering middleware?
- Our approach: software "**interferometry**"
- Does it work? **A case study**

Why?

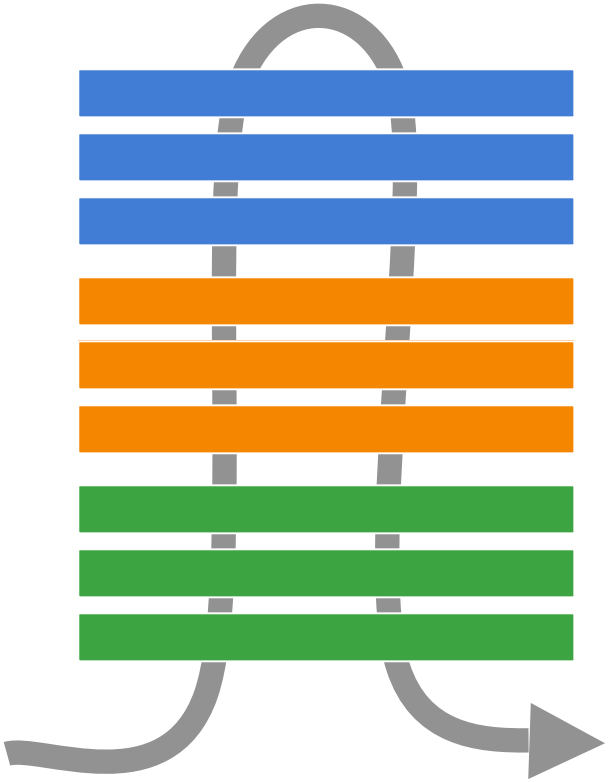
- Original goal
  - Control non-determinism in industry grade middleware
  - Observation of mutex activities
  - Link to request life cycle



Why?

- 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
  - **P1: Large behavioural data-set**
  - **P2: Cross-layer entangling**



P1: Large behavioural data-sets

- Example: **Globus**
- **Huge piece of** software (3.9.x)

  123,839 lines in Java (without reused libraries)

  (1,908,810 lines in C/C++, including reused libraries)
- **Many libraries** layered

  XML, WSDL (Descr. Lang), WSRF (Resource Fwork)

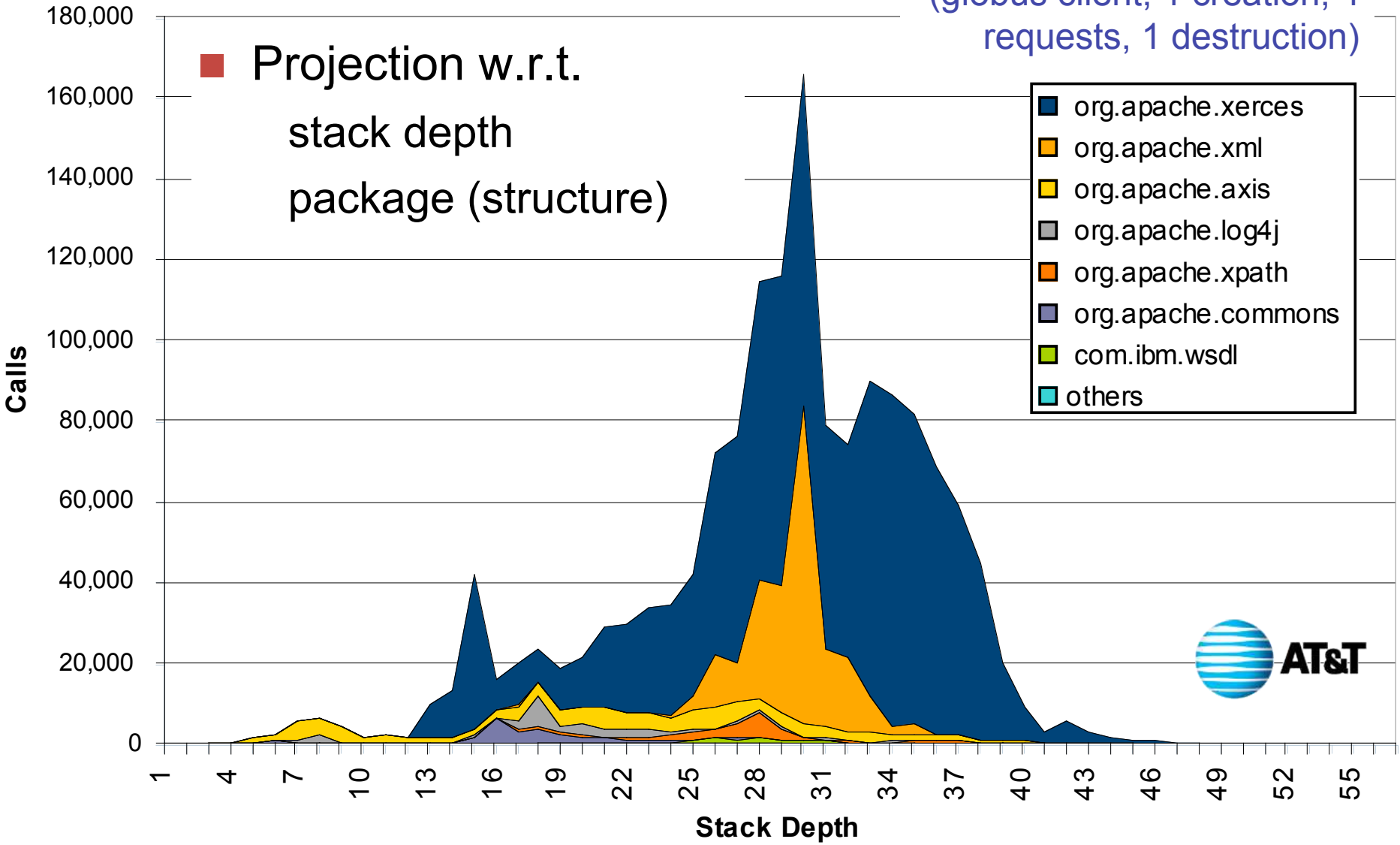
  Axis (SOAP), Xerces (XML Parsing), com.ibm.wsdl
- Java: **exhaustive tracing** (outside the JVM libs)

  client : **1,544,734** local method call (sic)

  server : **6,466,652** local method calls (sic) [+time out]

