Precursors of Security and Performance

Instrumentation and Tracing of Systems

Suchakra Sharma

17th August 2017

Security and DevOps Meetup, Santa Clara

ShiftLeft

Le Plan

Observability Layers

Instrumentation of Systems

- Challenges
- Techniques

Foundations

- Performance
- Security

Security Tooling

- Preventive (Isolation)
- Passive (Monitoring)
- Active (Protection)

whoami

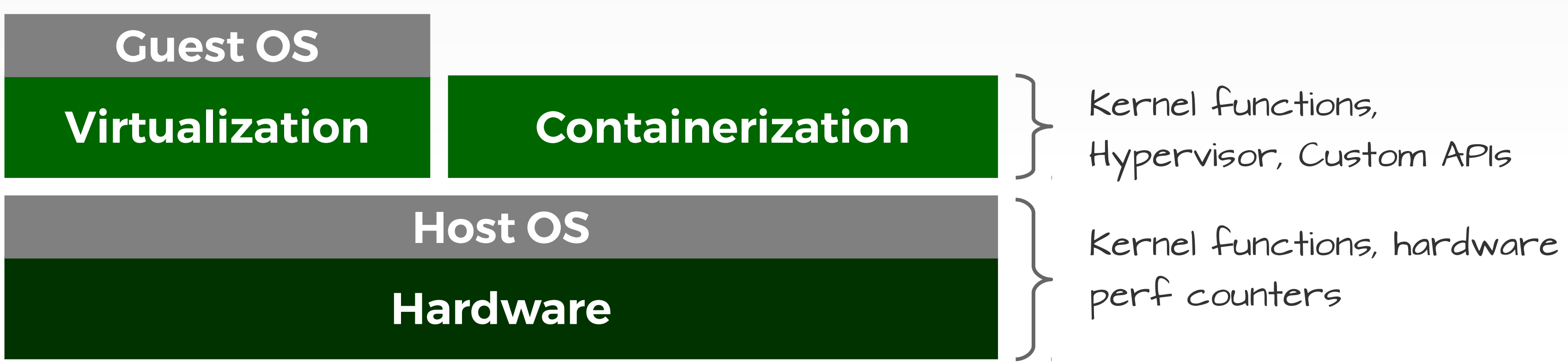
Suchakra

- Staff Scientist, ShiftLeft Inc.
- PhD, DORSAL Lab, *Polytechnique Montréal - University of Montréal*
- Loves tracing, security, performance analysis, hardware dev, poutine and samosas
- @tuxology

