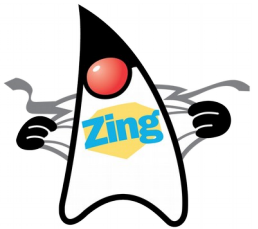
Profilers Are Lying Hobbitses

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

Thanks!

I work on Zing!



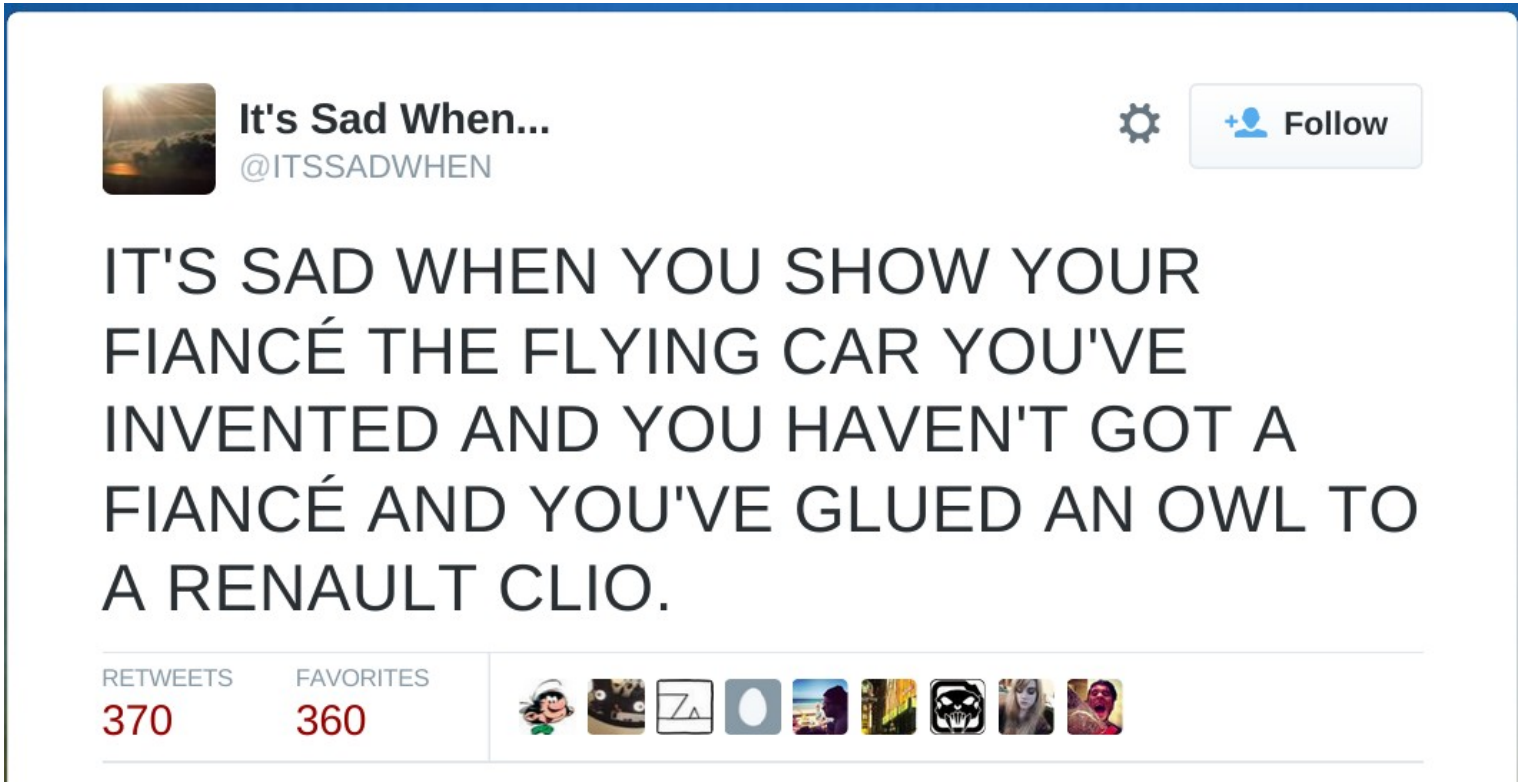
- Awesome JVM
- Only on Linux/x86
- Aimed at server side systems
- Highly focused on responsiveness
  - ✔ C4 – Fully concurrent GC
  - ✔ ReadyNow! - Persisted profile data

But Also:

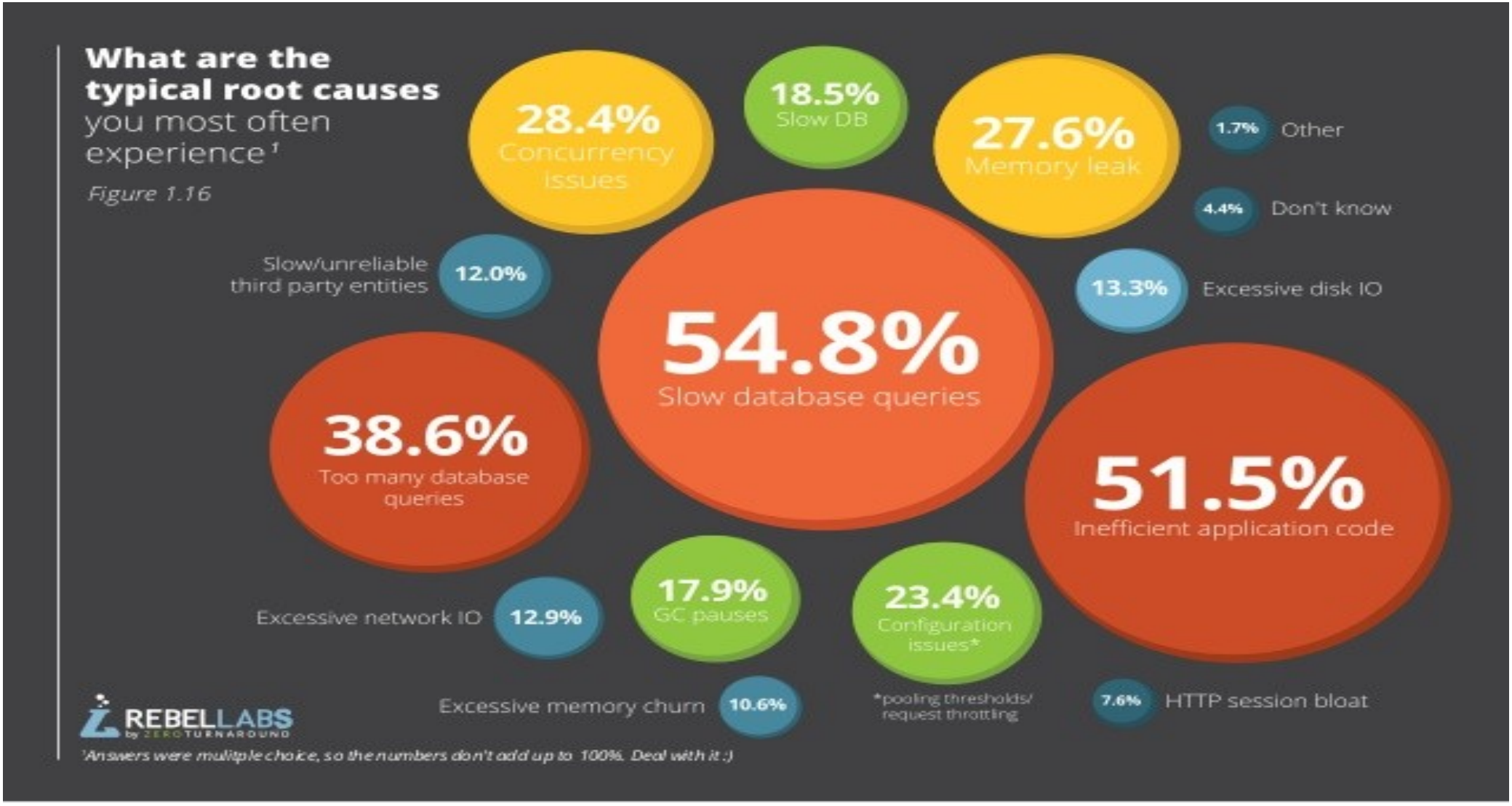
- Blog: http://psy-lob-saw.blogspot.com
- Open Source developer/contributor:
  - JCTools
  - Aeron/Agrona
  - Netty/Akka/RxJava/YCSB/HdrHistogram
  - Honest-Profiler/perf-map-agent
- Cape Town Java Meetup Organizer

Why profile?

Also:



https://twitter.com/ITSSADWHEN/status/645557218851557376


What are the typical root causes you most often experience[1]
Figure 1.16
28.4% Concurrency issues
18.5% Slow DB
27.6% Memory leak
1.7% Other
4.4% Don't know
Slow/unreliable third party entities 12.0%
54.8% Slow database queries
13.3% Excessive disk IO
38.6% Too many database queries
51.5% Inefficient application code
Excessive network IO 12.9%
17.9% GC pauses
23.4% Configuration issues*
Excessive memory churn 10.6%
*pooling thresholds/ request throttling
7.6% HTTP session bloat
REBELLABS by ZEROTURNAROUND
[1]Answers were multiple choice, so the numbers don't add up to 100%. Deal with it :)

Which profiler?

Which tools do you use for application profiling?