P1: Large behavioural data-sets

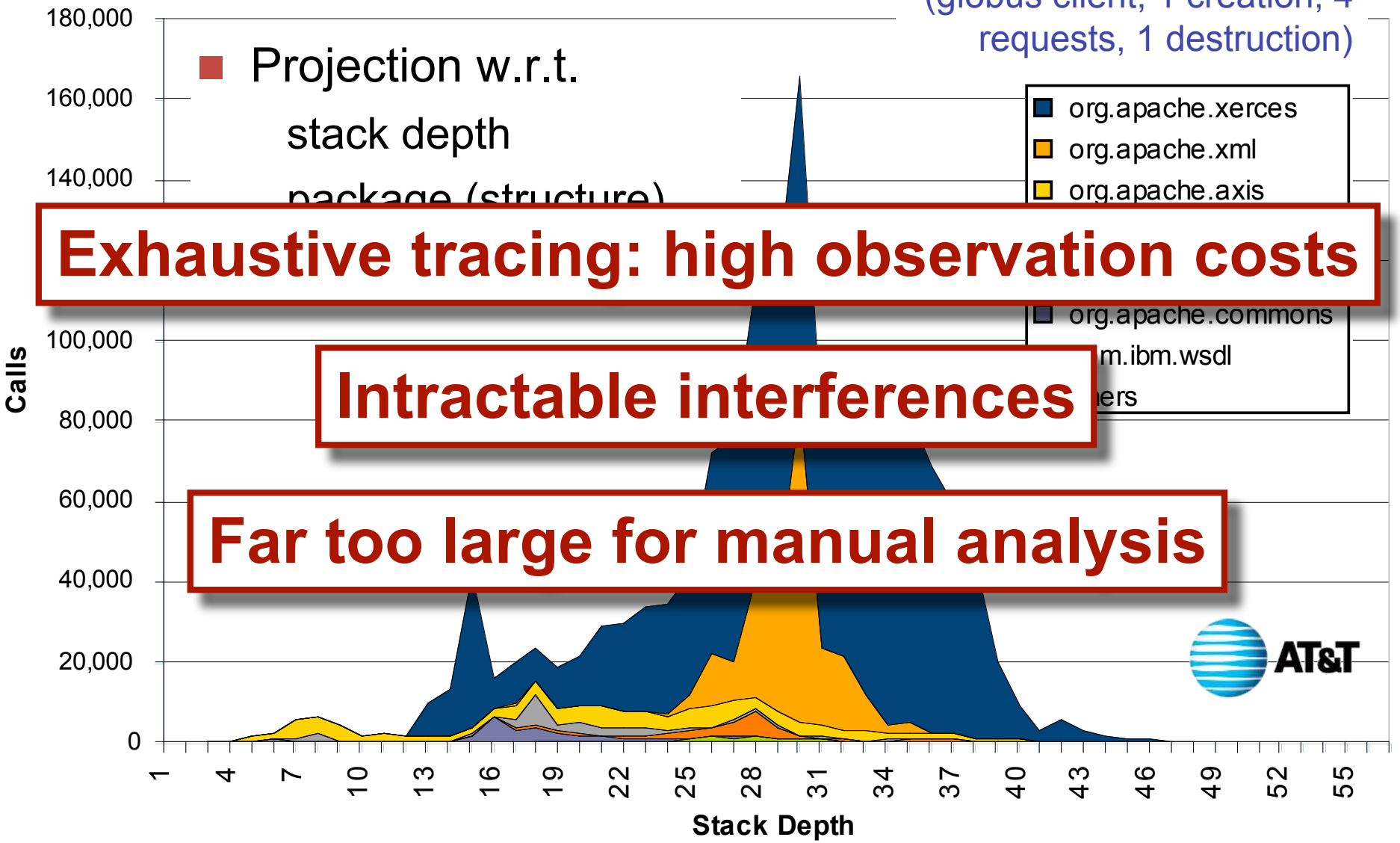
(globus client, 1 creation, 4 requests, 1 destruction)



P1: Large behavioural data-sets

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

- Projection w.r.t.
  - stack depth
  - package (structure)



Exhaustive tracing: high observation costs

Intractable interferences

Far too large for manual analysis

P2: Cross-layer entangling

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

P2: Cross-layer entangling

dummy1

dummy2

```
int main () {
   pthread_t threadN1, threadN2 ;
   pthread_create(&threadN1, NULL, dummy1, NULL) ;
   pthread_create(&threadN2, NULL, dummy2, NULL) ;
   pthread_join  ( thread1, NULL) ;
   pthread_join  ( thread2, NULL) ;
};
```

?

