Specialized and Secure Unikernels

GI Fachgruppentreffen SYS 2020

Simon Kuenzer

Sharan Santhanam

Felipe Huici

Intelligent Software Systems (ISS)
Data Science and System Platforms Division
NEC Laboratories Europe GmbH
Kurfürsten-Anlage 36, 69115 Heidelberg

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements no. 825377 ("UNICORE"). 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.

\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.

NEC Laboratories

NEC's Global R&D Activities

NEC Labs. Europe GmbH

Open innovation of core technologies

- **Data Science / AI**
- **Blockchain and IoT Security**
- **5G Networks**
- **ML & IoT Platforms**

NEC Labs. China

Developing core technologies

- Learning-based optimization
- Wireless communication networks

Central Research Laboratories (Japan)

The control tower for research

- Data science
- Security
- System platforms

~ 80 researchers from across the globe in **Heidelberg** + 20 other staff

Collaboration with top European universities and research institutes

www.neclab.eu

bs. Singapore

Developing solutions with governments and enterprise customers

- Public Safety and Security
- City Mobility
- Healthcare

NEC Labs. America

Developing core technologies

- Complex system analysis
- Advanced network and sensing IoT
- Security technology

~ 1000 staff

The Unikraft Core-Team

Felipe Huici

Chief Researcher

Simon Kuenzer

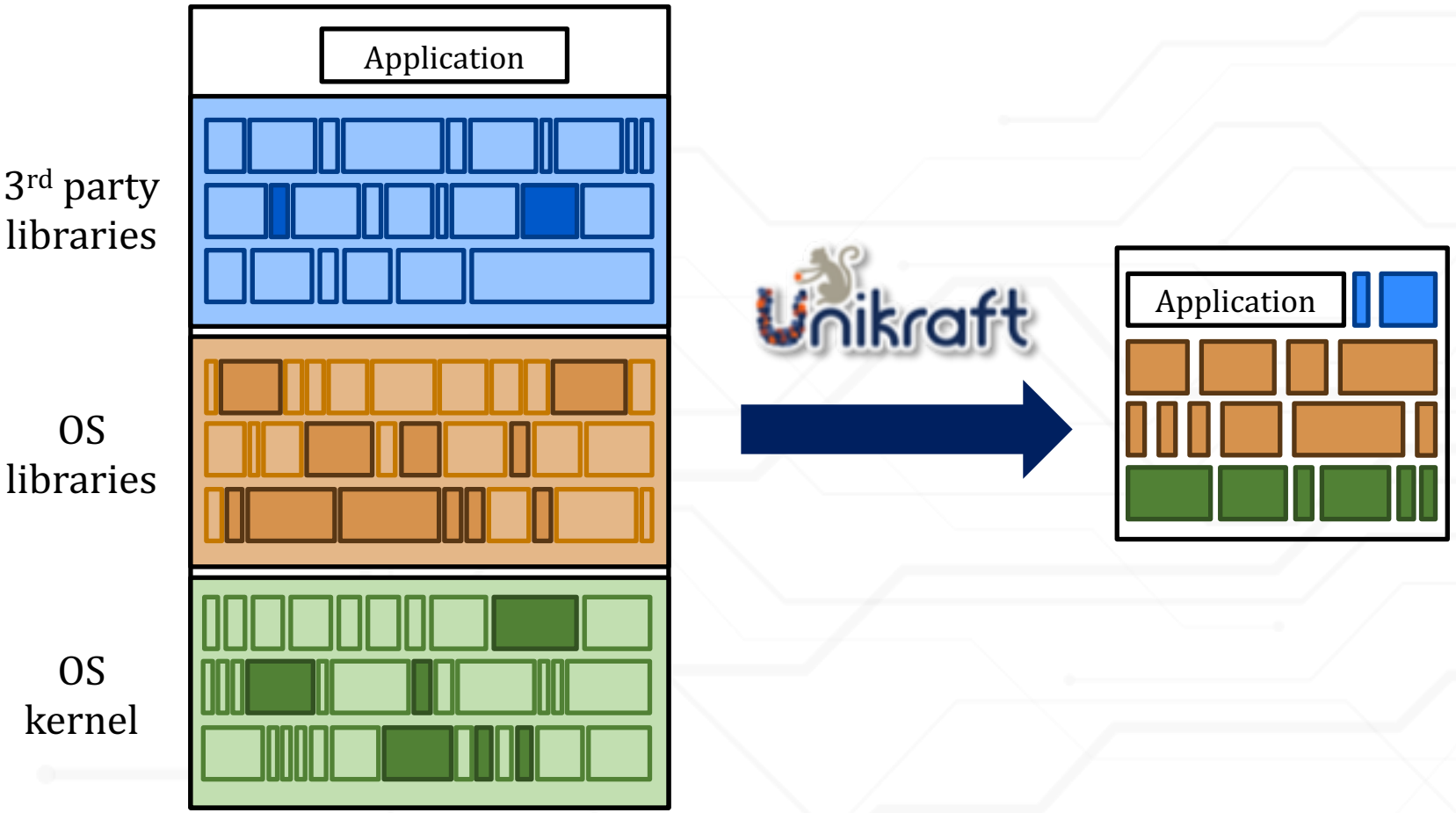
Senior Researcher

Sharan Santhanam

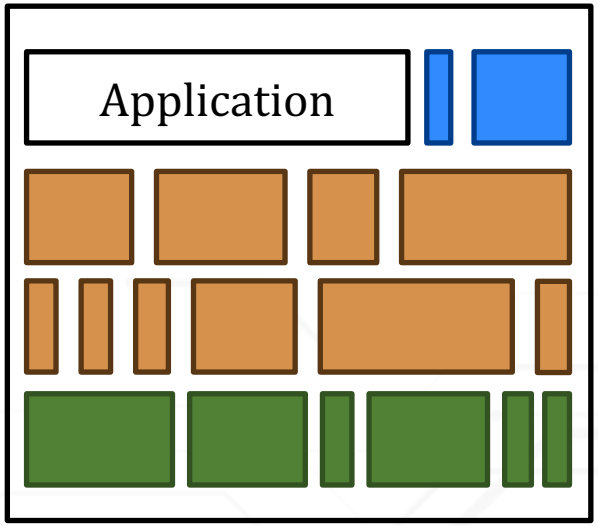
