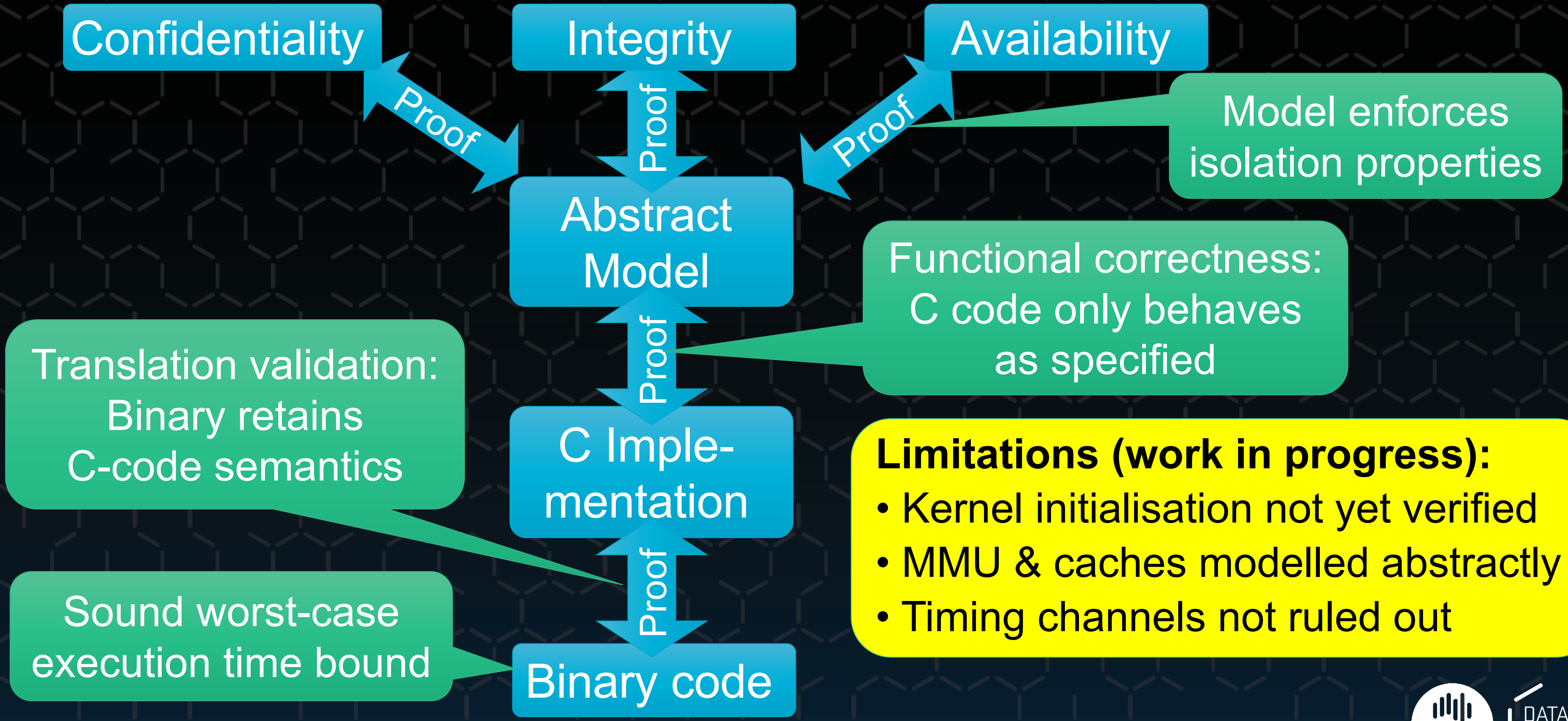
Making the (Software) TCB Trustworthy

Gernot Heiser | gernot@unsw.edu.au | @GernotHeiser

- CPS-VO FMaS, Menlo Park, 9 October 2019

https://trustworthy.systems

seL4: Base for Trustworthy Systems



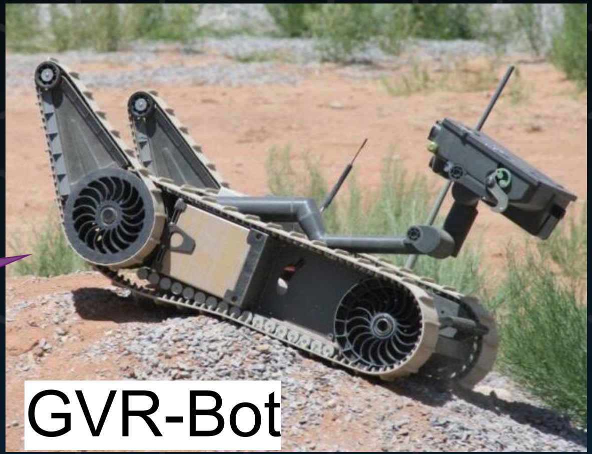
Confidentiality
Integrity
Availability
Proof
Proof
Proof
Model enforces isolation properties
Abstract Model
Proof
Functional correctness: C code only behaves as specified
Translation validation: Binary retains C-code semantics
C Imple-mentation
Proof
Sound worst-case execution time bound
Binary code

Limitations (work in progress):
- Kernel initialisation not yet verified
- MMU & caches modelled abstractly
- Timing channels not ruled out

Real-World Use: Incremental Cyber Retrofit

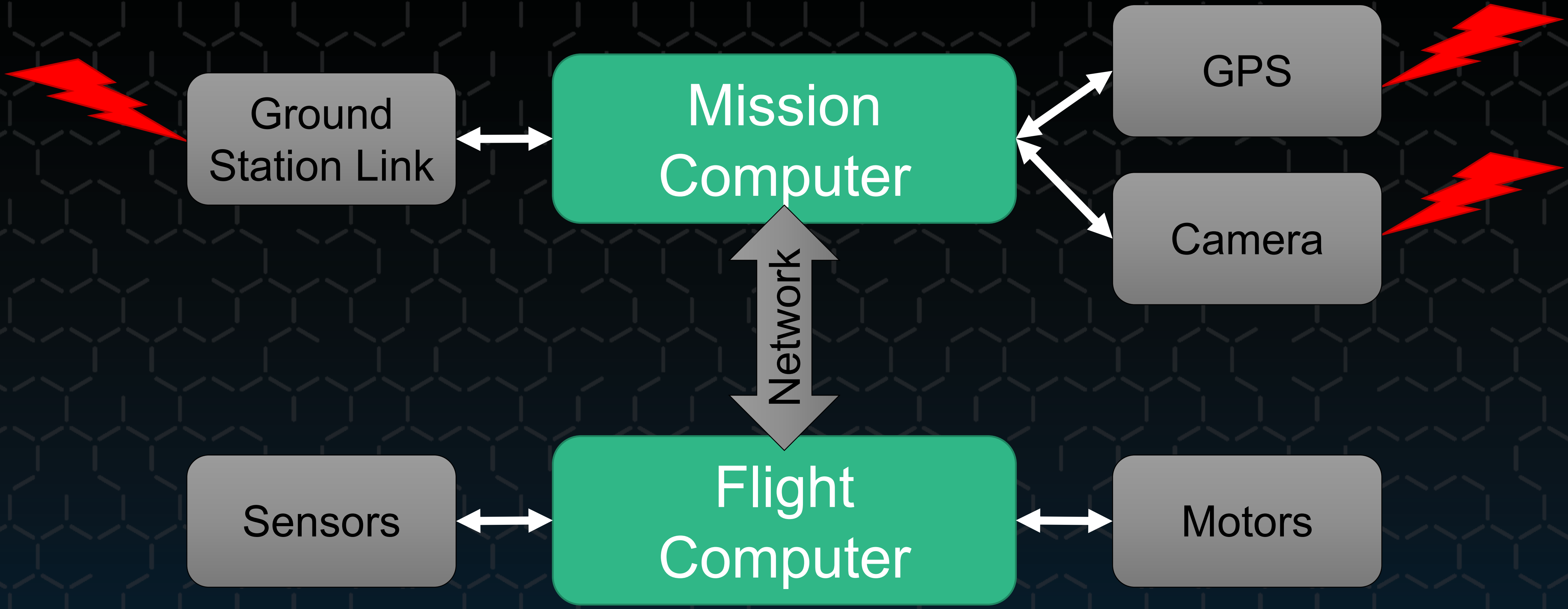
DARPA HACMS

Retrofit existing system!

