Now You See Me Too

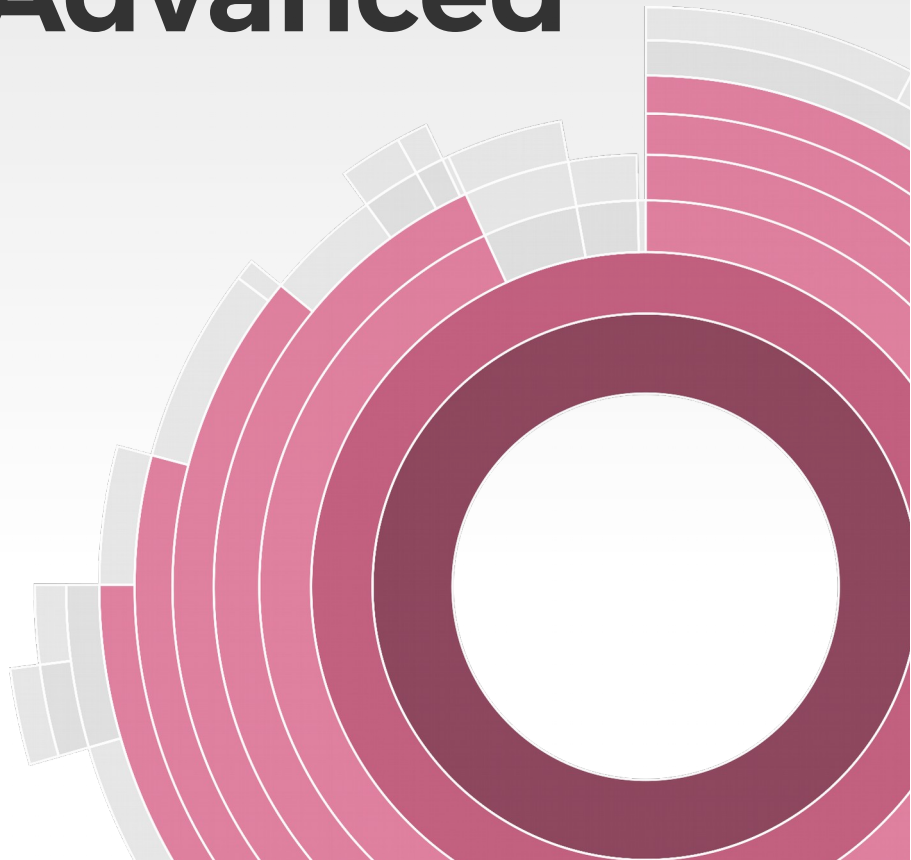
Visual Tooling for Advanced Systems Analysis

Suchakra Sharma

2nd Nov 2017 - San Francisco, CA

usenix LISA17

ShiftLeft



whoami

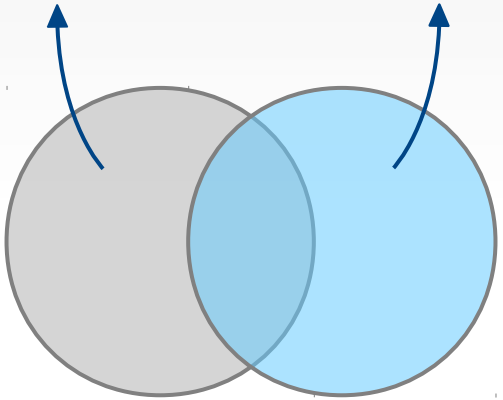
Suchakra

- Staff Scientist, ShiftLeft Inc.
- PhD, DORSAL Lab, *École Polytechnique de Montréal (University of Montréal)*
- Loves tracing, security, performance analysis, low-level hardware stuff, **poutine** and **samosas**
- @tuxology

whoami

Suchakra

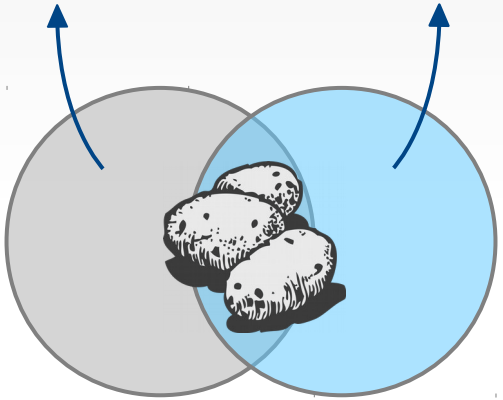
- Staff Scientist, ShiftLeft Inc.
- PhD, DORSAL Lab, *École Polytechnique de Montréal (University of Montréal)*
- Loves tracing, security, performance analysis, low-level hardware stuff, **poutine** and **samosas**
- @tuxology



whoami

Suchakra

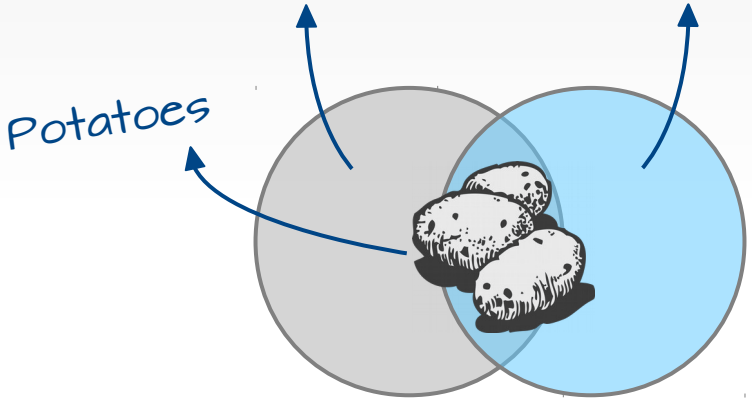
- Staff Scientist, ShiftLeft Inc.
- PhD, DORSAL Lab, *École Polytechnique de Montréal (University of Montréal)*
- Loves tracing, security, performance analysis, low-level hardware stuff, **poutine** and **samosas**
- @tuxology



whoami

Suchakra

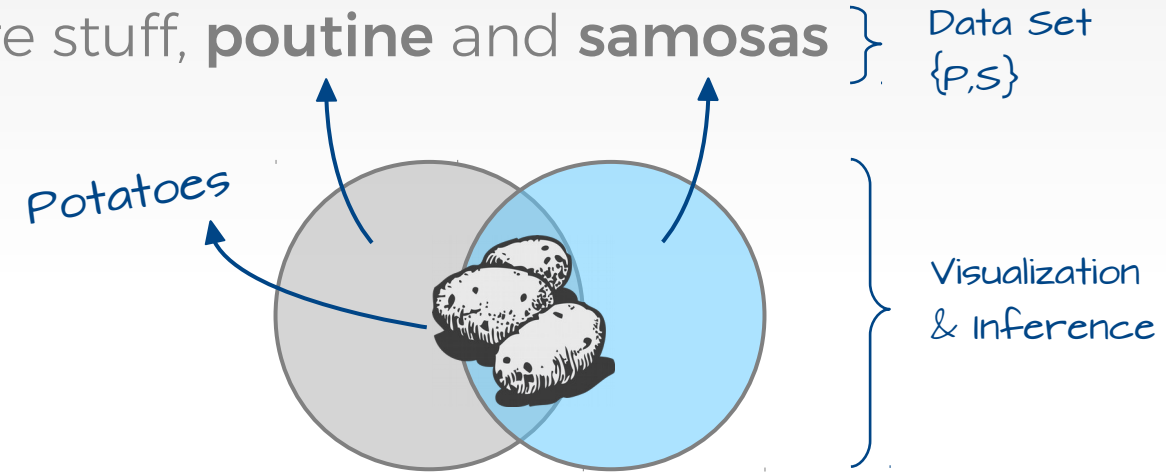
- Staff Scientist, ShiftLeft Inc.
- PhD, DORSAL Lab, *École Polytechnique de Montréal (University of Montréal)*
- Loves tracing, security, performance analysis, low-level hardware stuff, **poutine** and **samosas**
- @tuxology



whoami

Suchakra

- Staff Scientist, ShiftLeft Inc.
- PhD, DORSAL Lab, *École Polytechnique de Montréal (University of Montréal)*
- Loves tracing, security, performance analysis, low-level hardware stuff, **poutine** and **samosas**
- @tuxology



Agenda

Humans and Perception

Software Systems Visualization

- Analysis Taxonomy
- Gathering Systems Data
- Store and Visualize
- Examples
- Role of Colors

Sample Visuals

- FlameGraphs/FlameCharts, Timelines, Heatmaps

Demos

Future Stuff



Get **Inspired**

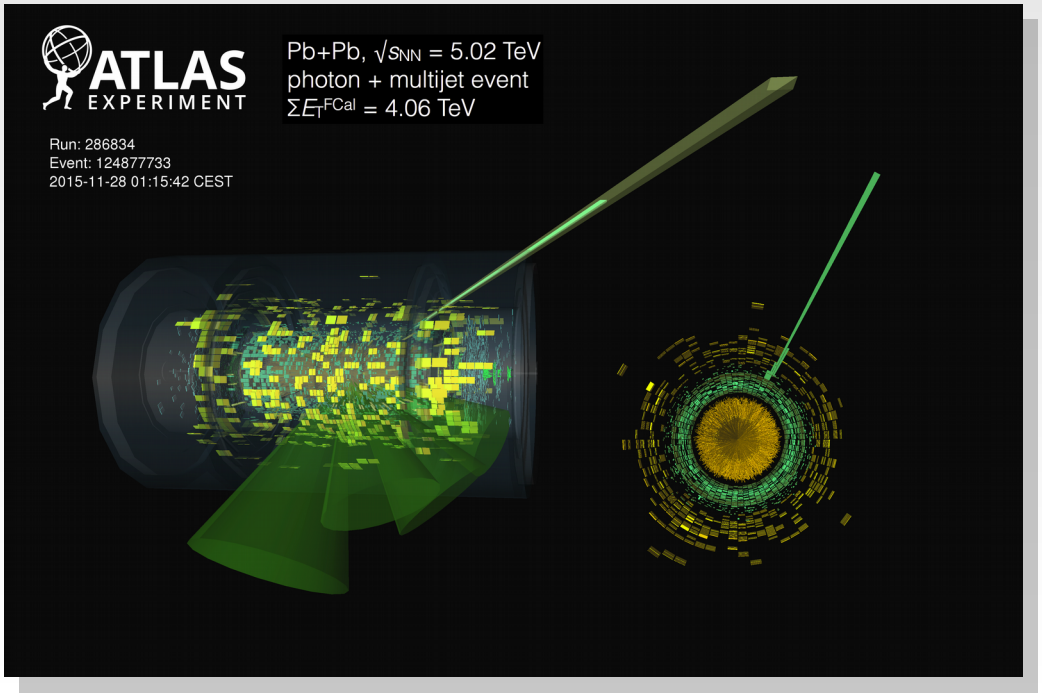
Humans and Perception



Hand stencils from a cave in Sulawesi, Indonesia (~40,000 years old)

https://www.smithsonianmag.com/science-nature/rockart-ages-indonesian-cave-paintings-are-40000-years-old-180952970/

Humans and Perception

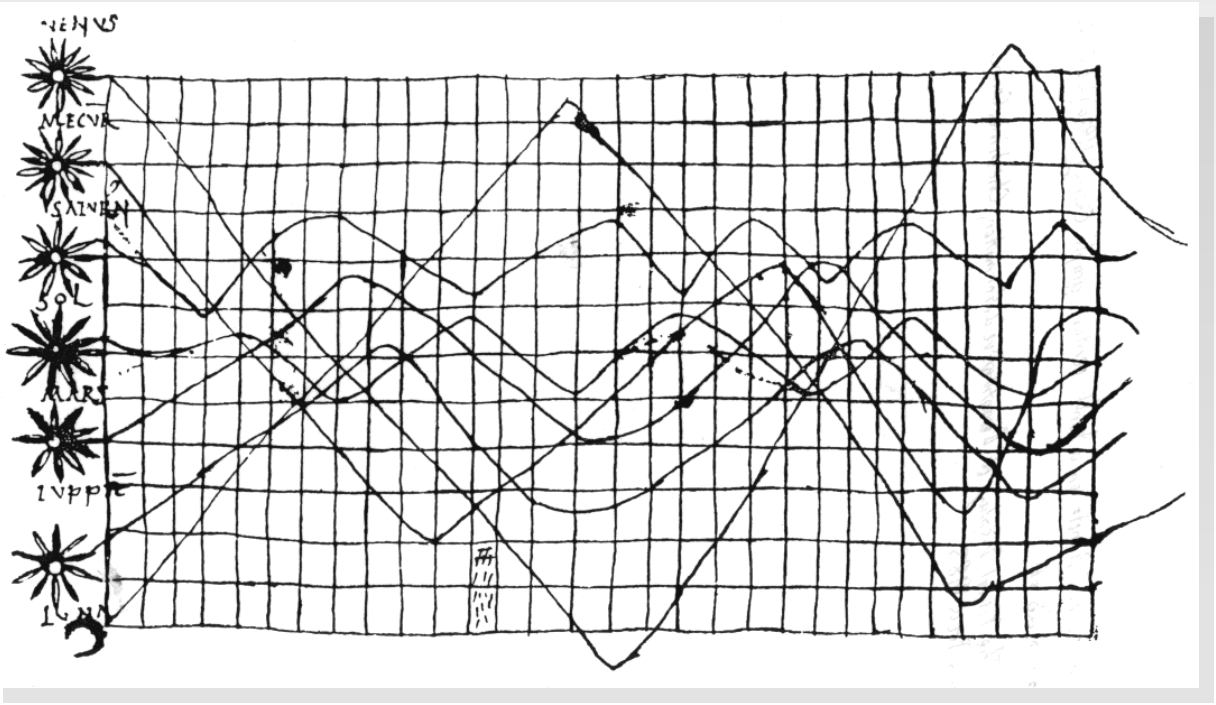


Event display of a lead-lead collision with a large transverse momentum photon (2017, CERN)

http://atlas.cern/updates/physics-briefing/photon-tagged-jet-quenching-quark-gluon-plasma

Visualizing Data

Brief Historical Perspective



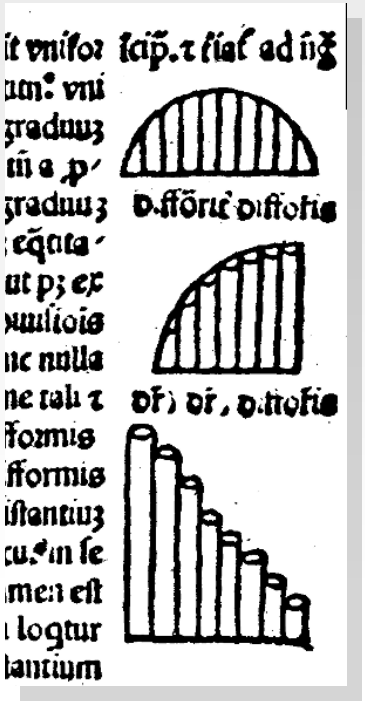
Planetary movements shown as cyclic Inclinations over time Y axis is inclination, X axis is time. (Unknown, 10th Century)

Earliest form of time-series - data is statistically less rigorous

A Brief History of Data Visualization
Michael Friendly : http://www.datavis.ca/papers/hbook.pdf

Visualizing Data

Brief Historical Perspective



Early prototype of bar Graphs,
- *The Latitude of Forms*
(Nicole Orseme, 1350)

Velocity of an object with time

Bar Charts before they were cool!

Visualizing Data

Brief Historical Perspective



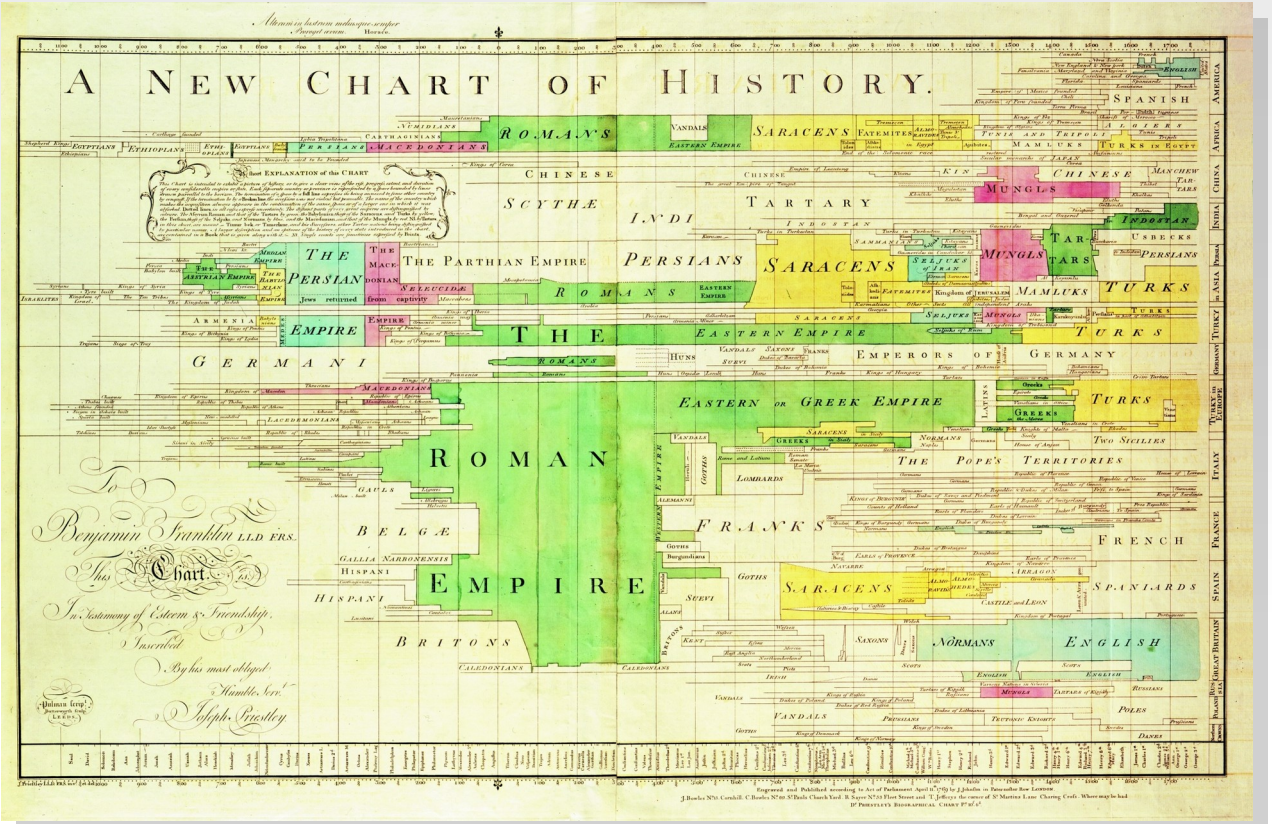
A New and Correct Chart Showing the Variations of the Compass in the Western & Southern Oceans (Edmund Halley, c. 1749)

Spatial Data - Shipping and cartography pushed the limits!

