SISTEMA ERAGILEAK
SISTEMAS OPERATIVOS

Curso 2019/2020 Ikasturtea

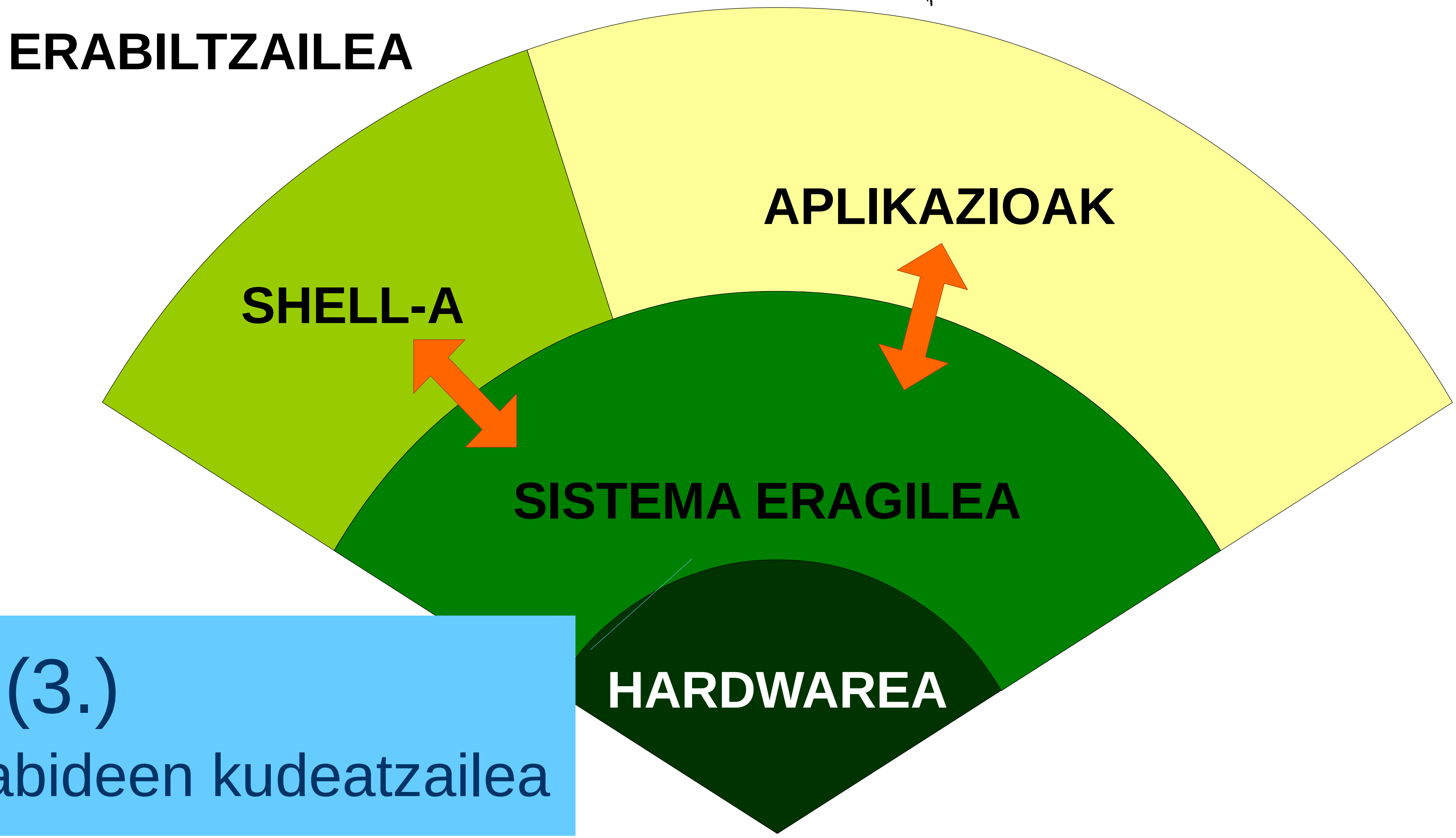
Informatika Ingeniaritzako Gradua
Grado en Ingeniería en Informática
Informatika Fakultatea

