Sasha Goldshtein
CTO, Sela Group

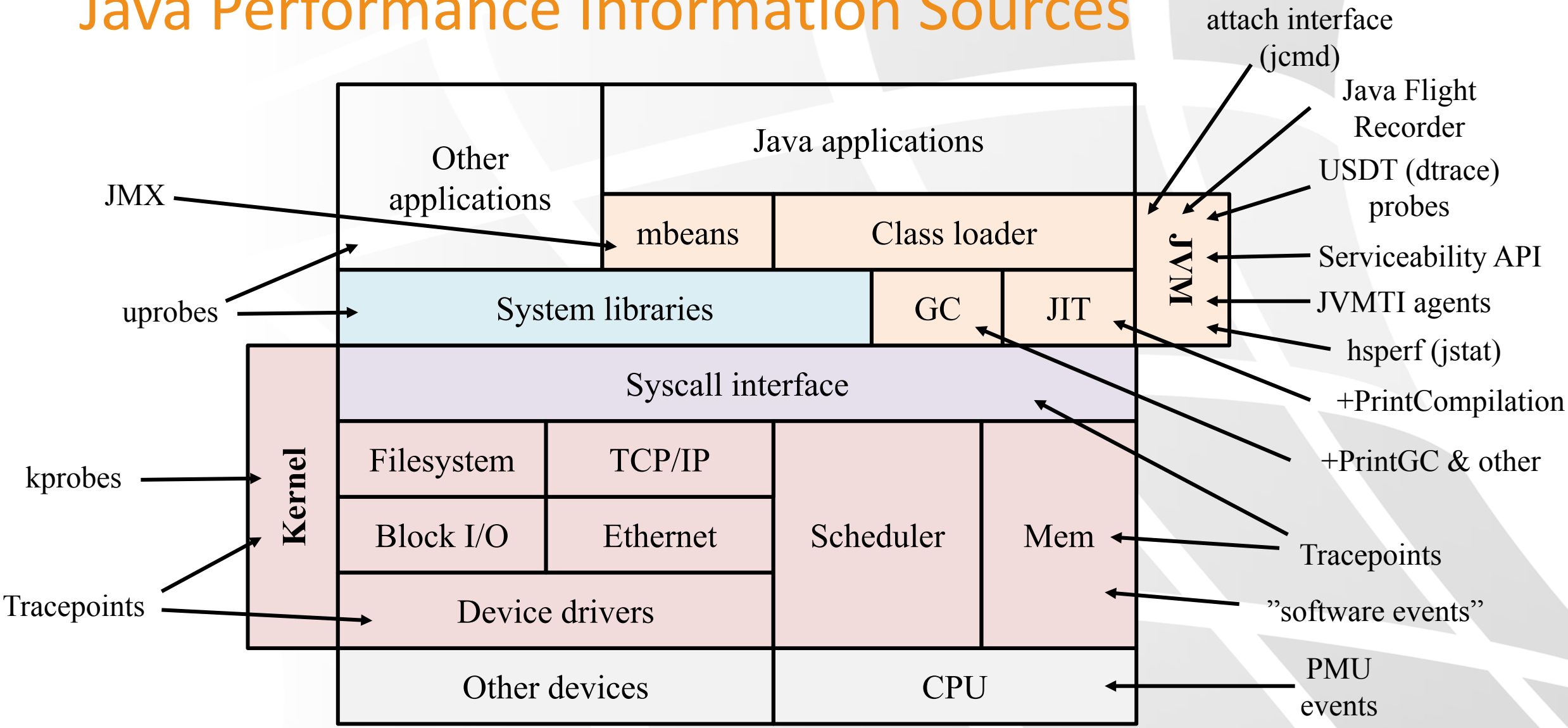
@goldshtn
github.com/goldshtn

Linux Container Performance Tools for JVM Applications

Agenda

- Mission:
  Apply modern production-ready tools for performance monitoring and profiling of Java applications in Linux containers
- Objectives:
  - ❑ Identifying overloaded resources in containers
  - ❑ Understanding which tools work and which don't in container scenarios
  - ❑ Profiling CPU bottlenecks
  - ❑ Visualizing and exploring stack traces using flame graphs
  - ❑ Analyzing off-CPU time and CPU throttling

Java Performance Information Sources



Linux Containers Under The Hood

Control Groups

- Restrict usage (place quotas)
- cpu,cpuacct: used to cap CPU usage and apply CPU shares

```
docker run --cpus --cpu-shares
```

- memory: used to cap user and kernel memory usage

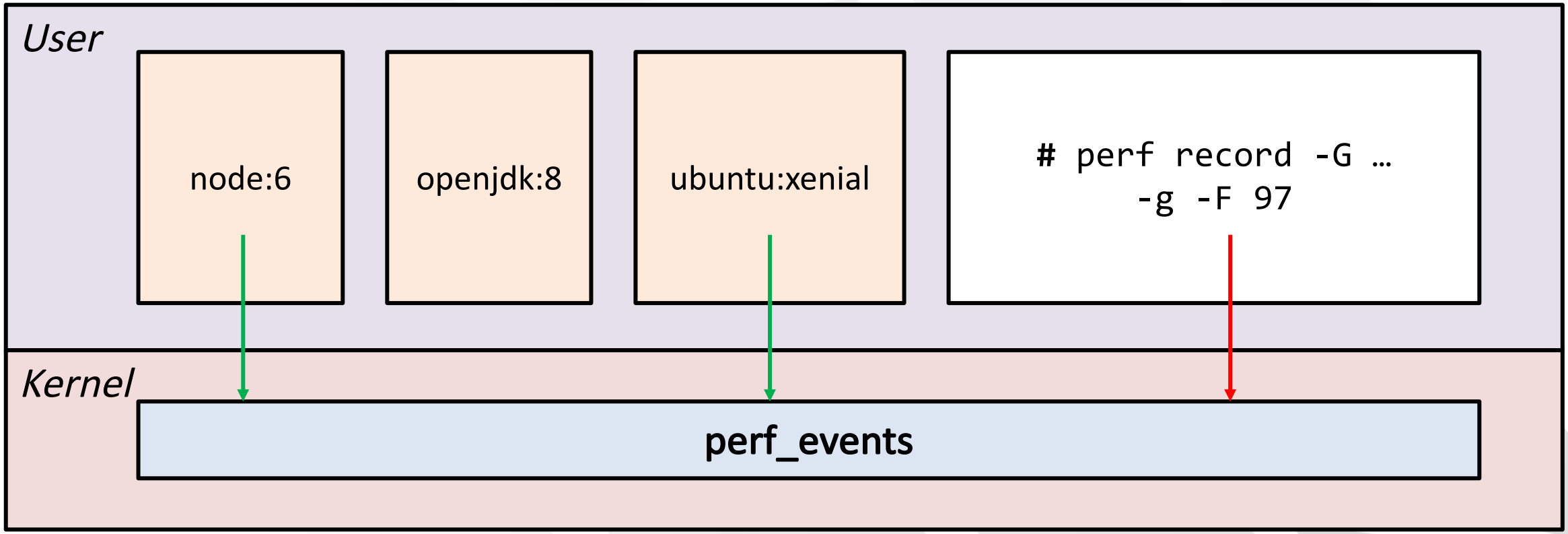
```
docker run --memory
           --kernel-memory
```