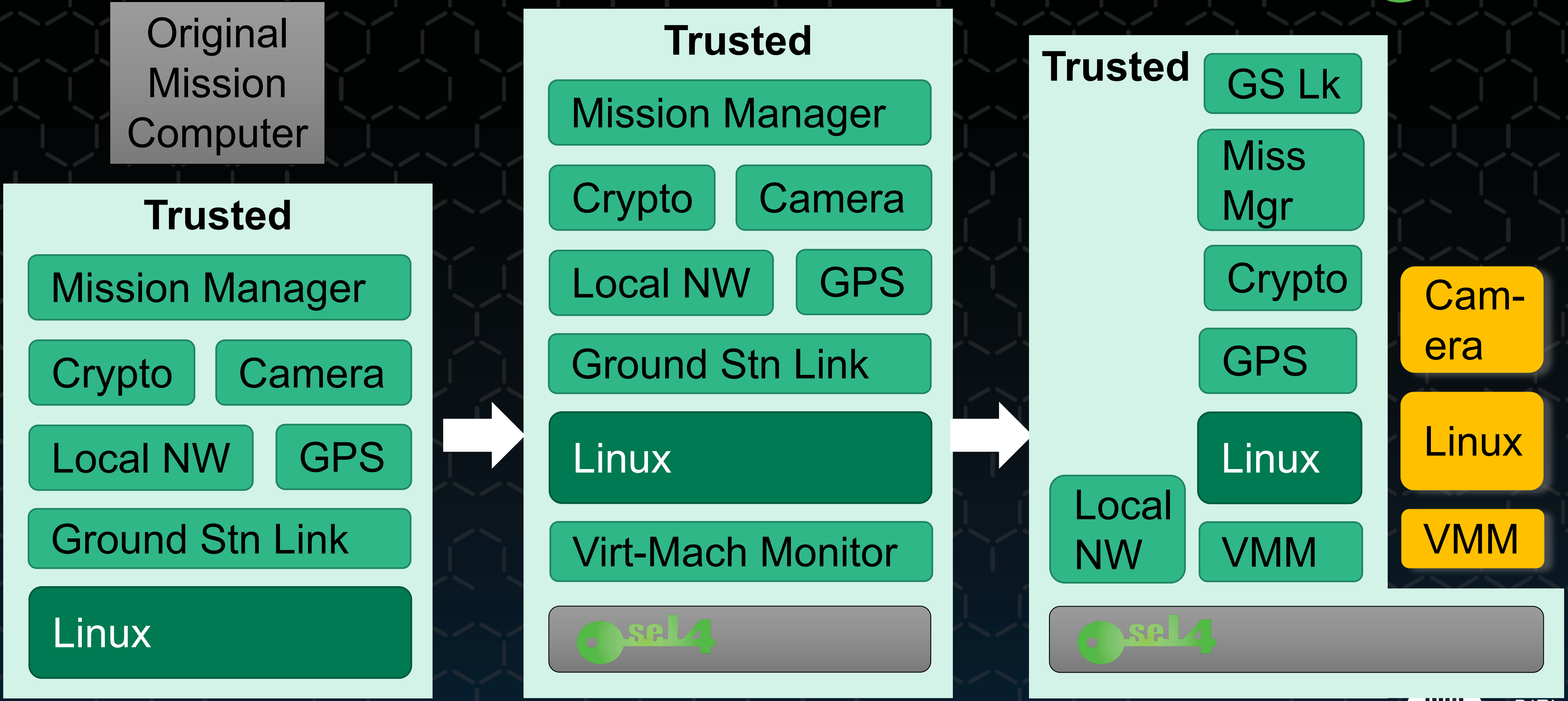


Develop technology

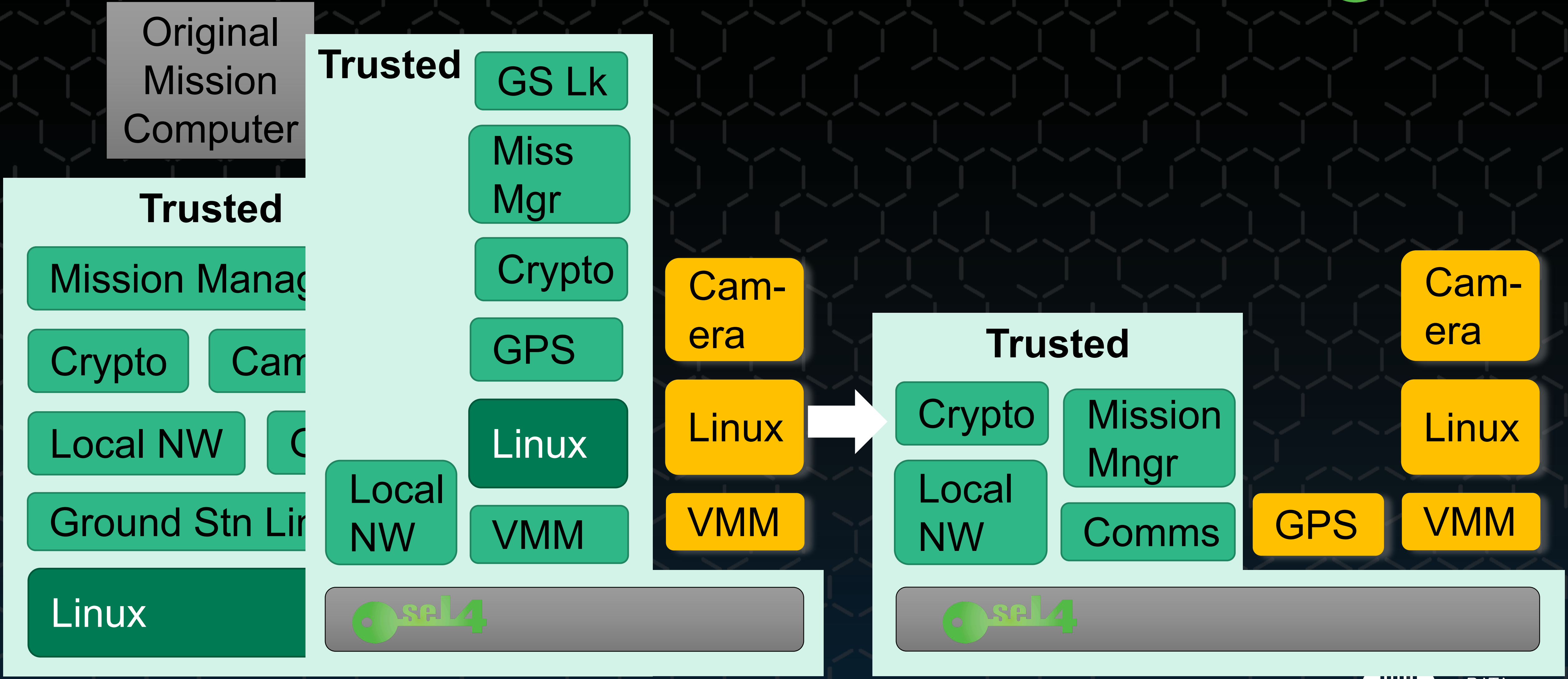
ULB Architecture



Incremental Cyber Retrofit

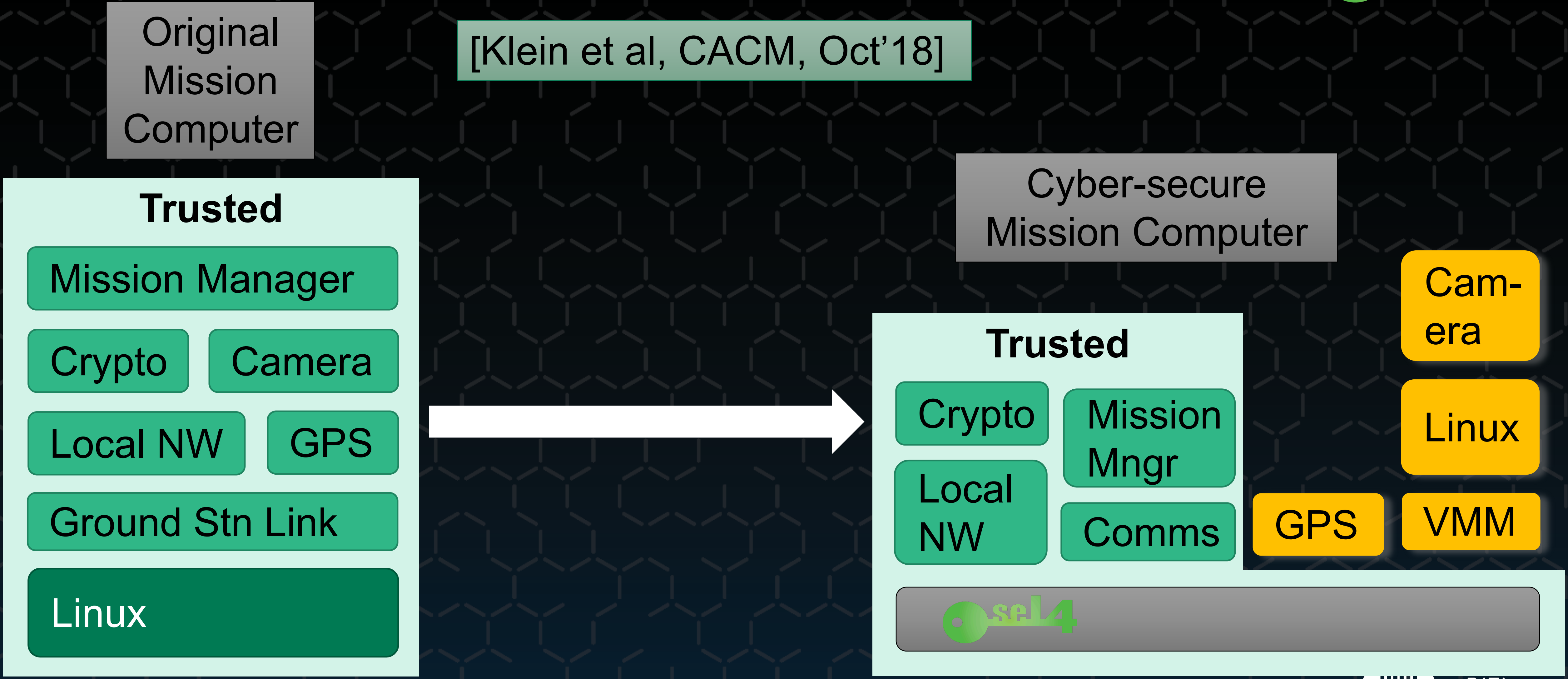


ULB Incremental Cyber Retrofit



Incremental Cyber Retrofit

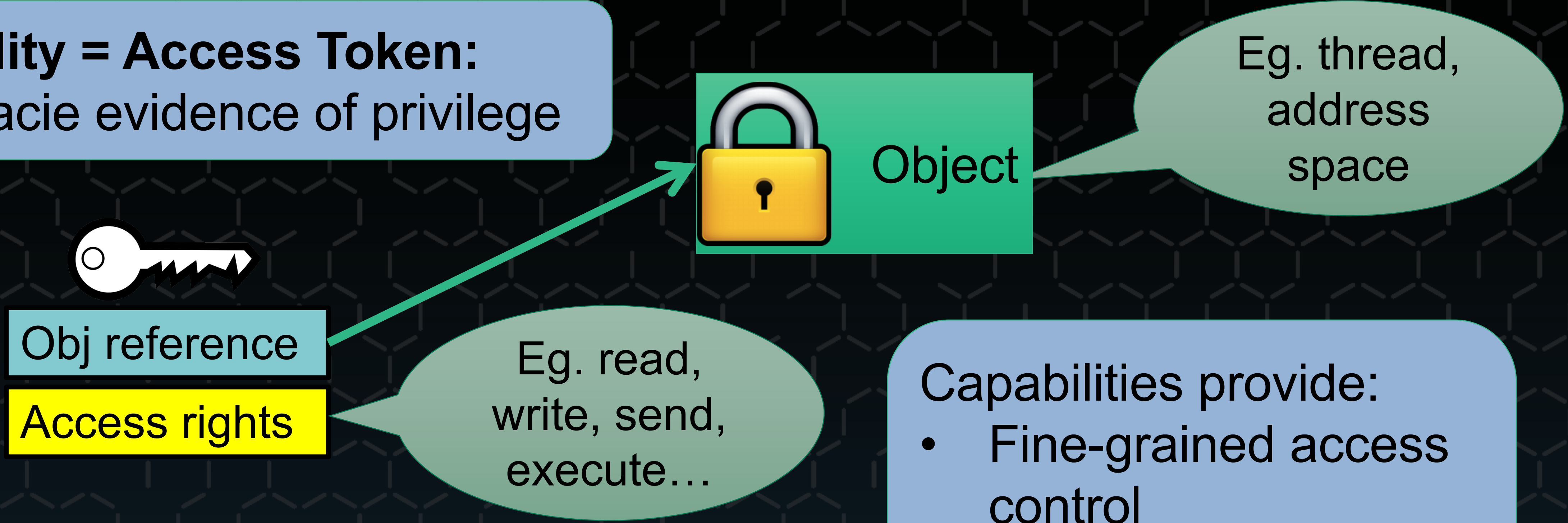
[Klein et al, CACM, Oct'18]



Security by Architecture

Core Security Mechanism: Capability

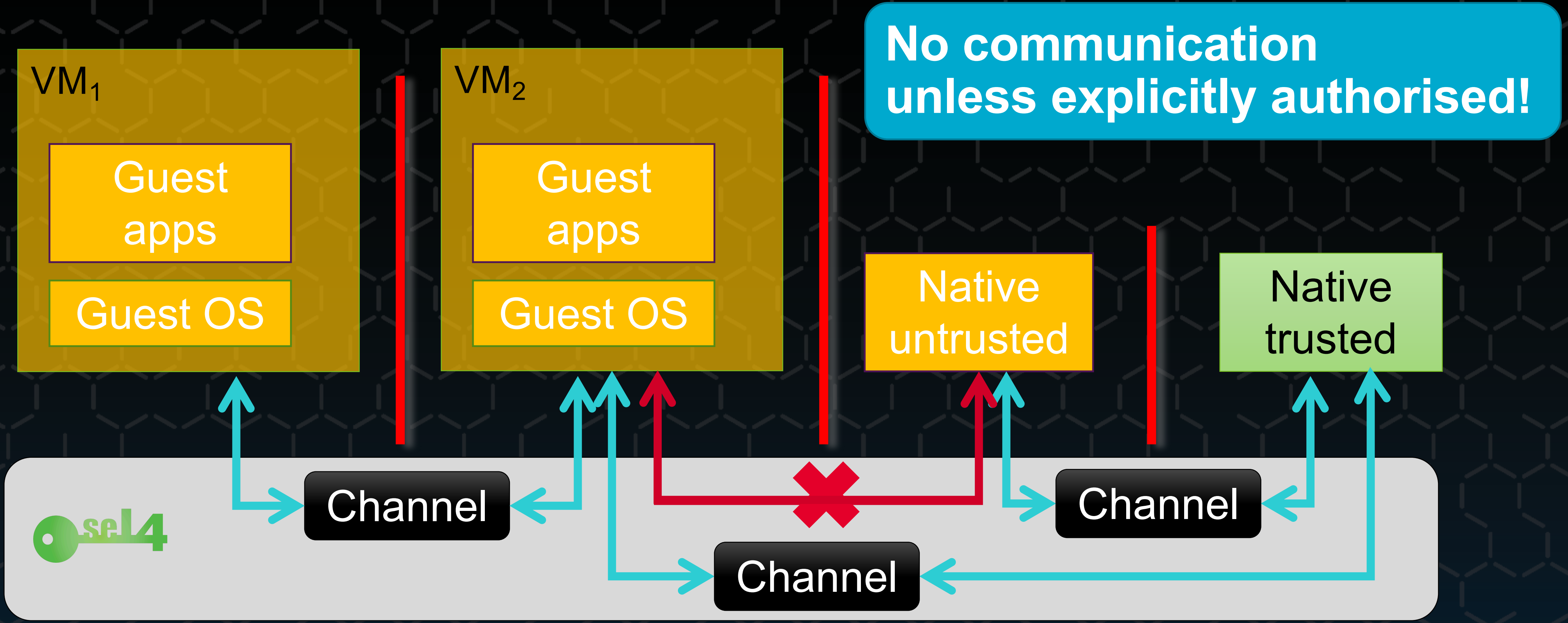
Capability = Access Token:
Prima-facie evidence of privilege



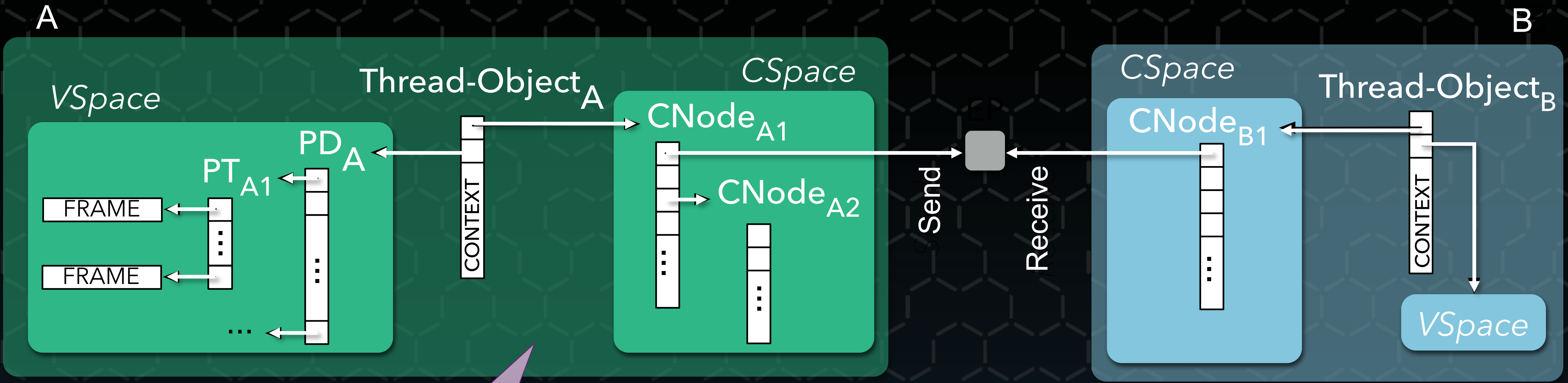
Capabilities provide:
- Fine-grained access control
- Reasoning about information flow

Any system call is invoking a capability:
err = method(cap, args);

Controlled Communication via Caps

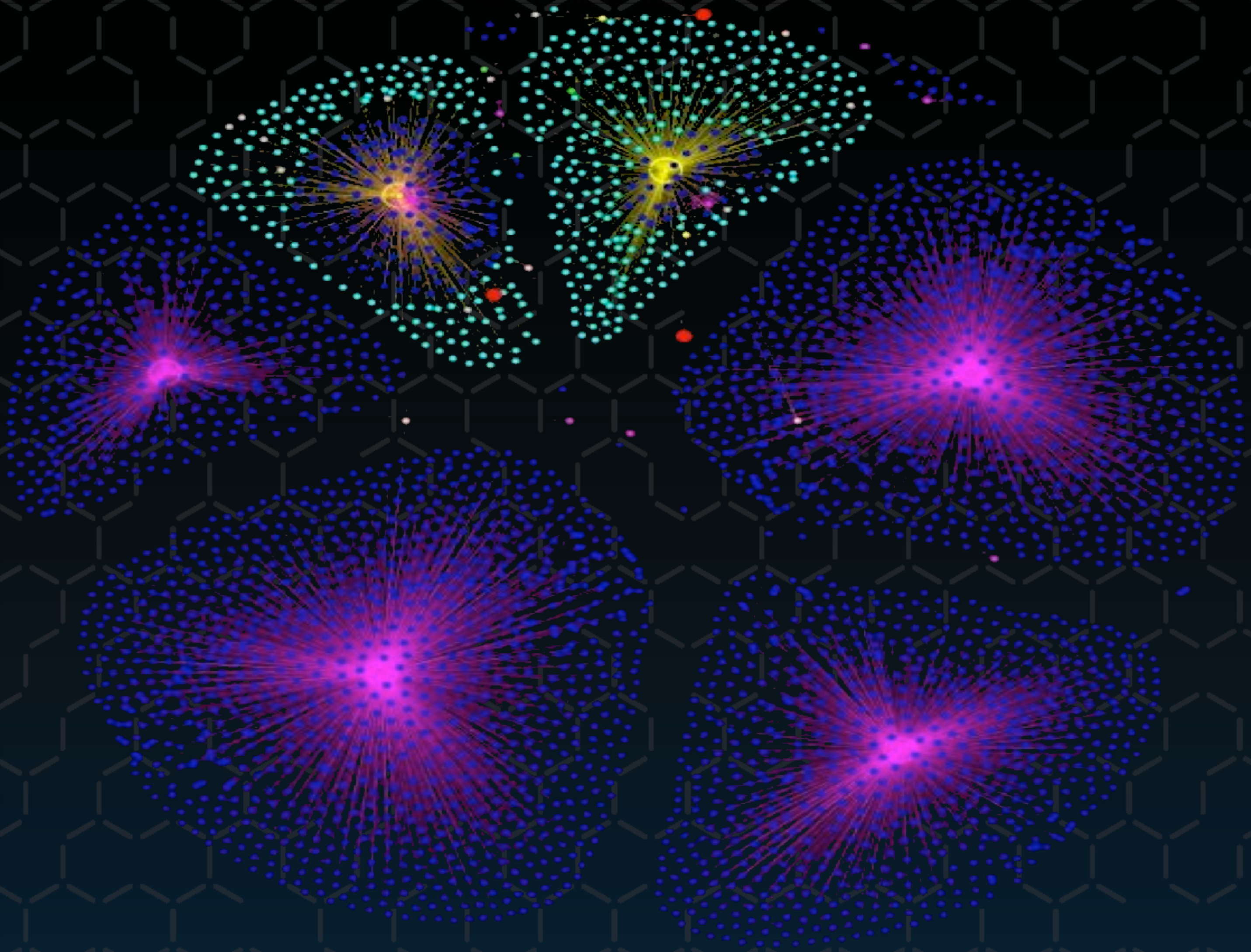


Issue: Capabilities are Low-Level



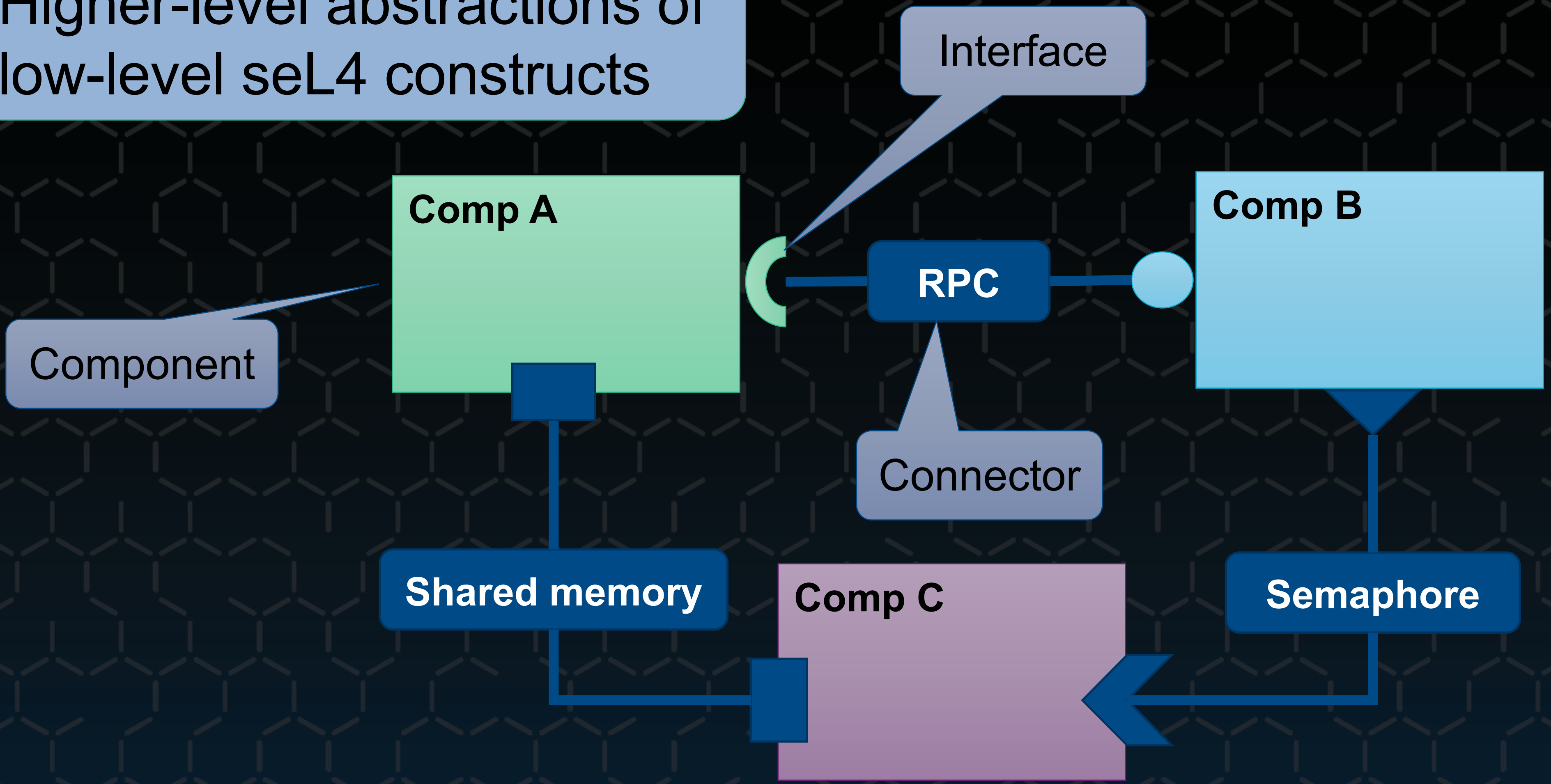
>50 capabilities
for trivial program!

