

# FlexOS: Towards Flexible OS Isolation

**Hugo Lefeuvre**<sup>1</sup>, Vlad-Andrei Bădoiu<sup>2</sup>, Alexander Jung<sup>3,4</sup>, Stefan Teodorescu<sup>2</sup>,  
Sebastian Rauch<sup>5</sup>, Felipe Huici<sup>6,4</sup>, Costin Raiciu<sup>2,7</sup>, Pierre Olivier<sup>1</sup>

*<sup>1</sup>The University of Manchester, <sup>2</sup>Politehnica Bucharest, <sup>3</sup>Lancaster University, <sup>4</sup>Unikraft.io,  
<sup>5</sup>Karlsruhe Institute of Technology, <sup>6</sup>NEC Labs Europe, <sup>7</sup>Correct Networks*

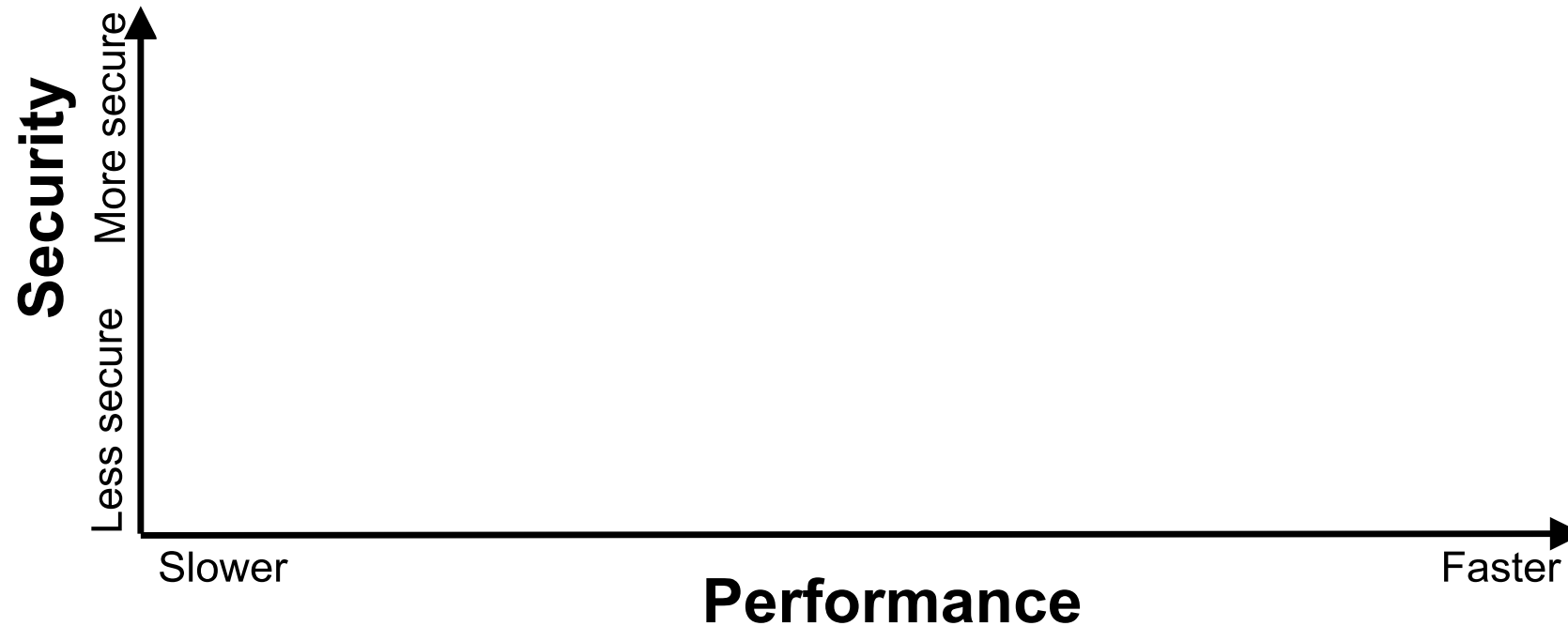
FGBS'22, 17<sup>th</sup> - 18<sup>th</sup> March 2022



# Current OS Designs

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

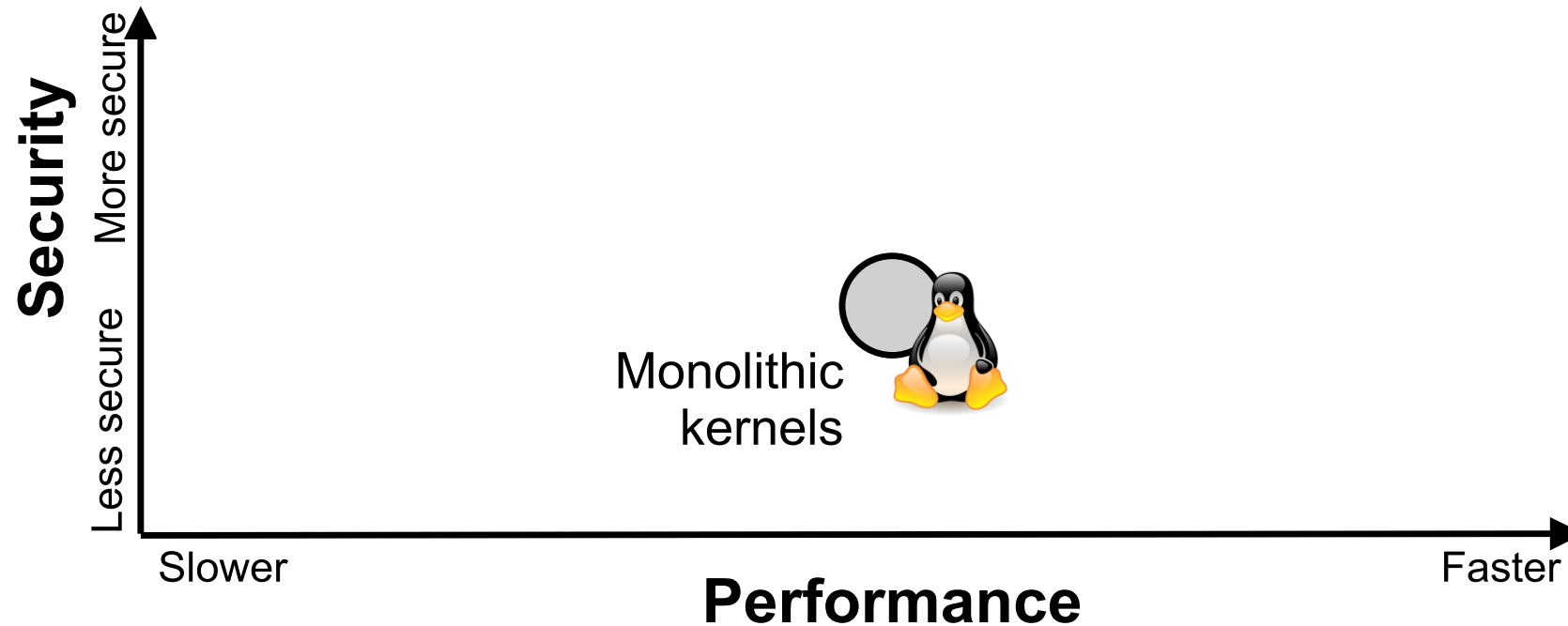
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



# Current OS Designs

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

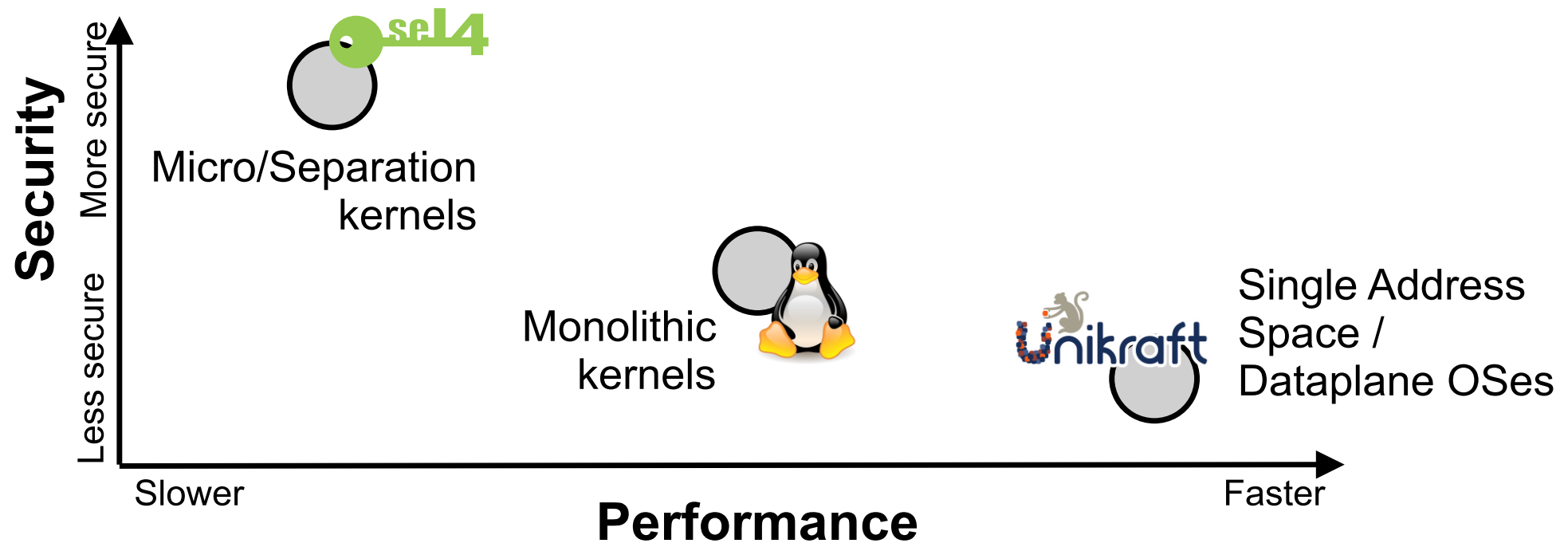
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



# Current OS Designs

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

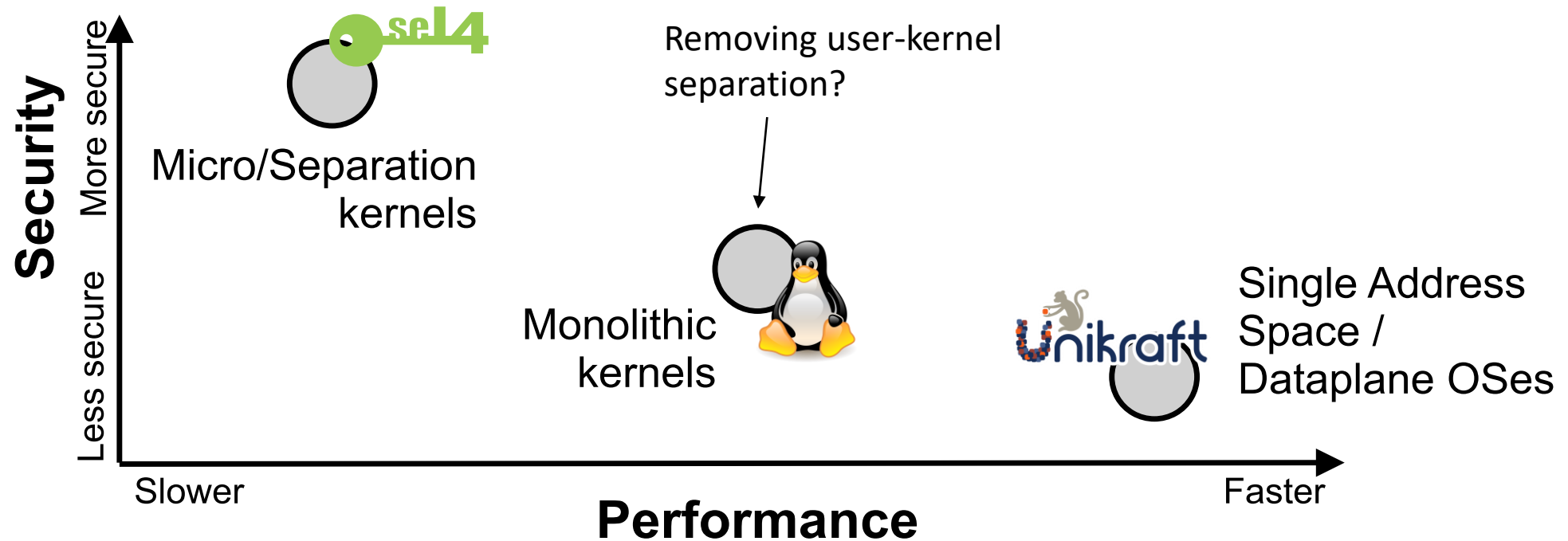
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



# Current OS Designs

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

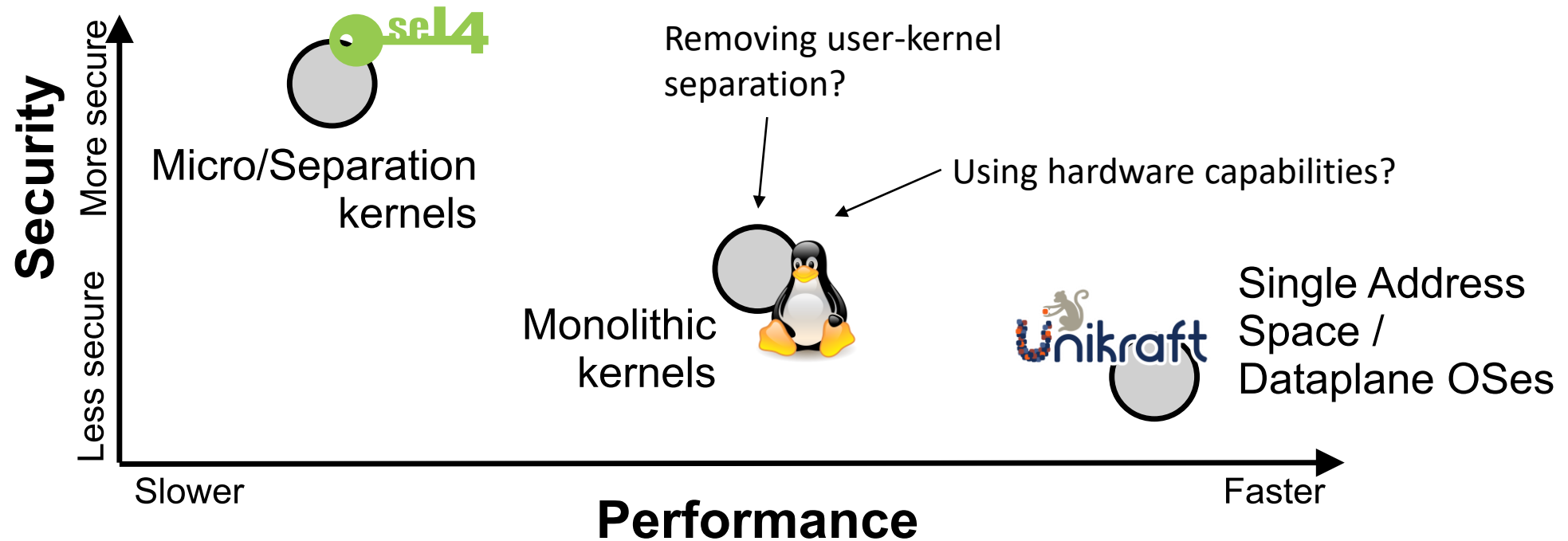
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



# Current OS Designs

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

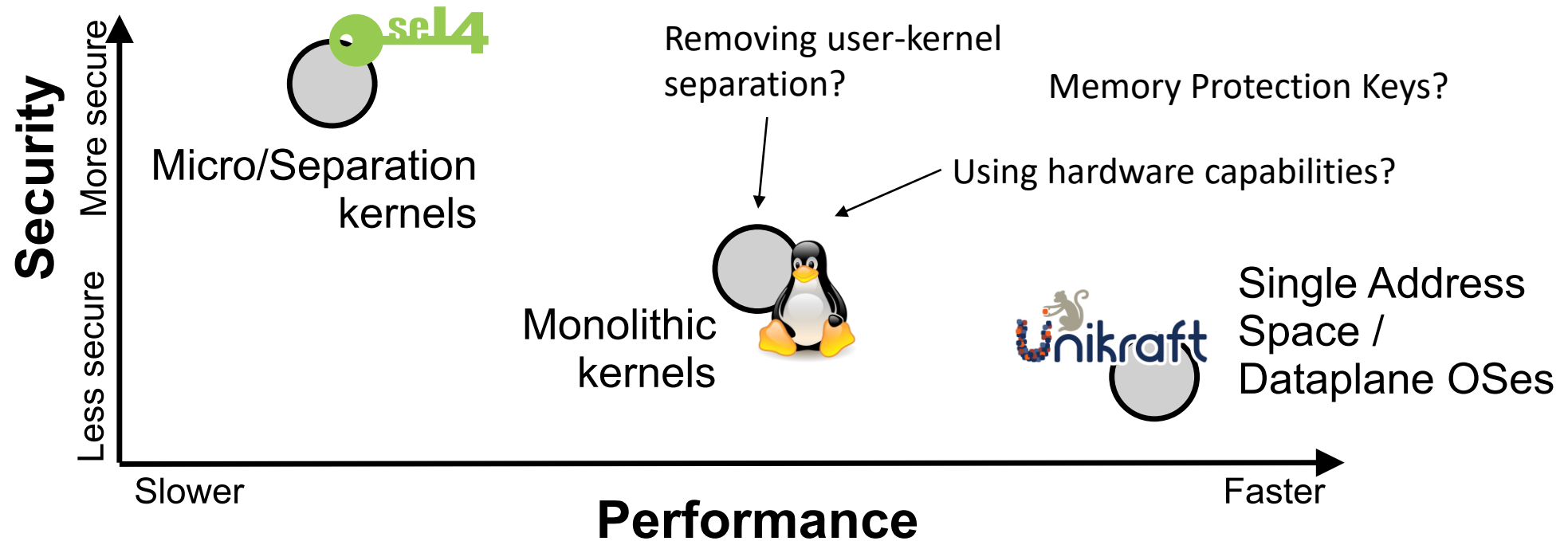
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



# Current OS Designs

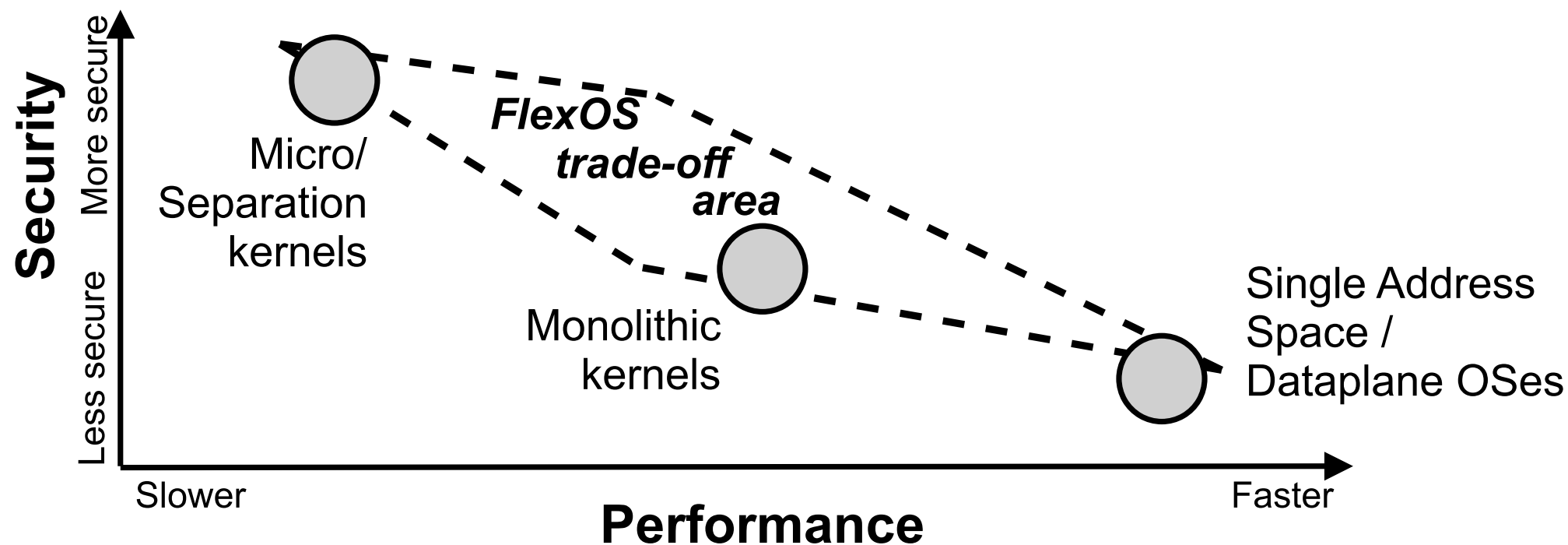
OS security/isolation strategies are **fixed** at design time!

