Performance Through Parallelism: Java HotSpot™ GC Improvements

John Coomes, Tony Printezis

Sun Microsystems, Inc.
http://java.sun.com

TS-1168

Our Goal—Convert This:

Q: Why does GC take so long?
<grumble, grumble...>

A: It starts at the roots and...

Into This:

Q: How does GC go so fast?

A: It uses parallelism and concurrency and...

Agenda

Motivation

Terms and Taxonomy

Parallel Scavenge

Concurrent Mark Sweep

Parallel Compaction

What's Cooking in the Lab

Summary

Agenda

Motivation

Terms and Taxonomy

Parallel Scavenge

Concurrent Mark Sweep

Parallel Compaction

What’s Cooking in the Lab

Summary

Why Parallel?

Hardware Trends

- Multiple…
  - Chips per box
  - Cores per chip
  - Threads per core
- Sun, AMD, IBM, Intel
  - Shipping or developing multi-threaded, multi-core products
- 64-bit addressing
  - Allows huge heaps

Why Parallel?

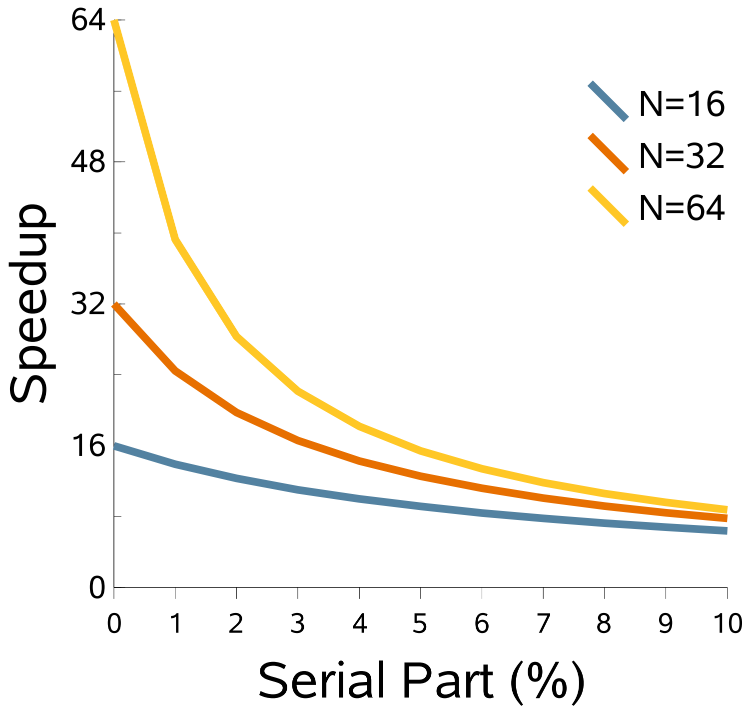
Java Programming Language

- Built-in support
- Threads allow parallelism
- java.util.concurrent APIs enable parallelism
  - Lock-free data structures
  - Locking primitives
  - Available in Java platform 5 and later

Why Parallel?

Amdahl's Law

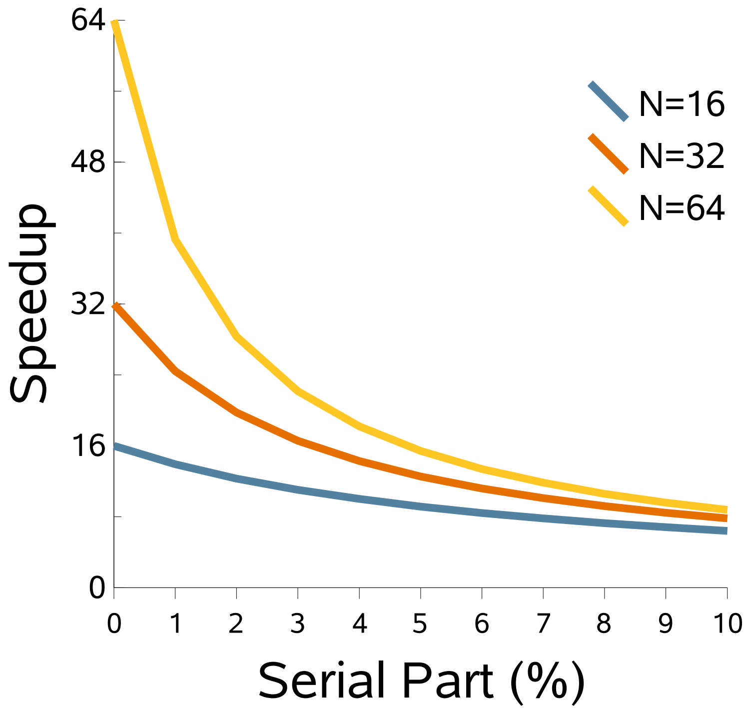
- Speedup =
  1/(Serial Part + (Parallel Part / N))



Why Parallel?

Amdahl's Law

- Many threads create garbage
- Just one thread to clean it up?
  - GC would become a serial bottleneck



Agenda

Motivation

Terms and Taxonomy

Parallel Scavenge

Concurrent Mark Sweep

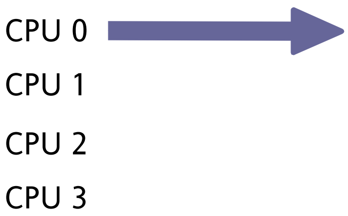
Parallel Compaction

What's Cooking in the Lab

Summary

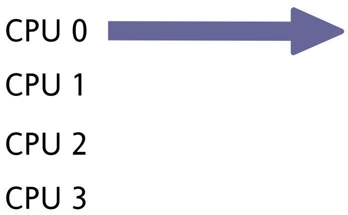
Terms

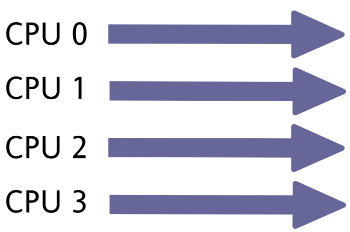
- Serial
  - One thing happens at a time



Terms

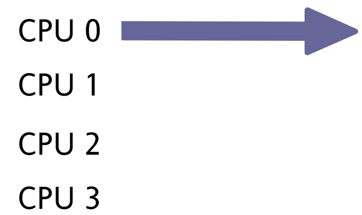
- Serial
  - One thing happens at a time
- Parallel
  - Multiple things happen at once
  - Single task
    - Split into parts
    - Executed simultaneously



