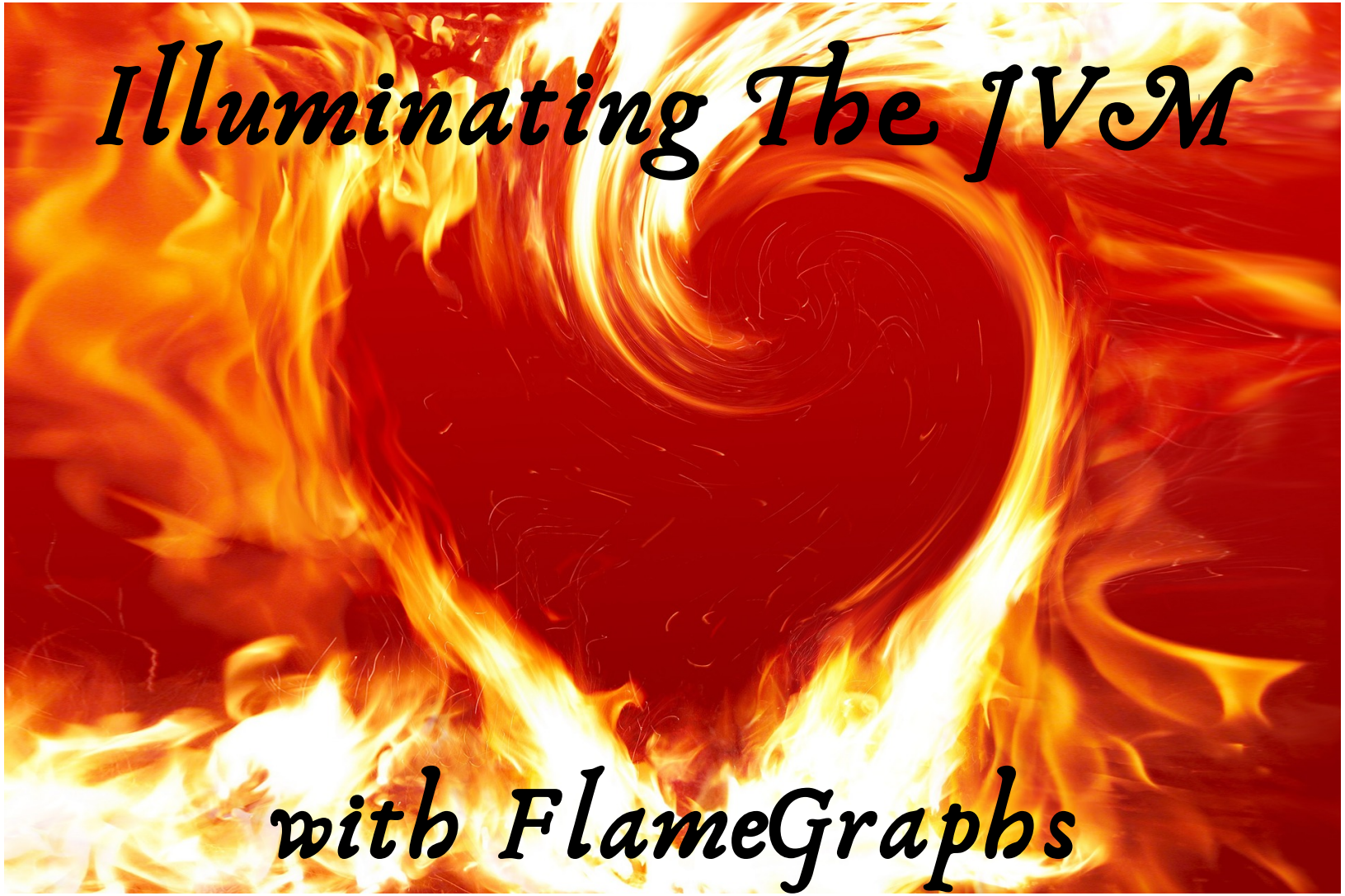