OS

observation by tracing

P2: Cross-layer entangling

dummy1

dummy2

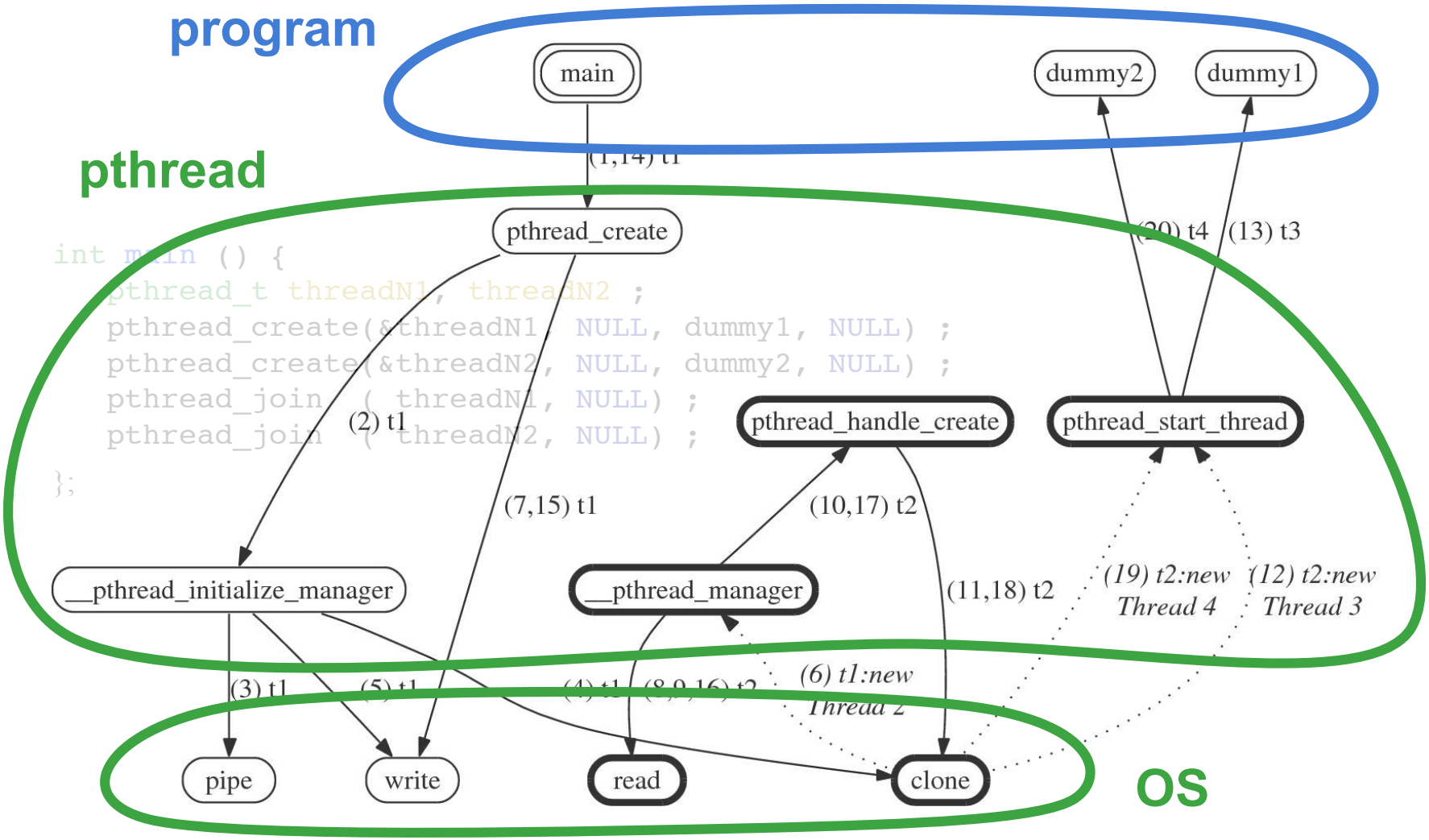
```
int main () {
   pthread_t threadN1, threadN2 ;
   pthread_create(&threadN1, NULL, dummy1, NULL) ;
   pthread_create(&threadN2, NULL, dummy2, NULL) ;

};
```

Can we reconstruct main's behaviour?

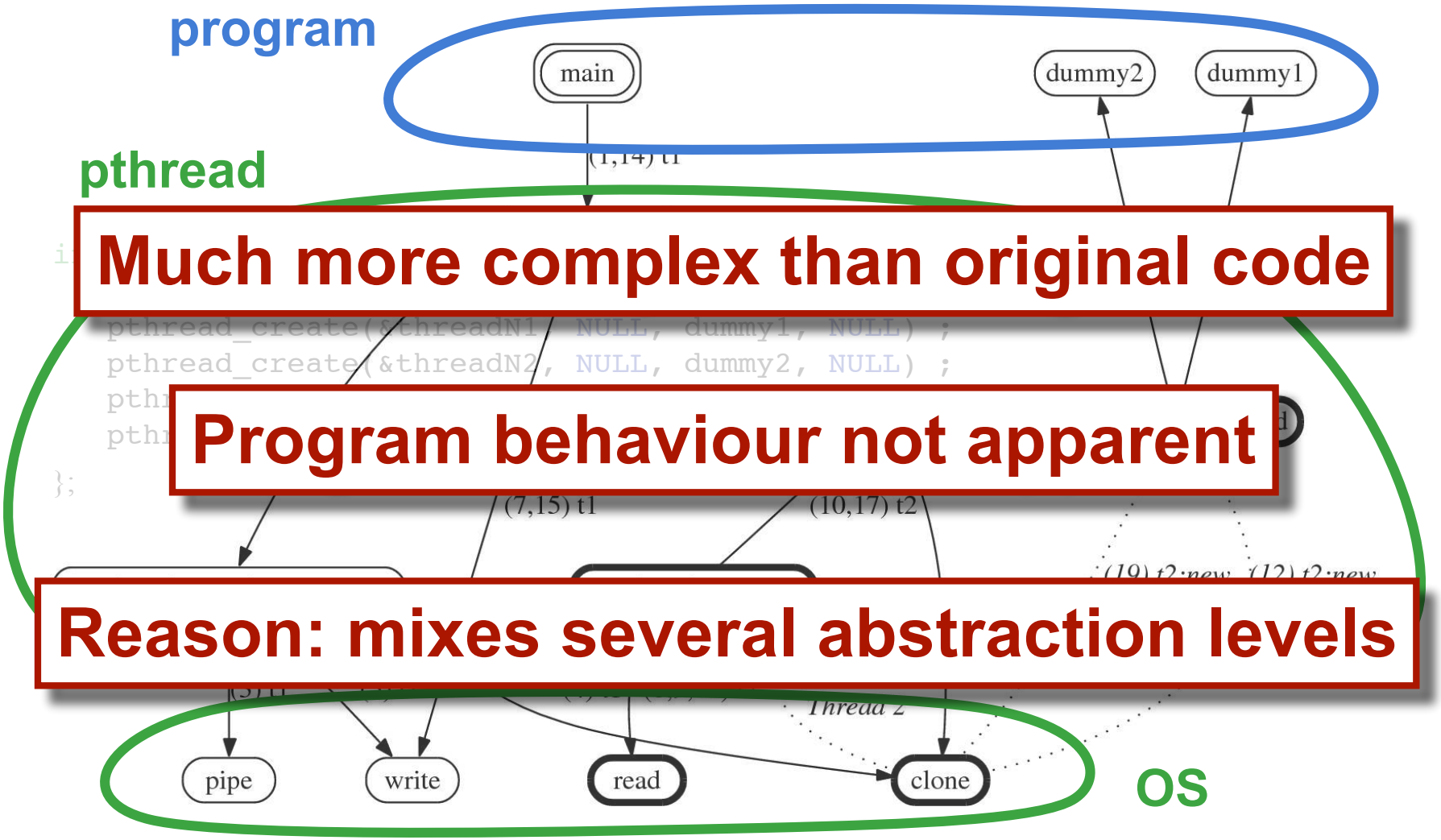
OS

observa by tracing

P2: Cross-layer entangling



P2: Cross-layer entangling



Outline

- **Why** reverse engineering middleware?
- Our approach: software "**interferometry**"
- Does it work? **A case study**

