FlexOS: Illustrating the Research Potential of Unikraft

Hugo Lefeuvre (*The University of Manchester*)
hugo.lefeuvre@manchester.ac.uk

Unikraft Lyon Hackathon, 14th May 2022

FlexOS: Illustrating the Research Potential of Unikraft

Hugo Lefeuvre (*The University of Manchester*)
hugo.lefeuvre@manchester.ac.uk

Outline:

FlexOS: Illustrating the Research Potential of Unikraft

Hugo Lefeuvre (*The University of Manchester*)
hugo.lefeuvre@manchester.ac.uk

Outline:

1. High-Level Presentation of FlexOS

FlexOS: Illustrating the Research Potential of Unikraft

Hugo Lefeuvre (*The University of Manchester*)
hugo.lefeuvre@manchester.ac.uk

Outline:

1. High-Level Presentation of FlexOS
2. Technical Intro: Hello World!

FlexOS: Illustrating the Research Potential of Unikraft

Hugo Lefeuvre (*The University of Manchester*)
hugo.lefeuvre@manchester.ac.uk

Outline:

1. High-Level Presentation of FlexOS
2. Technical Intro: Hello World!
3. Technical Intro: Redis

FlexOS: Illustrating the Research Potential of Unikraft

Hugo Lefeuvre (*The University of Manchester*)
hugo.lefeuvre@manchester.ac.uk

Outline:

1. High-Level Presentation of FlexOS
2. Technical Intro: Hello World!
3. Technical Intro: Redis
4. Hands-On: Port Your Lib/App

FlexOS: Illustrating the Research Potential of Unikraft

Hugo Lefeuvre (*The University of Manchester*)
hugo.lefeuvre@manchester.ac.uk

Outline:

1. High-Level Presentation of FlexOS
2. Technical Intro: Hello World!
3. Technical Intro: Redis
4. Hands-On: Port Your Lib/App

Technical and hands-on!

These slides at: https://project-flexos.github.io/slides/flexos-lyon-tutorial.pdf

FlexOS: Illustrating the Research Potential of Unikraft

Hugo Lefeuvre (*The University of Manchester*)
hugo.lefeuvre@manchester.ac.uk

Outline:

1. High-Level Presentation of FlexOS
2. Technical Intro: Hello World!
3. Technical Intro: Redis
4. Hands-On: Port Your Lib/App

Technical and hands-on!

FlexOS: Illustrating the Research Potential of Unikraft

Hugo Lefeuvre (*The University of Manchester*)
hugo.lefeuvre@manchester.ac.uk

Outline:

1. **High-Level Presentation of FlexOS**
2. Technical Intro: Hello World!
3. Technical Intro: Redis
4. Hands-On: Port Your Lib/App

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]

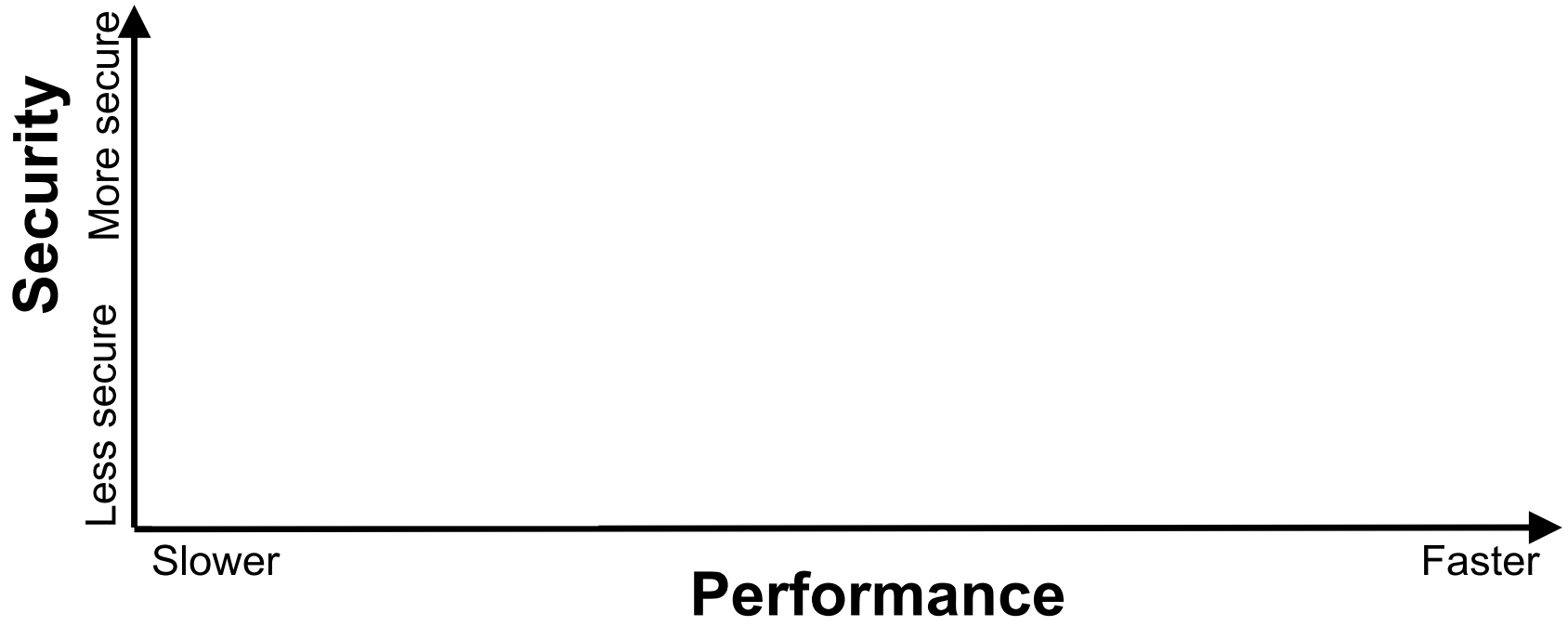
[1]The University of Manchester, [2]Politehnica Bucharest, [3]Lancaster University, [4]Unikraft.io, [5]Karlsruhe Institute of Technology, [6]NEC Labs Europe, [7]Correct Networks

Unikraft Lyon Hackathon, 14th May 2022

Current OS Designs

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

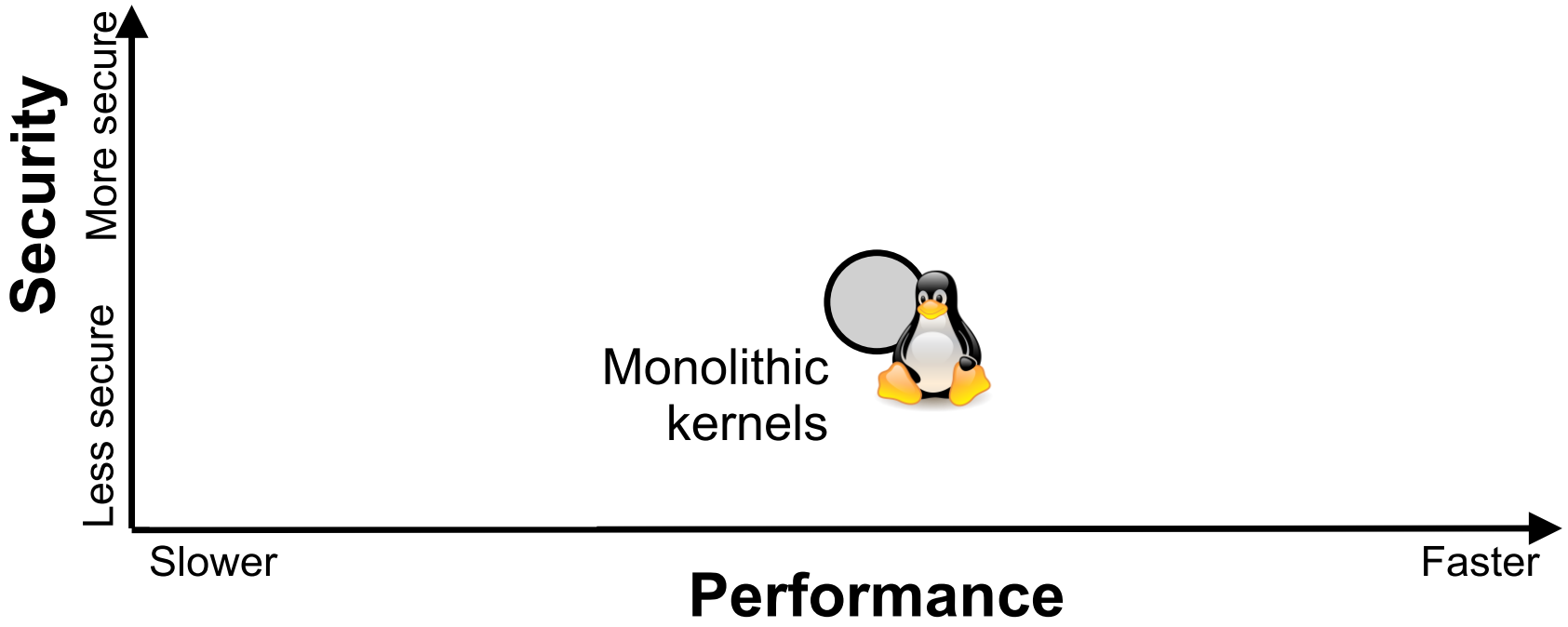
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



Current OS Designs

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

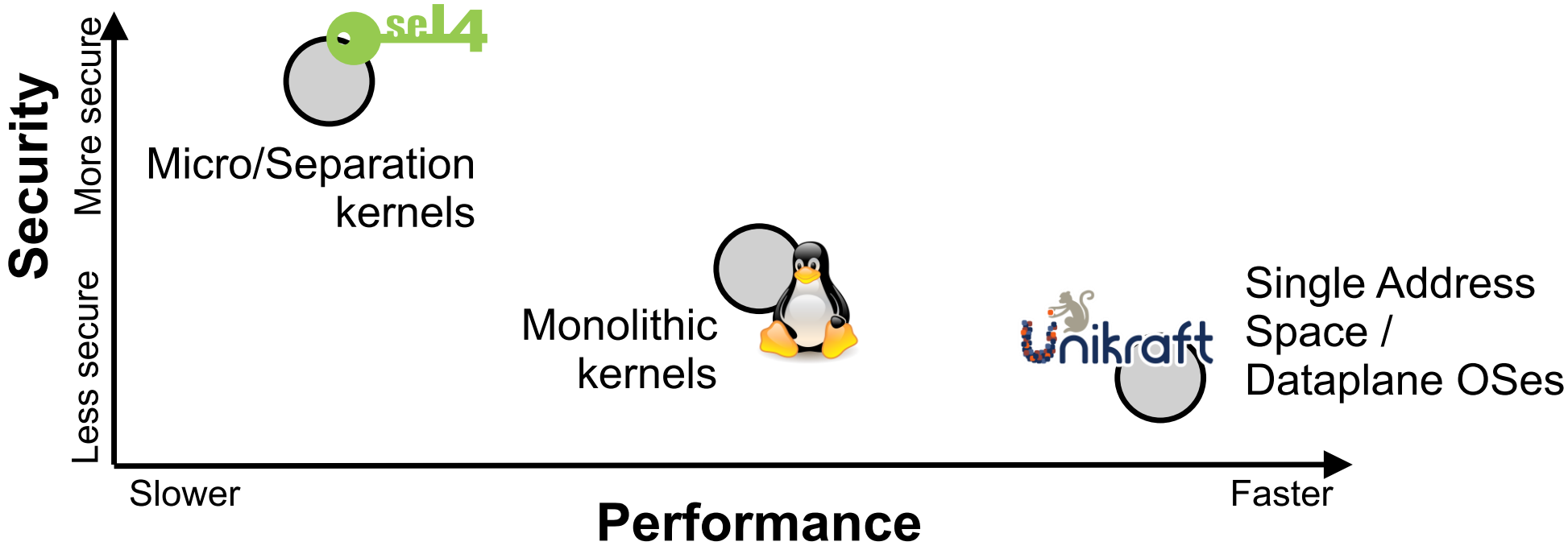
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



Current OS Designs

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

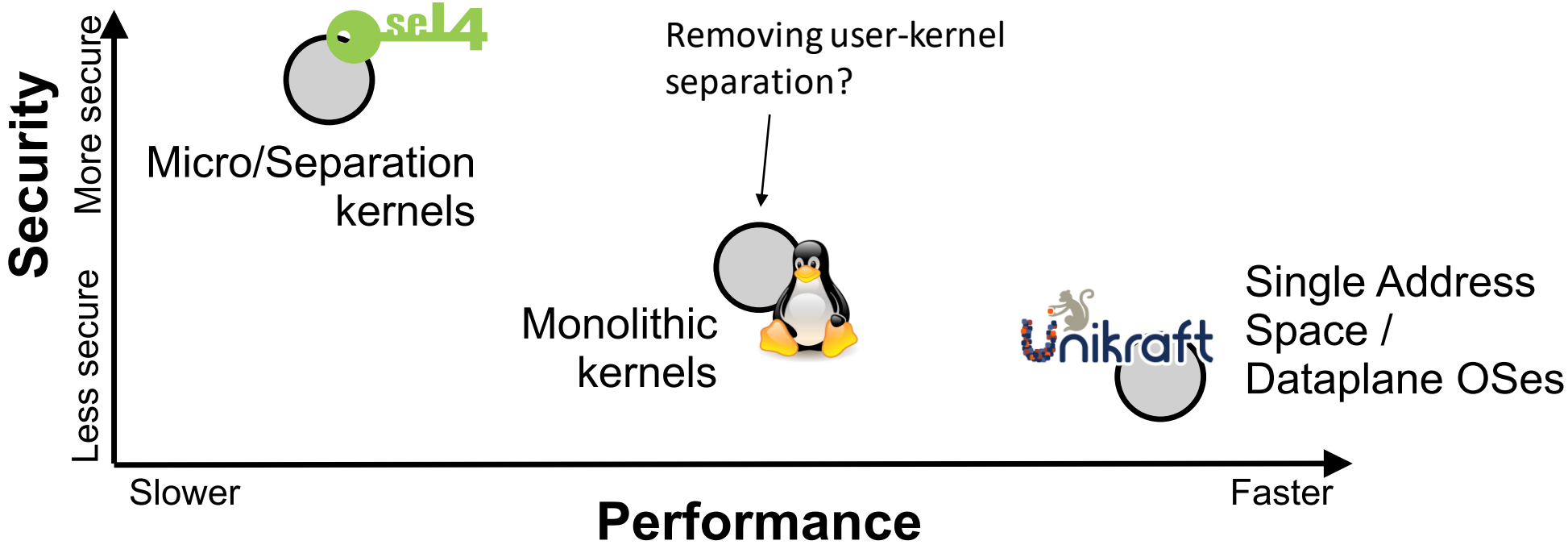
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



Current OS Designs

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

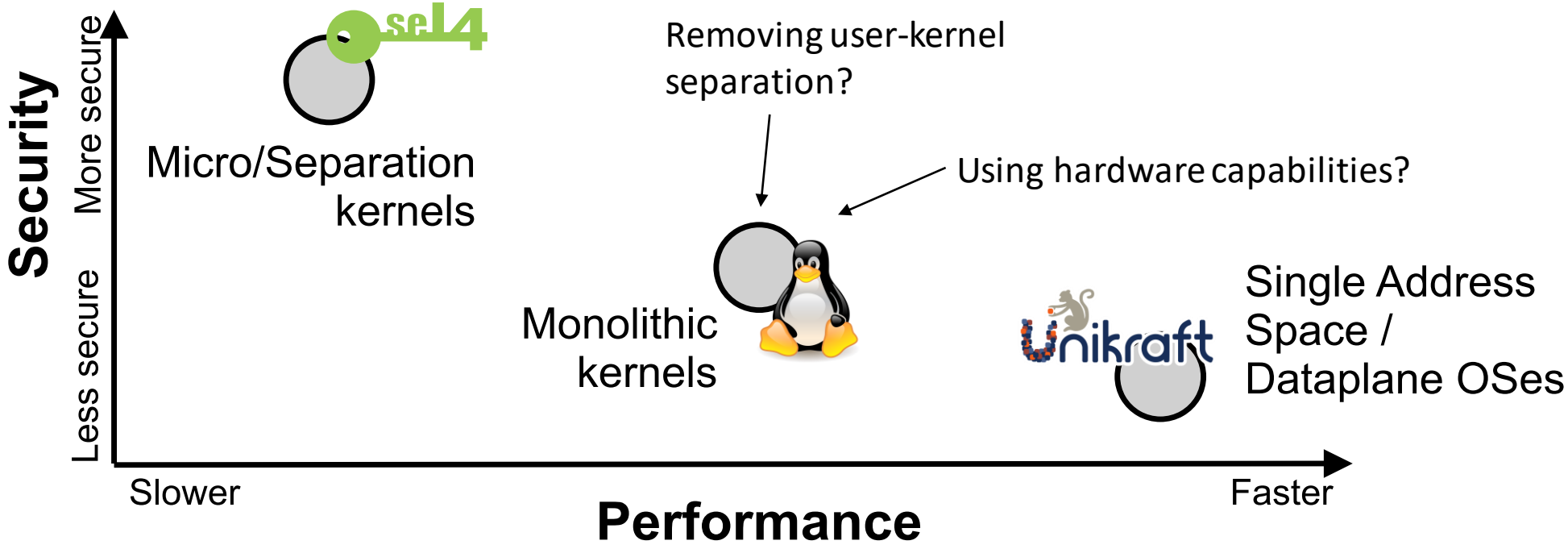
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



Current OS Designs

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

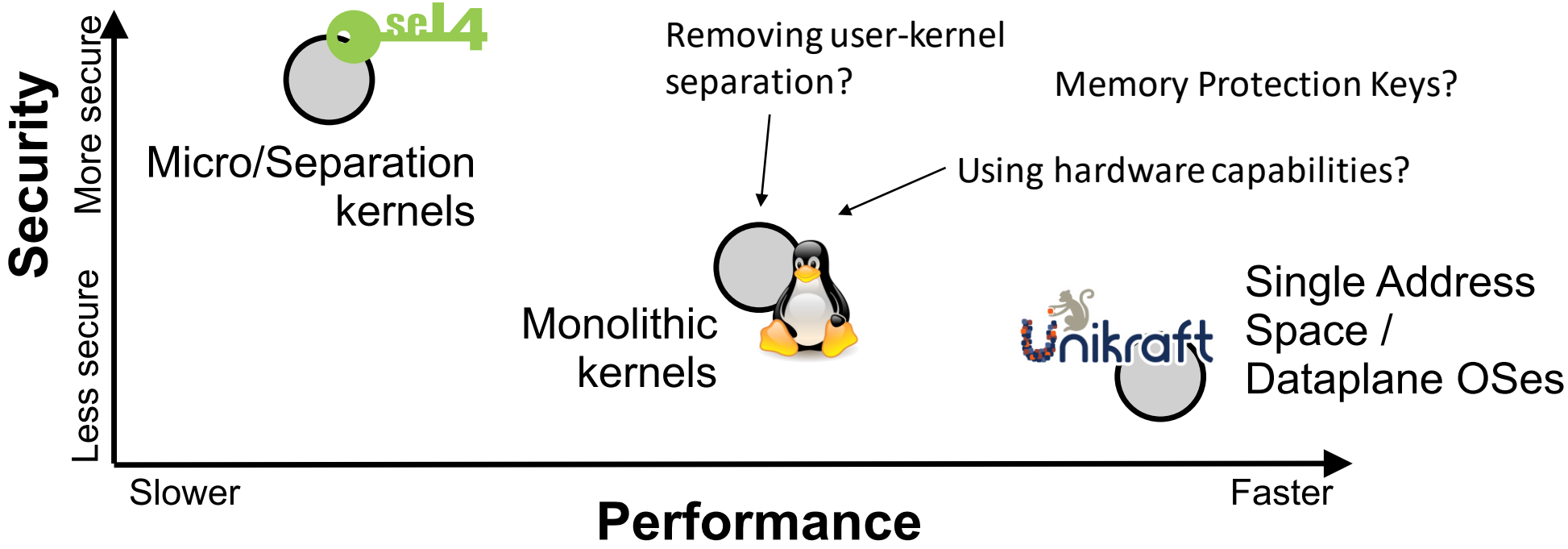
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



Current OS Designs

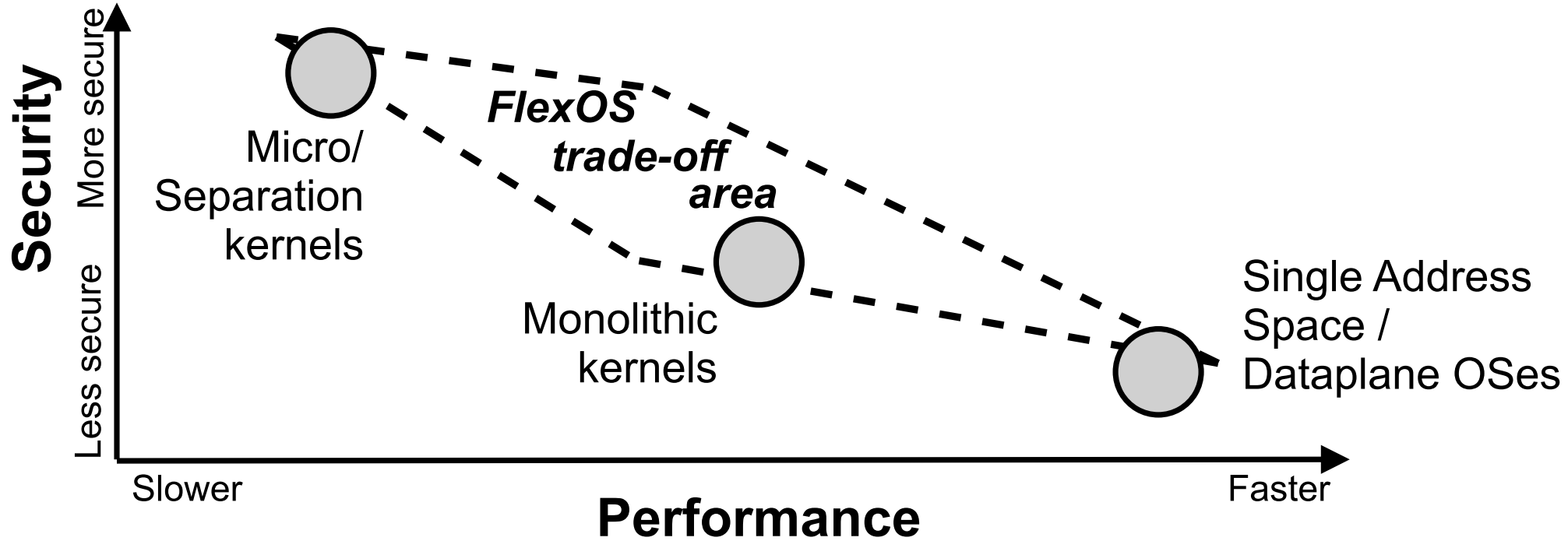
OS security/isolation strategies are **fixed** at design time!