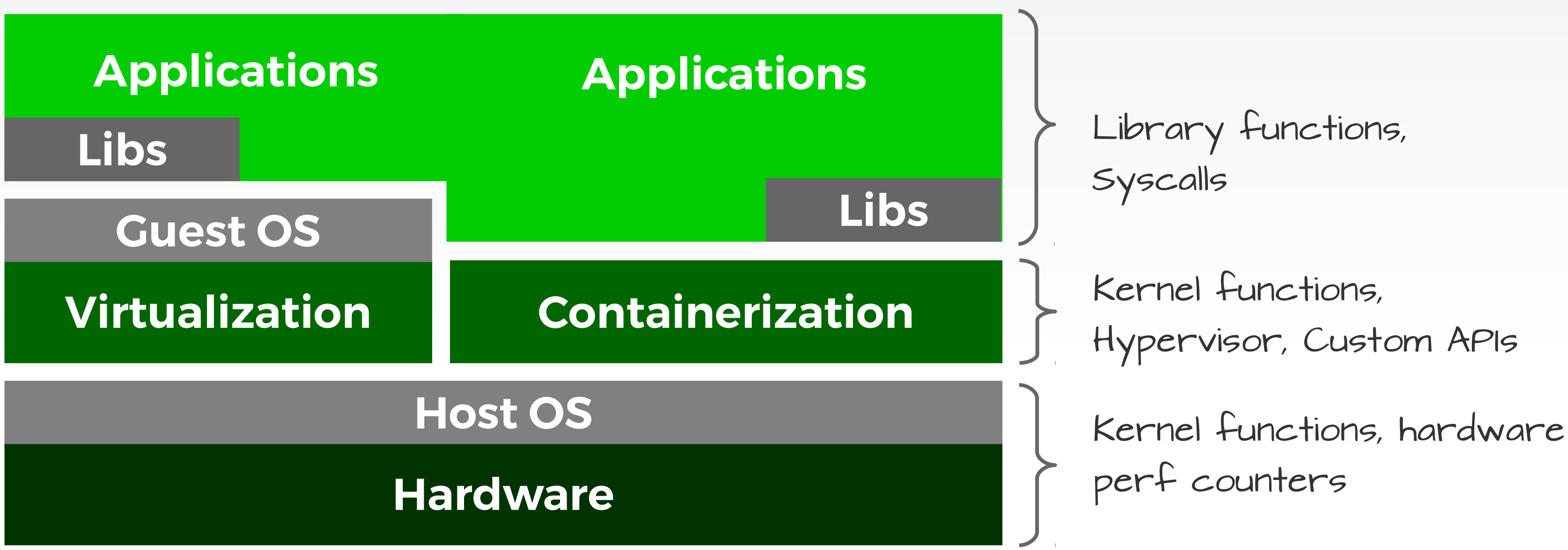
Observability Layers

Observability Layers

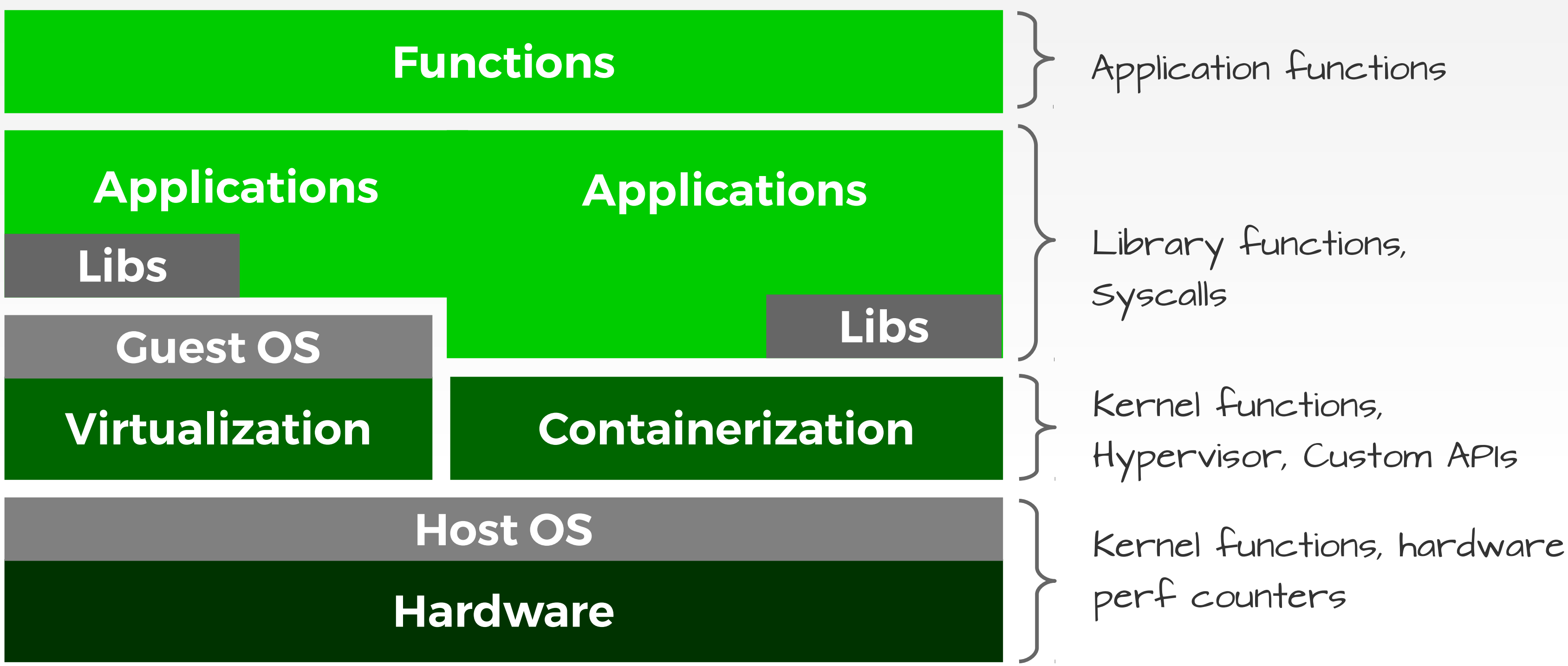
Observability Layers



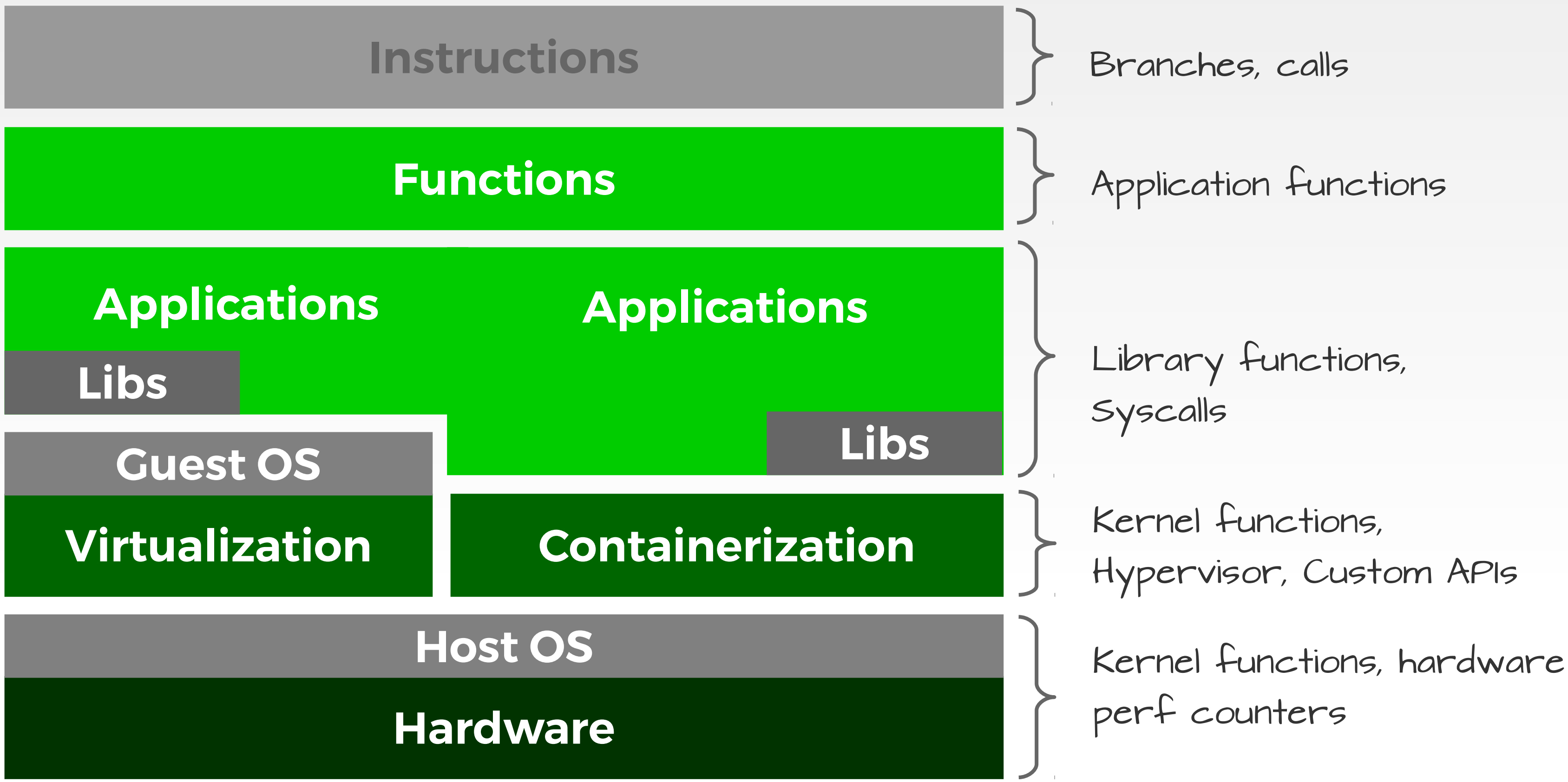
Observability Layers



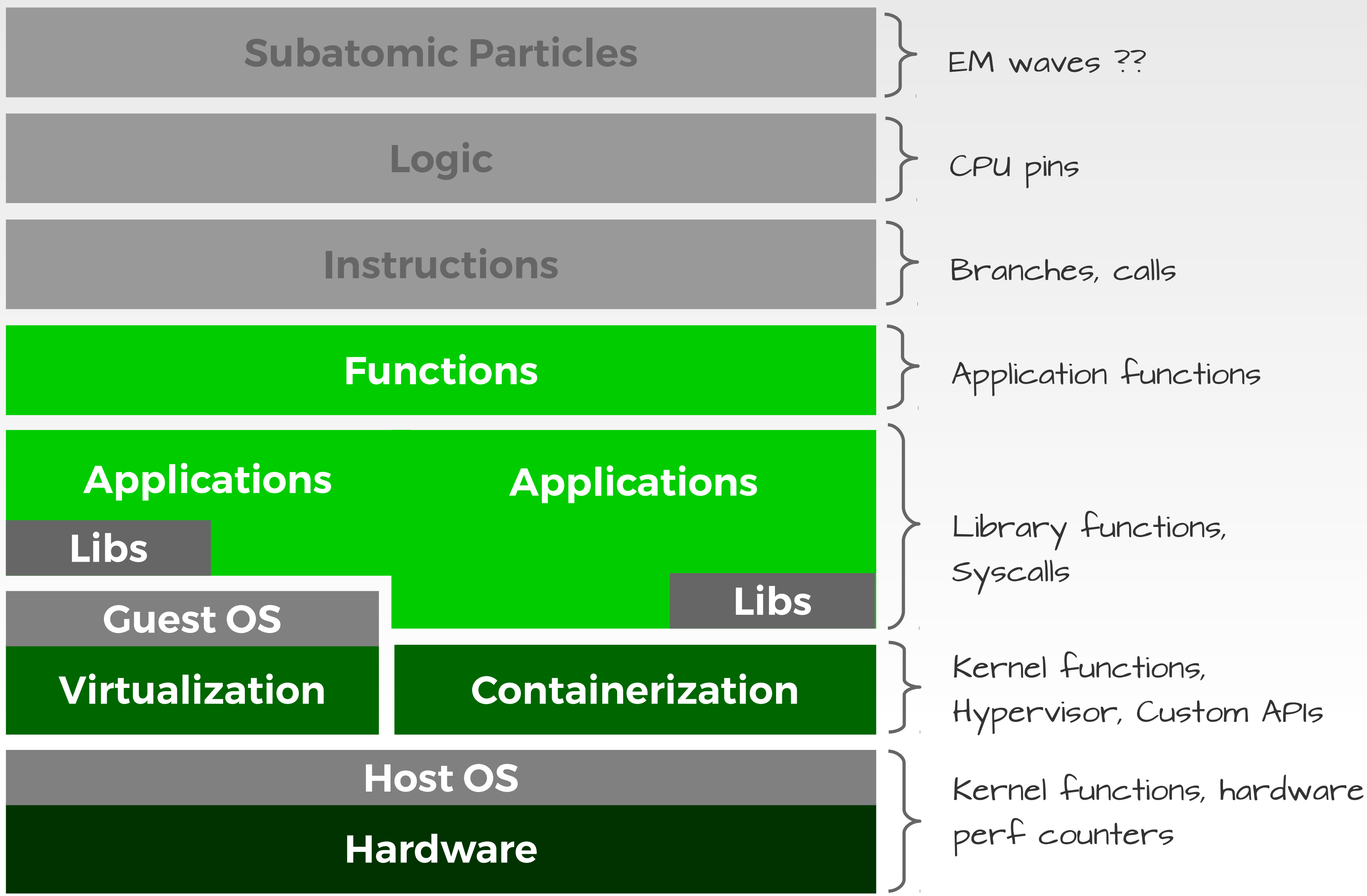
Observability Layers



Observability Layers



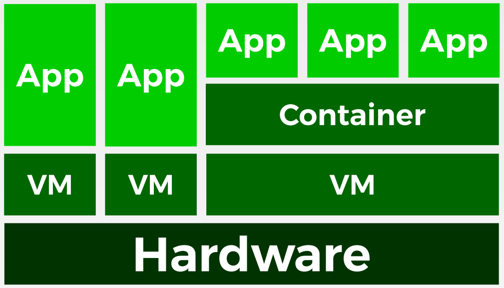
Observability Layers



Observing Modern Systems

Challanges

- Horizontal spread of services has increased

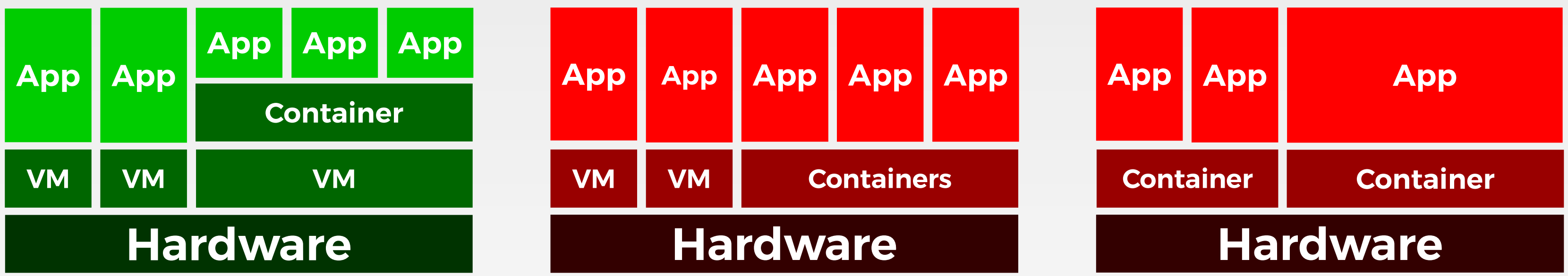


- Apps are distributed across machines and geographies
- Visibility across horizontal and vertical layers
- Preventive, Monitoring and Enforcing security for cloud-native applications is non-trivial now
- Developers need awareness of production setups in the modern world

Observing Modern Systems

Challanges

- Horizontal spread of services has increased



- Apps are distributed across machines and geographies
- Visibility across horizontal and vertical layers
- Preventive, Monitoring and Enforcing security for cloud-native applications is non-trivial now
- Developers need awareness of production setups in the modern world

 Suchakrapani Datt Sharma

Instrumenting Systems

Why

- Applications can now assist you in performance and security analysis
  - Understand program and data flow
  - Analyze timings and compare executions
- Powerful debugging using traces where debugging is prohibitively expensive

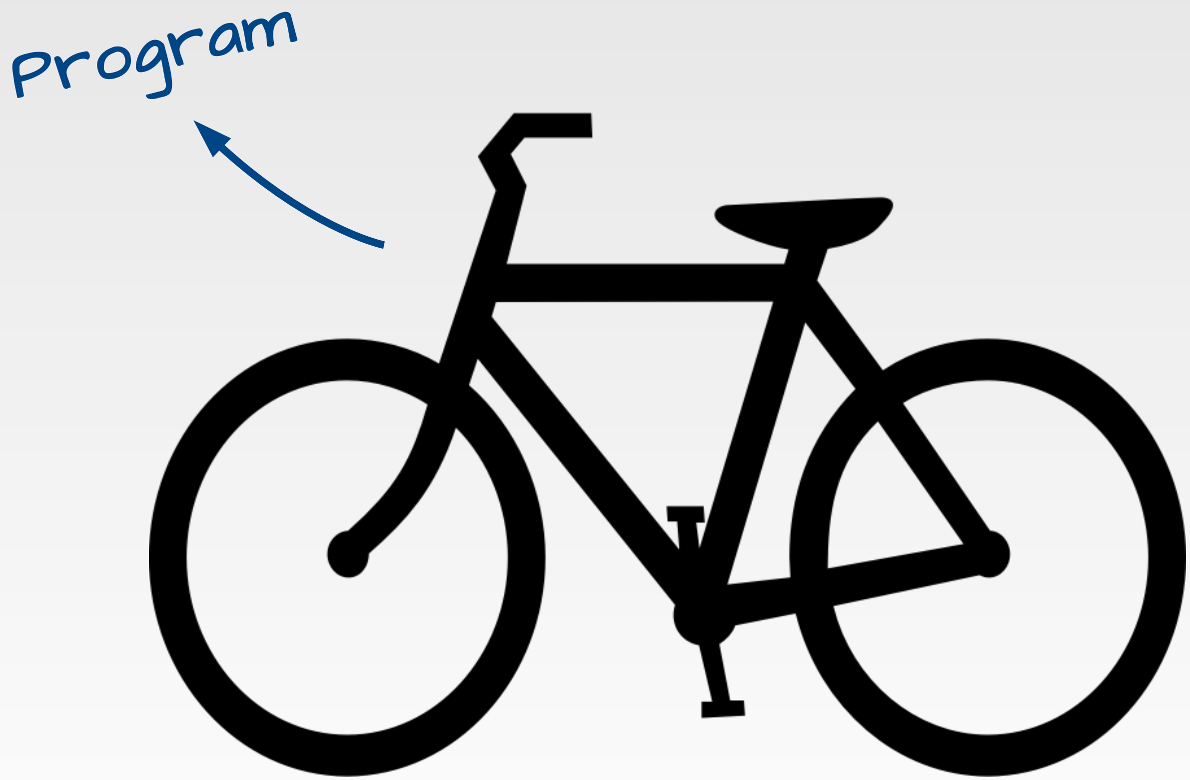
How

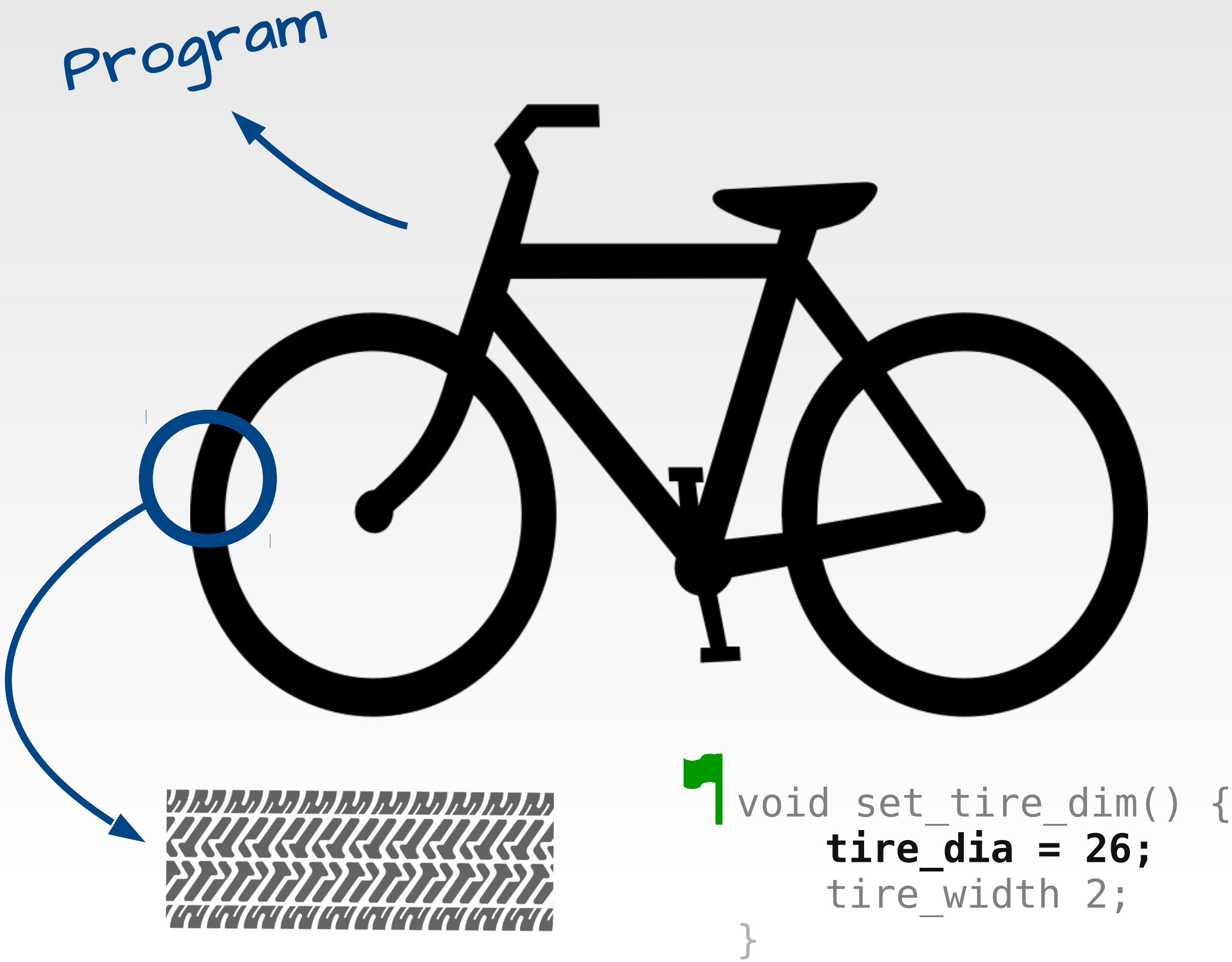
- Simple. Insert extra code at desired locations in any layer of the system (app, library, host OS)
- Add a `printf()`. Congrats ☺

