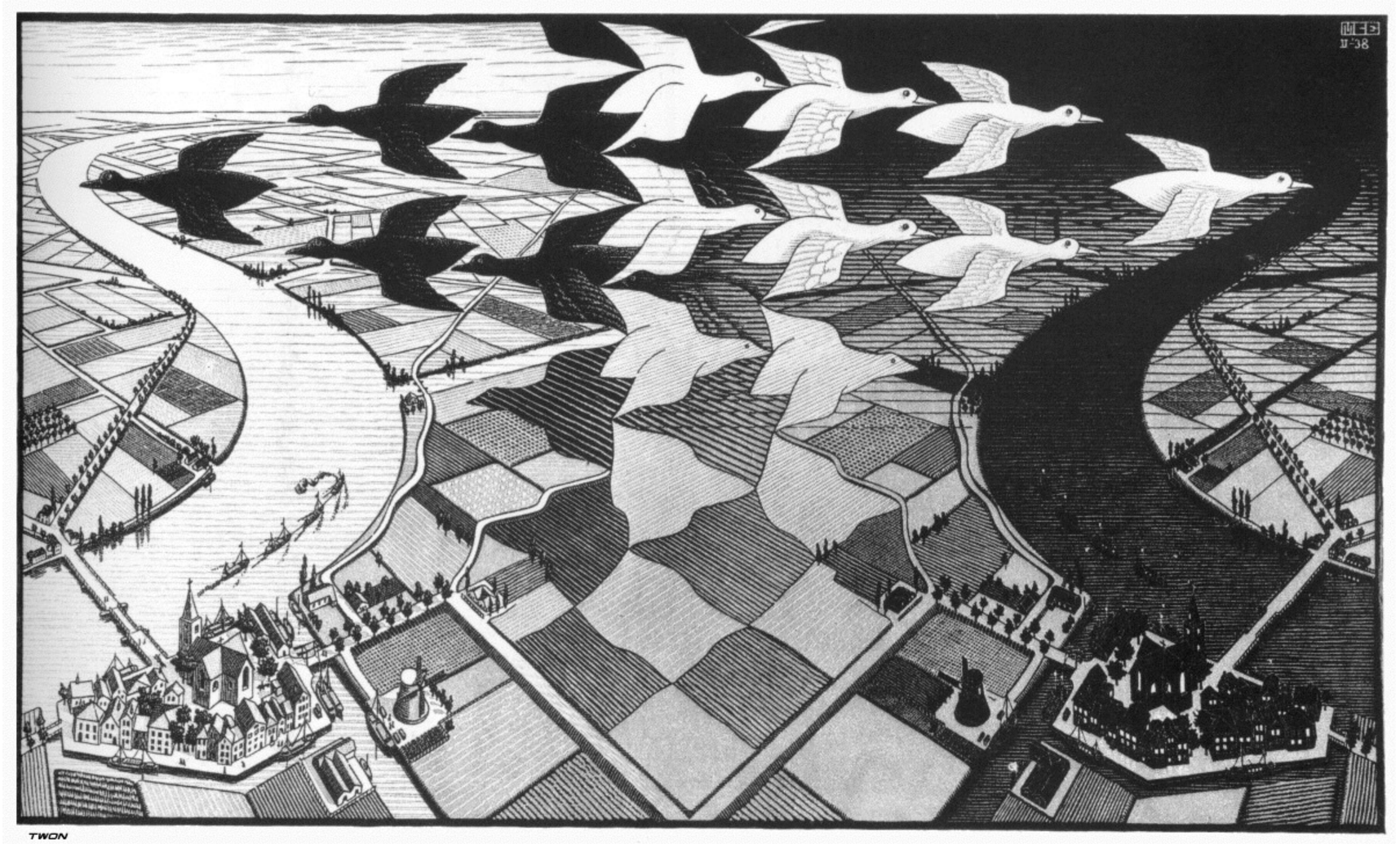
The Illusion Of Execution



http://www.wikiart.org/en/m-c-escher/day-and-night

Thanks!

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

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!}

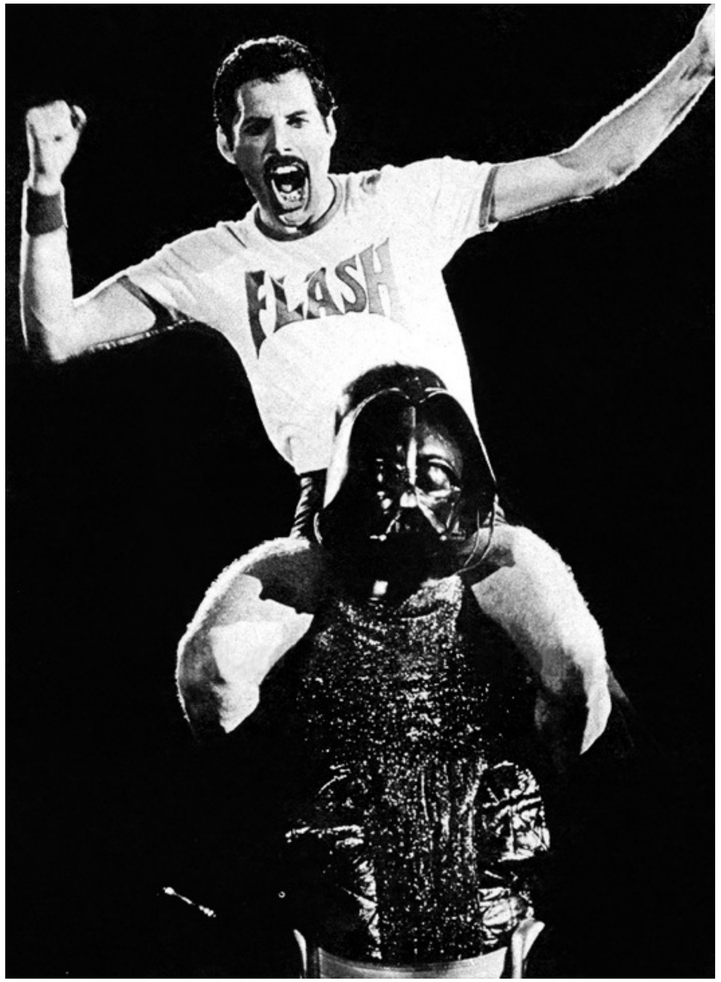
Escape Analysis was a much celebrated optimisation added to the JVM in Java 6u23:

> "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

The JVM is a Magical Place

- You write some Java/Scala/Clojure/….
- Compile/Pack/Deploy
- … MAGIC! …
- JVM executes the “code”



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)…

“I fucking hate assembly”

–Daniel San

All the assembly you need (for this talk)

- Each line is a single instruction
- Instructions take 1 to N CPU cycles
- MOV DST, SRC (this is Intel syntax)
- CMP A,B + JE DST -> if(A == B) goto DST
- INC A -> A++;
- **DON’T PANIC!!!!!**