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



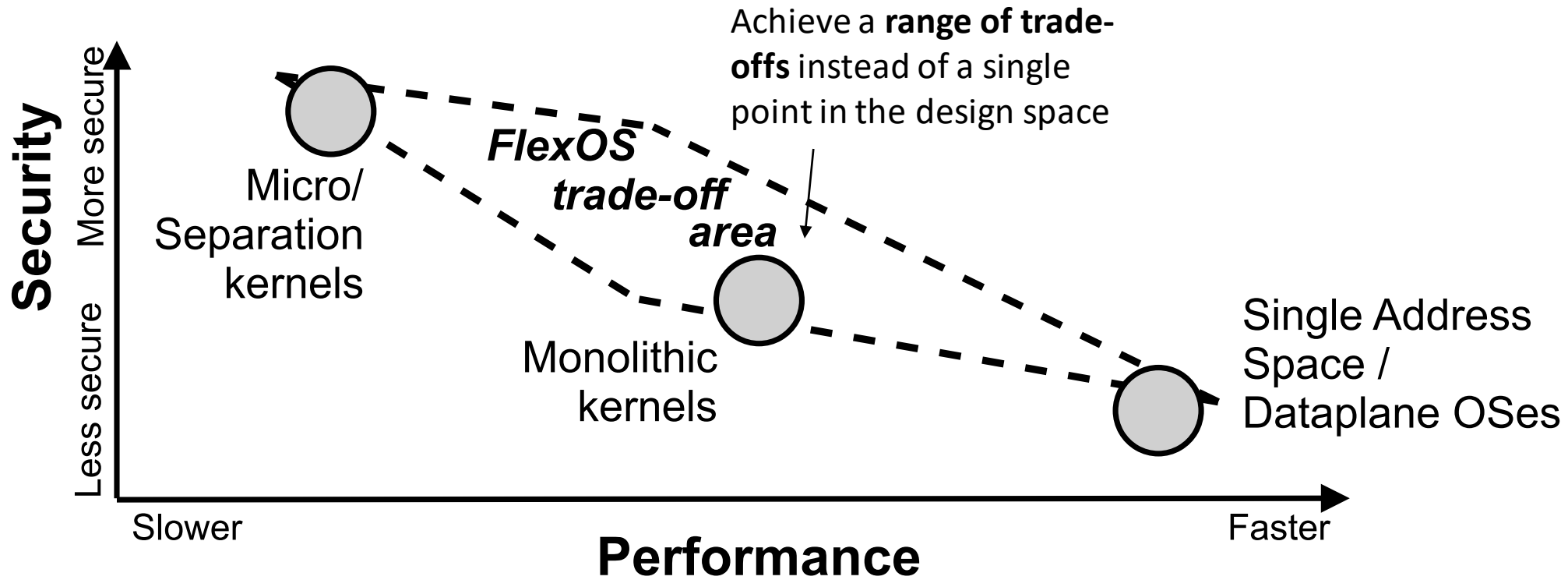
FlexOS: Flexible Isolation

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



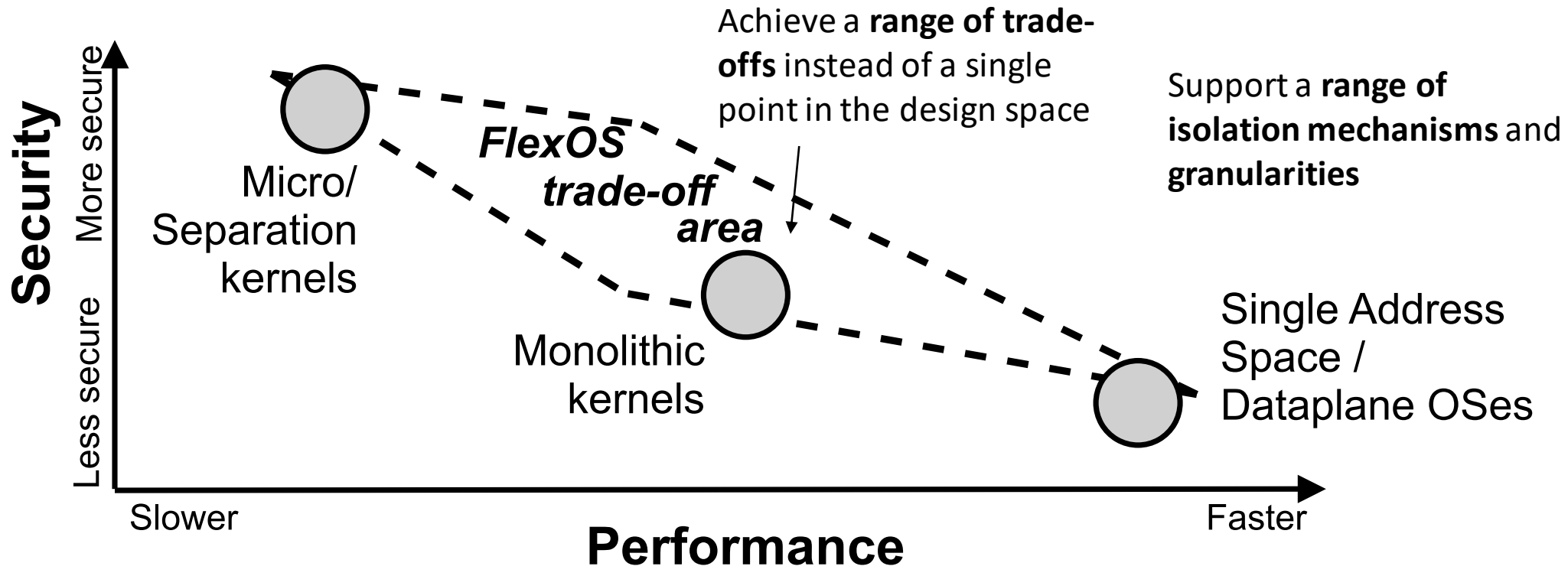
FlexOS: Flexible Isolation

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



FlexOS: Flexible Isolation

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



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

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

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

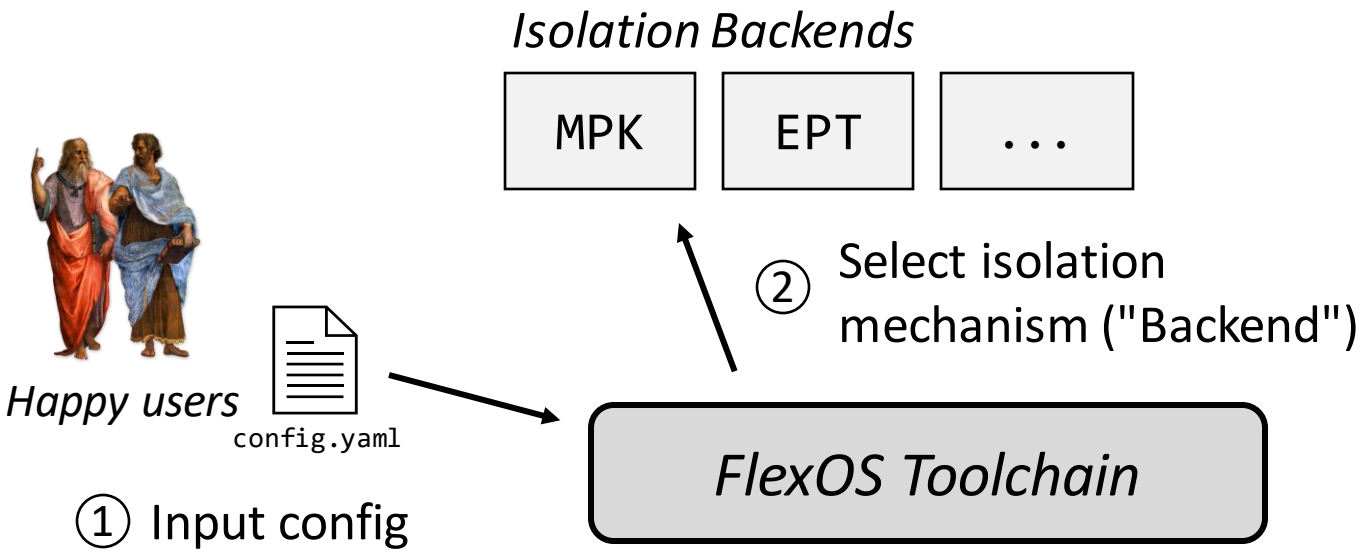
FlexOS Toolchain

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



FlexOS 101: Overview

Happy users

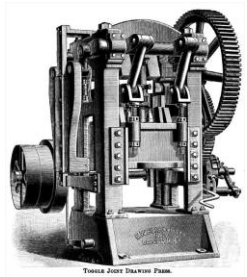
config.yaml

① Input config

Isolation Backends

MPK | EPT | ...

② Select isolation mechanism ("Backend")



FlexOS Toolchain

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

Core Libraries

boot | sched | mm

Kernel & User Libs

ramfs | vfs | ...

FlexOS 101: Overview

Core Libraries

boot | sched | mm

Isolation Backends

MPK | EPT | ...

Kernel & User Libs

ramfs | vfs | ...

Happy users

config.yaml

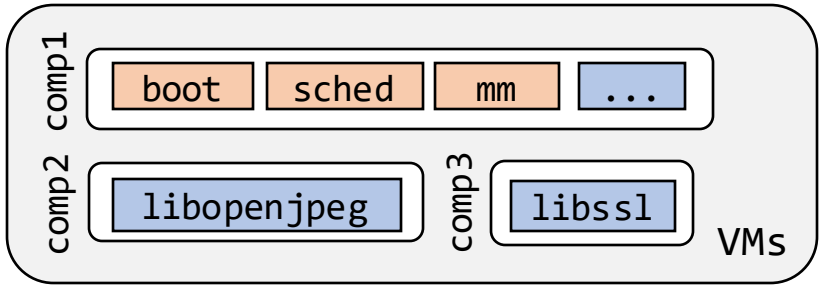
① Input config

FlexOS Toolchain

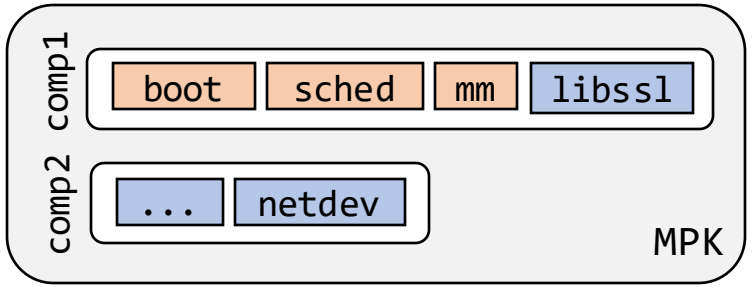
② Select isolation mechanism ("Backend")

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

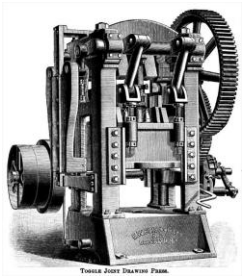
Happy users

config.yaml

① Input config

Isolation Backends

MPK | EPT | ...



② Select isolation mechanism ("Backend")

FlexOS Toolchain

Core Libraries

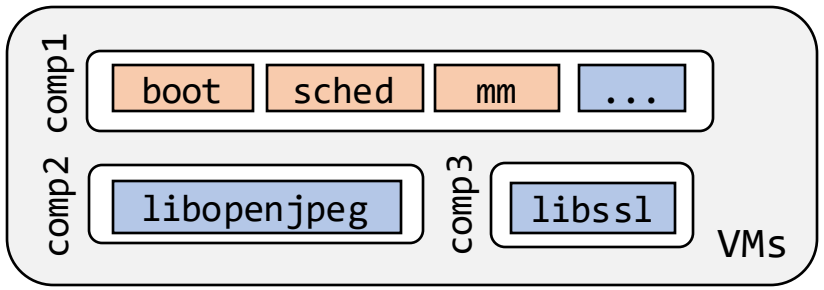
boot | sched | mm

Kernel & User Libs

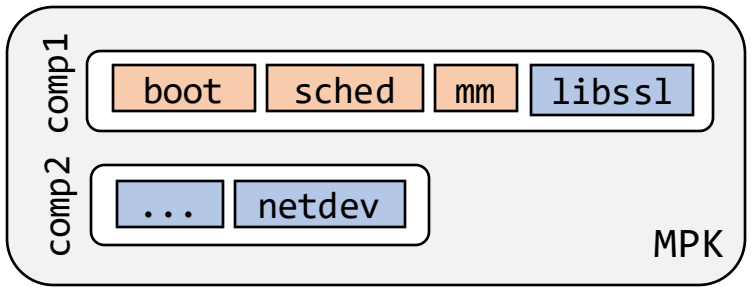
ramfs | vfs | ...

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

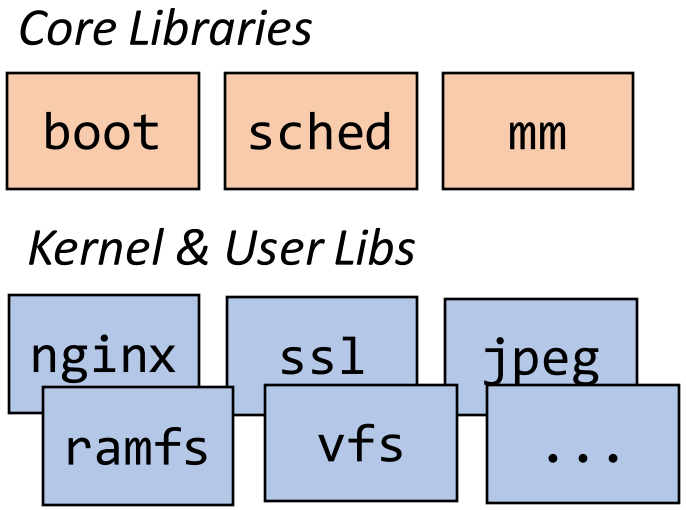
FlexOS 101: Mechanism Abstraction

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

FlexOS 101: Mechanism Abstraction

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

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

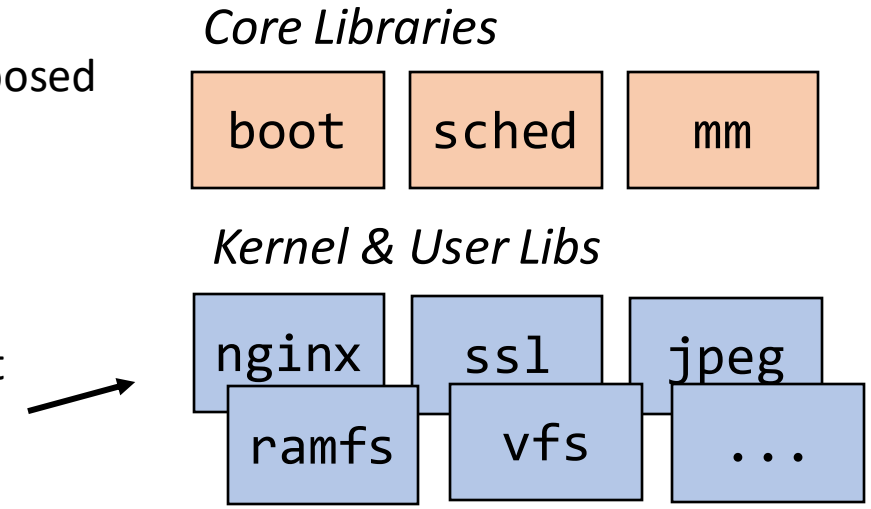


FlexOS 101: Mechanism Abstraction

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

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

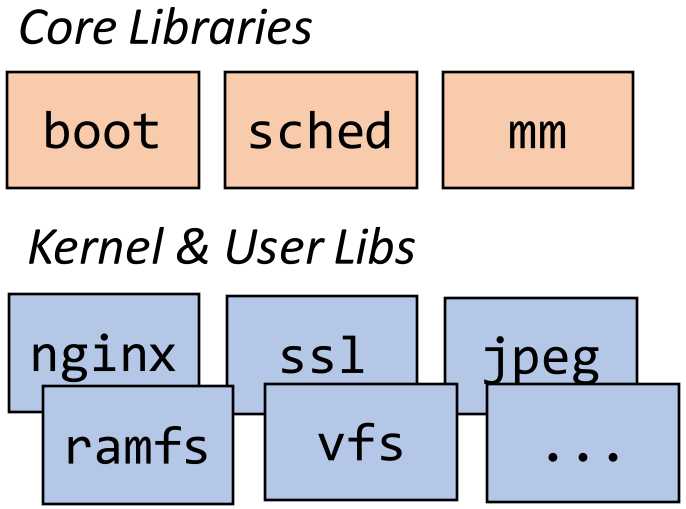
Reuse libraries as finest granularity of compartmentalization



FlexOS 101: Mechanism Abstraction

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

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)

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

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 | ...

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**.

MPK | VMs | TEEs | ...

