FlexOS: Making OS Isolation Flexible

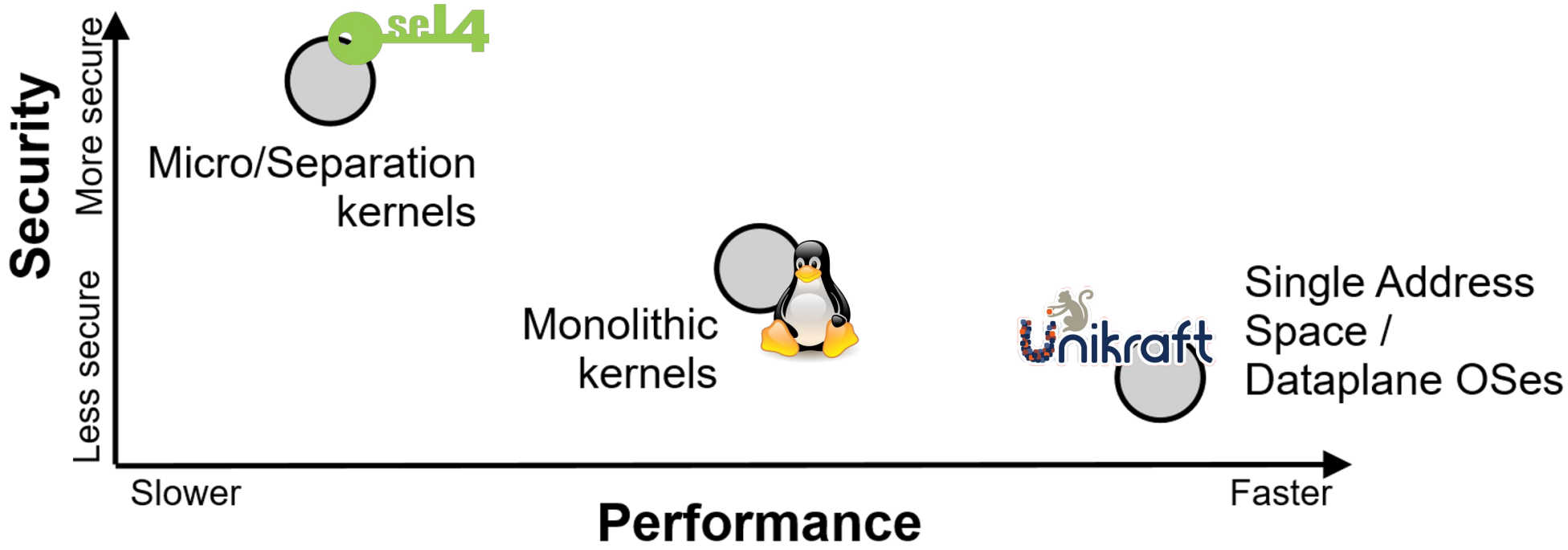
H. Lefeuvre, V-A. Badoiu, S.L. Teodorescu, P. Olivier,
T. Mosnoi, R. Deaconescu, F. Huici, C. Raiciu

