LightVM

Emmett Witchel

CS380L

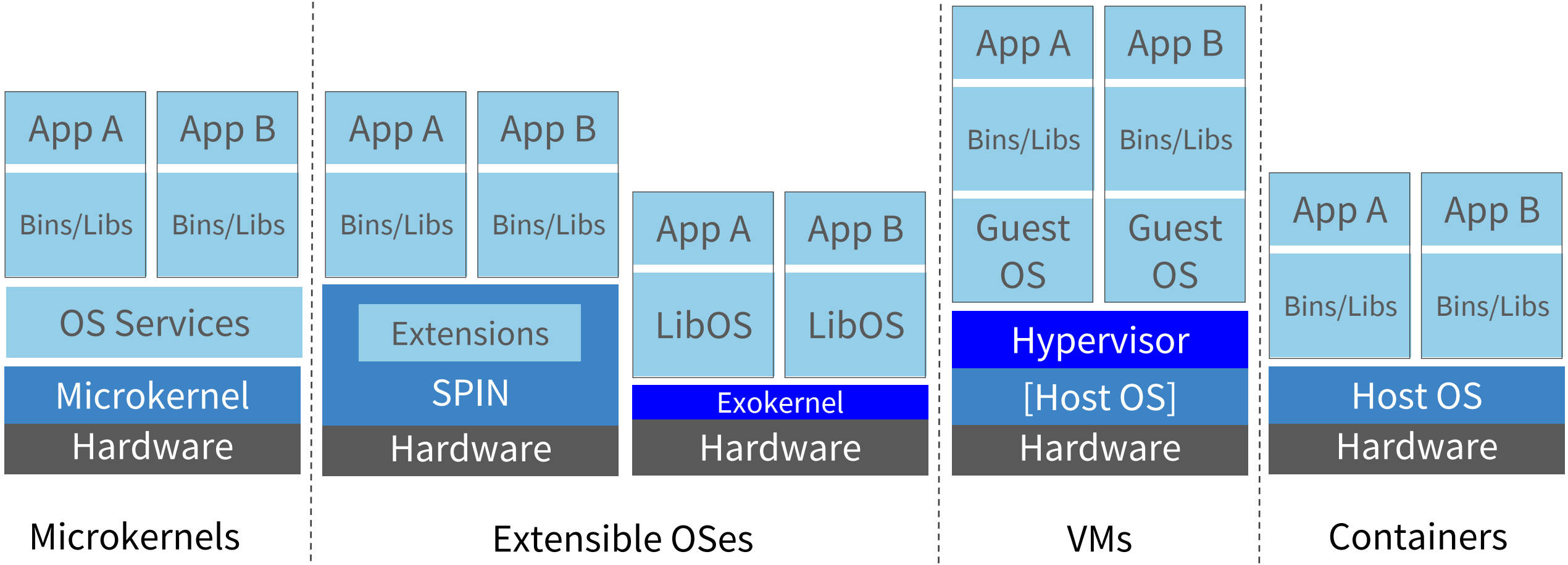
Faux quiz (pick 2, 5 min)

- What is the difference between an API and an ABI?
- Why can guest X-LibOSes turn off KPTI and seccomp?
- Discuss ways in which X-Containers trade inter- vs intra-container security
- What techniques do X-Containers use to reduce syscall overhead? How is that different from techniques in Xen and Arrakis?
- Compare/contrast container-level vs hypervisor-level virtualization.
- Compare and contrast interposition techniques in ESX, Xen, Arrakis, X-Container

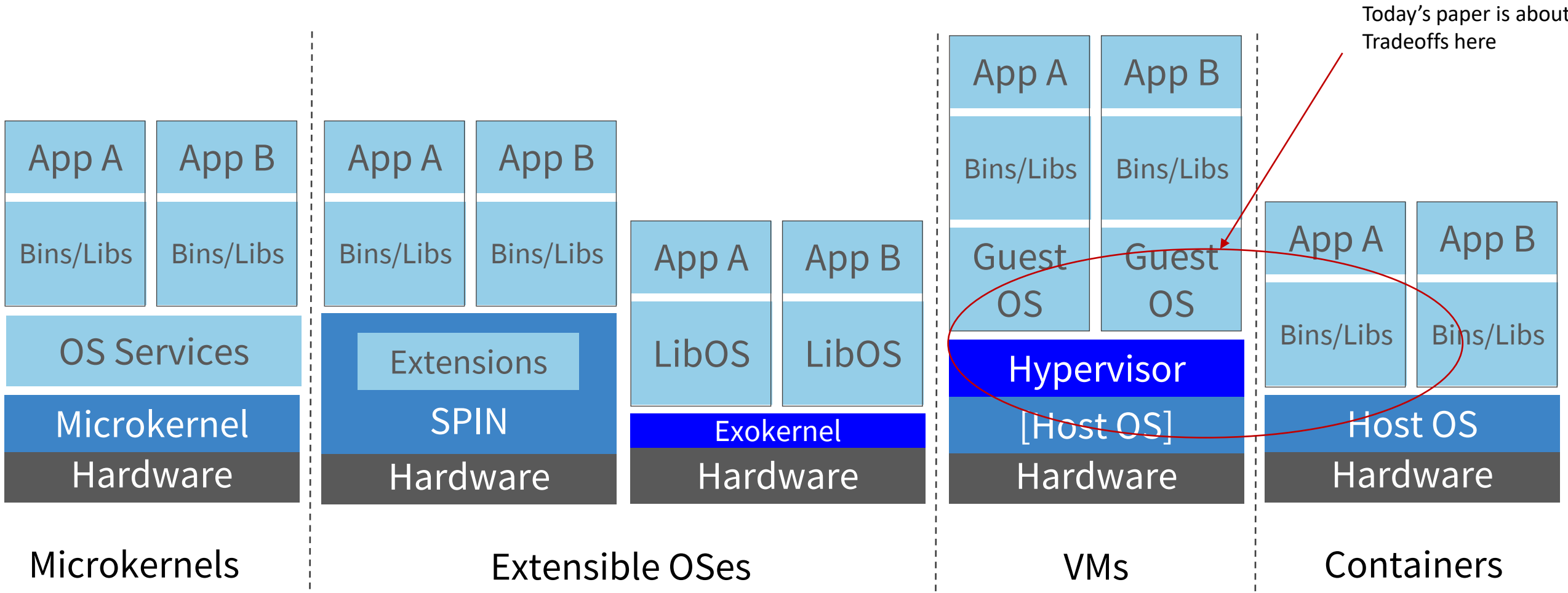
Acknowledgments

Some slides from LightVM talk at SOSP.
Some slides drawn from the X-Containers ASPLOS talk deck (thx to H. Weatherspoon)
Some slides drawn from Anjali, Caraza-Harter, Swift VEE ‘20 talk deck (thx!)

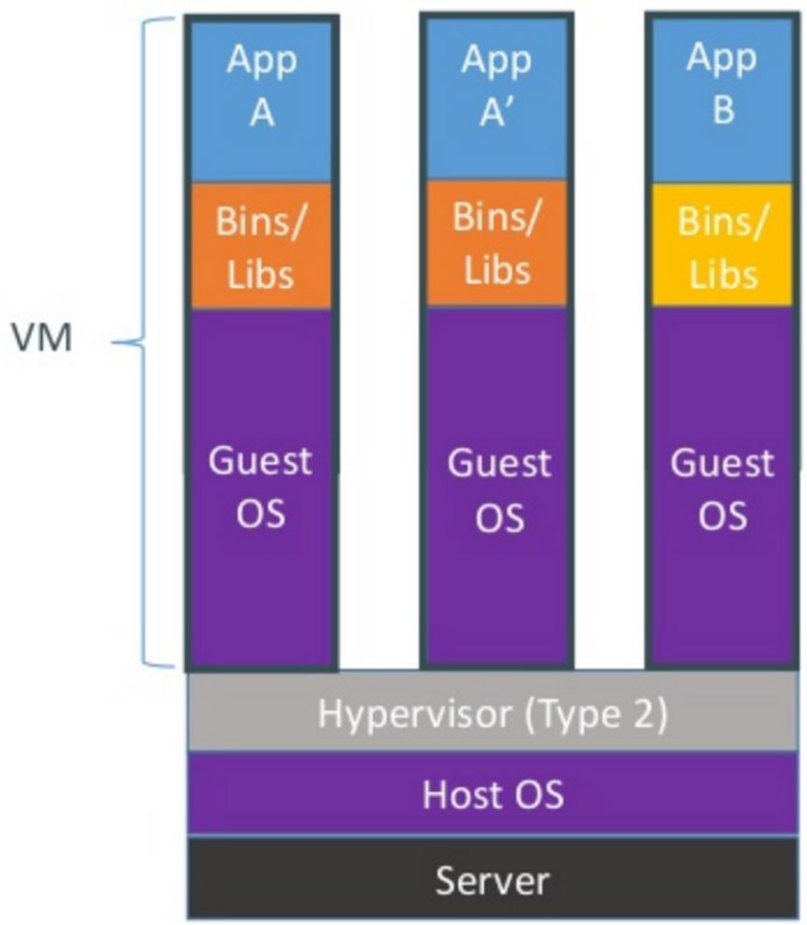
Box drawing Potpourri: OSes, VMs, Containers



Box drawing Potpourri: OSes, VMs, Containers

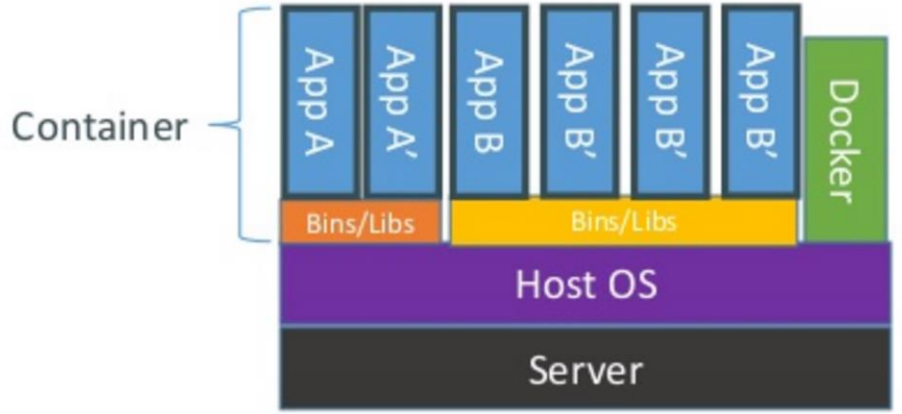


Containers vs VMs

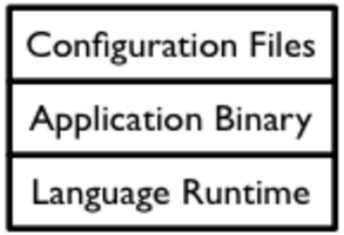


Containers are isolated, but share OS and, where appropriate, bins/libraries

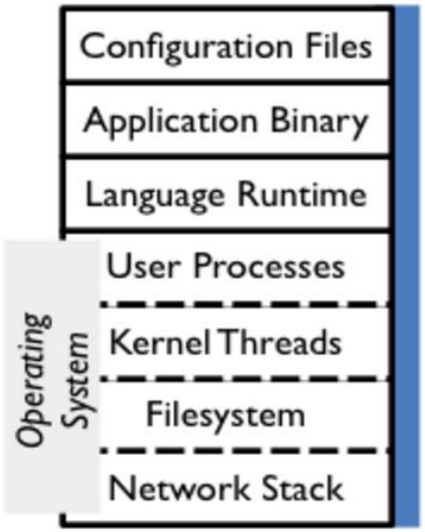
...result is significantly faster deployment, much less overhead, easier migration, faster restart



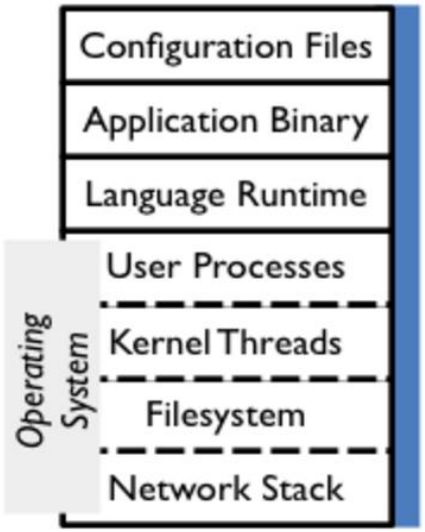
Application Logic



App + OS

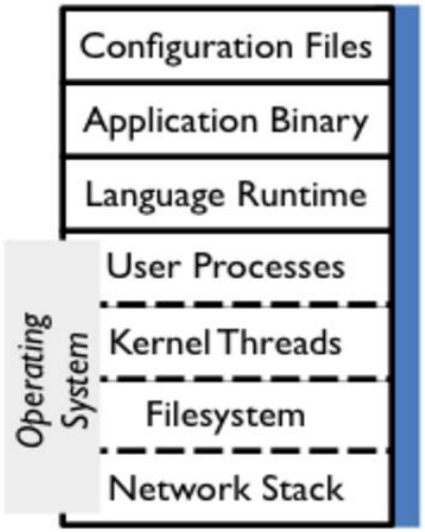

Configuration Files
Application Binary
Language Runtime
User Processes
Kernel Threads
Filesystem
Network Stack
Operating System

App + OS



What should I do if I have multiple applications and I want them to *share* the hardware, but not interfere with each other?

App + OS

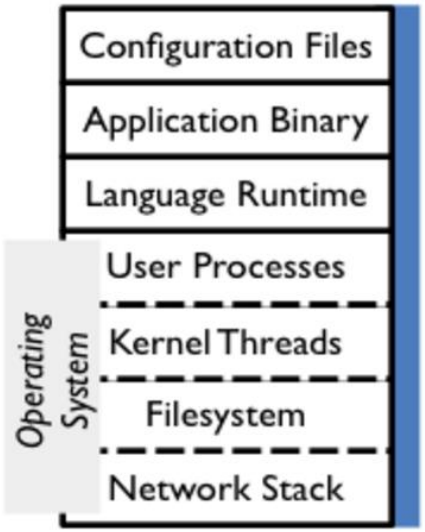


What should I do if I have multiple applications and I want them to *share* the hardware, but not interfere with each other?

Possible Answers:

A. Don't do it. Get more machines.
B. Let the OS arbitrate/enforce: that's it's job!
C. Get a hypervisor and run them in separate VMs!
D. All of the above

App + OS



What should I do if I have multiple applications and I want them to *share* the hardware, but not interfere with each other?

Possible Answers:

A. Don't do it. Get more machines.
B. Let the OS arbitrate/enforce: that's it's job!
C. Get a hypervisor and run them in separate VMs!
D. All of the above

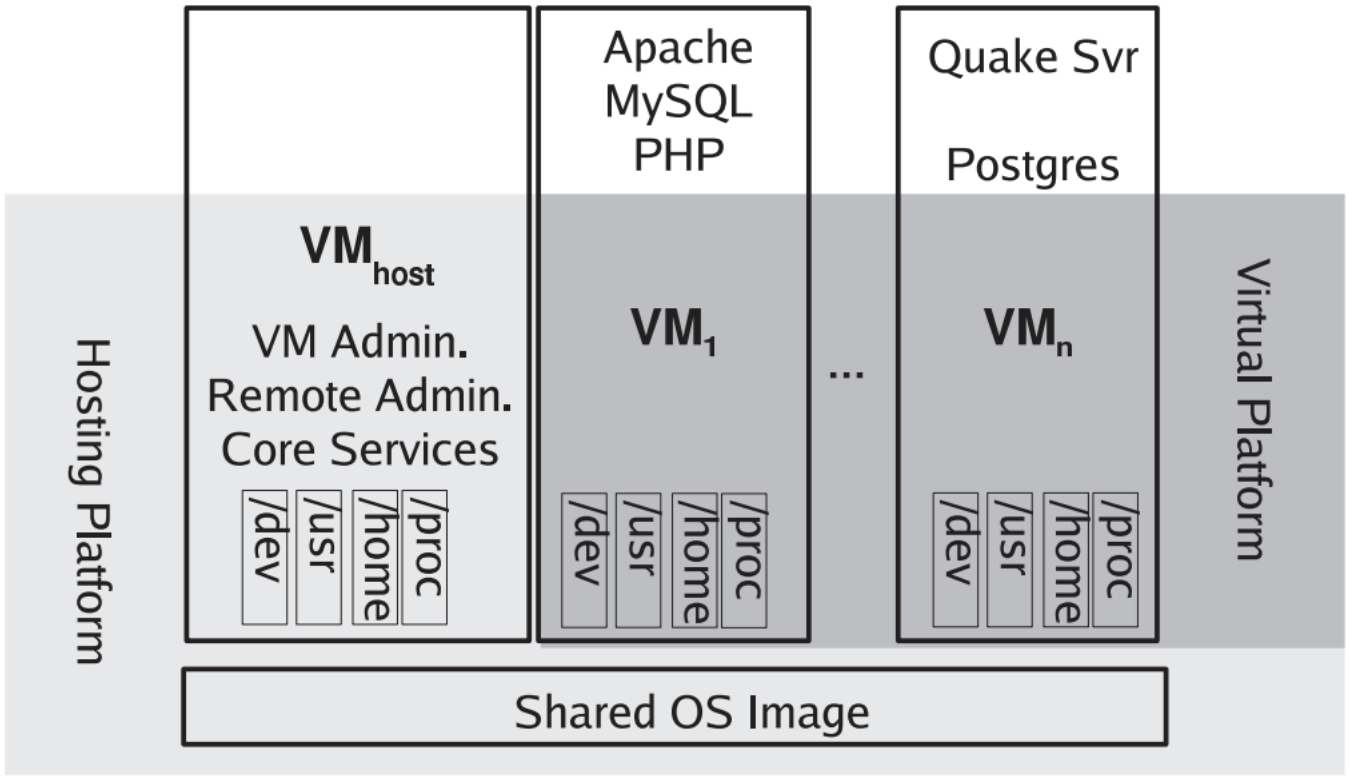
Containers motivation:

- B really should be the answer, but the process is the unit of isolation and resource management in Oses
- Many applications use multiple processes
- Many applications use/share state through OS (FS, Net, etc) that gets complicated to arbitrate

Solution: introduce another level of resource/isolation management

Containers

- Think of as lightweight VM
- Separate Process space, network interface
- Setuid/root access possible
- Share kernel with host → no I/O emulation, VM overheads
- chroot, cgroups



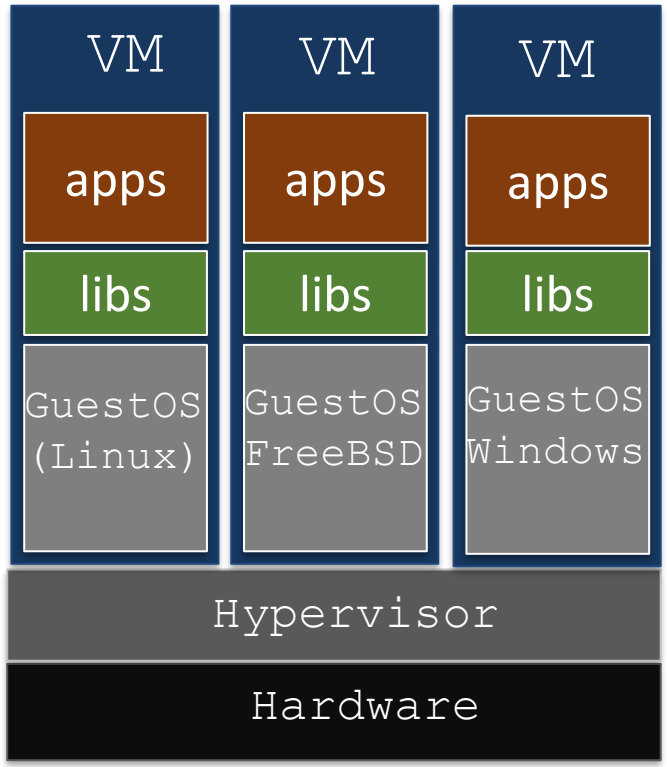
Containers isolated by OS

- Containers provides the ability to package and run an application in a loosely isolated environment called a container. (docs.docker.com)
  - The isolation and security allows you to run many containers simultaneously on a given host.
- OS capabilities enable docker
  - Process ID namespaces (e.g., each container has an init process pid==1)
  - Network communication (e.g., iptables)
  - Control group (cgroups)
  - Layered/union file system (e.g., AuFS)
- Tradeoffs
  - More secure than user-level packaging solutions like conda
  - OS struggles to achieve performance isolation
  - Container isolation is inferior to virtual machines

Isolating workloads

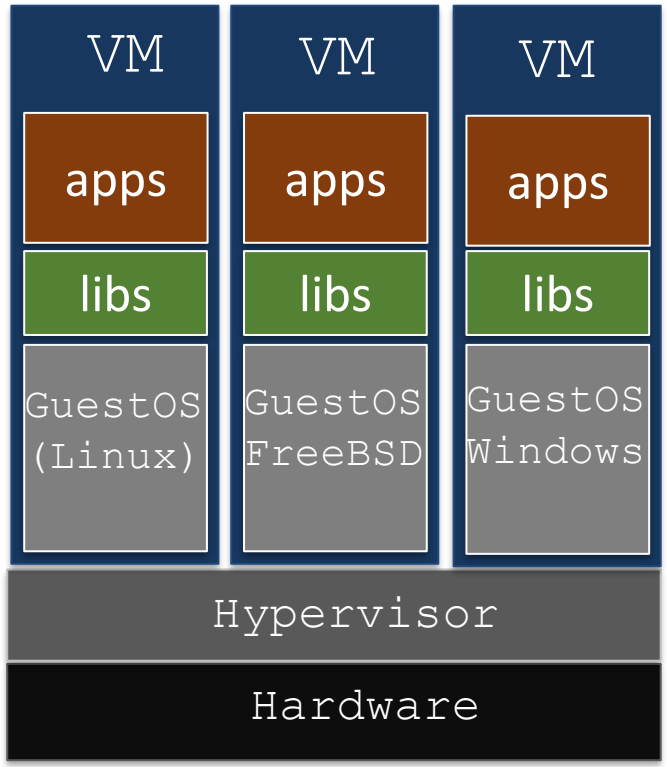
Isolating workloads

Virtualization



Isolating workloads

Virtualization

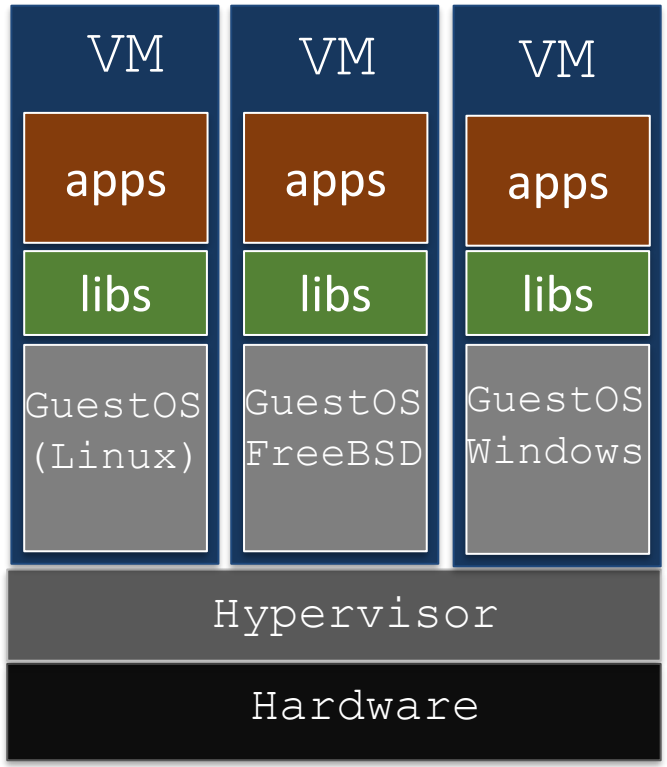


✅ Strong isolation

❌ Heavy weight

Isolating workloads

Virtualization

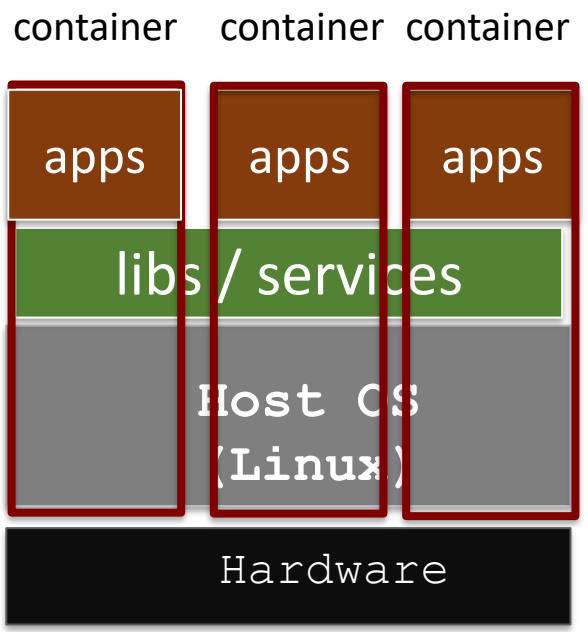


✅ Strong isolation

❌ Heavy weight

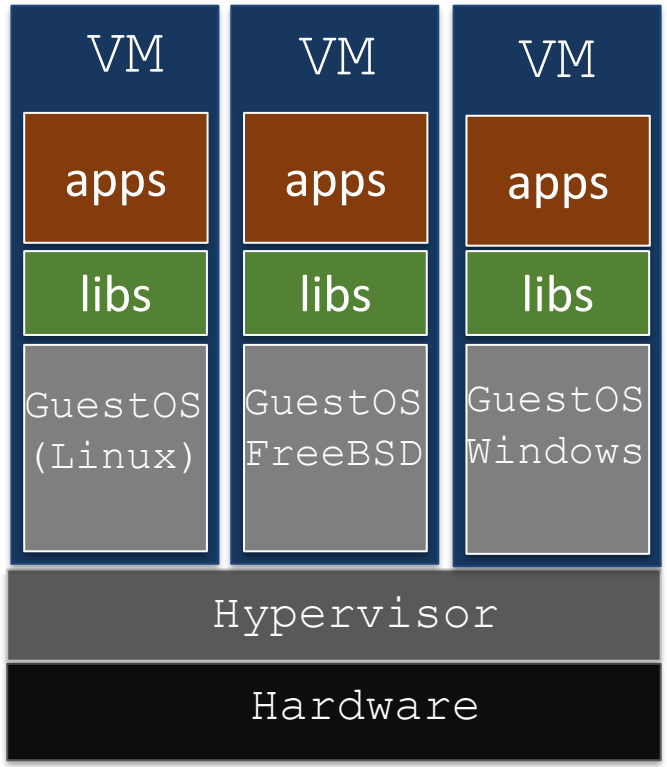
vs.

