NICTA

Formally Verified System Initialisation

Andrew Boyton
June Andronick, Callum Bannister, Matthew Fernandez, Xin Gao, David Greenaway, Gerwin Klein, Corey Lewis and Thomas Sewell

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

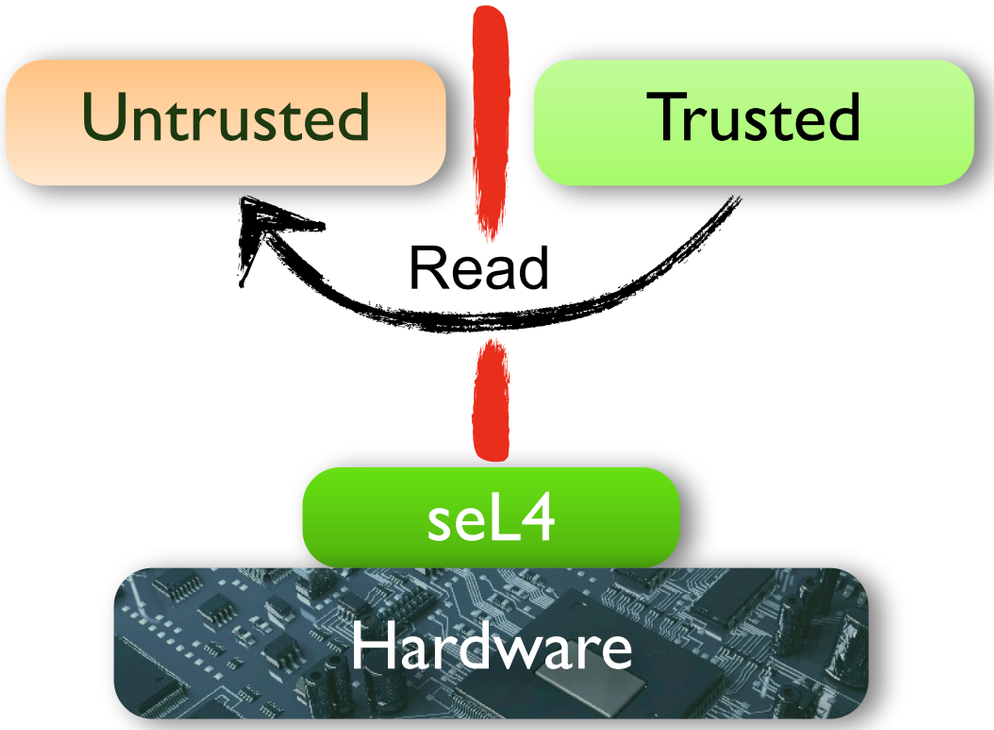
NICTA Funding and Supporting Members and Partners

Australian National University
UNSW THE UNIVERSITY OF NEW SOUTH WALES
NSW GOVERNMENT Trade & Investment
ACT GOVERNMENT
State Government Victoria
THE UNIVERSITY OF MELBOURNE
THE UNIVERSITY OF SYDNEY
Queensland Government
Griffith UNIVERSITY
QUT Queensland University of Technology
THE UNIVERSITY OF QUEENSLAND AUSTRALIA

Motivation

NICTA

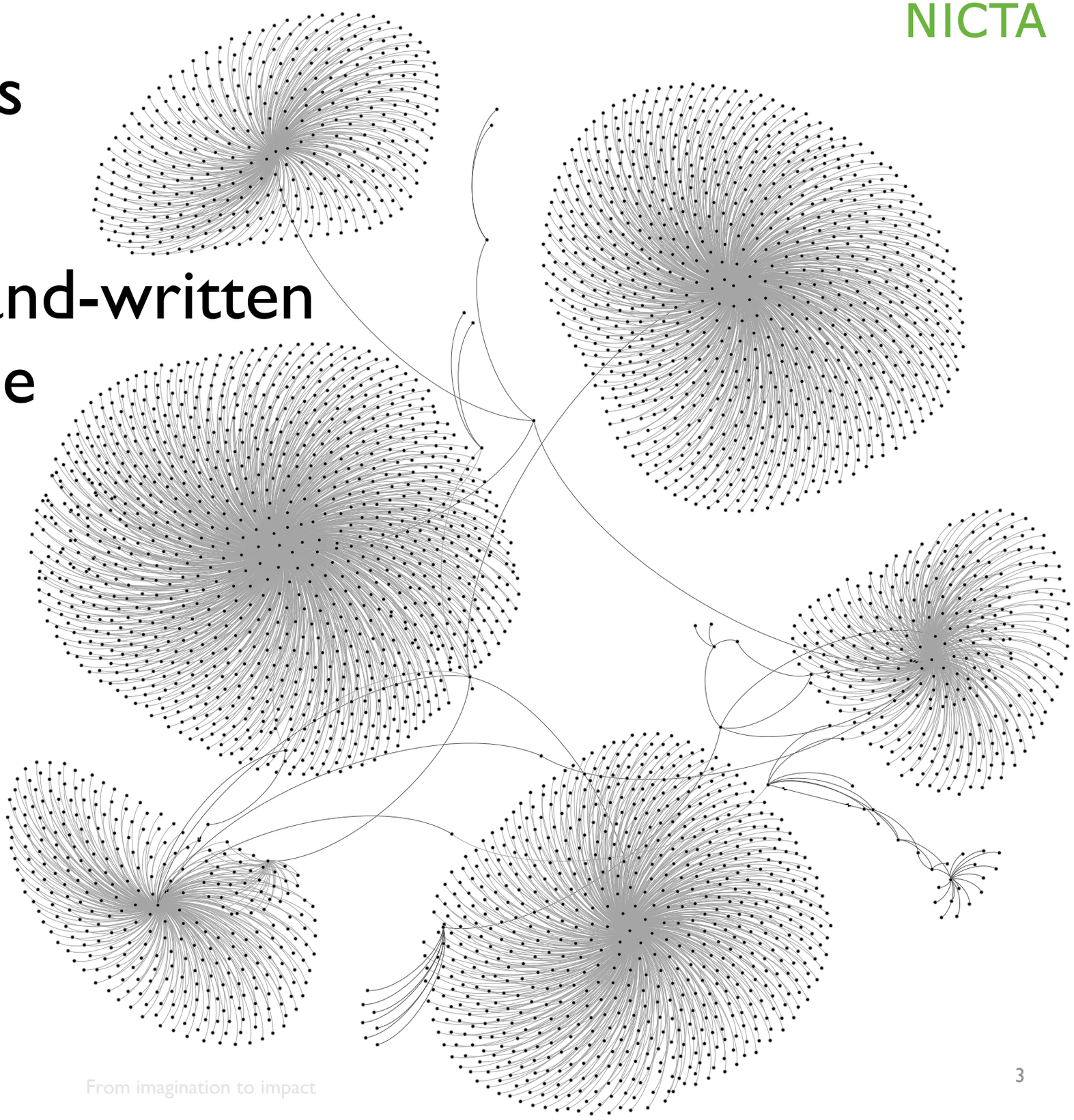
- **Create high-assurance systems**
  - Pacemaker
  - Unmanned aerial vehicle (UAV)
  - Secure access terminal



- **Capability-based system**
  - seL4 microkernel
- **Two problems:**
  - Does our capability system correctly enforce the capabilities?
  - ***How do we set up a system with the desired capabilities?***

Initialisation code can get big

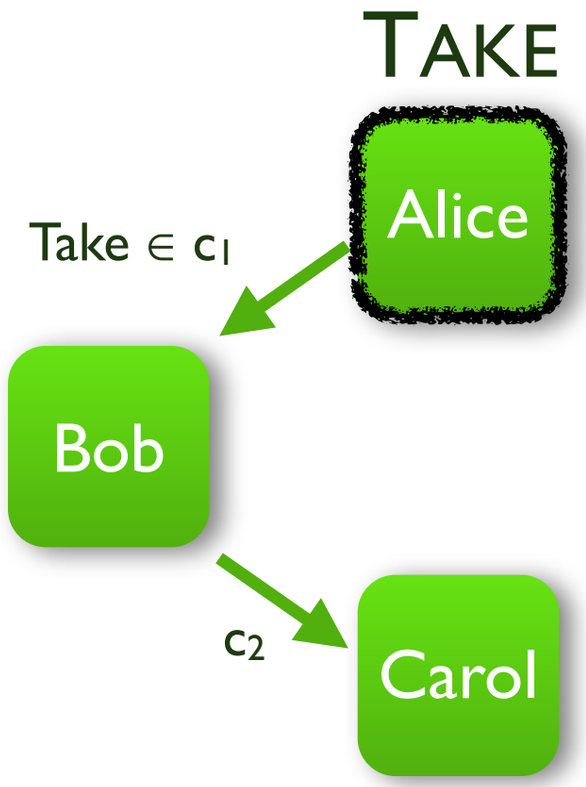
- 3,600 capabilities
- 3,500 objects
- 2,500 lines of hand-written initialisation code



Capability systems

NICTA

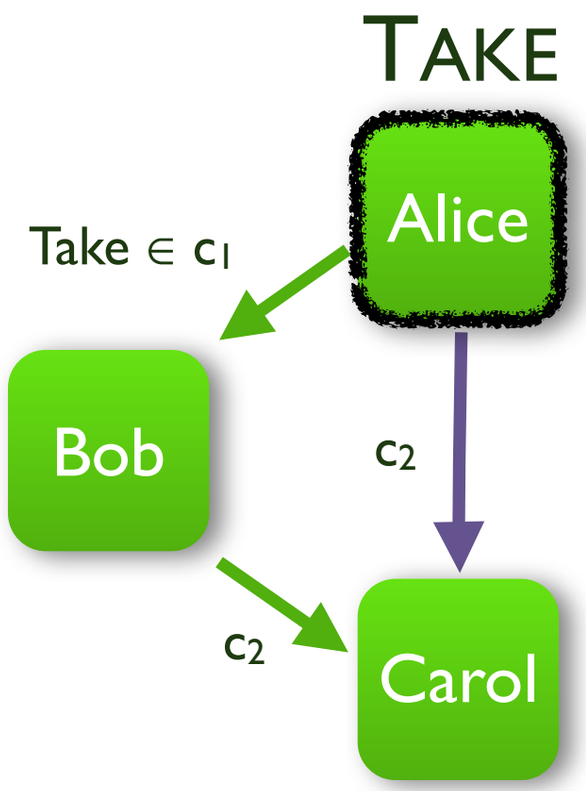
- Take-grant model



Capability systems

NICTA

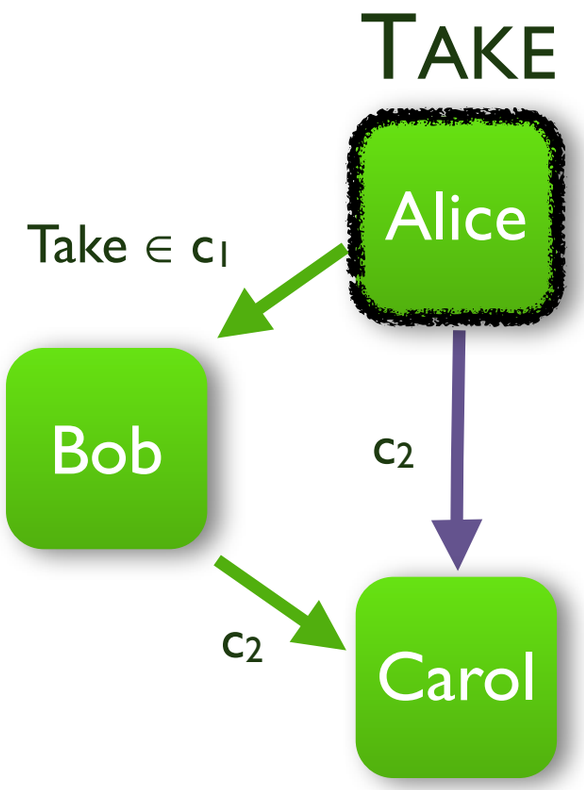
- Take-grant model

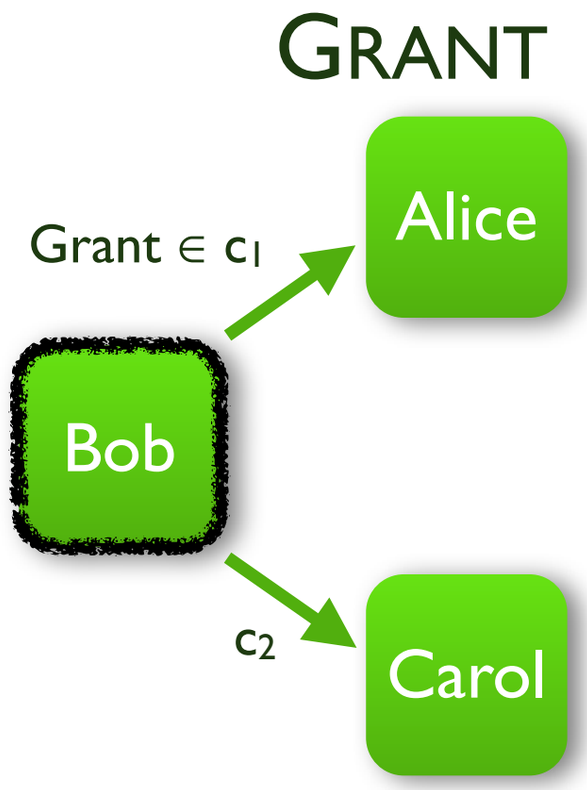


Capability systems

NICTA

- Take-grant model

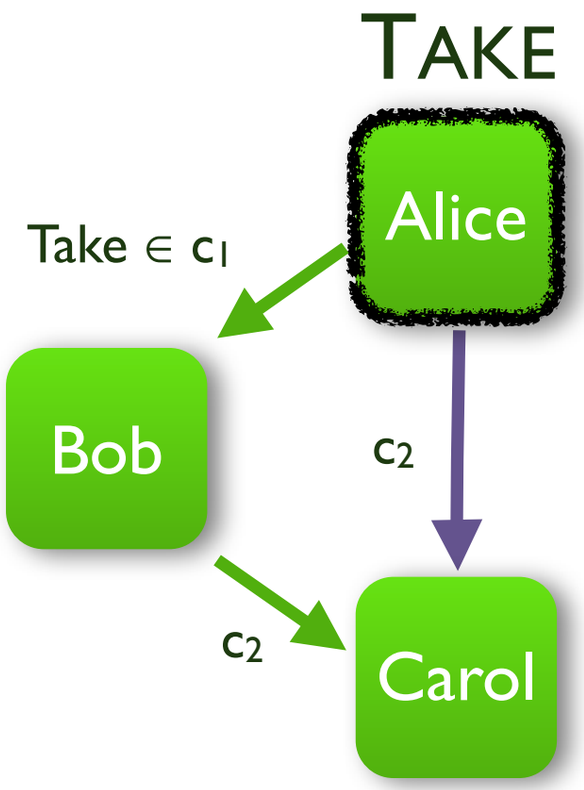


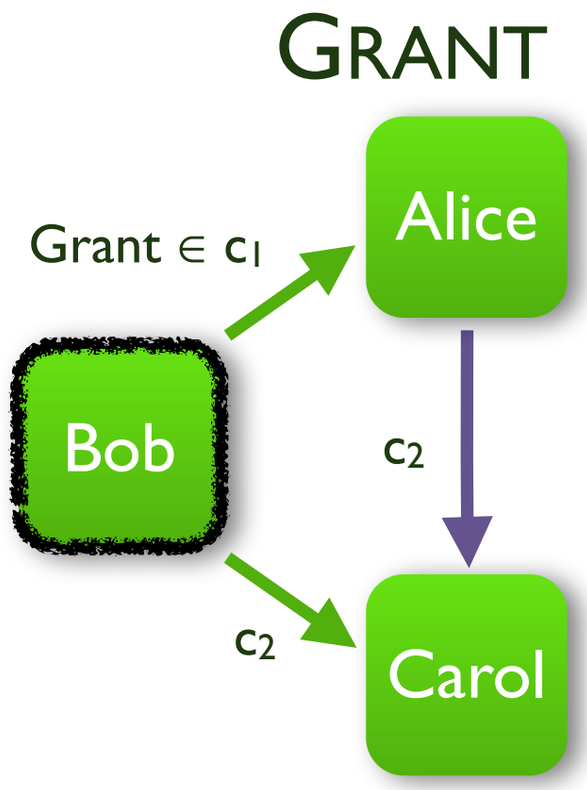


Capability systems

NICTA

- Take-grant model





seL4 microkernel

NICTA

Access Control Policy ⟺ Information Flow

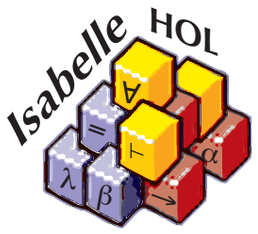
Functional Specification

High-Performance C Implementation

Binary

seL4 microkernel

NICTA



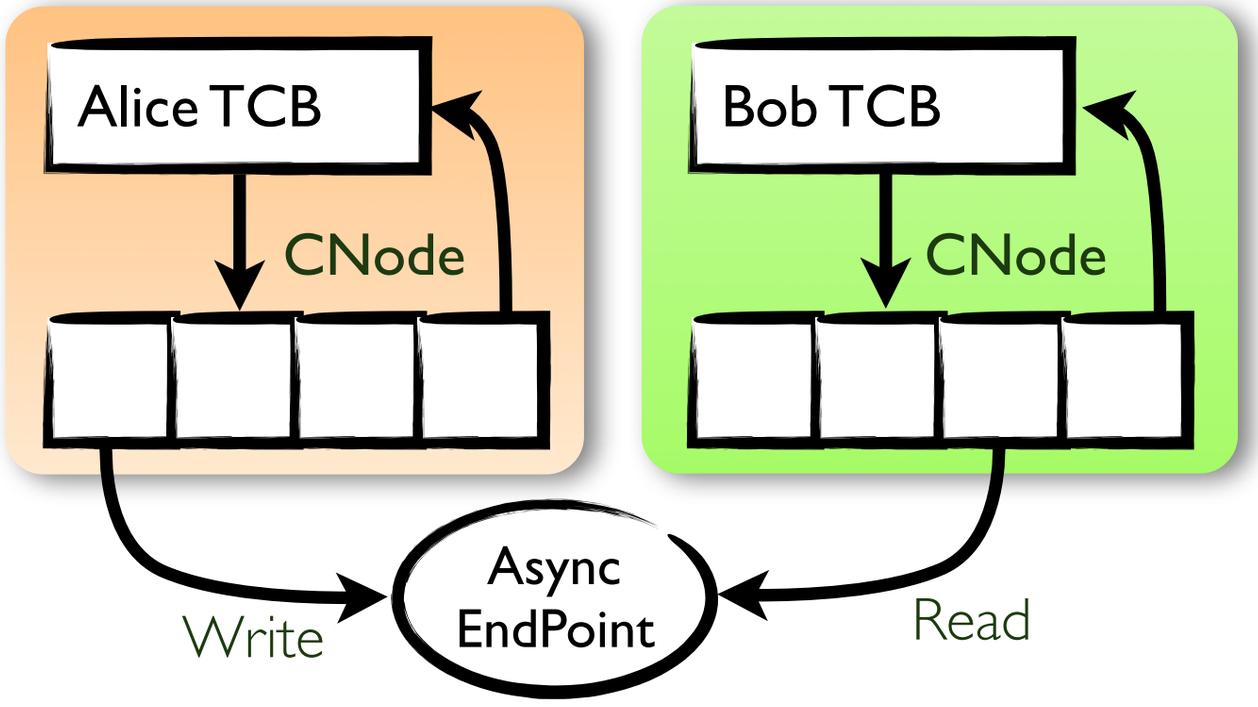
Access Control Policy ⟺ Information Flow