Figure 1.12

REBELLABS
by ZEROTURNAROUND

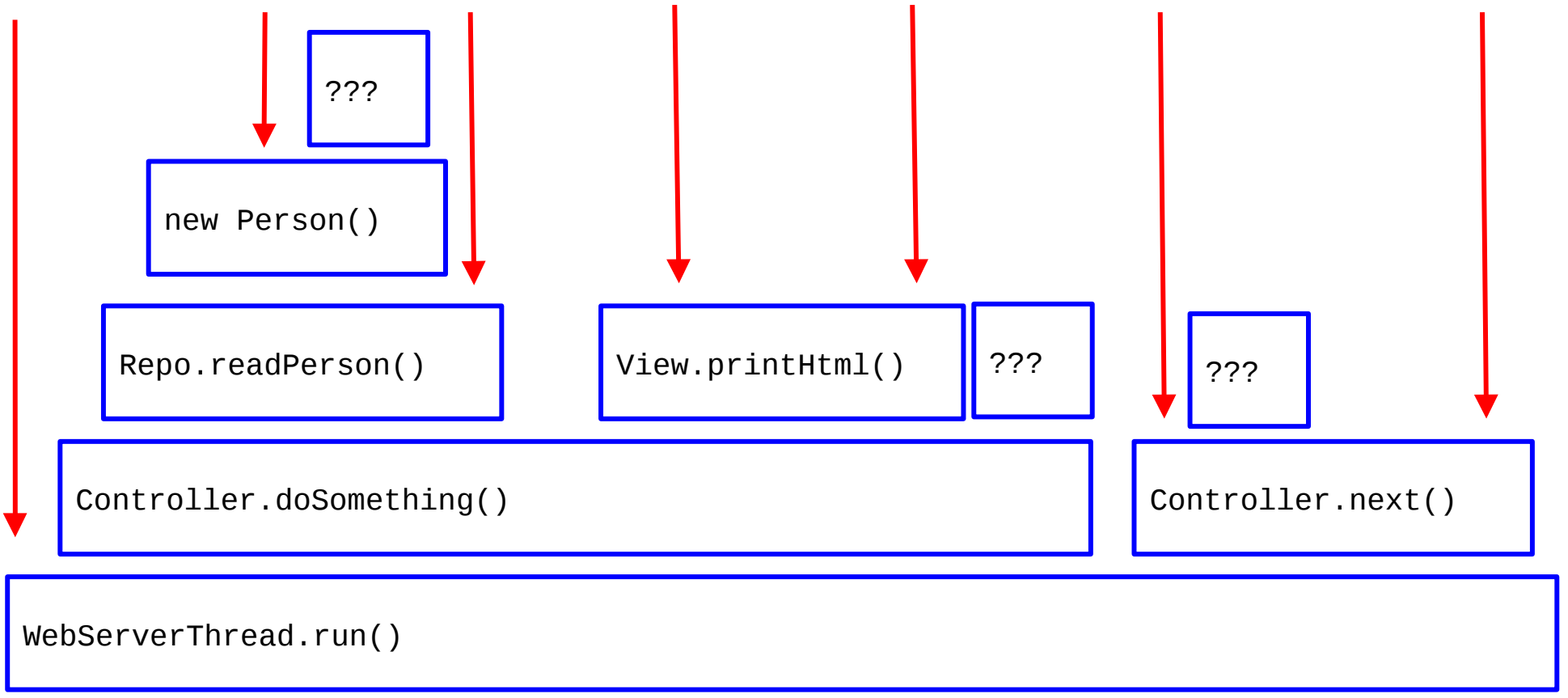
LIVE DEMO TIME!!!!!

Sampling Profilers

- Sample program on interval
- Distribution of samples highlights hotspots
- **Assumption**: *Samples are 'random'*
- **Assumption**: Sample distribution approximates 'Time Spent' distribution

Sampling?

Sampling Profilers



Not enough samples

Solution: Switch to tracing profiler

Solution: Shorter sampling interval

Solution: Patience

Sampling interval matching application life cycle

Solution: Shorter interval

Solution: Randomized interval

Sample taking is expensive

Solution: Switch sampling method

Solution: Accept overhead

Solution: Longer interval

Sample is biased/inaccurate

Solution: Switch sampling method

Solution: Widen your scope

Problems with JVisualVM*?

- Reports all threads (running or not)
- Uses **GetStackTrace****:
  - High overhead
  - **Safepoint**** Biased

* And all other JVMTI::GetStackTrace based profilers
** Will be explained shortly…

GetStackTrace: the official API

- Input: Thread
- Output:
  - Error code (failure IS an option)
  - List of frames (jmethodId, jlocation)

https://docs.oracle.com/javase/8/docs/platform/jvmti/jvmti.html#GetStackTrace

jlocation, where J-Lo be at?

