Rethinking the OS for Isolation Flexibility with FlexOS

Hugo Lefeuvre

The University of Manchester

FOSDEM 2022, 5th February 2022

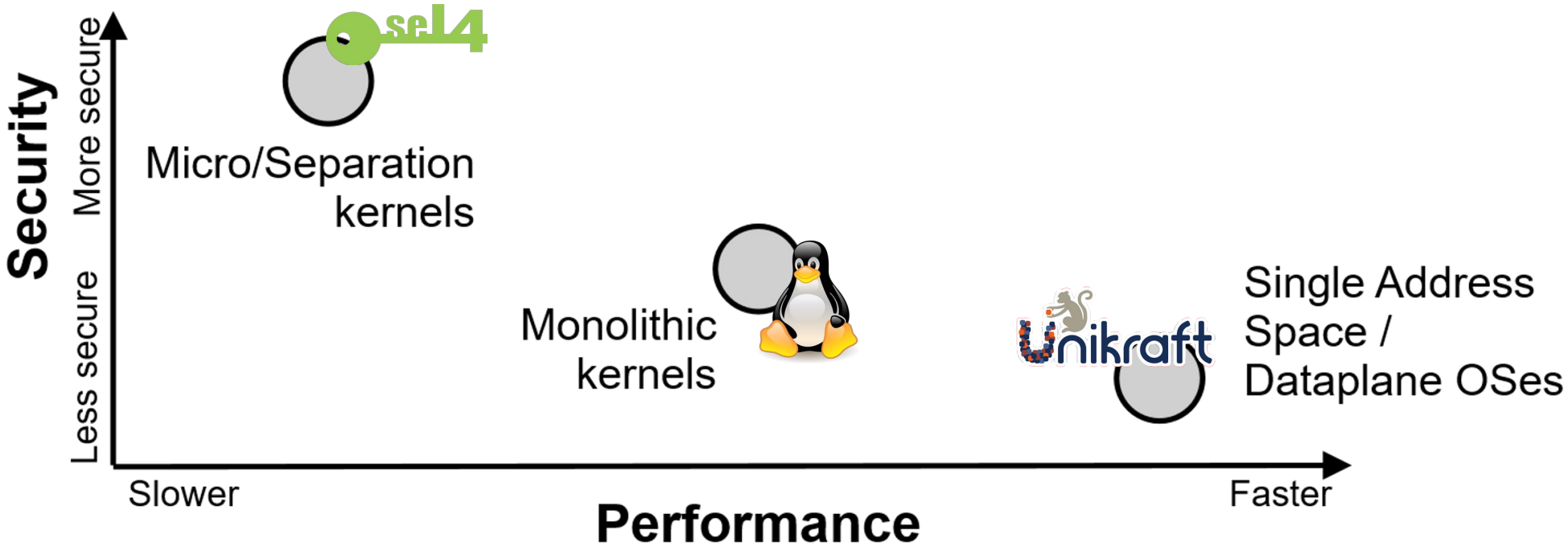
Paper @ HotOS'21

Paper @ ASPLOS'22

Current OS Designs

OS security/isolation strategies are **fixed** at design time!

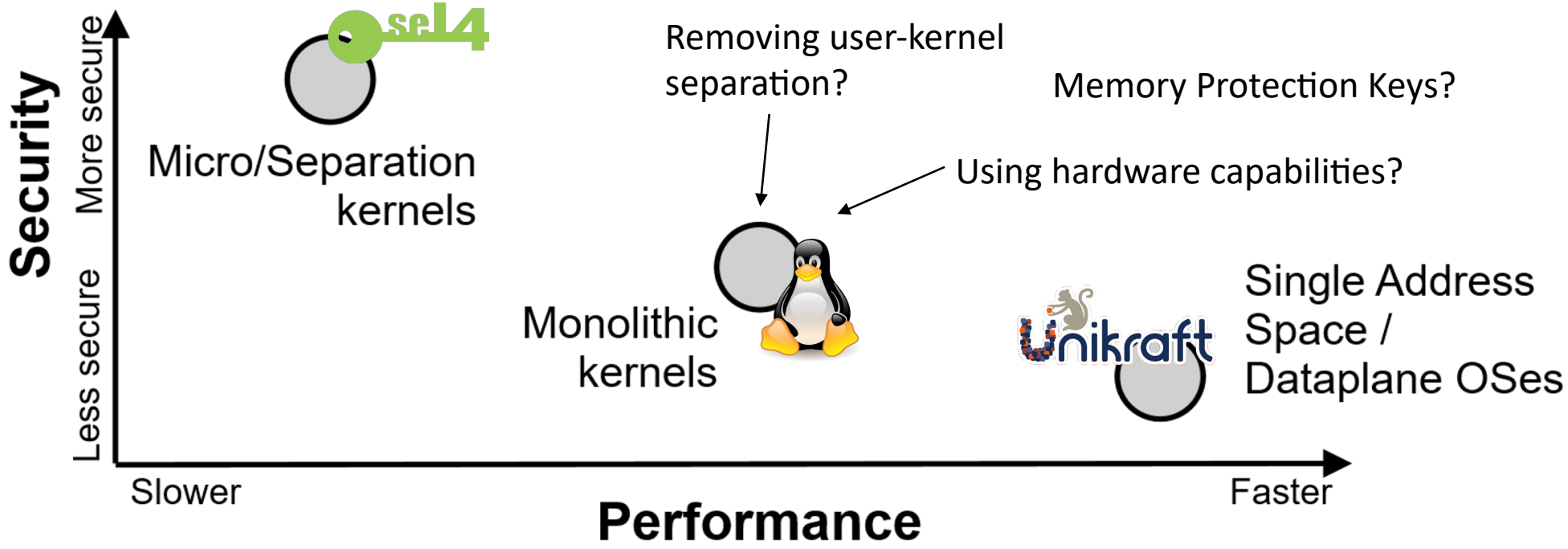
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