CapDL Model

Functional Specification

High-Performance C Implementation

Binary



seL4 microkernel

NICTA

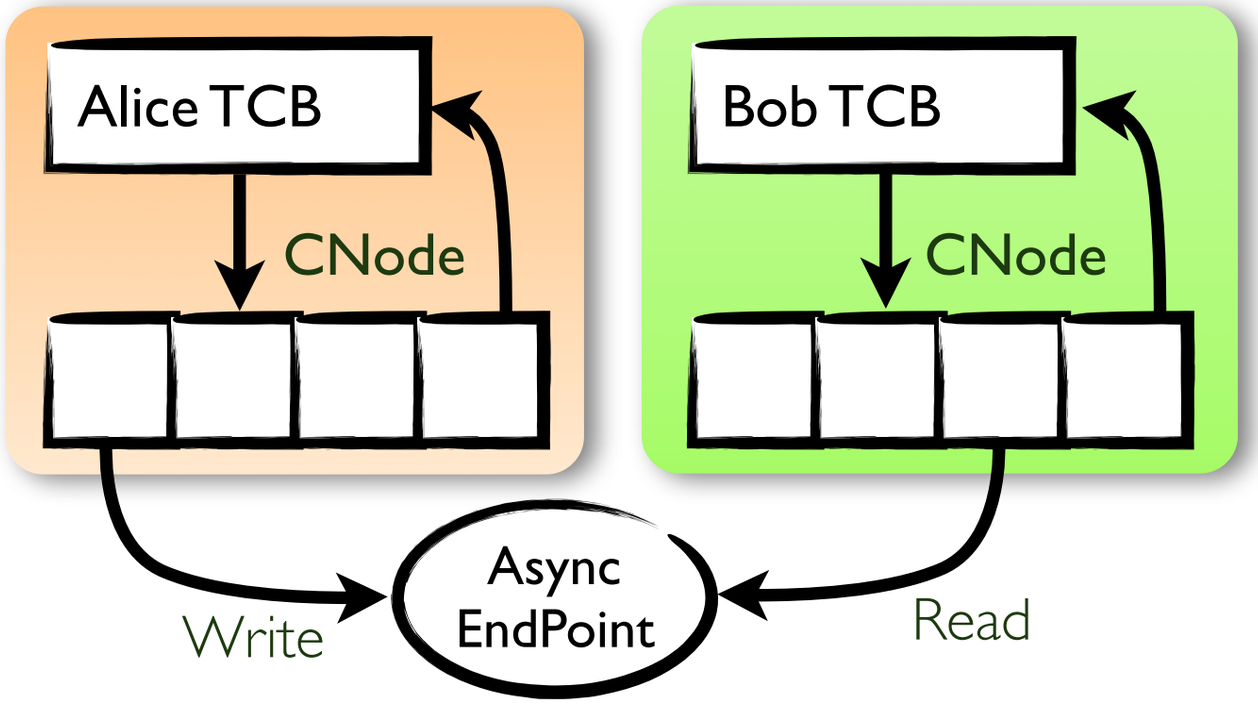
Access Control Policy ⟺ Information Flow

CapDL Model

Functional Specification

High-Performance C Implementation

Binary



seL4 microkernel

NICTA

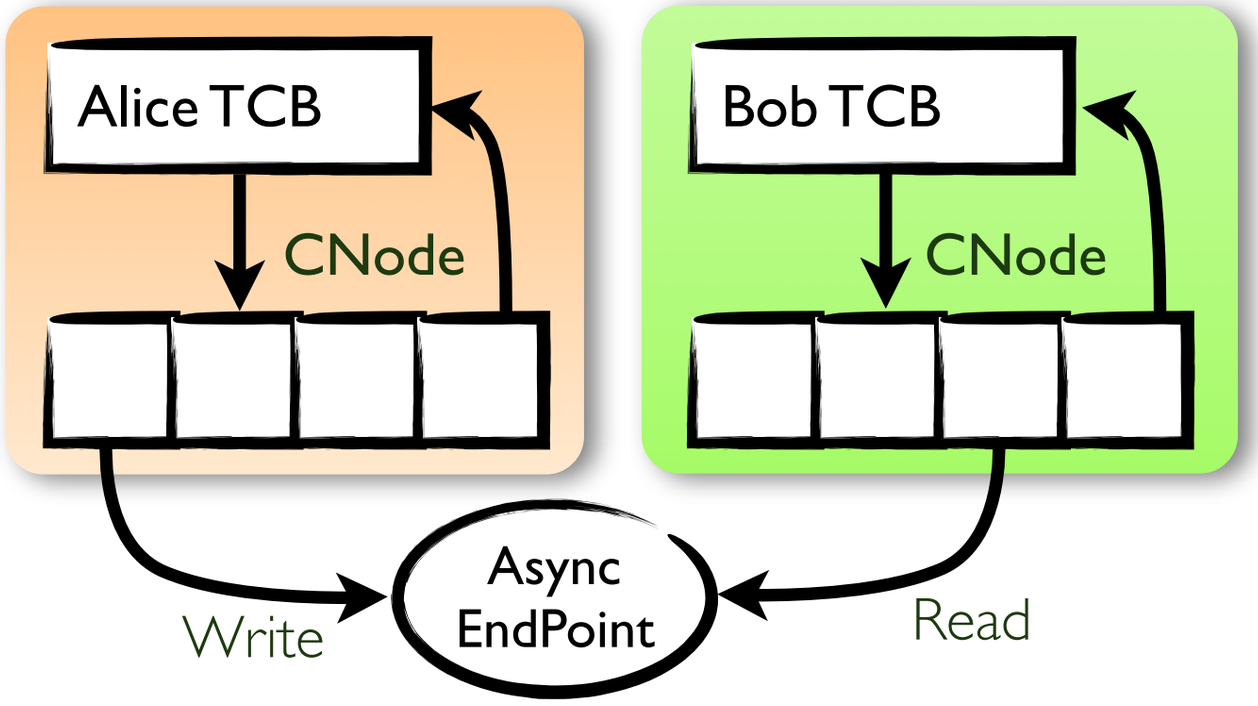
Access Control Policy ⟺ Information Flow

CapDL Model

Functional Specification

High-Performance C Implementation

Binary



seL4 microkernel

NICTA

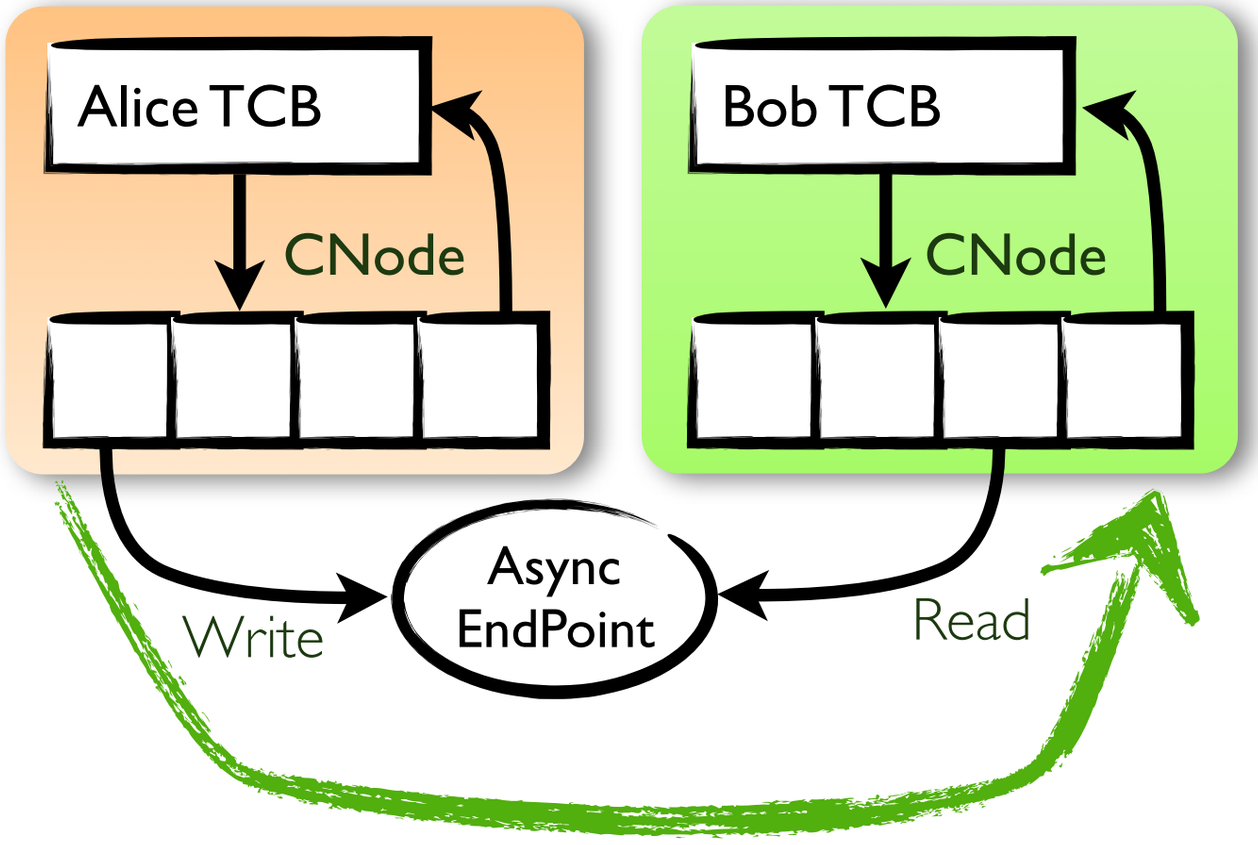
Access Control Policy ⟺ Information Flow