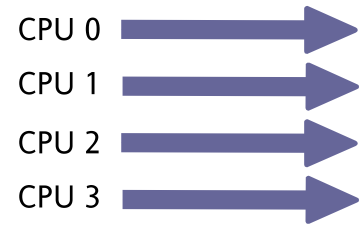


Terms

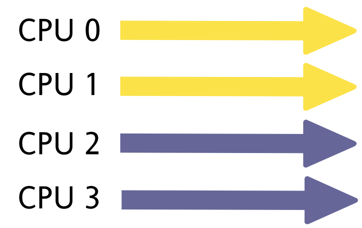
- Serial
  - One thing happens at a time



- Parallel
  - Multiple things happen at once
  - Single task
    - Split into parts
    - Executed simultaneously



- Concurrent
  - Multiple things happen at once
  - Multiple tasks
    - Different purposes
    - Execute simultaneously
    - Here: Java technology tasks vs. GC tasks

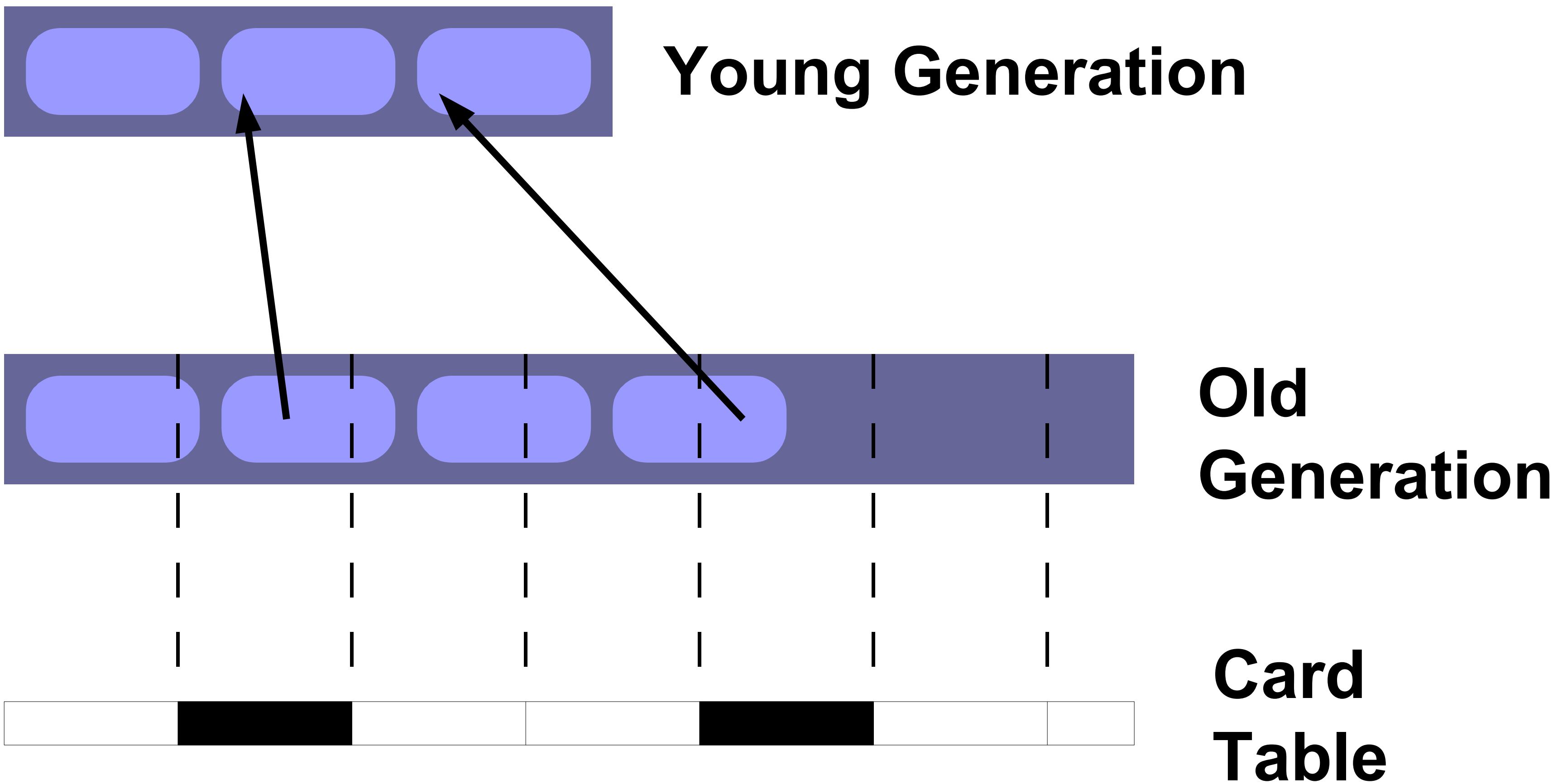


Taxonomy

- Generational GC
  - Observation: most objects die young
  - Segregate objects
    - New objects—allocated in the ‘young generation’
    - Old objects—promoted to the ‘old generation’
  - Collect young generation more frequently
    - Use algorithm optimized for ‘mostly dead space’
  - Pros:
    - Efficient—work a little, reclaim a lot
    - Most pauses are shorter—scan only part of the heap
  - Cons:
    - Some extra bookkeeping
    - Eventually must collect entire heap

Generations in HotSpot

Old-to-Young References



Taxonomy

- Stop-the-World GC
- Typical cycle:
  - Stop all Java technology threads
  - Do GC work
  - Restart all Java technology threads
- Pros:
  - Simpler—heap is frozen, objects not changing
- Cons:
  - Some applications sensitive to pause times

Taxonomy

- Concurrent GC
- Typical cycle:
  - Start GC
  - Java technology threads continue to run
    - During most or all of GC cycle
  - Finish GC
- Pros:
  - Pause times are short (or non-existent)
- Cons:
  - Must take extra care—objects are changing
  - Some overhead—performance, heap size