Tracing

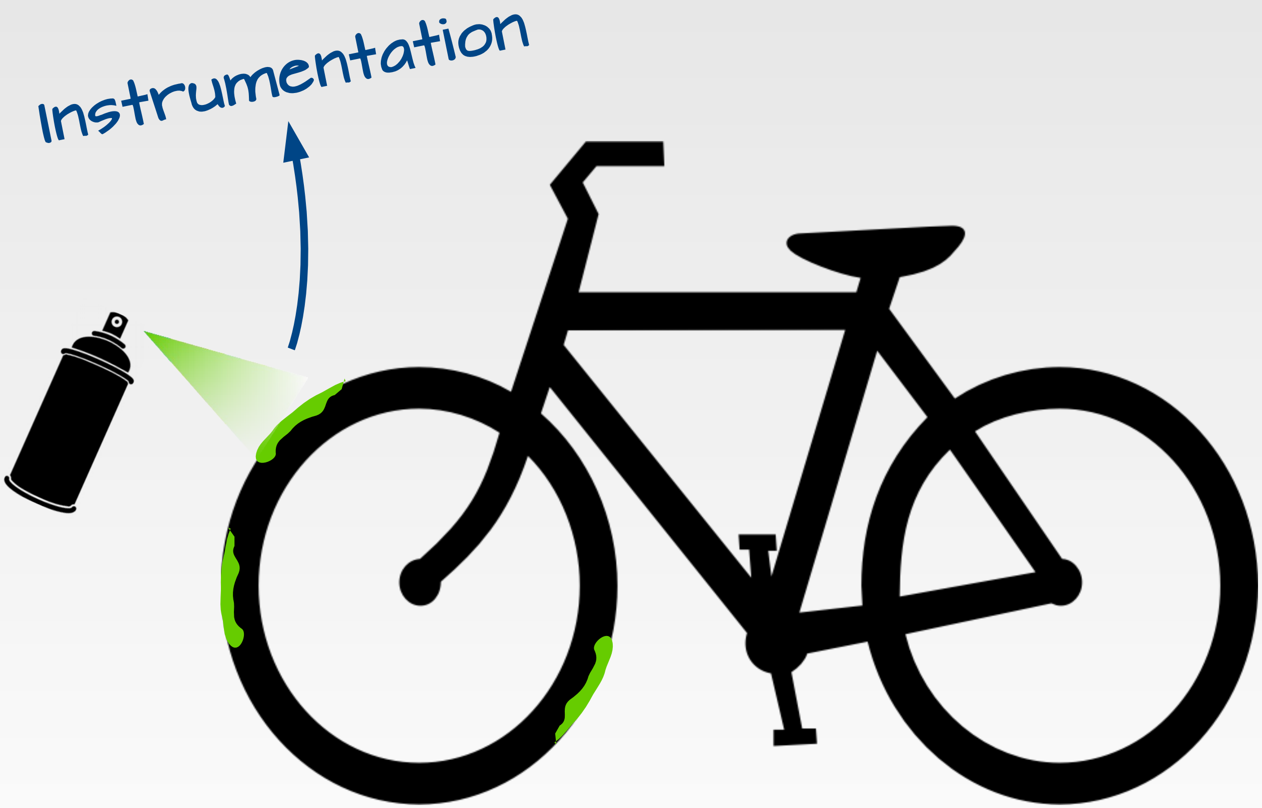
101

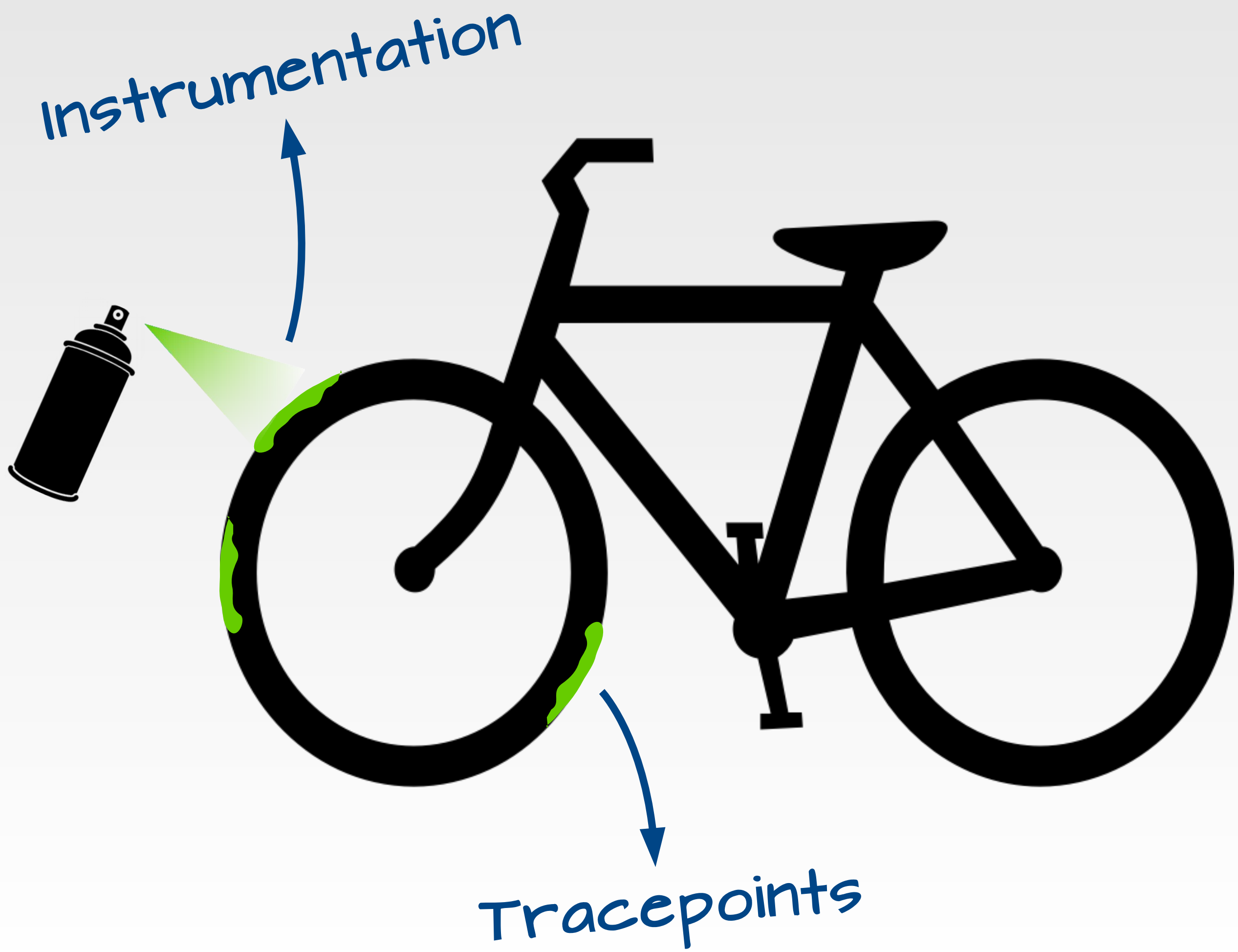
Suchakrapani Datt Sharma

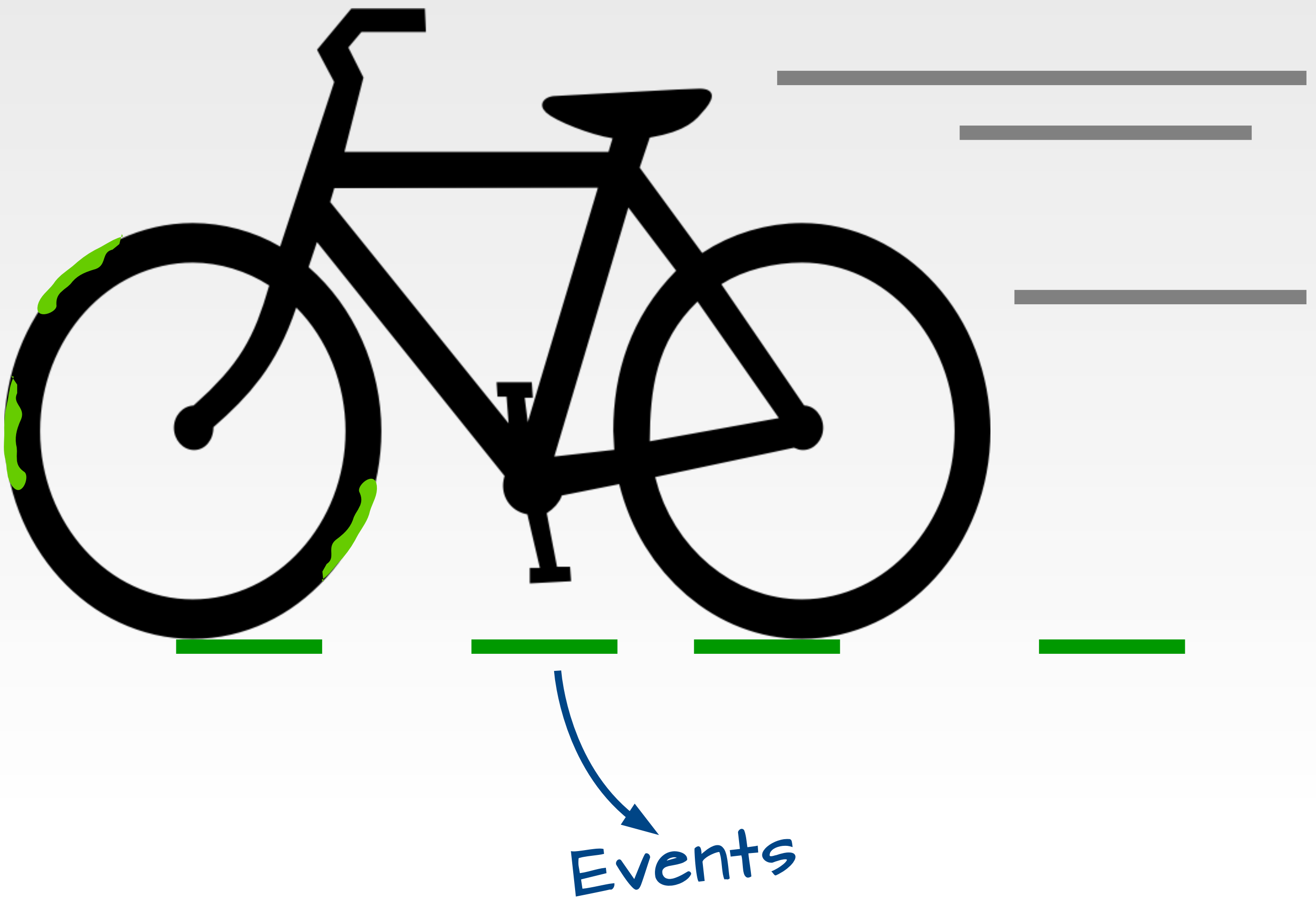

Program

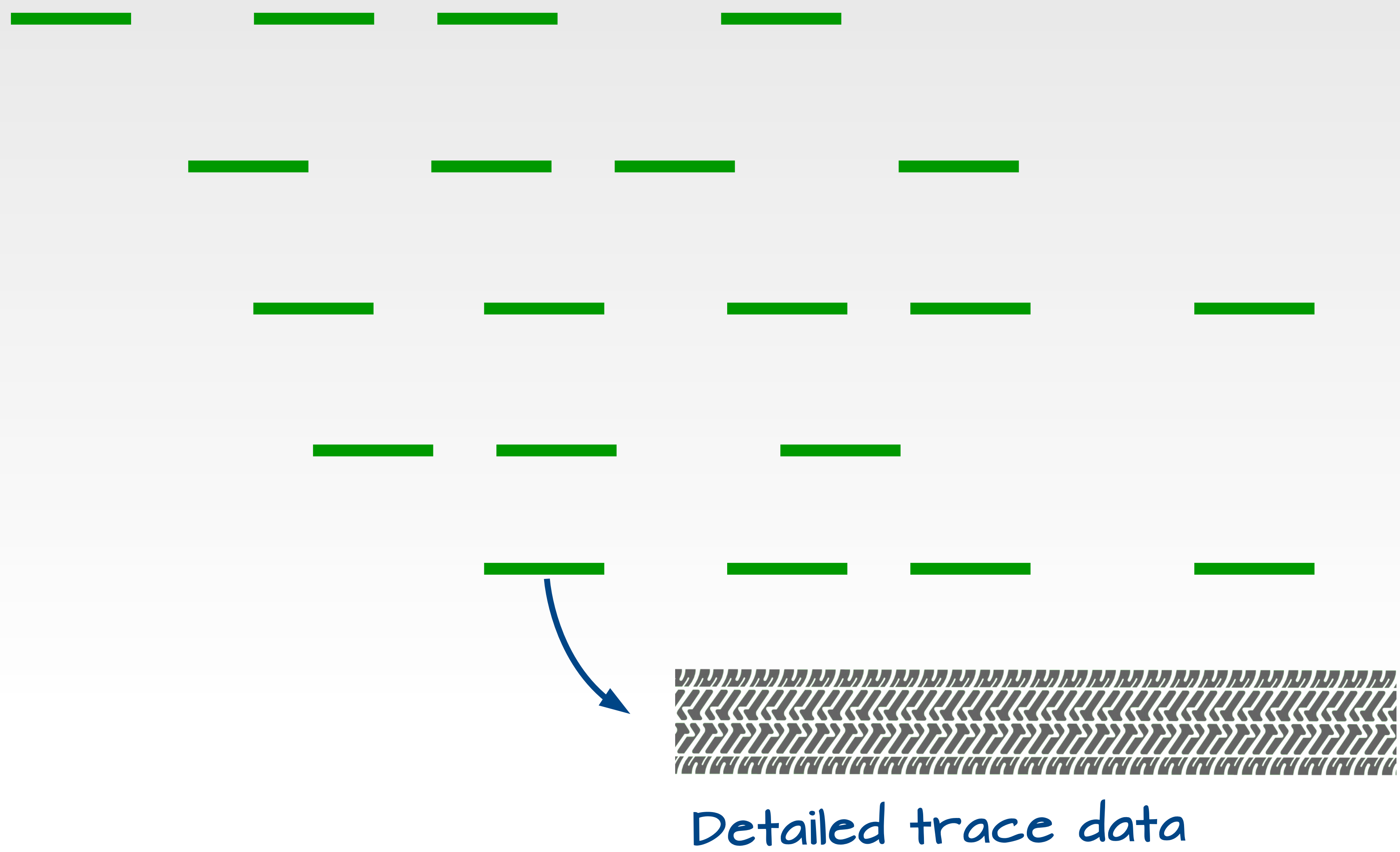


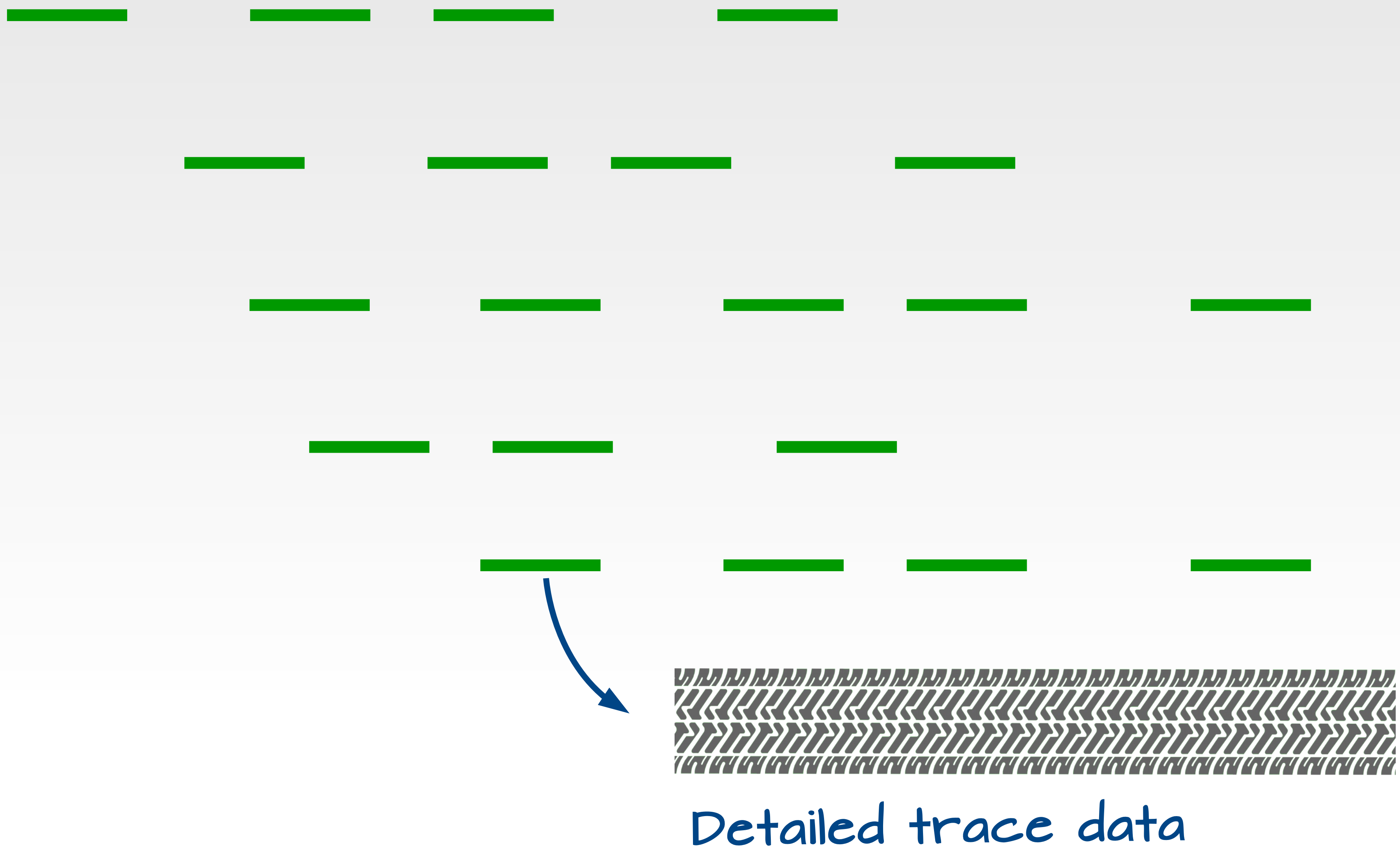
Suchakrapani Datt Sharma


Instrumentation

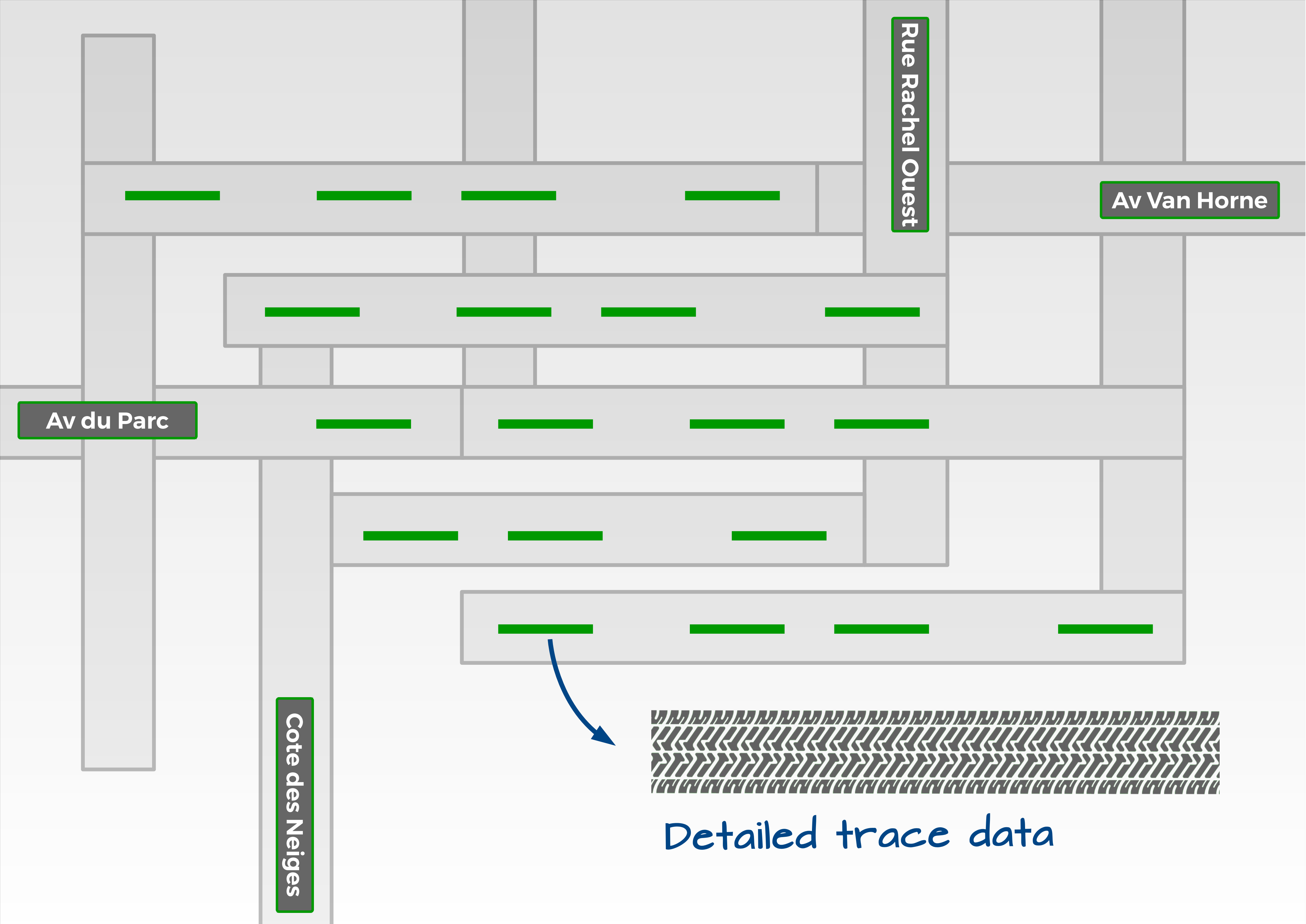

Instrumentation
Tracepoints


Events

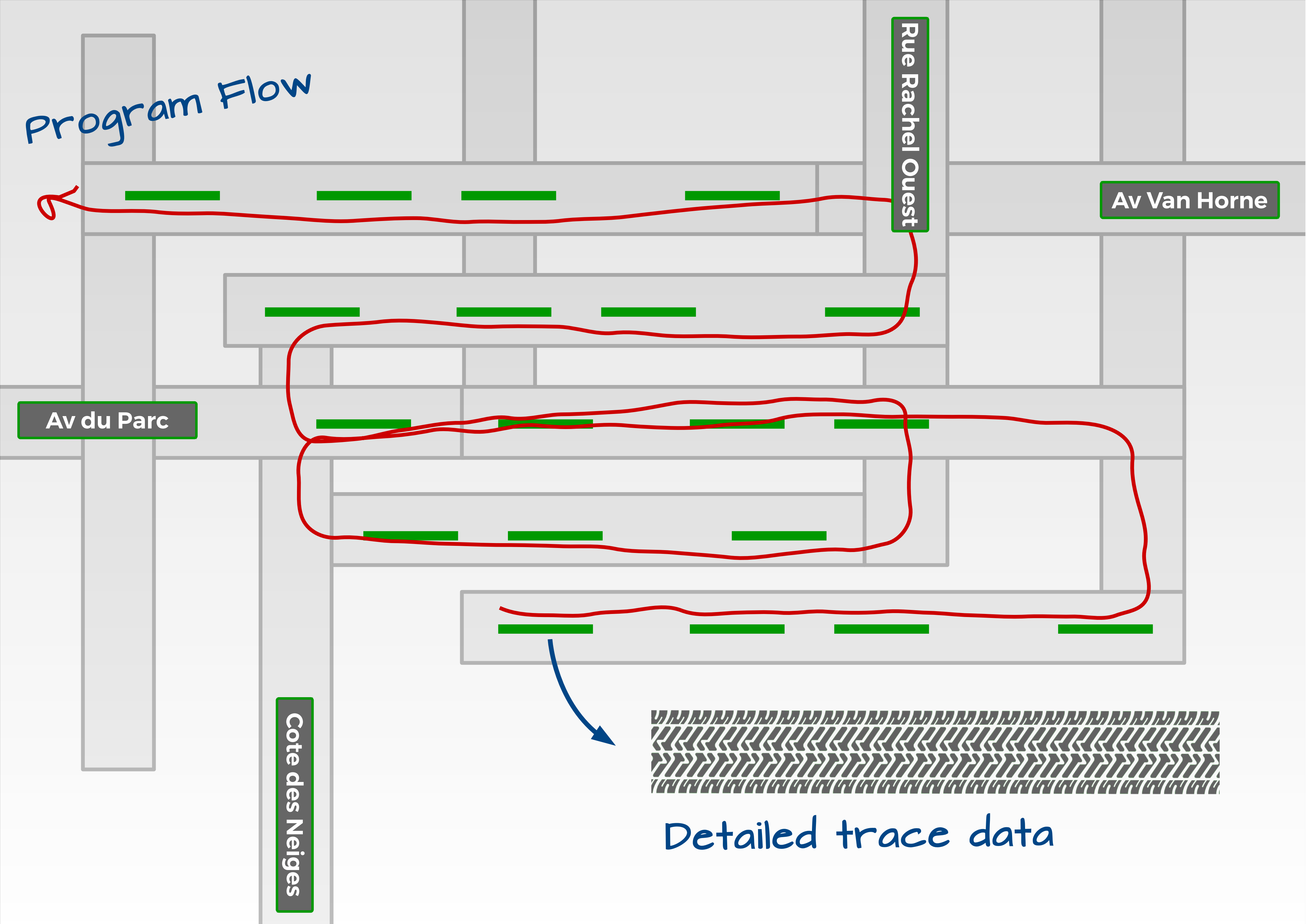

Detailed trace data



Detailed trace data



 Suchakrapani Datt Sharma



Suchakrapani Datt Sharma

