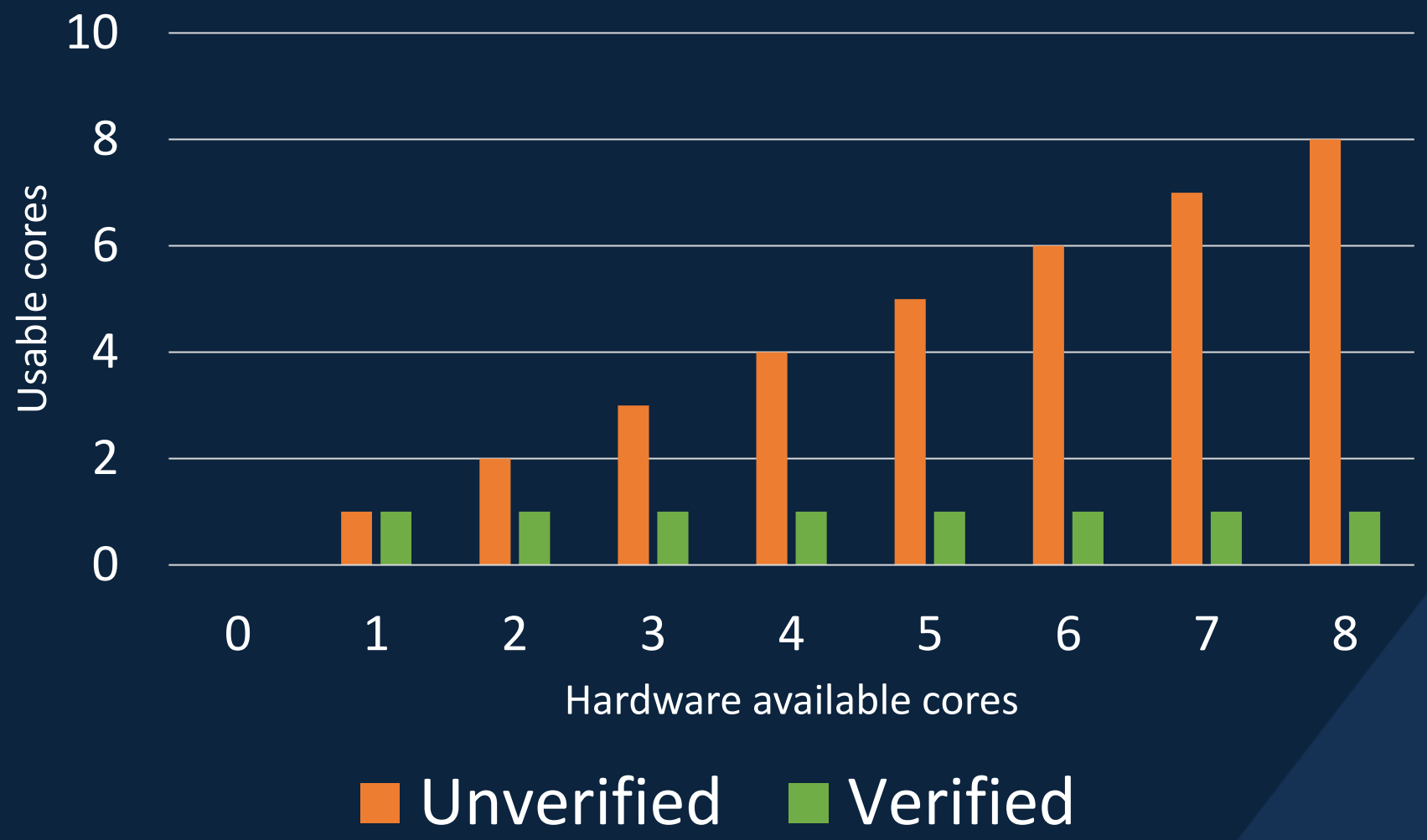
Multiprocessing on seL4 with verified kernels

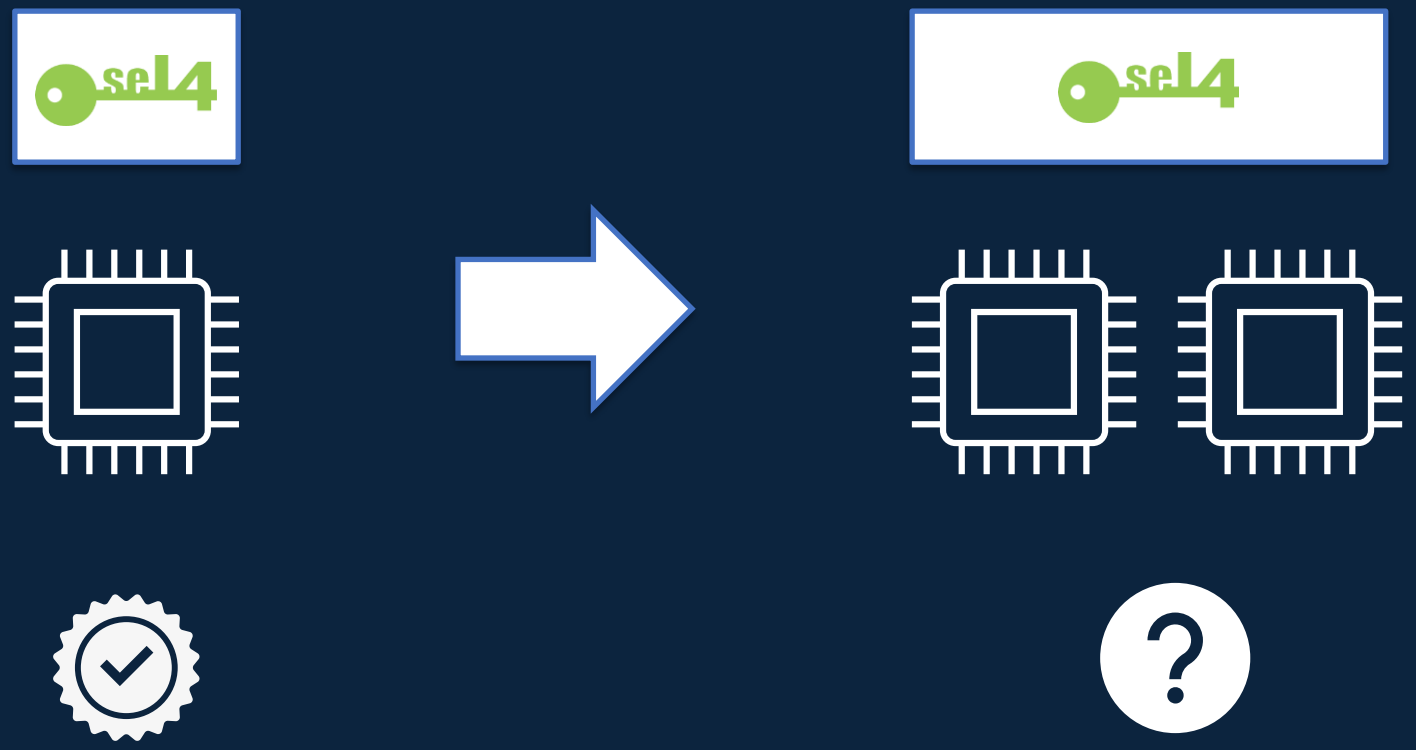
Kent McLeod | seL4 Summit 2022 | Munich, Germany

