

School of Computer Science & Engineering

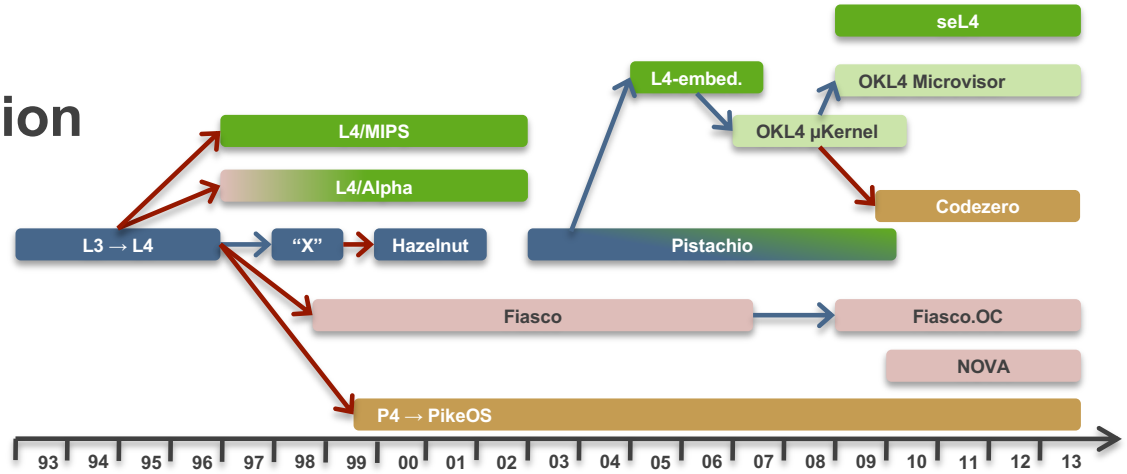
COMP9242 Advanced Operating Systems

2026 T3 Week 07 Part 1

Microkernel Design & Implementation

Continued…

@GernotHeiser



Copyright Notice

These slides are distributed under the
Creative Commons Attribution 4.0 International (CC BY 4.0) License

- You are free:
  - to share—to copy, distribute and transmit the work
  - to remix—to adapt the work
- under the following conditions:
  - **Attribution:** You must attribute the work (but not in any way that suggests that the author endorses you or your use of the work) as follows:

    "Courtesy of Gernot Heiser, UNSW Sydney"

The complete license text can be found at
http://creativecommons.org/licenses/by/4.0/legalcode

Lessons & Principles

© Gernot Heiser 2019 – CC BY 4.0

Reflecting on Lessons of 2nd Generation

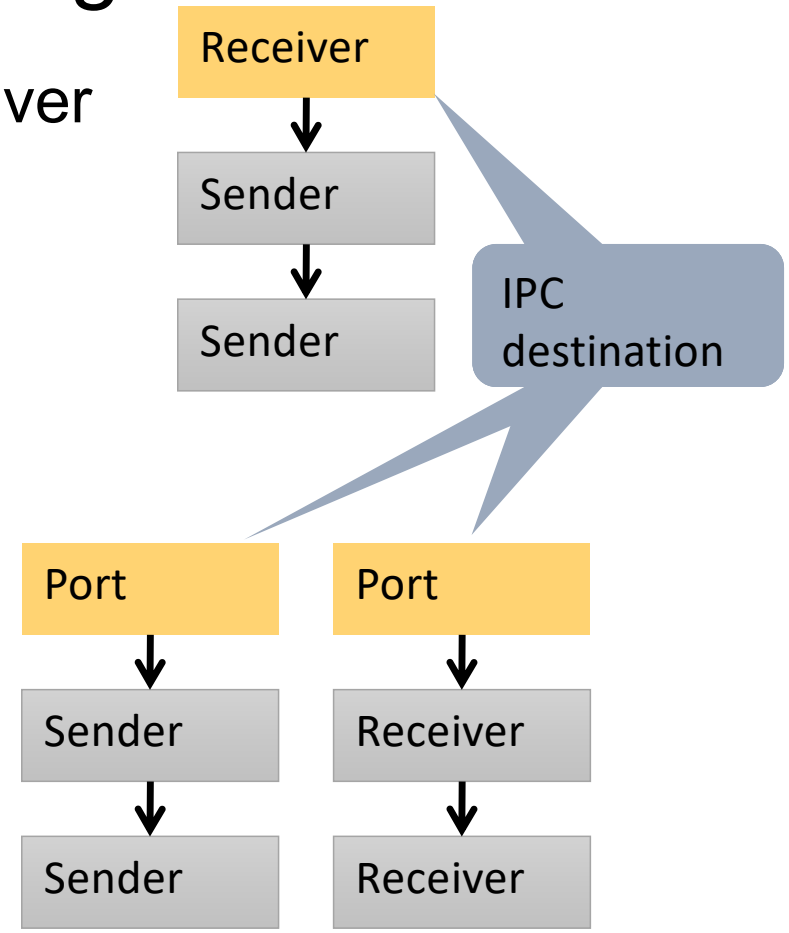
Original L4 design had two major shortcomings:

1. Insufficient/impractical resource control
   - Poor information hiding (IPC addressed to threads)
   - Poor/non-existent control over kernel memory use
   - Inflexible & costly process hierarchies (policy!)
   - Arbitrary limits on number of address spaces and threads (policy!)
   - Insufficient mechanisms for authority delegation
2. Over-optimised IPC abstraction, mangles:
   - Communication, incl bulk data copy
   - Synchronisation
   - Timed wait
   - Memory management – sending mappings
   - Scheduling – time-slice donation

