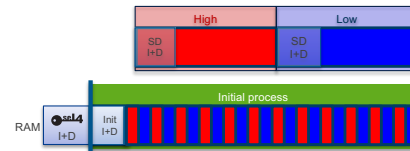


2020 T2 Week 10b  
Local OS Research  
@GernotHeiser



## Copyright Notice

These slides are distributed under the **Creative Commons Attribution 3.0 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/3.0/legalcode>

# 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

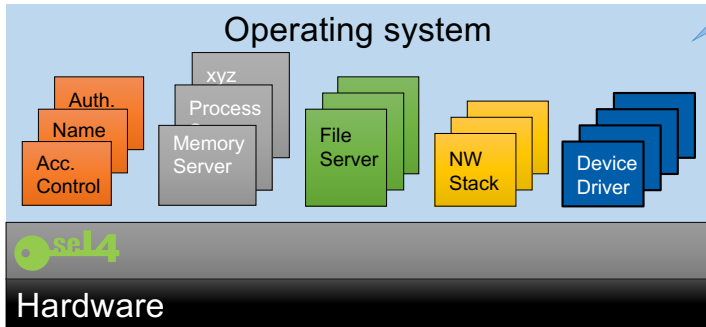
115 critical  
Linux CVEs  
to Nov'17

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

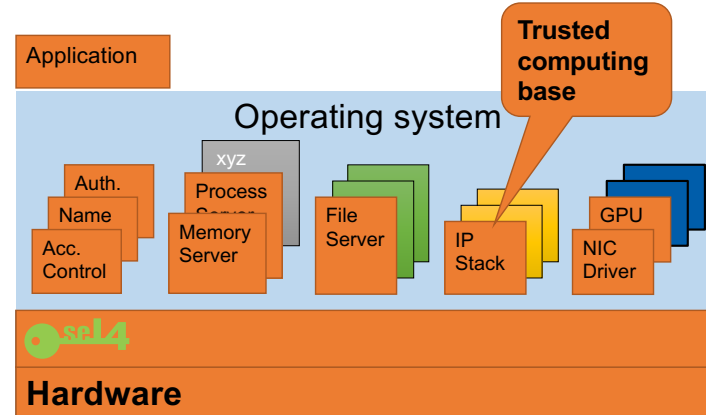
## seL4 Hypothetical seL4-based OS

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

Functionality comparable to Linux



## seL4 Hypothetical Security-Critical App

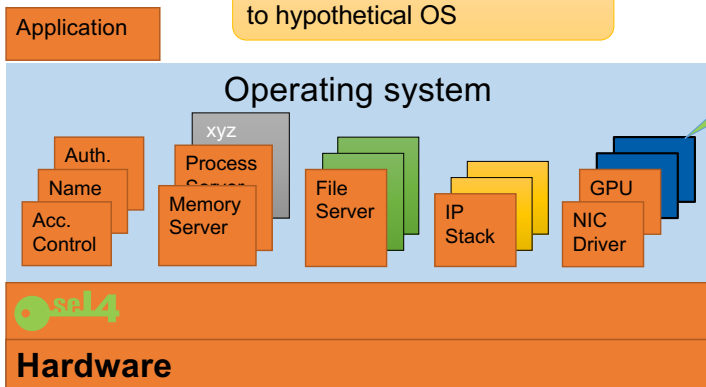


App requires:

- IP networking
- File storage
- Display output

## seL4 Analysing CVEs

Map compromised component to hypothetical OS

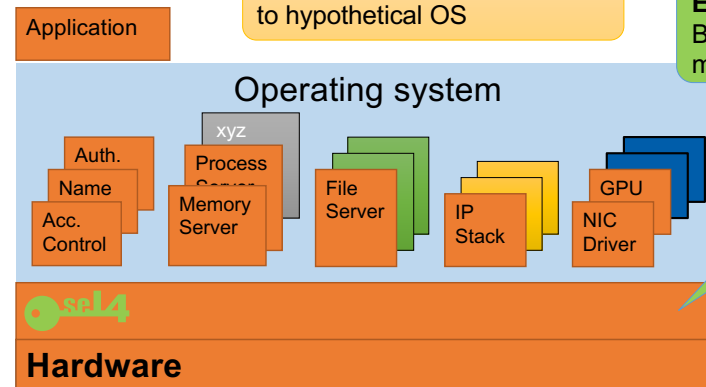


Not in TCB: Attack defeated

Example: USB driver bug

## seL4 Analysing CVEs

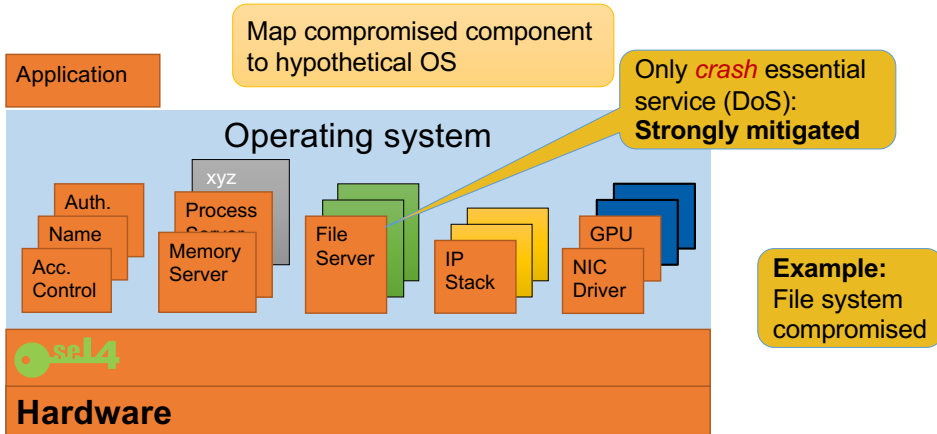
Map compromised component to hypothetical OS



