Introduction to seL4

Security is no excuse for poor performance!

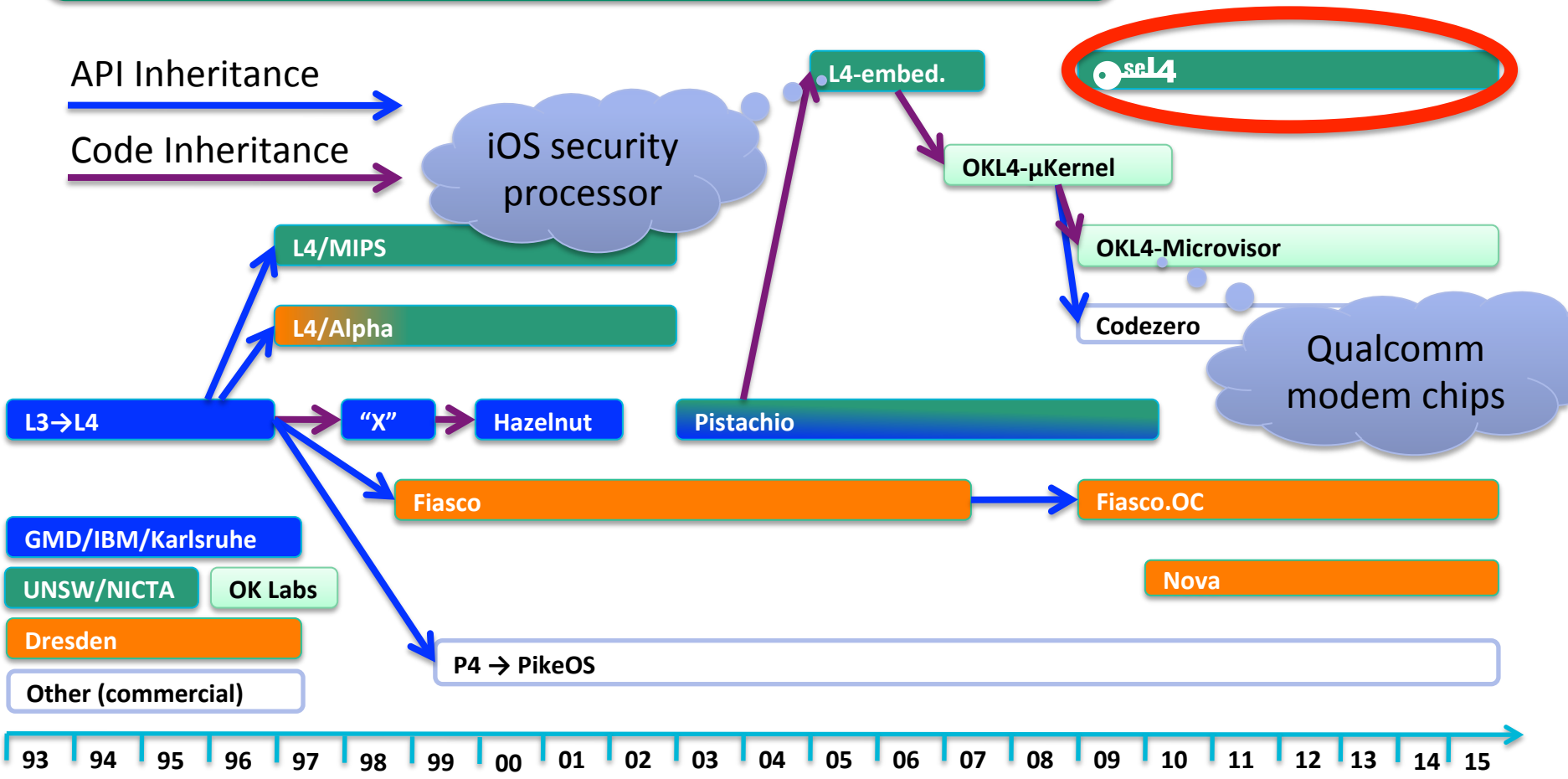
Gernot Heiser | gernot.heiser@data61.csiro.au | @GernotHeiser

https://sel4.systems

Copyright Notice

L4 Family Tree

seL4: The latest (and most advanced) member of the L4 microkernel family – 20 years of history and experience



