Kubecon China 2018

Nabla containers: a new approach to container isolation

Brandon Lum, Ricardo Koller , Dan Williams, Sahil Suneja

IBM Research
https://nabla-containers.github.io

Containers are not securely Isolated

Containers are not securely Isolated

- What does this exactly mean?

- Why are VMs considered secure but not containers?

- How do we improve container isolation?

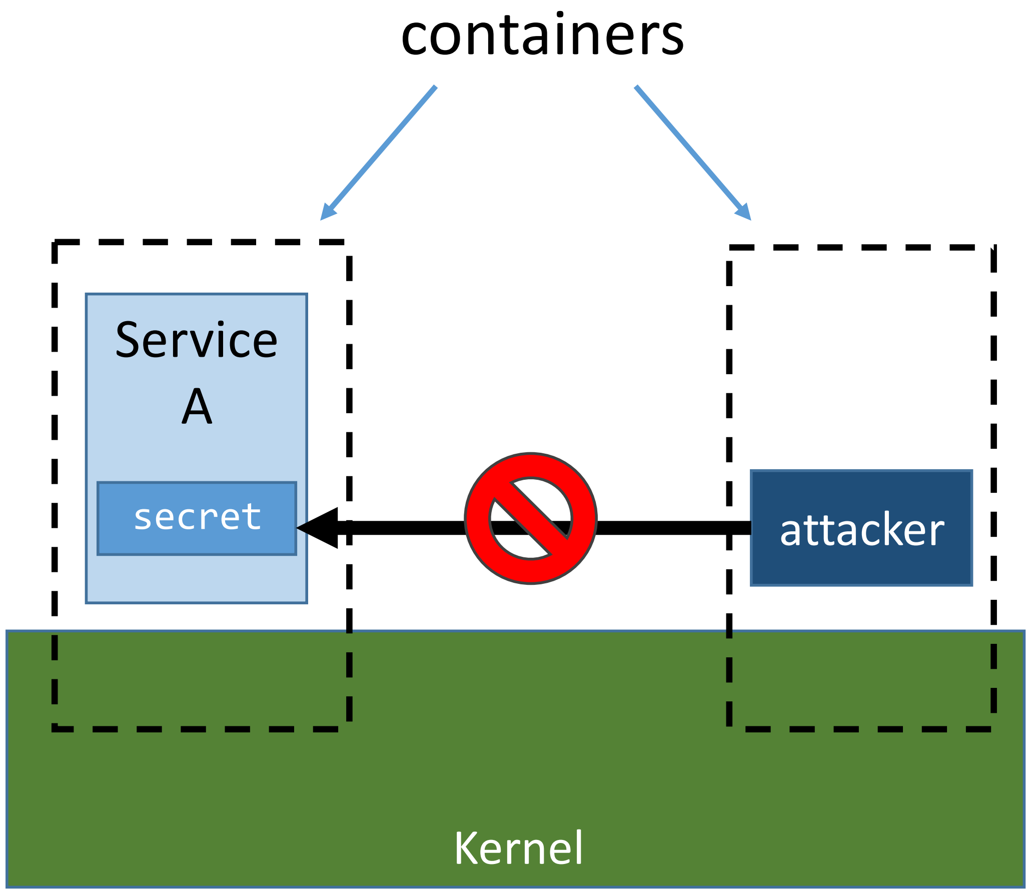
Overview

- Threat Model: Isolation
- Isolation through surface reduction
- Our approach: Nabla
- Measuring Isolation
- Nabla vs VMs?

What does it mean to be isolated?

- Containers that are co-located should not be able to access data of another

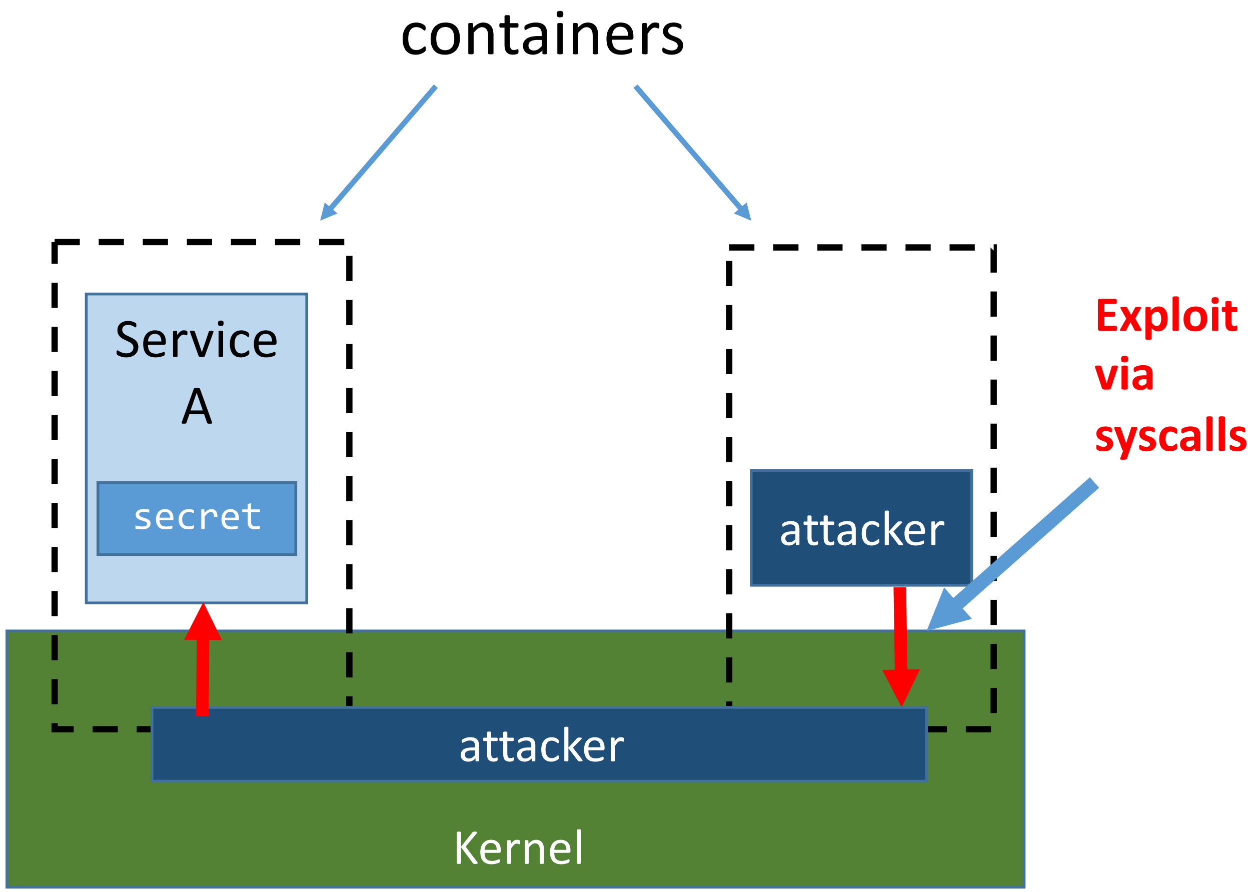
- Scenarios:
  - Horizontal attacks from vulnerable services
  - Container-native multi-tenant cloud



Container Isolation Reality

- Containers == namespaced processes → Kernel exploits mostly work
  - Sep 2018: CVE-2018-14634
  - DirtyCOW (CVE-2016-5195)
  - Many more (CVE database), 2018: Codexec (3), Mem. Corrupt (8)

- Horizontal attack possible via shared privileged component (kernel)



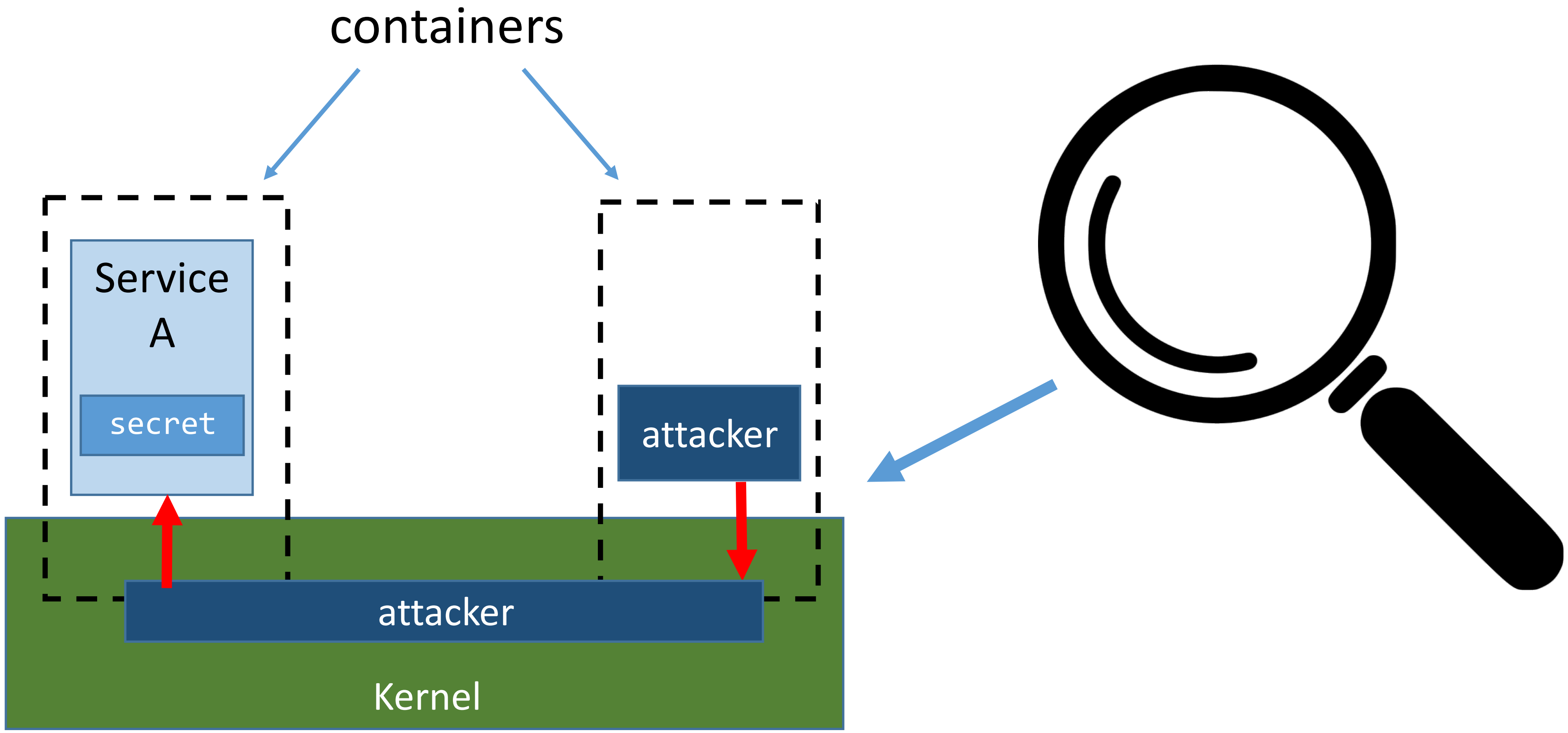
DirtyCOW

- DityCow Exploit Sketch:
  - **mmap** a page
  - Create a thread that invokes **madvise**
  - Create a thread that invokes **Read/Write procfs**