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

NAZIOARTEKO BIKAINTASUN CAMPUSA
CAMPUS DE EXCELENCIA INTERNACIONAL

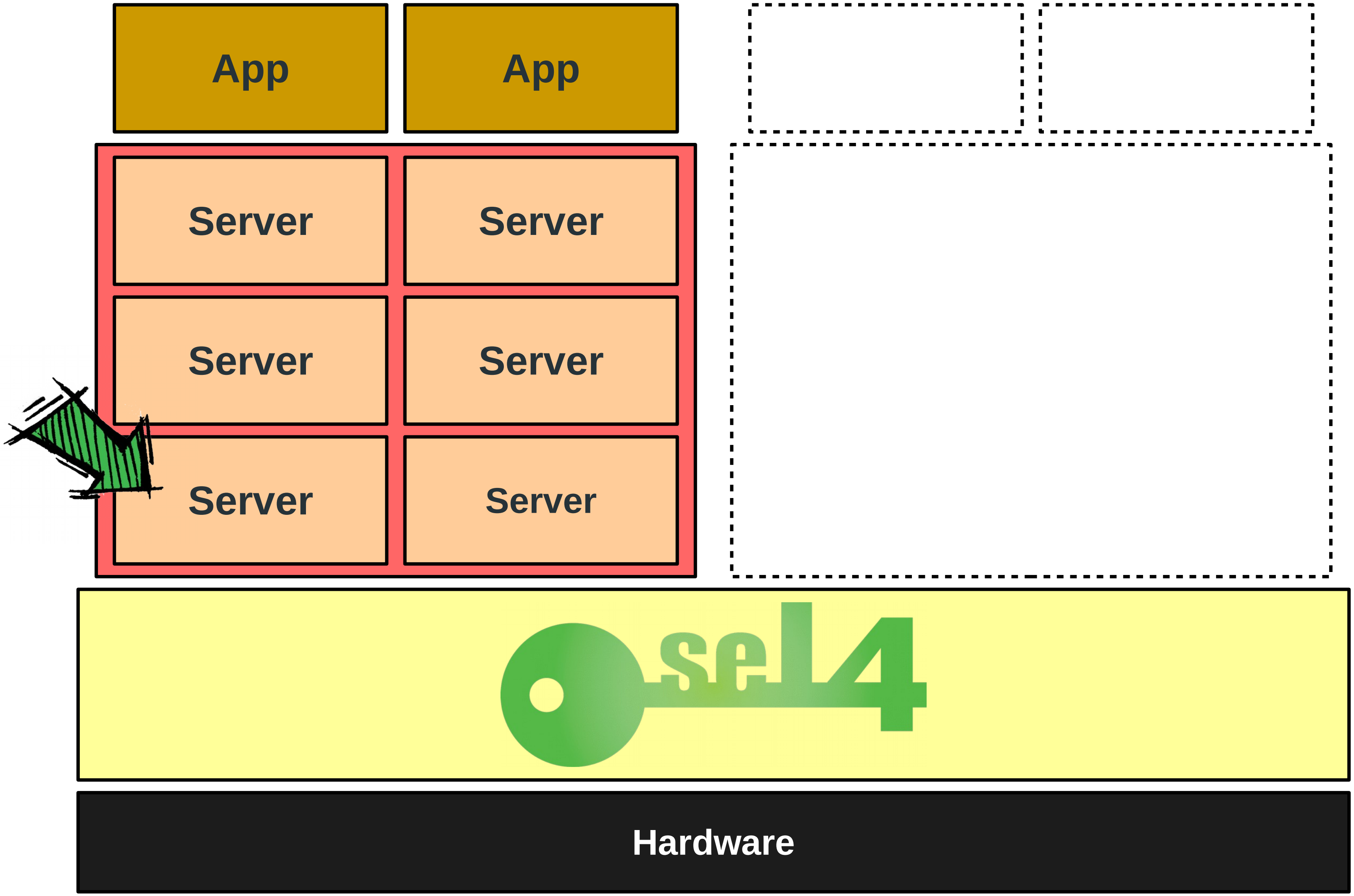
Ikuspegia

SEO (2.)
Makina abstraktoa
Sistema-deiak



SE (3.)
Baliabideen kudeatzailea

Lan praktikoa

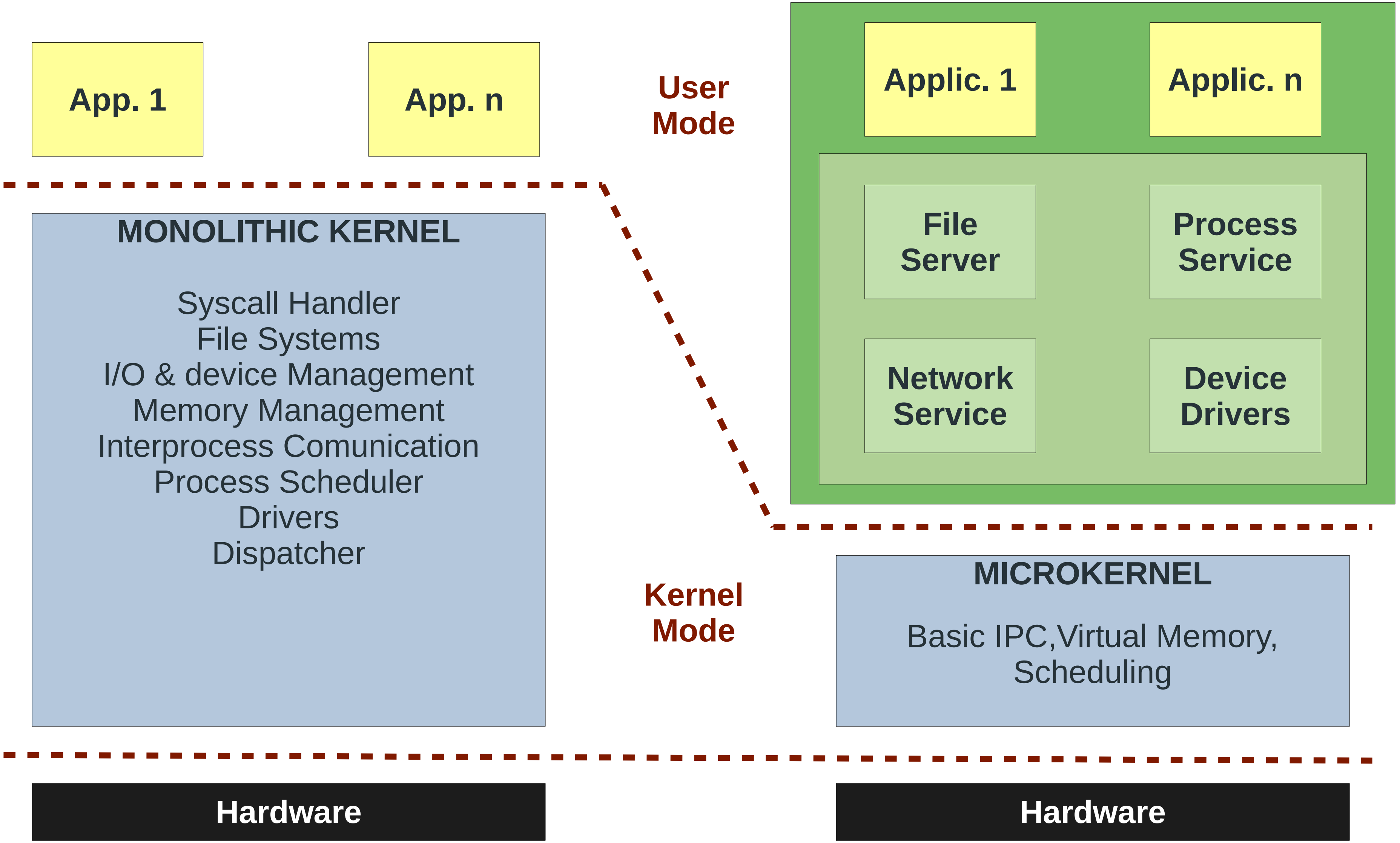


MONOLITHIC KERNEL

MICROKERNEL

INTERNAL STRUCTURE

Two design approaches:



MONOLITHIC KERNEL

- Just one address space
- User services and kernel services are implemented under same address space.
- Internal communication: shared memory
- Kernel provides all required functions using system calls
- xBSD, Linux, Windows.