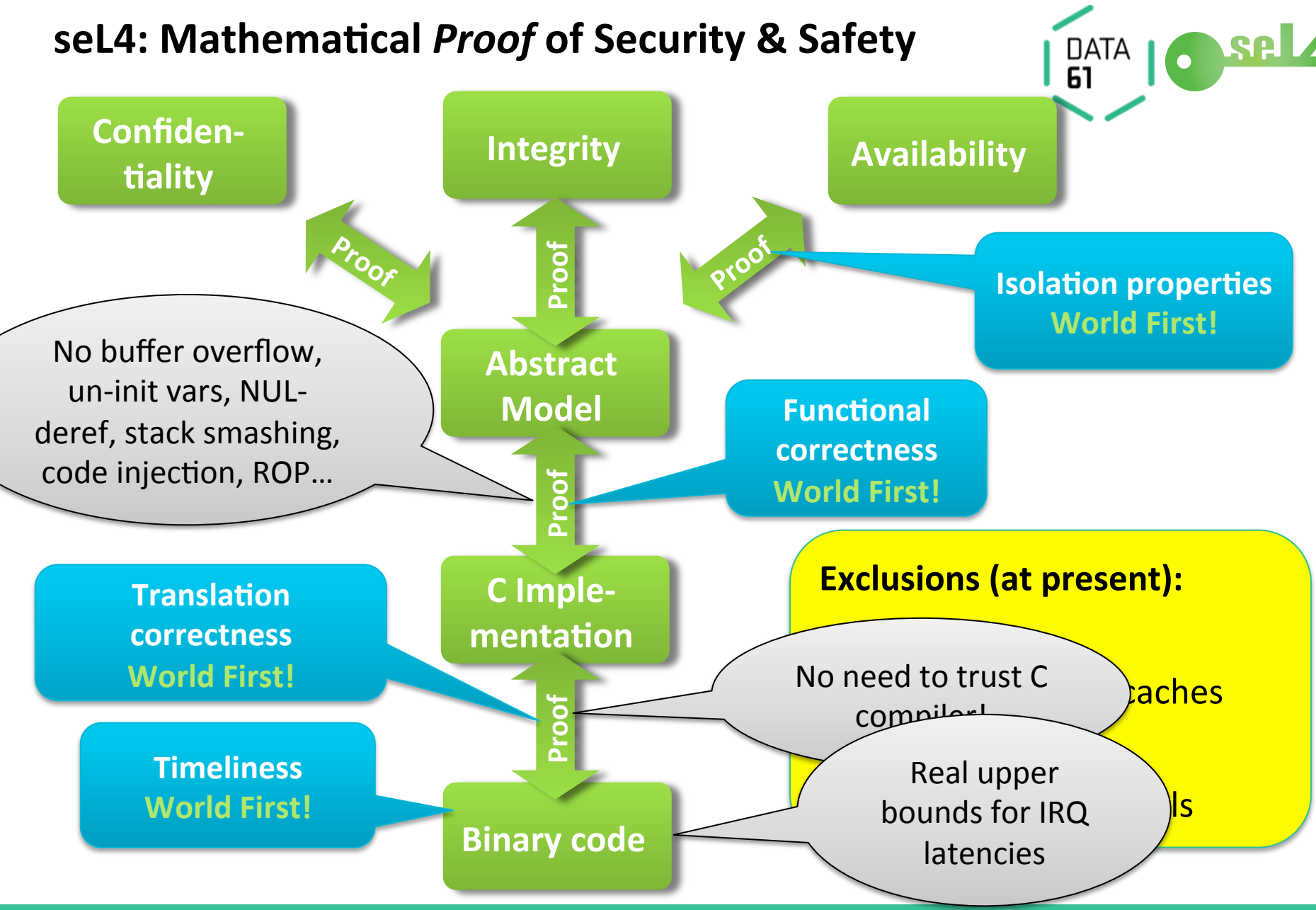
Philosophy Underlying seL4

1. Security is paramount and drives design

2. Security is no excuse for bad performance

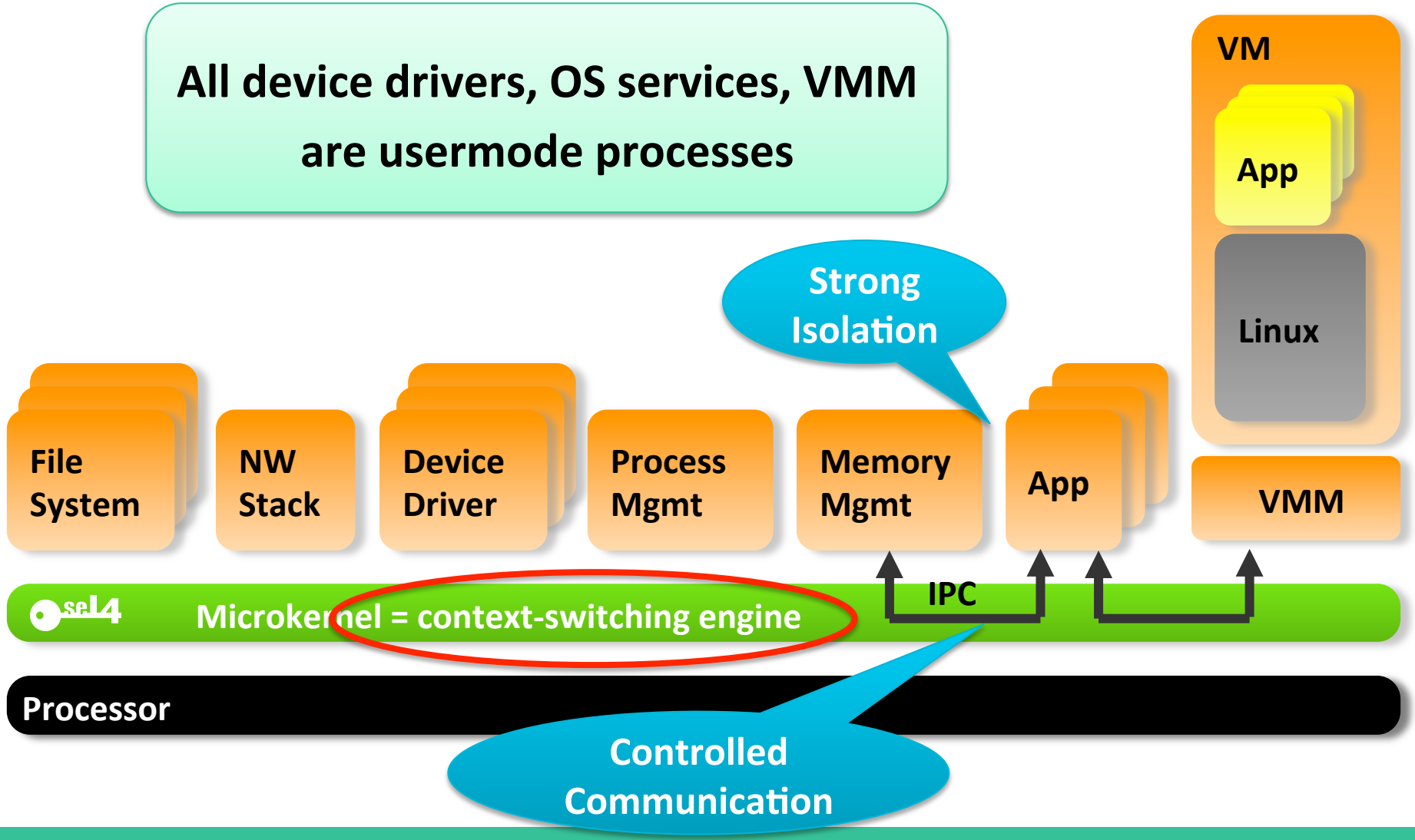
3. General-purpose platform for wide range of use cases