Containers



Isolating workloads

Virtualization

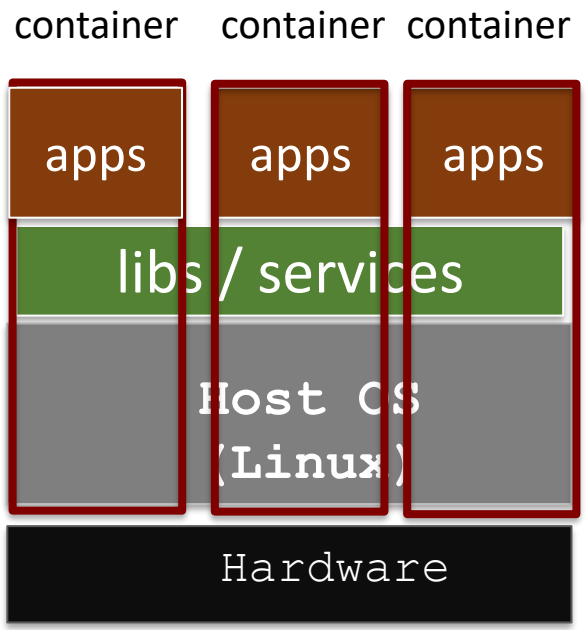


✅ Strong isolation

❌ Heavy weight

vs.

Containers

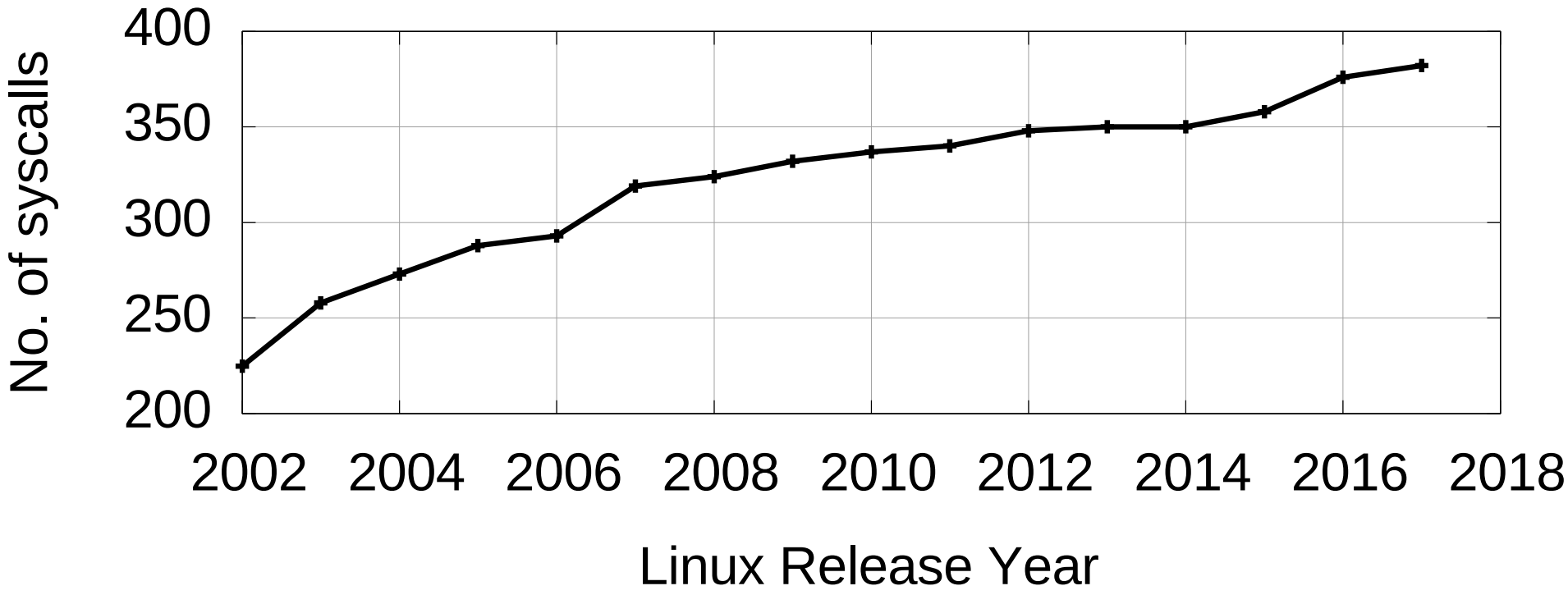


✅ Lightweight

❌ Iffy isolation

Isolating workloads

Virtualization **vs.** Containers



✅ Strong isolation

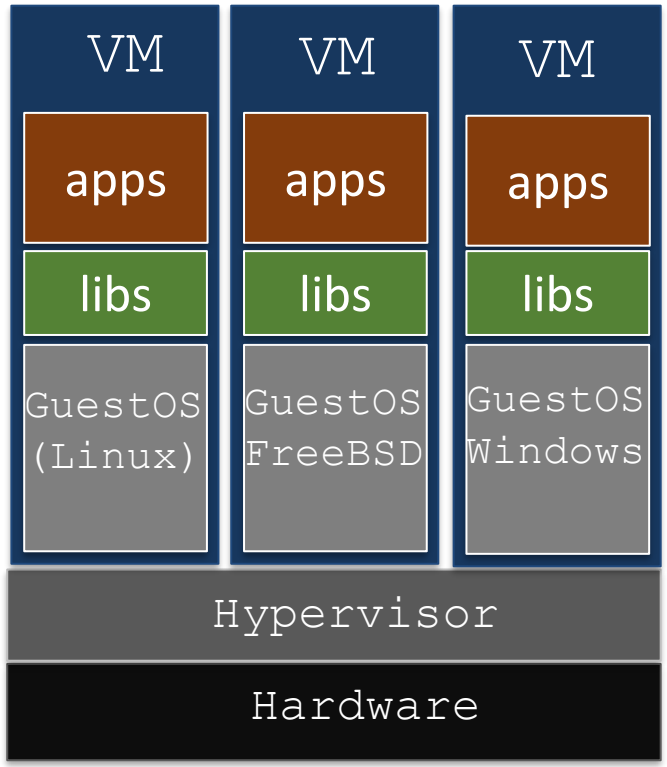
❌ Heavy weight

✅ Lightweight

❌ Iffy isolation

Isolating workloads

Virtualization

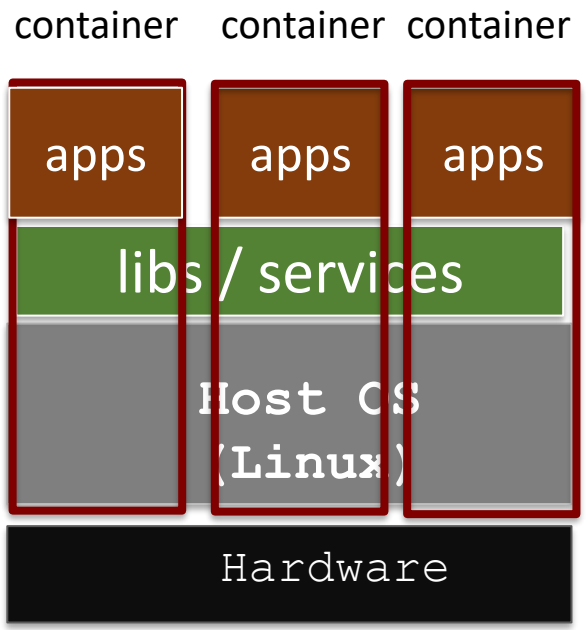


✅ Strong isolation

❌ Heavy weight

vs.

Containers

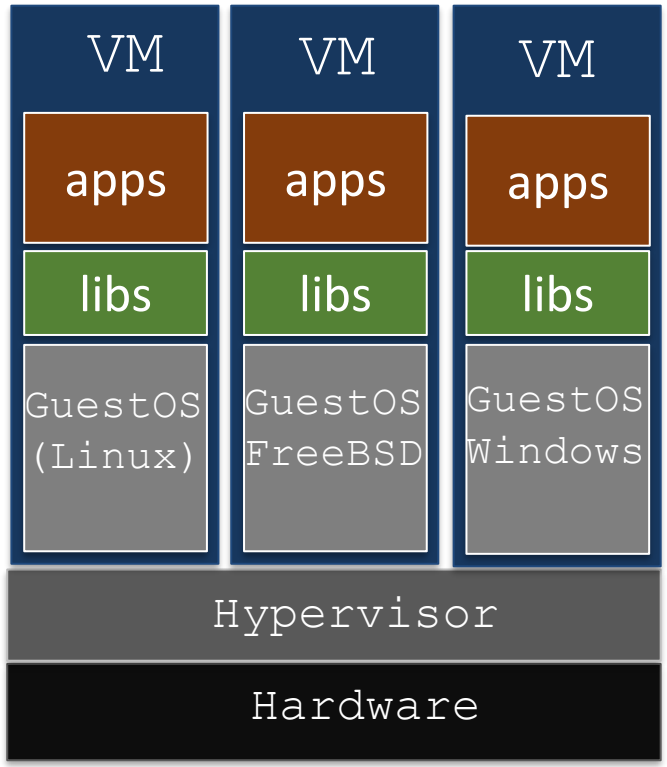


✅ Lightweight

❌ Iffy isolation

Pick Your Poison

Virtualization

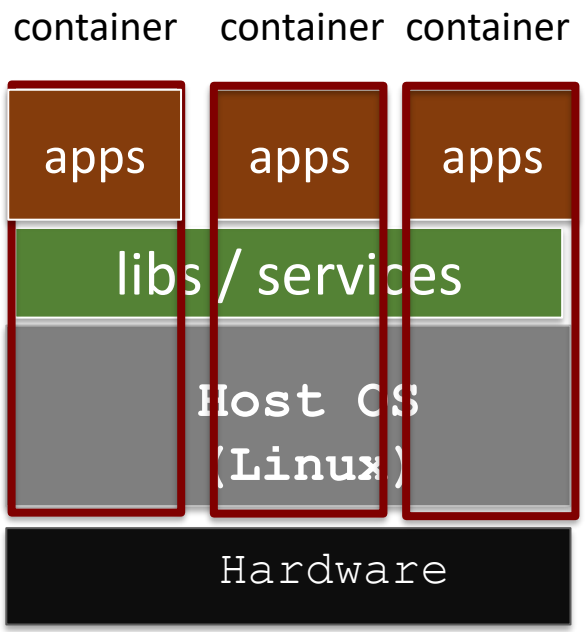


✅ Strong isolation

❌ Heavy weight

vs.

Containers

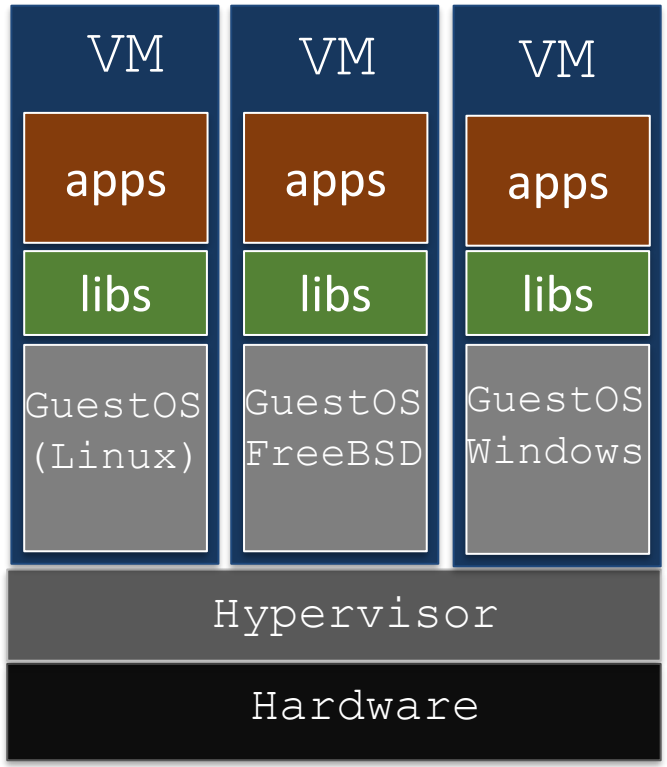


✅ Lightweight

❌ Iffy isolation

Pick Your Poison

Virtualization



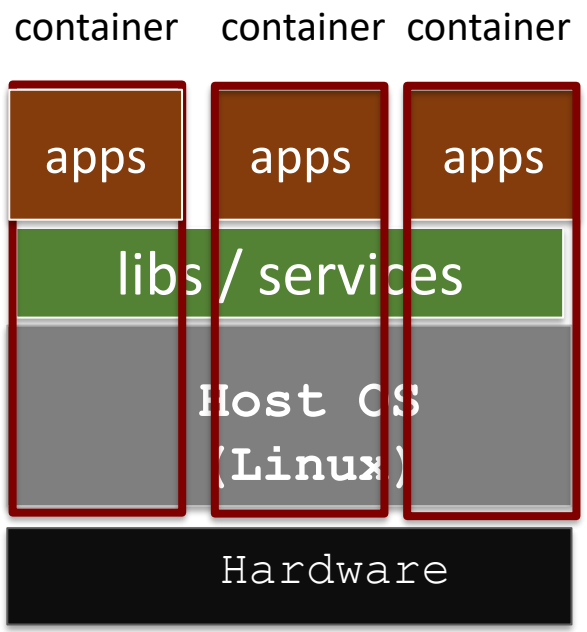
✅ Strong isolation

❌ Heavy weight

vs.

?

Containers

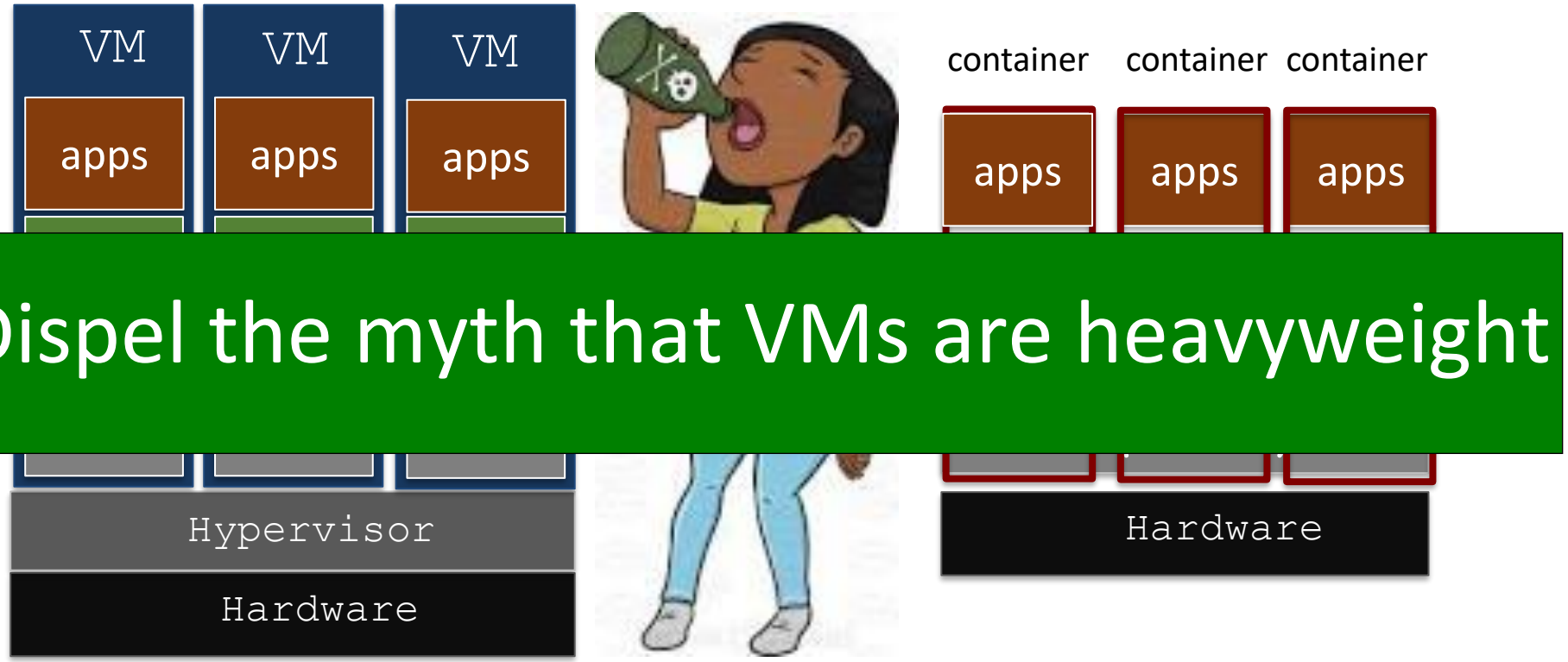


✅ Lightweight

❌ Iffy isolation

Pick Your Poison

Virtualization **vs.** Containers



Dispel the myth that VMs are heavyweight

✅ Strong isolation

❌ Heavy weight

?

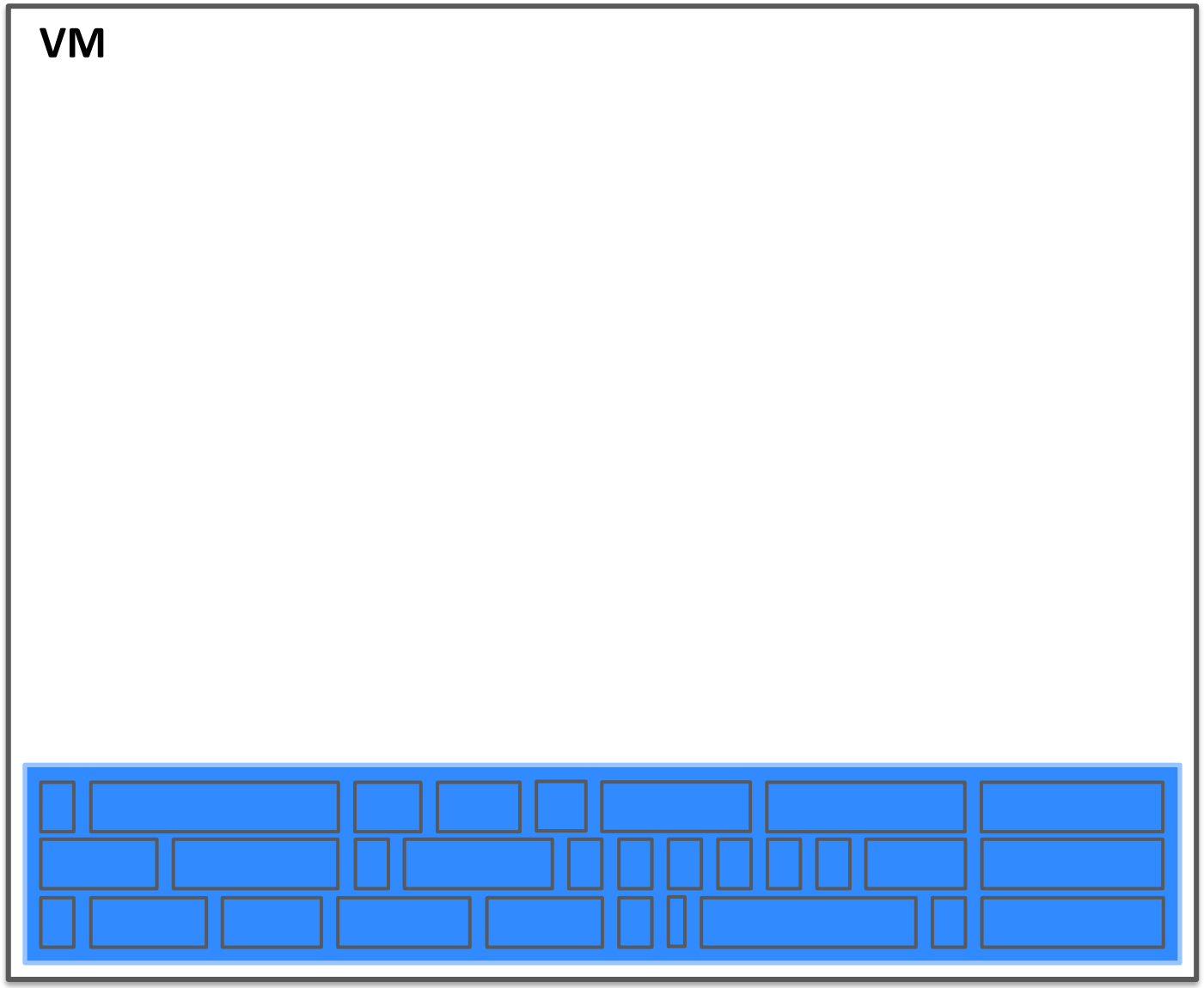
✅ Lightweight

❌ Iffy isolation

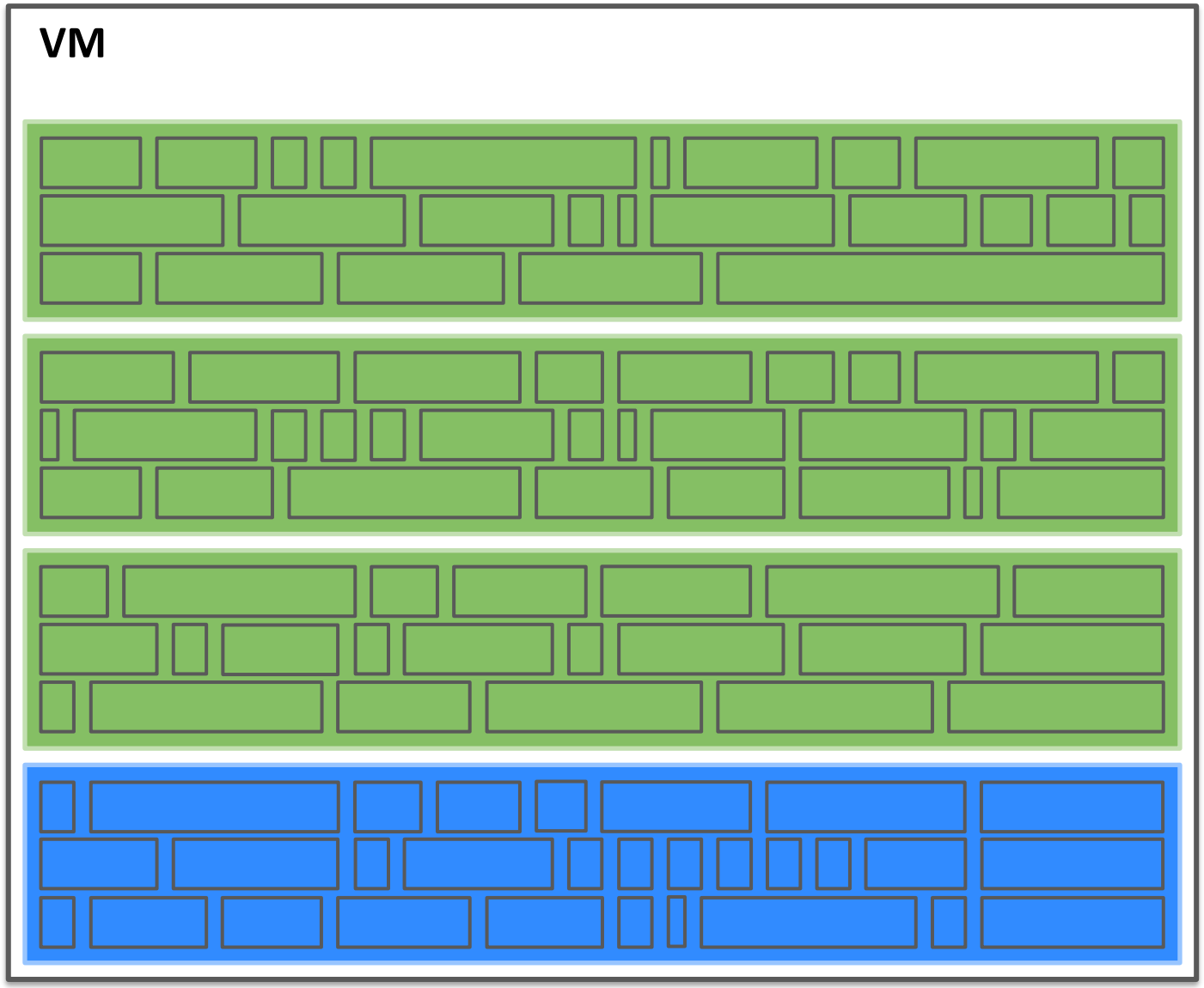
LightVM
VMs as fast and efficient as containers

- Fast instantiation, destruction and migration
  - **10s of milliseconds** or less
- Low per-instance memory footprint
  - **10s of MBs** or less
- High density
  - **1-10K** guests on a single host

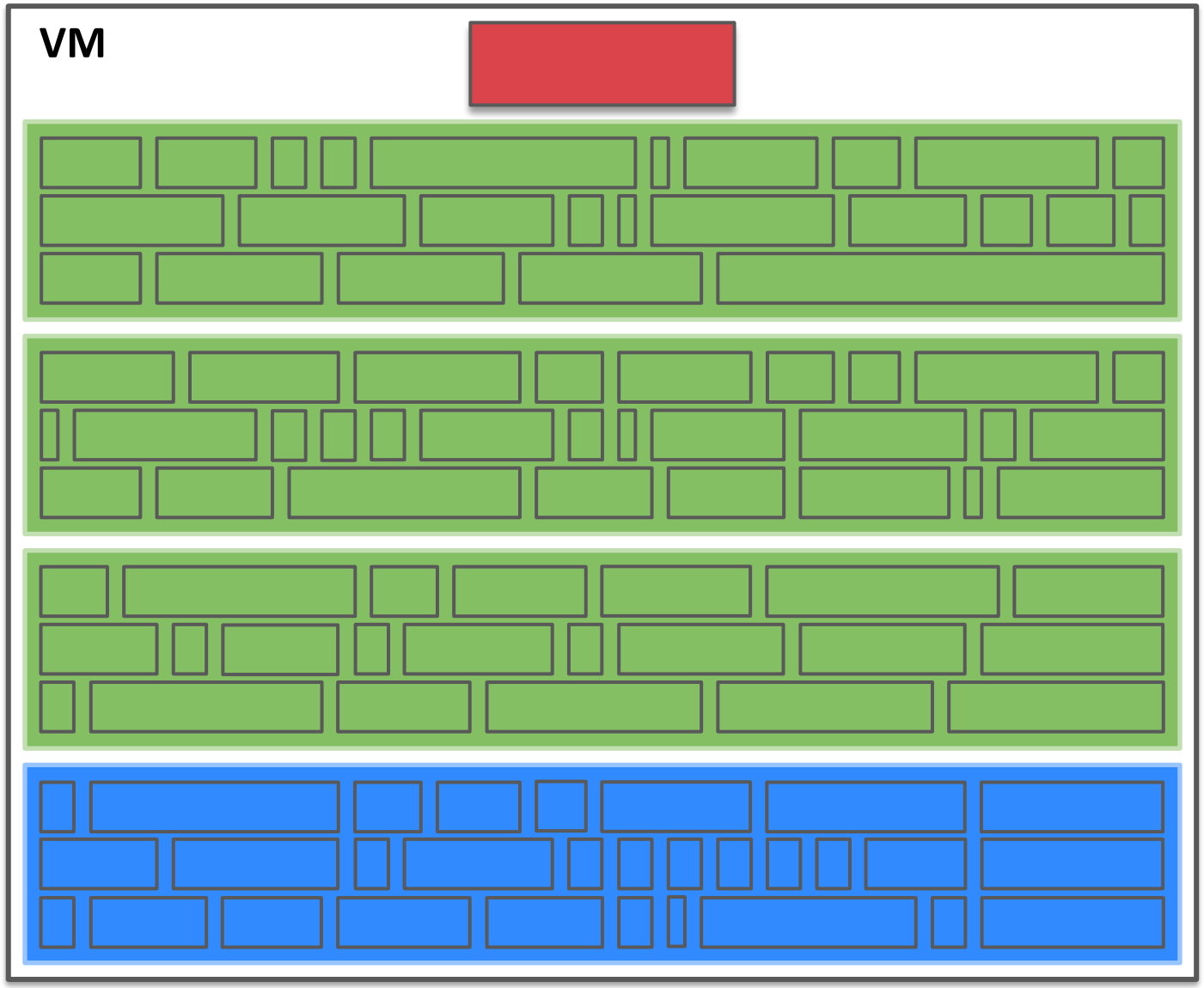
Standard VM: Application on Top of Distro



