Orchestrating a brighter world

NEC

EU H2020 Superfluidity

LINUX FOUNDATION

Xen™

Unleashing the Power of Unikernels with Unikraft

Felipe Huici felipe.huici@neclab.eu
Systems and Machine Learning Group
NEC Laboratories GmbH, Heidelberg

Who am I?

Chief Researcher @

- NEC Laboratories in Heidelberg, Germany
- Systems and Machine Learning Group

The Container Craze

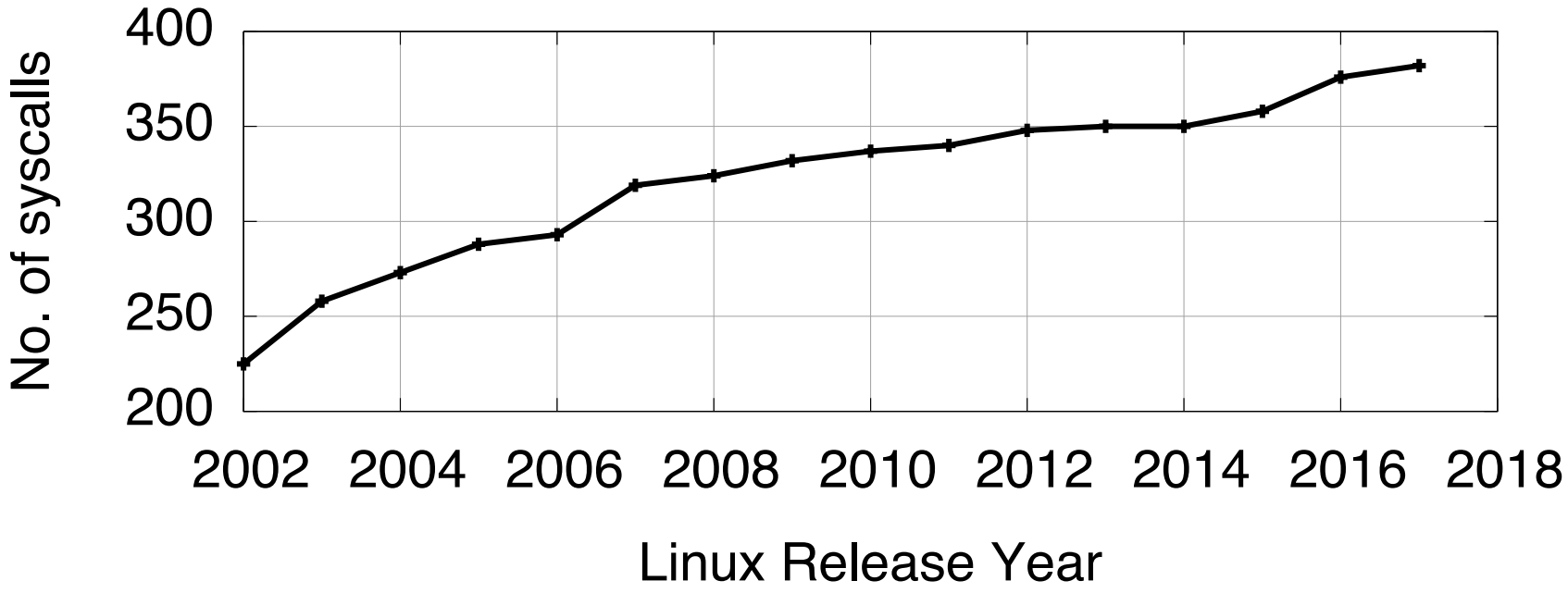
- "Google runs all services in containers"
- Container-as-a-Service services:
  - Amazon Lambda, EC2 Container Service
  - Google Container Engine
  - Azure Container service
  - *and so on…*

The Container Advantage

- Very fast instantiation times
  - 100s ms
- Small per-instance memory footprints
  - 10-100 MBs
- High density
  - 100-1000 instances on same host

The Unsafe Container

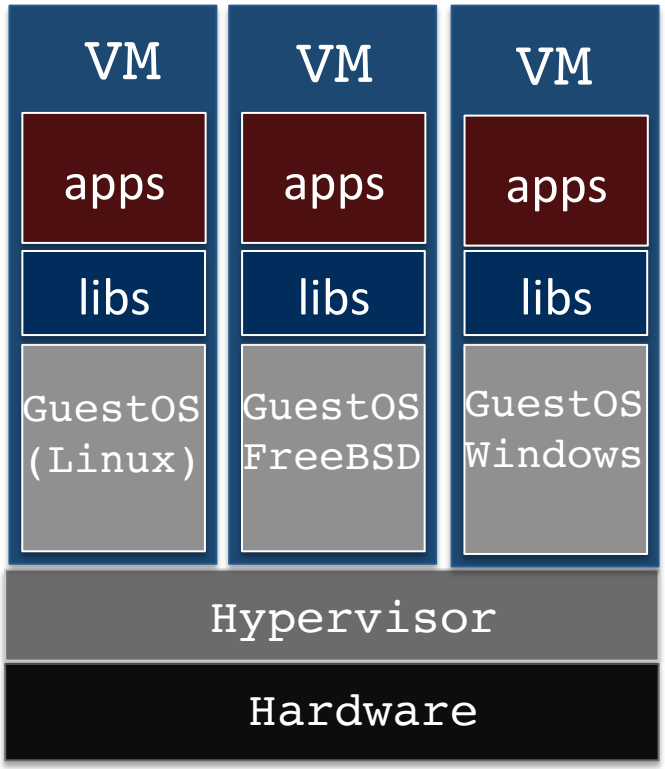
- Kernel API difficult to secure, lots of exploits
- An unsafe container affects all other containers on host
  - This includes DoS/exhaustion attacks (e.g., forkbombs, etc.)



Isolating Workloads

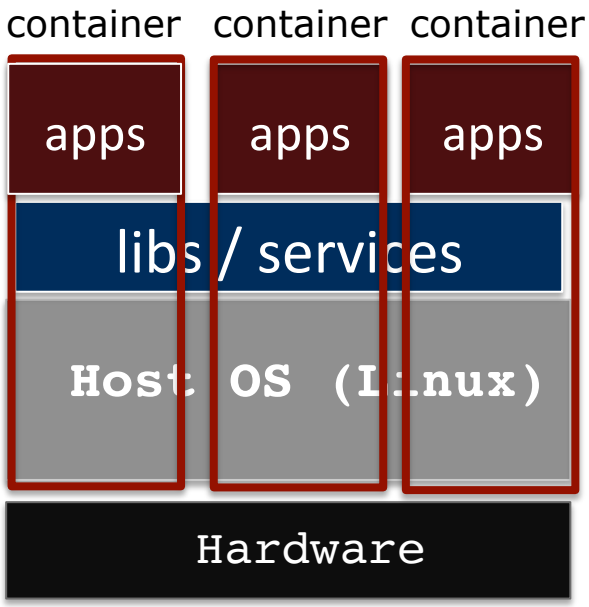
Pick Your Poison

Virtualization



vs.

Containers



✅ Strong isolation

❌ Heavy weight

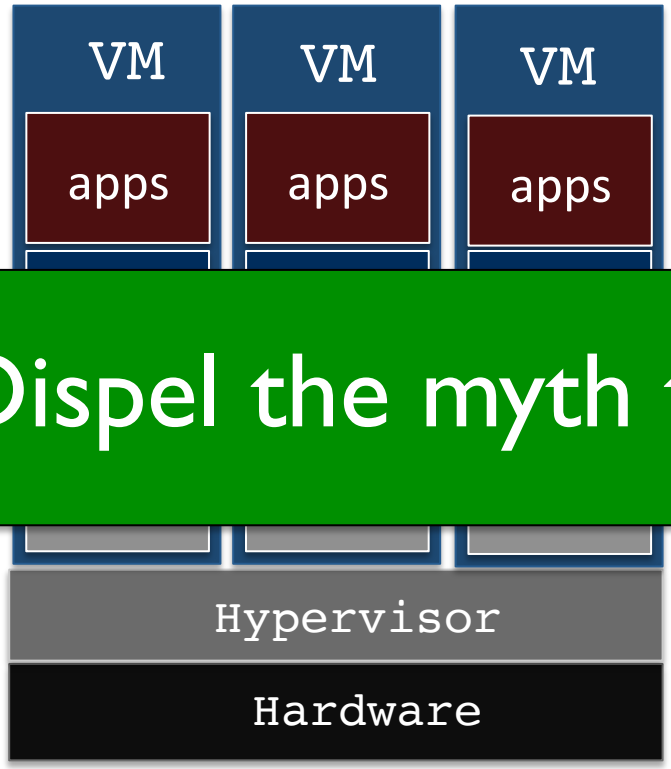
?

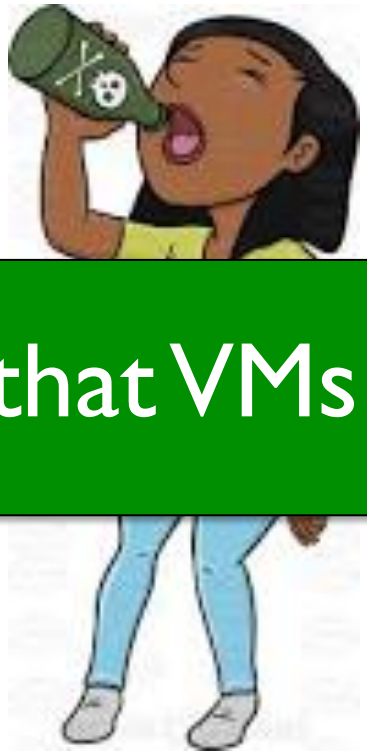
✅ Lightweight

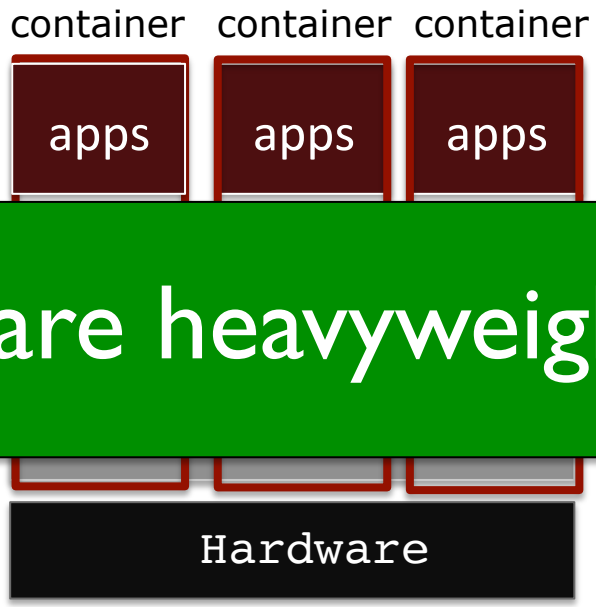
❌ Iffy isolation

Isolating Workloads

Virtualization vs. Containers







Dispel the myth that VMs are heavyweight

✅ Strong isolation

❌ Heavy weight

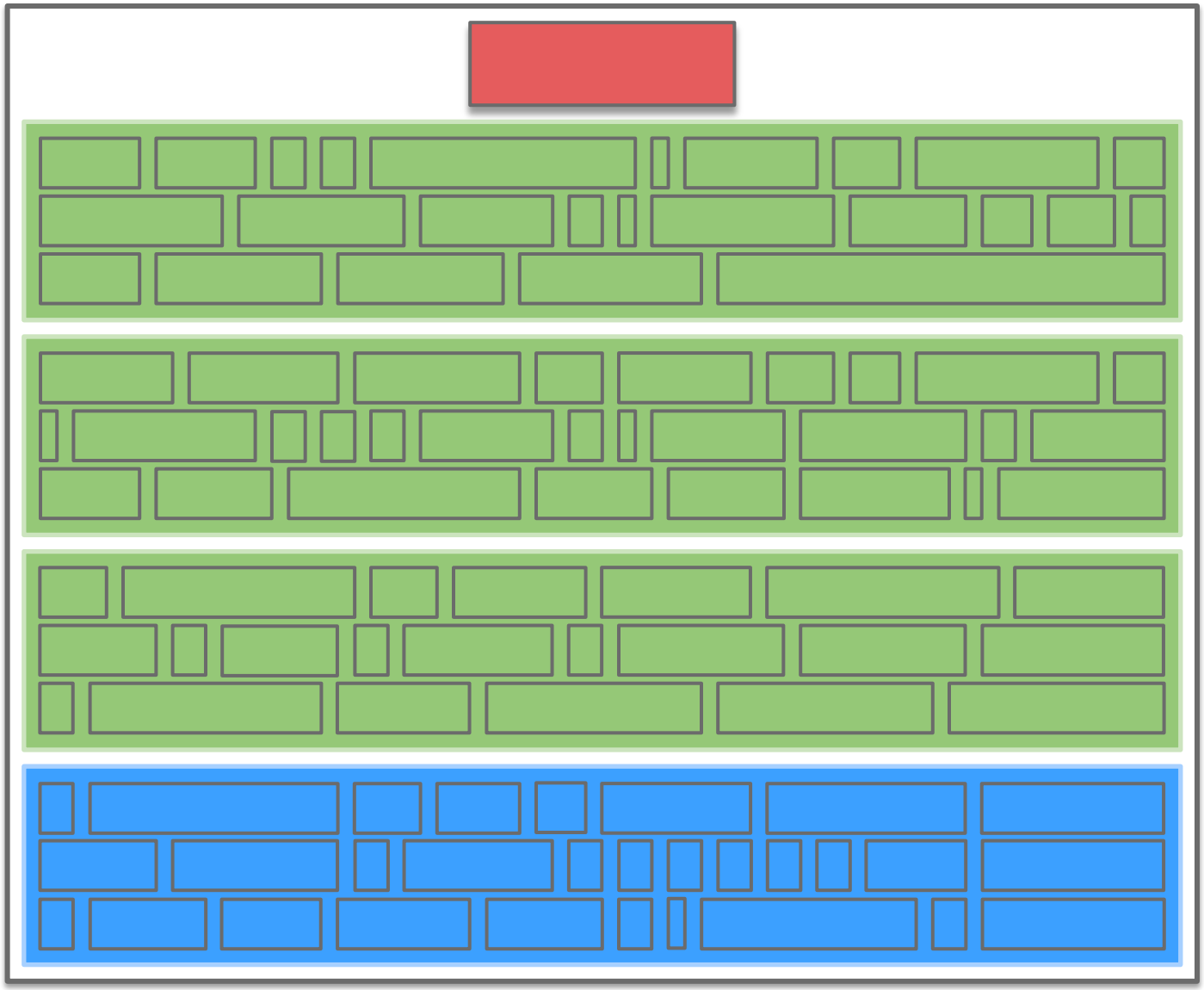
?

✅ Lightweight

❌ Iffy isolation

What's a Unikernel Then?

Standard VM: Application on Top of Distro



User Application(s)

