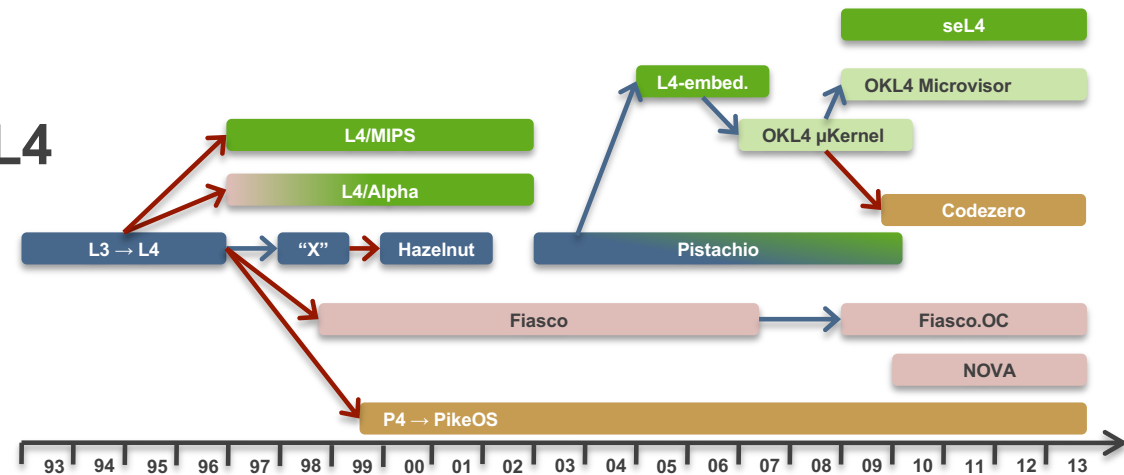


2026 T3 Week 01 Part 1

Introduction: Microkernels and seL4 @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>

Why Advanced Operating Systems?

- Understand OS (especially microkernels) in real depth
- Understand how to design an OS
- Learn to build a sizable system with great deal of independence
- Learn to cope with the complexity of systems code
- Tackle a real challenge
- Get a glimpse of OS research, and preparation for it
- Obtain skills highly sought-after in industry
- **Have fun while working hard!**

Why NOT Advanced Operating Systems?

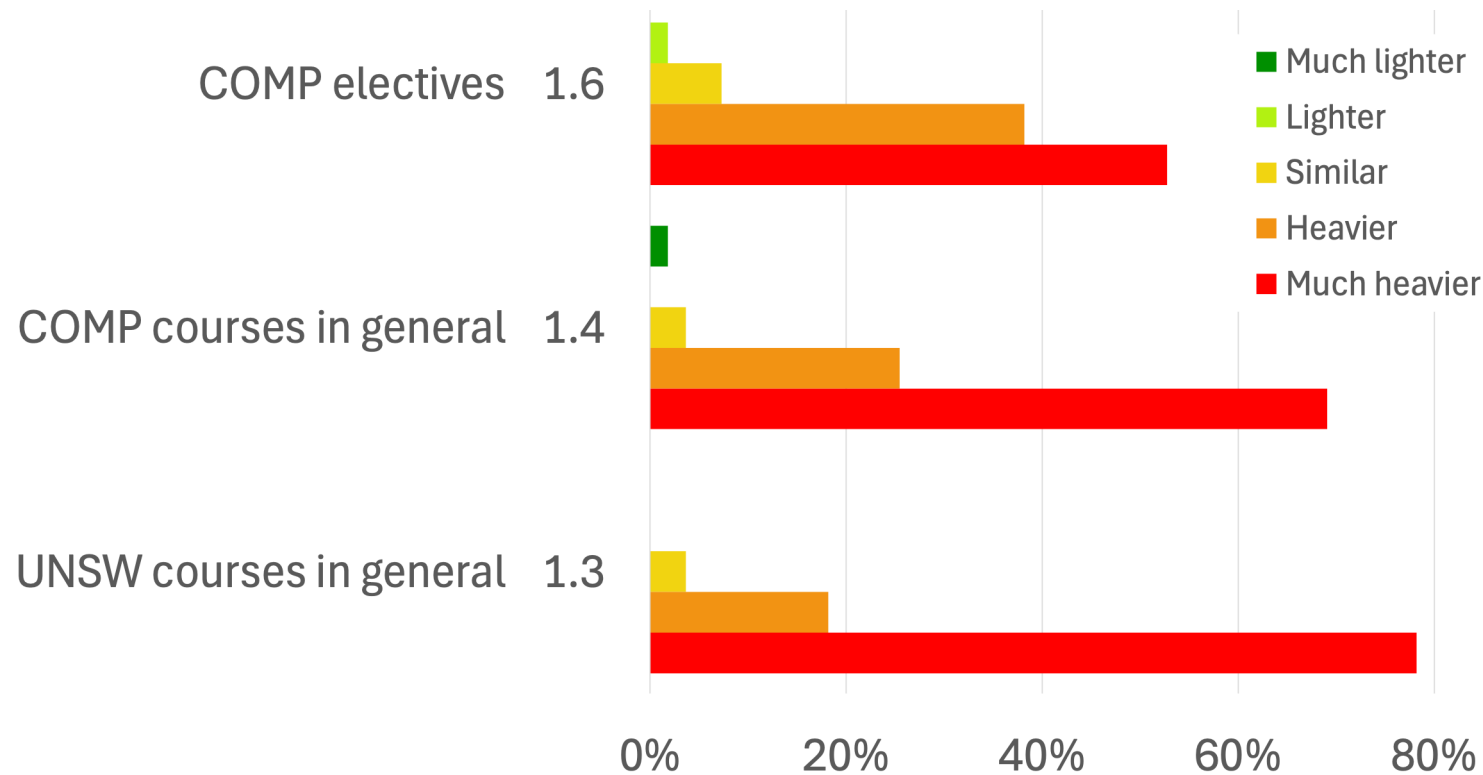
- Easy HD
- AI will help me pass and get a good job

Startup Checklist (see course page)

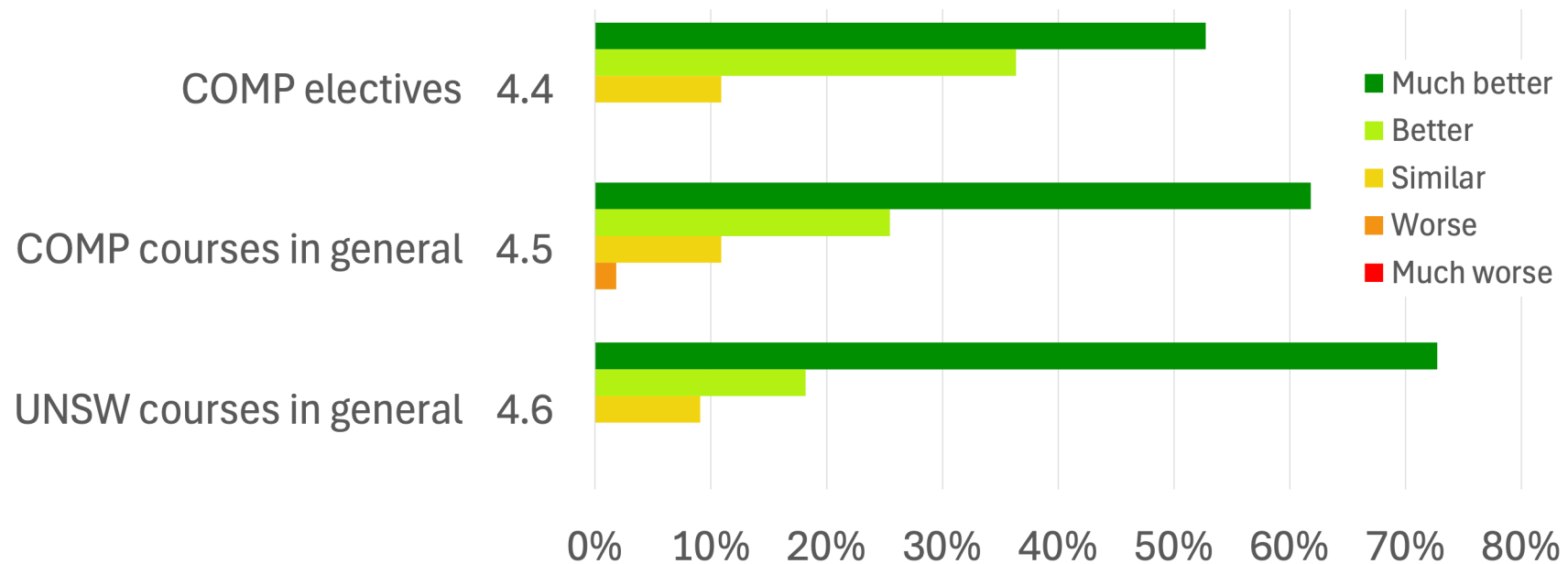
The following you should complete ASAP (if you haven't already):

- Check the [course surveys](#) and lecturer response, at least for the most recent offering, to confirm that this course is for you!