seL4: Mathematical *Proof* of Security & Safety



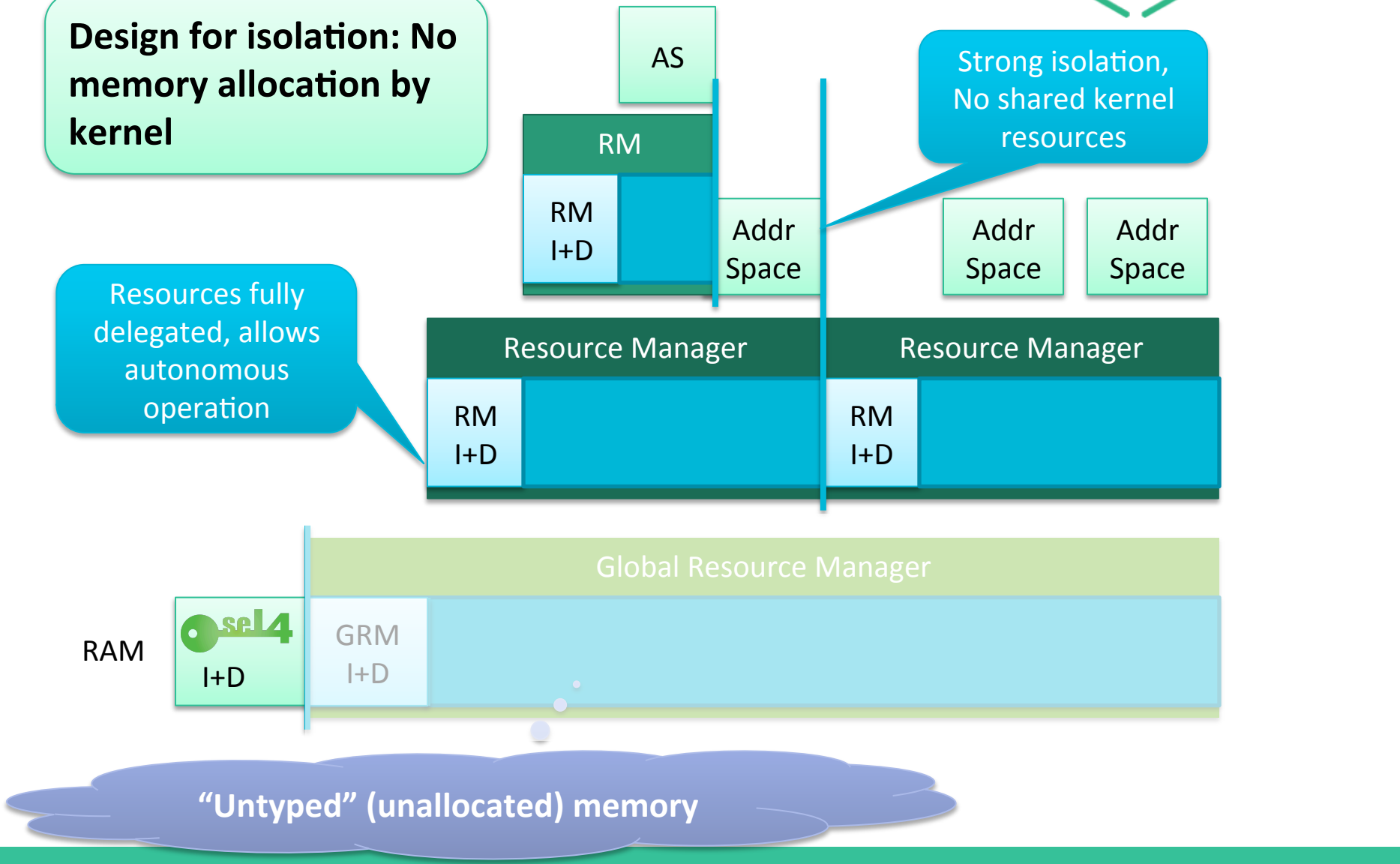
What seL4 Is Not: An Operating System

All device drivers, OS services, VMM are usermode processes

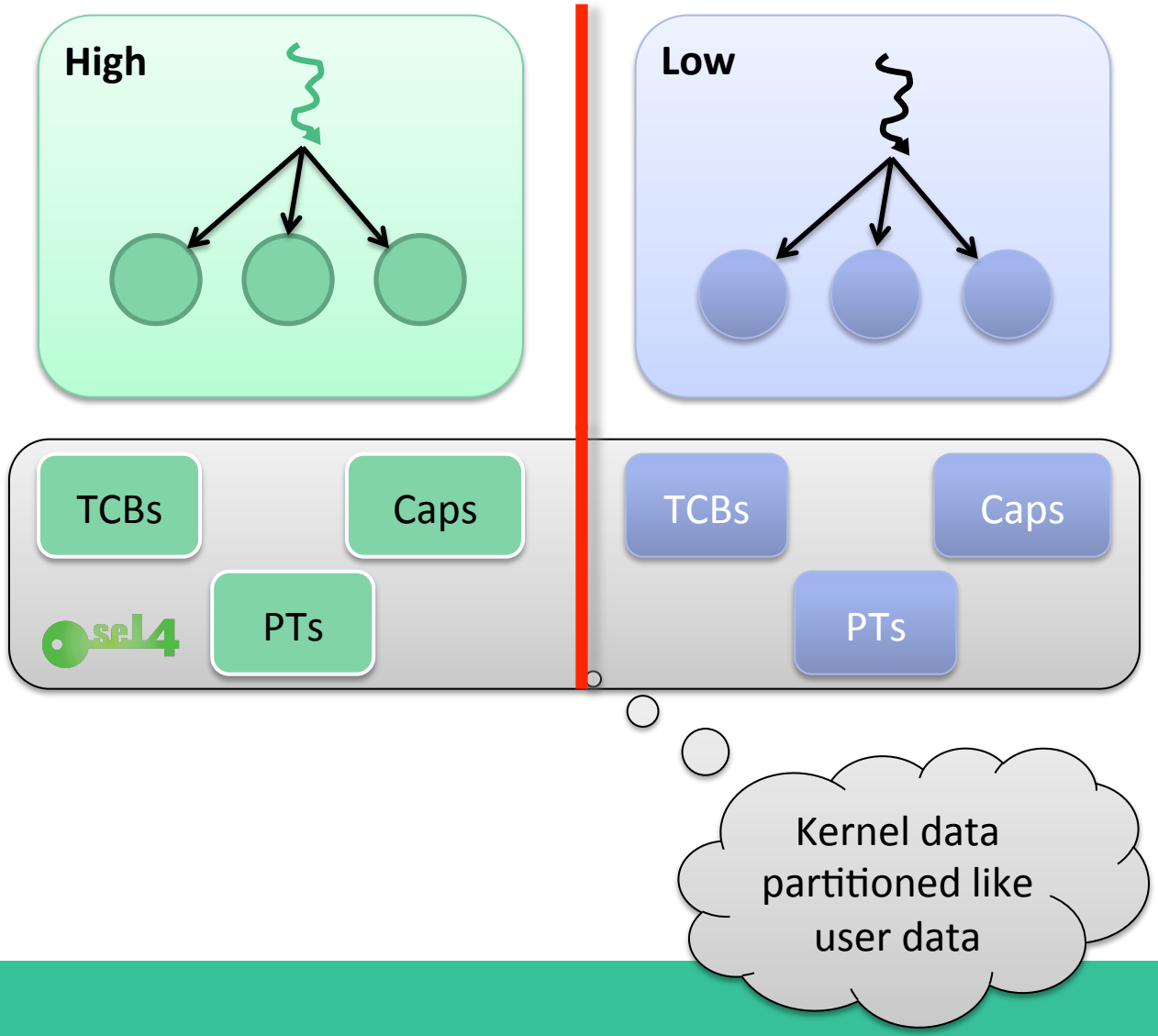


What's Different to Other Microkernels?

