cs5965 Advanced OS Implementation

Lecture 01 – Introduction

Anton Burtsev

Why are we interested in building new Oses?

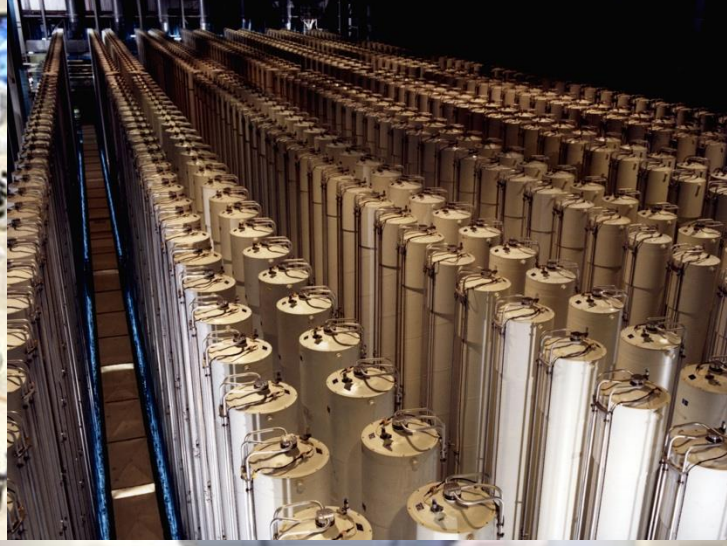
- Security
- Performance

Security

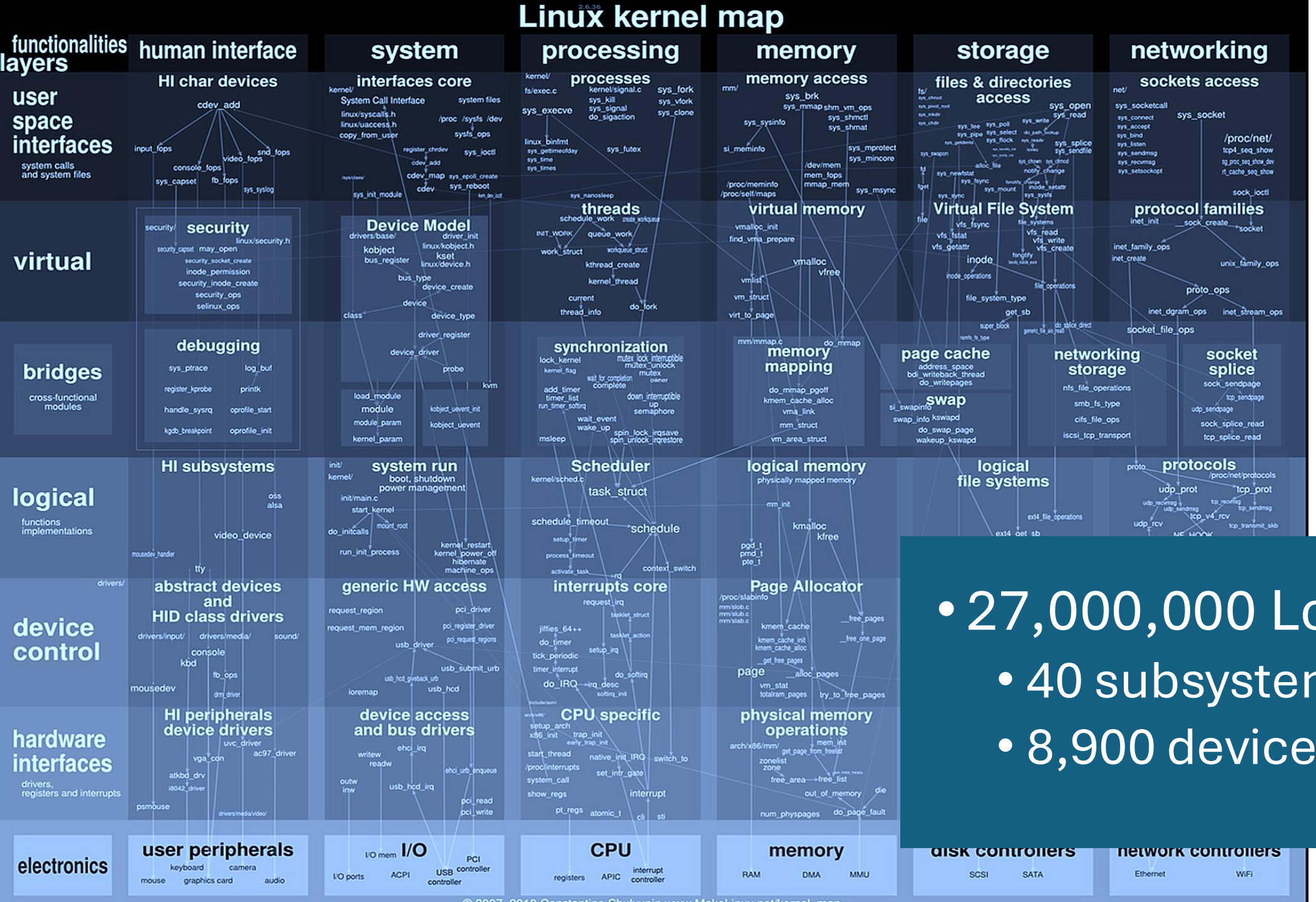
Operating systems haven't changed for decades

UNIX and in general nearly *all* of modern systems stack

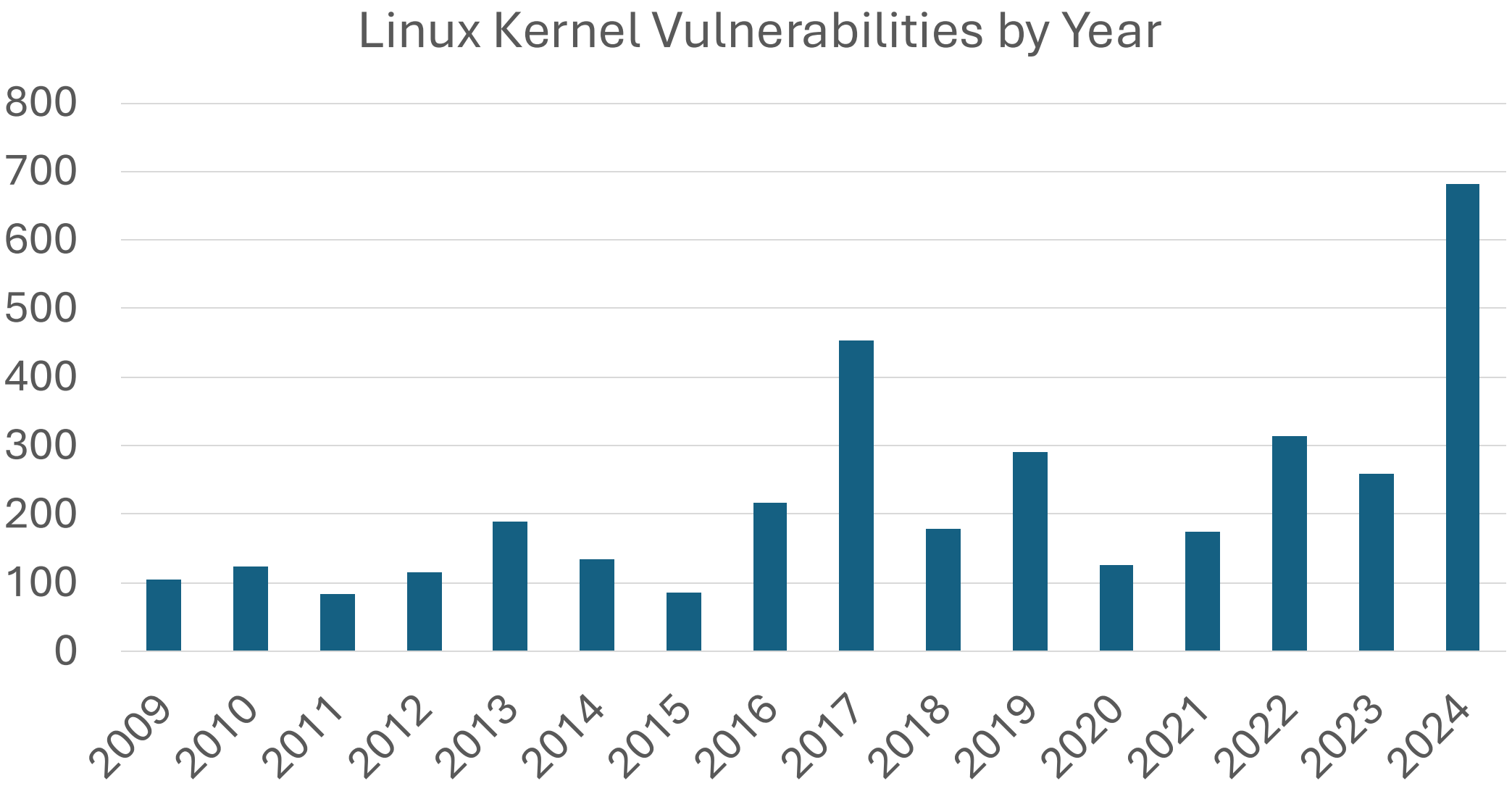
Ken Thompson (sitting) and Dennis Ritchie working together at a PDP-11 (1972)