HotOS 2021, 1 June–3 June 2021

Current OS Designs

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

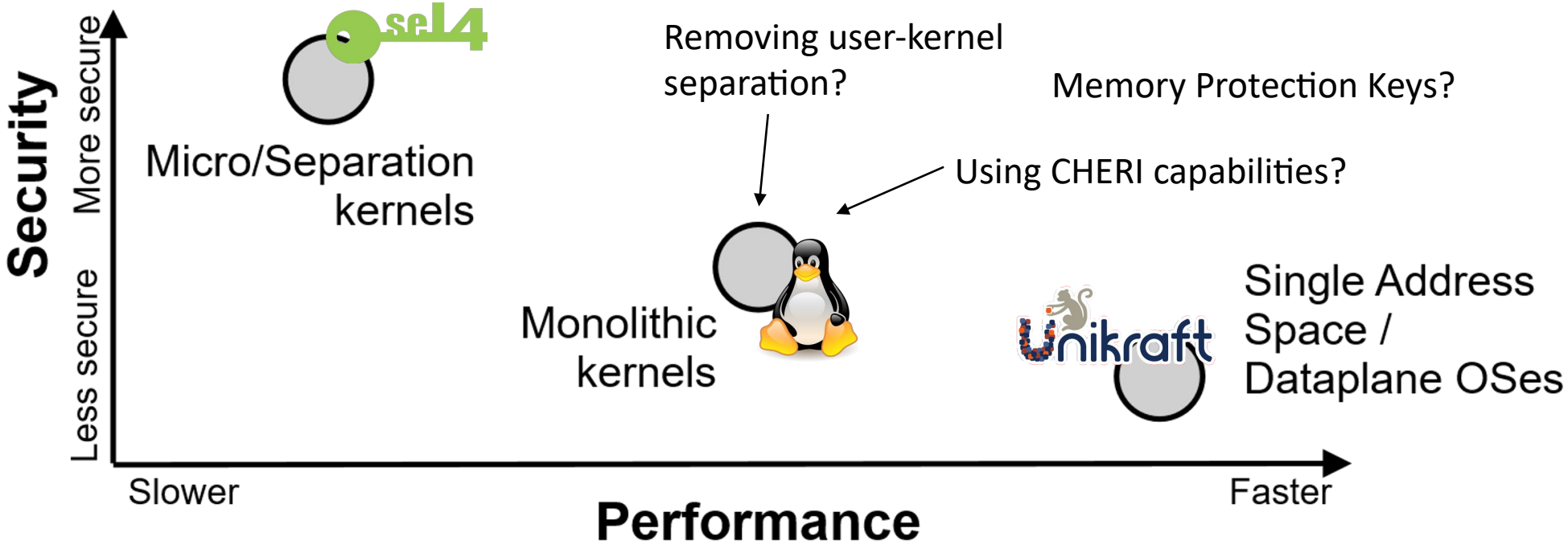
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)



Current OS Designs

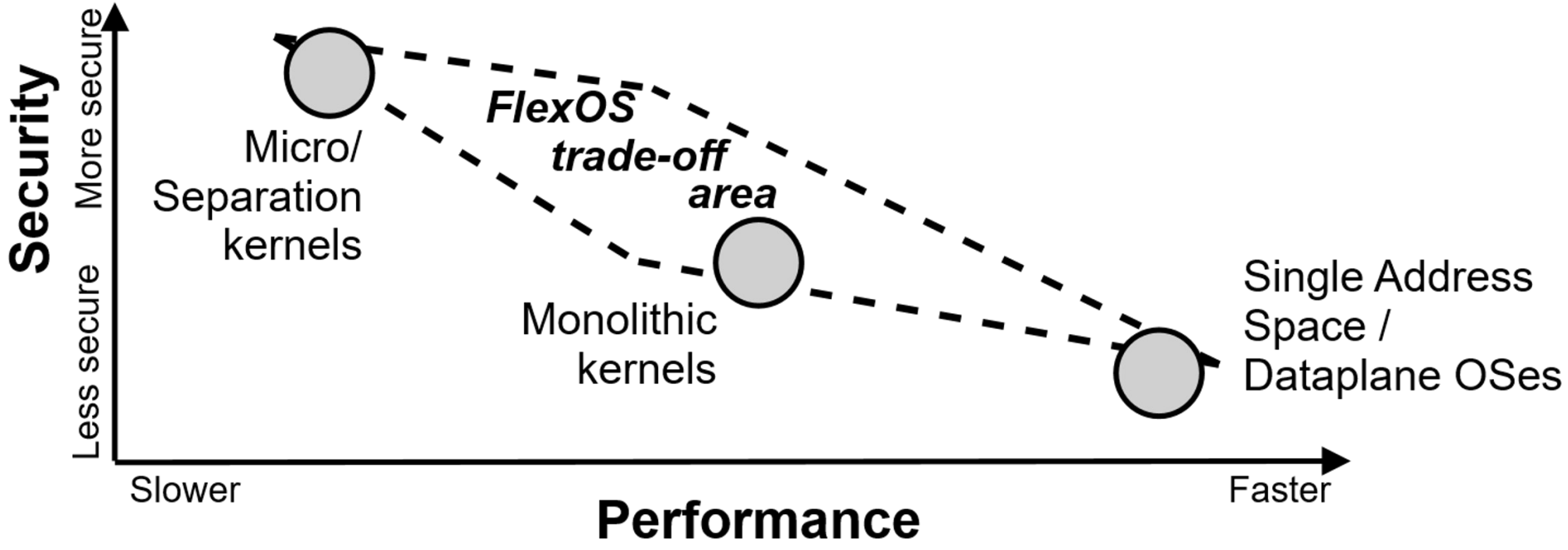
OS security/isolation strategies are **fixed** at design time!