Current OS Designs

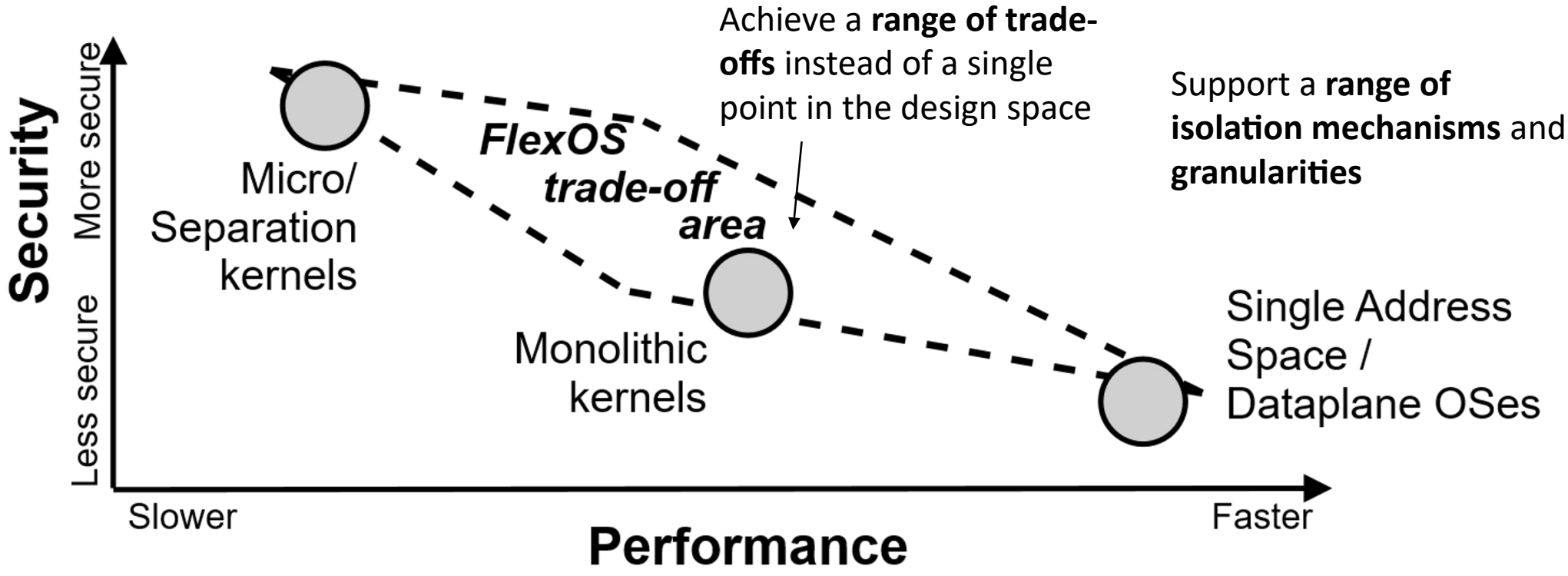
OS security/isolation strategies are **fixed** at design time!

Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)

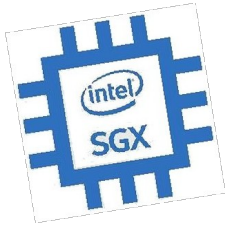


FlexOS: Flexible Isolation

Decouple security/isolation decisions **from the OS design**

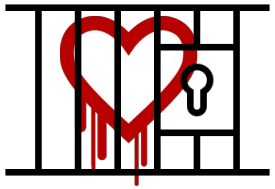


Other Use-Cases for Flexible Isolation



Deployment to heterogeneous hardware

Make optimal use of each machine/architecture's safety mechanisms with the same code



Quickly isolate vulnerable libraries

React easily and quickly to newly published vulnerabilities while waiting for a full patch

Incremental verification of code-bases

Mix and match verified and non-verified code-bases while preserving guarantees

FlexOS 101: Approach in 4 points

1 Focus on **single-purpose appliances** such as cloud microservices

...the more applications run together, the least specialization you can achieve

FlexOS 101: Approach in 4 points

1 Focus on **single-purpose appliances** such as cloud microservices

Full-system (*OS+app*) understanding of compartmentalization 2

Not "only application" or "only kernel":
consider everything and **specialize**

Embrace the **library OS philosophy:** everything is a library...
network stack, nginx, libopenssl, sound driver, etc.

FlexOS 101: Approach in 4 points

1 Focus on **single-purpose appliances** such as cloud microservices

Full-system (*OS+app*) understanding of compartmentalization 2

3 **Abstract away** the technical details of isolation mechanisms

Page table, MPK, CHERI, TEEs? Not the same guarantees,
but **a similar interface can be achieved**.

FlexOS 101: Approach in 4 points

1 Focus on **single-purpose appliances** such as cloud microservices

Full-system (*OS+app*) understanding of compartmentalization 2

3 **Abstract away** the technical details of isolation mechanisms

Flexibility must not **get into the way of performance** 4

FlexOS 101: Overview

Happy users

config.yaml

FlexOS Toolchain

① Input config

config.yaml

```
compartments:
- comp1:
  mechanism: intel-mpk
  default: True
- comp2:
   mechanism: intel-mpk
  hardening: [cfi, asan]
libraries:
- libredis: comp1
- libopenjpg: comp2
- lwip: comp2
```

"Redis image with two compartments, isolate libopenjpeg and lwip together"

FlexOS 101: Overview

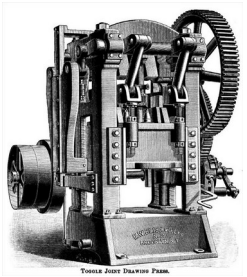
Happy users

config.yaml

① Input config

Isolation Backends

MPK | EPT | ...



