Towards High Performance Processing in Modern Java-based Control Systems

Marek Misiowiec
Wojciech Buczak, Mark Buttner
CERN
ICalepcs 2011

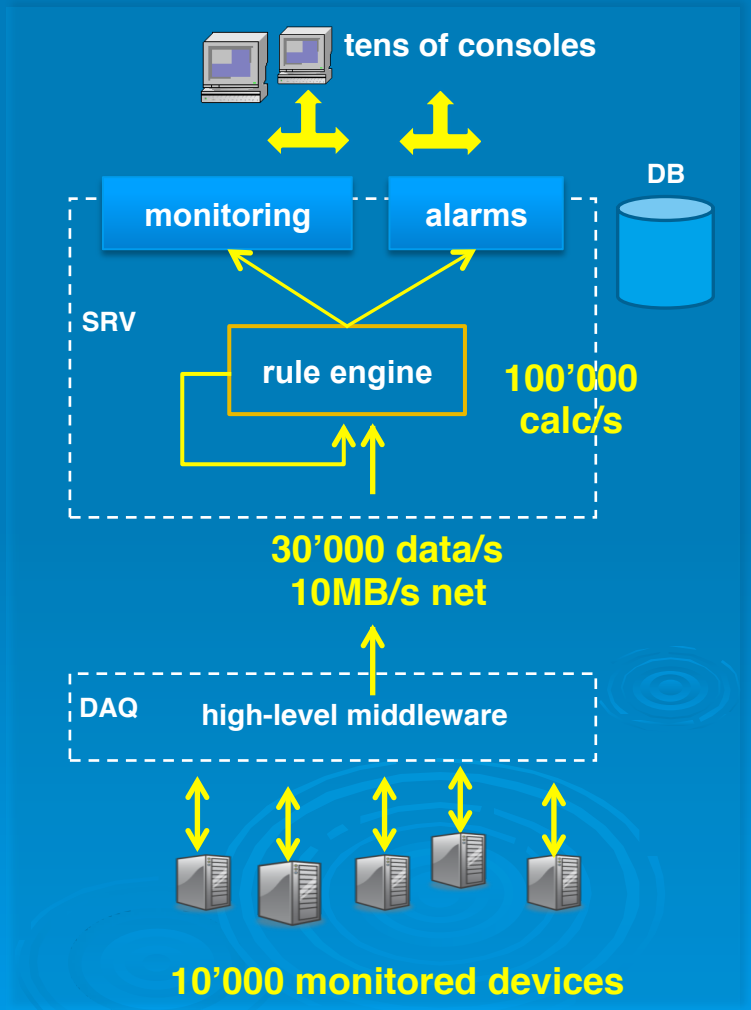
Performance with soft real time

Distributed system - Monitoring & Alarms at CERN

- collect data from over 10’000 devices
- heterogenous environment

Performance in middle-tier

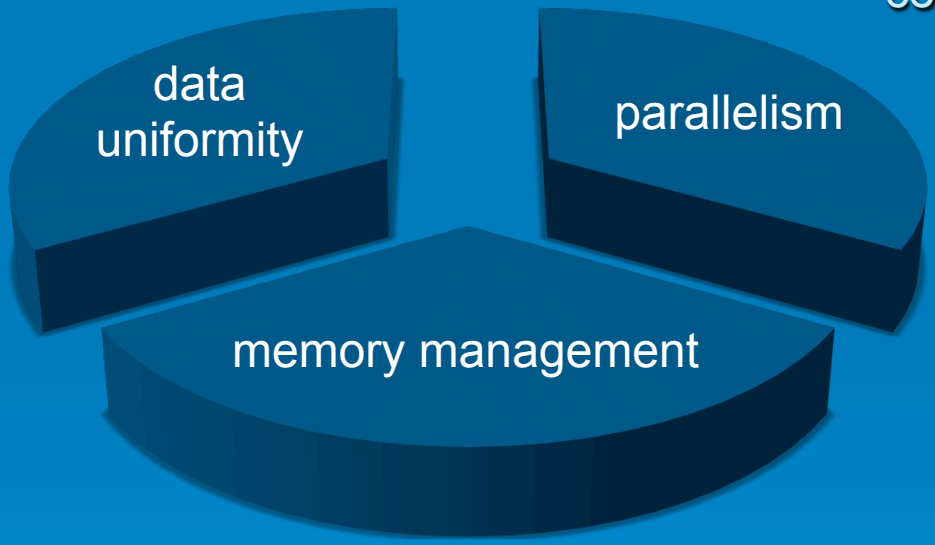
- process with soft real-time constraints
  - lose no data during calculations
  - deliver results within time frame
- build on standard JDK