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



# FlexOS: Flexible Isolation

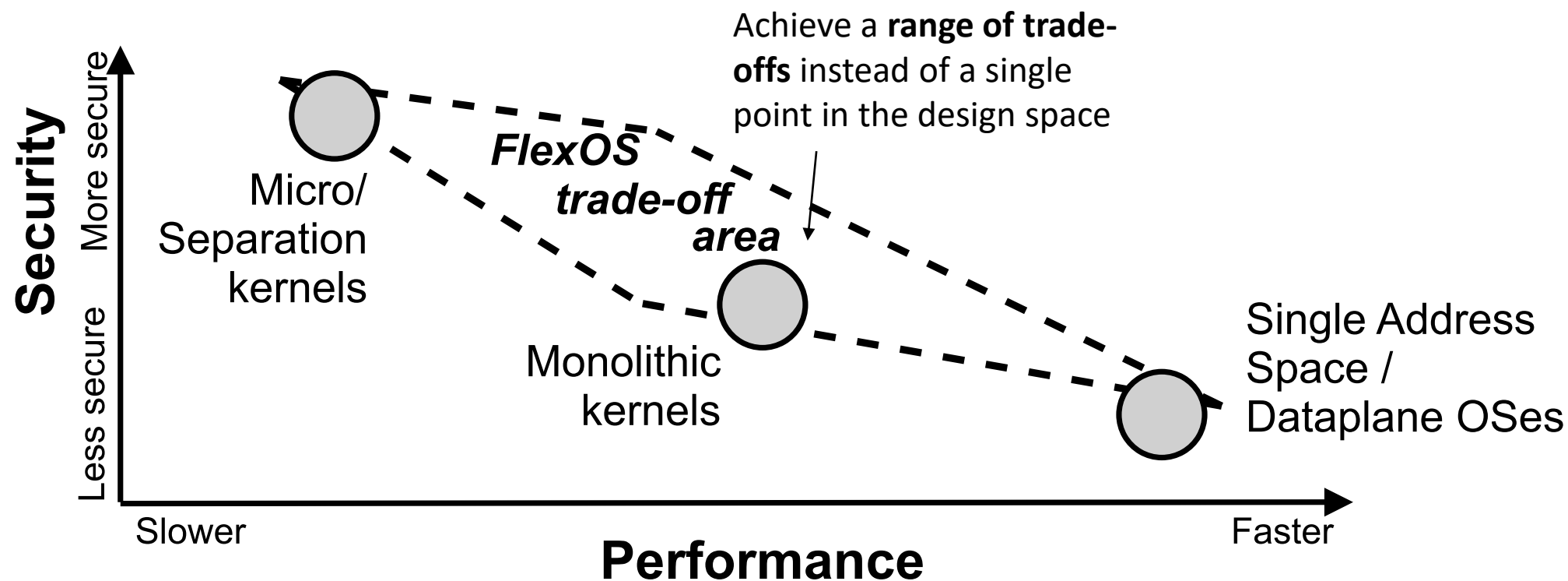
**Decouple security/isolation decisions from the OS design**





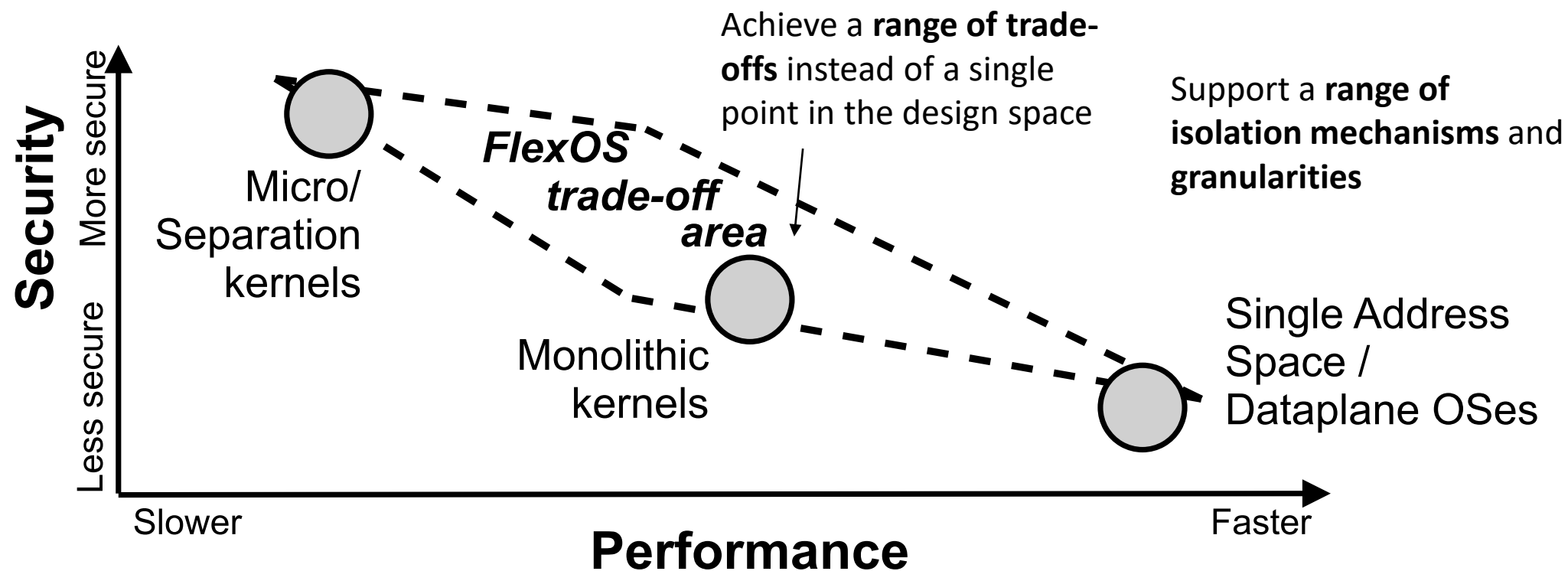
# FlexOS: Flexible Isolation

**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

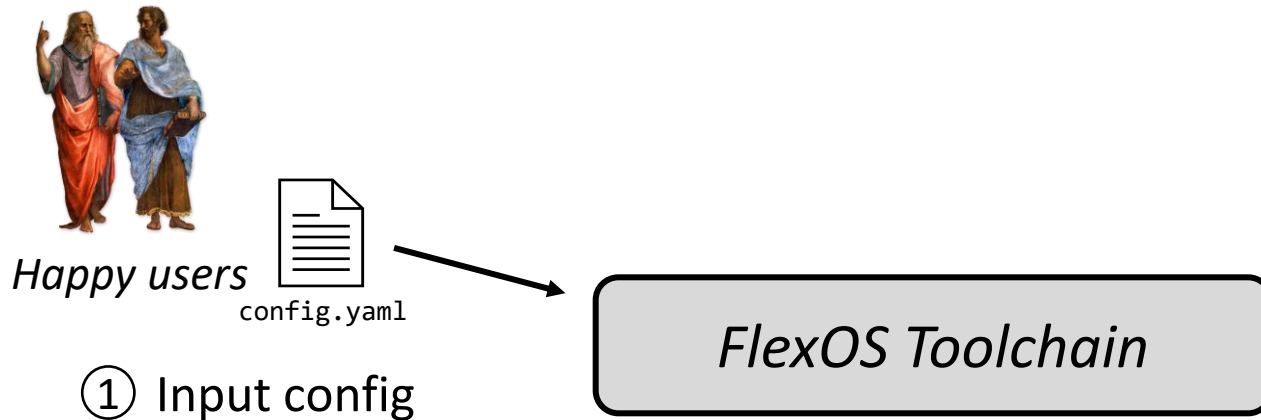
Flexibility must not **get into the way of performance**

