\Orchestrating a brighter world

EU H2020 Superfluidity

FOSDEM 2018

Unleashing the Power of Unikernels with

Unikraft

Simon Kuenzer <simon.kuenzer@neclab.eu>
NEC Laboratories GmbH

\Orchestrating a brighter world

NEC brings together and integrates technology and expertise to create
the ICT-enabled society of tomorrow.
We collaborate closely with partners and customers around the world,
orchestrating each project to ensure all its parts are fine-tuned to local needs.

Every day, our innovative solutions for society contribute to
greater safety, security, efficiency and equality,
and enable people to live brighter lives.

Unikernels

What are they? What are they good for?

Orchestrating a brighter world NEC

The Container Craze

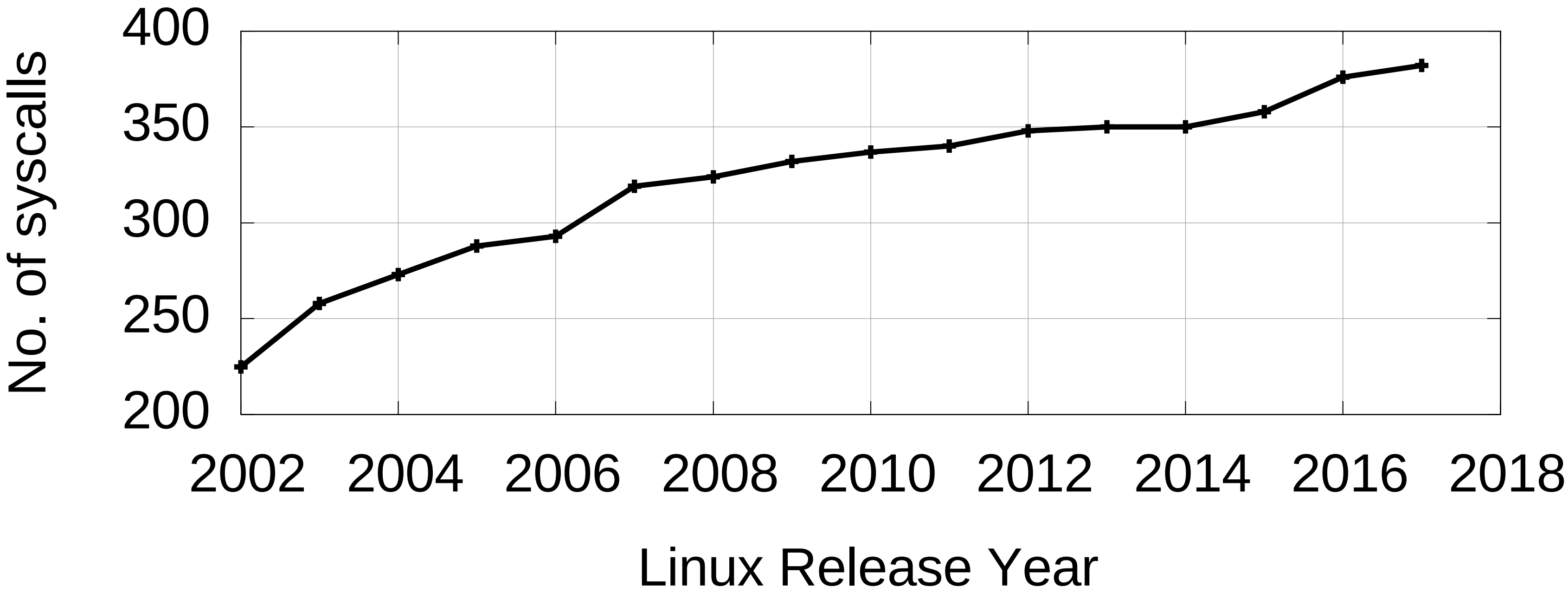
- “Google runs all services in containers”
- Container-as-a-Service services:
  - Amazon Lambda, EC2 Container Service
  - Google Container Engine
  - Azure Container service
  - *and so on…*

The Container Advantage for Service Deployments

- 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.)



Picking a Poison

CONTAINERS

docker

- ✅ Lightweight
- ❌ Iffy isolation

HYPERVISORS

Xen™

KVM

vmware®

- ✅ Strong isolation
- ❌ Heavy weight

CONTAINERS

HYPERVISORS

Can we break the myth of VMs being heavy weight?

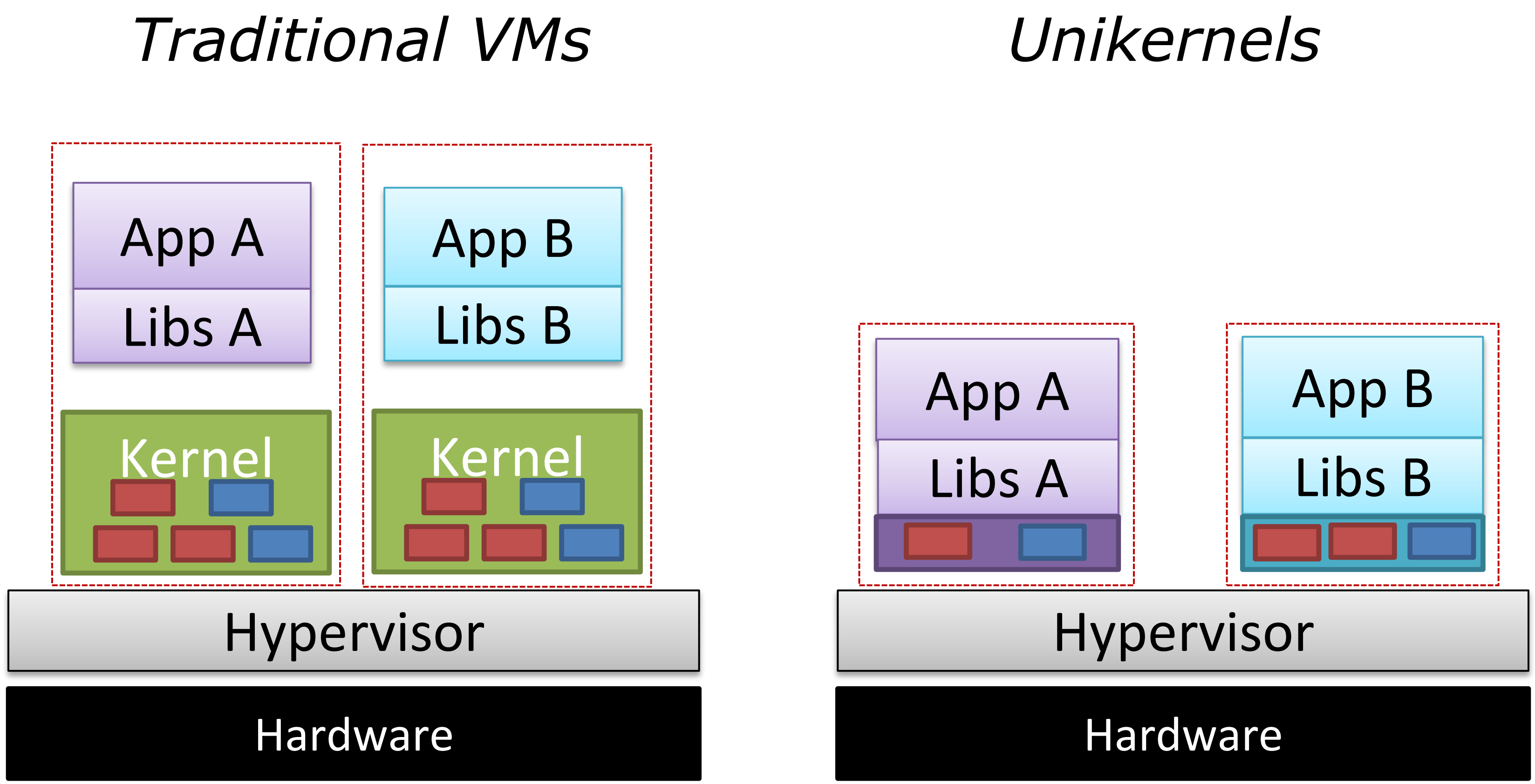
Yes, with Unikernels!

on

❌ Iffy isolation

❌ Heavy weight ?