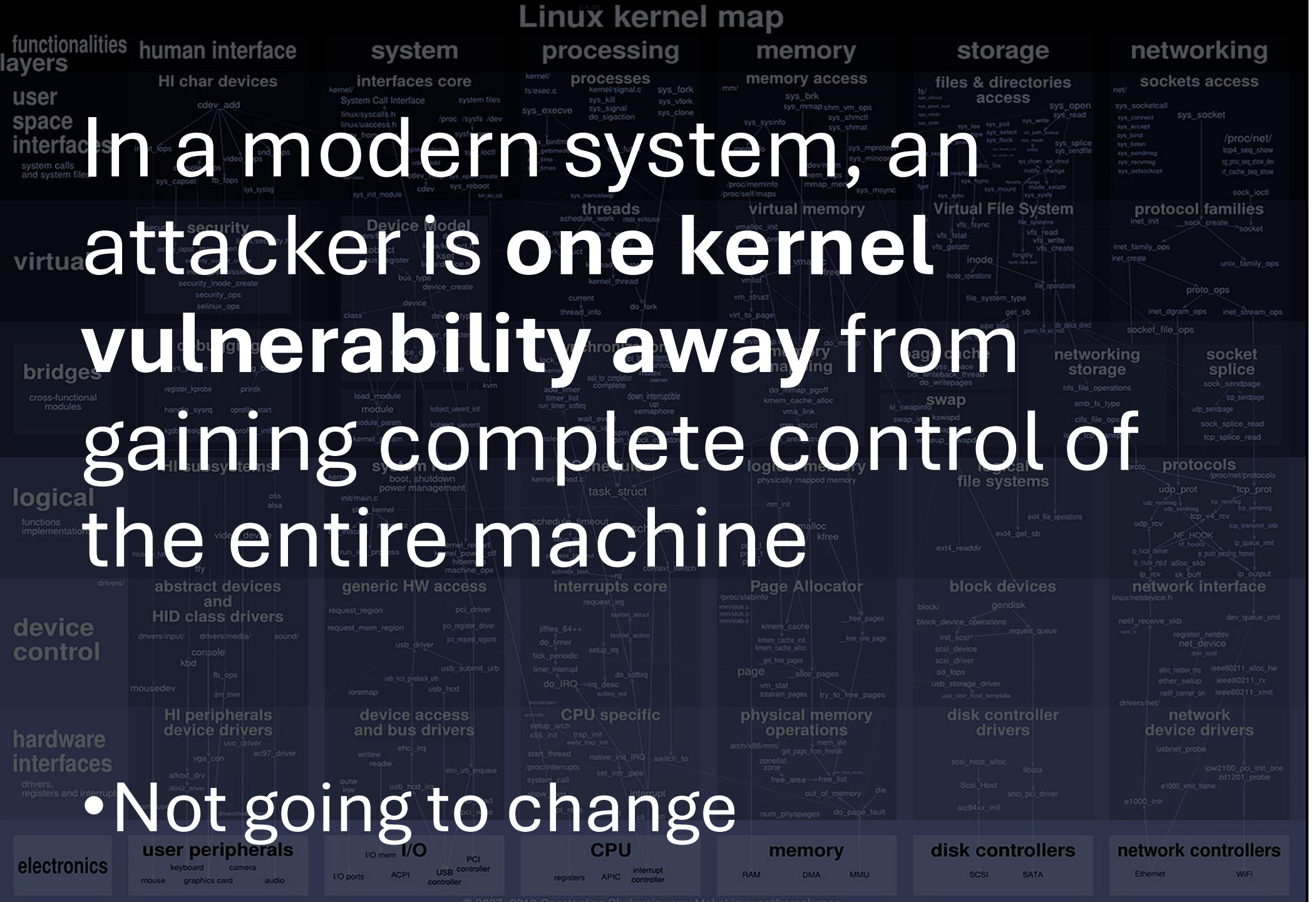


- Endless range of mission critical systems
- Same OS kernel (50 years old!)



- 27,000,000 LoC
  - 40 subsystems
  - 8,900 device drivers


Linux Kernel Vulnerabilities by Year
800
700
600
500
400
300
200
100
0
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024

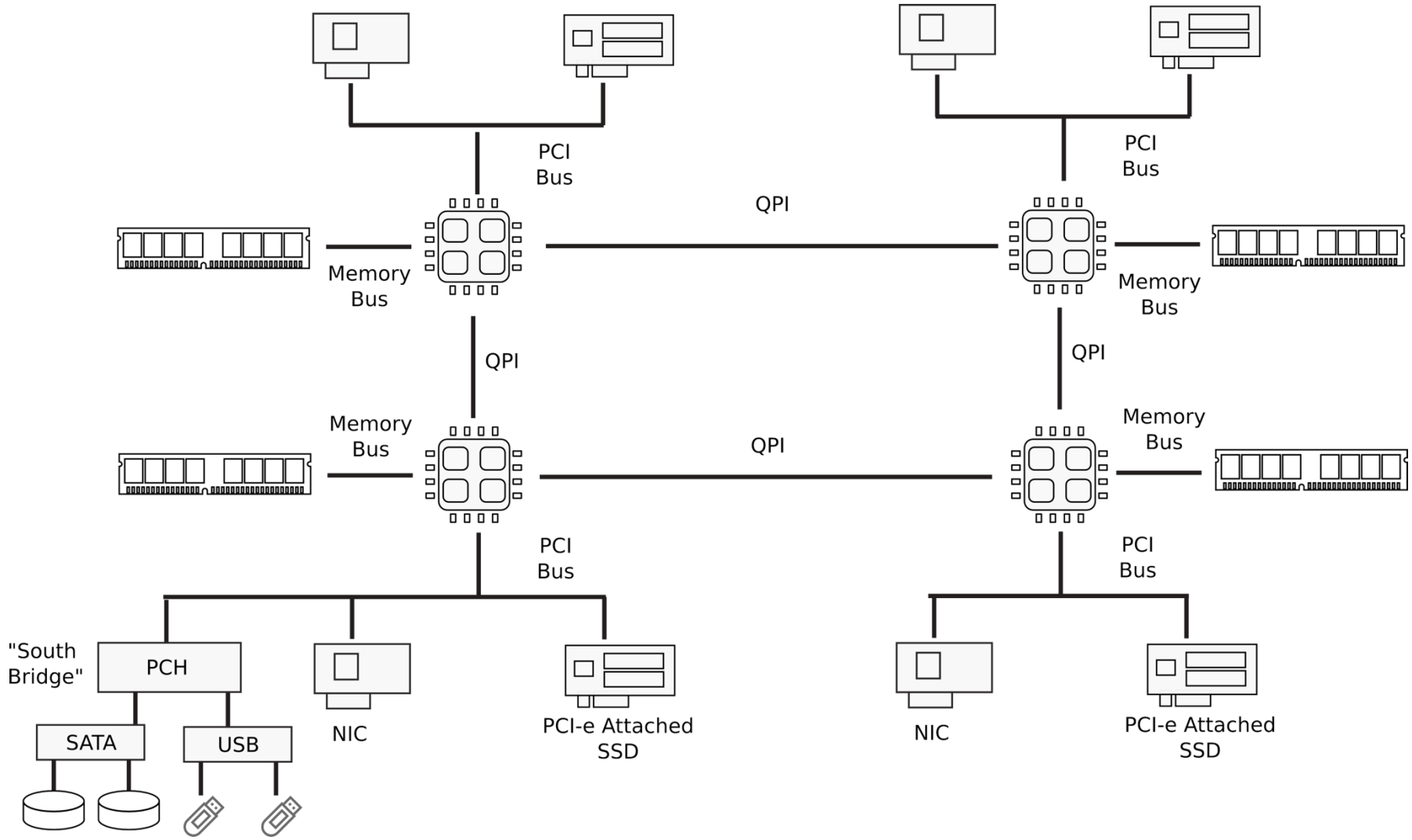


In a modern system, an attacker is **one kernel vulnerability away** from gaining complete control of the entire machine

