FlexOS: Towards Flexible OS Isolation

Hugo Lefeuvre[1], Vlad-Andrei Bădoiu[2], Alexander Jung[3,4], Stefan Teodorescu[2], Sebastian Rauch[5], Felipe Huici[6,4], Costin Raiciu[2,7], Pierre Olivier[1]

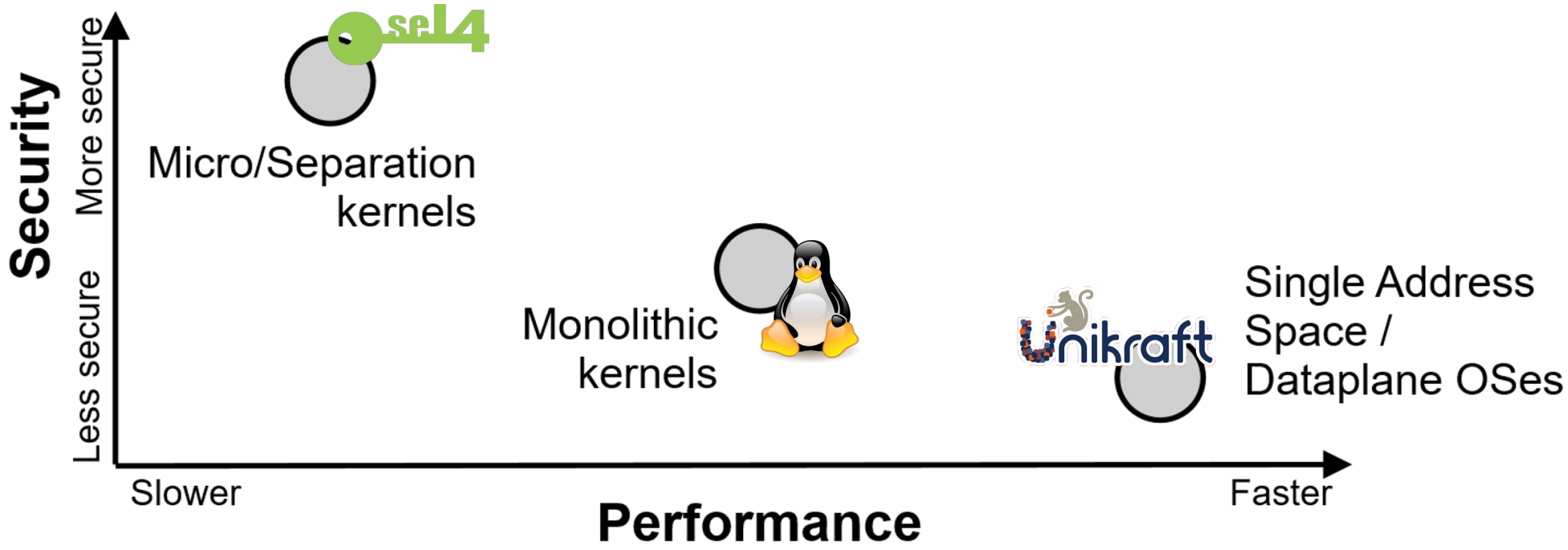
[1]The University of Manchester, [2]Politehnica Bucharest, [3]Lancaster University, [4]Unikraft.io, [5]Karlsruhe Institute of Technology, [6]NEC Labs Europe, [7]Correct Networks

ASPLOS'22, 28th February - 4th March 2022

Current OS Designs

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

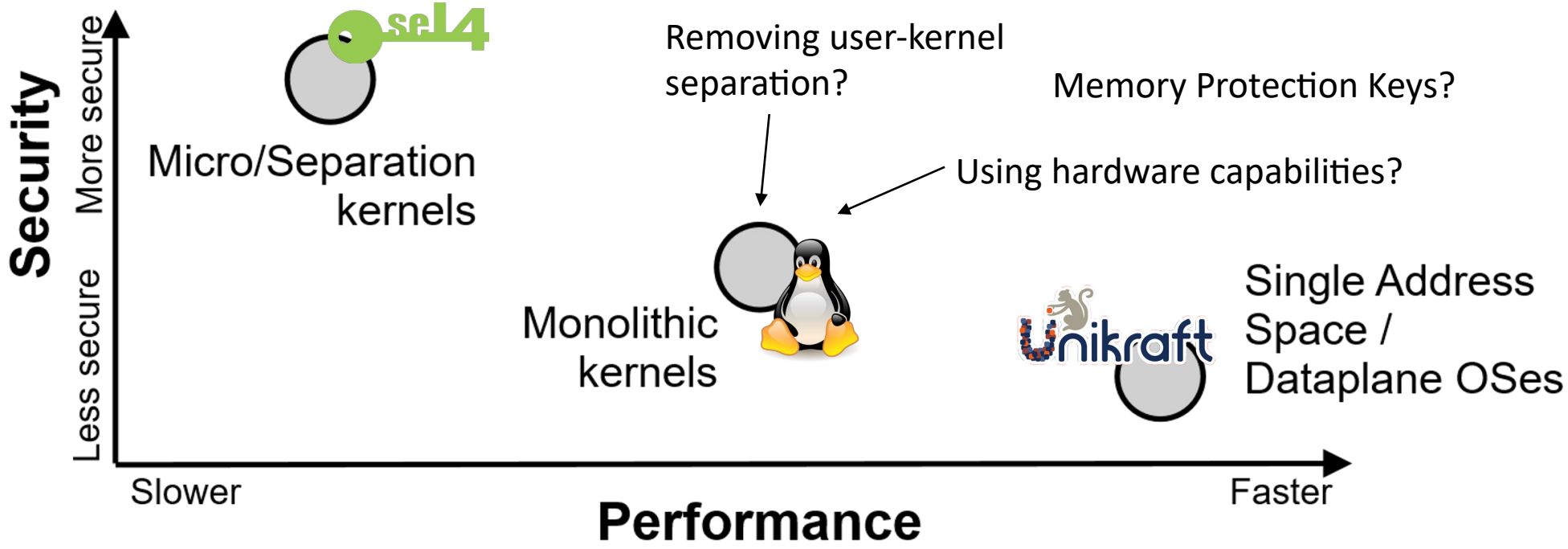
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