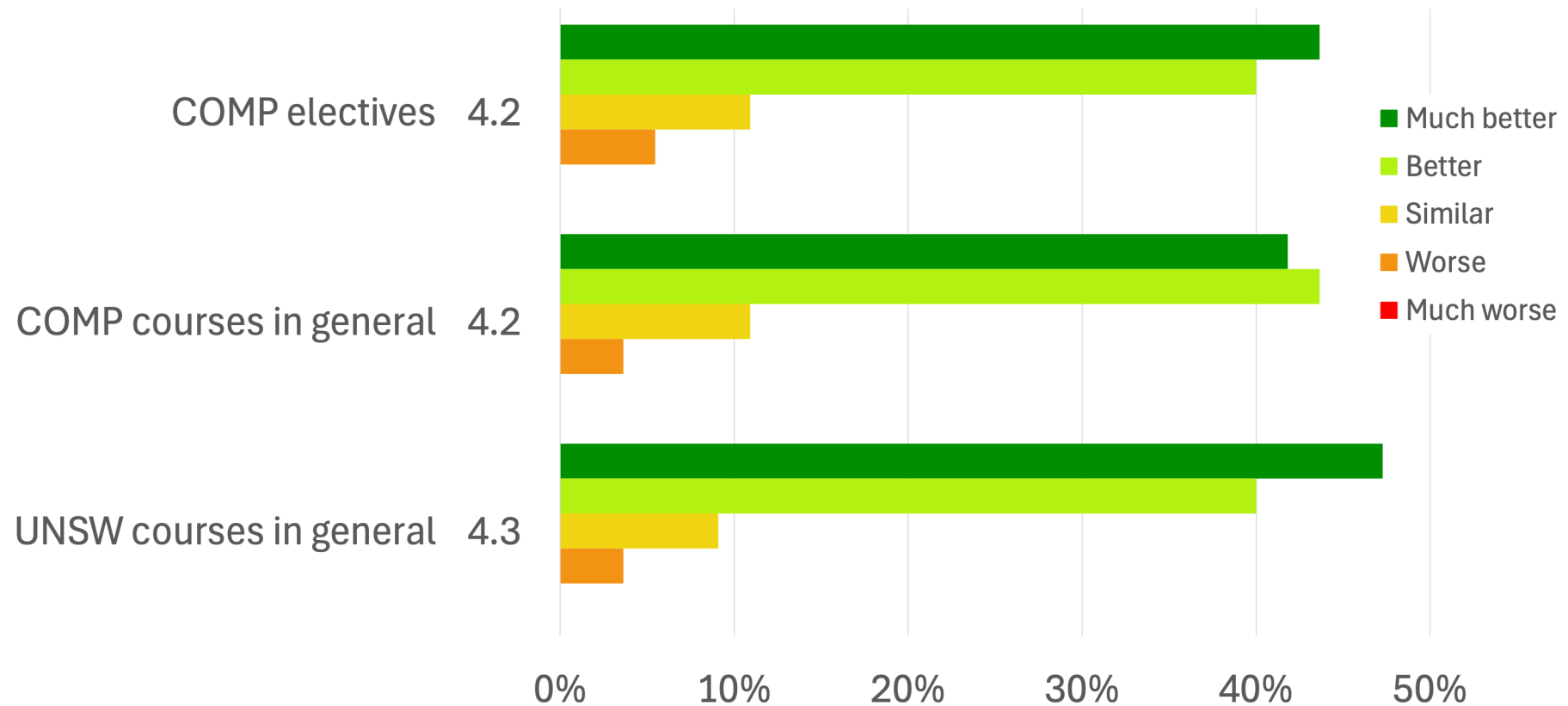
Survey Q8: How does the workload compare?



Survey Q10: How does quality/value compare?



Survey Q9: How does your experience compare?



Startup Checklist (see course page)

The following you should complete ASAP (if you haven't already):

- Check the [course surveys](#) and lecturer response, at least for the most recent offering, to confirm that this course is for you!
- If you have **not** taken COMP3891/9283 Extended Operating Systems (EOS), you should by now have familiarised yourself with the lecture on Microkernels ([2026T2 Week 9](#))
- You will also be expected to be familiar with the EOS lecture on Object Capabilities (2026T2 Week 8)
- Find a partner! While Milestone 0 is to be done individually, partnering up early is highly recommended.

Things to Note (see course page)

No vibe coding!

A core objective of this course is to teach you to think for yourself. Using an AI tool to regurgitate other people's thoughts is contrary to this objective. We will have zero tolerance for vibe coding. If you're unable or unwilling to understand the problem thoroughly and code the solution yourself, you're in the wrong course.

Things to Note (see course page)

No vibe coding!

Demo early!

- If the demonstrator finds your solution insufficient, you'll be sent back (aka "show-stopped"). If this happens on the last lab on Friday, you're guaranteed a late penalty. If you do your demo earlier, you have a chance to try again without penalty!
- Also, show up early for the lab, we use a FCFS ticketing system. If you show up 10' before the end, you may not be able to demo, and this will be your own fault.
- Note that we monitor lab usage and add or cancel labs based on demand.

Things to Note (see course page)

No vibe coding!

Demo early!

New exam format!

Due to AI cheating, the established exam format we used for years no longer has any integrity. Instead, the exam will be invigilated, you'll have to answer questions about a research paper you'll be given 24h before.

All supplementary exams will be oral.

Today's Lecture

- Whirlwind refresher of microkernels and intro context of seL4
- seL4 principles and concepts
- seL4 Mechanisms
 - PPC
 - Notifications

Aim: Get you ready for the project quickly

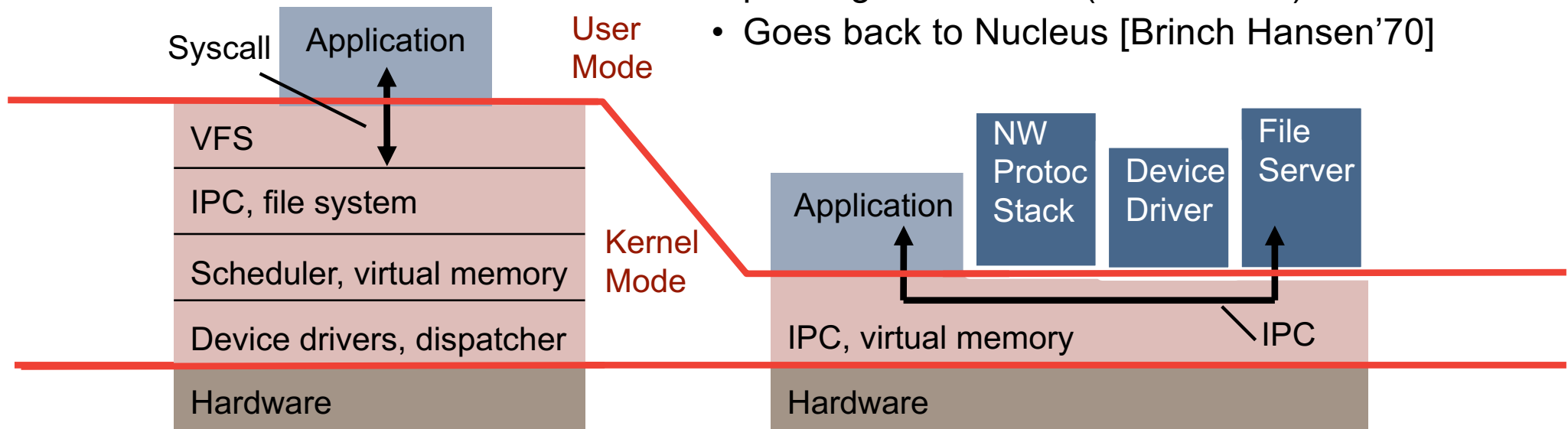
Microkernels

Microkernels: Reducing the Trusted Computing Base

Very quick summary, check EOS lecture on object capabilities!

