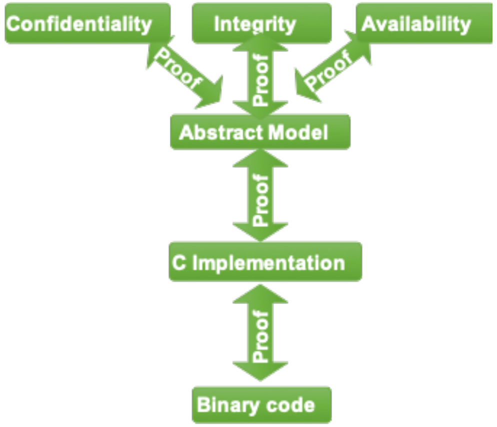
School of Computer Science & Engineering

COMP9242 Advanced Operating Systems

2019 T2 Week 09b

Local OS Research

@GernotHeiser



Copyright Notice

Quantifying Security Impact of Operating-System Design

Quantifying OS-Design Security Impact

Approach:

- Examine all ***critical*** Linux CVEs (vulnerabilities & exploits database)

- easy to exploit
- high impact
- no defence available
- confirmed

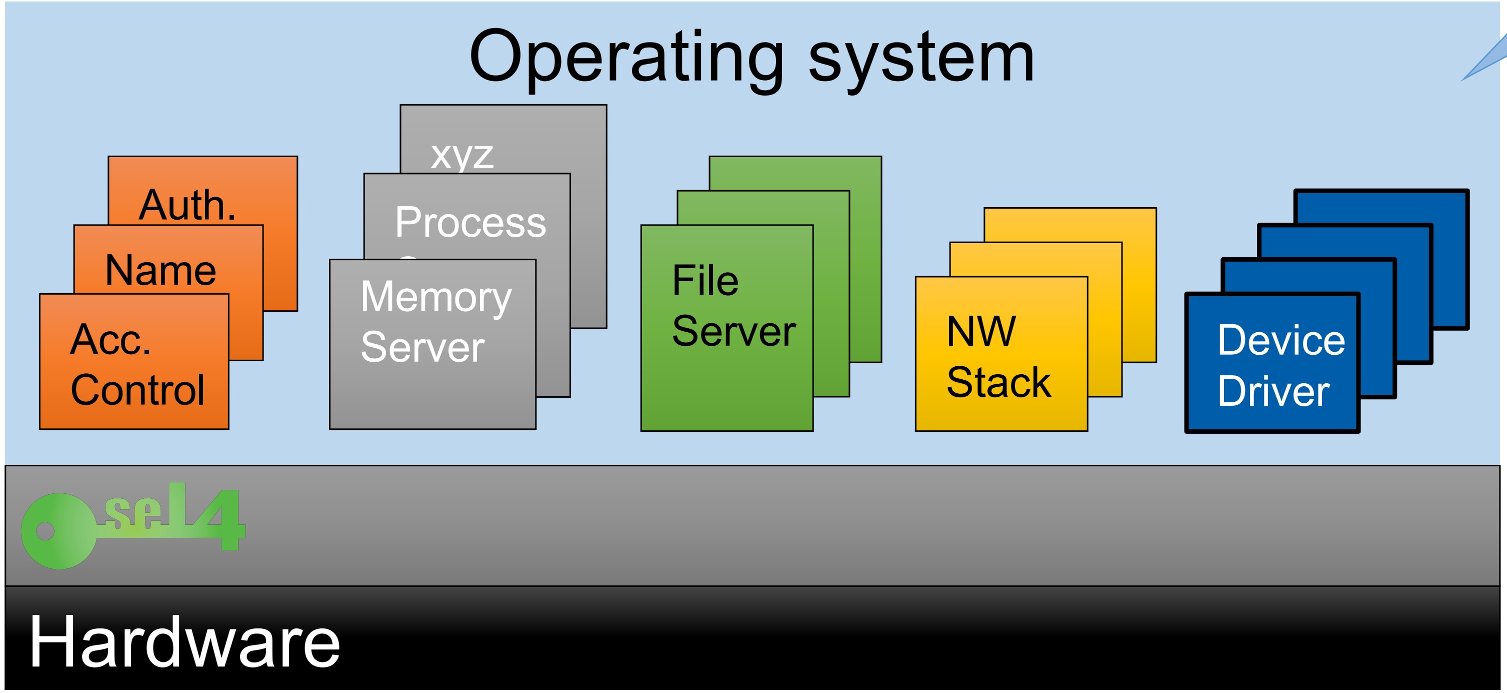
115 critical Linux CVEs to Nov'17

- For each establish how microkernel-based design would change impact

Hypothetical seL4-based OS

OS structured in *isolated* components, minimal inter-component dependencies, *least privilege*

Functionality comparable to Linux



Operating system

xyz

Process Server

Memory Server

Auth.

