Thoughts on Non-functional Challenges in seL4

Gabriel Parmer
The George Washington University

Non-Functional Goals

- Timing / predictability
- Rate Allocations (CPU, devices)
- Scalability
- Performance
- Fault tolerance / dependability

Non-functional *Composability*

If we create *functional dependencies* between *multiple protection domains*, are *non-functional constraints* impacted?

- End-to-end predictability?
  - Scheduling policy, bounded execution
- End-to-end scalability?
  - Kernel impact on all thread's operations

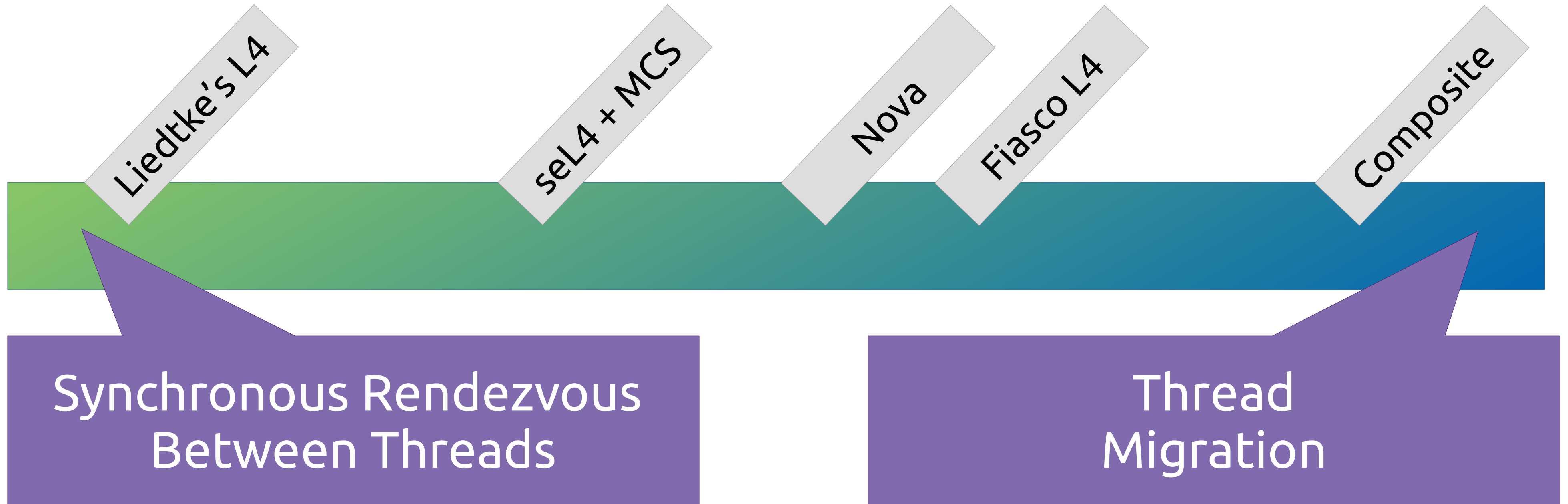
Non-functional *Composability*

If we create *functional dependencies* between *multiple protection domains*, are *non-functional constraints* impacted?

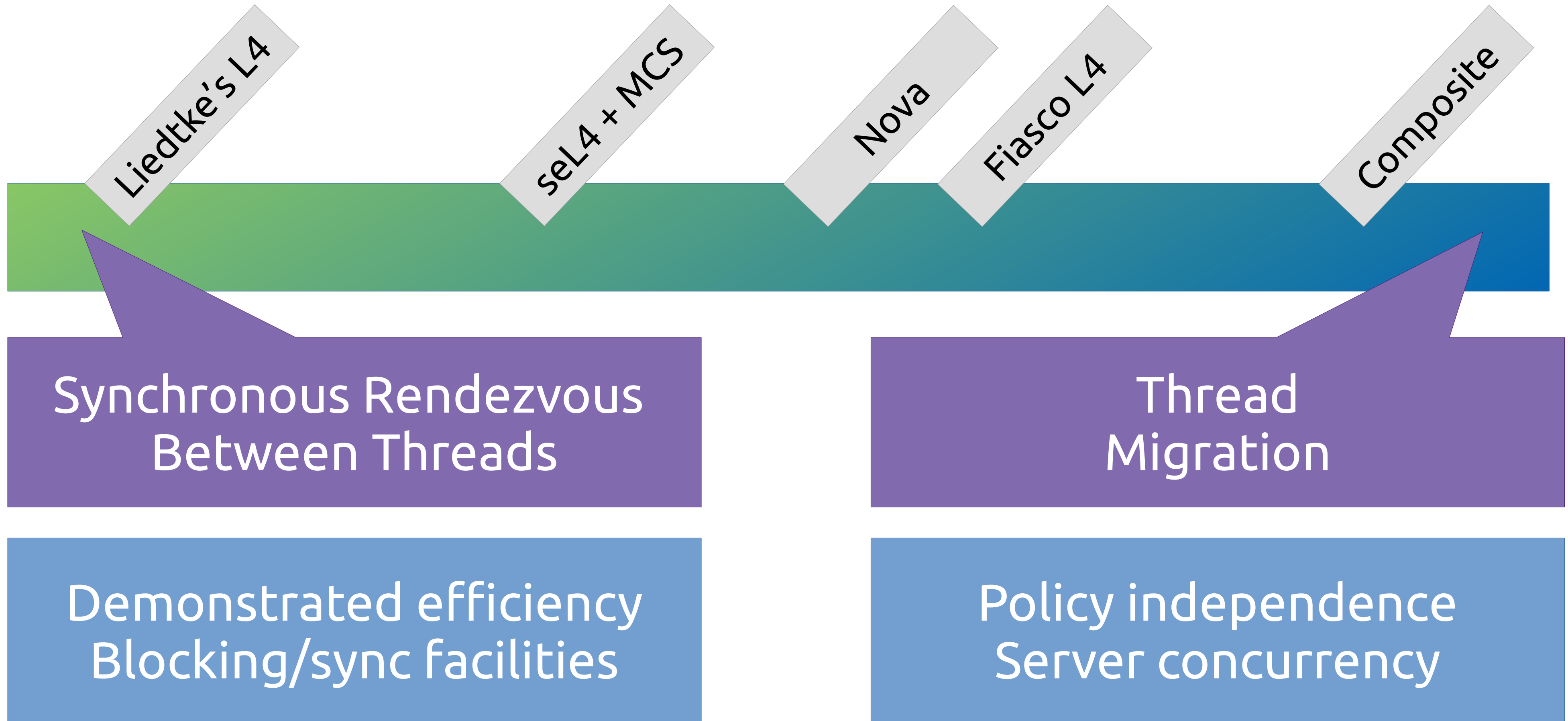
- End-to-end predictability?
  - Scheduling policy, bounded execution
- End-to-end scalability?
  - Kernel impact on all thread's operations

Spectrum of Sync. IPC Mechanisms

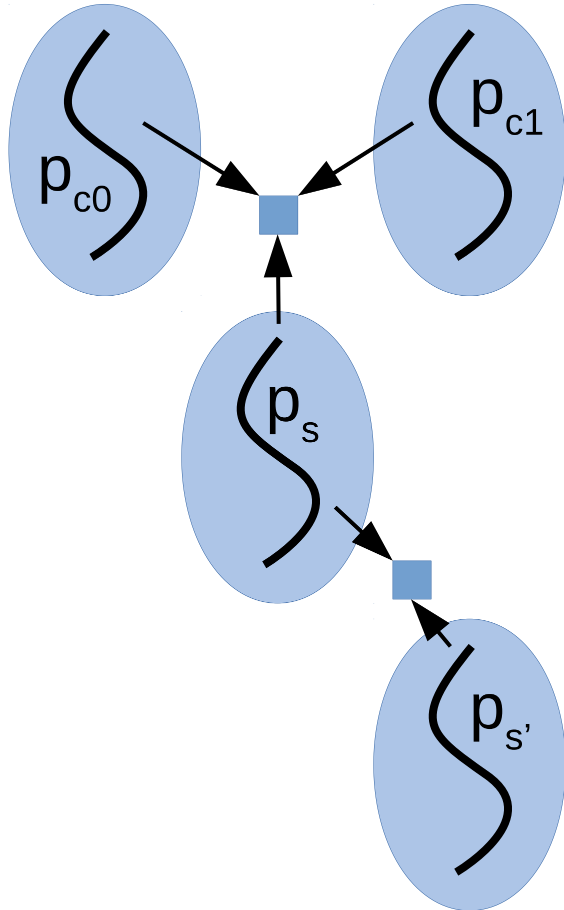
Spectrum of IPC Mechanisms



Spectrum of IPC Mechanisms

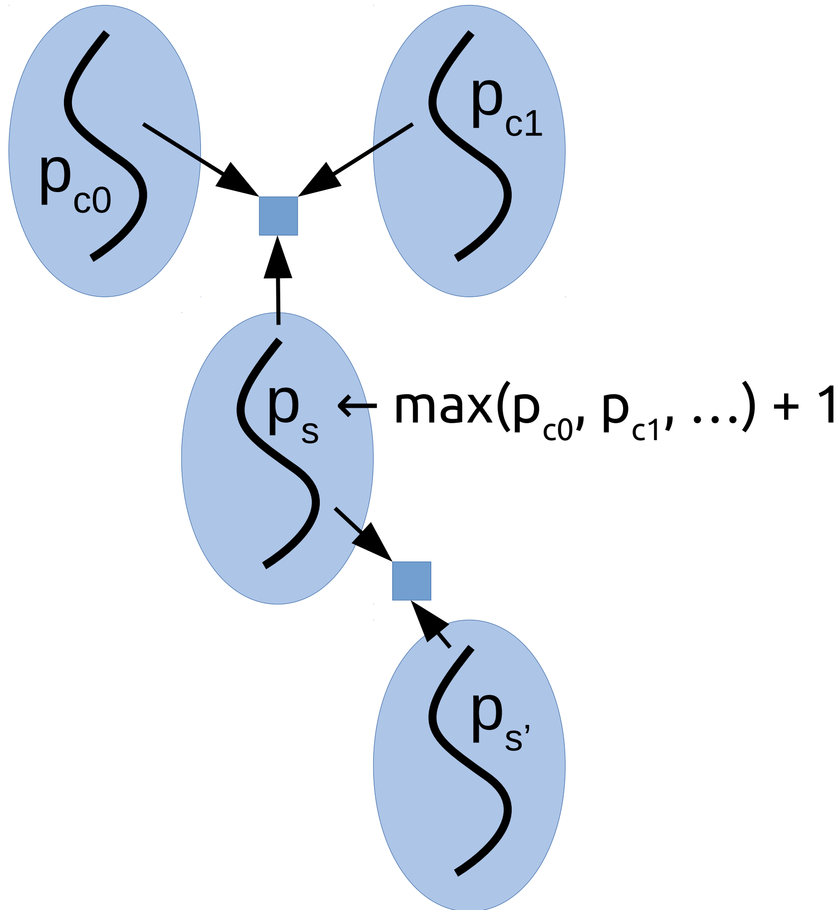


Sync IPC w/ Rendezvous



- Priority/budget assignment?

Sync IPC w/ Rendezvous

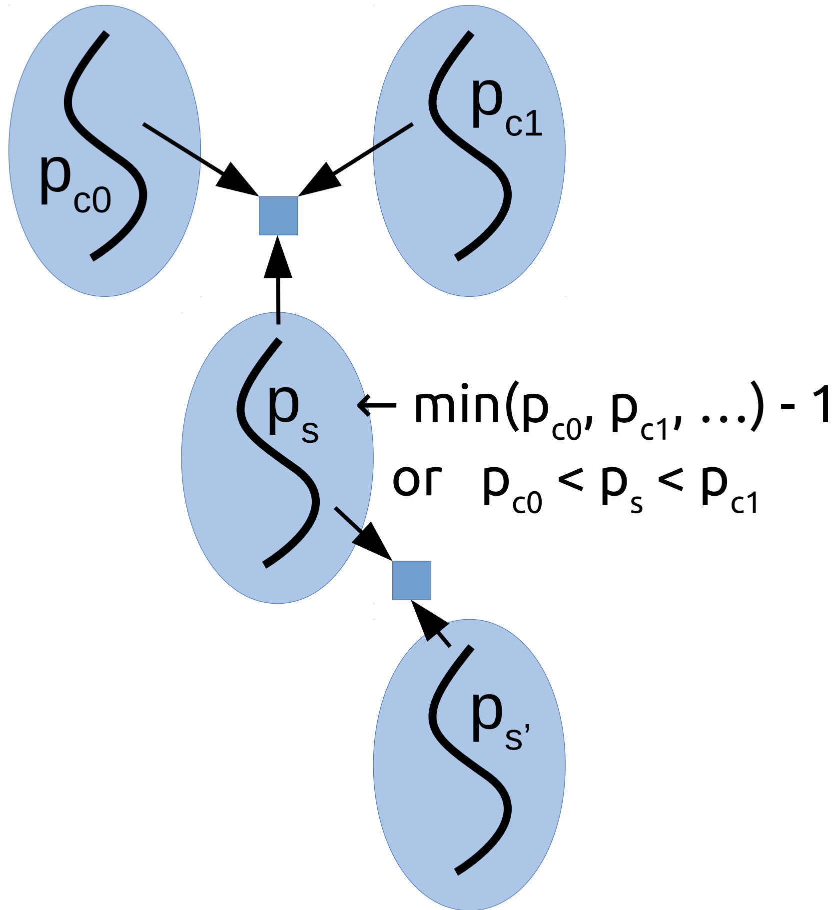


- Deterministic
  - Prioritization
  - Accounting

Priority Ceiling assignment

- Pessimistic *interference* (PCP) → service execution at HP
- DoS on limited server “budget”

