# Tolerating Memory Leaks

Michael D. Bond Kathryn S. McKinley

THE UNIVERSITY OF TEXAS AT AUSTIN

## Bugs in Deployed Software

- Deployed software fails
  - Different environment and inputs → different behaviors
- Greater complexity & reliance

## Bugs in Deployed Software

- Deployed software fails
  - Different environment and inputs → different behaviors
- Greater complexity & reliance

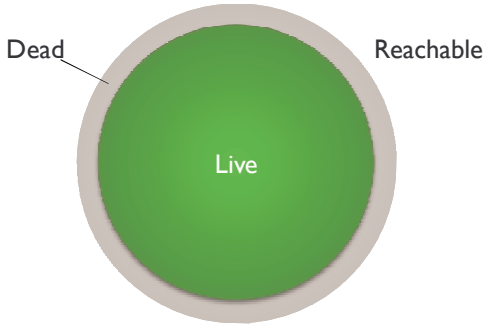
- Memory leaks are a real problem
- Fixing leaks is hard

## Memory Leaks in Deployed Systems

- Memory leaks are a real problem
  - Managed languages do not eliminate them

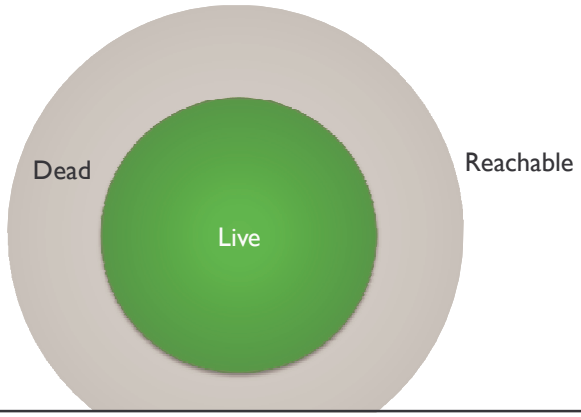
## Memory Leaks in Deployed Systems

- Memory leaks are a real problem
  - Managed languages do not eliminate them



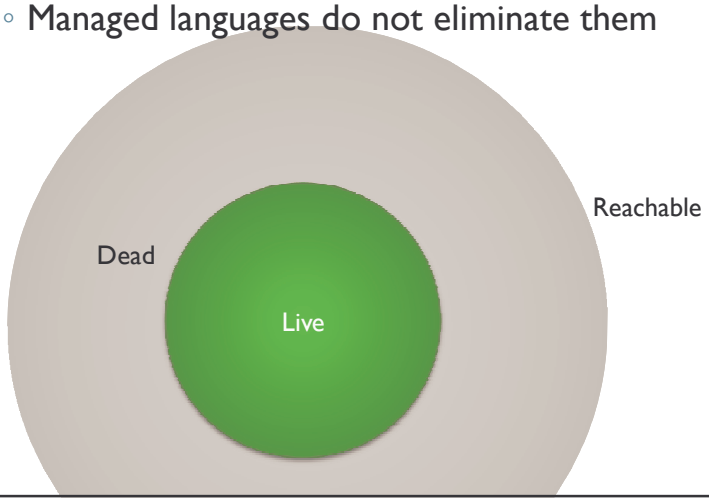

## Memory Leaks in Deployed Systems

- Memory leaks are a real problem
  - Managed languages do not eliminate them



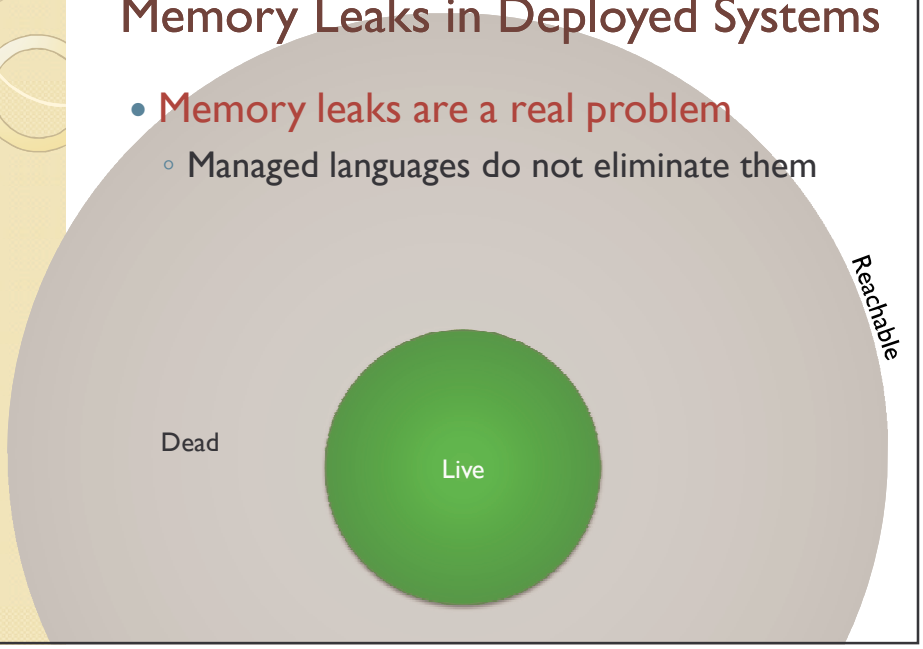
## Memory Leaks in Deployed Systems

- Memory leaks are a real problem
  - Managed languages do not eliminate them



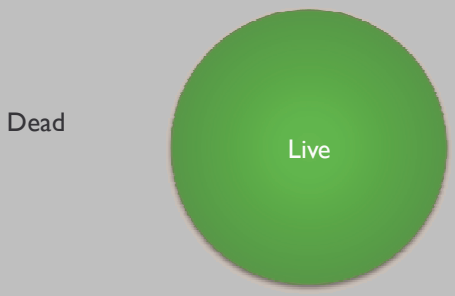

## Memory Leaks in Deployed Systems

- Memory leaks are a real problem
  - Managed languages do not eliminate them



## Memory Leaks in Deployed Systems

- Memory leaks are a real problem
  - Managed languages do not eliminate them
  - Slow & crash real programs




## Memory Leaks in Deployed Systems

- Memory leaks are a real problem
  - Managed languages do not eliminate them
  - Slow & crash real programs
  - Unacceptable for some applications

## Memory Leaks in Deployed Systems

- Memory leaks are a real problem
  - Managed languages do not eliminate them
  - Slow & crash real programs
  - Unacceptable for some applications
- Fixing leaks is hard
  - Leaks take time to materialize
  - Failure far from cause

## Example

- Driverless truck
  - 10,000 lines of C#
- Leak: past obstacles remained reachable

http://www.codeproject.com/KB/showcase/IfOnlyWedUsedANTSProfiler.aspx

## Example

- Driverless truck
  - 10,000 lines of C#
- Leak: past obstacles remained reachable
- No immediate symptoms

  "This problem was pernicious because it only showed up after 40 minutes to an hour of driving around and collecting obstacles."

http://www.codeproject.com/KB/showcase/IfOnlyWedUsedANTSProfiler.aspx

## Example

- Driverless truck
  - 10,000 lines of C#
- Leak: past obstacles remained reachable
- No immediate symptoms

  "This problem was pernicious because it only showed up after 40 minutes to an hour of driving around and collecting obstacles."
- Quick "fix": restart after 40 minutes

http://www.codeproject.com/KB/showcase/IfOnlyWedUsedANTSProfiler.aspx

## Example

- Driverless truck
  - 10,000 lines of C#
- Leak: past obstacles remained reachable
- No immediate symptoms

  "This problem was pernicious because it only showed up after 40 minutes to an hour of driving around and collecting obstacles."
- Quick "fix": restart after 40 minutes
- Environment sensitive
  - More obstacles in deployment
  - Unresponsive after 28 minutes

http://www.codeproject.com/KB/showcase/IfOnlyWedUsedANTSProfiler.aspx

## Uncertainty in Deployed Software

- Unknown leaks; unexpected failures
- Online leak diagnosis helps
  - Too late to help failing systems

## Uncertainty in Deployed Software

- Unknown leaks; unexpected failures
- Online leak diagnosis helps
  - Too late to help failing systems
- Also tolerate leaks

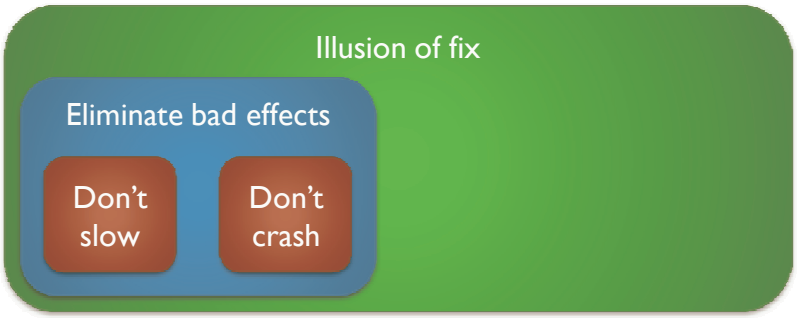
## Uncertainty in Deployed Software

- Unknown leaks; unexpected failures
- Online leak diagnosis helps
  - Too late to help failing systems
- Also tolerate leaks

Illusion of fix

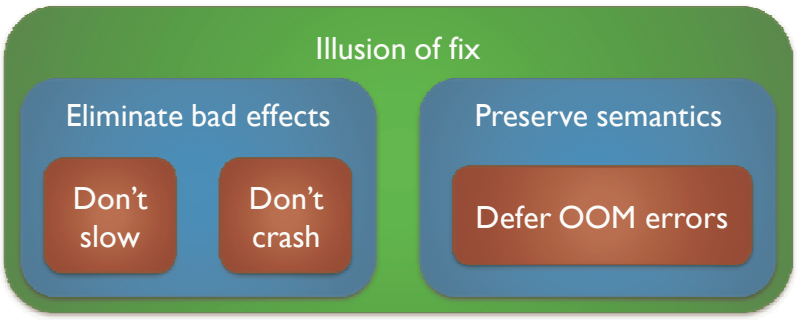
## Uncertainty in Deployed Software

- Unknown leaks; unexpected failures
- Online leak diagnosis helps
  - Too late to help failing systems
- Also tolerate leaks