Deep Dive

 Suchakrapani Datt Sharma

Instrumenting Systems

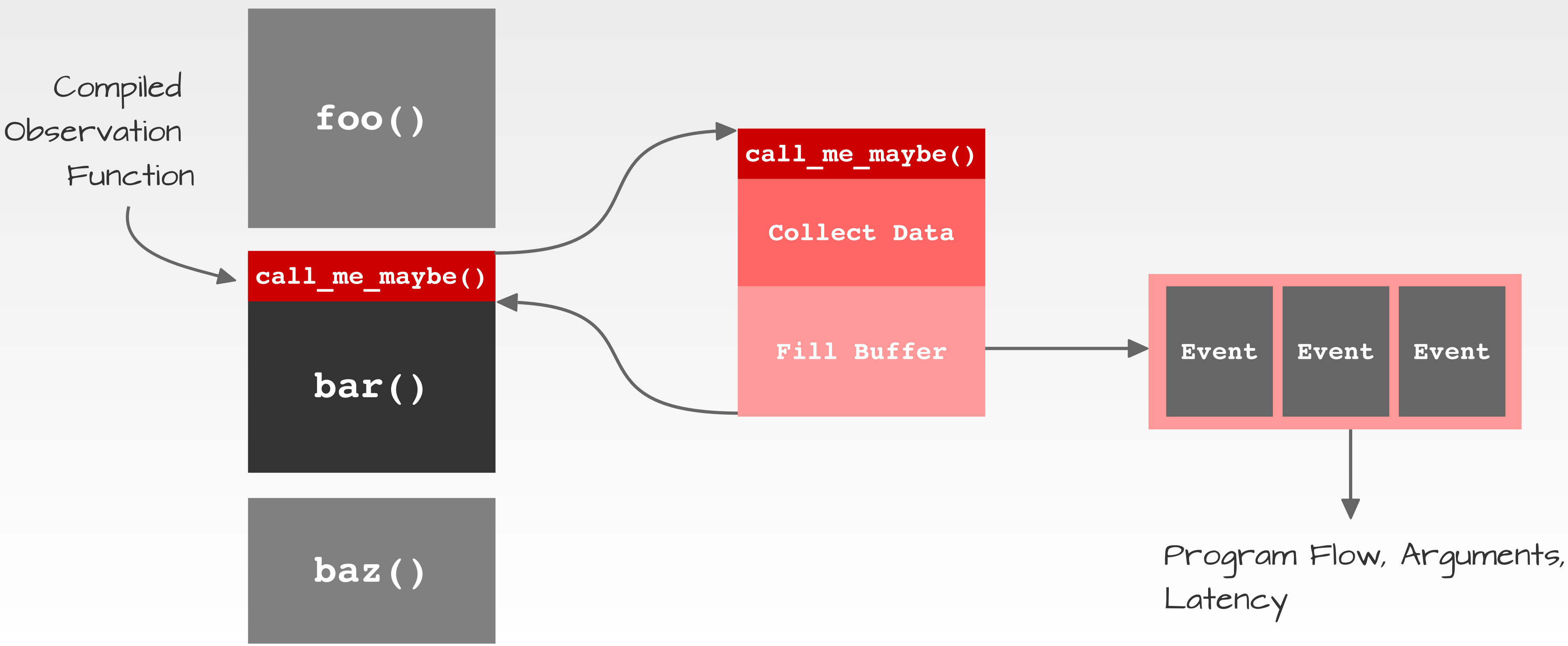
Static Instrumentation

- Development time, eg. insert code that takes a timestamp at function entry and saves it
- Compile time. Compiler inserts hooks that you can latch onto at runtime

Dynamic Instrumentation

- Patch a live application, insert your own observation code, let it run
  - Reliability
  - Security
- Translate code to another form, instrument it, run it *synthetically*

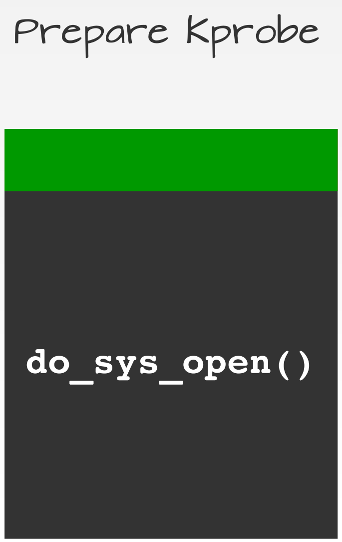