Sync IPC w/ Rendezvous

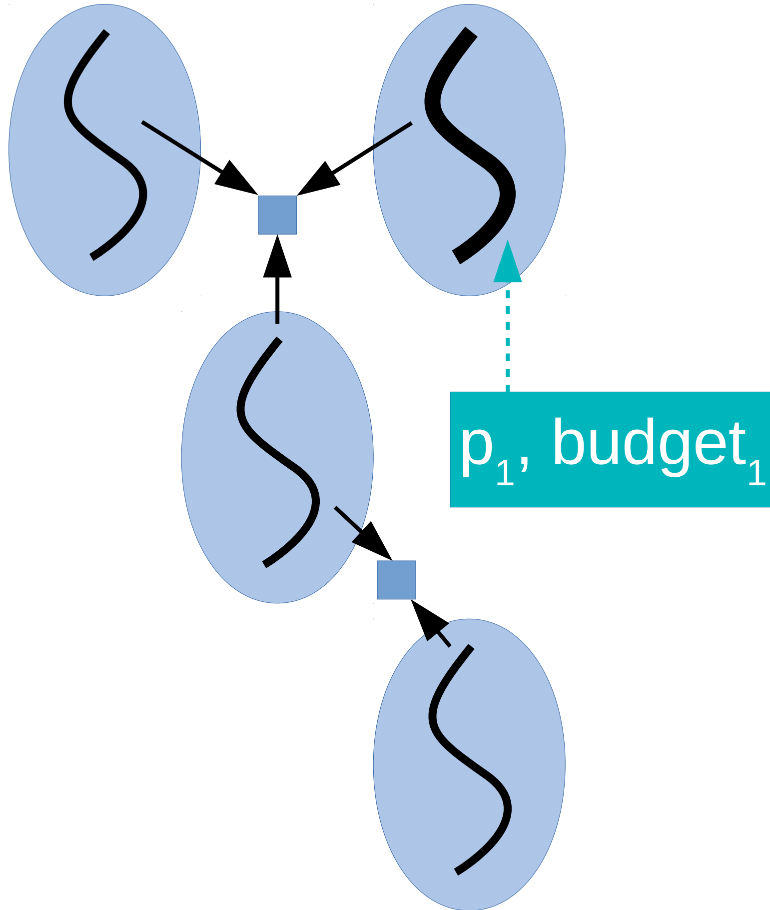


- Deterministic
  - Prioritization
  - Accounting

Low priority assignment

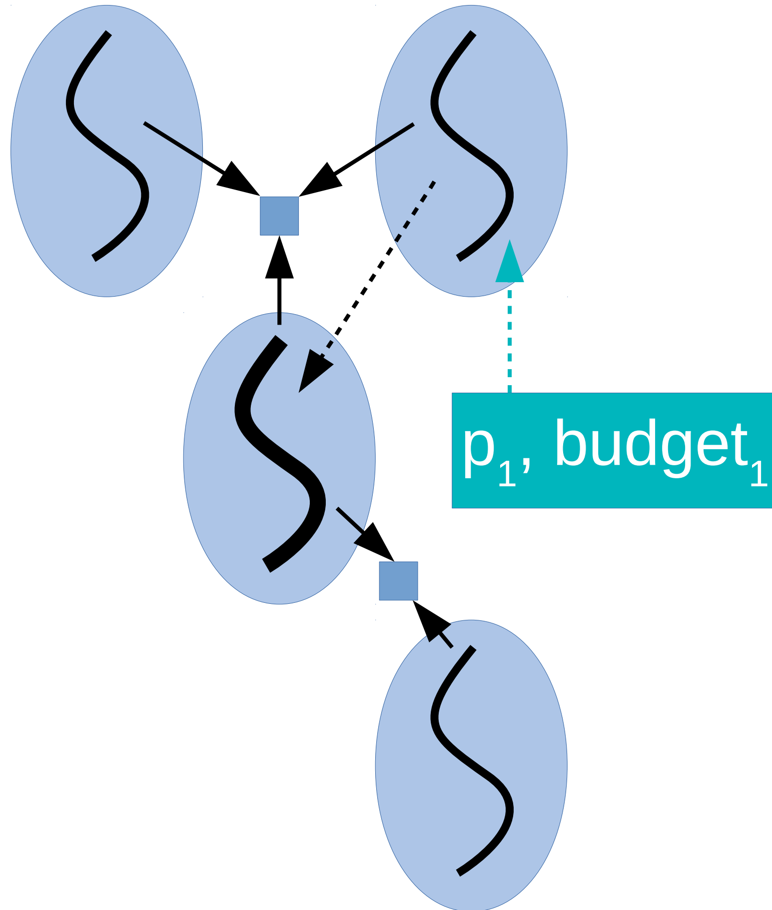
- Unbounded priority inversion

L4: Lazy Scheduling + TS Donation



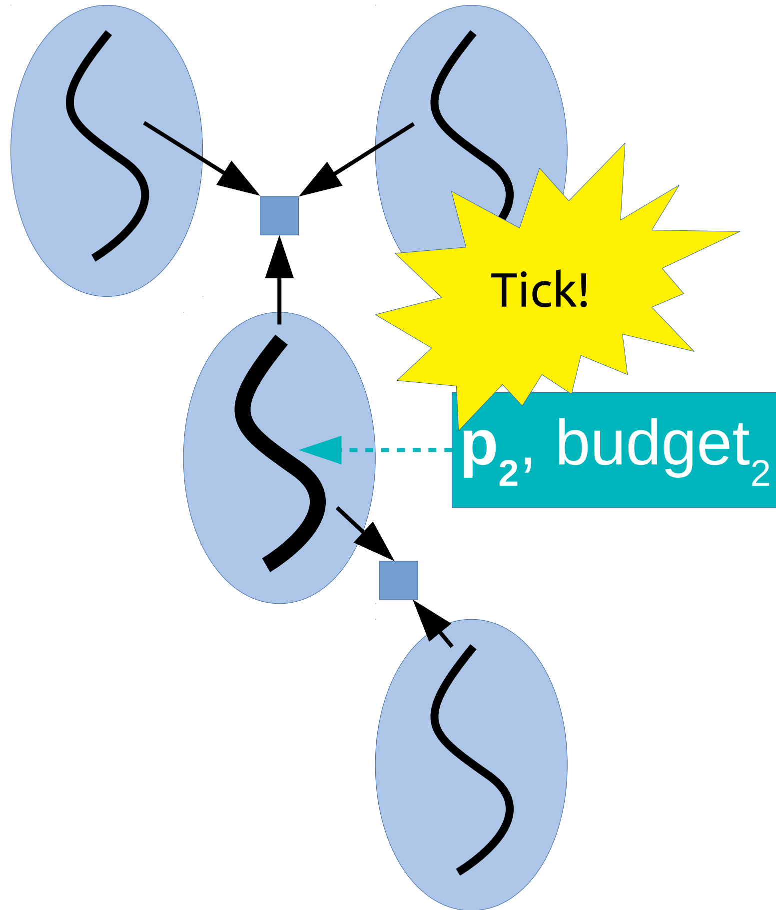
- Accounting/prio dependent on timer → non-deterministic

L4: Lazy Scheduling + TS Donation



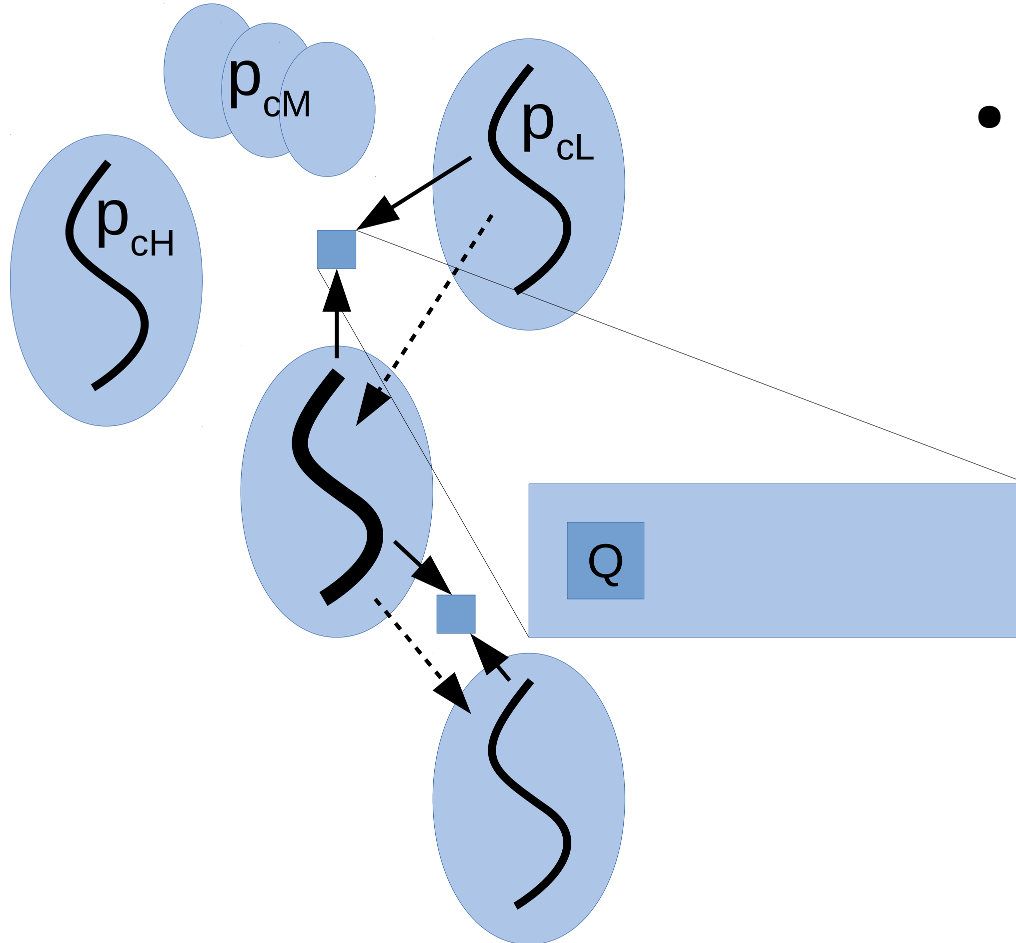
- Accounting/prio dependent on timer → non-deterministic

L4: Lazy Scheduling + TS Donation



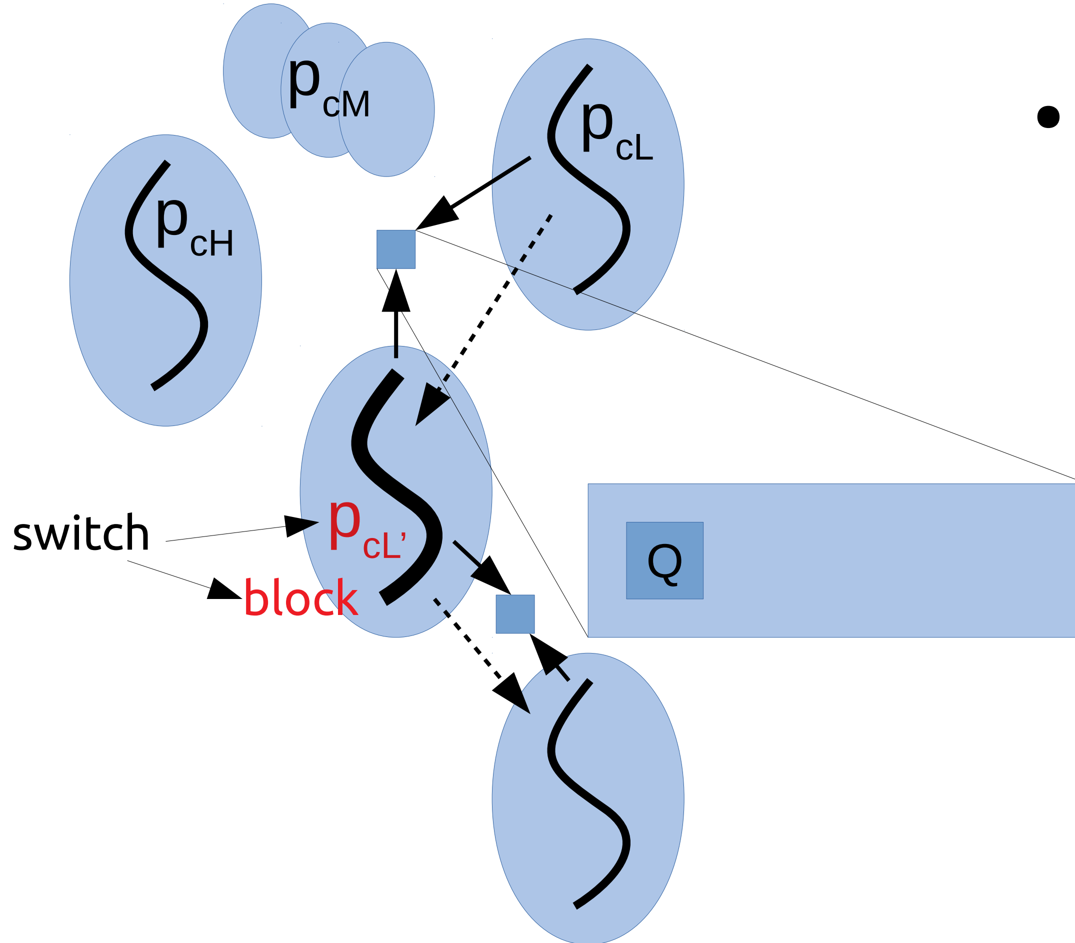
- Accounting/prio dependent on timer → non-deterministic

L4: Priority Inversion



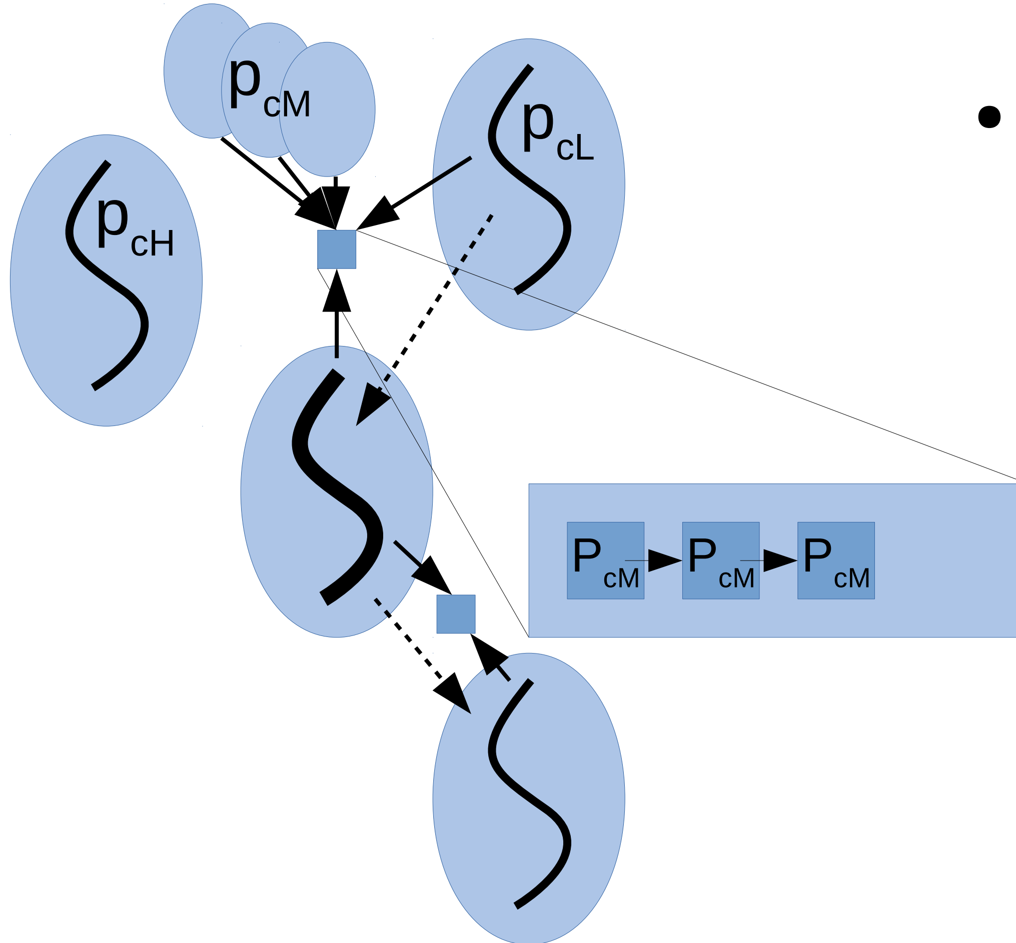
- Priority inversion in shared end-points
  - O(1) → inversion

L4: Priority Inversion



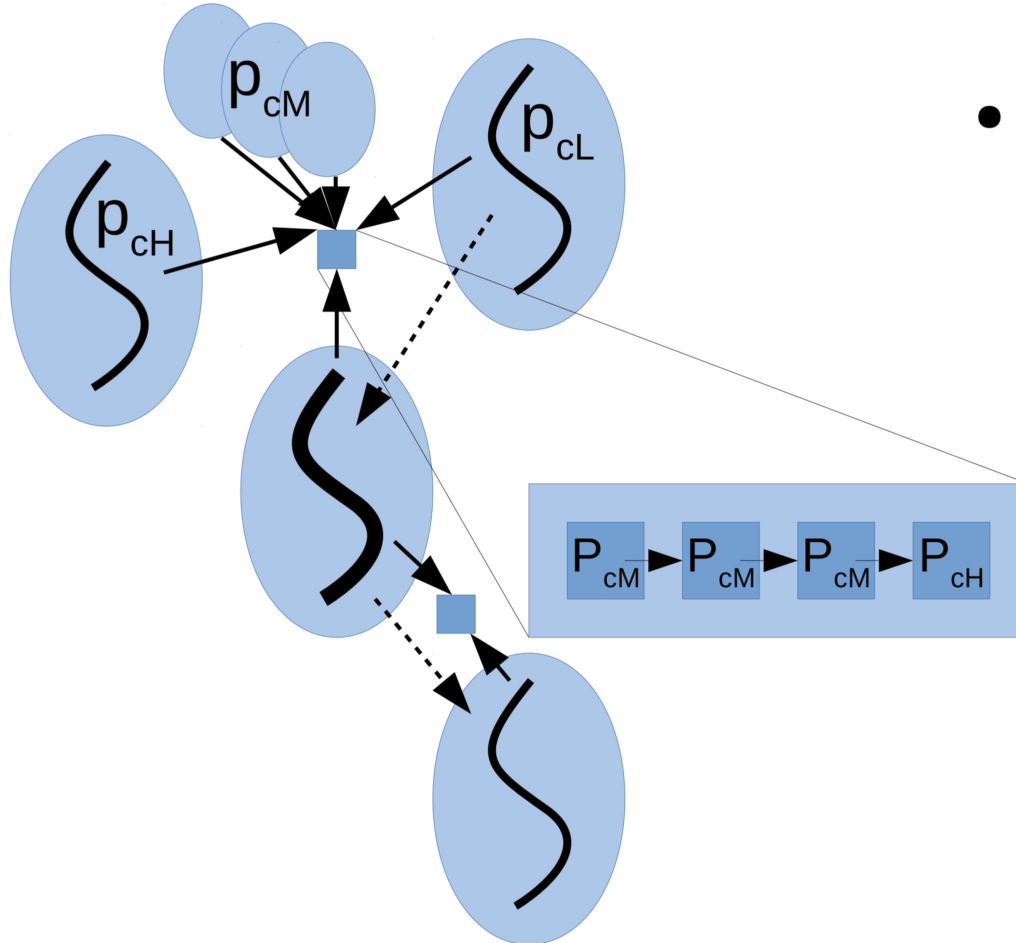
- Priority inversion in shared end-points
  - O(1) → inversion