© Gernot Heiser 2019 – CC BY 4.0

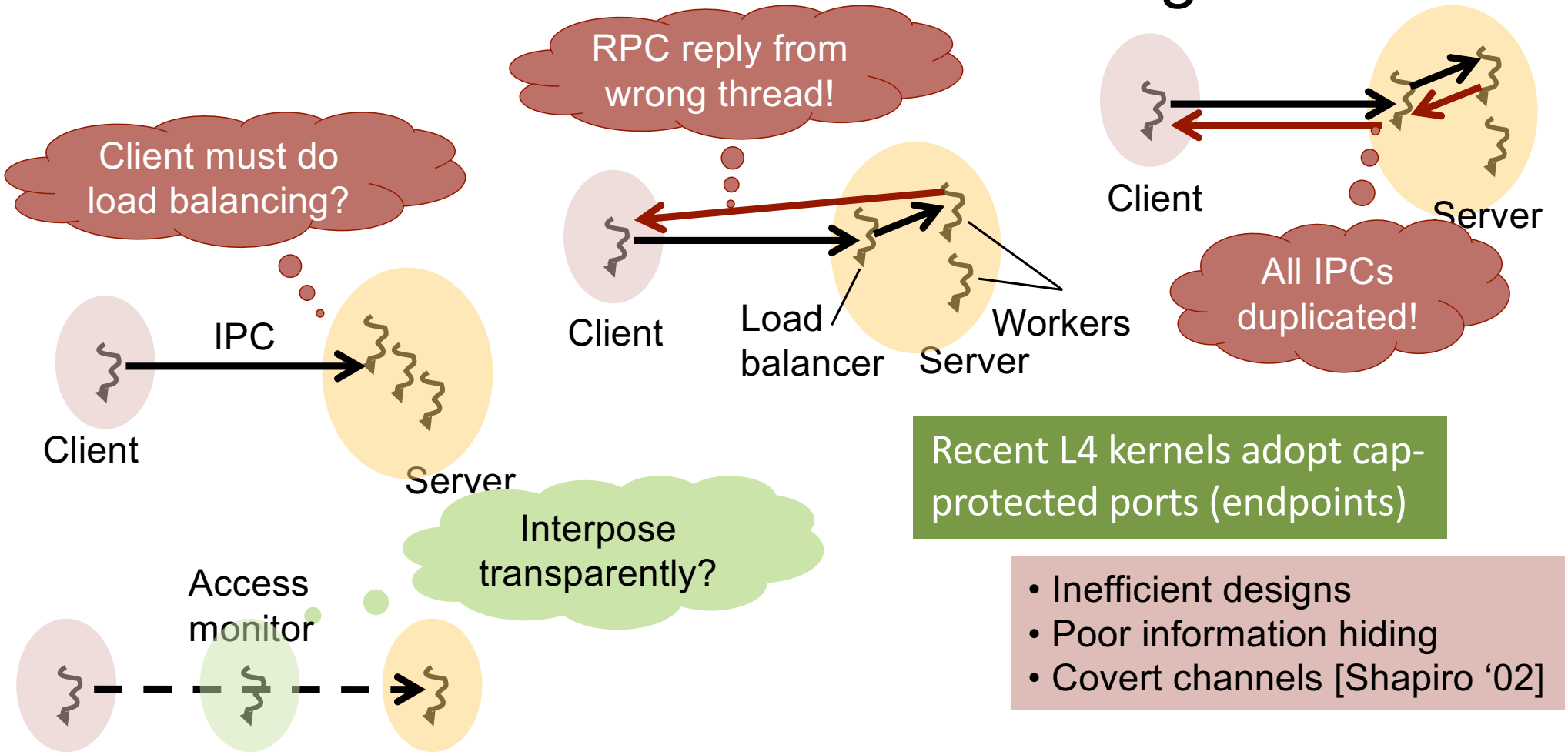
Direct vs Indirect IPC Addressing

- Direct: Queue senders/messages at receiver
  - Need unique thread IDs
  - Kernel guarantees identity of sender
    - useful for authentication
- Indirect: Mailbox/port object
  - Just a user-level handle for the kernel-level queue
  - Extra object type – extra weight?
  - Communication partners are anonymous
    - Need separate mechanism for authentication



© Gernot Heiser 2019 – CC BY 4.0

Other Issues with L4 IPC Adressing



Recent L4 kernels adopt cap-protected ports (endpoints)

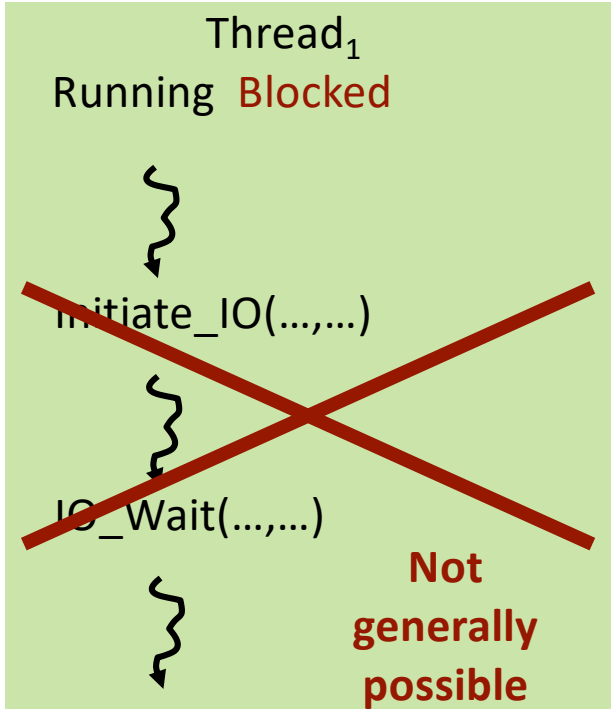
- Inefficient designs
- Poor information hiding
- Covert channels [Shapiro '02]

