Towards Anomaly Comprehension

Using Structural Compression to Navigate Profiling Call-Trees

Shen Lin, François Taïani,
Thomas C. Ormerod, Linden J. Ball

Lancaster University (UK)

InfoLab21
School of Computing and Communications

Why visualise anomalies?

- Software easier to construct than to **understand**
  - ➔ as climbing up vs. climbing down
- Particularly true for **emergent dynamic behaviours**
  - ➔ e.g. dependability and **performance**
  - ➔ arise from interactions within/ without target system
  - ➔ phenomena might span large set of components
  - ➔ at different levels of abstraction
- **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: Performance

impossible to vary level of abstraction (methods)

impossible to abstract locally (methods)

Nodes	Executi...	Percentage	Percentage
sun.reflect.DelegatingMethodAccessorImpl.invoke	67	58.77%	59%
sun.reflect.NativeMethodAccessorImpl.invoke	67	58.77%	59%
sun.reflect.NativeMethodAccessorImpl.invoke0	67	58.77%	59%
org.globus.wsrf.container.ServiceContainer.main	64	56.14%	56%
org.globus.wsrf.container.ServiceContainer.startSecurityContainer	52	45.61%	46%
org.globus.wsrf.container.ServiceContainer.createContainer	52	45.61%	46%
java.lang.reflect.Constructor.newInstance	49	42.98%	43%
sun.reflect.DelegatingConstructorAccessorImpl.newInstance	49	42.98%	43%
sun.reflect.NativeConstructorAccessorImpl.newInstance	49	42.98%	43%
sun.reflect.NativeConstructorAccessorImpl.newInsta...	49	42.98%	43%
java.lang.Class.forName	3	2.63%	3%
org.globus.wsrf.container.ServiceContainer.listServices	12	10.53%	11%
org.globus.wsrf.container.ServiceContainer.<clinit>	3	2.63%	3%
org.globus.wsrf.container.ServiceThread.run	47	41.23%	41%
org.globus.wsrf.container.GSIServiceThread.process	47	41.23%	41%
org.globus.gsi.gssapi.net.GssSocket.getOutputStream	33	28.95%	29%
org.globus.gsi.gssapi.net.GssSocket.startHandshake	33	28.95%	29%
org.globus.gsi.gssapi.net.GssSocket.authenticateServer	33	28.95%	29%
org.globus.gsi.gssapi.GlobusGSSContextImpl.acceptSecContext	20	17.54%	18%
COM.claymoresystems.ptls.SSLHandshake.processHandshake	14	12.28%	12%
org.globus.gsi.gssapi.GlobusGSSContextImpl.verifyChain	3	2.63%	3%
org.globus.gsi.gssapi.GlobusGSSContextImpl.setCredential	2	1.75%	2%
org.globus.gsi.gssapi.GlobusGSSContextImpl.init	1	0.88%	1%
org.globus.gsi.gssapi.net.impl.GSIGssSocket.readToken	13	11.40%	11%
org.globus.wsrf.container.ServiceThread.process	14	12.28%	12%

difficult to relate different parts of the data

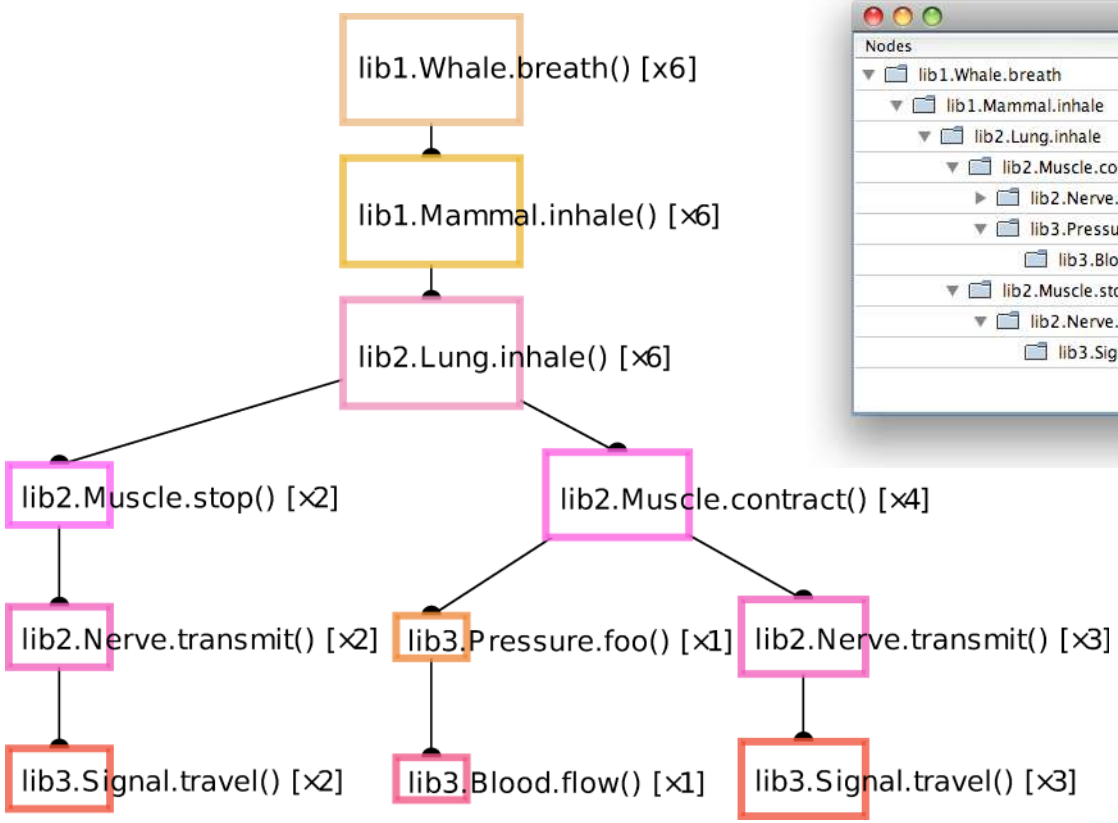
- Piece of grid middleware (Globus ws-core)