② Select isolation mechanism ("Backend")

FlexOS Toolchain

Core Libraries

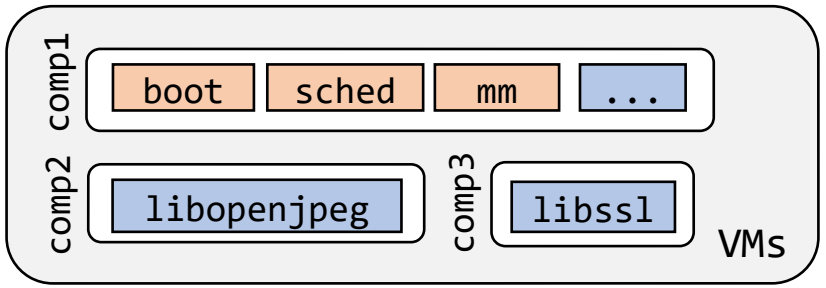
boot | sched | mm

Kernel & User Libs

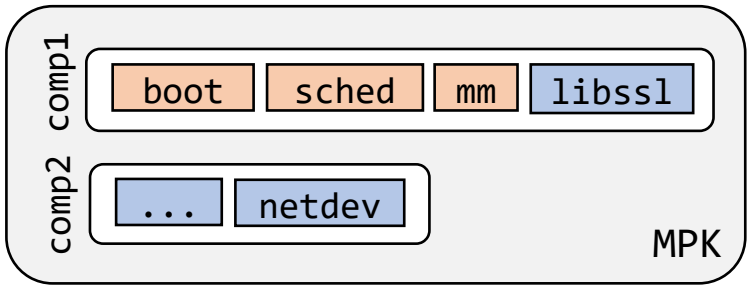
ramfs | vfs | ...

③ Select libraries (kernel and app), *rewrite*, and statically put in compartments

④ Generate image with appropriate isolation properties



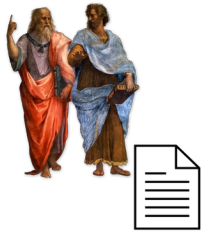
Possible Image 1



Possible Image 2

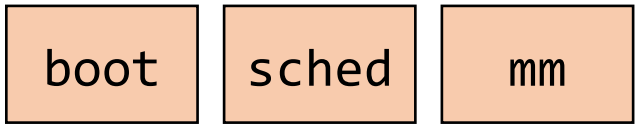
FlexOS 101: Mechanism Abstraction

Based on a **highly modular LibOS design** (Unikraft)

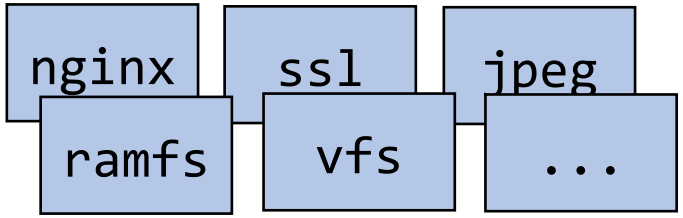


Such libOSes are composed of *fine-granular*, *independent* libraries

Core Libraries



Kernel & User Libs



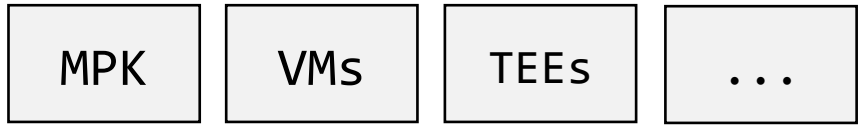
Reuse libraries as finest granularity of compartmentalization

"Pre-compartmentalize" them

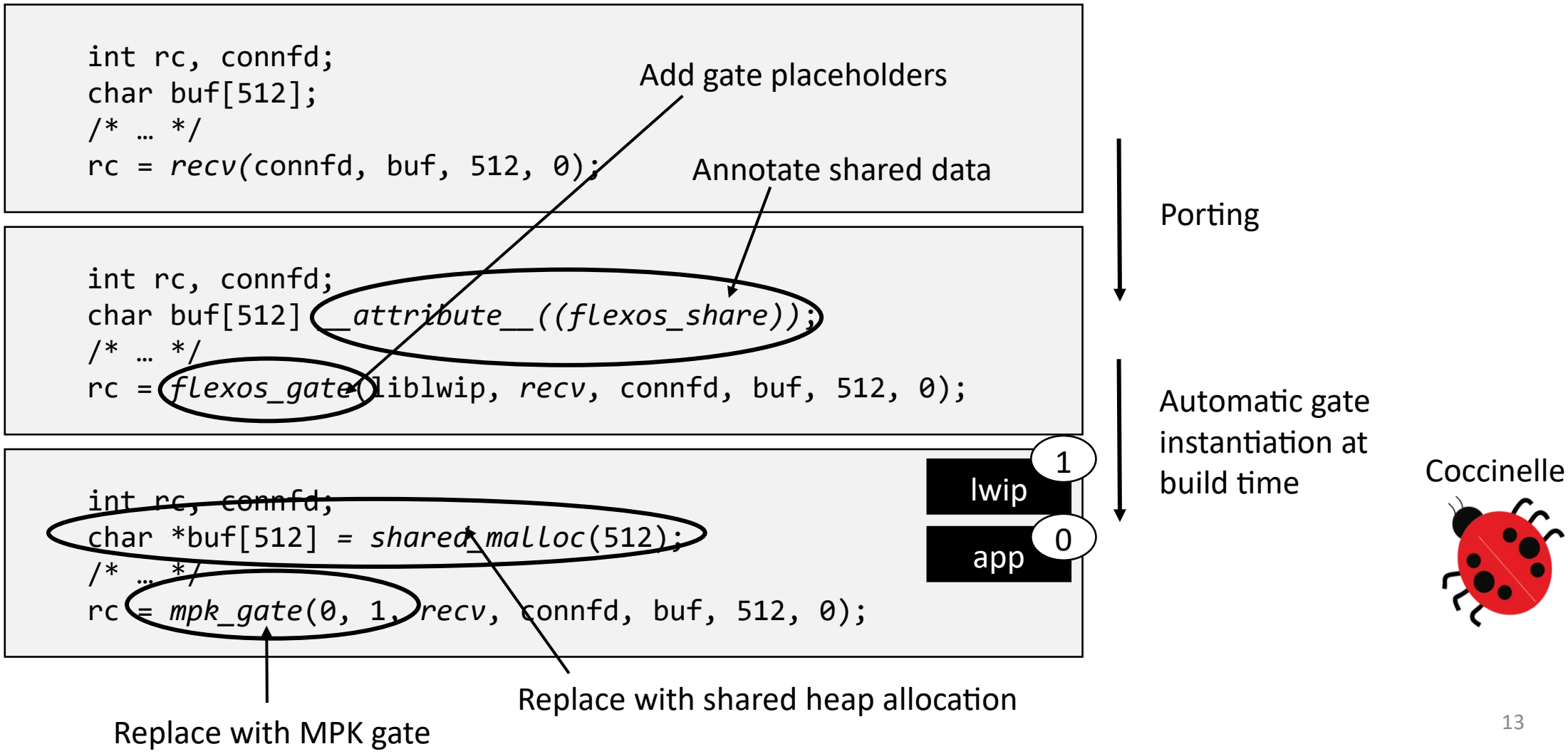
Cross-library **calls and shared data** are replaced by an **abstract construct** (**gates**, data sharing primitives)