Example: Bug in page-table management

In microkernel: Attack defeated by verification

## sel4 Analysing CVEs



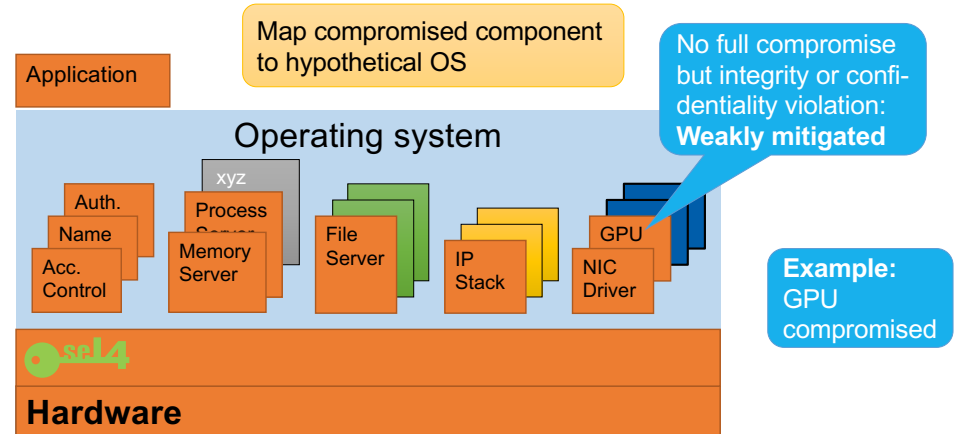
8

COMP9242 2020T2 W10b: Local OS Research

© Gernot Heiser 2019 – CC Attribution License



## sel4 Analysing CVEs



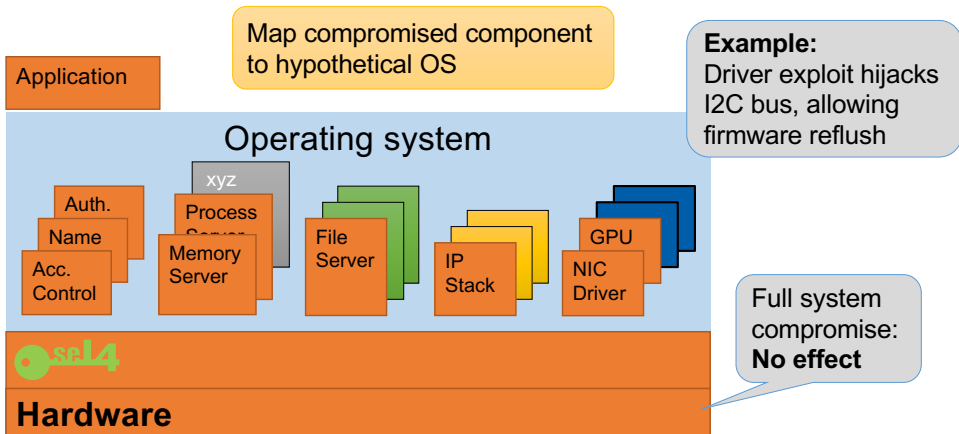
9

COMP9242 2020T2 W10b: Local OS Research

© Gernot Heiser 2019 – CC Attribution License



## sel4 Analysing CVEs



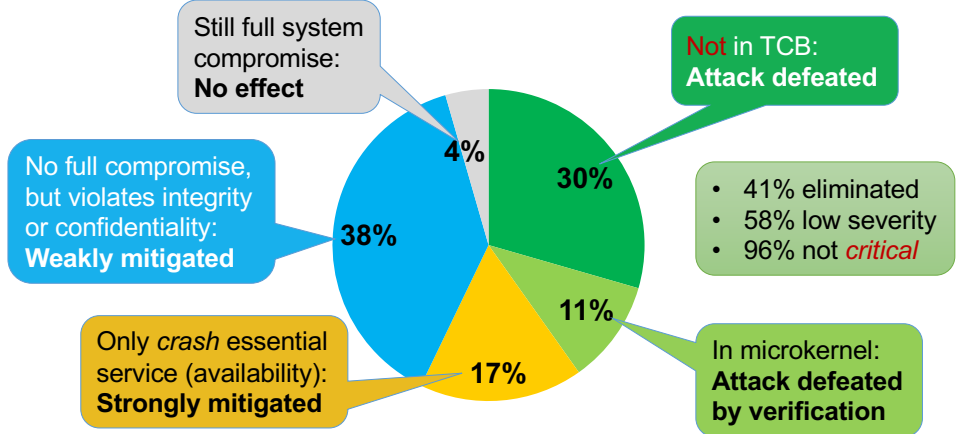
10

COMP9242 2020T2 W10b: Local OS Research

© Gernot Heiser 2019 – CC Attribution License



## sel4 All Critical Linux CVEs to 2017



11

COMP9242 2020T2 W10b: Local OS Research

© Gernot Heiser 2019 – CC Attribution License



# Summary

## OS structure matters!

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

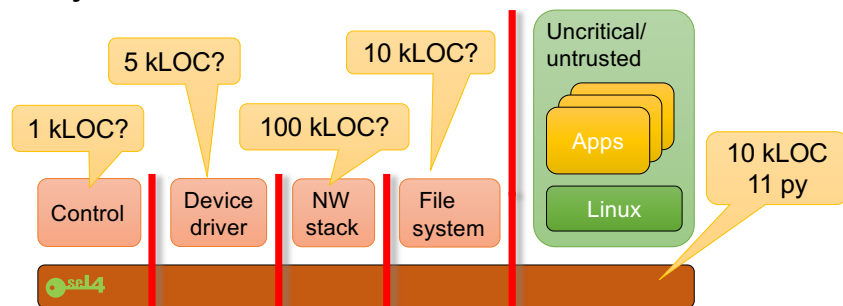
[Biggs et al., APSys'18]

