Aspects of Memory Management

Celina Gibbs
Yvonne Coady

The MOD(ularity) SQUAD
University of Victoria

finding boundary violations

- a subsystem evolves to be separate/portable
  - eg., memory management in a JVM
    - strict interface introduced
    - subsystem code refactored to use the interface
- aspect-oriented programming (AOP) provides useful feedback on design invariants…

finding boundary violations

violations:
  all calls made to Jikes
  from within MMTk
  that do not use the interface

what happens as the system evolves?

finding boundary violations

```
aspect MMTkBoundaryChecking {

       pointcut boundaryViolation():
              call(* com..*(..))
              && within(org.mmtk..*)
              && !within(org.mmtk.vm.*);

       declare warning: boundaryViolation():
              "call made to code OUTSIDE OF MMTk";

}
```

MMTk/Jikes Boundary	Jan 2004	Oct 2004
warnings generated	881	2

but what about *core* system concerns?

- age old tension…
  - evolvable
  - efficient
- typical ways to separate concerns
  - hierarchical decomposition

but what about *core* system concerns?

- age old tension…
  - evolvable
  - efficient
- typical ways to separate concerns
  - hierarchical decomposition
  - global flags

but what about *core* system concerns?

perhaps distasteful, but cheap!

- age old tension…
  - evolvable
  - efficient
- typical ways to separate concerns
  - hierarchical decomposition
  - global flags
  - *empty interfaces (flags for lower-level concerns)*
  - *preprocessor directives*

the problem…

- can aspects ease evolvability within this (inhospitable) systems domain?
  - can the current combinations of these mechanisms be reconciled into one?
  - does the design intent of intricate core concerns become more clear with localization?
  - are aspects fast enough to compete?
- case study using the Jikes RVM and MMTk

case study

- strengths
  - real system
  - intense evolution
  - diverse internal structure
    - dynamic analysis tools
    - domain specific design patterns
    - design invariants
- weaknesses
  - Jikes build process not Eclipse friendly
    - lack of AOP tool support (but coming?)
  - generating additional warnings

outline

- background
  - Jikes Research Virtual Machine (RVM)
  - Aspect-Oriented Programming (AOP) with AspectJ
- case study
  - GCSpy analysis tool
  - prepare/release staged protocol
  - VM_Interruptible design invariant
- case study analysis
  - impact on evolution and adaptation
- conclusions and ongoing work

Jikes RVM

- new virtual machine technologies and design alternatives (> 100 research papers)
  - dynamic compilation, adaptive optimization, thread scheduling, synchronization and *garbage collection*
- open source, 1500+ Java files
- multi-platform
  - OS X/PowerPC, Linux/PowerPC, Linux/IA-32

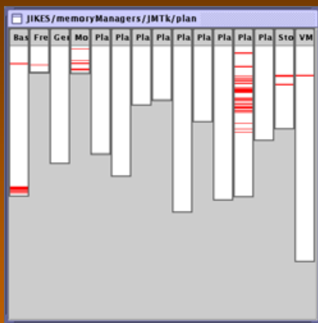
Memory Management Toolkit (MMTk)

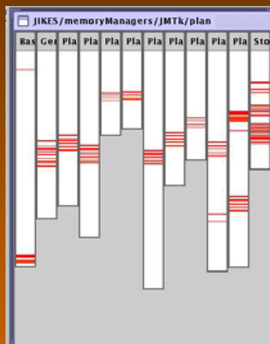
- simplifies development of Garbage Collectors
- 8 different stop the world collectors in MMTk
  - copyMS, markSweep, refCount, genMS, …
- MMTk recently separated from the rest of Jikes
  - general framework
  - ports to other systems

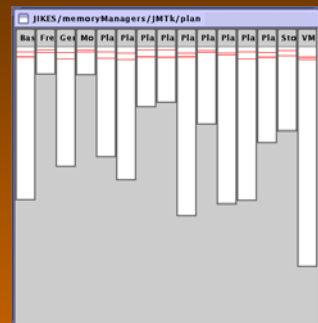
crosscutting concerns

- well defined design, but scattered and tangled implementation
- difficult to modularize with traditional approaches
- an aspect provides a modular implementation of a crosscutting concern
  - localizes the implementation
  - makes internal structure and external interaction explicit