A Brief History of Data Visualization
Michael Friendly : http://www.datavis.ca/papers/hbook.pdf

Visualizing Data

Brief Historical Perspective

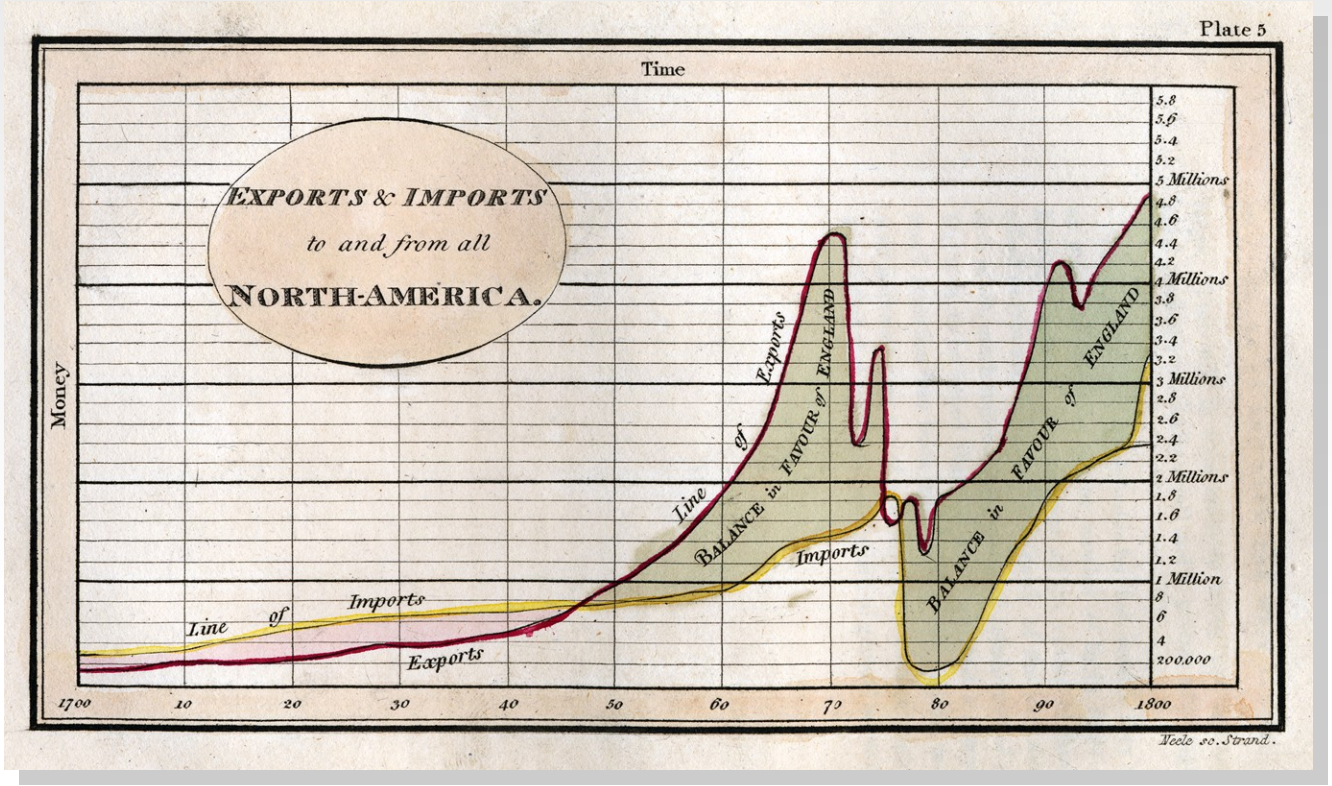


A New Chart of History
(Joseph Priestly - 1769)

Time and Space, development of Timelines, colors and their significance

Visualizing Data

Brief Historical Perspective



Imports and Exports from America in 18th Century (William Playfair)

Modern Time-series and line graphs, statistical rigor



This is not a presentation about **Data Viz**

Visualizing Data

Stats & Data 101

- **OpenIntro Statistics & Advanced High School Statistics**
  - https://www.openintro.org/stat/

Classification and Process 101

- **Angela Zoss**
  - https://guides.library.duke.edu/c.php?g=289678&p=1930713
  - https://www.slideshare.net/amzoss/everything-data
- **Udemy Course**
  - https://www.udemy.com/introduction-to-data-visualization/

Software and Systems
Visualizations



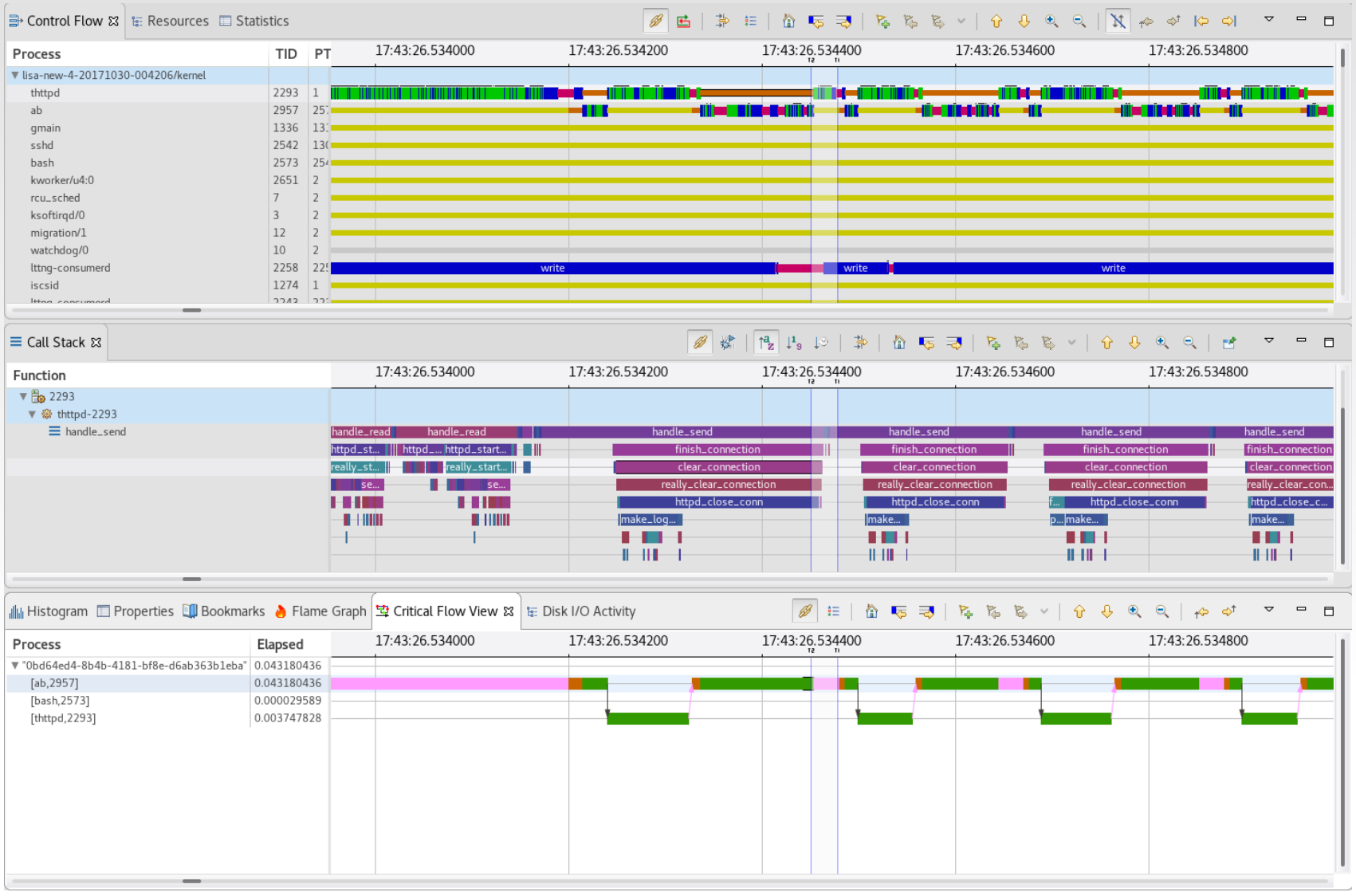
Get **Inspired**



Get **Inspired**

...again

```
[00:43:52.346539963] (+0.000001999) ubuntu-lisa kmem_cache_alloc: { cpu_id = 1 }, { call_site =
0xFFFFFFFF81211A3C, ptr = 0xFFFF880036C5BB00, bytes_req = 256, bytes_alloc = 256, gfp_flags = 37781696 }
[00:43:52.346540662] (+0.000000699) ubuntu-lisa kmem_kmalloc: { cpu_id = 1 }, { call_site =
0xFFFFFFFF8139460C, ptr = 0xFFFF88003117DCC0, bytes_req = 24, bytes_alloc = 32, gfp_flags = 37781696 }
[00:43:52.346546536] (+0.000005874) ubuntu-lisa kmem_cache_free: { cpu_id = 1 }, { call_site =
0xFFFFFFFF8121FCF4, ptr = 0xFFFF88007ADC6000 }
[00:43:52.346547182] (+0.000000646) ubuntu-lisa syscall_exit_open: { cpu_id = 1 }, { ret = 3 }
[00:43:52.346555162] (+0.000007980) ubuntu-lisa syscall_entry_newfstat: { cpu_id = 1 }, { fd = 3 }
[00:43:52.346556962] (+0.000001800) ubuntu-lisa syscall_exit_newfstat: { cpu_id = 1 }, { ret = 0, statbuf
= 140730433466352 }
[00:43:52.346559844] (+0.000002882) ubuntu-lisa kmem_mm_page_alloc: { cpu_id = 1 }, { page =
0xFFFFEA0000B7A040, pfn = 188033, order = 0, gfp_flags = 37912778, migratetype = 1 }
[00:43:52.346560899] (+0.000001055) ubuntu-lisa syscall_entry_read: { cpu_id = 1 }, { fd = 3, count =
4096 }
[00:43:52.346563739] (+0.000002840) ubuntu-lisa syscall_exit_read: { cpu_id = 1 }, { ret = 19, buf =
94392247288688 }
[00:43:52.346569483] (+0.000005744) ubuntu-lisa syscall_entry_read: { cpu_id = 1 }, { fd = 3, count =
4096 }
[00:43:52.346570064] (+0.000000581) ubuntu-lisa syscall_exit_read: { cpu_id = 1 }, { ret = 0, buf =
94392247288688 }
[00:43:52.346571514] (+0.000001450) ubuntu-lisa syscall_entry_close: { cpu_id = 1 }, { fd = 3 }
[00:43:52.346572986] (+0.000001472) ubuntu-lisa syscall_exit_close: { cpu_id = 1 }, { ret = 0 }
[00:43:52.346573738] (+0.000000752) ubuntu-lisa kmem_kfree: { cpu_id = 1 }, { call_site =
0xFFFFFFFF811B51FD, ptr = 0xFFFF88003117DCC0 }
[00:43:52.346577658] (+0.000003920) ubuntu-lisa kmem_mm_page_alloc: { cpu_id = 1 }, { page =
0xFFFFEA0001DFA600, pfn = 491160, order = 0, gfp_flags = 37912778, migratetype = 1 }
[00:43:52.346580365] (+0.000002707) ubuntu-lisa kmem_mm_page_alloc: { cpu_id = 1 }, { page =
0xFFFFEA0001443E40, pfn = 332025, order = 0, gfp_flags = 37912778, migratetype = 1 }
[00:43:52.346583219] (+0.000002854) ubuntu-lisa syscall_entry_getuid: { cpu_id = 1 }, { }
[00:43:52.346583541] (+0.000000322) ubuntu-lisa syscall_exit_getuid: { cpu_id = 1 }, { ret = 0 }
```


Control Flow
Resources
Statistics
Process
TID
PT
17:43:26.534000
17:43:26.534200
17:43:26.534400
17:43:26.534600
17:43:26.534800
lisa-new-4-20171030-004206/kernel
thttpd 2293
ab 2957
gmain 1336
sshd 2542
bash 2573
kworker/u4:0 2651
rcu_sched 7
ksoftirqd/0 3
migration/1 12
watchdog/0 10
lttng-consumerd 2258
iscsid 1274
write
Call Stack
Function
2293
thttpd-2293
handle_send
handle_read
finish_connection
clear_connection
really_clear_connection
httpd_close_conn
Histogram
Properties
Bookmarks
Flame Graph
Critical Flow View
Disk I/O Activity
Elapsed
"0bd64ed4-8b4b-4181-bf8e-d6ab363b1eba"
0.043180436
[ab,2957]
0.043180436
[bash,2573]
0.000029589
[thttpd,2293]
0.003747828

Systems Analysis

A Brief Taxonomy of Analysis

Systems Analysis

A Brief Taxonomy of Analysis

Conception

Structural Analysis of software and systems

eg. Static analysis of kernel source code for program flow and dependency graphs

Systems Analysis

A Brief Taxonomy of Analysis

Conception

Structural Analysis of software and systems

eg. Static analysis of kernel source code for program flow and dependency graphs

Live

Behavioral and Execution Analysis of software systems

Dynamic analysis by instrumenting applications, buffer the events and display results live or at short samples

eg. Live traces, sampled resource usage (CPU, memory)

Systems Analysis

A Brief Taxonomy of Analysis

Conception

Structural Analysis of software and systems

eg. Static analysis of kernel source code for program flow and dependency graphs

Live

Behavioral and Execution Analysis of software systems

Dynamic analysis by instrumenting applications, buffer the events and display results live or at short samples

eg. Live traces, sampled resource usage (CPU, memory)

Post-Mortem

Behavioral, Execution and Root Cause Analysis of software systems

Gather as much data with as much granularity. Characterize, visualize and infer.

eg. Root cause analysis of abnormal interrupt latency, software/hardware traces

Visualizing Systems Data

From Problem to Insight

Visualizing Systems Data

From Problem to Insight

Question

Program is making
a bunch of syscalls,
some are slow.

Which? Why?
Where?

Visualizing Systems Data

From Problem to Insight

Question → **Gather**

Program is making a bunch of syscalls, some are slow.

Which? Why? Where?

Instrument syscalls with timestamps and observe them as they occur in an execution

Record stack of all functions called inside the syscall and compare executions

Sample or **Trace**

Visualizing Systems Data

From Problem to Insight



Program is making a bunch of syscalls, some are slow.

Which? Why? Where?

Instrument syscalls with timestamps and observe them as they occur in an execution

Record stack of all functions called inside the syscall and compare executions

Sample or **Trace**

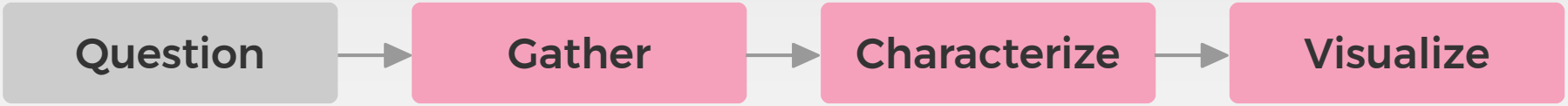
Frequency Analysis

Latency Analysis

Multi-modal Analysis (Hierarchy + Time)

Visualizing Systems Data

From Problem to Insight



Program is making a bunch of syscalls, some are slow.

Which? Why? Where?

Instrument syscalls with timestamps and observe them as they occur in an execution

Record stack of all functions called inside the syscall and compare executions

Sample or **Trace**

Frequency Analysis

Latency Analysis

Multi-modal Analysis (Hierarchy + Time)

Histograms, Scatter-plots

Heat-maps

Graphs, Timelines, Flamecharts, Flamegraphs

Visualizing Systems Data

Case Study - **Conception**

Questions

How would the code flow in my web-server?