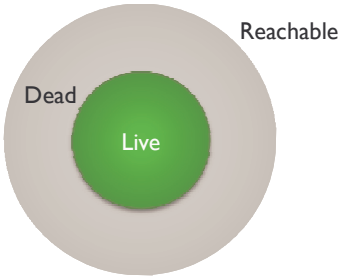

## Uncertainty in Deployed Software

- Unknown leaks; unexpected failures
- Online leak diagnosis helps
  - Too late to help failing systems
- Also tolerate leaks



## Predicting the Future

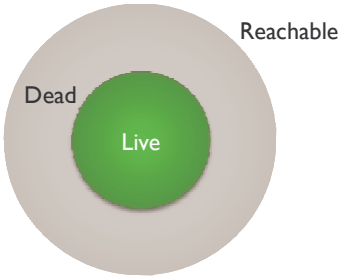
- Dead objects ⇔ not used again
- Highly stale objects ⇨ likely leaked




## Predicting the Future

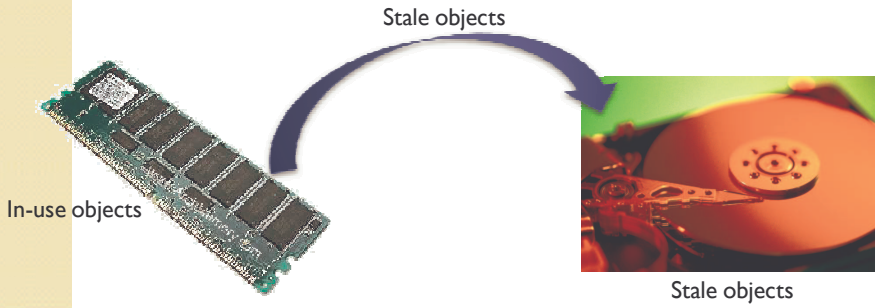
- Dead objects ⇔ not used again
- Highly stale objects ⇨ likely leaked

[Chilimbi & Hauswirth '04]
[Qin et al. '05]
[Bond & McKinley '06]

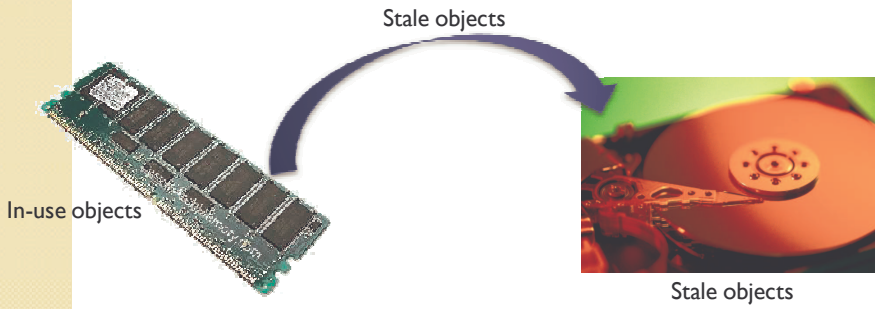


# Tolerating Leaks with **Melt**

- Move **highly stale** objects to disk
  - Much larger than memory
  - Time & space proportional to live memory
  - Preserve semantics



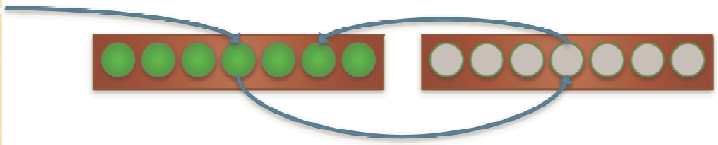

# Sounds like Paging!



## Sounds like Paging!

- Paging insufficient for managed languages
  - Need object granularity

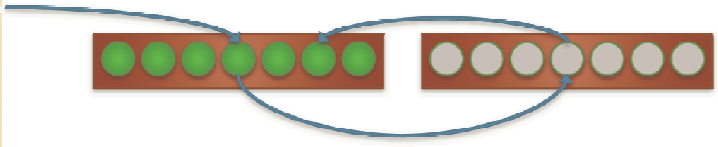
  - GC's working set is all reachable objects



## Sounds like Paging!

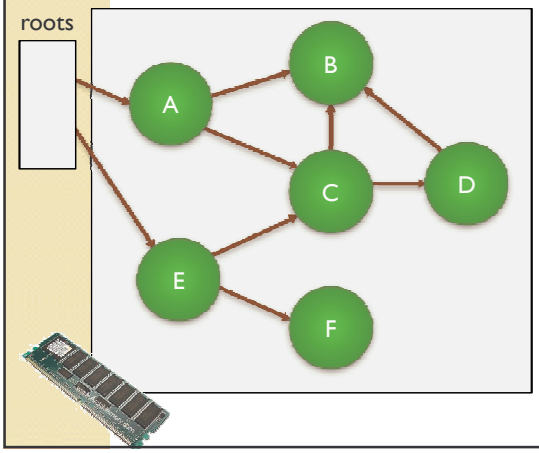
- Paging insufficient for managed languages
  - ☒ Need object granularity

  - ☑ GC's working set is all reachable objects

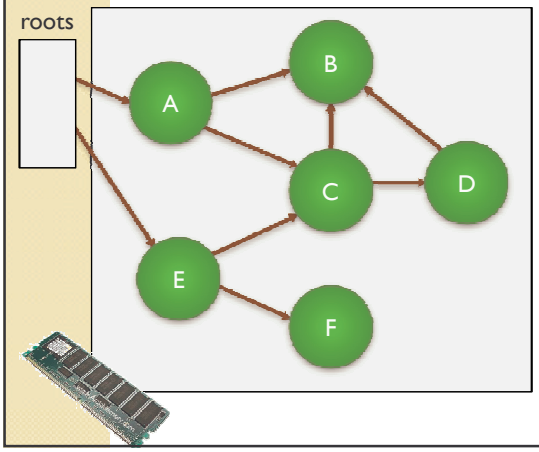


Bookmarking collection [Hertz et al. '05]

## Challenge #1: How does Melt identify stale objects?



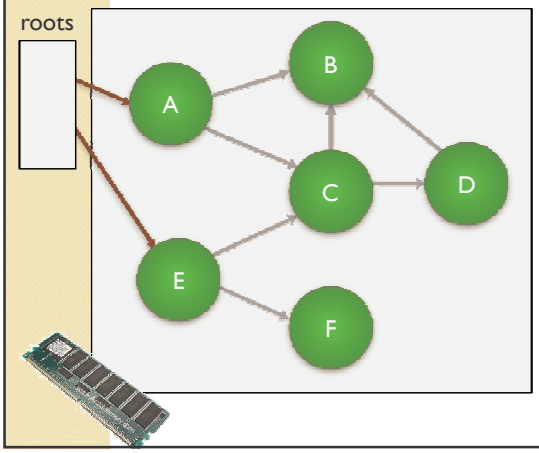

## Challenge #1: How does Melt identify stale objects?




GC:

```
for all fields a.f
  a.f |= 0x1;
```

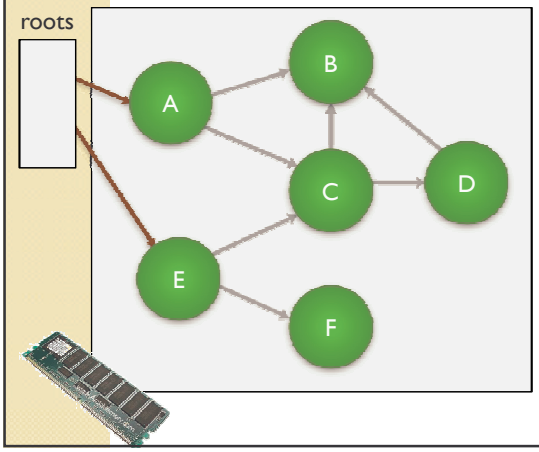
## Challenge #1: How does Melt identify stale objects?




GC:

```
for all fields a.f
  a.f |= 0x1;
```

## Challenge #1: How does Melt identify stale objects?



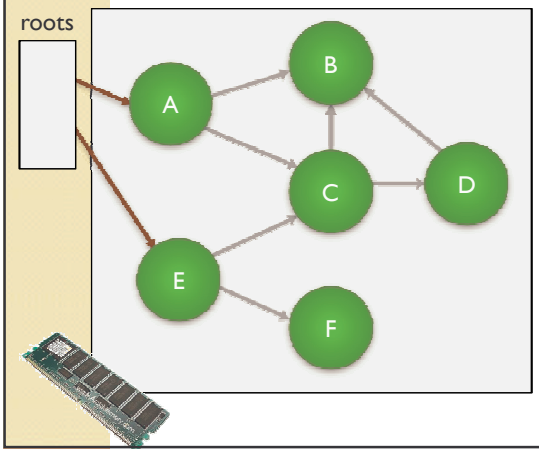

GC:

```
for all fields a.f
  a.f |= 0x1;
```

Application:

```
b = a.f;
if (b & 0x1) {
  b &= ~0x1;
  a.f = b; [atomic]
}
```

## Challenge #1: How does Melt identify stale objects?




GC:

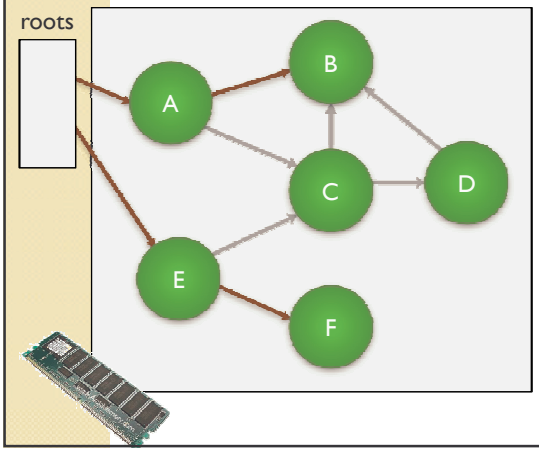
```
for all fields a.f
  a.f |= 0x1;
```