Design for isolation: No memory allocation by kernel

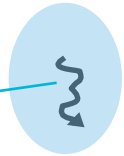
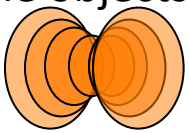
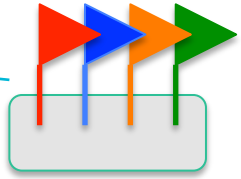
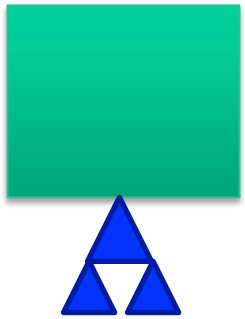


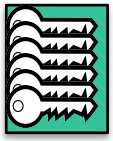
seL4 Isolation Goes Deep



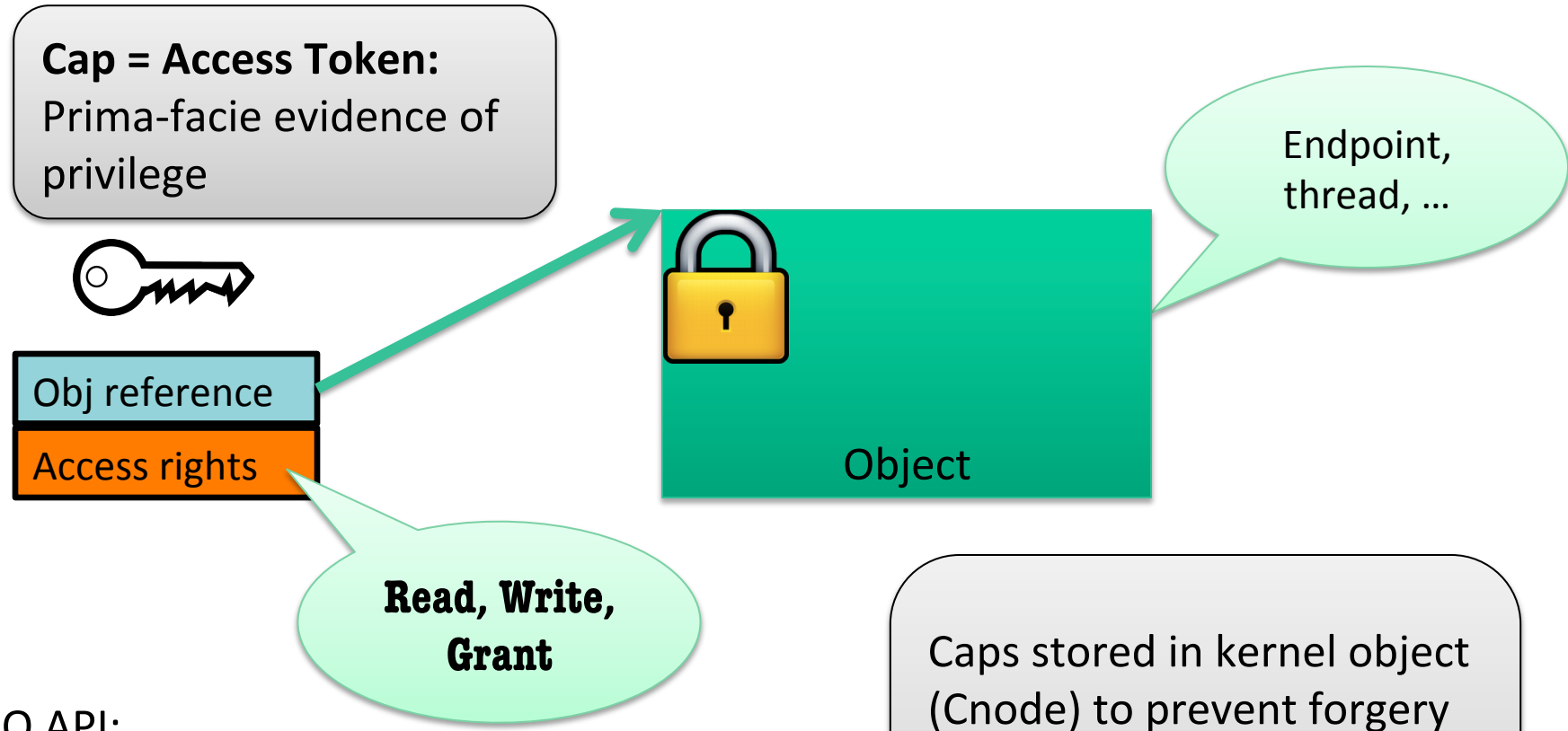
seL4 Concepts

- Capabilities (Caps)
  - mediate access
- Kernel objects:
  - Threads (thread-control blocks: TCBs)
  - Address spaces (page table objects: PDs, PTs)
  - Endpoints
  - Notifications
  - Capability spaces (CNodes)
  - Frames
  - Interrupt objects
  - Untyped memory
- System calls
  - Send, Wait (and variants)
  - Yield




Key Mechanism: seL4 Capabilities



- OO API:

  err = method(cap, args);

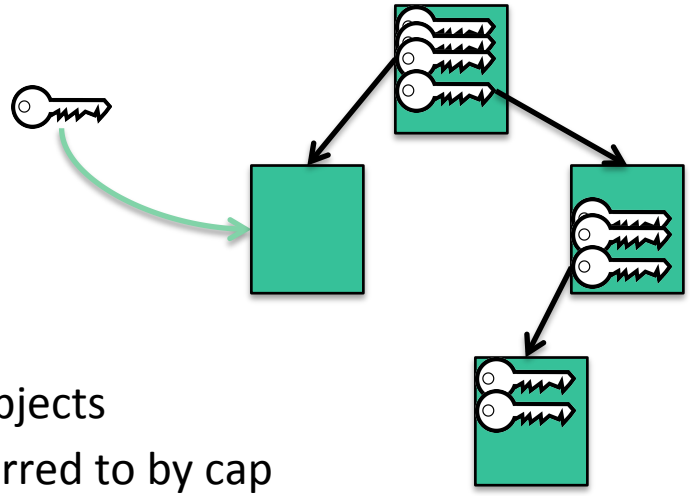
- Used in some earlier microkernels:
  - KeyKOS [‘85], Mach [‘87], EROS [‘99]