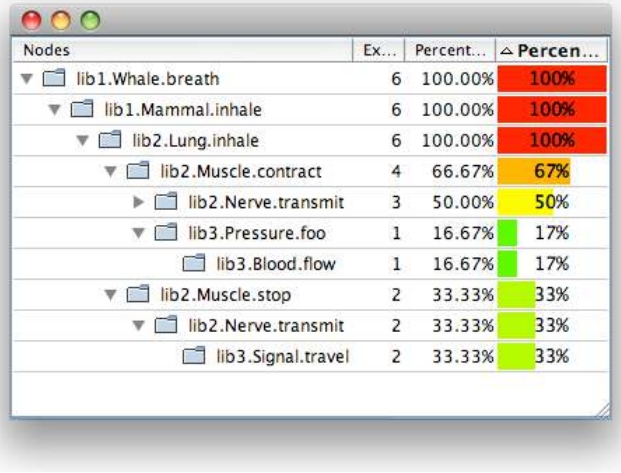
Visualising Dynamic Data

- 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 take in this work
  - ➔ 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**

Dynamic Profiling Data

- Toy example: weighted call tree (sample based profiling)

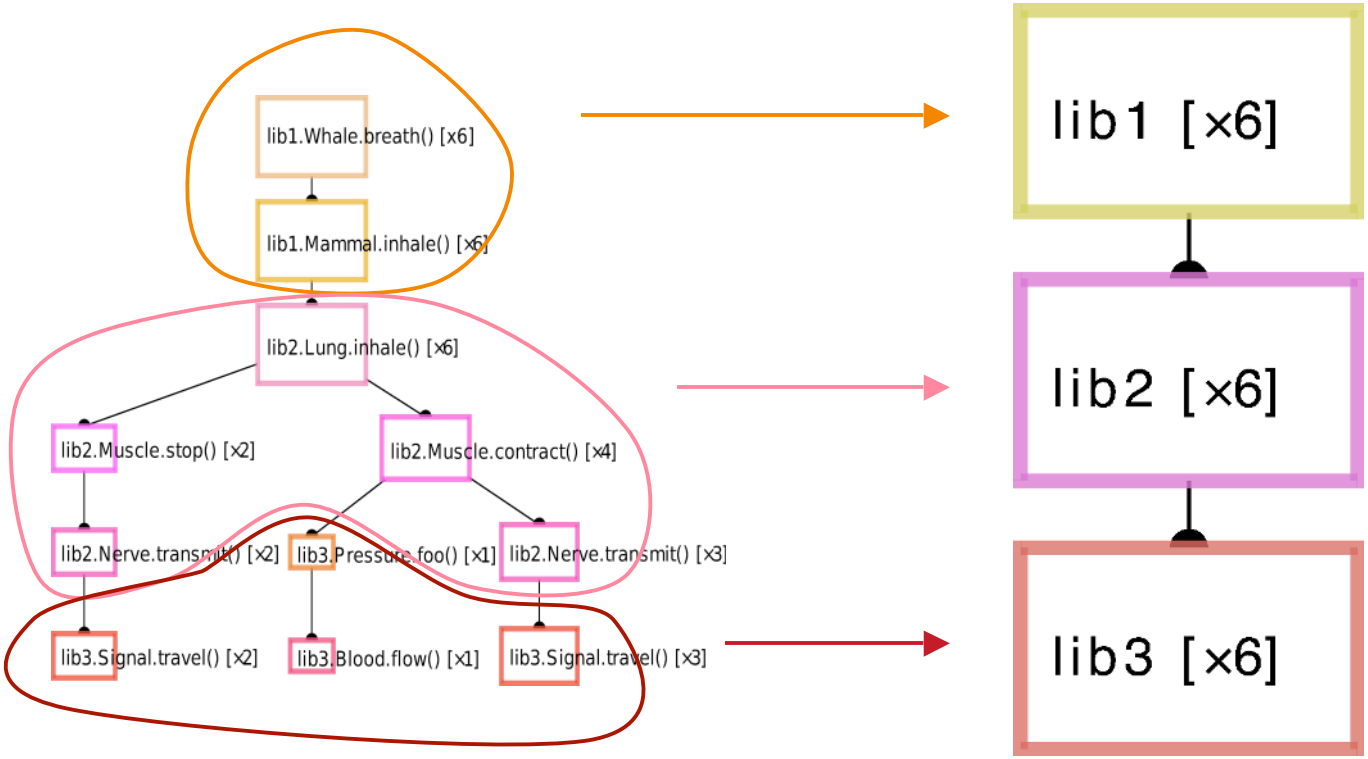




Nodes	Ex...	Percent...	Percen...
lib1.Whale.breath	6	100.00%	100%
lib1.Mammal.inhale	6	100.00%	100%
lib2.Lung.inhale	6	100.00%	100%
lib2.Muscle.contract	4	66.67%	67%
lib2.Nerve.transmit	3	50.00%	50%
lib3.Pressure.foo	1	16.67%	17%
lib3.Blood.flow	1	16.67%	17%
lib2.Muscle.stop	2	33.33%	33%
lib2.Nerve.transmit	2	33.33%	33%
lib3.Signal.travel	2	33.33%	33%