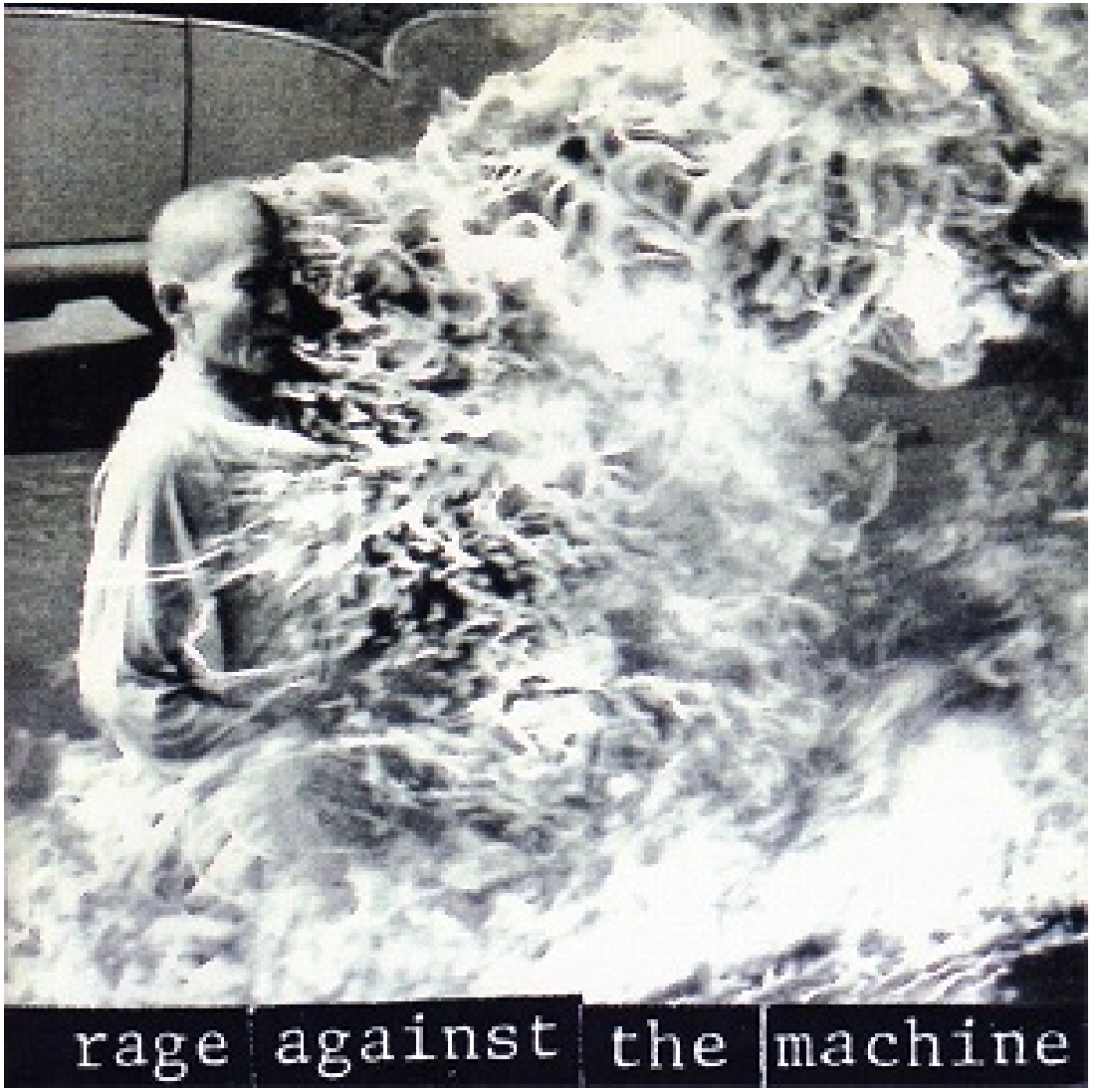



