Presented at
ASPLOS'22

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]

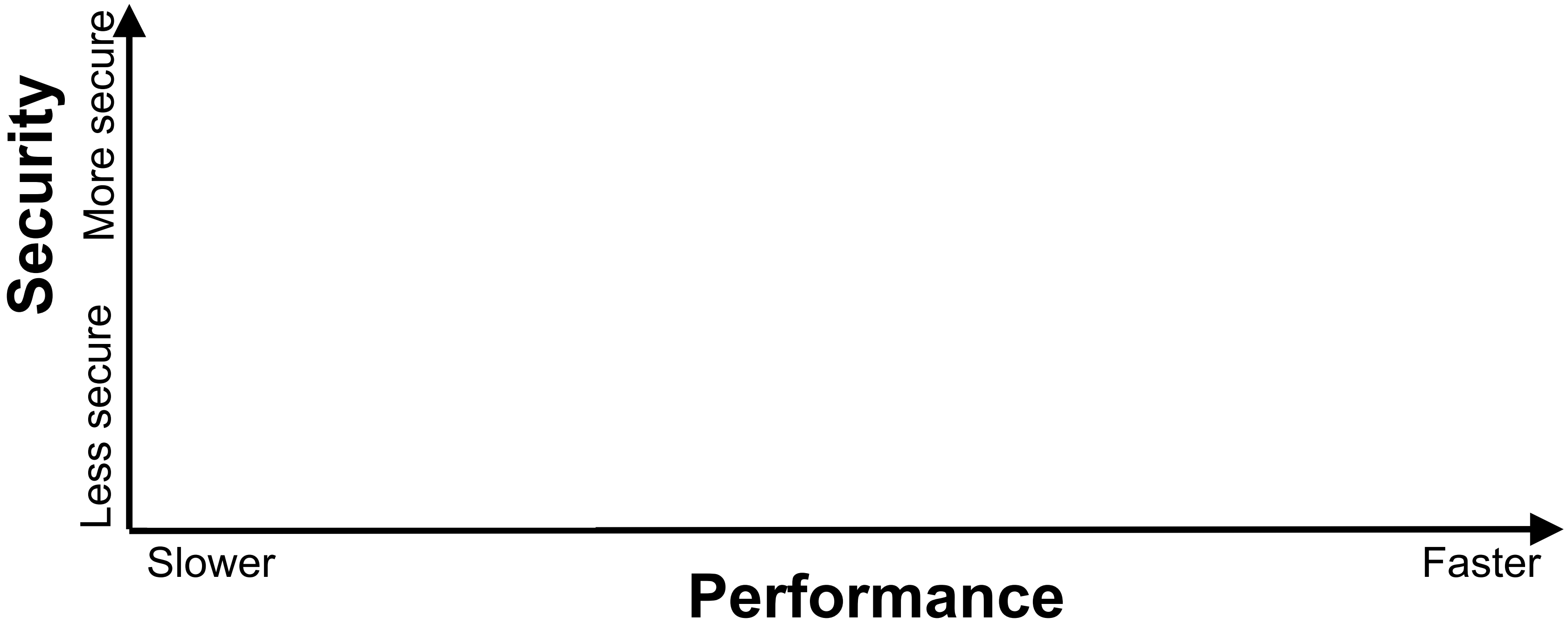
[1] The University of Manchester, [2] Politehnica Bucharest, [3] Lancaster University, [4] Unikraft.io,
[5] Karlsruhe Institute of Technology, [6] NEC Labs Europe, [7] Correct Networks

Future Device Technology Summit, 10th October 2023

Current OS Designs

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

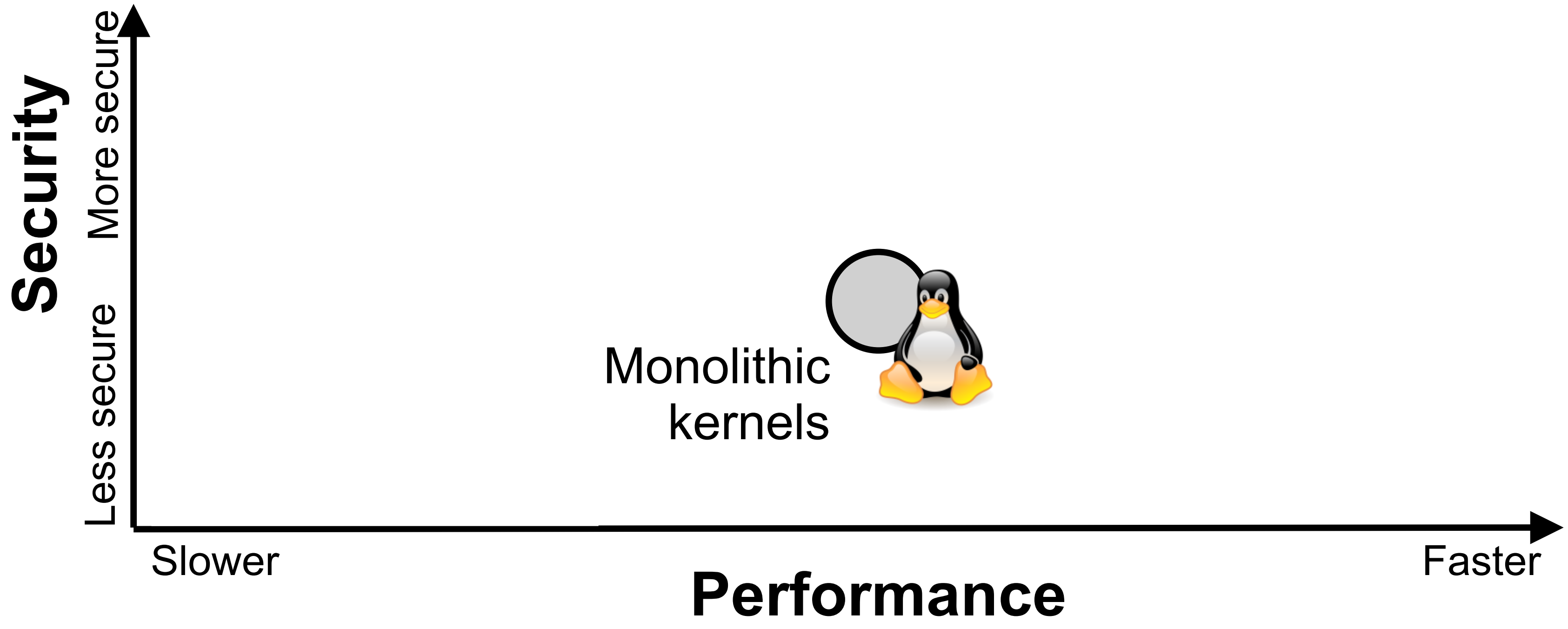
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



Current OS Designs

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

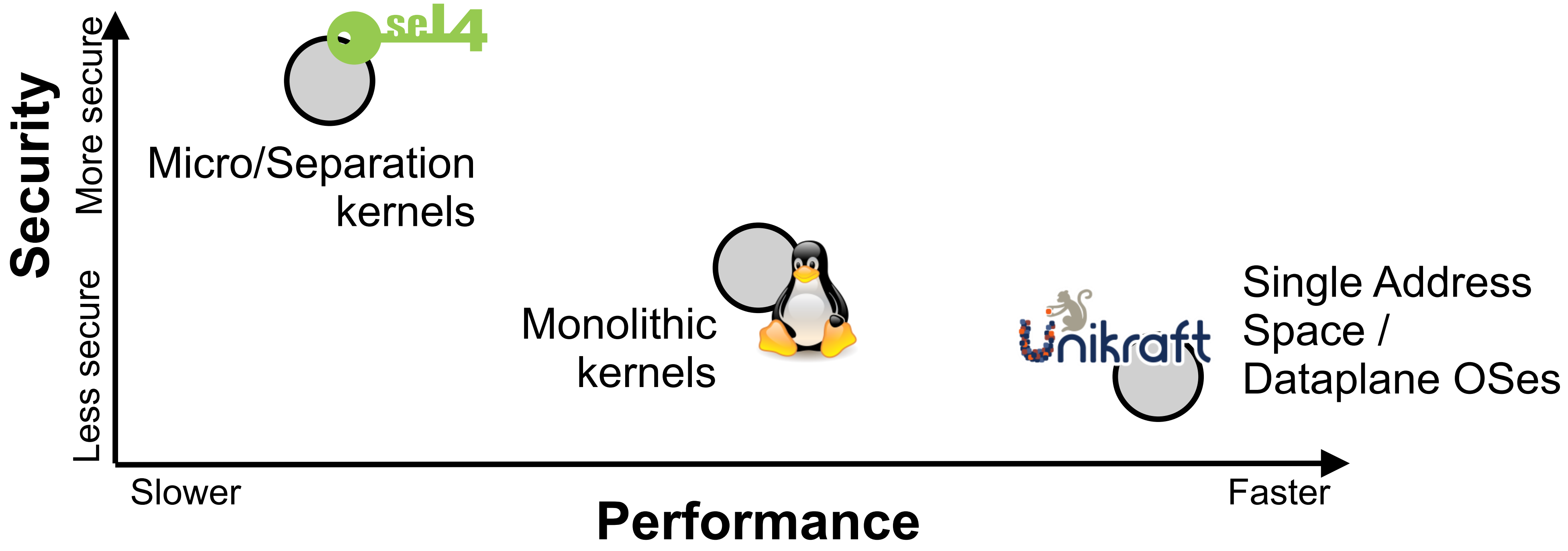
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



Current OS Designs

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

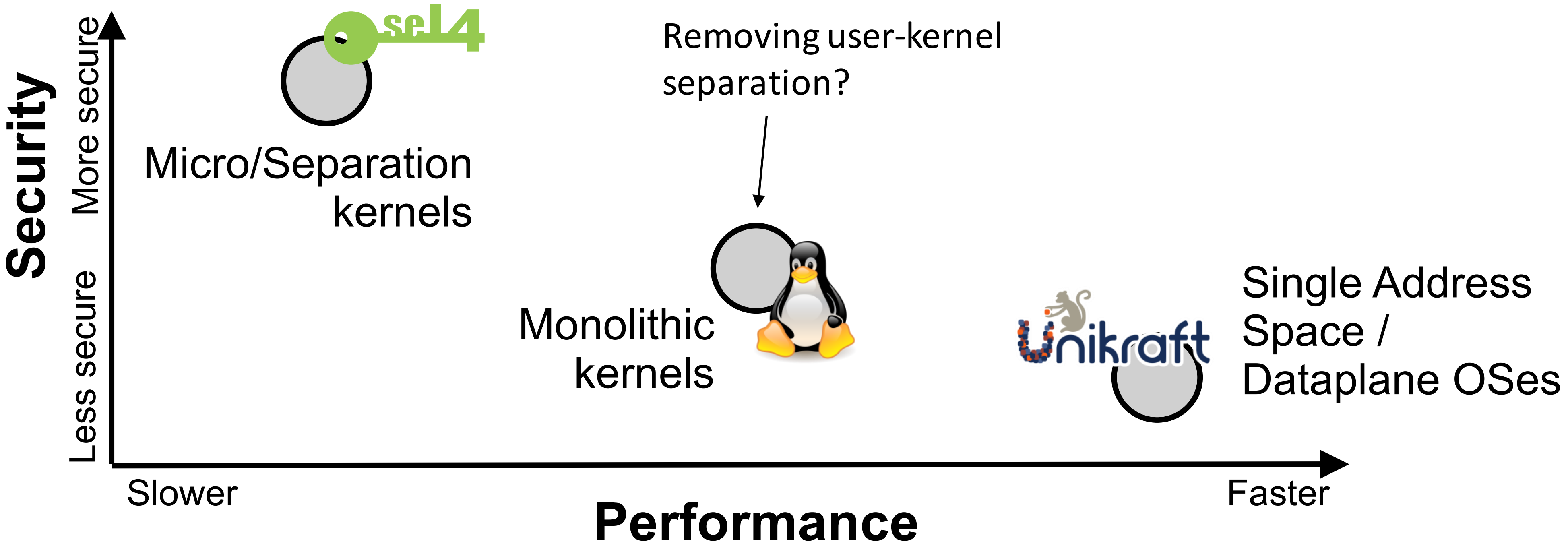
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



Current OS Designs

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

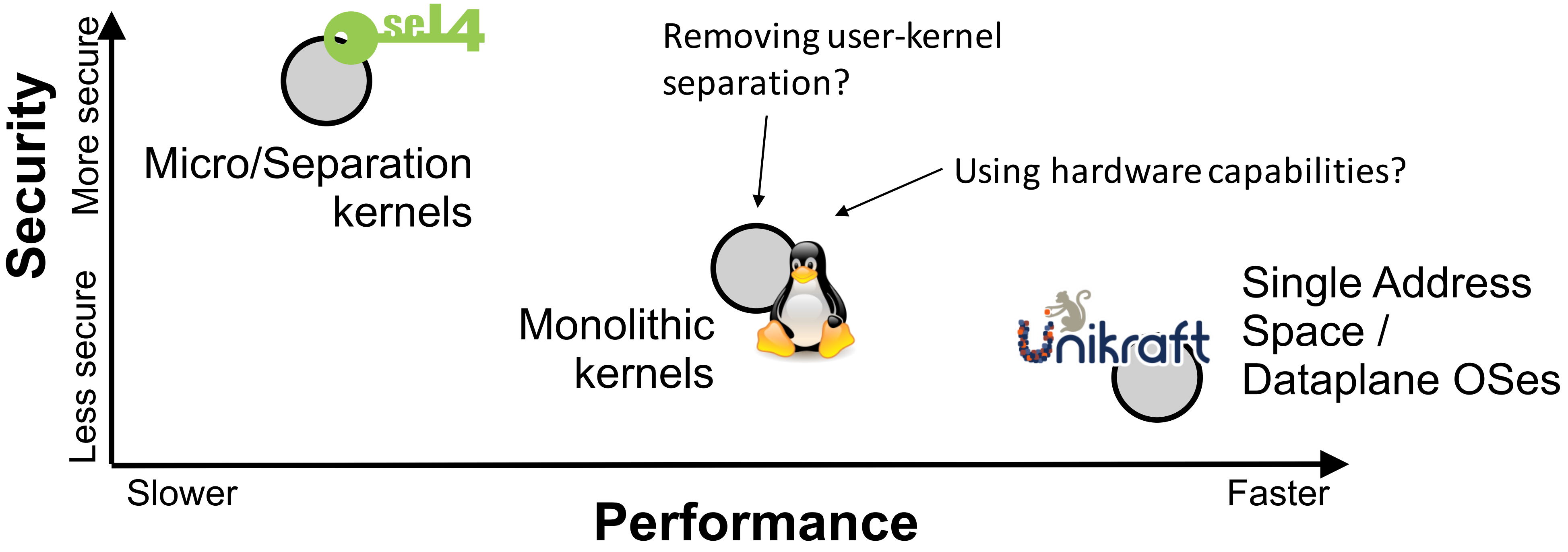
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



Current OS Designs

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

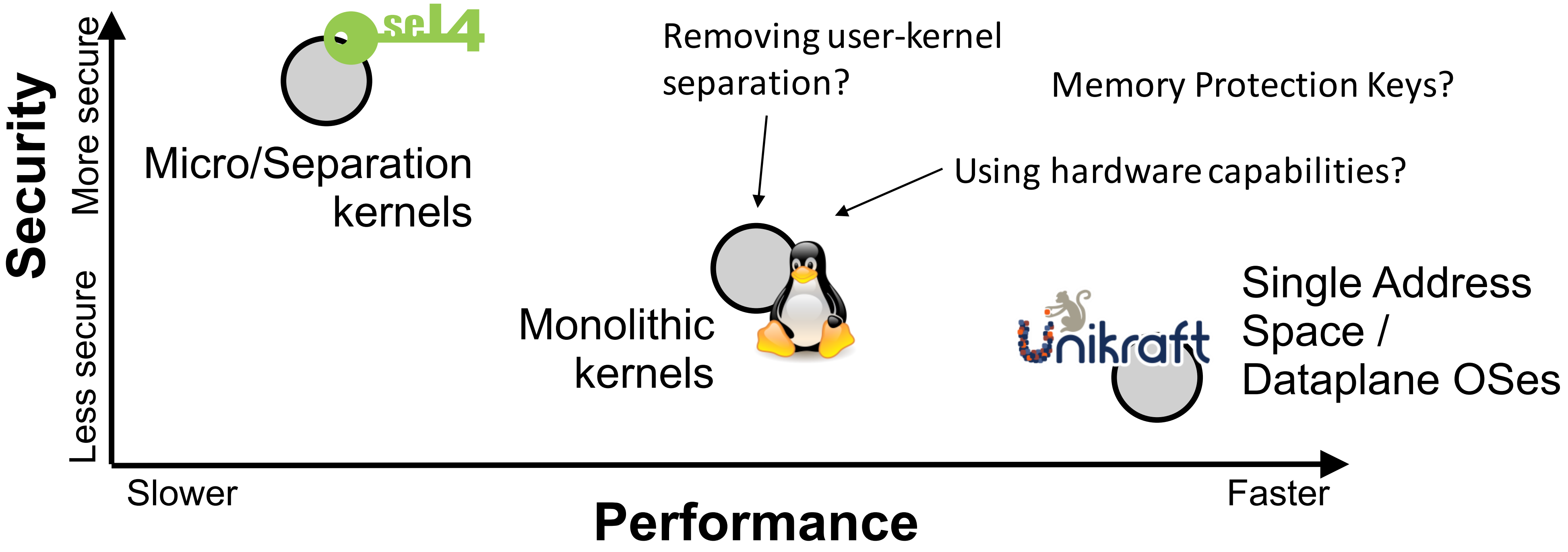
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



Current OS Designs

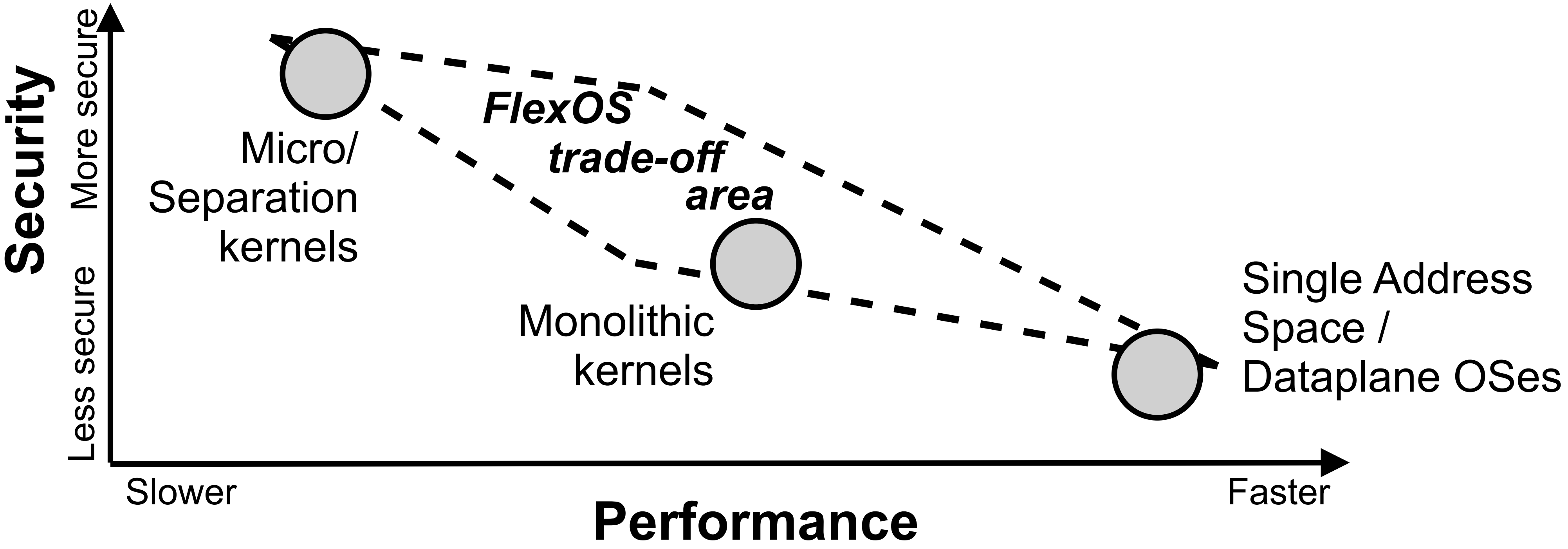
OS security/isolation strategies are **fixed** at design time!

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



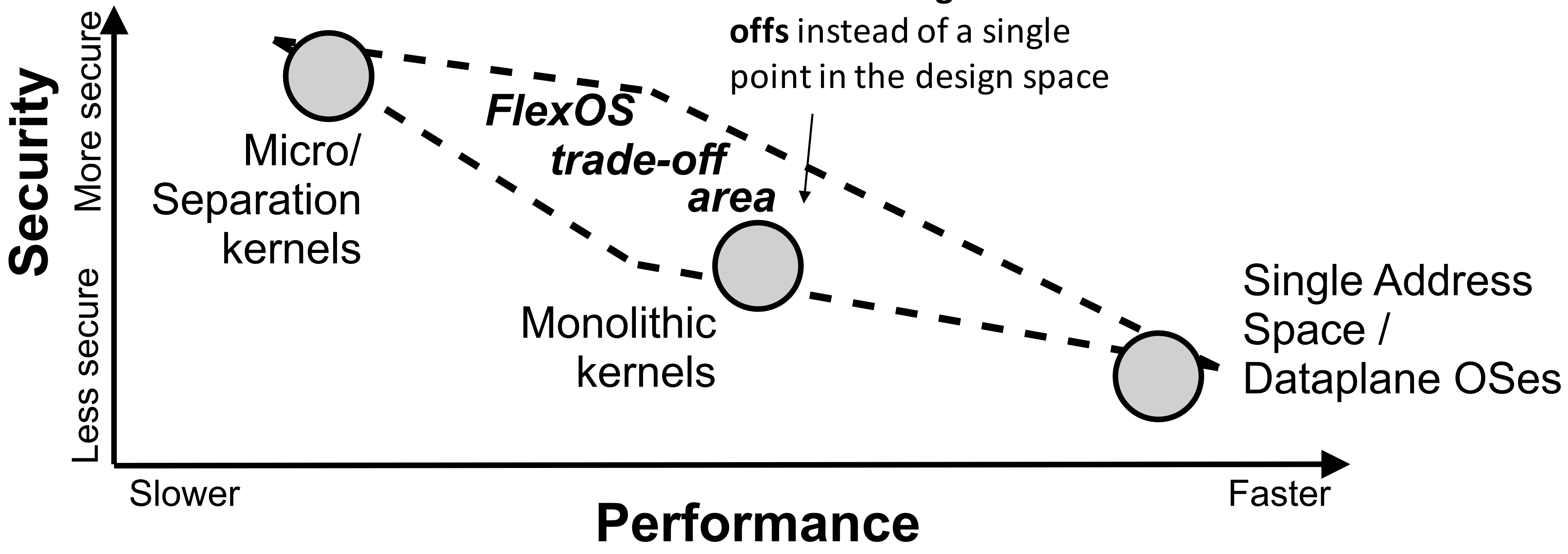
FlexOS: Flexible Isolation

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


Security
More secure
Less secure
Micro/ Separation kernels
FlexOS trade-off area
Monolithic kernels
Single Address Space / Dataplane OSes
Slower
Faster
Performance

FlexOS: Flexible Isolation

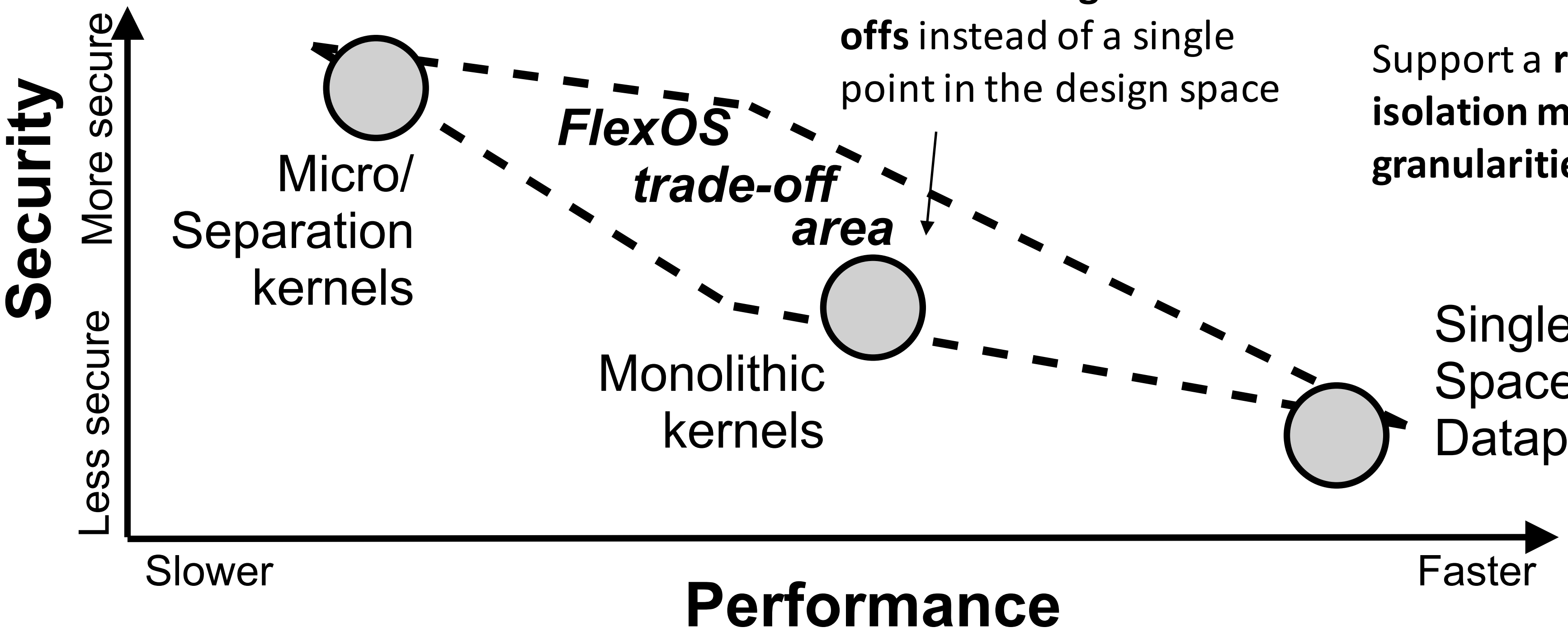
Decouple security/isolation decisions **from the OS design**