Application:

```
b = a.f;
if (b & 0x1) {
  b &= ~0x1;
  a.f = b; [atomic]
}
```

Add 6% to application time

## Challenge #1: How does Melt identify stale objects?



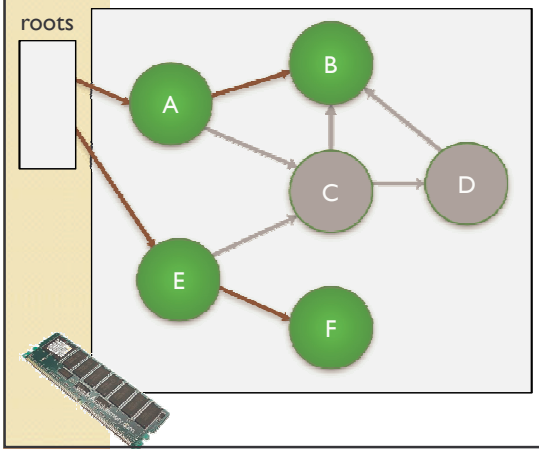

GC:

```
for all fields a.f
  a.f |= 0x1;
```

Application:

```
b = a.f;
if (b & 0x1) {
  b &= ~0x1;
  a.f = b; [atomic]
}
```

## Challenge #1: How does Melt identify stale objects?




GC:

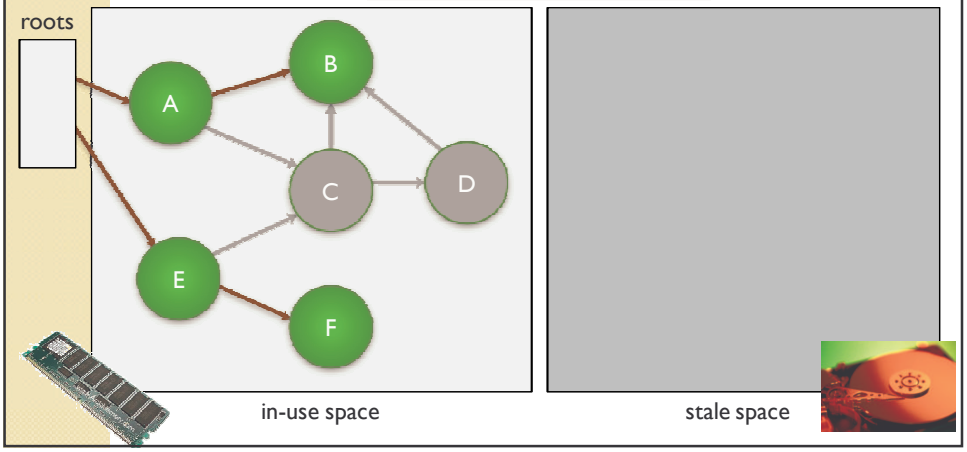
```
for all fields a.f
  a.f |= 0x1;
```

Application:

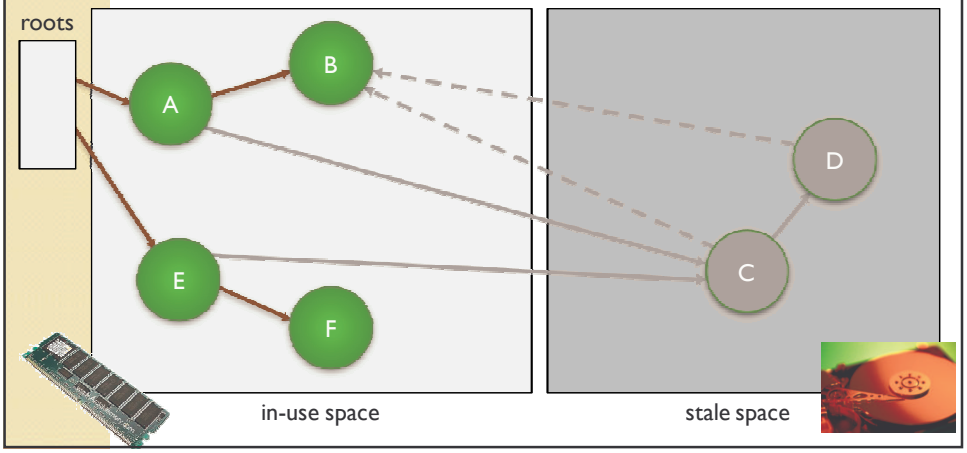
```
b = a.f;
if (b & 0x1) {
  b &= ~0x1;
  a.f = b; [atomic]
}
```