Simple But Non-Trivial System

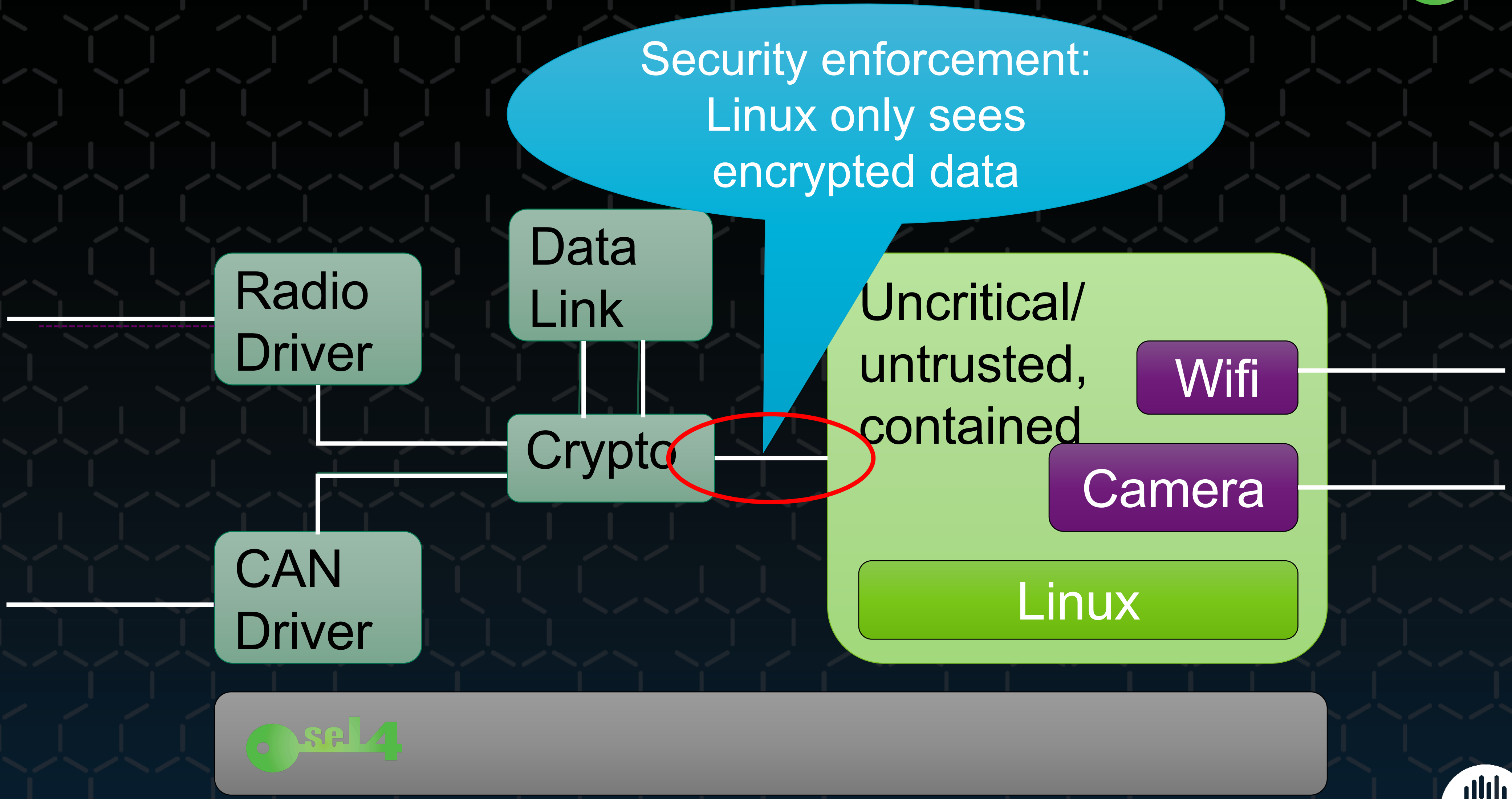


Component Middleware: CAmkES

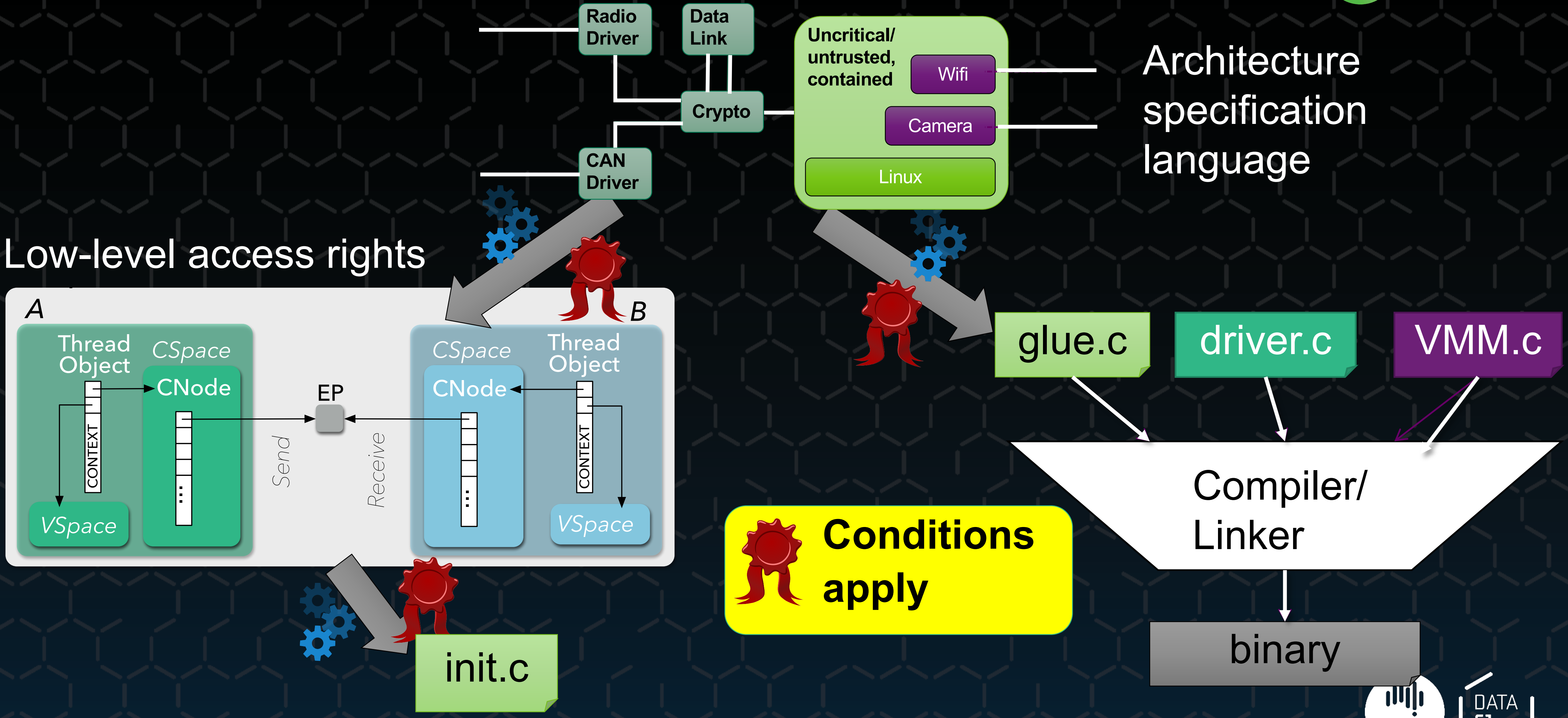
Higher-level abstractions of low-level seL4 constructs



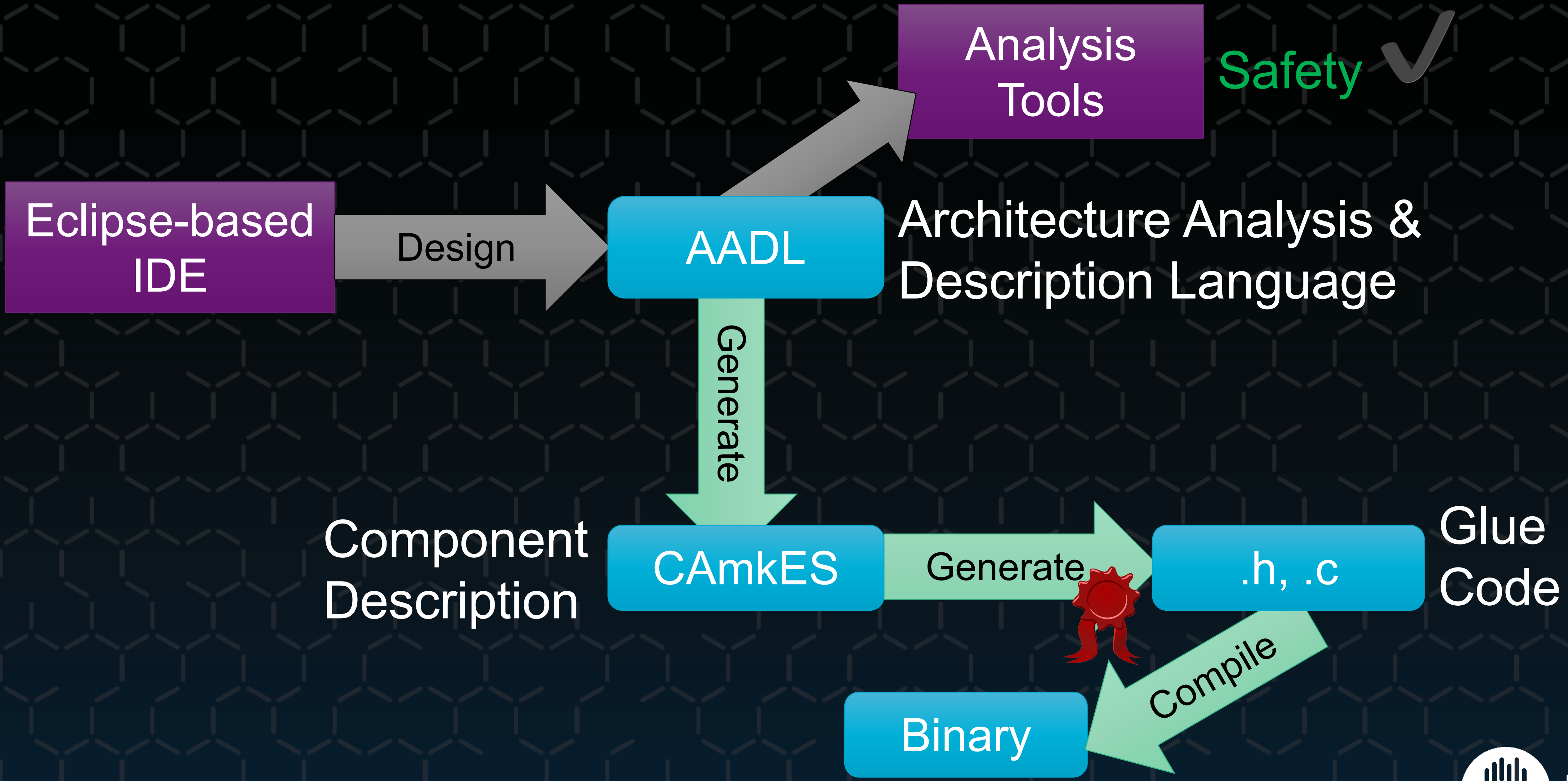
HACMS UAV Architecture


Security enforcement: Linux only sees encrypted data
Radio Driver
Data Link
Crypto
CAN Driver
Uncritical/ untrusted, contained
Wifi
Camera
Linux
seL4

