Tolerating Memory Leaks

Michael Bond
Kathryn McKinley

The University of Texas at Austin

October, 2008

Presented by: Maria Martin
Concurrency and Memory Management Seminar, winter term 2010
University of Salzburg, Department of Computer Science

Outline

- Introduction to the problem (why?)
- Leak tolerance: Melt
- Tolerating memory leaks
- Implementation
- Results
- Conclusion
- Related work

Outline

- Introduction to the problem (why?)
- Leak tolerance: Melt
- Tolerating memory leaks
- Implementation
- Results
- Conclusion
- Related work

Introduction

- (*what?)* A **memory leak:**
- (*why?)* computer program **consumes memory**
- (*how?)* they are **unable to release** it back to the operating system
- (result): out of memory

Introduction

- (*what?*) A **memory leak:**
- (w
  n

MEMORY LEAKS ARE A REAL PROBLEM

FIXING THEM IS HARD

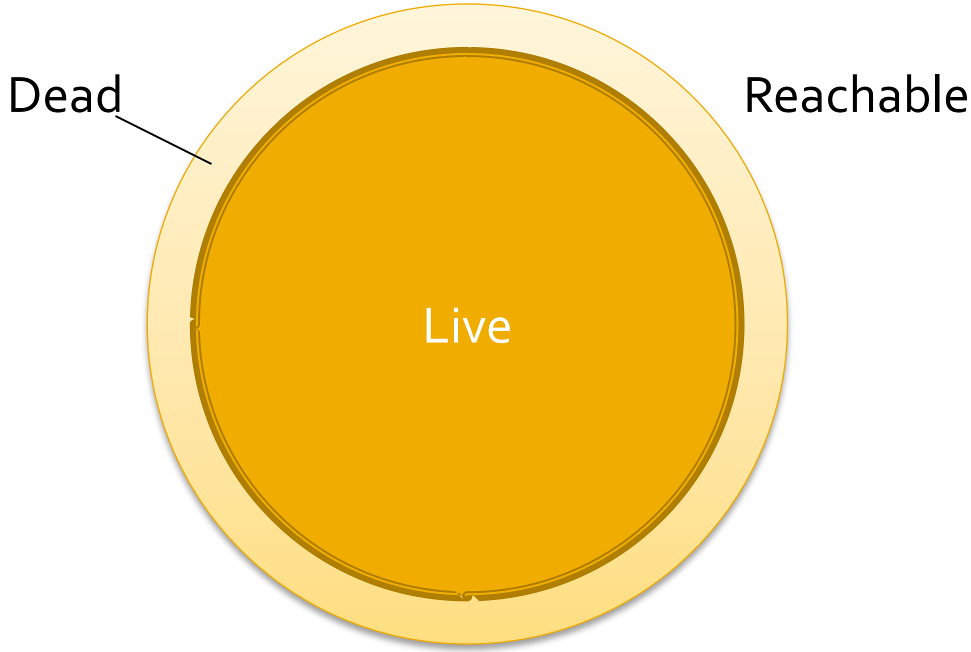
- (
  operating system
- (result): out of memory

Why?

- Managed languages do not eliminate them
  - Type safety and garbage collection (reliability)

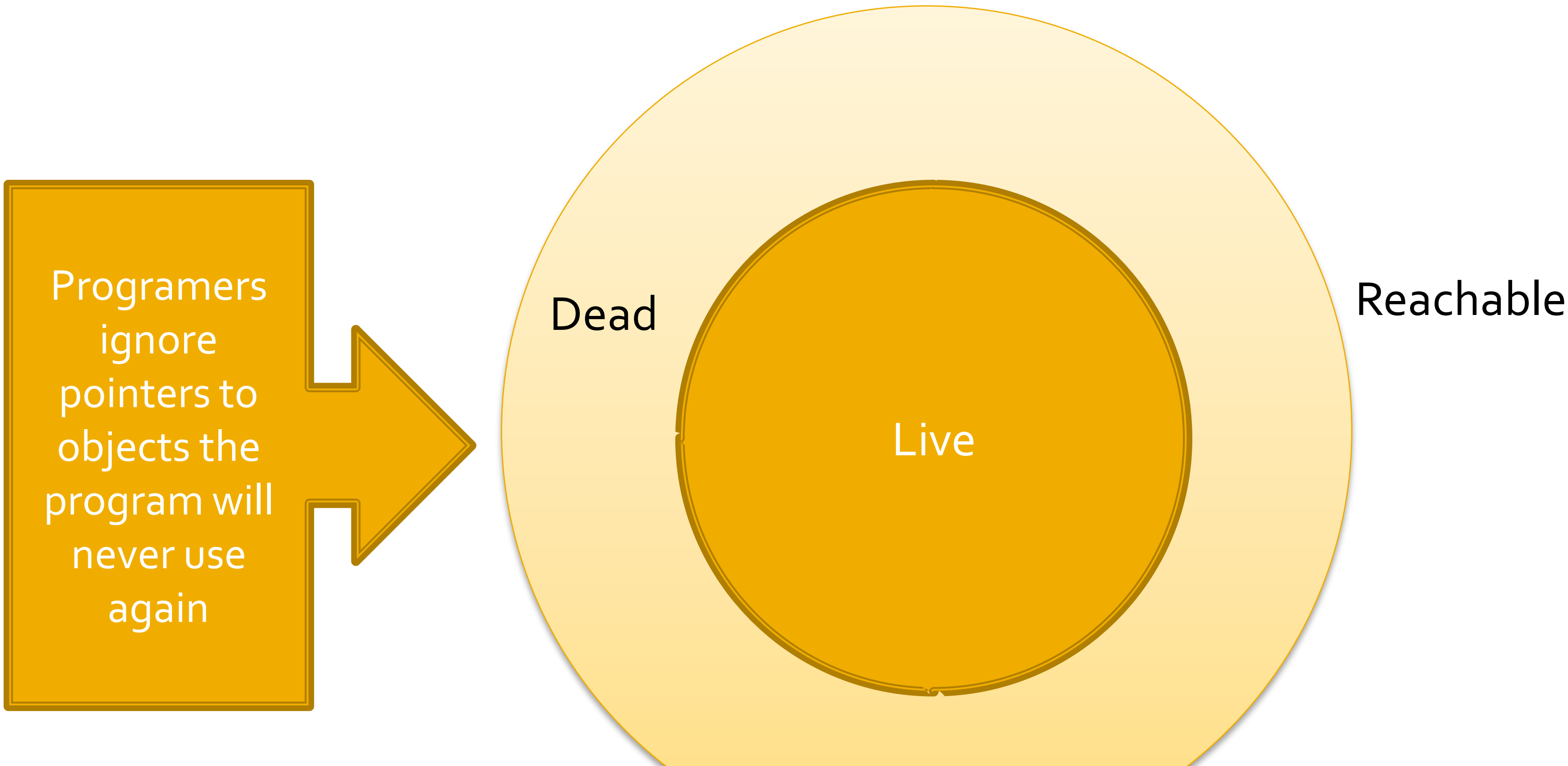
Why?

- Managed languages do not eliminate them
  - Type safety and garbage collection (reliability)



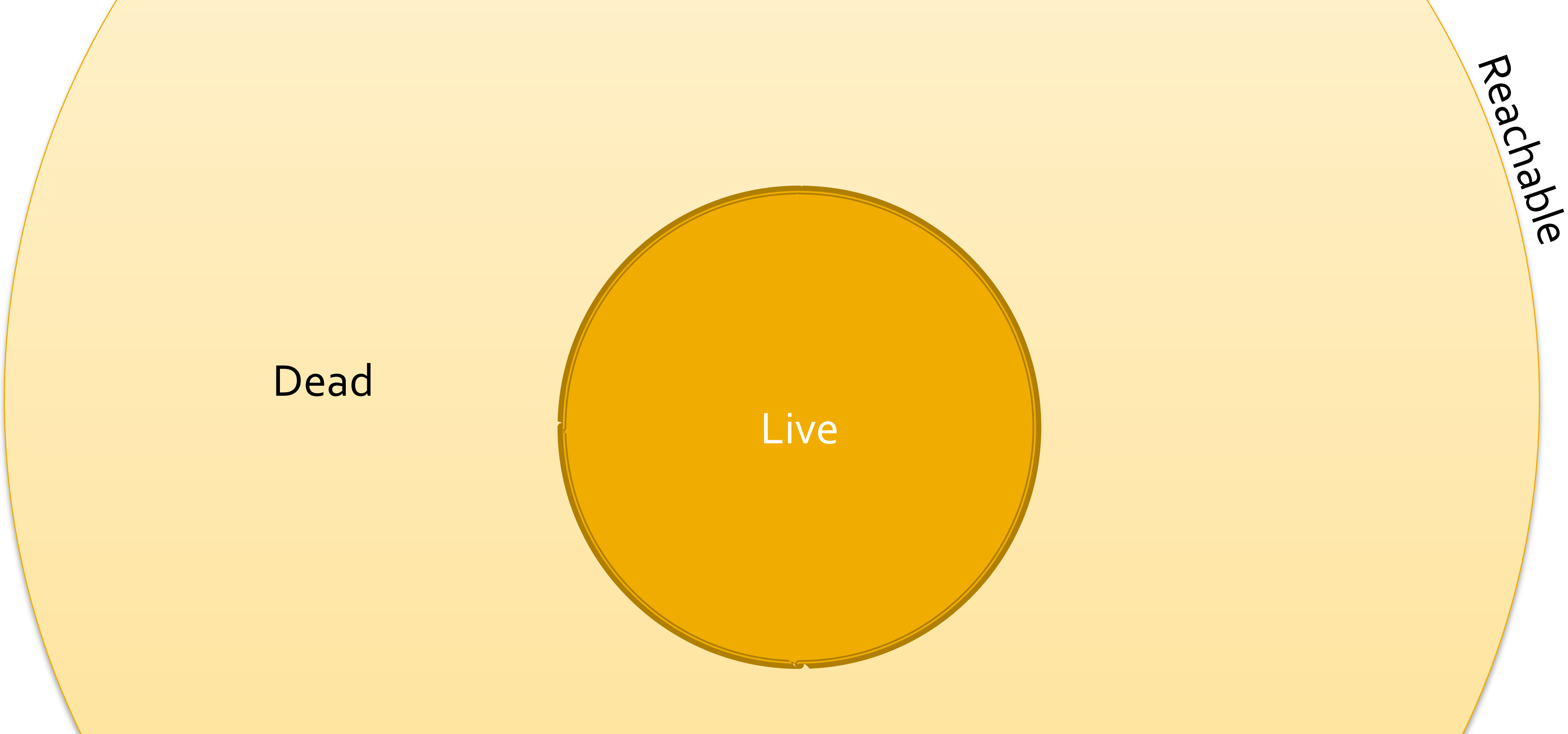
Why?

- Managed languages do not eliminate them
  - Type safety and garbage collection (reliability)



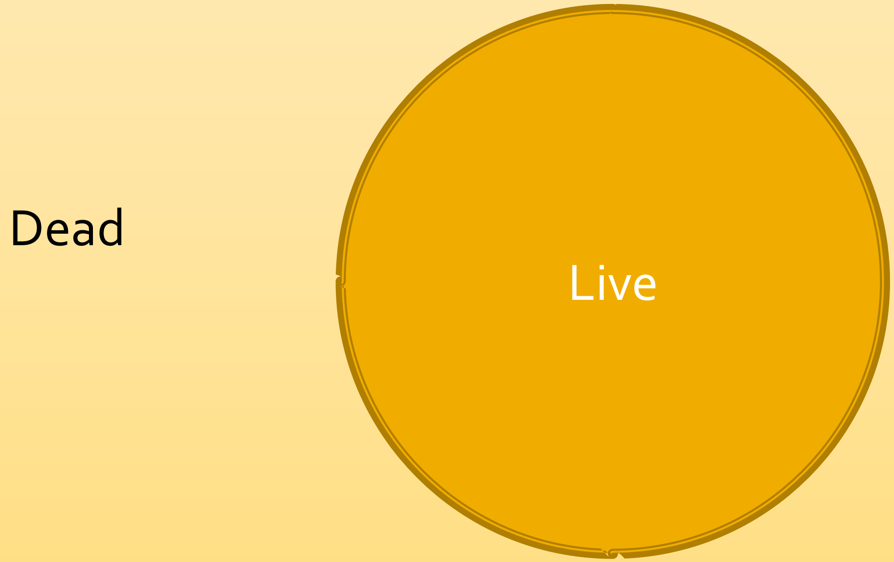
Why?

