MC^2: High Performance GC for Memory-Constrained Environments

Narendran Sachindran

J. Eliot B. Moss

Emery D. Berger

University of Massachusetts Amherst

Motivation

- Java widely used
  - Safety
  - Portability
- Garbage collector requirements
  - High throughput
  - Short pauses
  - Good memory utilization

Motivation

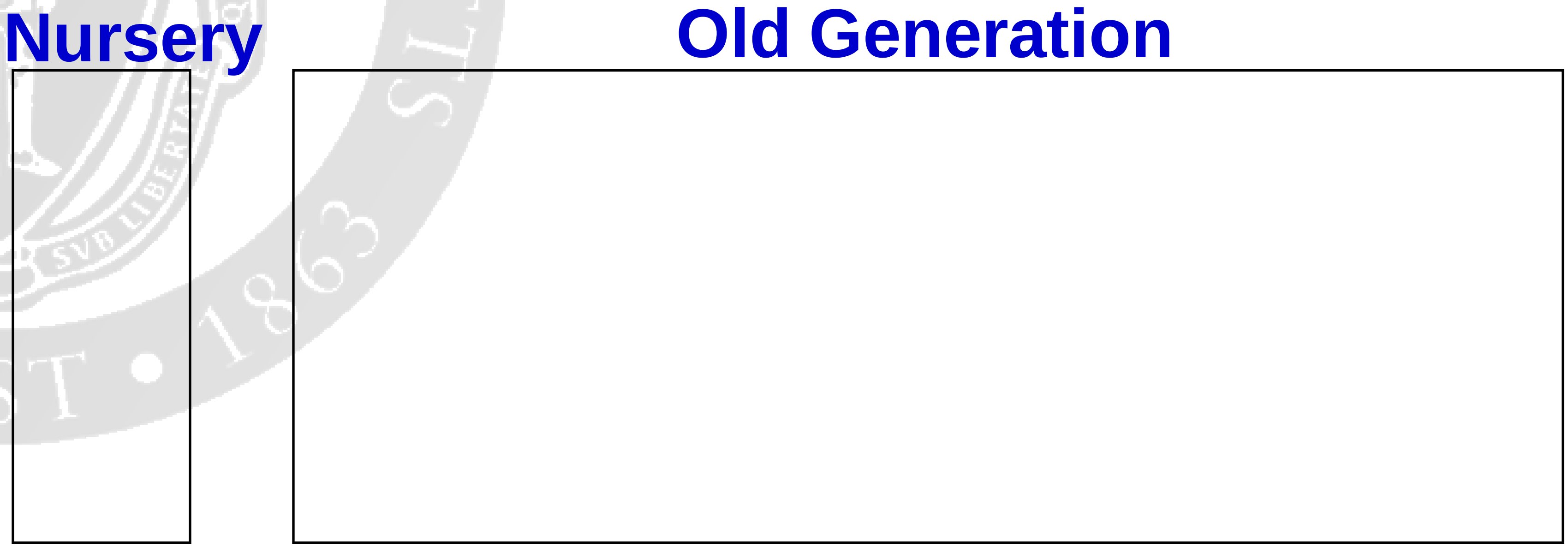
- Handheld Devices
  - Cellular phones, PDAs widely used
  - Constrained memory
- Diverse applications
  - Media players
  - Video games
  - Digital cameras
  - GPS
  - Scaled down desktop apps (e-mail, browser etc.)
- Require high throughput, short response time

Talk Outline

- Generational collection
- MC^2 overview
- Algorithmic Details
- Experimental Results
- Conclusions

Generational Collection

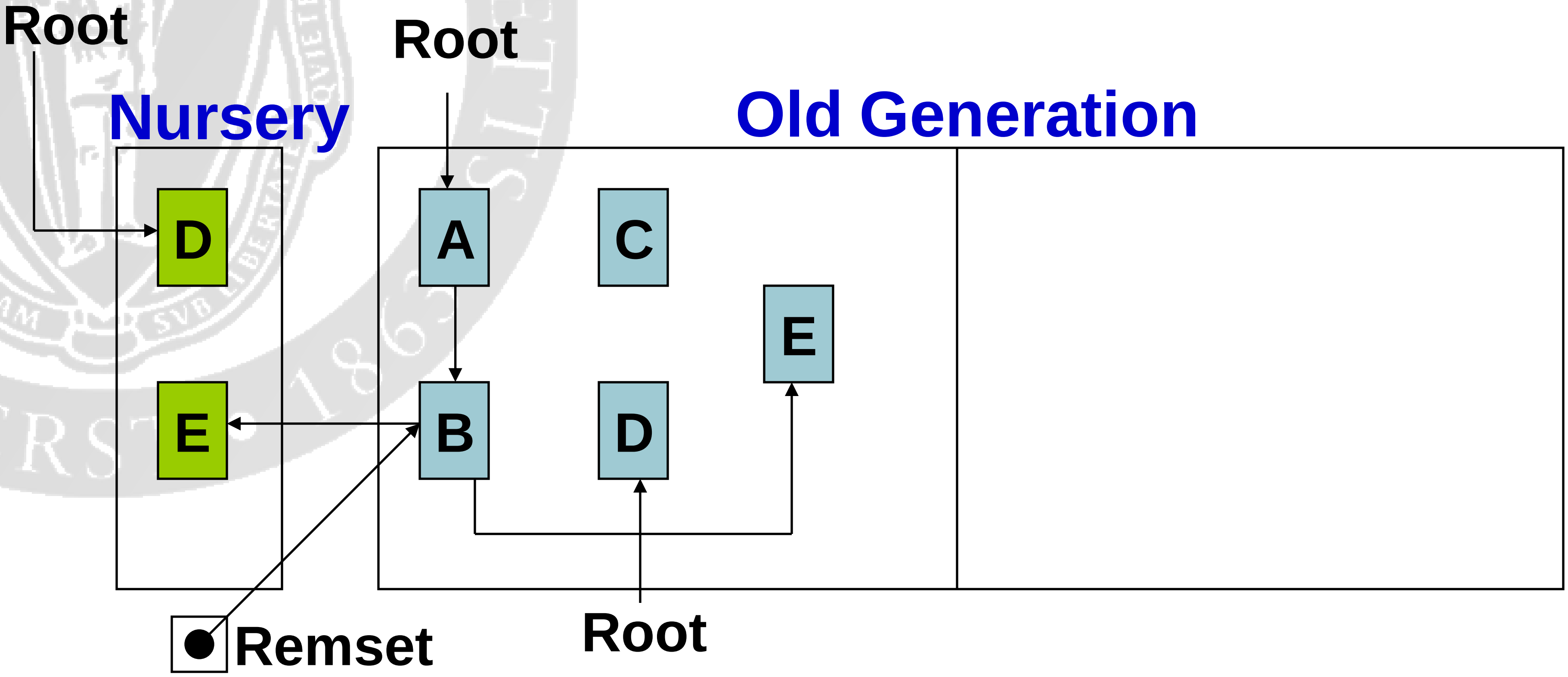
- Divide heap into regions called generations
- Generations segregate objects by age
- Focus GC effort on younger objects