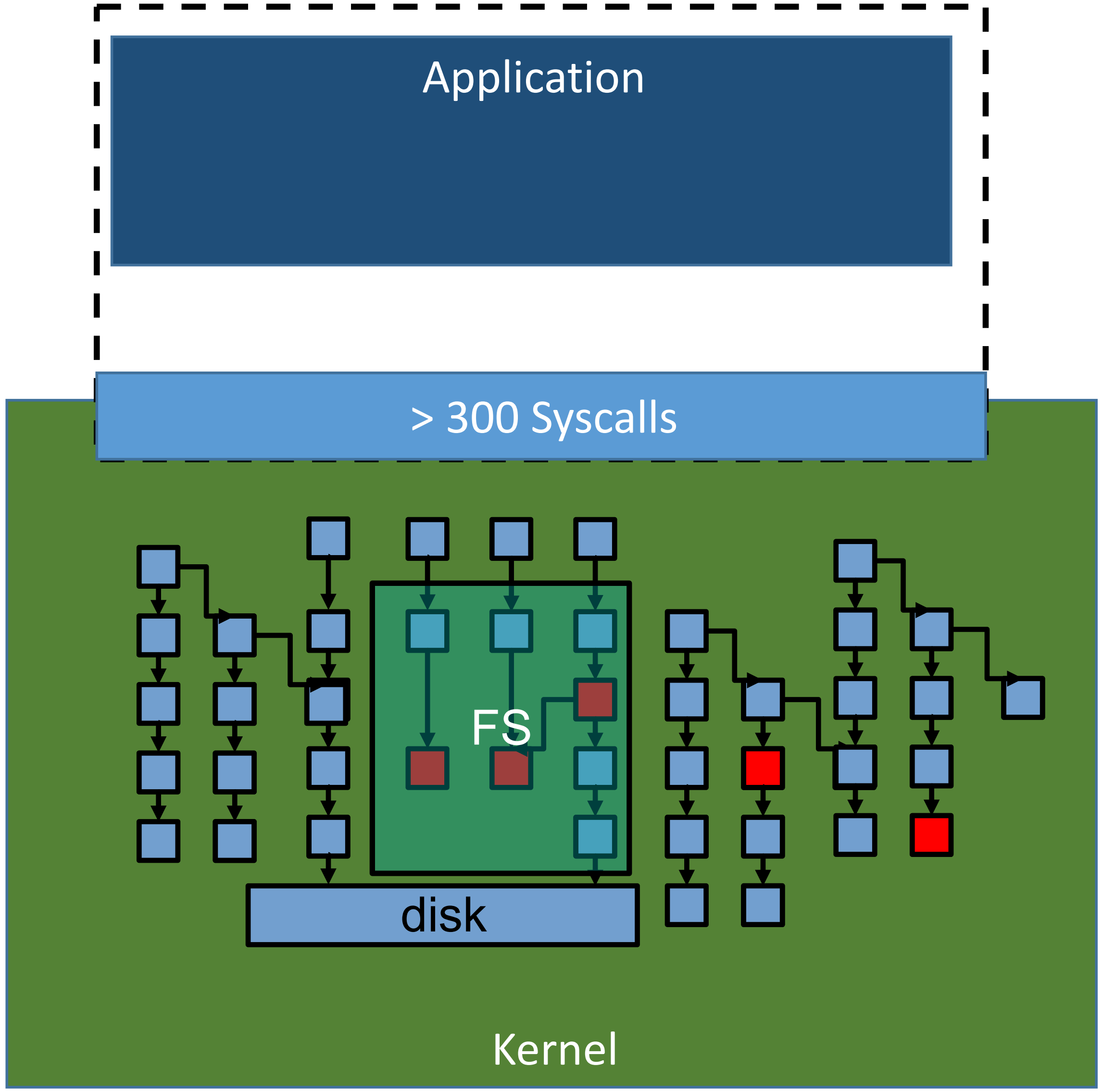
- Triggers race condition in Kernel Mem. management code

```
// FROM: https://dirtycow.ninja/


map=mmap(NULL,st.st_size,PROT_READ,MAP_PRIVATE
,f,0); printf("mmap %zx\n\n",(uintptr_t) map);


/* You have to do it on two threads. */
pthread_create(&pth1,NULL,madviseThread,argv[1
]);  //madvise
pthread_create(&pth2,NULL,procselfmemThread,ar
gv[2]);
// R/W procfs


/* You have to wait for the threads to finish.
*/ pthread_join(pth1,NULL);
pthread_join(pth2,NULL); return 0;
```

Container Isolation Reality



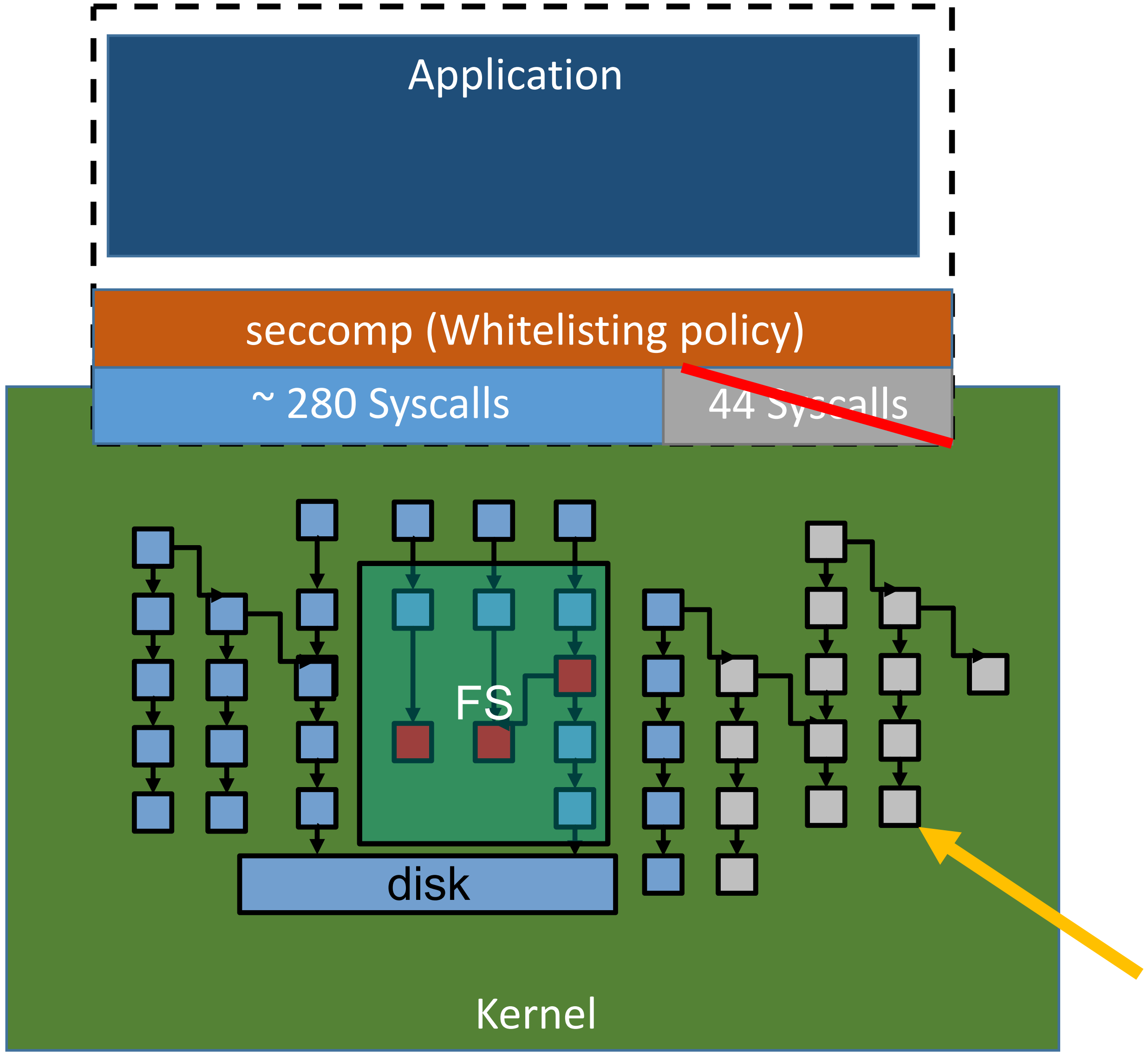
Kernel Footprint



- Exploits target vulnerable part of kernel via syscalls.

- If we restrict the number of syscalls
- → Less reachable kernel functions
- → Less potential vulnerabilities
- → Less possible exploits

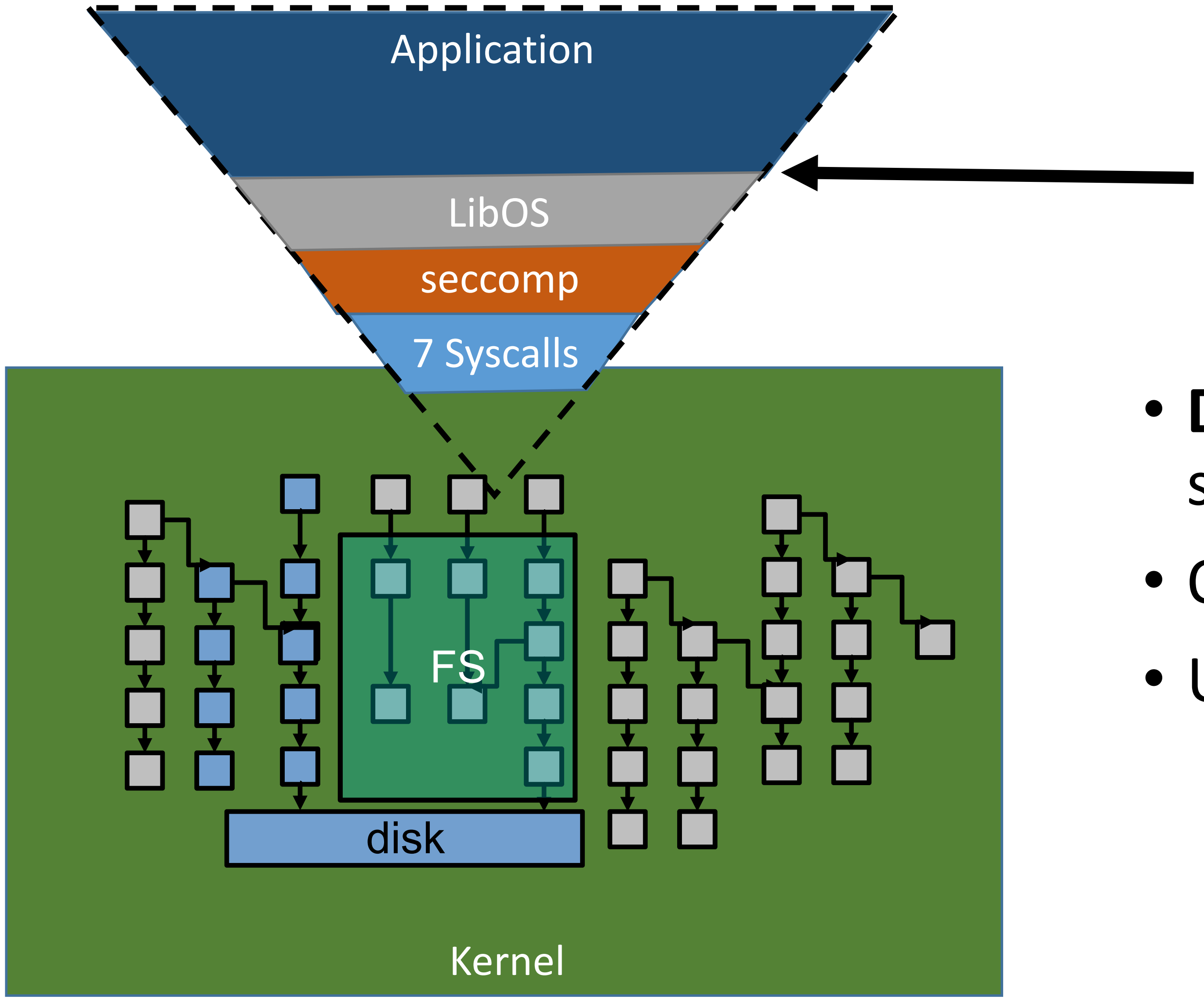
Docker Default Seccomp Policy



- Docker default seccomp policy
  - disables around 44 system calls out of 300+.
- Generic seccomp policies – hard to create s.t. it is secure
- Syscall profiling is mostly heuristic based

Greyed – unreachable functions

Nabla



Original 300+ Syscall interface*

- **Deterministic** and **generic** seccomp policy
- Only 7 syscalls!
- Uses LibOS techniques

Nabla

“Unikernels as Processes”

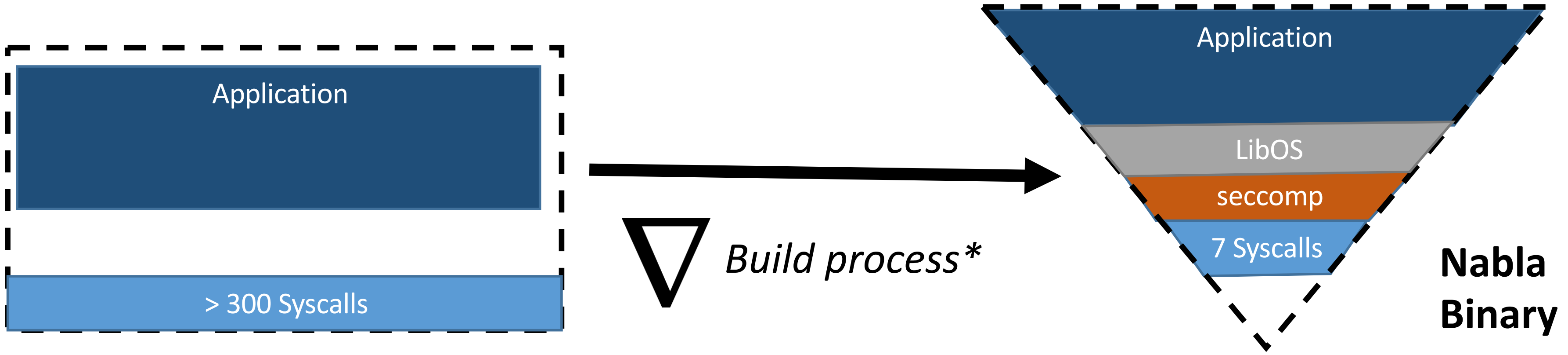
(ACM SoCC ’18)

