Now You See Me Too
Visual Tooling for Advanced Systems Analysis

Suchakra Sharma
2nd Nov 2017 - San Francisco, CA
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
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)

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
Visualizing Data

Brief Historical Perspective

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

Velocity of an object with time
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
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?q=289678&p=1930713
  - https://www.slideshare.net/amzoss/everything-data
- Udemy Course
Software and Systems
Visualizations
Get Inspired
Get Inspired

...again
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

e.g. 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

e.g. 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**

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

**Gather**

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

**Question**

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

**Gather**

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

**Characterize**

Frequency Analysis

Latency Analysis

Multi-modal Analysis (Hierarchy + Time)

**Sample or Trace**

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

**Which? Why? Where?**
Visualizing Systems Data

From Problem to Insight

**Question**

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

**Which? Why? Where?**

**Gather**

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

**Characterize**

Frequency Analysis

Latency Analysis

Multi-modal Analysis (Hierarchy + Time)

**Visualize**

Histograms, Scatter-plots

Heat-maps

Graphs, Timelines, Flamecharts, Flamegraphs

Sample or Trace
How would the code flow in my web-server?

What are terminal sources and sinks?

Questions

Conception

Gather

Visualize

Callgraph Extraction From RTL

Flow Classification

CFLAGS=-fdump-rtl-expand

thttpd.c

thttpd.c.212r.expand
Case Study - Live

How do we observe disk I/O latencies?

Questions

Visualize Systems Data

eBPF Program

kprobe/kretprobes

req_start/stop

HISTOGRAM MAP

Gather

Characterize

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

**************
***************************
****************************************
**************************
*
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?

SimpleWebServer.jar

perf-map-agent

Perf

perf-<pid>.data

Callstack Extraction (count + hierarchy)

Gather

Characterize

Visualize
Gathering Systems Data
Observability Layers
Observability Layers

<table>
<thead>
<tr>
<th>Host OS</th>
<th>Kernel functions, hardware perf counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td></td>
</tr>
</tbody>
</table>
Observability Layers

- Guest OS
  - Virtualization
  - Containerization
- Host OS
  - Hardware

Kernel functions, Hypervisor, Custom APIs
Kernel functions, hardware perf counters
Observability Layers

- Library functions, Syscalls
- Kernel functions, Hypervisor, Custom APIs
- Kernel functions, hardware perf counters
Observability Layers

- **Functions**
  - Kernel functions, hardware
  - Hypervisor, Custom APIs
  - Library functions, Syscalls
  - Kernel functions, hardware, perf counters

- **Applications**
  - Application functions

- **Libs**

- **Guest OS**

- **Virtualization**

- **Containerization**

- **Host OS**

- **Hardware**

Suchakrapani Datt Sharma
Observability Layers

- **Instructions**
  - Branches, calls
- **Functions**
  - Application functions
  - Library functions, Syscalls
- **Applications**
  - Kernel functions, Hypervisor, Custom APIs
  - Kernel functions, hardware perf counters
- **Libs**
- **Guest OS**
- **Virtualization**
  - Host OS
- **Containerization**
- **Hardware**

- **Observability Layers**
  - Kernel functions, hardware perf counters
  - Hypervisor, Custom APIs
  - Library functions, Syscalls
  - Branches, calls
Observability Layers

Subatomic Particles
- Kernel functions, hardware
- Perf counters

Logic
- Hypervisor, Custom APIs

Instructions
- Library functions, Syscalls
- Application functions

Functions
- Kernel functions, Hypervisor, Custom APIs
- Kernel functions, hardware perf counters

Applications
- EM waves ??
- CPU pins
- Branches, calls

Host OS

Virtualization

Containerization

Libs

Guest OS

Hardware

Libs
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

foo() 

bar() 

call_me_maybe() 

baz() 

call_me_maybe() 

Collect Data 

Fill Buffer 

Event 

Event 

Event 

Program Flow, Arguments, Latency, Data Flow
Observing Functions

Kernel Example
- Dynamically instrument with Kprobes

```
do_sys_open()
jmp
Save registers
CALL pre_handler
Restore registers
ORIG INSN
JMP back
Custom handler
eBPF program
```