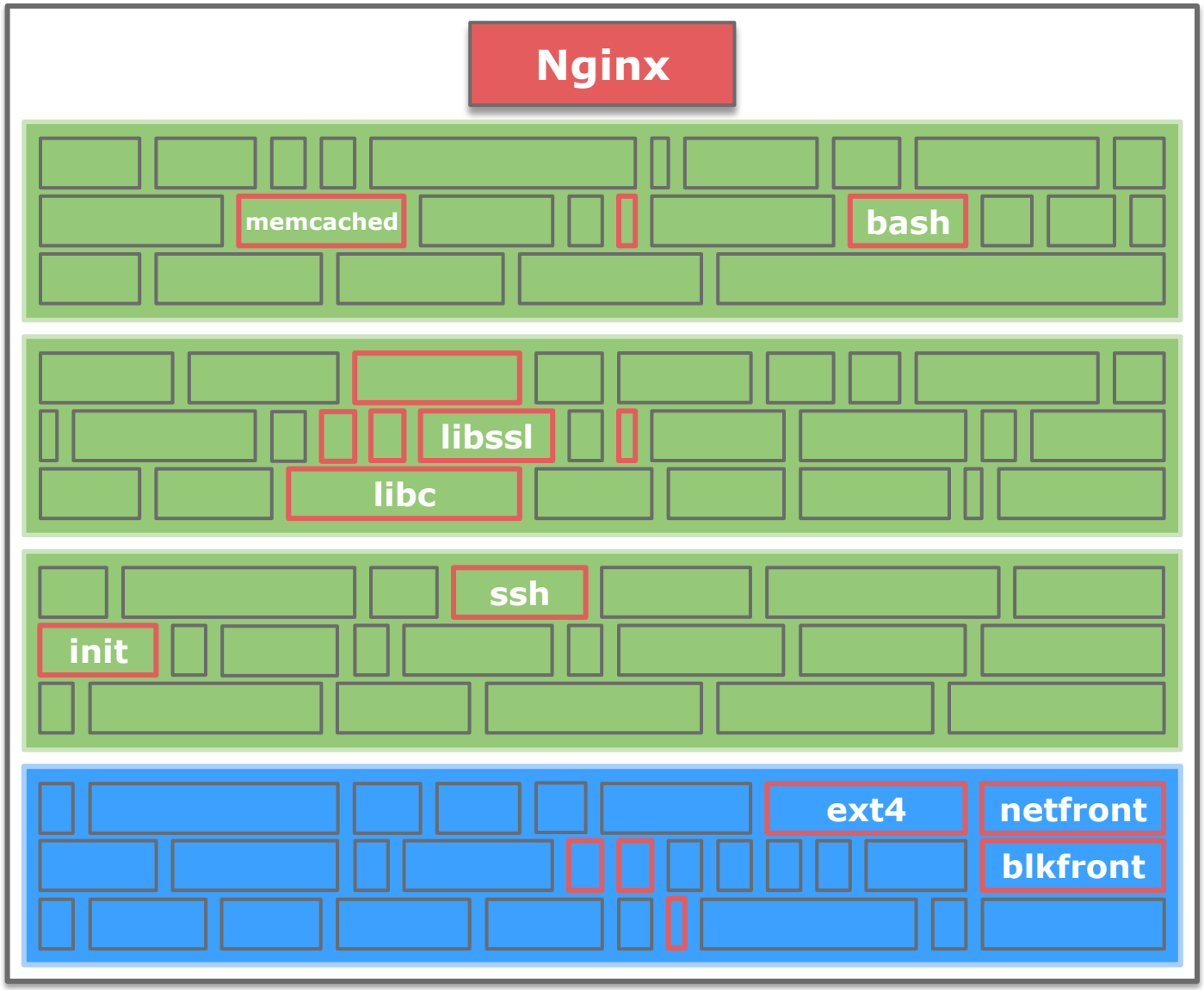
3rd Party Applications

Libraries

Services

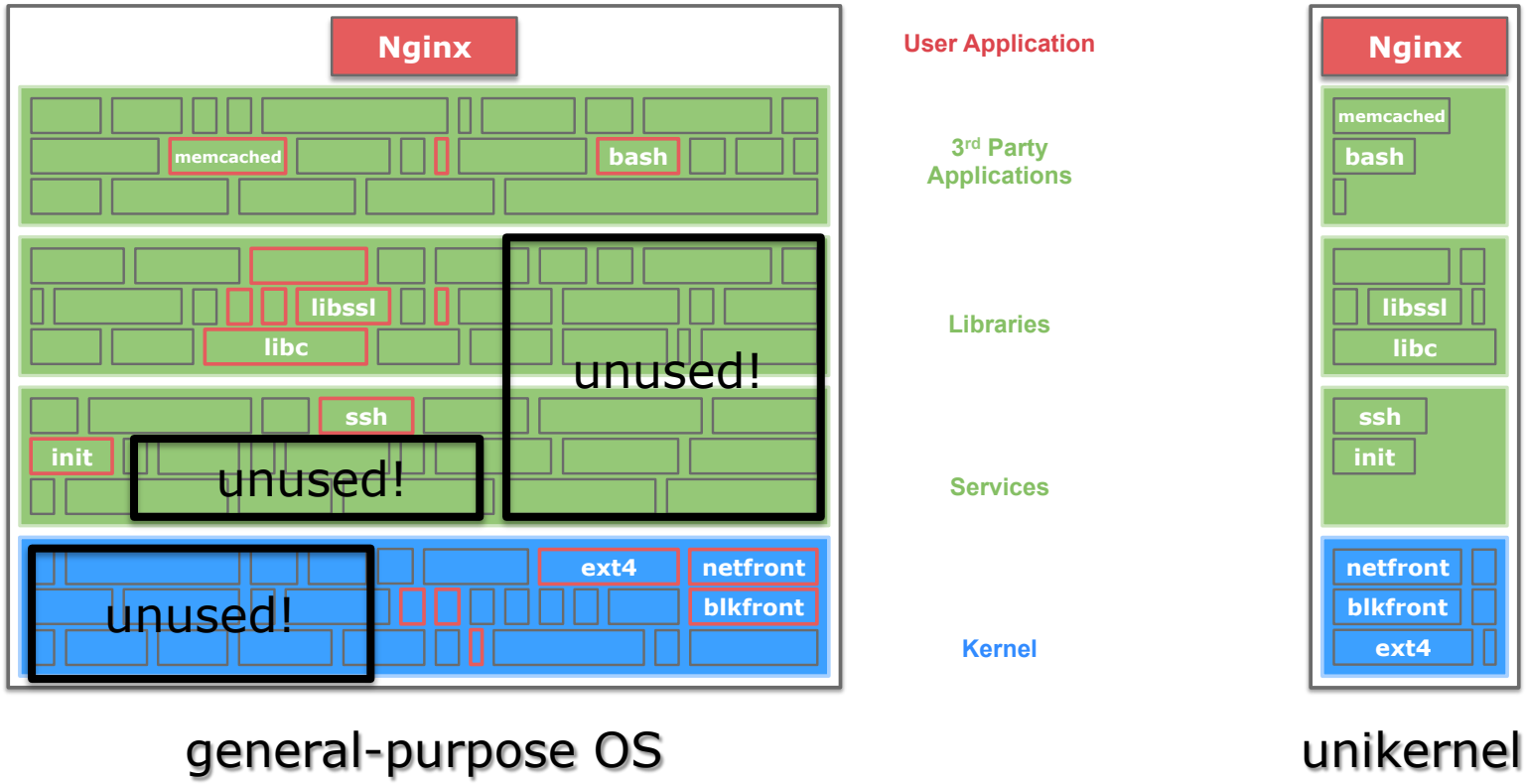
Kernel

Most of the VM not Used…



Specialization in Practice

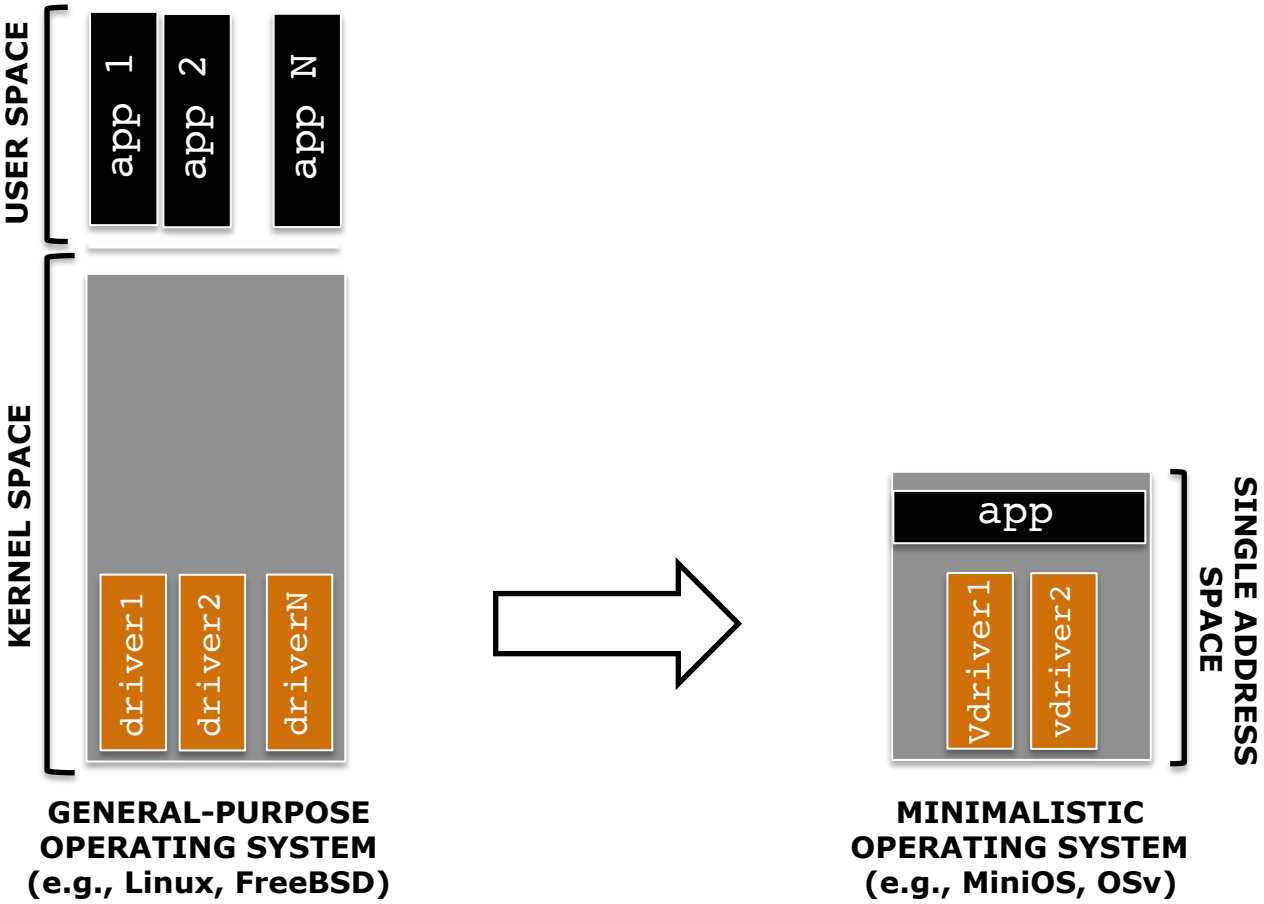
- Standard OS/VM/container: ***lots*** of unnecessary code = ***lots*** of overhead (and potentially bugs/security issues)!
- Unikernel: only what's needed is there



general-purpose OS

unikernel

Unikernels - A Few More Details



When do I want a Unikernel?

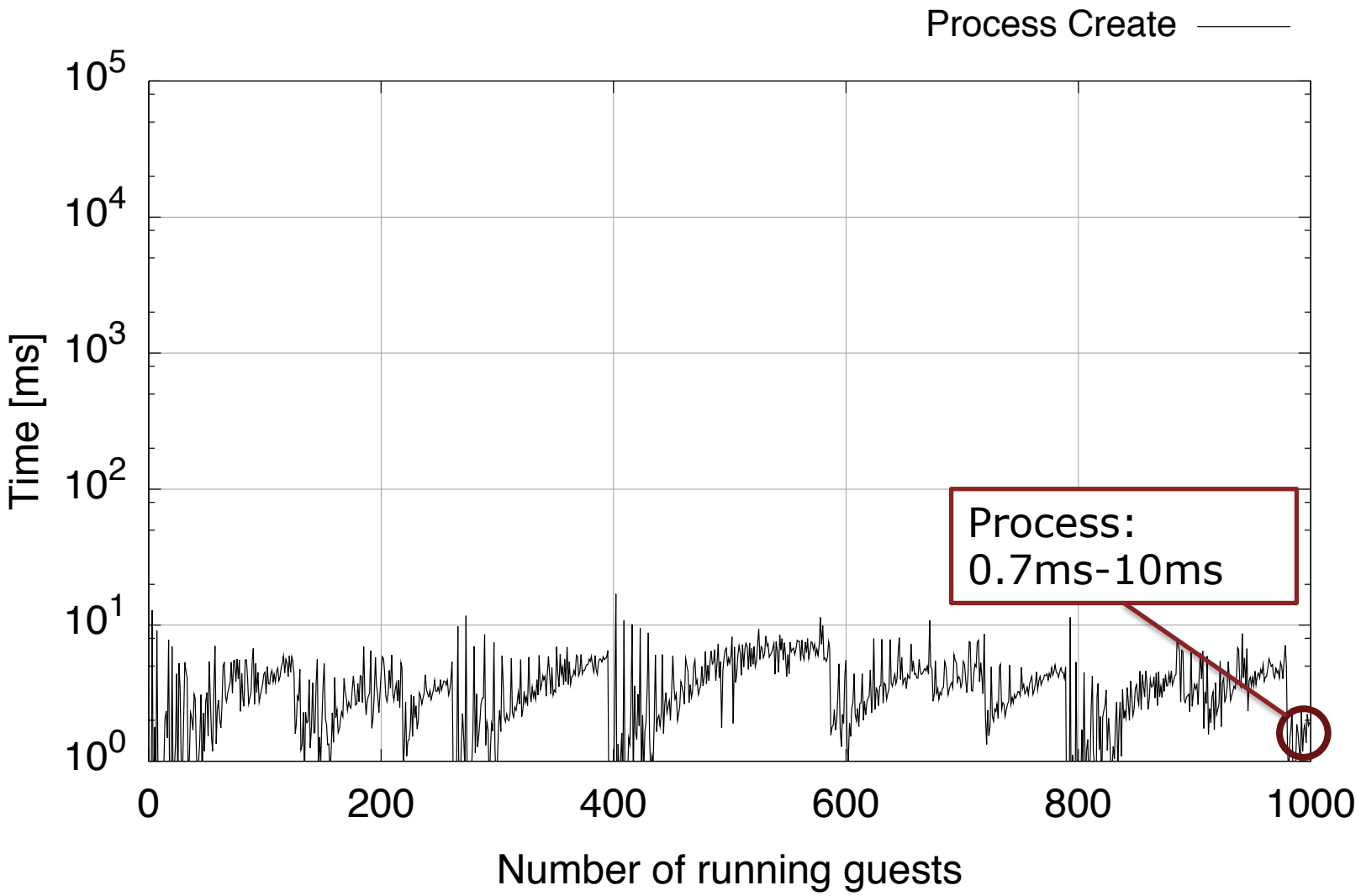
- Unikernels are applicable to many domains!