(https://dl.acm.org/citation.cfm?id=3267845)

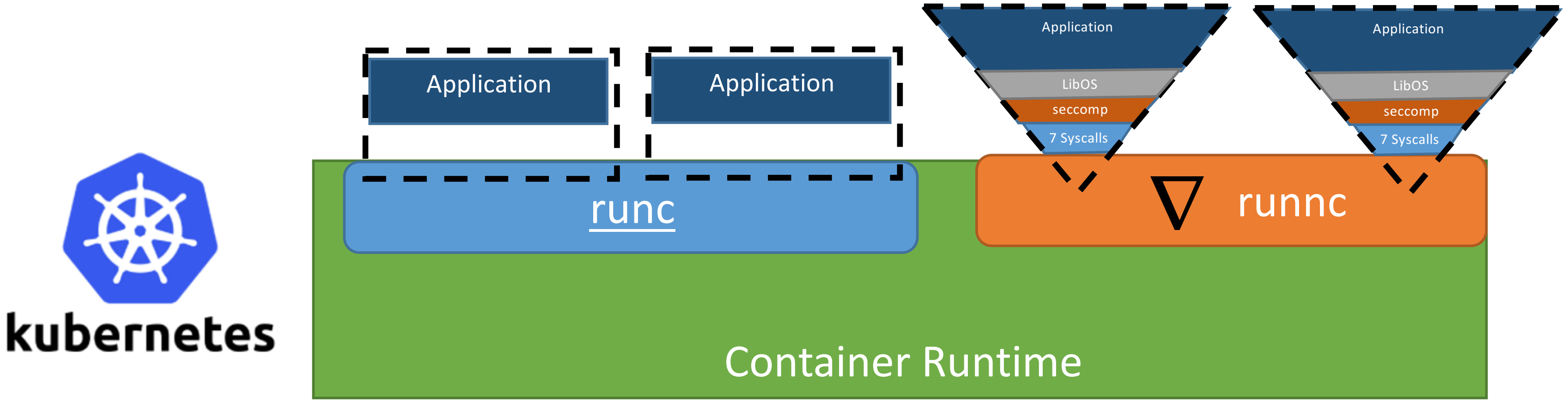
- Taking unikernel ideas and putting it into containers
- Using tools/technologies from the rumprun and solo5 community
- Modify unikernel to work as a process

Making and running a Nabla

- Build app. with custom build process*



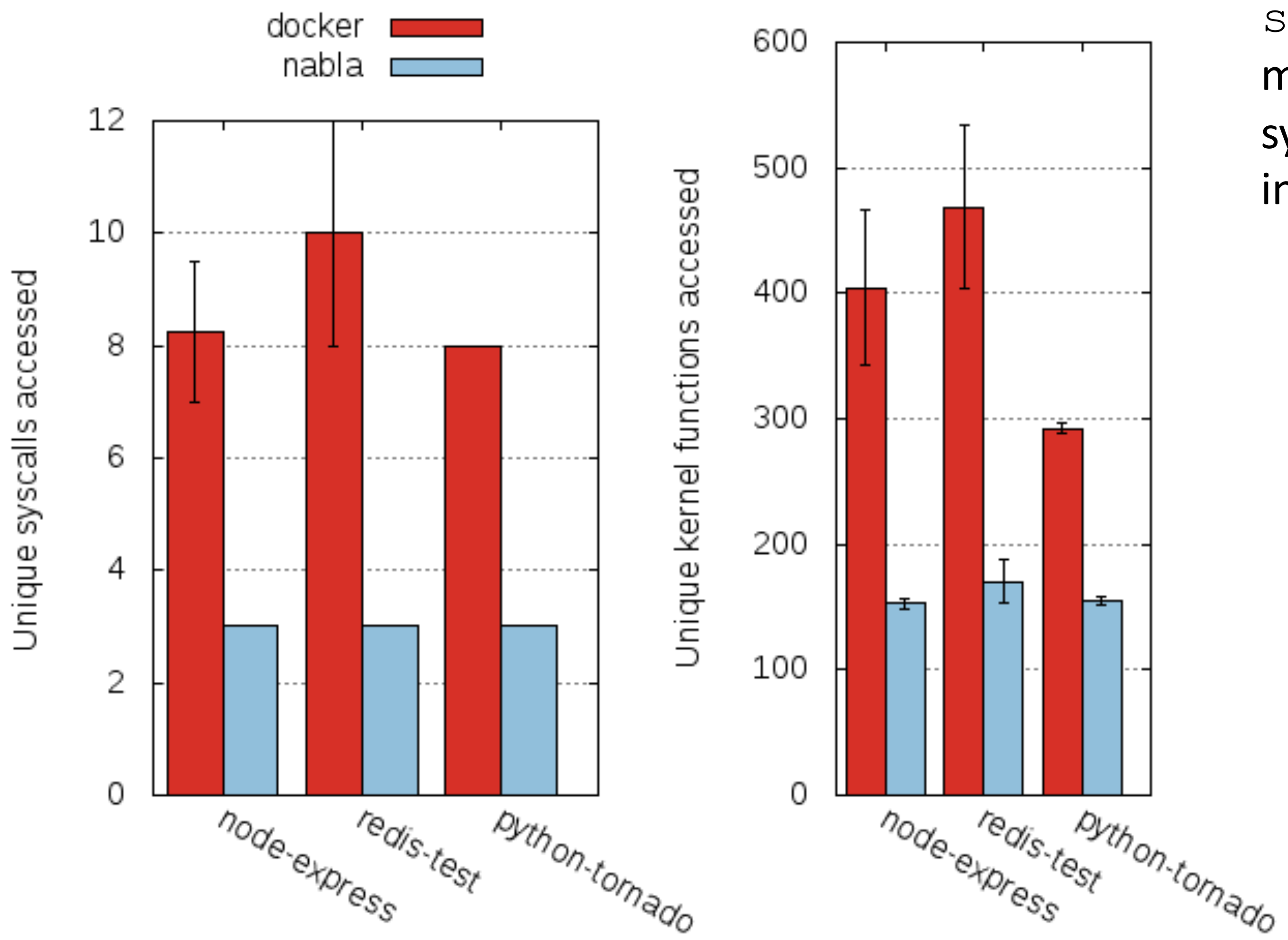
- Nabla runtime, ***runnc*** loads the nabla binaries and sets up seccomp profiles



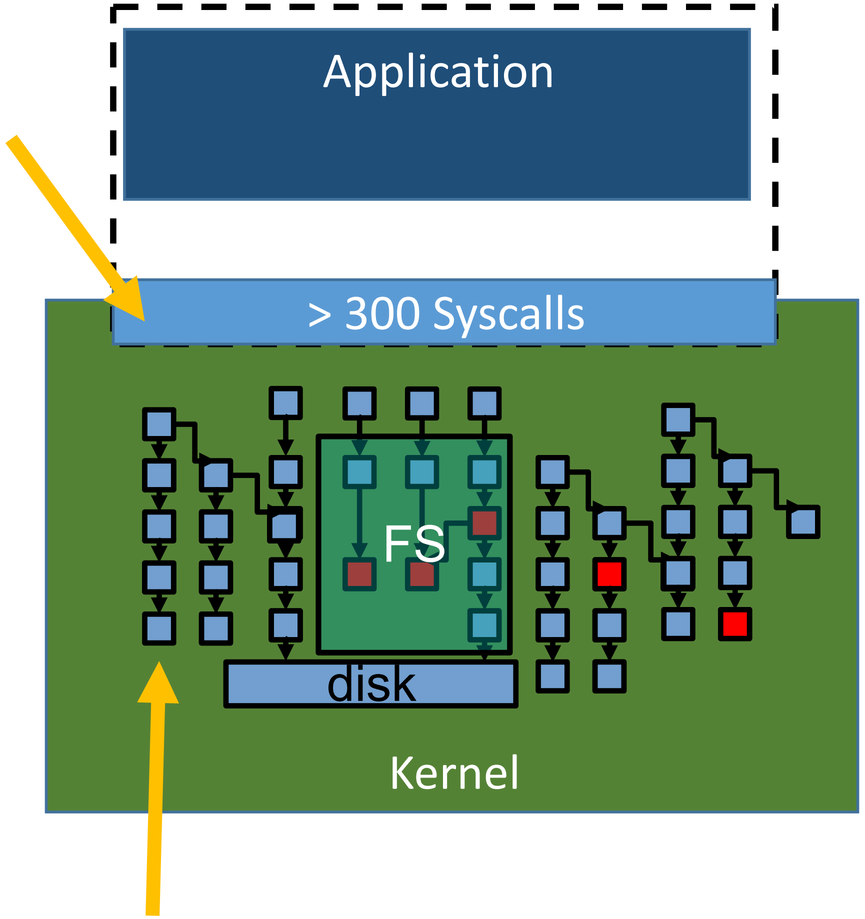
*current limitation of build process, we are investigating ways to consider removing a custom build process

Demo

strace/ftrace measurements (Low is good)

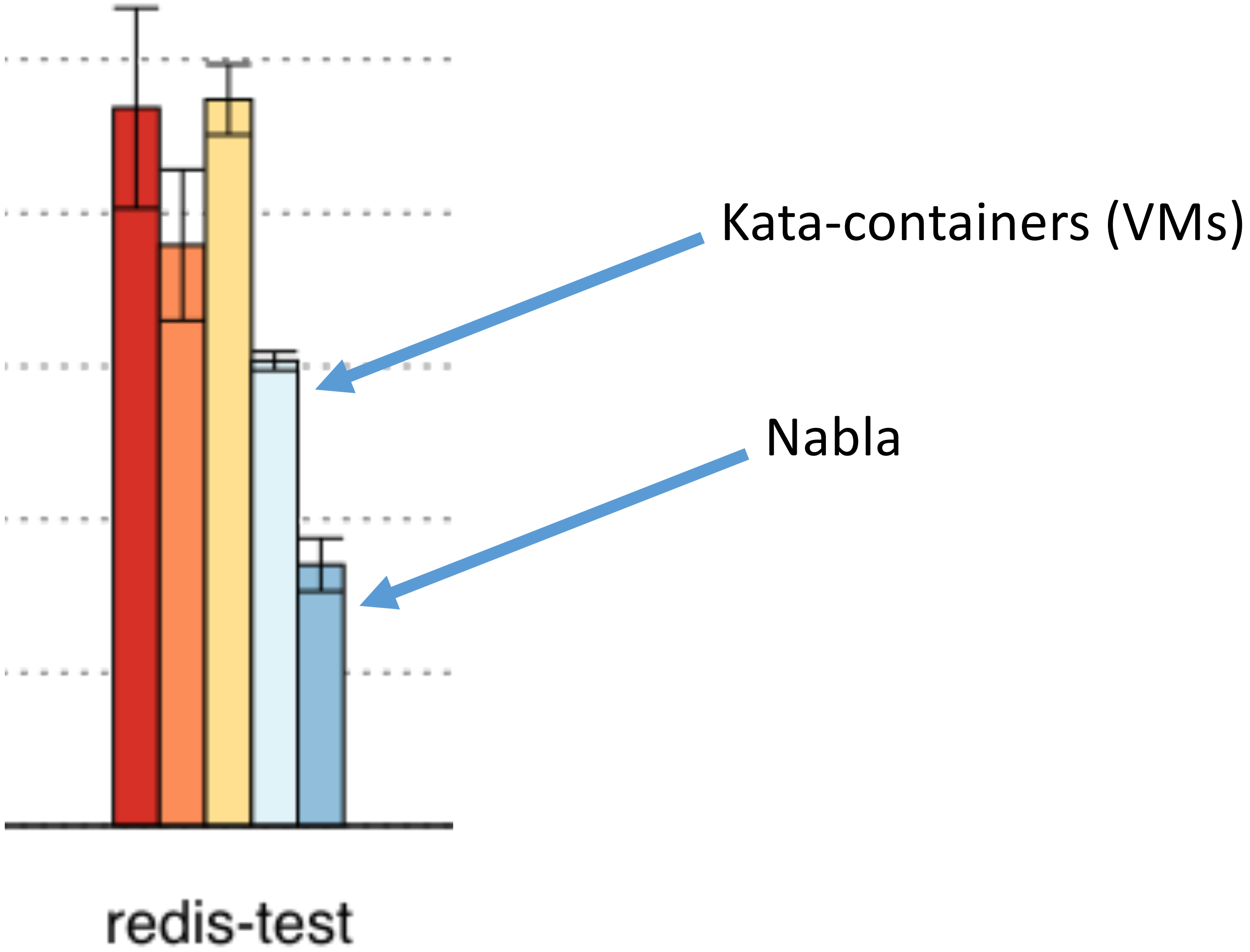


strace measures syscalls invoked.



ftrace measures number of boxes touched.

`ftrace` measurements (lower is better)



Kata-containers (VMs)

Nabla

What does this say about our isolation vs VMs?

Have we surpassed VM isolation?