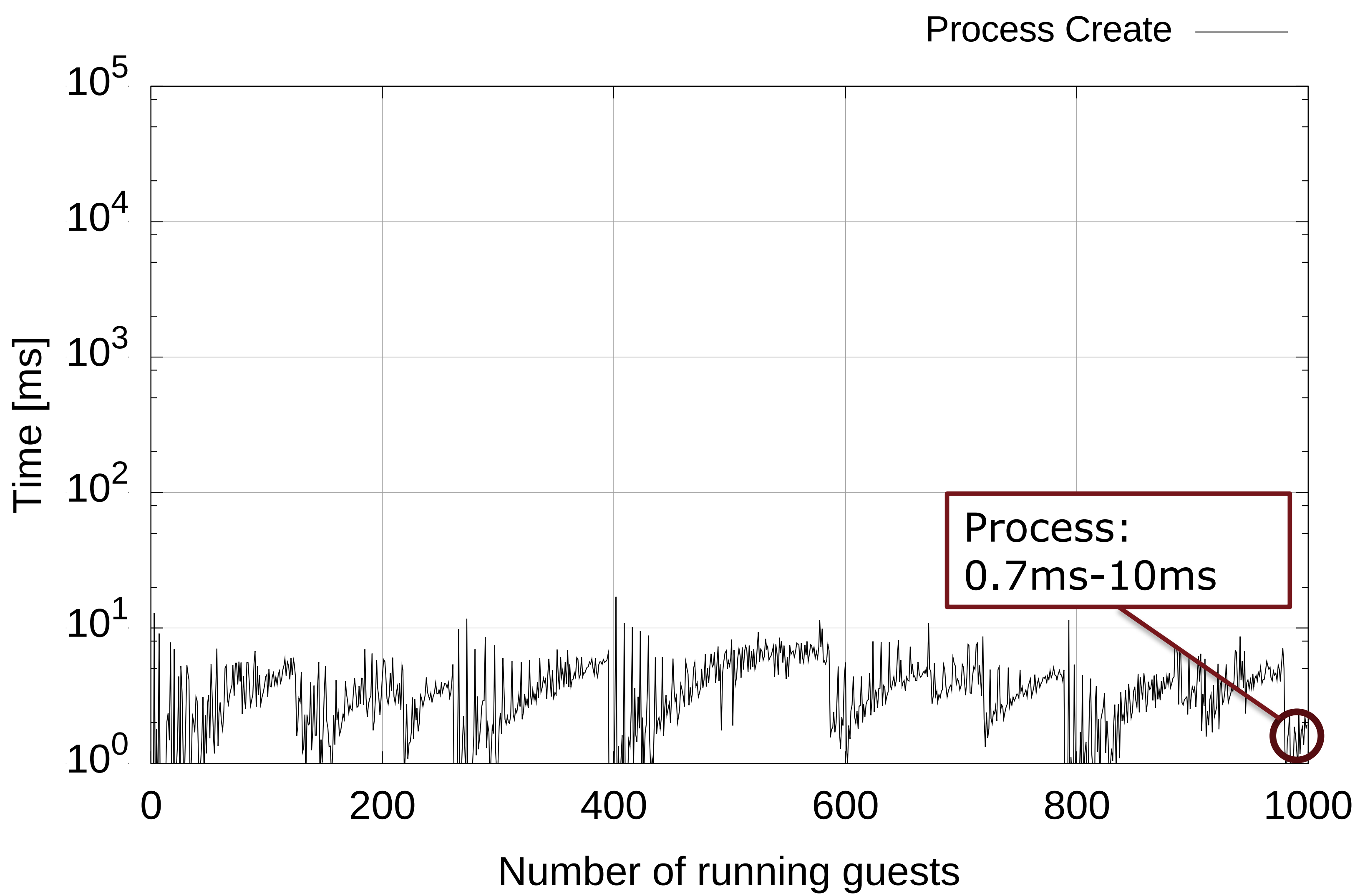
Unikernel



- Unikernels are purpose-built
  - Thin kernel layer (only what application needs)
  - One application → Flat and single address space
- Further advantages from specialization

