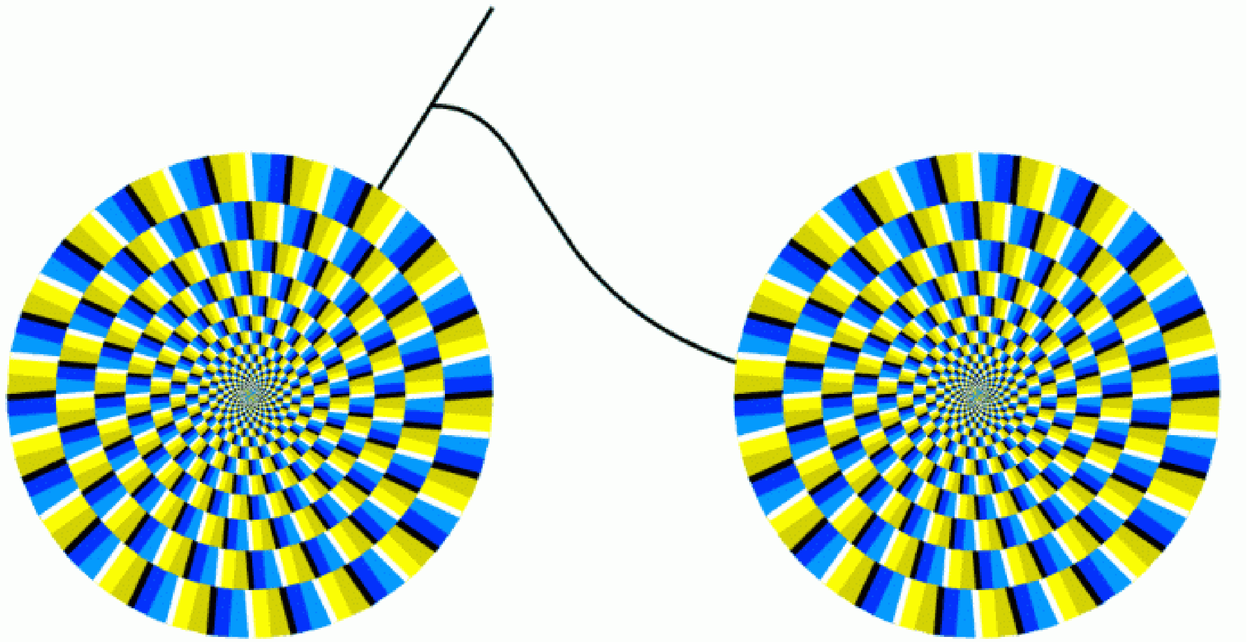
# *The Illusion Of Execution*

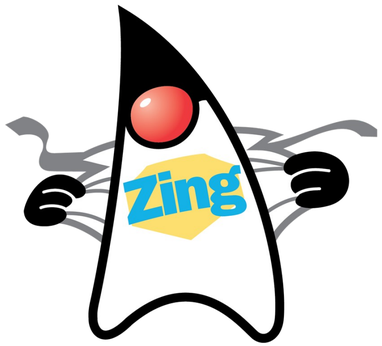


Nitsan Wakart (@nitsanw)
Lead Performance Engineer, Azul Systems

Thanks!

# I work on Zing!

- Awesome JVM, will be mentioned in slides
- Only on Linux/x86
- Aimed at server side systems
- Highly focused on responsiveness



# Psychosomatic, Lobotomy, Saw

It's X, you'll need Y, I'll get Z

**Monday, 1 December 2014**

## *The Escape of ArrayList.iterator()*

**{This post assumes some familiarity with JMH. For more JMH related content start at the new and improved JMH Resources Page and branch out from there!}**

Escape Analysis was a much celebrated optimisation added to the JVM in Java 6u23:

> "Based on escape analysis, an object's escape state might be one of the following:
>
> - `GlobalEscape` – An object escapes the method and thread. For example, an object stored in a static field, or, stored in a field of an escaped object, or, returned as the result of the current method.
> - `ArgEscape` – An object passed as an argument or referenced


http://psy-lob-saw.blogspot.com

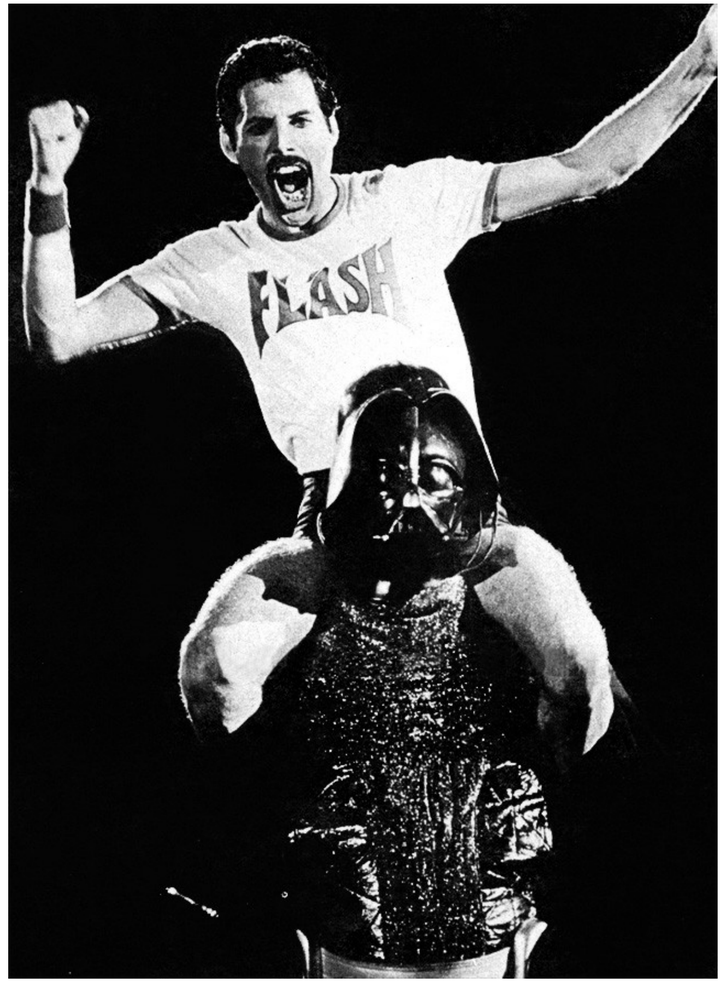

https://github.com/JCTools/JCTools

# The JVM is a Magical Place

- You write some Java/Scala/Clojure/JavaScript/Ruby
- Compile/Pack/Deploy

BOOM!

- JVM “executes” the “code”
- Use infinite memory!
- Run infinite threads!



# The Way Down

Your (byte) code...