Code -> Assembly

```
public class Holder<T> {
  T hold;
  int setCount;

  boolean set(T love) {
    if(love == hold) {
      return false;
    }
    else if(love != null
        && love.equals(hold)) {
      return false;
    }
    hold = love;
    setCount++;
    return true;
  }
}
```

```
[Constants]
  # {method} 'set' '(Ljava/lang/Object;)Z' in 'Holder'
  # this:     rsi:rsi   = 'Holder'
  # parm0:    rdx:rdx   = 'java/lang/Object'
  #           [sp+0x40]  (sp of caller)
  0x000000010b065e20: mov    r10d,DWORD PTR [rsi+0x8]
  0x000000010b065e24: shl    r10,0x3
  0x000000010b065e28: cmp    rax,r10
  0x000000010b065e2b: jne    0x0000000010b037960  ;   {runtime_call}
  0x000000010b065e31: data32 xchg ax,ax
  0x000000010b065e34: nop    DWORD PTR [rax+rax*1+0x0]
  0x000000010b065e3c: data32 data32 xchg ax,ax
[Verified Entry Point]
  0x000000010b065e40: mov    DWORD PTR [rsp-0x14000],eax
  0x000000010b065e47: push   rbp
  0x000000010b065e48: sub    rsp,0x30
  0x000000010b065e4c: mov    r10d,DWORD PTR [rsi+0x10]
  0x000000010b065e50: mov    r11,r10
  0x000000010b065e53: shl    r11,0x3
  0x000000010b065e57: xor    eax,eax
  0x000000010b065e59: cmp    rdx,r11
  0x000000010b065e5c: je     0x0000000010b065eb2  ;*if_acmpne
                                                  ; - Holder::set@5 (line 8)
  0x000000010b065e5e: mov    r8d,DWORD PTR [rdx+0x8]  ; implicit exception: dispatches to
0x000000010b065f11
  0x000000010b065e62: cmp    r8d,0xef5d495b
  0x000000010b065e69: jne    0x000000010b065ebe  ;*invokevirtual equals
                                                  ; - Holder::set@19 (line 11)
  0x000000010b065e6b: mov    r8d,DWORD PTR [r12+r10*8+0x8]
                                                  ; implicit exception: dispatches to
0x000000010b065ef9
  0x000000010b065e70: cmp    r8d,0xef5d495b     ;   {oop('java/lang/Integer')}
  0x000000010b065e77: jne    0x000000010b065edd  ;*instanceof
                                                  ; - java.lang.Integer::equals@1 (line 764)
  0x000000010b065e79: mov    r8d,DWORD PTR [rdx+0xc]
  0x000000010b065e7d: shl    r10,0x3
  0x000000010b065e81: mov    r11d,DWORD PTR [r10+0xc]
  0x000000010b065e85: cmp    r8d,r11d
  0x000000010b065e88: je     0x000000010b065eb2  ;*if_icmpne
                                                  ; - java.lang.Integer::equals@8 (line 765)
                                                  ; - Holder::set@19 (line 11)
```