Some Unikernel Numbers…

My VM is Lighter (and Safer) than your Container
SOSP 2017

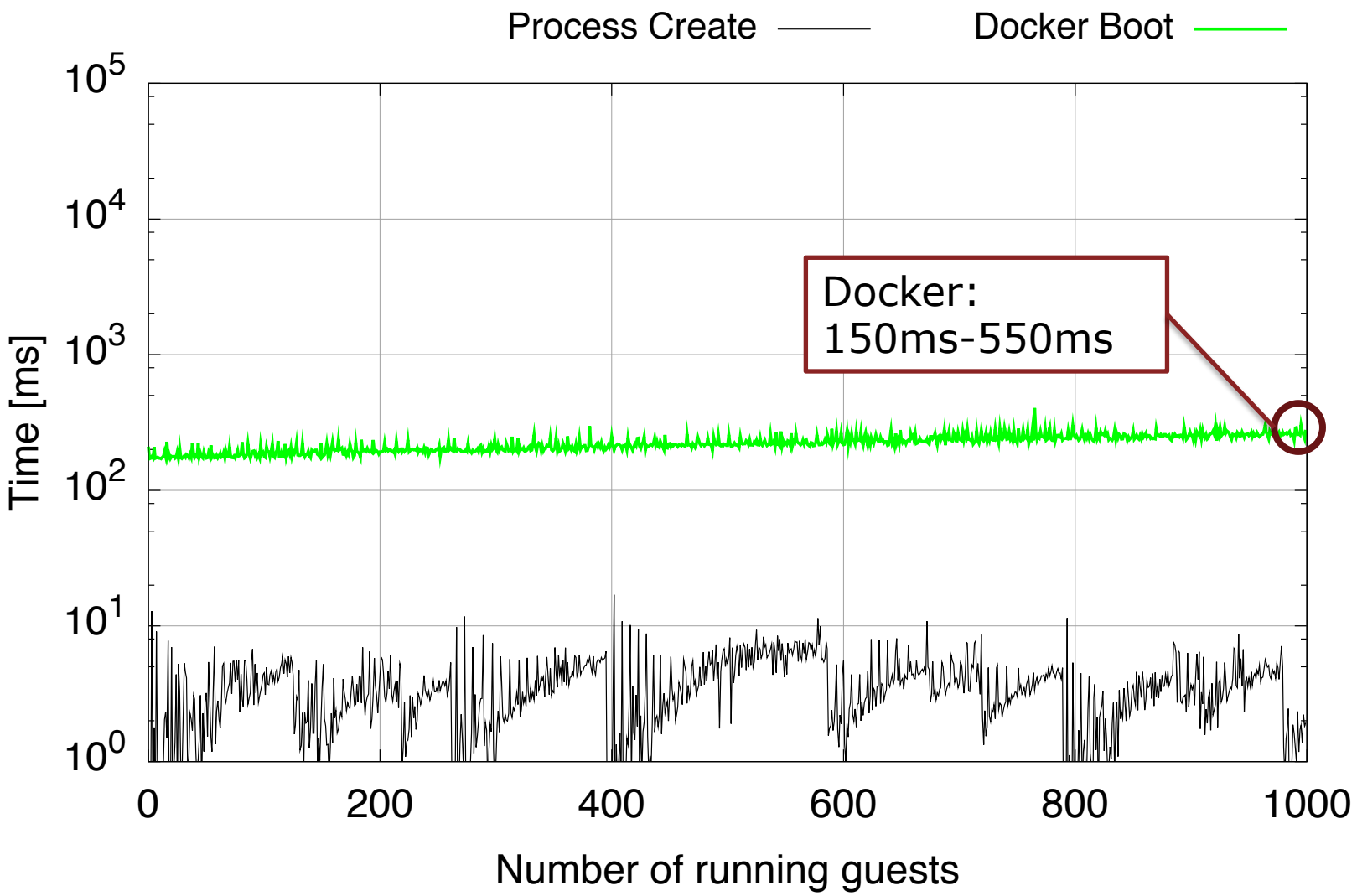
In Numbers: Instantiation Times

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



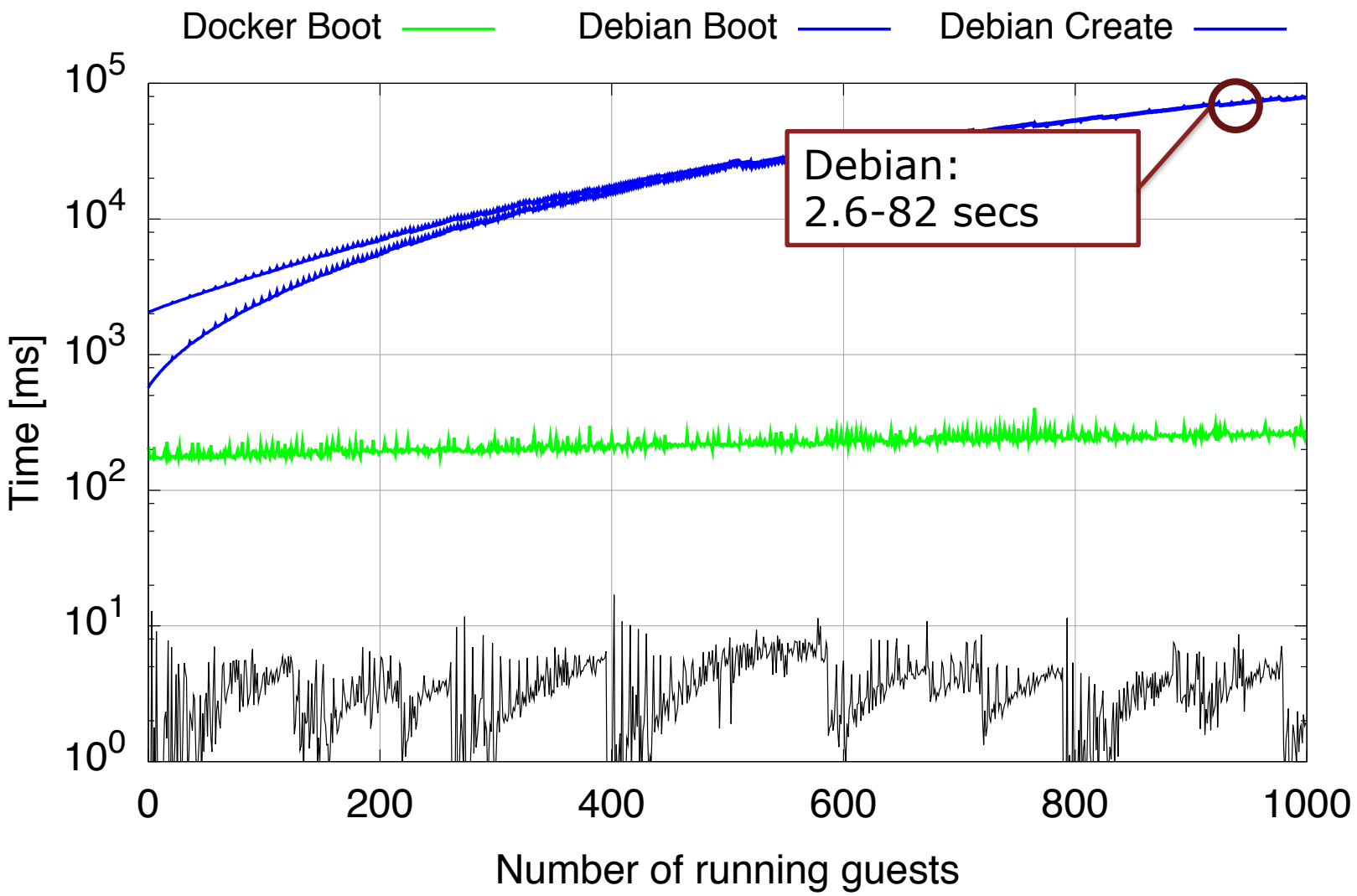
In Numbers: Instantiation Times

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



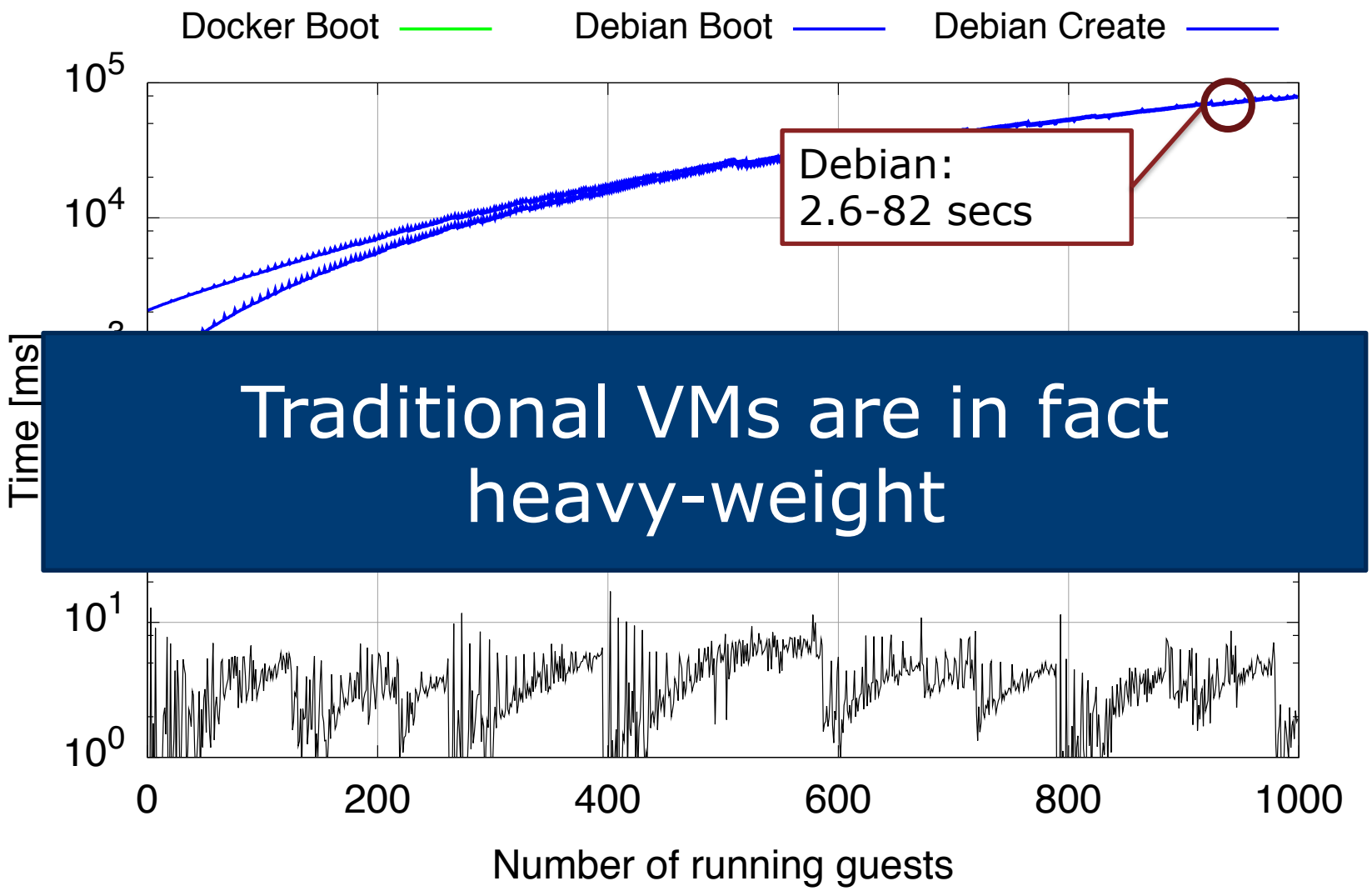
In Numbers: Instantiation Times

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



In Numbers: Instantiation Times

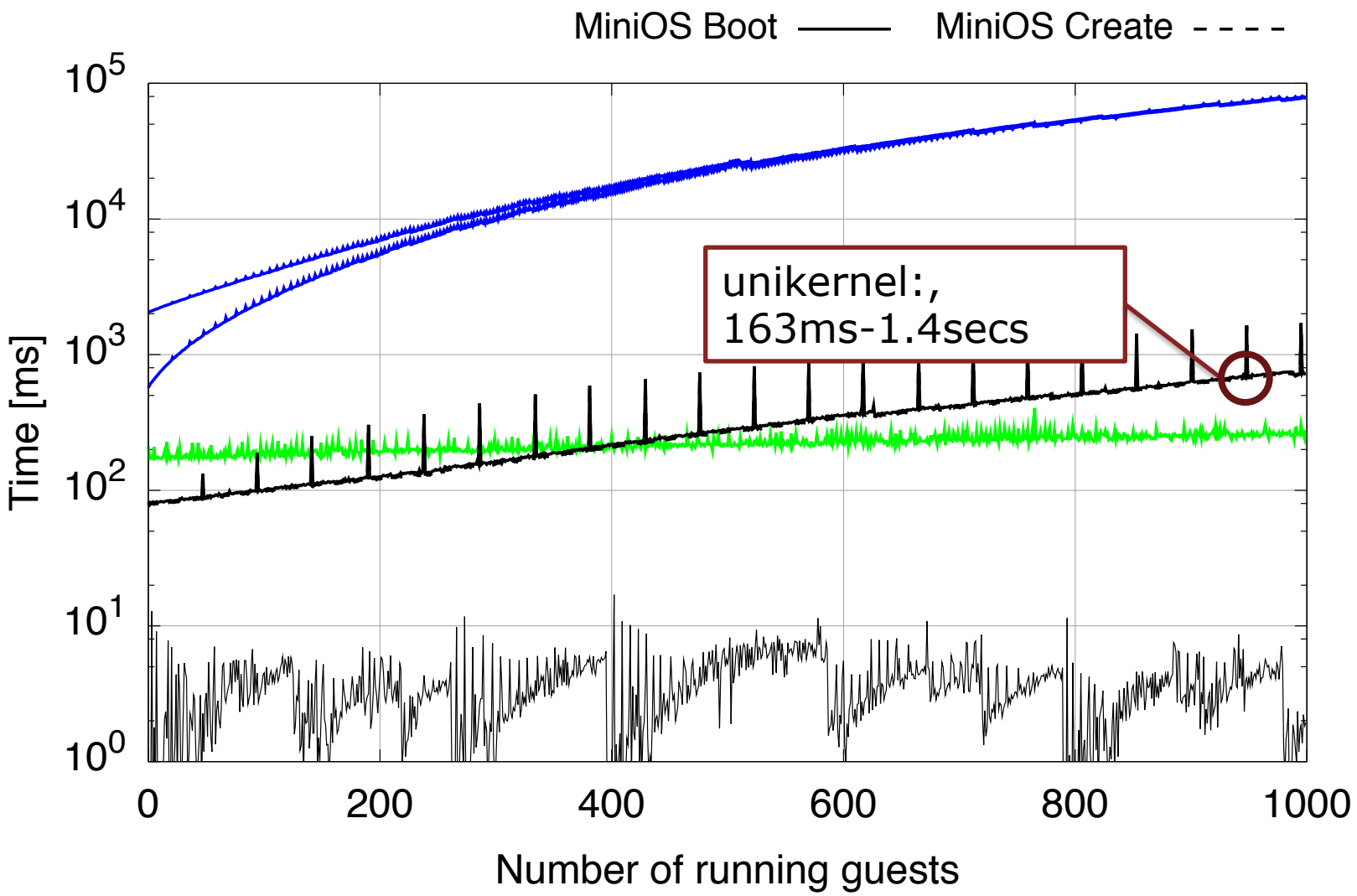
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8



Traditional VMs are in fact heavy-weight

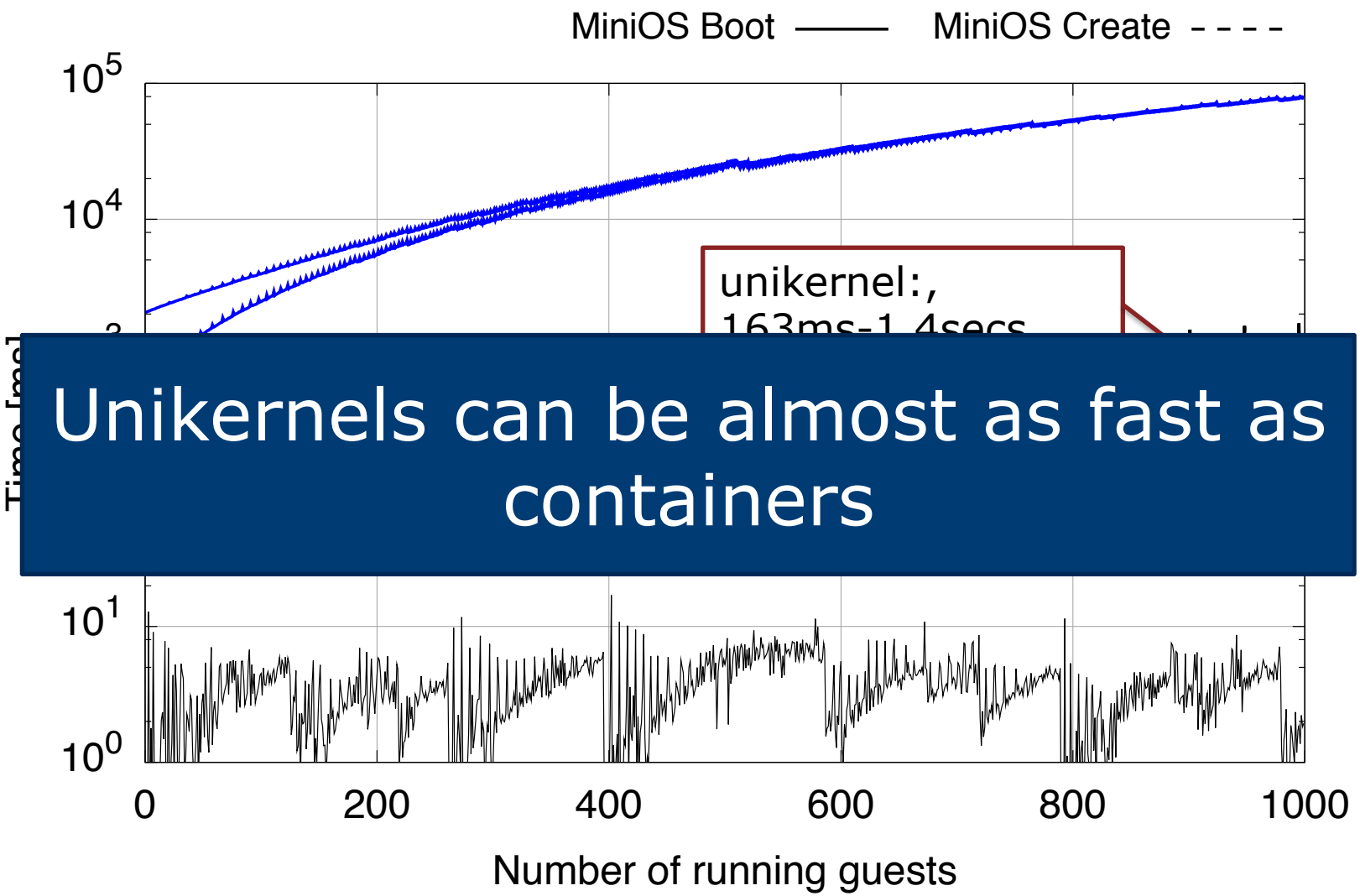
In Numbers: Instantiation Times

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



In Numbers: Instantiation Times

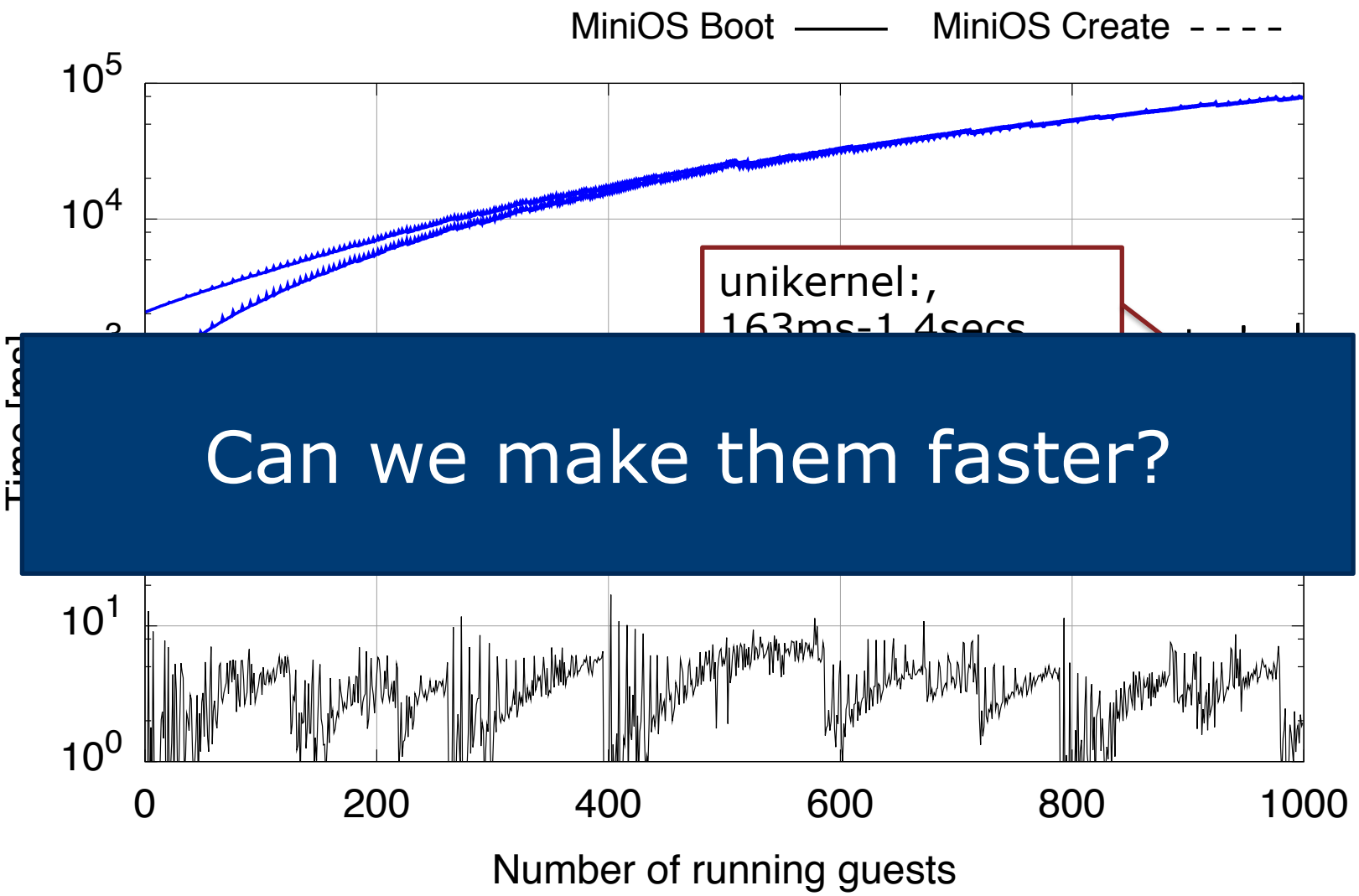
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8