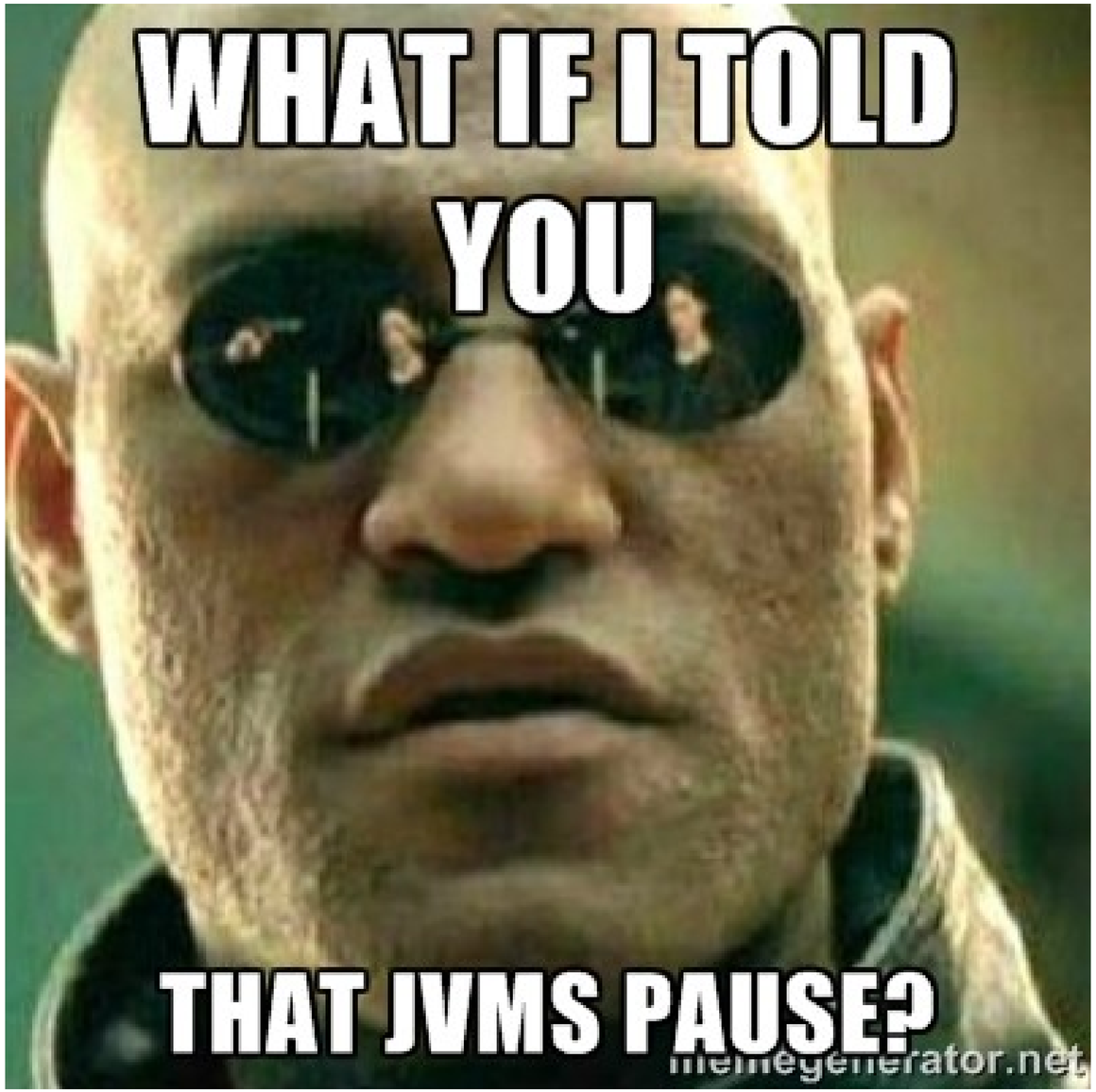
Is executed (interpreter/C1/C2)...

by a JVM (OpenJDK/Oracle/IBM/Zing)...

Which is a process of an OS (specific native libraries)...

Which is running on some hardware (specific instruction set)...

```
public void foo(Bar bar) {
  int nogCount = 0;
  for (int i = 0; i < 10; i++) {
    if (bar.getZog(i).isNog()) nogCount++;
  }
  while (nightIsYoung) {
    nogCount += hit(bar);
  }
  if (nogCount > MAX_NOG)
    throw new NogOverflowError();
}
```


WHAT IF I TOLD YOU
THAT JVMS PAUSE?
memegenerator.net

# Why Stop The World?

- Some GC phases
- Deoptimization
- Stack trace dump (and other JVMTI activities)
- Lock un-biasing
- Class redefinition
- And more!

See excellent talks:
https://www.youtube.com/watch?v=Y39kllzX1P8 : “With GC Solved, What Else Makes a JVM Pause?” by John Cutherson
https://vimeo.com/120533011 : “When Does the JVM JIT & Deoptimize?” by Doug Hawkins

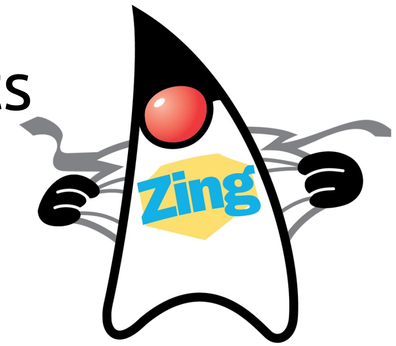
# Can one Java thread stop all other Java threads?

# You too can trigger a STW pause!

- On normal allocation (Young Gen full)
- On large object allocation (Old Gen full)
- On synchronized block (unbiasing)
- Hitting cold code/loading new classes (deoptimization)
- Profiling (JVMTI *GetStackTrace*())

# Zing: The pause-**less** JVM

- C4 - Fully Concurrent GC (including young gen)
- ReadyNow! - Persistent code profile, fighting Deopts
- No biased locking
- JVMTI *GetStackTrace* does not STW
- … some STW pauses exist (smaller than OS hiccups)

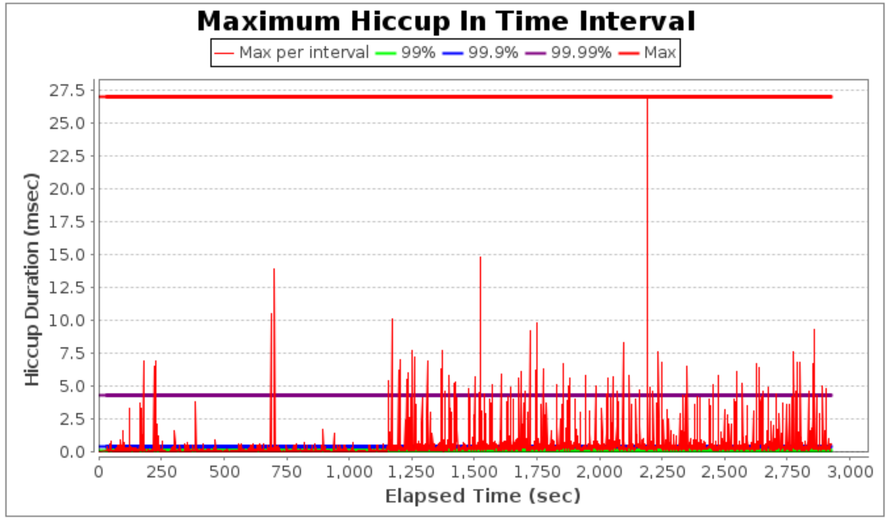


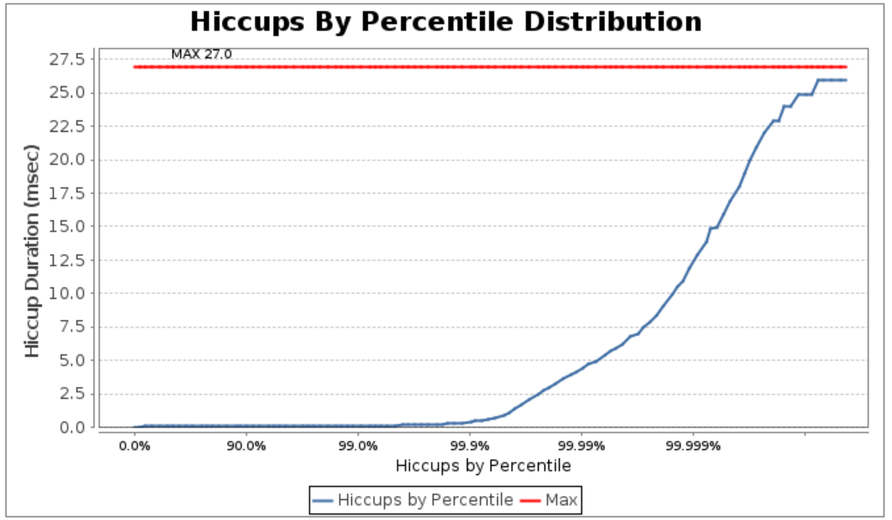
# The JVM...

