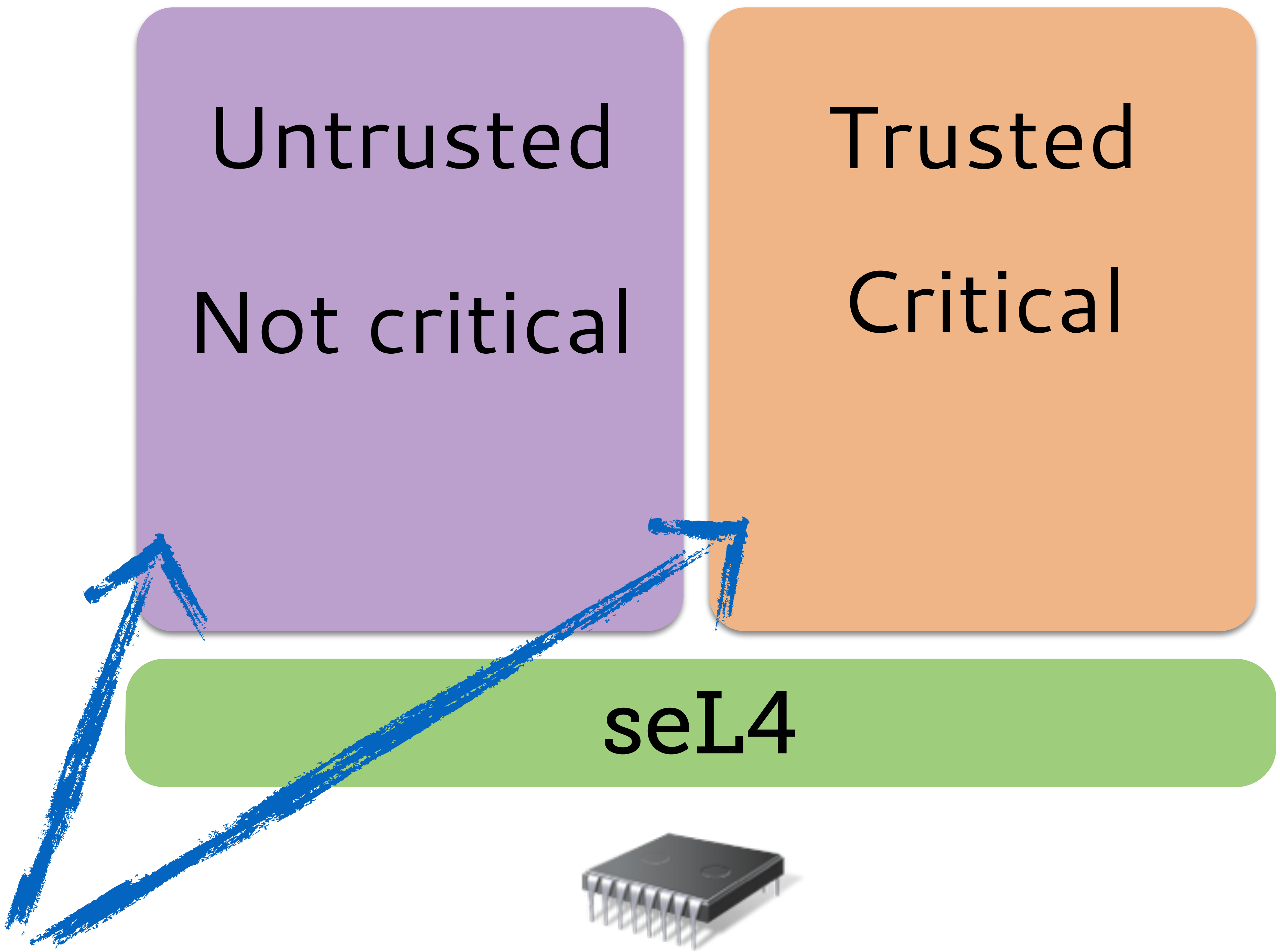
Mixed-Criticality Support in a High-Assurance, General-Purpose Microkernel

Anna Lyons, Gernot Heiser

UNSW Australia & NICTA



Could be OS guests

Untrusted

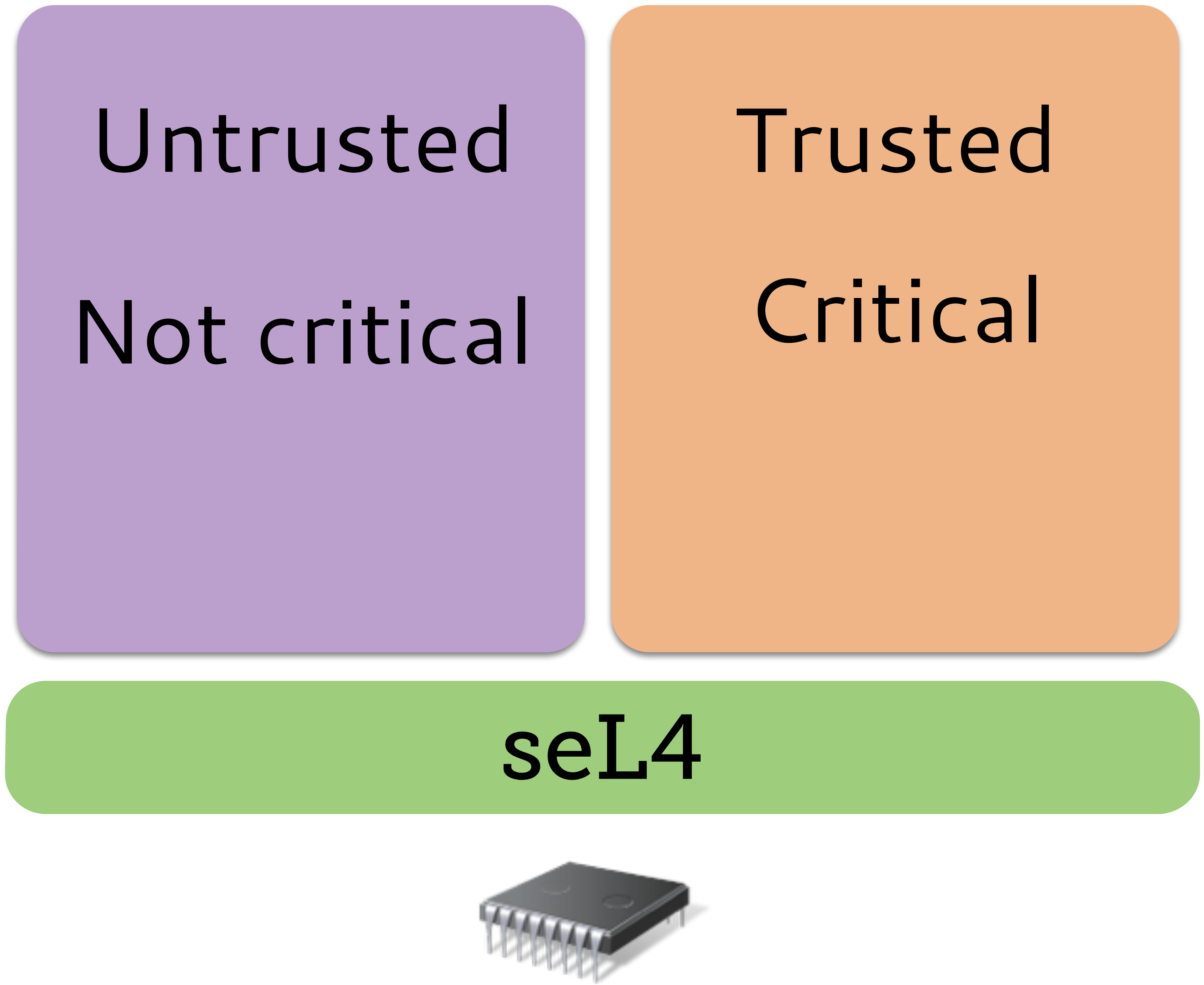
Not critical

Trusted

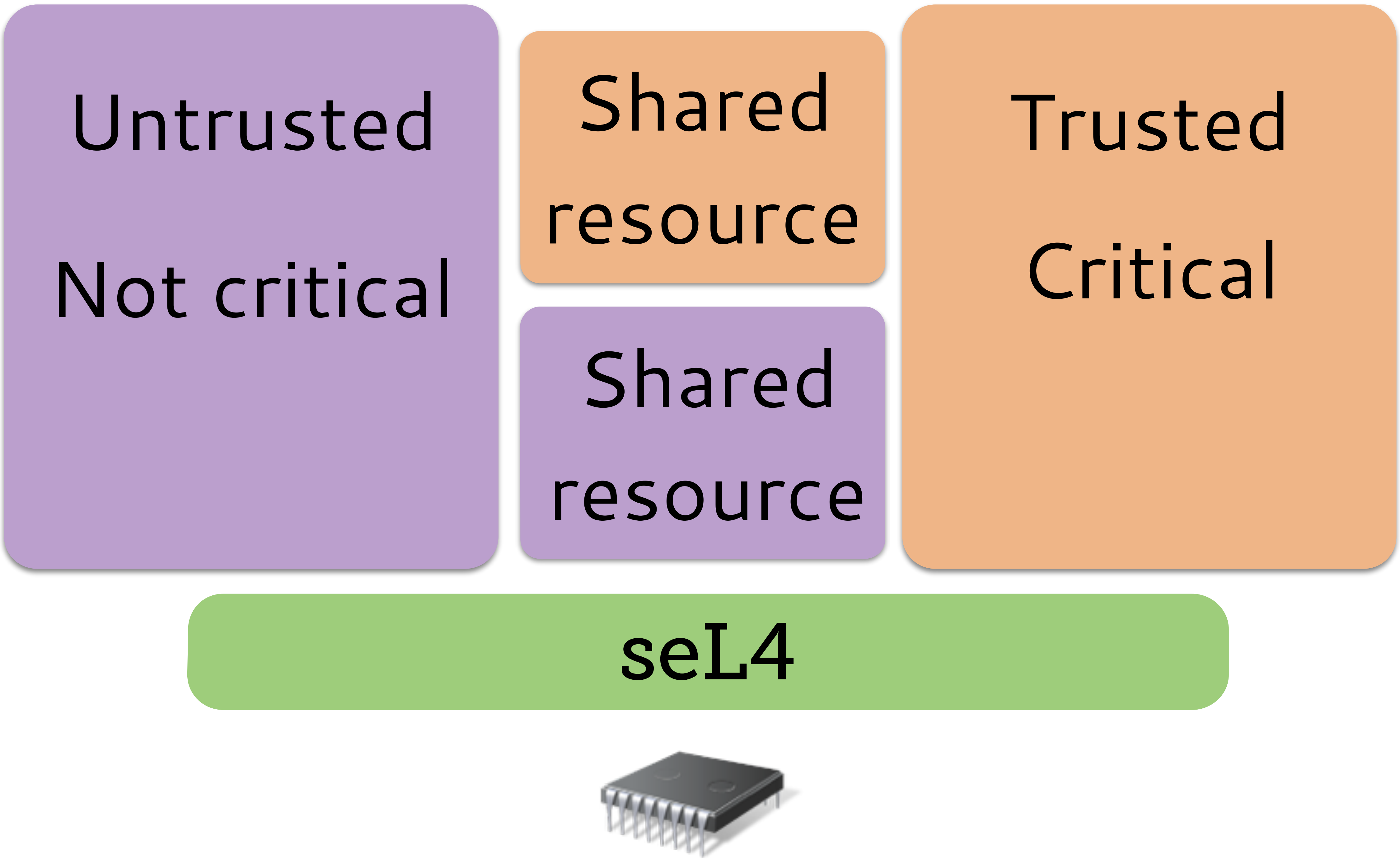
Critical

seL4

Single core (for now)
Has memory management unit


Untrusted
Not critical
Trusted
Critical
seL4

NICTA


Untrusted
Not critical
Shared resource
Shared resource
Trusted
Critical
seL4

seL4

- ☑ Functional Correctness [SOSP'09]
- ☑ Integrity [ITP'11]
- ☑ Timeliness (known WCET) [RTSS'11,EuroSys'12]
- ☑ Translation Correctness [PLDI'13]
- ☑ Non-interference [S&P'13]
- ☑ Fast (258 cycle IPC roundtrip on 1GHz Cortex-A9)
- ☑ Minimal TCB (~9000 SLoC)
- ☐ Safety: specifically temporal properties.

Goals of this work

- Real-time scheduling support
- Temporal isolation (beyond total static partitions)
- Asymmetric temporal protection
  - support for criticality mode changes
- Bounded resource sharing
  - across criticalities

TIME = FIRST CLASS RESOURCE

Mechanisms

1. Scheduling contexts
2. Thread criticalities
3. Temporal exceptions

This talk

1) seL4 concepts
2) Time as a resource
3) Mode switch support
4) Resource sharing

1) seL4 concepts
2) Time as a resource
3) Mode switch support
4) Resource sharing

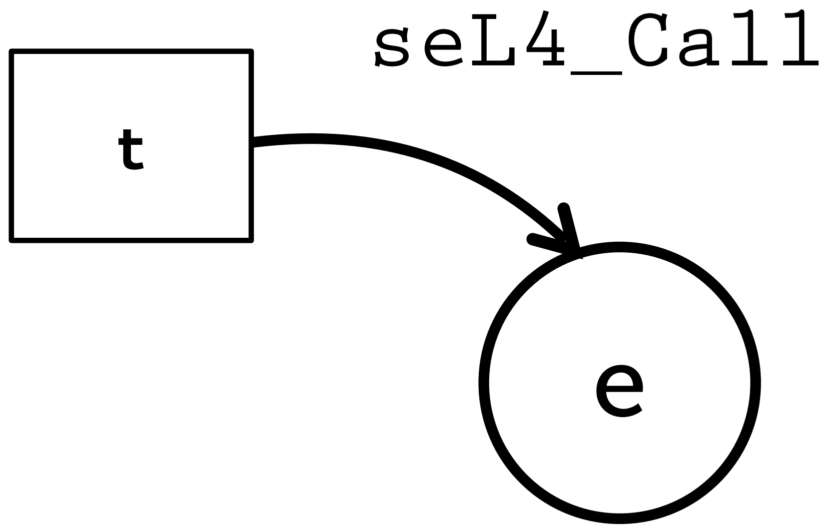
seL4 design principles

- Minimality principle
- Fast
- Possible to verify
  - avoid concurrency
  - avoid unnecessary complexity
  - kernel should not require re-verification if user-level changes

What is a capability?

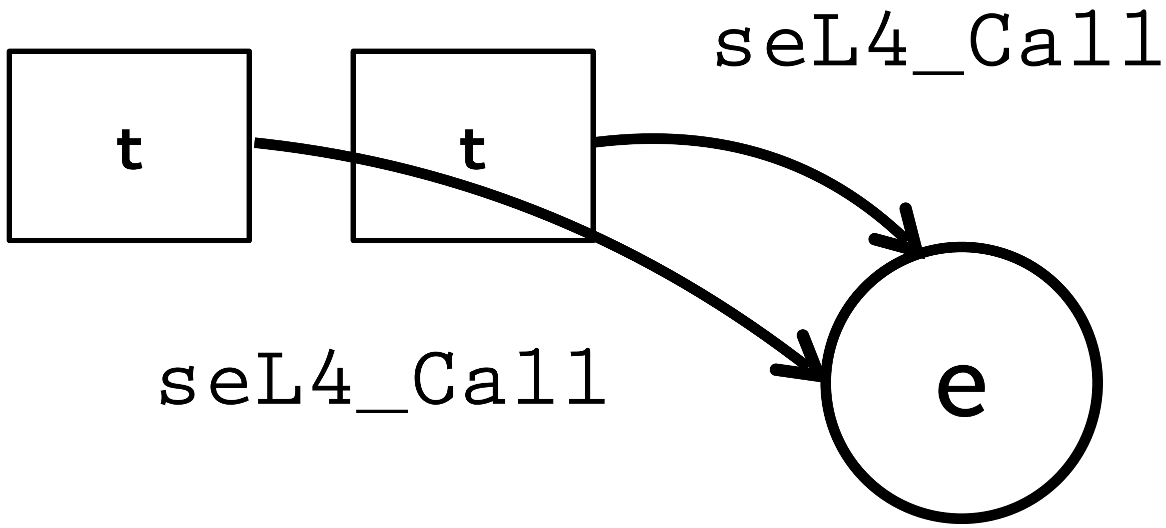
- unforgeable access token
- stored in the **c-space** of an app
  - threads can share c-spaces
- **invoked** by user-level to perform an action
  - no capability, no action
- can be copied, moved between c-spaces

seL4 basics: sync endpoints



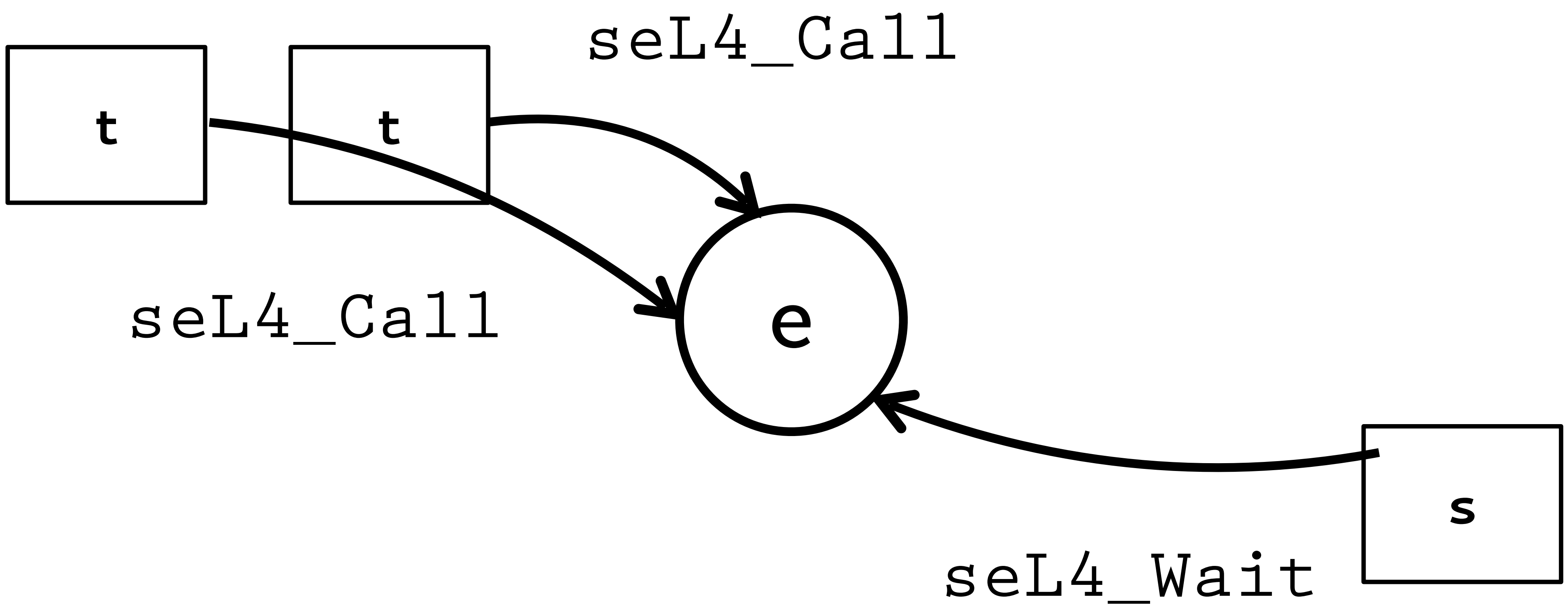
Synchronous endpoints: essentially message ports, which senders/waiters queue on until both are present to receive a message