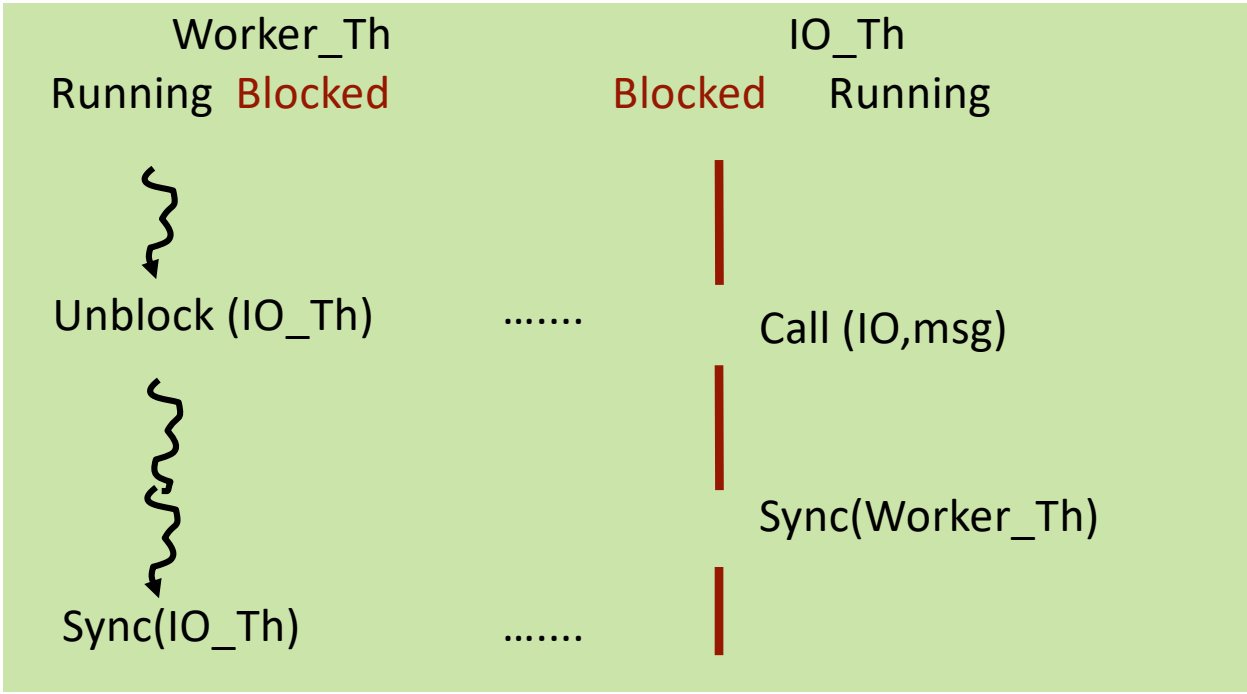
© Gernot Heiser 2019 – CC BY 4.0

Synchronous IPC issues

- Sync IPC forces multi-threaded code or select()!
- Also poor choice for multi-core

seL4:
Notification binding



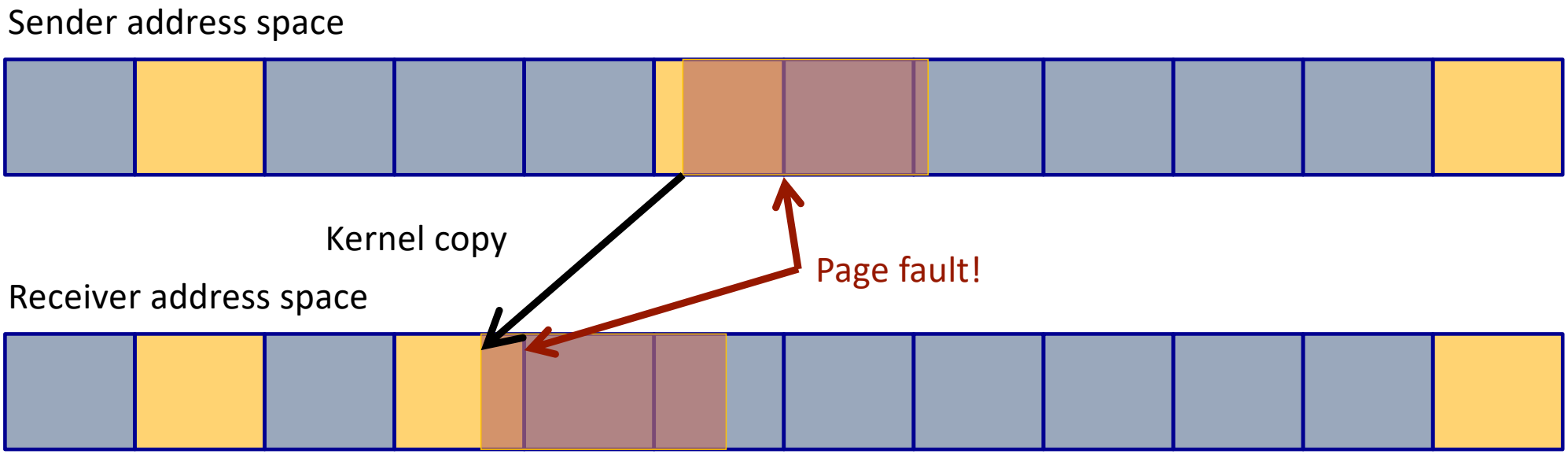


© Gernot Heiser 2019 – CC BY 4.0

L4 “Long” IPC

seL4: Removed

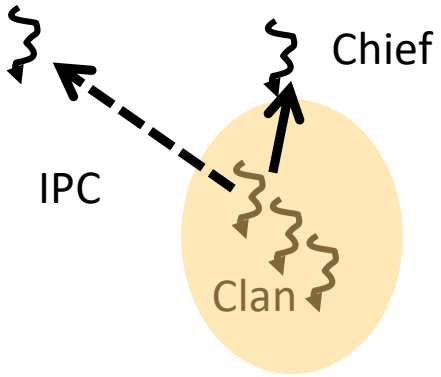
- Not minimal
- Source of kernel complexity:
  - nested exceptions
  - concurrency in kernel
  - must upcall PF handlers during IPC
  - timeouts to prevent DOS attacks



© Gernot Heiser 2019 – CC BY 4.0

Other L4 Design Issues

IPC Control: "Clans & Chiefs"



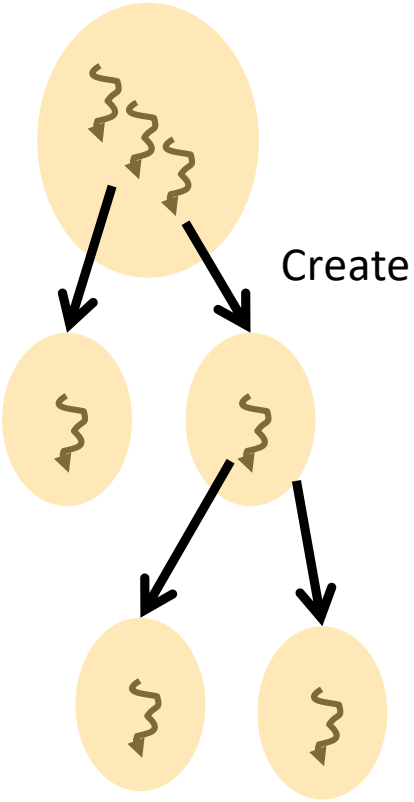
IPC outside clan re-directs to chief

seL4: Capabilities

- Policy!
- Inflexible, clumsy, inefficient hierarchies!
- Fundamental problem: no rights delegation