IPC performance is critical!

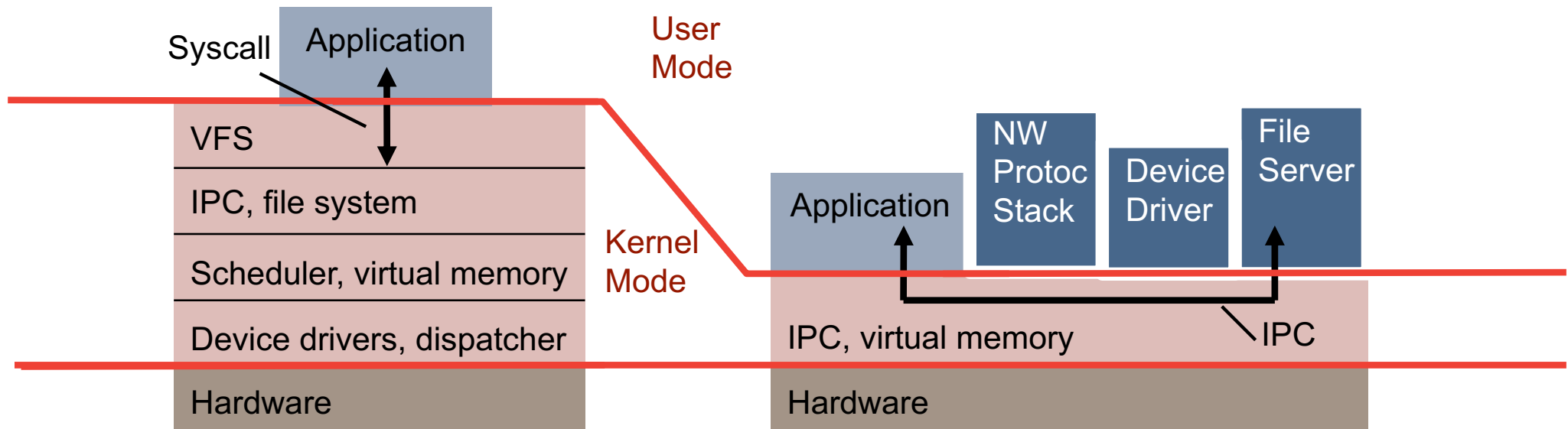
- Idea of microkernel:
 - Flexible, minimal platform
 - **Mechanisms, not policies**
 - OS functionality provided by usermode servers
 - Servers invoked by kernel-provided message-passing mechanism (called “IPC”)
 - Goes back to Nucleus [Brinch Hansen’70]



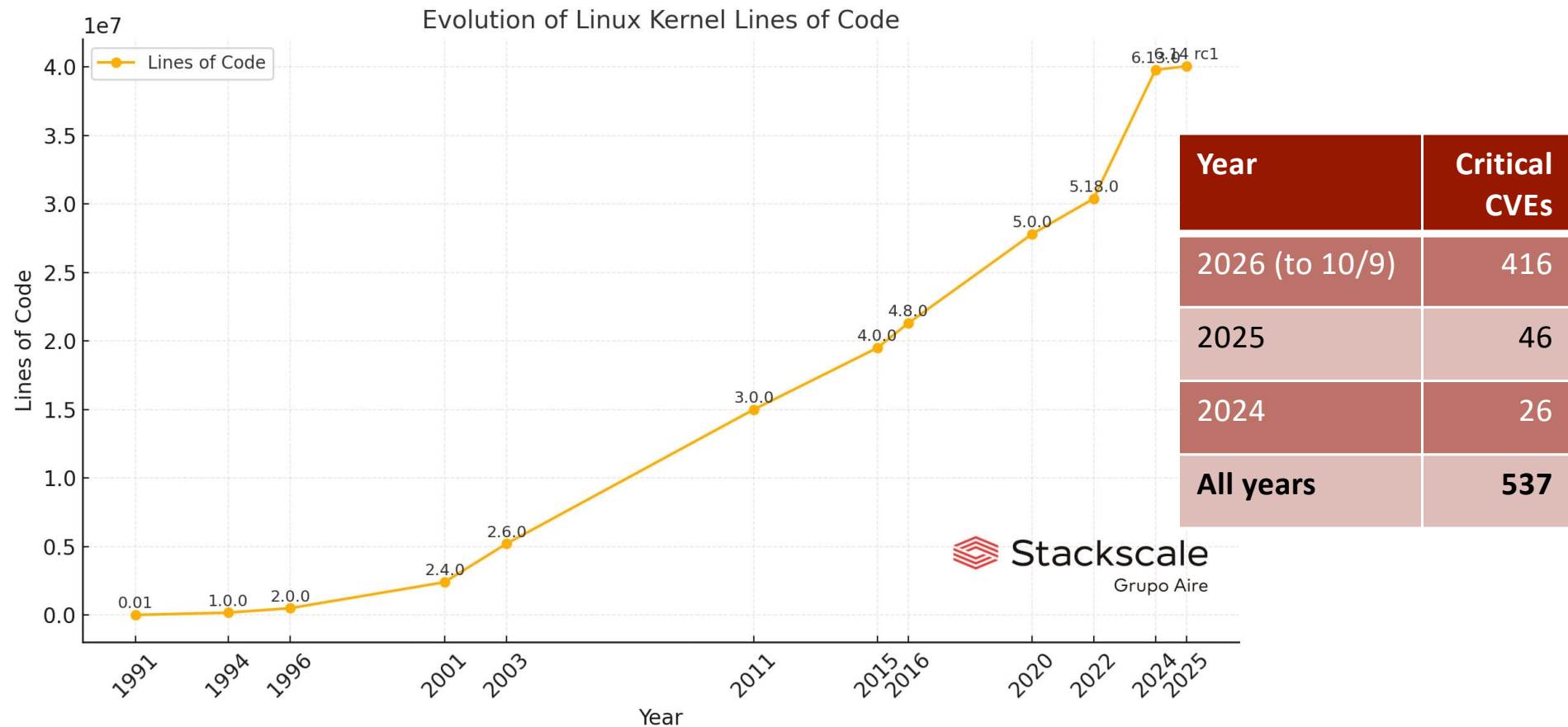
Monolithic vs Microkernel OS Evolution

Monolithic OS

- New features add kernel code
- New policies add kernel code
- Kernel complexity grows