Caps stored in kernel object (Cnode) to prevent forgery

➢ user references cap through handle: CPTR

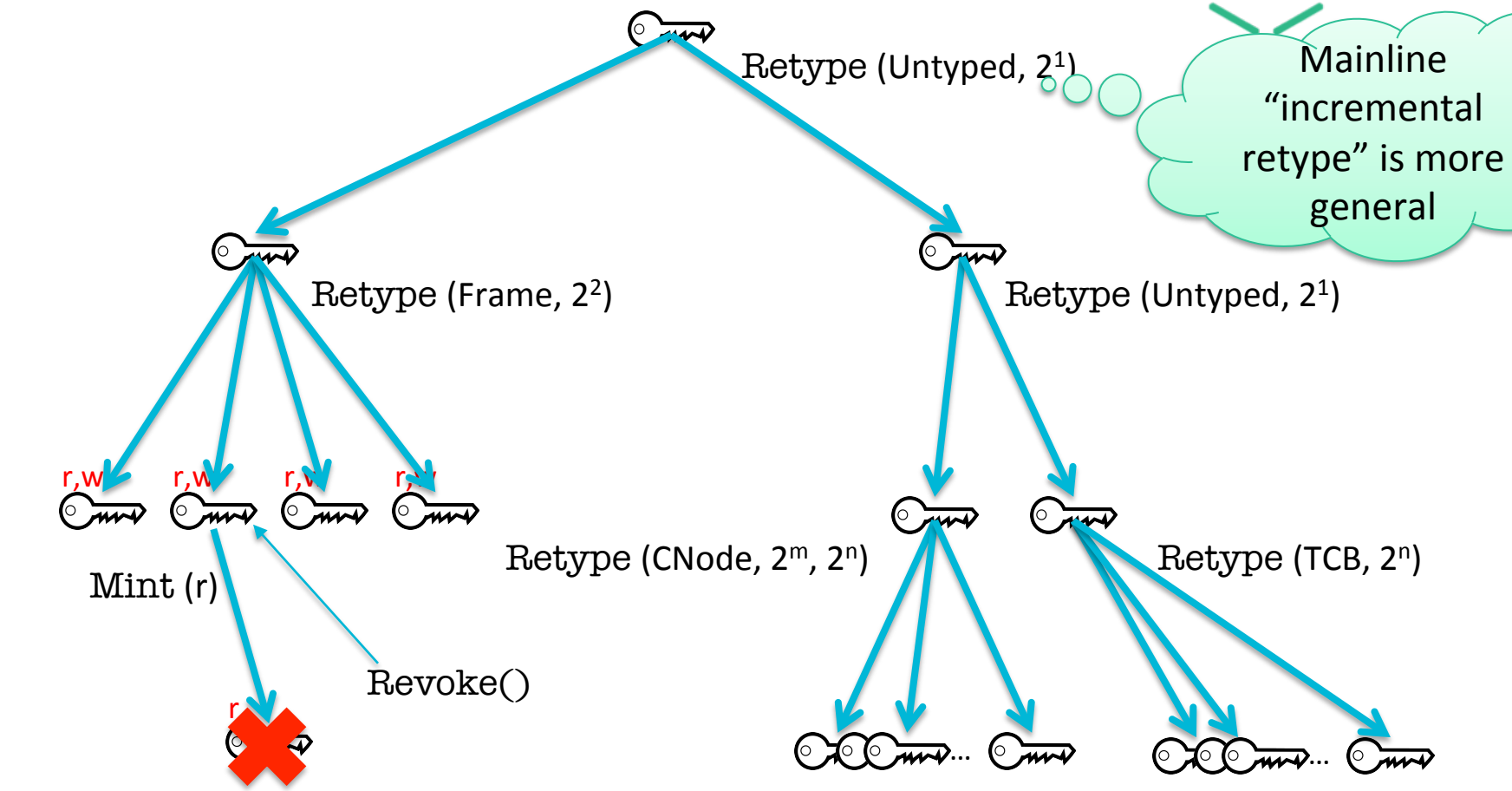
seL4 Capabilities

- Stored in cap space (*CSpace*)
  - Kernel object made up of *CNodes*
  - each an array of cap "slots"

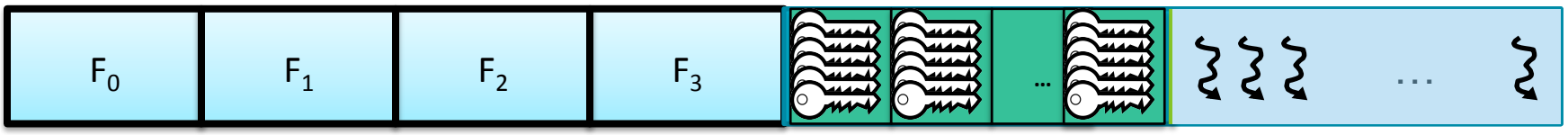


- Main operations on caps:
  - *Retype*: on **Untyped** only, creates other objects
  - *Invoke*: perform operation on object referred to by cap
    - Possible operations depend on object type
  - *Copy*/*Mint*/*Grant*: create copy of cap with *same*/*lesser* privilege
  - *Move*/*Mutate*: transfer to different address with same/lesser privilege
  - *Delete*: invalidate slot
    - Only affects object if last cap is deleted
  - *Revoke*: delete any derived (eg. copied or minted) caps

Memory Management Mechanics: Retype

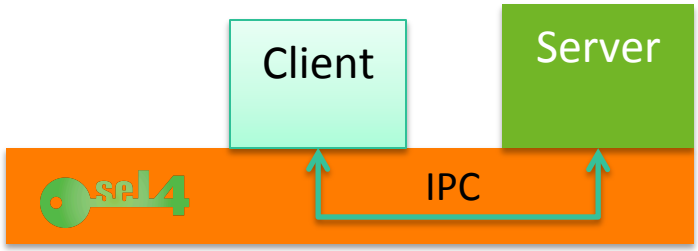


Mainline "incremental retype" is more general

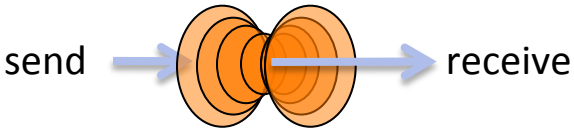


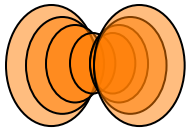
Inter-Process Communication (IPC)

- Fundamental microkernel operation
  - Kernel provides no services, only mechanisms
  - OS services provided by (protected) user-level server processes
  - invoked by IPC