Define them as part of the **FlexOS API**

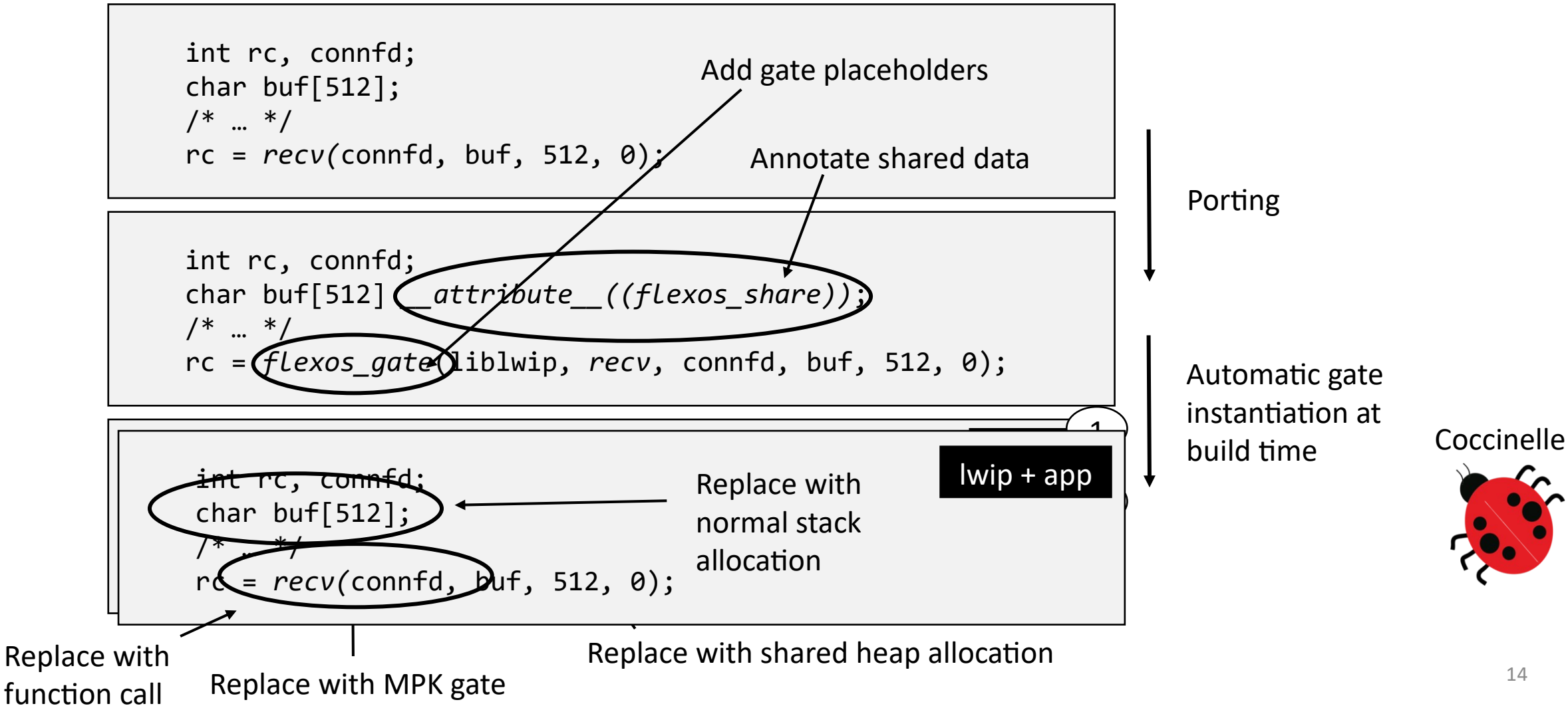
At build time, these abstract constructs are replaced with a particular implementation by the toolchain. These implementations are defined by the **backends**.



FlexOS 101: Compartmentalization API



FlexOS 101: Compartmentalization API



Prototype

Implementation **on top of Unikraft**

Backend implementations for **Intel MPK** and **VMs (EPT)**

Port of libraries: network stack, scheduler, filesystem, time subsystem

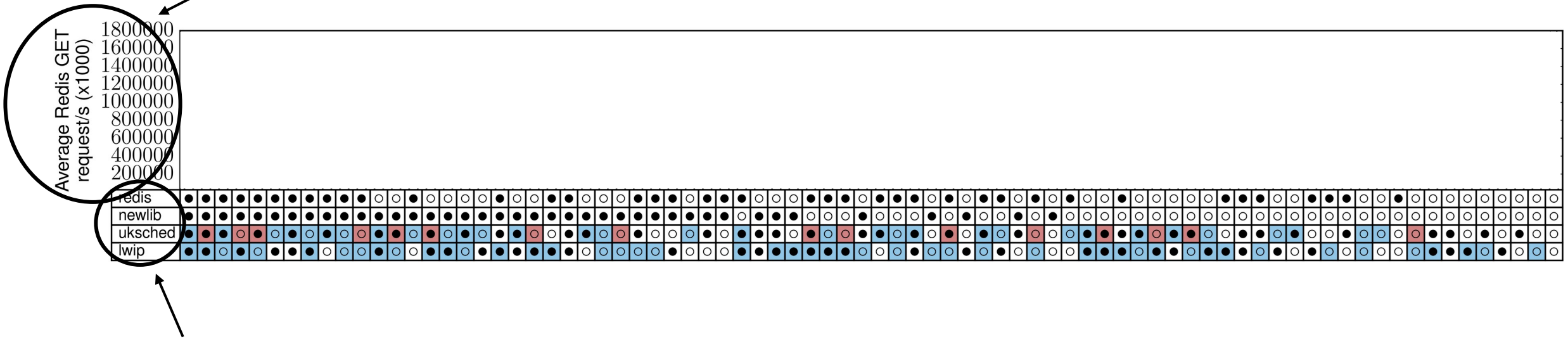
Port of applications: Redis, Nginx, SQLite, iPerf server