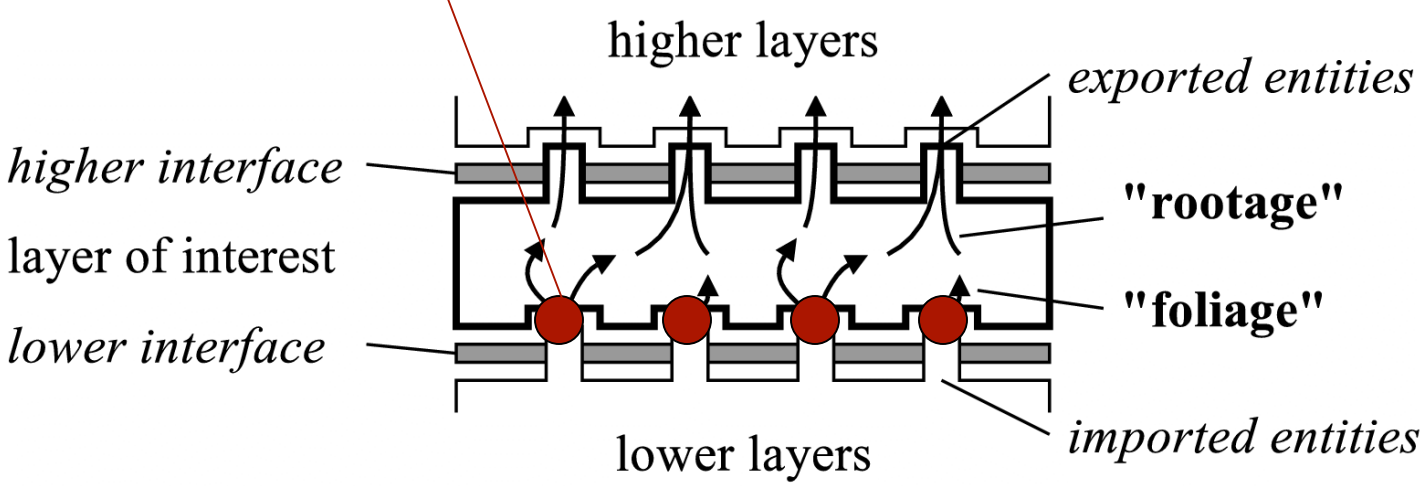
Approach: “interferometry”

- **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.

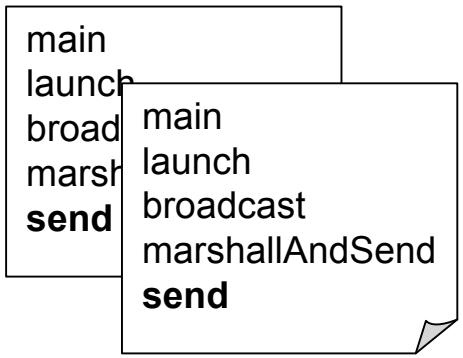
More for less

```
#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 ()
```