Enforcing the Architecture



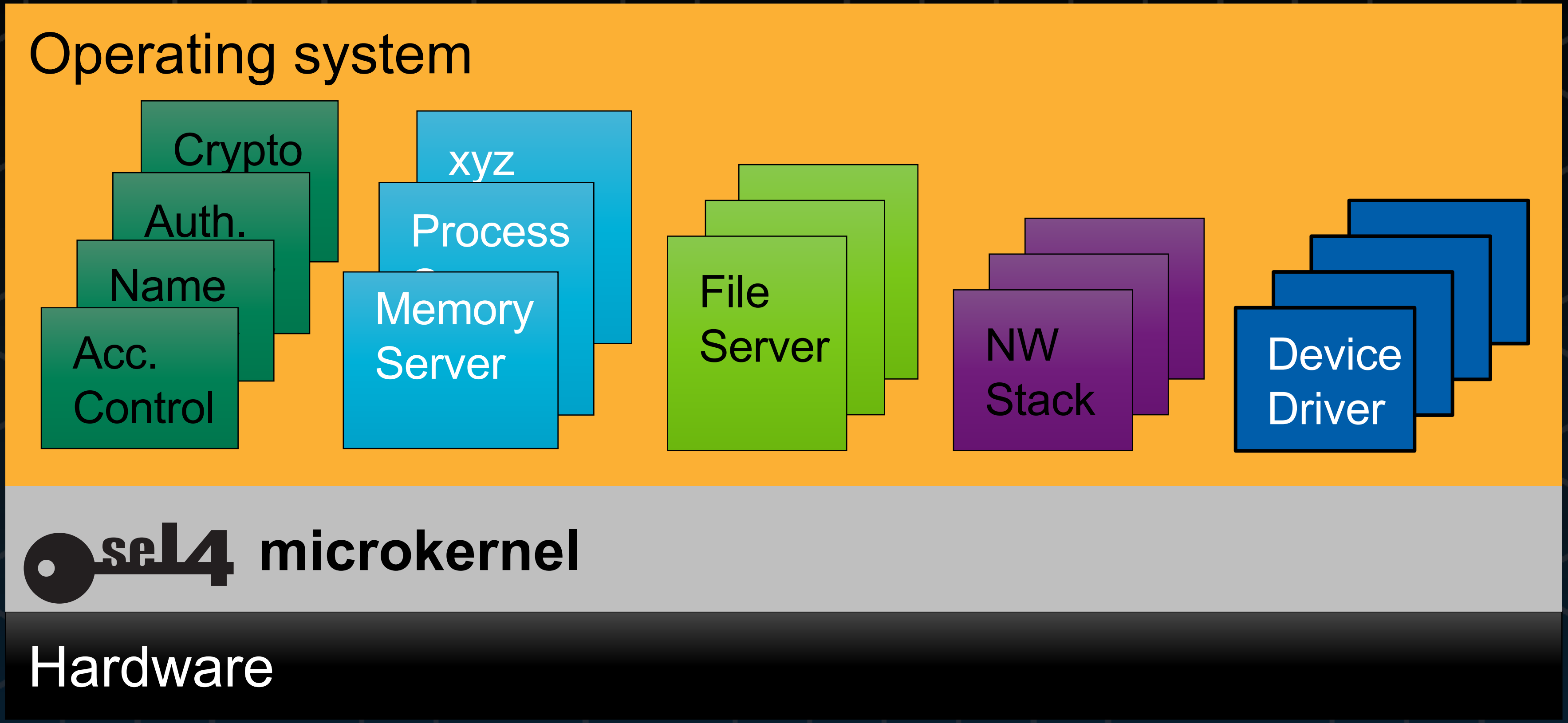
Architecture Analysis



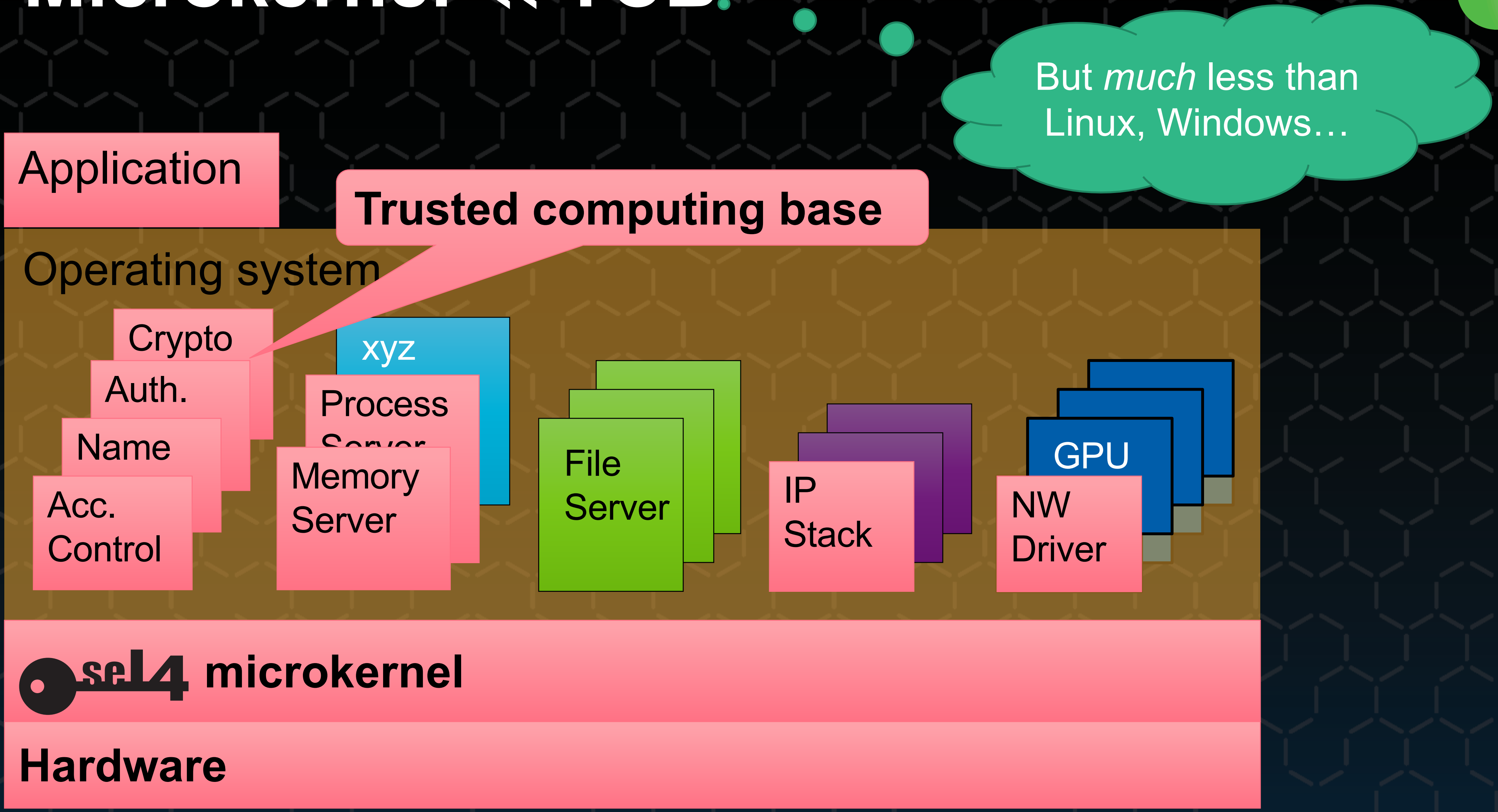
High Assurance Beyond the Kernel

Microkernel ≪ TCB

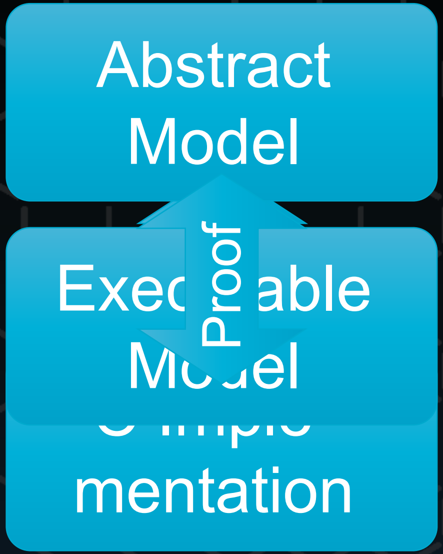
OS structured in *isolated* components, minimal inter-component dependencies, *least privilege*