This talk: focus on demonstrating **flexibility and performance**

more results in our paper 🙂

Flexibility

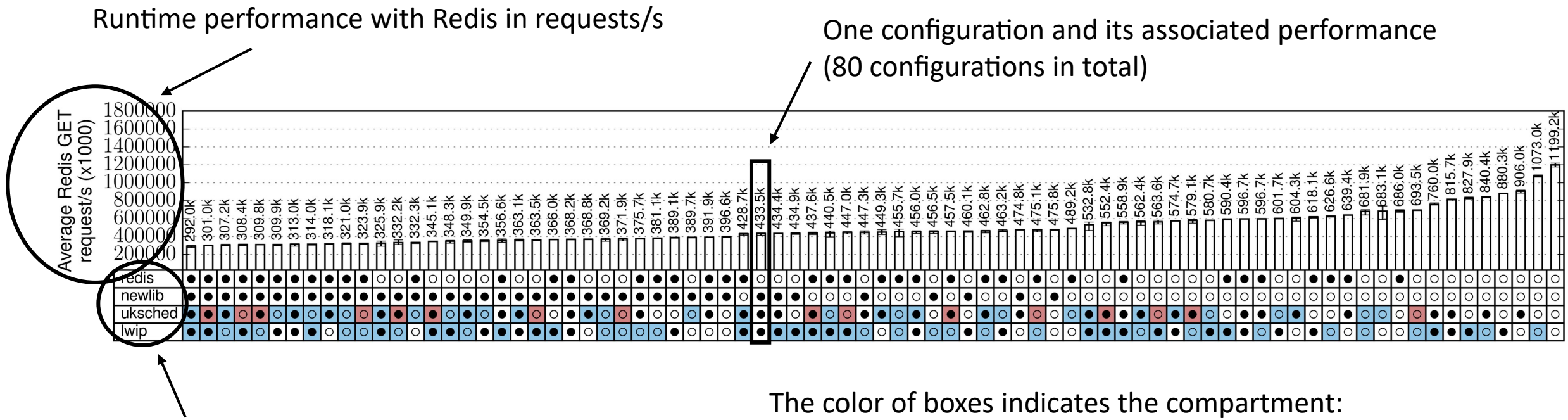
Runtime performance with Redis in requests/s



FlexOS libraries used in the Redis image
(only a subset for readability):

- Redis application
- C standard library (newlib)
- FlexOS scheduler (*uksched*)
- Network stack (lwip)

Flexibility



FlexOS libraries used in the Redis image (only a subset for readability):

- Redis application
- C standard library (newlib)
- FlexOS scheduler (*uksched*)
- Network stack (lwip)

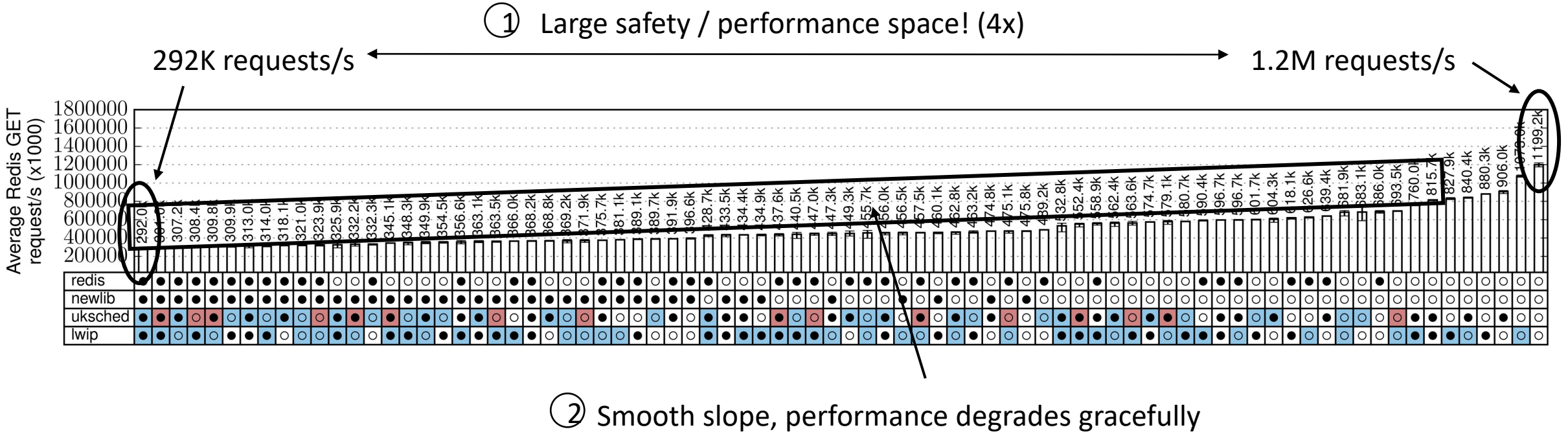
The color of boxes indicates the compartment:

Compartment 1 Compartment 2 Compartment 3

The dot whether hardening (ASan, Safestack, etc.) is enabled:

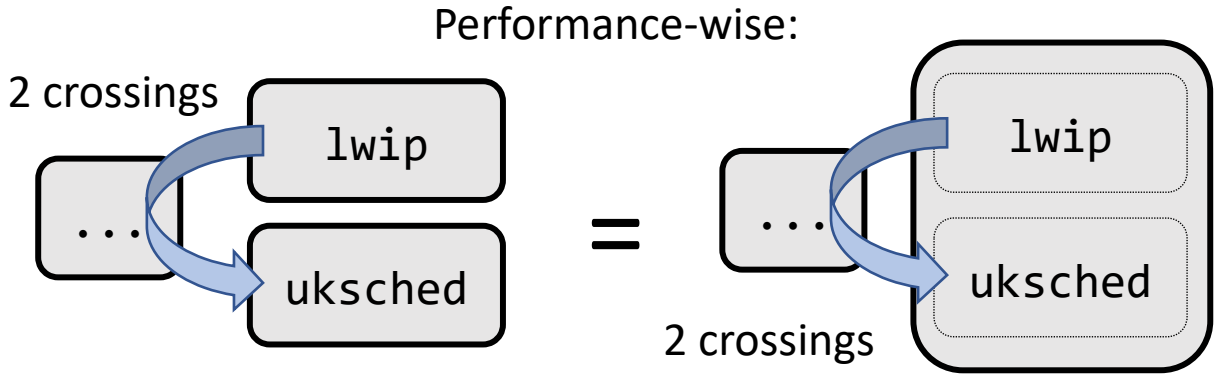
Hardening on Hardening off

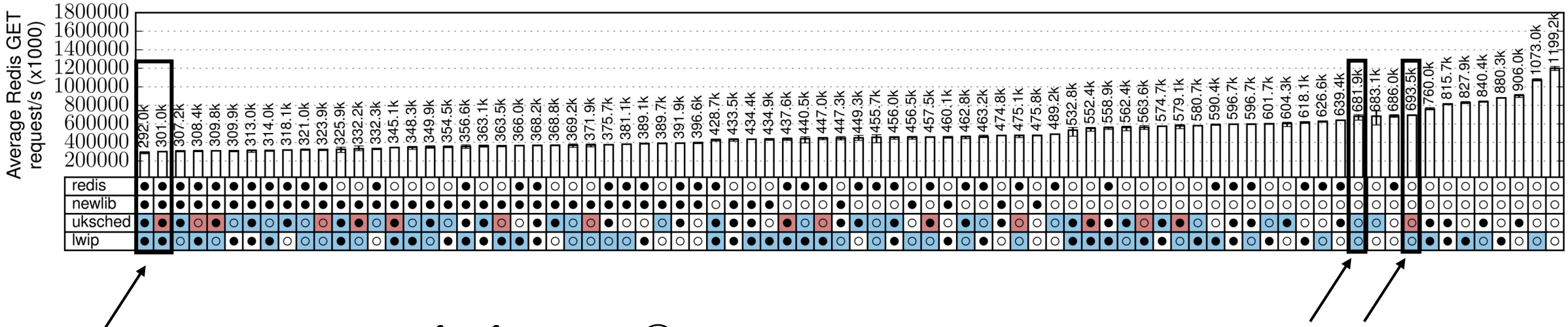
Flexibility



Hardening on | Hardening off | Compartment 1 | Compartment 2 | Compartment 3

Flexibility