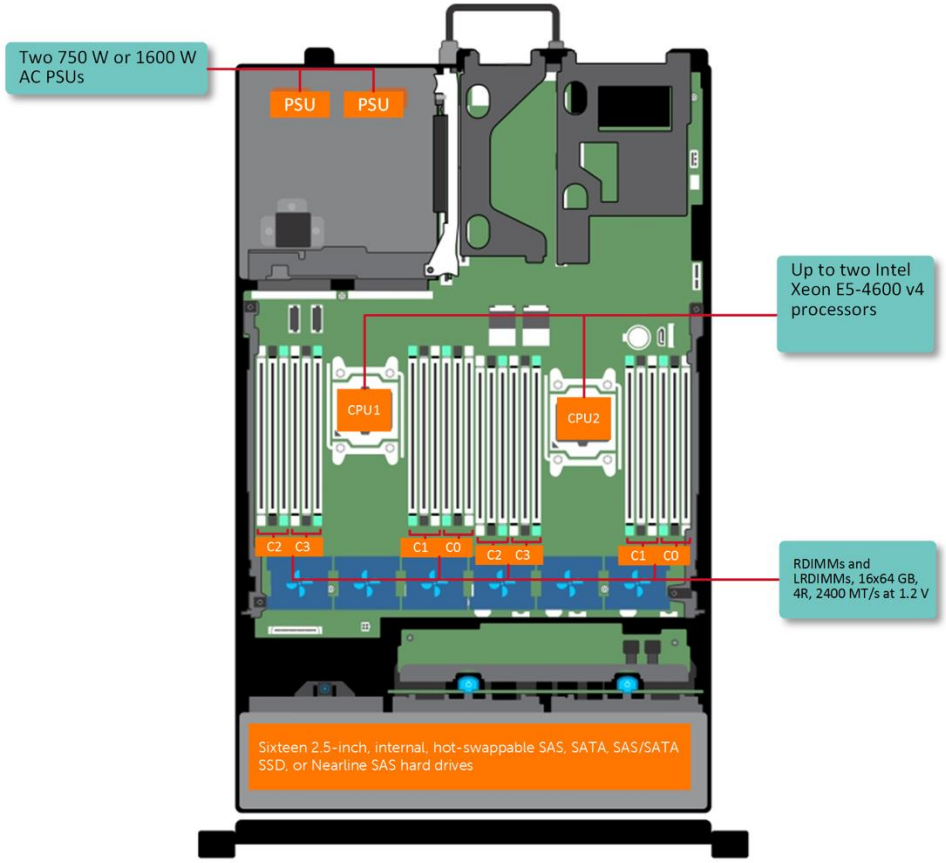
- Not going to change

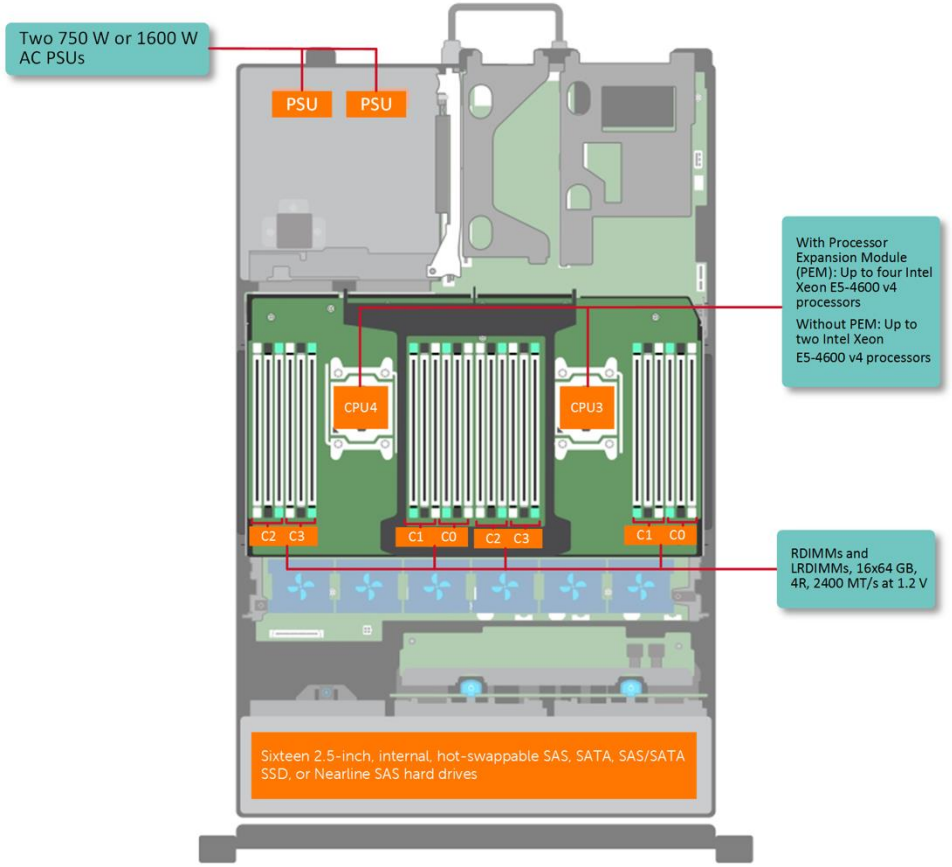
Performance

Modern hardware: Multi-socket machines



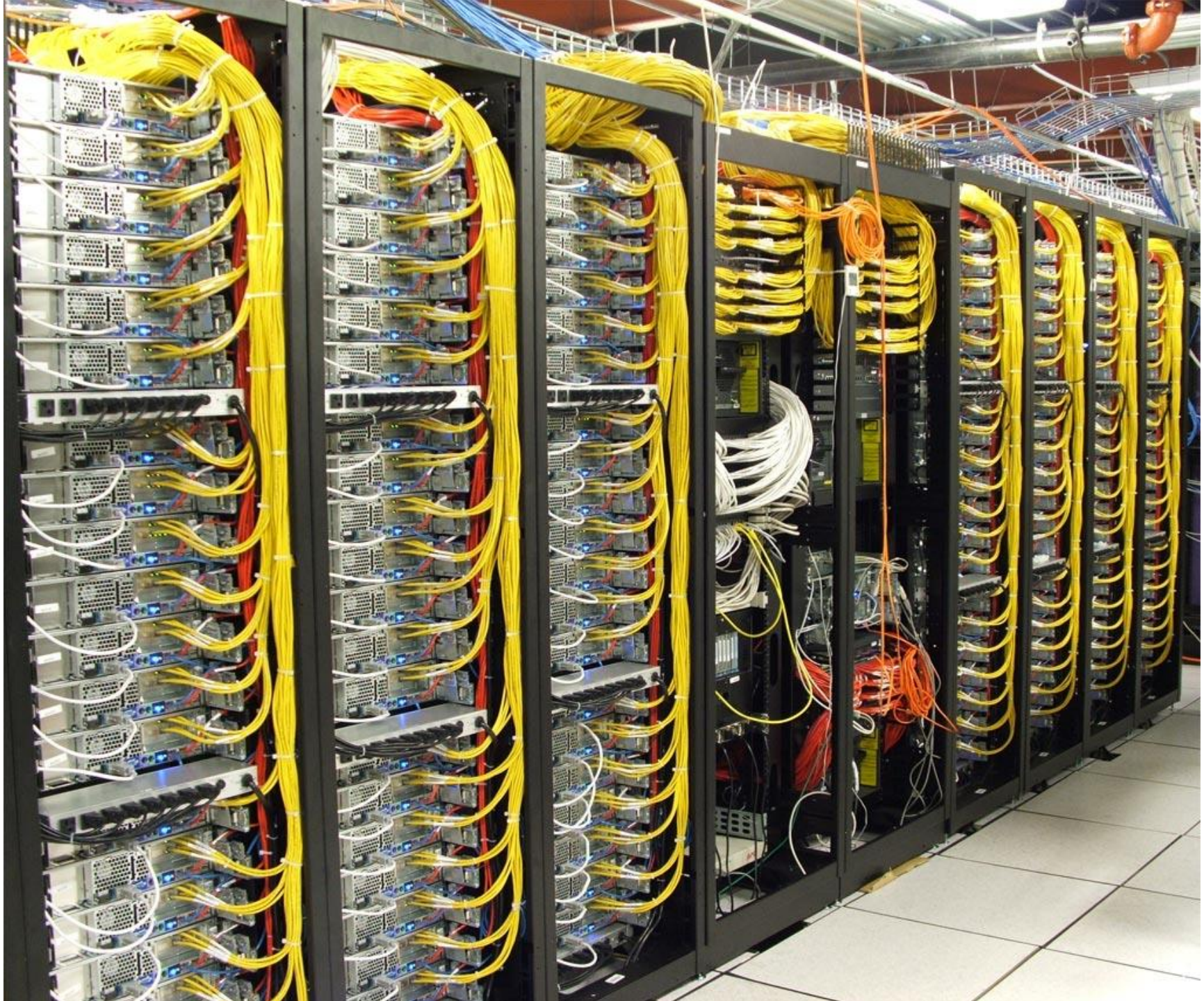
Dell R830 4-socket server





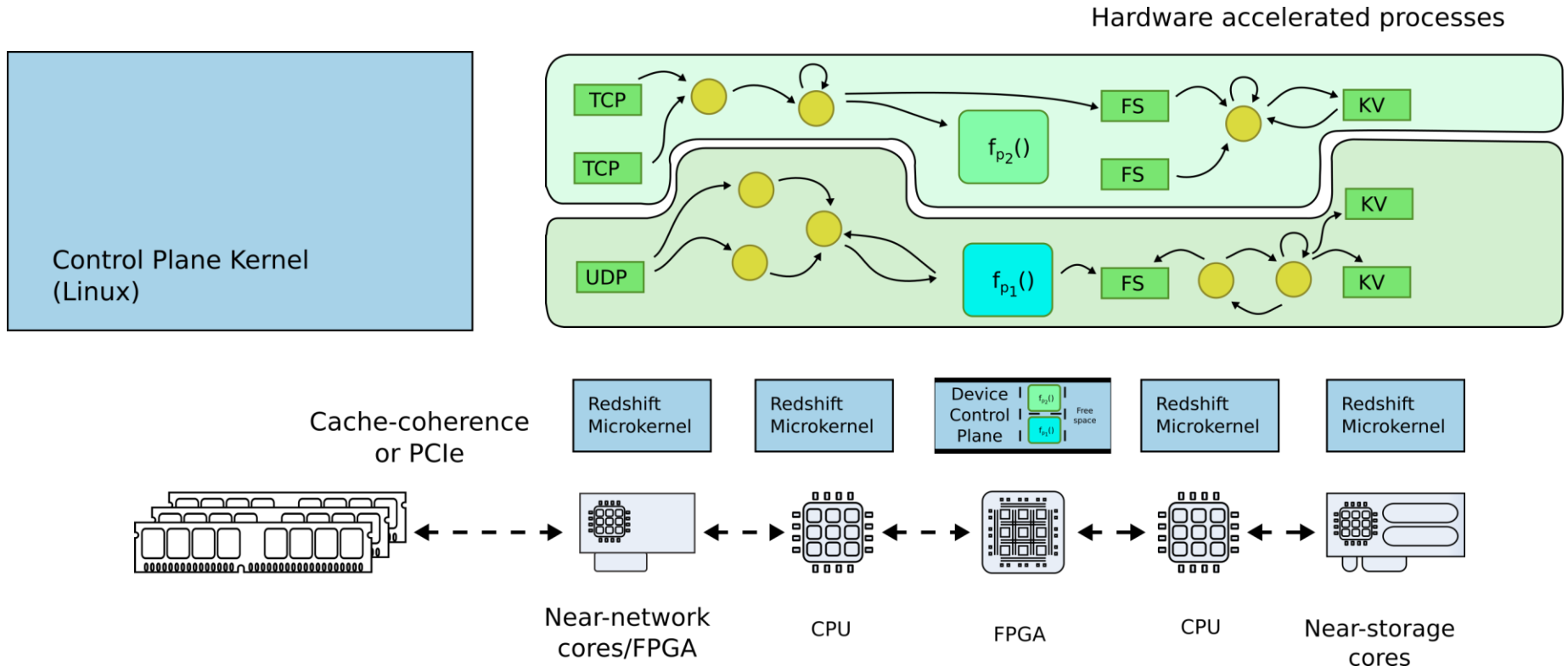
Dell Poweredge R830 System Server with 2 sockets on the main floor and 2 sockets on the expansion

http://www.dell.com/support/manuals/us/en/19/poweredge-r830/r830_om/supported-configurations-for-the-poweredge-r830-system?guid=guid-01303b2b-f884-4435-b4e2-57bec2ce225a&lang=en-us



But what will it look like in 5-10 years?

- Massively heterogeneous
  - Not just many-cores
  - GPUs, AI accelerators, near-storage and near-network cores
- But also
  - Fine-grained hardware ASICs accelerators
  - Programmable hardware (FPGA)


Hardware accelerated processes
Control Plane Kernel
(Linux)
TCP
TCP
UDP
$f_{p_2}()$
$f_{p_1}()$
FS
FS
FS
KV
KV
KV
Cache-coherence
or PCIe
Redshift
Microkernel
Redshift
Microkernel
Device
Control
Plane
Free
space
Redshift
Microkernel
Redshift
Microkernel
