# Visualizing Dynamic Metrics with Profiling Blueprints

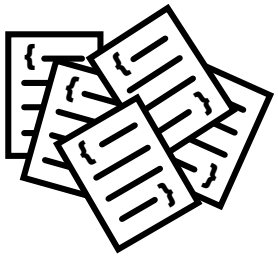
*Alexandre Bergel, Romain Robbes, Walter Binder*

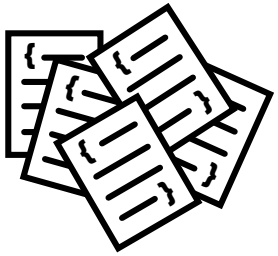
University of Chile
Chile

University of Lugano

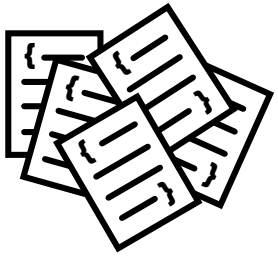
"*software profiling* is the investigation of a *program's behavior* using information gathered as the program *executes*"

- Wikipedia

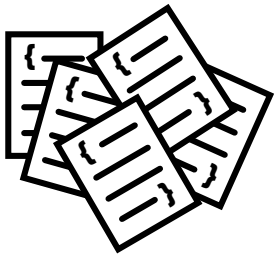




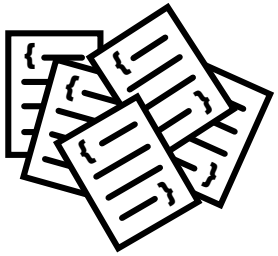

Mo



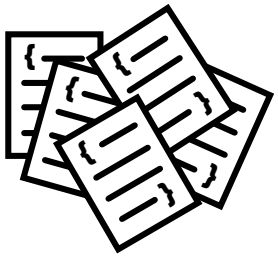

1.4



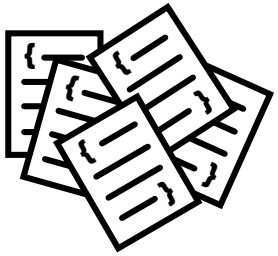
1.4
5.2



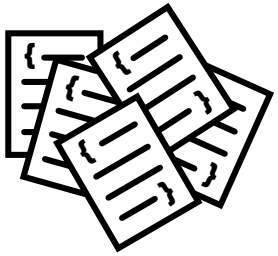
1.4
5.2
5.6



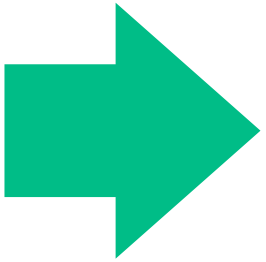
1.4
5.2
5.6
2.3

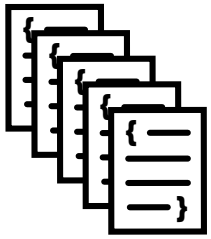


1.4
5.2
5.6
2.3
0.5
...



1.4
5.2
5.6
2.3
0.5
...





Understanding why

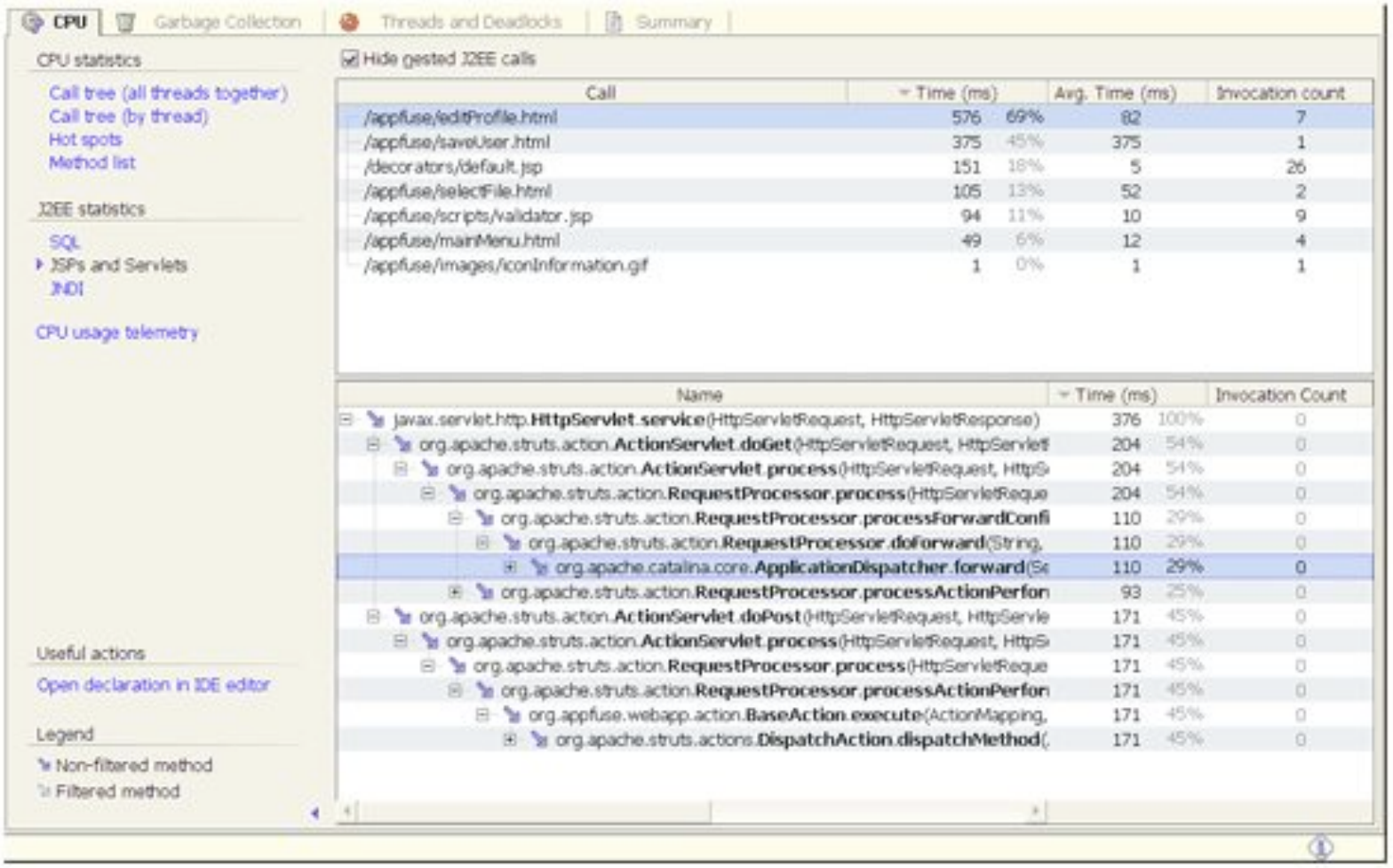
# gprof: flat profile

```
Flat profile:

Each sample counts as 0.01 seconds.

 %   cumulative   self              self     total
time   seconds   seconds    calls  ms/call  ms/call  name
33.34      0.02     0.02     7208     0.00     0.00  open
16.67      0.03     0.01      244     0.04     0.12  offtime
16.67      0.04     0.01        8     1.25     1.25  memccpy
16.67      0.05     0.01        7     1.43     1.43  write
16.67      0.06     0.01                             mcount
 0.00      0.06     0.00      236     0.00     0.00  tzset
 0.00      0.06     0.00      192     0.00     0.00  tolower
 0.00      0.06     0.00       47     0.00     0.00  strlen
 0.00      0.06     0.00       45     0.00     0.00  strchr
```

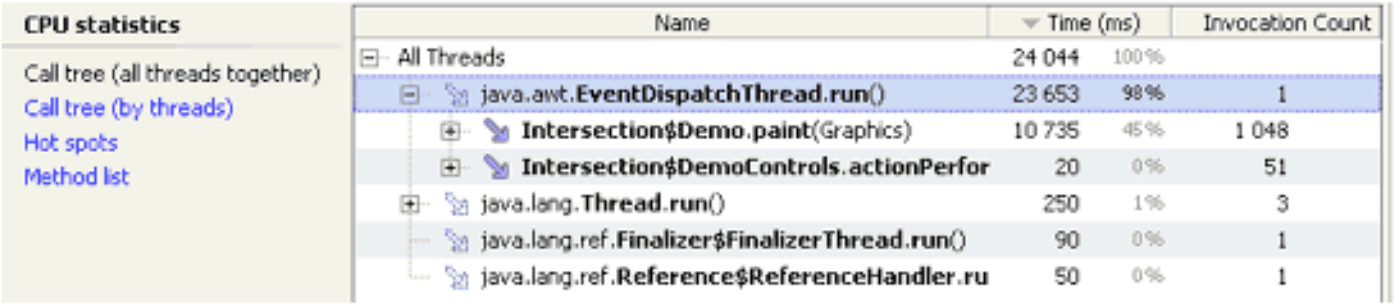
# gprof: call graph (~1984)

```
     index % time    self  children    called     name
                                                  <spontaneous>
[1]    100.0    0.00    0.05                 start [1]
                0.00    0.05       1/1           main [2]
                0.00    0.00       1/2           on_exit [28]
                0.00    0.00       1/1           exit [59]
-----------------------------------------------
                0.00    0.05       1/1           start [1]
[2]    100.0    0.00    0.05       1          main [2]
                0.00    0.05       1/1           report [3]
-----------------------------------------------
                0.00    0.05       1/1           main [2]
[3]    100.0    0.00    0.05       1          report [3]
                0.00    0.03       8/8           timelocal [6]
                0.00    0.01       1/1           print [9]
                0.00    0.01       9/9           fgets [12]
                0.00    0.00      12/34          strncmp <cycle 1> [40]
                0.00    0.00       8/8           lookup [20]
                0.00    0.00       1/1           fopen [21]
                0.00    0.00       8/8           chewtime [24]
                0.00    0.00       8/16          skipspace [44]
-----------------------------------------------
[4]     59.8    0.01        0.02       8+472     <cycle 2 as a whole>    [4]
                0.01        0.02     244+260         offtime <cycle 2> [7]
                0.00        0.00     236+1           tzset <cycle 2> [26]
-----------------------------------------------
```