KRY10

Usable CPU count by kernel configuration



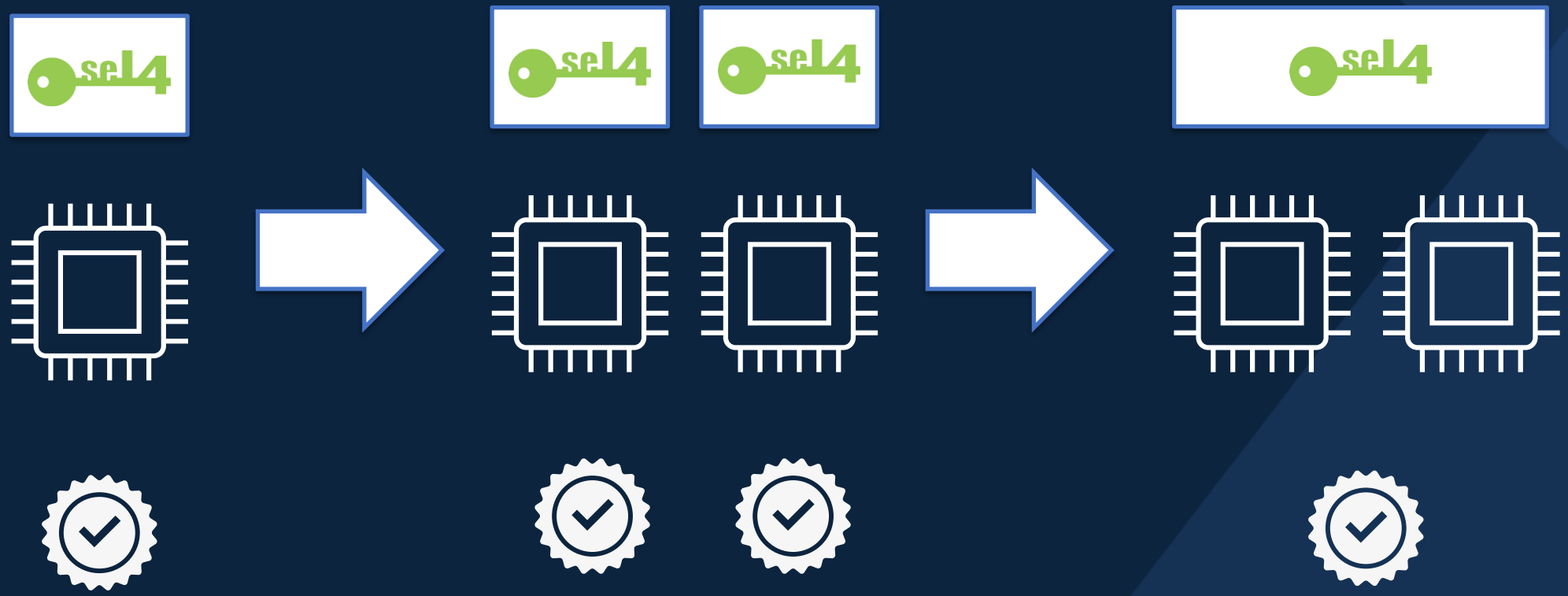
SMP seL4 configurations



Unicore

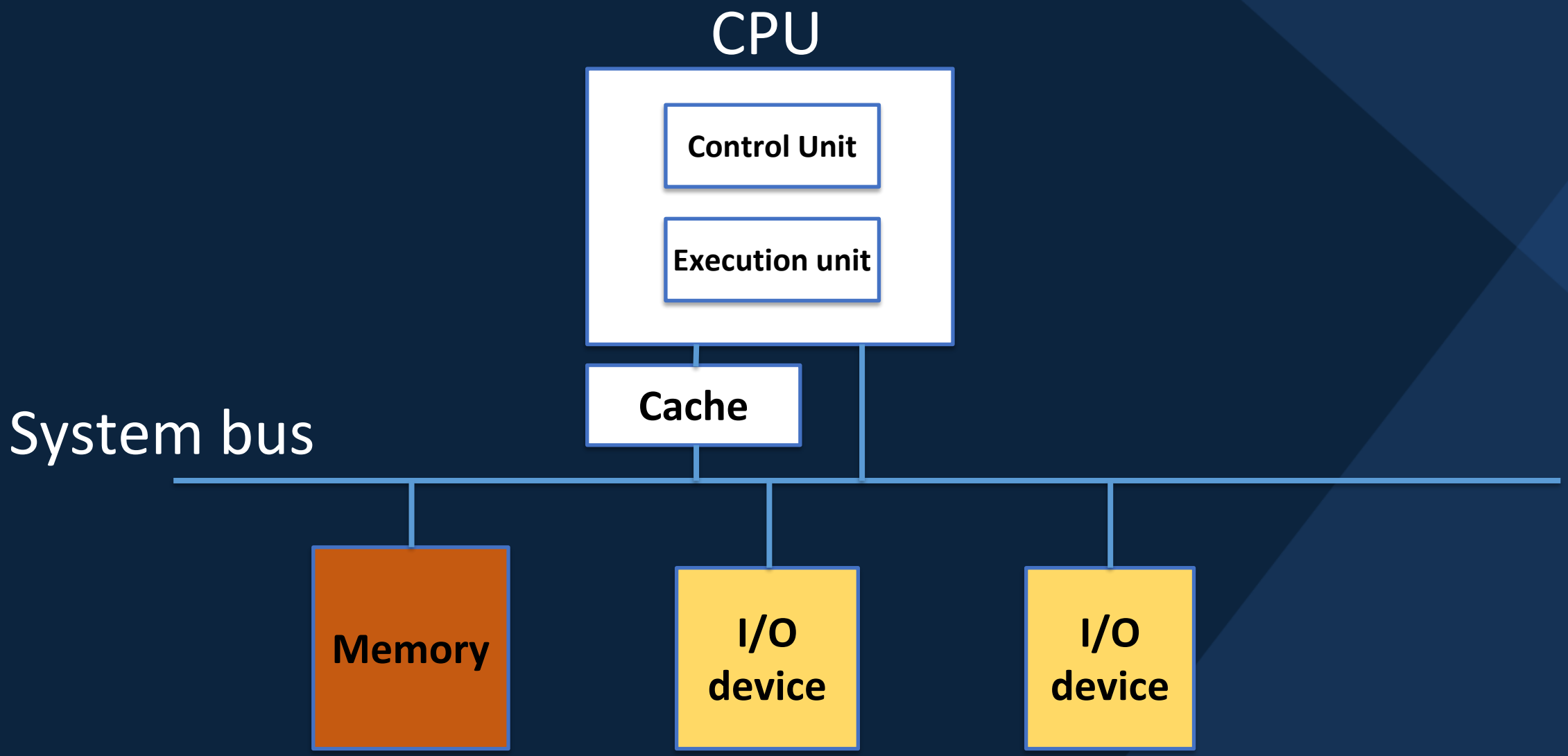
SMP

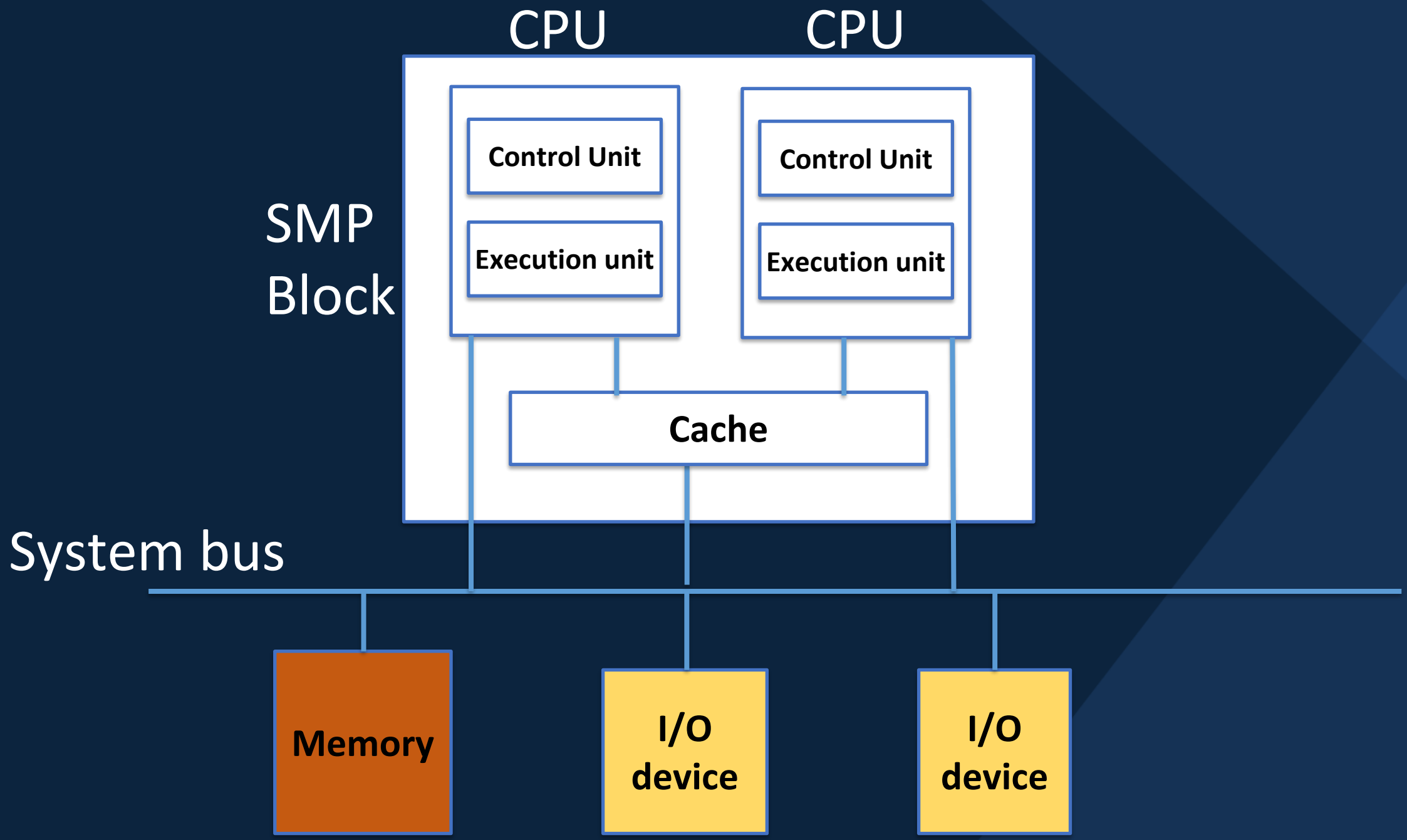
(Re)Introducing: Partitioned multikernel