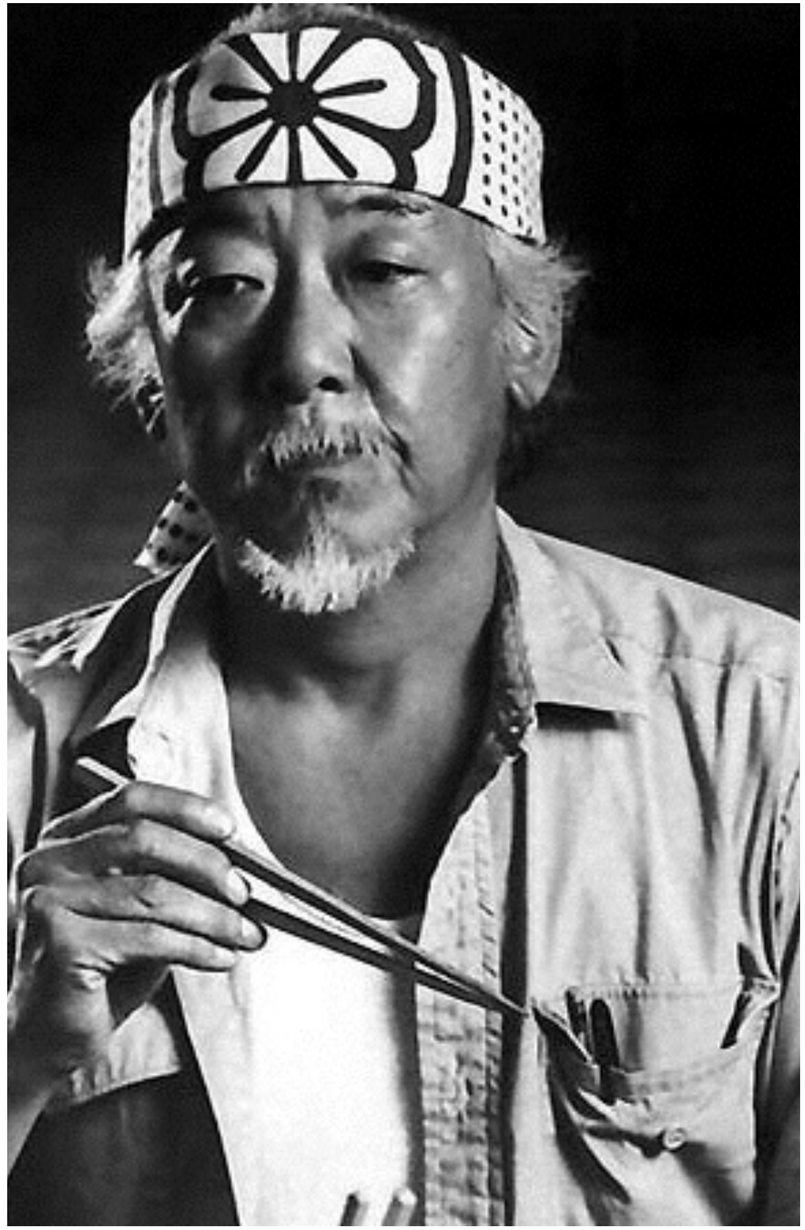
```
0x000000010b065e8a: inc    DWORD PTR [rsi+0xc]  ;*putfield setCount
                                                ; - Holder::set@39 (line 15)
0x000000010b065e8d: mov    r10,rsi
0x000000010b065e90: mov    r11,rdx
0x000000010b065e93: shr    r11,0x3
0x000000010b065e97: mov    DWORD PTR [rsi+0x10],r11d
0x000000010b065e9b: shr    r10,0x9
0x000000010b065e9f: movabs r11,0x10a4e9000
0x000000010b065ea9: mov    BYTE PTR [r11+r10*1],r12b  ;*putfield hold
                                                ; - Holder::set@29 (line 14)
0x000000010b065ead: mov    eax,0x1              ;*getfield hold
                                                ; - Holder::set@2 (line 8)
0x000000010b065eb2: add    rsp,0x30
0x000000010b065eb6: pop    rbp
0x000000010b065eb7: test   DWORD PTR [rip+0xffffffffff71c143],eax
                                                ;   {poll_return}
0x000000010b065ebd: ret
0x000000010b065ebe: mov    rbp,rsi
                    ;*synchronization entry
0x000000010b065ec1: mov    QWORD PTR [rsp+0x8],rdx
                    ; - Holder::set@-1 (line 8)
0x000000010b065ec6: mov    DWORD PTR [rsp+0x10],r10d
0x000000010b065ecb: mov    esi,0xfffffffe
                    ;*getfield hold
0x000000010b065ed0: data32 xchg ax,ax
                    ; - Holder::set@2 (line 8)
0x000000010b065ed3: call   0x000000010b038f20  ; OopMap{rbp=Oop [8]=Oop [16]=NarrowOop off=184}
                                                ;*invokevirtual equals
                                                ; - Holder::set@19 (line 11)
                                                ;   {runtime_call}
0x000000010b065ed8: call   0x0000000109fe8202  ;*invokevirtual equals
                                                ; - Holder::set@19 (line 11)
                                                ;   {runtime_call}
0x000000010b065edd: mov    rbp,rsi
0x000000010b065ee0: mov    QWORD PTR [rsp+0x8],rdx
0x000000010b065ee5: mov    DWORD PTR [rsp+0x14],r10d
0x000000010b065eea: mov    esi,0xfffffffe
                    ; - Holder::set@19 (line 11)
0x000000010b065eef: call   0x000000010b038f20  ; OopMap{rbp=Oop [8]=Oop [20]=NarrowOop off=212}
                                                ;*instanceof
                                                ; - java.lang.Integer::equals@1 (line 764)
                                                ; - Holder::set@19 (line 11)
                                                ;   {runtime_call}
0x000000010b065ef4: call   0x0000000109fe8202  ;   {runtime_call}
                    ; - java.lang.Integer::equals@1 (line 764)
0x000000010b065ef9: mov    rbp,rsi
                    ; - Holder::set@19 (line 11)
0x000000010b065efc: mov    QWORD PTR [rsp+0x8],rdx
0x000000010b065f01: mov    esi,0xfffffff4
                    ;*checkcast
0x000000010b065f06: nop
                    ; - java.lang.Integer::equals@12 (line 765)
0x000000010b065f07: call   0x000000010b038f20  ; OopMap{rbp=Oop [8]=Oop off=236}
                    ; - Holder::set@19 (line 11)
                                                ;*instanceof
                                                ; - java.lang.Integer::equals@1 (line 764)
                                                ; - Holder::set@19 (line 11)
                                                ;   {runtime_call}
0x000000010b065f0c: call   0x0000000109fe8202  ;   {runtime_call}
                    ; - java.lang.Integer::equals@1 (line 764)
                    ; - Holder::set@19 (line 11)
```

```
0x000000010b065f11: mov    rbp,rsi
0x000000010b065f14: mov    QWORD PTR [rsp+0x8],rdx
0x000000010b065f19: mov    esi,0xffffffad
0x000000010b065f1e: nop
0x000000010b065f1f: call   0x000000010b038f20  ; OopMap{rbp=Oop [8]=Oop off=260}
                                                ;*ifnull
                                                ; - Holder::set@11 (line 11)
                                                ;   {runtime_call}
0x000000010b065f24: call   0x0000000109fe8202  ;*ifnull
                                                ; - Holder::set@11 (line 11)
                                                ;   {runtime_call}
0x000000010b065f29: hlt
0x000000010b065f2a: hlt
0x000000010b065f2b: hlt
0x000000010b065f2c: hlt
0x000000010b065f2d: hlt    # 0x000000010a782000
0x000000010b065f2e: hlt
0x000000010b065f2f: hlt
0x000000010b065f30: hlt
0x000000010b065f31: hlt
0x000000010b065f32: hlt
0x000000010b065f33: hlt
0x000000010b065f34: hlt
0x000000010b065f35: hlt
0x000000010b065f36: hlt
0x000000010b065f37: hlt
0x000000010b065f38: hlt
0x000000010b065f39: hlt
0x000000010b065f3a: hlt
0x000000010b065f3b: hlt
0x000000010b065f3c: hlt
0x000000010b065f3d: hlt
0x000000010b065f3e: hlt
0x000000010b065f3f: hlt
[Exception Handler]
[Stub Code]
0x000000010b065f40: jmp    0x000000010b05f0a0  ;   {no_reloc}
[Deopt Handler Code]
0x000000010b065f45: call   0x000000010b065f4a
0x000000010b065f4a: sub    QWORD PTR [rsp],0x5
0x000000010b065f4f: jmp    0x000000010b038b00  ;   {runtime_call}
0x000000010b065f54: hlt
0x000000010b065f55: hlt
0x000000010b065f56: hlt
0x000000010b065f57: hlt
OopMapSet contains 4 OopMaps
#0
OopMap{rbp=Oop [8]=Oop [16]=NarrowOop off=184}
#1
OopMap{rbp=Oop [8]=Oop [20]=NarrowOop off=212}
#2
OopMap{rbp=Oop [8]=Oop off=236}
#3
OopMap{rbp=Oop [8]=Oop off=260}
```

How to print assembly: psy-lob-saw.blogspot.com/2013/01/java-print-assembly.html

‘Focus!’

Master Miagi



Method Prologue

```
  # {method} 'set' '(Ljava/lang/Object;)Z' in 'Holder'
   # this:      rsi:rsi   = 'Holder'
   # parm0:     rdx:rdx   = 'java/lang/Object'
  #           [sp+0x40]  (sp of caller)
  0x000000010bec3aa0: mov    r10d,DWORD PTR [rsi+0x8]
  0x000000010bec3aa4: shl    r10,0x3
  0x000000010bec3aa8: cmp    rax,r10
  0x000000010bec3aab: jne    0x000000010be97960  ;   {runtime_call}
  0x000000010bec3ab1: data32 xchg ax,ax
  0x000000010bec3ab4: nop    DWORD PTR [rax+rax*1+0x0]
  0x000000010bec3abc: data32 data32 xchg ax,ax
[Verified Entry Point]
  0x000000010bec3ac0: mov    DWORD PTR [rsp-0x14000],eax
  0x000000010bec3ac7: push   rbp
  0x000000010bec3ac8: sub    rsp,0x30           ;*synchronization entry
                                                ; - Holder::set@-1 (line 6)
```

Assuming hold != love

```
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            ;*getfield hold/Holder::set@(line 8)
add    rsp,0x30; This is 0x000000010b065eb2
pop    rbp
test   DWORD PTR [rip+0xffffffffff71c143],eax        # 0x000000010a782000;   {poll_return}
ret
```

Conditional Inlining

```
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  ; - Integer::equals@(line 765)/Holder::set@(line 11)
```

- The JIT compiler inlined Integer::equals
- Optimistically compiled, but checks class to trigger deoptimization
- Our code is compiled based on it’s usage!