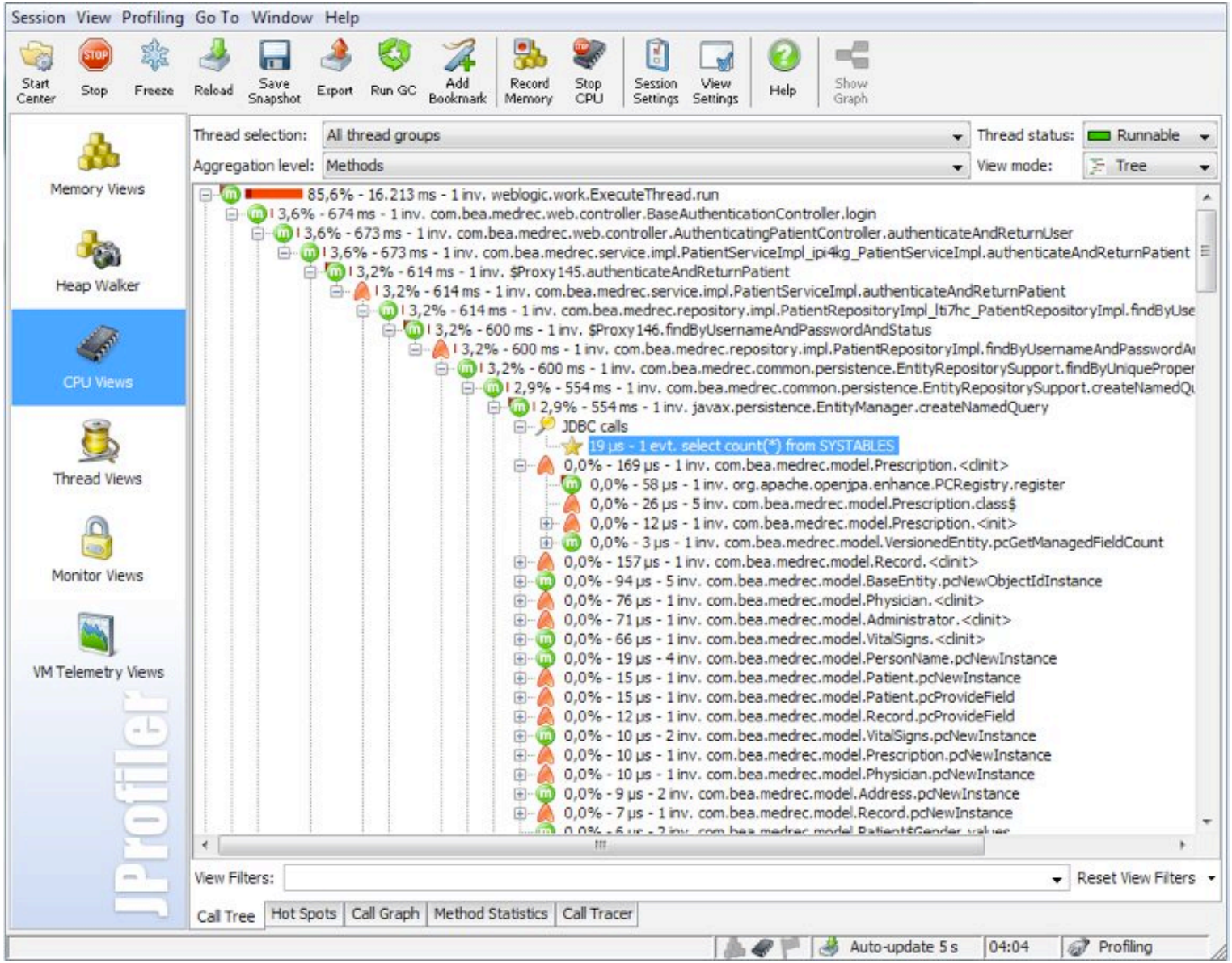
# YourKit


CPU
Garbage Collection
Threads and Deadlocks
Summary
CPU statistics
Call tree (all threads together)
Call tree (by thread)
Hot spots
Method list
J2EE statistics
SQL
JSPs and Servlets
JNDI
CPU usage telemetry
Useful actions
Open declaration in IDE editor
Legend
Non-filtered method
Filtered method
Hide gested J2EE calls
Call
Time (ms)
Avg. Time (ms)
Invocation count
/appfuse/editProfile.html 576 69% 82 7
/appfuse/saveUser.html 375 45% 375 1
/decorators/default.jsp 151 18% 5 26
/appfuse/selectFile.html 105 13% 52 2
/appfuse/scripts/validator.jsp 94 11% 10 9
/appfuse/mainMenu.html 49 6% 12 4
/appfuse/images/iconInformation.gif 1 0% 1 1
Name
Time (ms)
Invocation Count
javax.servlet.http.HttpServlet.service(HttpServletRequest, HttpServletResponse) 376 100% 0
org.apache.struts.action.ActionServlet.doGet(HttpServletRequest, HttpServlet 204 54% 0
org.apache.struts.action.ActionServlet.process(HttpServletRequest, HttpS 204 54% 0
org.apache.struts.action.RequestProcessor.process(HttpServletReque 204 54% 0
org.apache.struts.action.RequestProcessor.processForwardConfi 110 29% 0
org.apache.struts.action.RequestProcessor.doForward(String, 110 29% 0
org.apache.catalina.core.ApplicationDispatcher.forward(Se 110 29% 0
org.apache.struts.action.RequestProcessor.processActionPerfor 93 25% 0
org.apache.struts.action.ActionServlet.doPost(HttpServletRequest, HttpServle 171 45% 0
org.apache.struts.action.ActionServlet.process(HttpServletRequest, HttpS 171 45% 0
org.apache.struts.action.RequestProcessor.process(HttpServletReque 171 45% 0
org.apache.struts.action.RequestProcessor.processActionPerfor 171 45% 0
org.appfuse.webapp.action.BaseAction.execute(ActionMapping, 171 45% 0
org.apache.struts.actions.DispatchAction.dispatchMethod( 171 45% 0

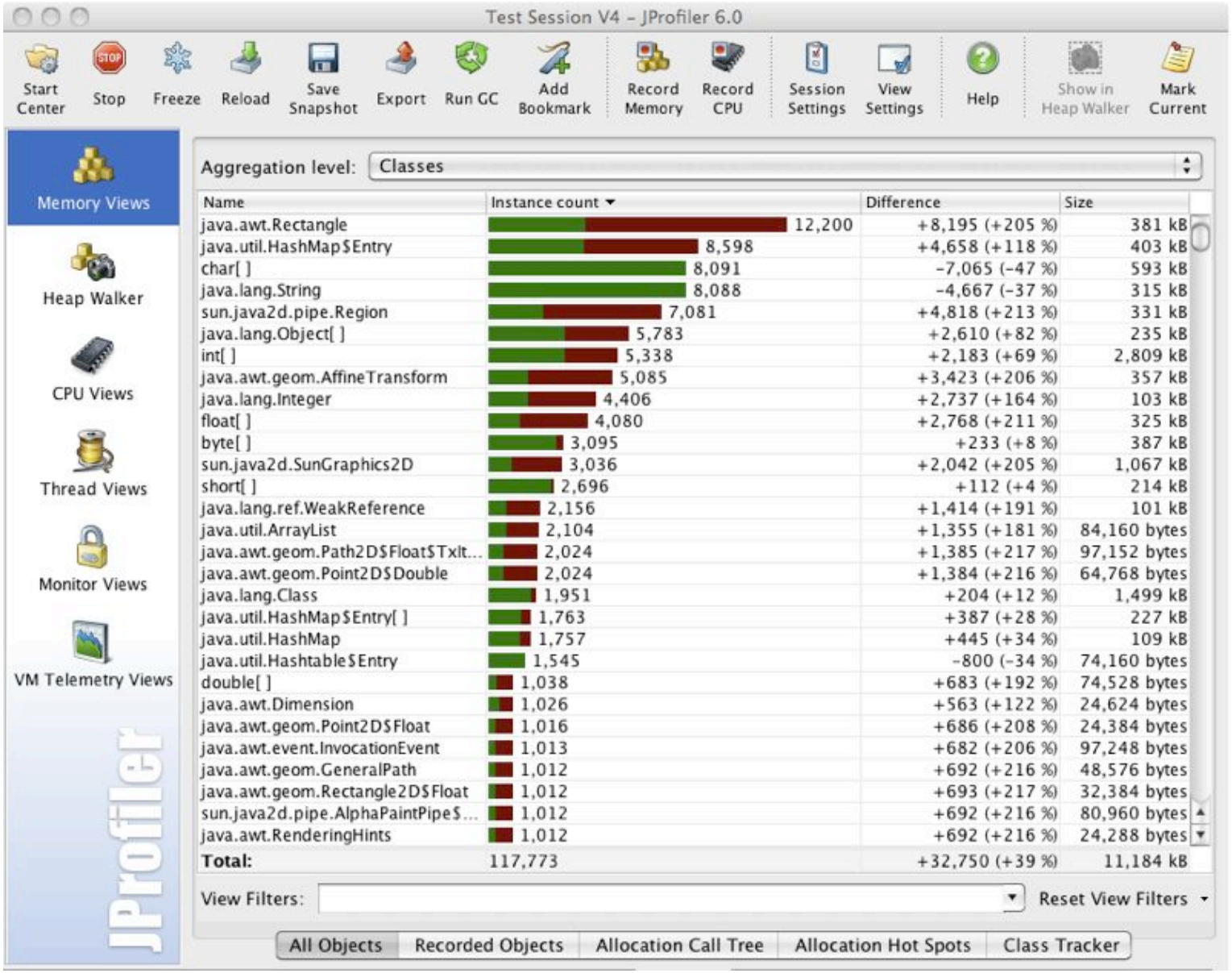
# YourKit


CPU statistics
Call tree (all threads together)
Call tree (by threads)
Hot spots
Method list
Name
Time (ms)
Invocation Count
All Threads 24 044 100 %
java.awt.EventDispatchThread.run() 23 653 98 % 1
Intersection$Demo.paint(Graphics) 10 735 45 % 1 048
Intersection$DemoControls.actionPerfor 20 0 % 51
java.lang.Thread.run() 250 1 % 3
java.lang.ref.Finalizer$FinalizerThread.run() 90 0 % 1
java.lang.ref.Reference$ReferenceHandler.ru 50 0 % 1