MONOLITHIC KERNEL

Advantages:

- Performance
- Single static binary file (+ modules)

Disadvantages:

- Complex to debug and maintain
- If a service fails it leads to the entire system failure.

MICROKERNEL

- Minimal core of an operating system
- Does not provide high-level abstractions over the hardware (files, processes, sockets, etc)
- Minimal mechanisms for controlling access to physical address space, interrupts, and processor time
- Policy-freedom

MICROKERNEL

- Services executed at user level.
- Communication: memory sharing and IPC
- Microkernels are (and must be) very small
- L4 (seL4), Mac OS X (Darwin, Match)?, QNX.

MICROKERNEL

"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]

MICROKERNEL

Advantages:

- Smaller, easier to maintain and debug
- Services implemented as userspace processes
- More secure and reliable

Disadvantages:

- Poor performance? (communication latency)

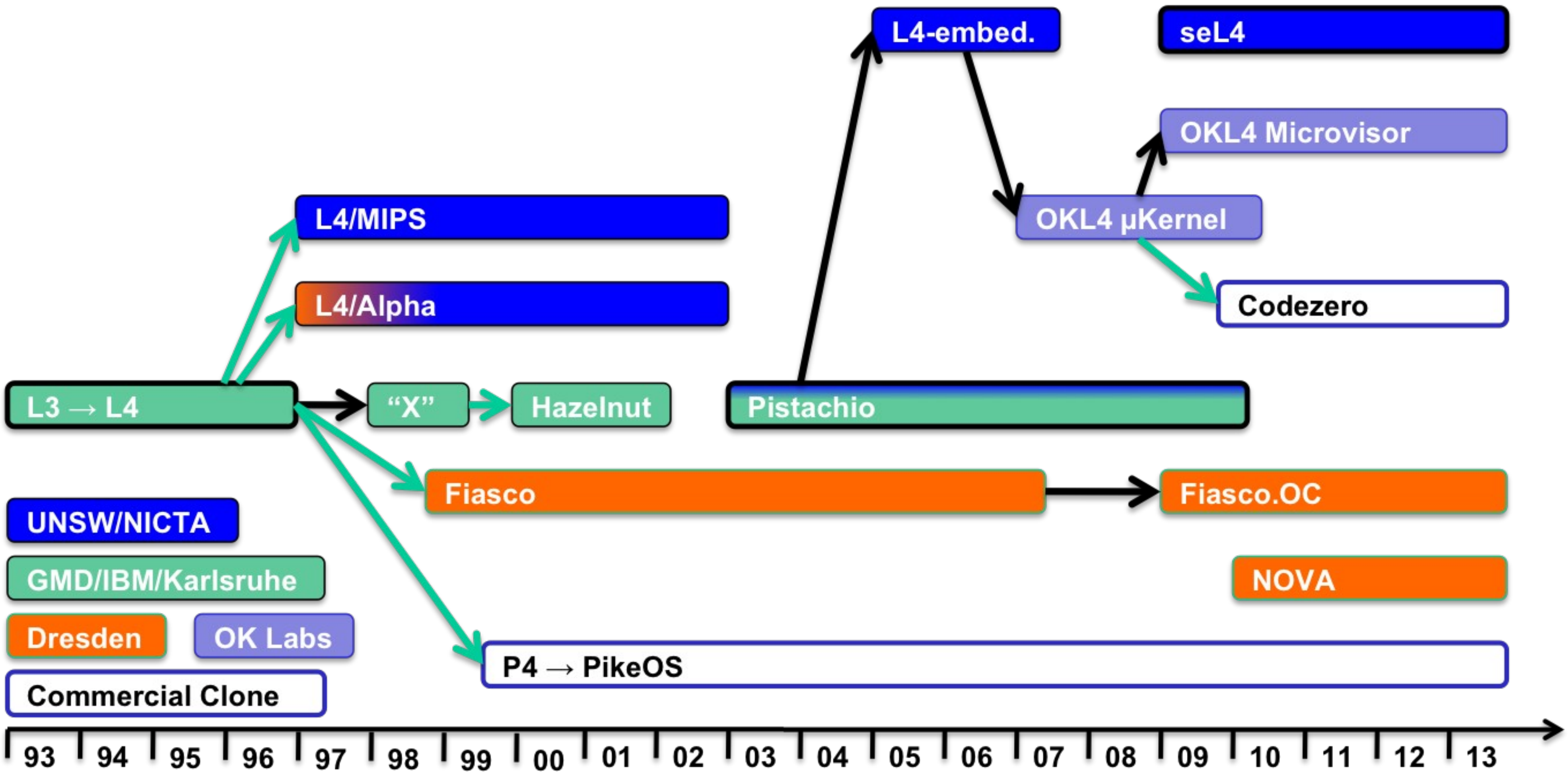
seL4 MICROKERNEL

seL4 MICROKERNEL