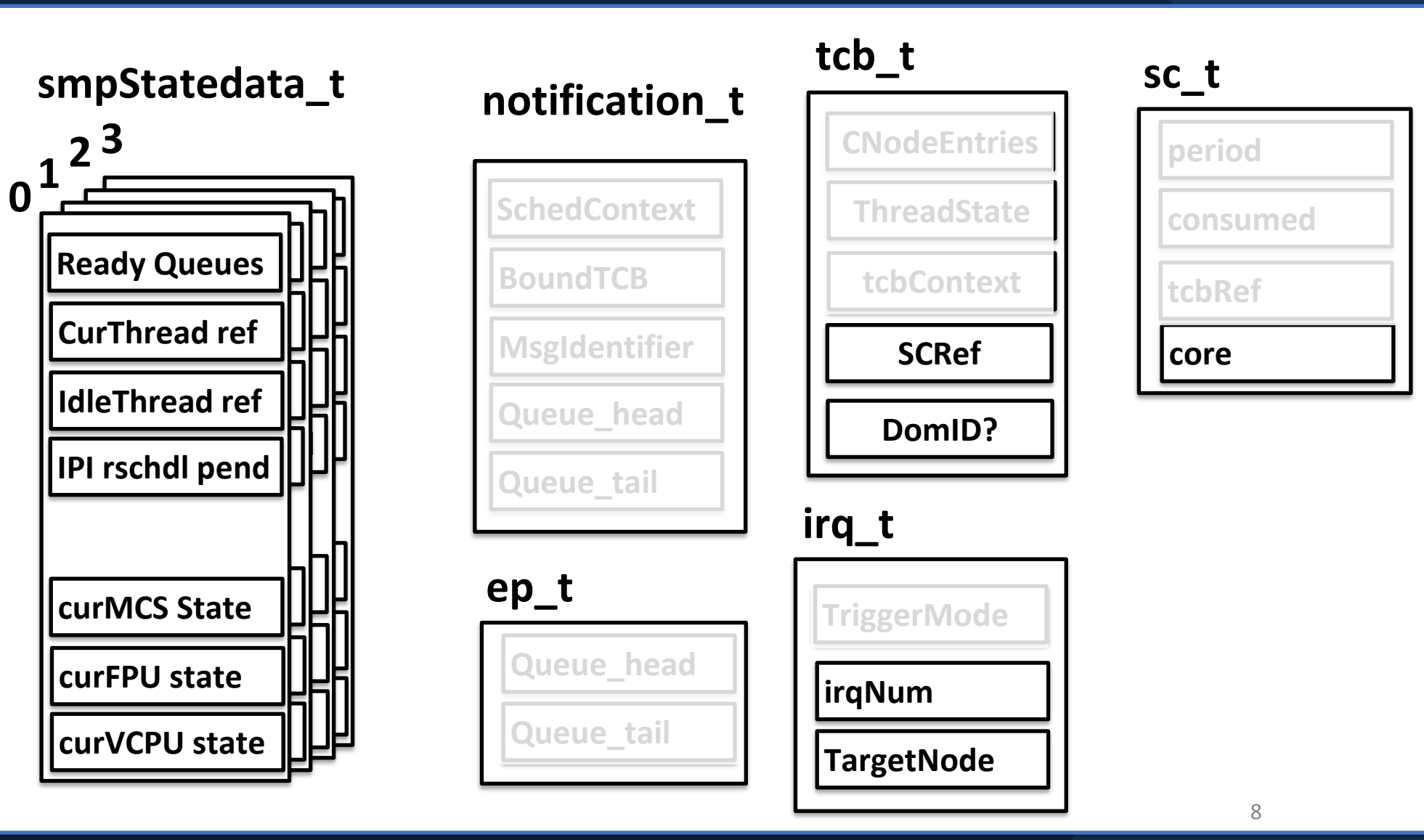
What are the trade-offs?

	Multikernel	SMP
Kernel State	Partitioned	Shared
Concurrency in Kernel	No - better verification	Yes - hard to verify
Cross-core communications	Implemented at userlevel	Implemented by kernel