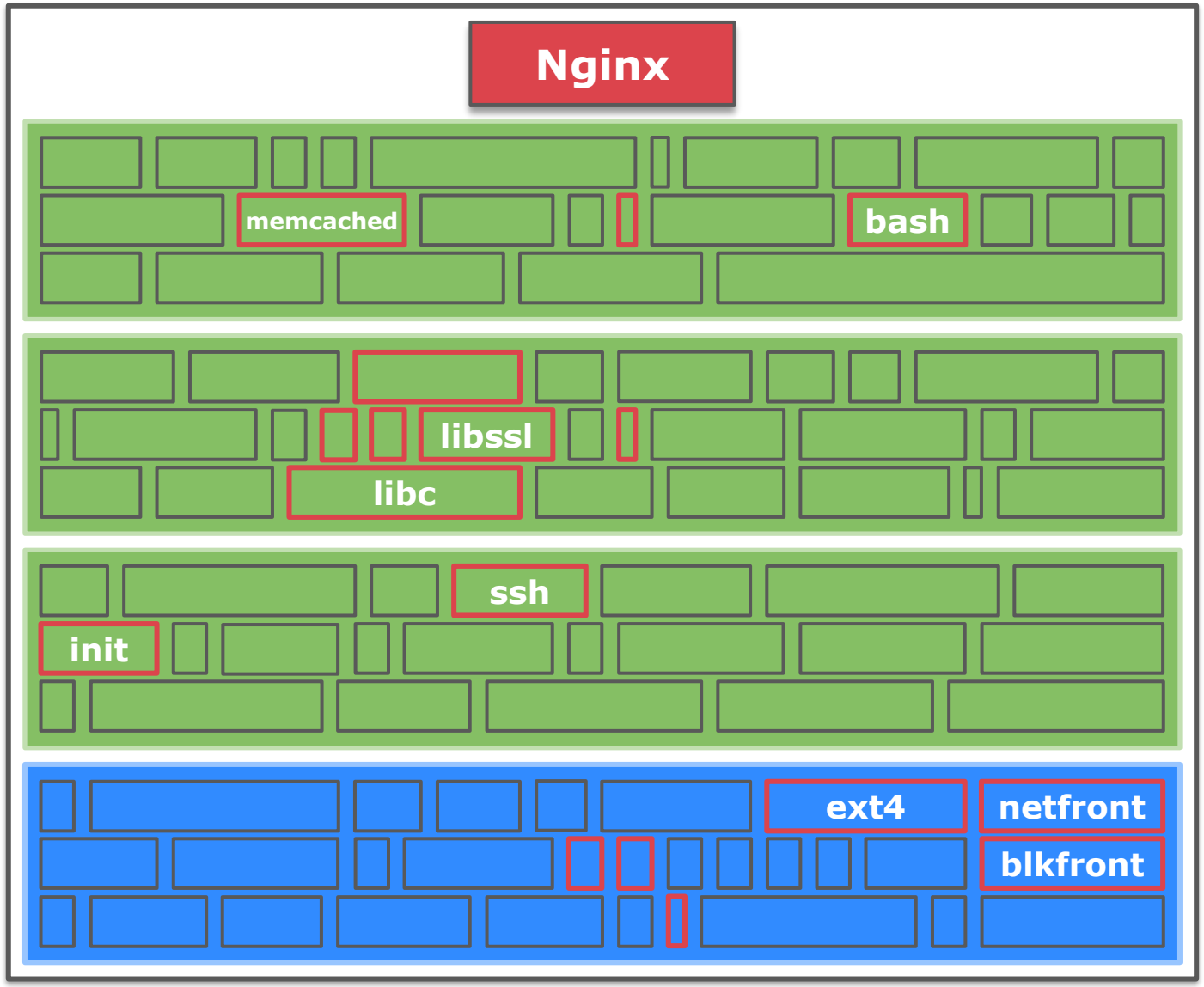
Standard VM: Application on Top of Distro



Standard VM: Application on Top of Distro



Most of the VM not Used…



User Application

3rd Party Applications

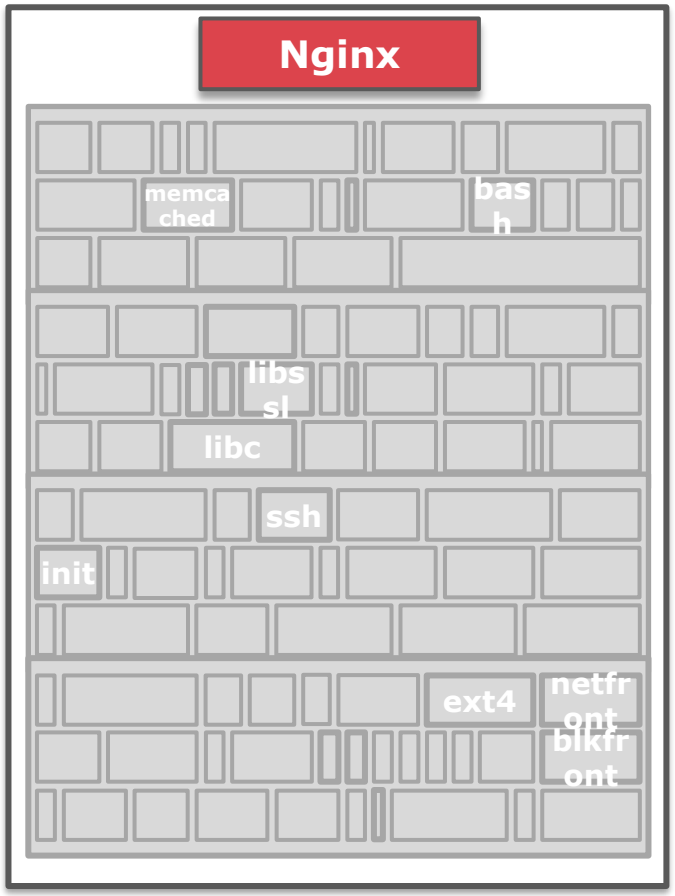
Libraries

Services

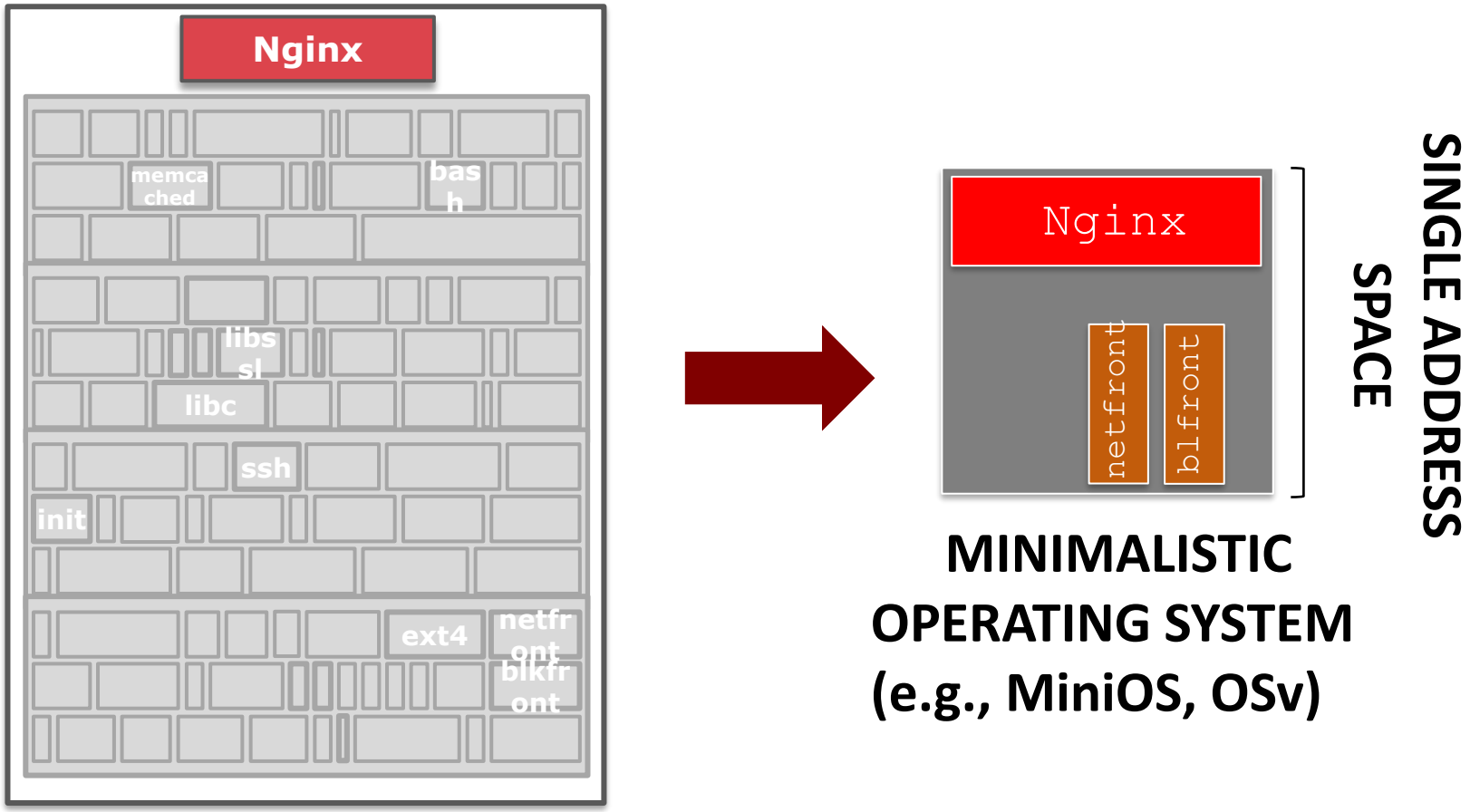
Kernel

Unikernel : single app+minimalistic OS

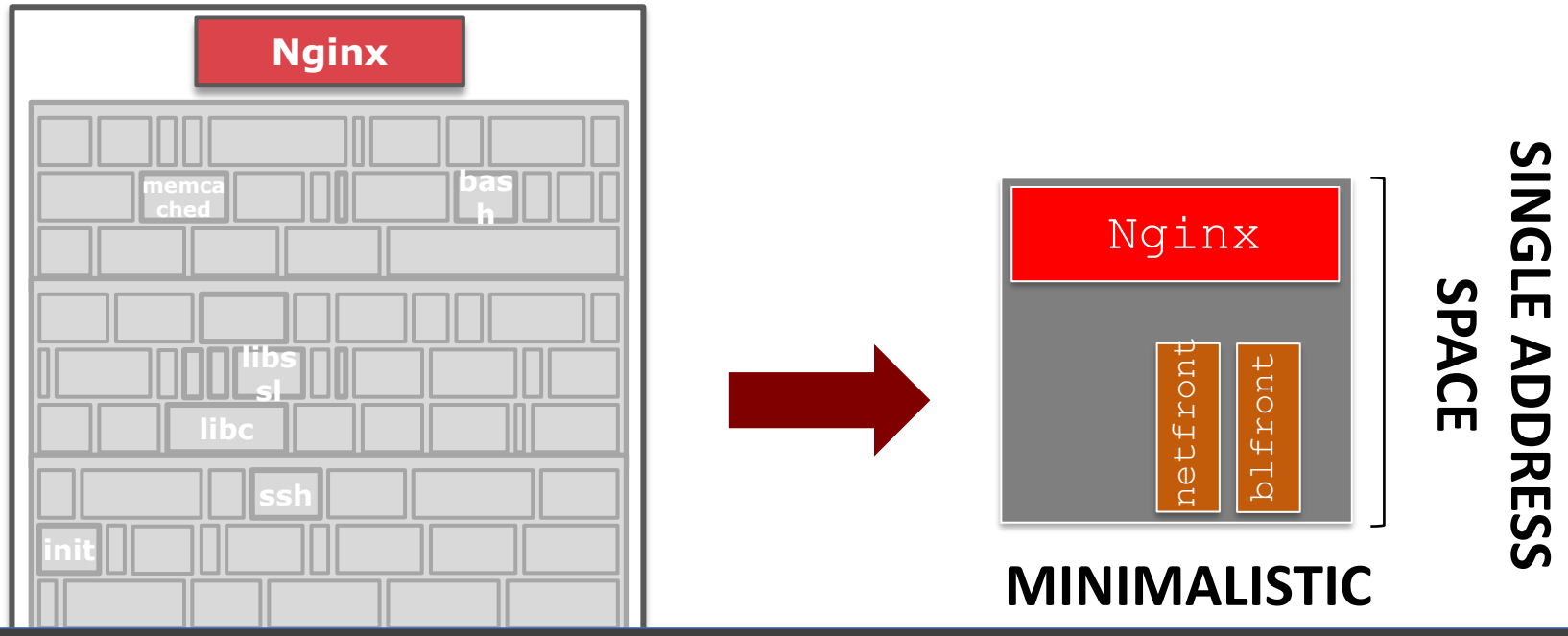
Unikernel : single app+minimalistic OS



Unikernel : single app+minimalistic OS



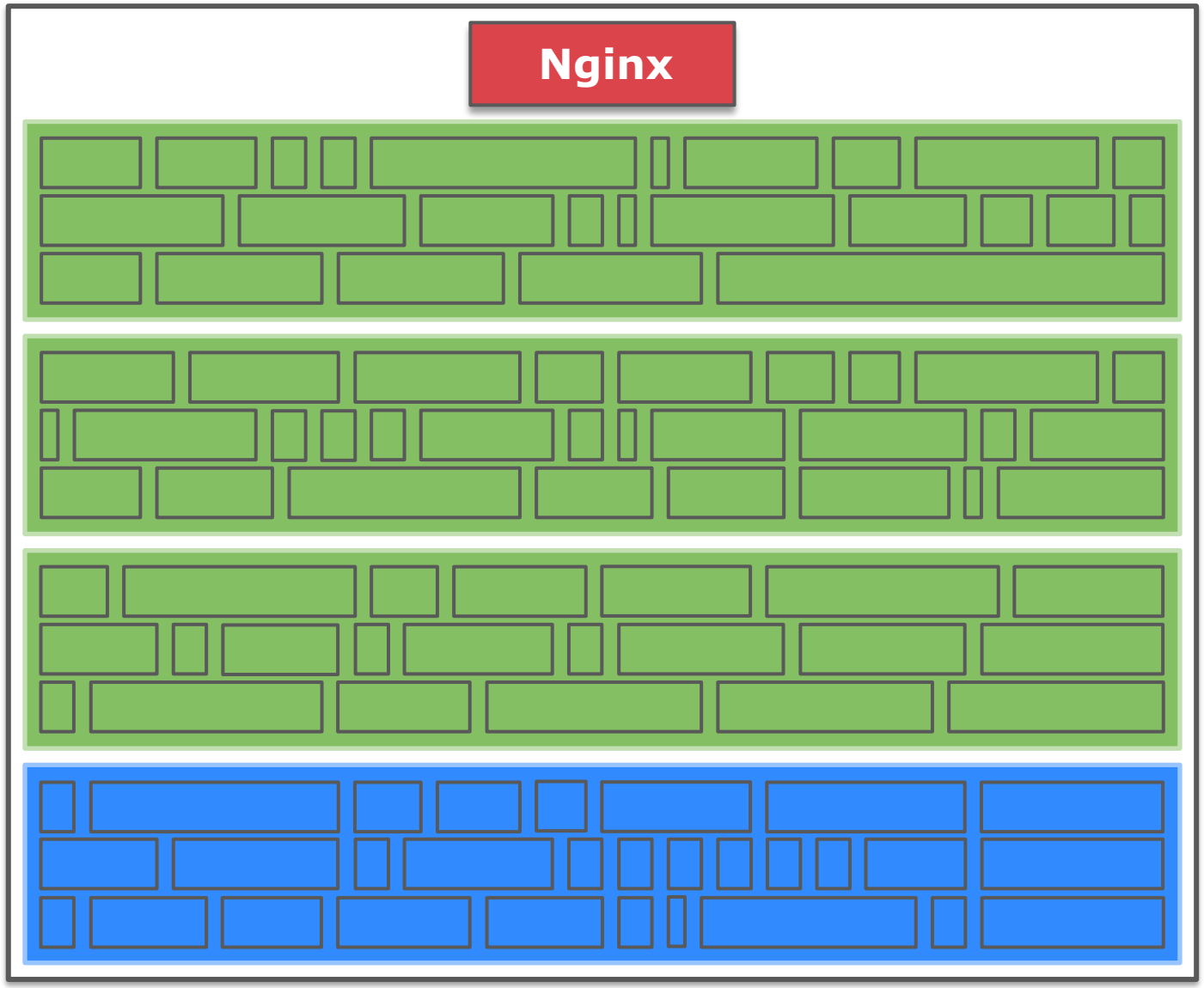
Unikernel : single app+minimalistic OS



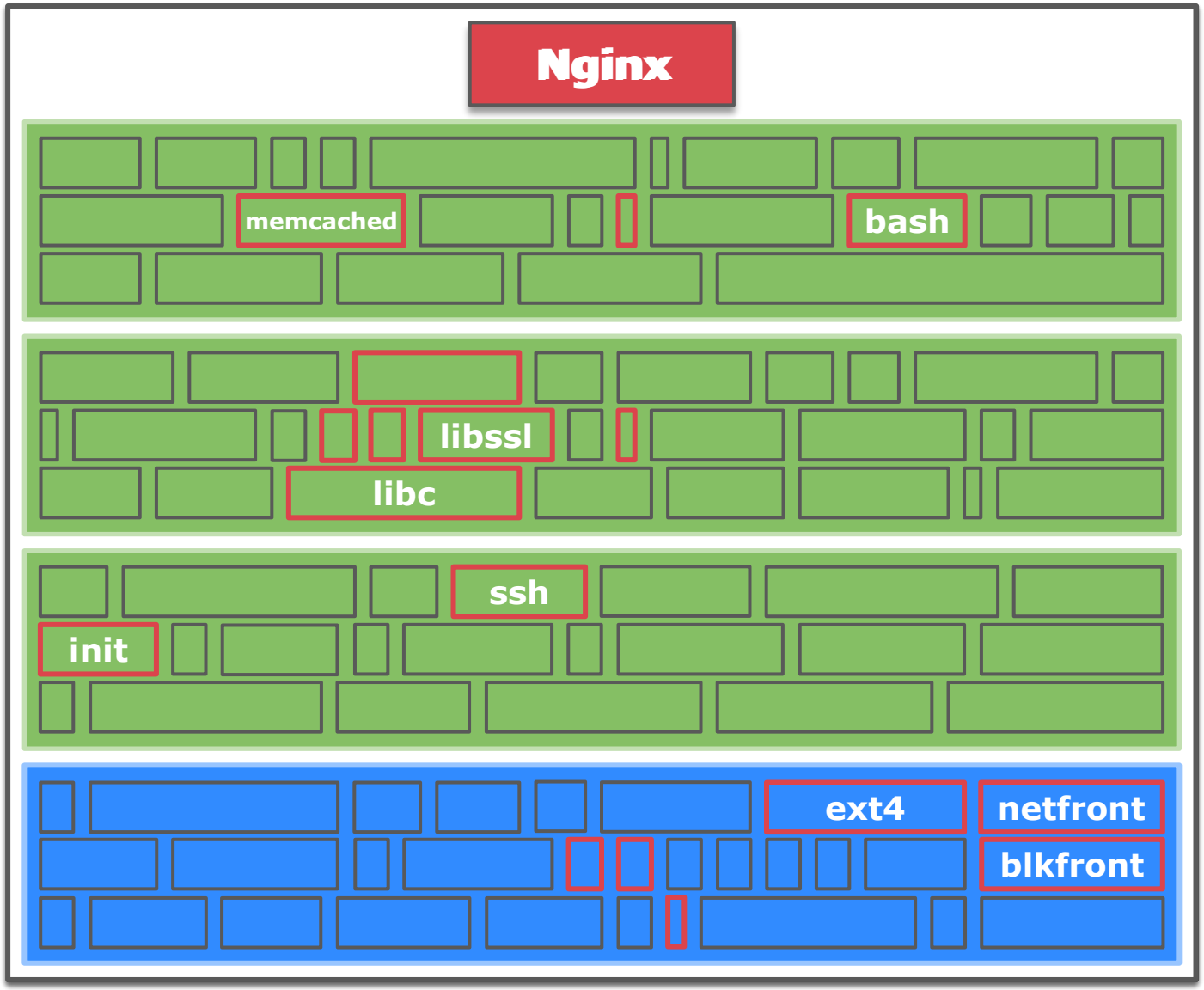
Unikernels are lightweight:

- **Daytime -** 480KB disk, 3.4MB RAM
- **Minipython -** 3.52MB disk, 8MB RAM
- **TLS termination proxy** – 3.58MB disk, 8MB RAM

Tinyx: Small Linux Distro for Target App



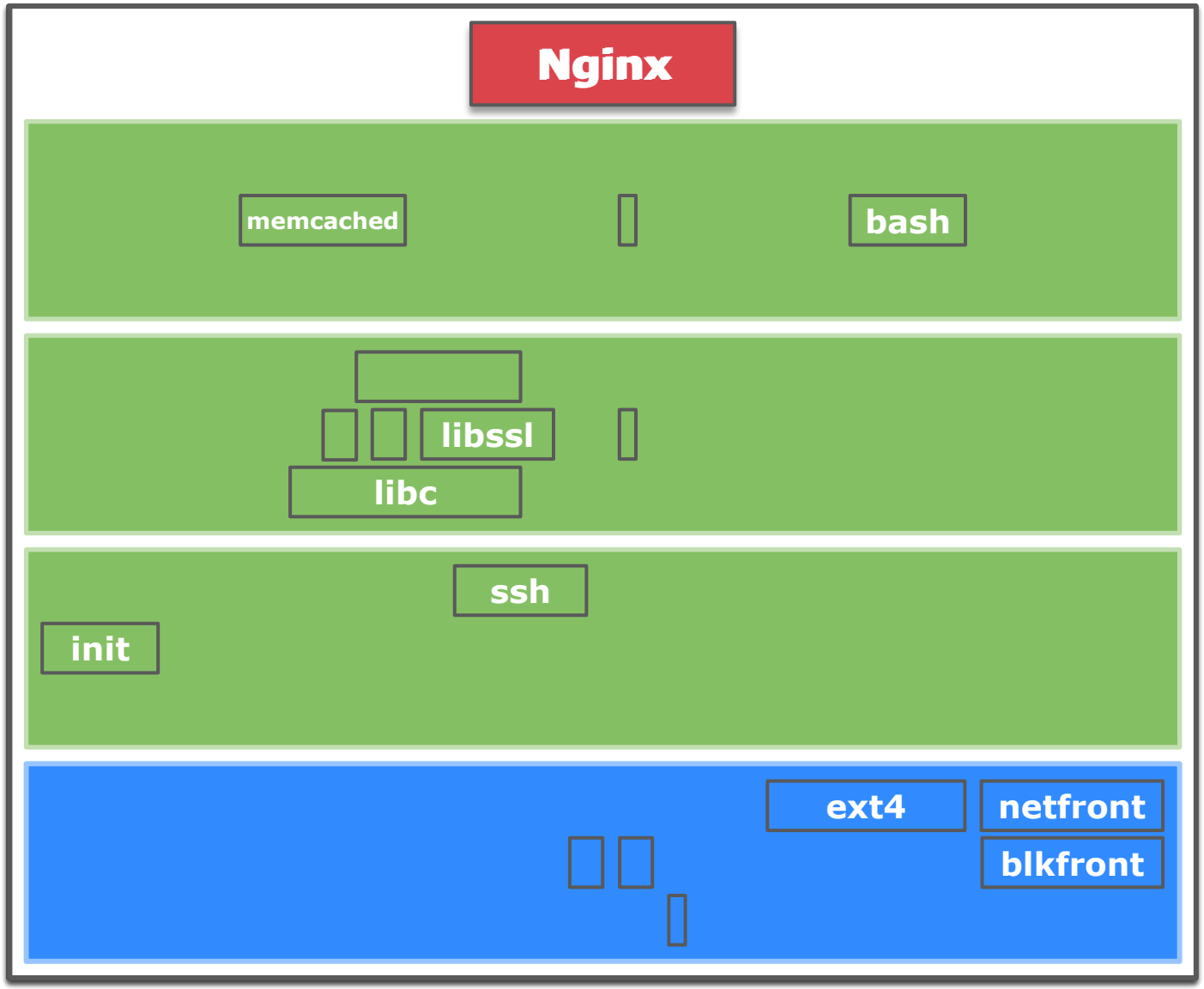
Tinyx: Small Linux Distro for Target App