What are terminal sources and sinks?

thttpd.c

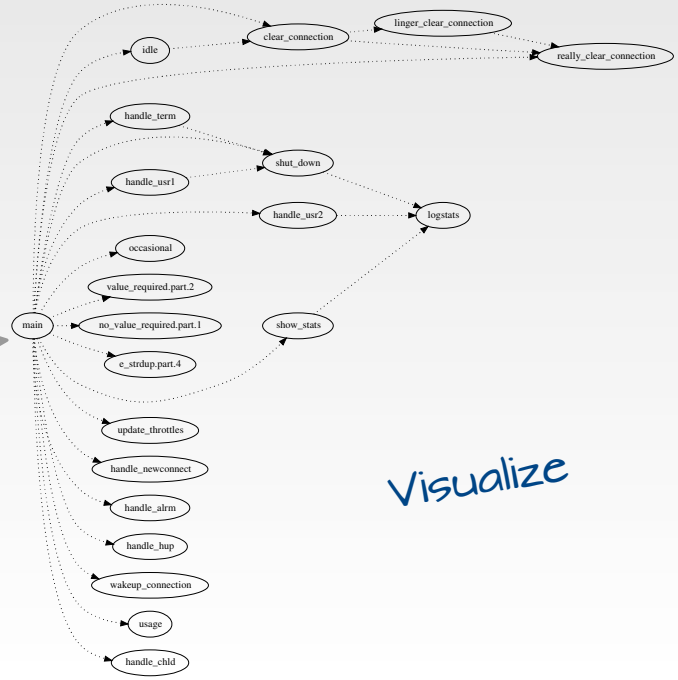
`CFLAGS=-fdump-rtl-expand`

Gather

thttpd.c.212r.expand

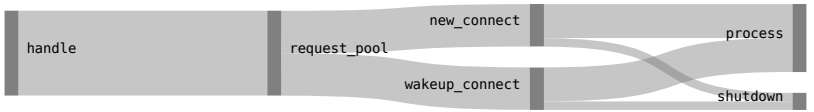
Callgraph Extraction From RTL

Visualize



Flow Classification

Characterize

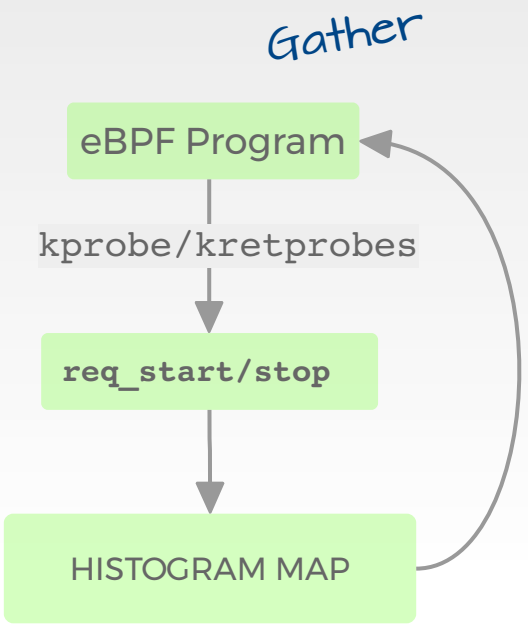


Visualizing Systems Data

Case Study - **Live**

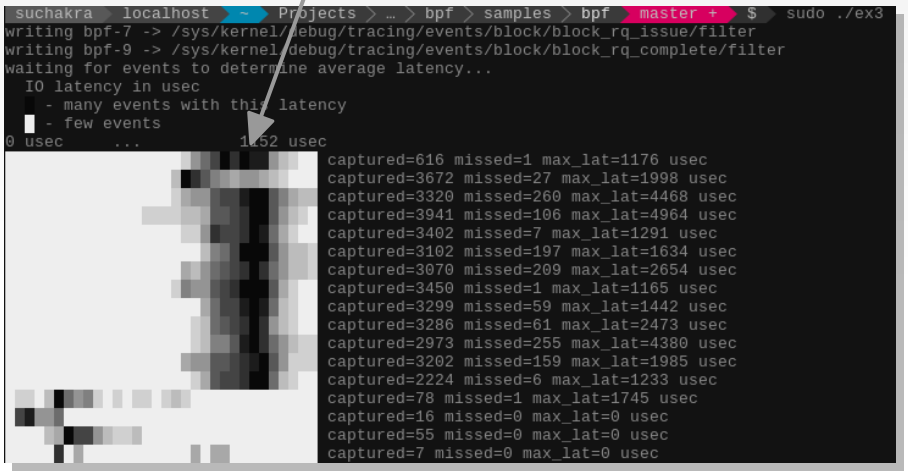
Questions

How do we observe disk I/O latencies?




```
usecs               : count     distribution
    0 -> 1          : 0        |                                        |
    2 -> 3          : 0        |                                        |
    4 -> 7          : 0        |                                        |
    8 -> 15         : 0        |                                        |
   16 -> 31         : 115      |**                                      |
   32 -> 63         : 553      |**********                              |
   64 -> 127        : 1087     |********************                    |
  128 -> 255        : 1572     |*****************************           |
  256 -> 511        : 1040     |*******************                     |
  512 -> 1023       : 54       |*                                       |
```

Visualize

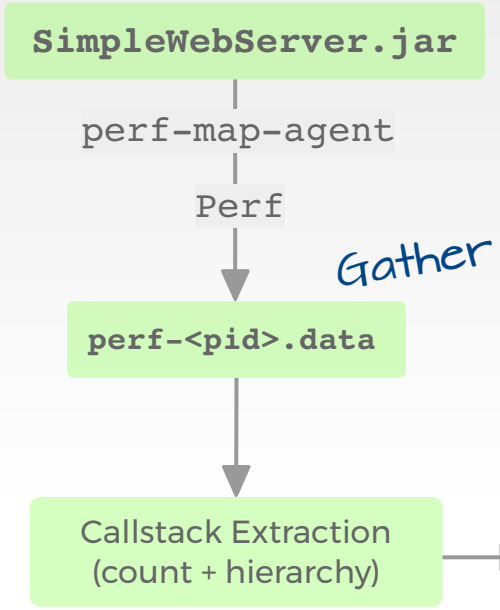


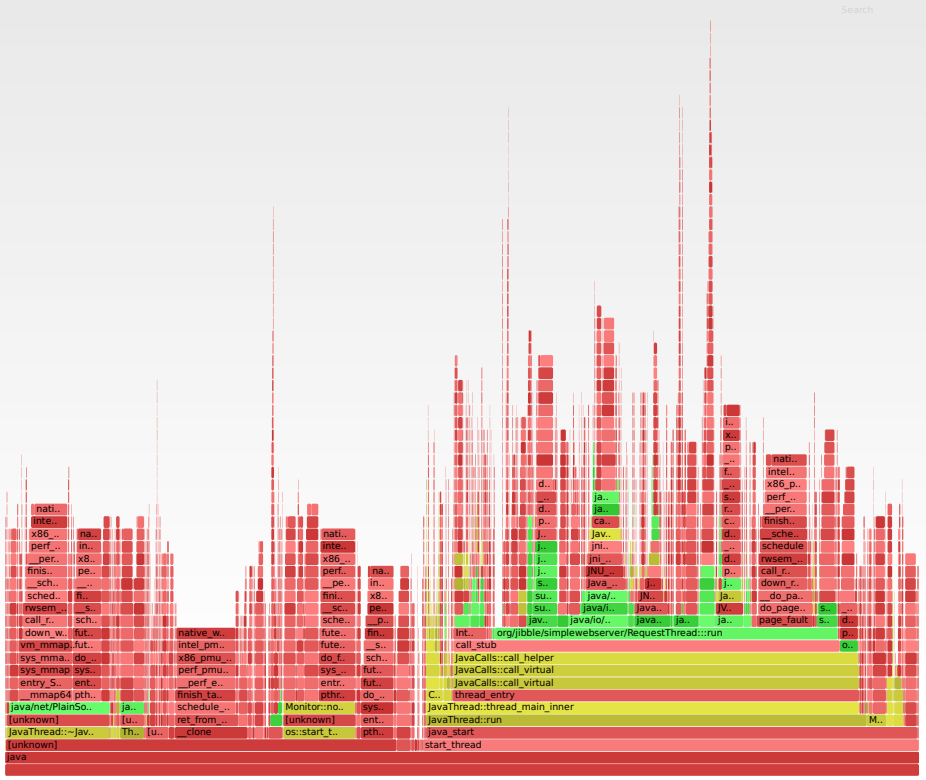
Visualizing Systems Data

Case Study - **Post-Mortem**

Questions

What were most frequent calls made? What did those call-stacks looked like? How did they interact with the underlying system?