Technical focus

- common view on data and devices
- immutability favors parallelism

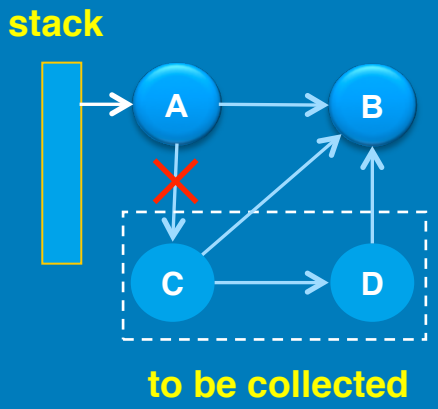
- decomposition for concurrency
- multithreaded communication



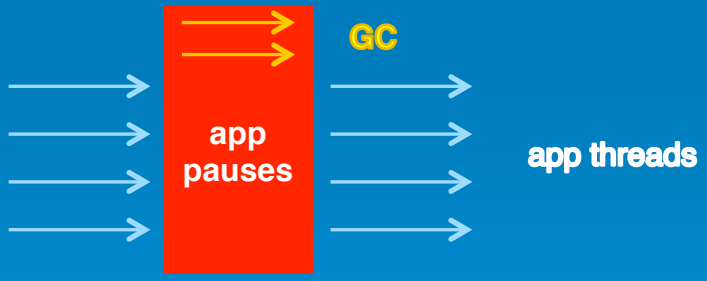
- optimal structures and algorithms
- garbage collectors, 32 vs 64 bit, Java Virtual Machine settings

Memory Management

Garbage Collection (GC)



- introduces non-deterministic behaviour
- slows the application with potentially long *stop-the-world pauses*