Find dependencies

- objdump
- dpkg

Tinyx: Small Linux Distro for Target App



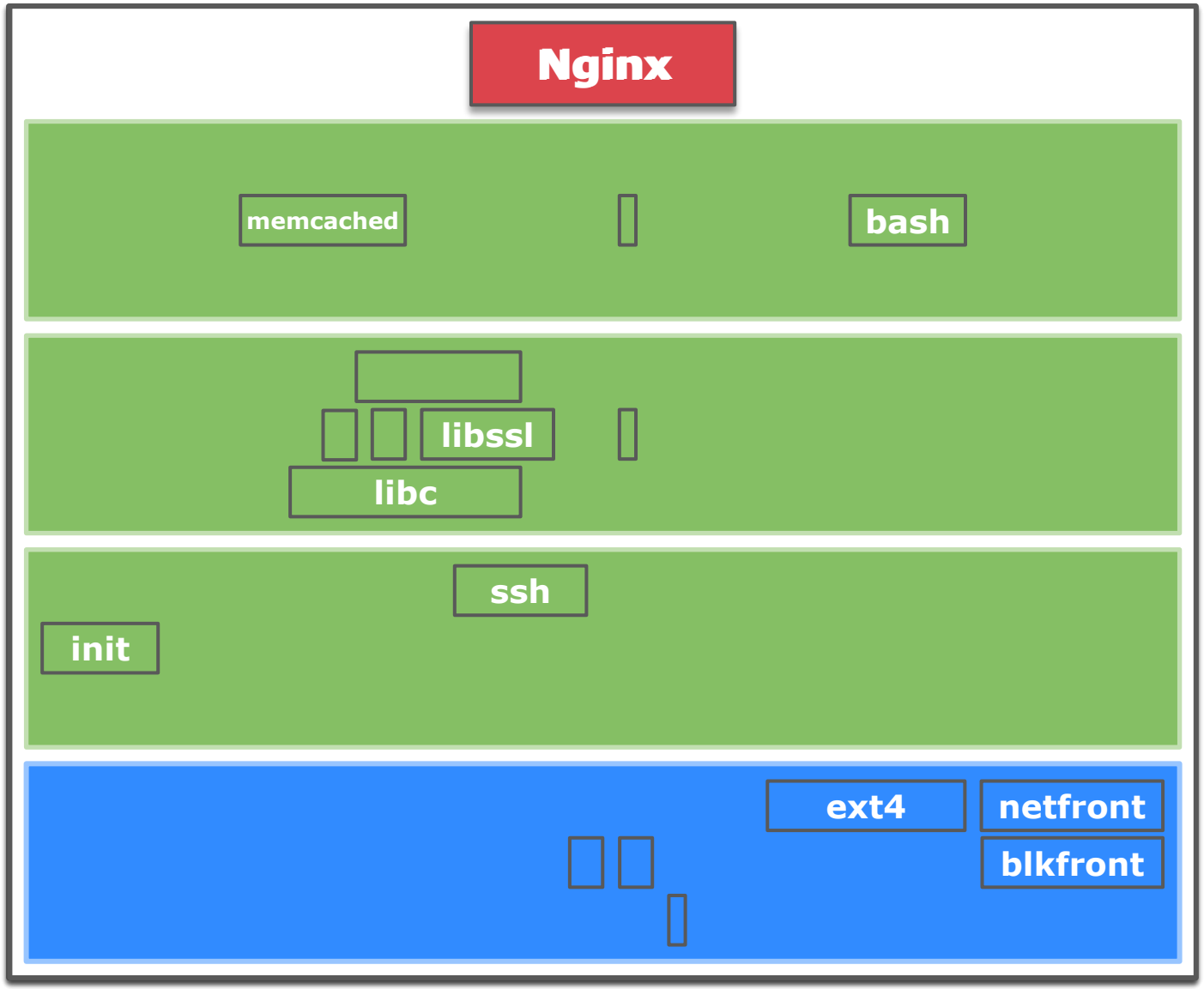
Find dependencies

- objdump
- dpkg

Install app & deps

- OverlayFS
- Cleanup

Tinyx: Small Linux Distro for Target App

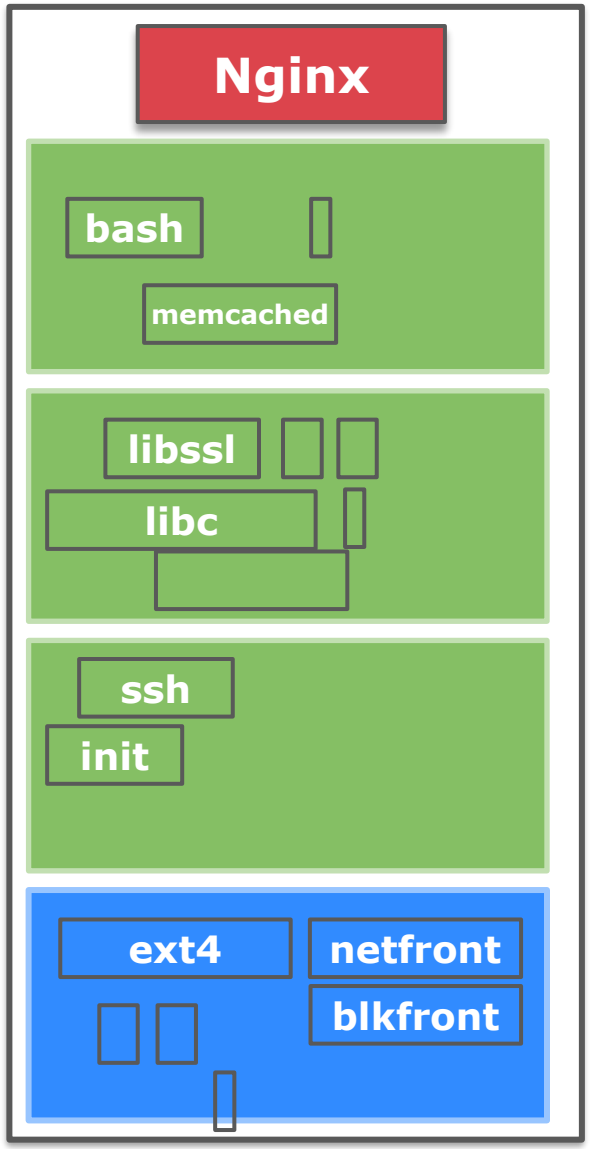


Find dependencies
- objdump
- dpkg

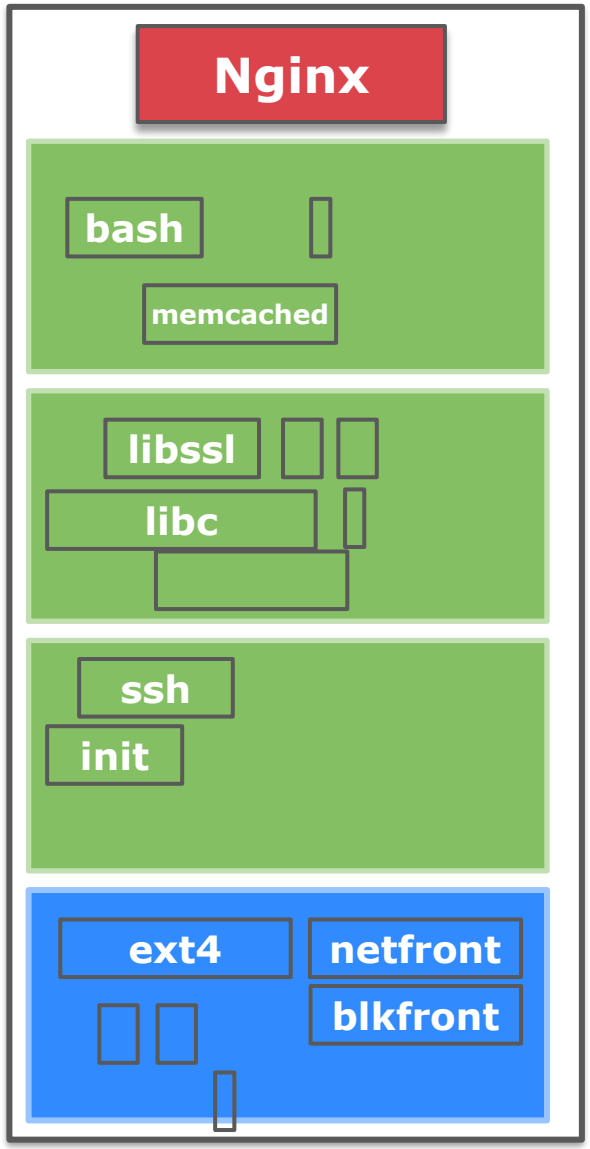
Install app & deps
- OverlayFS
- Cleanup

Small kernel:
- Remove drivers
- User config opts

Tinyx: Small Linux Distro for Target App



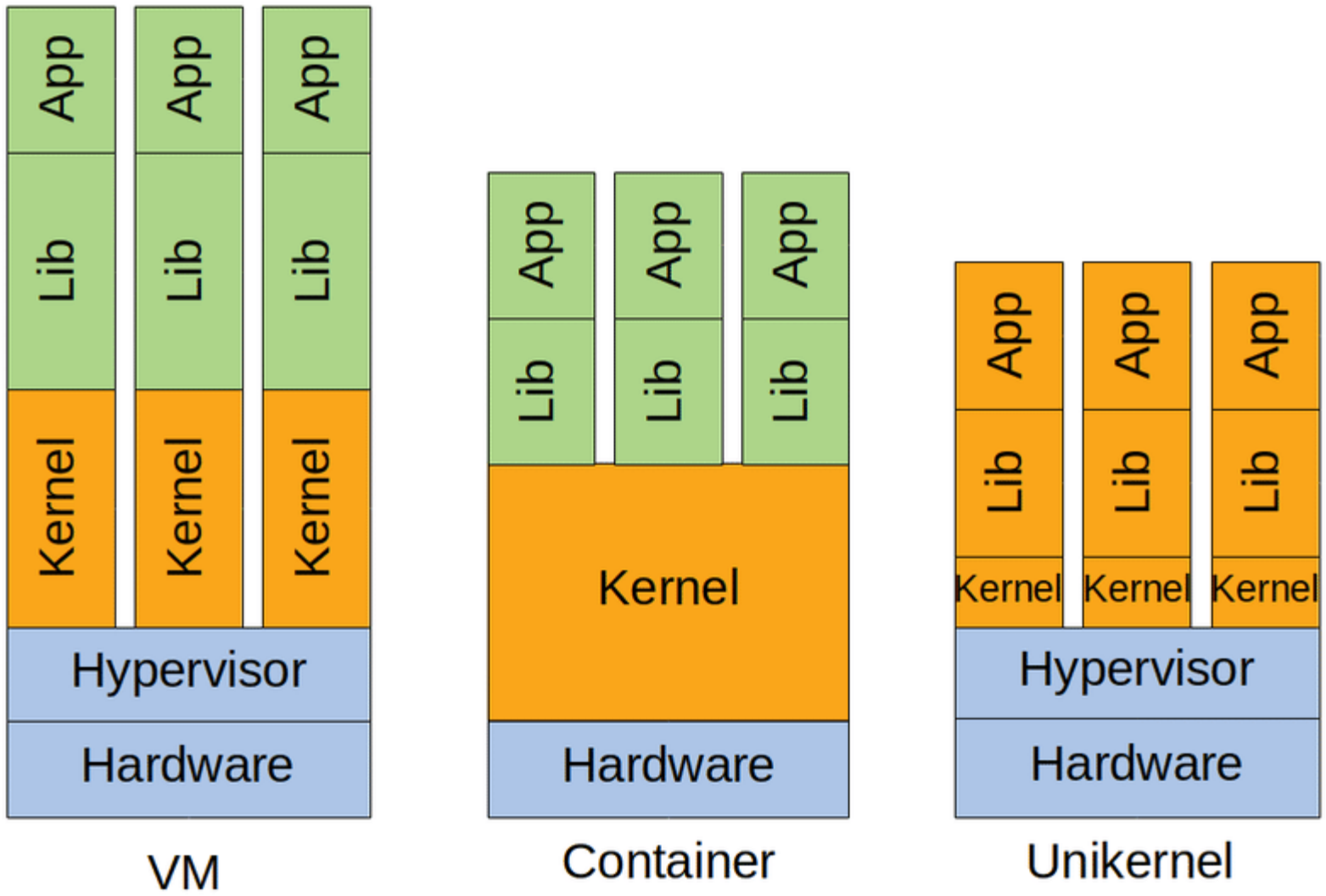
Tinyx: Small Linux Distro for Target App



Tinyx VMs are also lightweight:

- Kernel: 1.5MB (compared to 8MB)
- Image size – 10-30MB (compared to 1GB).
- Boot: 200ms instead of 2s.

Unikernels

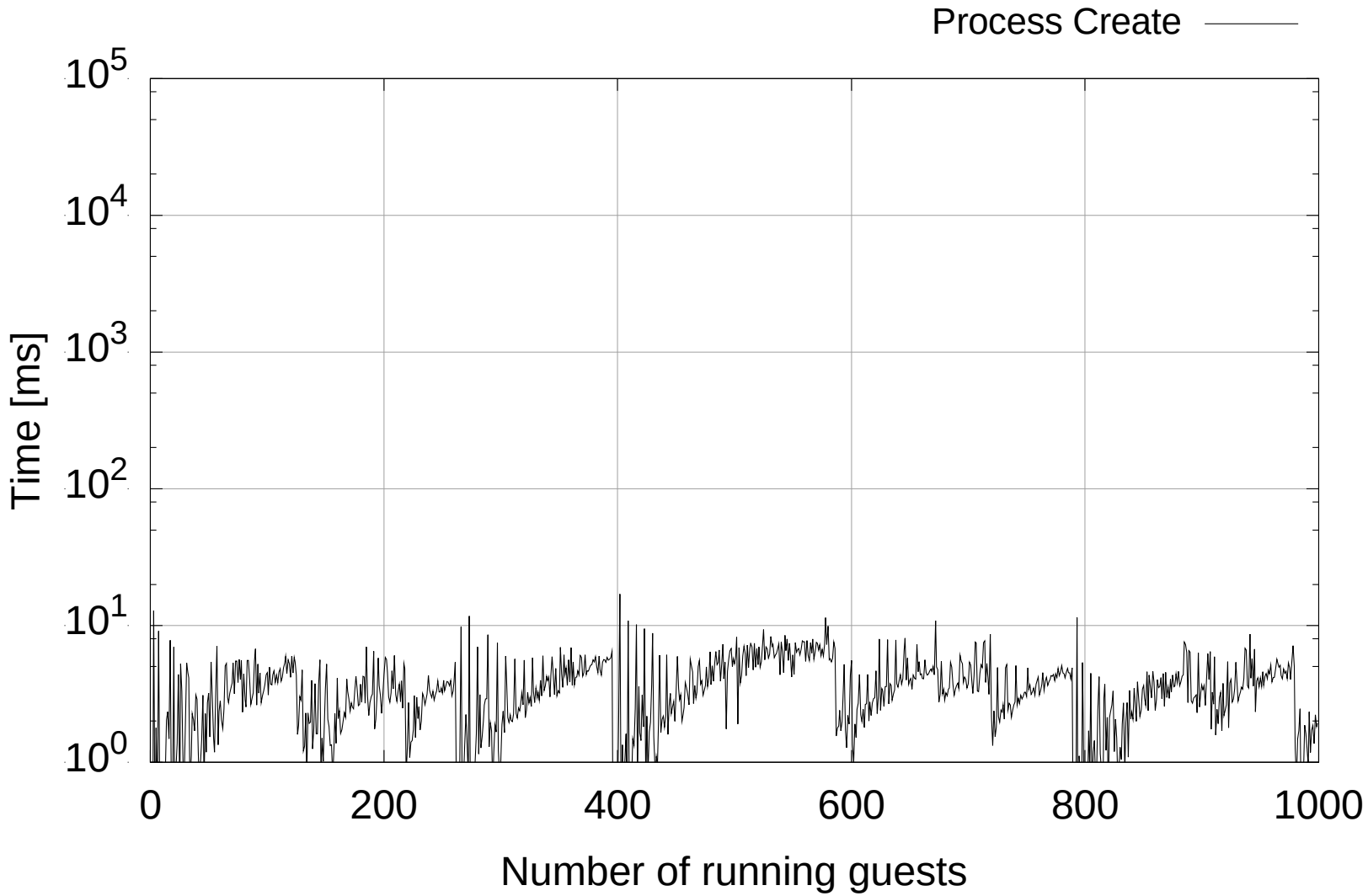

App
App
App
Lib
Lib
Lib
Kernel
Kernel
Kernel
Hypervisor
Hardware
VM
App
App
App
Lib
Lib
Lib
Kernel
Hardware
Container
App
App
App
Lib
Lib
Lib
Kernel
Kernel
Kernel
Hypervisor
Hardware
Unikernel

Unikernel perspective

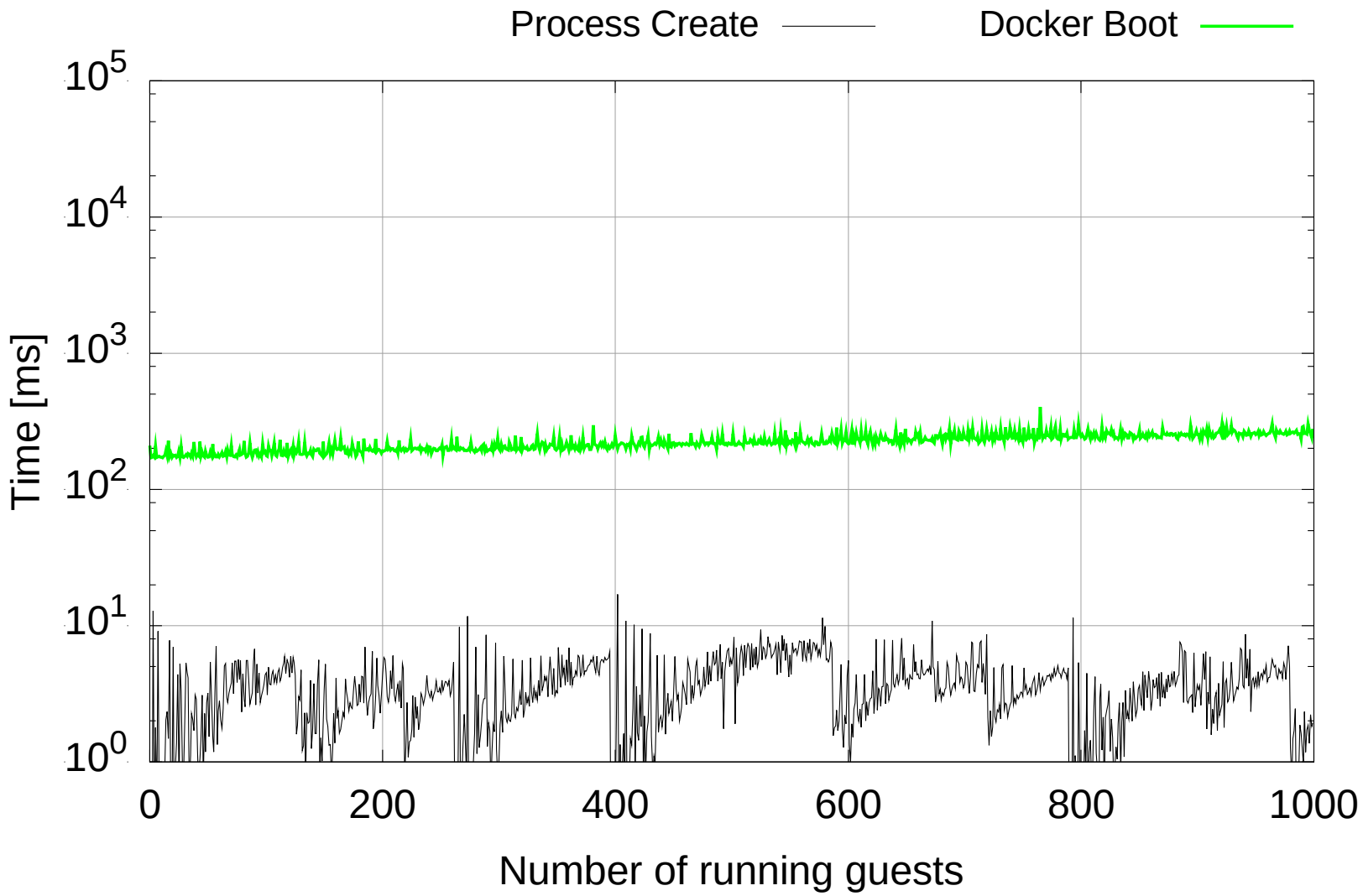
- Producing unikernel images is complicated and requires deep knowledge
  - Producing a unikernel is specific to an application
  - Standard tooling never materialized
- Application frameworks are complex and not unikernel friendly
  - Applications like fork
- No standard way to coordinate processes in a unikernel
- Linux customization almost as good and better supported
  - Tinyx!

Are lightweights VMs enough for good performance?

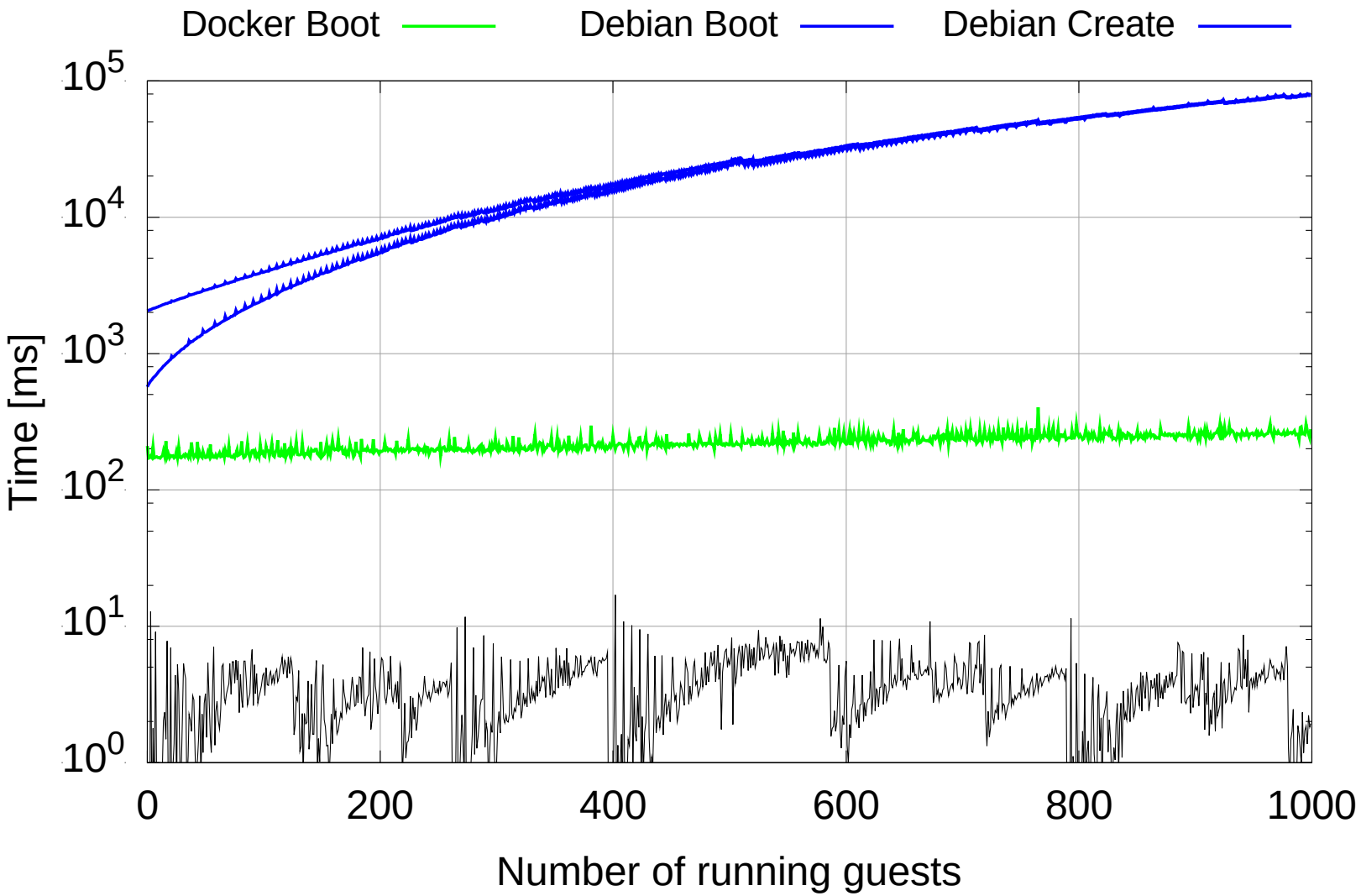
VM, Container and Process Creation Times



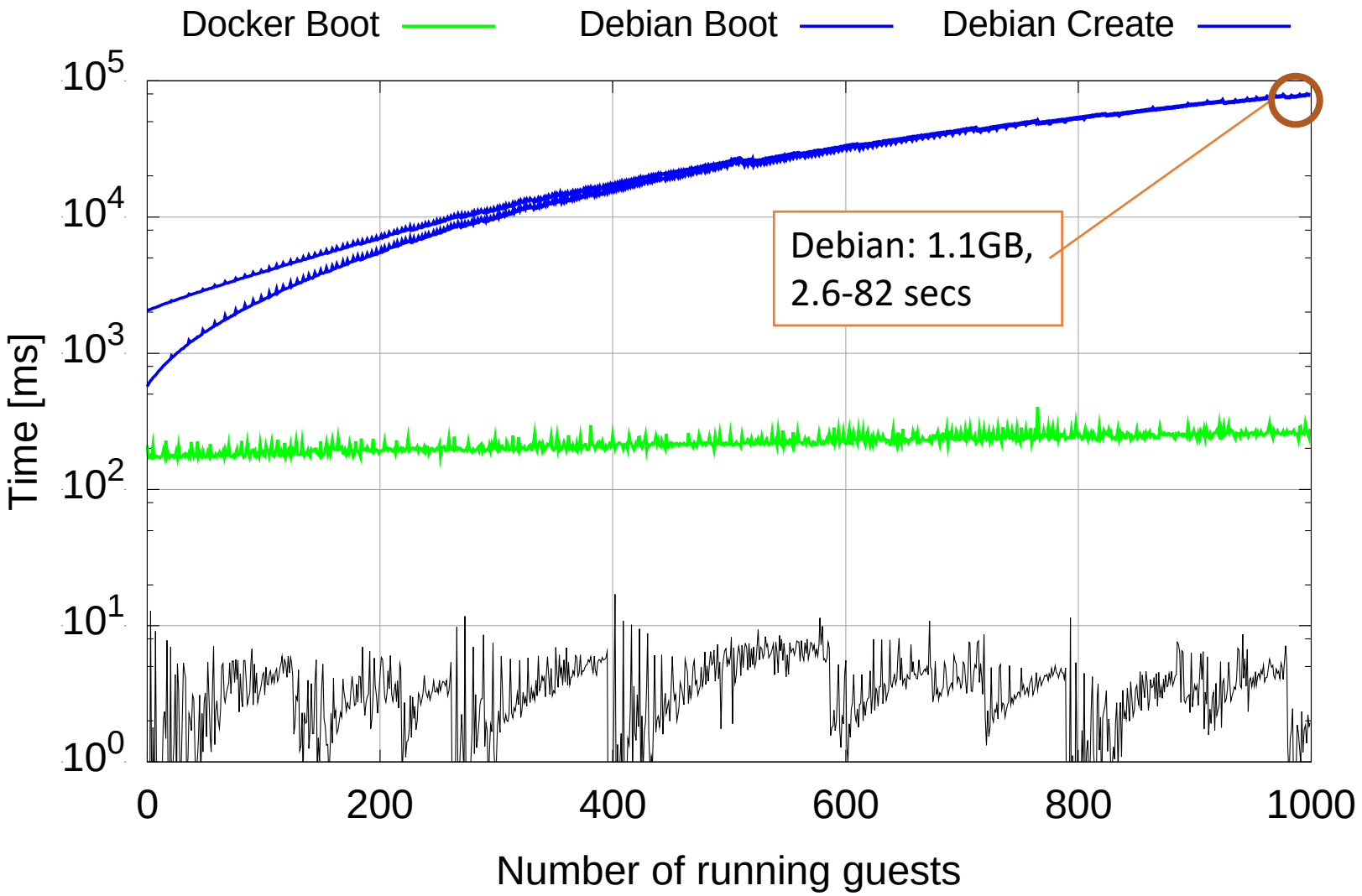
VM, Container and Process Creation Times