Name

Acc. Control

File Server

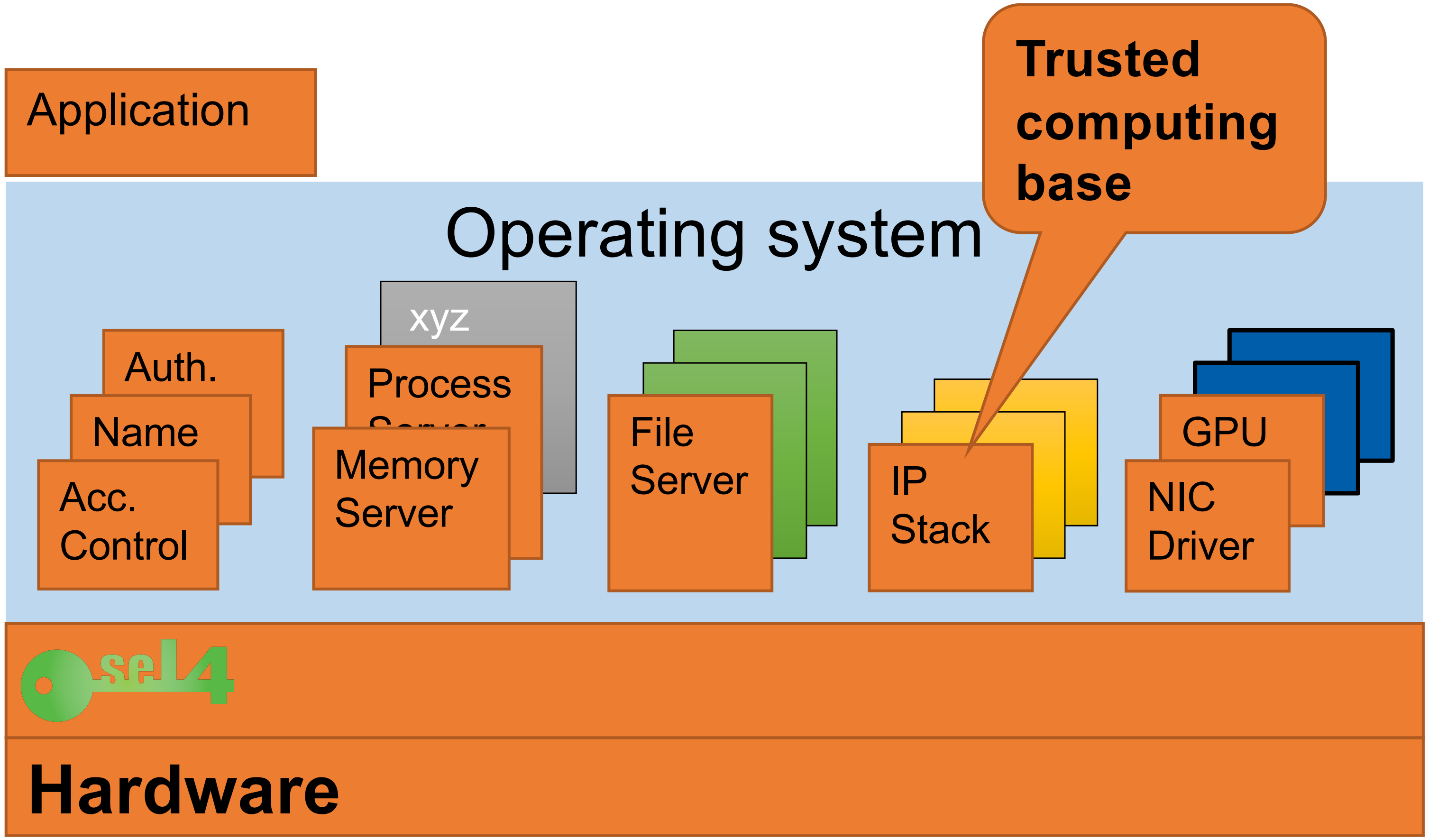
NW Stack

Device Driver

seL4

Hardware

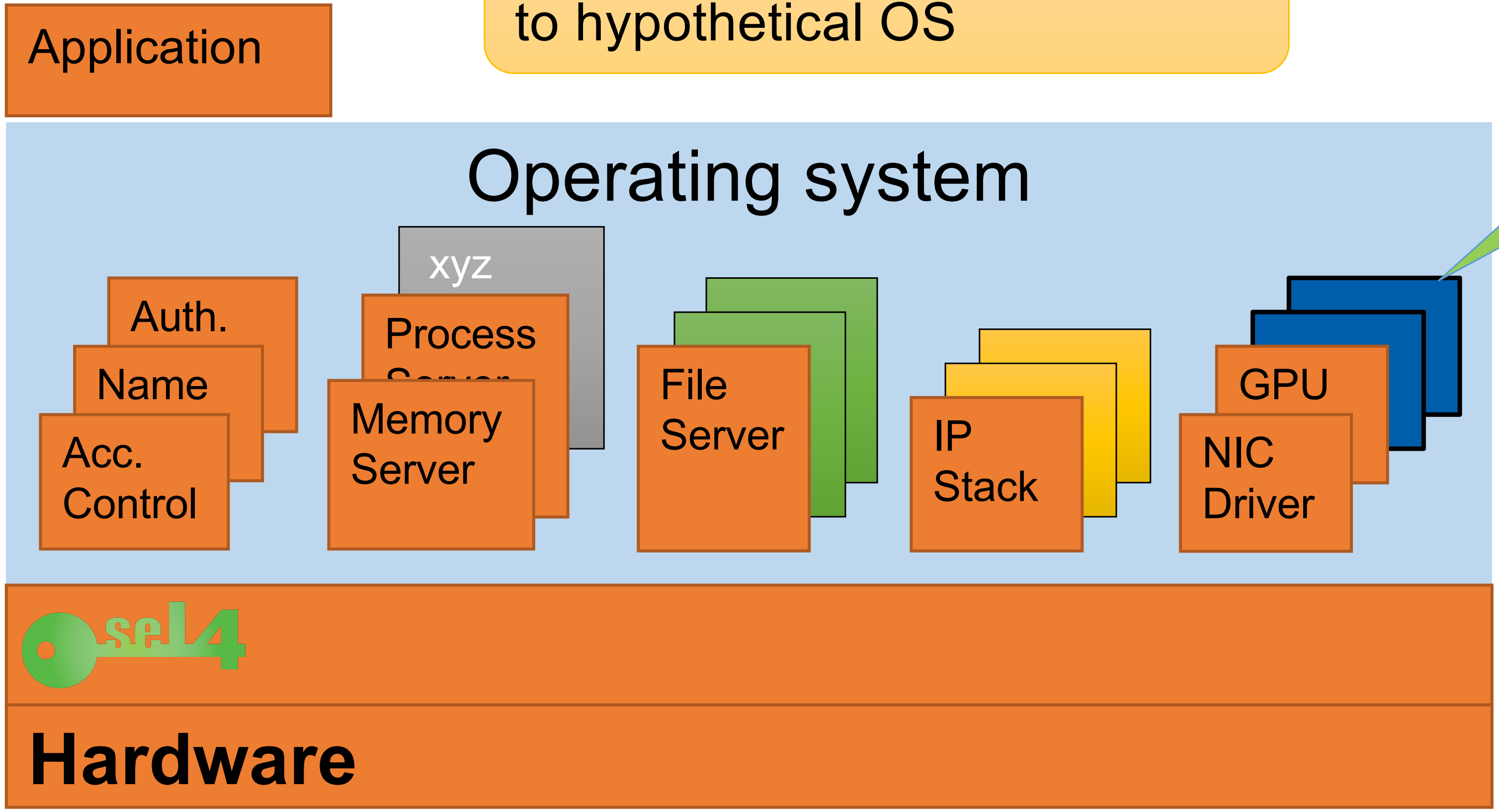
Hypothetical Security-Critical App



App requires:
- IP networking
- File storage
- Display output

Analysing CVEs

Map compromised component to hypothetical OS

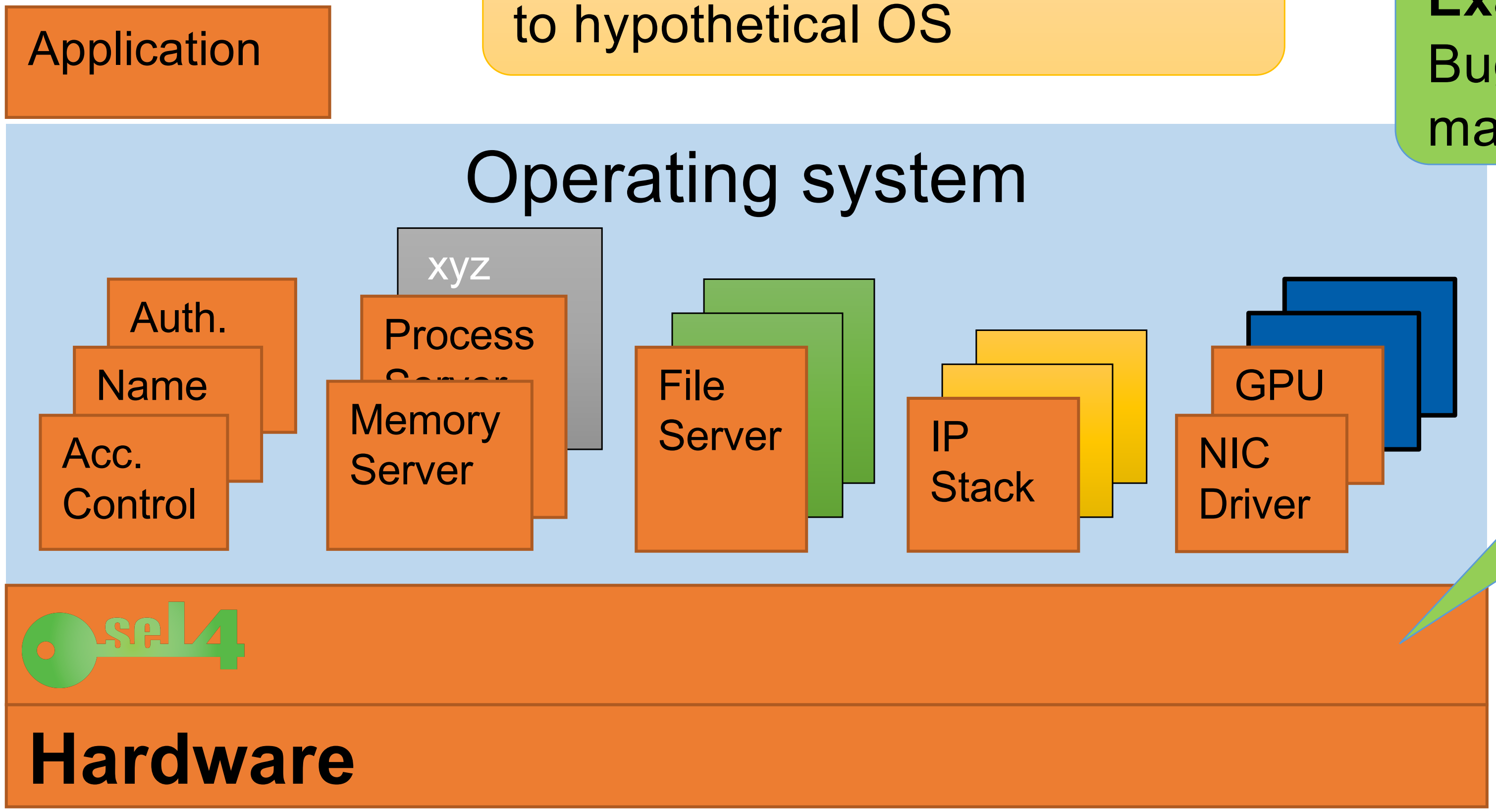


Not in TCB:
Attack defeated

Example:
USB driver bug

Analysing CVEs

Map compromised component to hypothetical OS

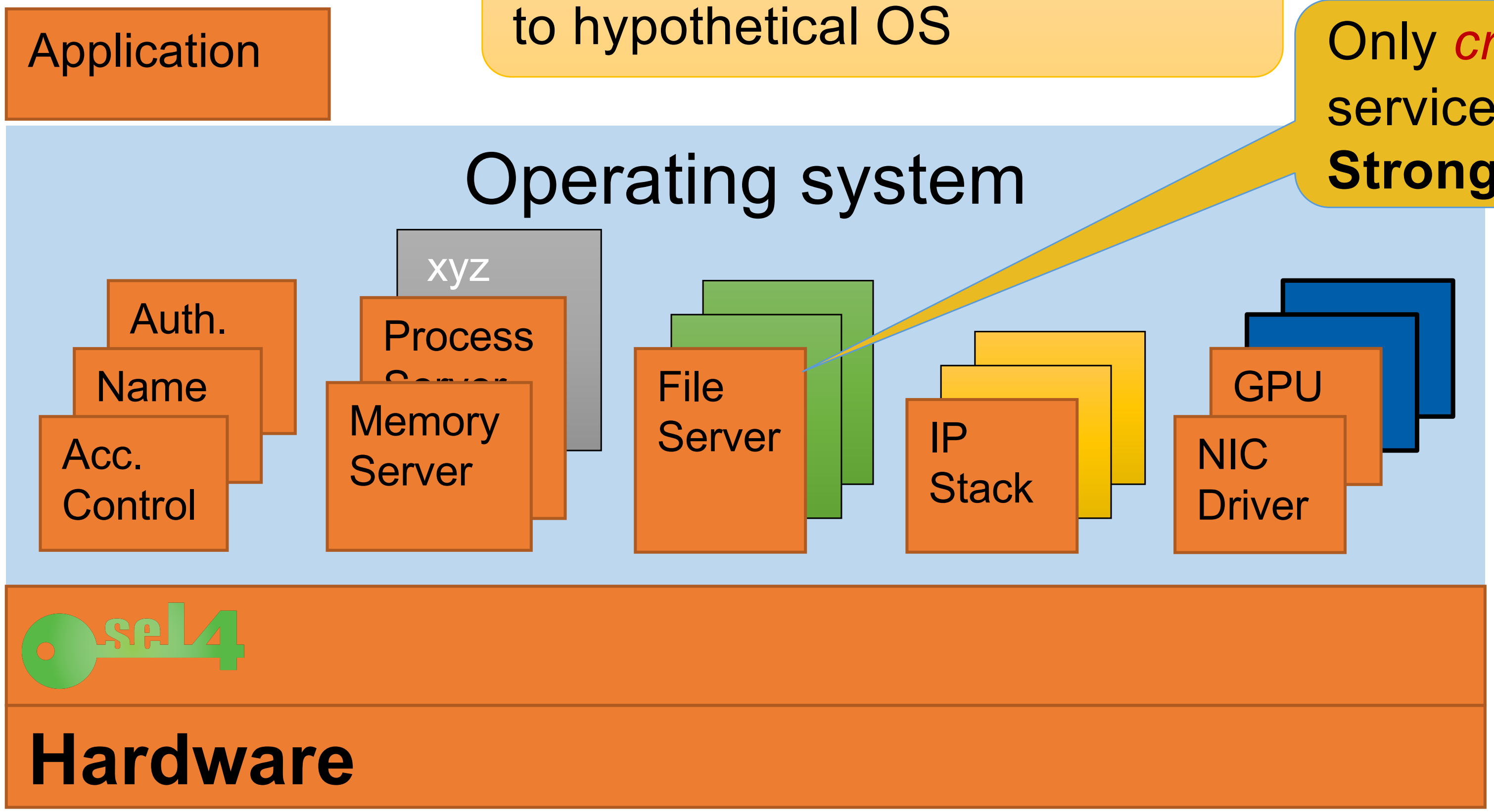


Example:
Bug in page-table management

In microkernel:
Attack defeated by verifiation

Analysing CVEs

Map compromised component to hypothetical OS

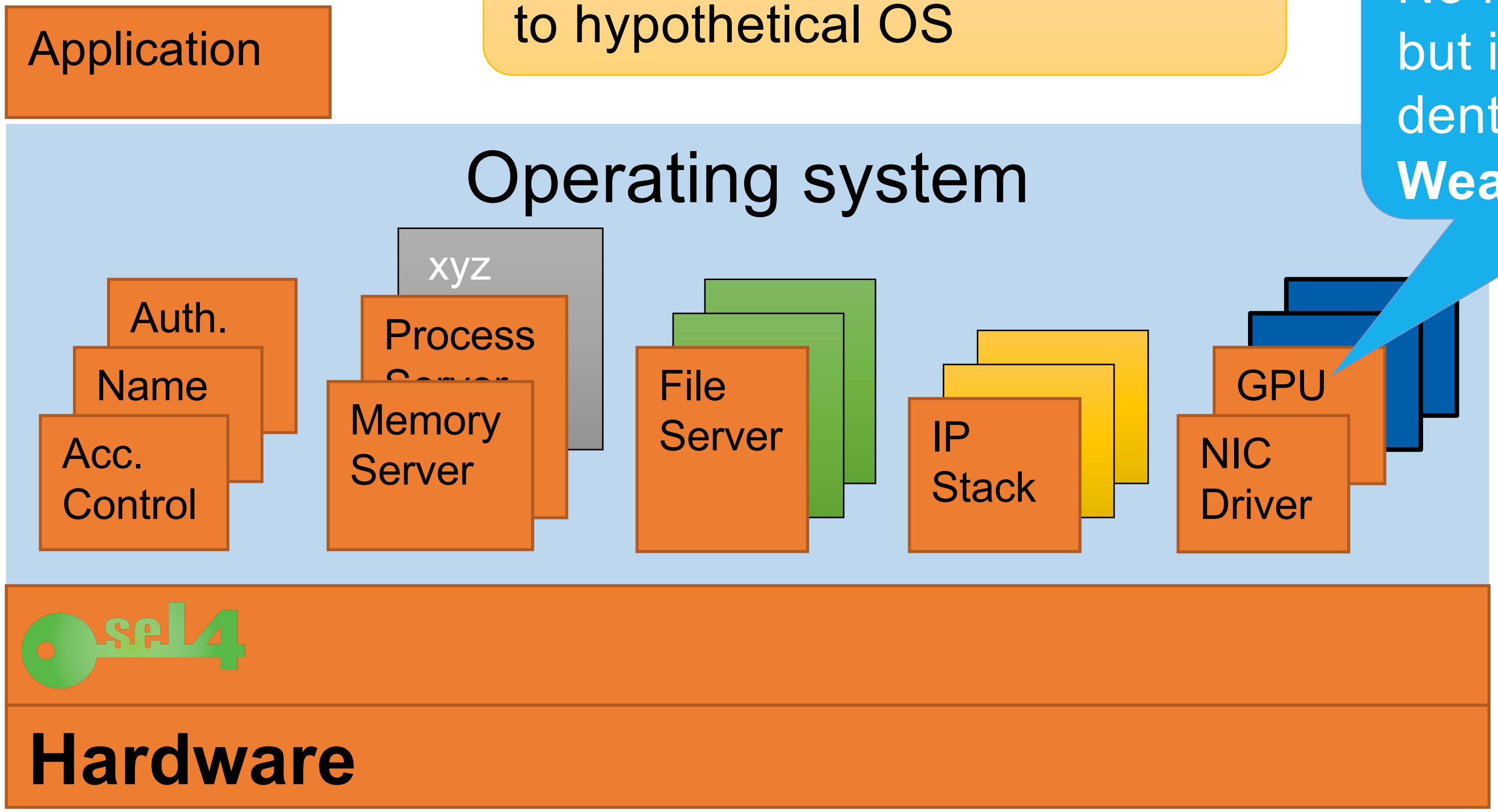


Only *crash* essential service (DoS): **Strongly mitigated**

Example: File system compromised

Analysing CVEs

Map compromised component to hypothetical OS

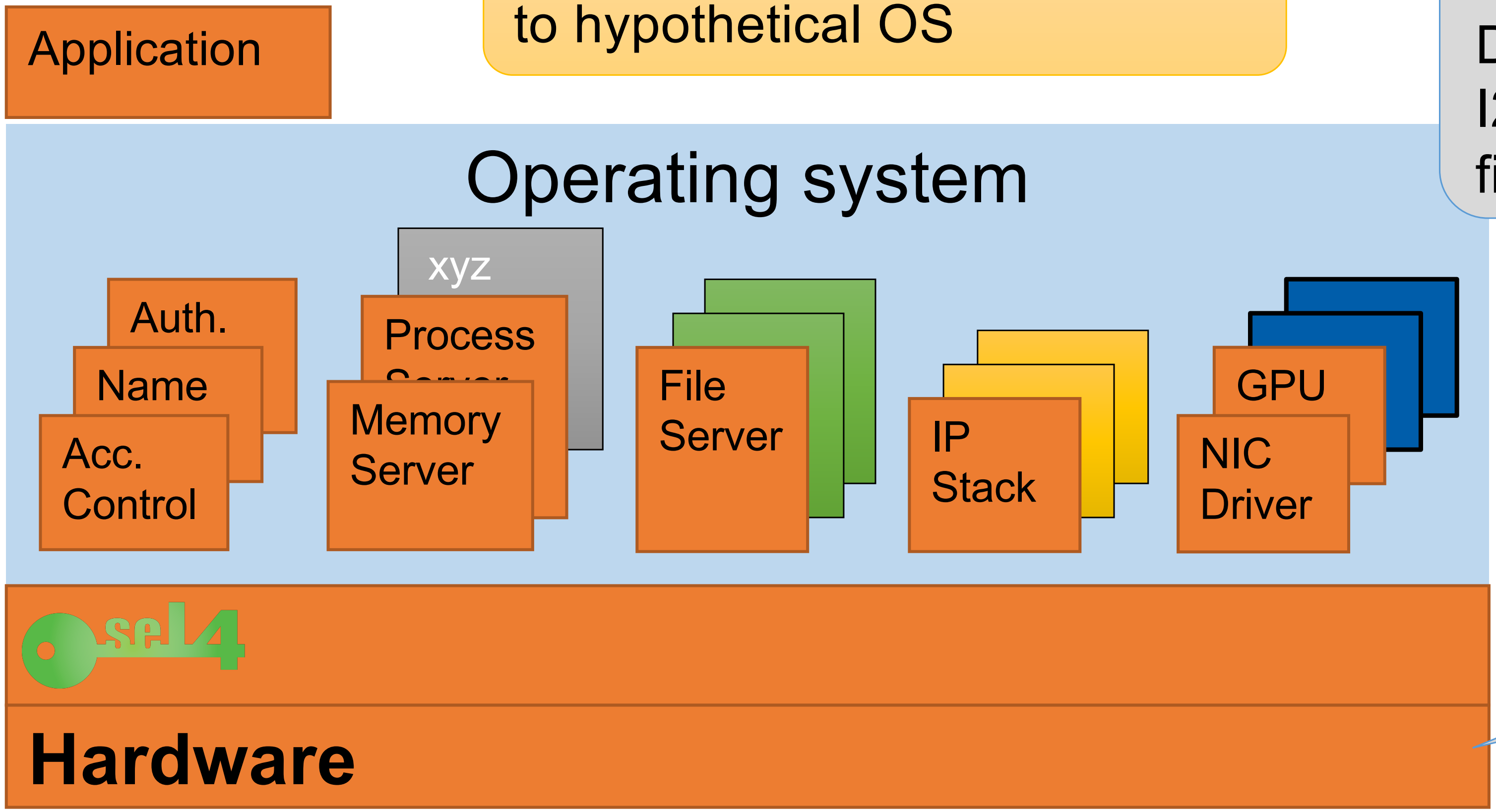


No full compromise but integrity or confidentiality violation: **Weakly mitigated**

Example: GPU compromised

Analysing CVEs

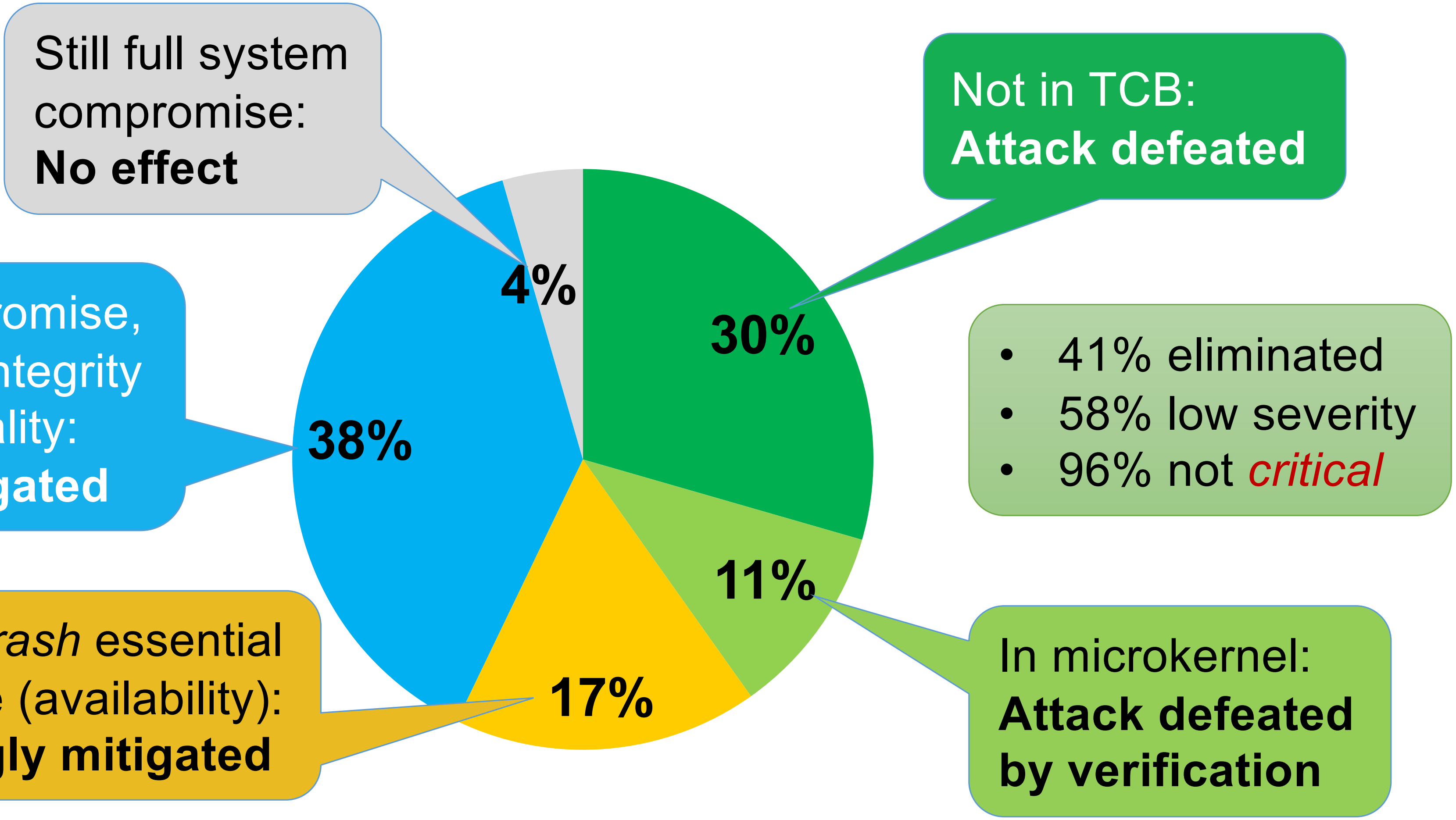
Map compromised component to hypothetical OS



Example:
Driver exploit hijacks I2C bus, allowing firmware reflush

Full system compromise:
No effect

All Critical Linux CVEs to 2017



- 41% eliminated
- 58% low severity
- 96% not *critical*

Summary

OS structure matters!

