University of Glasgow

AspectJ for Runtime Behaviour Modification of Java Libraries

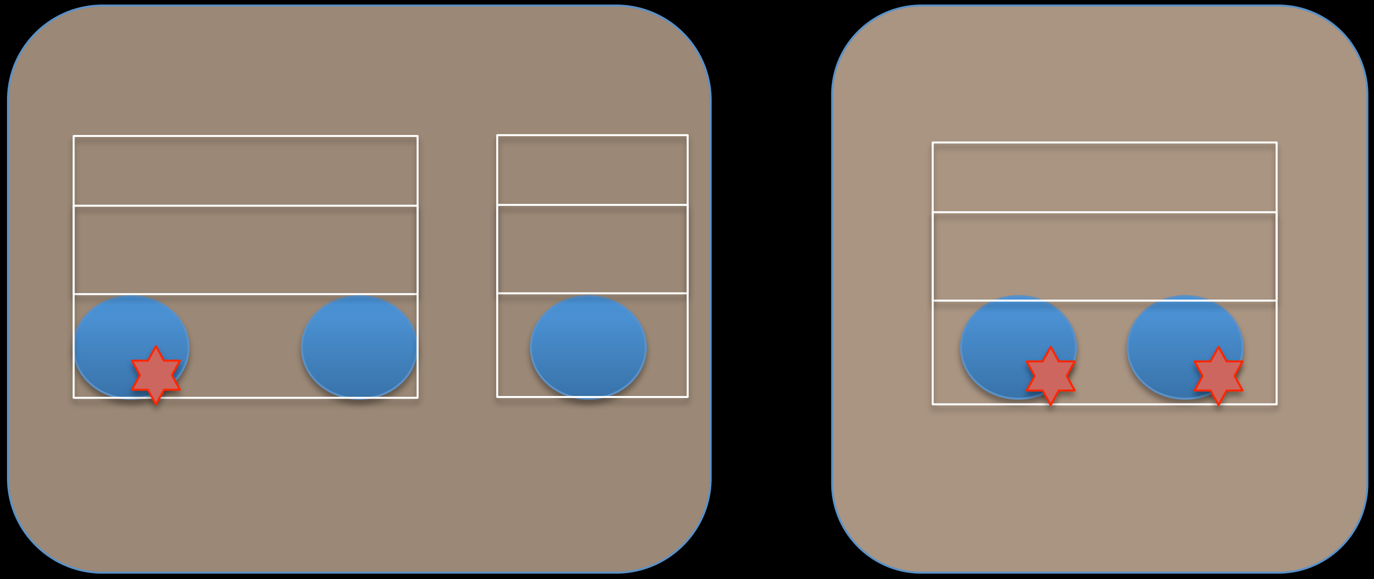
Jeremy.Singer@glasgow.ac.uk

@jsinger_compsci

1. What is AspectJ?
2. What are approximate data structures?
3. Using AspectJ to support approx data structures

1. What is AspectJ?

- Aspect-oriented programming enables clean modularization of cross-cutting concerns.



Example Aspects

- Logging code
  - log each access to a resource
  - instrument each method entry point
- Resource management code
  - reference counting
  - implicit destruction

AspectJ Syntax

- **Pointcut** – where to intercept execution

- **Advice** – what to compute

Aspect Specification (1)

- **Pointcut** – where to intercept execution

```
public aspect HelloAspect {
  pointcut greeting():
    HelloWorld.sayHello(..);
}
```

Aspect Specification (2)

- **Advice** – what to compute

```
public aspect HelloAspect {
 after() returning: greeting() {
  System.out.println(“hello world”);
 }
}
```

Is AspectJ a *domain-specific language?*

Problems with AspectJ

- confusing syntax
  - what is **`thisEnclosingJoinPointStaticPart`?**
- evolving syntax
  - incremental changes to language, tracks Java releases
- poor and inconsistent documentation
  - ask David!

2. What are approximate data structures?

Conventionally, when you store a value at an indexed array location, you expect to retrieve that same value when you look up the same location later.