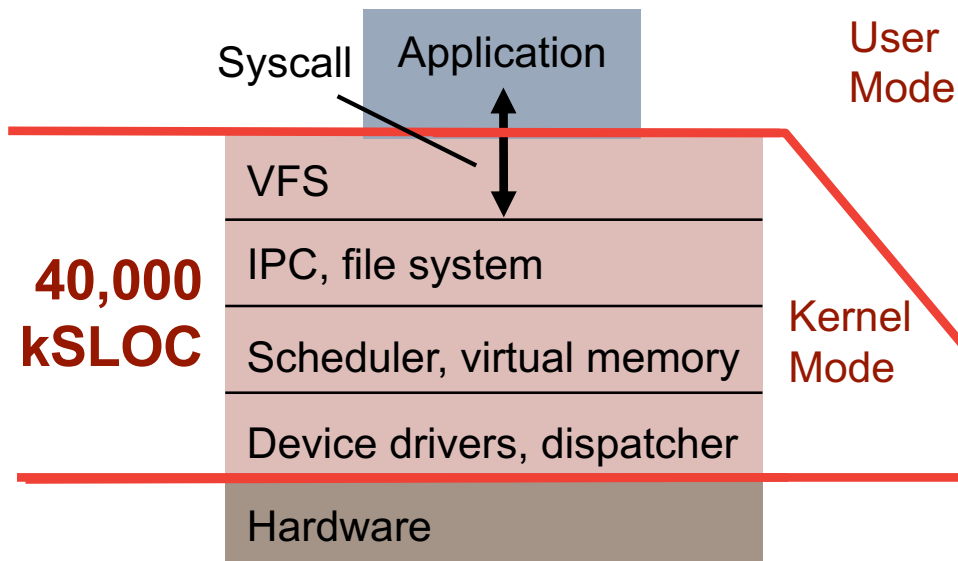
Example Linux



Monolithic vs Microkernel OS Evolution

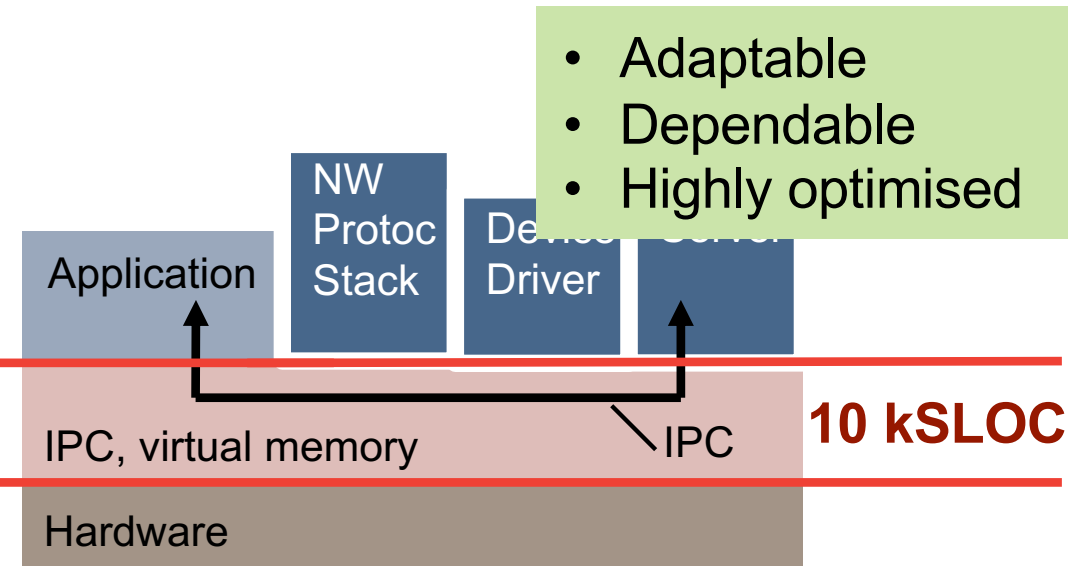
Monolithic OS

- New features add kernel code
- New policies add kernel code
- Kernel complexity grows



Microkernel OS

- Features add usermode code
- Policies replace usermode code
- Kernel complexity is stable



Microkernel Principle: Minimality

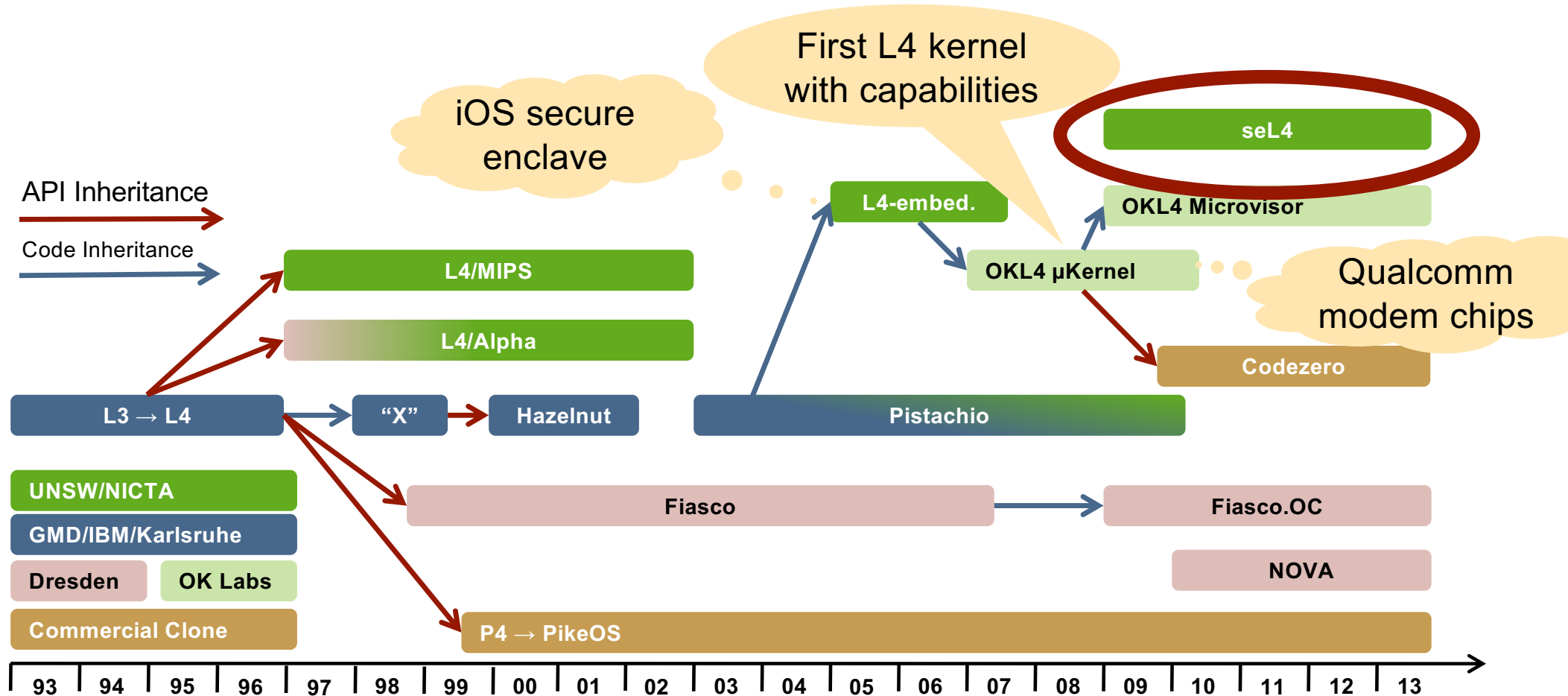


A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e. permitting competing implementations, would prevent the implementation of the system's required functionality. [Liedtke SOSP'95]

- Small *trusted computing base*
 - Easier to get right
 - Small attack surface
- Challenges:
 - API design: **generality** despite small code base
 - Kernel design and implementation for high performance

Needs policy-freedom!