- blkio: used to cap IOPS and throughput per block device, and to assign

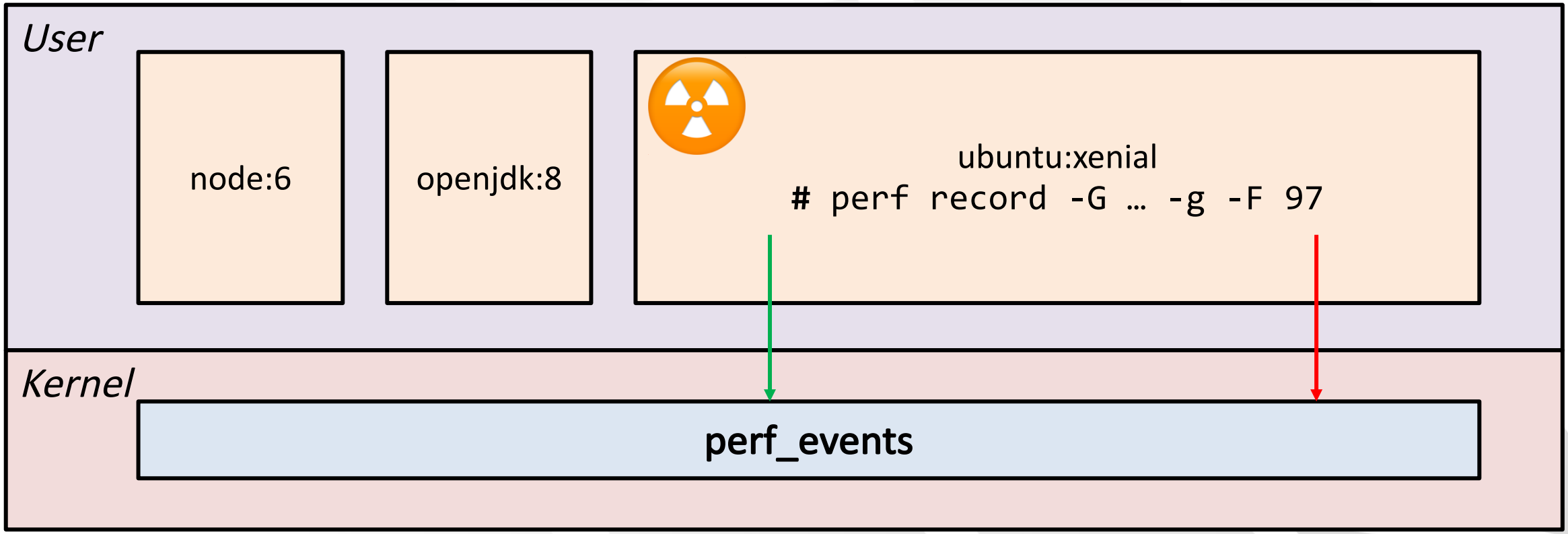
Namespaces

- Restrict visibility
- PID: container gets its own PIDs
- mnt: container gets its own mount points
- net: container gets its own network interfaces
- user: container gets its own user and group ids
- ...etc.

Tool Deployment: On The Host



Tool Deployment: In Container



Problems

- On the host:
  - Need privileged access to the host (most container orchestrators try to abstract this away from you)
  - Tools need to understand container pid namespace, mount namespace, and other details (discussed later)
- In the container:
  - Need to run the container with extra privileges (e.g. for perf: enable `perf_event_open` syscall, perf_event_paranoid sysctl)
  - Bloats container images with performance tools that are not always used, and increases attack surface
  - Performance tools may be throttled by quotas placed on the container

Examples of JVM Tools That Fail

Problems

- JVM attach interface:
  - Mount namespace – attach mechanism relies on file and UNIX domain socket
  - User namespace – attach mechanism requires user ids to match
- JVM performance data:
  - Mount namespace – container's /tmp/hsperfdata_UID/PID not visible to host