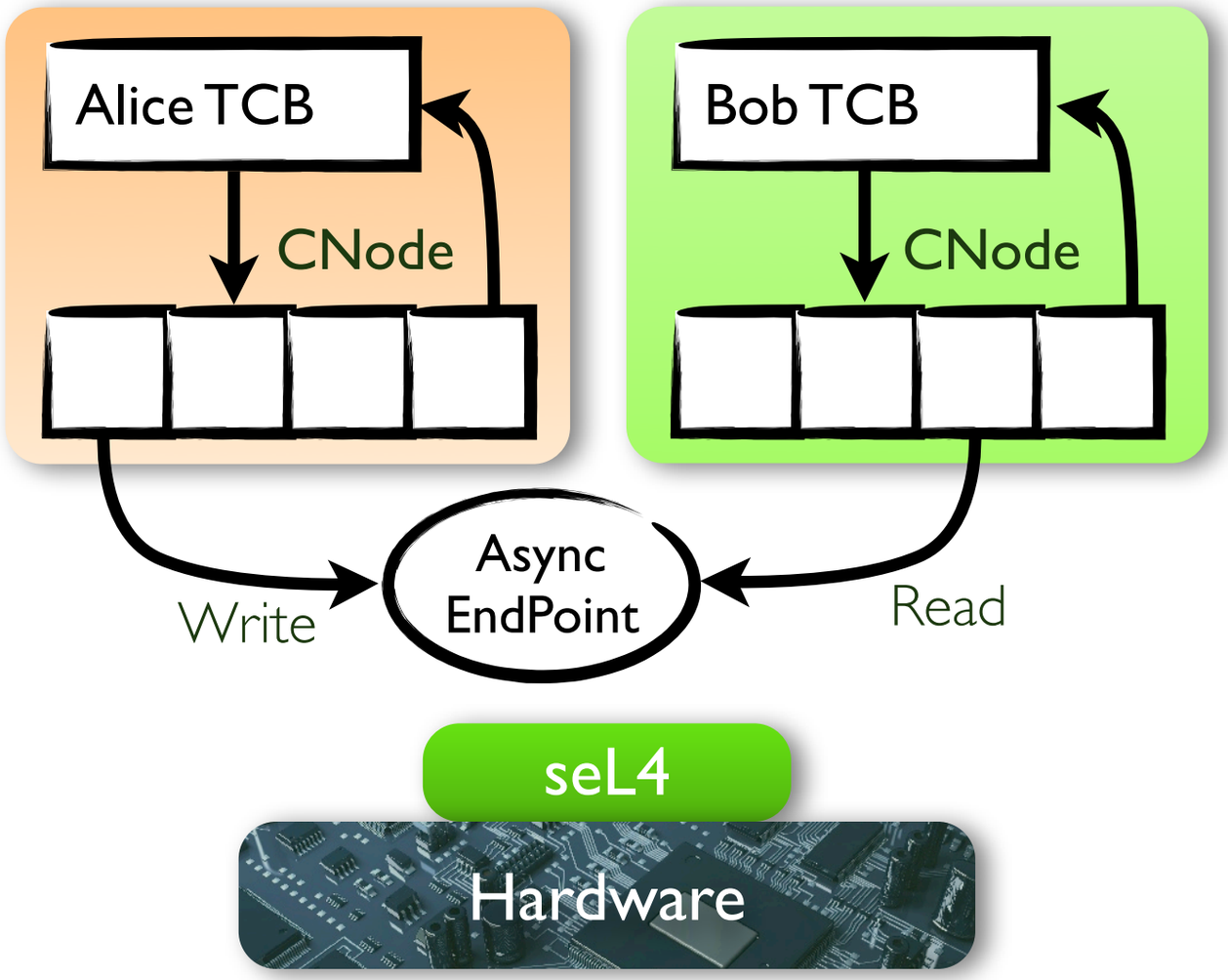
CapDL Model

Functional Specification

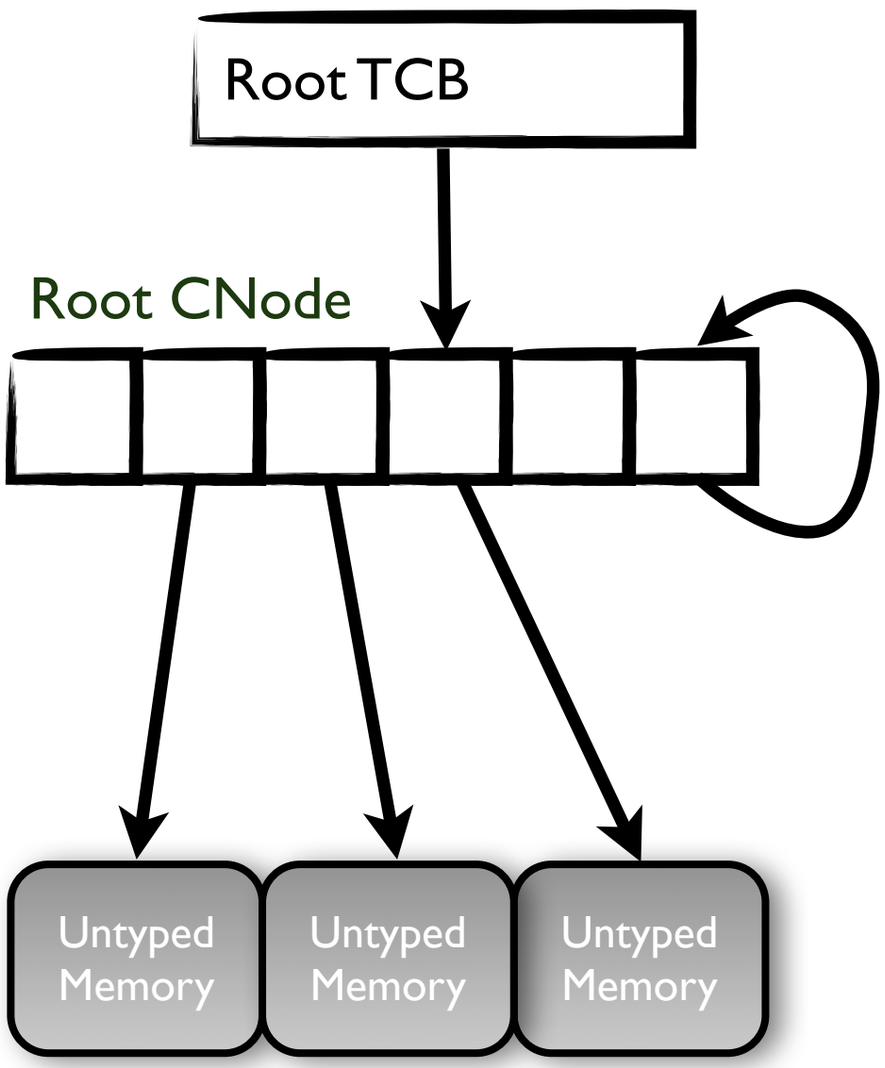
High-Performance C Implementation

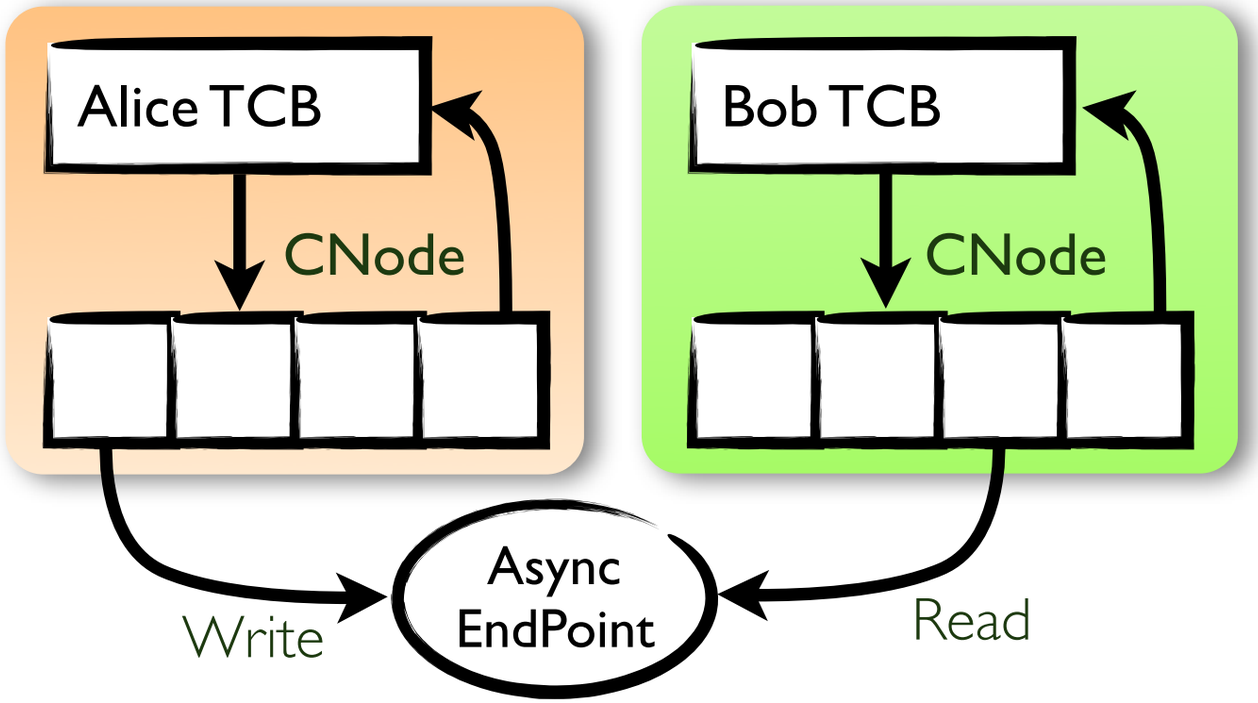
Binary



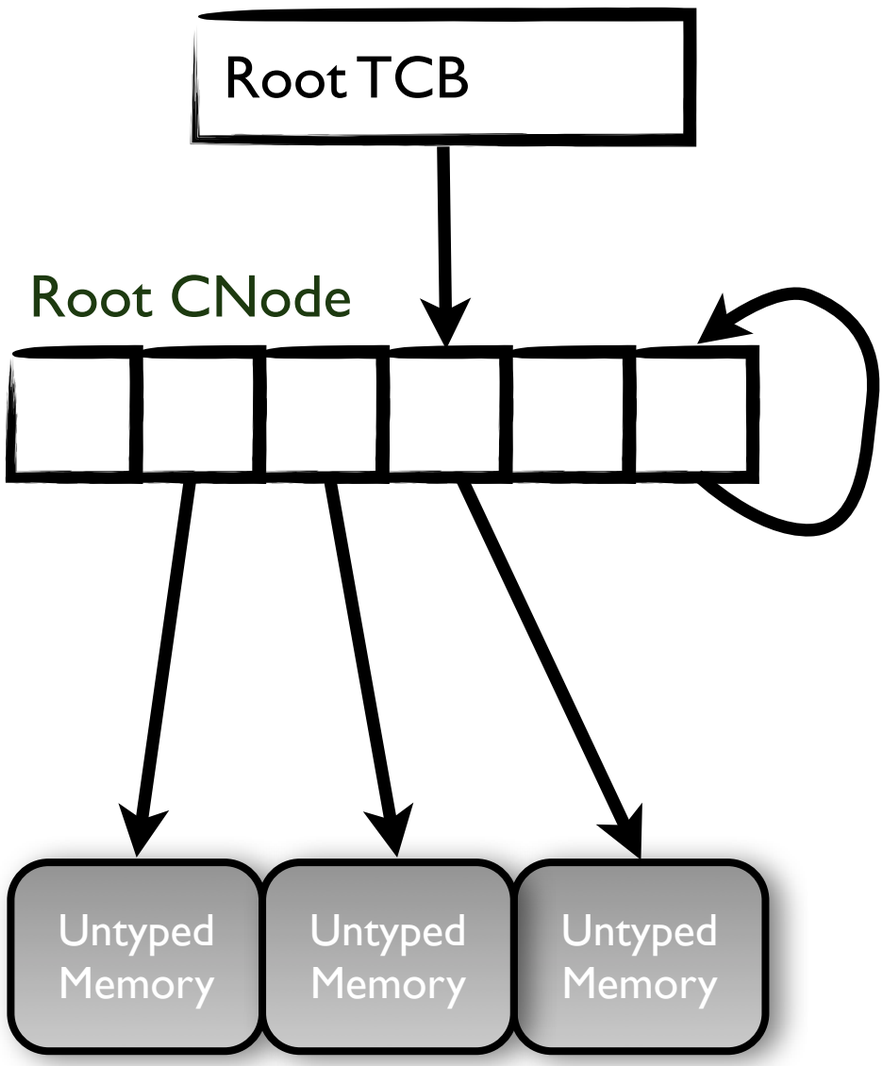

Alice TCB
CNode
Bob TCB
CNode
Async
EndPoint
Write
Read
seL4
Hardware

Start State

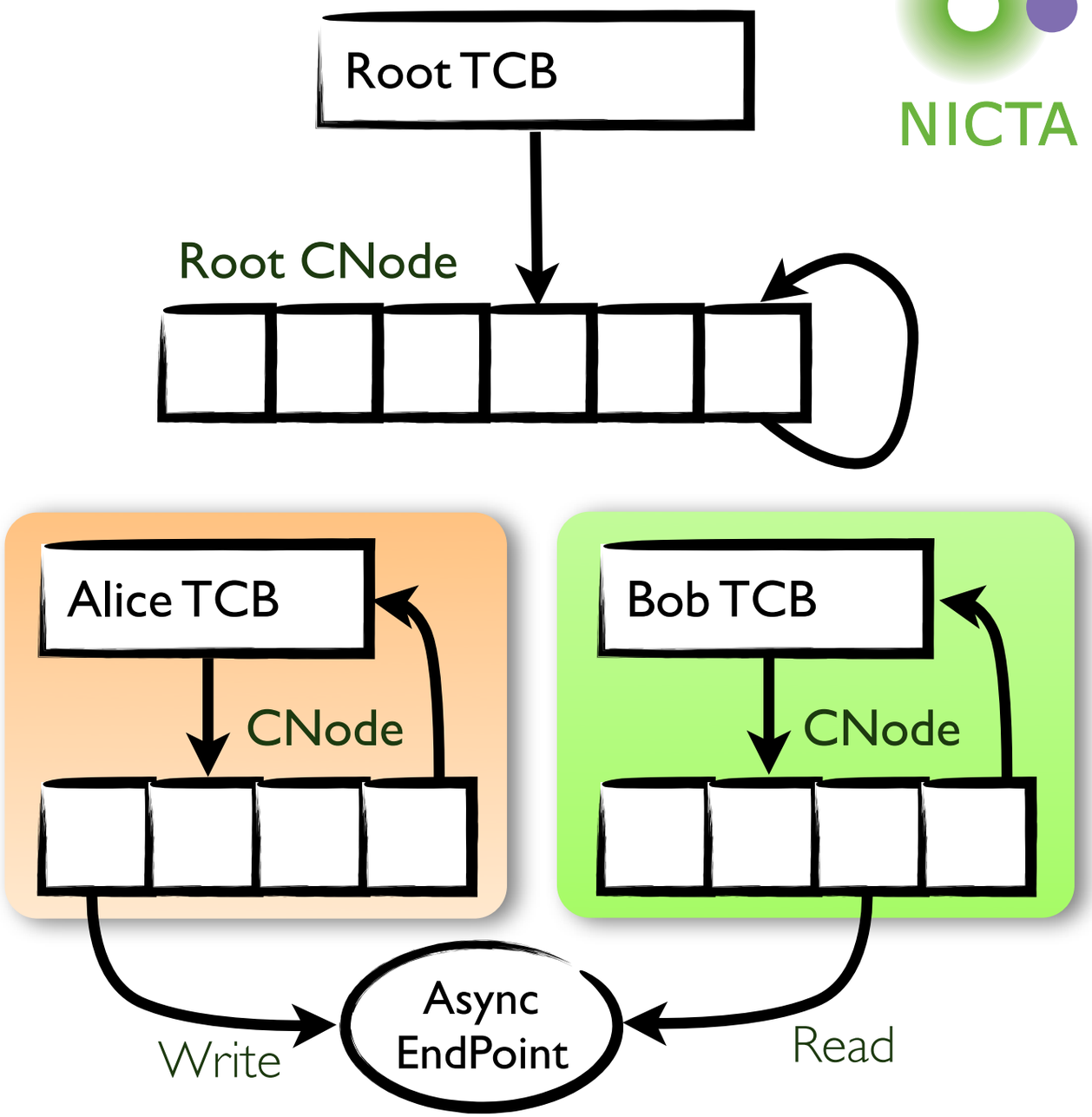




Start State

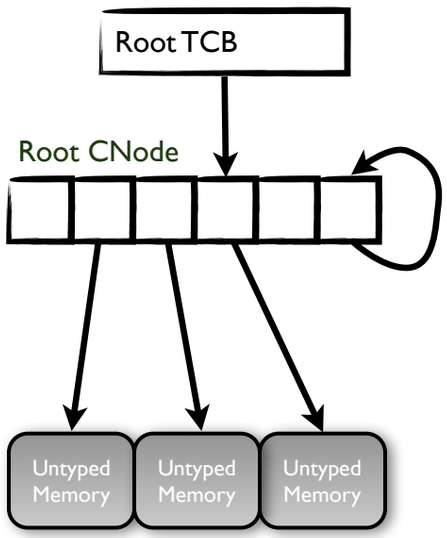


Initialised State

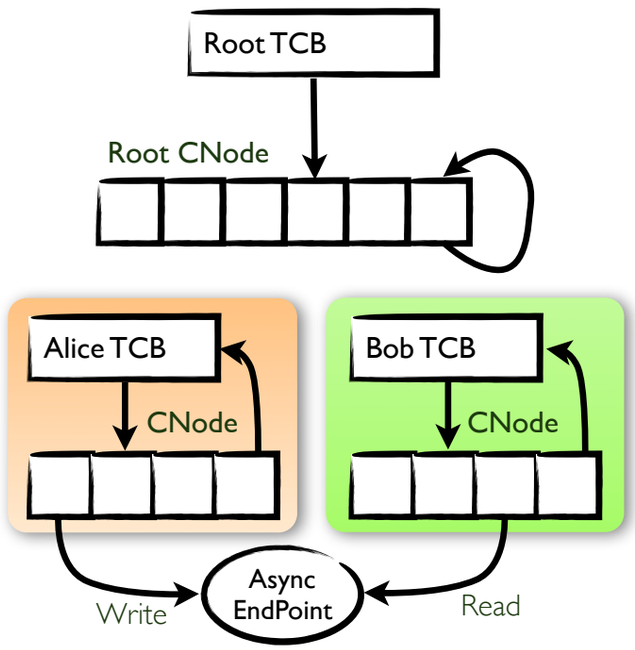


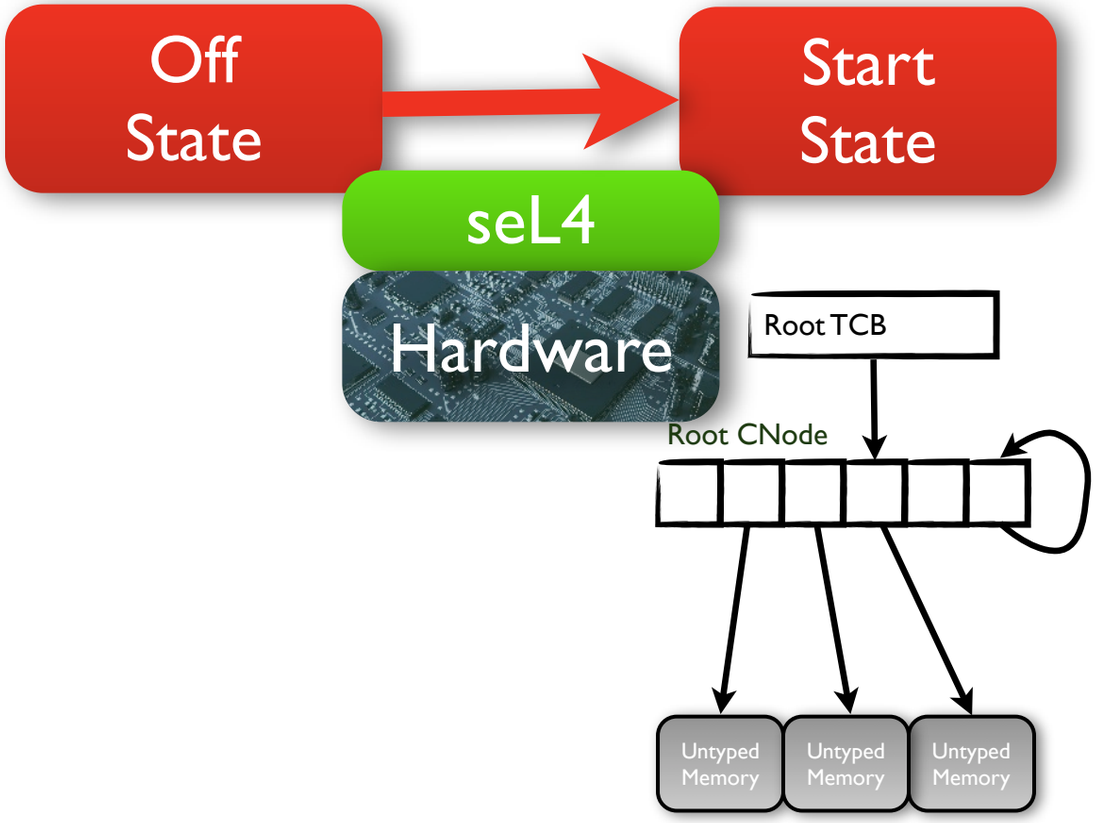
NICTA

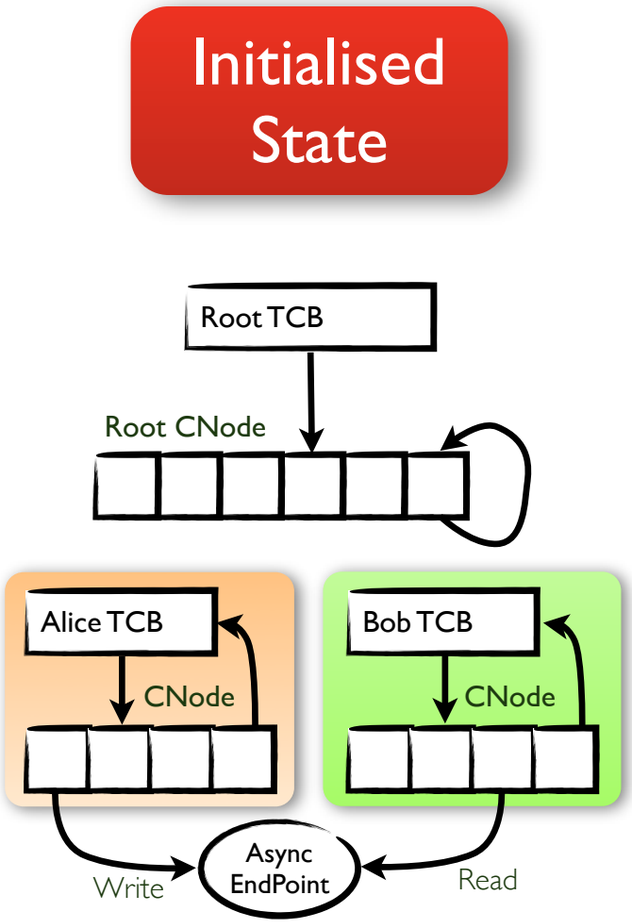
Start State



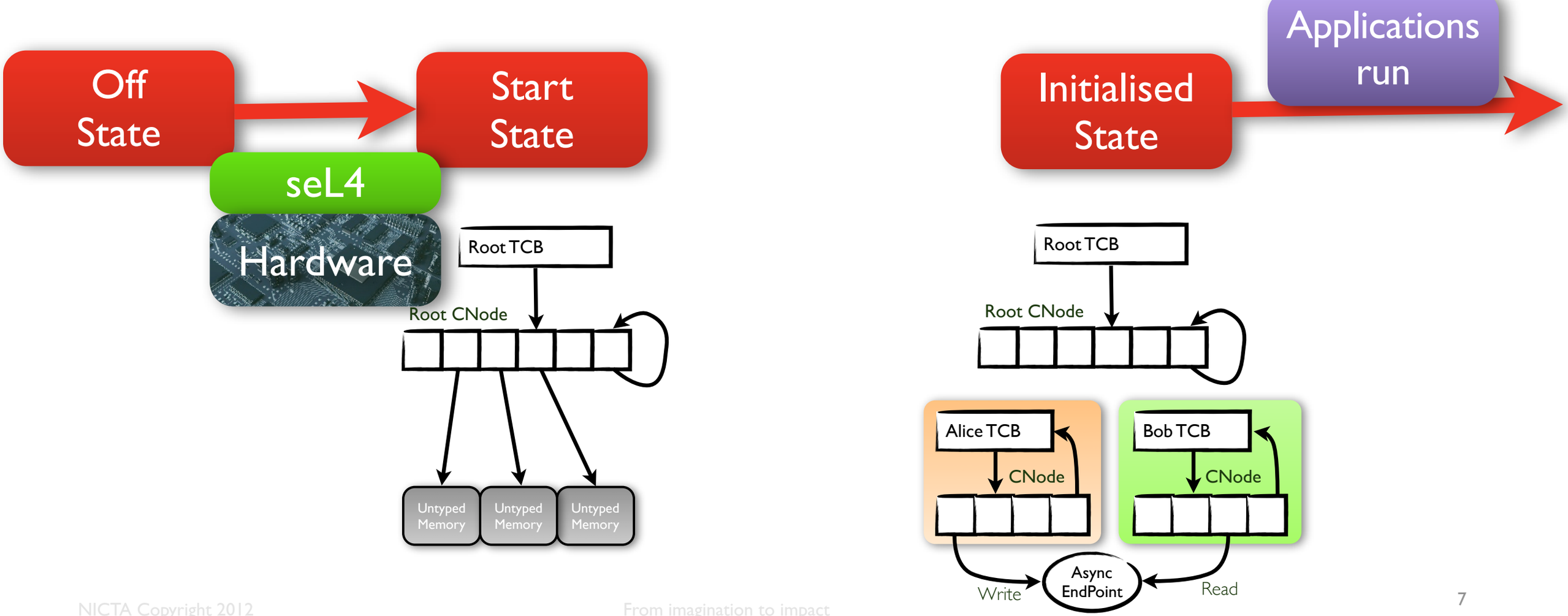
Initialised State







NICTA

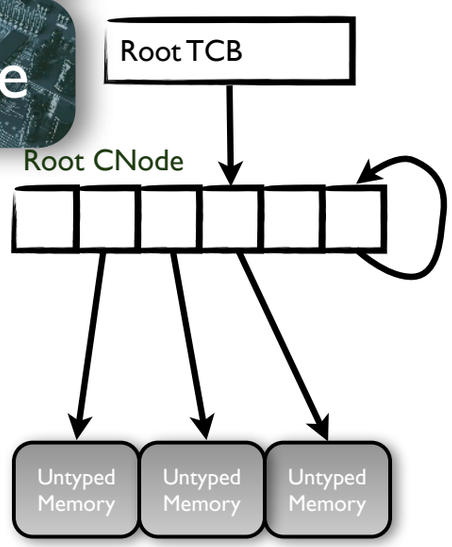

Off State
Start State
seL4
Hardware
Root TCB
Root CNode
Untyped Memory
Untyped Memory
Untyped Memory
Initialised State
Applications run
Root TCB
Root CNode
Alice TCB
CNode
Bob TCB
CNode
Async EndPoint
Write
Read

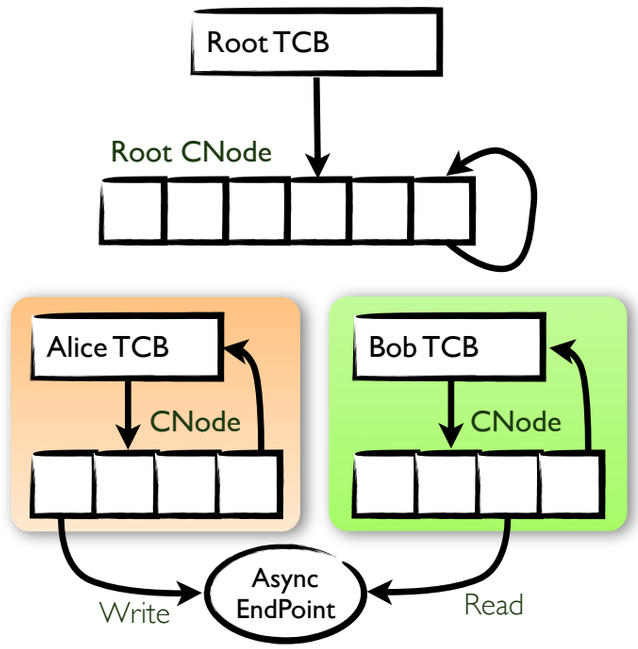
NICTA

Off State → Start State → System Initialiser → Initialised State → Applications run