L4: Priority Inversion



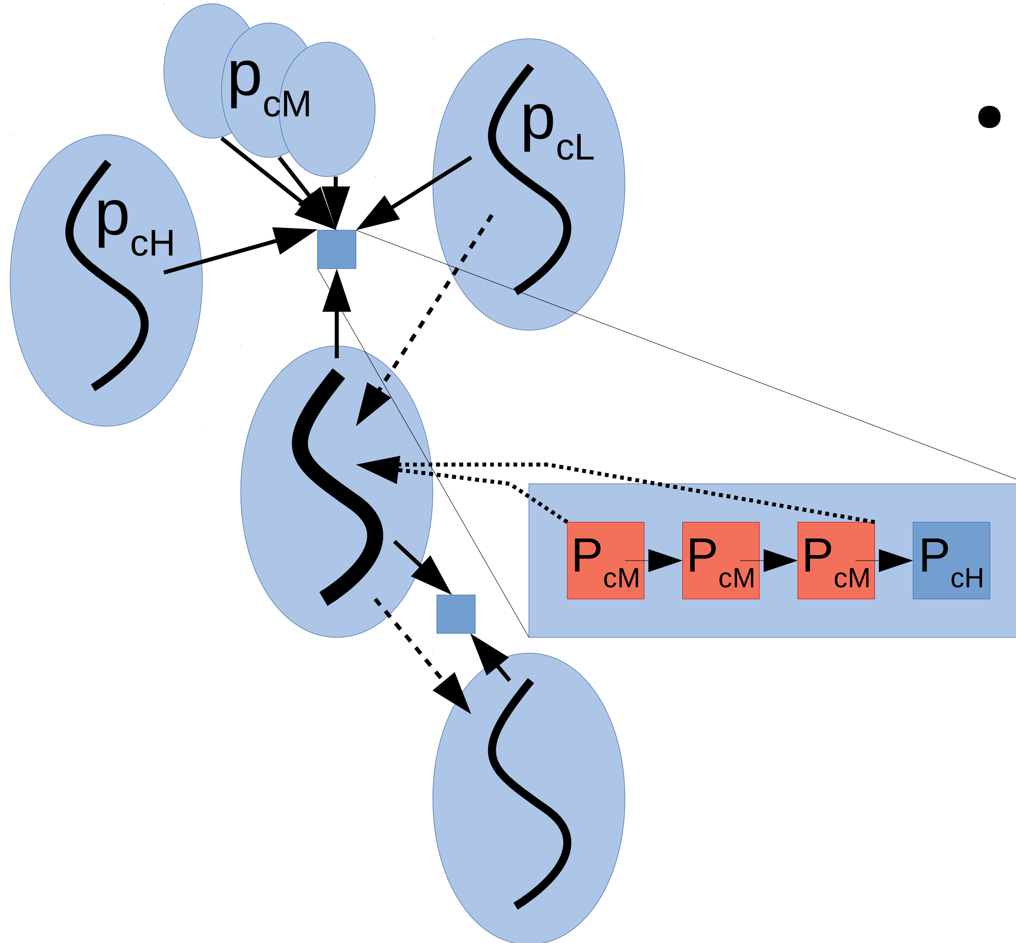
- Priority inversion in shared end-points
  - O(1) → inversion

L4: Priority Inversion



- Priority inversion in shared end-points
  - O(1) → inversion

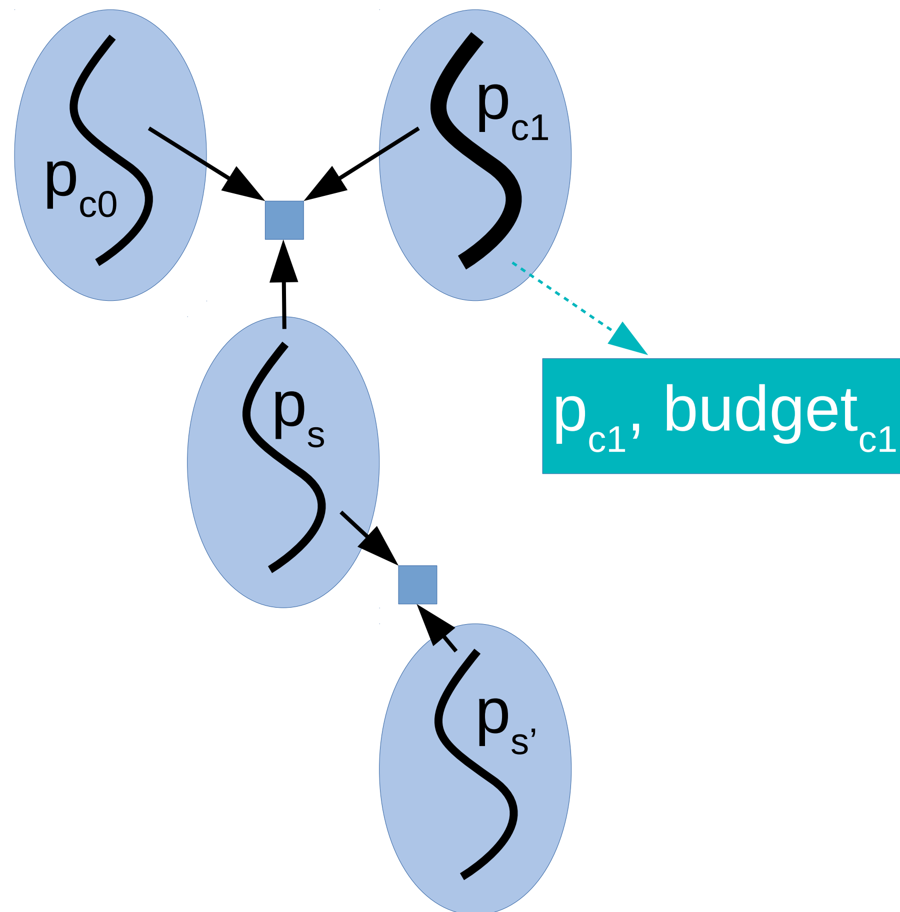
L4: Priority Inversion



- Priority inversion in shared end-points
  - O(1) → inversion
  - O(N) / O(1) / O(log n) → complex queues, preemption points

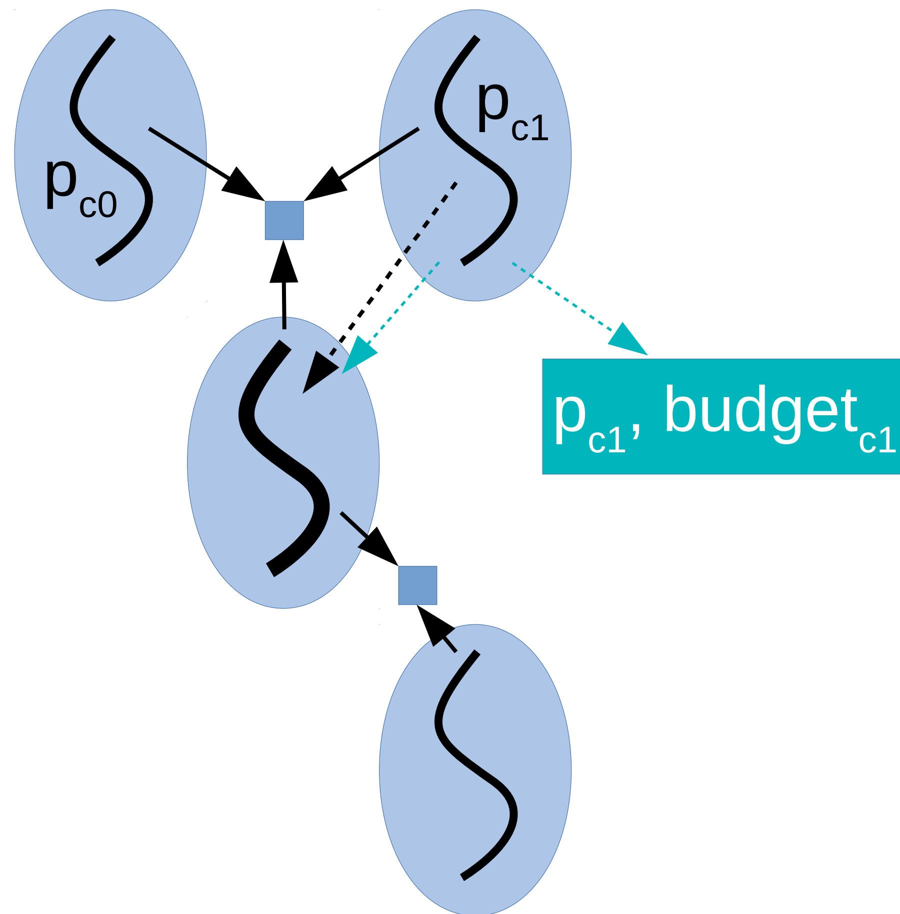
Fiasco L4/Nova – Credo

- Decouple *scheduling context & execution context*
  - *Scheduling context*: priority + budget
  - *Execution context*: registers + stack



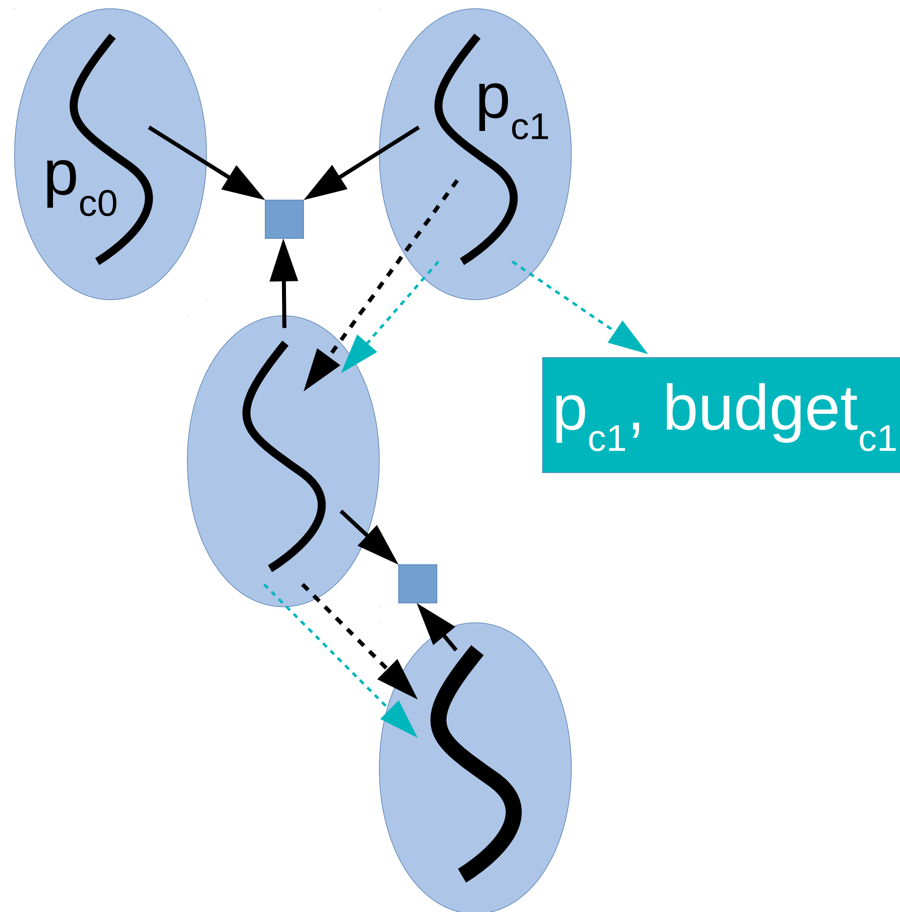
Fiasco L4/Nova – Credo

- Decouple *scheduling context & execution context*
  - *Scheduling context*: priority + budget
  - *Execution context*: registers + stack



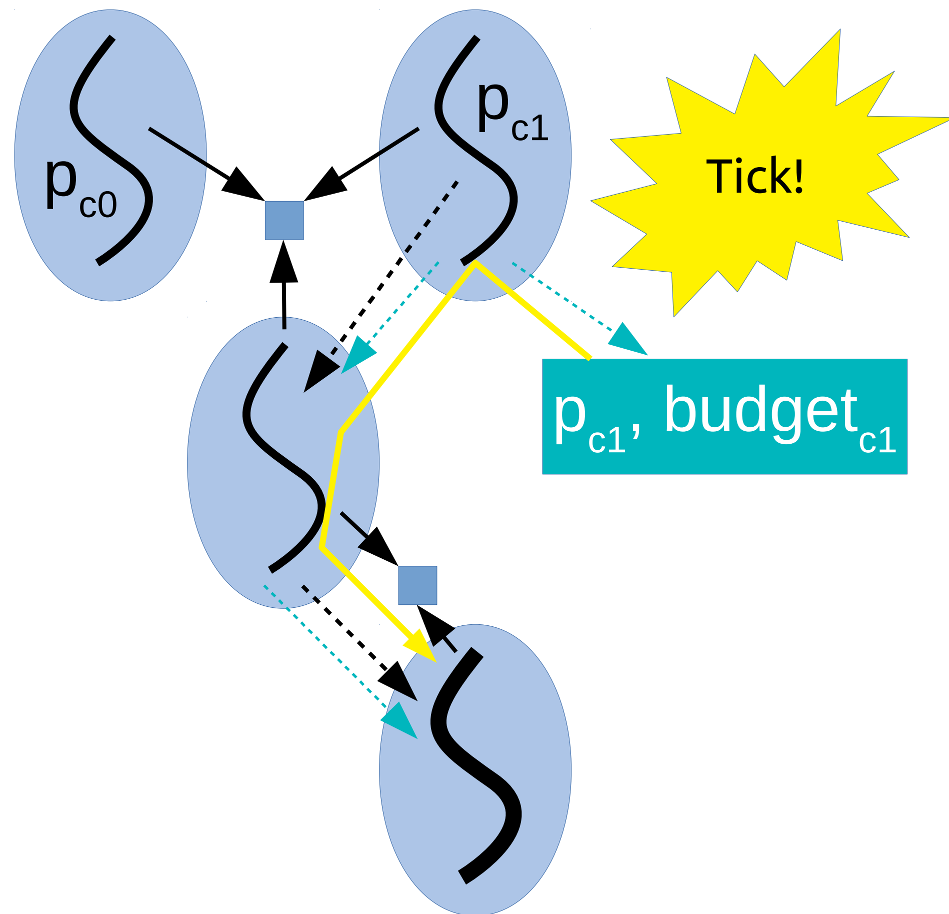
Fiasco L4/Nova – Credo

- Decouple *scheduling context & execution context*
  - *Scheduling context*: priority + budget
  - *Execution context*: registers + stack



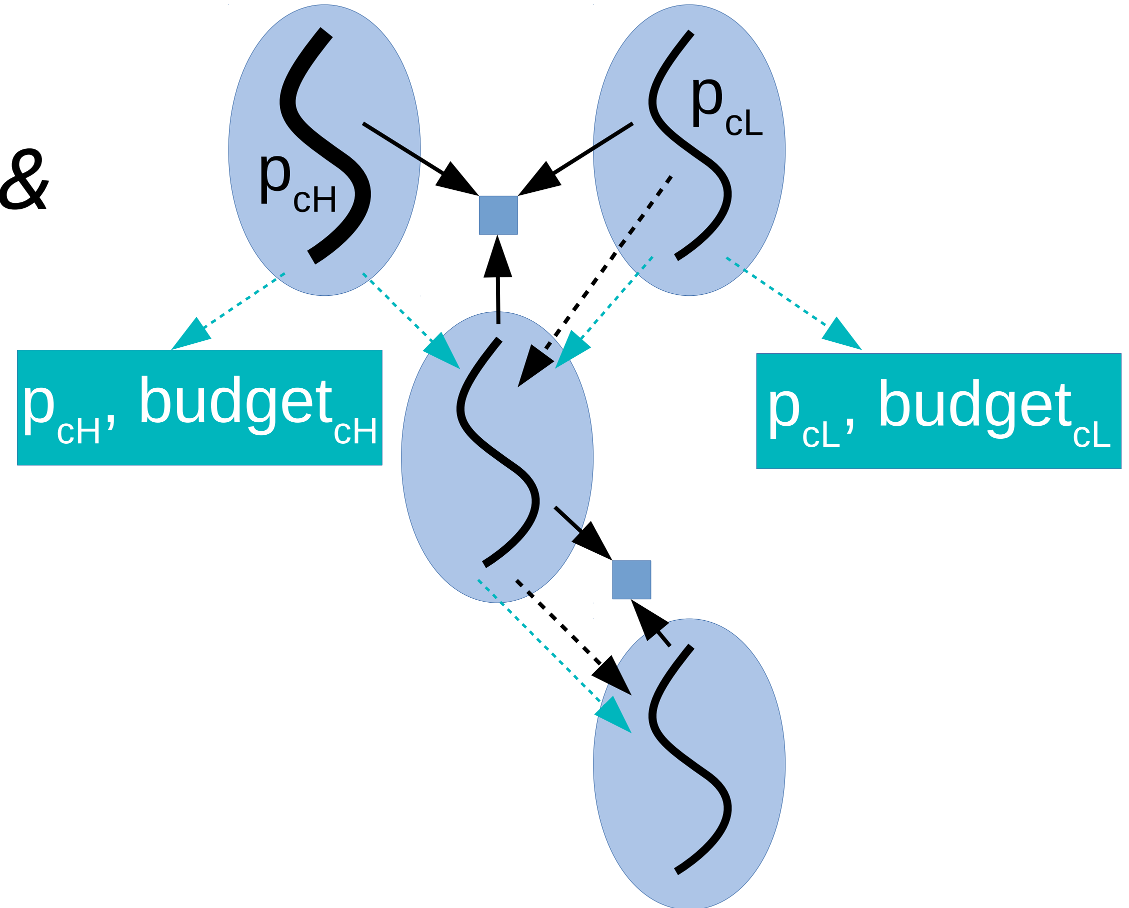
Fiasco L4/Nova – Credo

- Decouple *scheduling context & execution context*
  - *Scheduling context*: priority + budget
  - *Execution context*: registers + stack