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



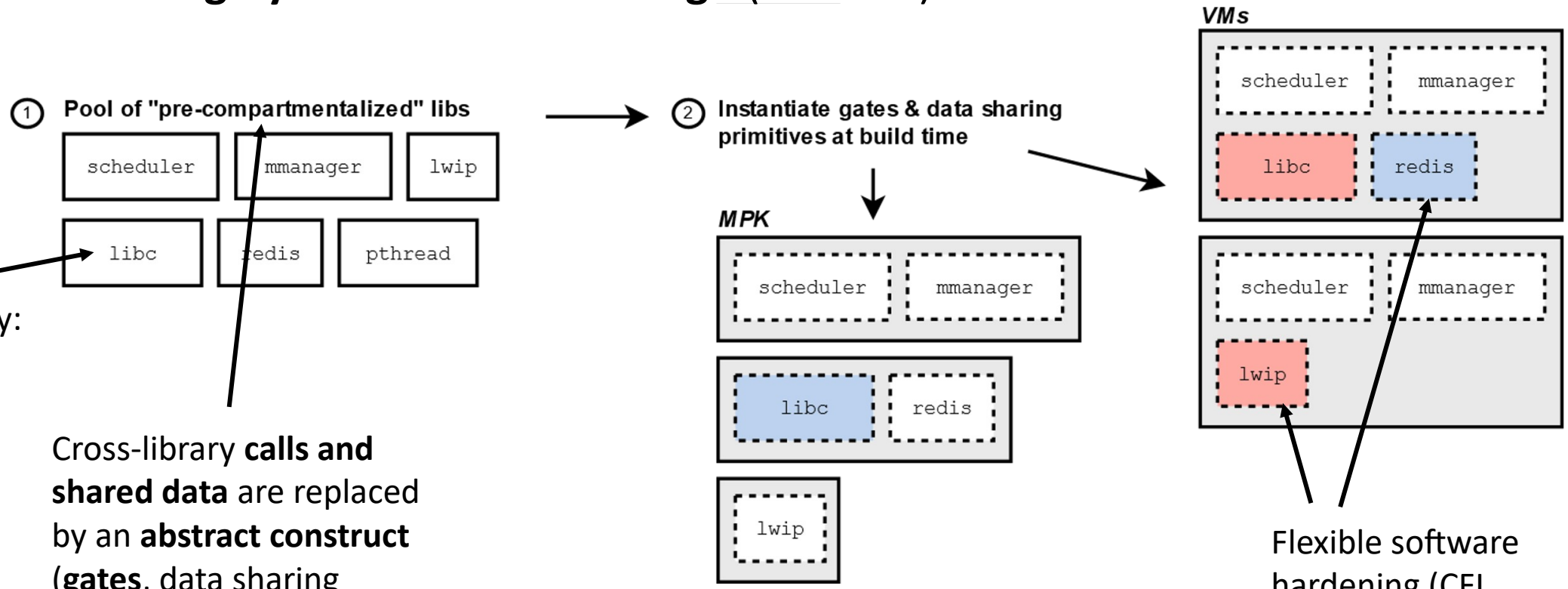
FlexOS: Flexible Isolation

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



FlexOS 101: Flexible Builds

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



Finest granularity: **a library**

Cross-library **calls and shared data** are replaced by an **abstract construct** (**gates**, data sharing primitives) defined as part of the **FlexOS API**

Flexible software hardening (CFI, SafeStack...)