- Managed languages do not eliminate them
  - Type safety and garbage collection (reliability)



Memory Leaks in Deployed Systems

- No immediate symptoms (reproduce, fix, find)
- Escape developers detection (tools for leaks detection)
- Slow & crash real programs (memory exhausted)



Outline

- Introduction to the problem (why?)
- Leak tolerance: Melt
- Tolerating memory leaks
- Implementation
- Results
- Conclusion
- Related work

Leak tolerance goal

- **USERS: ILLUSION FIX**

Eliminate bad effects

Don't slow

Don't crash

- **DEVELOPERS**

Not a replacement

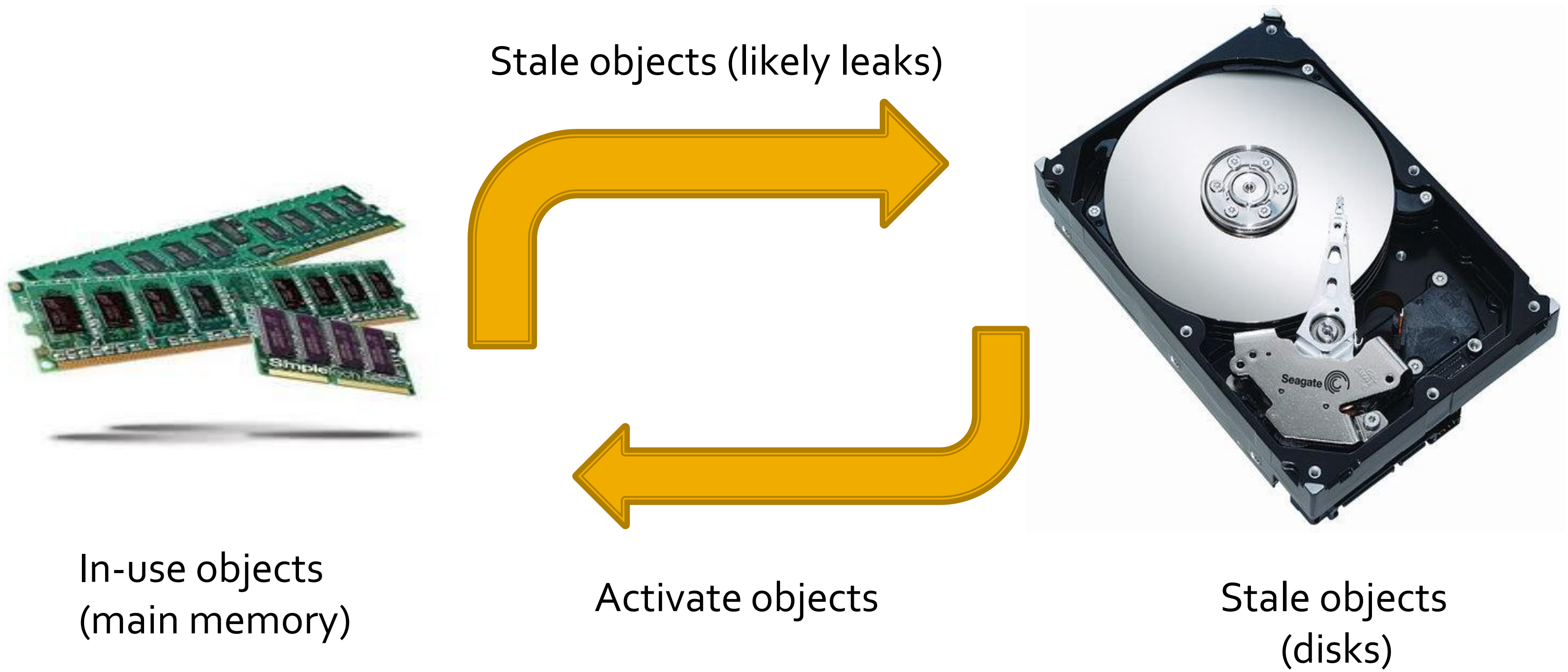
Information

Time

Melt movements

- Predicts that **stale** objects (not used for a while) are likely l**eaks → disk**
- The application try to access an object on disk, it a**ctivates** it → **main memory**

Melt movements



Melt

- **Melt leak tolerance**:
  transfers likely leaked objects to **disk**

Melt

- **Melt leak tolerance**:
  transfers likely leaked objects to **disk**

  - freeing physical and virtual memory
  - much larger than memory → delays exhaustion

Outline

- Introduction to the problem (why?)
- Leak tolerance: Melt
- Tolerating memory leaks
- Implementation
- Results
- Conclusion
- Related work

Tolerating Memory Leaks

- Primary objective: **illusion**
- Invariants:
  - **Stale space**: separate stale objects from in-use ones (disk)
  - Accesses to stale space:
    - ***Collector*** *moves* objects
    - ***Application*** *actives* objects

#1: identify stale objects

- Each collection, objects the program has **not accessed** since the last one, will be **stale** ones

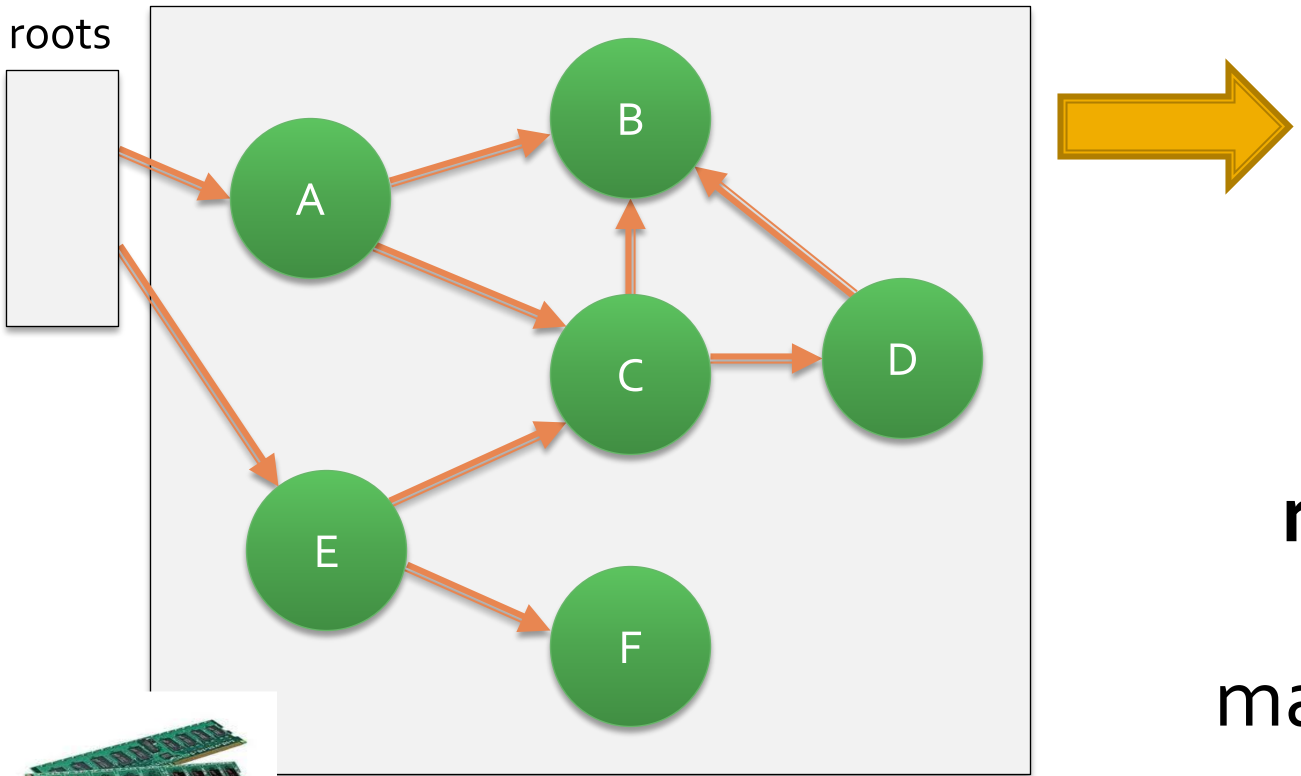
- *Collector* **marks** objects as **stale** every collection

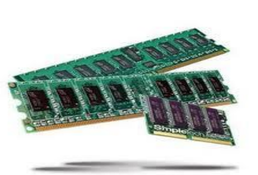
#1: identify stale objects

- Each collection, objects the program has been **accessed**, will be **unmarked**

- *Compiler* and its instrumentation **unmarks** objects every collection

#1: identify stale objects

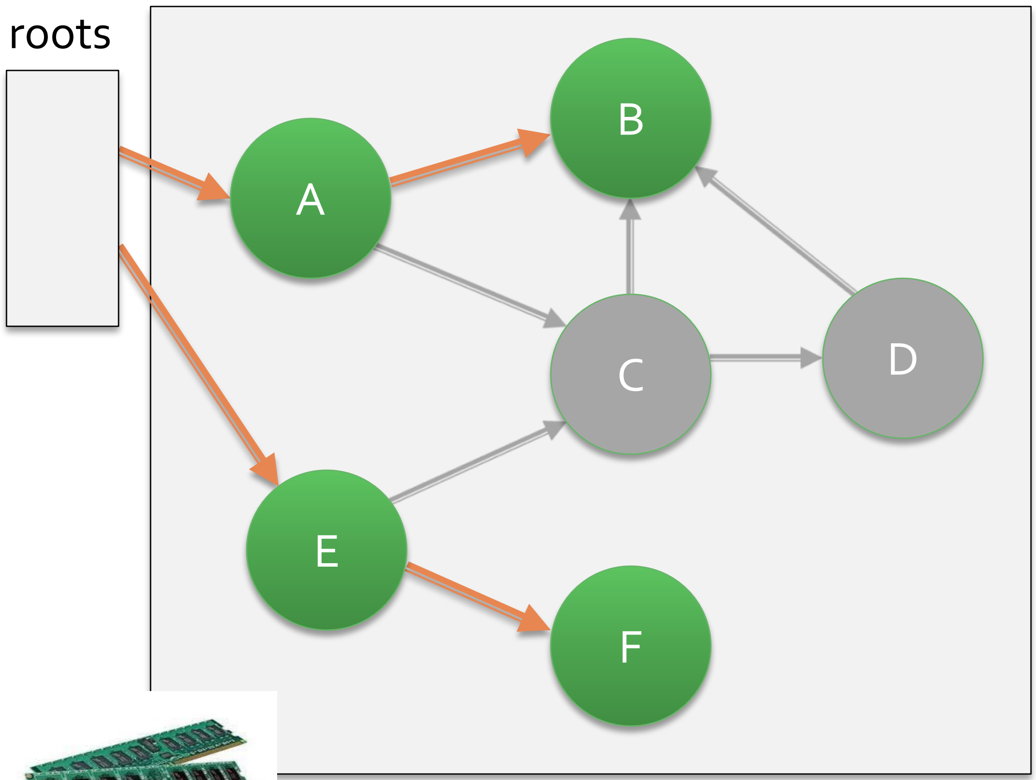


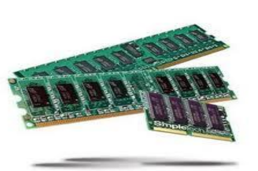


in-use space

Look for **marked references**, instead of marked objects

#1: identify stale objects



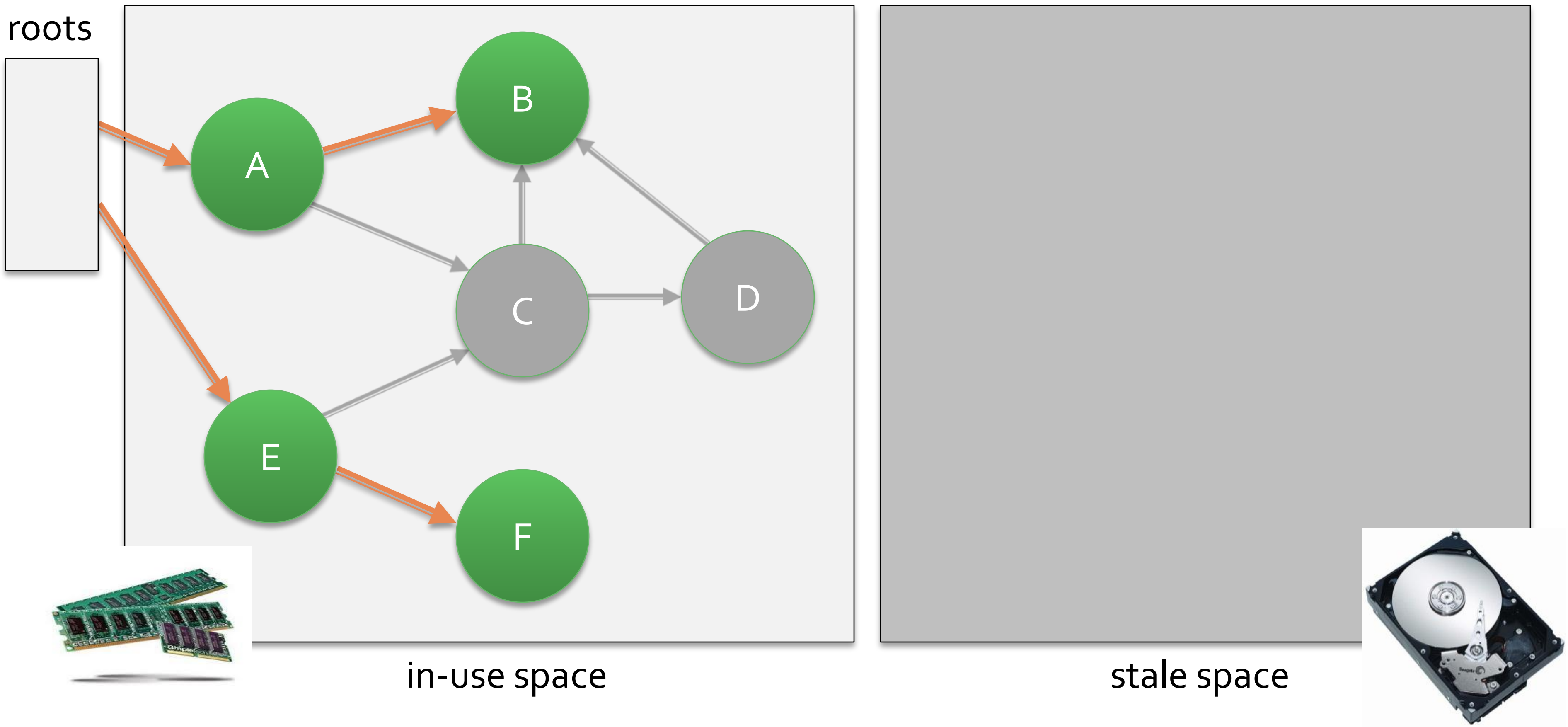


C,D not accessed since the last collection, **all** their incoming references are stale

