# The Composite Component-Based Operating System

**Gabriel Parmer**

Based on research with:
Runyu Pan, Yuxin Ren, Phani Kishore Gadepalli,
Wenyuan Shao, Qi Wang, Jiguo Song, and many others

The George Washington University
gparmer@gwu.edu

# Requirements Convergence

Embedded Systems

Predictability
Simplicity
SWaP-C

Multi-tenant Cloud

Performance
Isolation
Elasticity

# Requirements Convergence

**Embedded Systems**

Predictability
Simplicity
SWaP-C

→

**Predictability/Tail-latency**
**Performance**
**Isolation & Security**

←

**Multi-tenant Cloud**

Performance
Isolation
Elasticity

# Requirements Convergence

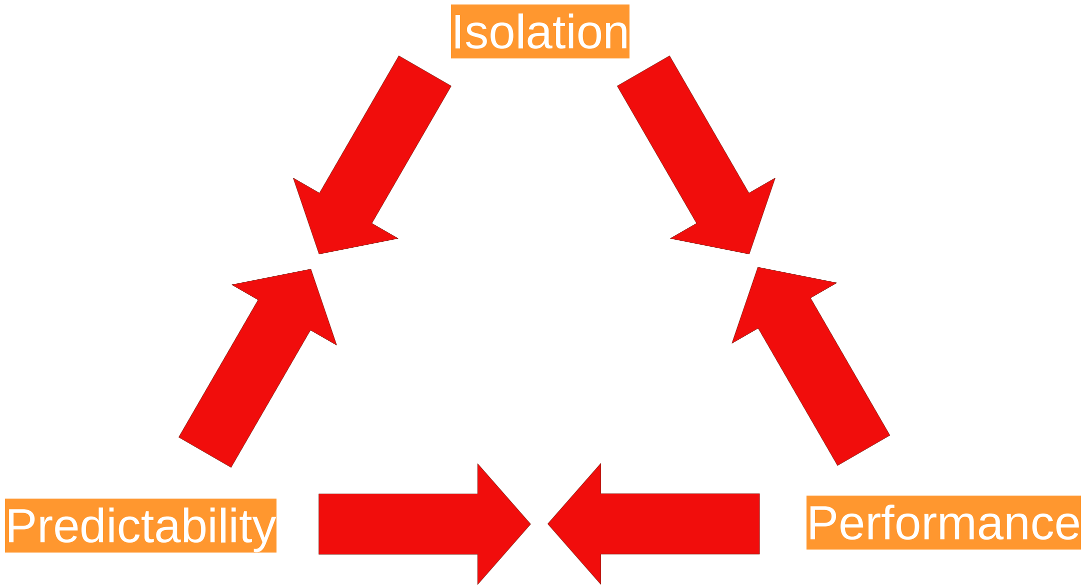
**Embedded Systems**

Predictability
Simplicity
SWaP-C

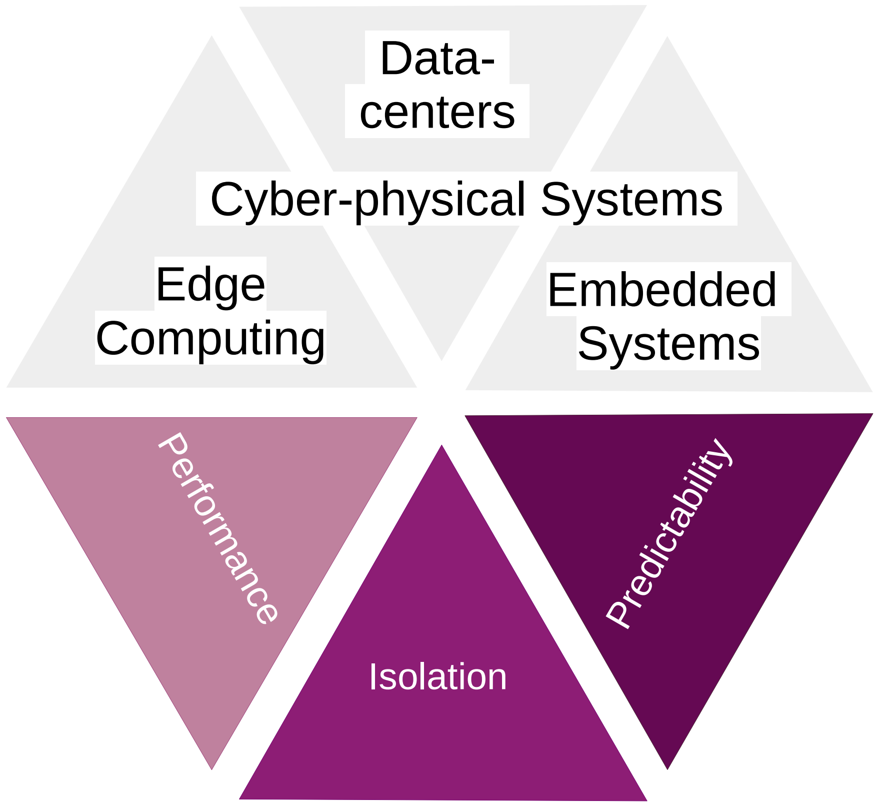
**Predictability/Tail-latency**
**Performance**
**Isolation & Security**

**Multi-tenant Cloud**

Performance
Isolation
Elasticity

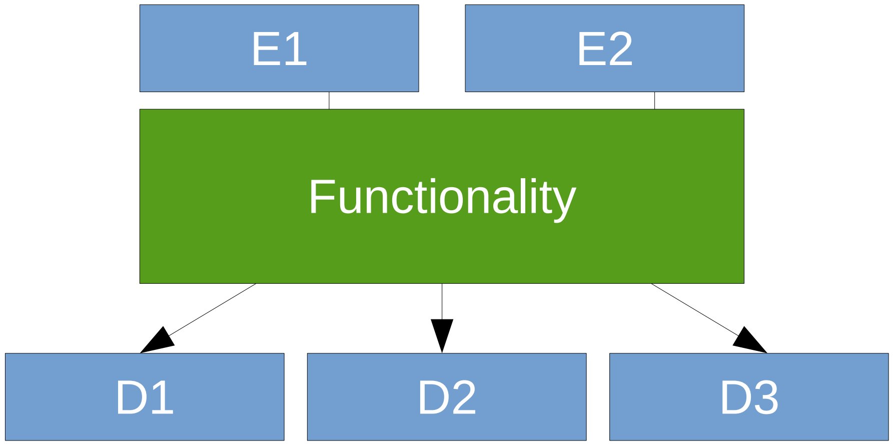


# Requirements Convergence



# Component-based System Design

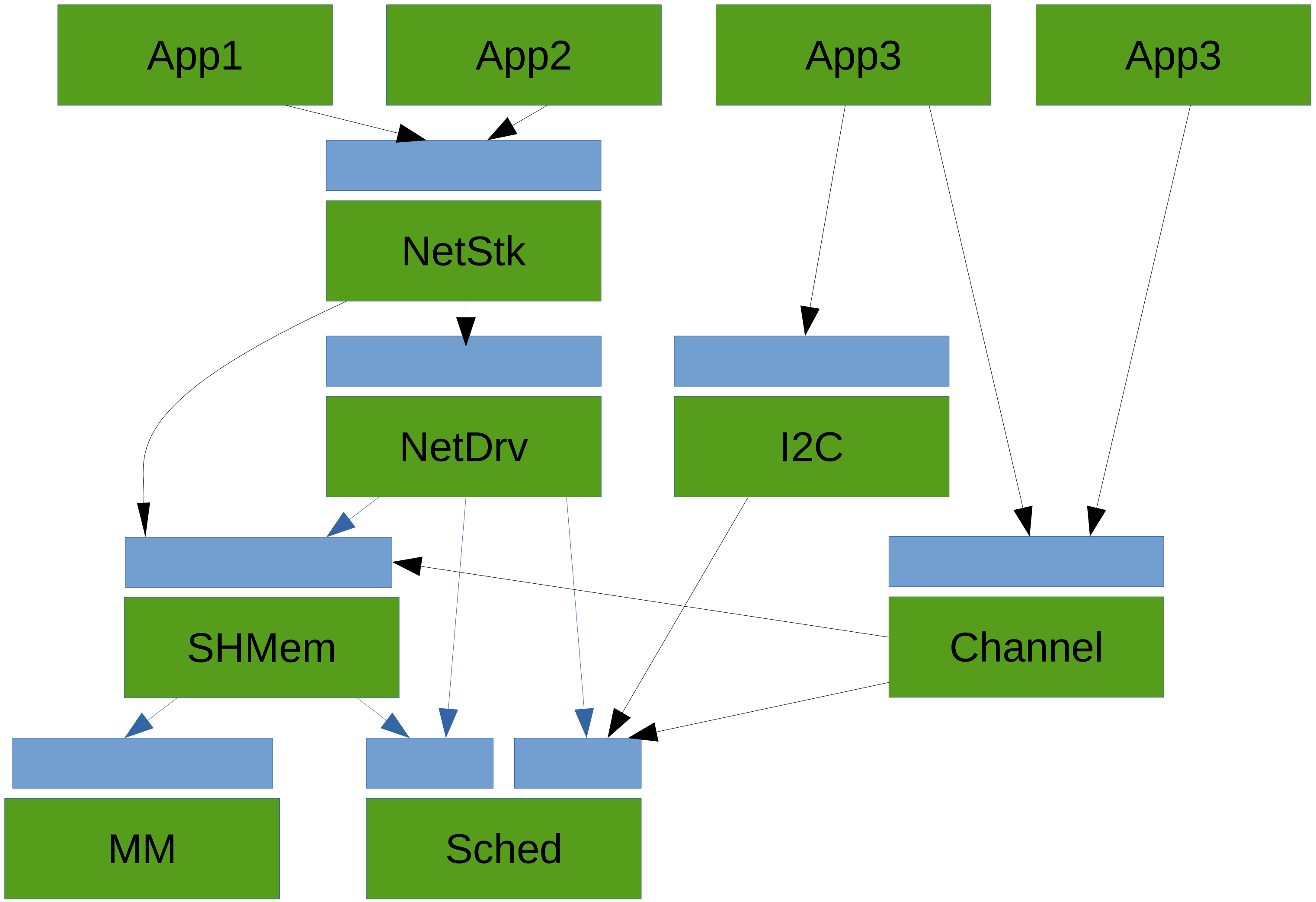
- Code, data, user-level
- Export APIs (`E1 = {fn, ...}`)
- Explicit dependencies
- Unit of reuse & isolation




- *Minimize* functionality for the *necessary APIs*
- Strong, *fine-grained isolation*

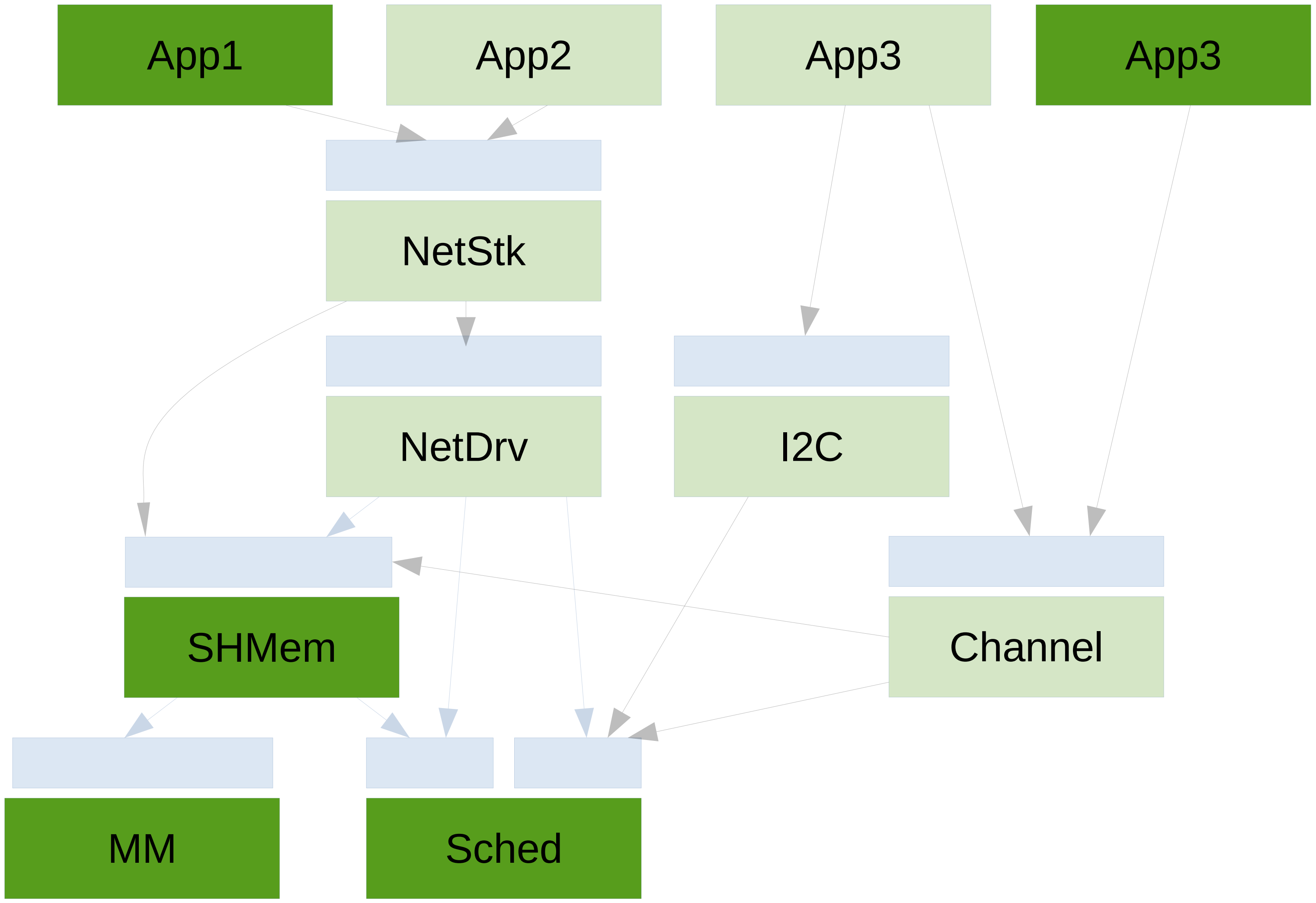
# Component-based System Design

- System *composed* from components
- Limit *scope* of
  - compromises
  - faults
  - unpredictability



# Component-based System Design

- System *composed* from components
- Limit *scope* of
  - compromises
  - faults
  - unpredictability



# Guidance for µ-Kernel Design

”...a concept is *tolerated inside the µ-kernel* only if *moving it outside the kernel*, i.e. permitting competing implementations, would **prevent the implementation of the system’s required functionality**.”

- Liedtke '95

# Guidance for µ-Kernel Design

”...a concept is *tolerated inside the µ-kernel* only if *moving it outside the kernel*, i.e. permitting competing implementations, would **prevent the implementation of the system’s required functionality**.”

- Liedtke '95

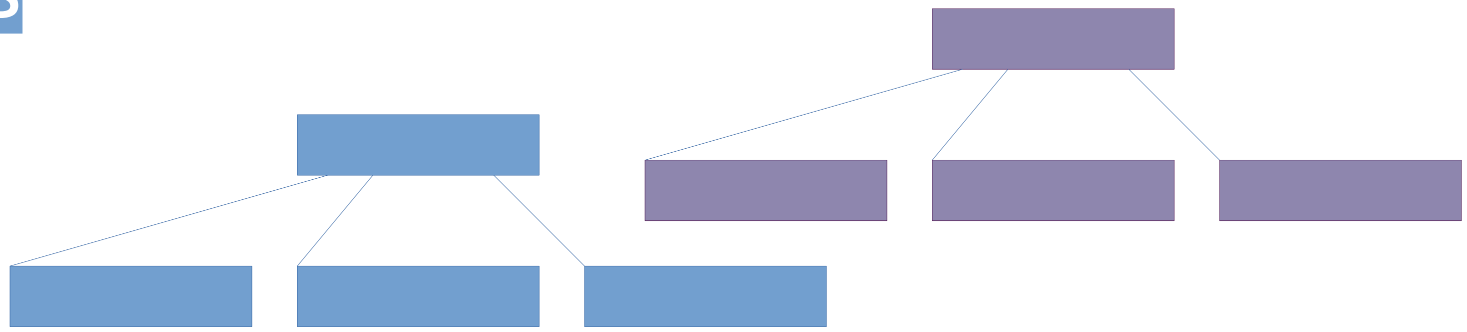
*Composite*: push this to the extreme w/ component-defined

- scheduling
- parallel scalability
- concurrency
- capability delegation/revocation
- ...