Fiasco L4/Nova – Credo

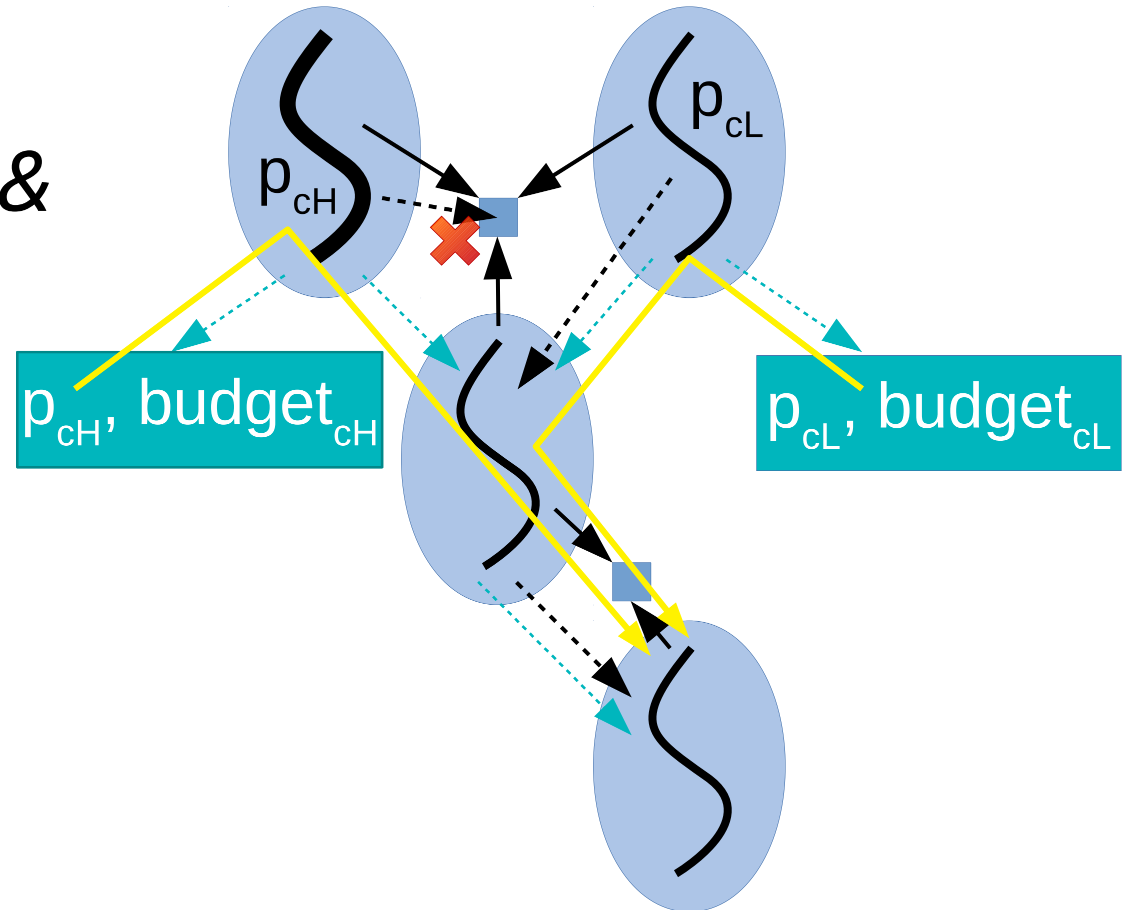
- Decouple *scheduling context & execution context*
  - *Scheduling context*: priority + budget
  - *Execution context*: registers + stack



Steinburg et al. *Fast Component Interaction for Real-Time Systems*, ECRTS
Steinburg et al. *Timeslice Donation in Component-Based Systems*, OSPERT

Fiasco L4/Nova – Credo

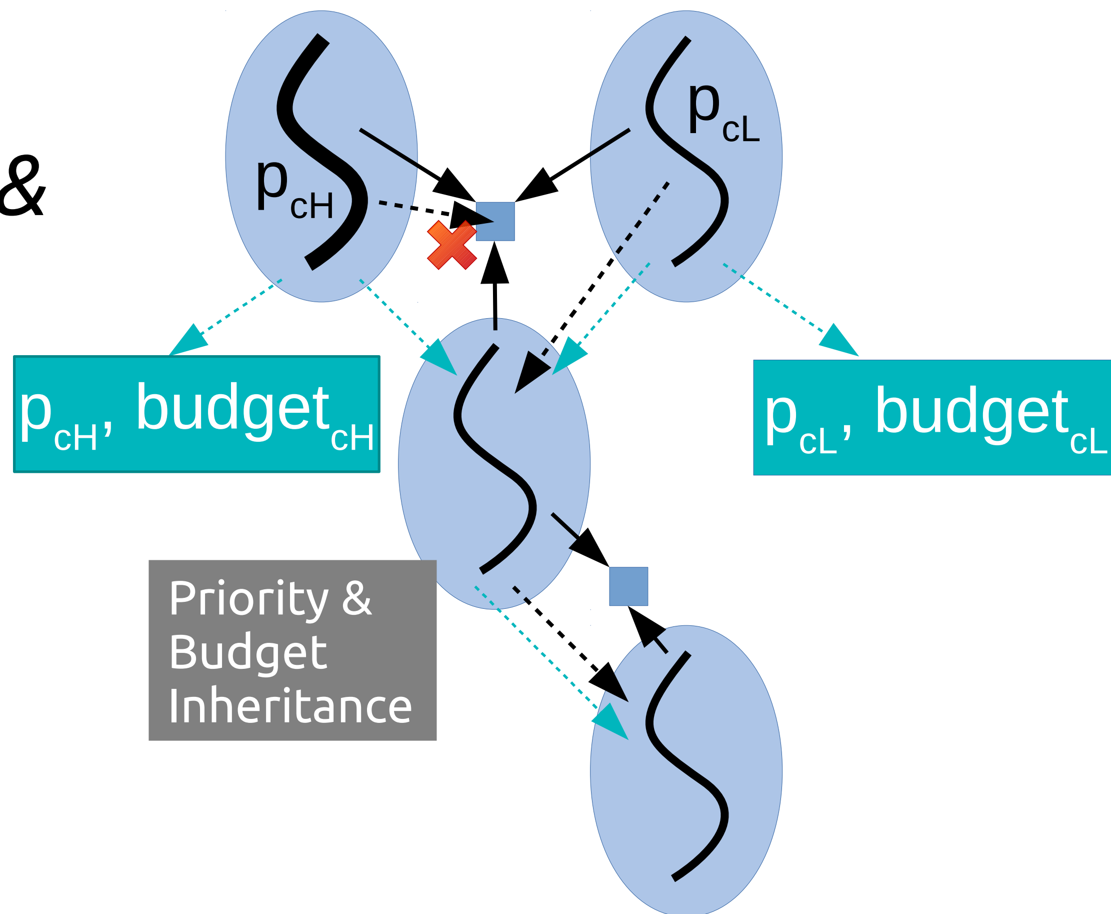
- Decouple *scheduling context & execution context*
  - *Scheduling context*: priority + budget
  - *Execution context*: registers + stack



Steinburg et al. *Fast Component Interaction for Real-Time Systems*, ECRTS
Steinburg et al. *Timeslice Donation in Component-Based Systems*, OSPERT

Fiasco L4/Nova – Credo

- Decouple *scheduling context & execution context*
  - *Scheduling context*: priority + budget
  - *Execution context*: registers + stack

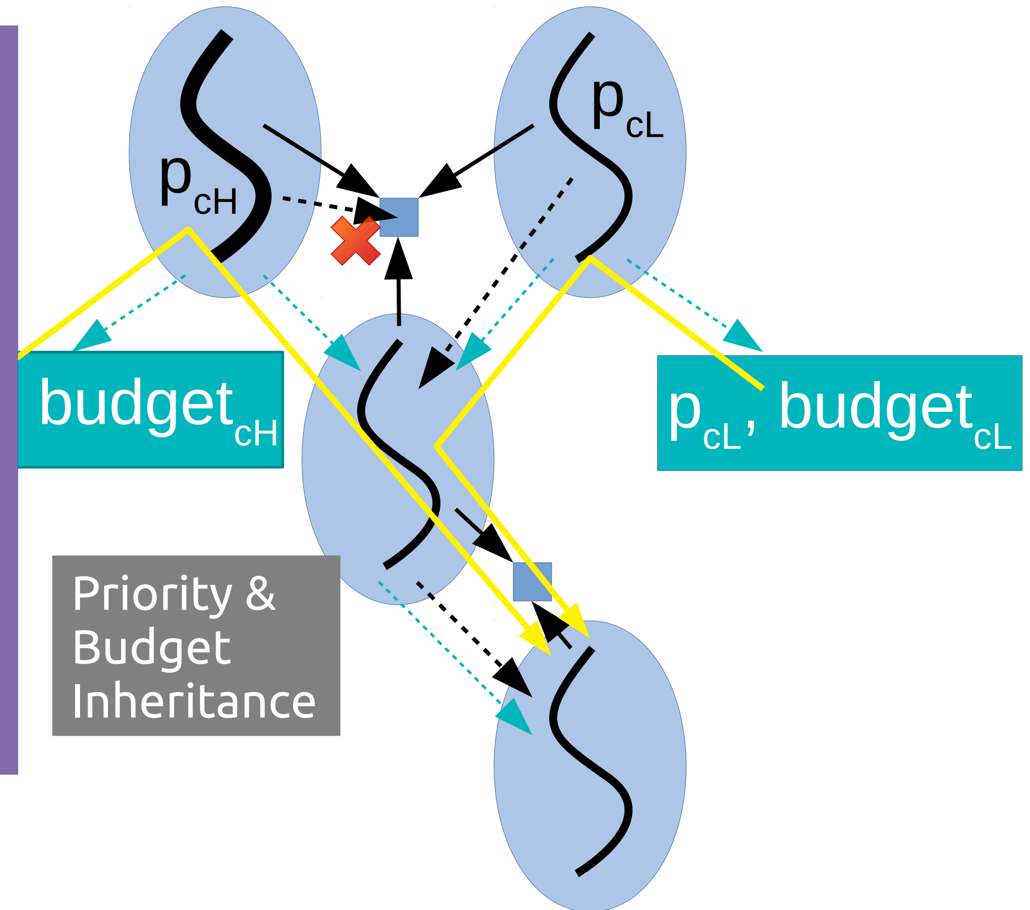


Steinburg et al. *Fast Component Interaction for Real-Time Systems*, ECRTS
Steinburg et al. *Timeslice Donation in Component-Based Systems*, OSPERT

Fiasco L4/Nova – Credo

Benefits:

- Maintains lazy scheduling
- Efficient
- prio = max(path), budget = client

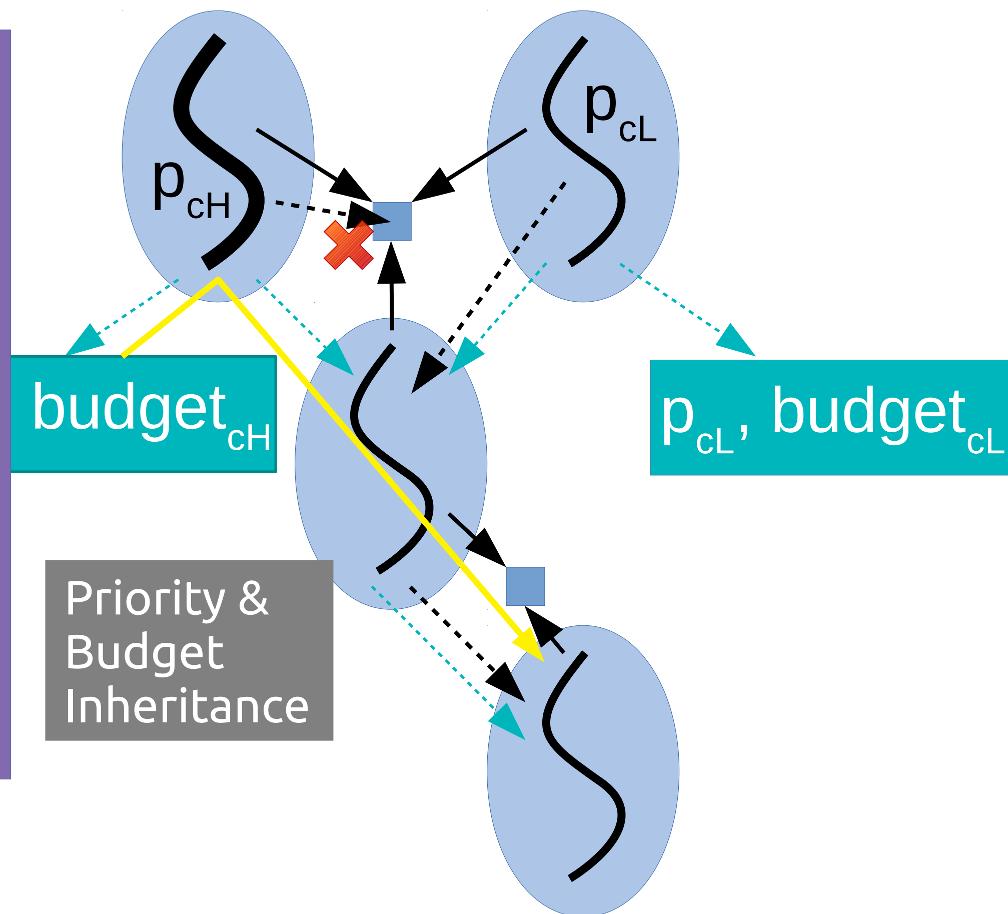


Steinburg et al. *Fast Component Interaction for Real-Time Systems*, ECRTS
Steinburg et al. *Timeslice Donation in Component-Based Systems*, OSPERT

Fiasco L4/Nova – Credo

Benefits:
- Maintains lazy scheduling
- Efficient
- prio = max(path), budget = client

Downsides:
- Preemptions require chasing dependencies (O(n))
- Policy in the kernel
- OPCP challenging
- **Running out of budget in server**



Steinburg et al. *Fast Component Interaction for Real-Time Systems*, ECRTS
Steinburg et al. *Timeslice Donation in Component-Based Systems*, OSPERT