Nitsan Wakart (@nitsanw)

# *Illuminating The JVM with FlameGraphs*

Nitsan Wakart (@nitsanw)




By Source, Fair use, https://en.wikipedia.org/w/index.php?curid=196363

# Thanks!

# I, Programmer

- Performance Engineer
- Blog: http://psy-lob-saw.blogspot.com
- Open Source developer/contributor:
  - JCTools
  - Aeron/Agrona
  - Honest-Profiler/perf-map-agent
- Cape Town JUG Organizer

# What is the ROOT of ALL EVIL?

We should forget about **small efficiencies**, say about 97% of the time: **premature** optimization is the root of all evil.

– Donald Knuth

# Solution?

- Get requirements
- Measure!
- **Profile!**
- Measure!

- ✓ Java?
- ✗ FlameGraphs?
- ✗ Perf?

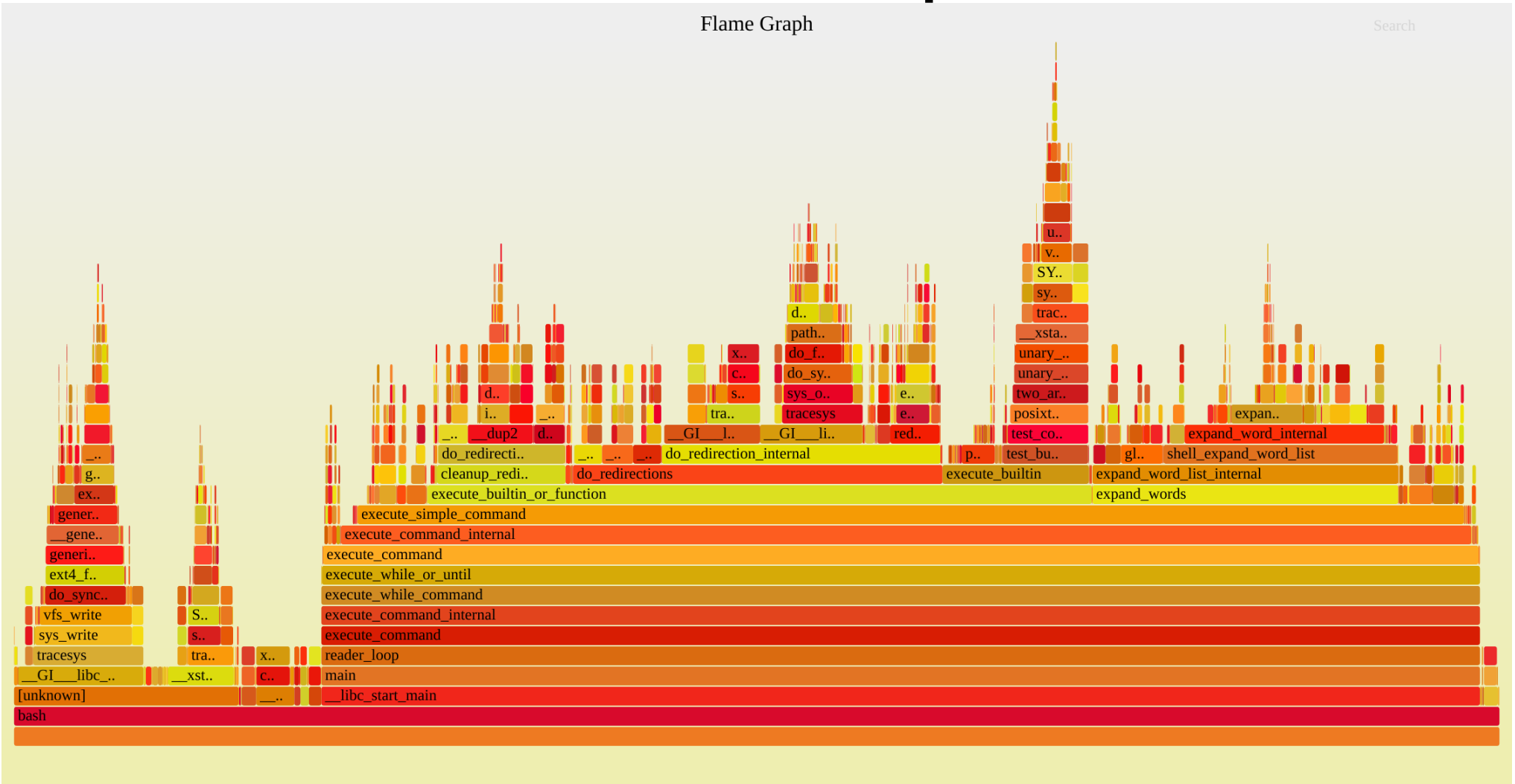
- Brendan Gregg, Netflix
- Super performance dude
- Invented FlameGraphs:

  http://queue.acm.org/detail.cfm?id=2927301

“Flame graphs are a visualization of profiled software, allowing the most frequent code-paths to be identified quickly and accurately.”

- see: http://www.brendangregg.com/flamegraphs.html
- git clone https://github.com/brendangregg/FlameGraph.git

# FlameGraph


Flame Graph
Search
bash
__libc_start_main
main
reader_loop
execute_command
execute_command_internal
execute_while_command
execute_while_or_until
execute_simple_command
execute_command_internal
execute_builtin_or_function
execute_builtin
expand_words
expand_word_list_internal
shell_expand_word_list
expand_word_internal
do_redirections
do_redirection_internal
cleanup_redi..
do_redirecti..
__libc_start_main
__GI___libc_..
[unknown]
tracesys
sys_write
vfs_write
do_sync..
ext4_f..
generi..
__gene..
gener..
ex..
g..
_..
__dup2
__GI___l..
__GI___li..
tracesys
sys_o..
do_sy..
do_f..
path..
d..
x..
c..
s..
d..
i..
tra..
x..
c..
__xst..
S..
s..
tra..
__..
_..
_..
_..
_..
red..
e..
e..
p..
test_co..
test_bu..
posixt..
two_ar..
unary_..
unary_..
__xsta..
trac..
sy..
SY..
v..
u..
gl..
expan..

# Input: Sampling Profilers

- Collect stacks
- X samples per second
- Present data
  - Flat view
  - Tree view
  - FlameGraph