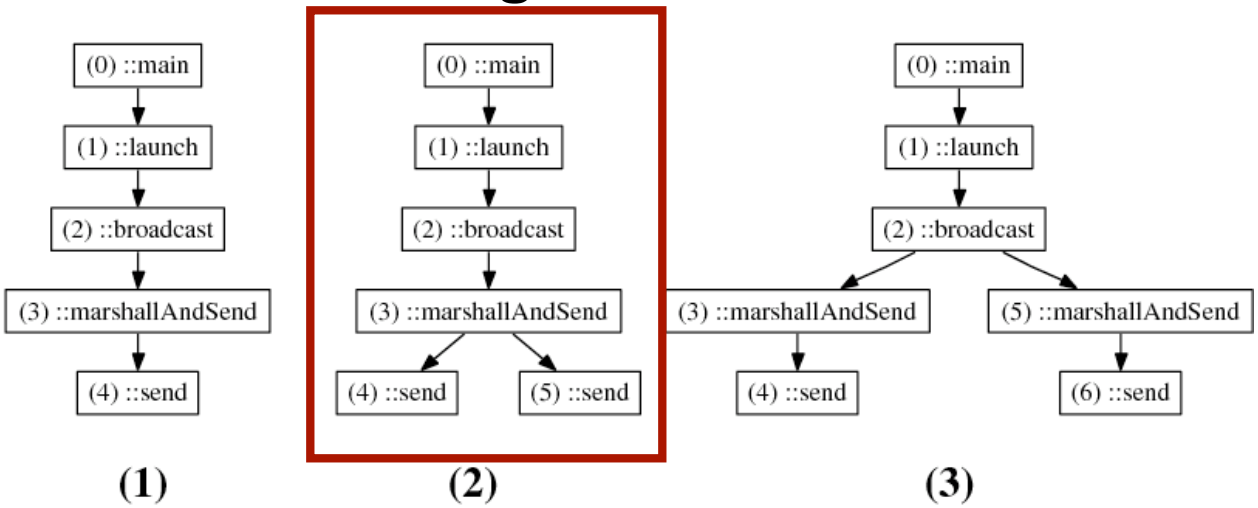


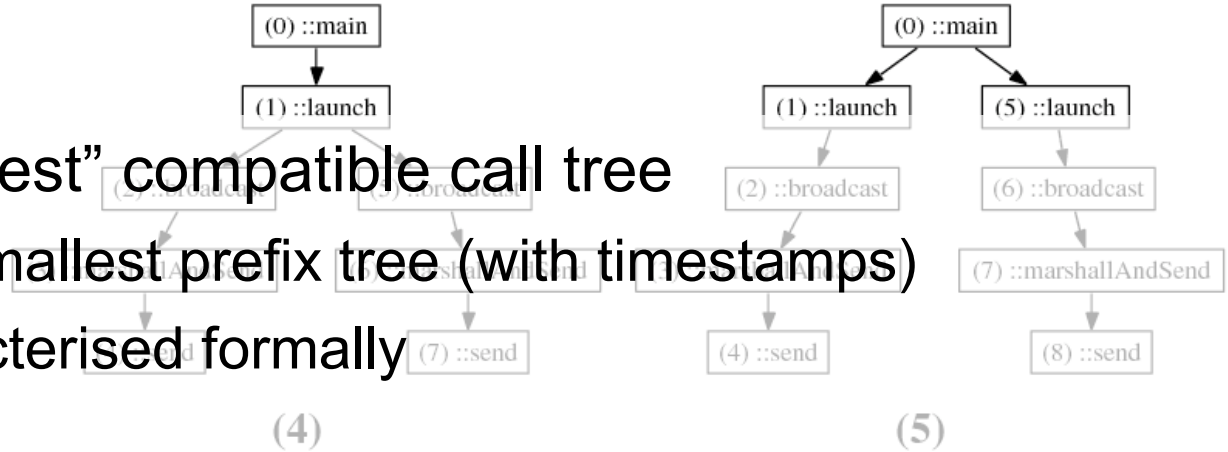
Reconstructing a call tree

- 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

Graph manipulation filters

Provides **variables** to store intermediary results

A number of graph **operators**

- **selection**

  `put ::pthread_create* G CREATE`

- **recursive extension**

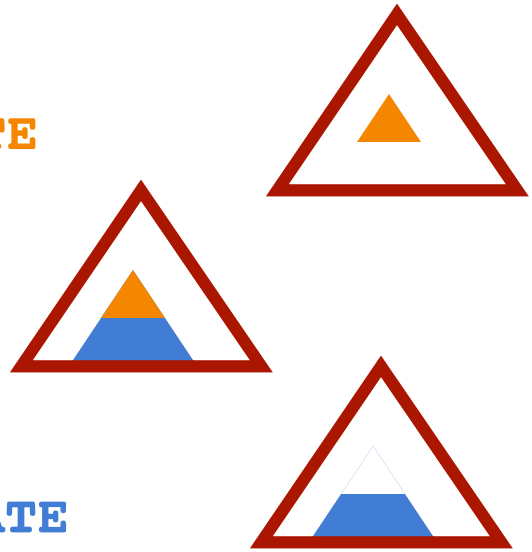
  `forward CREATE G`

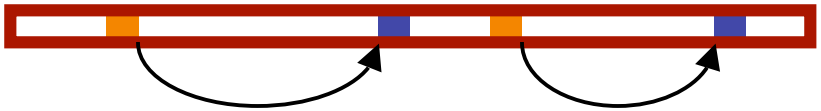
- **boolean algebra**

  `remove ::pthread_create* CREATE`

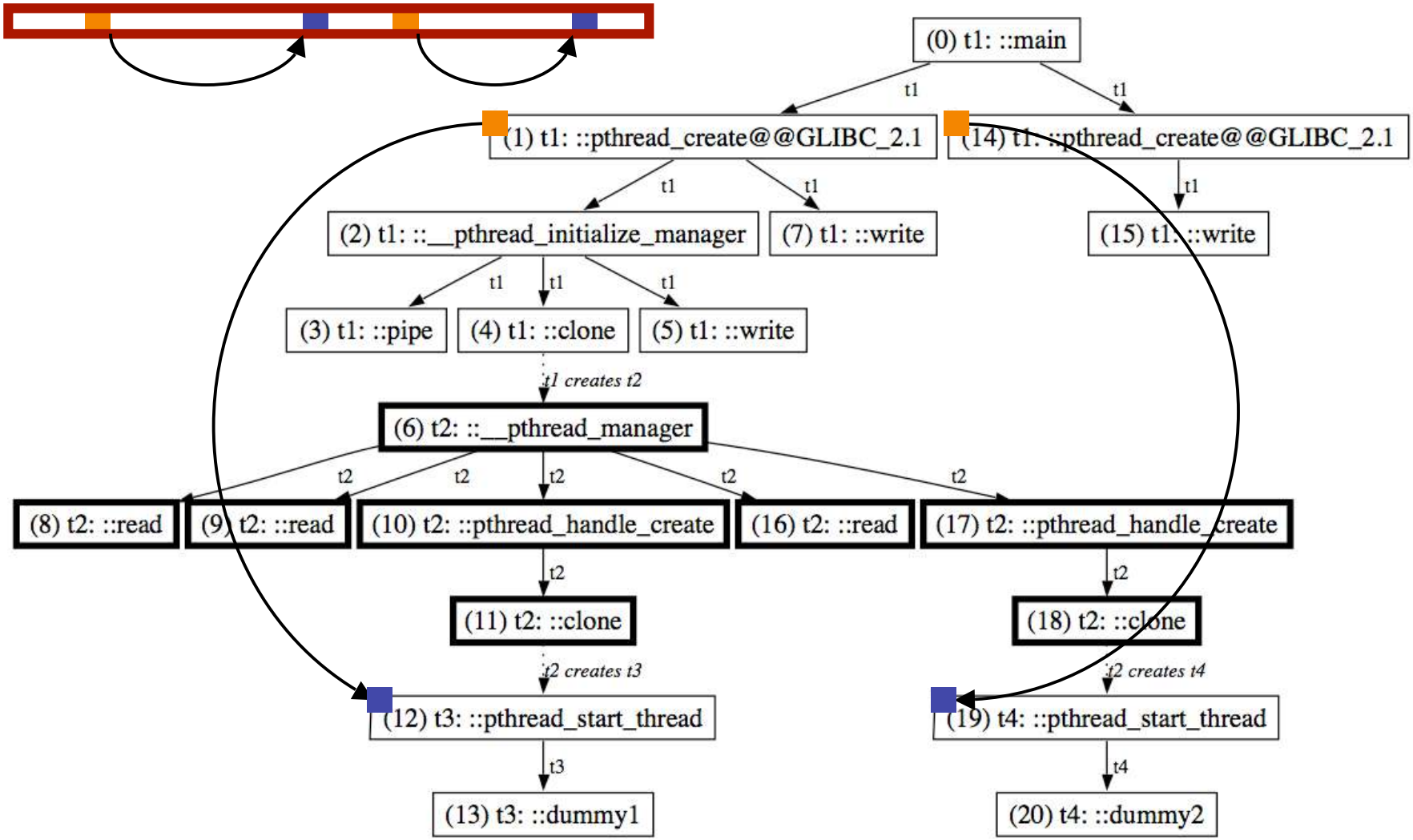
- **temporal operations**

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