Near-network
cores/FPGA
CPU
FPGA
CPU
Near-storage
cores

But still: what is there for you?

- Low-level systems still pay
  - And will likely continue to pay even in the age of AI
- Performance
  - Cycle-level optimizations to extract last bits of performance from your datacenter-scale workloads
  - AI needs fast systems too
- Security
  - People tend to care less about security
    - Doesn’t translate to $$$ directly
    - But understanding of low-level details of the execution runtime (i.e., OS and everything up) still pays

At a high-level we teach a 3-class sequence

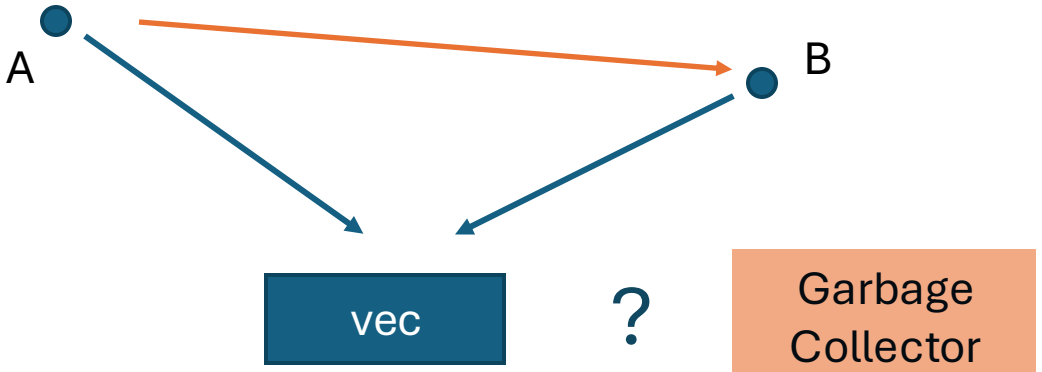
- 5460 Operating systems
- CS 5965 - Advanced OS Implementation
  - This class: learn low-level details by building
  - Will be able to understand low-level details of other Oses
  - Will be able to hack and build new ones
- CS 6465 - Advanced OS (Research Topics)
  - Research directions in the systems community
  - Modern hypervisors, microkernels, jails, virtualization, cloud, etc.
  - Will be able to lead development

Why Rust?