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



# Cogent

# Beyond the Kernel



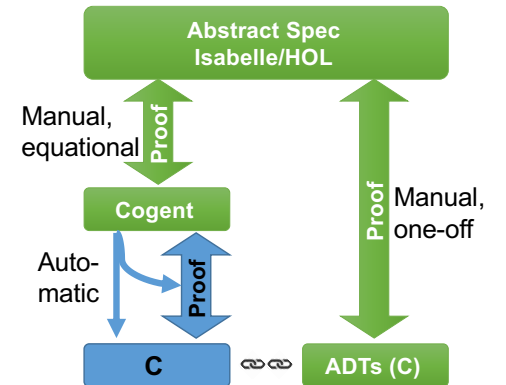
Aim: Verified TCB at affordable cost!

# 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]

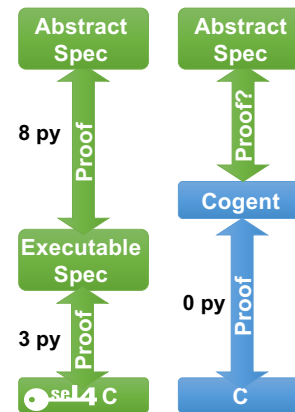


## Manual Proof Effort

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

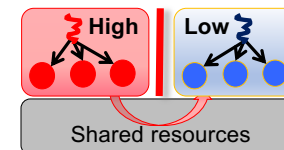
Spec reuse!

### Work in progress:

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

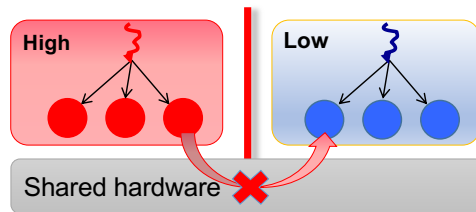
## Time Protection

## Refresh: Microarchitectural Timing Channels



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

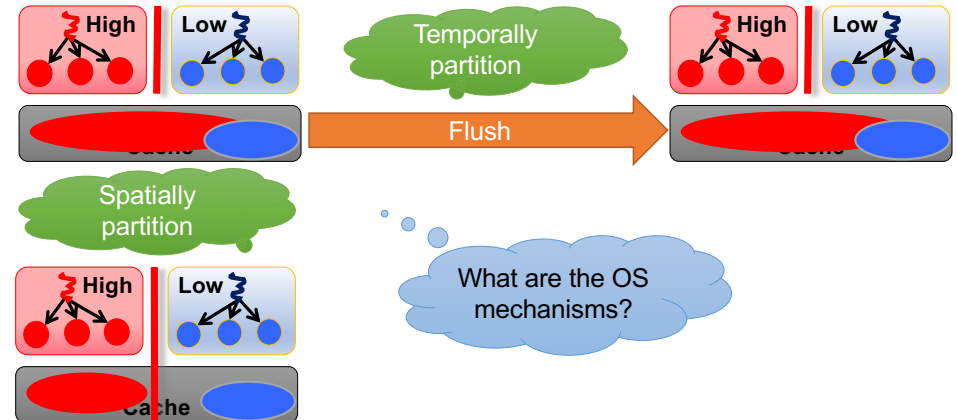
## OS Must Enforce *Time Protection*



**Preventing interference is core duty of the OS!**

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

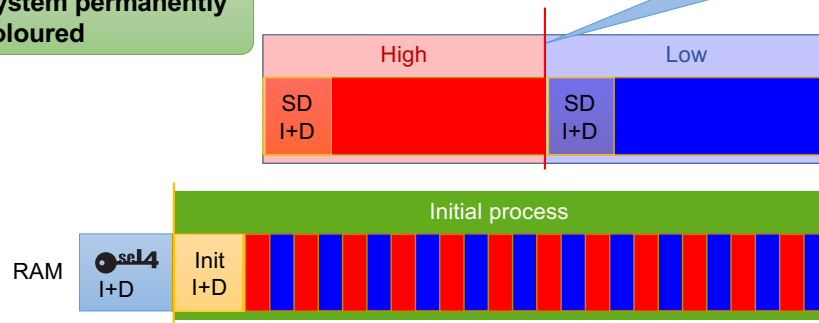
## Time Protection: No Sharing of HW State