C,D are **stale** objects

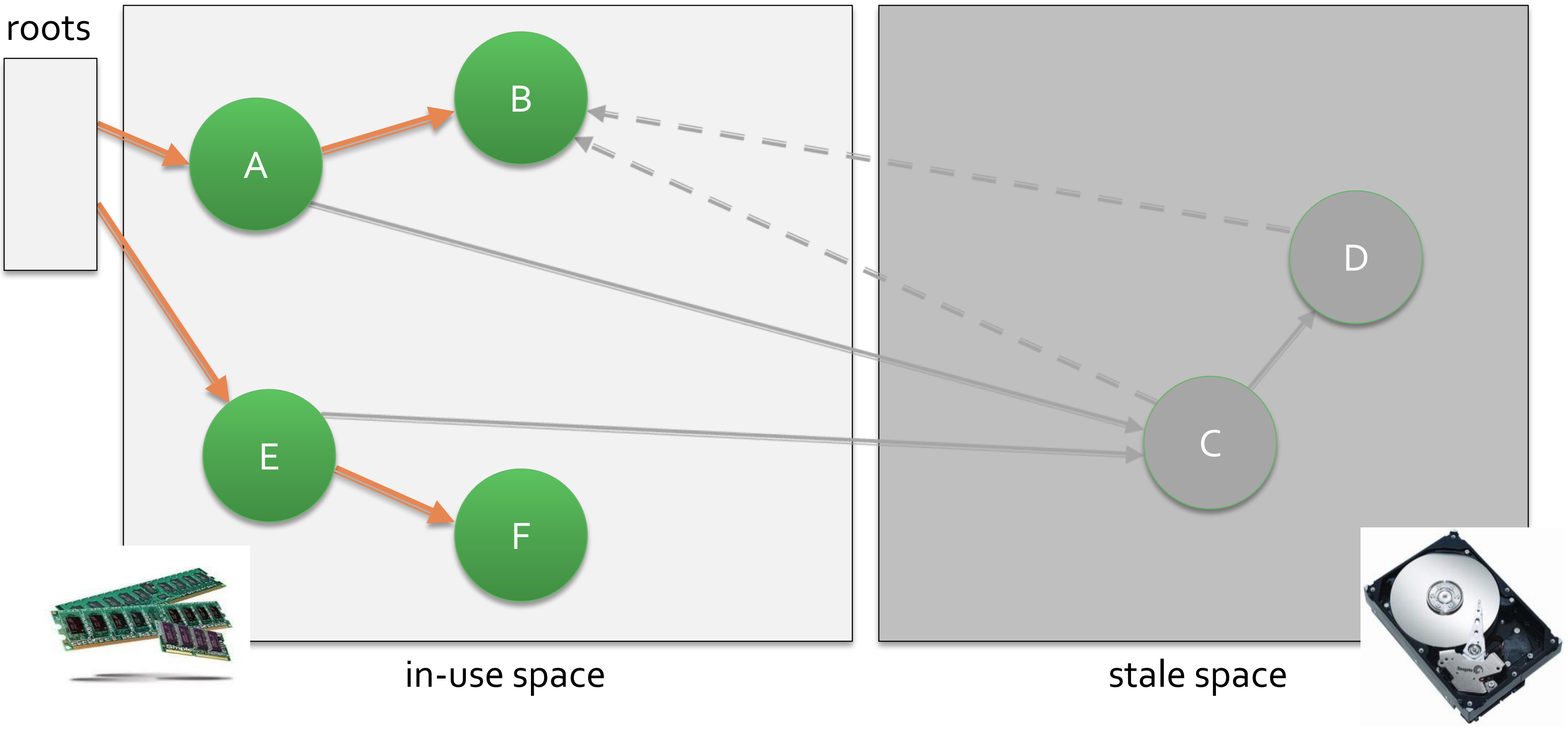
#2: Stale Space

Heap nearly full → move stale objects to disk



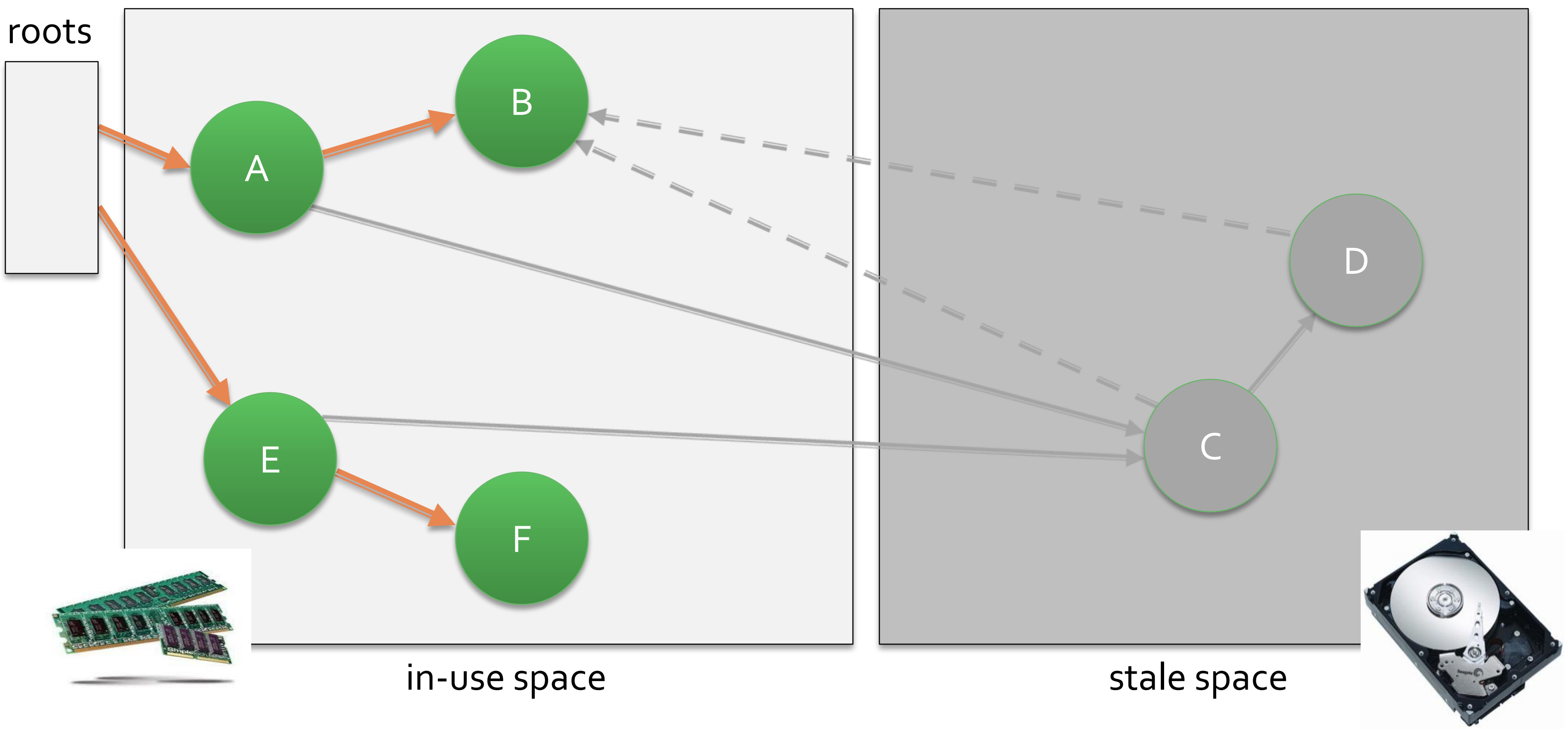
#2: Stale Space

Heap nearly full → move stale objects to disk



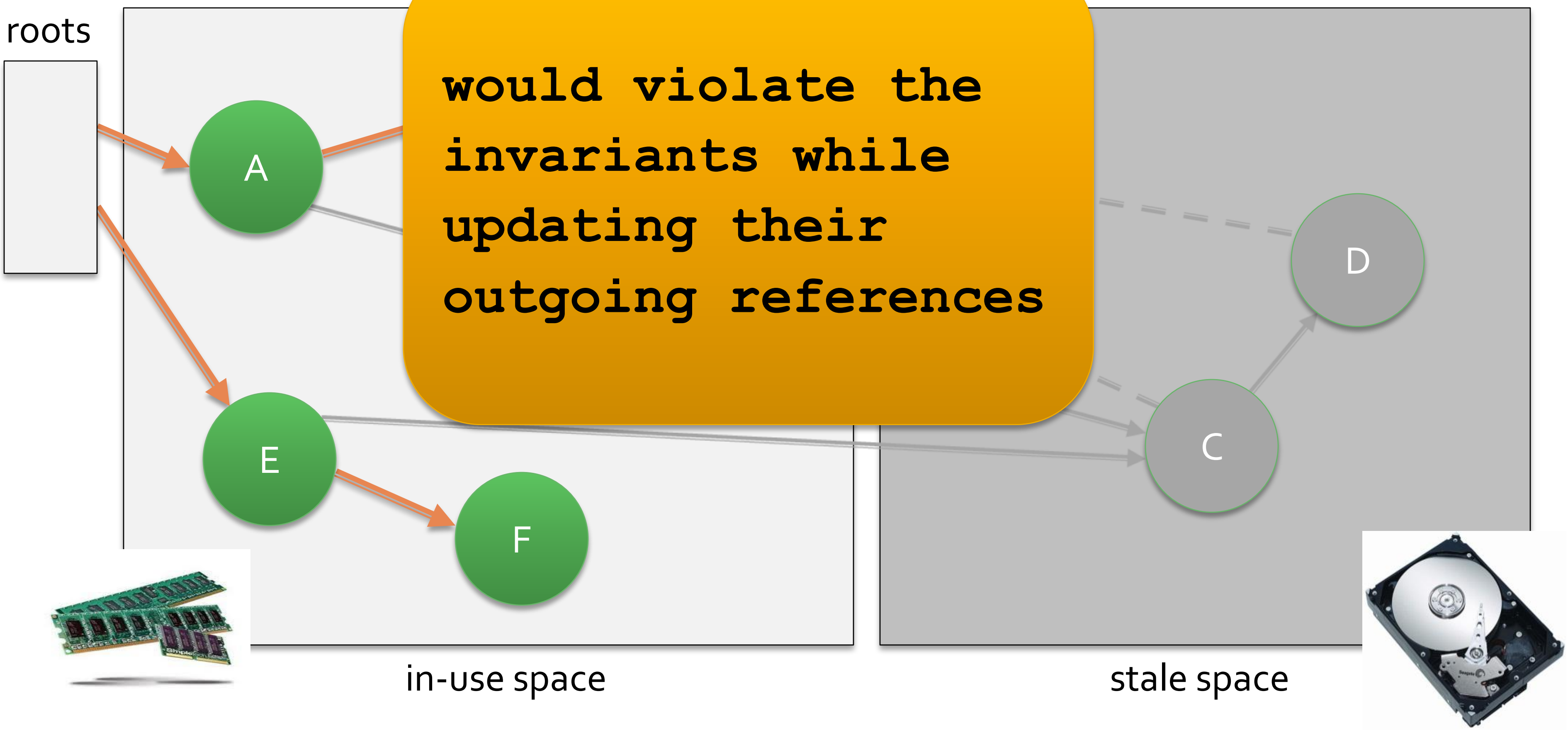
#2: Stale Space

Stale to in-use references → problematic



#2: Stale Space