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

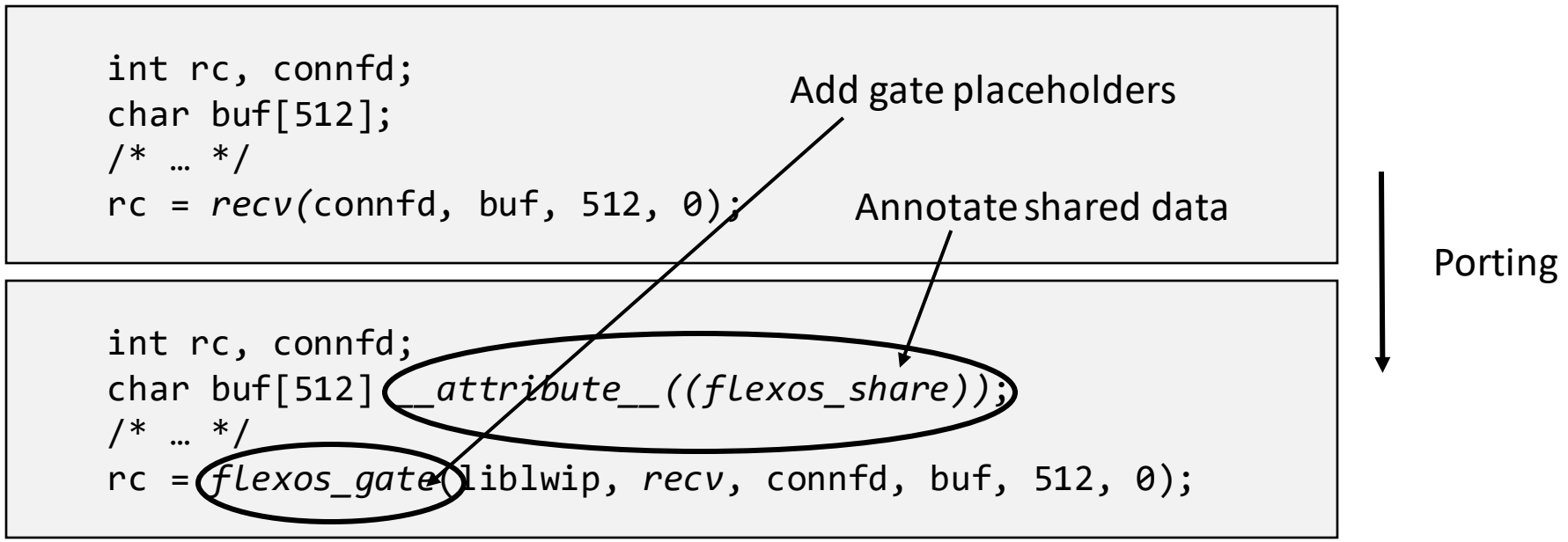
```
int rc, connfd;
char buf[512];
/* … */
rc = recv(connfd, buf, 512, 0);
```

Annotate shared data

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

Porting

FlexOS 101: Compartmentalization API



FlexOS 101: Compartmentalization API

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

Add gate placeholders

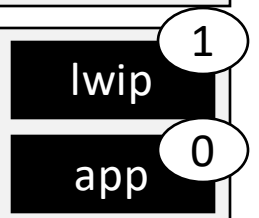
Annotate shared data

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

Porting

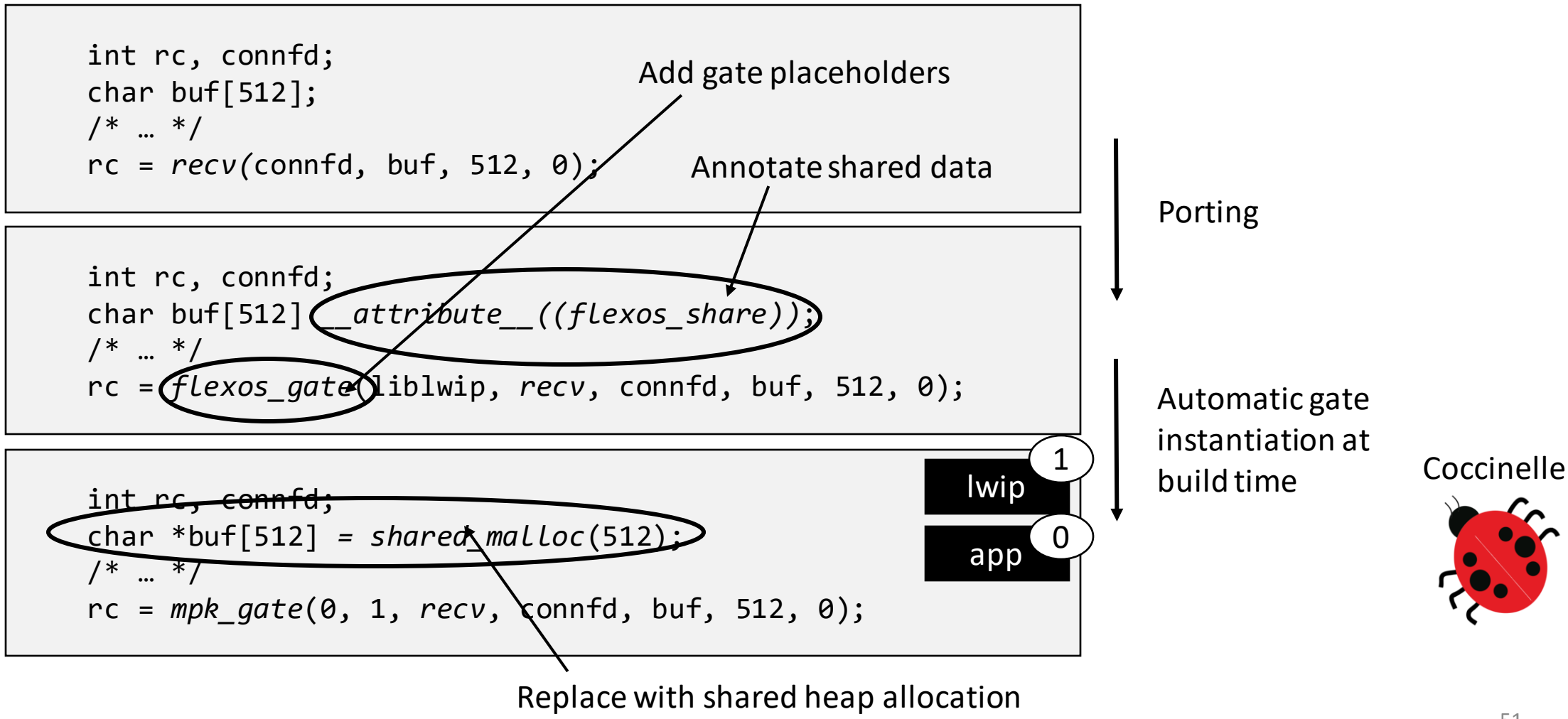
Automatic gate instantiation at build time

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

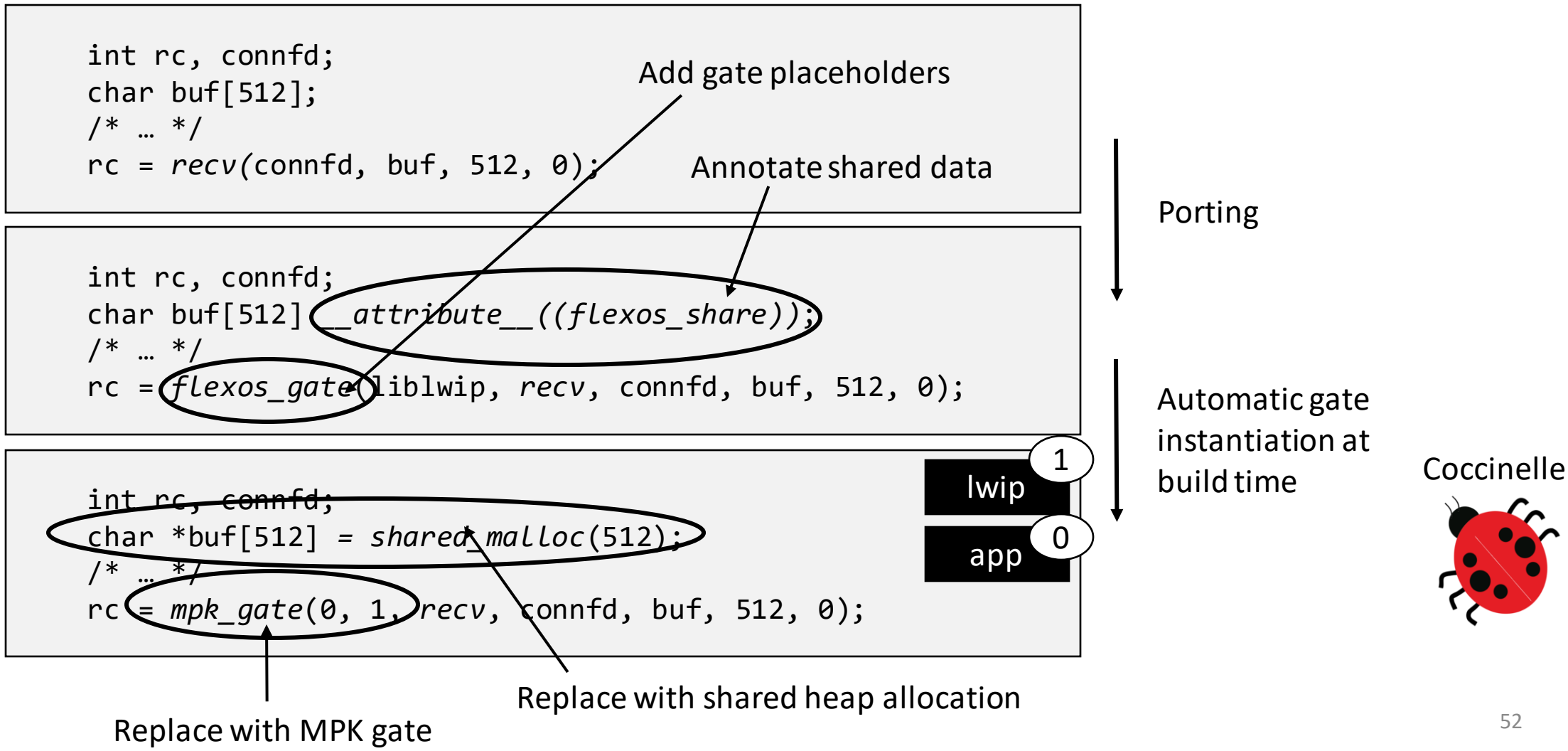


Coccinelle

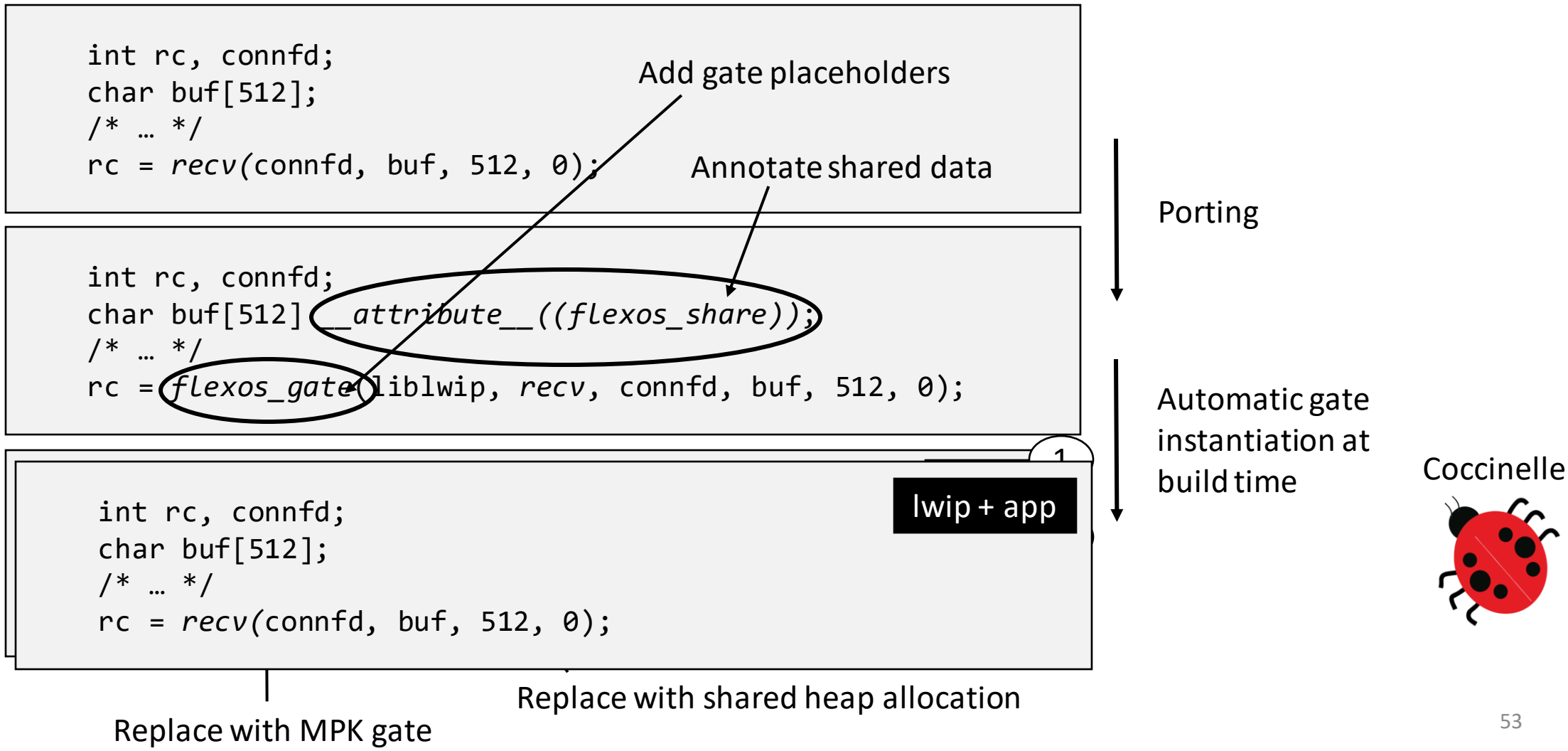
FlexOS 101: Compartmentalization API



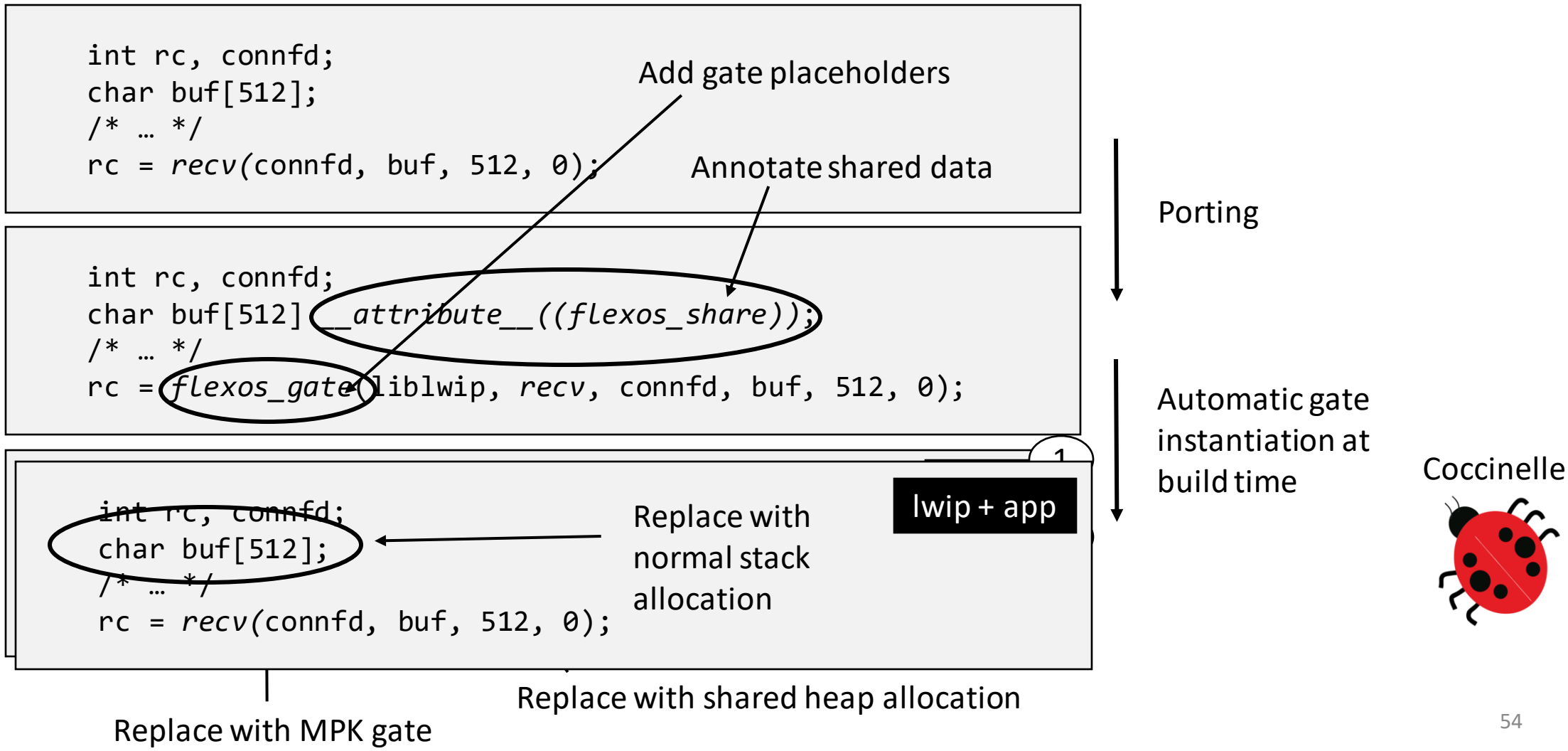
FlexOS 101: Compartmentalization API



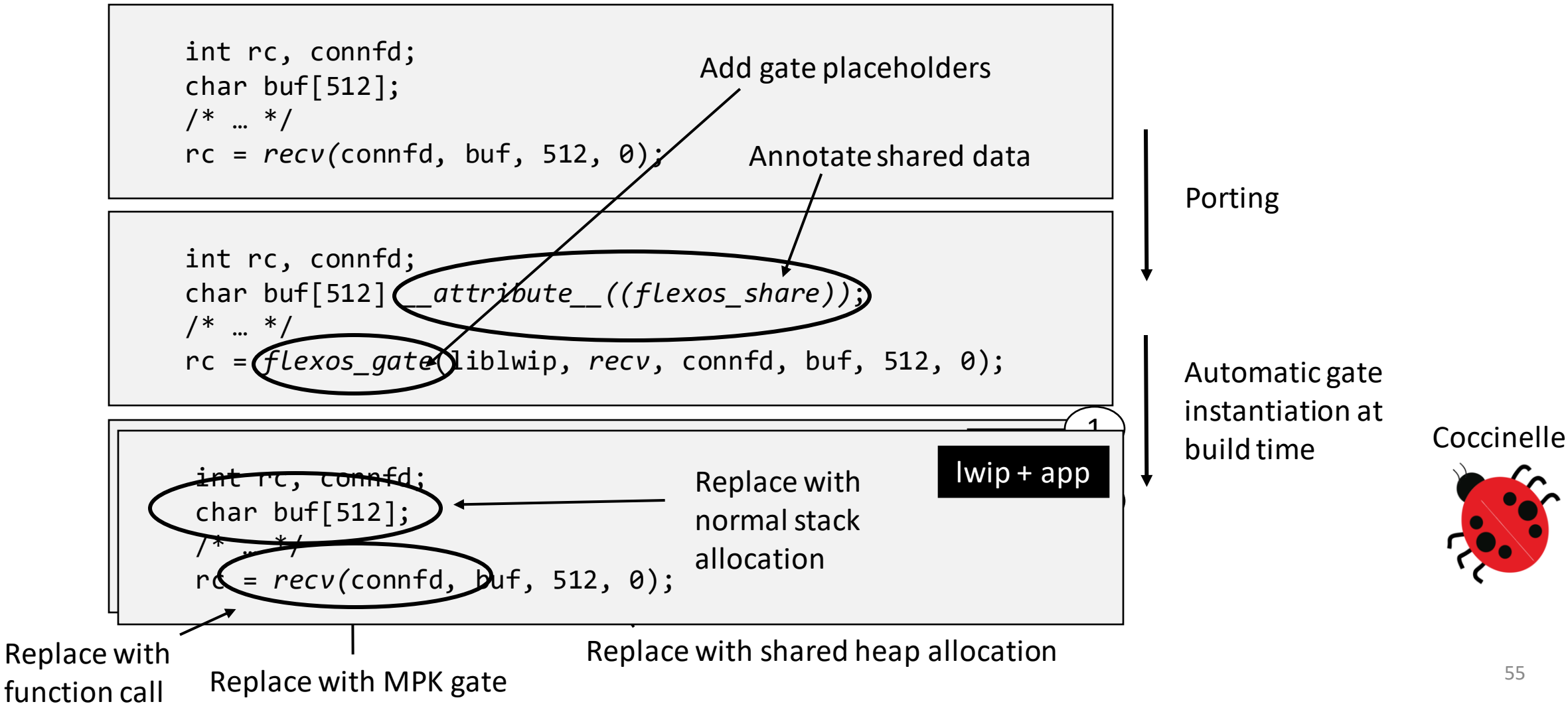
FlexOS 101: Compartmentalization API



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

Prototype

Implementation **on top of Unikraft**

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

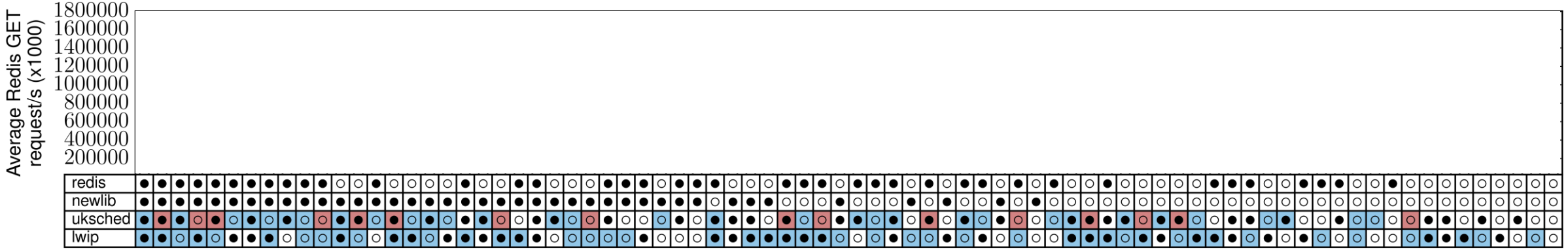
Port of libraries: network stack, scheduler, filesystem, time subsystem