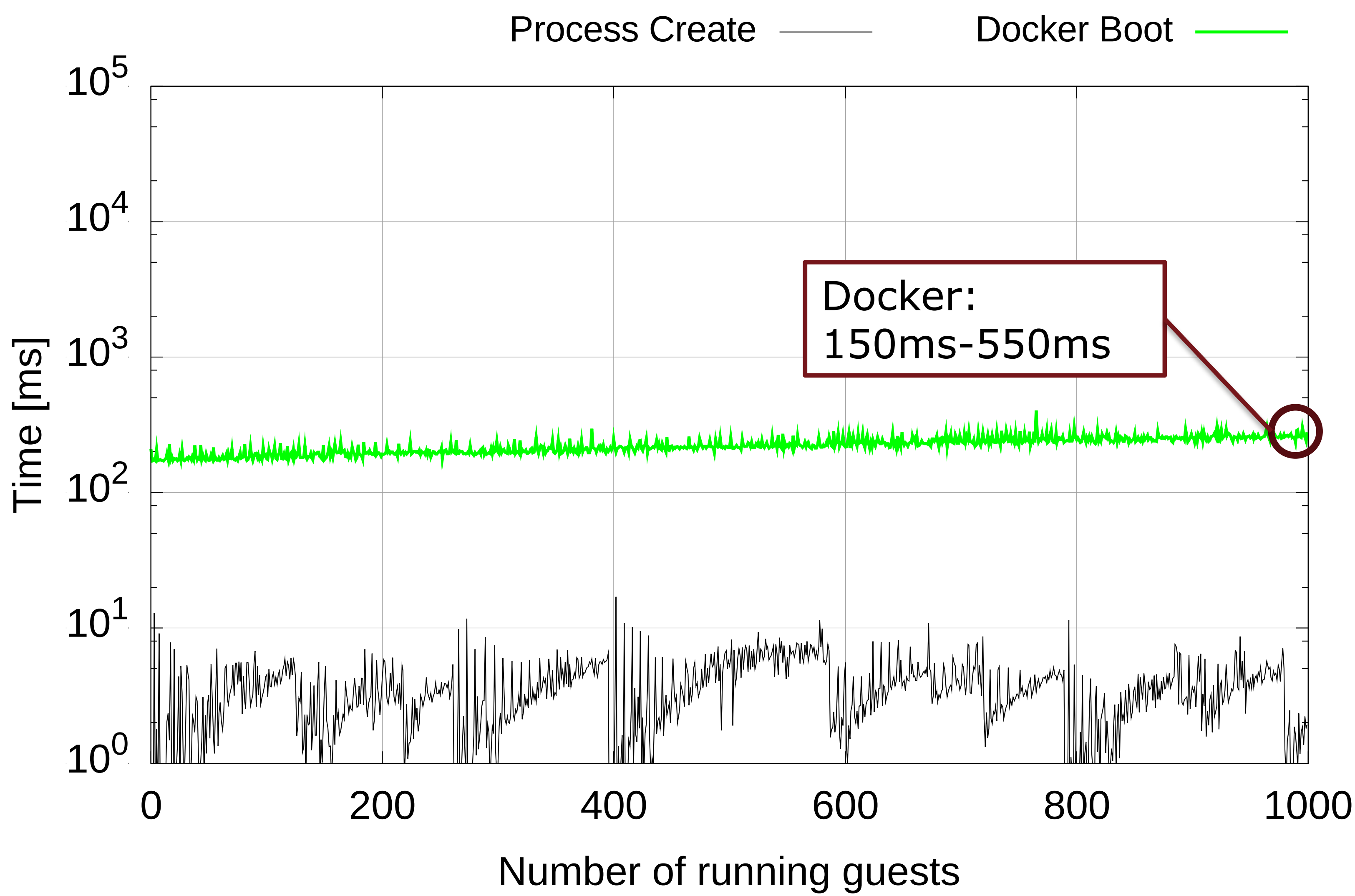
In Numbers: Instantiation Times

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



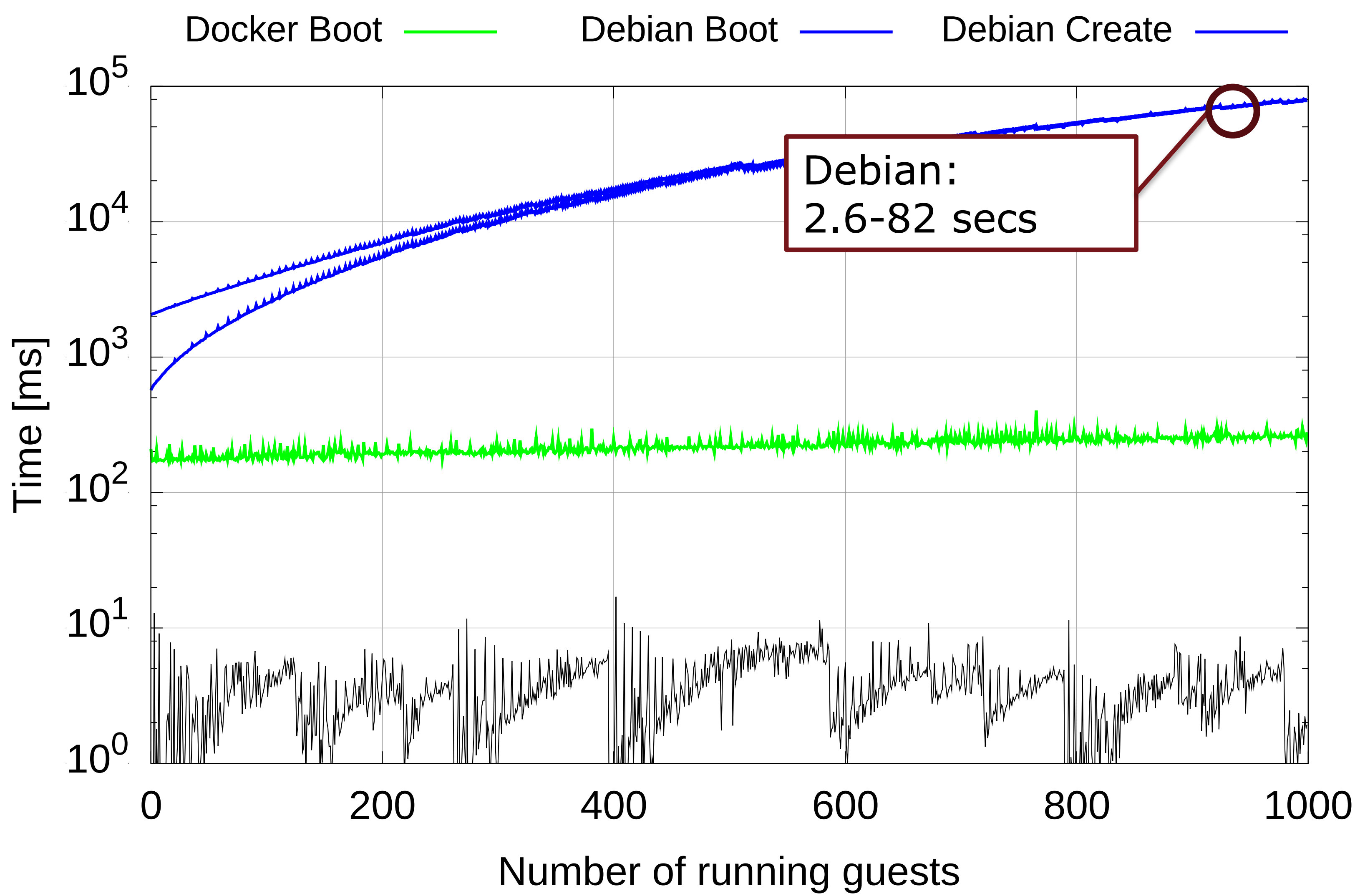
In Numbers: Instantiation Times

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



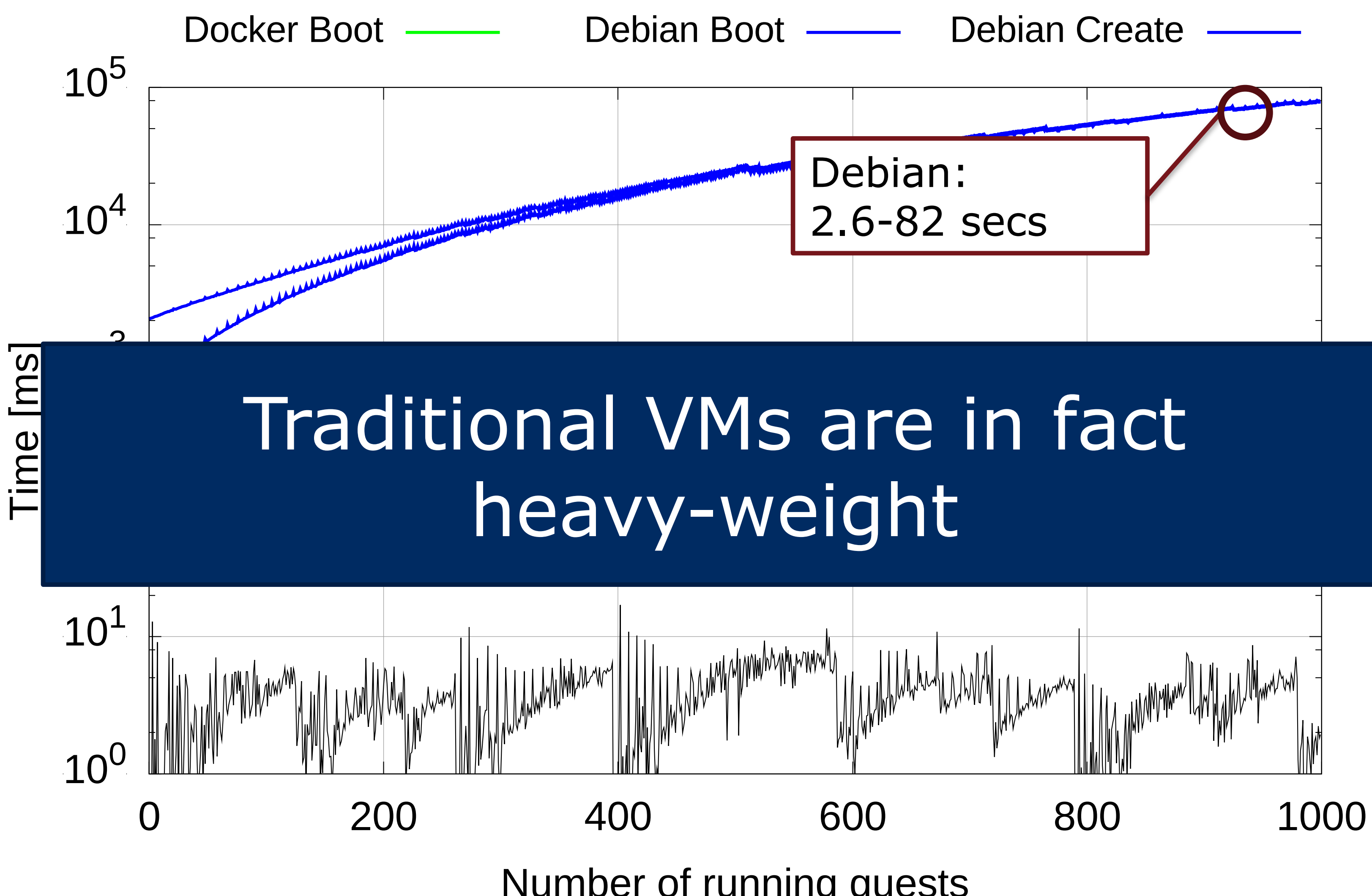
In Numbers: Instantiation Times

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



In Numbers: Instantiation Times

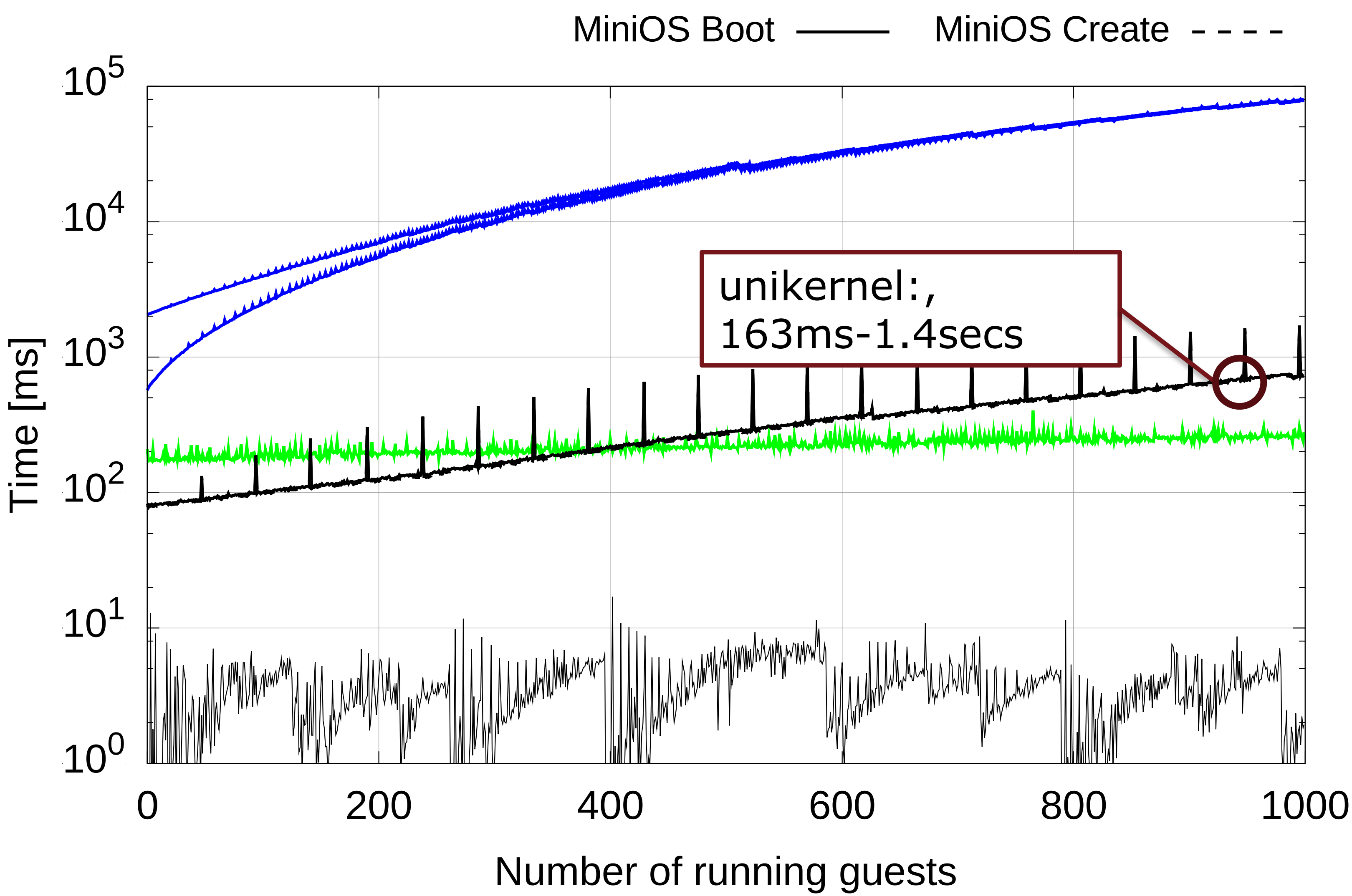
Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8