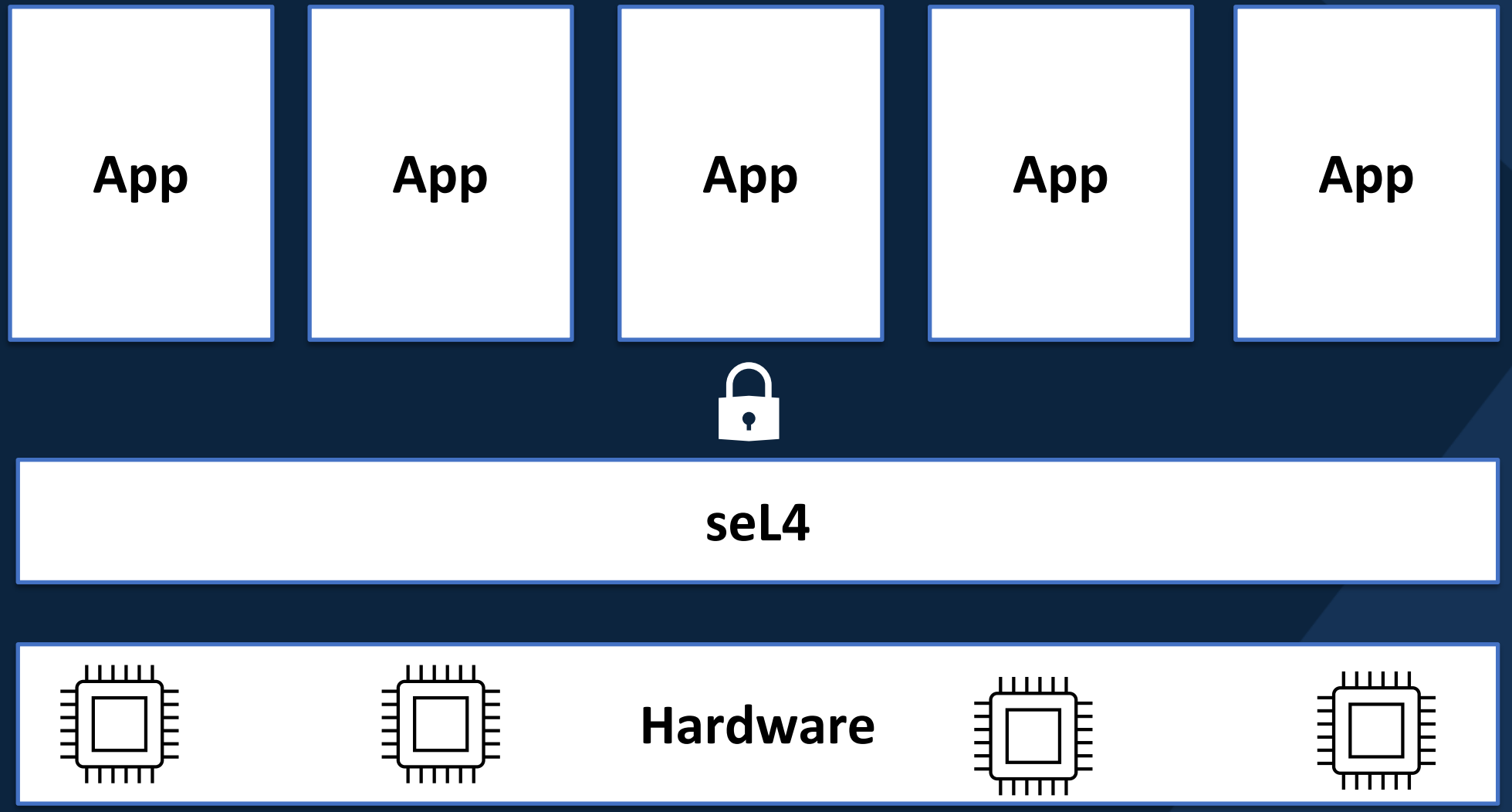

CPU
Control Unit
Execution unit
Cache
System bus
Memory
I/O device
I/O device


CPU
CPU
SMP
Block
Control Unit
Execution unit
Control Unit
Execution unit
Cache
System bus
Memory
I/O
device
I/O
device

seL4 SMP kernel (Big lock)

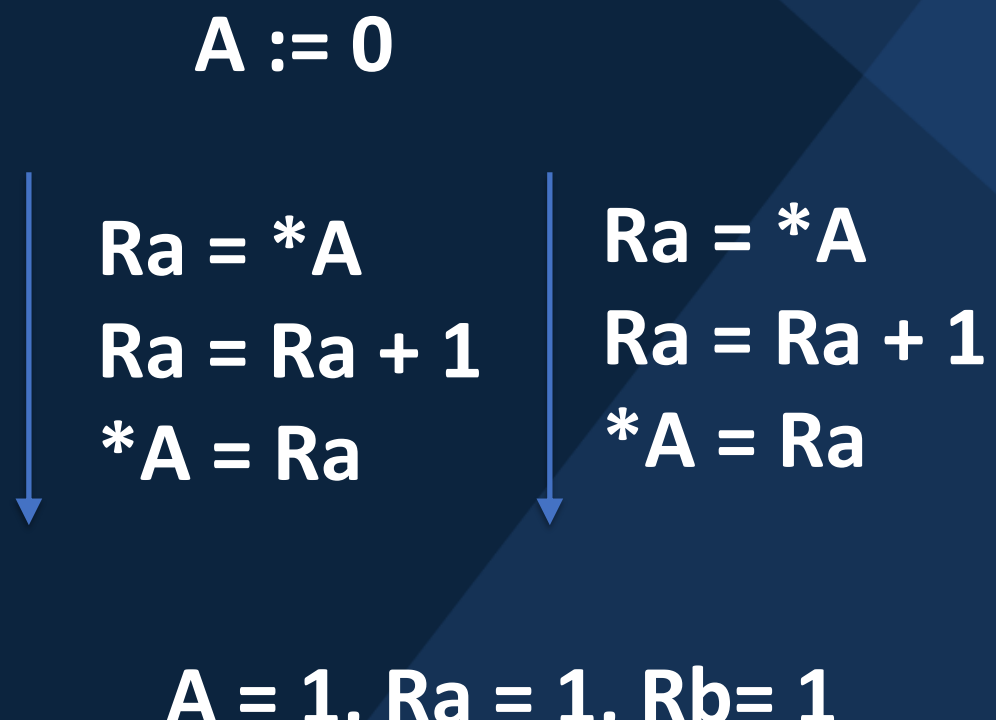


Multicore (SMP)