## Stale Space

Heap nearly full → move stale objects to disk

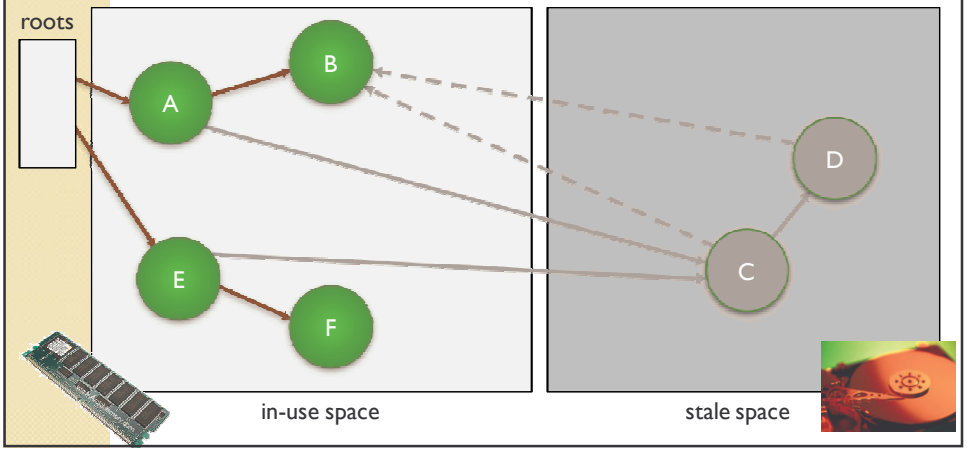


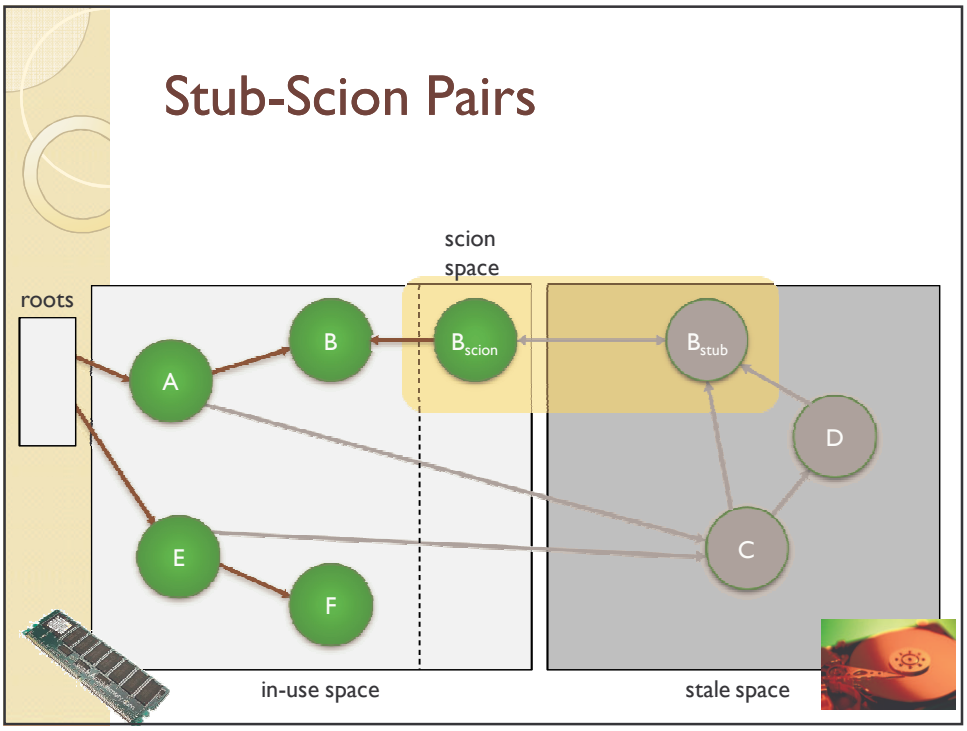
# Stale Space

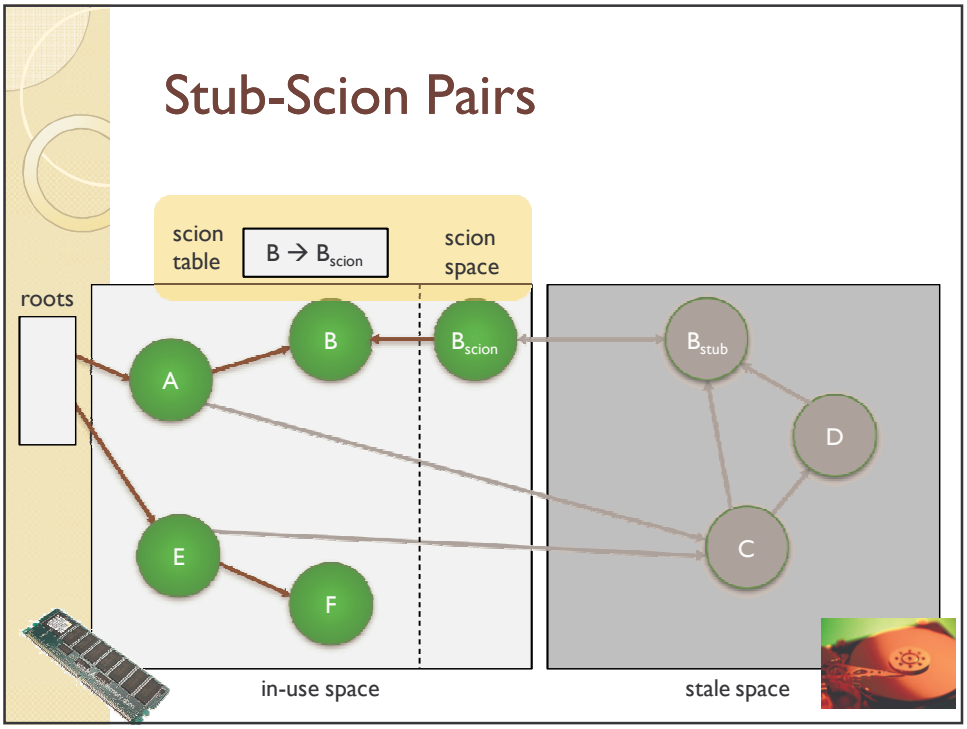


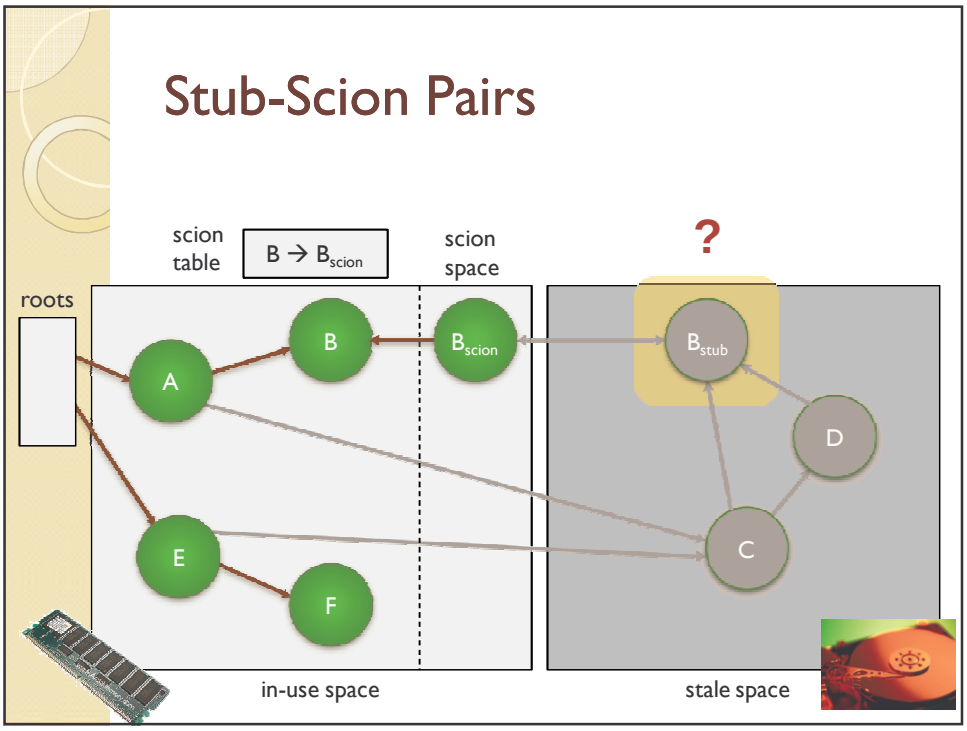

# Challenge #2

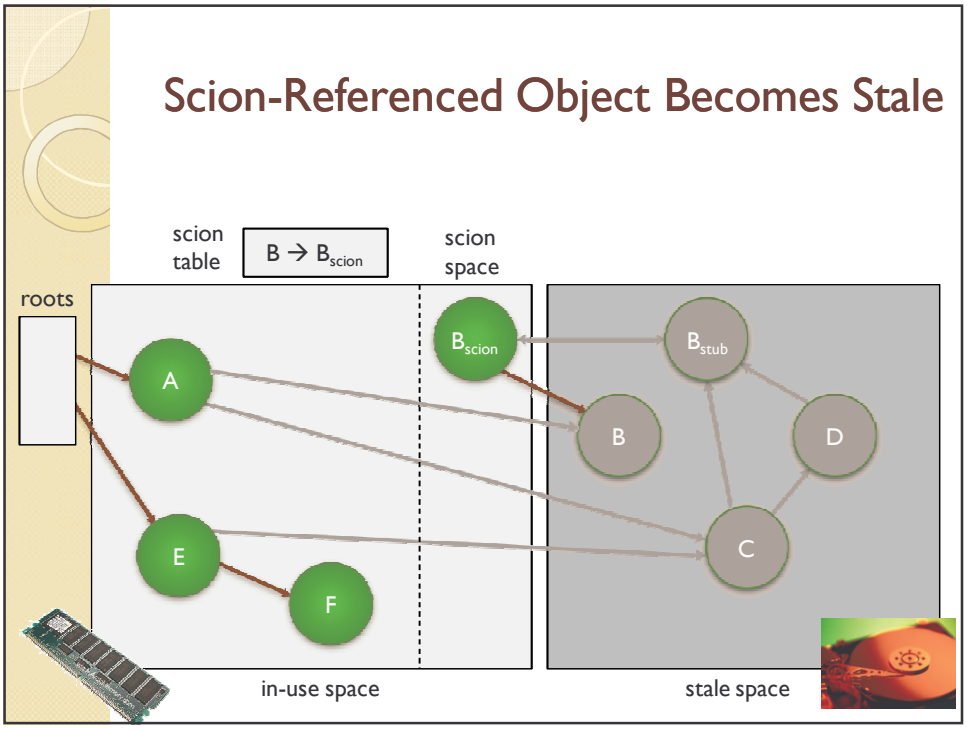
How does Melt maintain pointers?



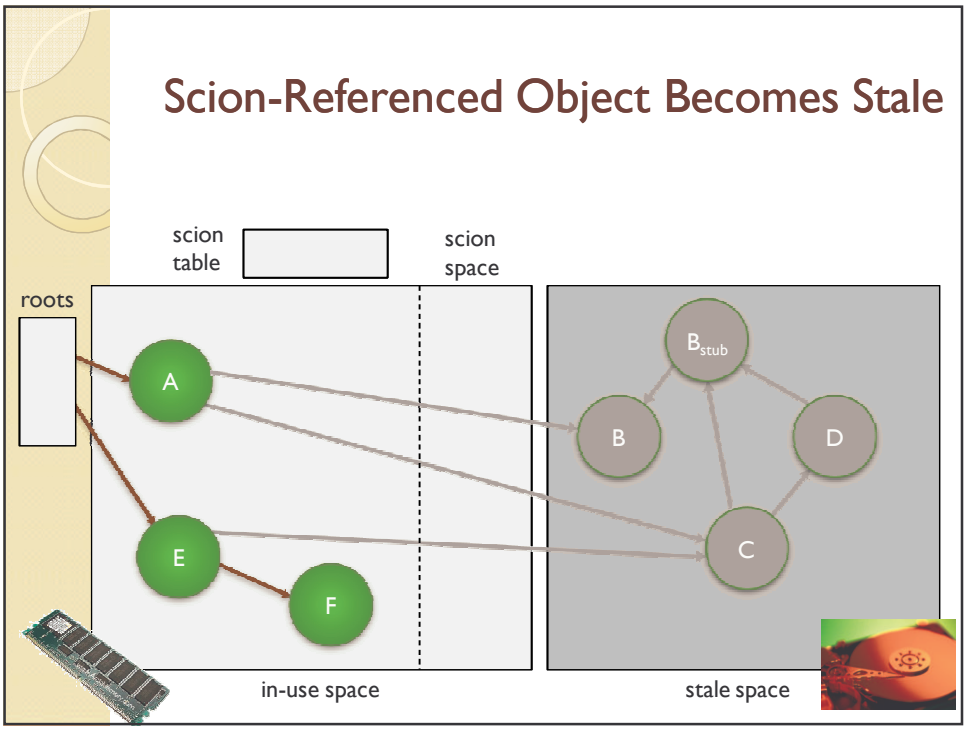

Stub-Scion Pairs
scion
space
roots
A
B
$B_{scion}$
$B_{stub}$
D
E
C
F
in-use space
stale space


Stub-Scion Pairs
scion
table
B → $B_{scion}$
scion
space
roots
A
B
$B_{scion}$
$B_{stub}$
D
E
C
F
in-use space
stale space


Stub-Scion Pairs
scion table
B → B_scion
scion space
?
roots
A
B
B_scion
B_stub
D
E
C
F
in-use space
stale space

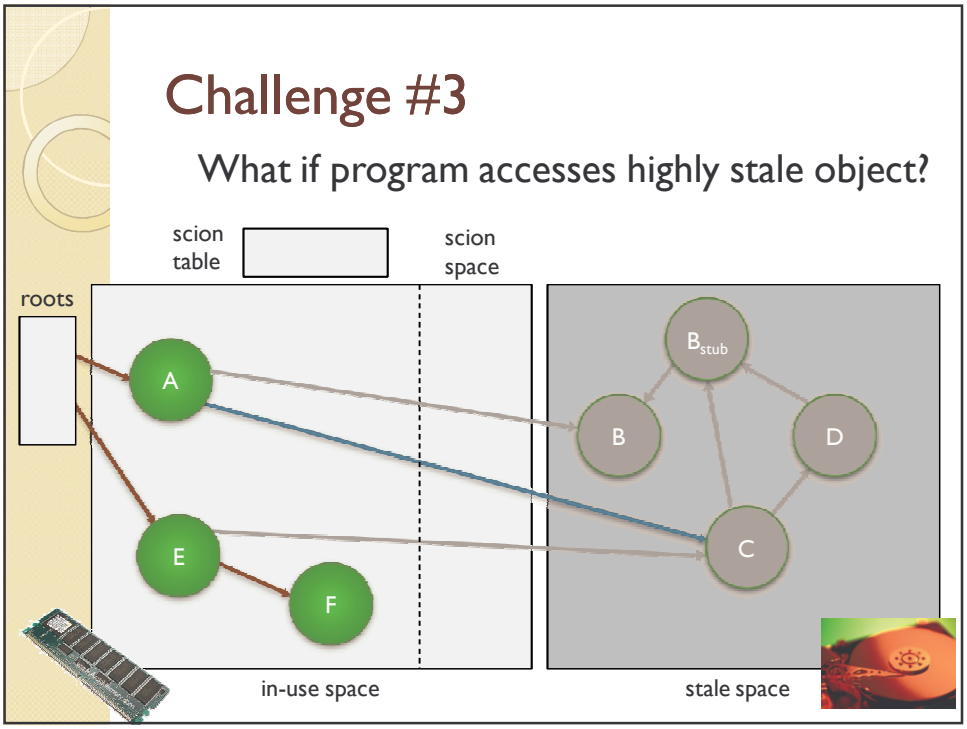

Scion-Referenced Object Becomes Stale
scion table
B → B_scion
scion space
roots
A
B_scion
B_stub
B
D
E
C
F
in-use space
stale space