Solutions

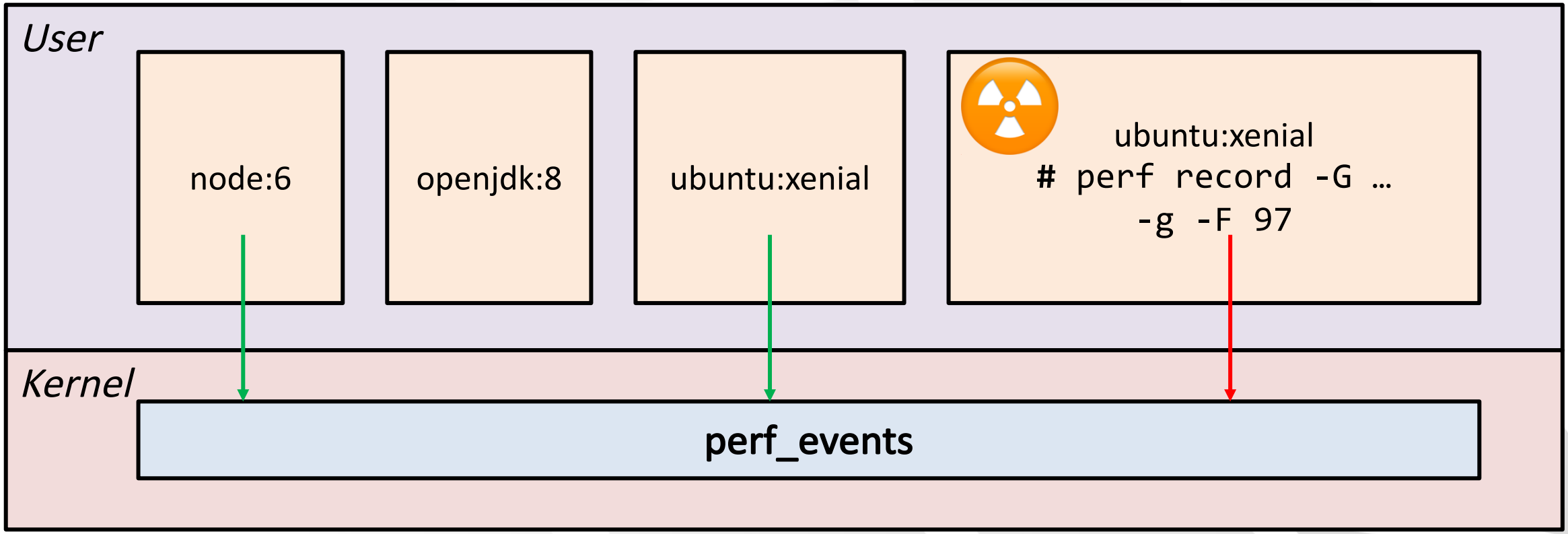
- `jps` and `jstat` will work if volume-mapping /tmp from the container and running as the same user
- `jcmd`, `jmap`, `jinfo`, and `jstack` will work by using jattach with my PR that enables namespace support:

```
$ sudo jattach $PID $COMMAND
```

Using JVM Tools on The Host

Demo

Tool Deployment: “Sidecar” Container



Retrieving Container Resource Utilization

- *High-level, Docker-specific:* `docker stats`
- `htop` *with cgroup column (highly unwieldy)*
- `systemd-cgtop`
- *Control group metrics*
  - E.g. **/sys/fs/cgroup/cpu,cpuacct/*/*/***
- *Third-party monitoring solutions: cAdvisor, Intel Snap, PCP, Prometheus, collectd*

In-Container Monitoring

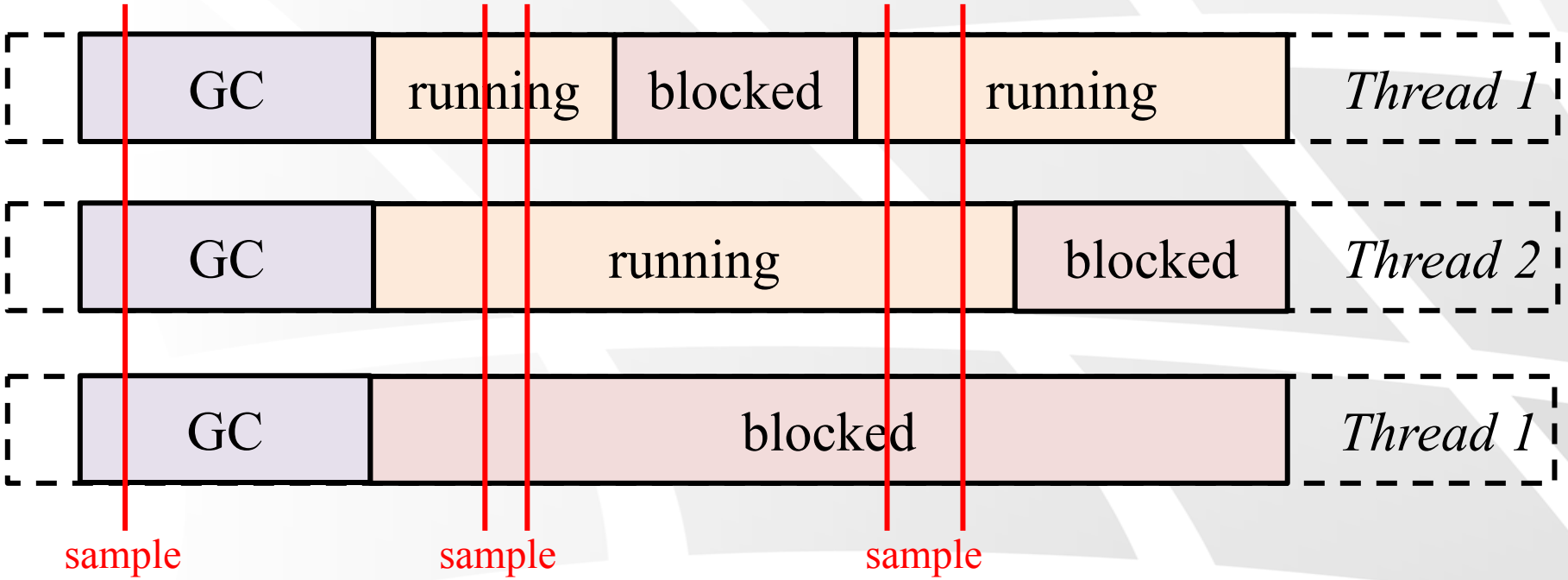
- Execute a command "in the container":
  - `nsenter -t $PID -p -m top`
  - `docker exec -it $CID top`
- "Attach" a container with debug tools to the target container:
  - `docker run -it --pid=container:target \`
    `--cap-add sys_admin \`
    `debugcontainer …`
  - The target's filesystem is in `/proc/1/root` if you need it