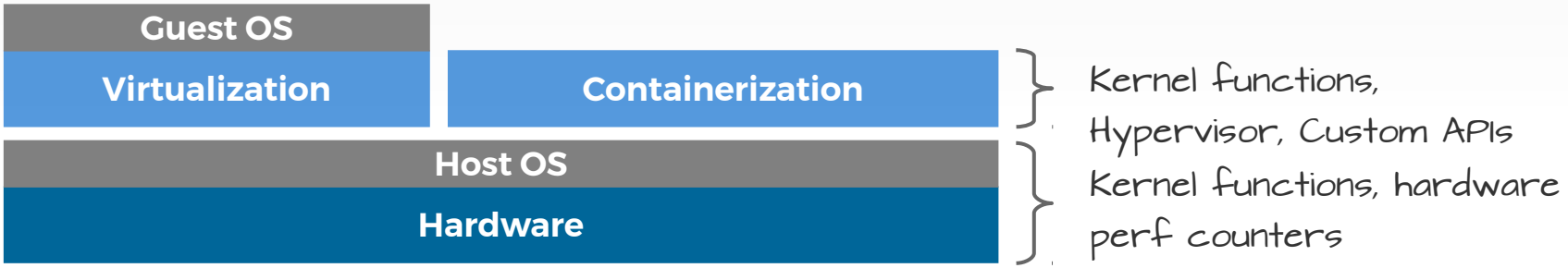
Gathering **Systems Data**

Observability Layers

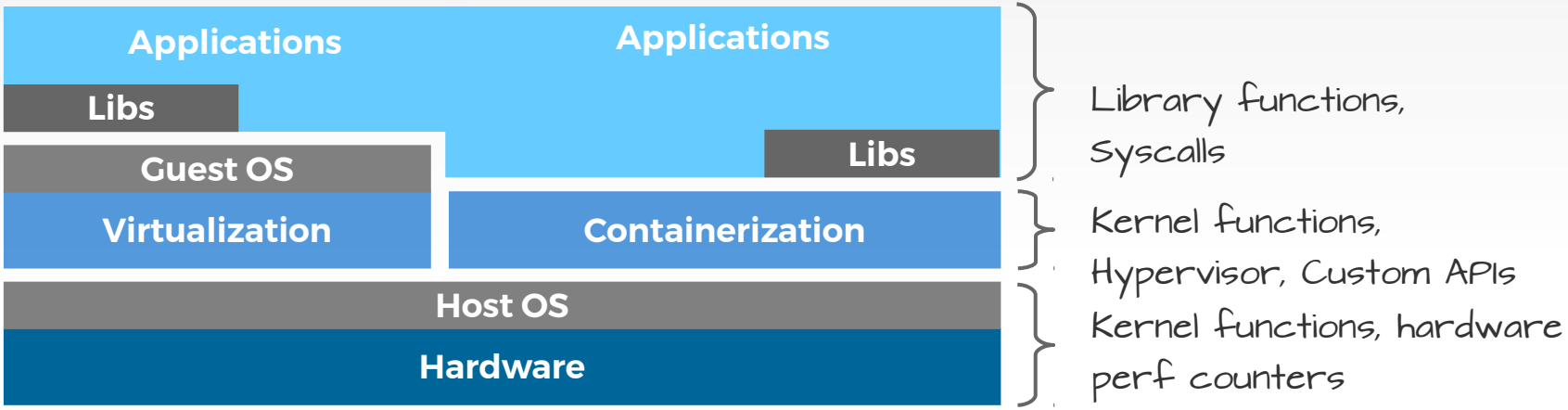
Observability Layers



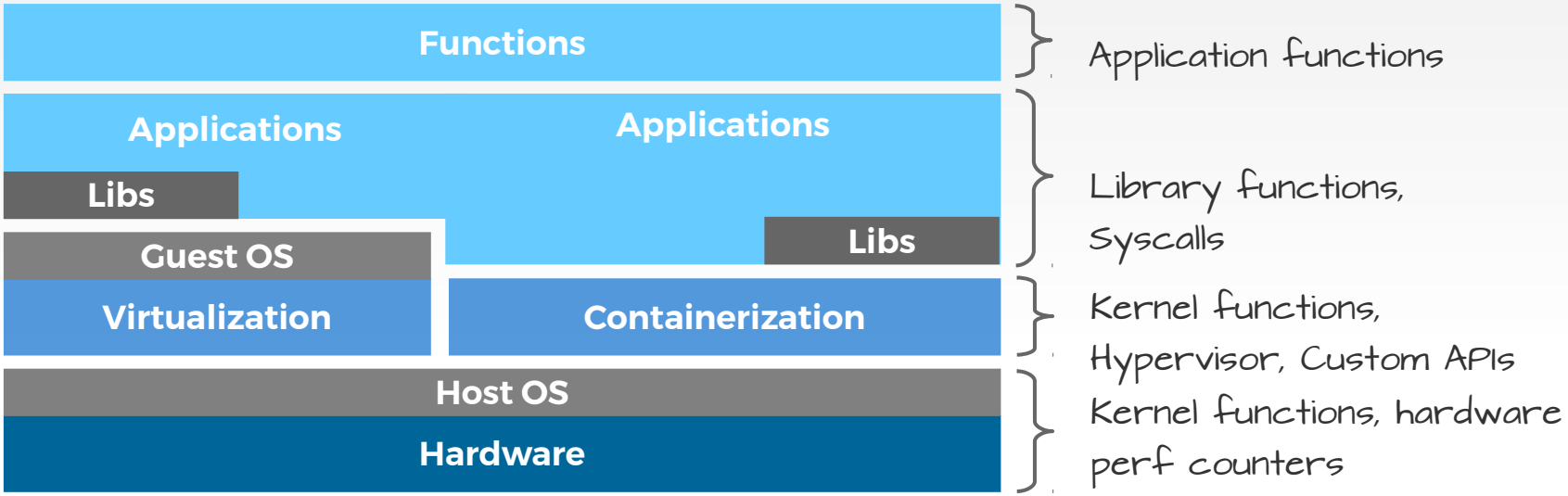
Observability Layers



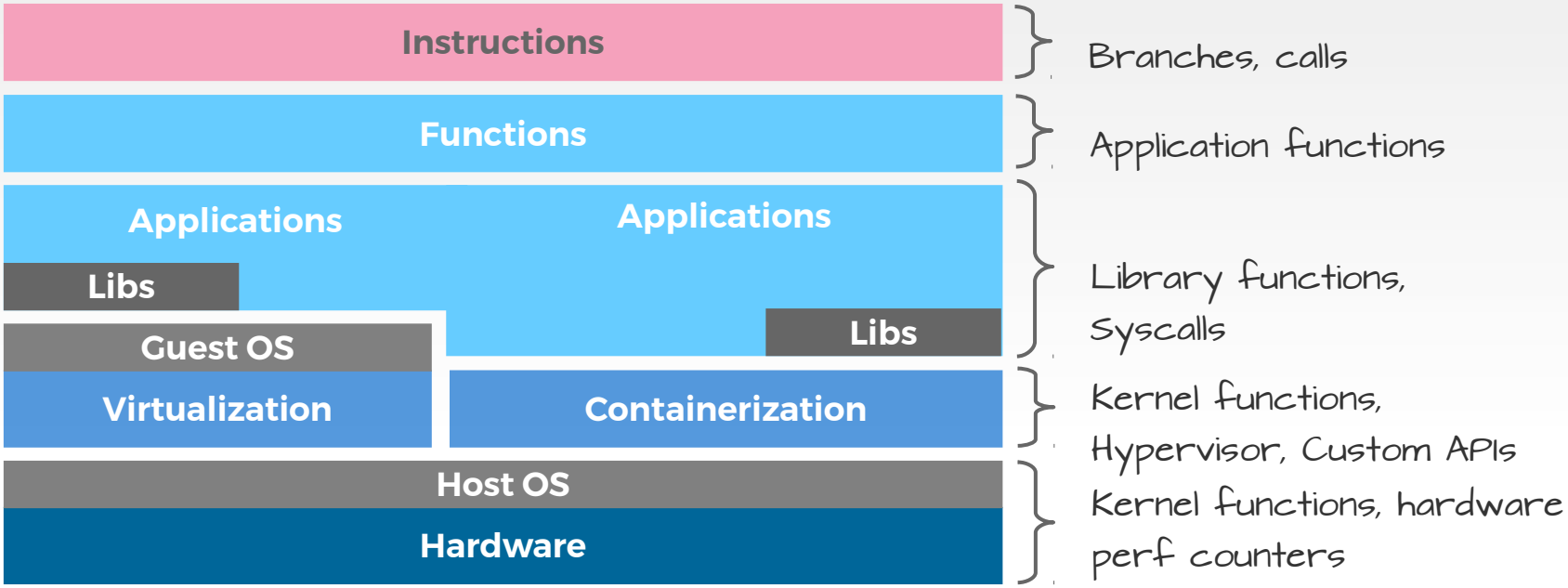
Observability Layers



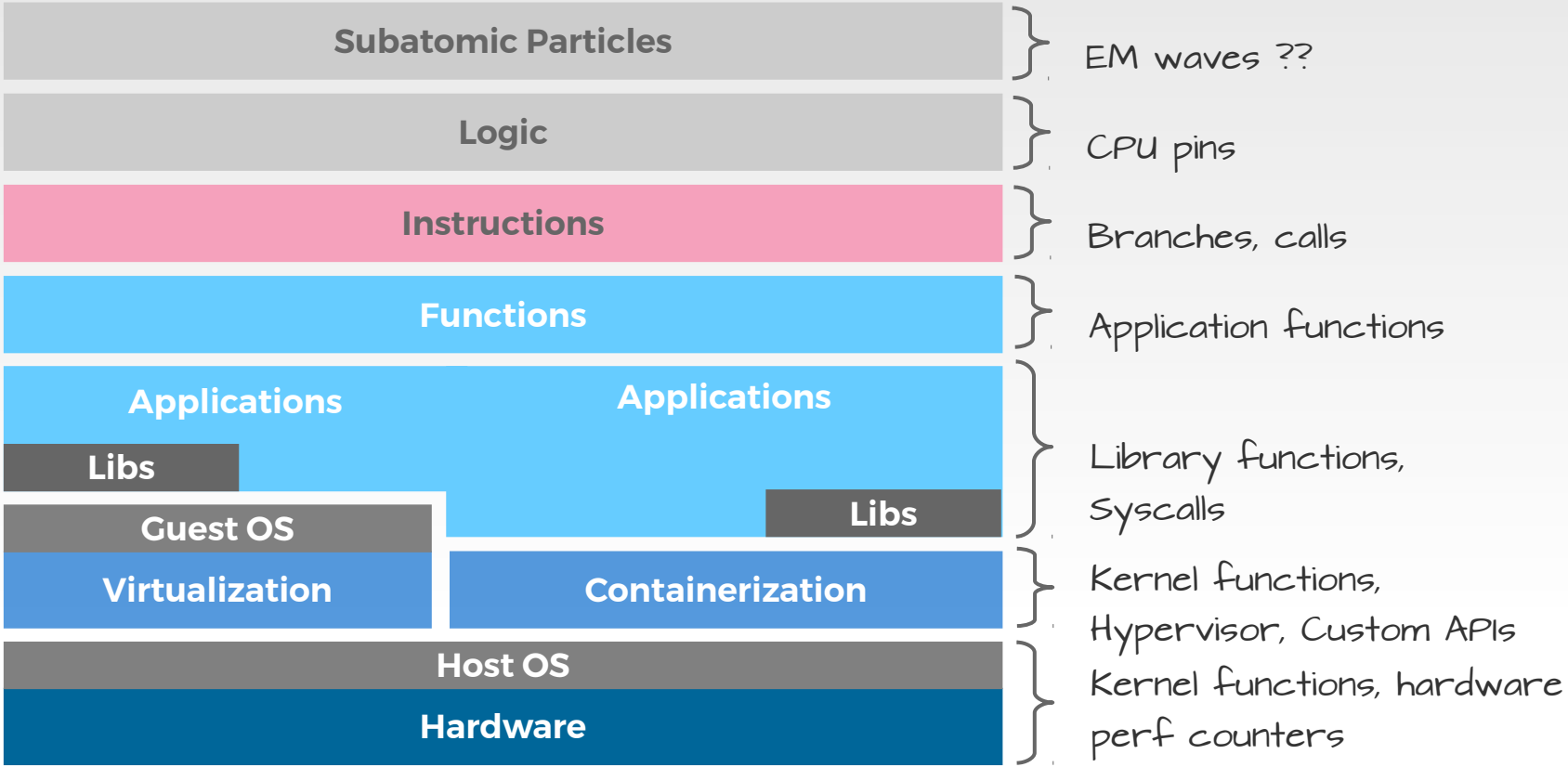
Observability Layers



Observability Layers



Observability Layers



Data Sources

System

- Functions: Static Tracing (Tracepoints, Ftrace)
- Functions: Dynamic Tracing (Kprobes, Kretprobes)
- PMU Events (Perf)
- Hardware Counters (Perf)
- Hardware Tracing Blocks (Intel PT, ARM CoreSight)
- Kernel Logs (LSM Audit logs)

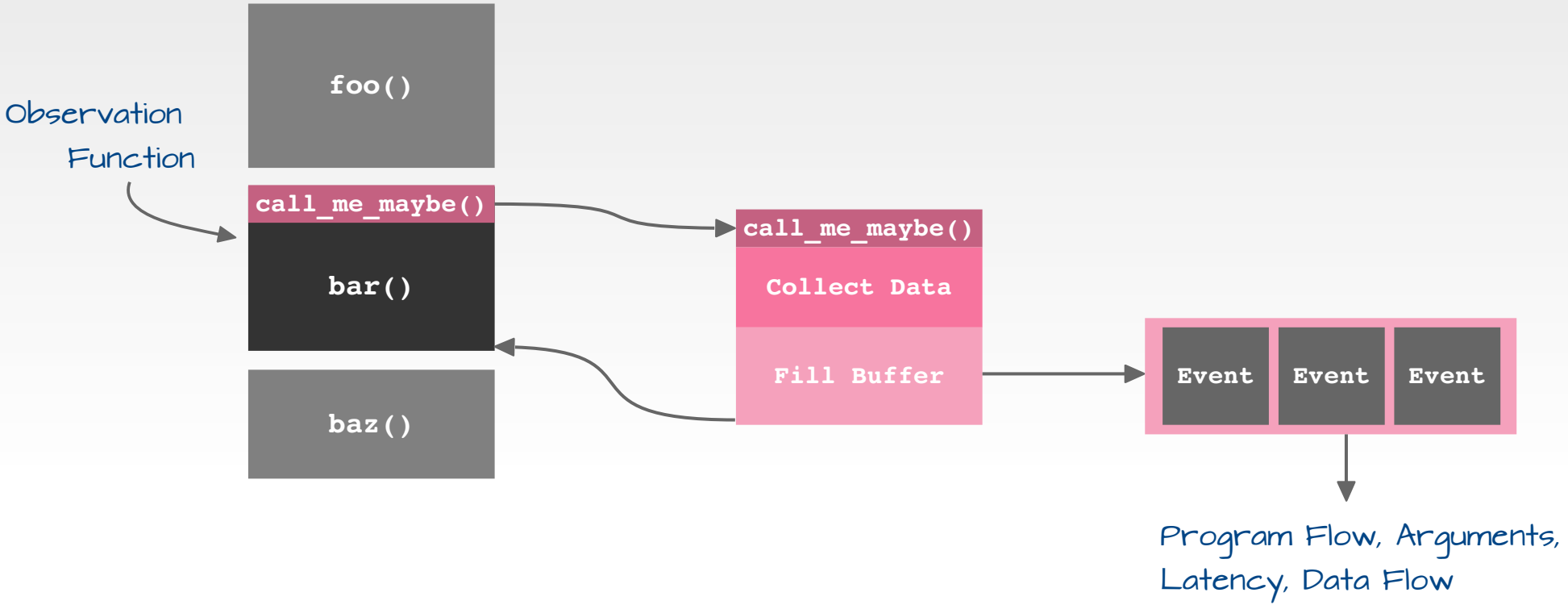
Applications

- Application Logs/Traces (JUL)
- Userspace tracing (eg. Uftrace, compiler instrumentation)



Deep Dive
Tracing

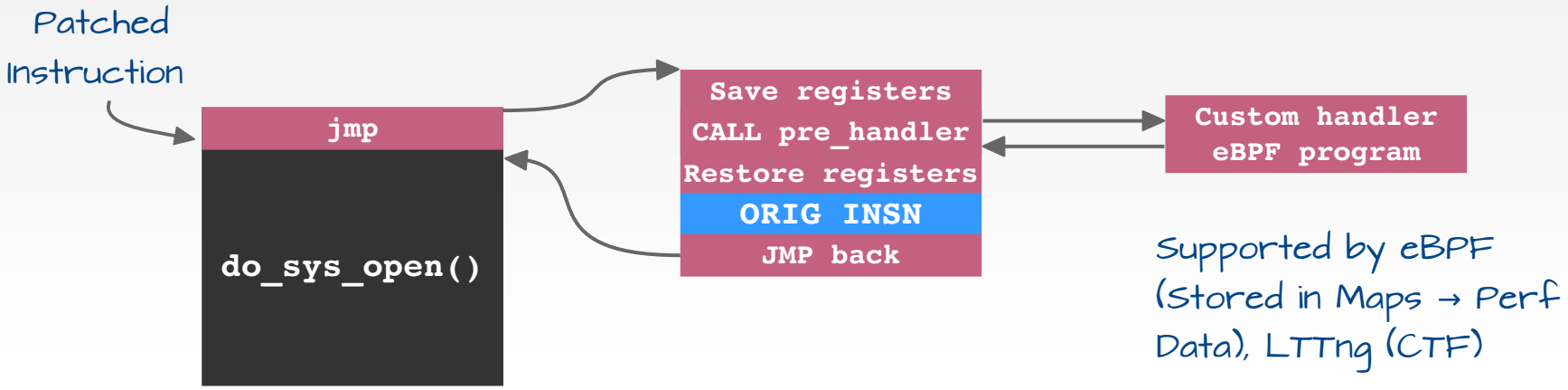
Observing Functions



Observing Functions

Kernel Example

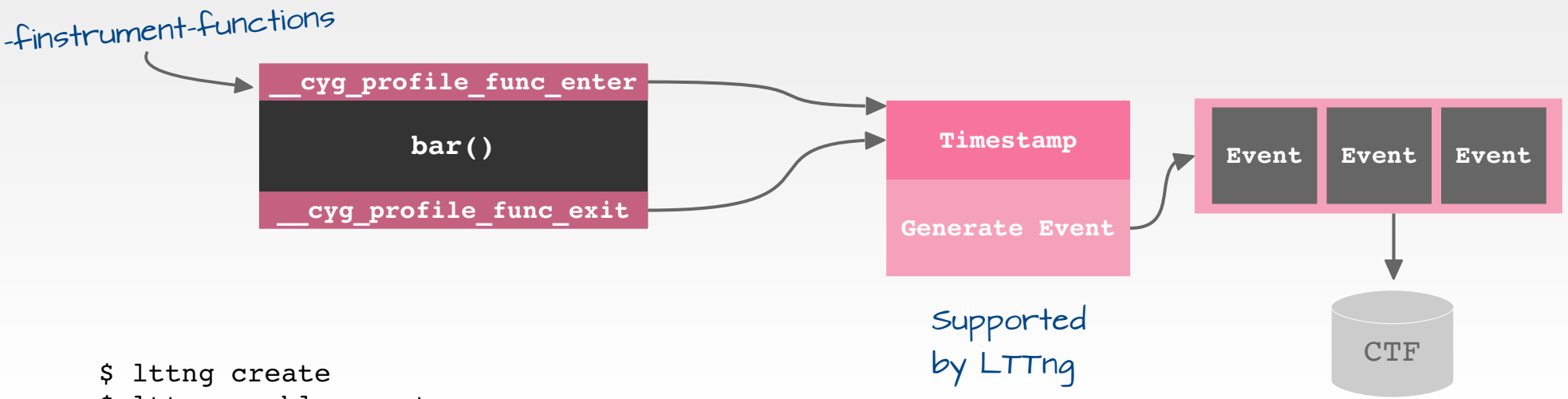
- Dynamically instrument with Kprobes



Observing Functions

Userspace Example

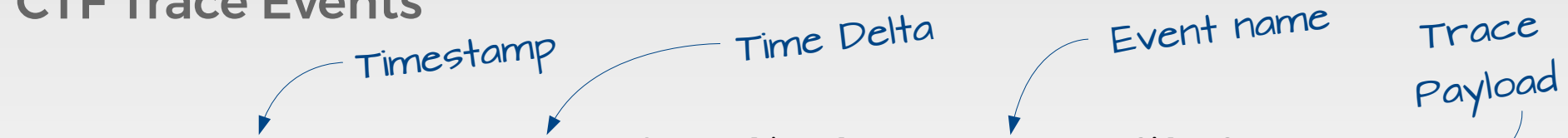
- Compiler-assisted instrumentation




```
$ lttng create
$ lttng enable-event -a -u
$ lttng add-context -u -t vpid -t vtid -t procname
$ lttng start
$ LD_PRELOAD=/usr/lib/liblttng-ust-cyg-profile.so ./thttpd
$ lttng stop
```

Observing Functions

CTF Trace Events




```
[17:43:26.548312332] (+0.000000098) ubuntu-lisa lttng_ust_cyg_profile:func_entry:
{ cpu_id = 0 }, { vpid = 2293, vtid = 2293, procname = "thttpd" }, { addr = 0x410B50,
call_site = 0x411414 }

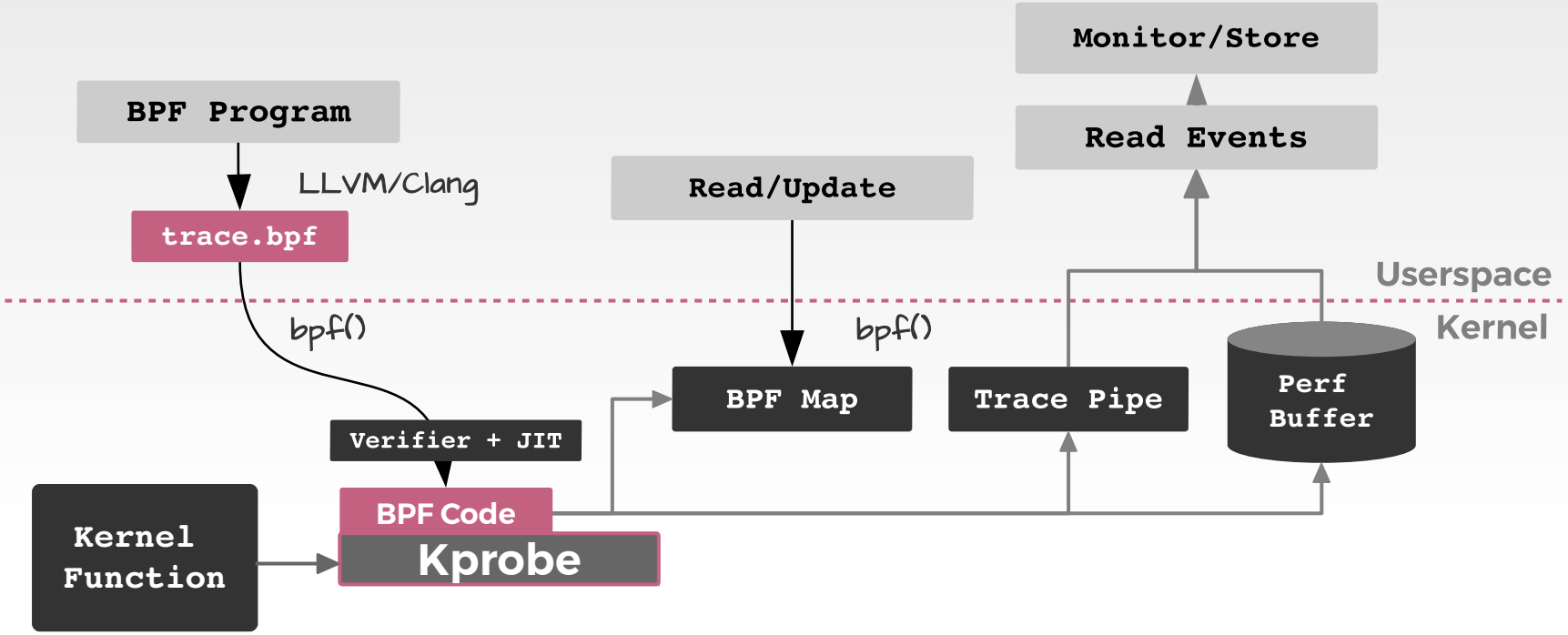
[17:43:26.548312980] (+0.000000104) ubuntu-lisa lttng_ust_cyg_profile:func_entry:
{ cpu_id = 0 }, { vpid = 2293, vtid = 2293, procname = "thttpd" }, { addr = 0x410AE0,
call_site = 0x410B86 }

[17:43:26.548313787] (+0.000000103) ubuntu-lisa lttng_ust_cyg_profile:func_exit: { cpu_id
= 0 }, { vpid = 2293, vtid = 2293, procname = "thttpd" }, { addr = 0x410AE0, call_site =
0x410B86 }

[17:43:26.548314427] (+0.000000103) ubuntu-lisa lttng_ust_cyg_profile:func_exit: { cpu_id
= 0 }, { vpid = 2293, vtid = 2293, procname = "thttpd" }, { addr = 0x410B50, call_site =
0x411414 }
```