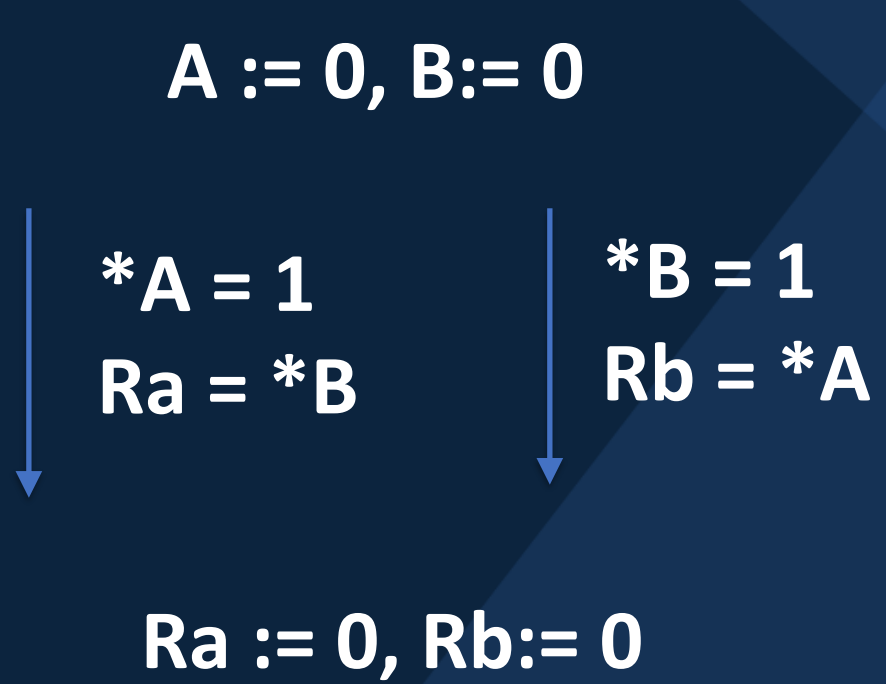
Concurrency in the kernel

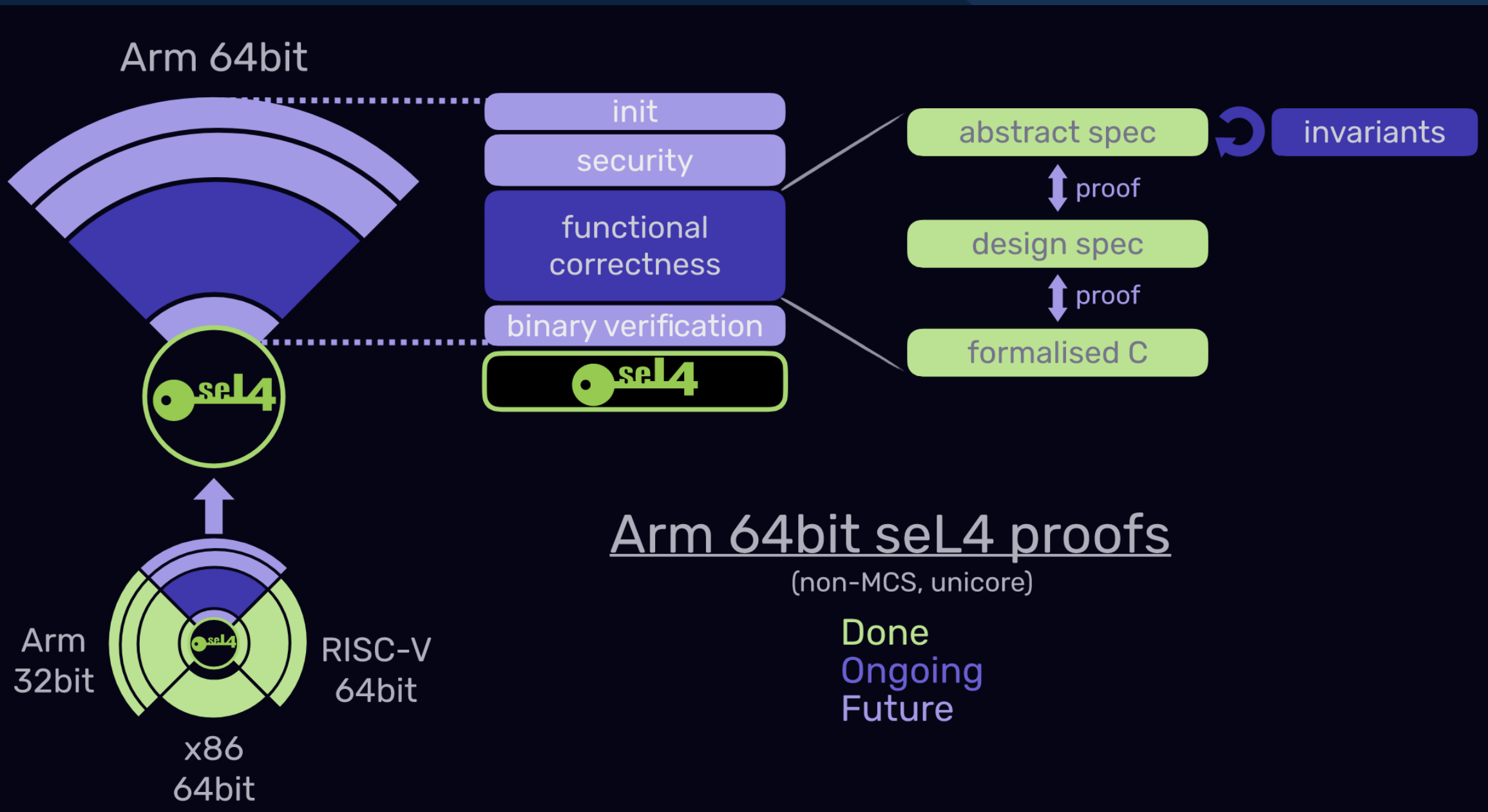
- seL4 proofs model sequential execution
- proof has to cover all conceptual scenarios that can arise from concurrent execution
- Want to introduce mutual exclusion primitives and prove them correct
- Kernel entry code runs outside of the lock



Weak memory models

- Modern ISAs don’t have strong memory models
- Can observe behaviours that aren’t sequentially consistent
- Concurrent code that works on same core may not work on separate cores




Arm 64bit
init
security
functional correctness
binary verification
seL4
abstract spec
invariants
proof
design spec
proof
formalised C
Arm 32bit
RISC-V 64bit
x86 64bit
Arm 64bit seL4 proofs
(non-MCS, unicore)
Done
Ongoing
Future

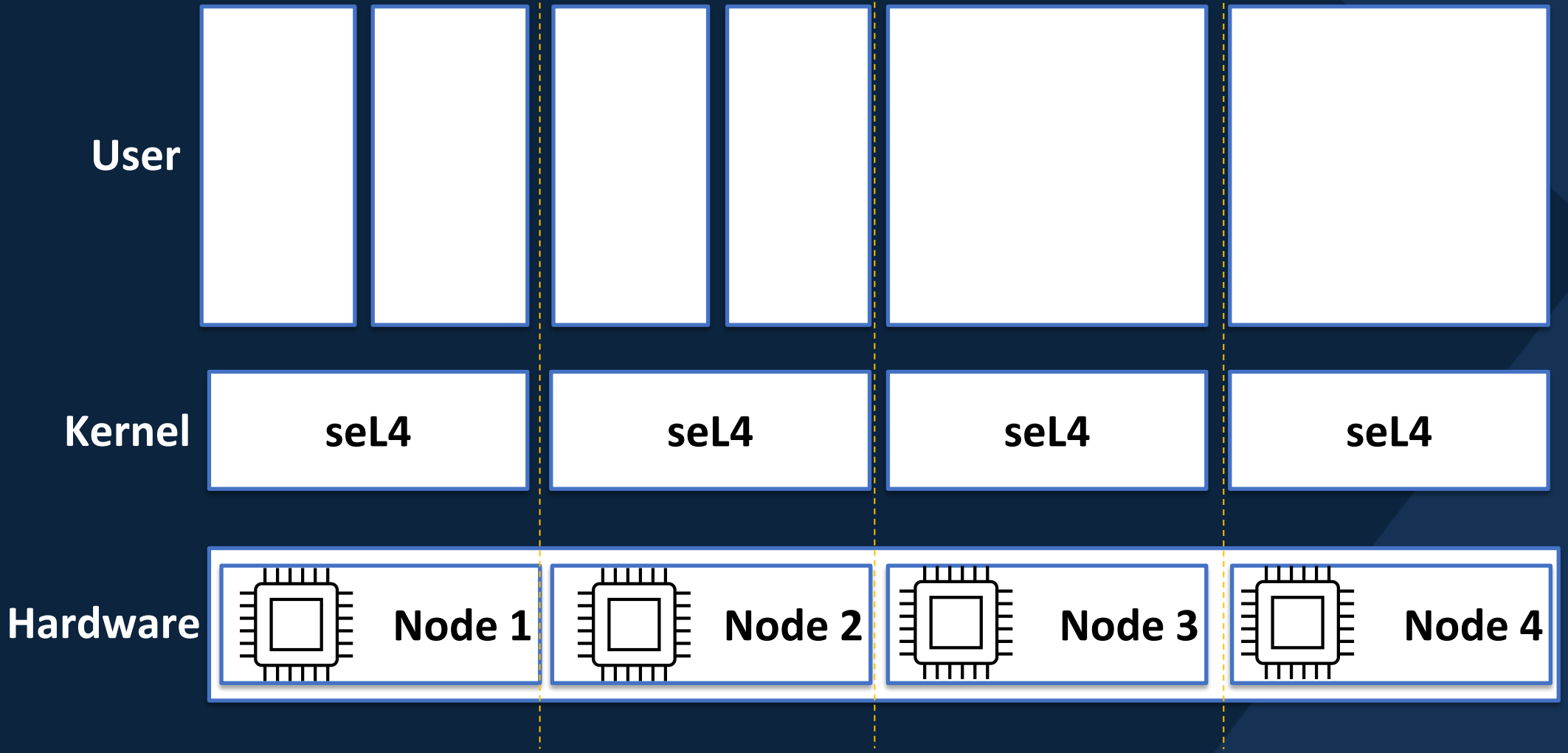
Check-point

- SMP kernel has shared state
- Concurrency in the kernel
- Big kernel lock:
  - Simplifies verification, but not by a lot initially
  - Adds locking overhead to all kernel operations
- Non-negligible code changes for implementing SMP design

Multiprocessing on seL4 with verified kernels?