Agenda

Motivation

Terms and Taxonomy

Parallel Scavenge

Concurrent Mark Sweep

Parallel Compaction

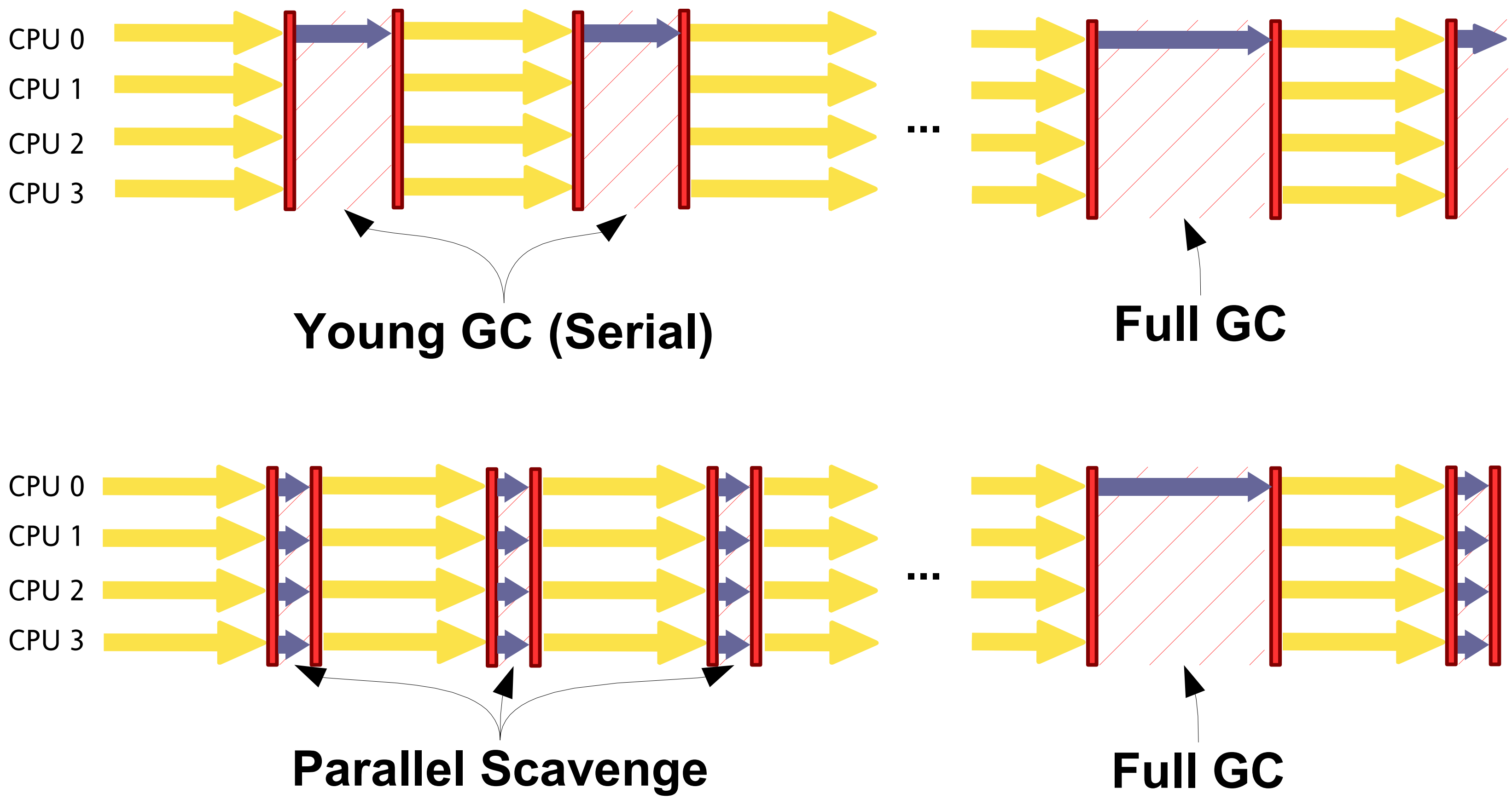
What's Cooking in the Lab

Summary

Parallel Scavenge

- Generational, stop-the-world, parallel
- Collects young generation only
- Available since JDK™ 1.4.2 software
  - Experimental in JDK 1.4.1 software
- First parallel GC in Java HotSpot performance engine
  - For many applications
  - Majority of GC time is spent on young generation
- -XX:+UseParallelGC