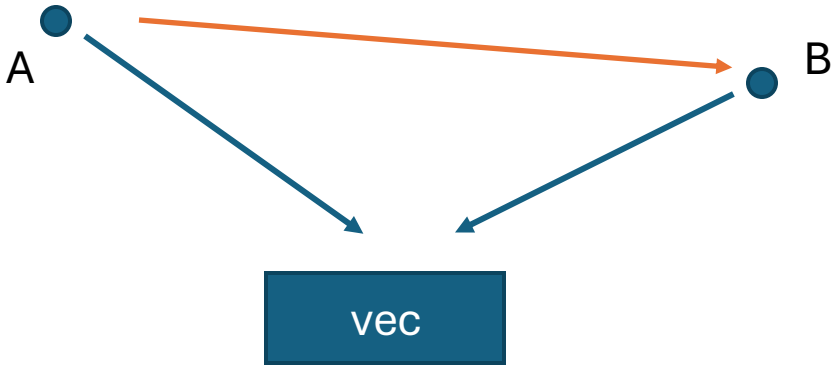
Rust

- Safe language build around idea of linear types
  - Normally, safety requires a garbage collector
    - Multiple pointers can point into an object
    - Even if one pointer is deallocated we don't know if there are other aliases
  - In Rust there are no aliases!
    - No need to walk the heap

Java, C#



Rust



Rust is the first safe alternative to C for low-level systems code

- Safe code remains fast
  - No garbage collection

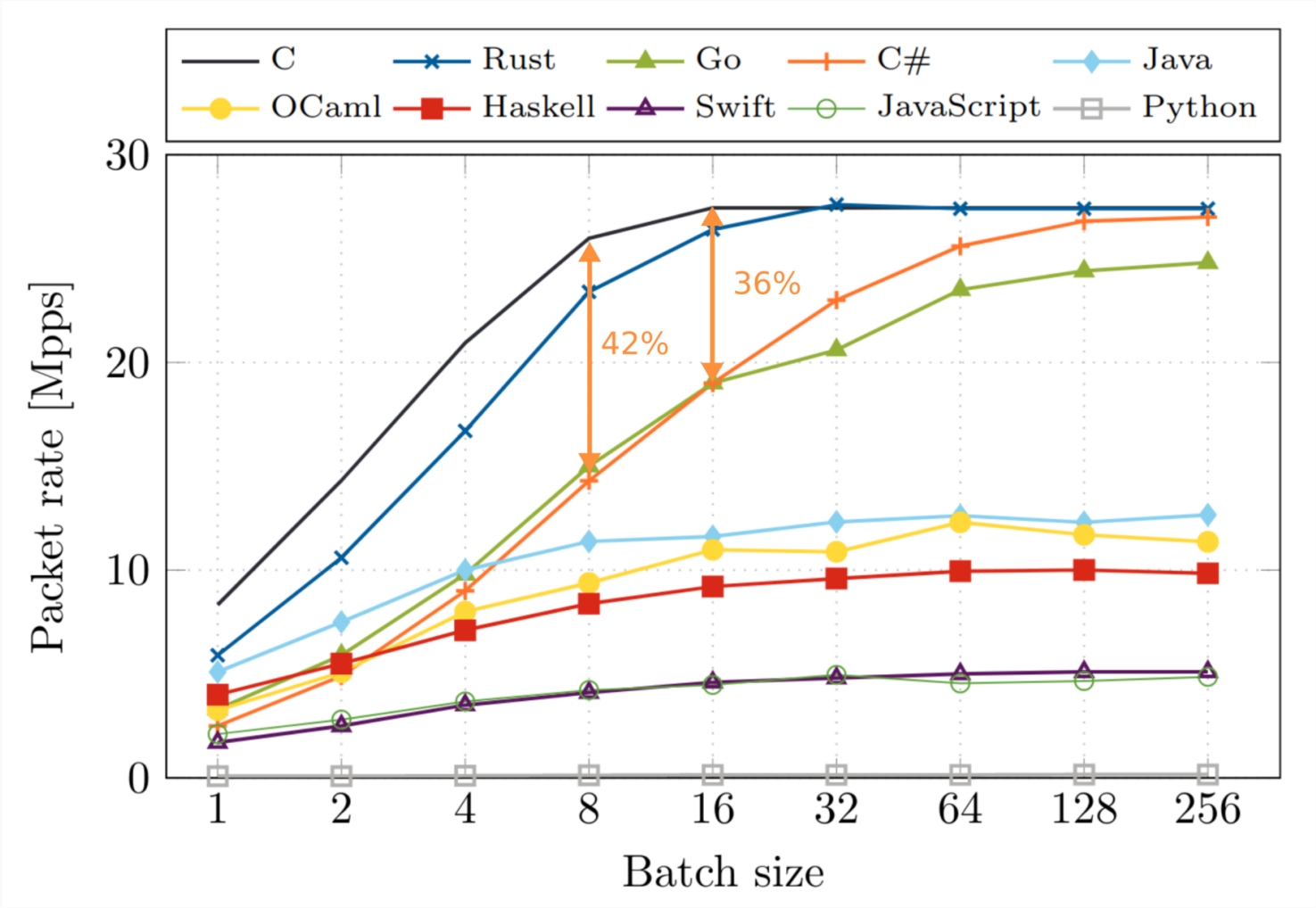


Figure 1: Forwarding rate for a minimal DPDK-like device driver implemented in 10 different languages. The driver uses one CPU core to forward packets on two 10 Gbit/s Intel X520 NICs.[2]

[2] The Case for Writing Network Drivers in High-Level Programming Languages, ANCS 2019

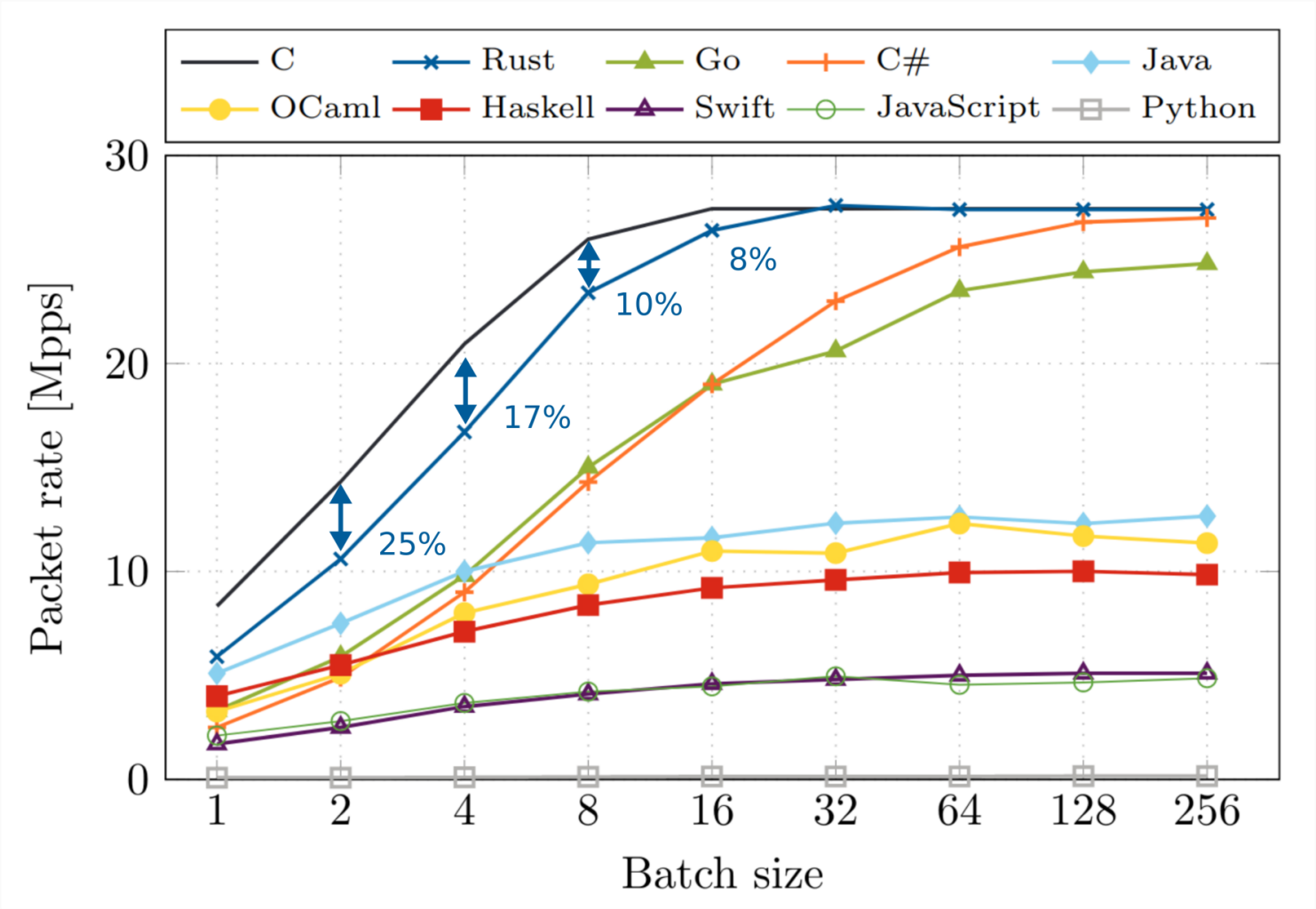


Figure 1: Forwarding rate for a minimal DPDK-like device driver implemented in 10 different languages. The driver uses one CPU core to forward packets on two 10 Gbit/s Intel X520 NICs.[2]

[2] The Case for Writing Network Drivers in High-Level Programming Languages, ANCS 2019