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

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

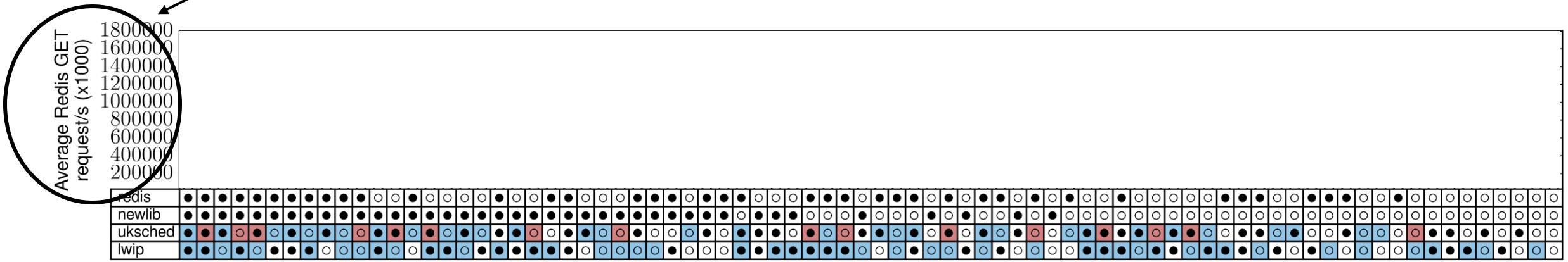
more results in our paper 🙂

Flexibility



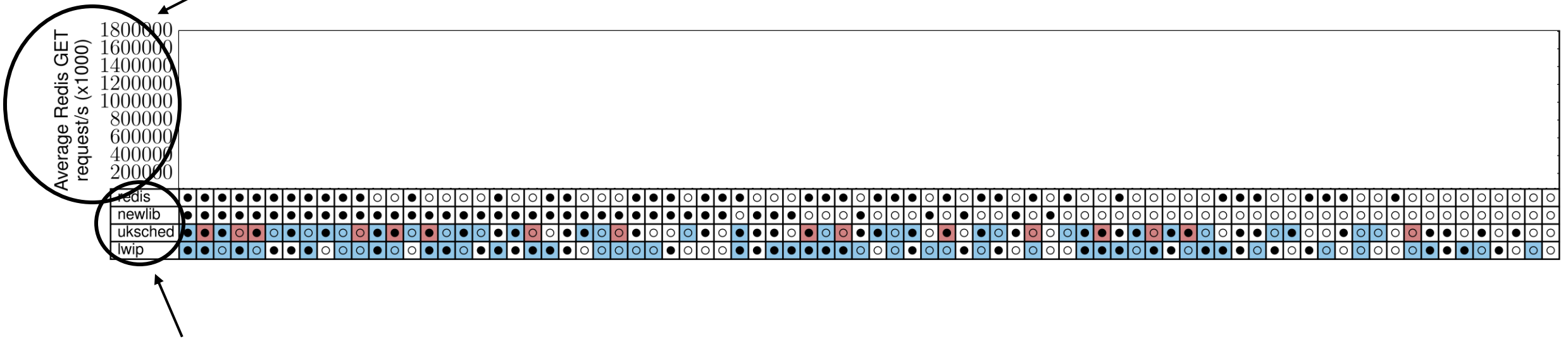
Flexibility

Runtime performance with Redis in requests/s



Flexibility

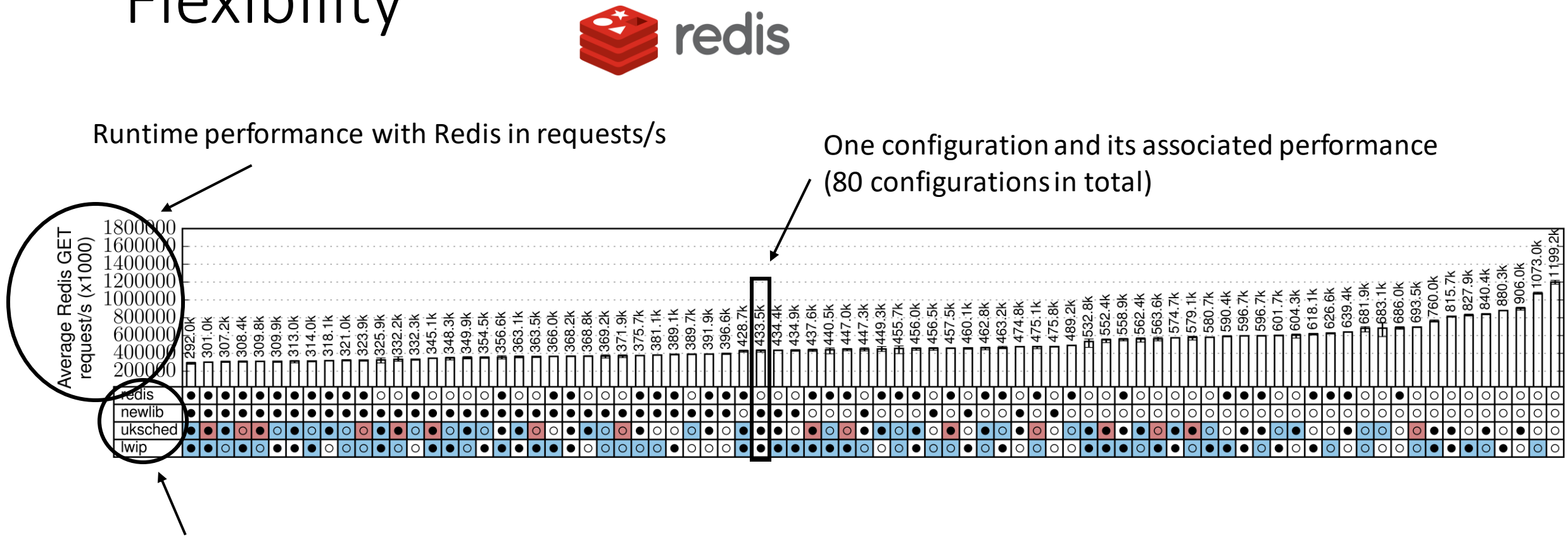
Runtime performance with Redis in requests/s



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

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

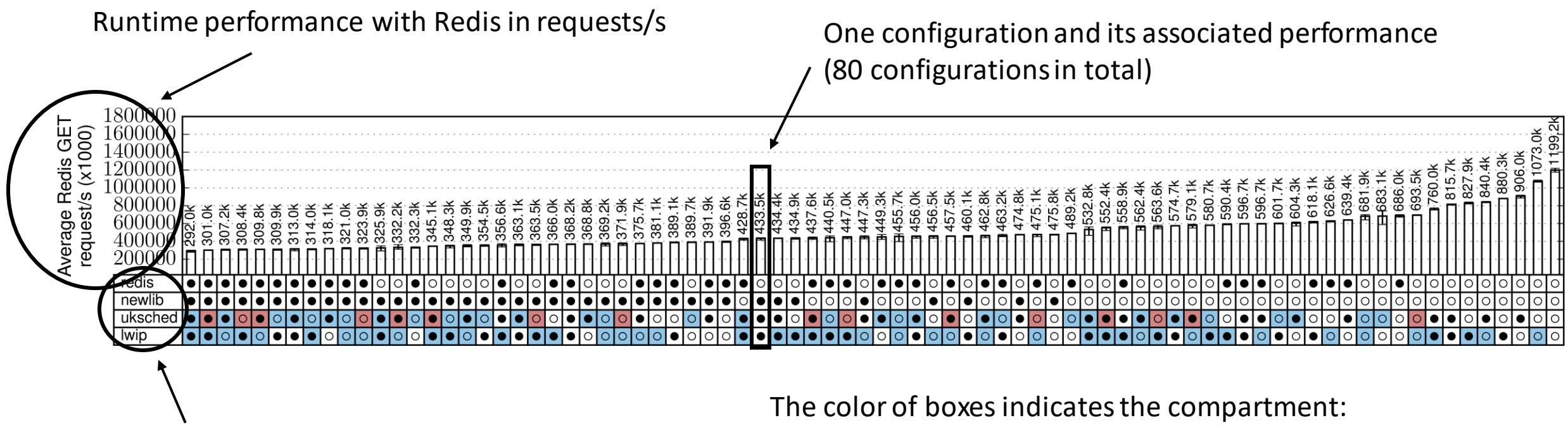
Flexibility



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



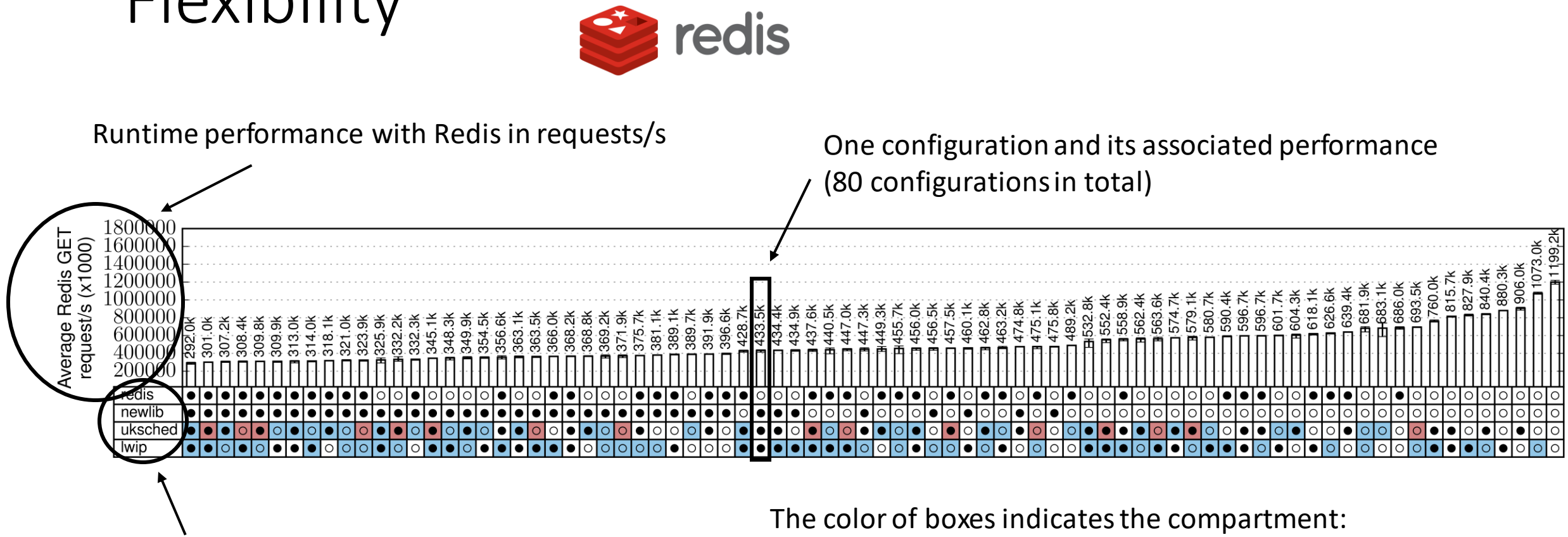
FlexOS libraries used in the Redis image (only a subset for readability):

- 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

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)

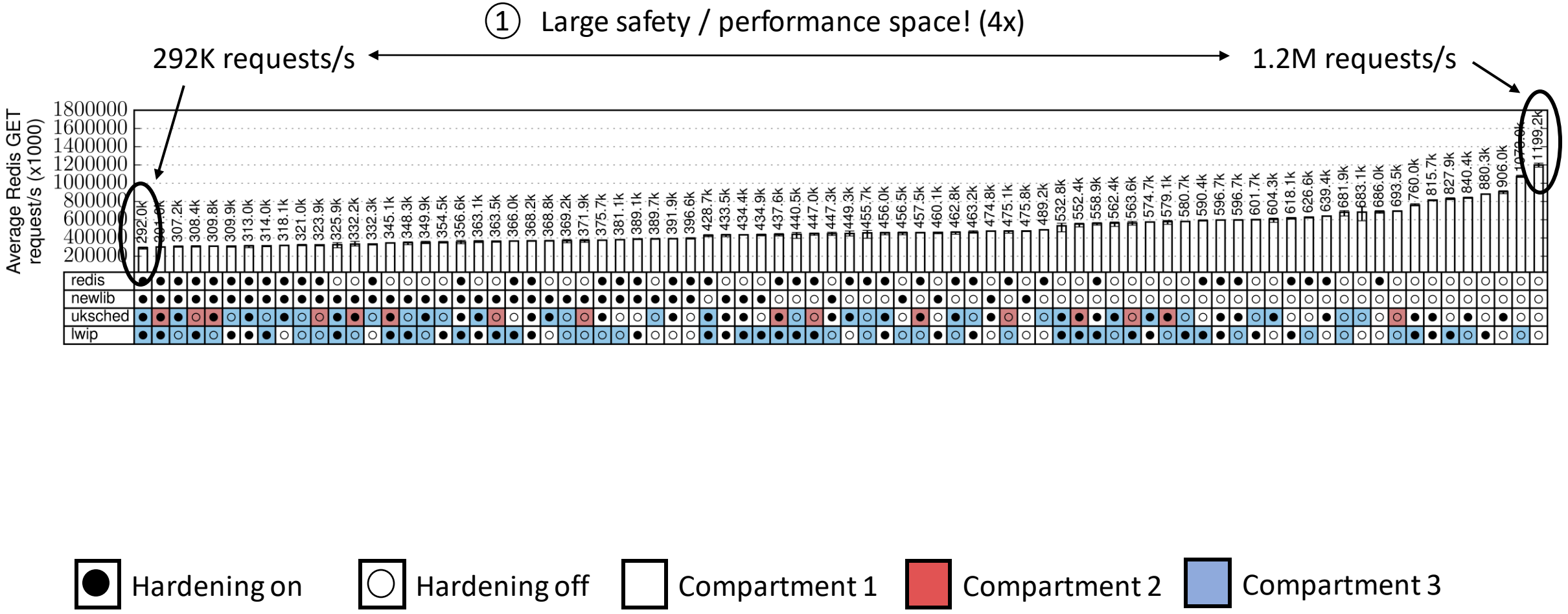
The color of boxes indicates the compartment:

Compartment 1 Compartment 2 Compartment 3

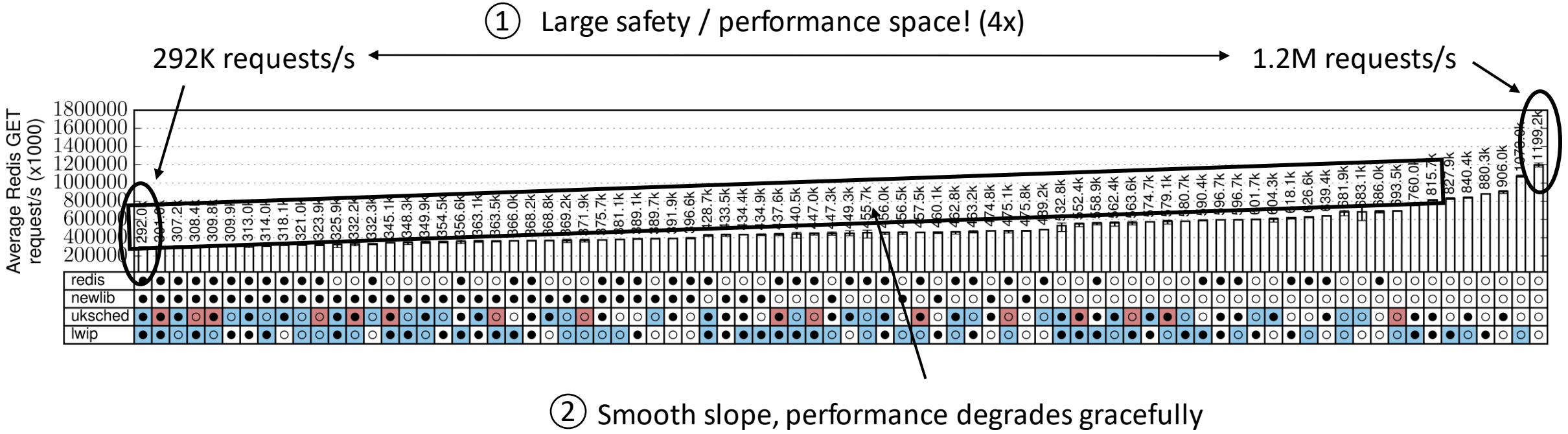
The dot whether hardening (ASan, Safestack, etc.) is enabled:

Hardening on Hardening off

Flexibility

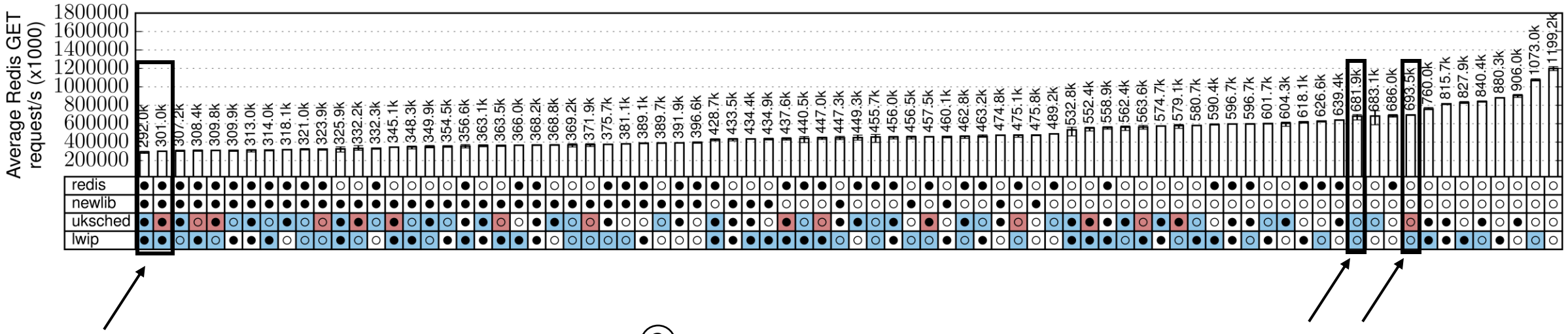


Flexibility



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

Flexibility

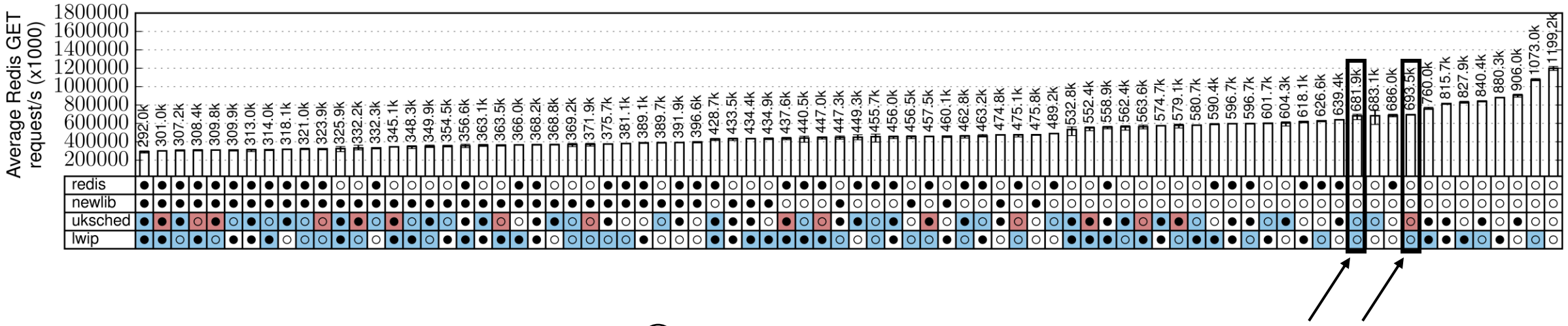


③ Similar performance, very different properties!

need to reason about communication patterns, fast paths

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

Flexibility

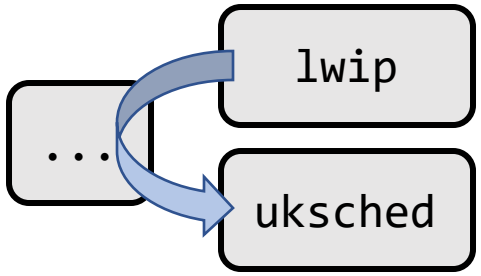


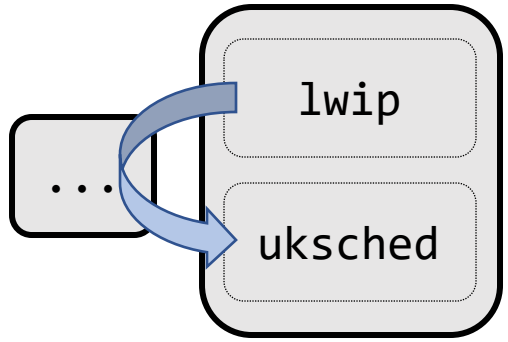
③ Similar performance, very different properties!