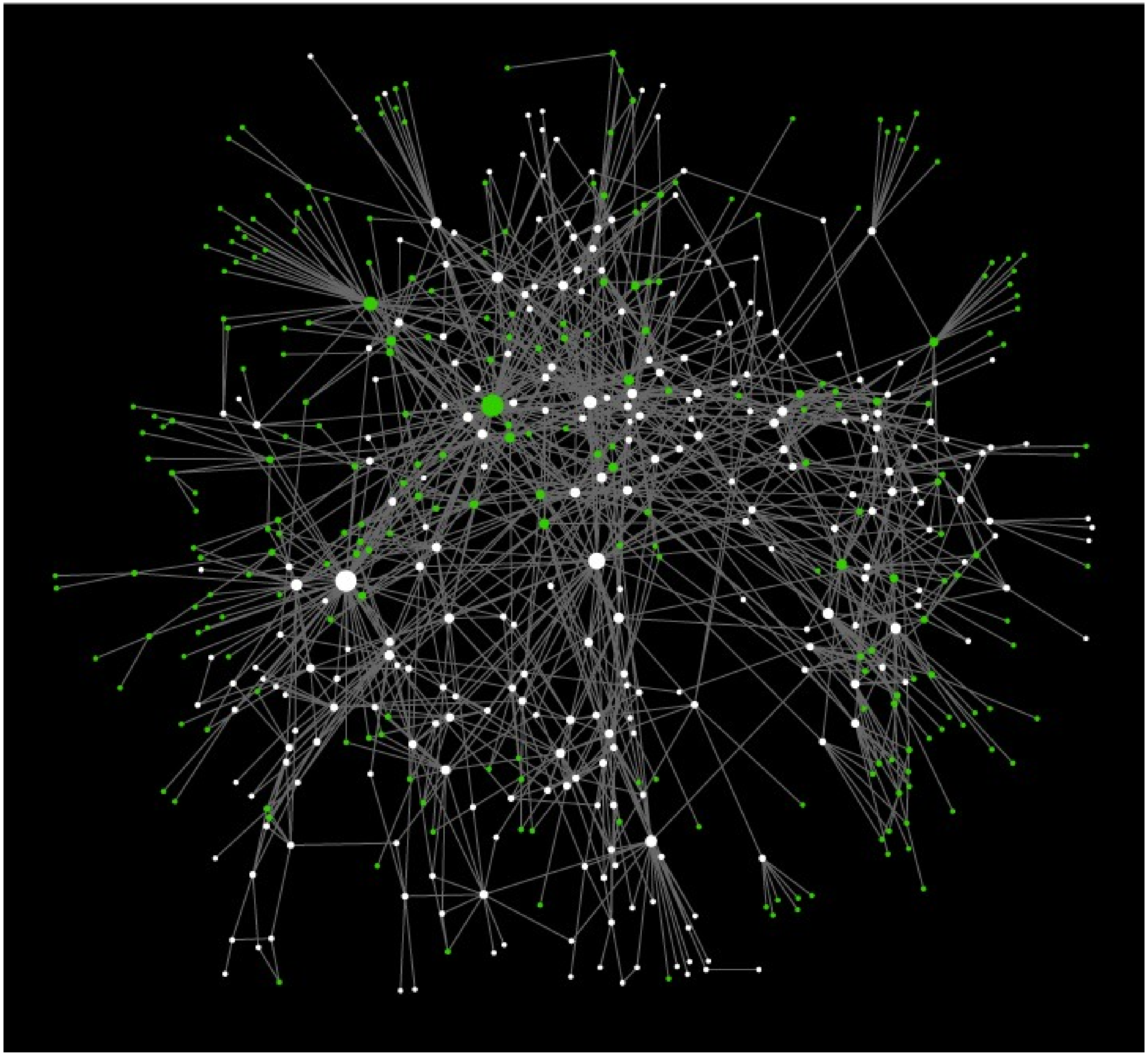
"The seL4 microkernel is an operating-system kernel designed to be a **secure**, **safe**, and **reliable foundation** for systems. As a microkernel, it provides a **small number of services** to applications, such as abstractions to create and manage virtual address spaces, threads, and inter-process communication (IPC)".

seL4 manual

L4 family


L4-embed.
seL4
OKL4 Microvisor
L4/MIPS
OKL4 µKernel
L4/Alpha
Codezero
L3 → L4
"X"
Hazelnut
Pistachio
Fiasco
Fiasco.OC
UNSW/NICTA
GMD/IBM/Karlsruhe
NOVA
Dresden
OK Labs
P4 → PikeOS
Commercial Clone
93 94 95 96 97 98 99 00 01 02 03 04 05 06 07 08 09 10 11 12 13

seL4 MICROKERNEL COMPLEXITY



https://ts.data61.csiro.au/projects/TS/l4.verified/visual.pml

seL4 MICROKERNEL PERFORMANCE

?

seL4 characteristics

- Small number of services
- Small implementation
- Just 8700 Lo**C** (v10.1.1, October 2018)
- A little bit bigger in the current version (MCS)
- Formally proven in Isabelle
- Worst-case execution time analysis