L4: 30 Years High-Performance Microkernels



The seL4 Microkernel

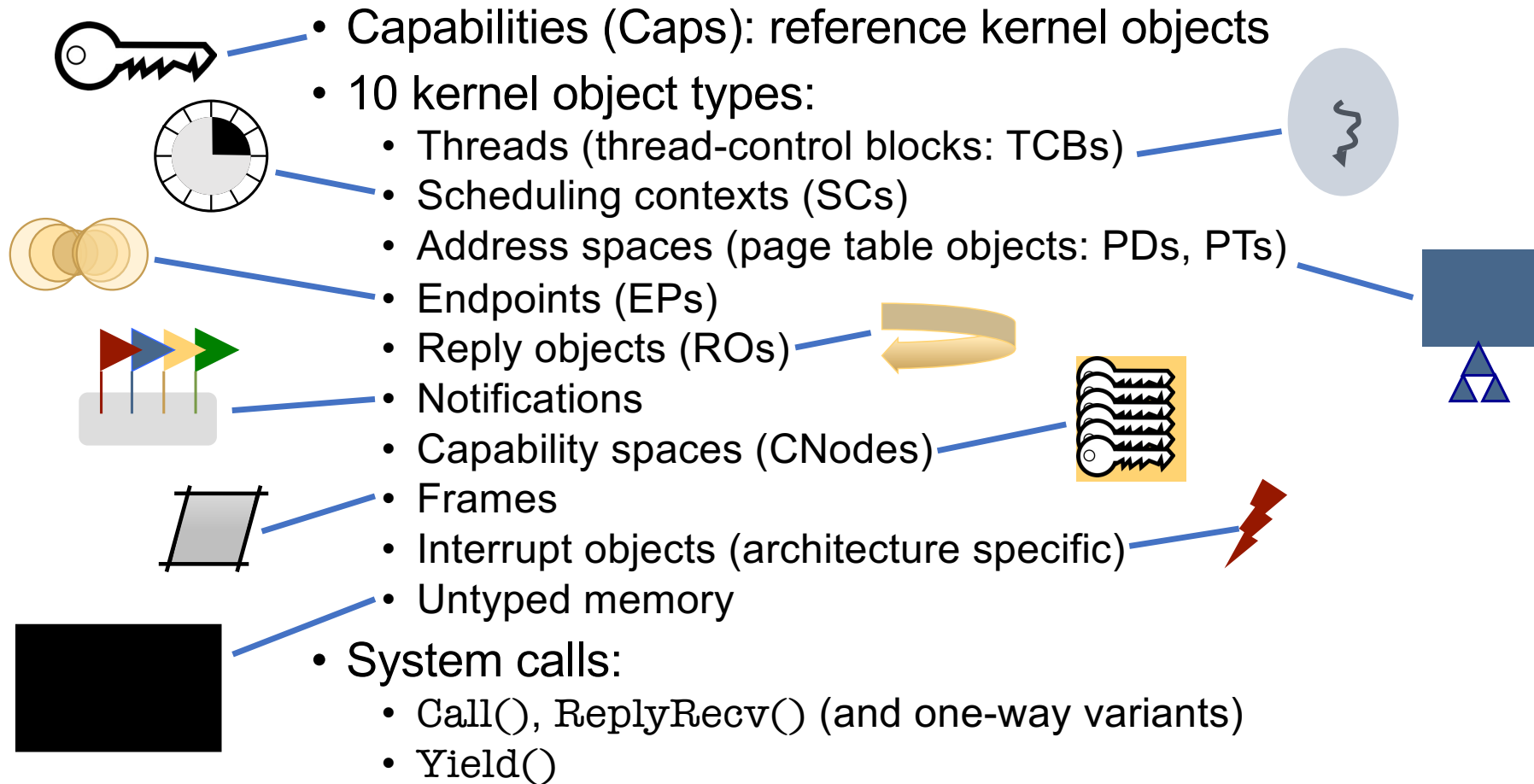
seL4 Principles

- Single protection mechanism: capabilities
 - Now also for time: MCS configuration [Lyons et al, EuroSys'18]
- All resource-management policy at user level
 - Painful to use
 - Need to provide memory-management library for COMP9242
 - Results in L4-like programming model
- Suitable for formal verification
 - Proof of implementation correctness
 - Attempted since '70s
 - Finally achieved by L4.verified project at NICTA/UNSW [Klein et al, SOSP'09]

More on principles in my blog: <https://bit.ly/34uI8FI>



seL4 Concepts in a Slide



seL4 Not a Concept: Hardware Abstraction

Why?

- Hardware abstraction *violates minimality*
- Hardware abstraction *introduces policy*

Limits
generality!

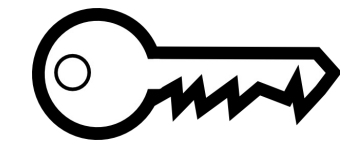
True microkernel:

- Minimal wrapper of hardware, just enough to safely multiplex
- policy-free
- “CPU driver” [Charles Gray]
 - Similarities with Exokernels [Engeler '95]

seL4 What Are (Object) Capabilities?

Capability = Access Token

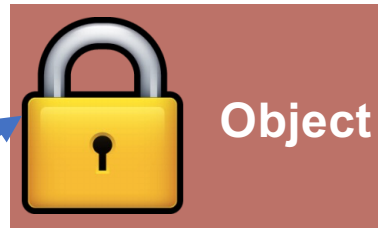
Prima facie evidence of privilege



Obj reference

Access rights

E.g. read, write,
send, execute...



Object

E.g. thread,
address space

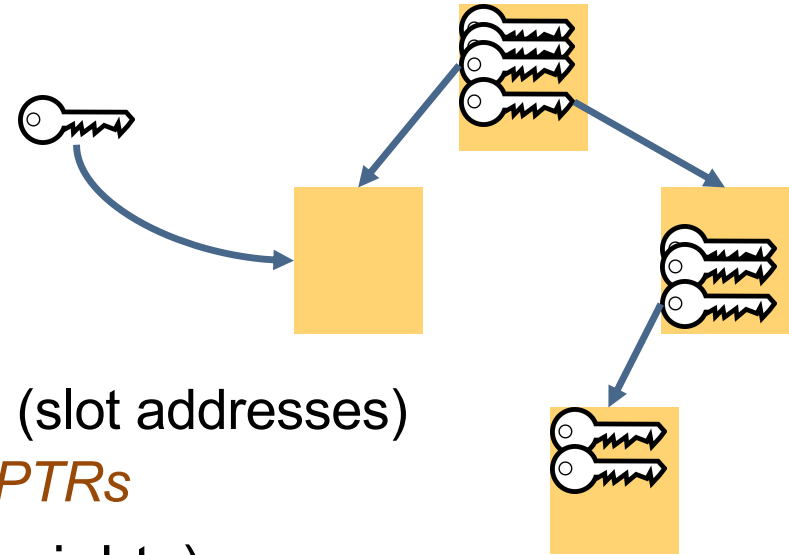
**Fine-grained
access control**

**Very quick summary, check EOS
lecture on object capabilities!**

Any operation is invoking a capability:
`err = cap.method( args );`

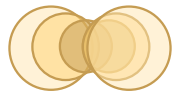
seL4 Capabilities

- Stored in cap space (*CSpace*)
 - Kernel object made up of *CNodes*
 - each an array of cap “slots”
- Inaccessible to userland
 - But referred to by pointers into CSpace (slot addresses)
 - These CSpace addresses are called *CPTRs*
- Caps convey specific privilege (access rights)
 - Read, Write, Execute, GrantReply (Call), Grant (cap transfer)
- Can invoke a cap or derive cap of less or equal strength
 - Details later



seL4 Mechanisms

PPC & Notifications

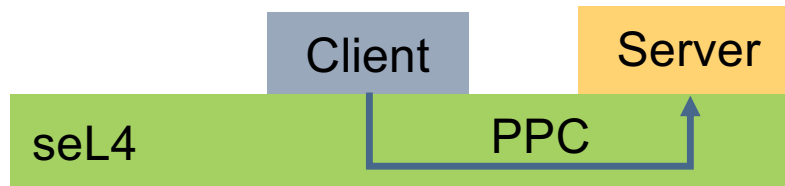


Protected Procedure Calls (PPC)

Fundamental microkernel operation

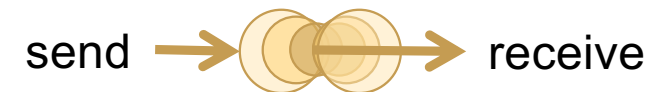
- Kernel provides no services, only mechanisms
- OS services provided by (protected) user-level server processes
- Invoked by *protected procedure call* (PPC)

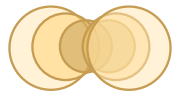
Historically called
“IPC” – bad term!



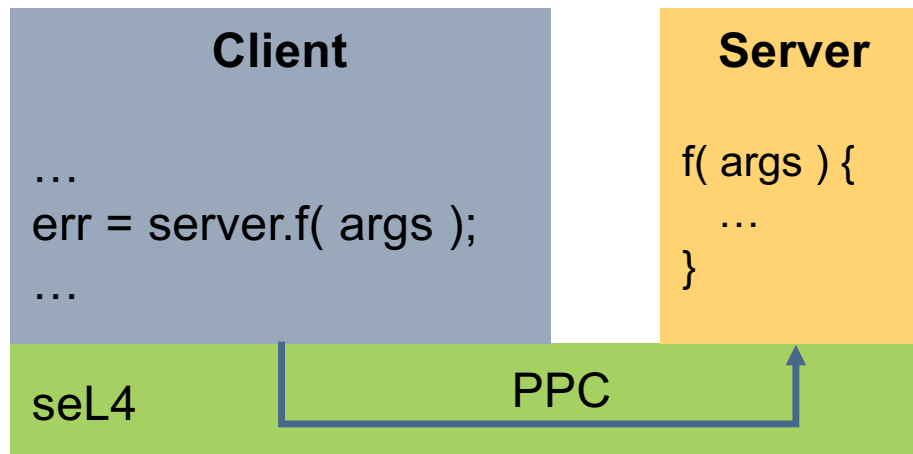
seL4 PPC uses a handshake through *Endpoints*:

- Transfer points without storage capacity
- Arguments must be transferred instantly
 - Single-copy user → user by kernel





seL4 PPC: Cross-Domain Invocation



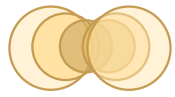
seL4 PPC is: A user-controlled context switch “with benefits”:

- change protection context
- pass arguments / result

Microkernel
equivalent of a
Linux system call!