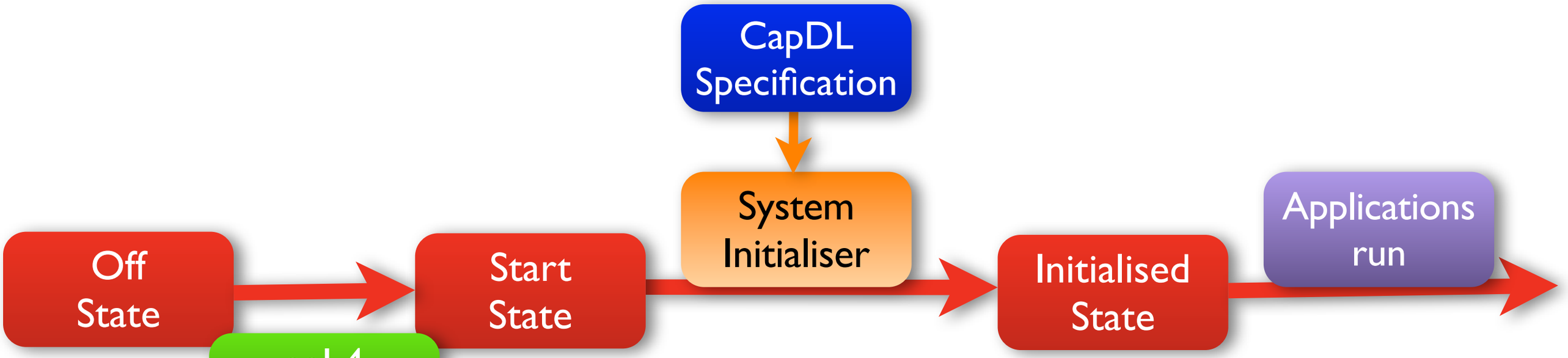
seL4

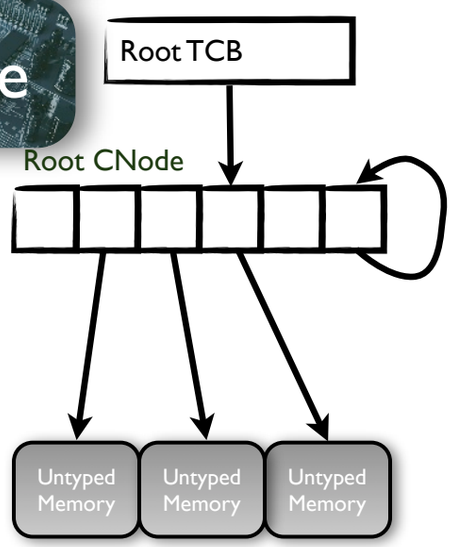
Hardware

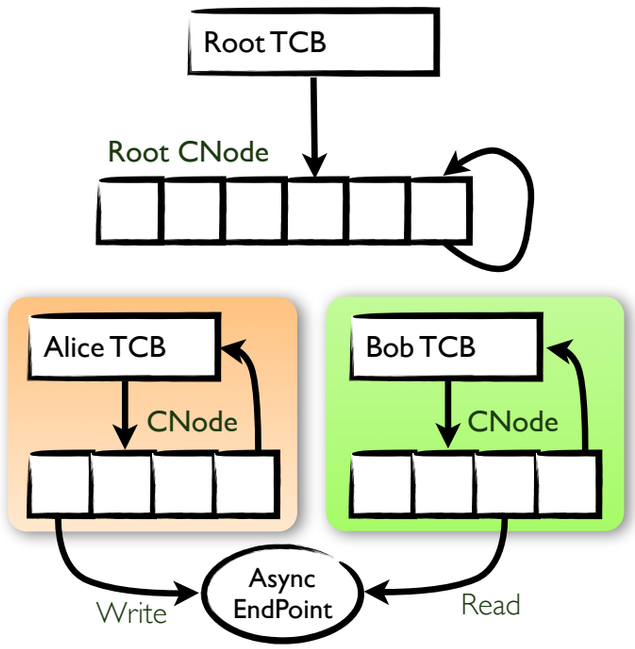




NICTA







```
objects { tcb_alice = tcb ()
          cnode_alice = cnode (2 bits)
          cnode_bob = cnode (2 bits)
          tcb_bob = tcb ()
          aep_shared = aep}
caps { tcb_alice { cspace: cnode_alice (guard: 0, guard_size: 30) }
       tcb_bob { cspace: cnode_bob (guard: 0, guard_size: 30) }
       cnode_alice { 0x0: aep_shared (W) }
       cnode_bob { 0x2: aep_shared (R) } }
```

NICTA

CapDL Specification

System Initialiser

Off State

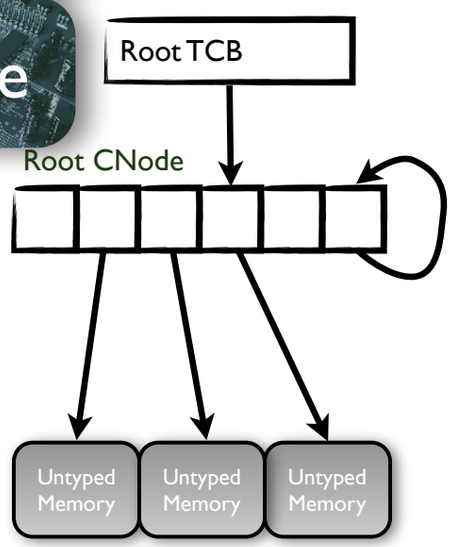
Start State

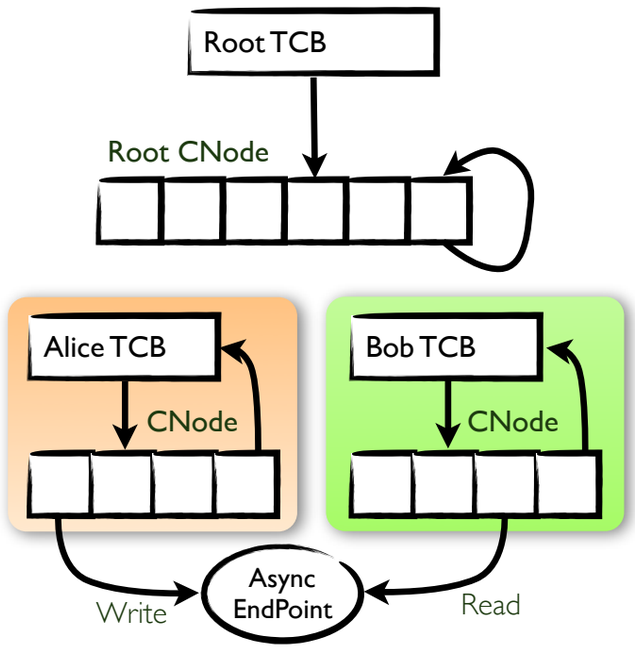
Initialised State

Applications run

seL4

Hardware





```
objects { tcb_alice = tcb ()
          cnode_alice = cnode (2 bits)
          cnode_bob = cnode (2 bits)
          tcb_bob = tcb ()
          aep_shared = aep}
caps { tcb_alice { cspace: cnode_alice (guard: 0, guard_size: 30) }
       tcb_bob { cspace: cnode_bob (guard: 0, guard_size: 30) }
       cnode_alice { 0x0: aep_shared (W) }
       cnode_bob { 0x2: aep_shared (R) } }
```

NICTA

CapDL Specification

Equivalent

System Initialiser

Off State

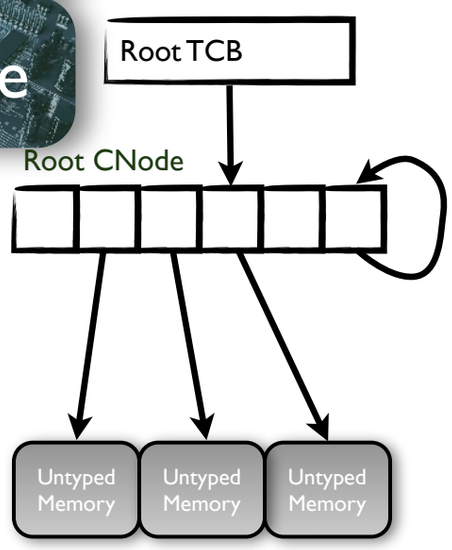
Start State

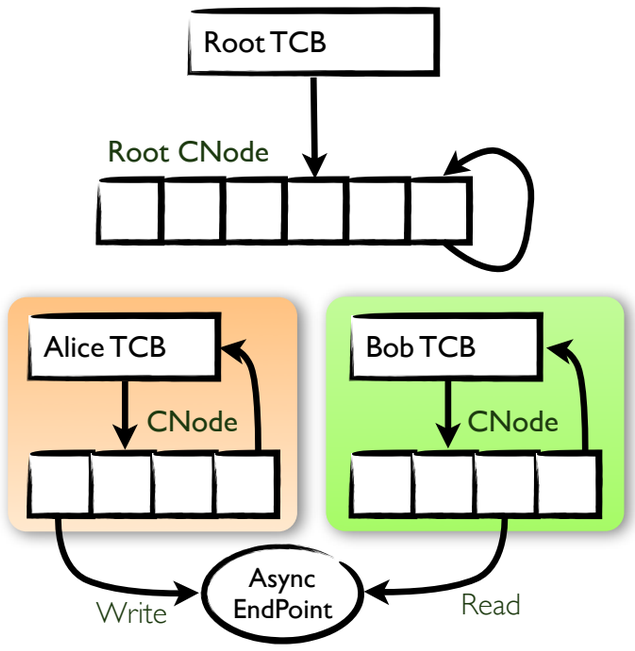
Initialised State

Applications run

seL4

Hardware





```
objects { tcb_alice = tcb ()
          cnode_alice = cnode (2 bits)
          cnode_bob = cnode (2 bits)
          tcb_bob = tcb ()
          aep_shared = aep}
caps { tcb_alice { cspace: cnode_alice (guard: 0, guard_size: 30) }
       tcb_bob { cspace: cnode_bob (guard: 0, guard_size: 30) }
       cnode_alice { 0x0: aep_shared (W) }
       cnode_bob { 0x2: aep_shared (R) } }
```