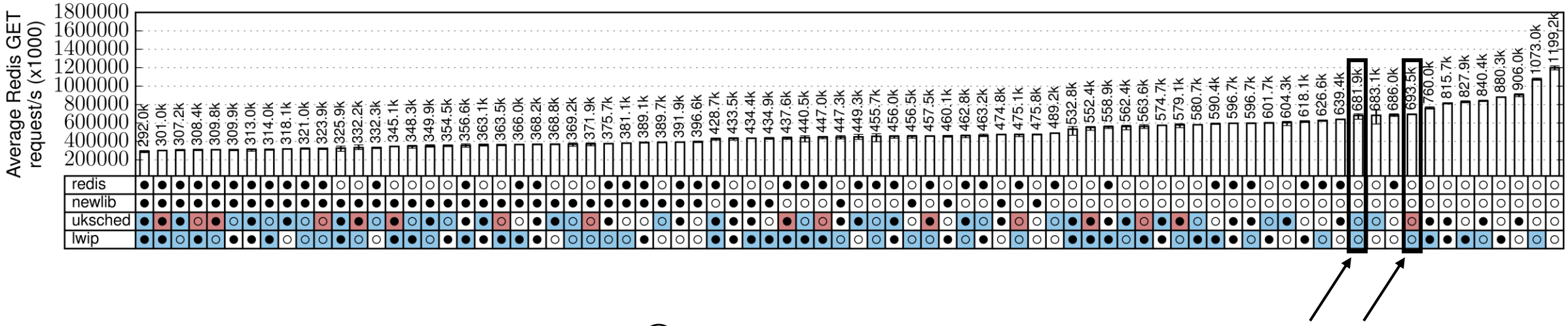
need to reason about communication patterns, fast paths

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

Flexibility





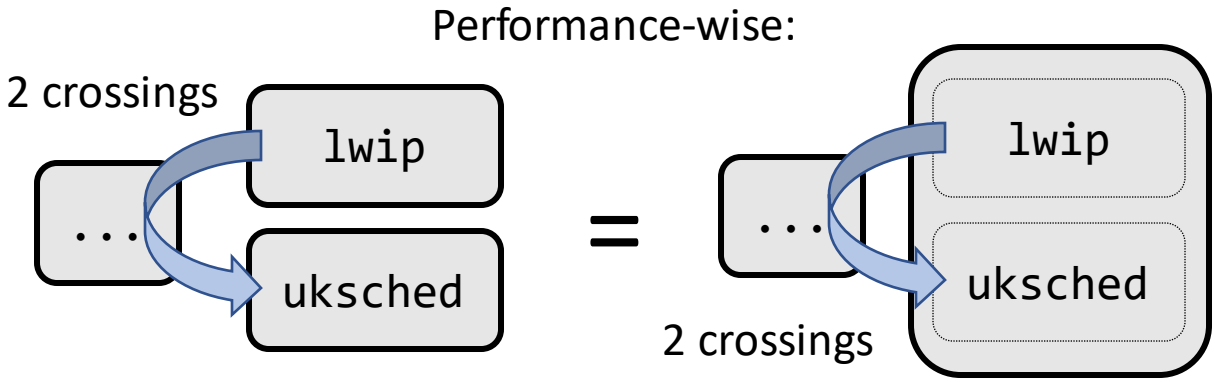


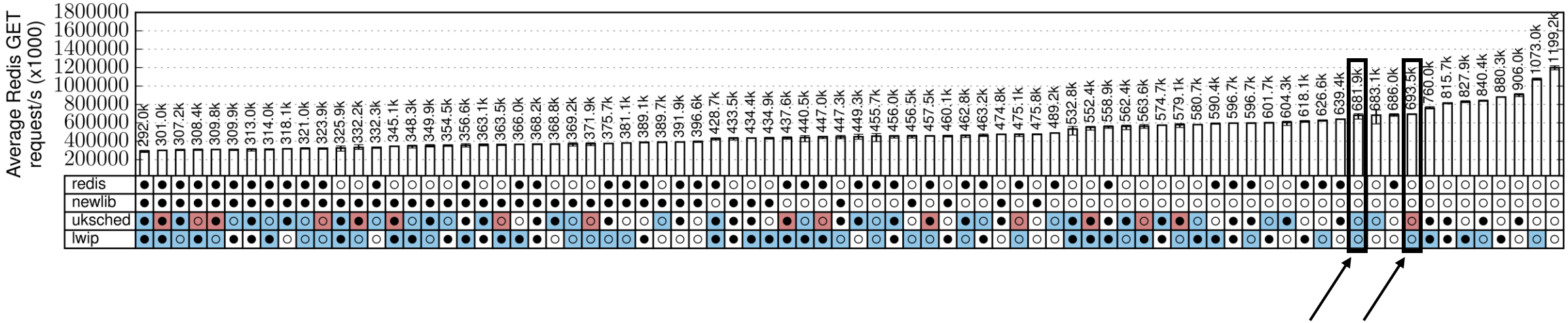
③ Similar performance, very different properties!

need to reason about communication patterns, fast paths

● Hardening on ○ Hardening off □ Compartment 1 ■ Compartment 2 ■ Compartment 3

Flexibility



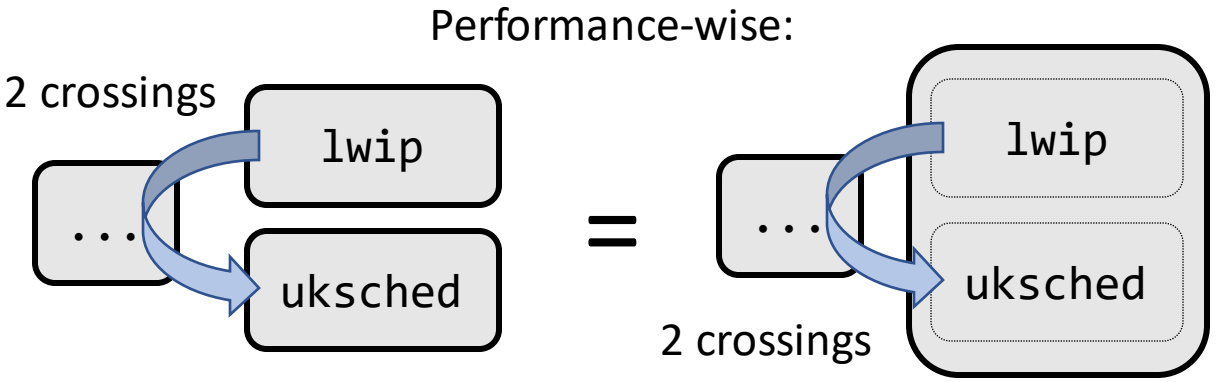


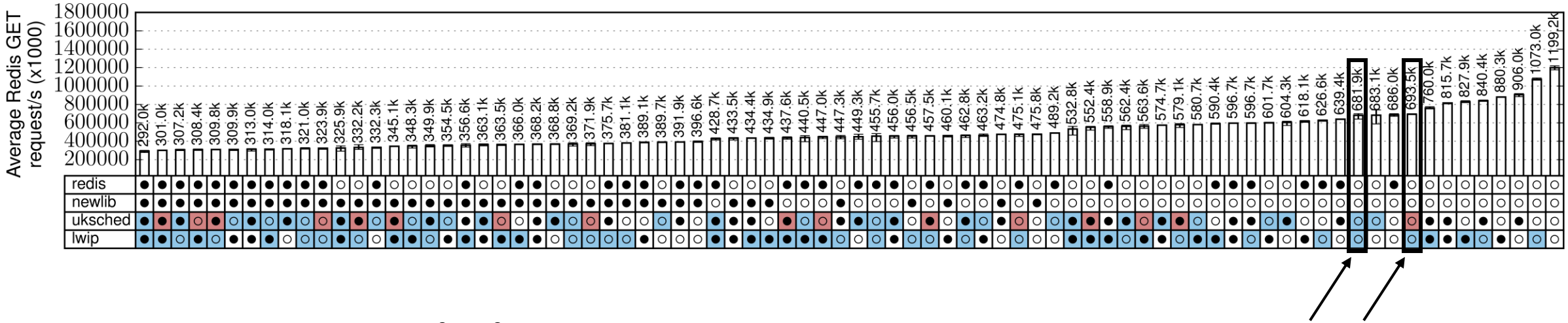
③ Similar performance, very different properties!

need to reason about communication patterns, fast paths

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

Flexibility



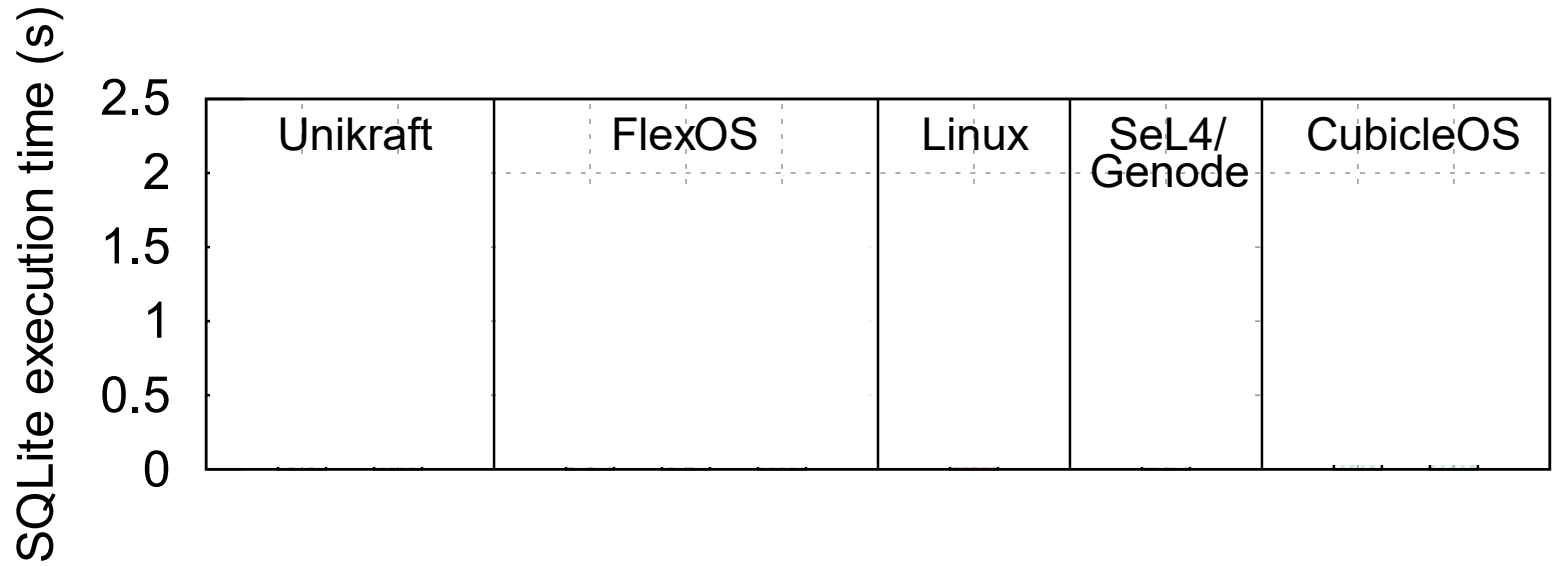


You can get some safety for free by exploring intelligently

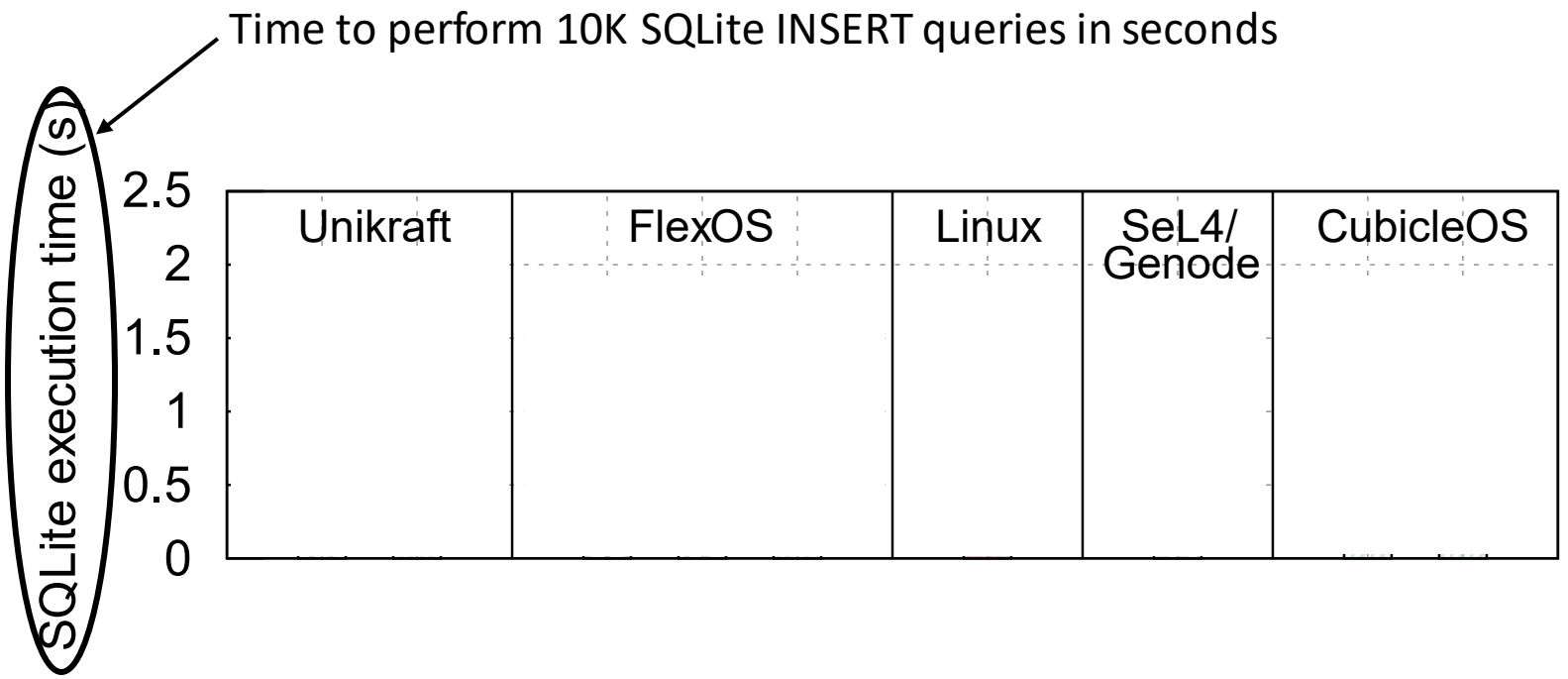
③ Similar performance, very different properties!
need to reason about communication patterns, fast paths