Observing Functions

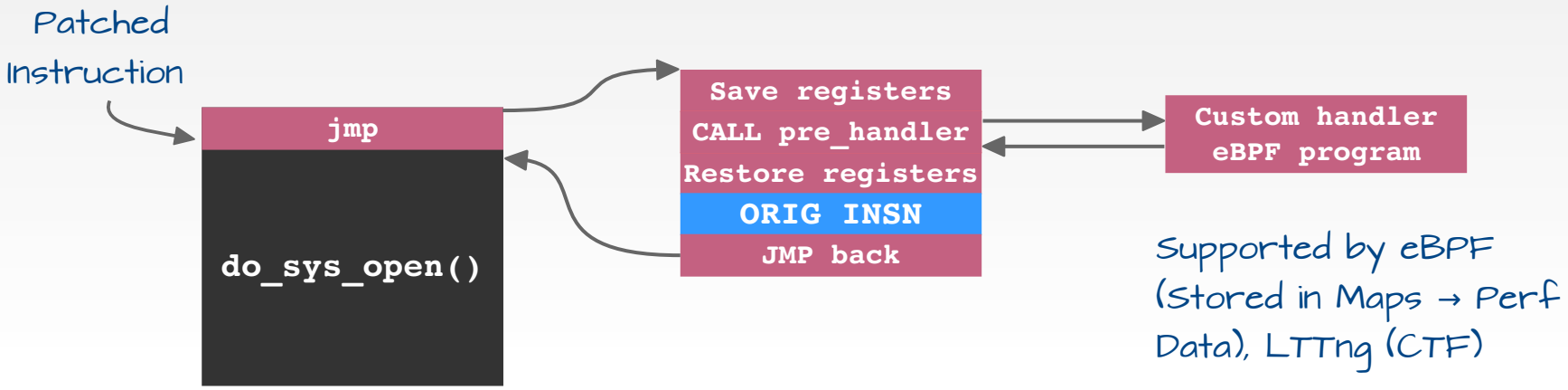
eBPF + Kprobes



Observing Functions

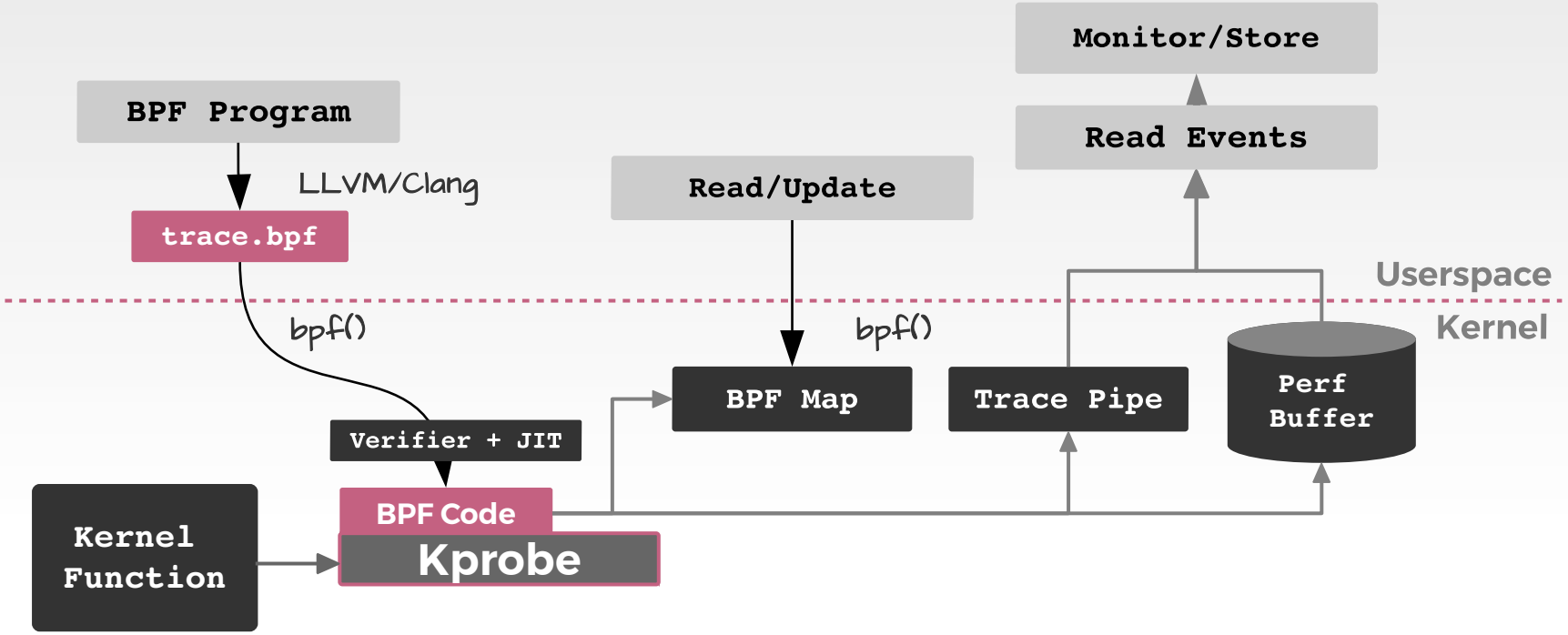
Kernel Example

- Dynamically instrument with Kprobes



Observing Functions

eBPF + Kprobes



Observing Functions

eBPF + Kprobes

- IOVisor BCC – Python, C++, Lua, Go (`gobpf`) APIs
- Compile BPF programs directly via LLVM interface
- Helper functions to manage maps, buffers, probes

Complete Program
trace_fields.py

```
from bcc import BPF

prog = """
int hello(void *ctx) {
  bpf_trace_printk("Hello, World!\\n");
  return 0;
}
"""
b = BPF(text=prog)
b.attach_kprobe(event="sys_clone", fn_name="hello")
print "PID MESSAGE"
b.trace_print(fmt="{1} {5}")
```

prog compiled to BPF bytecode

Attach to Kprobe event

Print trace pipe

Some Formats

Formats and Storage

- Scaled Storage (Time-series): OpenTSDB, Graphite, InfluxDB, Prometheus
- Compact Storage (Binary formats): CTF, Perf
- Graph Storage (Source Analysis): CPG (Titan/JanusDB)
- Custom
    - JSON (Uftrace)
    - XML Traces

Characterization and Visualization of **Data**

Popular Means

From Events to Visuals

- Latency/Density (Syscalls) → Histograms → Heatmaps
- Latency (Syscalls) vs Time → Scatter Plots
- Throughput → Line Charts
- Resource Consumption → Line Charts
- Per-Entity (Process/Host/VM) Events → Timeline/Gantt
- Event Summaries → Pie, Donut, Bar Charts
- Function Graphs (Time) → Flame Charts
- Function Graphs (Count) → Flame Graphs
- Callgraphs : Graphs/TreeMap/Sunburst

Histograms

Latency Histograms

- Periodic snapshots in simple ASCII

eBPF-BCC Tools

```
$ sudo ./biolatency.py
     usecs               : count     distribution
         0 -> 1          : 0        |                                        |
         2 -> 3          : 0        |                                        |
         4 -> 7          : 0        |                                        |
         8 -> 15         : 0        |                                        |
        16 -> 31         : 115      |**                                      |
        32 -> 63         : 553      |**********                              |
        64 -> 127        : 1087     |********************                    |
       128 -> 255        : 1572     |******************************          |
       256 -> 511        : 1040     |********************                    |
       512 -> 1023       : 54       |*                                       |
```

Histograms

Latency Histograms

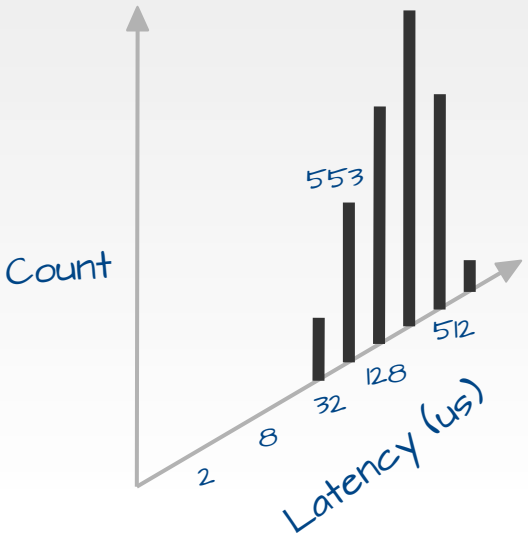
- Periodic snapshots in simple ASCII

```
$ sudo ./biolatency.py

     usecs               : count     distribution
         0 -> 1          : 0        |                                        |
         2 -> 3          : 0        |                                        |
         4 -> 7          : 0        |                                        |
         8 -> 15         : 0        |                                        |
        16 -> 31         : 115      |****                                    |
        32 -> 63         : 553      |*********************                   |
        64 -> 127        : 1087     |*****************************************|
       128 -> 255        : 1572     |*****************************************|
       256 -> 511        : 1040     |*****************************************|
       512 -> 1023       : 54       |**                                      |
```