Fiasco L4 Scheduling Contexts

- Budget + priority

N threads >> **M** SCs >> **1** vCPU

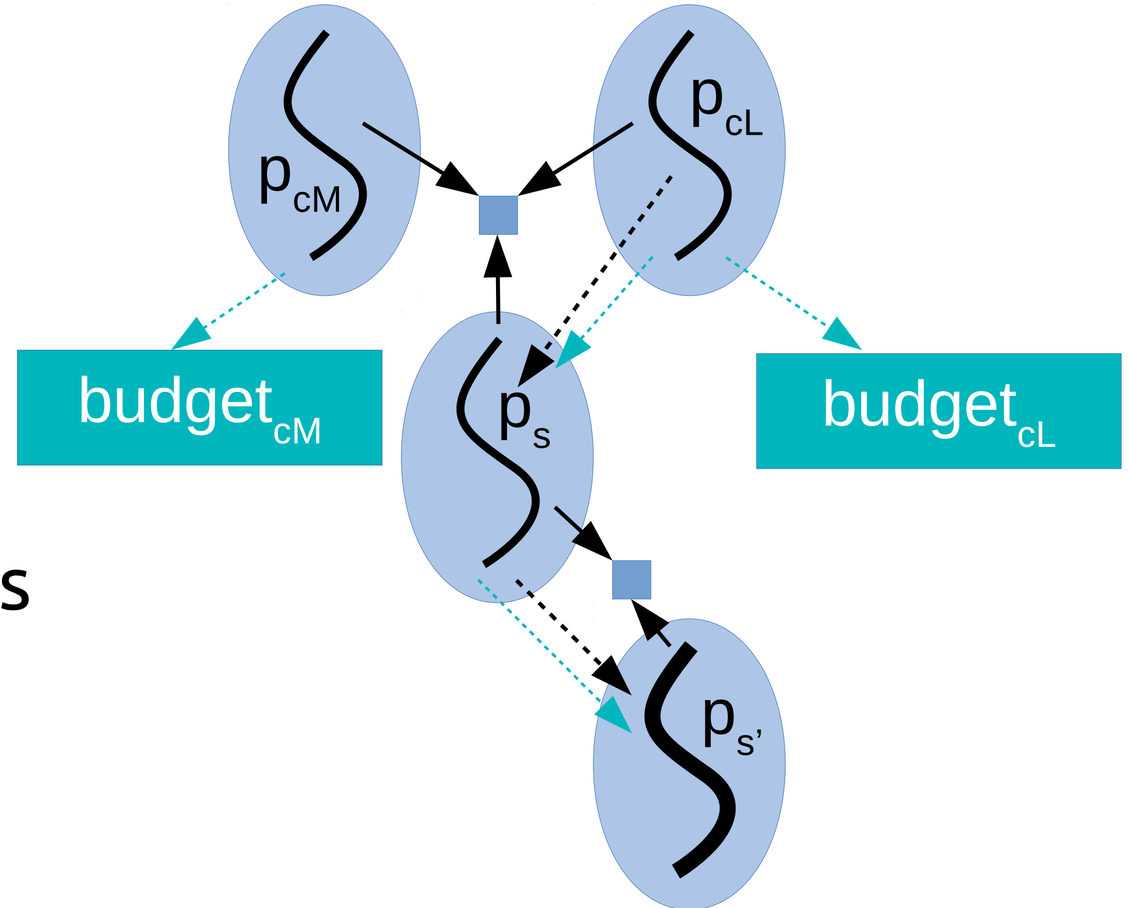
→ different scheduling contexts in same VM

- Budgets consumed/replenished using deferrable servers (every period)

Lackorzynski et al. *Fattening Hierarchical Scheduling*, EMSOFT '12

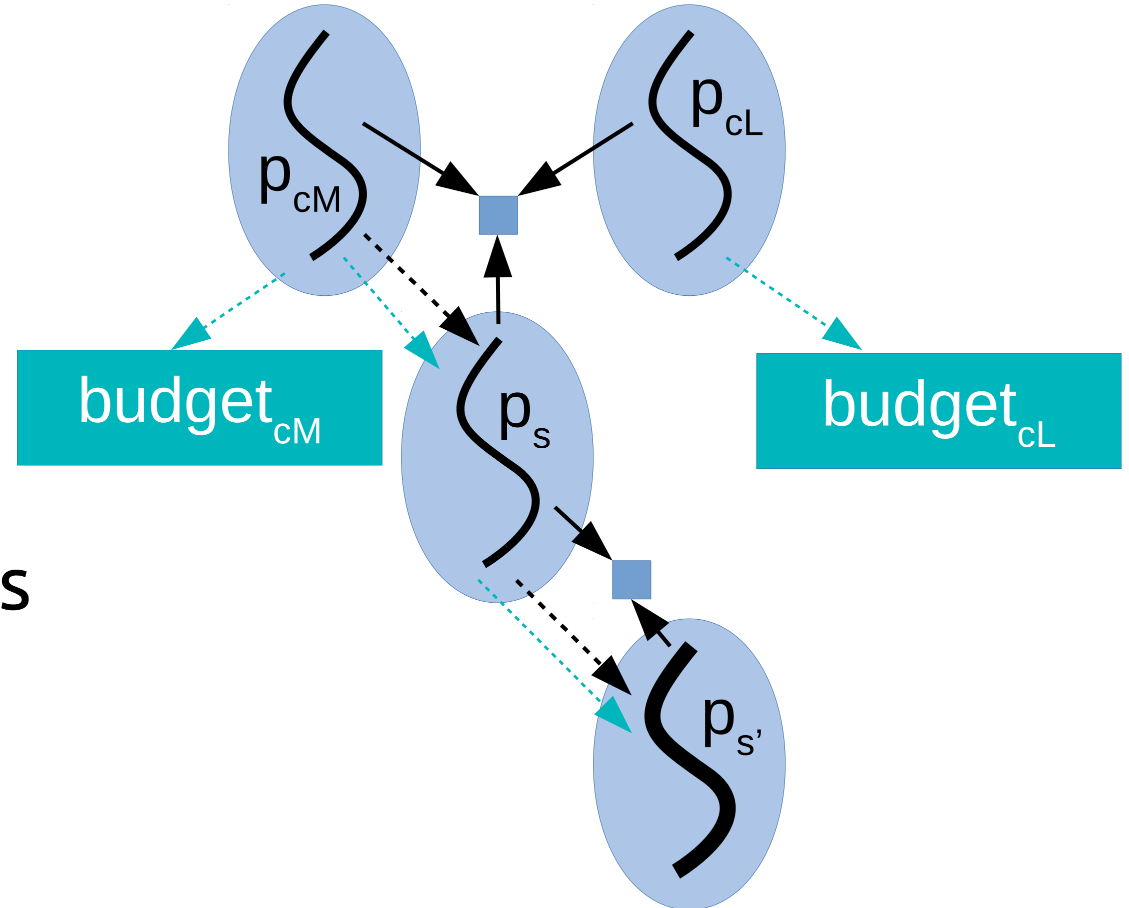
seL4 MCS Extension

- Budgets inherited
- Priorities aren’t
- Budgets
  - Sporadic servers
  - Timeout exceptions



seL4 MCS Extension

- Budgets inherited
- Priorities aren’t
- Budgets
  - Sporadic servers
  - Timeout exceptions

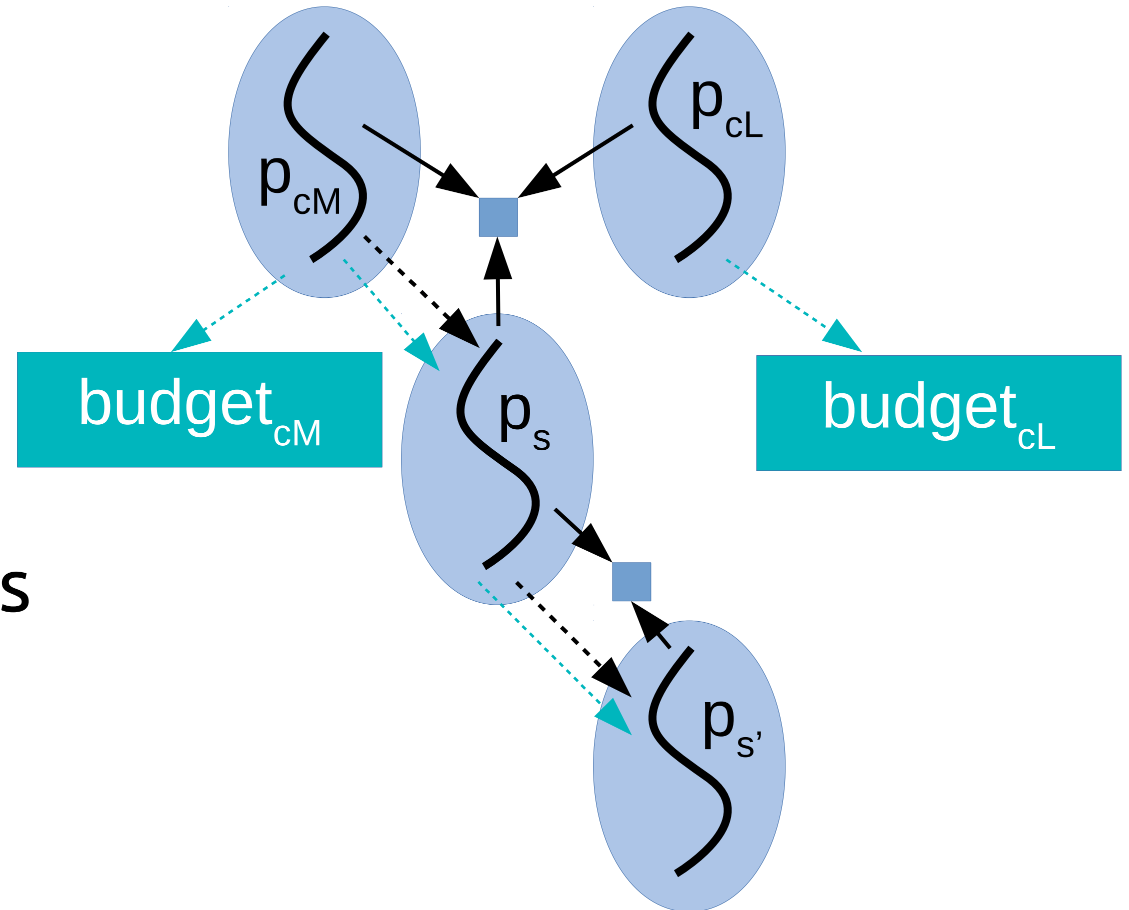


seL4 MCS Extension

- Budgets inherited
- Priorities aren't
- Budgets

Sporadic servers

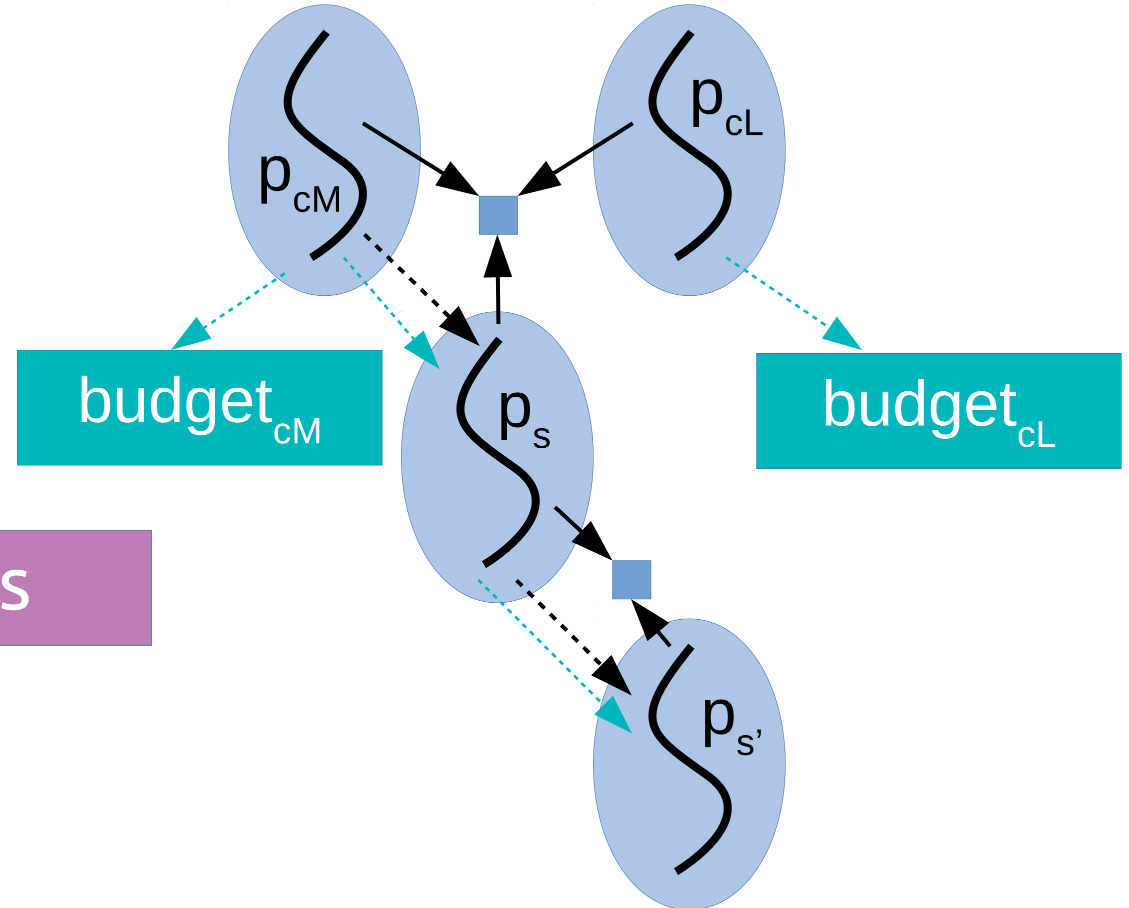
  - Timeout exceptions



seL4 MCS Extension

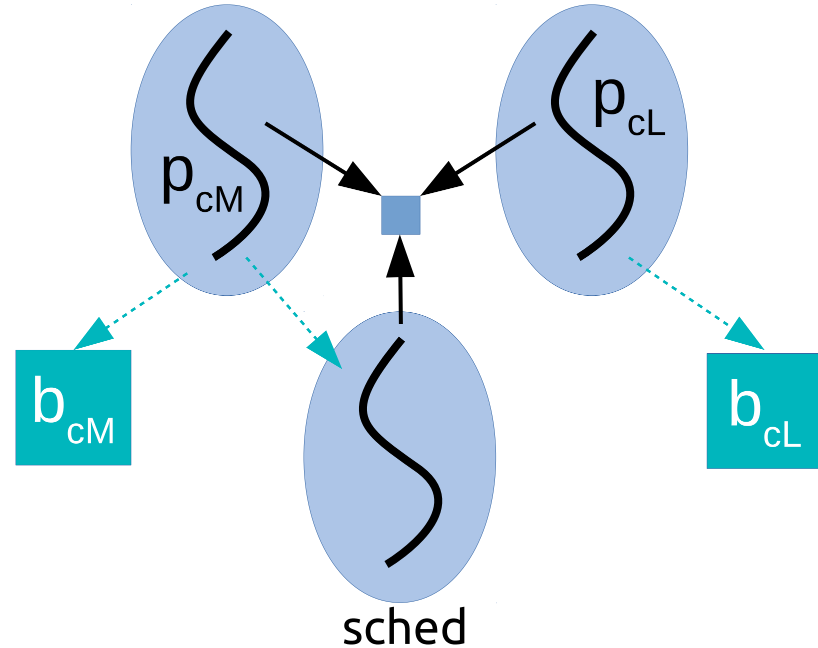
- Budgets inherited
- Priorities aren't
- Budgets
  - Sporadic servers