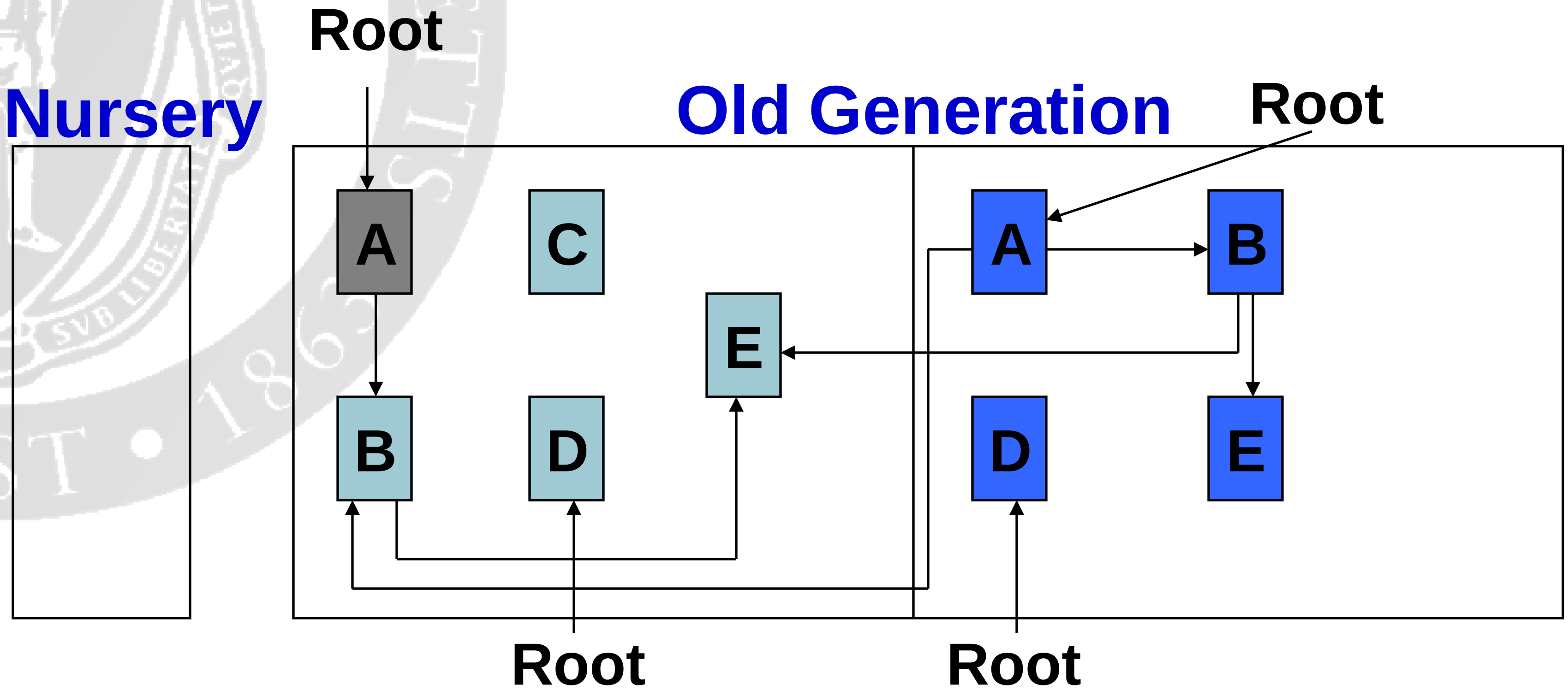
Nursery

Old Generation

Generational Copying Collection

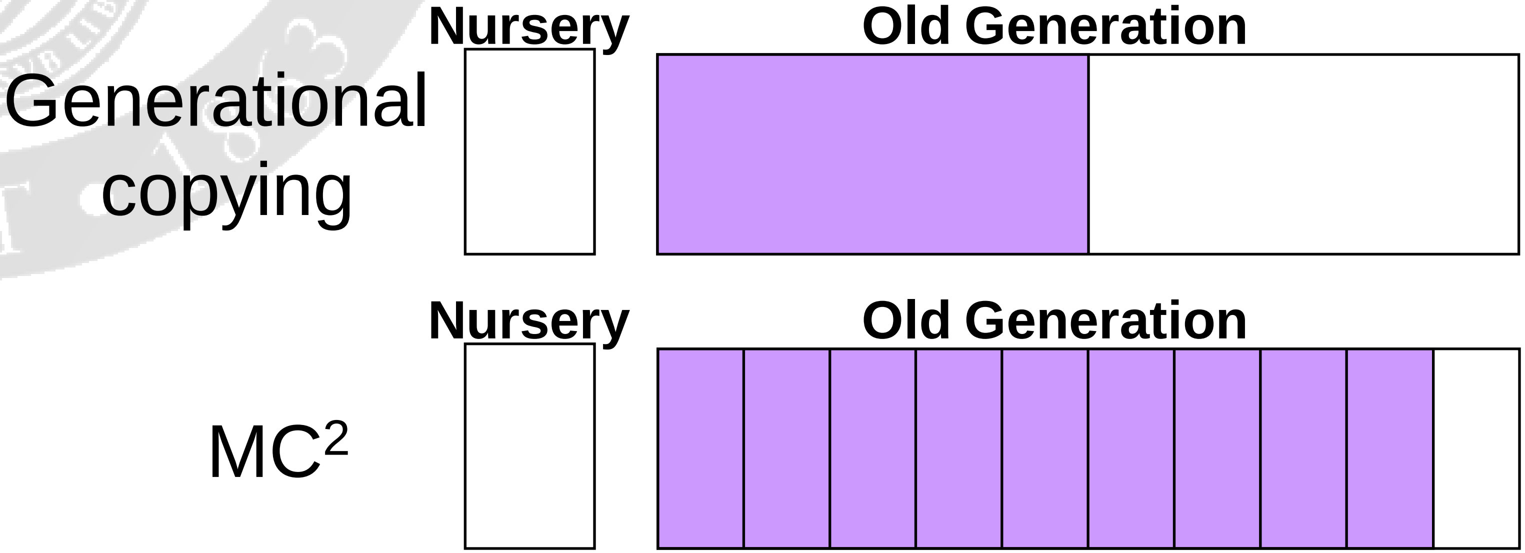


Generational Copying Collection



MC^2 Overview

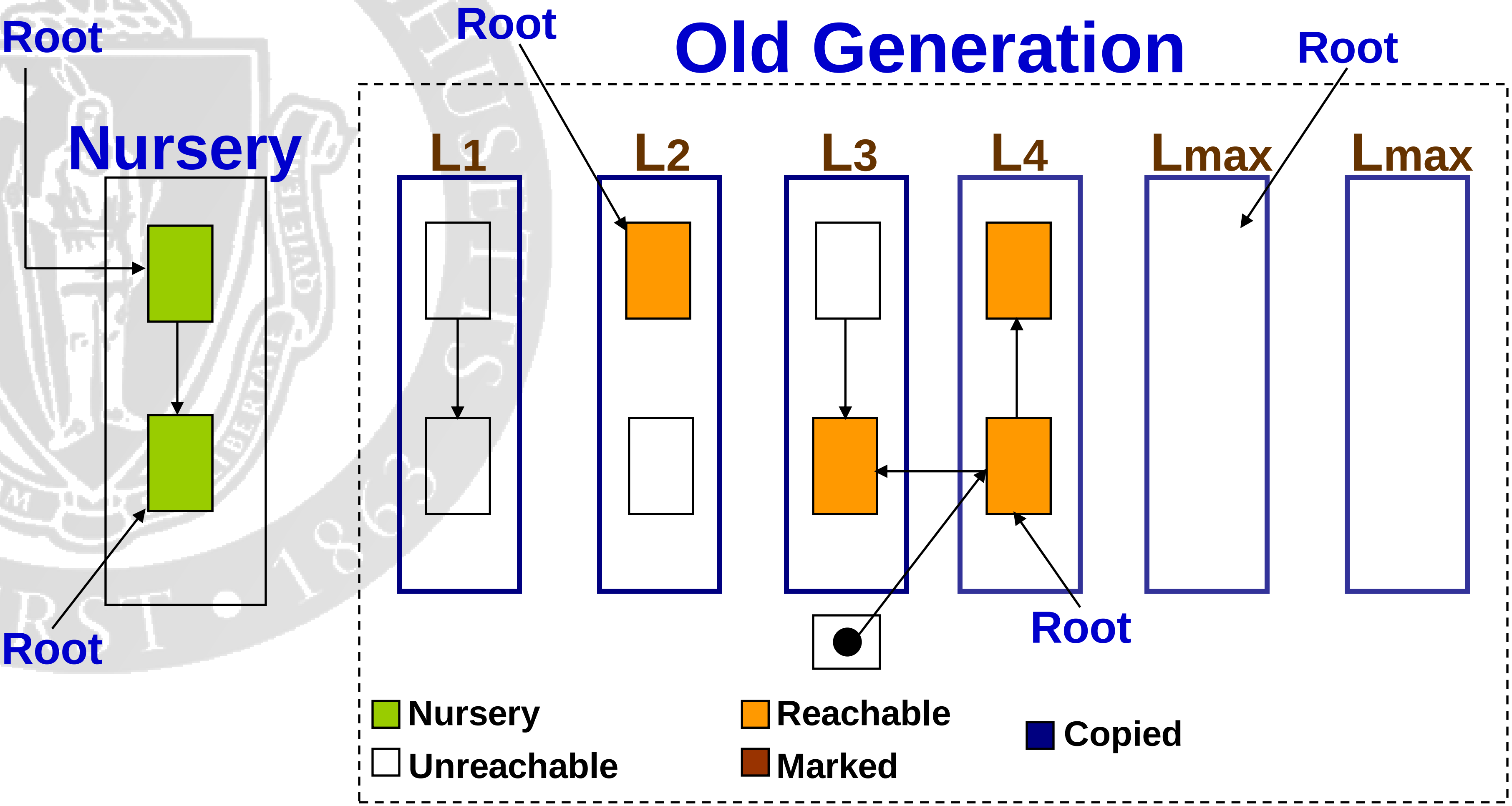
- Extends gen. copying, overcomes 2X overhead
- Divides space into equal size *windows*
- Reserves one or more windows for copying
- Collects in two phases: *mark* and *copy*



MC^2 – Mark Phase

- Logically order old gen. windows
- Three mark phase tasks:
  - Mark reachable objects
  - Calculate live data volume in each window
  - Build per-window remembered sets
- Start when old gen. getting full: 80%, say
- Interleave marking with nursery allocation
  - Do some marking after every *n* bytes allocated
  - Reduces mark phase pauses

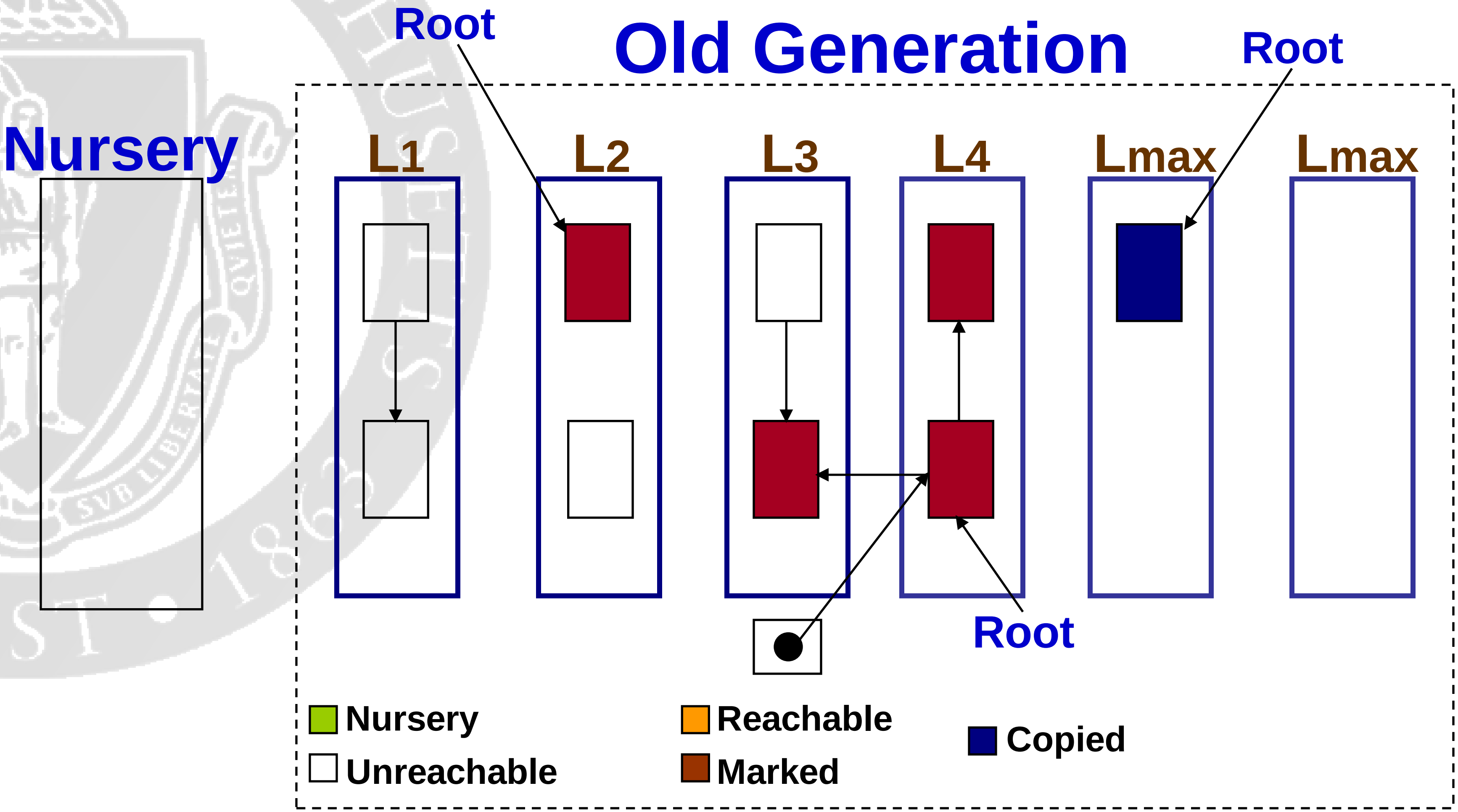
MC^2 Example – Mark Phase



🟩 Nursery
□ Unreachable
🟧 Reachable
🟫 Marked
🟦 Copied

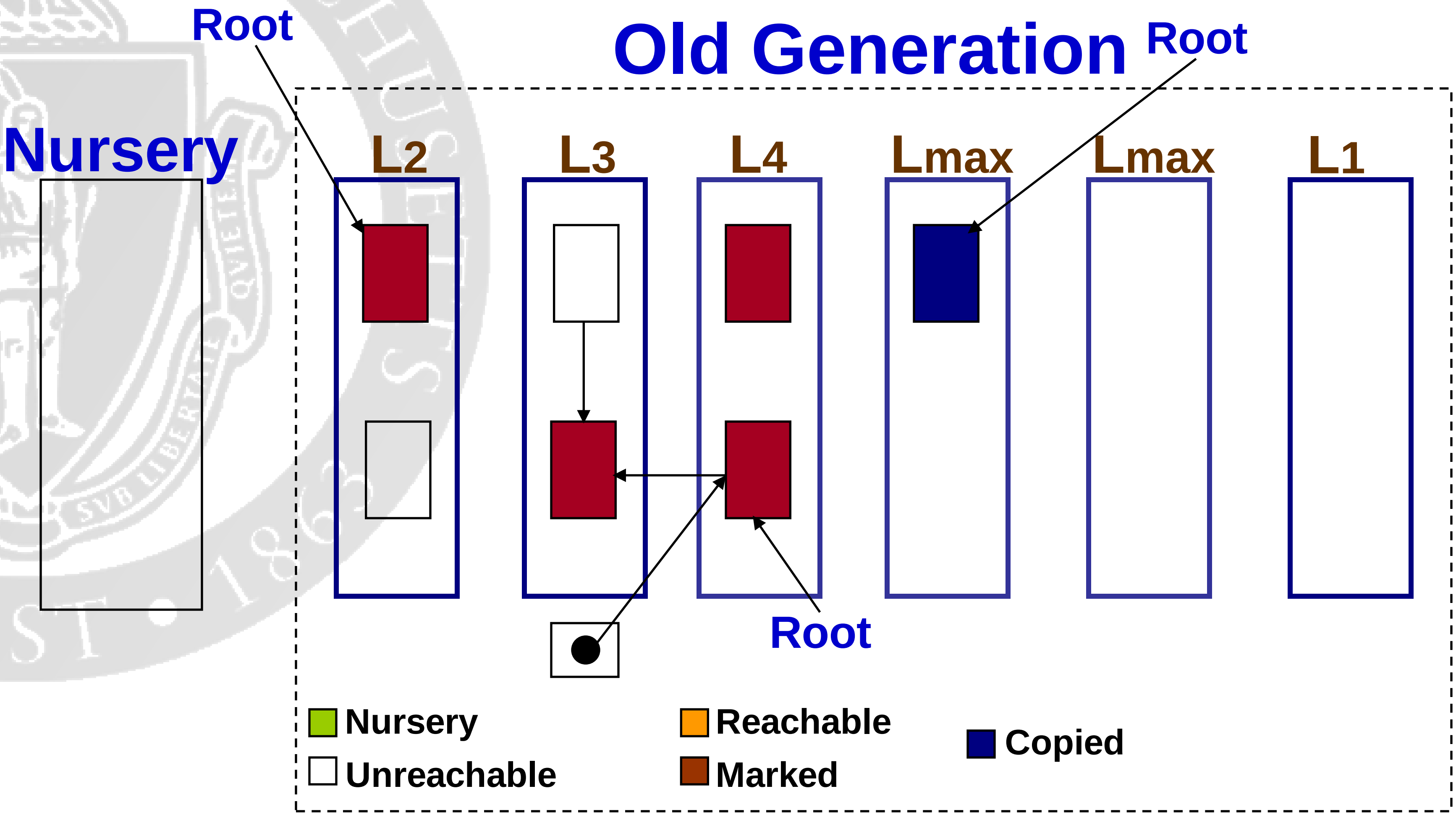
- MC^2 marks two objects during nursery allocation