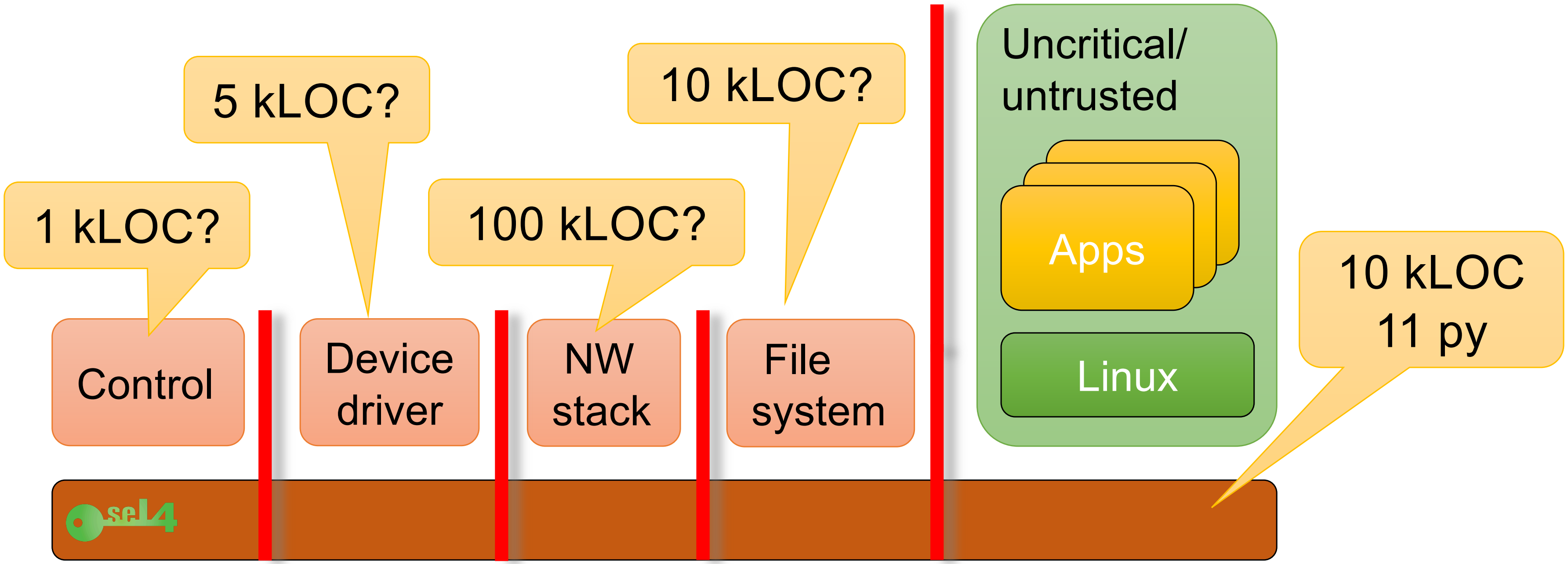
- Microkernels definitely improve security
- Monolithic OS design is *fundamentally flawed from security point of view*

[Biggs et al., APSys'18]

Use of a monolithic OS in security- or safety-critical scenarios is professional malpractice!

Cogent

Beyond the Kernel



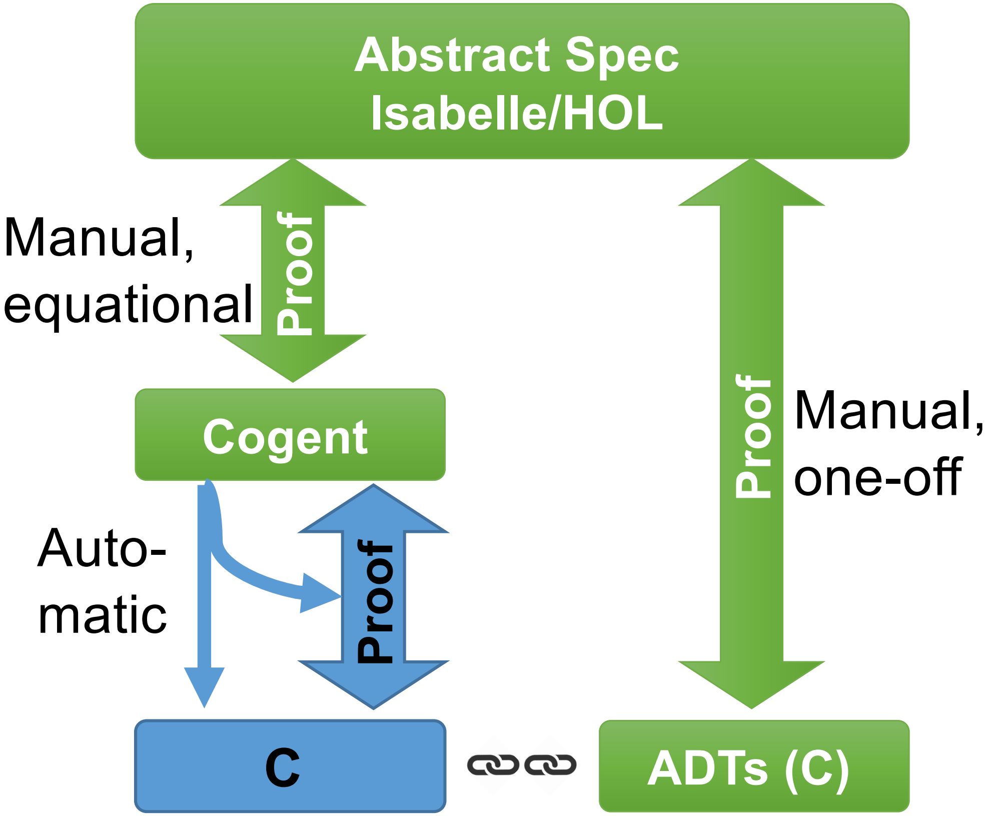
Aim: Verified TCB at affordable cost!

Cogent: Code & Proof Co-Generation

Aim: Reduce cost of verified systems code

- Restricted, purely functional *systems* language
- Type- and memory safe, not managed
- Turing incomplete
- File system case-studies: BilbyFs, ext2, F2FS, VFAT

[O'Connor et al, ICFP'16; Amani et al, ASPLOS'16]

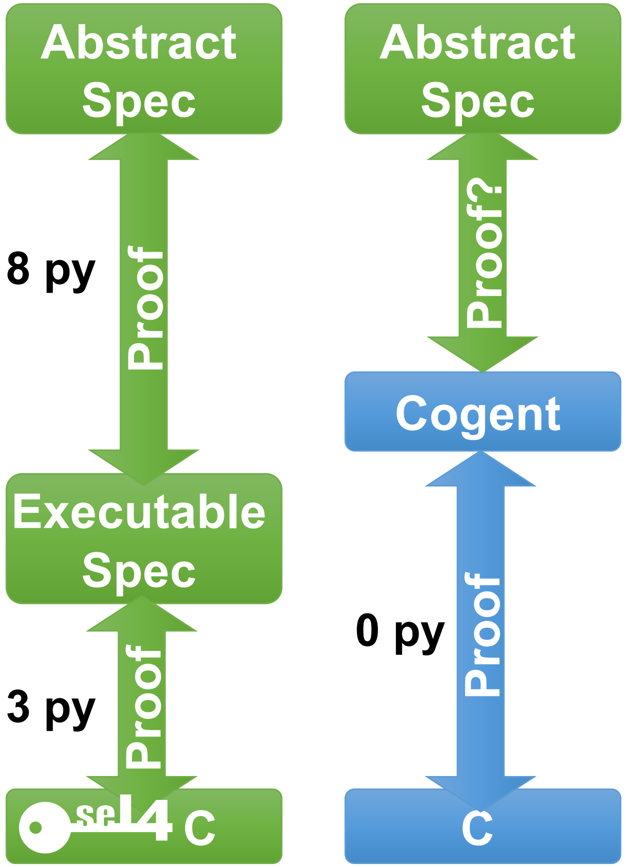


Manual Proof Effort

BilbyFS functions	Effort	Isabelle LoP	Cogent SLoC	Cost $/SLoC	LoP/ SLOC
isync()/ iget() library	9.25 pm	13,000	1,350	150	10
sync()-specific	3.75 pm	5,700	300	260	19
iget()-specific	1 pm	1,800	200	100	9
seL4	12 py	180,000	8,700 C	350	20

BilbyFS: 4,200 LoC Cogent

Addressing Verification Cost



Dependability-cost tradeoff:

- Reduced faults through safe language
- Property-based testing (QuickCheck)
- Model checking
- Full functional correctness proof

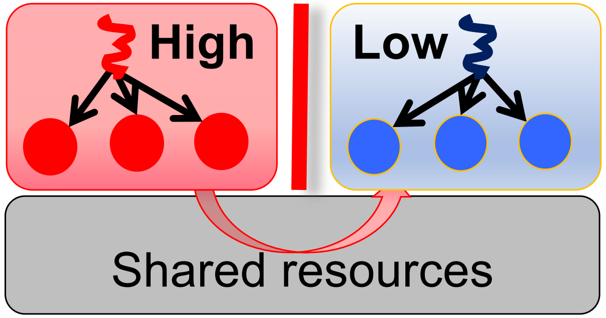
Spec reuse!

Work in progress:

- Language expressiveness
- Reduce boiler-plate code
- Network stacks
- Device drivers

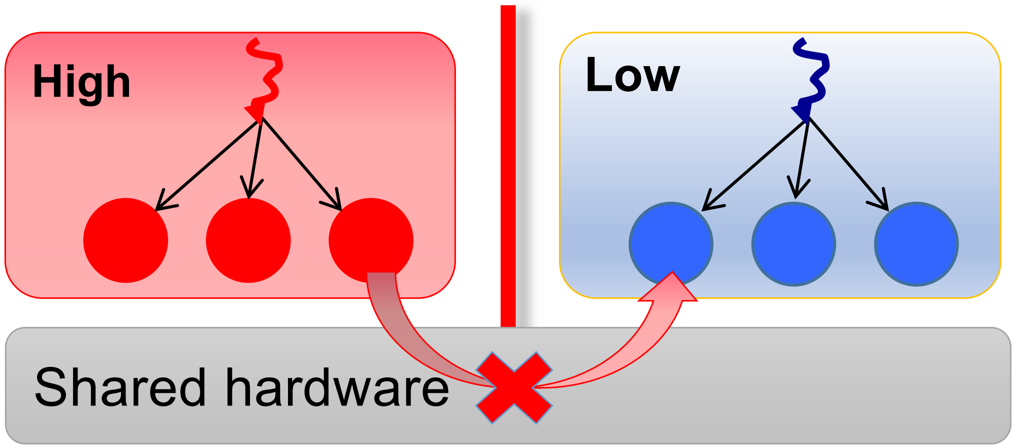
Time Protection

Refresh: Microarchitectural Timing Channels



Contention for shared hardware resources affects execution speed, leading to timing channels

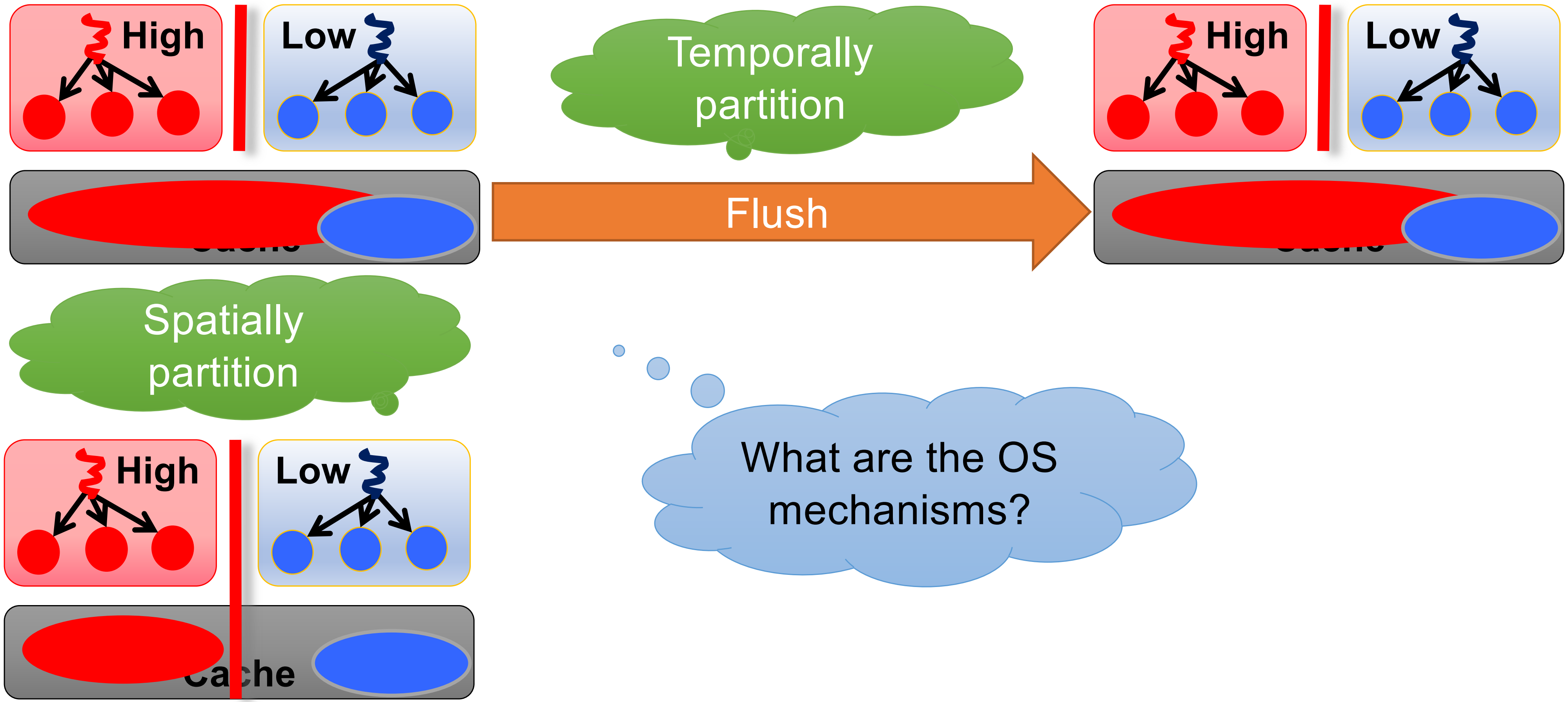
OS Must Enforce *Time Protection*



Preventing interference is core duty of the OS!

- *Memory protection* is well established
- *Time protection* is completely absent

Time Protection: No Sharing of HW State

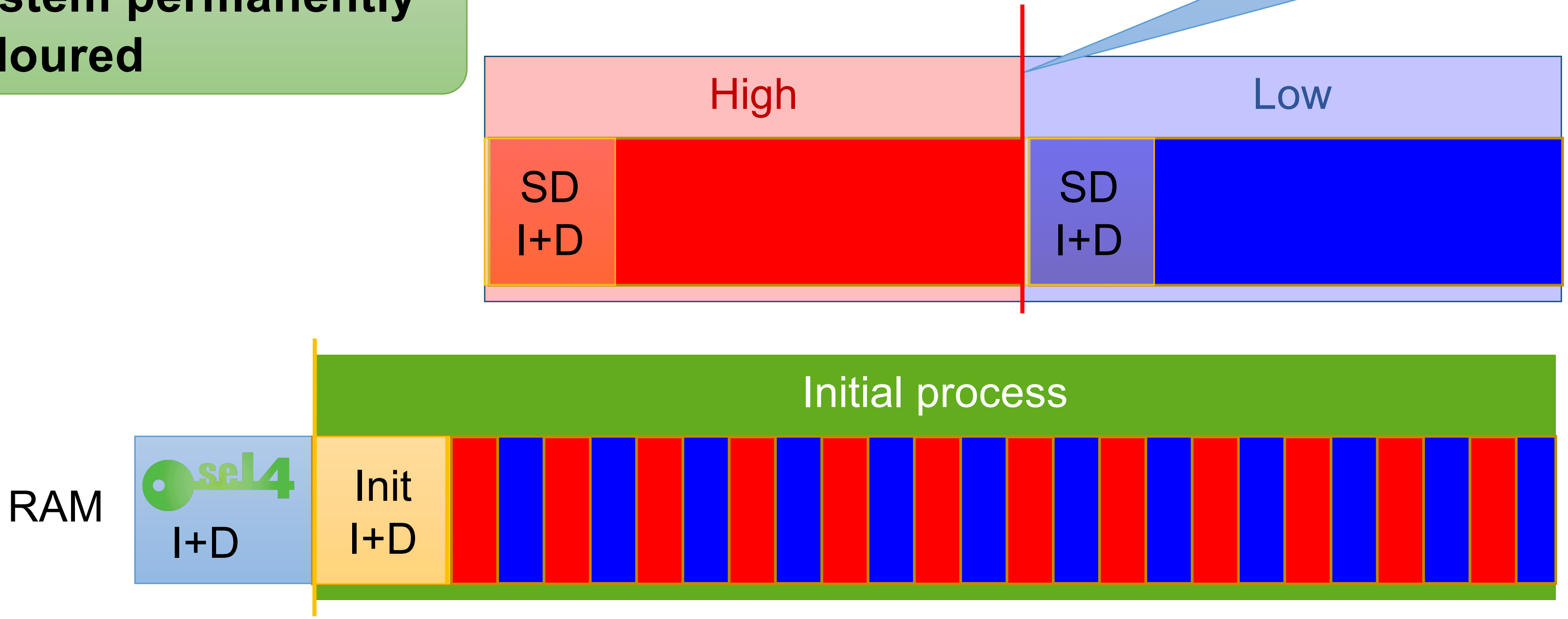


What are the OS mechanisms?