Microkernel ≪ TCB



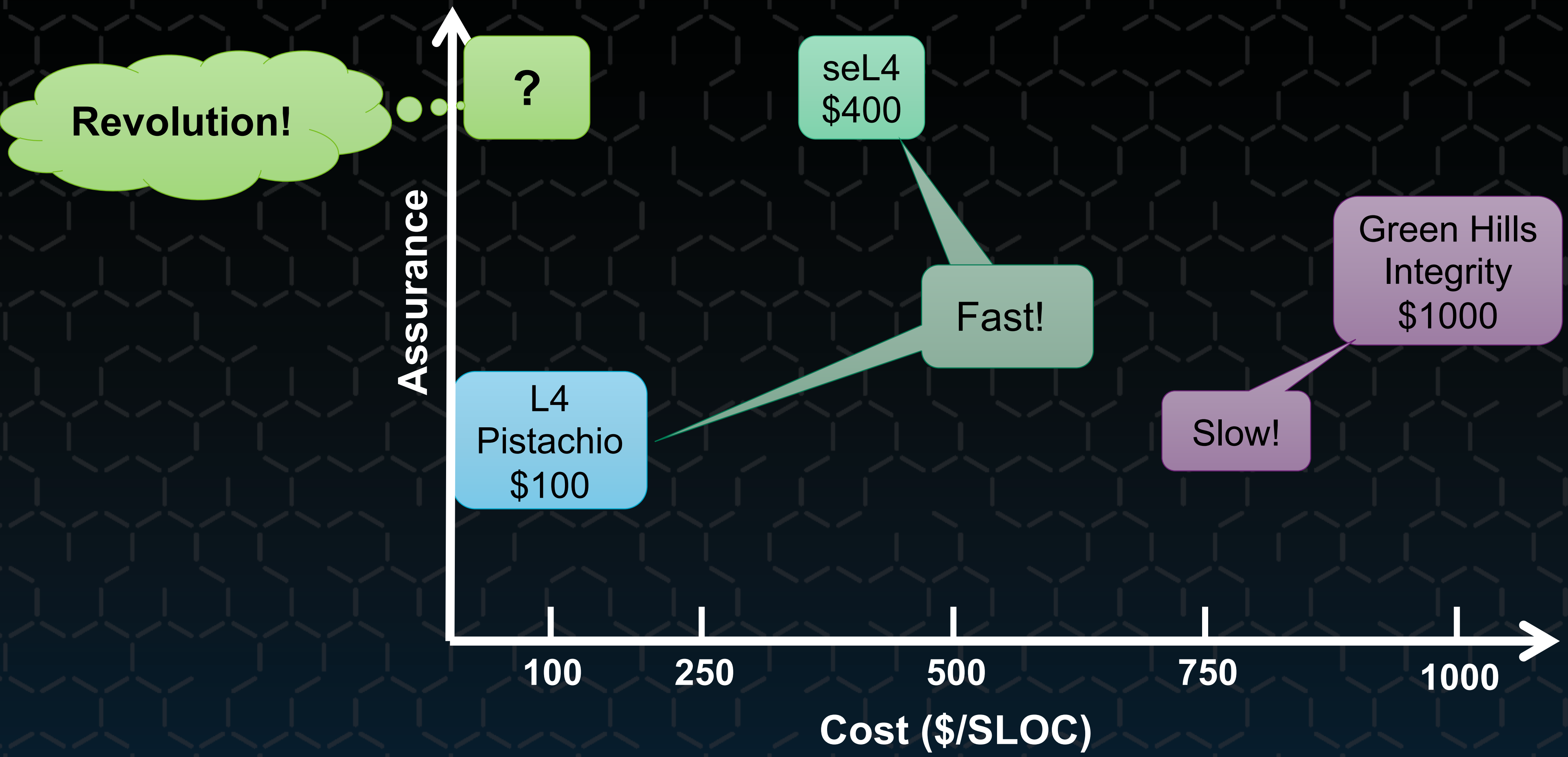
Verification Cost



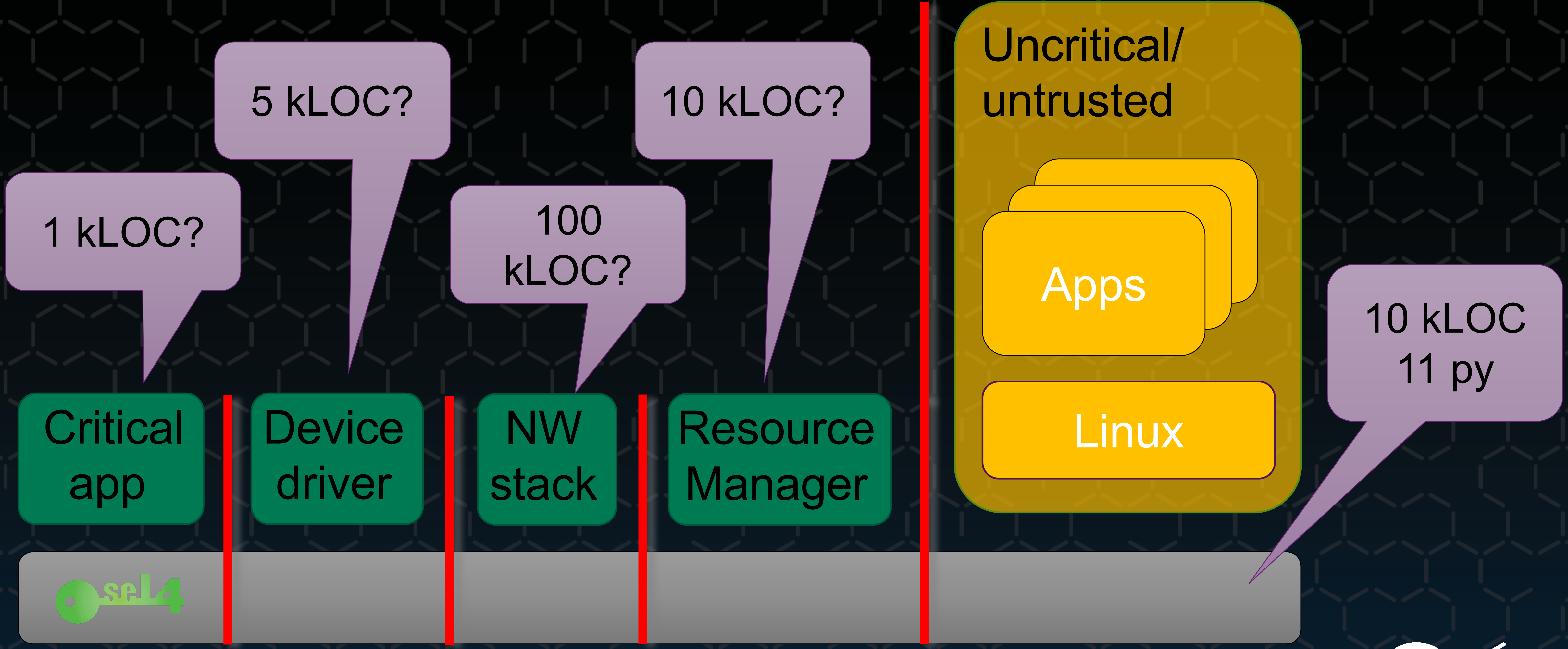
120,000 LoP, 8 py

50,000 LoP, 3 py

Life-Cycle Cost in Context



Beyond the Kernel

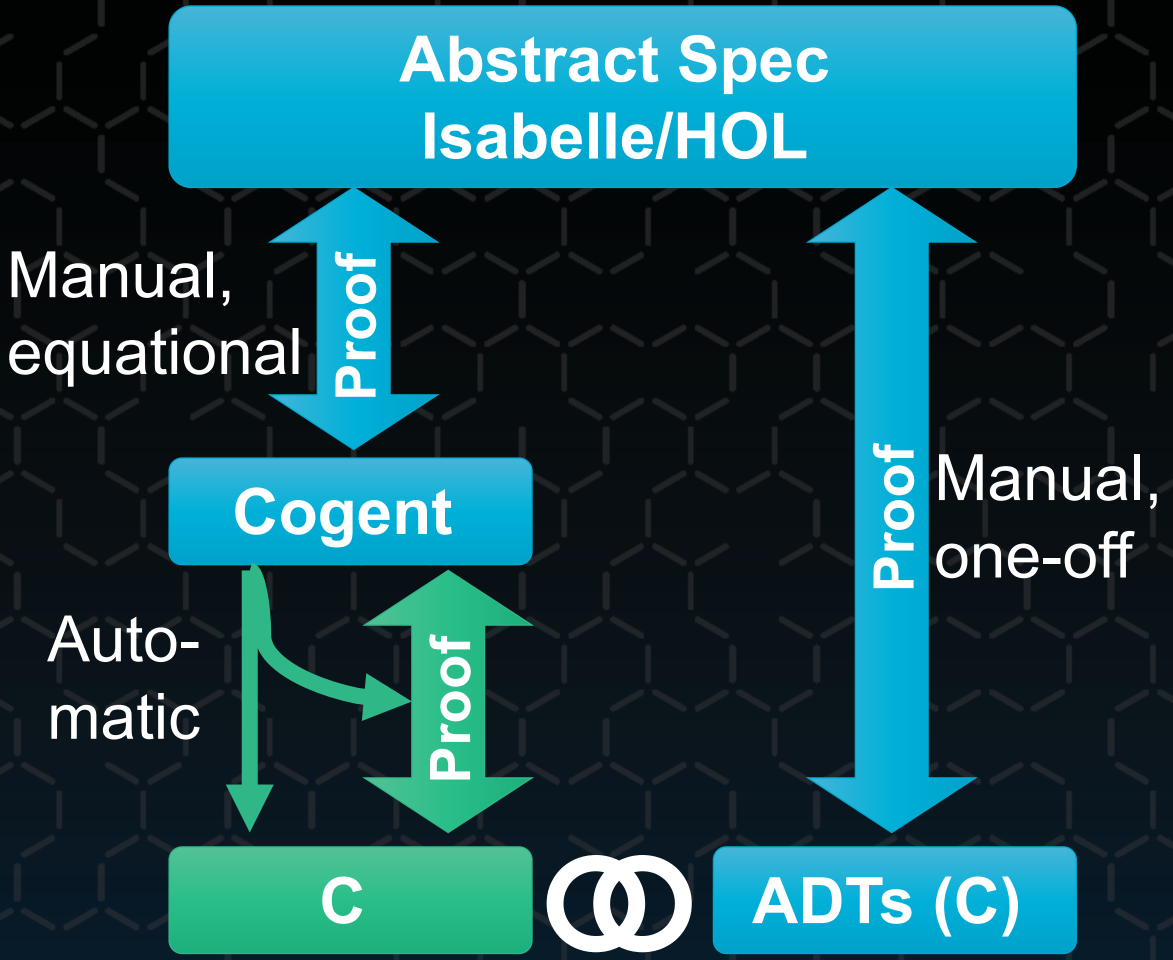


Cogent: Code & Proof Co-Generation

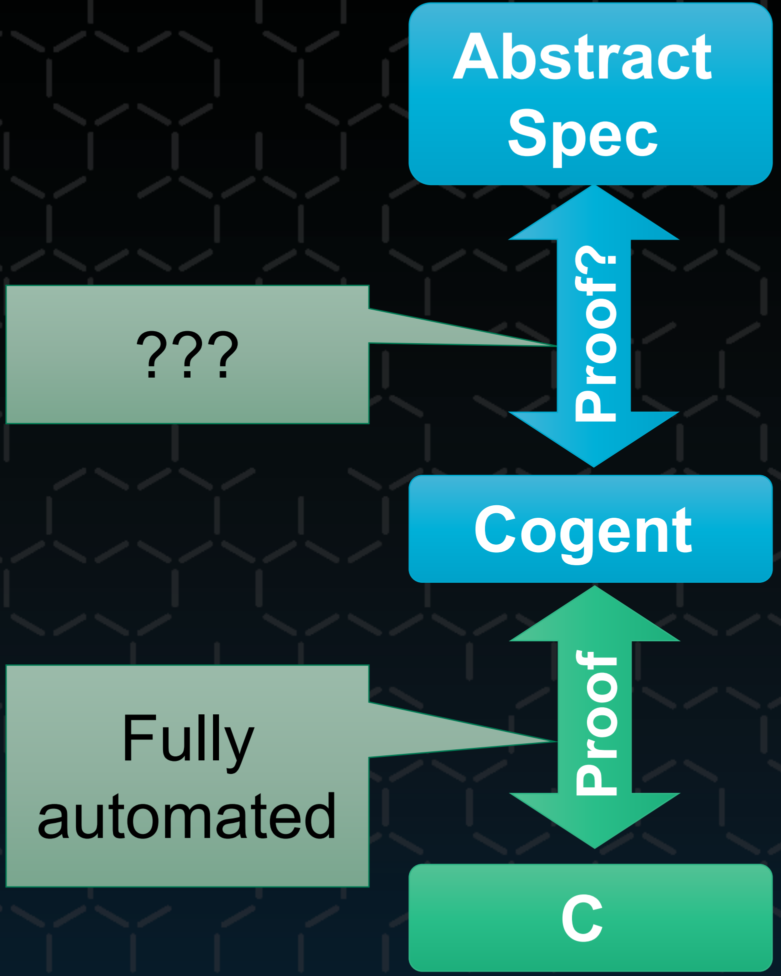
Aim: Reduce cost of verified *systems* code

- 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]



Dependable And Affordable?



Dependability-cost tradeoff:

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

Spec reuse!

Work in progress:

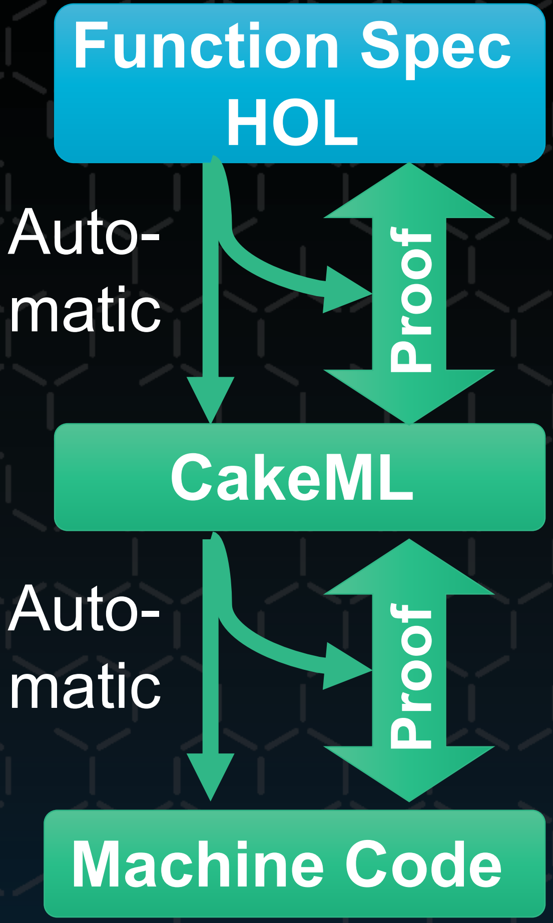
- Language expressiveness
- Reduce boiler-plate code
- Use for network stacks
- Use for device drivers

CakeML: Syntesising Code & Proofs

Aim: Reduce cost of verified *applications* code

- Impure, general-purpose functional language
- Type-safe, managed, garbage-collected, not memory-safe, Turing complete
- Verified run-time (GC etc)
- Compiles to binary for Armv6/8, x86, MIPS62, RISC-V
- Competitive performance

[Tan et al., ICFP'16]



CAmkES glue-code verification in progress

Time Protection: Systematic Prevention of Timing Channels

Threats

=

Speculation

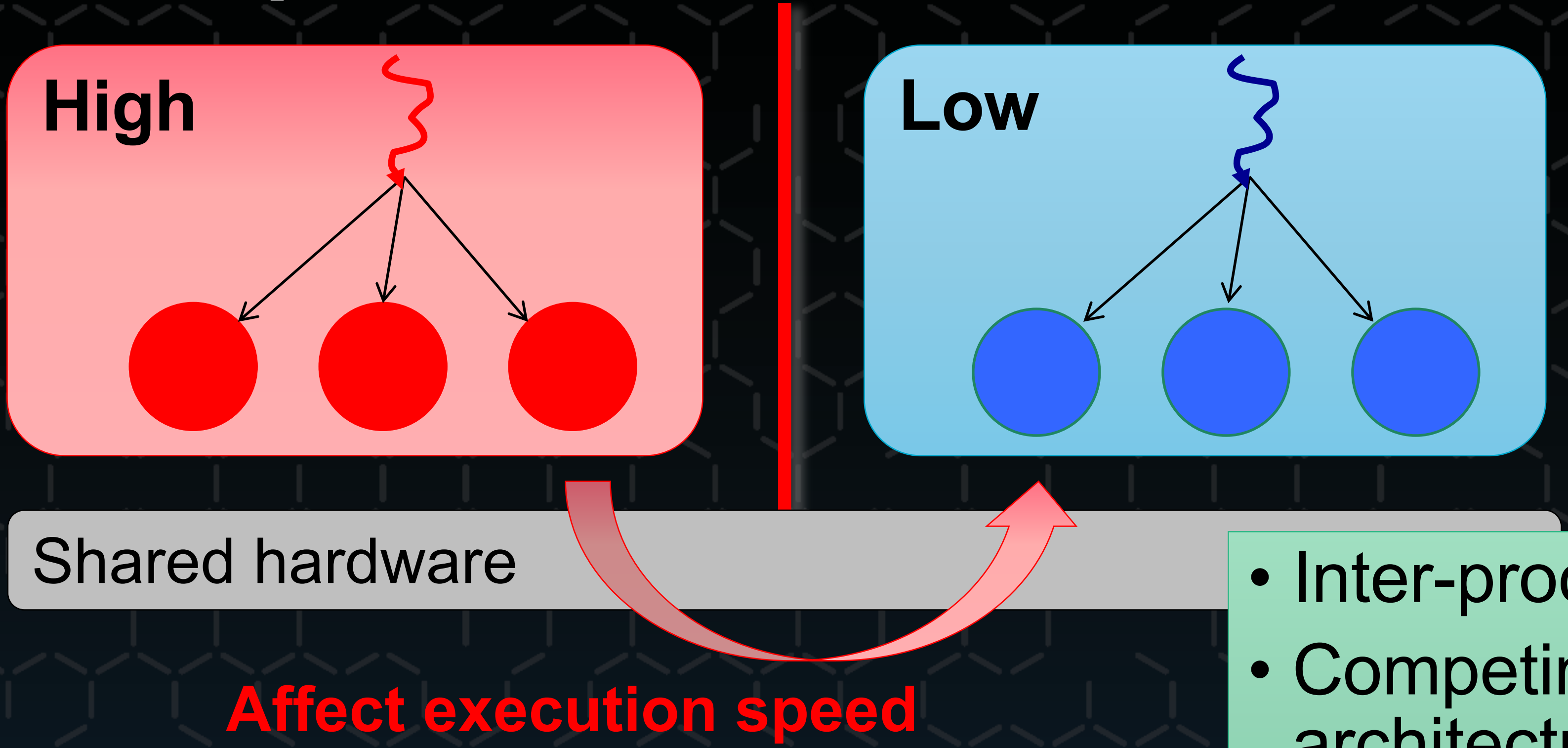
An "unknown unknown" until recently

+

A "known unknown" for decades

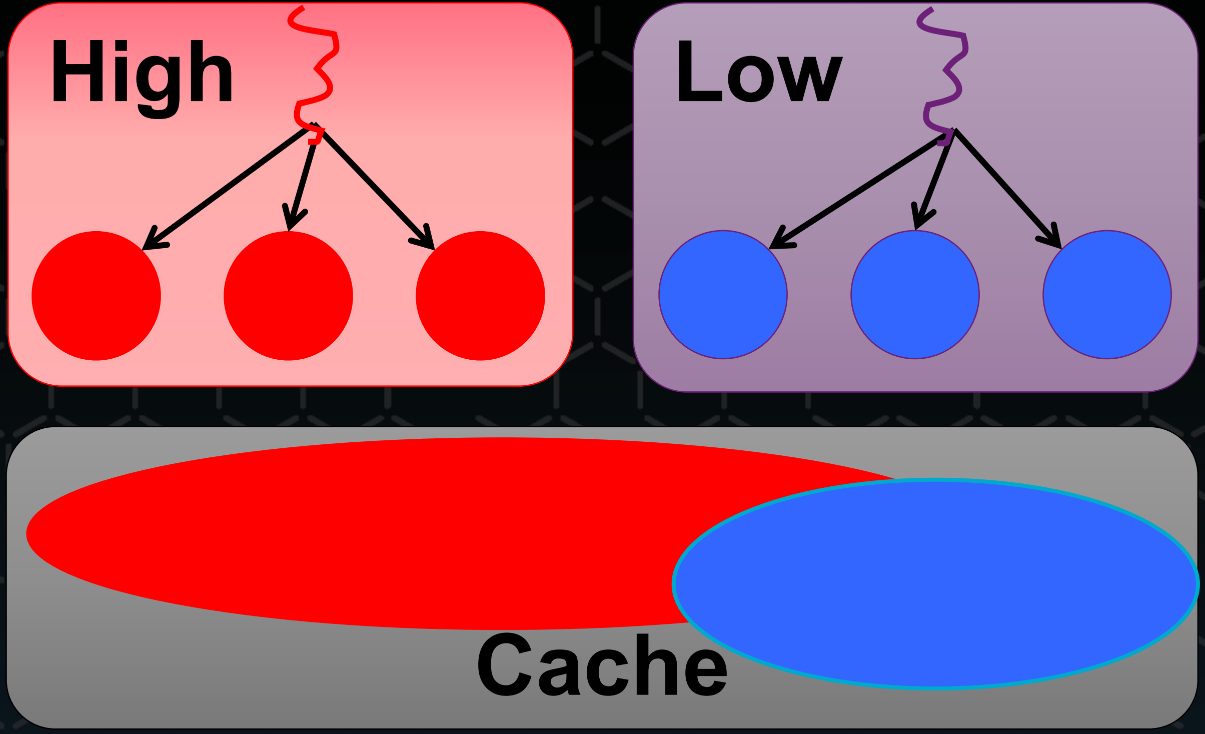
Microarchitectural Timing Channel

Cause of Timing Channels: Competition for HW Resources



- Inter-process interference
- Competing access to micro-architectural features
- Hidden by the HW-SW contract!

Sharing: Stateful Hardware



HW is *capacity-limited*

- Interference during
  - concurrent access
  - time-shared access
- Collisions reveal addresses
- *Usable as side channel*

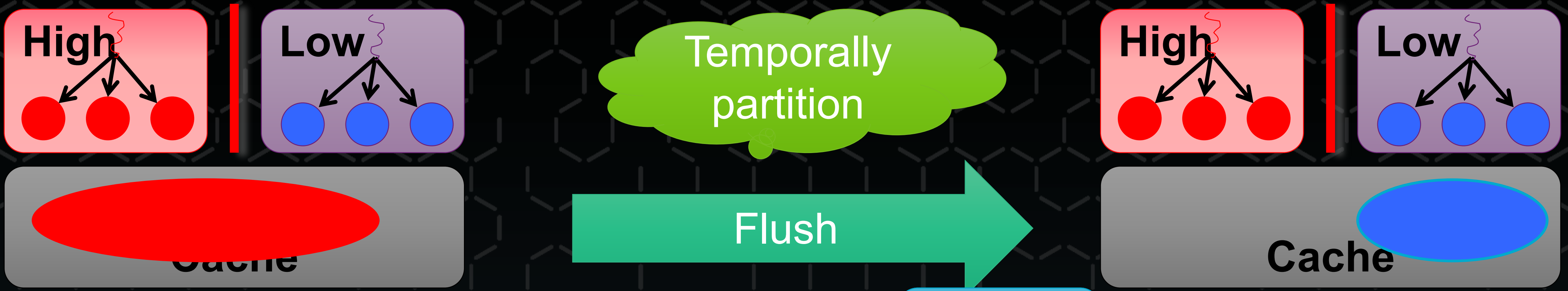
Any state-holding microarchitectural feature:

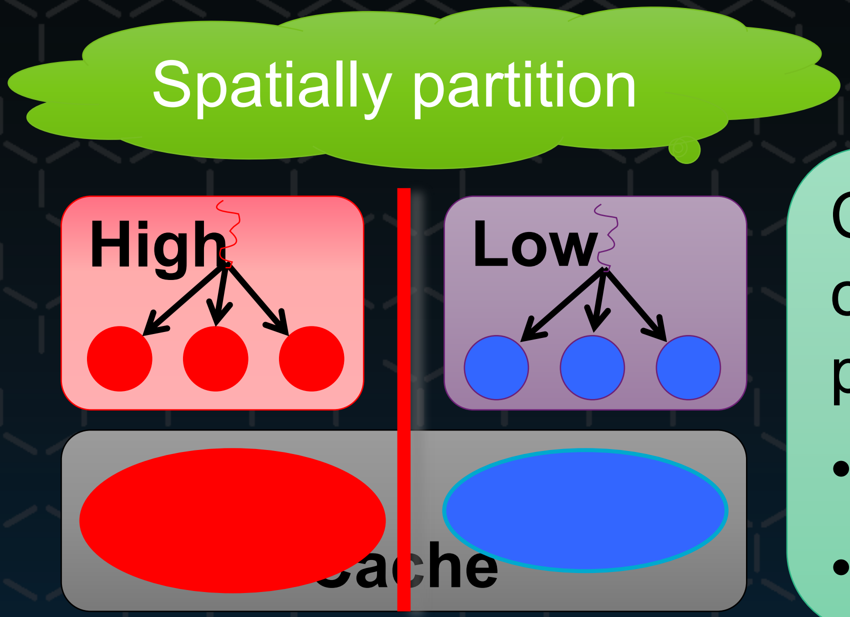
- cache, branch predictor, pre-fetcher states

Timing-channel prevention: Partition hardware:

- spatially
- temporally (time shared)

Time Protection: Partition Hardware





Need both!