Software Specialist

Unikraft Primer

Full Stack Specialization



Unikernel

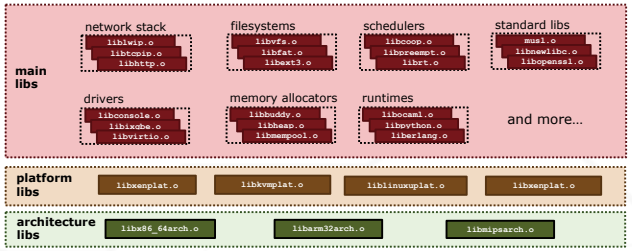


- One application → Flat and single address space
  - Isolation of multiple Unikernels/Applications by Hypervisor
- Thin kernel layer, *only what application needs*
  - Single monolithic binary *that contains OS and application*
- Further advantages from specialization
  - Performance and efficiency
  - Small Trusted Compute Base
  - Small memory footprint

Unikraft: A Specialized Library-Unikernel

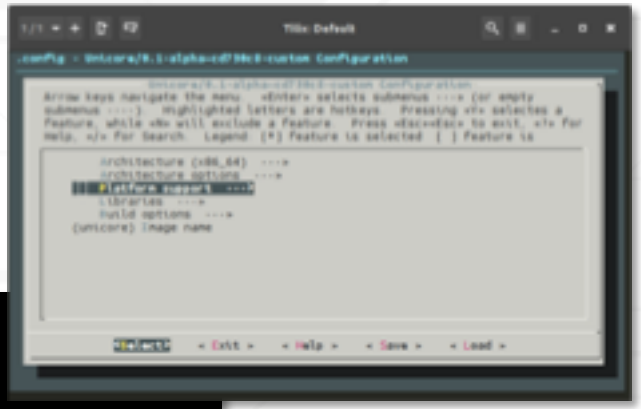
Everything is a (micro-)library

- Decomposed OS functionality
  - Schedulers, memory allocators, VFS, network stacks, …
- Architectures, platform support, drivers
  - Virtualization environments, bare-metal
- Application interfaces
  - POSIX, Linux system call ABI, language runtimes
- Specialization
  - Highly configurable
  - Only required features



Linux-style configuration and building

- `Kconfig`-based configuration
- `make`-based build system:
  Builds each library and links them



```
> make menuconfig
> make
```

Application Domains

Minimal SW Stack

Reactive vNFs,
Serverless,
Lambda functions,
IoT,
etc.

Boot, migration, and destroy times in ms

Resource efficient

Minimal SW Stack

Serverless,
(Per-customer) vNFs,
IoT,
Edge computing,
etc.

Specialization

HPC,
NFV,
MEC,
etc.

High performance

Mission Critical

Small code base
→ Low TCB
→ Potential cheaper verification

Automotive,
(Industrial) IoT,
etc.