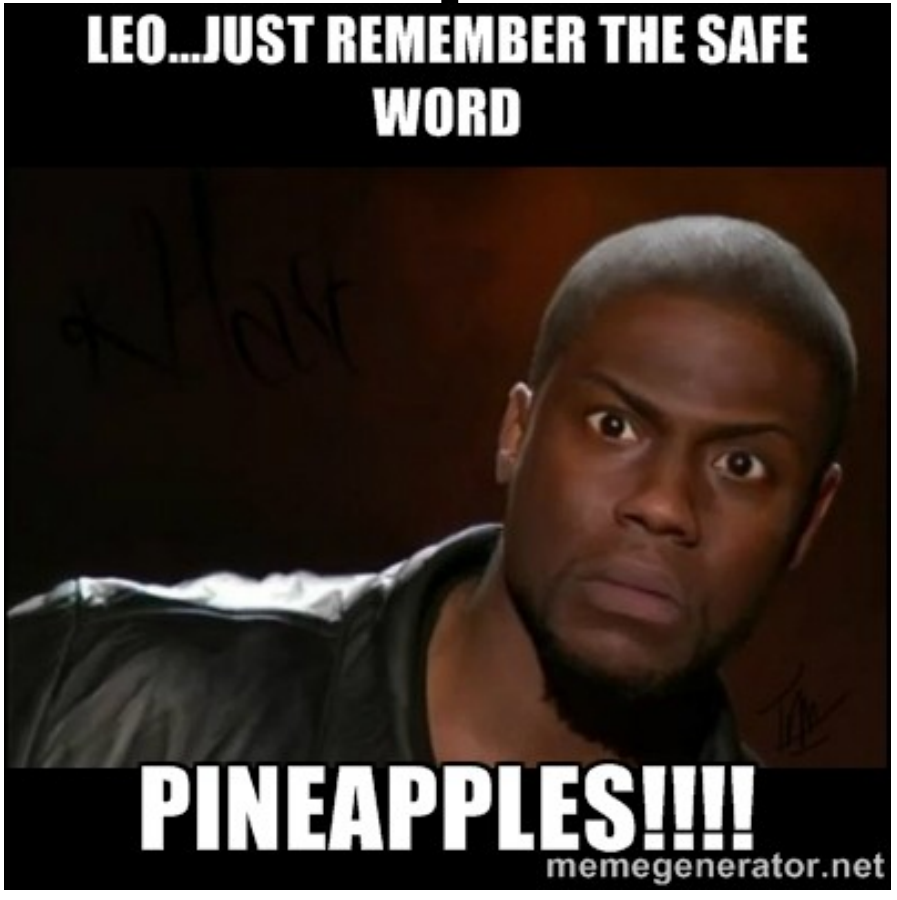
BCI → Line of Code

- BCI – Byte Code Index
- Not every BCI has a line of code
- Find the closest...

Look in hprof for example: <OPENJDK-HOME>/demo/jvmti/hprof

GetStackTrace samples at a Safepoint

Safepoint?


LEO...JUST REMEMBER THE SAFE WORD
PINEAPPLES!!!!
memegenerator.net

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

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”

Gil Tene, on “Mechanical Sympathy” mailing list:
https://groups.google.com/d/msg/mechanical-sympathy/GGByLdAzlPw/cF1_XW1AbpEJ

Why bring threads to Safepoint?

- 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

How does a JVM bring a thread to a 'Safepoint'?

1) Raise **Safepoint** ‘flag’
2) Wait for thread to **poll Safepoint** 'flag'
3) Thread transitions to **Safepoint** state

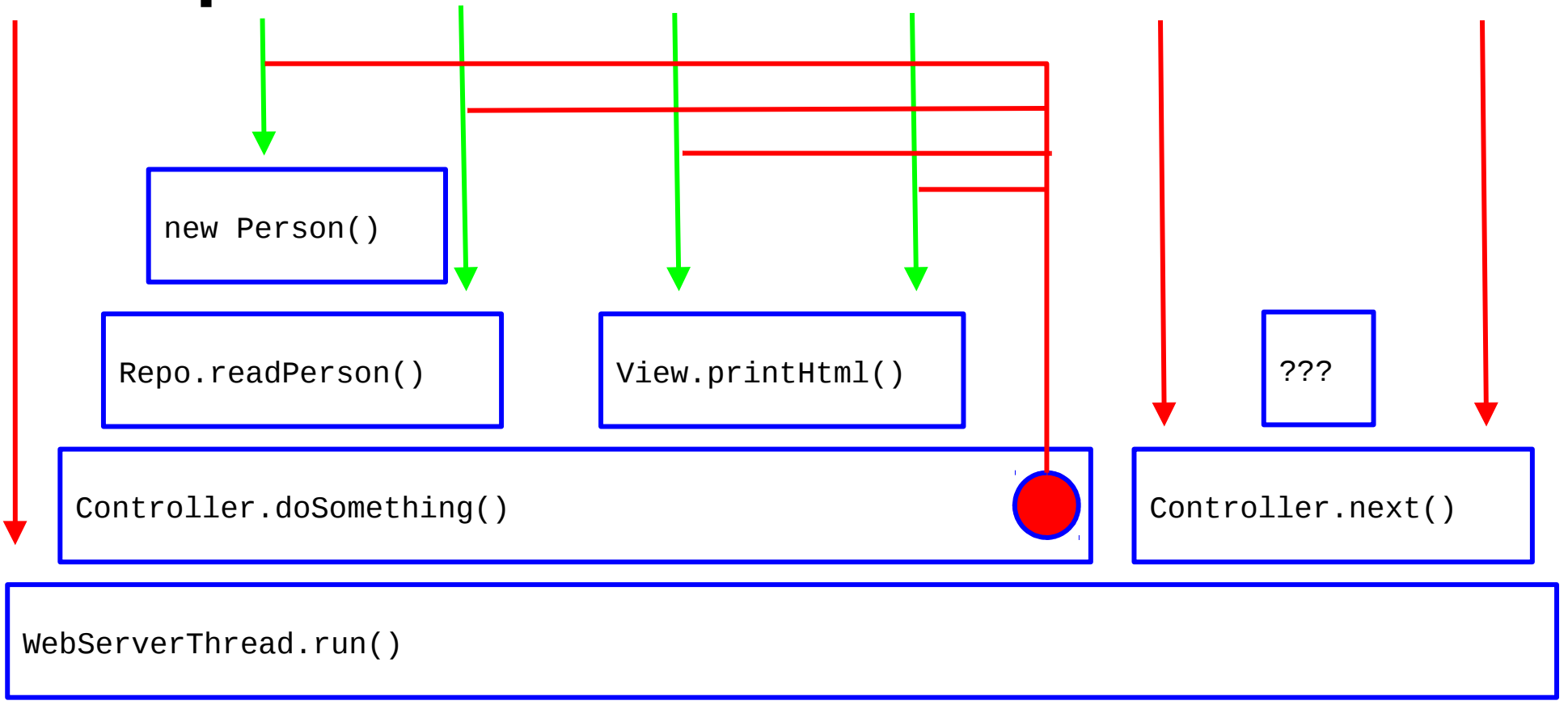
Where do we see a Safepoint poll?

