HP-UX 11i Knowledge-on-Demand

HP technical Webcast series: software optimization

Technology for better business outcomes

© 2007 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice

HP-UX 11i v3 Knowledge-on-Demand

- Objective: Support software development partners and customers in achieving better business outcomes with HP-UX 11i.
- What HP is providing: a series of technical on-demand training Webcasts
  - Focused on helping developers increase performance through application optimization for HP-UX 11i v3 on HP Integrity servers
  - Access to HP for follow-up questions
  - Available at www.hp.com/go/knowledgeondemand

hp invent

HP-UX 11i v3 Knowledge-on-Demand Webinars – planned curriculum

- Foundation Track
  - Module 1: How to upgrade to HP-UX 11i v3
  - Module 2: HP-UX open source resources
  - Module 3: Unified file cache
  - Module 4: Caliper
  - Module 5: NUMA Tuning: Getting the Most Out of Your Cellular Server by using NUMA
  - Module 6: The Mercury Library – Increasing Application Performance
  - Module 7: Software Transition Kit's (STK's) for HP-UX 11i v3
- Java Developers Track
  - Module 8: Java Memory Management - Internals and Performance
  - Module 9: HPjmeter – measure Java application performance on HP-UX 11i
  - Module 10: Solving Java performance problems
- C/C++ Developers Track
  - Module 11: pthreads enhancements in HP-UX 11i v3
  - Module 12: Kernel tracing & profiling tools (internal tools)
  - Module 13: Using compilers to get optimal performance
  - Module 14: HP Code Advisor: A Powerful New C/C++ Analysis Tool for HP-UX
  - Module 15: Montecito Hyper-Threading on HP-UX 11i v3

Additional Webinars published going forward!

hp invent

Related HP-UX 11i v3 resources

- All developers' resources
  - HP-UX 11i developers' content
    www.hp.com/go/hpuxdev
  - HP-UX 11i v3 news, functionality, product download and services resources
    www.hp.com/go/hpux11i
  - HP Integrity server ISV resources for DSPP members
    www.hp.com/go/dspp_integrity
  - HP Integrity server product information
    www.hp.com/go/integrity
- Software partner promotional opportunity
  - HP promotion for HP-UX 11i v3-ready software partner application
    www.hp.com/go/v3promotion

invent

Enjoy this Knowledge-on-Demand topic!

Thank you for taking time to learn about HP-UX 11i v3 and related technologies.

Please send comments on today's topic and/or requests for future topics to:

hpuxquestions@hp.com

hp invent

Java memory management: Internals and performance

An HP-UX 11i Knowledge-on-Demand software optimization Webcast

Technology for better business outcomes

© 2007 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice

Introducing today's speaker

- Noubar is a member of the Java Compilers, and Tools Lab in Cupertino California, he has worked on the Java Virtual Machine since late 1999, prior to Java, he was a member of the C/C++ Low Level Optimizer team. During the past several years, Noubar's main focus has been performance of key Java benchmarks as well as end user applications. When he is not working on improving the performance of the JVM in the Lab, Noubar is out in the field working directly with customer on identifying and resolving performance issues in their applications.

hp invent

Agenda

- Java and memory management overview
- Generational garbage collection
- Internals of supported GC policies on HP-UX
- GC policy defaults and how are they set
- Performance analysis and tuning

hp invent

Java and memory management overview

- Commonly known as garbage collection (GC)
  - Automatic memory management for applications written in Java
  - JVM service executing periodically
- Many types of garbage collectors
  - Reference counting
  - Stop-the-world mark-sweep, mark-sweep-compact
  - Incremental/concurrent
- Java memory management on HP-UX
  - Generational copying collectors

hp invent

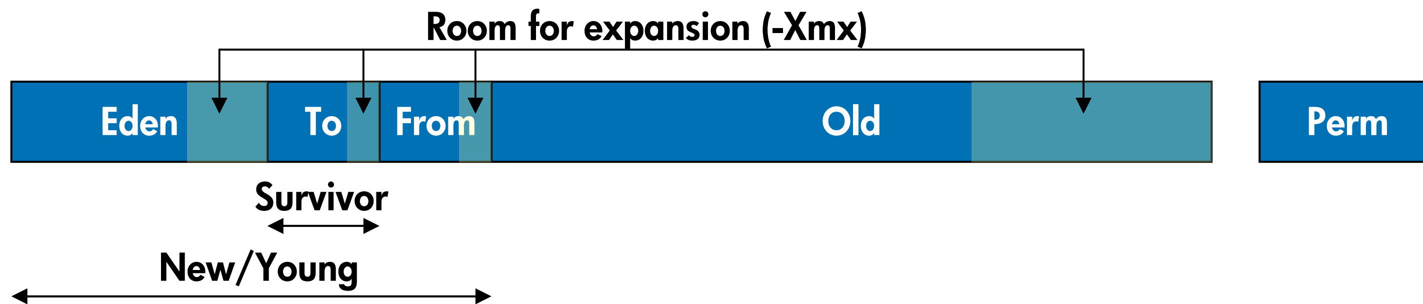
Agenda

- Java and memory management overview
- Generational garbage collection
- Internals of supported GC policies on HP-UX
- GC policy defaults and how are they set
- Performance analysis and tuning

Generational GC basics

The heap

- Java heap organized into multiple object pools (generations)
  - Cost of GC is proportional to the number of objects
  - Most objects live for a short time, reduce GC cost for common case



New + Old = initial heap size (-Xms)

New + Old + expansion = maximum heap size (-Xmx)

Eden + To + From = new/young generation size (-Xmn)

Perm (-XX:PermSize=n)

hp invent

Generational GC basics

Basic controls

- In addition to –Xmx, -Xms and -Xmn
- Two more high-level controls
  - XX:SurvivorRatio
    - The size of **ONE** survivor space compared to the size of the Eden (SurvivorRatio = Eden/From)
  - XX:MaxTenuringThreshold
    - The age of a surviving object in terms of the number of times it survived a minor collection

hp invent

Generational GC basics

Collection cost

- Two types of collections with two different costs
  - Minor collection: Inexpensive collection of the new/young generation triggered by the generation filling up
  - Major collection: Triggered by the old/perm generation becoming full
    - Expensive because it requires scanning/processing of **ALL** live objects

hp invent

Generational GC basics

How does it work?

- New objects are always allocated to Eden until it fills up

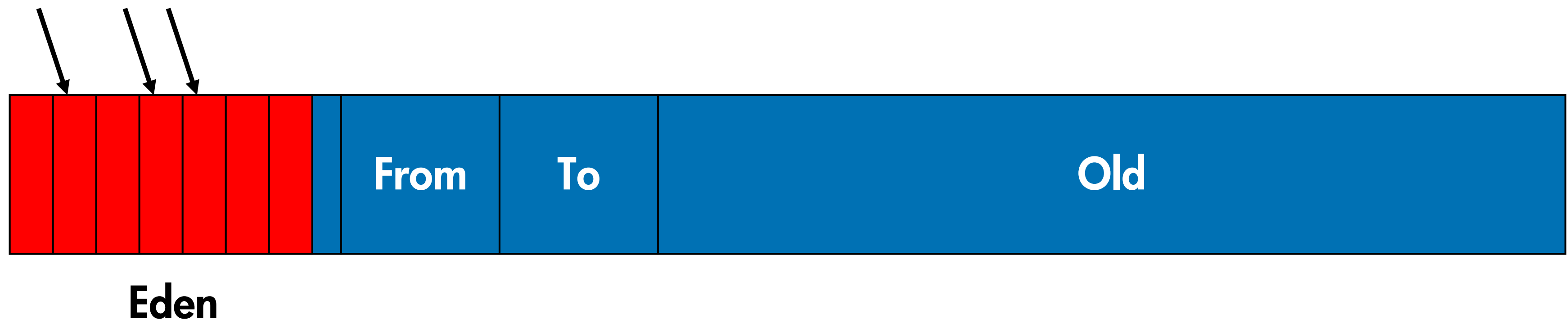
MaxTenuringThreshold=2

For the purpose of illustration

Generational GC basics

How does it work?