Some numbers

Boot Times

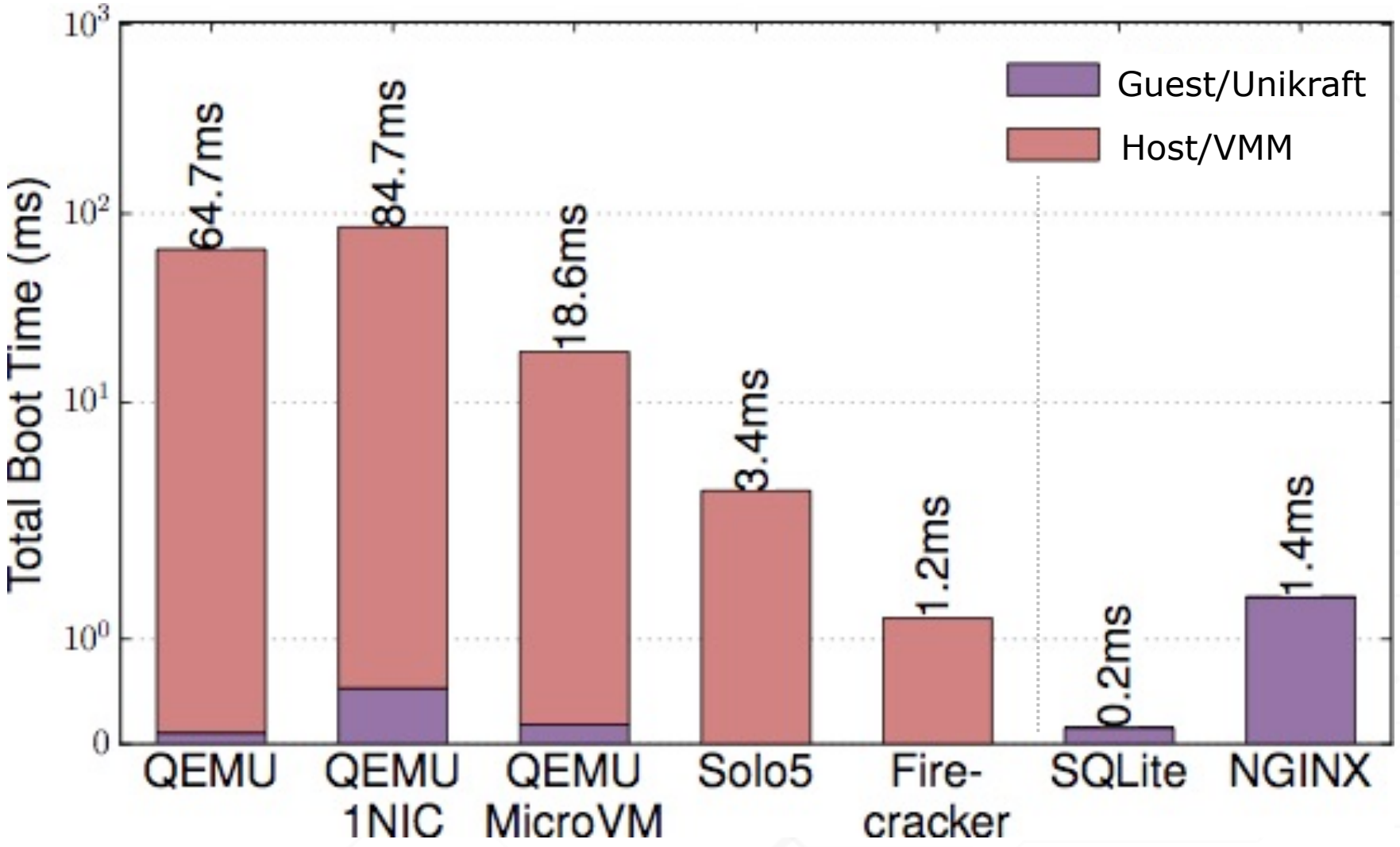