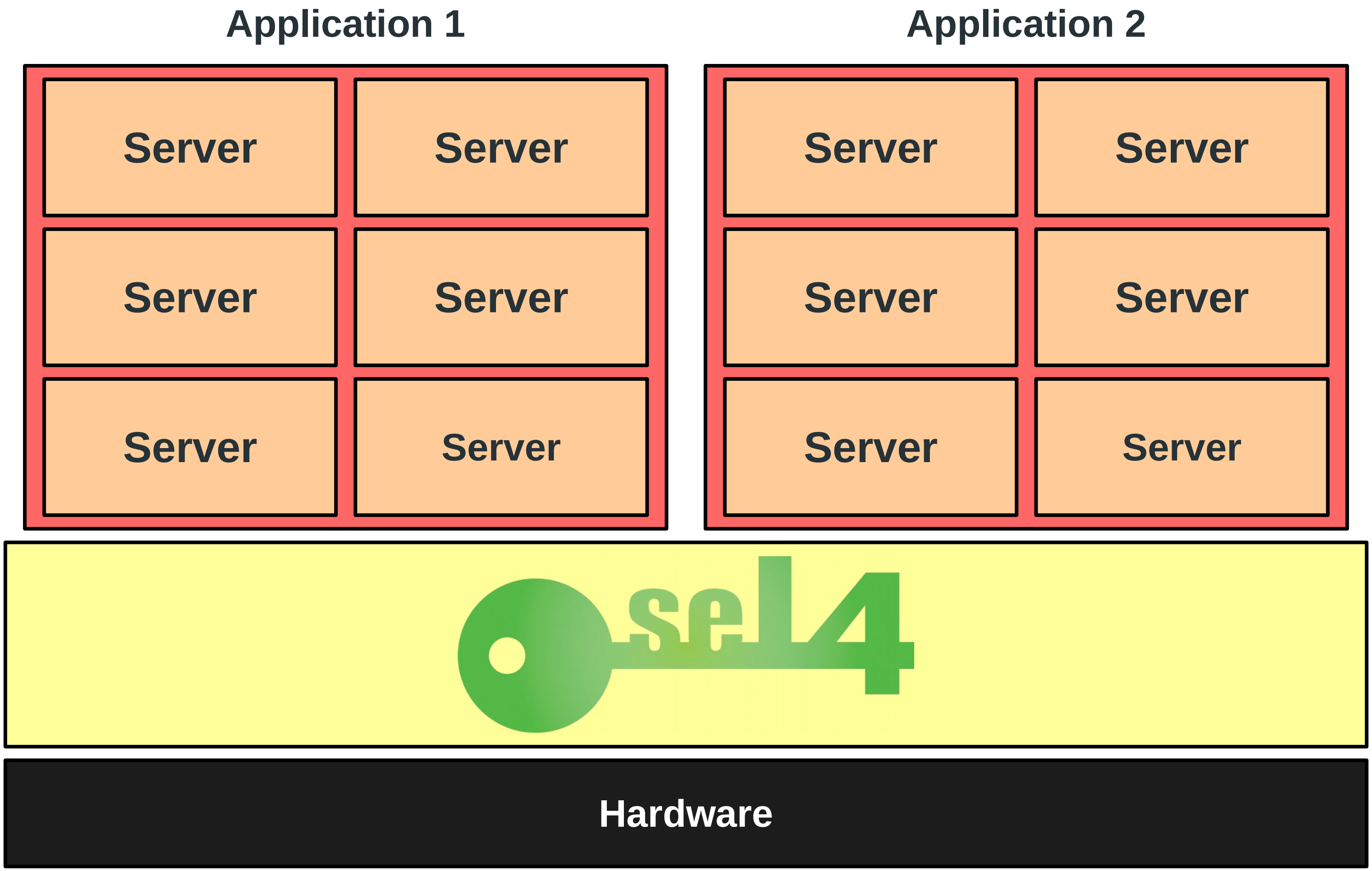
Where?

- Embedded systems with security or reliability requirements
- Virtual memory protection
- Application areas that need isolation between different parts of the software
- Financial, medical, automotive, avionics and defense sectors
- In Linux…

Where?

- Embedded systems with security or reliability requirements
- Virtual memory protection
- Application areas that need isolation between different parts of the software.
- Financial, medical, automotive, avionics and defense sectors.
- In Linux…OMG no!!! Linux in seL4 (VMM)

seL4


Application 1
Application 2
Server
Server
Server
Server
Server
Server
Server
Server
Server
Server
Server
Server
seL4
Hardware

Servers

- How do we implement them?
  - **seL4 threads**
- How many threads?
  - **Single- or multi-threaded**
- How do they communicate?
  - **IPC with endpoints**
- How much memory?
  - **seL4 untyped memory**
- How do they manage hardware interruptions?
  - **Notifications (binary semaphores)**
- What about security?
  - **Capabilities using CSpaces**