Monitoring Container Resource Utilization

Demo

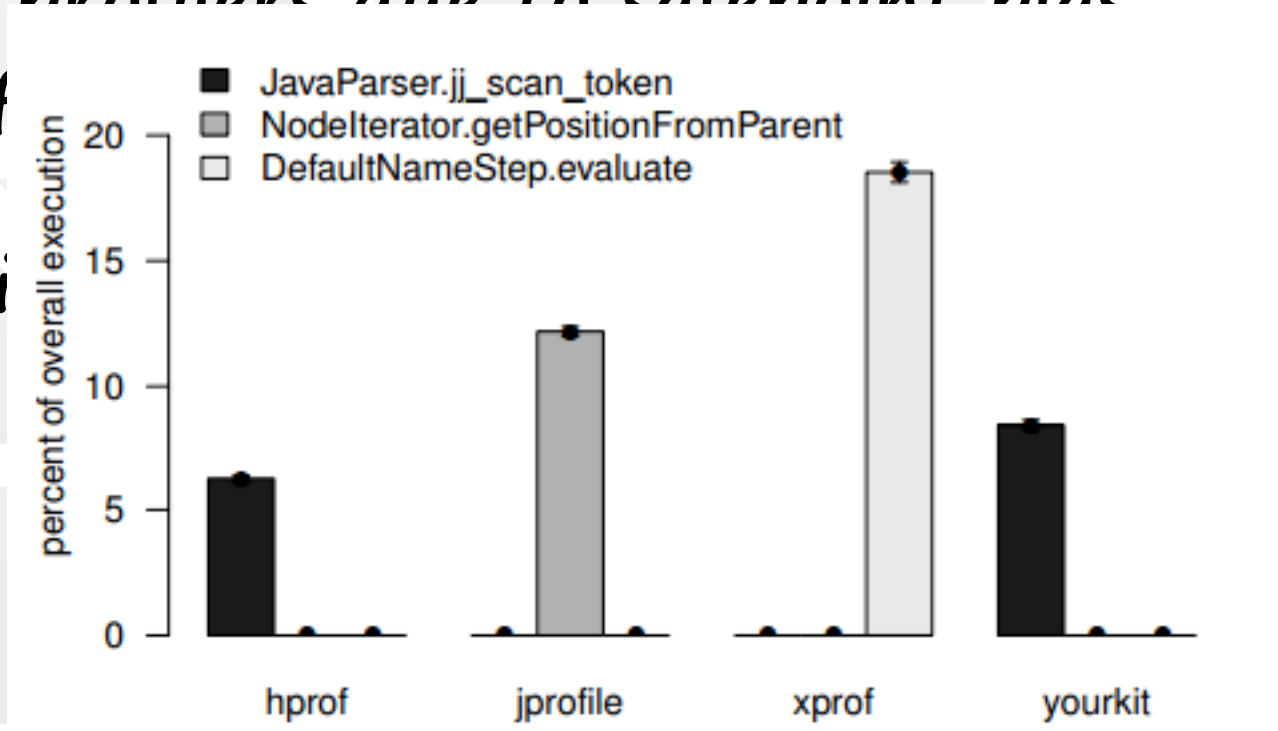
JVM Stack Sampling

- Traditional CPU profilers sample all thread stacks periodically (e.g. 100 times per second)
  - Typically use the JVMTI `GetAllStackTraces` API
  - `jstack`, JVisualVM, YourKit, JProfiler, and a lot of others



Safepoint Bias

- Samples are captured only at safepoints
- Research Evaluating The Accuracy of Java Profilers by Mytkowicz, Diwan, Hauswirth, Sweeney shows wild variety of results between profilers due to safepoint bias
- Additionally, capturing a f
  stack trace for all threads
  quite expensive (think Spri

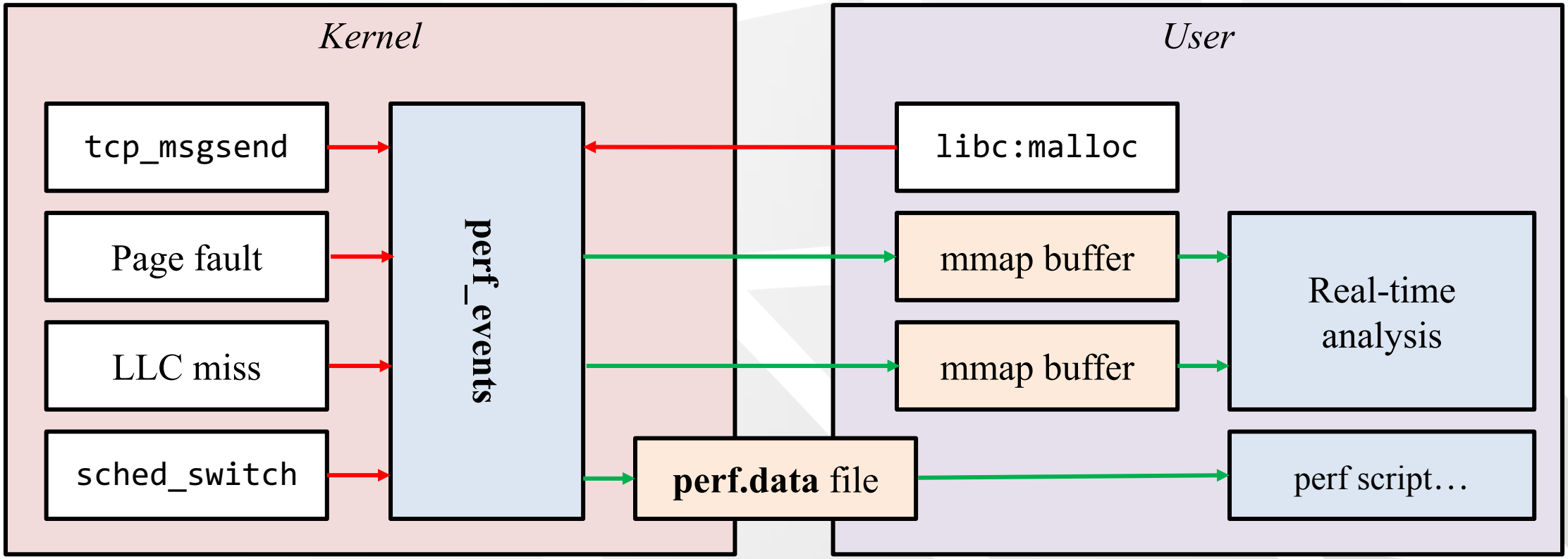


perf

- `perf` *is a Linux multi-tool for performance investigations*
- *Capable of both tracing and sampling*
- *Developed in the kernel tree, must match running kernel's version*

- *Debian:* `apt install linux-tools-common`
- *Red Hat:* `yum install perf`

perf_events Architecture



Recording CPU Stacks With perf

- *To find a CPU bottleneck, record stacks at timed intervals:*

```
# system-wide
perf record -ag -F 97
# specific process
perf record -p 188 -g -F 97
# specific cgroup
perf record -G docker-1ae… -g -F 97
```

Legend

-a	all CPUs
-p	specific process
-G	specific cgroup
-g	capture call stacks
-F	frequency of samples (Hz)
-c	# of events in each sample