... Must give us pause.

# jHiccup: A STW Pause Detector

- Try to run every X milliseconds
- Record deviations from schedule
- Make lovely graphs!




See: https://github.com/giltene/jHiccup

# How does a JVM stop the world?

To ‘Stop The World’ the JVM brings all **Mutator** threads to a **Safepoint**

- Raise **Safepoint** ‘flag’
- Wait for ALL threads to reach **Safepoint** state

# Safe == Good

# Safepoints == Goodpoints?

# At a Safepoint

“...the thread's representation of it's Java machine state is well described, and can be safely manipulated and observed by other threads in the JVM”

See full quote from Gil Tene, originally posted on the “Mechanical Sympathy” mailing list:
http://chriskirk.blogspot.ru/2013/09/what-is-java-safepoint.html

# Safepoint

(noun.)

A JVM thread state

- `Waiting/Idle/Blocked →  @Safepoint`
- `Running Java code    → !@Safepoint`
- `Running JNI code     →  @Safepoint`

http://blog.ragozin.info/2012/10/safepoints-in-hotspot-jvm.html
http://psy-lob-saw.blogspot.com/2014/03/where-is-my-safepoint.html

# Where do we see a Safepoint poll?

- Between every 2 bytecodes (interpreter)
- Backedge of non-'counted' loops (C1/C2)
- Method exit (C1/C2)
- JNI call exit

```
public void foo(Bar bar) {
  int nogCount = 0;
  for (int i = 0; i < 10; i++) {
    if (bar.getZog(i).isNog()) nogCount++;
  }
  while (nightIsYoung) {
    nogCount += hit(bar);
  }
  if (nogCount > MAX_NOG)
    throw new NogOverflowError();
}
```

```
public void foo(Bar bar) {
  int nogCount = 0;
  for (int i = 0; i < 10; i++) {
    if (bar.getZog(i).isNog()) nogCount++;
  }
  while (nightIsYoung) {
    nogCount += hit(bar);
    // Safepoint poll
  }
  if (nogCount > MAX_NOG)
    throw new NogOverflowError();
  // Safepoint poll
}
```

# Safepoint poll: **OpenJDK**

- Read from a special page:

```
”test   DWORD PTR [rip+0xfffffffffe690e53],eax”
```

- JVM Sets the page to protected, polling threads trap a SEGV and go to safepoint
- Look for **`{poll}`** or **`{poll_return}`** in the assembly comments

# Safepoint poll: **Zing**

- Read the thread local safe point flag:

```
”gs:cmp4i [0x40 tls._please_self_suspend],0
 jnz 0x500a0186; Where the safepoint code be”
```

- JVM Sets the thread flag to 1, polling threads hop to
- Look for **”tls._please_self_suspend”**

```
public void foo(Bar bar) {
  int nogCount = 0;
  for (int i = 0; i < 10; i++) {
    if (bar.getZog(i).isNog()) nogCount++;
  }
  while (nightIsYoung) {
    nogCount += hit(bar);
    if (MUST_SAFEPOINT) gotoSafepoint();
  }
  if (nogCount > MAX_NOG)
    throw new NogOverflowError();
  if (MUST_SAFEPOINT) gotoSafepoint();
}
```

# Safepoint polls == Overhead

It's just a harmless lil' safepoint they said

# How Bad?

```
private static final class DataSet {
    private final int[] data;

    int intSize() { return data.length; }
    int intGet(int index) { return data[index]; }
    void intSet(int index, int v) { data[index] = v; }

    long longSize() { return data.length; }
    int longGet(long index) { return data[(int) index]; }
    void longSet(long index, int v) { data[(int) index] = v; }
}
```

See: https://github.com/netty/netty/pull/3969#issuecomment-132559757

```
@Benchmark
public int sumNoSafepoint() {
  int sum = 0;
  for (int index = 0; index < datasetA.intSize(); ++index) {
    sum += datasetA.intGet(index);
  }
  return sum;
}

@Benchmark
public int sumSafepoint() {
  int sum = 0;
  for (long index = 0; index < datasetA.longSize(); ++index) {
    sum += datasetA.longGet(index);
  }
  return sum;
}
```

# How Bad?

```
Benchmark                          Score      Error  Units
Benchmark.copyNoSafepoint         76.943 ±   14.256  ns/op
Benchmark.copySafepoint          796.583 ±   55.583  ns/op ← 10x worse!
Benchmark.equalsNoSafepoint      367.192 ±   16.398  ns/op
Benchmark.equalsSafepoint        806.421 ± 196.106  ns/op ← 2.1x worse!
Benchmark.fillNoSafepoint         84.075 ±   19.033  ns/op
Benchmark.fillSafepoint          567.866 ±   10.154  ns/op ← 7x worse!
Benchmark.sumNoSafepoint         338.204 ±   44.529  ns/op
Benchmark.sumSafepoint           585.657 ± 105.808  ns/op ← 1.5x worse!
```

# Safepoint poll side effects

- Can prevent loop unrolling
- Can prevent SuperWord optimizations
- Can prevent OptimizeFill optimization
- … Inhibits the compiler

# Safepoint polls == BAD!

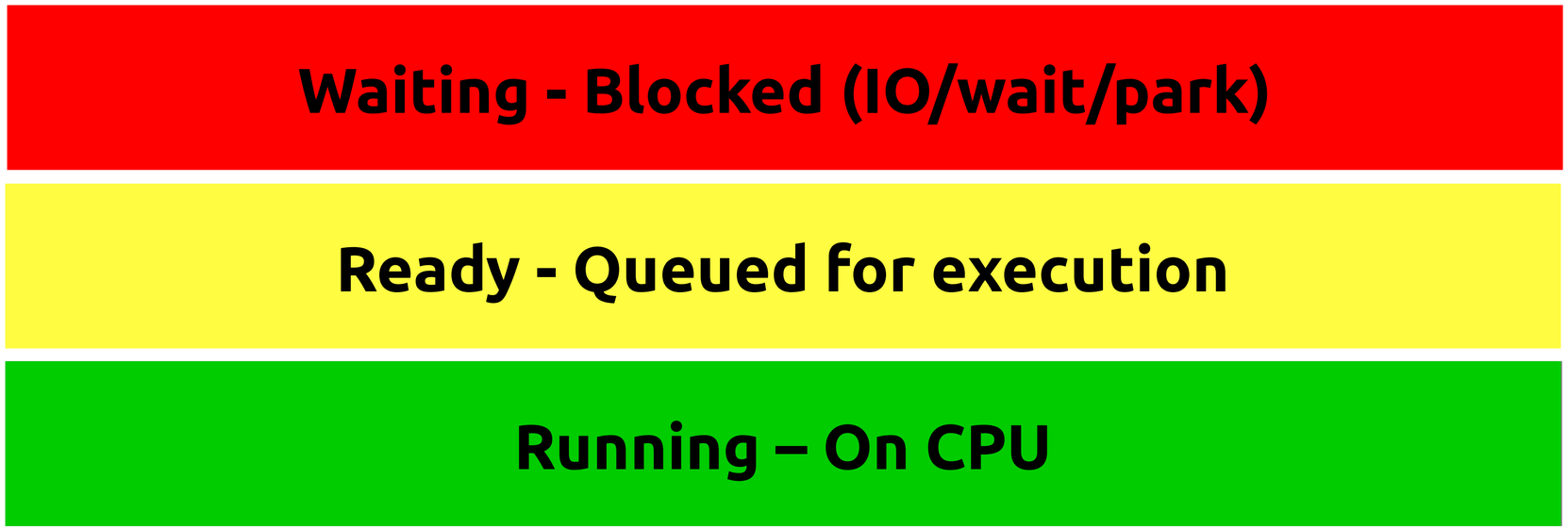
# DEMO: When will it exit?