Achieve a **range of trade-offs** instead of a single point in the design space

FlexOS: Flexible Isolation

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



Achieve a **range of trade-offs** instead of a single point in the design space

Support a **range of isolation mechanisms** and **granularities**

Other Use-Cases for Flexible Isolation

Other Use-Cases for Flexible Isolation

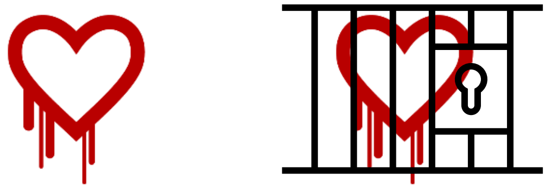
Deployment to heterogeneous hardware

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

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

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

FlexOS 101: Approach in 4 points

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

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

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**

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

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

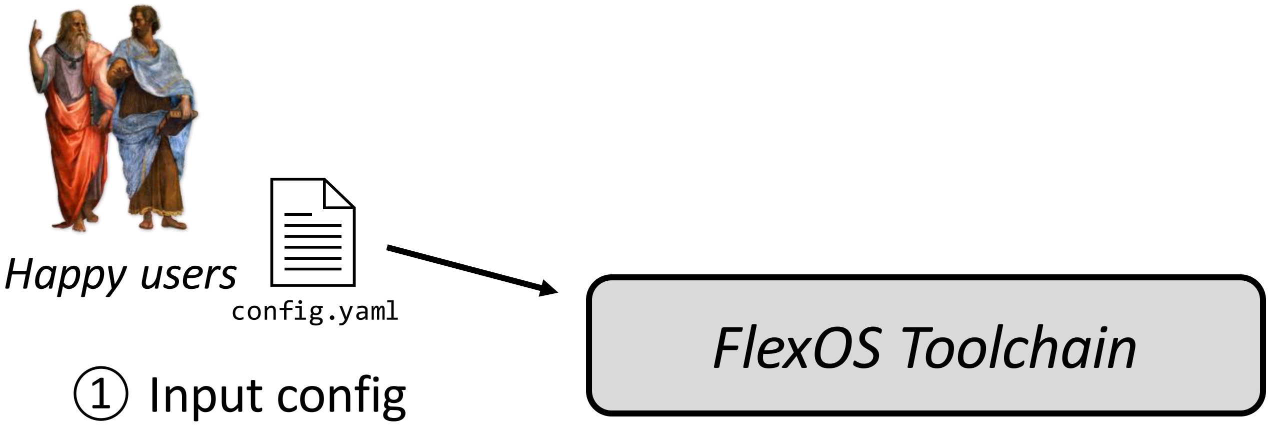
Full-system (*OS+app*) understanding of compartmentalization 2

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

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

FlexOS 101: Overview

FlexOS 101: Overview

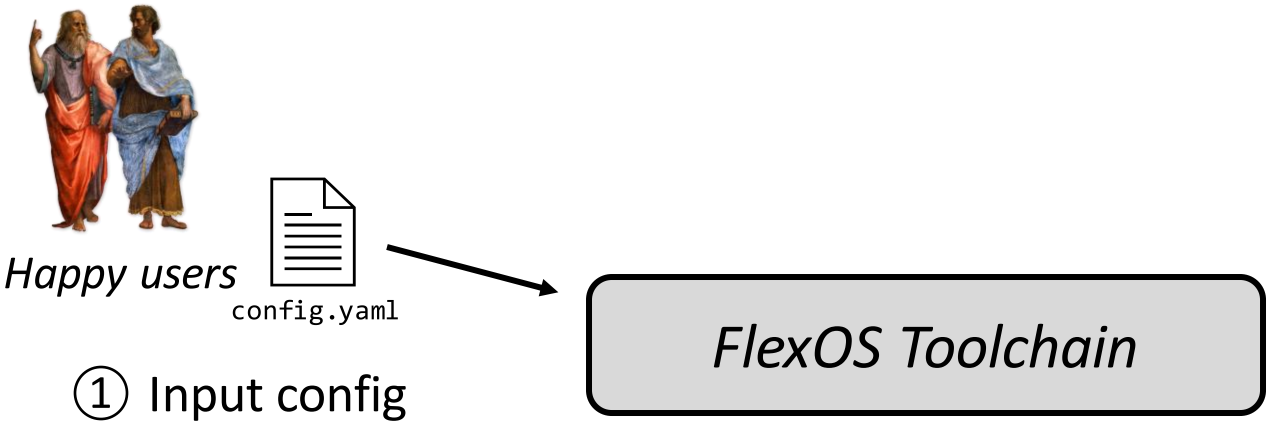


config.yaml

```
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

FlexOS Toolchain

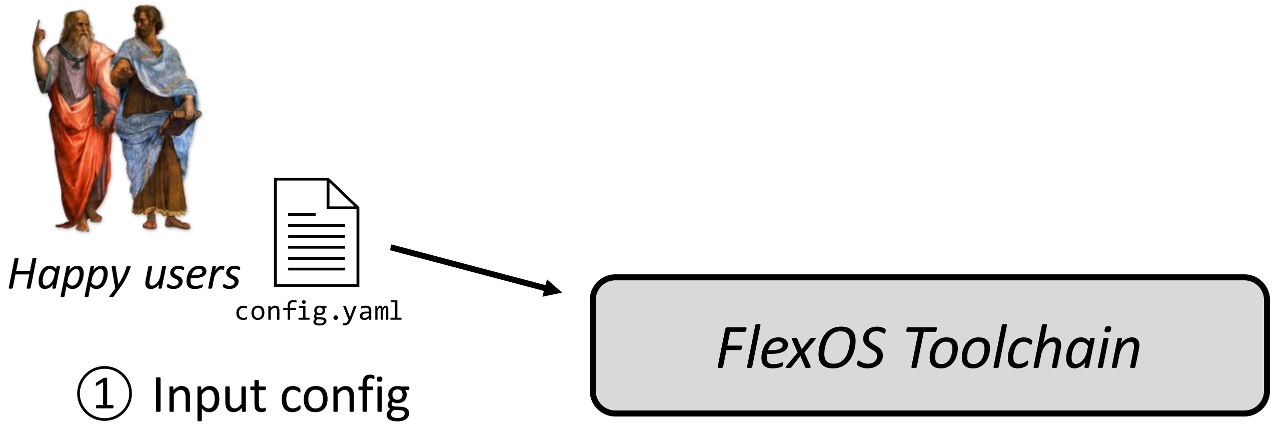
① Input config

```
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

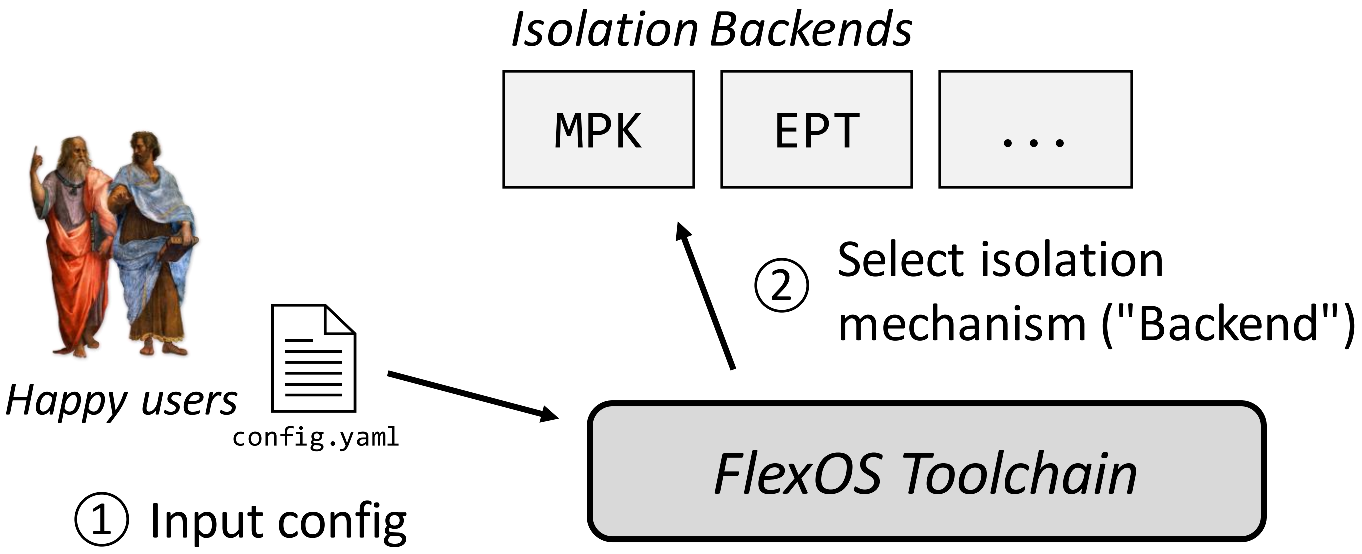


config.yaml

```
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



Isolation Backends

MPK

EPT

...

② Select isolation mechanism ("Backend")

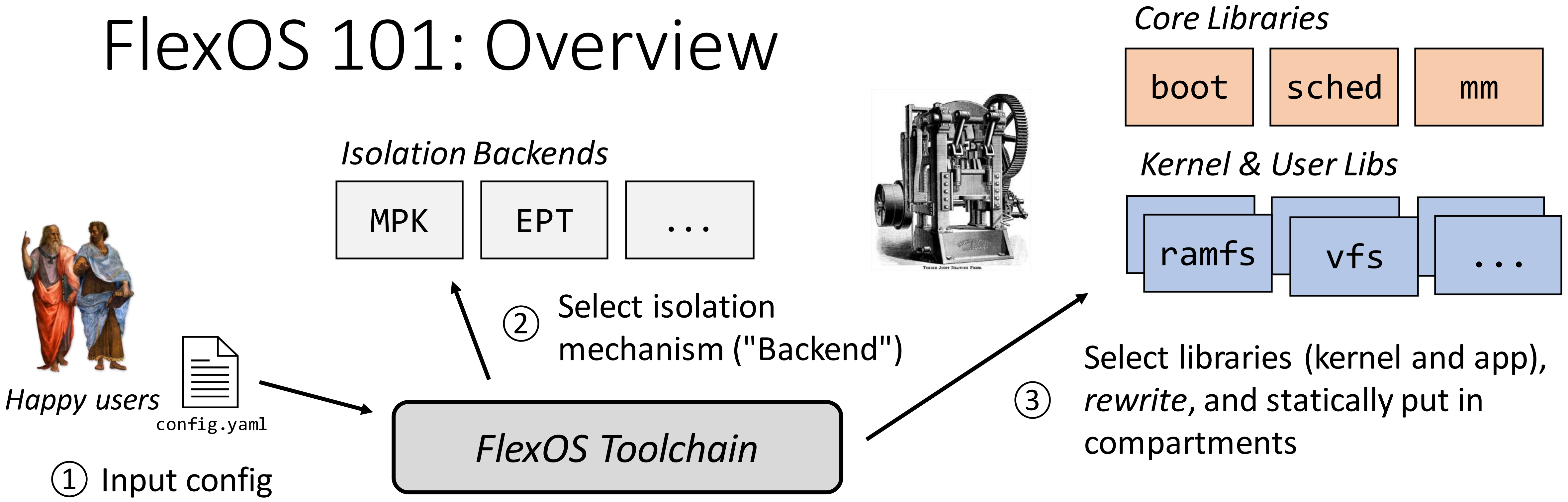
Happy users

config.yaml

① Input config

FlexOS Toolchain

FlexOS 101: Overview



Core Libraries

boot | sched | mm

Kernel & User Libs

ramfs | vfs | ...

Isolation Backends

MPK | EPT | ...

Happy users

`config.yaml`

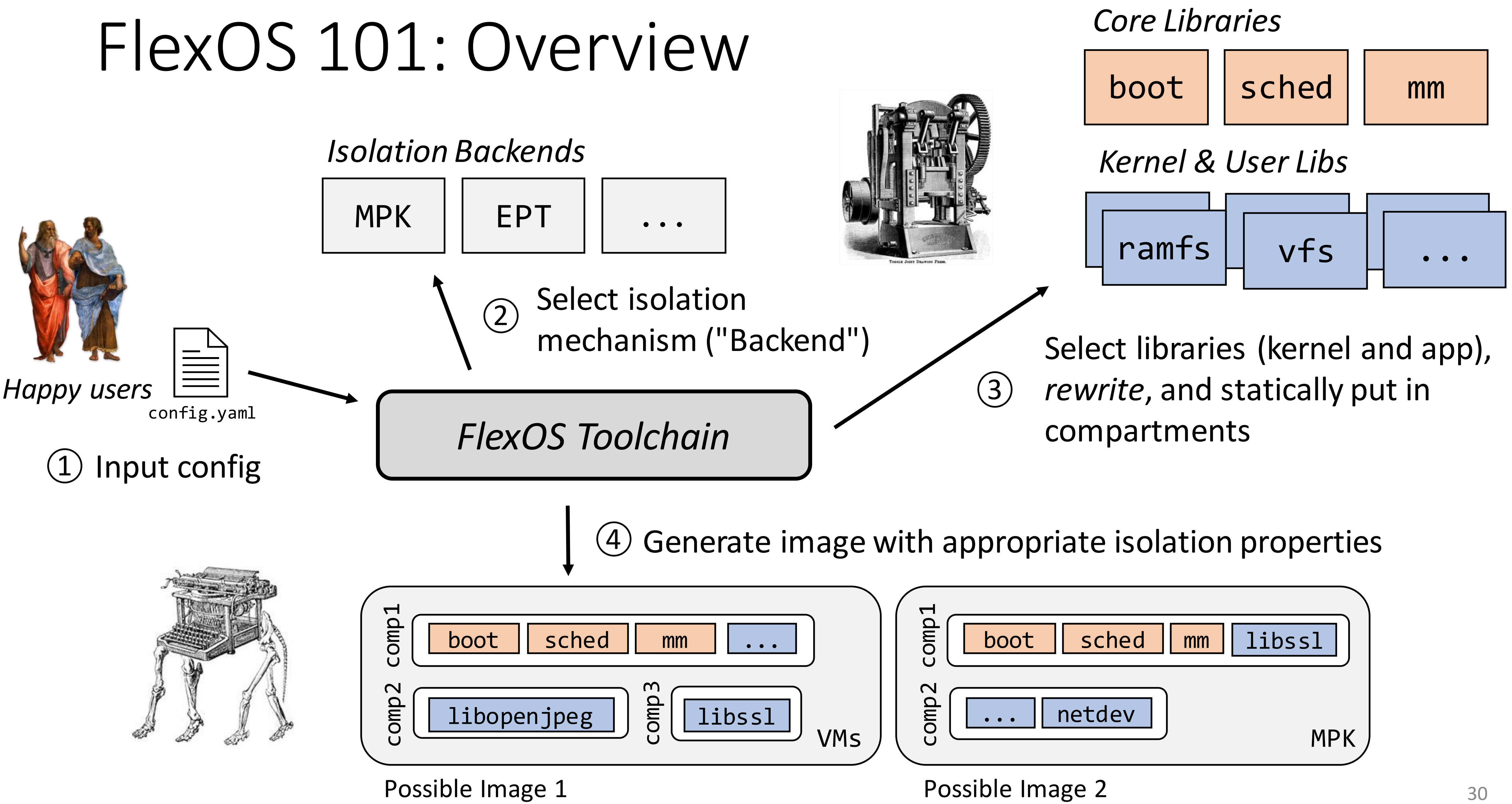
① Input config

FlexOS Toolchain

② Select isolation mechanism ("Backend")

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

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")

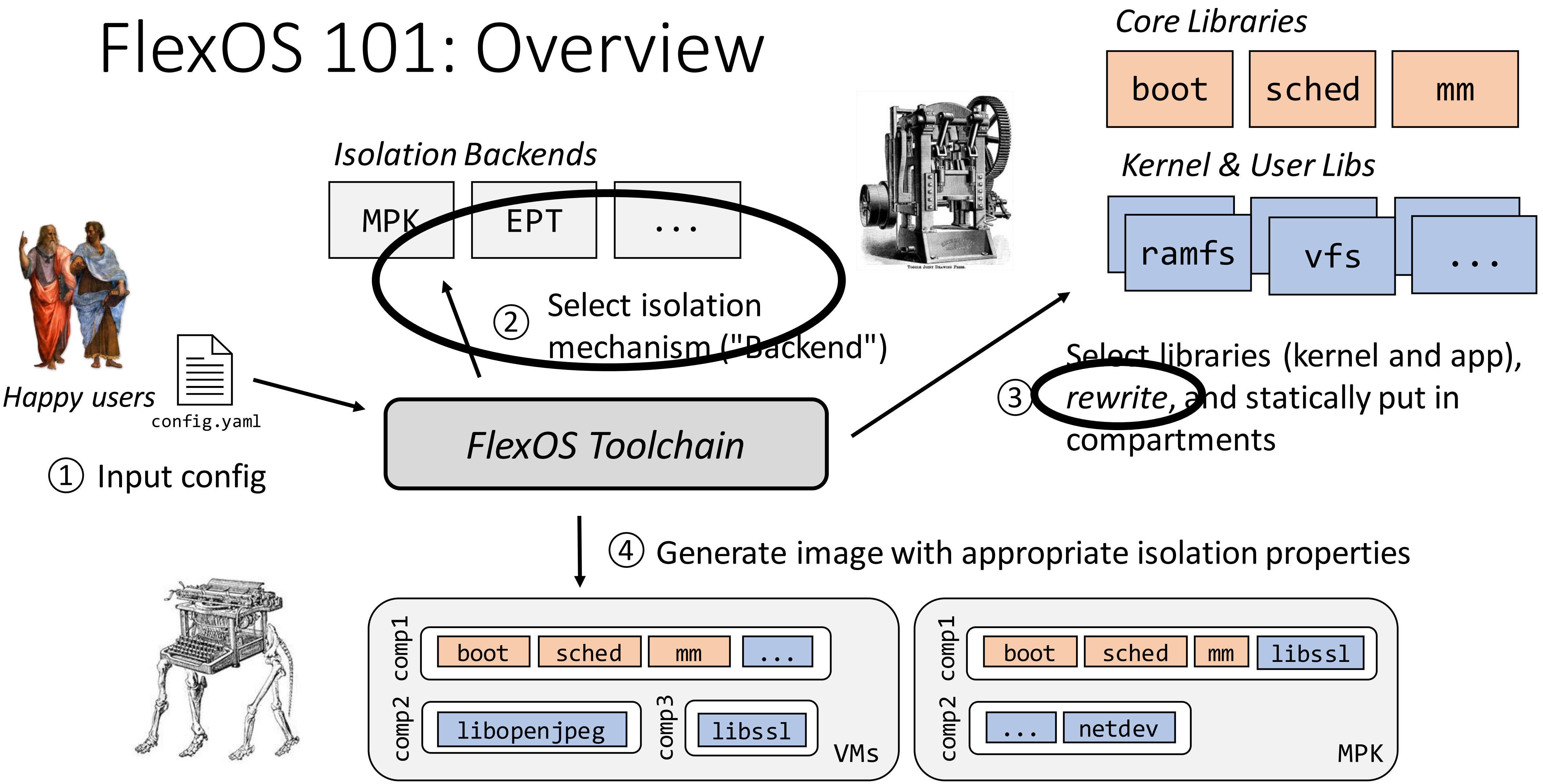
③ Select libraries (kernel and app), *rewrite*, and statically put in compartments

④ Generate image with appropriate isolation properties

Possible Image 1

Possible Image 2

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

Possible Image 1

Possible Image 2

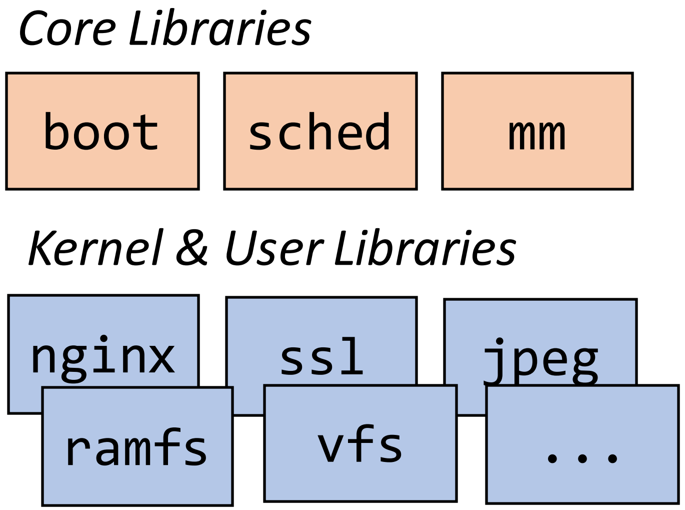
FlexOS 101: Mechanism Abstraction

Based on a **highly modular LibOS design** (Unikraft, *EuroSys'21*)

FlexOS 101: Mechanism Abstraction

Based on a **highly modular LibOS design** (Unikraft, *EuroSys'21*)

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



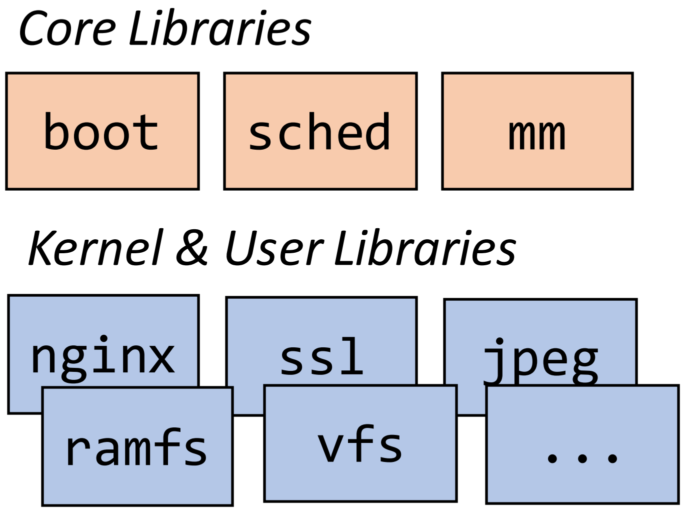
Core Libraries

Kernel & User Libraries

FlexOS 101: Mechanism Abstraction

Based on a **highly modular LibOS design** (Unikraft, *EuroSys'21*)

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



Core Libraries

boot sched mm

Kernel & User Libraries

nginx ssl jpeg

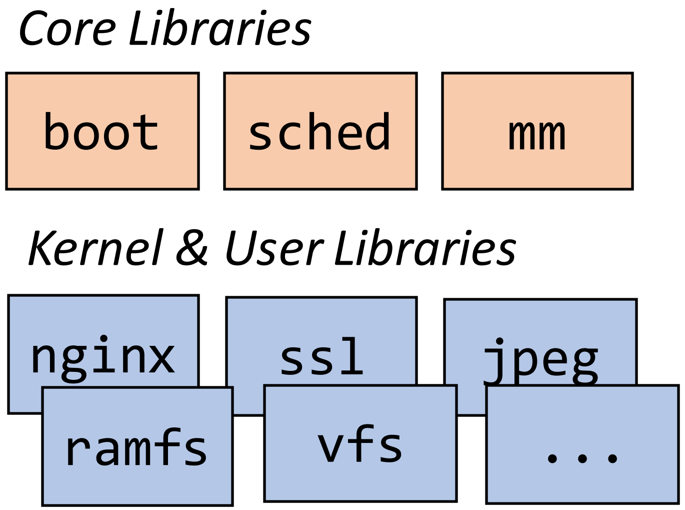
ramfs vfs ...

Reuse libraries as finest granularity of compartmentalization

FlexOS 101: Mechanism Abstraction

Based on a **highly modular LibOS design** (Unikraft, *EuroSys'21*)

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



Reuse libraries as finest granularity of compartmentalization

"Pre-compartmentalize" them

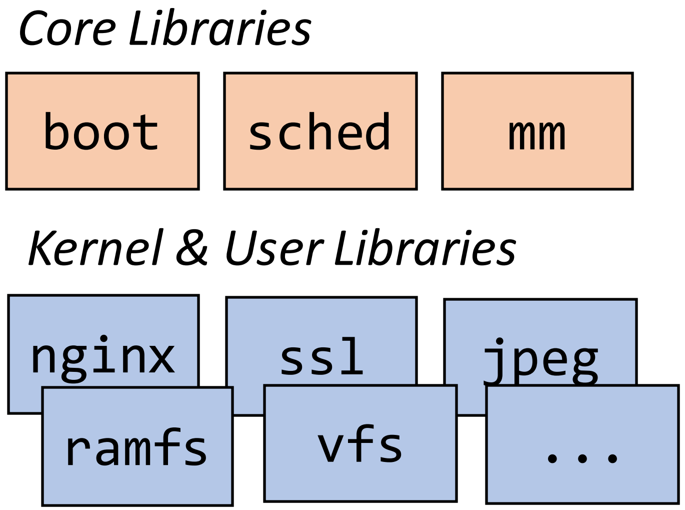
Cross-library **calls and shared data** are replaced by an **abstract construct** (**gates**, data sharing primitives)

Defined as part of the **FlexOS API**

FlexOS 101: Mechanism Abstraction

Based on a **highly modular LibOS design** (Unikraft, *EuroSys'21*)

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



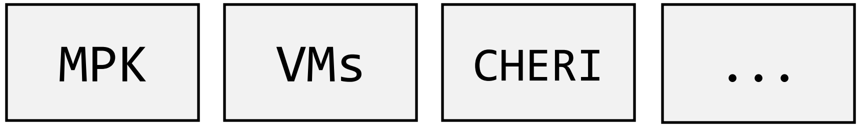
Reuse libraries as finest granularity of compartmentalization

"Pre-compartmentalize" them

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

Defined as part of the **FlexOS API**

At build time, the toolchain replaces these constructs with particular implementations. Implementations are defined by the **backends**.



FlexOS 101: Compartmentalization API

```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

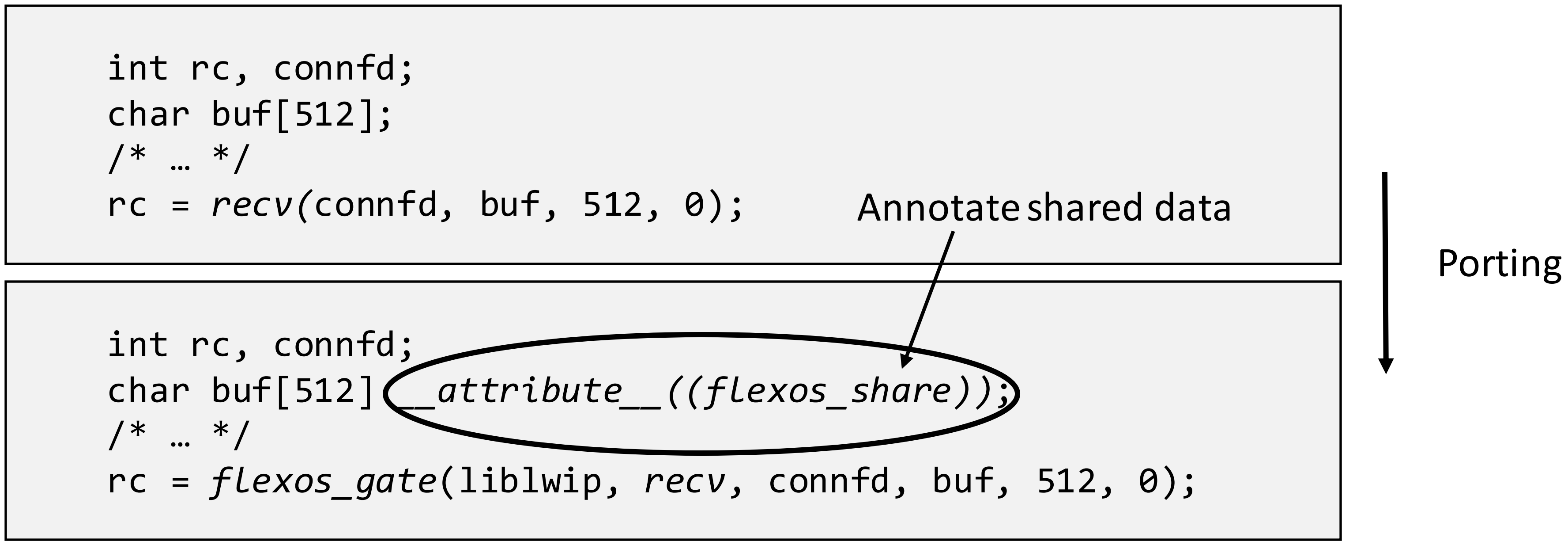
FlexOS 101: Compartmentalization API

```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

Porting