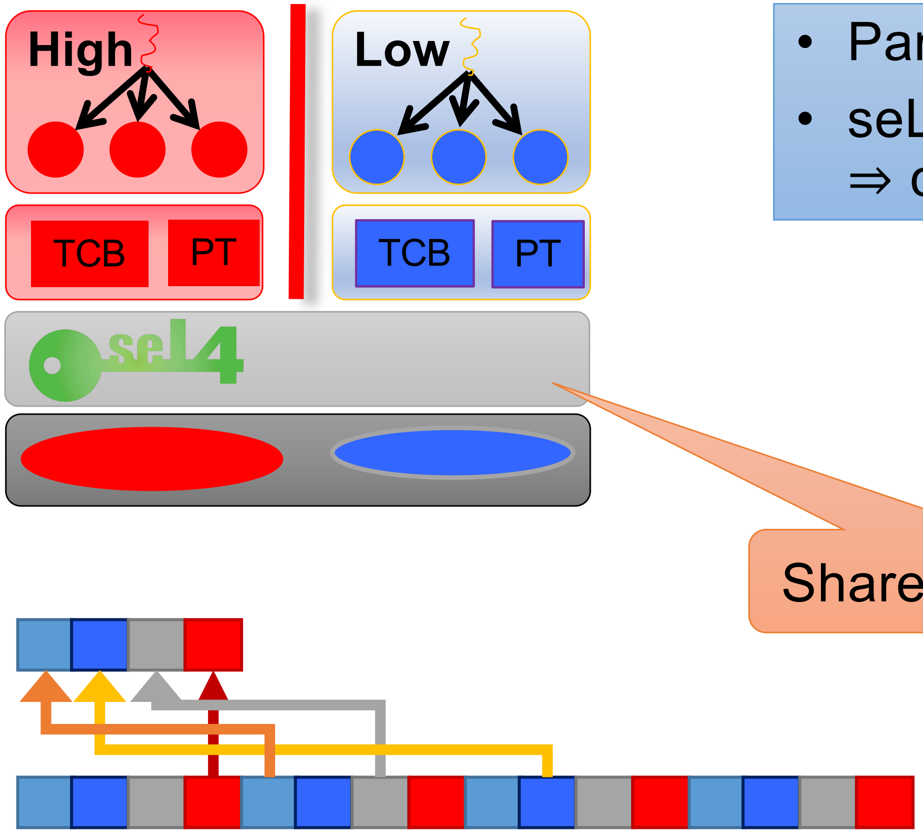
Spatial Partitioning: Cache Colouring

System permanently coloured

Partitions restricted to coloured memory



Spatial Partitioning: Cache Colouring

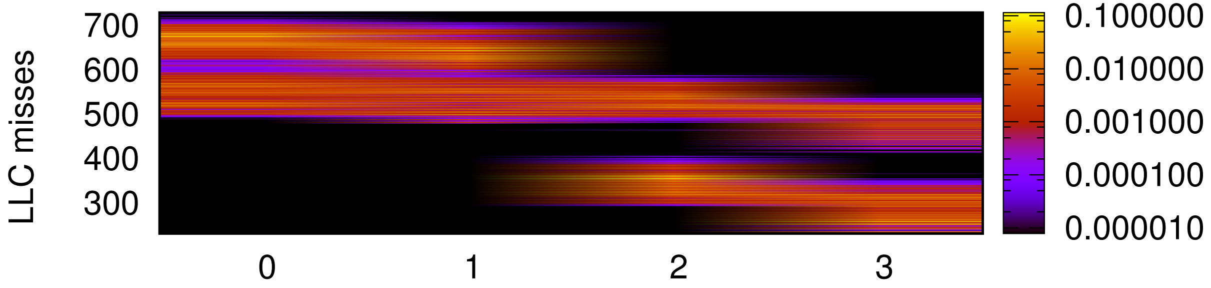


- Partitions get frame pools of disjoint colours
- seL4: userland supplies kernel memory
⇒ colouring userland colours kernel memory

Shared kernel image

Channel Through Kernel Code

Raw channel



Channel matrix: Conditional probability of observing output signal (time) given input signal (system-call number)

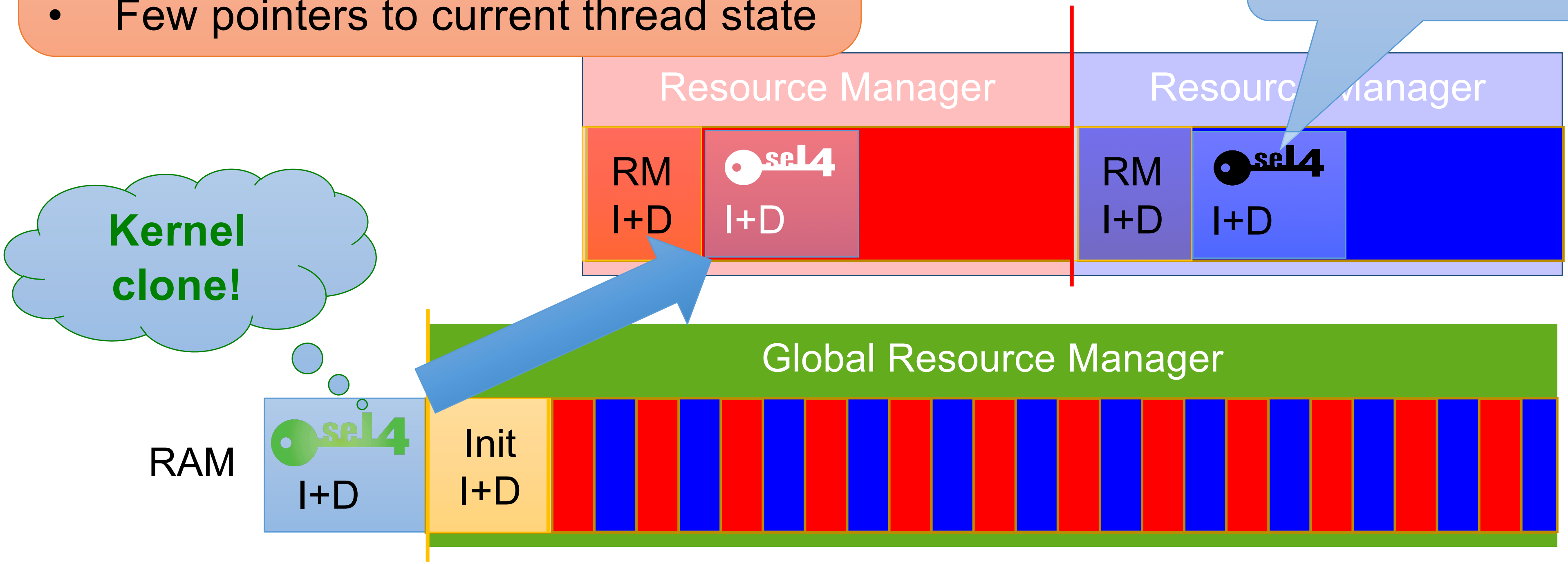
Colouring the Kernel

Remaining shared kernel data:

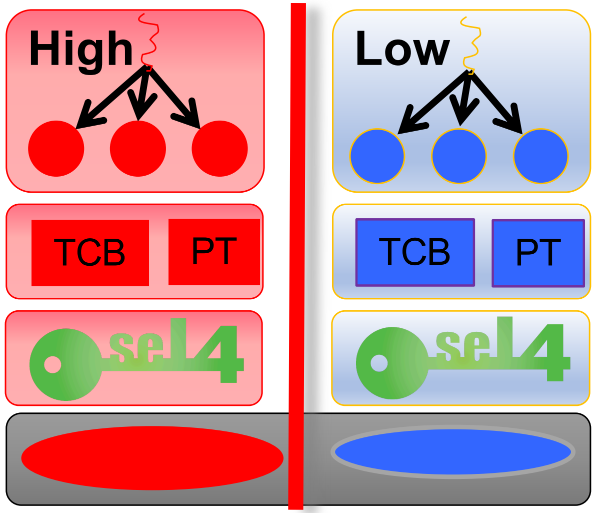
- Scheduler queue array & bitmap
- Few pointers to current thread state

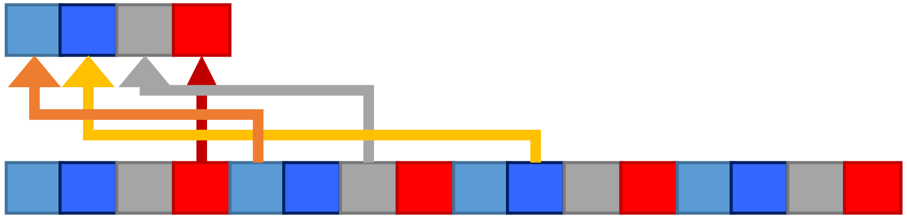
Ensure deterministic access!

Each partition has own kernel image



Spatial Partitioning: Cache Colouring





- Partitions get frame pools of disjoint colours
- seL4: userland supplies kernel memory
  ⇒ colouring userland colours kernel memory
- Per-partition kernel image to colour kernel

Ensure deterministic access!

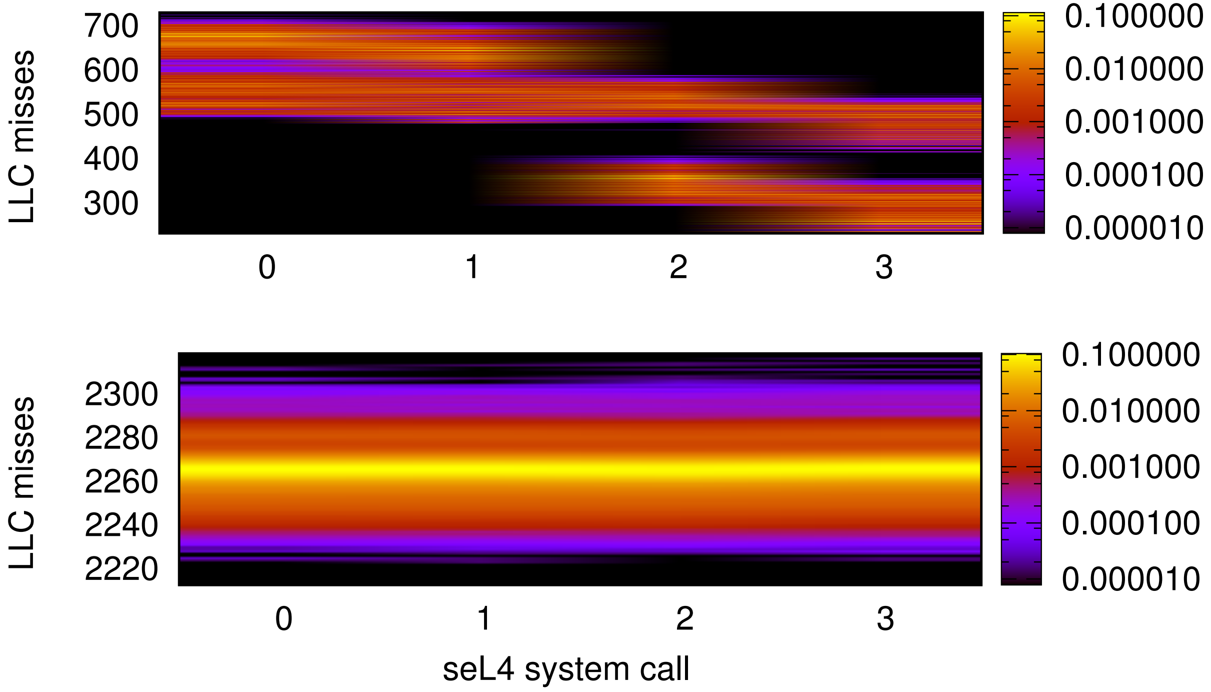
Remaining shared kernel data:

- Scheduler queue array & bitmap
- Few pointers to current thread state

Channel Through Kernel Code

Raw channel

Channel with cloned kernel



Temporal Partitioning: Flush on Switch

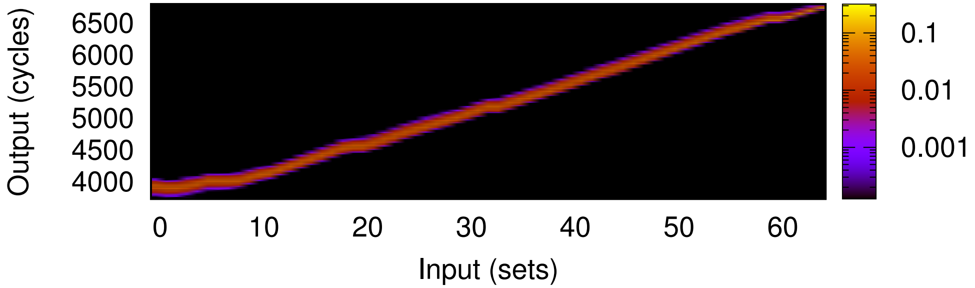
Must remove any history dependence!

2. Switch user context
3. Flush on-core state

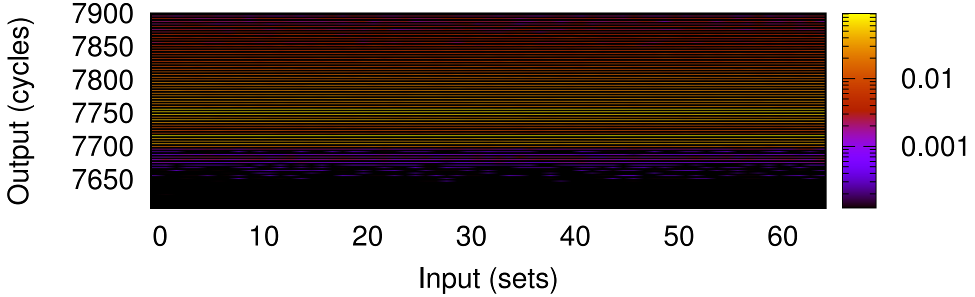
6. Reprogram timer
7. return

D-Cache Channel

Raw channel

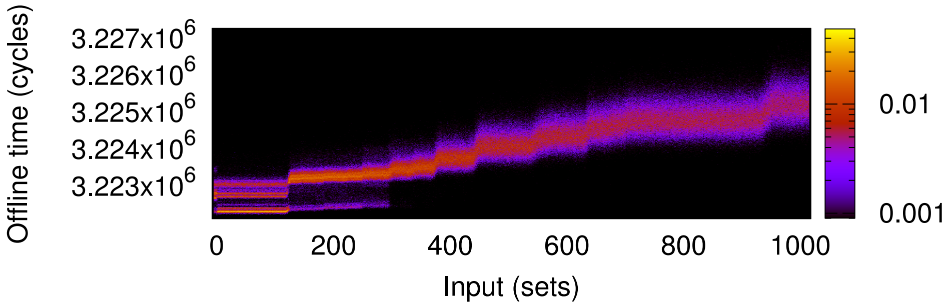


Channel with flushing



Flush-Time Channel

Raw channel



Temporal Partitioning: Flush on Switch

Must remove any history dependence!