# Composite Kernel Objects

Capability-table nodes

Page-table nodes

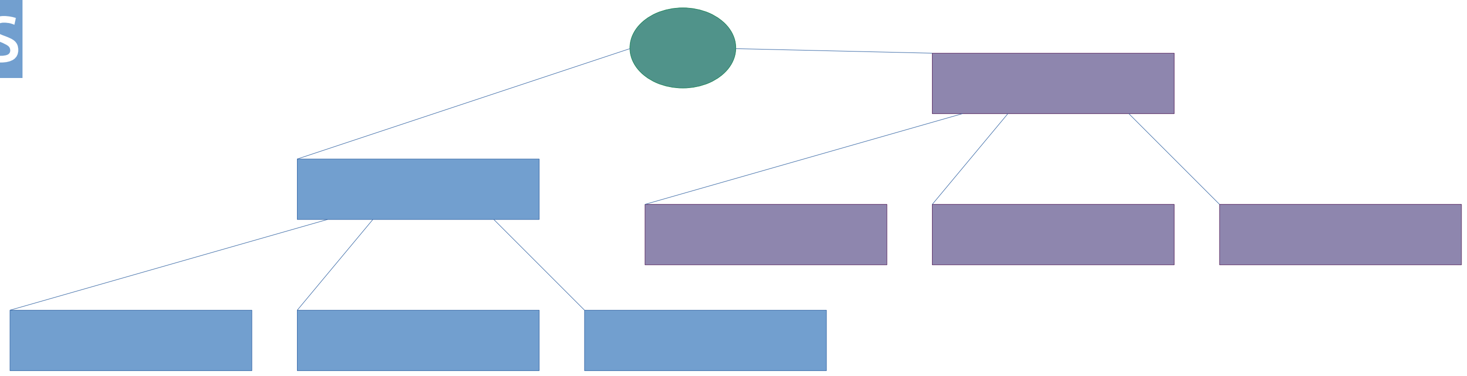


# Composite Kernel Objects

Capability-table nodes

Page-table nodes

Components



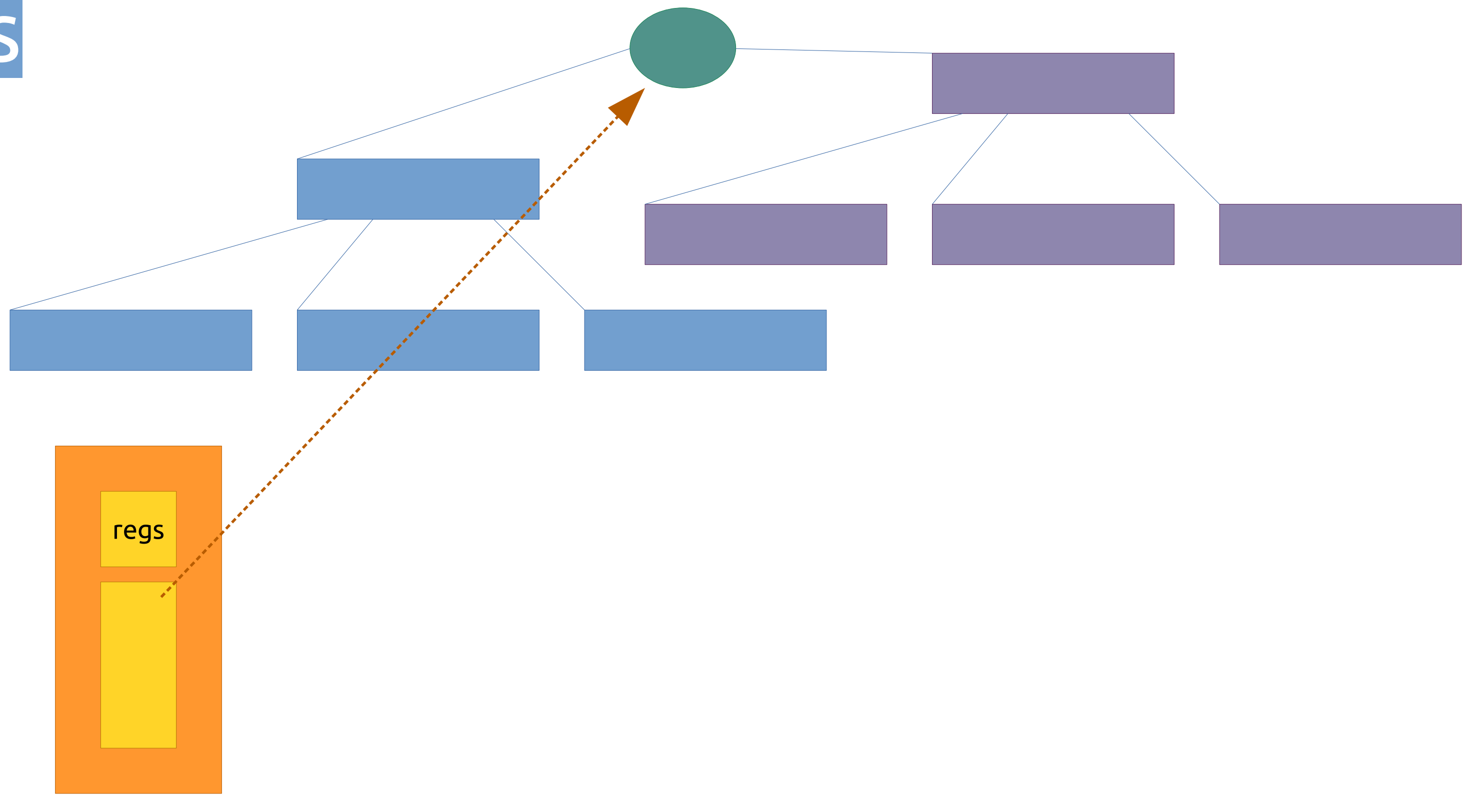
# Composite Kernel Objects

Capability-table nodes

Page-table nodes

Components

Threads



# Composite Kernel Objects

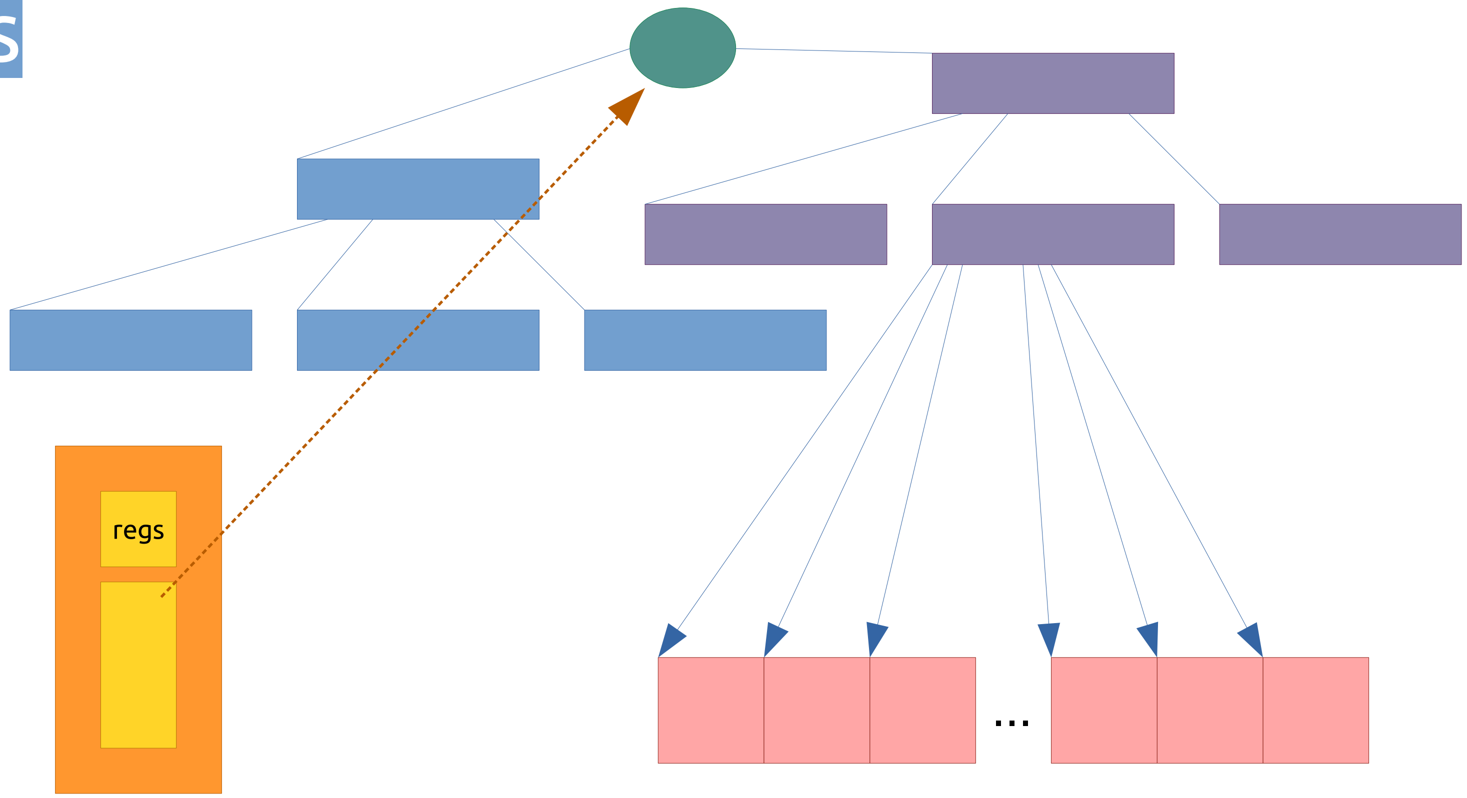
Capability-table nodes

Page-table nodes

Components

Threads

Untyped Memory



# Composite Kernel Objects

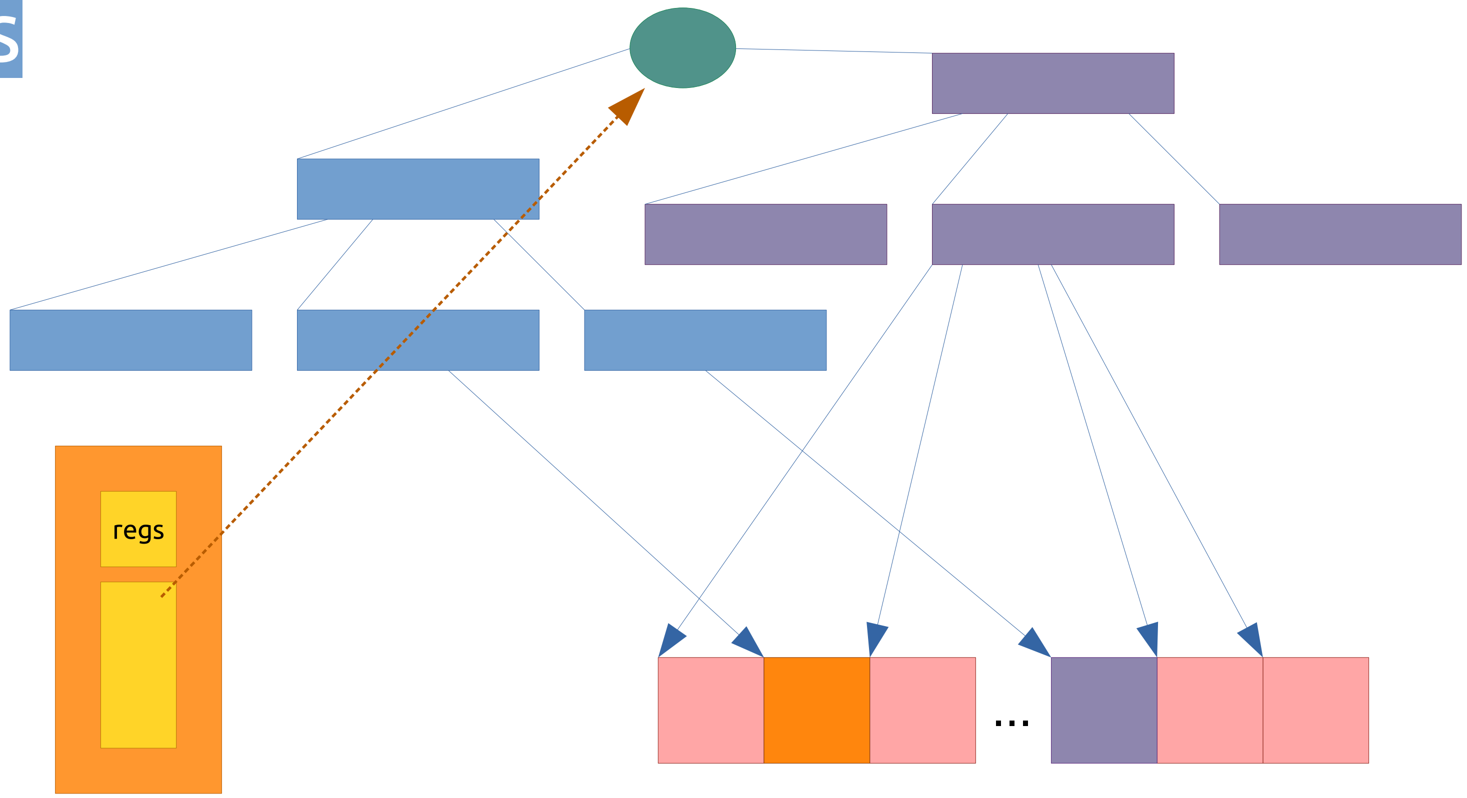
Capability-table nodes

Page-table nodes

Components

Threads

Untyped Memory



# Composite Kernel Objects

Capability-table nodes

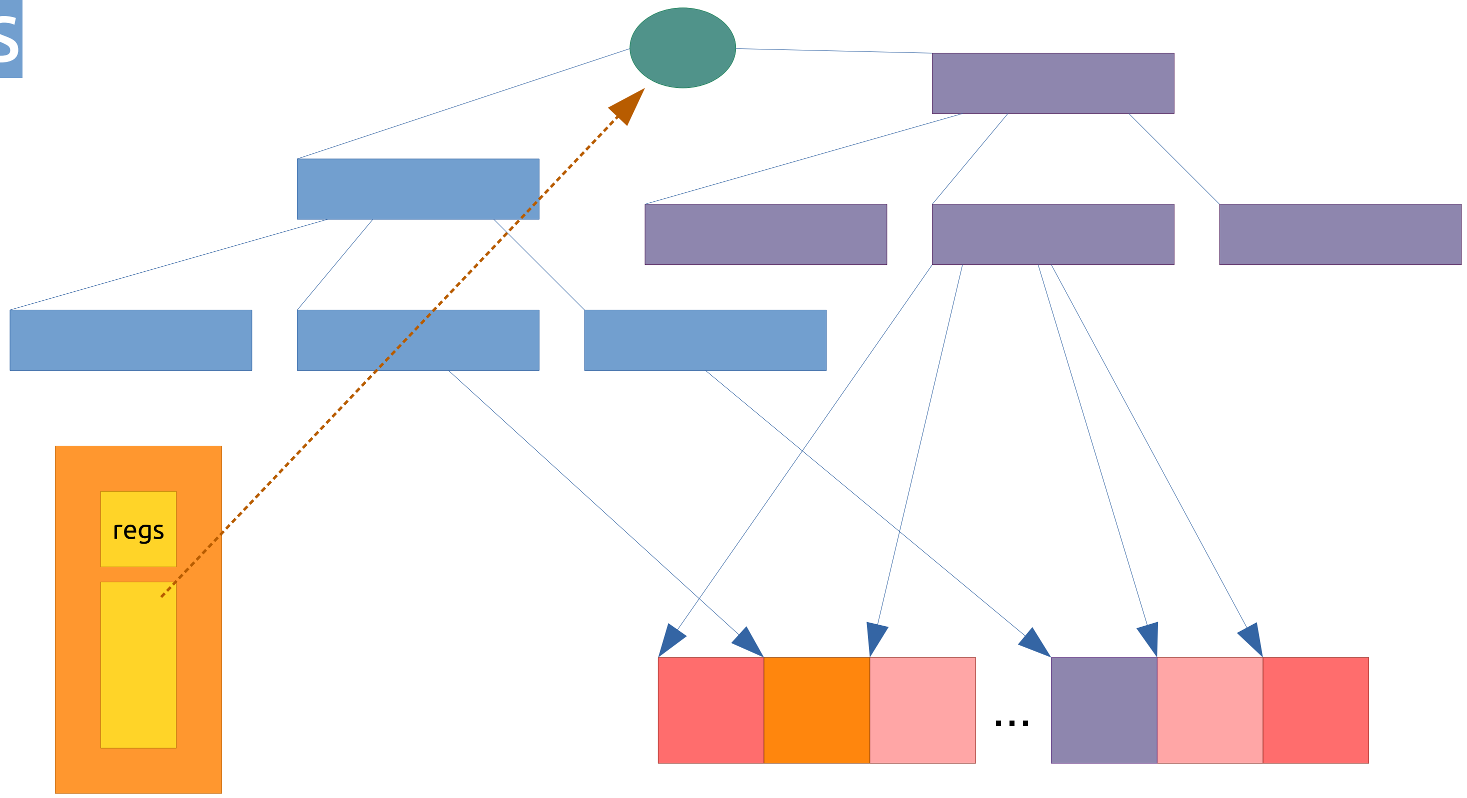
Page-table nodes

Components

Threads

Untyped Memory

Virtual Memory