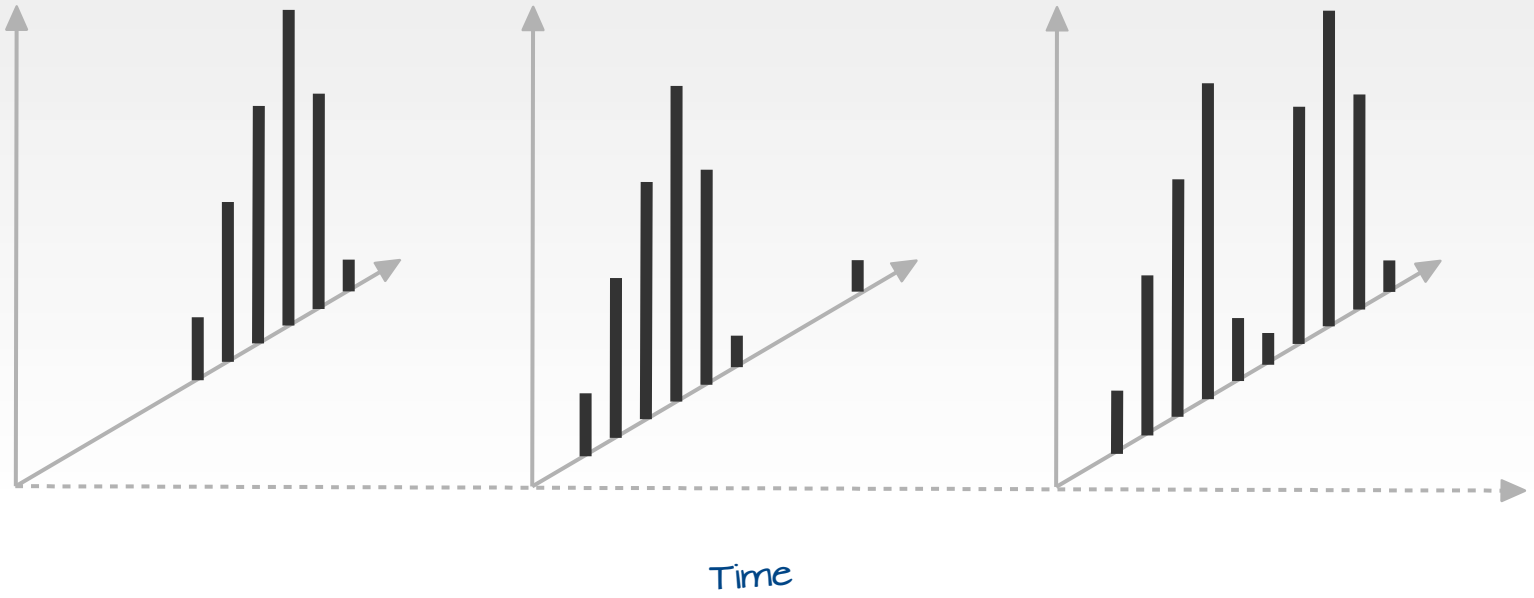
Heatmaps

Latency Heatmaps



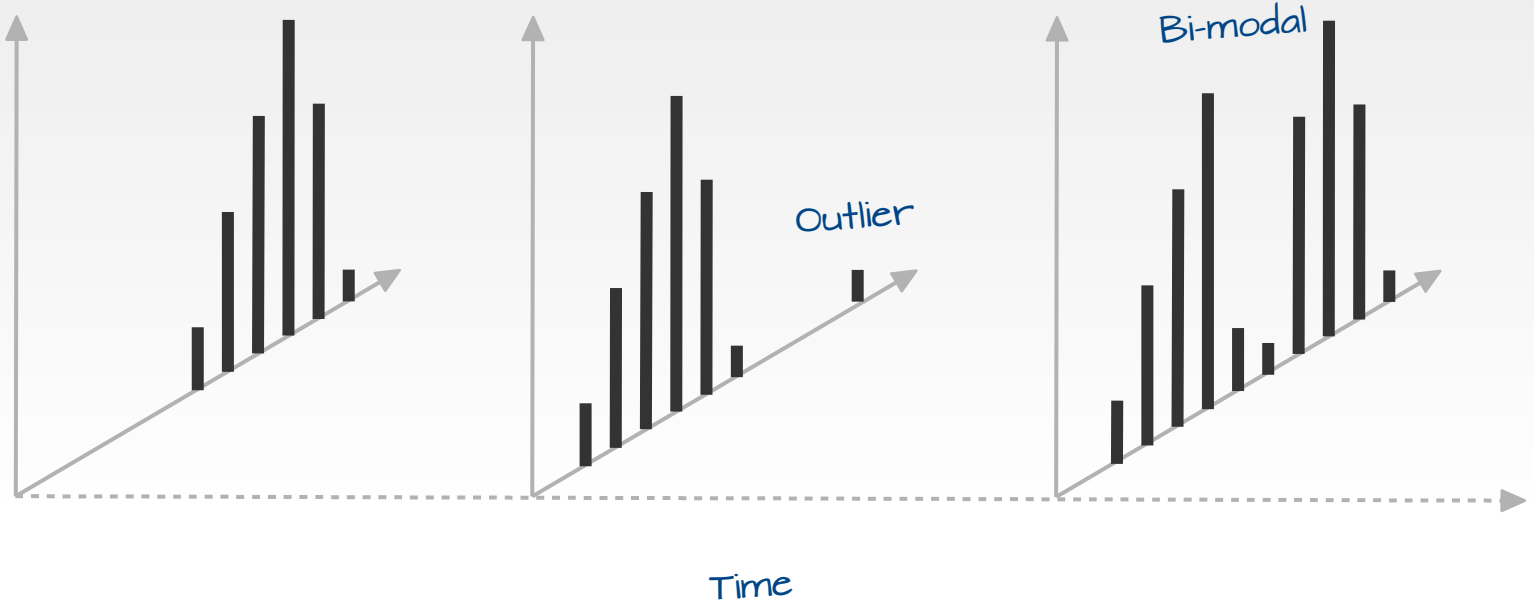
Heatmaps

Latency Heatmaps



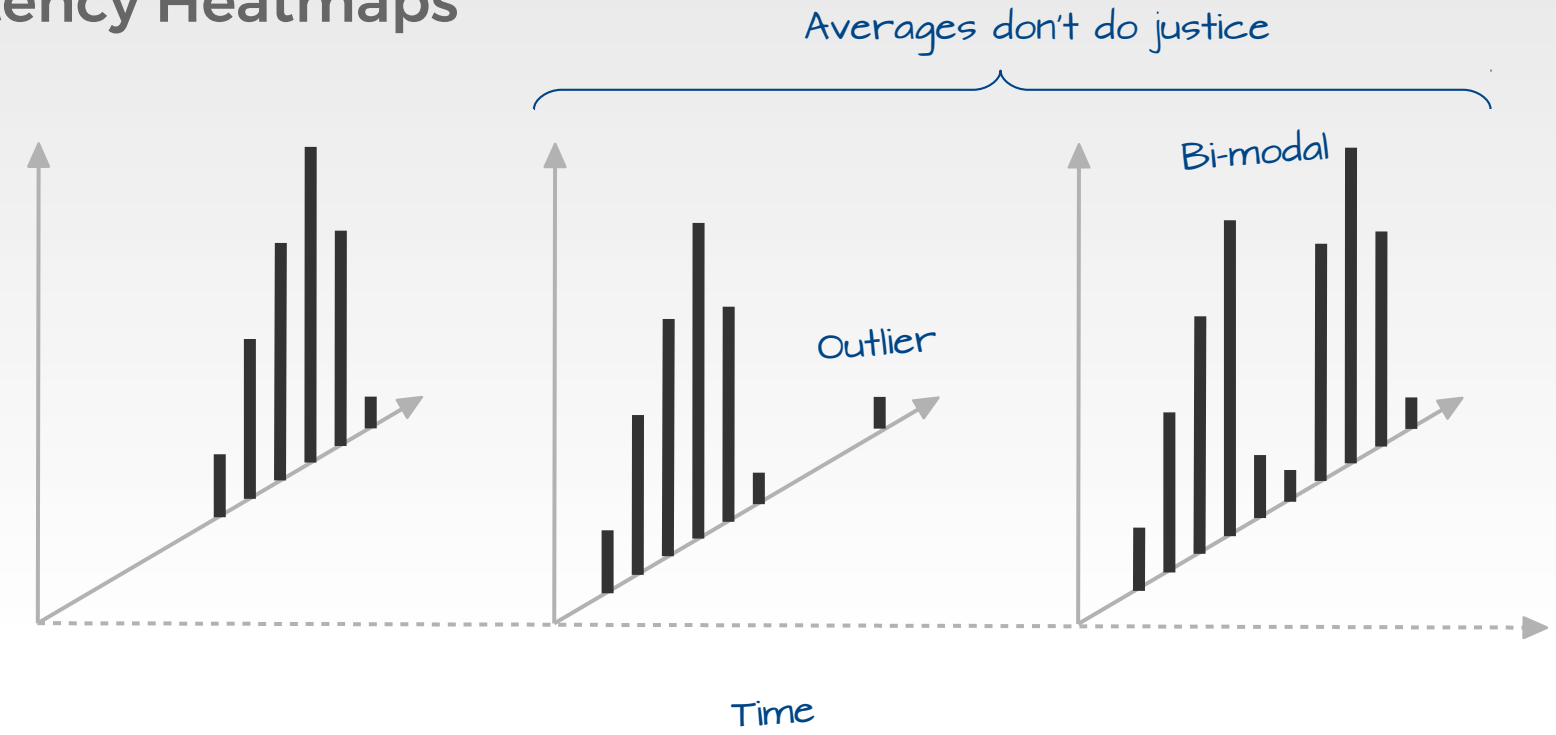
Heatmaps

Latency Heatmaps



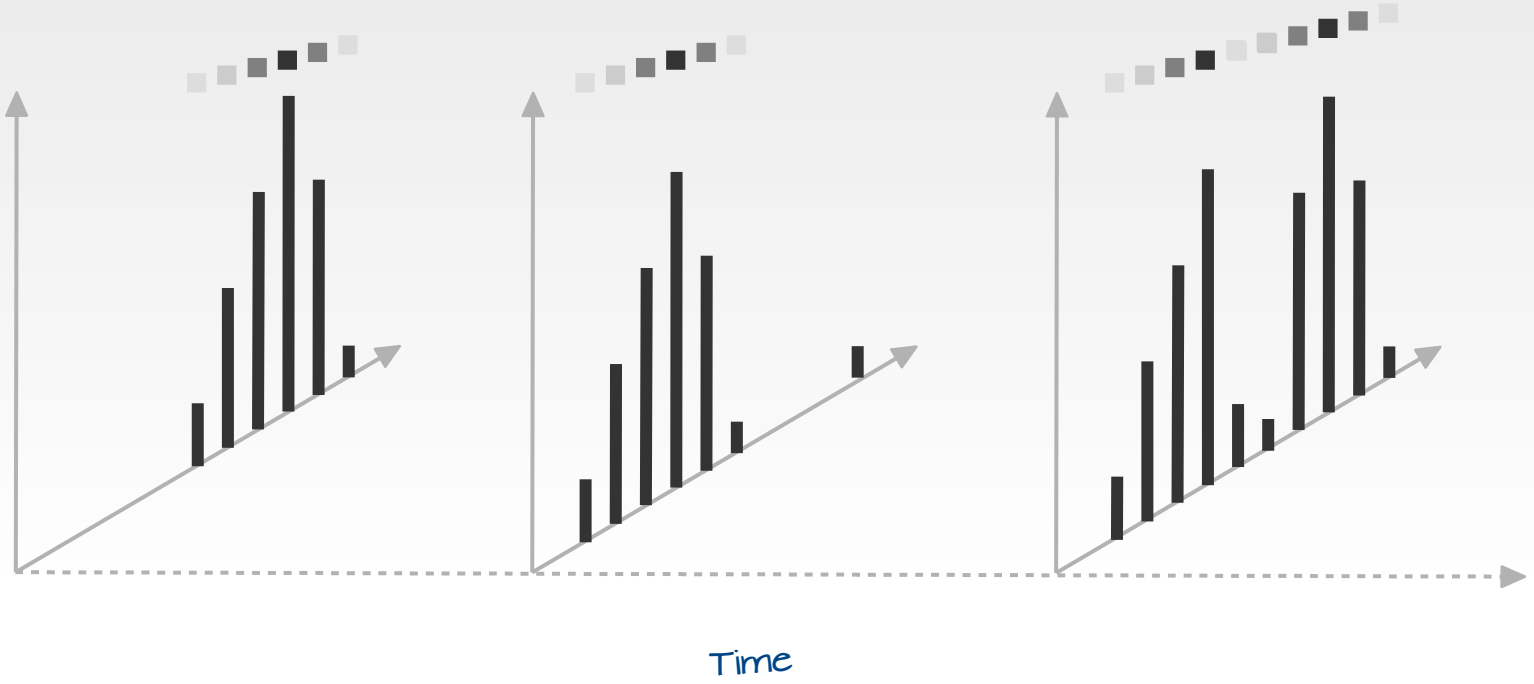
Heatmaps

Latency Heatmaps



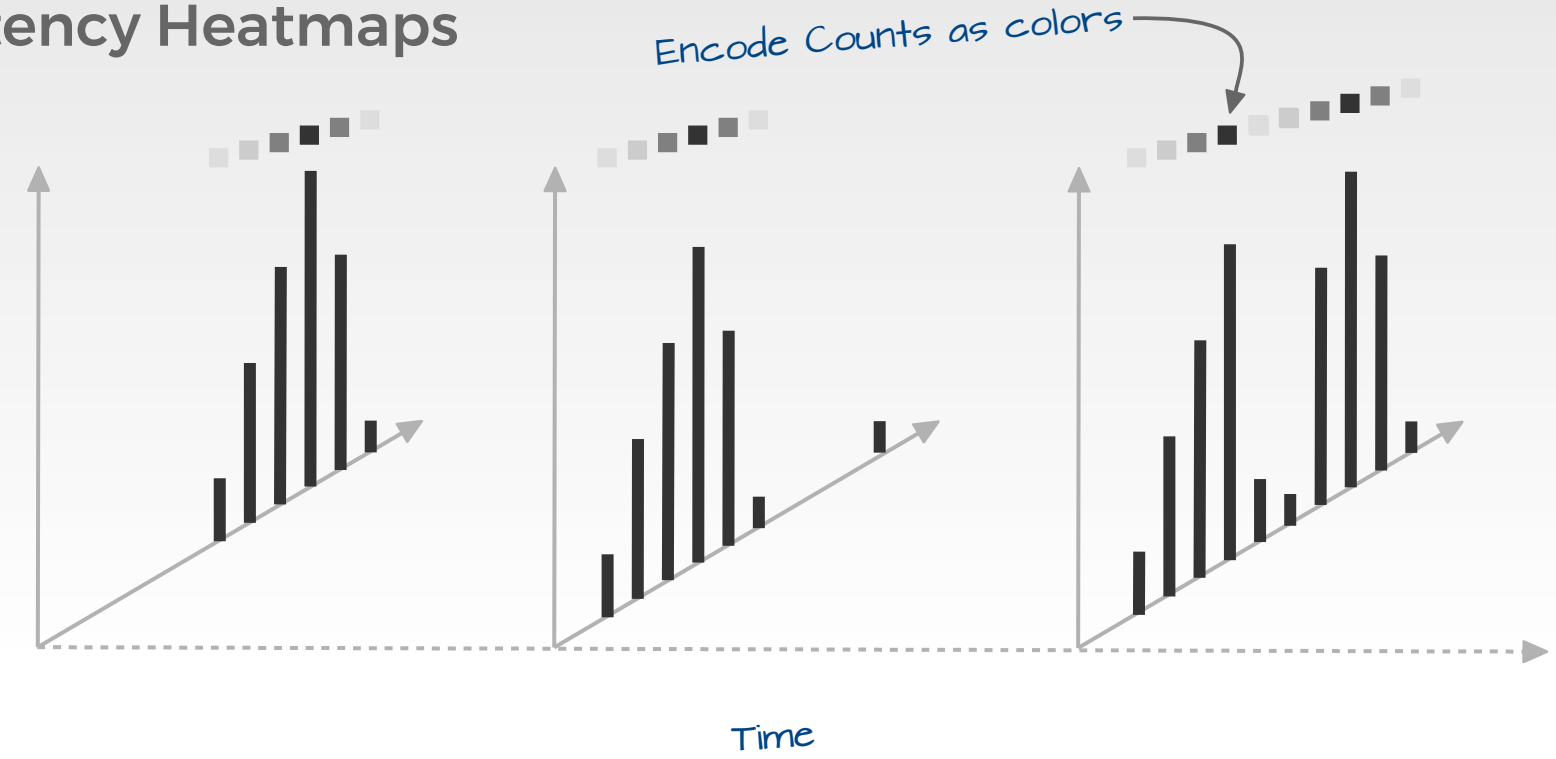
Heatmaps

Latency Heatmaps



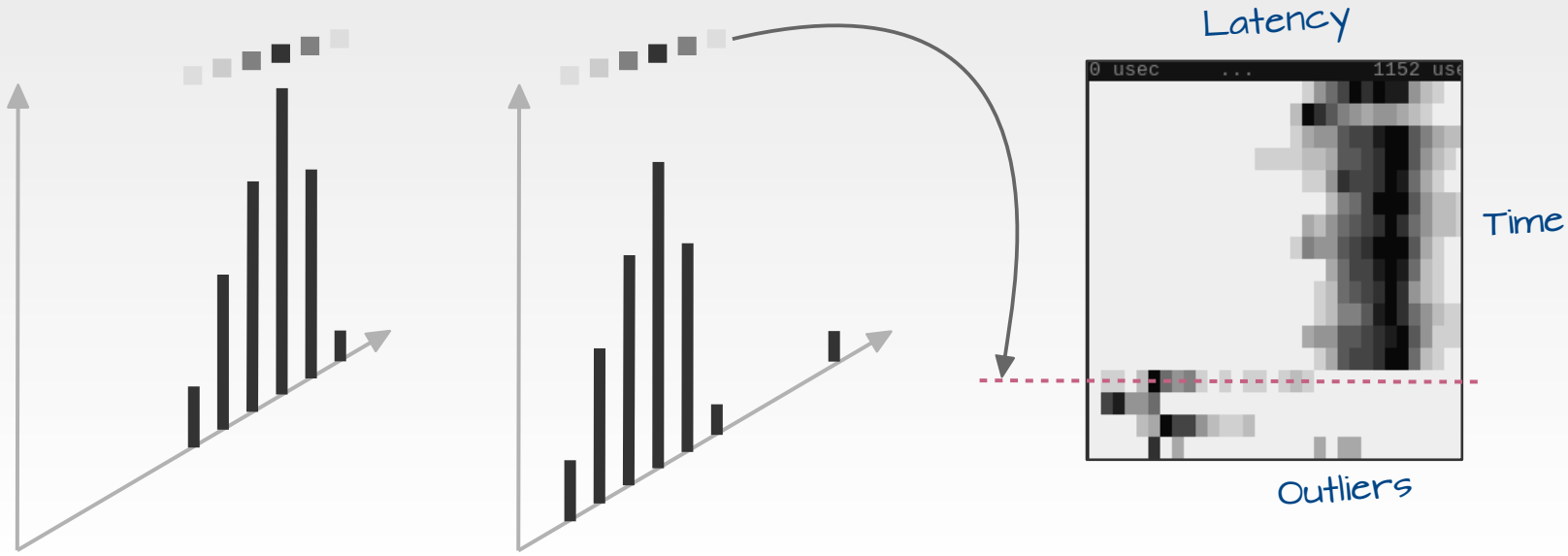
Heatmaps

Latency Heatmaps



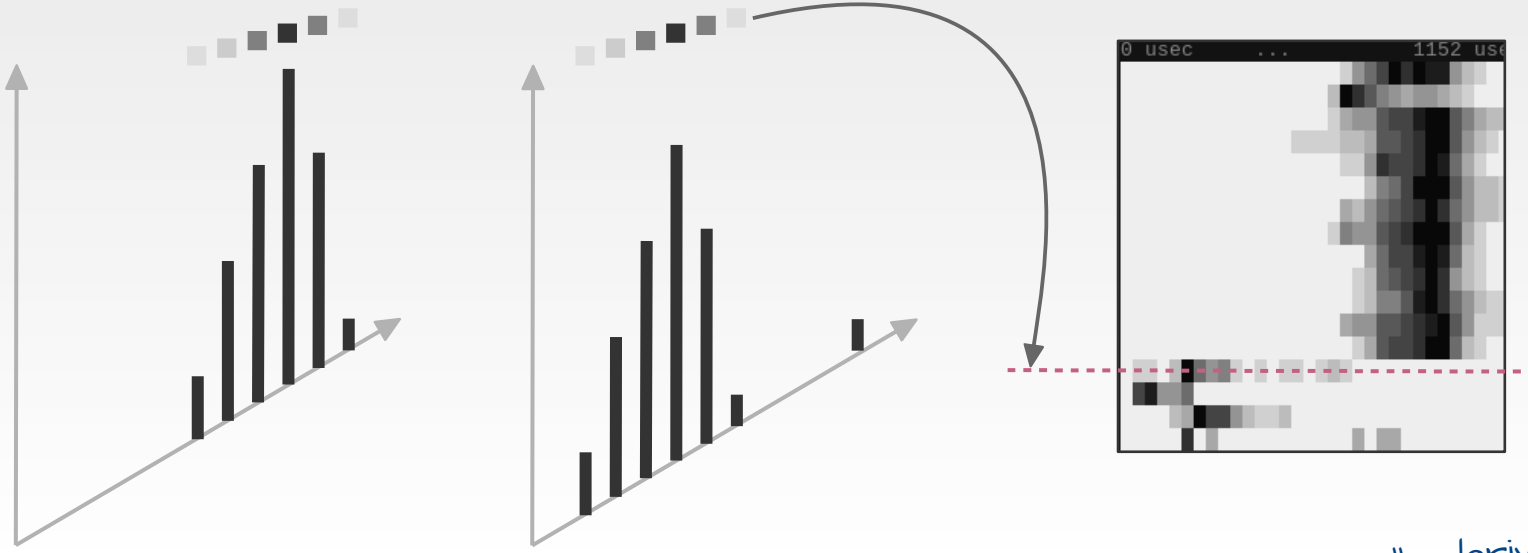
Heatmaps

Latency Heatmaps



Heatmaps

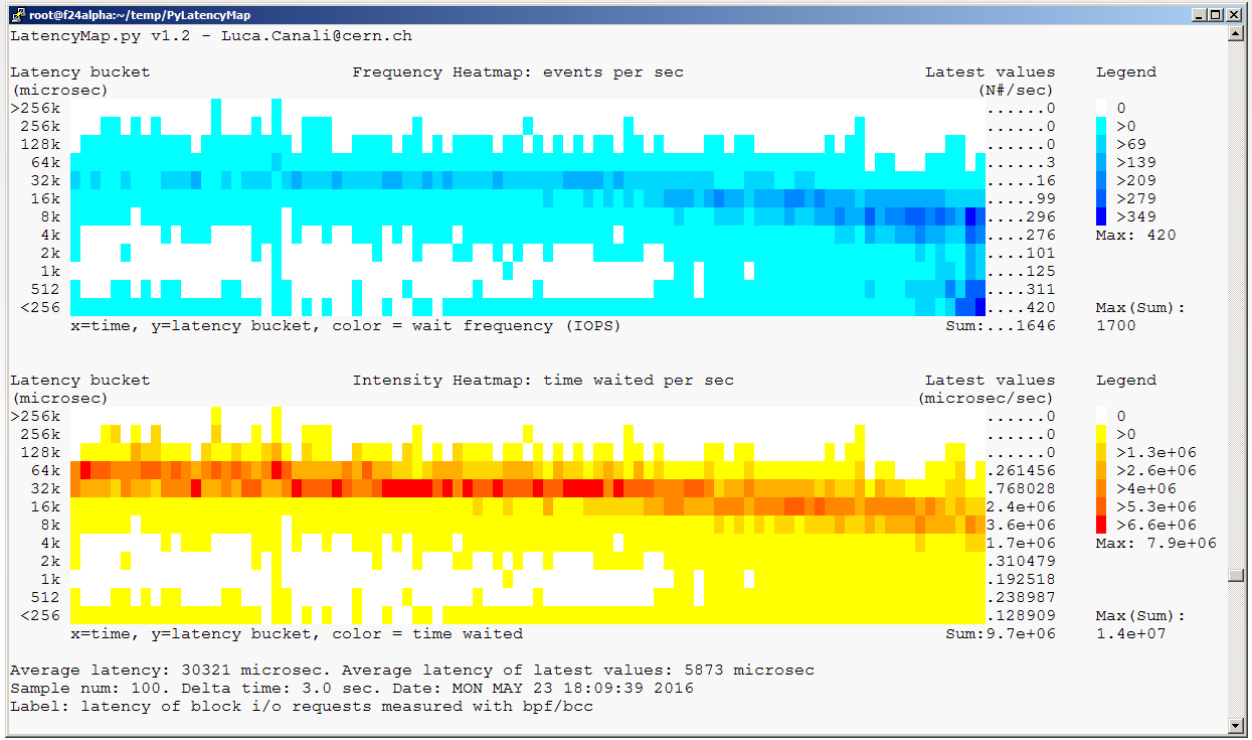
Latency Heatmaps



Btw, this is really derived from an eBPF tool

Heatmaps

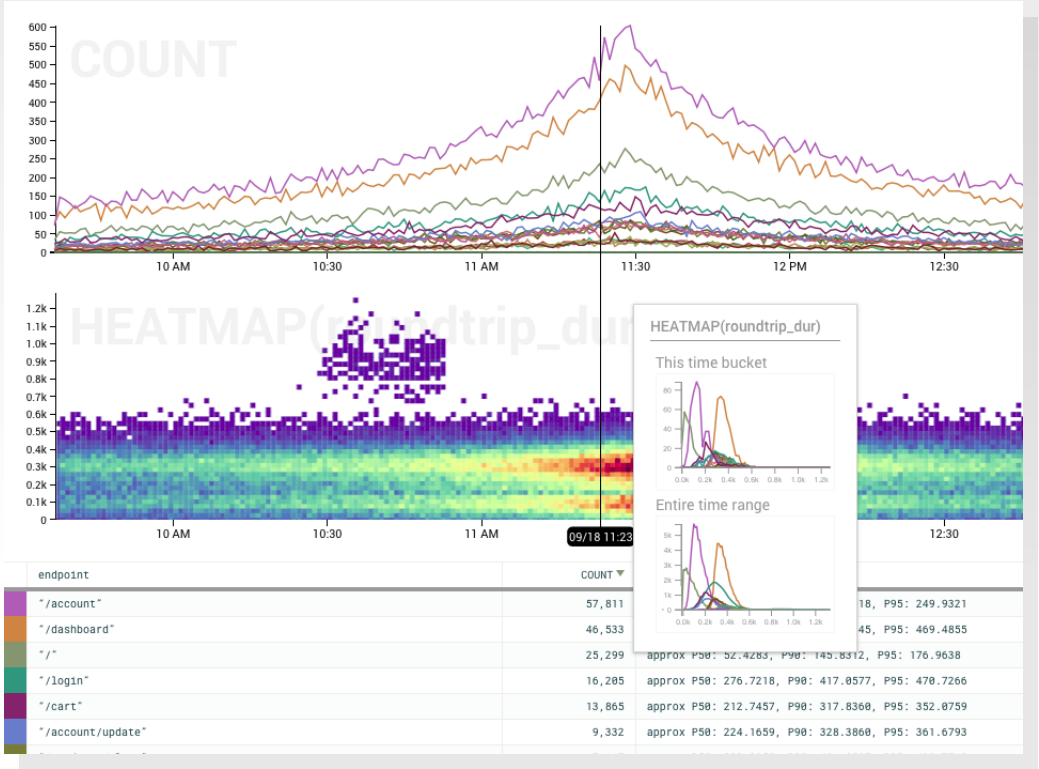
Moaaaarr! Heatmaps



Luca Canali from CERN uses this!