4

# FlexOS 101: Overview



# FlexOS 101: Overview

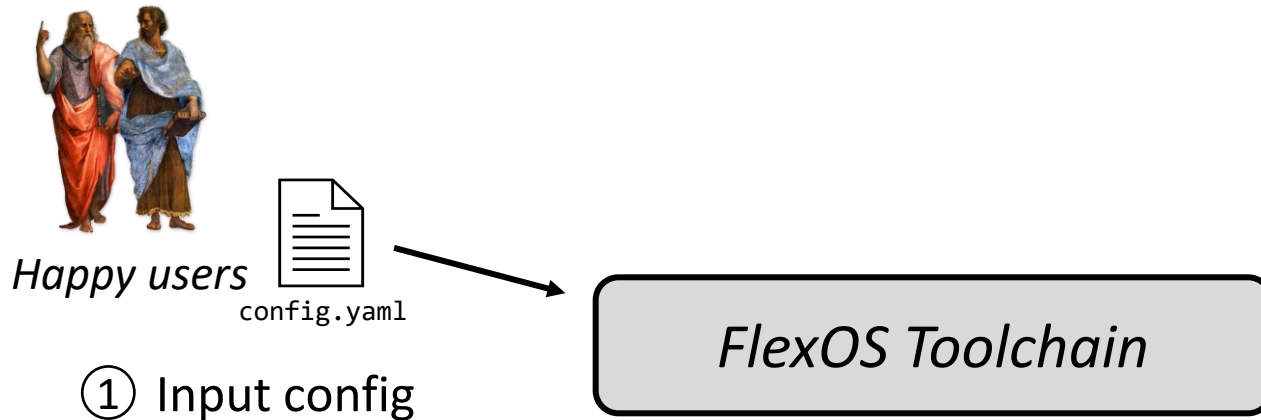


config.yaml

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

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

# FlexOS 101: Overview

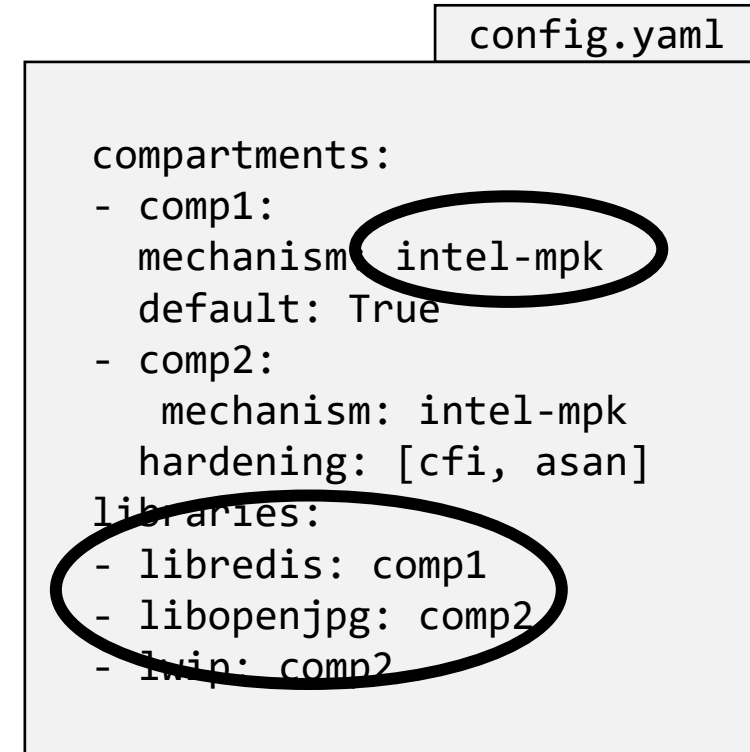
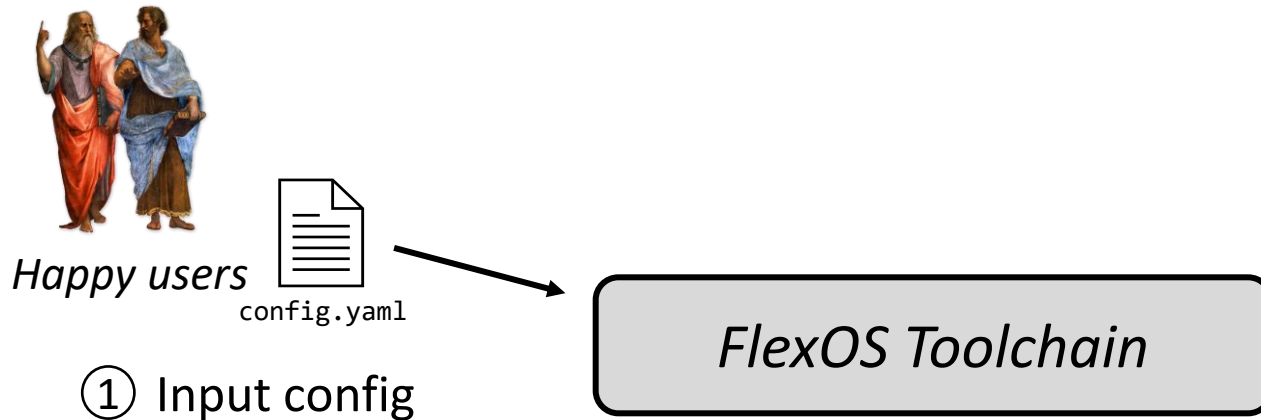


```
config.yaml

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

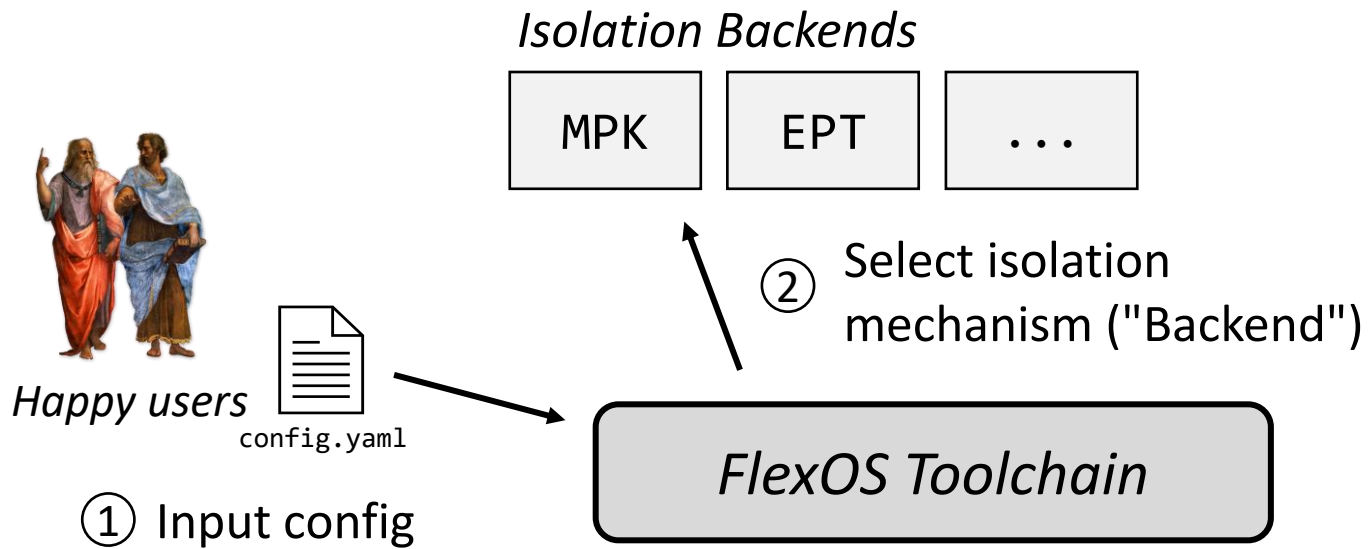
*"Redis image with two compartments,  
isolate libopenjpeg and lwip together"*

# FlexOS 101: Overview

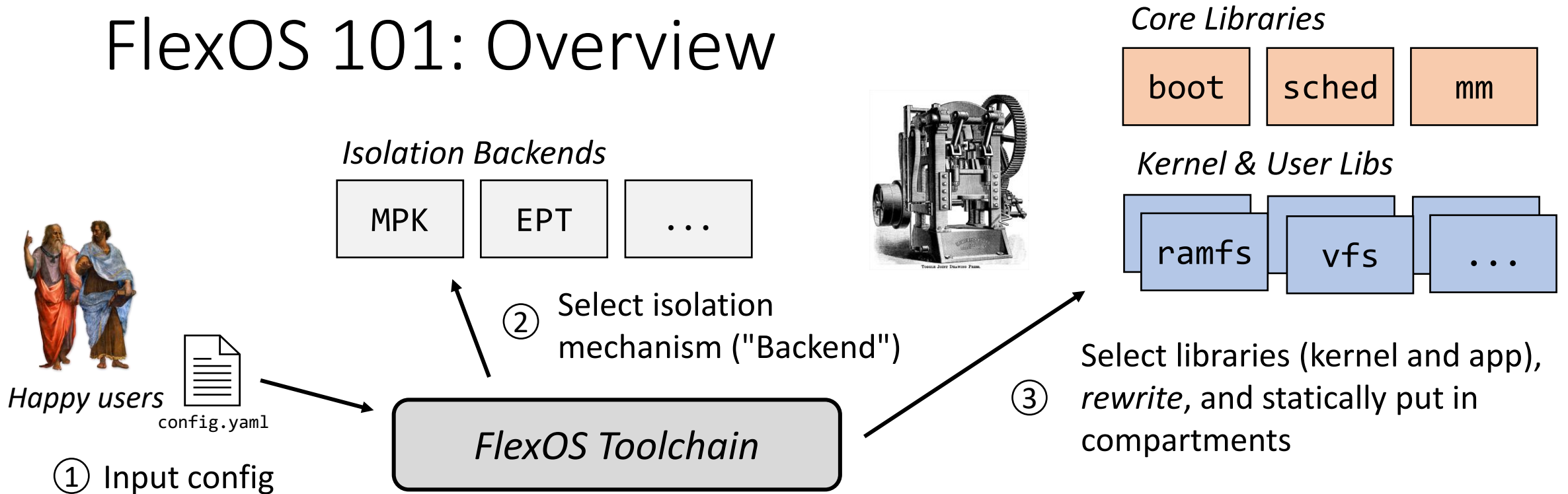


*"Redis image with two compartments,  
isolate libopenjpeg and lwip together"*

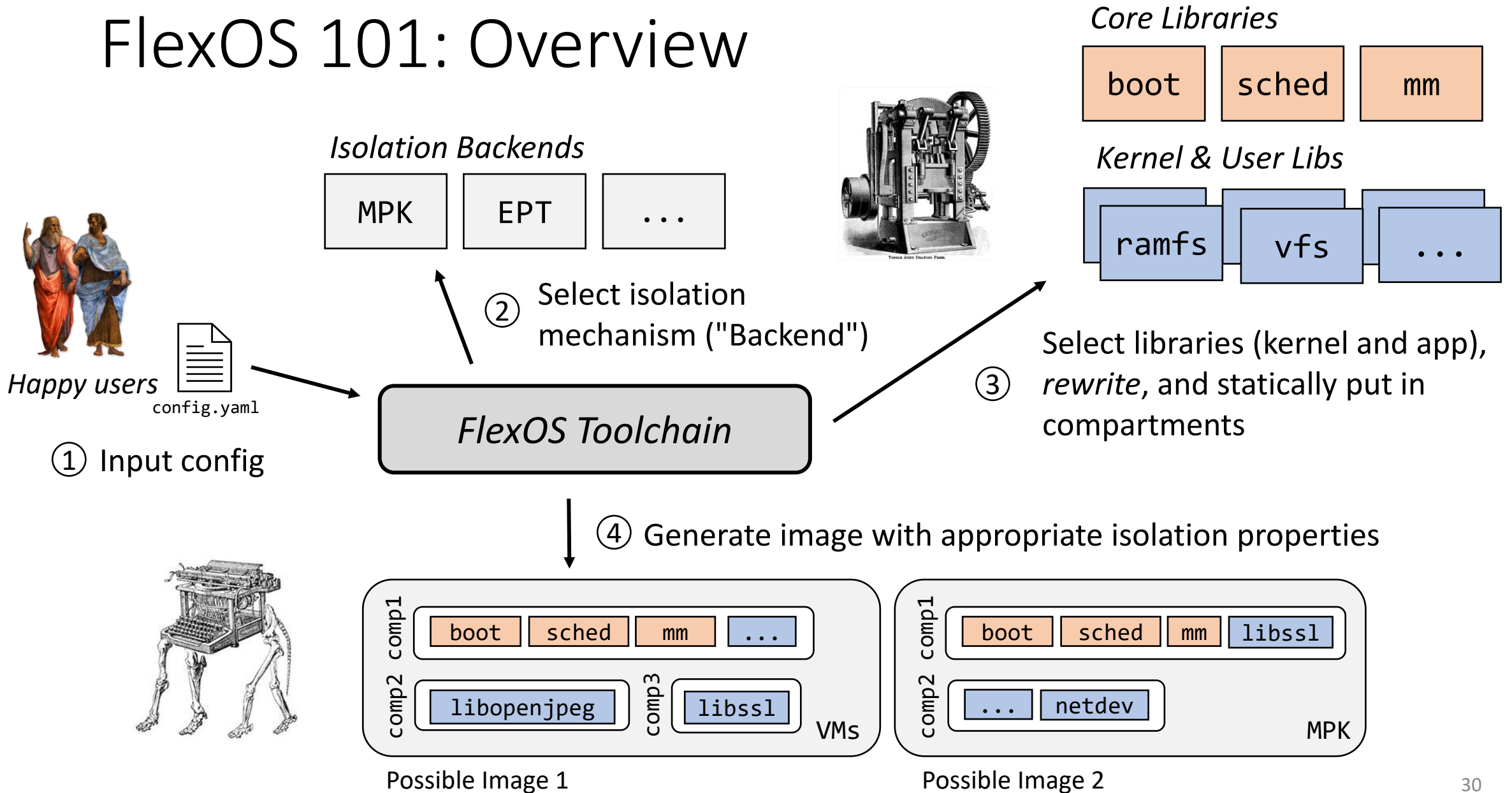
# FlexOS 101: Overview



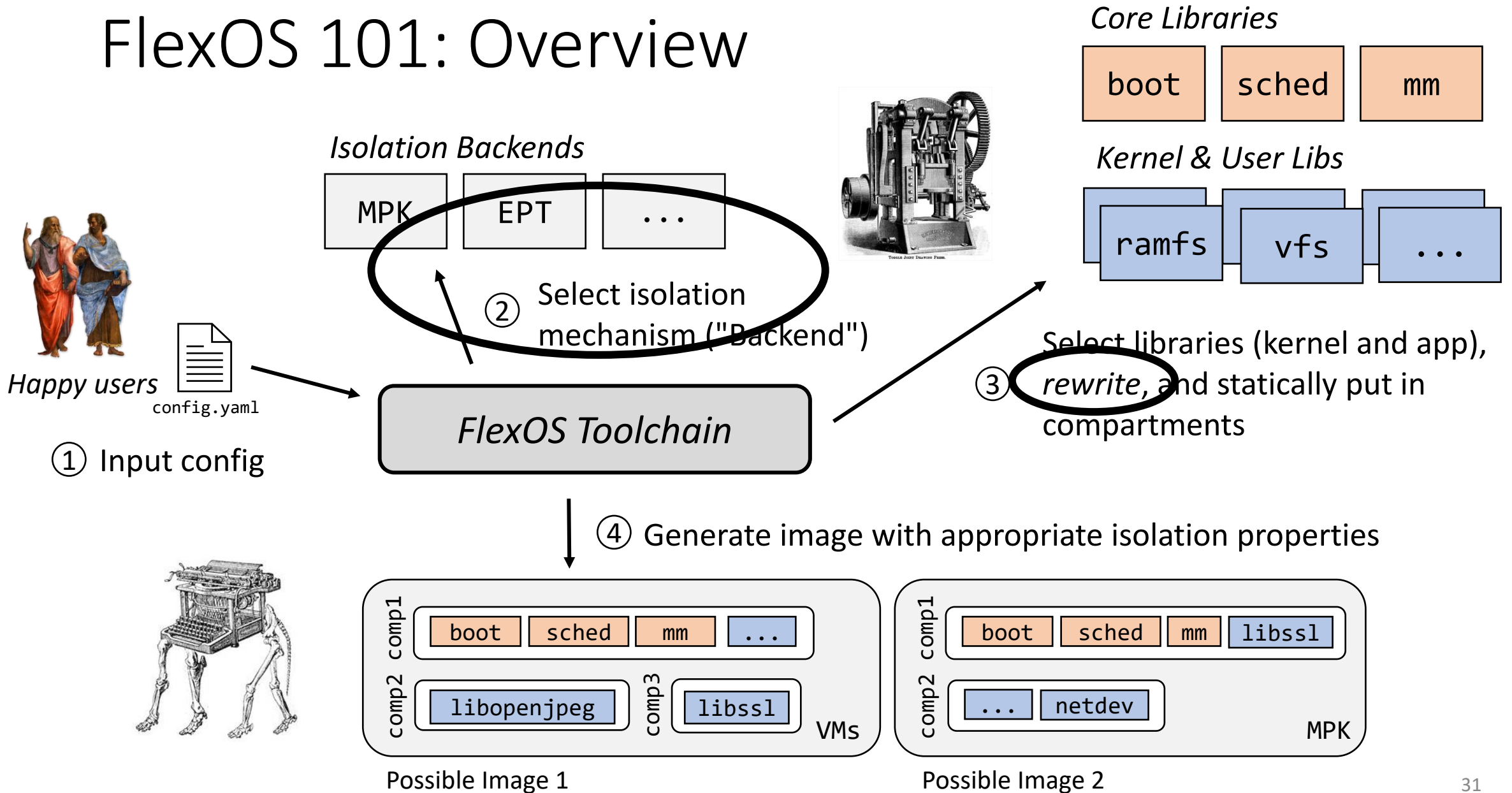
# FlexOS 101: Overview



# FlexOS 101: Overview



# FlexOS 101: Overview



# FlexOS 101: Mechanism Abstraction

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





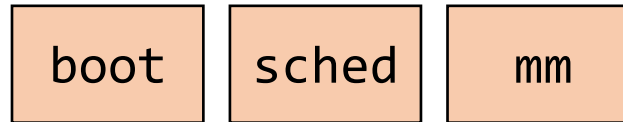
# FlexOS 101: Mechanism Abstraction

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

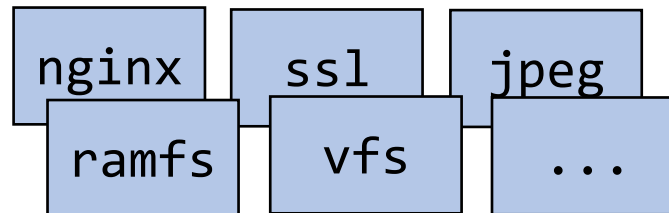


Such libOSes are composed  
of *fine-granular*,  
*independent* libraries

## *Core Libraries*



## *Kernel & User Libs*



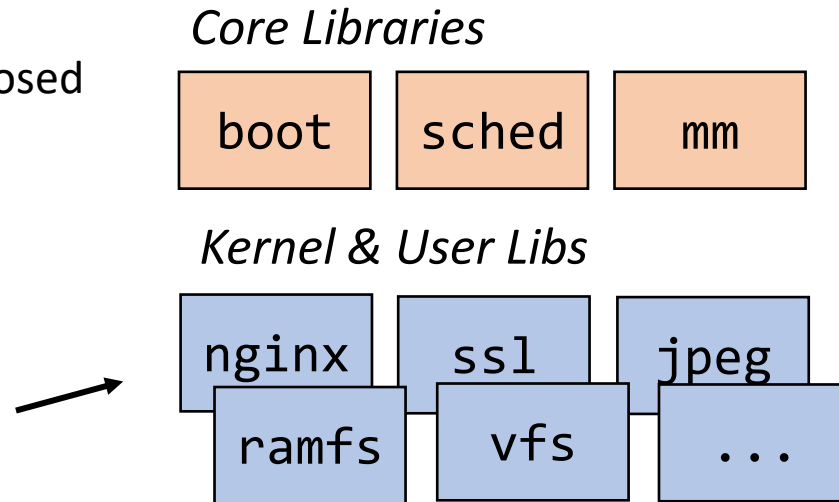
# FlexOS 101: Mechanism Abstraction

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



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

Reuse libraries as finest  
granularity of  
compartmentalization



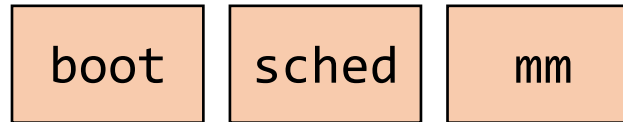
# FlexOS 101: Mechanism Abstraction

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

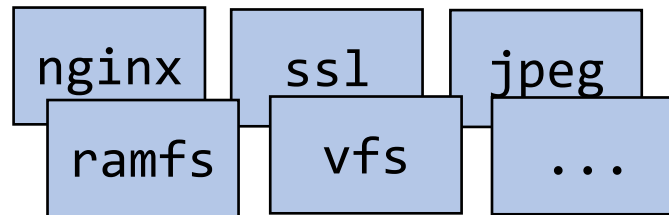


Such libOSes are composed of *fine-granular, independent* libraries

*Core Libraries*



*Kernel & User Libs*



Reuse libraries as finest granularity of compartmentalization



"Pre-compartmentalize" them



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

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

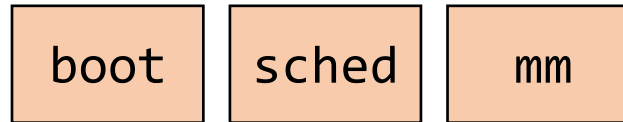
# FlexOS 101: Mechanism Abstraction

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

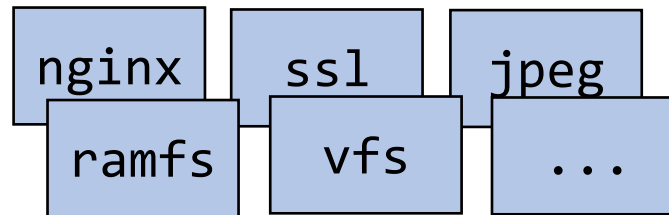


Such libOSes are composed of *fine-granular, independent* libraries

*Core Libraries*



*Kernel & User Libs*



Reuse libraries as finest granularity of compartmentalization

"Pre-compartmentalize" them

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

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

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



# FlexOS 101: Compartmentalization API