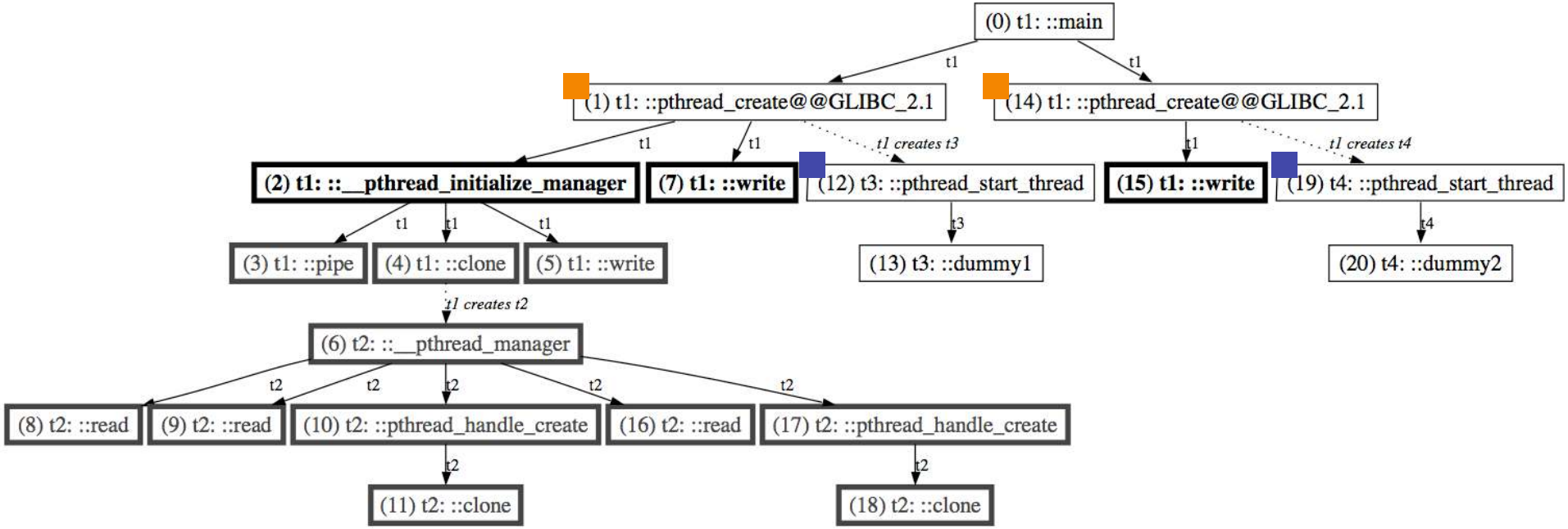
Revisiting pthread




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

Revisiting pthread

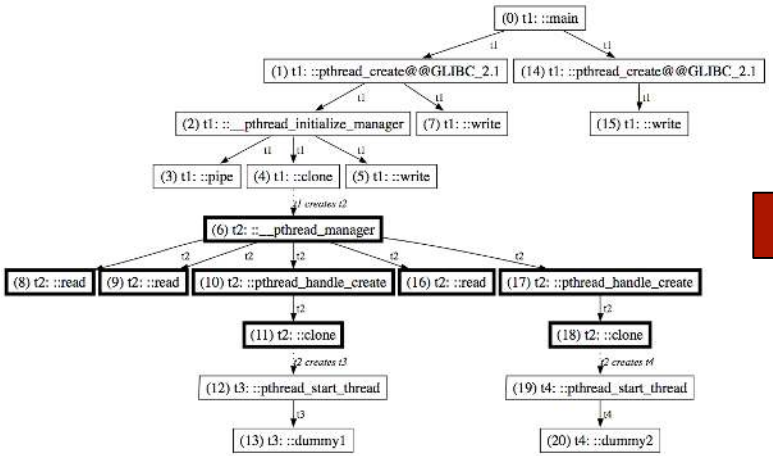
- ‘Reconstructing’ data-flow from temporal succession

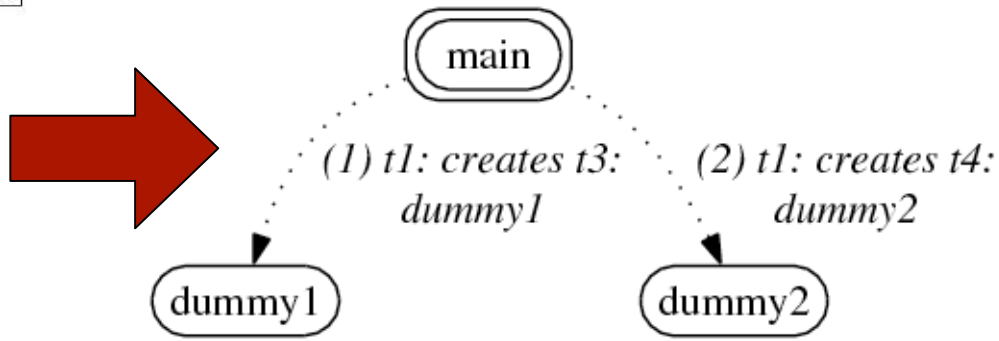



```
fuse ::pthread_create* ::pthread_start_thread* G
```

Revisiting pthread

```
fuse     ::pthread_create* ::pthread_start_thread* G
put      ::pthread_create* G CREATE
forwN    1 CREATE G
remove   ::pthread_create* CREATE
remove   ::pthread_start_thread* CREATE
forward CREATE G
exclude CREATE G
absPatern ::pthread_create* G
absPatern ::pthread_start_thread* G
```