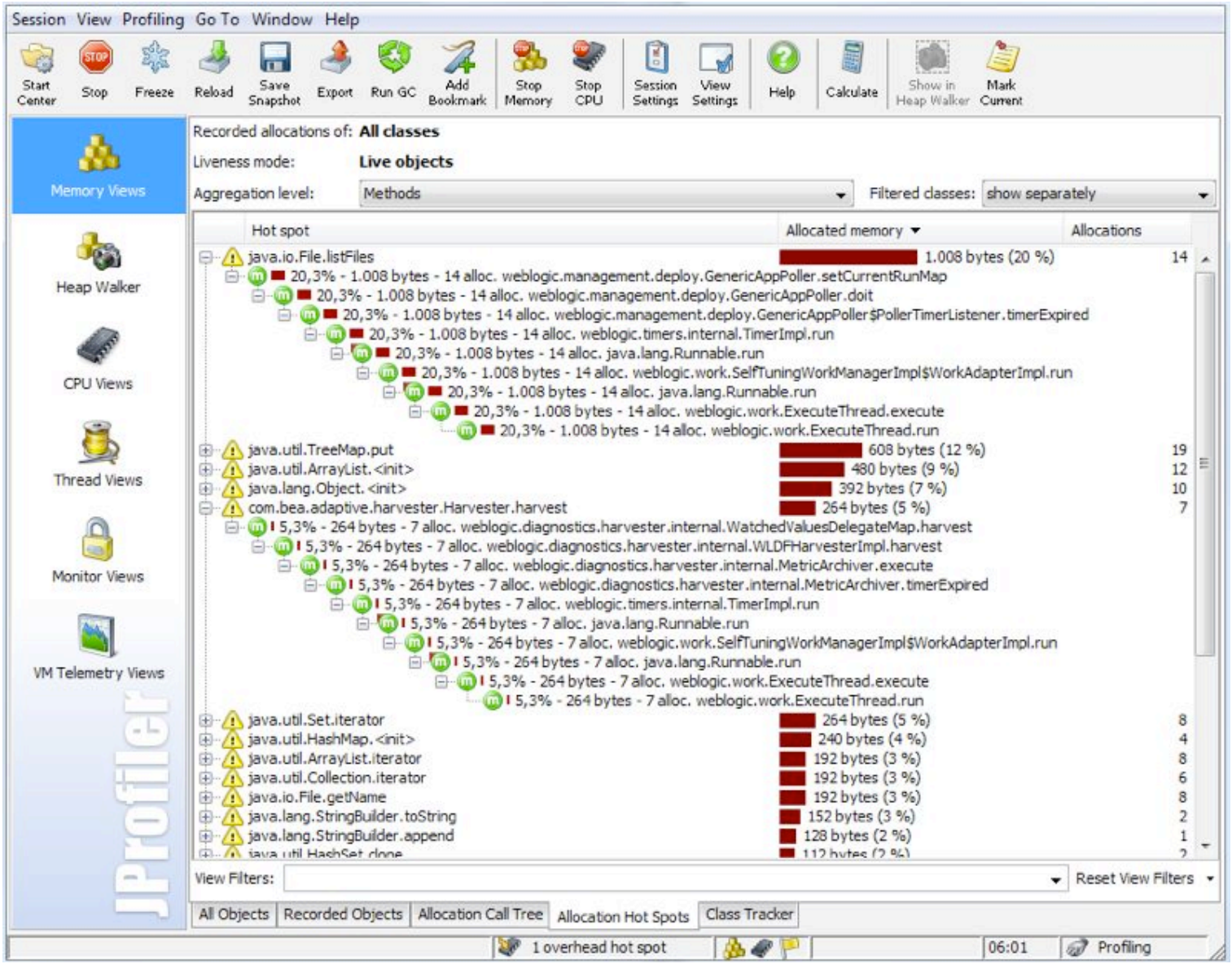
# JProfiler


Session View Profiling Go To Window Help
Start Center
Stop
Freeze
Reload
Save Snapshot
Export
Run GC
Add Bookmark
Record Memory
Stop CPU
Session Settings
View Settings
Help
Show Graph
Memory Views
Heap Walker
CPU Views
Thread Views
Monitor Views
VM Telemetry Views
JProfiler
Thread selection: All thread groups
Thread status: Runnable
Aggregation level: Methods
View mode: Tree
85,6% - 16.213 ms - 1 inv. weblogic.work.ExecuteThread.run
3,6% - 674 ms - 1 inv. com.bea.medrec.web.controller.BaseAuthenticationController.login
3,6% - 673 ms - 1 inv. com.bea.medrec.web.controller.AuthenticatingPatientController.authenticateAndReturnUser
3,6% - 673 ms - 1 inv. com.bea.medrec.service.impl.PatientServiceImpl_jpi4kg_PatientServiceImpl.authenticateAndReturnPatient
3,2% - 614 ms - 1 inv. $Proxy145.authenticateAndReturnPatient
3,2% - 614 ms - 1 inv. com.bea.medrec.service.impl.PatientServiceImpl.authenticateAndReturnPatient
3,2% - 614 ms - 1 inv. com.bea.medrec.repository.impl.PatientRepositoryImpl_lti7hc_PatientRepositoryImpl.findByUse
3,2% - 600 ms - 1 inv. $Proxy146.findByUsernameAndPasswordAndStatus
3,2% - 600 ms - 1 inv. com.bea.medrec.repository.impl.PatientRepositoryImpl.findByUsernameAndPasswordA
3,2% - 600 ms - 1 inv. com.bea.medrec.common.persistence.EntityRepositorySupport.findByUniqueProper
2,9% - 554 ms - 1 inv. com.bea.medrec.common.persistence.EntityRepositorySupport.createNamedQ
2,9% - 554 ms - 1 inv. javax.persistence.EntityManager.createNamedQuery
JDBC calls
19 µs - 1 evt. select count(*) from SYSTABLES
0,0% - 169 µs - 1 inv. com.bea.medrec.model.Prescription.<clinit>
0,0% - 58 µs - 1 inv. org.apache.openjpa.enhance.PCRegistry.register
0,0% - 26 µs - 5 inv. com.bea.medrec.model.Prescription.class$
0,0% - 12 µs - 1 inv. com.bea.medrec.model.Prescription.<init>
0,0% - 3 µs - 1 inv. com.bea.medrec.model.VersionedEntity.pcGetManagedFieldCount
0,0% - 157 µs - 1 inv. com.bea.medrec.model.Record.<clinit>
0,0% - 94 µs - 5 inv. com.bea.medrec.model.BaseEntity.pcNewObjectIdInstance
0,0% - 76 µs - 1 inv. com.bea.medrec.model.Physician.<clinit>
0,0% - 71 µs - 1 inv. com.bea.medrec.model.Administrator.<clinit>
0,0% - 66 µs - 1 inv. com.bea.medrec.model.VitalSigns.<clinit>
0,0% - 19 µs - 4 inv. com.bea.medrec.model.PersonName.pcNewInstance
0,0% - 15 µs - 1 inv. com.bea.medrec.model.Patient.pcNewInstance
0,0% - 15 µs - 1 inv. com.bea.medrec.model.Patient.pcProvideField
0,0% - 12 µs - 1 inv. com.bea.medrec.model.Record.pcProvideField
0,0% - 10 µs - 2 inv. com.bea.medrec.model.VitalSigns.pcNewInstance
0,0% - 10 µs - 1 inv. com.bea.medrec.model.Prescription.pcNewInstance
0,0% - 10 µs - 1 inv. com.bea.medrec.model.Physician.pcNewInstance
0,0% - 9 µs - 2 inv. com.bea.medrec.model.Address.pcNewInstance
0,0% - 7 µs - 1 inv. com.bea.medrec.model.Record.pcNewInstance
View Filters:
Reset View Filters
Call Tree
Hot Spots
Call Graph
Method Statistics
Call Tracer
Auto-update 5 s
04:04
Profiling

# JProfiler


Test Session V4 - JProfiler 6.0
Start Center
Stop
Freeze
Reload
Save Snapshot
Export
Run GC
Add Bookmark
Record Memory
Record CPU
Session Settings
View Settings
Help
Show in Heap Walker
Mark Current
Memory Views
Heap Walker
CPU Views
Thread Views
Monitor Views
VM Telemetry Views
JProfiler
Aggregation level: Classes
Name | Instance count | Difference | Size
java.awt.Rectangle 12,200 +8,195 (+205 %) 381 kB
java.util.HashMap$Entry 8,598 +4,658 (+118 %) 403 kB
char[ ] 8,091 -7,065 (-47 %) 593 kB
java.lang.String 8,088 -4,667 (-37 %) 315 kB
sun.java2d.pipe.Region 7,081 +4,818 (+213 %) 331 kB
java.lang.Object[ ] 5,783 +2,610 (+82 %) 235 kB
int[ ] 5,338 +2,183 (+69 %) 2,809 kB
java.awt.geom.AffineTransform 5,085 +3,423 (+206 %) 357 kB
java.lang.Integer 4,406 +2,737 (+164 %) 103 kB
float[ ] 4,080 +2,768 (+211 %) 325 kB
byte[ ] 3,095 +233 (+8 %) 387 kB
sun.java2d.SunGraphics2D 3,036 +2,042 (+205 %) 1,067 kB
short[ ] 2,696 +112 (+4 %) 214 kB
java.lang.ref.WeakReference 2,156 +1,414 (+191 %) 101 kB
java.util.ArrayList 2,104 +1,355 (+181 %) 84,160 bytes
java.awt.geom.Path2D$Float$TxIt... 2,024 +1,385 (+217 %) 97,152 bytes
java.awt.geom.Point2D$Double 2,024 +1,384 (+216 %) 64,768 bytes
java.lang.Class 1,951 +204 (+12 %) 1,499 kB
java.util.HashMap$Entry[ ] 1,763 +387 (+28 %) 227 kB
java.util.HashMap 1,757 +445 (+34 %) 109 kB
java.util.Hashtable$Entry 1,545 -800 (-34 %) 74,160 bytes
double[ ] 1,038 +683 (+192 %) 74,528 bytes
java.awt.Dimension 1,026 +563 (+122 %) 24,624 bytes
java.awt.geom.Point2D$Float 1,016 +686 (+208 %) 24,384 bytes
java.awt.event.InvocationEvent 1,013 +682 (+206 %) 97,248 bytes
java.awt.geom.GeneralPath 1,012 +692 (+216 %) 48,576 bytes
java.awt.geom.Rectangle2D$Float 1,012 +693 (+217 %) 32,384 bytes
sun.java2d.pipe.AlphaPaintPipe$... 1,012 +692 (+216 %) 80,960 bytes
java.awt.RenderingHints 1,012 +692 (+216 %) 24,288 bytes
Total: 117,773 +32,750 (+39 %) 11,184 kB
View Filters:
Reset View Filters
All Objects
Recorded Objects
Allocation Call Tree
Allocation Hot Spots
Class Tracker