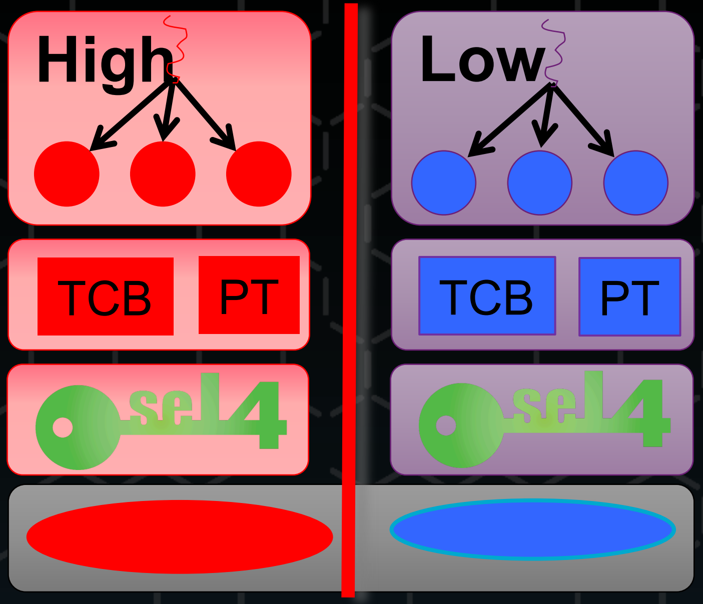
Cannot spatially partition on-core caches (L1, TLB, branch predictor, pre-fetchers)

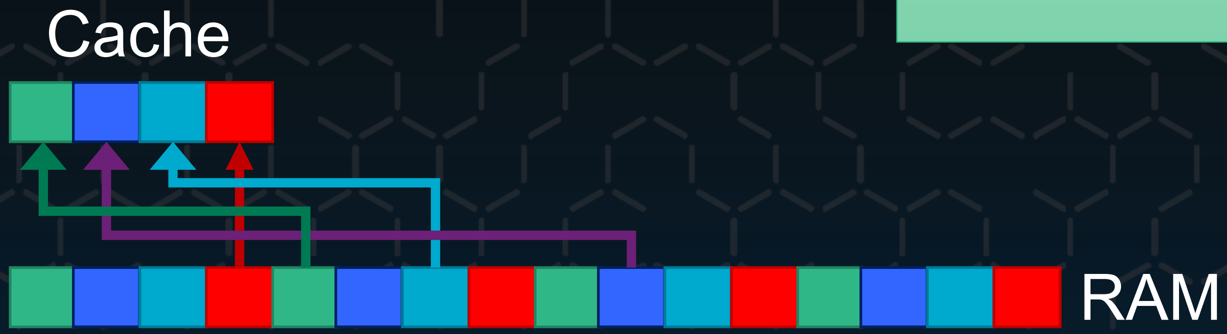
- virtually-indexed
- OS cannot control

Flushing useless for concurrent access

- HW threads
- cores

Spatially Partition: Cache Colouring





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

[Ge et al. EuroSys'19]

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

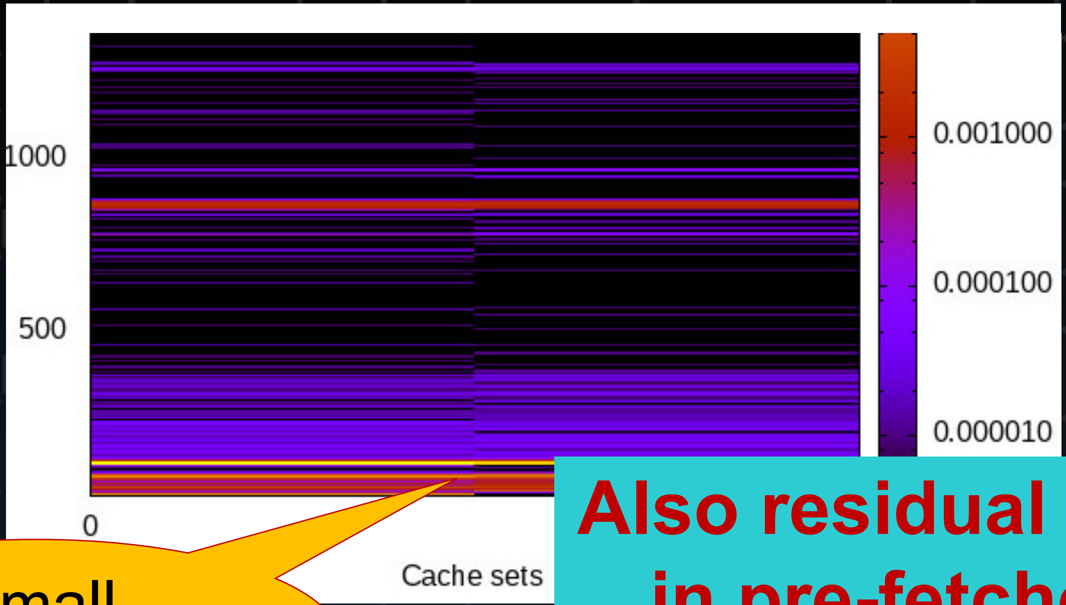
Ensure deterministic execution

Challenge: Broken Hardware

- Systematic study of COTS hardware (Intel and Arm) [Ge et al, APSys'18]:
  - contemporary processors hold state that cannot be reset
  - need a new hardware-software contract to enable real security

RISC-V will provide suitable contract

Security Standing Committee agrees

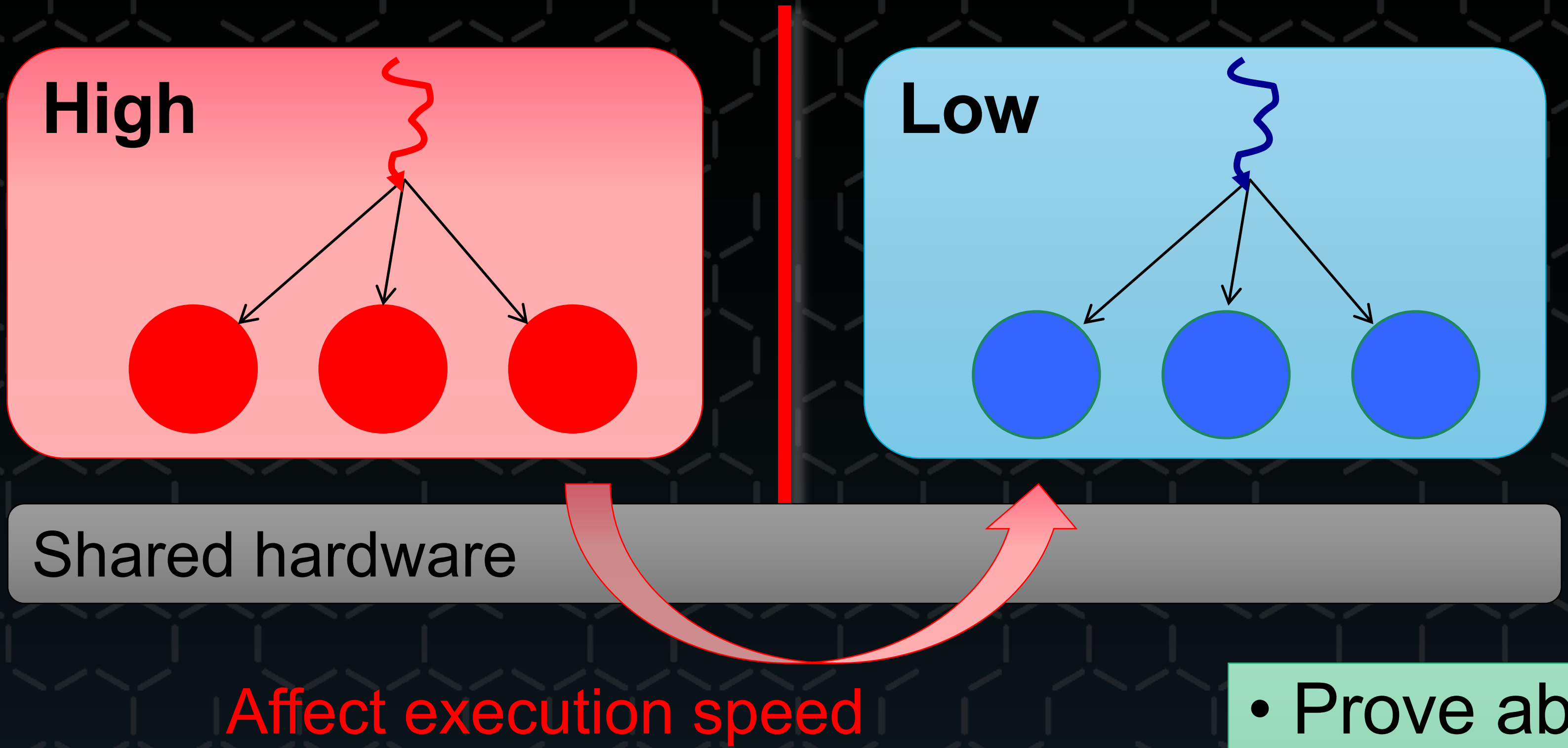


Small channel!

Also residual state in pre-fetchers

Can We Verify Time Protection?

Competition for HW Causes Channels



- Prove absence of interference, ⇒ no channels possible
- Must prove correct partitioning!

Can Time Protection Be Verified?

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

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!

How Can We Prove Time Padding?

- Idea: Minimal formalisation of hardware clocks (logical 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

Functional property

THANK YOU

Gernot Heiser | gernot@unsw.edu.au | @GernotHeiser
FM@Scale, 9 Oct 2019

https://trustworthy.systems

New HW/SW Contract: aISA

Augmented ISA supporting time protection

Security Standing Committee agrees

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
   - Implies 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
7. Desirable: Flush only affects state that *must* be flushed

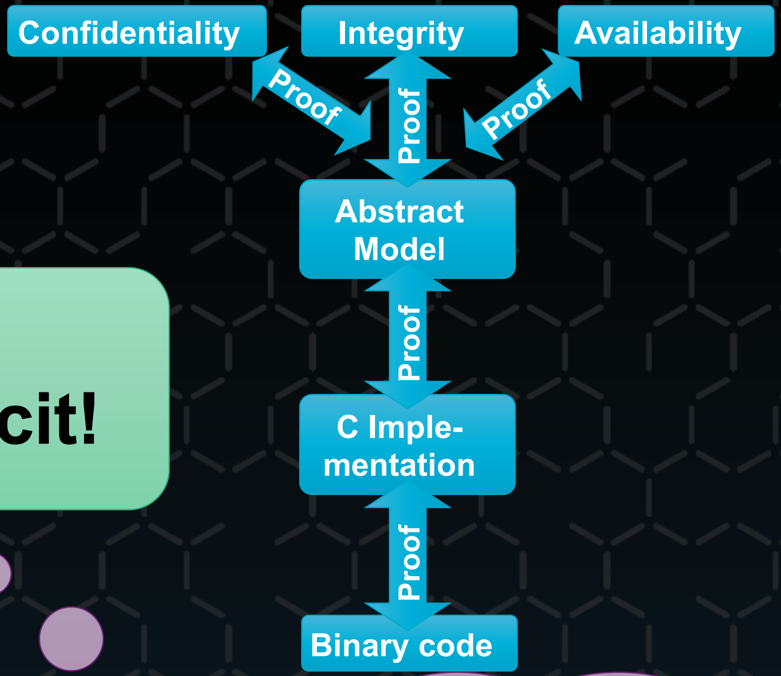
Verification Guarantees

Verification rules out unspecified behaviour:

- Buffer/stack overflow
- Null-pointer dereference
- Code injection
- Use after free
- Memory leaks
- Kernel crash
- Privilege escalation
- Covert *storage* channels, …

… as long as the assumptions are satisfied!

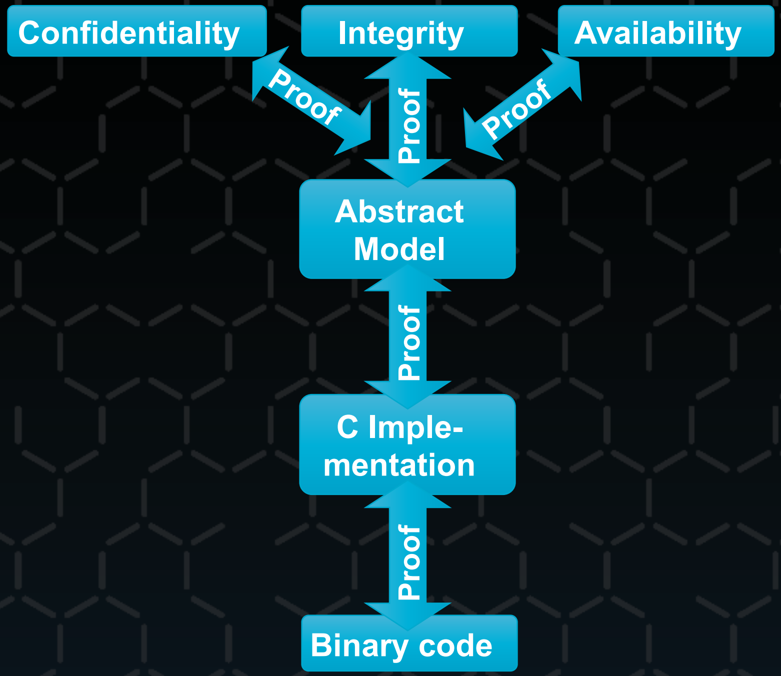
Verification forces you to **make assumptions explicit!**



Reason many bugs are found just from writing the spec!

Verification Assumptions

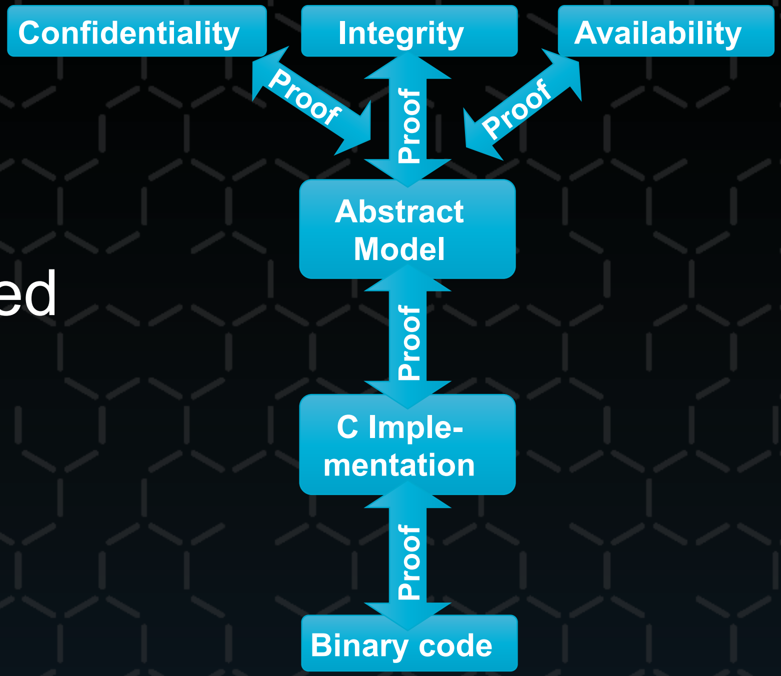
1. Hardware behaves as expected
   - Formalised hardware-software contract (ISA)
   - Hardware implementation free of bugs, Trojans, …
2. Spec matches expectations
   - Can only prove “security” if specify what “security” means
   - Spec may not be what we think it is
3. Proof checker is correct
   - Isabel/HOL checking core that validates proofs against logic