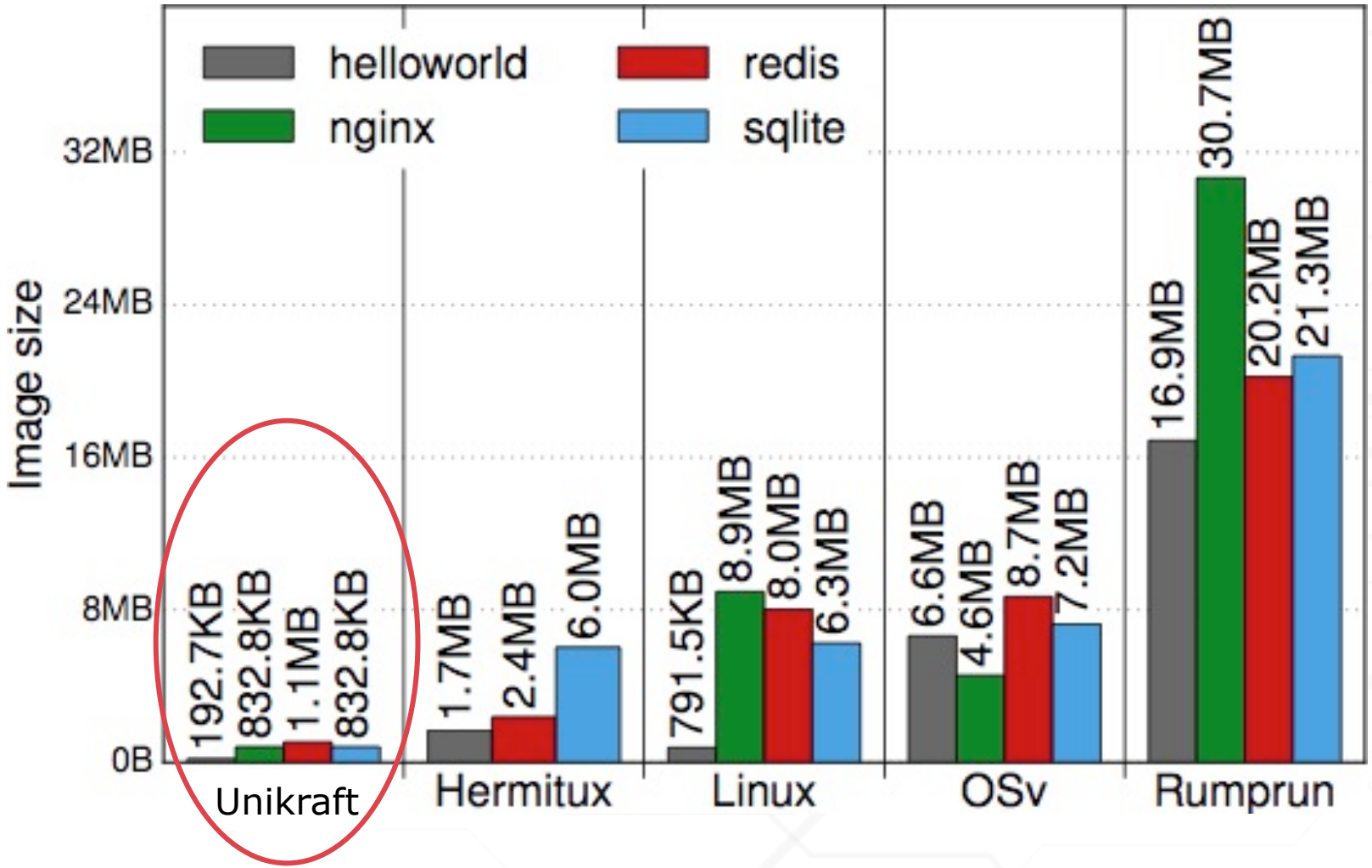
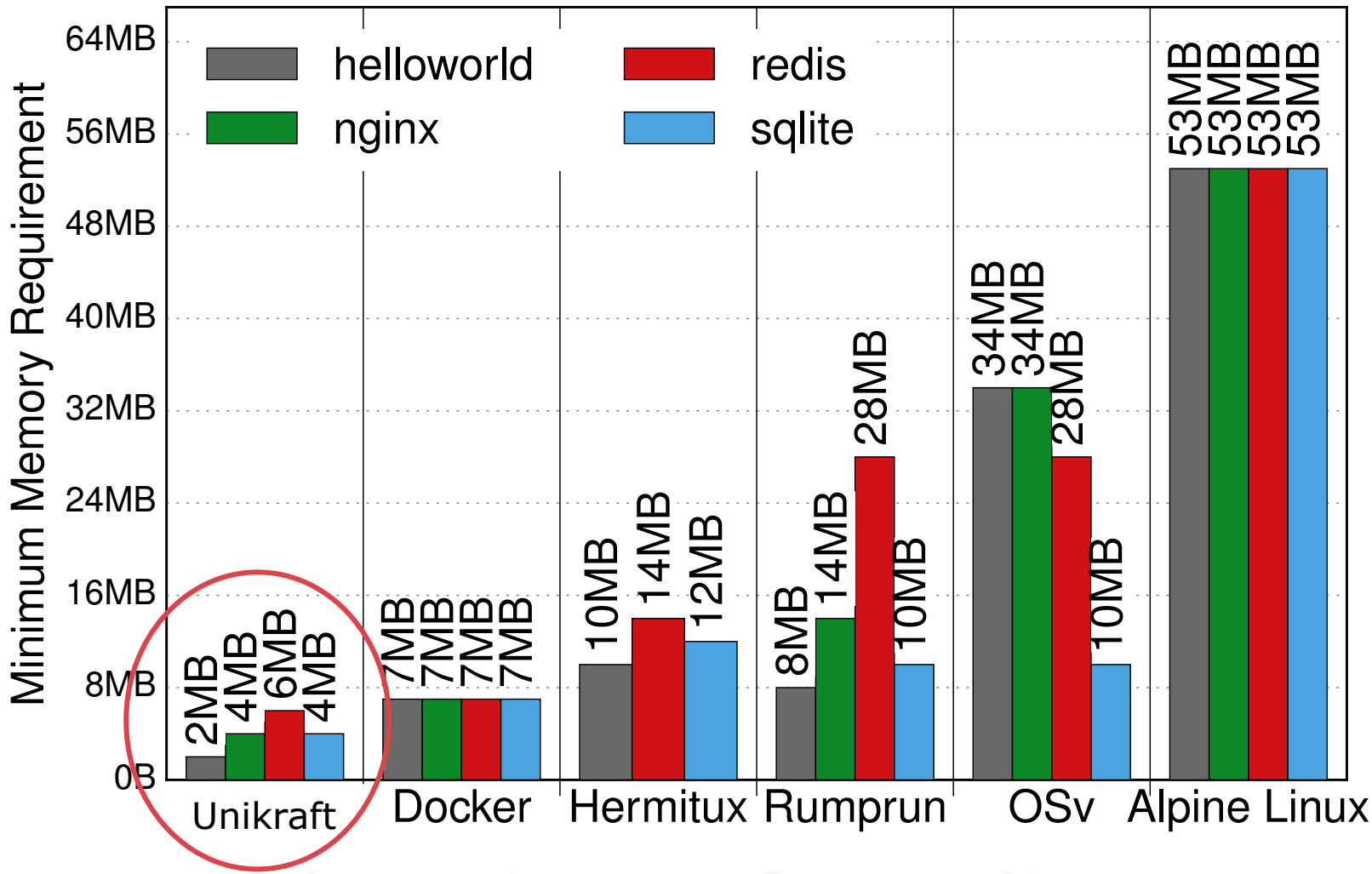