Typical GC Pattern

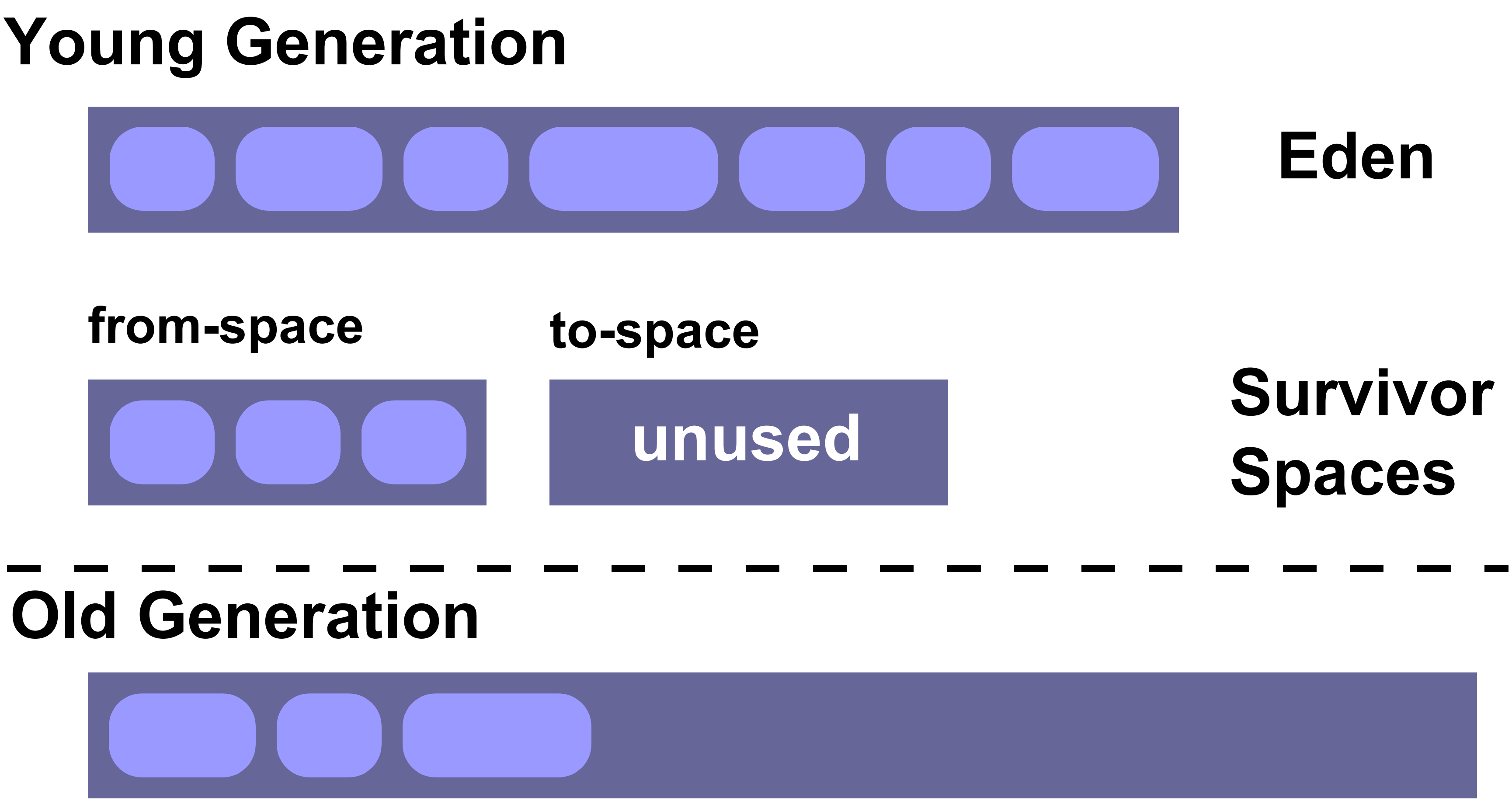


Parallel Scavenge

Basic Algorithm

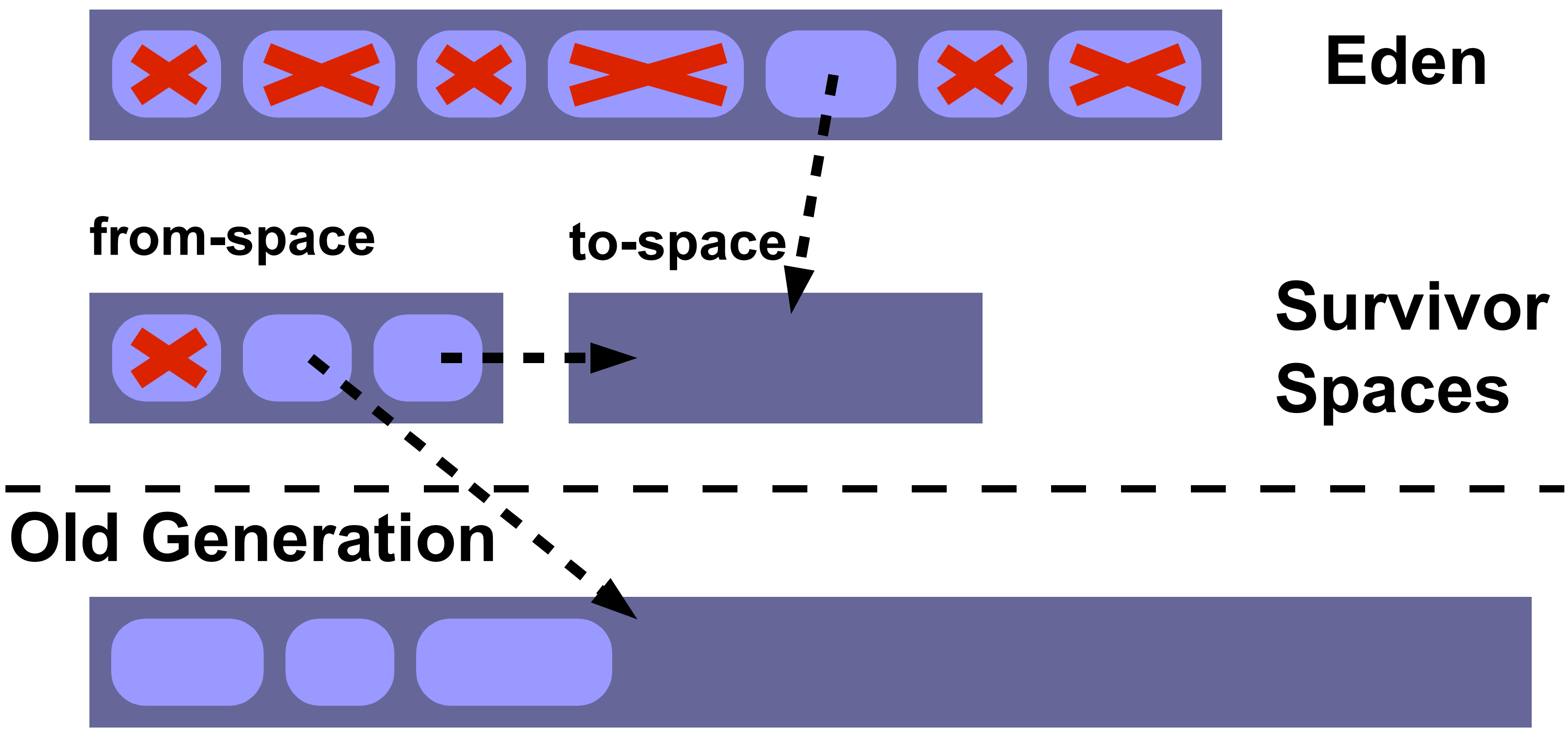
- Divide root set among GC threads
- Trace reachable objects in young generation
  - Atomic instruction (CAS) used to claim an object
- As objects are claimed, they are copied
  - Young-ish objects copied to to-space
  - Old-ish objects promoted into old generation
  - Use per-thread buffers in destination spaces
    - Fast, lock-free allocation
- At end, eden and from-space are empty
  - from-space and to-space switch roles