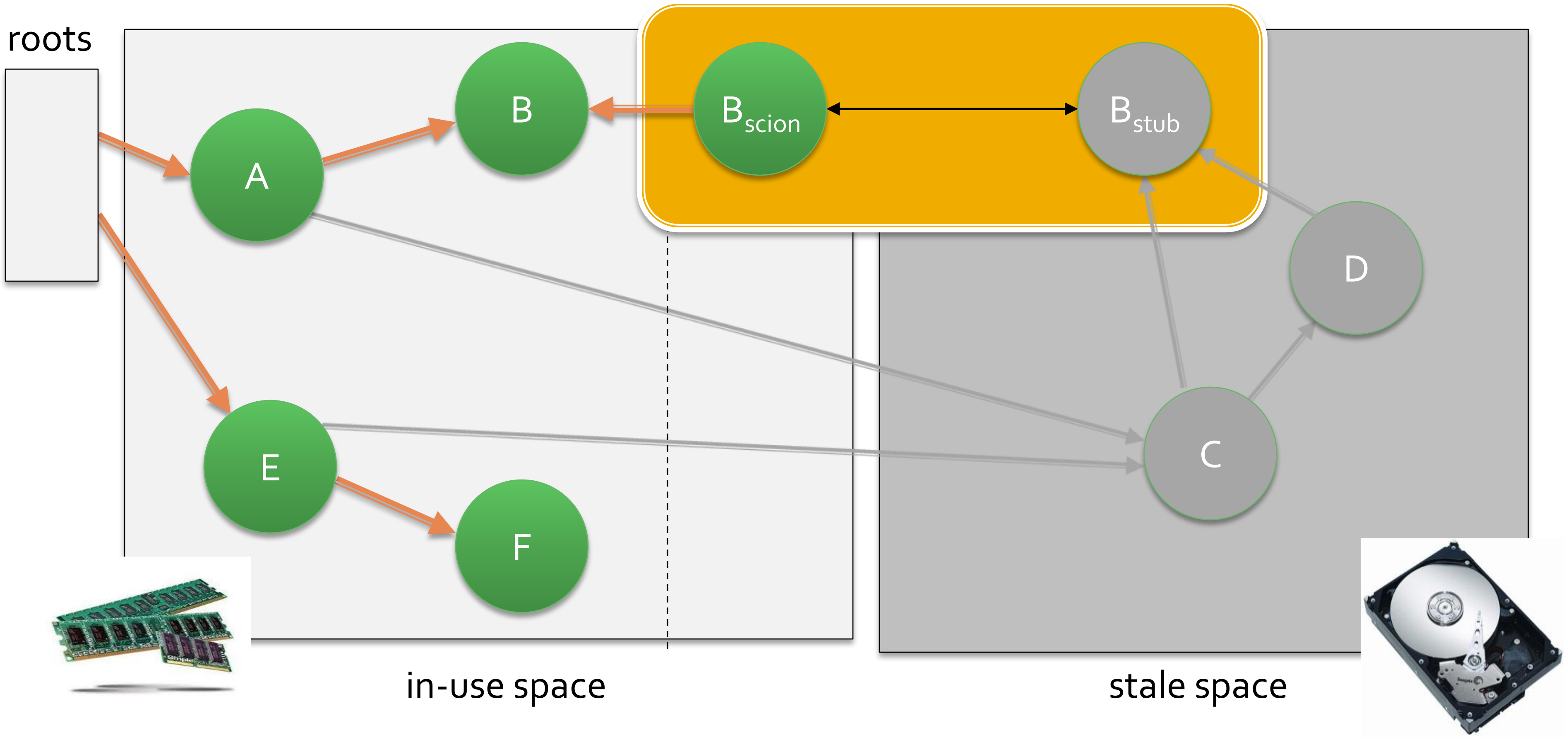
Stale to in-use references → problematic



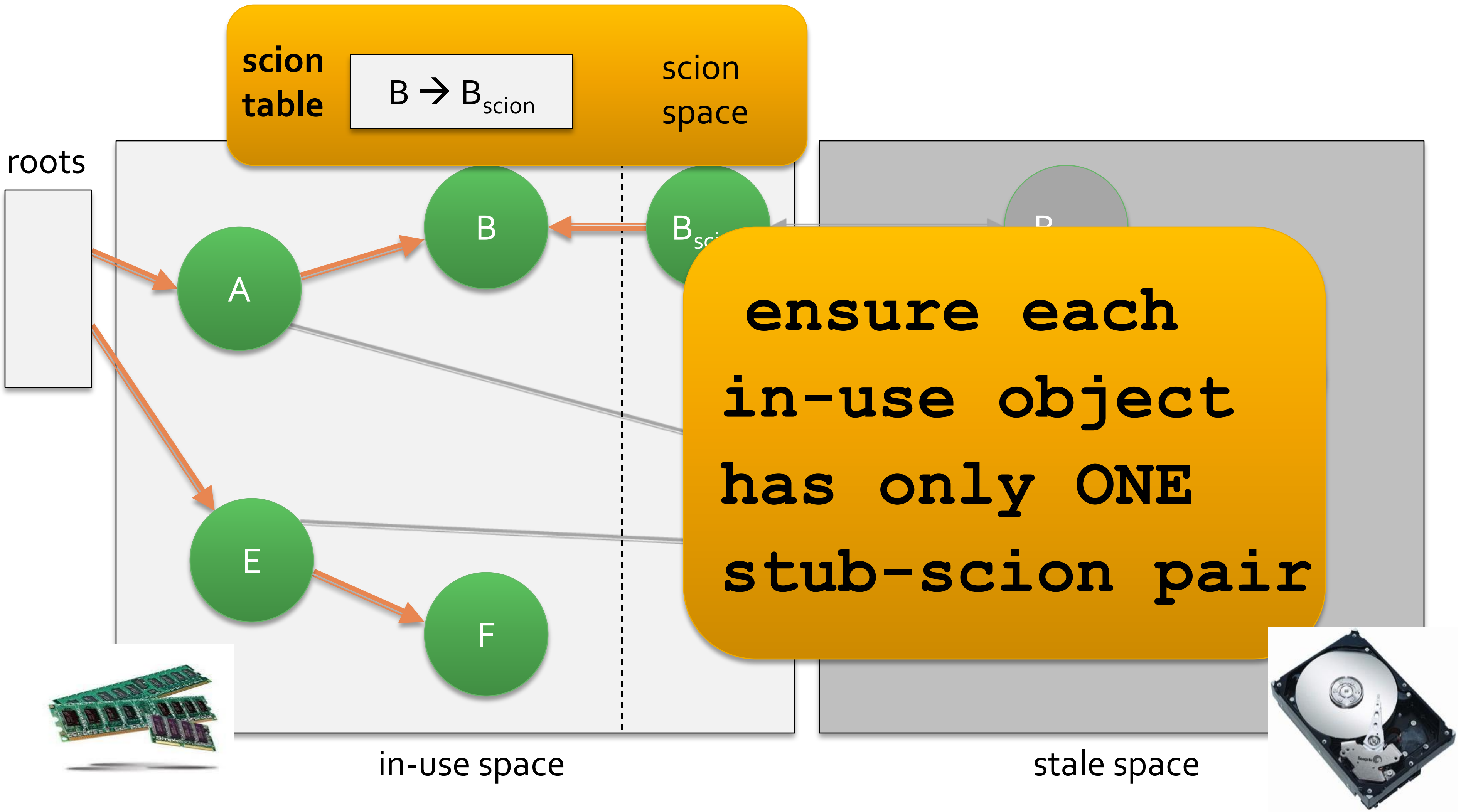
would violate the invariants while updating their outgoing references

#2: Stub-scion pairs

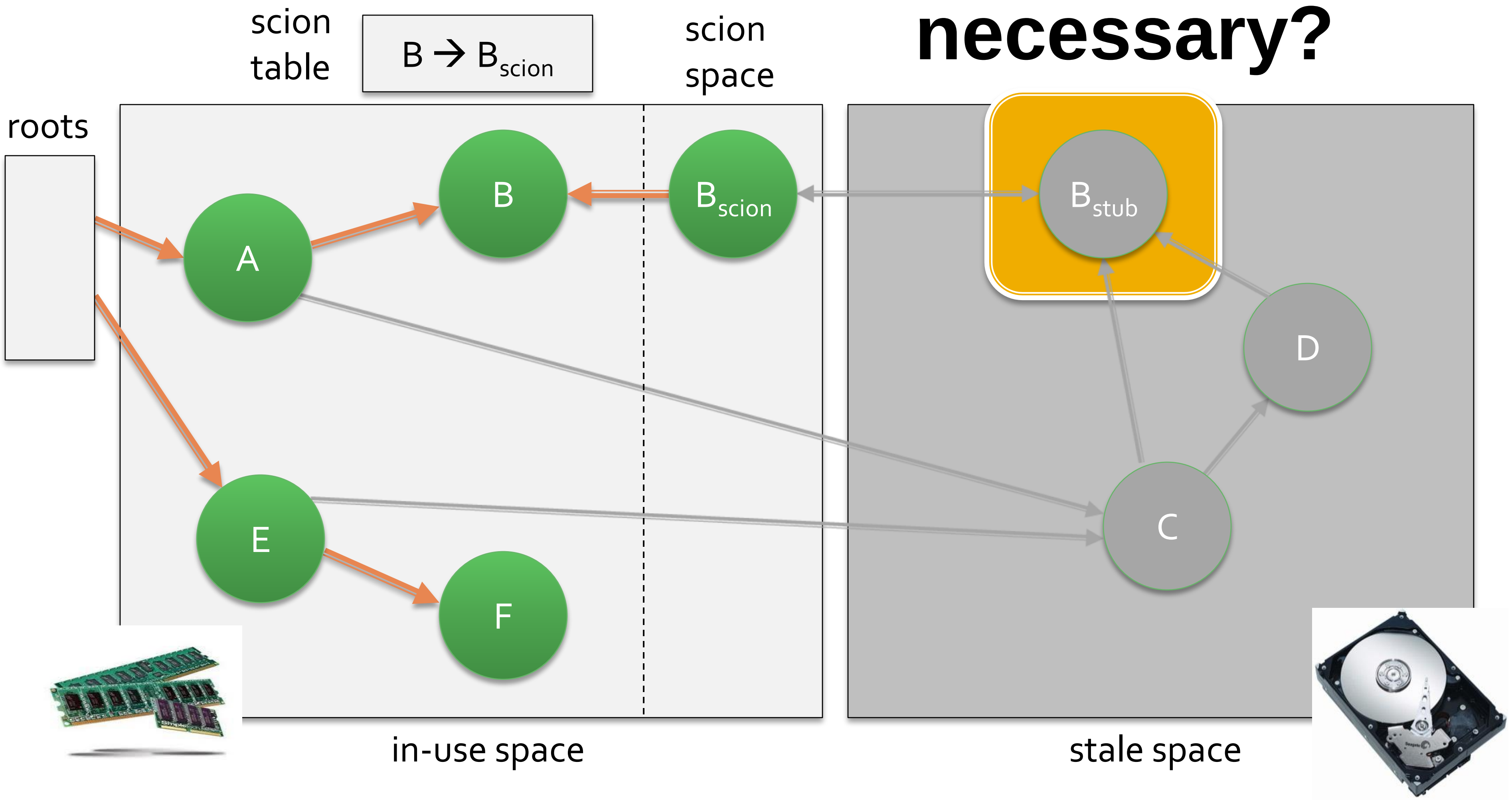
Stub (stale space)-**scion** (in-use space) **pairs** for each in-use object referenced by at least one stale object