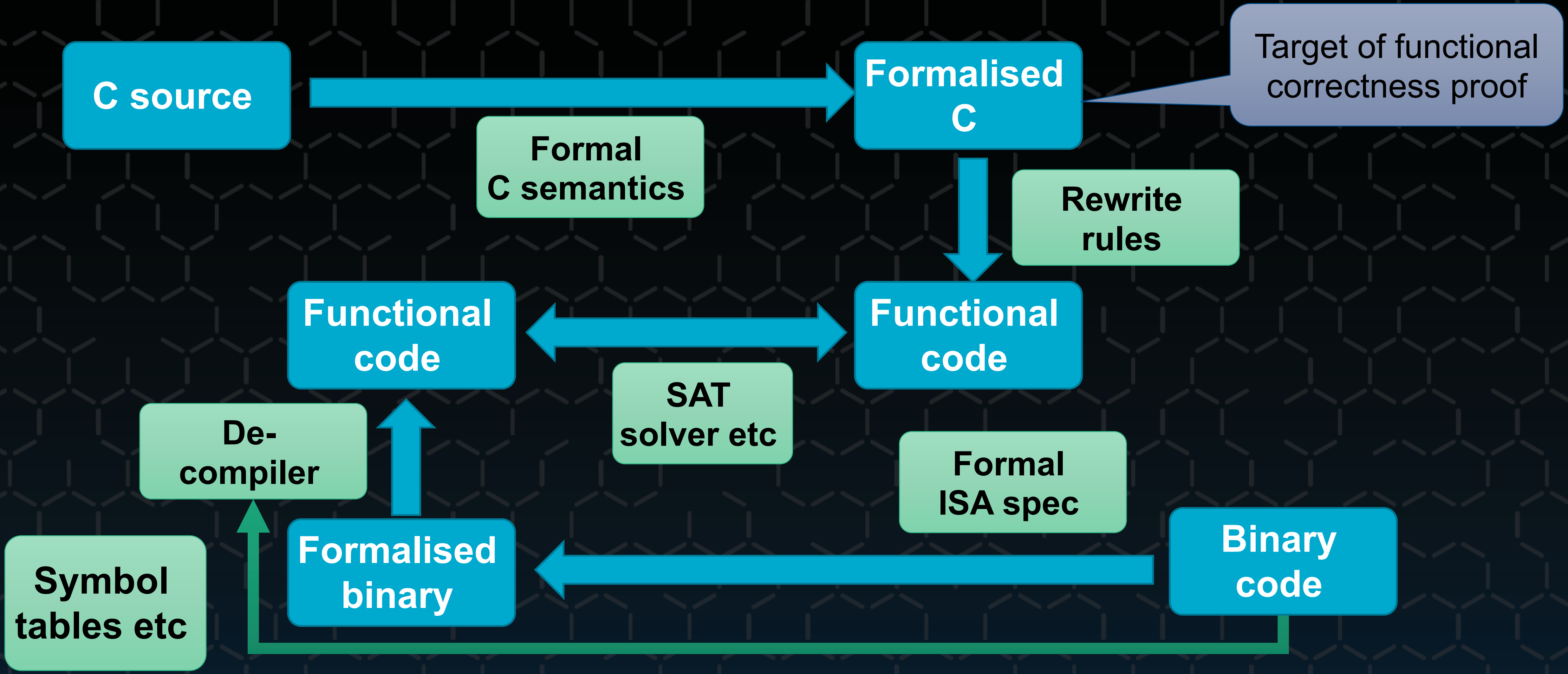
With binary verification do **not** need to trust C compiler!

Present Verification Limitations

- Not verified boot code
  - **Assume** it leaves kernel in safe state
  - Verification in progress
- Caches/MMU presently modeled at high level / axiomised
  - This is in progress of being fixed, MMU model done
- Not proved any temporal properties
  - Presently not proved scheduler observes priorities, properties needed for RT
  - Worst-case execution-time analysis applies only to dated ARM11/A8 cores
  - No proofs about timing channels yet



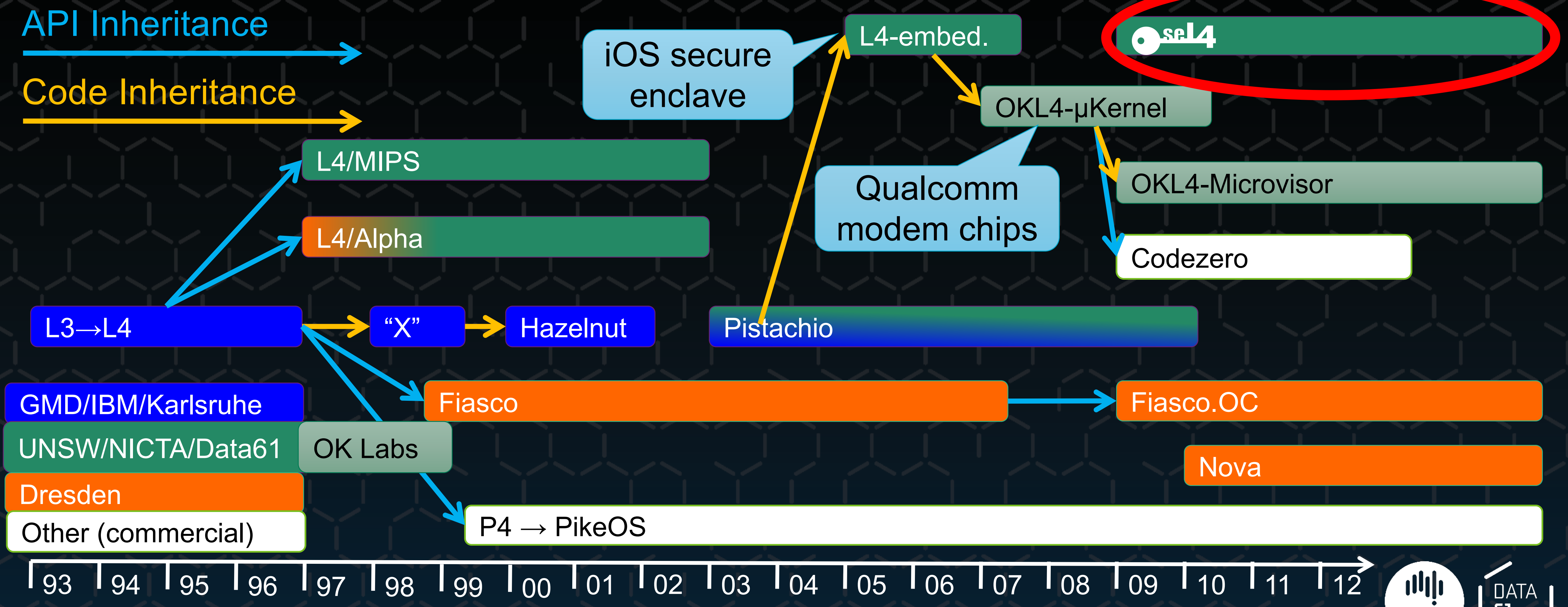
Translation Validation



What is seL4?

L4: 25 Years High-Performance Microkernels

seL4: The latest member of the L4 microkernel family

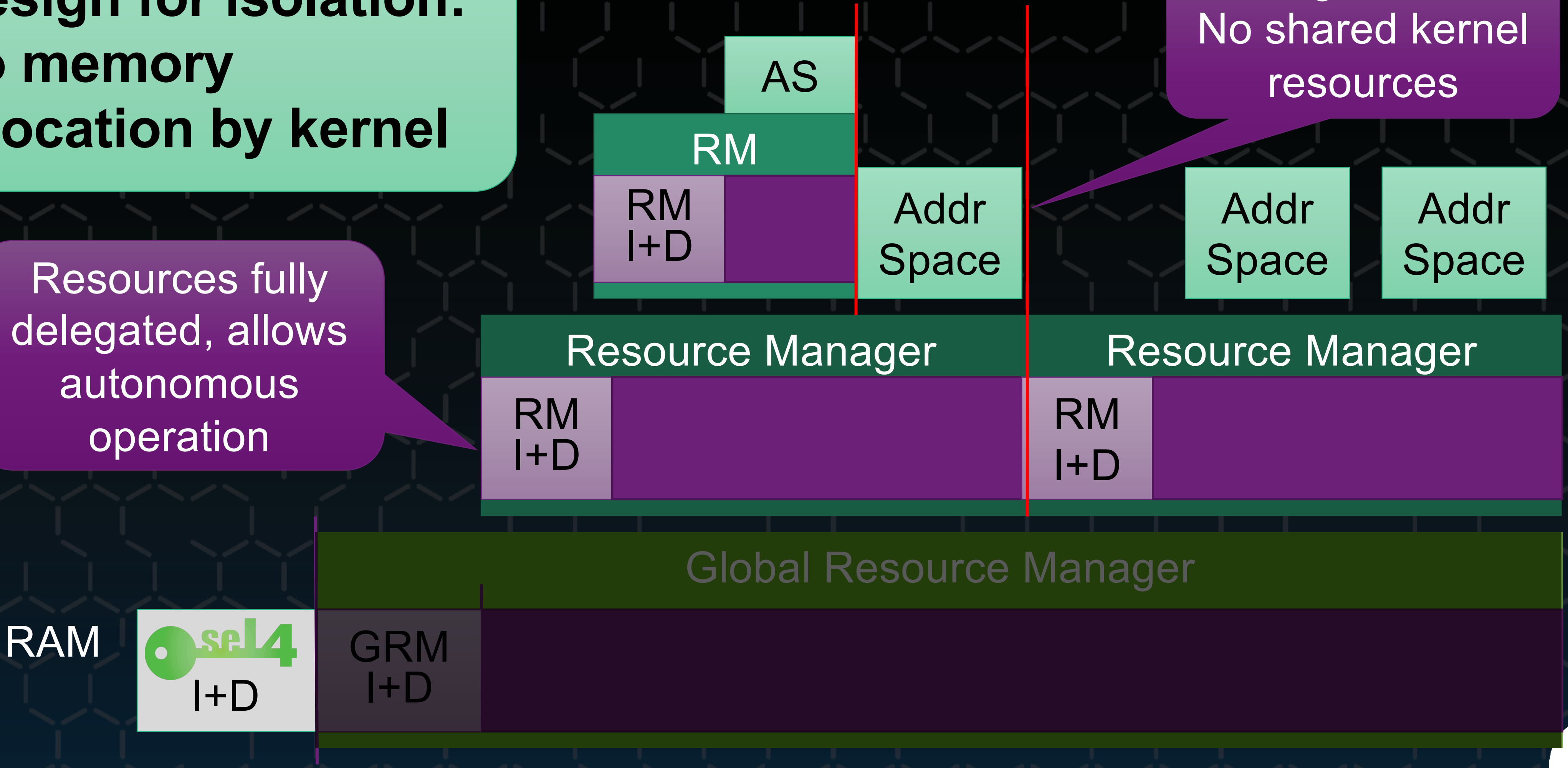


Difference To Other OS Kernels

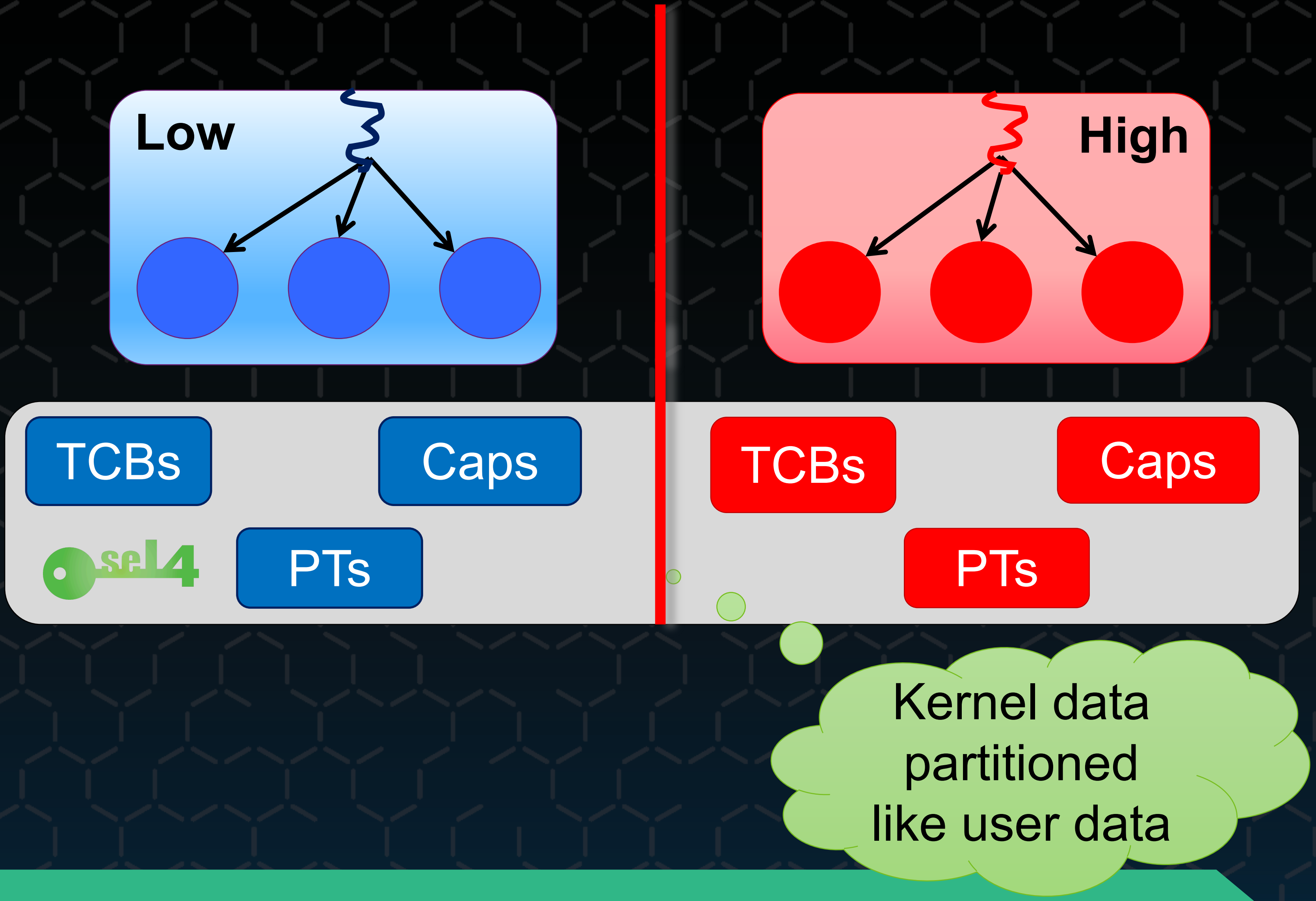
Design for isolation: no memory allocation by kernel

Resources fully delegated, allows autonomous operation

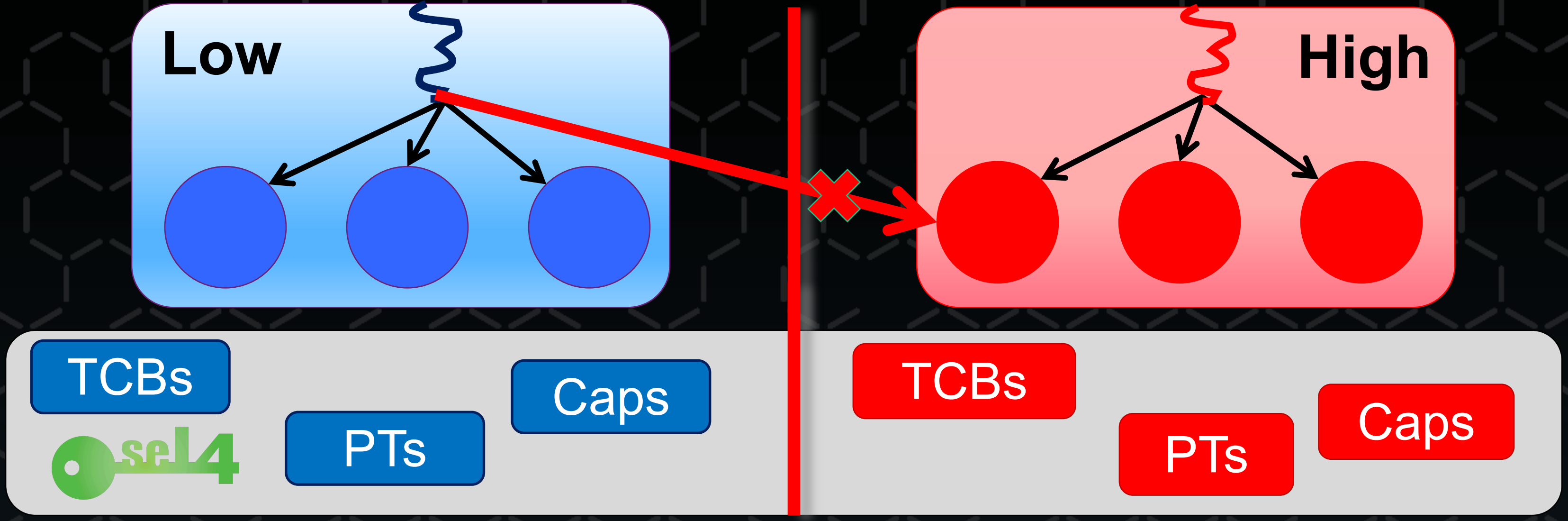
Strong isolation, No shared kernel resources