Process Hierarchy

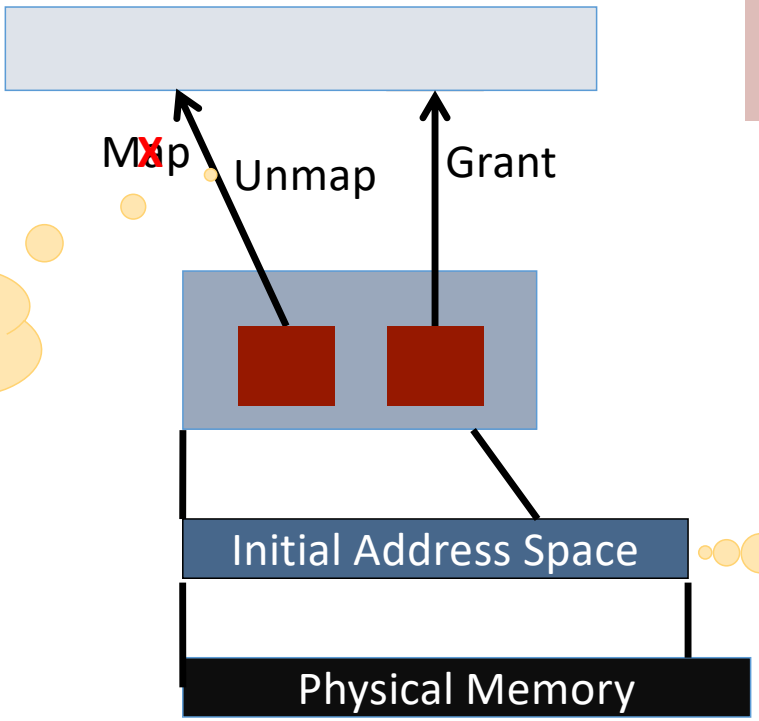
Hierarchical resource management



© Gernot Heiser 2019 – CC BY 4.0

Traditional L4: Recursive Address Spaces

Issues:
- Complex mapping DB
- Exhaustion of kernel memory
- Complex IPC semantics



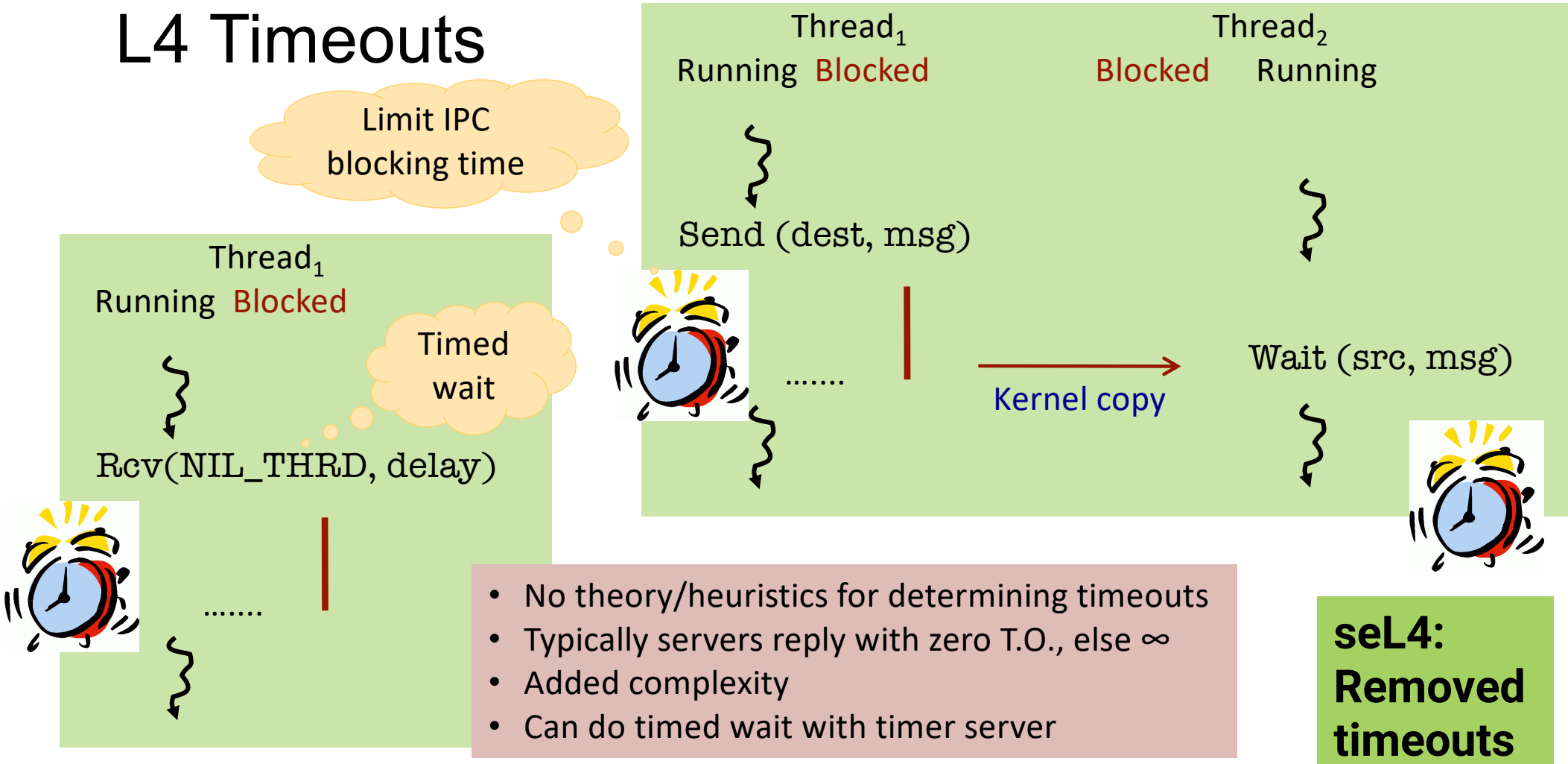
Mappings are page → page, sent by IPC

seL4:
Map frame caps

Magic initial AS to anchor recursion (map of PM)