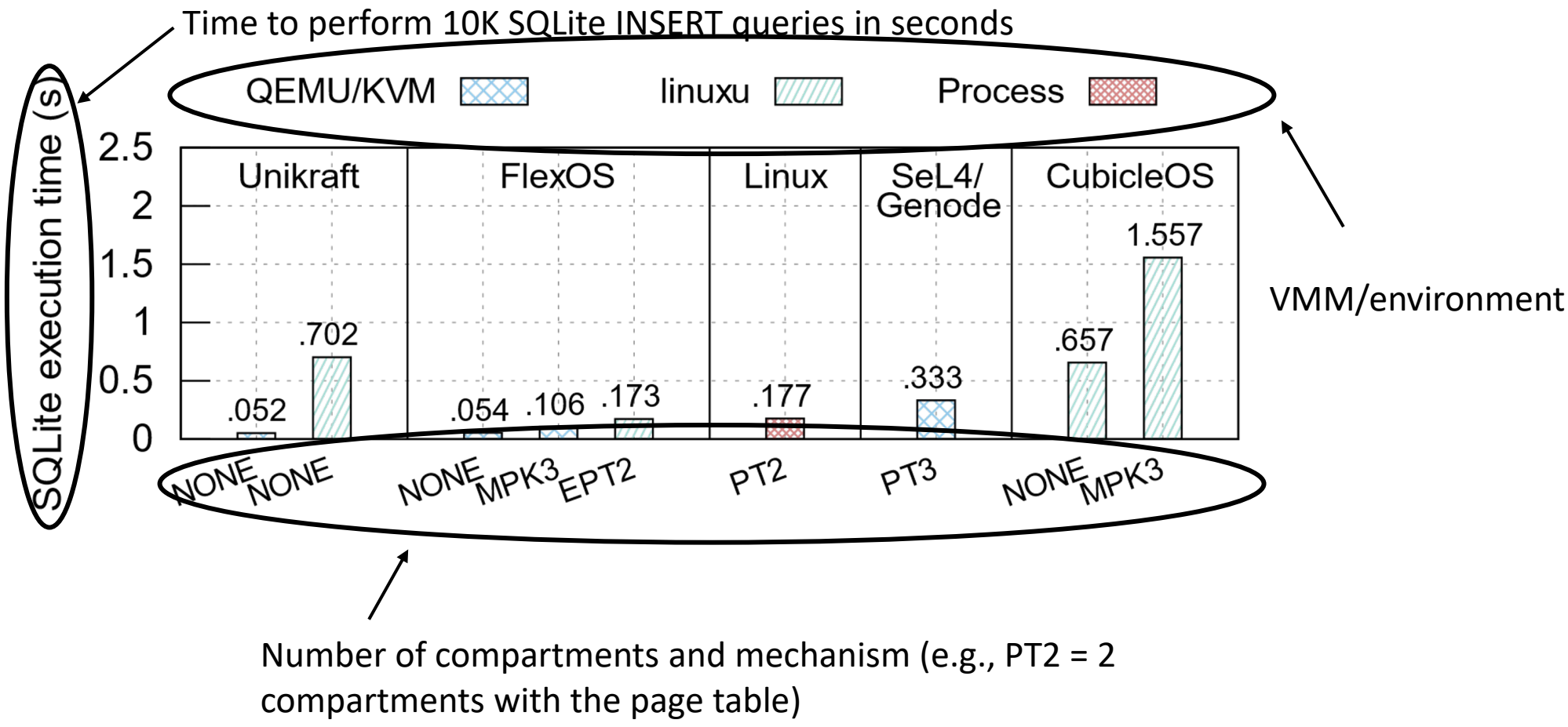


You can get some safety for free by exploring intelligently

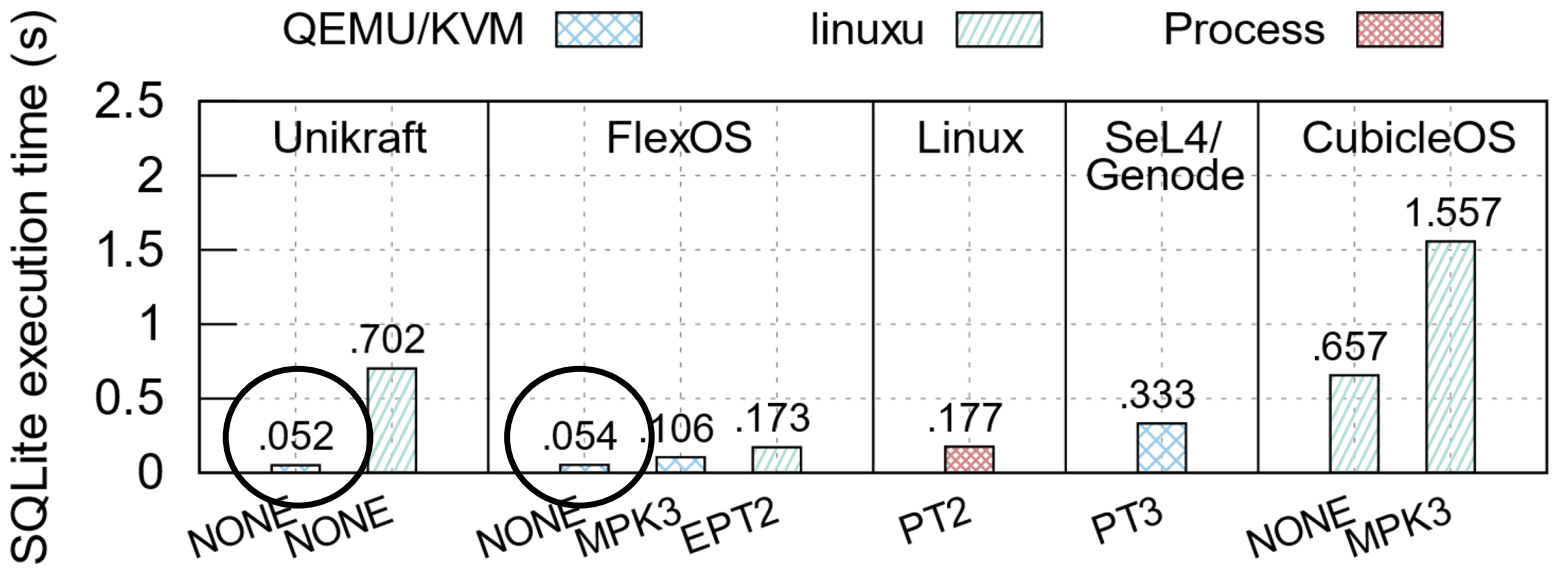
③ Similar performance, very different properties!
need to reason about communication patterns, fast paths