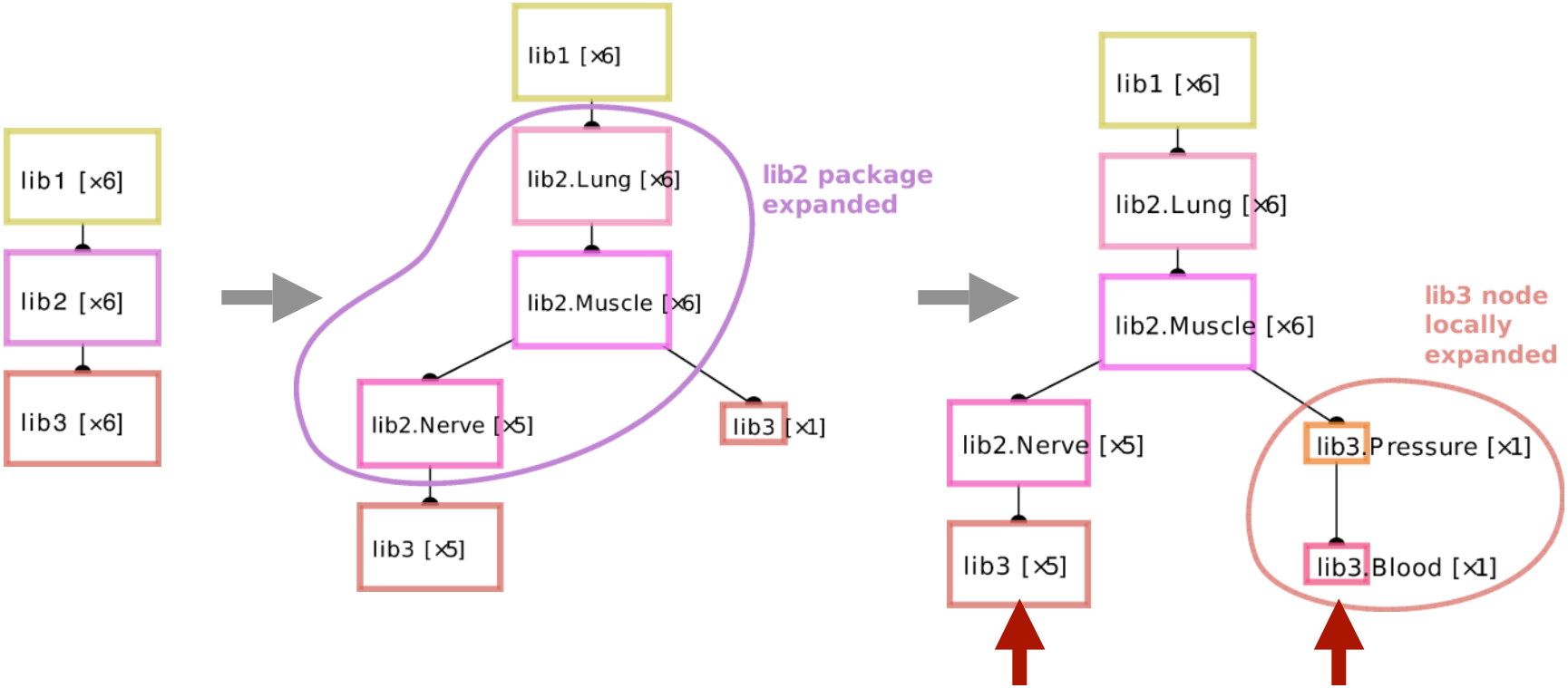
Full compaction

- 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

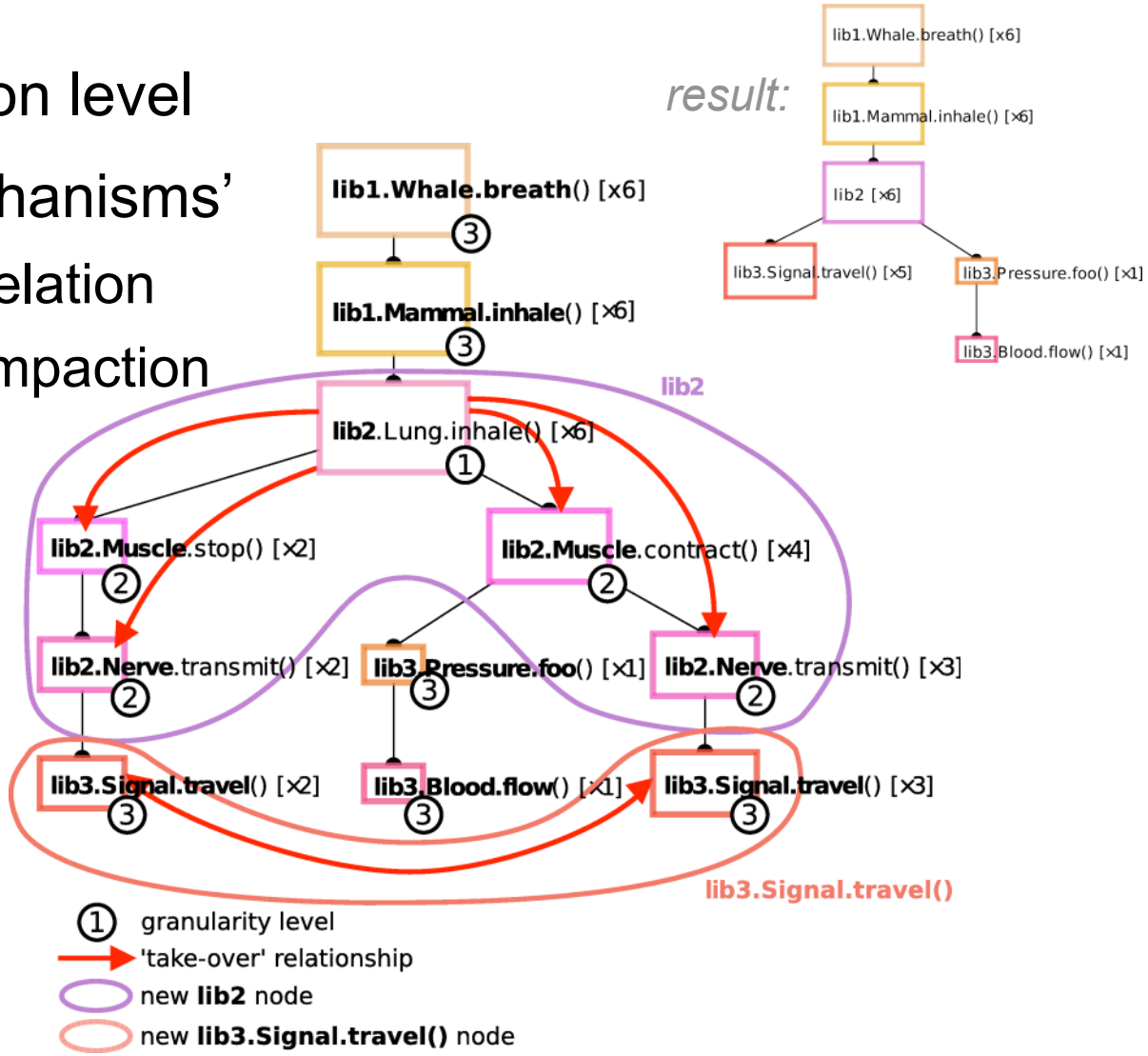
Realisation

- Uniform compaction reasonably well-understood
  - ➔ e.g. merge all ‘org.globus.xx’ nodes together
  - ➔ used by other tools (e.g. Creole for static analysis)
- **Challenges** here:
  - ➔ same entity → different compaction in different areas
  - ➔ but uniform compaction locally (otherwise confusing)
- **Question**
  - ➔ How do we encode these different ‘localised’ levels?
  - ➔ Which mechanisms to drive ‘local’ compaction?