Compiler Reordering

```
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)
```

- The JIT compiler reordered lines 15 and 14
- This can happen to YOU!

Instructions you have?
Get cycles, you must!
Hmmm, yes….

```
$jstack 12345
```

How many threads are running?

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)

Thread States

Waiting - Thread is blocked (IO/wait()/park()…)

Ready - You are queued for execution (yield)

Running - CPU is busy with your logic

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

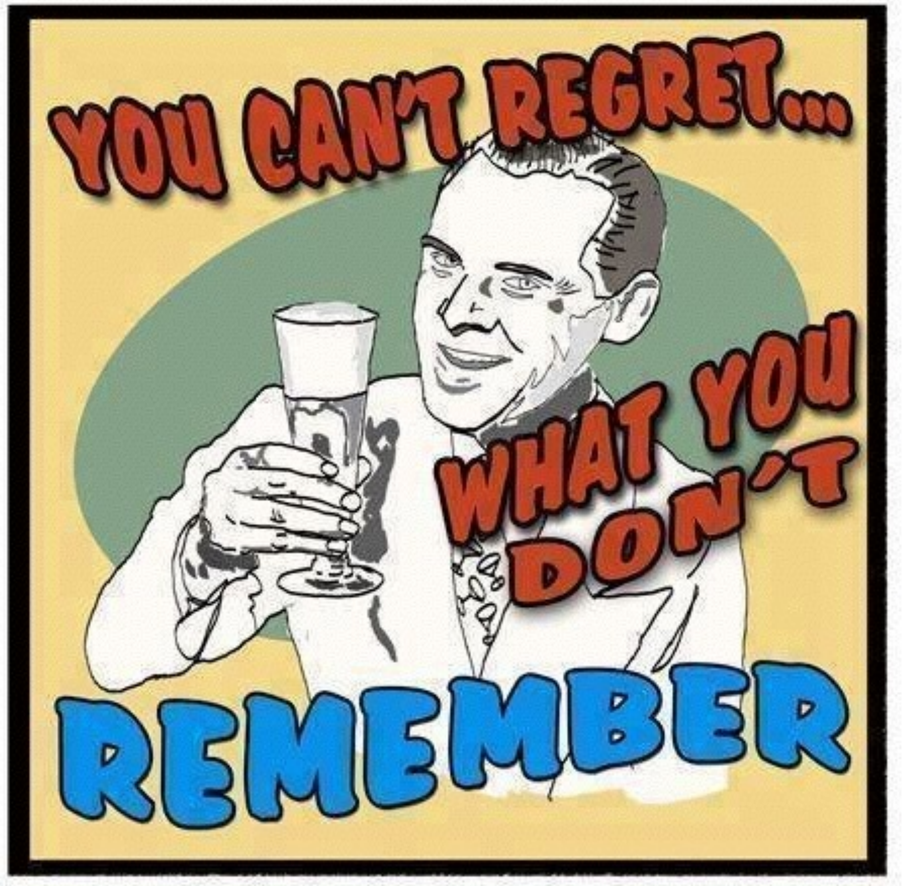
(Oracle JDK8u20, on i5/dual core laptop, no args)

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

So What?

- Try and avoid having more **ready/running** threads than cores
  - Look out for long run queue (avg. load), ctx switches
- Consider controlling OS resource management
  - Use taskset/numactl/isocpus to reduce contention
  - Note that JVM is not aware of taskset/numactl/isocpus
- Consider controlling JVM thread counts

The Memory Illusion

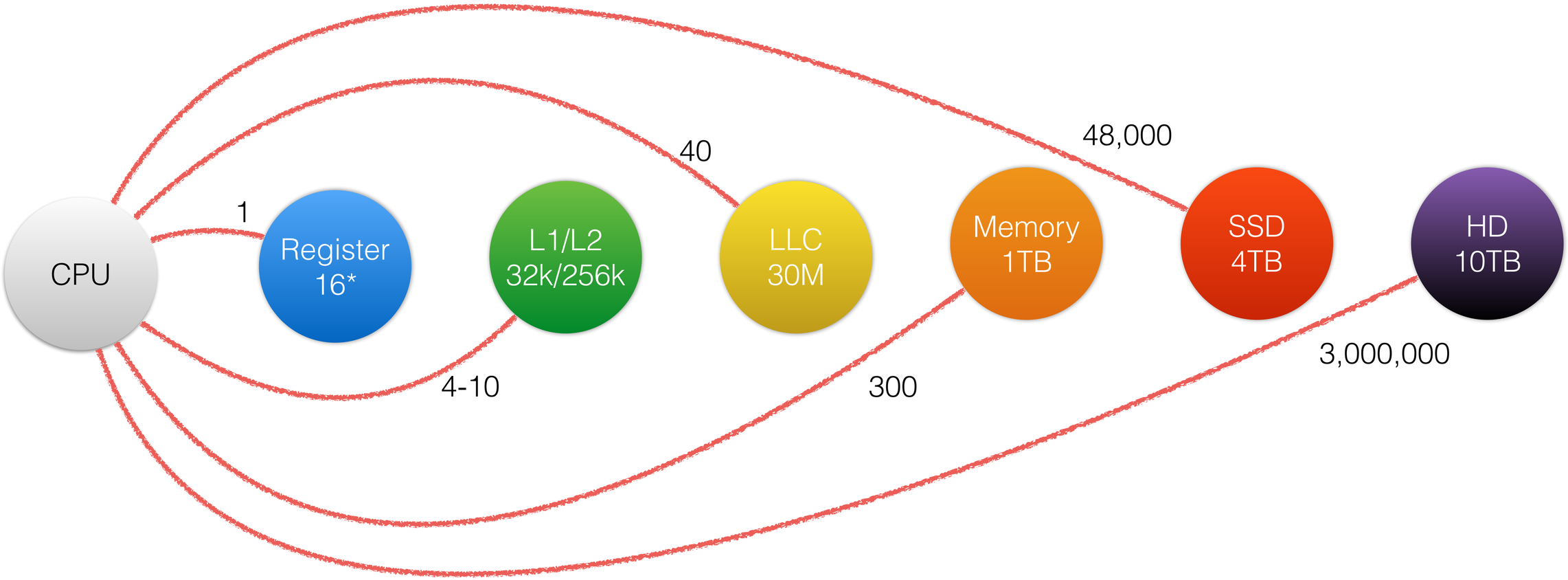


How long to pub.getBeer()?