# Composite Kernel Objects

Capability-table nodes

Page-table nodes

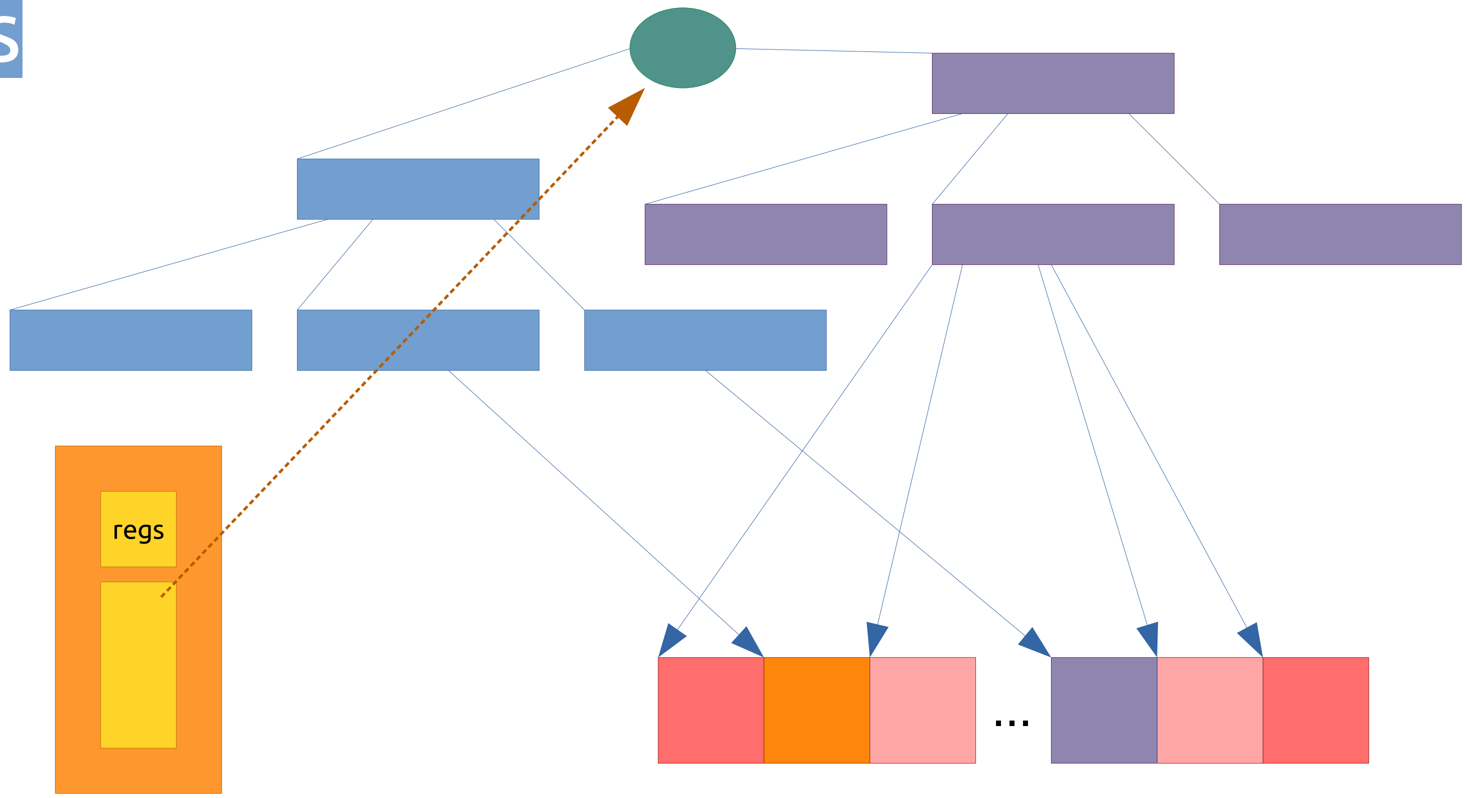
Components

Threads

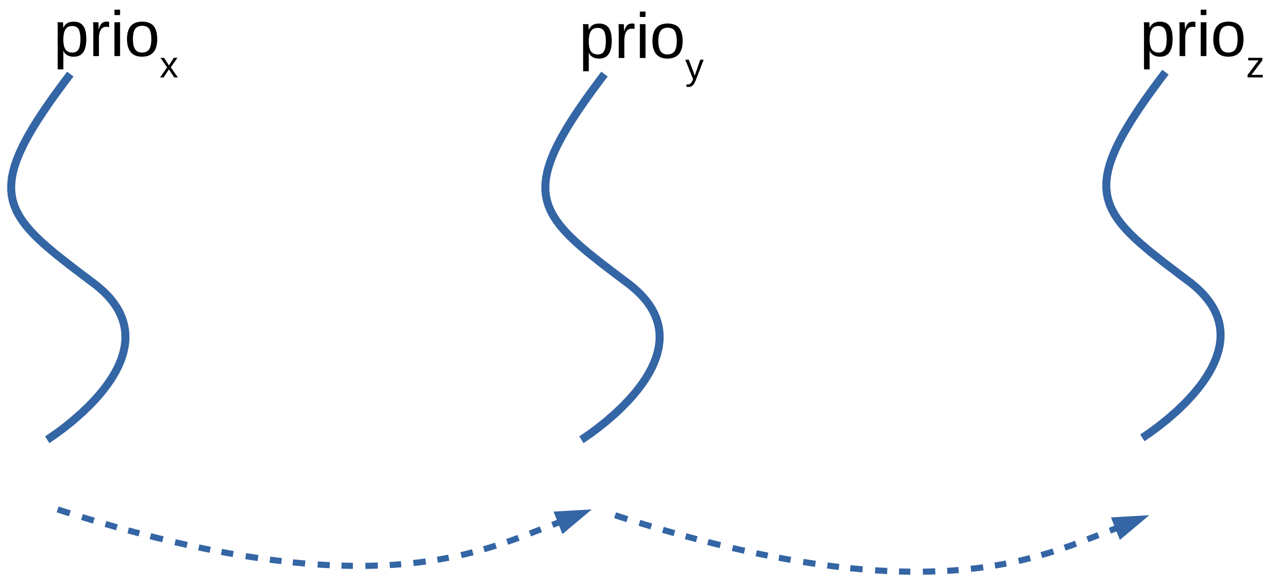
Untyped Memory

Virtual Memory

IPC: sync & async

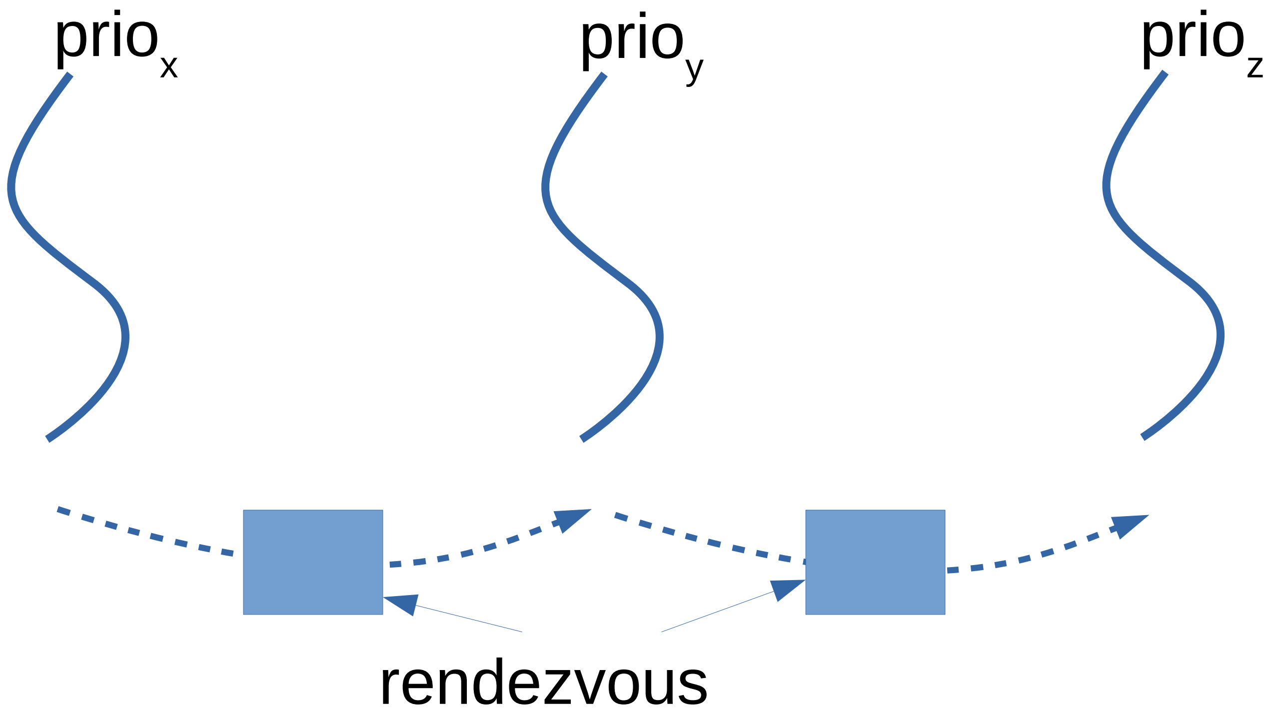


# Asynchronous IPC




End-to-end timing requires dependency analysis

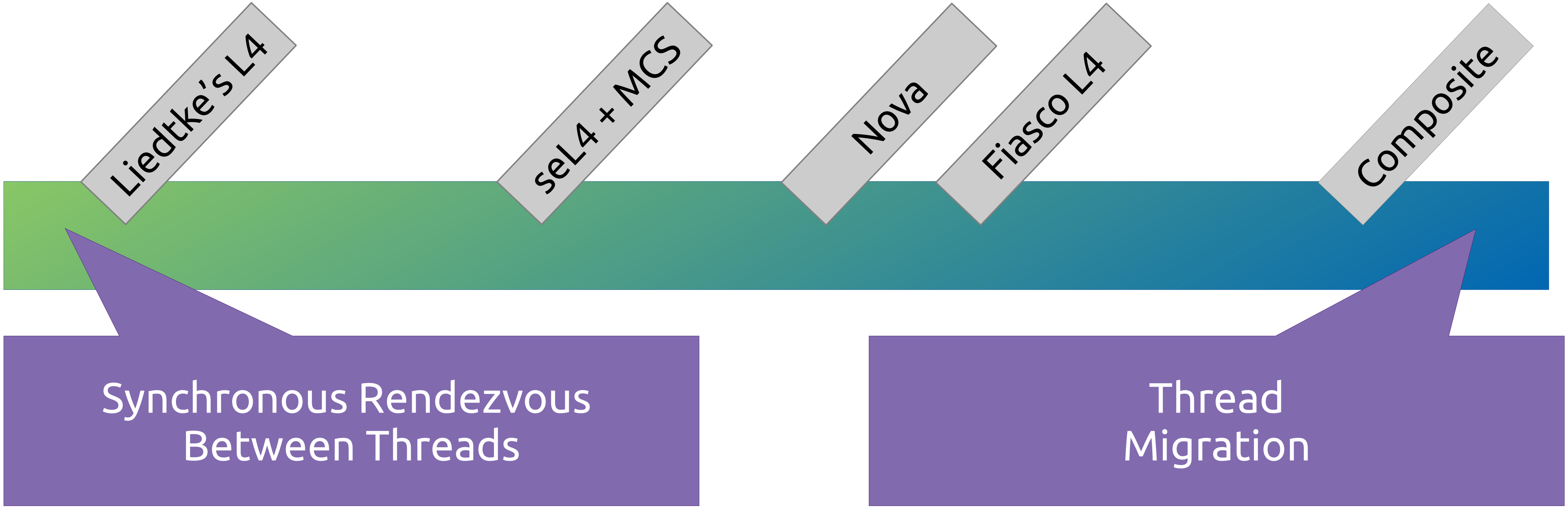
# Synchronous IPC between Threads




## End-to-end timing based on kernel mechanims

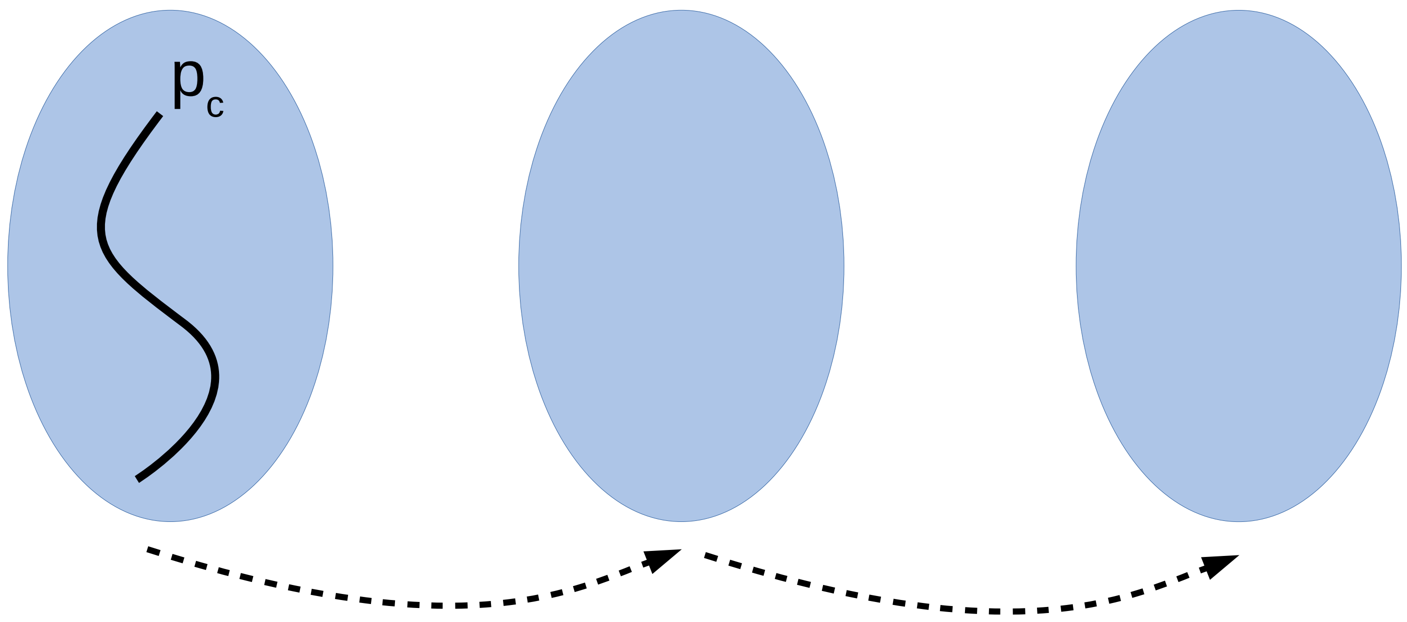
- In-kernel priority inheritance machinery, ceiling w/ limited prio
- Budget management and experation

# Spectrum of IPC Mechanisms

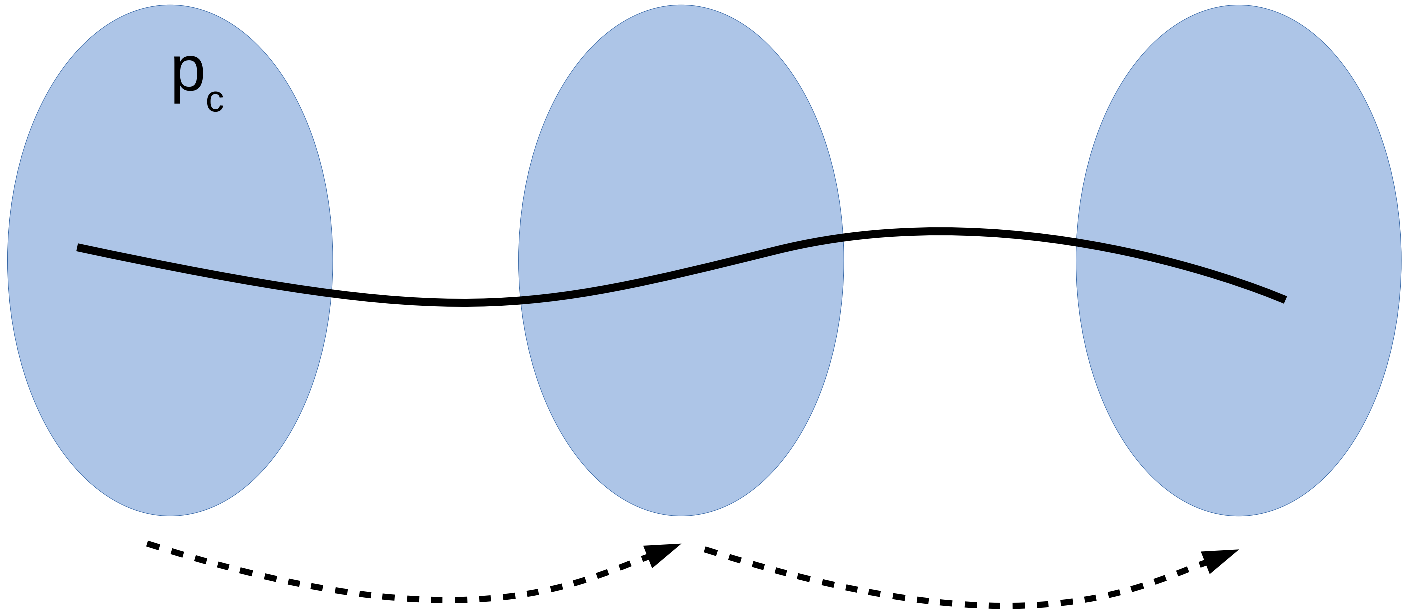



Ford et al. *Evolving Mach 3.0 to a Migrating Thread Model*, USENIX Winter '94
Gabber et al. *The Pebble Component-Based Operating System*, USENIX ATC '99