```
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);
```

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



Porting

# 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

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

Add gate placeholders

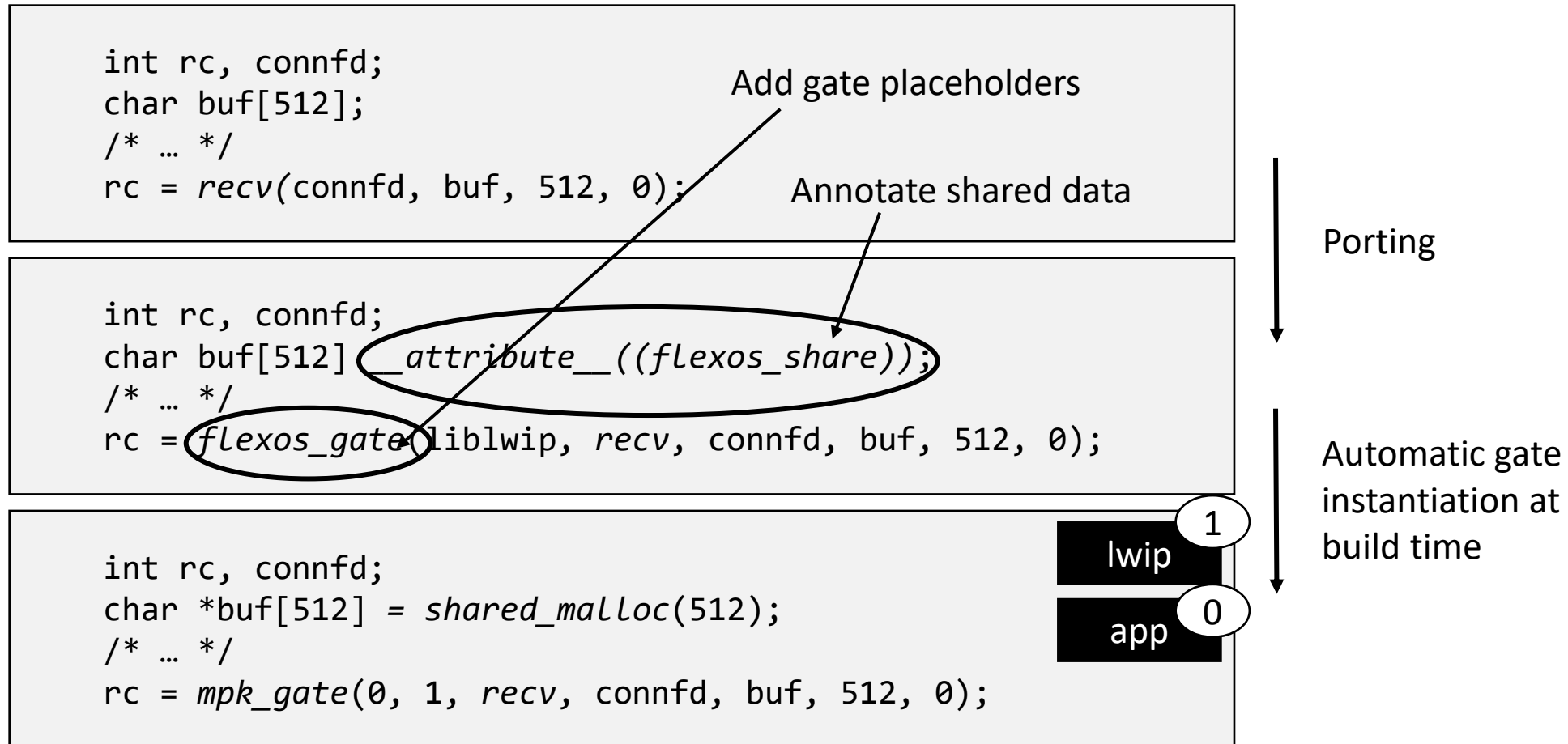
Annotate shared data

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

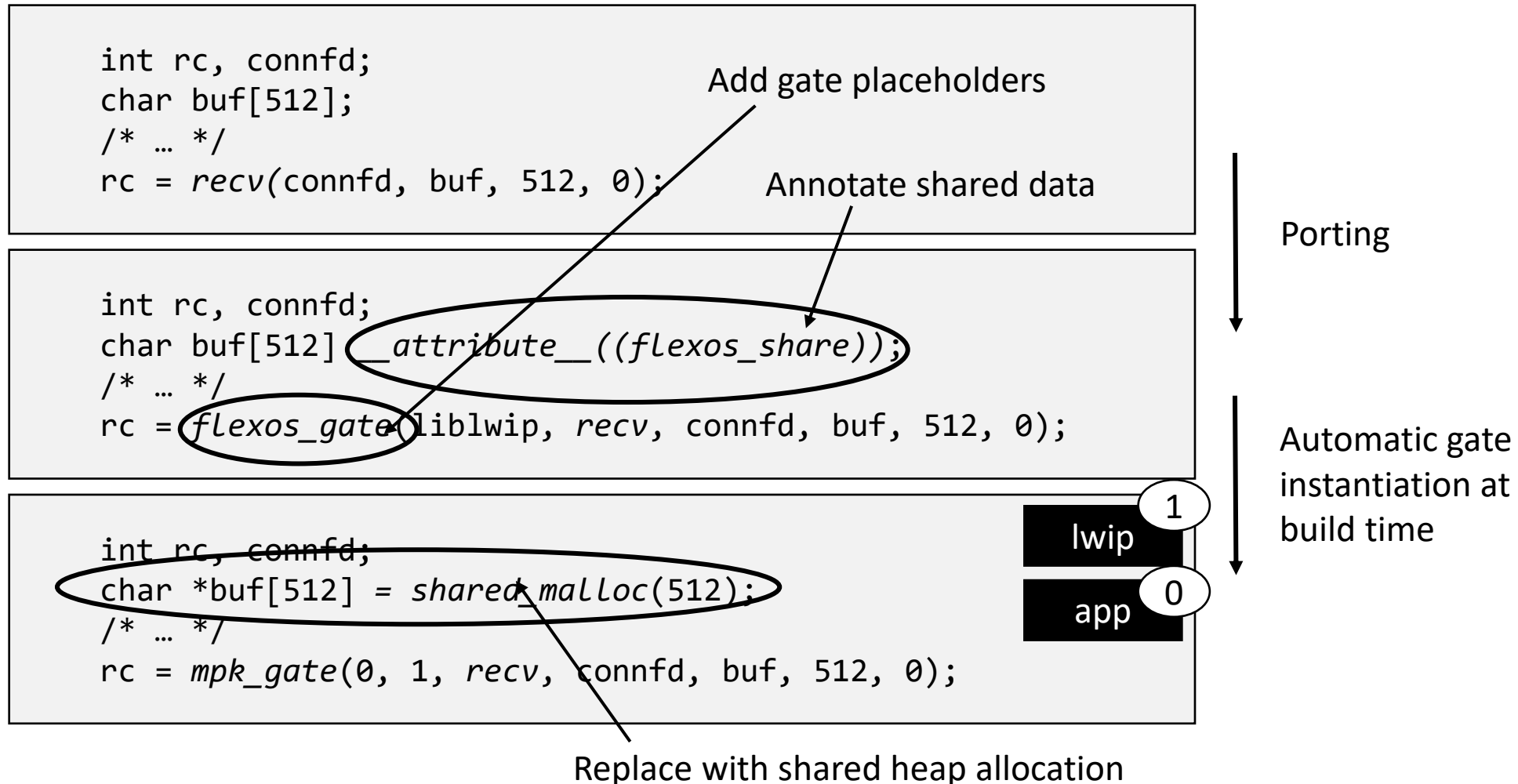
Porting



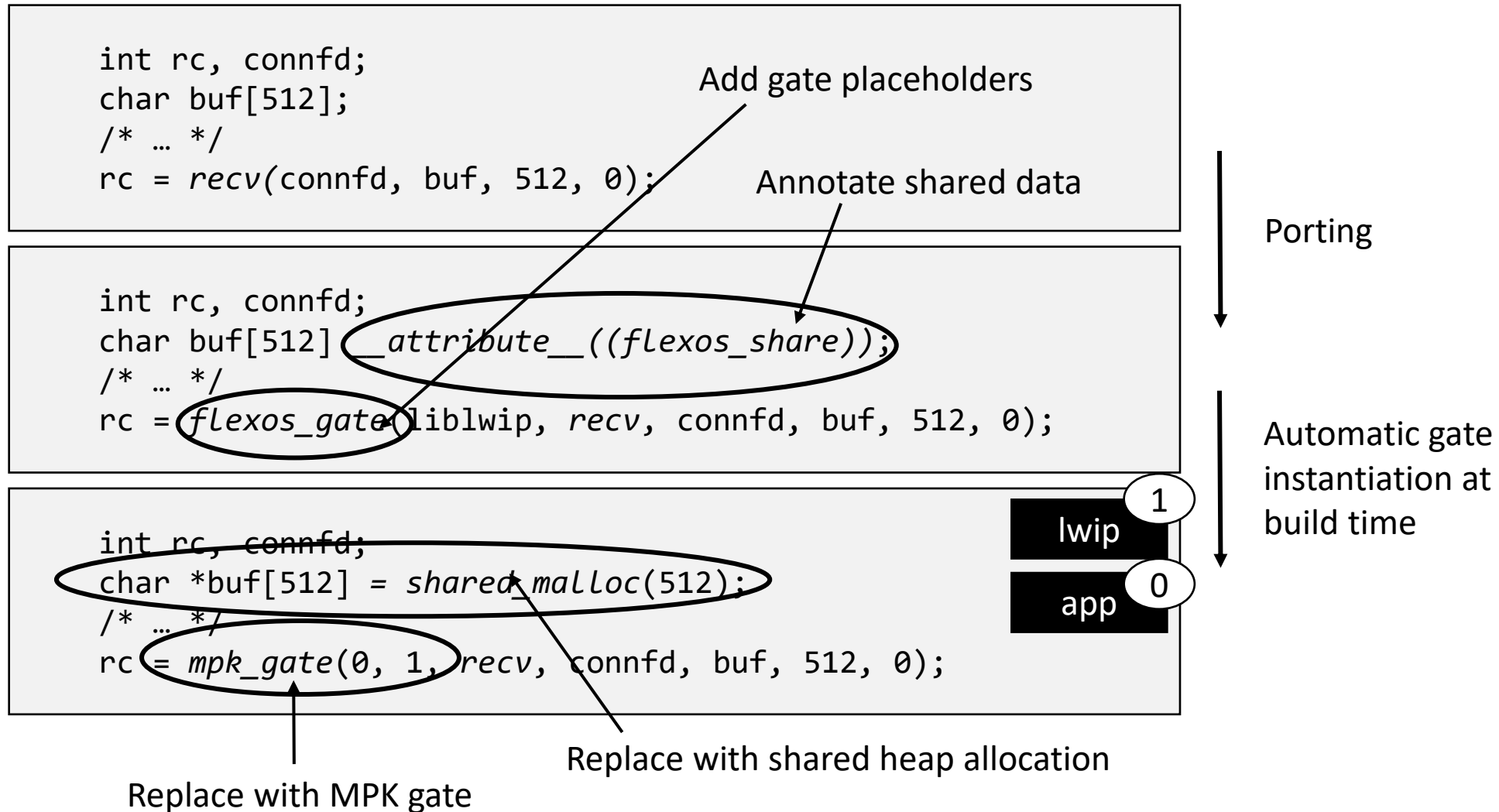
# FlexOS 101: Compartmentalization API



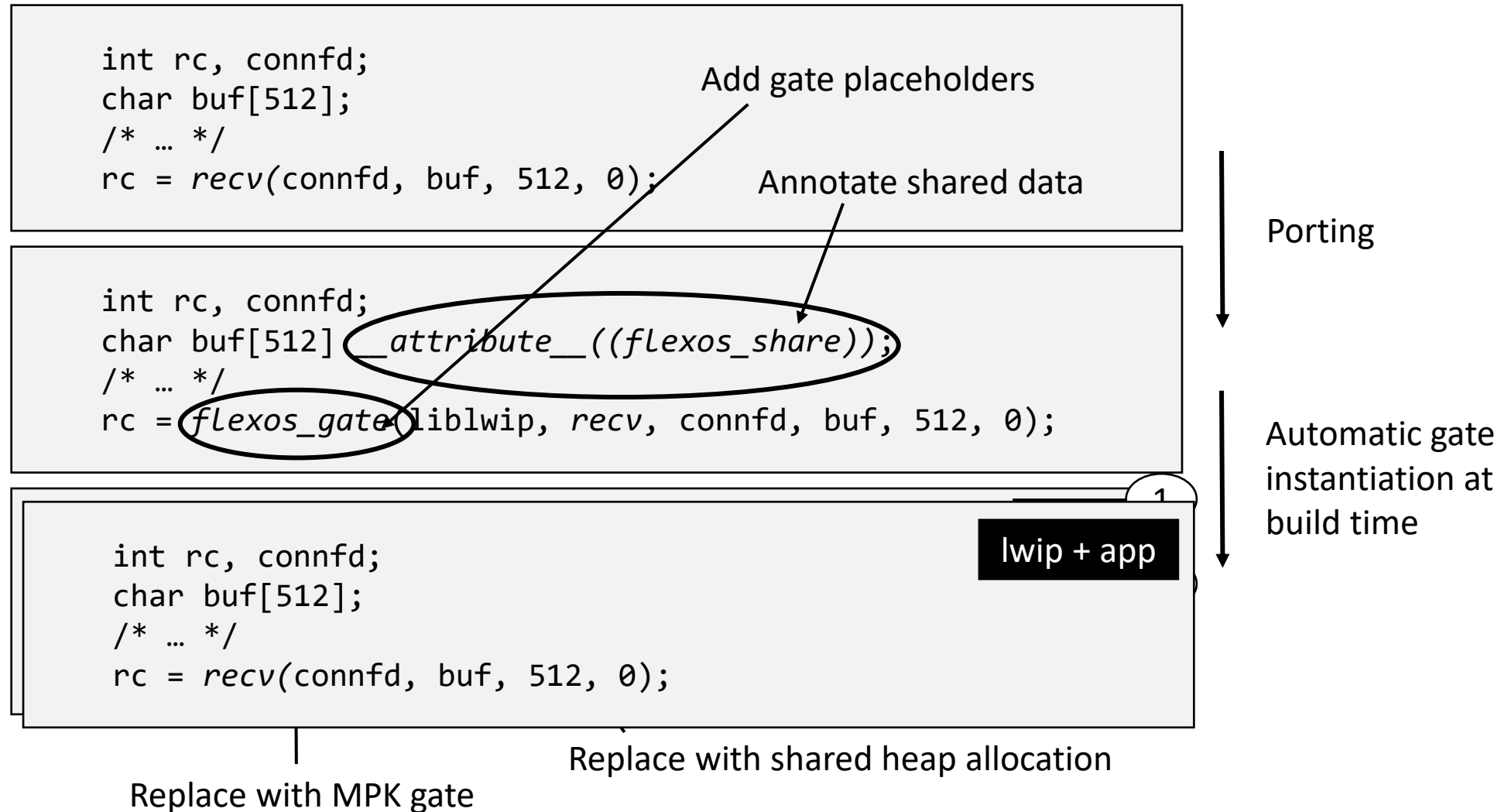
# FlexOS 101: Compartmentalization API



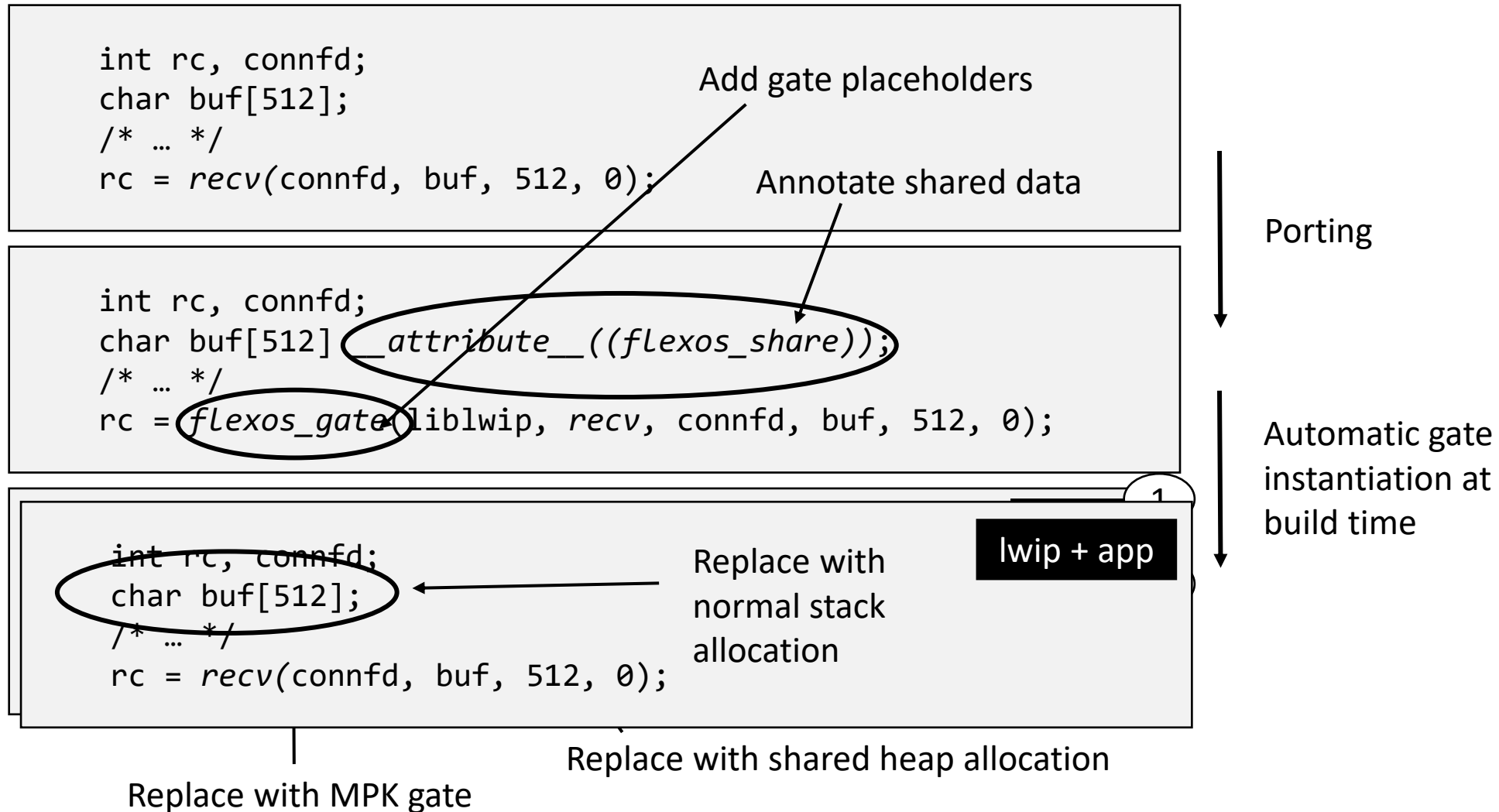
# FlexOS 101: Compartmentalization API



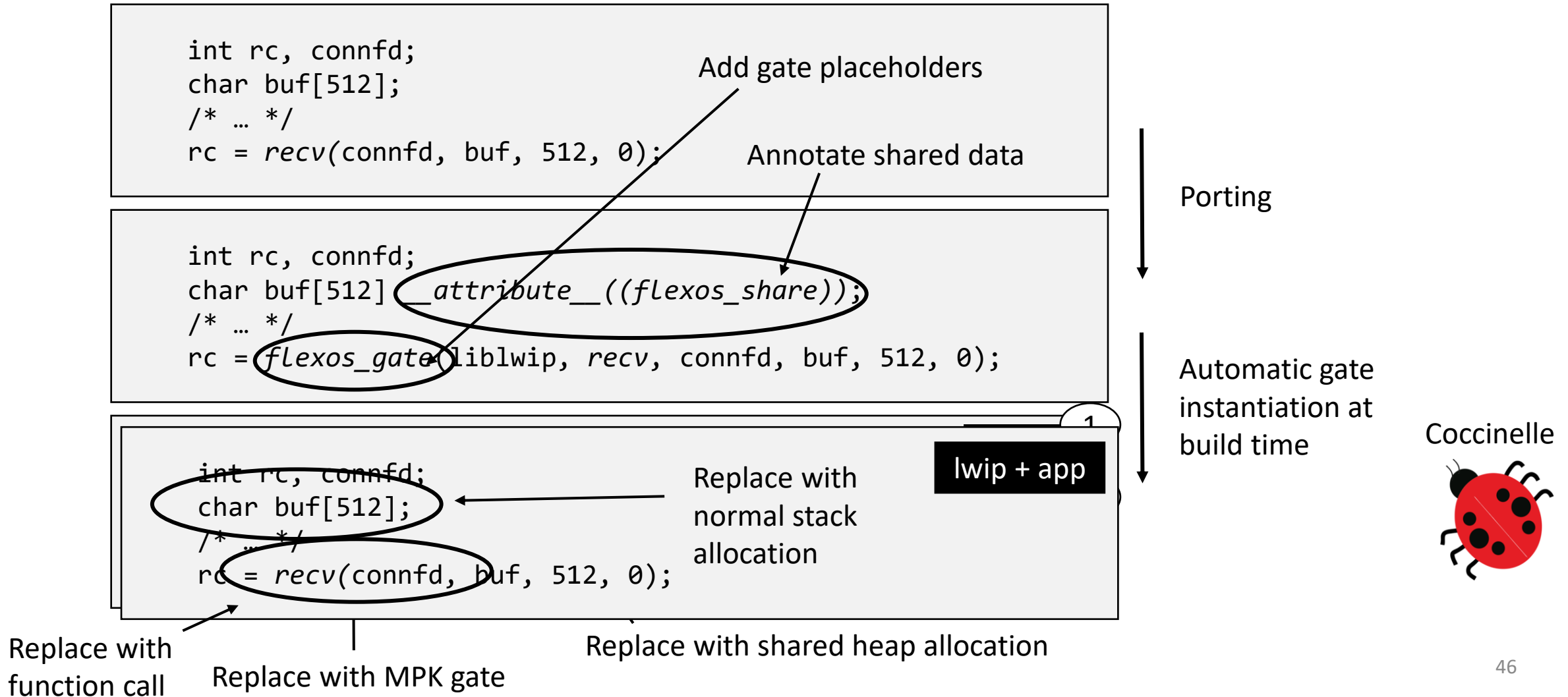
# FlexOS 101: Compartmentalization API



# 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



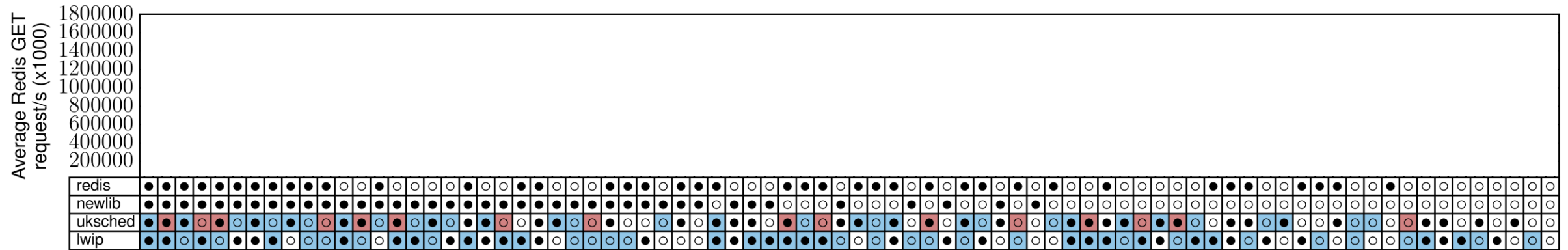
This talk: focus on demonstrating **flexibility and performance**



more results in our paper 😊



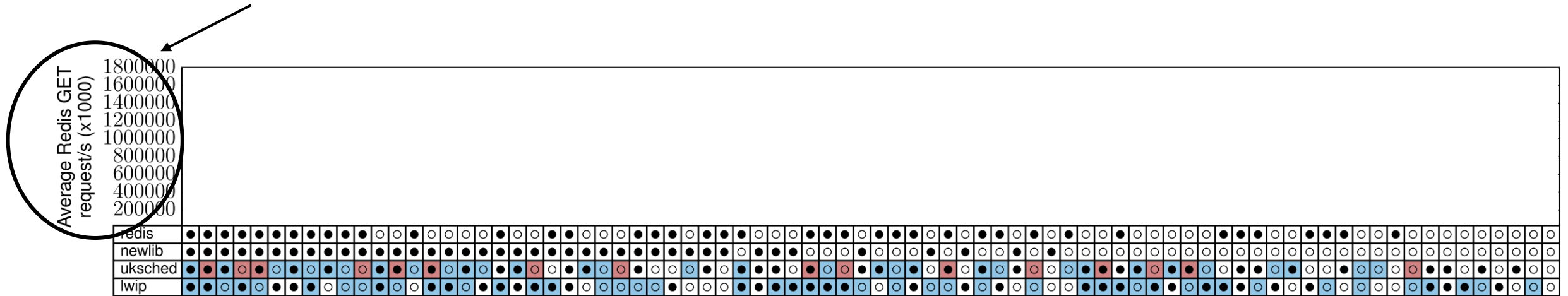
# Flexibility



# Flexibility



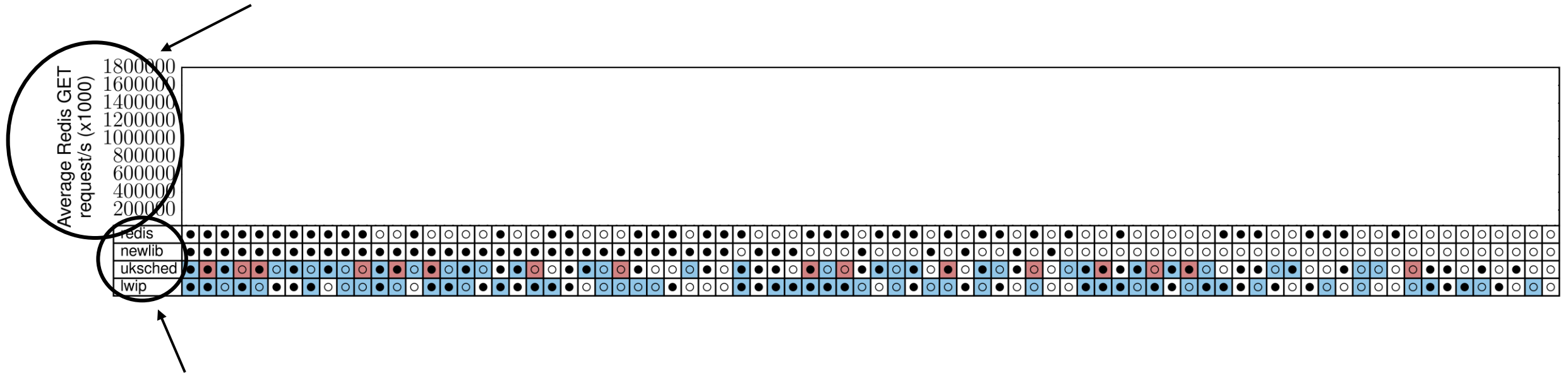
Runtime performance with Redis in requests/s



# Flexibility



Runtime performance with Redis in requests/s