Symbols

- perf needs symbols to display function names (beyond modules and addresses)
  - For compiled languages (C, Go, …) these are often embedded in the binary
  - Or installed as separate debuginfo (usually /usr/lib/debug)

```
$ objdump -tT /usr/bin/bash | grep readline
0000000000306bf8 g    DO .bss  0000000000000004  Base        rl_readline_state
00000000000a46c0 g    DF .text 00000000000001d4  Base        readline_internal_char
00000000000a3cc0 g    DF .text 0000000000000126  Base        readline_internal_setup
000000000078b80 g     DF .text 0000000000000044  Base        posix_readline_initialize
00000000000a4de0 g    DF .text 0000000000000081  Base        readline
00000000003062d0 g    DO .bss  0000000000000004  Base        bash_readline_initialized
…
```

Container-Specific Challenges

- Address to module and symbol resolution, dynamic instrumentation require access to debug information
  - Because of mount namespace, container's binaries and debuginfo are not visible to host (/lib64/libc.so.6 – what?)
  - Need to enter the container's namespace or share the binaries
- `perf` handles this automatically in Linux 4.13+, as do the BCC tools (discussed later)

Generating Map Files

- For interpreted or JIT-compiled languages, map files need to be generated at runtime
- Java: perf-map-agent `./create-java-perf-map.sh $PID`
- Node: `node --perf-basic-prof-only-functions app.js`
- .NET Core: `export COMPlus_PerfMapEnabled=1`
- When profiling from the host:
  - PID namespace – always /tmp/perf-1.map in the container, not the host
  - Mount namespace – container's /tmp/perf-1.map not visible to host
  - Again, `perf` and BCC tools can handle this automatically

Container CPU Profiling

Demo

Running perf in a Container

- Some syscalls are blocked by default:
  - perf_event_open
  - Blocked syscalls can be whitelisted in the seccomp configuration, but this opens up risks for the host system [source]
- Need to put profiling tools in the container
  - Bloats the image, increases attack surface
- Need a couple of sysctl tweaks to run unprivileged:

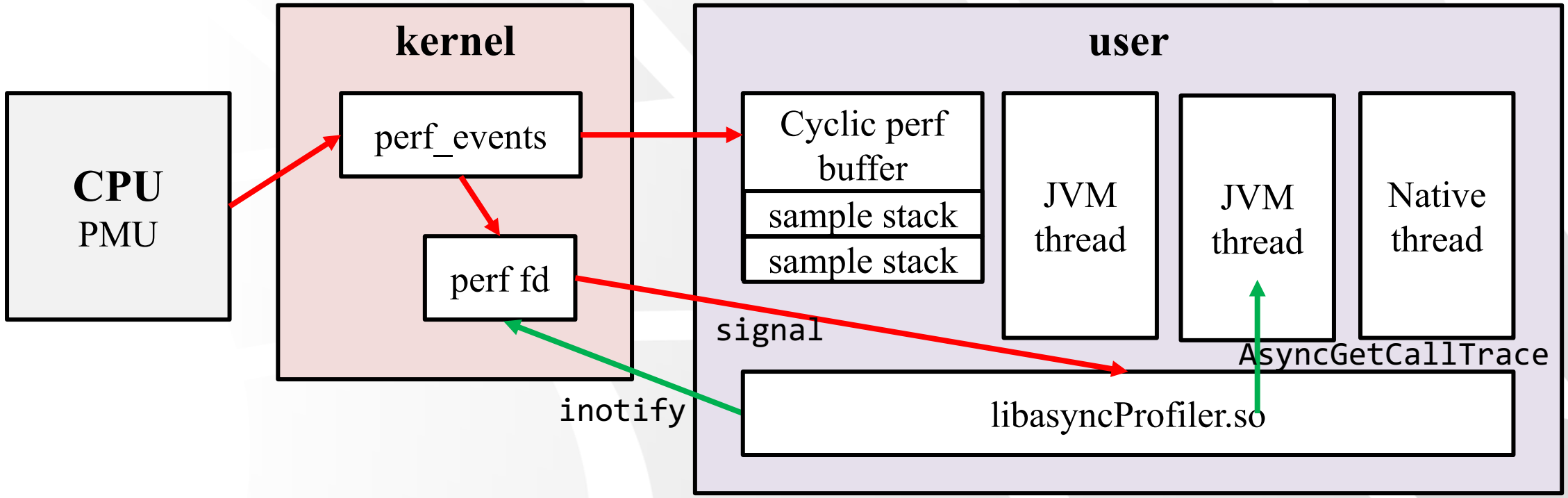
```
# echo 0  > /proc/sys/kernel/kptr_restrict
# echo -1 > /proc/sys/kernel/perf_event_paranoid
```

AsyncGetCallTrace

- Internal API introduced to support lightweight profiling in Oracle Developer Studio
- Produces a single thread's stack without waiting for safepoint
- Designed to be called from a signal handler
- Used by Honest Profiler (by Richard Warburton and contributors): https://github.com/jvm-profiling-tools/honest-profiler

async-profiler

- Open source profiler by Andrei Pangin and contributors: https://github.com/jvm-profiling-tools/async-profiler



Profilers, Compared

perf

- Java ≧8u60 to disable FPO
- Disabling FPO has a perf penalty
- Need a map file
- Interpreter frames are not supported
- System-wide profiling is possible