{valid_boot_info ***boot_info spec***}

init_system ***spec boot_info***

{objects_initialised ***spec***}

CapDL Specification

Equivalent

System Initialiser

Off State

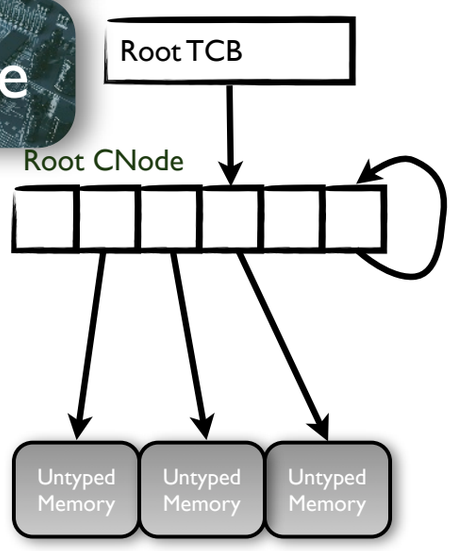
Start State

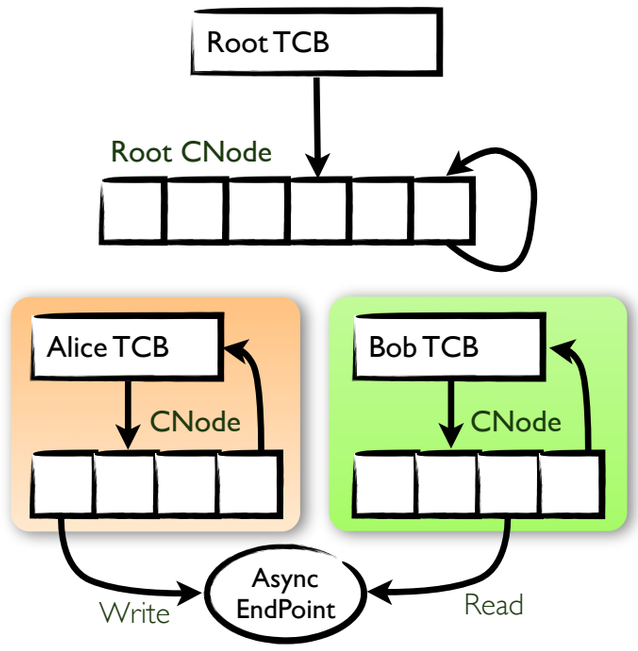
Initialised State

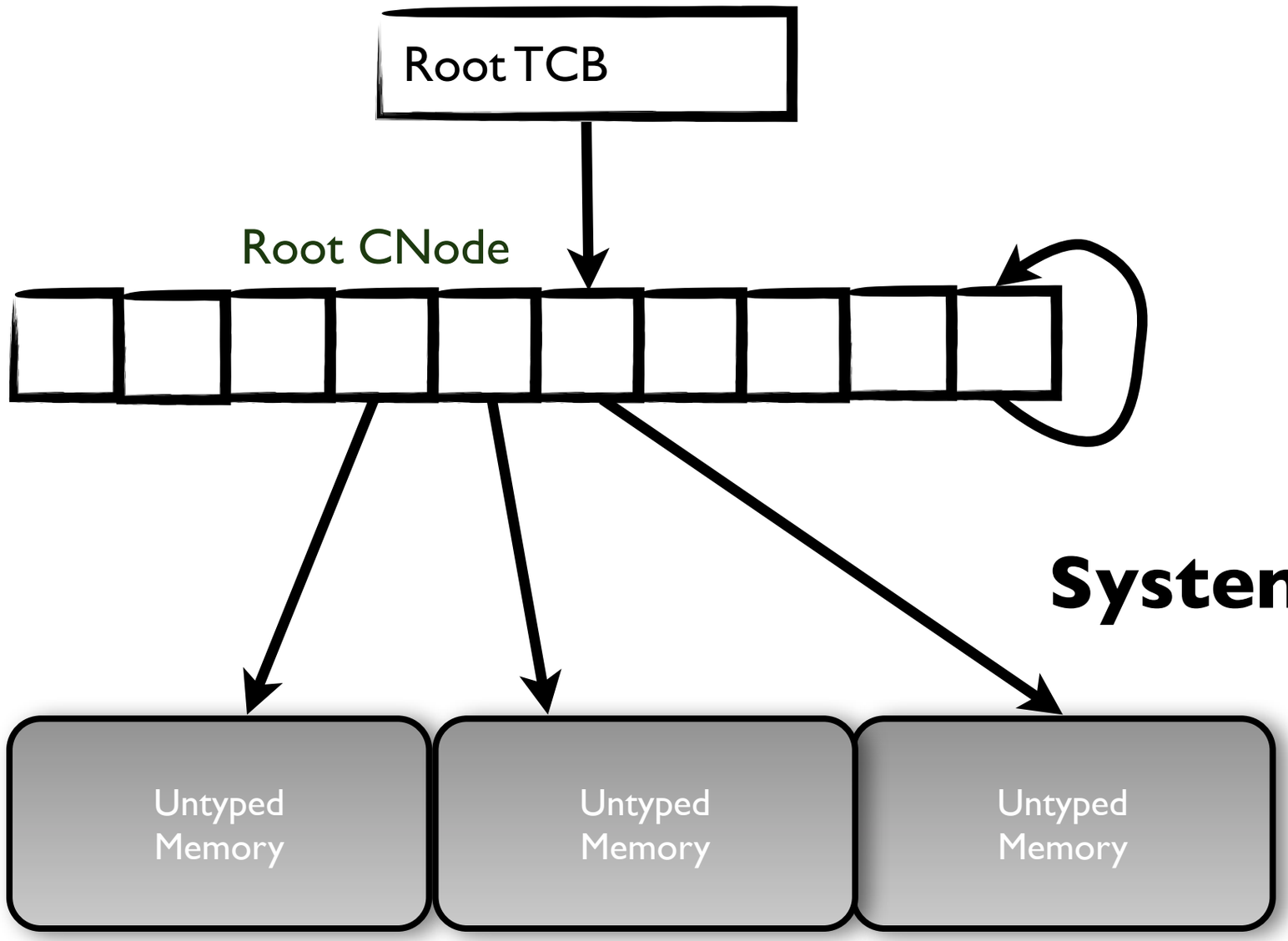
Applications run

seL4

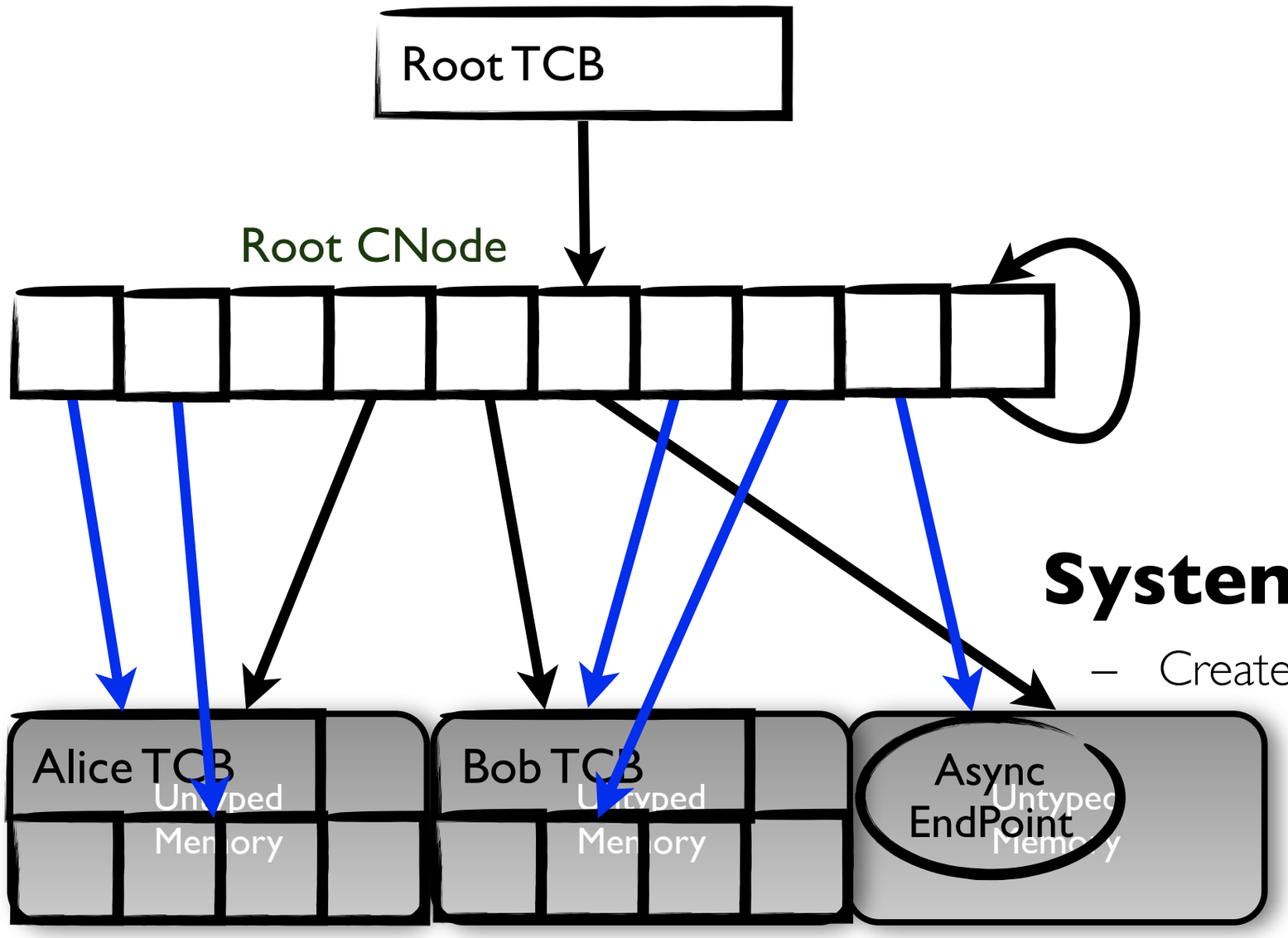
Hardware





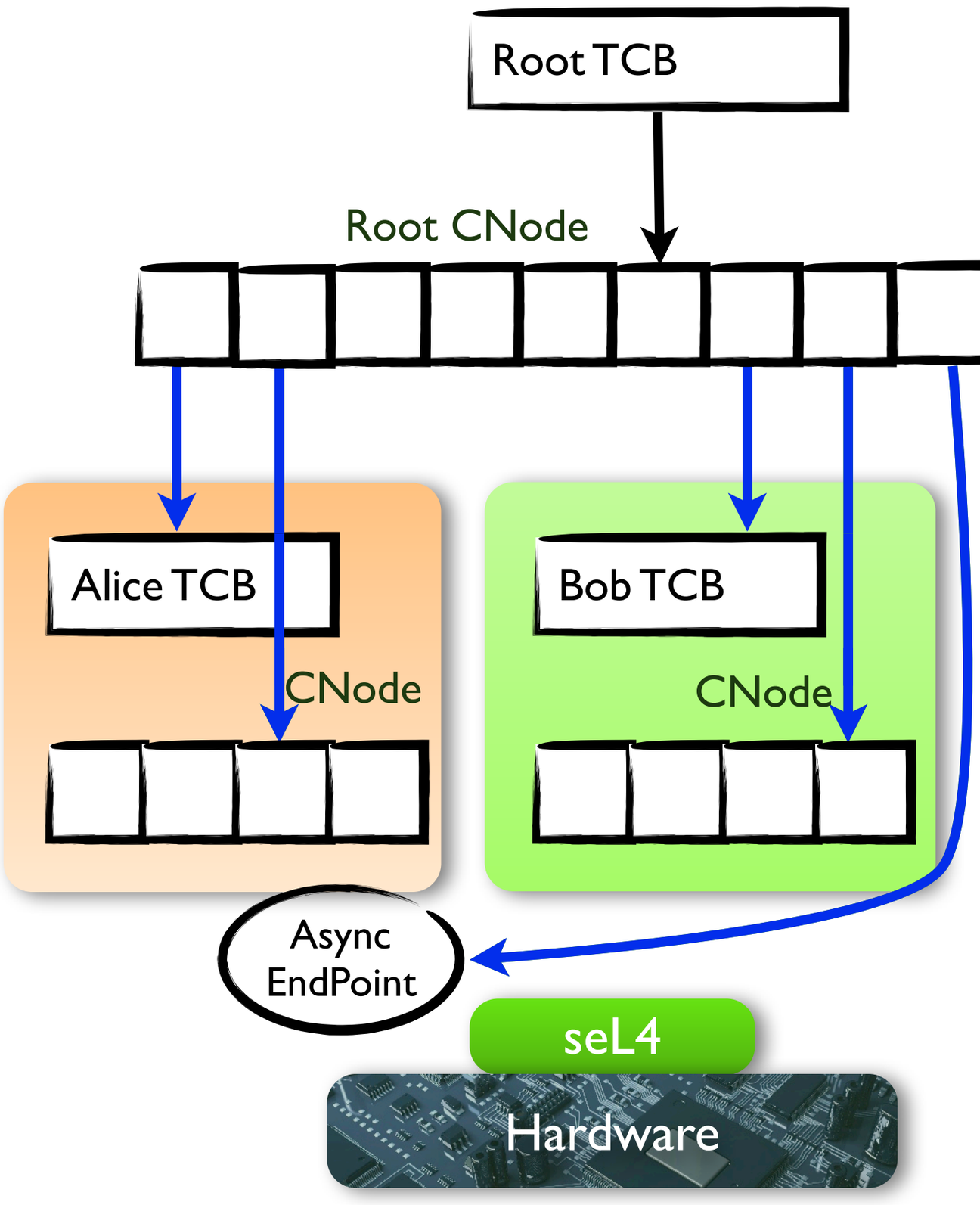


System initialisation:



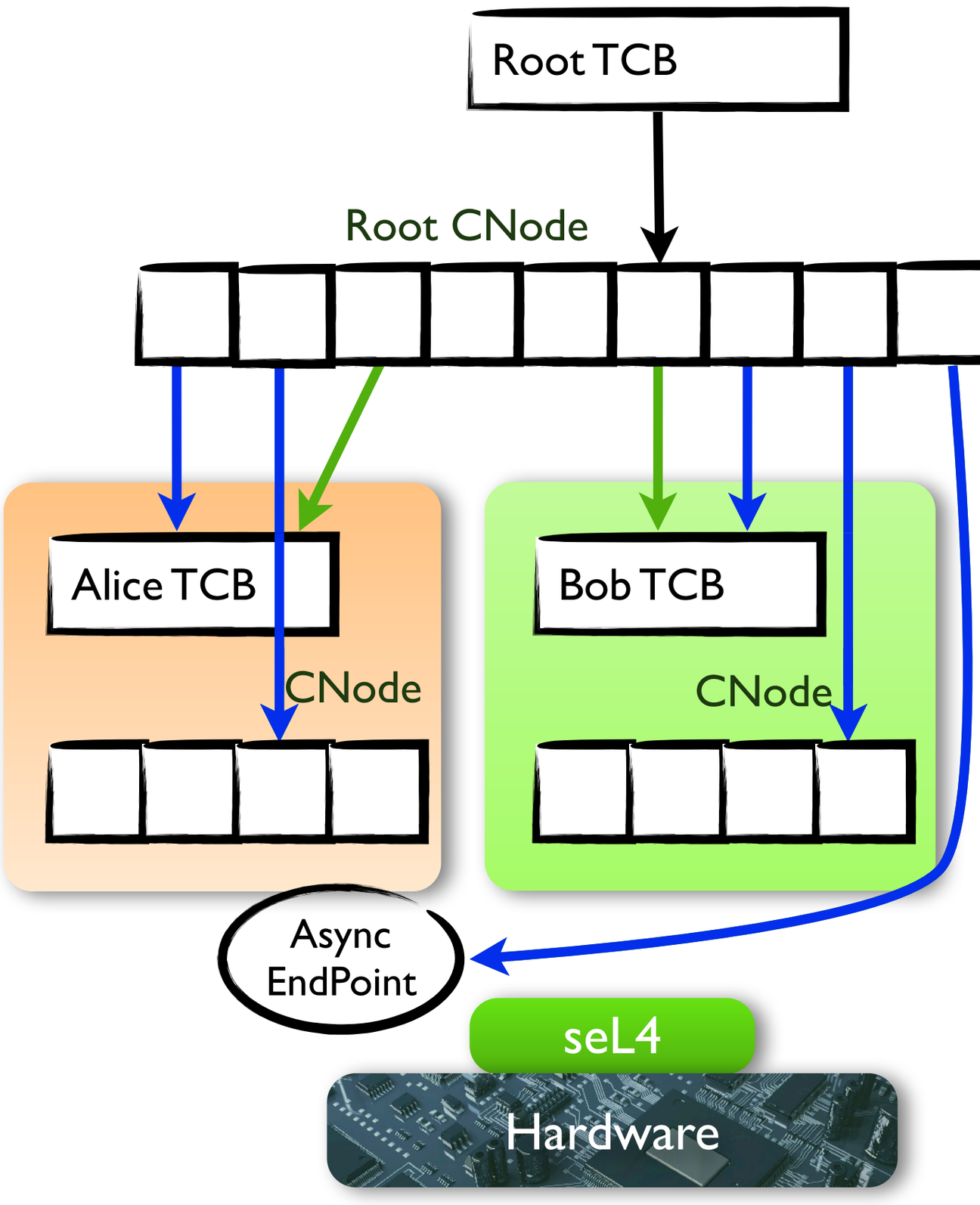
System initialisation:

- Create objects



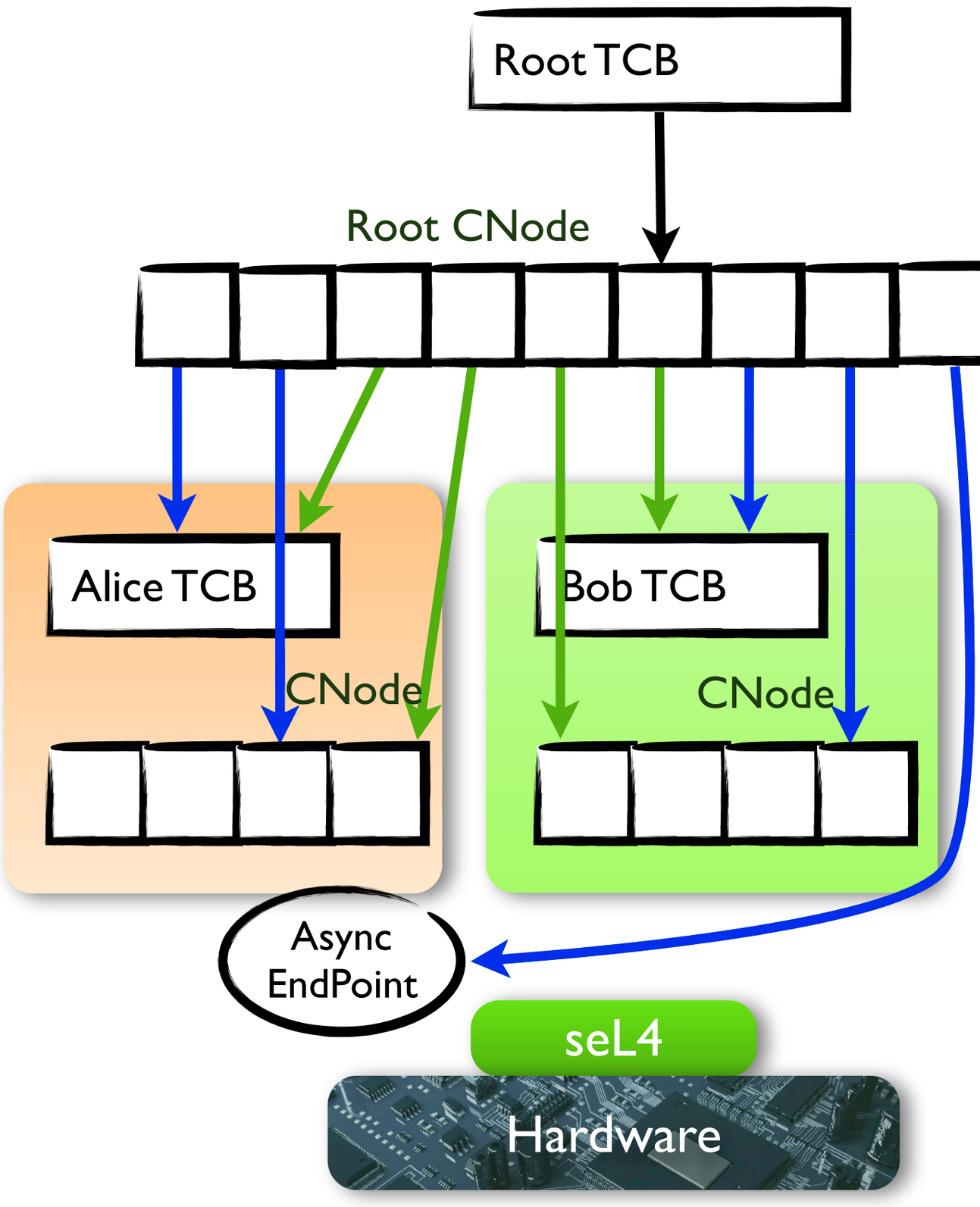
System initialisation:

- Create objects
- Duplicate capabilities



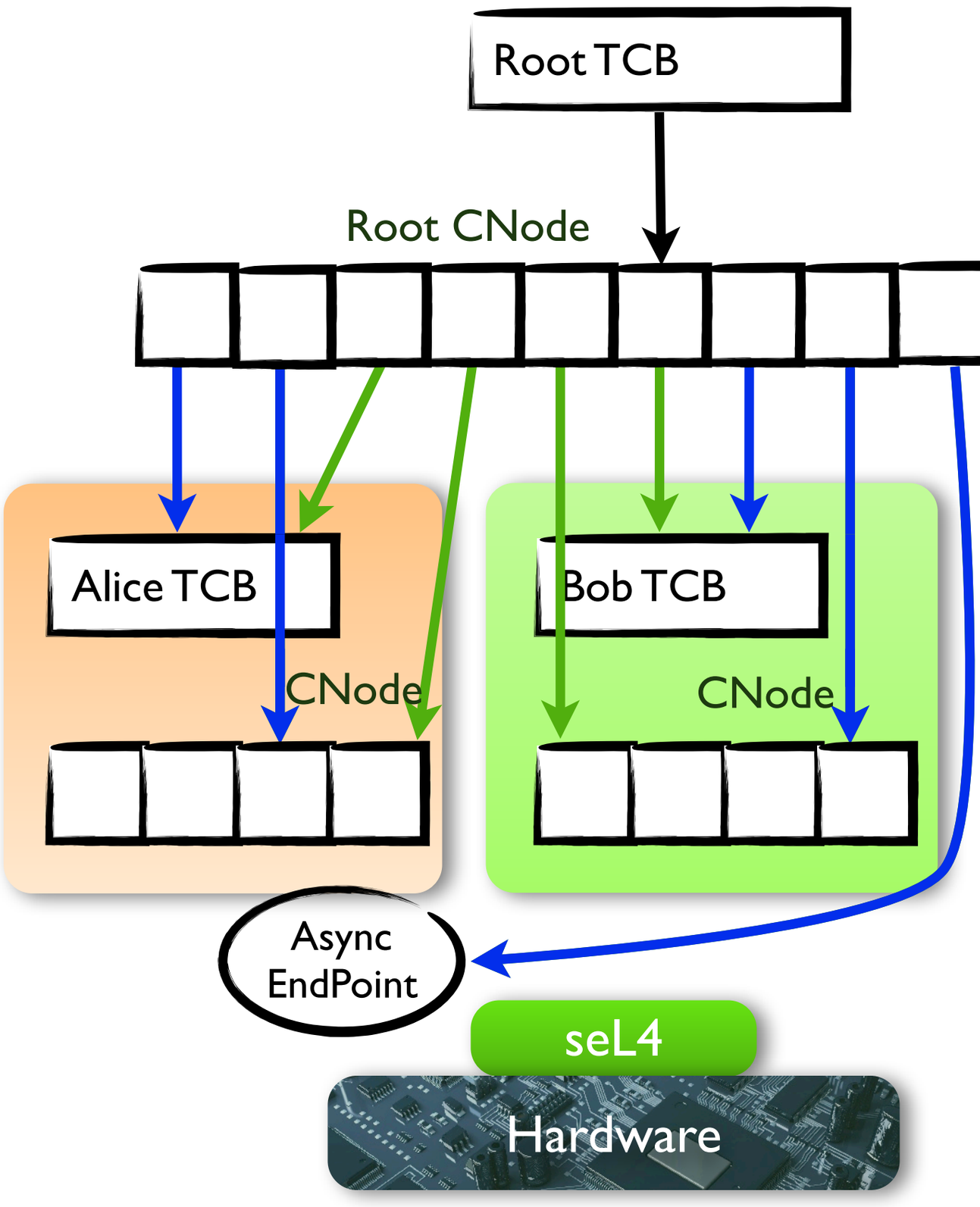
System initialisation:

- Create objects
- Duplicate capabilities



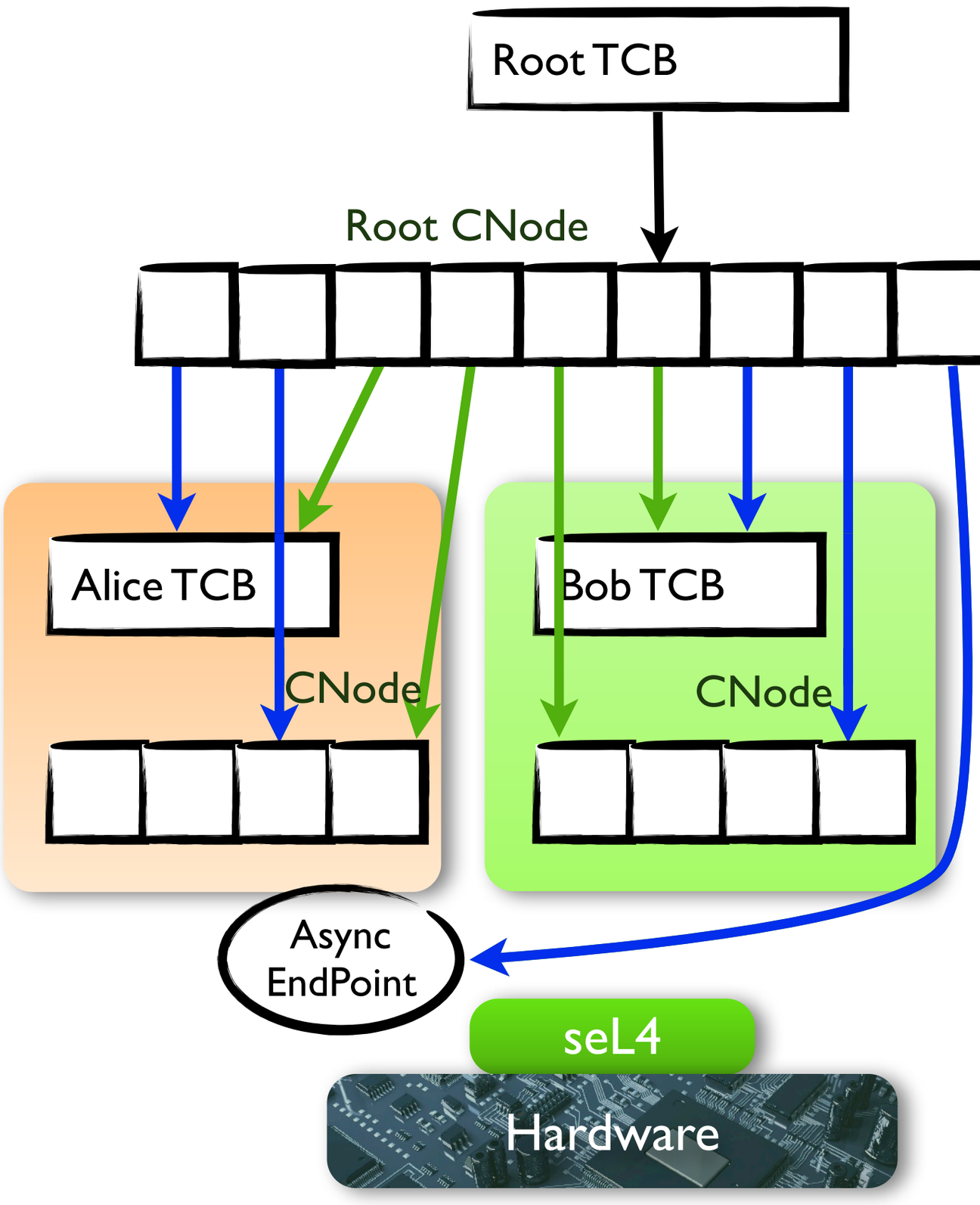
System initialisation:

- Create objects
- Duplicate capabilities



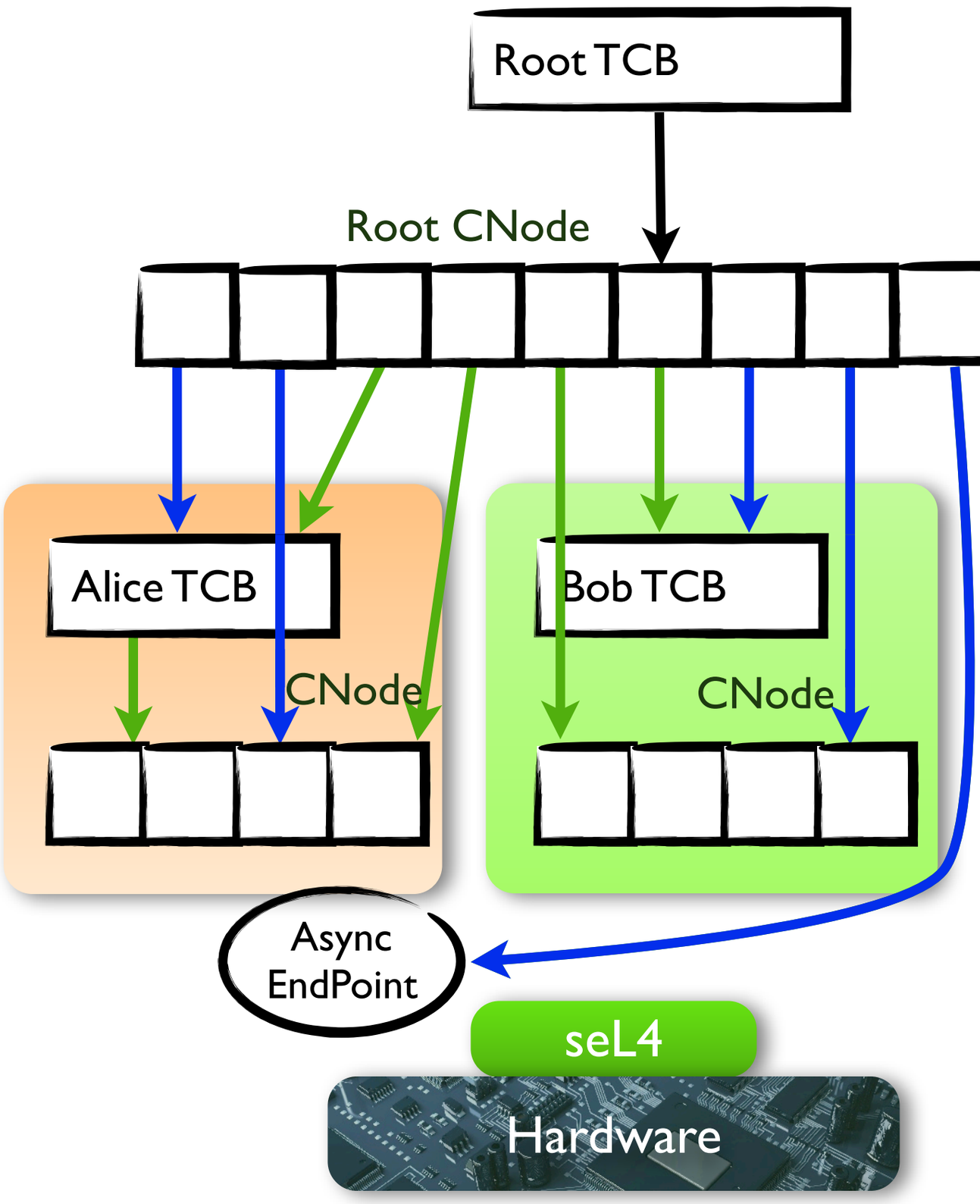
System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space



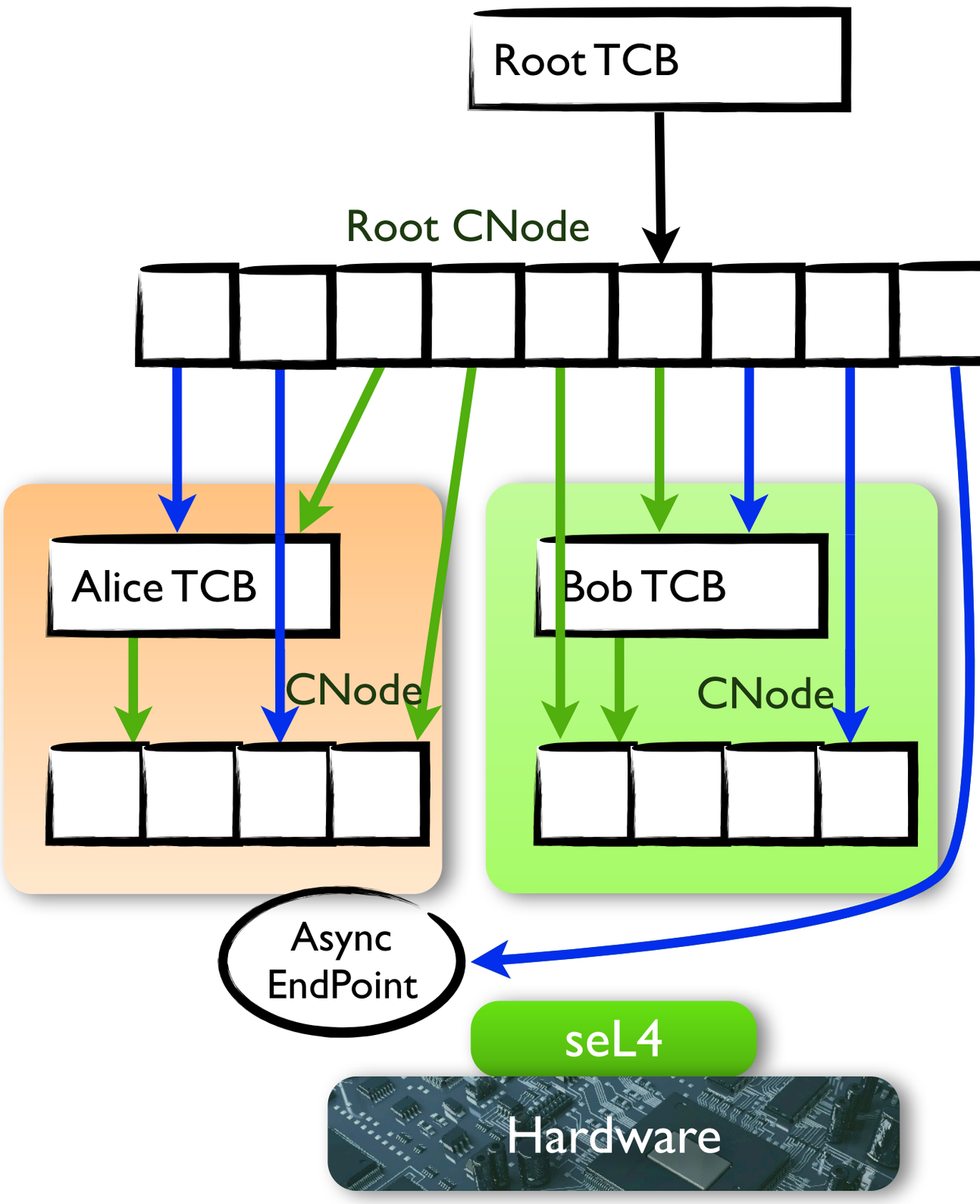
System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space
- Initialise thread control blocks



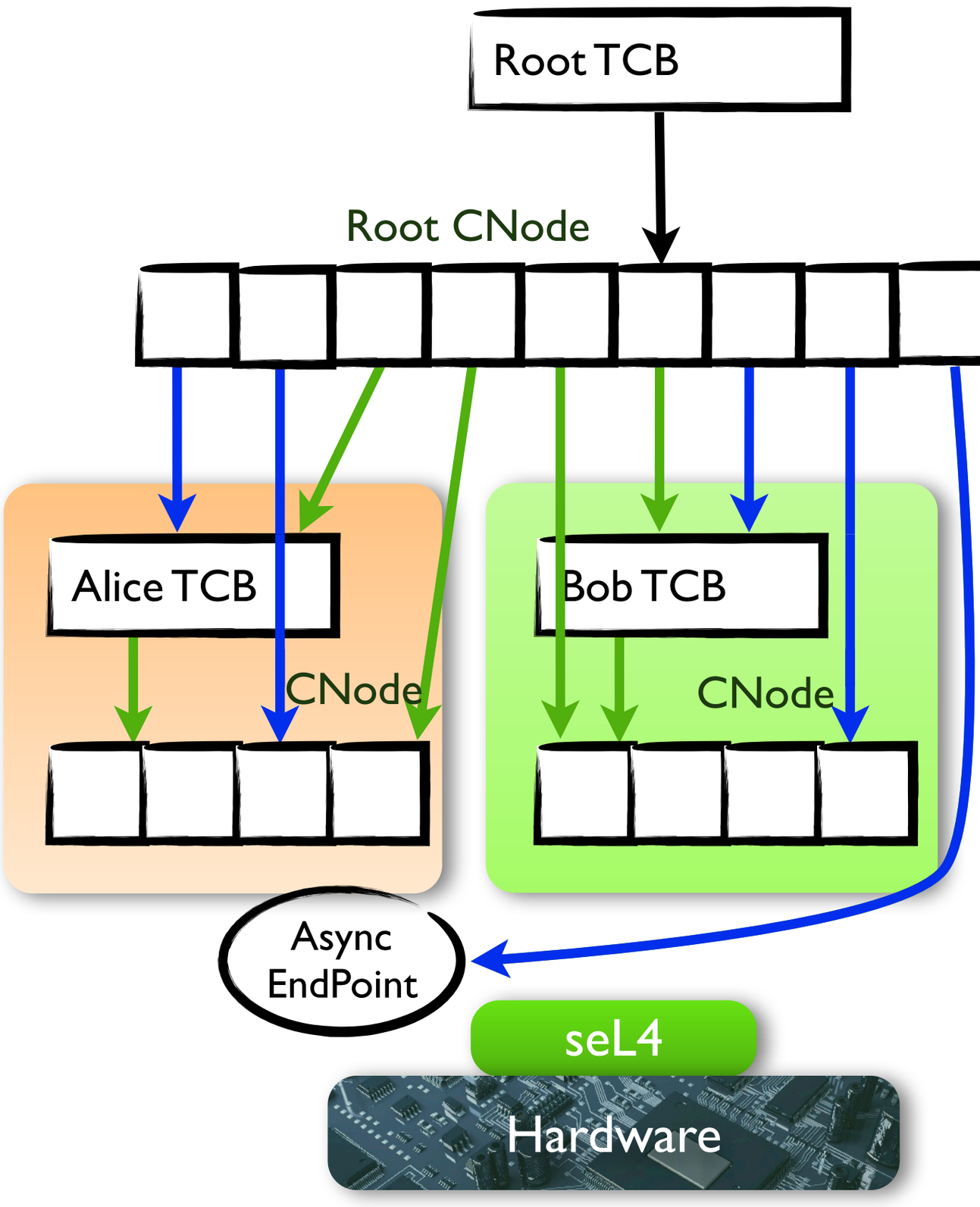
System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space
- Initialise thread control blocks



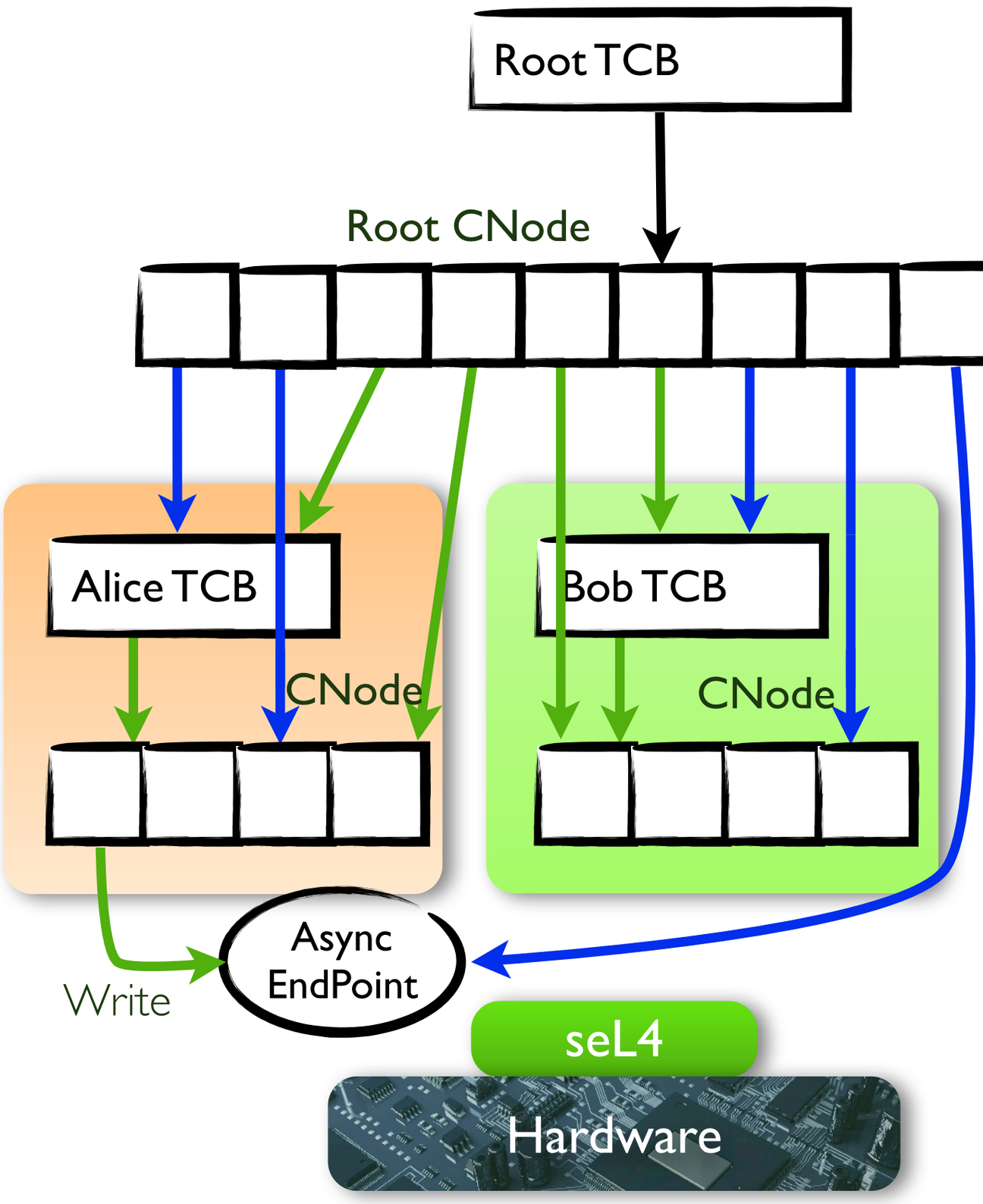
System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space
- Initialise thread control blocks



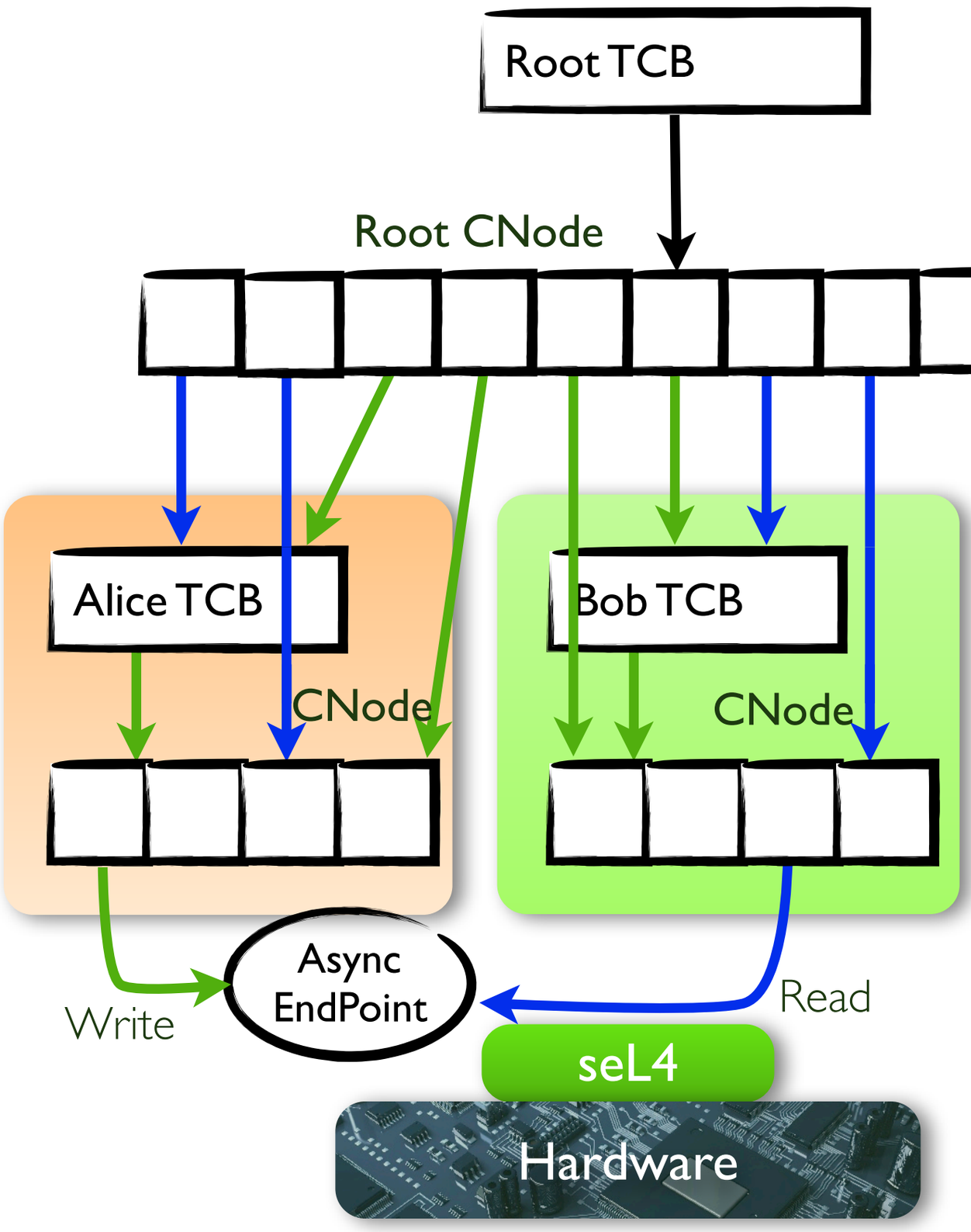
System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space
- Initialise thread control blocks
- Initialise capability address space



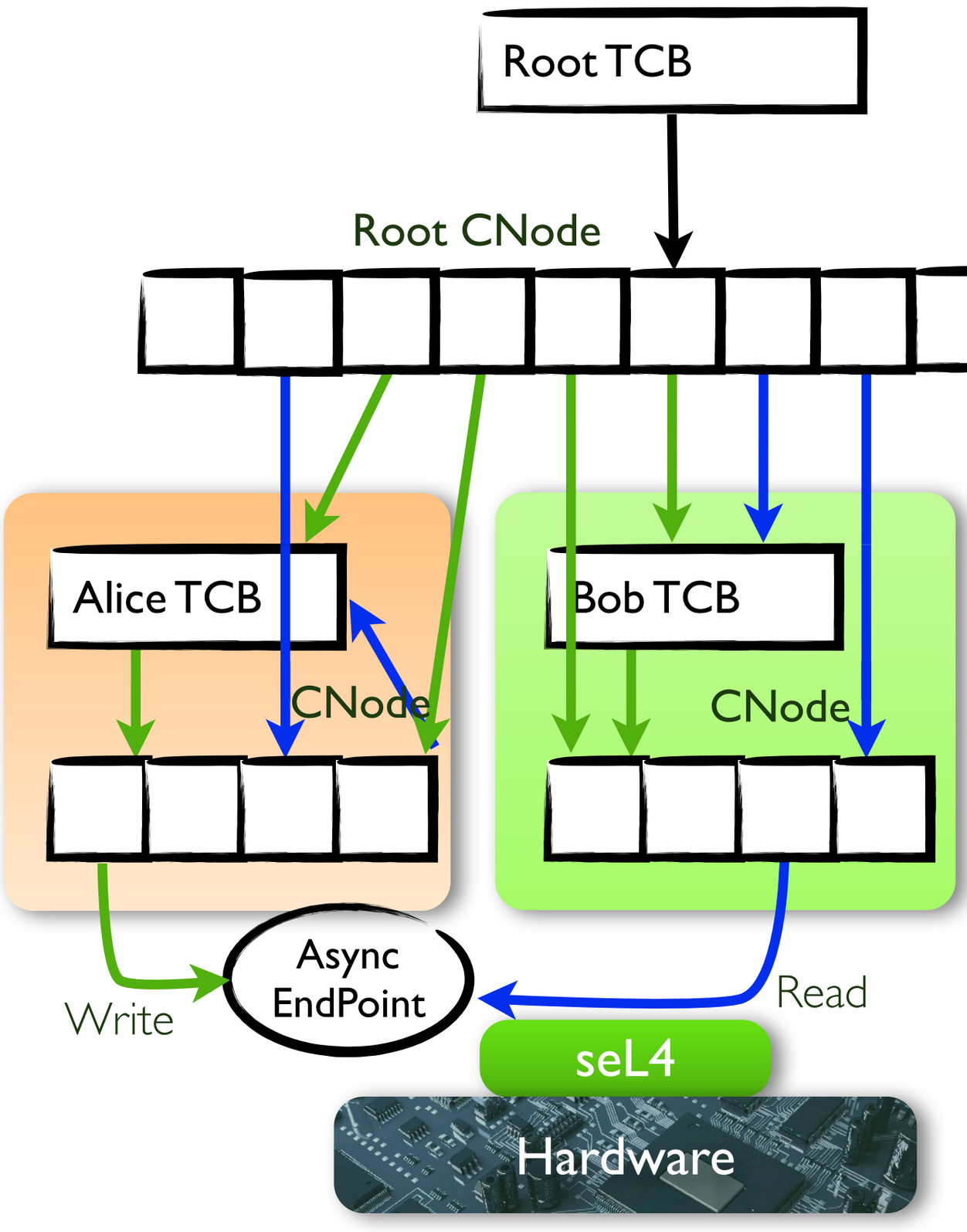
System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space
- Initialise thread control blocks
- Initialise capability address space