seL4 basics: sync endpoints



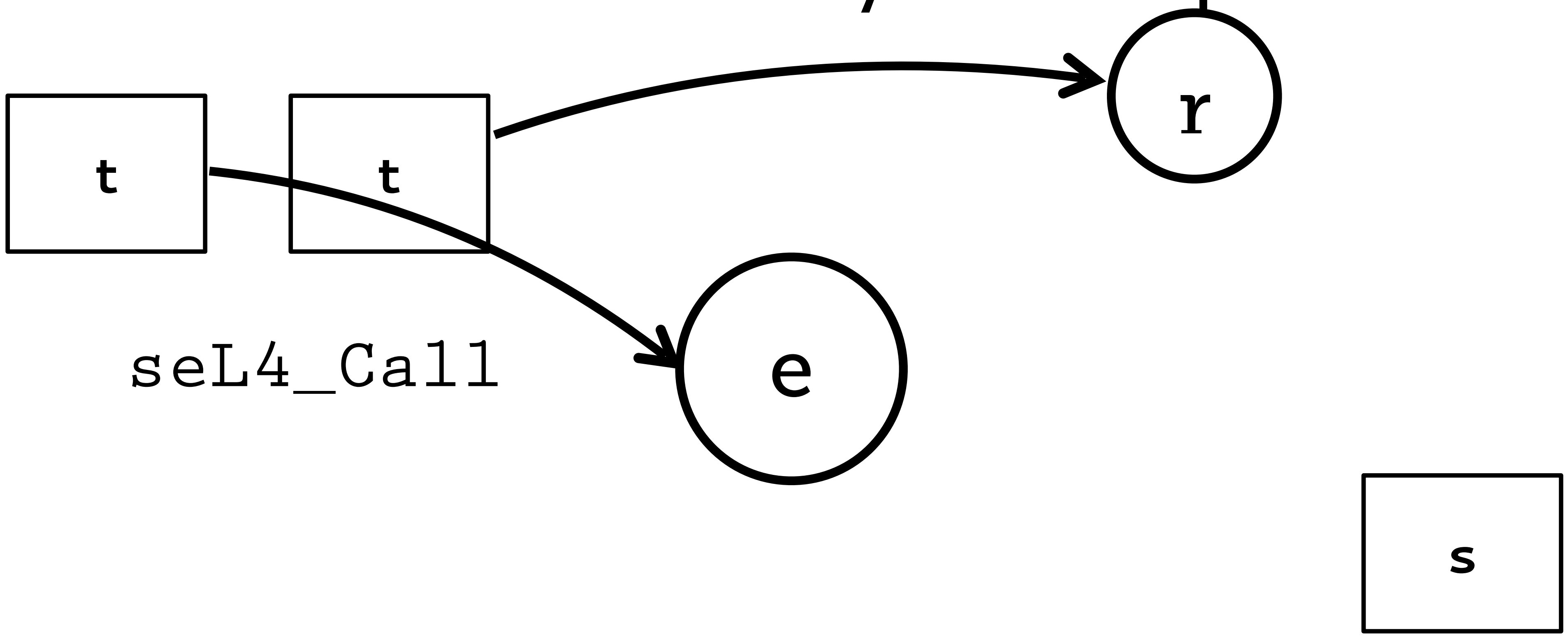
Synchronous endpoints: essentially message ports, which senders/waiters queue on until both are present to receive a message

seL4 basics: sync endpoints



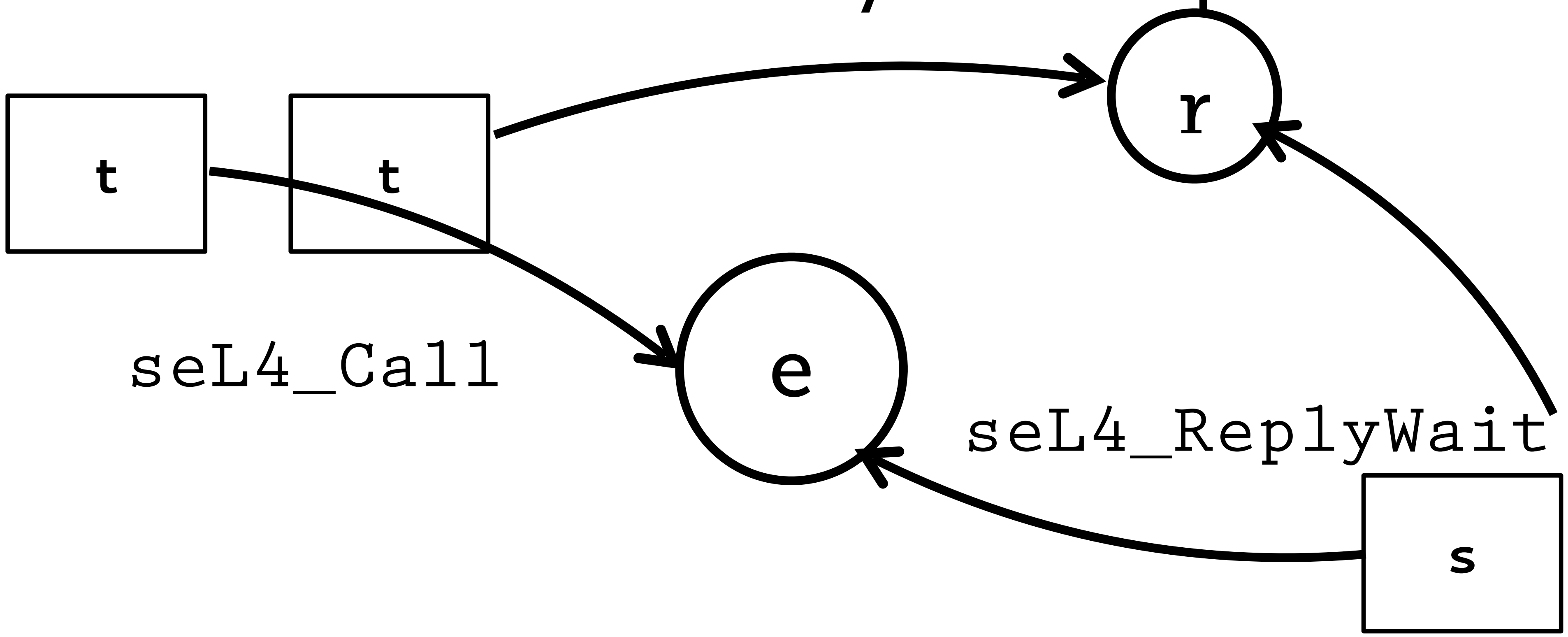
Synchronous endpoints: essentially message ports, which senders/waiters queue on until both are present to receive a message

seL4 basics: sync endpoints



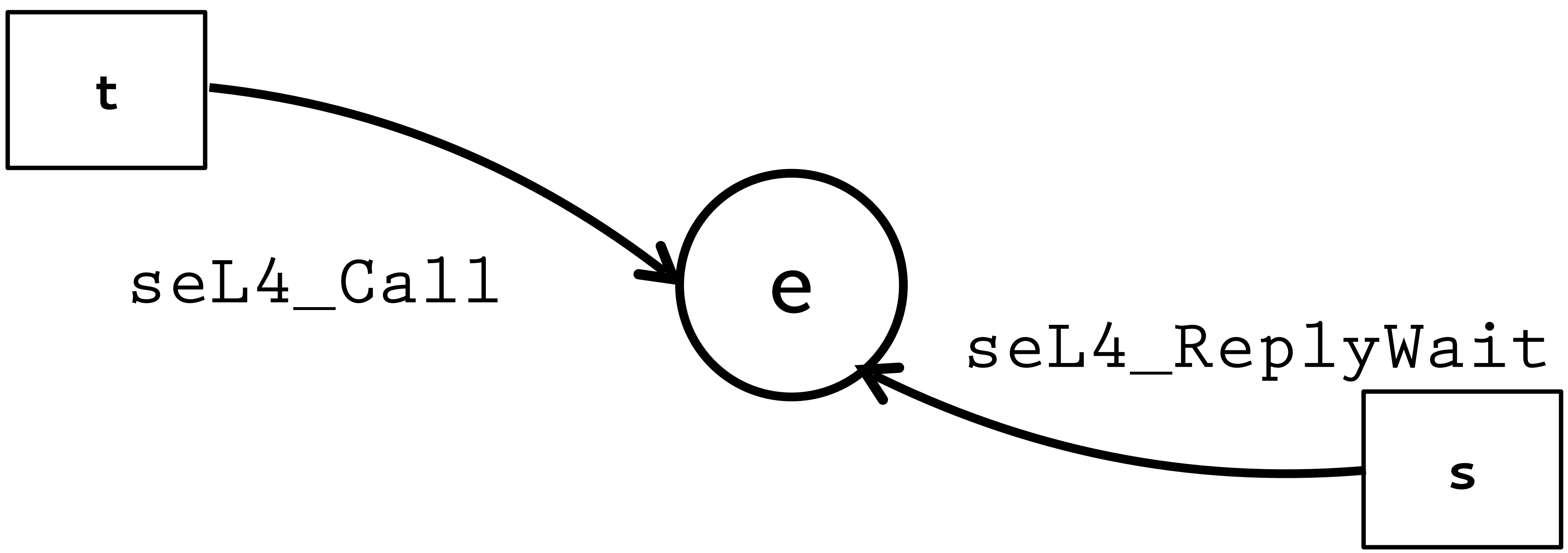
Synchronous endpoints: essentially message ports, which senders/waiters queue on until both are present to receive a message

seL4 basics: sync endpoints



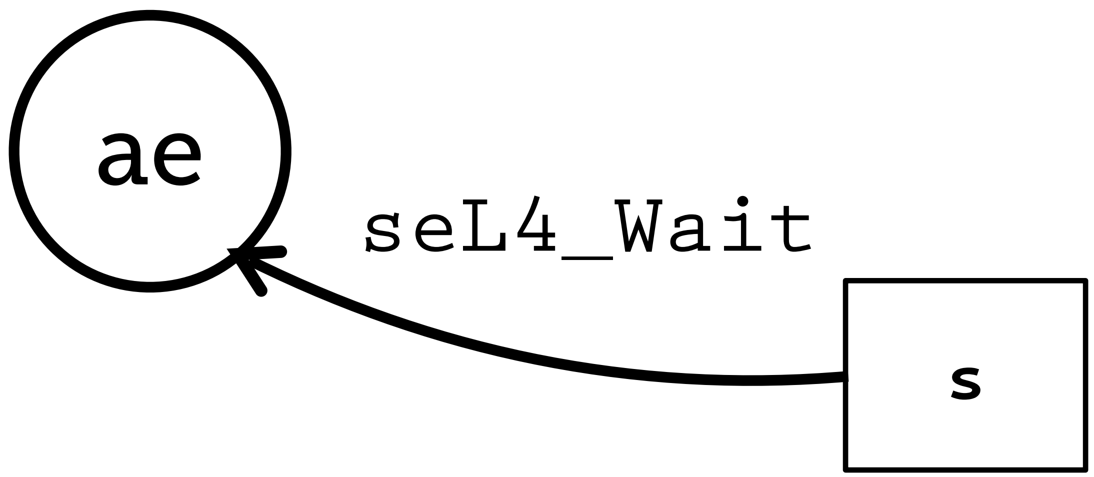
Synchronous endpoints: essentially message ports, which senders/waiters queue on until both are present to receive a message

seL4 basics: sync endpoints



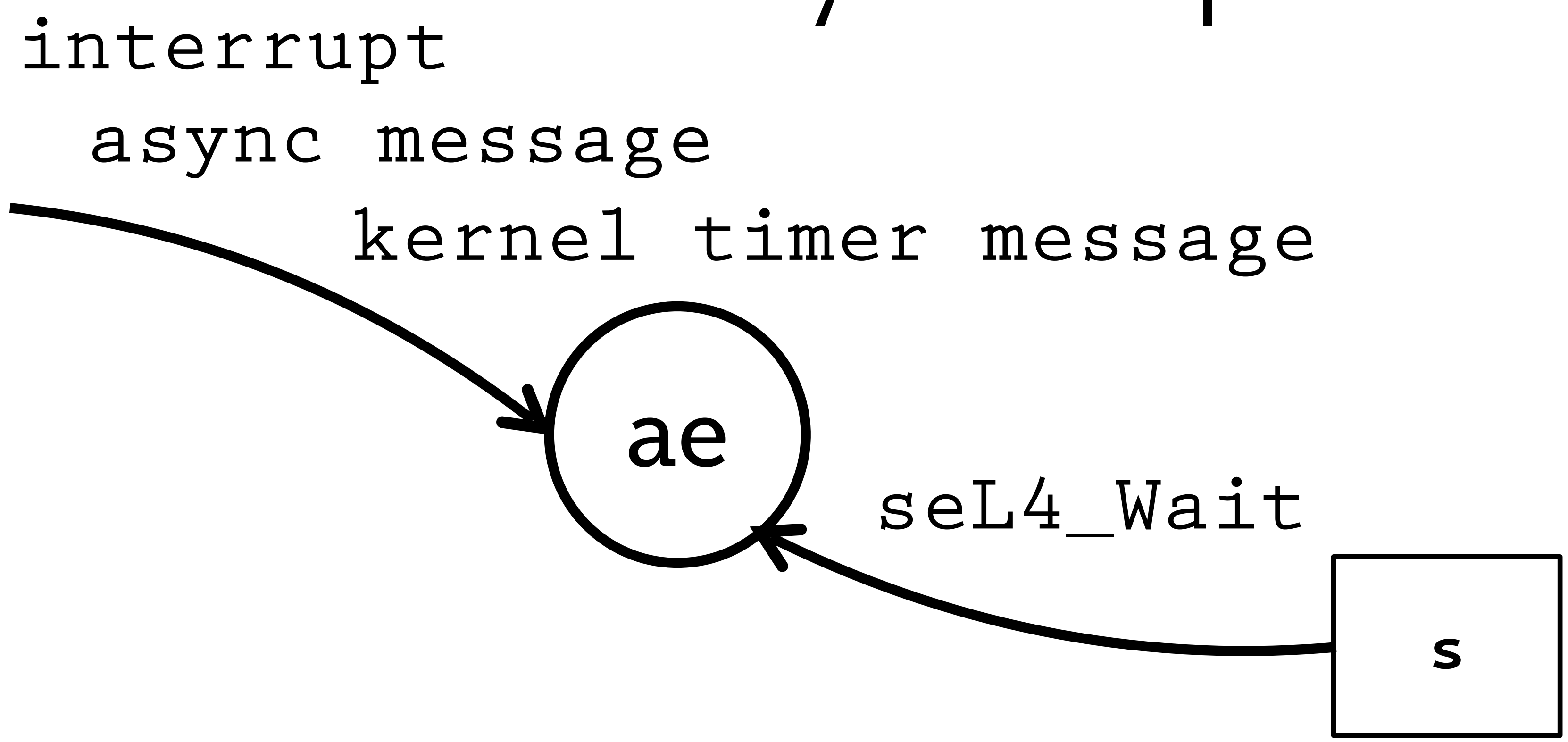
Synchronous endpoints: essentially message ports, which senders/waiters queue on until both are present to receive a message

seL4 basics: async endpoints



Async endpoints (AE): essentially message ports, which accumulate messages until a waiter is present. Waiters queue until a message is present.

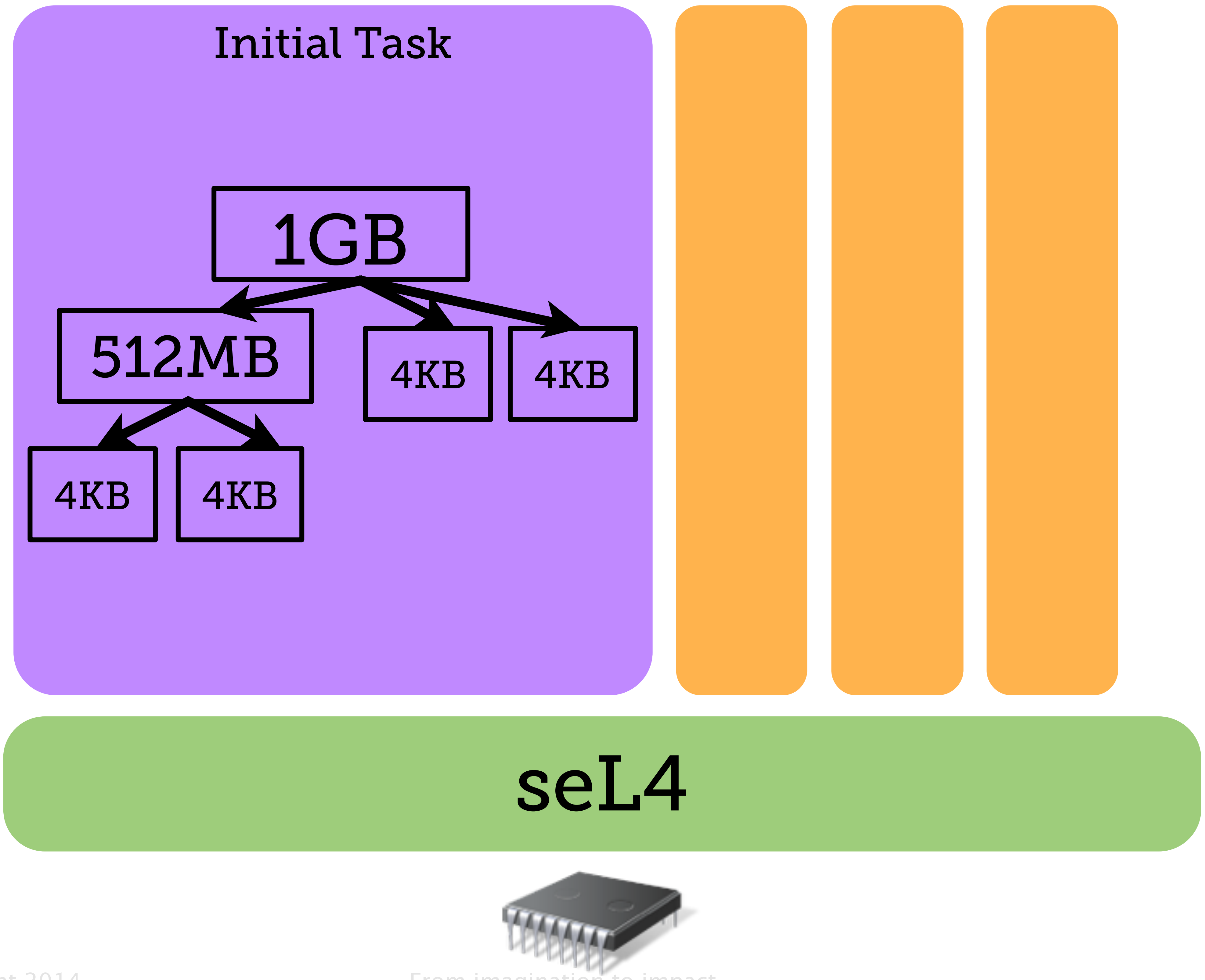
seL4 basics: async endpoints



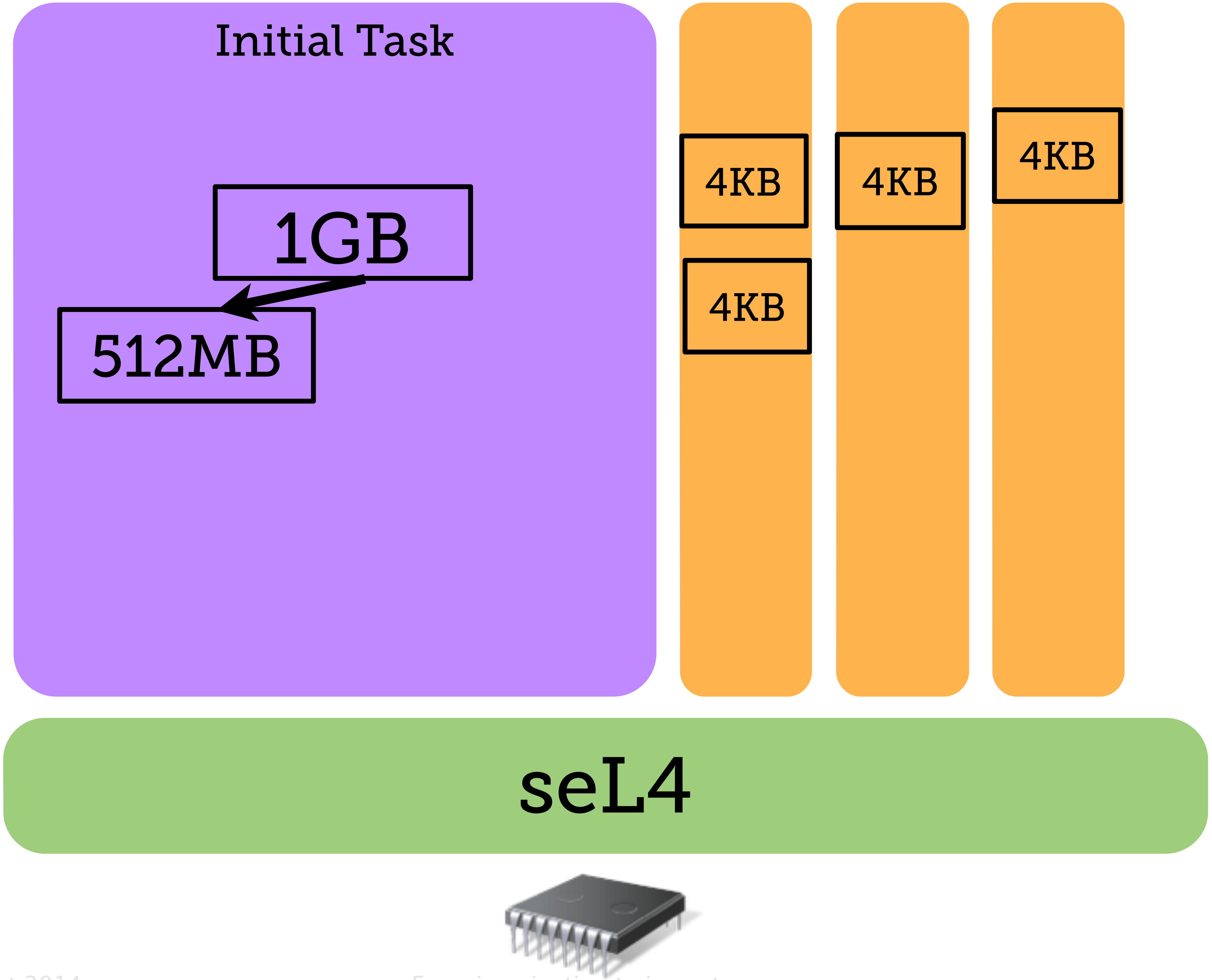
Async endpoints (AEP): essentially message ports, which accumulate messages until a waiter is present. Waiters queue until a message is present.