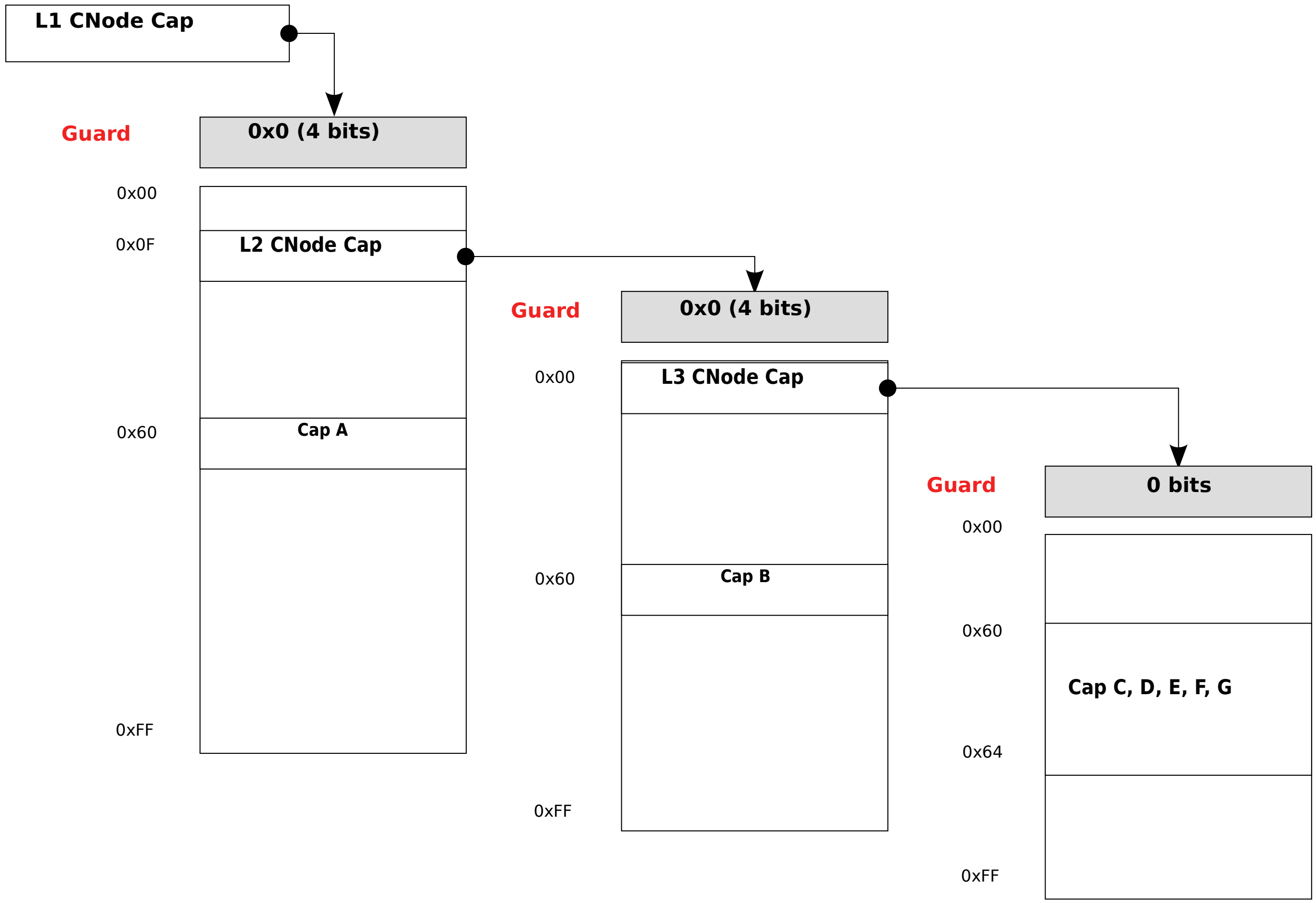
SeL4 services

- Threads
- Address spaces
- Inter-process communication (IPC) via endpoints
- Notifications
- Capability spaces
- ~~Device primitives~~

Capability spaces

- Each userspace thread a Cspace
- A Cspace contains capabilities
- Capabilities inside Cnodes with slots
- A Cspace address is an individual slot in some Cnode in the Cspace

Capability spaces



Inter-process communication

- Message-passing IPC mechanism
- Communication between threads (endpoints)
- Communication with the kernel
- Message words placed in message registers
- If not enough registers, placed in the **IPC buffer**

Notifications

- Simple, not blocking signaling mechanism
- Logically represents a set of binary semaphores
- seL4_Signal()
- **seL4_Wait() - blocking**
- seL4_Poll()

Threads and Execution

- Represent an execution context
- Manage processor time
- Represented as a TCB
- Each TCB a Cspace and a Vspace
- Also an **IPC buffer**?
- Belongs to one security domain
- Scheduling of threads? RR with 256 priorities

Address spaces and Virtual memory

- Vspace = Virtual address space
- Objects for managing virtual memory
- Top-level Vspace is architecture dependent
- Pages represent physical frames

Kit-kat: Supported architectures

- IA-32
- x86-64
- AArch32
- AArch64
- RISC-V 32
- **RISC-V 64**

Vspace in RISC-V 64

- seL4 supports three levels of paging
- PageTables indexed by 9 bits

Object	Address Bits	Level
PageTable	30-38	0
PageTable	21-29	1
PageTable	12-20	2

Constant	Size	Mapping Level
seL4_PageBits	4 KiB	2
seL4_LargePageBits	2 MiB	1
seL4_HugePageBits	1 GiB	0

System Bootstrapping

- Minimal environment for the initial thread
- TCB, Cspace and Vspace with the code/data of the thread and the IPC buffer
- All that information in the BootInfo Frame (struct)

LABS

Labs (1-4)

- SESO (Root task) reclaims untyped memory
- Initialise the Cspace
- Define a minimal system-call interface (Endpoint)
- Launch the test application (interface)
- Waits on synchronous IPC

Labs (5-)

- Group work (2)
- Define the syscall interface for a memory allocator inside SESO
- Programs such as Test App can request memory (malloc) and release memory (free)
- Implement a policy to manage SESO’s memory
- Start simple, contiguous memory and from there make improvements