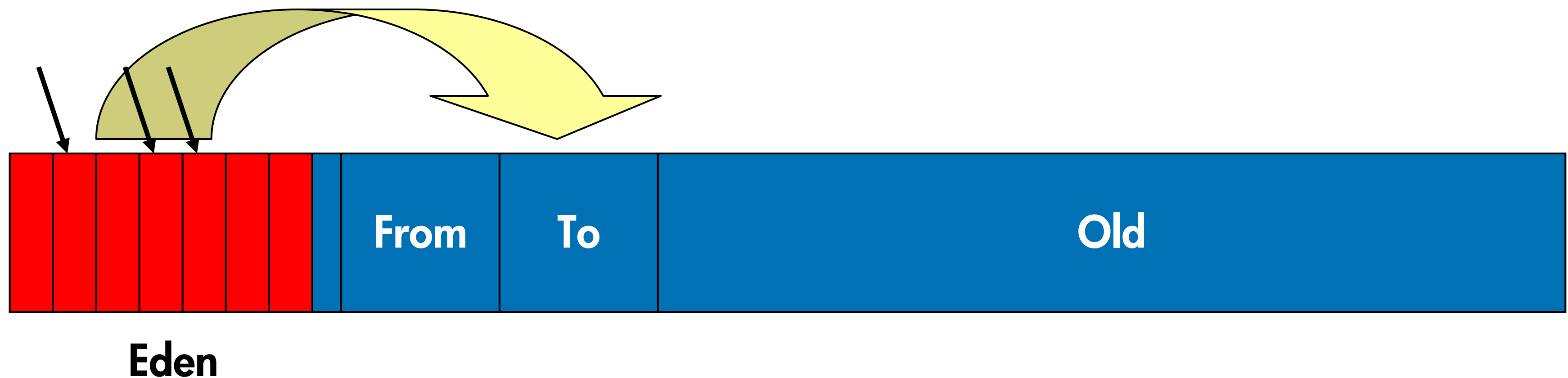
- Eden is full, (1) identify live objects



Generational GC basics

How does it work?

- Eden is full, (2) copy live objects to “To” space



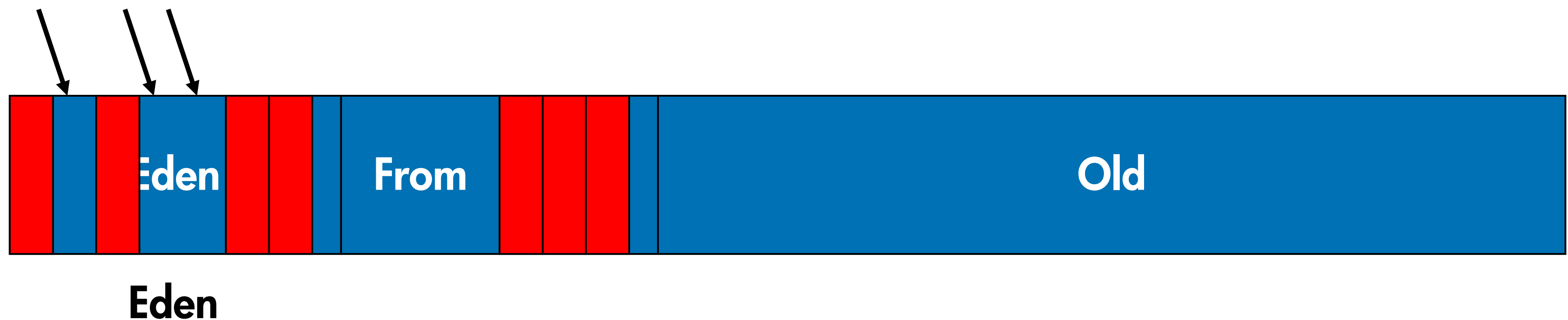
MaxTenuringThreshold=2

For the purpose of illustration

Generational GC basics

How does it work?

- Eden is full, (2) copy live objects to “To” space



Generational GC basics

How does it work?

- Eden is full, (3) scavenge Eden (minor collection)
- Eden is always completely empty after a minor collection

Eden

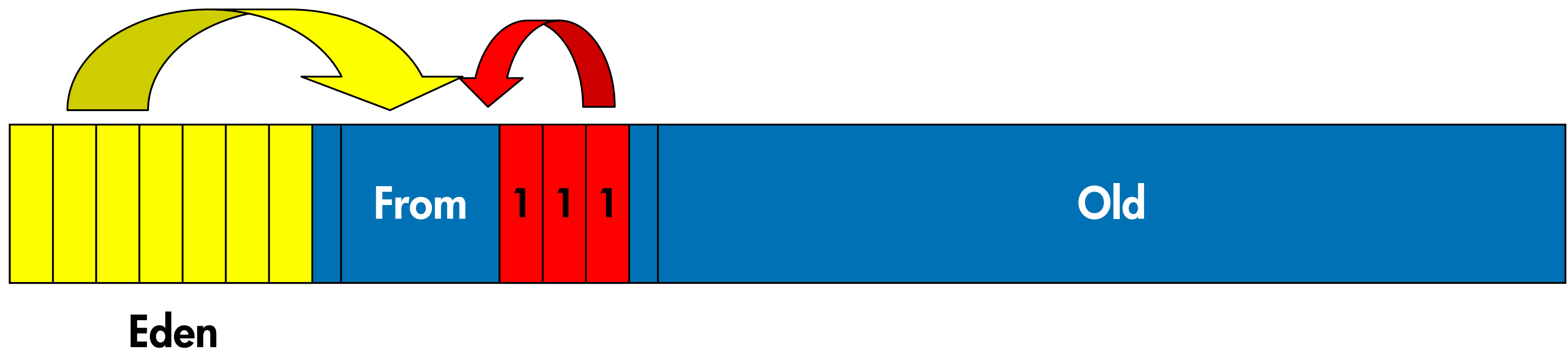
MaxTenuringThreshold=2

For the purpose of illustration

Generational GC basics

How does it work?

- (4) More objects are allocated, and Eden is full again



Generational GC basics

How does it work?

- (5) Copy live objects from both Eden and “To” space to the “From” space (note how the age of surviving objects is increased)

MaxTenuringThreshold=2

For the purpose of illustration

Generational GC basics

How does it work?

- (6) Scavenge Eden and “To” space and rename “To” and “From”

Eden

MaxTenuringThreshold=2

For the purpose of illustration

Generational GC basics

How does it work?

- (7) More objects are allocated in Eden and it is full again

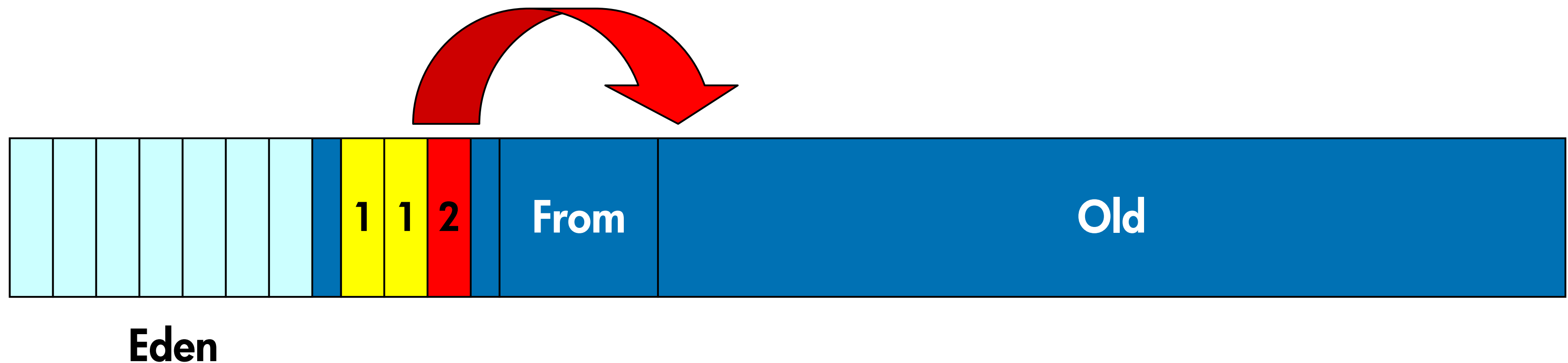
MaxTenuringThreshold=2

For the purpose of illustration

Generational GC basics

How does it work?