Traditional VMs are in fact heavy-weight

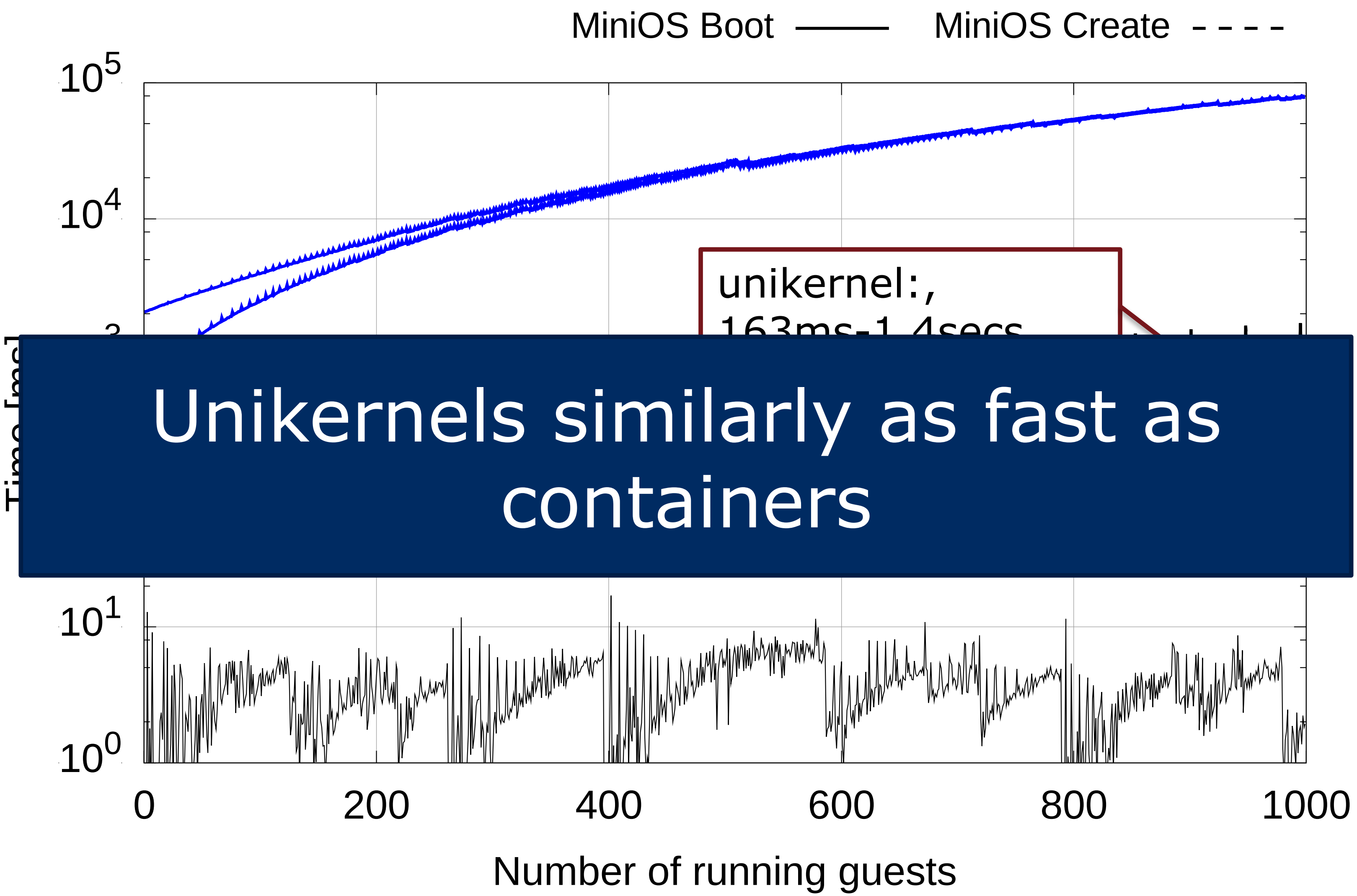
In Numbers: Instantiation Times

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



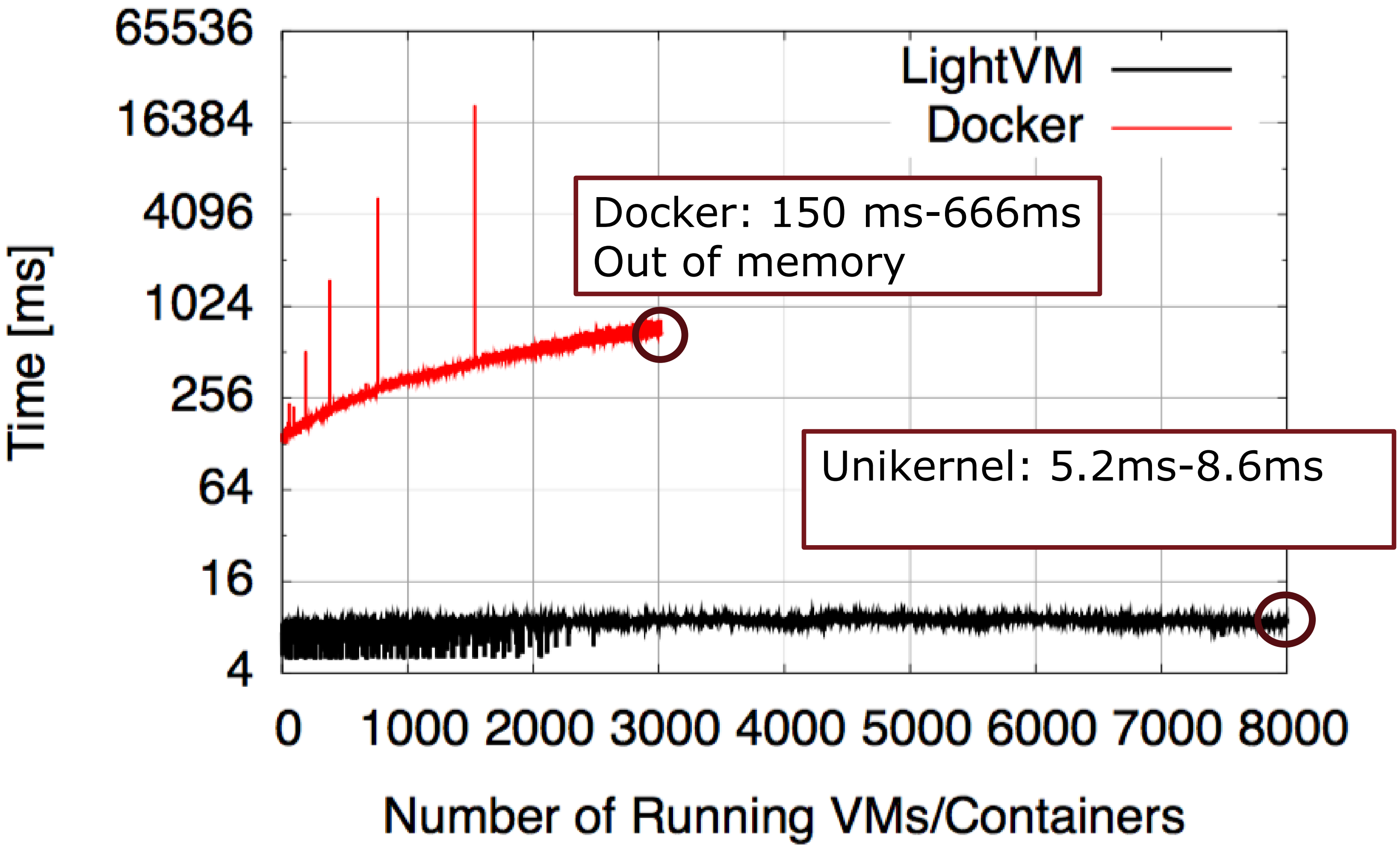
In Numbers: Instantiation Times

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



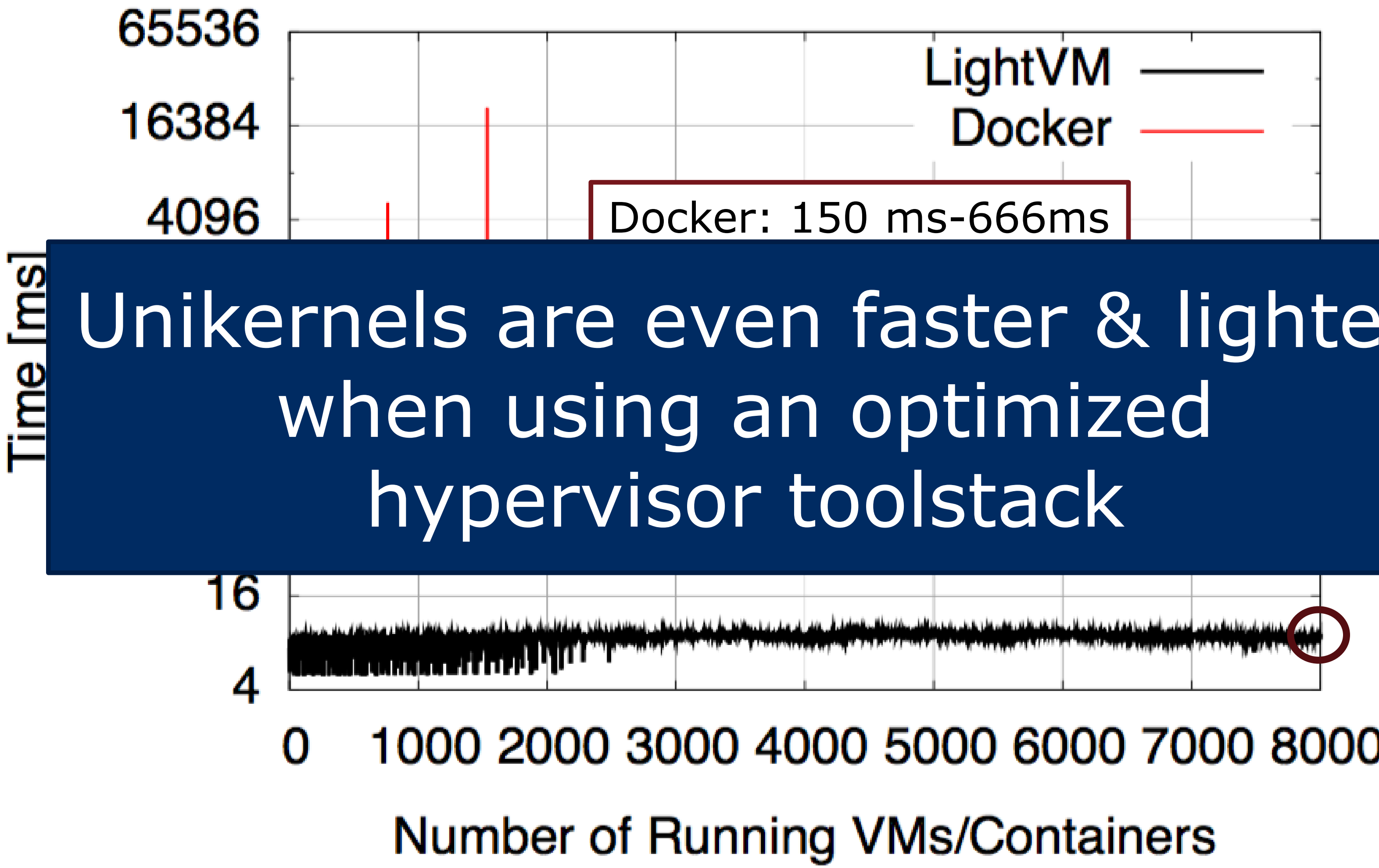
Unikernels similarly as fast as containers

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

Unikernel Gains

Fast instantiation, destruction and migration time

- 10s of milliseconds
  (LigthVM [Manco SOSP 2017], Jitsu [Madhvapeddy, NSDI 2015])

Low memory footprint

- Few MB of RAM *(ClickOS [Martins NSDI 2014])*

High density

- 10k guests on a single server node *(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

- Less components exist in Unikernel
- Strong isolation by hypervisor

When do I want a Unikernel?

- Unikernels are applicable to many domains!