Timeout exceptions



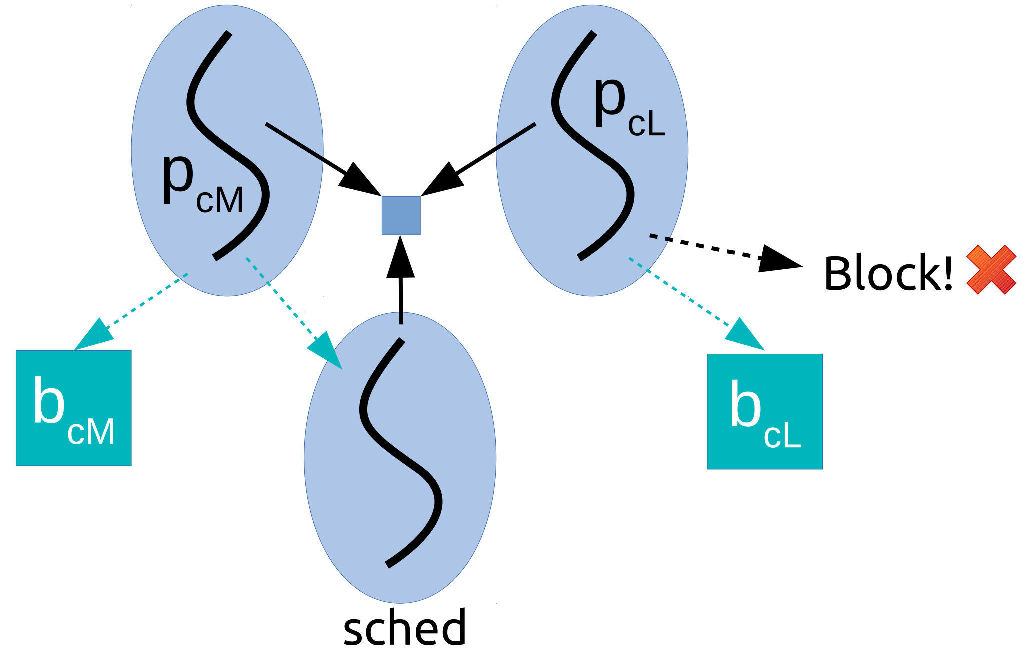
seL4 MCS → User-Level Scheduling

- Sync. IPC
  → coop switch
- Timeout exceptions
  → preemption



seL4 MCS → User-Level Scheduling

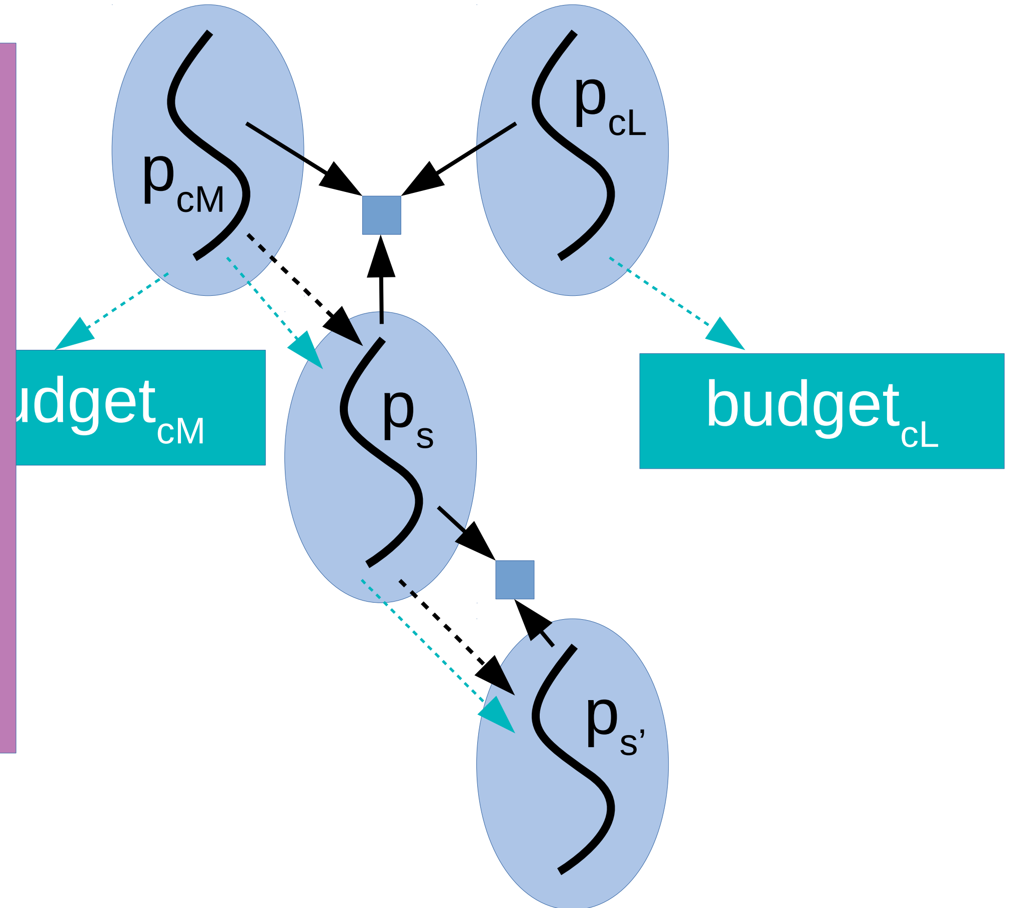
- Sync. IPC
  → coop switch
- Timeout exceptions
  → preemption
- Challenge
  - Blocking kernel APIs



seL4 MCS Extension

Benefits

- Avoids DoS on server
- User-level scheduling w/ only timeout exceptions + cooperative blk



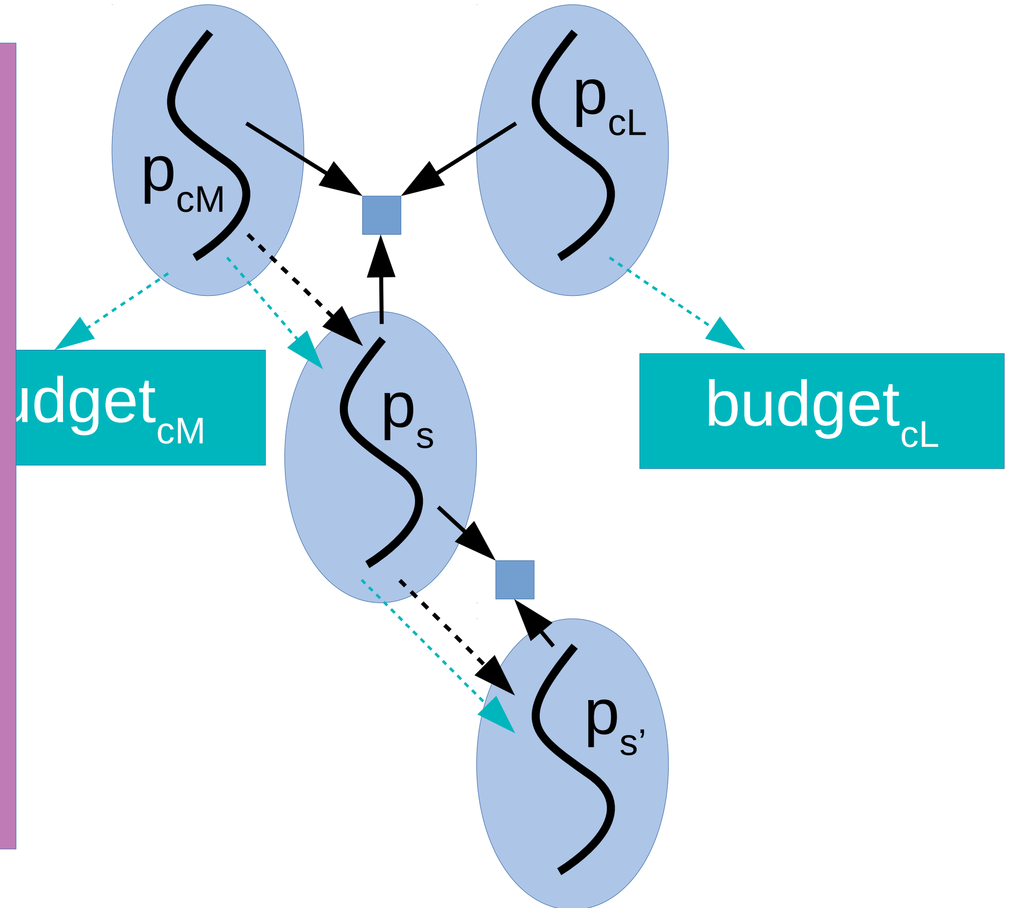
seL4 MCS Extension

Benefits

- Avoids DoS on server
- User-level scheduling w/ only timeout exceptions + cooperative blk

Downsides

- Prioritization challenges
- Care in using the sporadic servers
  → unpredictable preemptions
  → sorting timeouts
- No user-level scheduling w/ blocking APIs



Priority Solutions

1) Maintain verification:

- Server must
  - have higher prio than any client, and
  - must not block (even transitively during sync. IPC)
- Separate server threads & endpoints per set of clients
  - Service verification? Shared resources in service?
- Don’t have shared services – partitioning

Priority Solutions

1) Maintain verification:

- Server must
  - have higher prio than any client, and
  - must not block (even transitively during sync. IPC)
- Separate server threads & endpoints per set of clients
  - Service verification? Shared resources in service?
- Don’t have shared services – partitioning

VMM Implementation

2) Priority inheritance + budget + ep Q sorting

- ala Nova/Credo – O(dependency length)
- sort endpoint queues

Priority Solutions

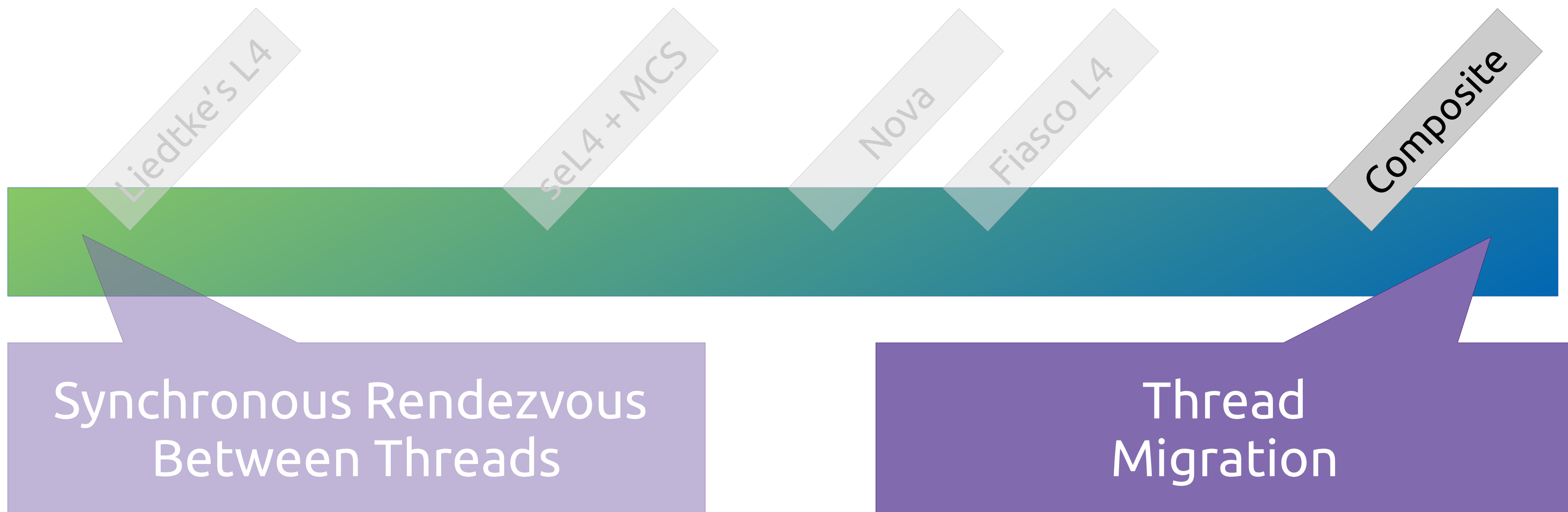
1) Maintain verification:

- Server must
  - have higher prio than any client, and
  - must not block (even transitively during sync. IPC)
- Separate server threads & endpoints per set of clients
  - Service verification? Shared resources in service?
- Don’t have shared services – partitioning

2) Priority inheritance + budget + ep Q sorting

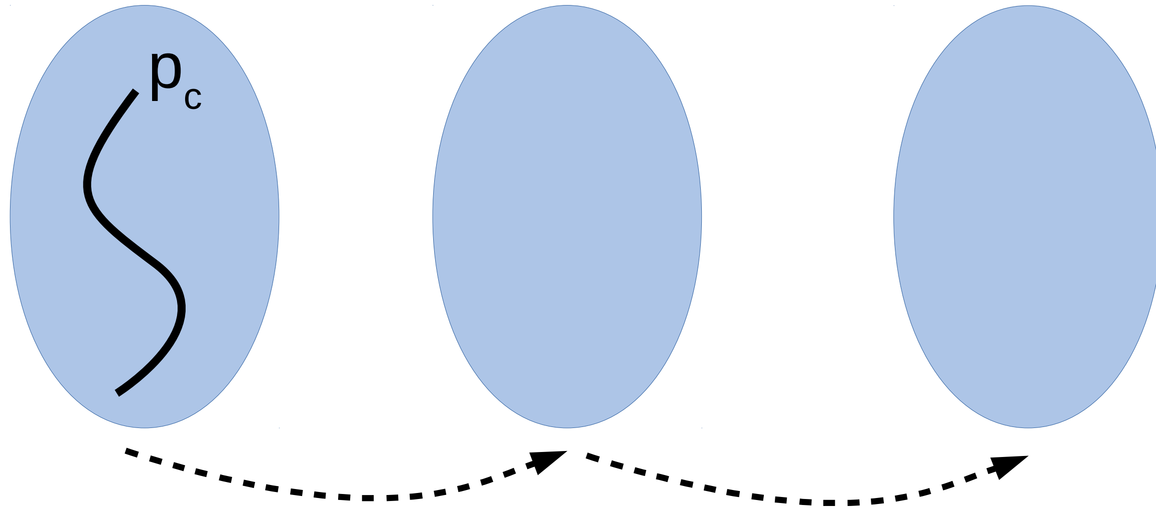
- ala Nova/Credo – O(dependency length)
- sort endpoint queues

Spectrum of IPC Mechanisms

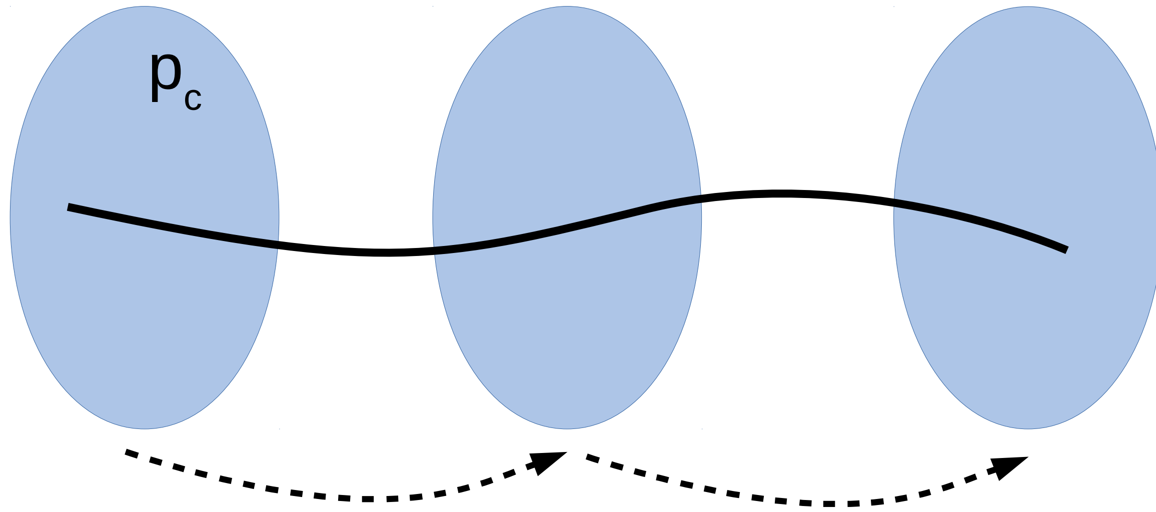


Ford et al. *Evolving Mach 3.0 to a Migrating Thread Model*, USENIX Winter '94
Gabber et al. *The Pebble Component-Based Operating System*, USENIX ATC '99