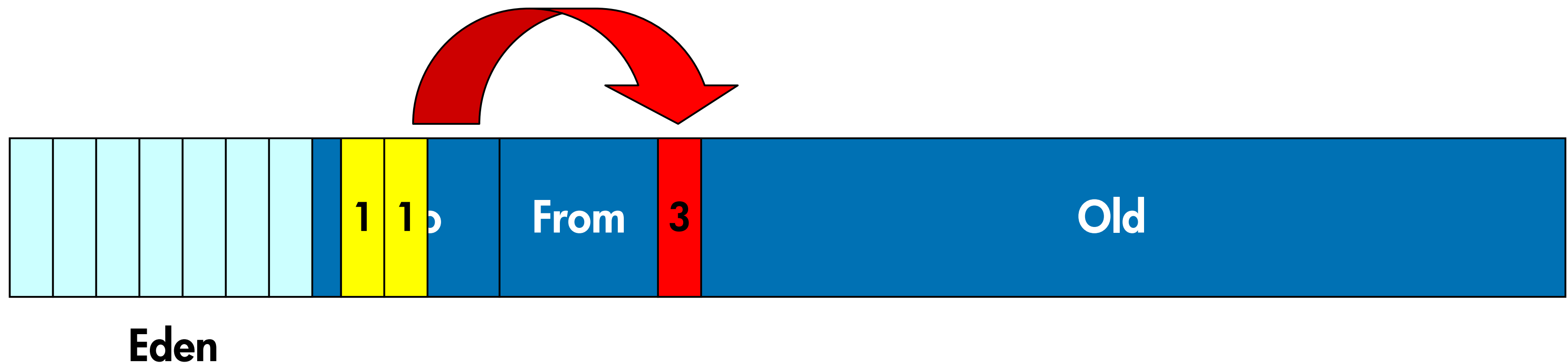
- (8) First promote (tenure) surviving object



Generational GC basics

How does it work?

- (9) First promote (tenure) surviving object



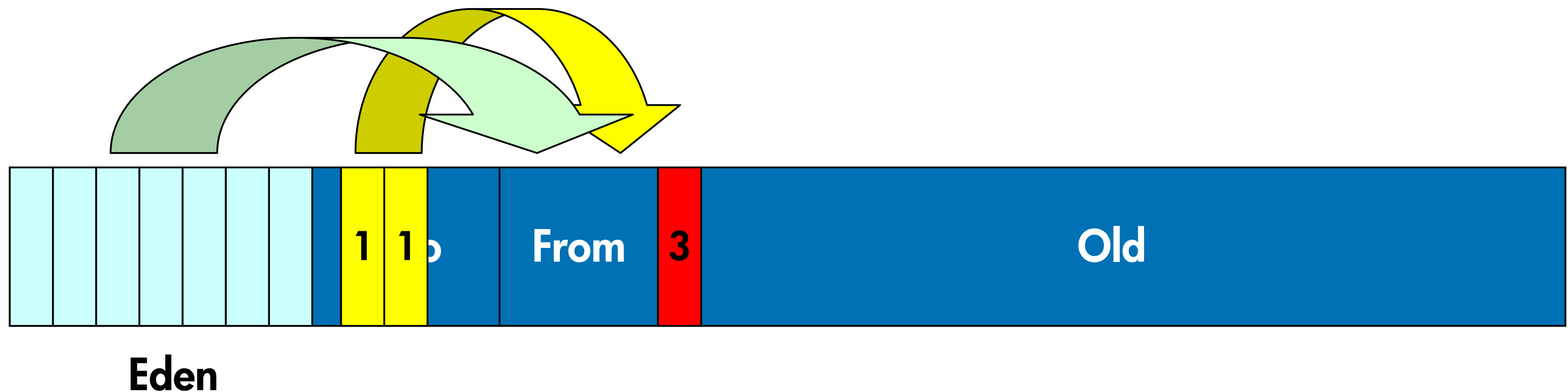
MaxTenuringThreshold=2

For the purpose of illustration

Generational GC basics

How does it work?

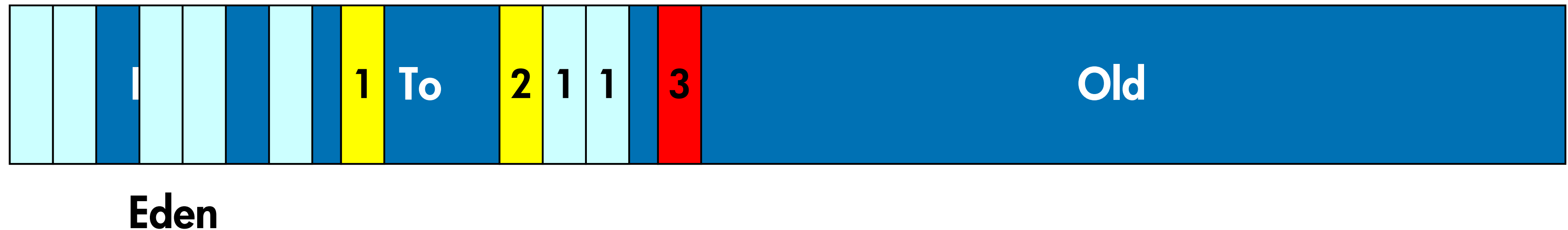
- (10) copy ALL surviving objects to the "From" space



Generational GC basics

How does it work?

- (10) copy ALL surviving objects to the “From” space



MaxTenuringThreshold=2

For the purpose of illustration

Generational GC basics

How does it work?

- (11) scavenge (remove all dead objects – minor collection)
- Rename "To" to "From"

Eden

MaxTenuringThreshold=2

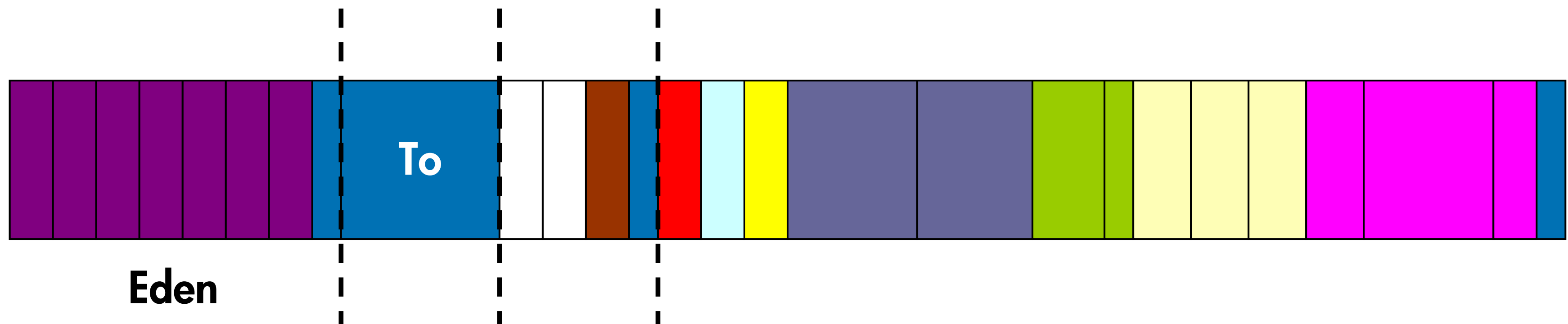
For the purpose of illustration

hp invent

Generational GC basics

How does it work?