async-profiler

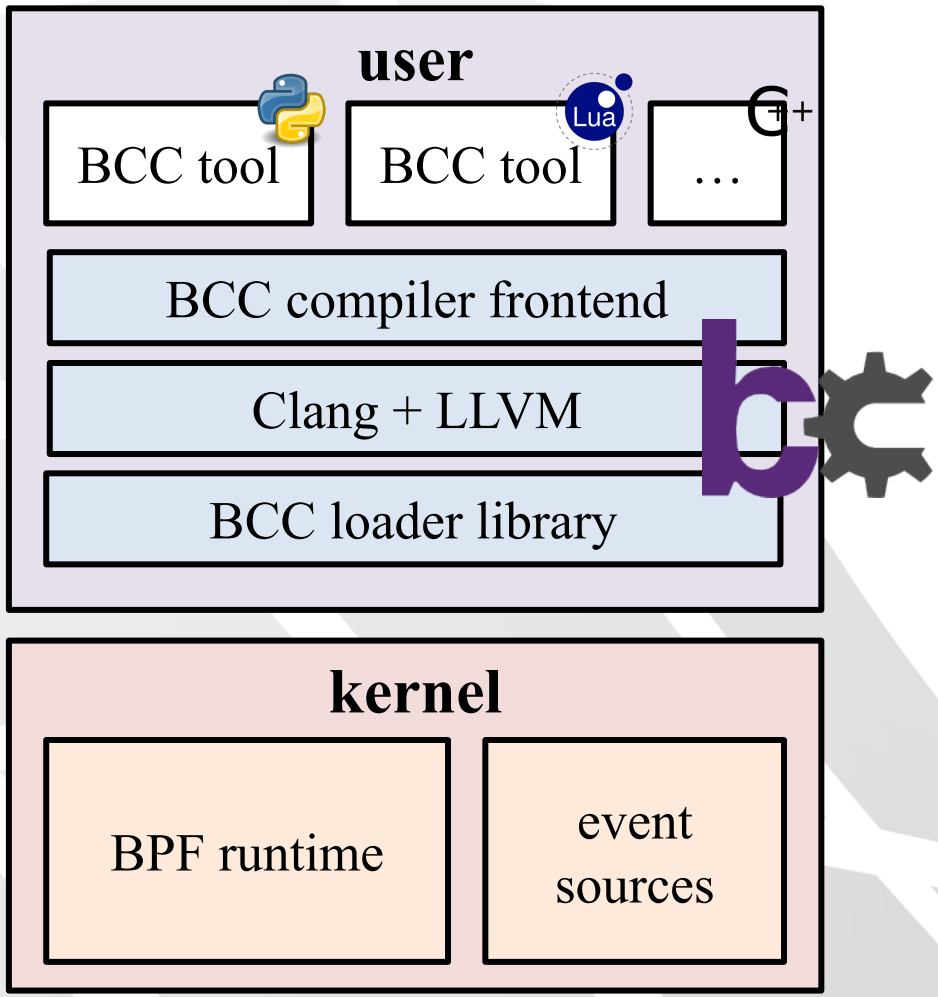
- Works on older Java versions
- FPO can stay on
- No map file is required
- Interpreter frames are supported
- In theory, native and Java stacks don't always sync