MC^2 Example – Classify Windows



- Locate high-occupancy windows and discard remsets

MC^2 Example – Classify Windows



Root

Old Generation

Root

Nursery

L2

L3

L4

Lmax

Lmax

L1

Root

🟩 Nursery

□ Unreachable

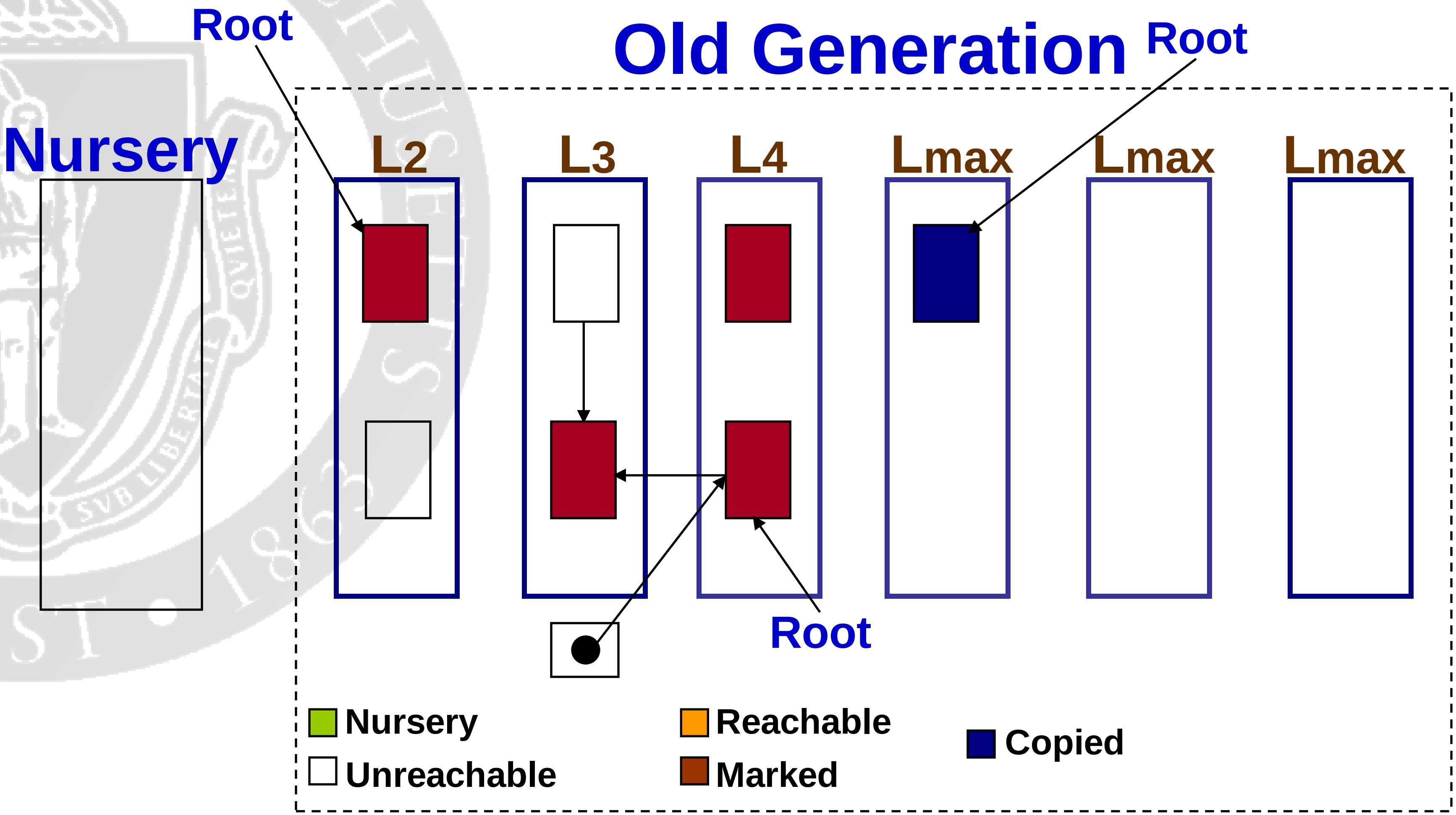
🟧 Reachable

🟫 Marked

🟦 Copied

- Locate high-occupancy windows and discard remsets

MC^2 Example – Classify Windows

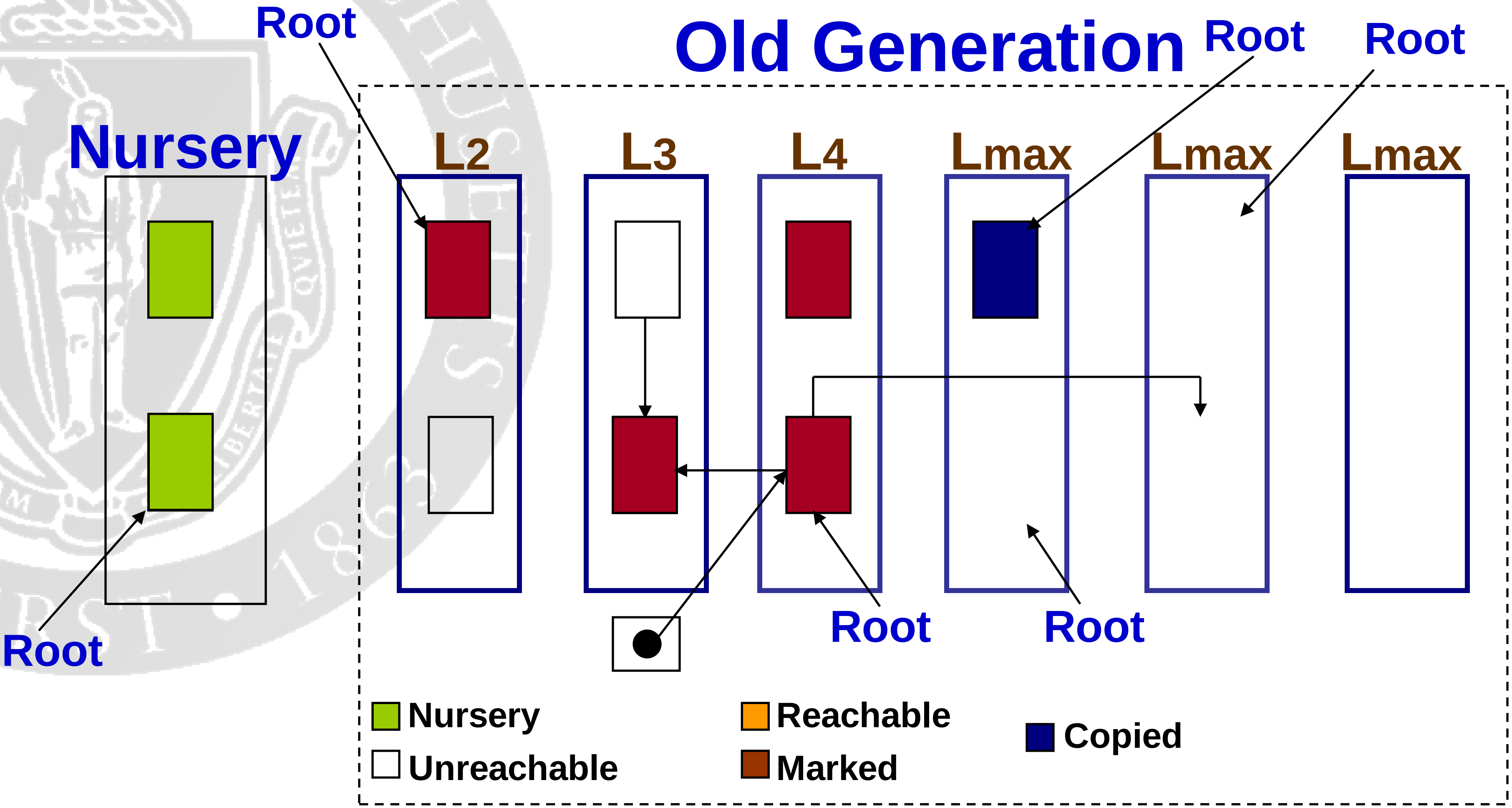


- Locate high-occupancy windows and discard remsets

MC^2 – Copy Phase

- Copy and compact reachable data
- Performed in small increments
  - One windowful of live data copied per increment
- One increment per nursery collection
- High-occupancy windows copied *logically*

MC^2 Example – Copy Phase



Root

Old Generation

Root

Root

Nursery

L2

L3

L4

Lmax

Lmax

Lmax

Root

Root

Root

🟩 Nursery

□ Unreachable

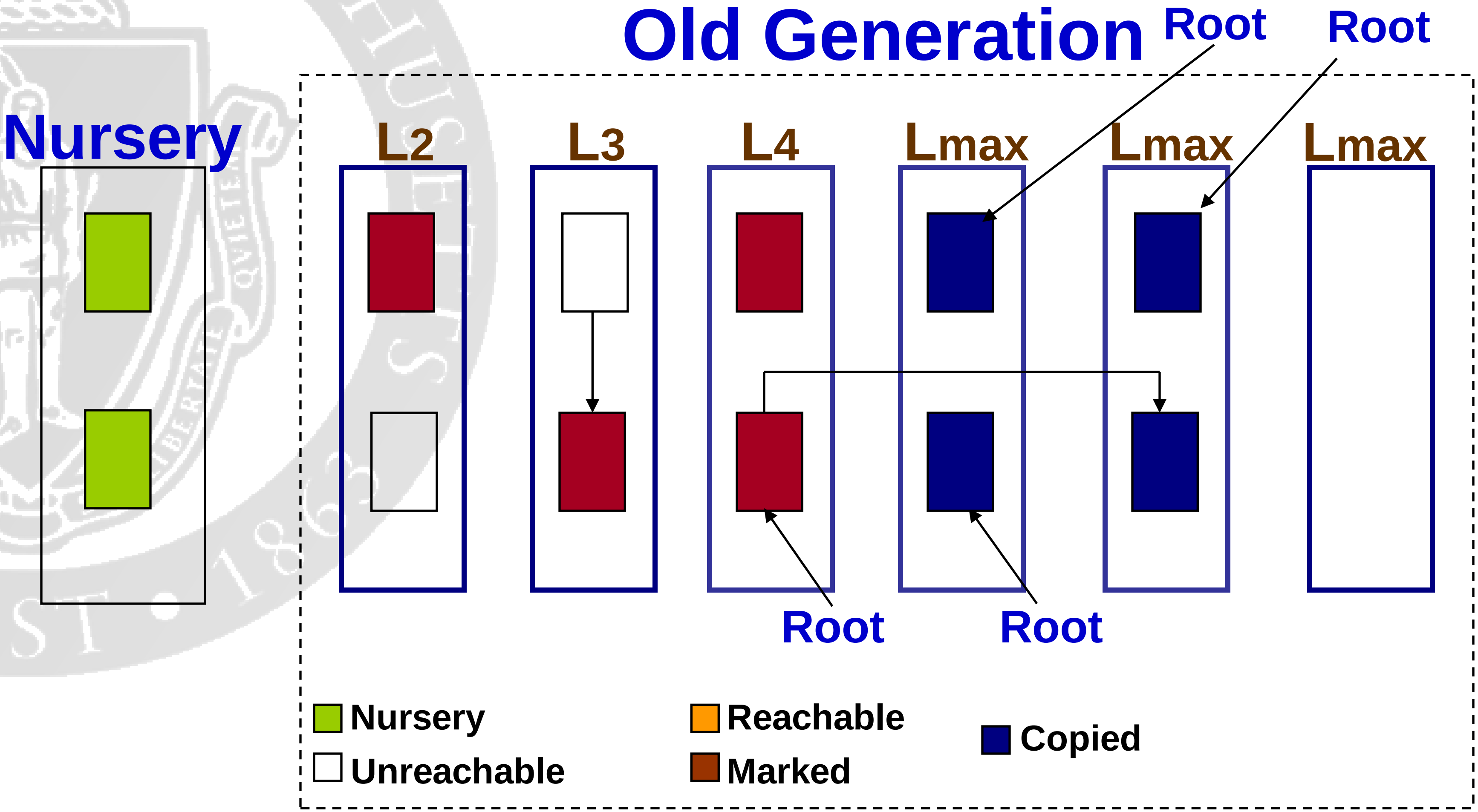
🟧 Reachable

🟫 Marked

🟦 Copied

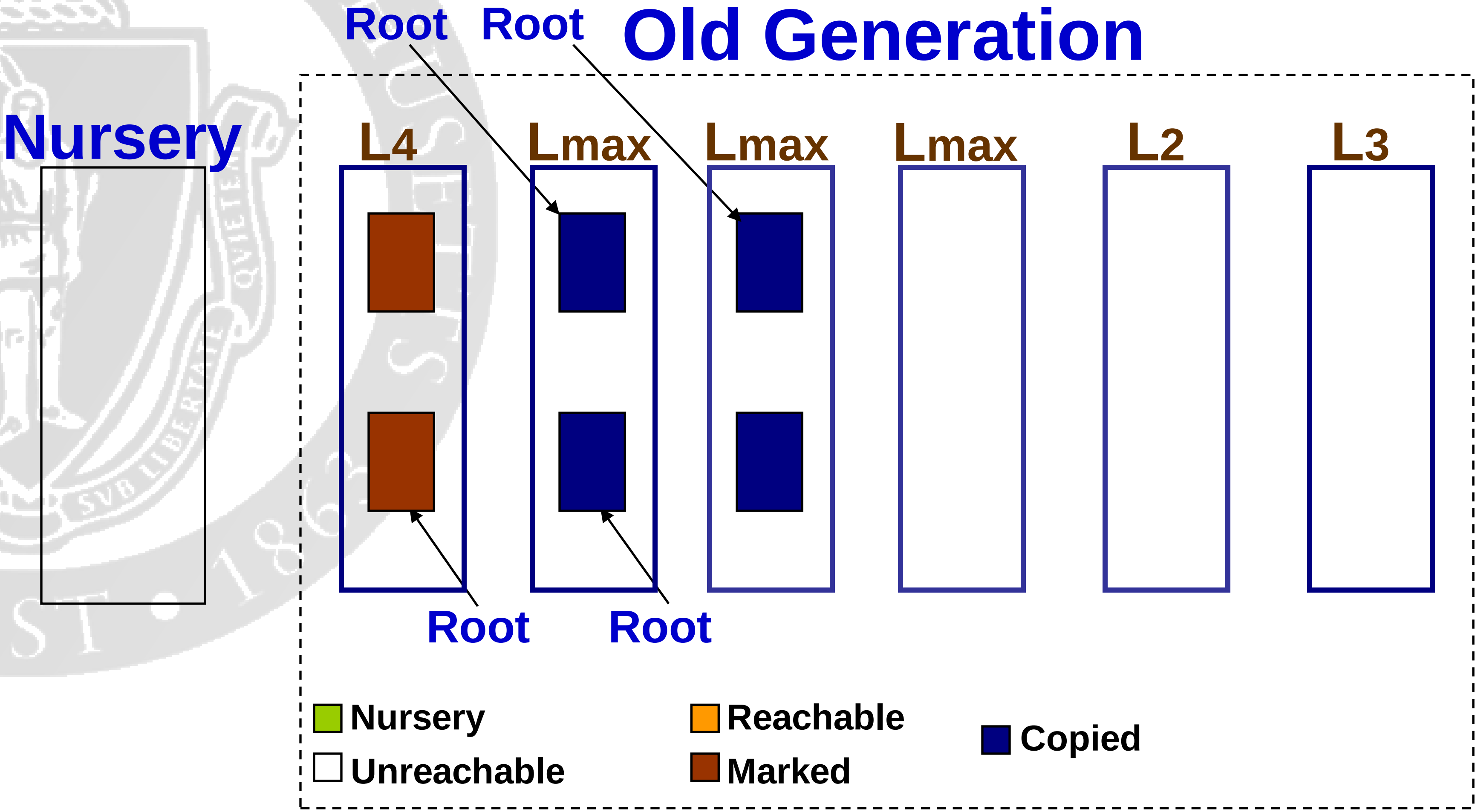
- Copy a window worth of data during nursery collection