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

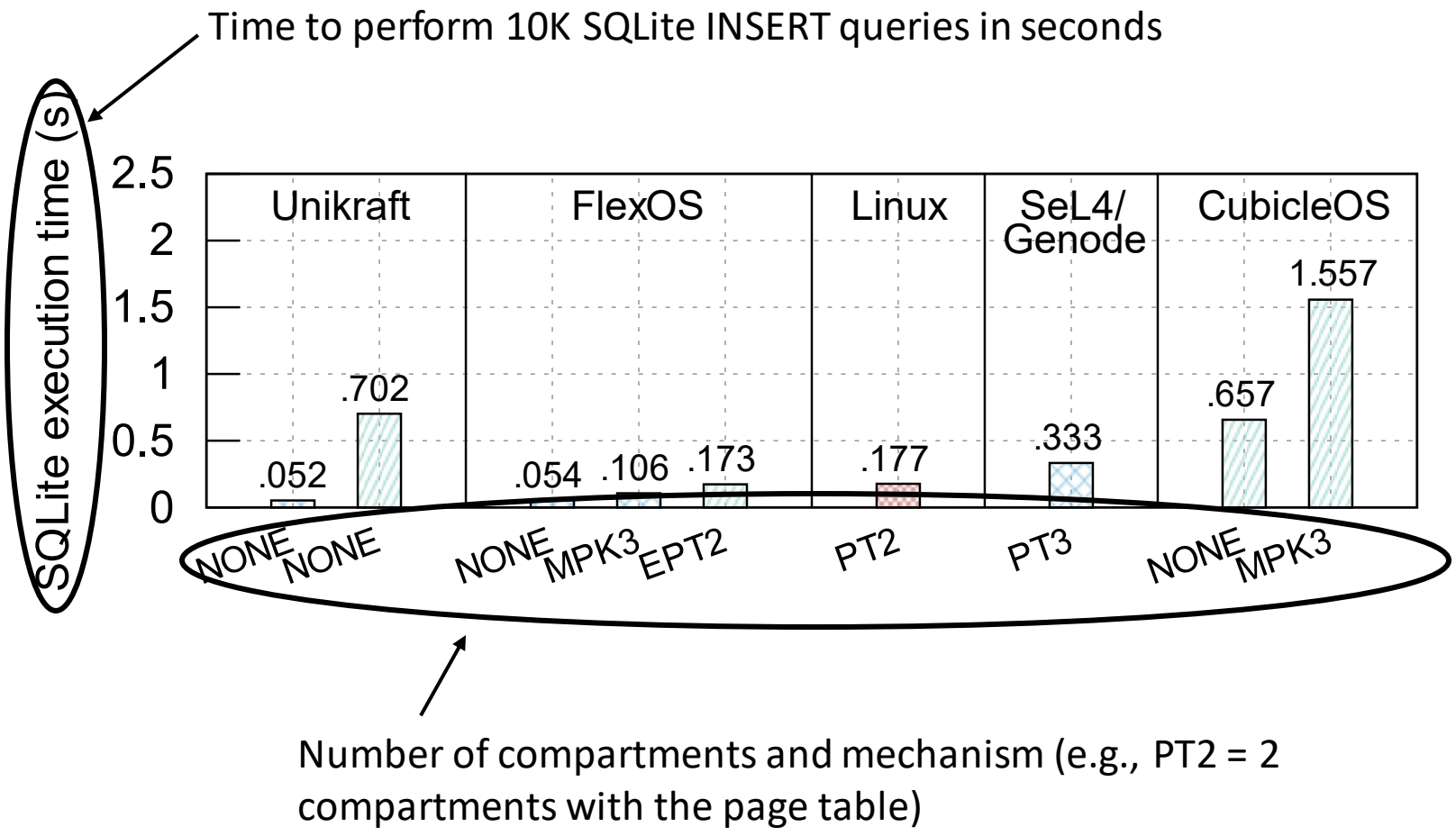
Performance



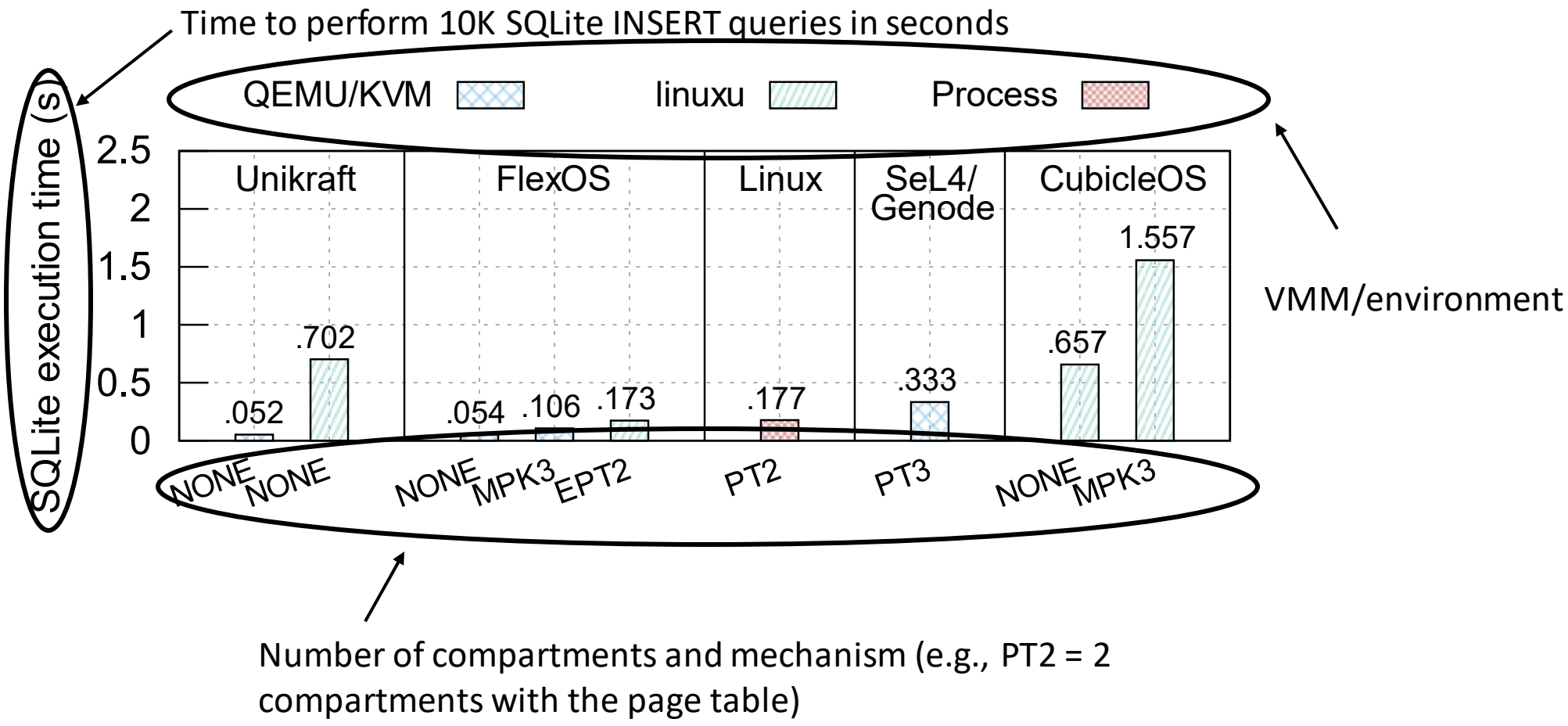
Performance



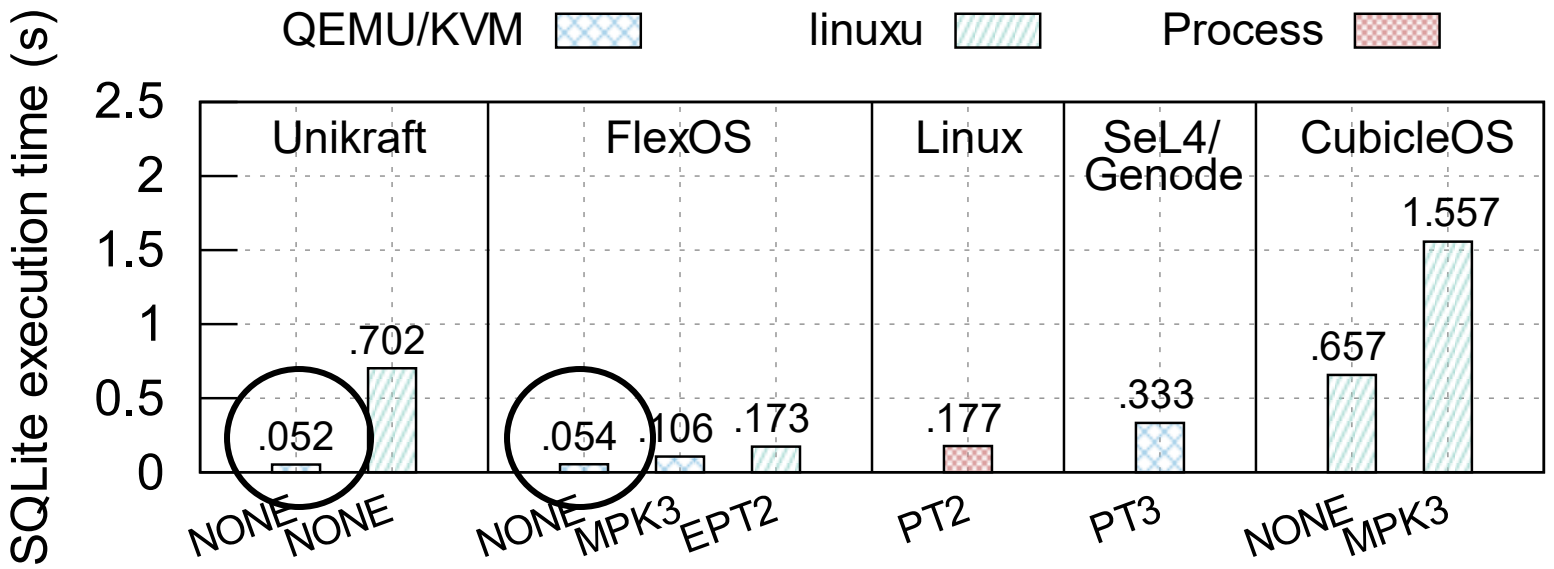
Performance



Performance

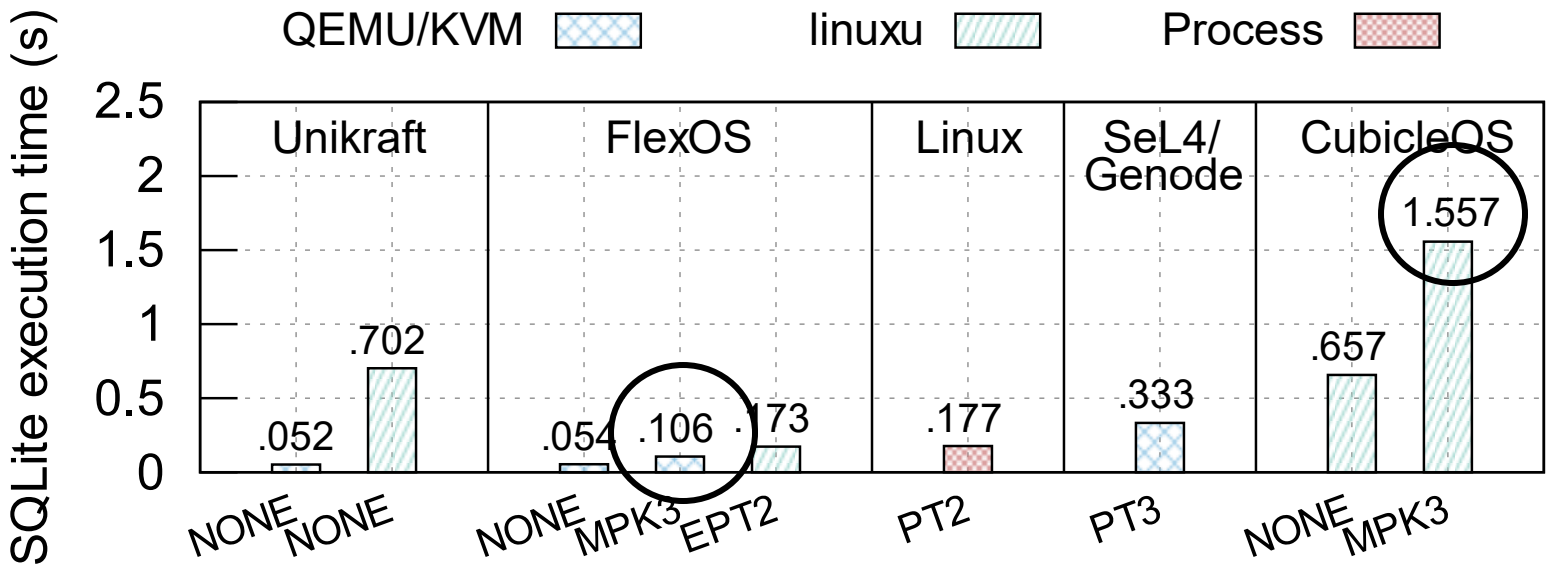


Performance



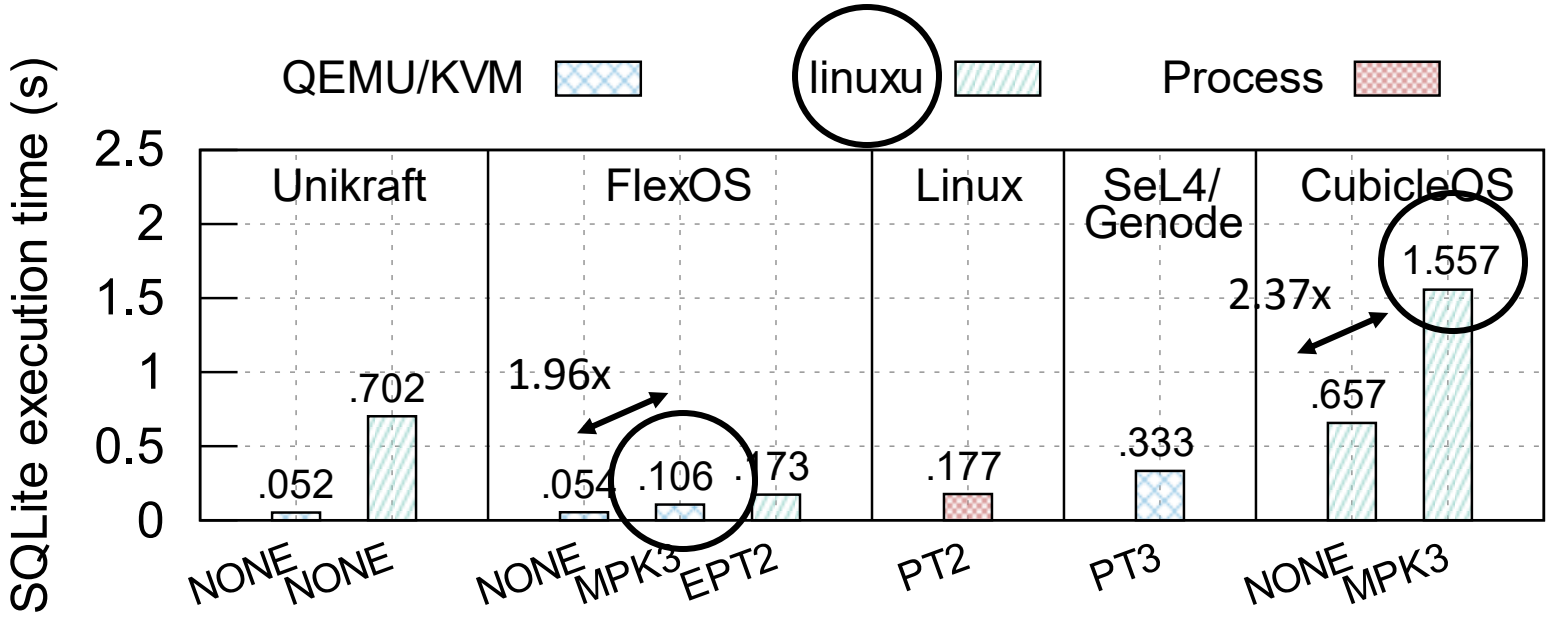
① No overhead when disabling isolation – you only pay for what you get

Performance



② The MPK backend compares very positively to competing solutions

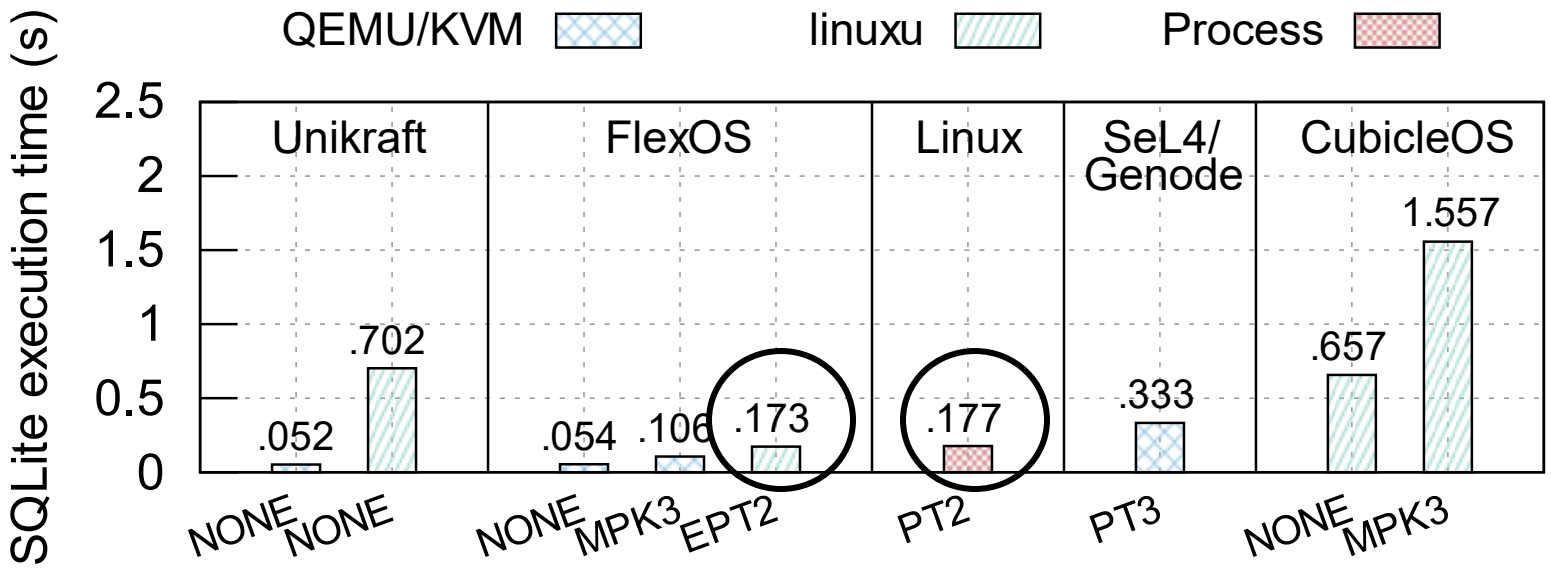
Performance



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!

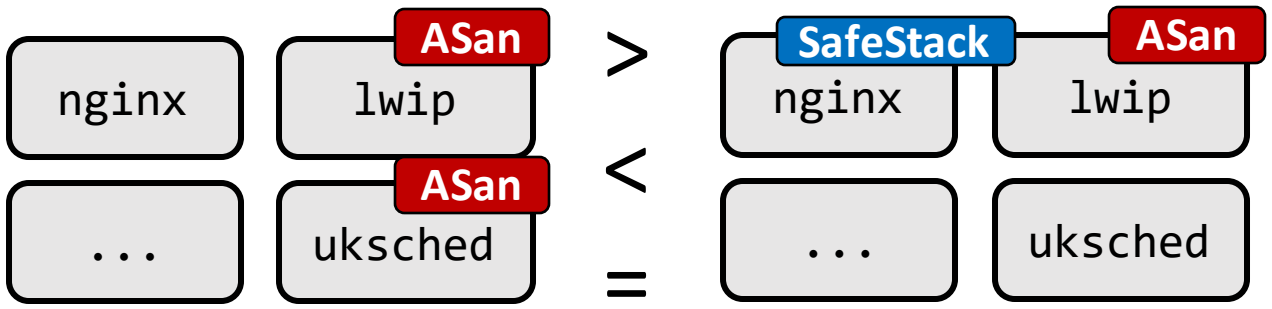
Problem: some configurations are not comparable

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

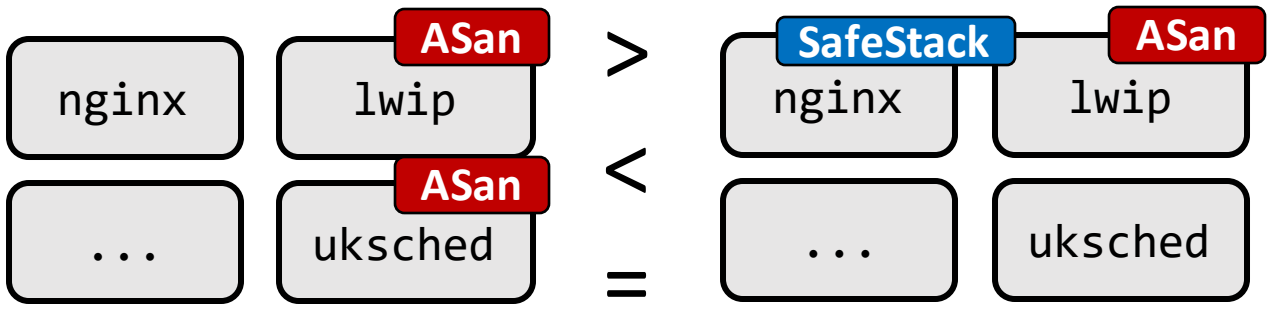


Exploring the Design Space

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



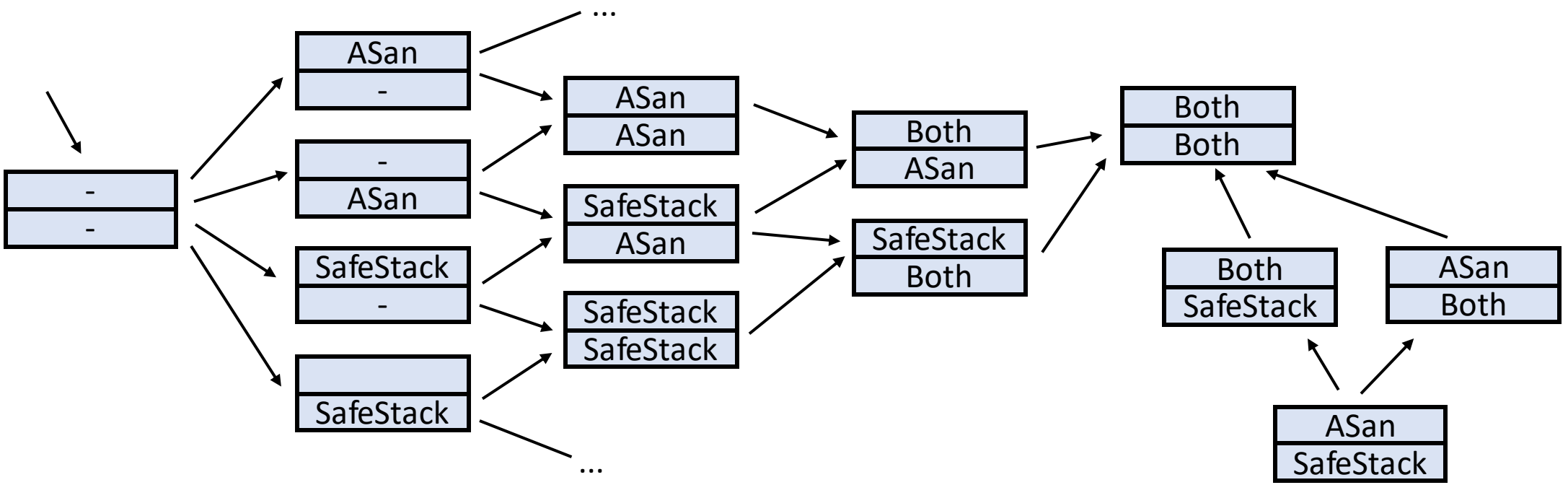
How can we reason about security/performance trade-offs?