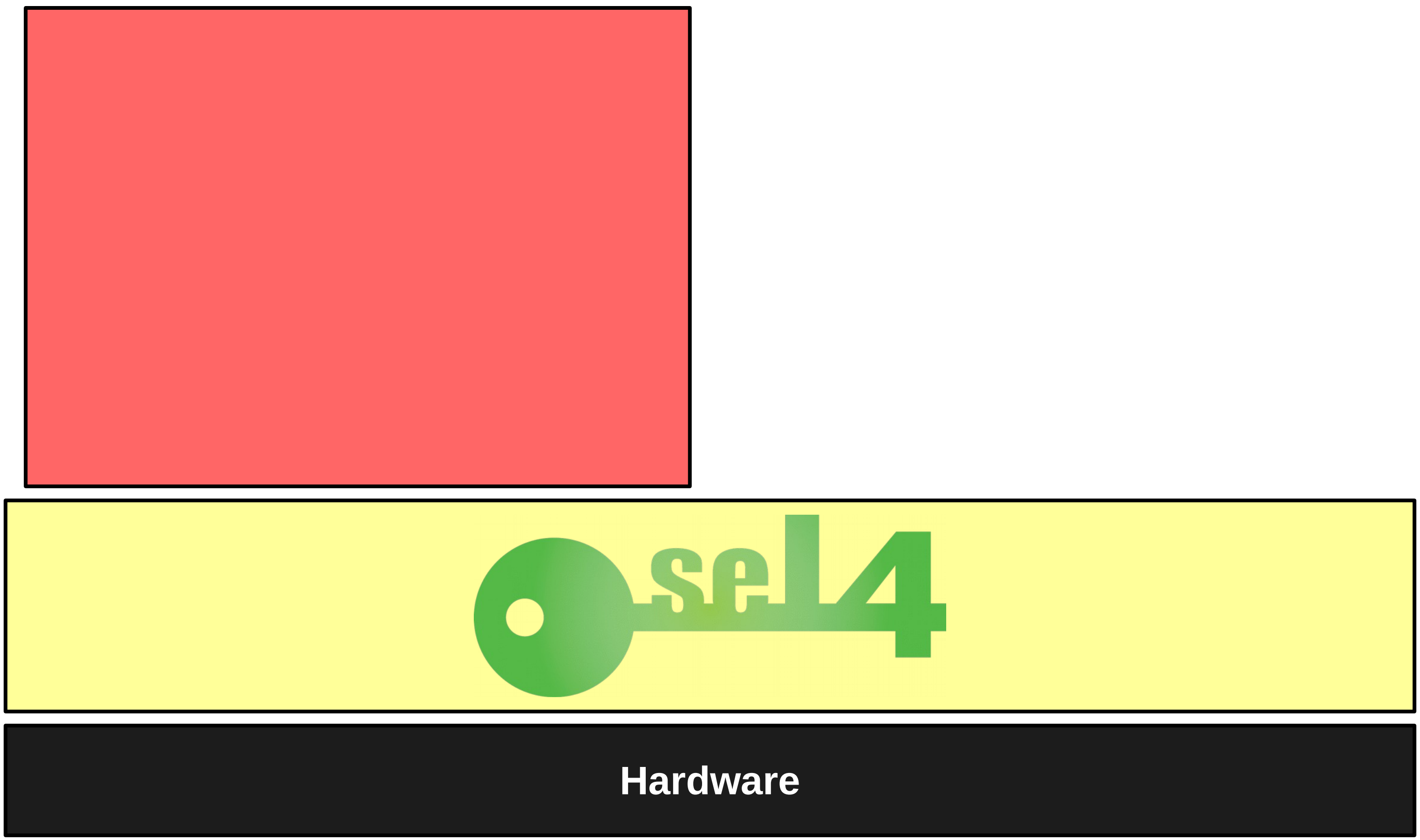
SESO

Hardware (simulated using qemu)

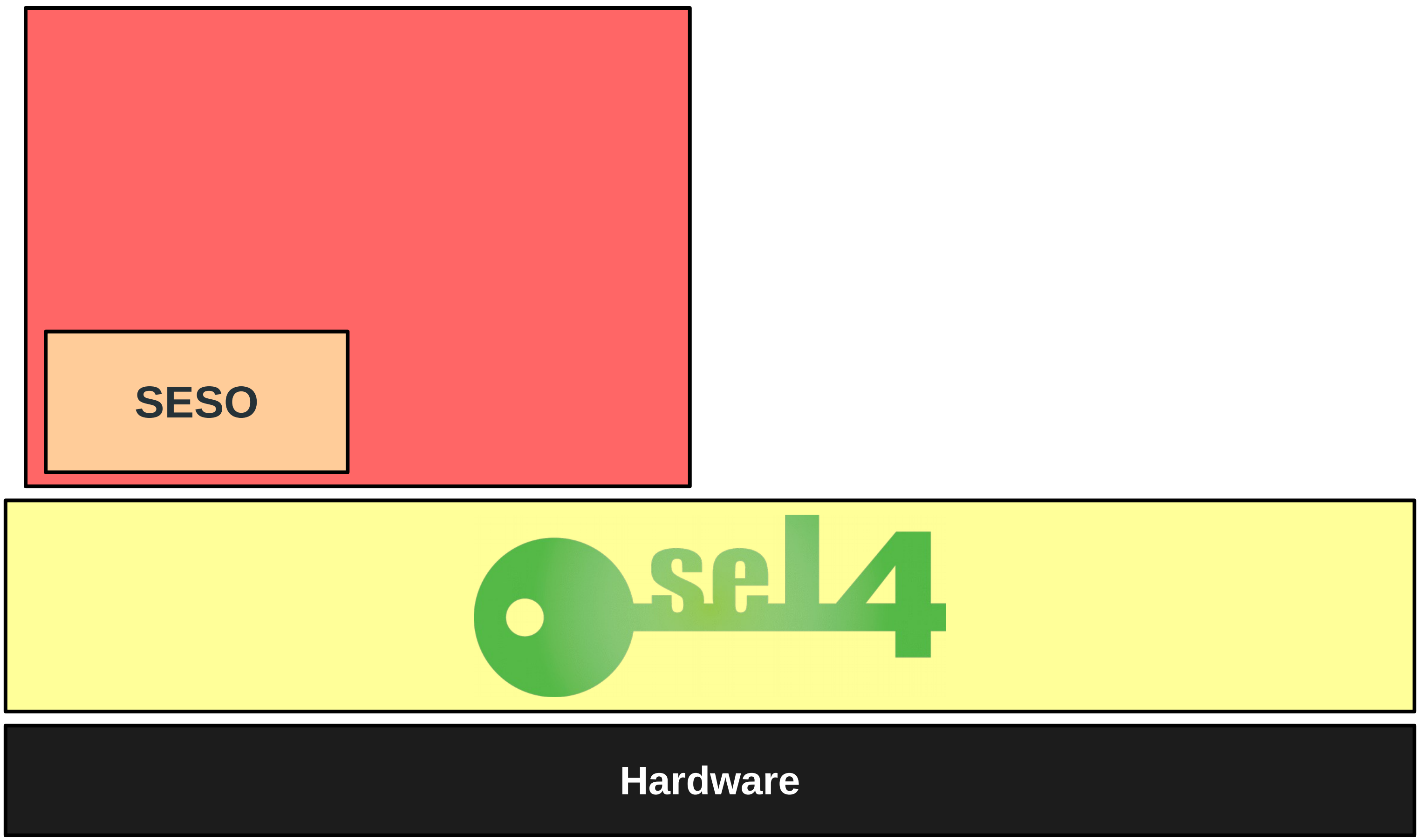
SESO

Hardware (simulated using qemu)