Current OS Designs

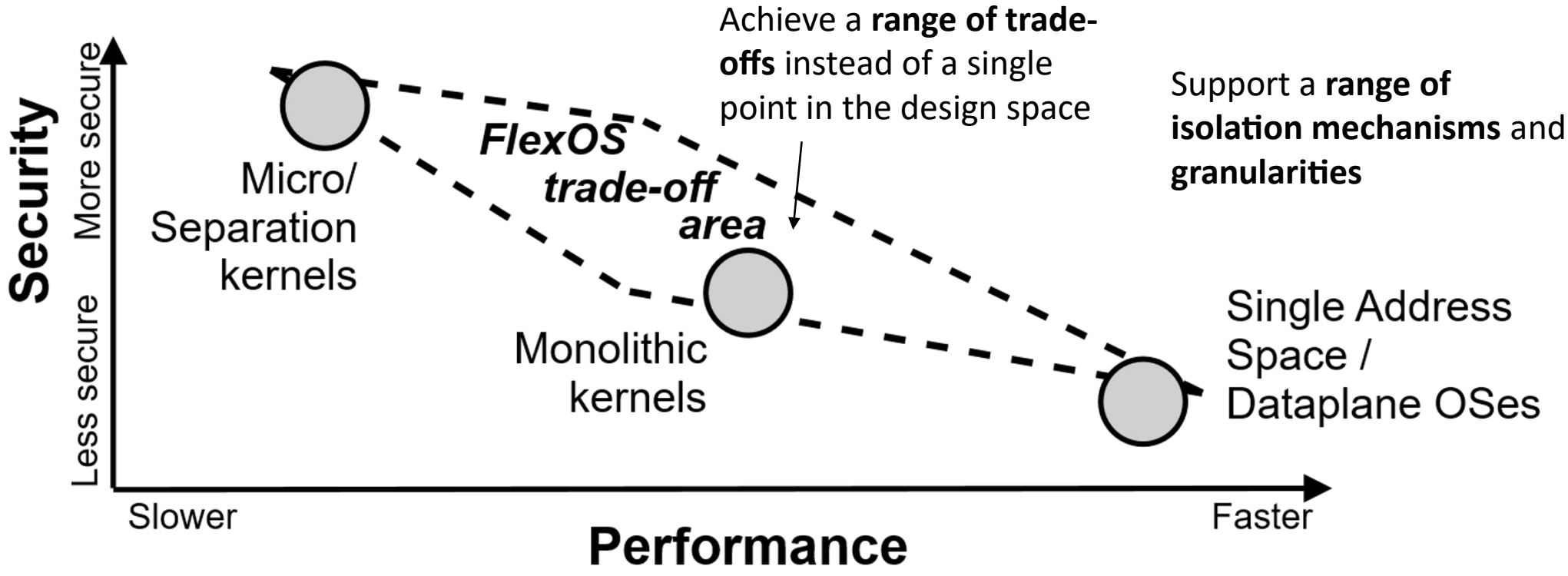
OS security/isolation strategies are **fixed** at design time!

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

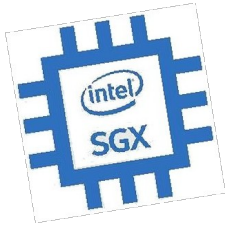


FlexOS: Flexible Isolation

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

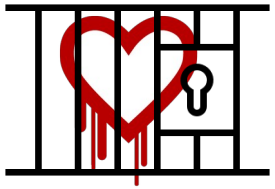


Other Use-Cases for Flexible Isolation



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

Core Libraries

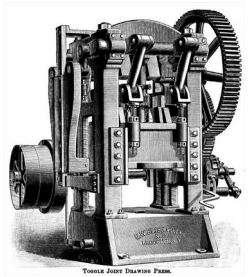
boot | sched | mm

Kernel & User Libs

ramfs | vfs | ...

Isolation Backends

MPK | EPT | ...



Happy users

config.yaml

① Input config

FlexOS Toolchain

② Select isolation mechanism ("Backend")

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

④ Generate image with appropriate isolation properties

comp1: boot | sched | mm | ...

comp2: libopenjpeg

comp3: libssl

VMs

Possible Image 1

comp1: boot | sched | mm | libssl

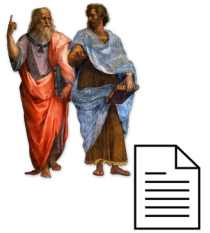
comp2: ... | netdev

MPK

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

boot | sched | mm

Kernel & User Libs

nginx | ssl | jpeg | ramfs | vfs | ...