# Flat View

**Hot Methods**

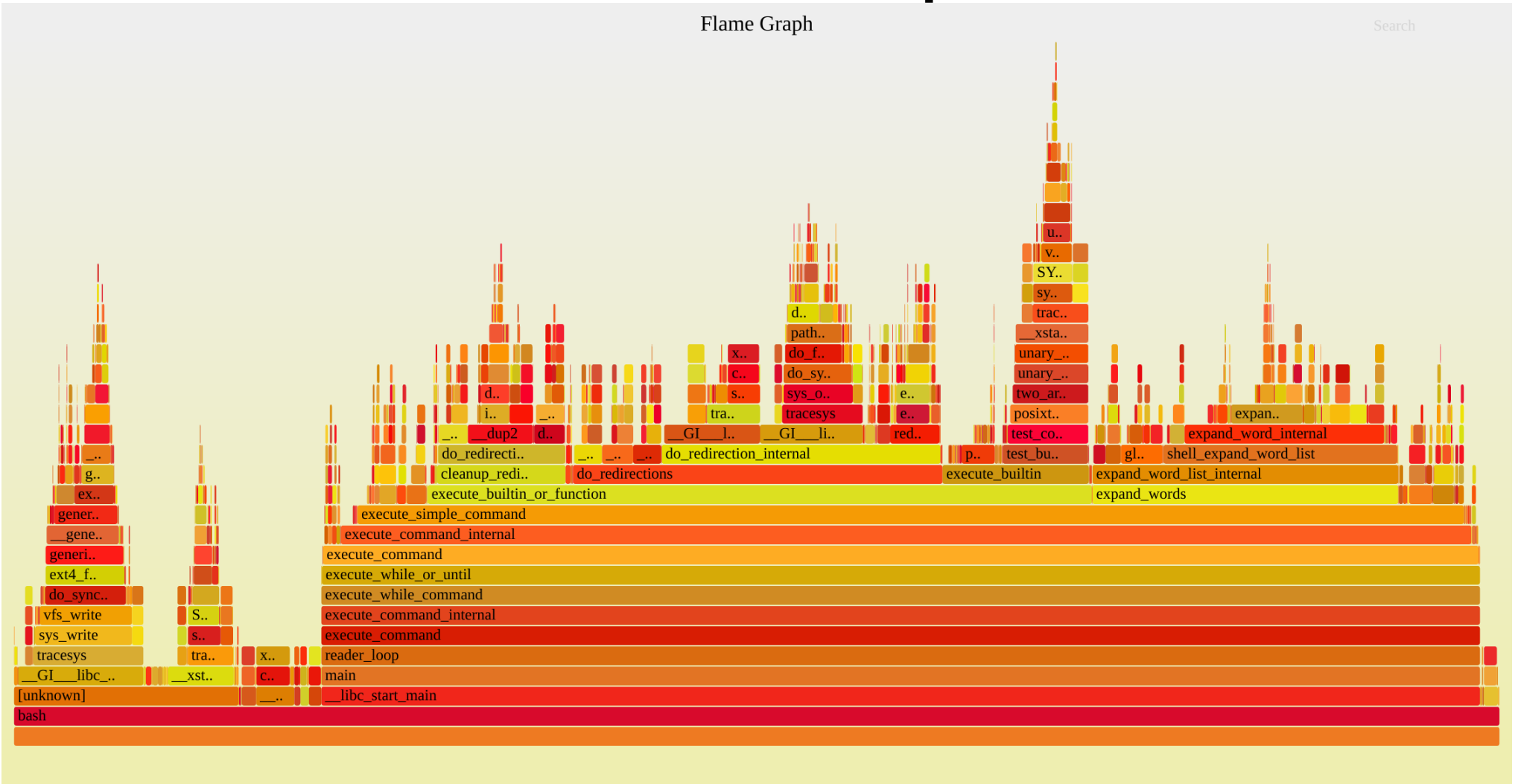
Filter Column: Stack Trace

| Stack Trace | Sample Count | Percentage |
|---|---|---|
| io.netty.handler.codec.DefaultHeaders.<init>(HashingStrategy, ValueConverter, | 19 | 4.33% |
| java.lang.Thread.blockedOn(Interruptible) | 12 | 2.73% |
| io.netty.util.ResourceLeakDetector.reportLeak(ResourceLeakDetector$Level) | 10 | 2.28% |
| sun.nio.ch.IOUtil.writeFromNativeBuffer(FileDescriptor, ByteBuffer, long, Nat | 9 | 2.05% |
| java.nio.Buffer.position(int) | 8 | 1.82% |
| io.netty.handler.codec.http.HttpObjectDecoder$HeaderParser.process(byte) | 8 | 1.82% |
| io.netty.util.ReferenceCountUtil.touch(Object, Object) | 8 | 1.82% |
| io.netty.channel.ChannelOutboundBuffer.nioBuffers() | 7 | 1.59% |
| io.netty.handler.codec.http.HttpHeadersEncoder.encoderHeader(CharSequence, Ch | 7 | 1.59% |
| sun.nio.ch.SocketChannelImpl.read(ByteBuffer) | 6 | 1.37% |
| io.netty.handler.codec.http.DefaultHttpHeaders.validateHeaderNameElement(byte | 6 | 1.37% |
| io.netty.util.internal.InternalThreadLocalMap.get() | 6 | 1.37% |
| sun.nio.ch.IOUtil.write(FileDescriptor, ByteBuffer, long, NativeDispatcher) | 6 | 1.37% |
| io.netty.buffer.WrappedByteBuf.writerIndex() | 6 | 1.37% |
| io.netty.buffer.AbstractByteBuf.forEachByteAsc0(int, int, ByteProcessor) | 5 | 1.14% |
| io.netty.channel.AbstractChannelHandlerContext.invokeChannelReadComplete() | 5 | 1.14% |