# Delaying a Safepoint == BAD!

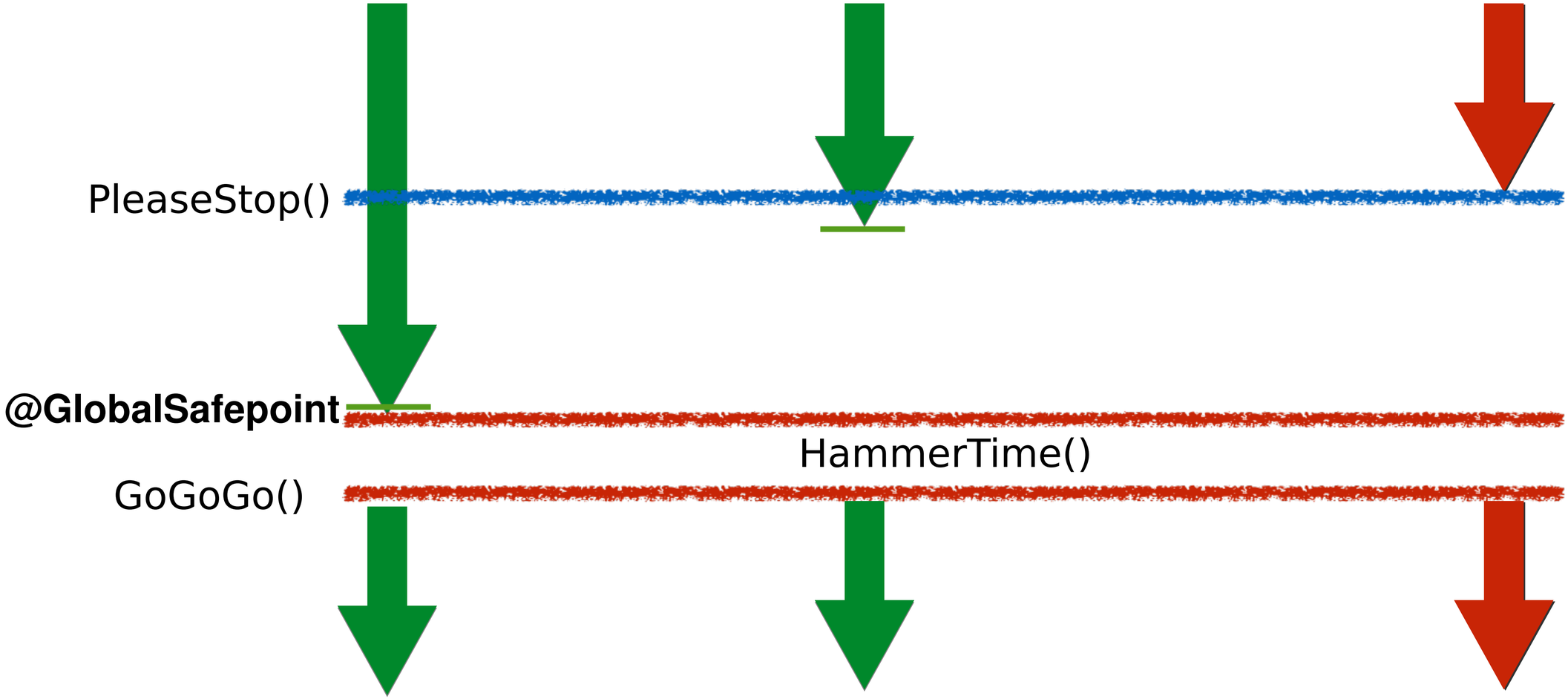
# How long was this GC pause?

```
[GC [PSYoungGen: 109884K->14201K(139904K)]
                 691015K→595332K(1119040K),
                 0.0454530 secs]
```

# Thread States Legend



# TTSP - Time To Safepoint



# Where is TTSP reported?

- Zing: Reported as part of pause time (with break down)
- OpenJDK: Not included in GC pause time
- OpenJDK: Must use -XX:+PrintGCApplicationStoppedTime

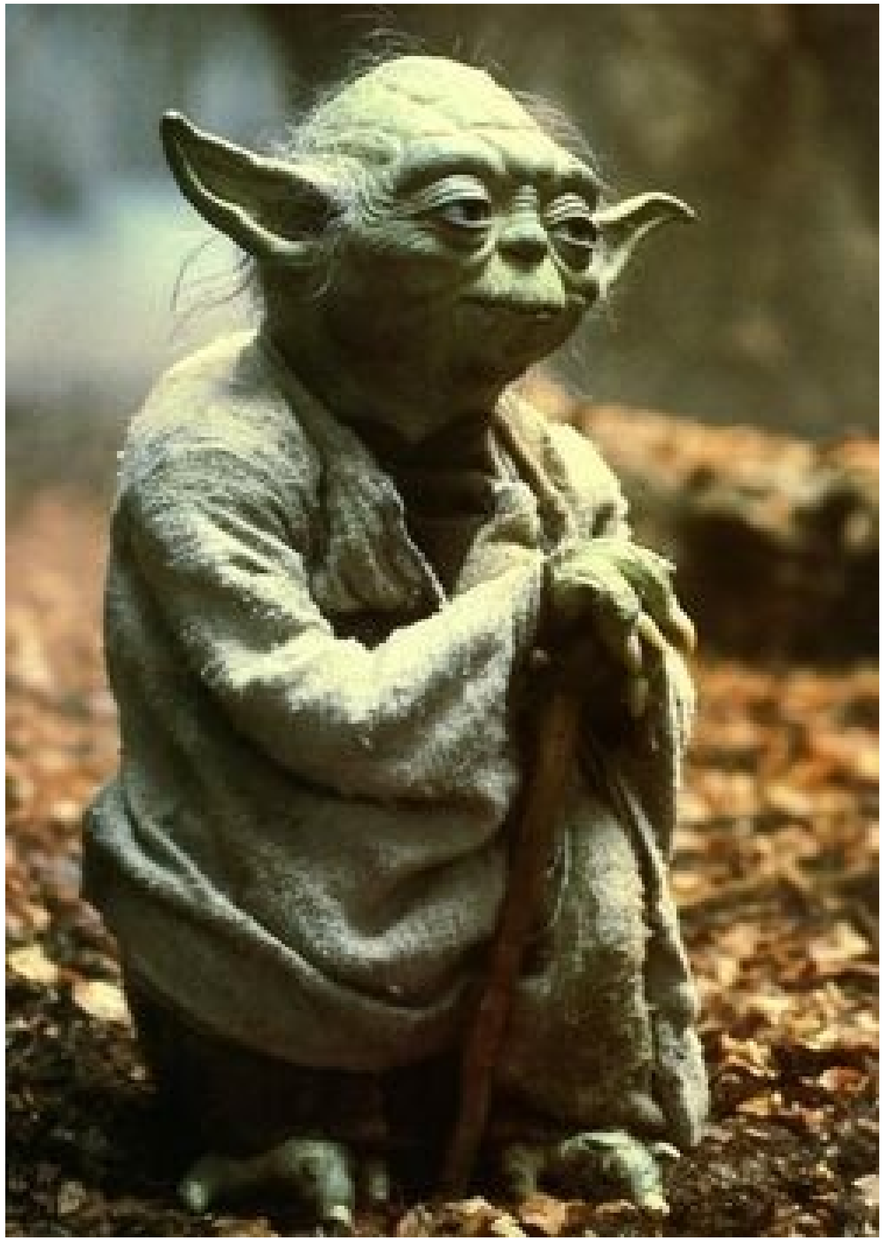
```
Total time for which application threads were stopped: 4.2399590
seconds, Stopping threads took: 4.2398886 seconds
```