In approximate data structures, this ‘contract’ is probabilistic.

Approximate data structures are useful for dynamically adjusting application heap footprints based on resource availability.

Motivate – make `OutOfMemoryErrors` extinct

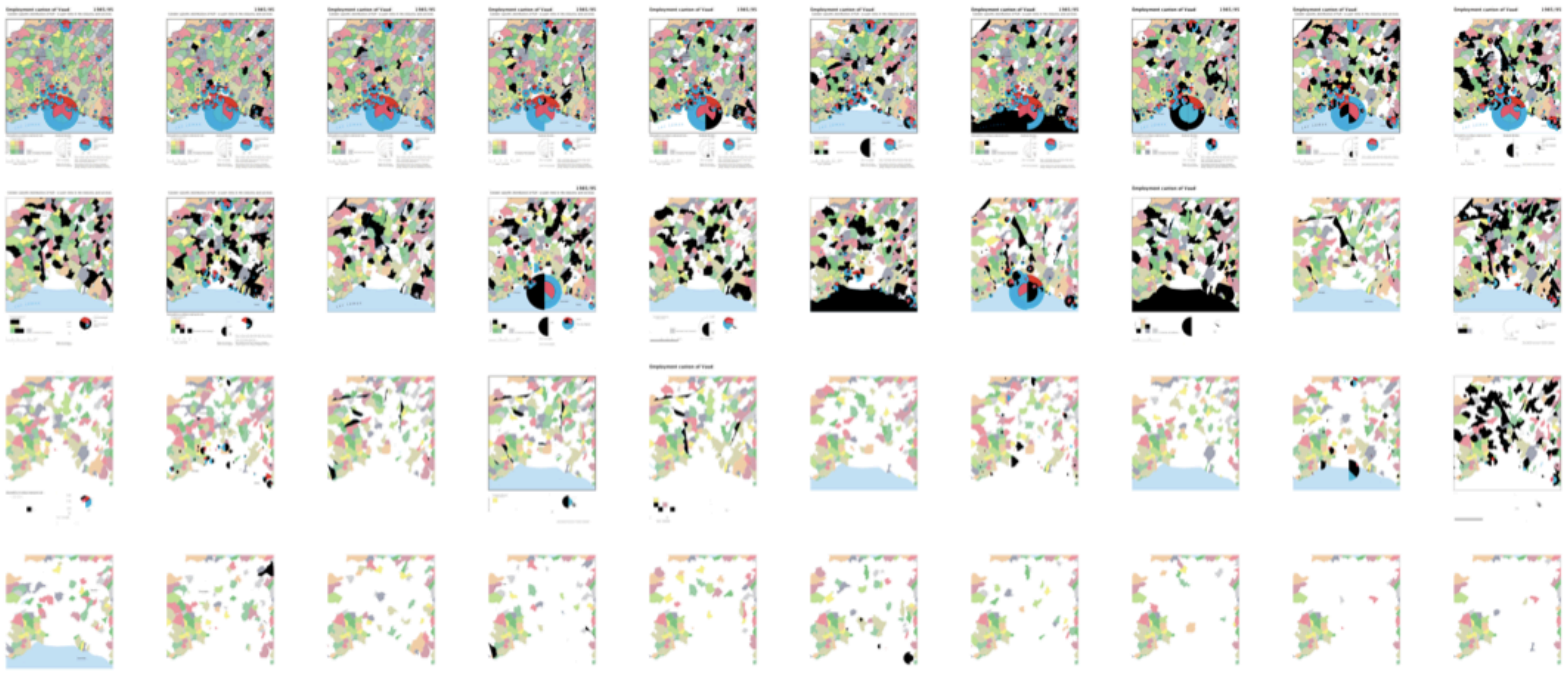
Possible application *genres* include:

- big data processing
- long-running server applications

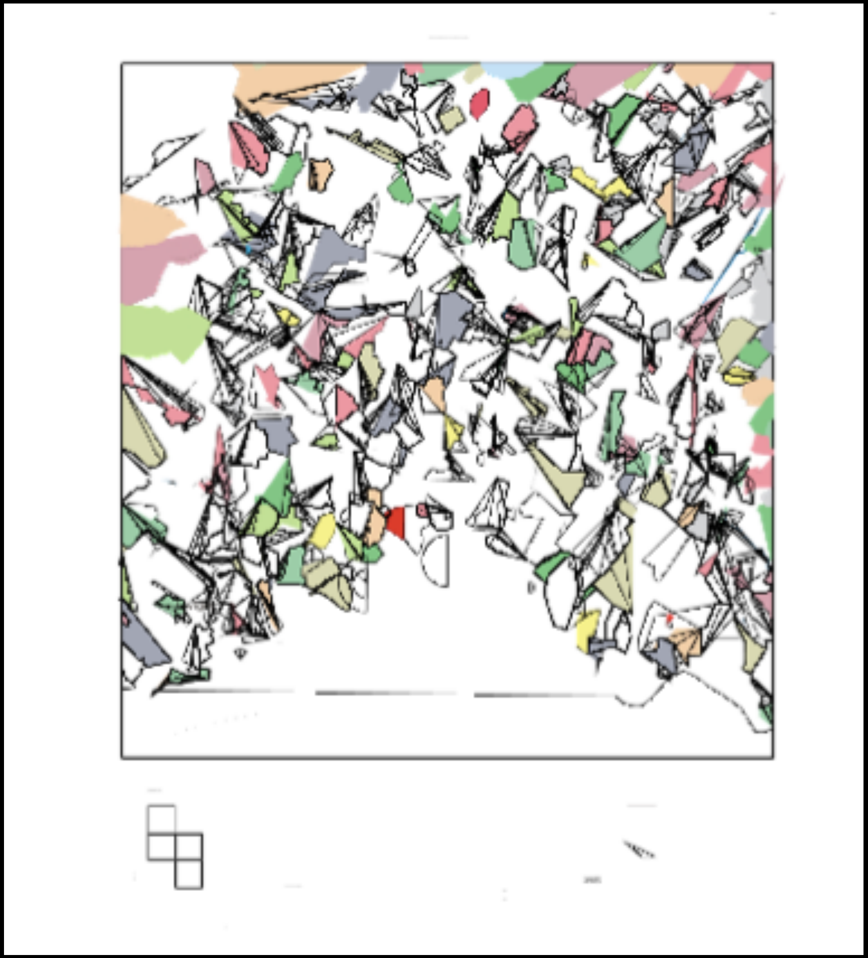
Example (1)

- the batik SVG processing tool

Example (1)