# Tree View

| Stack Trace | Sample Count | Percentage |
|---|---|---|
| java.lang.Thread.run() | 432 | 98.41% |
| io.netty.util.concurrent.SingleThreadEventExecutor$5.run() | 404 | 92.03% |
| io.netty.channel.nio.NioEventLoop.run() | 404 | 92.03% |
| io.netty.channel.nio.NioEventLoop.processSelectedKeys() | 384 | 87.47% |
| io.netty.channel.nio.NioEventLoop.processSelectedKeysOptimized(SelectionKey[]) | 384 | 87.47% |
| io.netty.channel.nio.NioEventLoop.processSelectedKey(SelectionKey, AbstractNioChannel) | 384 | 87.47% |
| io.netty.channel.nio.AbstractNioByteChannel$NioByteUnsafe.read() | 384 | 87.47% |
| io.netty.channel.DefaultChannelPipeline.fireChannelRead(Object) | 264 | 60.14% |
| io.netty.channel.AbstractChannelHandlerContext.invokeChannelRead(AbstractChannelHandlerCont | 264 | 60.14% |
| io.netty.channel.AbstractChannelHandlerContext.invokeChannelRead(Object) | 264 | 60.14% |
| io.netty.channel.DefaultChannelPipeline$HeadContext.channelRead(ChannelHandlerContext, Ob | 264 | 60.14% |
| io.netty.channel.AbstractChannelHandlerContext.fireChannelRead(Object) | 264 | 60.14% |
| io.netty.channel.AbstractChannelHandlerContext.invokeChannelRead(AbstractChannelHandlerC | 263 | 59.91% |
| io.netty.channel.AbstractChannelHandlerContext.invokeChannelRead(Object) | 261 | 59.45% |
| io.netty.channel.CombinedChannelDuplexHandler.channelRead(ChannelHandlerContext, Objec | 260 | 59.23% |
| io.netty.handler.codec.ByteToMessageDecoder.channelRead(ChannelHandlerContext, Object) | 260 | 59.23% |
| io.netty.handler.codec.ByteToMessageDecoder.fireChannelRead(ChannelHandlerContext, Co | 193 | 43.96% |
| io.netty.channel.CombinedChannelDuplexHandler$DelegatingChannelHandlerContext.fireCh | 192 | 43.74% |
| io.netty.channel.AbstractChannelHandlerContext.fireChannelRead(Object) | 192 | 43.74% |
| io.netty.channel.AbstractChannelHandlerContext.invokeChannelRead(AbstractChannelHan | 192 | 43.74% |
| io.netty.channel.AbstractChannelHandlerContext.invokeChannelRead(Object) | 186 | 42.37% |
| io.netty.microbench.http_req_res.HttpServer$HttpServerHandler.channelRead(ChannelH | 183 | 41.69% |
| io.netty.channel.AbstractChannelHandlerContext.write(Object) | 103 | 23.46% |
| io.netty.channel.AbstractChannelHandlerContext.write(Object, ChannelPromise) | 103 | 23.46% |
| io.netty.channel.AbstractChannelHandlerContext.write(Object, boolean, ChannelPro | 103 | 23.46% |
| io.netty.channel.AbstractChannelHandlerContext.invokeWrite(Object, ChannelPromi | 103 | 23.46% |
| io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(Object, ChannelPro | 103 | 23.46% |
| io.netty.channel.CombinedChannelDuplexHandler.write(ChannelHandlerContext, Obj | 103 | 23.46% |
| io.netty.handler.codec.MessageToMessageEncoder.write(ChannelHandlerContext, C | 103 | 23.46% |