- 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();
}
```

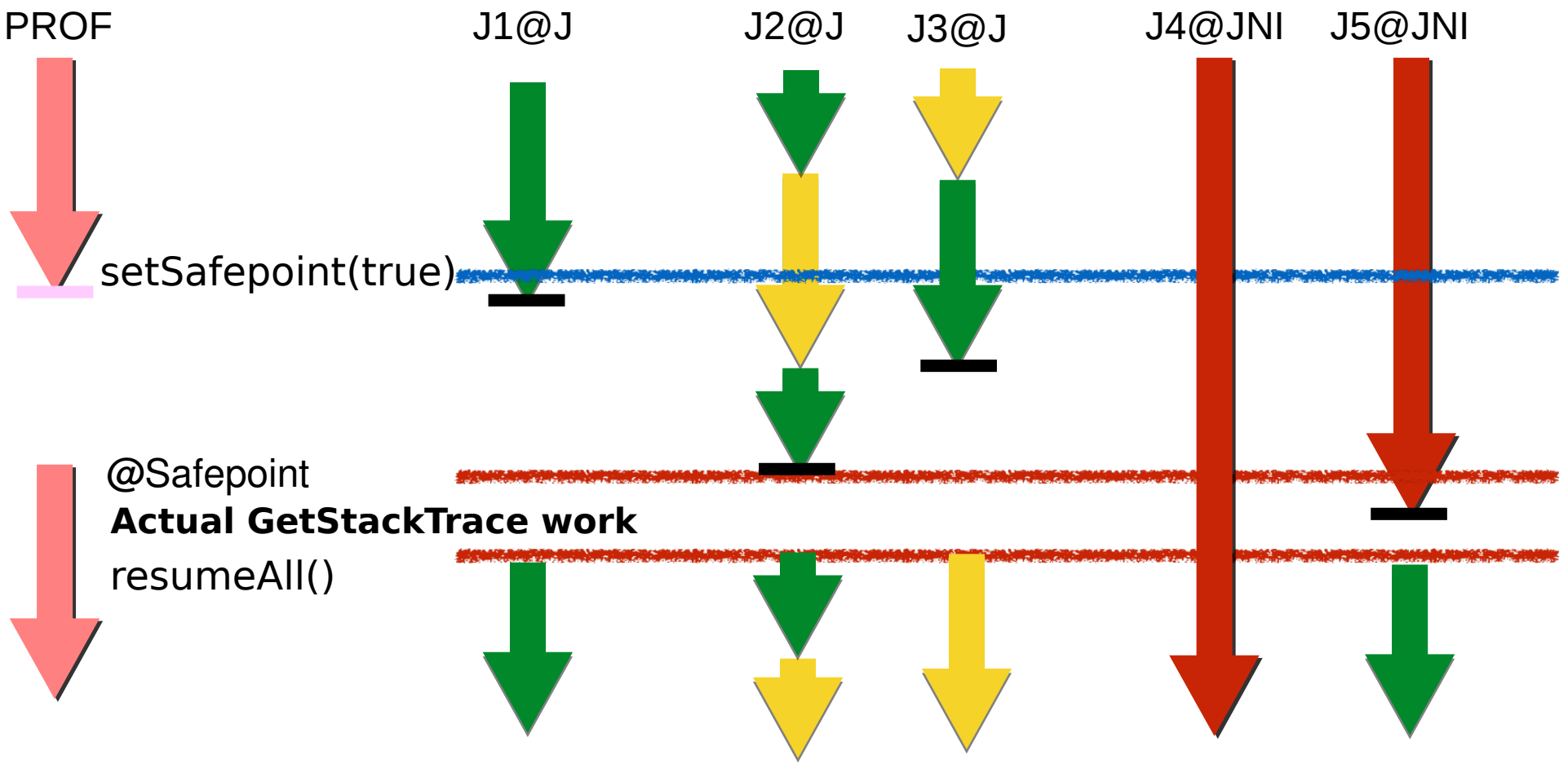
```
public void foo(Bar bar) {
  int nogCount = 0;
  for (int i = 0; i < 10; i++) {
    if (bar.getZog(i).isNog()) nogCount++;
  }
  while (nightIsYoung) {
    nogCount += hit(bar);
    // Safepoint poll
  }
  if (nogCount > MAX_NOG)
    throw new NogOverflowError();
  // Safepoint poll
}
```

Safepoint Bias



It's just a harmless lil' safepoint they said

GetStackTrace Overheads



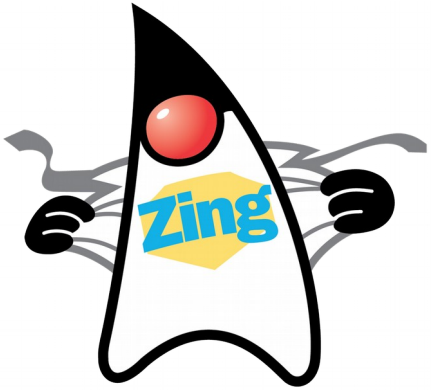
GetStackTrace overhead (OpenJDK)

- Stop ALL Java threads
- Collect single/all thread call traces
- Resume ALL stopped threads

Use -XX:+PrintGCApplicationStoppedTime to log pause times

GetStackTrace overhead (**Zing**)

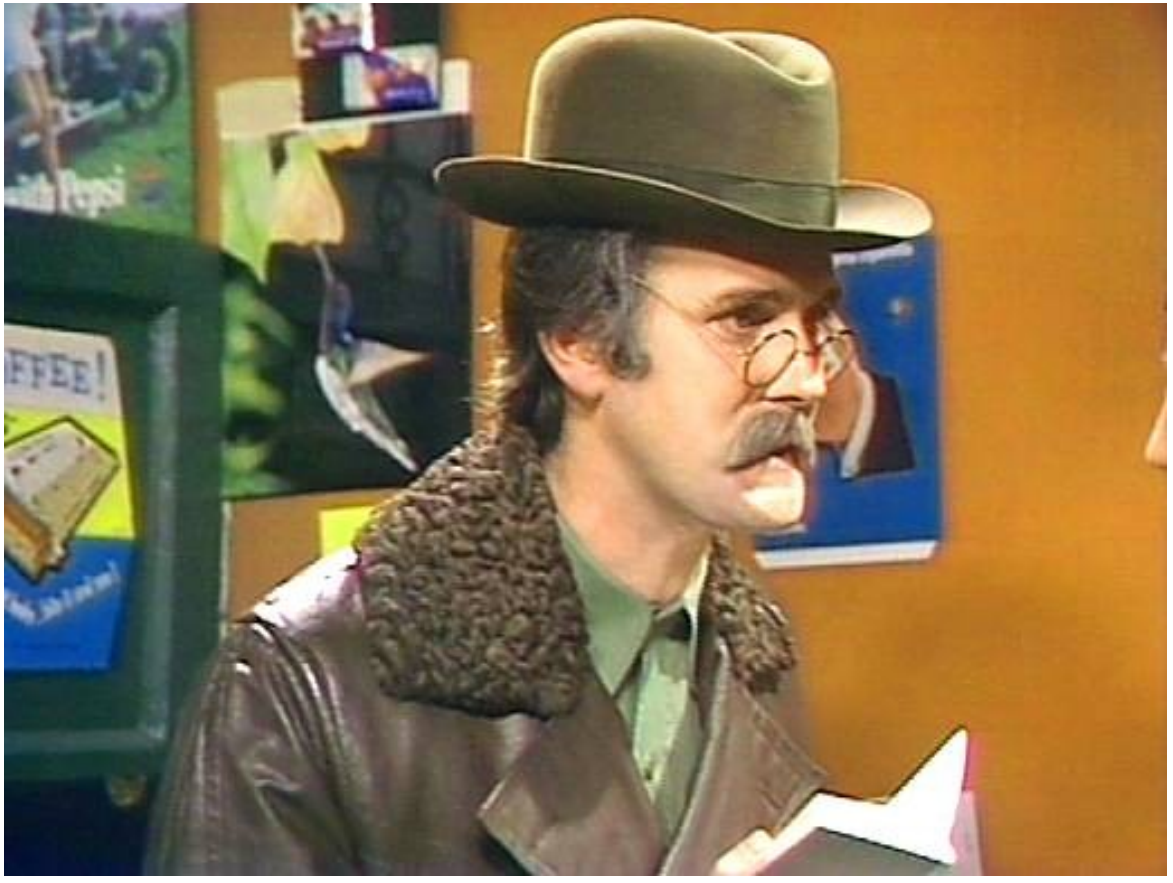
- Stop sampled Java thread
- Collect single thread call trace
- Resume stopped thread