- Question is about differentiators:

Use cases	Differentiator	Differentiator	Use cases
Minimal SW stack Reactive VNFs, Lambda Functions, …	**Fast boot, migration destroy**	**Resource efficient**	Minimal SW stack Per-customer VNFs, IoT, MEC, …
Specialization NFV, MEC, …	**High performance**	**Mission critical**	Small code base → cheap verification Strong isolation Automotive, Industrial-grade, …

Challenges with Unikernels

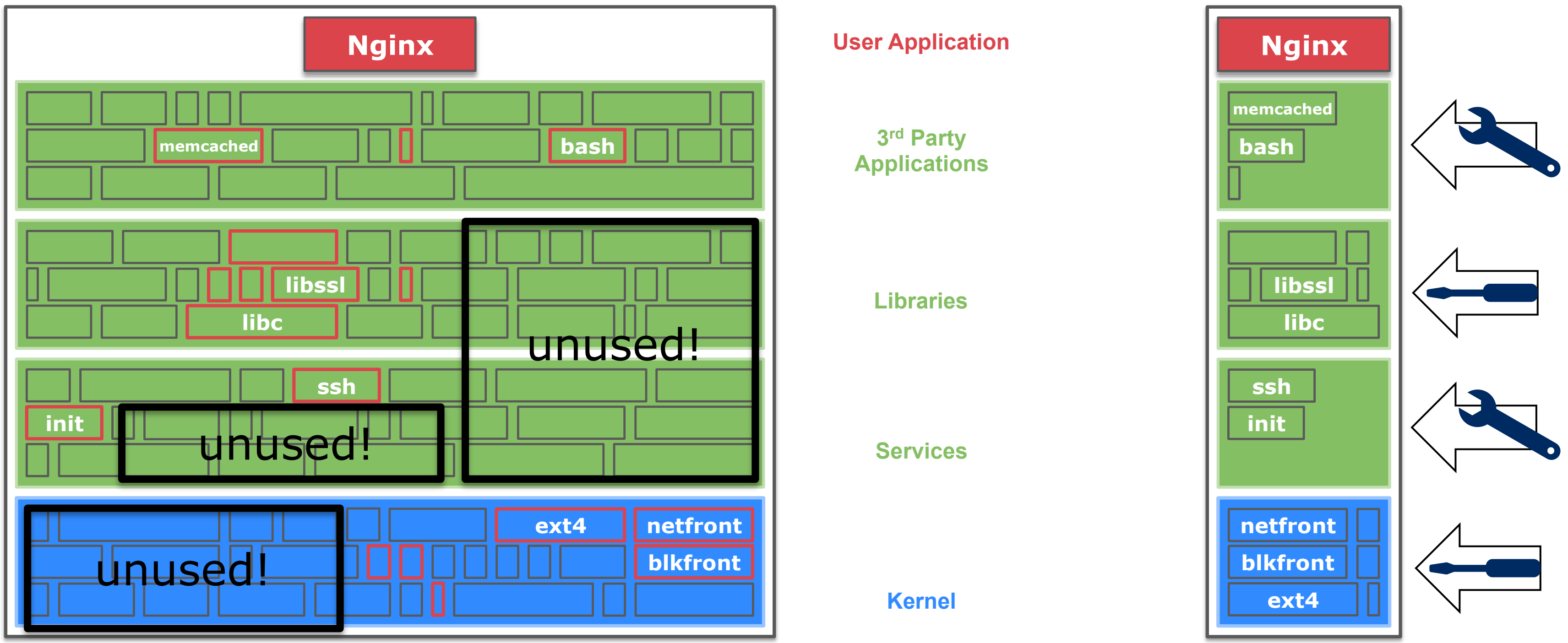
Cool, but why aren't they more widespread?

The Devil is in the Details

- Each optimized Unikernel is manually built
- Building takes several months or longer
- Process needed to be repeated for each target application

Specialization in Practice

- Standard OS/VM/container: ***lots*** of unnecessary code = ***lots*** of overhead!
- Unikernel: only what's needed is there ***but lots*** of development time!