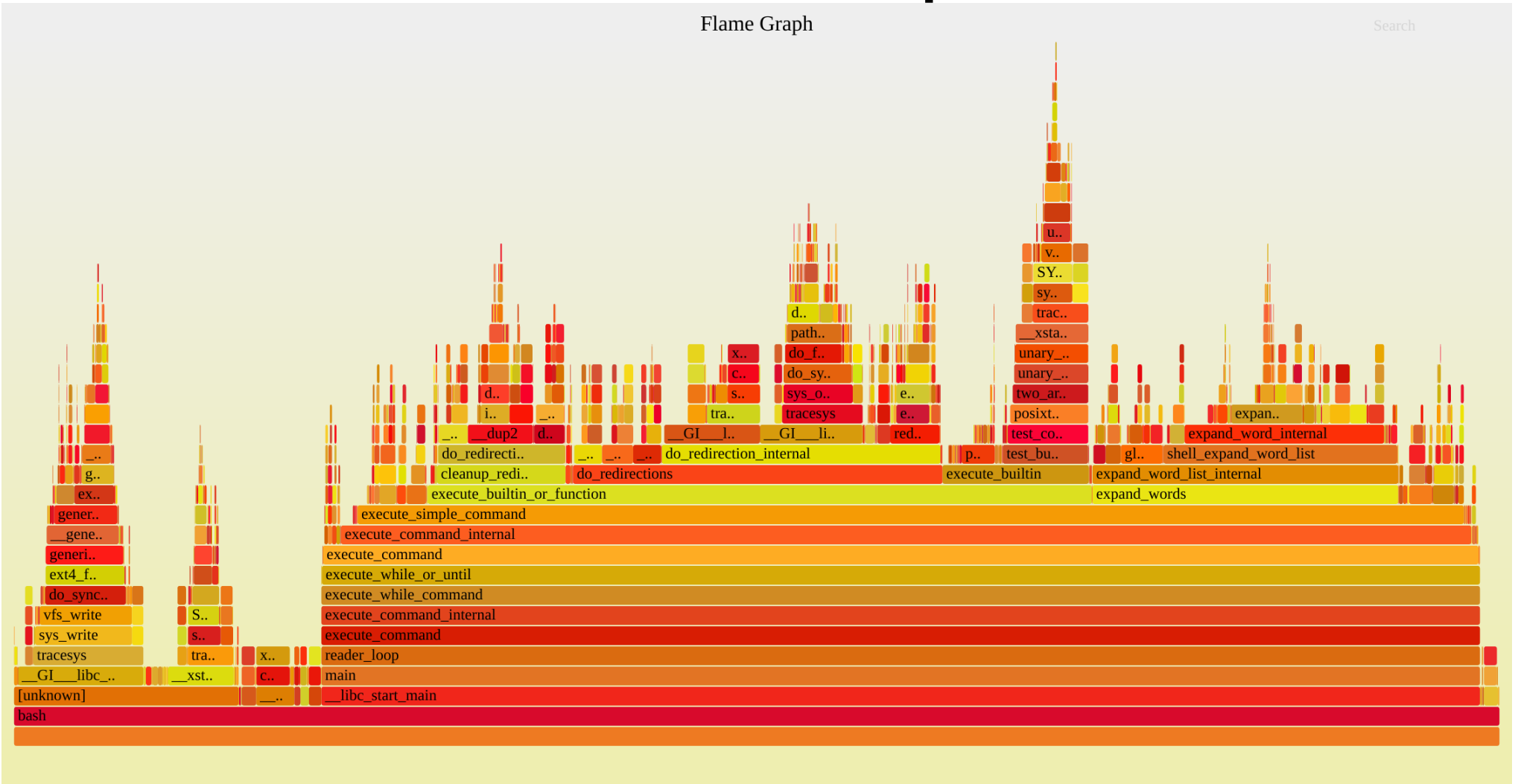
# FlameGraph


Flame Graph
Search
bash
__libc_start_main
main
reader_loop
execute_command
execute_command_internal
execute_while_command
execute_while_or_until
execute_simple_command
execute_command_internal
execute_builtin_or_function
execute_builtin
expand_words
expand_word_list_internal
shell_expand_word_list
expand_word_internal
do_redirections
do_redirection_internal
cleanup_redi..
do_redirecti..
__libc_start_main
__GI___libc_..
[unknown]
tracesys
sys_write
vfs_write
do_sync..
ext4_f..
generi..
__gene..
gener..
ex..
g..
_..
__dup2
__GI___l..
__GI___li..
tracesys
sys_o..
do_sy..
do_f..
path..
d..
x..
c..
s..
d..
i..
tra..
x..
c..
__xst..
S..
s..
tra..
__..
_..
_..
_..
_..
red..
e..
e..
p..
test_co..
test_bu..
posixt..
two_ar..
unary_..
unary_..
__xsta..
trac..
sy..
SY..
v..
u..
gl..
expan..

# How Can I Get One?

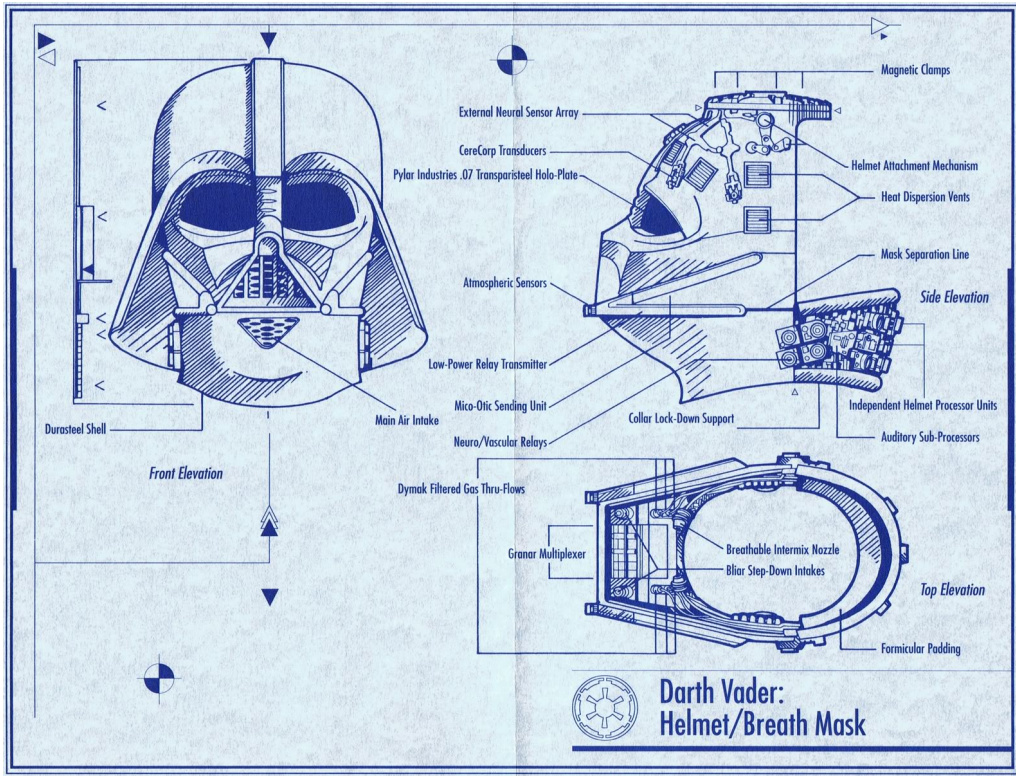
- Profiler => stack traces (e.g. a JFR file or hprof file)
- Stack traces => ./stackcollapse.pl -> collapsed stacks
  - Text transformation => **HACKABLE!**
- Collapsed stacks -> ./flamegraph.pl -> SVG
  - Text transformation => **SUPER HACKABLE!**