© Gernot Heiser 2019 – CC BY 4.0

L4 Timeouts

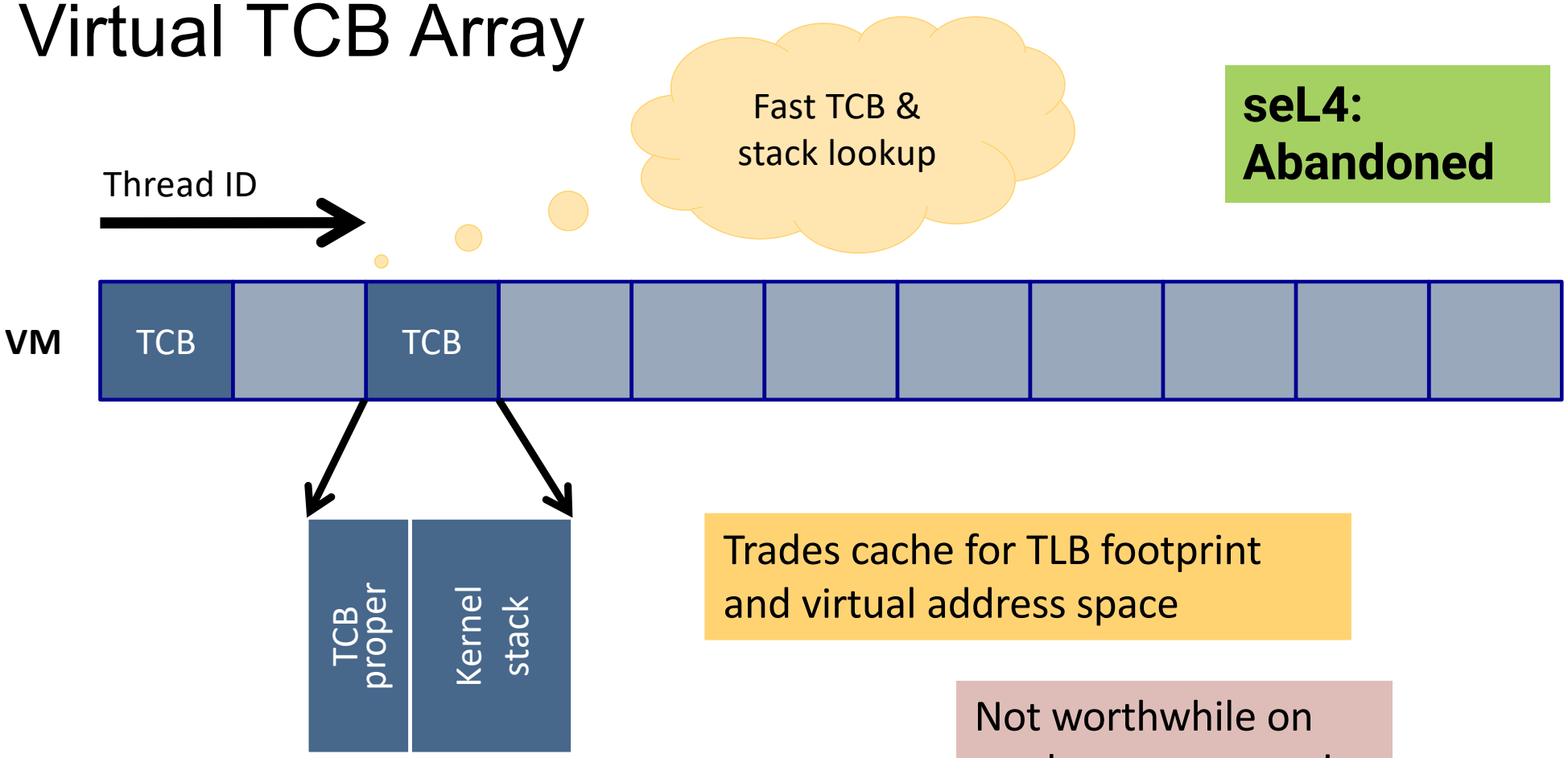


- No theory/heuristics for determining timeouts
- Typically servers reply with zero T.O., else ∞
- Added complexity
- Can do timed wait with timer server

seL4: Removed timeouts

© Gernot Heiser 2019 – CC BY 4.0

Virtual TCB Array



seL4: Abandoned

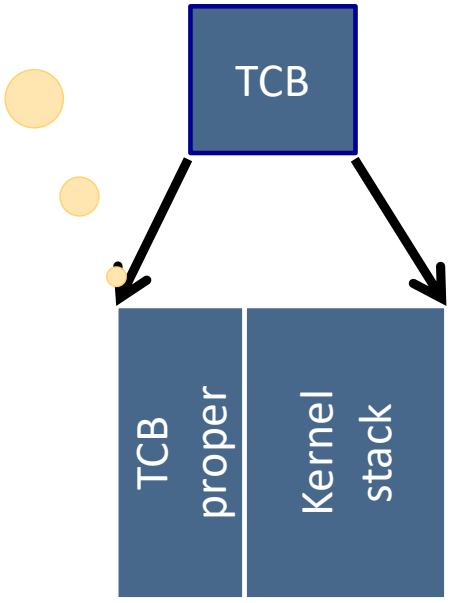
Trades cache for TLB footprint and virtual address space

Not worthwhile on modern processors!

© Gernot Heiser 2019 – CC BY 4.0

Process Kernel: Per-Thread Kernel Stack

Get own TCB base by masking stack pointer

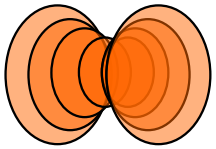


- Not worthwhile on modern processors!
- Stacks dominate kernel memory use!