A **bound async endpoint** has a special 1:1 relationship with a thread – and only the bound thread is allowed to wait a bound AEP

seL4 Memory Model



seL4 Memory Model



Meet seL4: Summary

- capability based
- communication via endpoints
  - synchronous or asynchronous
- all resources managed at user-level
- initial task gets capabilities to everything in the system

1) seL4 concepts

2) Time as a resource

3) Mode switch support

4) Resource sharing

Resource kernels*

- Timeliness of resource access
  – reservations
- Efficient resource utilisation
- Enforcement & Protection
- ~~Access to multiple resource types~~

* [Rajkumar et al. 2001]

Resource kernel mechanisms

- Admission
- Scheduling
- Enforcement
- Accounting

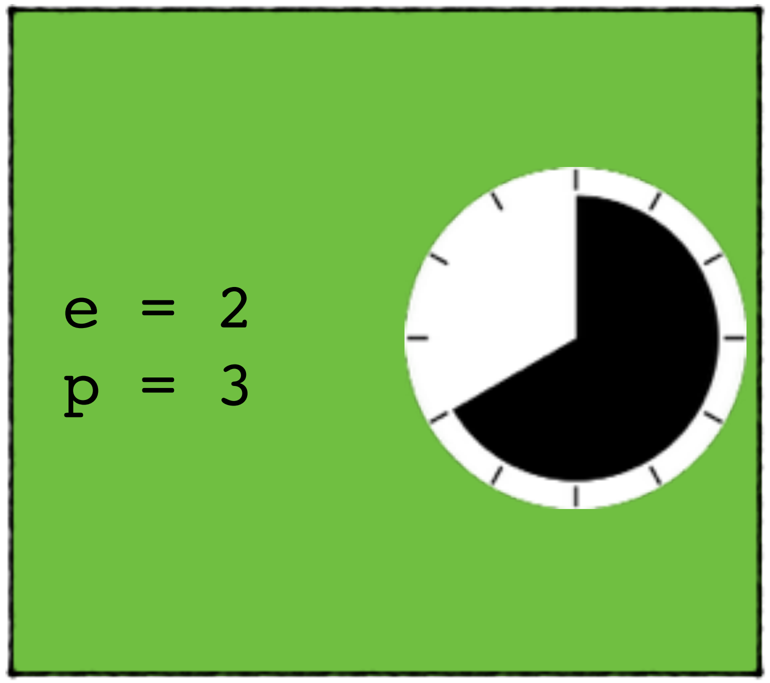
Which mechanisms belong in a microkernel?

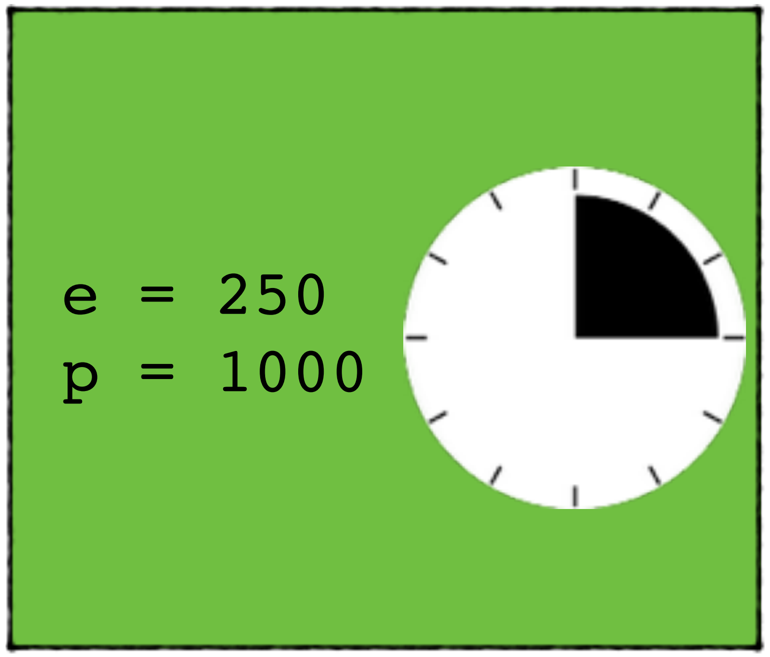
Resource kernel mechanisms

- ~~Admission~~ (policy)
- Scheduling
- Enforcement
- Accounting

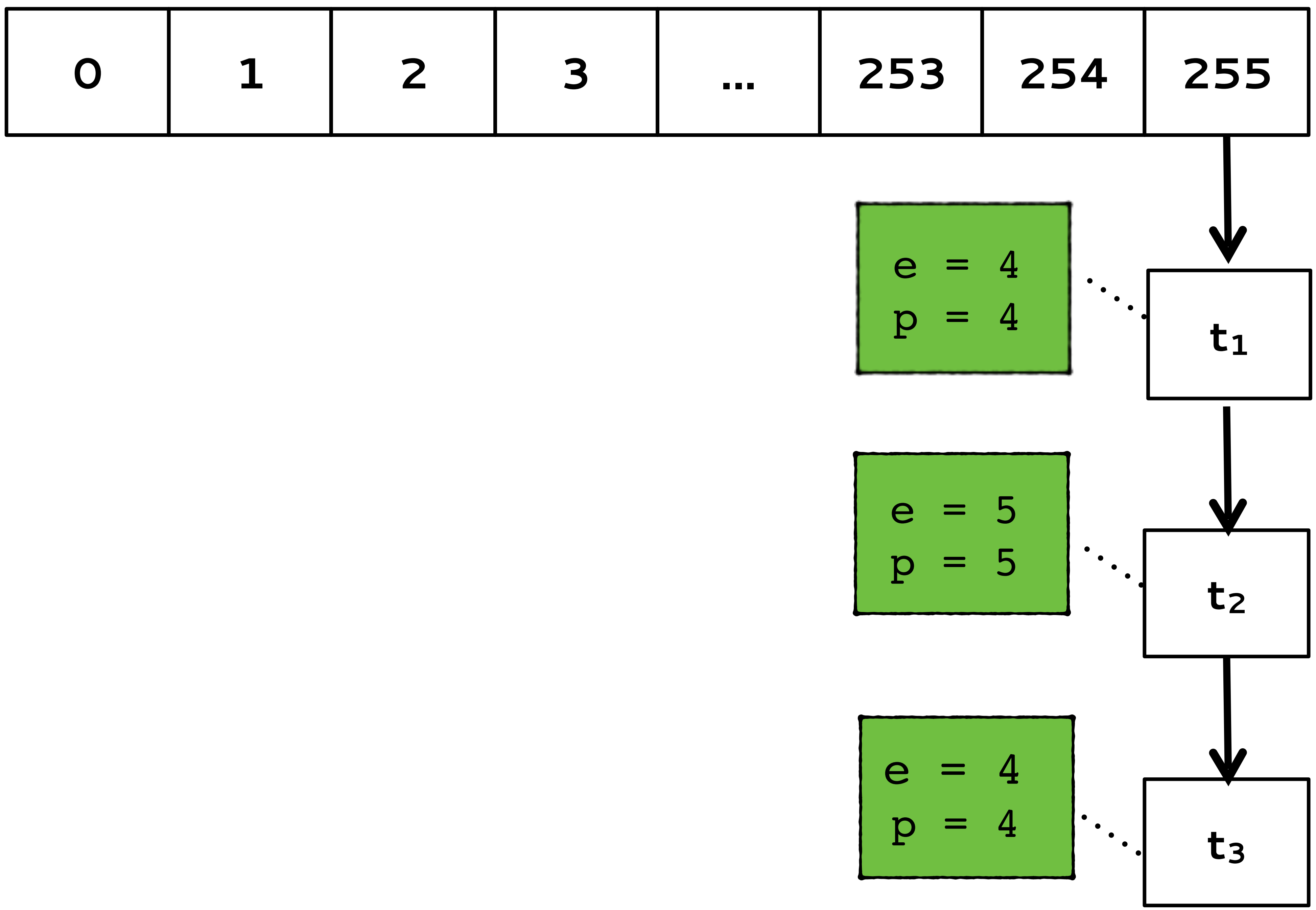
Scheduling Contexts

- Implements processor "reservation"
- adapted from Fiasco [Steinberg 2010]
- Upper bound
- No priority
- Rate = e / p
- **Full** or **Partial**
- Only 1 per thread