# FlameGraph


Flame Graph
Search
bash
__libc_start_main
main
reader_loop
execute_command
execute_command_internal
execute_while_command
execute_while_or_until
execute_simple_command
execute_command_internal
execute_builtin_or_function
execute_builtin
expand_words
expand_word_list_internal
shell_expand_word_list
expand_word_internal
do_redirections
do_redirection_internal
cleanup_redi..
do_redirecti..
__libc_start_main
__GI___libc_..
[unknown]
tracesys
sys_write
vfs_write
do_sync..
ext4_f..
generi..
__gene..
gener..
ex..
g..
_..
__dup2
__GI___l..
__GI___li..
tracesys
sys_o..
do_sy..
do_f..
path..
d..
x..
c..
s..
d..
i..
tra..
x..
c..
__xst..
S..
s..
tra..
__..
_..
_..
_..
_..
red..
e..
e..
p..
test_co..
test_bu..
posixt..
two_ar..
unary_..
unary_..
__xsta..
trac..
sy..
SY..
v..
u..
gl..
expan..

# Enjoying Your New Helmet!

- Y-Axis: Stack depth
  - Top methods are the leaf methods
  - Bottom methods are roots of the stack (e.g. Thread::run)
- X-Axis: Profile populations sorted alphabetically
  - Wider frames == more samples == where ‘time’ is spent
  - Roots are wide, callees get narrower, tops are thin spikes



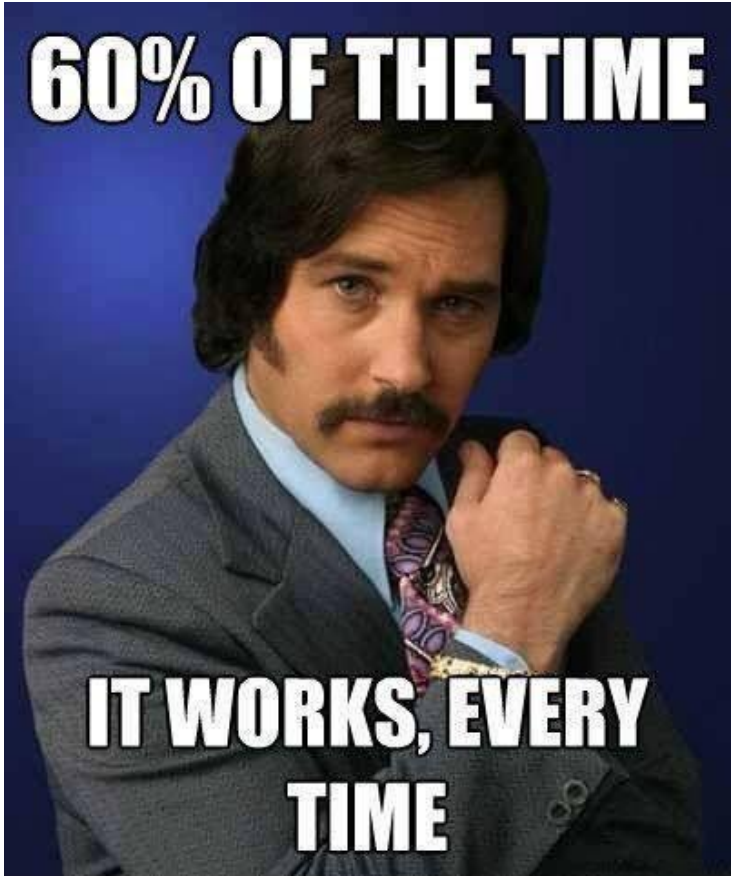
SWITCH TO BROWSER

SVGs In Slides suck...

What can we feed to the flames?