Patched Instruction

Supported by eBPF
(Stored in Maps → Perf Data), LTTng (CTF)
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
```

Supported by LTTng

Suchakrapani Datt Sharma
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

BPF Program

- LLVM/Clang
- trace.bpf
- bpf()

Kernel Function

- Kprobe

BPF Code

Verifier + JIT

BPF Map

Read/Update

bpf()

Monitor/Store

Read Events

Trace Pipe

Perf Buffer

Userspace

Kernel

Suchakrapani Datt Sharma
Observing Functions

Kernel Example
- Dynamically instrument with Kprobes

Patched Instruction

```
do_sys_open()
```

Save registers
CALL pre_handler
Restore registers
ORIG INSN
JMP back

Custom handler
eBPF program

Supported by eBPF
(Stored in Maps → Perf Data), LTTng (CTF)
Observing Functions

eBPF + Kprobes

**BPF Program**
- LLVM/Clang
- `trace.bpf`
- `bpf()`

**Verifier + JIT**

**Kernel Function**

**BPF Code**

**Kprobe**

**BPF Program**
- LLVM/Clang
- `trace.bpf`
- `bpf()`

**Verifier + JIT**

**Kernel Function**

**BPF Code**

**Kprobe**

**BPF Map**

**Trace Pipe**

**Perf Buffer**

**Userspace**

**Kernel**

**Monitor/Store**

**Read Events**

Suchakrapani Datt Sharma
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