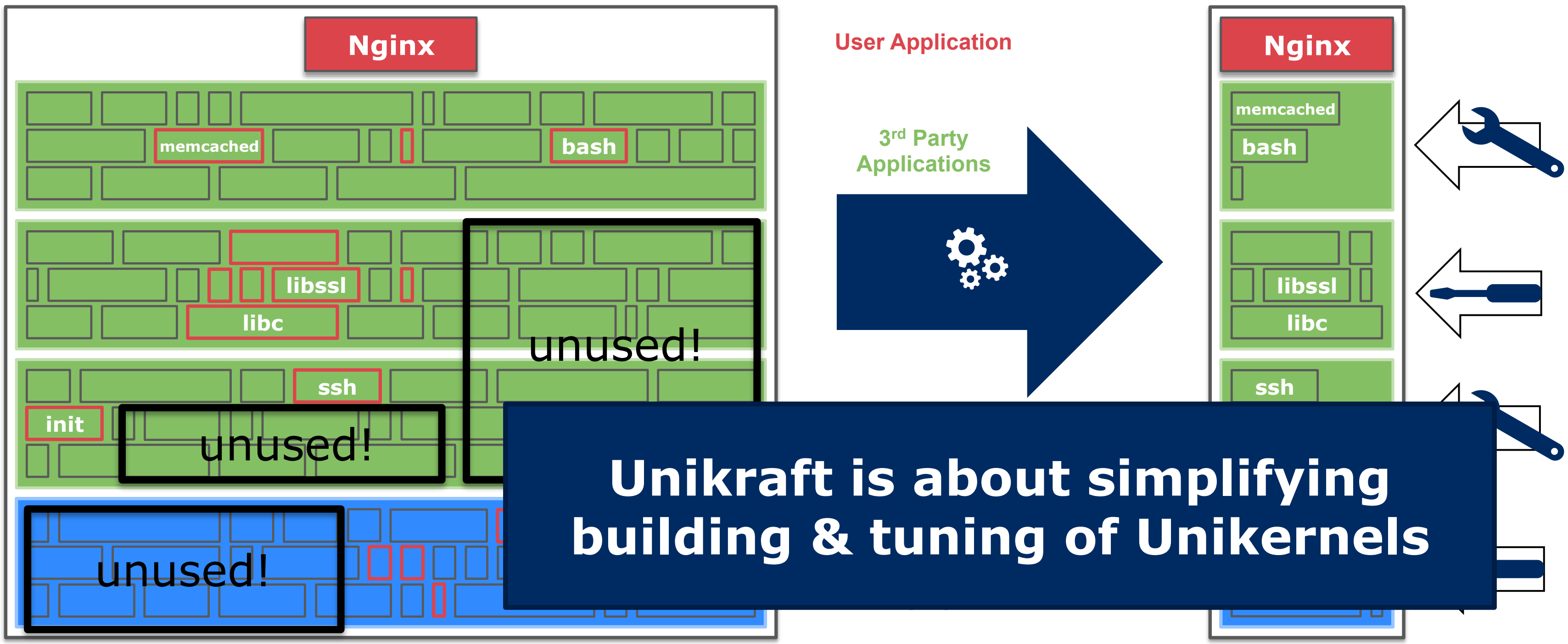


General Purpose OS

Unikernel

Specialization in Practice

- Standard OS/VM/container: ***lots*** of unnecessary code = ***lots*** of overhead!
- Unikernel: only what's needed is there ***but lots*** of development time!



Unikraft is about simplifying building & tuning of Unikernels

General Purpose OS

Unikernel

Unikraft

A Unikernel Build Framework

Unikraft - A Unikernel Framework

Motivation

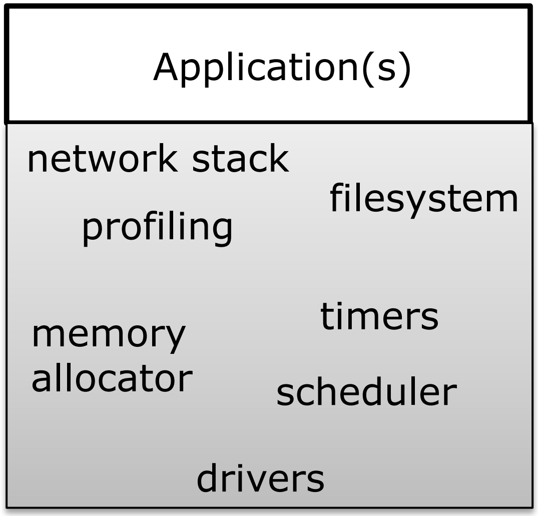
- Core principle: Support everyone's use case!
- Simplify building and optimizating
- Concentrate efforts of each Unikernel project to a single base

Our Approach

- Concept: “Everything is a library”
- Decomposed OS functionality as libraries
- Unikraft’s 2 components:
  - Library Pool
  - Build Tool

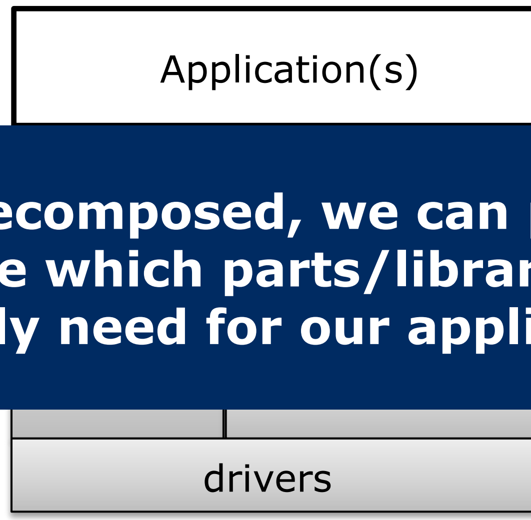
Operating System Decomposition

- Standard operating systems are ***monolithic:*** they are not modular so they are ***not*** designed for getting separated into their parts



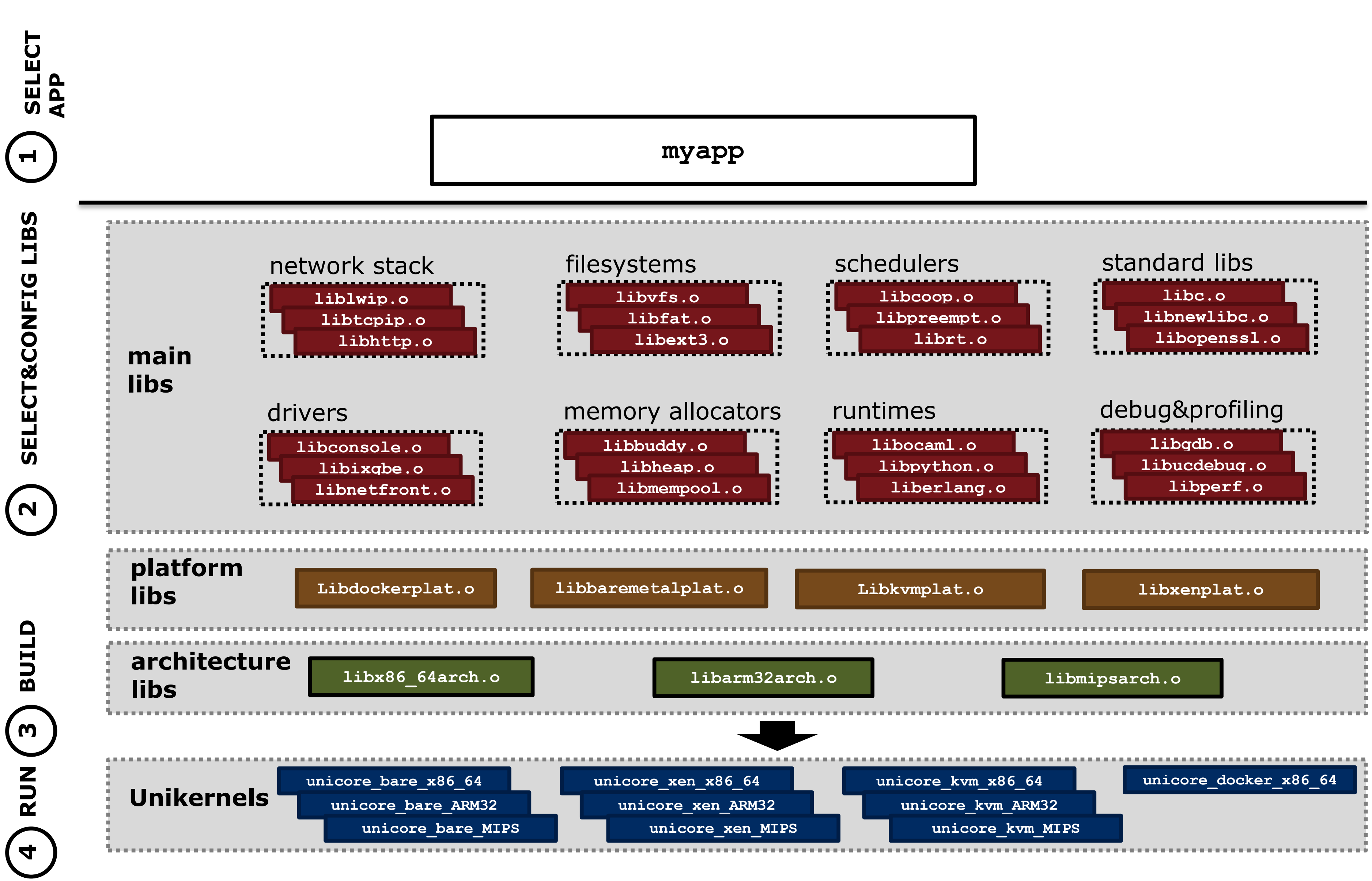
Operating System Decomposition

- Challenge: Have decomposed OS functionality as libraries, i.e., break apart an operating system