# JProfiler


Session View Profiling Go To Window Help
Start Center
Stop
Freeze
Reload
Save Snapshot
Export
Run GC
Add Bookmark
Stop Memory
Stop CPU
Session Settings
View Settings
Help
Calculate
Show in Heap Walker
Mark Current
Memory Views
Heap Walker
CPU Views
Thread Views
Monitor Views
VM Telemetry Views
JProfiler
Recorded allocations of: All classes
Liveness mode: Live objects
Aggregation level: Methods
Filtered classes: show separately
Hot spot
Allocated memory
Allocations
java.io.File.listFiles 1.008 bytes (20 %) 14
20,3% - 1.008 bytes - 14 alloc. weblogic.management.deploy.GenericAppPoller.setCurrentRunMap
20,3% - 1.008 bytes - 14 alloc. weblogic.management.deploy.GenericAppPoller.doit
20,3% - 1.008 bytes - 14 alloc. weblogic.management.deploy.GenericAppPoller$PollerTimerListener.timerExpired
20,3% - 1.008 bytes - 14 alloc. weblogic.timers.internal.TimerImpl.run
20,3% - 1.008 bytes - 14 alloc. java.lang.Runnable.run
20,3% - 1.008 bytes - 14 alloc. weblogic.work.SelfTuningWorkManagerImpl$WorkAdapterImpl.run
20,3% - 1.008 bytes - 14 alloc. java.lang.Runnable.run
20,3% - 1.008 bytes - 14 alloc. weblogic.work.ExecuteThread.execute
20,3% - 1.008 bytes - 14 alloc. weblogic.work.ExecuteThread.run
java.util.TreeMap.put 608 bytes (12 %) 19
java.util.ArrayList.<init> 480 bytes (9 %) 12
java.lang.Object.<init> 392 bytes (7 %) 10
com.bea.adaptive.harvester.Harvester.harvest 264 bytes (5 %) 7
5,3% - 264 bytes - 7 alloc. weblogic.diagnostics.harvester.internal.WatchedValuesDelegateMap.harvest
5,3% - 264 bytes - 7 alloc. weblogic.diagnostics.harvester.internal.WLDFHarvesterImpl.harvest
5,3% - 264 bytes - 7 alloc. weblogic.diagnostics.harvester.internal.MetricArchiver.execute
5,3% - 264 bytes - 7 alloc. weblogic.diagnostics.harvester.internal.MetricArchiver.timerExpired
5,3% - 264 bytes - 7 alloc. weblogic.timers.internal.TimerImpl.run
5,3% - 264 bytes - 7 alloc. java.lang.Runnable.run
5,3% - 264 bytes - 7 alloc. weblogic.work.SelfTuningWorkManagerImpl$WorkAdapterImpl.run
5,3% - 264 bytes - 7 alloc. java.lang.Runnable.run
5,3% - 264 bytes - 7 alloc. weblogic.work.ExecuteThread.execute
5,3% - 264 bytes - 7 alloc. weblogic.work.ExecuteThread.run
java.util.Set.iterator 264 bytes (5 %) 8
java.util.HashMap.<init> 240 bytes (4 %) 4
java.util.ArrayList.iterator 192 bytes (3 %) 8
java.util.Collection.iterator 192 bytes (3 %) 6
java.io.File.getName 192 bytes (3 %) 8
java.lang.StringBuilder.toString 152 bytes (3 %) 2
java.lang.StringBuilder.append 128 bytes (2 %) 1
View Filters:
Reset View Filters
All Objects
Recorded Objects
Allocation Call Tree
Allocation Hot Spots
Class Tracker
1 overhead hot spot
06:01
Profiling

# Retrospective on profiling

Information conveyed hasn't evolved since gprof

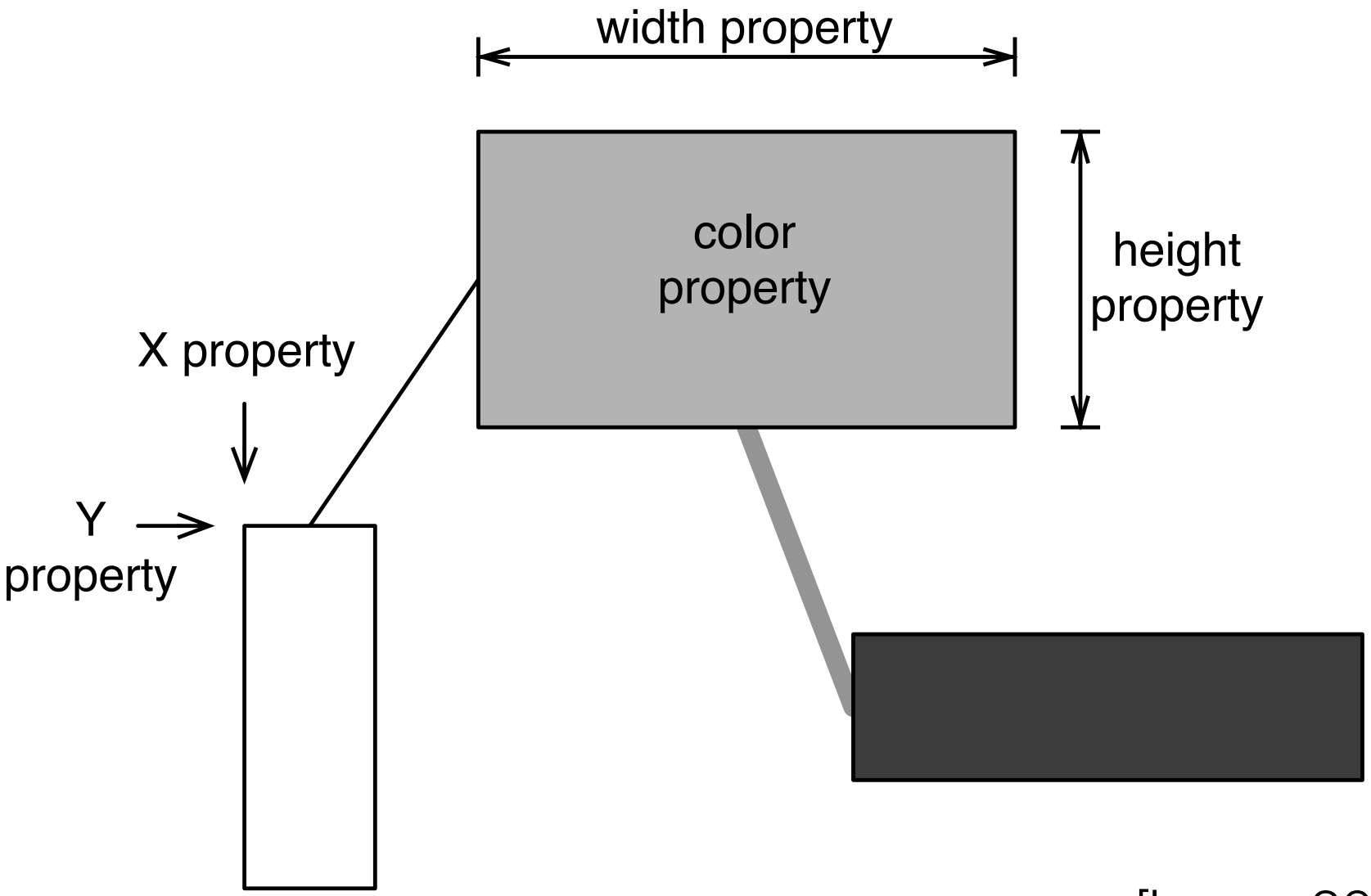
Useful to understand *what* happened

But is of little help to understand *why* and *how*

# Roadmap

1. Polymetric views
2. Profiling Blueprint
3. Implementation

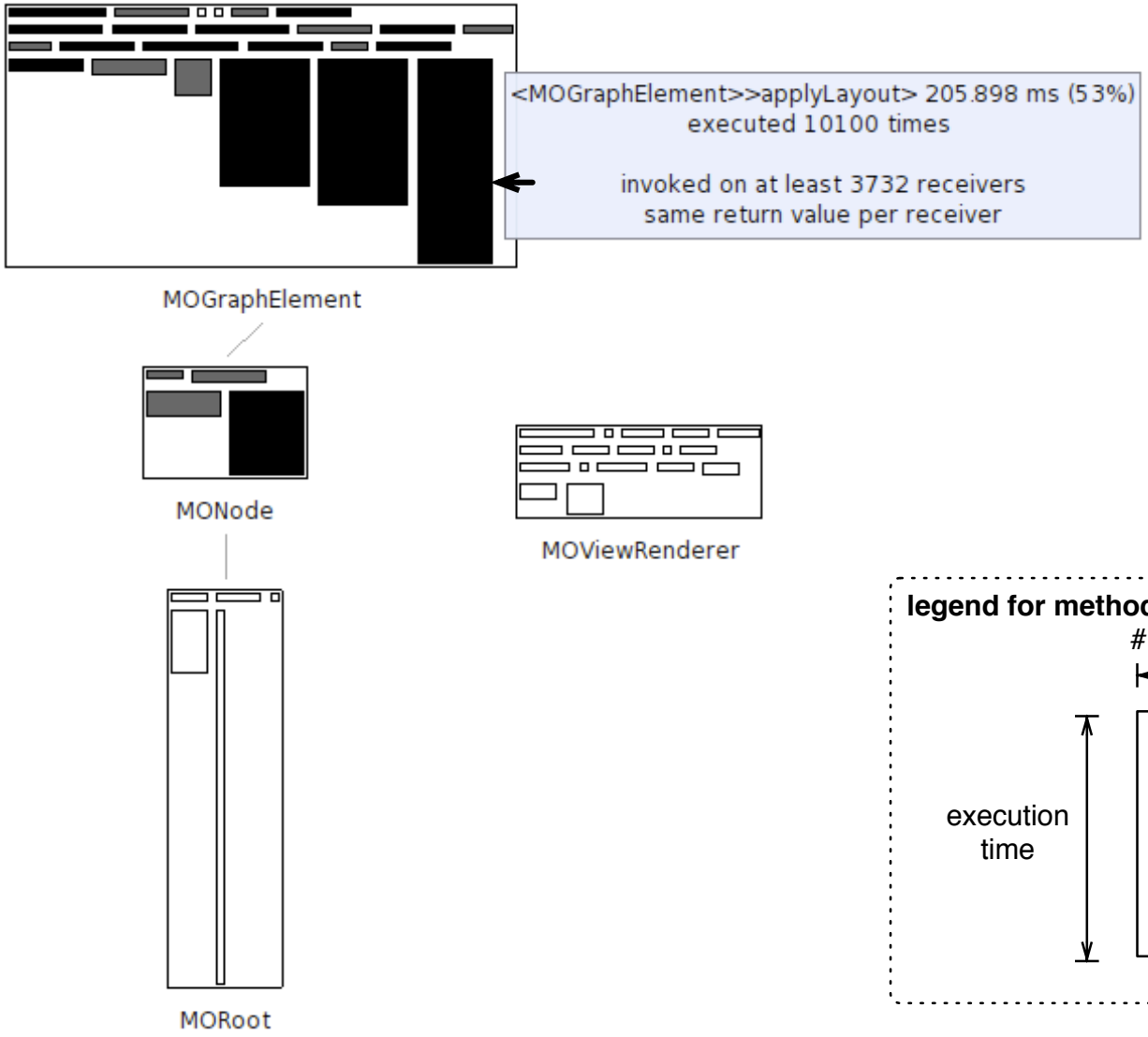
# *Polymetric view* can map up to 5 dimensions