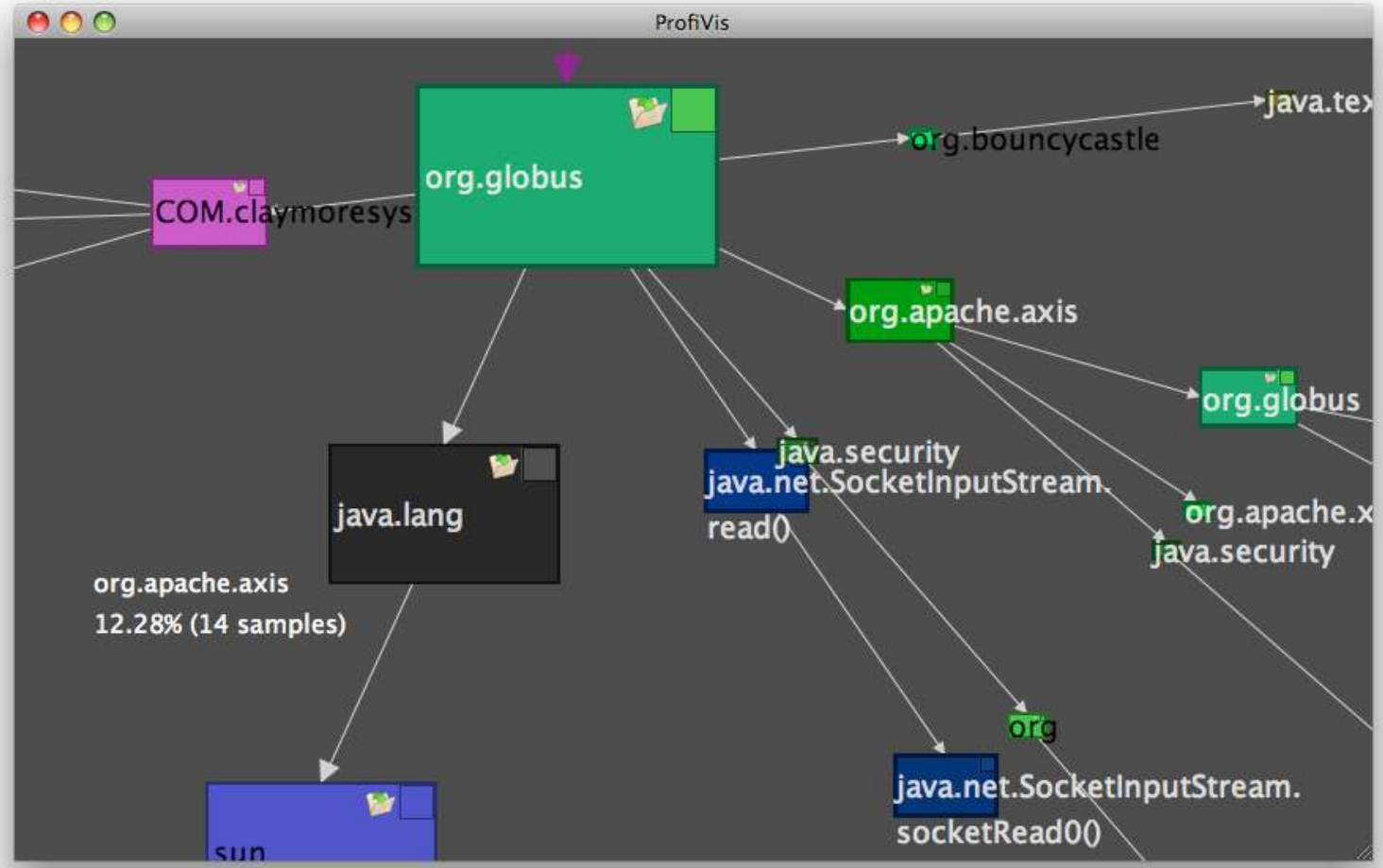
Compaction Algorithm

- Local compaction level
- 'Take over mechanisms'
  - ➔ use 'vicinity' relation
  - ➔ cascading compaction