## seL4 Spatial Partitioning: Cache Colouring

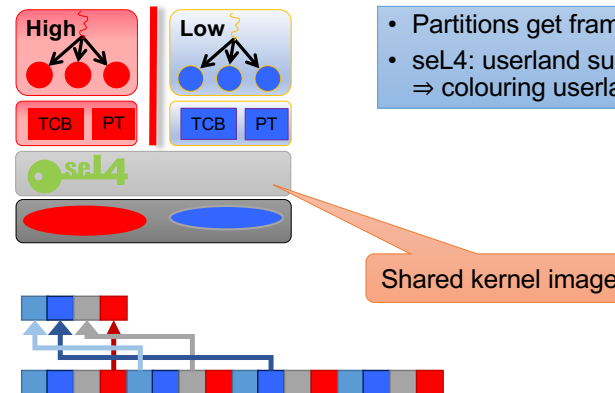
System permanently coloured

Partitions restricted to coloured memory

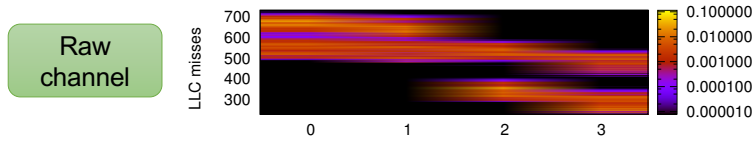


## seL4 Spatial Partitioning: Cache Colouring

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



## seL4 Channel Through Kernel Code



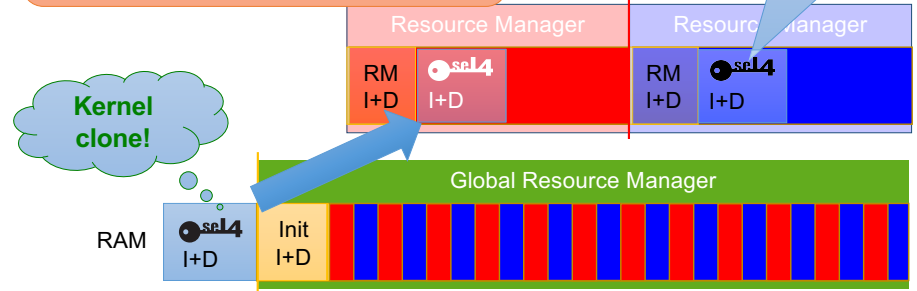
Channel matrix: Conditional probability of observing output signal (time) given input signal (system-call number)

## seL4 Colouring the Kernel

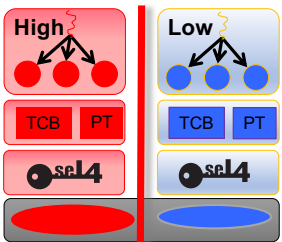
Remaining shared kernel data:

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

Each partition has own kernel image



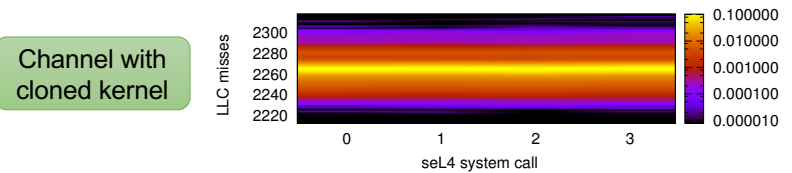
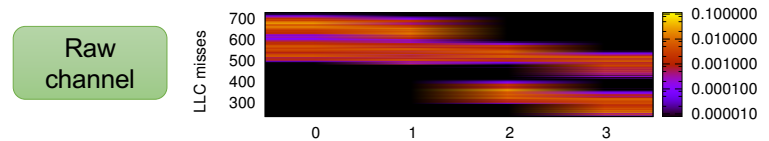
## seL4 Spatial Partitioning: Cache Colouring



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

Must ensure deterministic access to remaining shared kernel state!

## seL4 Channel Through Kernel Code



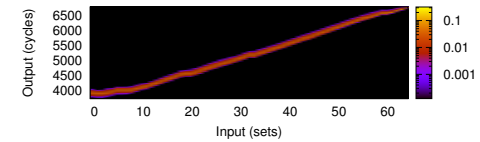
## sel4 Temporal Partitioning: Flush on Switch

Must remove any history dependence!

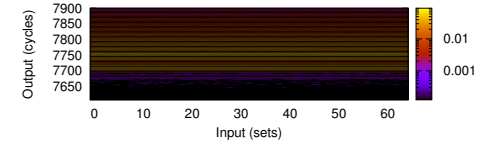
2. Switch user context
3. Flush on-core state
6. Reprogram timer
7. return

## sel4 D-Cache Channel

Raw channel

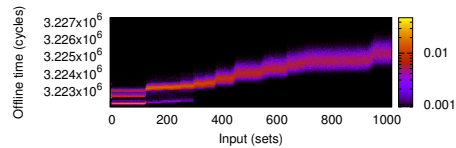


Channel with flushing



## sel4 Flush-Time Channel

Raw channel



## sel4 Temporal Partitioning: Flush on Switch

Must remove any history dependence!

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

Latency depends on prior execution!

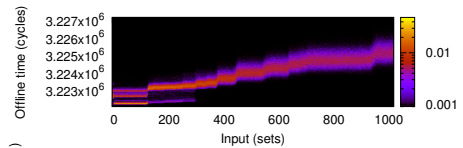
Time padding to remove dependency

Ensure deterministic execution

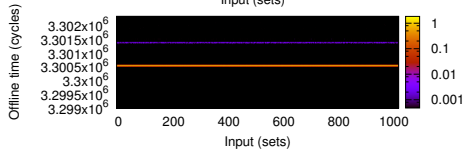


## seL4 Flush-Time Channel

Raw channel

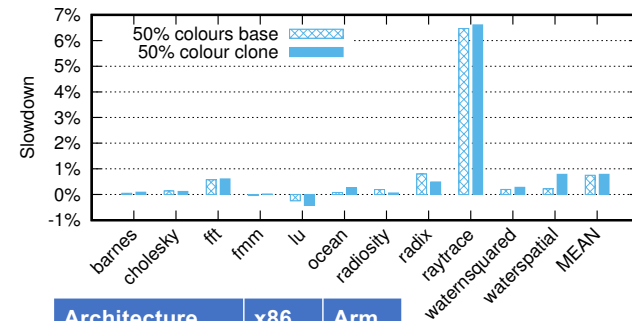


Channel with deterministic flushing



## seL4 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 $\mu$ s	257 $\mu$ s
Arm	608 $\mu$ s	4,300 $\mu$ s

## A New HW/SW Contract

For all shared microarchitectural resources:

aISA: augmented ISA

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 Cannot share HW threads across security domains!
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

[Ge et al., APSys'18]

## seL4 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  $\geq T_0 + WCET$

[Heiser et al., HotOS'19]