```
int rc, connfd;
char buf[512] __attribute__((flexos_share));
/* … */
rc = flexos_gate(liblwip, recv, connfd, buf, 512, 0);
```

FlexOS 101: Compartmentalization API



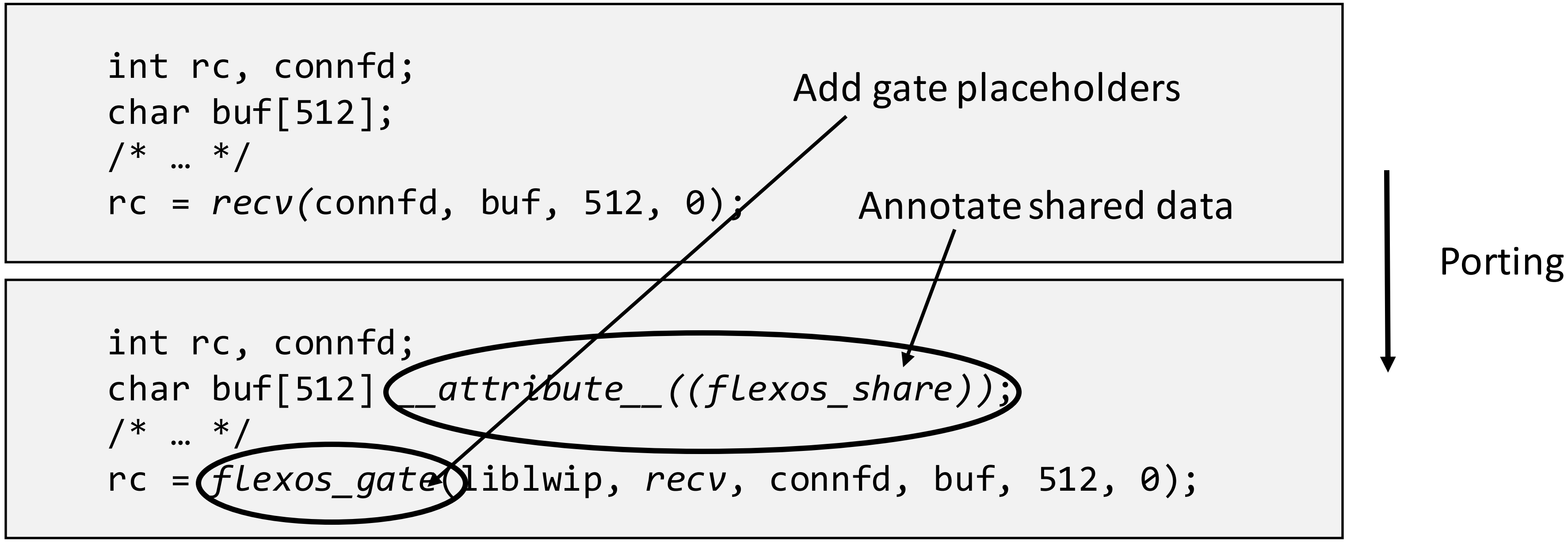
```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

Annotate shared data

Porting

```
int rc, connfd;
char buf[512] __attribute__((flexos_share));
/* … */
rc = flexos_gate(liblwip, recv, connfd, buf, 512, 0);
```

FlexOS 101: Compartmentalization API



```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

```
int rc, connfd;
char buf[512] __attribute__((flexos_share));
/* … */
rc = flexos_gate(liblwip, recv, connfd, buf, 512, 0);
```

Add gate placeholders

Annotate shared data

Porting

FlexOS 101: Compartmentalization API

```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

Add gate placeholders

Annotate shared data

```
int rc, connfd;
char buf[512] __attribute__((flexos_share));
/* … */
rc = flexos_gate(liblwip, recv, connfd, buf, 512, 0);
```

```
int rc, connfd;
char *buf[512] = shared_malloc(512);
/* … */
rc = mpk_gate(0, 1, recv, connfd, buf, 512, 0);
```

lwip 1

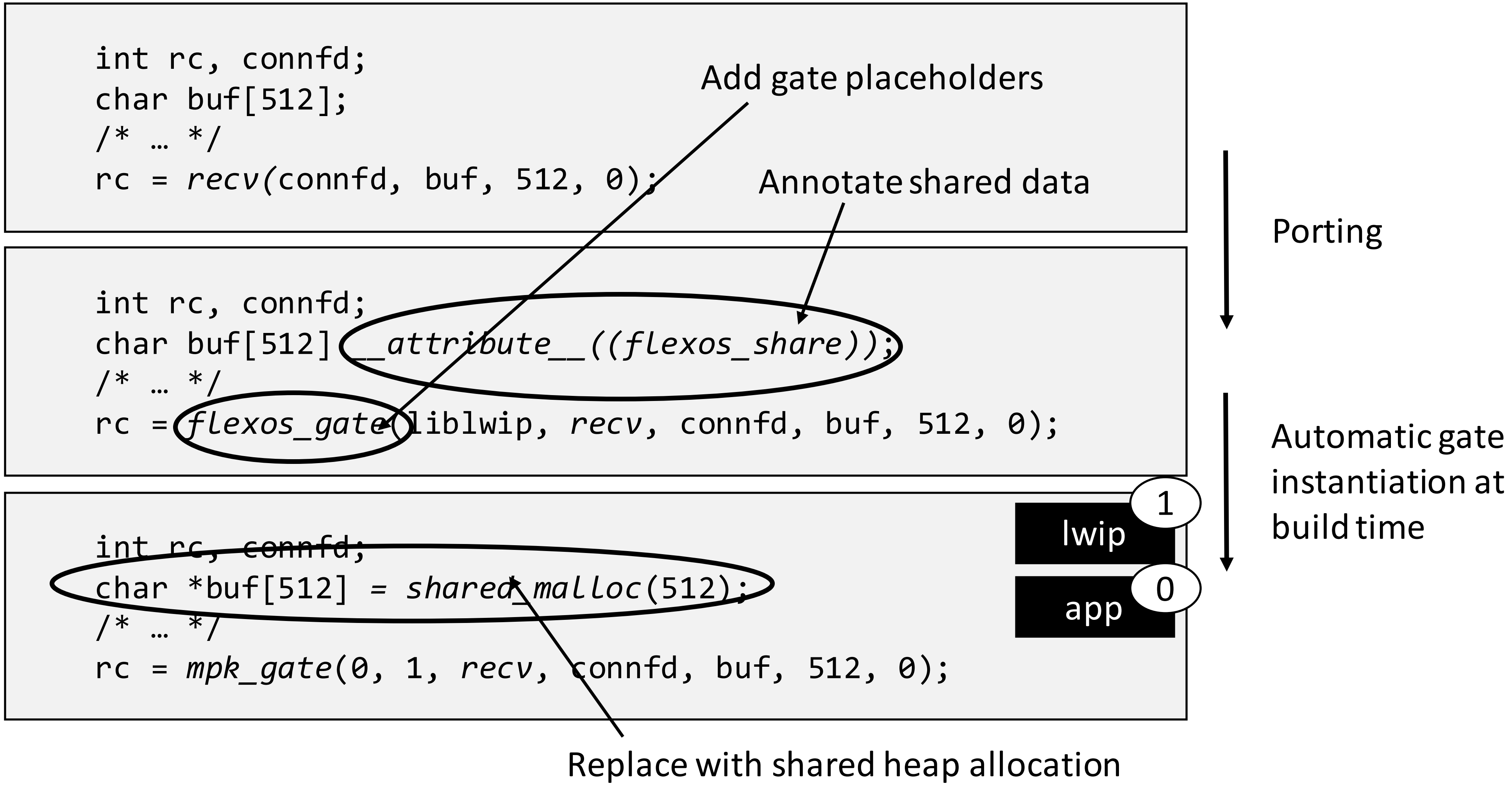
app 0

Porting

Automatic gate instantiation at build time

Coccinelle

FlexOS 101: Compartmentalization API



```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

```
int rc, connfd;
char buf[512] __attribute__((flexos_share));
/* … */
rc = flexos_gate(liblwip, recv, connfd, buf, 512, 0);
```

```
int rc, connfd;
char *buf[512] = shared_malloc(512);
/* … */
rc = mpk_gate(0, 1, recv, connfd, buf, 512, 0);
```

Add gate placeholders

Annotate shared data

Replace with shared heap allocation

Porting

Automatic gate instantiation at build time

Coccinelle

FlexOS 101: Compartmentalization API

```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

```
int rc, connfd;
char buf[512] __attribute__((flexos_share));
/* … */
rc = flexos_gate(liblwip, recv, connfd, buf, 512, 0);
```

```
int rc, connfd;
char *buf[512] = shared_malloc(512);
/* … */
rc = mpk_gate(0, 1, recv, connfd, buf, 512, 0);
```

Add gate placeholders

Annotate shared data

Replace with MPK gate

Replace with shared heap allocation

lwip 1

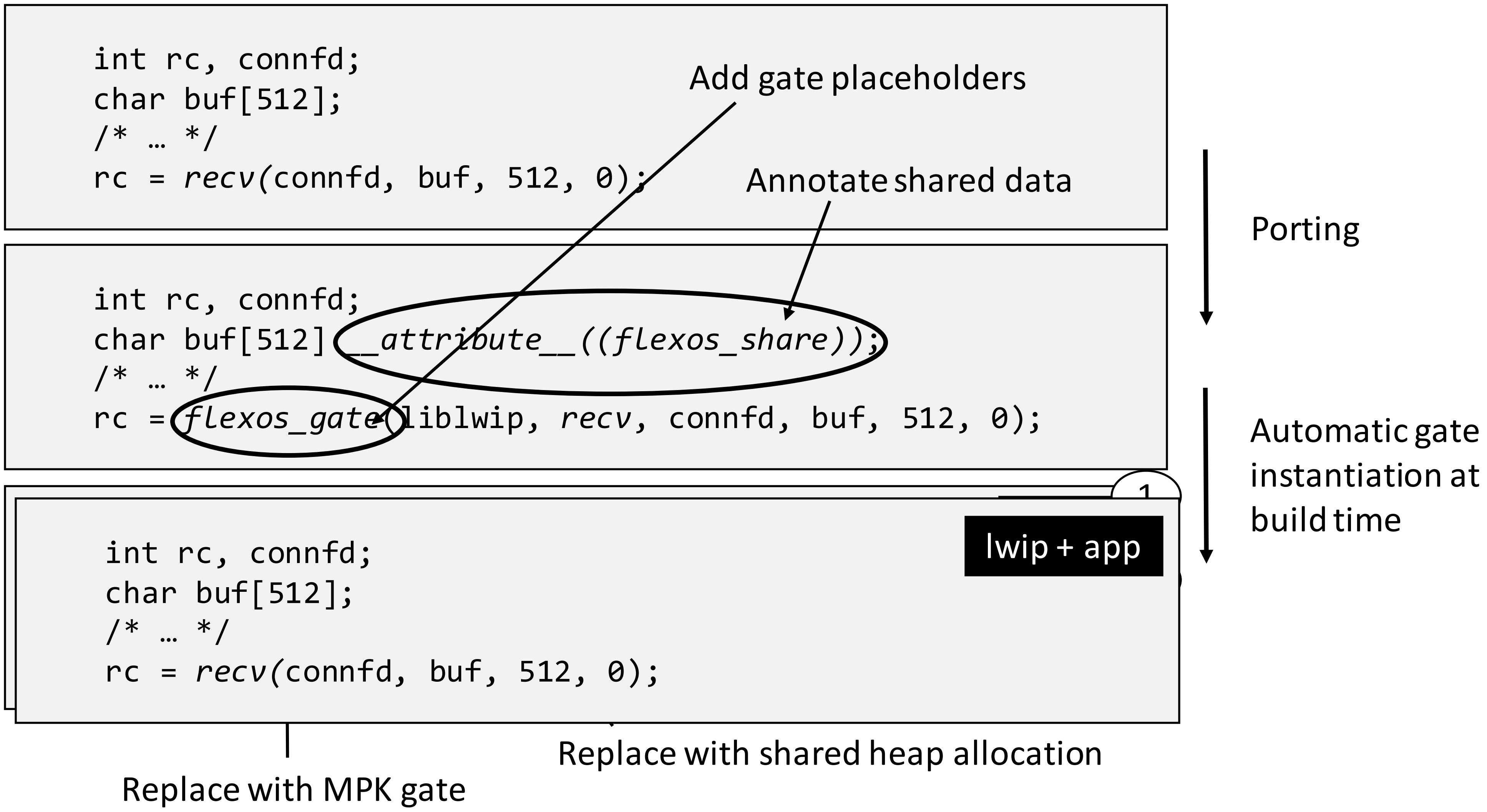
app 0

Porting

Automatic gate instantiation at build time

Coccinelle

FlexOS 101: Compartmentalization API



```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

Add gate placeholders

Annotate shared data

Porting

```
int rc, connfd;
char buf[512] __attribute__((flexos_share));
/* … */
rc = flexos_gate(liblwip, recv, connfd, buf, 512, 0);
```

Automatic gate instantiation at build time

lwip + app

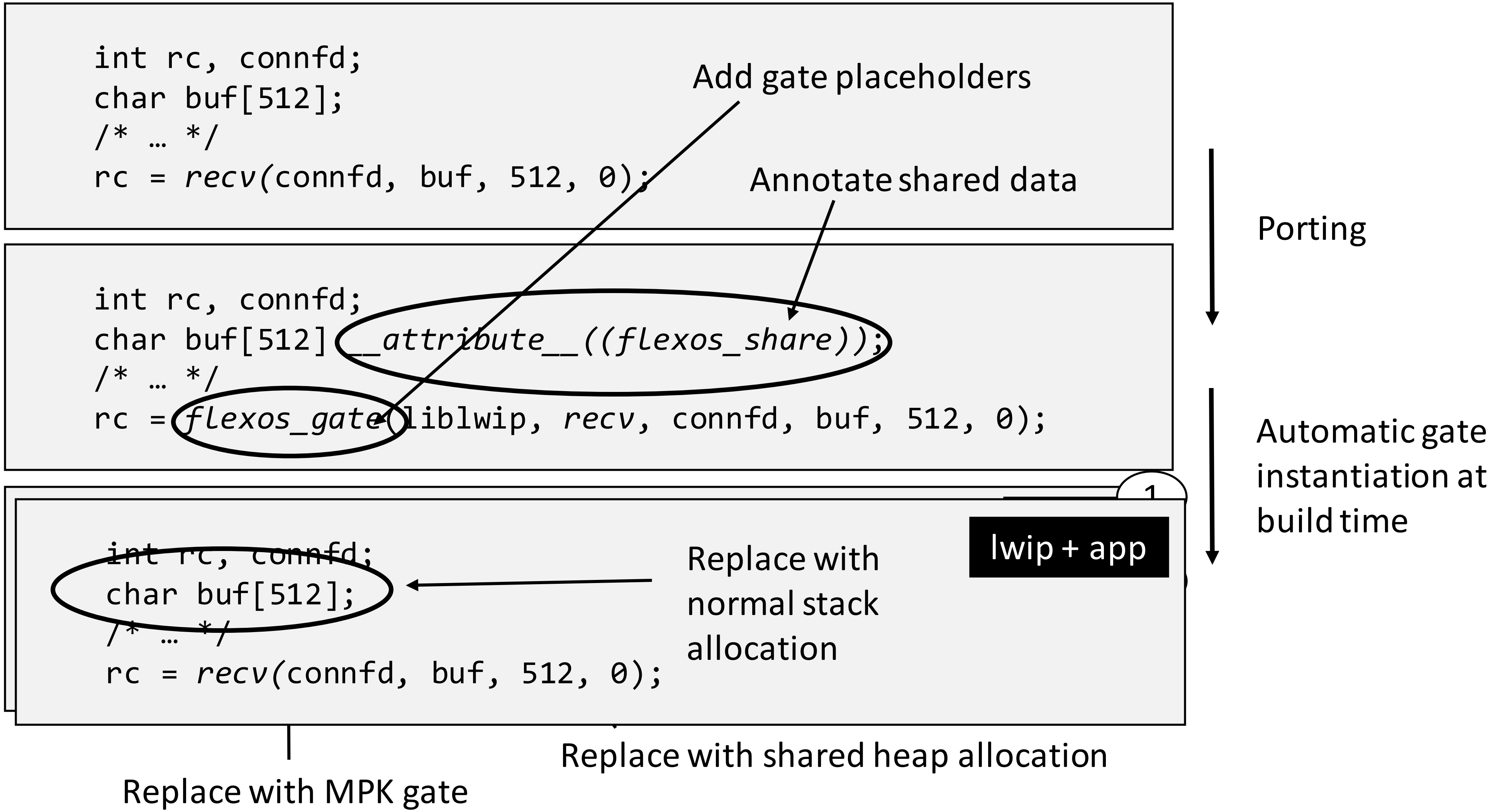
```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

Replace with shared heap allocation

Replace with MPK gate

Coccinelle

FlexOS 101: Compartmentalization API



```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

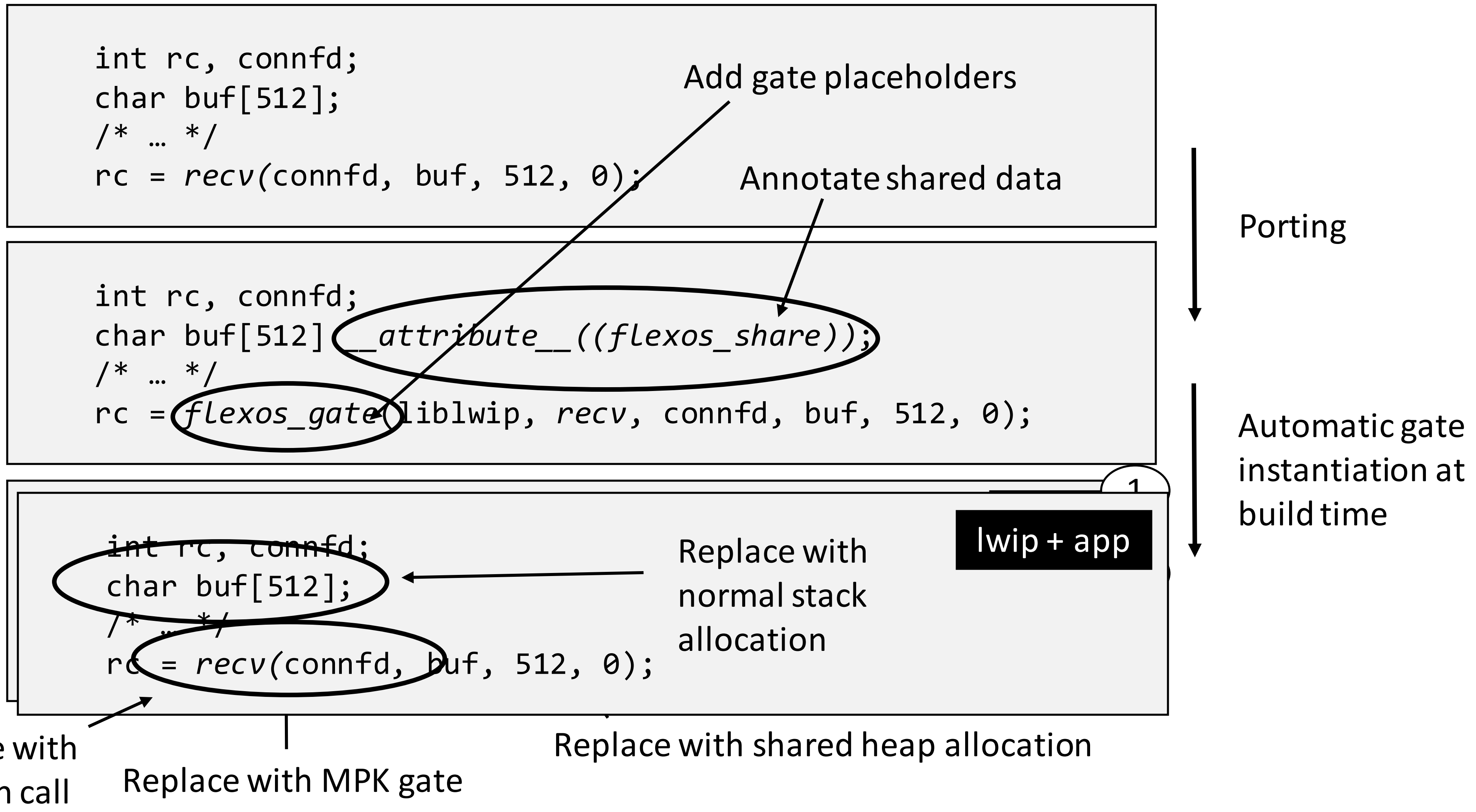
```
int rc, connfd;
char buf[512] __attribute__((flexos_share));
/* … */
rc = flexos_gate(liblwip, recv, connfd, buf, 512, 0);
```

```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

Add gate placeholders
Annotate shared data
Porting
Automatic gate instantiation at build time
1
lwip + app
Replace with normal stack allocation
Replace with shared heap allocation
Replace with MPK gate

Coccinelle

FlexOS 101: Compartmentalization API



```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

```
int rc, connfd;
char buf[512] __attribute__((flexos_share));
/* … */
rc = flexos_gate(liblwip, recv, connfd, buf, 512, 0);
```

```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

Add gate placeholders
Annotate shared data
Porting
Automatic gate instantiation at build time
1
lwip + app
Replace with normal stack allocation
Replace with function call
Replace with MPK gate
Replace with shared heap allocation

Coccinelle

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

Prototype

Implementation **on top of Unikraft**

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

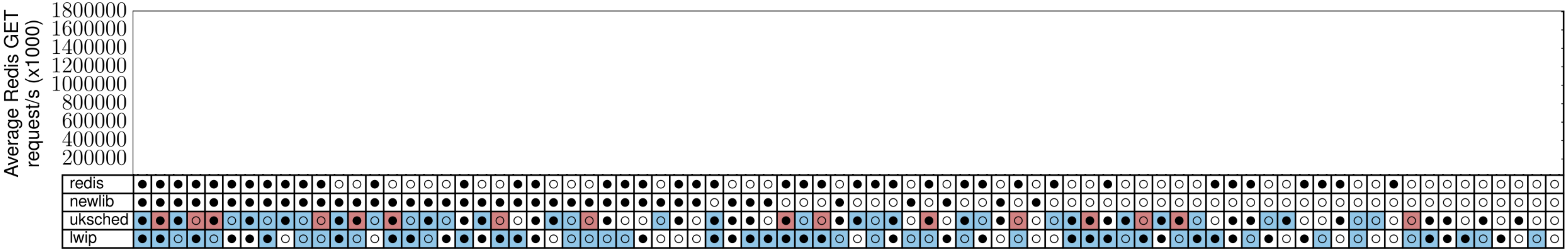
Port of libraries: network stack, scheduler, filesystem, time subsystem

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

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

more results in our paper 🙂

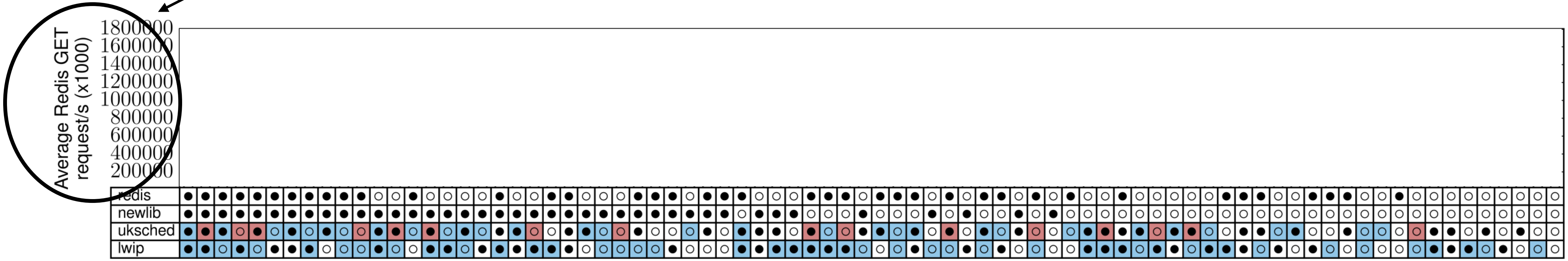
Flexibility



Flexibility

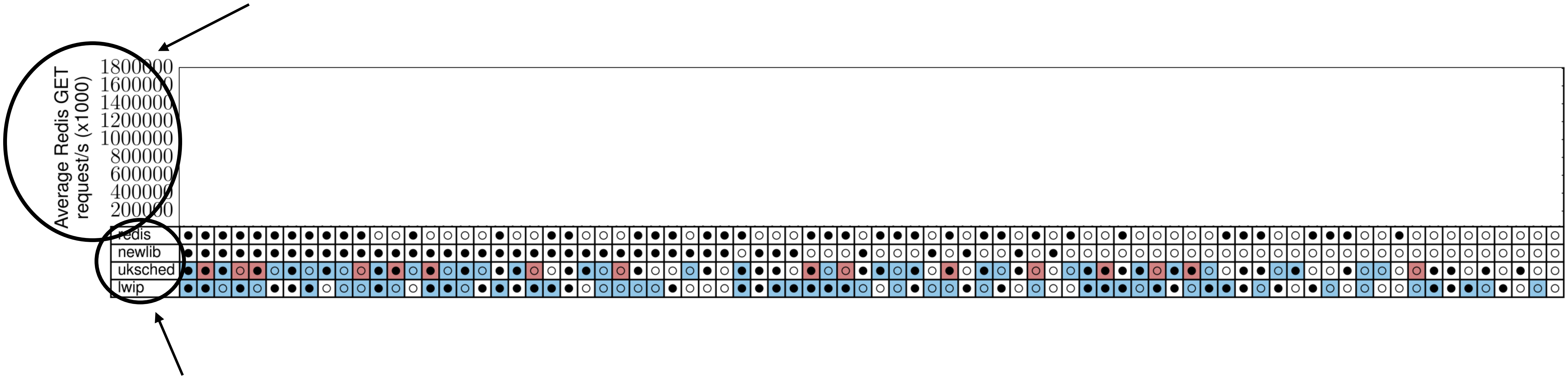
redis

Runtime performance with Redis in requests/s



Flexibility

Runtime performance with Redis in requests/s



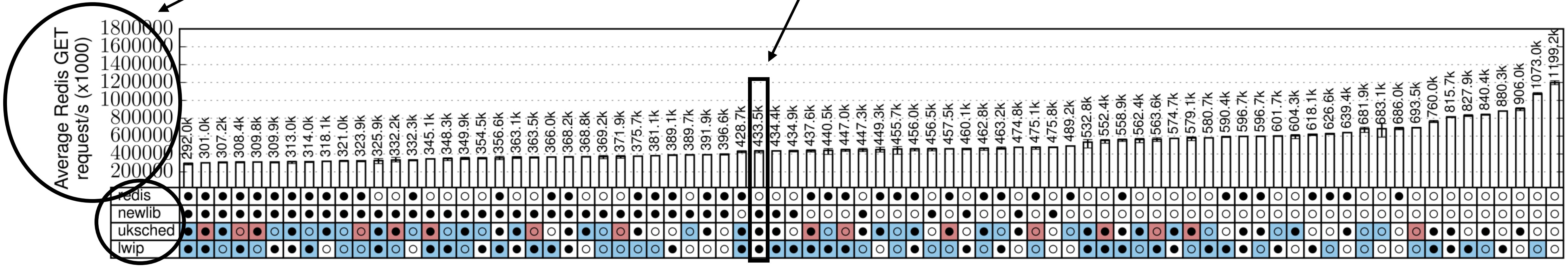
FlexOS libraries used in the Redis image (only a subset for readability):