MC^2 Example – Copy Phase



- Copy a window worth of data during nursery collection

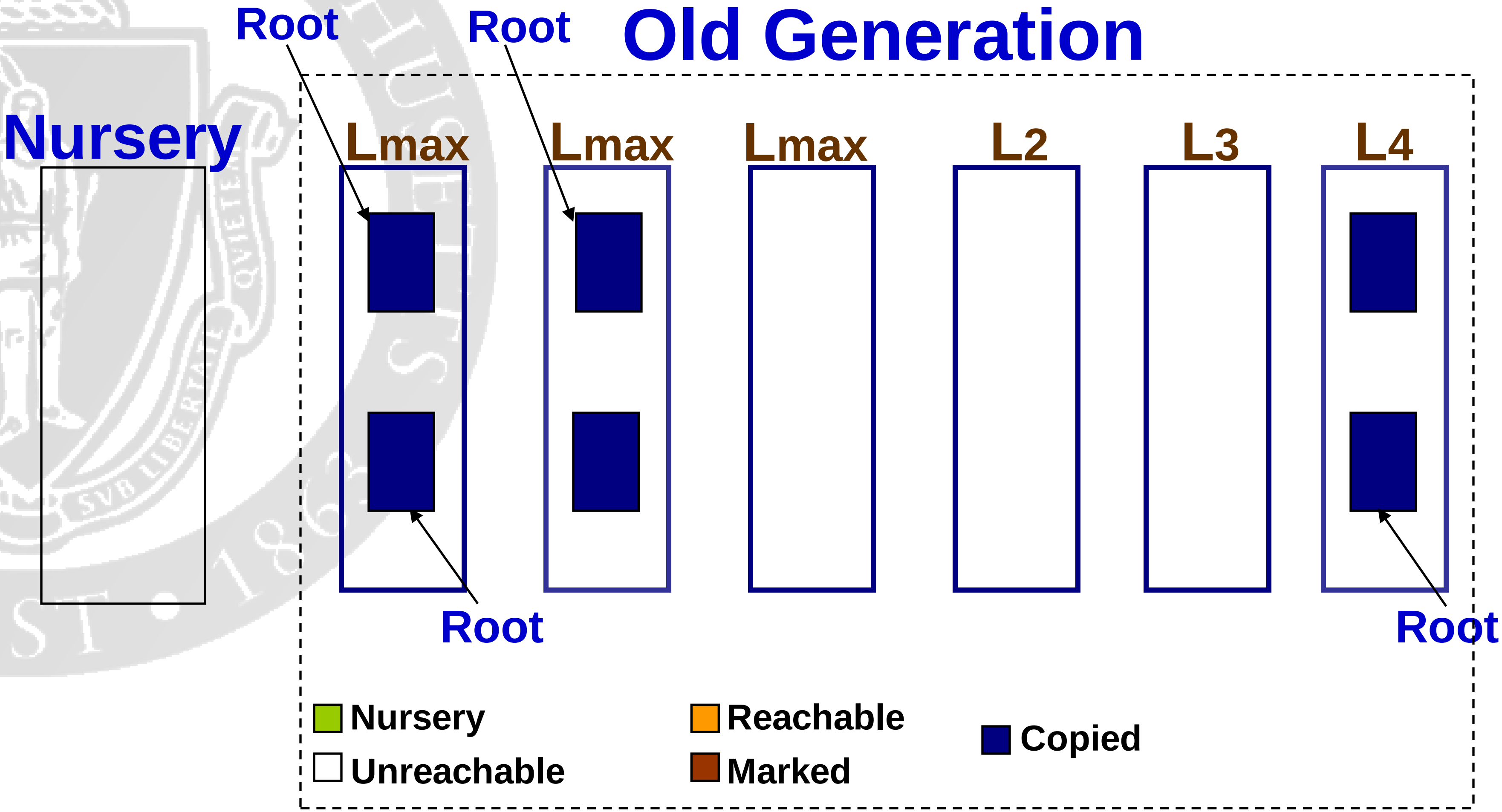
MC^2 Example – Copy Phase



🟩 Nursery
□ Unreachable
🟧 Reachable
🟫 Marked
🟦 Copied

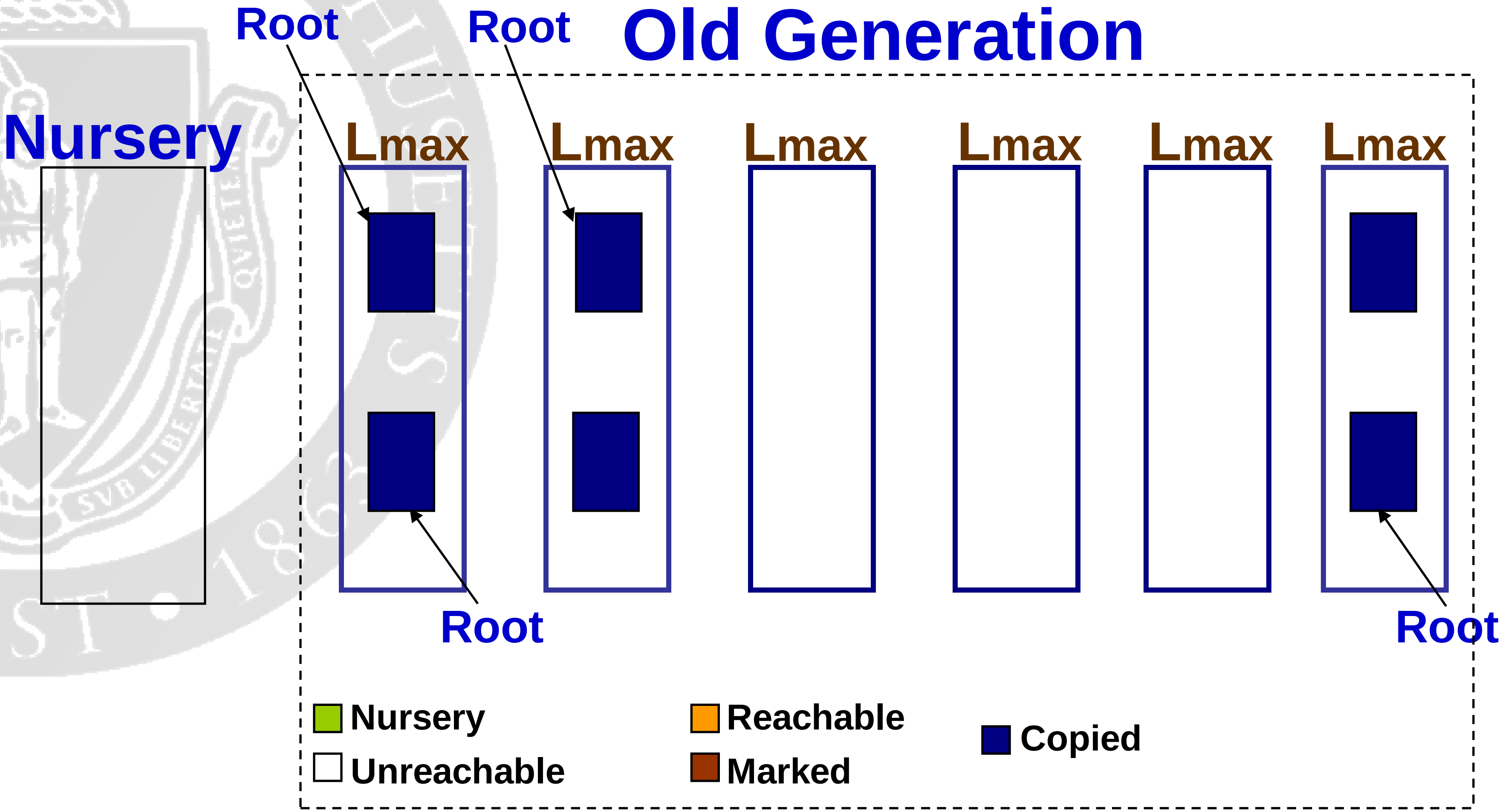
- Copy a window worth of data during nursery collection

MC^2 Example – Copy Phase



- Copy a window worth of data during nursery collection

MC^2 Example – Copy Phase

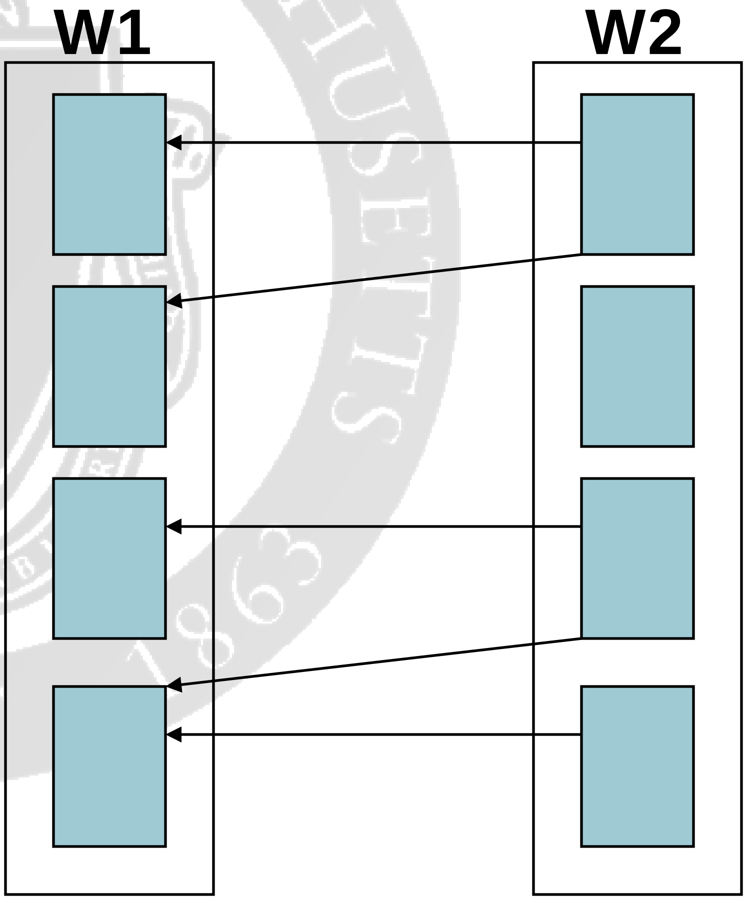


- Copy a window worth of data during nursery collection

Algorithmic Details

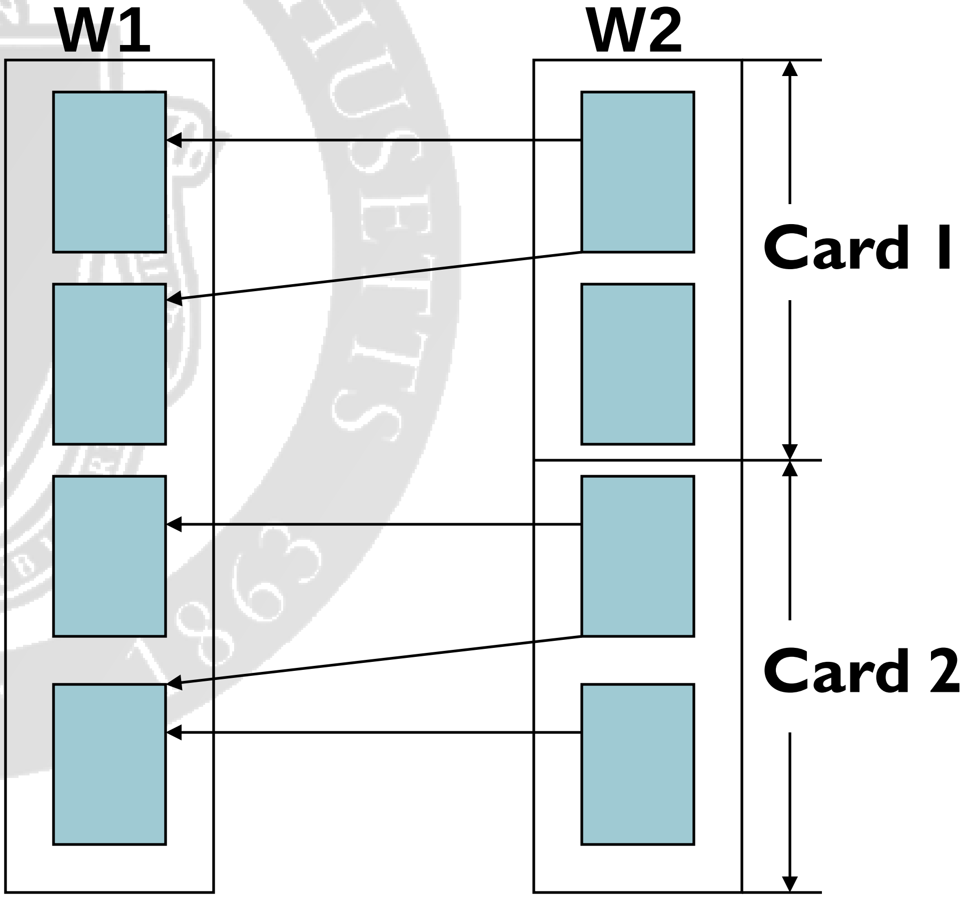
- Large remembered sets
  - Bounding space overhead
- Popular objects
  - Preventing long pauses

Handling large remembered sets



- Normal remembered set for W1 stores 5 pointers (20 bytes on a 32 bit machine)

Handling large remembered sets



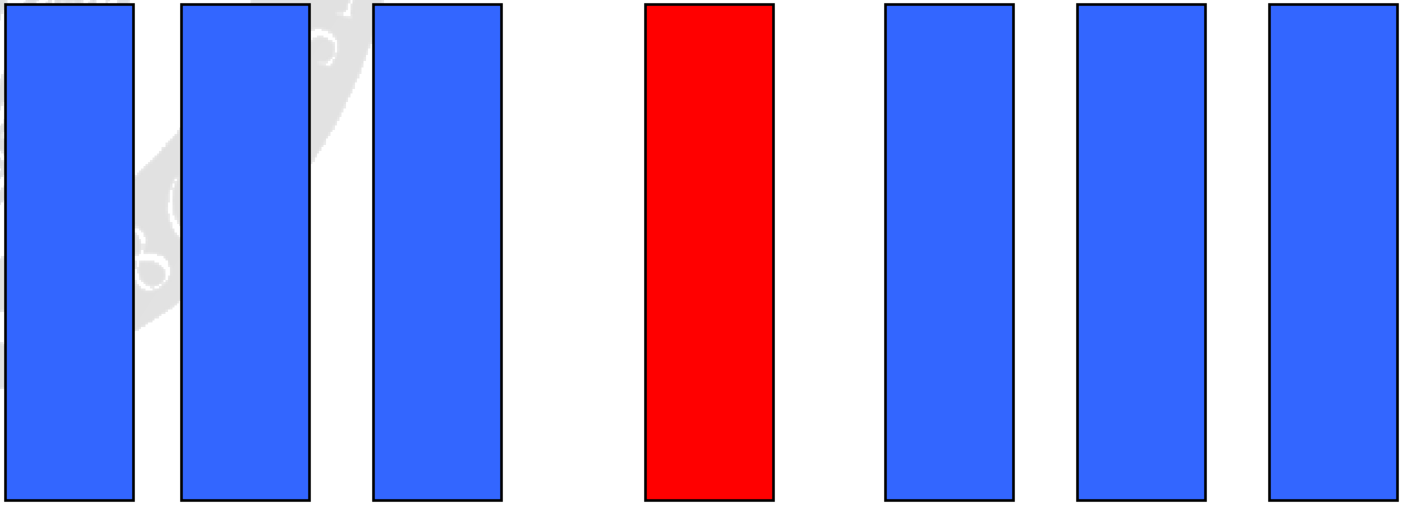
- Card table requires only 2 bytes (one per card)

Handling large remembered sets

- Set a limit on the total remembered set size (e.g., 5% of total heap space).
- Replace large remsets with card table when total size approaches limit
- Good tradeoff between speed and space utilization

Handling popular objects

- Divide heap into small *physical* windows
- Normally copy a group of physical windows
- Popular physical windows not grouped