Scavenge Example—Before

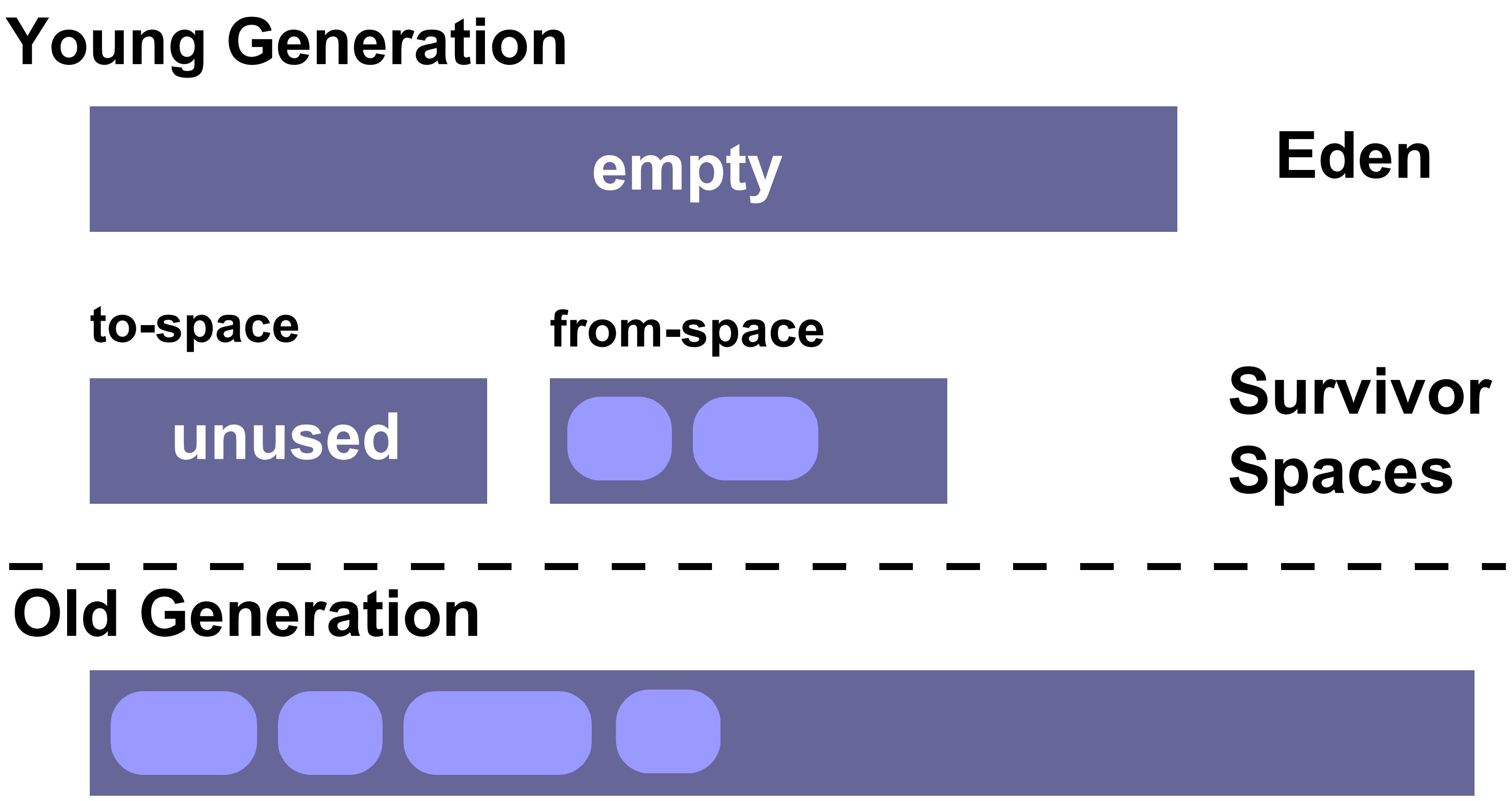


Scavenge Example—During

Young Generation



Scavenge Example—After



Agenda

- Motivation
- Terms and Taxonomy
- Parallel Scavenge
- Concurrent Mark Sweep
- Parallel Compaction
- What's Cooking in the Lab
- Summary

Concurrent Mark Sweep

- Generational, mostly concurrent, parallel, non-moving
  - Collects old and permanent generations only
  - Paired with Parallel Scavenge
    - (Serial scavenge on uniprocessor)
  - Most work done concurrently with Java technology threads
  - Available since JDK 1.4.2 software
    - Experimental in JDK 1.4.1 software
  - -XX:+UseConcMarkSweepGC