- We explored and contested this idea in our paper:

“Say Goodbye to Virtualization for a Safer Cloud”
(USENIX HotCloud 2018)
(https://www.usenix.org/conference/hotcloud18/presentation/williams)

- Maybe… But several questions:
  - Implementation specific comparisons? KVM vs other hypervisors
  - Hardware inclusive threat model (Spectre/Meltdown, etc.)
  - Other metrics

What’s Next?

- We want to engage the community:

- Development work for runnc/nabla-base-build/nabla-demo-apps
  - Remove need to rebuild nabla containers (Support for dynamic linking LibOS)
  - Create new images and more language support for applications

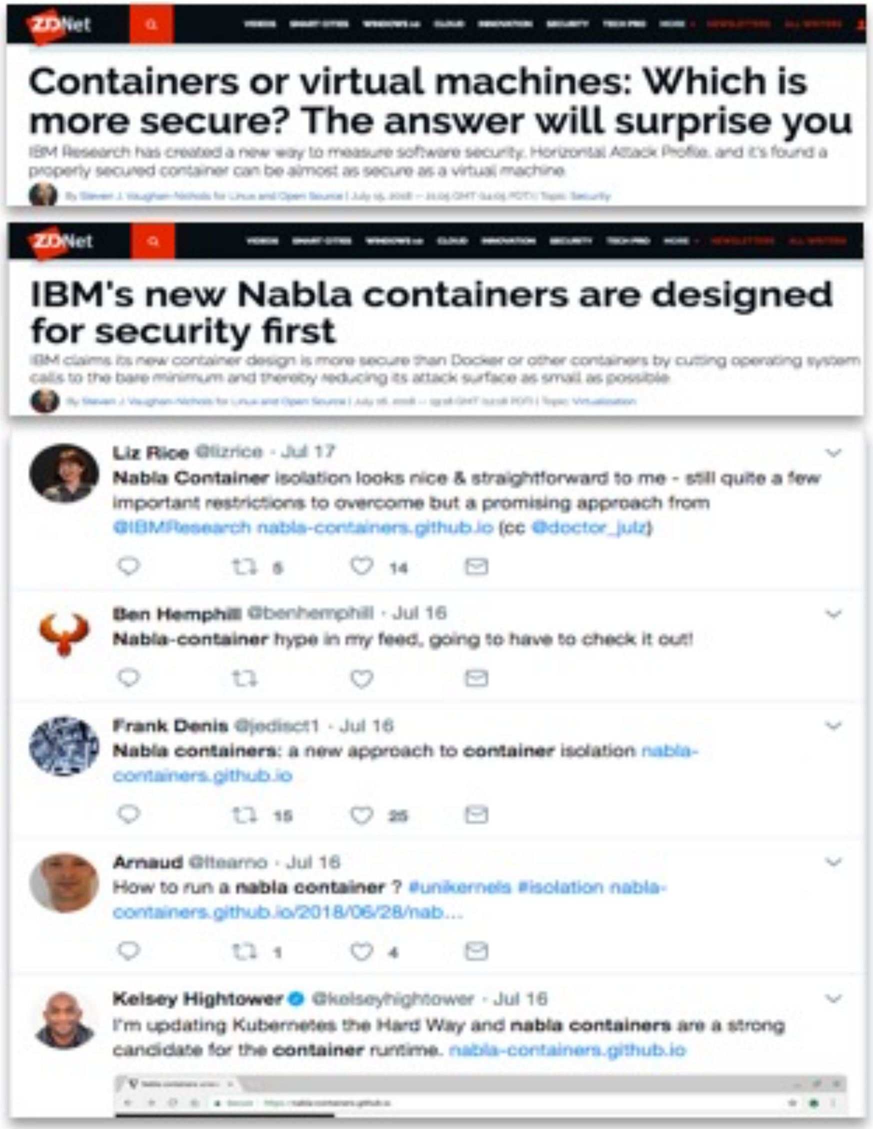
- Chime in on Improving Security Analysis/Metrics
  - https://github.com/nabla-containers/nabla-measurements

Thank You!

https://nabla-containers.github.io

Brandon Lum (@lumjjb) – BRANDON.LUM@ibm.com

#NablaContainers



Containers or virtual machines: Which is more secure? The answer will surprise you

IBM Research has created a new way to measure software security, Horizontal Attack Profile, and it's found a properly secured container can be almost as secure as a virtual machine

By Steven J. Vaughan-Nichols for Linux and Open Source | July 15, 2018 -- 21:05 GMT (14:05 PDT) | Topic: Security

IBM's new Nabla containers are designed for security first

IBM claims its new container design is more secure than Docker or other containers by cutting operating system calls to the bare minimum and thereby reducing its attack surface as small as possible

By Steven J. Vaughan-Nichols for Linux and Open Source | July 16, 2018 -- 19:08 GMT (12:08 PDT) | Topic: Virtualization

Liz Rice @lizrice · Jul 17
Nabla Container isolation looks nice & straightforward to me - still quite a few important restrictions to overcome but a promising approach from @IBMResearch nabla-containers.github.io (cc @doctor_julz)
5 14

Ben Hemphill @benhemphill · Jul 16
Nabla-container hype in my feed, going to have to check it out!

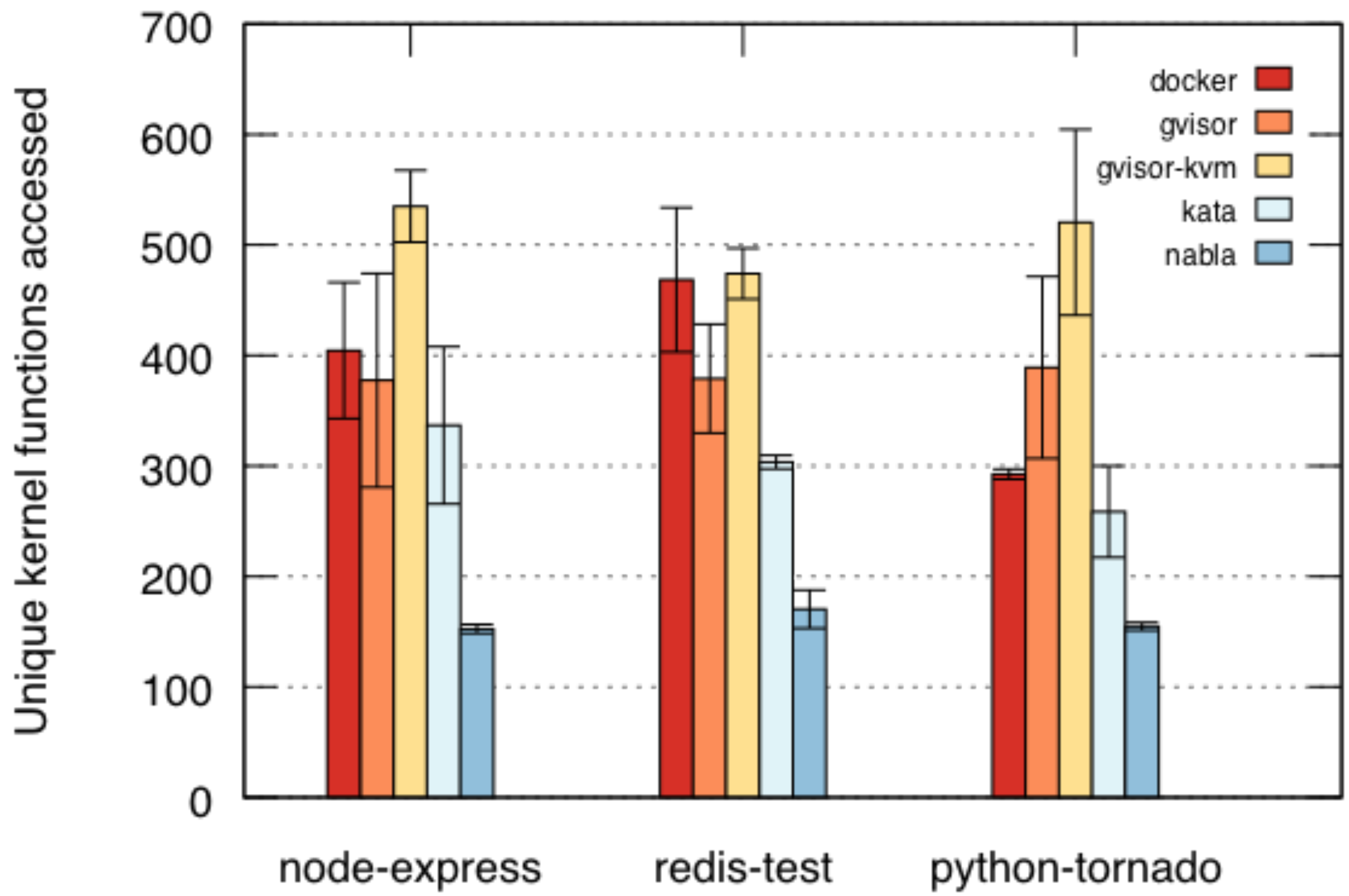
Frank Denis @jedisct1 · Jul 16
Nabla containers: a new approach to **container** isolation nabla-containers.github.io
15 25

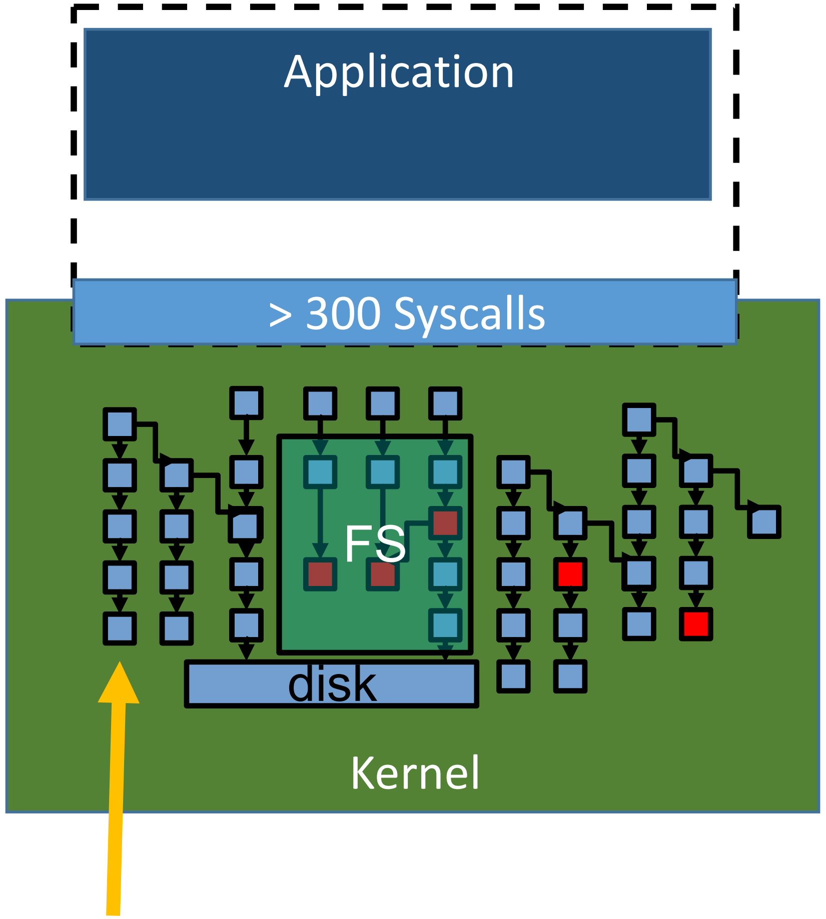
Arnaud @lteamo · Jul 16
How to run a **nabla container** ? #unikernels #isolation nabla-containers.github.io/2018/06/28/nab...
1 4

Kelsey Hightower @kelseyhightower · Jul 16
I'm updating Kubernetes the Hard Way and **nabla containers** are a strong candidate for the **container** runtime. nabla-containers.github.io

Backup

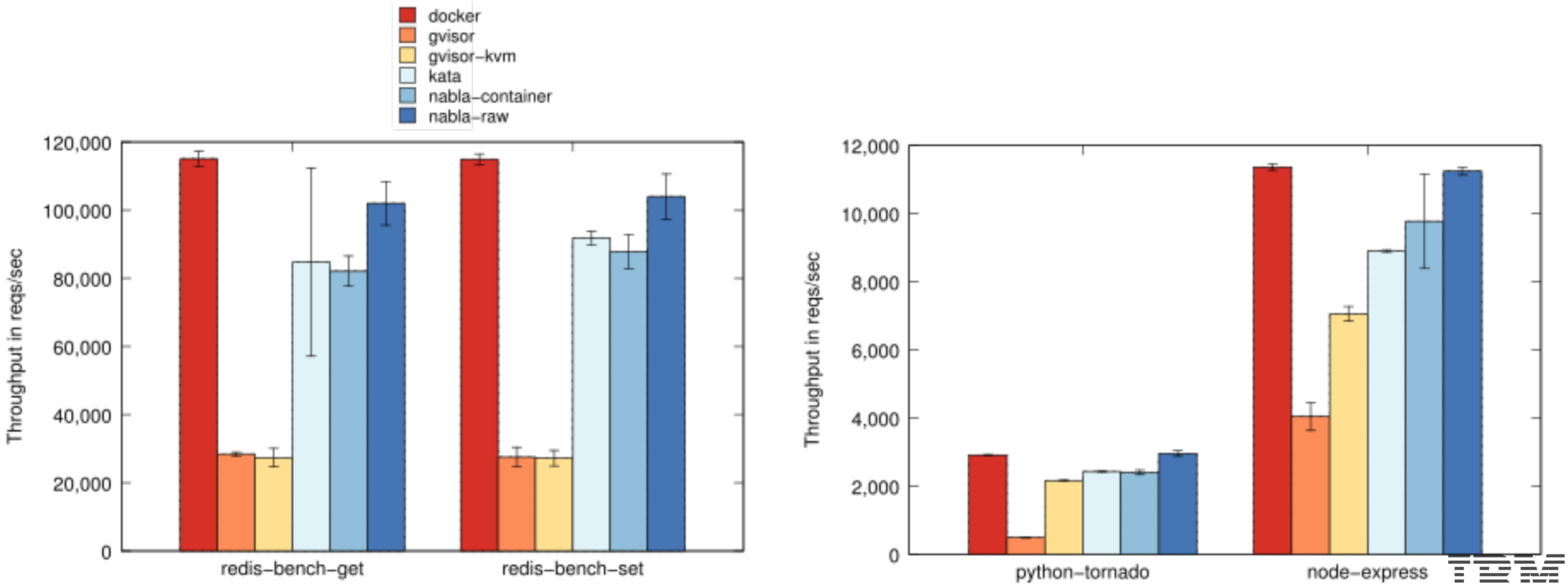
`ftrace` measurements (lower is better)





Measuring number of boxes
Touched.