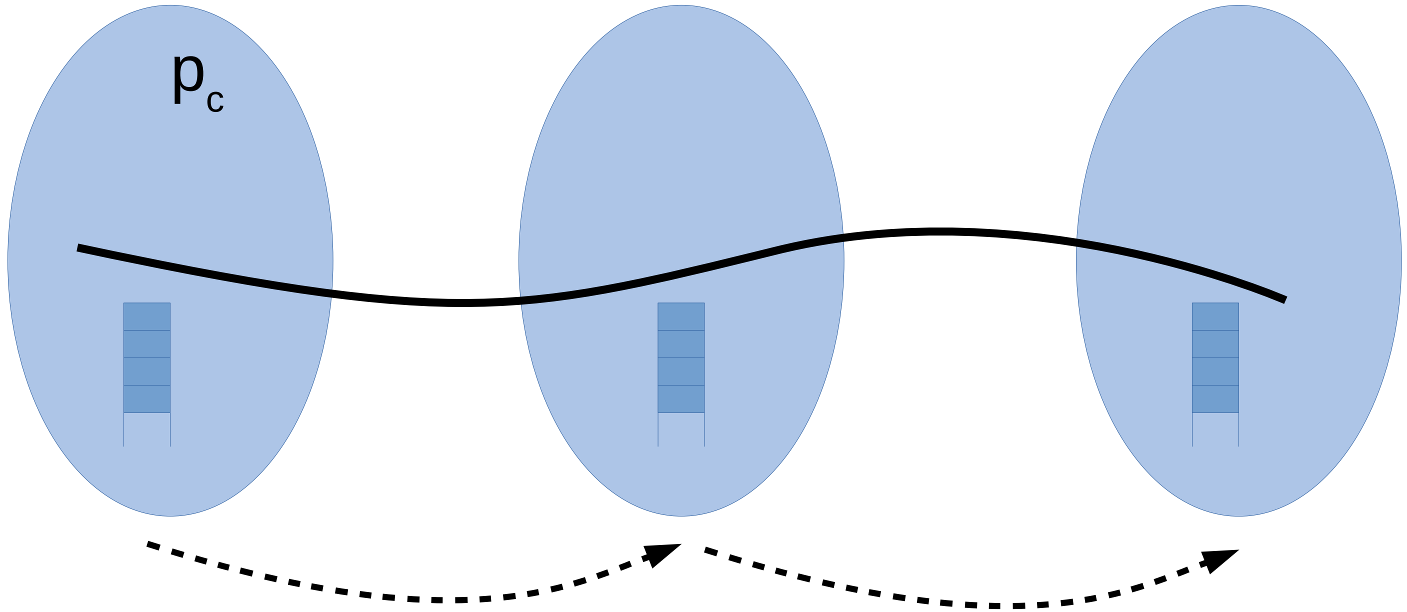
# Thread Migration



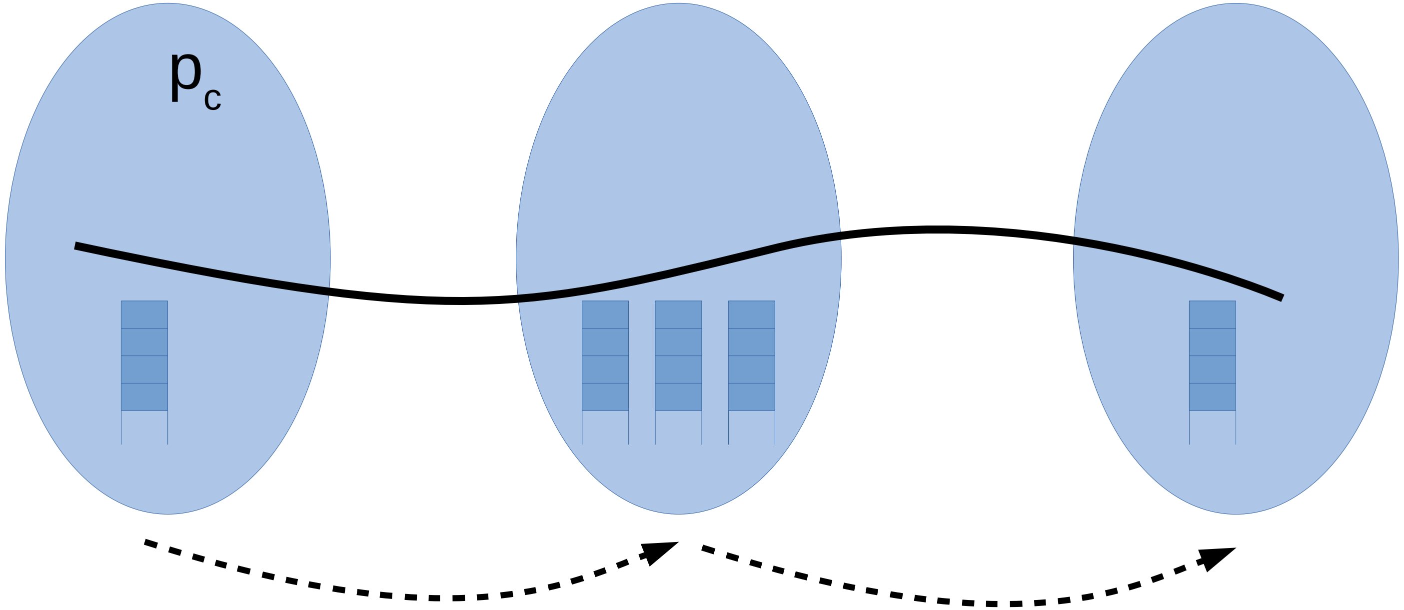
# Thread Migration



# Thread Migration

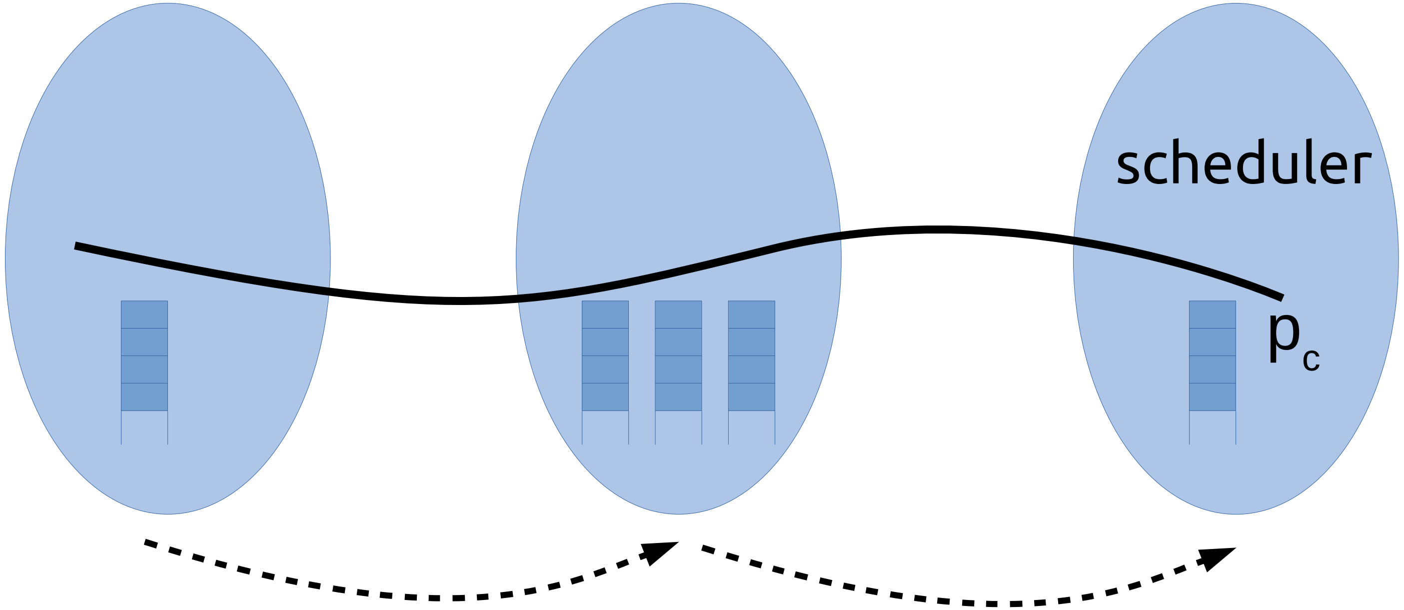


# Thread Migration

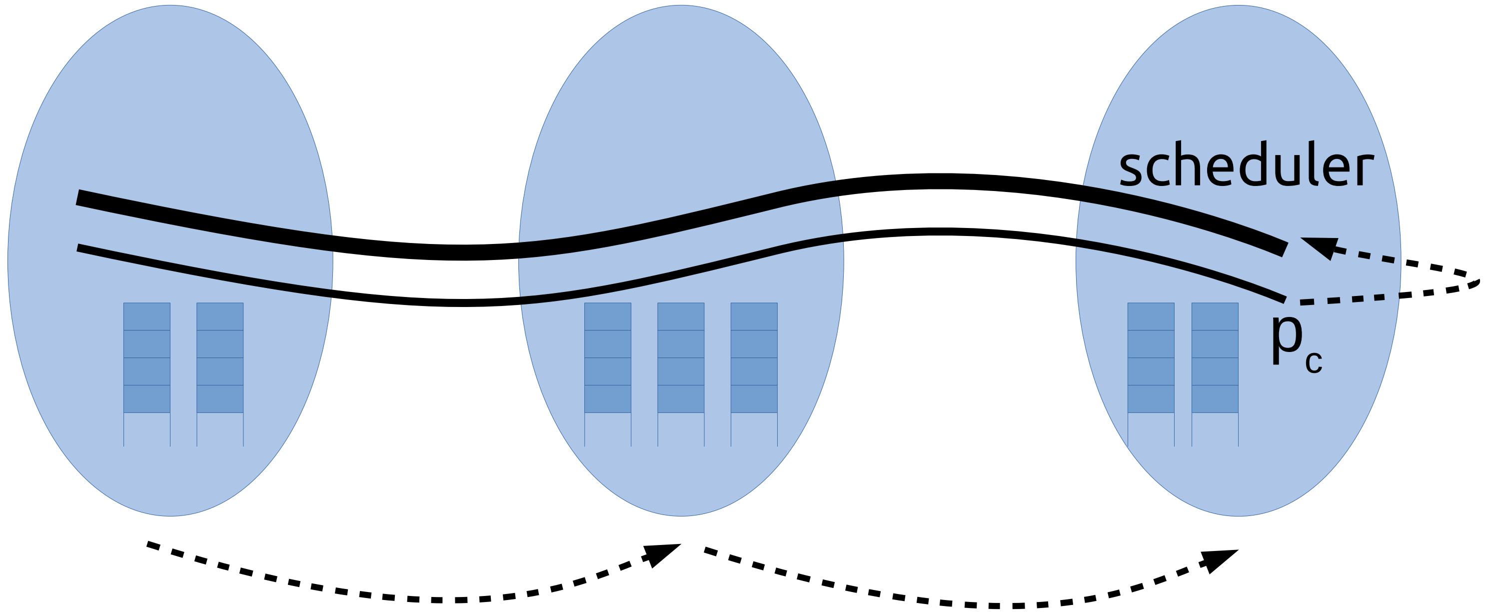



Wang et al. *Execution Stack Management for Hard Real-Time Computation in a Component-Based OS*, RTSS '11