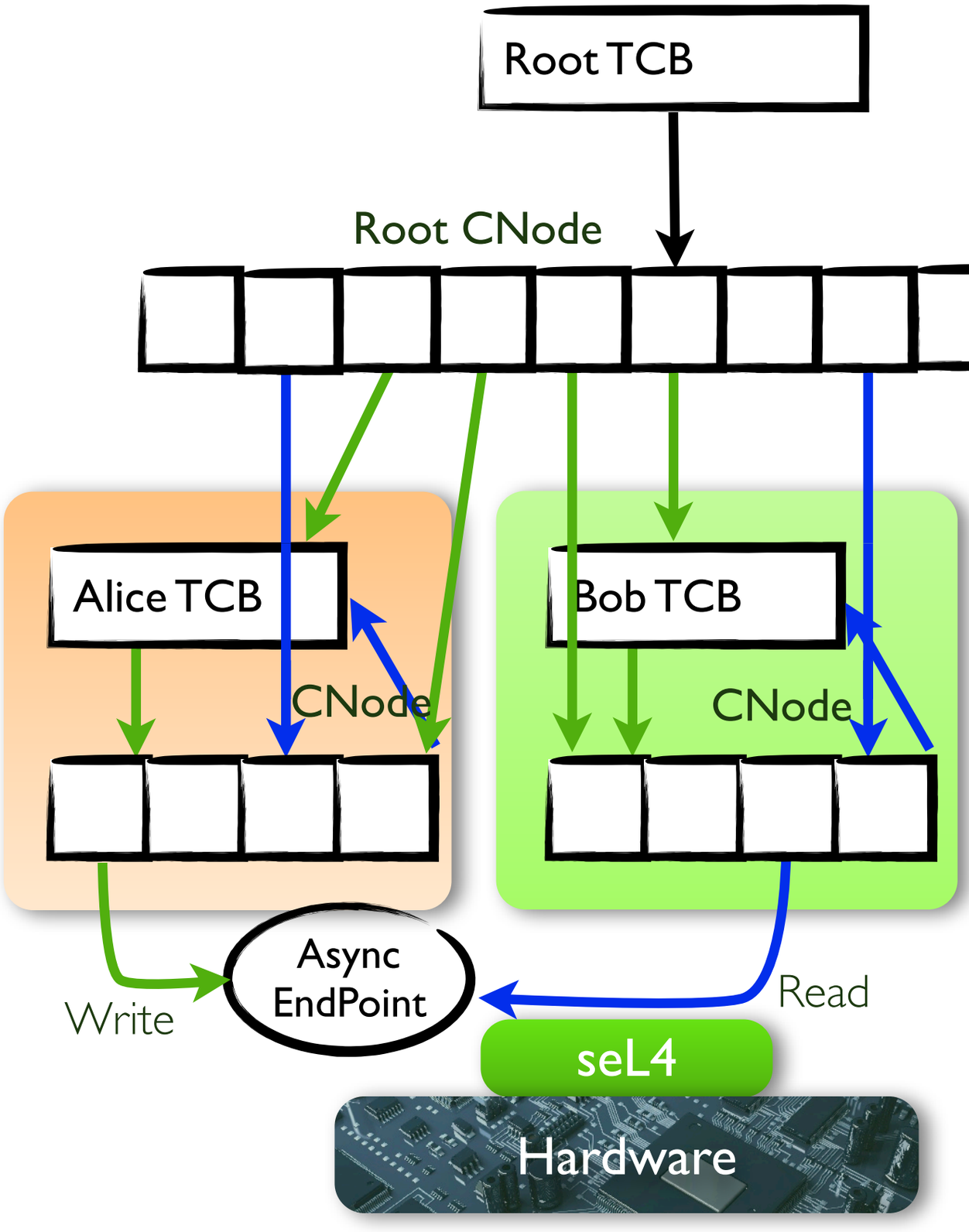
System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space
- Initialise thread control blocks
- Initialise capability address space



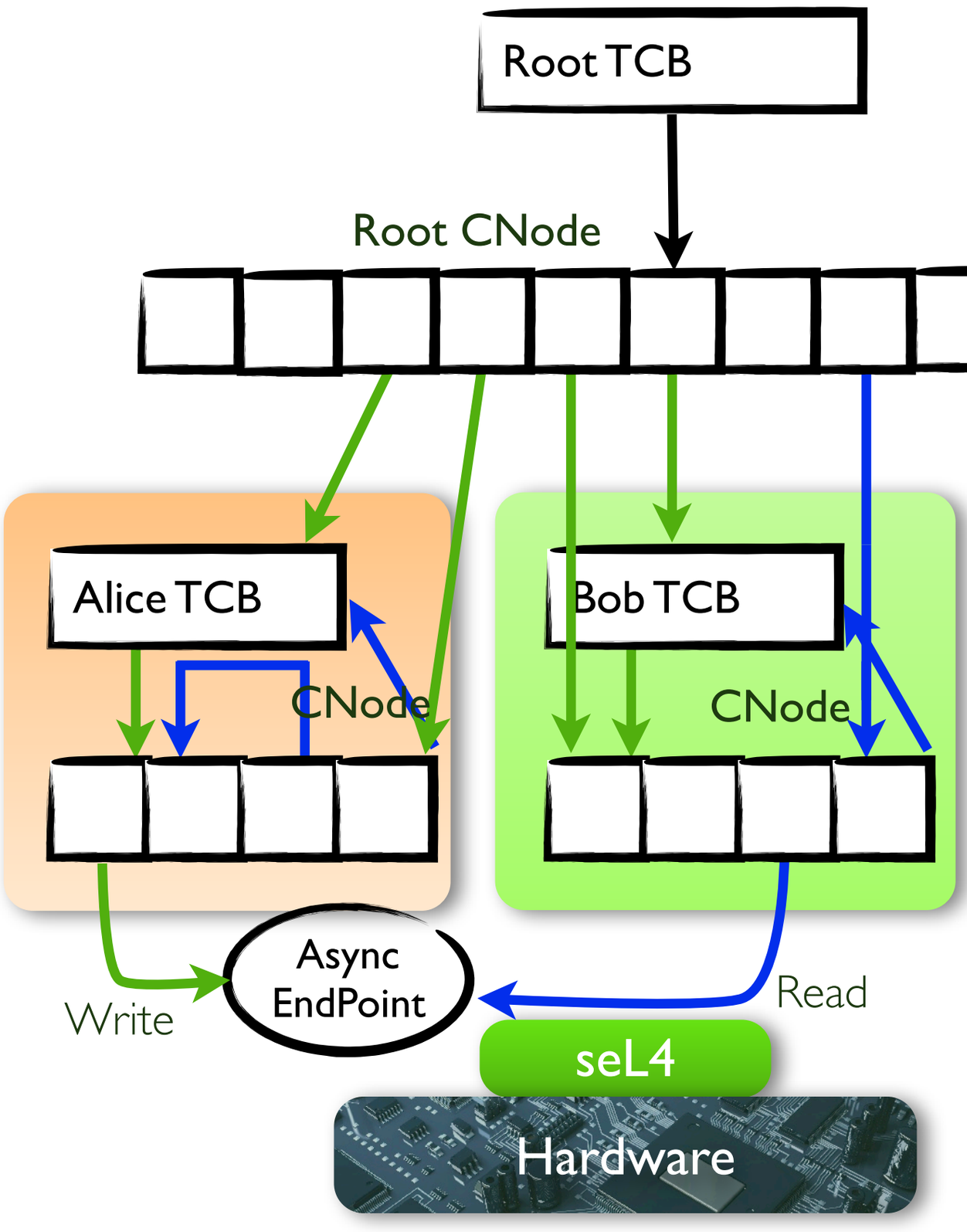
System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space
- Initialise thread control blocks
- Initialise capability address space



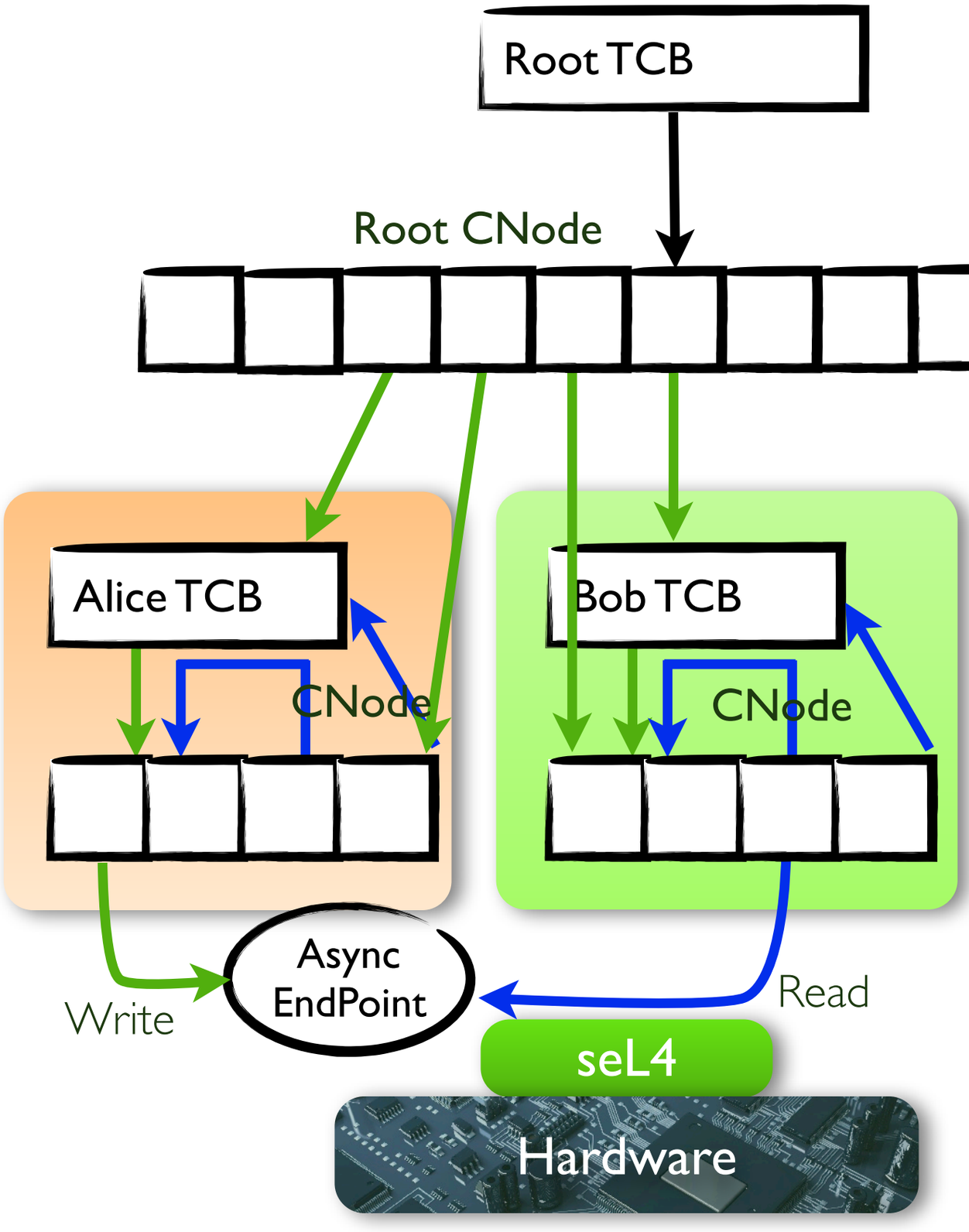
System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space
- Initialise thread control blocks
- Initialise capability address space



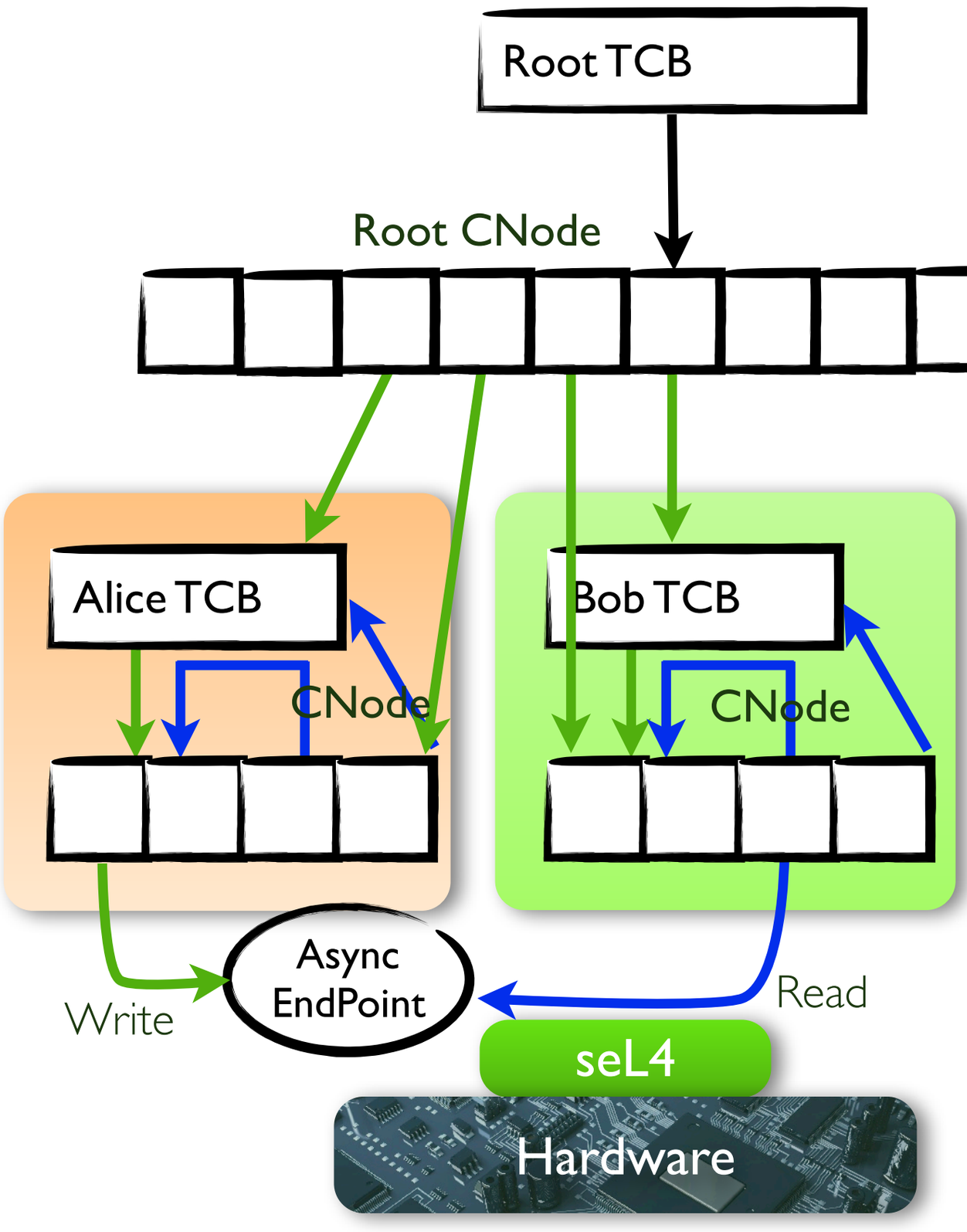
System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space
- Initialise thread control blocks
- Initialise capability address space



System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space
- Initialise thread control blocks
- Initialise capability address space



System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space
- Initialise thread control blocks
- Initialise capability address space
- Start threads

NICTA

No objects

Empty/default objects

Duplicate capabilities to CNodes and TCBs

Virtual memory objects set up correctly

Virtual memory and thread objects set up correctly

All objects set up correctly

System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space
- Initialise thread control blocks
- Initialise capability address space
- Start threads

Objects = Virtual Memory + TCBs + CNodes + Capless

NICTA

No objects

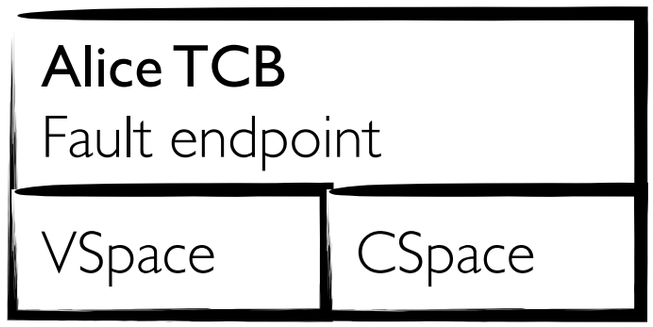
Empty/default objects

Duplicate capabilities to CNodes and TCBs

Virtual memory objects set up correctly

Virtual memory and thread objects set up correctly

All objects set up correctly



System initialisation:

- Create objects
- Duplicate capabilities
- Initialise virtual address space
- Initialise thread control blocks
- Initialise capability address space
- Start threads

Objects = Virtual Memory + TCBs + CNodes + Capless

Separation logic

NICTA

- Setting up capabilities

$$\{\mathrm{slot}_1 \mapsto_c \mathrm{cap} \wedge^* \mathrm{slot}_2 \mapsto_c - \wedge^* R\}$$
$$\quad \mathrm{move\_cap}\ \mathrm{slot}_1\ \mathrm{slot}_2$$
$$\{\mathrm{slot}_1 \mapsto_c \mathrm{NullCap} \wedge^* \mathrm{slot}_2 \mapsto_c \mathrm{cap} \wedge^* R\}$$

- Monadic maps (loops)

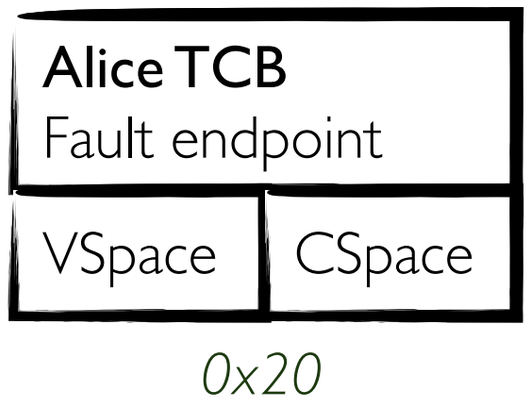
$$\left(\bigwedge R\ x \in xs.\ \{P\ x \wedge^* R\}\ \ f\ x\ \ \{Q\ x \wedge^* R\}\right) \Longrightarrow$$
$$\{\wedge^*\ \mathrm{map}\ P\ xs \wedge^* R\}\ \ \mathrm{mapM}\ f\ xs\ \ \{\wedge^*\ \mathrm{map}\ Q\ xs \wedge^* R\}$$