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

Flexibility

Runtime performance with Redis in requests/s

One configuration and its associated performance (80 configurations in total)



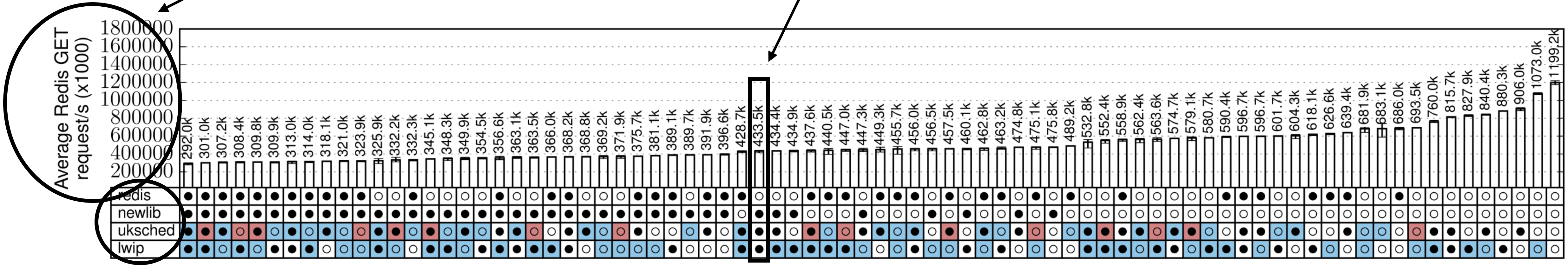
FlexOS libraries used in the Redis image (only a subset for readability):

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

Flexibility

Runtime performance with Redis in requests/s

One configuration and its associated performance (80 configurations in total)



The color of boxes indicates the compartment:

□ Compartment 1 🟥 Compartment 2 🟦 Compartment 3

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

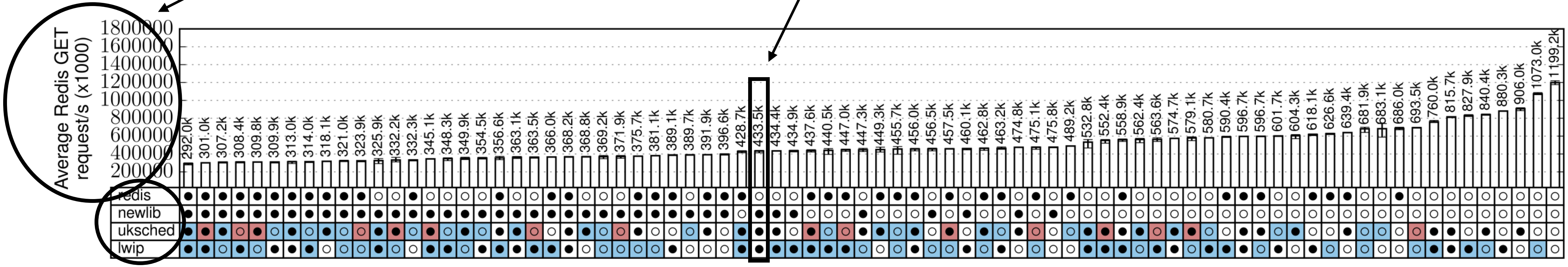
- Redis application
- C standard library (newlib)
- FlexOS scheduler (*uksched*)
- Network stack (lwip)

Flexibility

redis

Runtime performance with Redis in requests/s

One configuration and its associated performance (80 configurations in total)



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

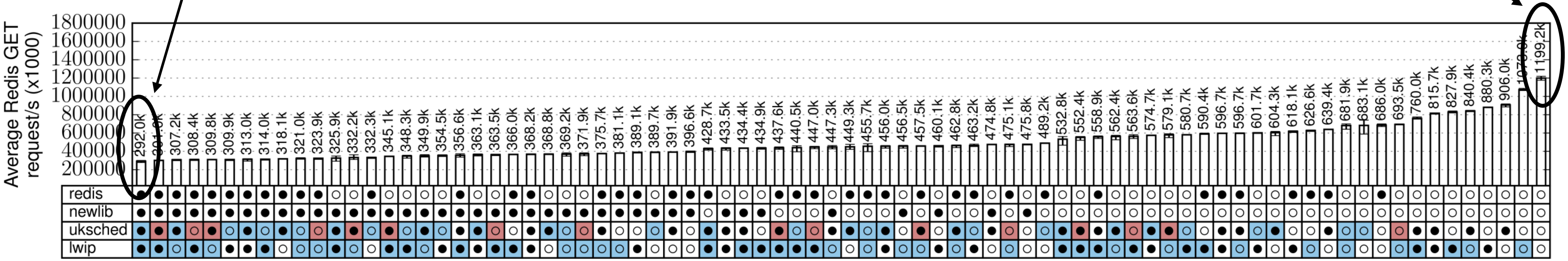
The dot whether hardening (ASan, Safestack, etc.) is enabled:

● Hardening on ○ Hardening off

Flexibility

① Large safety / performance space! (4x)

292K requests/s ⟷ 1.2M requests/s

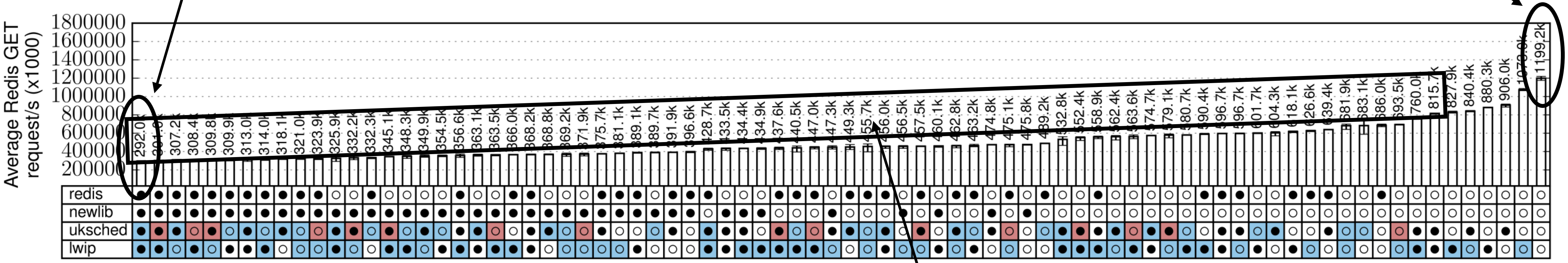


● Hardening on ○ Hardening off □ Compartment 1 🟥 Compartment 2 🟦 Compartment 3

Flexibility

① Large safety / performance space! (4x)

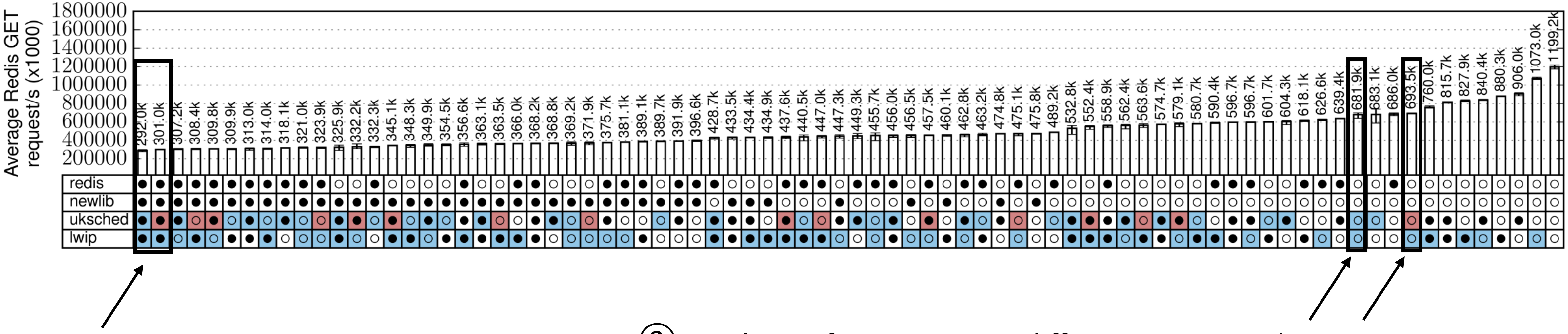
292K requests/s ⟵⟶ 1.2M requests/s



② Smooth slope, performance degrades gracefully

● Hardening on ○ Hardening off □ Compartment 1 🟥 Compartment 2 🟦 Compartment 3

Flexibility



③ Similar performance, very different properties!

need to reason about communication patterns, fast paths

● Hardening on

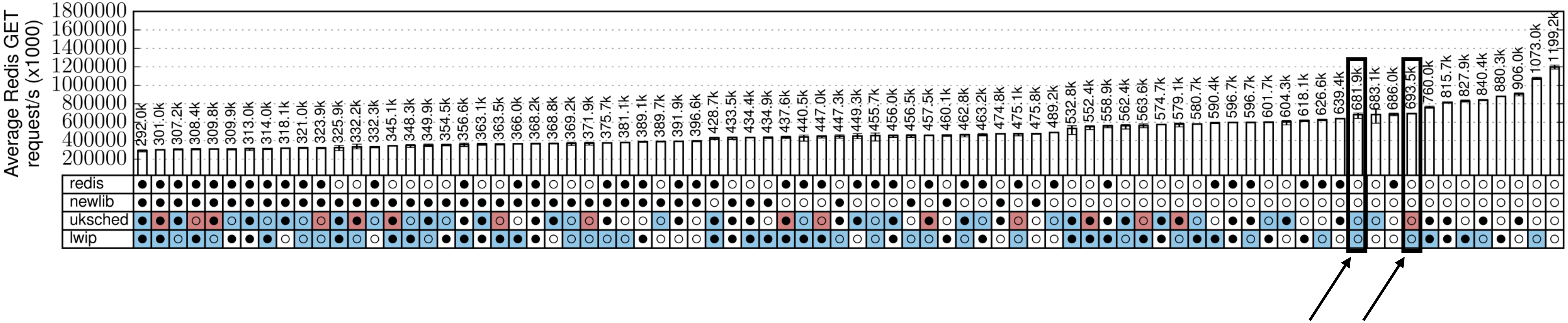
○ Hardening off

□ Compartment 1

🟥 Compartment 2

🟦 Compartment 3

Flexibility

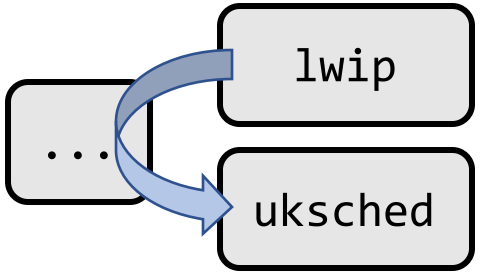


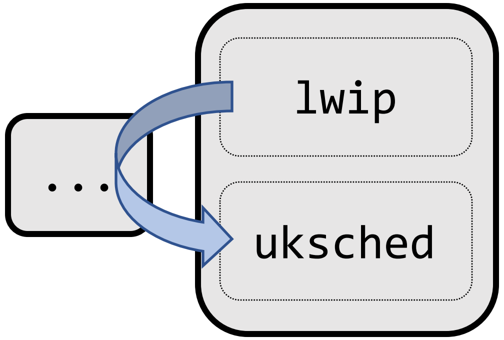
③ Similar performance, very different properties!

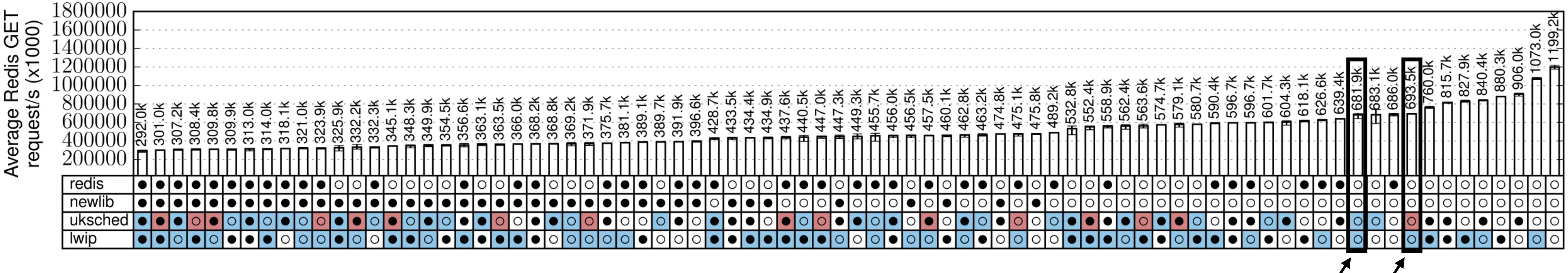
need to reason about communication patterns, fast paths

● Hardening on ○ Hardening off □ Compartment 1 🟥 Compartment 2 🟦 Compartment 3

Flexibility







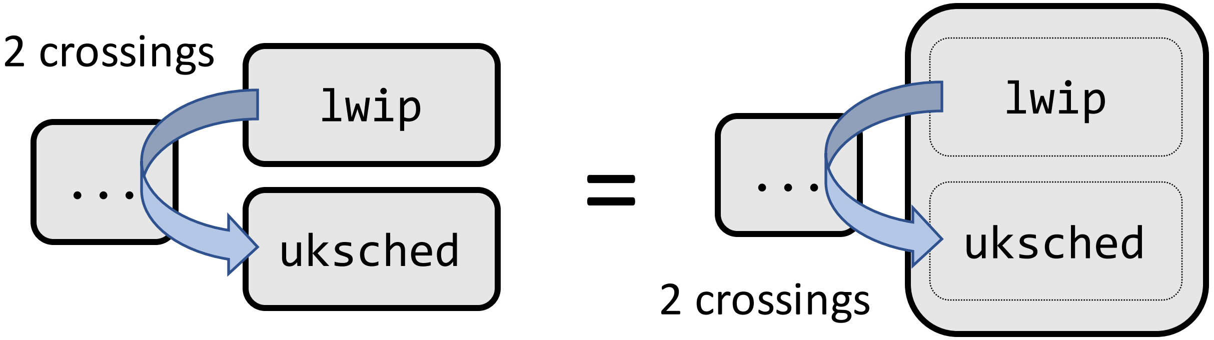
③ Similar performance, very different properties!

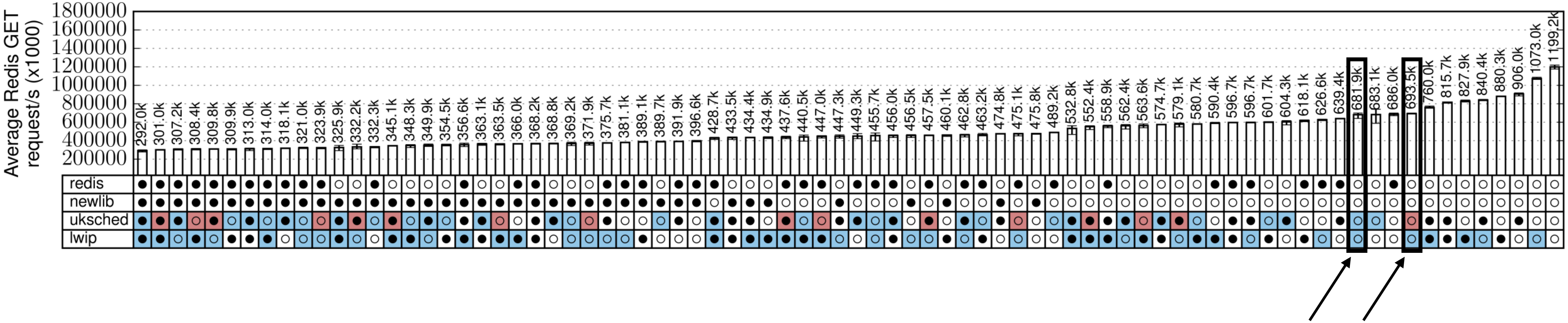
need to reason about communication patterns, fast paths

● Hardening on ○ Hardening off □ Compartment 1 🟥 Compartment 2 🟦 Compartment 3

Flexibility

Performance-wise:





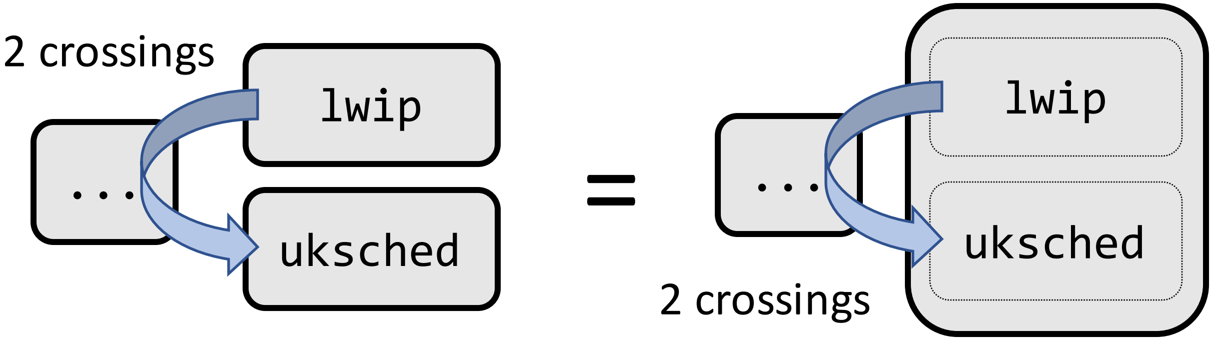
③ Similar performance, very different properties!

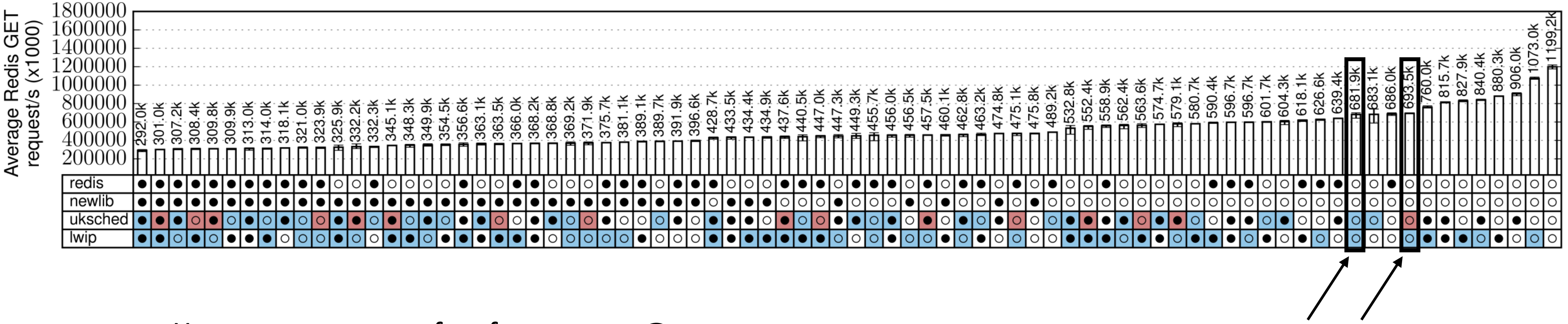
need to reason about communication patterns, fast paths

● Hardening on ○ Hardening off □ Compartment 1 🟥 Compartment 2 🟦 Compartment 3

Flexibility

Performance-wise:





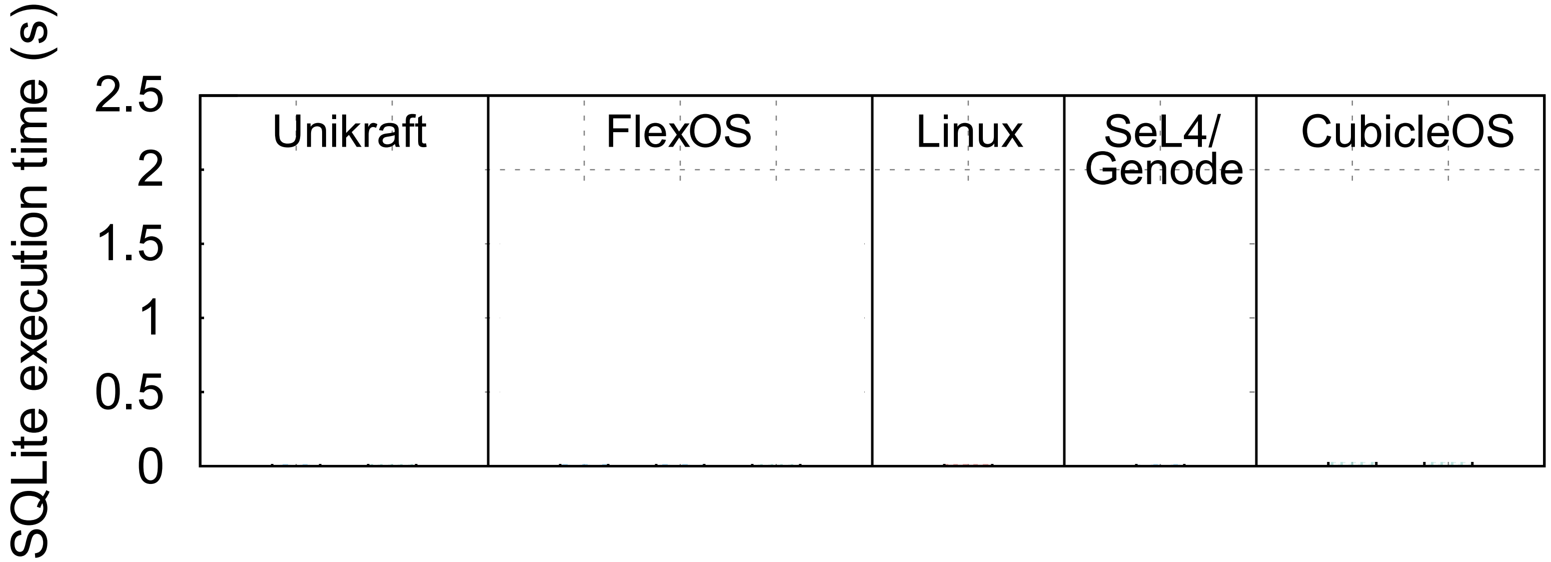
You can get some safety for free by exploring intelligently

③ Similar performance, very different properties!

need to reason about communication patterns, fast paths

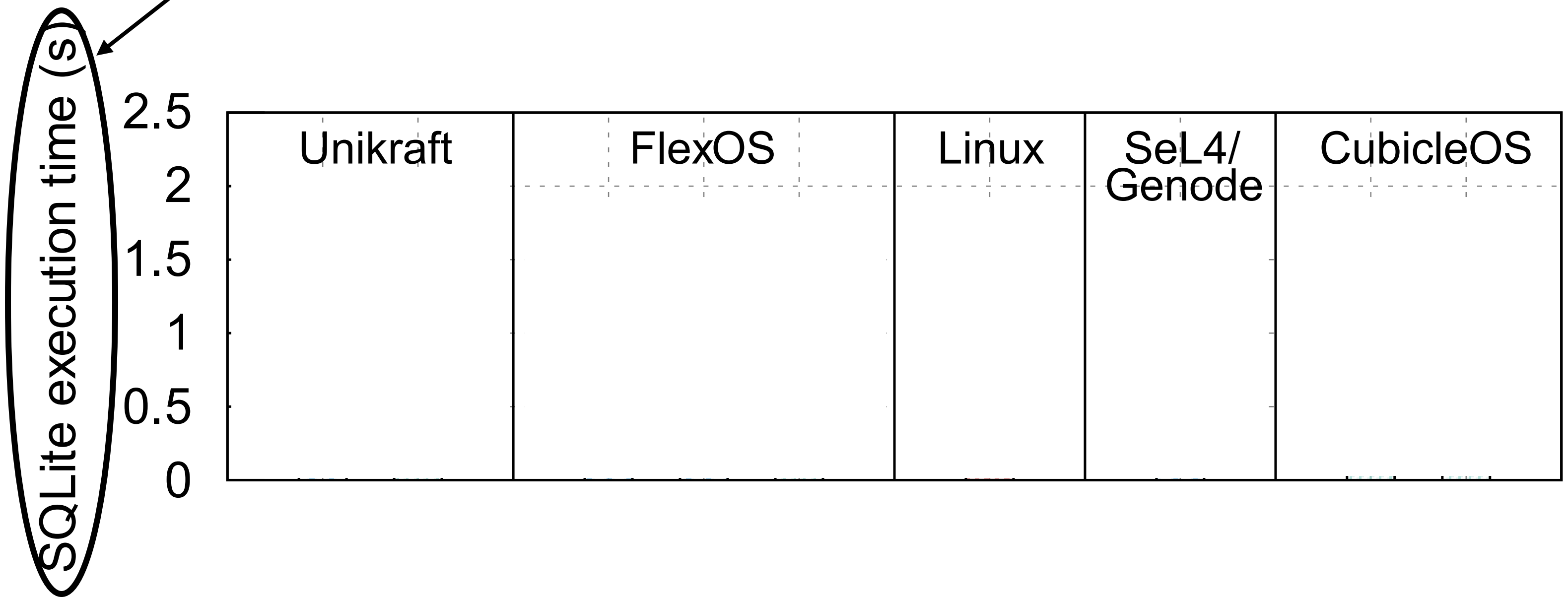
● Hardening on ○ Hardening off □ Compartment 1 🟥 Compartment 2 🟦 Compartment 3