Exploring the Design Space

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

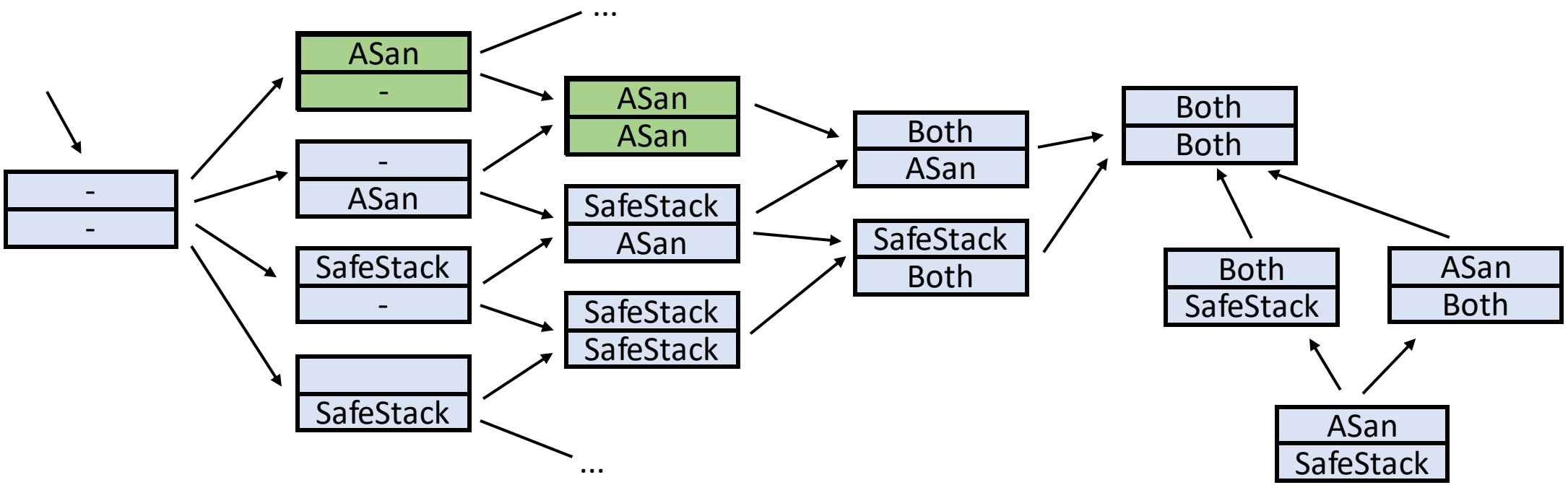
Exploring the Design Space

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



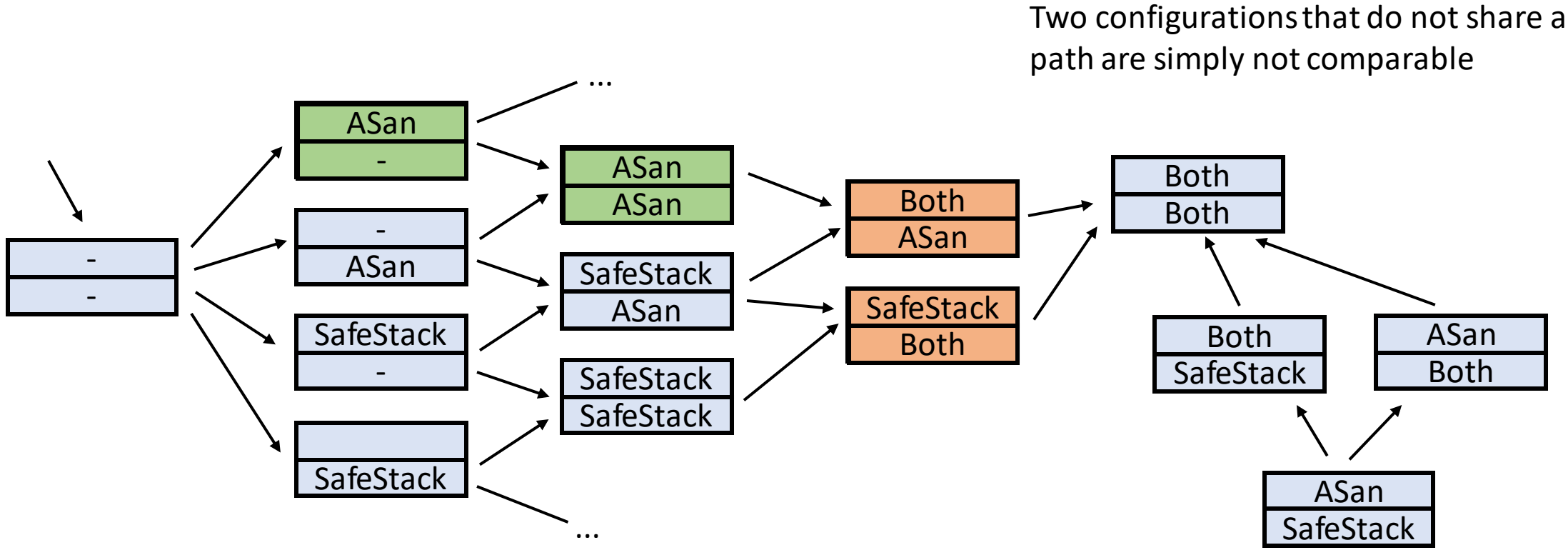
Exploring the Design Space

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



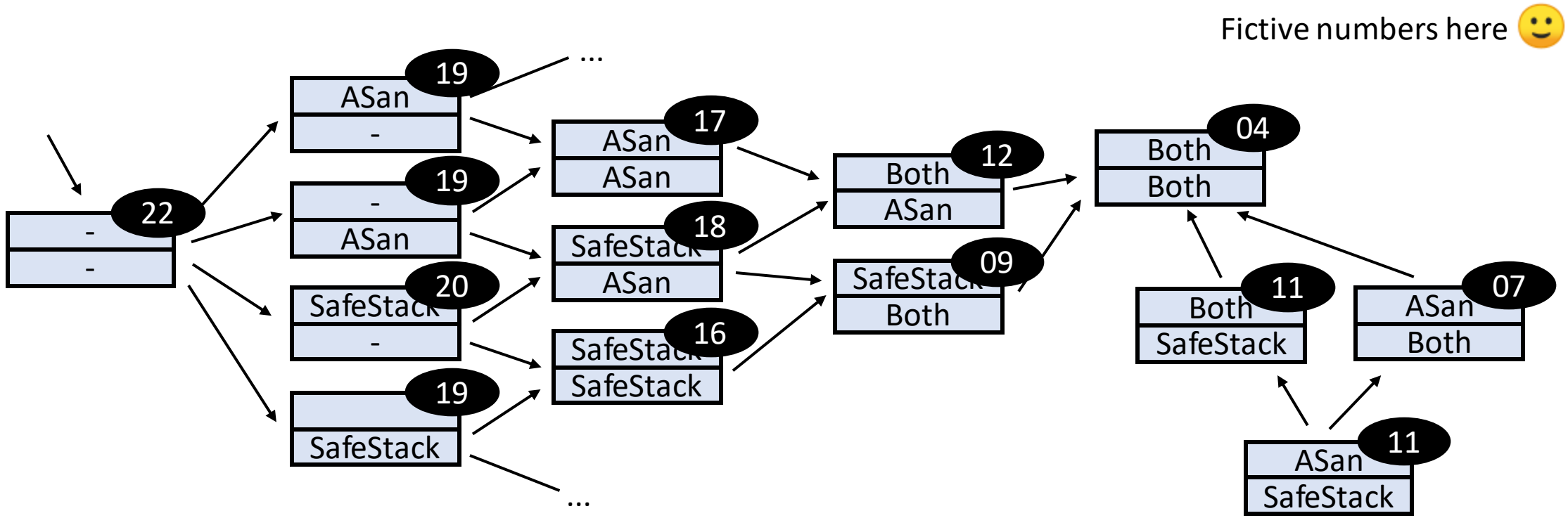
Exploring the Design Space

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



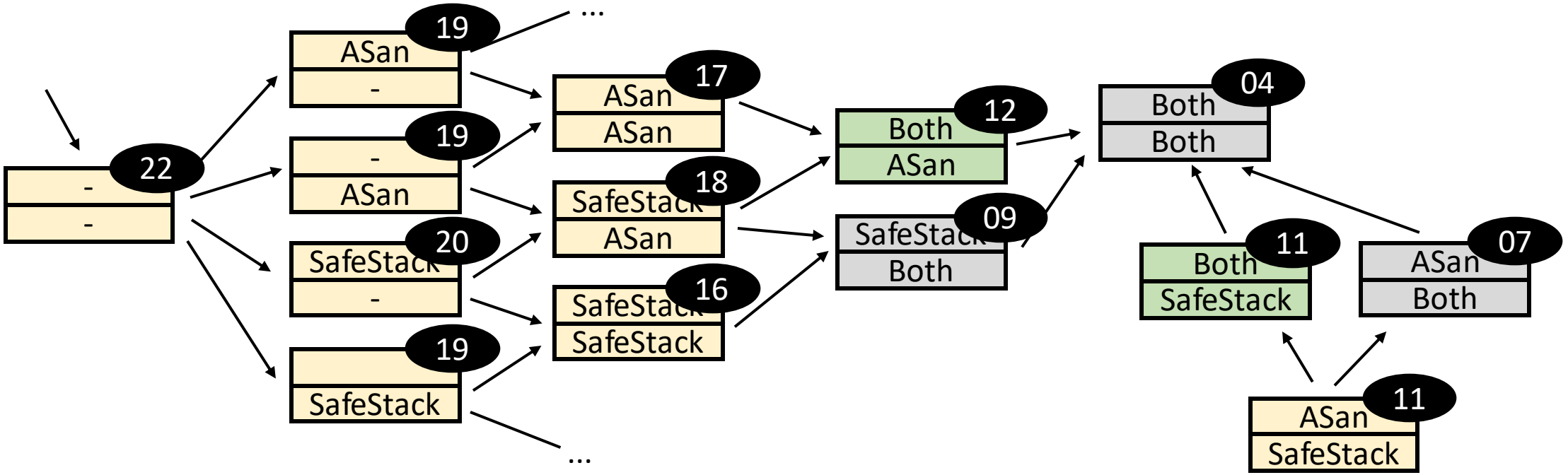
Exploring the Design Space

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



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

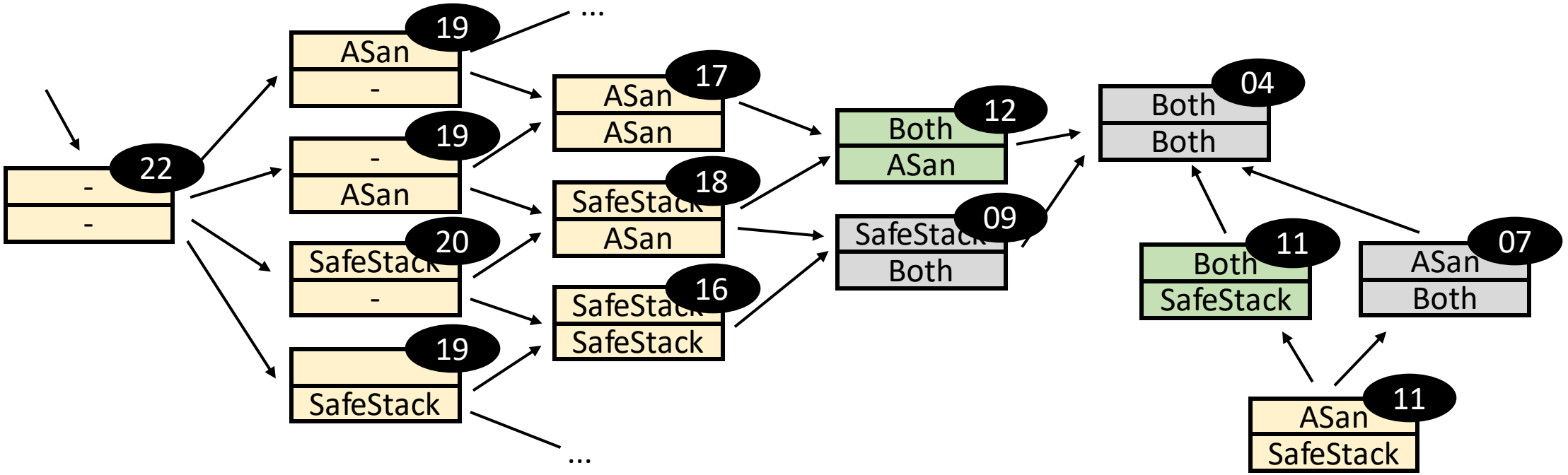
Let the user do the final choice

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

Both
ASan

Both
SafeStack

Curated list of optimal configurations



Exploring the Design Space

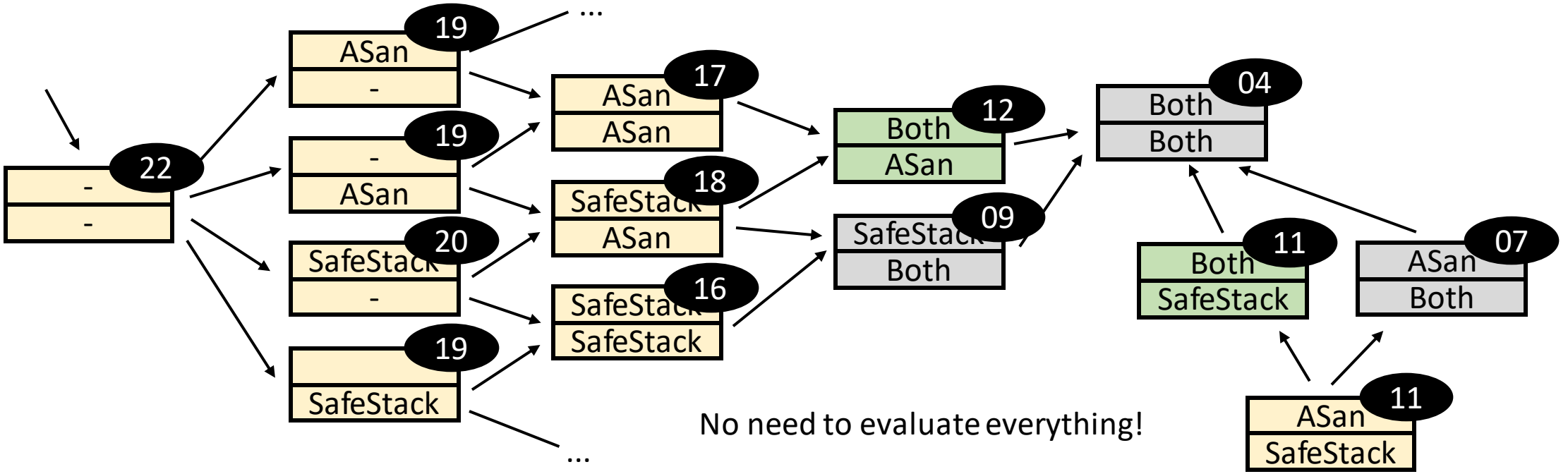
Let the user do the final choice

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

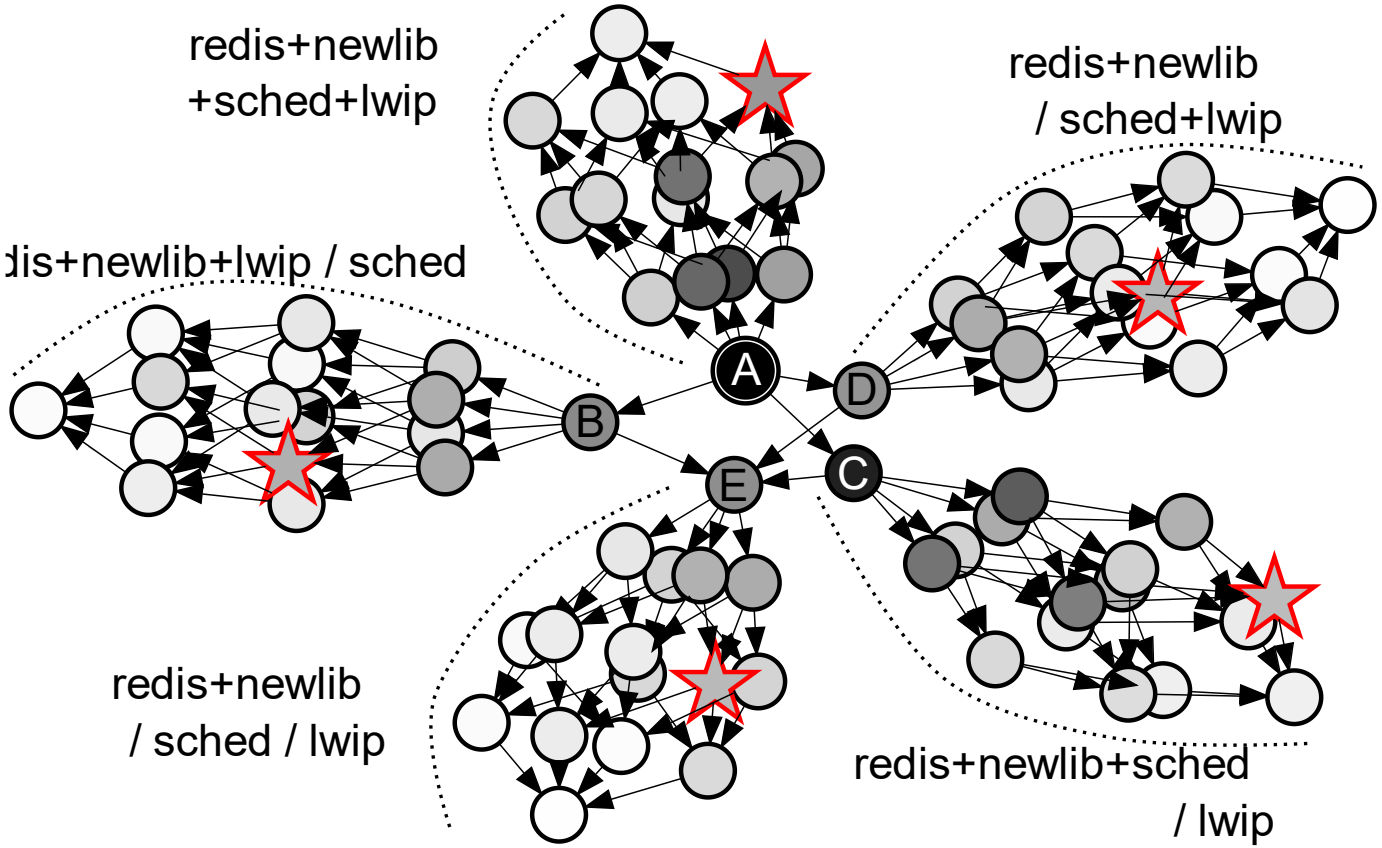
Both
ASan

Both
SafeStack

Curated list of optimal configurations

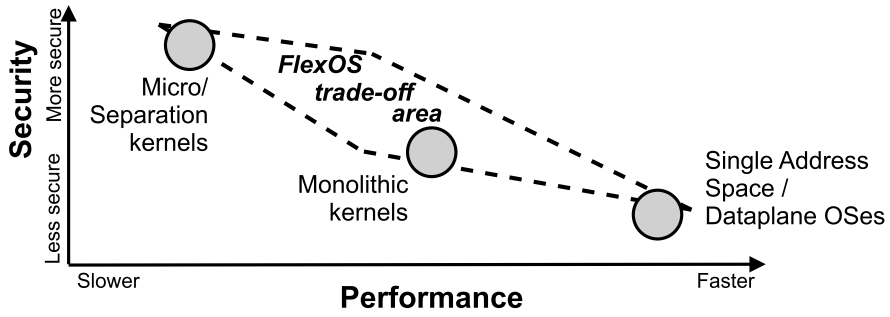


Applying POSets on Redis



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**

Paper-Related Links

Webpage: https://project-flexos.github.io/
ASPLOS'22 paper: https://owl.eu.com/papers/flexos-asplos22.pdf
Contact by e-mail: `hugo.lefeuvre@manchester.ac.uk`

Artifact Evaluation Repository: https://github.com/project-flexos/asplos22-ae
Distinguished Artifact Award!

Practical note: most of the bug reports are reported in the AE repository.
Also check known issues and doc in: https://github.com/project-flexos/unikraft

License: 3-Clause BSD License (like Unikraft)

FlexOS: Illustrating the Research Potential of Unikraft

Hugo Lefeuvre (*The University of Manchester*)
hugo.lefeuvre@manchester.ac.uk

Outline:

1. High-Level Presentation of FlexOS
2. **Technical Intro: Hello World!**
3. Technical Intro: Redis
4. Hands-On: Port Your Lib/App

Get Your FlexOS Dev Shell

- We set up 48 Docker Containers on an MPK-enabled machine at Manchester
- They contain everything we need for this workshop!

Get Your FlexOS Dev Shell

- We set up 48 Docker Containers on an MPK-enabled machine at Manchester
- They contain everything we need for this workshop!

Credentials will be given during the tutorial!

Disclaimer :-)

- This is a research prototype
- It has been written by a grad student with limited time
- It is not fit for production
- The VM/EPT backend that we are going to use is not the final one (not merged yet unfortunately)

Disclaimer :-)

- This is a research prototype
- It has been written by a grad student with limited time
- It is not fit for production
- The VM/EPT backend that we are going to use is not the final one (not merged yet unfortunately)

But:

- It works (modulo bugs)

Technical Intro: Hello World!

Recap of the FlexOS compartmentalization process:

Technical Intro: Hello World!

Recap of the FlexOS compartmentalization process:

1. Apps/Libs are ported by an expert

Technical Intro: Hello World!

Recap of the FlexOS compartmentalization process:

1. Apps/Libs are ported by an expert
2. At build time, users define a compartmentalization conformation

Technical Intro: Hello World!

Recap of the FlexOS compartmentalization process:

1. Apps/Libs are ported by an expert
2. At build time, users define a compartmentalization conformation
3. The toolchain automatically produces a matching image

Technical Intro: Hello World!

Recap of the FlexOS compartmentalization process:

1. **Apps/Libs are ported by an expert**
2. At build time, users define a compartmentalization conformation
3. The toolchain automatically produces a matching image

Technical Intro: Hello World!

Application main (slightly simplified)

```
void callback(int foo)
{
        printf("callback called!\n");
}

/* static buffer that we pass to the library */
static char static_buf[32];

/* a private static buffer */
static char static_app_secret[32];

int main(int __unused argc, char __unused *argv[])
{
        /* shared stack integer */
        int stack_int = 21;

        /* shared stack buffer */
        char stack_buf[32];

        /* call library function */
        lib_func(&callback, static_buf,
                 &stack_int, stack_buf);

        return 0;
}
```

Technical Intro: Hello World!