LIVE DEMO TIME!!!!!

GetStackTrace demo points

- Use -XX:+PrintGCApplicationStoppedTime
- Safepoint location is 'arbitrary'
- Overhead scales with number of threads
- Widen scope up the call tree?



I will not buy this RECORD, it is SCRATCHED!!!!

AsyncGetCallTrace: unofficial API

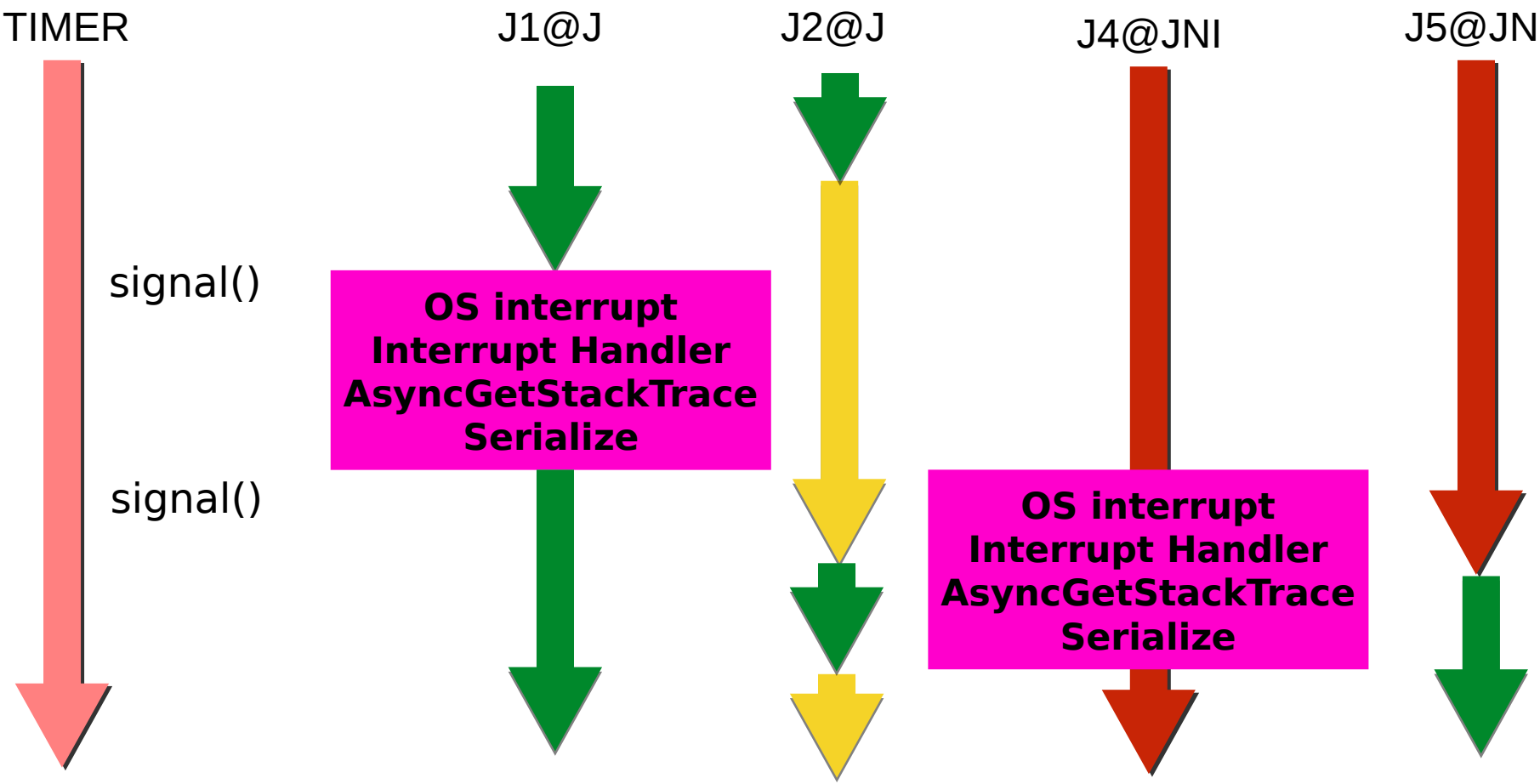
- Input: signal context and JNI env
  - Context will provide PC/FP/SP
- Output:
  - Error code (failure IS an option)
  - List of frames (jmethodId, lineno)
  - lineno == BCI

Why Use AsyncGetCallTrace?

- Built for sampling in **signal handler**
- Does not require a safepoint
- Samples the **interrupted** thread
- Interrupted thread need not be at safepoint

http://jeremymanson.blogspot.co.za/2007/05/profiling-with-jvmtijvmpi-sigprof-and.html
http://jeremymanson.blogspot.co.za/2013/07/lightweight-asynchronous-sampling.html

AsyncGetCallTrace sequence