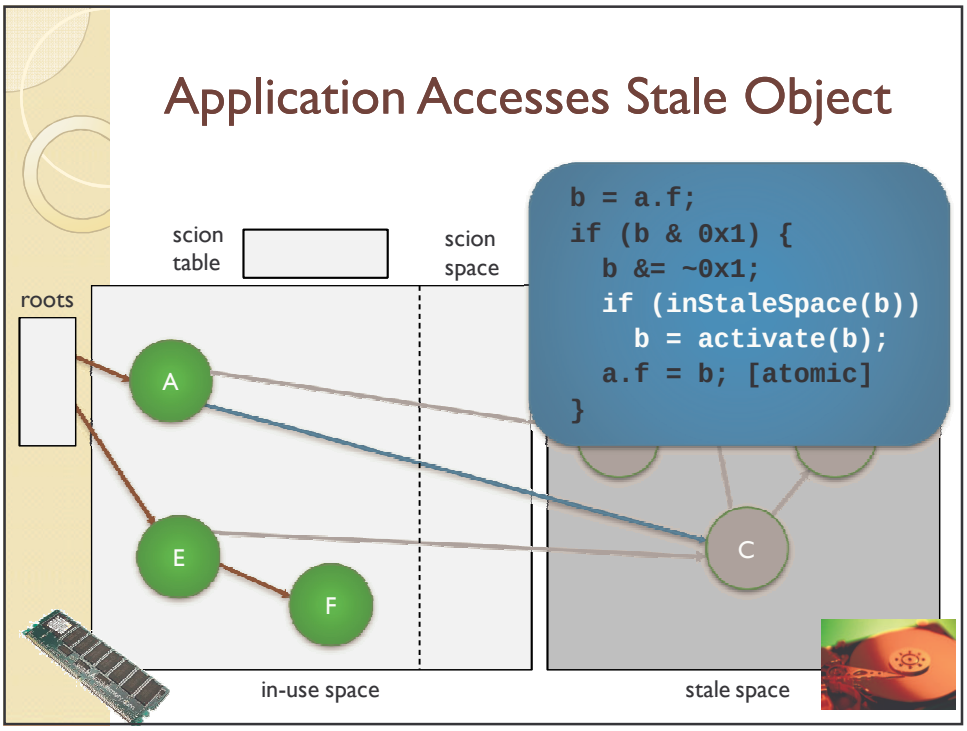
## Scion-Referenced Object Becomes Stale

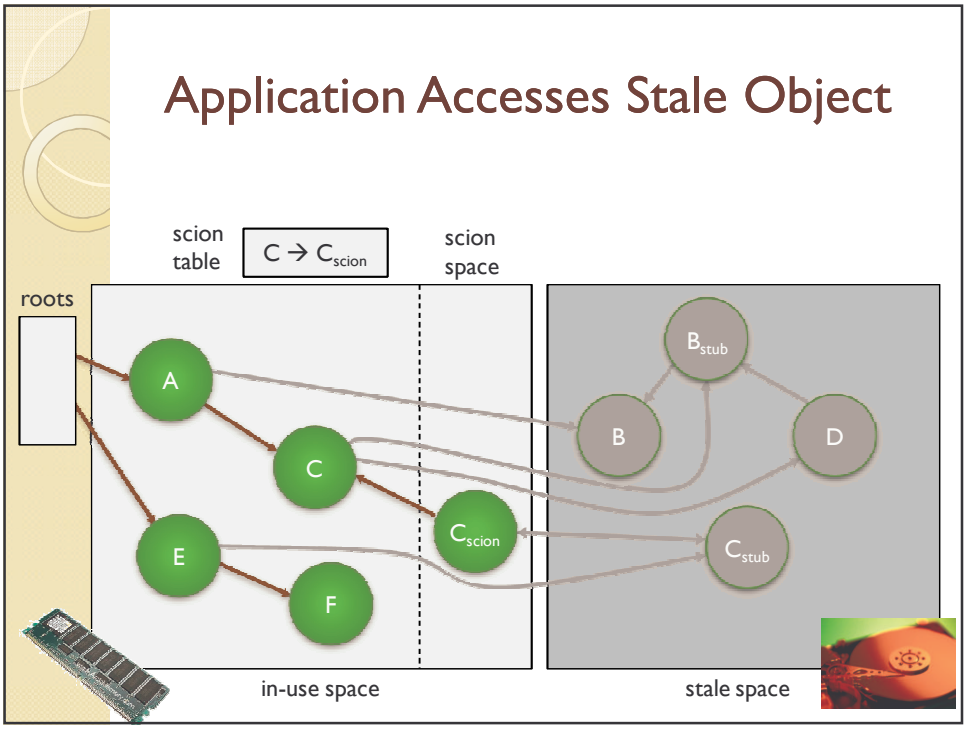


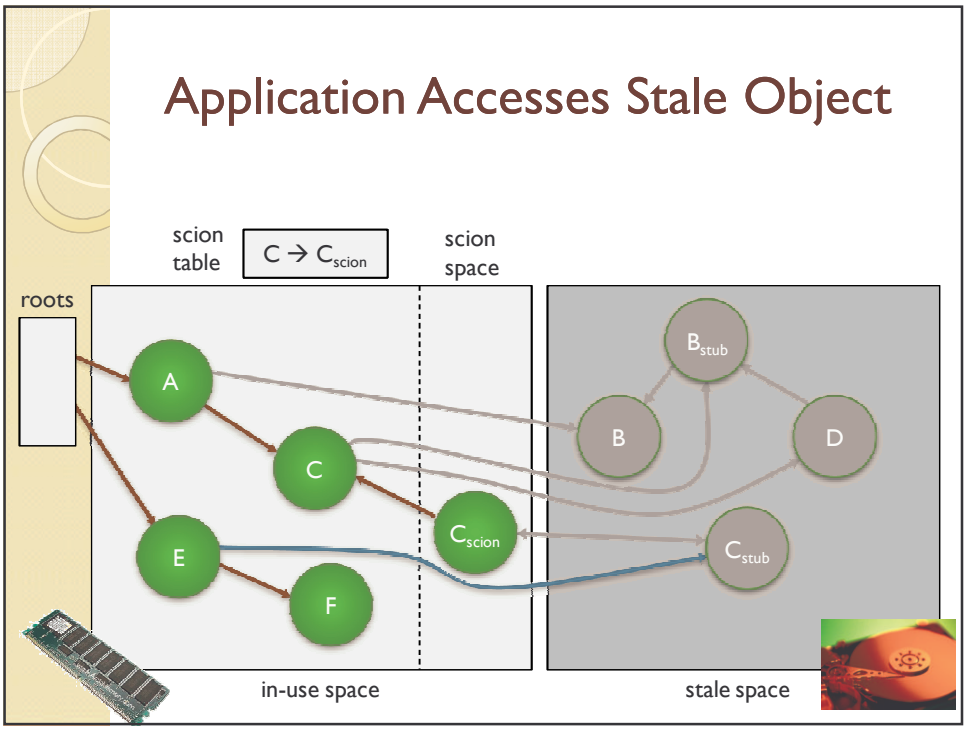

## Challenge #3

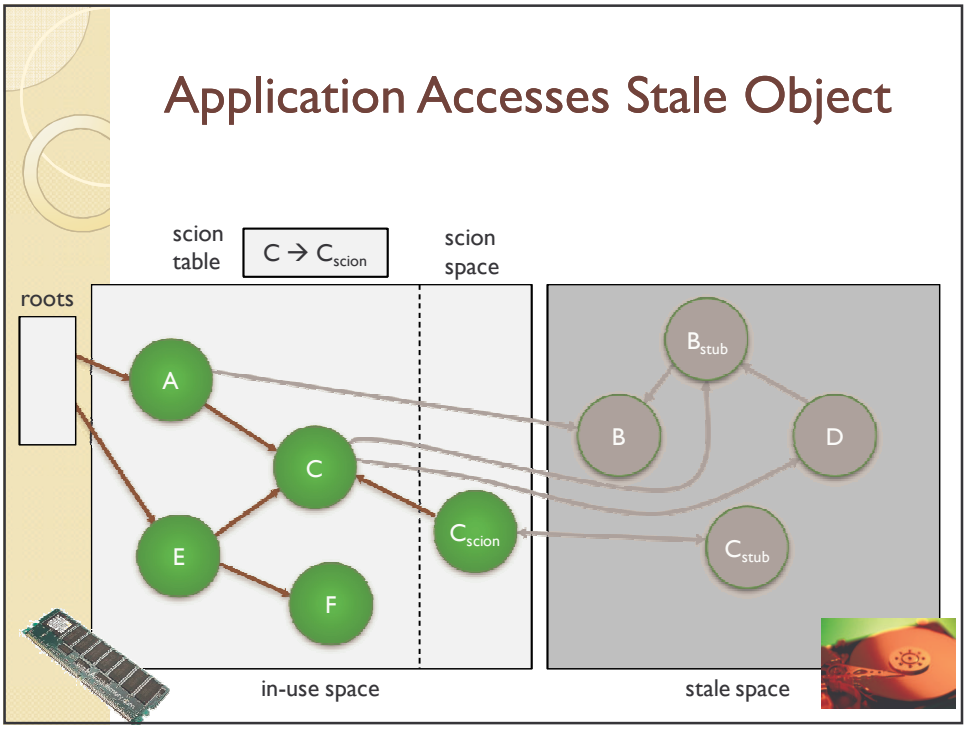
What if program accesses highly stale object?