In practice it's a bit trickier

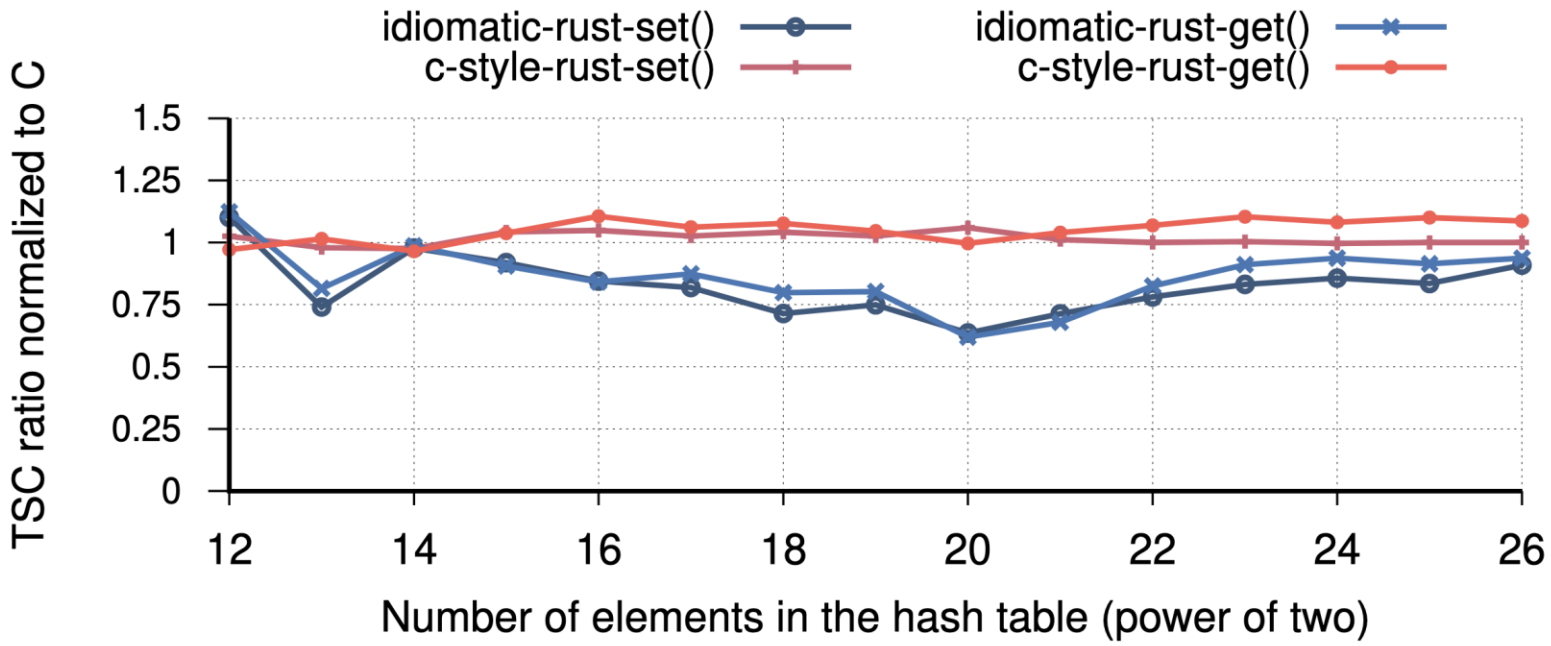


Figure 5: C vs Rust performance comparison

- From RedLeaf [OSDI'2020]

Rust is the first safe alternative to C for low-level systems code

- Safe code remains fast
  - No garbage collection
- Safety
  - 70% reduction in low-level vulnerabilities

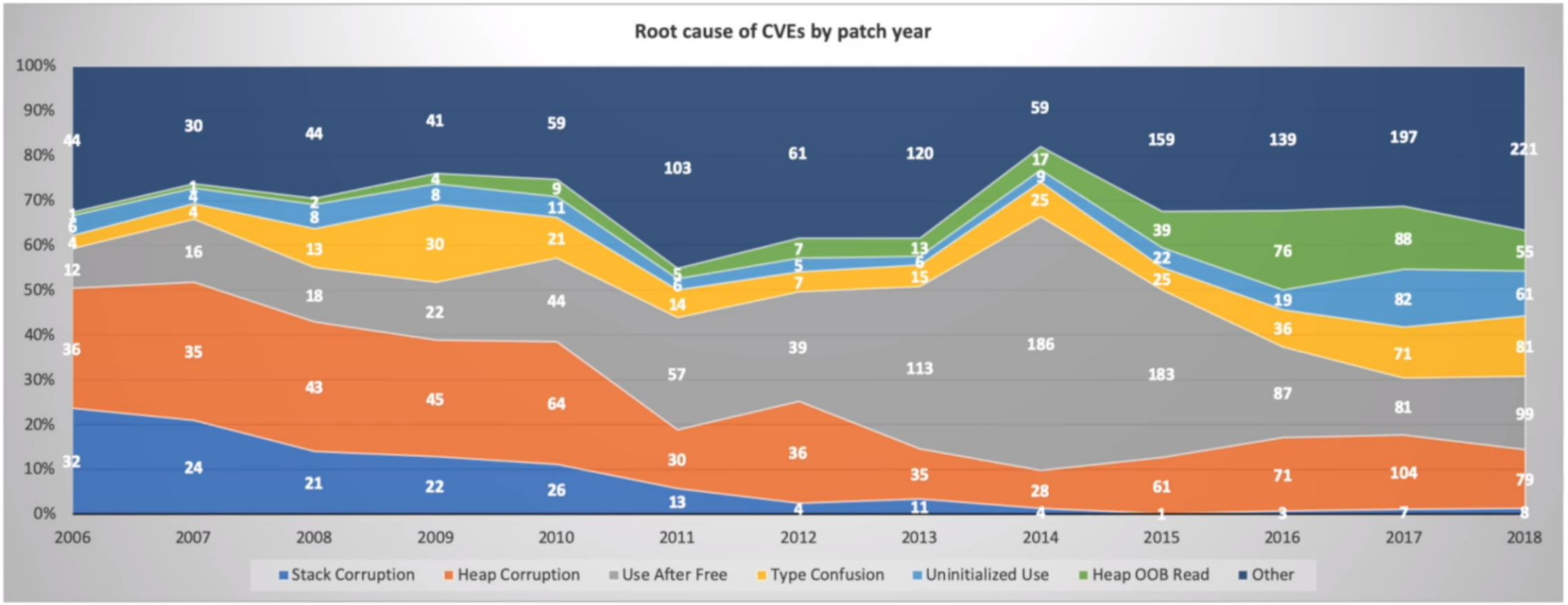


Figure 2: Breakdown of root causes for CVEs by year in Microsoft products.[2]Only 221 out 604 are not safety related.

[2] Digital Security by Design: Security and Legacy at Microsoft. https://vimeo.com/ 376180843, 2019.

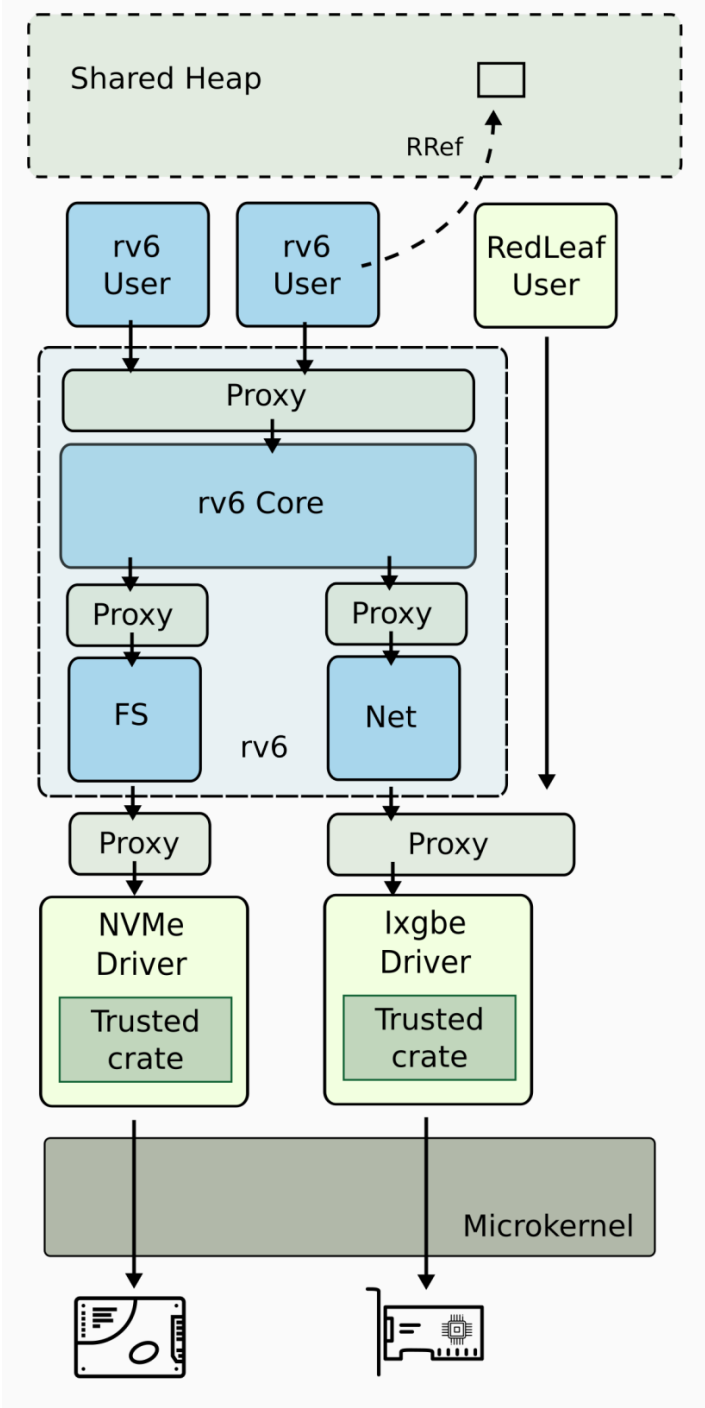
Rust is the first safe alternative to C for low-level systems code