Throughput (higher is better)



Demo

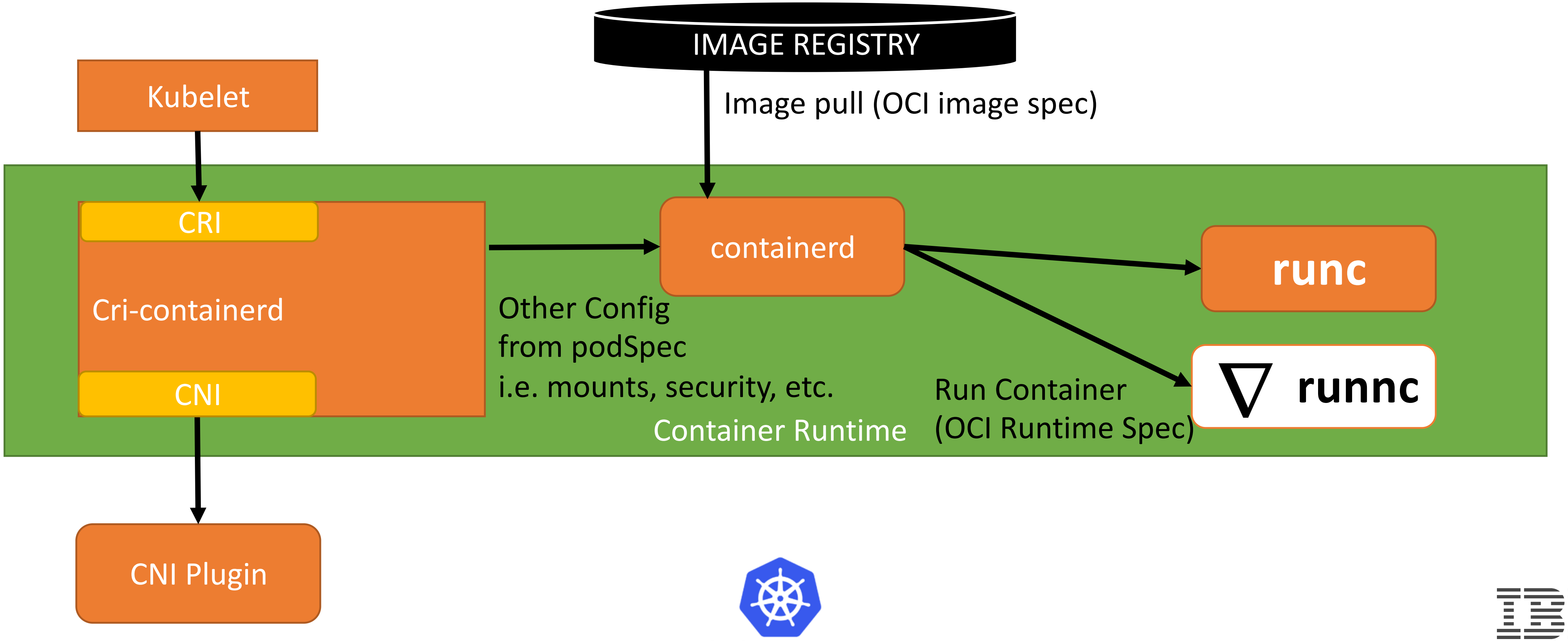
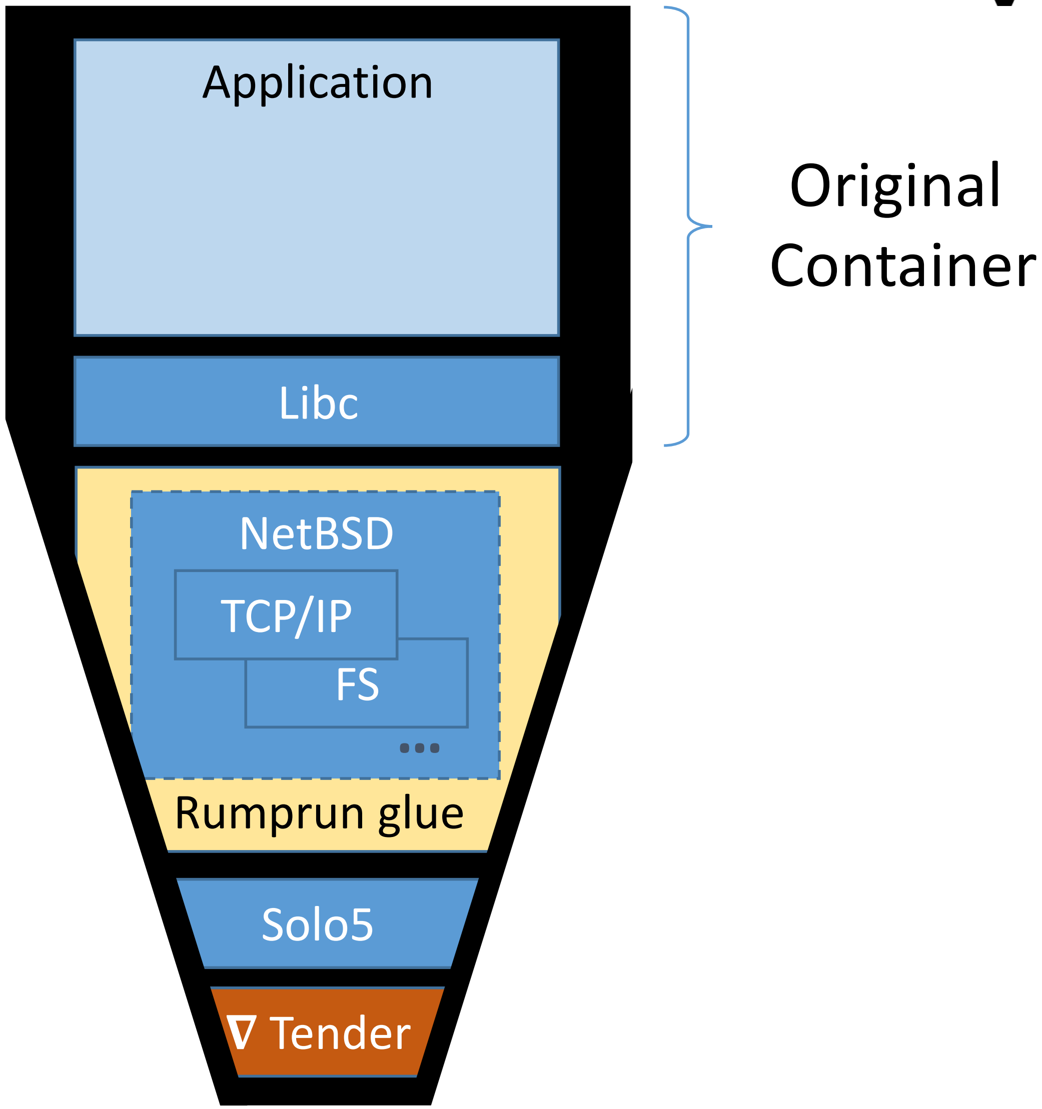


IMAGE REGISTRY
Image pull (OCI image spec)
Kubelet
CRI
Cri-containerd
CNI
containerd
Other Config
from podSpec
i.e. mounts, security, etc.
runc
runnc
Run Container
(OCI Runtime Spec)
Container Runtime
CNI Plugin

Inside a Nabla container

- Unmodified user code (e.g., Node.js, redis, nginx, etc.)
- Rumprun library OS
  - Unmodified NetBSD code + some glue
  - Runs on thin Solo5 unikernel interface
- Nabla Tender
  - Setup of seccomp policy
  - Translates Solo5 calls to system calls



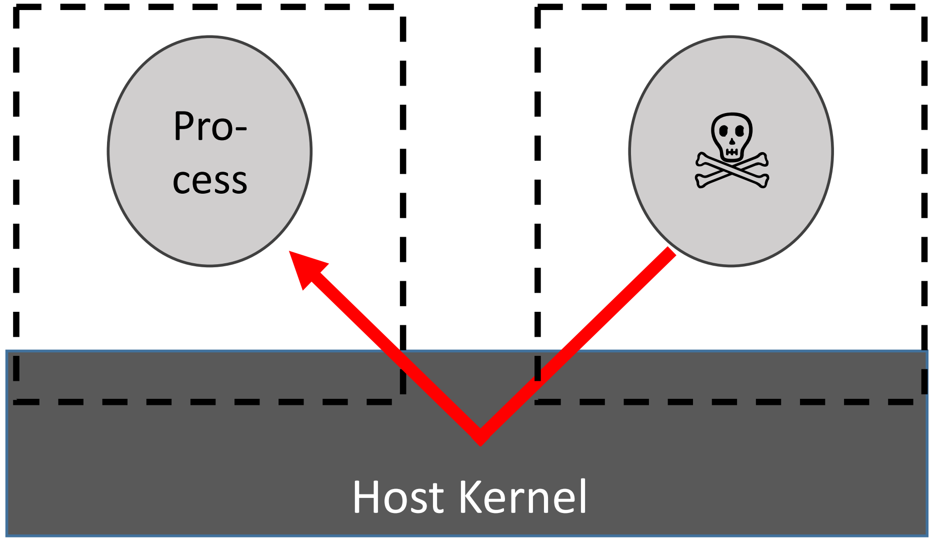
Backup: Containers vs VMs

Overview

- Threat Model: Isolation
- What makes VMs isolated?
- Nabla: How do we get those isolation properties without overhead?

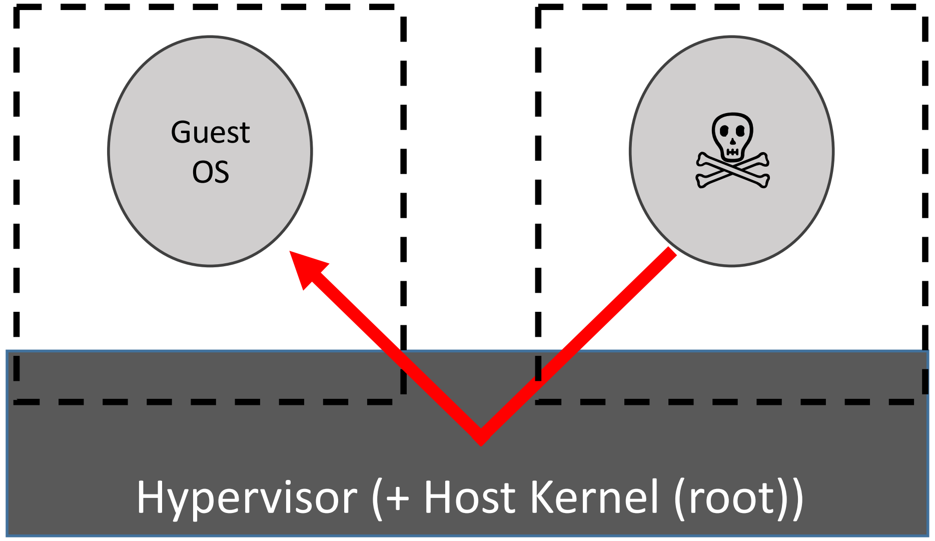
Disclaimer: In this talk, we are doing a 1:1 comparison. Defense in depth is a valid discussion with a different set of trade-offs.