Functional property

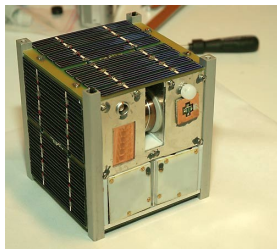
# Making COTS Hardware Dependable

# Satellites: SWaP vs Dependability

- Space is becoming commoditized:
- many, small (micro-) satellites
  - increasing cost pressure

Harsh environment for electronics:
 

- temperature fluctuations
- ionising radiation

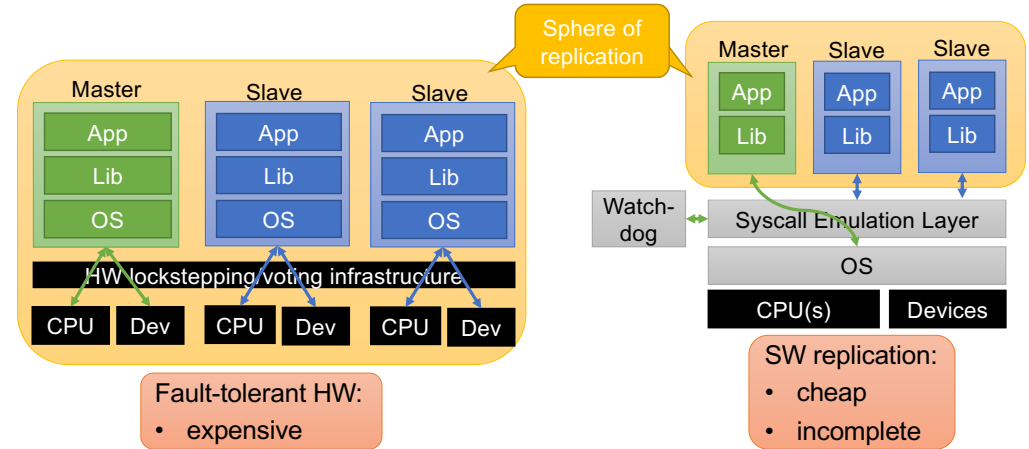


NCUBE2 by Bjørn Pedersen, NTNU (CC BY 1.0)

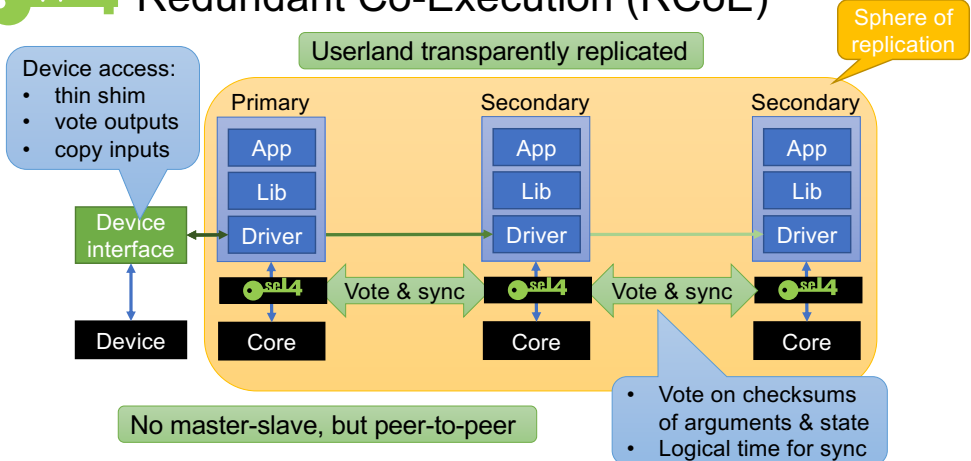
Radiation-hardened processors are slow, bulky and expensive

Use redundancy of cheap COTS multicores

# Traditional Redundancy Approaches



# sel4 Redundant Co-Execution (RCoE)



# RCoE: Two Variants

## Loosely-coupled RCoE

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

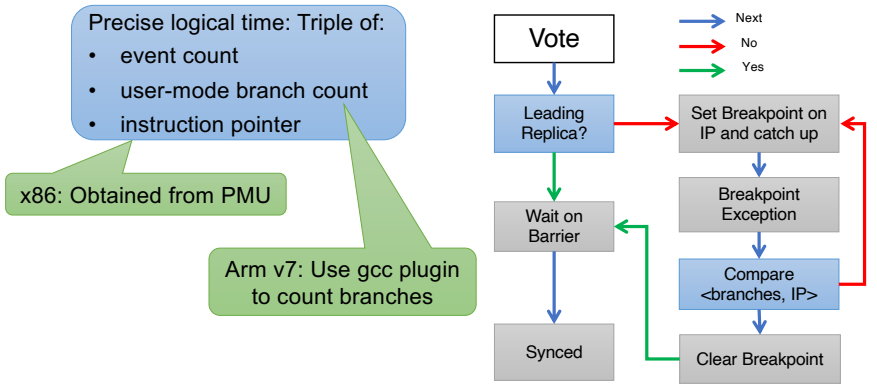
- Low overhead
- Cannot support racy apps, threads, virtual machines

## Closely-coupled RCoE

- Sync on instruction
- Precise preemptions

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

# sel4 Closely-Coupled RCoE Implementation



# sel4 Performance: Microbenchmarks

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

LC has low overhead for CPU-bound

CC has usually low inherent overhead for CPU-bound

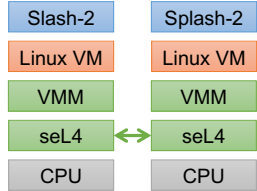
CC has high overhead for tight loops

## seL4 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
RADIOSTY	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-core VM