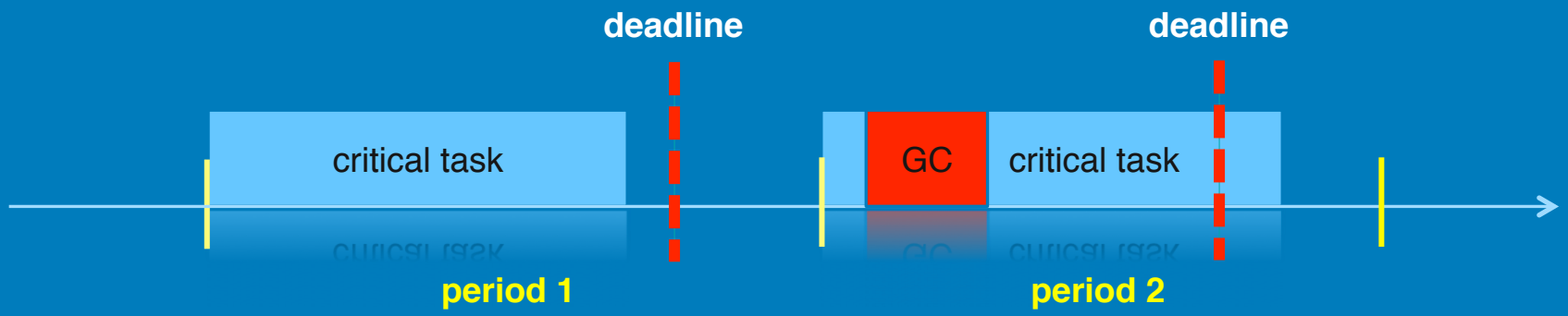


largest problem for performance

- makes it hard to achieve real-time

Soft real-time with GC

Real-time is not about speed



hard real-time: fatal

soft real-time: undesireable

Translates into requirements for GC

- we expect a degree of determinism
- number of stop-the-world pauses limited for a period

Solution

- steady progress in Garbage Collection techniques
- tuning JVM with over 50 properties
  - memory sizes,
    number of GC threads,…

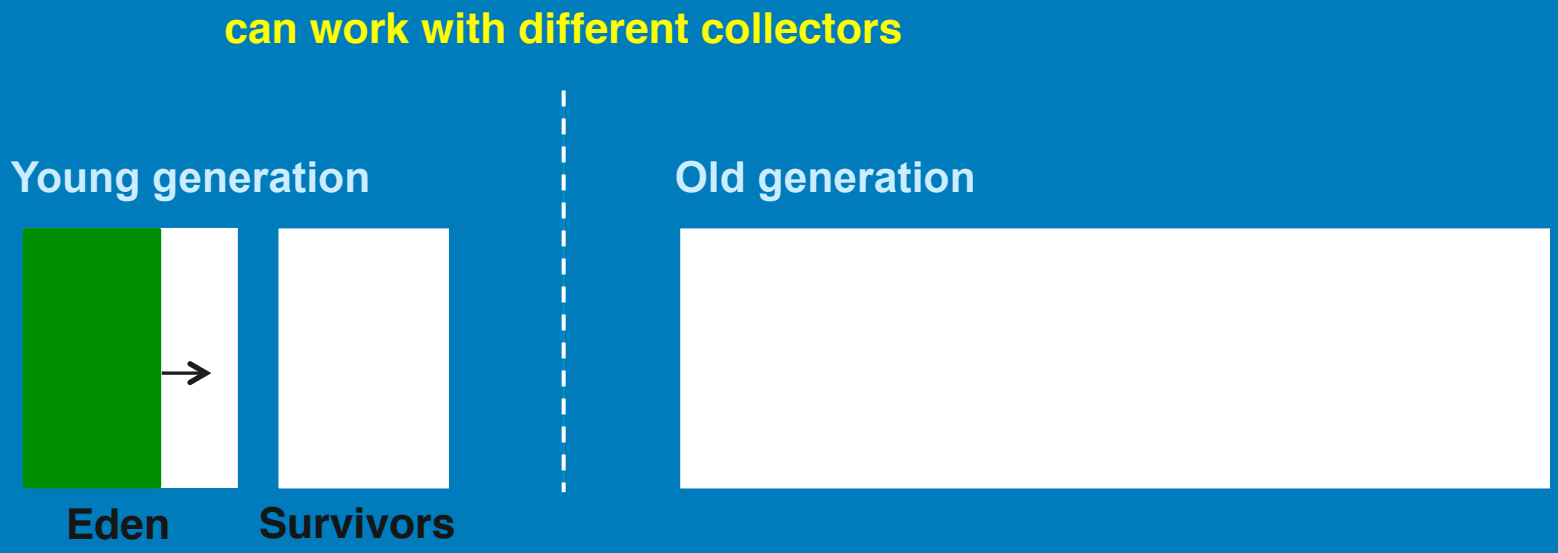
JVM GC history

90's	Serial Collector
00's	Parallel-Compacting, Concurrent Mark-Sweep
now	GarbageFirst