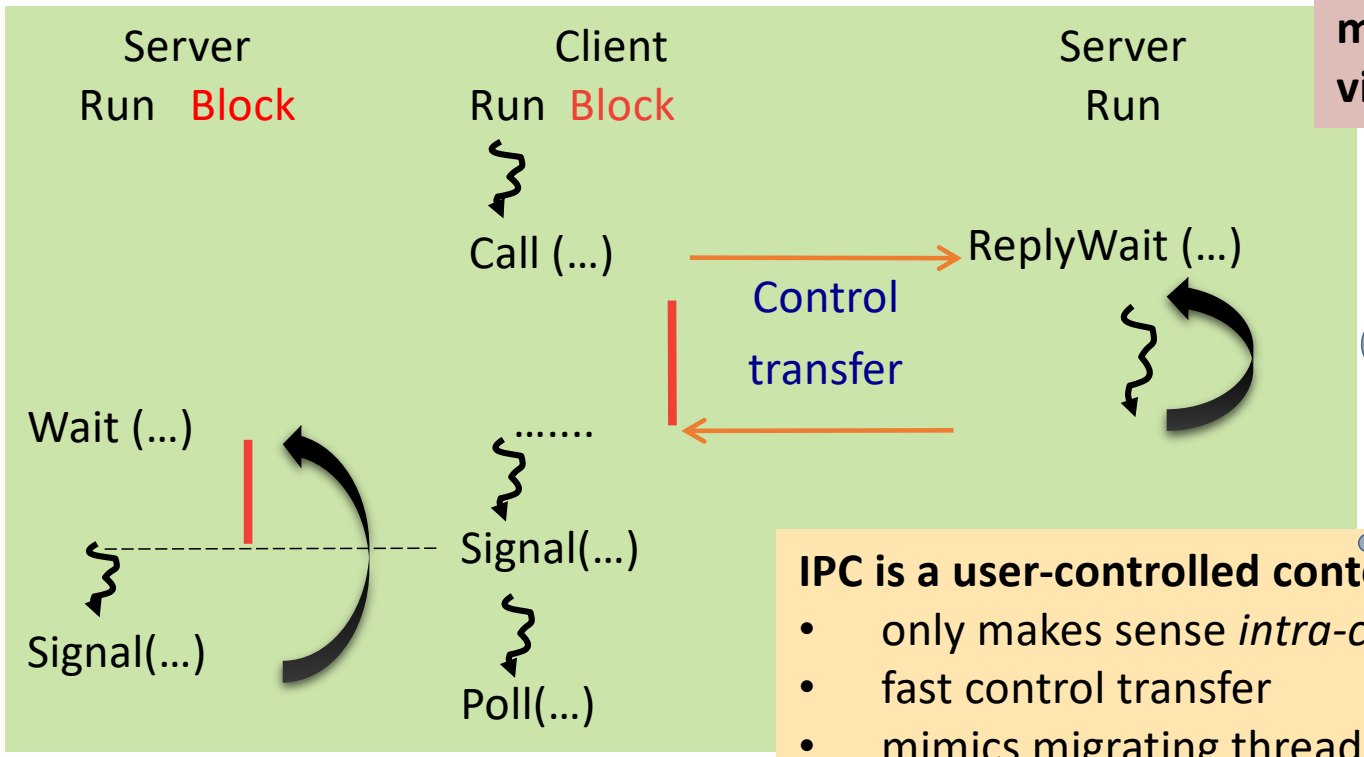
seL4: Event kernel

- Reduces TLB footprint at cost of cache and kernel memory
- Easier to deal with blocking

© Gernot Heiser 2019 – CC BY 4.0



Is IPC Redundant?



2 communication mechanisms: Minimality violation?

More principled with MCS *passive servers*

IPC is a user-controlled context switch

- only makes sense *intra-core*
- fast control transfer
- mimics migrating threads
- enables scheduling context donation
  - ➢ useful for real-time systems

© Gernot Heiser 2019 – CC BY 4.0

Original L4 Design and Implementation

Implement. Tricks [SOSP'93]

- ~~Process kernel~~
- ~~Virtual TCB array~~
- Lazy scheduling
- Direct process switch
- Non-preemptible
- ~~Non-portable~~
- ~~Non-standard calling convention~~
- ~~Assembler~~

Modified

Retained

Design Decisions [SOSP'95]

- Synchronous IPC
- ~~Rich message structure, arbitrary out-of-line messages~~
- Zero-copy register messages
- User-mode page-fault handlers
- ~~Threads as IPC destinations~~
- ~~IPC timeouts~~
- ~~Hierarchical IPC control~~
- User-mode device drivers
- ~~Process hierarchy~~
- ~~Recursive address-space construction~~

Added Notifications

© Gernot Heiser 2019 – CC BY 4.0

Other Lessons from 2nd Generation

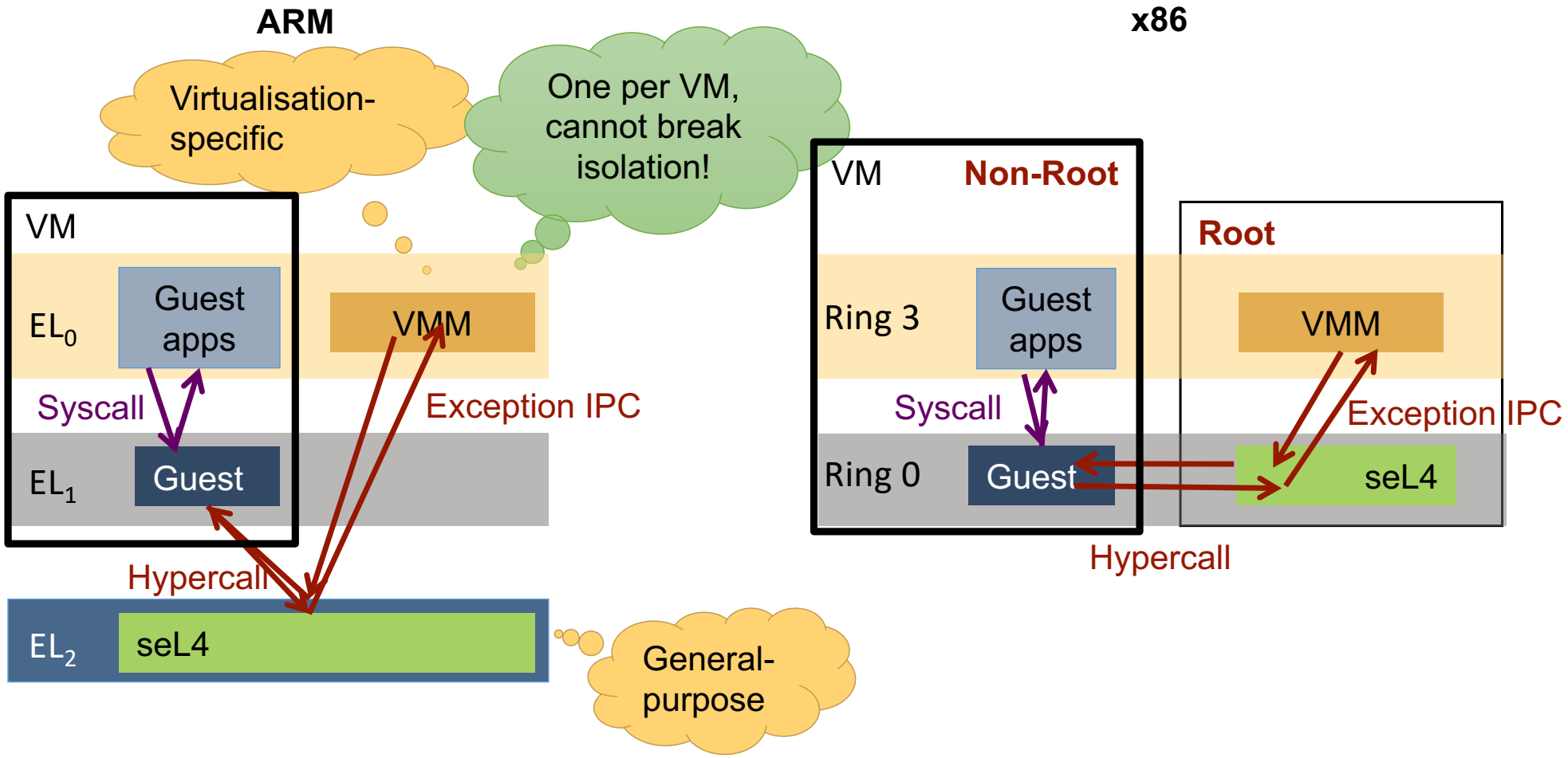
- Programming languages:
  - original i496 kernel [’95]: all assembler, use all registers
  - UNSW MIPS and Alpha kernels [’96,’98]: half assembler, half C
  - Fiasco [TUD ’98], Pistachio [’02]: C++ with assembler “fast path”
  - seL4 [‘09], OKL4 [‘09]: all C
- Lessons:
  - C++ sux: code bloat, no real benefit
  - Changing calling conventions not worthwhile
    - Conversion cost in library stubs and when entering C in kernel
    - Reduced compiler optimization
  - Assembler unnecessary for performance
    - C entry from assembler cheap if calling conventions maintained
    - seL4 performance with C-only pastpath as good as other L4 kernels [Blackham & Heiser ‘12]