Who Uses AsyncGetCallTrace?

- Solaris Studio (but not only AGCT...)
- Java Flight Recorder
- Lightweight-Java-Profiler
- Honest-Profiler

LIVE DEMO TIME!!!!!

AGCT demo points

- Use: -XX:+UnlockDiagnosticVMOptions -XX:+DebugNonSafepoints
- Only Java stack is covered
- Only on CPU is sampled
- Lookout for failed samples

Oh?
You want the truth?


YOU CAN'T HANDLE the TRUTH!!!
mongertravels.com

Reality is complex...

- There is no Line Of Code
- There's no BCI
- Only instructions
- And more than just Java

Stack Frame → Call Trace Frame

- Stack frame:
  - PC – program counter
  - FP – frame pointer (optional)
  - SP – stack pointer
- Call trace frame:
  - jmethodid
  - BCI

PC → BCI

- PC points to the 'current' instruction
- Not every instruction has a BCI
- Find the closest...

Funny Thing About PCs...

“> I think Andi mentioned this to me last year --
> that instruction profiling was no longer reliable.

`It never was.`”

http://permalink.gmane.org/gmane.linux.kernel.perf.user/1948
Exchange between Brenden Gregg and Andi Kleen

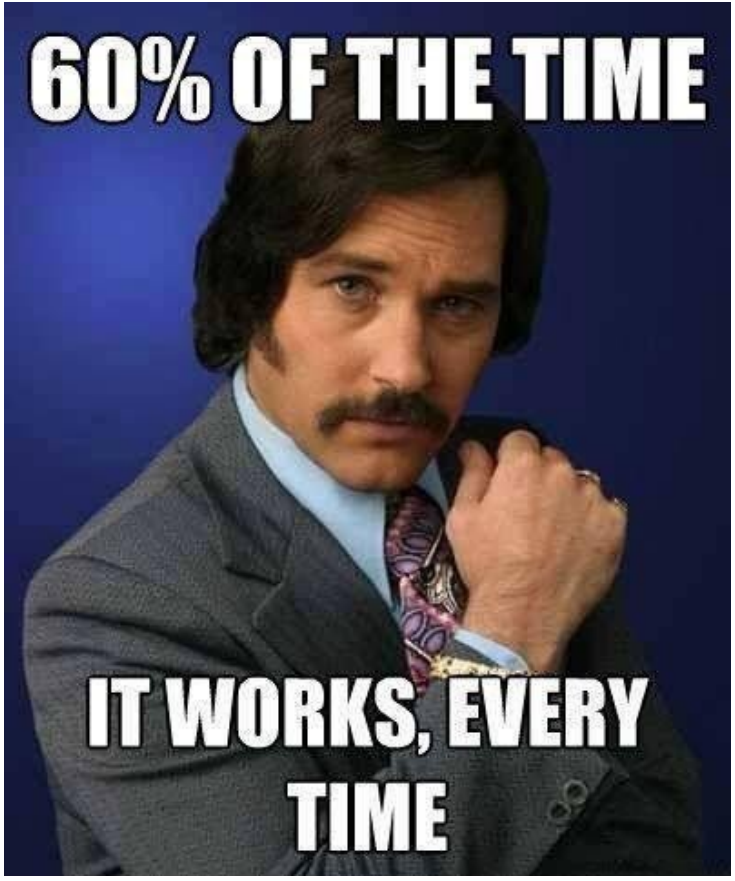
Skid

- Super Scalar CPU
- Speculative execution
- Signal latency

The blamed instruction is often shortly after where the big cost lies

PC → BCI → Line of Code

- This is as good as it gets
- Mostly it's good enough
- Look for other suspects nearby



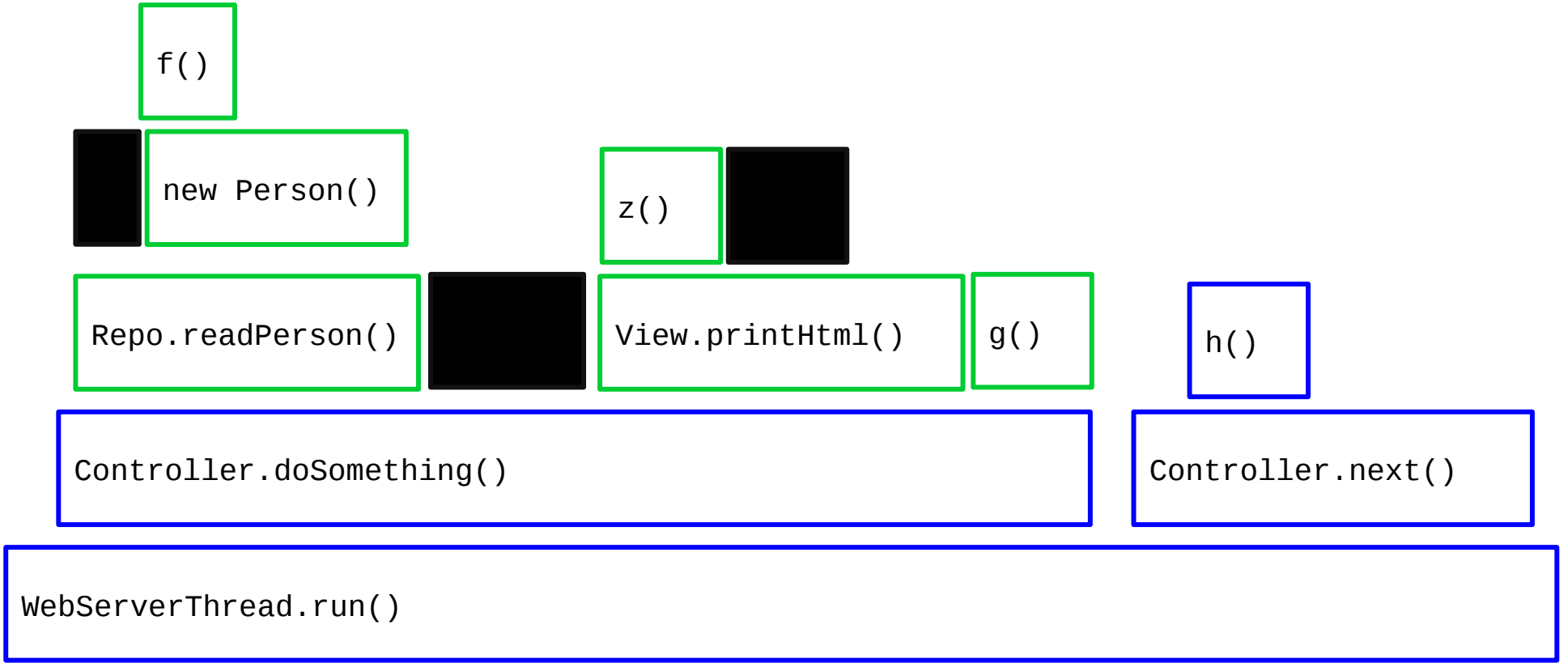
Nearby? Nearby where?

LIVE DEMO TIME!!!!!

Perf-map-agent demo points

- Use: -XX:+UnlockDiagnosticVMOptions -XX:+DebugNonSafepoints
- No LOC info (fixable)
- Only on CPU is sampled
- Opportunity to differentiate virtual/real frames

Inlining JIT Compilation



Take Aways

- Know your profiler
- There's no perfect profiler
- Try an 'unbiased' profiler, give honest-profiler/perf-map-agent a go!