Containers



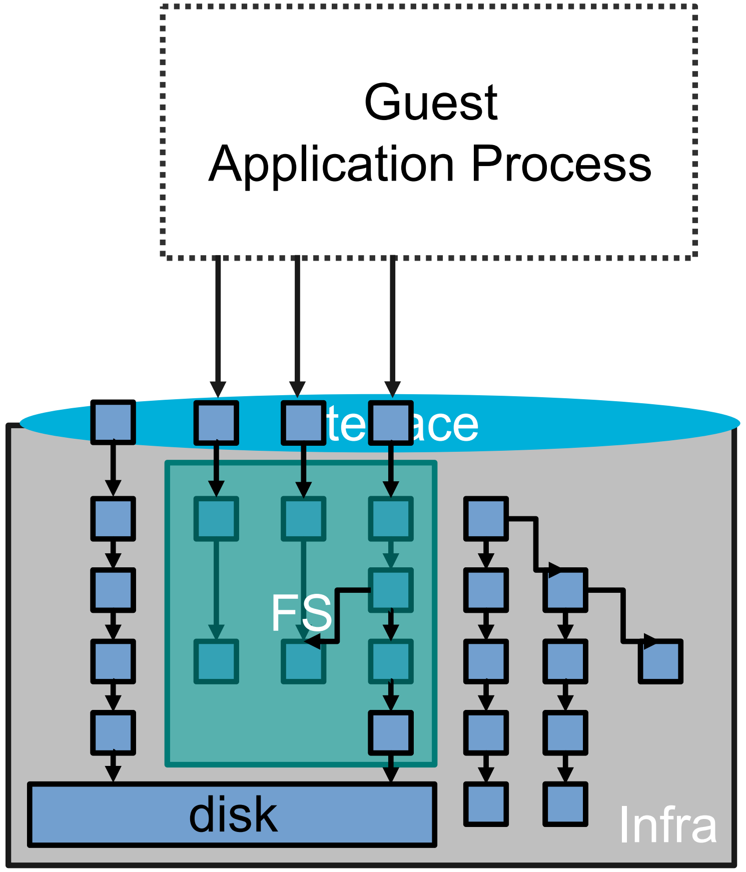
High Level - Syscalls:
Filesystem interface,
socket interface,
etc.

VMs



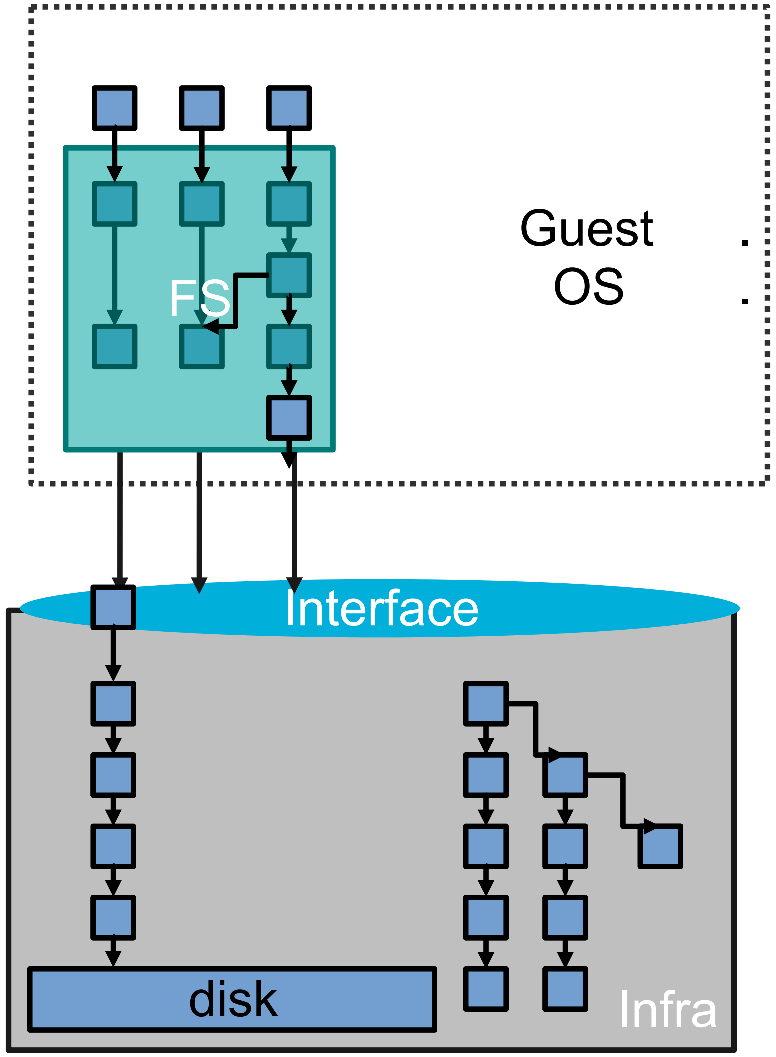
Low Level – VT:
Block Dev. Interface,
TAP interface,
etc.

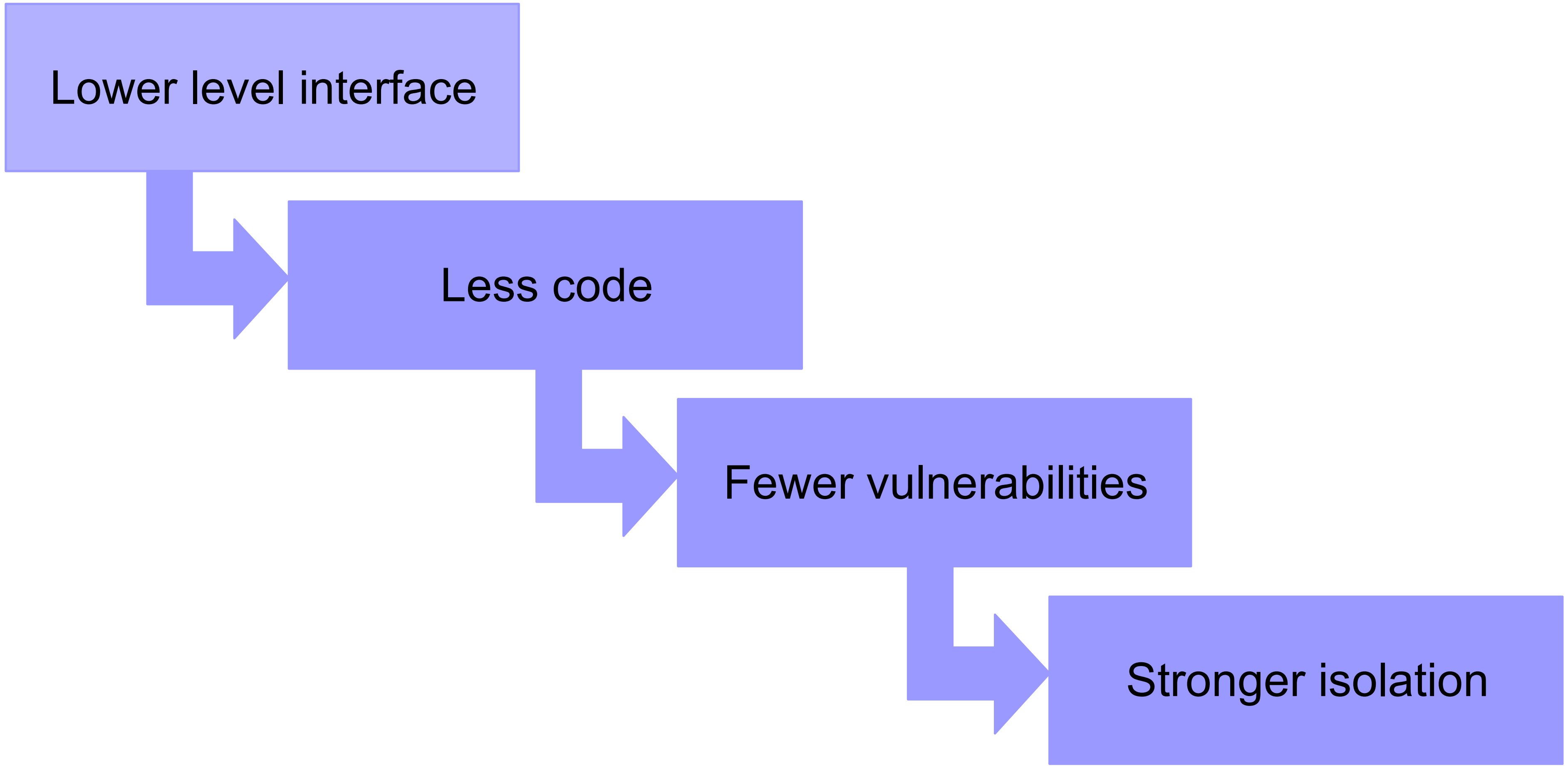
Containers



A LOT more exploitable code in the infrastructure!!!