```python
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\}"")
```

---

Suchakrapani Datt Sharma
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

$ sudo ./biolatency.py

<table>
<thead>
<tr>
<th>usecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>0</td>
<td>***</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>0</td>
<td>***</td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>0</td>
<td>***</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>115</td>
<td>**</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>553</td>
<td>***********</td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>1087</td>
<td>*******************</td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>1572</td>
<td>*******************</td>
</tr>
<tr>
<td>128 -&gt; 255</td>
<td>1040</td>
<td>*******************</td>
</tr>
<tr>
<td>256 -&gt; 511</td>
<td>54</td>
<td>*</td>
</tr>
<tr>
<td>512 -&gt; 1023</td>
<td></td>
<td>**</td>
</tr>
</tbody>
</table>
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

Count

Latency (us)
Heatmaps

Latency Heatmaps

Time
Heatmaps

Latency Heatmaps

- Outlier
- Bi-modal
- Time
Heatmaps

Latency Heatmaps

Averages don’t do justice

Bi-modal

Outlier

Time
Heatmaps

Latency Heatmaps

Time
Heatmaps

Latency Heatmaps

Encode Counts as colors

Time
Heatmaps

Latency Heatmaps

Latency
Time
Outliers
Heatmaps

Latency Heatmaps

Btw, this is really derived from an eBPF tool
Heatmaps

Moaaaarr! Heatmaps

Luca Canali from CERN uses this!
Heatmaps

Moaaaarr! Heatmaps

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

.and apparently Honeycomb users as well.

Netflix as well!
More Stuff

More Latency Visualizations
- http://queue.acm.org/detail.cfm?id=1809426
- https://github.com/LucaCanali/PyLatencyMap

Extra Reading: Utilization Heatmaps
<table>
<thead>
<tr>
<th>Time</th>
<th>Function</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.198 ms</td>
<td><code>chroot();</code></td>
<td>6.198 ms</td>
</tr>
<tr>
<td>6.203 ms</td>
<td><code>strncpy();</code></td>
<td>6.203 ms</td>
</tr>
<tr>
<td>8.656 ms</td>
<td><code>syslog();</code></td>
<td>8.656 ms</td>
</tr>
<tr>
<td>8.657 ms</td>
<td><code>gethostname();</code></td>
<td>8.657 ms</td>
</tr>
<tr>
<td>18.768 us</td>
<td><code>chroot();</code></td>
<td></td>
</tr>
<tr>
<td>2.505 us</td>
<td><code>strncpy();</code></td>
<td></td>
</tr>
<tr>
<td>1.194 us</td>
<td><code>__fprintf_chk();</code></td>
<td>1.194 us</td>
</tr>
<tr>
<td>2.664 us</td>
<td><code>__fprintf_chk();</code></td>
<td>2.664 us</td>
</tr>
<tr>
<td>0.430 us</td>
<td><code>memcpy();</code></td>
<td>0.430 us</td>
</tr>
<tr>
<td>4.179 us</td>
<td><code>chdir();</code></td>
<td>4.179 us</td>
</tr>
<tr>
<td>8.285 us</td>
<td><code>sigset();</code></td>
<td>8.285 us</td>
</tr>
<tr>
<td>1.497 us</td>
<td><code>sigset();</code></td>
<td>1.497 us</td>
</tr>
<tr>
<td>1.385 us</td>
<td><code>sigset();</code></td>
<td>1.385 us</td>
</tr>
<tr>
<td>6.322 us</td>
<td><code>alarm();</code></td>
<td>6.322 us</td>
</tr>
<tr>
<td>0.667 us</td>
<td><code>tmr_init();</code></td>
<td>0.667 us</td>
</tr>
<tr>
<td>1.782 us</td>
<td><code>httpd_initialize();</code></td>
<td>1.782 us</td>
</tr>
<tr>
<td>0.435 us</td>
<td><code>check_options();</code></td>
<td>0.435 us</td>
</tr>
<tr>
<td>1.501 us</td>
<td><code>malloc();</code></td>
<td>1.501 us</td>
</tr>
<tr>
<td>0.910 us</td>
<td><code>__strdup();</code></td>
<td>0.910 us</td>
</tr>
<tr>
<td>0.744 us</td>
<td><code>__strdup();</code></td>
<td>0.744 us</td>
</tr>
<tr>
<td>0.572 us</td>
<td><code>__strdup();</code></td>
<td>0.572 us</td>
</tr>
<tr>
<td>0.448 us</td>
<td><code>httpd_set_logfp();</code></td>
<td>0.448 us</td>
</tr>
<tr>
<td>1.393 us</td>
<td><code>httpd_set_logfp();</code></td>
<td>1.393 us</td>
</tr>
<tr>
<td>1.393 us</td>
<td><code>httpd_set_logfp();</code></td>
<td></td>
</tr>
<tr>
<td>0.448 us</td>
<td><code>httpd_set_logfp();</code></td>
<td></td>
</tr>
<tr>
<td>1.782 us</td>
<td><code>httpd_initialize();</code></td>
<td></td>
</tr>
<tr>
<td>0.435 us</td>
<td><code>check_options();</code></td>
<td></td>
</tr>
<tr>
<td>1.501 us</td>
<td><code>malloc();</code></td>
<td></td>
</tr>
<tr>
<td>0.910 us</td>
<td><code>__strdup();</code></td>
<td></td>
</tr>
<tr>
<td>0.744 us</td>
<td><code>__strdup();</code></td>
<td></td>
</tr>
<tr>
<td>0.572 us</td>
<td><code>__strdup();</code></td>
<td></td>
</tr>
<tr>
<td>0.448 us</td>
<td><code>httpd_set_logfp();</code></td>
<td></td>
</tr>
<tr>
<td>1.393 us</td>
<td><code>httpd_set_logfp();</code></td>
<td></td>
</tr>
</tbody>
</table>

Snippets from the function graph (Uftrace)

- `tmr_init()`
- `httpd_initialize()`
Flames

Flame Chart (Uftrace)

Suchakrapani Datt Sharma

https://github.com/namhyung/uftrace
Flames

Flame Graph (Java mixed-mode)

Single Request
(SimpleWebServer.jar)

System
(Kernel)

Java Application

JVM
Old Flames

Mooolaaar! Flaming Stuff

Flame Charts.. before they were cool! (Java)

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

Moooaar! Flaming Stuff

DTrace + Solaris Kernel (CPU Stacks)

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

Suchakrapani Datt Sharma
Timelines

Control Flow (LTTng/Custom Trace)

Usermode → Interrupted → Usermode → Syscall

IRQ raised

Usermode → SoftIRQ 3 → Usermode → Syscall

Time

States

wget

CPU0

IRQ 3 (Network)
Identify the critical path of execution by following a process’ execution and its relation with other processes.
Tooling Demo
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.data → PTParse + VMPT →

Perfil
PT traces

vCPUs
Consumption

Process
Control Flow

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

- Linux Performance/Tracing
  - 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://lttnq.org/docs
  - http://tracecompass.org/#docs
  - 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
More References

- Research and Future


[Borkmann 2016] Advanced programmability and recent updates with tc’s cls_bpf NetDev 1.2 (2016) Tokyo
More References

- Research and Future


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
Fin

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