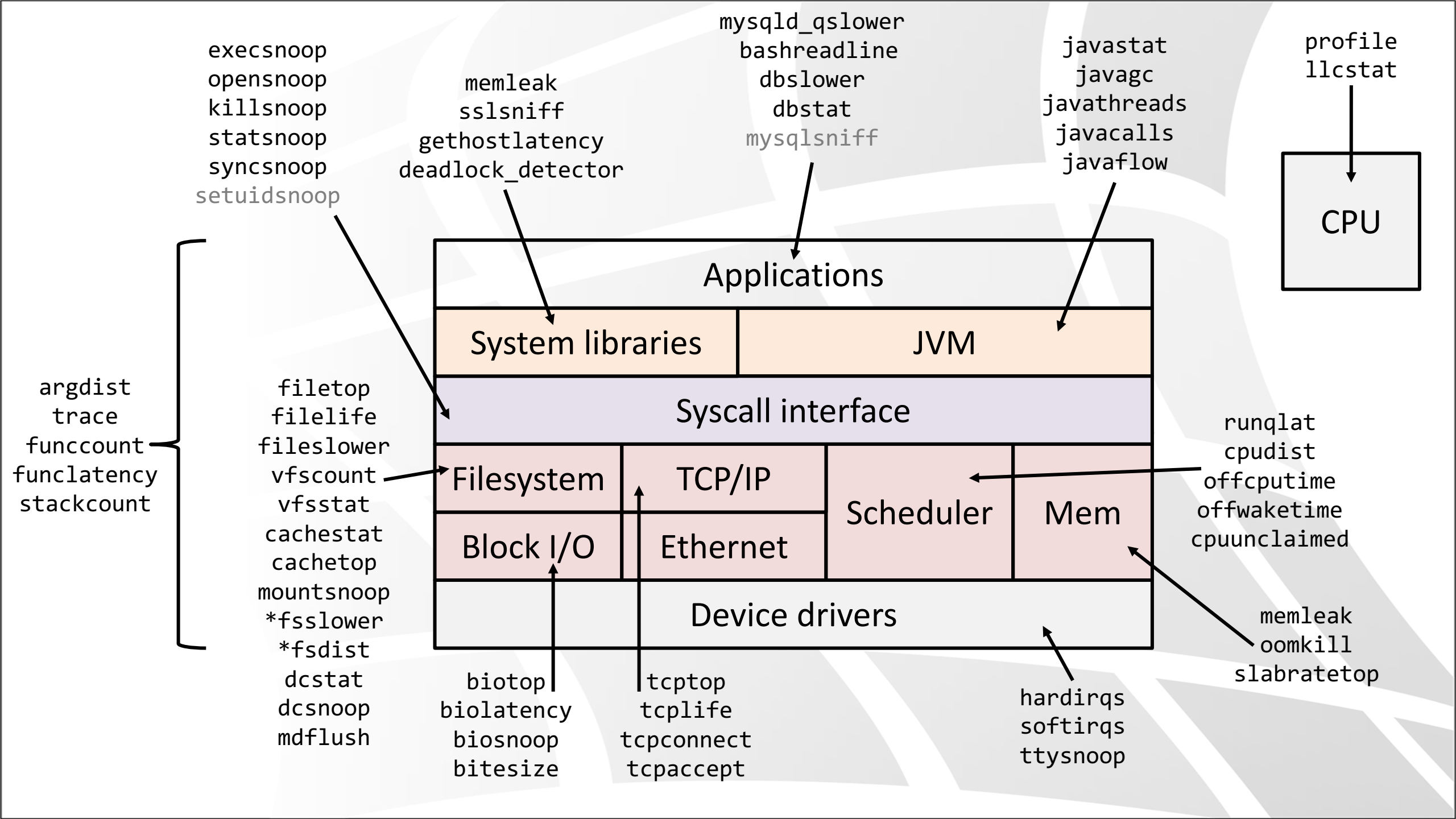
Using async-profiler

Demo

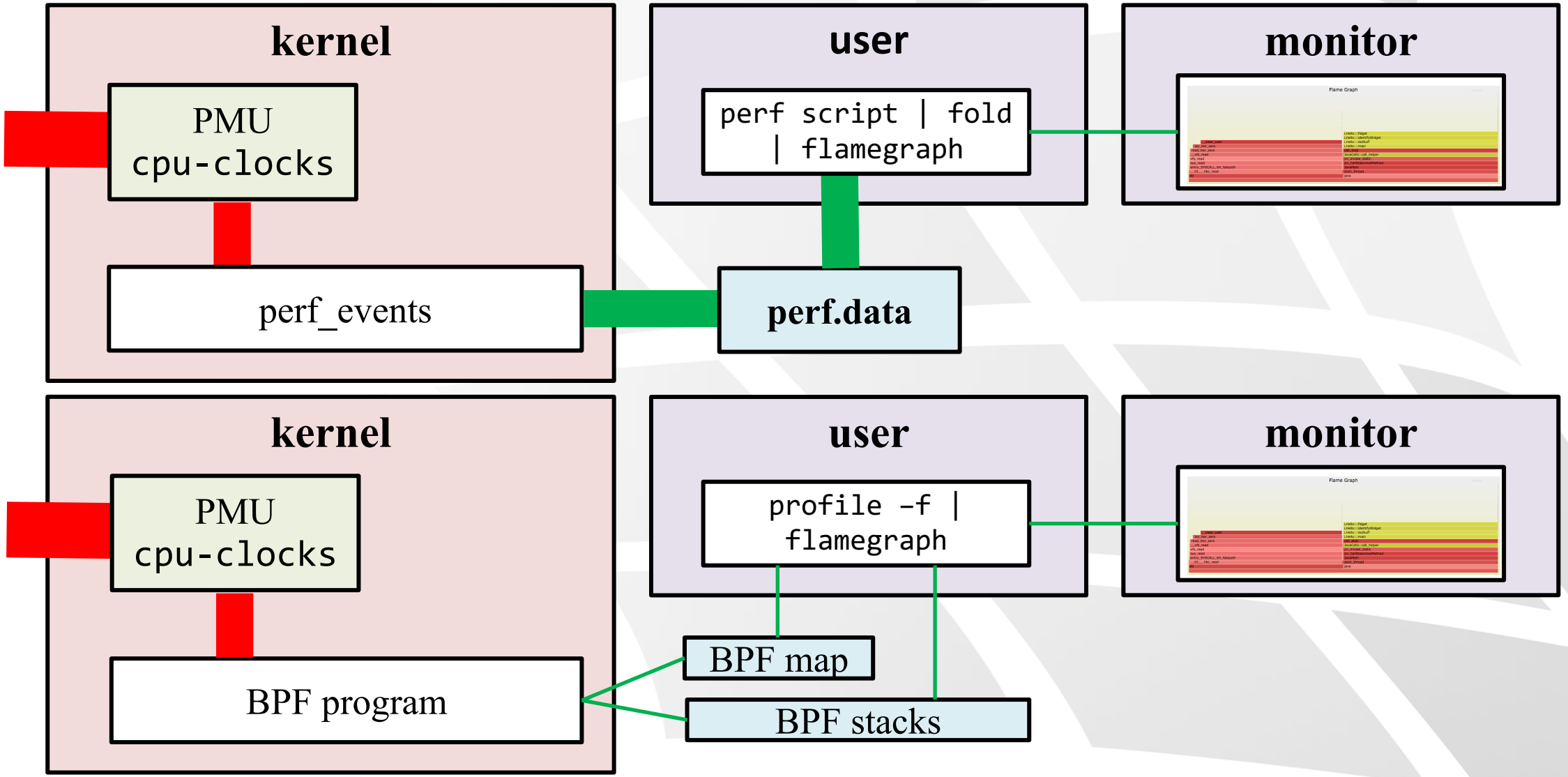
The BCC BPF Front-End

- https://github.com/iovisor/bcc
- BPF Compiler Collection (BCC) is a BPF frontend library and a massive collection of performance tools
  - Contributors from Facebook, PLUMgrid, Netflix, Sela
- Helps build BPF-based tools in high-level languages




execsnoop
opensnoop
killsnoop
statsnoop
syncsnoop
setuidsnoop
memleak
sslsniff
gethostlatency
deadlock_detector
mysqld_qslower
bashreadline
dbslower
dbstat
mysqlsniff
javastat
javagc
javathreads
javacalls
javaflow
profile
llcstat
CPU
Applications
System libraries
JVM
Syscall interface
Filesystem
TCP/IP
Block I/O
Ethernet
Scheduler
Mem
Device drivers
argdist
trace
funccount
funclatency
stackcount
filetop
filelife
fileslower
vfscount
vfsstat
cachestat
cachetop
mountsnoop
*fsslower
*fsdist
dcstat
dcsnoop
mdflush
biotop
biolatency
biosnoop
bitesize
tcptop
tcplife
tcpconnect
tcpaccept
runqlat
cpudist
offcputime
offwaketime
cpuunclaimed
hardirqs
softirqs
ttysnoop
memleak
oomkill
slabratetop

BCC's `profile` Tool



Identifying CPU Throttling

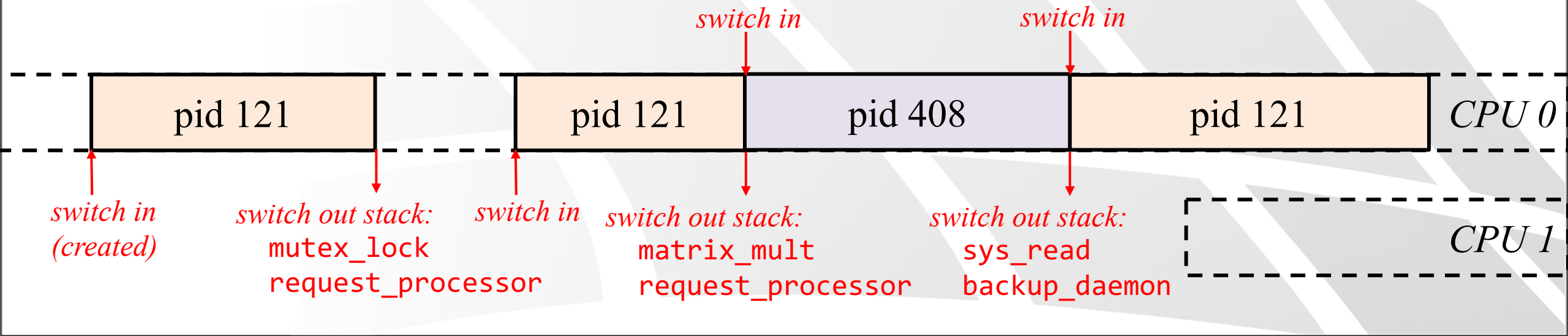
- CPU share throttling and CPU caps cause involuntary context switches
  - Host might have available CPUs, but the container's cgroup can't use them
  - Diagnose with /sys/fs/cgroup/cpu,cpuacct/docker/$CID/cpu.stat
  - Diagnose with /proc/$PID/status involuntary_ctxt_switches field
  - Diagnose with `runqlat`, `cpudist`