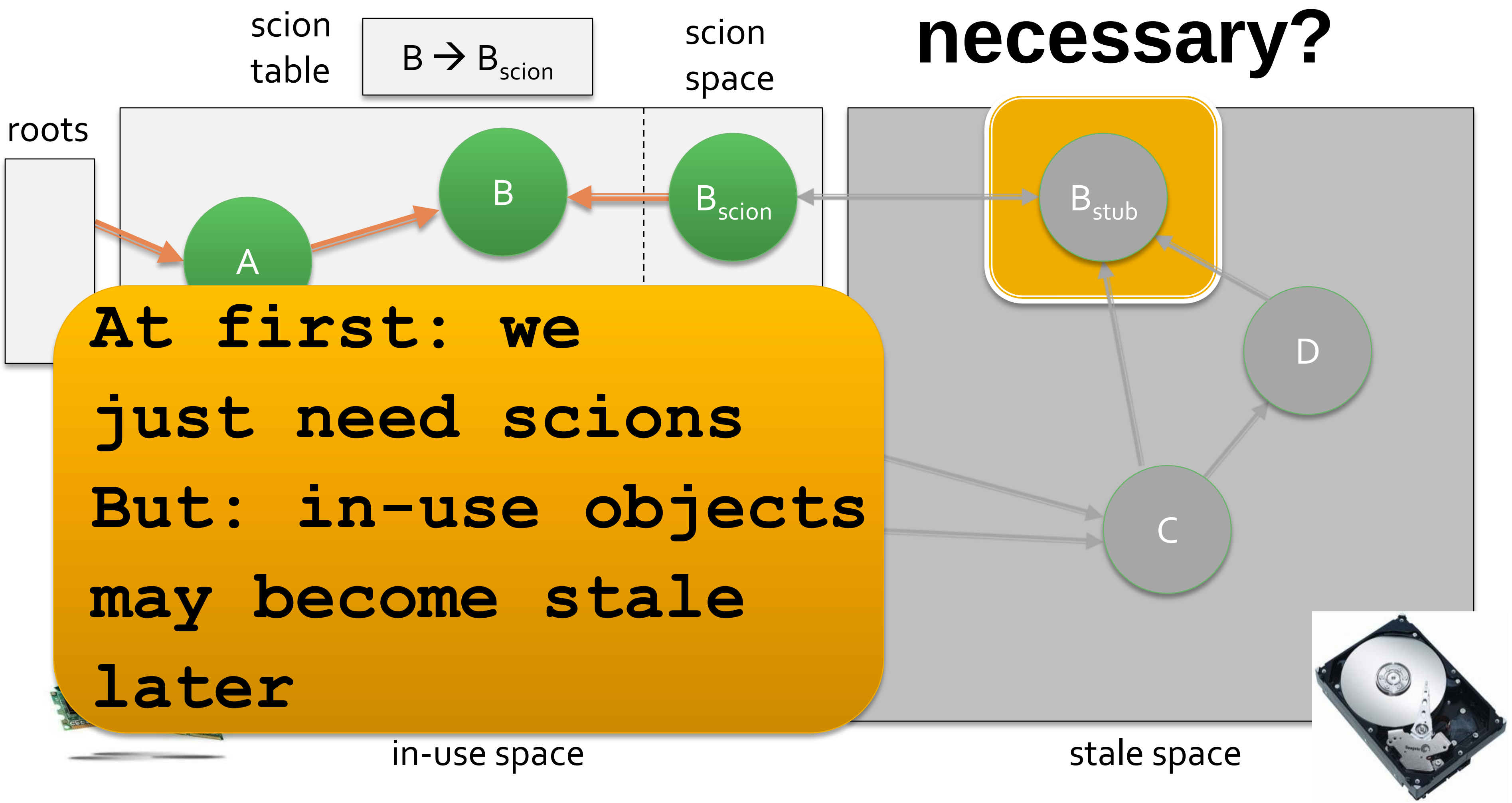
#2: Scion table



#2 :Stub-Scion Pairs



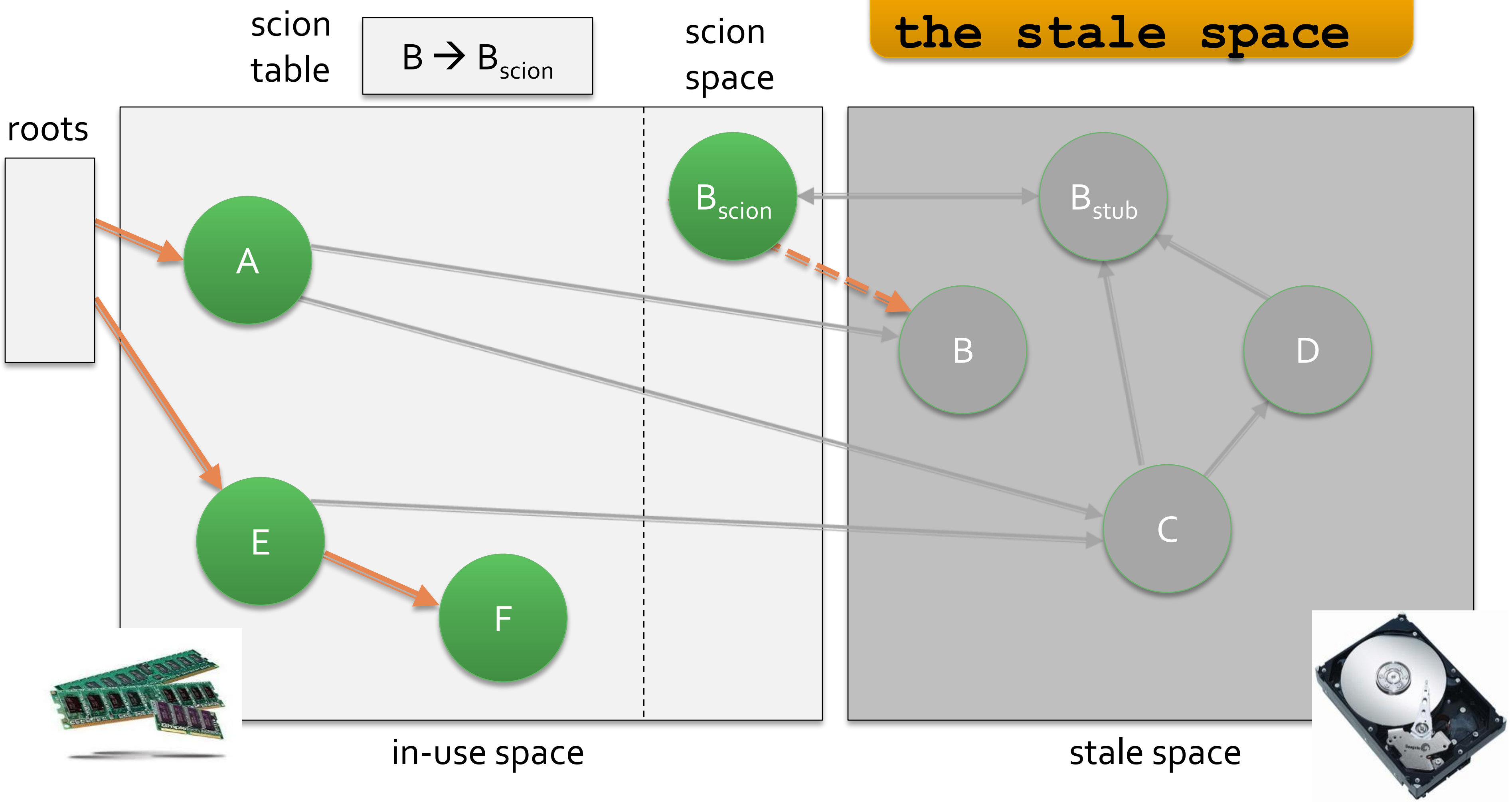
#2 :Stub-Scion Pairs



At first: we just need scions
But: in-use objects may become stale later

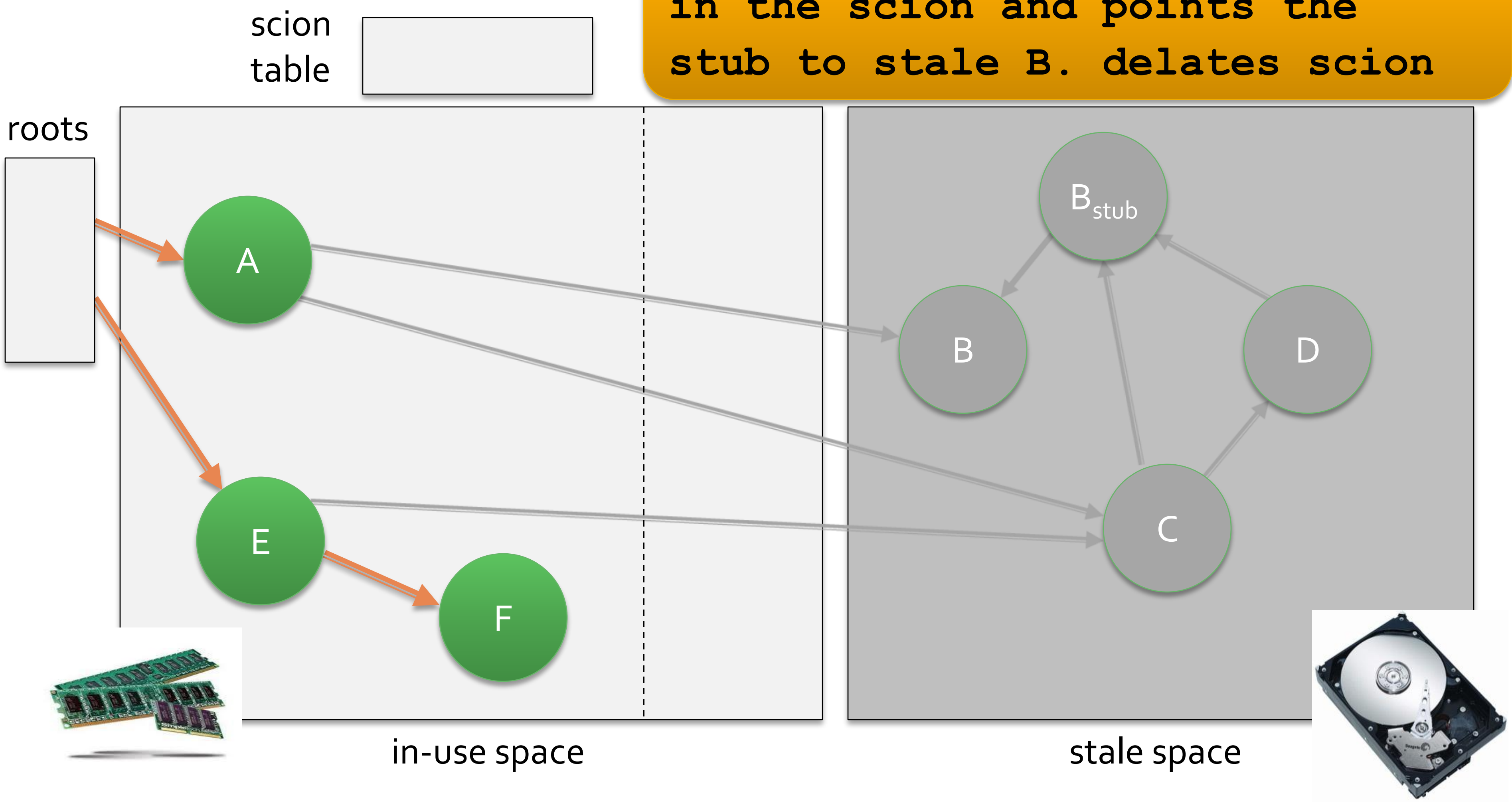
#2 : Scion-Referenced Object Becomes Stale