More details in paper

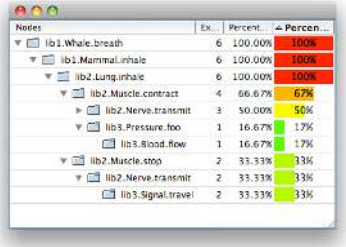


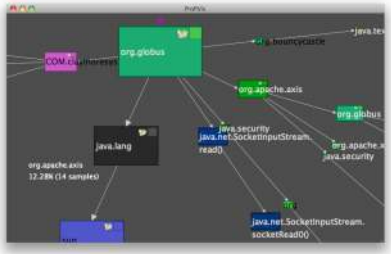
Demo


ProfiVis
org.globus
COM.claymoresys
org.bouncycastle
java.tex
org.apache.axis
org.globus
java.security
java.net.SocketInputStream.
read()
java.lang
org.apache.x
java.security
org.apache.axis
12.28% (14 samples)
org
java.net.SocketInputStream.
socketRead0()
sun

Evaluation

- Design
  - small scale and **exploratory** (4 users)
  - 2 categories of programs (‘**small**’ and ‘**large**’)
  - two tools: **ProfVis** and **Treetable**
  - each user applied both tools to both categories
  - task: identify performance issues

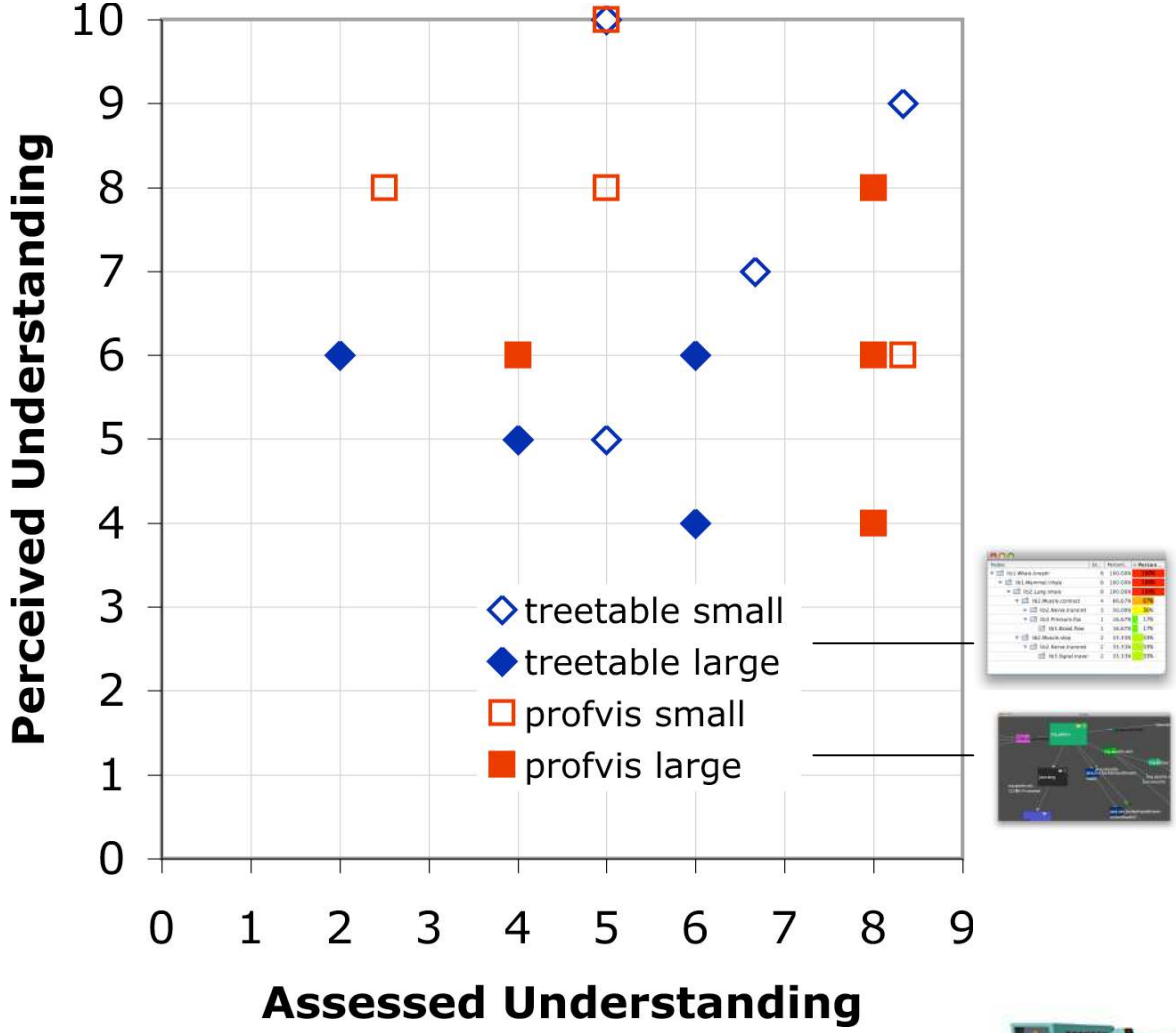




- Two quantitative measures
  - perceived understanding (from test user)
  - assessed understanding (from us)