Performance

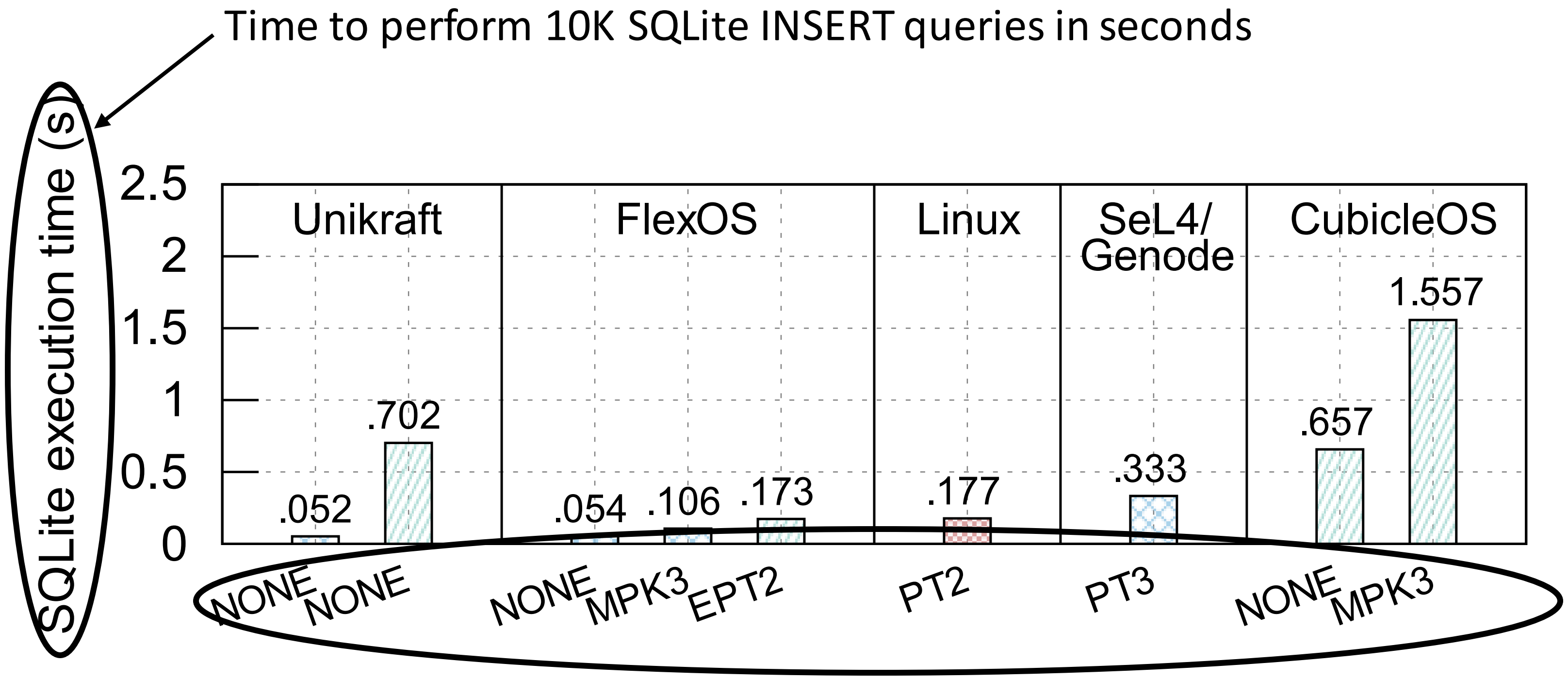


Performance

Time to perform 10K SQLite INSERT queries in seconds



Performance

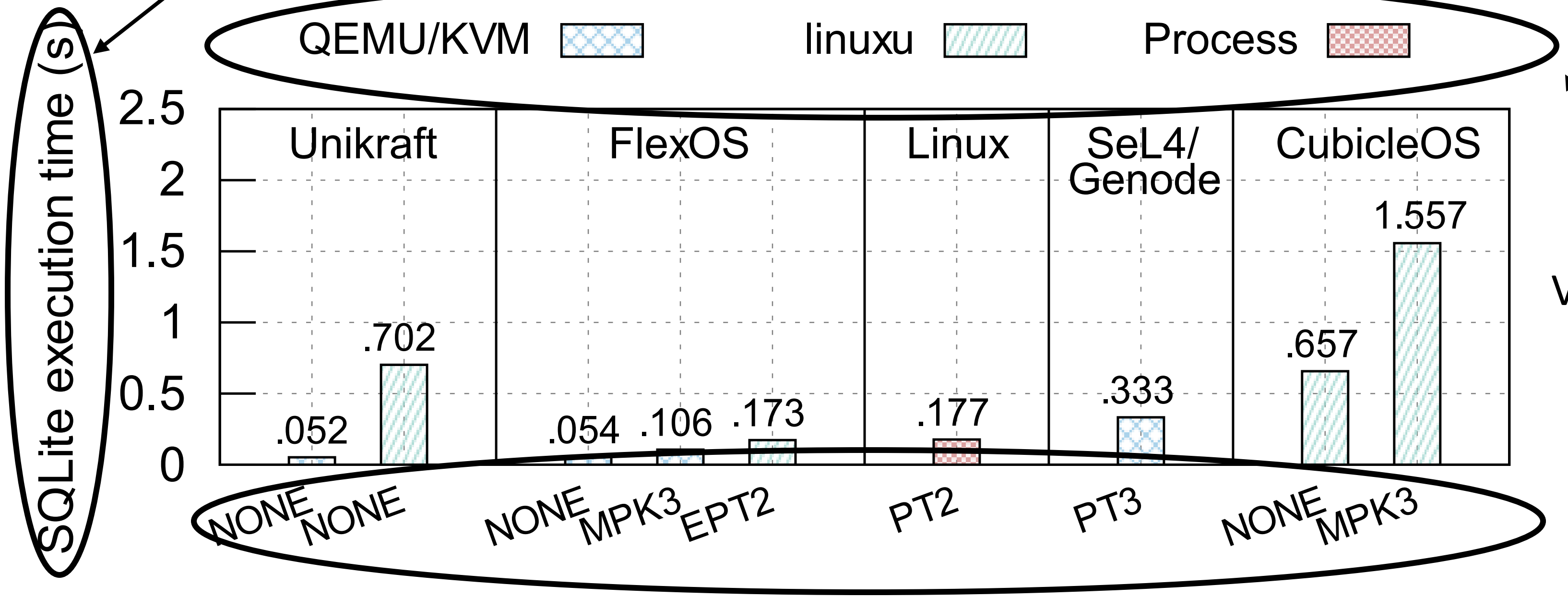


Time to perform 10K SQLite INSERT queries in seconds

Number of compartments and mechanism (e.g., PT2 = 2 compartments with the page table)

Performance

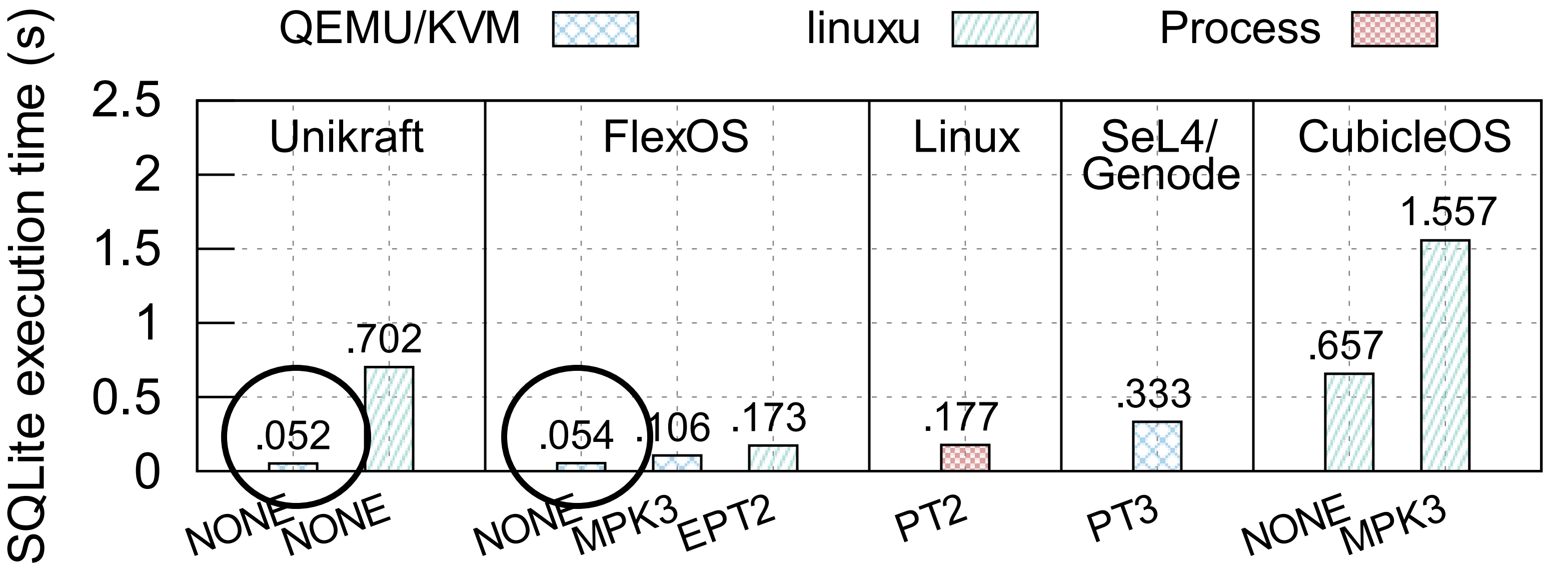
Time to perform 10K SQLite INSERT queries in seconds



VMM/environment

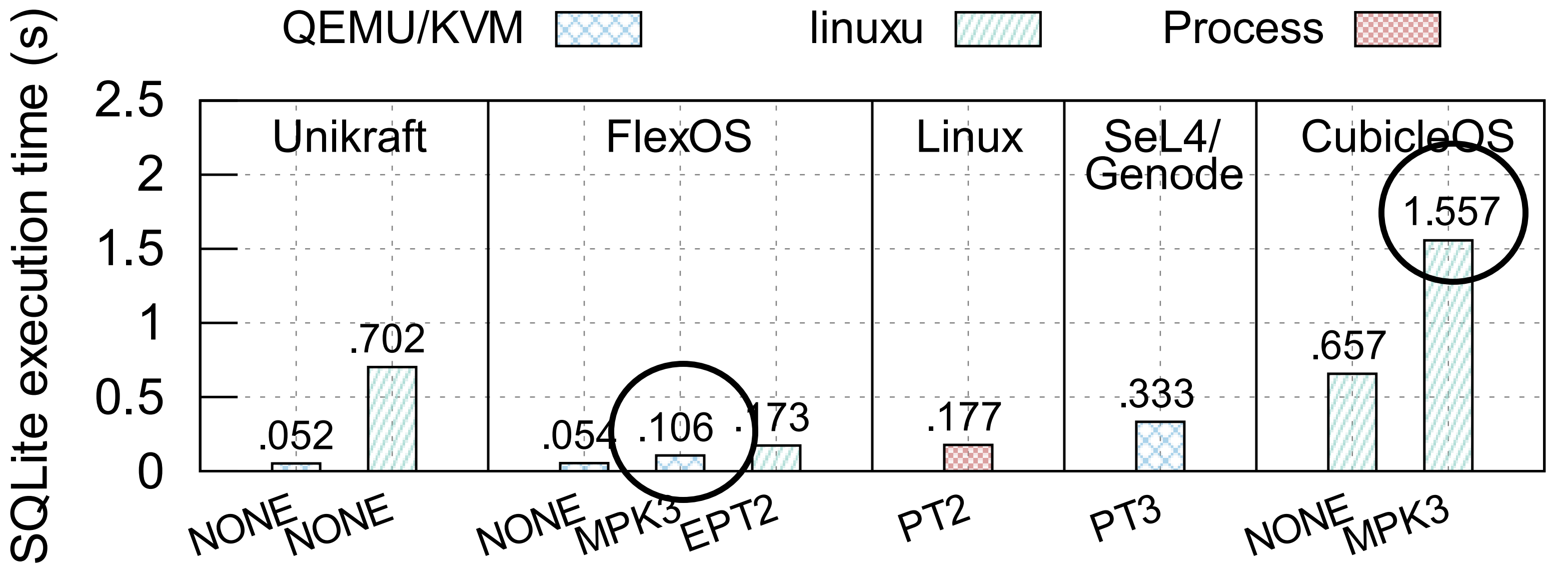
Number of compartments and mechanism (e.g., PT2 = 2 compartments with the page table)

Performance



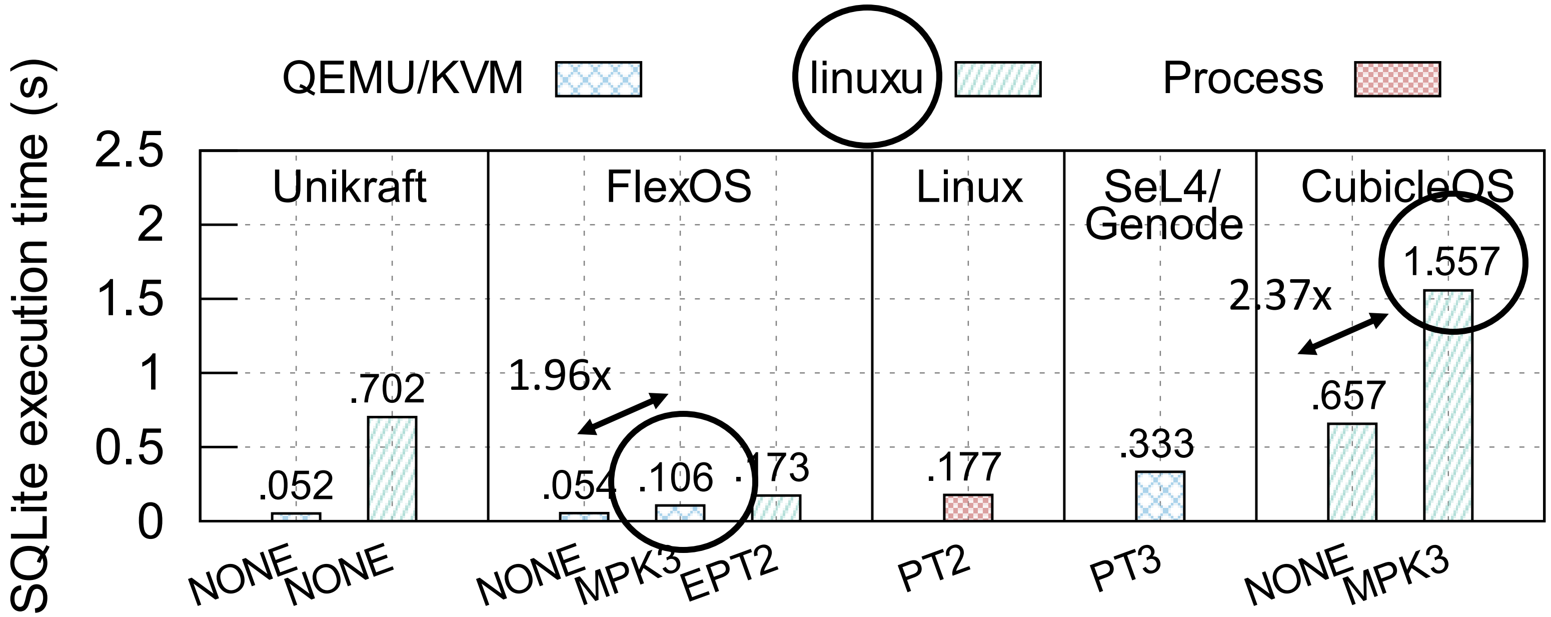
① No overhead when disabling isolation – you only pay for what you get

Performance



② The MPK backend compares very positively to competing solutions

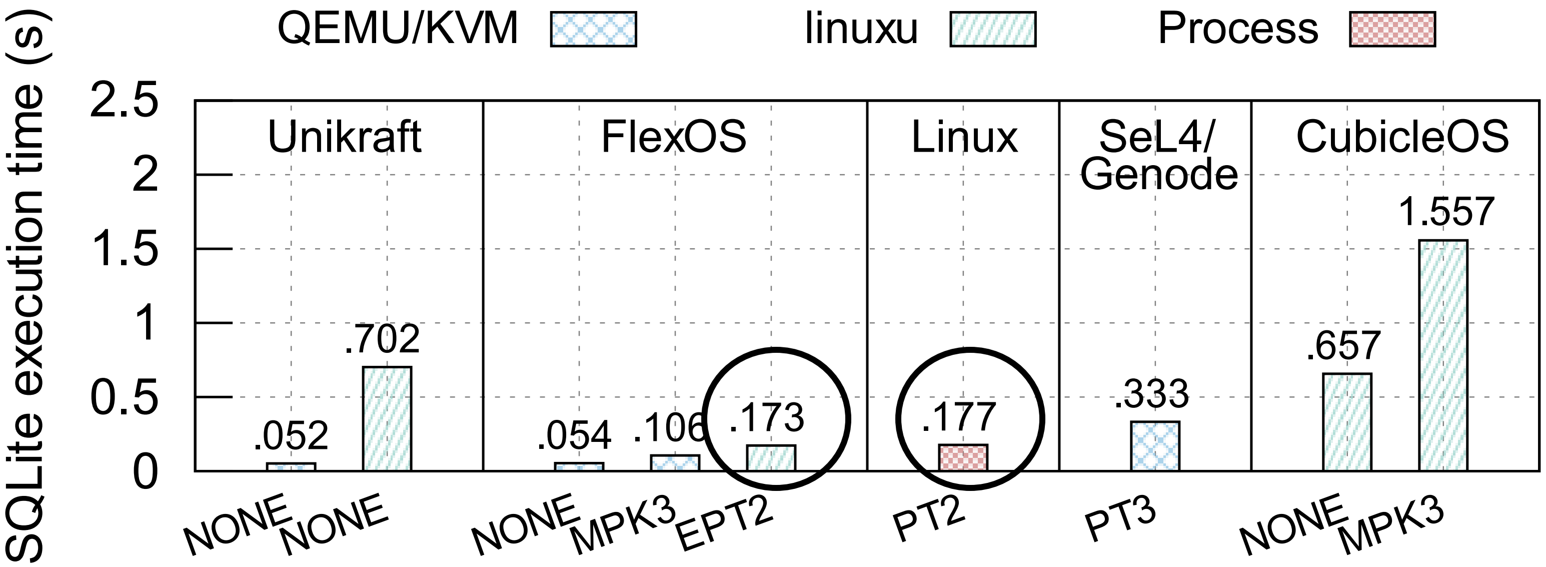
Performance



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!

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!

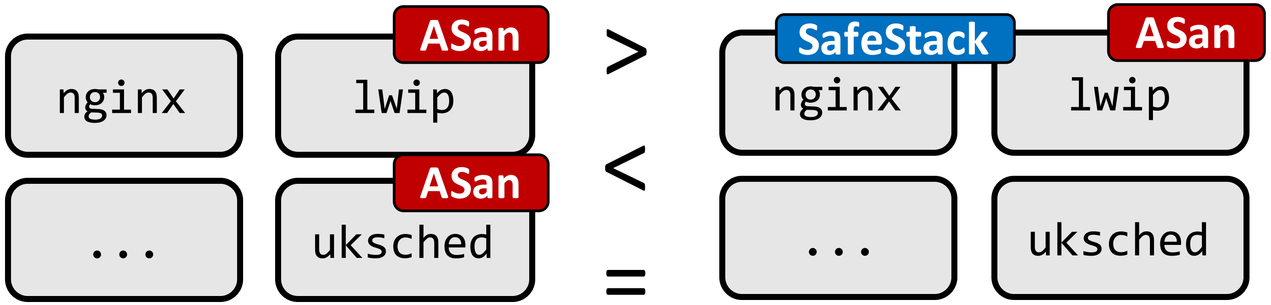
Problem: some configurations are not comparable

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



nginx

ASan lwip

...

ASan uksched

>

<

=

SafeStack nginx

ASan lwip

...

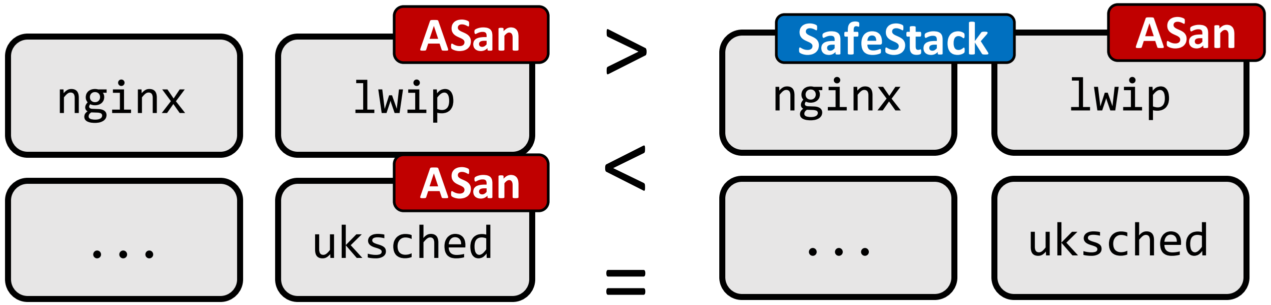
uksched

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



nginx

ASan lwip

...

ASan uksched

>
<
=

SafeStack nginx

ASan lwip

...

uksched

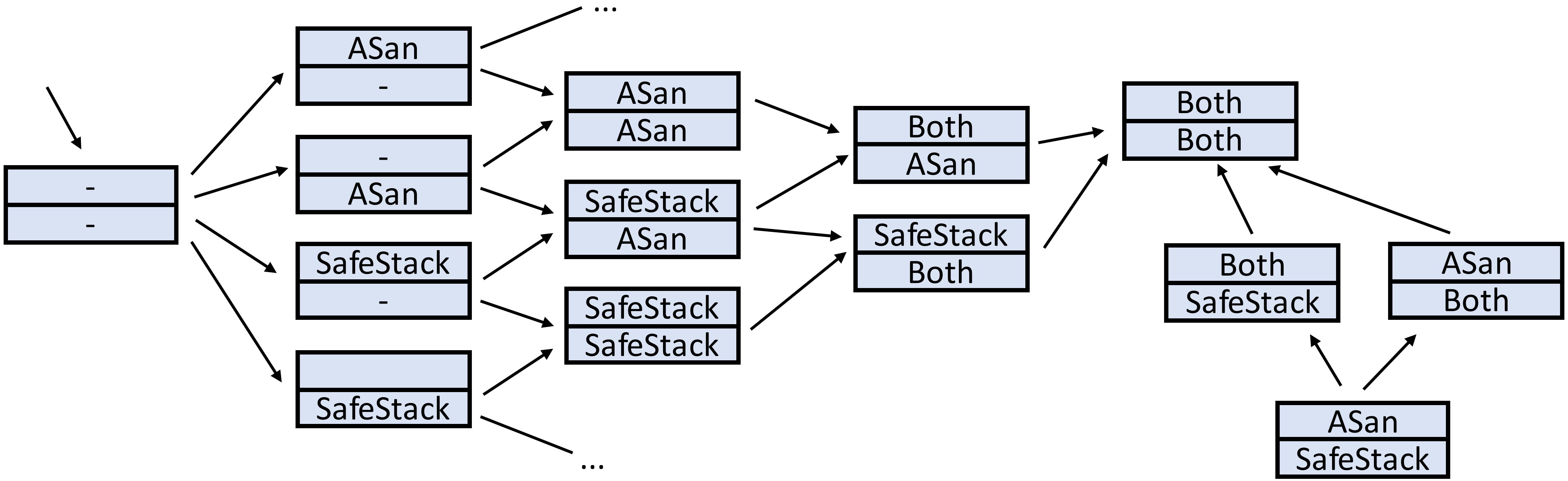
How can we reason about security/performance trade-offs?

Exploring the Design Space

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

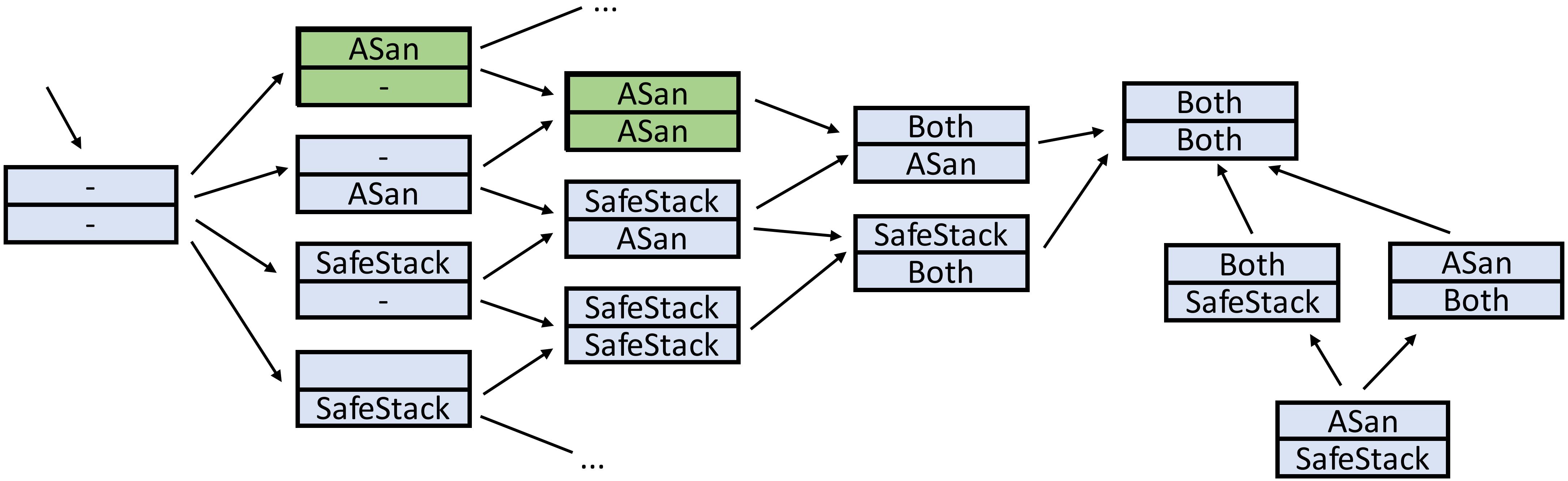
Exploring the Design Space

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



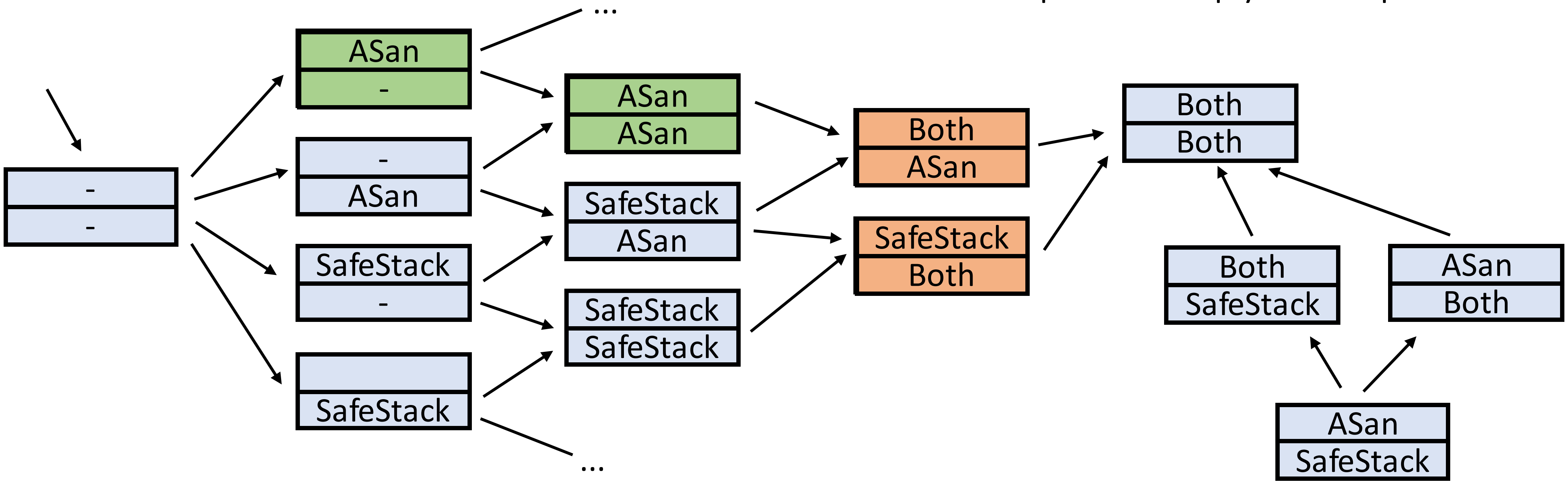
Exploring the Design Space

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



Exploring the Design Space

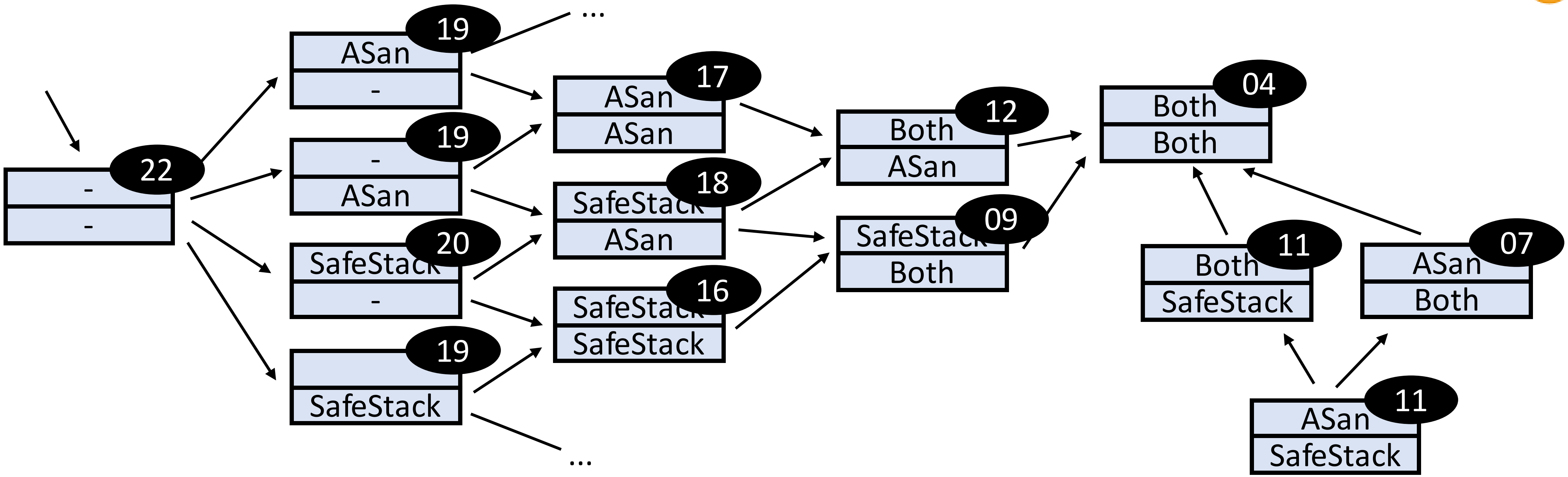
What we propose: consider configurations as a partially ordered set (poset)