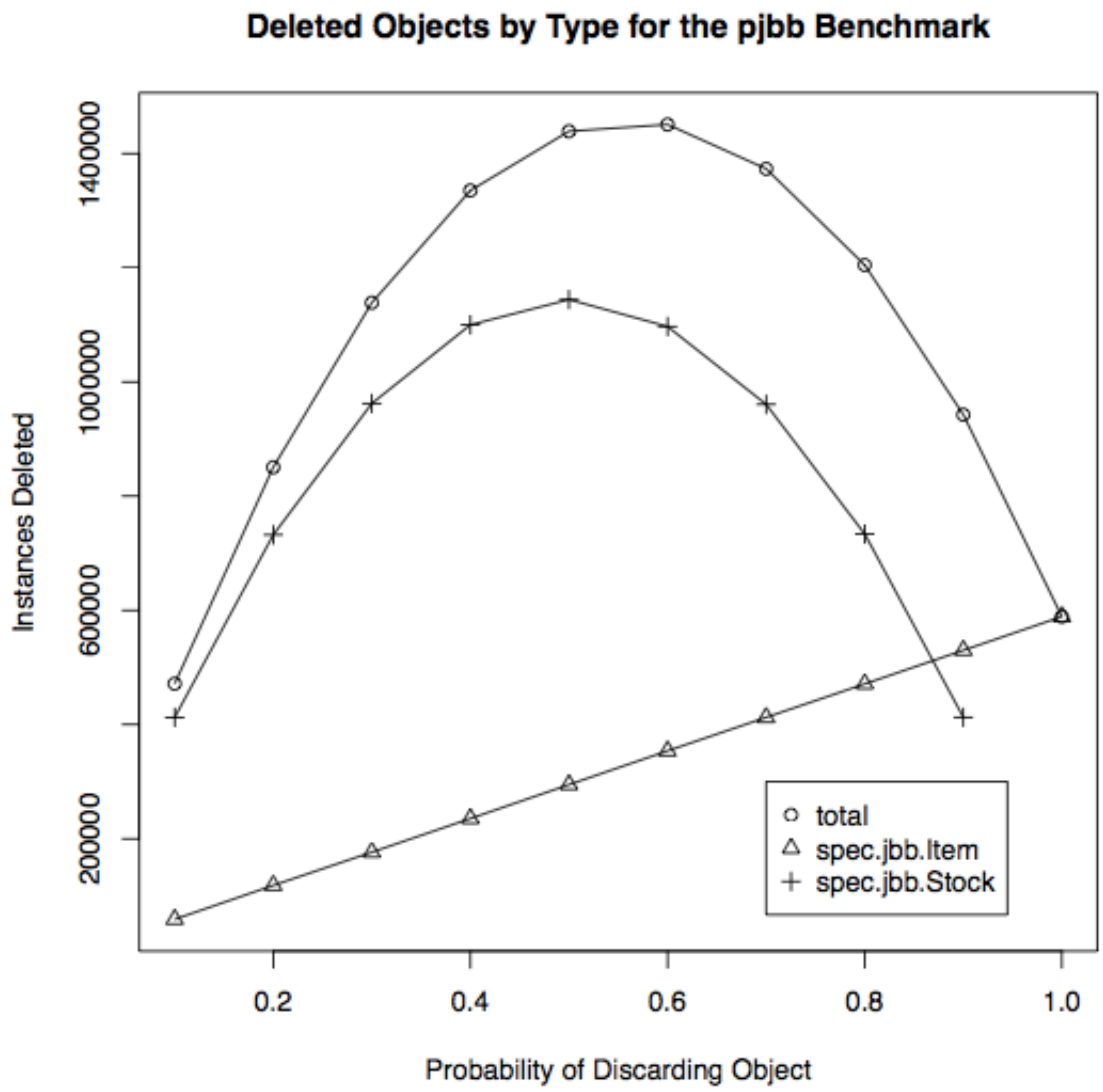


Example (1)