Separation logic

NICTA

CapDL object heap:

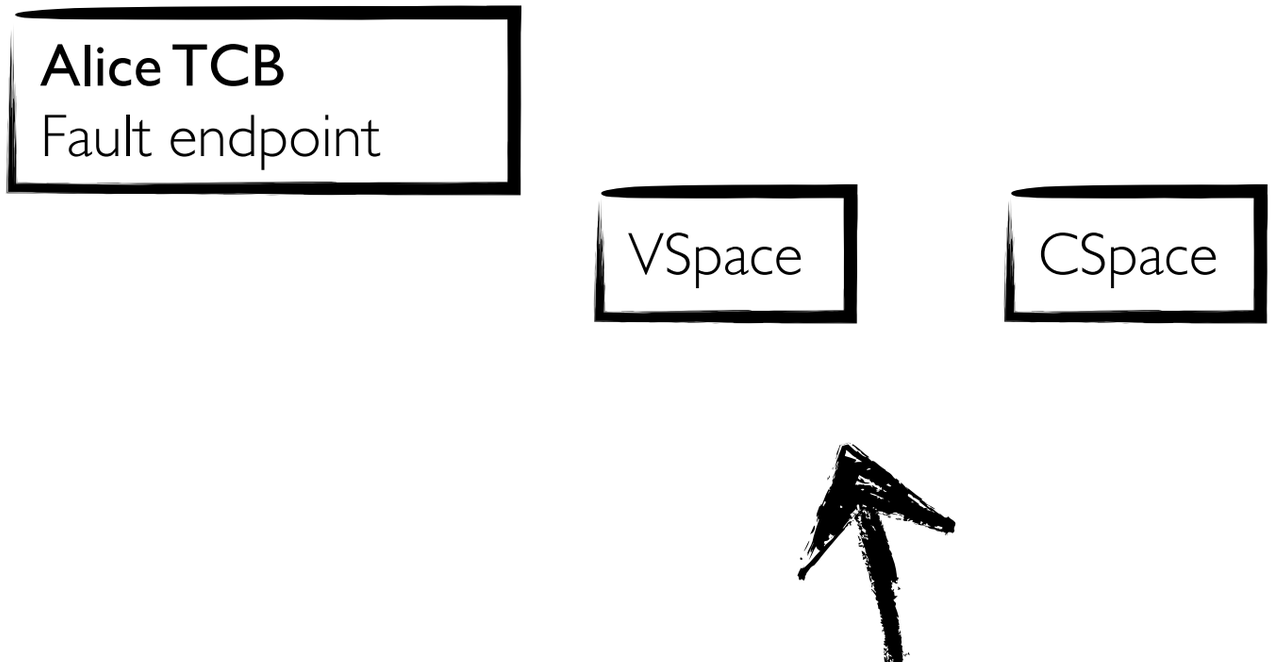
cdl_object_id ⇒ cdl_object option



Separation logic

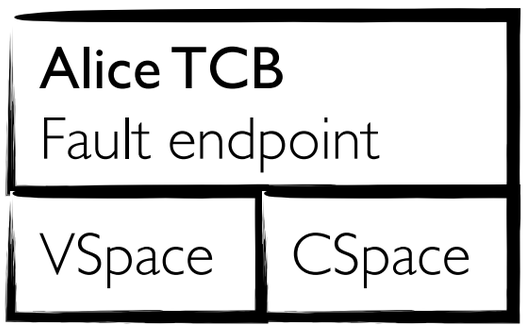
NICTA

Separation heap:



CapDL object heap:

cdl_object_id ⇒ cdl_object option



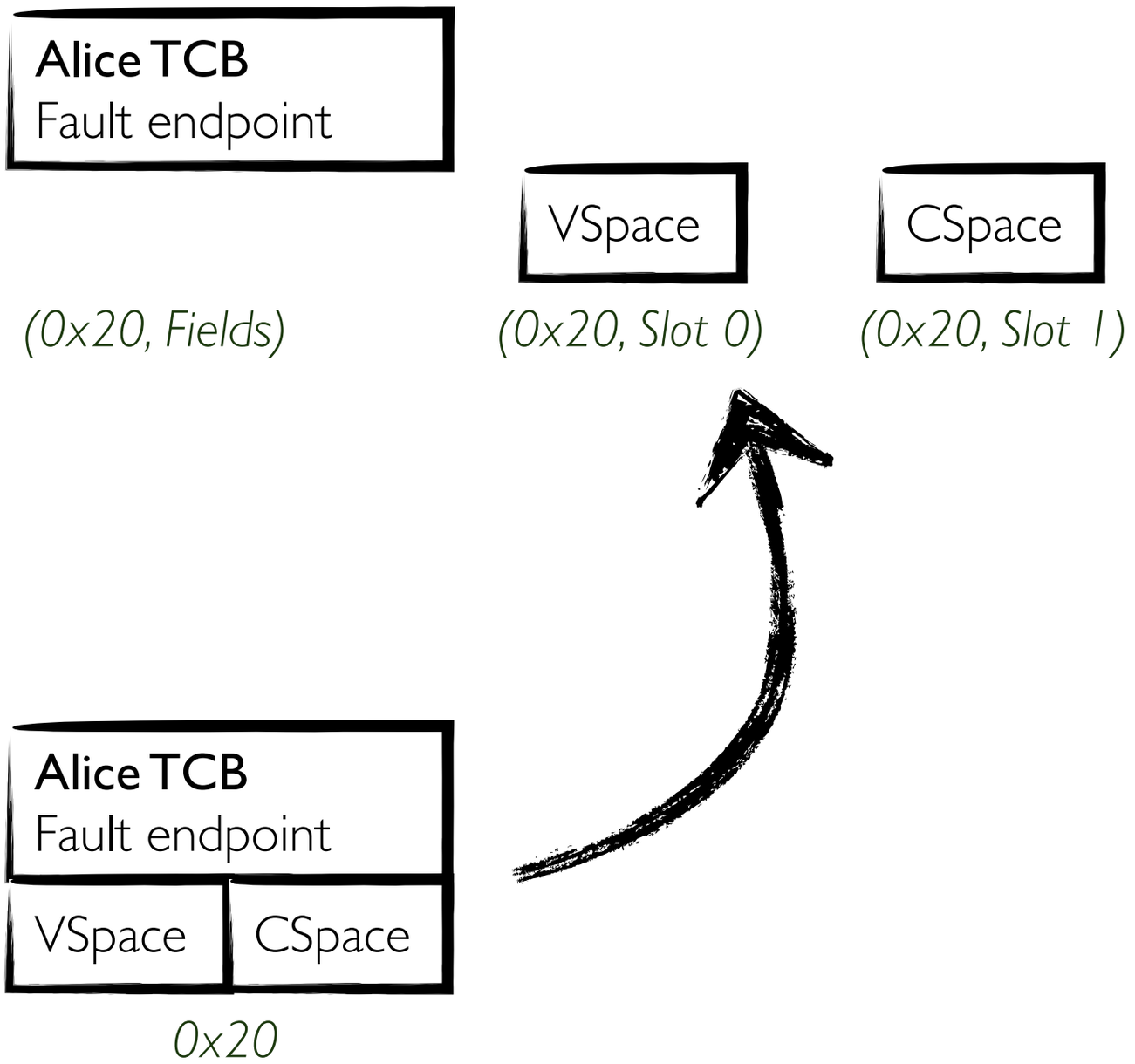
0x20

Separation logic

NICTA

Separation heap:

(cdl_object_id × cdl_component)
⇒ sep_entity option



CapDL object heap:

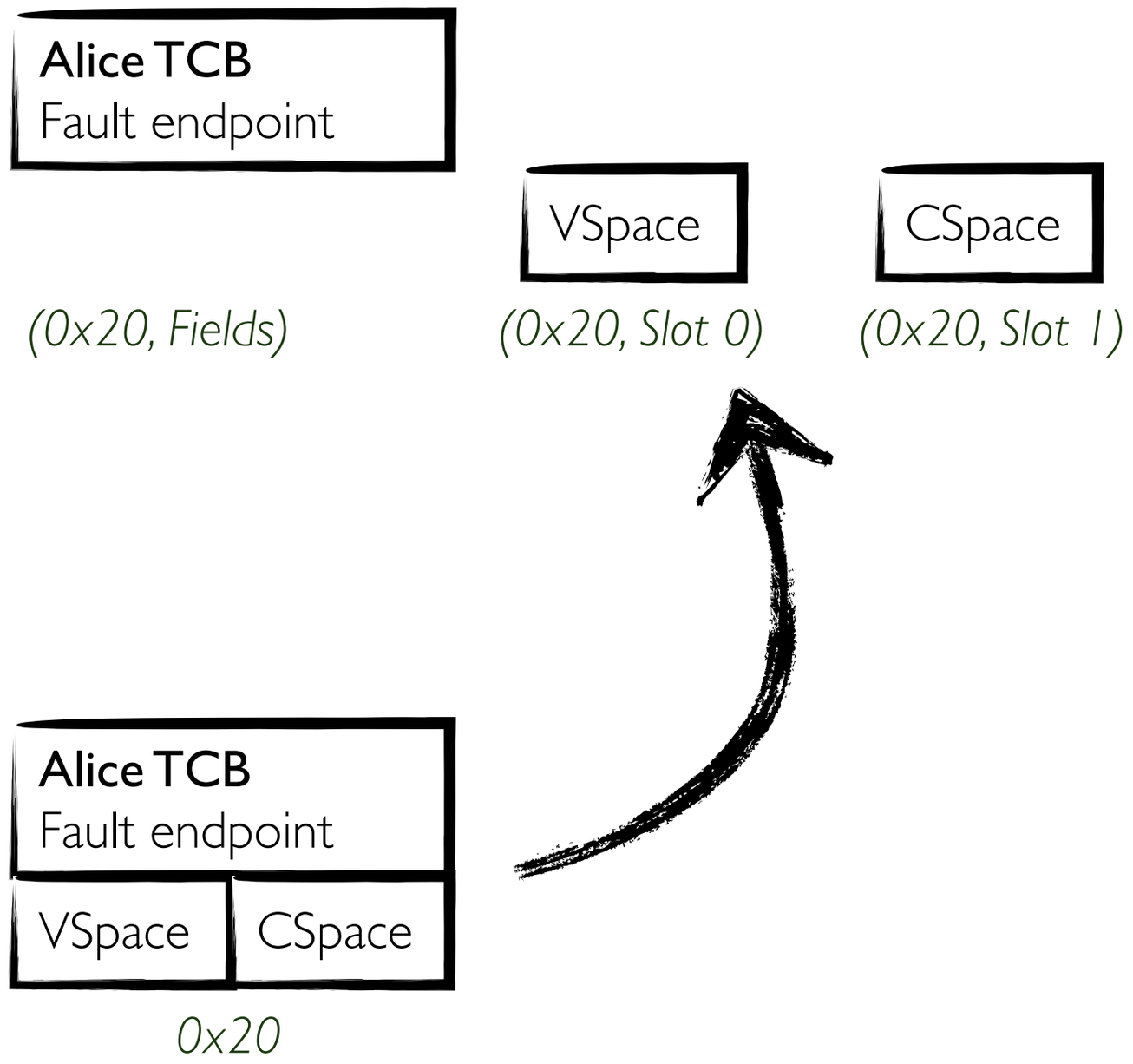
cdl_object_id ⇒ cdl_object option

Separation logic

NICTA

Separation heap:

(cdl_object_id × cdl_component)
⇒ sep_entity option



CapDL object heap:

cdl_object_id ⇒ cdl_object option

Instantiate separation heap in the standard manner

Overall theorem

NICTA

If well_formed spec **and**

obj_ids = dom (cdl_objects spec) **and**

distinct obj_ids **then**

{ «valid_boot_info bootinfo spec » }

init_system spec bootinfo obj_ids

{ $\boldsymbol{\lambda}$s. $\exists\varphi$. « $\bigwedge^*$ map (object_initialised spec φ) obj_ids $\wedge^*$

si_objects spec φ » s $\wedge$

injective φ $\wedge$ dom φ = obj_ids }

Proof decomposition

NICTA

⟦dom (slots_of obj_id spec) = slots; distinct slots⟧

$\Longrightarrow$ object_initialised spec φ obj_id =

(object_fields_ initialised spec φ obj_id $\wedge^*$

$\wedge^*$ map (object_slot_initialised spec φ obj_id) slots $\wedge^*$

object_empty_slots_initialised spec φ obj_id)

Proof decomposition

NICTA

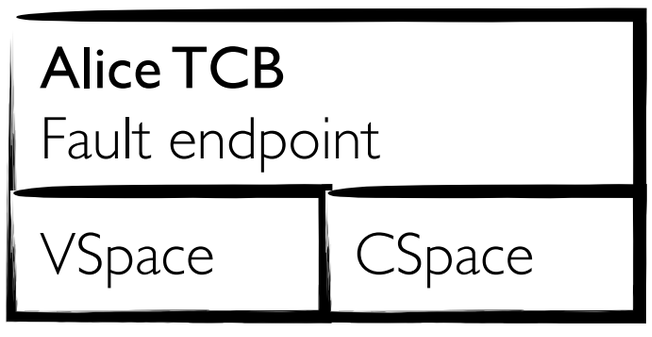
$[\![$dom (slots_of obj_id spec) = slots; distinct slots$]\!]$

$\Longrightarrow$ object_initialised spec φ obj_id =

(object_fields_ initialised spec φ obj_id $\wedge^*$

$\wedge^*$ map (object_slot_initialised spec φ obj_id) slots $\wedge^*$

object_empty_slots_initialised spec φ obj_id)



Proof decomposition

NICTA

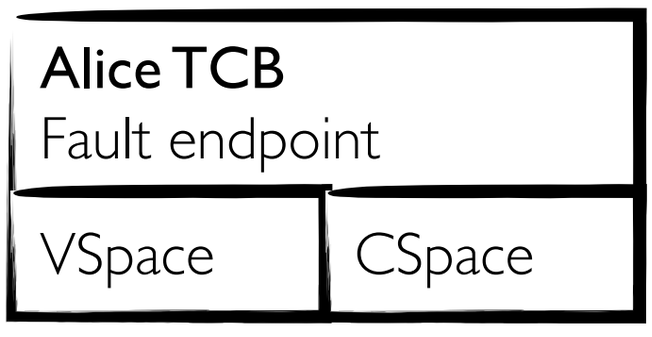
$[\![$dom (slots_of obj_id spec) = slots; distinct slots$]\!]$

$\Longrightarrow$ object_initialised spec φ obj_id =

(object_fields_ initialised spec φ obj_id $\wedge^*$

$\wedge^*$ map (object_slot_initialised spec φ obj_id) slots $\wedge^*$

object_empty_slots_initialised spec φ obj_id)



Connect to seL4 kernel API specifications we developed

Future work

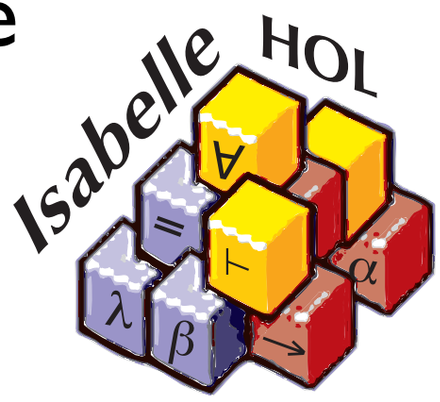
NICTA

- Assumptions/limitations
  - Scheduling
  - Don't support all capability types
- Refinement proof of C implementation

Summary

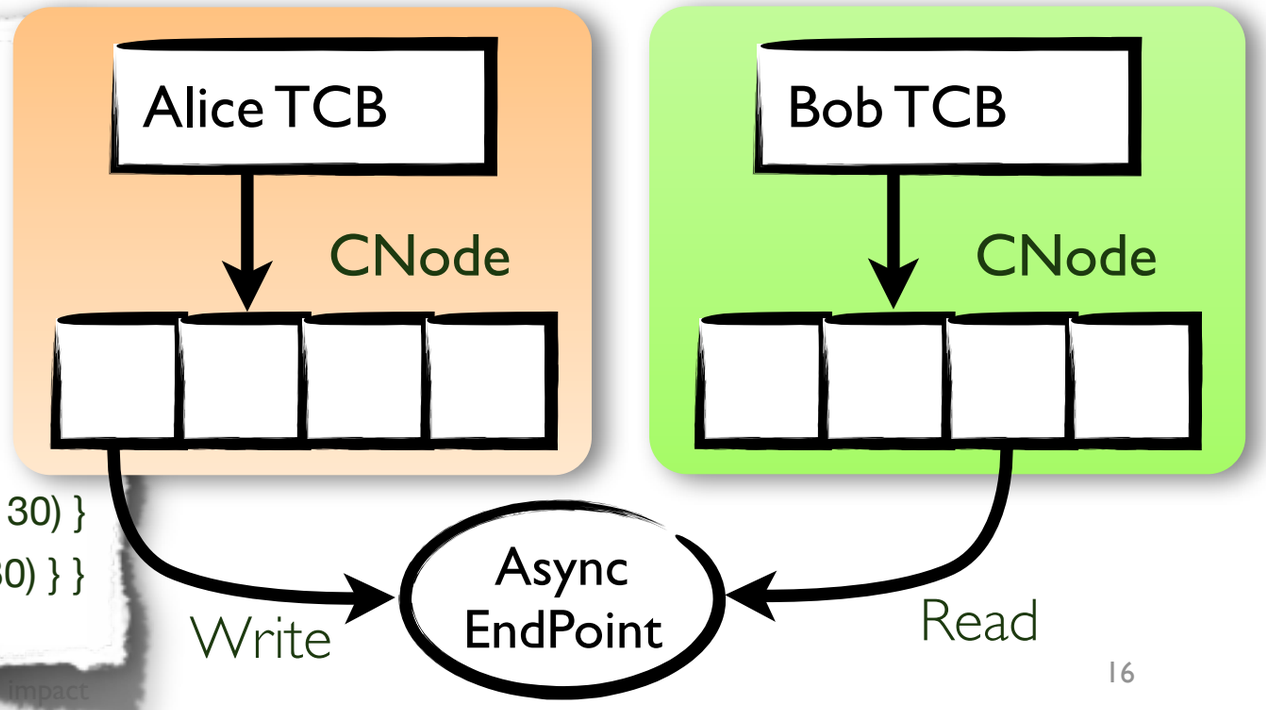
NICTA

- Verified formal algorithm for automatically initialising capability based systems with high-assurance
- Proven in Isabelle/HOL
  - System initialiser specification ~ 400 lines
  - Kernel API specification proofs ~ 10,500 lines
  - Proof of system initialiser specification ~ 8,100 lines



- Linked capDL descriptions to access control policy

```
objects { cnode_alice = cnode (2 bits)
          tcb_alice = tcb ()
          cnode_bob = cnode (2 bits)
          tcb_bob = tcb ()
          aep_shared = aep }
caps { cnode_alice {   0x0: aep_shared (W) }
       cnode_bob { 0x2: aep_shared (R) }
       tcb_alice { cspace: cnode_alice (guard: 0, guard_size: 30) }
       tcb_bob { cspace: cnode_bob (guard: 0, guard_size: 30) } }
```



NICTA

Questions?

Formally Verified System Initialisation

Andrew Boyton

June Andronick, Callum Bannister,
Matthew Fernandez, Xin Gao,
David Greenaway, Gerwin Klein,
Corey Lewis and Thomas Sewell