Unikernels can be almost as fast as containers

In Numbers: Instantiation Times

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

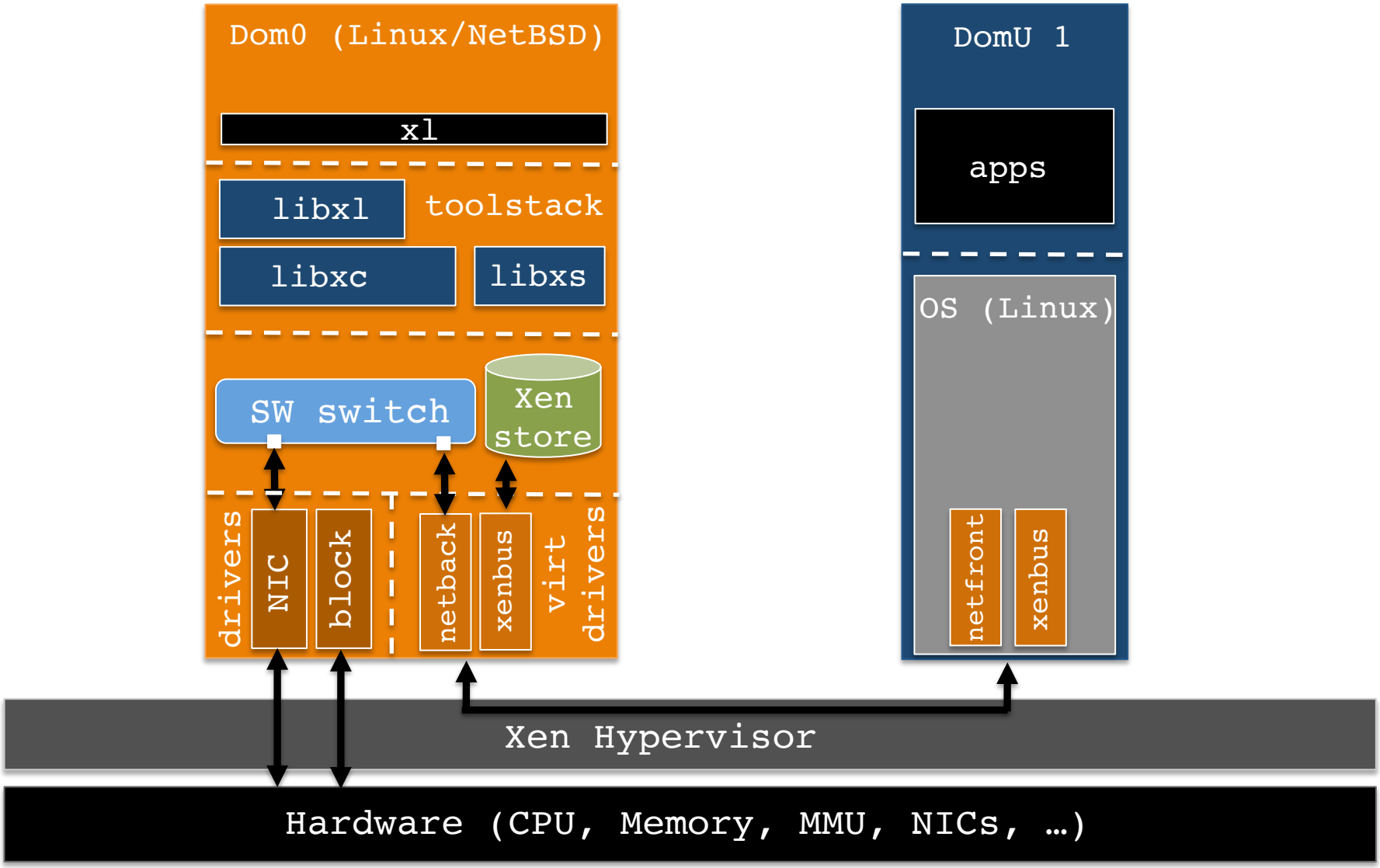


Can we make them faster?

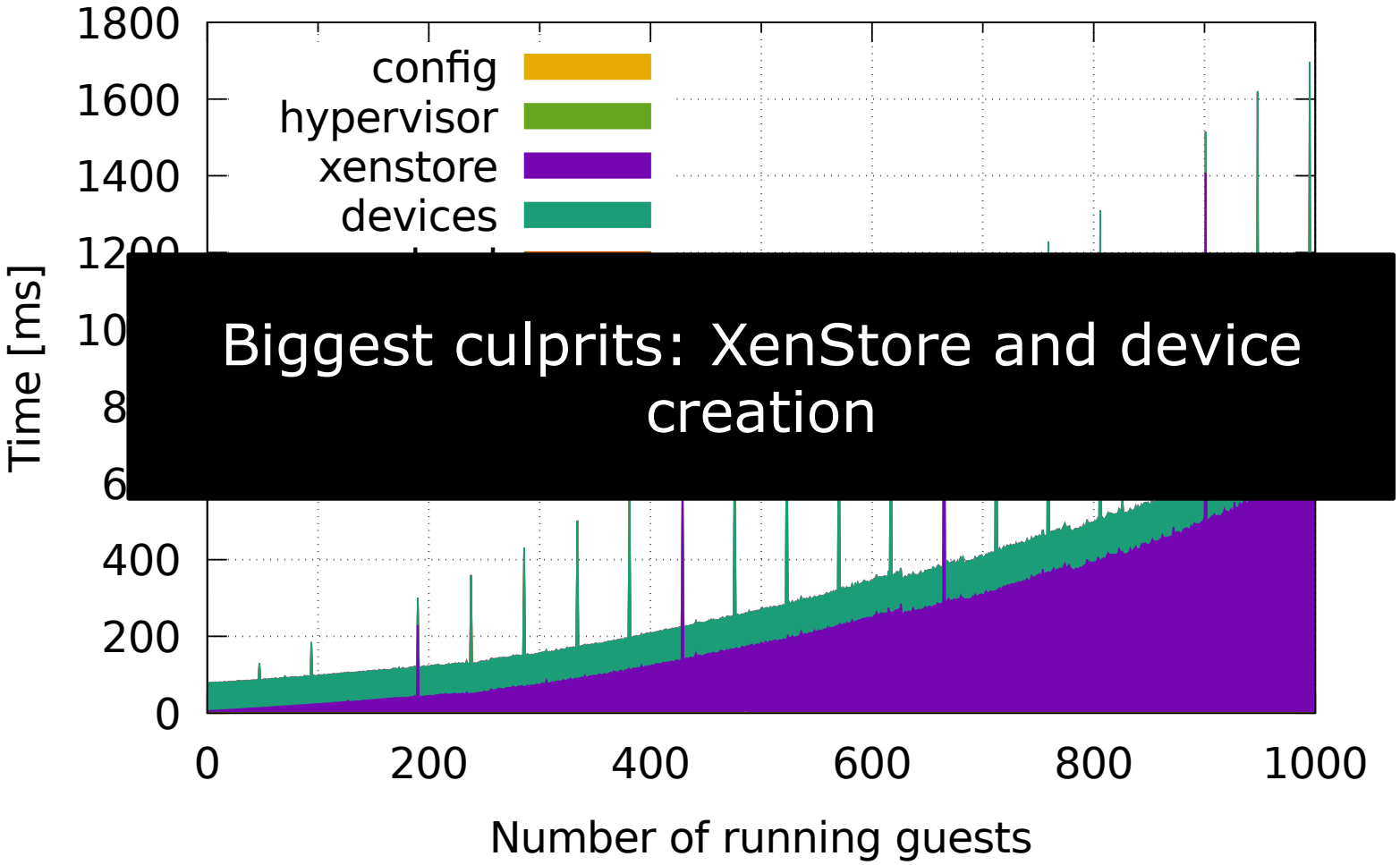
LightVM

A Lightweight Virtualization System

A Quick Xen Primer

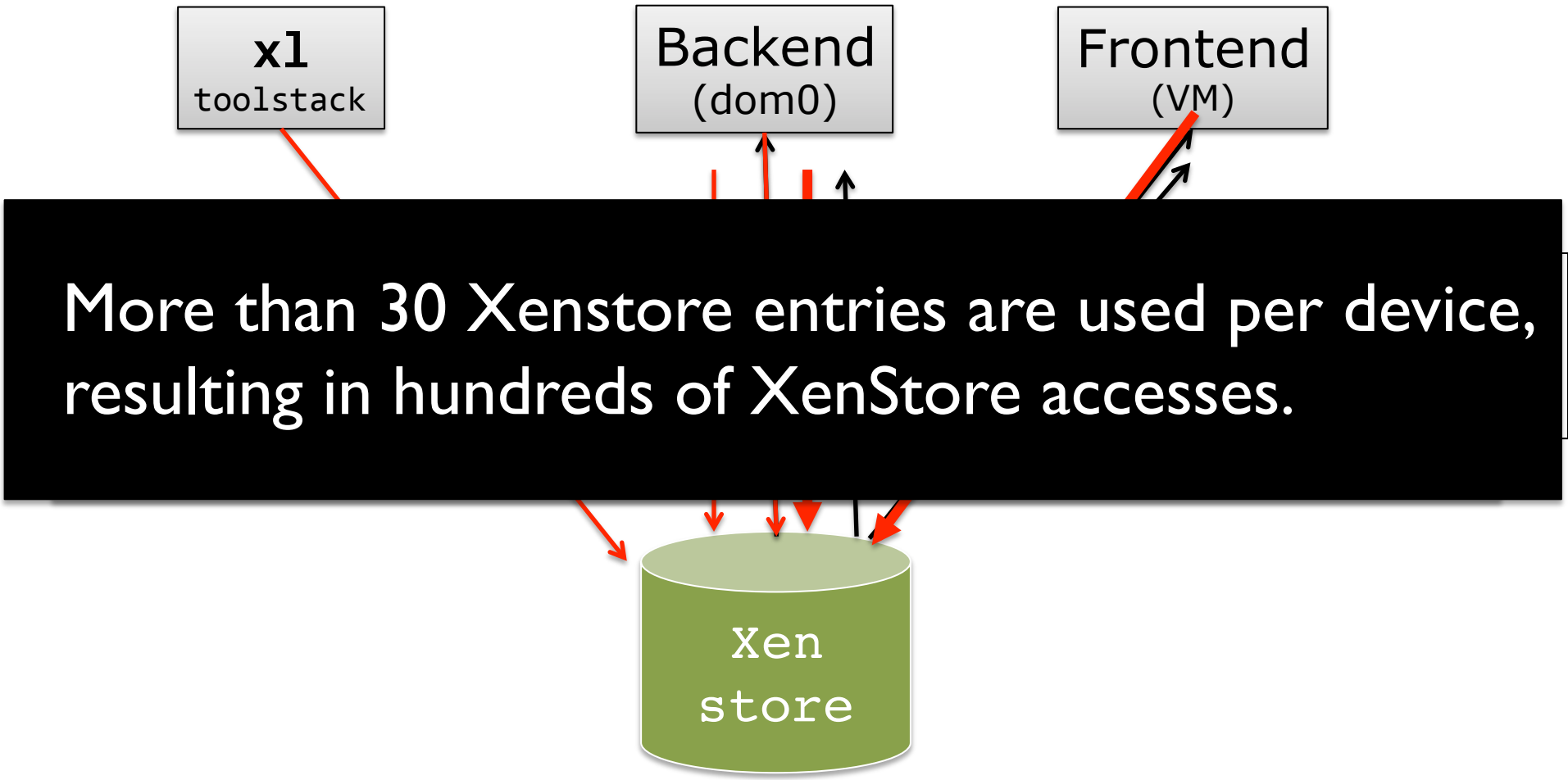


Creation Times – Breakdown (unikernel)

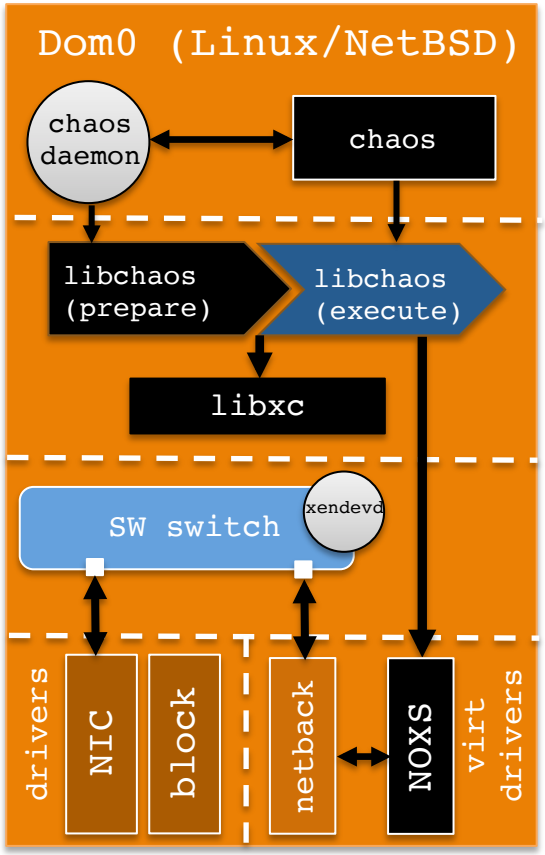


* *Note: Spikes in graph due to XenStore's log rotation*

What's Wrong with the XenStore?



LightVM Architecture

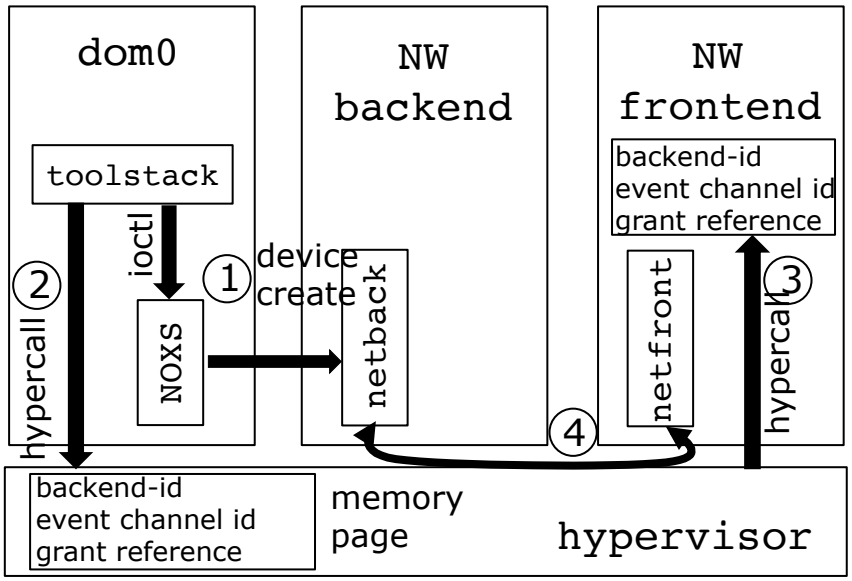


1. **Chaos** – specialized toolstack (xl replacement)
2. **Noxs** - no XenStore
3. **Split toolstack**
4. **Xendevd** – optimized hotplug script

1. Chaos - Specialized Toolstack

- Complete re-write of toolstack for paravirtualized guests
  - Replaces xl → chaos
  - Replaces libxl → libchaos
- Includes framework for easily plugging in different elements of a toolstack (e.g., use the XenStore or not)

2. NoXS – Xen without a XenStore



3. Split Toolstack

Insight: a significant portion of the code does not need to be executed at VM creation time

- Functionality common to all VMs (e.g., mem alloc) can be pre-executed

Split toolstack into two:

- **Pre-create phase:** run by a daemon, periodically
- **Run-time phase:** run when VM creation commands (and others) are issued

4. Xendevd – Hotplug Script

Xen uses either a bash script or udevd to add virtual interfaces to software switch

- Slow: 10s of milliseconds, slows creation/boot times

Replace with xendevd, binary daemon

- Listens to udev events from netback, execute pre-defined setup (e.g., add vif to OVS)