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

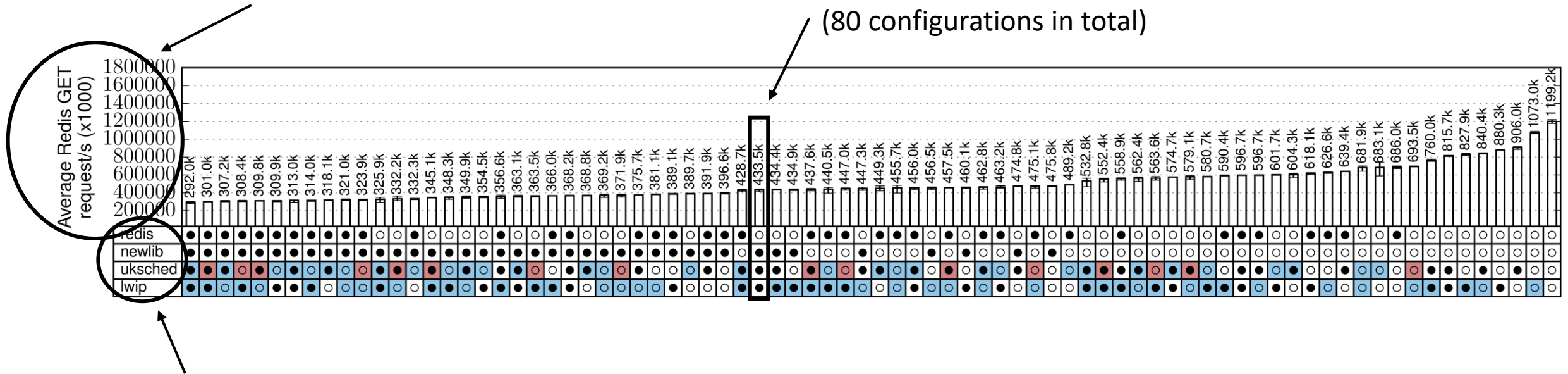
- Redis application
- C standard library (newlib)
- FlexOS scheduler (*uksched*)
- Network stack (*lwip*)

# Flexibility



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):

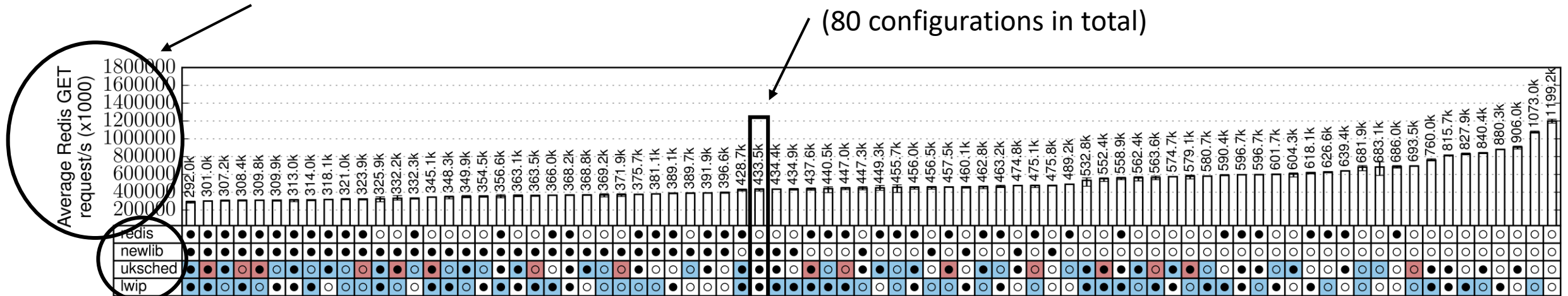
- Redis application
- C standard library (newlib)
- FlexOS scheduler (*uksched*)
- Network stack (*lwip*)

# Flexibility



Runtime performance with Redis in requests/s

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



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:

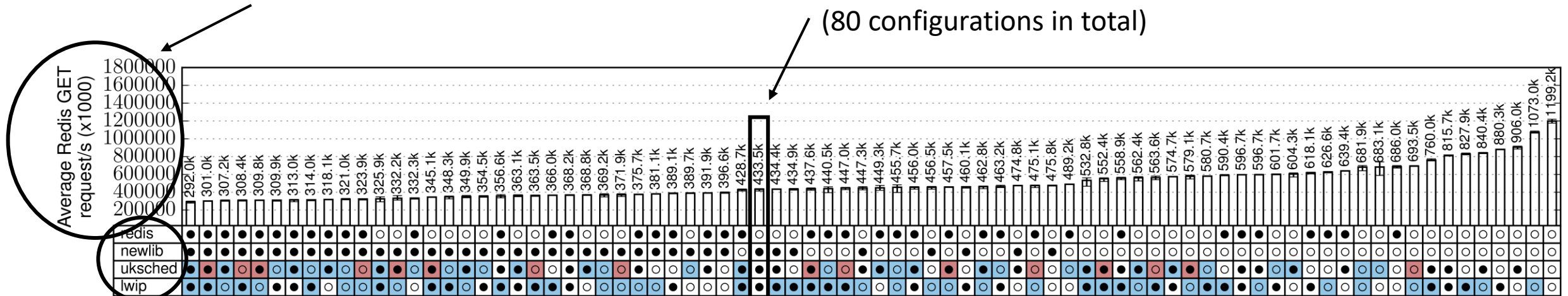


# Flexibility



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)

The color of boxes indicates the compartment:

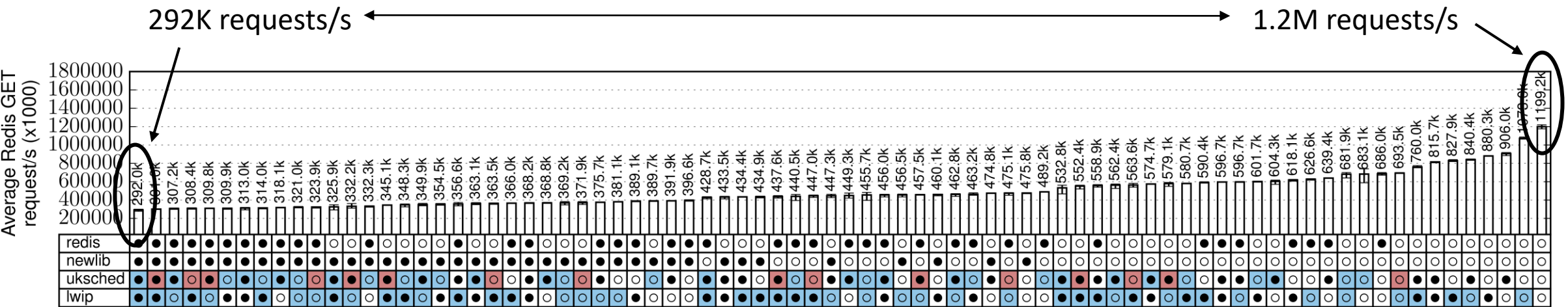
Compartment 1
  Compartment 2
  Compartment 3

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

Hardening on
  Hardening off

# Flexibility

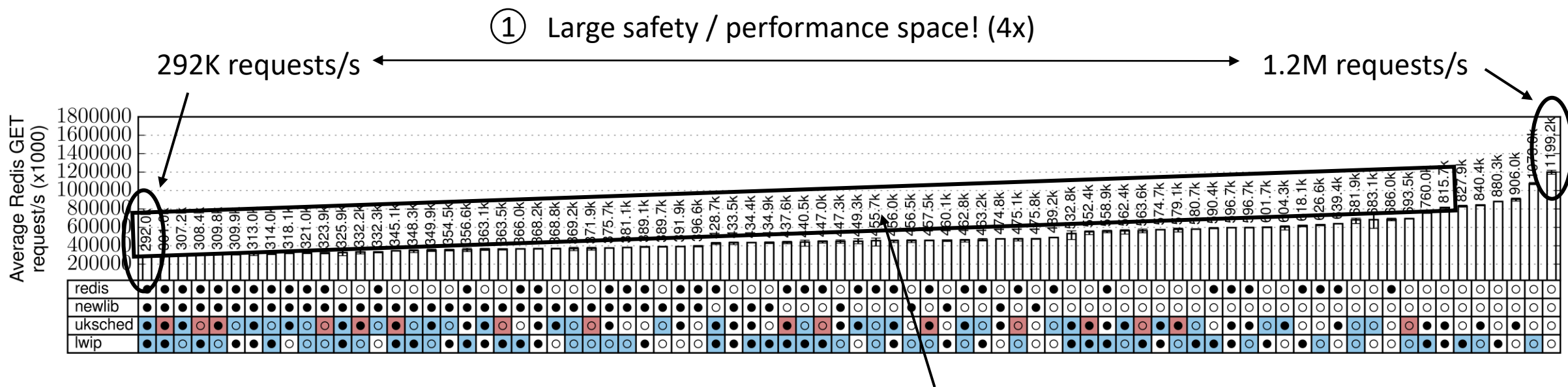
① Large safety / performance space! (4x)



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



# Flexibility

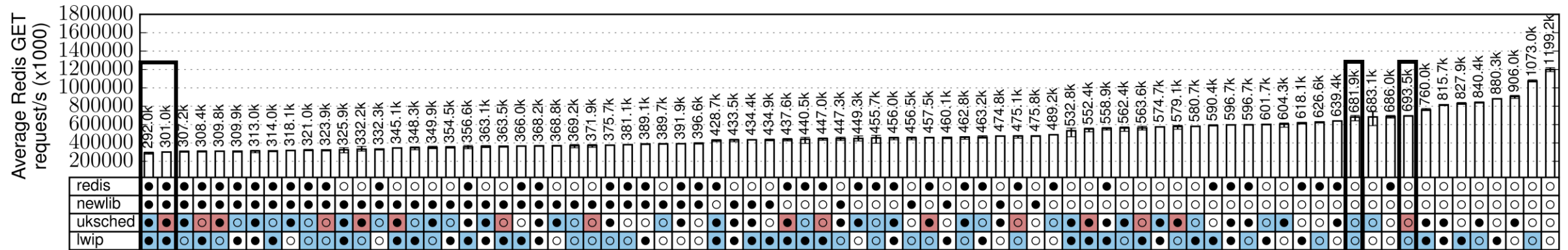


② Smooth slope, performance degrades gracefully