VMs





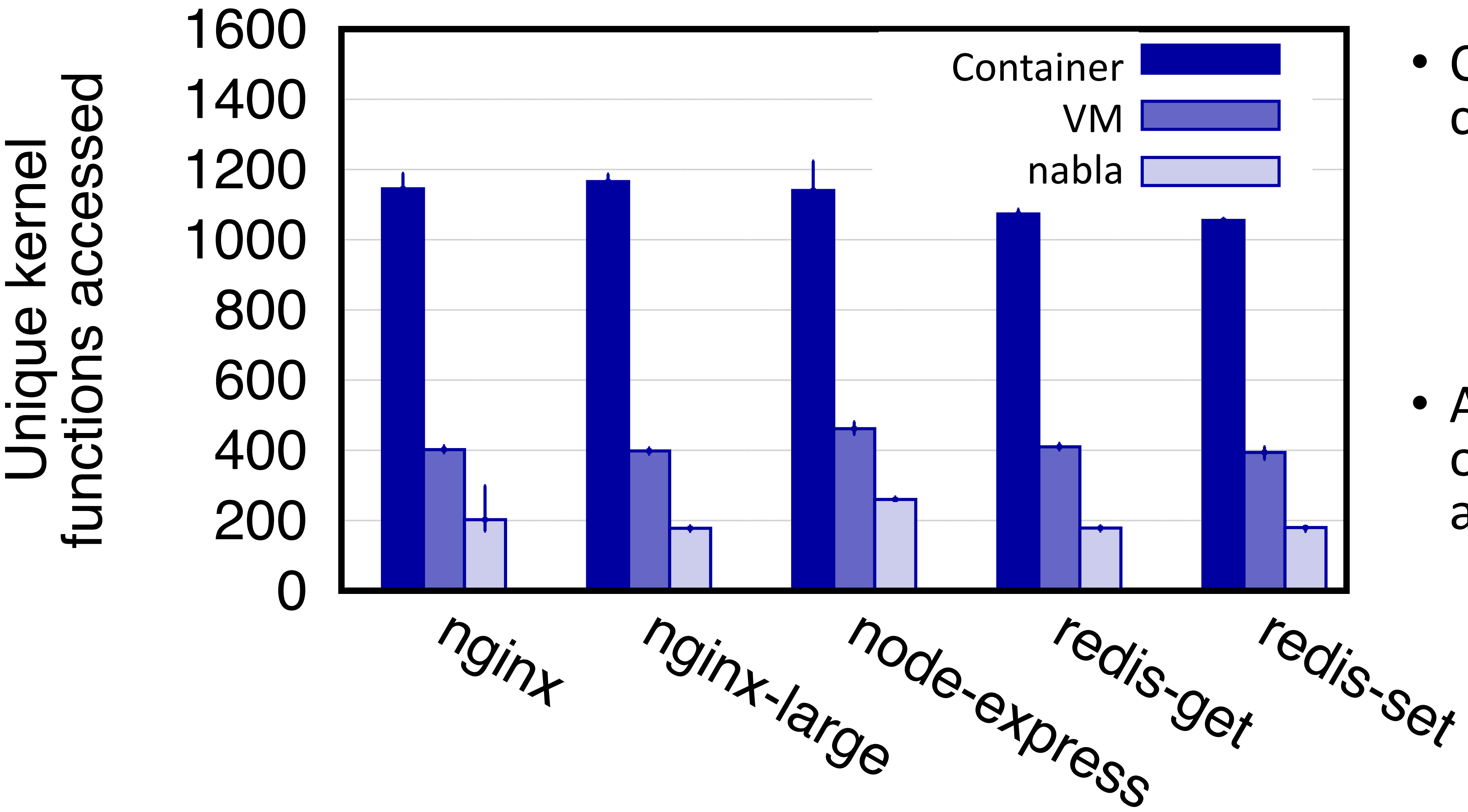
Lower level interface

Less code

Fewer vulnerabilities

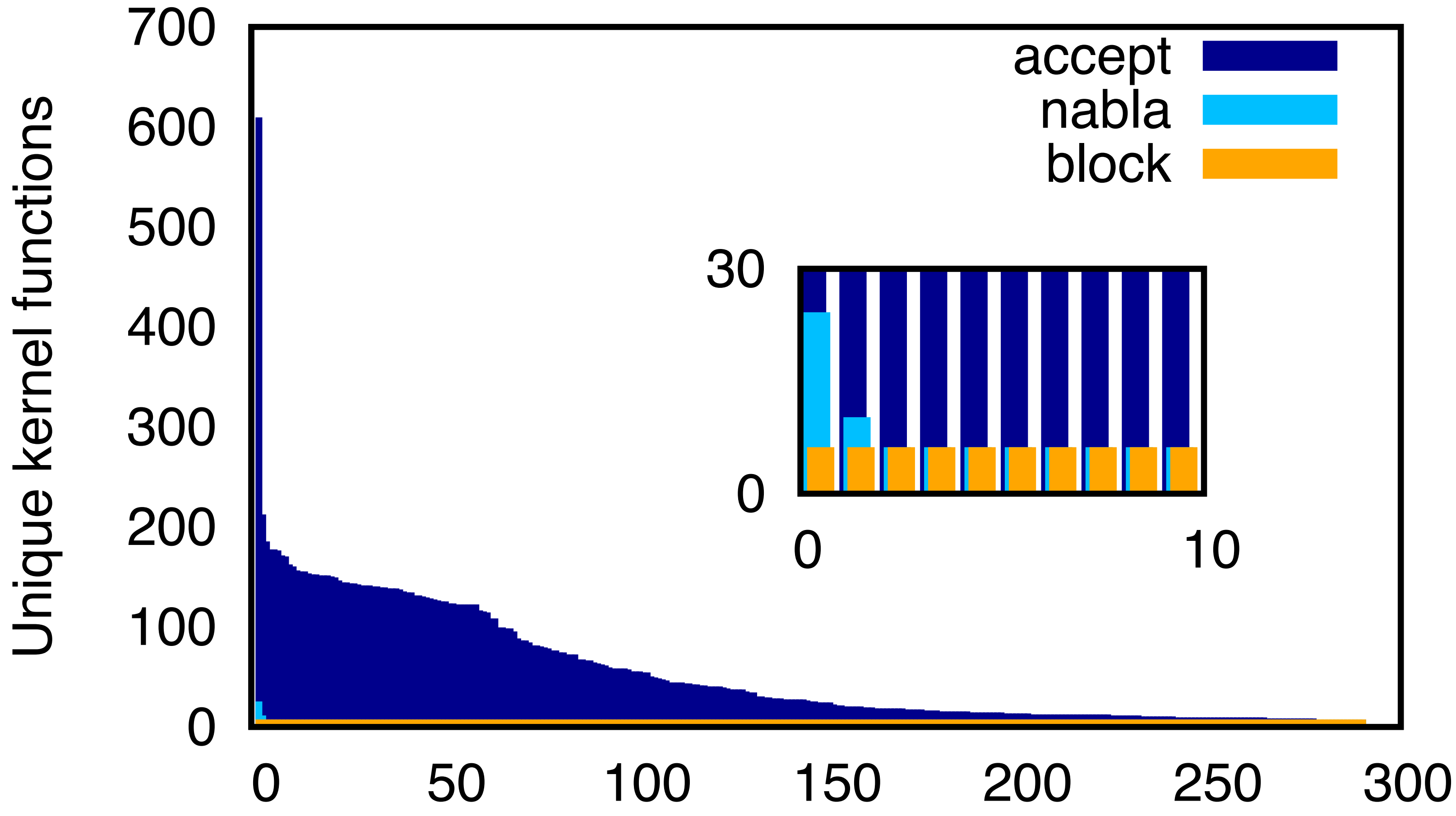
Stronger isolation

Kernel functions accessed by applications



- Compared to standard containers
  - 5-6x less kernel functions accessed
  - 8-14x fewer syscalls
- About half the number of kernel functions accessed as VMs!

Accessible kernel functions under Nabla policy



- Trinity kernel fuzz tester to try to access as much of kernel as possible

- Nabla policy reduces amount of accessible kernel functions by 98%

Unikernel isolation comes from the interface

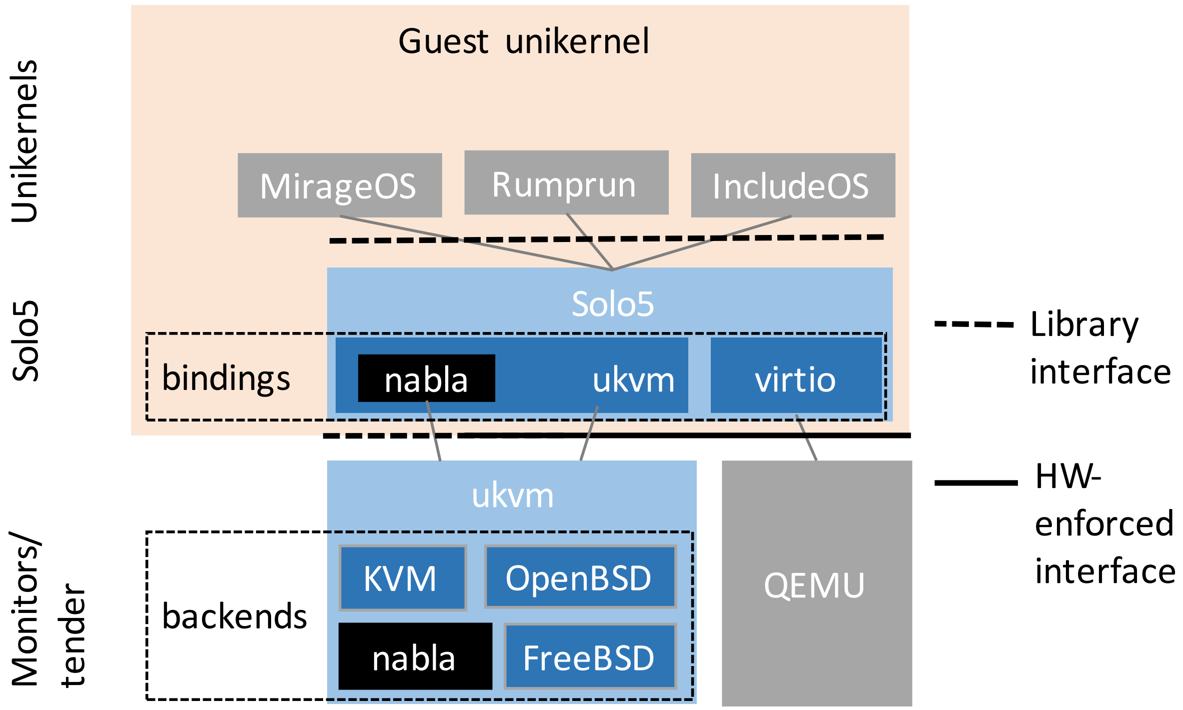
- Direct mapping between 10 hypercalls and system call/resource pairs
- 6 for I/O
  - Network: packet level
  - Storage: block level
- vs. >350 syscalls

Hypercall	System Call	Resource
`walltime`	`clock_gettime`	
`puts`	`write`	*stdout*
`poll`	`ppoll`	*net_fd*
`blkinfo`		
`blkwrite`	`pwrite64`	*blk_fd*
`blkread`	`pread64`	*blk_fd*
`netinfo`		
`netwrite`	`write`	*net_fd*
`netread`	`read`	*net_fd*
`halt`	`exit_group`	

SOCC

Implementation: nabla ∇

- Extended Solo5 unikernel ecosystem and ukvm
- Prototype supports:
  - MirageOS
  - IncludeOS
  - Rumprun
- https://github.com/solo5/solo5



Unikernels

Solo5

Monitors/ tender

Guest unikernel

MirageOS

Rumprun

IncludeOS

Solo5

bindings

nabla

ukvm

virtio

ukvm

backends

KVM

OpenBSD

nabla

FreeBSD

QEMU

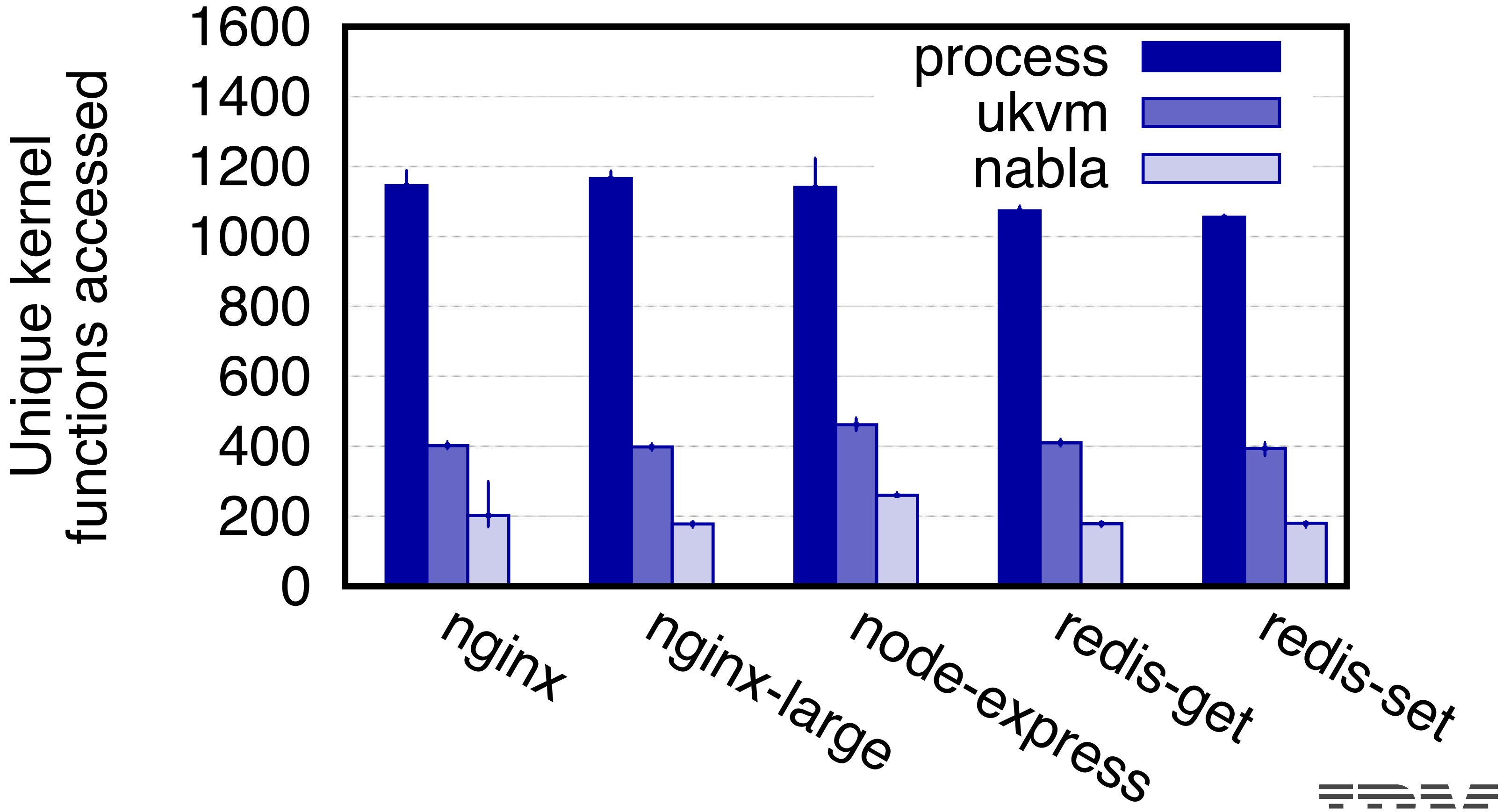
---- Library interface

— HW-enforced interface

Measuring isolation: common applications

- Code reachable through interface is a metric for attack surface
- Used kernel **`ftrace`**