- OpenJDK: -XX:+PrintSafepointStatistics -XX:PrintSafepointStatisticsCount=1

**Max Actual Pause = TTSP + work in safepoint**

# Long TTSP == BAD!

Safepoint? hmm yes...
Poll it you must



# So what?

- TTSP can dominate pause times, track it!
- No safepoint polls in counted loops:
  - ✔ On Zing can use -XX+KeepSafepointsInCountedLoops
- Some 'big' operations may have no safepoint polls:
  - ✗ System.arrayCopy (chunked on Zing)
  - ✗ Object clone/init (chunked on Zing)
  - ✗ String.indexOf
  - ✗ Others...

# Time To Safepoint
# vs.
# Safepoint poll overhead

# Safepoint poll insertion

- Backedge of non-'counted' loops
- Method exit (removed by inlining)

**Heuristics based on instruction count**

# Instruction count != Time

Is it faster to read from a HashMap or a file?

COVERBASH.COM

I Like to Move it...Move it..

He Likes to Move it...Move it..

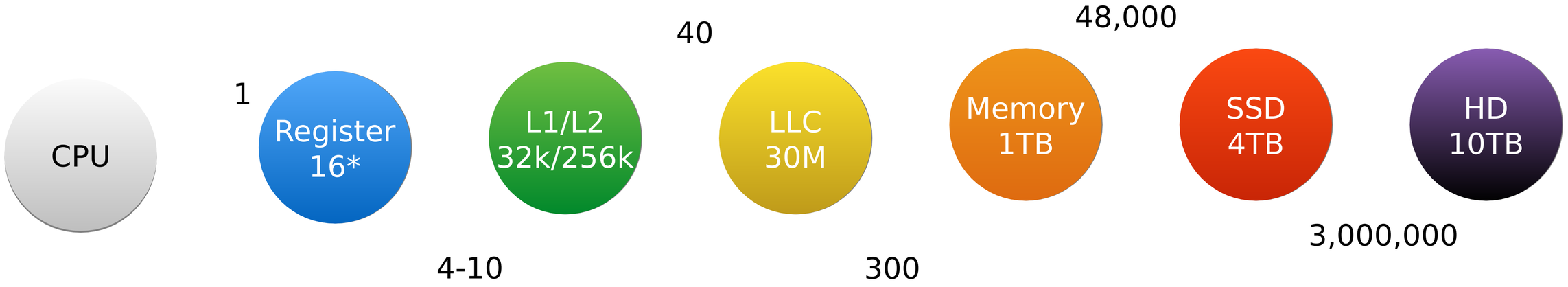
You Like To.....?

**MOVE IT !**

# Why is MOV important?

```
mov     r10d,DWORD PTR [rsi+0x10] ;*getfield hold/Holder::set@2 (line 8)
mov     r11,r10
shl     r11,0x3
xor     eax,eax
cmp     rdx,r11
je      0x000000010b065eb2  ; Holder::set@5 (line 8)
mov     r8d,DWORD PTR [rdx+0x8]  ; implicit exception: dispatches to  0x000000010b065f11
cmp     r8d,0xef5d495b      ;   {oop('java/lang/Integer')}
jne     0x000000010b065ebe  ;*invokevirtual equals/Holder::set@(line 11)
mov     r8d,DWORD PTR [r12+r10*8+0x8] ; implicit exception: dispatches to 0x000000010b065ef9
cmp     r8d,0xef5d495b      ;   {oop('java/lang/Integer')}
jne     0x000000010b065edd  ;*instanceof/Integer::equals@(line 764)/Holder::set@(line 11)
mov     r8d,DWORD PTR [rdx+0xc]
shl     r10,0x3             ;*checkcast/Integer::equals@(line 765)/Holder::set@(line 11)
mov     r11d,DWORD PTR [r10+0xc]
cmp     r8d,r11d
je      0x000000010b065eb2  ; - java.lang.Integer::equals@18 (line 765)/Holder::set@(line 11)
inc     DWORD PTR [rsi+0xc]  ;*putfield setCount/Holder::set@(line 15)
mov     r10,rsi
mov     r11,rdx
shr     r11,0x3
mov     DWORD PTR [rsi+0x10],r11d
shr     r10,0x9
movabs r11,0x10a4e9000
mov     BYTE PTR [r11+r10*1],r12b ;*putfield hold/Holder::set@(line 14)
mov     eax,0x1
add     rsp,0x30; This is 0x000000010b065eb2
pop     rbp
test    DWORD PTR [rip+0xffffffffff71c143],eax        # 0x000000010a782000;   {poll_return}
ret
```

# Memory Topology




Converting to cycles, assumed 3 cycles per nano-second
http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html

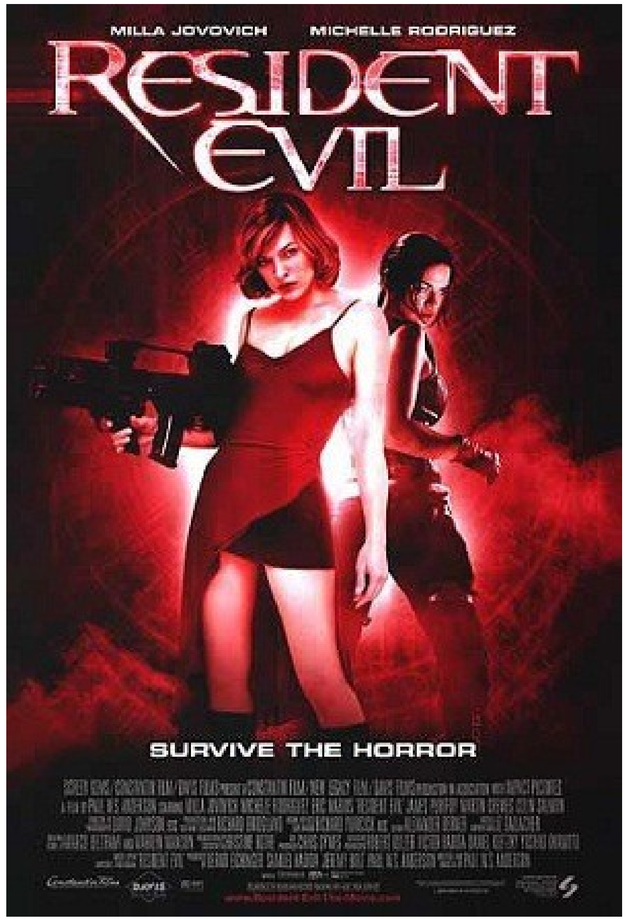
# Beer Cache Hierarchy

- Register - Bottle at your lips (1 sec)
- L1/L2 - … on table (4 - 10 secs)
- LLC - … waiting on the bar (40 secs)
- Main memory - … at store across the road (5 mins)
- SSD - … in another country? (13.3 hrs)
- HD - … on the moon? (34 days, 17 hrs)