Garbage Collection concepts

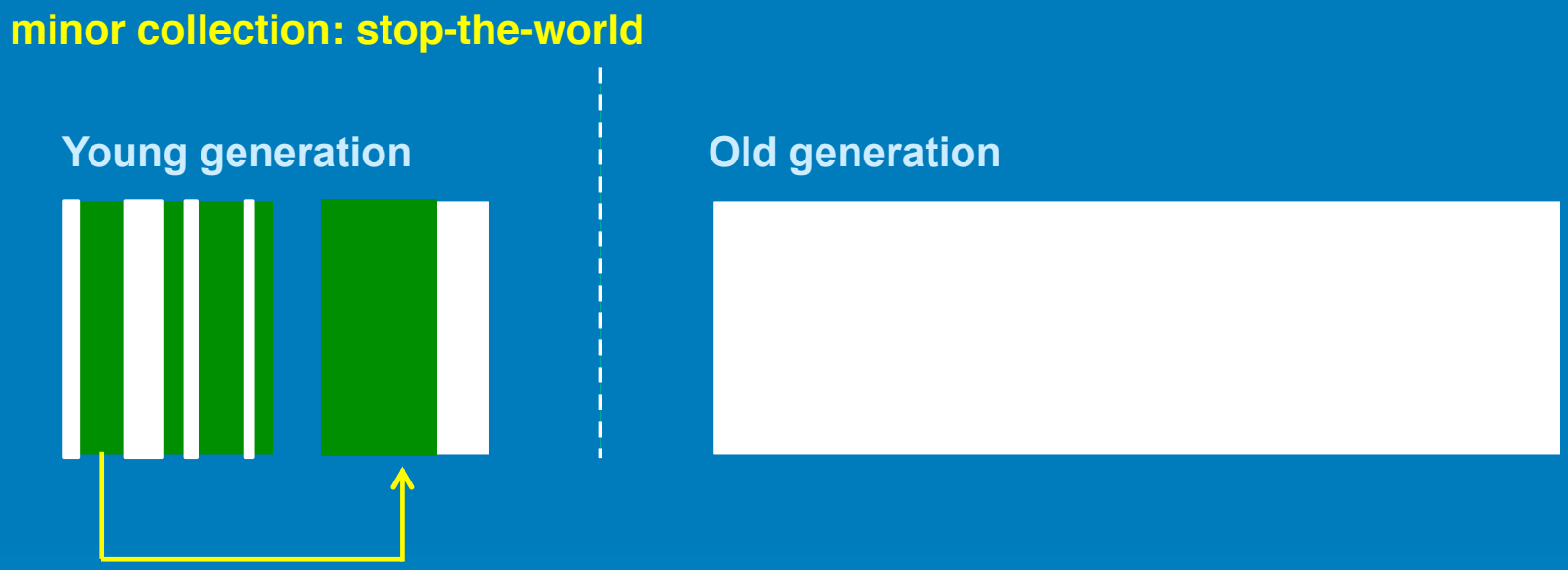
Heap

GC concepts



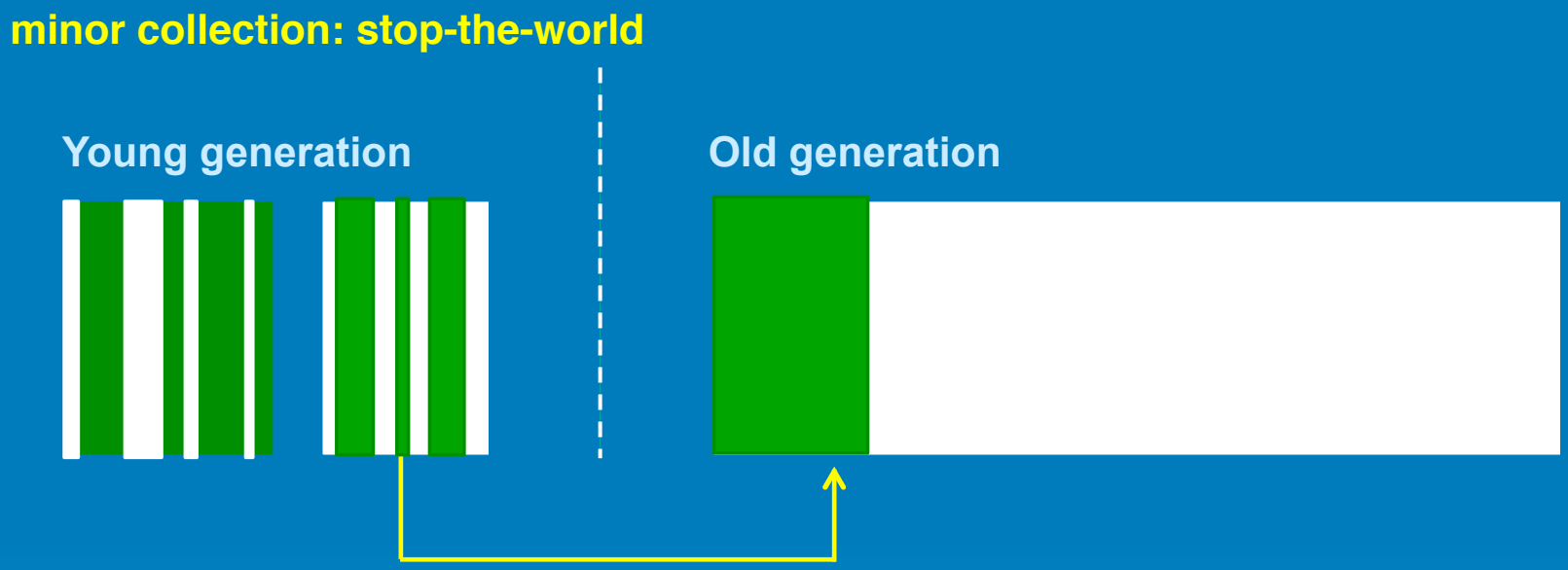
- Young much smaller than Old
- objects tend to live shortly
- new objects in Eden

GC concepts



- Young much smaller than Old
- objects tend to live shortly
- new objects in Eden
  moderate in Survivors

GC concepts

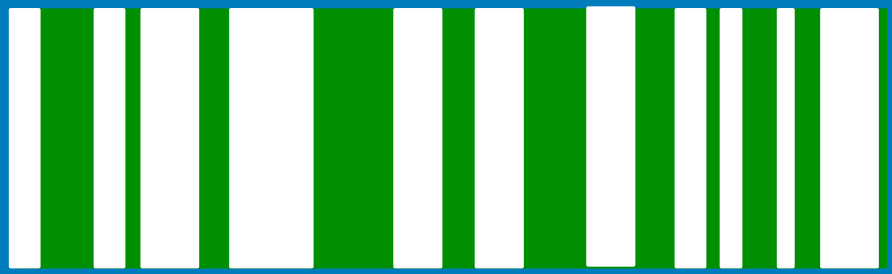


- Young much smaller than Old
- objects tend to live shortly
- new objects in Eden
  moderate in Survivors
  old in Old