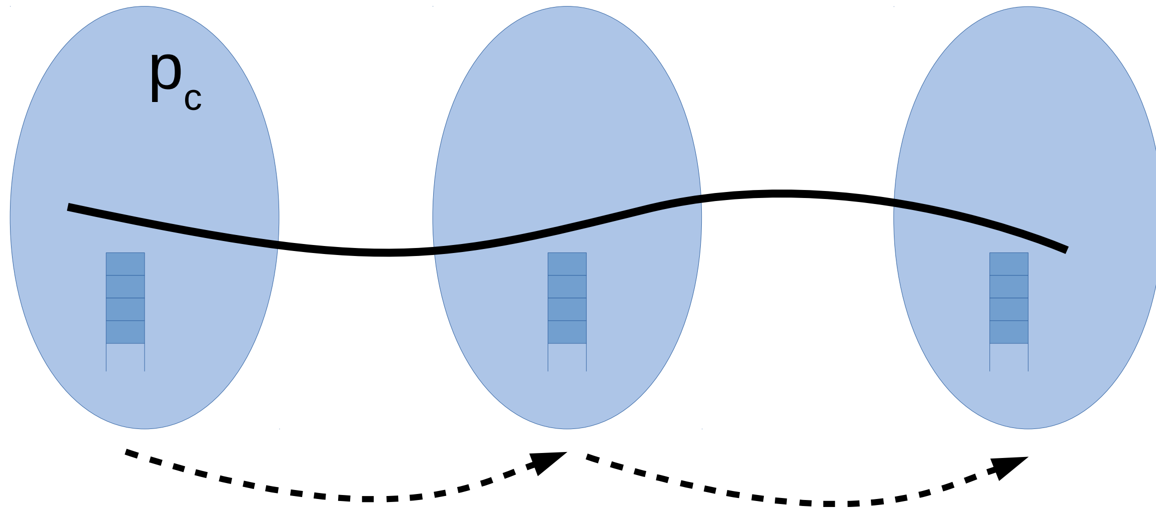
Thread Migration



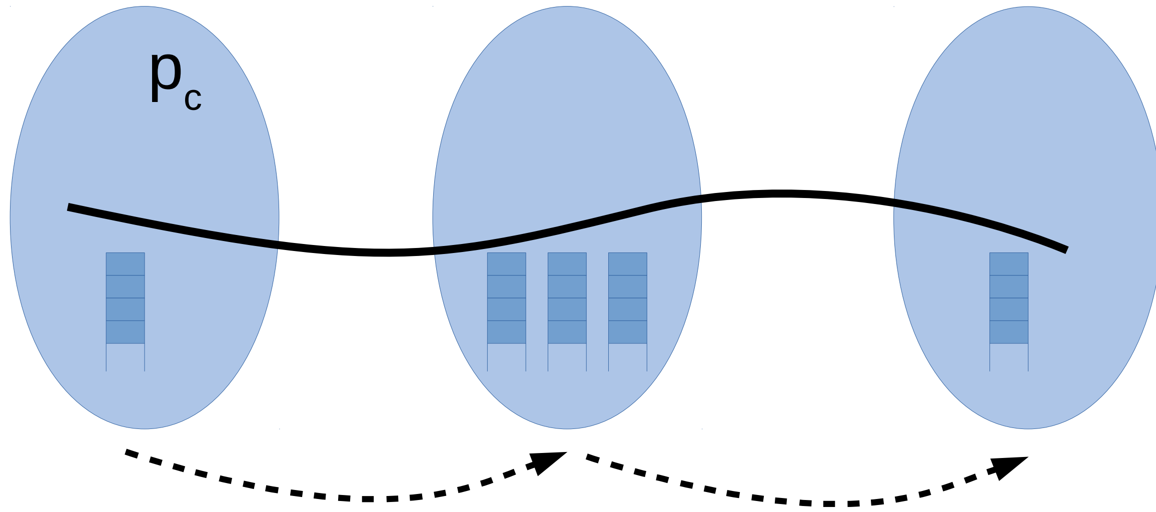
Thread Migration



Thread Migration

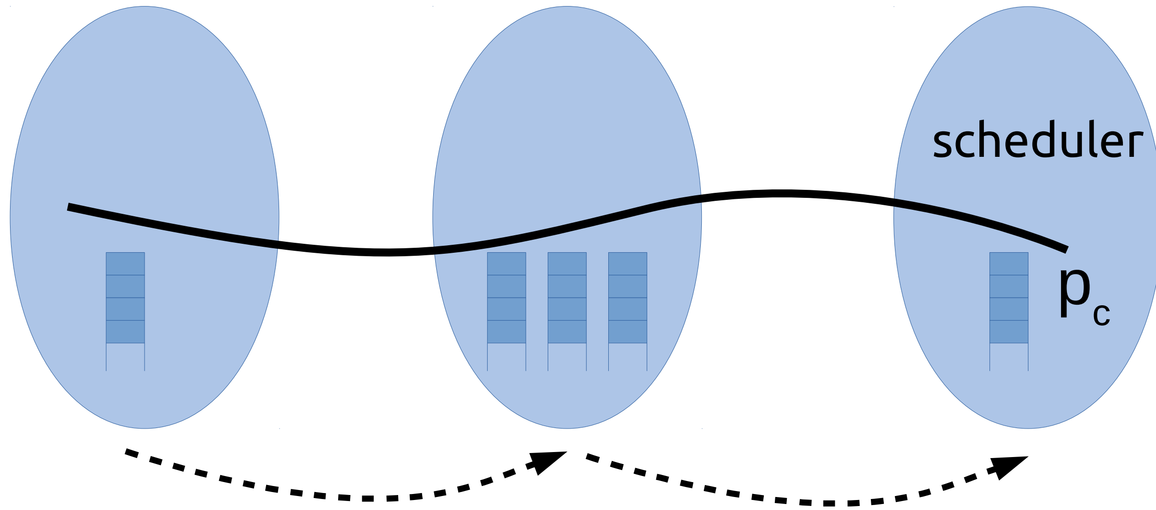


Thread Migration

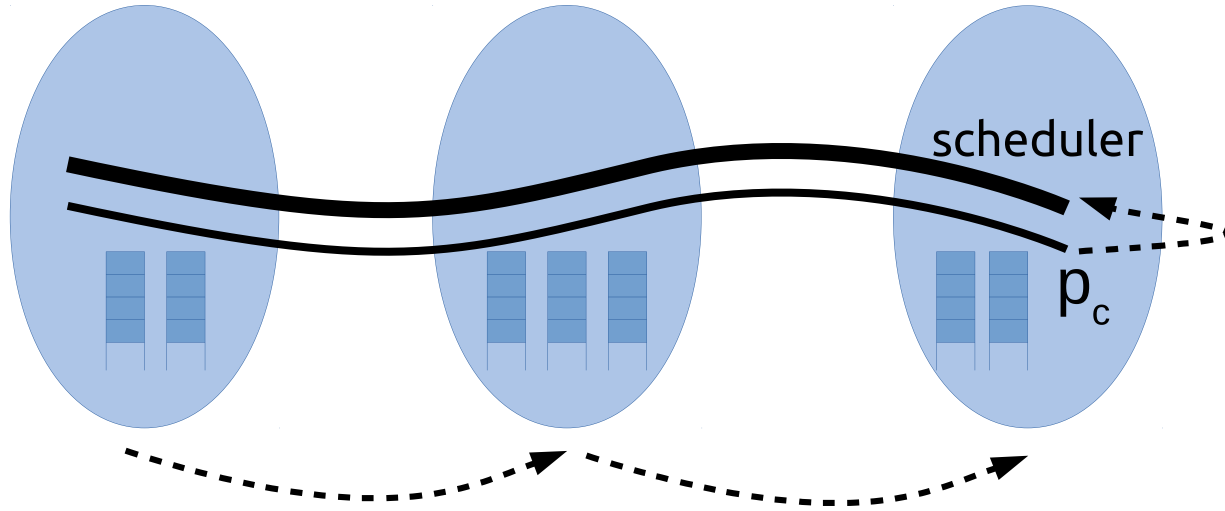


Wang et al. *Execution Stack Management for Hard Real-Time Computation in a Component-Based OS*, RTSS '11

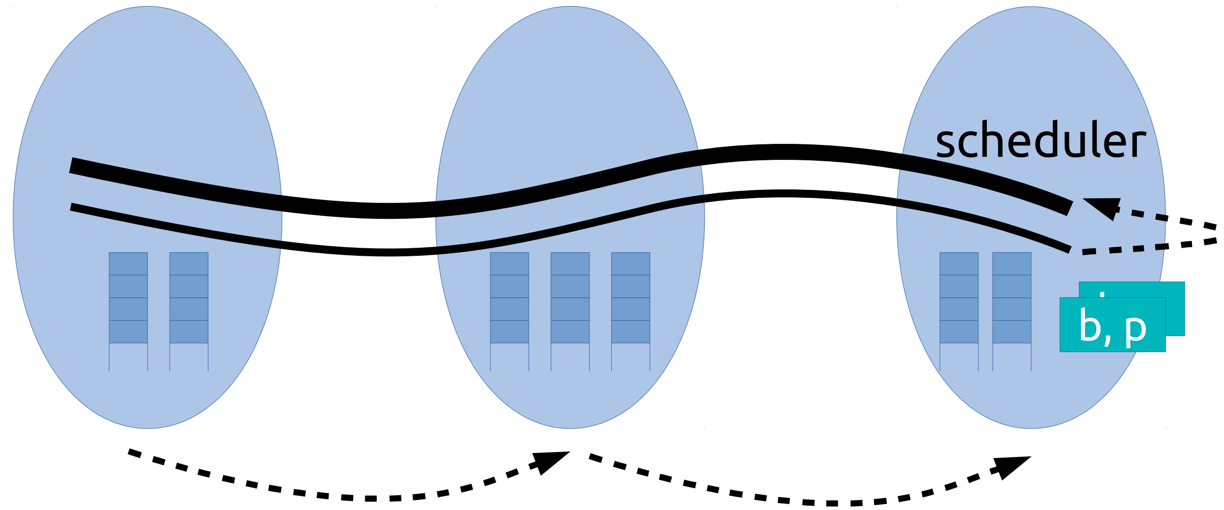
Thread Migration



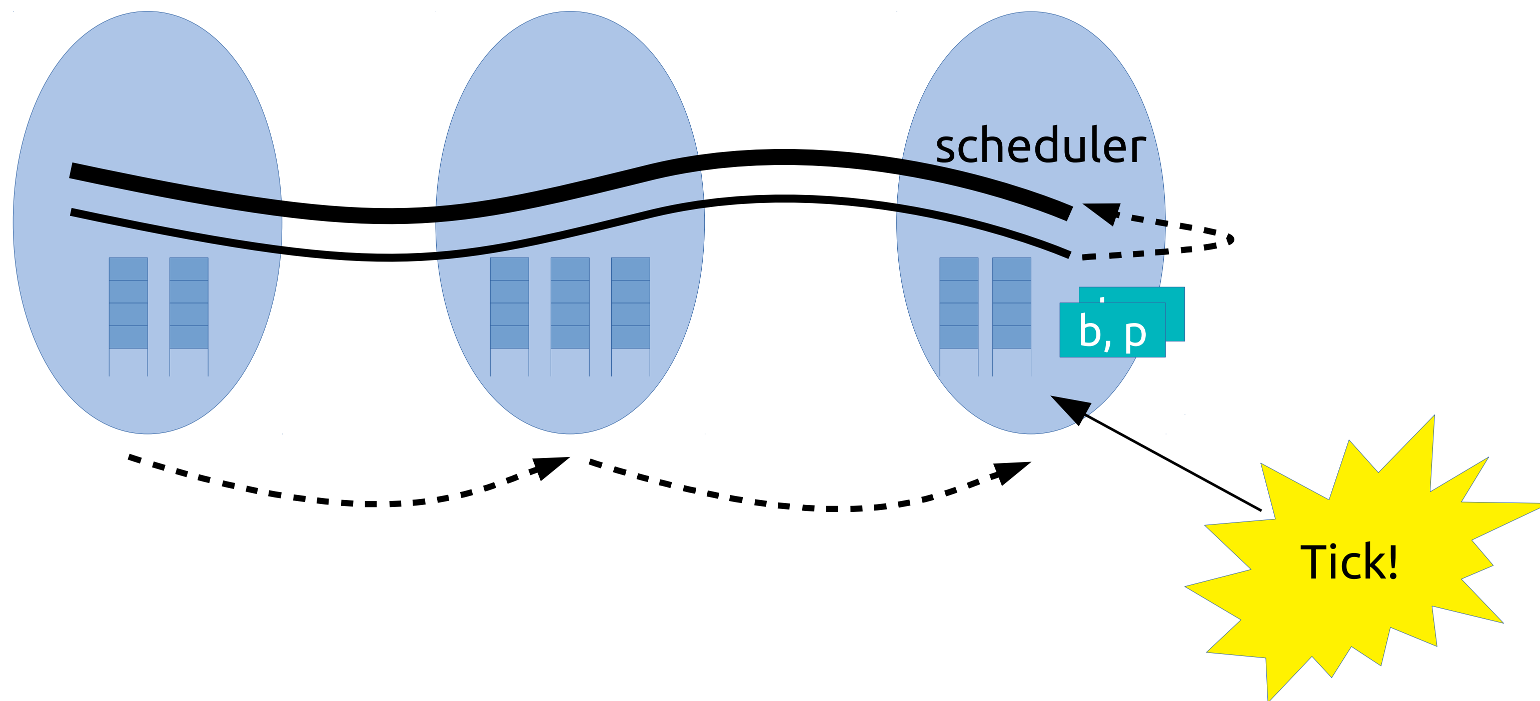
Thread Migration



Thread Migration

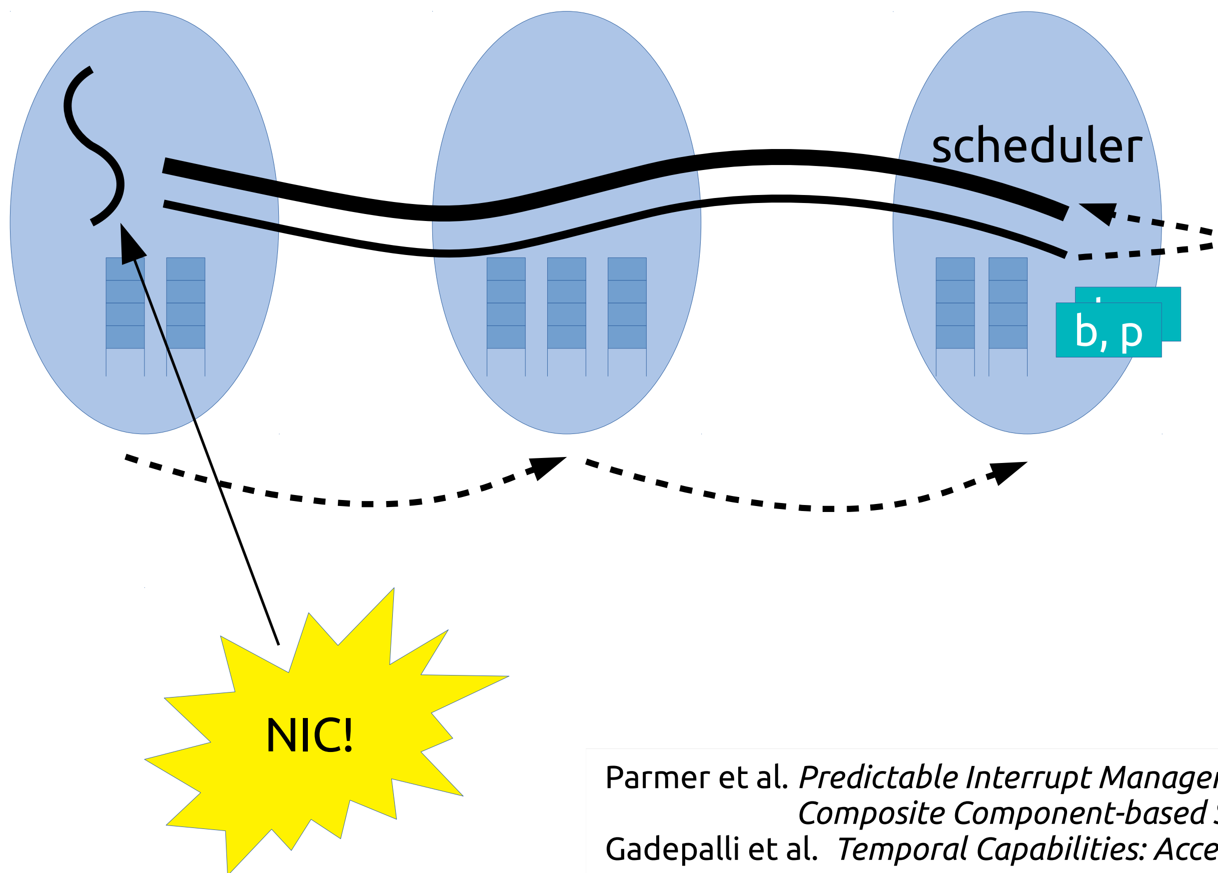


Thread Migration



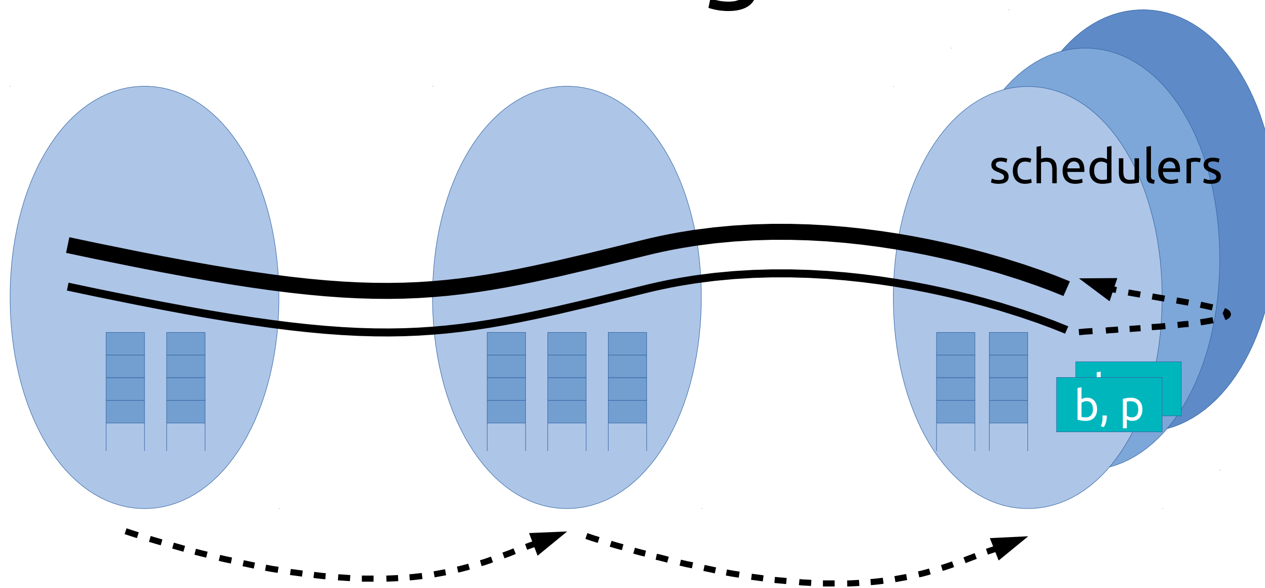
Parmer et al. *Predictable Interrupt Management and Scheduling in the Composite Component-based System*, RTSS '08
Gadepalli et al. *Temporal Capabilities: Access Control for Time*, RTSS '17