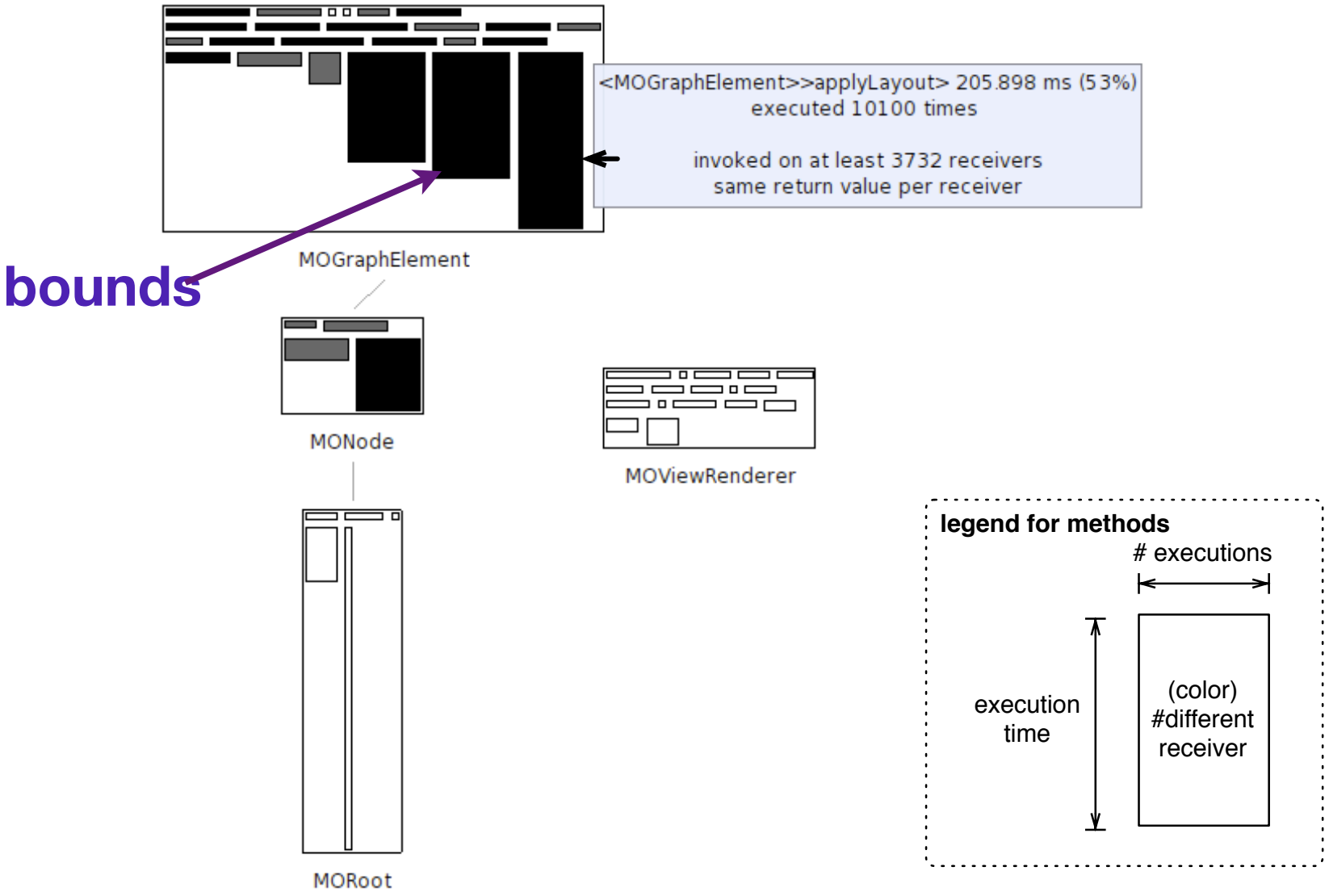

[Lanza 2003]

```
KaiProfiler
  viewProfiling: [
    | view |
    view := MOViewRenderer new.
    view
      nodes: (1 to: 100)
      forEach: [:each |
        view nodes: (1 to: 100)].
    view root applyLayout
  ]
```