GC concepts

major collection: stop-the-world

Old generation



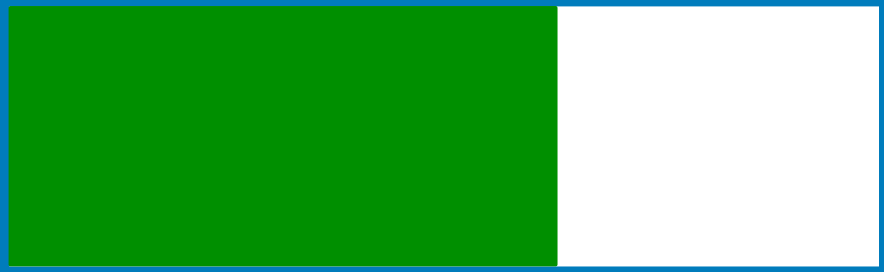
1st marking live objects

2nd sweeping memory

GC concepts

major collection: stop-the-world

Old generation

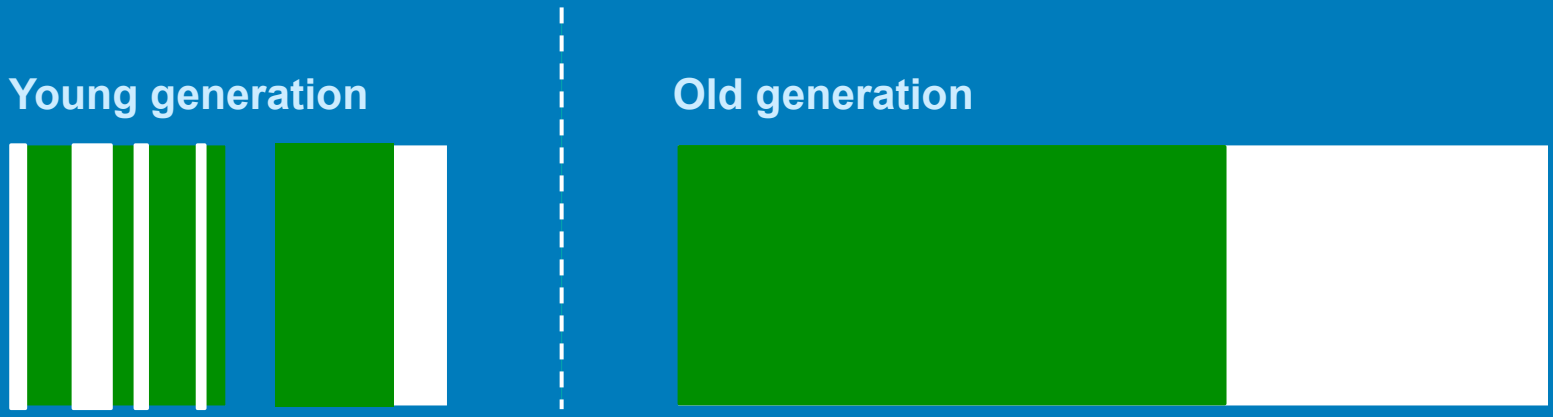


1st marking live objects

2nd sweeping memory

defragmentation: compacting

GC concepts



1st marking live objects

2nd sweeping memory defragmentation: compacting

Key improvements to collections:

- parallel – multiple GC threads
- concurrent – GC along with application

Concurrent Mark-Sweep (CMS)

parallel collection

parallel + concurrent marking & sweeping

Young generation

Old generation

- generational, incremental, parallel
- partially concurrent: marking & sweeping in stages
- no compacting

Well tuned, most effective in our tests

GarbageFirst (G1)

Meets soft real time goal with high probability

- default in JDK7, succeeds Concurrent Mark-Sweep
- targeted for multi-processors with large memories
  - heavy use of multithreading
  - heap with many equal regions, no generations
  - compaction
- algorithmically complex

enabling in Java 6:
-XX:+UnlockExperimentalVMOptions
-XX:+UseG1GC

Outcome

Performance analysis with Java Standard Edition 6

- fine-tuned CMS most effective, G1 close second

Observations

- 64 bit architecture
  - 4GB limit per JVM crossed
  - too much memory used - *performance penalty*
- repetitive nature of processing diminishes effects of dynamic *class loading*
- long *startup time* is negligible
- short lived objects, *locality*

Conclusions

- High Performance Computing with *soft real time* requirements can be achieved with modern JVMs
- JVM tuning is indispensable
  - select most fitting garbage collector
  - set JVM options
  - approach 64 bit boost with restraint
- constant improvement in memory management
  - G1 (Java 7) more efficient than CMS (Java 6)