Hardening on | Hardening off | Compartment 1 | Compartment 2 | Compartment 3

Performance

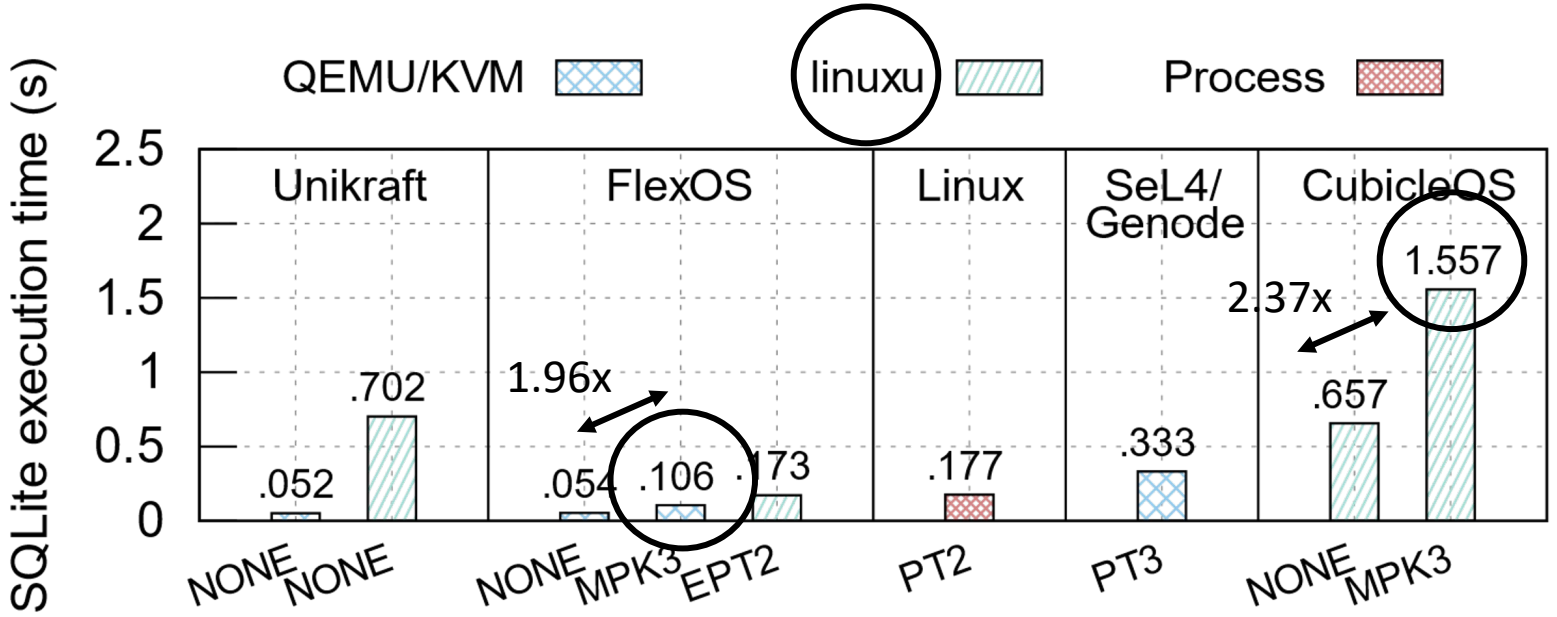


Performance



① No overhead when disabling isolation – you only pay for what you get

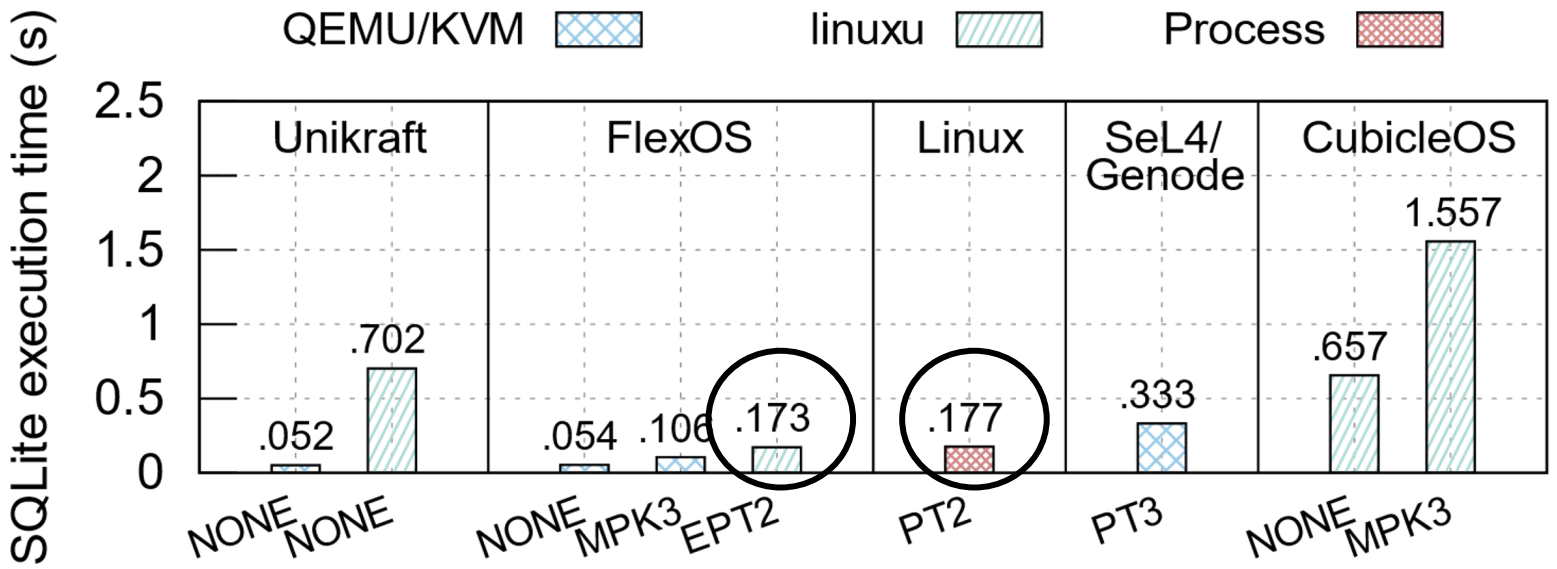
Performance



② The MPK backend compares very positively to competing solutions

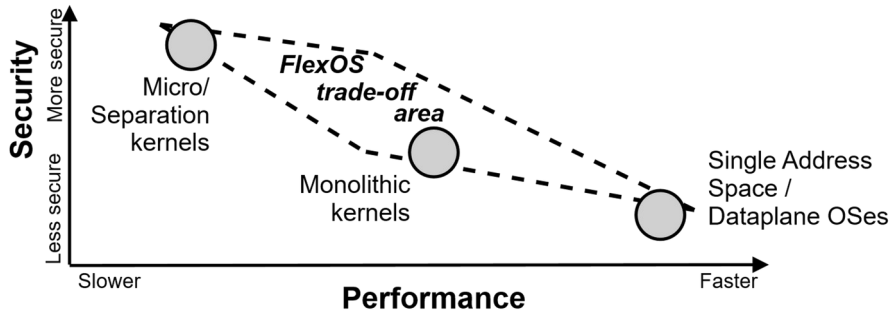
Tricky comparison with CubicleOS - they're using linuxu, a Linux userland debug platform of Unikraft

Performance



③ The EPT backend too compares positively to competing solutions

In a Nutshell



There is a **need for isolation flexibility**

- OS Specialization, hardware heterogeneity
- or quickly react to vulnerabilities!

Current approaches: **one isolation approach at design time**

Decouple isolation from the OS design:

- Make isolation decisions at **build time**
- Explore **performance v.s. security trade-offs**

Interested?

Webpage: https://project-flexos.github.io/
Pre-print of our ASPLOS'22 paper: https://arxiv.org/abs/2112.06566
By e-mail: `hugo.lefeuvre@manchester.ac.uk`

License: 3-Clause BSD License 🙂

Vlad-Andrei Bădoiu, Alexander Jung, Stefan Teodorescu, Sebastian Rauch, Felipe Huici, Costin Raiciu, Pierre Olivier