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

Reuse libraries as finest granularity of compartmentalization

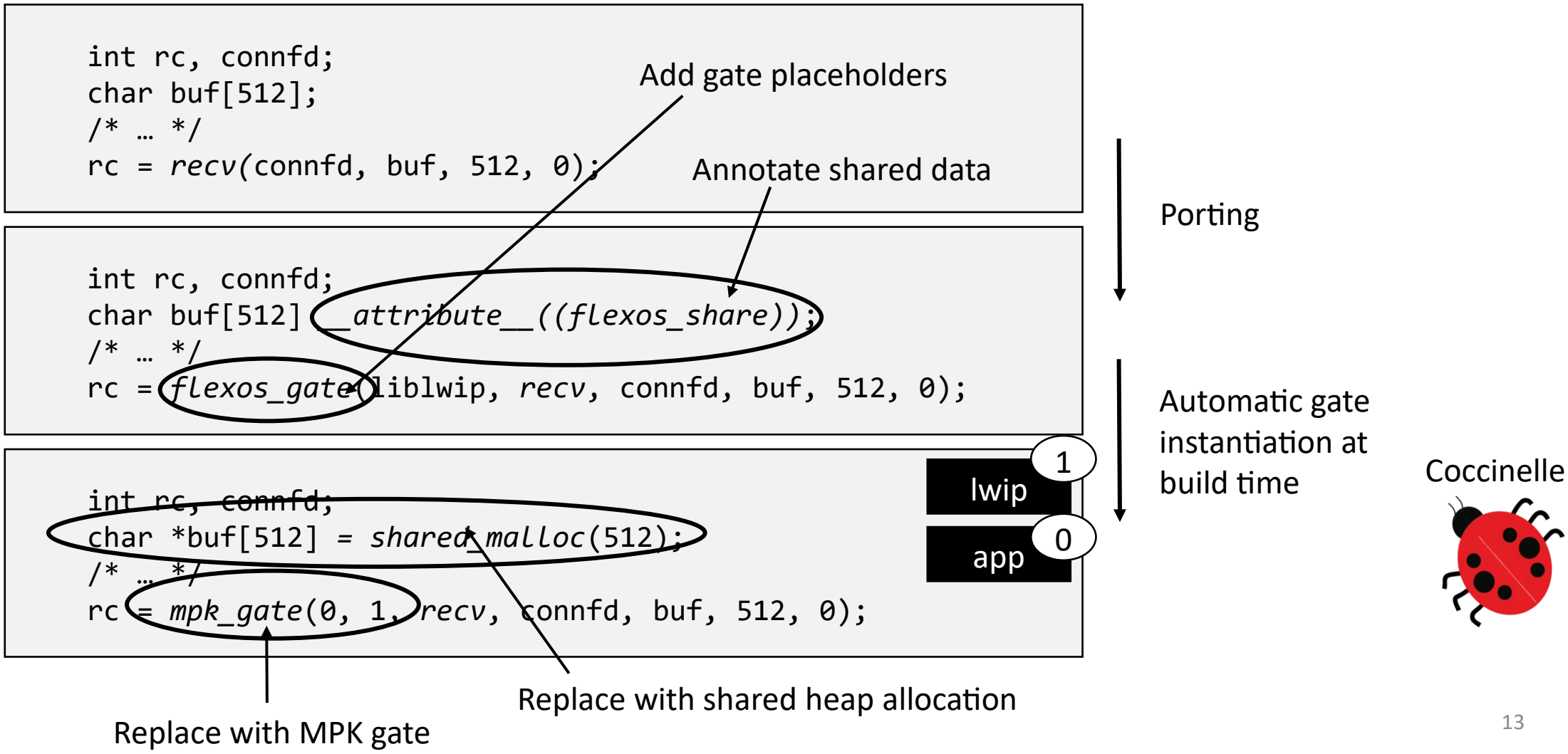
"Pre-compartmentalize" them

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

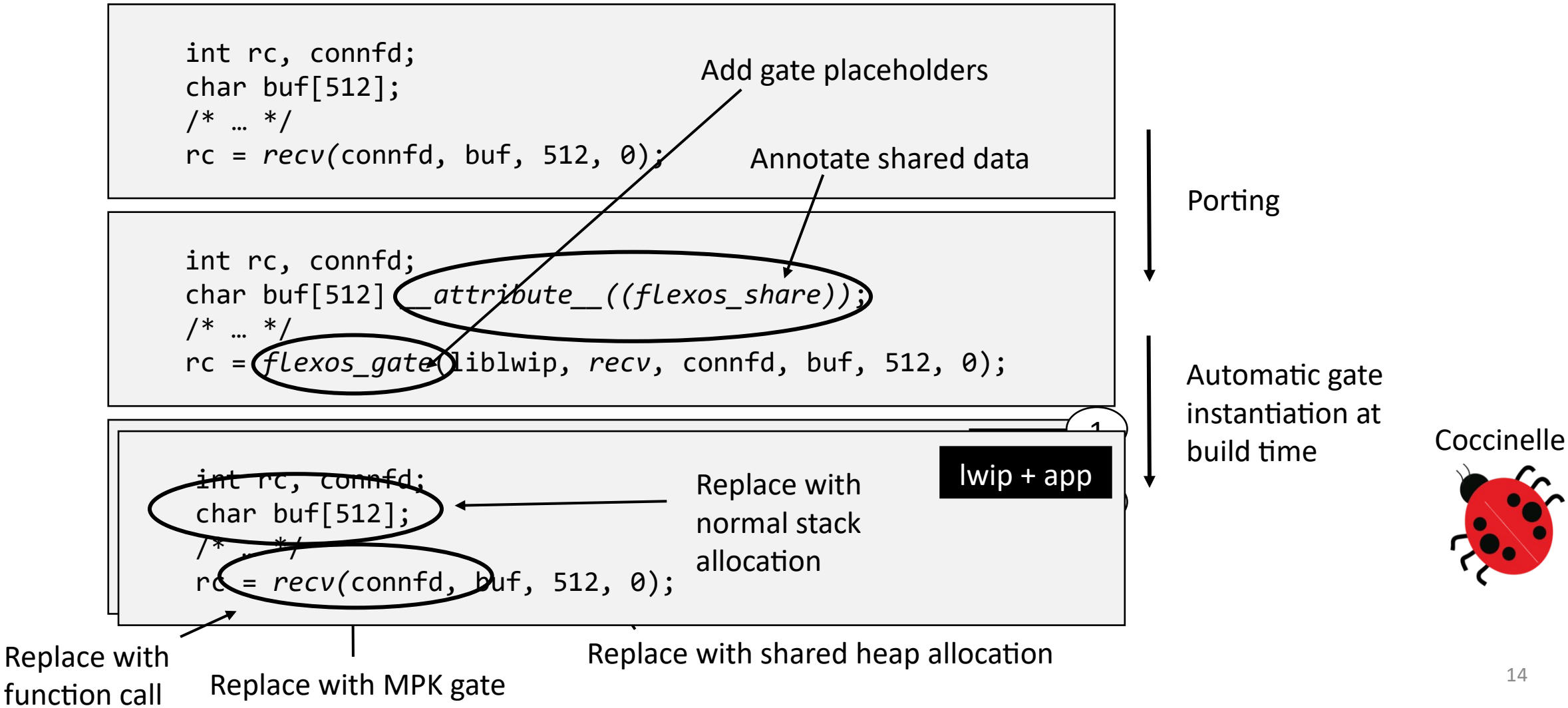
Define them as part of the **FlexOS API**

MPK | VMs | TEEs | ...

FlexOS 101: Compartmentalization API



FlexOS 101: Compartmentalization API



Prototype

Implementation **on top of Unikraft**

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

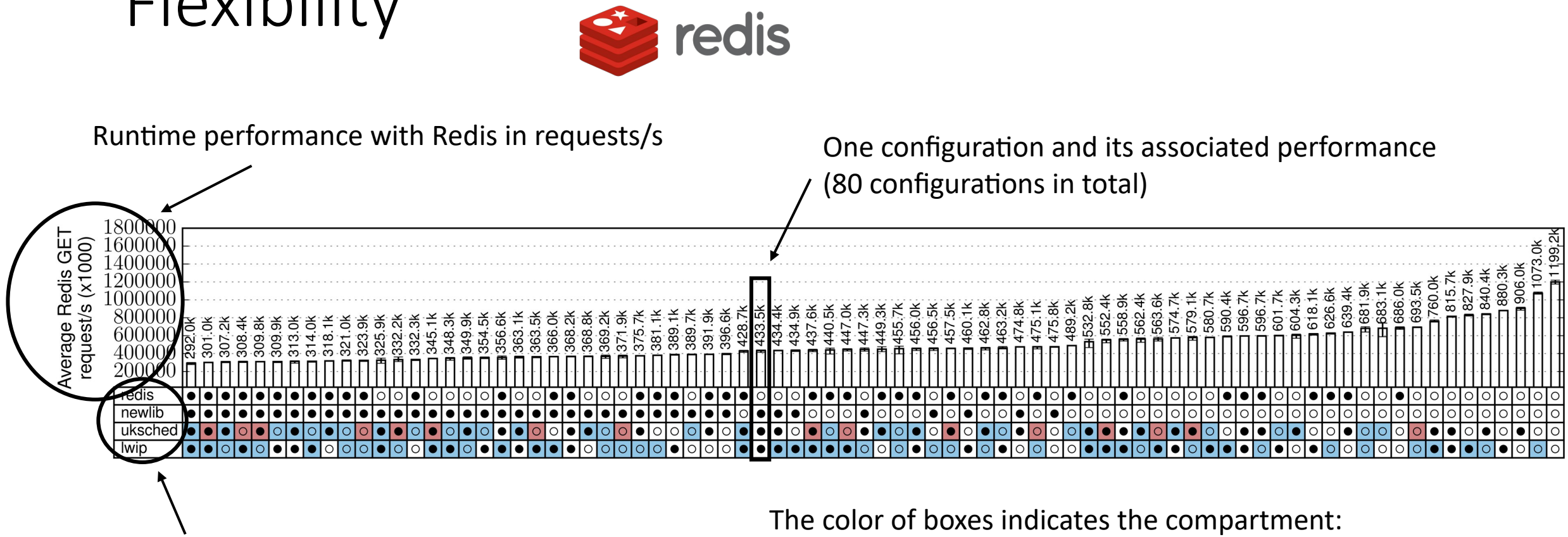
Port of libraries: network stack, scheduler, filesystem, time subsystem