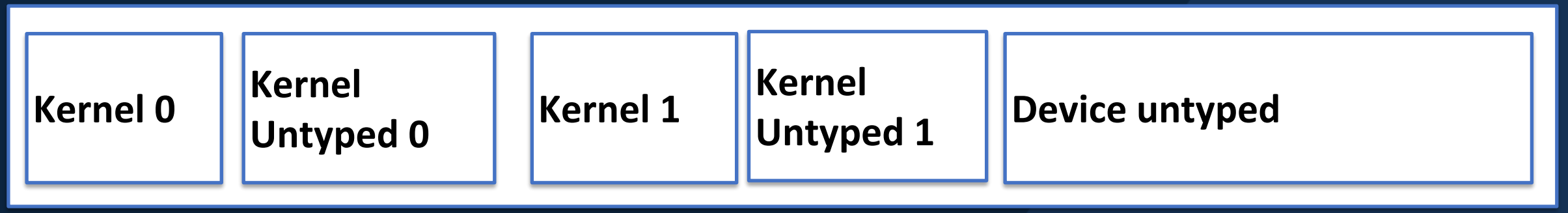
1. No concurrency in the kernel
2. No changes to verified C code
3. Still need to be able to build useful systems
4. Ideally possible to migrate to SMP

Partitioned kernels



Memory Partitioned kernel

- Kernel memory management via untypeds

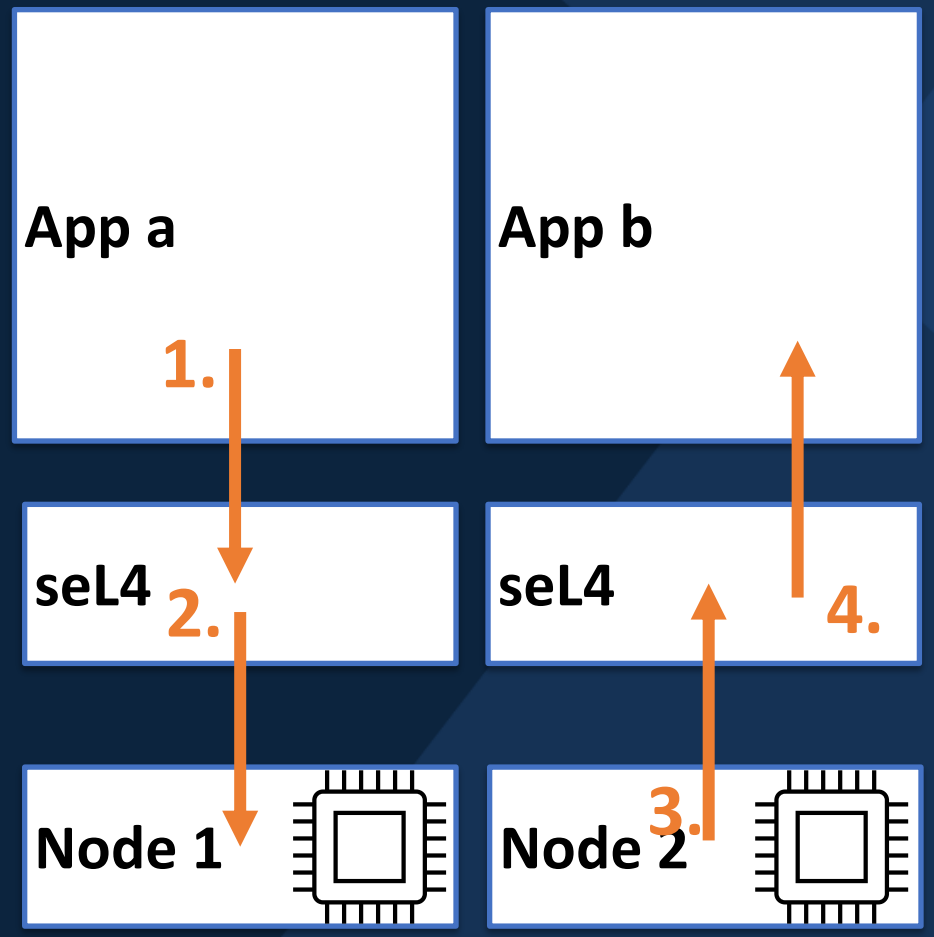


Inter-process Communication

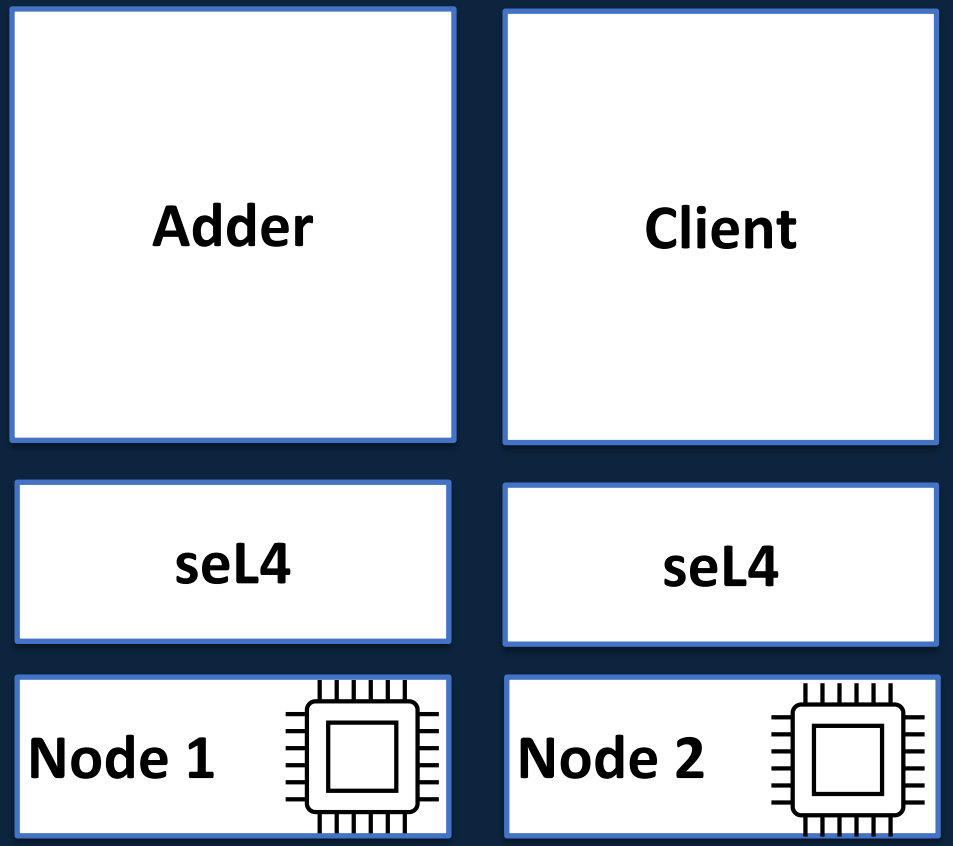
	Unicore and SMP	Multikernel
Shared memory	Duplicate frame mappings	Duplicate frame mappings of device untyped
Signalling	Notifications	Notifications or IPIs
PPC	Endpoints	Endpoints or implemented at userlevel

Signalling in multikernels

1. If the target is on a different core, invoke an IPI capability
2. Generate an IPI
3. Other core is interrupted and delivers IRQ to bound notification
4. Remote thread receives notification and acks IPI IRQ



Example: a multikernel system with CAmkES*



* Or other component architectures

Example: CAmkES ADL

```
component Adder {
    control;
    dataport Buf d;
    consumes DataAvailable irq;
    emits DataAvailable irq2;
}

component Client {
    control;
    dataport Buf d;
    consumes DataAvailable irq;
    emits DataAvailable irq2;
}
```