1. T_0 = current_time()
2. Switch user context
3. Flush on-core state
4. Touch all shared data needed for return
5. while (T_0+WCET < current_time()) ;
6. Reprogram timer
7. return

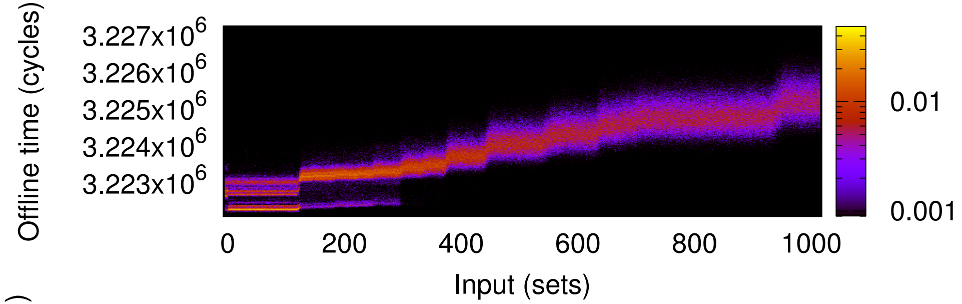
Latency depends on prior execution!

Time padding to remove dependency

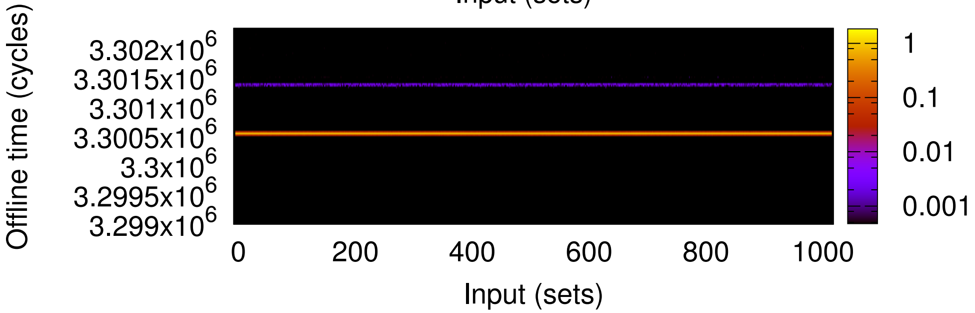
Ensure deterministic execution

Flush-Time Channel

Raw channel

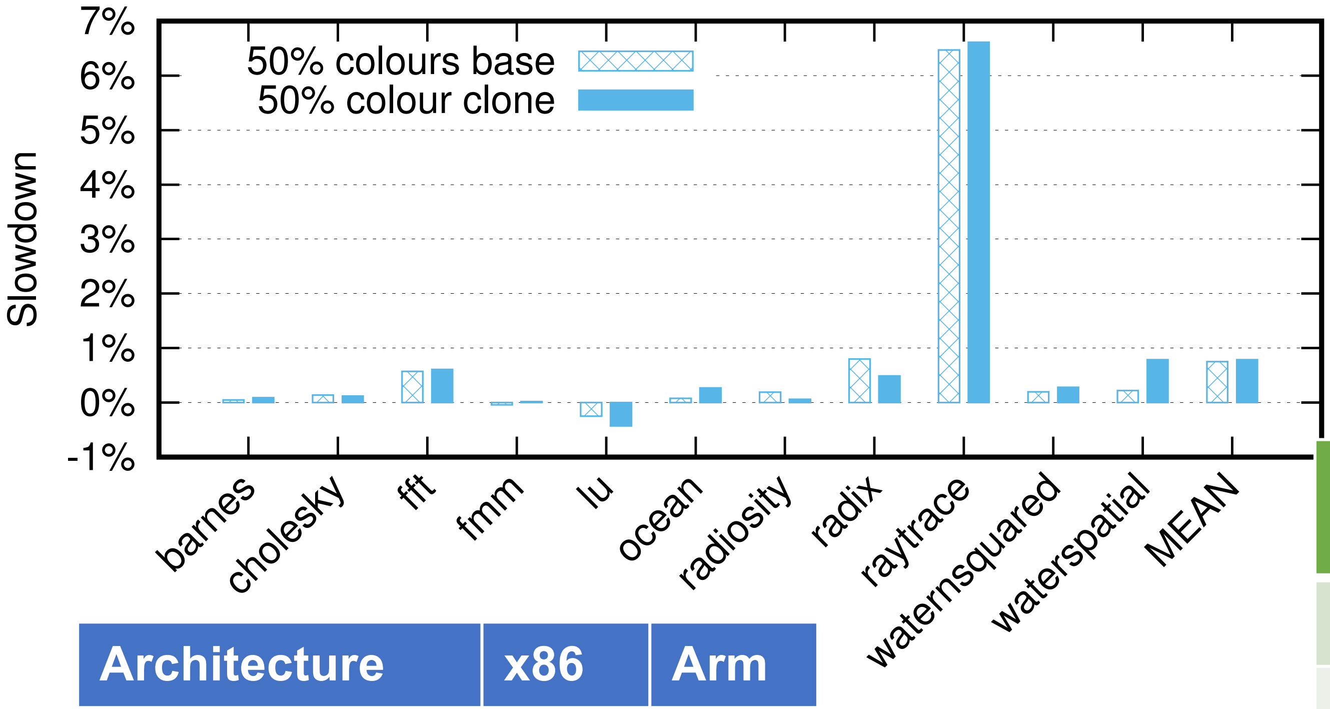


Channel with deterministic flushing



Performance Impact of Colouring

Splash-2 benchmarks on Arm A9



- Overhead mostly low
- Not evaluated is cost of not using super pages

[Ge et al., EuroSys'19]

Architecture	x86	Arm
Mean slowdown	3.4%	1.1%

Arch	seL4 clone	Linux fork+exec
x86	79 µs	257 µs
Arm	608 µs	4,300 µs

A New HW/SW Contract

aISA: augmented ISA

For all shared microarchitectural resources:

1. Resource must be spatially partitionable or flushable
2. Concurrently shared resources must be spatially partitioned
3. Resource accessed solely by virtual address must be flushed and not concurrently accessed
4. Mechanisms must be sufficiently specified for OS to partition or reset
5. Mechanisms must be constant time, or of specified, bounded latency
6. Desirable: OS should know if resettable state is derived from data, instructions, data addresses or instruction addresses

Cannot share HW threads across security domains!

[Ge et al., APSys'18]

Can Time Protection Be Verified?

1. Correct treatment of spatially partitioned state:
   - Need hardware model that identifies all such state (augmented ISA)
   - To prove:
     No two domains can access the same physical state

Functional property!

Transforms timing channels into storage channels!

2. Correct flushing of time-shared state
   - Not trivial: eg proving all cleanup code/data are forced into cache after flush
     - Needs an actual cache model
   - Even trickier: need to prove padding is correct
     - … without explicitly reasoning about time!

Functional property!

seL4 Verifying Time Padding

- Idea: Minimal formalisation of hardware clocks (abstract time)
  - Monotonically-increasing counter
  - Can add constants to time values
  - Can compare time values

To prove: padding loop terminates as soon as timer value ≥ T_0+WCET

Functional property

[Heiser et al., HotOS'19]

Making COTS Hardware Dependable

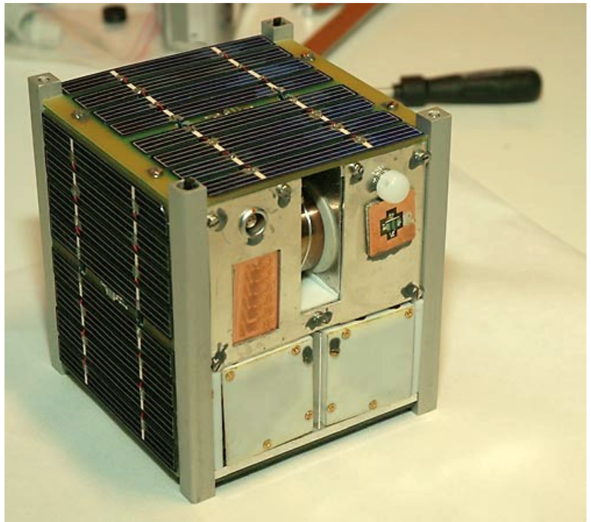
Satellites: SWaP vs Dependability

Space is becoming commodisized:

- many, small (micro-) satellites
- increasing cost pressure

Harsh evironment for electronics:

- temperature fluctuations
- ionising radiation

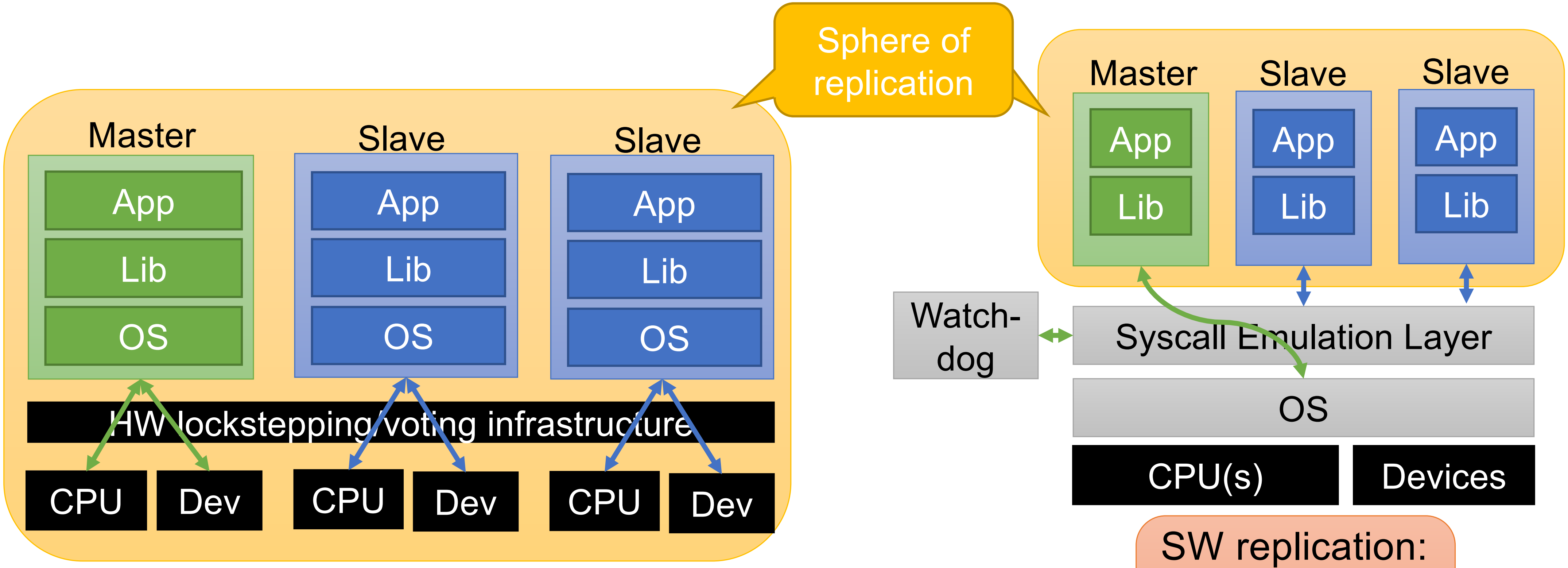


NCUBE2 by Bjørn Pedersen, NTNU (CC BY 1.0)

Radiation-hardened processors are slow, bulky and expensive

Use redundancy of cheap COTS multicores

Traditional Redundancy Approaches



Fault-tolerant HW:

- expensive

SW replication:

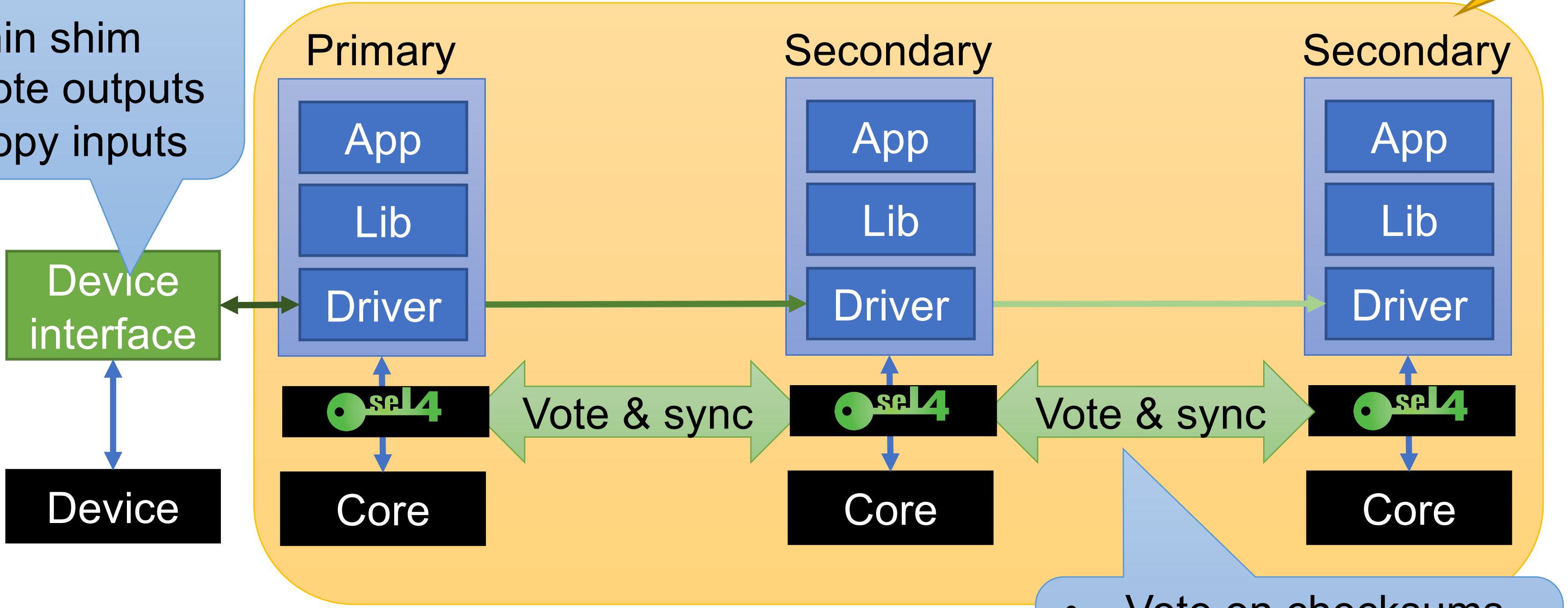
- cheap
- incomplete

Redundant Co-Execution (RCoE)

Userland transparently replicated

Sphere of replication

Device access:
- thin shim
- vote outputs
- copy inputs



No master-slave, but peer-to-peer

- Vote on checksums of arguments & state
- Logical time for sync

RCoE: Two Variants

Loosely-coupled RCoE

- Sync on syscalls & exceptions
- Preemptions in usermode not further synchronised (imprecise)

- Low overhead
- Cannot support racy apps, threads, virtual machines

Closely-coupled RCoE

- Sync on instruction
- Precise preemptions

- High overhead
- Supports all apps
- May need re-compile

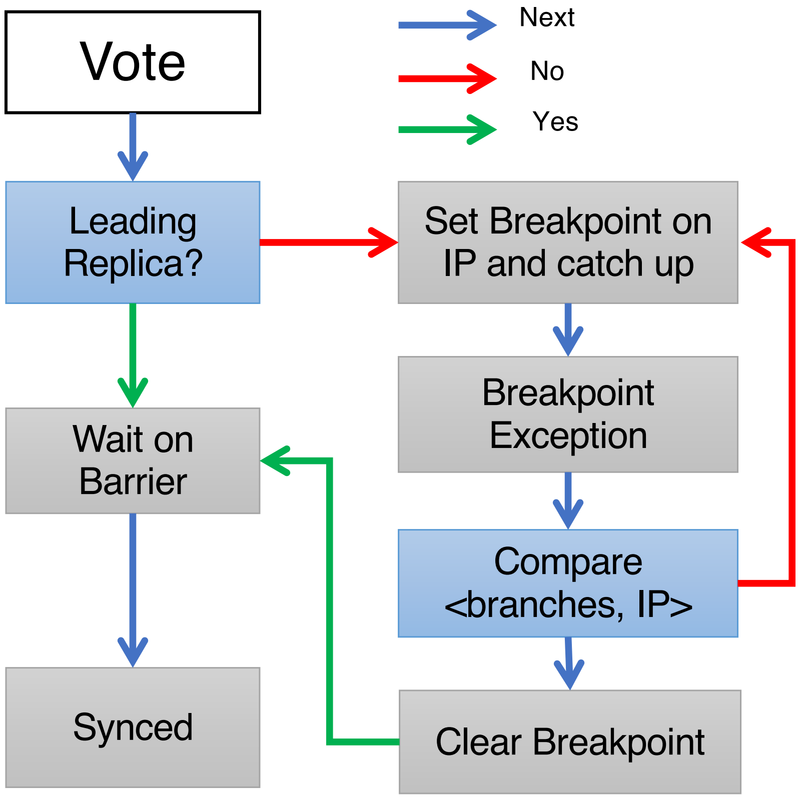
Closely-Coupled RCoE Implementation

Precise logical time: Triple of:

- event count
- user-mode branch count
- instruction pointer

x86: Obtained from PMU

Arm v7: Use gcc plugin to count branches



Performance: Microbenchmarks

	Dhrystone		Whetstone	
	Arm	x86	Arm	x86
Base	146.1	108.1	108.9	120.3
LC	147.0	108.6	109.8	120.4
CC	153.4	111.9	133.5	143.0

Loosely-coupled

Closely-coupled

LC has low overhead for CPU-bound

LC has usually low inherent overhead for CPU-bound

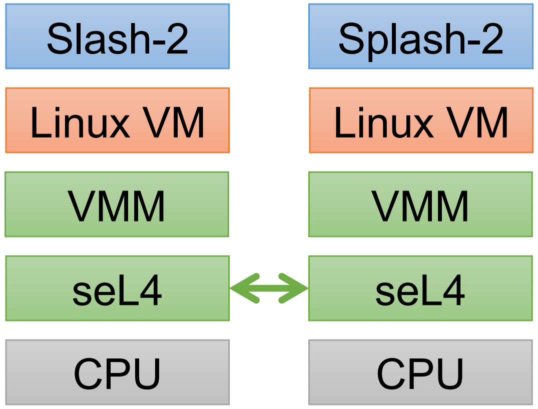
CC has high overhead for tight loops

Performance: SPLASH-2 on x86 VMs

Name	N	Base	CC-D	Factor
BARNES	30	61	93	1.52
CHOLESKY	300	66	792	12.08
FFT	100	64	142	2.22
FFM	20	76	160	2.11
LU-C	30	64	437	6.83
LU-NC	20	62	381	6.12
OCEAN-C	1000	64	173	2.71
OCEAN-NC	1000	65	171	2.65
RADIOSITY	25	66	75	1.12
RADIX	20	66	89	1.34
RAYTRACE	1000	60	65	1.09
VOLREND	100	86	133	1.54
WATER-NS	600	66	92	1.41
WATER-S	600	67	84	1.25

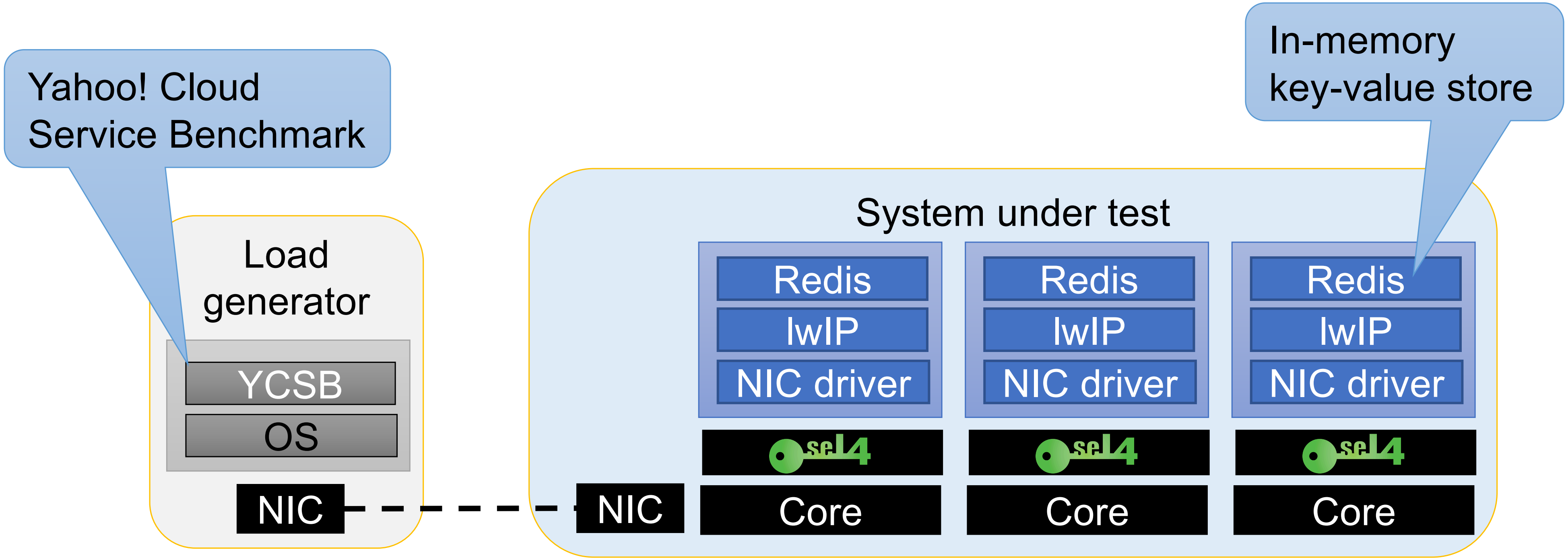
- Execution time in sec
- DMR configuration
- Base: unreplicated single-coreVM

Breakpoints in VM are expensive: trigger VM exits

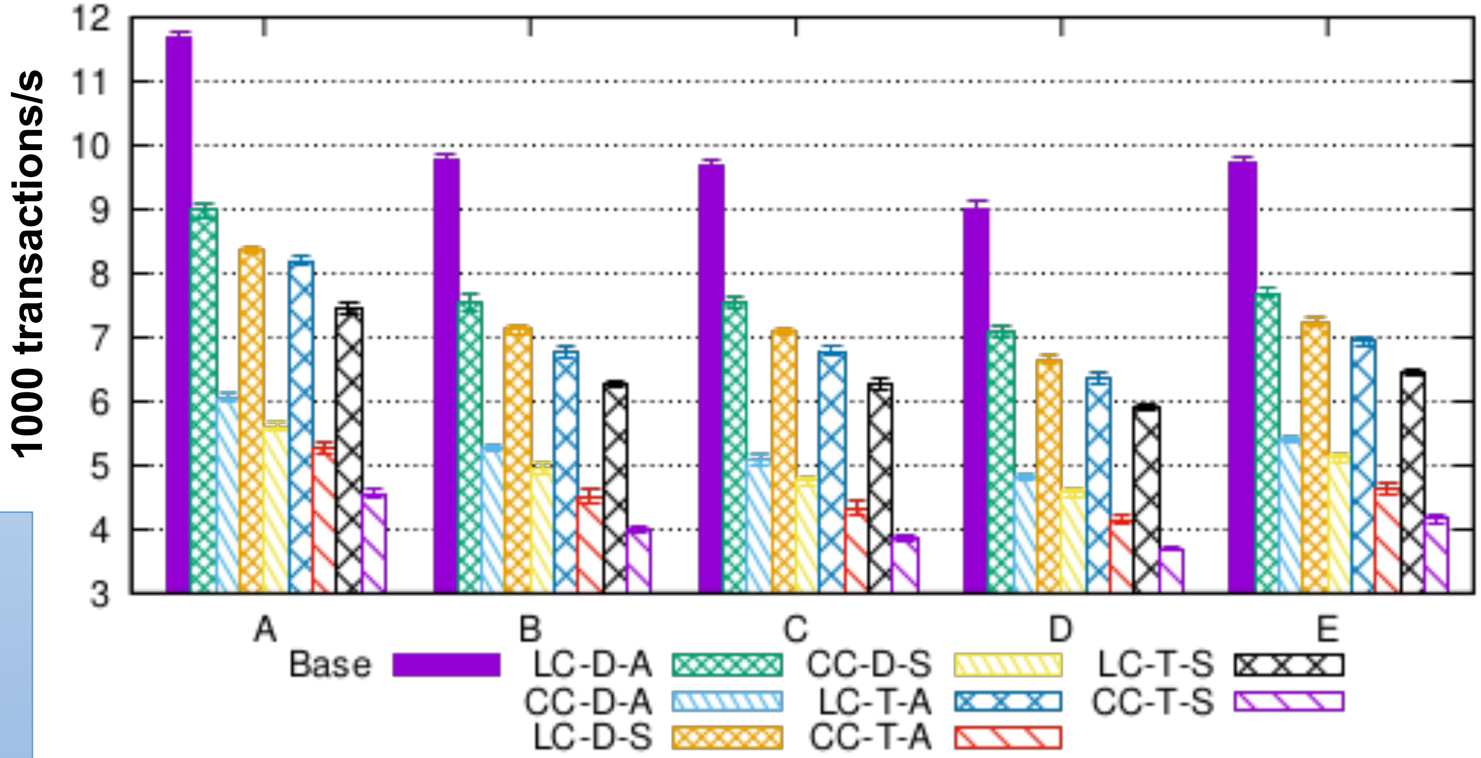


Geometric mean overhead: 2.3×

Benchmark: Redis – YCSB



Performance: Redis on Arm



LC: loosely-coupled

CC: closely-coupled

D: DMR

T: TMR

A: vote on interrupt

S: also vote on syscall

Overhead is 1.2–3 depending on configuration

Error Detection on Arm

Not checksumming network data

Checksumming NW data

	Base	LC-D	LC-T	LC-D-N	LC-T-N	CC-D	CC-T
Injected faults	243k	202k	184k	224k	214k	205k	185k
YCSB corruptions	647	3	1	**381**	**299**	3	0
YCSB errors	57	1	0	13	10	3	6
User errors	296	0	0	0	0	0	0
Kernel exceptions	0	0	0	0	0	0	0
Undetected	1000	4	1	394	309	6	6
RCoE detected	N/A	996	999	606	691	994	994
Observed errors	1000	1000	1000	1000	1000	1000	1000

Comparison to Rad-Hardened Processor

	Sabre Lite	RAD750
Cores @ clock	4 @ 800 MHz	1 @ 133 MHz
Performance	4 × 2,000 DMIPS	240 DMIPS
Power	< 5 W	< 6 W
Energy Efficiency	200 DMIPS/W	40 DMIPS/W
Cost	$200	$200,000
Perf/Cost	5 DMIPS/$	0.0002 DMIPS/$

2002 price

Assuming 2× overhead, TMR

[Shen et al., DSN'19]

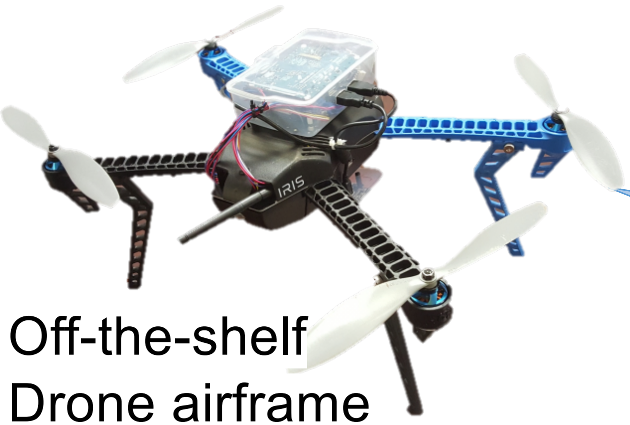
Real-World Use

DARPA HACMS

Unmanned Little Bird (ULB)

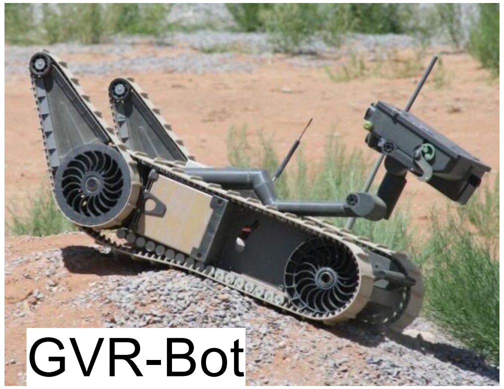
Retrofit existing system!