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

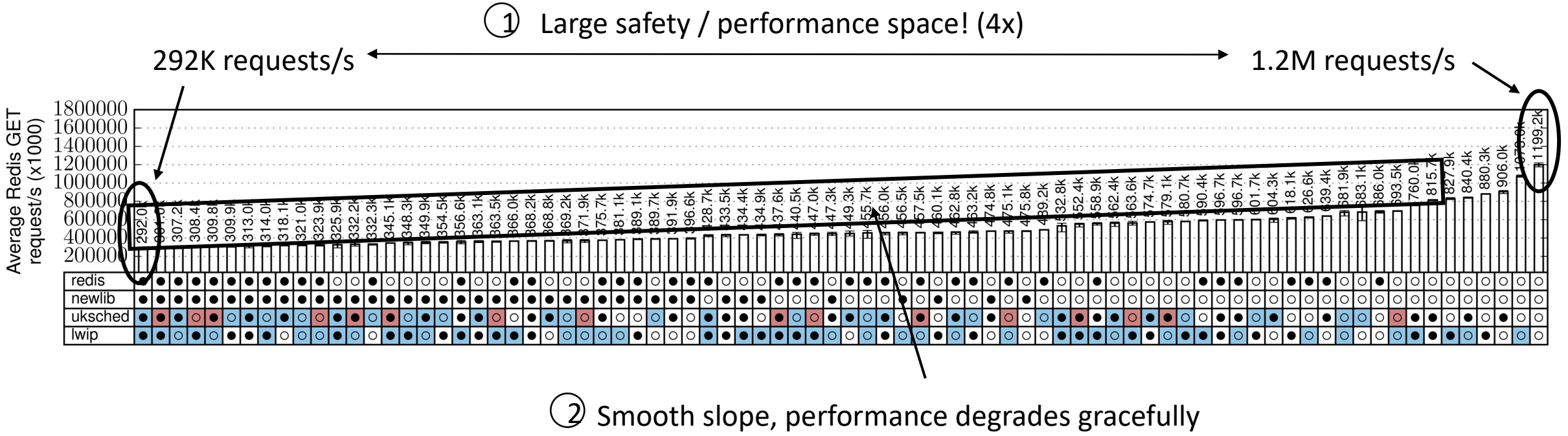
This talk: focus on demonstrating **flexibility and performance**