Image Sizes

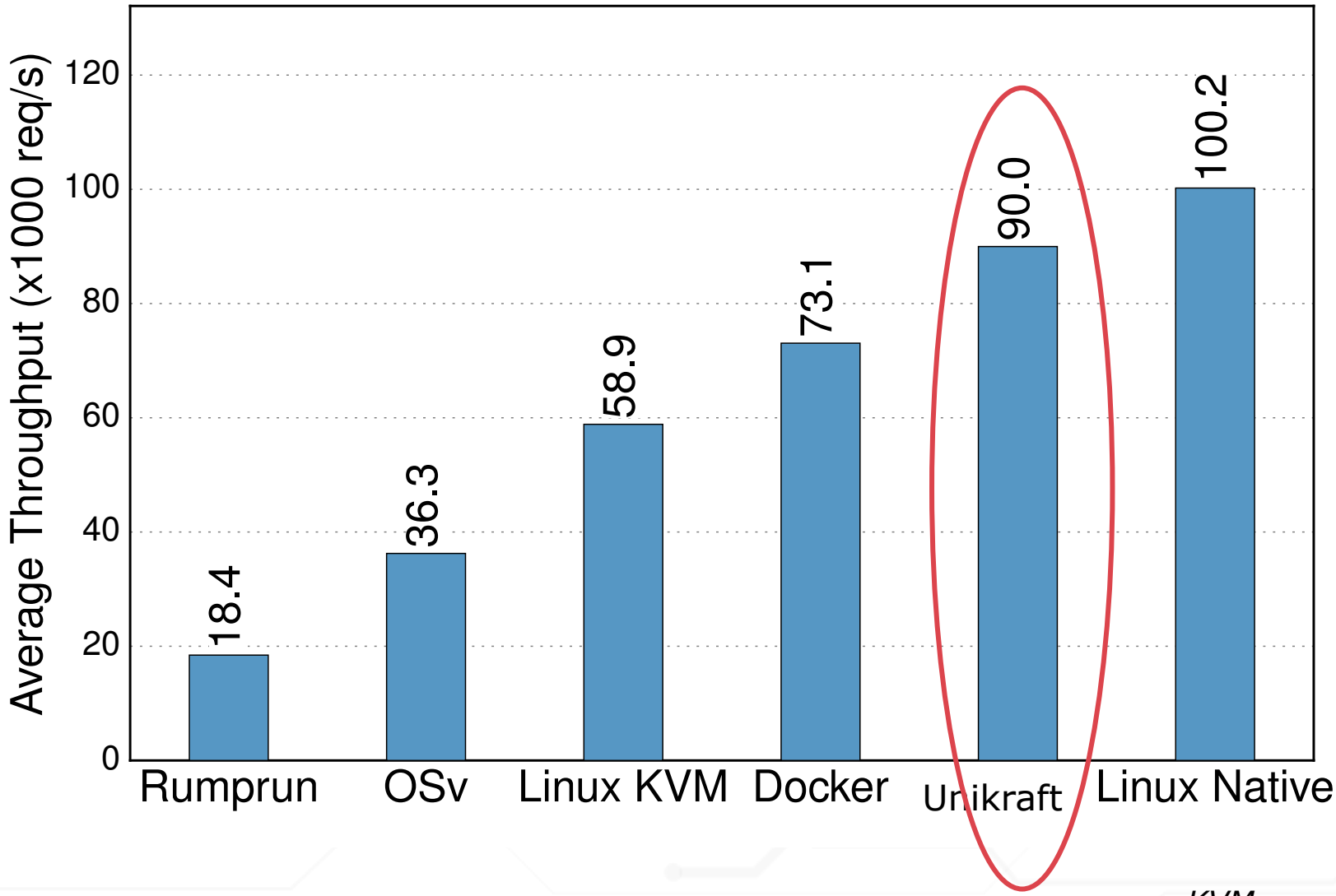


KVM guest image

Minimal Memory Requirement

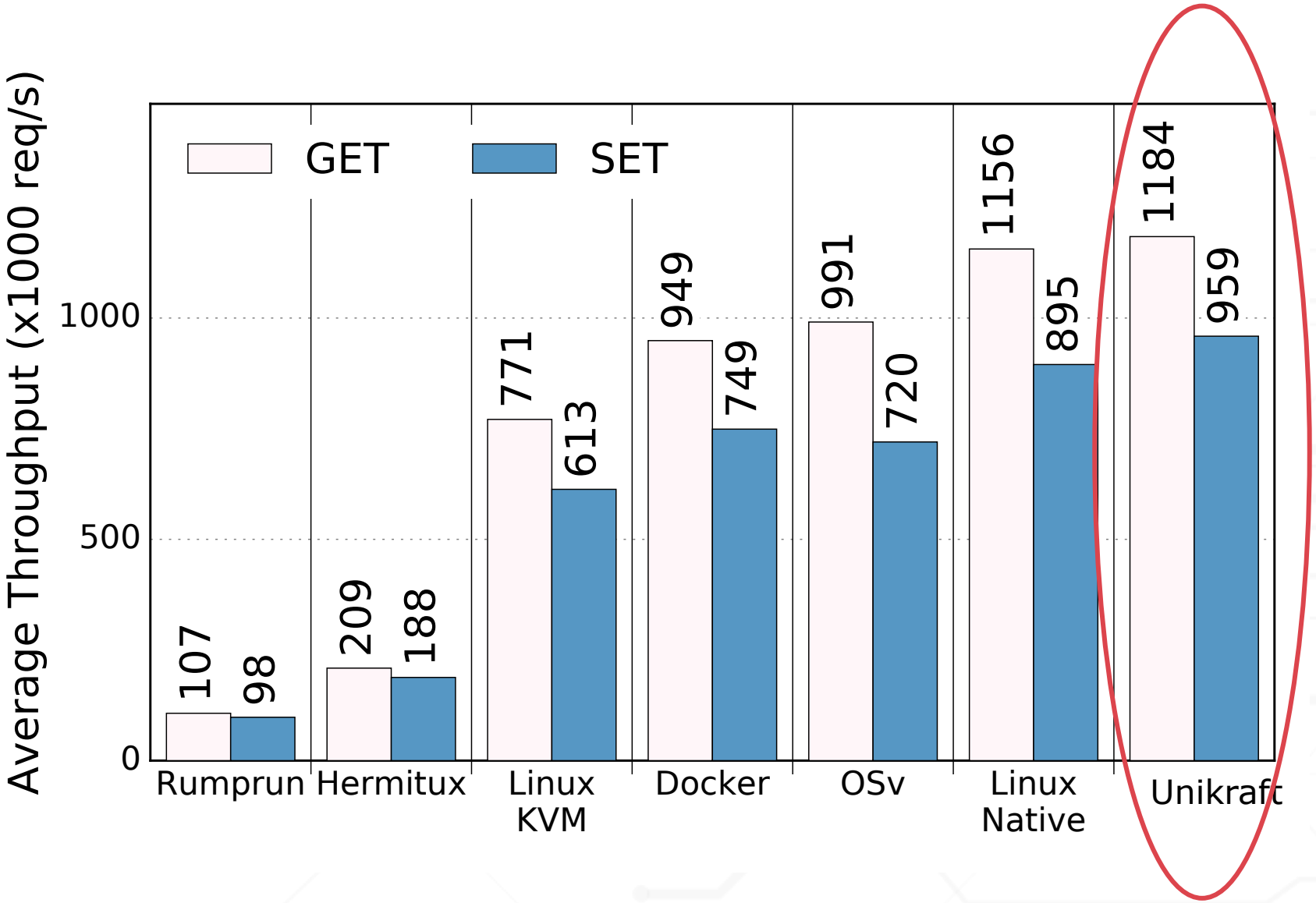


NGINX Performance (kReq/s)



KVM guest, 1 vcpu

Redis Performance



KVM guest, 1 vcpu

Unikraft and Security

Background

- April 2019
- NCC Group published "Assessing Unikernel Security"
- Focused on
  - Rumprun
  - IncludeOS
- Demonstrated vulnerability

Unikernels also need Mitigation Mechanisms!

- Most attack vectors can be mitigated with *common* techniques, *for example:*
  - Stack canaries
  - Page protection bits (Read/Write, Executable)
  - ASLR
  - Heap integrity checks
  - Compile-time obfuscation (e.g., *randstruct*)
  - Fuzzing and Testing

- Additional protection can be achieved with the help of hypervisor, *for example:*