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

Coccinelle

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

1 lwip

0 app

LLVM/clang

Replace with shared heap allocation

Replace with MPK gate

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

Coccinelle

LLVM/clang

lwip + app

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

Replace with shared heap allocation

Replace with MPK gate

FlexOS 101: Design Space

Huge design space! Not all configurations make sense from...

- a **security** perspective
- a **performance** perspective

Security perspective: how can we **guarantee that the properties of components hold**?

Enrich the OS with a tool that

1. selects configurations so that **properties hold**
2. **further prunes based on security and performance**

FlexOS 101: Do my properties hold?

Each component has a set of properties and expectations from other components!

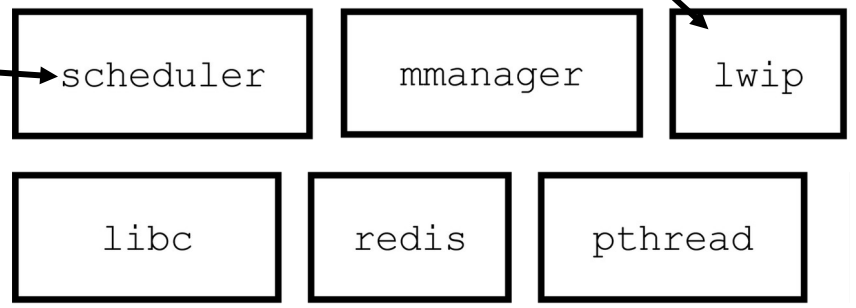
Describe them with a DSL

Plain-C NW stack

```
[Memory access] Read(*); Write(*)
[Call] *
```

Formally verified scheduler

```
[Memory access] Read(Own,Shared); Write(Own,Shared)
[Call] alloc::malloc, alloc::free
[API] thread_add (...); thread_rm(...); yield(...)
[Requires] *(Read,Own), *(Write,Shared),
*(Call, thread_add), *...
```



What is the component's API?

Requirements on other components?

What memory can this component access?

What functions can this component call?

FlexOS 101: Do my properties hold?

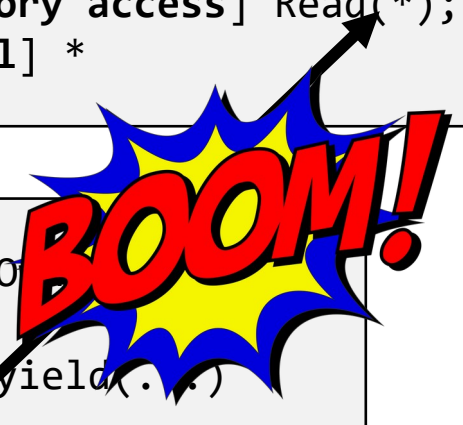
Using the DSL, we can determine **pair-wise compatibility such that properties hold**