Exploring the Design Space

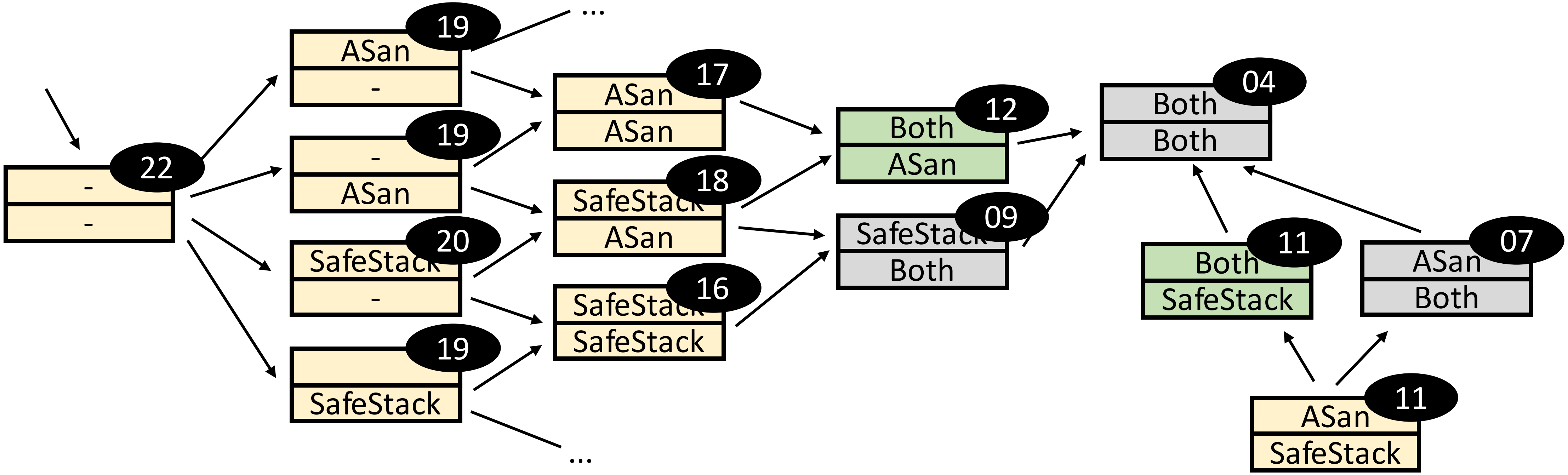
We can then label each node with performance characteristics (in practice no need to label everything)

Fictive numbers here 🙂



Exploring the Design Space

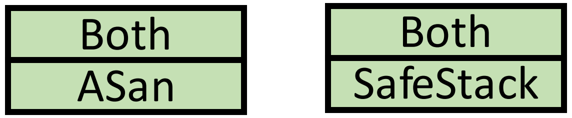
Based on this ordering and labeling we can choose the last node of each path that satisfies the performance constraints



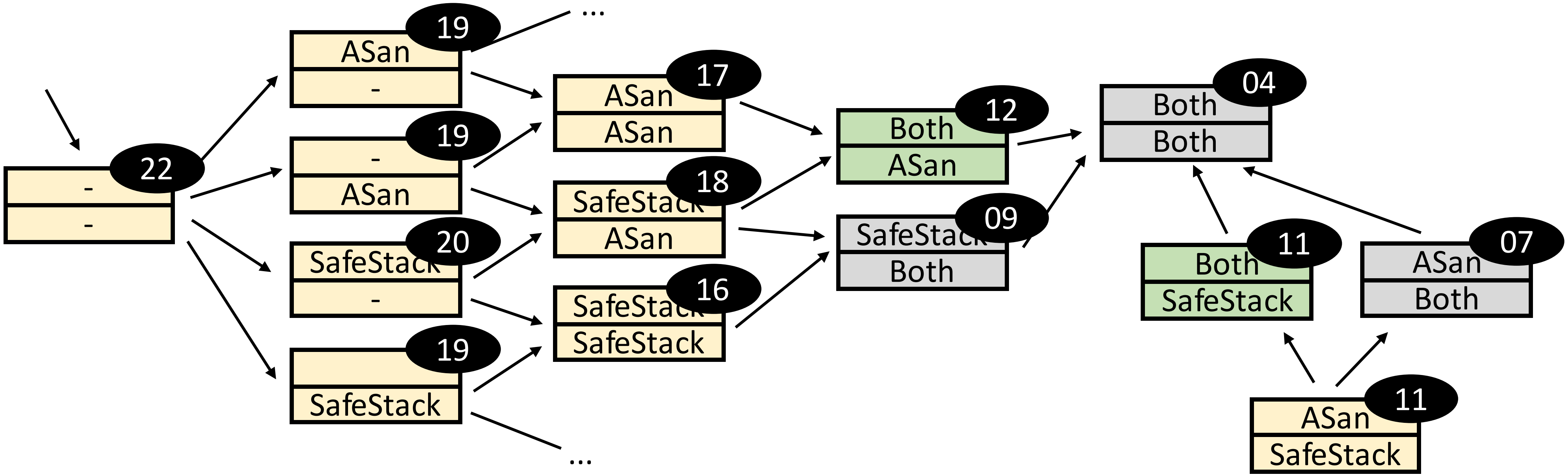
Exploring the Design Space

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



Curated list of optimal configurations



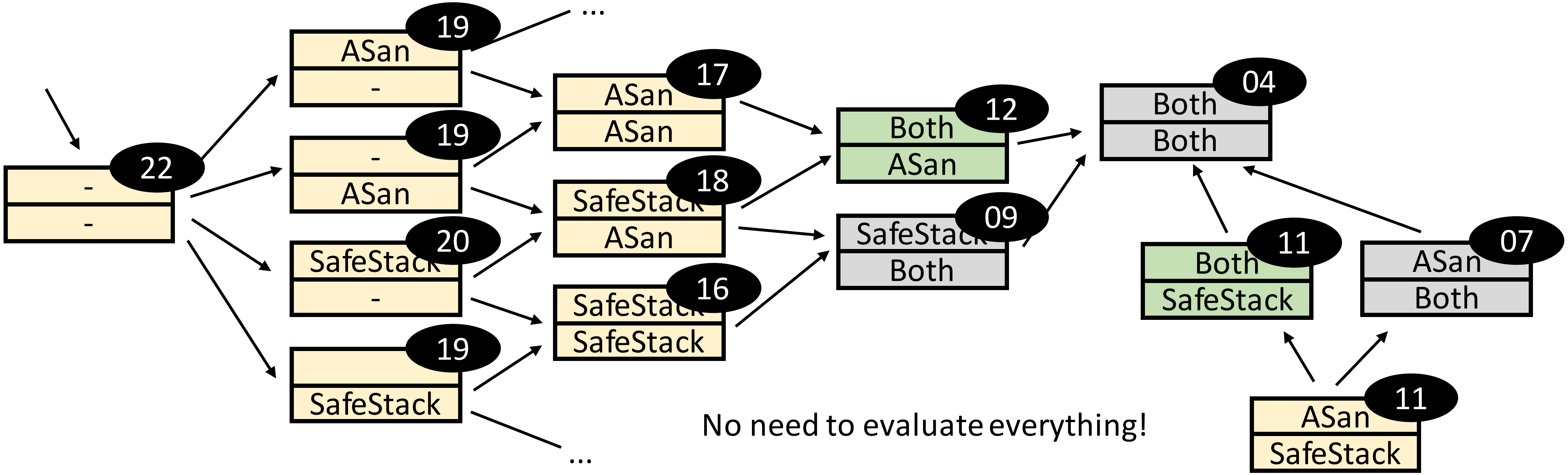
Exploring the Design Space

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

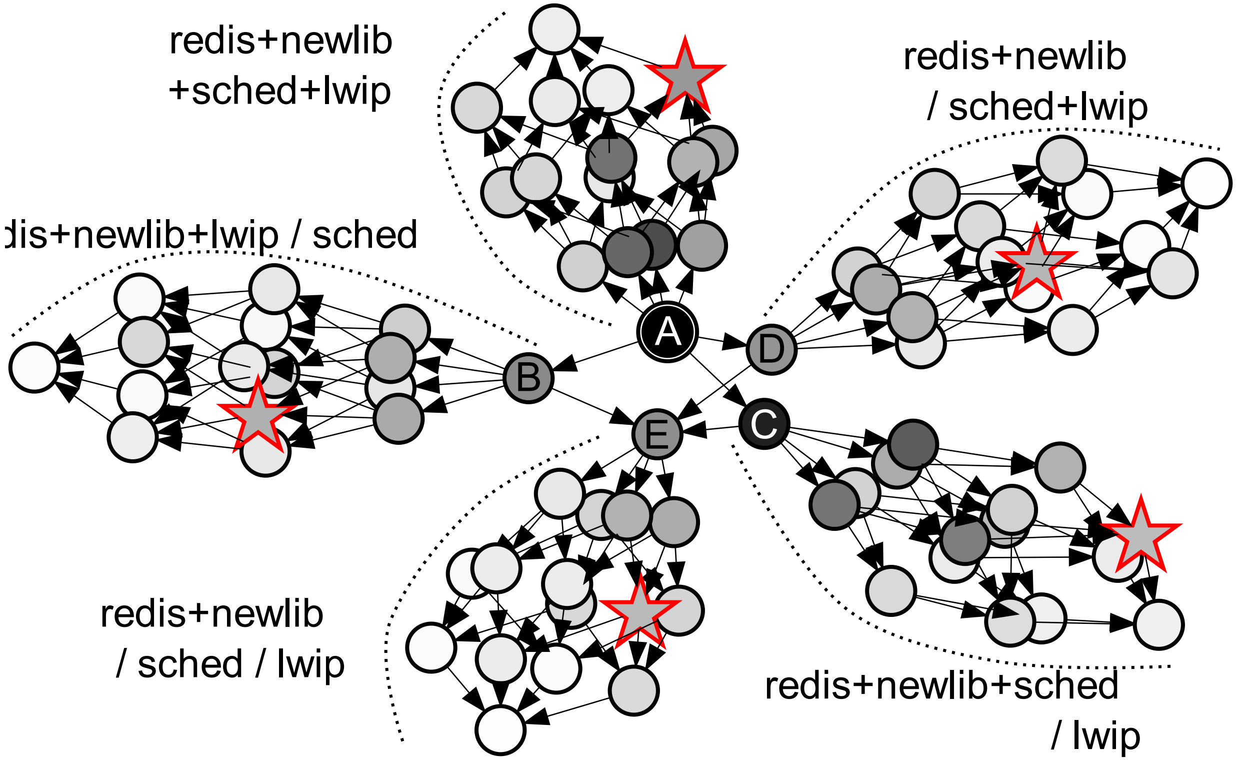
Both	Both
ASan	SafeStack

Curated list of optimal configurations



No need to evaluate everything!

Applying POSets to Redis



Reduction of 80 configurations to 5 candidates

Research Since ASPLOS'22

Research Since ASPLOS'22

- Applications of **FlexOS on CHERI** - *PLOS'23*

arXiv:2309.11332v2 [cs.CR] 21 Sep 2023

Software Compartmentalization Trade-Offs with Hardware Capabilities

John Alistair Kressel, Hugo Lefeuvre, Pierre Olivier
The University of Manchester
Manchester, UK

Abstract

Compartmentalization is a form of defensive software design in which an application is broken down into isolated but communicating components. Retrofitting compartmentalization into existing applications is often thought to be expensive from the engineering effort and performance overhead points of view. Still, recent years have seen proposals of compartmentalization methods with promises of low engineering efforts and reduced performance impact. ARM Morello combines a modern ARM processor with an implementation of Capability Hardware Enhanced RISC Instructions (CHERI) aiming to provide efficient and secure compartmentalization. Past works exploring CHERI-based compartmentalization were restricted to emulated/FPGA prototypes.

In this paper, we explore possible compartmentalization schemes with CHERI on the Morello chip. We propose two approaches representing different trade-offs in terms of engineering effort, security, scalability, and performance impact. We describe and implement these approaches on a prototype OS running bare metal on the Morello chip, compartmentalize two popular applications, and investigate the performance overheads. Furthermore, we show that compartmentalization can be achieved with an engineering cost that can be quite low if one is willing to trade off on scalability and security, and that performance overheads are similar to other intra-address space isolation mechanisms.

CCS Concepts: • **Security and privacy → Security in hardware; Operating systems security; Software and application security.**

Keywords: Compartmentalization, Hardware Capabilities

ACM Reference Format:
John Alistair Kressel, Hugo Lefeuvre, Pierre Olivier. 2023. Software Compartmentalization Trade-Offs with Hardware Capabilities. In *12th Workshop on Programming Languages and Operating Systems (PLOS '23), October 23, 2023, Koblenz, Germany.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3623759.3624550

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
PLOS '23, October 23, 2023, Koblenz, Germany
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0404-8/23/10.
https://doi.org/10.1145/3623759.3624550

1 Introduction

Software compartmentalization is one of the ways to enforce the principle of least privilege [31]. Compartmentalization enforces isolation between components of a software system, granting compartments only the minimal privileges they need to function. If a component of a compartmentalized system is subverted, the damage the attacker can do is limited to the privileges granted to the compromised compartment [17, 30]. Contrary to many other protection techniques, compartmentalization allows defending against yet unknown/future vulnerabilities in existing code bases [41]. Many approaches have been proposed in recent years, utilizing different hardware and software isolation mechanisms to compartmentalize libraries [5, 6, 15, 22, 25, 28, 32–34, 37, 44] as well as smaller pieces of code such as functions [3, 14, 36, 38].

Morello [26] is an extension to the ARMv8-A architecture implementing the Capability Hardware Enhanced RISC Instructions (CHERI), designed specifically to enable high-performance and scalable compartmentalization [48, 41, 45]. This is achieved by enforcing compartment bounds on most memory loads and stores in hardware, and letting communicating compartments securely lend memory to each other using so-called hardware capabilities, a mechanism similar to fat pointers [20] implemented in hardware to restrict accesses to shared memory at a fine (byte-level) granularity.

When retrofitting compartmentalization to existing code bases, a key challenge is keeping refactoring costs low [4]. This is crucial not only for reducing the cost of deployment, but also to reduce the number of errors made during the compartmentalization, which can undermine its efficiency in security guarantees [21]. Work exploring compartmentalization with CHERI is so far limited to solutions with relatively high porting costs [11, 32, 43], that require a non-negligible reworking to the code corresponding to inter-compartment communications. These existing works are further limited to MIPS/RISC-V emulated or FPGA prototypes, making it hard to understand the real-world performance one would observe on an ASIC processor. In that context, the recent availability of Morello raises the research questions we tackle in this paper:

1. Which compartment models are possible using Morello, using what programming abstractions, at which refactoring costs?
2. How does Morello's compartmentalization performance and security guarantees compare to other intra-address space compartmentalization mechanisms (e.g., MPK)?

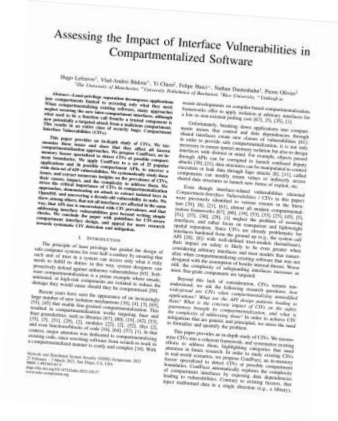
Research Since ASPLOS'22

- Applications of **FlexOS on CHERI** - *PLOS'23*
- **Interface safety** ("Compartment Interface Vulnerabilities") - *NDSS'23*

Software Compartmentalization Trade-Offs with Hardware Capabilities

John Alistair Kressel, Hugo Lefeuvre, Pierre Olivier
The University of Manchester
Manchester, UK

arXiv:2309.11332v2 [cs.CR] 21 Sep 2023



Assessing the Impact of Interface Vulnerabilities in Compartmentalized Software

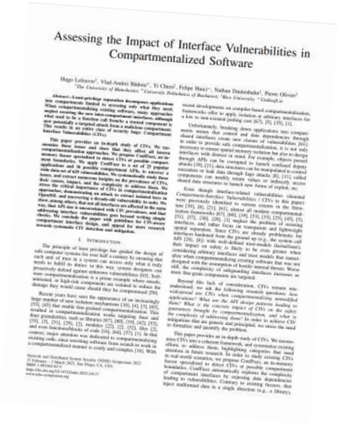
Hugo Lefeuvre, Vlad-Andrei Bădoiu, Yi Chen, Felipe Huici, Nathan Dautenhahn, Pierre Olivier

Research Since ASPLOS'22

- Applications of **FlexOS on CHERI** - *PLOS'23*
- **Interface safety** ("Compartment Interface Vulnerabilities") - *NDSS'23*
- ...and avenues to **achieve interface safety** - *HotOS'23*

Software Compartmentalization Trade-Offs with Hardware Capabilities

John Alistair Kressel, Hugo Lefeuvre, Pierre Olivier
The University of Manchester
Manchester, UK



Assessing the Impact of Interface Vulnerabilities in Compartmentalized Software

Towards (Really) Safe and Fast Confidential I/O

Hugo Lefeuvre, David Chisnall, Marios Kogias, Pierre Olivier

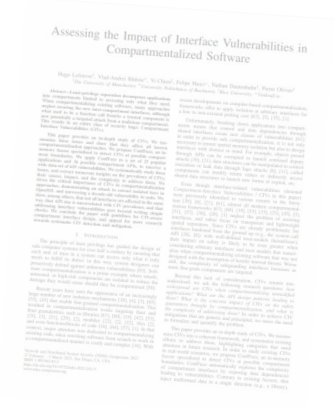
Research Since ASPLOS'22

- Applications of **FlexOS on CHERI** - *PLOS'23*
- **Interface safety** ("Compartment Interface Vulnerabilities") - *NDSS'23*
- …and avenues to **achieve interface safety** - *HotOS'23*

Software Compartmentalization Trade-Offs with Hardware Capabilities

John Alastair Kressel, Hugo Lefeuvre, Pierre Olivier
The University of Manchester
Manchester, UK

arXiv:2309.11332v2 [cs.CR] 21 Sep 2023



Assessing the Impact of Interface Vulnerabilities in Compartmentalized Software

Towards (Really) Safe and Fast Confidential I/O

Hugo Lefeuvre, David Chisnall, Marios Kogias, Pierre Olivier
The University of Manchester, Azure Research Microsoft, Imperial College London

To be presented at **PLOS'23**

flexOS + arm Morello Program

Software Compartmentalization Trade-Offs with Hardware Capabilities

John Alistair Kressel, **Hugo Lefeuvre**, Pierre Olivier

The University of Manchester

MANCHESTER 1824
The University of Manchester

FlexOS on CHERI Morello

- Design of a **CHERI backend** for FlexOS on the **ARM Morello SoC**

FlexOS on CHERI Morello

- Design of a **CHERI backend** for FlexOS on the **ARM Morello SoC**

- Motivating bits:
  - Explore hybrid compartmentalization models on CHERI

FlexOS on CHERI Morello

- Design of a **CHERI backend** for FlexOS on the **ARM Morello SoC**
- Motivating bits:
  - Explore hybrid compartmentalization models on CHERI
  - Further validate the genericity of the FlexOS design

FlexOS on CHERI Morello

- Design of a **CHERI backend** for FlexOS on the **ARM Morello SoC**

- Motivating bits:
  - Explore hybrid compartmentalization models on CHERI
  - Further validate the genericity of the FlexOS design
  - Compare (head-to-head) CHERI with EPT, MPK, etc.

FlexOS on CHERI Morello

- Design of a **CHERI backend** for FlexOS on the **ARM Morello SoC**

- Motivating bits:
  - Explore hybrid compartmentalization models on CHERI
  - Further validate the genericity of the FlexOS design
  - Compare (head-to-head) CHERI with EPT, MPK, etc.

- This is an initial exploration

Morello Backend Design

- Explore two approaches:

Morello Backend Design

- Explore two approaches:

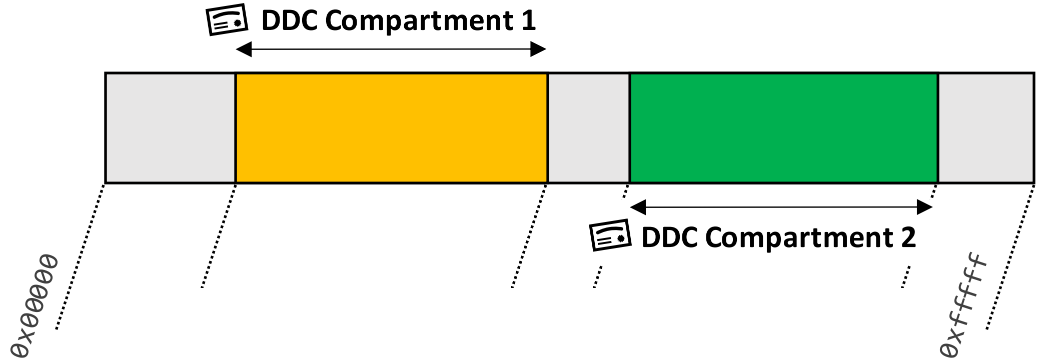
Approach 1: Use the *DDC* for isolation, exchange capabilities for sharing.

**DDC = Default Data Capability*

Morello Backend Design

- Explore two approaches:

Approach 1: Use the *DDC* for isolation, exchange capabilities for sharing.

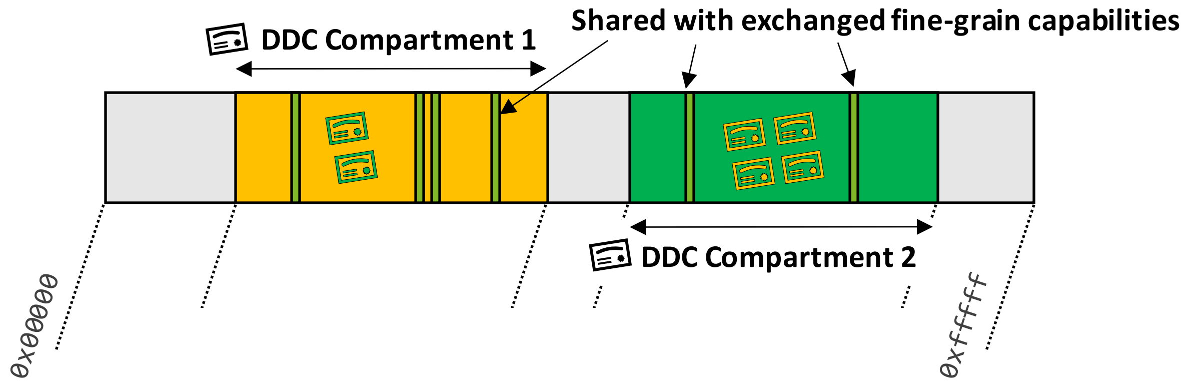


**DDC = Default Data Capability*

Morello Backend Design

- Explore two approaches:

Approach 1: Use the *DDC* for isolation, *exchange capabilities for sharing*.

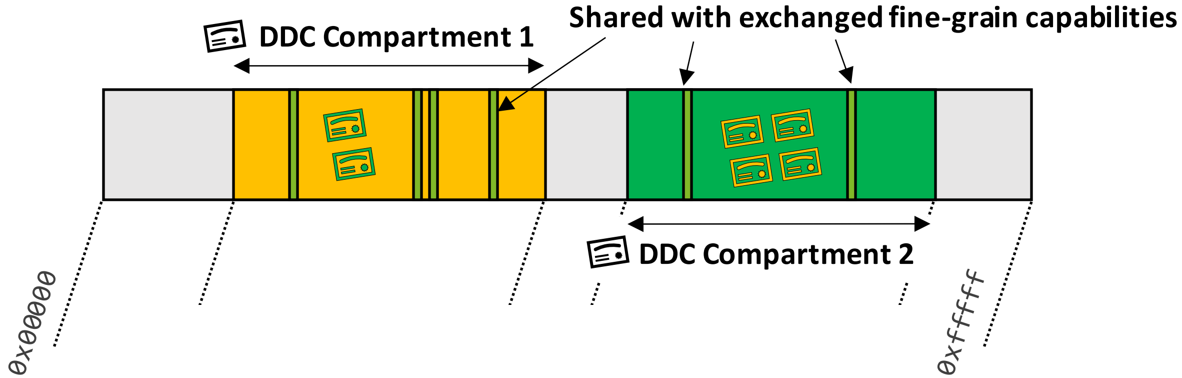


**DDC = Default Data Capability*

Morello Backend Design

- Explore two approaches:

Approach 1: Use the *DDC* for isolation, *exchange capabilities for sharing*.