Thread Migration



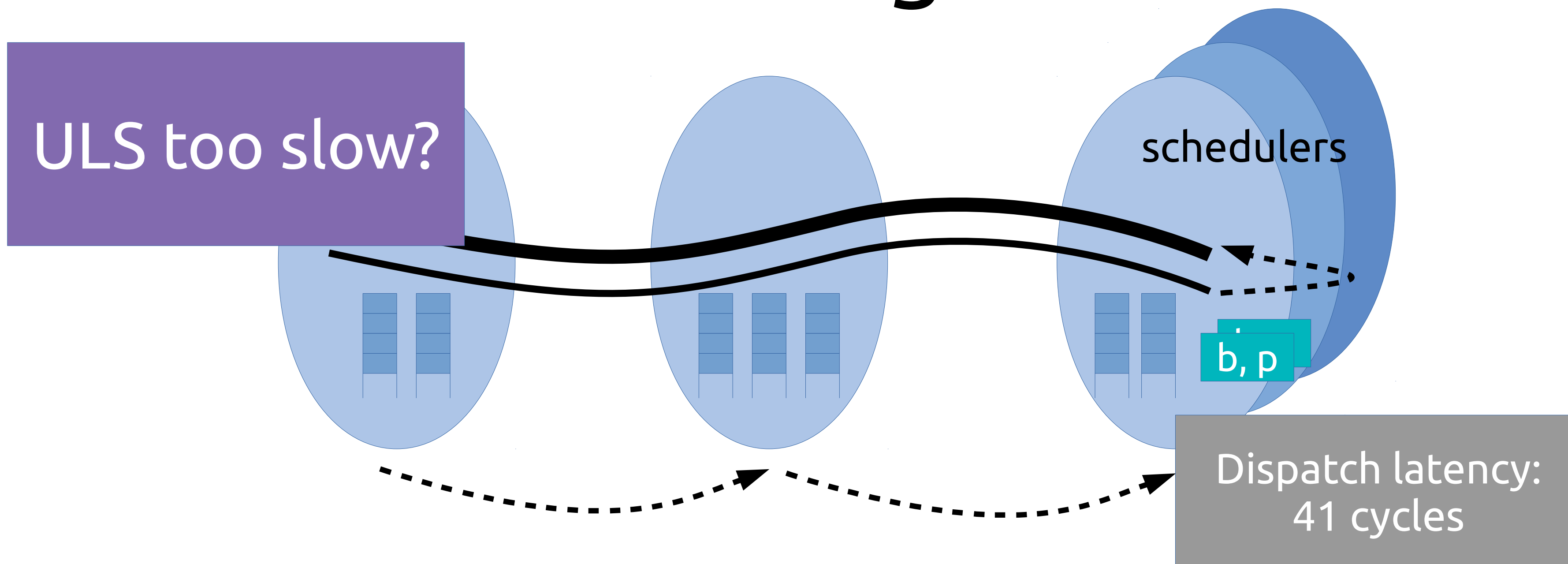
Parmer et al. *Predictable Interrupt Management and Scheduling in the Composite Component-based System*, RTSS ‘08
Gadepalli et al. *Temporal Capabilities: Access Control for Time*, RTSS ‘17

Thread Migration



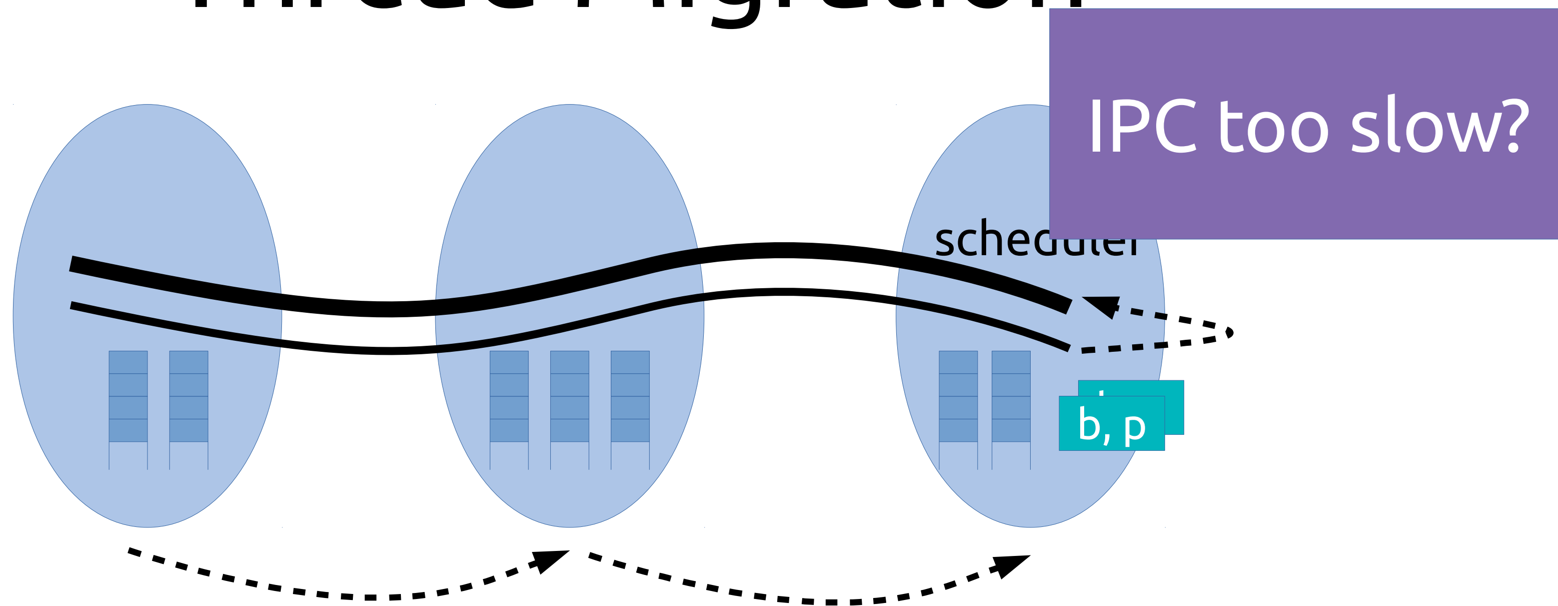
Gadepalli et al. *Temporal Capabilities: Access Control for Time*, RTSS '17
Parmer et al. *HiRes: a System for Predictable Hierarchical Resource Management*, RTAS '11

Thread Migration



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

Thread Migration



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
[Additional publication in submission]

Thread Migration Downsides

- Concurrency-by-default
- Requires *predictable* locks (via scheduler)
- Blocking APIs require IPC to sched

Open question: Is thread-migration completely incompatible with formal verification?

Non-functional Concern: seL4 Toward User-level Scheduling

- Kernel blocking semantics

  → send block/wakeup notifications to a scheduler thread

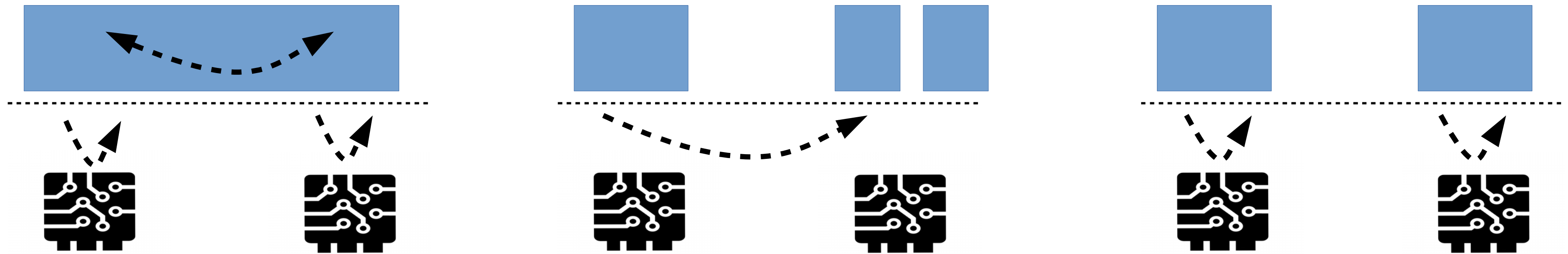
- Overhead on IPC, interrupts (Stoess neg. results)

Stoess, *Towards Effective User-Controlled Scheduling for Microkernel-Based Systems,* SIGOPS OSR, '07

Non-functional Concern: Scalability

Can kernel APIs

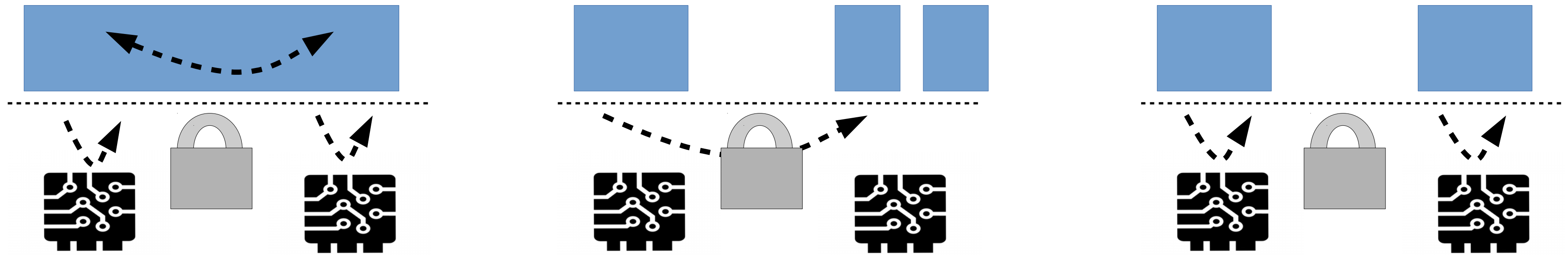
- Limit the scalability of components?
- Cause interference between components?



Non-functional Concern: Scalability

Can kernel APIs

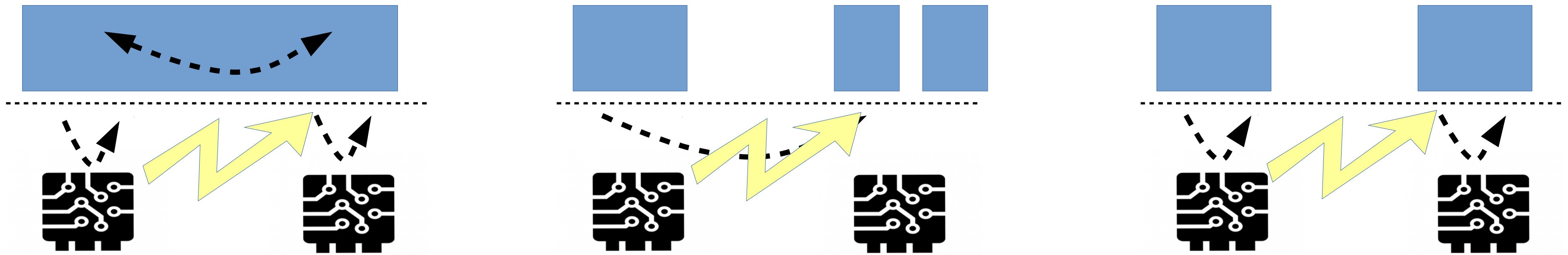
- Limit the scalability of components?
- Cause interference between components?



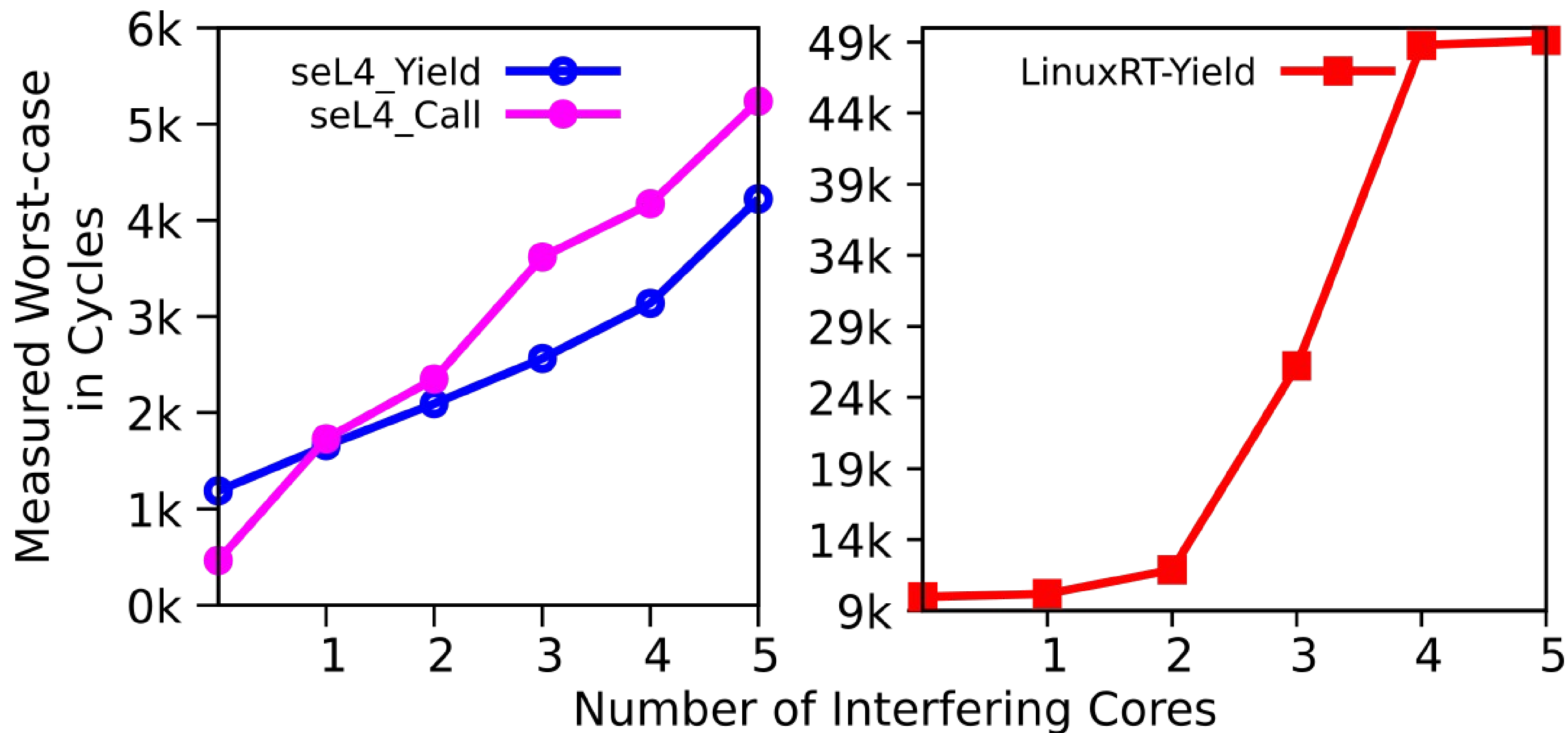
Non-functional Concern: Scalability

Can kernel APIs

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



Impact of Mutual Exclusion



Gadepalli et al. *Chaos: a System for Criticality-Aware, Multi-core Coordination*, RTAS ‘19

No locks in the kernel?

- Lock-free
  - Fiasco: selective application of non-blocking synchronization
  - Composite kernel: entirely wait-free kernel
    - Global progress guarantees
- Barrelfish: multikernel

Hohmuth et al. *Pragmatic nonblocking synchronization for real-time systems*, USENIX ATC '01
Wang et al. *SPeCK: A Kernel for Scalable Predictability*, RTAS '15
Baumann et al. *The Multikernel: A new OS architecture for scalable multicore systems*, SOSP '09

Cross Cutting Concerns: Scalability – seL4 multikernel

- Much more complexity in capability delegation/revocation
- Partitioning resource responsibility
  - Controlled sharing (shmem)
- High-priority IPI interference*?

* Gadepalli et al. *Chaos: a System for Criticality-Aware, Multi-core Coordination*, RTAS ‘19

SeL4 Summary

- Predictability in end-to-end IPC
  - Careful prioritization
  - Partitioning
  - Careful control over sporadic servers
- Predictable scalability
  - Single core
  - Multikernel + partitioning

SeL4 Summary

- Predictability in end-to-end IPC
  - Careful prioritization
  - Partitioning
  - Careful control over sporadic servers
- Predictable scalability
  - Single core
  - Multikernel + partitioning

Some utilization loss?

? || /* */

composite.seas.gwu.edu