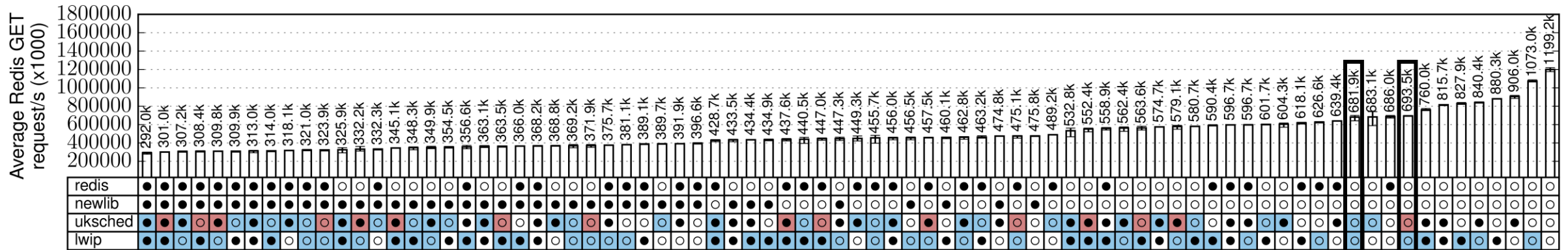
# Flexibility



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

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

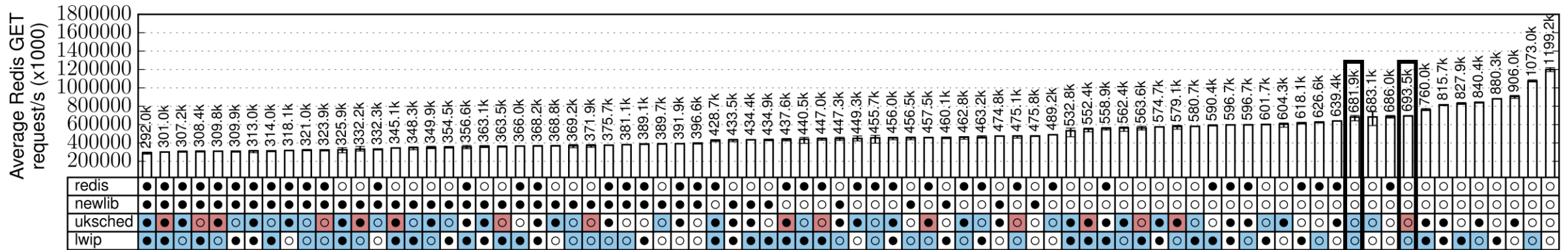
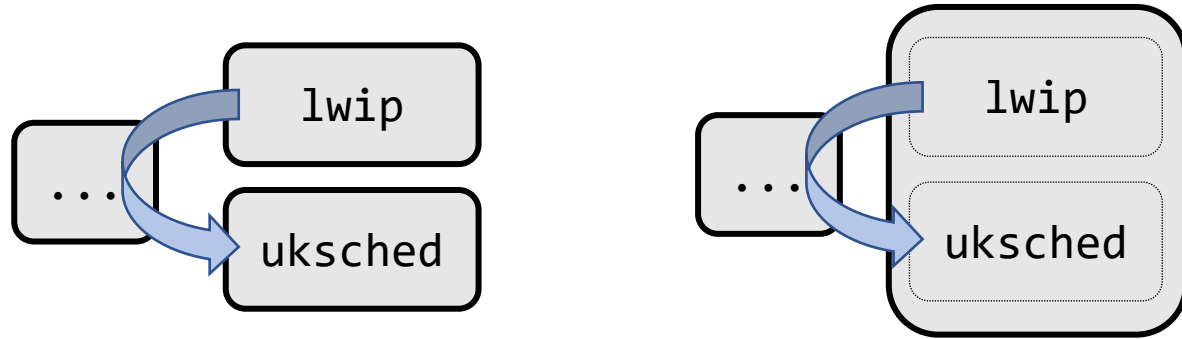
# Flexibility



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

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

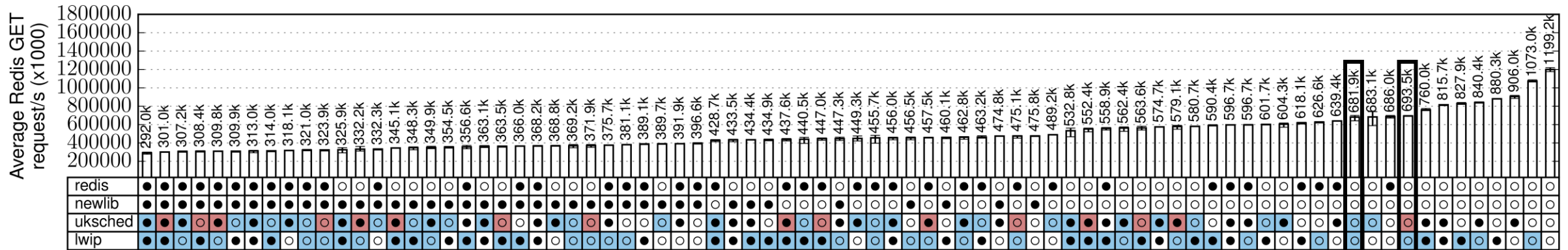
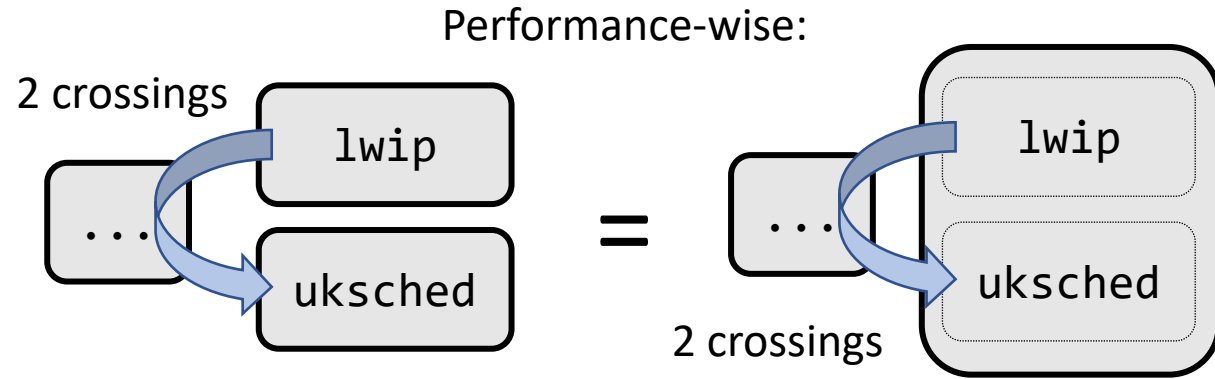
# Flexibility



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



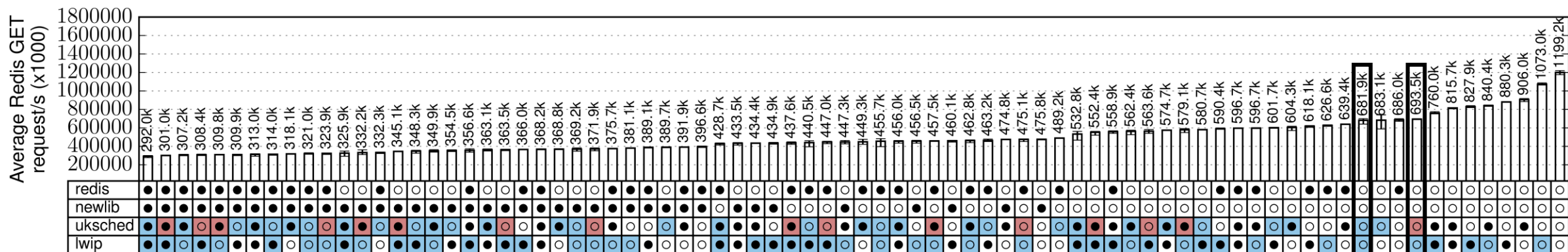
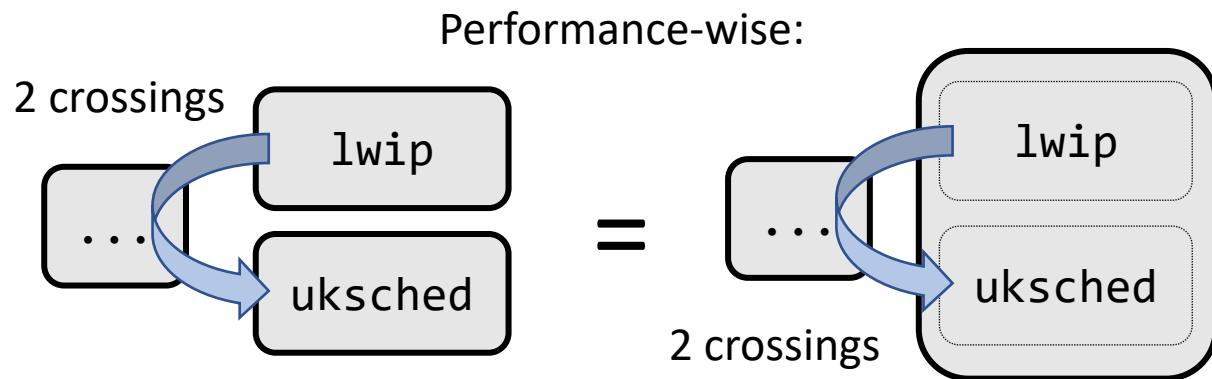
# Flexibility



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



# Flexibility

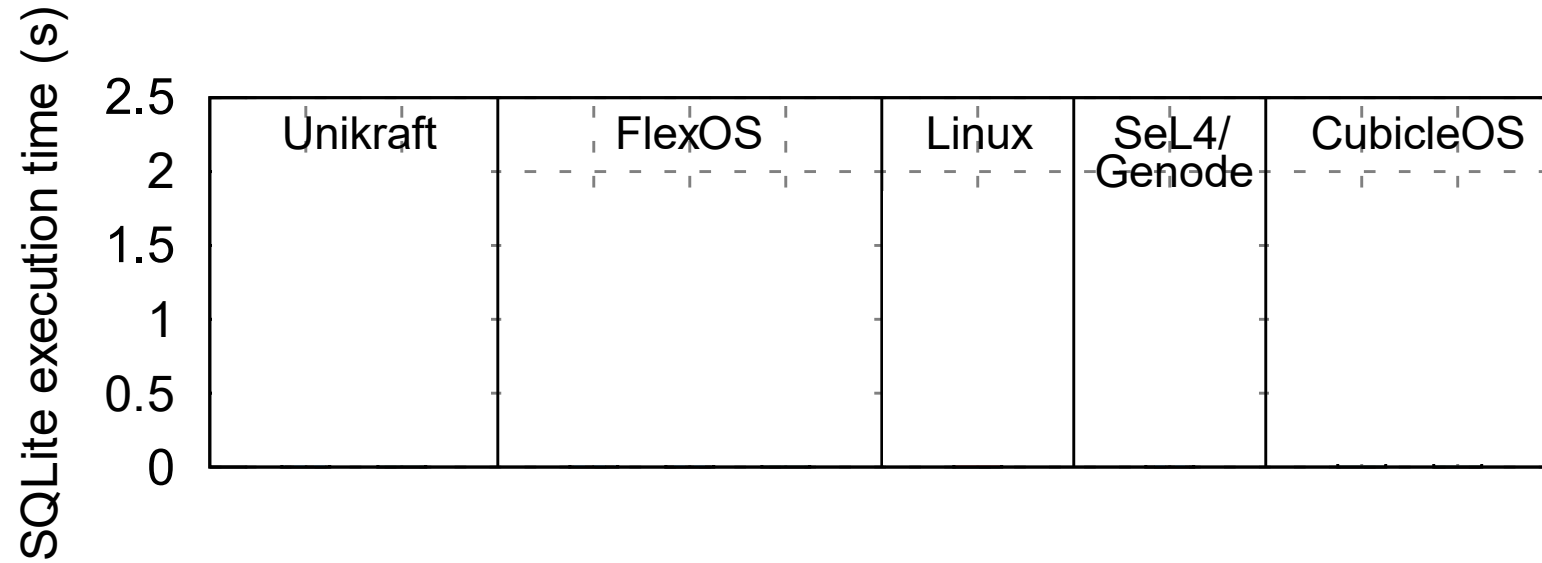


You can get some safety for free by exploring intelligently

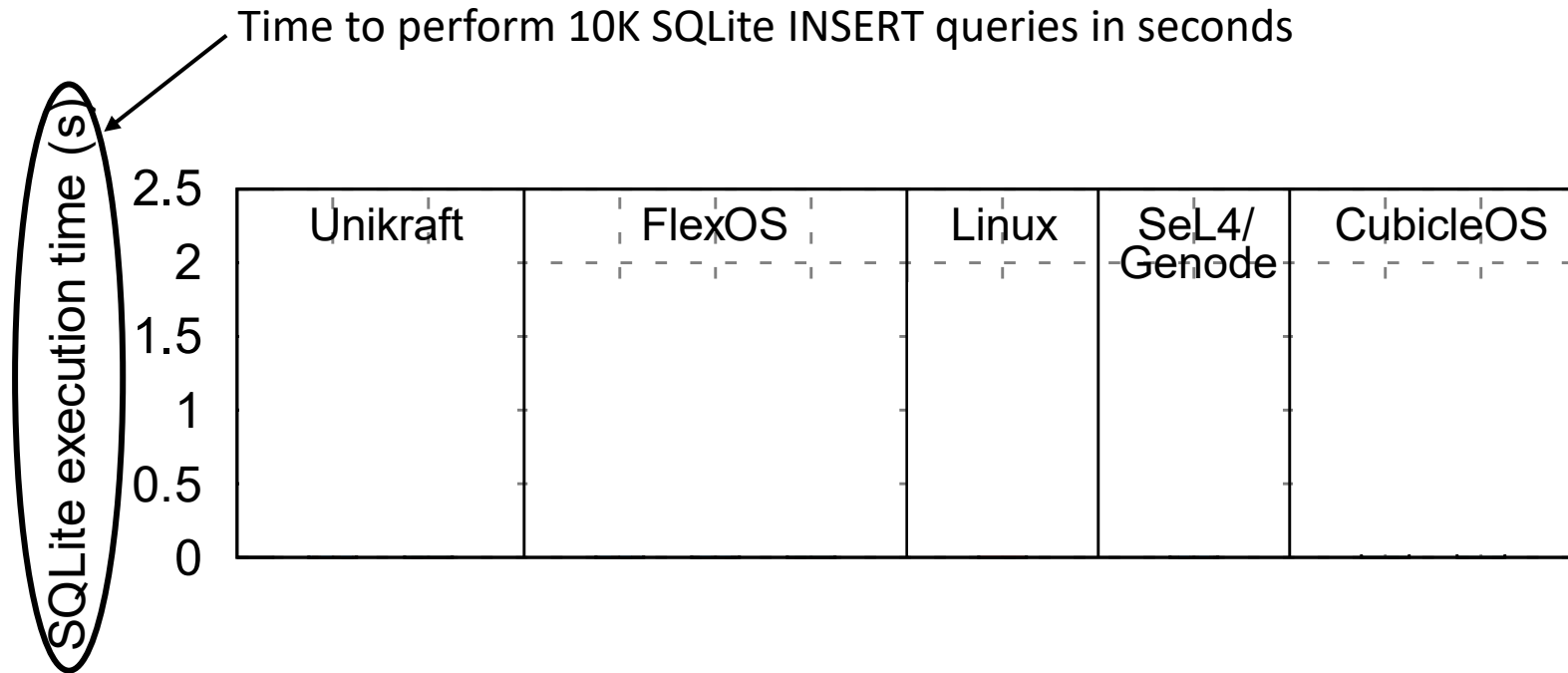
③ Similar performance, very different properties!  
need to reason about communication patterns, fast paths

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

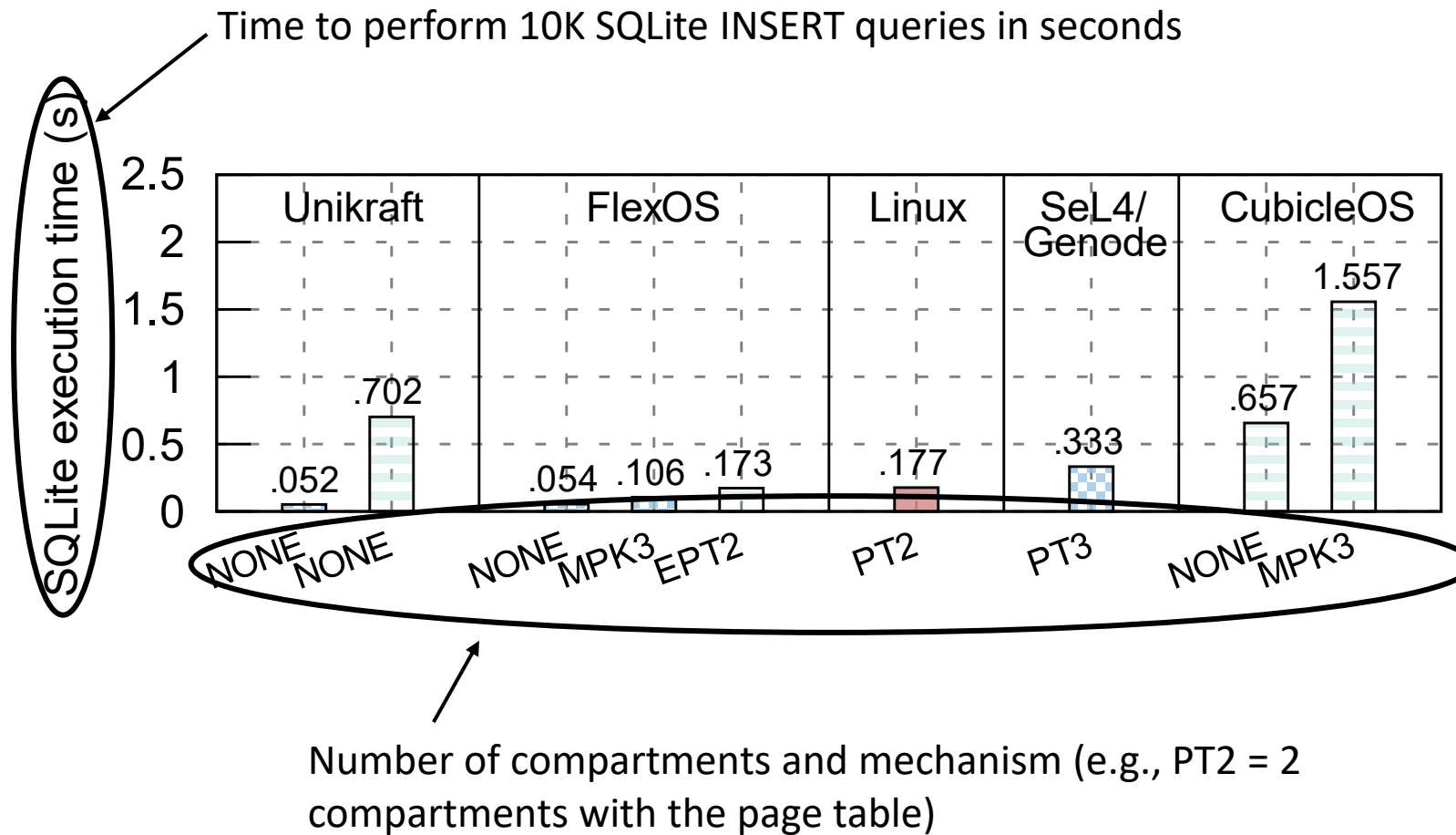
# Performance



# Performance

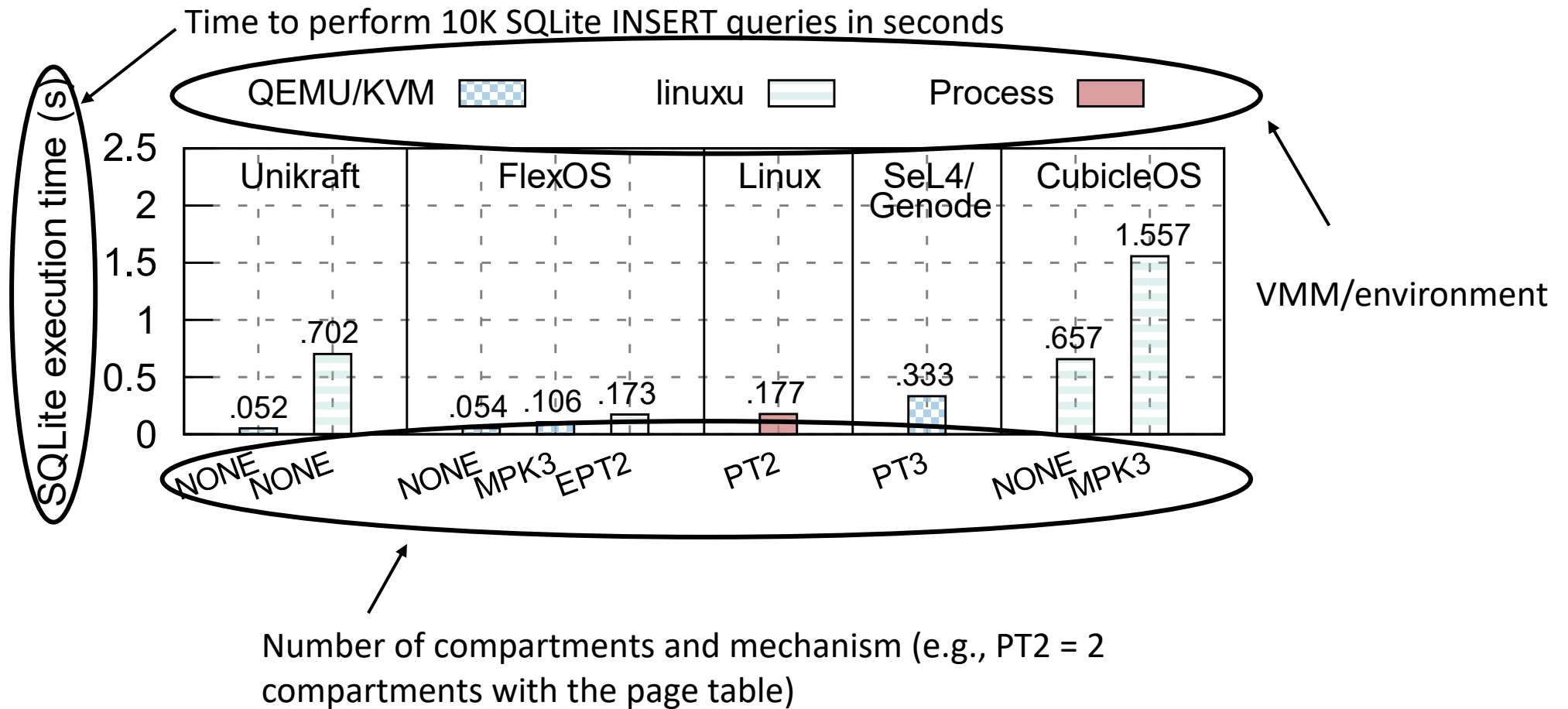


# Performance

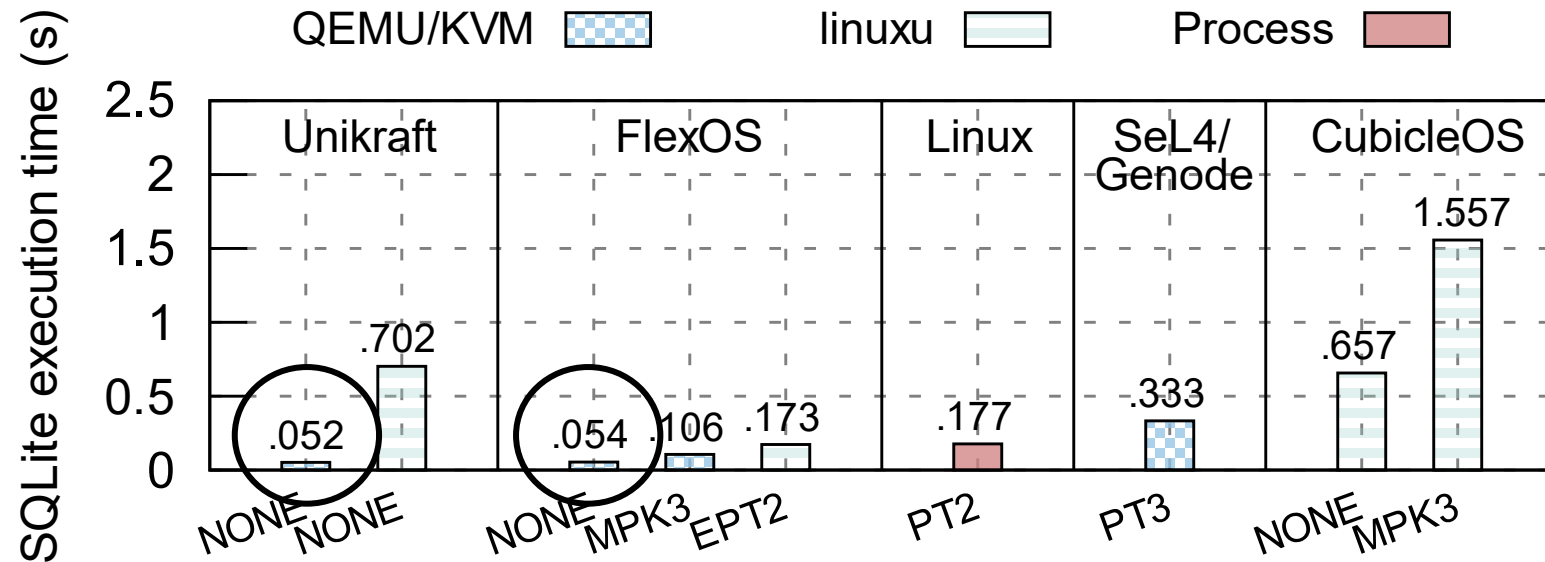




# Performance

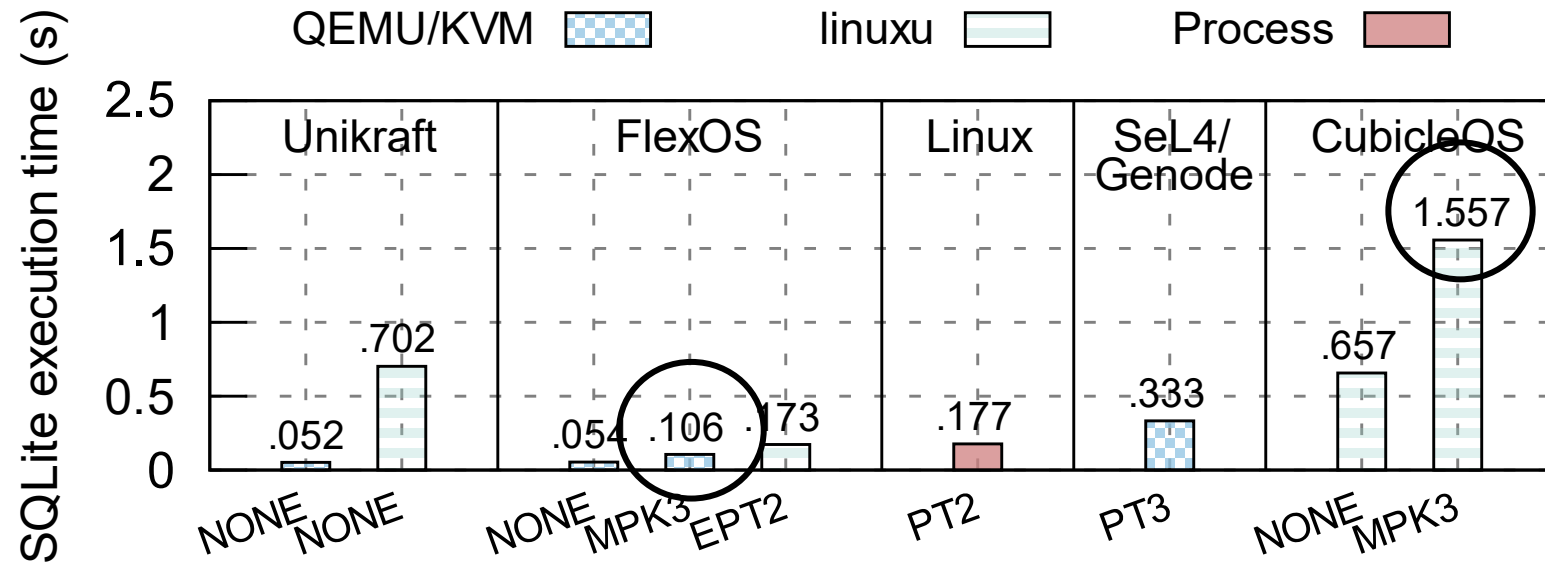