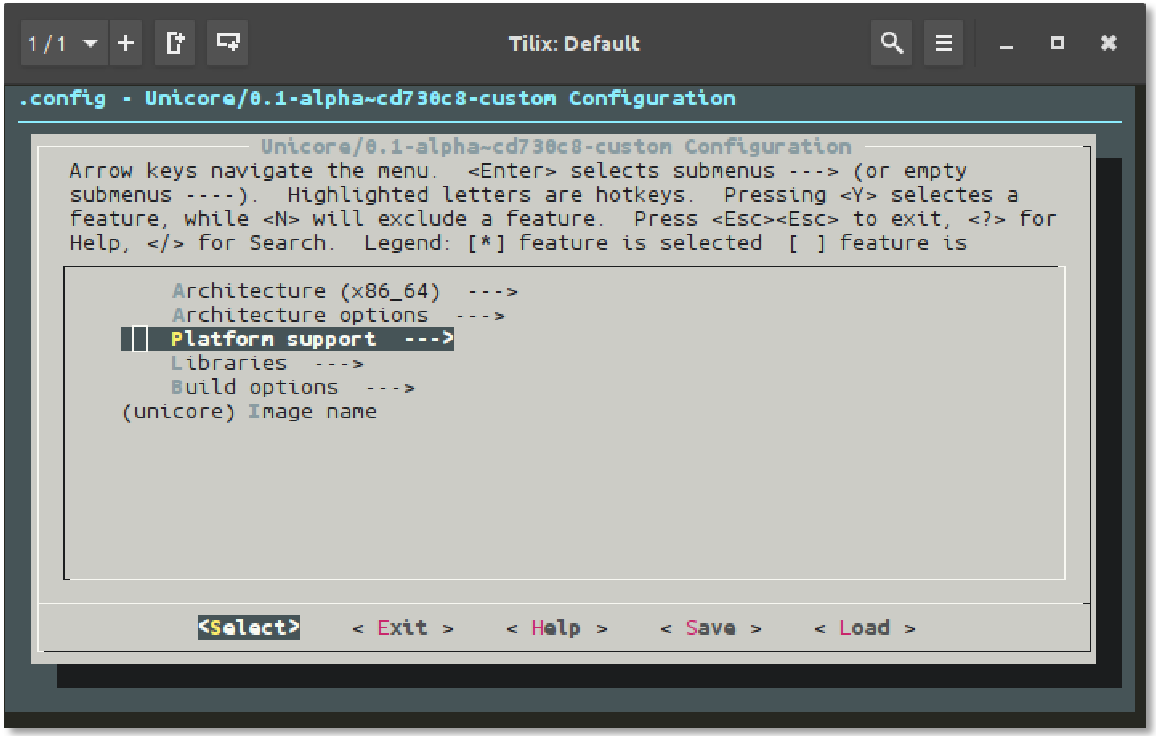
Once decomposed, we can pick and choose which parts/libraries we actually need for our application!

1) Library Pool



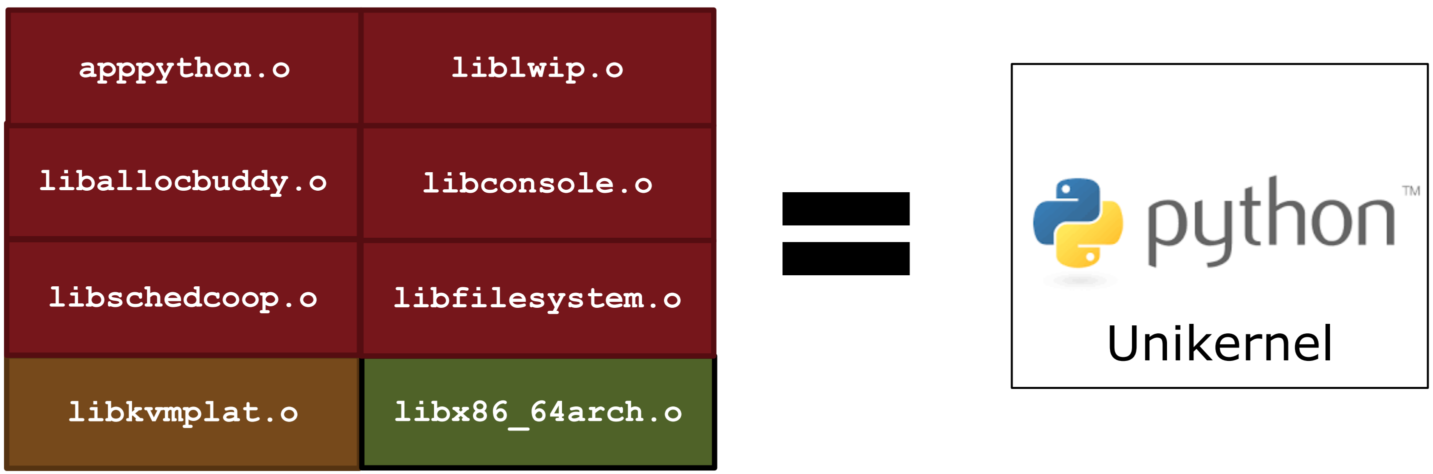
2) Build Tool

- KConfig based and Makefile "Magic"
- 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`"