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

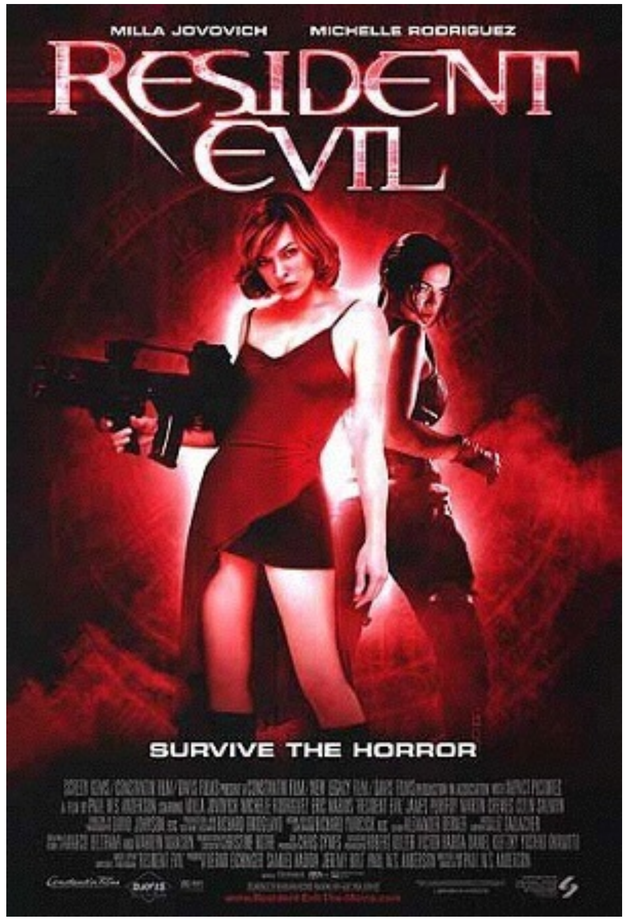
- Register - Bottle at your lips (1 sec)
- L1/L2 - … on table (4 - 10 secs)
- LLC - … waiting on the bar (40 secs)
- Main memory - … at store across the road (5 mins)
- SSD - … in another country? (13.3 hrs)
- HD - … on the moon? (34 days, 17 hrs)

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

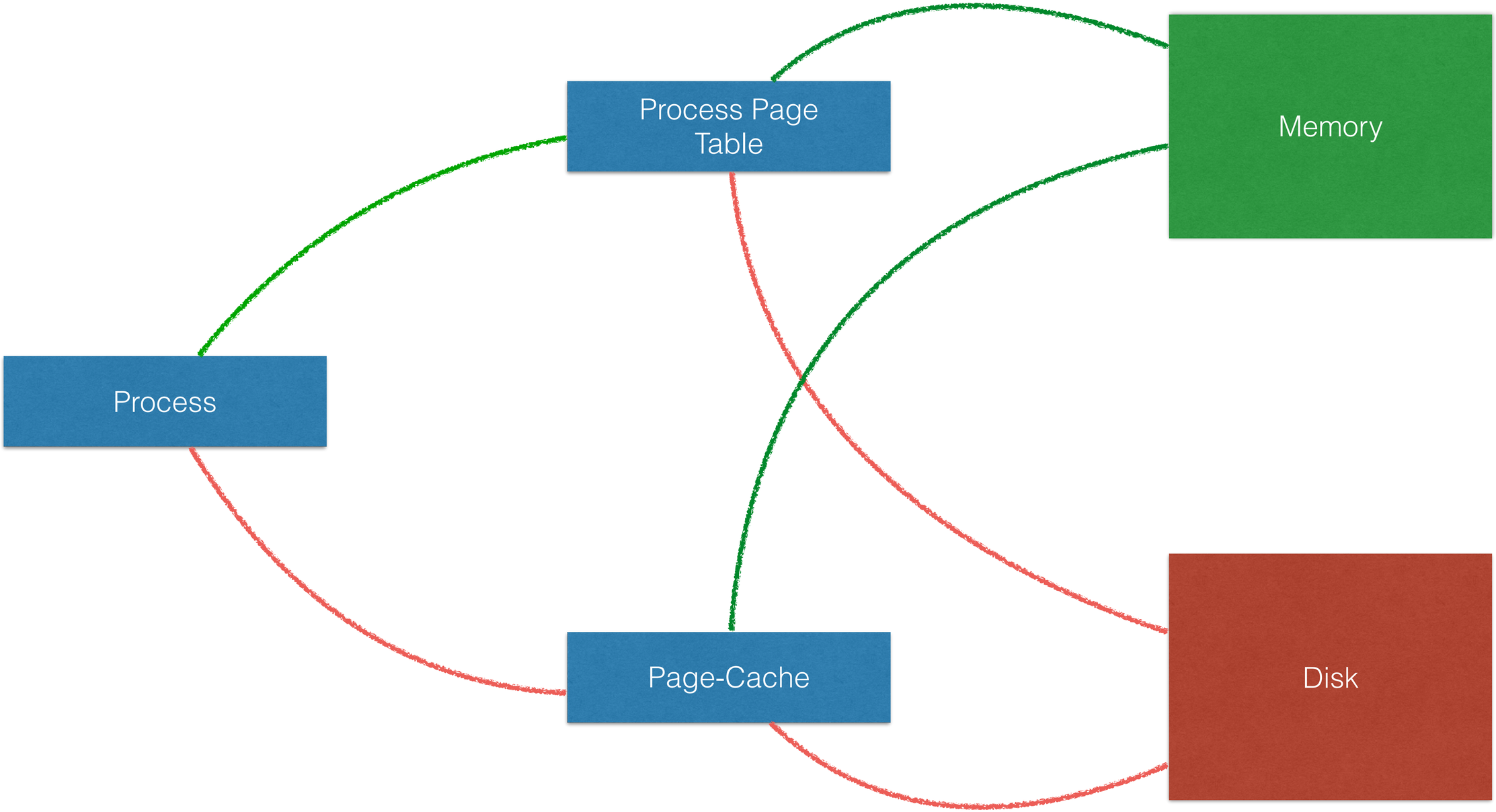
Is it faster to read from a HashMap or a file?