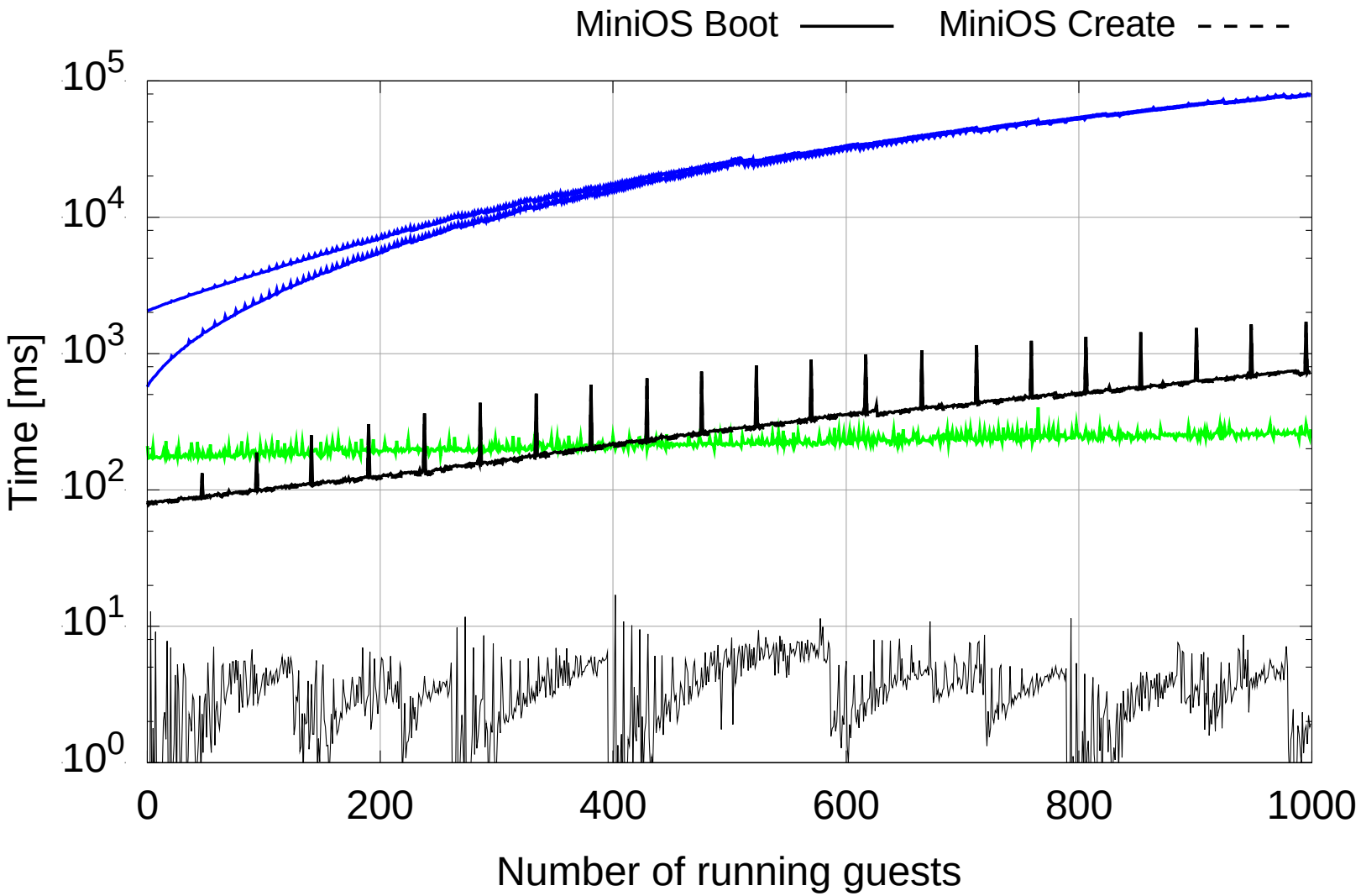
VM, Container and Process Creation Times



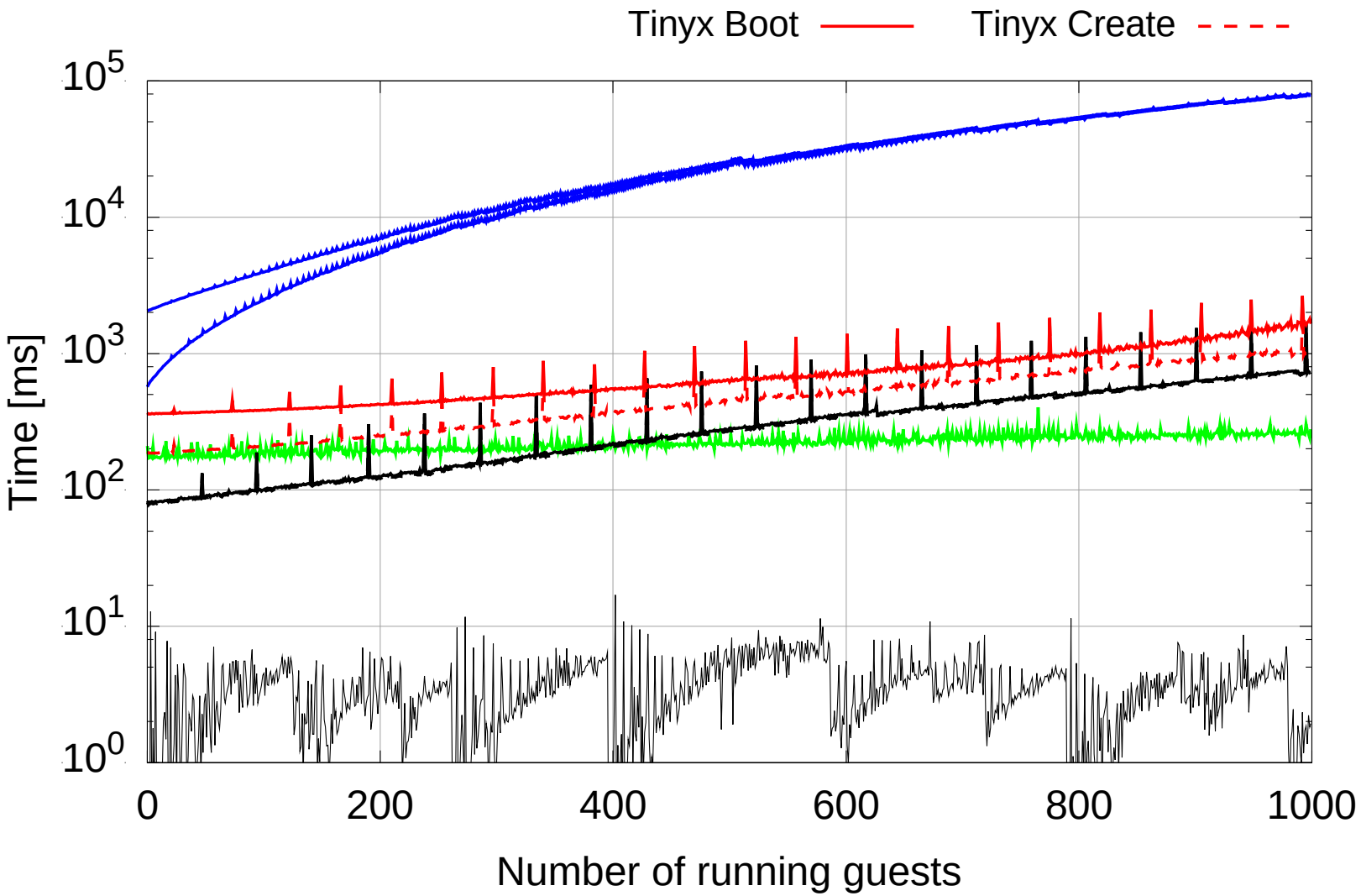
VM, Container and Process Creation Times



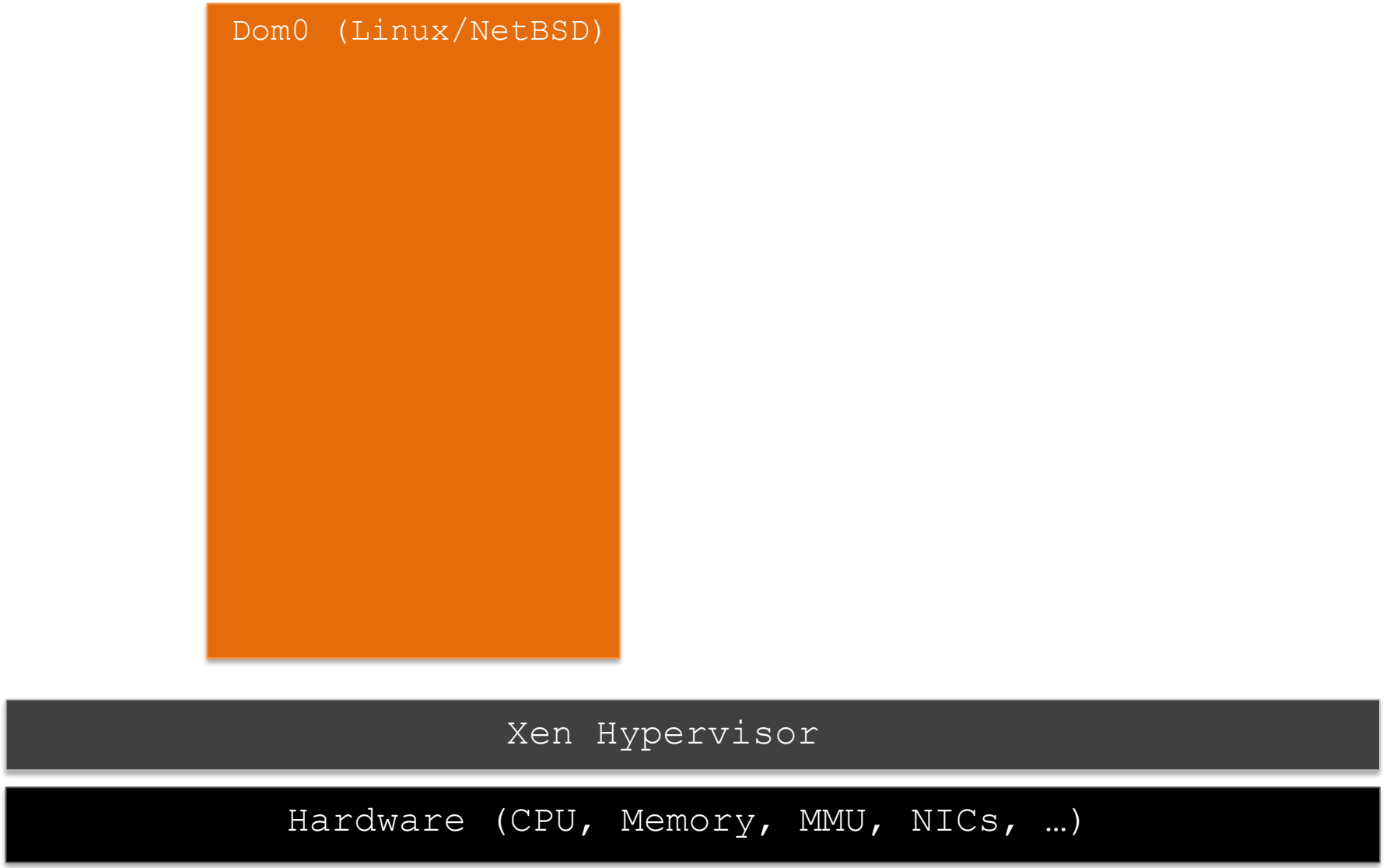
VM, Container and Process Creation Times



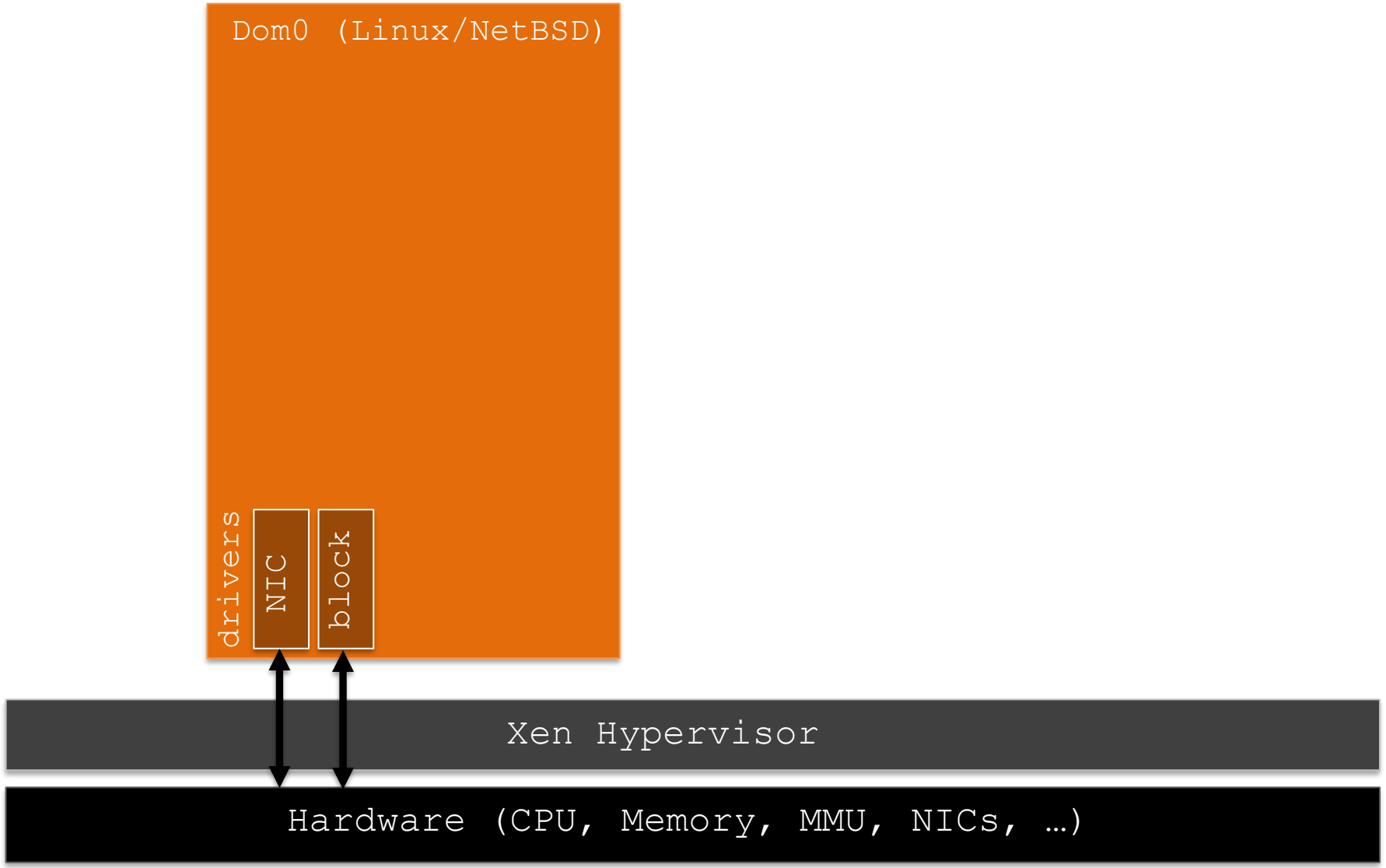
VM, Container and Process Creation Times