Breakpoints in VM are expensive: trigger VM exits

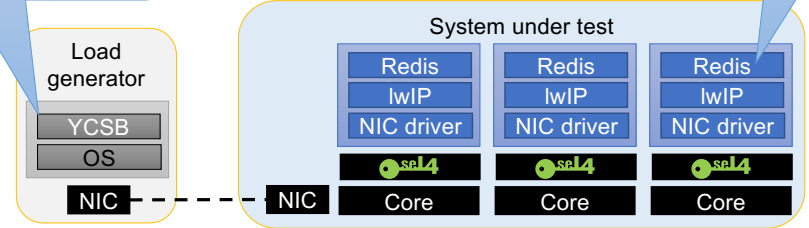


Geometric mean overhead: 2.3x

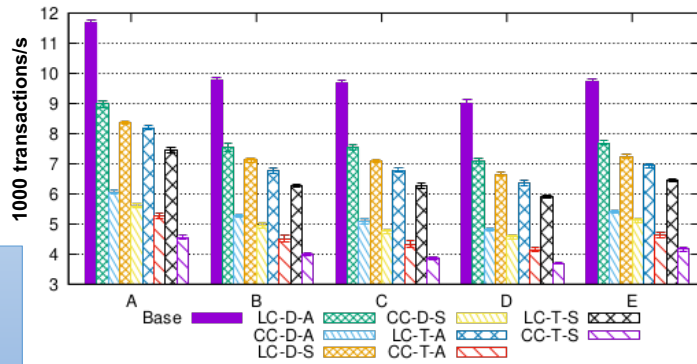
## seL4 Benchmark: Redis – YCSB

Yahoo! Cloud Service Benchmark

In-memory key-value store



## seL4 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

## seL4 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

## seL4 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/\$

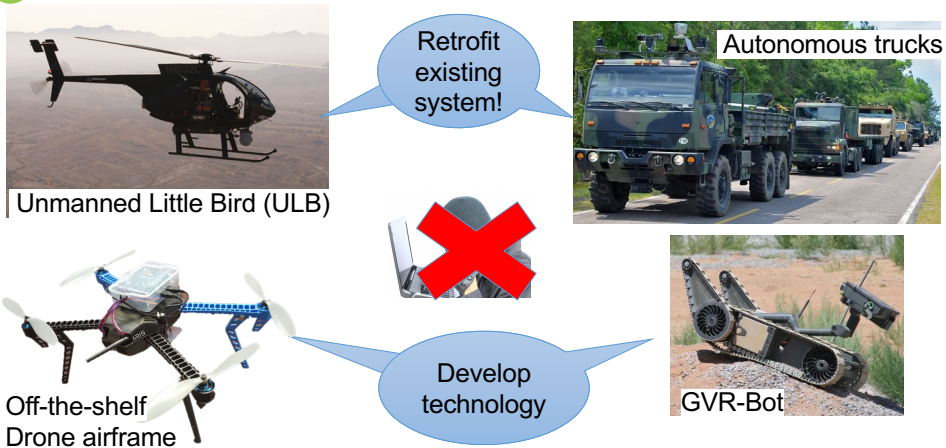
2002 price

Assuming 2× overhead, TMR

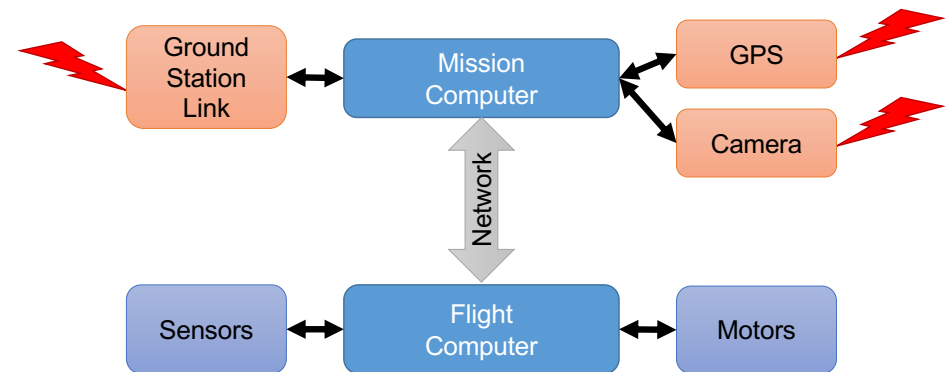
[Shen et al., DSN'19]