© Gernot Heiser 2019 – CC BY 4.0

Virtualisation: Microkernel as a Hypervisor

© Gernot Heiser 2019 – CC BY 4.0

Microkernel as Hypervisor (NOVA, seL4)



© Gernot Heiser 2019 – CC BY 4.0

Hypervisors vs Microkernels

- Both contain all code executing at highest privilege level
  - Although hypervisor may contain user-mode code as well
    - privileged part usually called “hypervisor”
    - user-mode part often called “VMM”
- Both need to abstract hardware resources
  - Hypervisor: abstraction closely models hardware
  - Microkernel: abstraction designed to support wide range of systems

Difference to traditional terminology!

To abstract:
- CPU
- Memory
- Communication
- I/O

© Gernot Heiser 2019 – CC BY 4.0

What *Is* the Difference?

Resource	Hypervisor	Microkernel
Memory	Virtual MMU (vMMU)	Address space
CPU	Virtual CPU (vCPU)	Thread or scheduler activation
Communication	Virtual NIC, with device driver and network stack	High-performance message-passing IPC
I/O	• Simplified virtual device • Driver in hypervisor • Virtual IRQ (vIRQ)	• IPC interface to user-mode driver • Interrupt IPC

Just page tables in disguise

Just kernel-scheduled activities

Modelled on HW, Re-uses SW

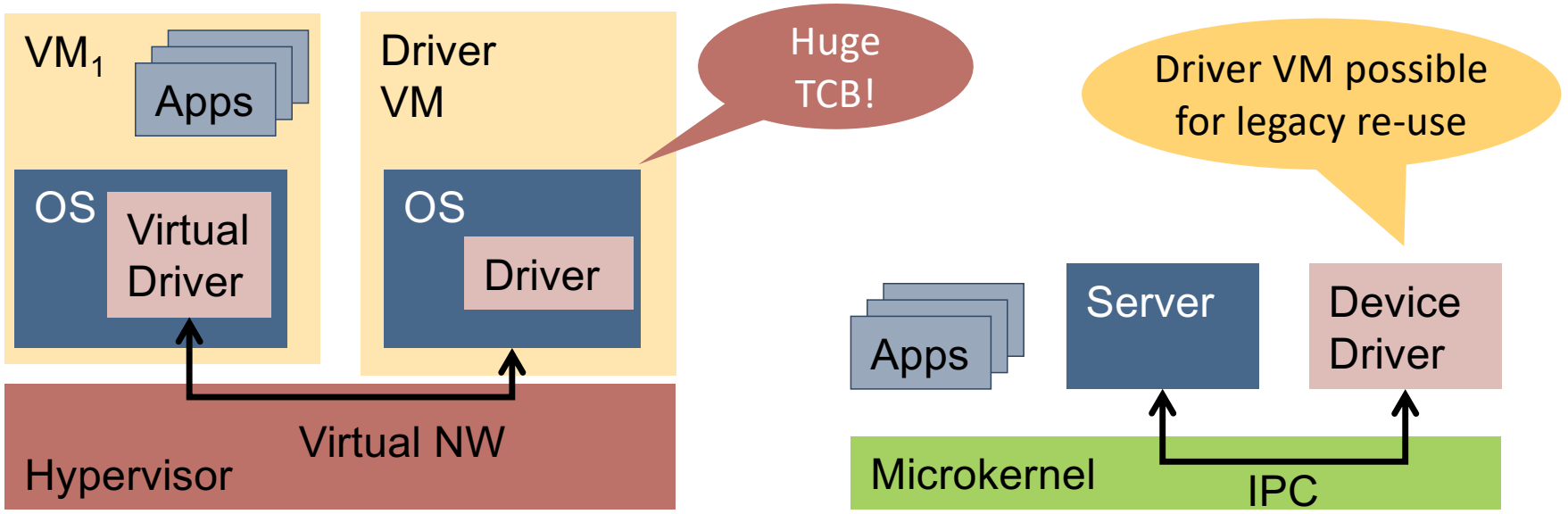
Minimal overhead, Custom API

Real Difference?

- Similar abstractions
- Optimised for different use cases

© Gernot Heiser 2019 – CC BY 4.0

Closer Look at Communication and I/O



Communication is critical for I/O

- Highly-optimised microkernel IPC
- Inter-VM communication is frequently a bottleneck in hypervisors

© Gernot Heiser 2019 – CC BY 4.0



Unresolved in Original seL4: Time

Simplistic fixed-priority scheduling

- Fixed by scheduling contexts

Priority violation through “lazy scheduling”

- Fixed by Benno scheduling in about 2015

Lack of temporal isolation in server invocations

- Fixed by SC donation to passive servers (MCS kernel)