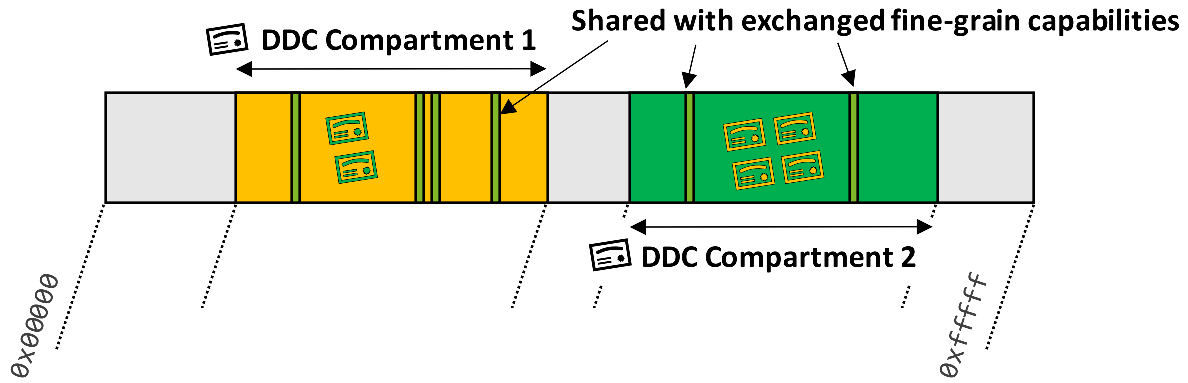
☑ Unlimited compartments
☒ Requires significant compiler engineering
☒ Performance: capabilities are not free

**DDC = Default Data Capability*

Morello Backend Design

- Explore two approaches:

Approach 1: Use the *DDC* for isolation, *exchange capabilities for sharing*.



☑ Unlimited compartments
☒ Requires significant compiler engineering
☒ Performance: capabilities are not free

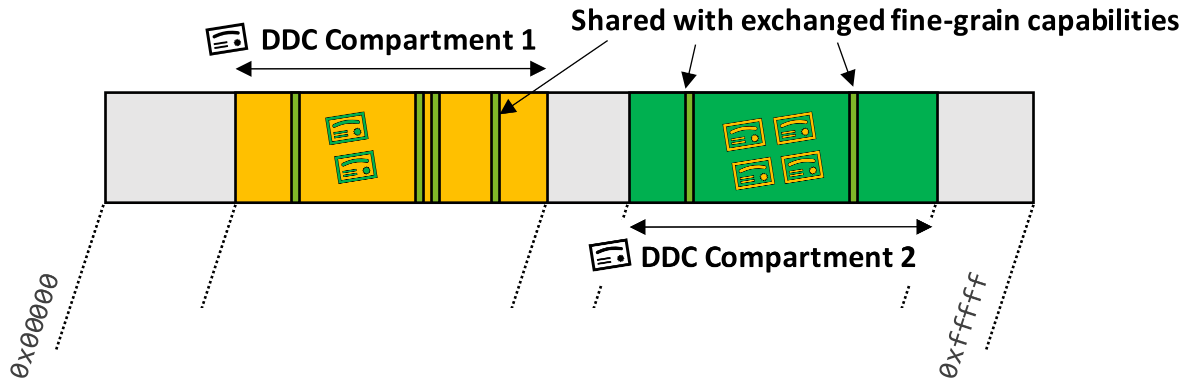
Approach 2: Use exclusively the *DDC* for isolation *and* sharing.

*DDC = Default Data Capability

Morello Backend Design

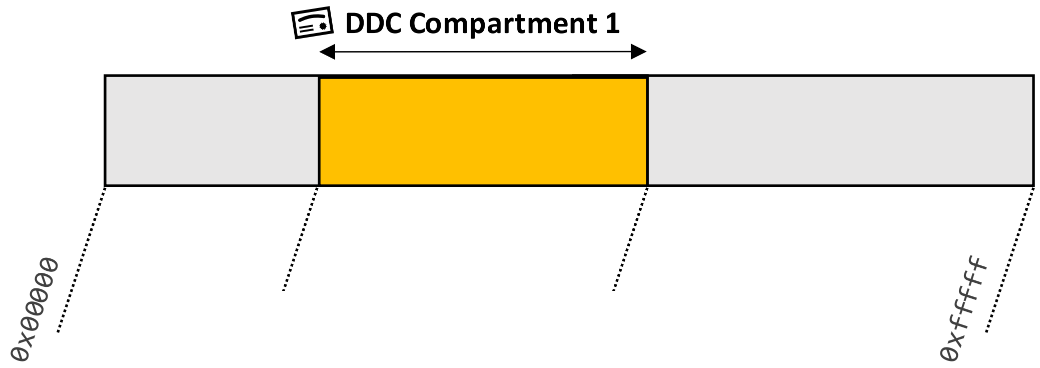
- Explore two approaches:

Approach 1: Use the *DDC* for isolation, *exchange capabilities for sharing*.



☑ Unlimited compartments
☒ Requires significant compiler engineering
☒ Performance: capabilities are not free

Approach 2: Use exclusively the *DDC* for isolation *and* sharing.

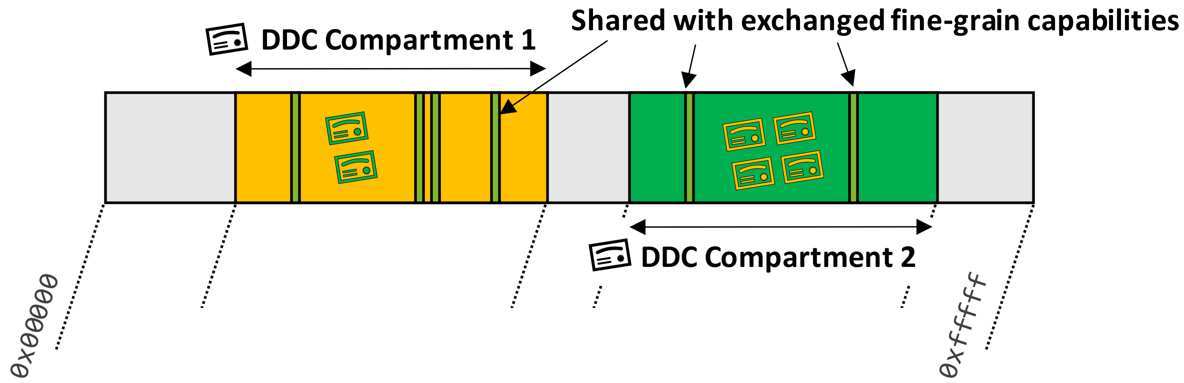


**DDC = Default Data Capability*

Morello Backend Design

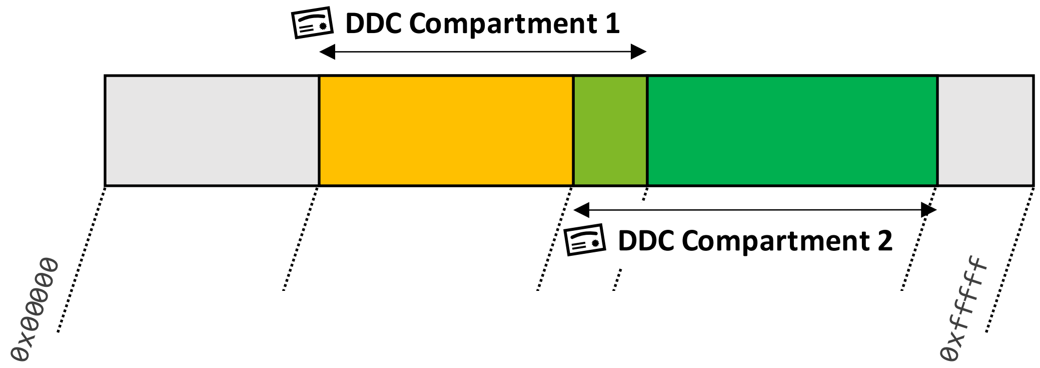
- Explore two approaches:

Approach 1: Use the *DDC* for isolation, *exchange capabilities for sharing*.



☑ Unlimited compartments
☒ Requires significant compiler engineering
☒ Performance: capabilities are not free

Approach 2: Use exclusively the *DDC* for isolation *and* sharing.

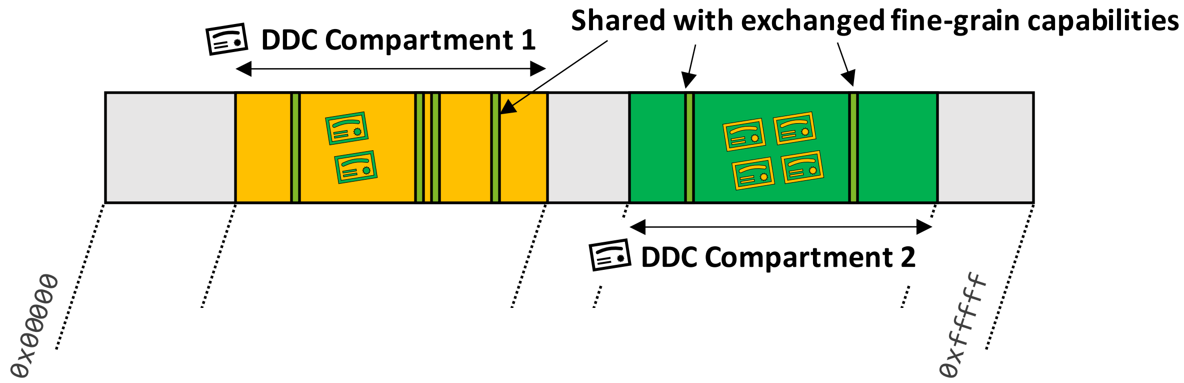


*DDC = Default Data Capability

Morello Backend Design

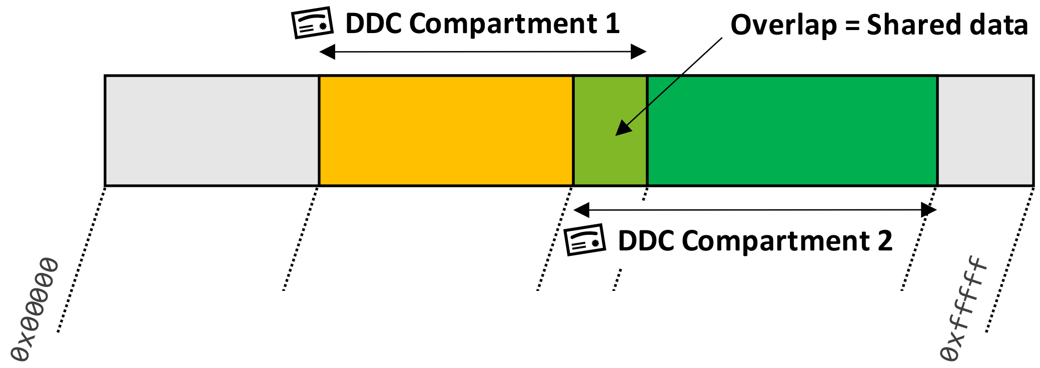
- Explore two approaches:

Approach 1: Use the *DDC* for isolation, *exchange capabilities for sharing*.



☑ Unlimited compartments
☒ Requires significant compiler engineering
☒ Performance: capabilities are not free

Approach 2: Use exclusively the *DDC* for isolation *and* sharing.



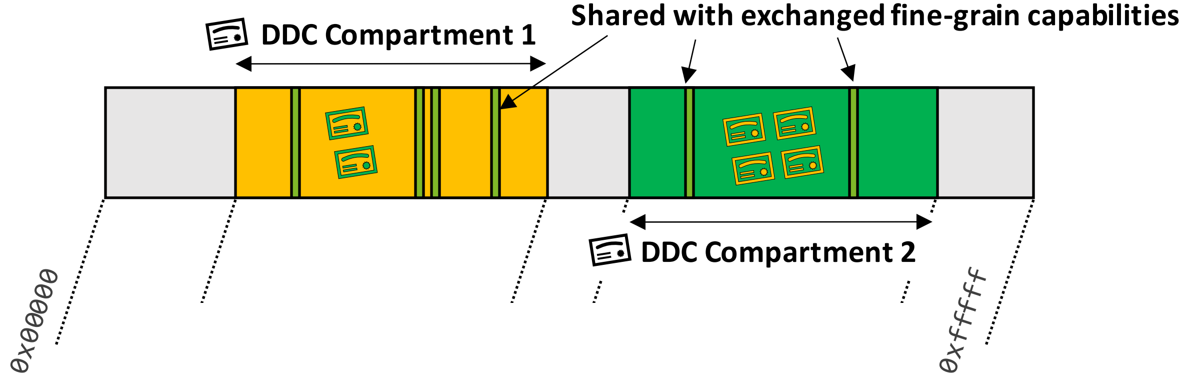
**DDC = Default Data Capability*

Morello Backend Design

Morello Program

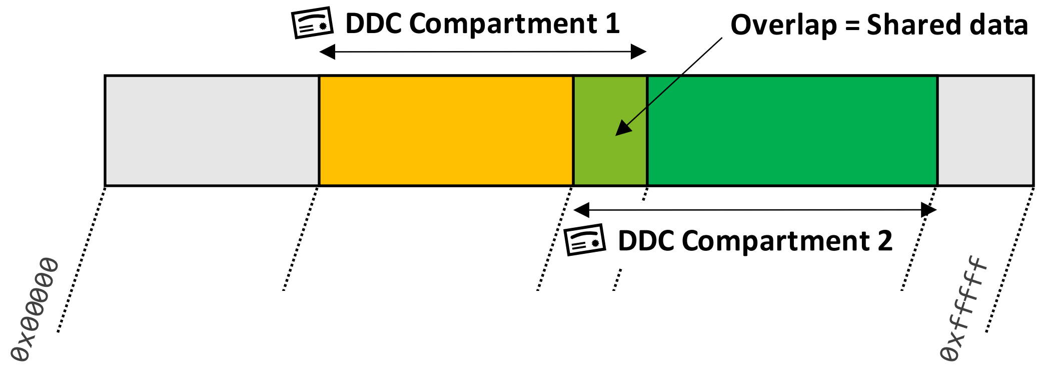
- Explore two approaches:

Approach 1: Use the *DDC* for isolation, *exchange capabilities for sharing*.



☑ Unlimited compartments
☒ Requires significant compiler engineering
☒ Performance: capabilities are not free

Approach 2: Use exclusively the *DDC* for isolation *and* sharing.



☑ Simple, fast
☒ Limits the number of compartments

**DDC = Default Data Capability*

Preliminary Results

Results on **Approach 2** (use DDC for isolation and sharing)

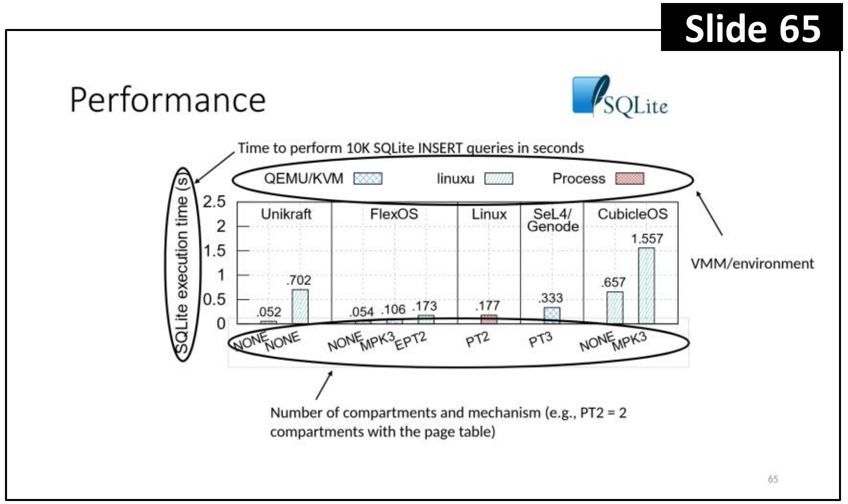
- How does our CHERI backend compare with MPK, EPT?

Preliminary Results

Results on **Approach 2** (use DDC for isolation and sharing)

- How does our CHERI backend compare with MPK, EPT?

Same SQLite experiment as ASPLOS'22, with our Morello backend



Slide 65

Performance

Time to perform 10K SQLite INSERT queries in seconds

VMM/environment

Number of compartments and mechanism (e.g., PT2 = 2 compartments with the page table)

Preliminary Results

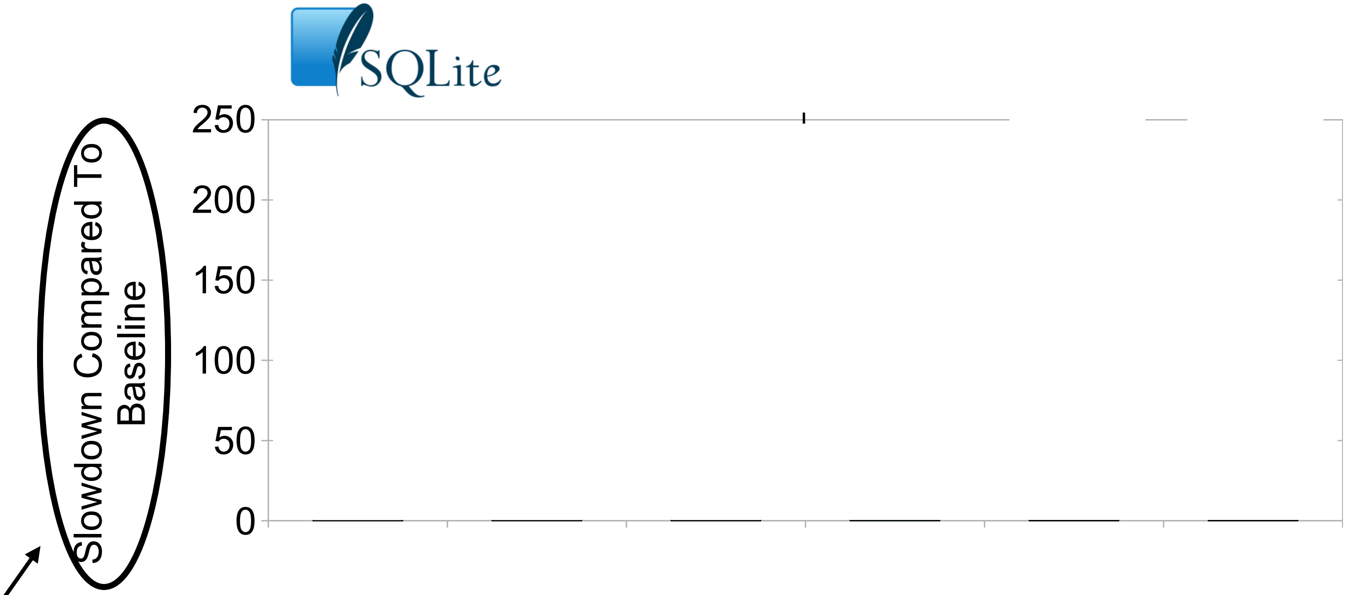
Results on **Approach 2** (use DDC for isolation and sharing)

- How does our CHERI backend compare with MPK, EPT?

Preliminary Results

Results on **Approach 2** (use DDC for isolation and sharing)

- How does our CHERI backend compare with MPK, EPT?



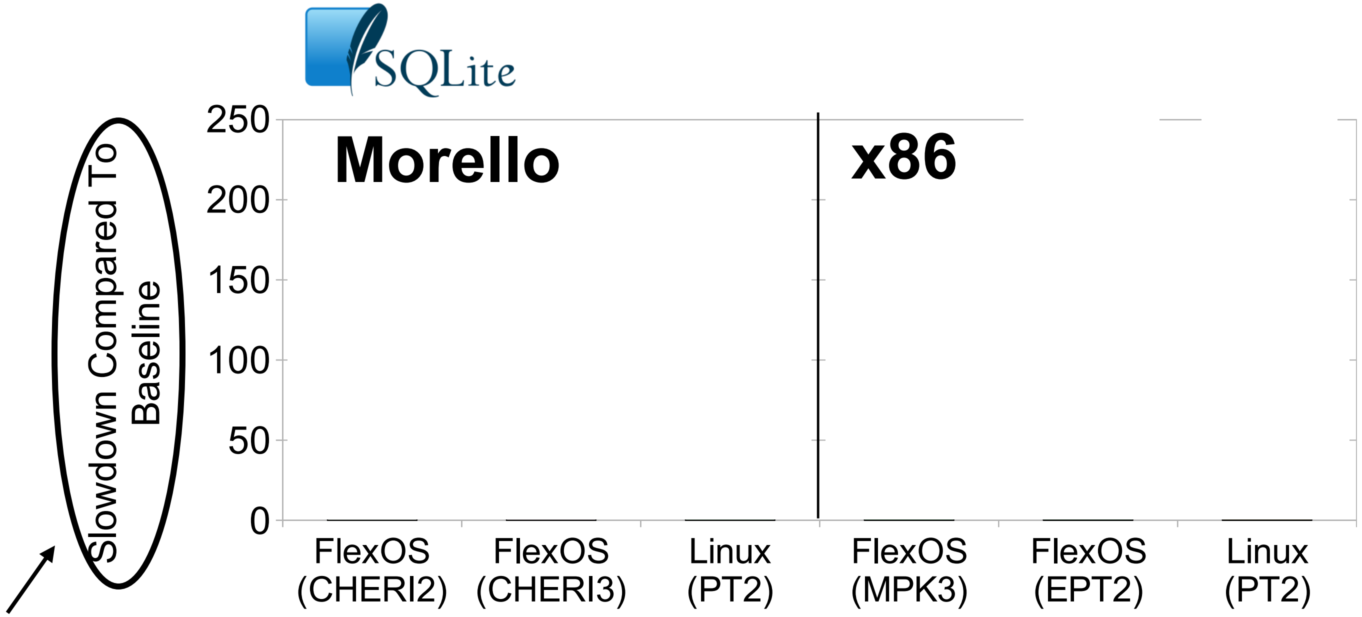
Overhead to perform 10K SQLite INSERT queries

Numbers are normalized: **uncompartmentalized baseline = 0**

Preliminary Results

Results on **Approach 2** (use DDC for isolation and sharing)

- How does our CHERI backend compare with MPK, EPT?



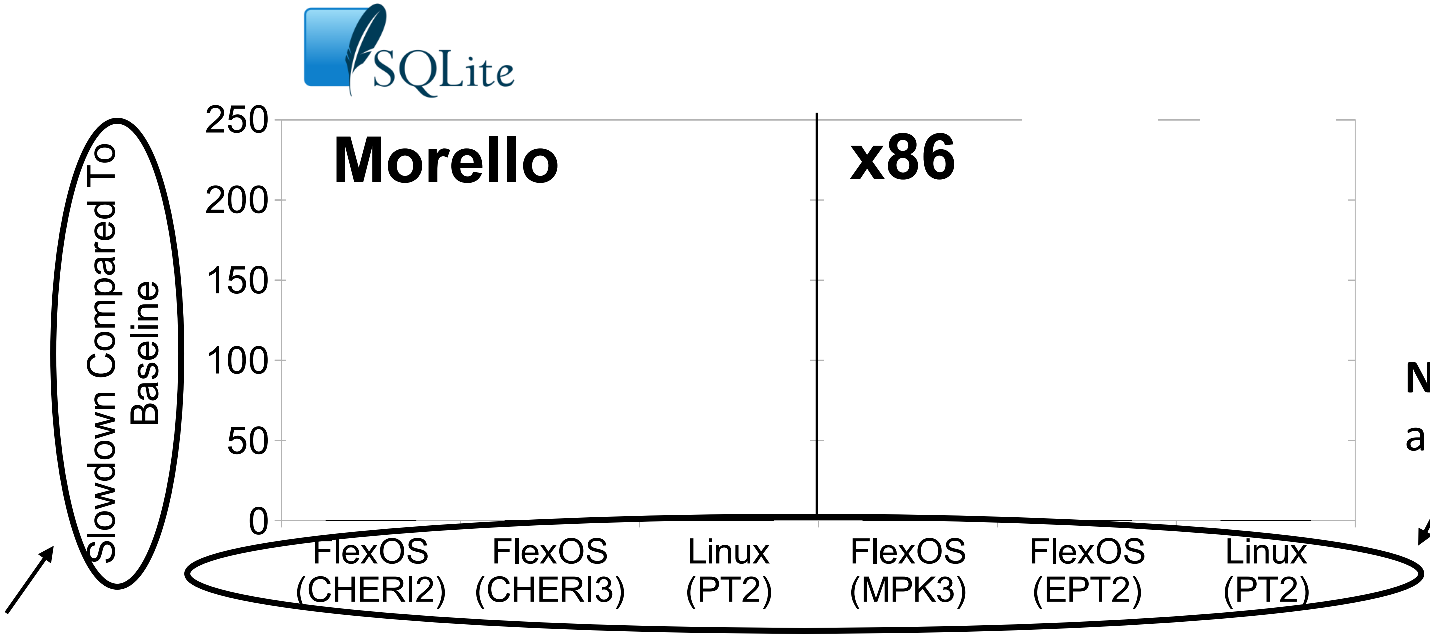
Overhead to perform 10K SQLite INSERT queries

Numbers are normalized: **uncompartmentalized baseline = 0**

Preliminary Results

Results on **Approach 2** (use DDC for isolation and sharing)

- How does our CHERI backend compare with MPK, EPT?