Heatmaps

Moaaaarr! Heatmaps



..and apparently Honeycomb users as well..

Netflix as well!

https://honeycomb.io/blog/2017/09/heatmaps-are-the-new-hotness/

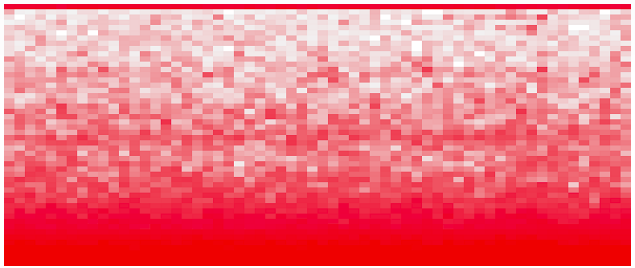
More Stuff

More Latency Visualizations

- http://www.brendangregg.com/HeatMaps/latency.html
- http://queue.acm.org/detail.cfm?id=1809426
- https://honeycomb.io/blog/2017/09/heatmaps-are-the-new-hotness/
- https://github.com/LucaCanali/PyLatencyMap

Extra Reading: Utilization Heatmaps

- http://www.brendangregg.com/HeatMaps/utilization.html



Flames

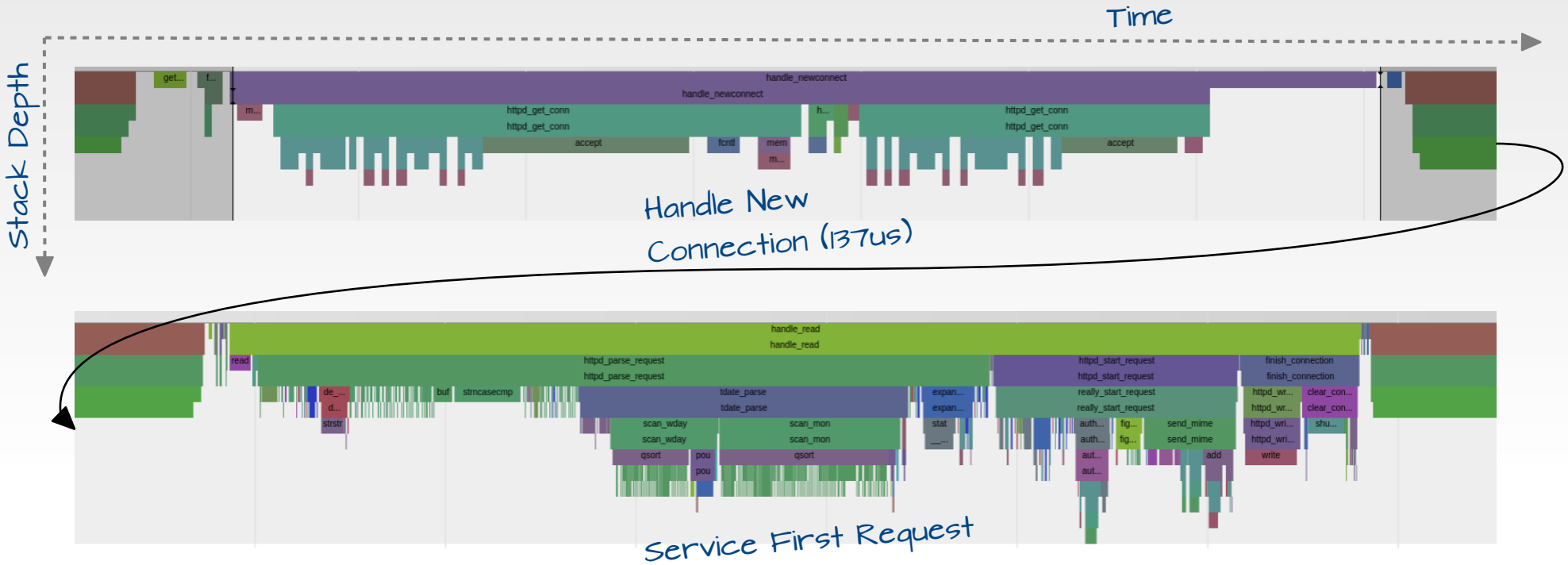
Function Graph (Uftrace)

snippet (thttpd request-response)

```
  6.198 ms [ 8898] |           } /* poll_init */
  6.203 ms [ 8898] |         } /* poll_init */
  8.656 ms [ 8898] |       } /* fdwatch_get_nfiles */
  8.657 ms [ 8898] |     } /* fdwatch_get_nfiles */
 18.768 us [ 8898] |     chroot();
  2.505 us [ 8898] |     strncmp();
           [ 8898] |     syslog() {
727.596 us [ 8898] |       __syslog_chk();
730.243 us [ 8898] |     } /* syslog */
           [ 8898] |     fprintf() {
  1.194 us [ 8898] |       __fprintf_chk();
  2.664 us [ 8898] |     } /* fprintf */
  0.430 us [ 8898] |     strcpy();
  4.179 us [ 8898] |     chdir();
  8.285 us [ 8898] |     sigset();
  1.497 us [ 8898] |     sigset();
  1.385 us [ 8898] |     sigset();
  6.322 us [ 8898] |     alarm();
           [ 8898] |     tmr_init() {
  0.667 us [ 8898] |       tmr_init();
  1.782 us [ 8898] |     } /* tmr_init */
           [ 8898] |     httpd_initialize() {
           [ 8898] |       httpd_initialize() {
           [ 8898] |         check_options() {
  0.435 us [ 8898] |           check_options();
  1.501 us [ 8898] |         } /* check_options */
  1.342 us [ 8898] |         malloc();
           [ 8898] |         gethostname() {
  4.893 us [ 8898] |           gethostname();
  6.201 us [ 8898] |         } /* gethostname */
  0.910 us [ 8898] |         __strdup();
  0.744 us [ 8898] |         __strdup();
  0.572 us [ 8898] |         __strdup();
           [ 8898] |         httpd_set_logfp() {
  0.448 us [ 8898] |           httpd_set_logfp();
  1.393 us [ 8898] |         } /* httpd_set_logfp */
```

Flames

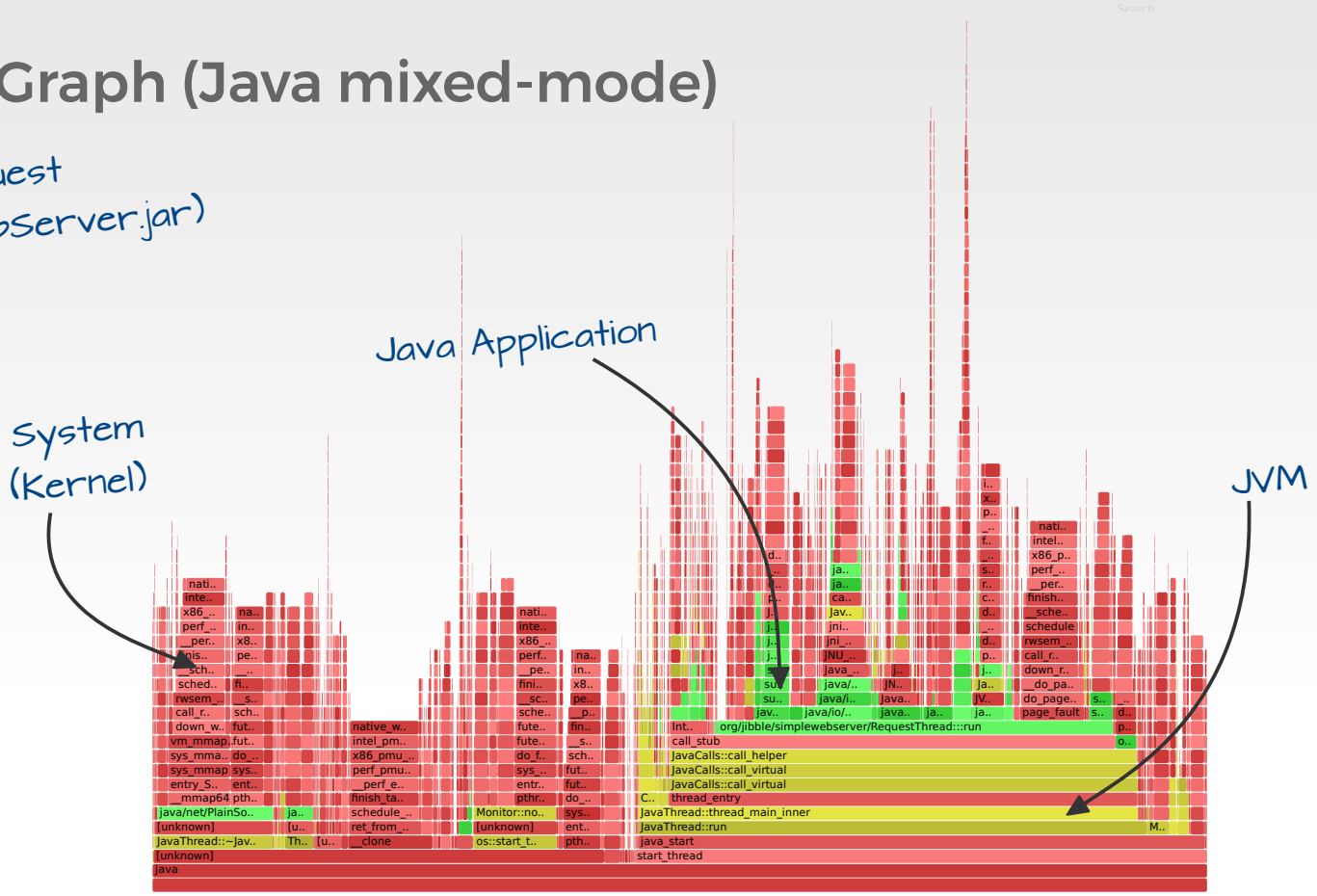
Flame Chart (Uftrace)



https://github.com/namhyung/uftrace

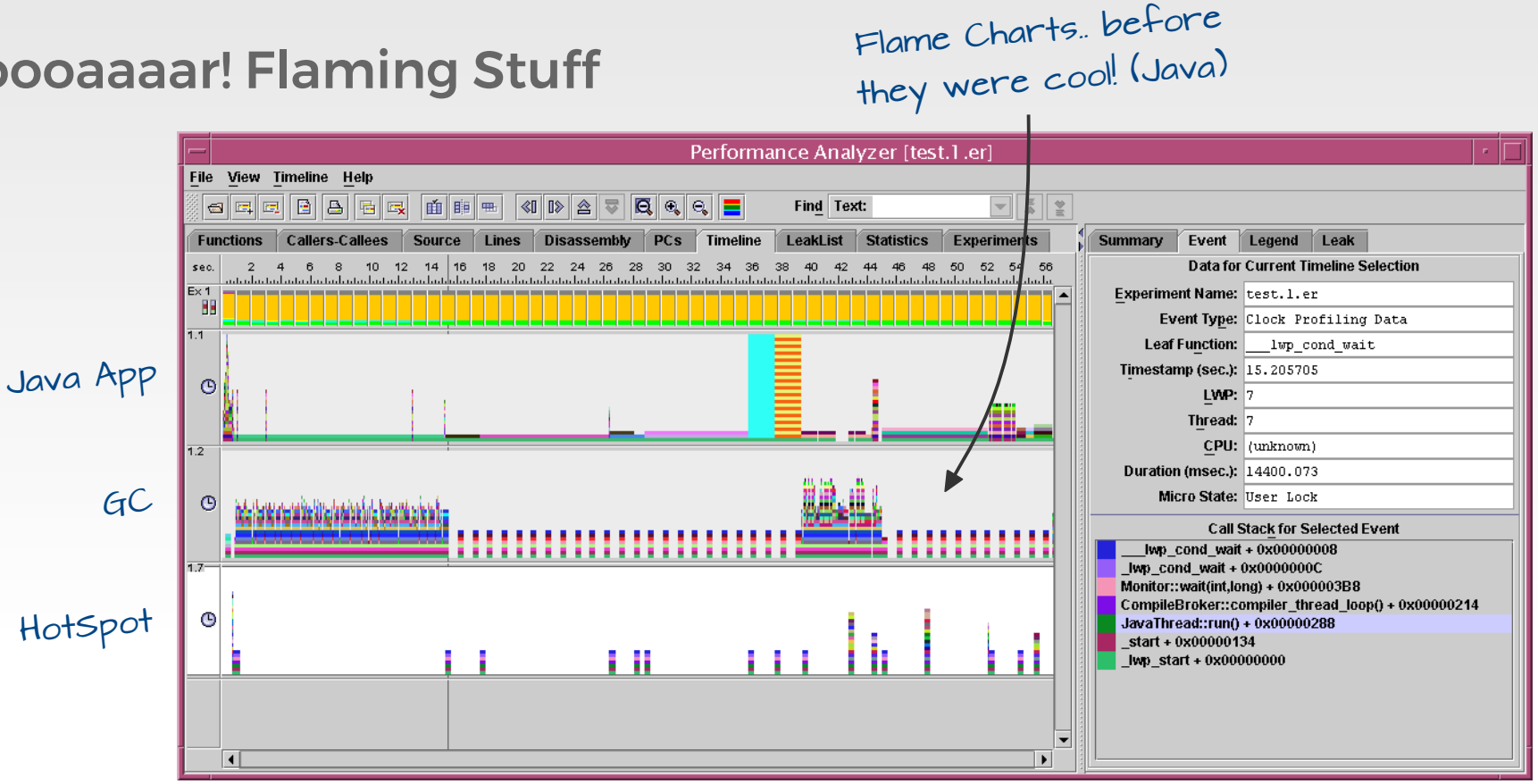
Flames

Flame Graph (Java mixed-mode)



Old Flames

Moooaaaar! Flaming Stuff



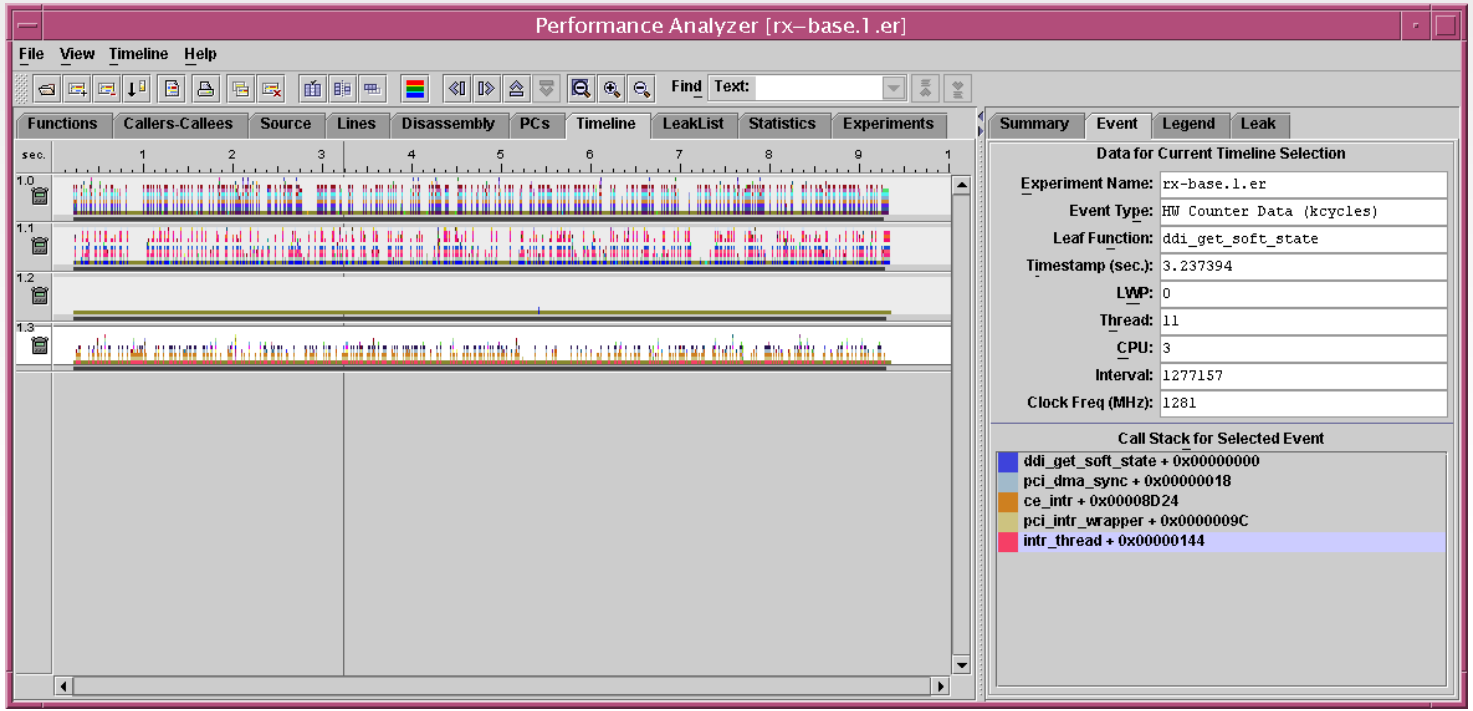
http://www.oracle.com/technetwork/server-storage/solaris/perftools-141913.html