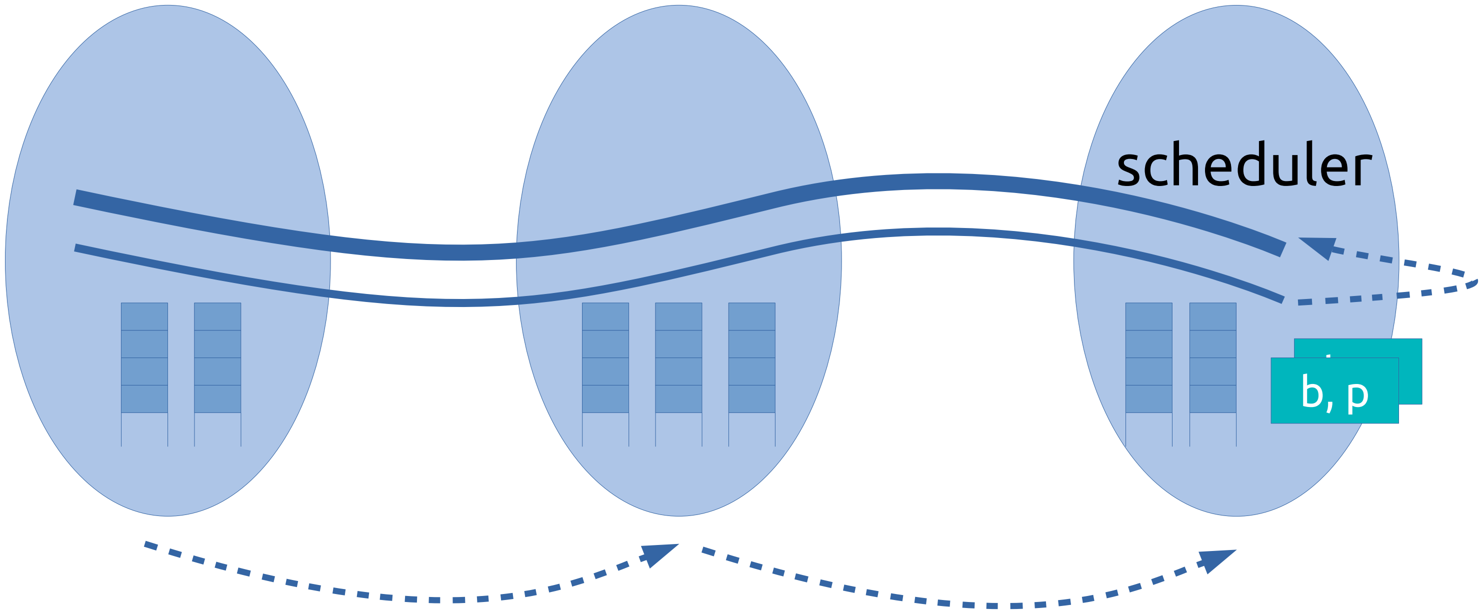
# Thread Migration



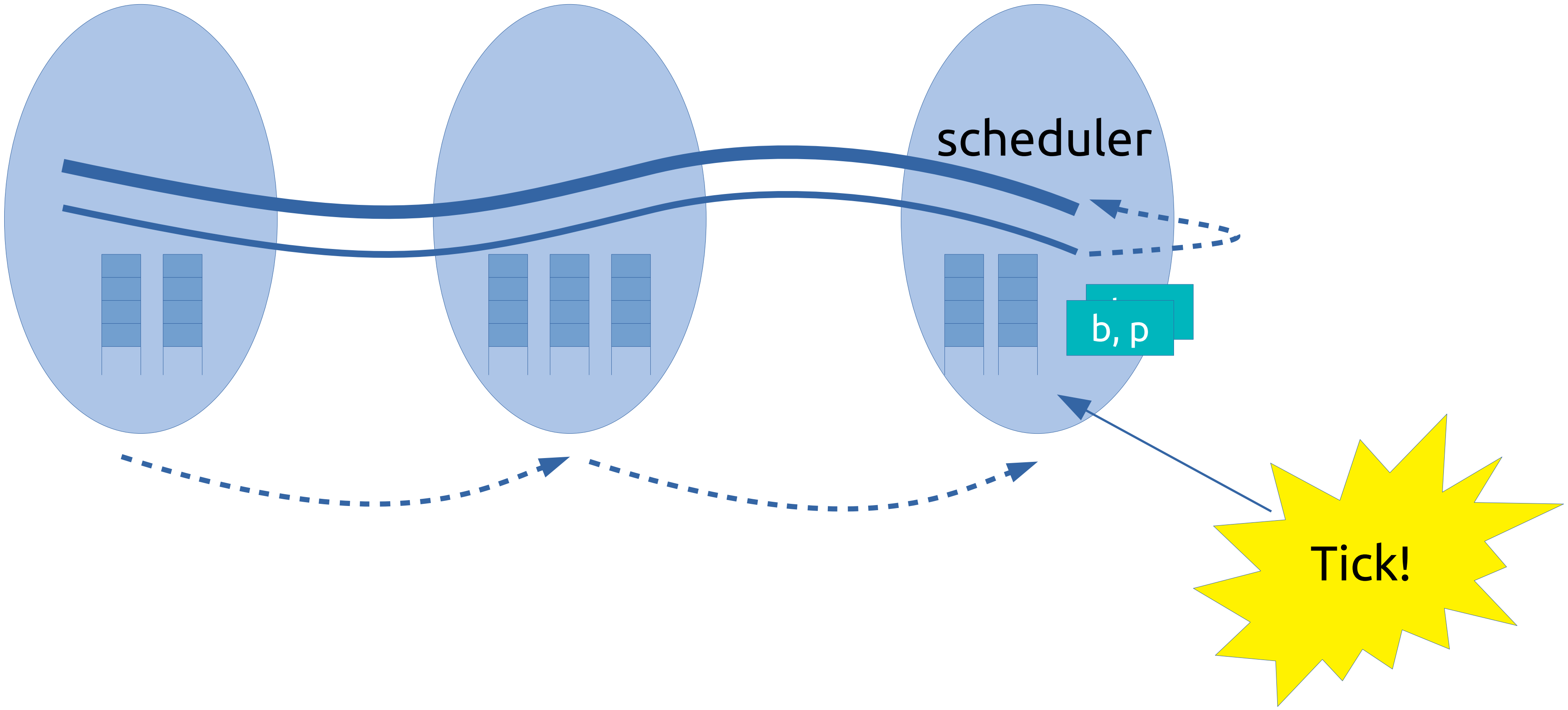
# Thread Migration



# Thread Migration

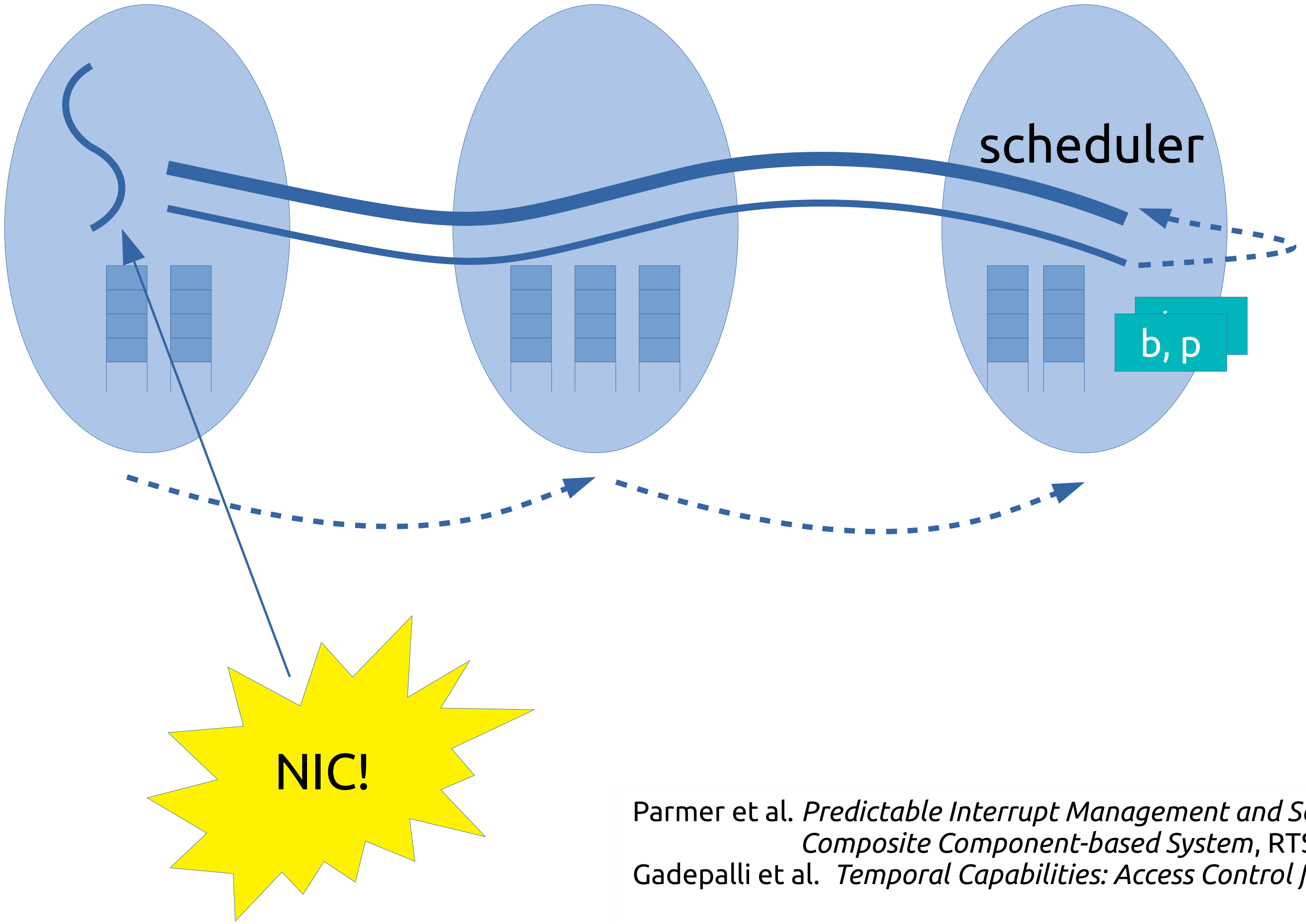


# Thread Migration



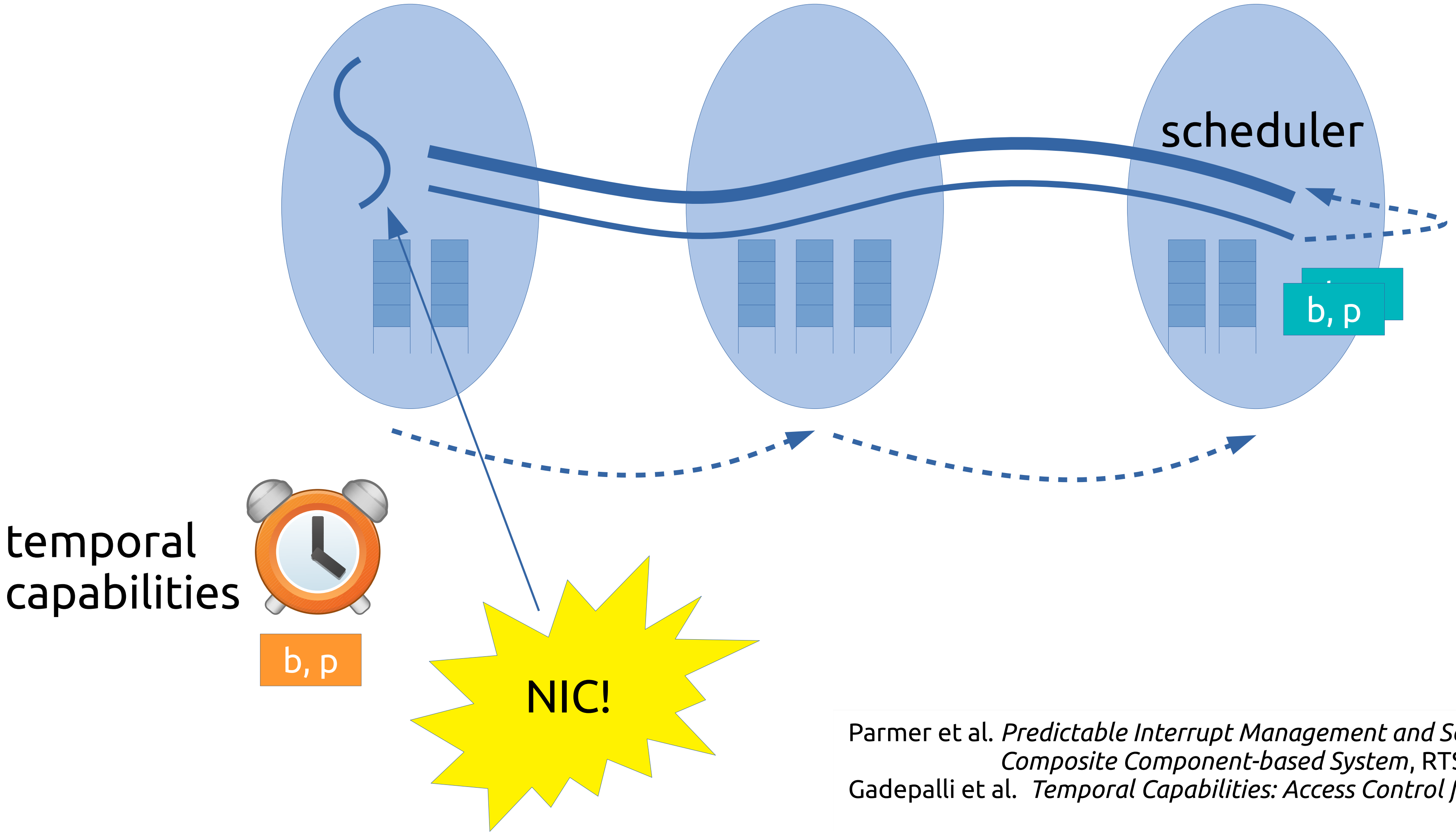

Parmer et al. *Predictable Interrupt Management and Scheduling in the Composite Component-based System*, RTSS '08
Gadepalli et al. *Temporal Capabilities: Access Control for Time*, RTSS '17

# Thread Migration



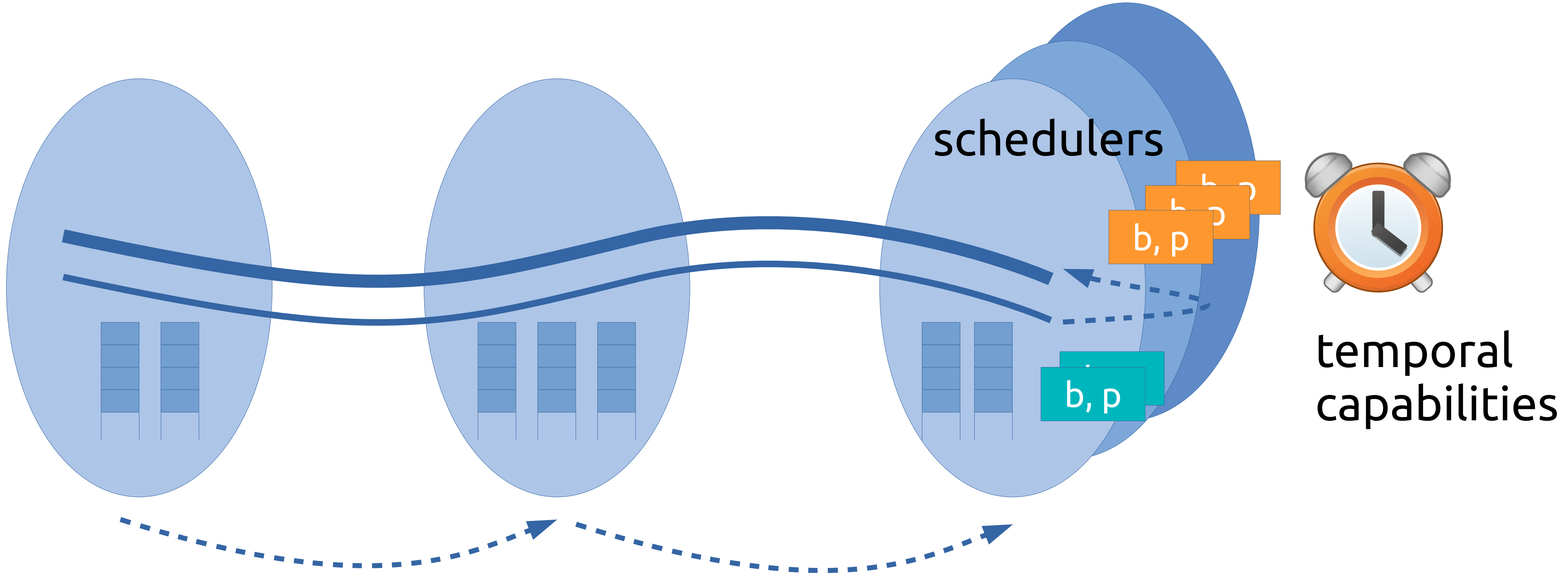

Parmer et al. *Predictable Interrupt Management and Scheduling in the Composite Component-based System*, RTSS '08
Gadepalli et al. *Temporal Capabilities: Access Control for Time*, RTSS '17

# Thread Migration



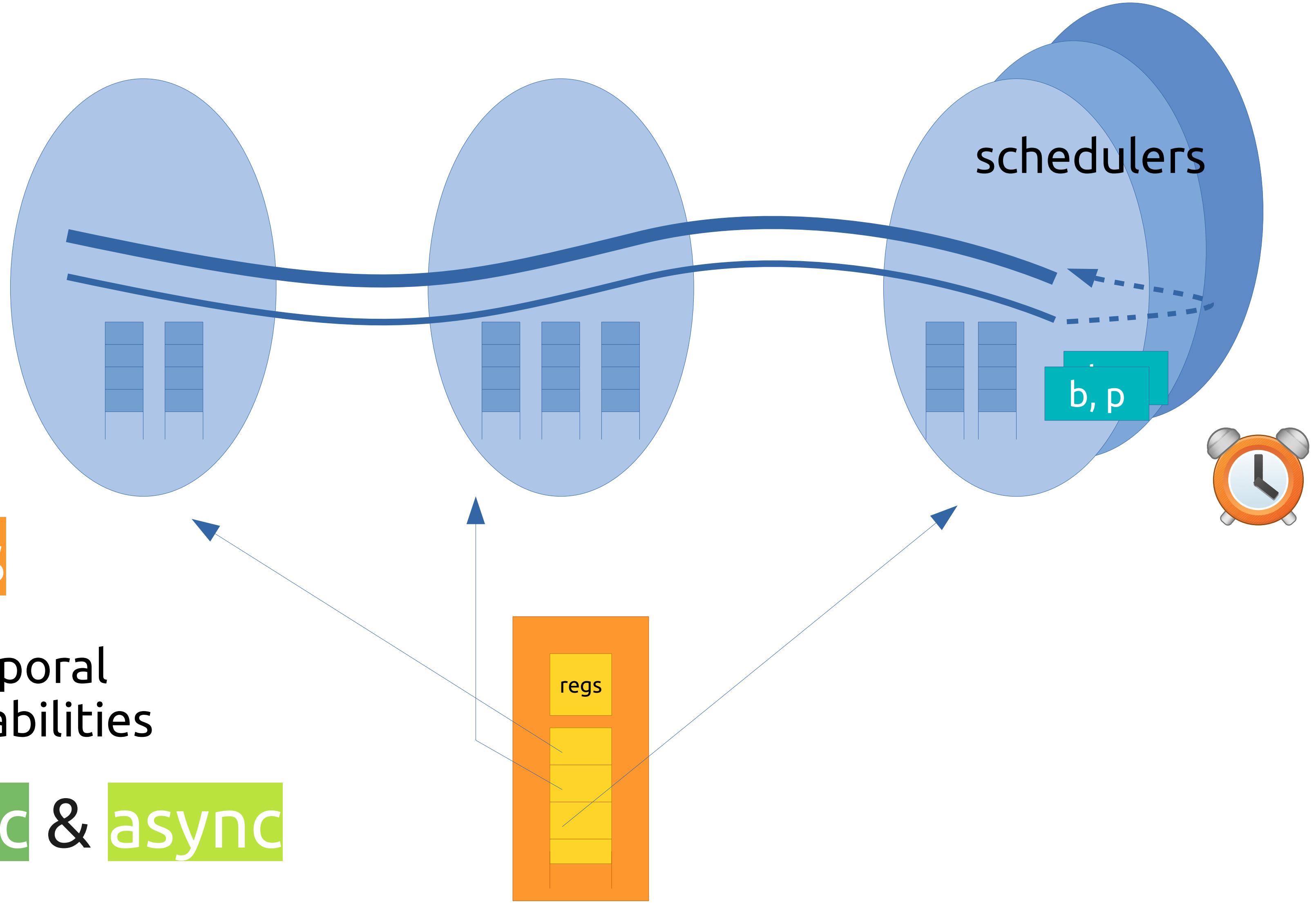

Parmer et al. *Predictable Interrupt Management and Scheduling in the Composite Component-based System*, RTSS '08
Gadepalli et al. *Temporal Capabilities: Access Control for Time*, RTSS '17

# Thread Migration




Gadepalli et al. *Temporal Capabilities: Access Control for Time*, RTSS '17
Parmer et al. *HiRes: a System for Predictable Hierarchical Resource Management*, RTAS '11

# Composite Kernel Objects

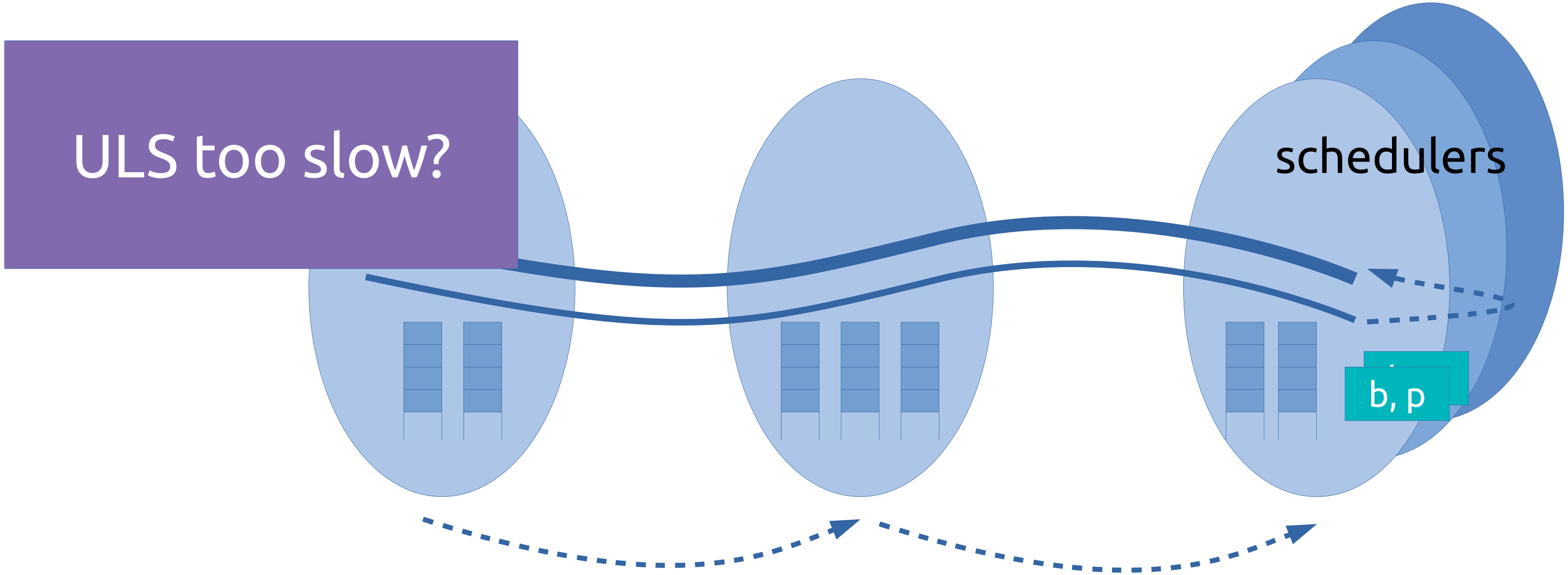



Threads

temporal capabilities

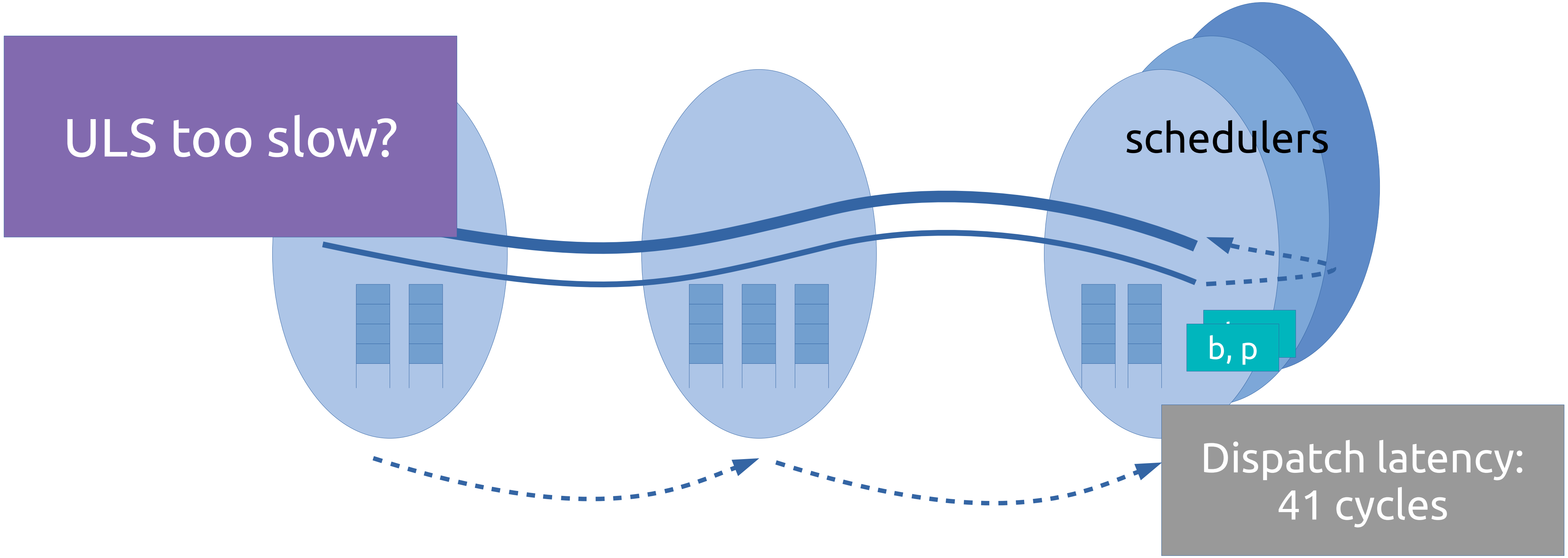
IPC: sync & async

# Thread Migration



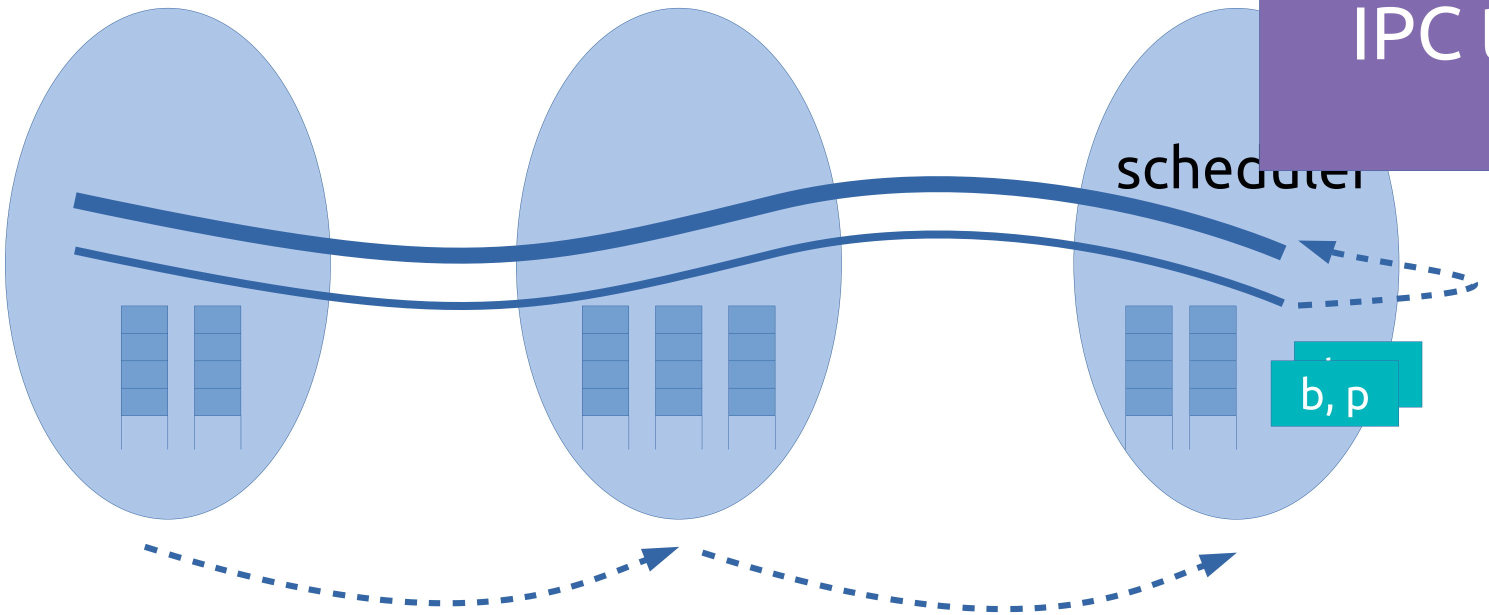

Gadepalli et al. *Slite: OS Support for Near Zero-Cost, Configurable Scheduling*, RTAS '20