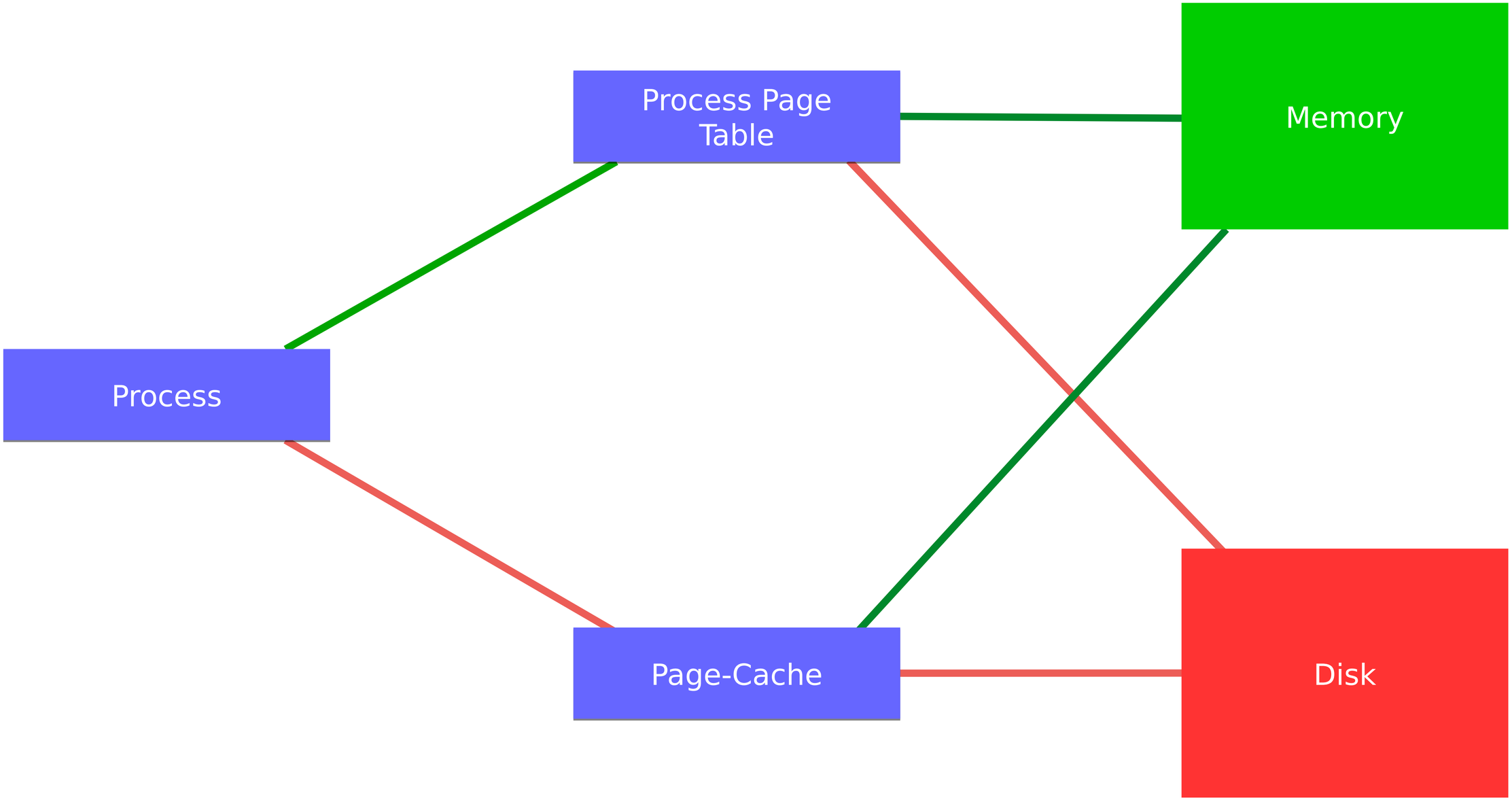
See: http://blog.netopyr.com/2014/11/28/reactions-to-the-beer-cache-hierarchy/

# Resident vs. Virtual Memory

- How much memory can a process use?
  - OS maps when actually used
- What happens when we exceed physical?
  - Swap
  - Page faults




"Resident evil ver4" by impawards. Licensed under Fair use of copyrighted material in the context of Resident Evil (film) via Wikipedia - http://en.wikipedia.org/wiki/File:Resident_evil_ver4.jpg#mediaviewer/File:Resident_evil_ver4.jpg


Process Page
Table
Memory
Process
Page-Cache
Disk

# So What?

- Memory access assumed short (1 instruction)
- Page fault → very large TTSP
- `MappedByteBuffer` → IO out of safepoint, mostly great… but see prev. point
  - ↙ Use `mlock` (requires JNI call, not part of JDK)?

# So What?

- Benchmarking? Use relevantly sized data sets and access patterns
- Avoid using more memory than available
  - ✔ disable swap? (Set swappiness=0)
  - ✔ Buy more memory
- Monitor Page faults
- Consider priming memory?

Have you ever written a single threaded Java application?

# The JVM Process: Threads

How many threads for an application?

- Application Java threads (Main, Thread etc)
- Application native threads (native lib)
- JVM Threads (GC, Compiler, JMX, RMI…)

**There’s no such thing as a single threaded Java application**

# Threads Example

(Oracle JDK8, on i5/dual core laptop, no args)

- Application threads
- 4 GC Threads (ParallelGC)
- 3 Compiler threads (1 C1 + 2 C2)
- … and others
- 15 threads reported by jstack, 19 by OS