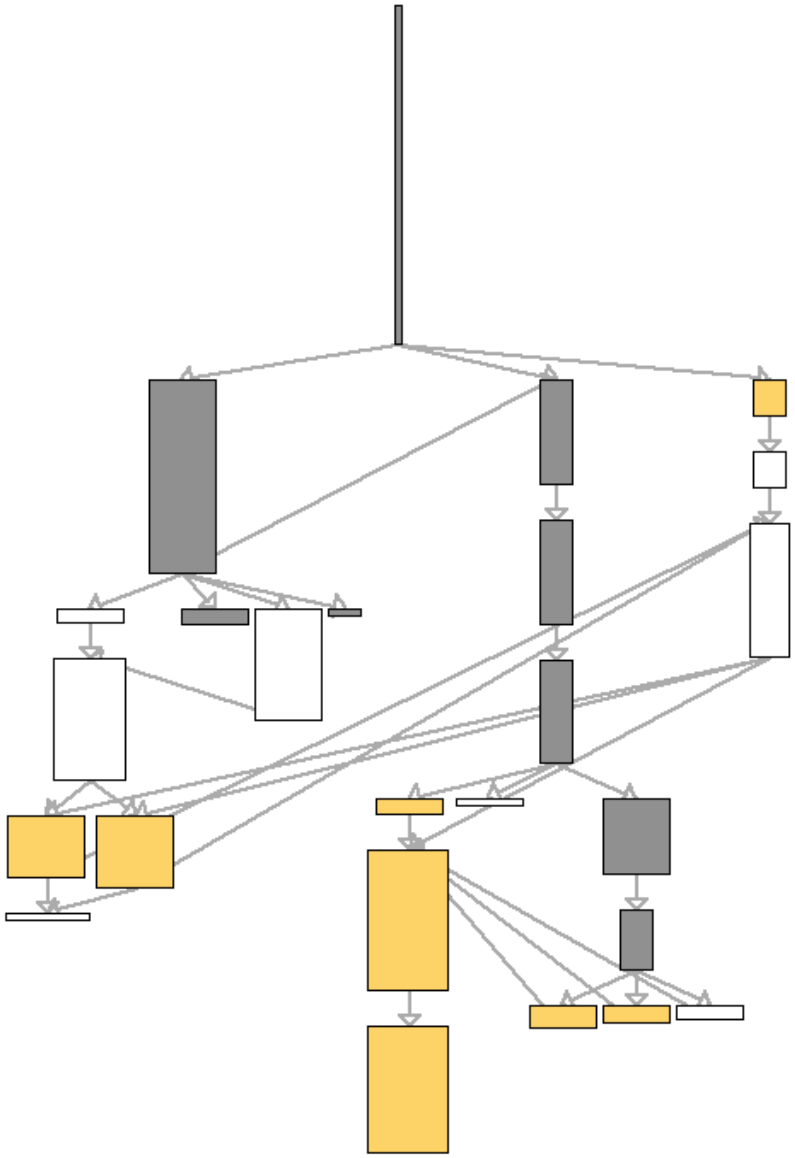
# Structural blueprint

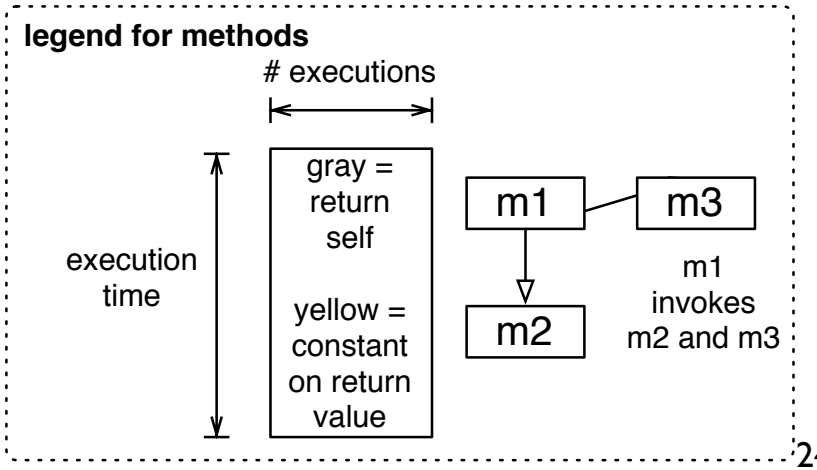


# Structural blueprint

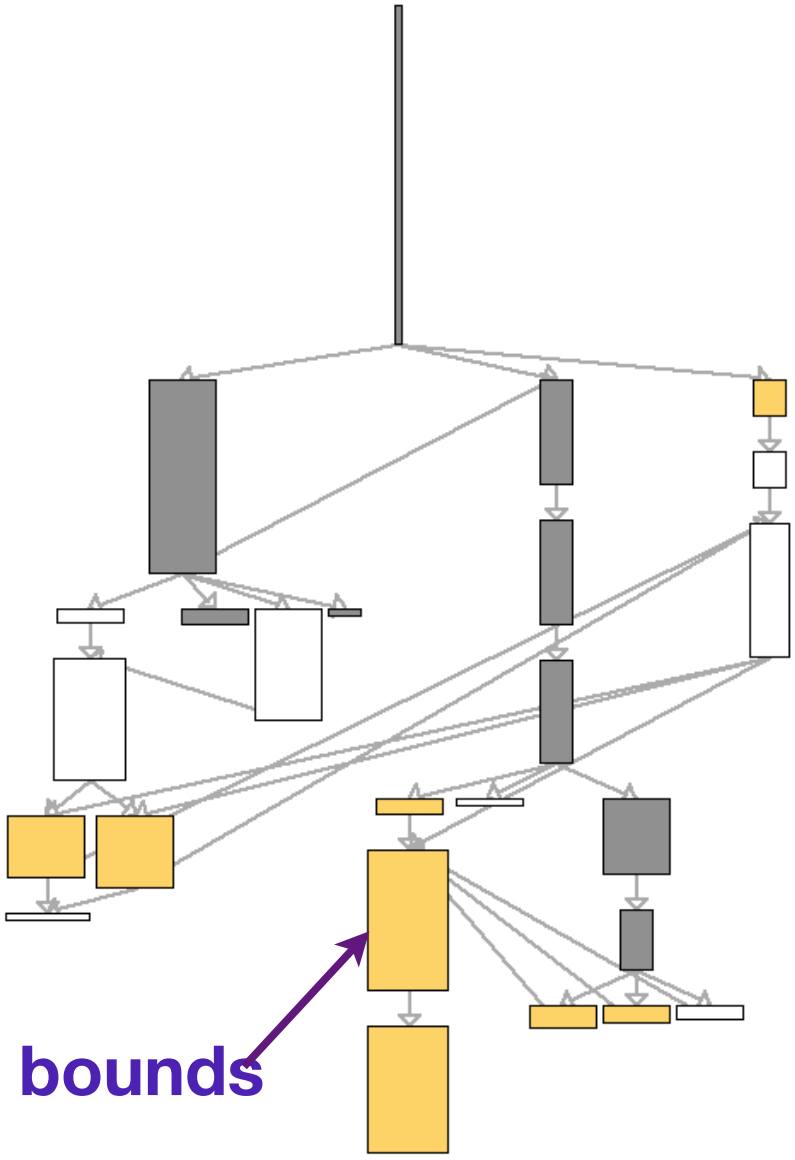


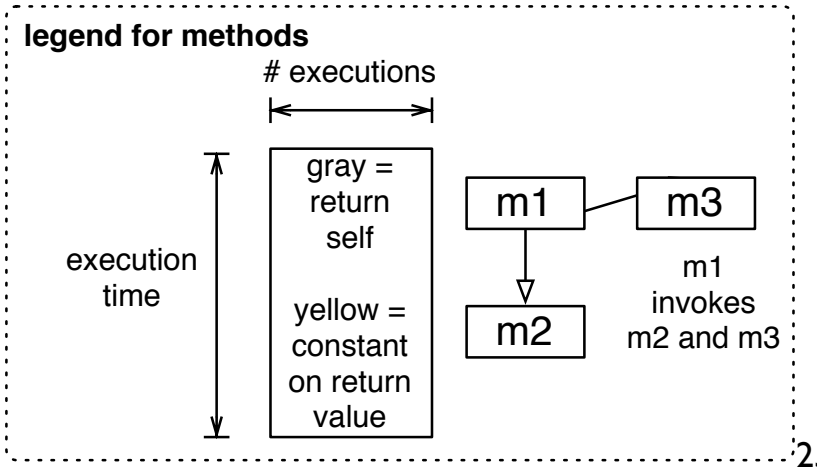
# Behavioral blueprint



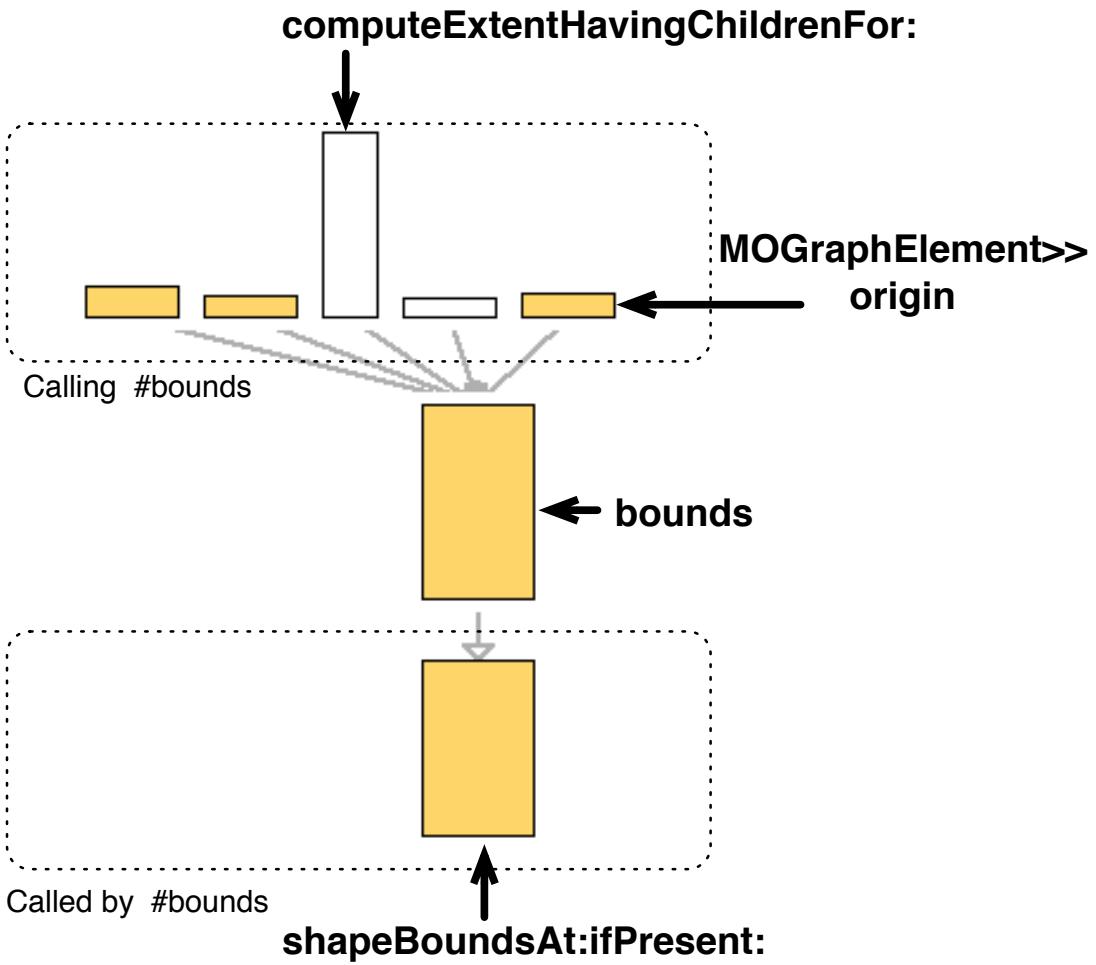


# Behavioral blueprint



# Detailed behavioral blueprint

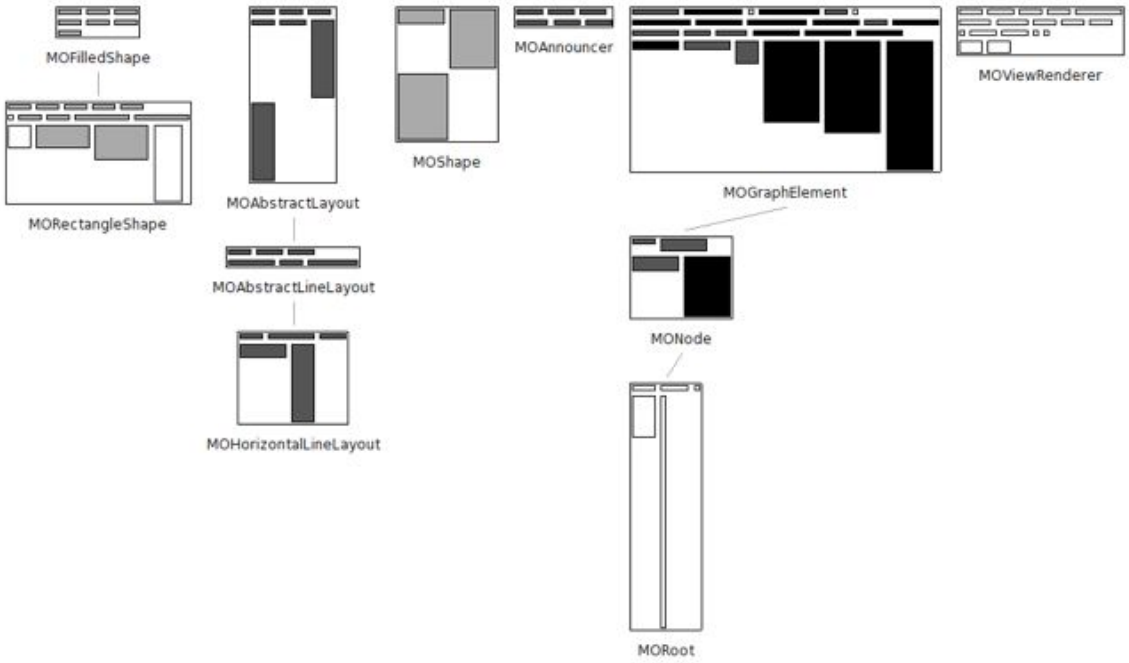


# Code of the bounds method

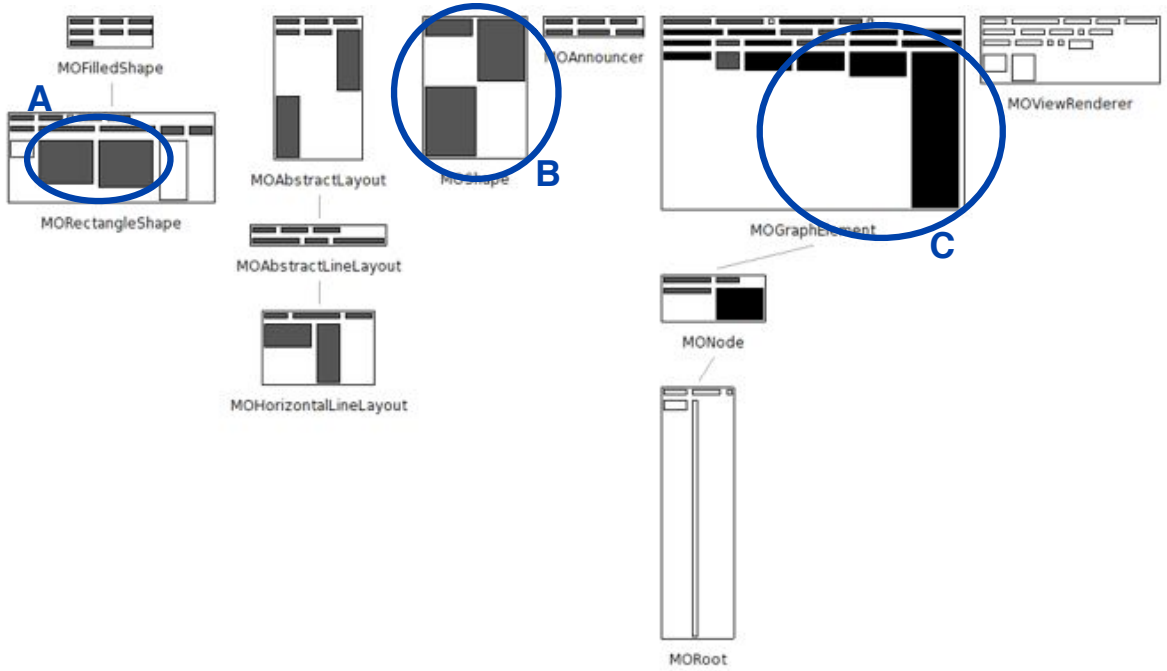
```
MOGraphElement>>bounds
  "Answer the bounds of the receiver."

  | basicBounds |

  self shapeBoundsAt: self shape ifPresent: [ :b | ^ b ].

  basicBounds := shape computeBoundsFor: self.
  self shapeBoundsAt: self shape put: basicBounds.

  ^ basicBounds
```