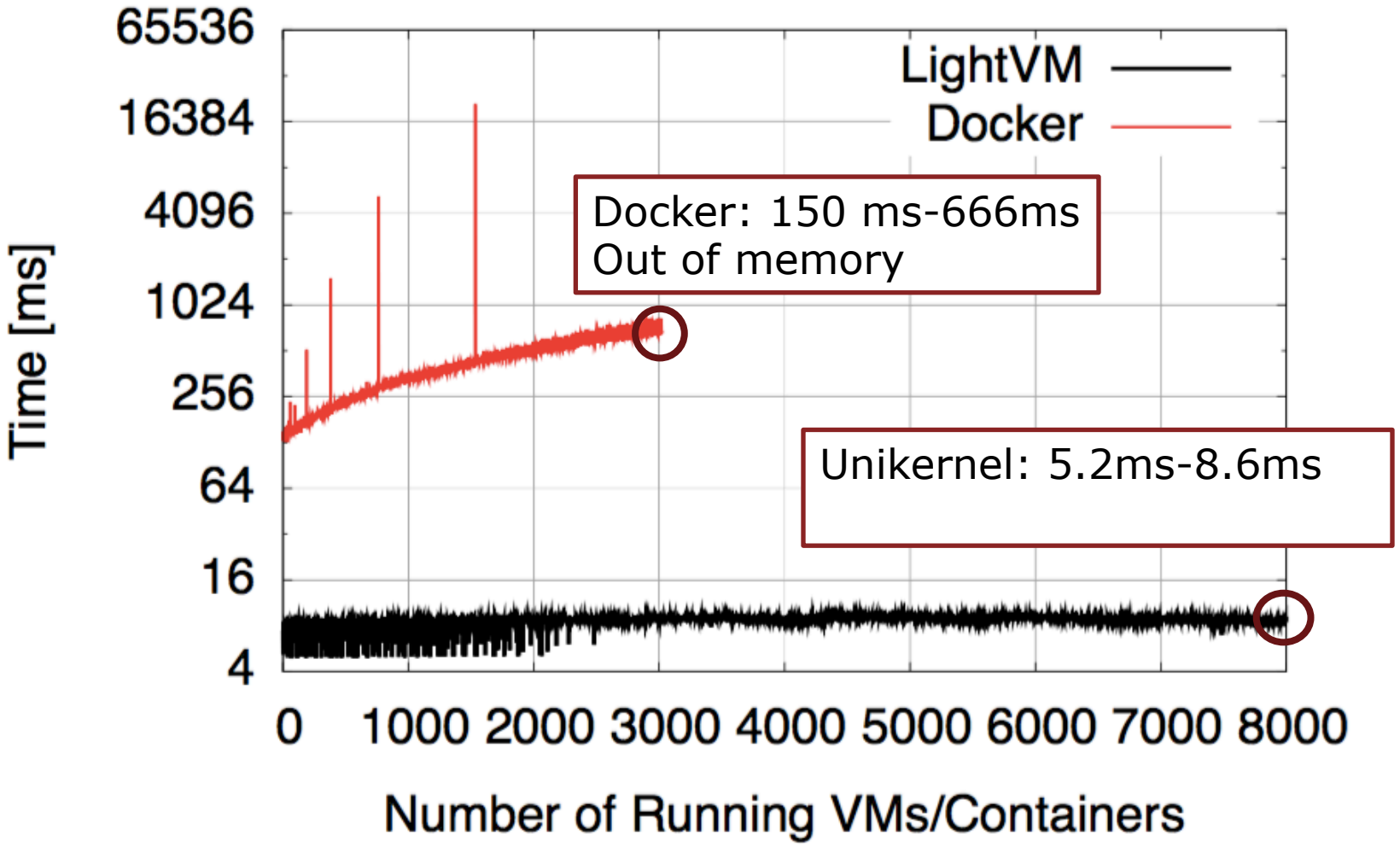
Evaluation – Unikernels We Used

- **Noop unikernel**: self explanatory ☺
- **Daytime unikernel**: TCP server that responds with current time
- **Minipython:** Port of Micropython to MiniOS
- **TLS unikernel:** TLS termination proxy
- **Ping unikernel:** also self-explanatory

Sample sizes:

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

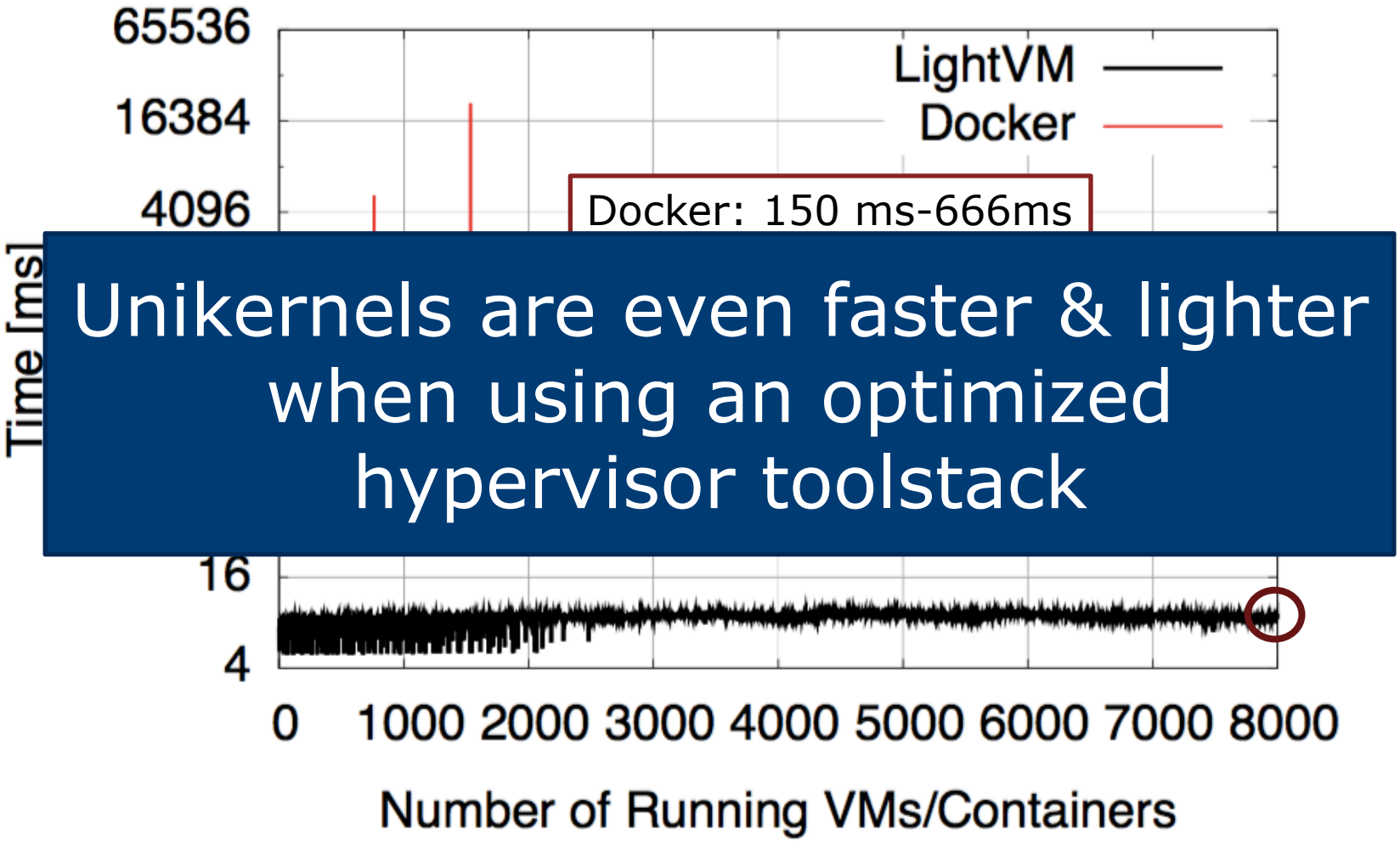
In Numbers: Instantiation Times with LightVM



Server: *4 x AMD Opteron 6376 CPU@2.3GHz (64 cores total), 128GB DDR3 RAM, Xen 4.8/Linux 4.8*
LightVM: http://sysml.neclab.eu/projects/lightvm

Guests: *Noop/Unikernel and Noop Docker image / No devices*

In Numbers: Instantiation Times with LightVM

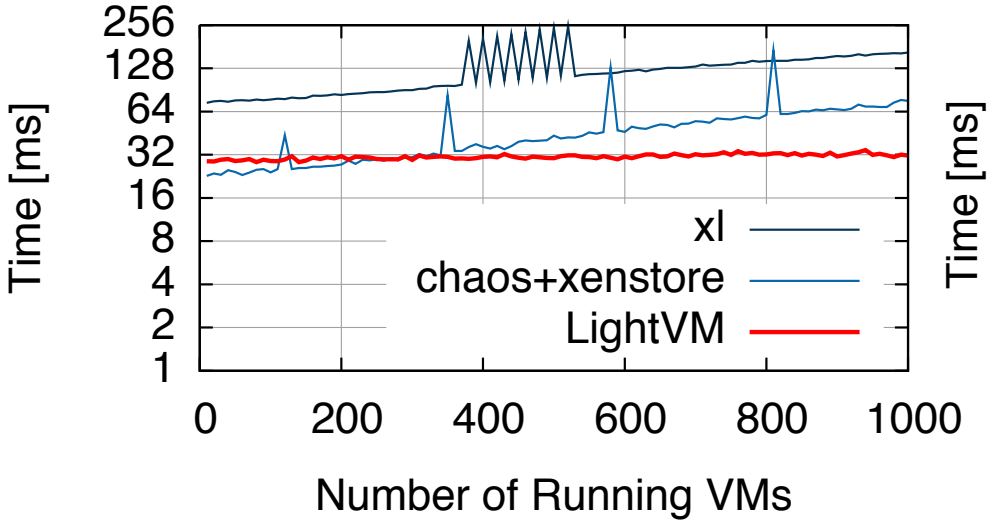


Unikernels are even faster & lighter when using an optimized hypervisor toolstack

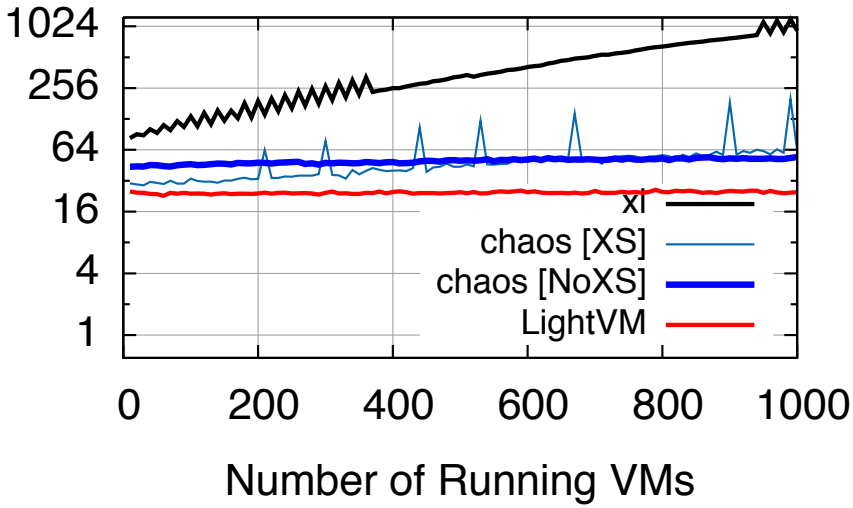
Server: *4 x AMD Opteron 6376 CPU@2.3GHz (64 cores total), 128GB DDR3 RAM, Xen 4.8/Linux 4.8*
LightVM: http://sysml.neclab.eu/projects/lightvm

Guests: *Noop/Unikernel and Noop Docker image / No devices*

Checkpointing – Daytime Unikernel



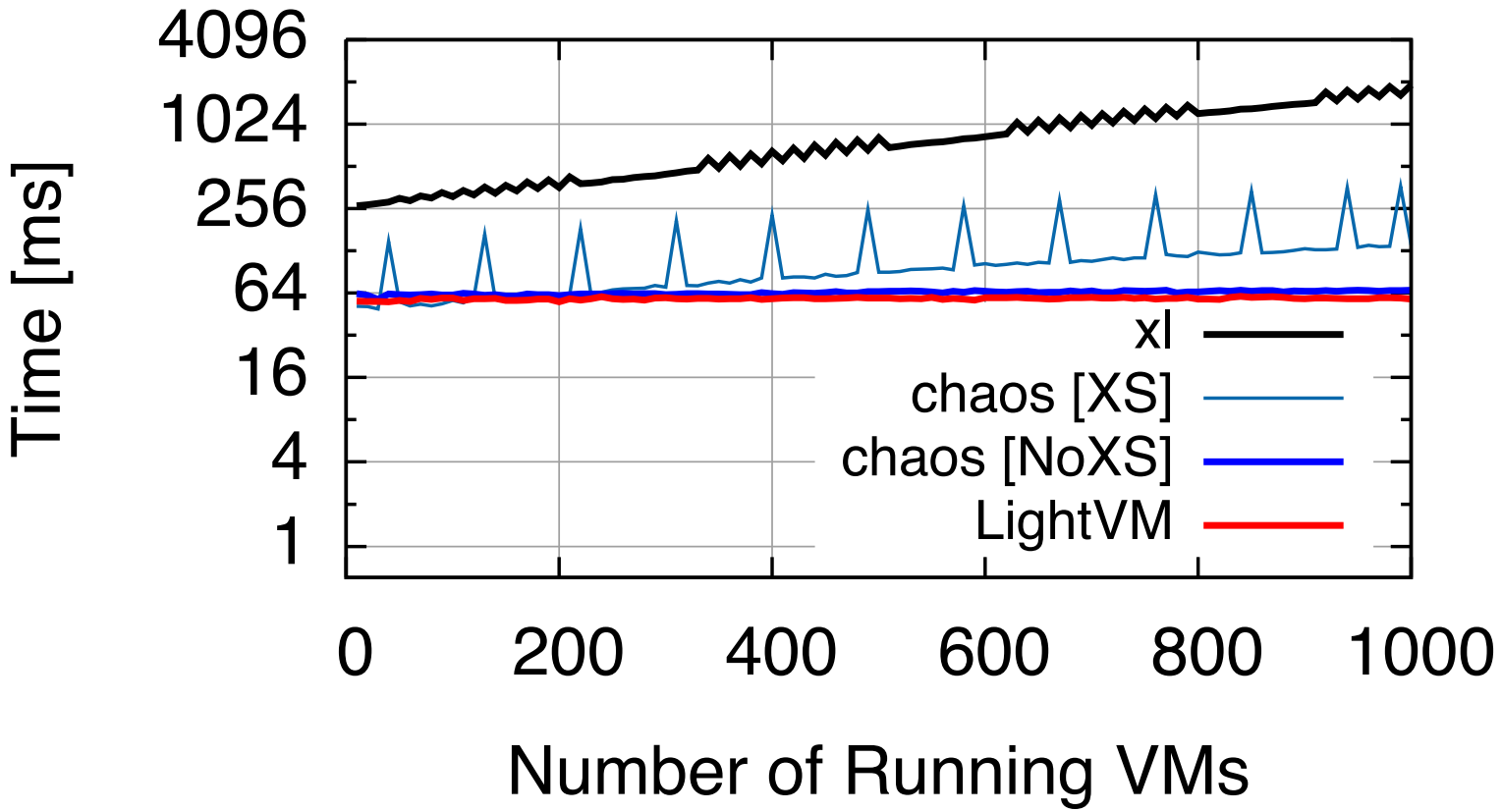
(a) Save



(b) Restore

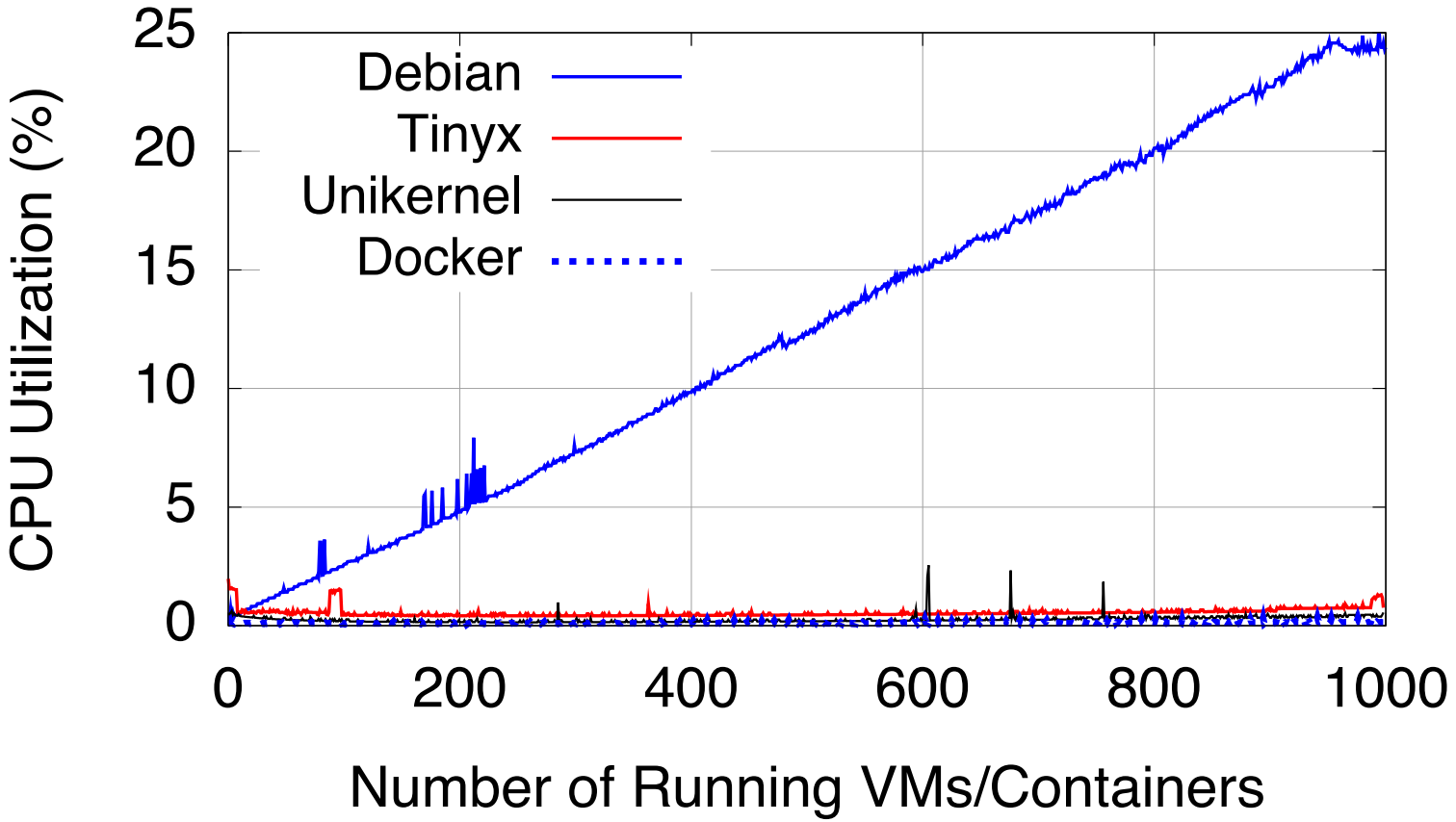
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Migration – Daytime Unikernel