Application main (slightly simplified)

```
void callback(int foo)
{
        printf("callback called!\n");
}

/* static buffer that we pass to the library */
static char static_buf[32];

/* a private static buffer */
static char static_app_secret[32];

int main(int __unused argc, char __unused *argv[])
{
        /* shared stack integer */
        int stack_int = 21;

        /* shared stack buffer */
        char stack_buf[32];

        /* call library function */
        lib_func(&callback, static_buf,
                 &stack_int, stack_buf);

        return 0;
}
```

We have shared callbacks

Technical Intro: Hello World!

Application main (slightly simplified)

```
void callback(int foo)
{
        printf("callback called!\n");
}

/* static buffer that we pass to the library */
static char static_buf[32];

/* a private static buffer */
static char static_app_secret[32];

int main(int __unused argc, char __unused *argv[])
{
        /* shared stack integer */
        int stack_int = 21;

        /* shared stack buffer */
        char stack_buf[32];

        /* call library function */
        lib_func(&callback, static_buf,
                 &stack_int, stack_buf);

        return 0;
}
```

We have shared callbacks

…shared static buffers

Technical Intro: Hello World!

Application main (slightly simplified)

```
void callback(int foo)
{
        printf("callback called!\n");
}

/* static buffer that we pass to the library */
static char static_buf[32];

/* a private static buffer */
static char static_app_secret[32];

int main(int __unused argc, char __unused *argv[])
{
        /* shared stack integer */
        int stack_int = 21;

        /* shared stack buffer */
        char stack_buf[32];

        /* call library function */
        lib_func(&callback, static_buf,
                 &stack_int, stack_buf);

        return 0;
}
```

We have shared callbacks

…shared static buffers

…shared stack buffers

Technical Intro: Hello World!

Application main (slightly simplified)

```
void callback(int foo)
{
        printf("callback called!\n");
}

/* static buffer that we pass to the library */
static char static_buf[32];

/* a private static buffer */
static char static_app_secret[32];

int main(int __unused argc, char __unused *argv[])
{
        /* shared stack integer */
        int stack_int = 21;

        /* shared stack buffer */
        char stack_buf[32];

        /* call library function */
        lib_func(&callback, static_buf,
                 &stack_int, stack_buf);

        return 0;
}
```

We have shared callbacks

…shared static buffers

…shared stack buffers

…and a library call

Technical Intro: Hello World!

Let's run it! (without isolation for now)

```
$ ssh $(your container)
# cd /root/.unikraft/apps/flexos-example
# cp kraft.yaml.fcalls kraft.yaml
# kraft configure
…
```

Why is `--initrd` needed? https://github.com/project-flexos/asplos22-ae/issues/1

Technical Intro: Hello World!

Let's run it! (without isolation for now)

```
$ ssh $(your container)
# cd /root/.unikraft/apps/flexos-example
# cp kraft.yaml.fcalls kraft.yaml
# kraft configure
…
# make prepare && kraft -v build --no-progress --fast --compartmentalize
…
```

Why is `--initrd` needed? https://github.com/project-flexos/asplos22-ae/issues/1

Technical Intro: Hello World!

Let's run it! (without isolation for now)

```
$ ssh $(your container)
# cd /root/.unikraft/apps/flexos-example
# cp kraft.yaml.fcalls kraft.yaml
# kraft configure
…
# make prepare && kraft -v build --no-progress --fast --compartmentalize
…
# kraft run --initrd /root/img.cpio -M 200
SeaBIOS (version 1.12.0-1)
Booting from ROM..[    0.000000] ERR:  [libkvmplat] <mm.c @  190> ...
callback called!
```

Why is `--initrd` needed? https://github.com/project-flexos/asplos22-ae/issues/1

Technical Intro: Hello World!

Recap of the FlexOS compartmentalization process:

1. Apps/Libs are ported by an expert
2. **At build time, users define a compartmentalization conformation**
3. The toolchain automatically produces a matching image

Technical Intro: Hello World!

Let's run it with isolation?

```
$ ssh $(your container)
# cd /root/.unikraft/apps/flexos-example
# kraftcleanup
# cp kraft.yaml.mpk kraft.yaml
# rm .config && kraft configure
…
```

`kraftcleanup` removes all stale source transformations

Technical Intro: Hello World!

Let's run it with isolation?

```
$ ssh $(your container)
# cd /root/.unikraft/apps/flexos-example
# kraftcleanup
# cp kraft.yaml.mpk kraft.yaml
# rm .config && kraft configure
…
# make prepare && kraft -v build --no-progress --fast --compartmentalize
…
```

`kraftcleanup` removes all stale source transformations

Technical Intro: Hello World!

Let's run it with isolation?

```
$ ssh $(your container)
# cd /root/.unikraft/apps/flexos-example
# kraftcleanup
# cp kraft.yaml.mpk kraft.yaml
# rm .config && kraft configure
…
# make prepare && kraft -v build --no-progress --fast --compartmentalize
…
# kraft run --initrd /root/img.cpio -M 200
…
```

`kraftcleanup` removes all stale source transformations

Technical Intro: Hello World!

Let's run it with isolation?

```
$ ssh $(your container)
# cd /root/.unikraft/apps/flexos-example
# kraftcleanup
# cp kraft.yaml.mpk kraft.yaml
# rm .config && kraft configure
…
# make prepare && kraft -v build --no-progress --fast --compartmentalize
…
# kraft run --initrd /root/img.cpio -M 200
…
```

`kraftcleanup` removes all stale source transformations

We need to port it :-)

Technical Intro: Hello World Porting

Application main (slightly simplified)

```
void callback(int foo)
{
        printf("callback called!\n");
}

/* static buffer that we pass to the library */
static char static_buf[32];

/* a private static buffer */
static char static_app_secret[32];

int main(int __unused argc, char __unused *argv[])
{
        /* shared stack integer */
        int stack_int = 21;

        /* shared stack buffer */
        char stack_buf[32];

        /* call library function */
        lib_func(&callback, static_buf,
                 &stack_int, stack_buf);

        return 0;
}
```

Technical Intro: Hello World Porting

Application main (slightly simplified)

```
__attribute__((libflexosexample_callback))
void callback(int foo)
{
        printf("callback called!\n");
}

/* static buffer that we pass to the library */
static char static_buf[32];

/* a private static buffer */
static char static_app_secret[32];

int main(int __unused argc, char __unused *argv[])
{
        /* shared stack integer */
        int stack_int = 21;

        /* shared stack buffer */
        char stack_buf[32];

        /* call library function */
        lib_func(&callback, static_buf,
                 &stack_int, stack_buf);

        return 0;
}
```

Annotate callbacks

Technical Intro: Hello World Porting

Application main (slightly simplified)

```
__attribute__((libflexosexample_callback))
void callback(int foo)
{
        printf("callback called!\n");
}

/* static buffer that we pass to the library */
static char static_buf[32] __attribute__((flexos_whitelist));

/* a private static buffer */
static char static_app_secret[32];

int main(int __unused argc, char __unused *argv[])
{
        /* shared stack integer */
        int stack_int = 21;

        /* shared stack buffer */
        char stack_buf[32];

        /* call library function */
        lib_func(&callback, static_buf,
                 &stack_int, stack_buf);

        return 0;
}
```

Annotate callbacks

Annotate static shared buffers

Technical Intro: Hello World Porting

Application main (slightly simplified)

```
__attribute__((libflexosexample_callback))
void callback(int foo)
{
        printf("callback called!\n");
}

/* static buffer that we pass to the library */
static char static_buf[32] __attribute__((flexos_whitelist));

/* a private static buffer */
static char static_app_secret[32];

int main(int __unused argc, char __unused *argv[])
{
        /* shared stack integer */
        int stack_int __attribute__((flexos_whitelist)) = 21;

        /* shared stack buffer */
        char stack_buf[32] __attribute__((flexos_whitelist));

        /* call library function */
        lib_func(&callback, static_buf,
                 &stack_int, stack_buf);

        return 0;
}
```

Annotate callbacks

Annotate static shared buffers

Annotate stack shared buffers

Technical Intro: Hello World Porting

Application main (slightly simplified)

```
__attribute__((libflexosexample_callback))
void callback(int foo)
{
        printf("callback called!\n");
}

/* static buffer that we pass to the library */
static char static_buf[32] __attribute__((flexos_whitelist));

/* a private static buffer */
static char static_app_secret[32];

int main(int __unused argc, char __unused *argv[])
{
        /* shared stack integer */
        int stack_int __attribute__((flexos_whitelist)) = 21;

        /* shared stack buffer */
        char stack_buf[32] __attribute__((flexos_whitelist));

        /* call library function */
        flexos_gate(libflexosexample, lib_func, &callback, static_buf,
                &stack_int, stack_buf);

        return 0;
}
```

Annotate callbacks

Annotate static shared buffers

Annotate stack shared buffers

Add gate placeholders

Technical Intro: Hello World!

Let's run it with isolation! (MPK)

```
$ ssh $(your container)
# cd /root/.unikraft/apps/flexos-example
# kraftcleanup
# git checkout lyon-workshop-ported
# cp kraft.yaml.mpk kraft.yaml
# rm .config && kraft configure
…
# make prepare && kraft -v build --no-progress --fast --compartmentalize
…
# kraft run --initrd /root/img.cpio -M 200
SeaBIOS (version 1.12.0-1)
Booting from ROM..[    0.000000] ERR:  [libkvmplat] <mm.c @  190> ...
callback called!
```

This branch = `lyon-workshop`, ported like in the previous slide

Technical Intro: Hello World Porting

Insert gates using our porting helpers:

```
$ ssh $(your container)
…
# kraftcleanup
# cd /root/.unikraft
# ./porthelper.sh apps/flexos-example/main.c
```

Technical Intro: Hello World Porting

Insert gates using our porting helpers:

```
$ ssh $(your container)
…
# kraftcleanup
# cd /root/.unikraft
# ./porthelper.sh apps/flexos-example/main.c
```

Does a pretty good job, but no rocket science. Does not handle shared data.

The true solution is in the compiler, and that's not a contribution of this paper.

(see PtrSplit @CCS'17, Cali @AsiaCCS'21, etc.)

Technical Intro: Hello World!

Let's run it with isolation! (EPT)

```
$ ssh $(your container)
# cd /root/.unikraft/apps/flexos-example
# kraftcleanup
# git checkout lyon-workshop-ported
# cp kraft.yaml.ept kraft.yaml
# rm .config && kraft configure
…
# make prepare && kraft -v build --fast --compartmentalize
…
# /root/.unikraft/run-ept.sh run build/flexos-example_kvm-x86_64
SeaBIOS (version 1.12.0-1)
Booting from ROM..[    0.000000] ERR:  [libkvmplat] <mm.c @  190> ...
callback called!
```

Do not put `--no-progress` here, this triggers a bug in the toolchain for EPT
https://github.com/project-flexos/asplos22-ae/issues/2

Do not use `kraft run` here; the integration has not been merged yet…
https://github.com/project-flexos/asplos22-ae/issues/3

Technical Intro: Hello World!

Recap of the FlexOS compartmentalization process:

1. Apps/Libs are ported by an expert
2. At build time, users define a compartmentalization conformation
3. **The toolchain automatically produces a matching image**

How to inspect the transformations performed by FlexOS?

How to debug FlexOS compartmentalization issues?

How can I peek at FlexOS' internals?

Technical Intro: Transformations

How do the transformations look?

Benefits of source transformations: use `git diff` and read `patch` output :-)

```
$ ssh $(your container)
# cd /root/.unikraft/apps/flexos-example
```

Technical Intro: Transformations

How do the transformations look?

Benefits of source transformations: use `git diff` and read `patch` output :-)

```
$ ssh $(your container)
# cd /root/.unikraft/apps/flexos-example && git diff
diff --git a/main.c b/main.c
index df230c3..5f4ba6d 100644
--- a/main.c
+++ b/main.c
@@ -34,20 +34,26 @@
… <snip>
@@ -58,15 +64,18 @@ int main(int __unused argc, char __unused *argv[])
     static_app_secret[0] = 'B';

     /* shared stack integer used by the library */
-    int stack_int __attribute__((flexos_whitelist)) = 21;
+    int * stack_int = uk_malloc(flexos_shared_alloc, sizeof(int));
+    *stack_int = 21;
… <snip>
}
```

Technical Intro: Debugging

How do you debug porting issues?

Technical Intro: Deb

How do you debug porting issues?

```
# say we forgot some shared data
        static_app_secret[0] = 'B';

        /* shared stack integer used by the library */
-       int stack_int __attribute__((flexos_whitelist)) = 21;
+       int stack_int = 21;

        /* shared stack buffer that we pass to the library */
        char stack_buf[32] __attribute__((flexos_whitelist));
```

Technical Intro: Deb

How do you debug porting issues?

```
# say we forgot some shared data
        static_app_secret[0] = 'B';

        /* shared stack integer used by the library */
-       int stack_int __attribute__((flexos_whitelist)) = 21;
+       int stack_int = 21;
```

```
$ ssh $(your container)
# …
# kraft run …
```


```
[    0.100770] CRIT: [libkvmplat] <traps.c @  198> Page fault at linear address 4000dff9c, rip 1962ac, regs 0x1dff50, sp
40011ffd0, our_sp 0x1dfee8, code 23
[    0.106650] CRIT: [libkvmplat] <traps.c @  198> PF_PK: protection key block access (WRITE)
[    0.110337] CRIT: [libkvmplat] <traps.c @  198> Target page 0x4000dff9c (section .heap) had key 0
[    0.114256] CRIT: [libkvmplat] <trace.c @  198> RIP: 00000000001962ac CS: 0008
[    0.117436] CRIT: [libkvmplat] <trace.c @  198> RSP: 000000040011ffd0 SS: 0010 EFLAGS: 00010202
[    0.121292] CRIT: [libkvmplat] <trace.c @  198> RAX: 0000000000108100 RBX: 000000040a001cc8 RCX: 000000040a001cc8
[    0.125842] CRIT: [libkvmplat] <trace.c @  198> RDX: 00000004000dff9c RSI: 0000000000105320 RDI: 0000000000000054
[    0.130382] CRIT: [libkvmplat] <trace.c @  198> RBP: 000000040011ffd0 R08: 000000040a001ca8 R09: 00000000fffffffe
[    0.134938] CRIT: [libkvmplat] <trace.c @  198> R10: 000000040a001ce8 R11: 00000000001b5330 R12: 00000004000dff9c
[    0.139488] CRIT: [libkvmplat] <trace.c @  198> R13: 000000040a001cc8 R14: 0000000005f66fa5 R15: 0000000000000001
[    0.144041] CRIT: [libkvmplat] <traps.c @  198> PKU: 000000003ffffff3
[    0.146893] CRIT: [libkvmplat] <trace.c @  198> base is 0x40011ffd0 caller is 0x1963f6
[    0.150432] CRIT: [libkvmplat] <trace.c @  198> base is 0x40011ffe8 caller is 0
```

Technical Intro: Deb

How do you debug porting issues?

```
# say we forgot some shared data
        static_app_secret[0] = 'B';

        /* shared stack integer used by the library */
-       int stack_int __attribute__((flexos_whitelist)) = 21;
+       int stack_int = 21;
```

```
$ ssh $(your container)
# …
# kraft run …
[    0.100770] CRIT: [libkvmplat] <traps.c @  198> Page fault at linear address 4000dff9c, rip 1962ac, regs 0x1dff50, sp
40011ffd0, our_sp 0x1dfee8, code 23
[    0.106650] CRIT: [libkvmplat] <traps.c @  198> PF_PK: protection key block access (WRITE)
[    0.110337] CRIT: [libkvmplat] <traps.c @  198> Target page 0x4000dff9c (section .heap) had key 0
[    0.114256] CRIT: [libkvmplat] <trace.c @  198> RIP: 00000000001962ac CS: 0008
[    0.117436] CRIT: [libkvmplat] <trace.c @  198> RSP: 000000040011ffd0 SS: 0010 EFLAGS: 00010202
[    0.121292] CRIT: [libkvmplat] <trace.c @  198> RAX: 0000000000108100 RBX: 000000040a001cc8 RCX: 000000040a001cc8
[    0.125842] CRIT: [libkvmplat] <trace.c @  198> RDX: 00000004000dff9c RSI: 0000000000105320 RDI: 0000000000000054
[    0.130382] CRIT: [libkvmplat] <trace.c @  198> RBP: 000000040011ffd0 R08: 000000040a001ca8 R09: 00000000fffffffe
[    0.134938] CRIT: [libkvmplat] <trace.c @  198> R10: 000000040a001ce8 R11: 00000000001b5330 R12: 00000004000dff9c
[    0.139488] CRIT: [libkvmplat] <trace.c @  198> R13: 000000040a001cc8 R14: 0000000005f66fa5 R15: 0000000000000001
[    0.144041] CRIT: [libkvmplat] <traps.c @  198> PKU: 000000003ffffff3
[    0.146893] CRIT: [libkvmplat] <trace.c @  198> base is 0x40011ffd0 caller is 0x1963f6
[    0.150432] CRIT: [libkvmplat] <trace.c @  198> base is 0x40011ffe8 caller is 0
```

Unikraft 0.5 (on which we based FlexOS) did not have symbolized stack traces…

Technical Intro: Deb

How do yo

```
# say we forgot some shared data
        static_app_secret[0] = 'B';
```

```
# addr2line 0x0000000001962ac -e build/flexos-example_kvm-x86_64.dbg
/root/.unikraft/libs/flexos-example/isolated.c:42
```

```
$ ssh $
# …
# kraft
[    0.10077
40011ffd0, c
[    0.10665
[    0.11033
[    0.11425
[    0.11743
[    0.12129
[    0.12584
[    0.13038
[    0.13493
[    0.139488] CRIT: [libkvmplat] <trace.c @  198> R13: 000000040a001cc8 R14: 000000000005f66fa5 R15: 0000000000000001
[    0.144041] CRIT: [libkvmplat] <traps.c @  198> PKU: 000000003ffffff3
[    0.146893] CRIT: [libkvmplat] <trace.c @  198> base is 0x40011ffd0 caller is 0x1963f6
[    0.150432] CRIT: [libkvmplat] <trace.c @  198> base is 0x40011ffe8 caller is 0
```

Unikraft 0.5 (on which we based FlexOS) did not have symbolized stack traces…

Technical Intro: Deb

How do yo

```
# say we forgot some shared data
        static_app_secret[0] = 'B';
```

```
# addr2line 0x0000000001962ac -e build/flexos-example_kvm-x86_64.dbg
/root/.unikraft/libs/flexos-example/isolated.c:42
# cat /root/.unikraft/libs/flexos-example/isolated.c
...
    36    void lib_func(void (*callback)(int), char *static_buf,
    37         int *stack_int, char *stack_buf)
    38    {
    39         /* use all arguments */
    40         *static_buf = '\0';
    41         *stack_buf = '\0';
    42         *stack_int = 42;
    43         callback(84);
    44    }
...
```

```
$ ssh $
# …
# kraft
[    0.1007
40011ffd0, c
[    0.1066
[    0.1103
[    0.1142
[    0.1174
[    0.1212
[    0.1258
[    0.1303
[    0.1349
[    0.1394
[    0.1440
[    0.1468
[    0.150432] CRIT: [libkvmplat] <trace.c @  198> base is 0x40011ffe8 caller is 0
```

Unikraft 0.5 (on which we based FlexOS) did not have symbolized stack traces…

FlexOS: Illustrating the Research Potential of Unikraft

Hugo Lefeuvre (*The University of Manchester*)
hugo.lefeuvre@manchester.ac.uk

Outline:

1. High-Level Presentation of FlexOS
2. Technical Intro: Hello World!
3. **Hands-On: Port Your Lib/App**

Hands-On: Port Your Lib/App

Stuff that we can try porting/running in isolation together:

- Libzlib + a zlib example
  - Library: https://github.com/project-flexos/lib-zlib
  - Application: https://github.com/project-flexos/app-zlib-example

(or whatever you want that runs on Unikraft)

Hands-On: Port Your Lib/App

Stuff that we can try porting/running in isolation together:

- Libzlib + a zlib example
  - Library: https://github.com/project-flexos/lib-zlib
  - Application: https://github.com/project-flexos/app-zlib-example

(or whatever you want that runs on Unikraft)

(how to run it)

```
$ ssh $(your container)
# cd /root/.unikraft/apps/zlib-example && kraftcleanup
# kraft configure
…
# make prepare && kraft -v build --no-progress --fast --compartmentalize
…
# kraft run --initrd ./zlib.cpio -M 200
SeaBIOS (version 1.12.0-1)
Booting from ROM..[    0.000000] ERR:  [libkvmplat] <mm.c @  190> ...
...
```