# Performance



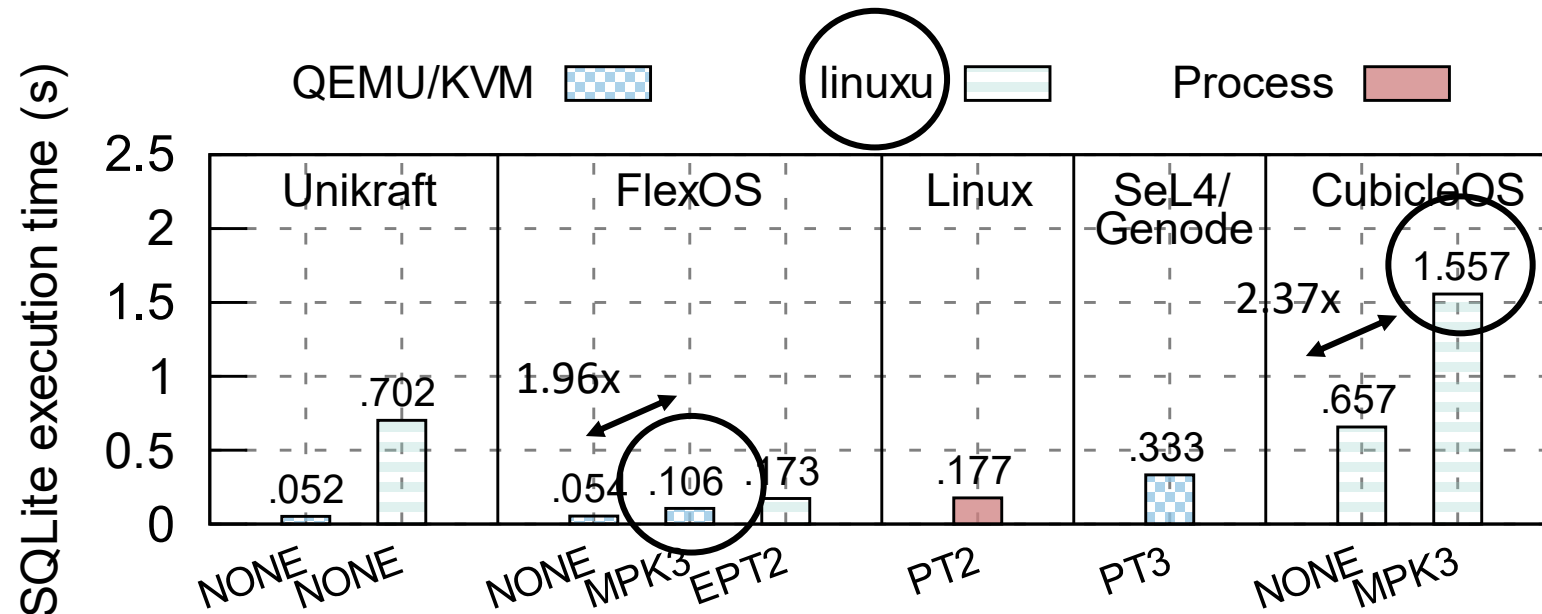
- ① No overhead when disabling isolation – you only pay for what you get

# Performance



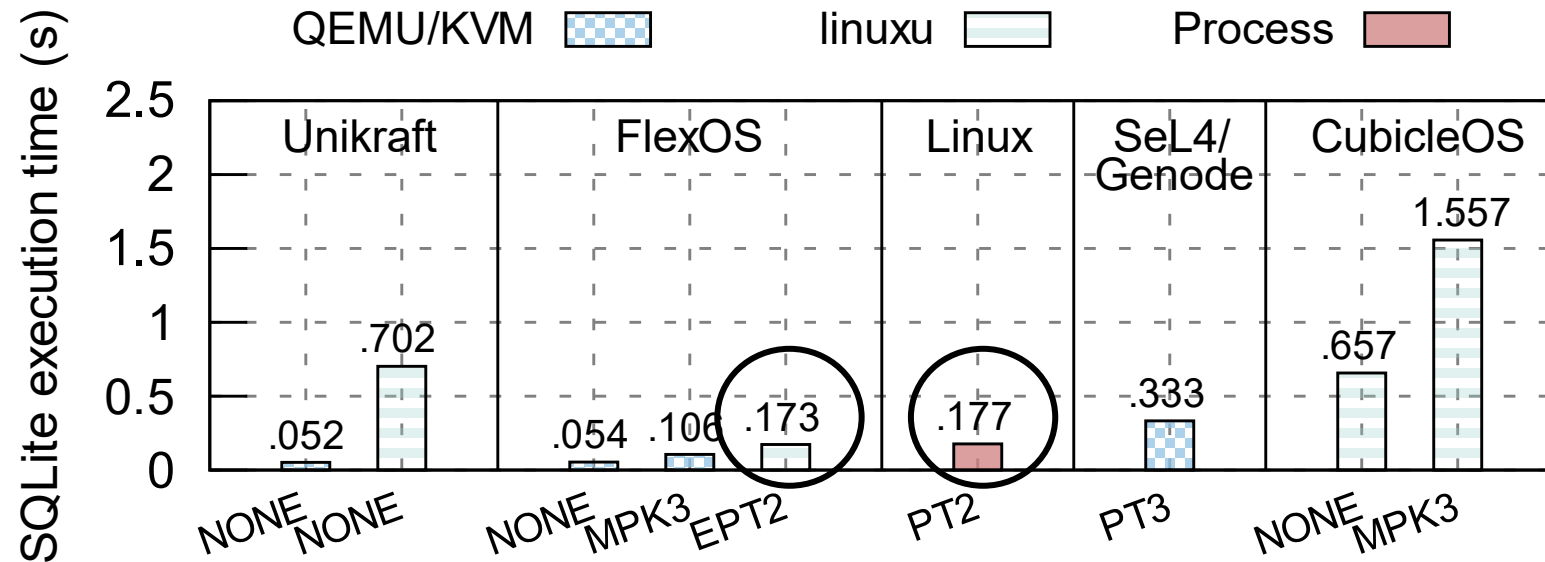
- ② The MPK backend compares very positively to competing solutions

# 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!

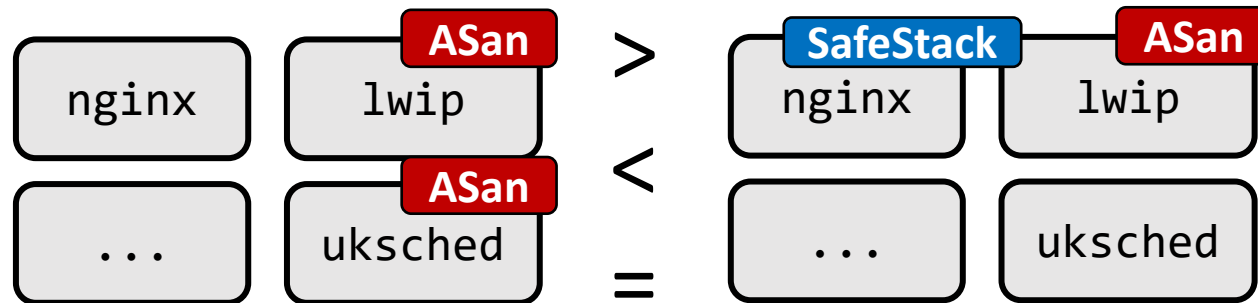
Problem: some configurations are not comparable

# 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



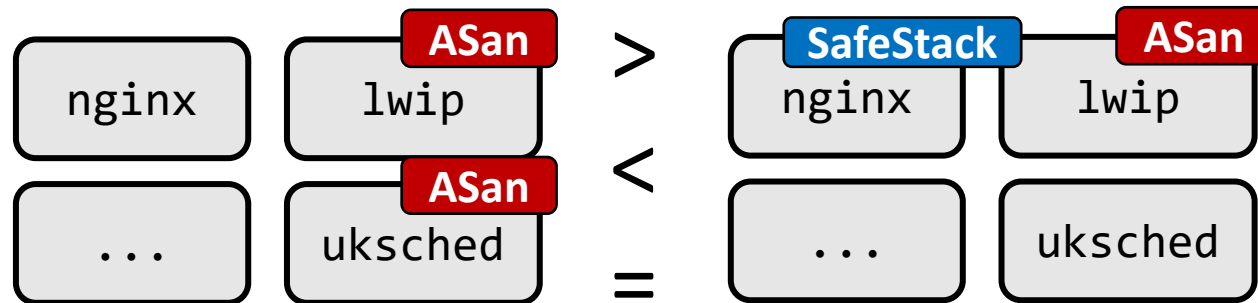


# Exploring the Design Space

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?

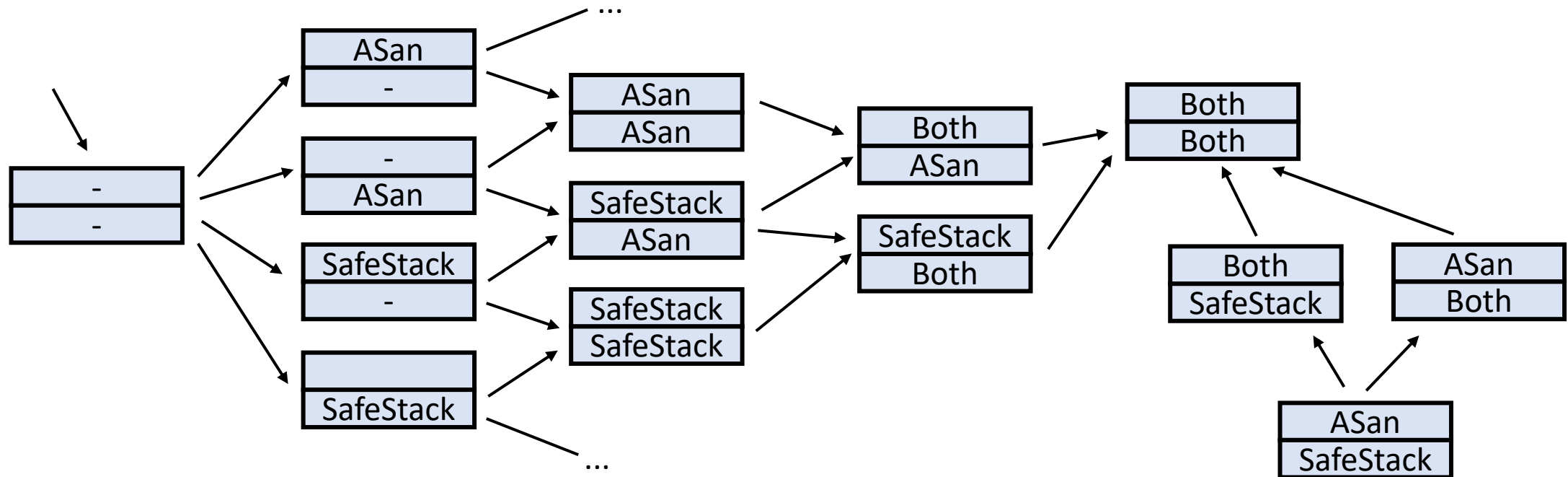


# Exploring the Design Space

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

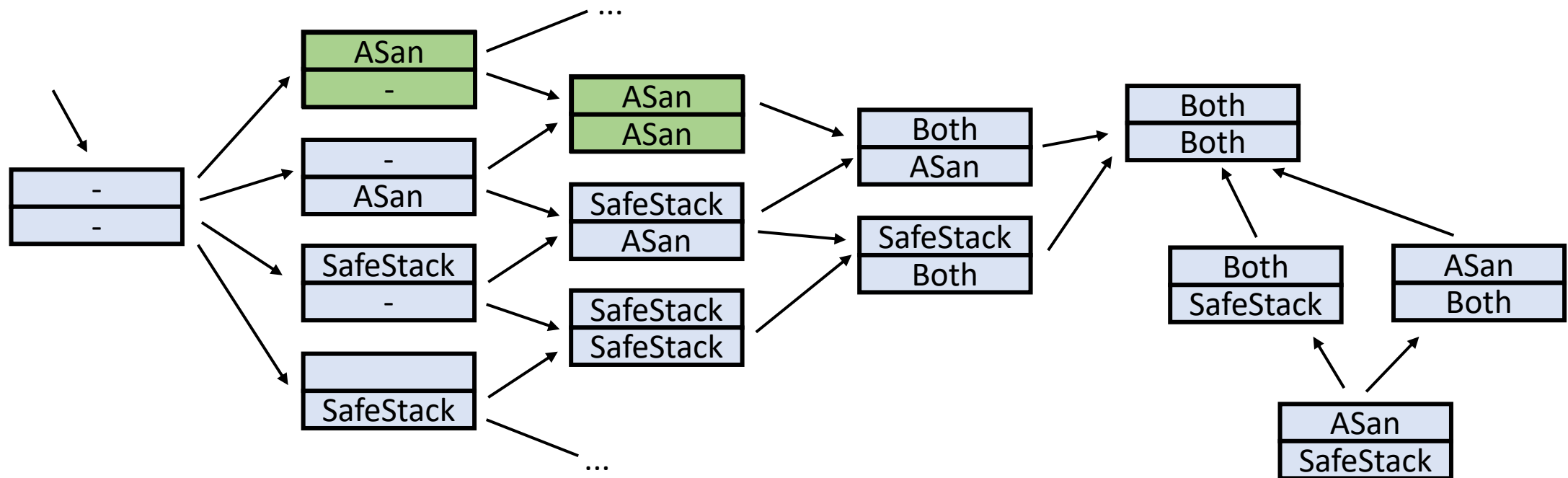
# Exploring the Design Space

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



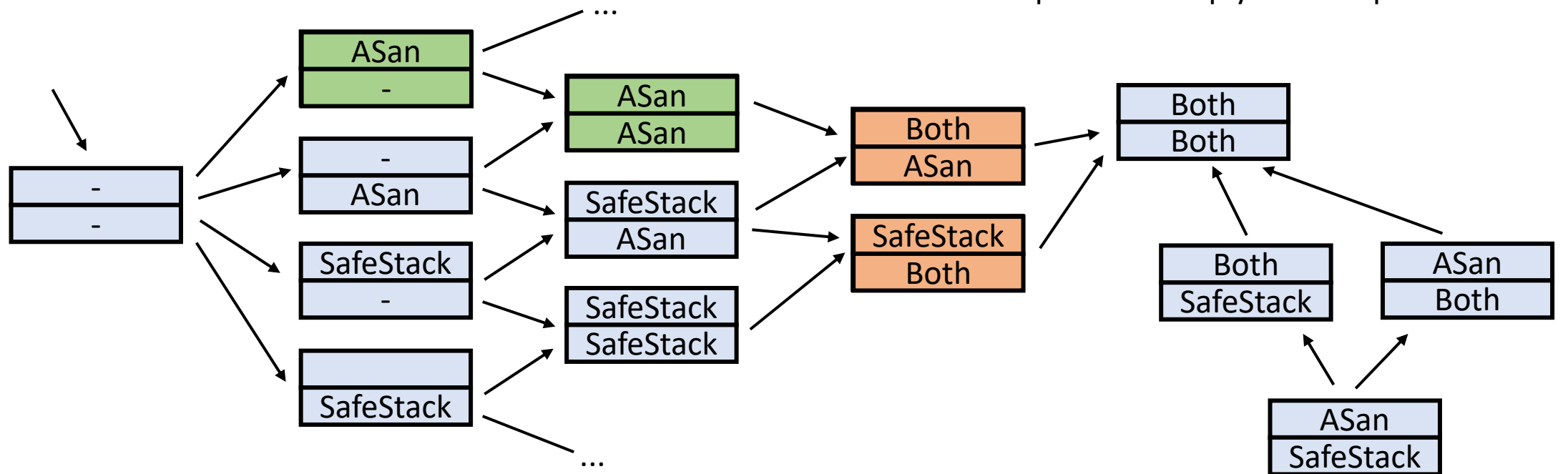
# Exploring the Design Space

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



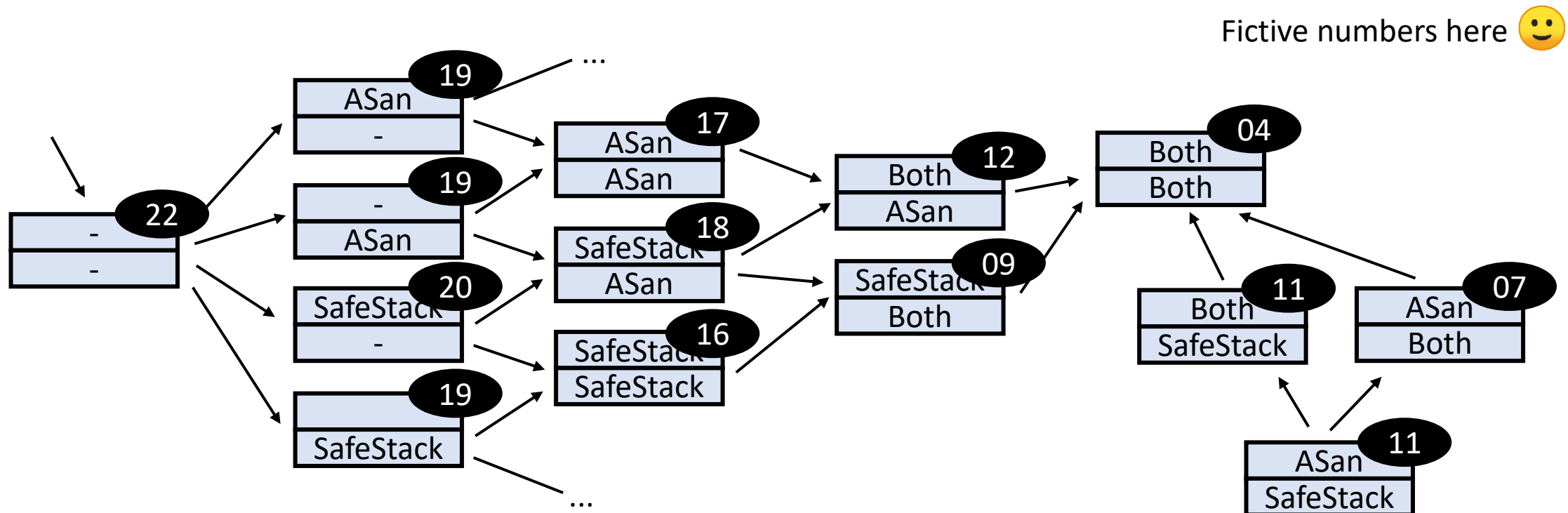
# Exploring the Design Space

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



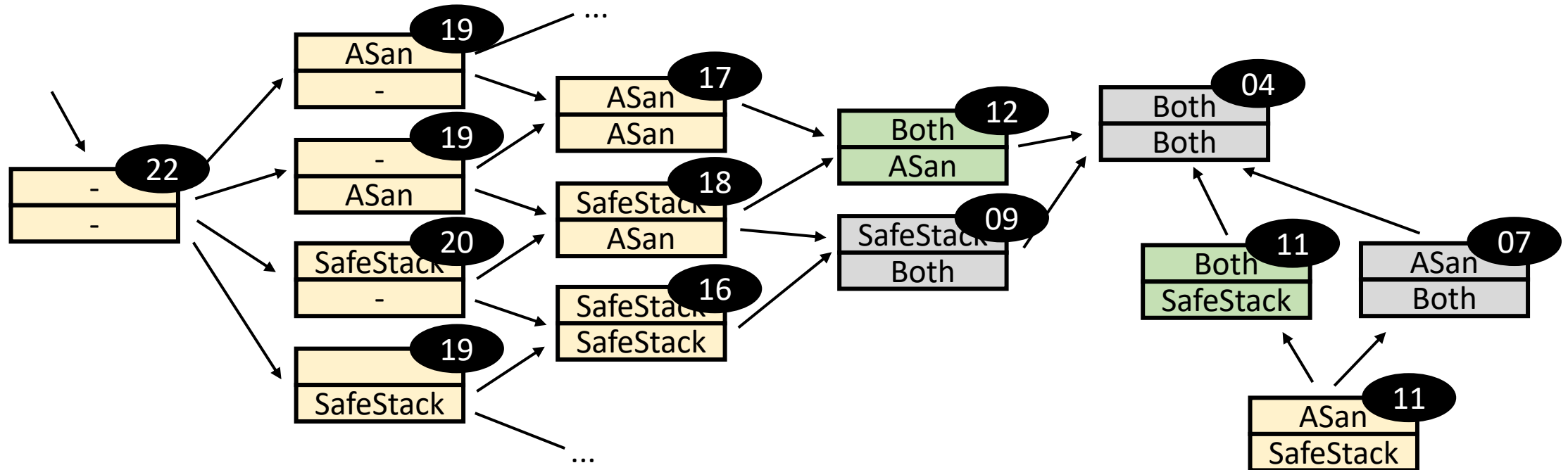