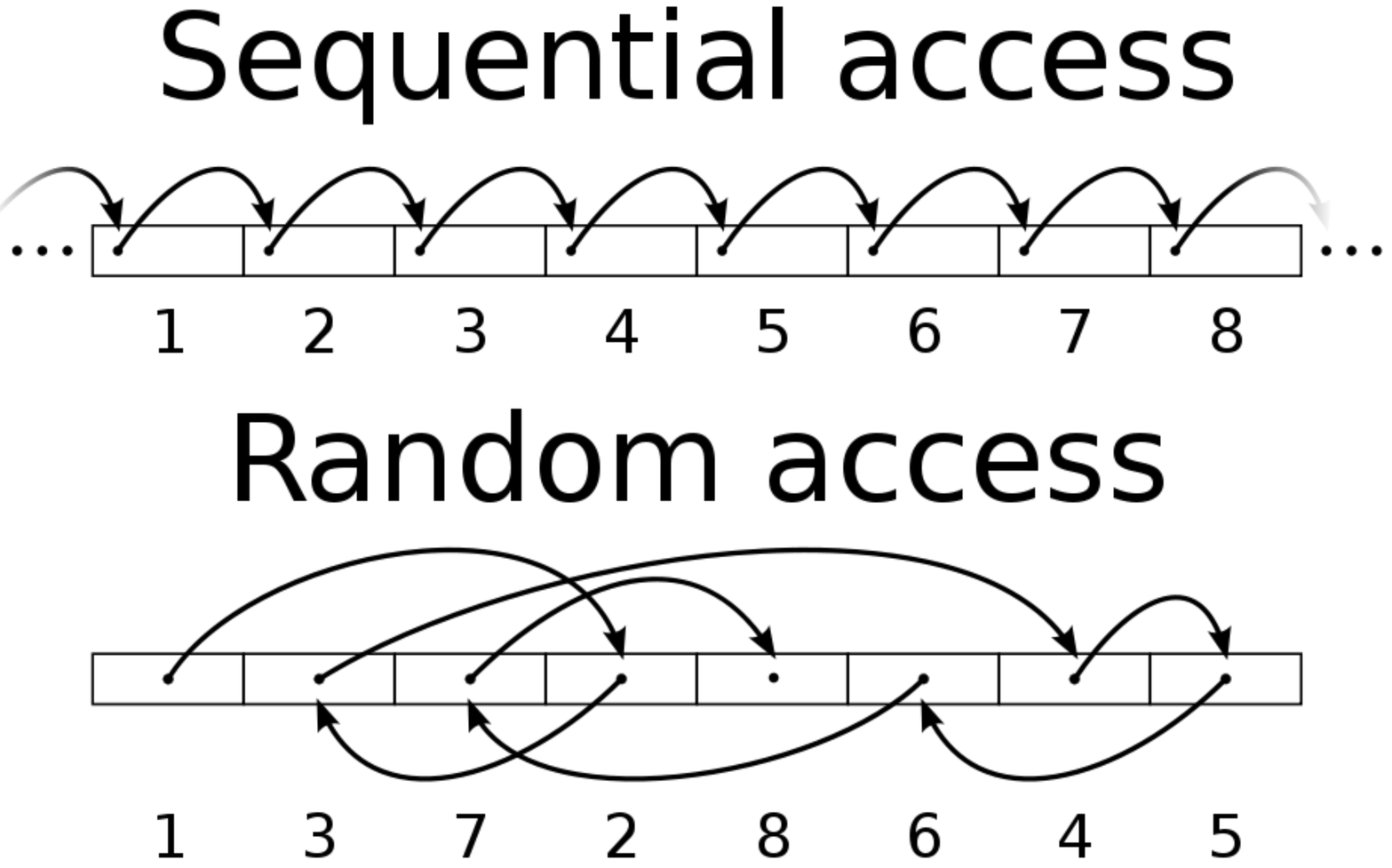
Resident vs. Virtual Memory

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



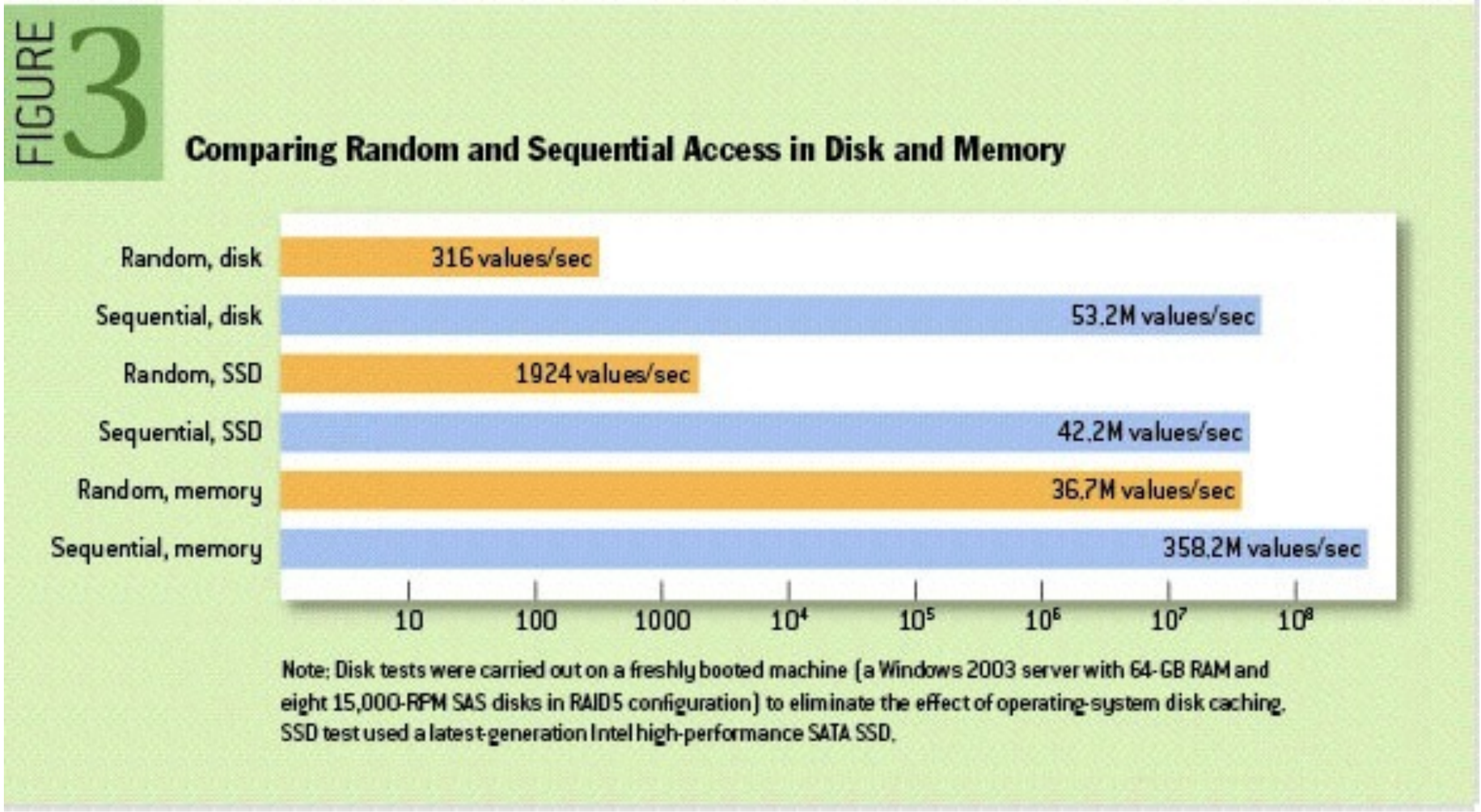
"Resident evil ver4" by impawards. Licensed under Fair use of copyrighted material in the context of Resident Evil (film) via Wikipedia - http://en.wikipedia.org/wiki/File:Resident_evil_ver4.jpg#mediaviewer/File:Resident_evil_ver4.jpg


Process Page Table
Memory
Process
Page-Cache
Disk



"Random vs sequential access" by (User:Intgr) - Own work. Licensed under Public Domain via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Random_vs_sequential_access.svg#mediaviewer/File:Random_vs_sequential_access.svg

Access Pattern Matters!