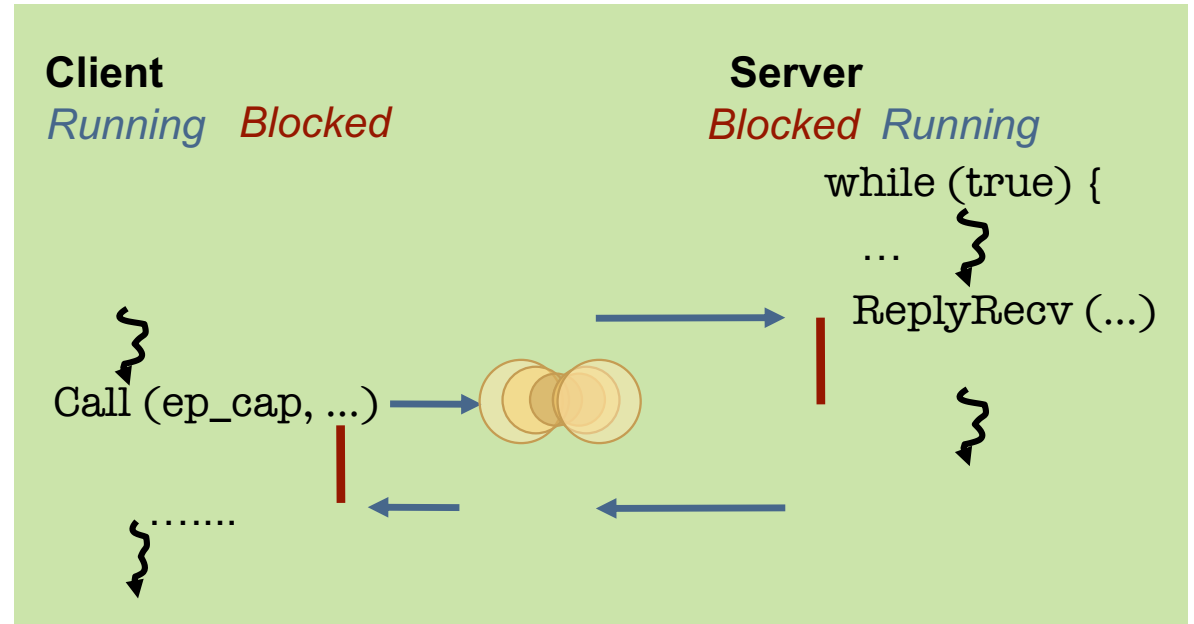
seL4 PPC is **not**:

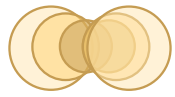
- A mechanism for shipping data
- A synchronisation mechanism
 - side effect, not purpose



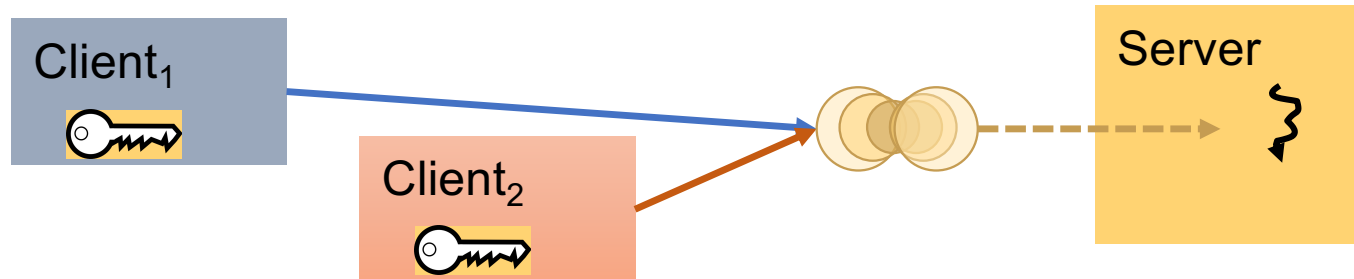
PPC: Endpoints

- Involves 2 threads, but always one blocked
- logically, thread moves between address spaces
- Threads must rendezvous
 - One side blocks until the other is ready
 - Implicit synchronisation
- Arguments copied from sender's to receiver's *message registers*
 - Combination of caps (by-reference arguments) and data words (by-value)
 - Presently max 121 words (484B on 32-bit archs, incl message “tag”, more on 64-bit)
 - Should never use anywhere near that much!

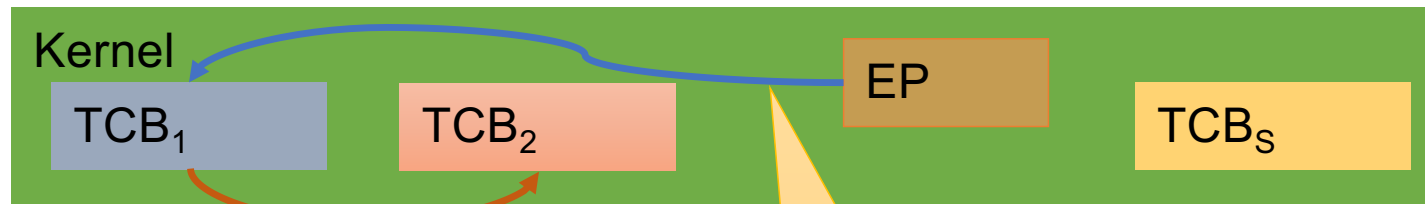




Endpoints are Thread Queues



Note: On single core should not get queues – server should be higher priority!

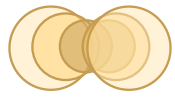


But: Reasonable for single-threaded (“passive”) server on multicore!

Further callers of same direction queue by priority

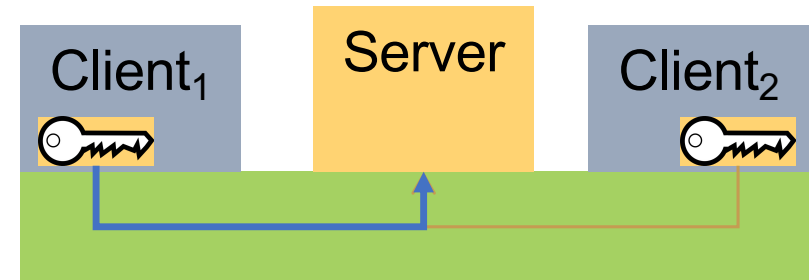
First invocation queues caller

- EP has no sense of direction
- May queue clients or servers
 - never both at the same time!
- *Server invocation needs 2 EPs!*



Server Invocation & Return

- Asymmetric relationship:
 - Server widely accessible, clients not
 - How can server reply back to client (distinguish between them)?
- Client can pass session cap in first request
 - server needs to maintain session state
 - forces stateful server design
- seL4 solution: Kernel creates reply channel in *reply object* (RO)
 - server provides RO in ReplyRecv() operation
 - kernel blocks client on RO when executing receive phase
 - server invokes RO for send phase (only one send until refreshed)
 - only works when client invokes with Call()



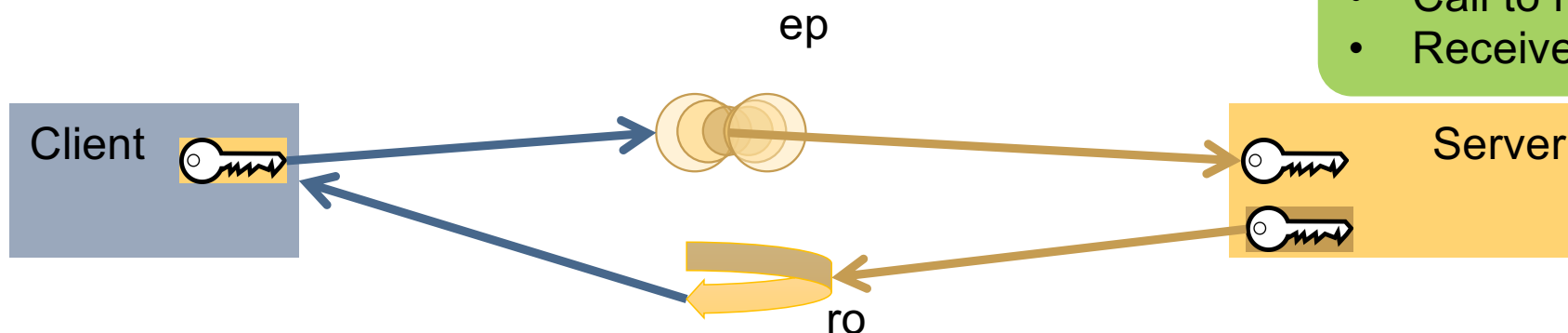
New MCS
kernel
semantics!



Call Semantics

Priorities:

- Call to high
- Receive from low!