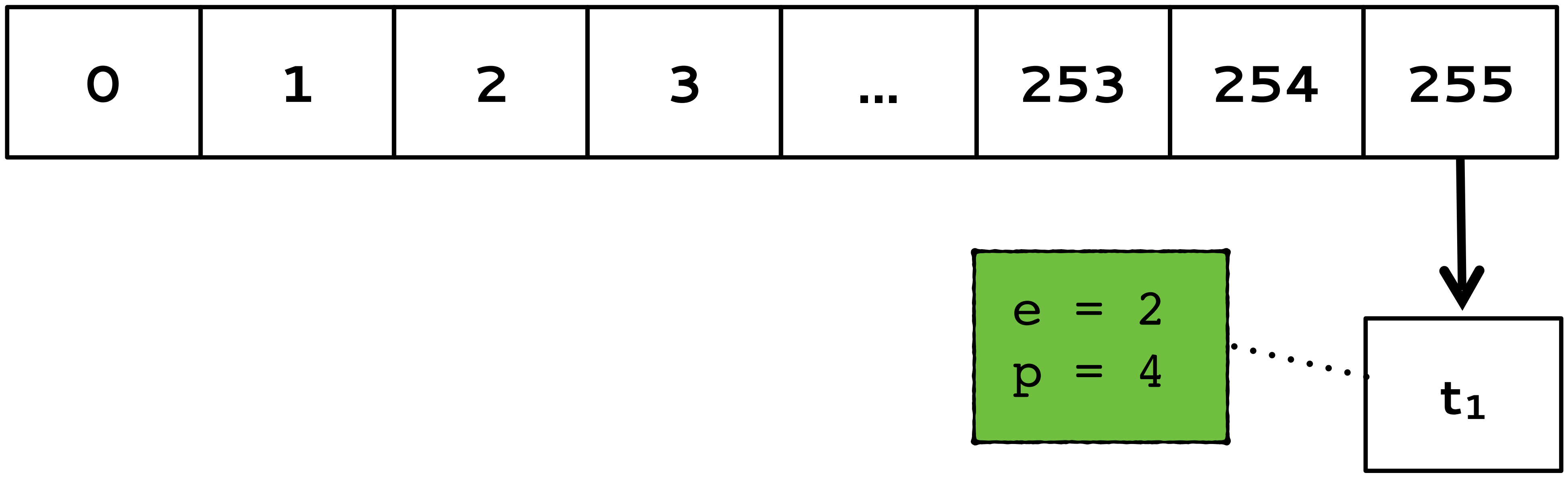




Full reservations

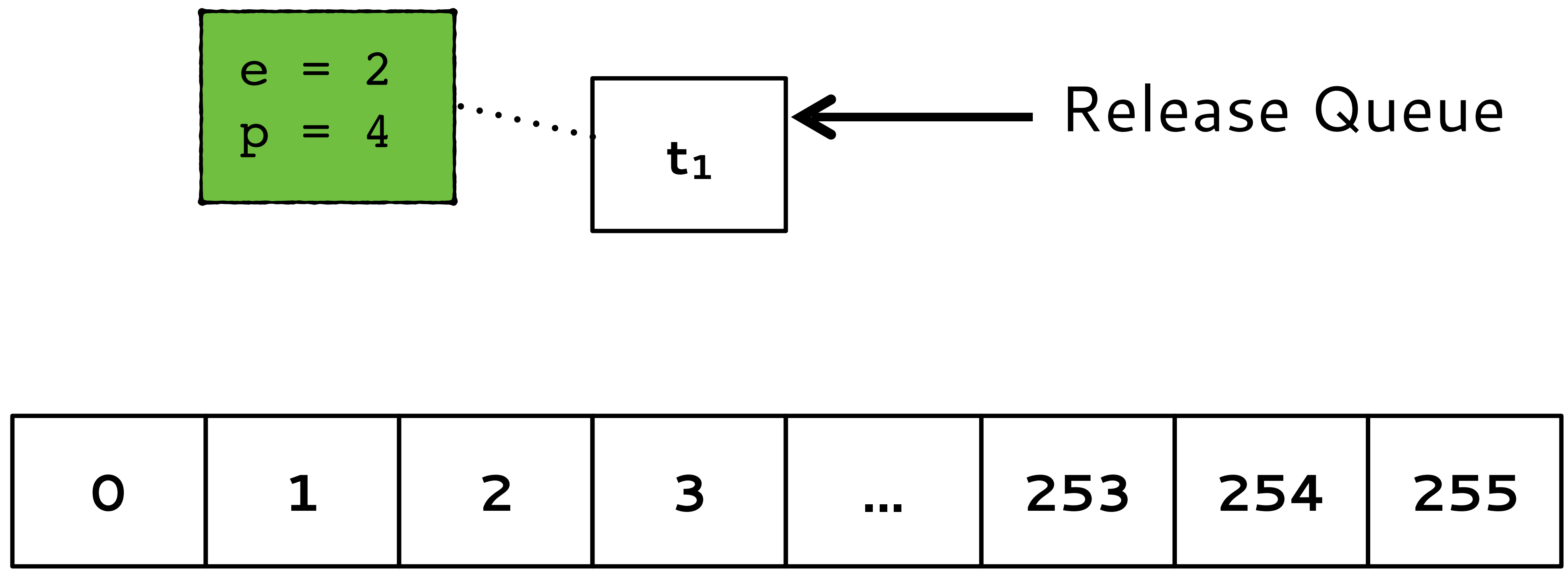


Partial reservations



Scheduling contexts
act as sporadic servers

Partial reservations

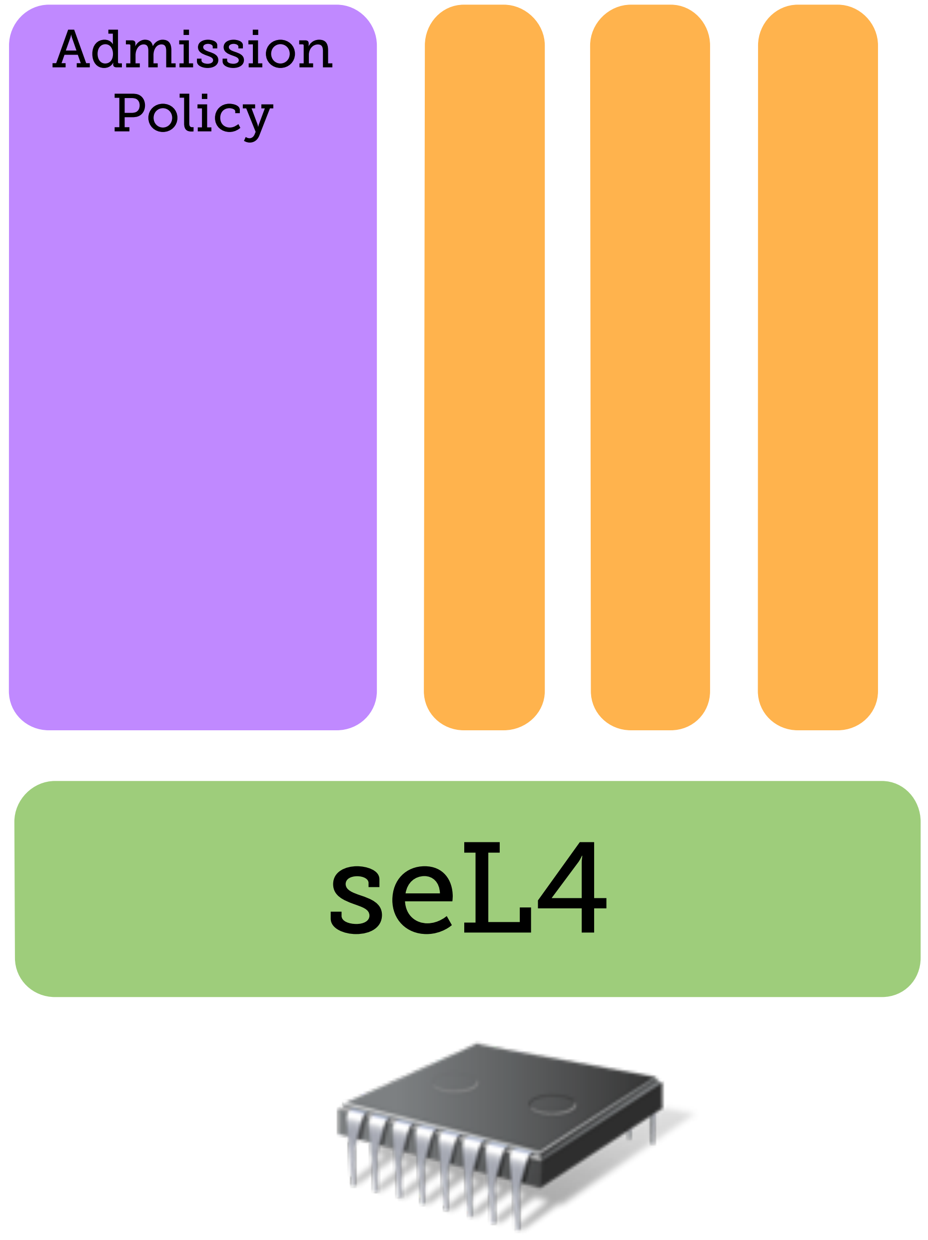


Scheduling contexts

act as sporadic servers

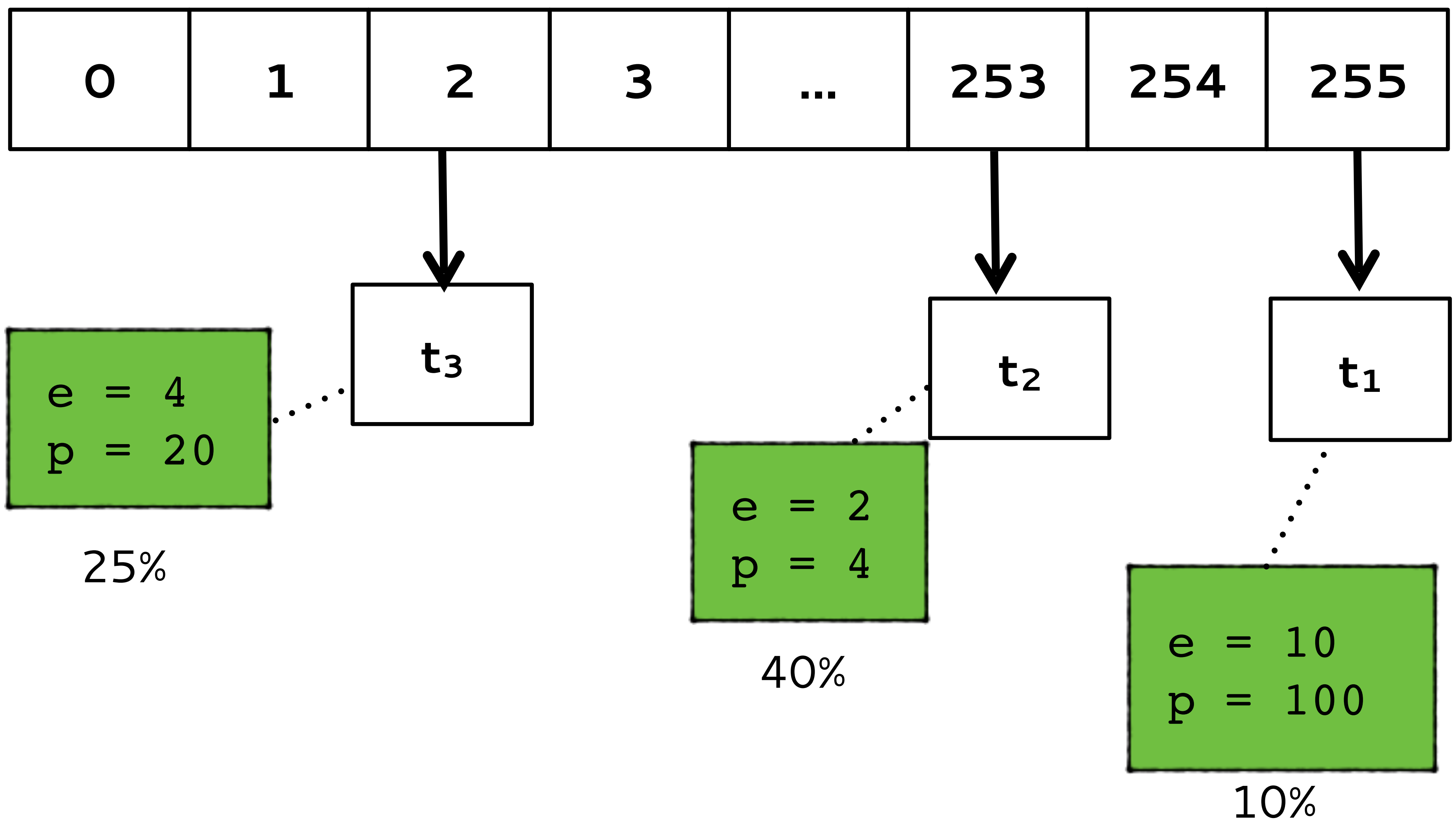
Admission

- New **control** capability, seL4_SchedControl.
- Controls population of scheduling context parameters.
- Must take into account priorities



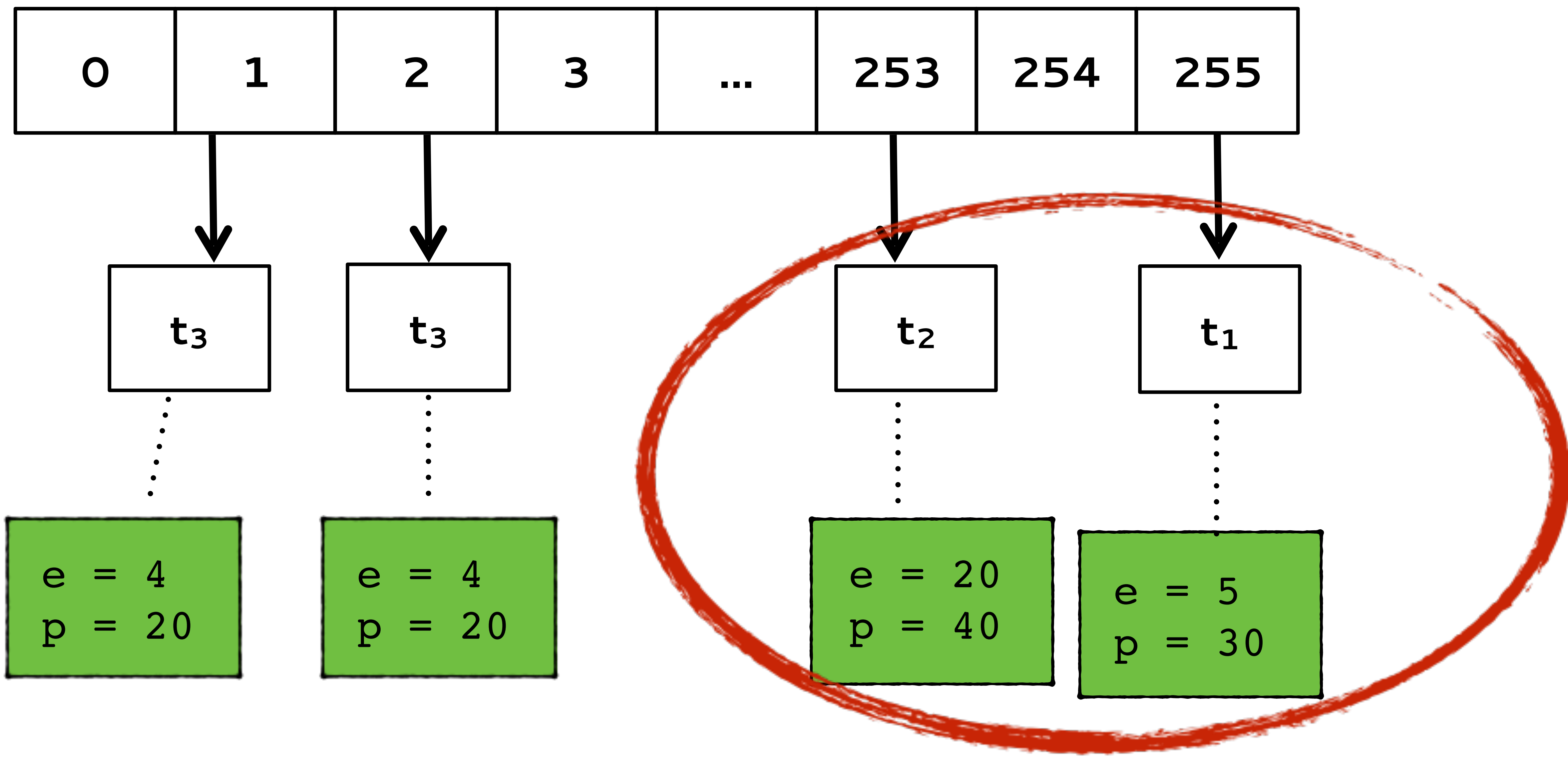
Scheduling

Basic Rate Monotonic



Scheduling

Low priority tasks in slack



Time as a resource: summary

- scheduling contexts
  - full or partial
  - act as upper bounds
  - disjoint from priority
- user-level admission
  - allows for mixed RT/RR scheduling
  - not full flexibility of user-level scheduling

This talk

1) seL4 concepts
2) Time as a resource
3) Mode switch support
4) Resource sharing

Task model

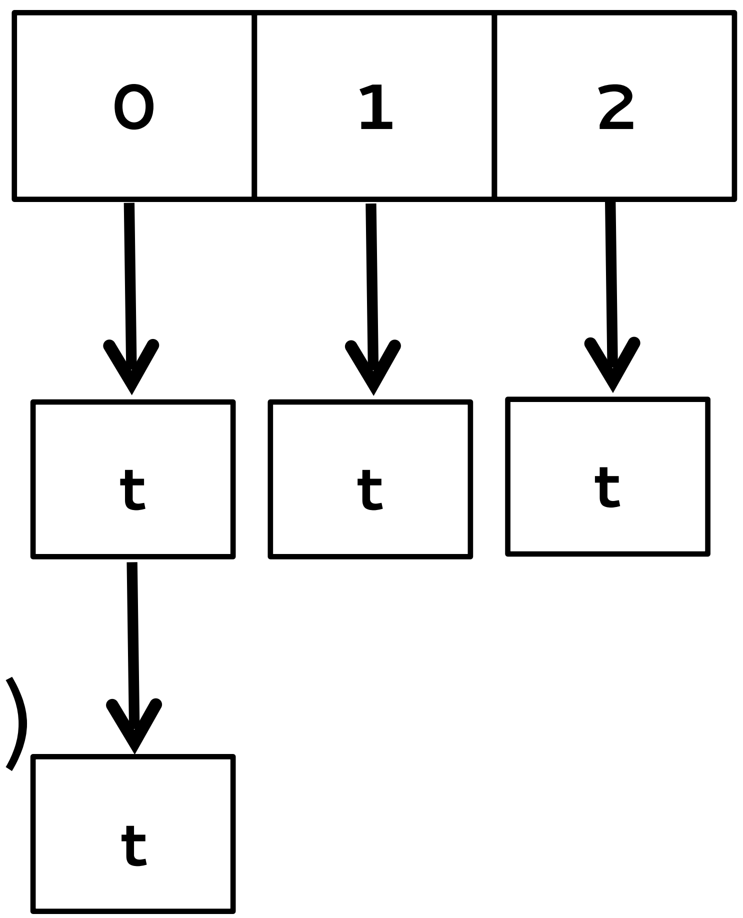
```
while (1) {
   /* job release */
   doJob();
   /* job completion */
  seL4_Wait(bep);
}
```

If job completion does not occur before the budget expires, send a **temporal exception** or rate-limit.

Bound async endpoint where device interrupts, async messages or kernel timer trigger job release

Criticality

- New thread field
- Range set at compile time
- seL4_SetCriticality
  - invokes sched_control cap
  - HI -> LO is lazy
  - LO -> HI is immediate, and O(n)



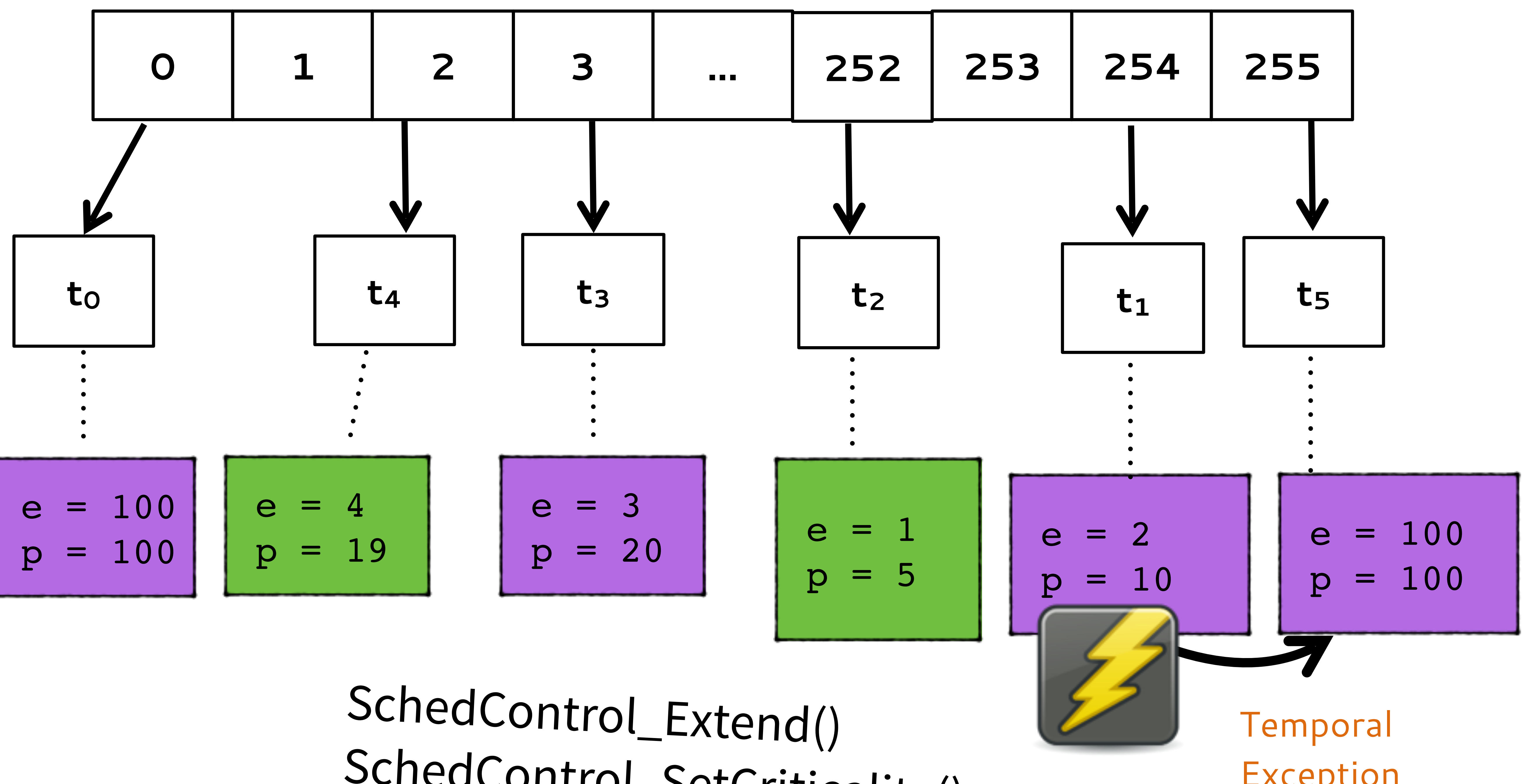
Criticality mode change

- Assumptions:
  - infrequent (if they occur at all)
  - short in duration

- Kernel provides ability to
  - change params of excepting thread
  - postpone all lower criticality threads
  - alter priorities of threads

Asymmetric Protection

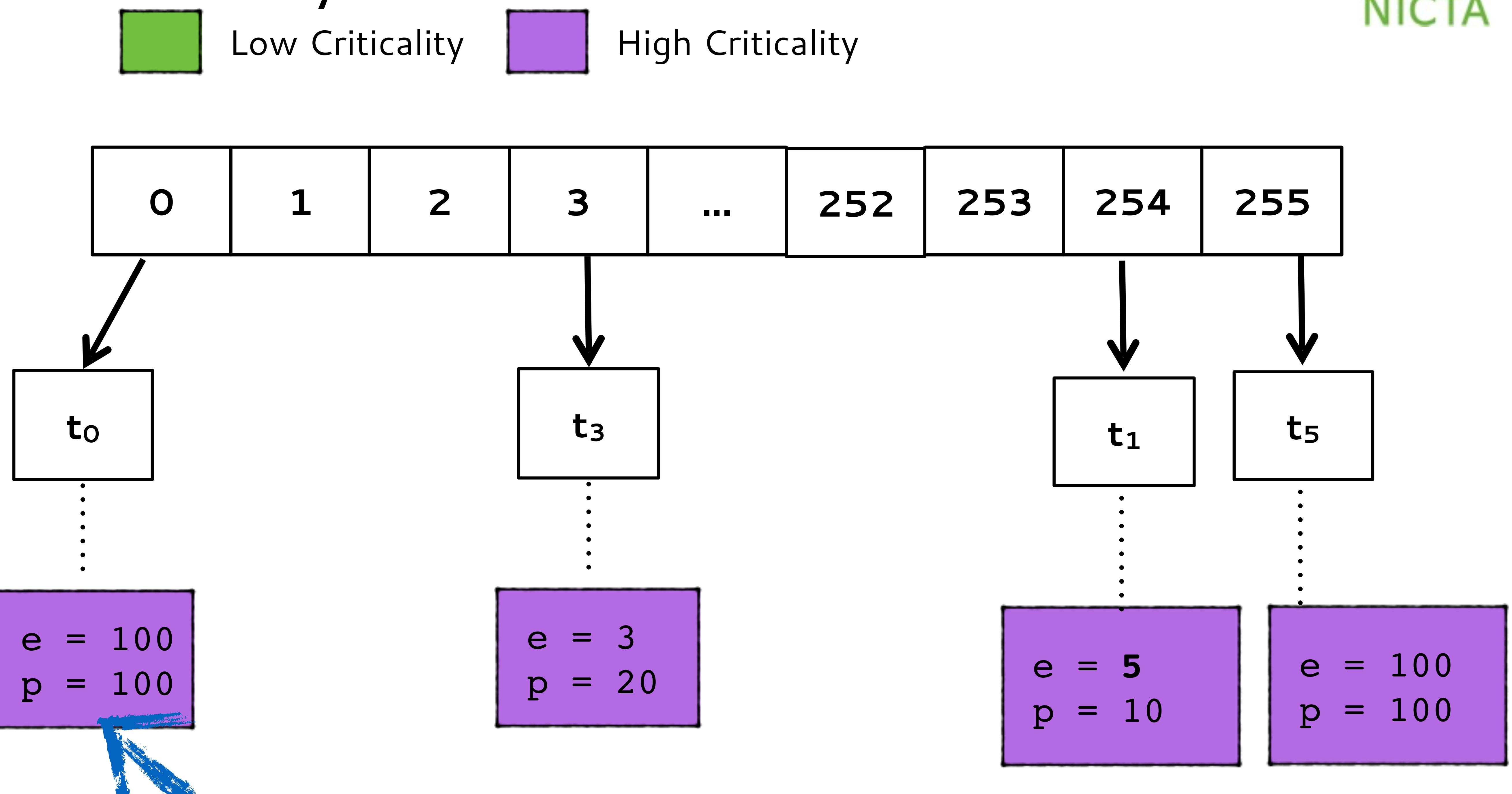
Low Criticality   High Criticality



SchedControl_Extend()

SchedControl_SetCriticality()