copies B into the stale space

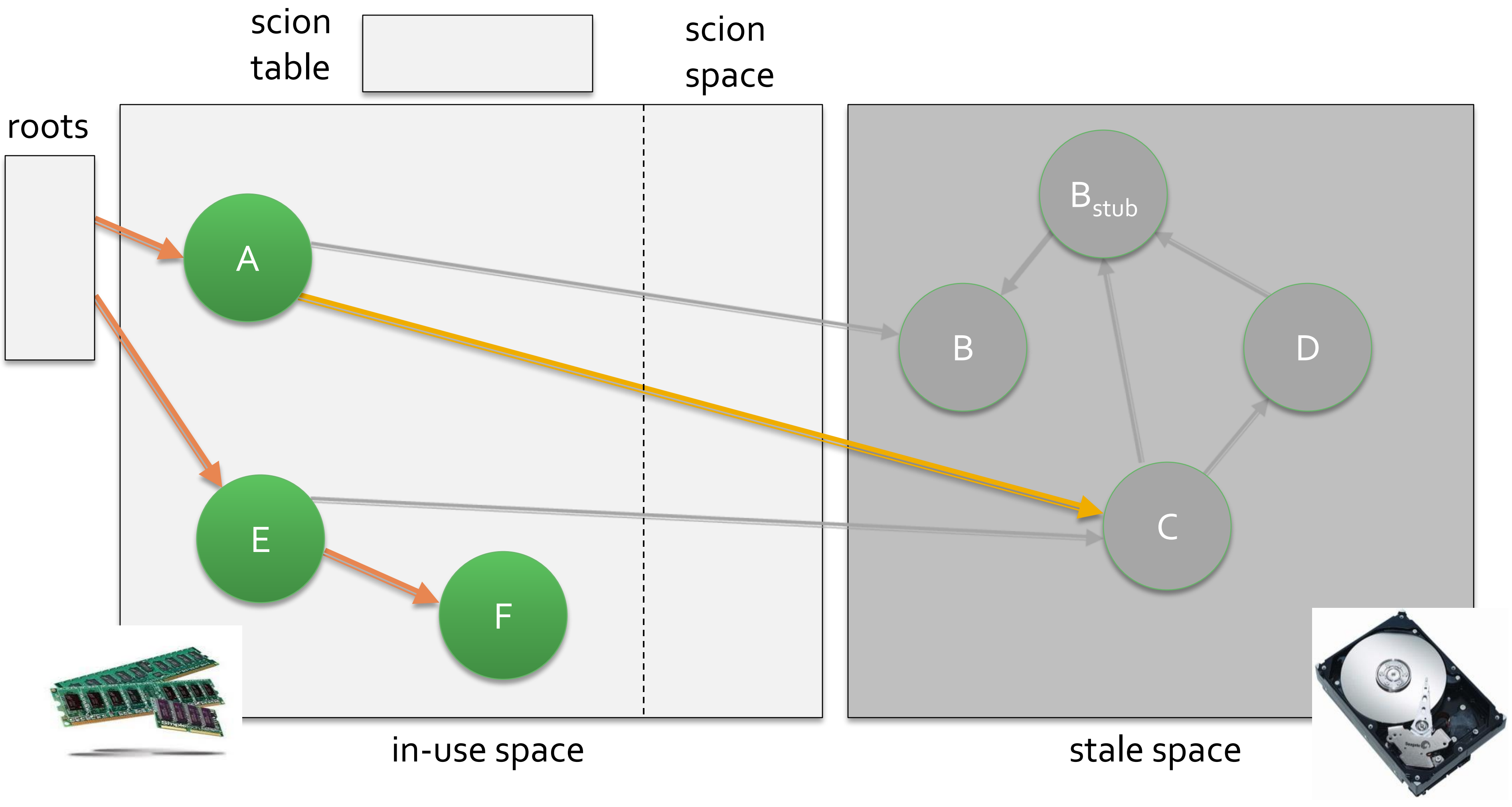


#2 : Scion-Referenced Object Becomes Stale

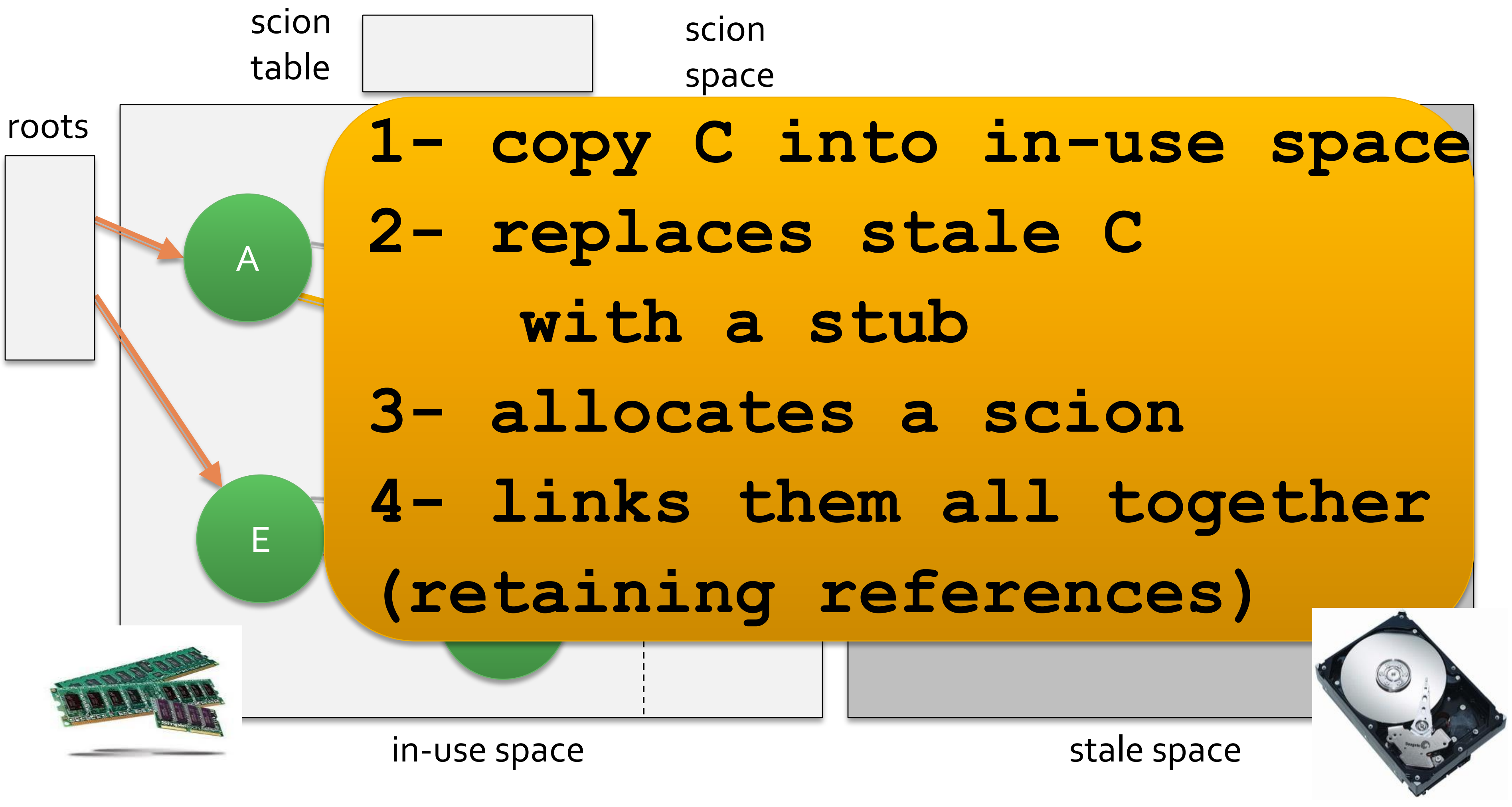
Looks up the stub location in the scion and points the stub to stale B. delates scion



#3: Activating Stale objects

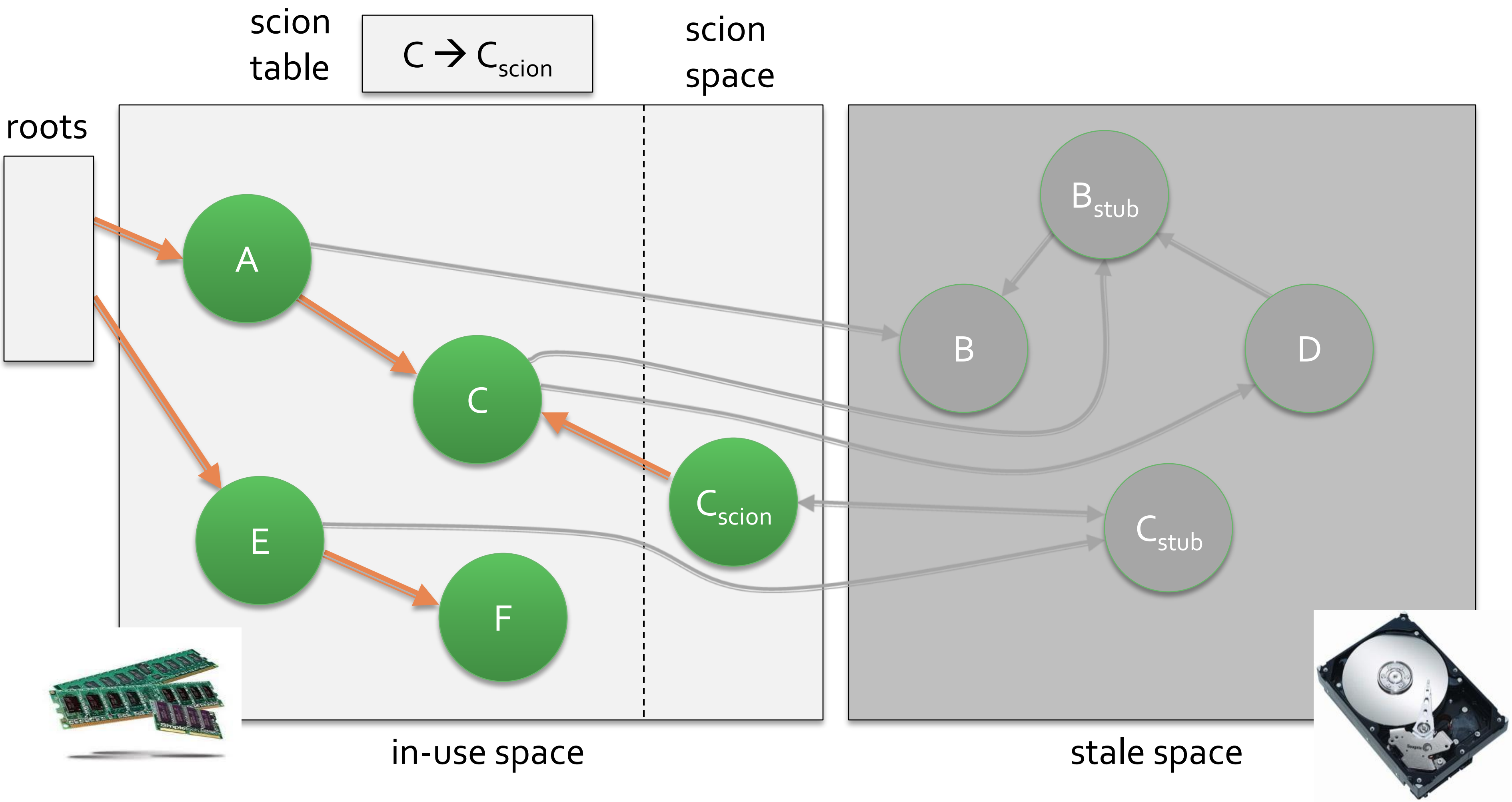


#3: Activating Stale objects

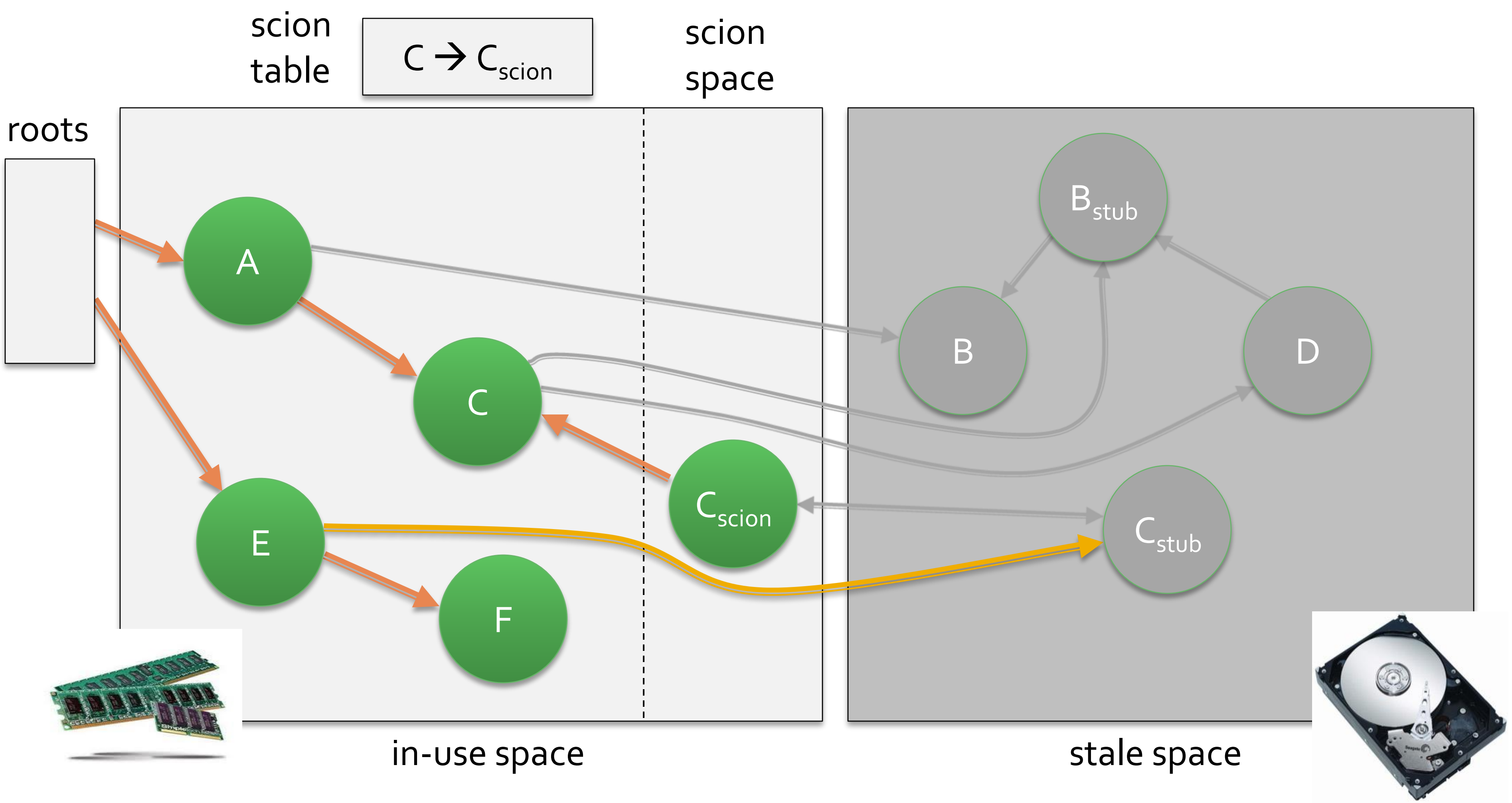


1- copy C into in-use space
2- replaces stale C with a stub
3- allocates a scion
4- links them all together (retaining references)