case study: diverse internal structure





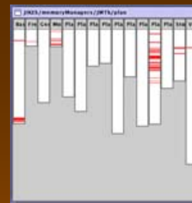


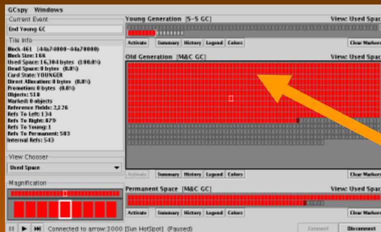
dynamic analysis:
GCSpy

design patterns:
prepare/release

design invariants:
VM_Interruptible

dynamic analysis: GCSpy

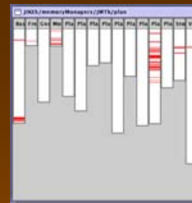


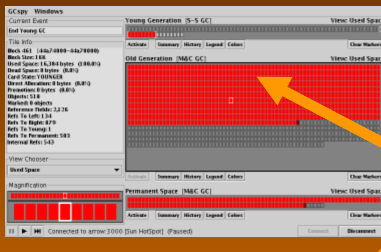


framework designed to visualize memory management systems
[OOPSLA’02, ISMM’02]

full/empty blocks
…how full?

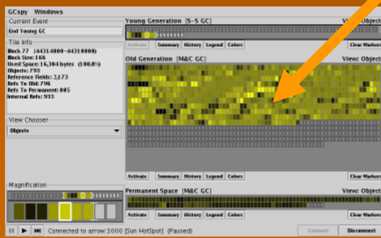
dynamic analysis: GCSpy





framework designed to visualize memory management systems
[OOPSLA’02, ISMM’02]

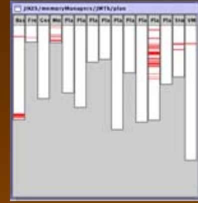
full/empty blocks
…how full?
brighter = fuller



“only the least possible amount of code is added to the system being visualized”

over 50 places in the code…
- 41 new methods,
- 14 `if (VM_Interface.GCSPY)`
- 12 `#-if RVM_WITH_GCSPY`

GCSpy in MMTk



VM_Interface.java

```
public static final boolean GCSPY =
    //-#if RVM_WITH_GCSPY
    true;
    //-#else
    false;
    //-#endif
    …
```

Instrumentation in MMTk (5 classes, 14 occurences)

```
if (VM_Interface.GCSPY)
    …
```

GCSpy in MMTk

MainThread.java

```
//-#if RVM_WITH_GCSPY
import
com.ibm.JikesRVM.memoryManagers.mmInterface.MM_Interface;
//-#endif

…
public void run () {
      //-#if RVM_WITH_GCSPY
      MM_Interface.startGCSpyServer();
      //-#endif
```


```
if (VM_Interface.GCSPY)
    …
```

GCSpy in MMTk

MainThread.java

```
//-#if RVM_WITH_GCSPY
import
com.ibm.JikesRVM.memoryManagers.mmInterface.MM_Interface;
//-#endif

…
public void run () {
       //-#if RVM_WITH
       MM_Interface.st
       //-#endif
```

VM_BootRecord.java

```
//-#if RVM_WITH_GCSPY
   // GCspy entry points
   public VM_Address gcspyDriverAddStreamIP;
   public VM_Address gcspyDriverEndOutputIP;
   …
   public VM_Address gcspySprintfIP;
//-#endif
```

```
  if (VM_Interface.GCSPY)
       …
```

GCSpy in MMTk

MainThread.java

```
//-#if RVM_WITH_GCSPY
import
com.ibm.JikesRVM.memoryManagers.mmInterface.MM_Interface;
//-#endif

…
public void run () {
       //-#if RVM_WITH
       MM_Interface.st
       //-#endif
```

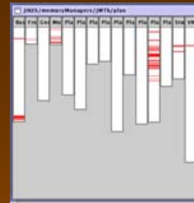
VM_BootRecord.java

```
//-#if RVM_WITH_GCSPY
   // GCspy entry points
   public VM_Address gcspyDriverAddStreamIP;
   public VM_Address gcspyDriverEndOutputIP;
   …
   public VM_Address gcspySprintfIP;
```


VM_Syscall.java

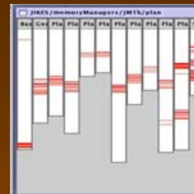
```
//-#if RVM_WITH_GCSPY
public static VM_Address
gcspyDriverAddStream (VM_Address driver, int it) {
  return null; }

public static void
gcspyDriverEndOutput (VM_Address driver) {}
…
//-#endif
```