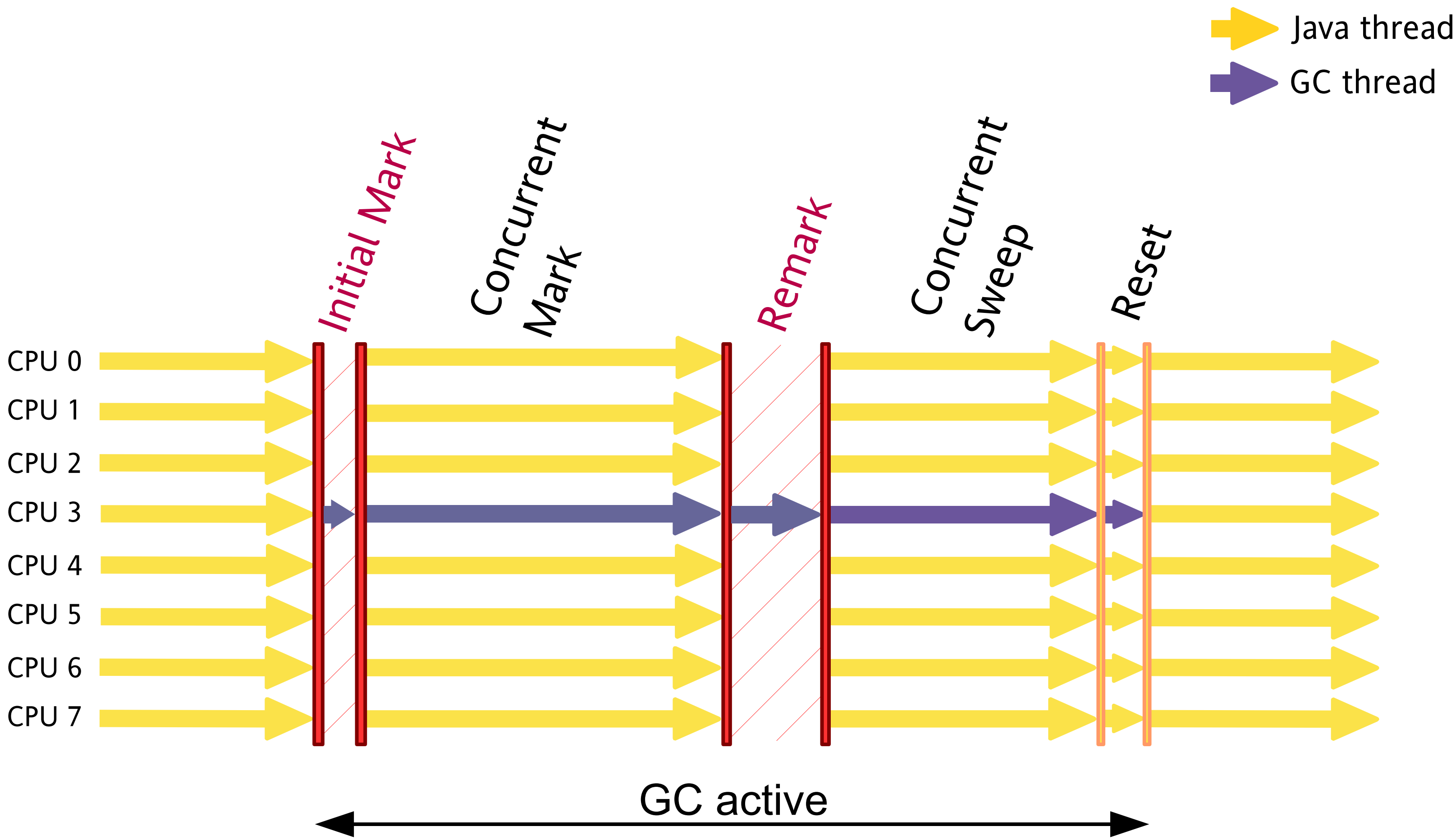
Concurrent Mark Sweep

- Non-moving
  - Once promoted into old generation, object does not move(*)
  - Free lists used for allocation
    - Somewhat slower than ‘pointer-bumping’
- Dealing with fragmentation
  - Track popular object sizes
  - Estimate future demand
  - Split or join free blocks to meet demand
- Serial Mark Sweep Compact used as fallback
  - (*) Will move old generation objects

Concurrent Mark Sweep

- Concurrent Mark Sweep Phases
  - Initial mark
    - Stop-the-world pause to mark from roots
    - Not a complete marking—only one level deep
  - Concurrent mark
    - Mark from the set of objects found during Initial Mark
  - Remark
    - Stop-the-world pause to complete marking cycle
    - Ensures a consistent view of the world
  - Concurrent Sweep
    - Reclaim dead space, adding it back onto free lists
  - Concurrent Reset