Plain-C NW stack

```
[Memory access] Read(*); Write(*)
[Call] *
```

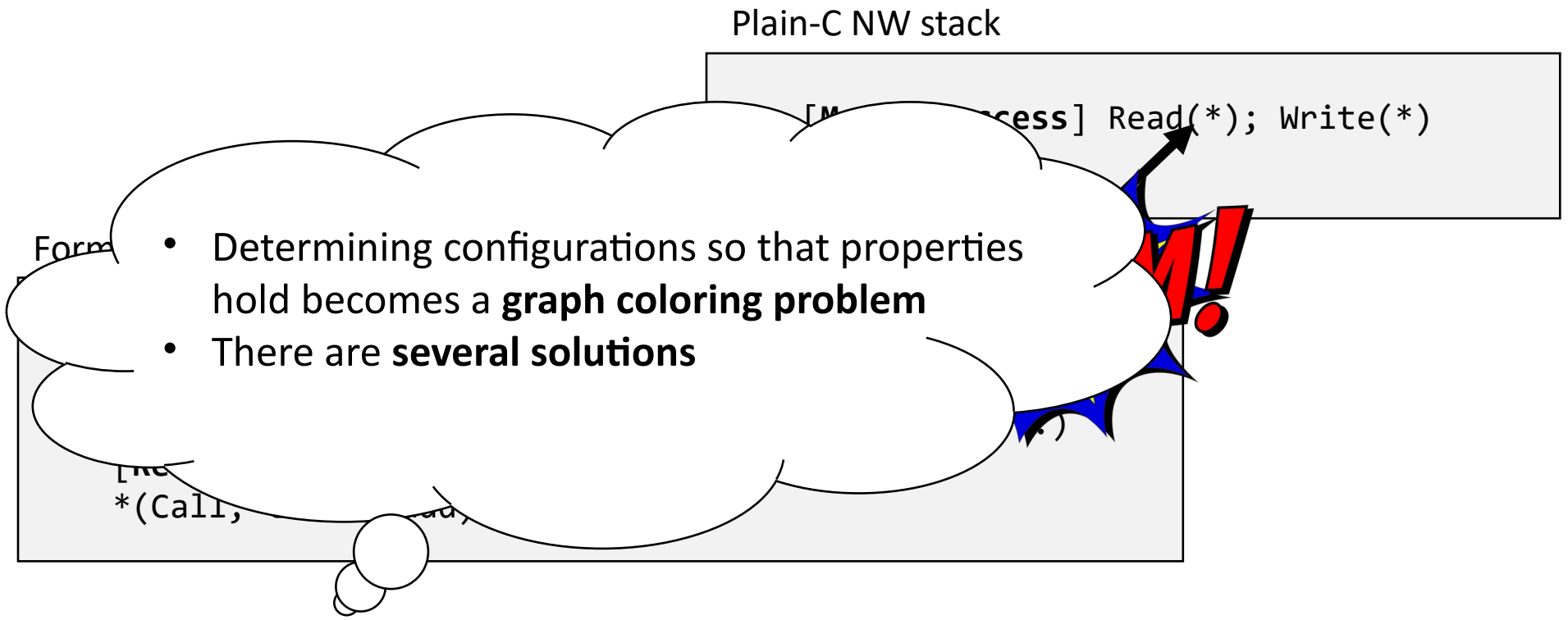
Formally verified scheduler

```
[Memory access] Read(Own,Shared); Write(O
[Call] alloc::malloc, alloc::free
[API] thread_add (...); thread_rm(...); yield(...)
[Requires] *(Read,Own), *(Write,Shared),
*(Call, thread_add), *...
```



FlexOS 101: Do my properties hold?

Using the DSL, we can determine **pair-wise compatibility such that properties hold**