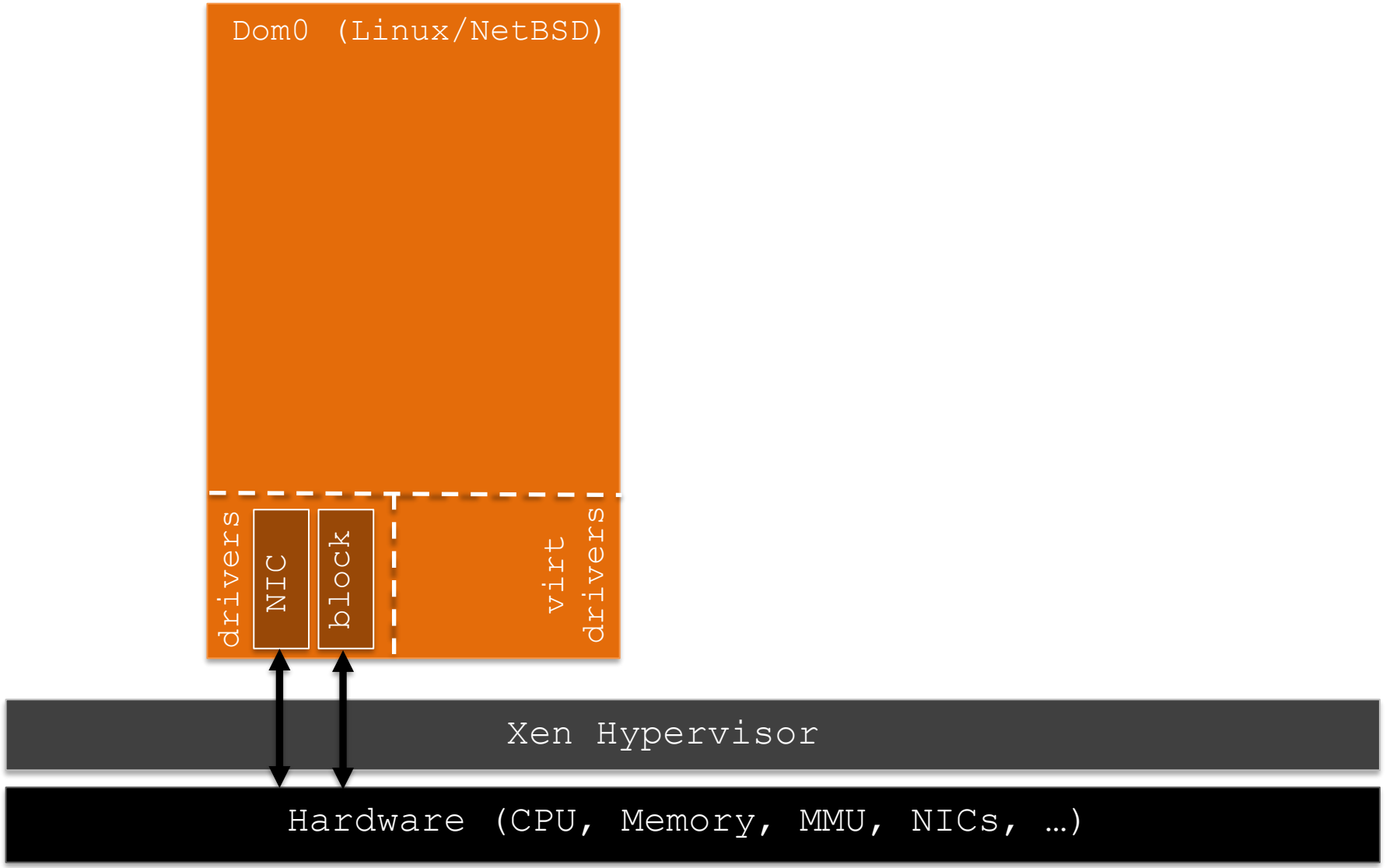
A Quick Xen Primer



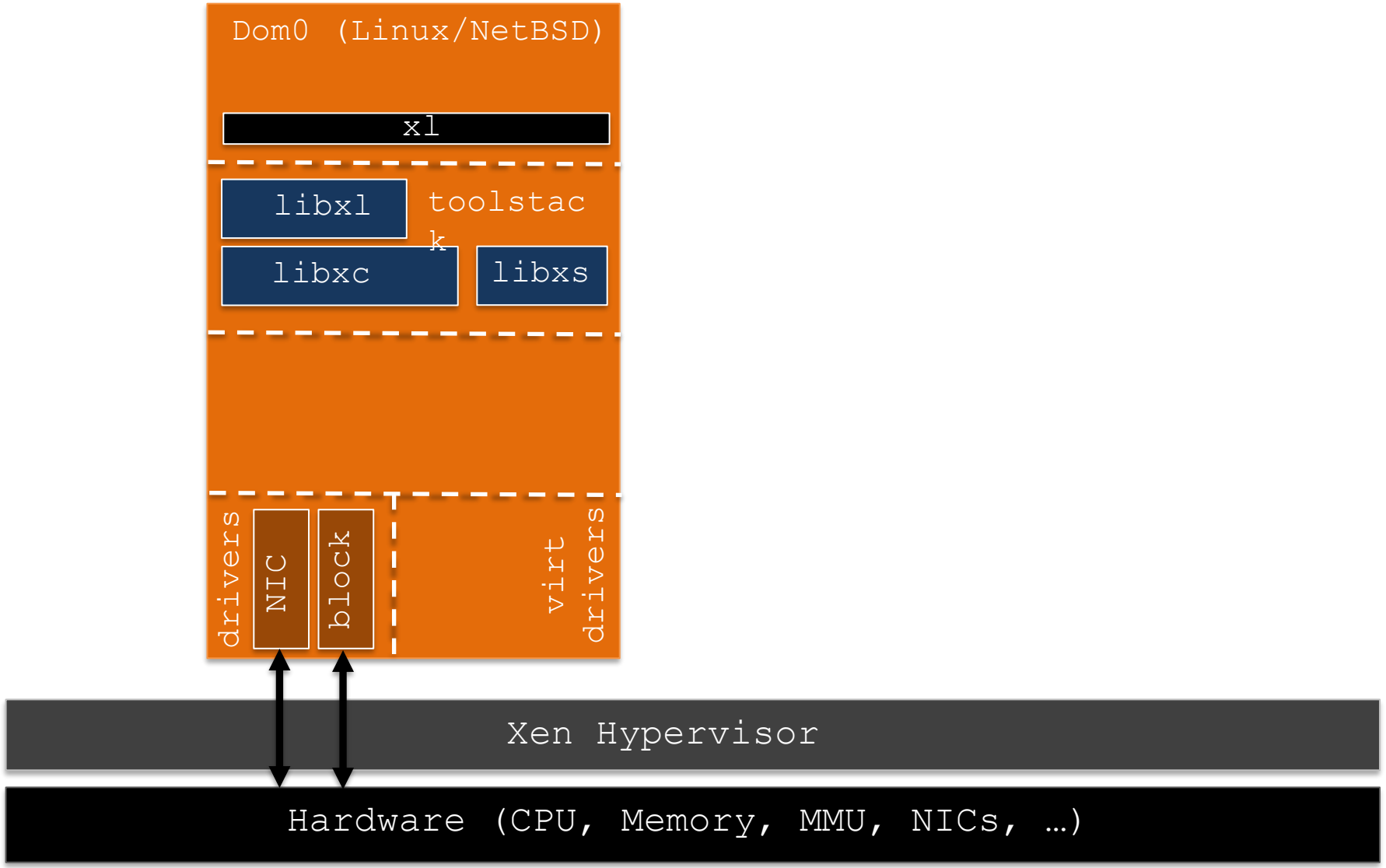
A Quick Xen Primer



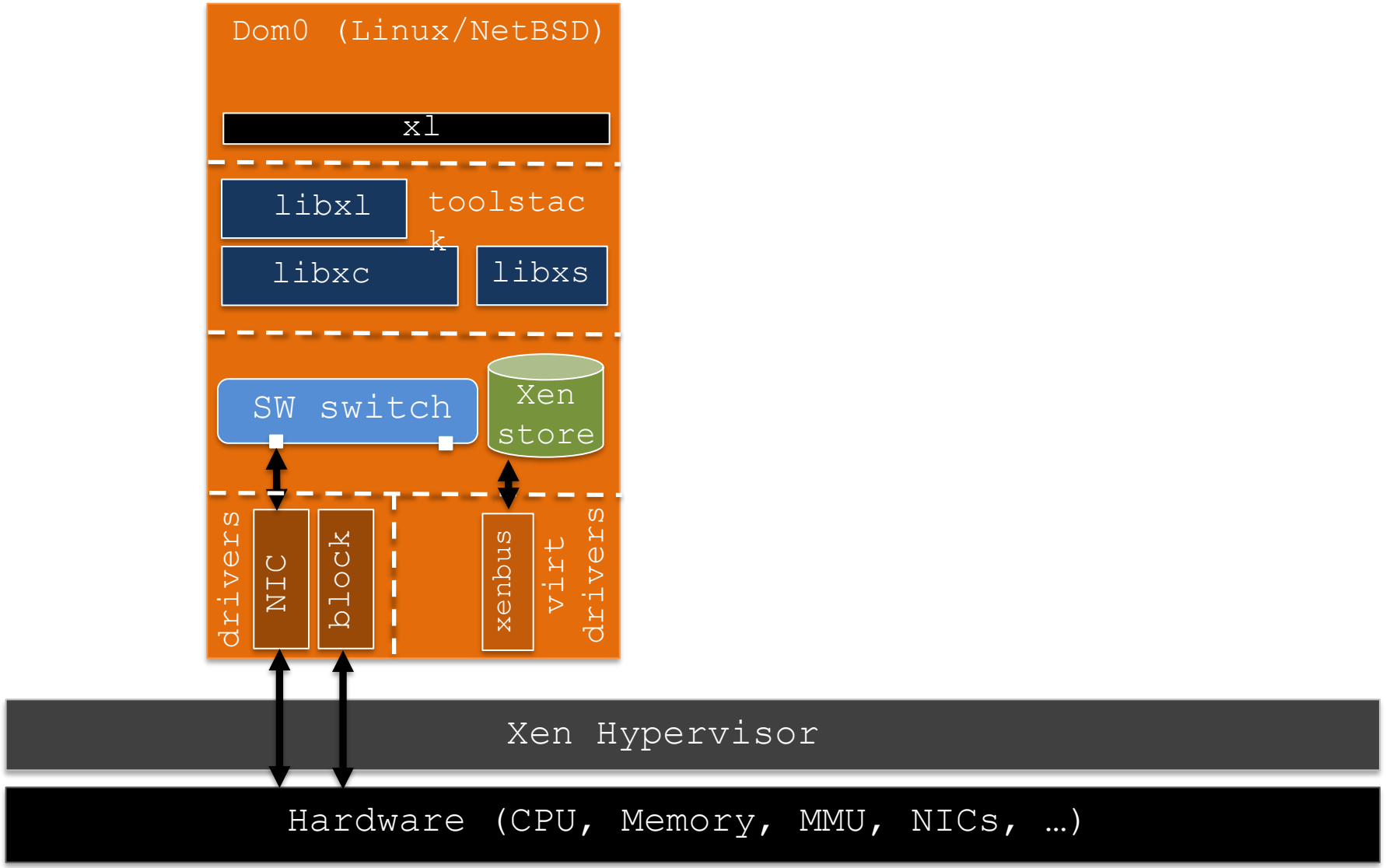
A Quick Xen Primer



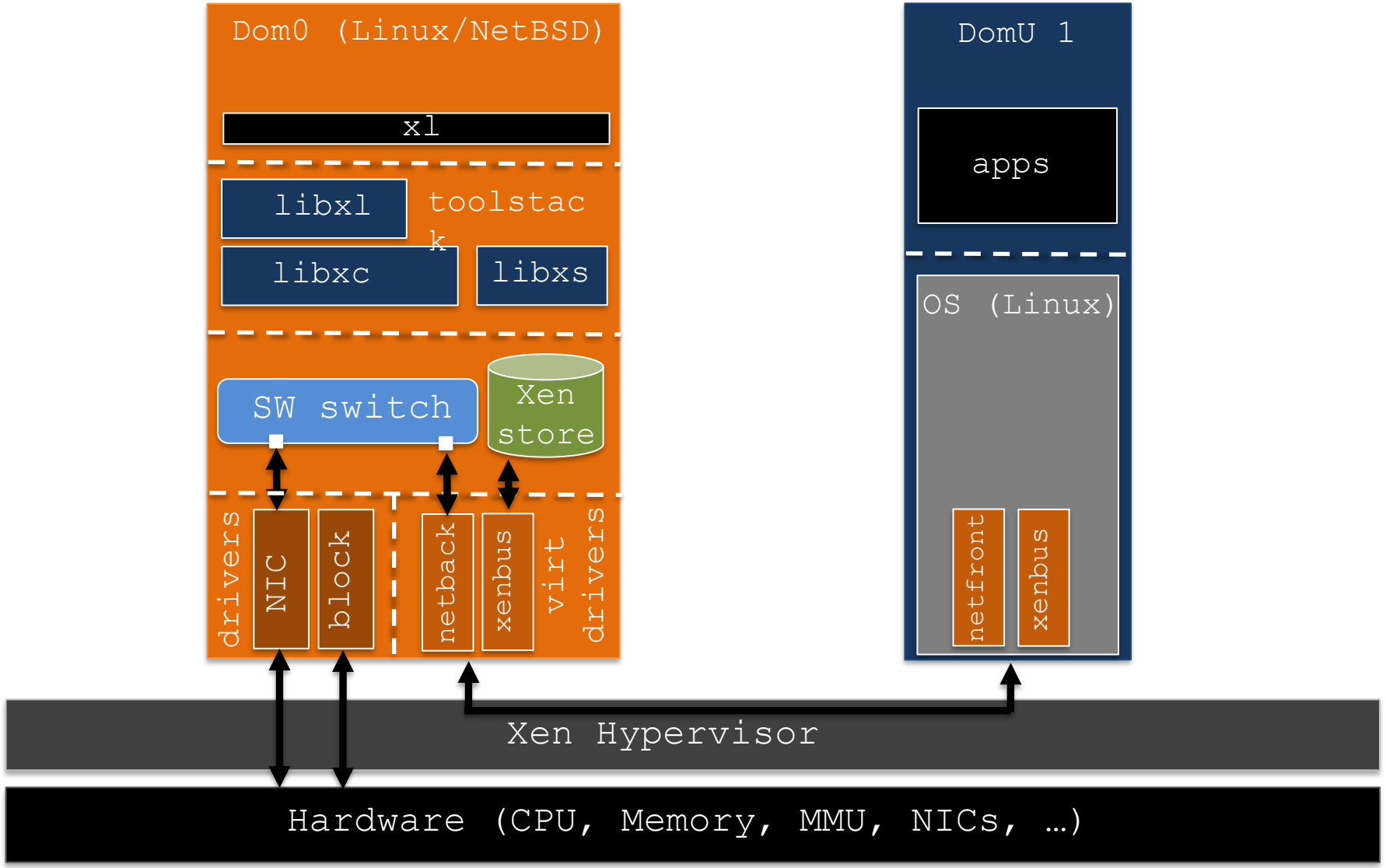
A Quick Xen Primer



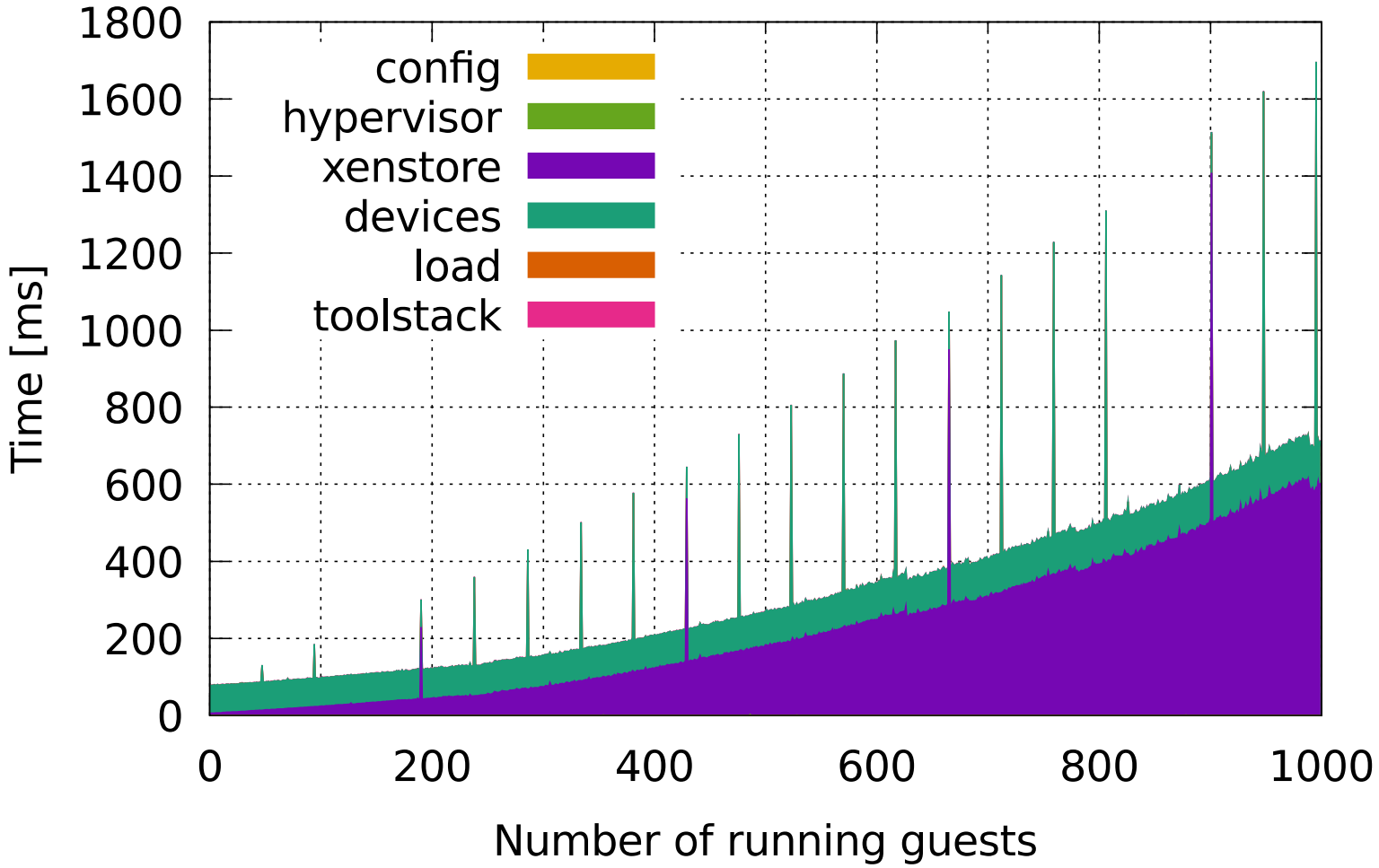
A Quick Xen Primer



A Quick Xen Primer

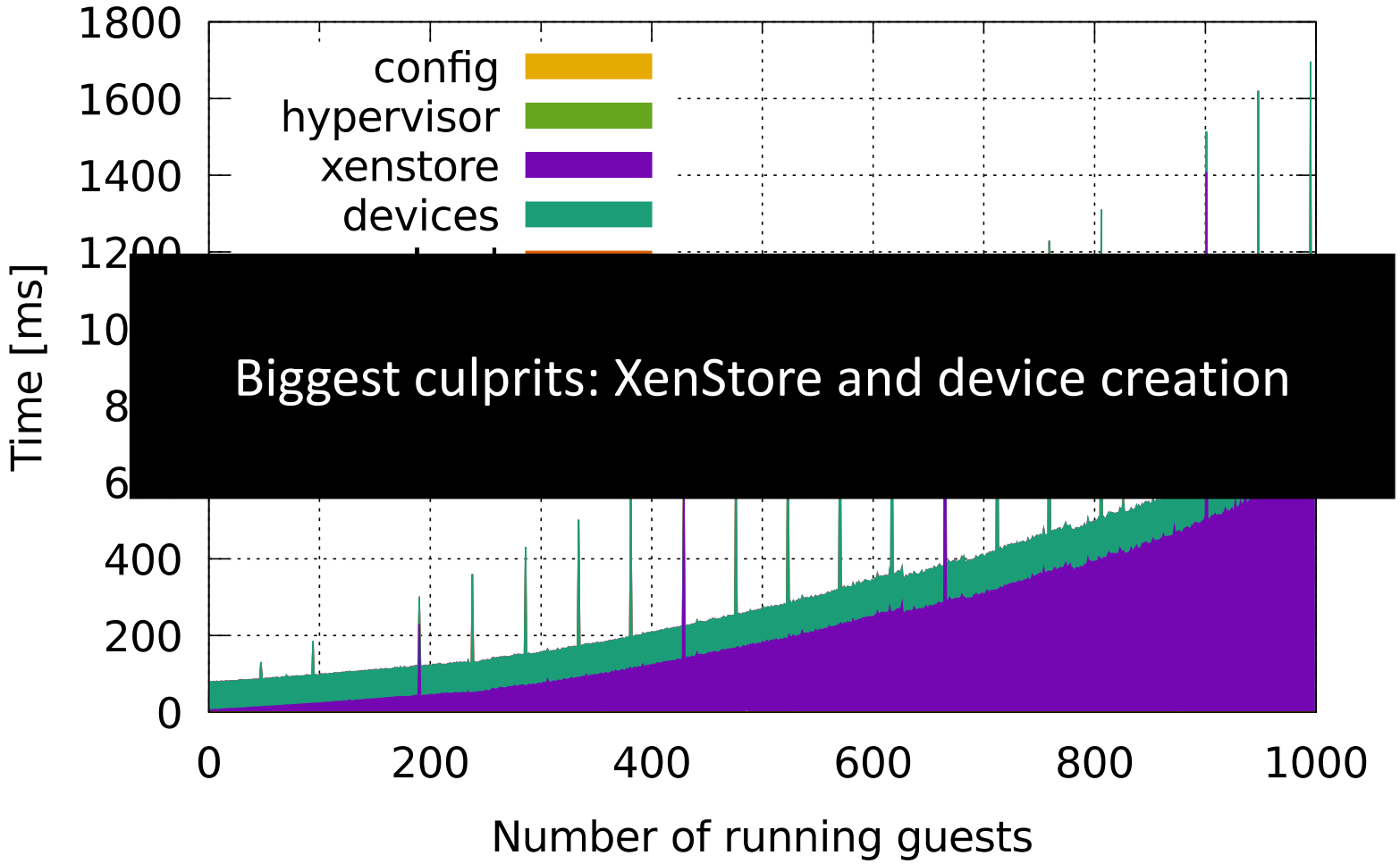


Creation Times – Breakdown (unikernel)



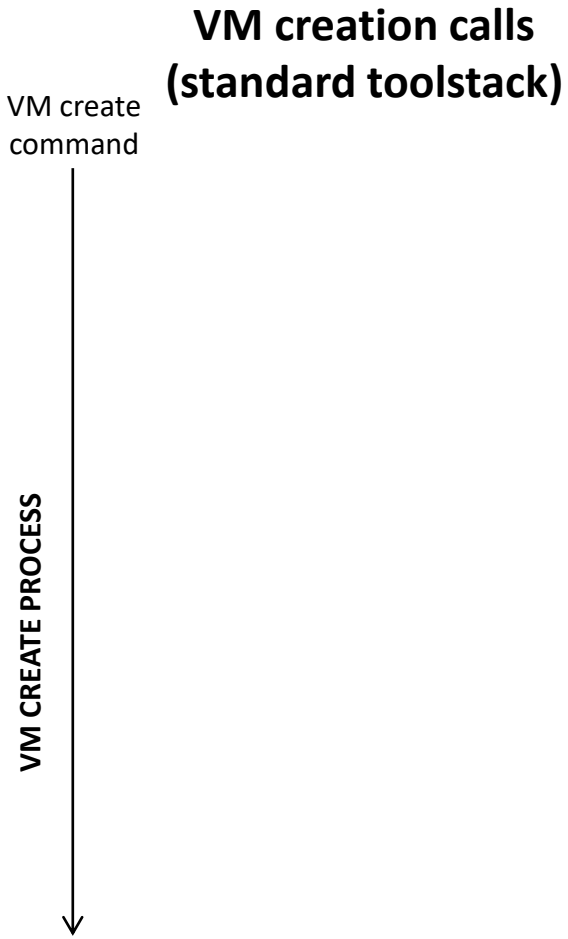
** Note: Spikes in graph due to XenStore's garbage collector (it's written in OCaml)*

Creation Times – Breakdown (unikernel)