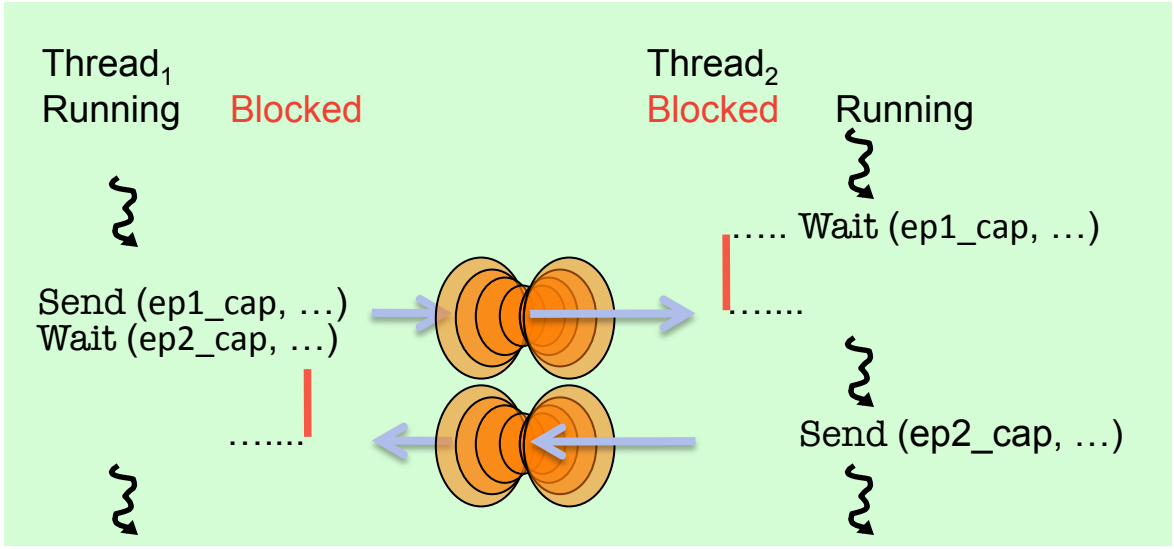


- seL4 IPC uses a handshake through *endpoints*:
  - Transfer points without storage capacity
  - Message must be transferred instantly
    - One partner may have to block
    - Single-copy user → user by kernel



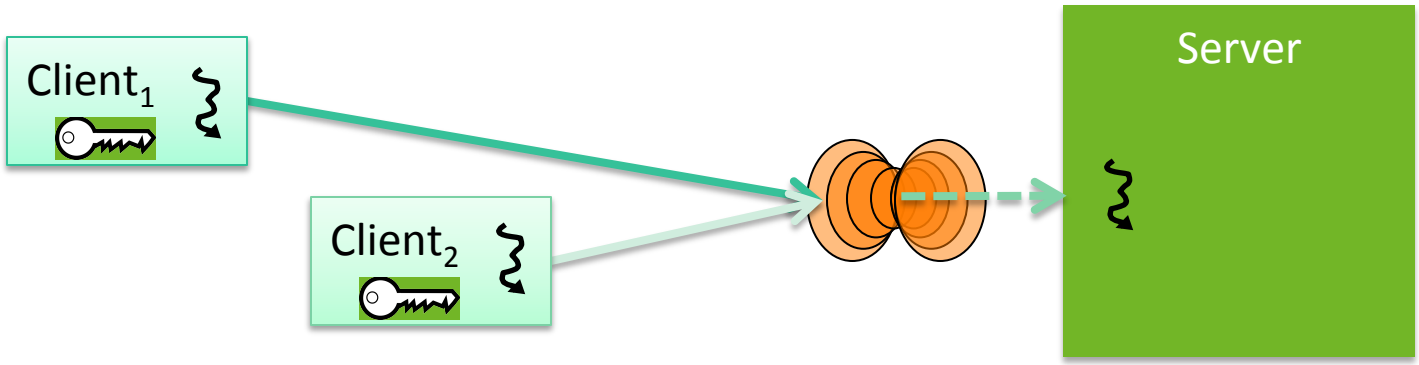


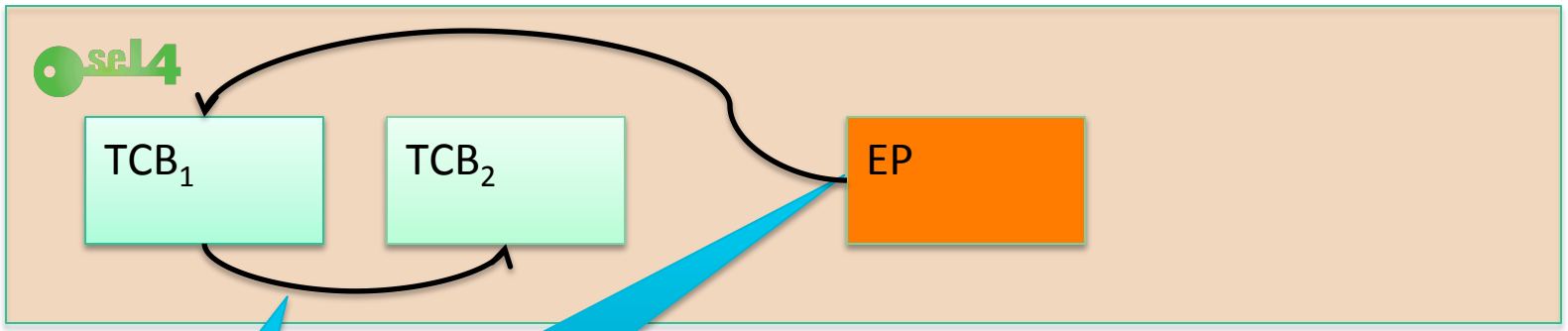
(IPC) Endpoints



- Threads must rendez-vous for message transfer
  - One side blocks until the other is ready
  - Implicit synchronisation
- Message copied from sender's to receiver's *message registers*
  - Message is combination of caps and data words
    - presently max 121 words (484B, incl message "tag")

Endpoints are Message Queues

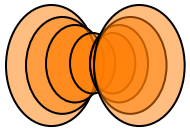




First invocation queues caller

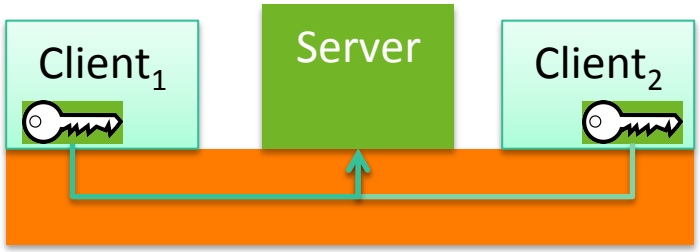
Further callers of same direction queue behind

- EP has no sense of direction
- May queue senders or receivers
  - never both at the same time!
- *Communication needs 2 EPs!*



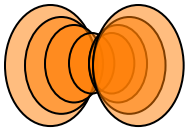
Client-Server Communication

- Asymmetric relationship:
  - Server widely accessible, clients not
  - How can server reply back to client (distinguish between them)?