Instrumenting Systems



Case Study - I

Kernel Tracing with Kprobes

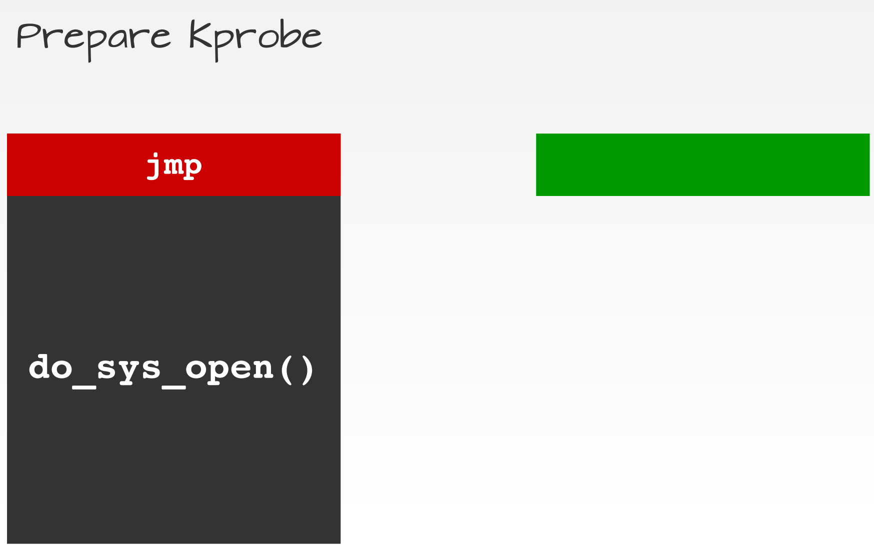
- Dynamic instrumentation based on trap or jump-pad based instrumentation



Case Study - I

Kernel Tracing with Kprobes

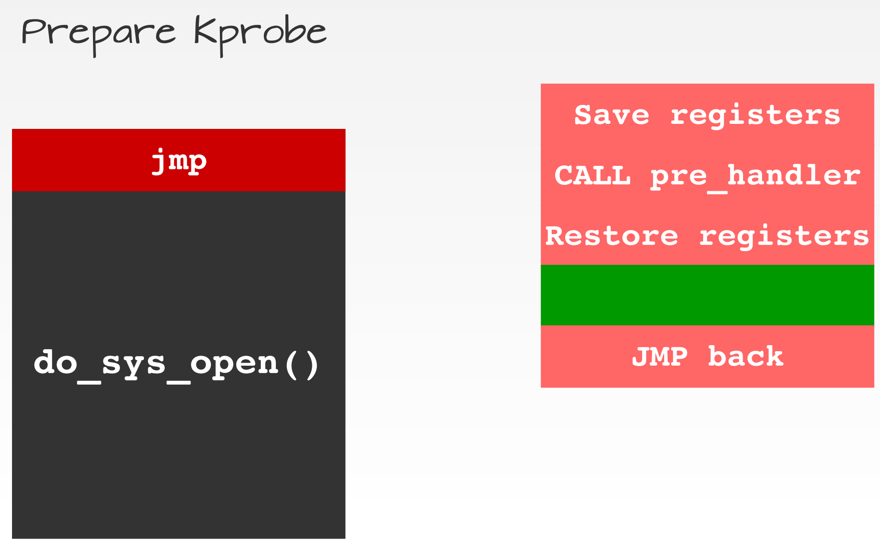
- Dynamic instrumentation based on trap or jump-pad based instrumentation



Case Study - I

Kernel Tracing with Kprobes

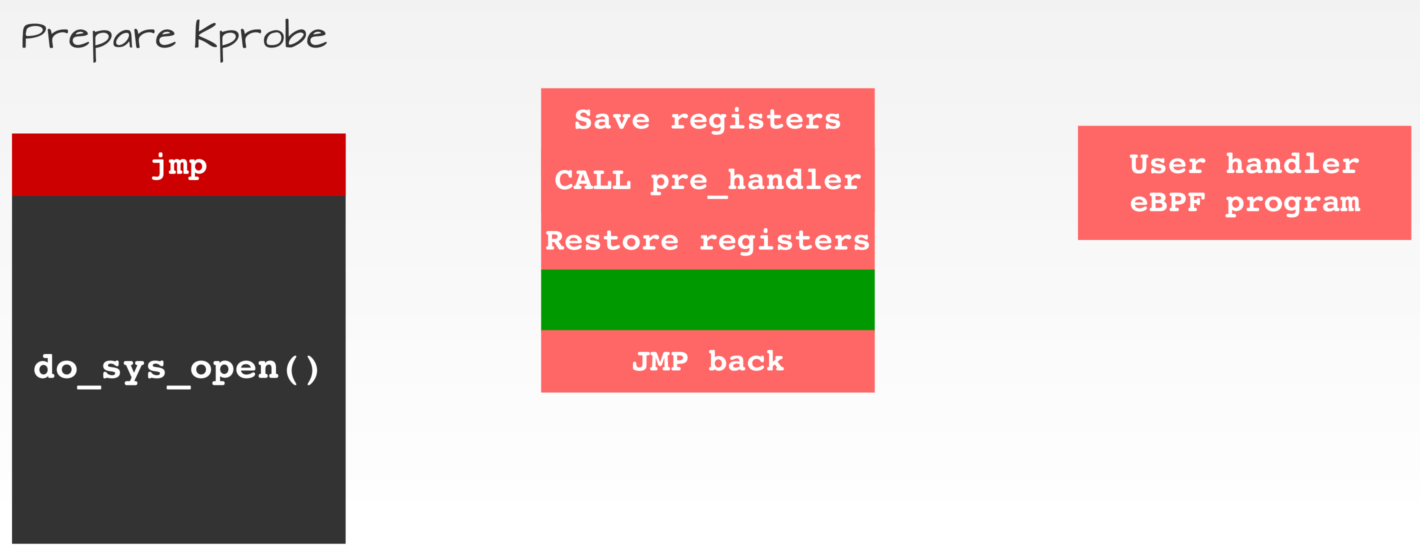
- Dynamic instrumentation based on trap or jump-pad based instrumentation