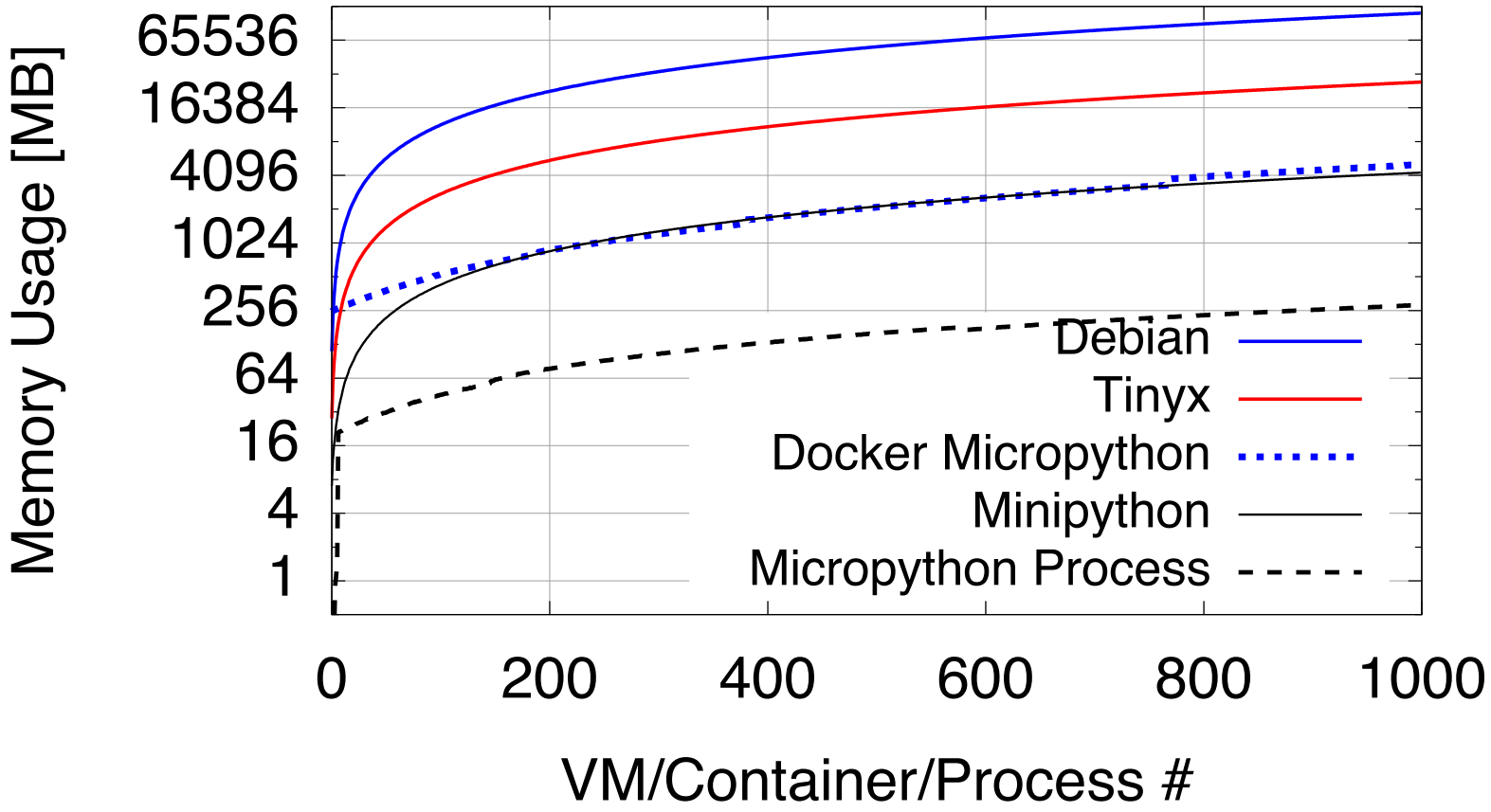
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

CPU Usage



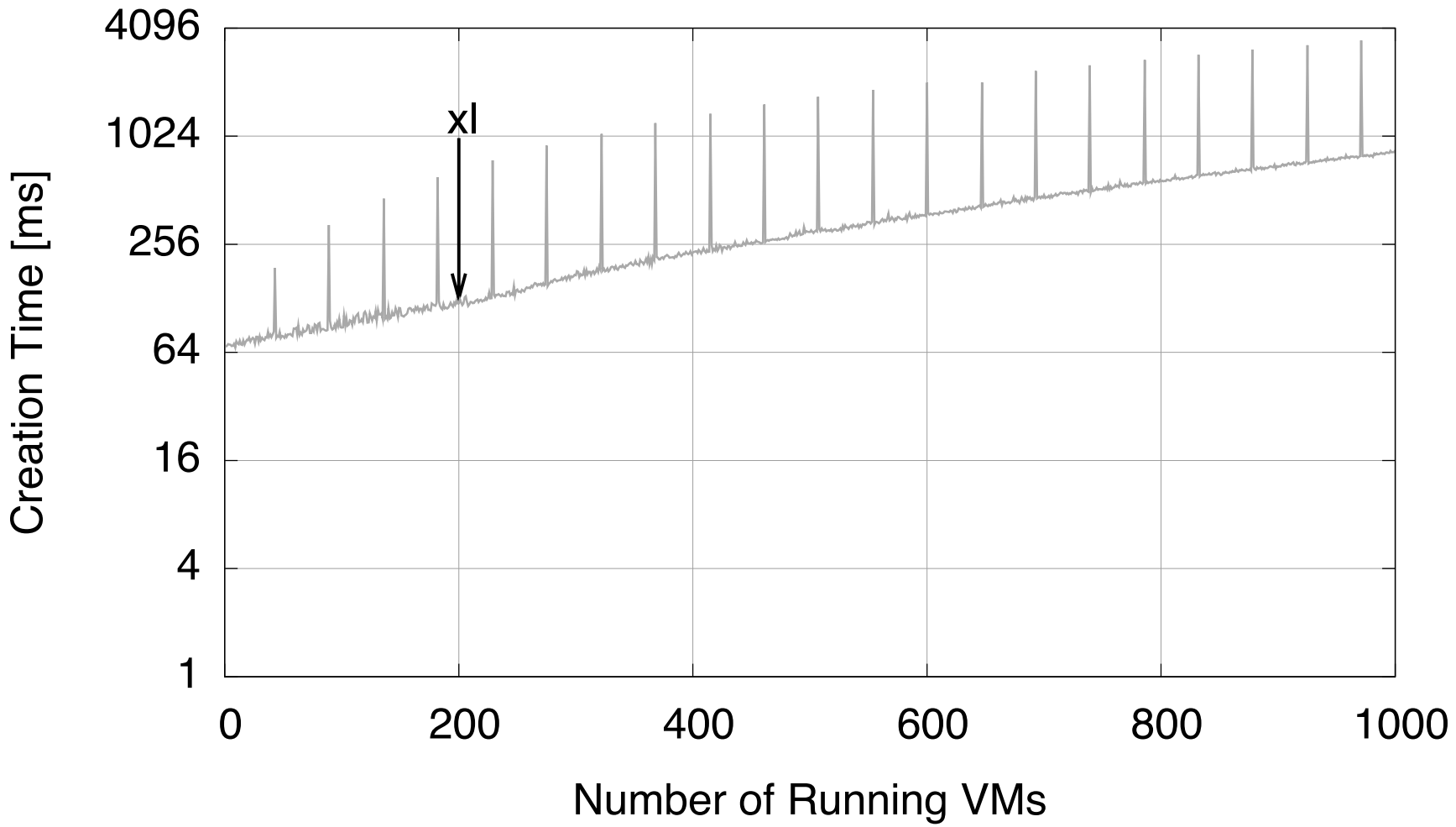
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Memory Usage



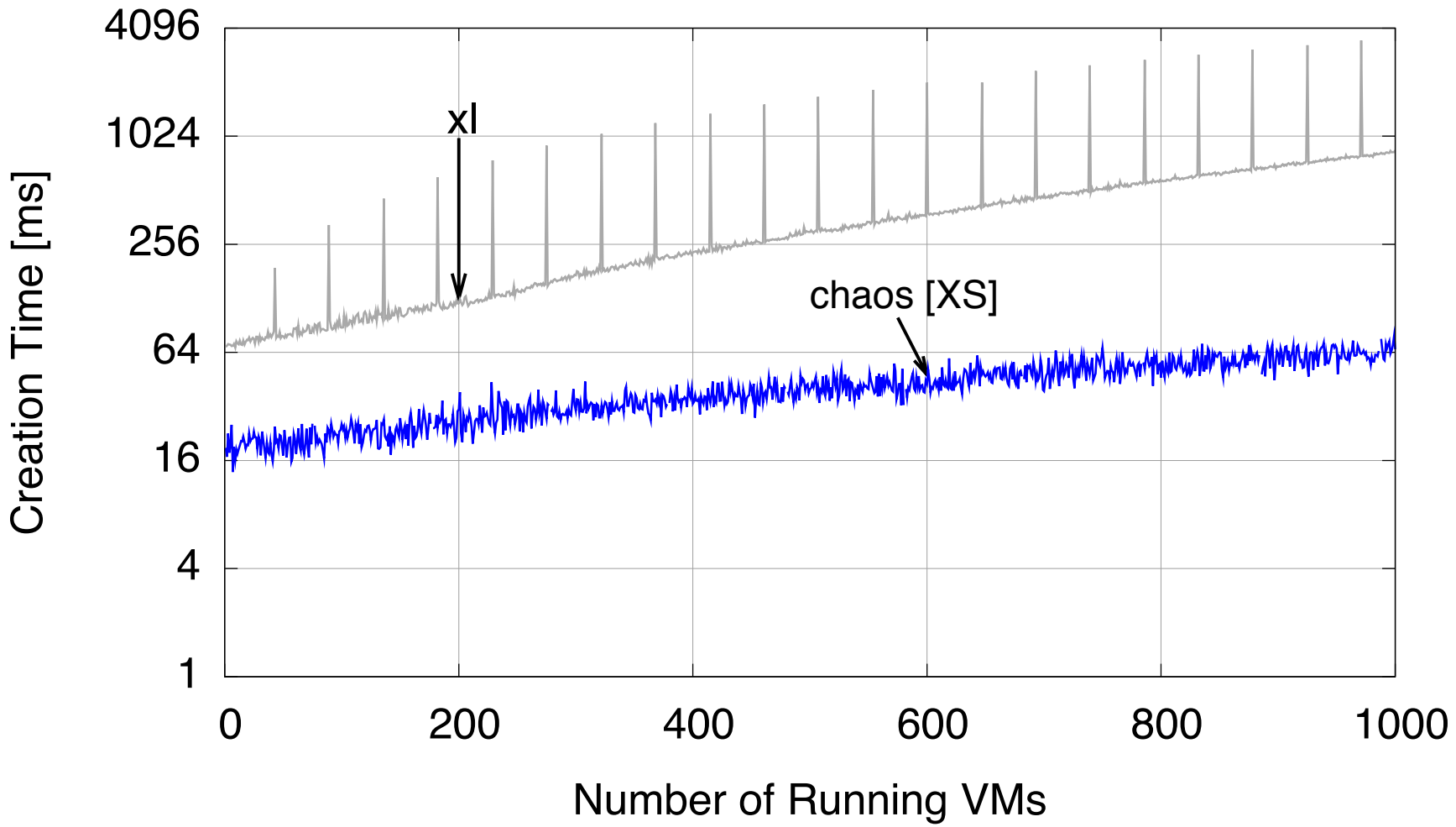
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Understanding LightVM's components



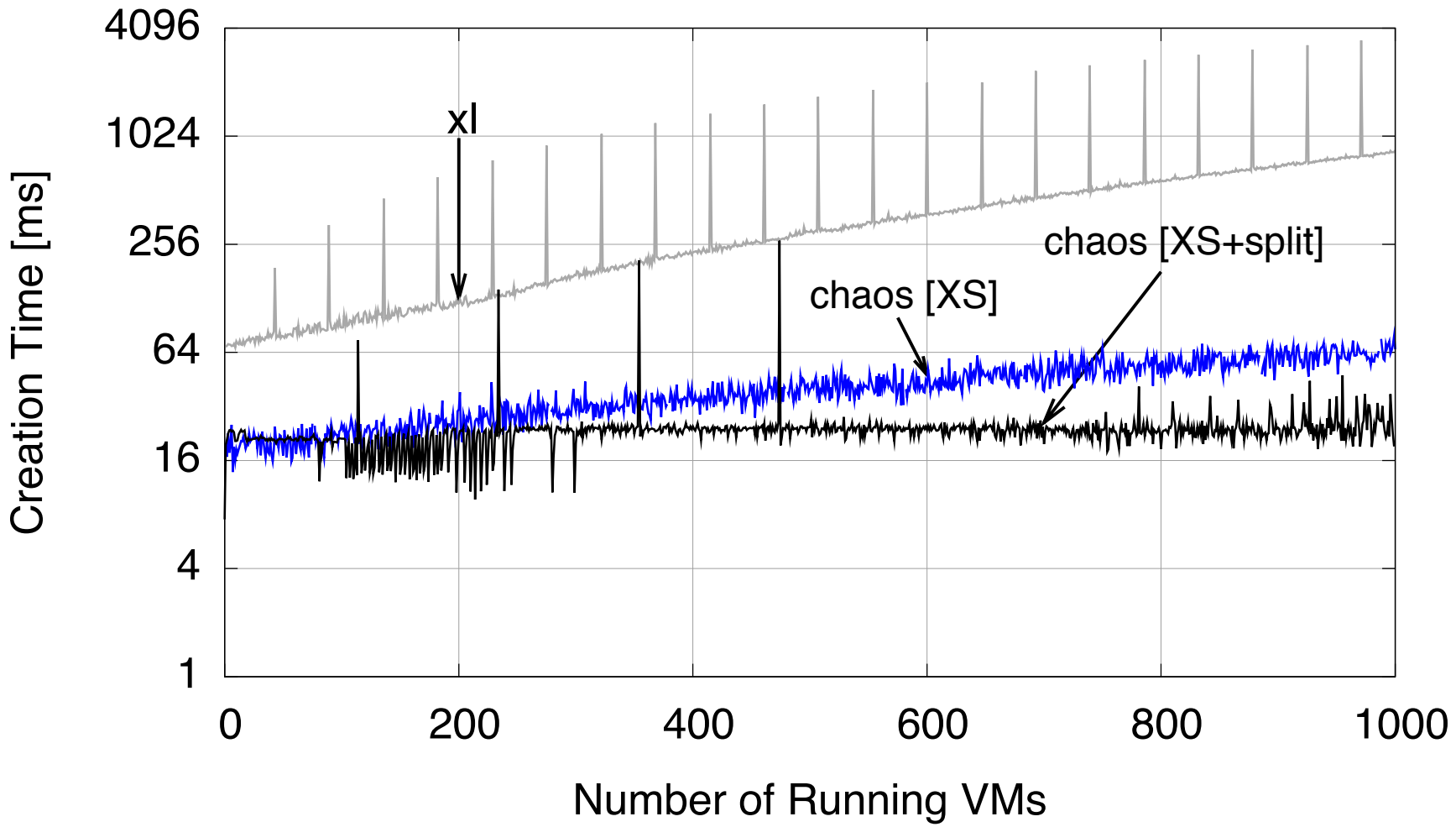
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Understanding LightVM's components