Prototype

Implementation **on top of Unikraft**

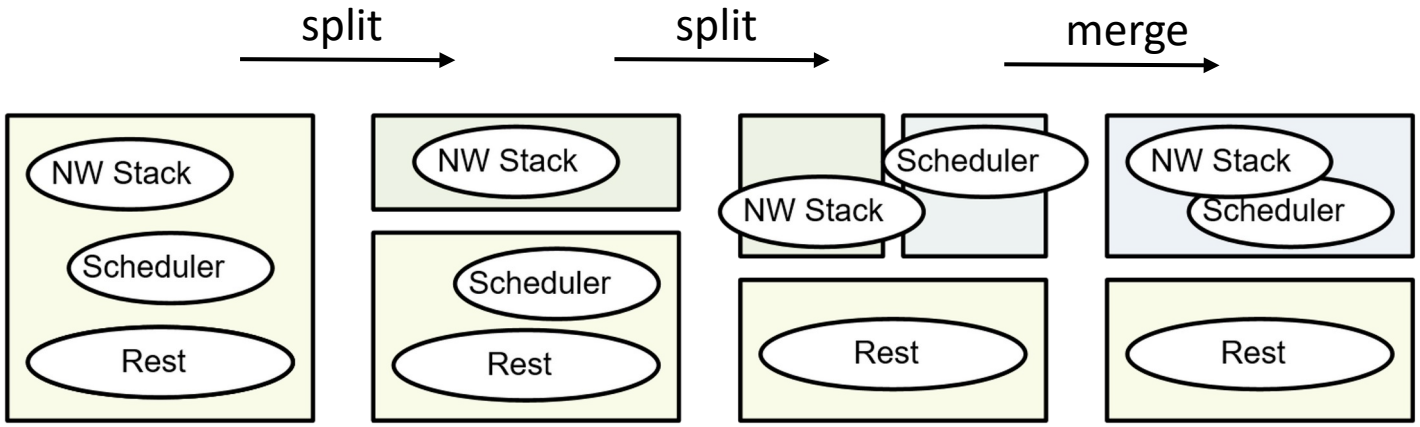
Gate implementations for **Intel MPK** and **VM/EPT**

Port of the **network stack** (lwip) and **scheduler** (uksched)