## Real-World Use

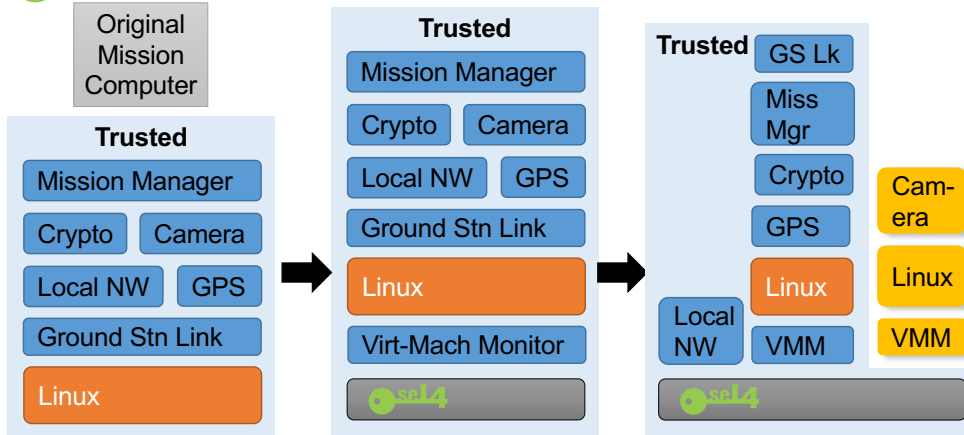
## seL4 DARPA HACMS



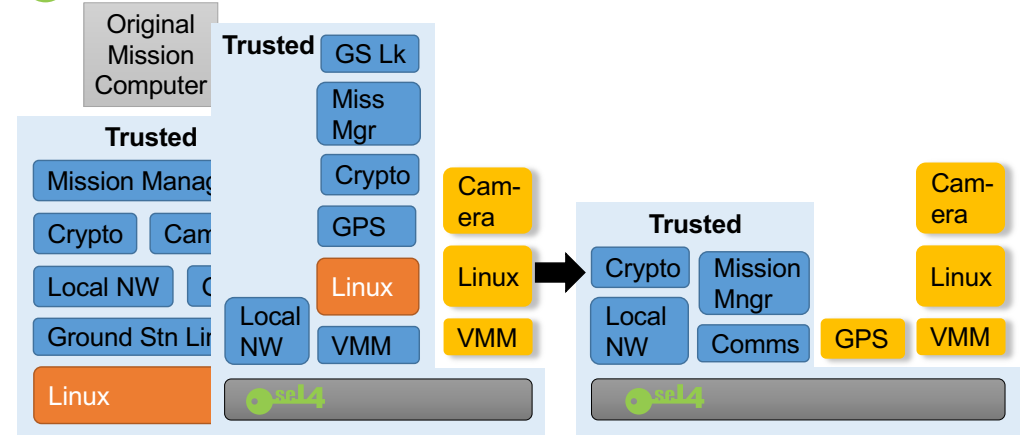
## seL4 ULB Architecture



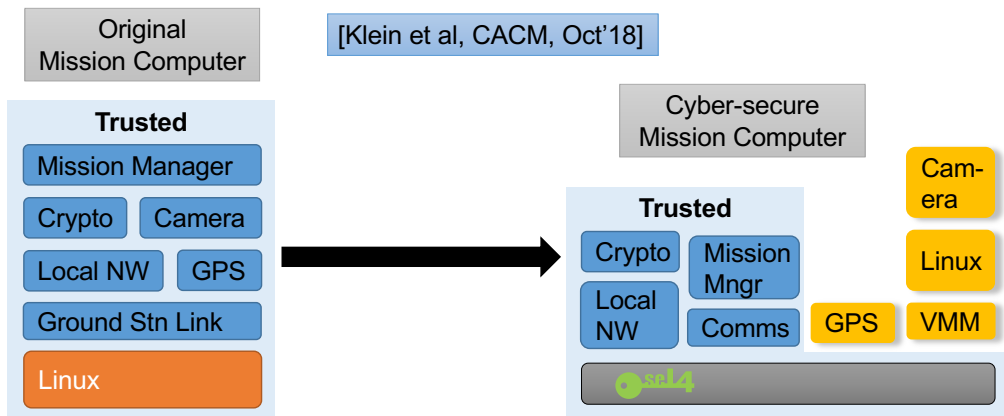
## sel4 Incremental Cyber Retrofit



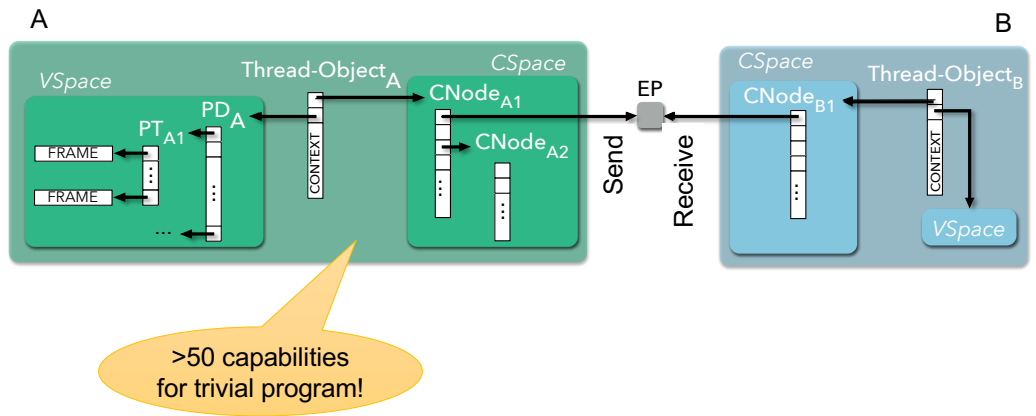
## sel4 Incremental Cyber Retrofit



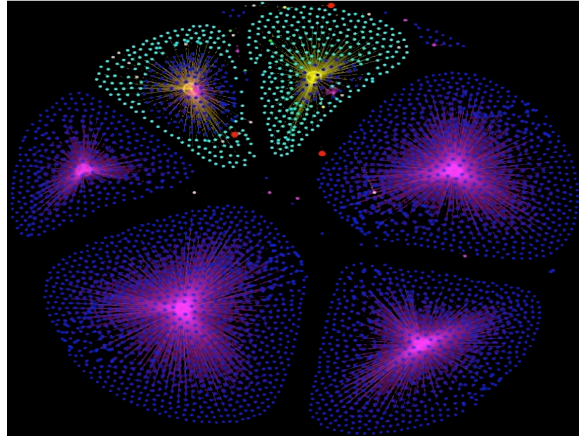
## sel4 Incremental Cyber Retrofit



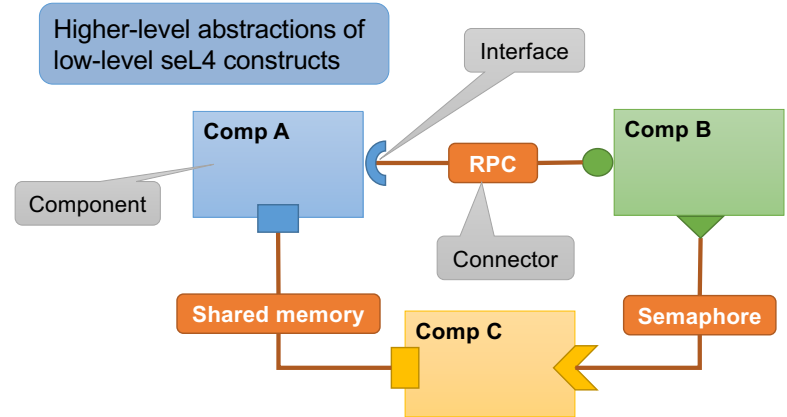
## sel4 Issue: Capabilities are Low-Level