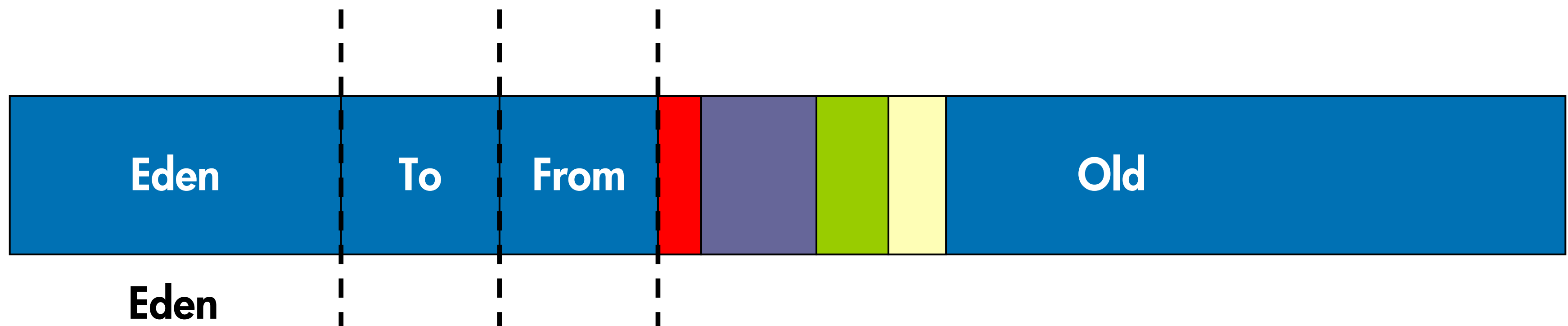
- After a while, the old generation becomes full, Eden becomes full, and one of the survivor spaces becomes full
- A full GC (a major collection is triggered)



Generational GC basics

How does it work?

- After a full GC, only long living objects remain in the heap
- Eden acts as a “filter” for short lived objects



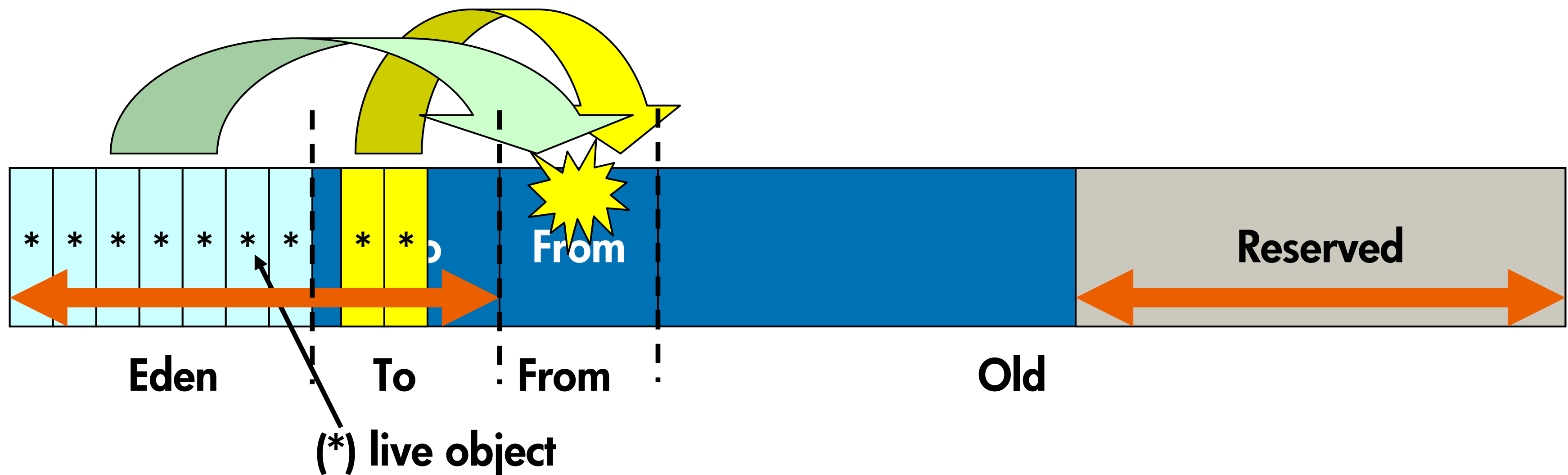
MaxTenuringThreshold=2

For the purpose of illustration

Generational GC basics

How does it work?

- Young generation guarantee
  - -XX:MaxLiveObjectEvacuationRatio=n%
  - -Xoptgc (sets MaxLiveObjectEvacuationRatio=0)



Agenda

- Java and memory management overview
- Generational garbage collection
- Internals of supported GC policies on HP-UX
- GC policy defaults and how are they set
- Performance analysis and tuning

hp invent

Internals of supported GC policies on HP-UX

How many? Why?

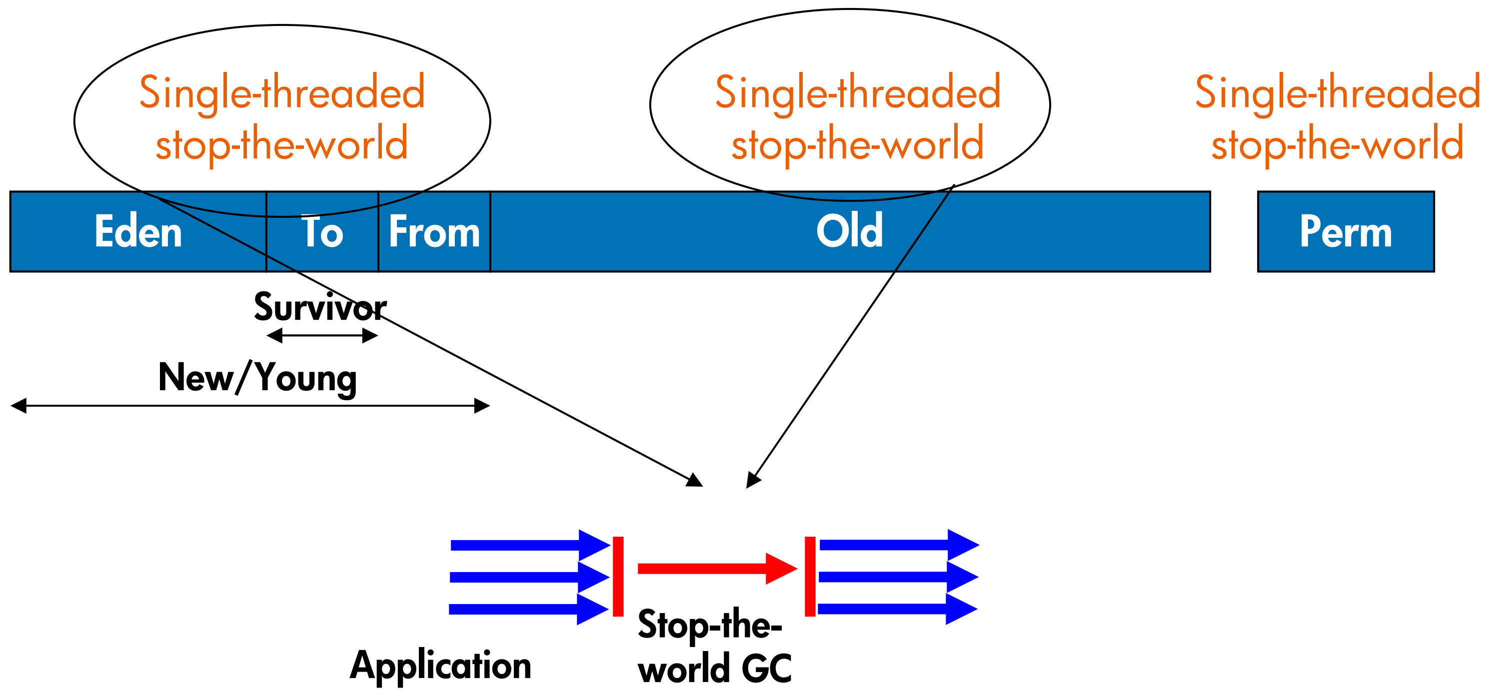
- One size does not always fit all
- Specialized solutions result in better performance
- Four major GC implementations
  - Serial stop-the-world GC policy
  - High-throughput parallel new generation GC policy
  - High-throughput parallel old generation GC policy
  - Low-pause concurrent mark sweep GC policy