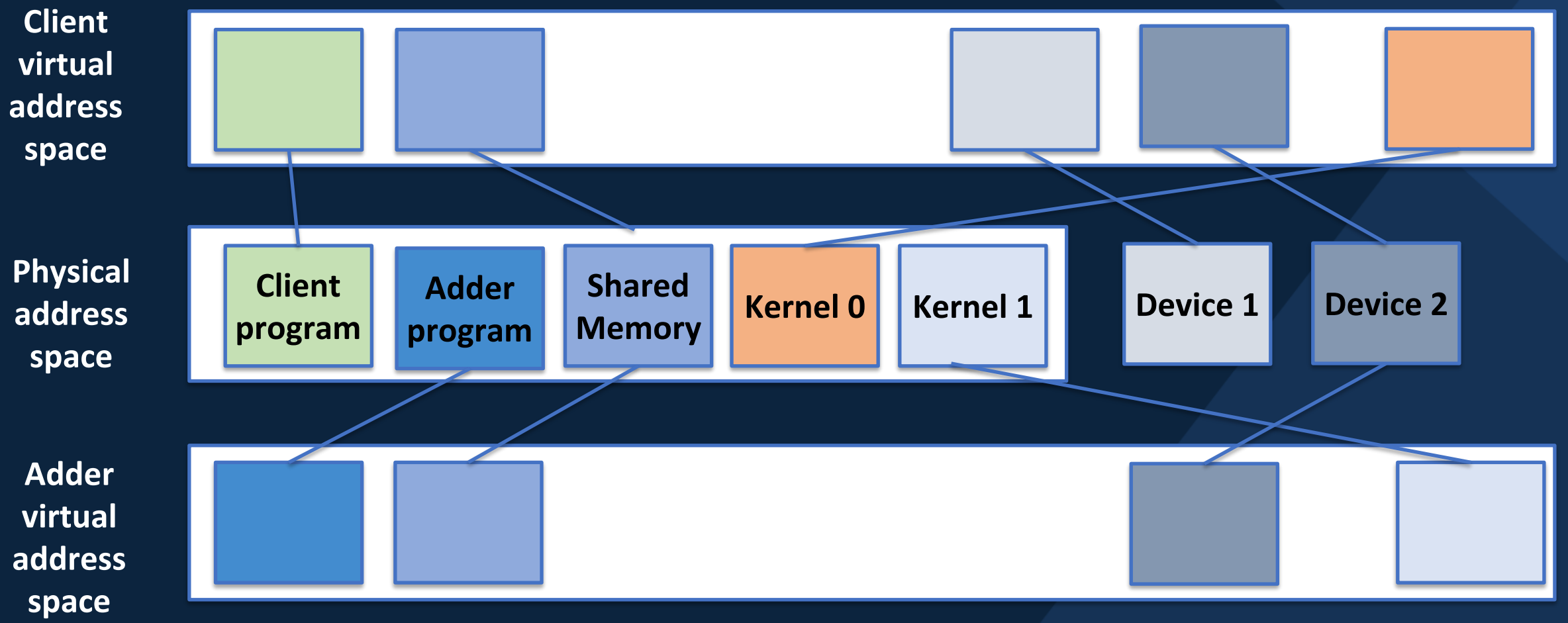
```
assembly {
    composition {
        component Adder adder;
        component Client client;

        connection seL4SharedData s(from adder.d, to client.d);
        connection seL4NotificationNative irq(from client.irq2, to adder.irq);
        connection seL4NotificationNative irq2(from adder.irq2, to client.irq);
    }

    configuration {
        adder.node = "node1";
        client.node = "node0";
    }
}
```

Example: CAmkES ADL

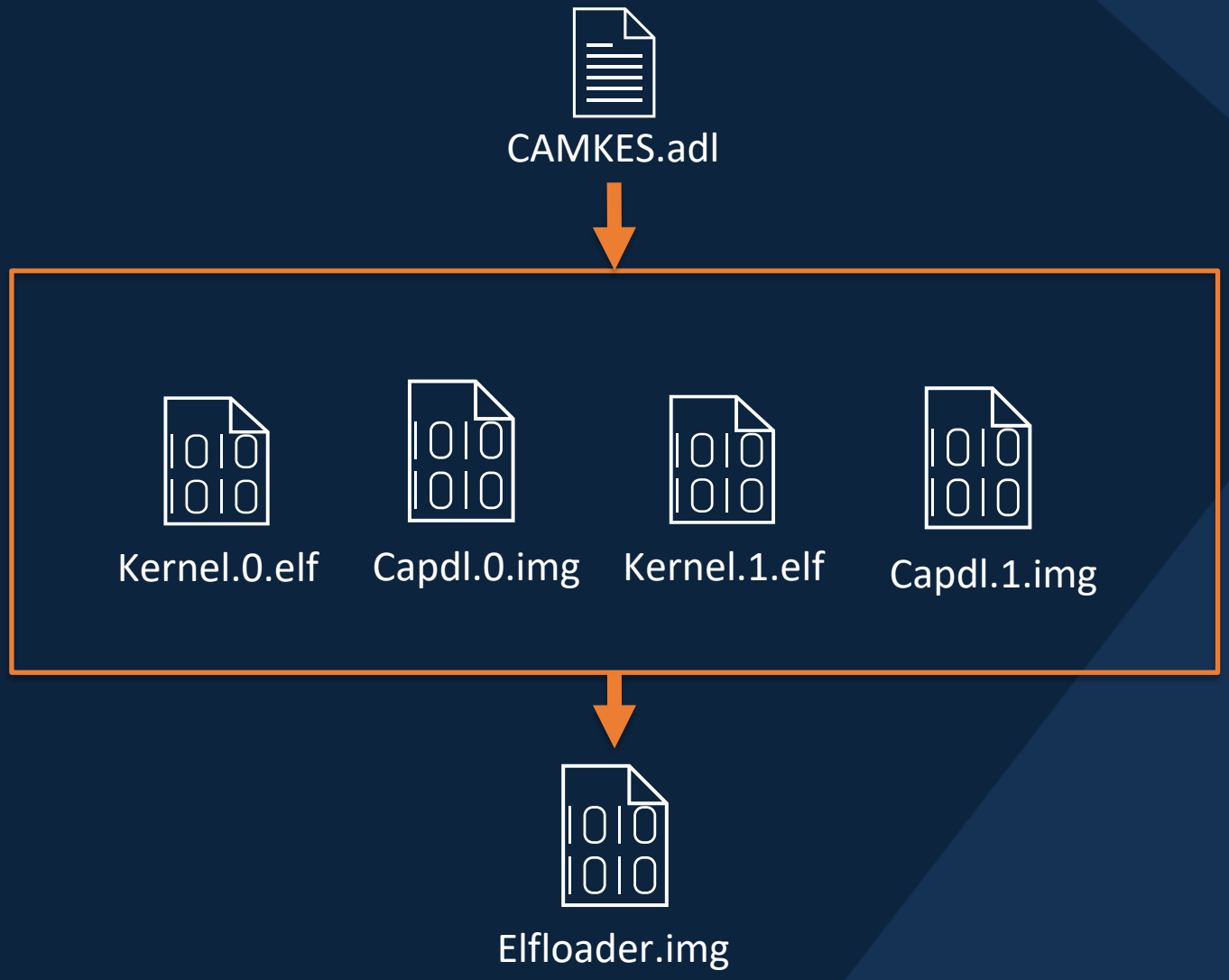
```
assembly {
    composition {
        component Kernel node0;
    }
    configuration {
        node0.node_id = 0;
        node0.memory = [{
            "node": "/memory@40000000",
            "reg": [
                {"start": 0x40000000, "size": 0x10000000 },
            ]}];
        node0.reserved = [
                {"start": 0x50000000, "size": 0x10000000 },
            ];
        node0.shared_pool = [
            {"start": 0x60000000, "size": 0x10000000 },
        ];
    }
}
```

```
assembly {
    composition {
        component Kernel node1;
    }
    configuration {
        node1.node_id = 1;
        node1.memory = [{
            "node": "/memory@40000000",
            "reg": [
                {"start": 0x50000000, "size": 0x10000000 },
            ]}];
        node1.reserved = [
                {"start": 0x40000000, "size": 0x10000000 },
            ];
        node1.shared_pool = [
            {"start": 0x60000000, "size": 0x10000000 },
        ];
    }
}
```