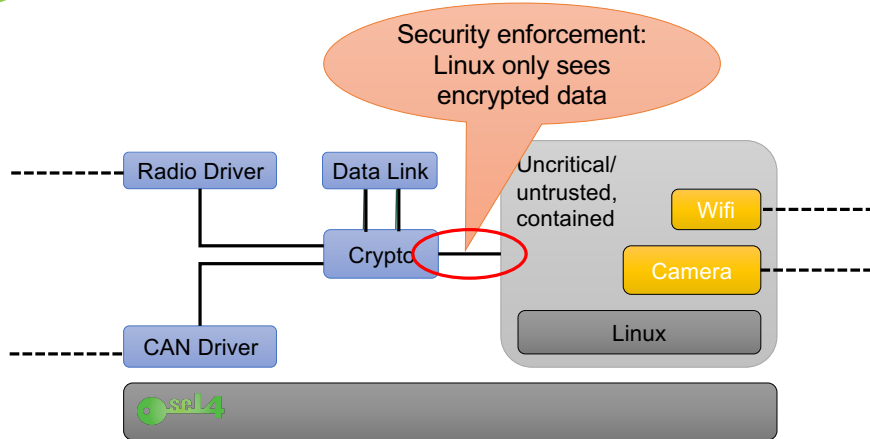
# seL4 Simple But Non-Trivial System



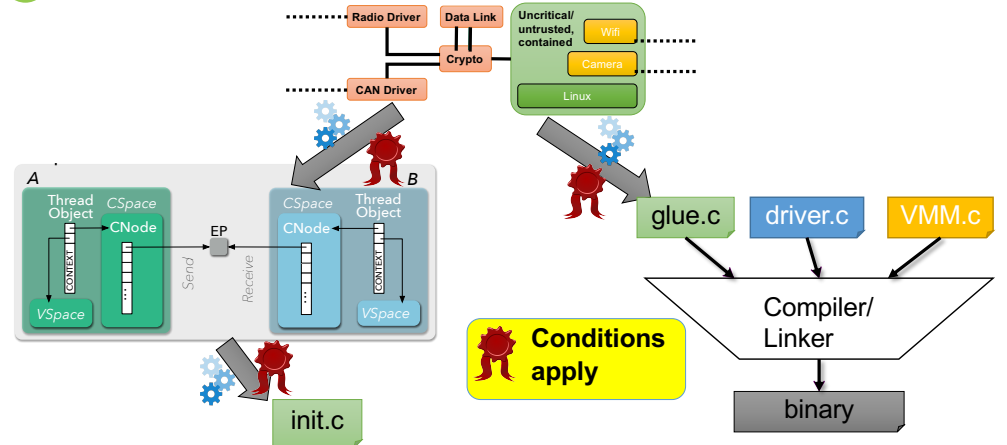
# seL4 Component Middleware: CAmkES



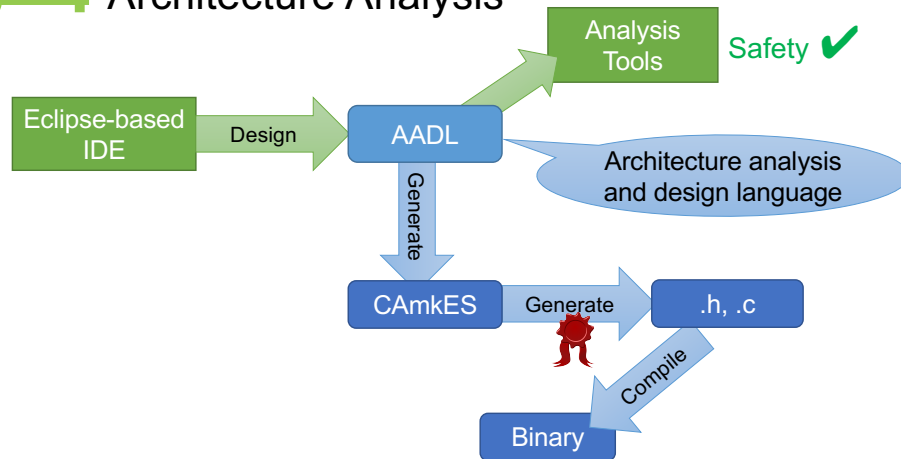
# seL4 HACMS UAV Architecture



# seL4 Enforcing the Architecture



## seL4 Architecture Analysis



## seL4 Military-Grade Security

### Cross-Domain Desktop Composer



#### Multi-level secure terminal

- Successful defence trial in AU
- Evaluated in US, UK, CA
- Formal security evaluation soon

Pen10.com.au crypto communication device in use in AU, UK defence



## seL4 Real-World Use Courtesy Boeing, DARPA



# Thanks!

To the AOS students for their interest and commitment

To the awesome, world-class Trustworthy Systems Team for making this all possible

To Trudy Weibel for her help with reviewing and editing this material