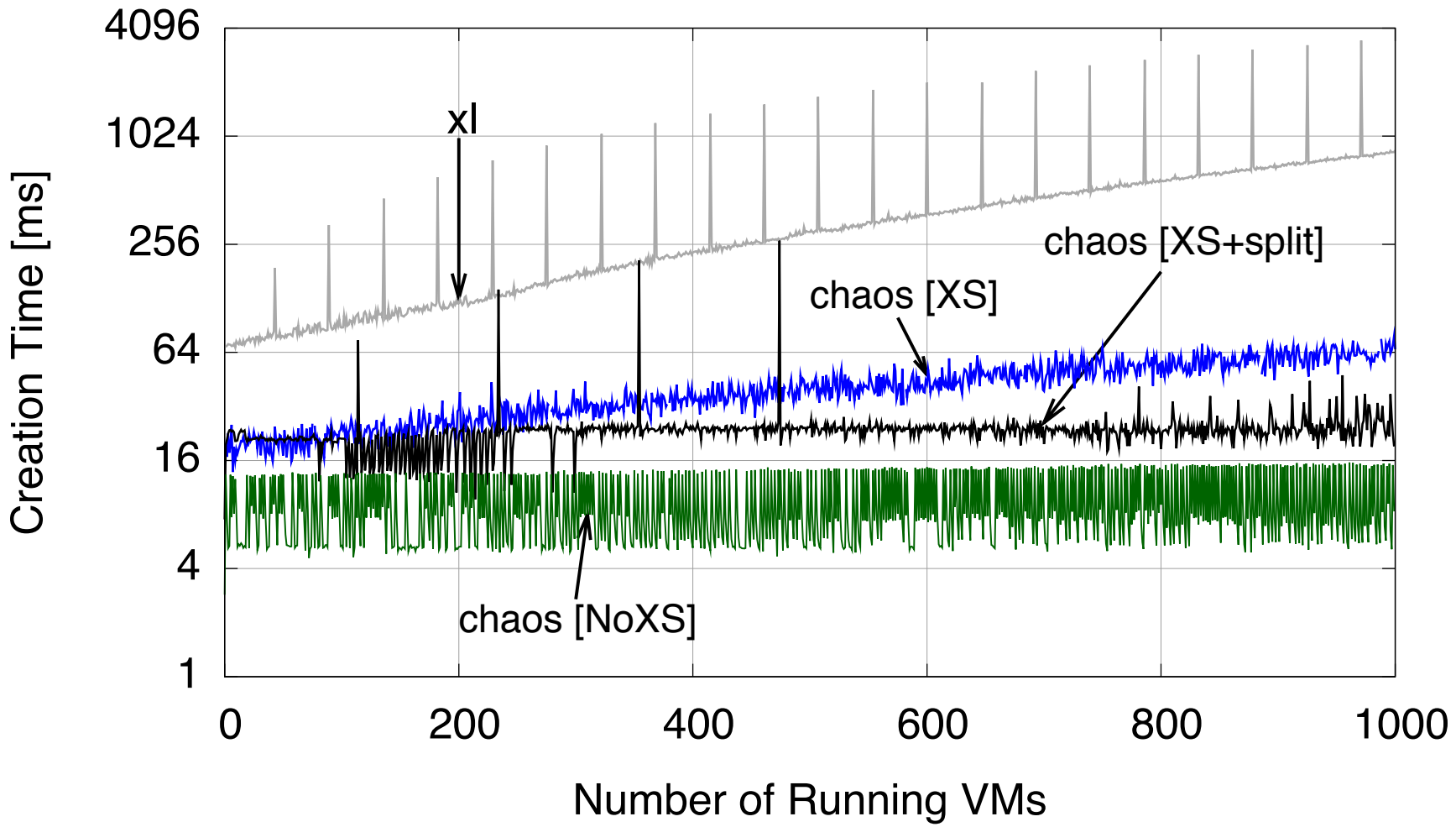
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Understanding LightVM's components



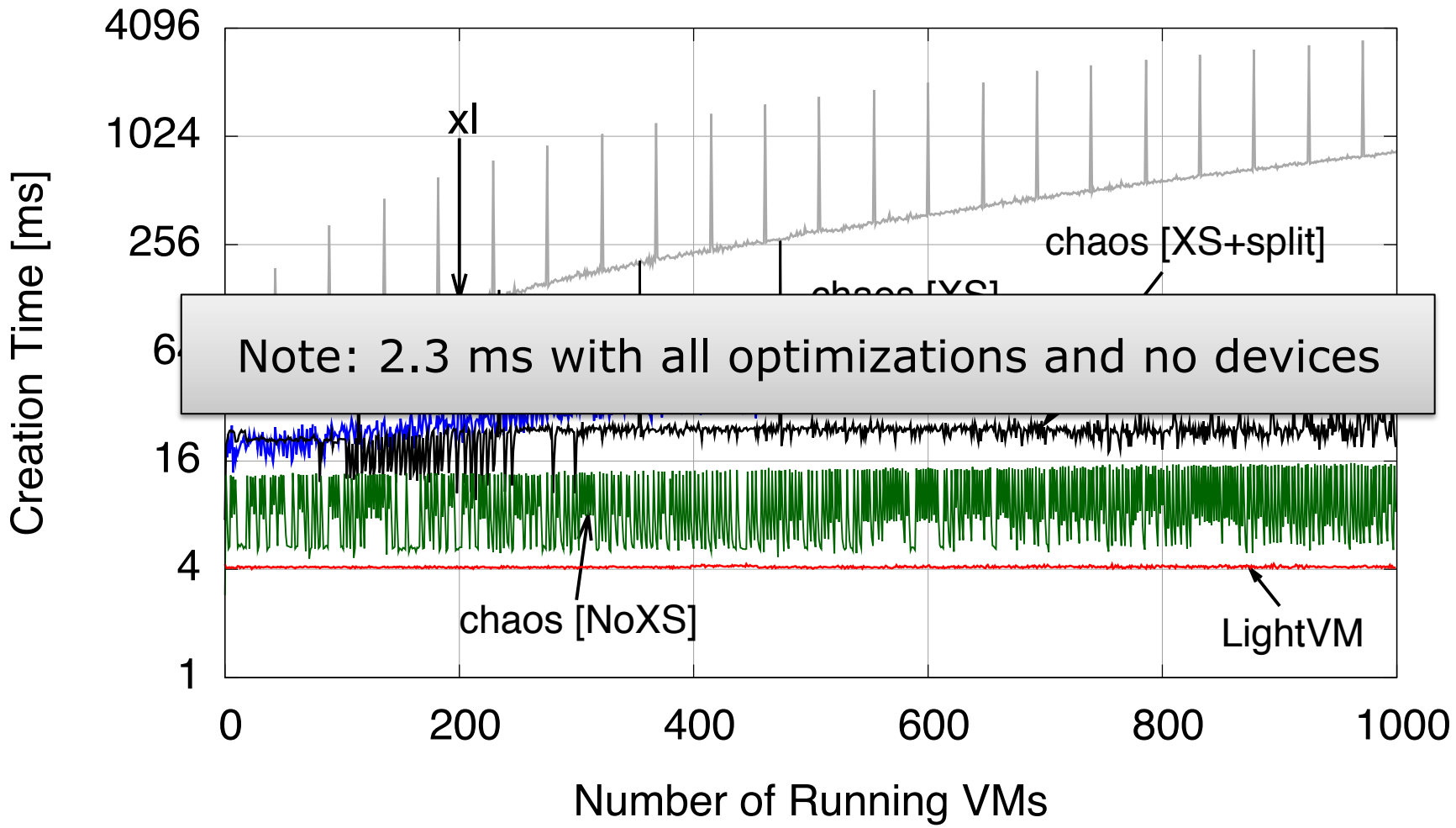
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Understanding LightVM's components



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

Understanding LightVM's components



Note: 2.3 ms with all optimizations and no devices

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

The Potential of Unikernels – A Summary

Fast instantiation, destruction and migration time

- 10s of milliseconds or less (and as little as 2.3ms)
  (LigthVM [Manco SOSP 2017], Jitsu [Madhvapeddy, NSDI 2015])

Low memory footprint

- Few MBs of RAM or less *(ClickOS [Martins NSDI 2014])*

High density

- 8k guests on a singlex86 server *(LigthVM [Manco SOSP 2017])*

High Performance

- 10-40Gbit/s throughput with a single guest CPU
  (ClickOS [Martins NSDI 2014], Elastic CDNs [Kuenzer VEE 2017])

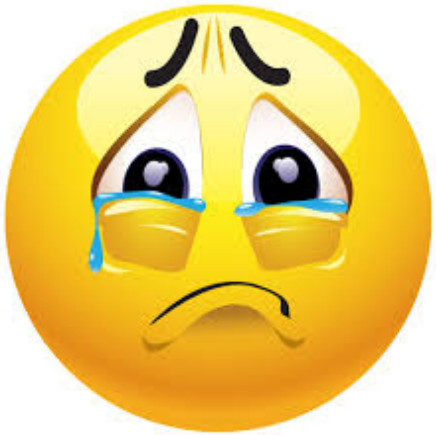
Reduced attack surface

- Small trusted compute base
- Strong isolation by hypervisor

So unikernels are perfect right?

The (Big) Downside with Unikernels

- Each optimized Unikernel is manually built
  - Building takes several months or longer
  - Potentially wash, rinse, repeat for each target application



Unikraft

A Unikernel and Specialized OS Build Framework

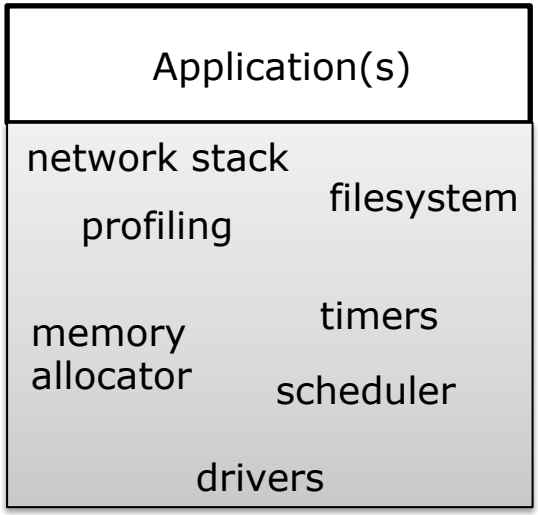
Unikraft Goals

- Severely reduce development time
- Simultaneously target multiple platforms
  - Xen
  - KVM
  - Bare metal (X86_64, ARM32, etc.)
- Allow simple customization of the unikernel *without* having to tweak the actual code

The Unikraft Way

Decompose OSes and libraries into primitives

Recompose them to meet the needs of particular applications

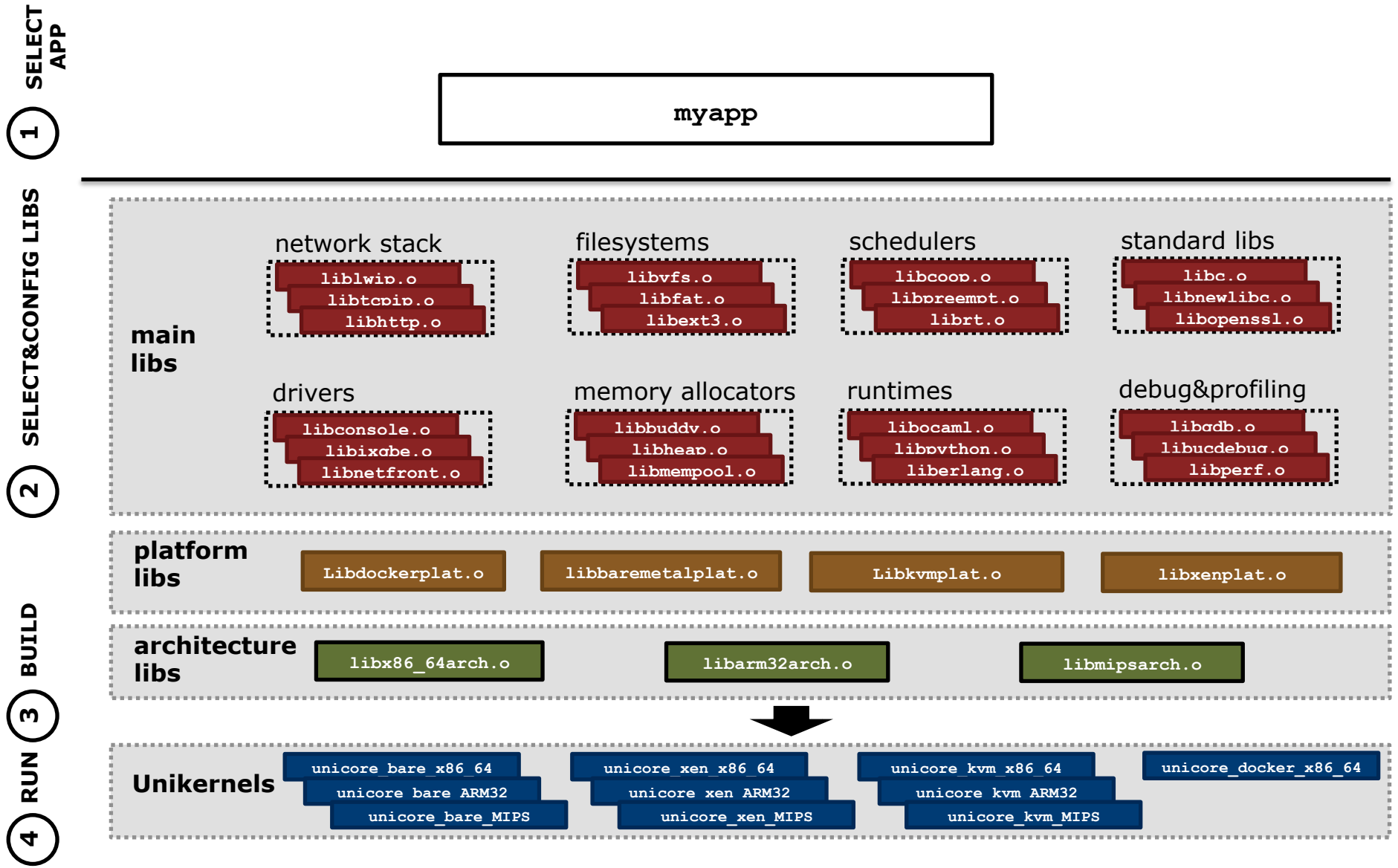


The Unikraft Way

Decompose OSes and libraries into primitives

Recompose them to meet the needs of particular applications

Unikraft Overview – Everything as a Library

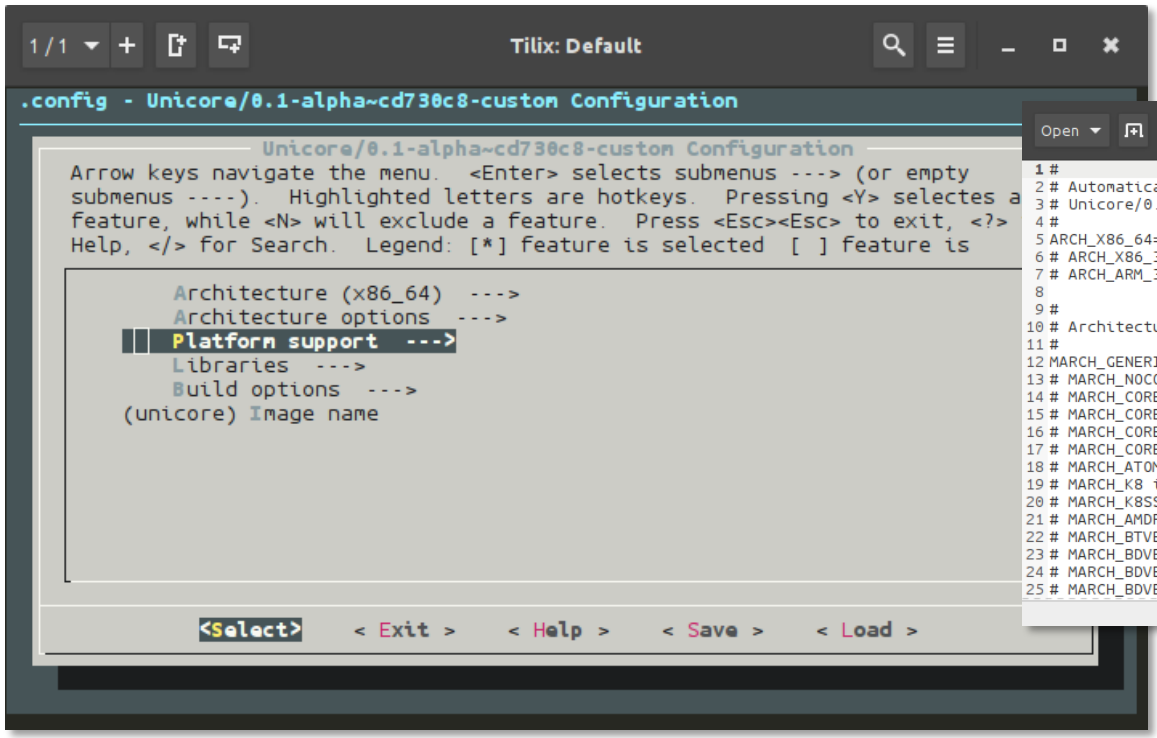


Two Library Types

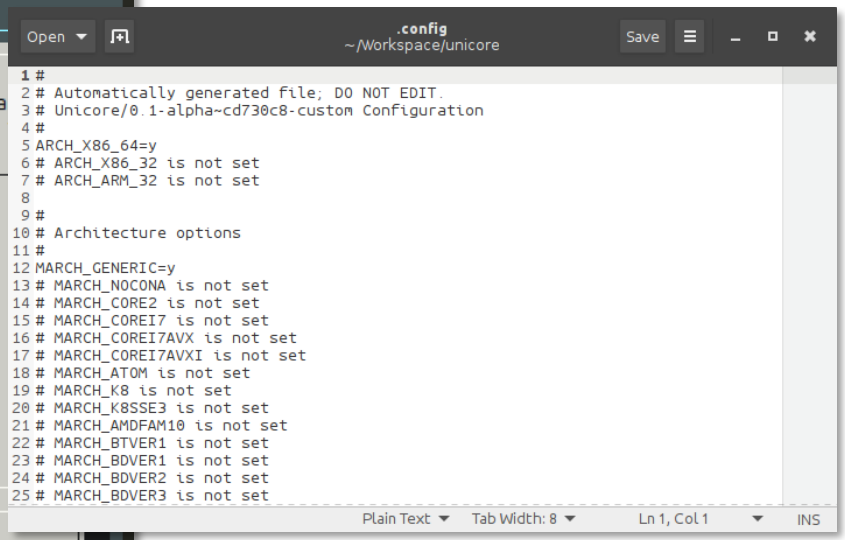
- **Built-in**: functionality specific to Unikraft (e.g., ukboot, ukschedpreempt, etc.)
- **External**: software projects external to Unikraft (e.g., lwip, micropython, etc.)