more results in our paper 🙂

Flexibility

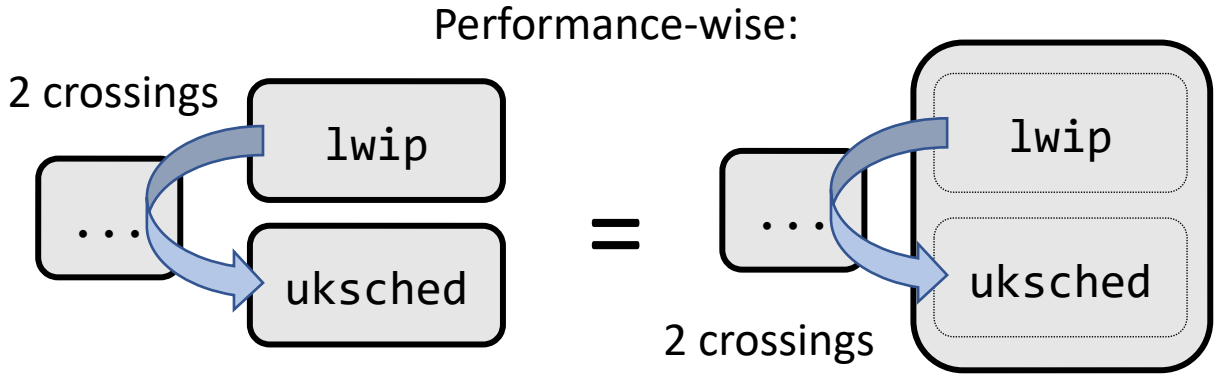


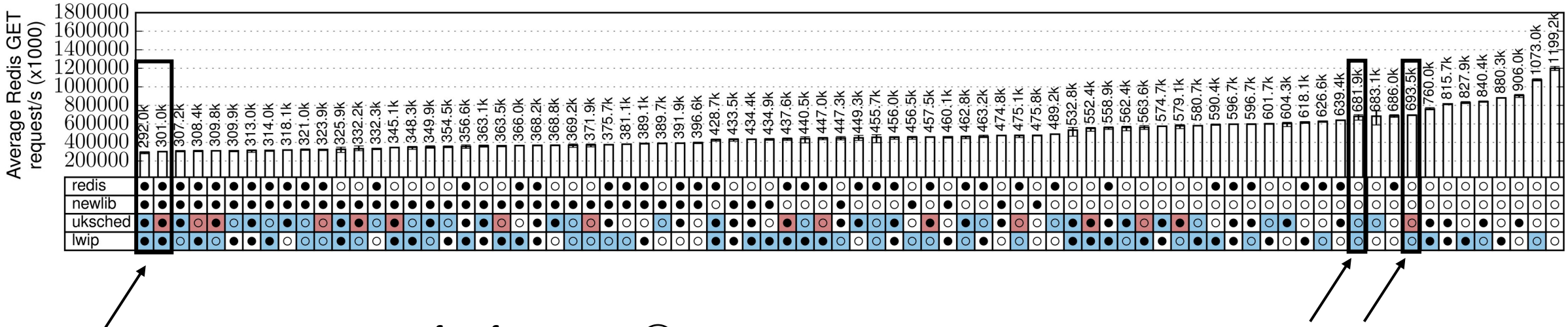
Flexibility



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

Flexibility



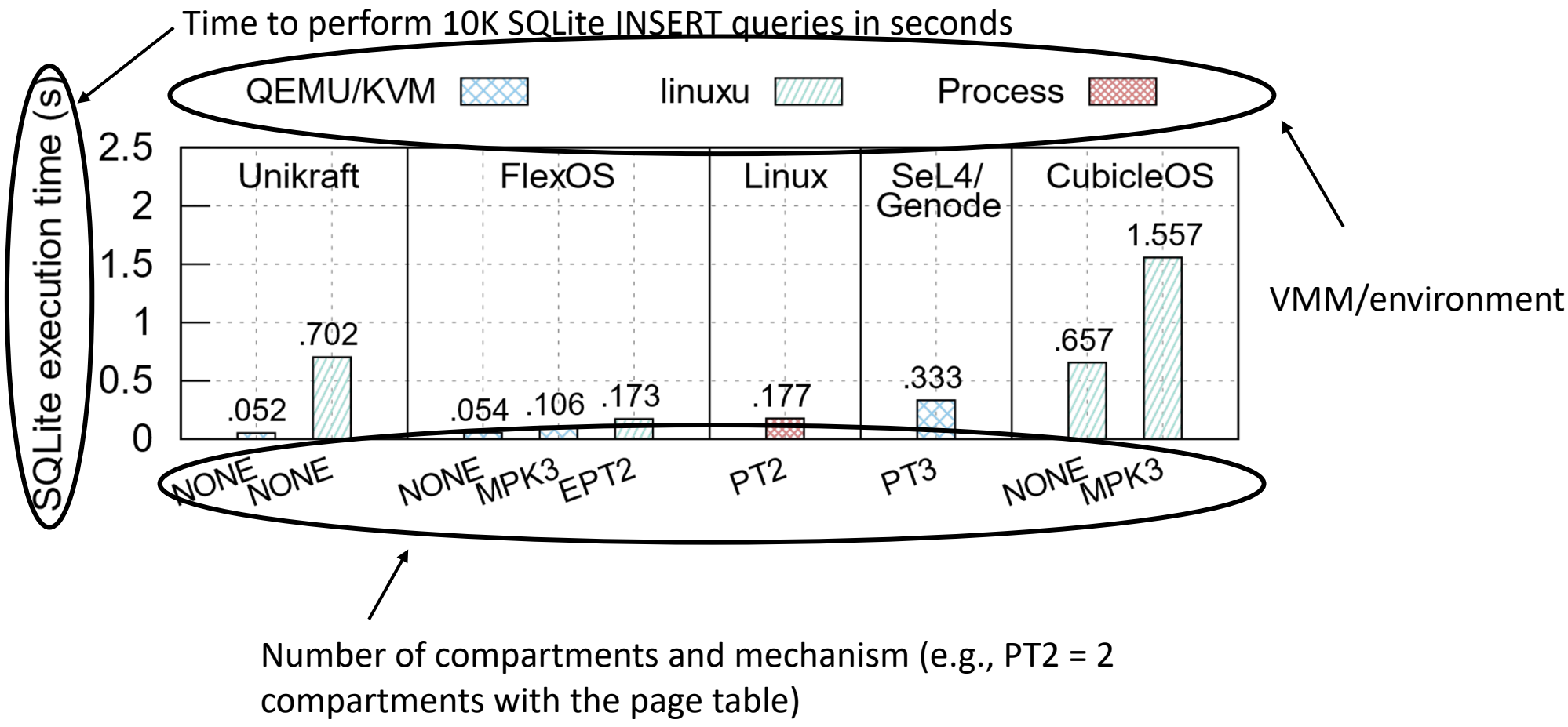


You can get some safety for free by exploring intelligently

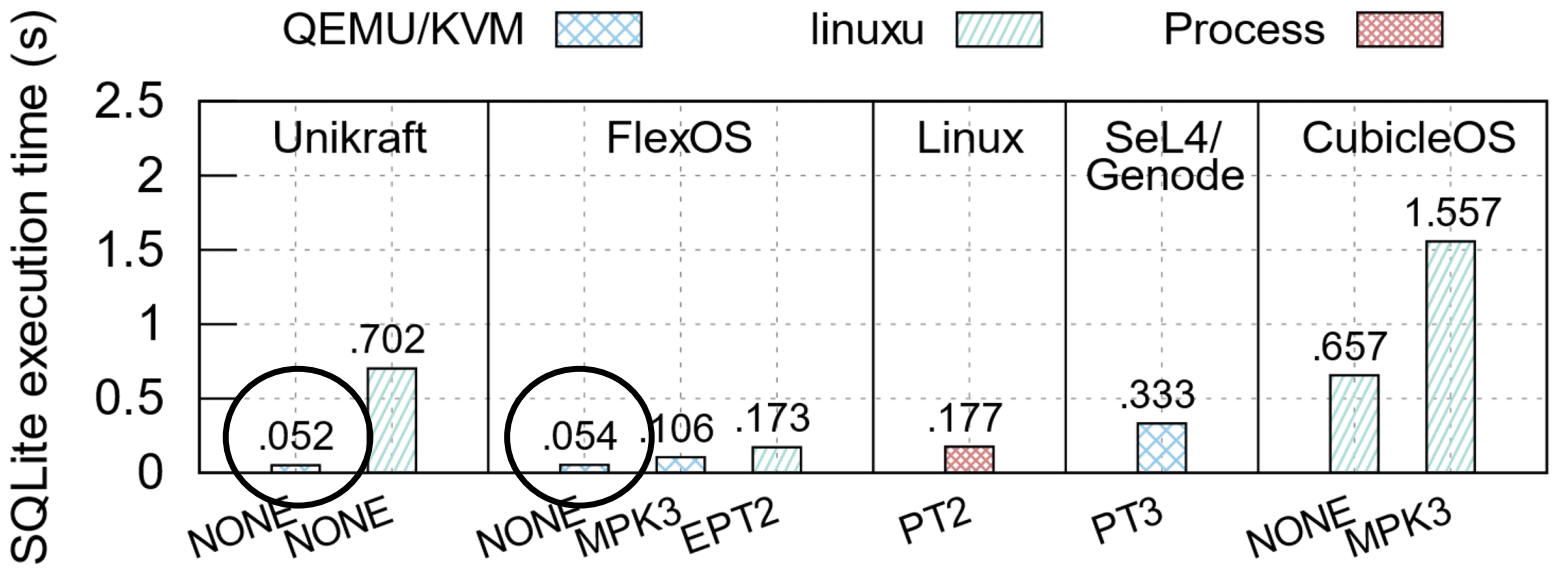
③ Similar performance, very different properties!
need to reason about communication patterns, fast paths