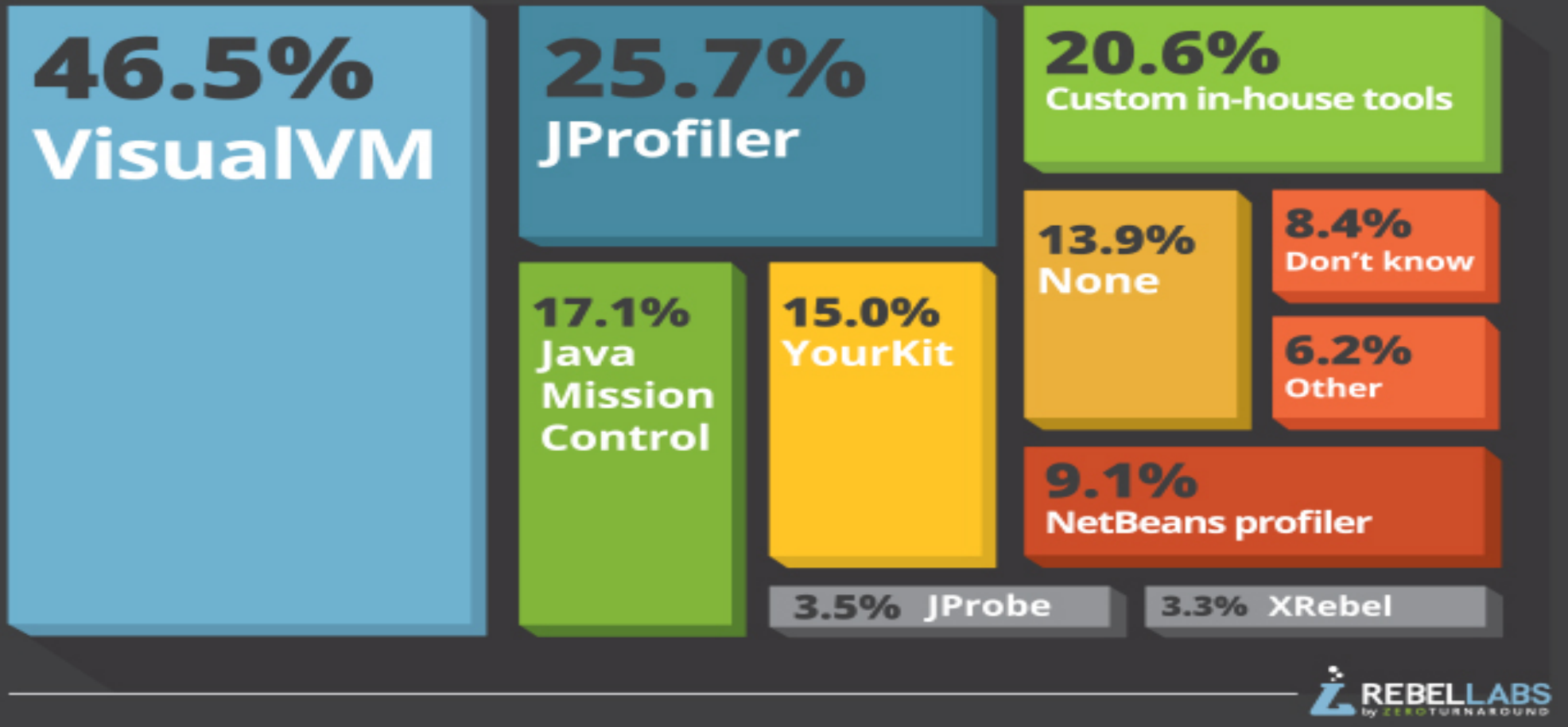
# Java Profilers (typically) Care About

- Only Java Code
- Only some of the time



**Which tools do you use** for application profiling?

*Figure 1.12*




REBELLABS
by ZEROTURNAROUND

# JVisualVM & co: Safepoint Bias

- Samples only at **safepoint polls**
- Each sample is a **safepoint operation**
- Each sample includes **all** threads
- ALWAYS AVAILABLE!
- Supported FlameGraph scripts:
  - ./stackcollapse-jstack.pl
  - ./stackcollapse-hprof.pl

http://psy-lob-saw.blogspot.co.uk/2016/02/why-most-sampling-java-profilers-are.html

# JMC/Honest-Profiler: one eyed kings

- No safepoint bias!
- Java stack only
- Blind spots: GC/Deopt/Runtime stubs
- OpenJDK/Oracle(1.6+ HP/1.7u40+ JFR) + recent Zing
- Custom stack collapse tools exist:
  - FlameGraphDumperApplication
  - https://github.com/chrishantha/jfr-flame-graph

http://psy-lob-saw.blogspot.co.uk/2016/06/the-pros-and-cons-of-agct.html

# Keeping it REAL

- OS
- JVM runtime (GC/Runtime/Compiler)
- Native libraries
- Your code?
  - Interpreter (cold code)
  - Compiled code (tiered compilation: 1..4)
  - Inlined compiled code

# Linux Perf (perf_events)

- System profiler
- Userspace + Kernel
- Standard tool
- Now works with Java!

https://perf.wiki.kernel.org/index.php/Main_Page

# Perf Profiling Java Credits

- Johannes Rudolph (@virtualvoid)
- Brendan Gregg (@brendangregg)
- OpenJDK Team
- Extras: @nitsanw + @tjake + others!

**!!! OSS FTW !!!**

# Java Perf Profiling

- Linux only
- Oracle/OpenJDK(1.8u60+) + latest Zing
- Need permissions/some Linux fu
- Need perf-map-agent

http://psy-lob-saw.blogspot.co.uk/2017/02/flamegraphs-intro-fire-for-everyone.html

# What Do We Win?

- Java + Native + Kernel stack!
- HW Counters/Events support!
- Low overhead, no safepoint bias

# What Do We Lose?

- Interpreter frames
- Broken stacks (might be fine on Java profilers)
- Limited stack depth (128)
- TOO MUCH INFORMATION!!!

Java Profile Portion

SVGs In Slides suck...

# Java Threads

- Stubs
- Inlining
- Call to native
- Safepoints
- Park costs

# Java Threads HACKAGE BONUS!

- Post process to sharpen
- Trim calls to native
- Collect BROKEN frames

Meta Profile

SVGs In Slides suck...

# JVM Threads

- CPU utilization info
- Internal operation insight
- Confusing blocking behaviour
- Multi-threading pain

# There’s MORE to explore!

- Machine level profile
- Application cluster profile
- Tons of perf features

# An invitation to hack

- Add method self-percent coloring
- Add core utilization indication
- Multi threaded profiles
- Java profile enrichment (e.g. thread names/alloc rate)

# Summary

- New tools for your belt!
- Tweak, hack & share!
- Profile at a wider scope!
- Enjoy :-)