Asymmetric Protection



Restores criticality when system is idle

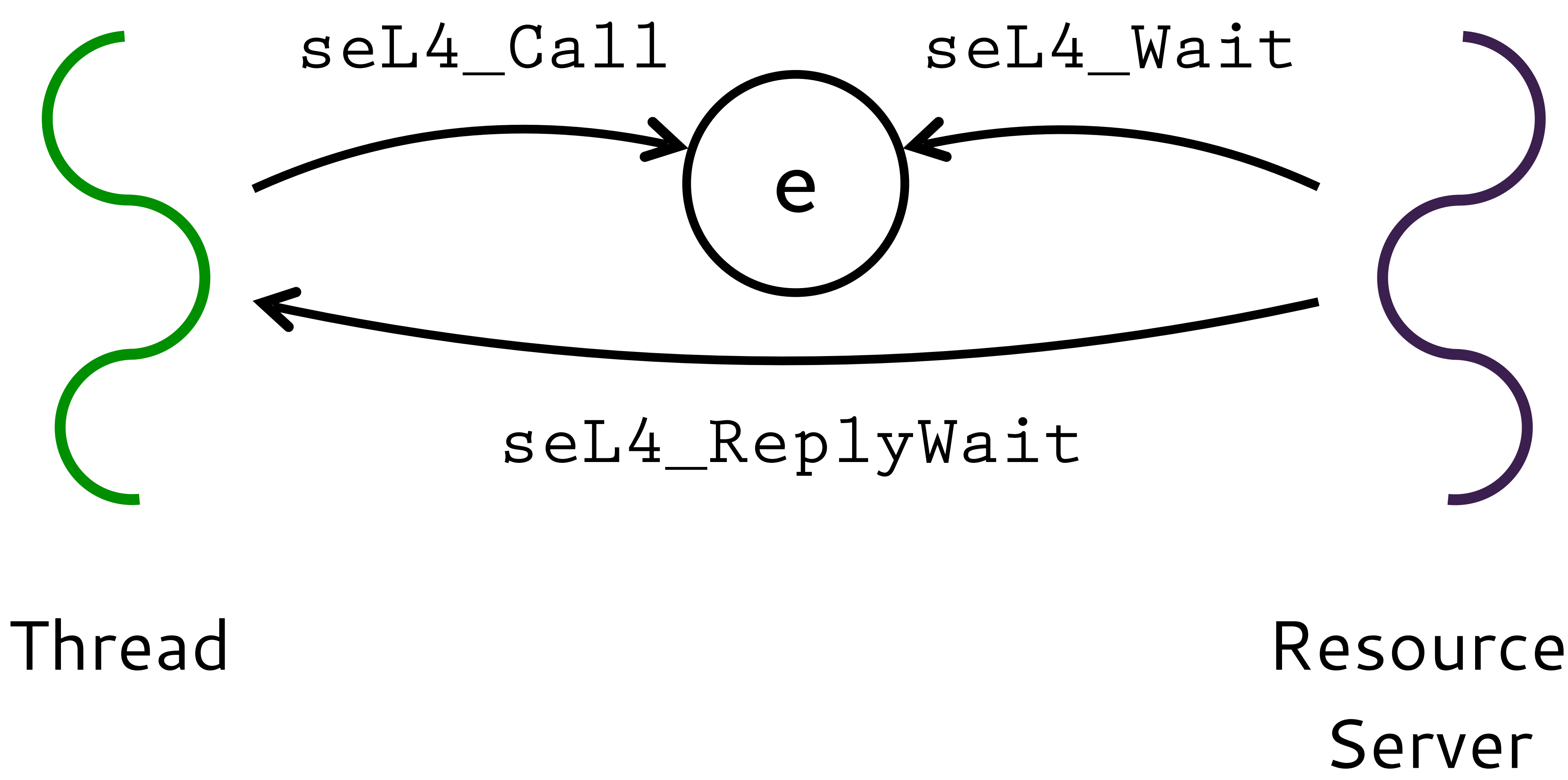
Criticality: Summary

- Temporal exceptions
  - optional (not required for rate-based threads)
  - handler must have own budget
- New thread field: criticality
- New kernel invocation: set criticality
  - although temporal exception handler can take other actions

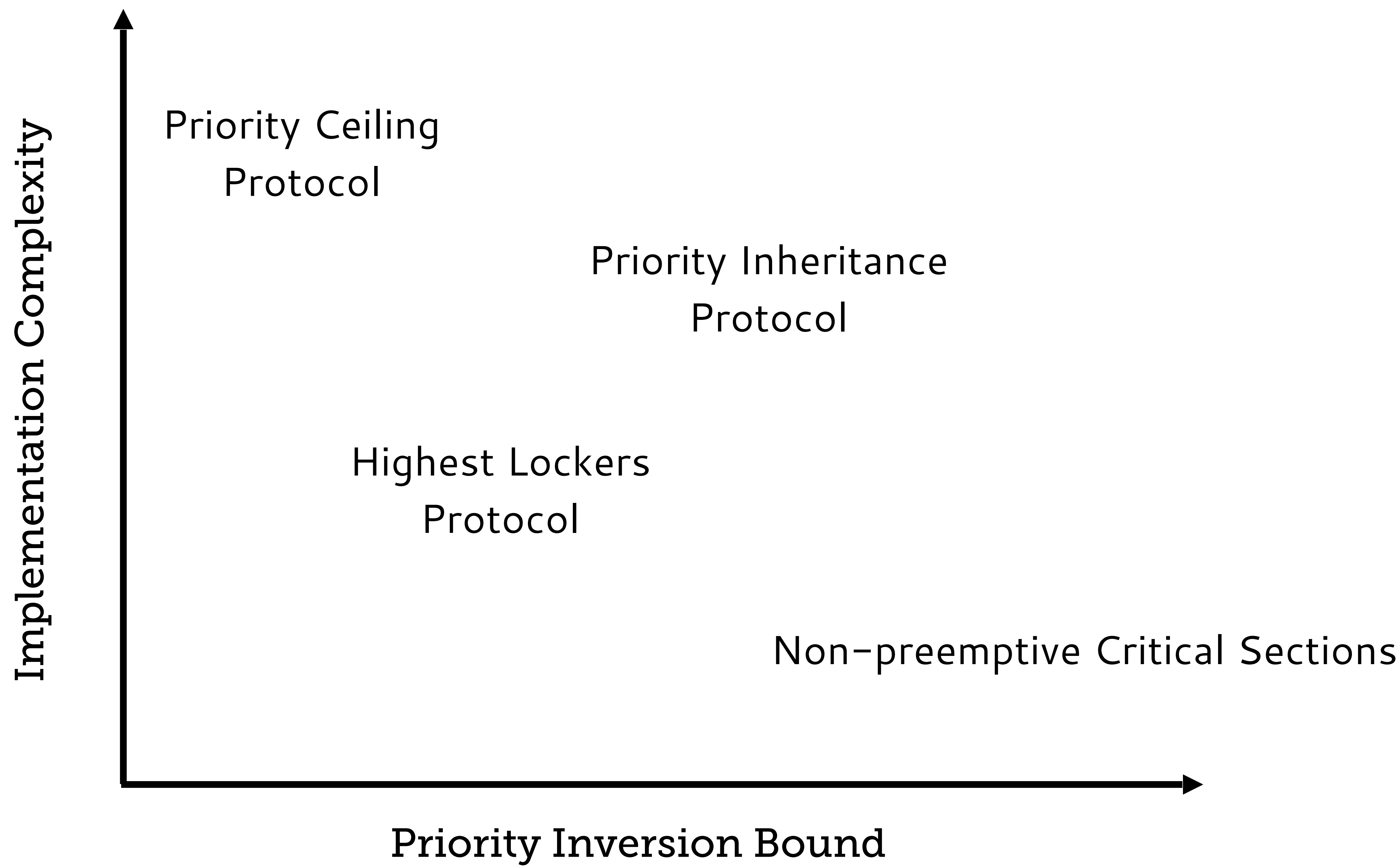
This talk

1) seL4 concepts
2) Time as a resource
3) Mode switch support
4) Resource sharing