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

Performance

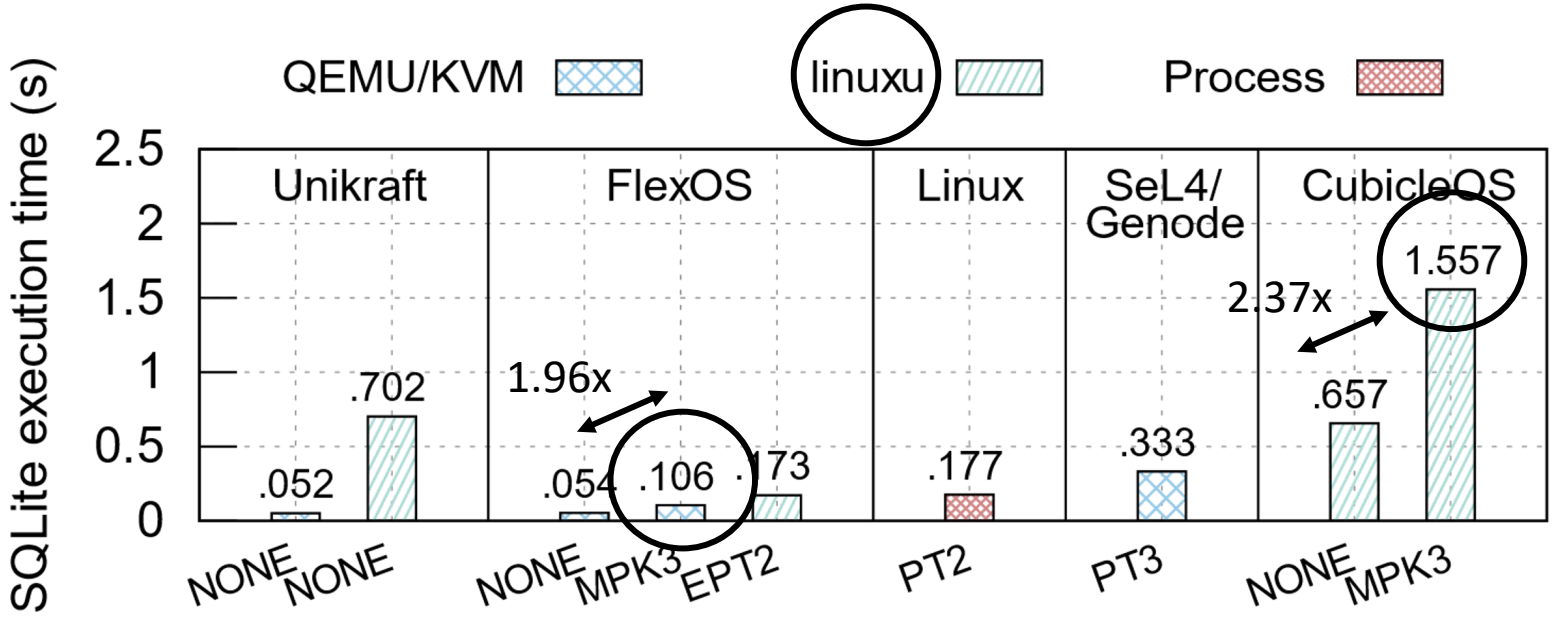


Performance



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

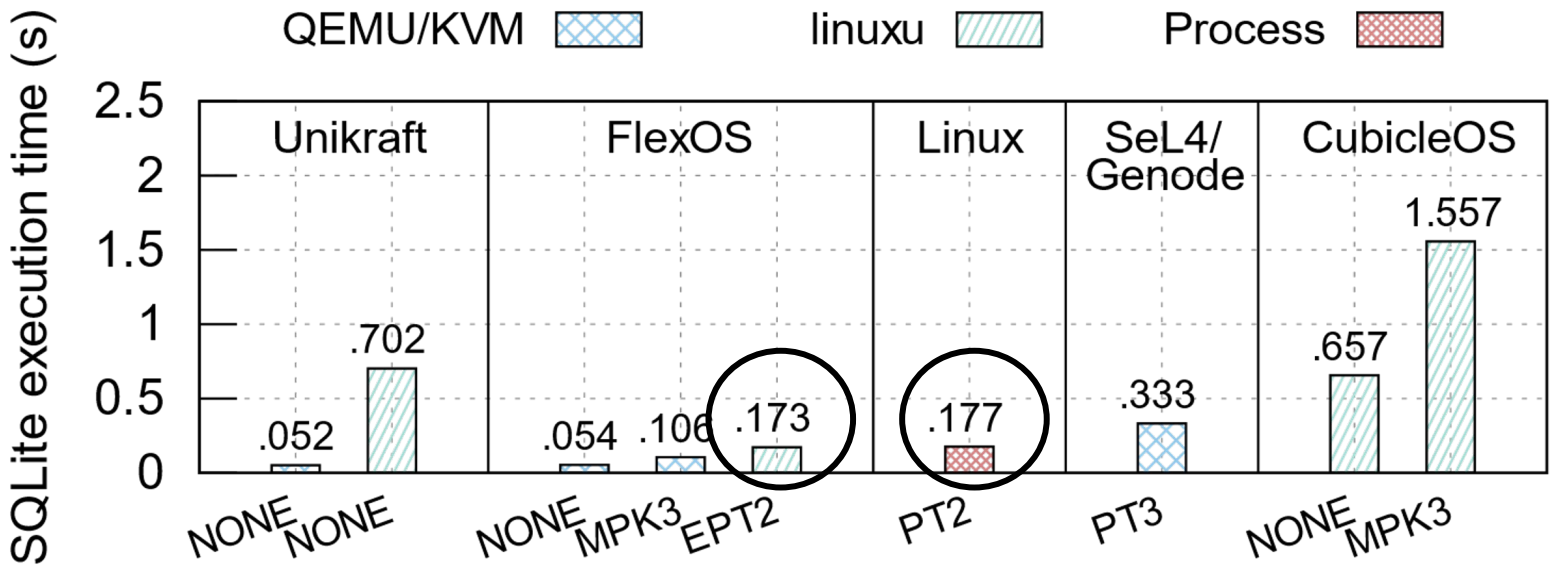
Performance



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



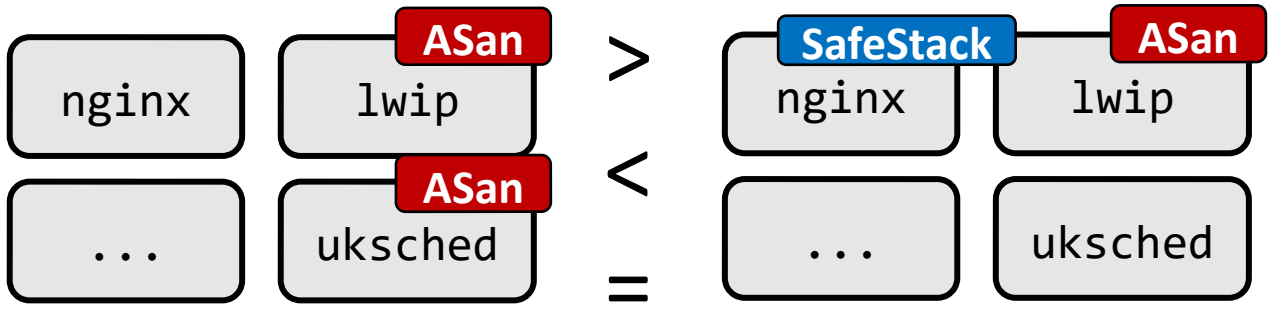
③ The EPT backend too compares positively to competing solutions

Exploring the Design Space

Now, we've a nice framework!