# Thread Migration




Gadepalli et al. *Slite: OS Support for Near Zero-Cost, Configurable Scheduling*, RTAS '20

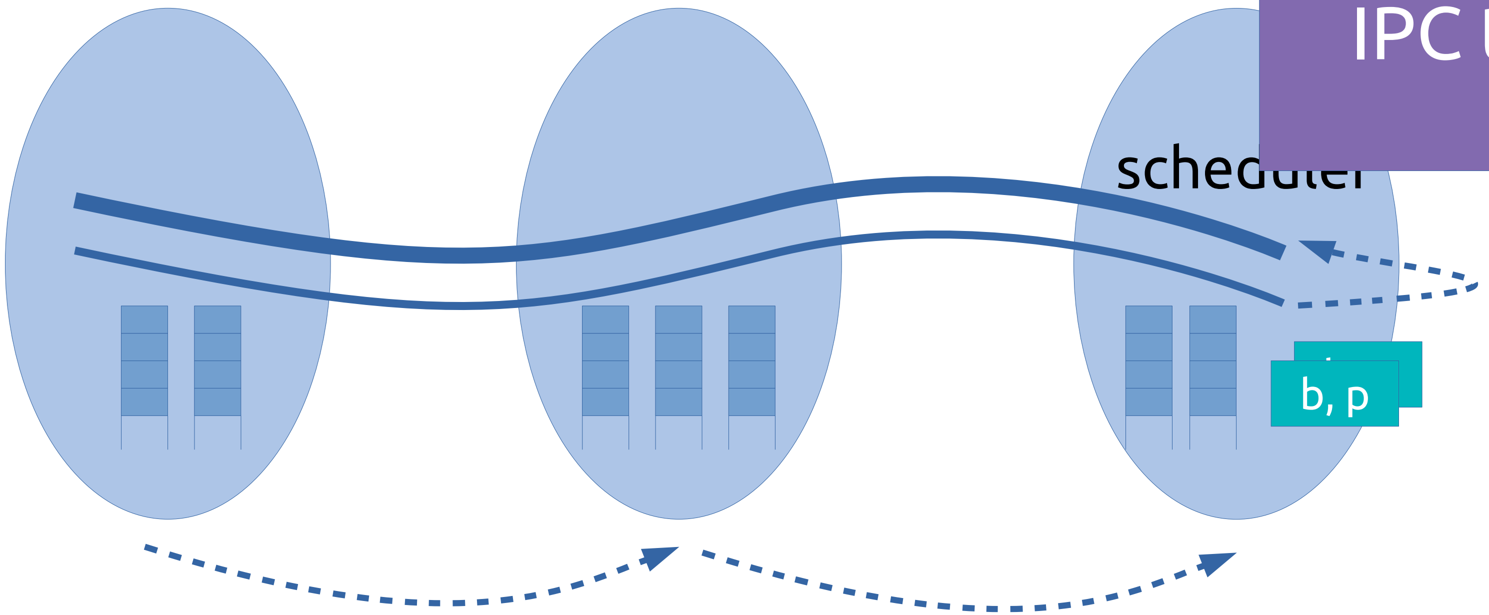
# Thread Migration




IPC too slow?

# Thread Migration

IPC too slow?



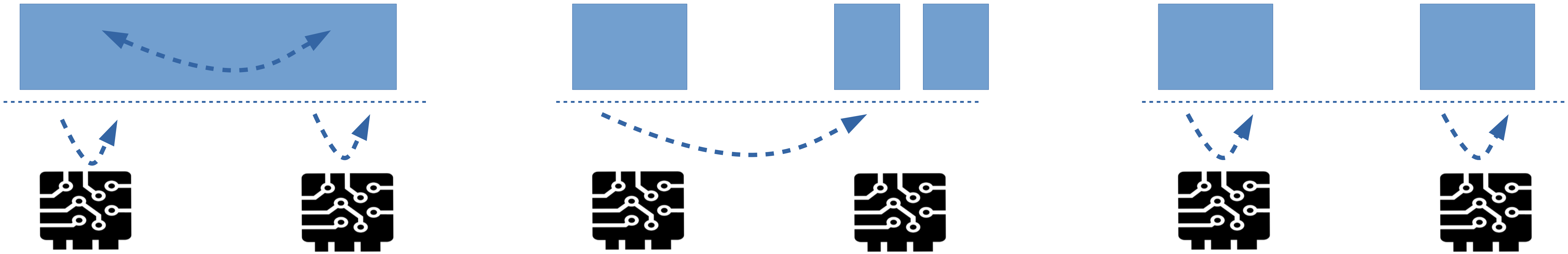

| Round-trip IPC | seL4 | composite |
|---|---|---|
| X86-32 (3.2GHz) | **934** | **741** |
| Cortex a9 (667 GHz, zynq) | **630** | **543** |

Gadepalli et al. *Chaos: a System for Criticality-Aware, Multi-core Coordination*, RTAS '19
Samuel Jero et al. *Practical Principle of Least Privilege for Secure Embedded Systems*, RTAS '21

# *Kernel Synchronization*

Can kernel APIs/implementation

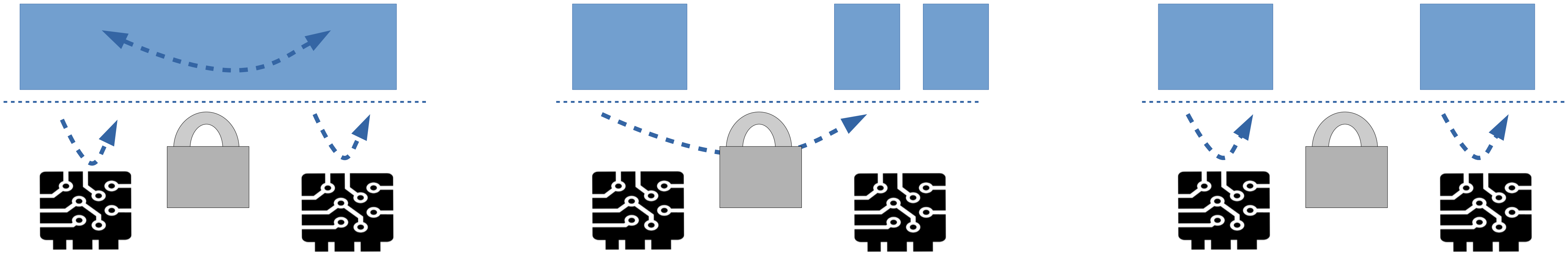
- Limit the scalability of components?
- Cause interference between components?



# *Kernel Synchronization*

Can kernel APIs/implementation

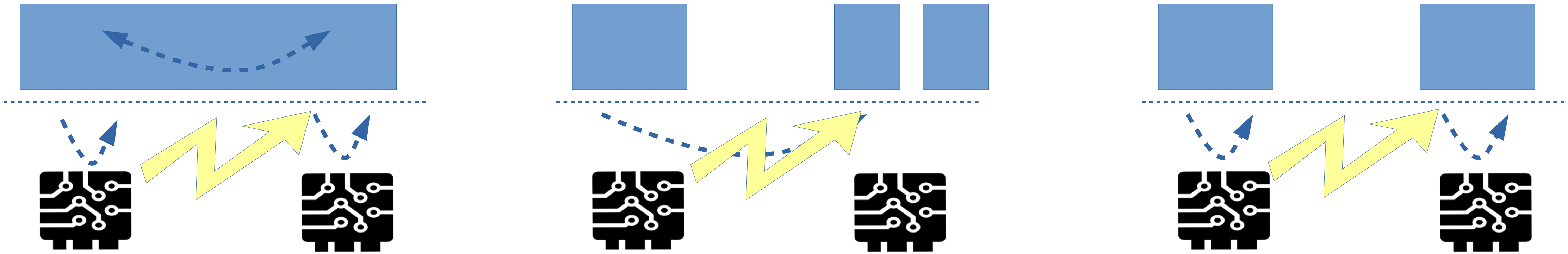
- Limit the scalability of components?
- Cause interference between components?



# *Kernel Synchronization*

Can kernel APIs/implementation

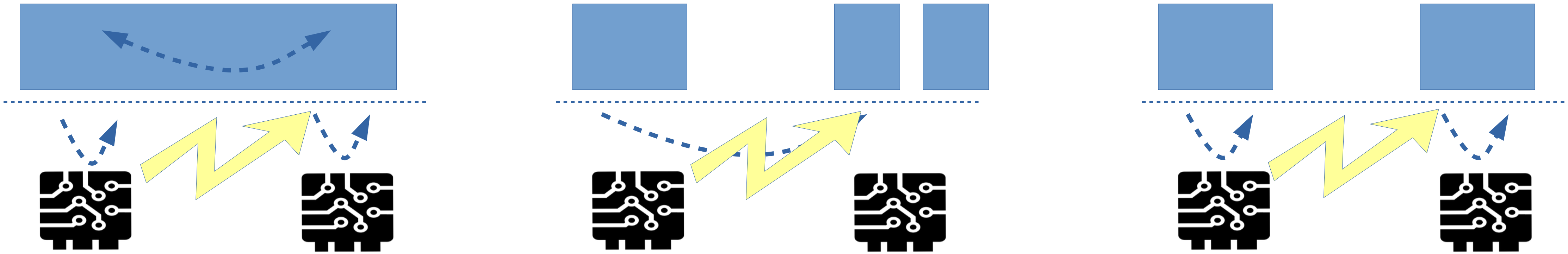
- Limit the scalability of components?
- Cause interference between components?



# *Kernel Synchronization*

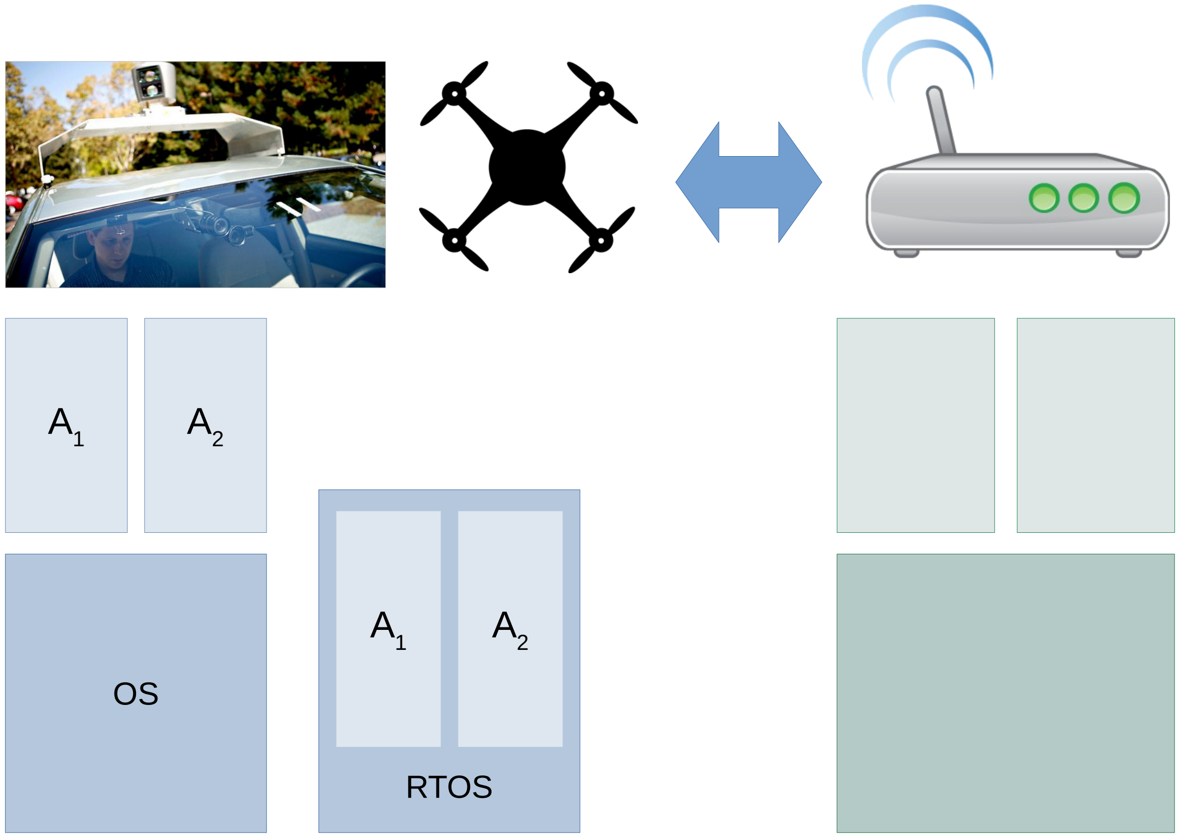
Component policies for predictability/scalability?

– Kernel must be **wait-free** w/ component-controlled cacheline contention
– Component-**controlled IPI facilities**

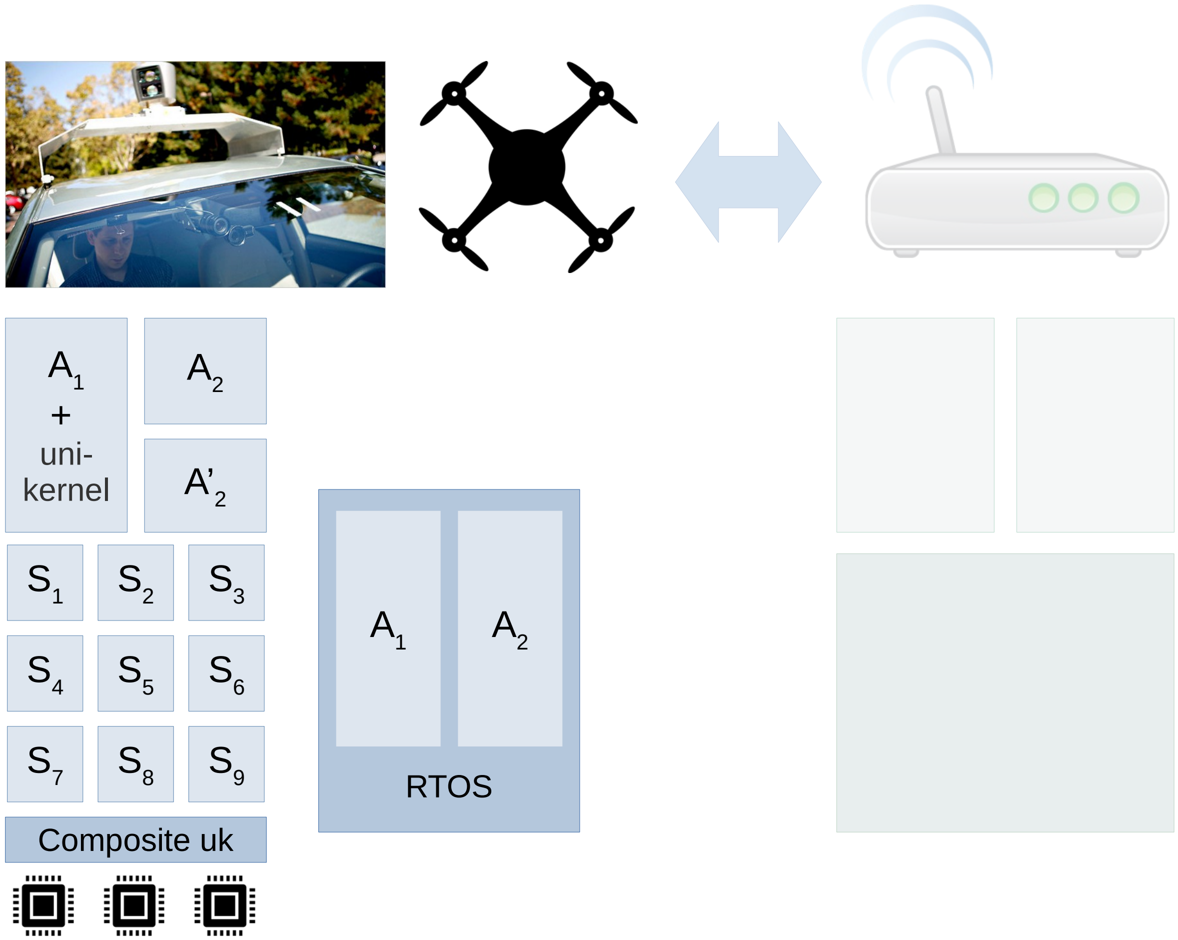


Gadepalli et al. *Chaos: a System for Criticality-Aware, Multi-core Coordination*, RTAS '19
Wang et al. *Speck: A Kernel for Scalable Predictability*, RTAS '15