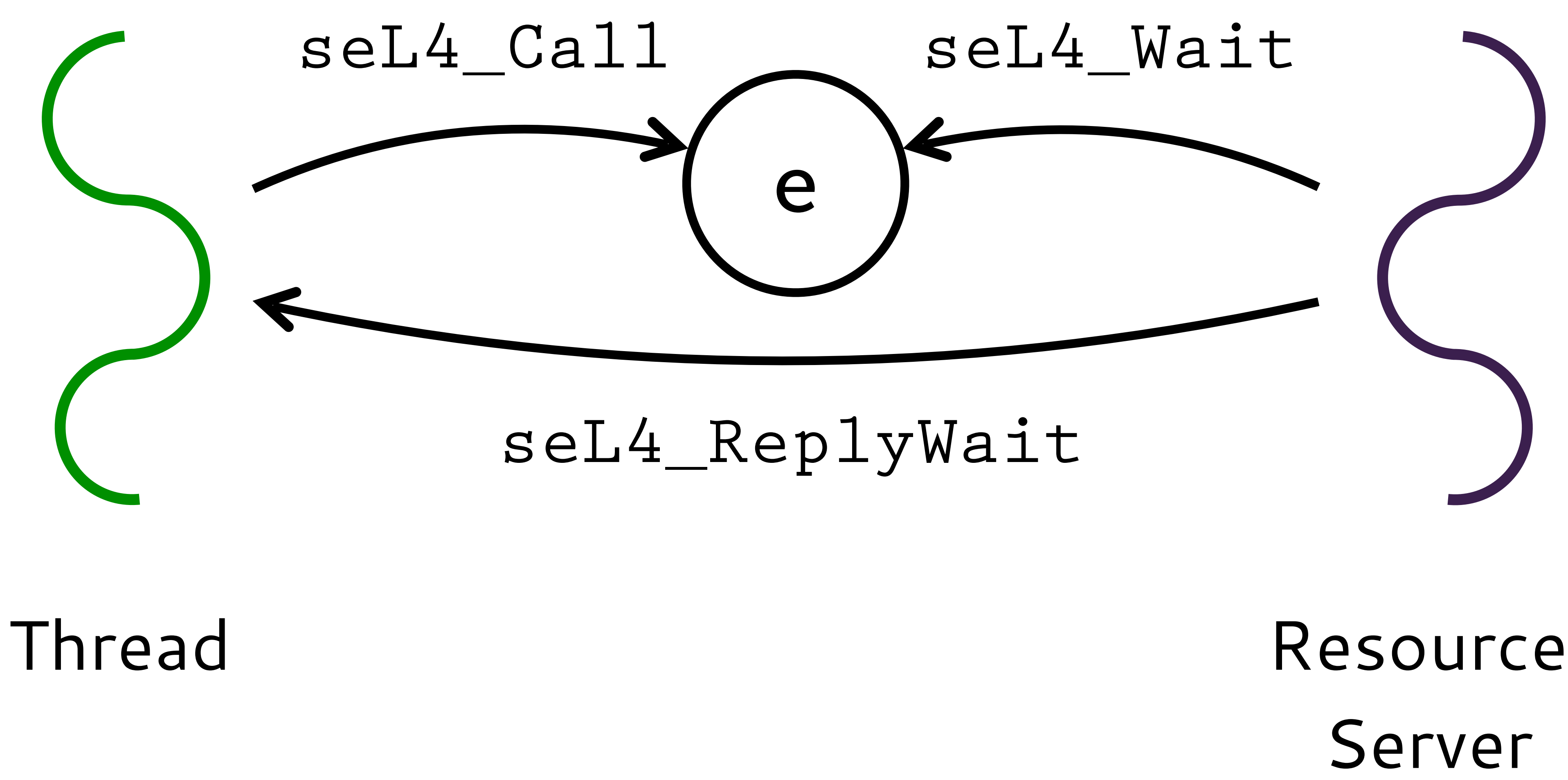
Resource Sharing



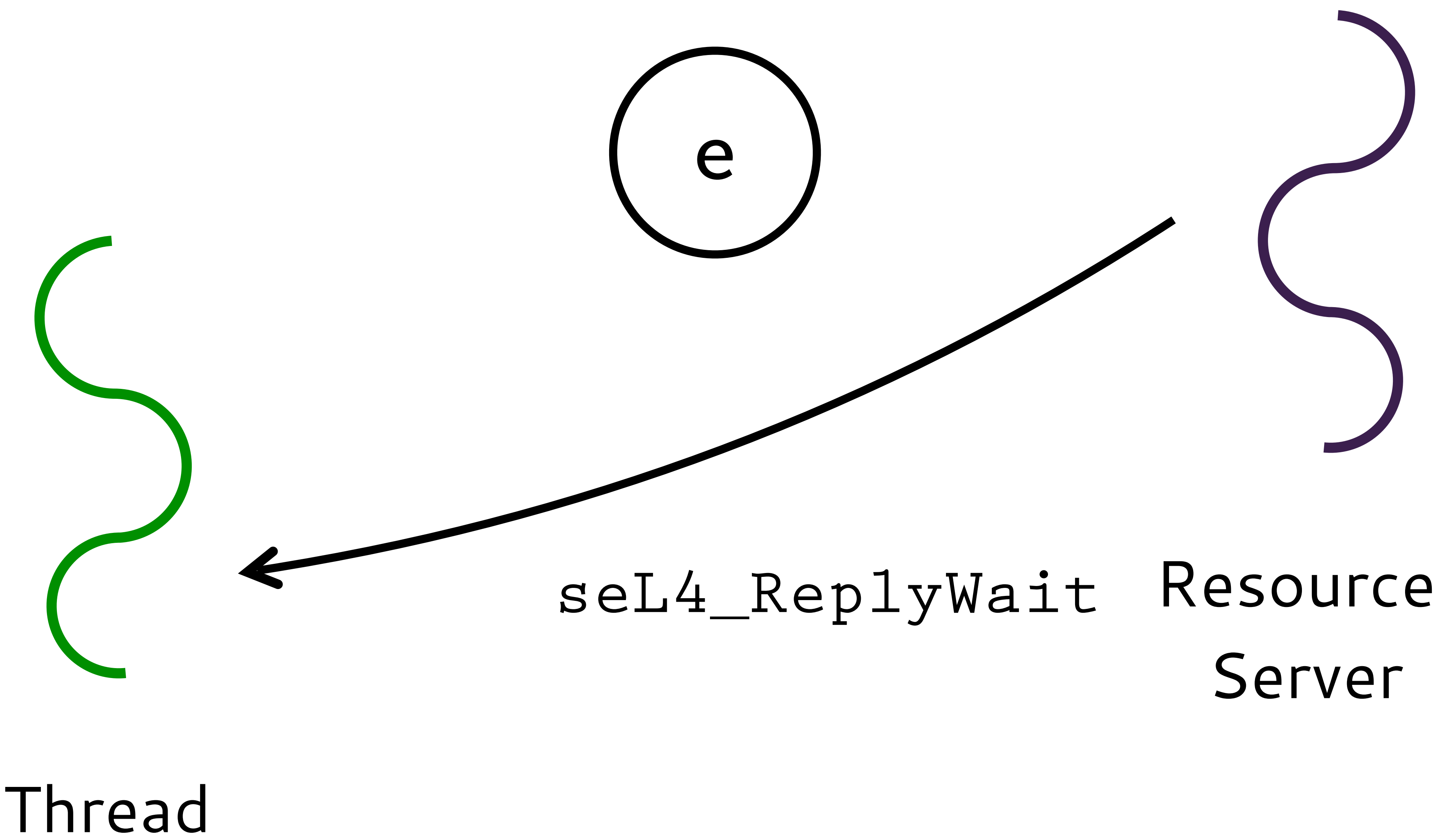
NCP vs. PIP vs HLP vs PCP



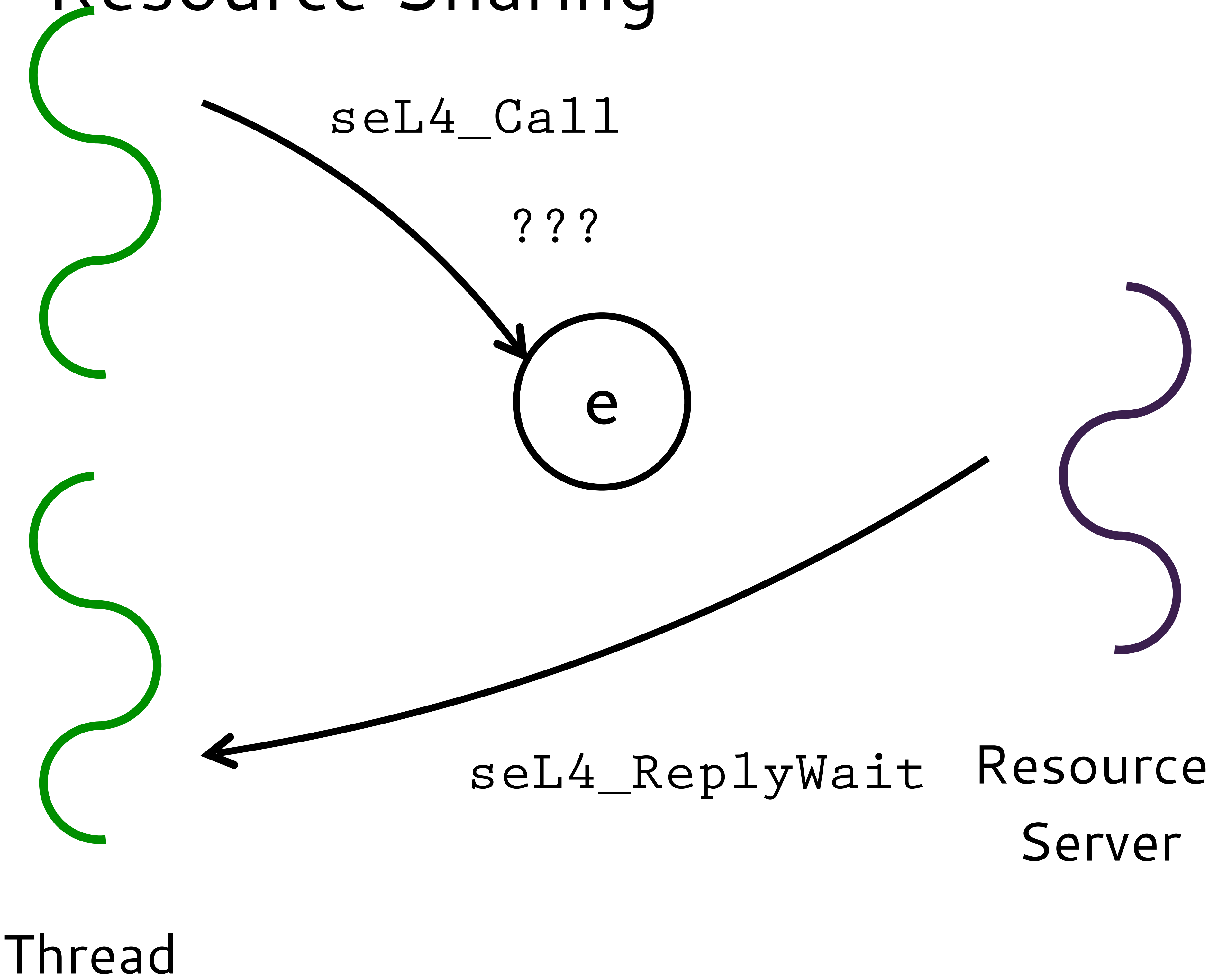
Resource Sharing



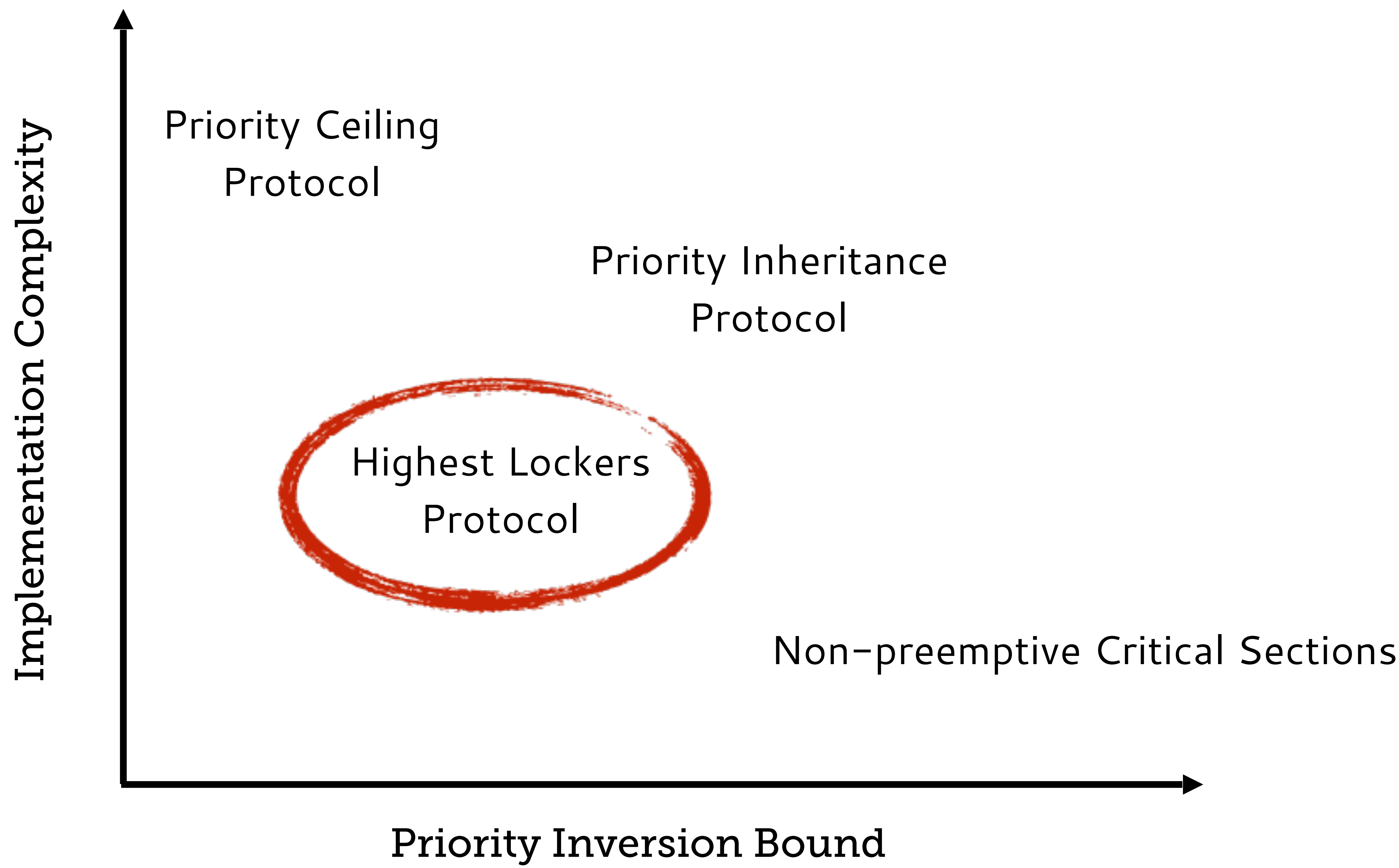
Resource Sharing



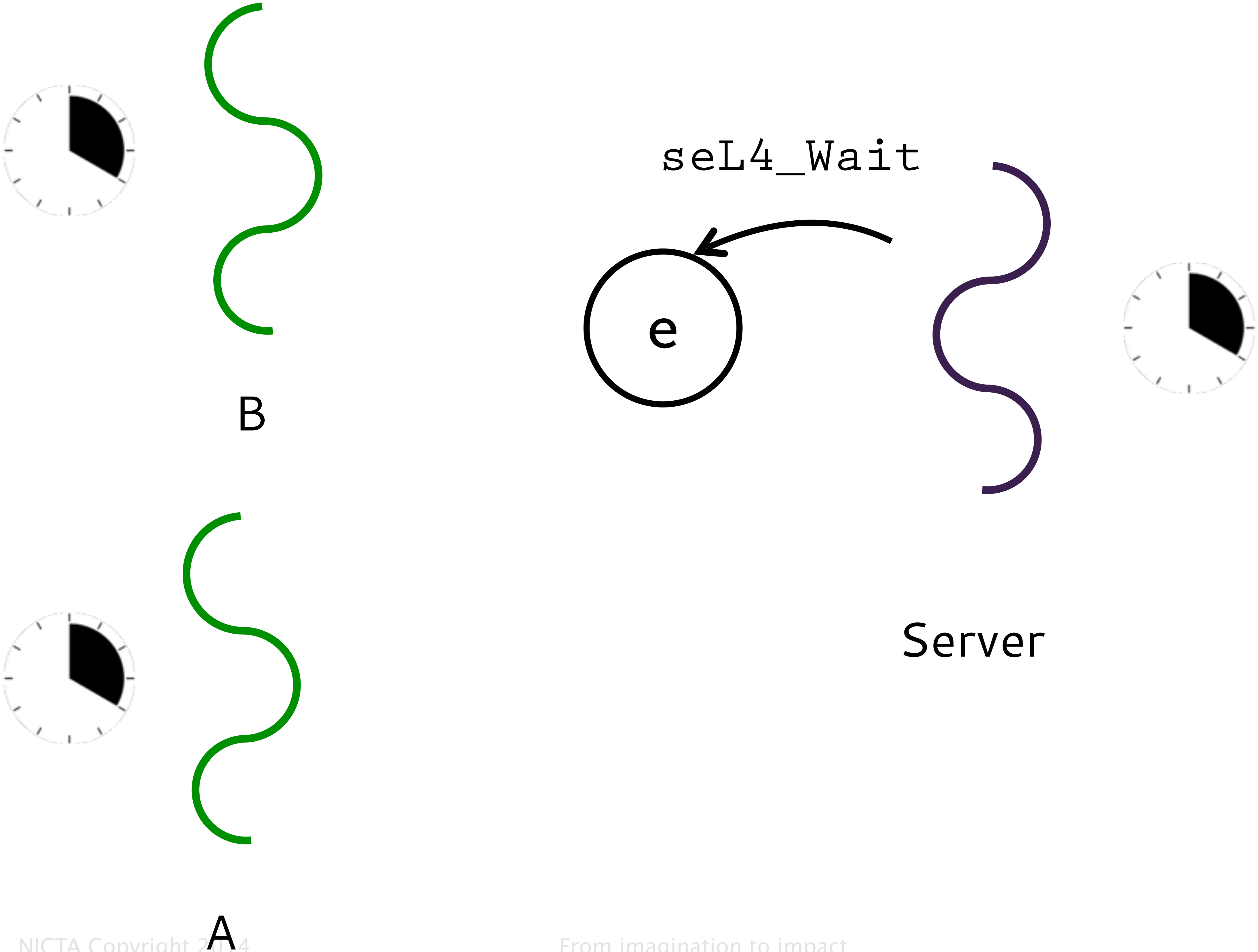
Resource Sharing



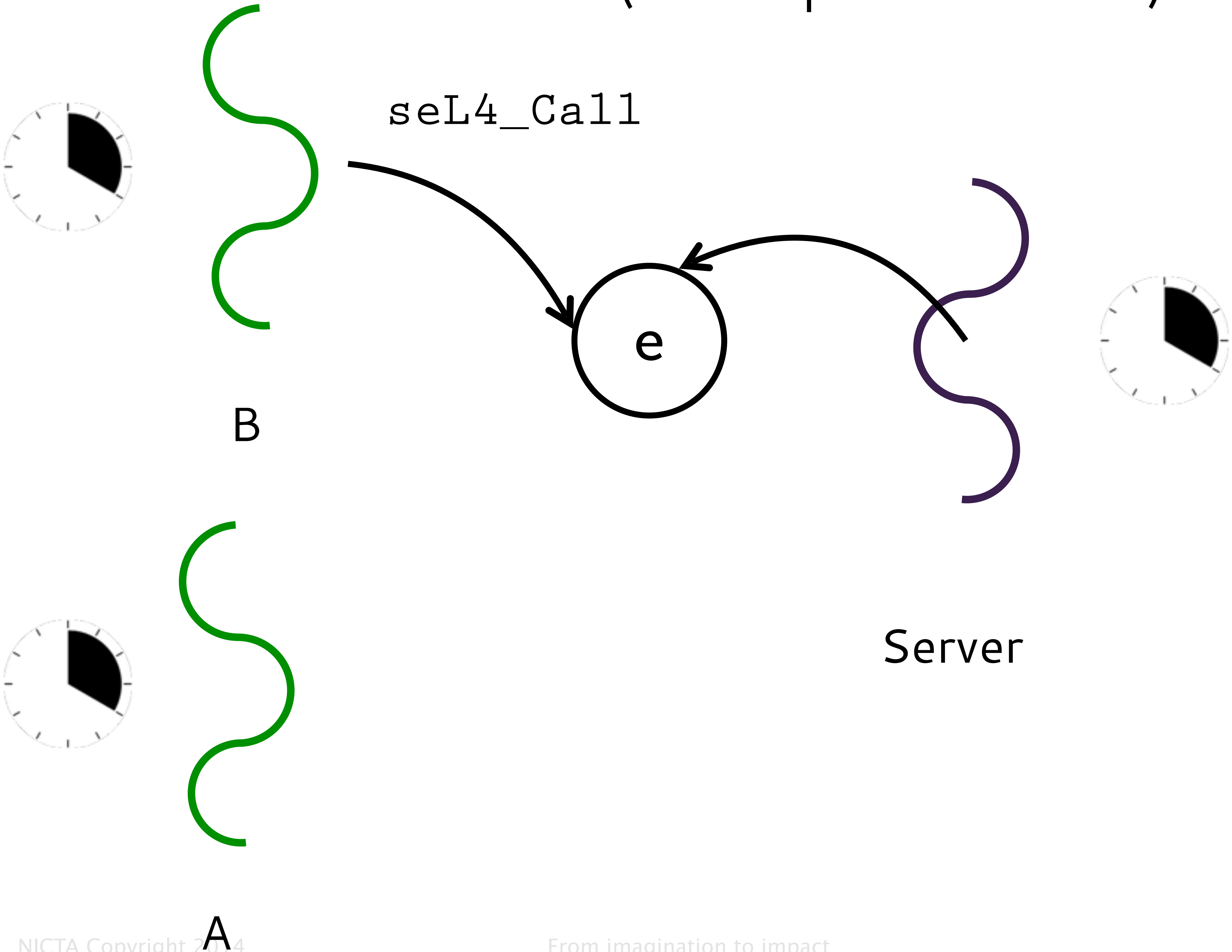
NCP vs. PIP vs HLP vs PCP



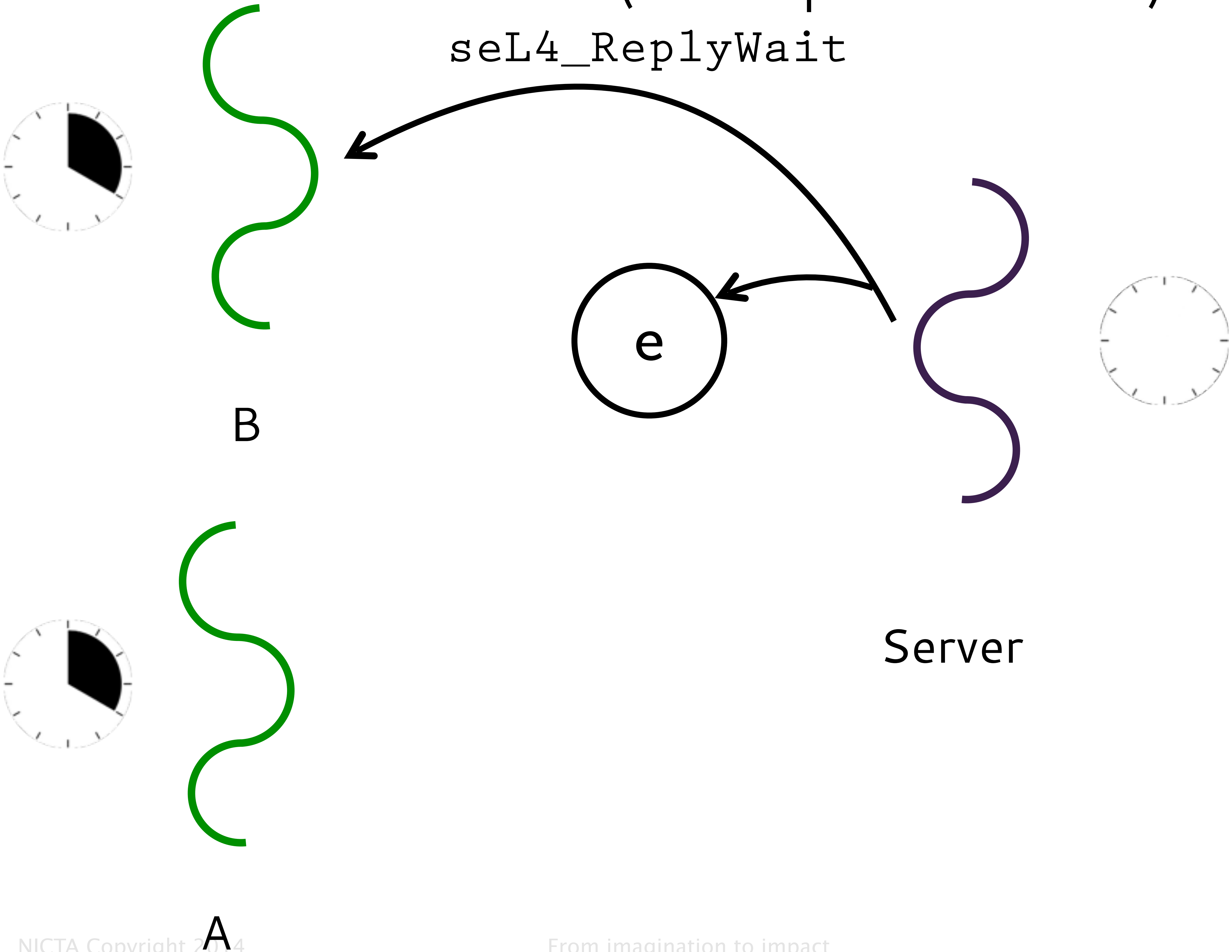
Active Servers (no temporal isolation)



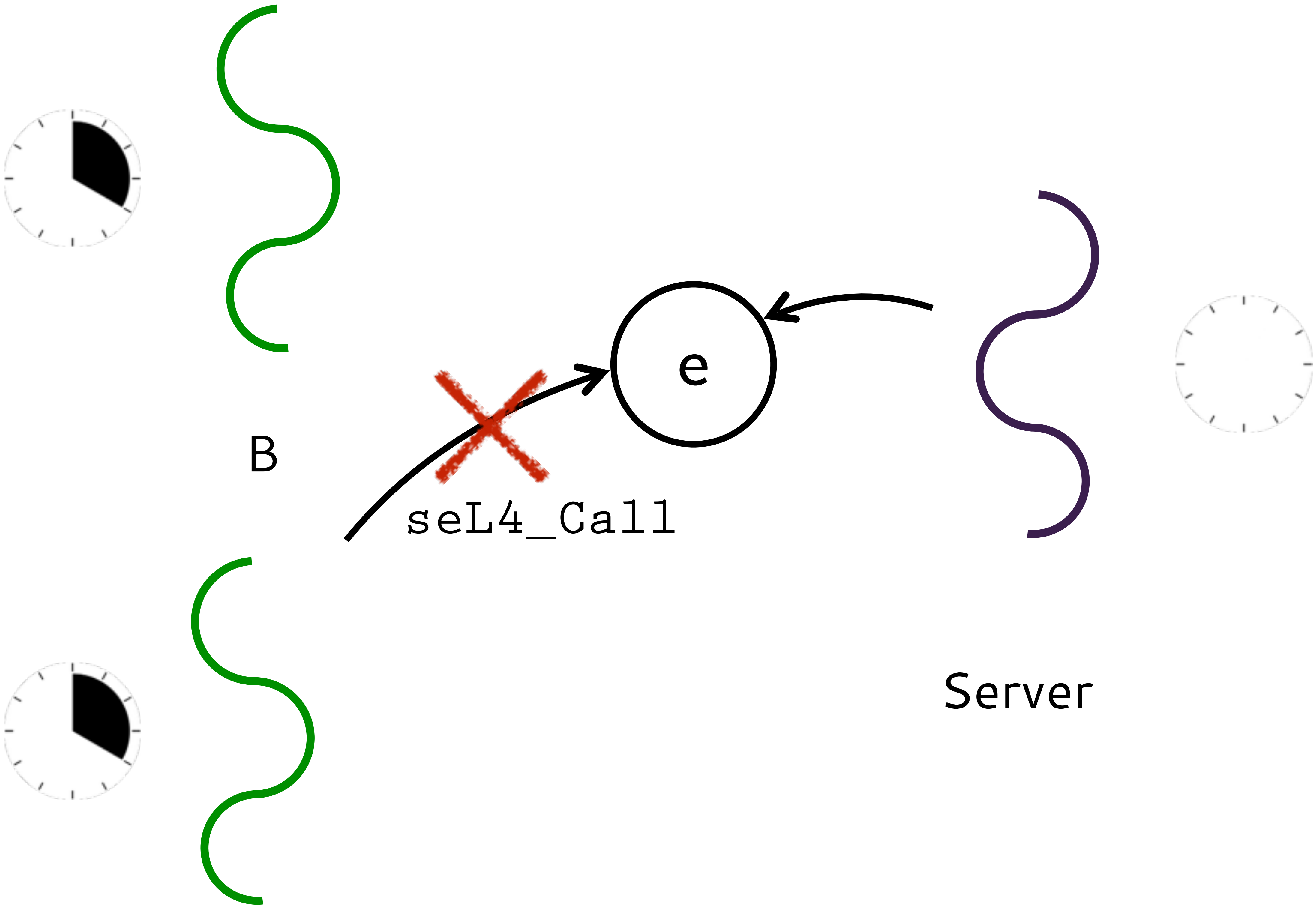
Active Servers (no temporal isolation)



Active Servers (no temporal isolation)



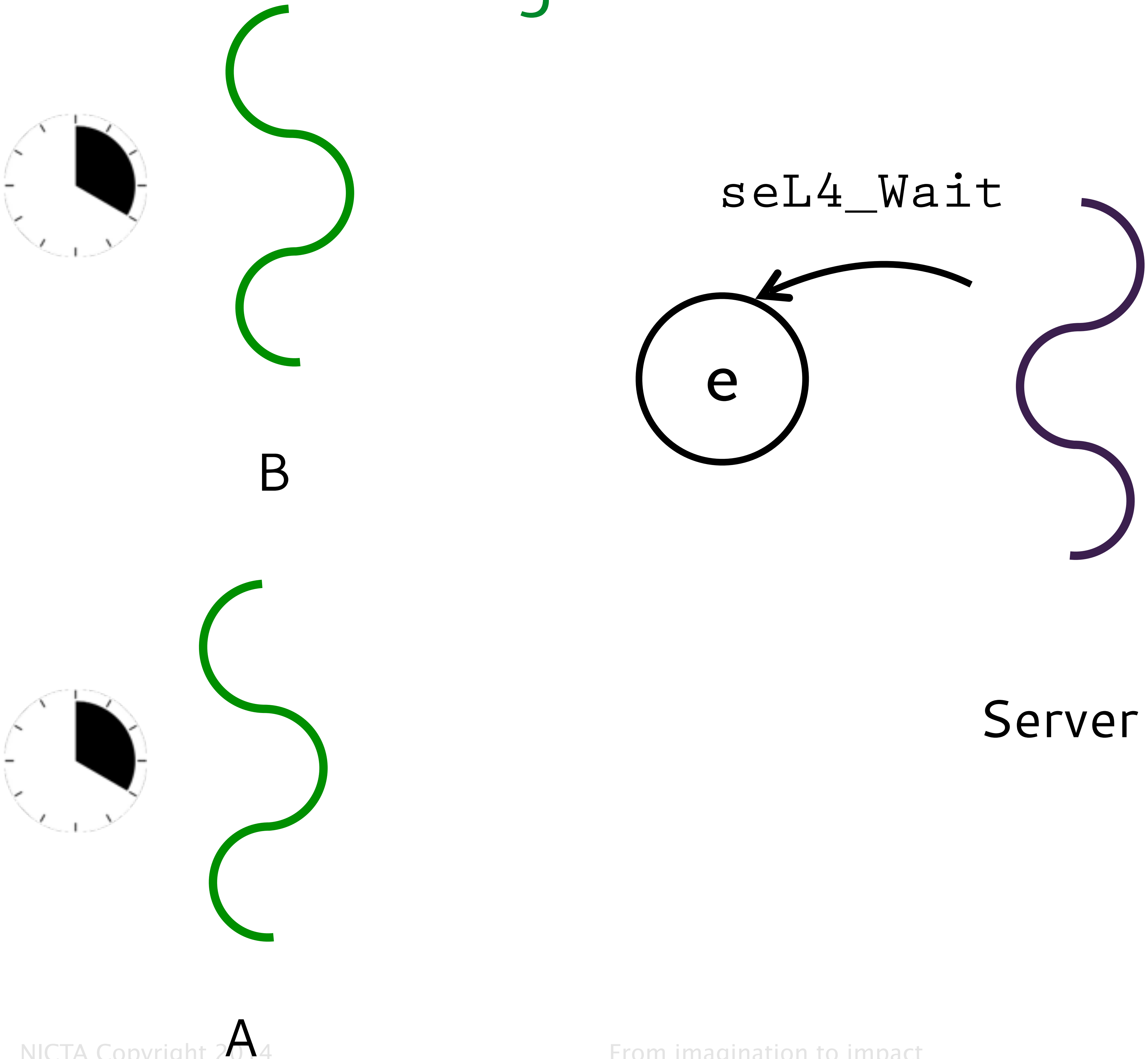
Active Servers (no temporal isolation)