# Multi-Tasking OS

More processes than cores

- Yay!!! Gimme ALL the threads!
- … unless they all want to run at the same time

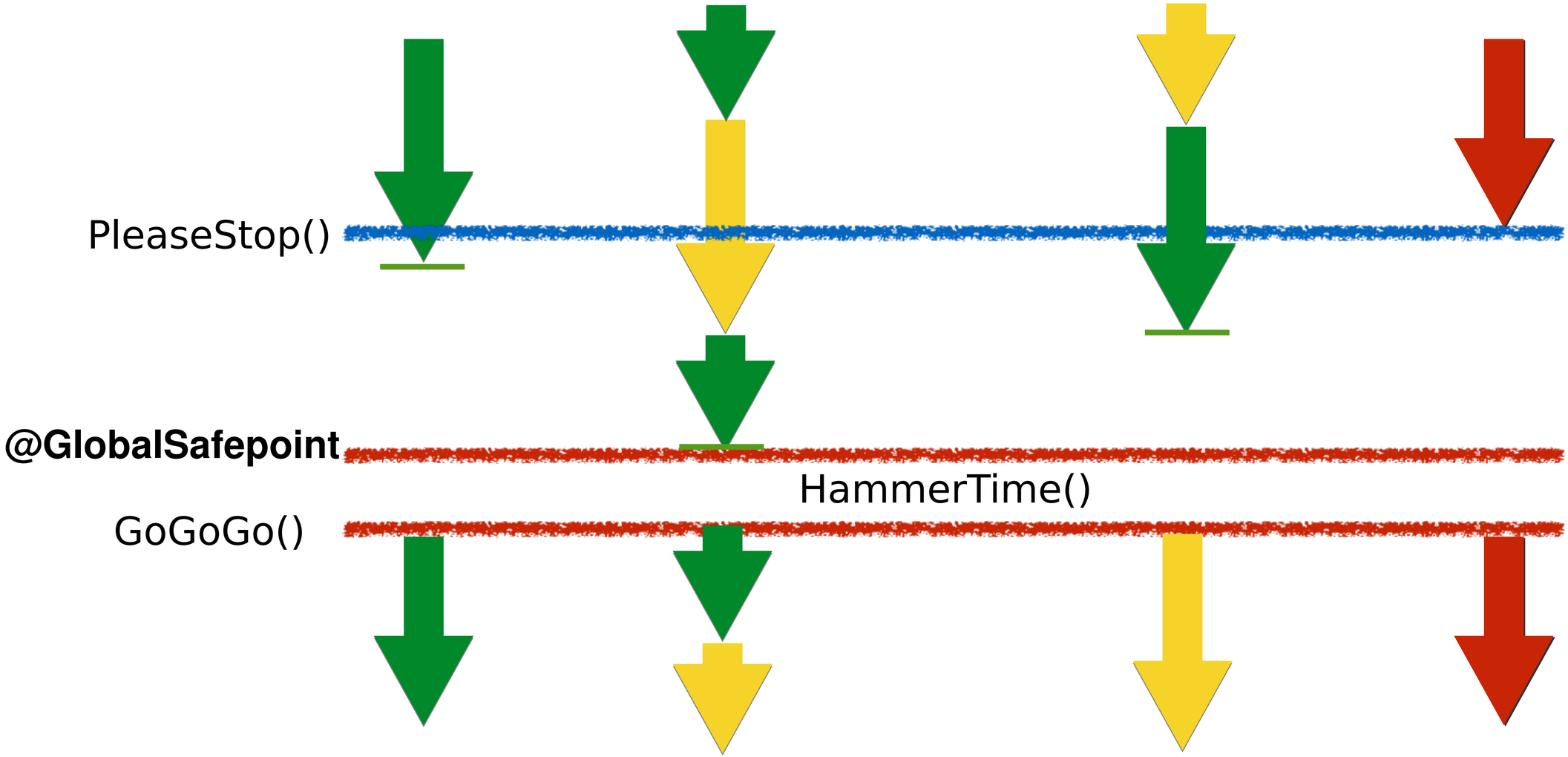
Scheduling and interrupts

- Fairness (thread priority, starvation)
- Context Switching (overhead)

# So What?

- OS can suspend Java threads
- Threads suspension – not necessarily @Safepoint
- All suspended threads must resume and reach safepoint before safepoint work can start
- Runnable Threads > Cores → Longer TTSP

# TTSP - Time To Safepoint



# So What?

- Try and avoid having more **ready**/**running** threads than cores
  - Monitor run queue (avg. load), ctx switches
- Consider controlling OS resource management
  - Use taskset/numactl
  - JVM is not aware of taskset/numactl/isocpus
- Consider controlling JVM thread counts

# OOPs

hold = love;

Compressed Oops

```
mov     r10,rsi
mov     r11,rdx
shr     r11,0x3
mov     DWORD PTR [rsi+0x10],r11d
shr     r10,0x9
movabs  r11,0x10a4e9000
mov     BYTE PTR [r11+r10*1],r12b
```

Card Marking

# What’s an OOP?

- Ordinary Object Pointer
- Java: Object reference -> JVM: OOP
- Pointers to managed data on the heap

# Java Hidden Symbols

```
void copyPoint(Point p1, Point p2) {
  p1.x = p2.x;
}
```

```
void copyPoint(oop p1, oop p2) {
 address a1 = readBarrier(p1);
 address a2 = readBarrier(p2);
 oop x = getObject(a2 + xFieldOffset);
 putObject(a1 + xFieldOffset, x);
 writeBarrier(a1, x);
 safepoint_poll();
}
```

# Memory Barrier

(not the JMM kind)

“…a block [of code, executed] on reading from or writing to certain memory locations by certain threads or processes.”

Memory Management Reference:
http://www.memorymanagement.org/glossary/b.html#term-barrier-1

# Compressed OOPs

-XX:+UseCompressedOops

- Want large heaps (> 4Gb) and 32bit addresses
- Objects aligned on 8 bytes*
- Can compress OOP by dropping last 3 bits (>>3)
- Must decompress address to use it (<<3)
- Max referenced heap size is now 4Gb * 8 = 32Gb

https://wikis.oracle.com/display/HotSpotInternals/CompressedOops

* Can change with -XX:ObjectAlignmentInBytes

# Compressed Oops Example(x86):

JAVA:

```
long v = this.l.longValue();
```

-XX:-UseCompressedOops:

```
mov r10, QWORD PTR [rsi+0x18]          ; r10= this.l
mov r10, QWORD PTR [r10+0x10]          ; r10= l.value
```

-XX:+UseCompressedOops:

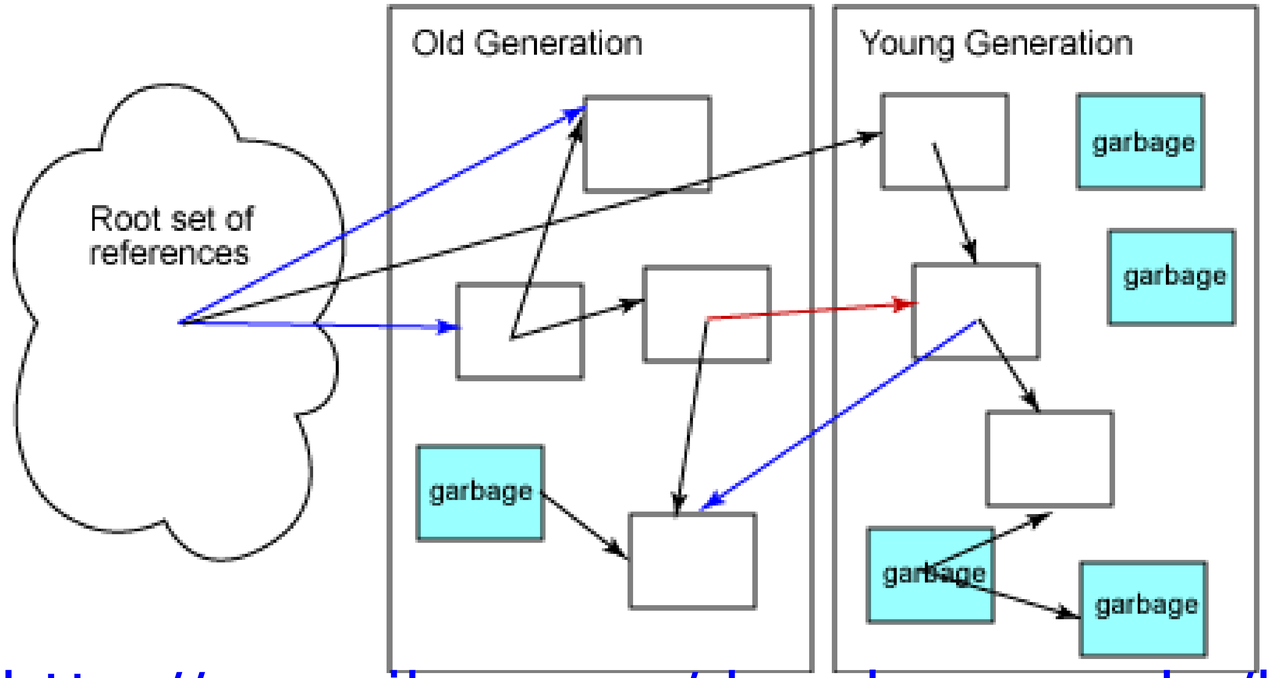
```
mov r11d, DWORD PTR [rsi+0x10]         ; r11d= this.l
mov r10,  QWORD PTR [r12+r11*8+0x10]; r10= l.value
```