Old Flames

Moooaaaar! Flaming Stuff

DTrace + Solaris
Kernel (CPU Stacks)

CPUs



http://www.oracle.com/technetwork/server-storage/solaris/perftools-141913.html

Flames

Flame Charts

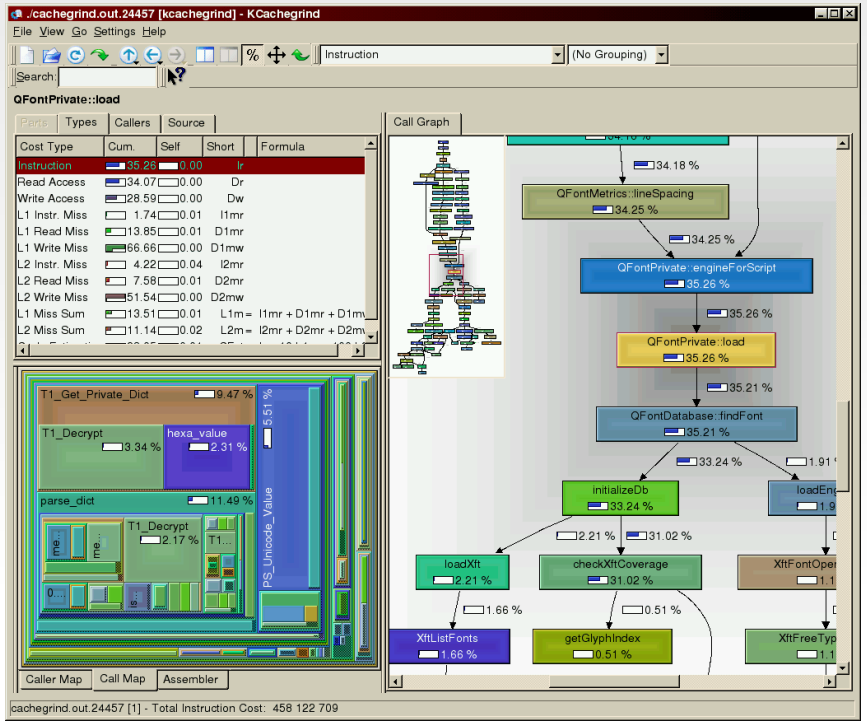
- Function Graphs (Time): Keeps on running. Horizontal-axis is time.
- Sourced from Chrome traces, LTTng traces, Uftrace

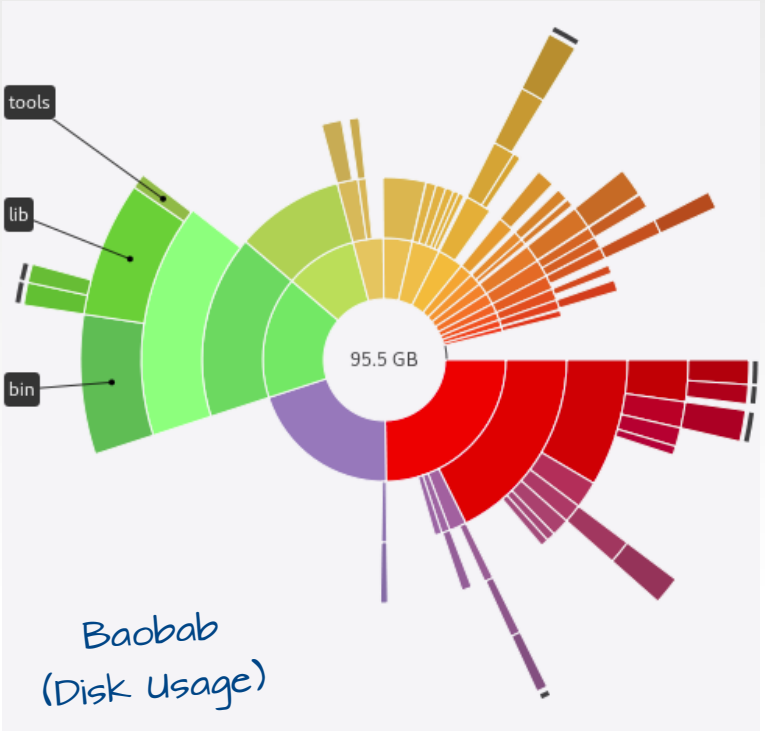
Flame Graphs

- Function Graphs (Count): Callstacks visualized during execution with population on horizontal-axis and depth of stack on vertical-axis
- Sourced from perf events, eBPF stack traces, LTTng traces

Callgraphs, Treemaps & Sunbursts

Valgrind + Callgrind

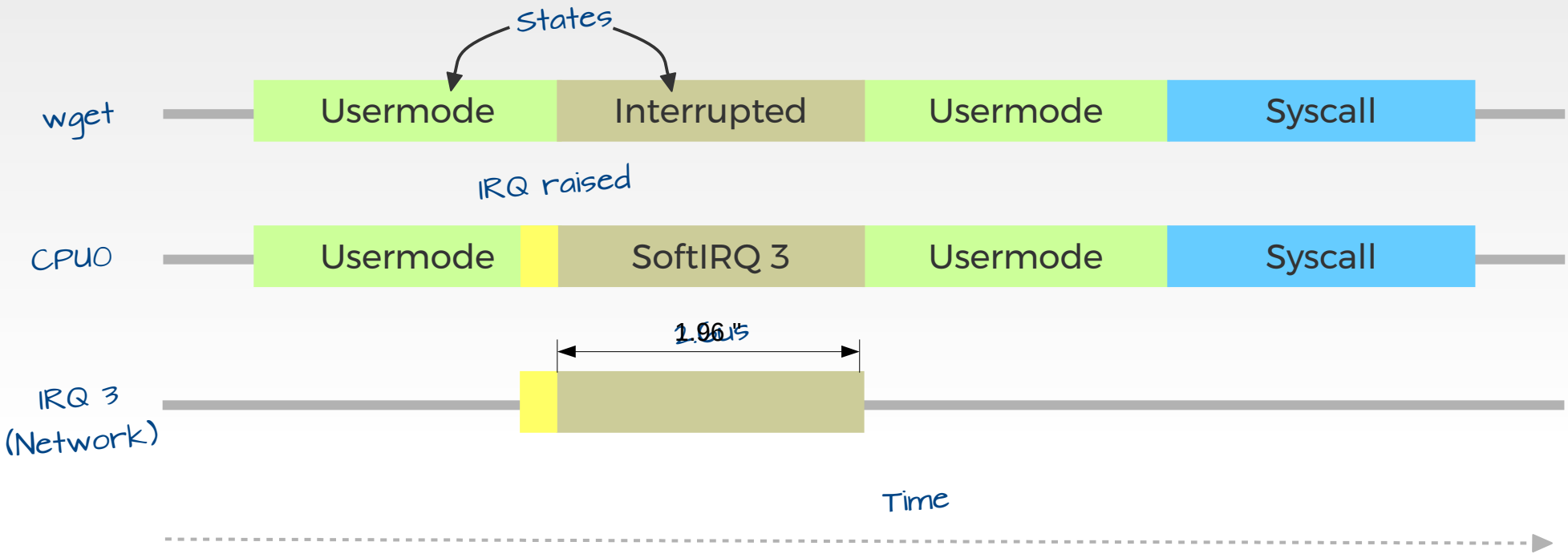




https://stackoverflow.com/a/6617588

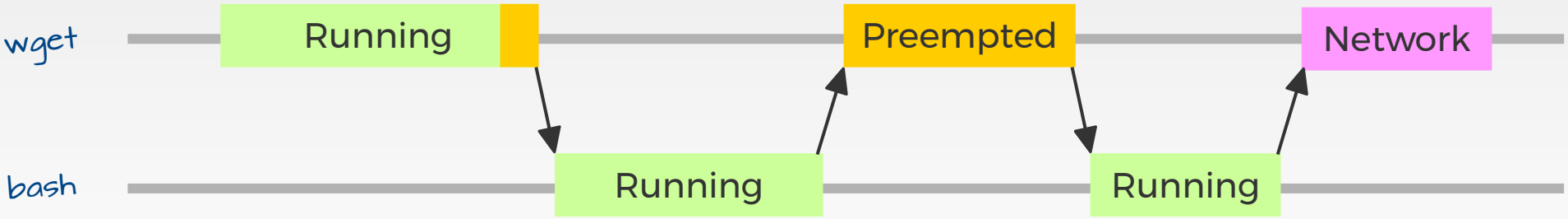
Timelines

Control Flow (LTTng/Custom Trace)



Timelines

Critical Flow (LTTng)



Identify the critical path of execution by following a process' execution and its relation with other processes



Tooling **Demo**



tracecompass.org

Short Note on **Colors**

Colors

Diverging Trends

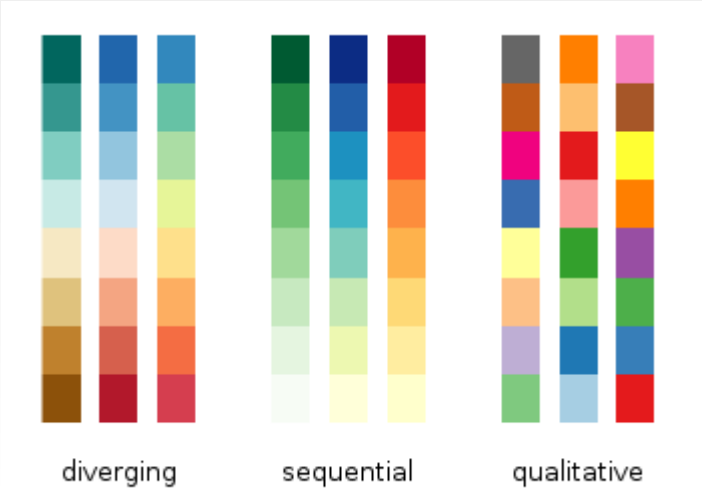
- Think of it as temperature (heatmaps)

Sequential Trend

- Related but changing (heatmaps, flamegraphs)

Qualitative Trend

- Distinct entities (process states, timelines)



http://colorbrewer2.org

What Next?

Wishlist for an Ultimate Analysis Pipeline

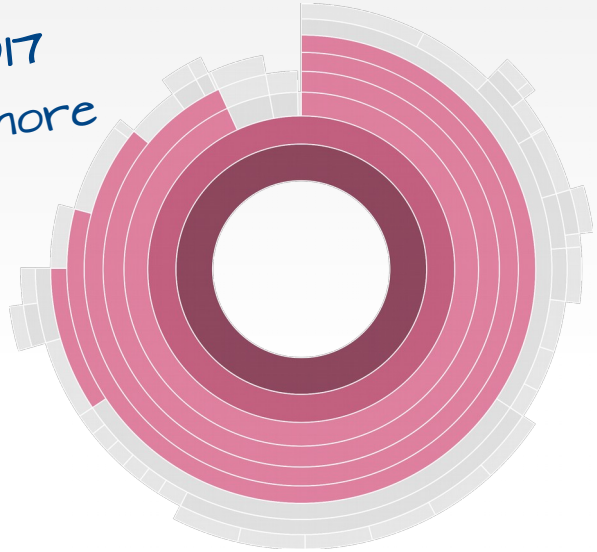
- Pluggable sources
- Distributed systems support
- Diffs of executions
- Time-synchronized views
- Summarization of events and observations
- Intuitive UIs
- Filtering
- Hardware Traces! *Lolwut?*

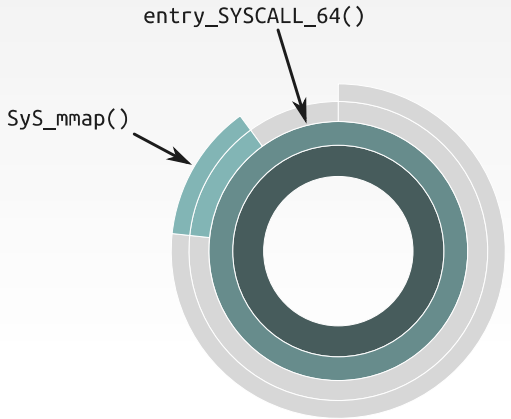
Future Fun

Hardware Trace Visualizations

- Targeted Intel Processor Trace (PT) snapshot of `mmap()`
- Sunburst of syscall latency (insn). *Tracerception!*

173 ns and 917 instructions more





Future Fun

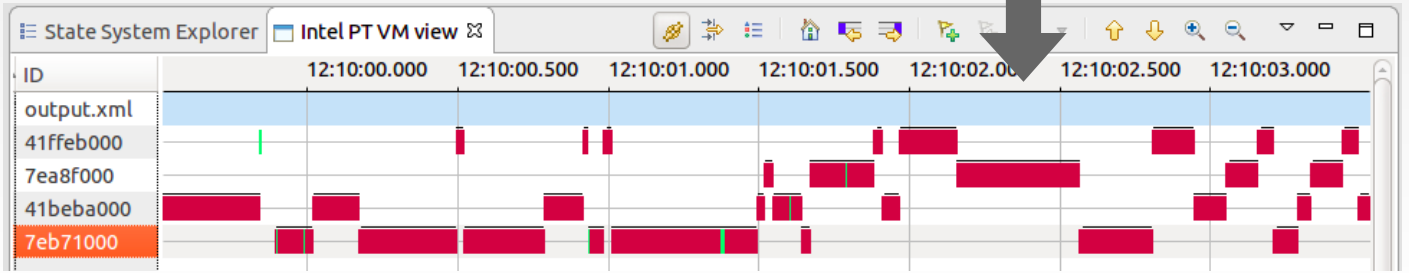
Data Driven Visuals (Intel PT)

Perf PT traces

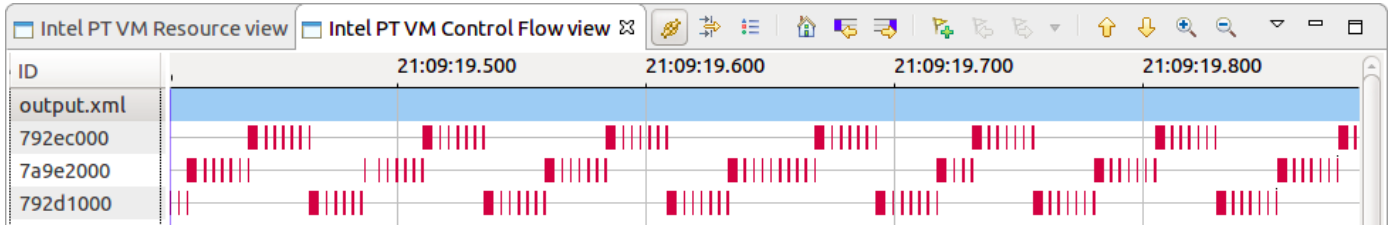
perf.data ➡ PTParse + VMPT ➡

```
<bundle>
      <PIP> 32423545 </PIP>
      <NR> 1 </NR>
      <VMCS> 243241334 </VMCS>
      <TSC> 2342353646 </TSC>
</bundle>
```

vCPUs Consumption



Process Control Flow



More References

- **Linux Performance/Tracing**
  - http://www.brendangregg.com/linuxperf.html
  - https://jvns.ca/blog/2017/07/05/linux-tracing-systems/
  - https://www.kernel.org/doc/Documentation/trace/ftrace.txt
  - https://github.com/namhyung/uftrace
  - http://lttng.org/docs
  - http://tracecompass.org/#docs
  - https://www.kernel.org/doc/Documentation/kprobes.txt
  - https://github.com/jvm-profiling-tools/perf-map-agent
  - https://github.com/goldshtn/linux-tracing-workshop

More References

- **Visualizations**
  - *A Tour Through the Visualization Zoo*
    http://queue.acm.org/detail.cfm?id=1805128
  - https://www.slideshare.net/brendangregg/javaone-2015-java-mixedmode-flame-graphs

More References

- **Research and Future**

[Begel et al. 1999] BPF+: exploiting global data-flow optimization in a generalized packet filter architecture, *ACM SIGCOMM '99*

[Desnoyers et al. 2009] The LTTng tracer: A low impact performance and behavior monitor for GNU/Linux, *Ottawa Linux Symposium 2009*

[Desnoyers et al.] *Low-Impact Operating Systems Tracing, PhD Thesis (polymtl.ca)*

[Clement 2016] Linux Kernel packet transmission performance in high-speed networks *Masters Thesis (2016), KTH, Stockholm*

[Borkmann 2016] Advanced programmability and recent updates with tc's cls_bpf *NetDev 1.2 (2016) Tokyo*

More References

- **Research and Future**

[Philippe Moret et al. 2010] Visualizing and exploring profiles with calling context ring charts, *Sofware: Practice and Experience, Wiley*

[J. Trümper et al.] Multiscale Visual Comparison of Execution Traces. *Proceedings of the International Conference on Program Comprehension, IEEE Computer Society, 2013.*

[Johannes Bohnet et. al.] Analyzing Feature Implementation by Visual Exploration of Architecturally-Embedded Call-Graphs. *4th International Workshop on Dynamic Analysis, ACM, 2006.*

[Sharma 2017] Low-Impact System Performance Analysis Using Hardware Assisted Tracing Techniques, *PhD Thesis* (*polymtl.ca*)

Ack

- LTTng developers
- eBPF and BCC developers
- Trace Compass developers
- Geneviève Bastien, Mohamad Gebai, Francis Giraldeau and Hani Nemati!
- Whole **ShiftLeft Inc.** team!
- DORSAL Lab @ Polytechnique Montreal



CC BY SA

Fin

suchakra@shiftleft.io
suchakrapani.sharma@polymtl.ca
@tuxology