Case Study - I

Kernel Tracing with Kprobes

- Dynamic instrumentation based on trap or jump-pad based instrumentation



Case Study - I

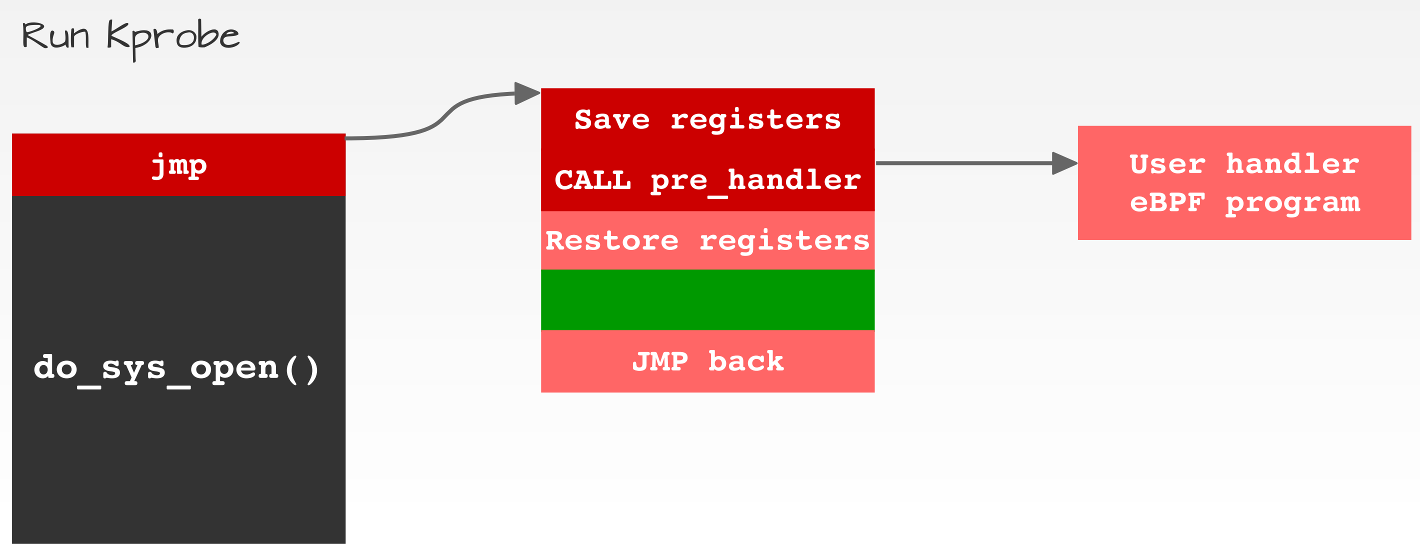
Kernel Tracing with Kprobes

- Dynamic instrumentation based on trap or jump-pad based instrumentation

Case Study - I

Kernel Tracing with Kprobes

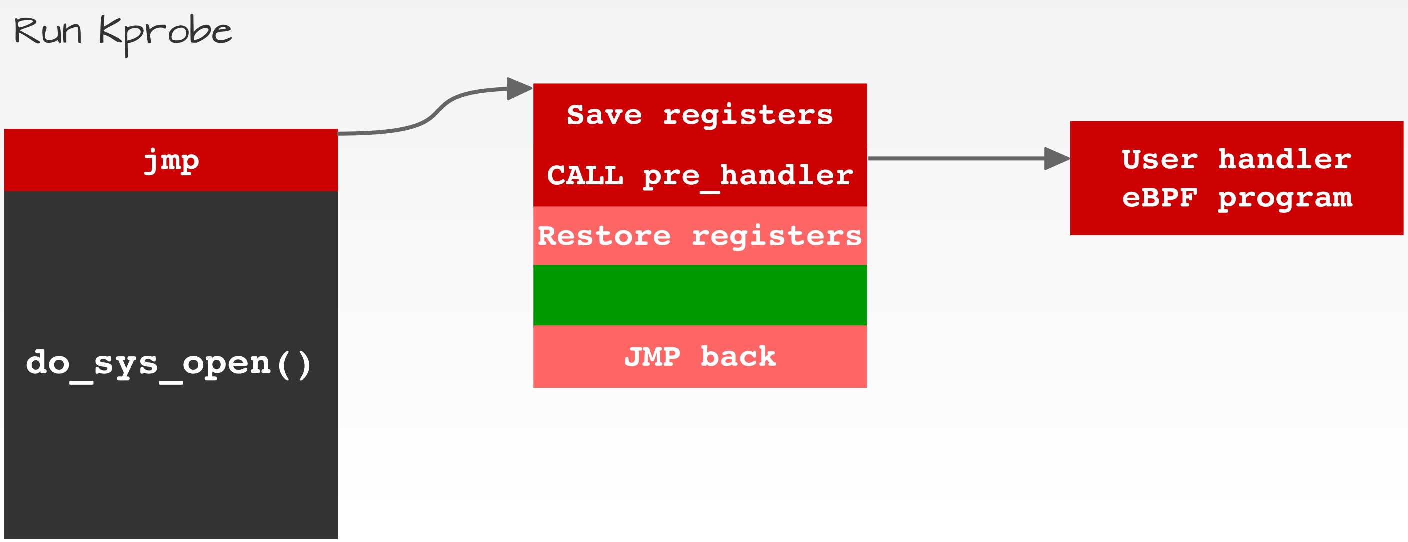
- Dynamic instrumentation based on trap or jump-pad based instrumentation



Case Study - I

Kernel Tracing with Kprobes

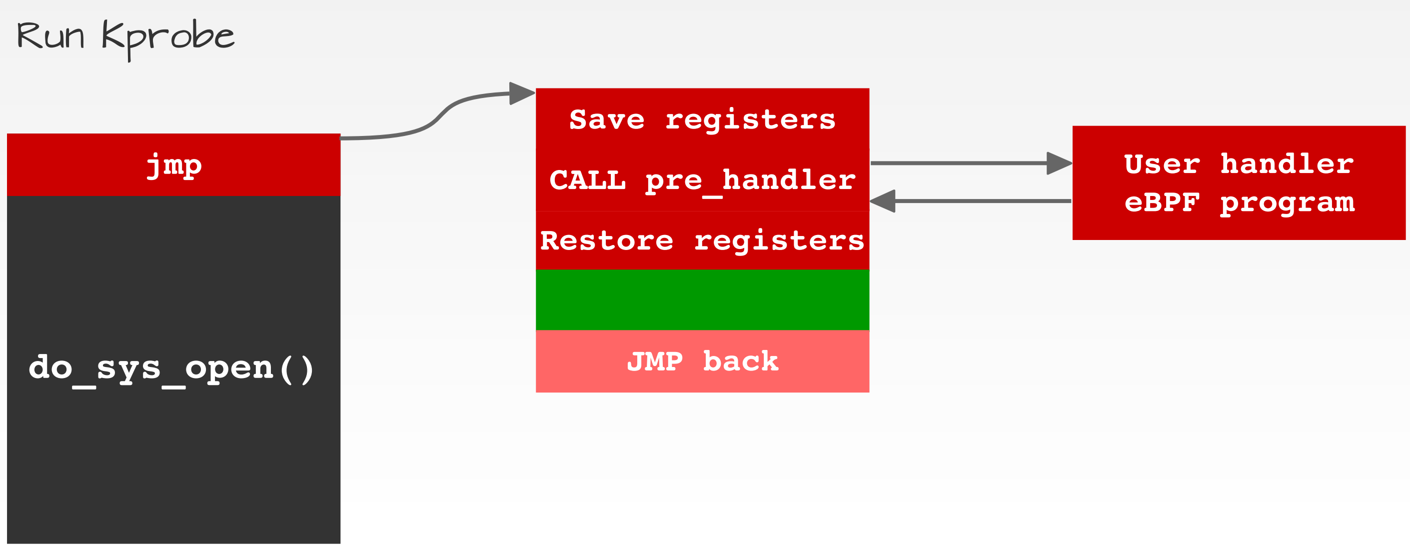
- Dynamic instrumentation based on trap or jump-pad based instrumentation



Case Study - I

Kernel Tracing with Kprobes

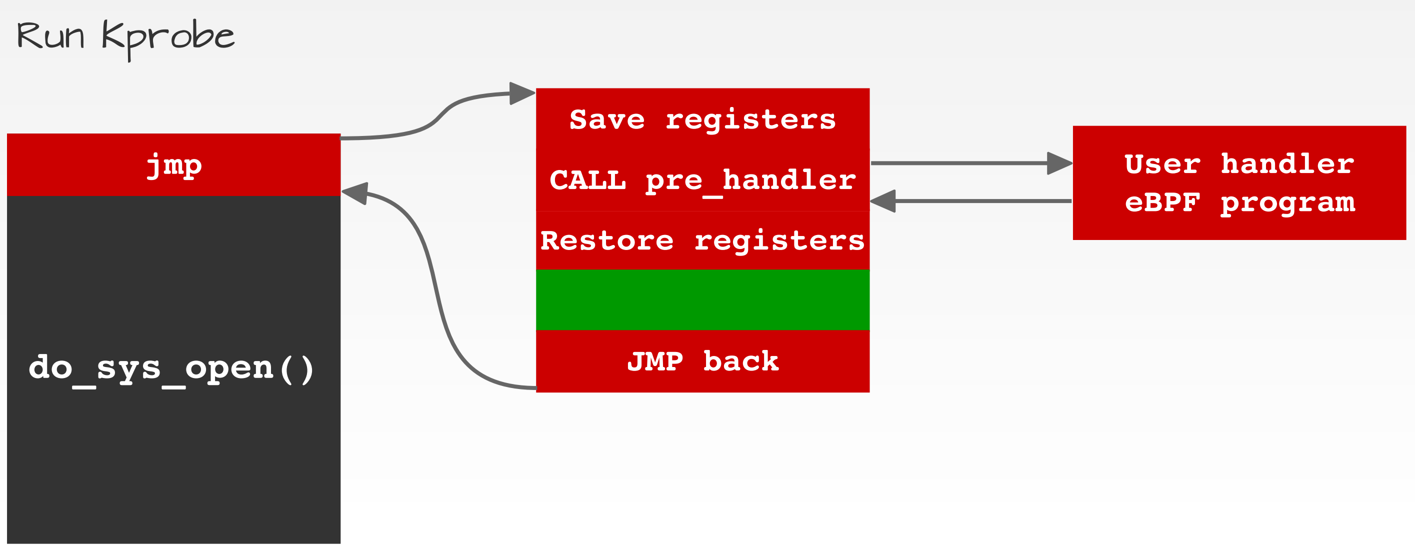
- Dynamic instrumentation based on trap or jump-pad based instrumentation



Case Study - I

Kernel Tracing with Kprobes

- Dynamic instrumentation based on trap or jump-pad based instrumentation



Other Techniques

Ftrace - Dynamic (Kernel)

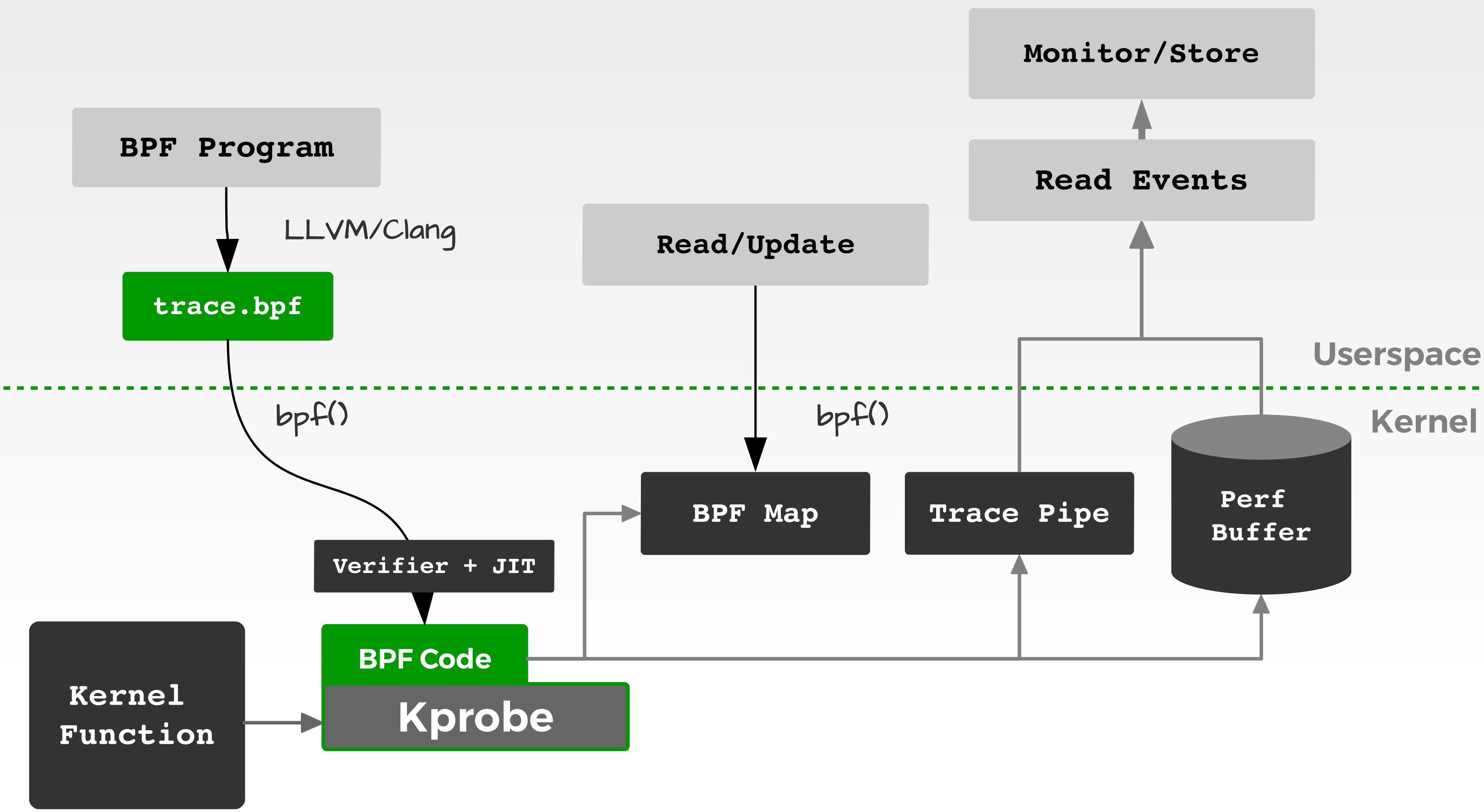
- Build kernel functions with `mcount` (`-pg` in GCC)
- Patch it to NOPs at boot. Add `jmp` to handler for activated functions
- Add hooks, save function arguments, timestamp
- Generate function graph

Static Tracepoints (Kernel)

- trace_* in most kernel functions
- Uses TRACE_EVENT based static tracepoints
- Well defined kernel trace events, can be attached to perf, Ftrace (Static)