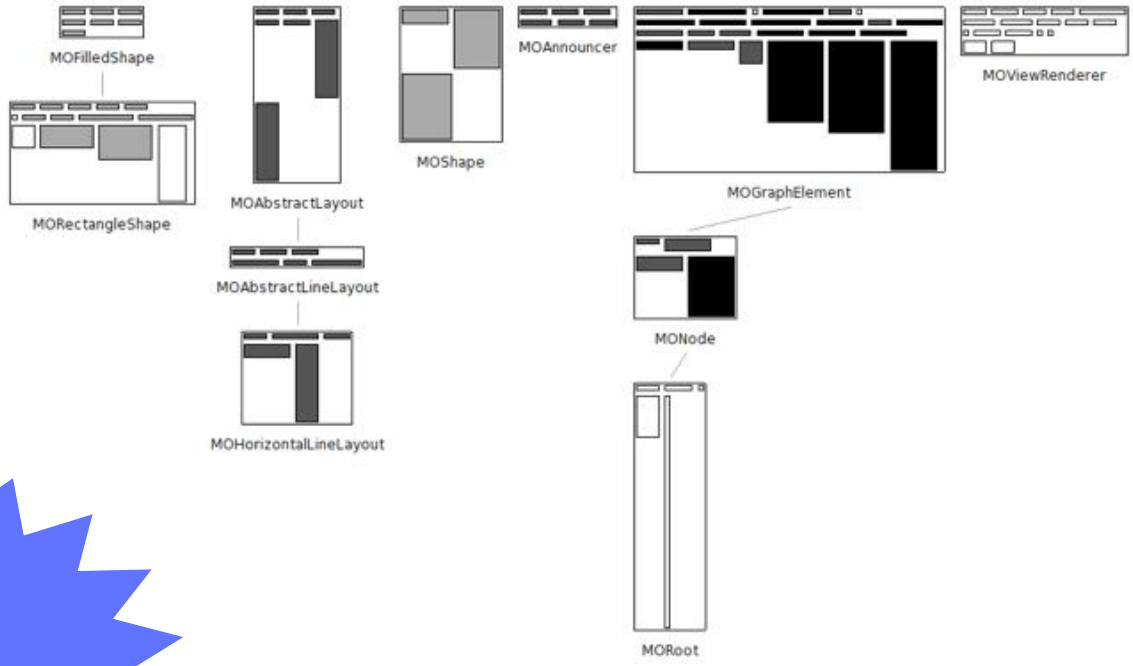
# Memoizing

```
MOGraphElement>>bounds
  "Answer the bounds of the receiver."

  | basicBounds |
  boundsCache ifNotNil: [ ^ boundsCache ].
  self shapeBoundsAt: self shape ifPresent: [ :b | ^ b ].

  basicBounds := shape computeBoundsFor: self.
  self shapeBoundsAt: self shape put: basicBounds.

  ^ boundsCache := basicBounds
```