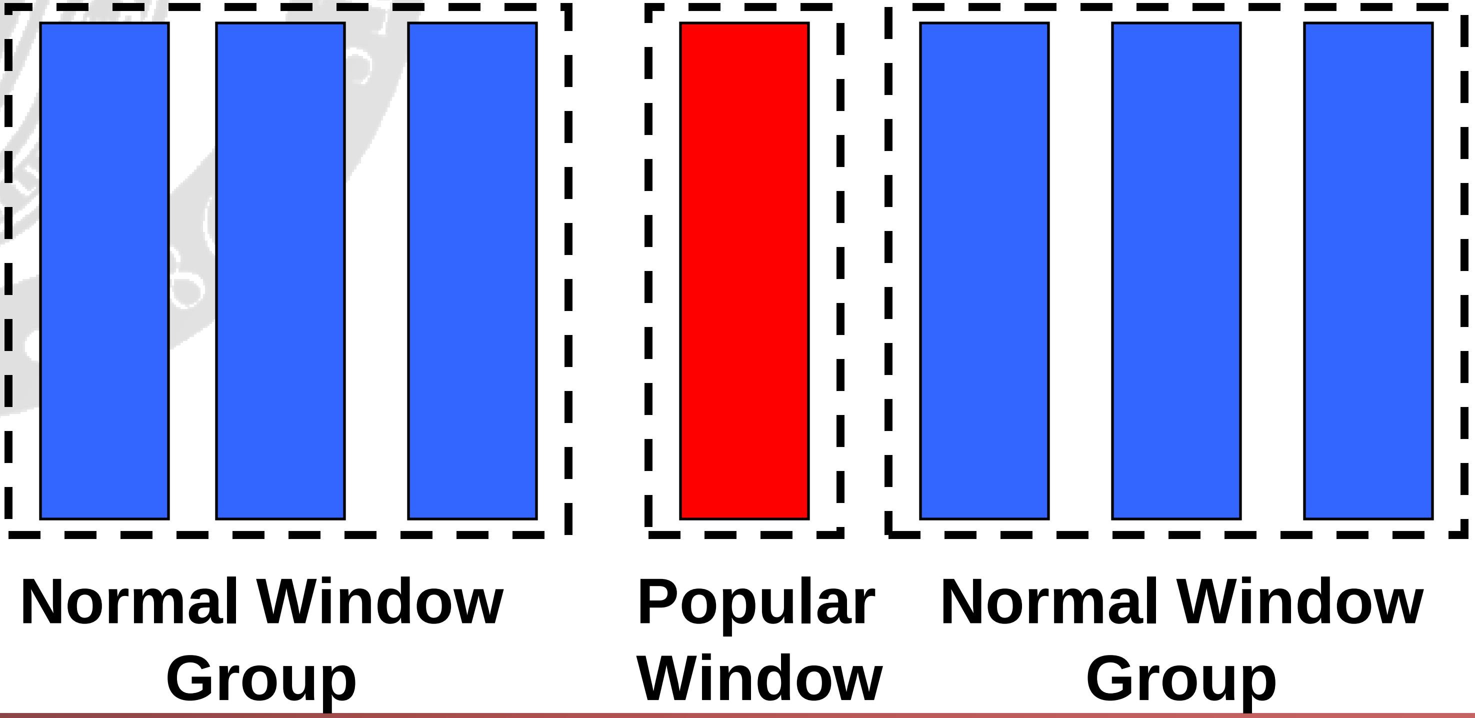
Normal Windows

Popular Window

Normal Windows

Handling popular objects

- Divide heap into small *physical* windows
- Normally copy a group of physical windows
- Popular physical windows not grouped



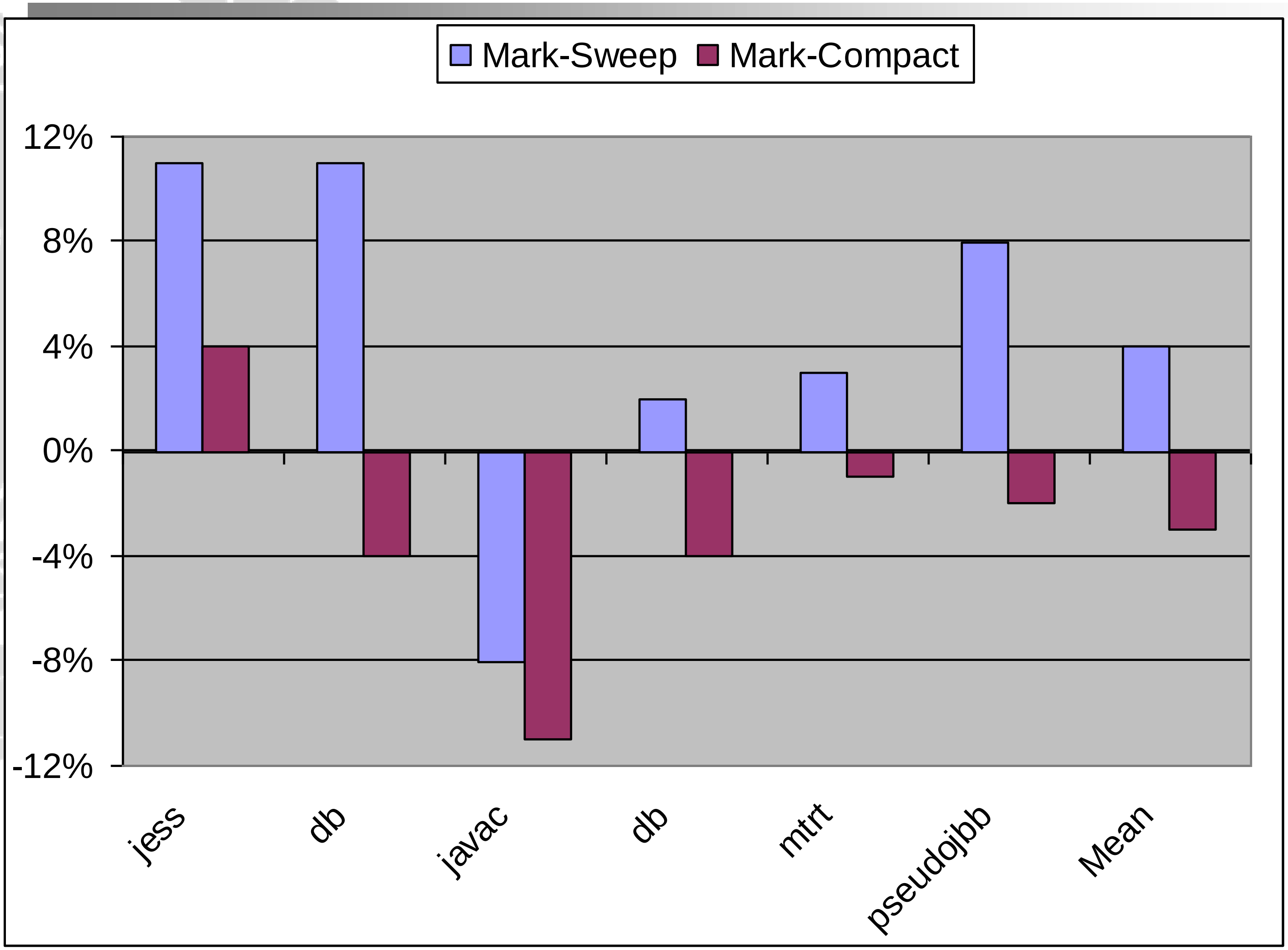
Handling popular objects

- Identify popular object while converting remset to card table
- Isolate popular object at high end of heap
  - Do not need to maintain references to the objects
- Copying a popular object can cause a long pause, but does not recur

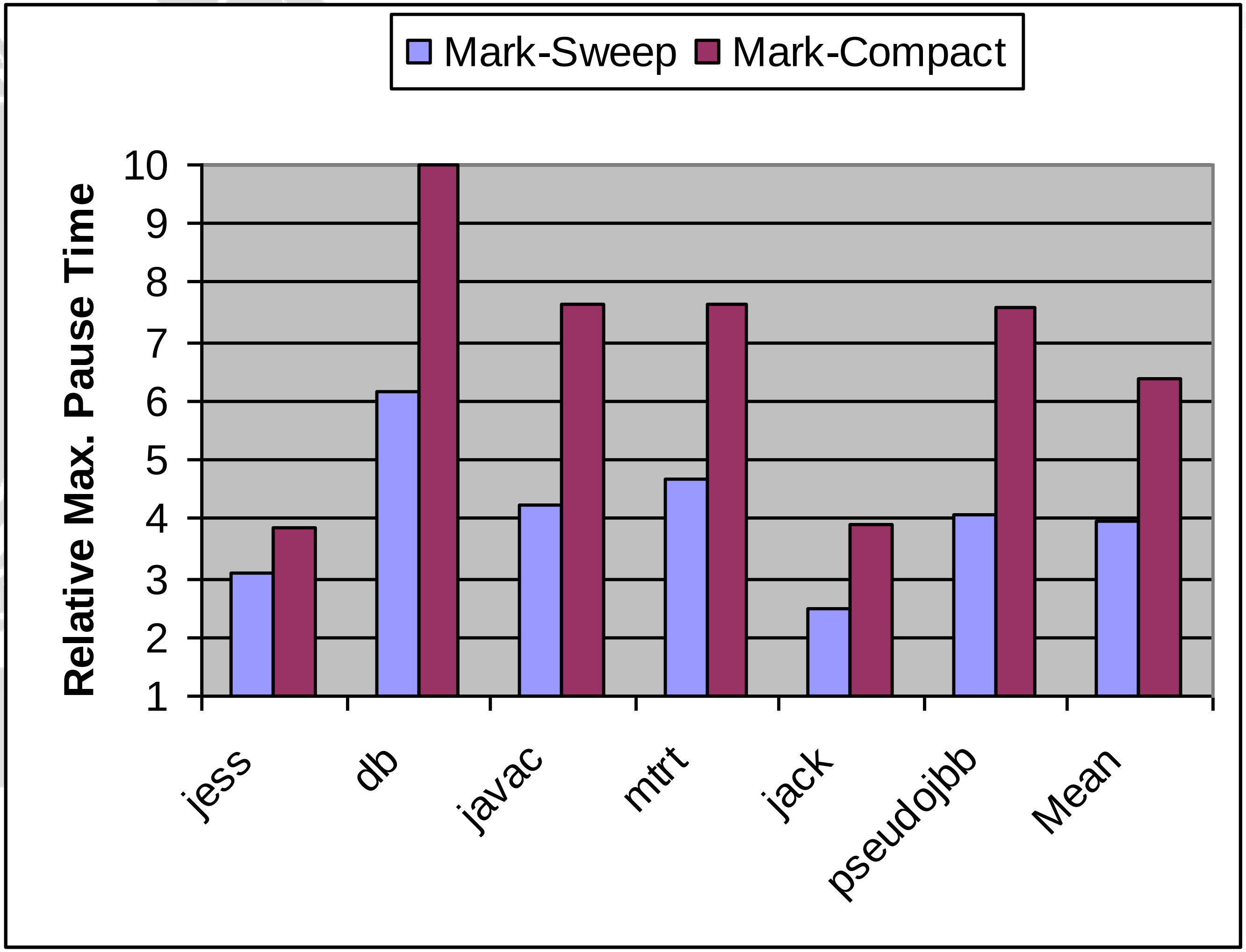
Experimental Results

- Implemented in Jikes RVM (2.2.3)/MMTk
- Pentium 4 1.7 GHz, 512MB memory, RedHat Linux 2.4.7-10
- Benchmarks: SPECjvm98, pseudojbb
- Collectors evaluated
  - Generational Mark-Sweep (MS)
  - Generational Mark-(Sweep)-Compact (MSC)
  - MC^2
- MSC, MC^2 use separate code and data spaces
- Results: Execution time, pause time in a heap 1.8x program live size

Execution Time relative to MC^2 (Heap Size = 1.8x max. live size)

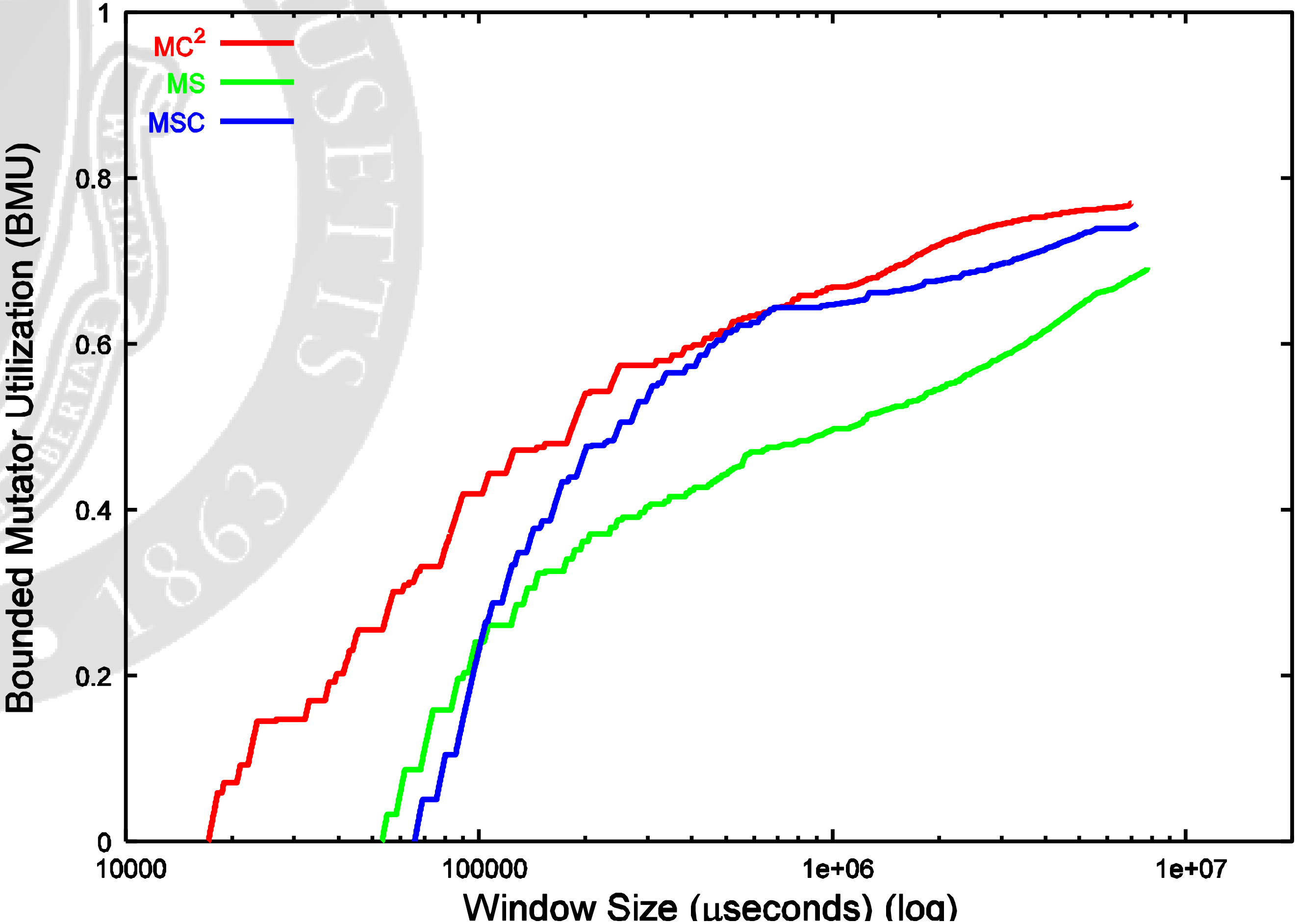


Max. Pause Time relative to MC^2 (Heap Size = 1.8x max. live size)