The Pathologies of Big Data - http://queue.acm.org/detail.cfm?id=1563874

So What?

- Benchmarking? Use relevantly sized data sets and access patterns
- Avoid random access if you can
- Avoid using more memory than available
- Performance counters(perf/likwid…):
  - Cache misses (L1/L2/LLC)
  - Page faults
- Disable swap? (Set `swappiness=0`)
- Consider priming memory?

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

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

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();
}
```

Compressed OOPs

-XX:+UseCompressedOops

- Want large heaps (> 4Gb) and 32bit addresses
- Objects aligned to 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 -> Read Barrier

- Must be decompressed before read ‘through’
- Can be copied without decompression
- Can be compared without decompression

LVB - Zing Read Barrier

- Looks like this

```
test8 rax,[gc_phase_trap_mask]; GC phase changed?
jnz 0x500d639b; GOTO LVB cold path
```

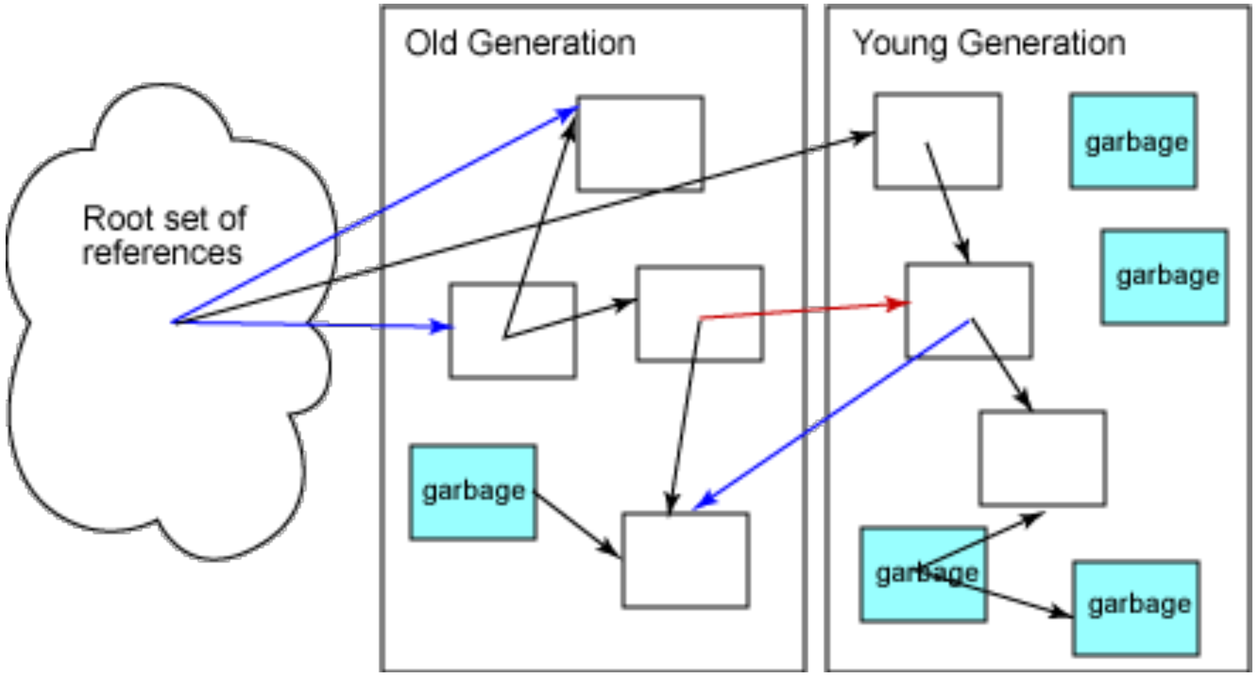
- Cold path: value has relocated
  - Mutator will fix up the loaded value
  - ‘Self healing’ - mutator participates in relocation
- Read more:

http://www.azulsystems.com/sites/default/files/images/c4_paper_acm.pdf

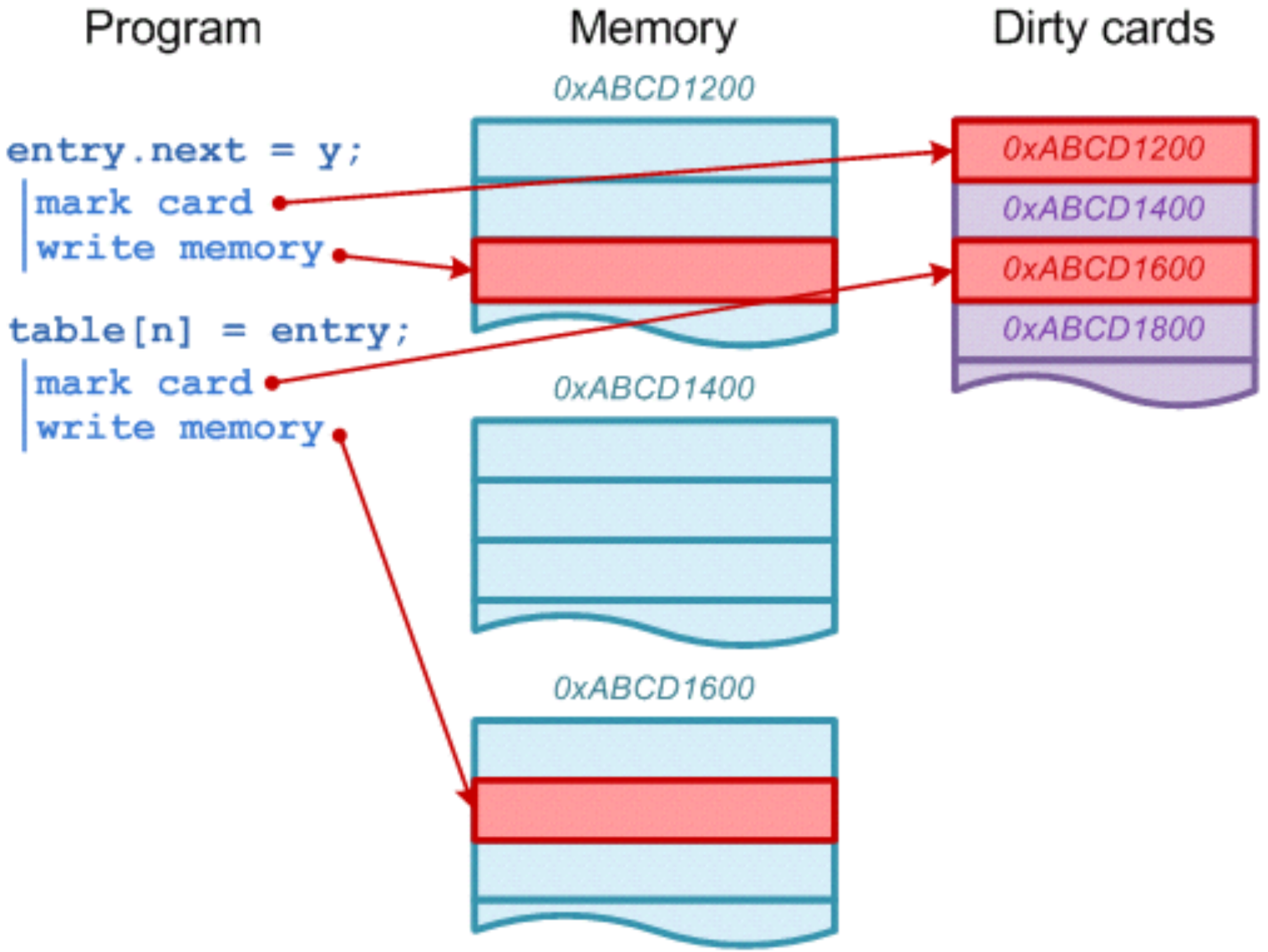
http://www.javaworld.com/article/2078661

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

Card Marking is a Write Barrier

- 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?)

The JVM…

… Must give us pause?

Why Stop The World?

- Some GC phases
- Deoptimization
- Stack trace dump
- Lock un-biasing
- Class redefinition

Can one Java thread stop all other Java threads?

You too can trigger a STW pause!

- On normal allocation (Young Gen exhausted*)
- On large object allocation (Old Gen exhausted*)
- On synchronized block (unbiasing*)
- Profiler sampling (JVMTI *GetStackTrace*()*)
- Hitting cold code (deoptimization**)