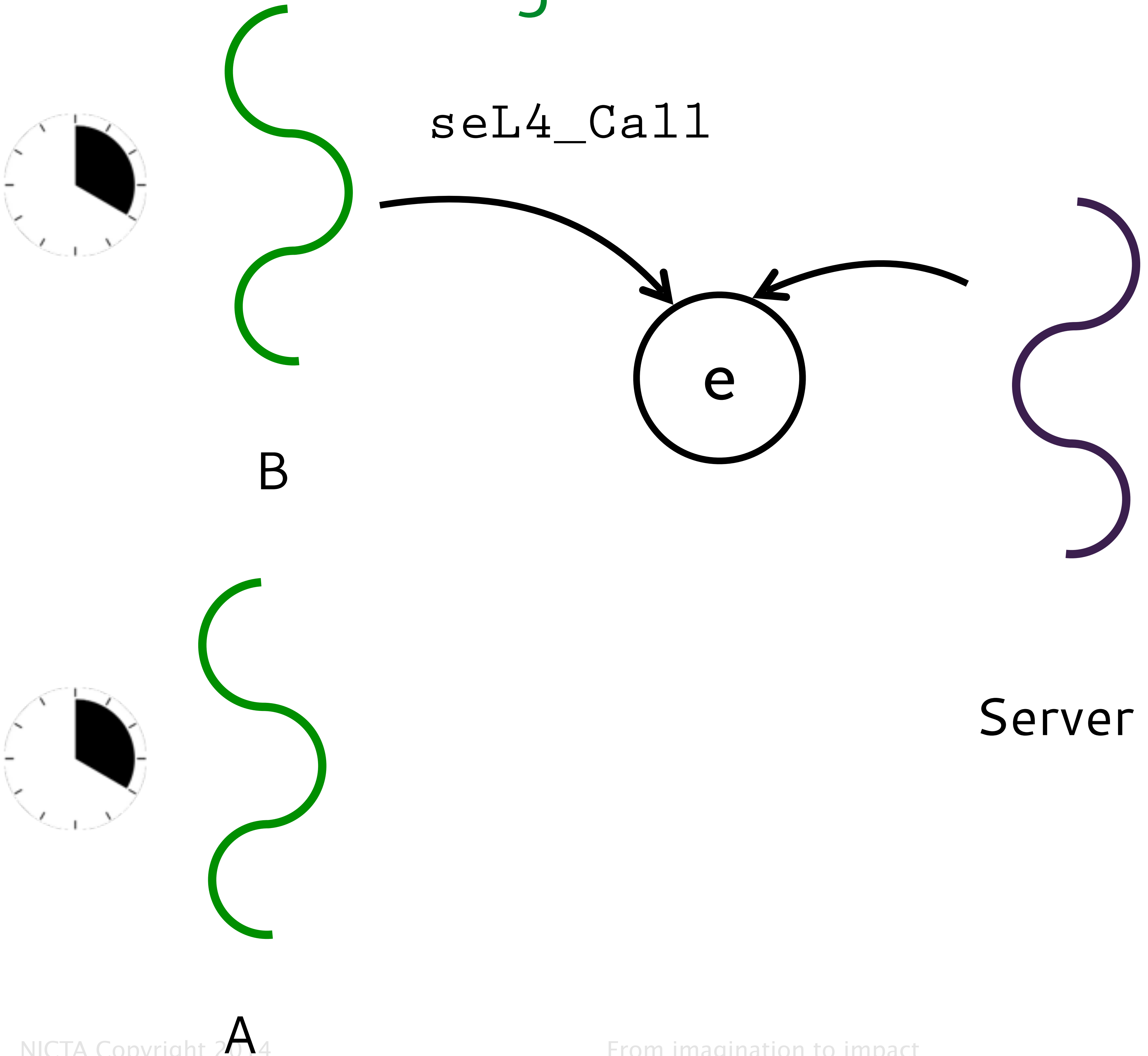
Scheduling context donation

- seL4_Call
  - where server is passive, donate scheduling context to server, otherwise do nothing
  - Must *trust* the server (use async for untrusted)
- seL4_ReplyWait
  - donates it back
  - reply cap represents a guarantee that the scheduling context will be returned

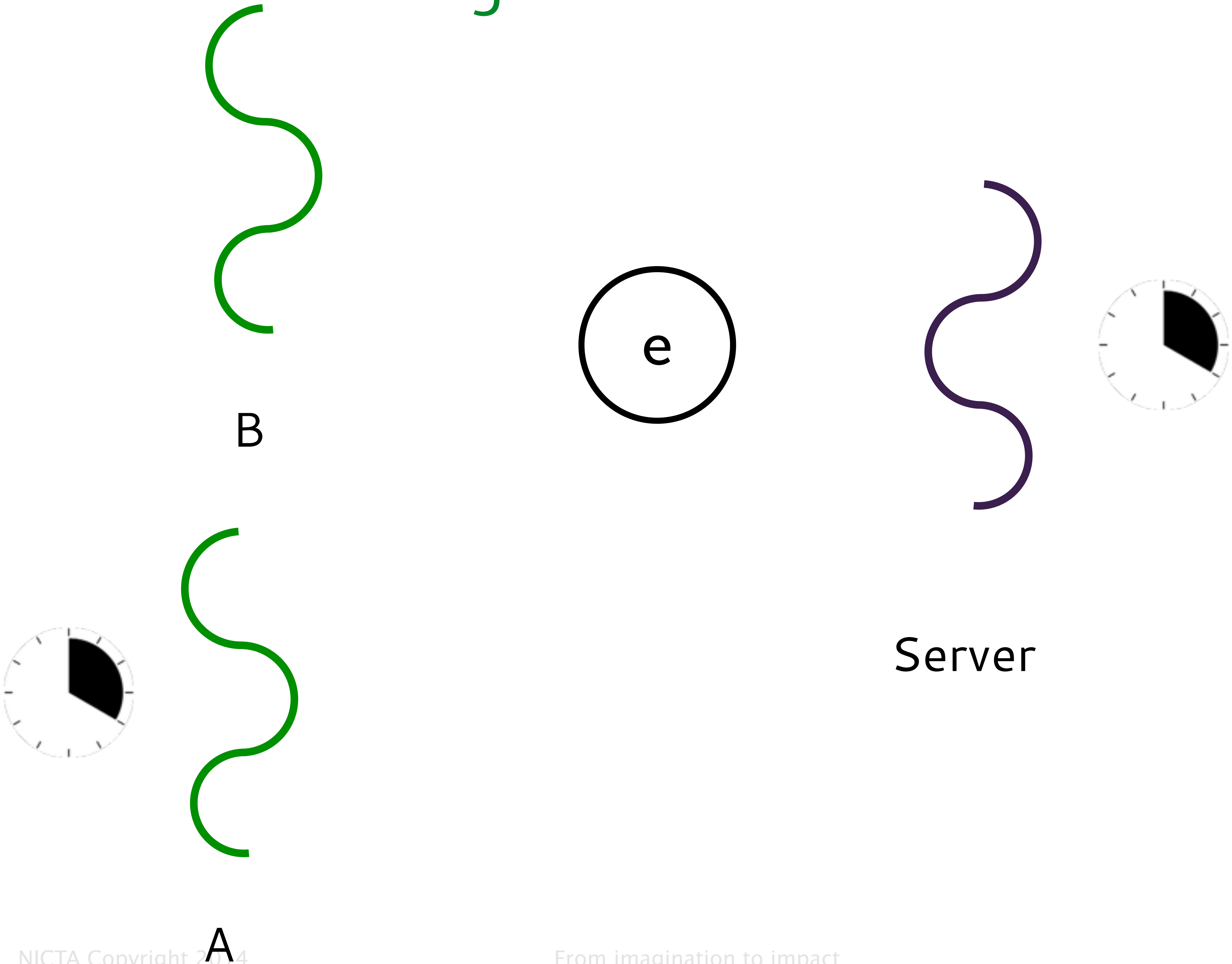
Scheduling context donation



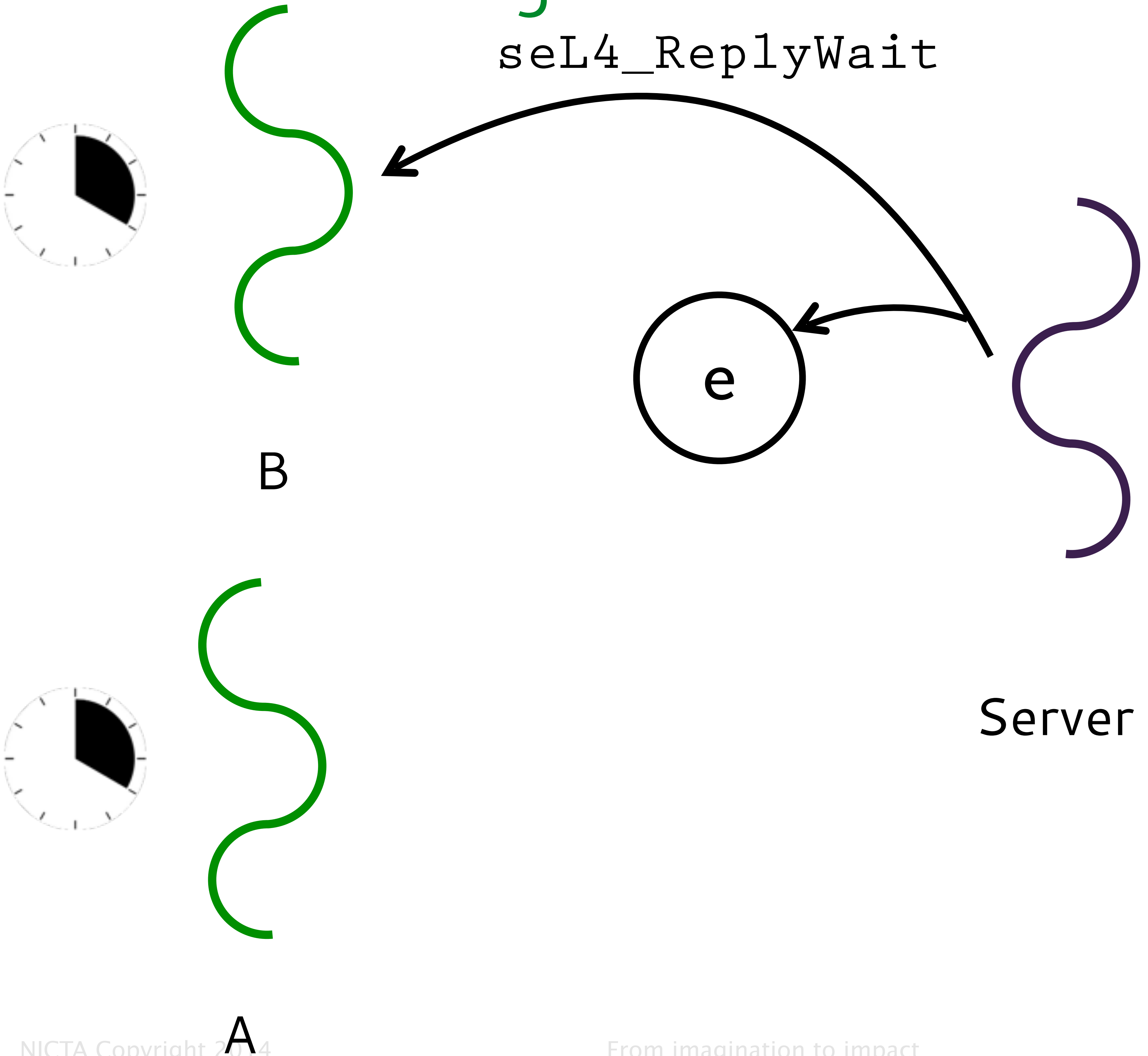
Scheduling context donation



Scheduling context donation



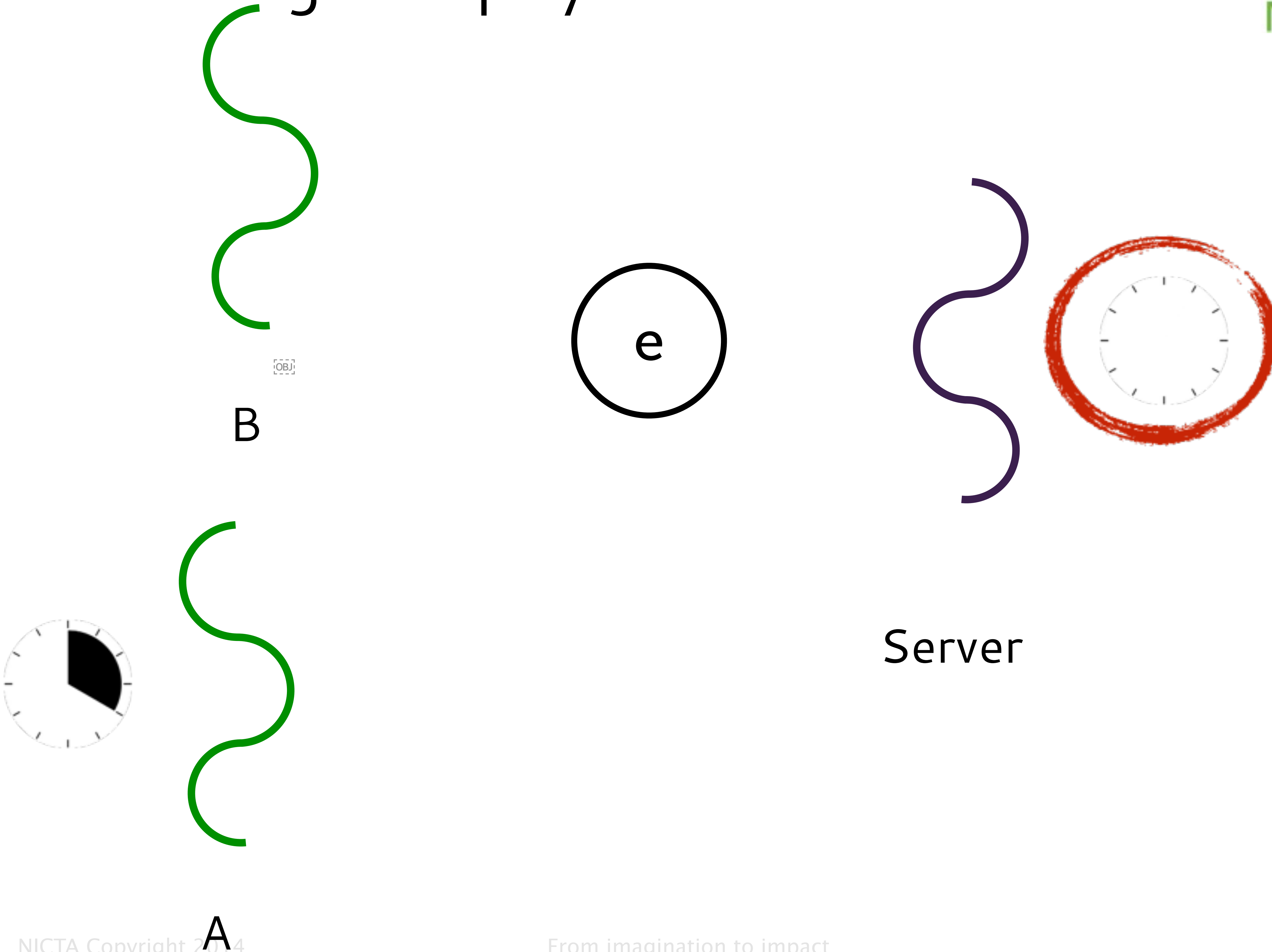
Scheduling context donation



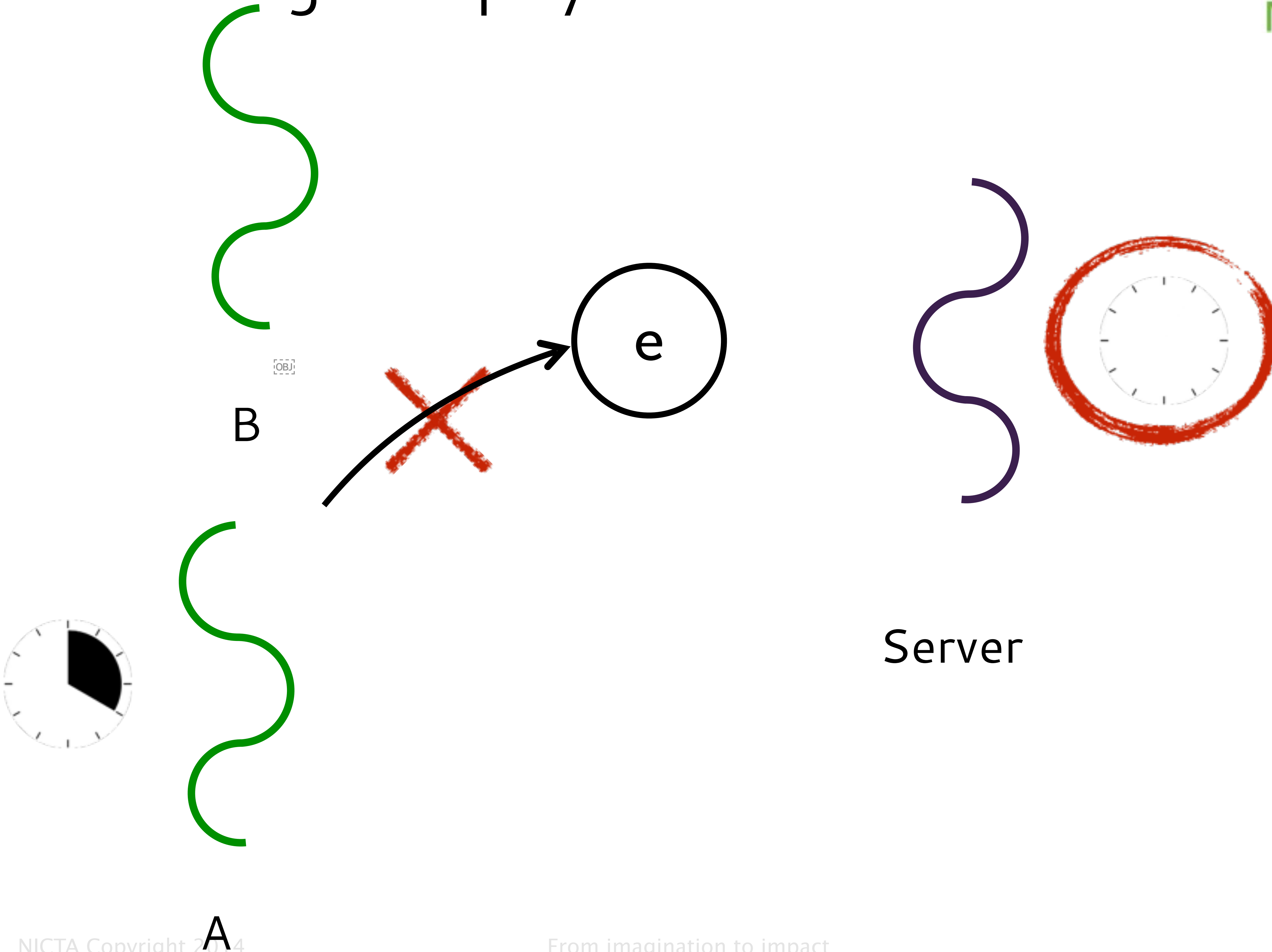
Summary: Resource sharing (so far)

- Scheduling context donation
  - only on Synchronous IPC with atomic send/recv operation
- Active and passive servers
  - Passive servers must always be trusted