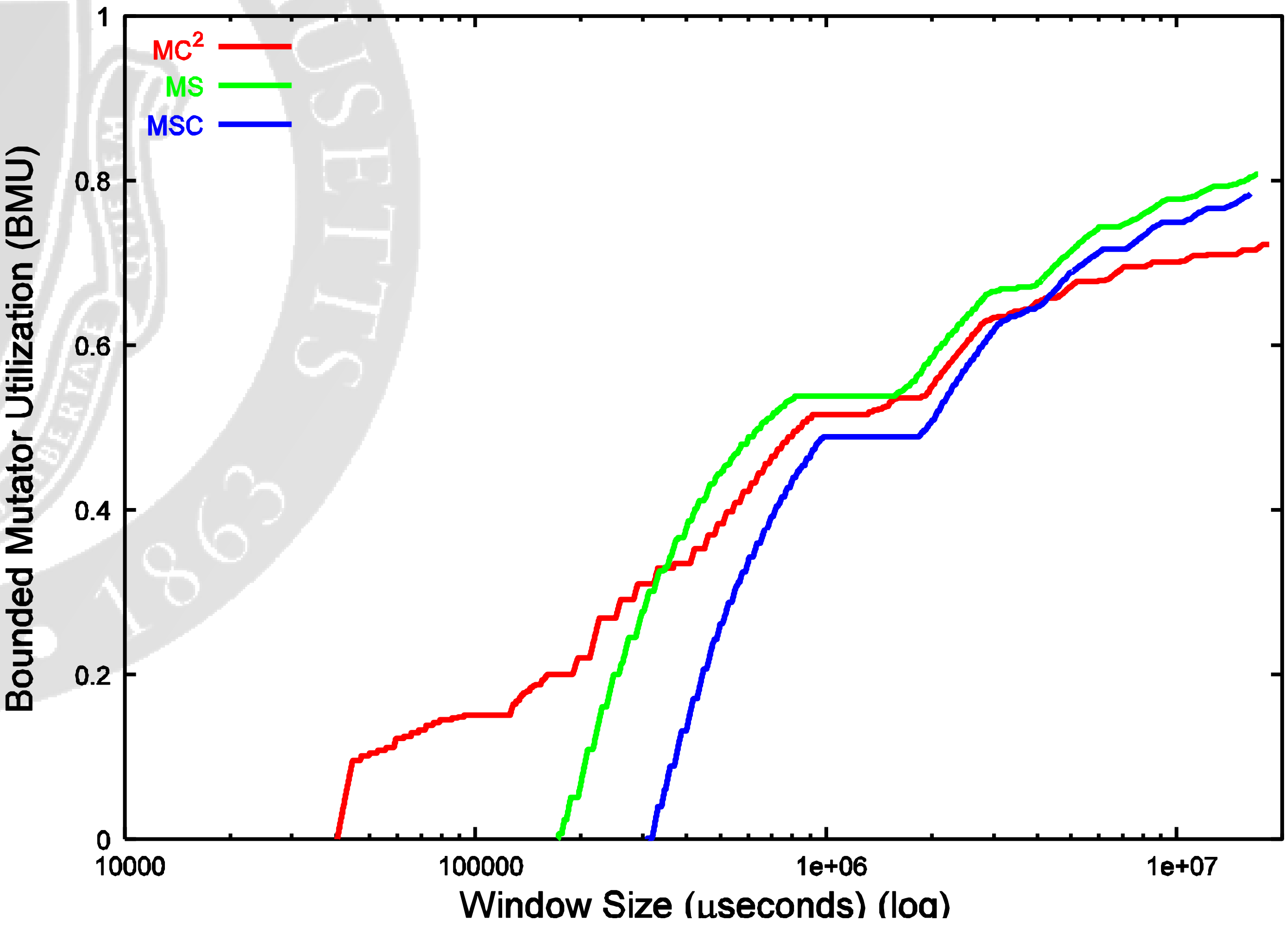


- MC^2 max. pause range: 17-41ms

Bounded Mutator Utilization (jess)



Bounded Mutator Utilization (javac)



Conclusions

- MC^2: suitable for handheld devices with soft real time requirements
  - Low space overhead (50-80%)
  - Good throughput (3-4% slower than non-incremental compacting collector)
  - Short pause times (17-41ms, factor of 6 lower than non-incremental compacting collector)
  - Well distributed pauses
- Also suitable for desktop environments

Backup Slides

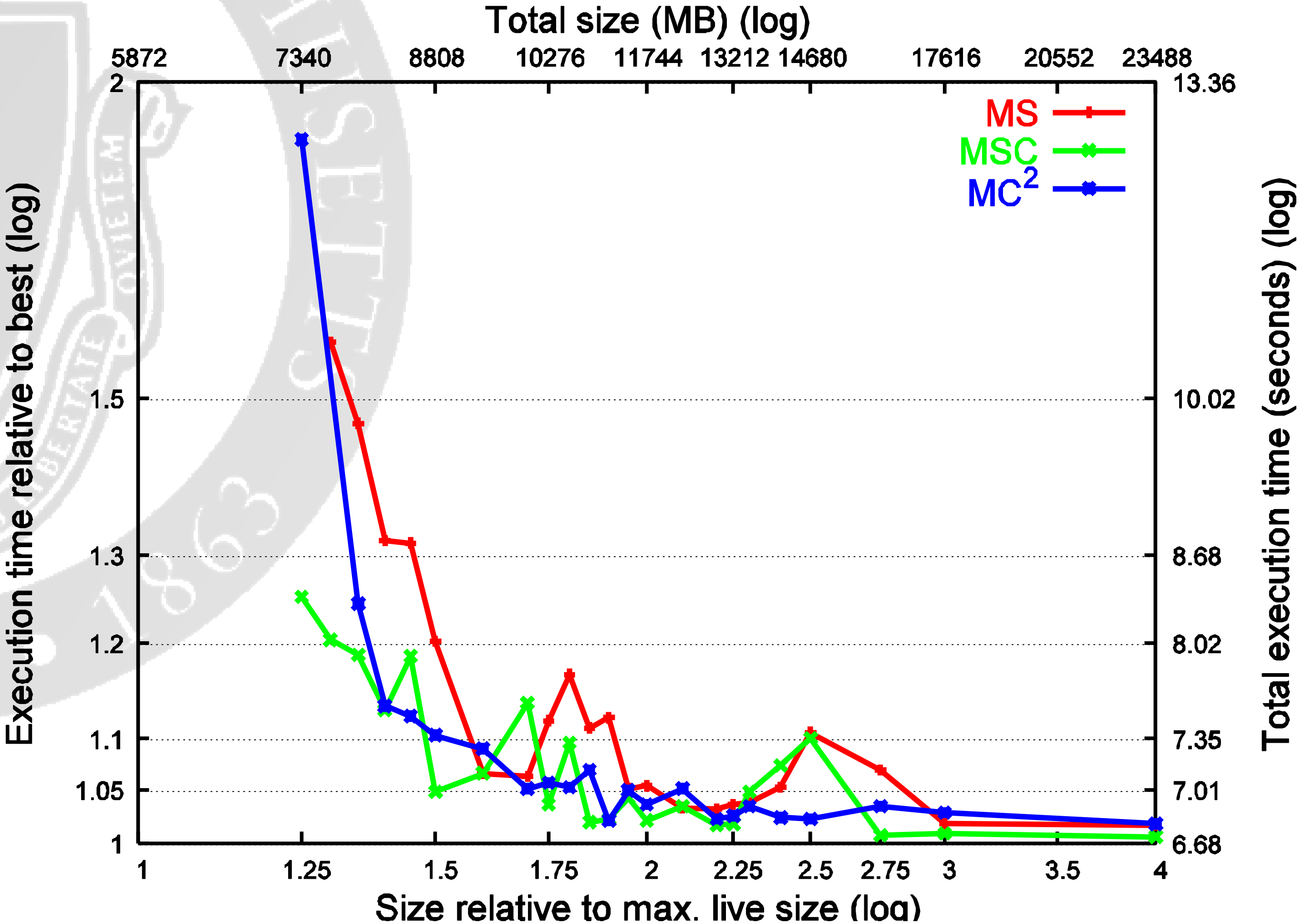
Traditional Tracing Collectors

- Mark-Sweep
  - Fragmentation
  - Locality effects
- Mark-Compact
  - Long pauses
- Copying
  - 2X space overhead

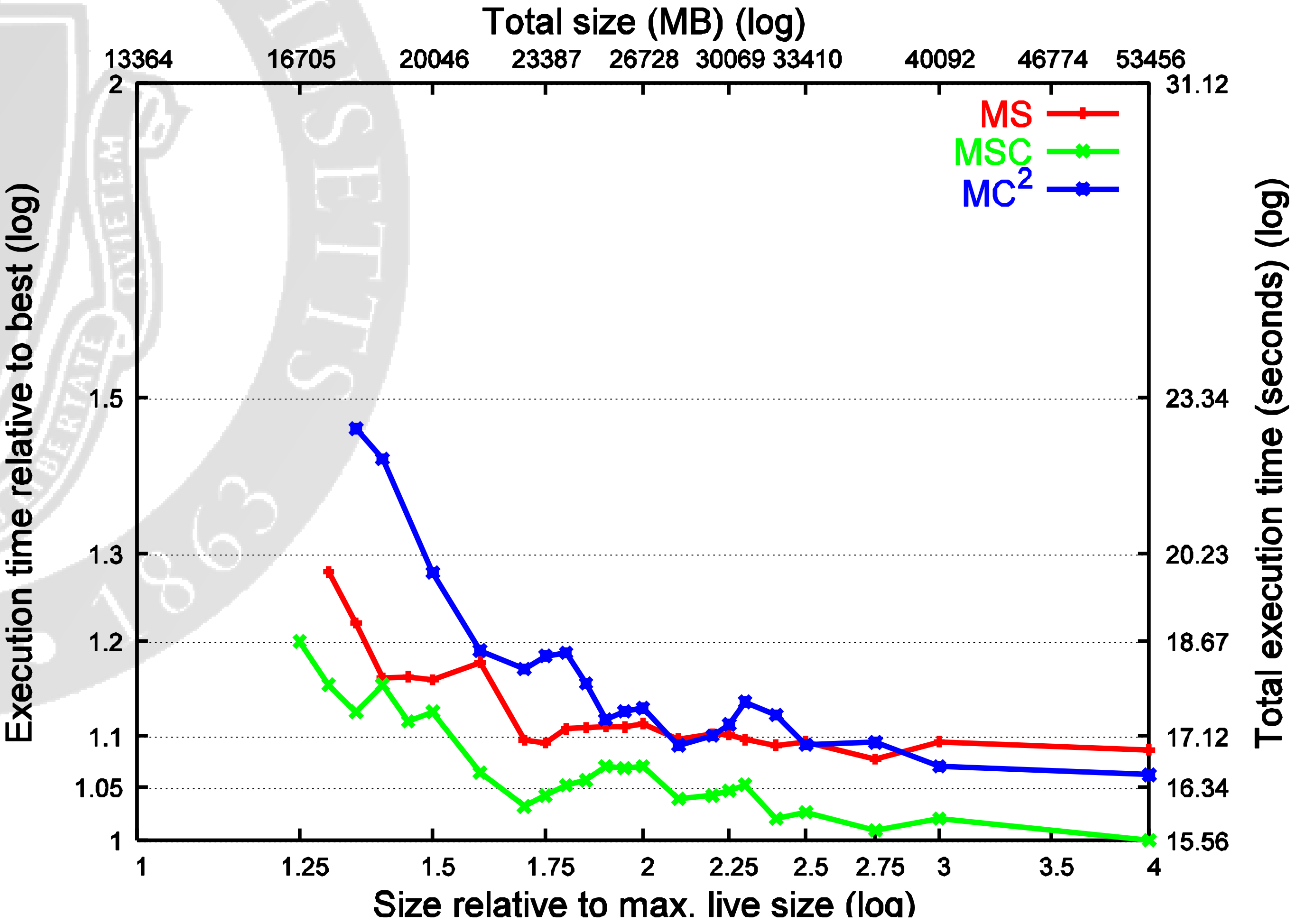
Modern incremental collectors

- Train collector – Hudson and Moss
  - Performs well in moderate size heaps
  - High copying overhead
- Lang and Dupont, Ben Yitzhak et al
  - Good throughput, pause times
  - Do not address metadata overheads
- Bacon, Cheng, Rajan
  - Address memory-constrained device requirements
  - Require advanced compiler optimizations