Complexity forced on servers through MCS budget timeouts

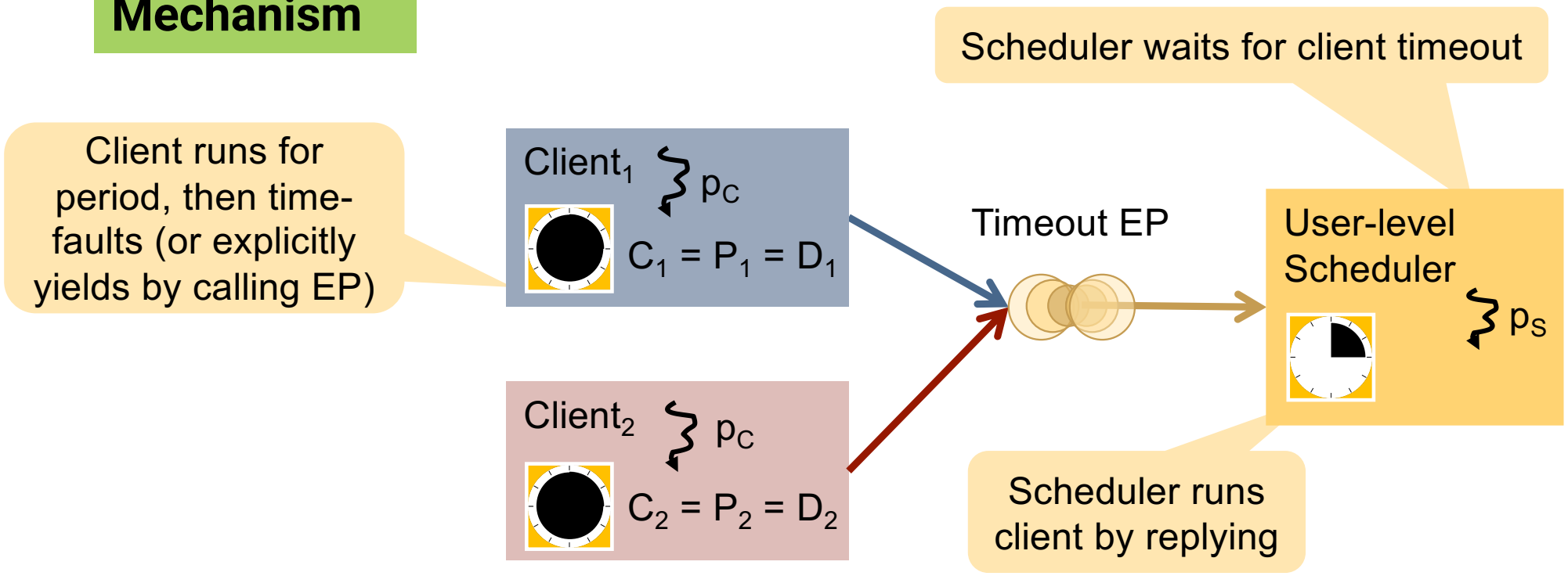
- Pending RFC for budget thresholds on endpoints

© Gernot Heiser 2019 – CC BY 4.0

Isn't a Fixed-Prio Scheduler Policy?

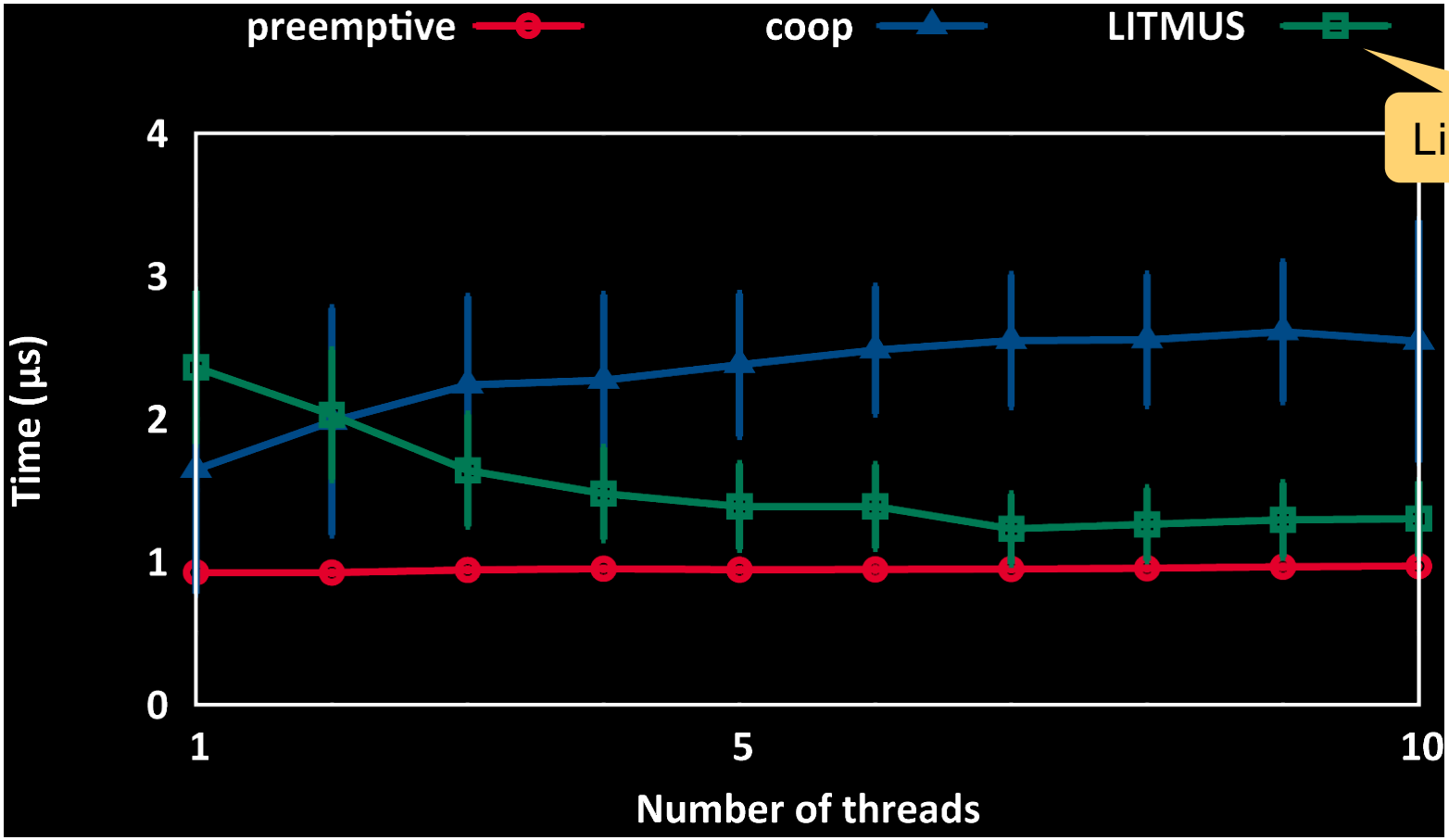
Prios + SCs = Mechanism

Implementing scheduling policy at user level



© Gernot Heiser 2019 – CC BY 4.0

User-Level EDF Scheduler Performance



© Gernot Heiser 2019 – CC BY 4.0



Design Principles

- Fully delegable access control via capabilities
- All resource management is subject to user-defined policies
  - Applies to kernel resources too!
- Performance on par with best-performing L4 kernels
  - Prerequisite for real-world deployment!
- Suitability for real-time use
  - Important for safety-critical systems
- Suitable for *formal verification*
  - Requires small size, avoid complex constructs

Largely in line with traditional L4 approach!

© Gernot Heiser 2019 – CC BY 4.0