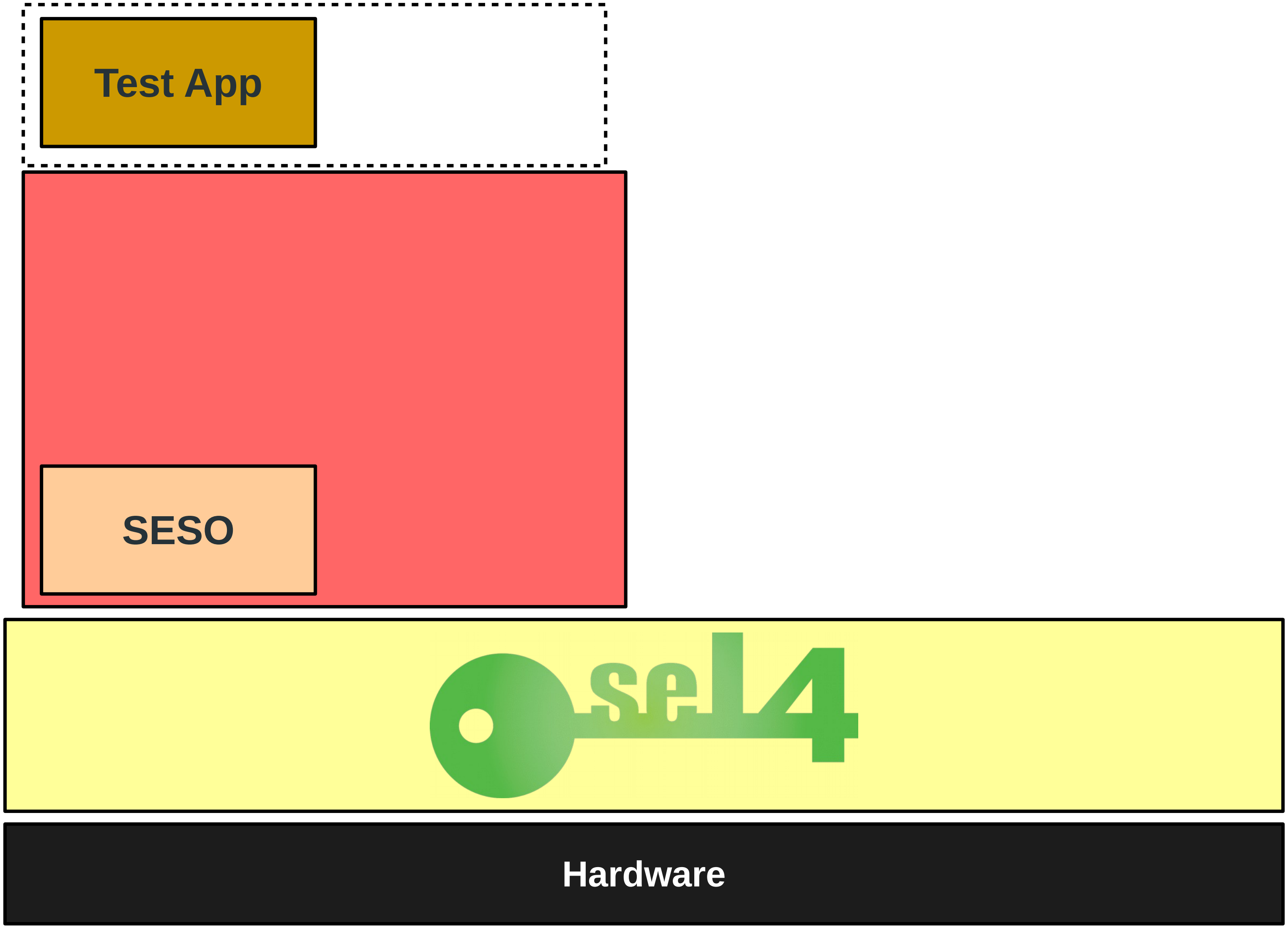
SESO



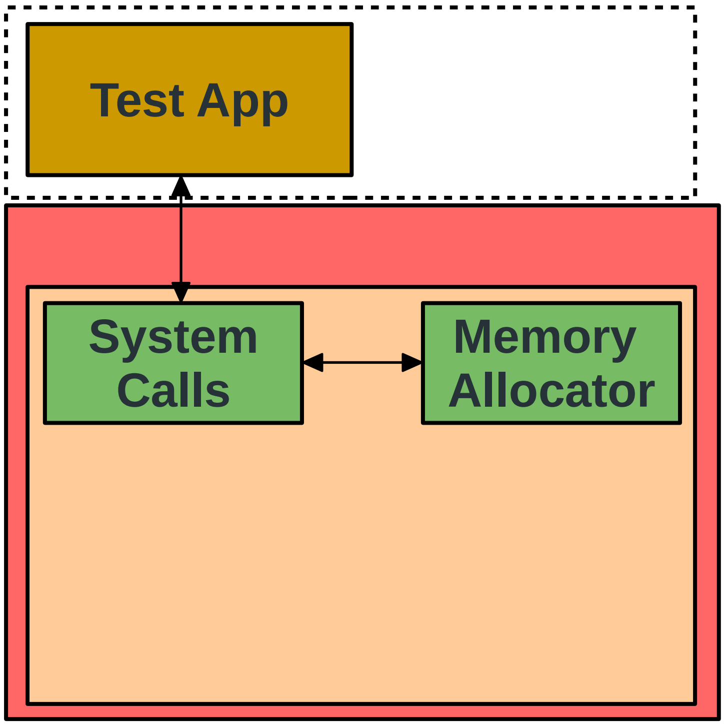
SESO



SESO



SESO



Methodology

- VM with Linux as development and deployment environment
- Today, we will install the required libraries and tools
- Hardware emulation using qemu

Install instructions

- Start SE-SO VM (link in Egela)
- Connect using ssh to localhost at port 8000
  - Username: seso Password: oses
- Easier to copy/paste and more…
- Create a directory called SESO…

  …and start cloning the required repos inside

Install instructions

- git clone --branch 10.1.1 https://github.com/seL4/seL4.git kernel
- git clone https://github.com/seL4/seL4_tools.git tools --branch 10.1.x-compatible
- mkdir projects
- git clone https://github.com/seL4/seL4_libs.git projects/sel4_libs --branch 10.1.x-compatible
- git clone https://github.com/seL4/musllibc.git projects/musllibc --branch 10.1.x-compatible
- git clone https://github.com/seL4/util_libs.git projects/util_libs --branch 10.1.x-compatible
- ln -s tools/cmake-tool/init-build.sh init-build.sh
- Create CMakeLists.txt
- mkdir build
- cd build
- ../init-build.sh -DPLATFORM=x86_64 -DSIMULATION=TRUE (**Fails**)
- Create root task
- ninja
- ./simulate

SISTEMA ERAGILEAK
SISTEMAS OPERATIVOS

Curso 2019/2020 Ikasturtea

Informatika Ingeniaritzako Gradua
Grado en Ingeniería en Informática
Informatika Fakultatea

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
NAZIOARTEKO BIKAINTASUN CAMPUSA
CAMPUS DE EXCELENCIA INTERNACIONAL