Identifying Throttling

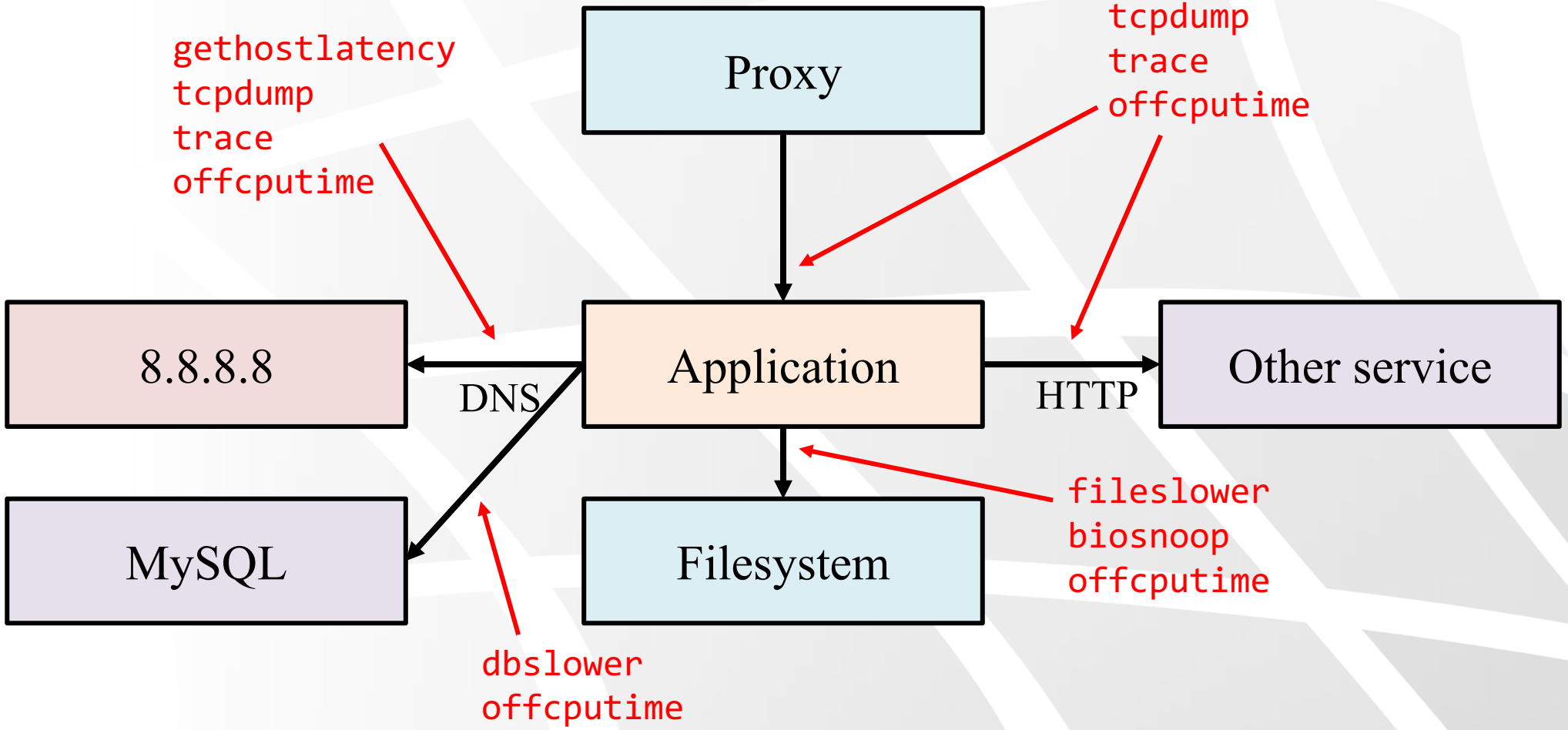
Demo

Blocked Thread Investigation

- CPU sampling only identifies time spent on-CPU
- Blocked time is a concern for most applications
  - Sleep, wait, lock, disk, network, database, …
- Blocked time can be traced using context switch events
  - Linux kernel tracepoint sched:sched_switch



Blocked Time Observability



Turnkey Container Monitoring Solutions

- Consider using a complete monitoring solution that overcomes container monitoring difficulties
  - cAdvisor, Sysdig, Datadog, New Relic, etc.
- There will always be room for low-level troubleshooting using perf/BPF/other tools

Summary

- Mission:
  Apply modern production-ready tools for performance monitoring and profiling of Java applications in Linux containers
- Objectives:
  - ✓ Identifying overloaded resources in containers
  - ✓ Understanding which tools work and which don't in container scenarios
  - ✓ Profiling CPU bottlenecks
  - ✓ Visualizing and exploring stack traces using flame graphs
  - ✓ Analyzing off-CPU time and CPU throttling

References

- perf and flame graphs
  - https://perf.wiki.kernel.org/index.php/Main_Page
  - http://www.brendangregg.com/flamegraphs.html
  - https://github.com/brendangregg/perf-tools
- BPF and BCC
  - https://github.com/iovisor/bcc/blob/master/docs/tutorial.md
  - https://github.com/iovisor/bpf-docs
- Container performance analysis
  - https://www.slideshare.net/brendangregg/container-performance-analysis
  - http://batey.info/docker-jvm-flamegraphs.html
- Performance tooling support
  - https://lkml.org/lkml/2017/7/5/771
  - https://github.com/iovisor/bcc/pull/1051
- Monitoring tools
  - https://github.com/intelsdi-x/snap
  - http://pcp.io/docs/guide.html
  - https://github.com/bobrik/collectd-docker
  - https://github.com/ivotron/docker-perf

Sasha Goldshtein
CTO, Sela Group

@goldshtn
github.com/goldshtn

Thank You!