Isolation Goes Deep



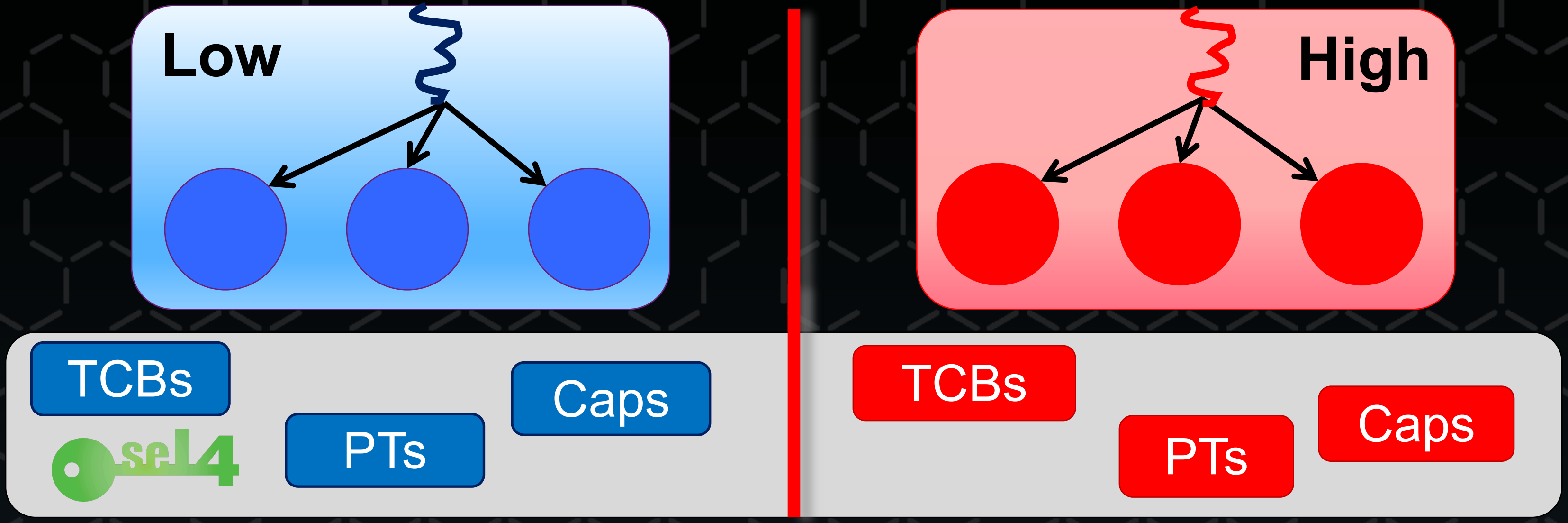
Verifying Isolation: Integrity



To prove: LOW doesn't have **write** *capabilities* to HIGH's objects
⇒ no action of LOW will modify HIGH state

- Specifically, *kernel does not modify HIGH on LOW's behalf!*
  - Event-based kernel operates on behalf of well-defined user thread
  - ***Prove**: kernel only allows write upon capability presentation*

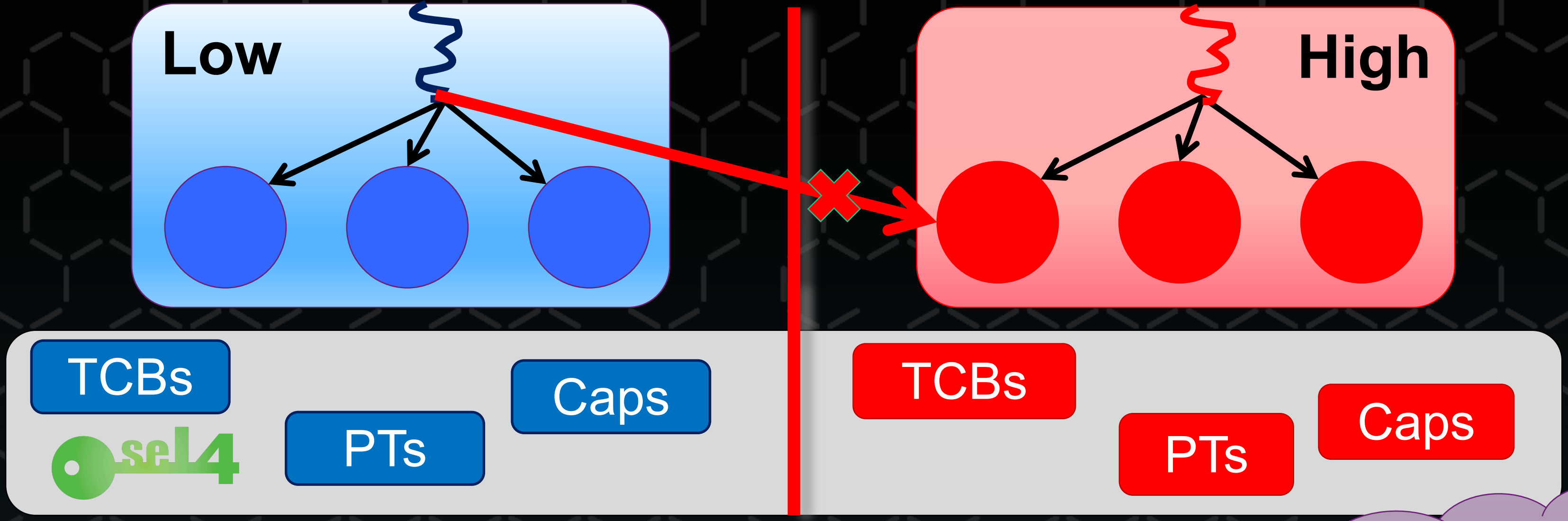
Verifying Isolation: Availability



To prove: HIGH can access promised resources when it wants to
⇒ *no action of LOW will lead to HIGH resources being denied*

- Strict separation of kernel resources, LOW cannot interfere with HIGH resources
- *Nothing to do: implied by other properties*

Verifying Isolation: Confidentiality



Violation not observable by High!

To prove: LOW doesn't have **read** *capabilities* to HIGH's objects
⇒ no action will reveal HIGH state to LOW

Non-interference proof :
- Evolution of ***LOW*** does not depend on ***HIGH*** state
- Also shows absence of covert *storage* channels

Verification Matrix

Feature	Core spec to C	C to binary	Security enforcem.	Mixed-criticality	Virtual machines	Multi-core
Arm 32	done	done	done	Q4’19	done	in progr.
Arm 64	unfunded	in progr.	unfunded	unfunded	unfunded	???
x64	done	no plans	no plans	easy?	no plans	???
RISC-V 64	Q4’19	Q3’19	unfunded	Q4’19	unfunded	???

- **Security**: CIA enforcement proofs
- **Mixed criticality**: advanced real-time support with temporal isolation; This will replace the mainline kernel once verified
- **Virtual machines**: verified use of hardware virtualisation support

L4 IPC Performance over 20 Years

Name	Year	Processor	MHz	Cycles	μs
Original	1993	i486	50	250	5.00
Original	1997	Pentium	160	121	0.75
L4/MIPS	1997	R4700	100	86	0.86
L4/Alpha	1997	21064	433	45	0.10
Hazelnut	2002	Pentium 4	1,400	2,000	1.38
Pistachio	2005	Itanium	1,500	36	0.02
OKL4	2007	XScale 255	400	151	0.64
NOVA	2010	i7 Bloomfield (32-bit)	2,660	288	0.11
seL4	2017	i7 Skylake (32-bit)	3,400	203	0.06
seL4	2017	I7 Skylake (64-bit)	3,400	138	0.04
seL4	2017	Cortex A53	1,200	225	0.19

Military-Grade Security

Cross-Domain Desktop Compositor

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

Manual Proof Effort

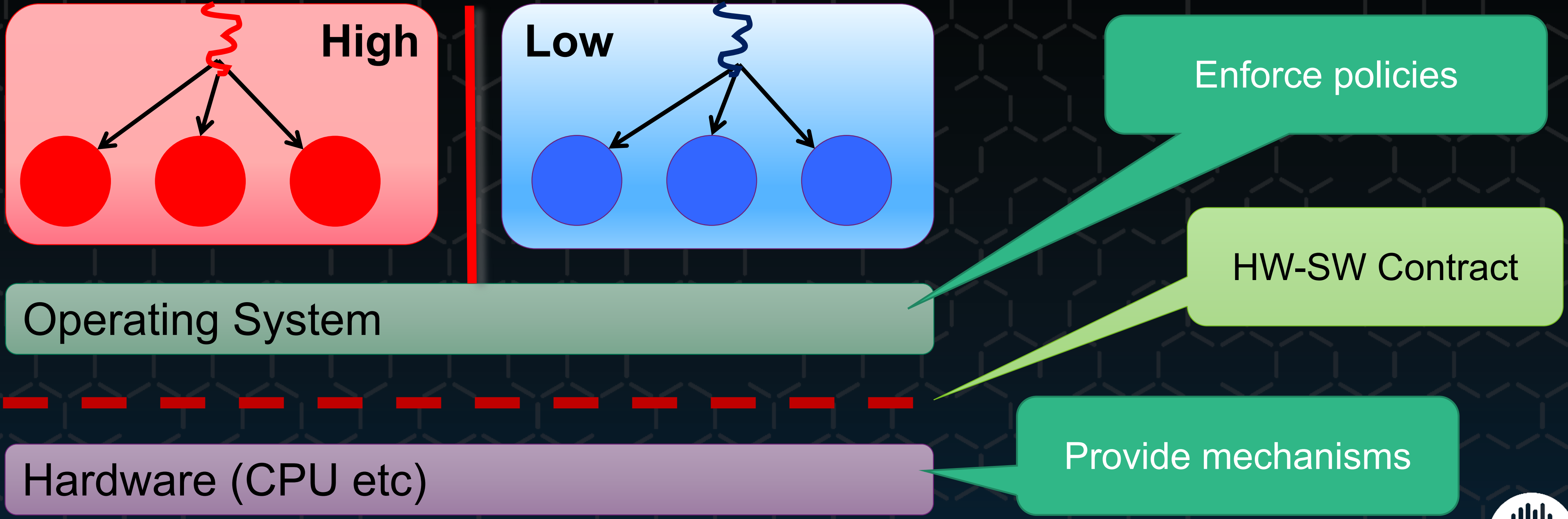
BilbyFS functions	Effort	Isabelle LoP	Cogent SLoC	Cost $/SLoC	LoP/ SLOC
isync()/ iget() library	9.25 pm	13,000	1,350	150	10
sync()-specific	3.75 pm	5,700	300	260	19
iget()-specific	1 pm	1,800	200	100	9
seL4	12 py	180,000	8,700 C	350	20

BilbyFS: 4,200 LoC Cogent

Security: A HW-SW Codesign Issue

Remember: Security Enforcement

Security enforcement must be **mandatory**, i.e. not dependent on application/user cooperation!



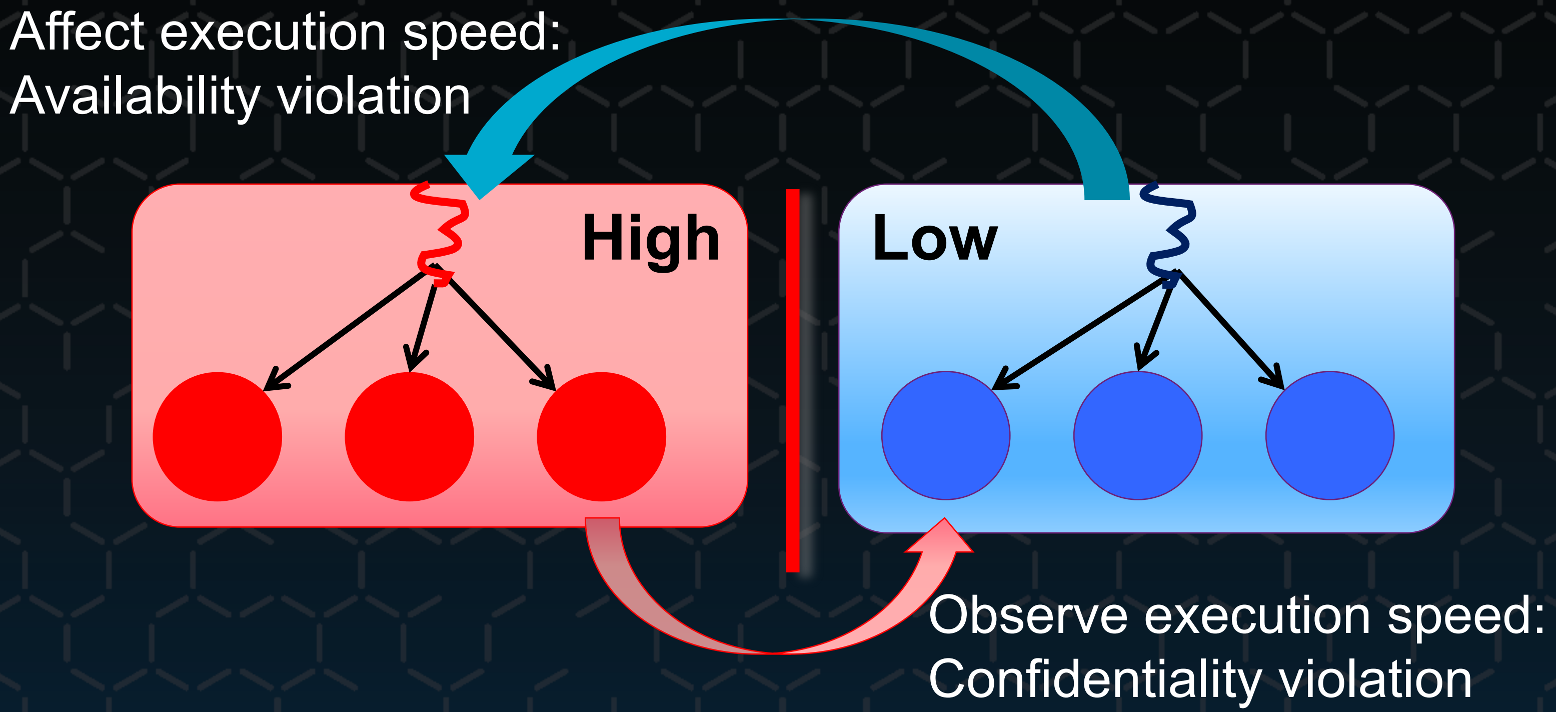
Hardware Cannot Do Security Alone!

- Security policies are high-level
  - Course-grain: “applications” are sets of cooperating processes
- Hardware mechanisms are fine-grain: instructions, pages, address spaces
  - Much semantics lost in mapping to hardware level
- Security policies are complex: “Can A talk to B?” is too simple
  - maybe one-way communication is allowed
  - maybe communication is allowed under certain conditions
  - maybe low-bandwidth leakage doesn’t matter
  - maybe secrets only matter for a short time
  - maybe only subset of {confidentiality, integrity, availability} is important

Why the ISA is an Insufficient Contract

- The ISA is a purely operational contract
  - Sufficient for ensuring functional correctness
  - Insufficient for ensuring confidentiality or availability

The ISA intentionally abstracts time away


Affect execution speed:
Availability violation
High
Low
Observe execution speed:
Confidentiality violation