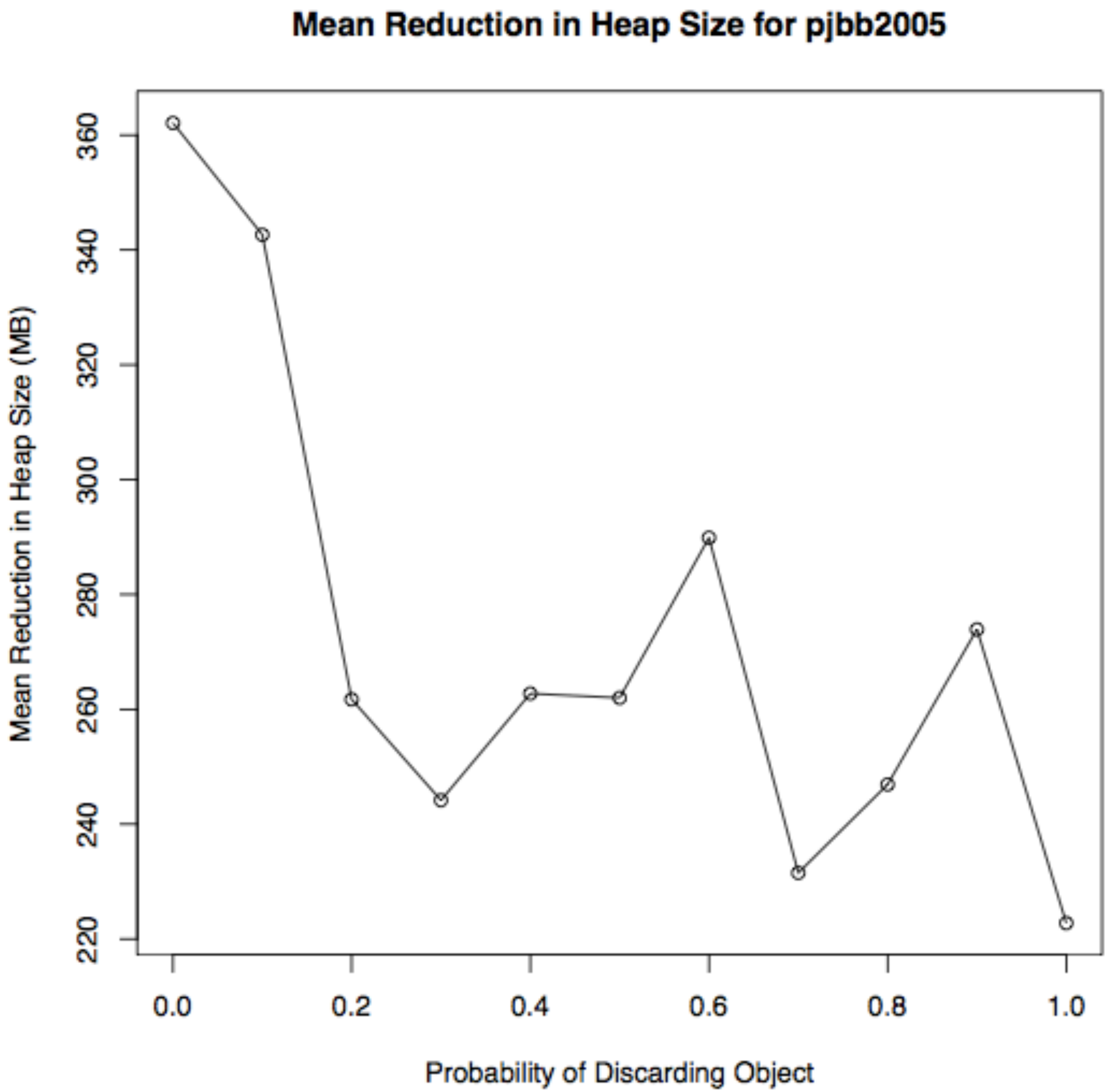


Example (2)

- the PseudoJBB benchmark



Deleted Objects by Type for the pjbb Benchmark
Instances Deleted
200000
600000
1000000
1400000
0.2
0.4
0.6
0.8
1.0
Probability of Discarding Object
○ total
△ spec.jbb.Item
+ spec.jbb.Stock


Mean Reduction in Heap Size for pjbb2005
Mean Reduction in Heap Size (MB)
220
240
260
280
300
320
340
360
0.0
0.2
0.4
0.6
0.8
1.0
Probability of Discarding Object

Our motivation

- AnyScale applications adapt their computation (e.g. parallelism, precision) based on current platform resources. The application execution may migrate to a different platform as directed by cost/benefit calculations.

- anyscale.org

3. Using AspectJ to support approximate data structures

Pragmatics

- We instrument Java Collections classes to intercept data stores (e.g. `List.add`, `HashMap.put` and modify their semantics.

- probability of interception
- alternative action – drop or replace

```
procedure ADD TO A COLLECTION($collection, input$)
    Increment $insertionCounter$ for $collection$
    if $collection$ is not empty then
        if $insertionCounter > threshold$ then
            Set $discard$ true with probability $p_{drop}$
        end if
    end if
    if $discard$ then
        for $o$ in a permutation of $collection$ do
            if $type(o) == type(input)$ then
                Replace $input$ with $o$
                Exit loop
            end if
        end for
    end if
    return proceed($collection$,$input$)
end procedure
```

Big Questions

1. Are approximate collections sensible?

2. How else can we forget data systematically?

3. Should the programmer, the compiler or the runtime be responsible for selecting data to forget?

4. Is AspectJ an appropriate language to support this concept?