Application Accesses Stale Object
b = a.f;
if (b & 0x1) {
b &= ~0x1;
if (inStaleSpace(b))
b = activate(b);
a.f = b; [atomic]
}
scion table
scion space
roots
A
E
F
C
in-use space
stale space


Application Accesses Stale Object
scion table
C → C_scion
scion space
roots
A
C
E
F
C_scion
B_stub
B
D
C_stub
in-use space
stale space


Application Accesses Stale Object
scion table
C → C_scion
scion space
roots
A
C
E
F
C_scion
B_stub
B
D
C_stub
in-use space
stale space

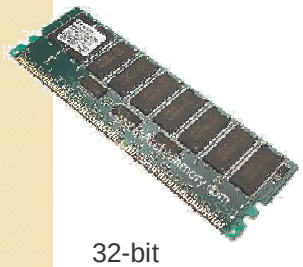

Application Accesses Stale Object
scion table
C → C_scion
scion space
roots
A
C
E
F
C_scion
B_stub
B
D
C_stub
in-use space
stale space

## Implementation

- Integrated into Jikes RVM 2.9.2
  - Works with any tracing collector
  - Evaluation uses generational copying collector

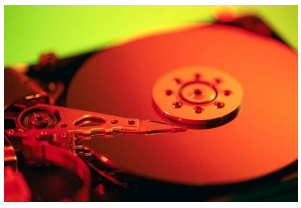
## Implementation

- Integrated into Jikes RVM 2.9.2
  - Works with any tracing collector
  - Evaluation uses generational copying collector



32-bit

2 GB

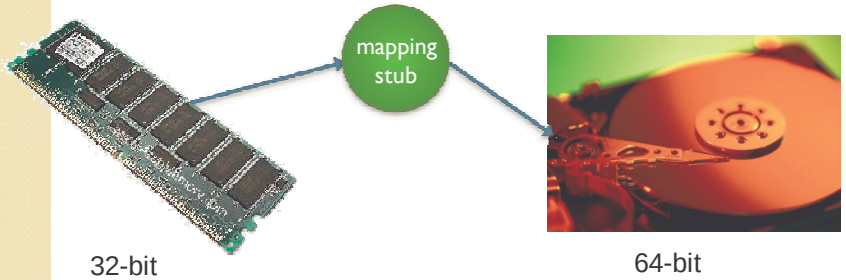


64-bit

120 GB

# Implementation

- Integrated into Jikes RVM 2.9.2
  - Works with any tracing collector
  - Evaluation uses generational copying collector




# Performance Evaluation

- Methodology
  - DaCapo, SPECjbb2000, SPECjvm98
  - Dual-core Pentium 4
  - Deterministic execution (replay)
- Results
  - 6% overhead (read barriers)
  - Stress test: still 6% overhead
    - Speedups in tight heaps (reduced GC workload)

## Tolerating Leaks

| Leak | Melt's effect |
|---|---|
| Eclipse "Diff" | Tolerates until 24-hr limit (1,000X longer) |
| Eclipse "Copy-Paste" | Tolerates until 24-hr limit (194X longer) |
| JbbMod | Tolerates until 20-hr crash (19X longer) |
| ListLeak | Tolerates until disk full (200X longer) |
| SwapLeak | Tolerates until disk full (1,000X longer) |
| MySQL | Some highly stale but in-use (74X longer) |
| Delaunay Mesh | Short-running |
| DualLeak | Heap growth is in-use (2X longer) |
| SPECjbb2000 | Heap growth is mostly in-use (2X longer) |
| Mckoi Database | Thread leak: extra support needed (2X longer) |

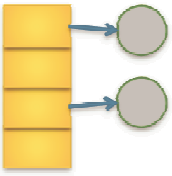
## Tolerating Leaks

| Leak | Melt's effect |
|---|---|
| Eclipse "Diff" | Tolerates until 24-hr limit (1,000X longer) |
| Eclipse "Copy-Paste" | Tolerates until 24-hr limit (194X longer) |
| JbbMod | Tolerates until 20-hr crash (19X longer) |
| ListLeak | Tolerates until disk full (200X longer) |
| SwapLeak | Tolerates until disk full (1,000X longer) |
| MySQL | Some highly stale but in-use (74X longer) |
| Delaunay Mesh | Short-running |
| DualLeak | Heap growth is in-use (2X longer) |
| SPECjbb2000 | Heap growth is mostly in-use (2X longer) |
| Mckoi Database | Thread leak: extra support needed (2X longer) |

# Tolerating Leaks

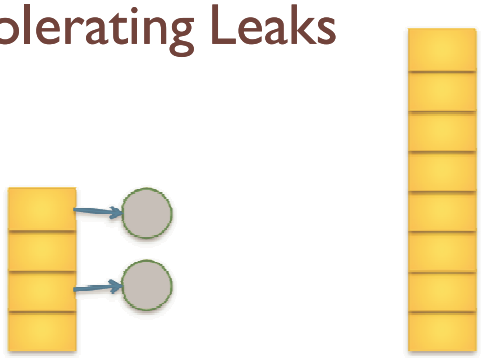
| Leak | Melt's effect |
|---|---|
| Eclipse "Diff" | Tolerates until 24-hr limit (1,000X longer) |
| Eclipse "Copy-Paste" | Tolerates until 24-hr limit (194X longer) |
| JbbMod | Tolerates until 20-hr crash (19X longer) |
| ListLeak | Tolerates until disk full (200X longer) |
| SwapLeak | Tolerates until disk full (1,000X longer) |
| MySQL | Some highly stale but in-use (74X longer) |
| Delaunay Mesh | Short-running |
| DualLeak | Heap growth is in-use (2X longer) |
| SPECjbb2000 | Heap growth is mostly in-use (2X longer) |
| Mckoi Database | Thread leak: extra support needed (2X longer) |

# Tolerating Leaks



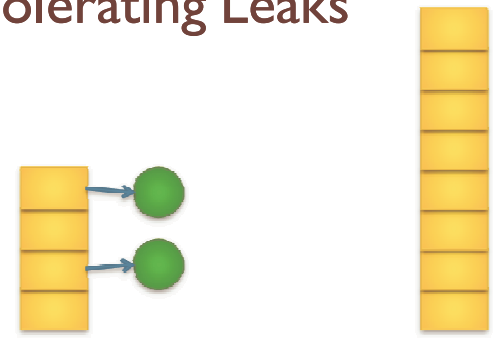
| | |
|---|---|
| MySQL | Some highly stale but in-use (74X longer) |
| Delaunay Mesh | Short-running |
| DualLeak | Heap growth is in-use (2X longer) |
| SPECjbb2000 | Heap growth is mostly in-use (2X longer) |
| Mckoi Database | Thread leak: extra support needed (2X longer) |

# Tolerating Leaks



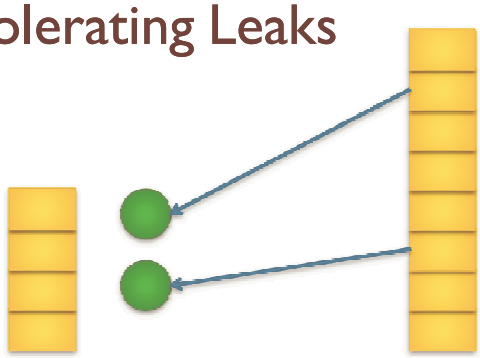
| | |
|---|---|
| MySQL | Some highly stale but in-use (74X longer) |
| Delaunay Mesh | Short-running |
| DualLeak | Heap growth is in-use (2X longer) |
| SPECjbb2000 | Heap growth is mostly in-use (2X longer) |
| Mckoi Database | Thread leak: extra support needed (2X longer) |

# Tolerating Leaks



| | |
|---|---|
| MySQL | Some highly stale but in-use (74X longer) |
| Delaunay Mesh | Short-running |
| DualLeak | Heap growth is in-use (2X longer) |
| SPECjbb2000 | Heap growth is mostly in-use (2X longer) |
| Mckoi Database | Thread leak: extra support needed (2X longer) |

## Tolerating Leaks

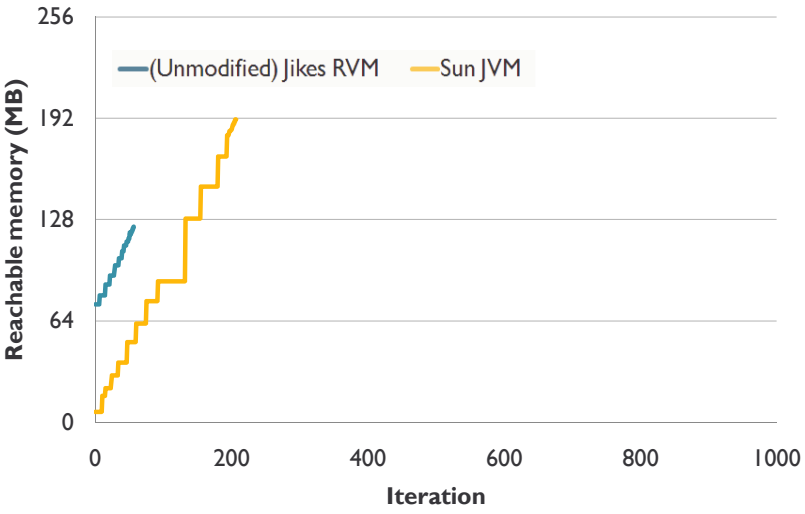


| | |
|---|---|
| MySQL | Some highly stale but in-use (74X longer) |
| Delaunay Mesh | Short-running |
| DualLeak | Heap growth is in-use (2X longer) |
| SPECjbb2000 | Heap growth is mostly in-use (2X longer) |
| Mckoi Database | Thread leak: extra support needed (2X longer) |