Prototype

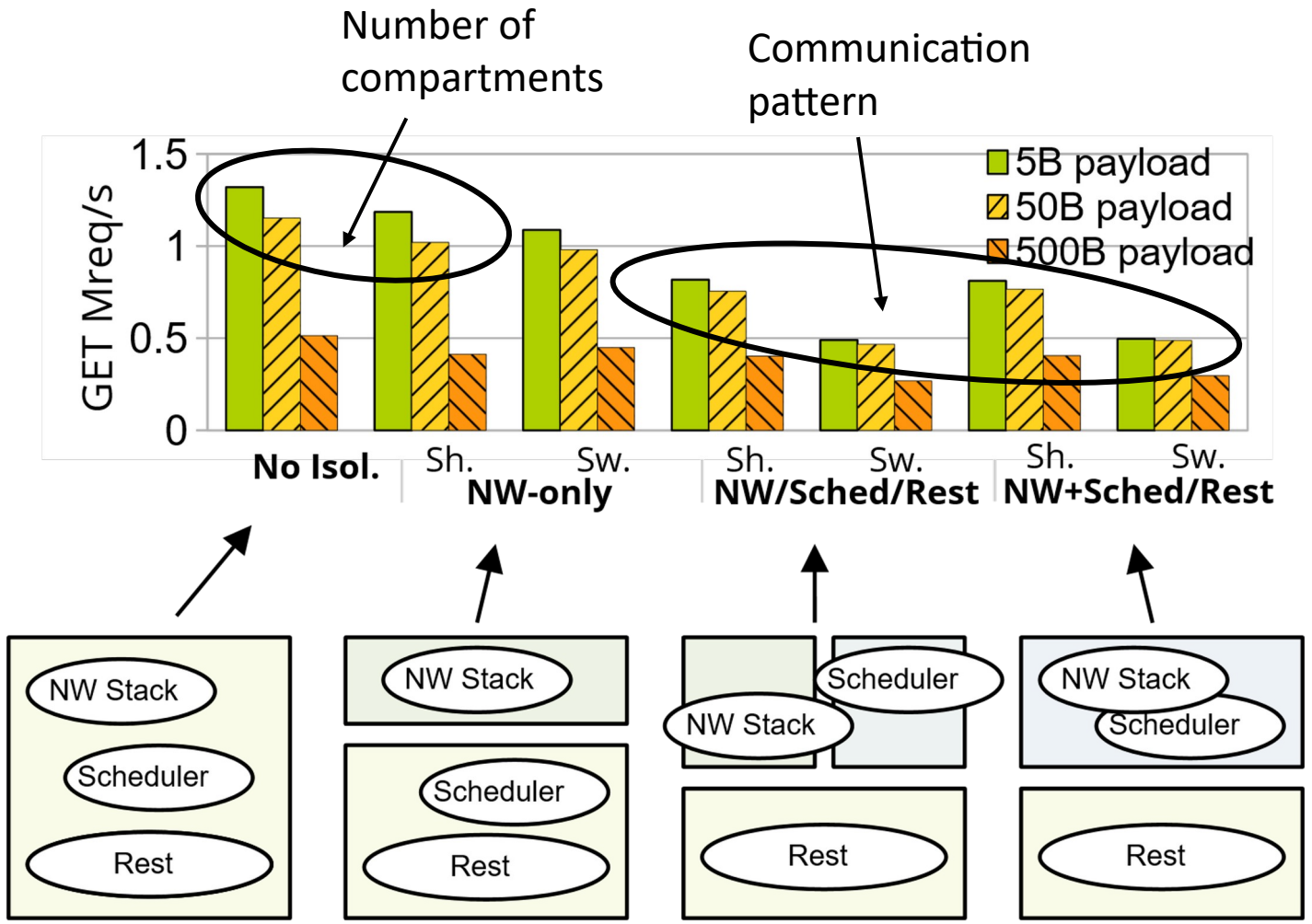
Here: **MPK and Redis**

4 configurations, up to 3 compartments



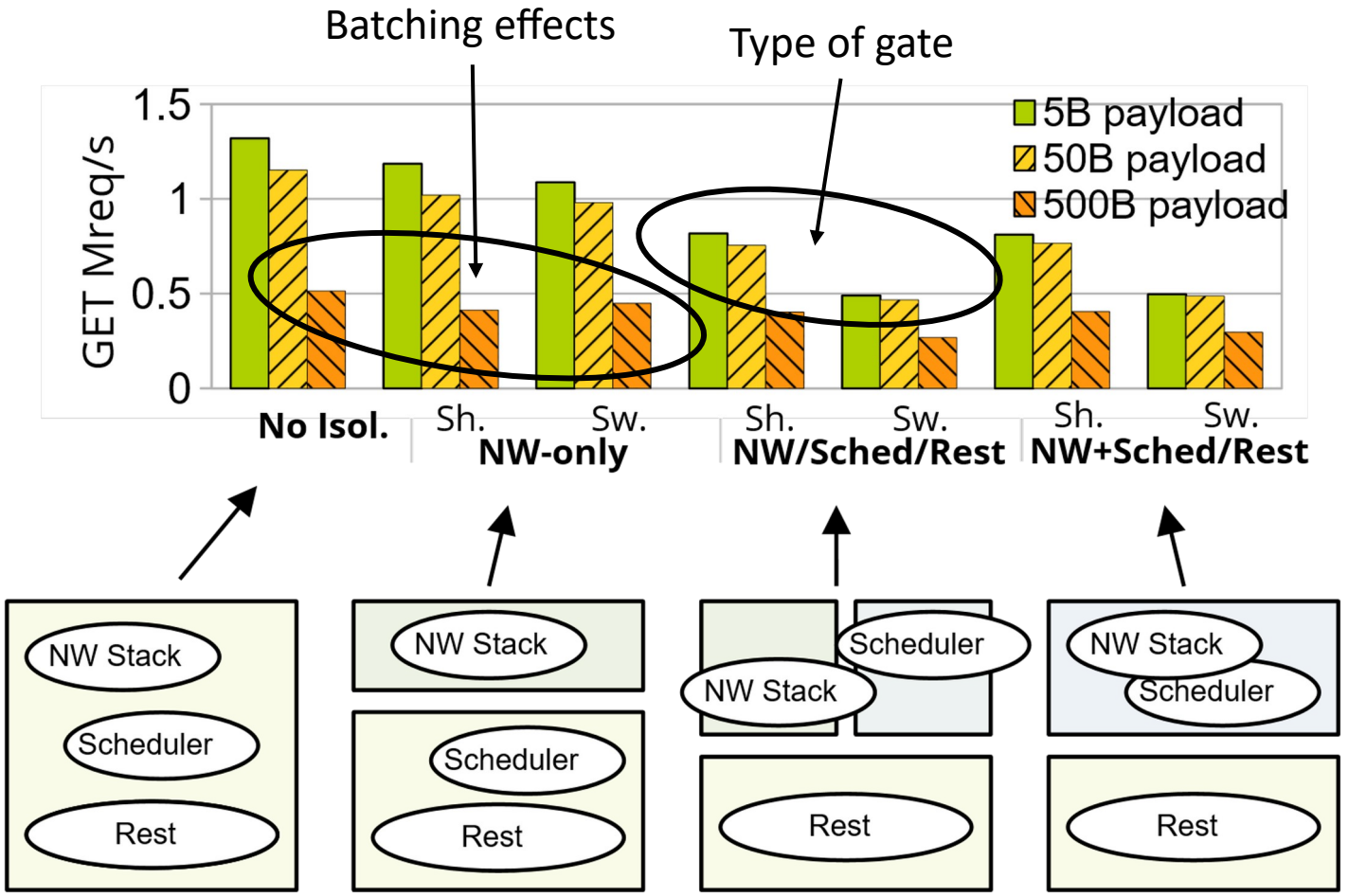
Prototype

Here: **MPK and Redis**

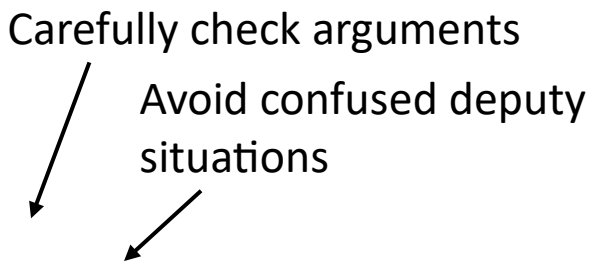


Prototype

Here: **MPK and Redis**



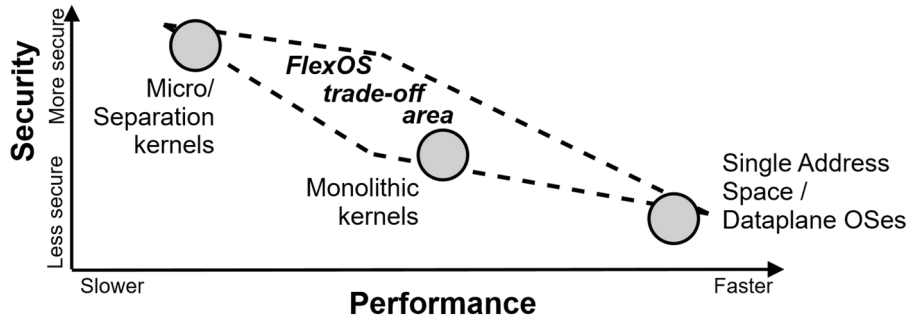
Some open questions...



1. **Security is not only a matter of isolation**
   - When APIs are **designed as a trust boundary,** swapping the isolation mechanism is easy!
   - *What if the API wasn't developed as a trust boundary?*

2. **How to minimize porting effort?**
   - FlexOS requires porting of app/kernel libraries
   - *How to automate the process?*
   - *How to make sure that DSL metadata is correct?*

In a Nutshell



There is a **need for isolation flexibility**

- OS Specialization, hardware heterogeneity
- or even breaking primitives!

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

Decouple isolation from the OS design:

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