#3: Application Accesses Stale Object

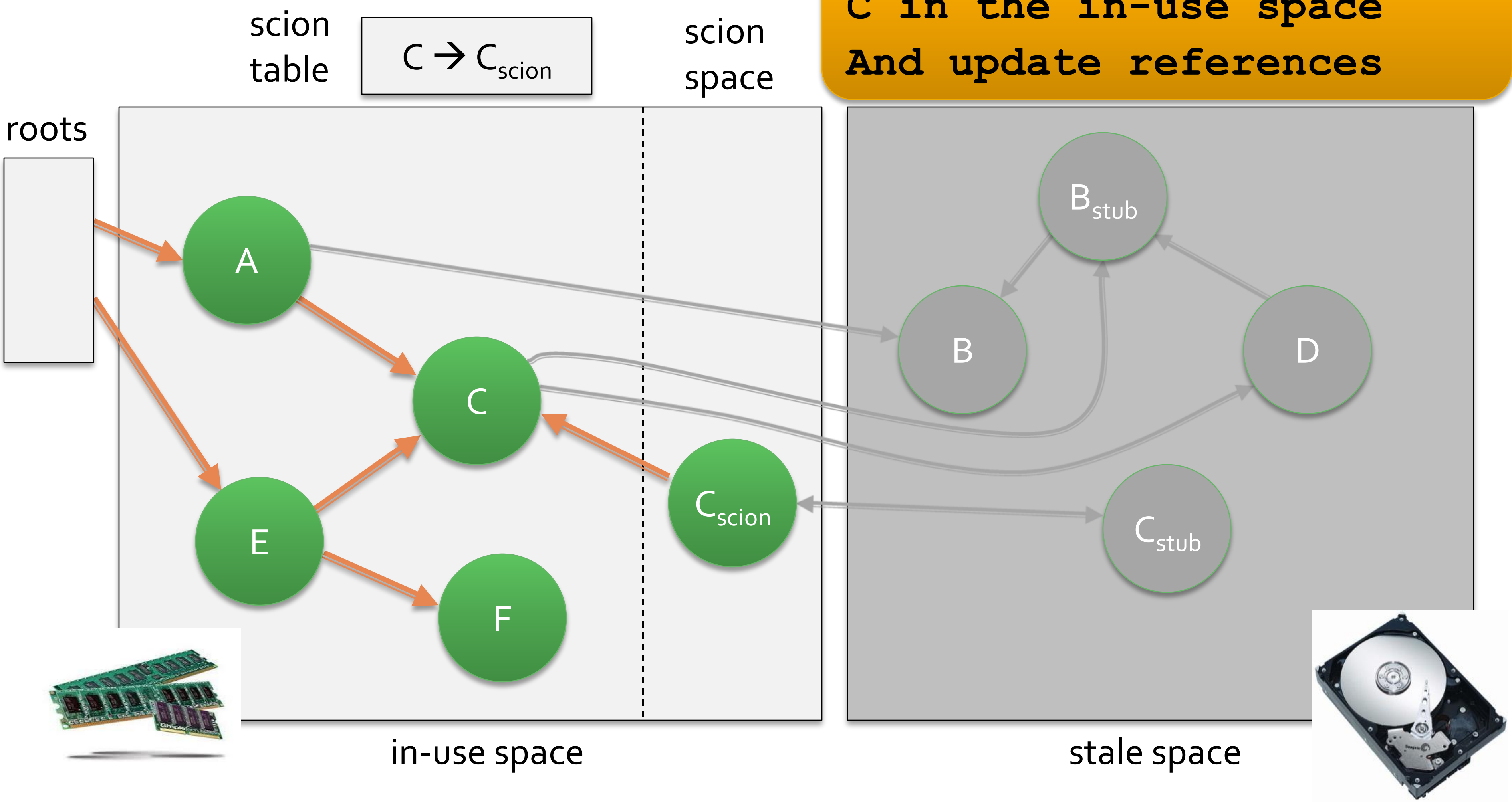


#3: Application Accesses Stale Object

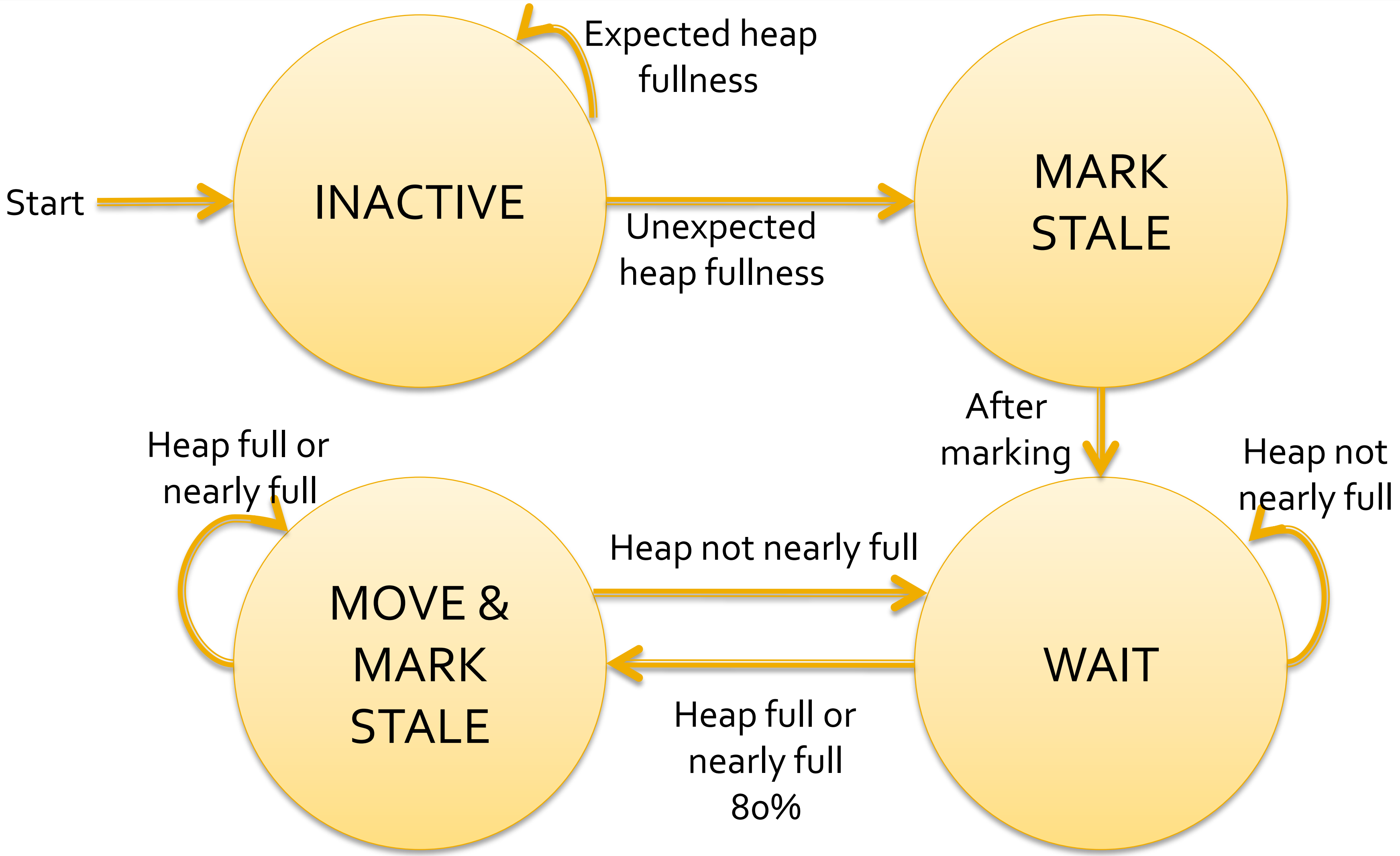


#3: Application Accesses Stale Object

Follows Cstub to Cscion to C in the in-use space And update references



State Melt diagram



Outline

- Introduction to the problem (why?)
- Leak tolerance: Melt
- Tolerating memory leaks
- Implementation
- Results
- Conclusion
- Related work

Implementation: VM issues

- Implemented in Jikes RVM 2.9.2
- Melt design compatible with any tracing collector
  - i.e., for the demonstration is used generational copying collector

Outline

- Introduction to the problem (why?)
- Leak tolerance: Melt
- Tolerating memory leaks
- Implementation
- Results
- Conclusion
- Related work

Performance Evaluation

- **Benchmarks**
  (to measure overhead):
  - DaCapo, SPECjbb2000, SPECjvm98
- **Platform**
  (where execution was experimented):
  - Dual-core Pentium 4
- **Results**
  - 6% overhead on average

Tolerating Leaks

- *Evaluation*: How well tolerates growing leaks by running them longer and maintaining program performance?
- **10 leaks** founded
  - 5 tolerated
  - 2 tolerated but with high overhead (activating many stale objects)
  - 3 doesn’t significally help

Tolerating Leaks

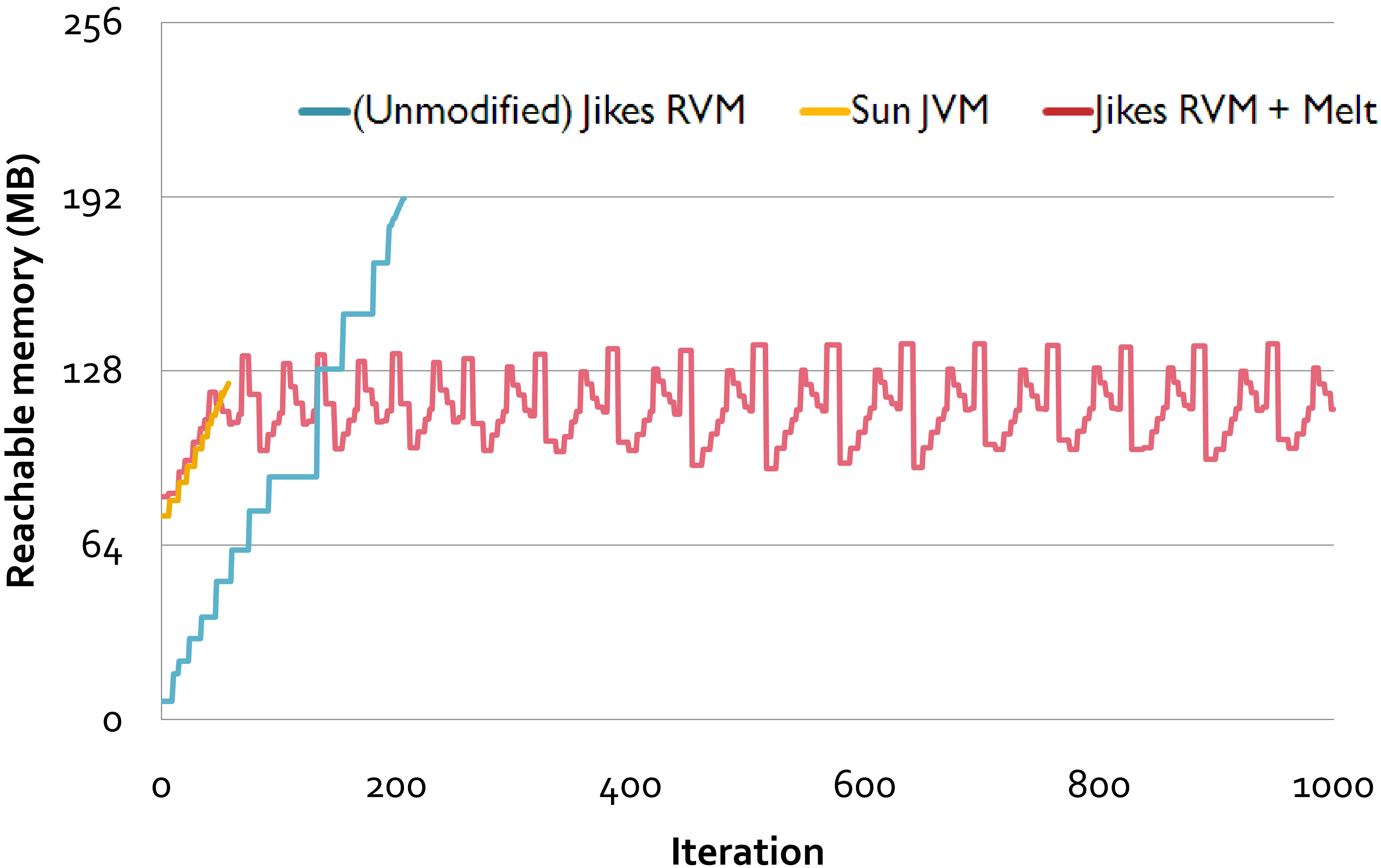
Leak	Melt's effect
Eclipse "Diff"	**Tolerates until 24-hr limit (1,000X longer)**
Eclipse "Copy-Paste"	Tolerates until 24-hr limit (194X longer)
JbbMod	Tolerates until 20-hr crash (19X longer)
ListLeak	Tolerates until disk full (200X longer)
SwapLeak	Tolerates until disk full (1,000X longer)
MySQL	Some 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 [illegible]il 24-hr limit (194X longer)
JbbMod	Tolera[illegible]-hr crash (19X longer)
ListLeak	[illegible] longer)
SwapLeak	[illegible]X longer)
MySQL	[illegible] (74X longer)
Delaunay Mesh	[illegible]
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)

Leaky program: has live leaks for improving longevity and performance significantly