jess Execution Time



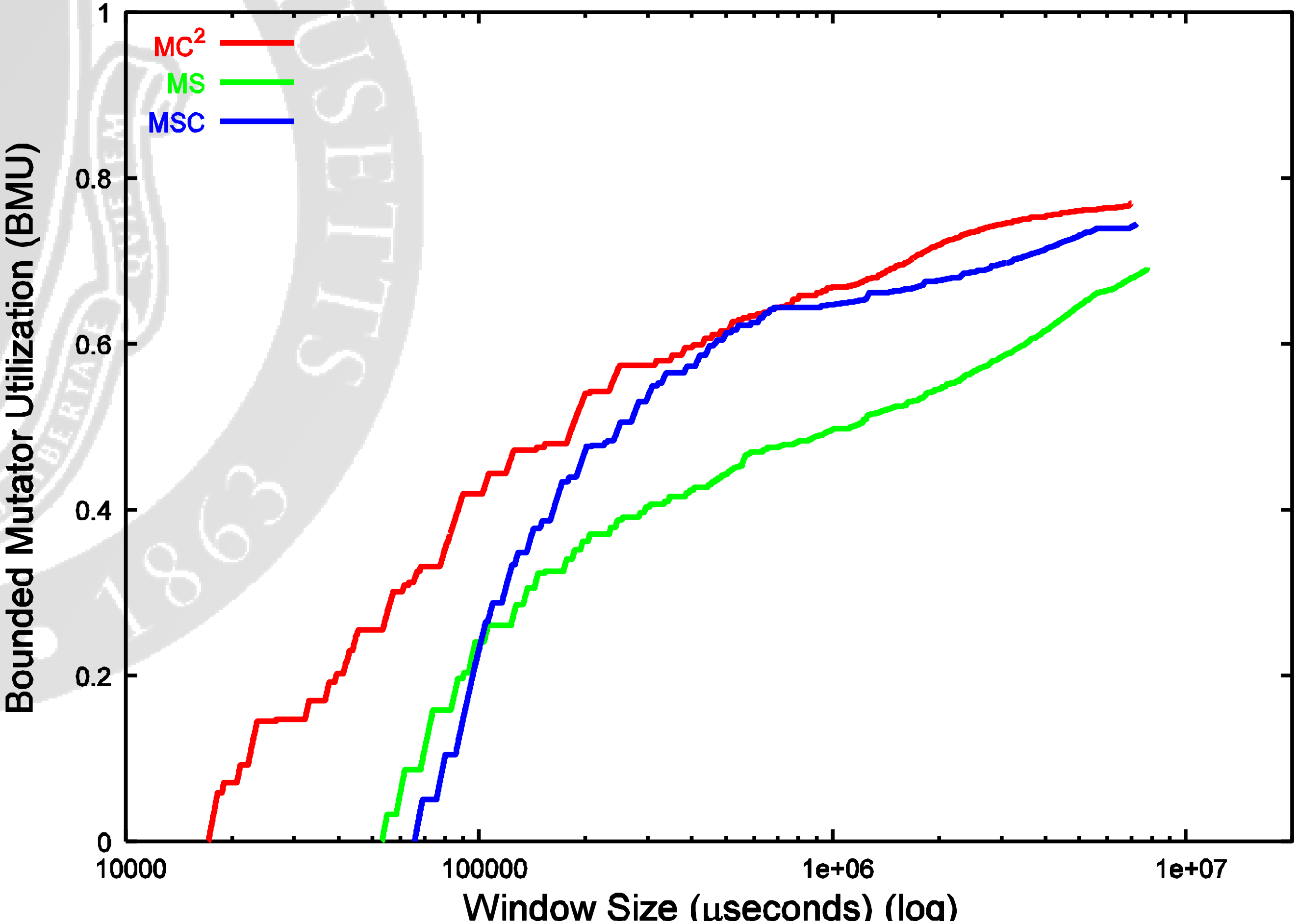
javac Execution Time



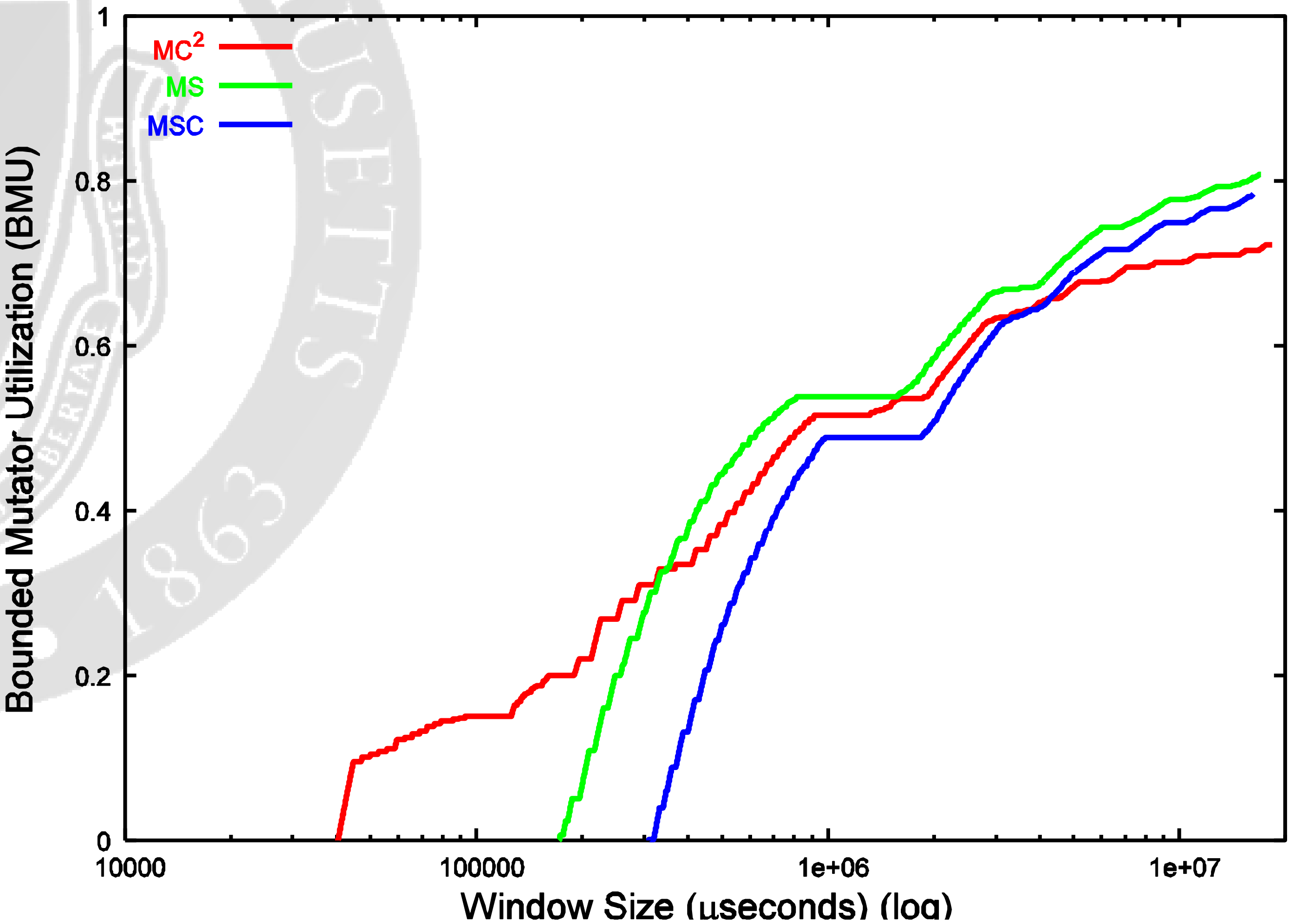
Pause Time, Execution Time Comparison (80% space overhead)

Benchmark	MC^2 MPT (ms)	MS MPT (ms)	MS/MC^2 ET	MSC MPT (ms)	MSC/MC^2 ET
_202_jess	17.2	53.2	1.11	65.7	1.04
_209_db	19.9	123.0	1.11	198.5	0.96
_213_javac	40.4	171.9	0.92	308.9	0.89
_227_mtrt	29.6	138.1	1.02	225.2	0.96
_228_jack	23.9	59.7	1.03	92.9	0.99
pseudojbb	41.5	168.2	1.08	314.5	0.98
Geo. Mean	**27.2**	**107.7**	**1.04**	**172.7**	**0.97**

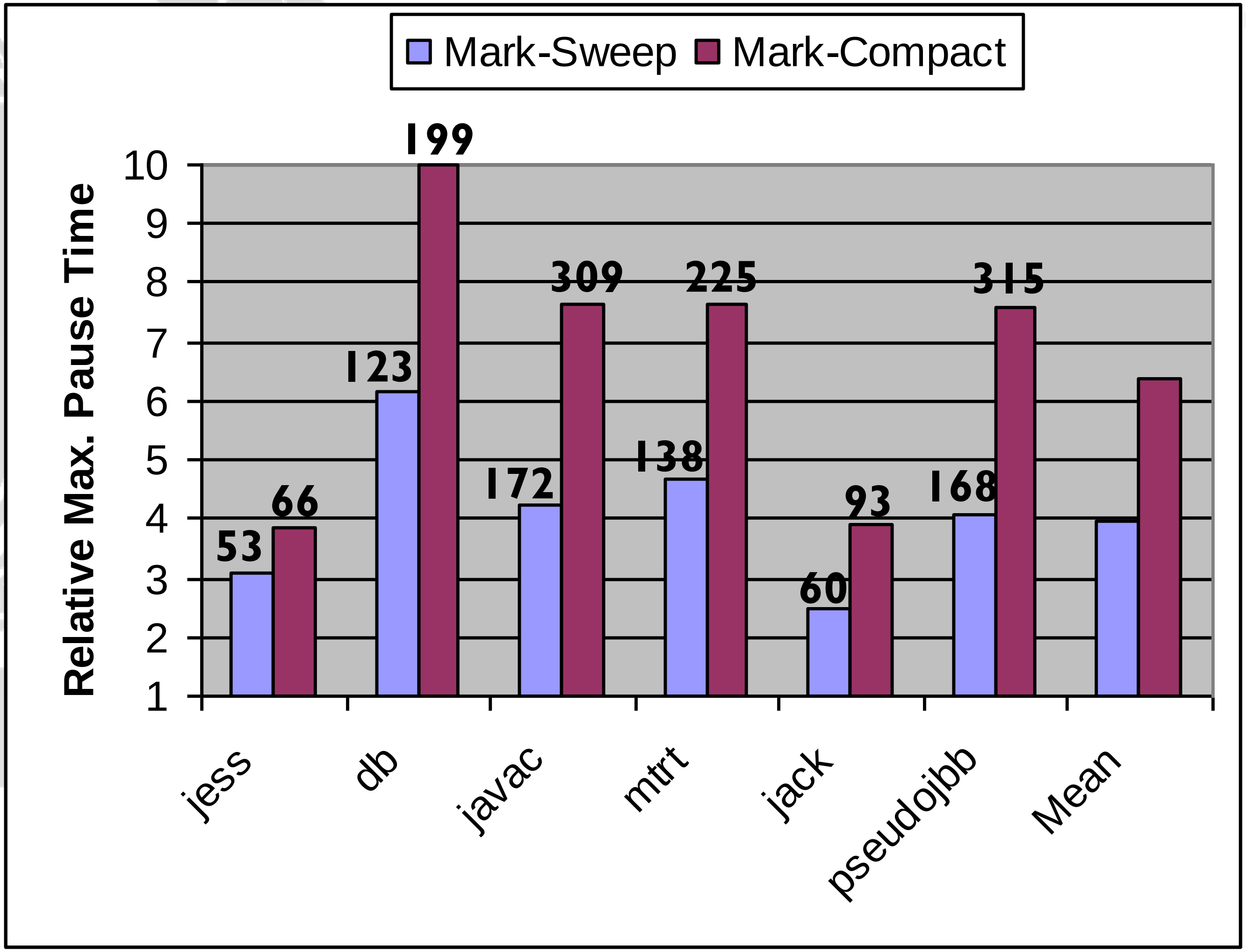
Bounded Mutator Utilization (jess)



Bounded Mutator Utilization (javac)

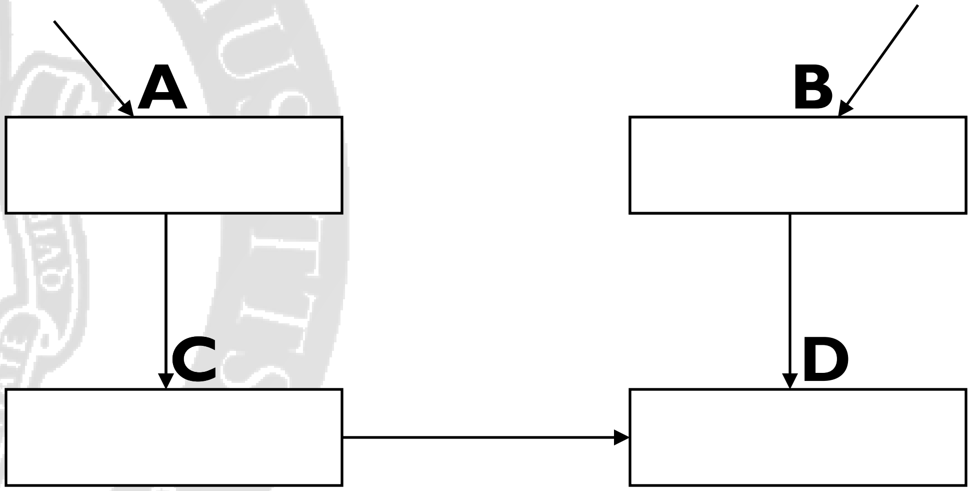


Max. Pause Time relative to MC^2 (Heap Size = 1.8x max. live size)