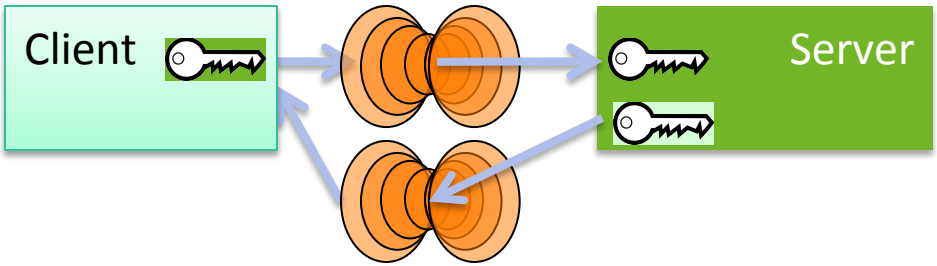


- Client can pass (session) reply cap in first request
  - server needs to maintain session state
  - forces stateful server design

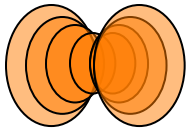
- seL4 solution: Kernel provides single-use *reply cap*
  - only for `Call` operation (`Send+Wait`)
  - allows server to reply to client
  - cannot be copied/minted/re-used but can be moved
  - one-shot (automatically destroyed after first use)



Call RPC Semantics



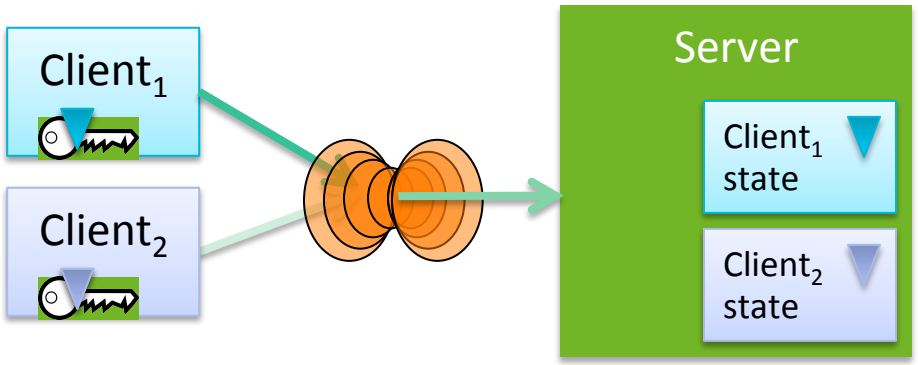
Client	Kernel	Server
		Wait(ep,&rep)
Call(ep,...)		
	mint rep	
	deliver to server	
		process
		Send(rep,...)
	deliver to client	
	destroy rep	
process		*process*



Identifying Clients

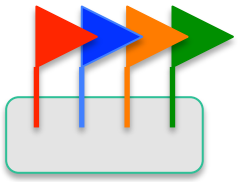
Stateful server serving multiple clients

- Must respond to correct client
  - Ensured by reply cap
- Must associate request with correct state

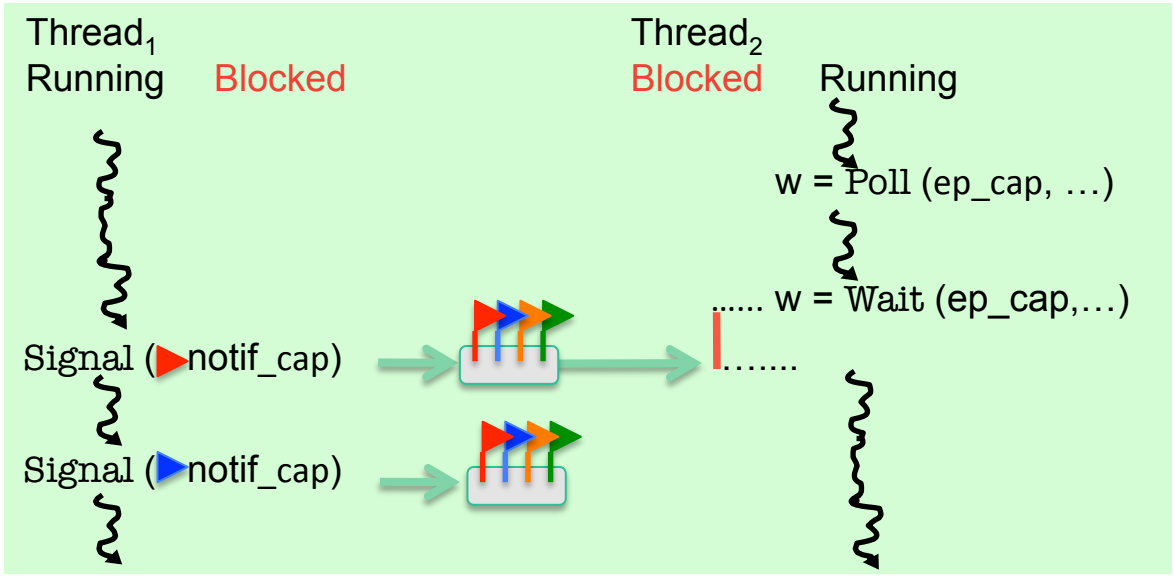


- Could use separate EP per client
  - endpoints are lightweight (16 B), but creating/deleting costs
  - also requires mechanism to wait on a set of EPs (like select)

- Instead, seL4 allows to individually mark ("badge") caps to same EP
  - server provides individually badged caps to clients
  - server tags client state with badge (through `Mint()`)
  - kernel delivers badge to receiver on invocation of badged caps