Number of compartments and **mechanism**

Overhead to perform 10K SQLite INSERT queries

Numbers are normalized: **uncompartmentalized baseline = 0**

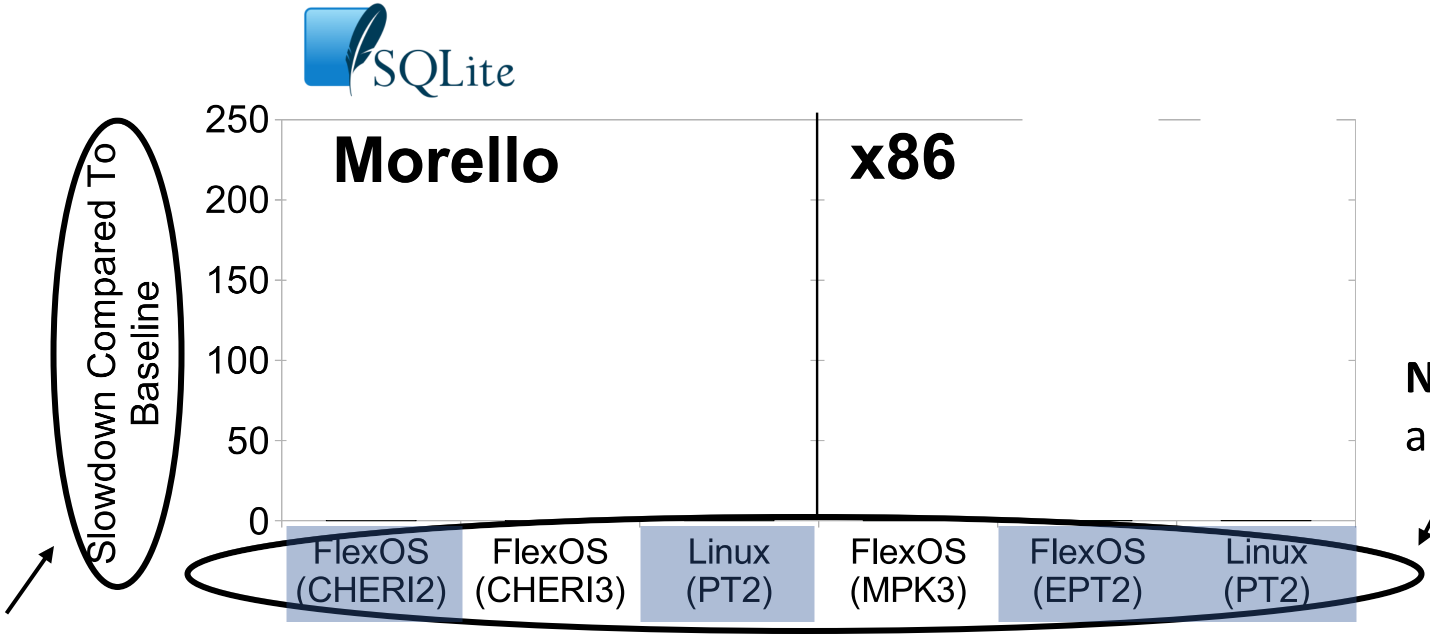
Preliminary Results

Results on **Approach 2** (use DDC for isolation and sharing)

- How does our CHERI backend compare with MPK, EPT?

Here:

- **2 compartments = SQLite/rest**



Number of compartments and **mechanism**

Overhead to perform 10K SQLite INSERT queries

Numbers are normalized: **uncompartmentalized baseline = 0**

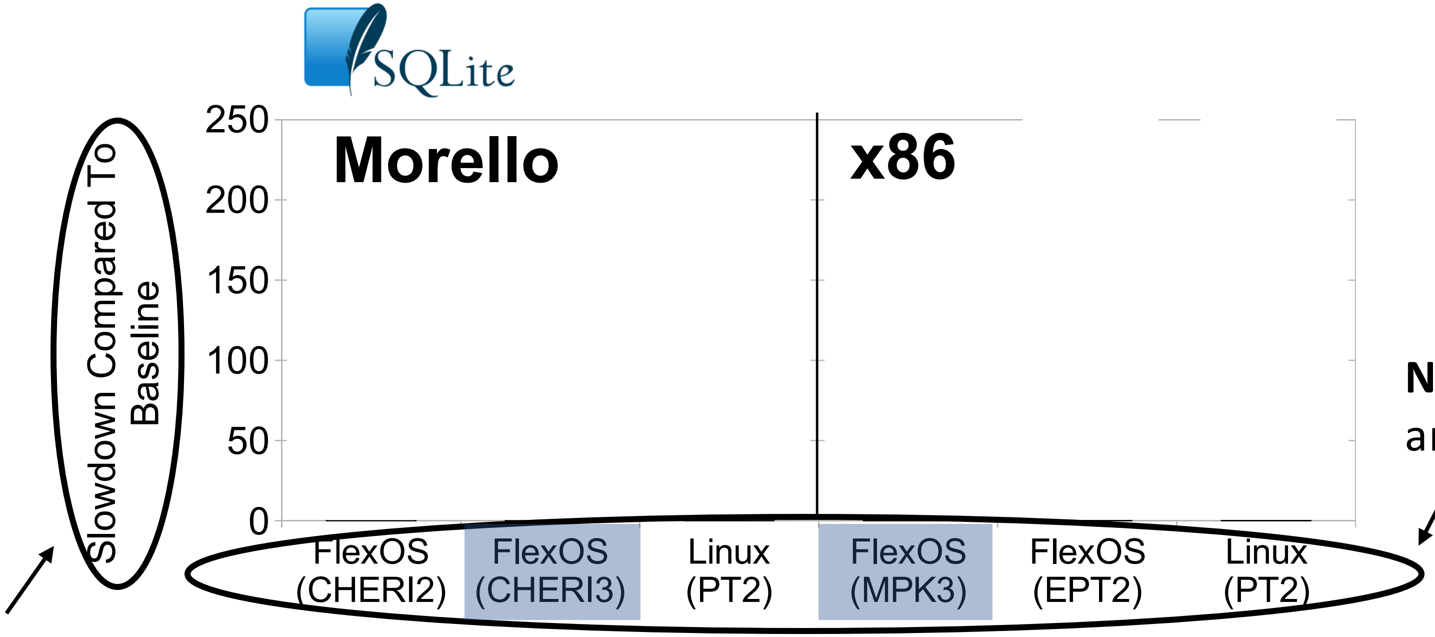
Preliminary Results

Results on **Approach 2** (use DDC for isolation and sharing)

- How does our CHERI backend compare with MPK, EPT?

Here:

- 2 compartments = SQLite/rest
- **3 compartments = SQLite/uktime/rest**



Number of compartments and **mechanism**

Overhead to perform 10K SQLite INSERT queries

Numbers are normalized: **uncompartmentalized baseline = 0**

Preliminary Results

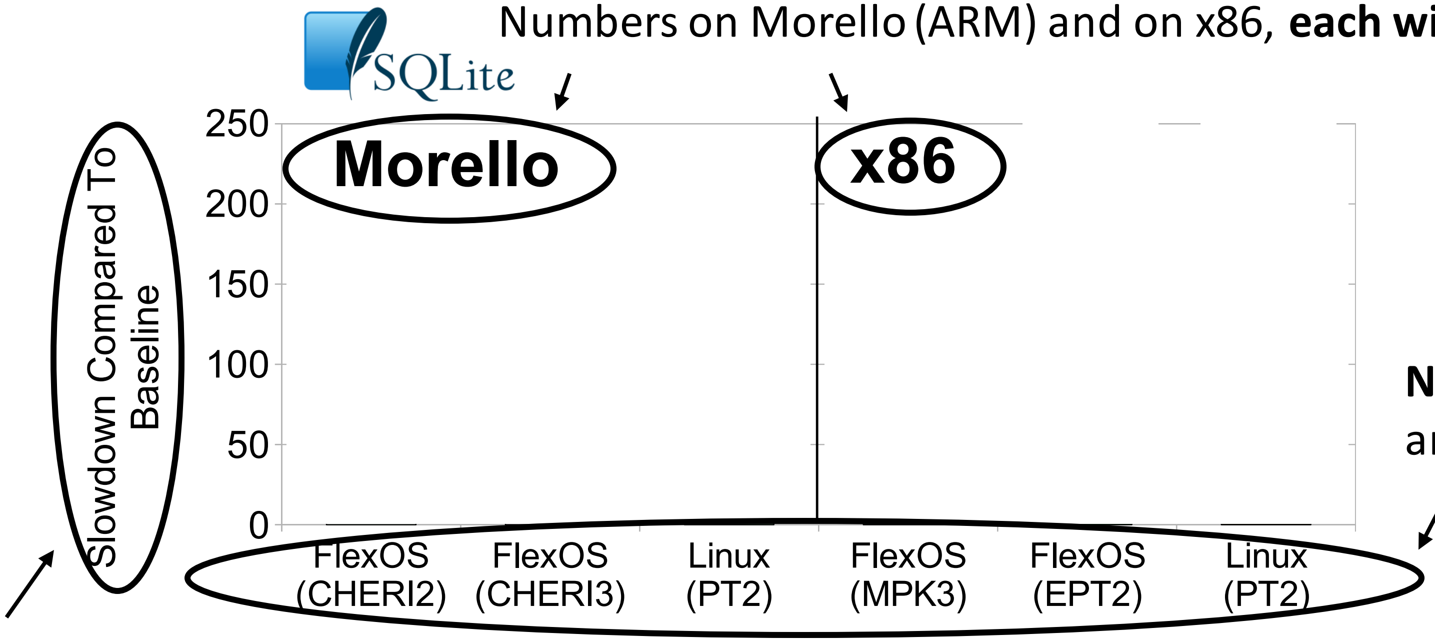
Results on **Approach 2** (use DDC for isolation and sharing)

- How does our CHERI backend compare with MPK, EPT?

Here:

- 2 compartments = SQLite/rest
- 3 compartments = SQLite/uktime/rest

Numbers on Morello (ARM) and on x86, **each with their own baseline**



Number of compartments and **mechanism**

Overhead to perform 10K SQLite INSERT queries in seconds

Numbers are normalized: **uncompartmentalized baseline = 0**

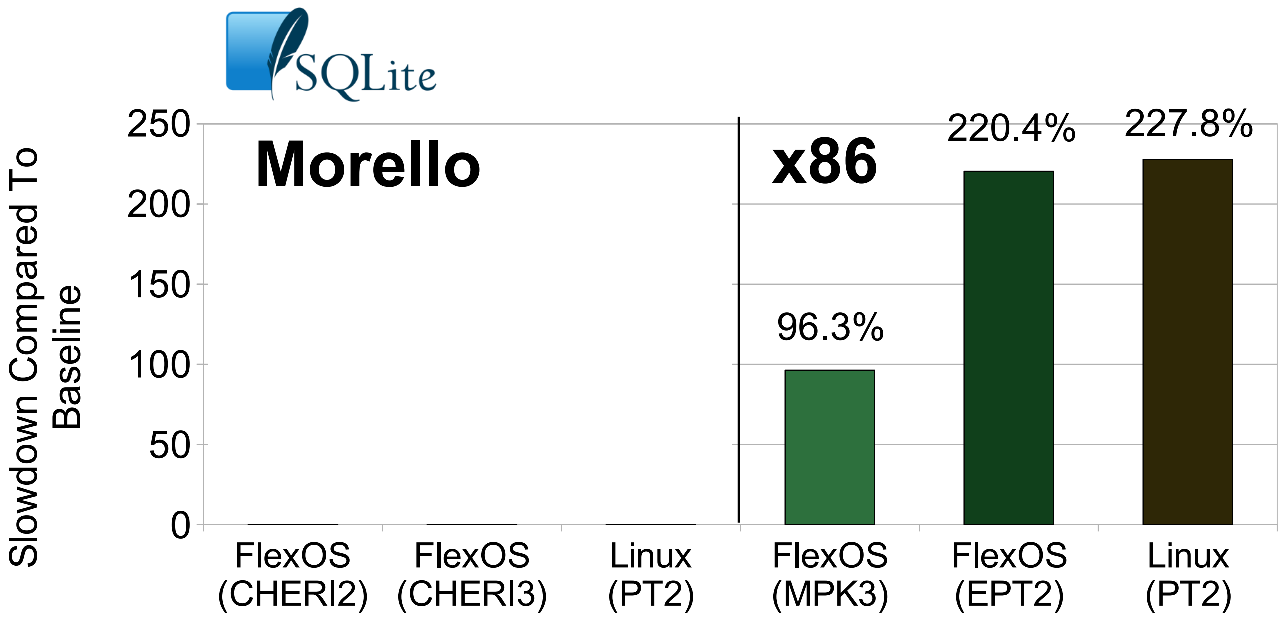
Preliminary Results

Results on **Approach 2** (use DDC for isolation and sharing)

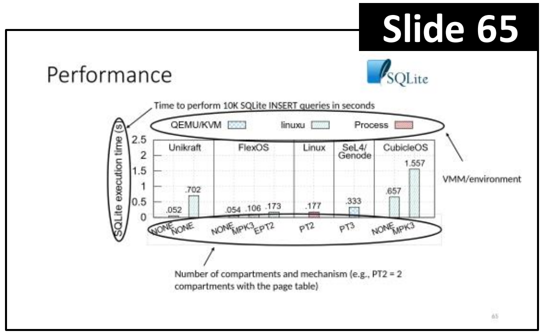
- How does our CHERI backend compare with MPK, EPT?

Here:

- 2 compartments = SQLite/rest
- 3 compartments = SQLite/uktime/rest



Slide 65


Performance

Time to perform 10K SQLite INSERT queries in seconds

VMM/environment

Number of compartments and mechanism (e.g., PT2 = 2 compartments with the page table)

The x86 numbers are from the FlexOS ASPLOS'22 paper

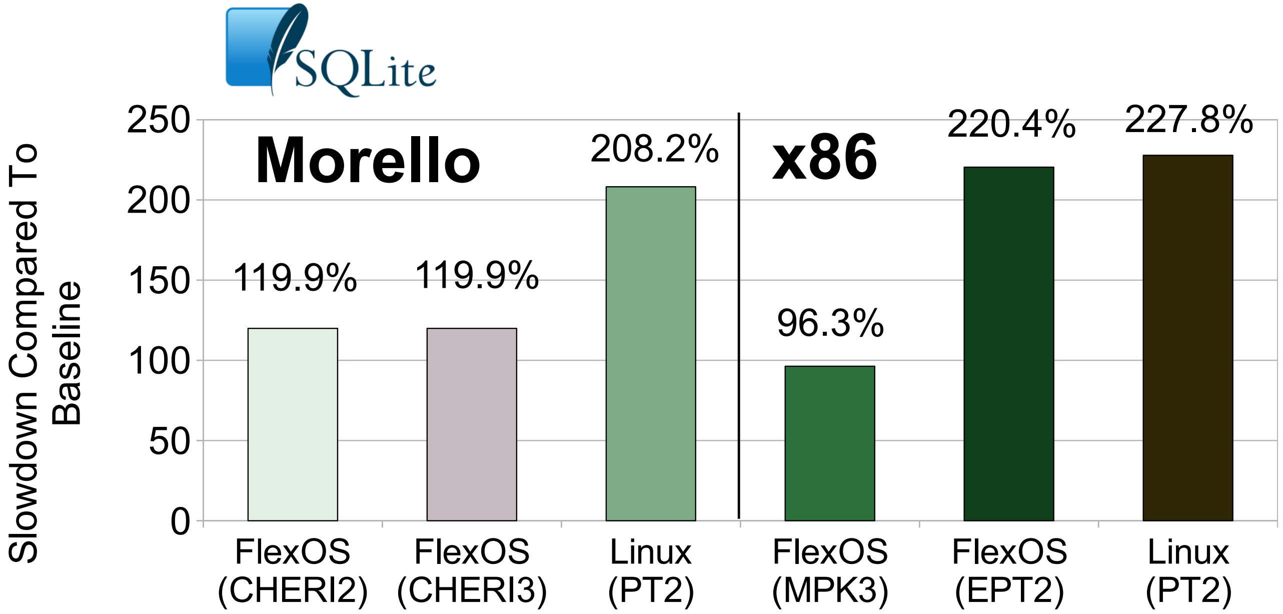
Preliminary Results

Results on **Approach 2** (use DDC for isolation and sharing)

- How does our CHERI backend compare with MPK, EPT?

Here:

- 2 compartments = SQLite/rest
- 3 compartments = SQLite/uktime/rest



Preliminary Results

flex OS

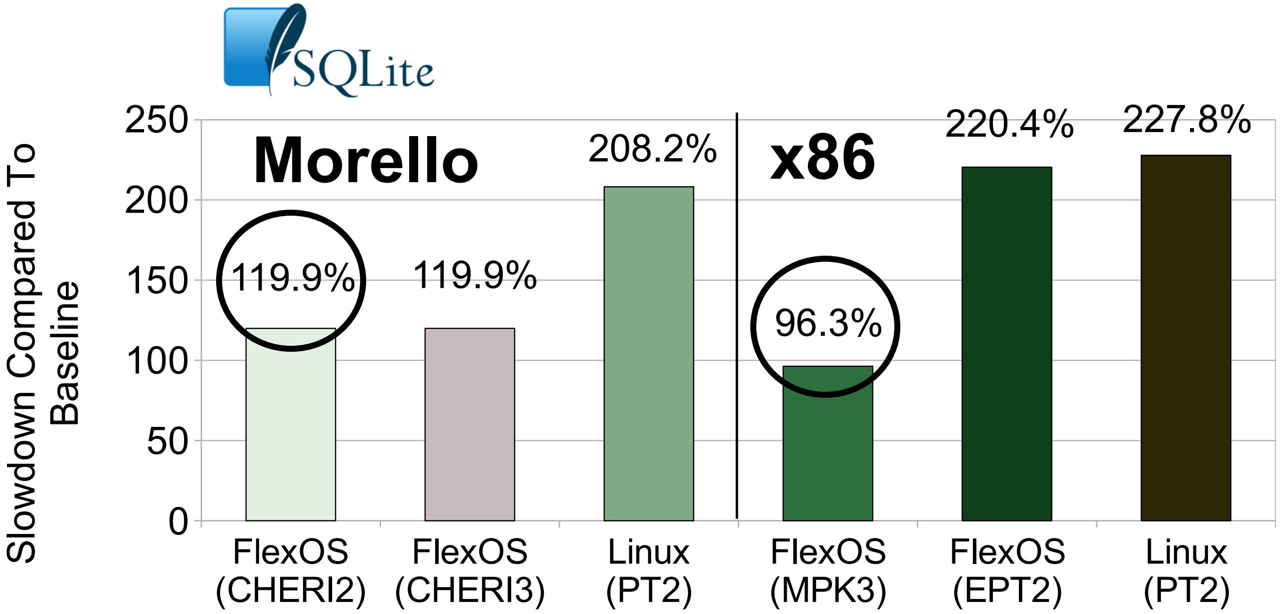
arm Morello Program

Results on **Approach 2** (use DDC for isolation and sharing)

- How does our CHERI backend compare with MPK, EPT?

Here:

- 2 compartments = SQLite/rest
- 3 compartments = SQLite/uktime/rest



① Looking at relative overheads, CHERI is 20-25% more expensive than MPK – that's pretty good!

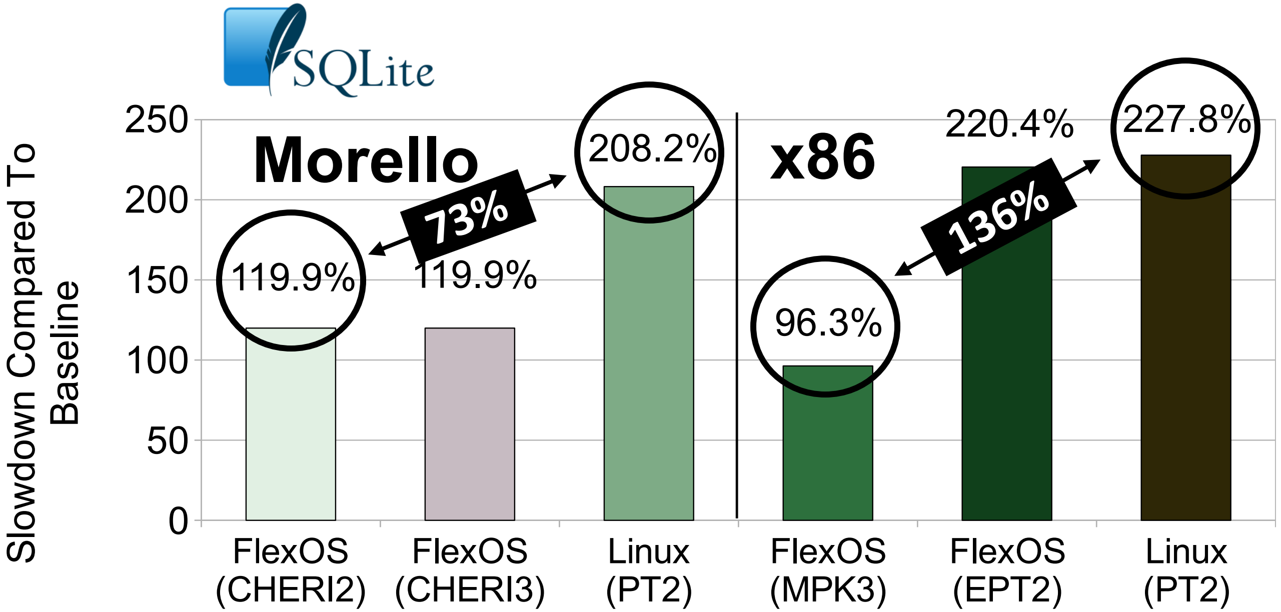
Preliminary Results

Results on **Approach 2** (use DDC for isolation and sharing)

- How does our CHERI backend compare with MPK, EPT?

Here:

- 2 compartments = SQLite/rest
- 3 compartments = SQLite/uktime/rest



② The PT is more expensive on x86 than on ARM, so taking that as a baseline would make CHERI look relatively more expensive

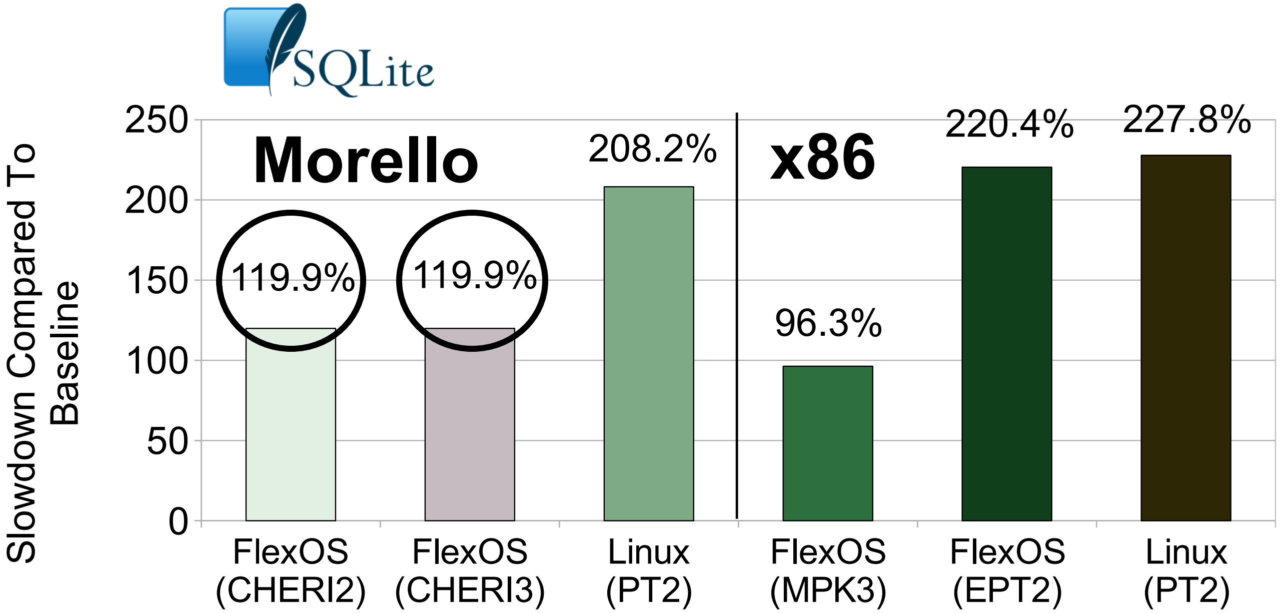
Preliminary Results

Results on **Approach 2** (use DDC for isolation and sharing)

- How does our CHERI backend compare with MPK, EPT?

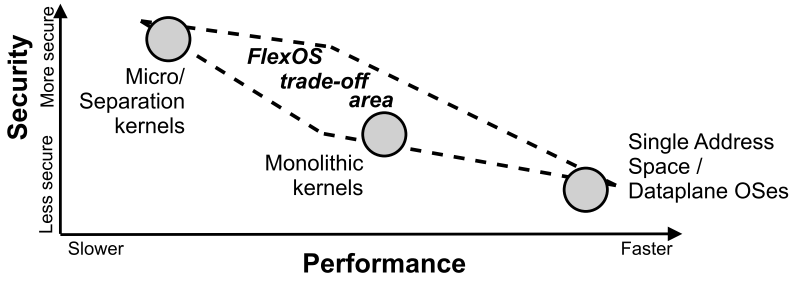
Here:

- 2 compartments = SQLite/rest
- 3 compartments = SQLite/**uktime**/rest



③ Same observation as earlier, some compartments are free because they are not on the critical path

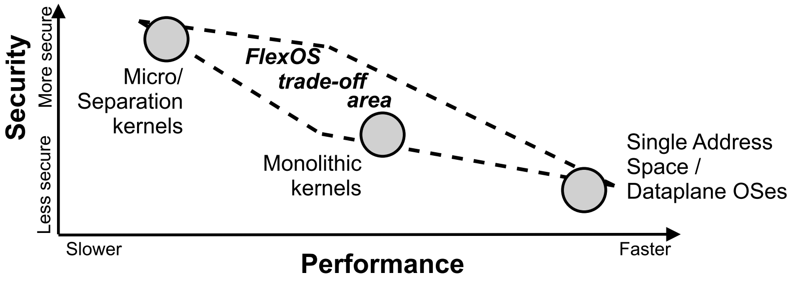
In a Nutshell



There is a **need for isolation flexibility**

- Specialization, hardware heterogeneity, etc.

In a Nutshell

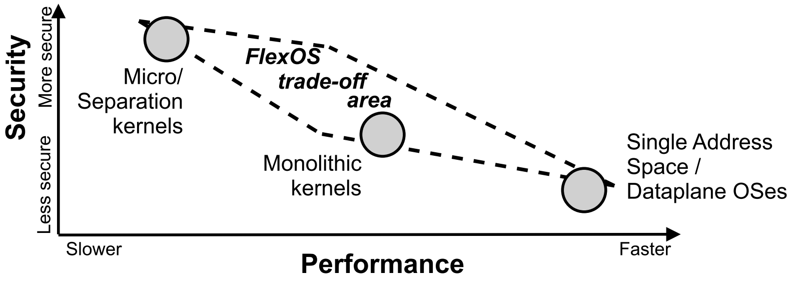


There is a **need for isolation flexibility**

- Specialization, hardware heterogeneity, etc.

State of the art: **one isolation approach at design time**

In a Nutshell



There is a **need for isolation flexibility**

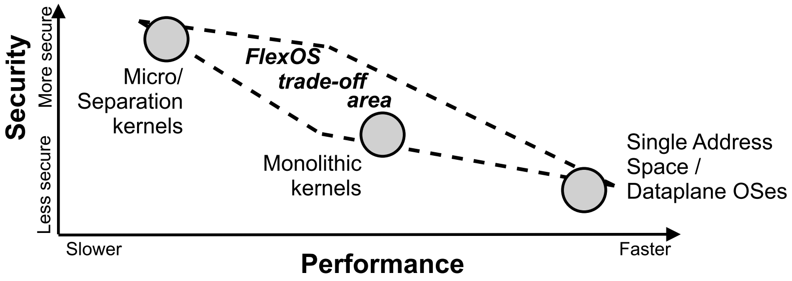
- Specialization, hardware heterogeneity, etc.

State of the art: **one isolation approach at design time**

Let's try to decouple isolation from the OS design?

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

In a Nutshell



There is a **need for isolation flexibility**

- Specialization, hardware heterogeneity, etc.

State of the art: **one isolation approach at design time**

Let's try to decouple isolation from the OS design?

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

Conclusion: We do get **very interesting trade-offs**

Opens for tons of interesting research

Interested?

Get in touch!

Webpage: https://project-flexos.github.io/
By e-mail: `hugo.lefeuvre@manchester.ac.uk`

License: 3-Clause BSD License 🙂