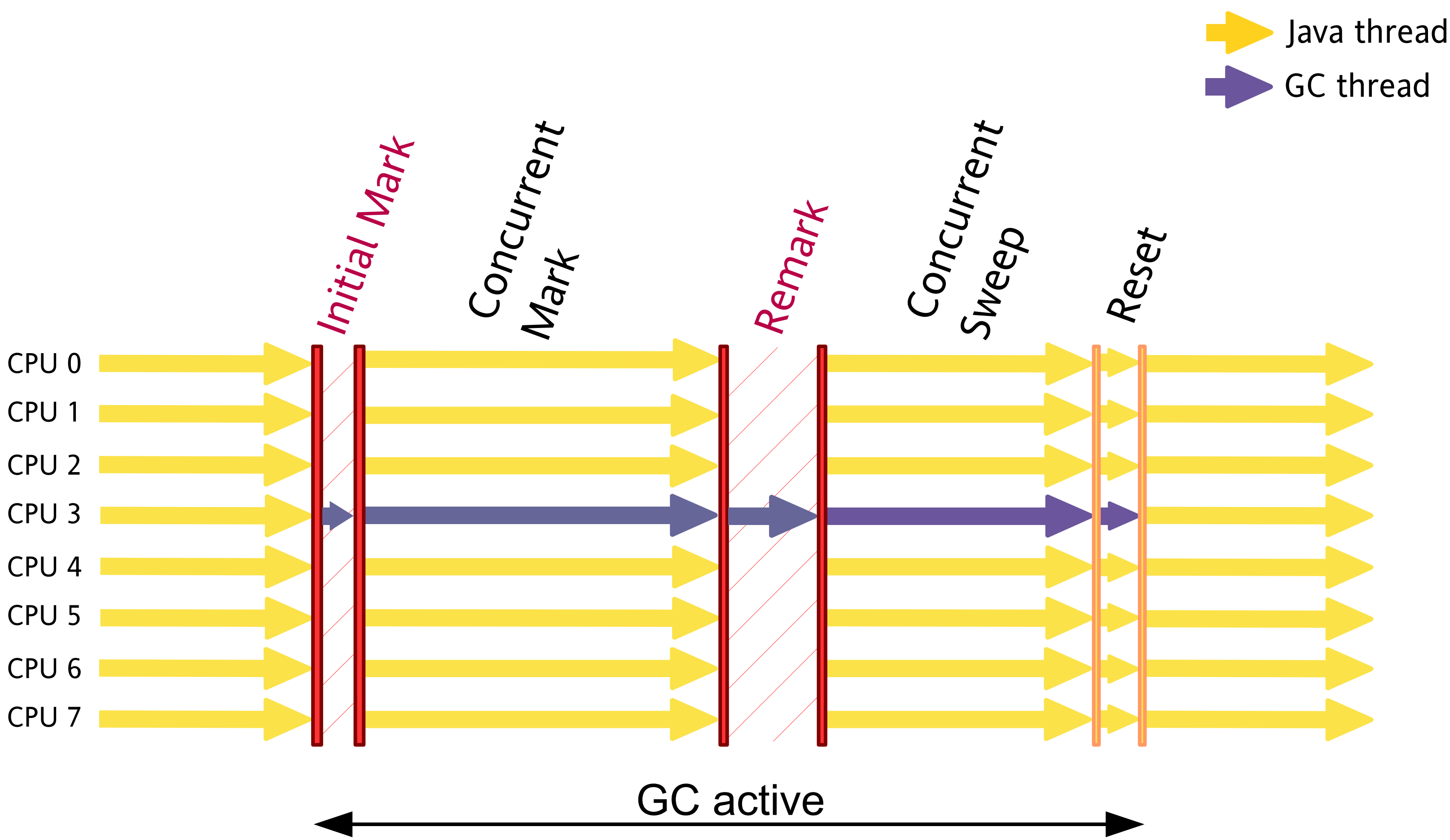
Concurrent Mark Sweep Phases



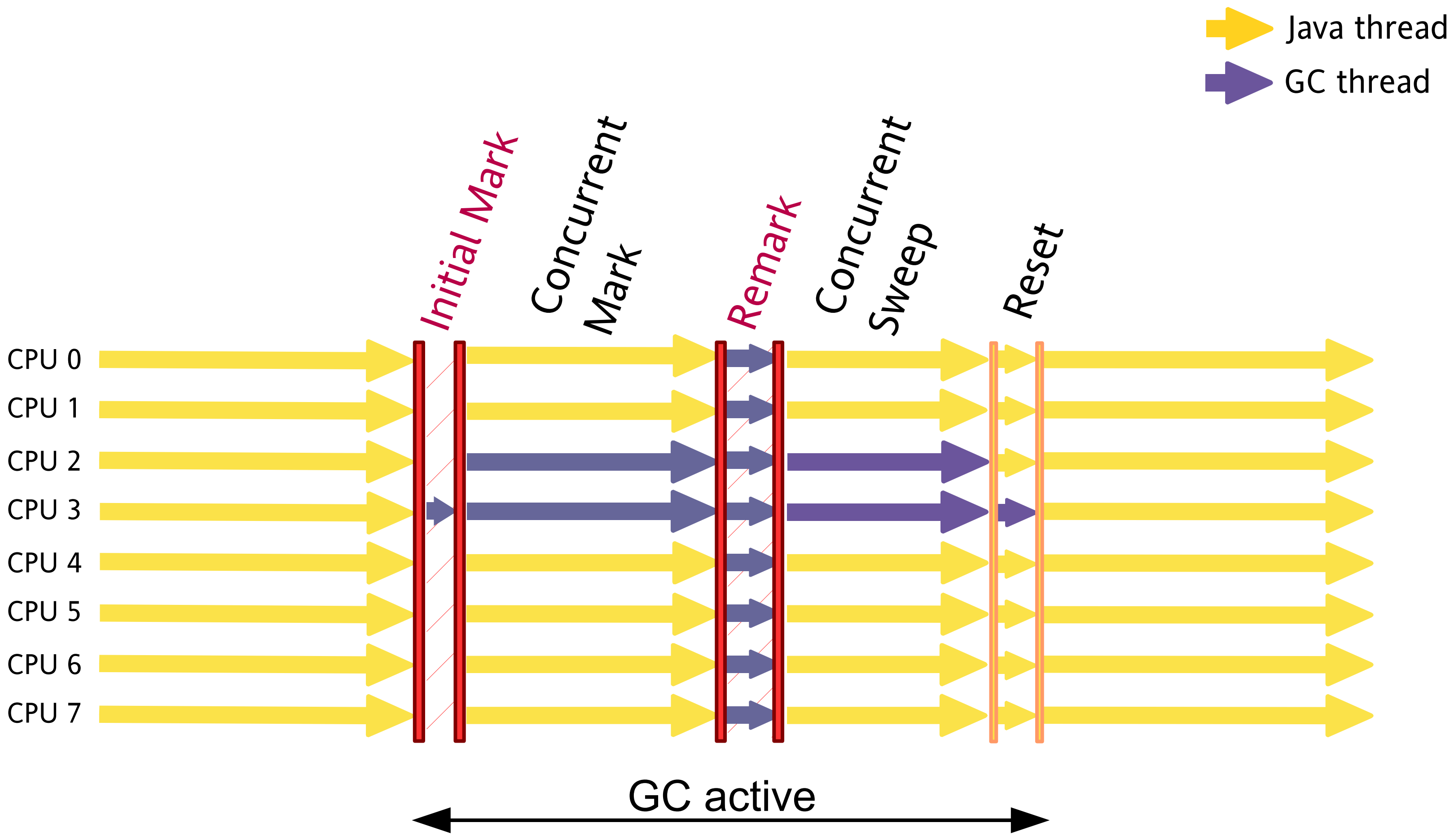
Concurrent Mark Sweep

- The need for Parallelism
  - Remark is typically the largest pause
    - Often larger than Young GC pauses
    - Parallel remark available since Java 5 platform
  - Single marking thread
    - Can keep up with ~4–8 cpus, usually not more
    - Parallel concurrent mark available in Java 6 platform
  - Single sweeping thread
    - Less of a bottleneck than marking
    - Parallel concurrent sweep coming soon

Concurrent Mark Sweep Phases



Concurrent Mark Sweep Phases



DEMO

Concurrent Mark Sweep

Agenda

Motivation

Terms and Taxonomy

Parallel Scavenge

Concurrent Mark Sweep

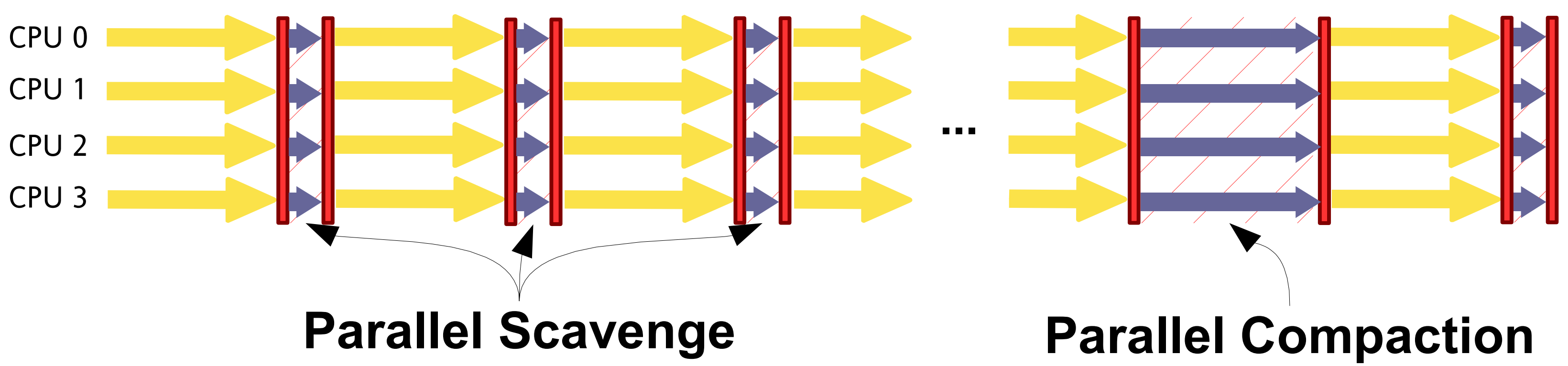
Parallel Compaction

What's Cooking in the Lab

Summary

Parallel Compaction

- Stop-the-world, parallel, sliding compaction
  - New! Available in Java EE 5 update 6
  - Full GC—collects entire heap
  - Paired with Parallel Scavenge
  - -XX:+UseParallelOldGC