# Research Approaches



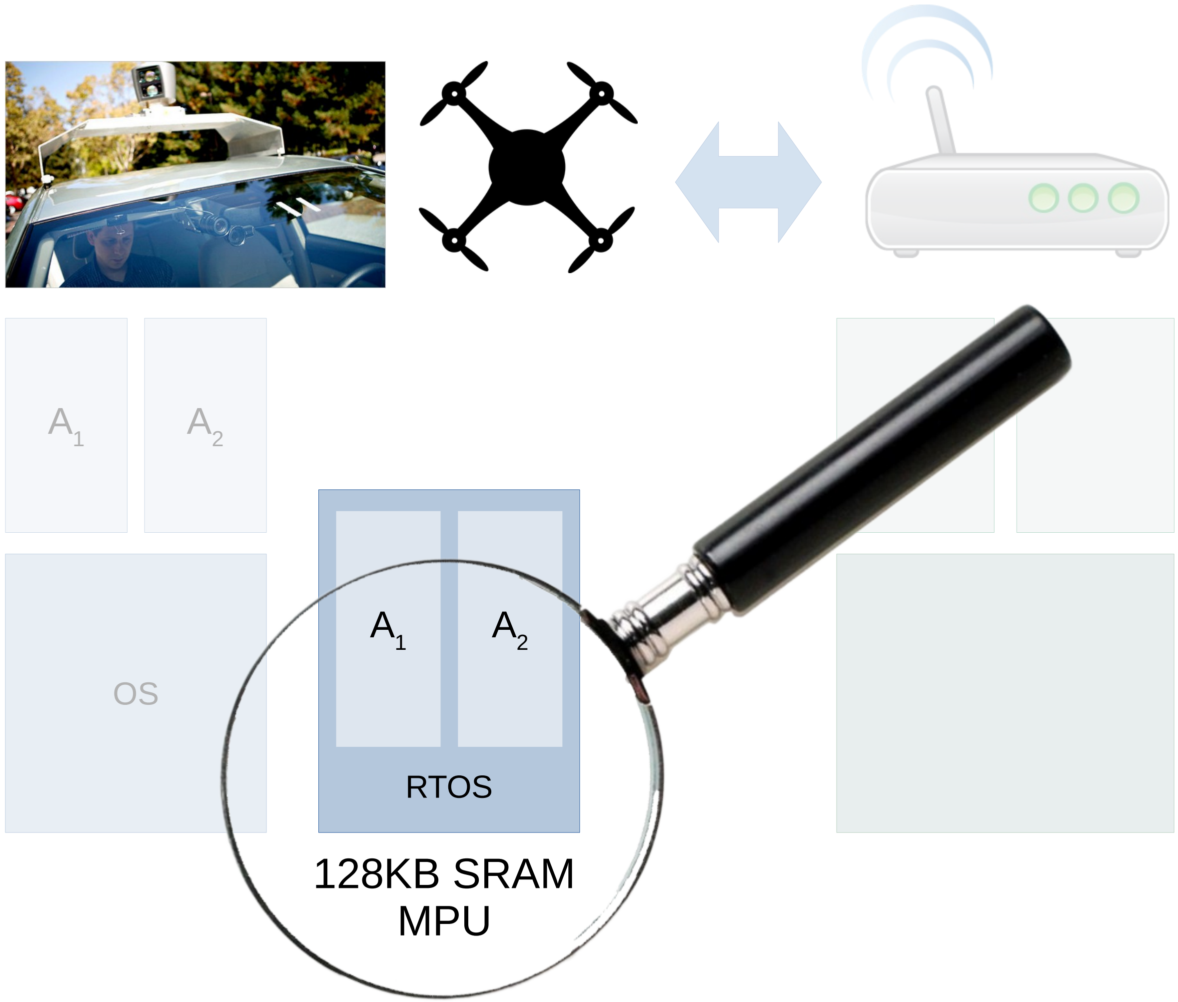
# Research Approaches




***Composite* Infrastructure**

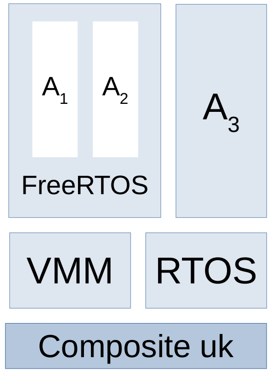
- PoLP-focused RTOS
- NetBSD Rumpkernels
- Xen-like driver domains
- NASA CFS
- Mixed-criticality system orchestration

# Research Approaches



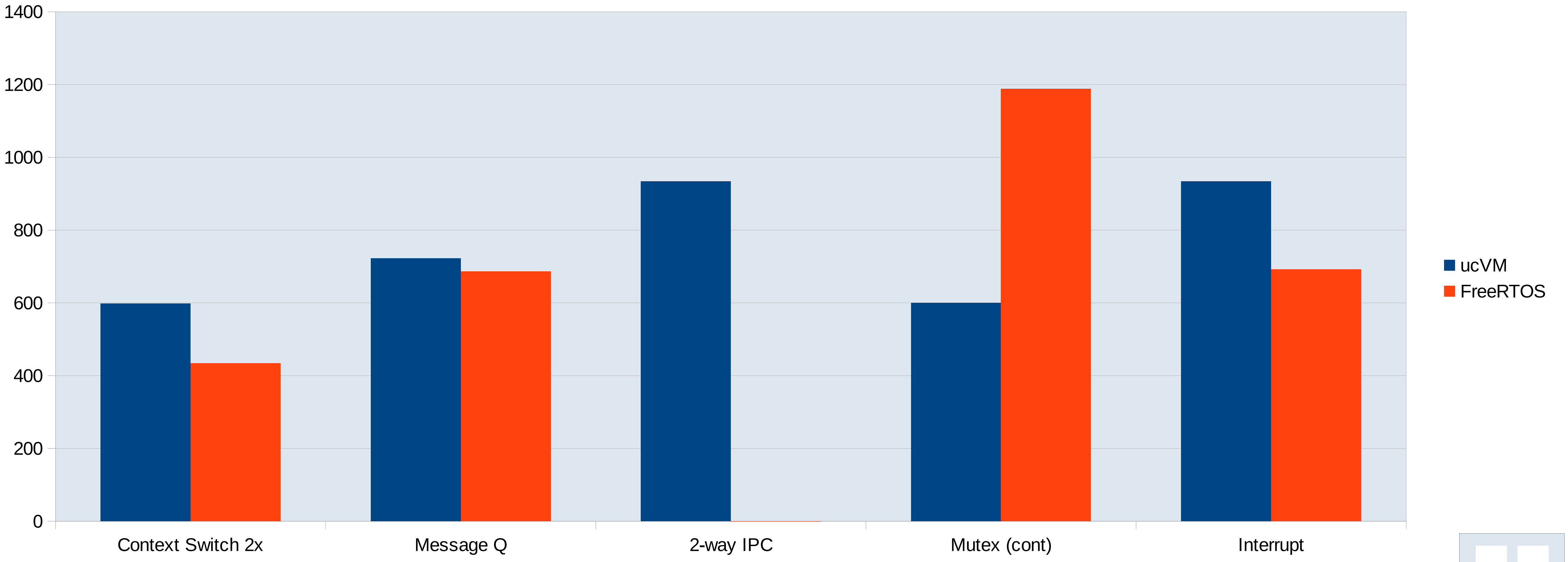
# ucVM Challenges

- MPU-based isolation vs. page-table software abstractions
  - path-compressed radix tries flattened into MPU regions
- Limited # of MPU protected regions
  - Solve memory layout: contiguous memory fits into # regions
  - Treated as dynamic software protection cache
- Efficient coordination and event processing
  - Kernel-bypass + micro-optimization
- ~8 VMs in 128KiB SRAM



# ucVM vs. FreeRTOS

Arm Cortex-M7, 216 Mhz

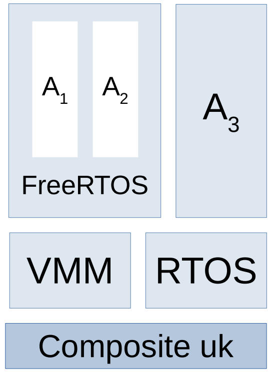



## Also: Secure Bare-metal interrupts

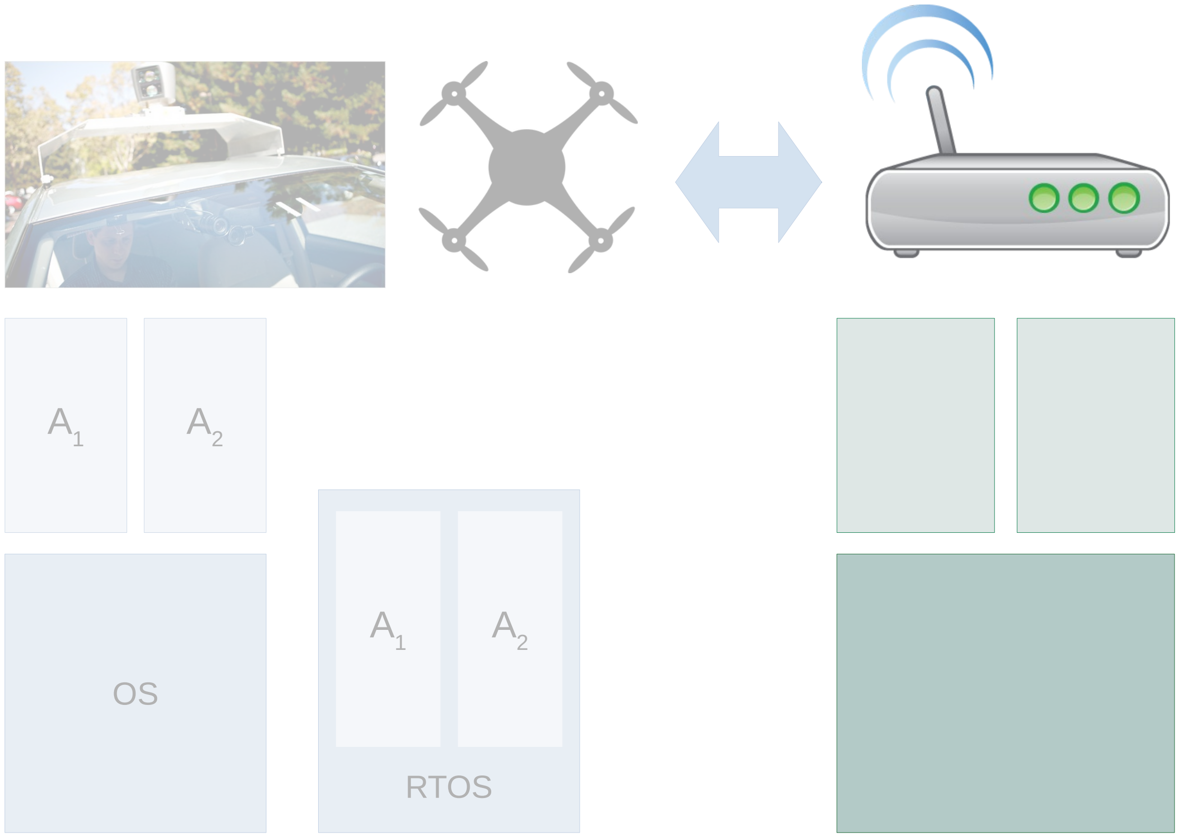
- Use Trustzone-M as secure kernel bypass for interrupts


Pan et al. *Predictable Virtualization on Memory Protection Unit-based Microcontrollers*, RTAS '18

Pan et al. *MxU: Towards Predictable, Flexible, and Efficient Memory Access Control for the Secure IoT*, EMSOFT '19

# Research Approaches




Density
Multi-tenancy
Latency

# The Edge: Options

| | Isolation | Scalability | Startup Time | High-performance networking |
|---|---|---|---|---|
| Processes | ✗ | ✓ | ! | ! |
| Containers | ! | ! | ✗ | ! |
| VMs | ✓ | ✗ | ✗ | ✗ |

# The Edge: Options

| | Isolation | Scalability | Startup Time | High-performance networking |
|---|---|---|---|---|
| Processes | ✗ | ✓ | ! | ! |
| Containers | ! | ! | ✗ | ! |
| VMs | ✓ | ✗ | ✗ | ✗ |
| EdgeOS: Feather-weight Processes | ✓ | ✓ | ✓ | ✓ |

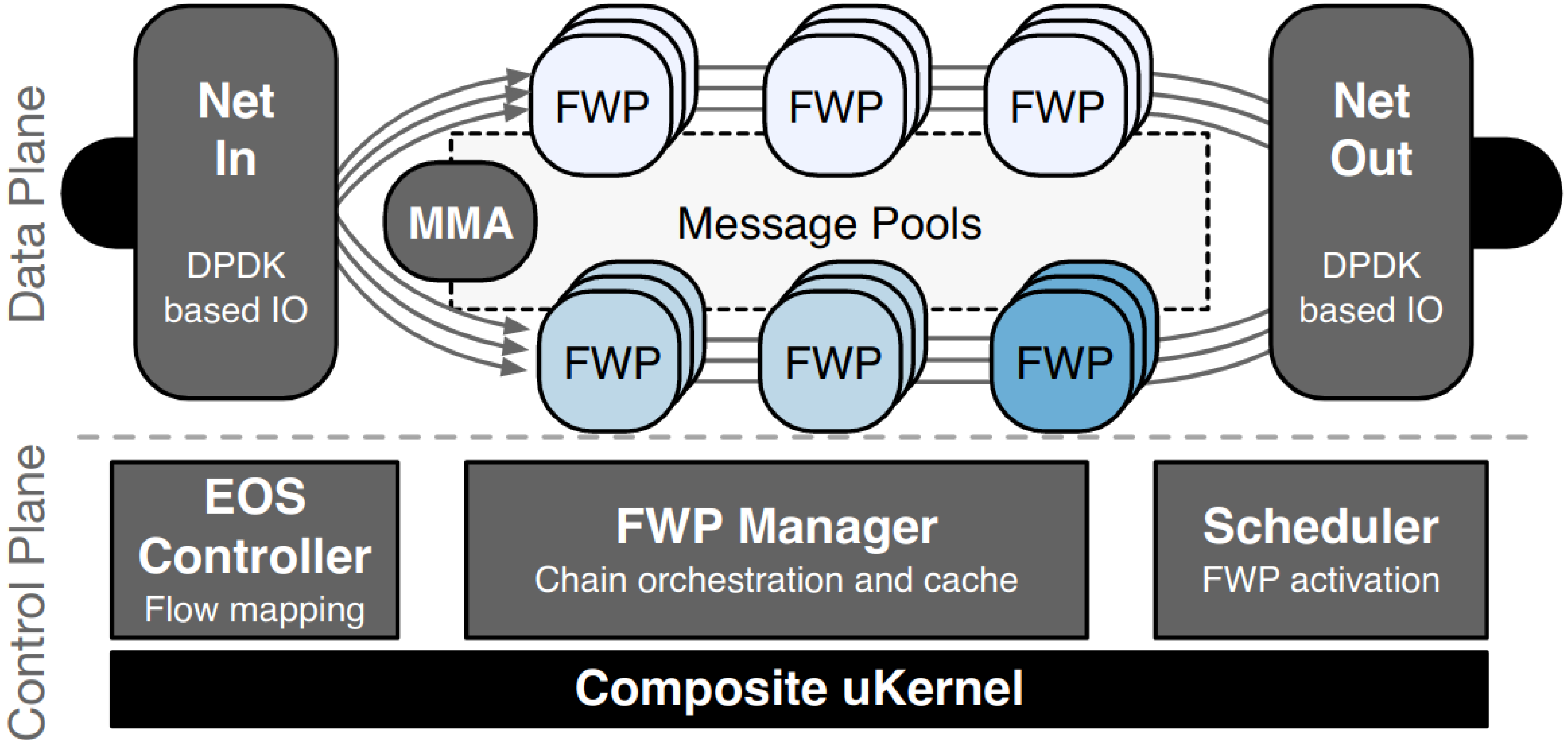
# The Edge: Options

| | Isolation | Scalability | Startup Time | High-performance networking |
|---|---|---|---|---|
| EdgeOS: Feather-weight Processes | ✓ | ✓ | ✓ | ✓ |

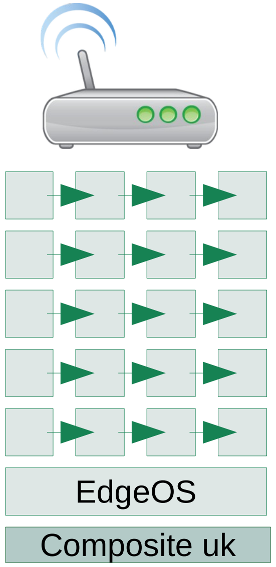
***EdgeOS*: Isolation, Predictability, Performance, and Scale**

- High speed data movement (10Gbps+) without sacrificing isolation
- Startup > **100X** faster than fork+exec
- Scales to 1000s of services per host = 1 service per user!