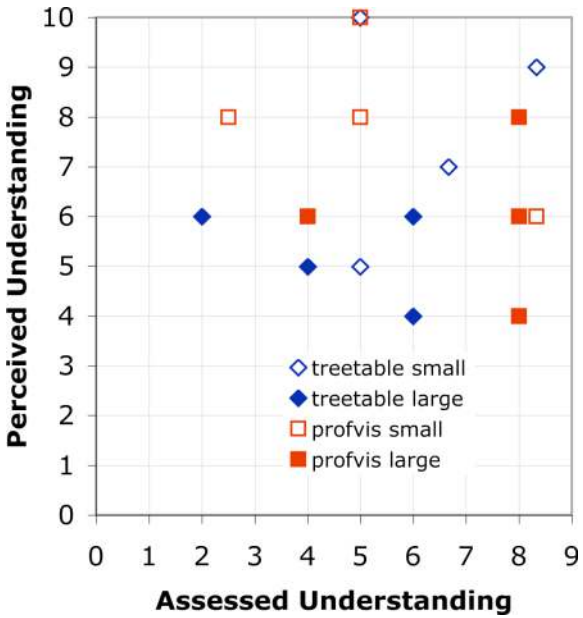
	LoC	classes	methods	prof. tree
BubbleSort	59	2	7	100
Simulation	140	5	16	71
OPSBrowser	13624	172	1002	1059
ws-core-3.9.4	42477	432	2550	1341

Results


Perceived Understanding
Assessed Understanding
0
1
2
3
4
5
6
7
8
9
10
treetable small
treetable large
profvis small
profvis large

Results (2)

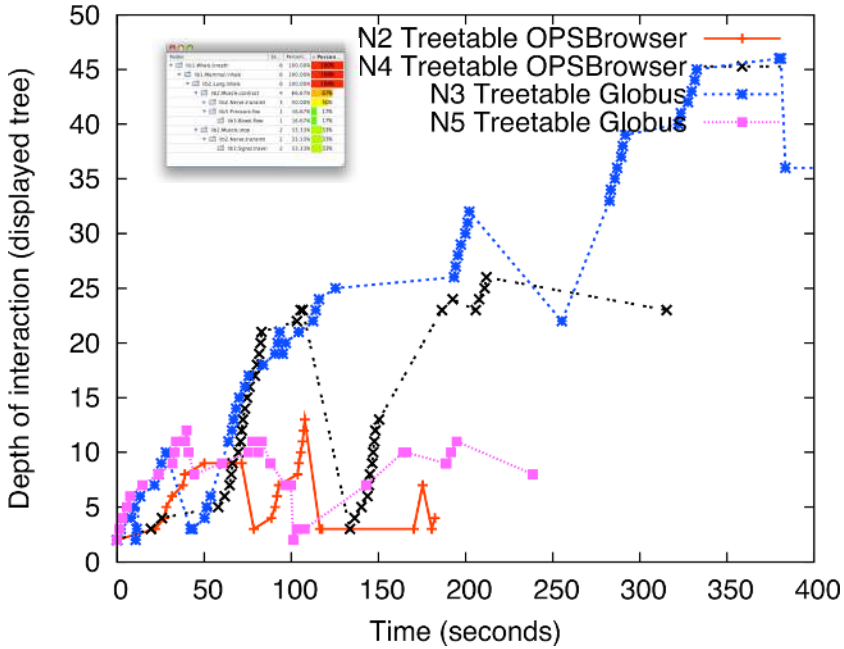
- Users seem
  - to **overestimate** their understanding on large programs with TreeTable on small programs with ProfVis
  - to **underestimate** their understanding on large program with ProfVis