Case Study - I

eBPF + Kprobe



Case Study

eBPF + Kprobe

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

Example

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

Case Study

eBPF + Uprobes Example

Program Excerpt

```
bpf_text = """
#include <uapi/linux/ptrace.h>
#include <uapi/linux/limits.h>

int get_fname(struct pt_regs *ctx) {
    if (!ctx->si)
      return 0;
    char str[NAME_MAX] = {};
    bpf_probe_read(&str, sizeof(str), (void *)ctx->si);
    bpf_trace_printk("%s\\n", &str);
    return 0;
};
"""
b = BPF(text=bpf_text)
b.attach_uprobe(name="/usr/bin/vim", sym="readfile", fn_name="get_fname")
```

Get 2nd argument → ctx->si

Process → "/usr/bin/vim"

Symbol → "readfile"

Output

```
# ./vim-test.py
TASK    PID     FILENAME
vim     23707   /tmp/wololo
```

Case Study - II

Linux Security Modules (LSM)

- Static Instrumentation in the kernel code
- Hooks to attach LSM implementations for defining and inserting MAC policies
  - **SELinux**
  - AppArmor, LandLock LSM (eBPF)

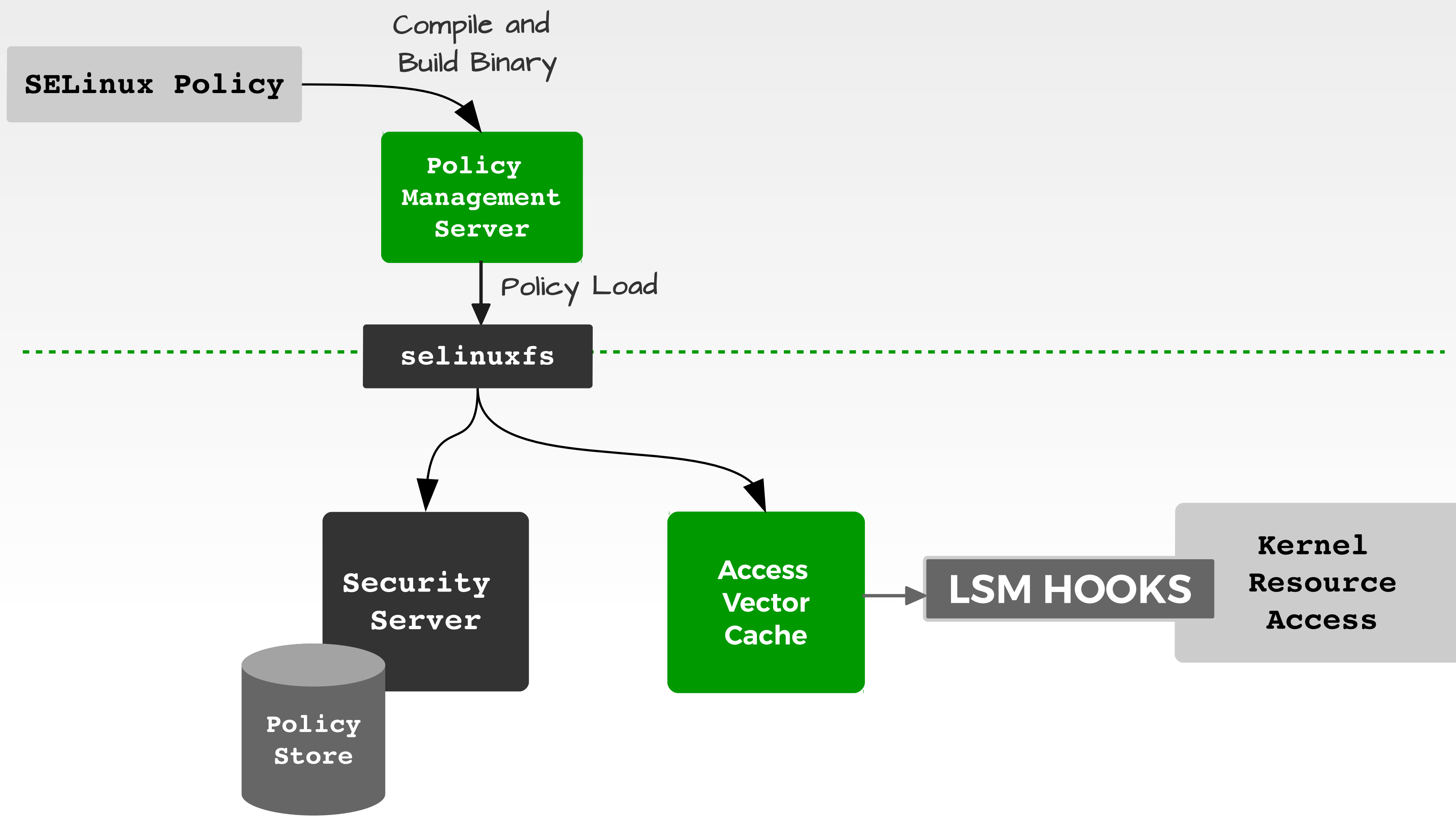
Syscall from userspace

```
open()
  do_sys_open()
    do_filp_open()
      path_openat()
      ..
        vfs_open()
          do_dentry_open()
            security_file_open()
```

LSM call

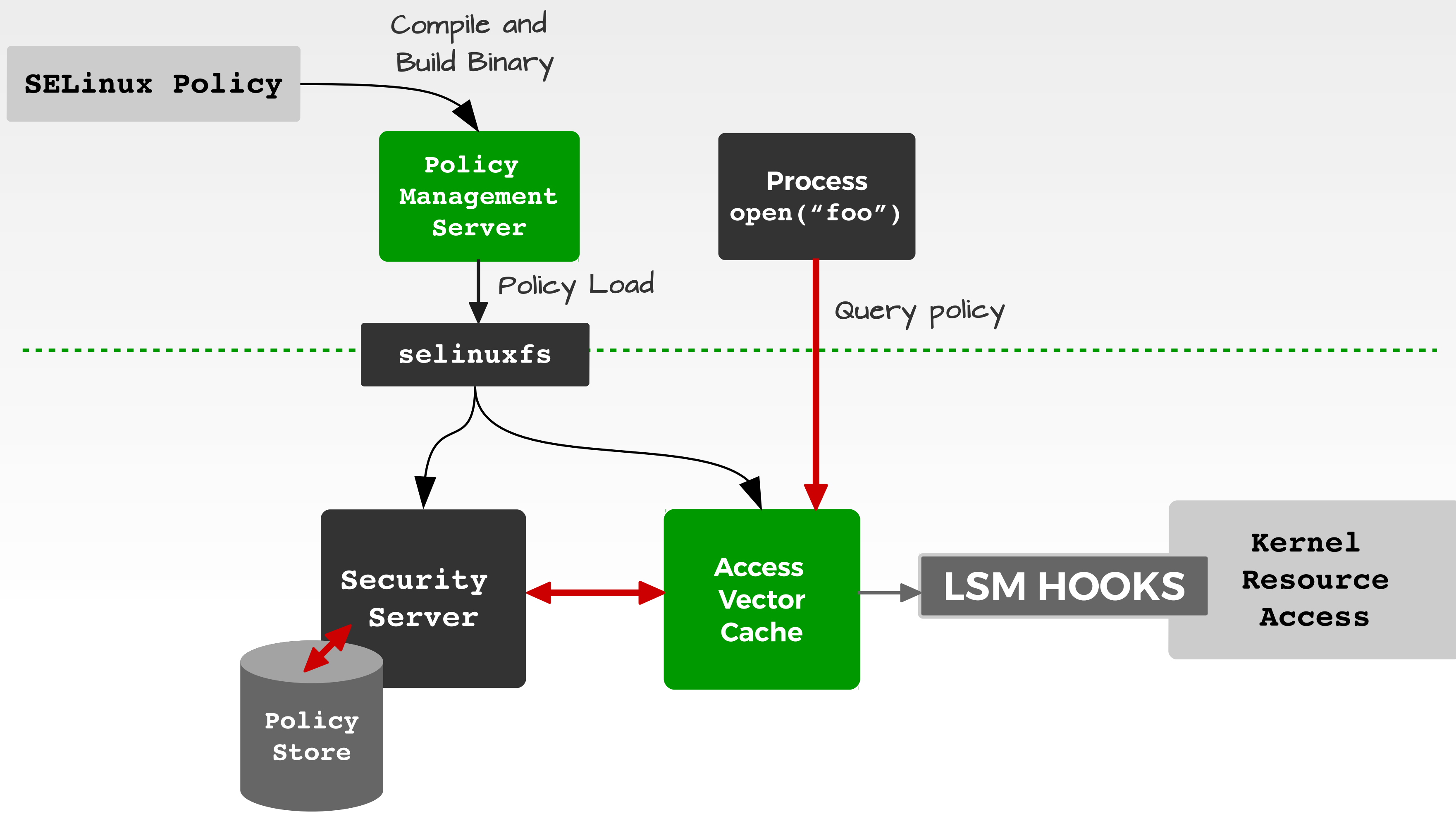
Case Study - II

SELinux



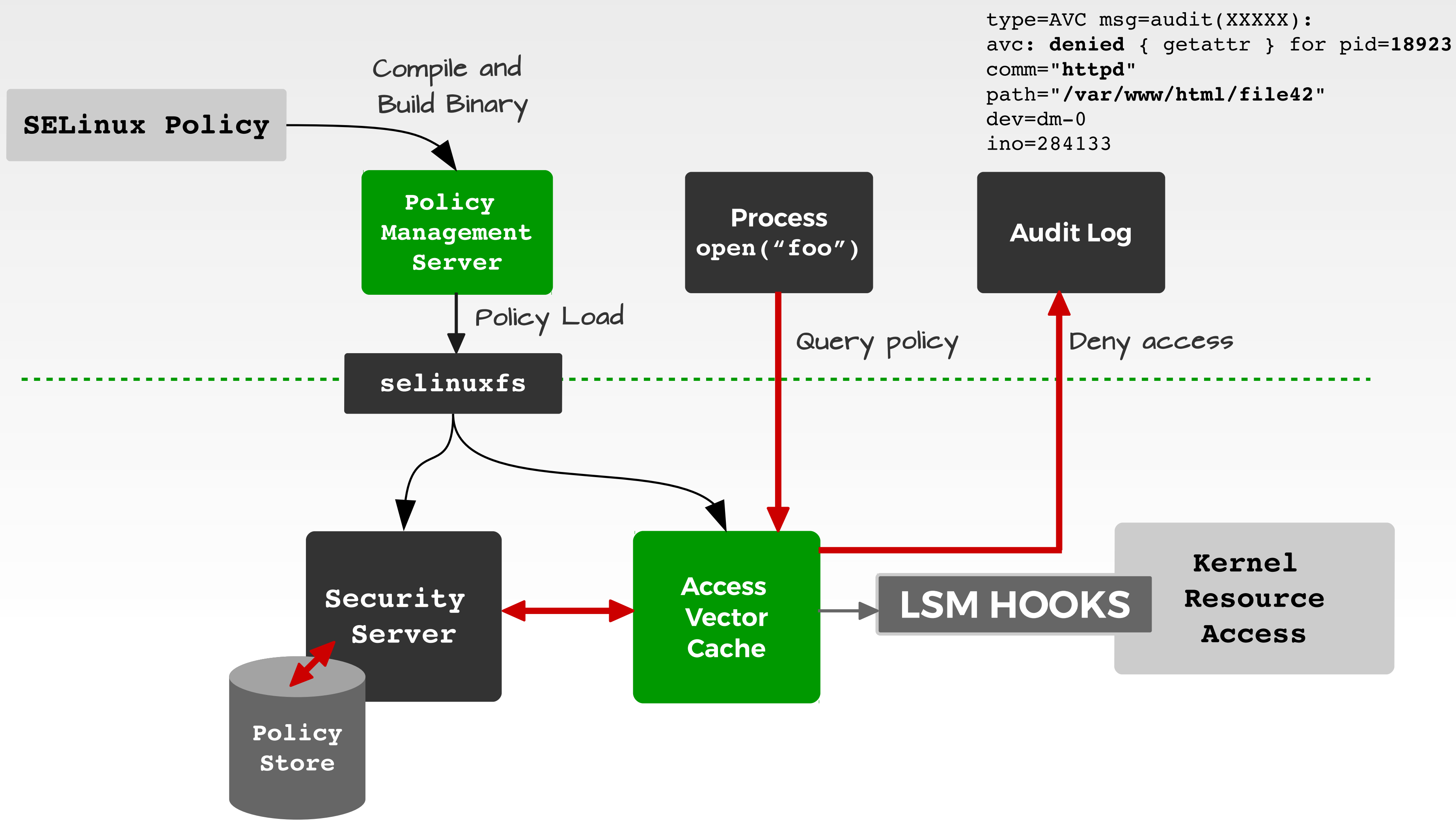
Case Study - II

SELinux



Case Study - II

SELinux



Securing **Stuff**

Securing Strategy

Preventive (Isolation)

- Virtualization **(Hypervisor/VMs)**
  – Inherent isolation, by virtue of hardware/software design
  – Robust, smaller attack surface
- Linux Namespaces/Cgroups **(Containers)**
  – Isolation by host kernel/userspace support
  – Isolate resources and groups of processes
  – Used to *define* containers
- Linux Capabilities: Not just root/non-root now
- SECCOMP* **(Application)**
  – Can be used to sandbox process and allow/deny syscalls