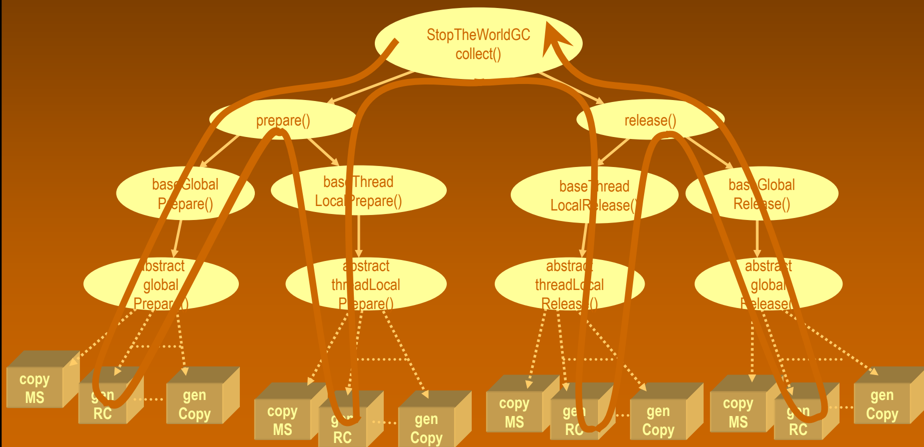
GCSpy Aspect



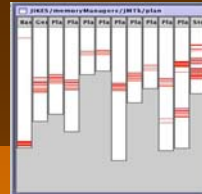
```
aspect GCSpy {

  public static final
  void Plan.startGCSpyServer(int port, boolean wait)
       throws VM_PragmaInterruptible { … }

  before(): execution(* Plan.boot())  {
       Plan.objectMap = new ObjectMap();
       Plan.objectMap.boot();
  }

  before(VM_Address ref) throws VM_PragmaUninterruptible:
      args(ref, ..) && execution(* Plan.postCopy(VM_Address, ..)) {

       if (GCSpy.getGCSpyPort() != 0)
               Plan.objectMap.alloc(VM_Magic.objectAsAddress(ref));
  }
  …
}
```

control flow of collector thread

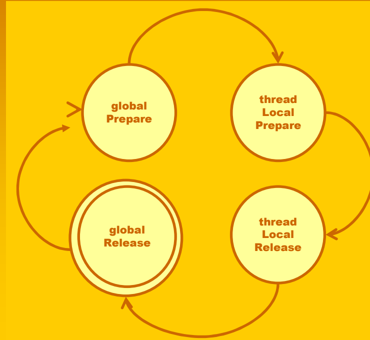




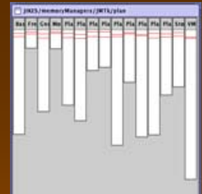
domain-specific design patterns



```
privileged aspect PolicyAspect {

  after(Plan p): target(p)
              && (execution(* Plan.globalPrepare(..))      ||
                  execution(* Plan.threadLocalPrepare(..)) ||
                  execution(* Plan.threadLocalRelease(..)) ||
                  execution(* Plan.globalRelease(..))) {
    switch(state){
      case(GLOBAL_PREPARE):
        CopySpace.prepare();
        Plan.msSpace.prepare();
        ImmortalSpace.prepare();
        Plan.losSpace.prepare();
        state++;
        break;
      case(LOCAL_PREPARE):
        …
      case(LOCAL_RELEASE):
        …
      case(GLOBAL_RELEASE):
        … }
    }
}
```



VM_Interruptible



- empty system interface
- used by lower-level concern

```
declare parents:
     org.mmtk.* &&
     !(*Header
         || org.mmtk.utility.AllocAdvice
         || org.mmtk.policy.BasePolicy
         || org.mmtk.utility.CallSite
         || com.ibm.JikesRVM.memoryManagers.mmInterface.MM_Constants
         || org.mmtk.vm.ScanStatics
         || com.ibm.JikesRVM.memoryManagers.mmInterface.SynchronizationBarrier
         || org.mmtk.utility.TracingConstants
         || com.ibm.JikesRVM.memoryManagers.mmInterface.VM_CollectorThread
         || com.ibm.JikesRVM.memoryManagers.mmInterface.VM_GCMapIteratorGroup
         || com.ibm.JikesRVM.memoryManagers.mmInterface.VM_Handshake)
     implements VM_Uninterruptible;
```

case study analysis…
the good, the bad and/or the ugly?