  - Hypercall to freeze/lock (parts of) guest page table

- Other ideas
  - Formal-verified micro libraries
  - Mixed-language Unikernel: Language selection for each micro library *e.g., critical libraries written with type-safe languages*

Address Space Layout Randomization

Unikraft: Micro-libraries

- Randomization including “kernel” components feasible
- Full-stack ASLR

Offline by linker (poor-man ASLR)

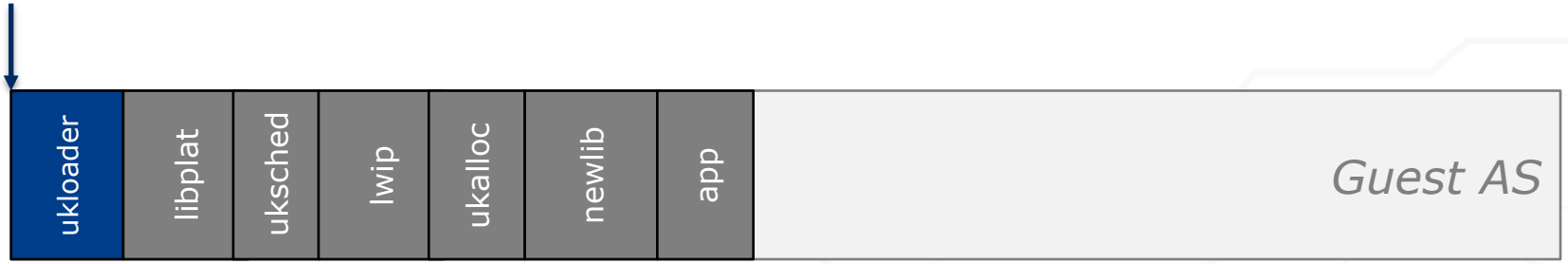
- Randomize base address(es) during link-time
- Same location for all instances ☹ but different for each build ☺

Online by early loader or VMM

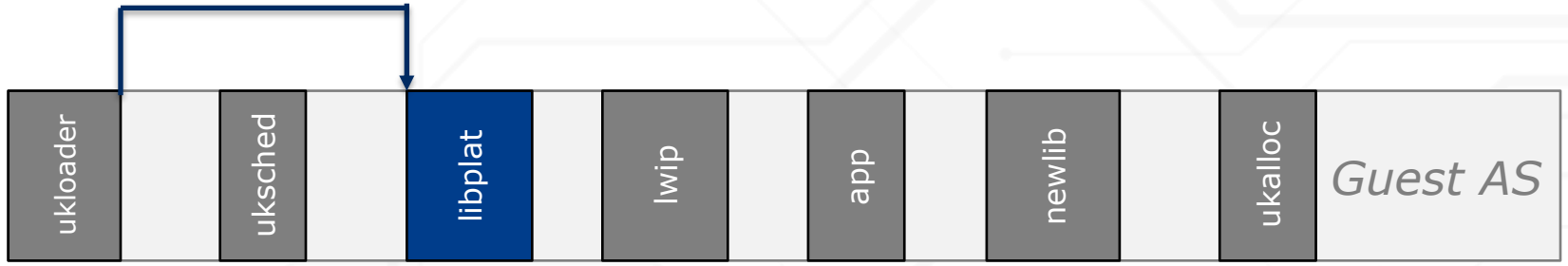
- Statically-linked PIE Unikernel
  - Relocate kernel image
- Dynamically-linked Unikernel
  - Relocate each micro-library