Client

Call(ep, args)

Kernel

deliver args to server
block client on RO

Server

ReplyRecv(ep,...,ro)

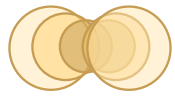
One per client for blocking calls!

process

ReplyRecv(ep,...,ro)

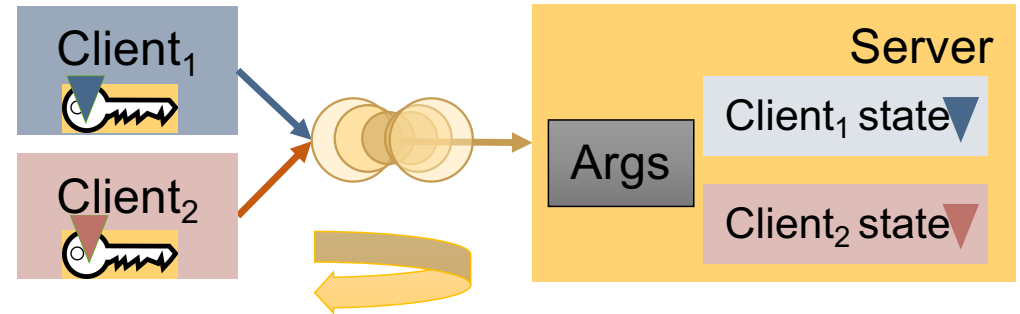
process

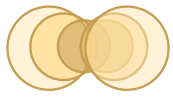
deliver result to client



Stateful Servers: Identifying Clients

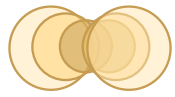
- Server must respond to correct client
 - Ensured by reply cap
- Must associate request with correct state
- Could use separate EP per client
 - endpoints are lightweight (16 B)
 - but would require mechanism to wait on a set of EPs (like Unix `select()`)
- Instead, seL4 allows to individually mark (“badge”) caps to same EP
 - server provides individually badged (session) caps to clients
 - separate endpoints for opening session, further invocations
 - server tags client state with badge
 - kernel delivers badge to receiver on invocation of badged caps





PPC Mechanics: Virtual Registers

- Like physical registers, virtual registers are thread state
 - context-switched by kernel
 - map to physical registers or thread-local memory (“IPC buffer”)
- Message registers
 - contain arguments/results transferred in PPC
 - architecture-dependent subset mapped to physical registers
 - presently 1 on x86, 4 on x64, Arm, RISC-V
 - library interface hides details
 - 1st transferred word is special, contains *message tag*
 - API: MR[0] refers to next word (not the tag!)
 - Use helper functions seL4_SetMR, seL4_GetMR



PPC Operations Summary

- Call (ep_cap, ...)
 - *Atomic*: guarantees caller is ready to receive reply
 - Sets up server's reply object
- ReplyRecv (ep_cap, ...)
 - Invokes RO (non-blocking), waits on EP, re-inits RO
- Recv (ep_cap, ...), Reply(...), Send (ep_cap, ...)
 - For initialisation and exception handling
 - needs Read, Write, Write permission, respectively
- NBSend (ep_cap, ...)
 - Polling send, message lost if receiver not ready

Not really
useful

Need error
handling
protocol !

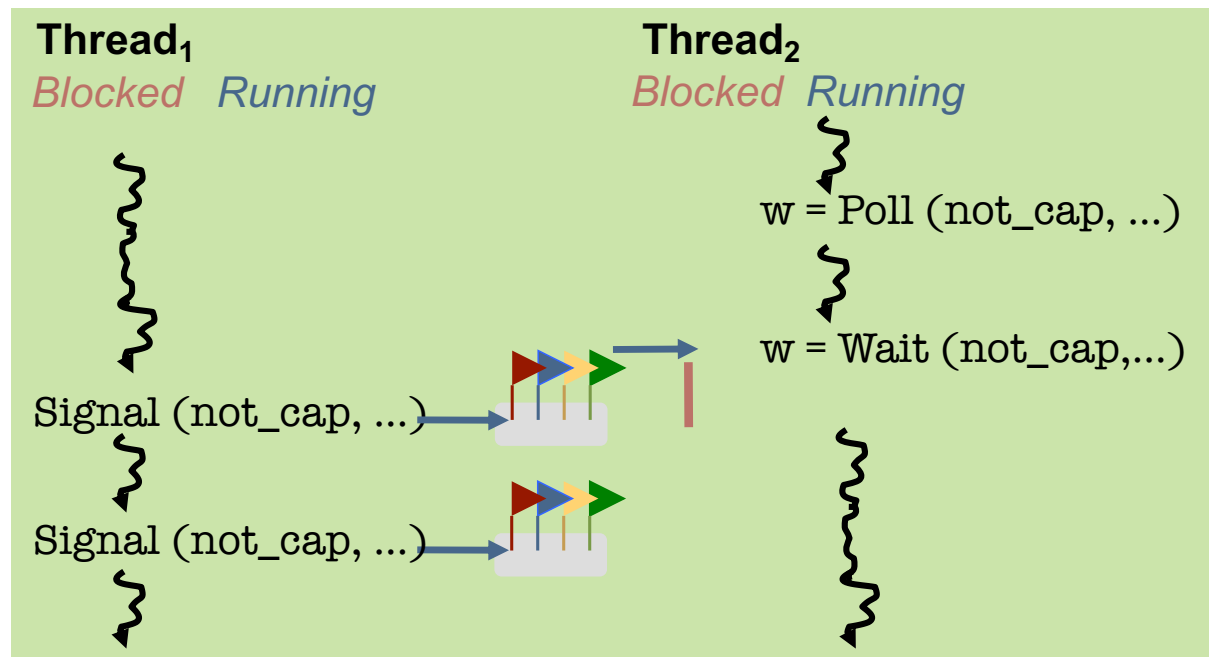
No failure notification where this reveals info on other entities!



Notifications – Synchronisation Objects

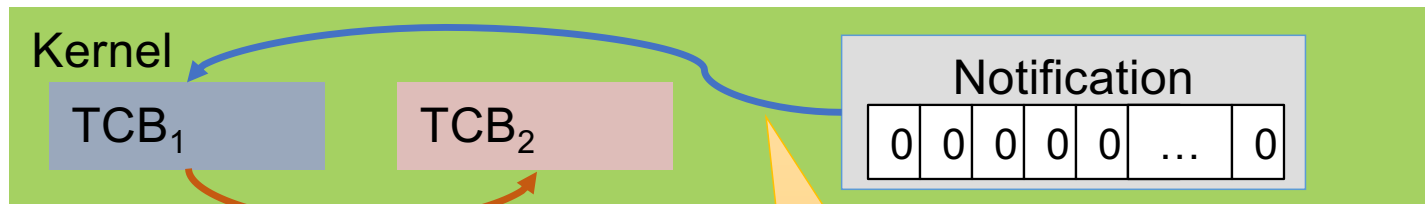
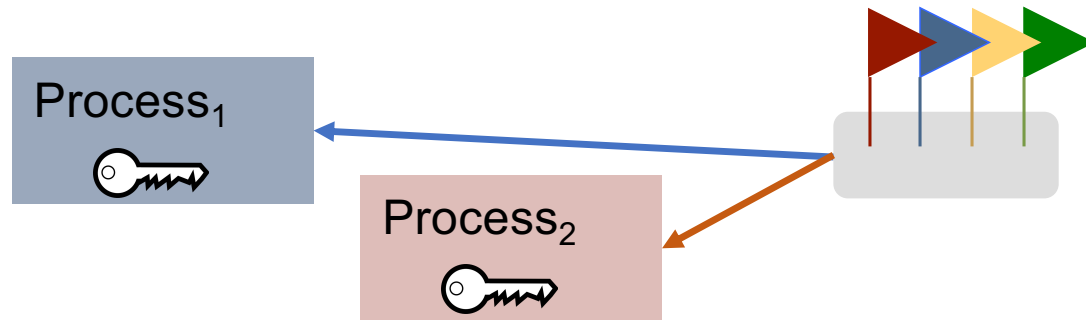
- Logically, a Notification is an array of binary semaphores
 - Multiple signalling, select-like wait
 - Not a message-passing operation!