hp invent

Serial GC policy

-XX:+UseSerialGC

- Single-threaded stop-the-world mark-sweep-compact for scavenge and full GC

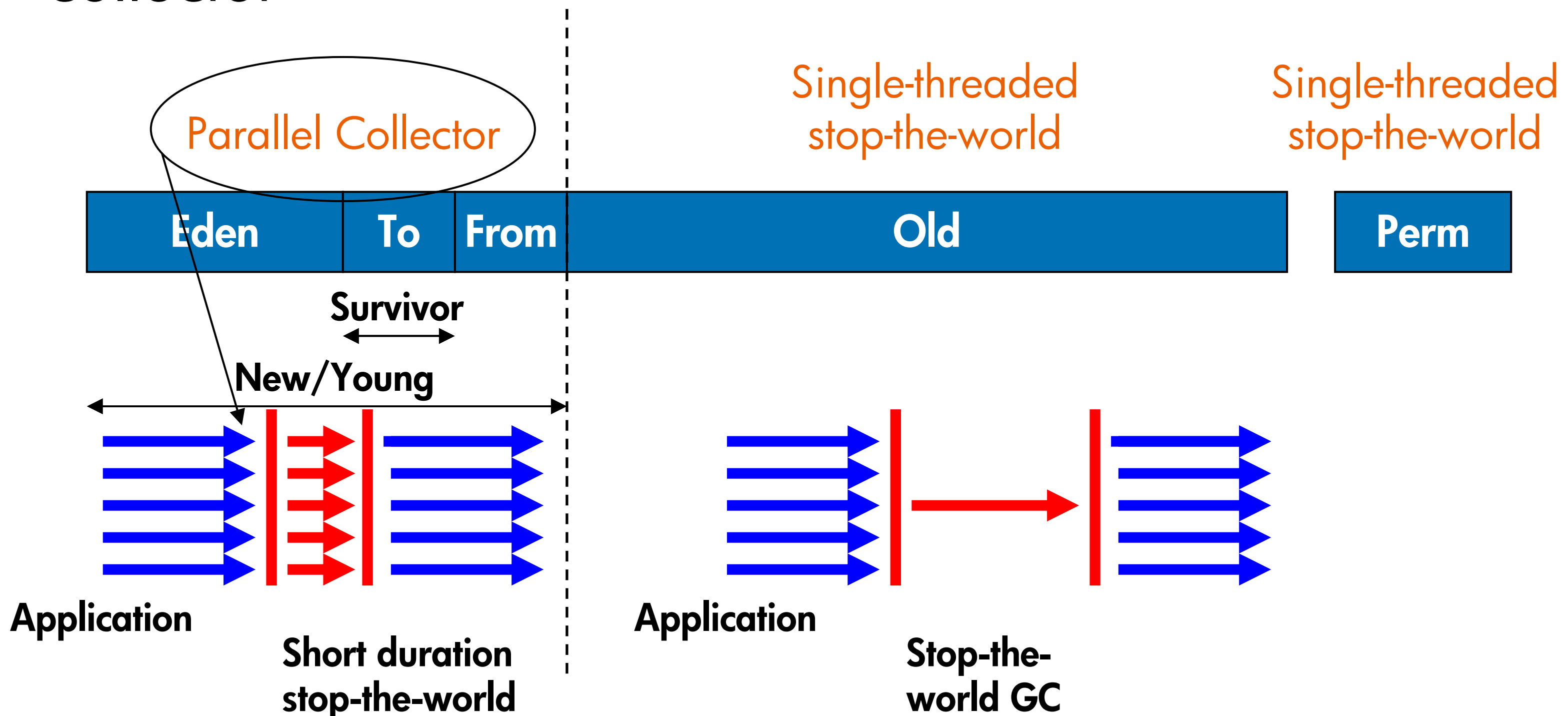


i n v e n t

Throughput/Parallel GC policy 1

-XX:+UseParNewGC

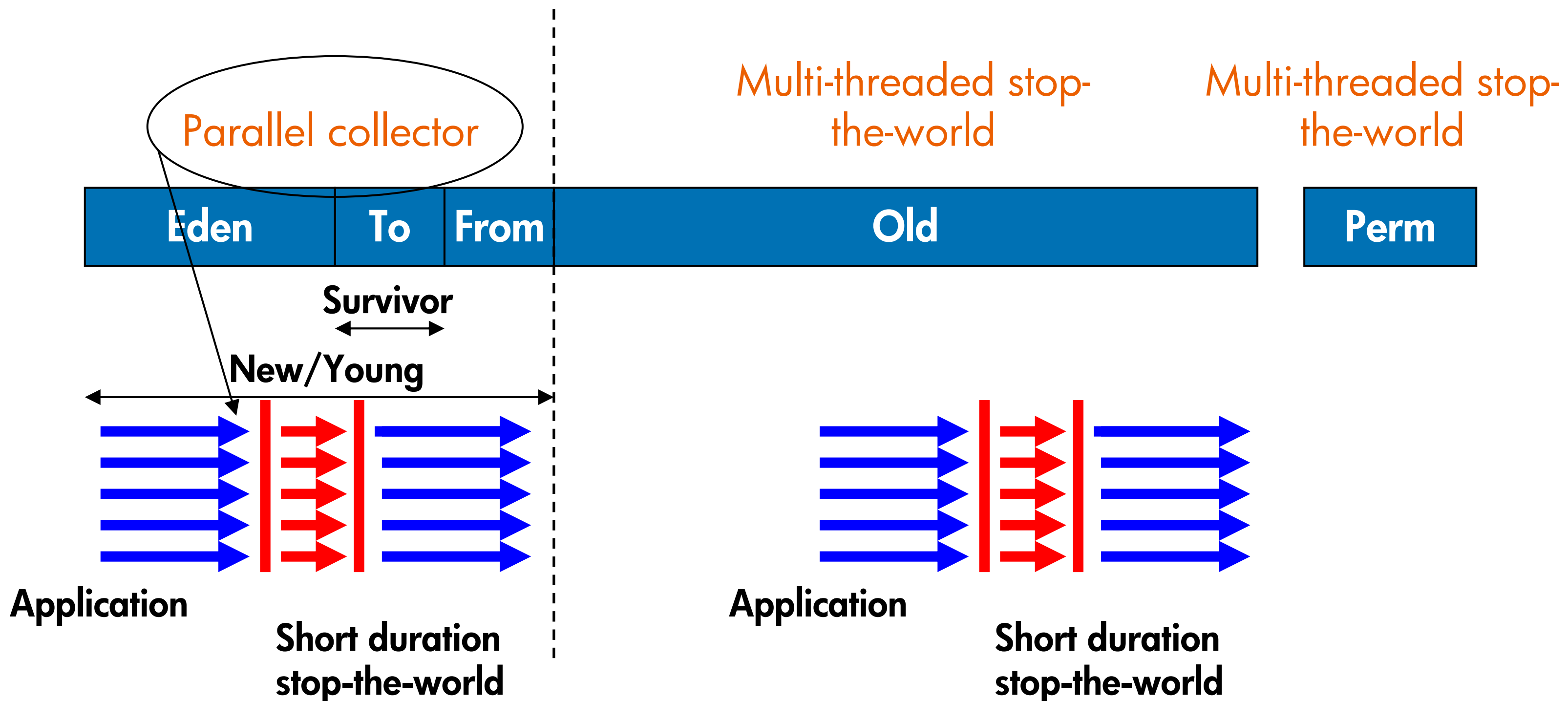
- Multi-threaded young generation collector and a single-threaded stop-the-world mark-sweep-compact old generation collector