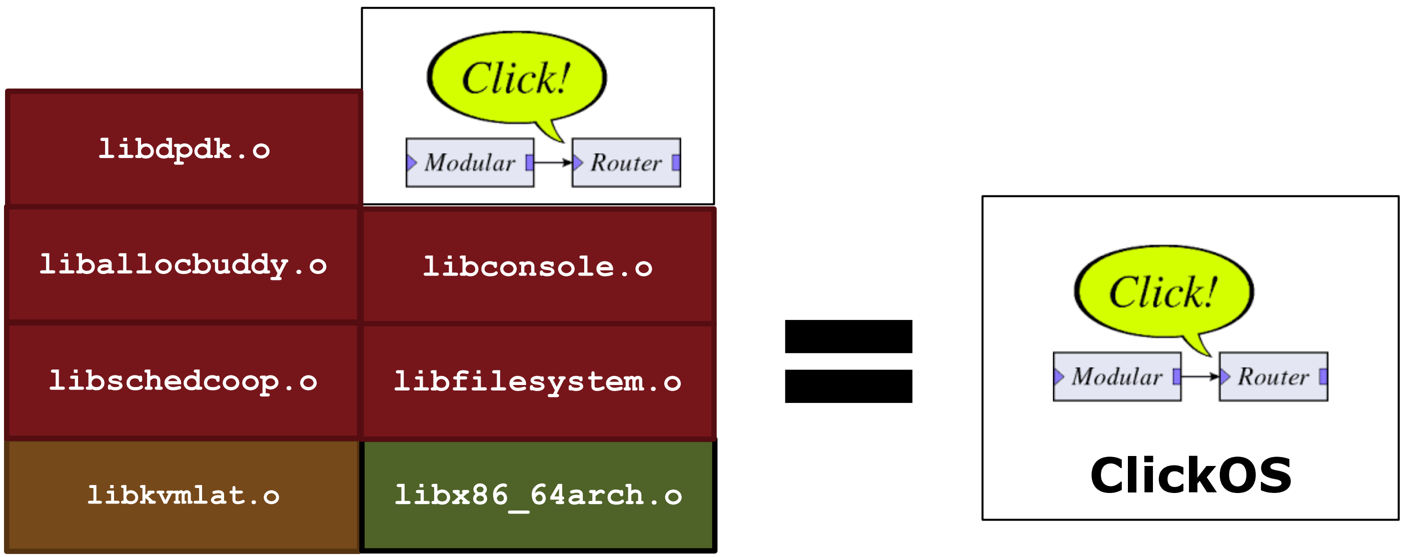
Example System

Python Unikernel for KVM on x86_64



Example System

VNF Unikernel for KVM on x86_64



Unikraft 0.2 Titan

Current Status

1) 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

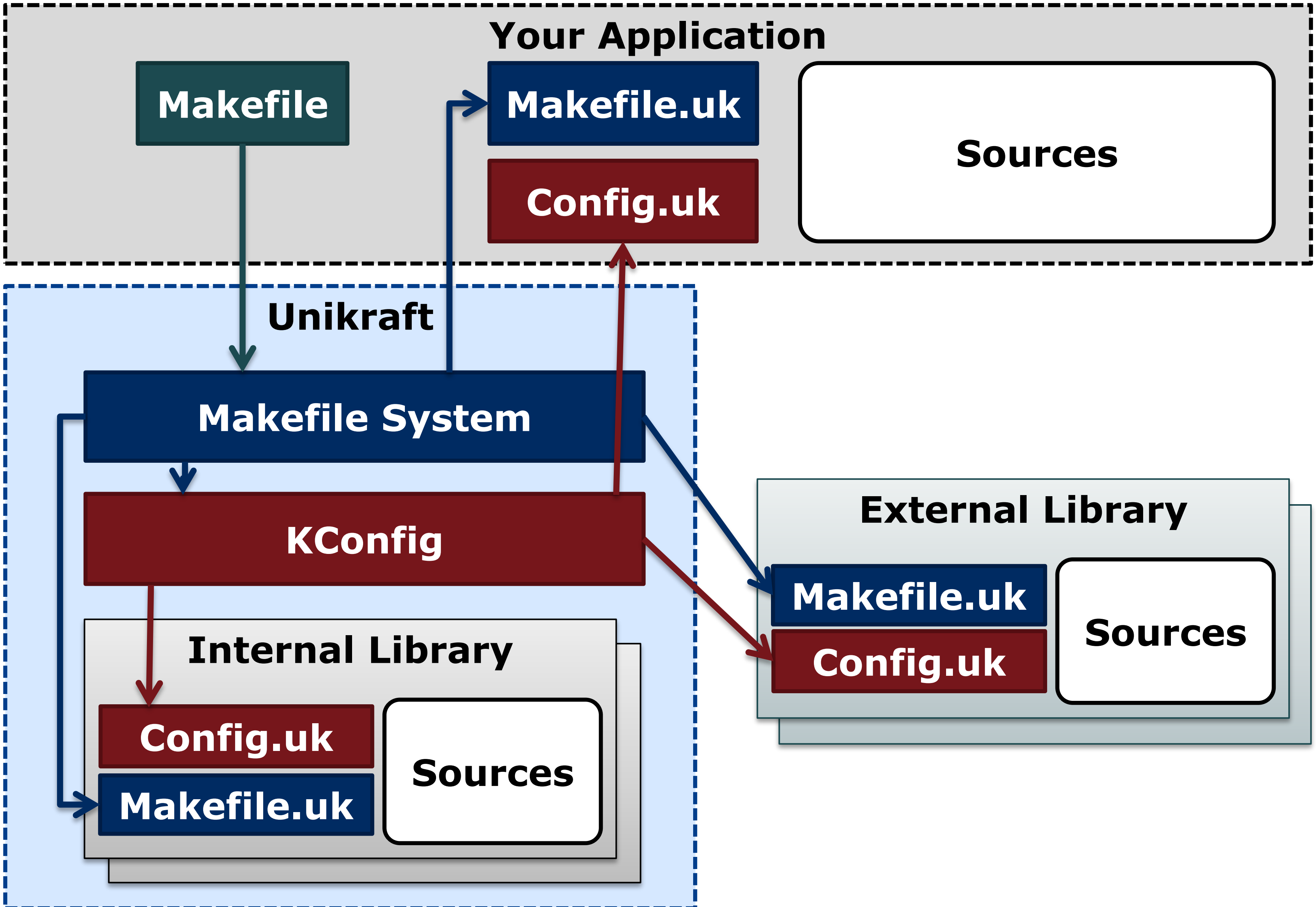
- libarmmath
  - 64bit arithmetics on ARMv7

Platform Libraries

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

More is coming!

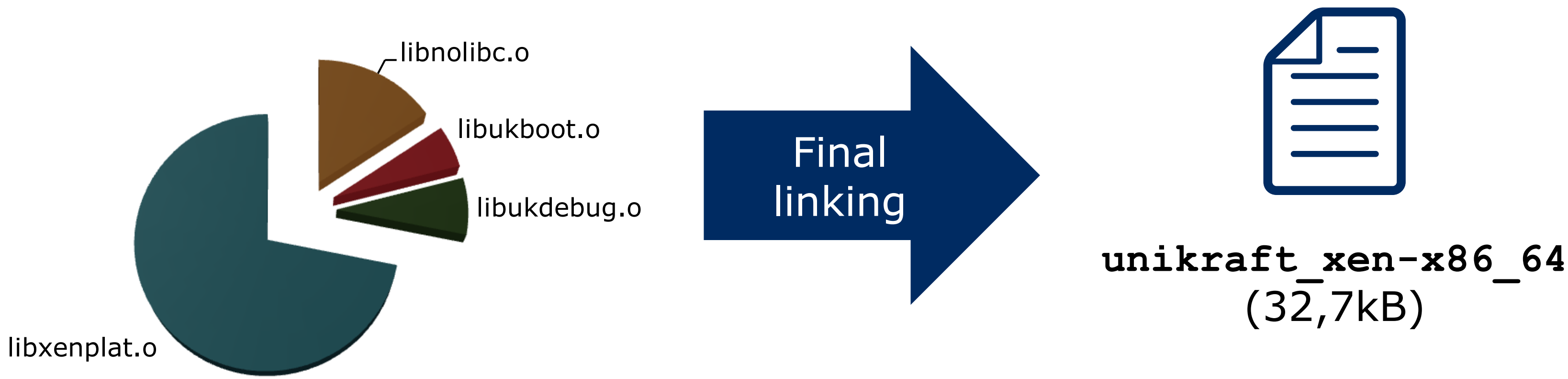
2) Build System



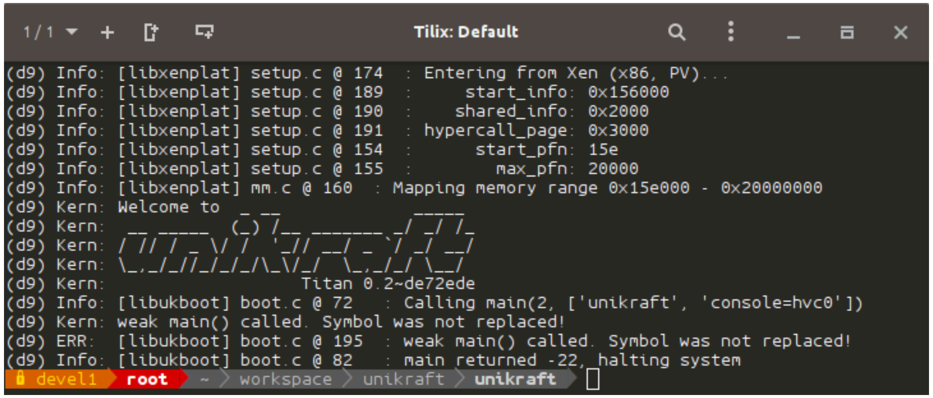
An Baseline Example…

- *Xen PV x86_64 binary*
- Compiles to a 32.7kB image

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



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




```
(d9) Info: [libxenplat] setup.c @ 174  : Entering from Xen (x86, PV)...
(d9) Info: [libxenplat] setup.c @ 189  :      start_info: 0x156000
(d9) Info: [libxenplat] setup.c @ 190  :     shared_info: 0x2000
(d9) Info: [libxenplat] setup.c @ 191  : hypercall_page: 0x3000
(d9) Info: [libxenplat] setup.c @ 154  :       start_pfn: 15e
(d9) Info: [libxenplat] setup.c @ 155  :         max_pfn: 20000
(d9) Info: [libxenplat] mm.c @ 160  : Mapping memory range 0x15e000 - 0x20000000
(d9) Kern: Welcome to  _ __             _____
(d9) Kern:  __ _____  (_) /__ _______ _/ __/ /_
(d9) Kern: / // / _ \/ /  '_// __/ _ `/ _/ __/
(d9) Kern: \_,_/_//_/_/_/\_\/_/  \_,_/_/ \__/
(d9) Kern:                   Titan 0.2~de72ede
(d9) Info: [libukboot] boot.c @ 72   : Calling main(2, ['unikraft', 'console=hvc0'])
(d9) Kern: weak main() called. Symbol was not replaced!
(d9) ERR:  [libukboot] boot.c @ 195  : weak main() called. Symbol was not replaced!
(d9) Info: [libukboot] boot.c @ 82   : main returned -22, halting system
devel1  root  ~  workspace  unikraft  unikraft
```

We want you!

It is Open Source!

Join Us!

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

- We need you for our ambiguous goals!

- 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!

Demo Time

It is real!

Unikraft Resources

Thanks!

- Wiki
  - https://wiki.xenproject.org/ (Search for Unikraft)
- Dokumentation
  - http://www.unikraft.org
- Sources (GIT)
  - http://xenbits.xen.org/gitweb/ (Namespace: Unikraft)
- Mailing list (shared with Mini-OS)
  - minios-devel@lists.xen.org
- IRC Channel on Freenode
  - #unikraft
- NEC-Team
  - 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.