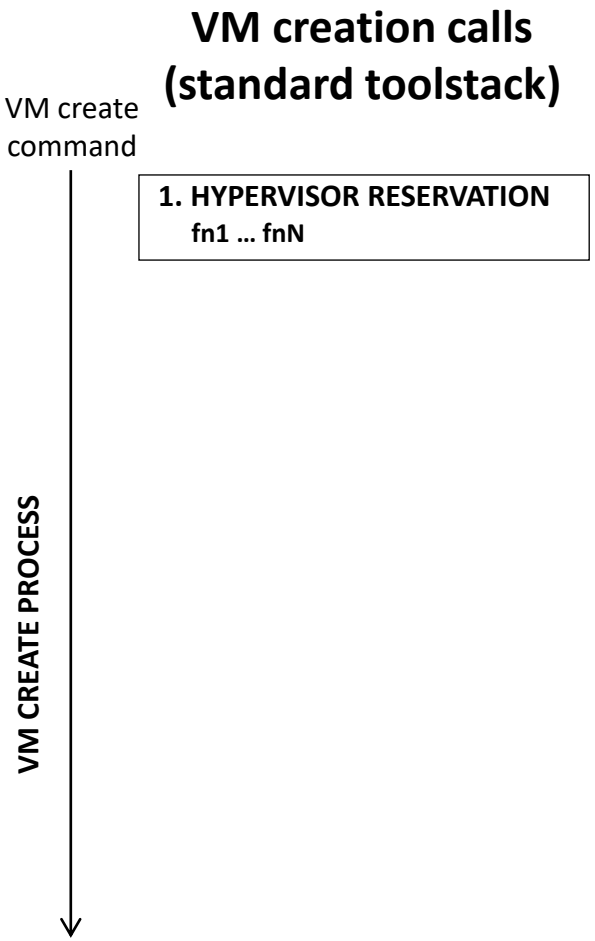


** Note: Spikes in graph due to XenStore's garbage collector (it's written in OCaml)*

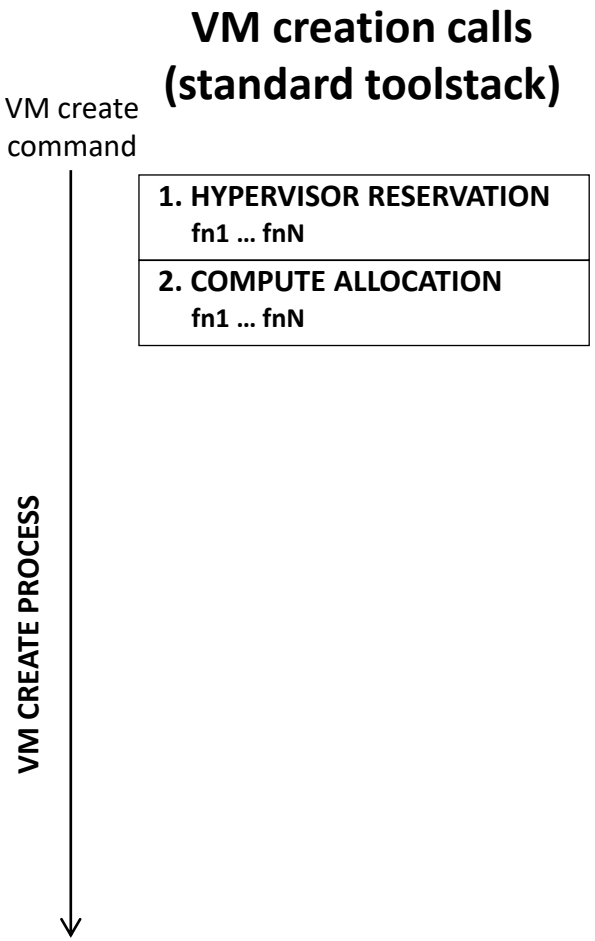
VM creation steps



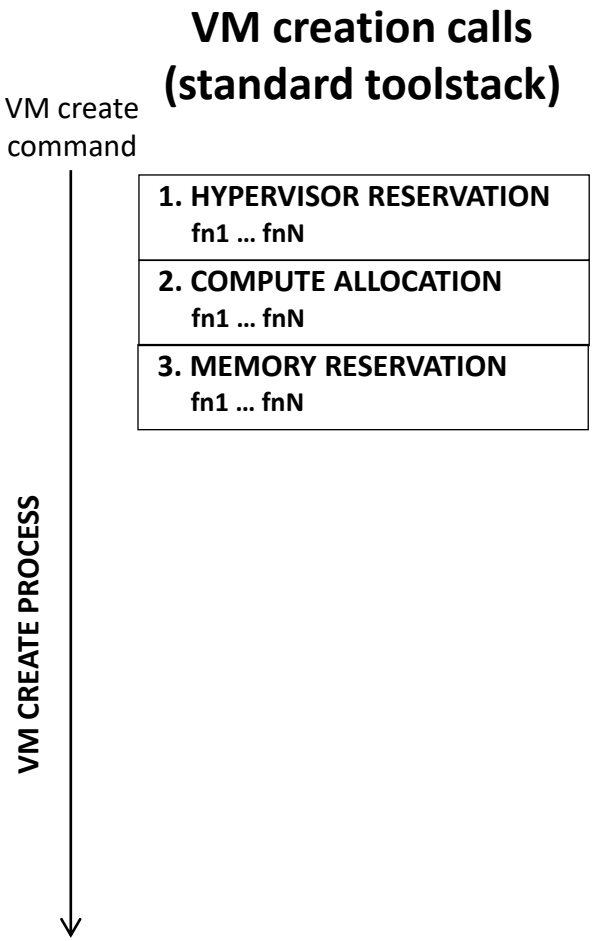
VM creation steps



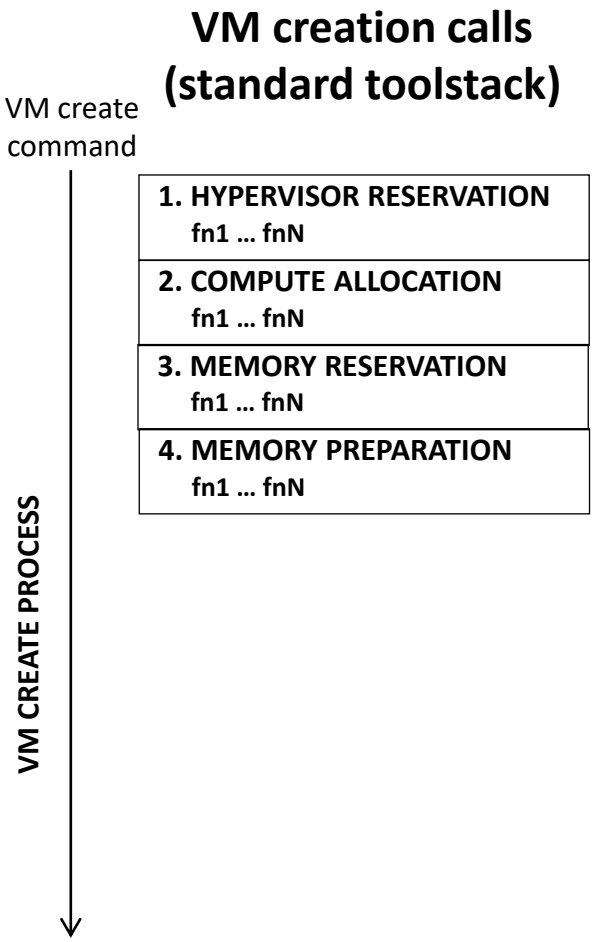
VM creation steps



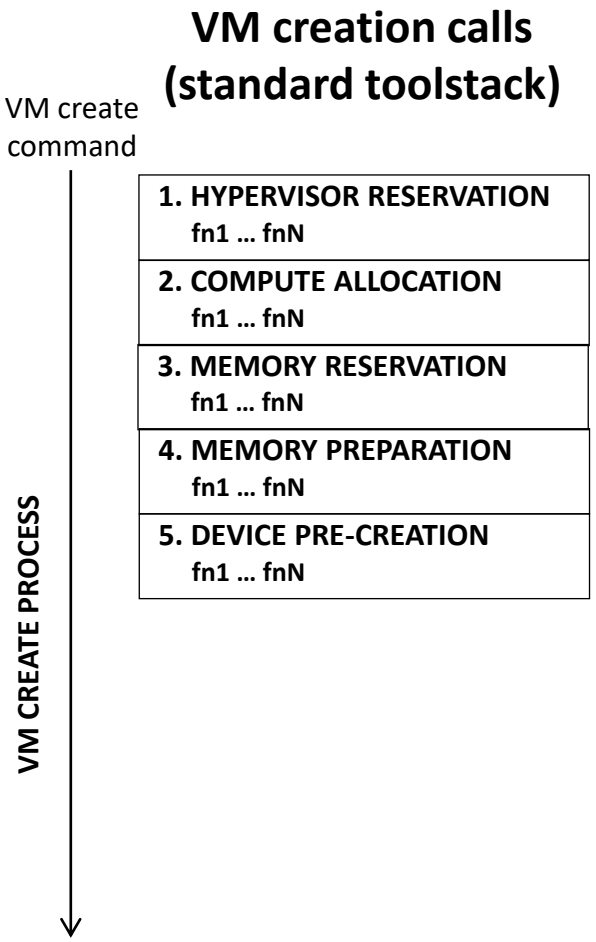
VM creation steps



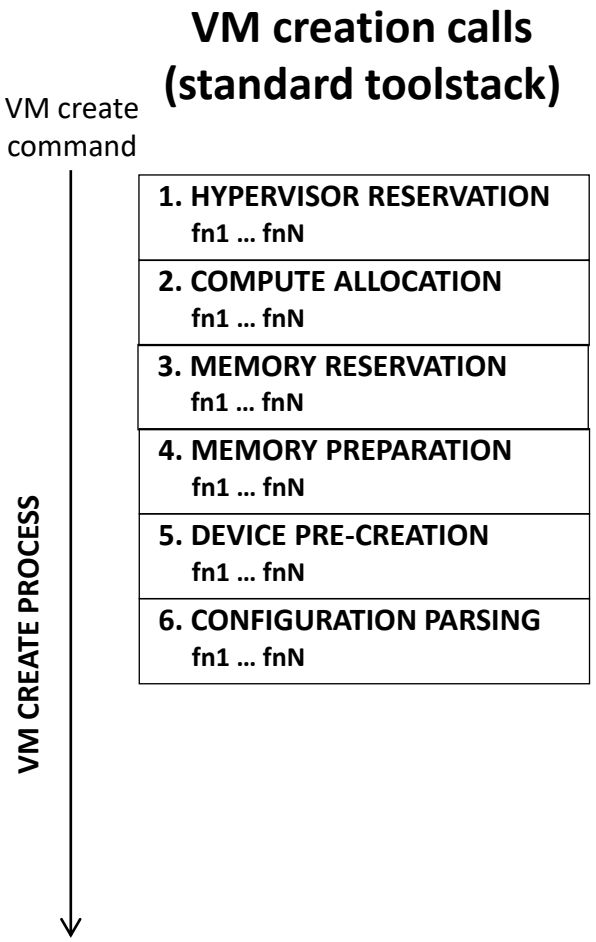
VM creation steps



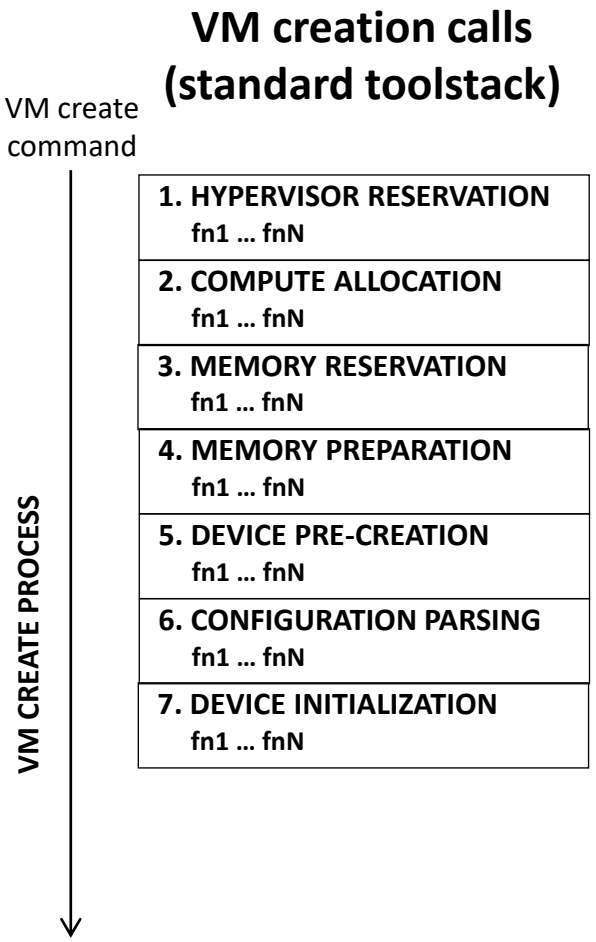
VM creation steps



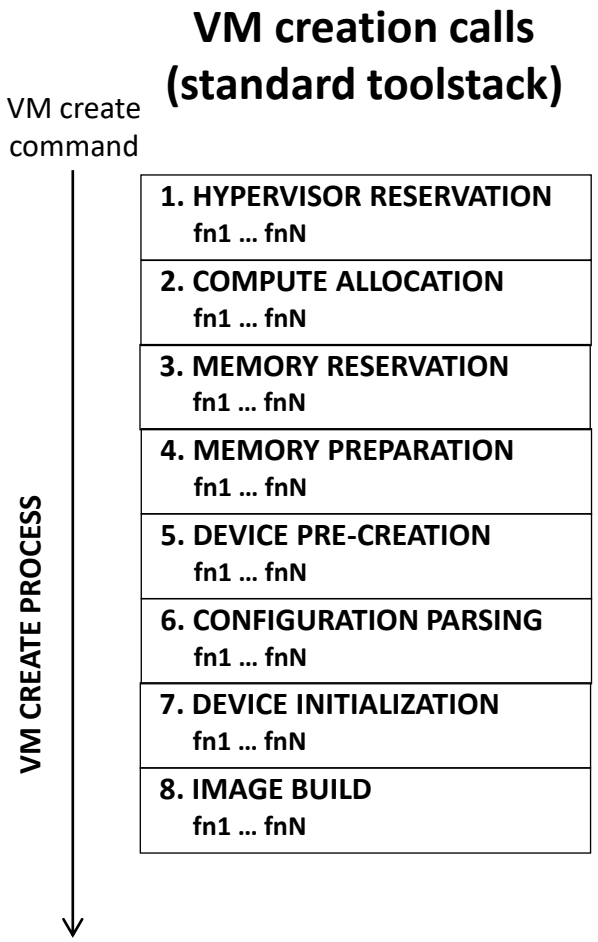
VM creation steps



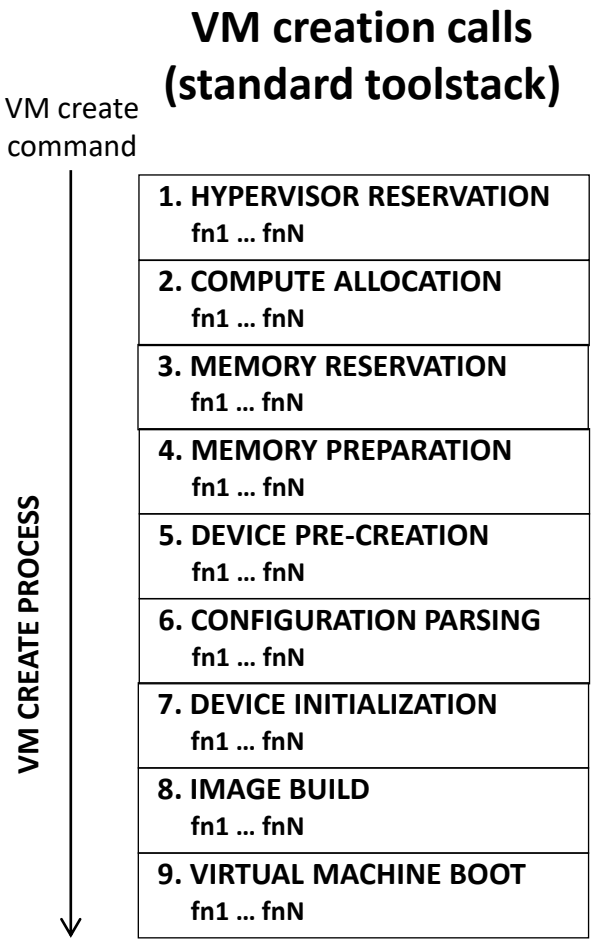
VM creation steps



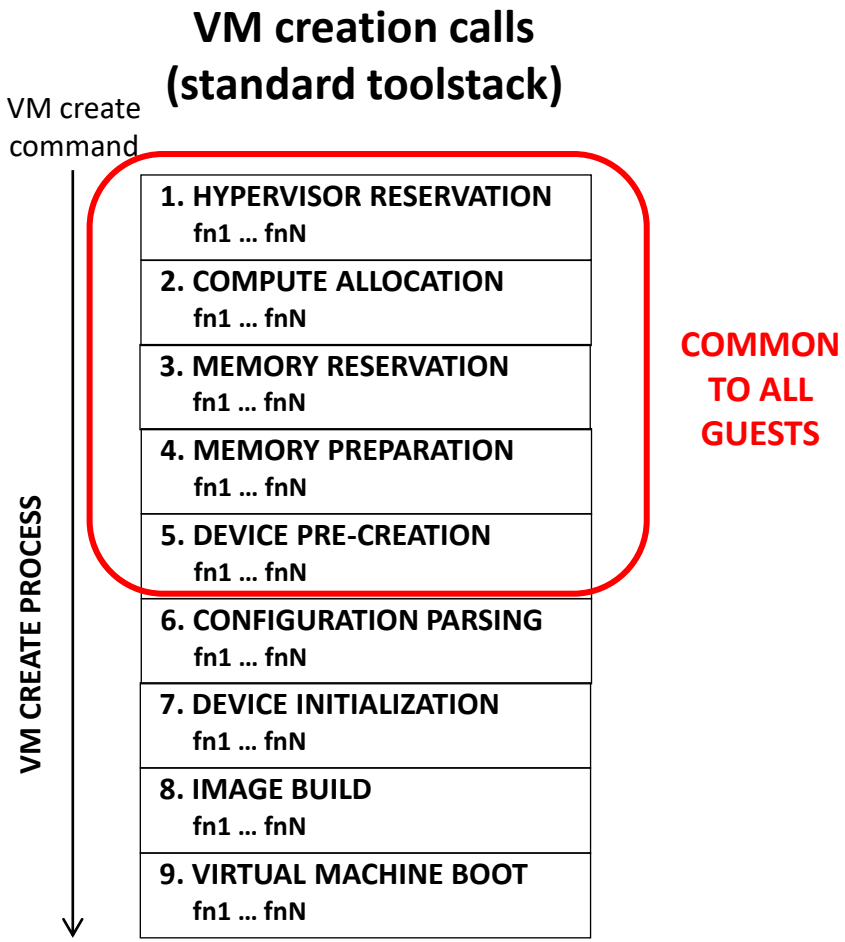
VM creation steps



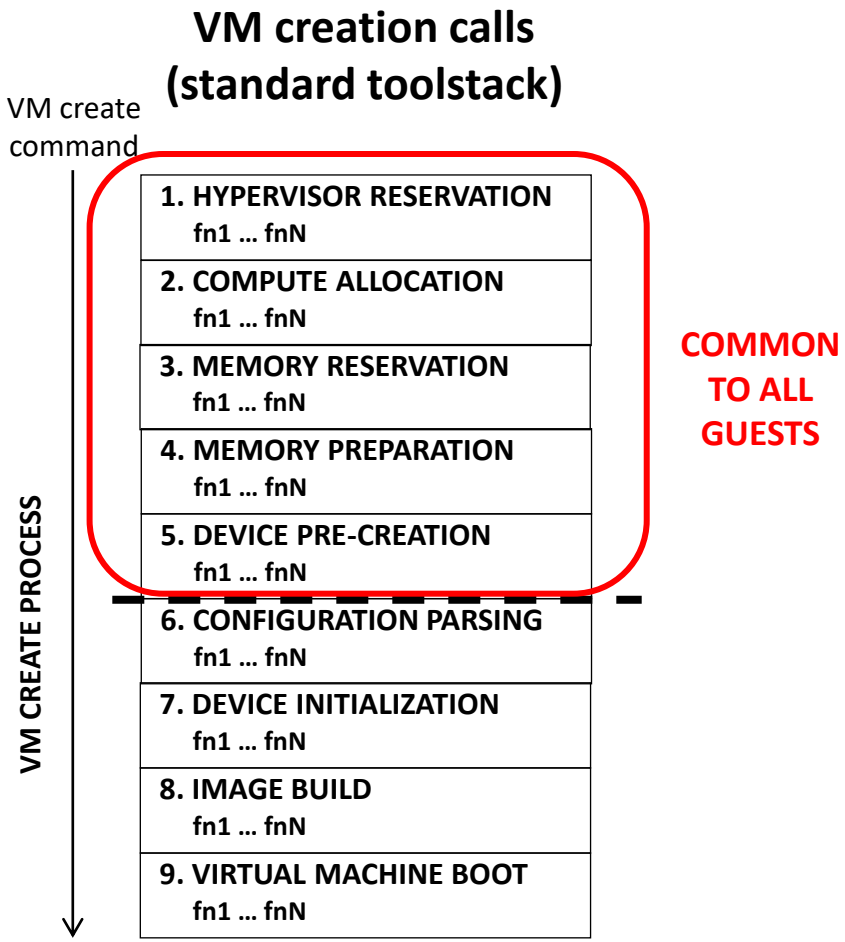
VM creation steps



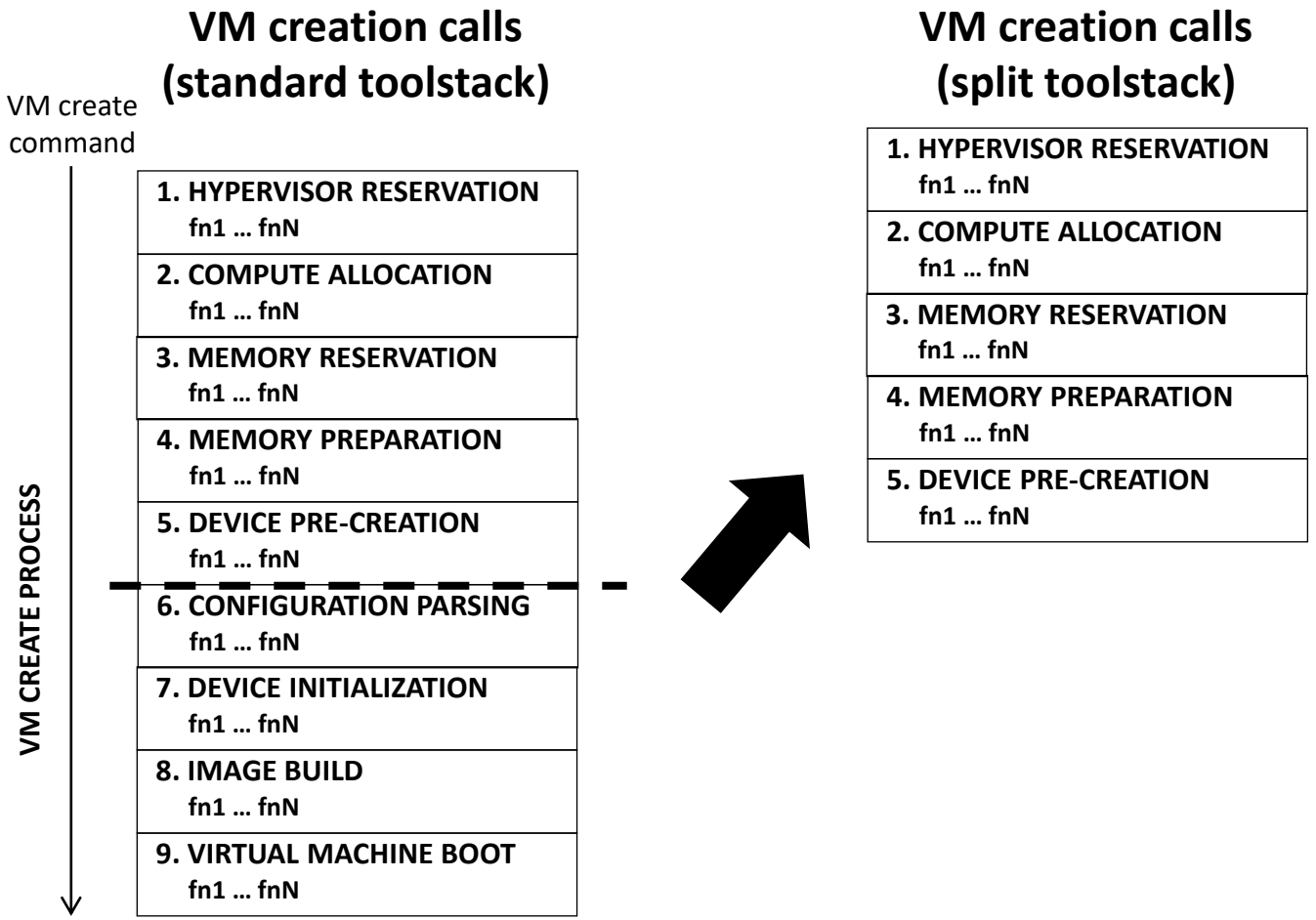
VM creation steps



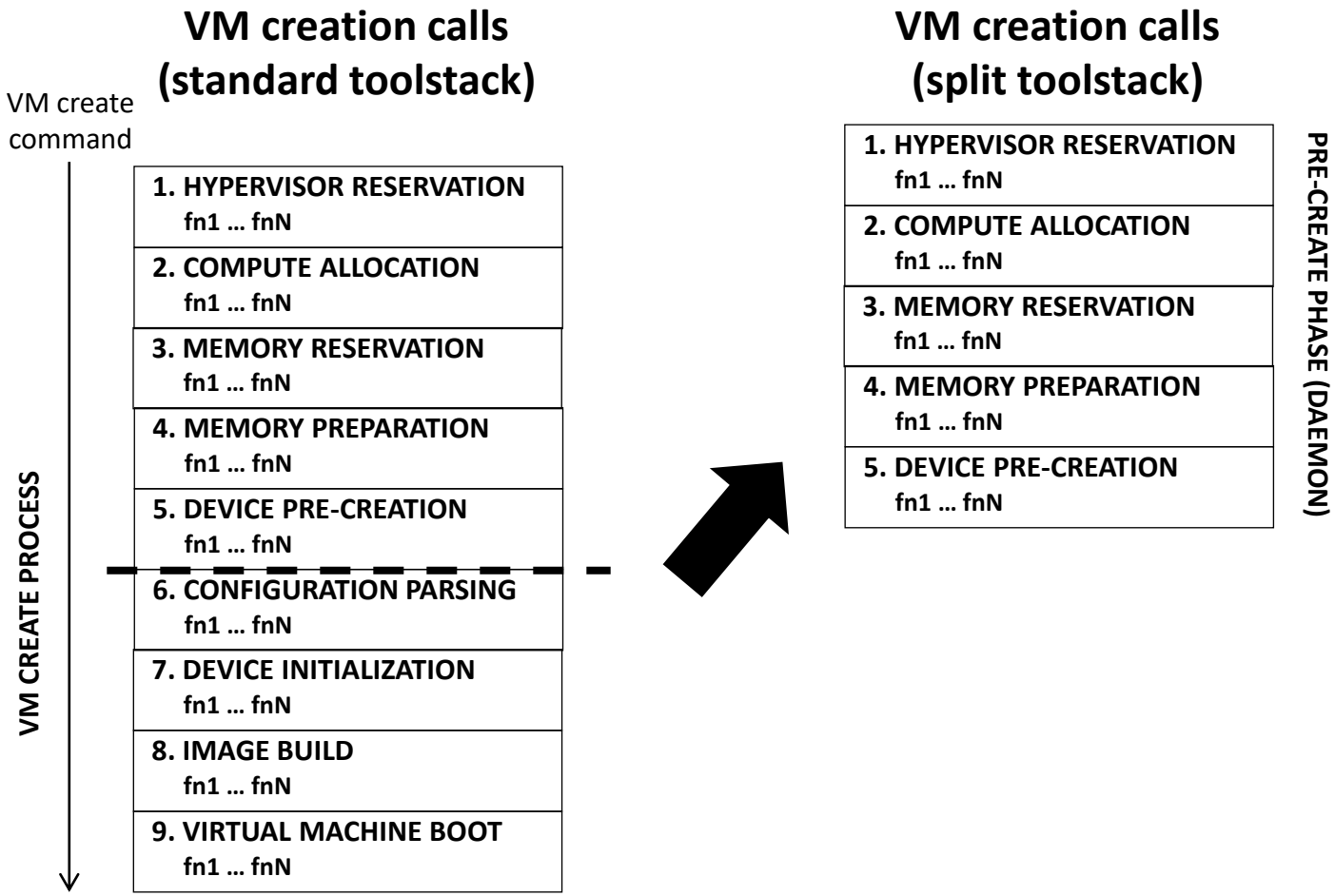
VM creation steps



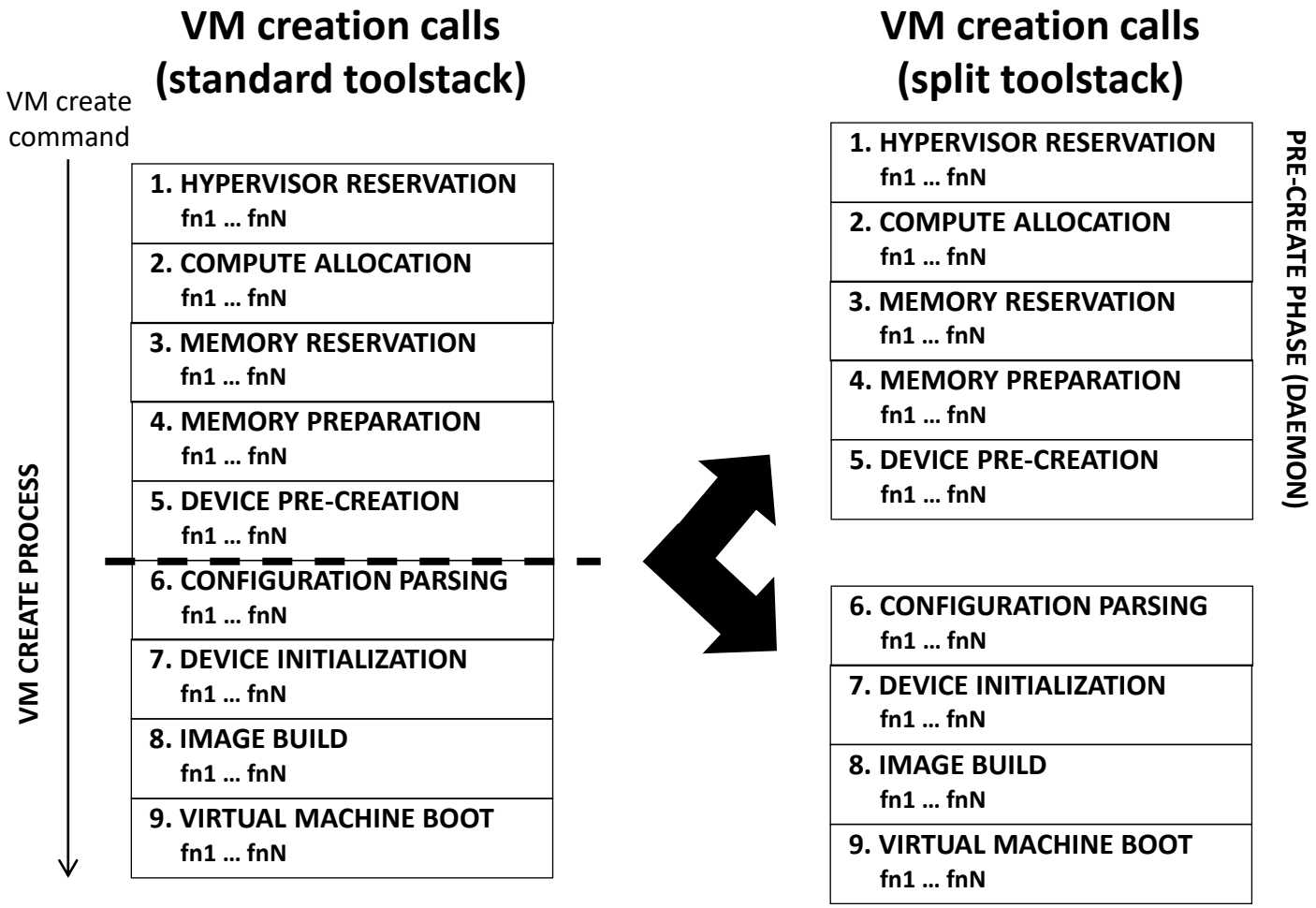
VM creation steps



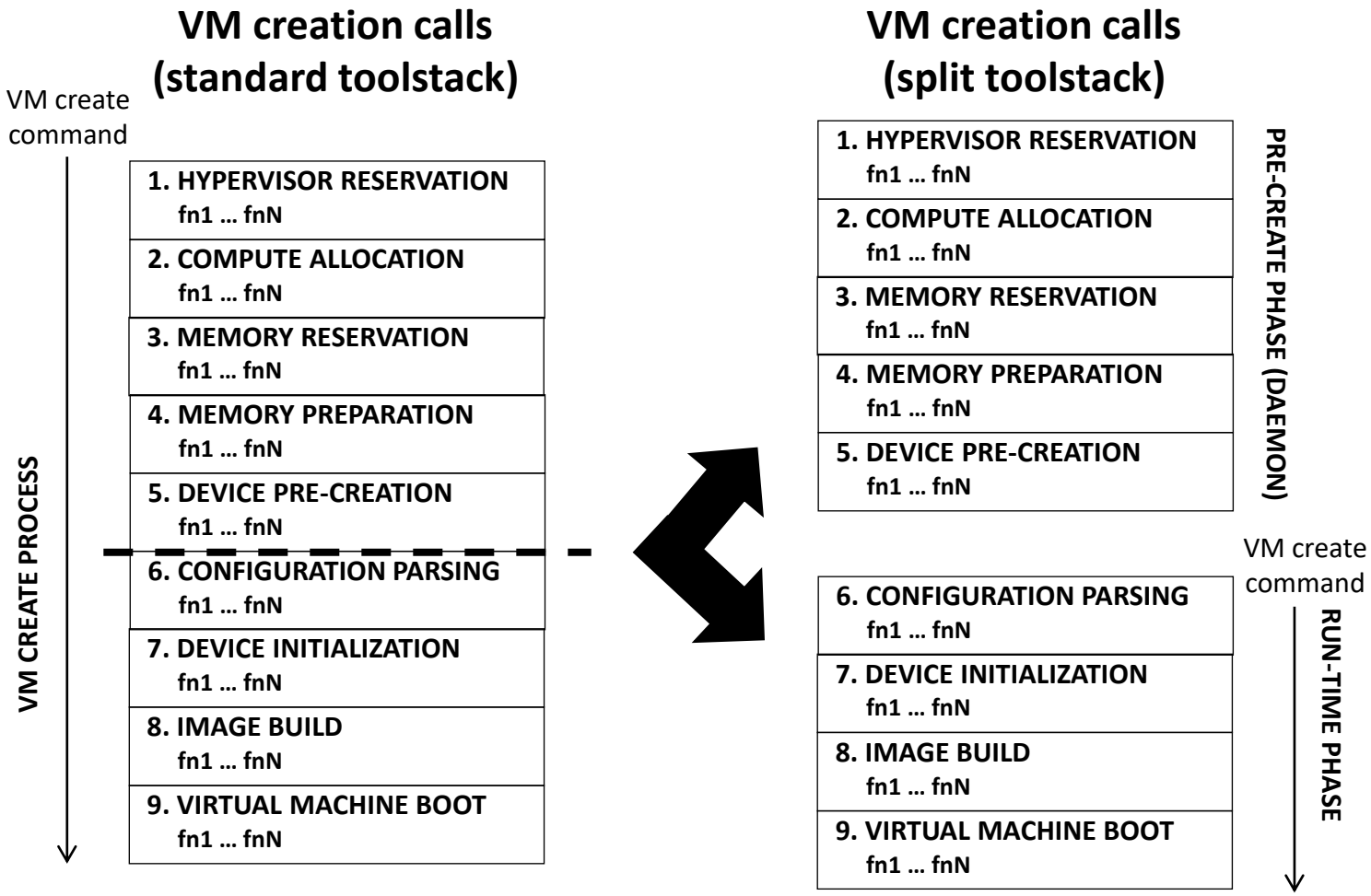
VM creation steps



VM creation steps



VM creation steps



Xenstore is used to:

- Store data
- Communicate between guests
- Synchronization

NoXS: Xen without the Xenstore

Xenstore is used to:

- Store data
- Communicate between guests
- Synchronization

NoXS

NoXS: Xen without the Xenstore

Xenstore is used to:

- Store data
- Communicate between guests
- Synchronization

NoXS

Shared memory

NoXS: Xen without the Xenstore

Xenstore is used to:

- Store data
- Communicate between guests
- Synchronization

NoXS

Shared memory

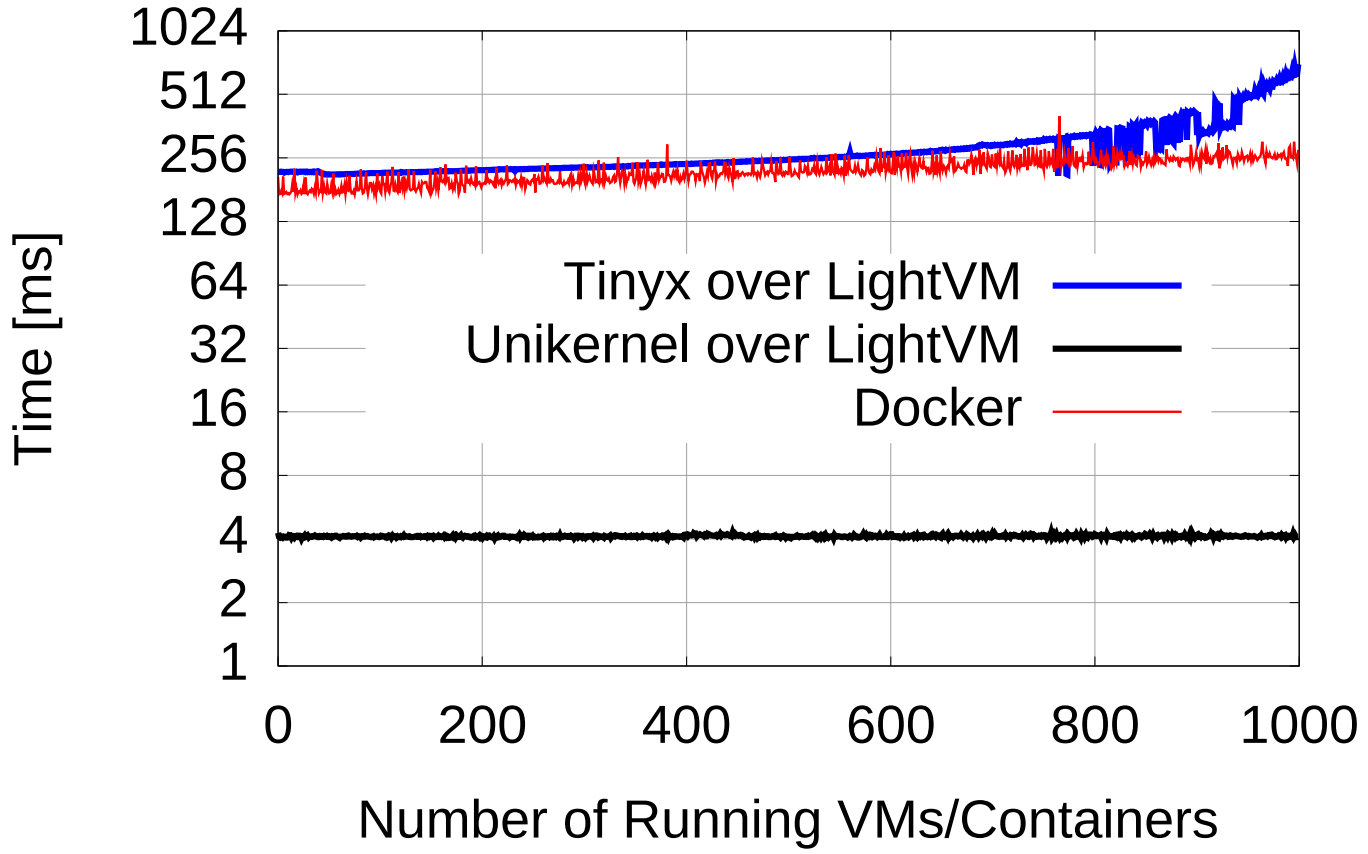
Event channels

Evaluation

Instantiation – Unikernels vs. Tinyx

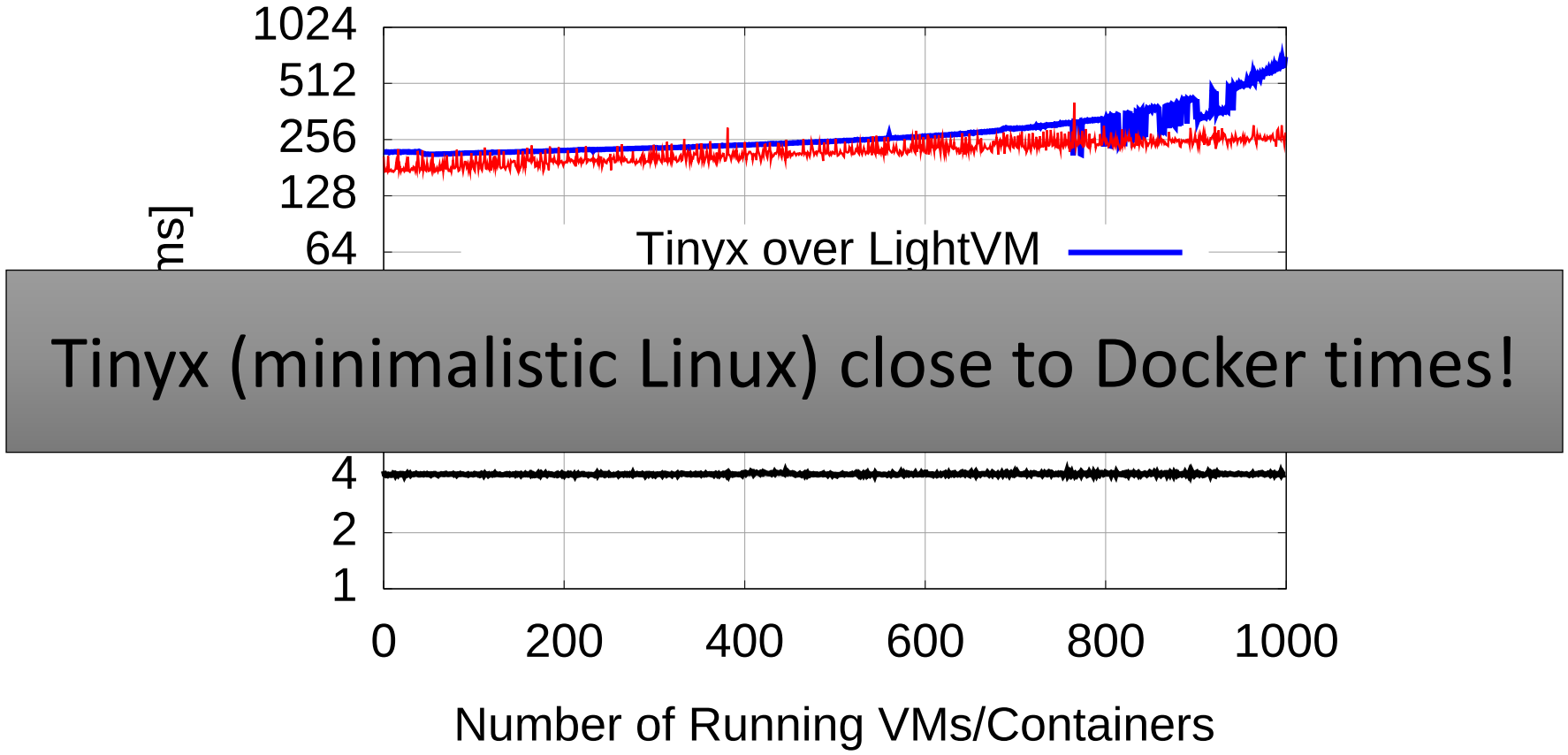
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Instantiation – Unikernels vs. Tinyx



Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Instantiation – Unikernels vs. Tinyx



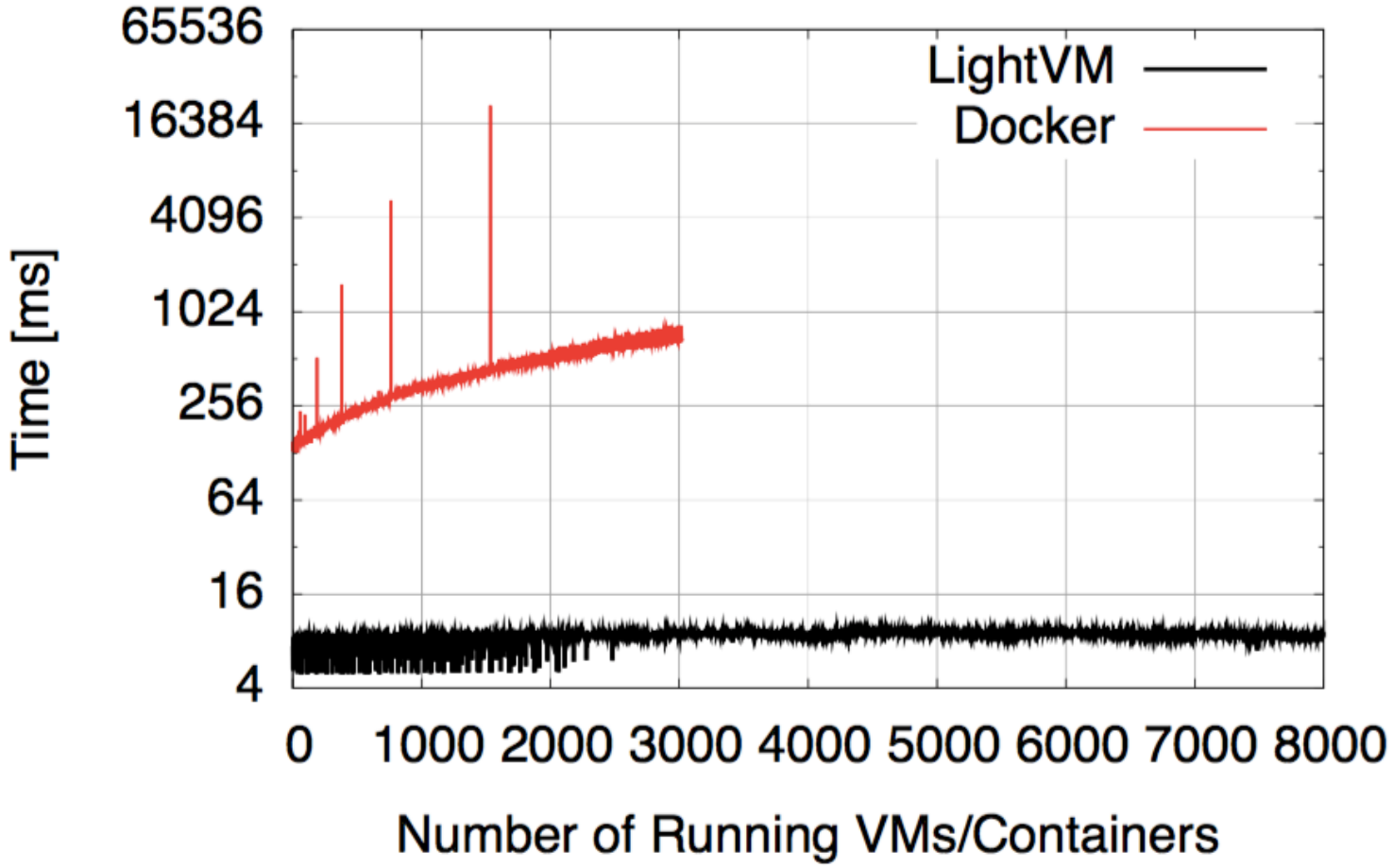
Tinyx (minimalistic Linux) close to Docker times!

Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Instantiation – High Density (noop unikernel)

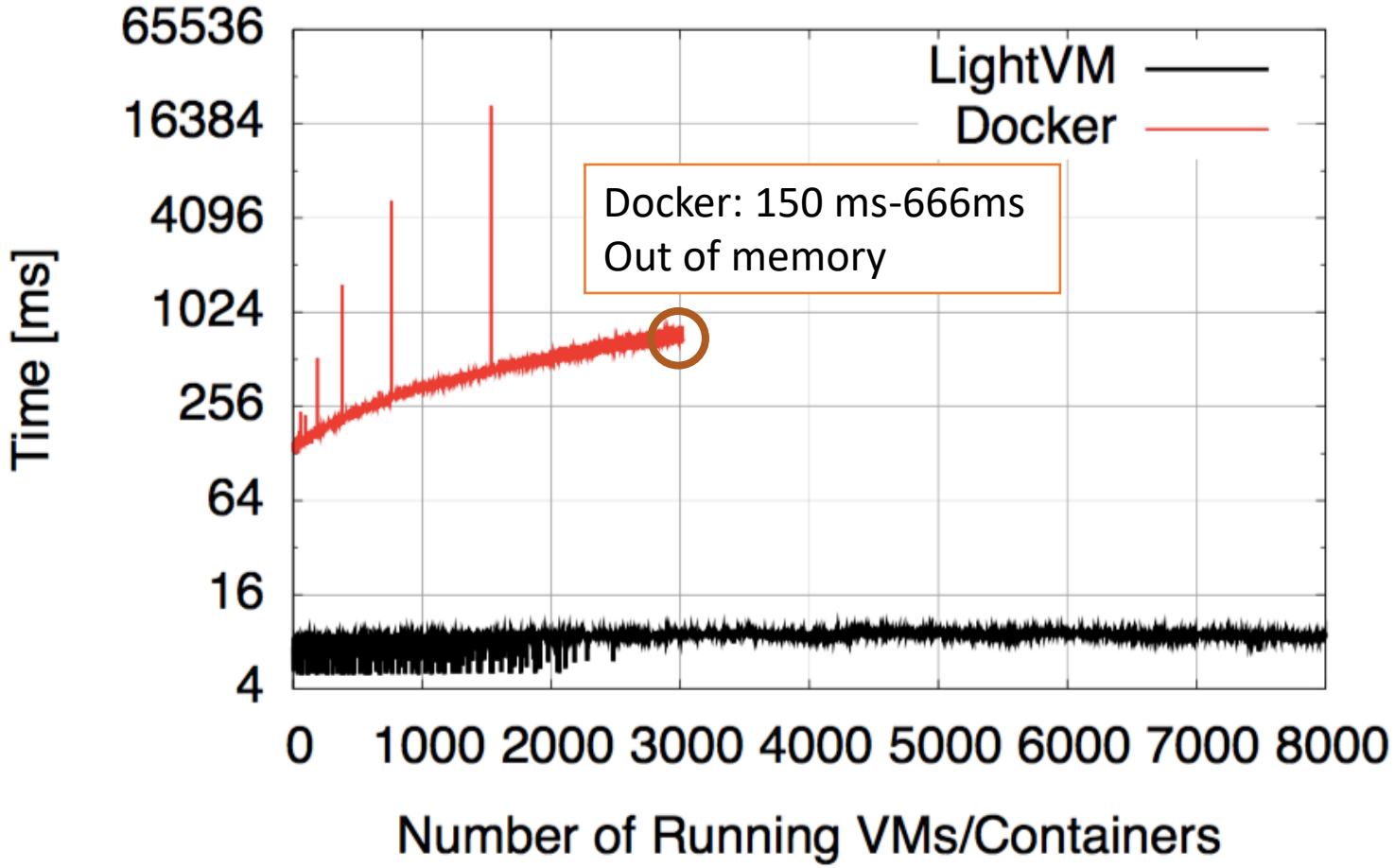
Server: 4 x AMD Opteron 6376 CPU@2.3GHz (64 cores total), 128GB DDR3 RAM, Xen/Linux versions 4.8

Instantiation – High Density (noop unikernel)



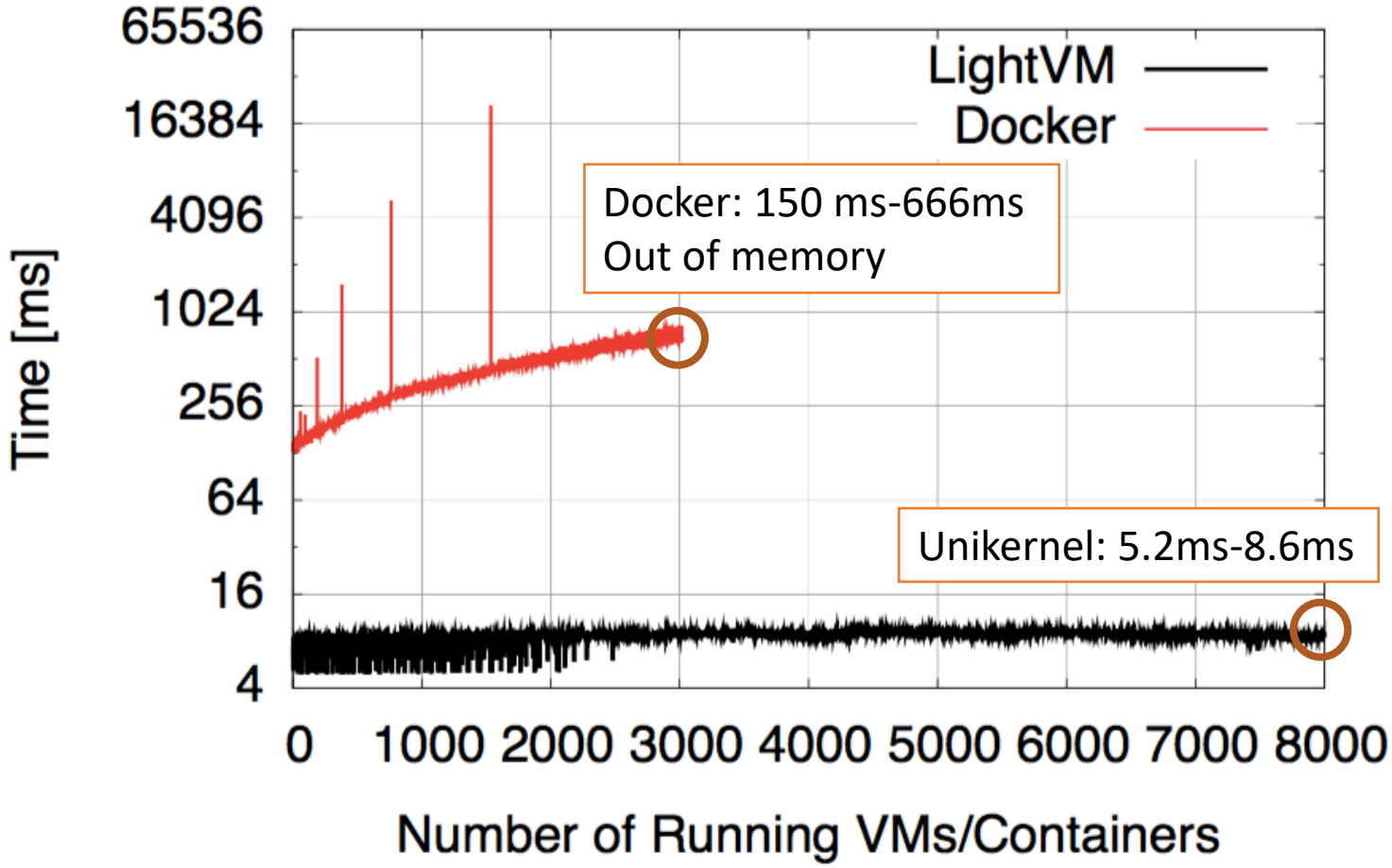
Server: 4 x AMD Opteron 6376 CPU@2.3GHz (64 cores total), 128GB DDR3 RAM, Xen/Linux versions 4.8

Instantiation – High Density (noop unikernel)



Server: 4 x AMD Opteron 6376 CPU@2.3GHz (64 cores total), 128GB DDR3 RAM, Xen/Linux versions 4.8

Instantiation – High Density (noop unikernel)



Server: 4 x AMD Opteron 6376 CPU@2.3GHz (64 cores total), 128GB DDR3 RAM, Xen/Linux versions 4.8

LightVM: lightweight virtual machines

Virtual machines can provide

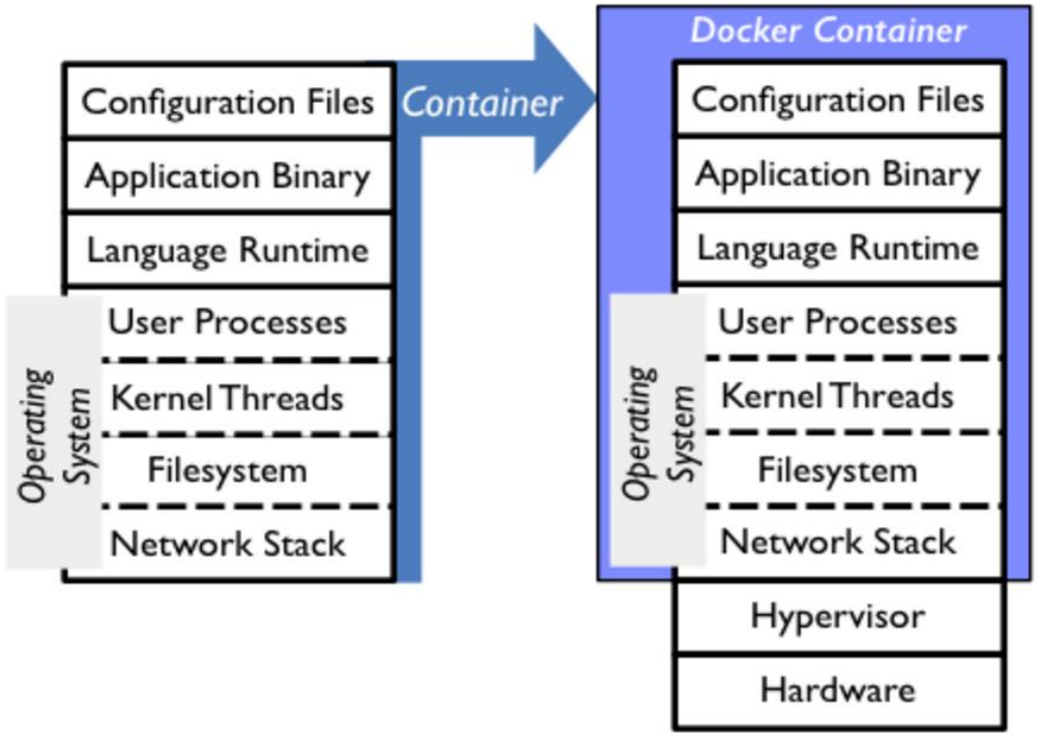
- strong isolation
- **and** be lightweight: 2-200 ms instantiation times, memory footprint of 10s of MBs.

Achieved through

- lightweight guests
- re-architected toolstack

App + OS + Container + Hypervisor

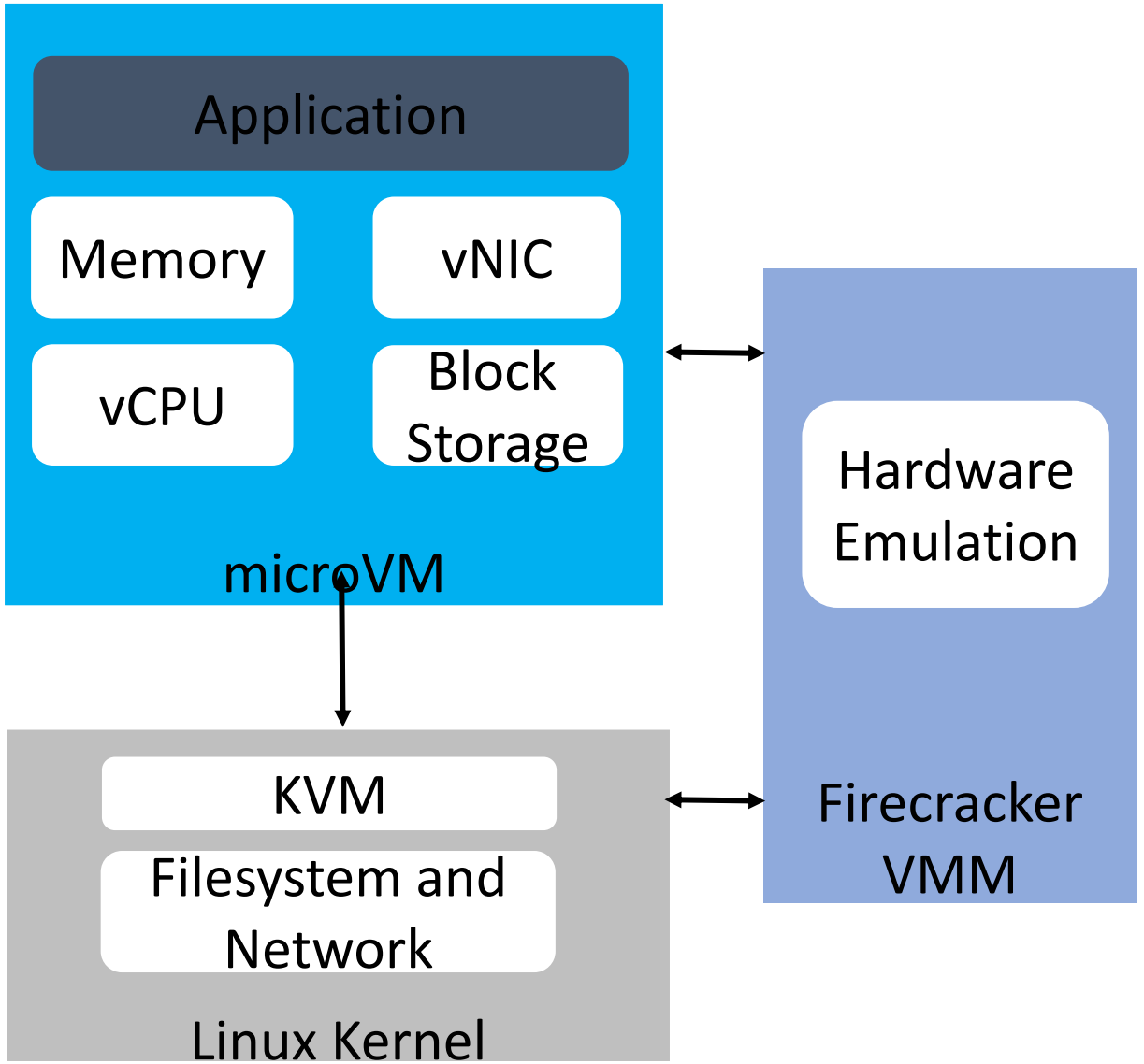
Typical cloud deployment scenario:



Variation: Firecracker: ditch the container

Strengthen isolation with MicroVM

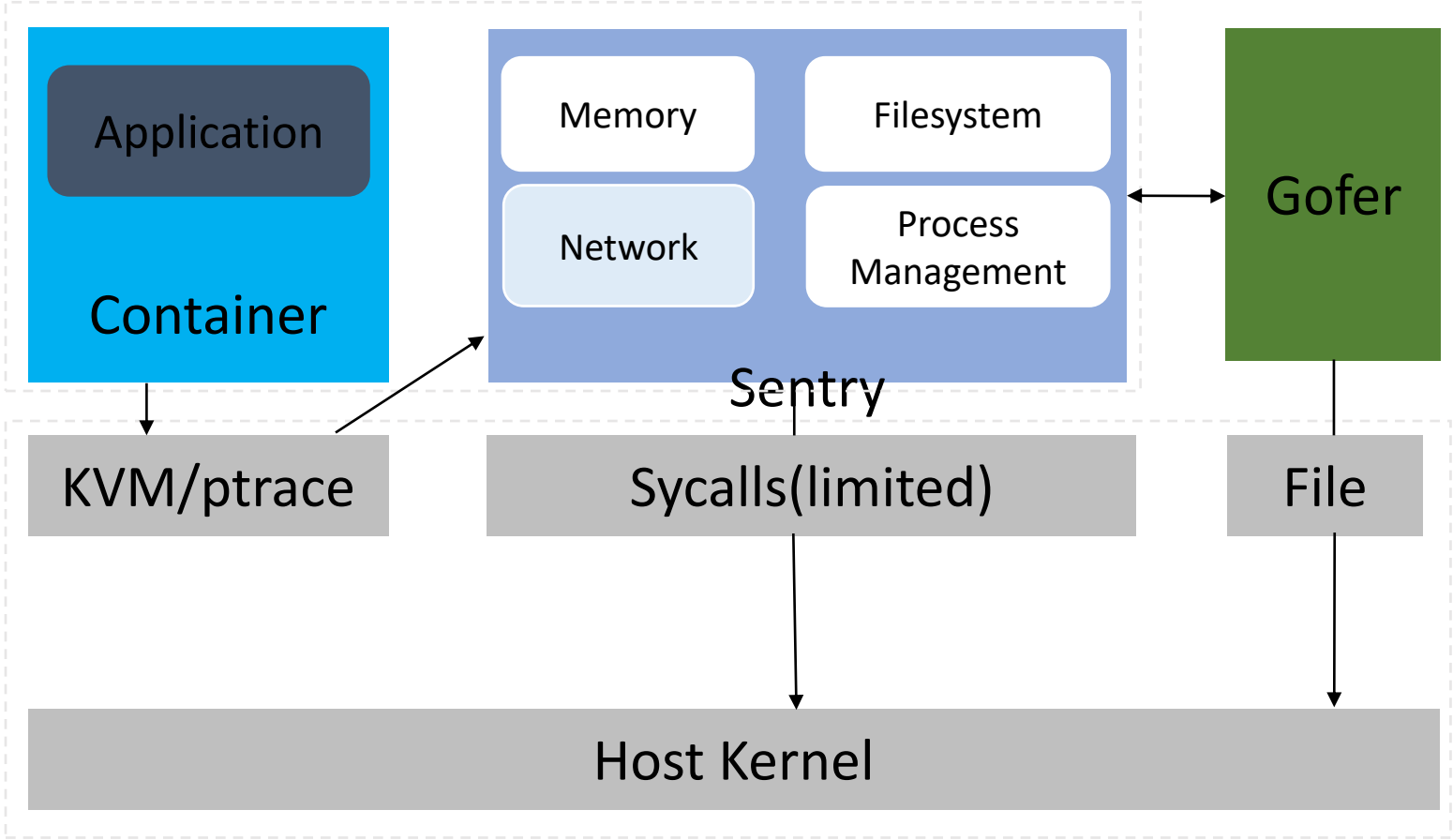
- Exports only 3 devices
- Limited system calls using SECure COMputing (seccomp) filters
- Written in Rust - type safe, memory safe, no unsafe C code etc.



Variation: gVisor: ditch the Hypervisor

Recover isolation with sandboxing

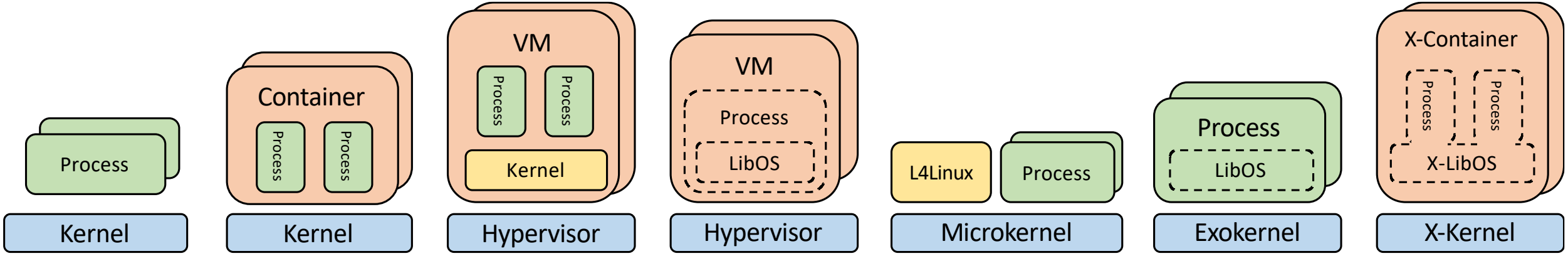
- Handles syscalls in sentry
- User space kernel written in Golang
- Sentry is heavily sandboxed
- Gofer for file access



Pros and Cons of the X-Container Architecture

	Container	gVisor	Clear-Container	LightVM	X-Container
Inter-container isolation	Poor	Good	Good	Good	Good
System call performance	Limited	Poor	Limited	Poor	Good
Portability	Good	Good	Limited	Good	Good
Compatibility	Good	Limited	Good	Good	Good
Intra-container isolation	Good	Good	Good	Good	Reduced
Memory efficiency	Good	Good	Limited	Limited	Limited
Spawning time	Short	Short	Moderate	Moderate	Moderate
Software licensing	Clean	Clean	Clean	Clean	Need discussion

Comparing Isolation Boundaries



Process | Container | Virtual Machine | Unikernel, Dune, EbbRT, OS^v | L4Linux (Microkernel) | Library OS (Exokernel) | X-Container

X-Containers Throughput Scalability