- can aspects ease evolvability within this (inhospitable) systems domain?
  - can current mechanisms be reconciled?
    - good: unification yields improvement
  - design intent more clear with localization?
    - good, bad & ugly: scalability is still a question
  - are aspects fast enough?
    - good & ugly: preliminary results are promising

the good

reconciling mechanisms

- unification of mechanisms accomplished
  - crosscuts hierarchical decomposition
  - subsumes global flags/empty interfaces/preprocessor
- improvement in configurability relative to original
- eg., GCSpy as an aspect
    - (un)pluggable at compile time
    - does not require system reconfiguration
    - all-or-nothing (no flag checking)
    - possibility to leverage dynamic aspects

the good

design intent

- localization yields improvement
  - internal structure is more clear
  - external interaction is explicit
- better separation of concerns
  - code that is crosscut benefits
- eg., prepare/release protocol
  - internal (FSM) structure is consistent across *plans*
  - possibility to leverage structure to facilitate dynamic plan switching

the good
(&& ugly)

performance

- benchmarks for GC are an active topic of research
- preliminary results from new Dacapo benchmark suite – but still noisy…

Benchmark	MMTK	MMTKao
Antlr	554115 msec	+/-10%
Fop	17303 msec	+/-10%
Jython	746678 msec	+/-8%
Pmd	460094 msec	+/-9%
Ps	842817 msec	+/-20%

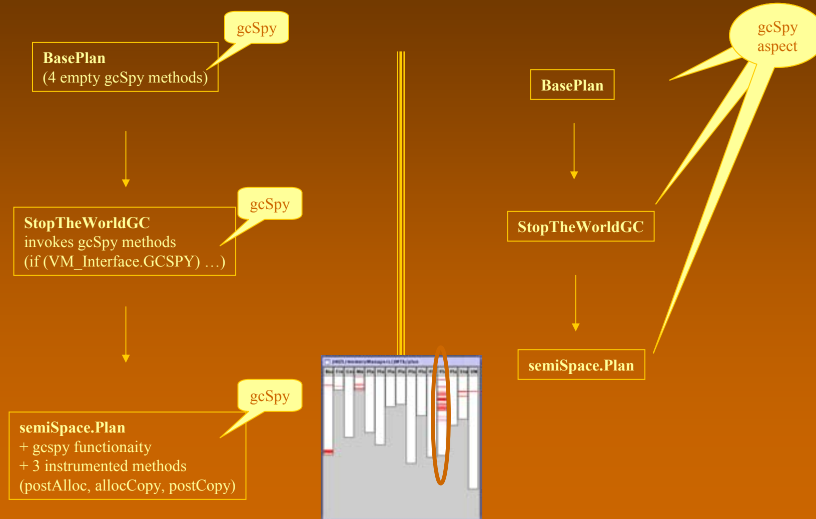
design intent

the bad and/or ugly

- evolution → ↑growth (↑plans)
- scalability is key for future evolution
  - in particular, as ↑plans
- will the internal structure scale?
  - all examples appear to benefit from composition of related aspects (not yet implemented)

GCSpy: scalability

the bad and/or ugly



GCSpy: scalability

the bad and/or ugly

BasePlan
(4 empty gcSpy methods)

gcSpy

StopTheWorldGC
invokes gcSpy methods
(if (VM_Interface.GCSPY) …)

gcSpy

gcSpy

(multiplan system…)

semiSpace.Plan
+ gcspy functionaity
+ 3 instrumented methods
(postAlloc, allocCopy, postCopy)

gcSpy

BasePlan

StopTheWorldGC

semiSpace.Plan

gcSpy core

gcSpy plans

(multiplan system…)

prepare/release: scalability

the bad and/or ugly

BasePlan
(4 empty gcSpy methods)