We can leverage FlexOS to get the most secure image for a given performance budget!

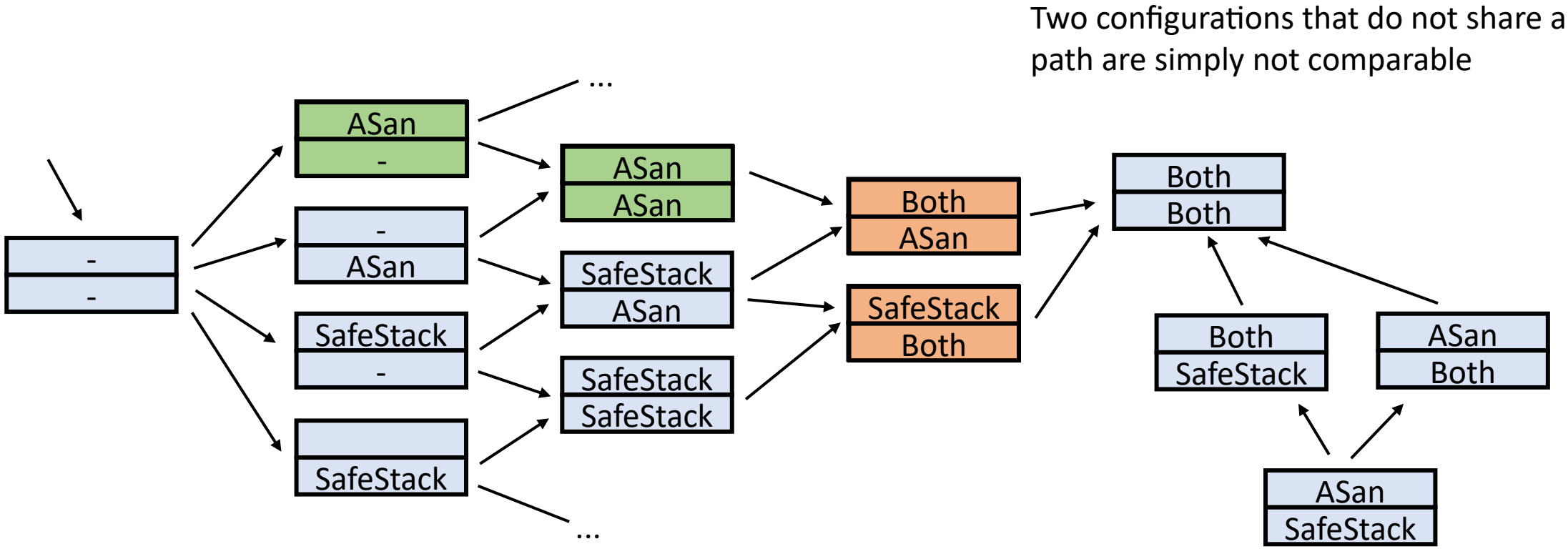
Problem: some configurations are not comparable



How can we reason about security/performance trade-offs?

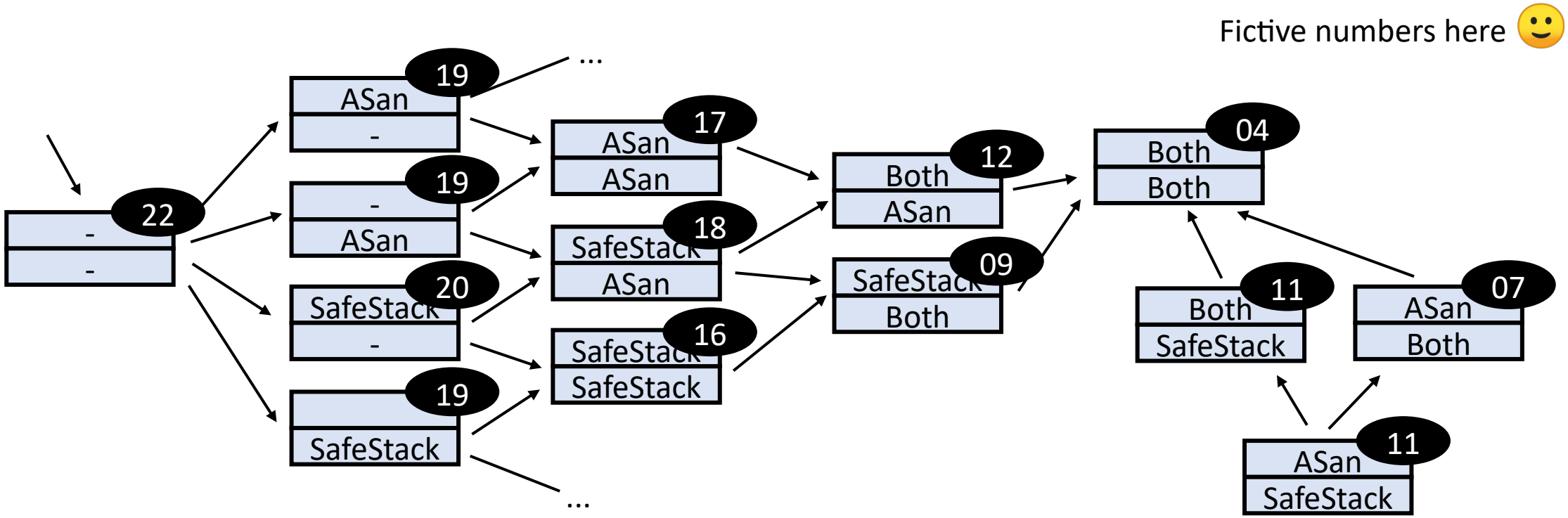
Exploring the Design Space

What we propose: consider configurations as a partially ordered set (poset)



Exploring the Design Space

We can then label each node with performance characteristics (in practice no need to label everything)