Securing Strategy

Passive (Monitoring)

- System level logs and audit messages (Auditd)
  - Get summary of AVC denials/syscalls to keep track of interesting events
- Hook to system events such as capability, syscalls custom userspace events
- Monitor network events across layers

Securing Strategy

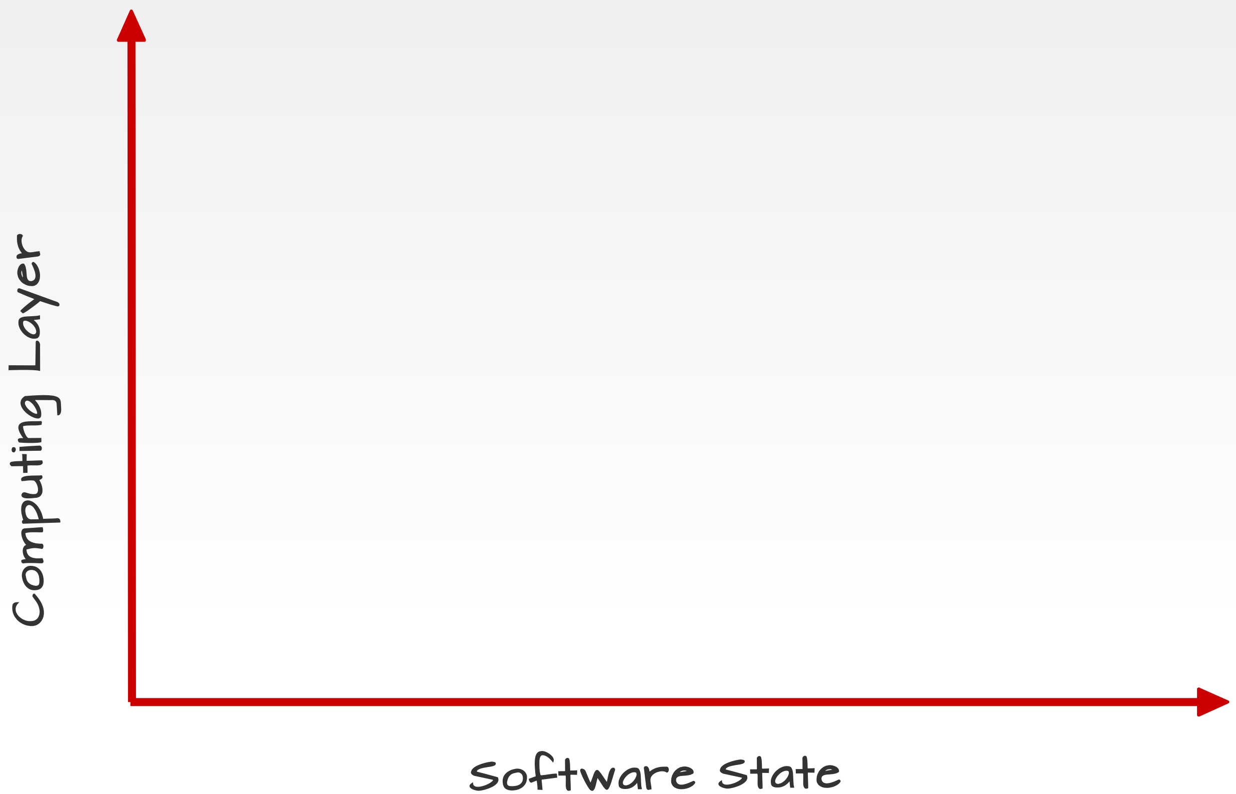
Active (Protection)

- LSM Modules
  - Protect infrastructure and implement policies
  - Policies can now be programmable (eBPF)
  - Support with modern containers (policies for Docker, rkt, K8s)
- SECCOMP
  - Policies for Docker, rkt, runc
- Custom application/library instrumentation

Securing Strategy

Insertion Spectrum

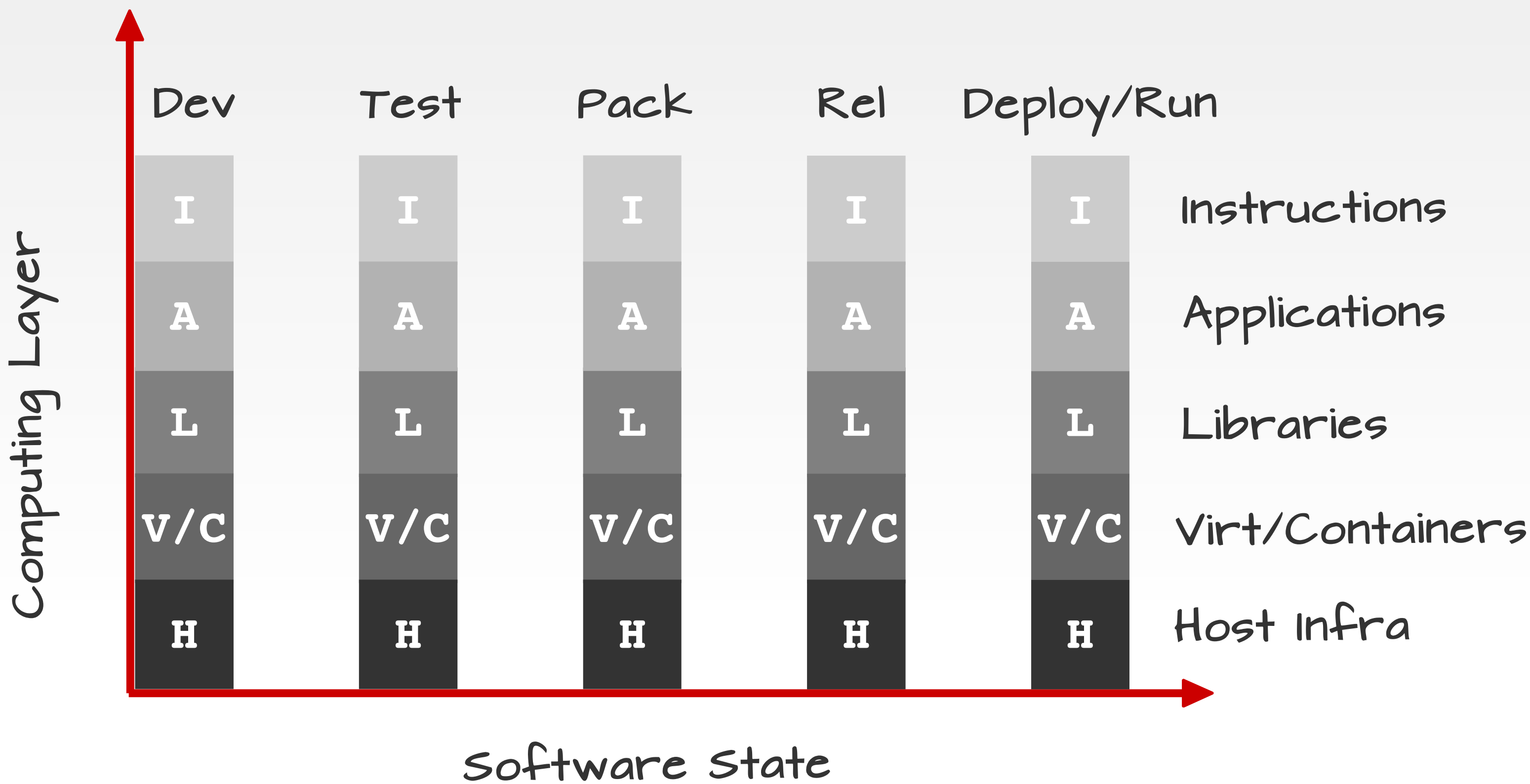
- Two variables for defining and inserting security



Securing Strategy

Insertion Spectrum

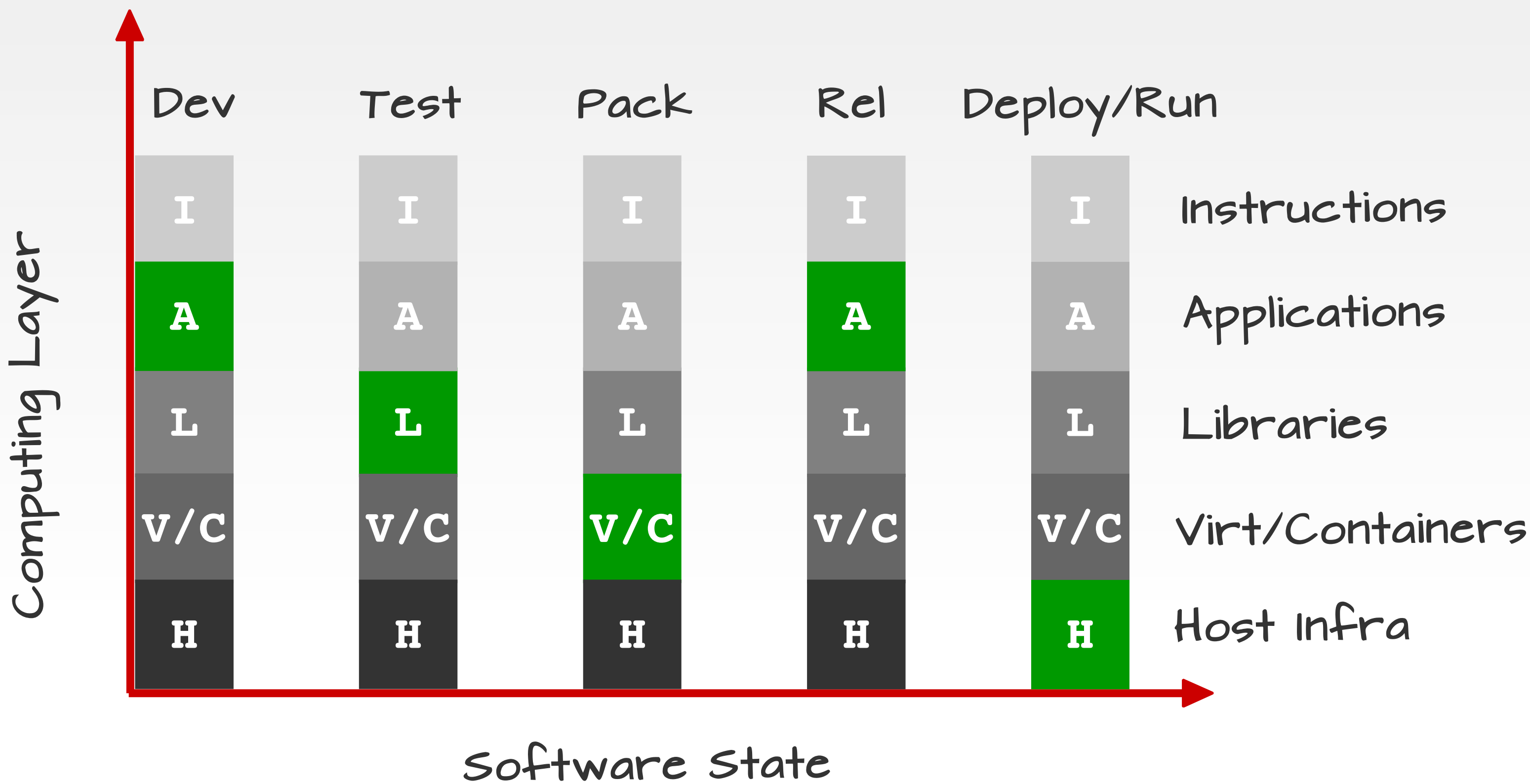
- Two variables for defining and inserting security



Securing Strategy

Insertion Spectrum

- Two variables for defining and inserting security



References

Links

- *Kprobes Kernel Docs*
  - https://www.kernel.org/doc/Documentation/kprobes.txt

- *Secrets of Ftrace Function Tracer (Steven Rostedt)*
  - https://lwn.net/Articles/370423/

- *Linux Performance/Tracing (Brendan Gregg & Julia Evans & Honeycomb)*
  - http://www.brendangregg.com/linuxperf.html
  - https://jvns.ca/blog/2017/07/05/linux-tracing-systems/
  - https://honeycomb.io/blog/categories/instrumentation/

- *Linux Security/Containers (Jessie Frazelle & Jérôme Petazzoni)*
  - *https://blog.jessfraz.com/post/a-rant-on-usable-security/*
  - *https://www.slideshare.net/jpetazzo/anatomy-of-a-container-namespaces-cgroups-some-filesystem-magic-linuxcon)*

- *eBPF/Landlock LSM (IOVisor devs, Mickaël Salaün et al.)*
  - *https://landlock-lsm.github.io/linux-doc/landlock-v5/security/landlock/index.html*
  - *http://www.brendangregg.com/ebpf.html*
  - *http://blogs.microsoft.co.il/sasha/2016/12/23/usdtbpf-tracing-tools-java-python-ruby-node-mysql-postgresql/*

Fin

suchakra@shiftleft.io
@tuxology