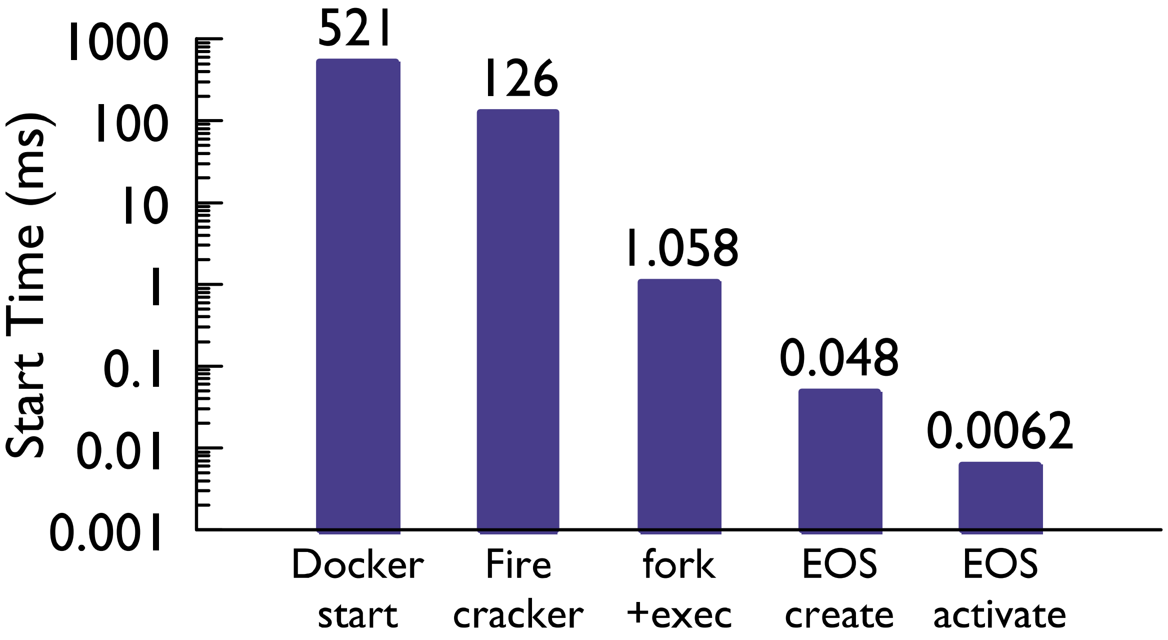
# EdgeOS: Strong Isolation & Performance



# Per-client Service Instantiation

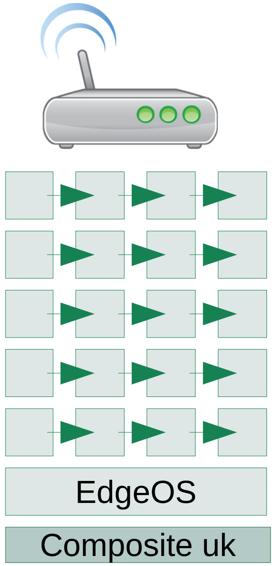



- Docker: the execution time of “docker start”
- Firecracker: the start time of the recommended “hello” image
- Linux: fork() + exec()

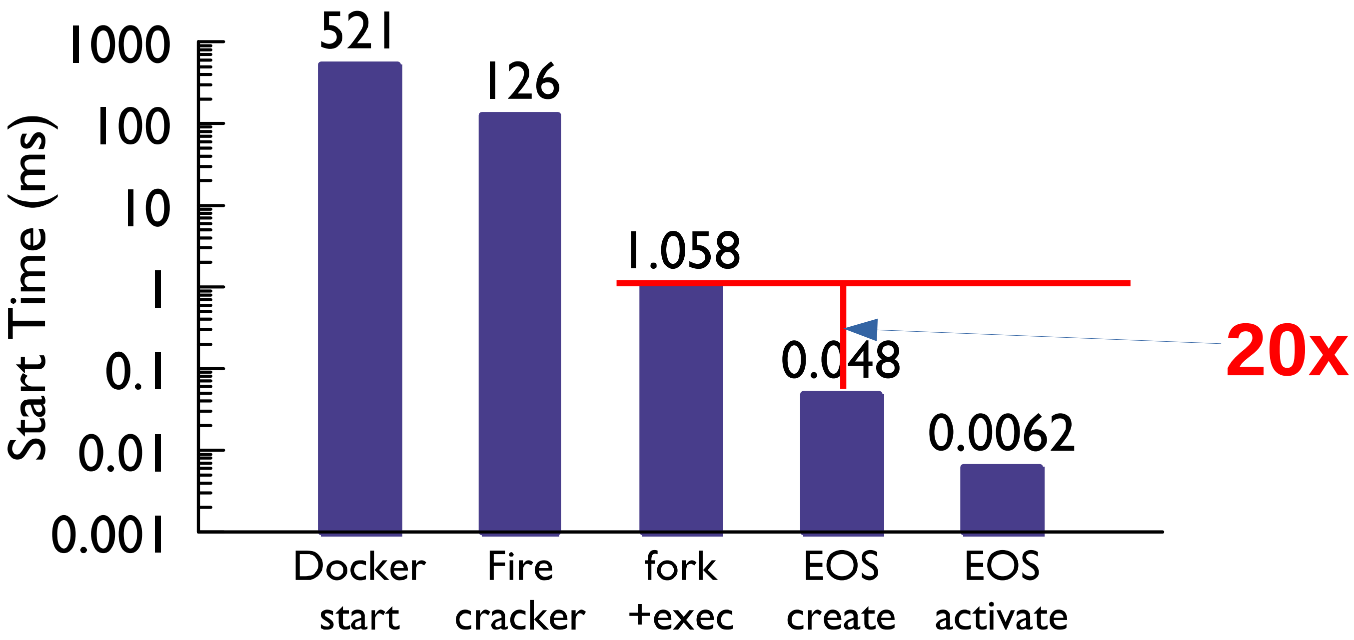



Ren et al. *Fine-Grained Isolation for Scalable, Dynamic, Multi-tenant Edge Clouds*, USENIX ATC ‘20

# Per-client Service Instantiation

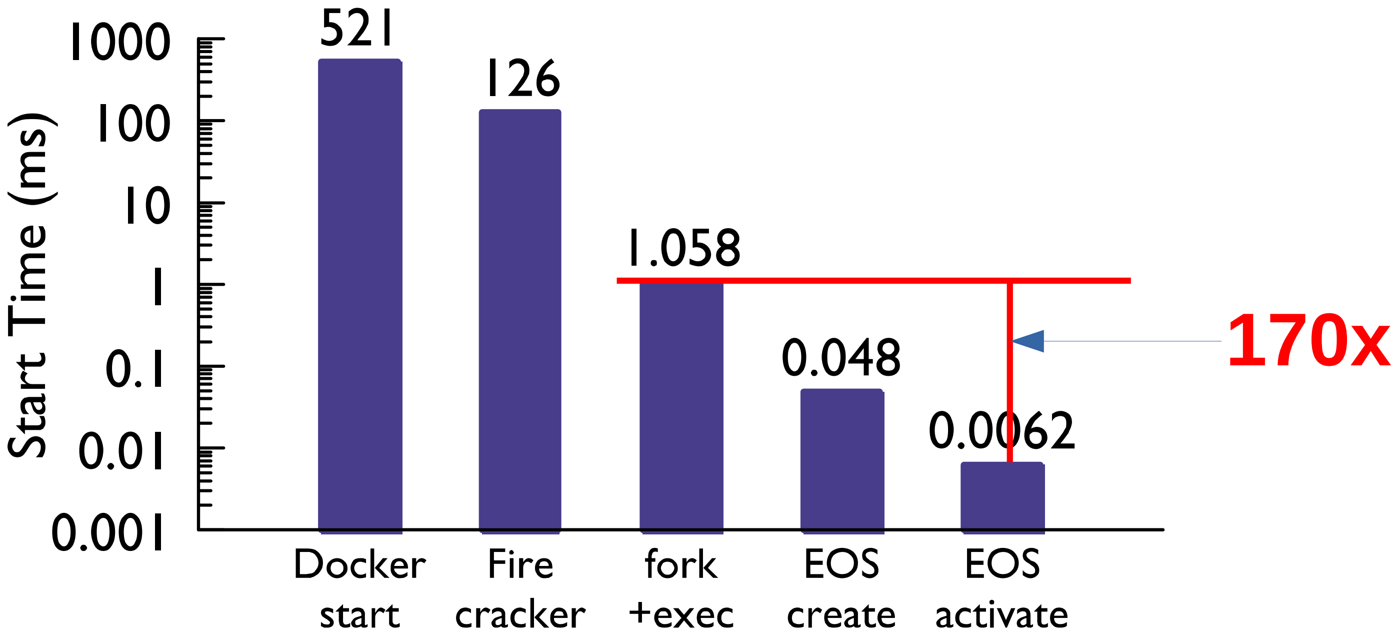


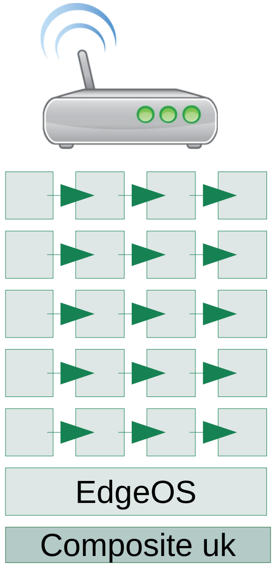

- Docker: the execution time of “docker start”
- Firecracker: the start time of the recommended “hello” image
- Linux: fork() + exec()



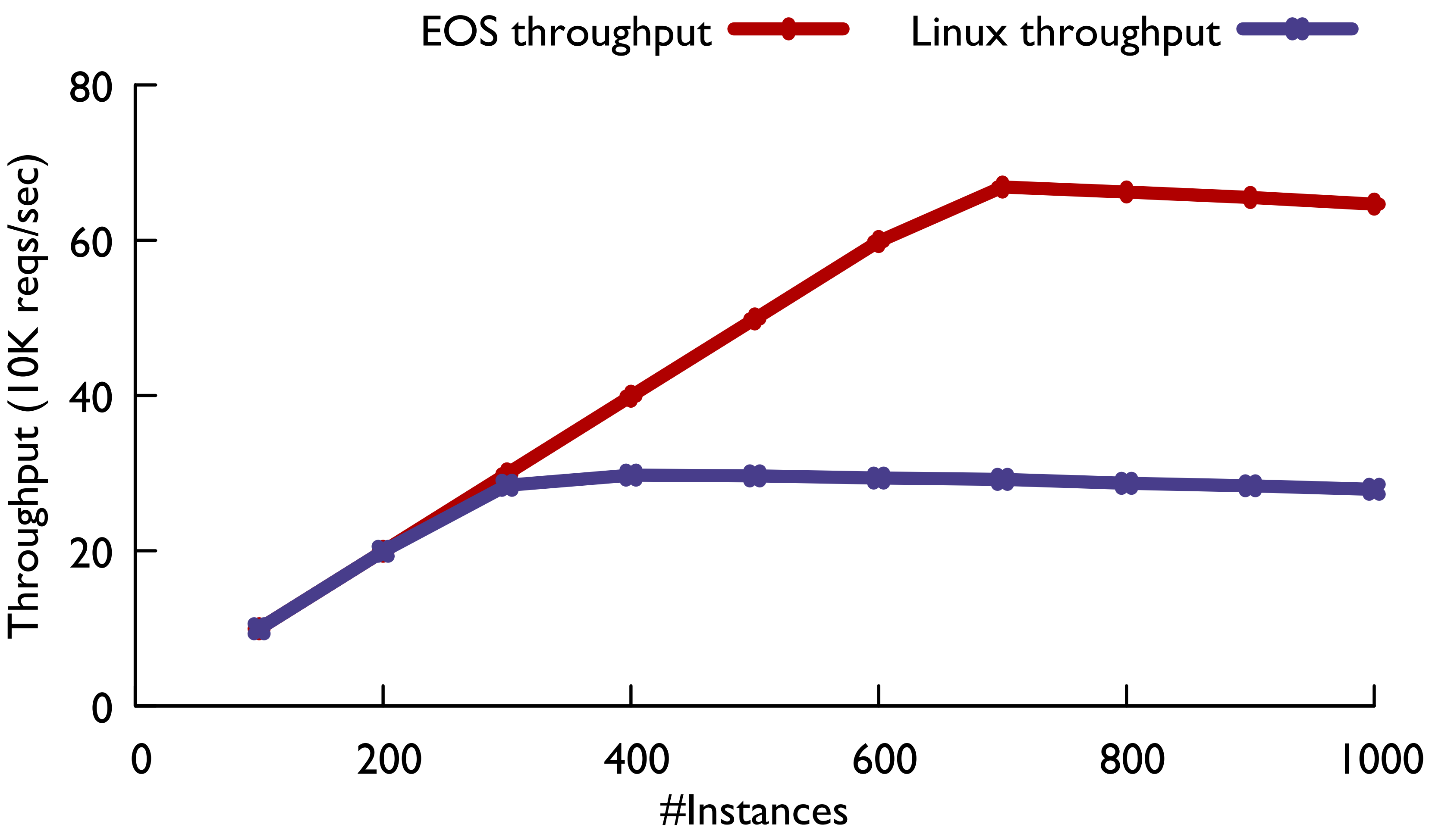
# Per-client Service Instantiation

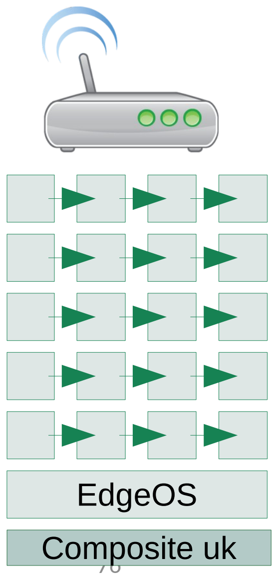
- Docker: the execution time of “docker start”
- Firecracker: the start time of the recommended “hello” image
- Linux: fork() + exec()



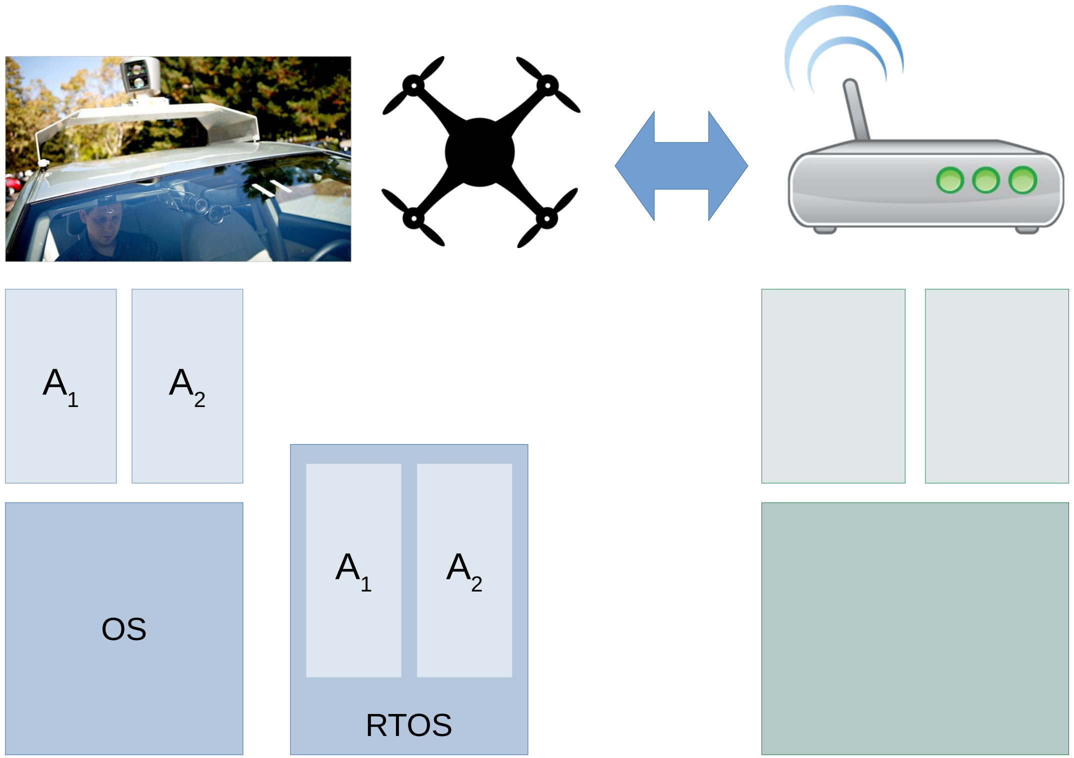



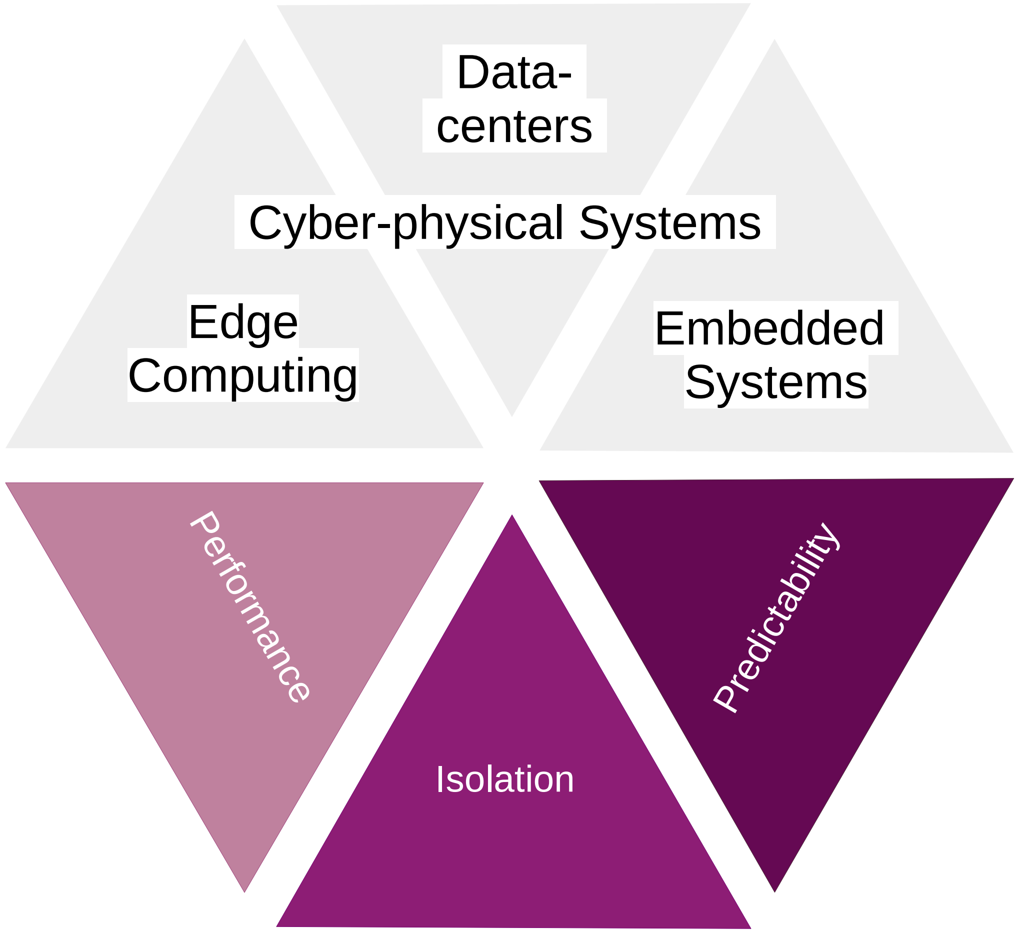
# EdgeOS: scalability for TLS proxies



# Conclusions



# ? || /* */

https://github.com/gwsystems/composite
https://composite.seas.gwu.edu

GW