StopTheWorldGC
invokes gcSpy methods
(if (VM_Interface.GCSPY) …)

p/r

p/r

(multiplan system…)

semiSpace.Plan
+ gcspy functionaity
+ 3 instrumented methods
(postAlloc, allocCopy, postCopy)

p/r

BasePlan

StopTheWorldGC

semiSpace.Plan

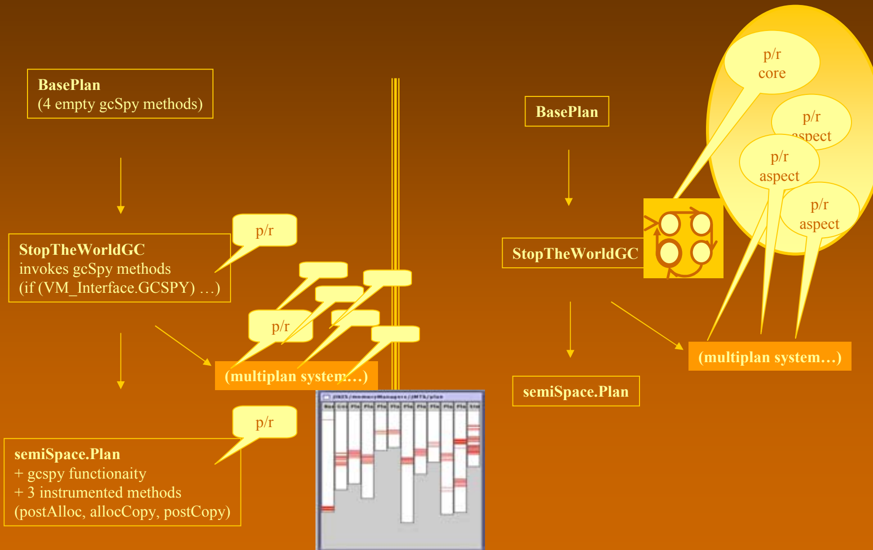
p/r aspect

p/r aspect

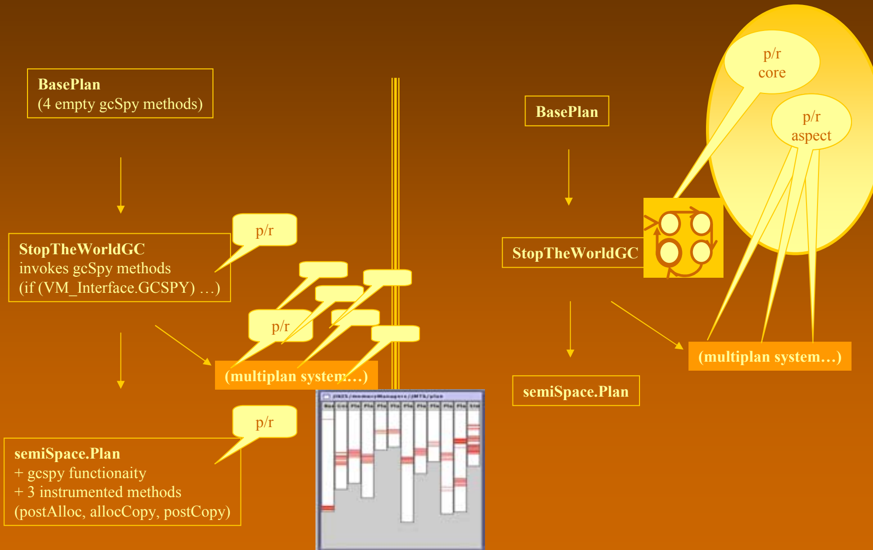
p/r aspect

p/r aspect

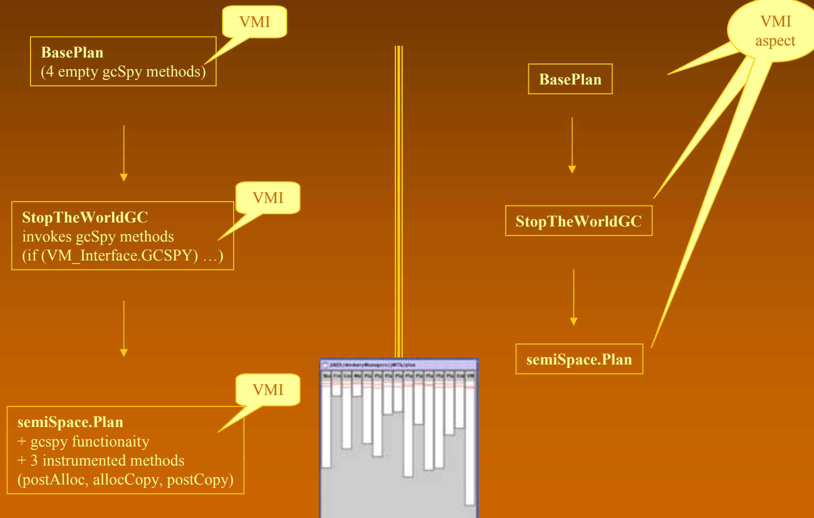
(multiplan system…)