Full-stack ASLR with Early Loader

1. ASLR loader relocates/loads each micro library



2. Hand-over control flow to platform library to continue bootstrapping



3. Wipe loader during bootstrapping (optional)

Heap Integrity: Secure Memory Allocators

We are in Ring-0

→Direct access to low-level system components

→In principle, lower costs for accessing them, too (no KS/US-divide)

→Potentially, lower costs for page table modifications during allocations requests (e.g., relocation, alias pages)

In the works: Port of OSCAR[1]

- Lock-and-key mechanism
- Page permissions

➢ How will it be perform as part of a Unikernel?

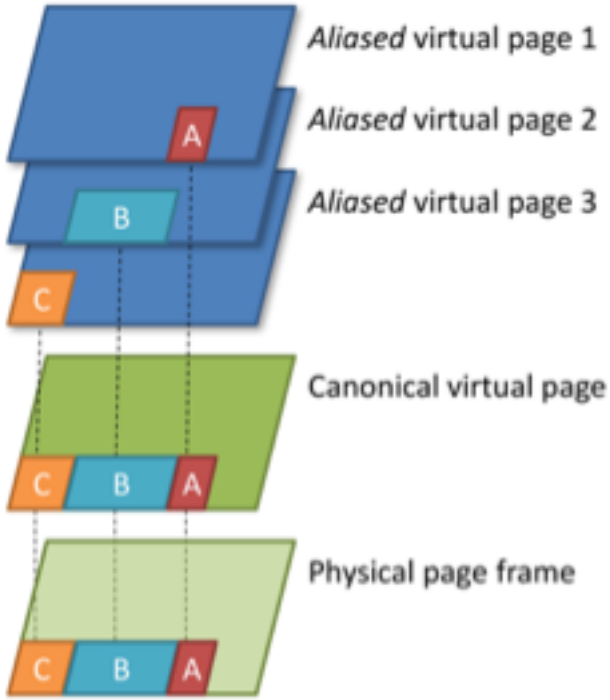


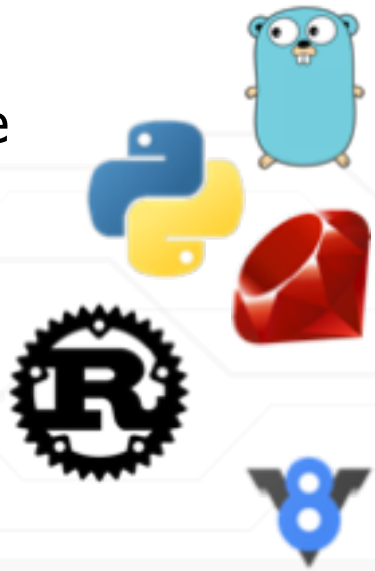
Image: [1]

[1] Oscar: A Practical Page-Permissions-Based Scheme for Thwarting Dangling Pointers
https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/dang

Other selected Topics

Language Runtimes

- Serverless, FaaS, and IoT
- Support applications written in higher-level language
  - C++, Rust, Go, Ruby, Javascript, Python, Lua, WebAssembly



- Example: WebAssembly for IoT
  - Seamless programming experience from browser, to cloud, to IoT
  - Trying with Mozilla and WebAssembly community

High Performance Networking

Package network function directly as VM (or bare-metal)

- Minimal OS overhead
- Minimal OS noise
- High networking performance & throughput
- Use cases: Cloud, Telco (VNFs)

First port of Intel DPDK

- Dataplane development Kit
- SDK for building high-performance VNFs
- Unikernel instead of kernel-bypassing
- Remove maintenance effort of hosting OS

DPDK
DATA PLANE DEVELOPMENT KIT

Other ported applications:

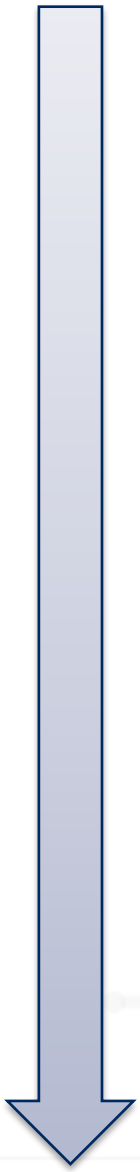
Redis

nginx

SQLite

It is Open Source!