Autonomous trucks



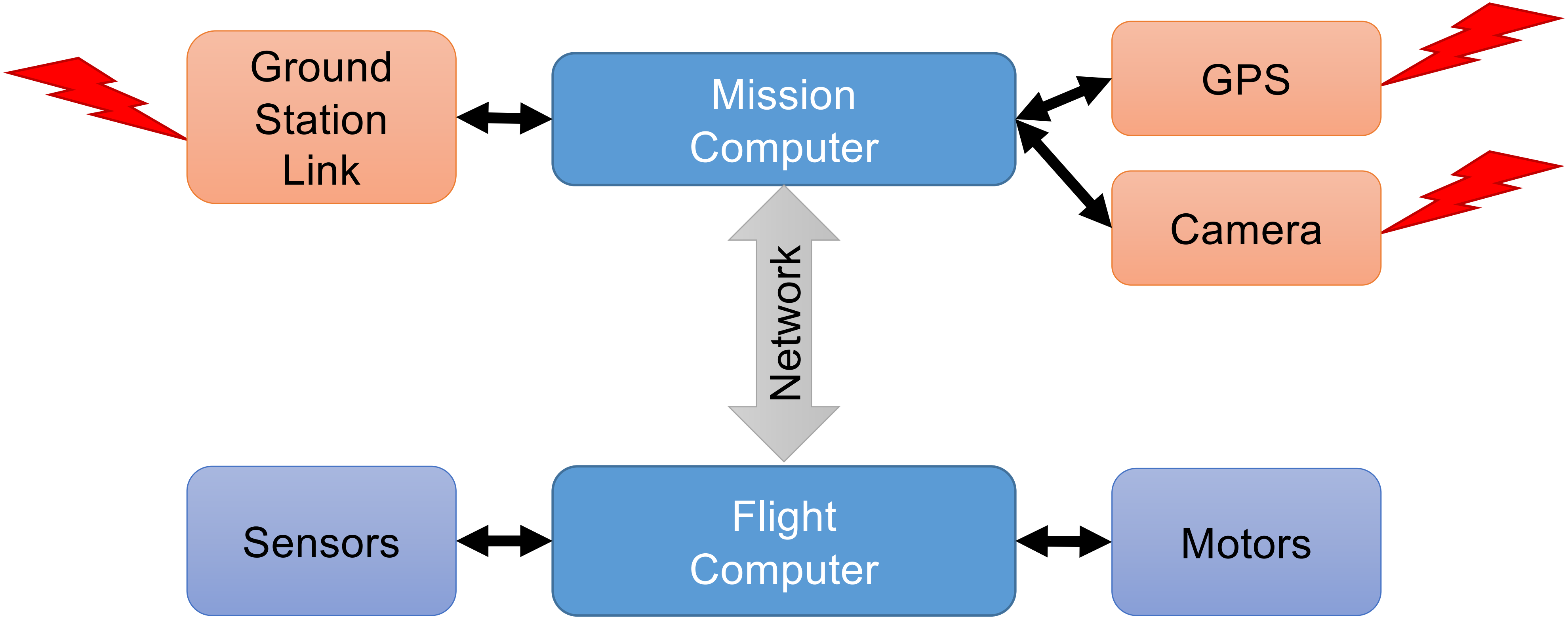
Off-the-shelf Drone airframe

Develop technology

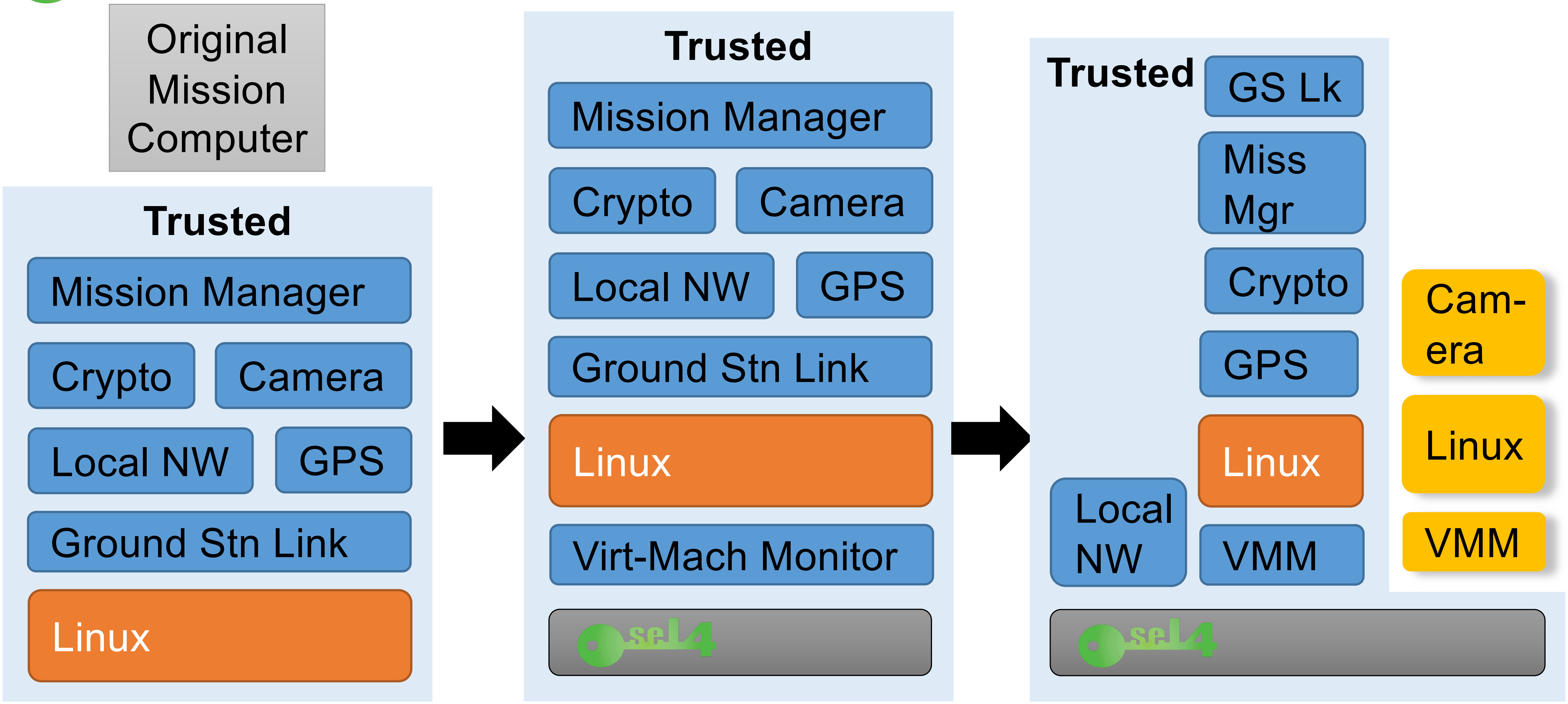


GVR-Bot

ULB Architecture



Incremental Cyber Retrofit



Original Mission Computer

Trusted

Mission Manager

Crypto

Camera

Local NW

GPS

Ground Stn Link

Linux

Trusted

Mission Manager

Crypto

Camera

Local NW

GPS

Ground Stn Link

Linux

Virt-Mach Monitor

Trusted

GS Lk

Miss Mgr

Crypto

GPS

Linux

Local NW

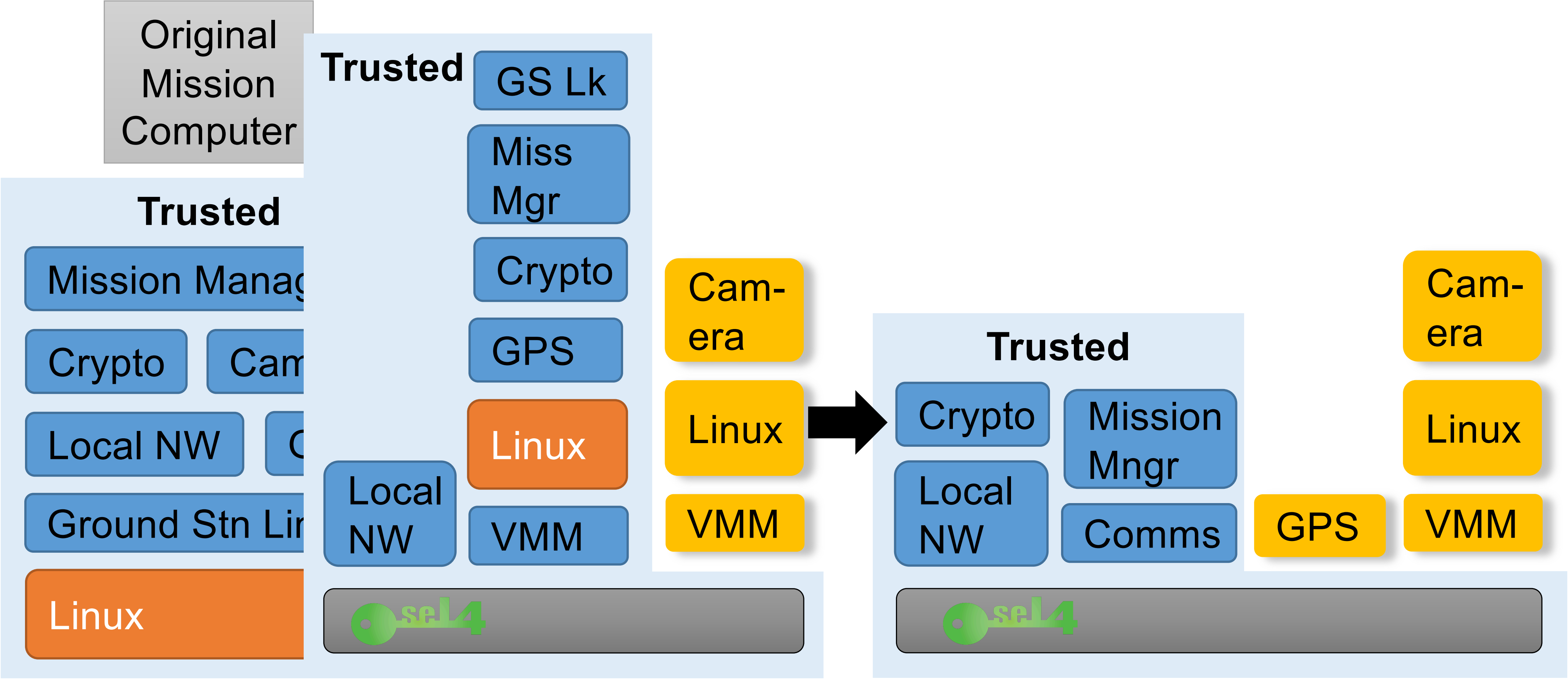
VMM

Cam-era

Linux

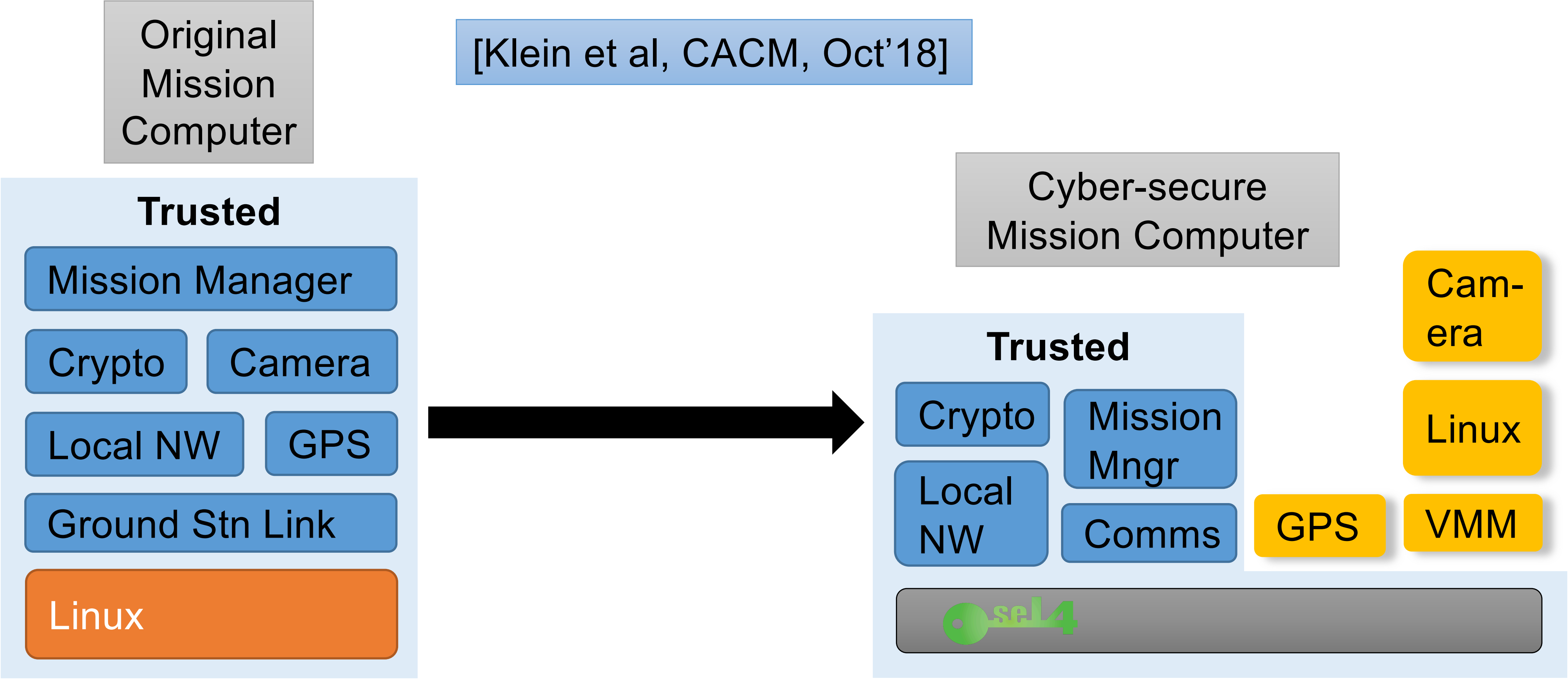
VMM

Incremental Cyber Retrofit

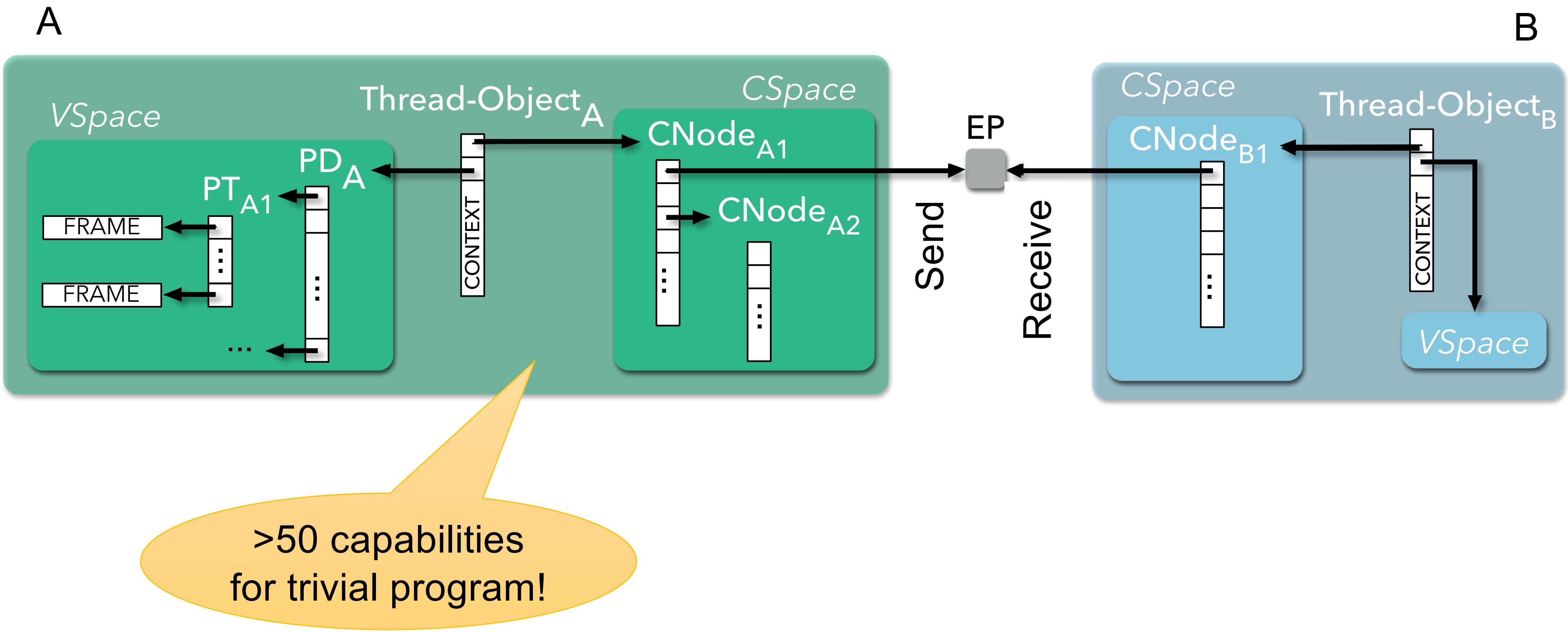


Incremental Cyber Retrofit

[Klein et al, CACM, Oct'18]