Exploring the Design Space

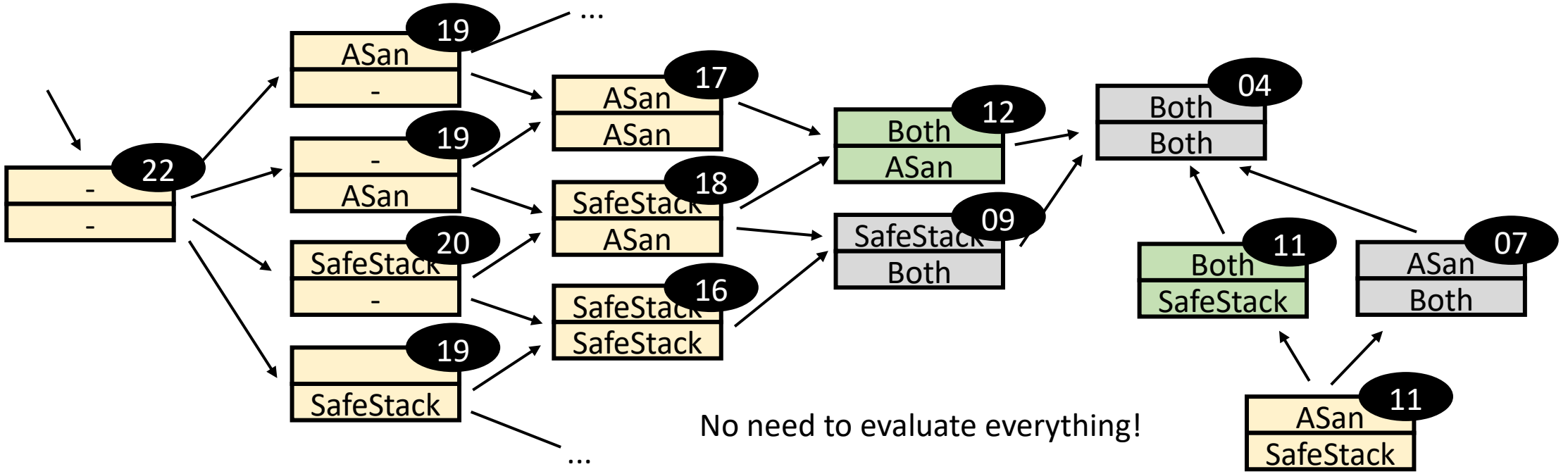
Let the user do the final choice

Based on this ordering and labeling we can choose the last node of each path that satisfies the performance constraints

Both
ASan

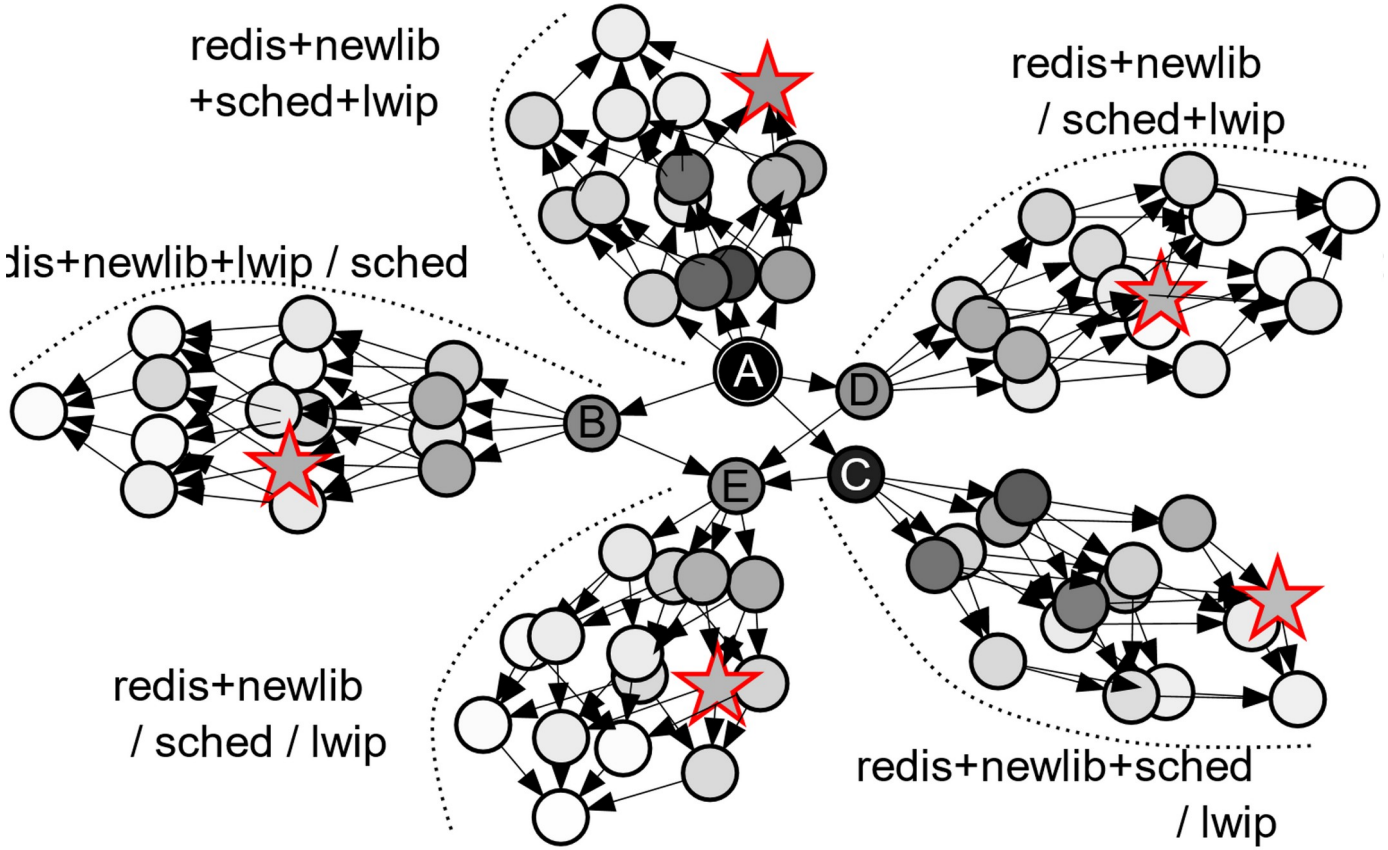
Both
SafeStack

Curated list of optimal configurations



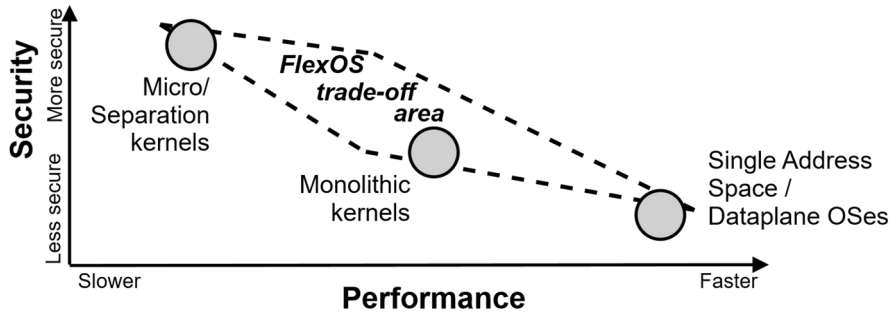
No need to evaluate everything!

Applying POSets on Redis



Reduction of 80 configurations to 5 candidates

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?

Get in touch!

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 🙂