Project Timeline



Early 2017: NEC-Internal project launch; 0.1

- Build system
- Initial port from Mini-OS and Solo5/KVM

Dec/2017: Public Launch; RELEASE-0.2 Titan

- As Xen Incubator project
- Arm32 Xen, x86 Xen, x86 KVM, x86 Linux
- Binary buddy allocator (heap)
- Cooperative scheduling

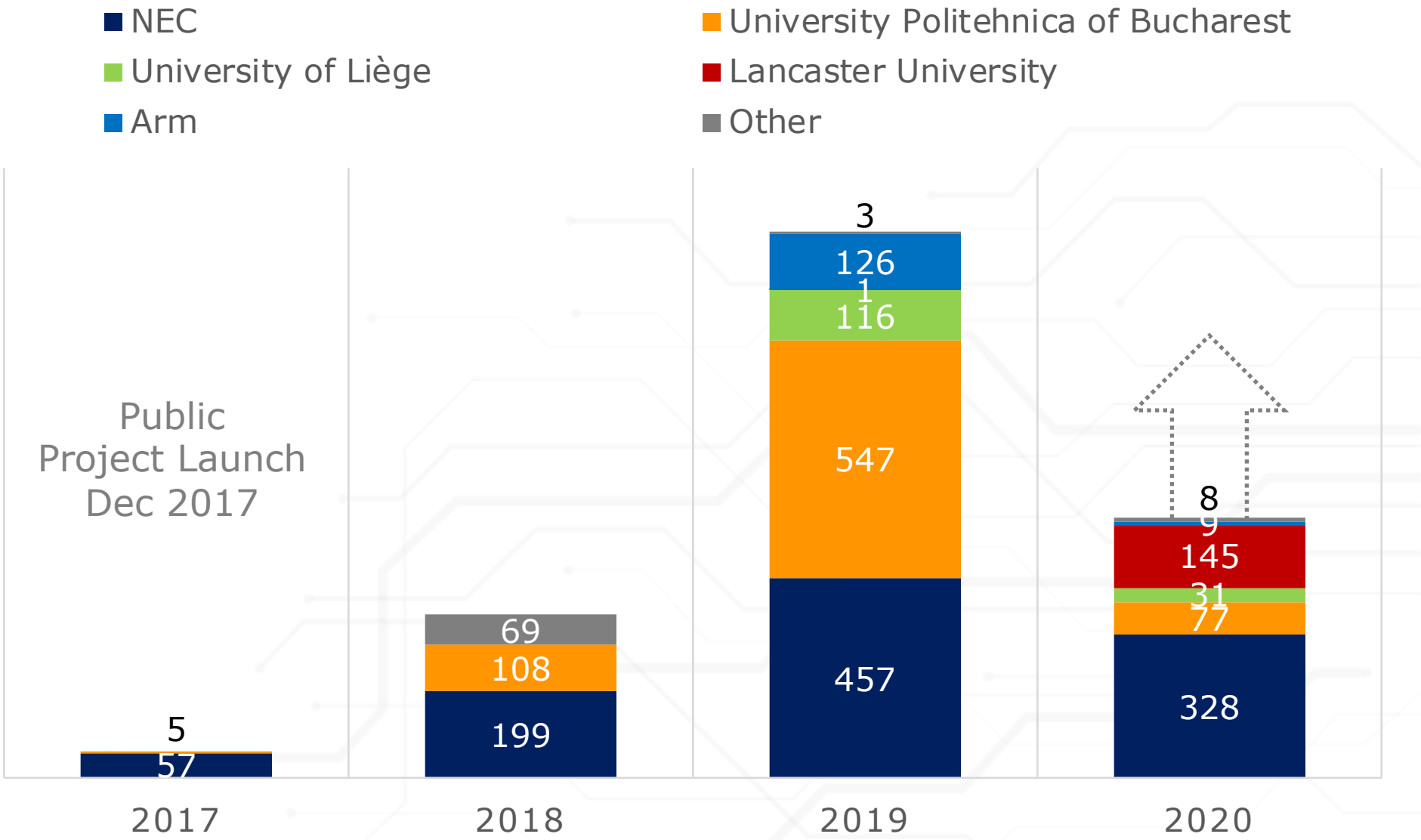
Feb/2019: RELEASE-0.3 Iapetus

- Arm64 support for KVM
- Networking (uknetdev, lwip, virtio-net)
- Initial VFS with in-RAM filesystem
- newlib

Feb/2020: RELEASE-0.4 Rhea

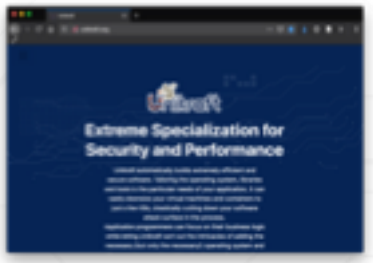
- Support for External platforms, starting with Solo5
- Language support: C++, Python, Go, Lua, JavaScript, WebAssembly, Ruby
- Tracepoint subsystem
- 9pfs filesystem support (Xen, KVM)
- Libraries: musl (initial) intel-intrinsics, libunwind, libuuid, pthread-embedded, compiler-rt, eigen, fp16, fxdiv, pthreadpool, etc.

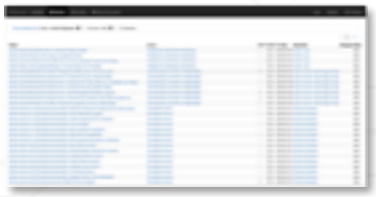
Contributions over Time (Signed-off-by's)



Snapshot from end of June 2020

Join us!





Project page

- www.unikraft.org

Sources

- github.com/unikraft

Documentation

- docs.unikraft.org

Contributing

- minios-devel@lists.xen.org (Shared mailing list)
- https://patchwork.unikraft.org


Orchestrating a brighter world
NEC