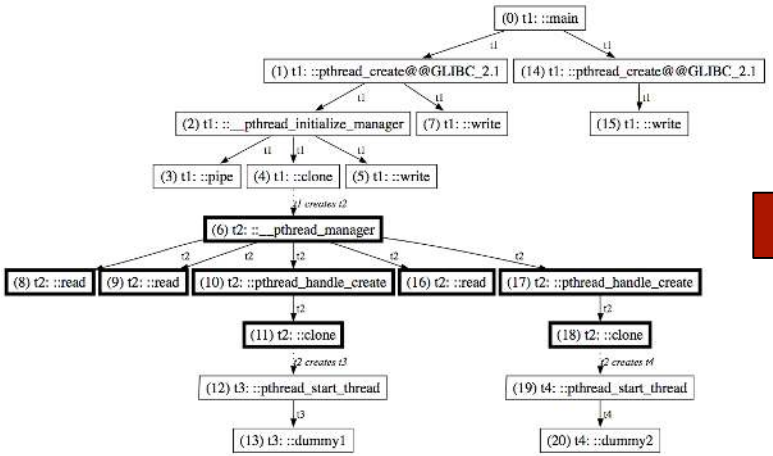


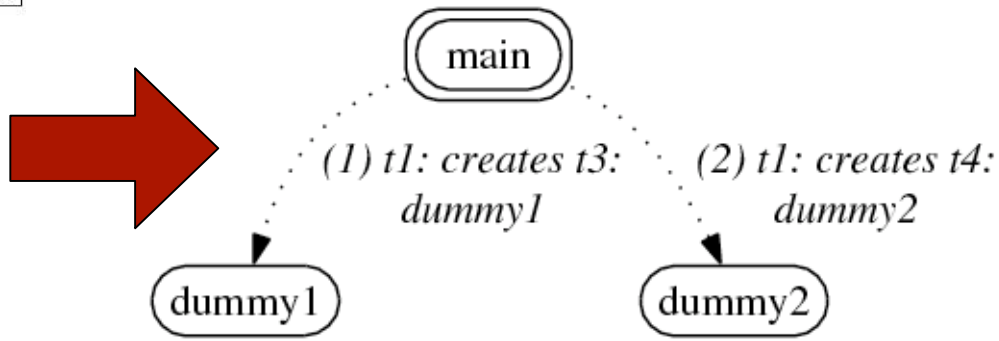
Revisiting pthread

```
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

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

Outline

- **Why** reverse engineering middleware?
- Our approach: software "**interferometry**"
- Does it work? **A case study**

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

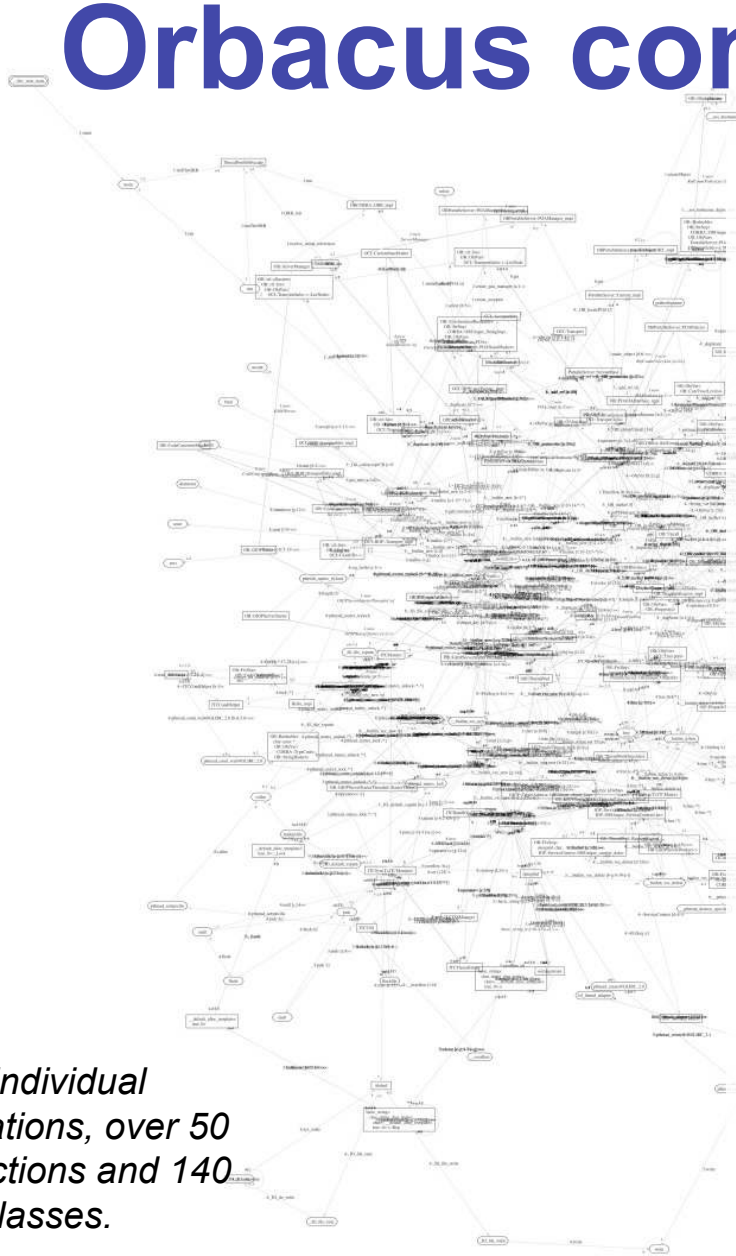
breakpoint activation = cost

reconstructed tree = added value

ratio = speedup

(with lock tracing disabled during initialisation)

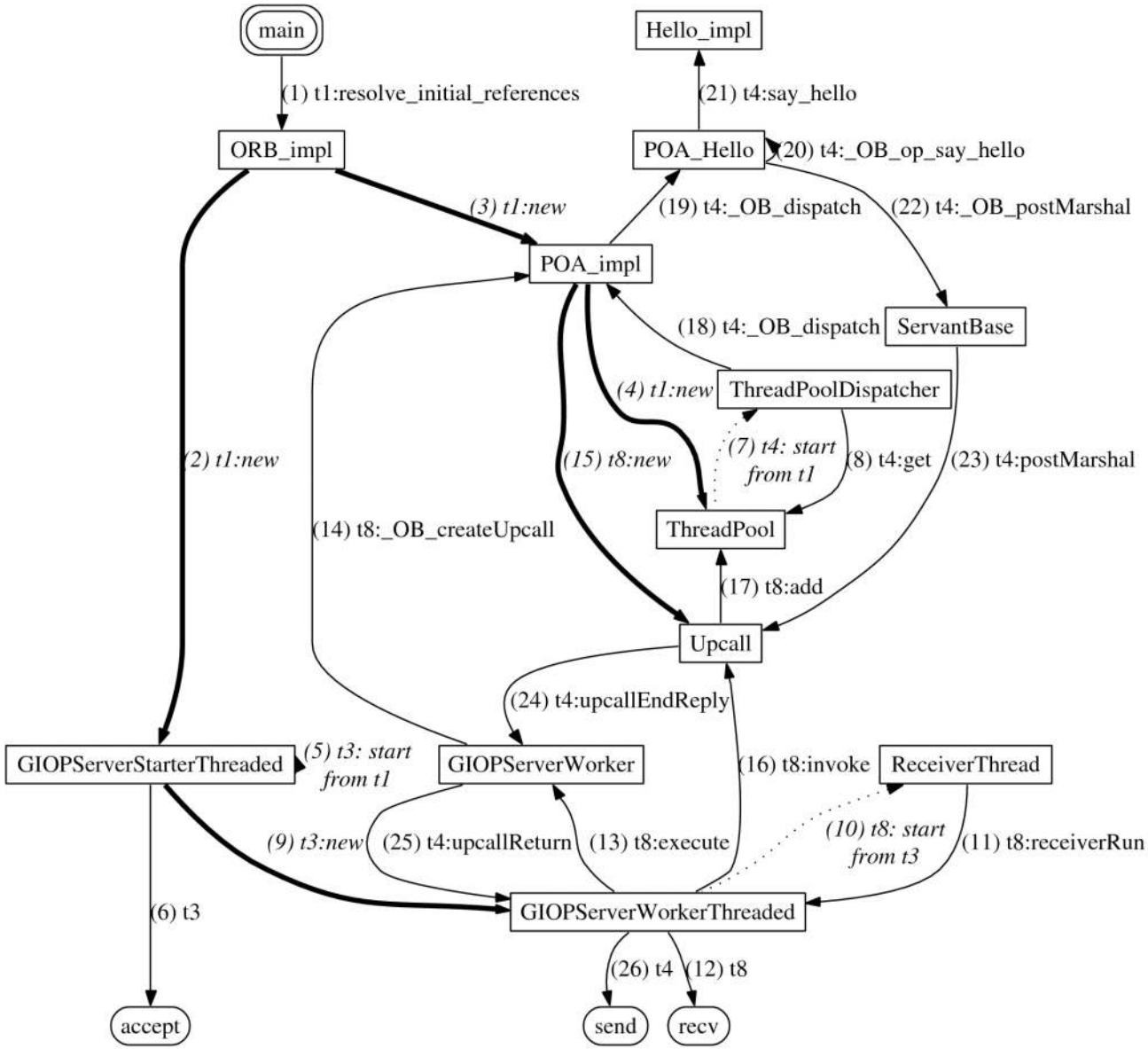
Orbacus complete graph