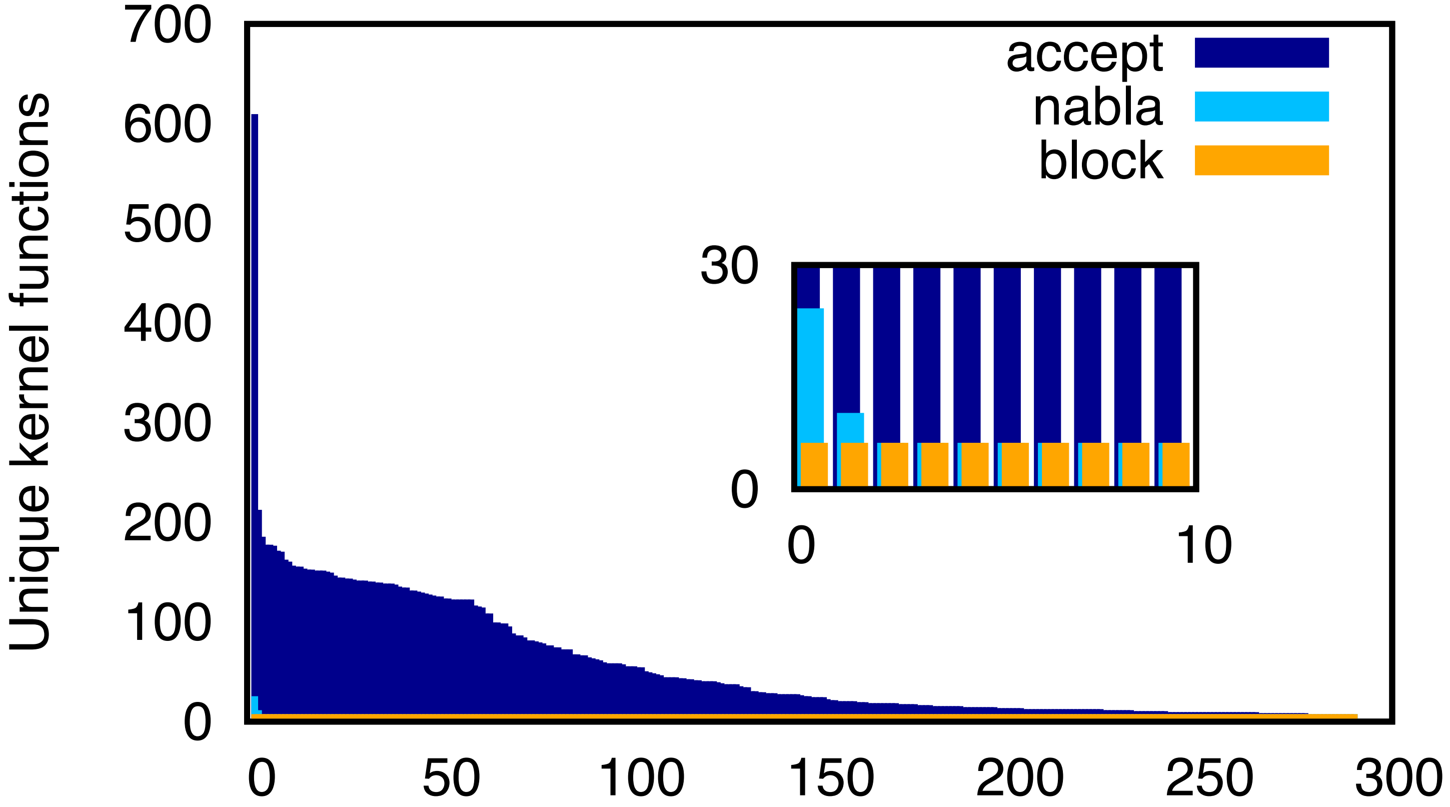
- Results:
  - Processes: 5-6x more
  - VMs: 2-3x more



Measuring isolation: fuzz testing

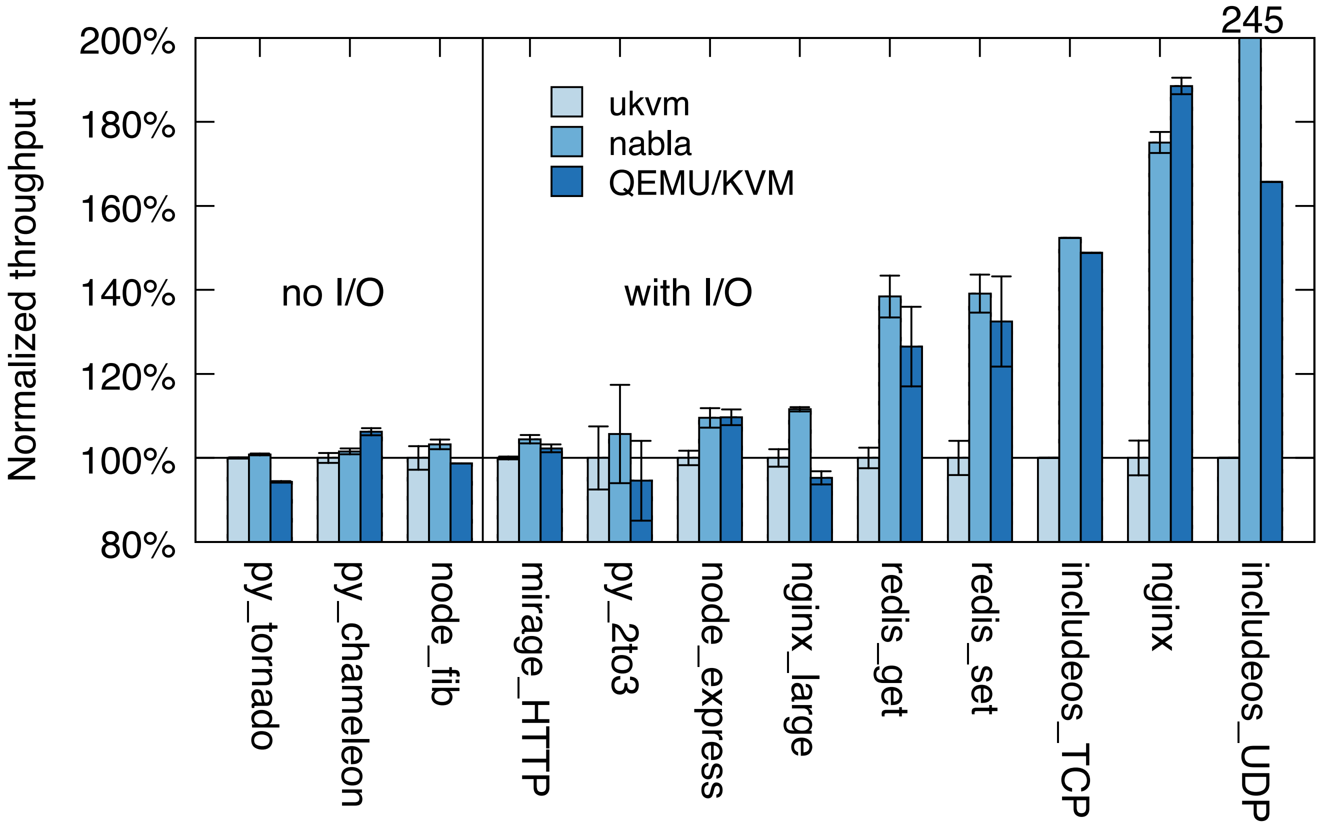
- Used kernel **`ftrace`**
- Used **`trinity`** system call fuzzer to try to access more of the kernel

- Results:
  - Nabla policy reduces by 98% over a "normal" process



Measuring performance: throughput

- Applications include:
  - Web servers
  - Python benchmarks
  - Redis
  - etc.

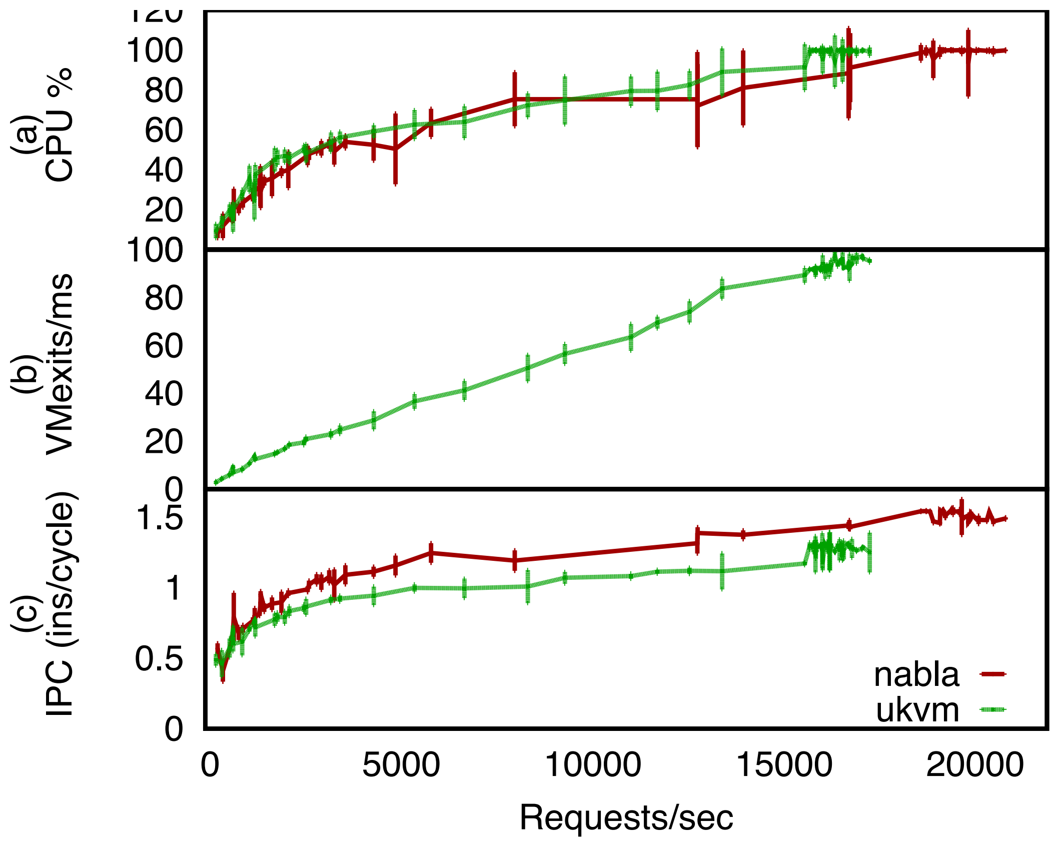


- Results:
  - 101%-245% higher throughput than ukvm

Measuring performance: CPU utilization

- `vmexits` have an effect on instructions per cycle

- Experiment with MirageOS web server
- Results:
  - 12% reduction in cpu utilization over ukvm

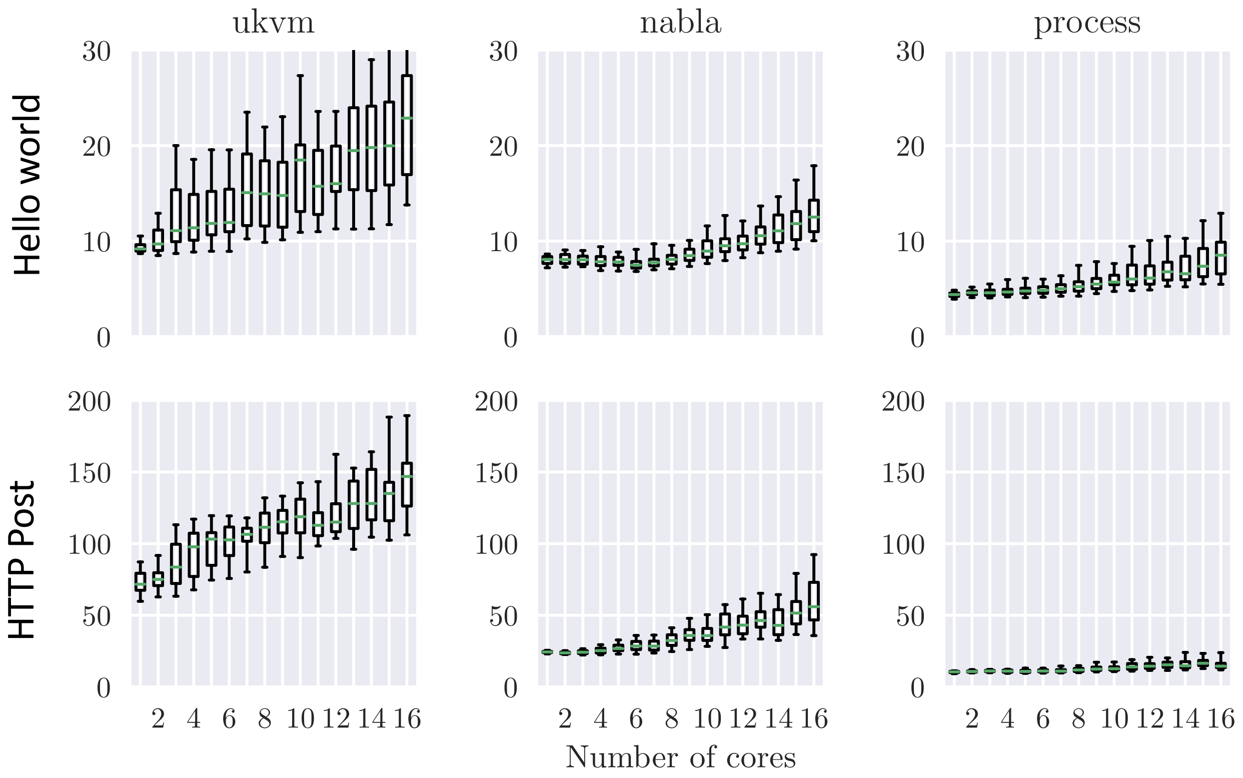


Measuring performance: startup time

- Startup time is important for serverless, NFV

- Results:
  - Ukvm has 30-370% higher latency than nabla

- Mostly due avoiding KVM overheads


ukvm
nabla
process
Hello world
HTTP Post
Number of cores