- Safe code remains fast
  - No garbage collection
- Safety
  - 70% reduction in low-level vulnerabilities
- Lightweight fine-grained software isolation
  - Like in RedLeaf

- Device drivers
- Rv6, a POSIX-like operating system
  - A collection of domains
  - File system, network stack, and system calls
  - And user processes
- Device pass through
- Shared heap

All code runs in **Ring 0**

RedLeaf



Rust is the first safe alternative to C for low-level systems code

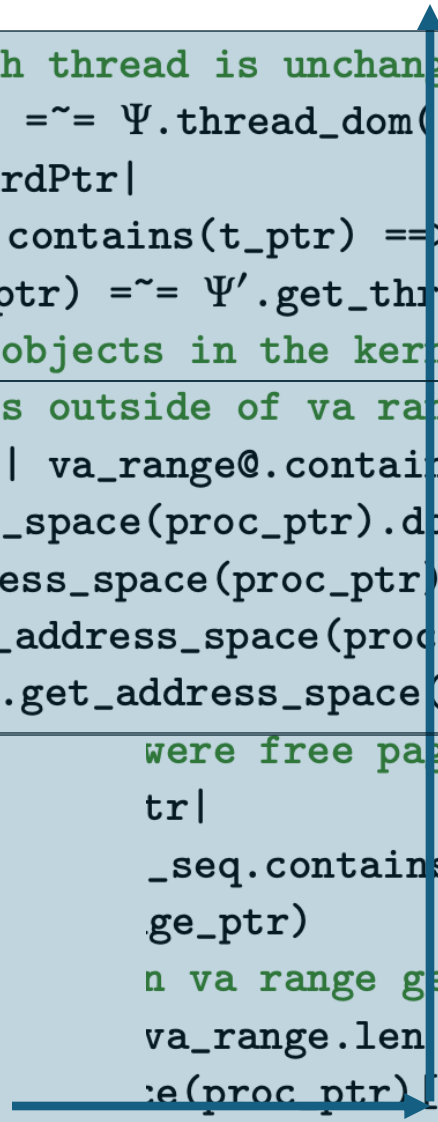
- Safe code remains fast
  - No garbage collection
- Safety
  - 70% reduction in low-level vulnerabilities
- Lightweight fine-grained software isolation
  - Like in RedLeaf
- Gateway to practical formal verification

Rust + Verus

```
1   pub struct PageTable{
2       pub cr3: usize
3       pub map: Ghost<Map<VAddr,MapEn
4       ... }
```

```
27   pub fn mmap(Ψ: &mut Kernal, t_ptr: ThrdPtr, va_range:
         VaRange4K) -> (ret: SyscallReturnStruct)
28     requires
29         old(Ψ).total_wf(), //global invariants
30         ...
31     ensures
32         syscall_mmap_spec(old(Ψ), Ψ, t_ptr, va_range, ret),
```

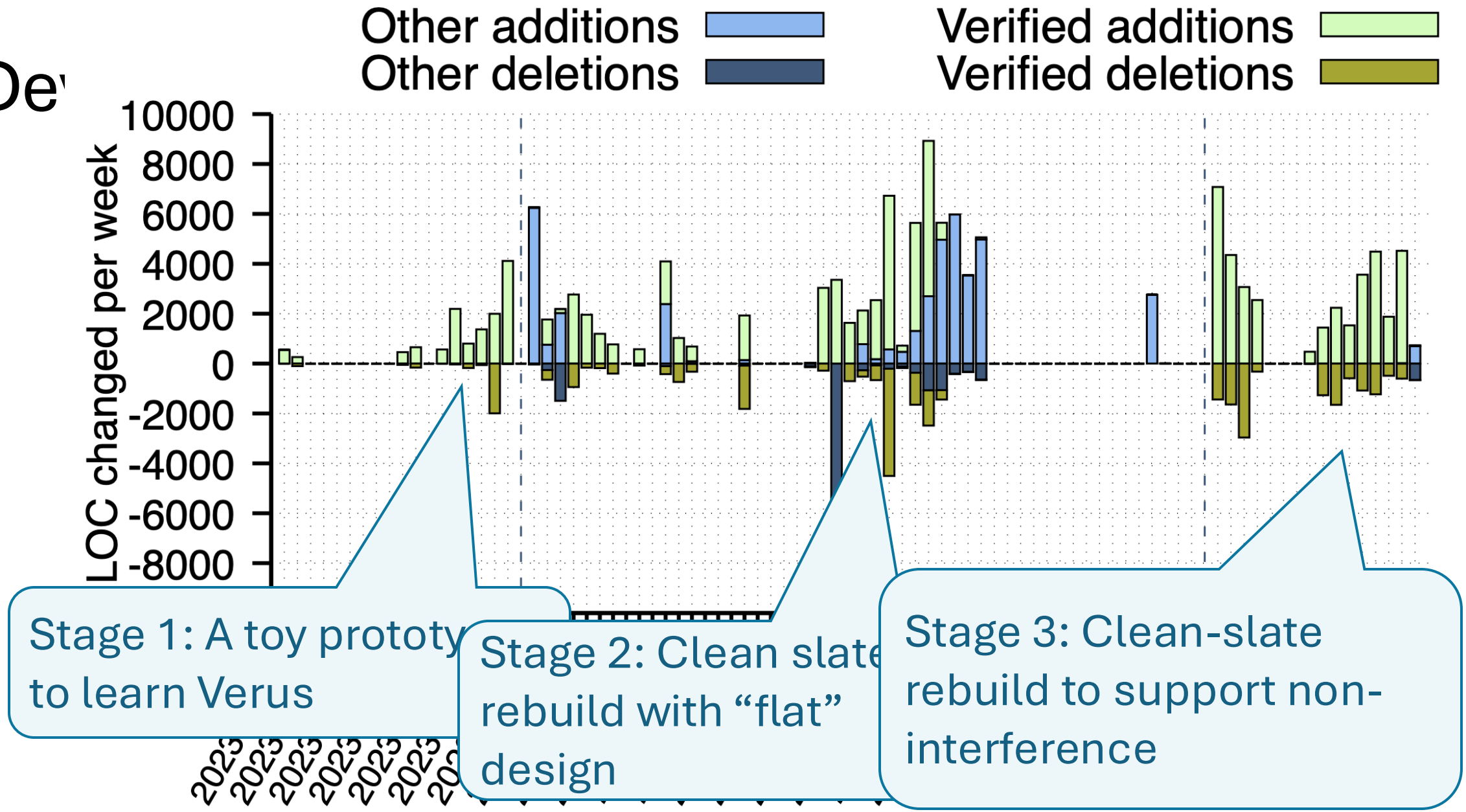
```
5   pub open spec fn syscall_mmap_spec(Ψ':Kernel, Ψ:Kernel, t_id
        : ThrdPtr, va_range: VaRange4K, perm_bits:MapEntryPerm,
        ret: SyscallReturnStruct) -> bool{
6     ...
7     // the state of each thread is unchanged
8     &&& Ψ'.thread_dom() =~= Ψ.thread_dom()
9     &&& forall|t_ptr:ThrdPtr|
10        Ψ'.thread_dom().contains(t_ptr) ==>
11        Ψ.get_thread(t_ptr) =~= Ψ'.get_thread(t_ptr)
12    ... // rest of the objects in the kernel
13    // virtual addresses outside of va range are not changed
14    &&& forall|va:VAddr| va_range@.contains(va) == false
15      ==> Ψ.get_address_space(proc_ptr).dom().contains(va)
16          == Ψ'.get_address_space(proc_ptr).dom().contains(va)
                && Ψ.get_address_space(proc_ptr)[va]
17                 =~= Ψ'.get_address_space(proc_ptr)[va]
                                    were free pages
                                    tr|
                                    _seq.contains(page_ptr)
                                    ge_ptr)
                                    n va range gets a unique page
                                    va_range.len
                                    e(proc_ptr)[va_range@[i]].addr
                  == mmapped_physical_pages_seq[i]
26    ...}
```