```
// [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
```

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

Result

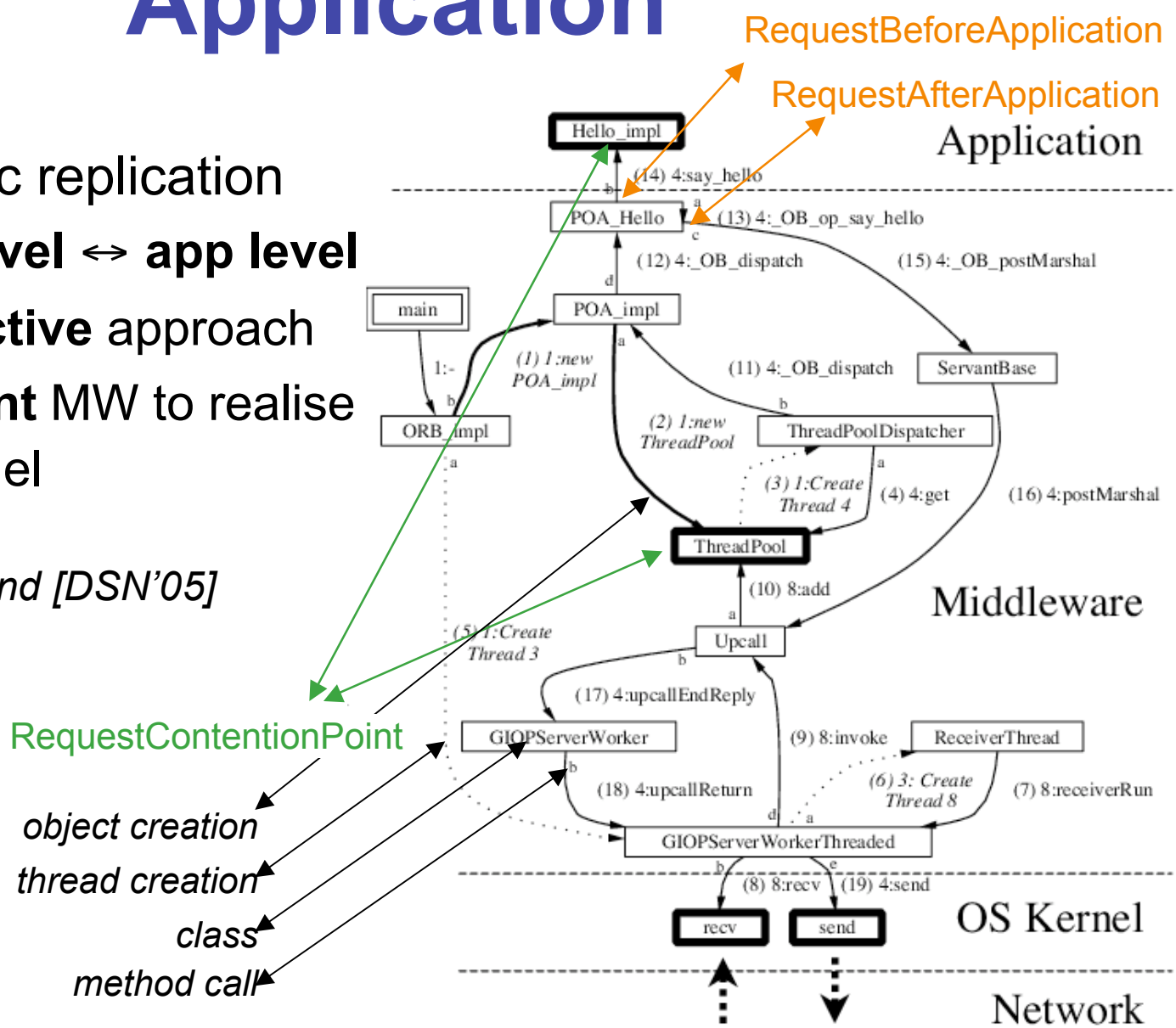


26 individual invocations, 4 C functions, and 23 C++ classes.

Application

- Deterministic replication
  - link **OS level** ↔ **app level**
  - use **reflective** approach
  - **instrument** MW to realise meta-model

ref: *[DSN'03] and [DSN'05]*



Conclusion

- **Pragmatic** approach to dynamic reverse engineering
  - addresses complex layered software
  - minimises interferences by using stack traces
  - reconstruction and graph manipulation for analysis
- Caveat
  - reverse engineering remains a **human** activity
  - constructed graphs provide a roadmap / guide
  - but must be supported by other activities
- Future
  - better analysis of help provided (**user studies**)
  - better formalisation of **consistency** in graph manipulation

(Some) references

- François Taïani, Marc-Olivier Killijian, Jean-Charles Fabre, CosmOpen: Dynamic reverse-engineering on a budget. Software: Practice and Experience, to appear [contact me for most recent version]
- 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
- 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

(Some) references (cont)

- Zaidman A, Calders T, Demeyer S, Paredaens J. Applying Webmining Techniques to Execution Traces to Support the Program Comprehension Process. In Proc. of the Ninth European Conf. on Software Maintenance and Reengineering (CSMR'2005). IEEE Computer Society, 2005; 134-142. DOI: 10.1109/CSMR.2005.12
- 39. 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
- http://ftaiani.ouvaton.org/7-software/

Thank you

Questions?