# Exploring the Design Space

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

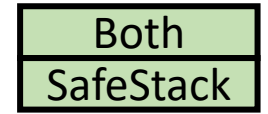
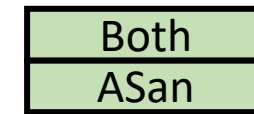


# Exploring the Design Space

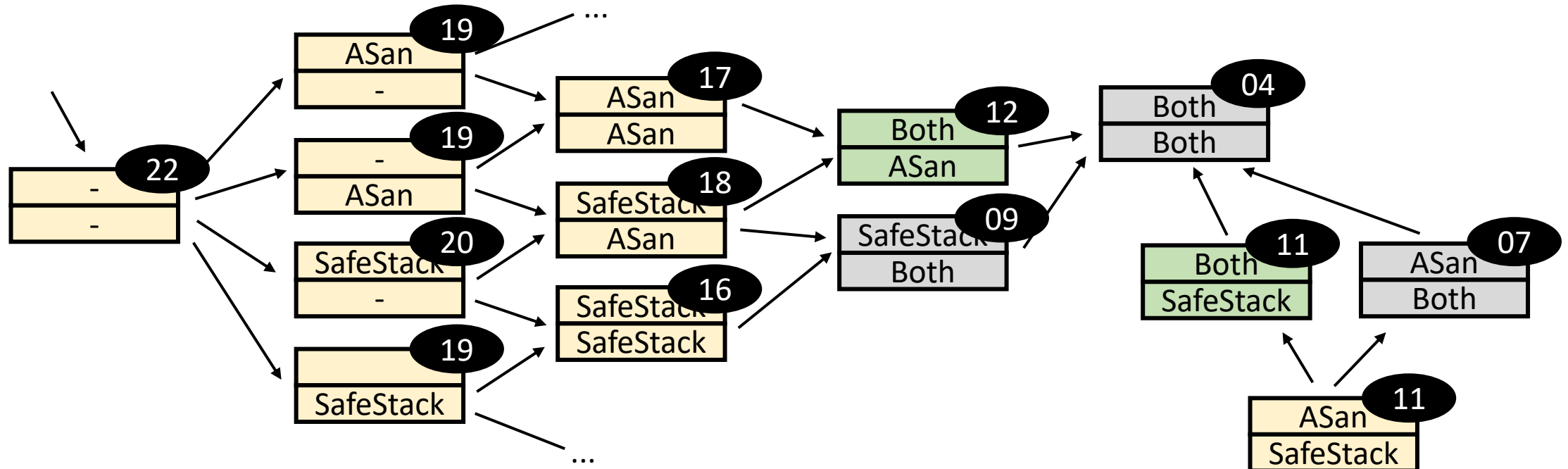
Let the user do  
the final choice



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



Curated list of optimal configurations



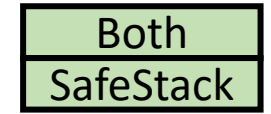
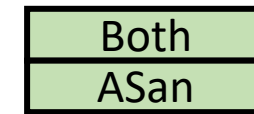


# Exploring the Design Space

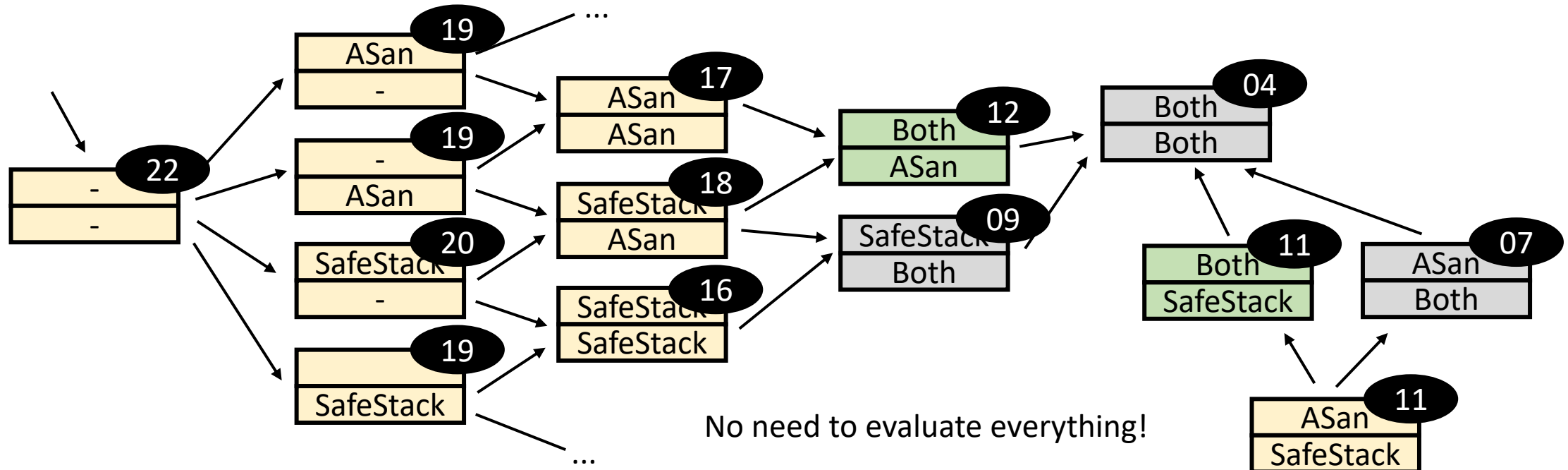
Let the user do  
the final choice



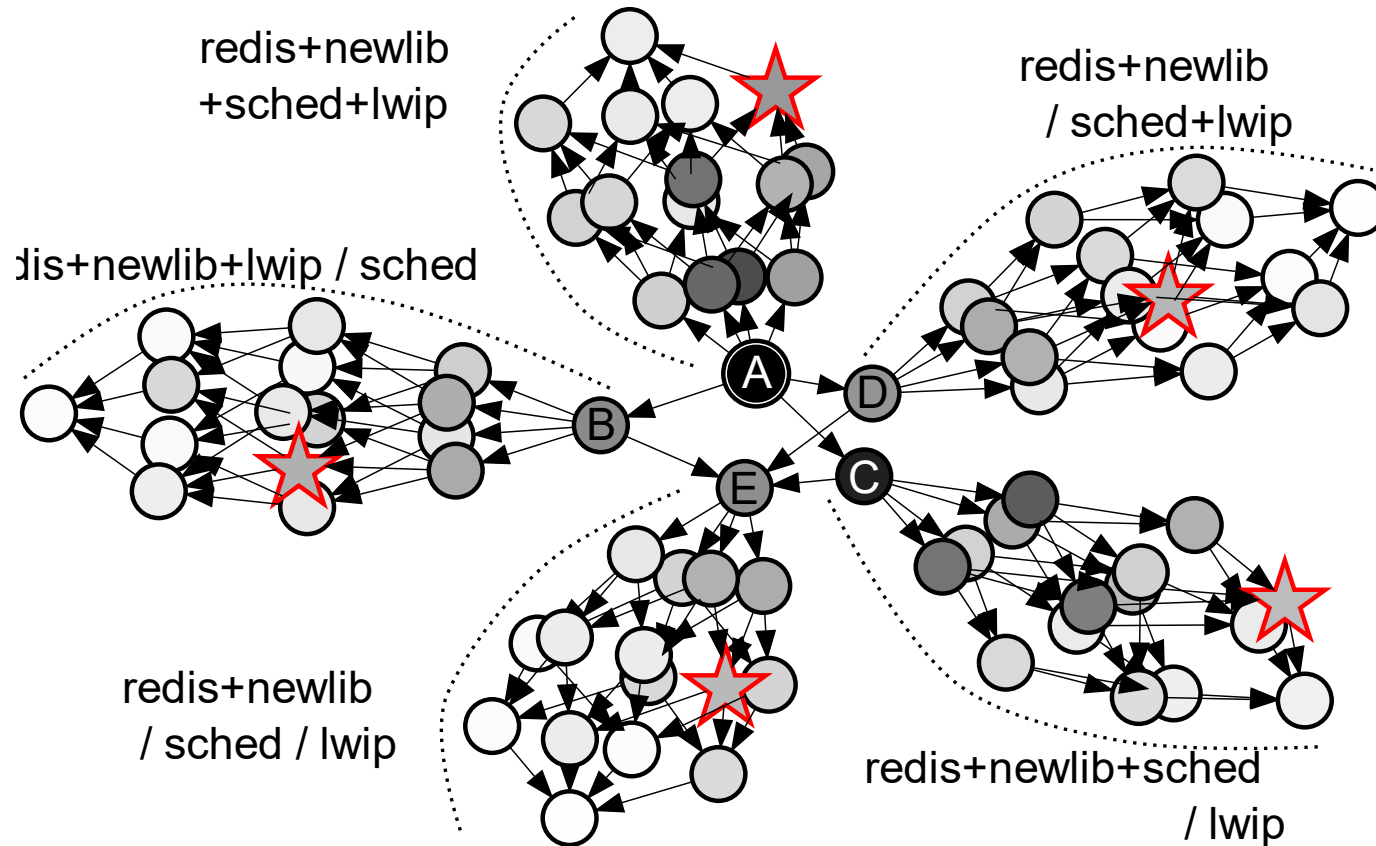
Based on this ordering and labeling we can choose the last node of each path that satisfies the performance constraints



Curated list of optimal configurations

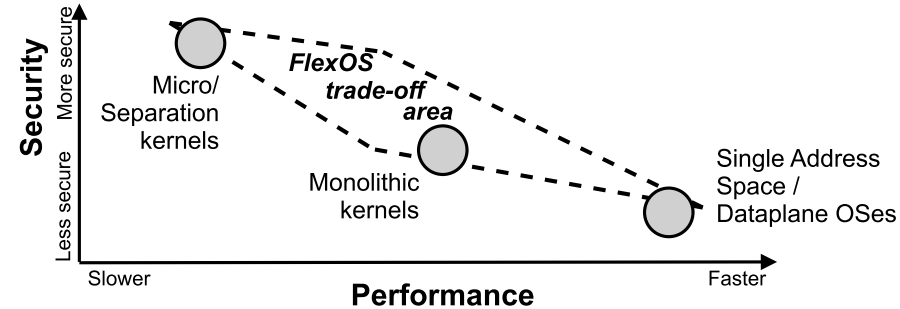


# Applying POSets on Redis



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**

# Interested?



## Get in touch!

Webpage: <https://project-flexos.github.io/>

Our ASPLOS'22 paper: <https://dl.acm.org/doi/10.1145/3503222.3507759>

By e-mail: [hugo.lefeuvre@manchester.ac.uk](mailto:hugo.lefeuvre@manchester.ac.uk)

License: 3-Clause BSD License 😊

## We're hiring! PhD and Postdoc positions at The University of Manchester.