Throughput/Parallel GC policy 2

-XX:+UseParallelOldGC

- Multi-threaded young and old generation collector

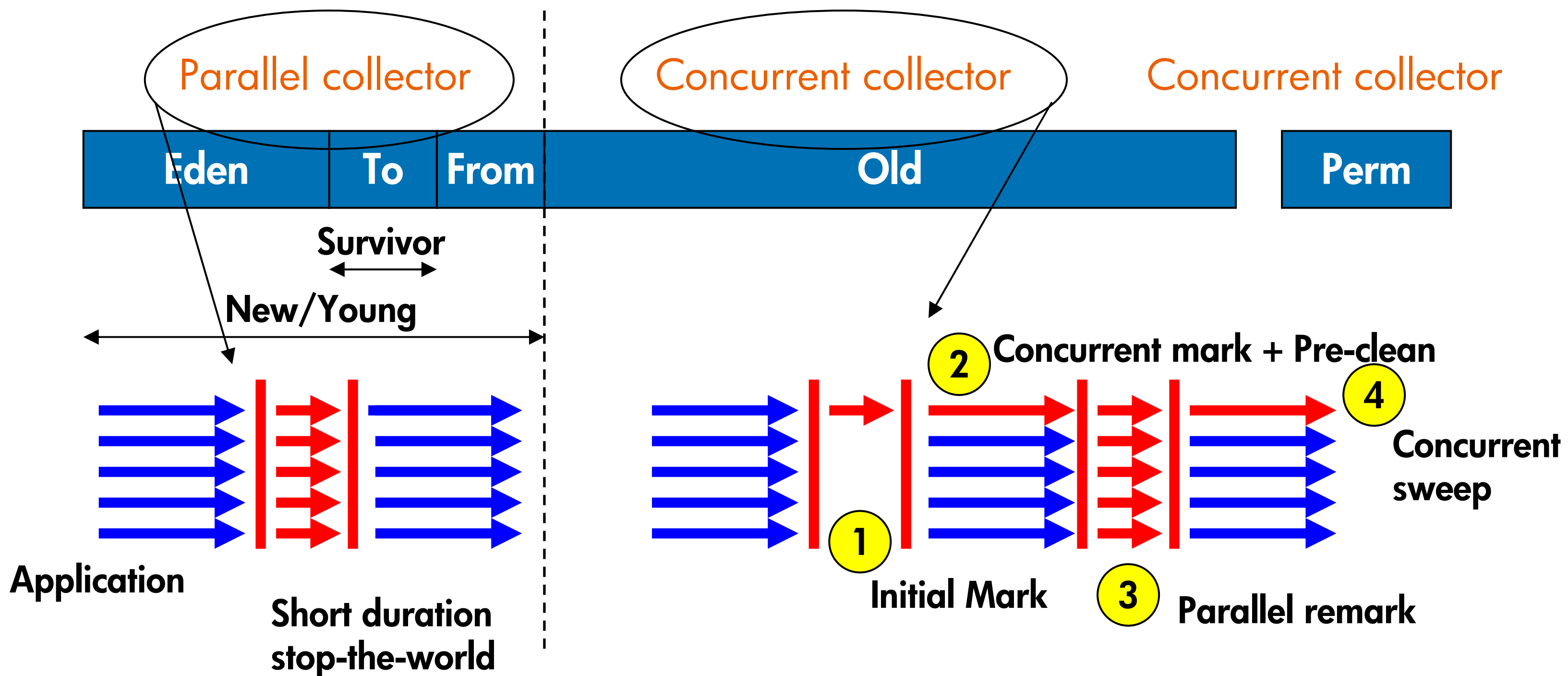


hp invent

Low-pause/CMS GC policy

-XX:+UseConcMarkSweepGC

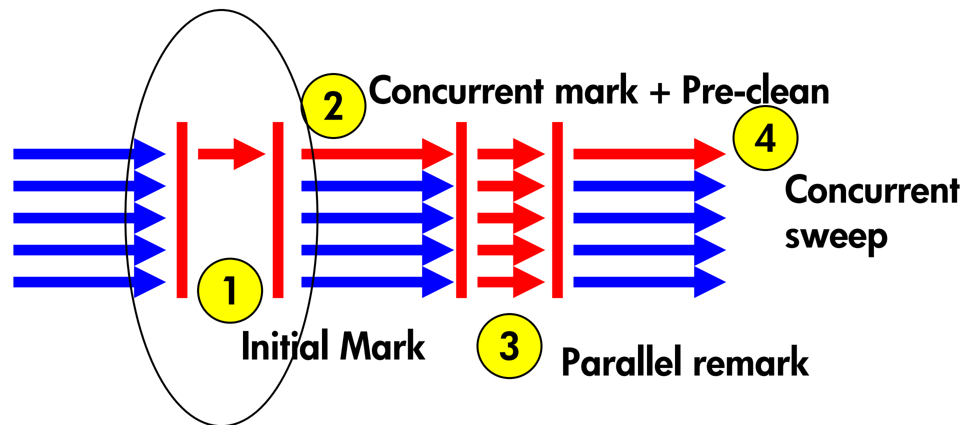
- Multi-threaded young generation collector and a low-pause "**mostly**" concurrent old generation collector



Low-pause/CMS GC policy

CMS cycle phase 1

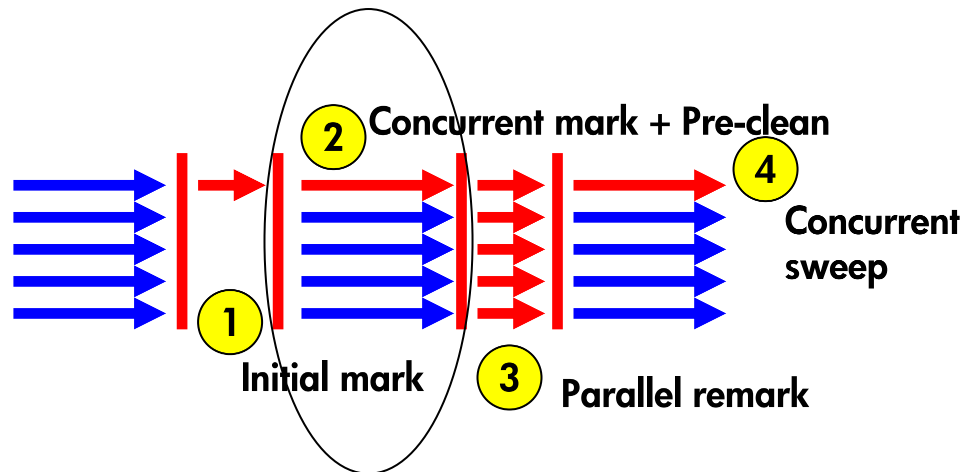
- Starts with a single threaded quick initial mark phase
- Initial mark identifies a subset of reachable objects
- Tools: Stop-the-world 1 (STW 1)