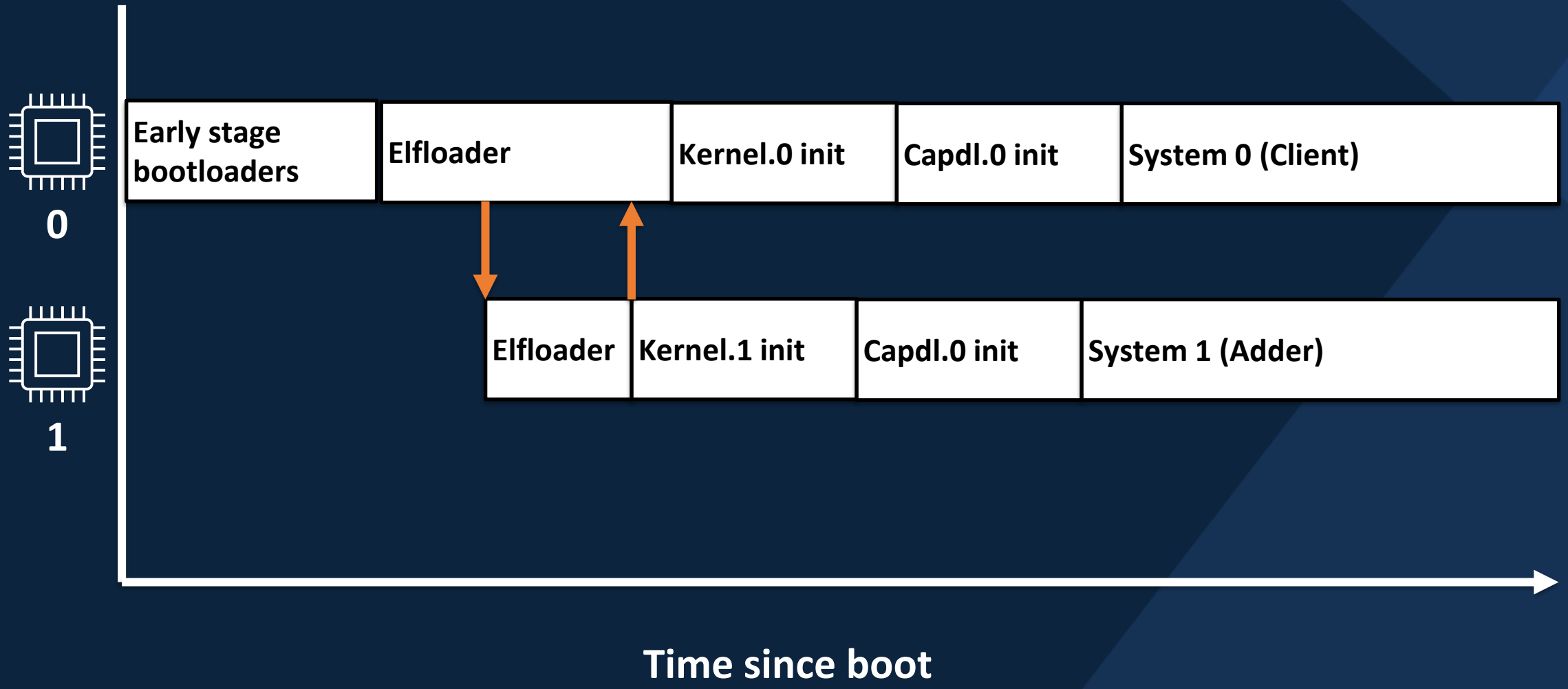
Example: Memory Layout



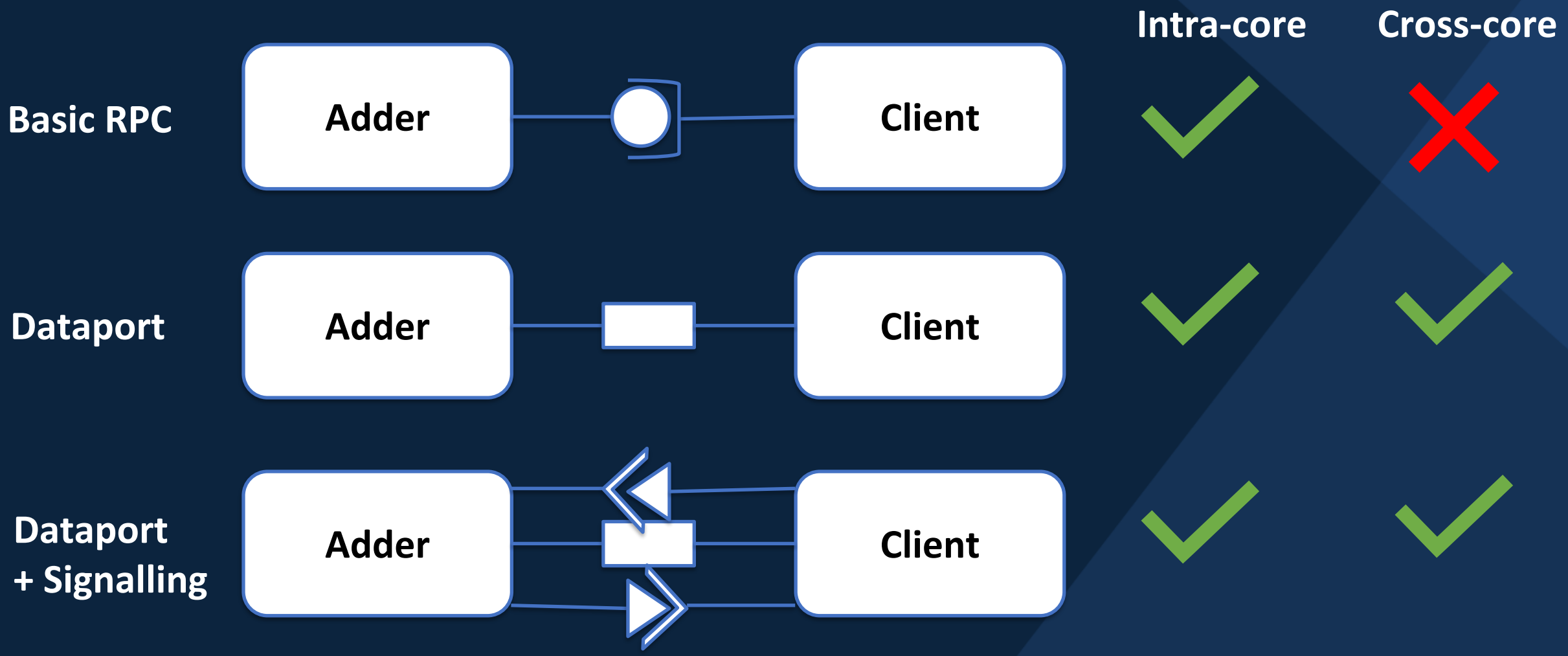
Example: CAmkES build



Example: Boot process



IPC with CAmkES



Summary of work so far

- Elfloader changed to support multiple kernel setups
- Kernel compilation tooling tweaked to support partitioned memory and IRQs
- IPI capability
- CAmkES extensions
  - Multiple kernels and capdl-loader-apps
  - Multikernel aware connectors
  - Future work to support SMP apps on multikernel

Steps towards verified SMP

1. Build out concurrent verifiection framework for multikernel
2. Unified domain scheduler
3. Unified kernel address space
4. Scalable notifications
5. Other forms of kernel resource sharing
6. General SMP

Follow-up steps

- SMP-like user apps
- Scalable cross-core notifications
- Investigating impact of replicated data on shared caches
- Transparent cross-core seL4RPCCall CAmkES connectors
- Finish off multi-vm multicore example

Overall summary

- Current verified configurations only single core
- SMP verified configurations still a long way away
- Should be possible to support verified multikernel configurations significantly sooner
- Restrictions from multikernel design may be less significant for static component architectures
- Multikernel verification projects also work towards eventual SMP verification

Discussion