```
*    Not true for Zing
**   Less true for Zing
```

Zing: The pause-**less** JVM

- C4 - Fully Concurrent GC (including young gen)
- ReadyNow! - Fighting Deopts
- No biased locking
- … some STW pauses exist (smaller than OS hiccups)

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

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

Safepoint

(noun.)

- A thread state
  - `Waiting/Idle/Blocked -> @Safepoint`
  - `Running/Ready Java -> !@Safepoint`
  - `Running/Ready native -> @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
http://chriskirk.blogspot.ru/2013/09/what-is-java-safepoint.html

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`

Where do we see a Safepoint poll?

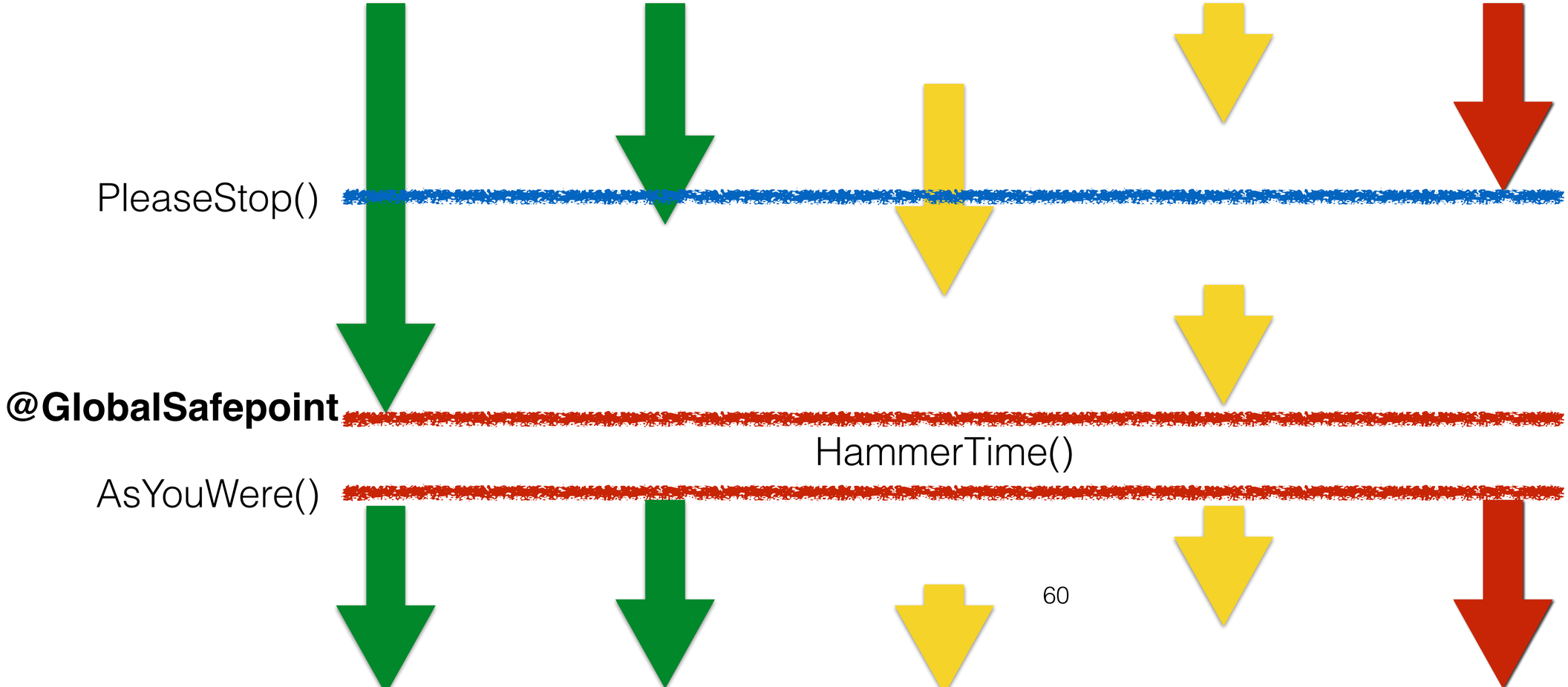
- While loop back edge
- For loop over longs back edge (for(long l=0;…))
- Method exit

How long was this GC pause?

```
8109.128: [GC [PSYoungGen: 109884K-
>14201K(139904K)] 691015K->595332K(1119040K),
0.0454530 secs]
```

TTSP - Time To Safepoint

- Not included in GC Time (use -XX:+PrintGCApplicationStoppedTime)
- Max Actual Pause = TTSP + pause time(in safepoint)



What can cause long TTSP?

- Inlining removes end of method safe point poll
- Long counted loops* (i.e: for(int i=0;i<MAX_INT;i++) doSomething();)
- Large memory copies (System.arrayCopy**/Unsafe.copyMemory)
- Interrupted threads
- Page Faults

* On Zing can use: -XX+KeepSafepointsInCountedLoops
** On Zing safepoint polls are introduced between copying chunks

So what?

- Use -XX:+PrintGCApplicationStoppedTime
- TTSP can dominate stop the world times
- Beware Safepoint biased profilers
- Chunk large memory copies

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