# CompressedOops **instanceof** ReadBarrier

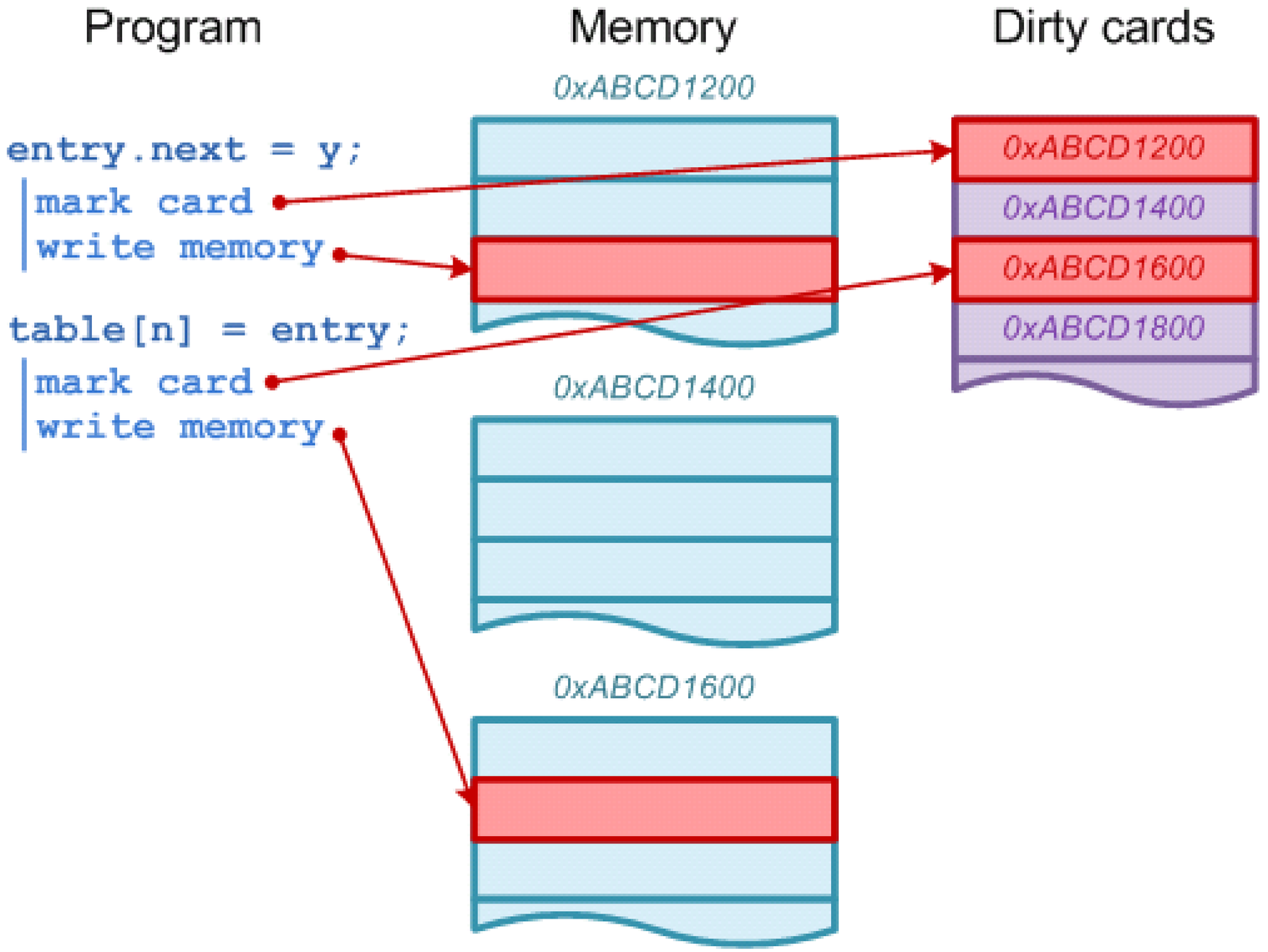
- Must be decompressed before read ‘through’
- Can be copied without decompression
- Can be compared without decompression
- Balance size vs. computation

# Card Marking

“The JVM maintains a card map, with one byte corresponding to each card in the heap. Each time a pointer field in an object in the heap is modified, the corresponding byte in the card map for that card is set.”




Brian Goetz - http://www.ibm.com/developerworks/library/j-jtp11253/




http://blog.griddynamics.com/2011/06/understanding-gc-pauses-in-jvm-hotspots.html

# CardMarking **instanceof** WriteBarrier

- An optimisation for young collections
- Reduce the impact of OldGen size on scan time
- Introduce a small overhead
- Comes in different flavours!

# CardMarking v1 (default)

```
; rsi is 'this' address
; rdx is setter param, reference to bar
; this.foo = bar
mov    QWORD PTR [rsi+0x20],rdx ; <- WHAT WE WANTED
mov    r10,rsi
; r10 = r10 >> 9;
shr    r10,0x9
; r11 is base of card table, byte[] CARD_TABLE
mov    r11,0x7ebdfcff7f00
; Mark 'this' card as dirty
; CARD_TABLE[this address >> 9] = 0
mov    BYTE PTR [r11+r10*1],0x0
```

# CardMarking v2 (-XX:+UseCondCardMark)

```
; rsi is 'this' address
; rdx is setter param, reference to bar
; r10 = this
mov   r10,rsi  <- WHAT WE WANTED
; r10 = r10 >> 9
shr    r10,0x9
; r11 = CARD_TABLE
mov    r11,0x7f7cb98f7000
; r11 = CARD_TABLE + (this >> 9)
add    r11,r10
; r8d = CARD_TABLE[this >> 9]
movsx  r8d,BYTE PTR [r11]
test   r8d,r8d
; if(CARD_TABLE[this >> 9] == 0) goto 0x00007fc4a1071d7d
je     0x00007fc4a1071d7d
; CARD_TABLE[this >> 9] = 0
mov    BYTE PTR [r11],0x0
0x00007fc4a1071d7d:
mov    QWORD PTR [rsi+0x20],rdx  ; this.foo = bar
```

# CardMarking v3 (-XX:+UseG1GC)

```
  movsx  edi,BYTE PTR [r15+0x2d0] ; read GC flag
  cmp    edi,0x0; if (flag != 0)
  jne    0x00000001066fc601; GOTO OldValBarrier
Label WRITE:

  mov    QWORD PTR [rsi+0x20],rdx; this.foo = bar
  mov    rdi,rsi; rdi = this
  xor    rdi,rdx; rdi = this XOR bar
  shr    rdi,0x14; rdi = (this XOR bar) >> 20
  cmp    rdi,0x0; If this and bar are not same gen
  jne    0x00000001066fc616; GOTO NewValBarrier
Label EXIT:
;...
Label OldValBarrier:
  mov    rdi,QWORD PTR [rsi+0x20]
  cmp    rdi,0x0; if(this.foo == null)
  je     0x00000001066fc5dd; GOTO WRITE
  mov    QWORD PTR [rsp],rdi ; setup rdi as parameter
  call   0x000000010664bca0  ;   {runtime_call}
  jmp    0x00000001066fc5dd; GOTO WRITE
Label NewValBarrier:
  cmp    rdx,0x0; bar == null
  je     0x00000001066fc5f5 goto Exit
  mov    QWORD PTR [rsp],rsi
  call   0x000000010664bda0  ;   {runtime_call}
  jmp    0x00000001066fc5f5  ; GOTO exit;
```

<- WHAT WE WANTED

# So What?

- References mean extra work
- Impact can change by option/GC/JVM
- ‘Normalized/Flat’ data structures can help
  - Inheritance vs. Composition?
  - Consider access patterns
- Value Types might help (Java 9?)
- ObjectLayout might help (Java 9?)

```
while (hasQ() && hasTime()){
   tryA();
}
return;
```