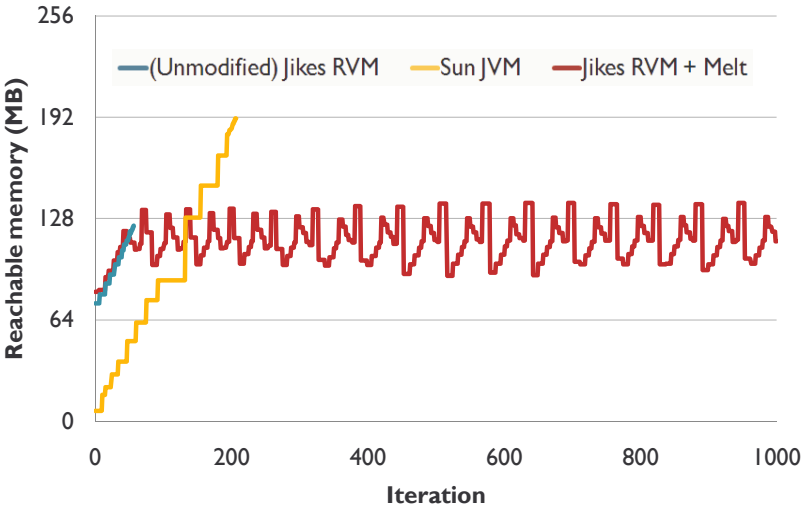
## Tolerating Leaks

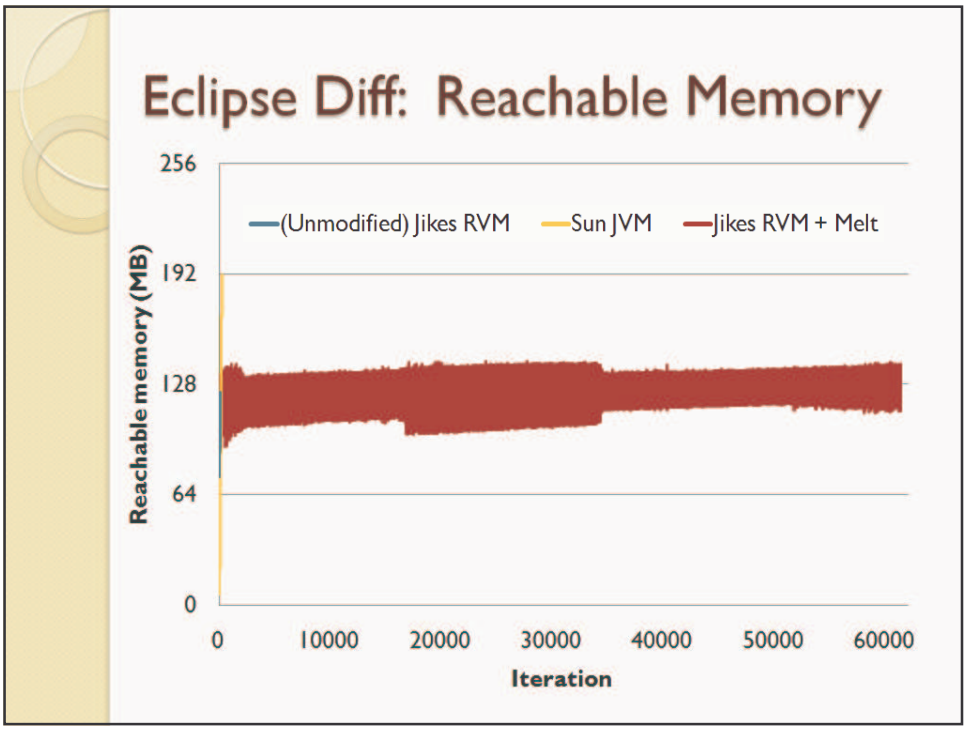
| Leak | Melt's effect |
|---|---|
| Eclipse "Diff" | Tolerates until 24-hr limit (1,000X longer) |
| Eclipse "Copy-Paste" | Tolerates until 24-hr limit (194X longer) |
| JbbMod | Tolerates until 20-hr crash (19X longer) |
| ListLeak | Tolerates until disk full (200X longer) |
| SwapLeak | Tolerates until disk full (1,000X longer) |
| MySQL | Some highly stale but in-use (74X longer) |
| Delaunay Mesh | Short-running |
| DualLeak | Heap growth is in-use (2X longer) |
| SPECjbb2000 | Heap growth is mostly in-use (2X longer) |
| Mckoi Database | Thread leak: extra support needed (2X longer) |