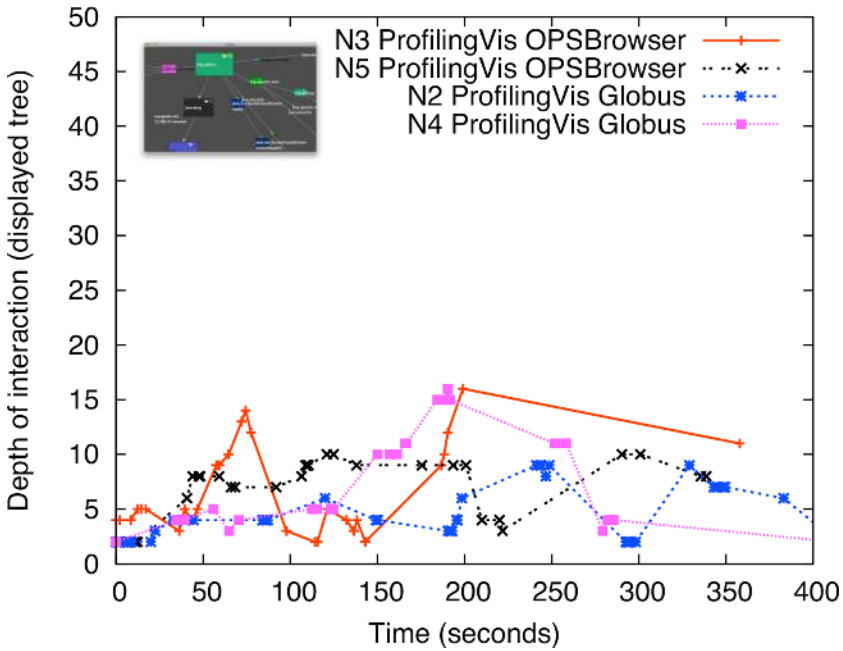


- Possible explanations
  - TreeTable hides full scope while ProfVis does not
  - As a result possible 'false sense of mastery'

Results (3)

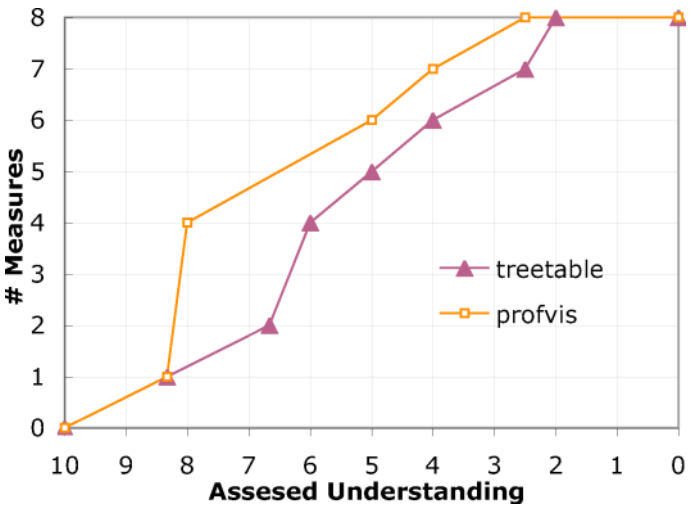
- Usage patterns seem to support this interpretation
  - ➔ users go deep w/ TreeTable, tend to hover w/ ProfVis

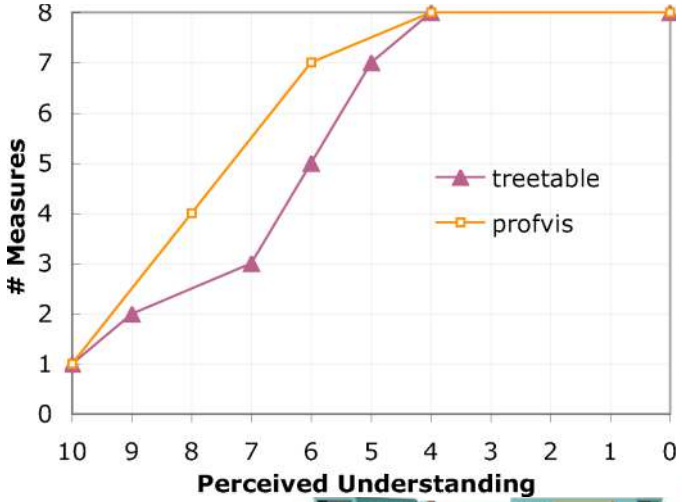




Results (4)

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





Conclusion

- A novel **localised structural compaction** algorithm
  - ➔ applied to performance profiling traces
- Preliminary **evaluation**
  - ➔ potential gap btw. user perception and their actual perf.
  - ➔ and btw. subjective preference and actual perf.
  - ➔ impact of presentation on search strategy
- **Future** work
  - ➔ larger study!
  - ➔ combine design dimensions (e.g. text + compaction)
  - ➔ explore approaches to avoid ‘false sense of mastery’