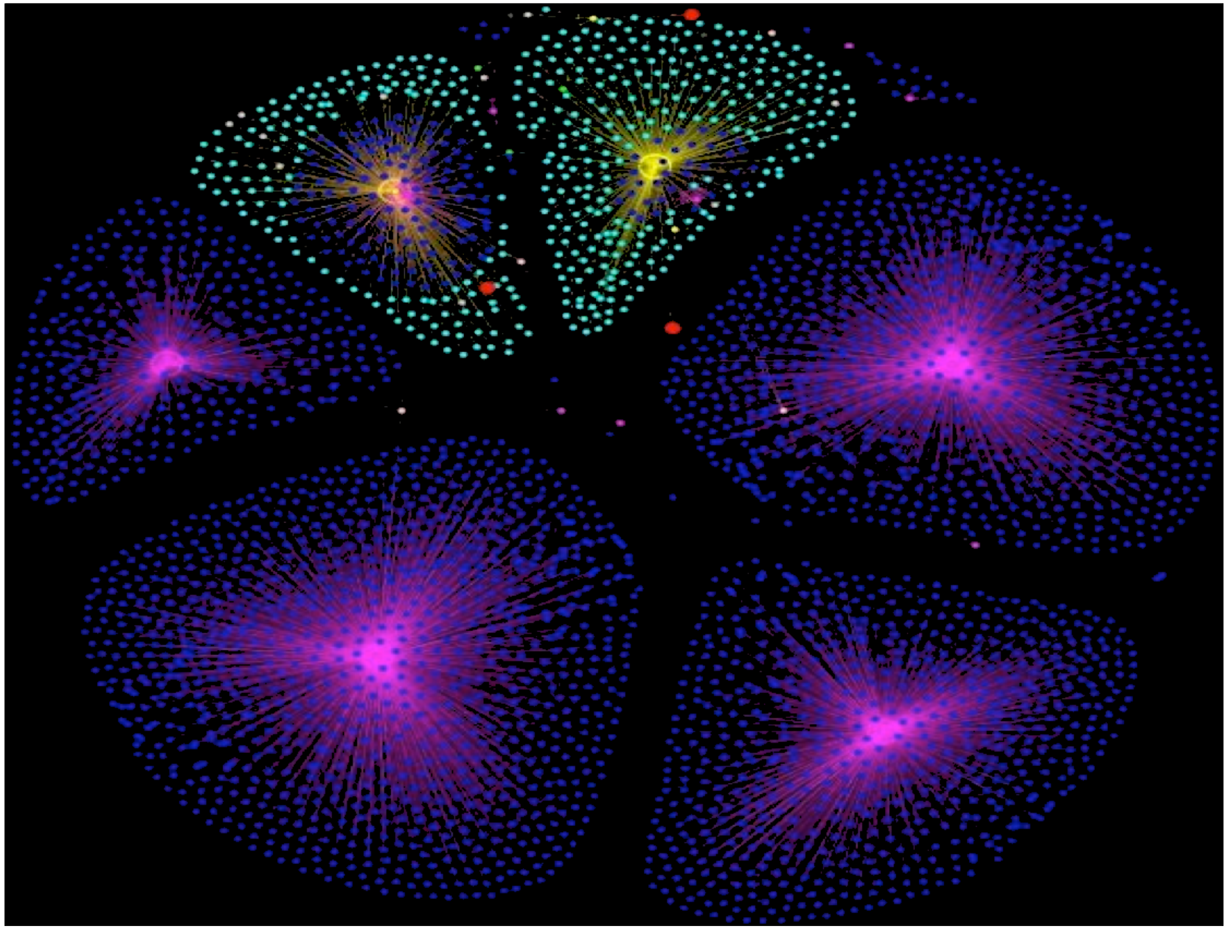


Issue: Capabilities are Low-Level



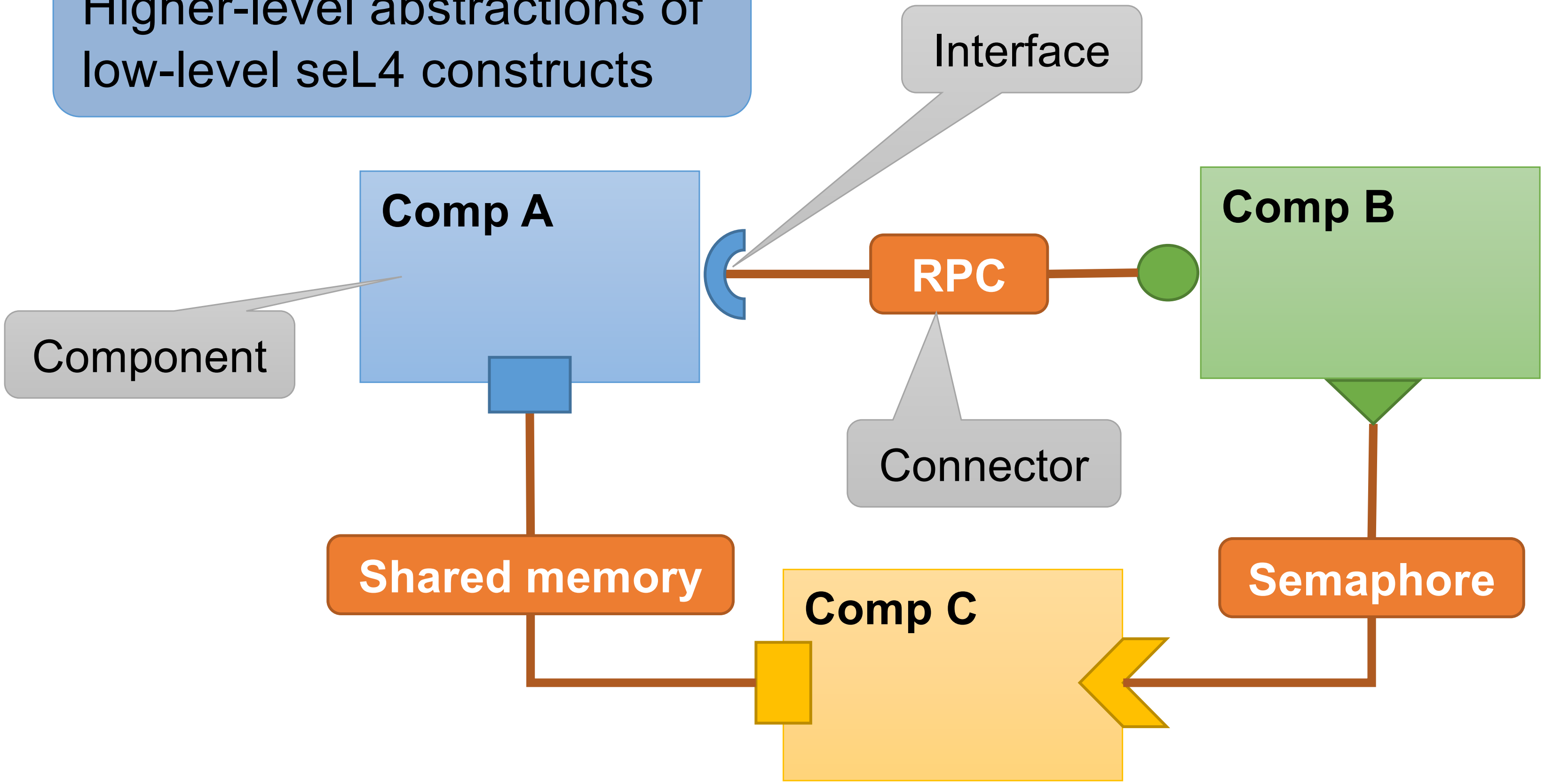
>50 capabilities
for trivial program!

Simple But Non-Trivial System

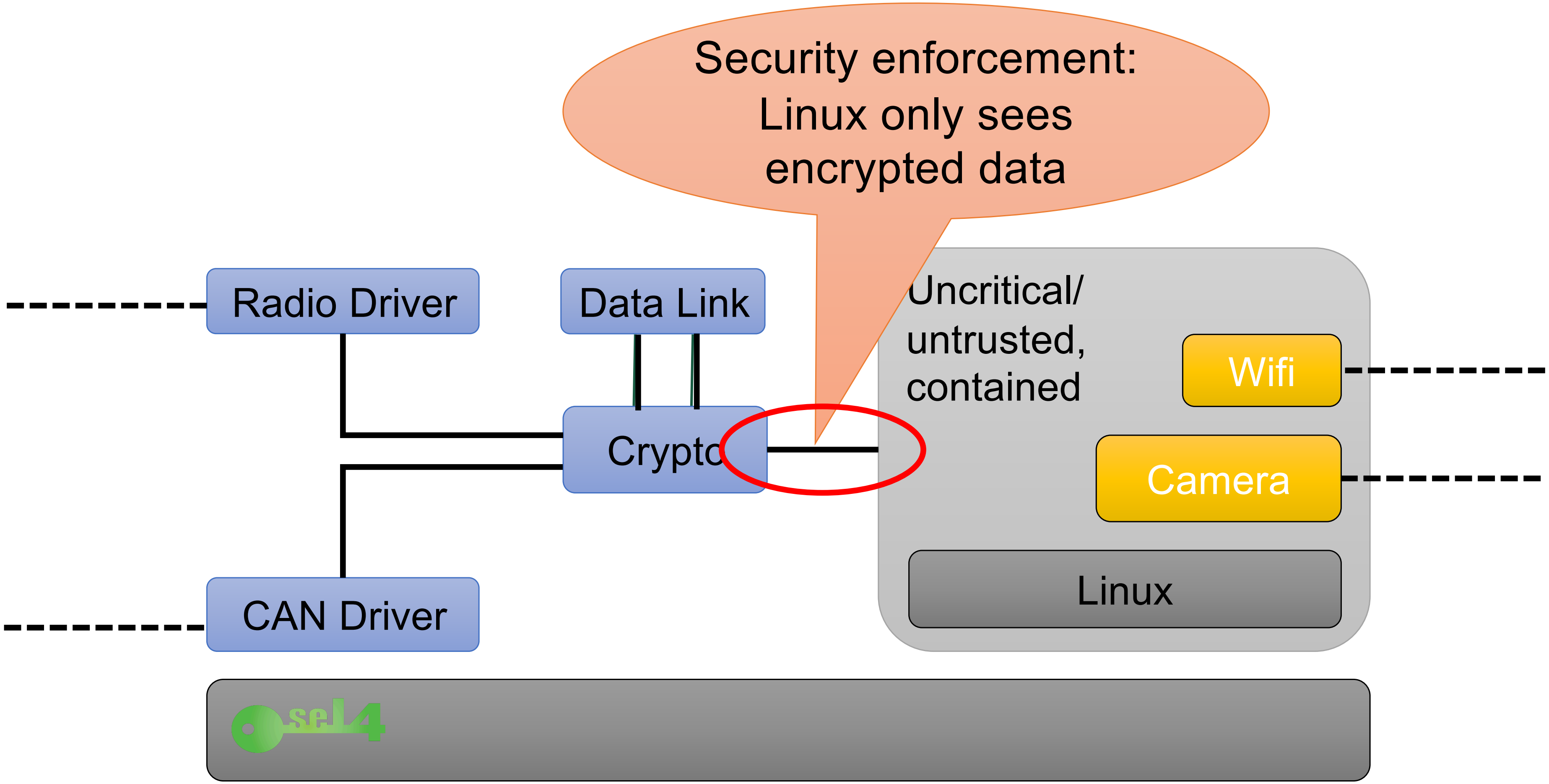


Component Middleware: CAmkES

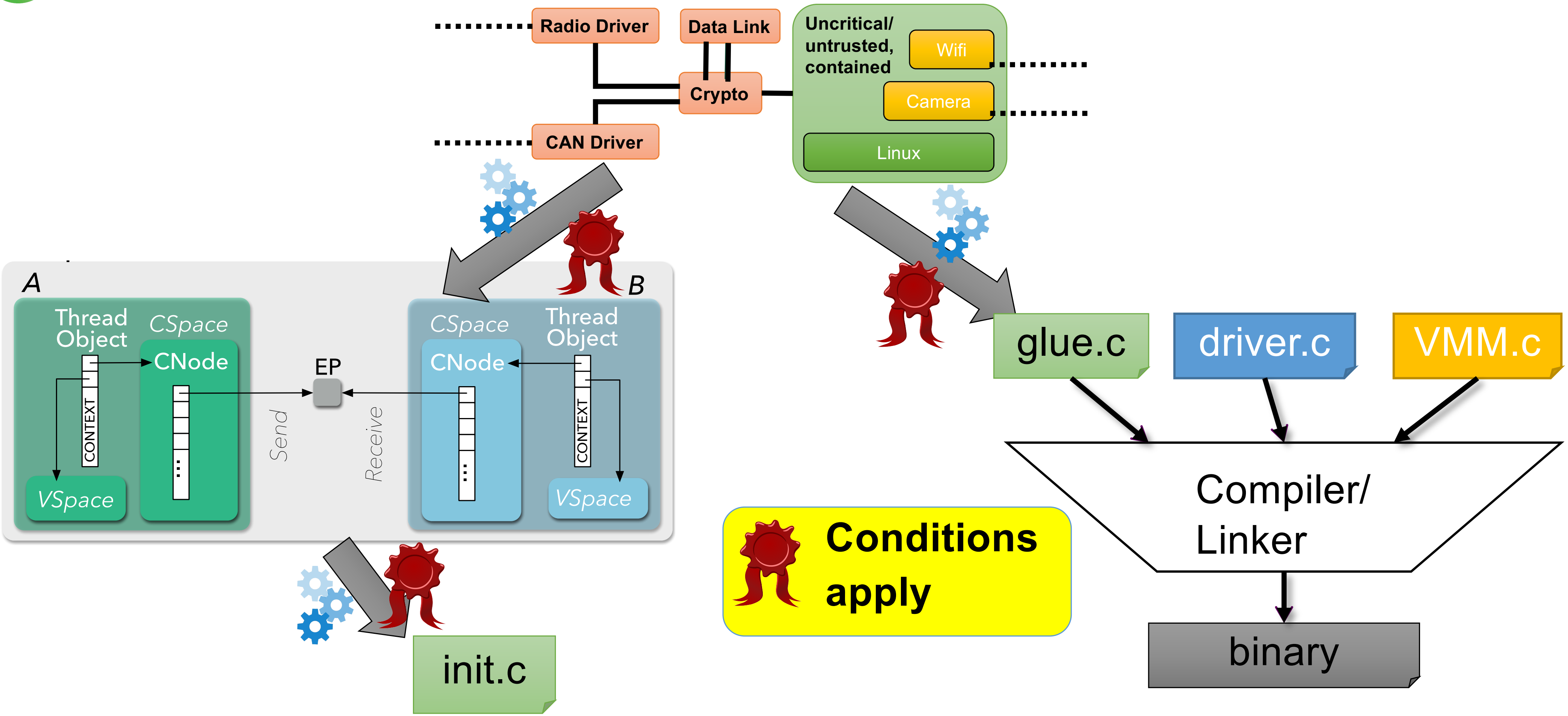
Higher-level abstractions of low-level seL4 constructs



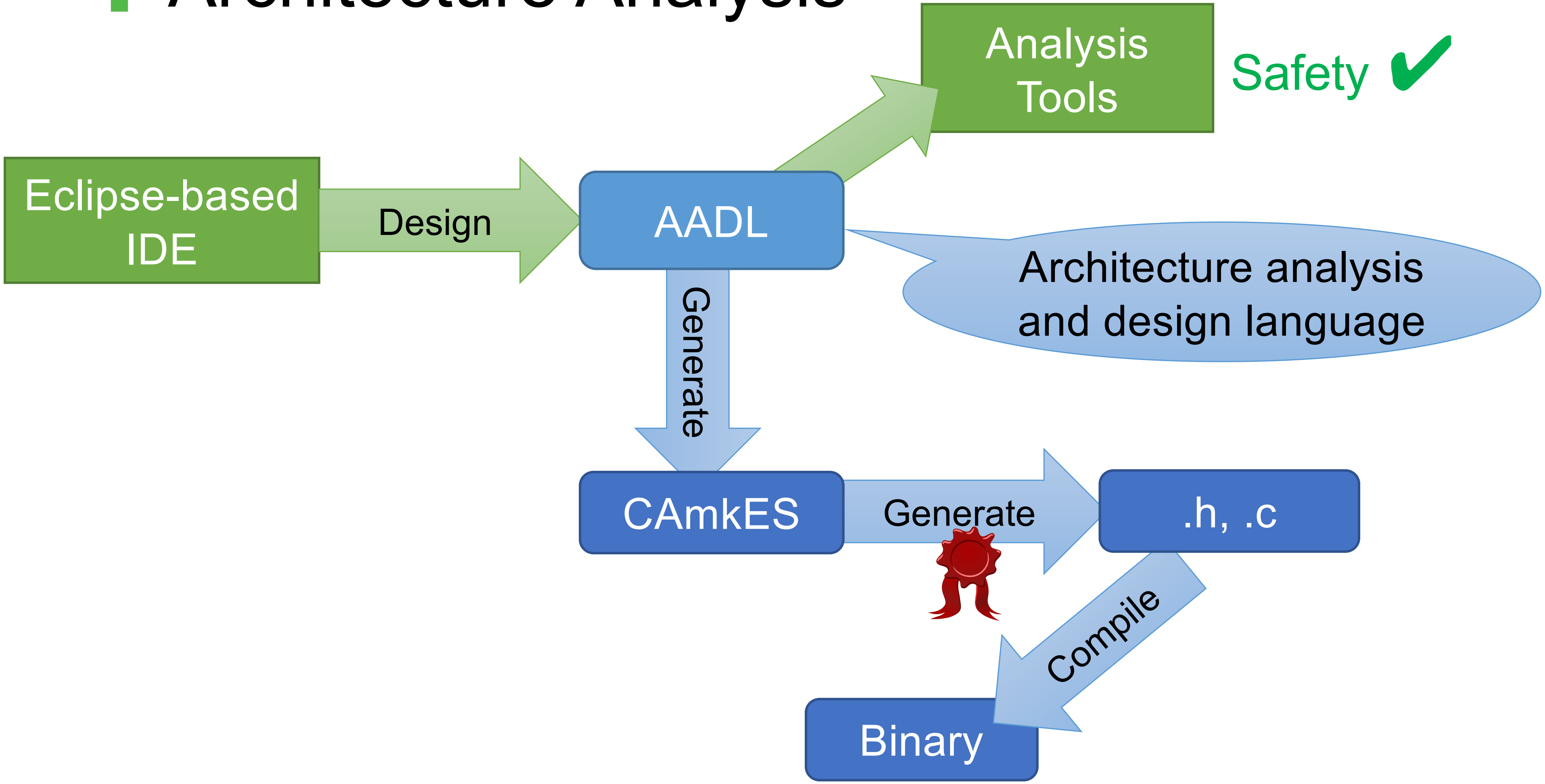
HACMS UAV Architecture



Enforcing the Architecture



Architecture Analysis



Real-World Use

Courtesy Boeing, DARPA