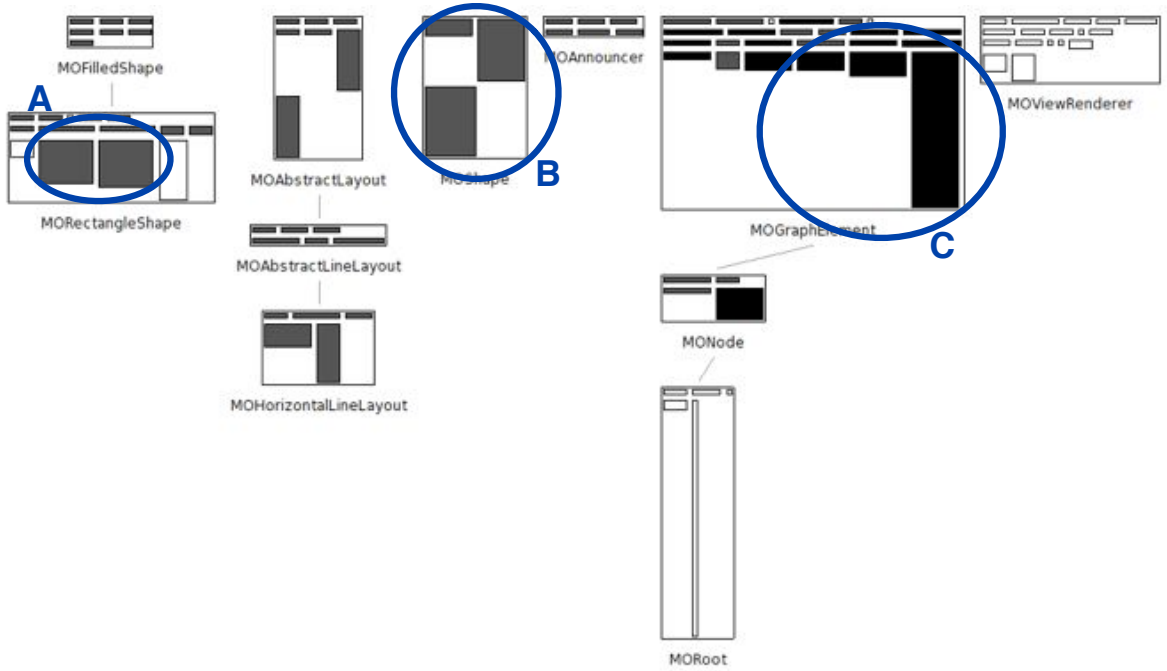

Upgrading
MOGraphElement>>bounds

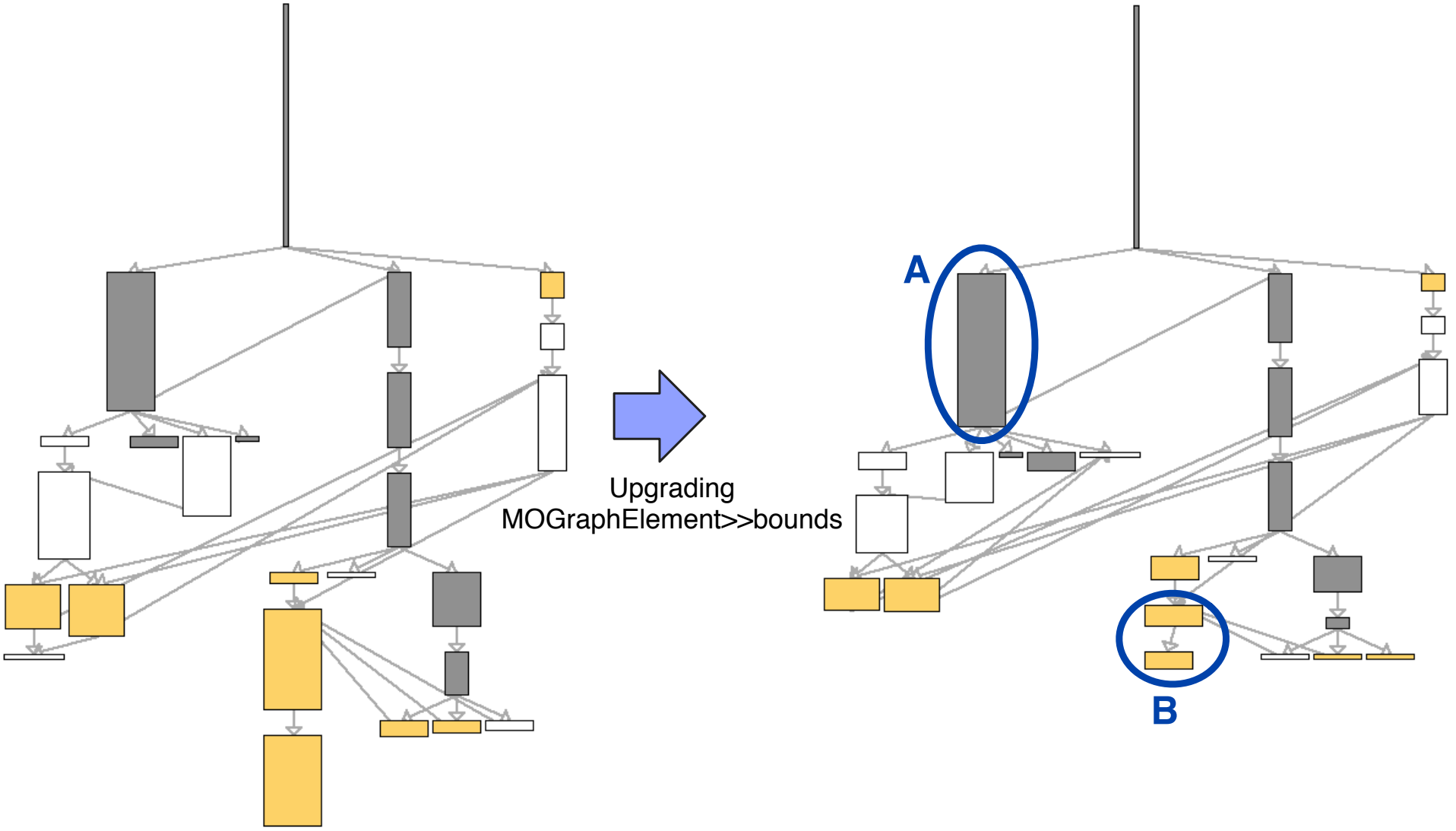


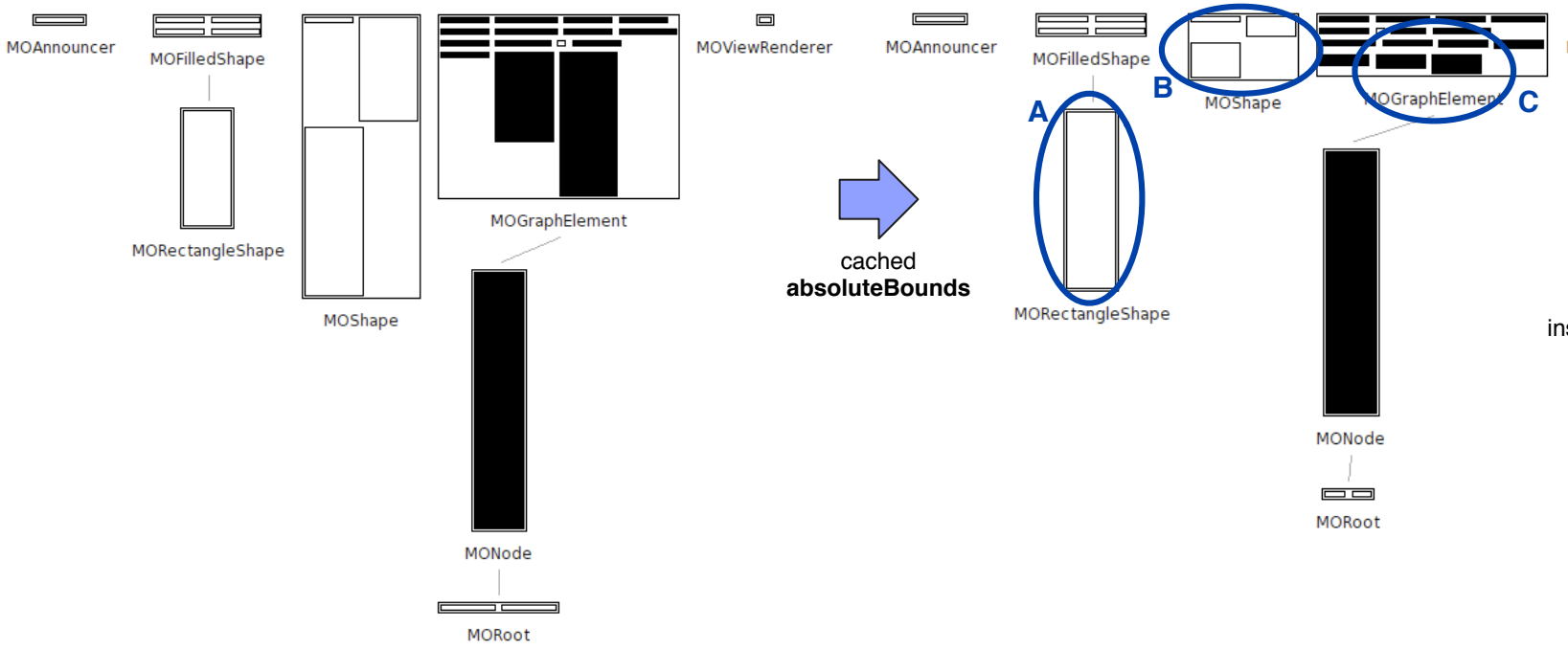


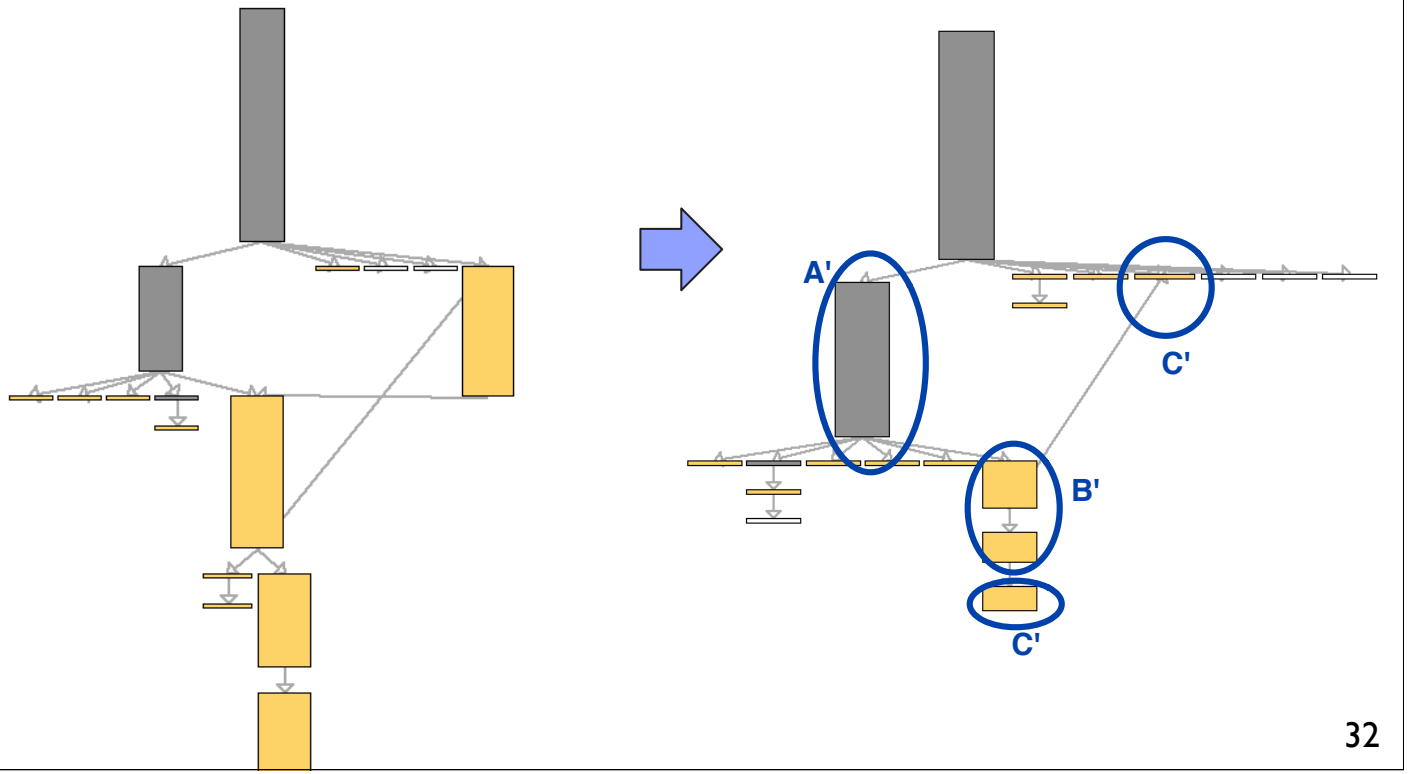

43% speedup

Upgrading
MOGraphElement>>bounds




Upgrading
MOGraphElement>>bounds
A
B


MOAnnouncer
MOFilledShape
MORectangleShape
MOShape
MOGraphElement
MONode
MORoot
MOViewRenderer
MOAnnouncer
MOFilledShape
A
B
MOShape
MOGraphElement
C
MORectangleShape
MONode
MORoot
cached
absoluteBounds


A'
B'
C'
C'

# Implementation

We use the following metrics:

- execution time for a method (% and ms)
- number of executions
- number of different object receivers

Dynamic properties

- a method performs a side effect
- a method is void (i.e., return self in Pharo)

# Naive (but effective) implementation

Code to profile is executed twice

- using a sampling method to get the execution time
- instrumentation to get all the remaining metrics

Use hash values to distinguish between different receiver objects

Built a kind of AOP mechanism for the low level instrumentation

# Implementation techniques

Visualizations are generated using a scripting languages

... in Mondrian

Limitation

- hash code collisions (problem in Pharo)
- need to do execute the code to profile twice (sampling and instrumentation)


Implemented in Pharo

Smalltalk dialect

Dynamically typed language

# Conclusion

Effective visualizations

Smooth integration in the programming environment

Implemented in Pharo

# Conclusion

A number of bottlenecks were identified

No general rule for pattern identification

Visualizations are effective for identifying potential candidate for optimization

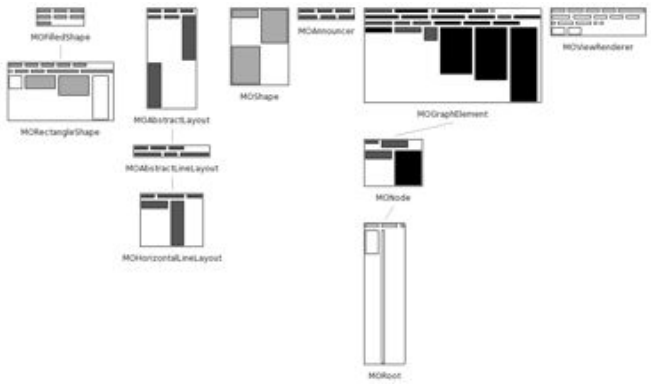
# Conclusion

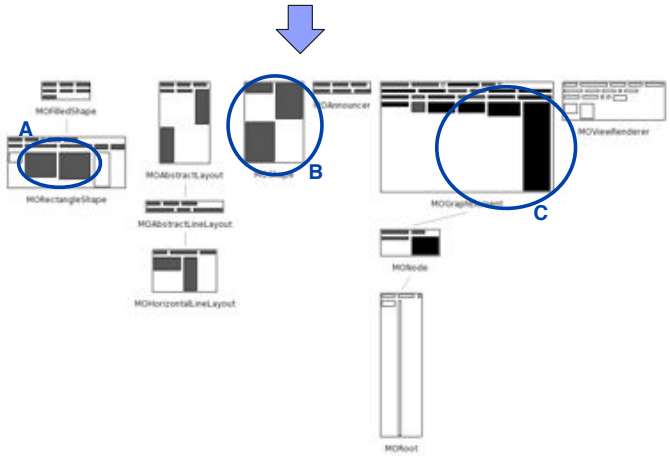
## Future work

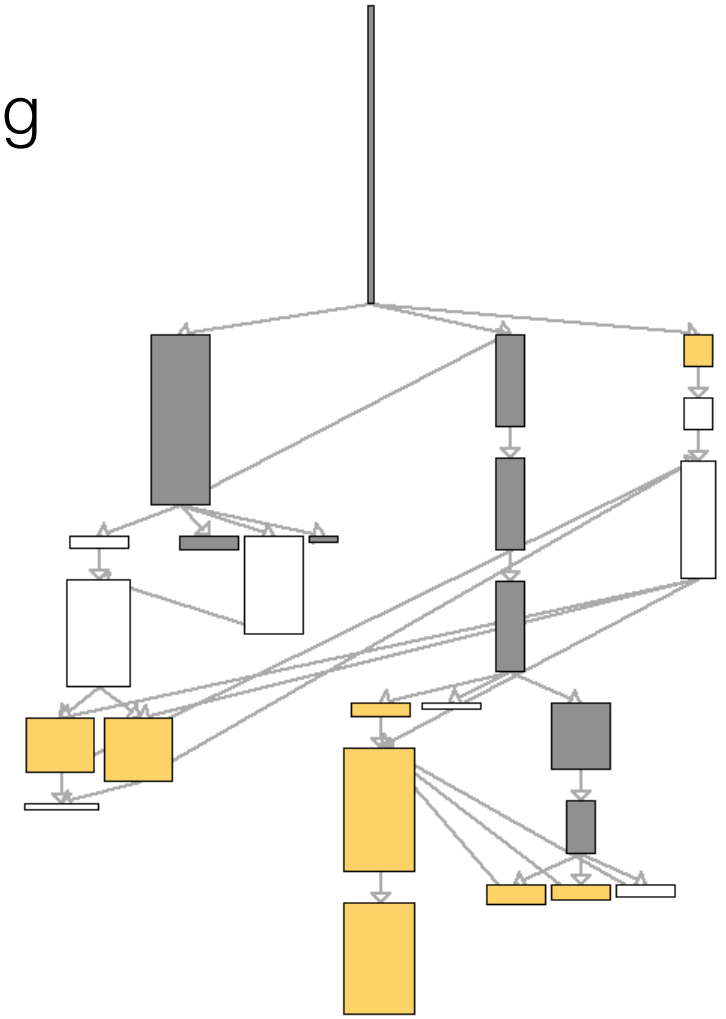
- close integration in the programming environment
- dedicated visualization for comparison
- additional metrics, e.g., the number of executed bytecodes, memory usage

# Visualizing Dynamic Metrics with Profiling Blueprints

`www.moosetechnology.org/tools/Spy`





Alexandre Bergel, Romain Robbes, Walter Binder

*abergel@dcc.uchile.cl*