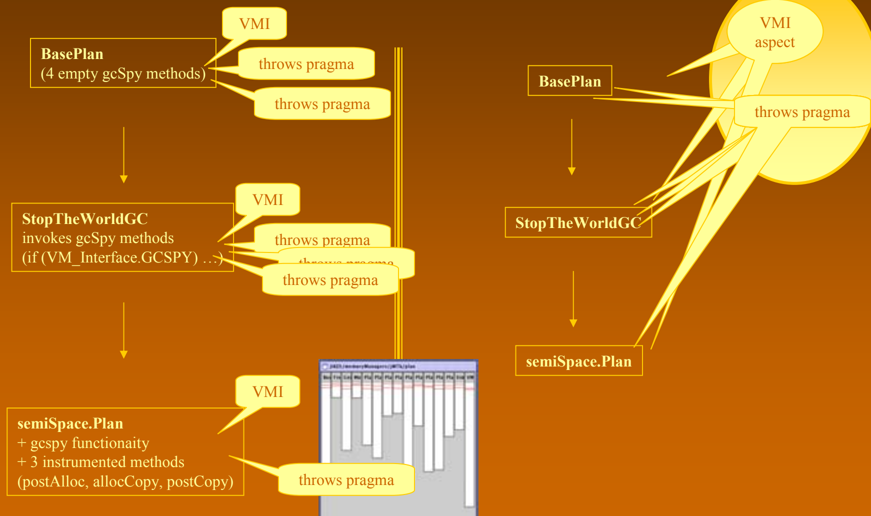


prepare/release:
scalability
the bad
and/or ugly
BasePlan
(4 empty gcSpy methods)
StopTheWorldGC
invokes gcSpy methods
(if (VM_Interface.GCSPY) …)
semiSpace.Plan
+ gcspy functionaity
+ 3 instrumented methods
(postAlloc, allocCopy, postCopy)
p/r
p/r
(multiplan system…)
p/r
BasePlan
StopTheWorldGC
semiSpace.Plan
(multiplan system…)
p/r
core
p/r
aspect
p/r
aspect
p/r
aspect



prepare/release:
scalability
the bad
and/or ugly
BasePlan
(4 empty gcSpy methods)
StopTheWorldGC
invokes gcSpy methods
(if (VM_Interface.GCSPY) …)
semiSpace.Plan
+ gcspy functionaity
+ 3 instrumented methods
(postAlloc, allocCopy, postCopy)
p/r
p/r
(multiplan system…)
p/r
BasePlan
StopTheWorldGC
semiSpace.Plan
(multiplan system…)
p/r
core
p/r
aspect


VM_Interruptible:
scalability
the bad
and/or ugly
VMI
BasePlan
(4 empty gcSpy methods)
VMI
StopTheWorldGC
invokes gcSpy methods
(if (VM_Interface.GCSPY) …)
VMI
semiSpace.Plan
+ gcspy functionaity
+ 3 instrumented methods
(postAlloc, allocCopy, postCopy)
VMI
aspect
BasePlan
StopTheWorldGC
semiSpace.Plan


VM_Interruptible:
scalability
the bad
and/or ugly
VMI
BasePlan
(4 empty gcSpy methods)
throws pragma
throws pragma
VMI
StopTheWorldGC
invokes gcSpy methods
(if (VM_Interface.GCSPY) …)
throws pragma
throws pragma
VMI
semiSpace.Plan
+ gcspy functionaity
+ 3 instrumented methods
(postAlloc, allocCopy, postCopy)
throws pragma
VMI
aspect
throws pragma
BasePlan
StopTheWorldGC
semiSpace.Plan

conclusions && ongoing work

- aspects ease system evolution by
  - reconciling existing system SOC mechanisms
  - clarifying design intent of crosscutting and interacting concerns
  - doing this efficiently
- ongoing study of evolution
  - scalable aspects as compositions
  - external interaction and evolution:
    - what happens when the code that is crosscut is completely overhauled?

questions & comments?