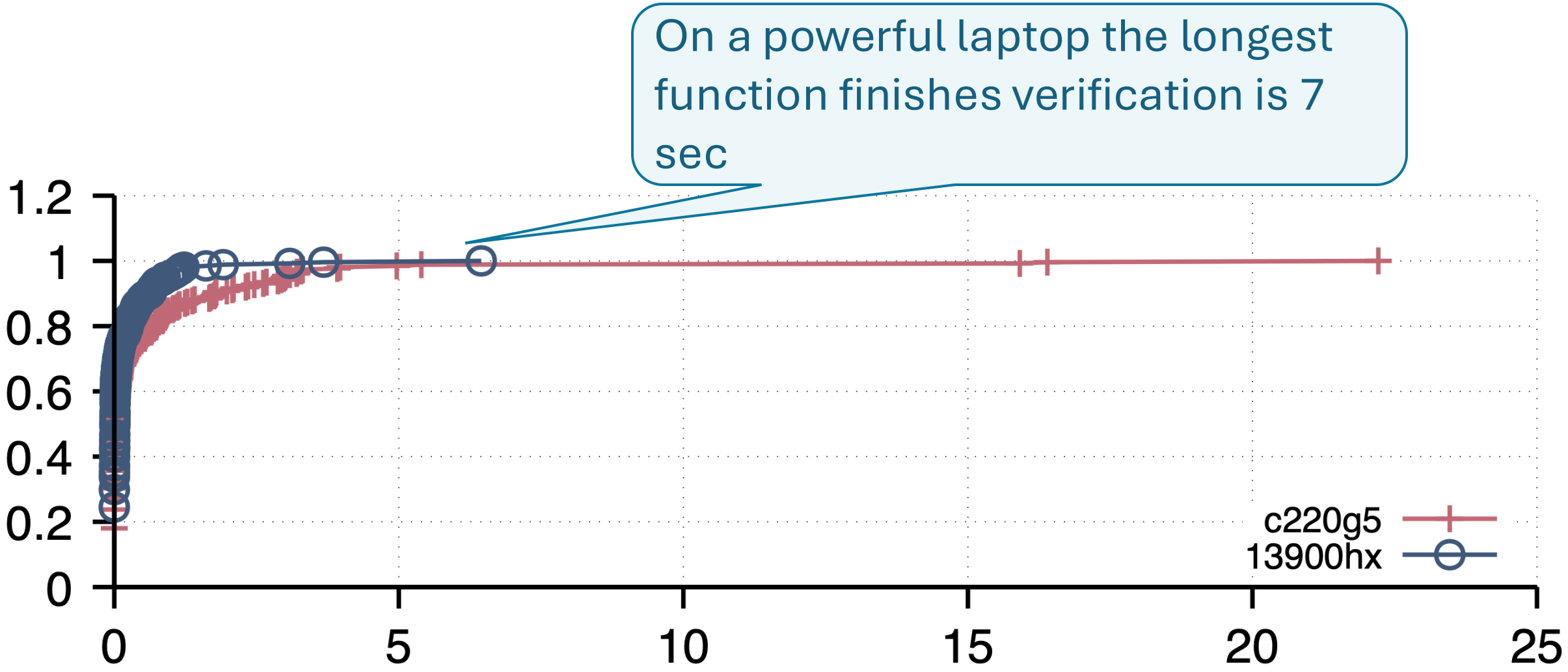


Proofs

- Functional correctness
  - System call specifications on the abstract state
  - Cross-cutting global invariants about all data structures in the kernel
- Safety and leak freedom
- Non-interference

De


Other additions
Other deletions
Verified additions
Verified deletions
LOC changed per week
10000
8000
6000
4000
2000
0
-2000
-4000
-6000
-8000
2023
2023
2023
2023
2023
2023
2023
Stage 1: A toy prototy
to learn Verus
Stage 2: Clean slate
rebuild with “flat”
design
Stage 3: Clean-slate
rebuild to support non-
interference



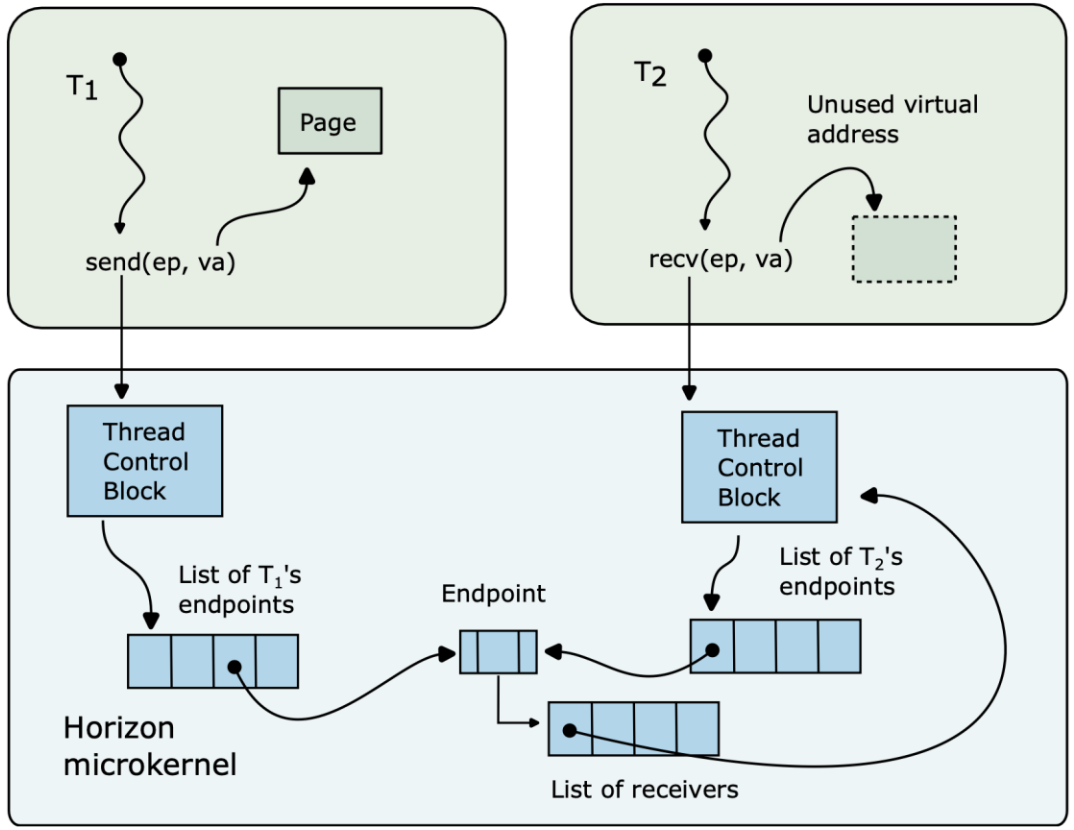
Verification time in seconds on a server (CloudLab c220g5) and a more powerful laptop (13900hx)

Proof effort

Name	Language	Spec Lang.	Proof-to-Code Ratio
seL4	C+Asm	Isabelle/HOL	20:1 [25]
CertiKOS	C+Asm	Coq	14.9:1 [19]
SeKVM	C+Asm	Coq	6.9:1 [30]
Ironclad	Dafny	Dafny	4.8:1 [22]
NrOS	Rust	Verus	10:1 [3]
VeriSMo	Rust	Verus	2:1 [42]
Atmosphere	Rust	Verus	3.2:1

Ok, what is our plan

We plan to build a small microkernel



- Conceptually, similar to classical microkernels
  - Processes, threads, endpoints for synchronous communication
  - Capability interface
- User-level device drivers

Thank you!