Low-pause/CMS GC policy

CMS cycle Phase 2

- Single threaded concurrent mark phase
- Identifies all live objects reachable from the set of live objects identified in Phase 1 (initial mark)
- Pre-cleaning scans all objects updated while concurrently marking (application still running)

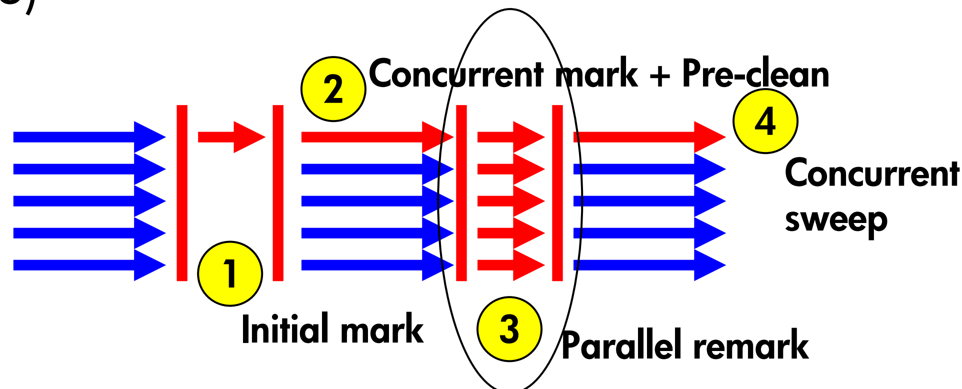


hp invent

Low-pause/CMS GC policy

CMS cycle Phase 3

- Parallel remark phase
- Not all live objects are guaranteed to be marked during Phase 2 (application still running)
- Phase 3 stops the world and does a quick remark to identify live objects missed during concurrent mark phase (Tools: STW 2)
  - Examines only objects modified during Phase 2 (uses updated card table)

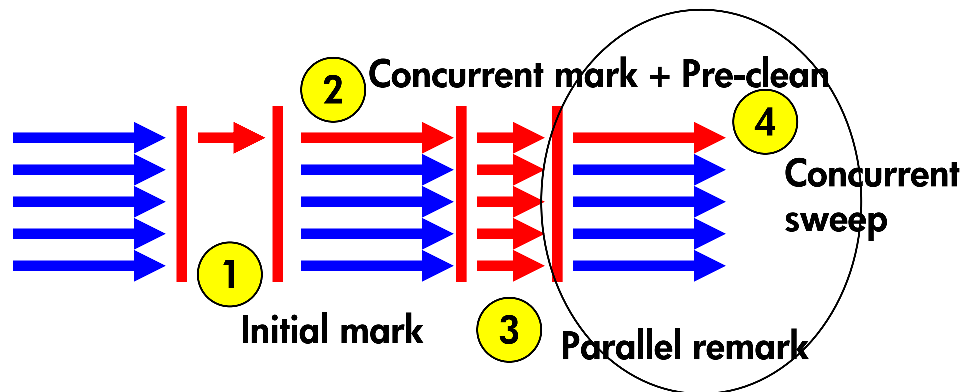


hp invent

Low-pause/CMS GC policy

CMS cycle Phase 4

- Single threaded concurrent sweep (clean-up) of all dead objects (all objects that were not marked live)



hp invent

Low-pause/CMS GC policy

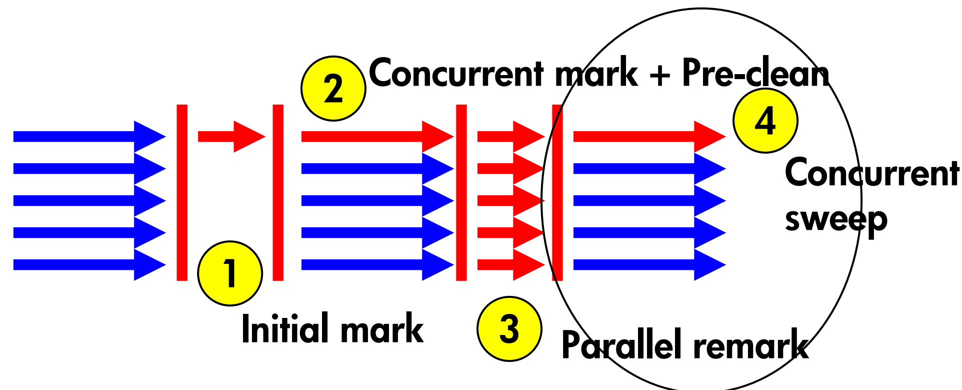
CMS phases

invent

Low-pause/CMS GC policy

New in JDK 6

- Multiple CMS threads
  - **-XX:ParallelCMSThreads=n**
- New mode for System.gc()
  - **-XX:+ExplicitGCInvokesConcurrent** normal old generation scavenge, no compaction
  - **-XX:-ExplicitGCInvokesConcurrent** single threaded full GC with compaction (default mode)



Low-pause/CMS GC policy

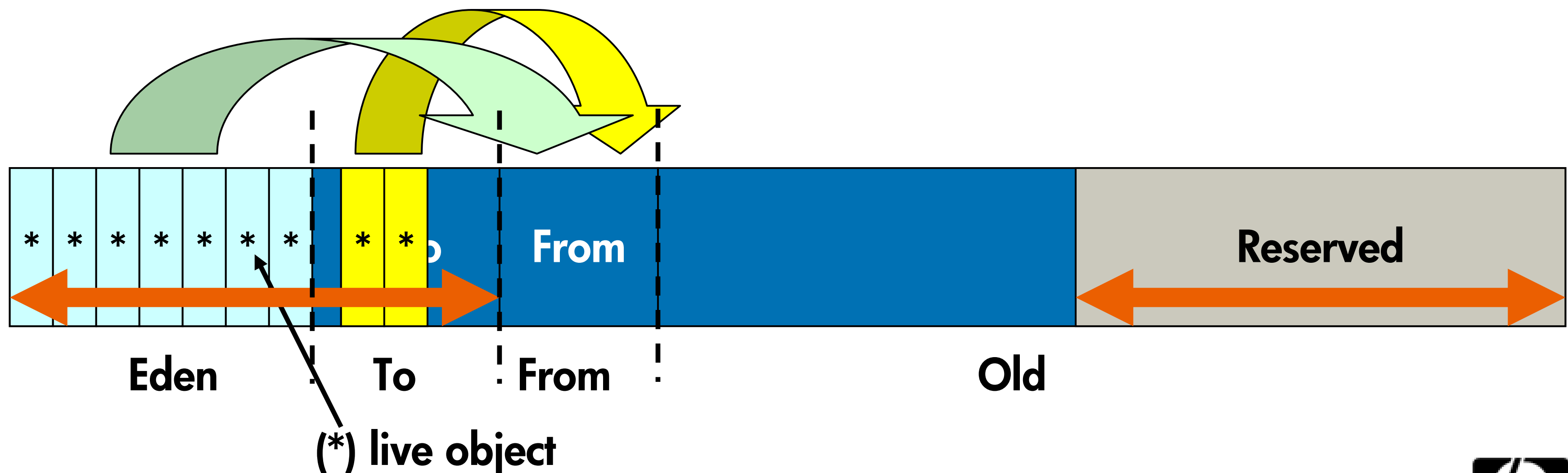
New in JDK 6

- Prior to JDK 6 the CMS collector promoted objects that survive a minor collection right away
  - Explicit **MaxTenuringThreshold** ignored (always 1)
  - Survivor spaces set artificially small to force early promotion (**SurvivorRatio=1024**)
  - Frequent concurrent collections
- Survivor spaces are reactivated in JDK 6