- Implemented by *data word* in Notification
 - Send 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





Notification Queues



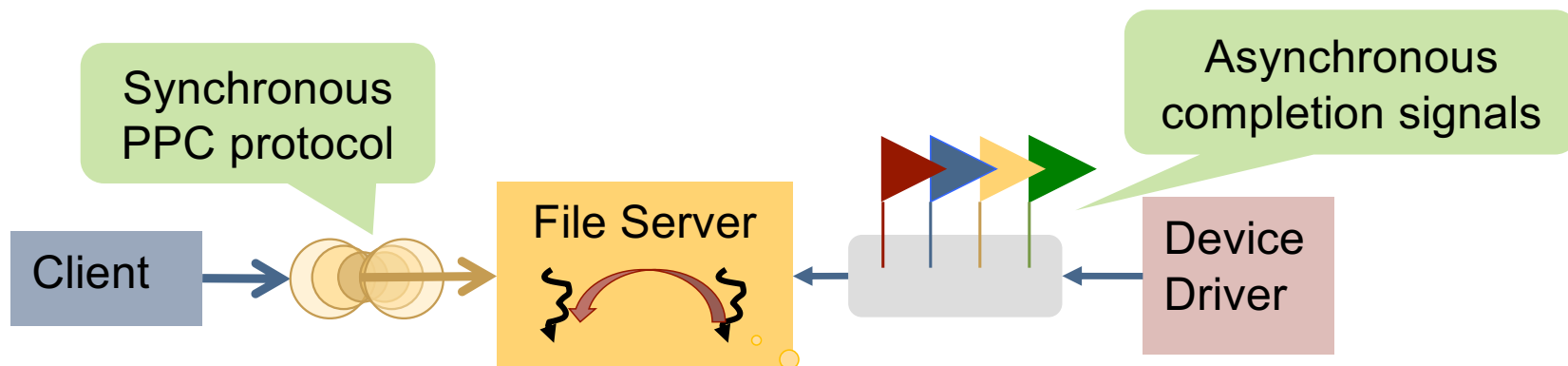
Further waiters
queued by priority

First invocation
queues waiter



Receiving from EP *and* Notification

Server with synchronous and asynchronous interface



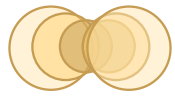
Better: single thread for both interfaces

- Notification “bound” to TCB
- Signal delivered as “PPC” from EP

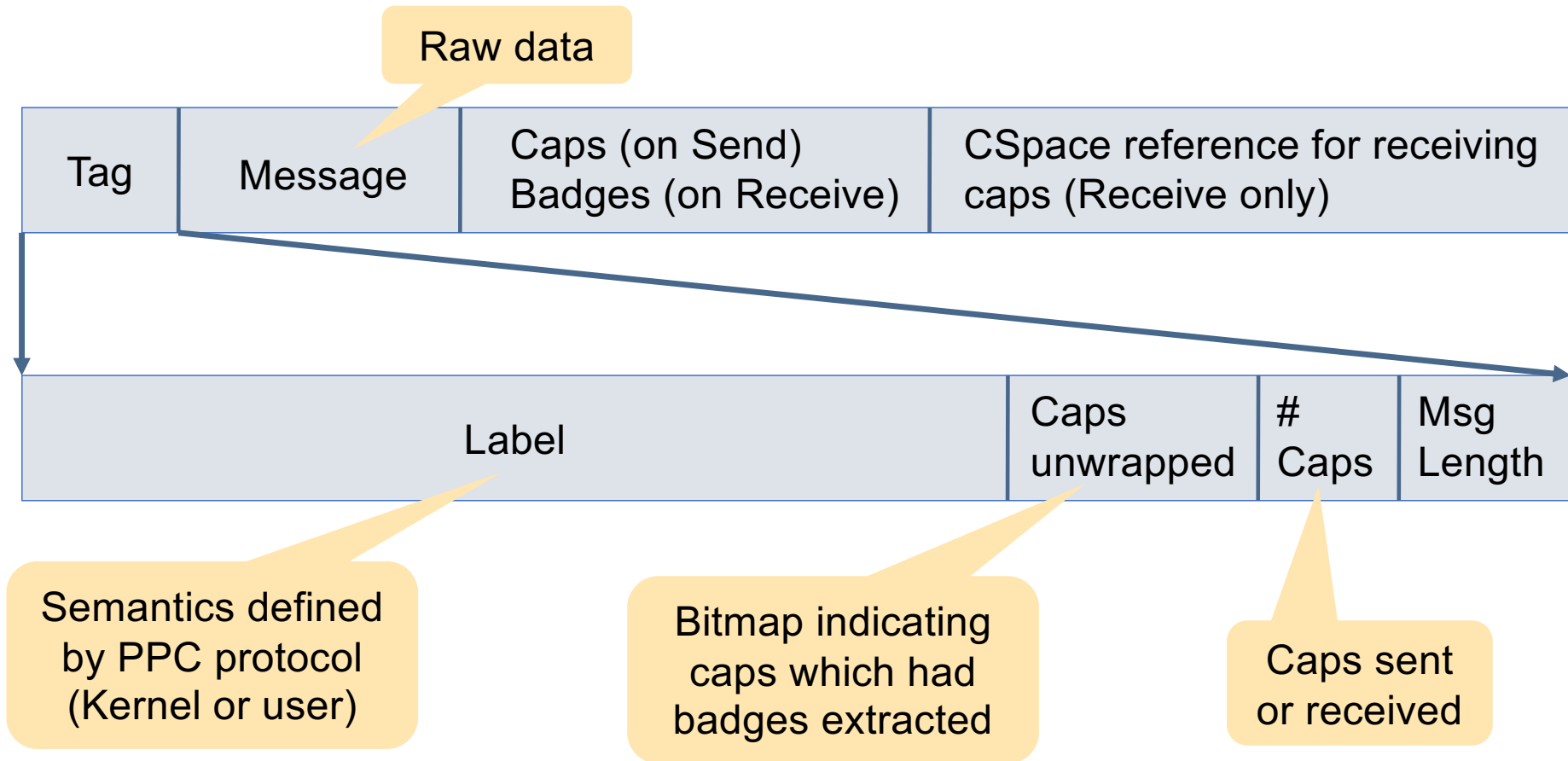
Separate thread per interface?

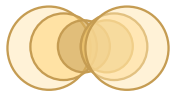
Concurrency control, complexity!

Must partition badge space to distinguish!



PPC Message Format





Client-Server PPC Example

```
seL4_MessageInfo_t tag = seL4_MessageInfo_new(0, 0, 0, 1);  
seL4_SetMR(0, value);  
seL4_Call(server_c, tag);
```

Client

Set message
register #0

CSPACE helper functions
in libseL4cspace

Server

```
ut_t* reply_ut = ut_alloc(seL4_ReplyBits, &cspace);  
seL4_CPtr reply = cspace_alloc_slot(&cspace);  
err = cspace_untyped_retype(&cspace, reply_ut->cap, reply,  
                             seL4_ReplyObject, seL4_ReplyBits);  
seL4_CPtr badged_ep = cspace_alloc_slot(&cspace);  
cspace_mint(&cspace, badged_ep, &cspace, ep, seL4_AllRights, 0xff);  
...  
seL4_Word badge;  
seL4_MessageInfo_t msg = seL4_Recv(ep, &badge, reply);  
...  
seL4_MessageInfo_t response = seL4_MessageInfo_new(0, 0, 0, 1);  
seL4_NBSend(reply, response);
```

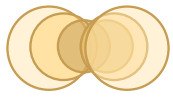
Allocate slot &
retype to RO

Reply to sender
identified by RO

Derive cap with
badge 0xff

Wait on EP, receiving
badge, setting RO

Note: this is for clarity, in
reality should use ReplyRecv!



Proper Server Loop

```
bool have_reply=FALSE;
seL4_MessageInfo_t msg;
while (1) {
    if(have_reply) {
        msg = seL4_ReplyRecv(ep, response, &badge, reply);
    } else {
        msg = seL4_Recv(ep, &badge, reply);
    }
    ... // process request, prepare response
}
```

EP to wait on

Return value

Client badge

Reply object

Reasons for no reply:

- Initialisation
- Received from bound Notification

Advertisement: Taste of Research Internships

- Official site: <https://www.unsw.edu.au/engineering/student-life/undergraduate-research-opportunities/>
- TS topics: <https://trustworthy.systems/students/internships>
- Applications deadline: 25 October
 - Talk to me before applying!



Complete the Student Experience Survey

Have your say & a chance to WIN \$1,000*!

Here's how:

1. Look out for your email invitation via the Social Research Centre (SRC).
2. Or scan the QR code for the student information page.
3. Click 'Complete Survey'

*T&Cs
apply



unsw.to/ses



Why is it important?

- We can hear your thoughts and make better decisions about our degree programs, and the services and resources we provide to you
- Results help future students make informed decisions on where and what they want to study, as data is available publicly
- The survey helps to improve the Australian higher education sector by benchmarking universities nationally.



unsw.to/ses