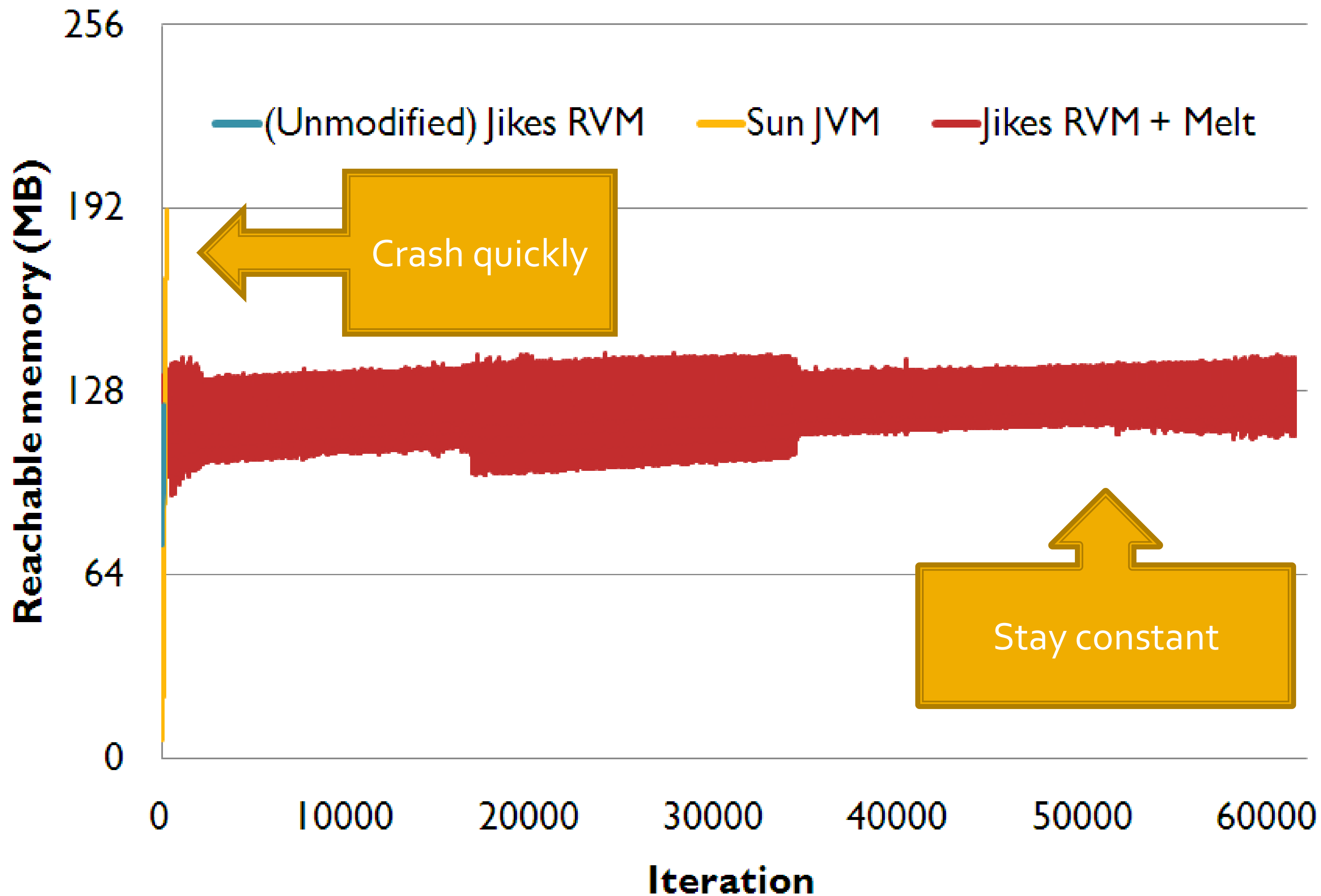
Eclipse Diff: Reachable Memory



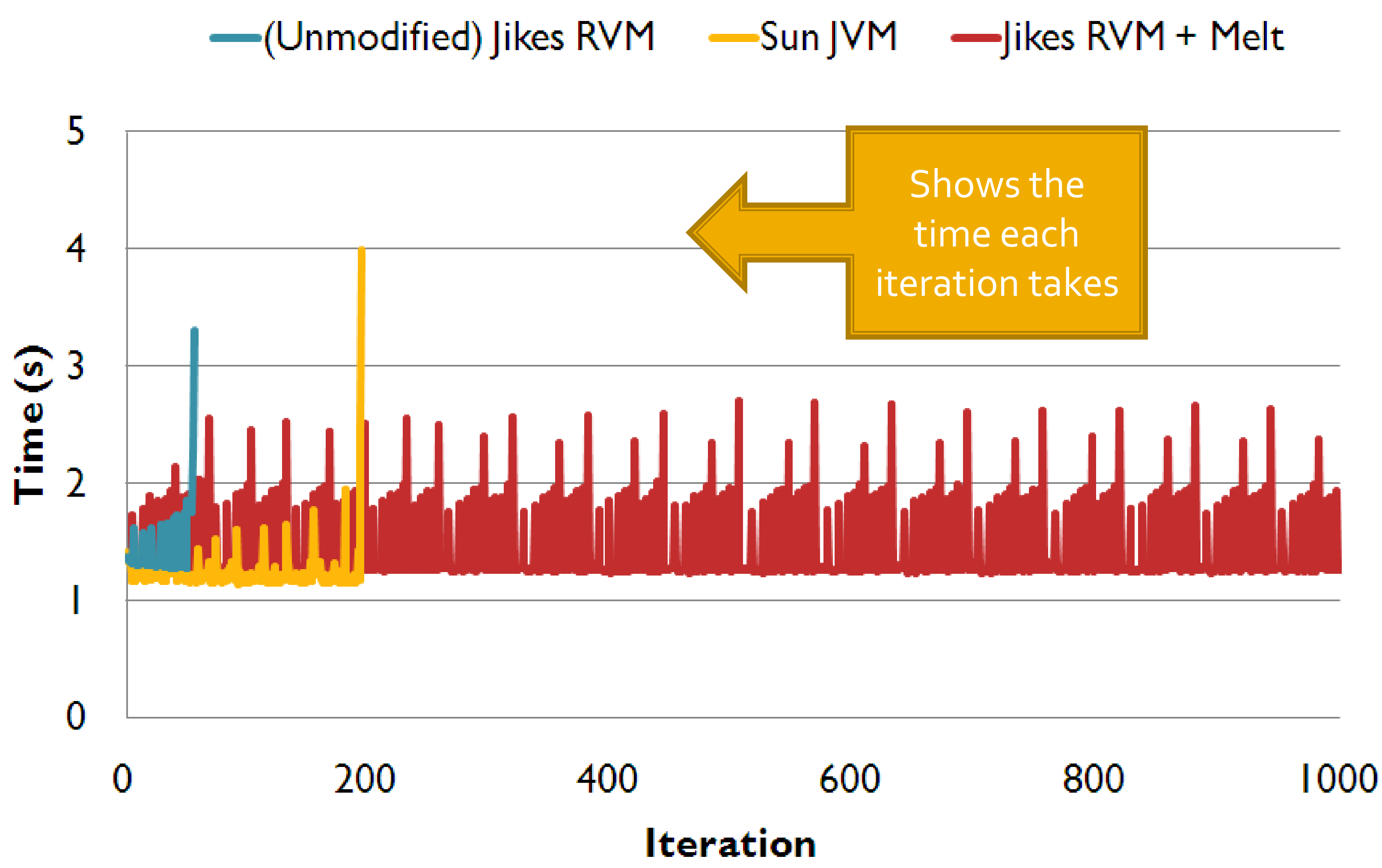
Eclipse Diff: Reachable Memory

- Conclusions comparing reachable memory for first 1000 iterations:
  - Jikes RVM and Sun JVM **fill the heap** as the leak grows
  - Melt starts moving stale objects (80% full) and keeps memory usage fairly **constant** between 100 and 130 MB

Eclipse Diff: Reachable Memory

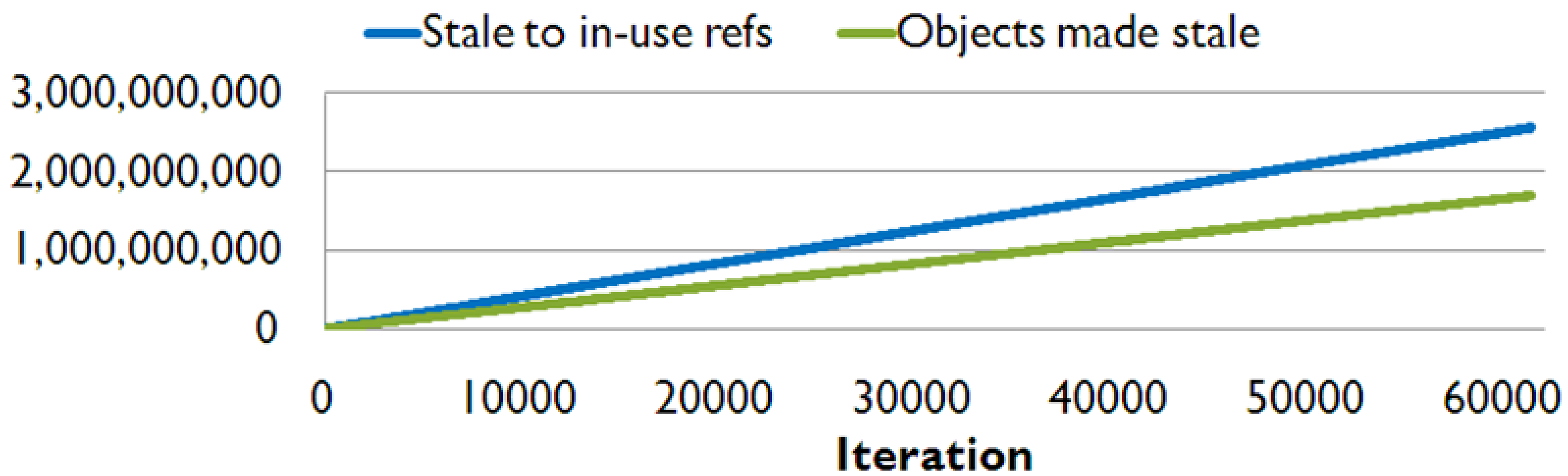


Eclipse Diff: Performance



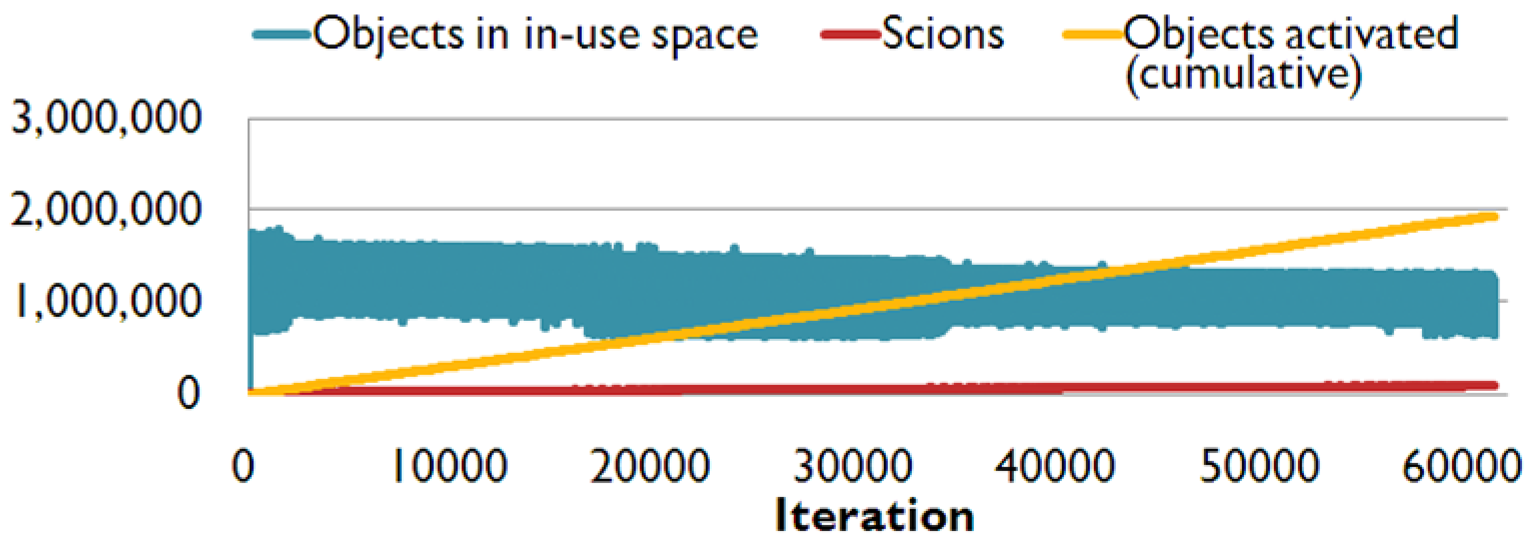
Eclipse Diff leak with Melt

- Grow linearly over iterations and have large magnitudes



Eclipse Diff leak with Melt

- In-use objects constant over iterations
- Scions grows linearly over time (small)
- Objects activated increase linearly



Outline

- Introduction to the problem (why?)
- Leak tolerance: Melt
- Tolerating memory leaks
- Implementation
- Results
- Conclusion
- Related work

Conclusion

- Finding bugs before deployment is hard
- Melt:

Conclusion

- Finding bugs before deployment is hard
- Melt:
  - Requires time & space proportional to in-use memory
  - Preserves safety (activating stale objects on disk)

Conclusion

- Finding bugs before deployment is hard
- Melt:
  - Requires time & space proportional to in-use memory
  - Preserves safety (activating stale objects on disk)
- Developers → time
- Users → illusion

Outline

- Introduction to the problem (why?)
- Leak tolerance: Melt
- Tolerating memory leaks
- Implementation
- Results
- Conclusion
- Related work

Related work

- Publishers follow-on work called **Leak Pruning** that *reclaims* (i.e., deletes) memory that seems to be leaked, instead of moving it to disk:

http://www.cse.ohio-state.edu/~mikebond/papers.html#leak-pruning

March 2009

THANK YOU

Maria.MartinCiviac@sbg.ac.at