Low-pause/CMS GC policy

Young generation guarantee handling

- If **-Xoptgc** or **–XX:MaxLiveObjectEvacuationRatio=n** not specified
  – Try a last ditch minor collection first
    - The hope is that some objects might have died
  – Do a full GC if minor collection not sufficient



Agenda

- Java and memory management overview
- Generational garbage collection
- Internals of supported GC policies on HP-UX
- GC policy defaults and how are they set
- Performance analysis and tuning

hp invent

GC defaults

No GC policy/heap sizing specified - ergonomics

- If no CG parameters are specified
  - The initial heap size is automatically set to 1/64 of total available memory (capped at 1GB)
  - The maximum heap size is automatically set to ¼ of available memory (capped at 1GB)
  - Parallel new GC policy selected for machines with two or more processors
  - Adaptive size policy active
    - New generation (and survivor spaces) starts small, adapts to load
    - Adaptive sizing of PLAB and TLABS

hp invent

GC defaults

Throughput/parallel new/old GC policy defaults

- If the number of parallel GC threads is not explicitly specified, it is set as follows:
  - Use 5/8th of a thread for every processor after the first eight
    - For a system with 72 CPUS:
      8 + (72 -8) * (5/8) = 48 parallel GC threads
- If **XX:ParallelGCThreads**=1 is set :
  - JVM switches policy to serial GC

hp invent

GC defaults
Low-pause CMS policy defaults

- If –**XX:+UseConcMarkSweepGC** is specified
  - A parallel garbage collection policy is used for the new generation (same algorithm in **–XX:+UseParNewGC**)
  - **ParallelGCThreads** count used if set to greater than 1
  - If not set, parallel GC threads is computed according to the following formula:
    - Use 5/8th of a thread for every processor after the first eight
      - For a system with 72 CPUS:
        8 + (72 - 8) * (5/8) = 48 parallel GC threads
  - If **ParallelGCThreads** is explicitly set to one, use serial collector for the new generation

hp invent

GC defaults

Low-pause CMS policy defaults

- If **–XX:+UseCocMarkSweepGC** is specified
  - Automatic young generation sizing to achieve short pauses
  - Automatic old generation sizing to be at least 3X young generation size
  - Early/fast promotion of young objects that survive a scavenge
- Automatic settings will be disturbed if user explicitly specifies:
  - **Xmx, Xms, XX:MaxTenuringThreshold, or XX:SurvivorRatio**

hp invent

Agenda

- Java and memory management overview
- Generational garbage collection
- Internals of supported GC policies on HP-UX
- GC policy defaults and how are they set
- Performance analysis and tuning

hp invent

GC performance

General guidelines – correct heap sizing

- Minimize costly full GC as a result of incorrect sizing of Eden/To/From
  - Size to avoid premature promotion of "short lived" objects into old space (unnecessary, expensive full GC)
  - Keep Tenuring Threshold high
  - Remember that a full GC will also result in "short lived" objects being copied into old space (New/To/From are completely empty after a full GC)

hp invent

GC performance

General guidelines – avoid costly full GC

- Minimize costly full GC by never calling System.gc() or Runtime.gc()
- In third party code, system GC can be disabled:
  - **-XX:+DisableExplicitGC**
- RMI forces frequent full GC as well:
  - -Dsun.rmi.dgc.server.gcInterval=60000 (1 minute)
  - -Dsun.rmi.dgc.client.gcInterval=60000 (1 minute)

hp invent

GC performance

General guidelines – policy selection

- Match selected GC policy to type of application
  - Serial GC: Small application running on a small system
  - Parallel GC: Large throughput-oriented application running on a system with more than two processors
  - Concurrent GC: Application where responsiveness is of utmost importance (Telco – SIP server, etc.)
- Running with Montecito Hyper Threading enabled
  - Set the number of GC threads equal to "physical" cores used by the JVM
    - **-XX:ParallelGCThreads=n**
  - Enable the OS to optimize GC threads scheduling
    - **-XX:-BindGCTaskThreadsToCPUs**

hp invent

GC performance

General guidelines – tools HPjmeter

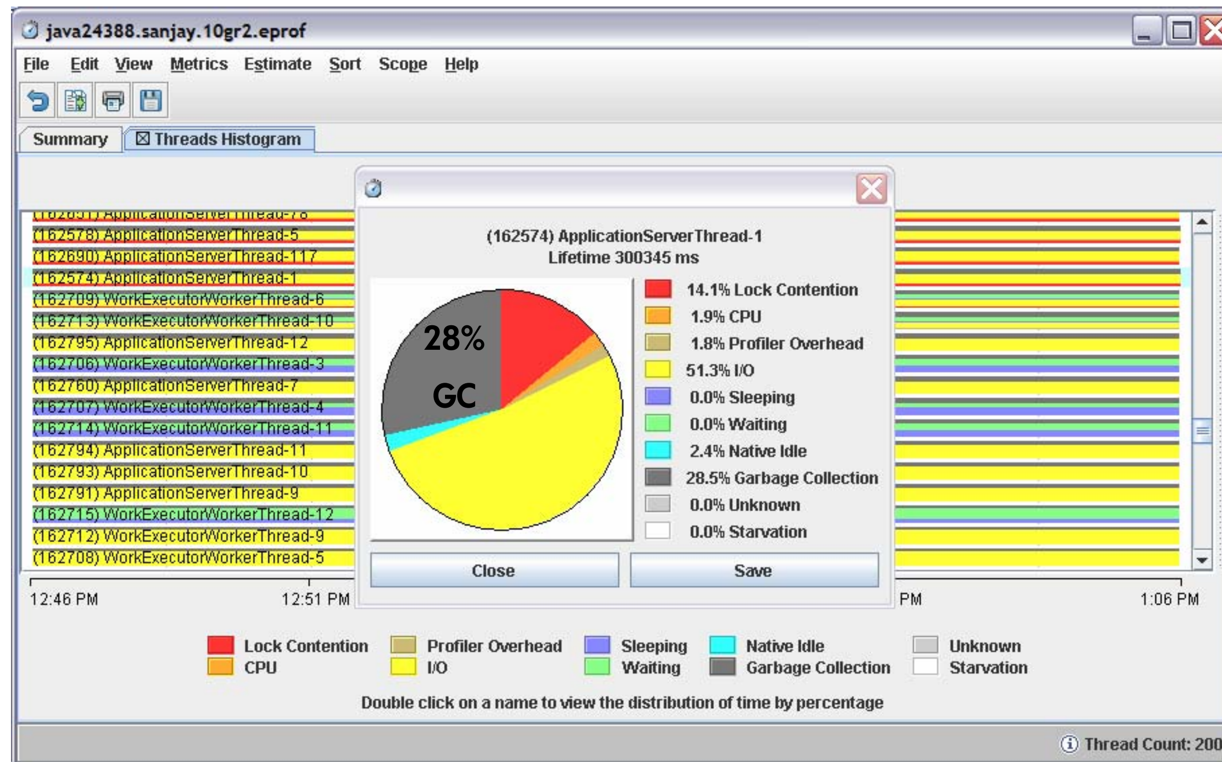
- Not sure about your application's memory requirements and allocation patterns over time?
  - Use the tools available on HP-UX
    - What is the application doing? Is GC responsible for the application running too slow?
    - Identify, analyze, and resolve GC related problems quickly
      - Is the heap sized correctly?
      - Discover heavy allocation sites, which module, which thread, which method in the application?

hp invent

GC performance

Tools – HPjmeter

- What is the application doing? Is GC responsible for the application running too slow?



GC performance

Tools – HPjmeter

- Discover heavy allocation sites and much more …

hp invent

Summary

- Be sure to understand how the different GC policies work before using any of them
- Understand the defaults for each policy
- Use the correct GC policy for your workload and environment
- Follow the tuning recommendations
- Use the tools to gain deeper understanding of how your application uses memory

hp invent