- MC^2 max. pause range: 17-41ms

Incremental Marking Error



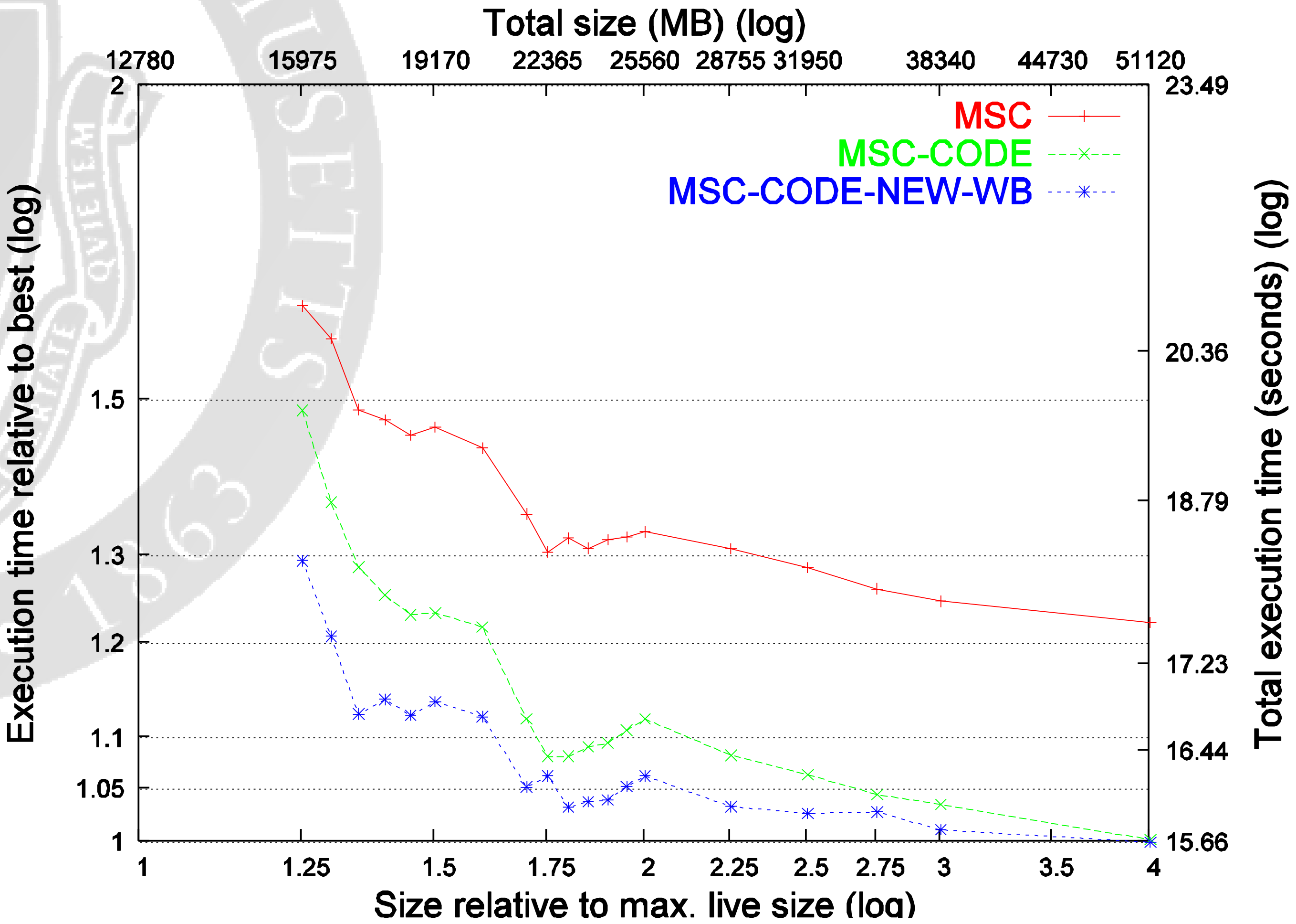
Handling marking error

- Track mutations using write barrier
- Record modified old generation objects
- Scan these modified objects at GC time
- Record interesting slots in remembered sets

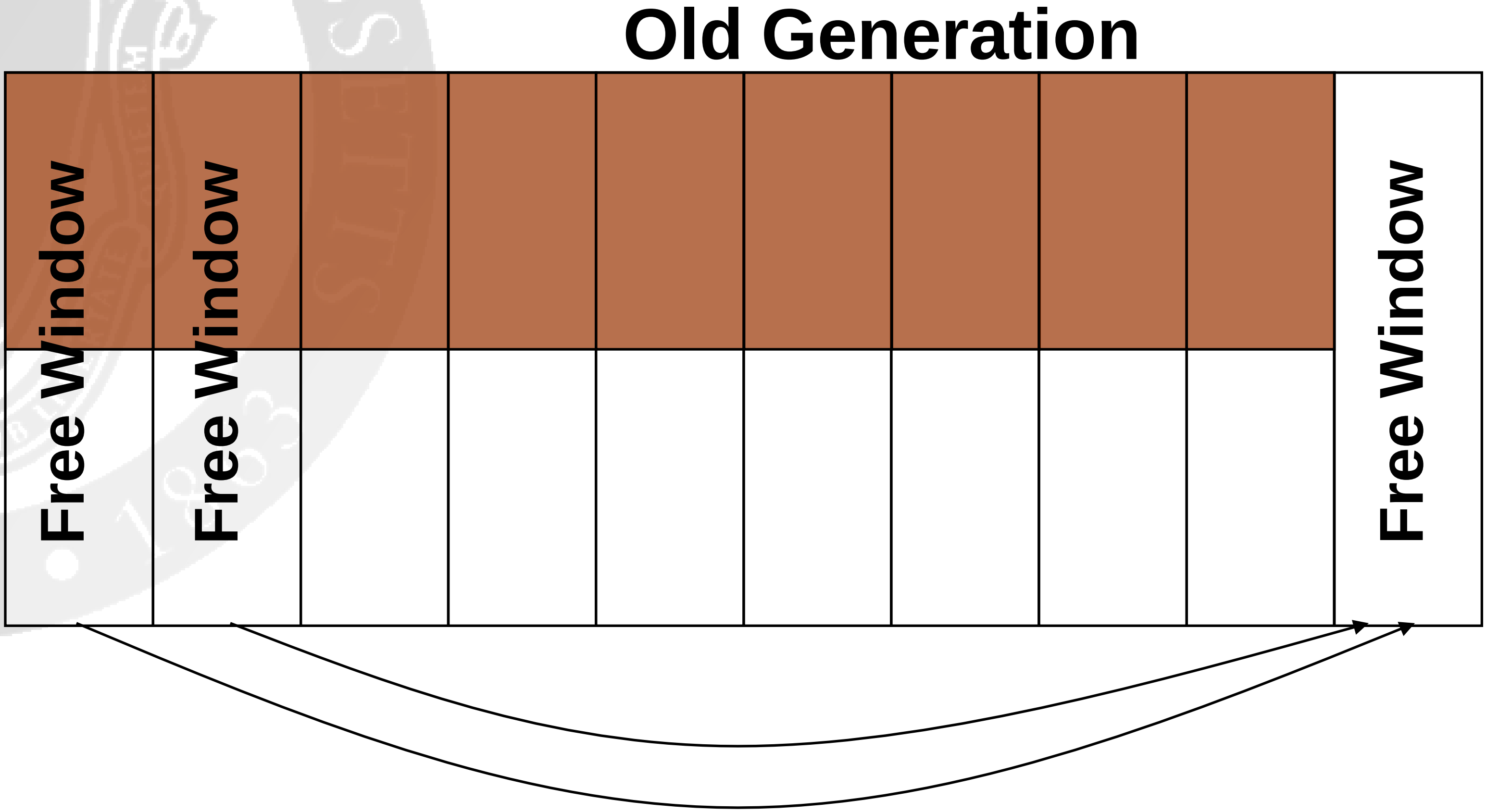
JMTk Heap Layout

Boot Image	Imm. Object Space	Large Object Space	Old Generation	Nursery

_213_javac (MSC execution time)

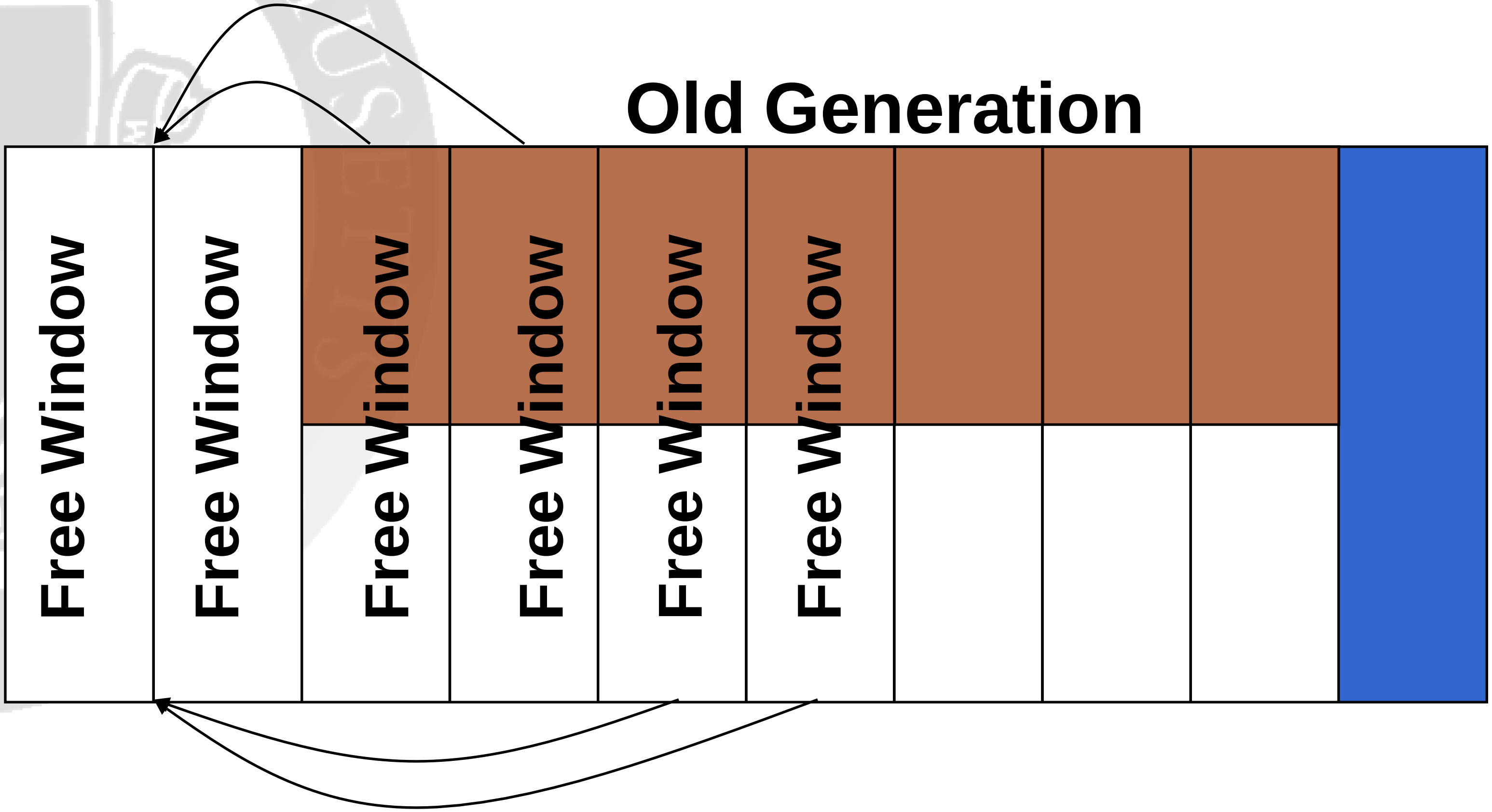


MC^2 Overview



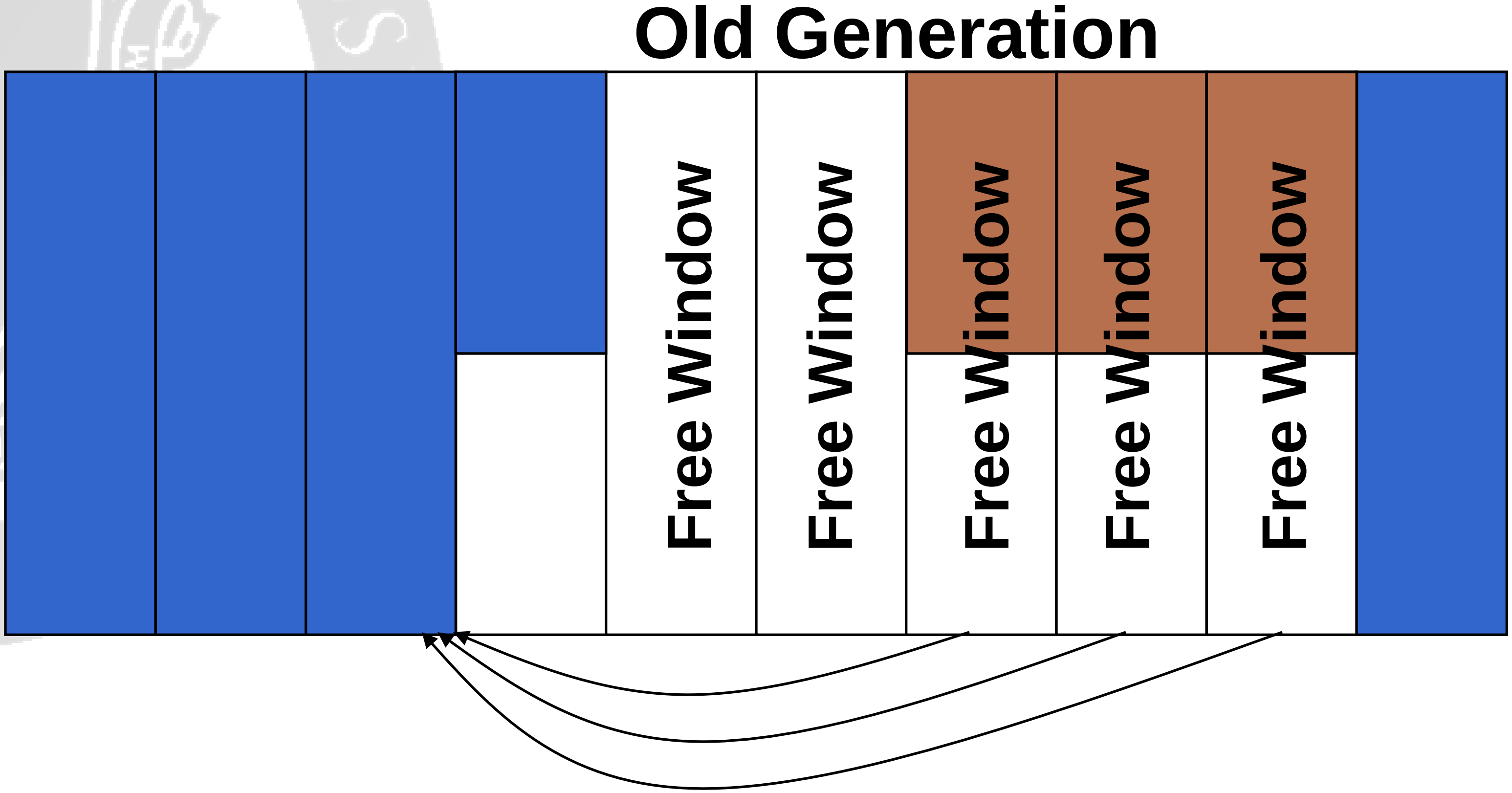
🟫 Reachable data □ Unreachable data 🟦 Copied data

MC^2 Overview



🟫 Reachable data □ Unreachable data 🟦 Copied data

MC^2 Overview



🟫 Reachable data □ Unreachable data 🟦 Copied data