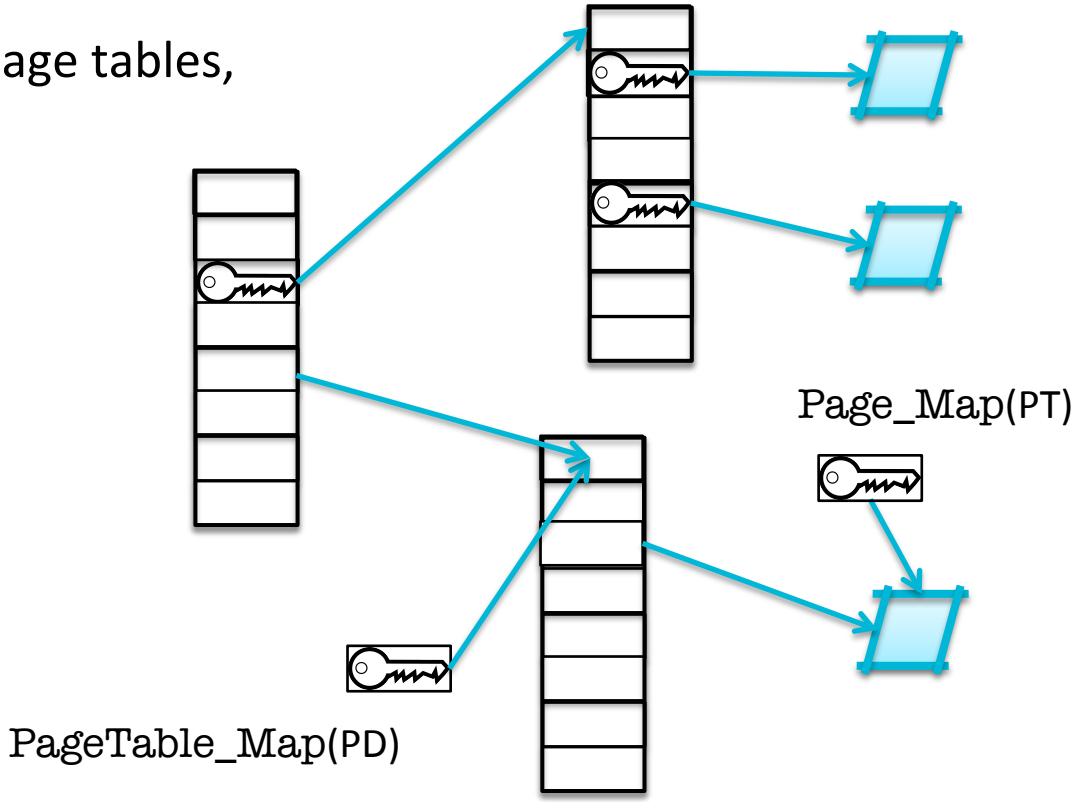
Notifications

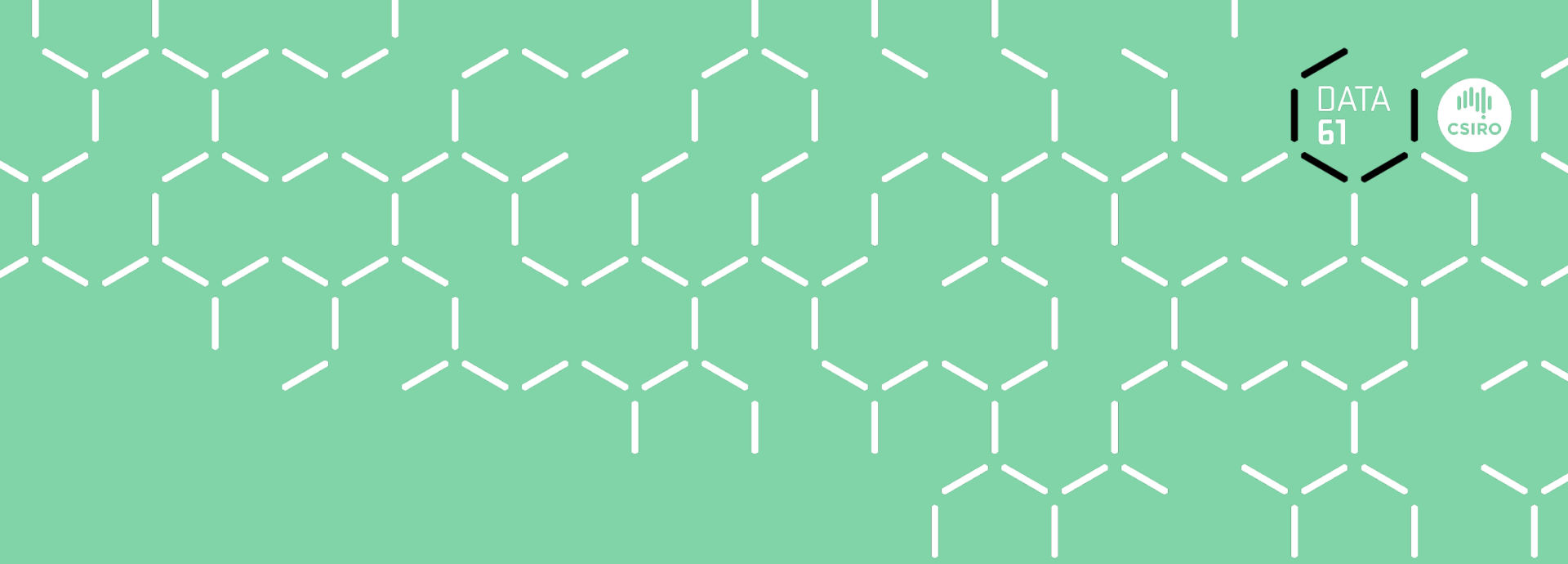


- Notification is an array of binary semaphores
  - Multiple signalling, select-like wait
  - Not a message-passing IPC operation!
- Implemented by *data word* in object
  - Signal operation OR-s sender's *cap badge* to data word
  - Receiver can poll or wait
    - waiting returns and clears data word
    - polling just returns data word

seL4 Address Spaces (VSpaces)

- Very thin wrapper around hardware page tables
  - Architecture-dependent
  - ARM and (32-bit) x86 are very similar
- Page directories (PDs) map page tables, page tables (PTs) map pages
- A VSpace is represented by a PD object:
  - Creating a PD (by Retype) creates the VSpace
  - Deleting the PD deletes the VSpace





seL4 Roadmap

Presently Released

- Verified implementation
  - seL4.ARMv6 Sabre
  - Properties:
    - Execution safety: to binary
    - Functional correctness: to binary
    - Integrity: to binary
    - Confidentiality (excluding timing channels): to binary
    - User-level system initialization: on model

- Unverified implementations:
  - seL4/ARMv7: various boards, incl. Zynq
  - seL4/ARMv7a: A15 virtualisation support (branch)
  - seL4/x86: VT-x, VT-d (**experimental** branch)
  - seL4 x86 and ARM: new RT scheduling model (**RT** branch)

Roadmap: Kernel Features Release (Subject to Change)

Development Roadmap

Q2'16: seL4/ARMv8 32-bit (master)

Q2'16: seL4 virtualisation support on ARM (master)

Q2'16: seL4 real-time & mixed-criticality scheduling [API change] (branch)

Q3'16: seL4/x64 implementation (master)

Q3'16: seL4 virtualization support on x86 (branch)

Q4'16: seL4 strict temporal partitioning [potential API change] (branch)

Q4'16: seL4 for multicore ARM and x86 (branch)

Q2'17: seL4/ARMv8 64-bit (master)

Q2'17: seL4 for multicore ARM-64 and x64 (branch)

Q3'16: CAmkES support for new real-time model (master)

Q2'17: full SMP VMM

Roadmap: Mainline Kernel (Subject to Change)

Verification Roadmap:

Q2'16 Generation of CAmkES RPC glue-code and capDL proofs

Q4'16: libsel4 verification

Q1'17: verified ARM virtualisation support

Q1'17: verified seL4/x64

Q2'18: verified RT kernel

???: verified multicore kernel

Summary

1. seL4 is the world's most demonstrably secure OS kernel
2. Security is no excuse for bad performance!

http://seL4.systems

Gernot.heiser@data61.csiro.au

http://microkerneldude.wordpress.com

@GernotHeiser