Budget Expiry



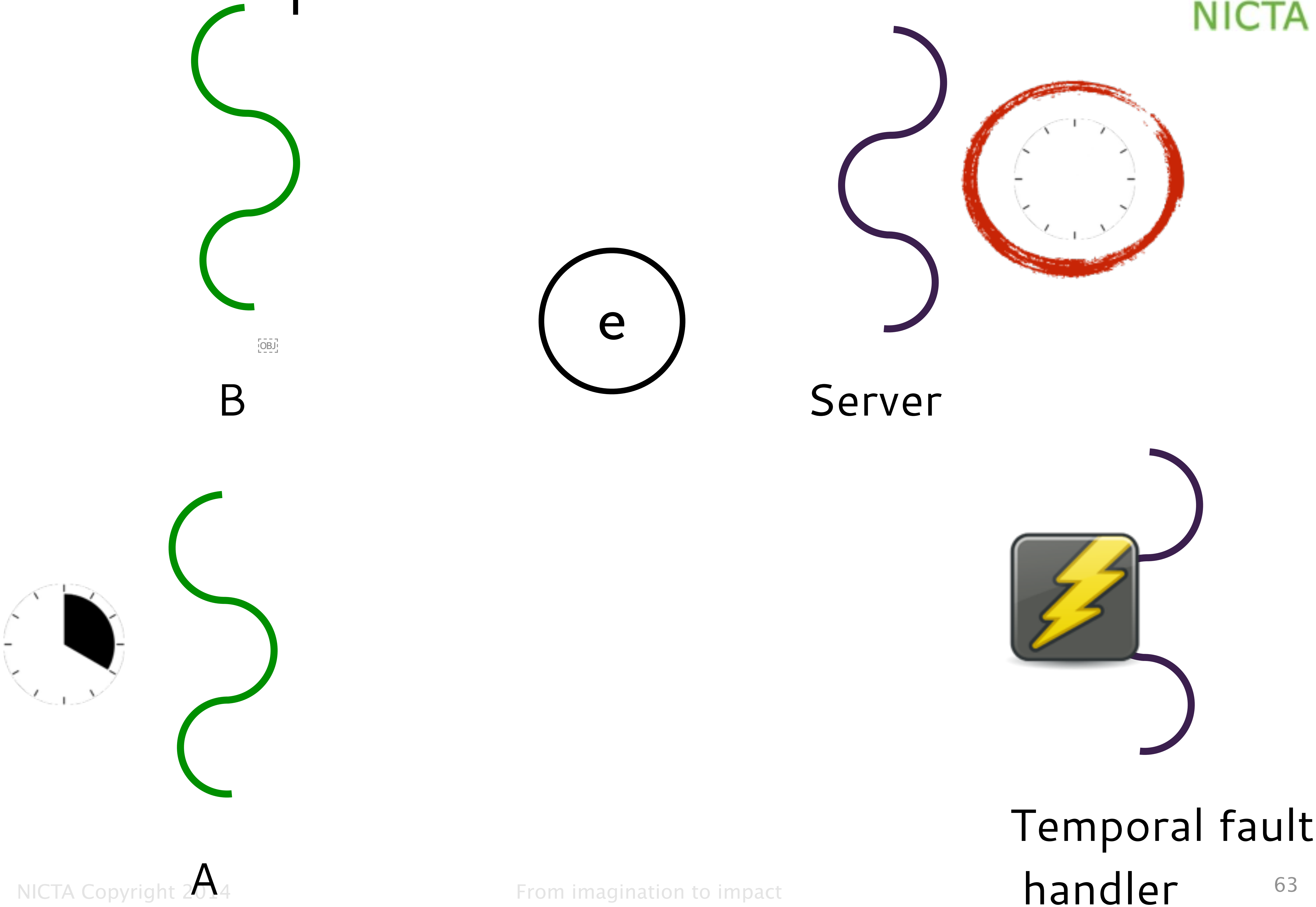
Budget Expiry



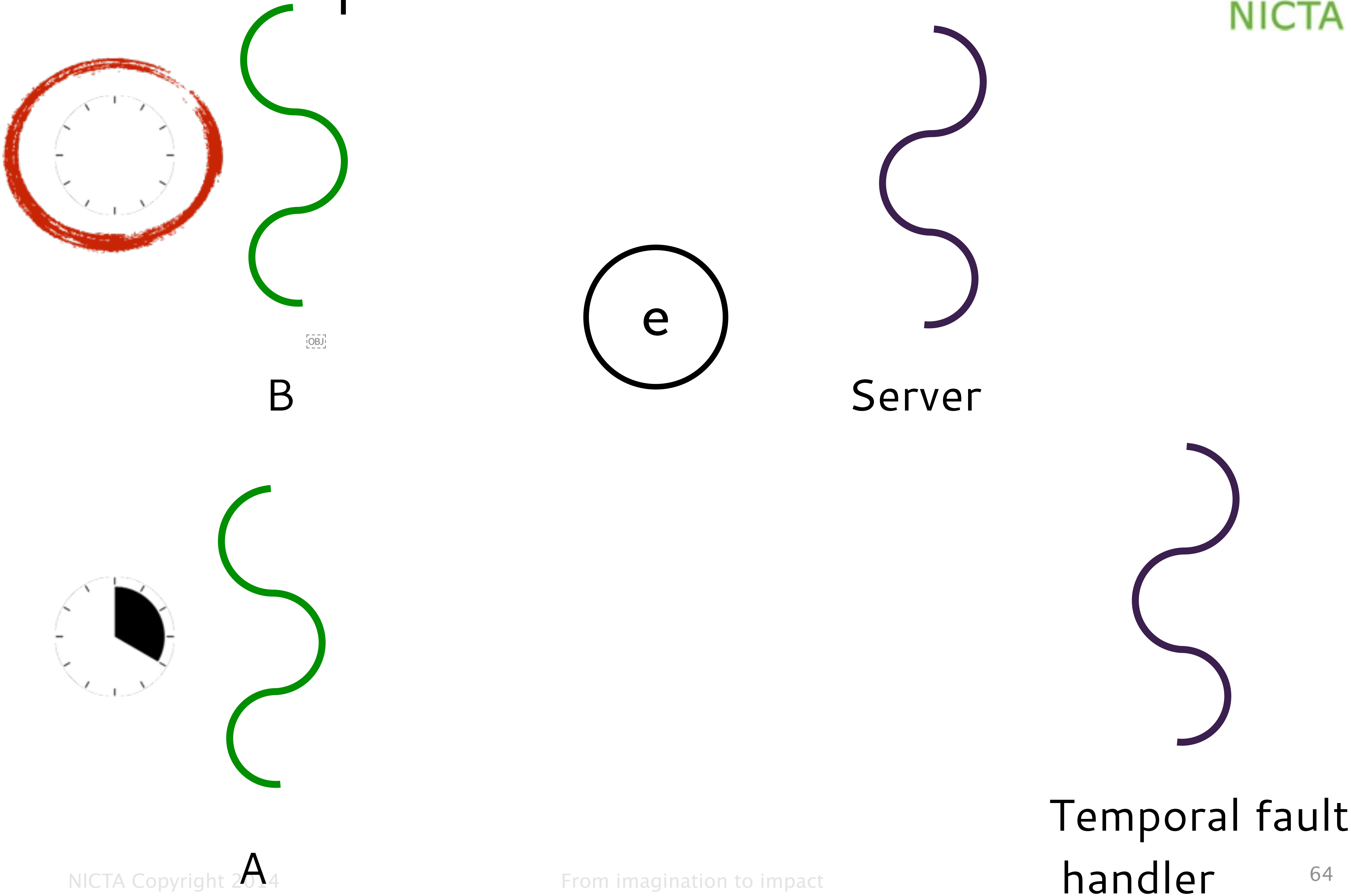
Alteratives for budget expiry

- Multithreaded servers
  - COMPOSITE [Parmer 2010]
  - possible with our impl.
- Bandwidth Inheritance + helping
  - Fiasco [Steinberg et.al. 2010]
  - we avoid this to avoid dependency trees/chains
- Temporal exceptions!

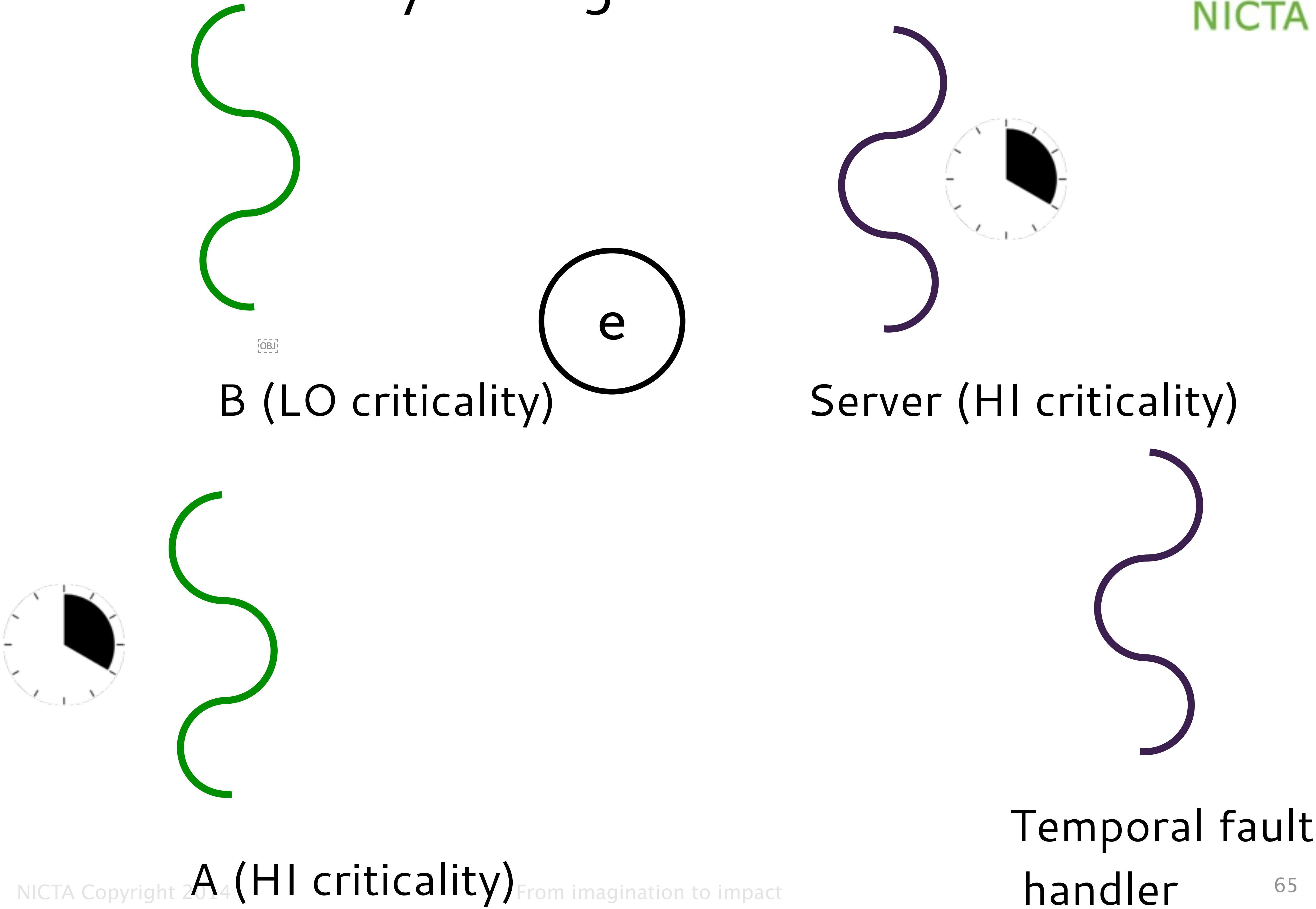
Exception + Rollback



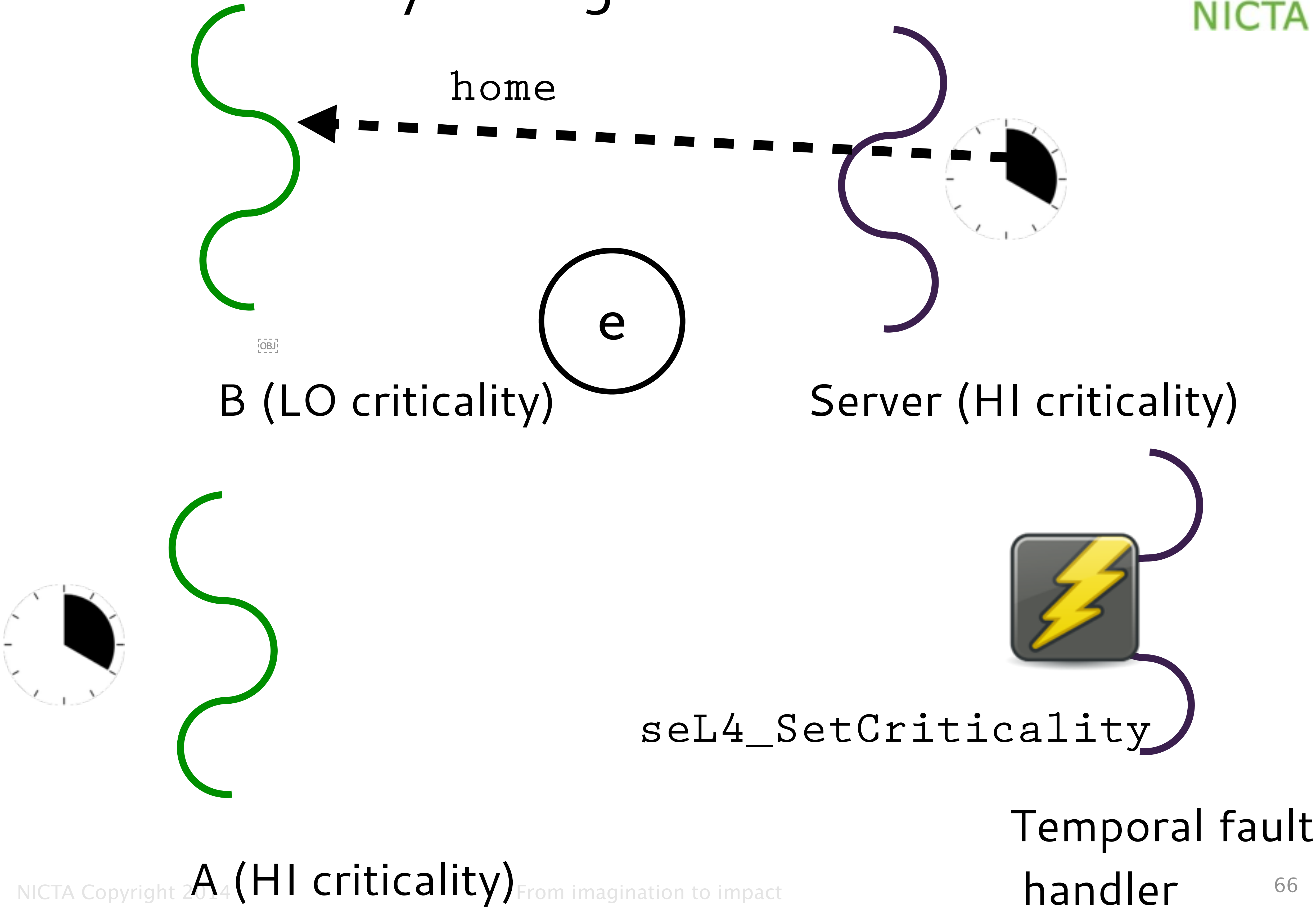
Exception + Rollback



Criticality change



Criticality change



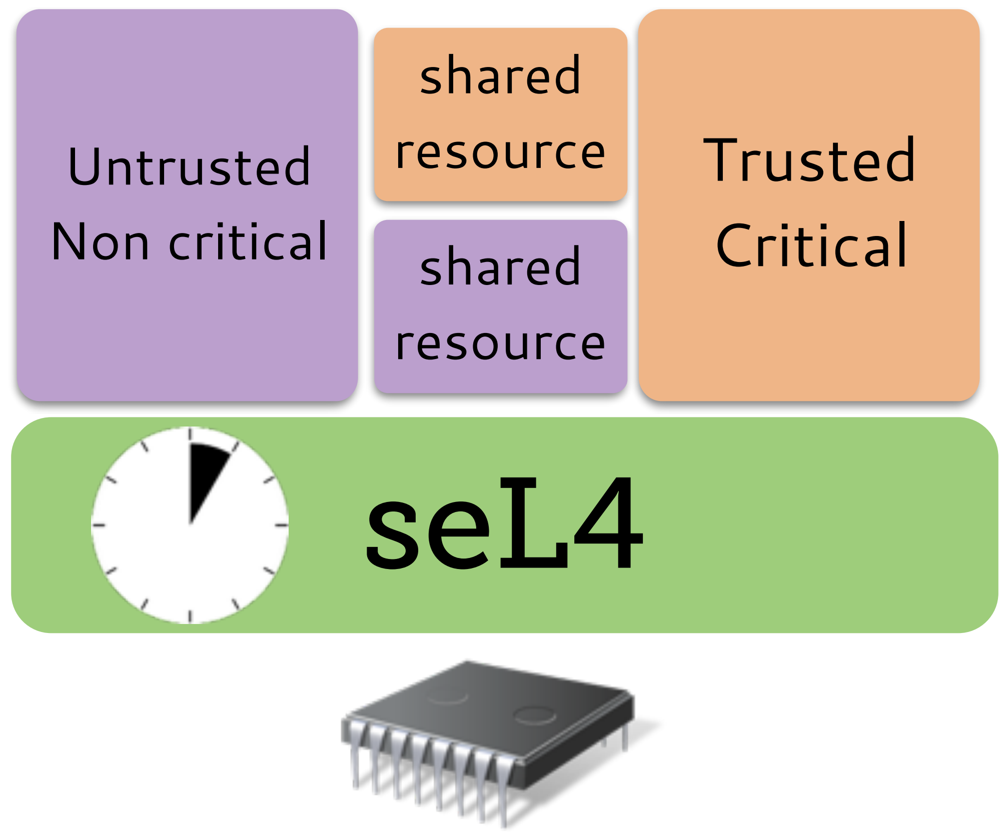
Exception + rollback

- Other actions possible on exception
  - like emergency reservation
- Rollback propagates to handle chains:
  - if a reply transfers an empty scheduling context, another temporal exception is raised
- User must implement rollback
  - middleware layer can do this

Summary: Resource sharing

- Multithreaded servers possible
- Budget expiry triggers temporal exceptions
  - which can be used to rollback or help a server
- So does criticality change
  - if lower criticality thread using server

Endgame



- Temporal isolation, asymmetric protection, safe bounded resource sharing achieved through scheduling contexts, criticality, temporal exceptions.

References + Credits

References

- B. Blackham, Y. Shi, S. Chattopadhyay, A. Roychoudhury and G. Heiser. Timing analysis of a protected operating system kernel. In 32nd RTSS, pp. 339–348, Vienna, Austria, November, 2009.
- DO178B Standard. http://en.wikipedia.org/wiki/DO-178B.
- G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal verification of an OS kernel. In 22nd SOSP, pages 207–220, Big Sky, MT, USA, Oct. 2009.
- A. K. Mok. Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment. PhD thesis, 1983.
- T .Murray, .D Matichuk, M. Brassil, P. Gammie, T. Bourke, S. Seefried, C. Lewis, X. Gao and G. Klein. seL4: From general purpose to a proof of information flow enforcement. IEEE Symposium on Security and Privacy, pp. 415-429, San Francisco, CA, May, 2013.

References

- Raj Rajkumar, Kanaka Juvva, Anastasio Molano, and Shuichi Oikawa. Resource kernels: a resource- centric approach to real-time and multimedia systems. In *Readings in multimedia computing and networking*, pages 476–490. Morgan Kaufmann Publishers Inc., 2001. ISBN 1-55860-651-3. URL http:// portal.acm.org.viviena.library.unsw.edu.au/citation.cfm?id=383915.
- Udo Steinberg, Alexander Bo¨ttcher, and Bernhard Kauer. Timeslice donation in component-based sys- tems. In *Workshop on Operating System Platforms for Embedded Real-Time Applications (OSPERT)*, Brussels, Belgium, 2010.
- Fiasco. http://os.inf.tu-dresden.de/fiasco/overview.html
- Gabriel Parmer. The case for thread migration: Predictable IPC in a customizable and reliable OS. In *Workshop on Operating System Platforms for Embedded Real-Time Applications (OSPERT)*, Brussels, Belgium, July 2010.

Image + Font Credits

- Fonts sourced from Font squirrel
- All other images are in the public domain (mostly from openclipart)