Parallel Compaction

- Three phases: marking, summary, compaction
- Heap divided into fixed-sized regions
  - Marking records information about each region
  - Used later to enable compaction in parallel
- Sliding compaction—preserves object order
  - Possible cache benefits

Marking Phase

- Divide root set among GC threads
- Trace all live objects, in parallel
  - Objects claimed atomically
  - Liveness recorded in an external bitmap
- Once object is claimed, update per-region data
  - Add object size to region total
  - Extra bookkeeping if object extends onto other regions

Summary Phase

- Compute the "dense prefix"
  - Block of very dense regions on the left
    - Nearly all objects are live
    - Remainder are not reclaimed (dead wood)
  - Per-region liveness data guides selection
  - Find the prefix with the best reclamation ratio
    - Space reclaimed/data copied
- Compute destination of each region
  - Where the first live byte in the region will go

Summary Phase (Cont.)

- Additional data saved for each region
  - Source region
  - Destination count
- Summary phase currently done serially
  - Can be parallelized
  - Not as important for performance
    - Operates on regions, not objects

Compaction Phase

- Fill regions, in parallel
- First identify available regions
  - Empty regions
  - Regions that compact only into themselves
- Then threads claim available regions atomically
  - Once claimed, no other synchronization required
  - Fill the region, repeat

Compaction Phase (Cont.)

- Filling a region
  - Find the first byte destined for this region
  - Copy objects until region is full (or nothing left to copy)
  - Do some bookkeeping for source regions
- Easy!

Compaction Phase (Cont.)

- Filling a region
  - Find the first byte destined for this region
    - Start with source region
    - May have to skip over some objects
      - Consult bitmap for liveness info
  - Copy objects until region is full (or nothing left to copy)
  - Do some bookkeeping for source regions

Compaction Phase (Cont.)

- Filling a region
  - Find the first byte destined for this region
  - Copy objects until region is full (or nothing left to copy)
    - Must find live objects and skip dead space
      - Consult bitmap
  - Do some bookkeeping for source regions

Compaction Phase (Cont.)

- Filling a region
  - Find the first byte destined for this region
  - Copy objects until region is full (or nothing left to copy)
    - Must find live objects and skip dead space
      - Consult bitmap
    - Must update interior references to point to new locations
      - Consult ...
  - Do some bookkeeping for source regions

Compaction Phase (Cont.)

- New location of object x
  - Start with destination of the region containing x
  - Add size of partial object extending onto the region
  - Add sizes of live objects that precede x in the region
    - Consult the bitmap :-)
- ```
  region(x).destination() +
  region(x).partial_obj_size() +
  bitmap.live_words_in_range(region_start, x)
  ```
  ```

Compaction Phase (Cont.)

- Filling a region
  - Find the first byte destined for this region
  - Copy objects until region is full (or nothing left to copy)
  - Do some bookkeeping for source regions
    - Decrement destination count
    - If count reaches 0, region can be filled

Compaction Phase (Cont.)

- Filling a region
  - Find the first byte destined for this region
  - Copy objects until region is full (or nothing left to copy)
  - Do some bookkeeping for source regions

Compaction Phase (Cont.)

- Filling a region
  - Find the first byte destined for this region
  - Copy objects until region is full (or nothing left to copy)
  - Do some bookkeeping for source regions
- Whew!

DEMO

Parallel Compaction

Agenda

Motivation
Terms and Taxonomy
Parallel Scavenge
Concurrent Mark Sweep
Parallel Compaction
What’s Cooking in the Lab
Summary

Parallel Successes

- Parallelism used successfully so far
  - Parallel Scavenge
  - Concurrent Mark Sweep
  - Parallel Compaction
- Naturally, we (and you!) want more…
  - Shorter, predictable pauses—less disruption
  - Without fragmentation problems

Evaluating a New Low-latency GC

- Generational
- Parallel
- Concurrent
- Predictable
  - Pauses managed to meet specified goals
- Compacting
  - Only part of the heap at a time
  - Keeps pauses small

Evaluating a New Low-latency GC

- Generational, without fixed generations
  - Single physical space
  - Divided into “regions”
  - Young and old regions can be intermixed
    - Absence of fixed boundary allows flexibility
- Marking phase
  - Concurrent and parallel (like CMS)
  - Calculates liveness info per region
  - Much shorter remark pause than CMS

Evaluating a New Low-latency GC

- Per-region liveness info
  - Used to identify empty and mostly-empty regions
  - Empty regions—reclaimed
  - Mostly-empty regions
    - Evacuated—live objects copied to other regions
    - Then reclaimed
- All collection through copying
  - Focus on mostly-empty regions
    - Maximizes GC efficiency
  - Both young and old regions collected

Evaluating a New Low-latency GC

- Predictability
  - Model GC costs
  - Predict and schedule GC activity
- Shorter pause times
  - Through tricks!

Agenda

Motivation
Terms and Taxonomy
Parallel Scavenge
Concurrent Mark Sweep
Parallel Compaction
What’s Cooking in the Lab
Summary

Summary

- Parallelism is pervasive
- Must be exploited for good performance
  - Throughput
  - Responsiveness
- Java HotSpot technology GC makes use of it today
  - In several different forms
- We can only expect more opportunities in the future

For More Information

Resources

- Ask the Experts
  - Today, 12:00 Noon—Pavilion, Booth 724
- BOF-0197
  - “Java HotSpot VM Q&A”
  - Thursday, 7:30pm
  - Moscone Center North Mtg Room 121/122
- GC Tuning Guides
  - http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html
  - http://java.sun.com/docs/hotspot/gc1.4.2/index.html
- Description of the GCs in the Java HotSpot VM
  - http://www.devx.com/Java/Article/21977/

Q&A

John Coomes, Tony Printezis

the POWER of JAVA™

Performance Through Parallelism: Java HotSpot™ GC Improvements

John Coomes, Tony Printezis

Sun Microsystems, Inc.
http://java.sun.com

TS-1168