Putting Things Together – The Unikraft Build Tool

- Kconfig/Makefile based
- Type "`make menuconfig`"
- Choose options in the menu that you want for your application
- Choose your target platform(s)
  - Xen, KVM, bare-metal, Linux
- Save your config and type "`make`"

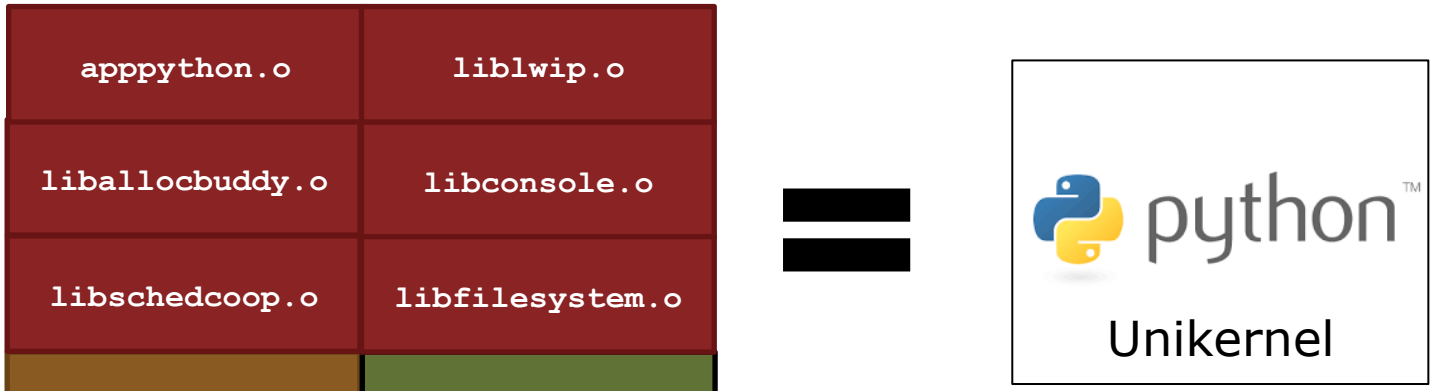


.config



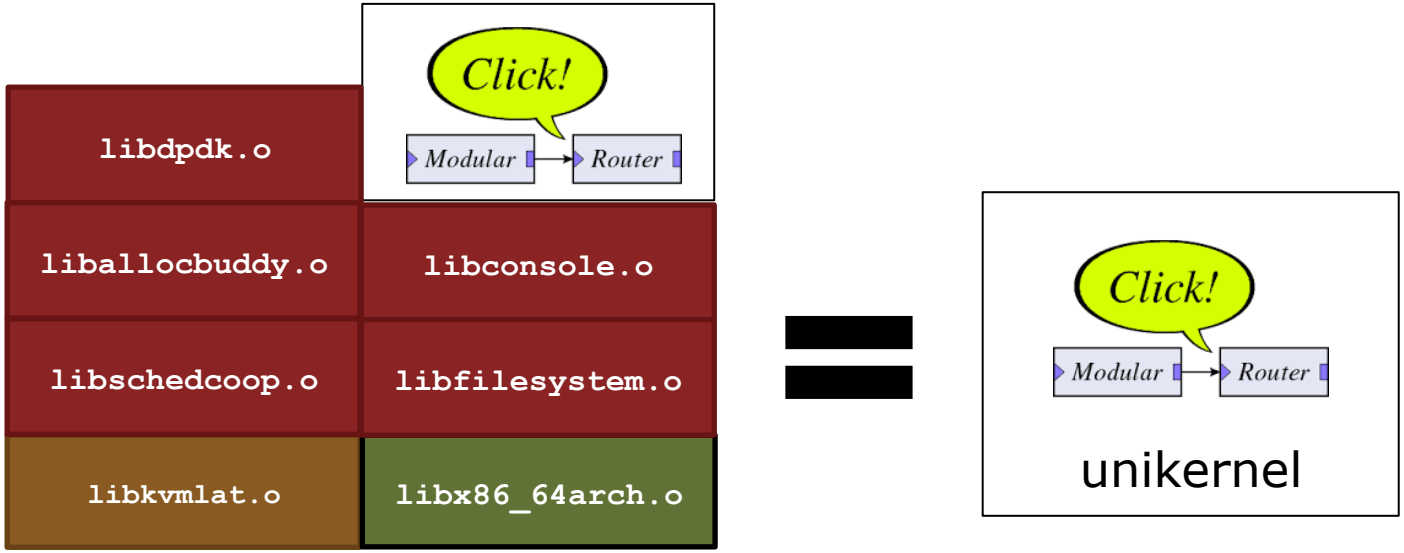
Example System

Python Unikernel for KVM on x86_64



Example System

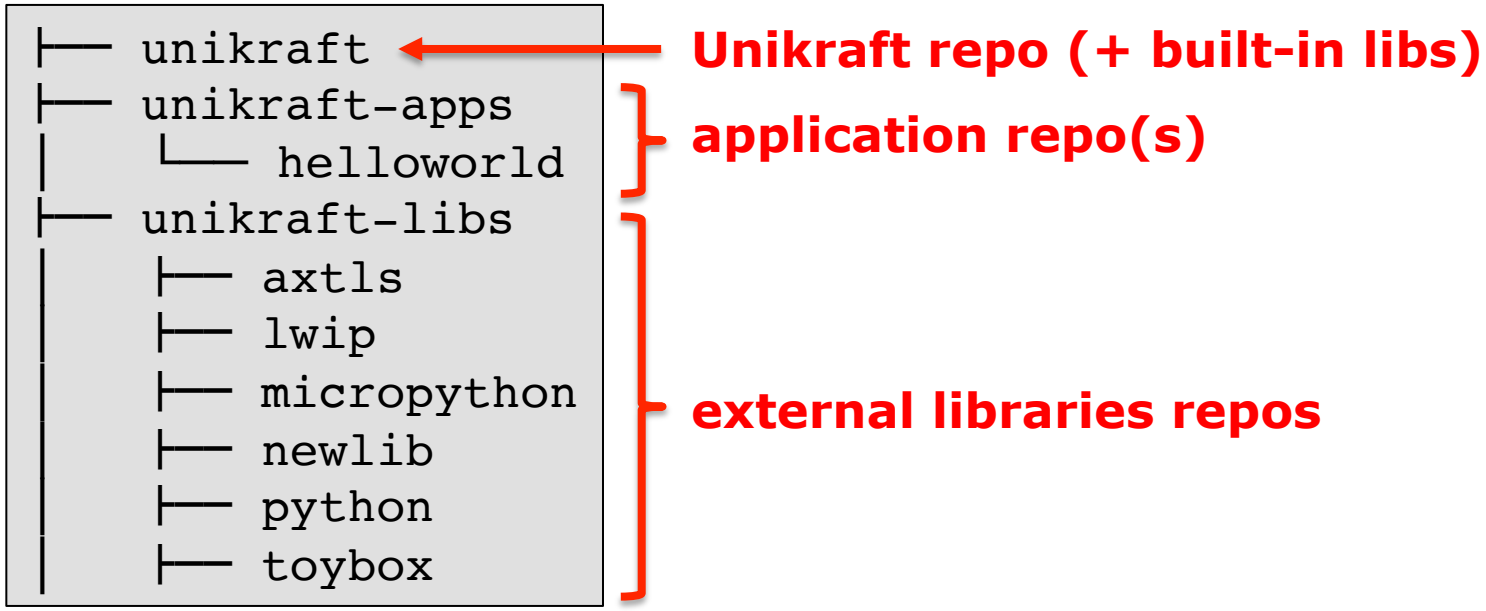
VNF Unikernel for KVM on x86_64



Building a Unikraft Hello World App

Repo Structure

- Clone the main Unikraft repo
- Clone any external library repos
- Create repo for the actual application



Hello World – Four Required Files (I)

Makefile: specify where the main Unikraft repo is, as well as repos for external libraries

```
UK_ROOT ?= $(PWD)/../../unikraft
UK_LIBS ?= $(PWD)/../../unikraft-libs
LIBS := $(UK_LIBS)/newlib

all:
	@make -C $(UK_ROOT) A=$(PWD) L=$(LIBS)

$(MAKECMDGOALS):
	@make -C $(UK_ROOT) A=$(PWD) L=$(LIBS) $(MAKECMDGOALS)
```

← **path to unikraft repo**

← **path to external libs**

← **external libs needed**

Hello World – Four Required Files (II)

Makefile.uk: specifies the sources to build for the application

```
$(eval $(call addlib,apphelloworld))

APPHELLOWORLD_SRCS-y += $(APPHELLOWORLD_BASE)/main.c
```

register app with uk Build system

Add main.c to build

Hello World – Four Required Files (III)

Config.uk: to populate Unikraft's menu with application-specific option

```
### Invisible option for dependencies
config APPHELLOWORLD_DEPENDENCIES
        bool
        default y
        select LIBNOLIBC if !HAVE_LIBC

### App configuration
config APPHELLOWORLD_PRINTARGS
        bool "Print arguments"
        default y
        help
          Prints argument list (argv) to stdout
```

Hello World – Four Required Files (IV)

main.c: source file to provide (at least) a `main()` function

```
#include <stdio.h>

/* Import user configuration: */
#include <uk/config.h>

int main(int argc, char *argv[])
{
        printf("Hello world!\n");

#if APPHELLOWORLD_PRINTARGS
        int i;

        printf("Arguments: ");
        for (i=0; i<argc; ++i)
                printf(" \"%s\"", argv[i]);
        printf("\n");
#endif
}
```

Unikernel entry point after boot

defined by Config.uk

Porting an External Library

How To Port an External Library

- Most of the work revolves around ensuring Unikraft supports your lib's dependencies → we're working on this! ☺
- Write `Makefile.uk` and add the external library source files to it
- Sometimes a bit of *glue* code is needed
  - E.g., in newlib to link POSIX thread creation to Unikraft's thread library

How To Port an External Library – Makefile.uk

```
################################################################################
# Library registration
################################################################################
$(eval $(call addlib_s,libaxtls,$(LIBAXTLS)))

################################################################################
# Sources
################################################################################
# Nothing here: sources are small and included directly in the uk library repo
# Note: sources are from axTLS-2.1.4.tar.gz .

################################################################################
# Library includes
################################################################################
CINCLUDES-y +=    -I$(LIBAXTLS_BASE)/include \
                  -I$(LIBAXTLS_BASE)/crypto  \
                  -I$(LIBAXTLS_BASE)/ssl

################################################################################
# sources
################################################################################
LIBAXTLS_SRCS-y += $(LIBAXTLS_BASE)/crypto/aes.c
LIBAXTLS_SRCS-y += $(LIBAXTLS_BASE)/crypto/bigint.c
…
LIBAXTLS_SRCS-y += $(LIBAXTLS_BASE)/crypto/sha512.c
```

Unikraft 0.2 Titan

Current Status

Available Libraries

Main Libraries

- libfdt
  - Flat device tree parser
- libnolibc
  - A tiny libC replacement
- libnewlib (external)
  - Porting ongoing
- libukalloc
  - Memory allocator abstraction
- libukallocbbuddy
  - Binary buddy allocator for libukalloc
- libukargparse
  - Argument parser library
- libukboot
  - Unikraft bootstrapping
- libukdebug
  - Debug and kernel printing
  - Assertions, hexdump
- libuksched
  - Scheduler abstraction
- libukschedcoop
  - Cooperative scheduler for libsched

Architecture Libraries

- libarmmath
  - 64bit arithmetic on ARMv7

Platform Libraries

- libxenplat
  - Xen (PV)
  - X86_64, ARMv7
- libkvmplat
  - X86_64
- liblinuxu
  - Linux userspace
  - X86_64, ARMv7

Coming Soon

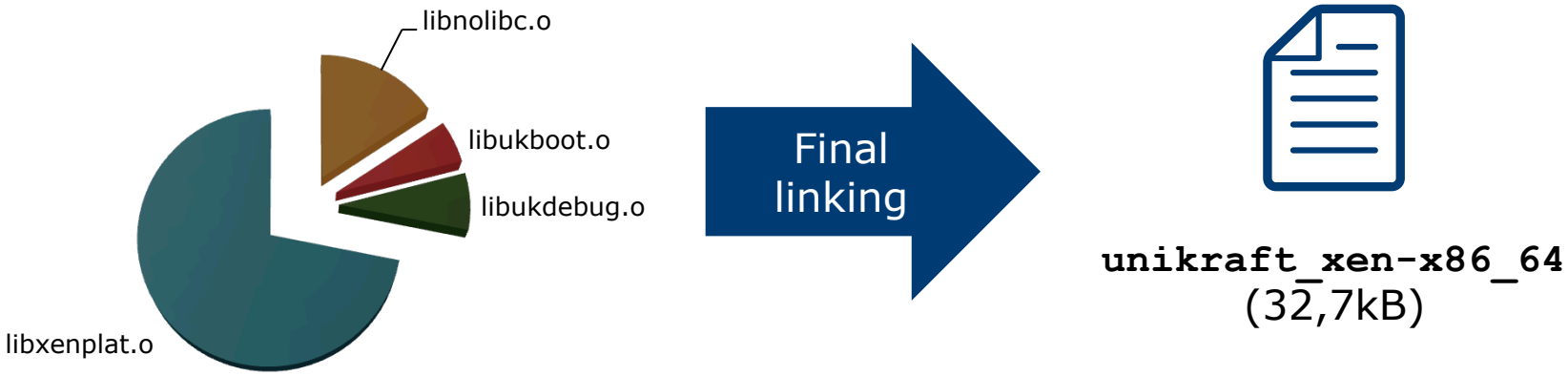
- libukschedpreempt
  - Pre-emptive scheduler
- libukswrand
  - Software random number generator
- libukvirtio
  - Virtio driver
- liblwip (external)
  - Network stack
- libclick (external)
  - The Click modular router (e.g., for NFV)
- libmicropython (external)
  - Micropython (Python for embedded devices)
- libcpython (external)
  - Standard Python
- libaxtls (external)
  - TLS support for embedded devices

More on the way!

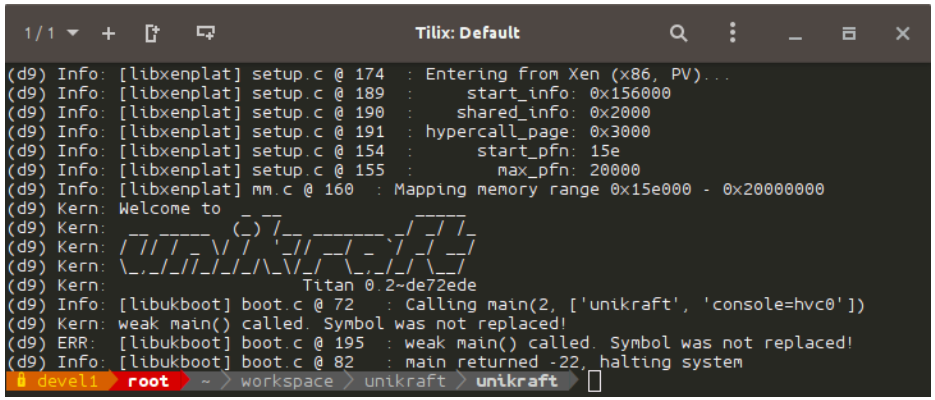
A Baseline Example...

Xen PV x86_64 binary

`unikraft_xen-x86_64.o` (50,2kB)



Boots and prints messages to debug console (with min. 208kB RAM)



Short Video/Demo

It is real!

It’s Open Source – Join Us!

- Recently: Unikraft got released as Xen incubator project
  - OpenSource development under Xen and LinuxFoundation umbrella

- Have look on the Xen Pages and Wiki
  - https://www.xenproject.org/developers/teams/unikraft.html
  - https://wiki.xenproject.org/
- Drop us an mail
  - minios-devel@lists.xen.org
- Join our IRC channel
  - #unikraft on Freenode
- We have plenty of open topics, get in touch with us!
  - http://sysml.neclab.eu

\Orchestrating a brighter world

NEC

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement no. 671566 ("Superfluidity"). This work reflects only the author's views and the European Commission is not responsible for any use that may be made of the information it contains.