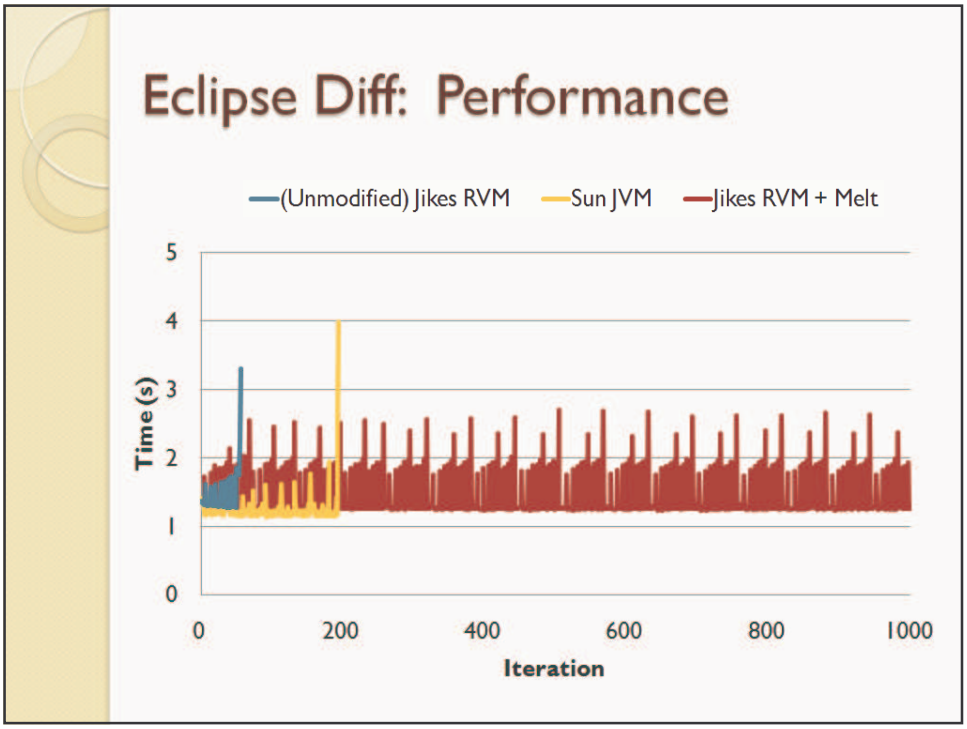
# Eclipse Diff: Reachable Memory

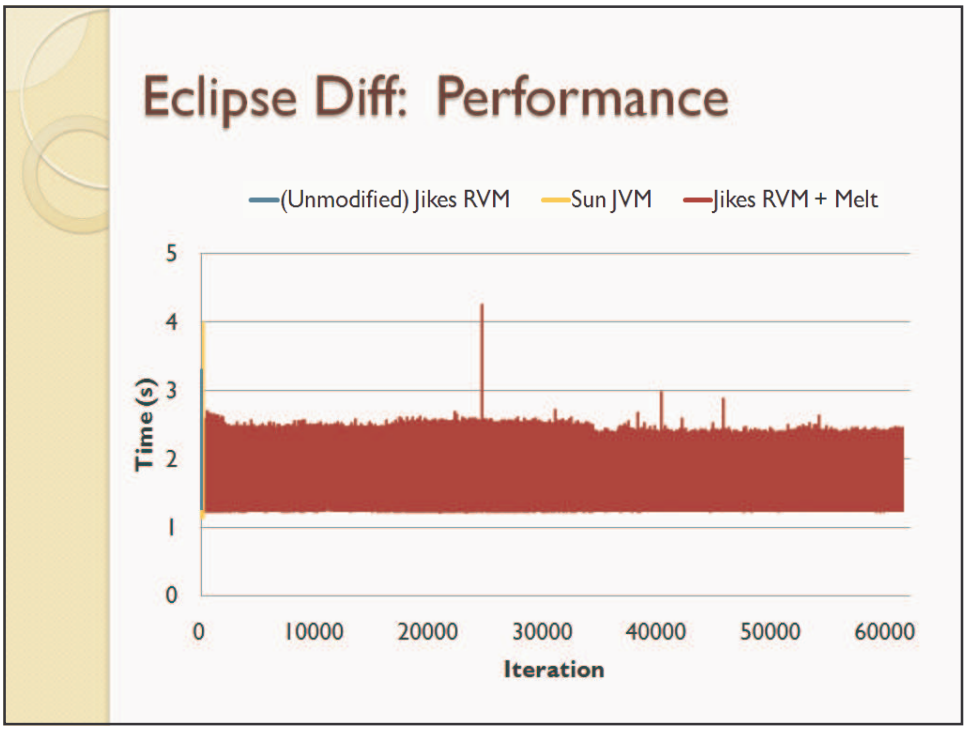


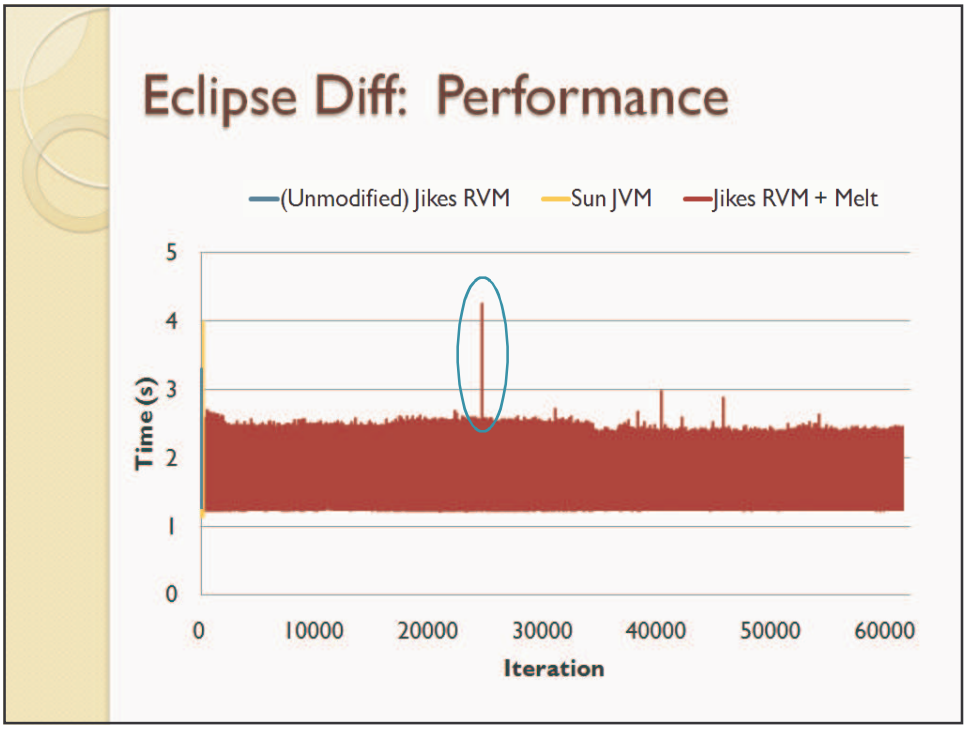

# Eclipse Diff: Reachable Memory



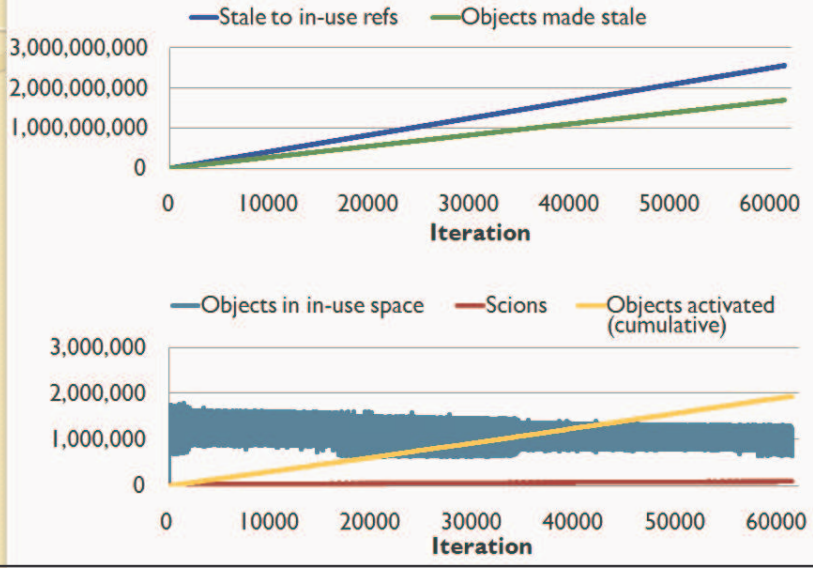

Eclipse Diff: Reachable Memory
(Unmodified) Jikes RVM
Sun JVM
Jikes RVM + Melt
Reachable memory (MB)
256
192
128
64
0
0
10000
20000
30000
40000
50000
60000
Iteration


Eclipse Diff: Performance
(Unmodified) Jikes RVM
Sun JVM
Jikes RVM + Melt
Time (s)
5
4
3
2
1
0
0
200
400
600
800
1000
Iteration


Eclipse Diff: Performance
(Unmodified) Jikes RVM
Sun JVM
Jikes RVM + Melt
Time (s)
0
1
2
3
4
5
0
10000
20000
30000
40000
50000
60000
Iteration


Eclipse Diff: Performance
(Unmodified) Jikes RVM
Sun JVM
Jikes RVM + Melt
Time (s)
0
1
2
3
4
5
0
10000
20000
30000
40000
50000
60000
Iteration

## Eclipse Diff: Melt's Scalability




## Related Work

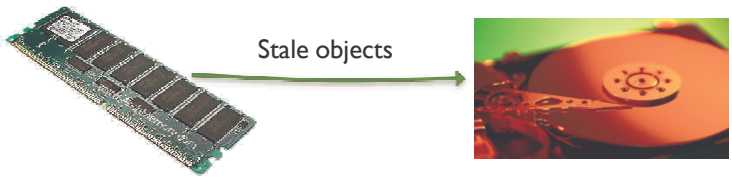
- Managed [**LeakSurvivor, Tang et al. '08**]
  [Panacea, Goldstein et al. '07, Breitgand et al. '07]
  - Don't guarantee time & space proportional to live memory
- Native [Cyclic memory allocation, Nguyen & Rinard '07]
  [Plug, Novark et al. '08]
  - Different challenges & opportunities
  - Less coverage or change semantics
- Orthogonal persistence & distributed GC
  - Barriers, swizzling, object faulting, stub-scion pairs

## Conclusion

- Finding bugs before deployment is hard


## Conclusion

- Finding bugs before deployment is hard
- Online diagnosis helps developers
- Help **users** in meantime
- Tolerate leaks with Melt: **illusion of fix**



## Conclusion

- Finding bugs before deployment is hard
- Online diagnosis helps developers
- Help **users** in meantime
- Tolerate leaks with Melt: **illusion of fix**
  - Time & space proportional to live memory
  - Preserve semantics

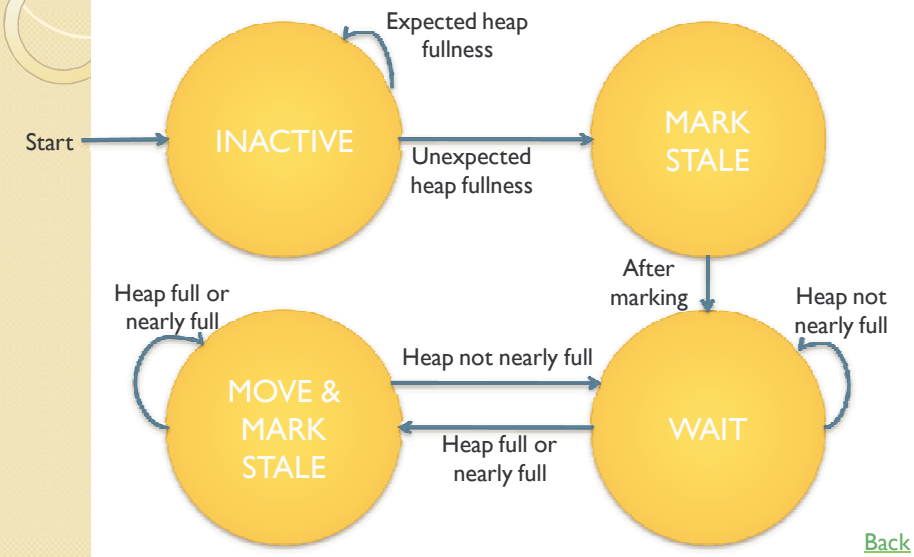
## Conclusion

- Finding bugs before deployment is hard
- Online diagnosis helps developers
- Help **users** in meantime
- Tolerate leaks with Melt: **illusion of fix**
  - Time & space proportional to live memory
  - Preserve semantics
- Buys developers time to fix leaks

Thank you!

# Backup

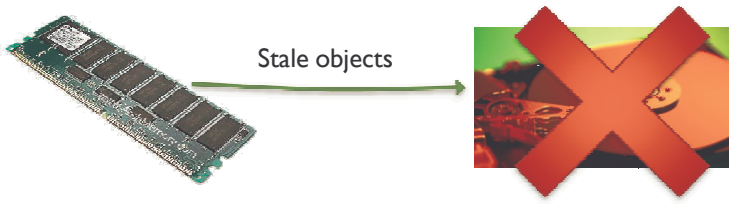
# Triggering Melt




Back

## Conclusion

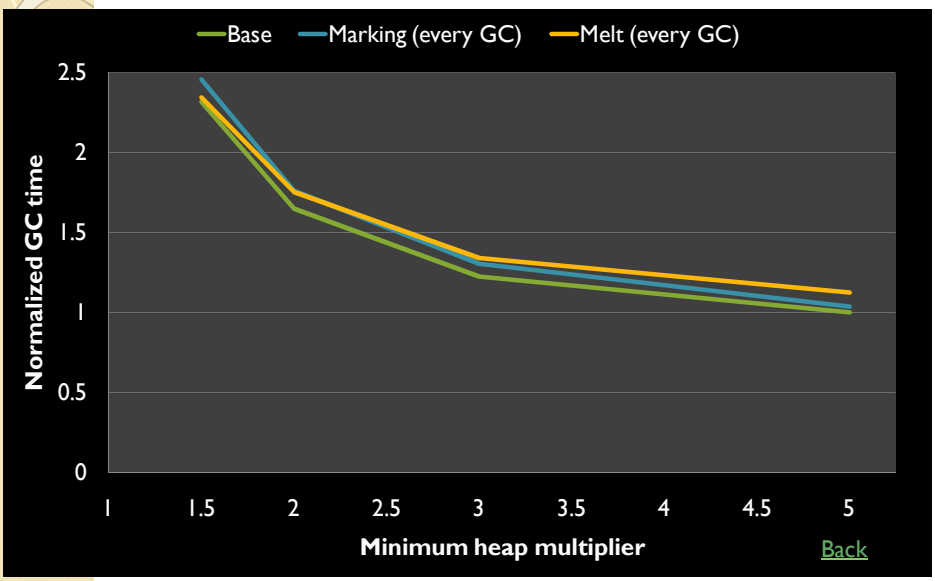
- Finding bugs before deployment is hard
- Online diagnosis helps developers
- To help **users** in meantime, tolerate bugs
- Tolerate leaks with Melt: **illusion of fix**




## Related Work: Tolerating Bugs

- Nondeterministic errors
  [Atom-Aid] [DieHard] [Grace] [Rx]
  - Memory corruption: perturb layout
  - Concurrency bugs: perturb scheduling
- General bugs
  - Ignore failing operations [FOC]
  - Need higher level, more proactive approaches

# Melt's GC Overhead




Back