A Verified OS Kernel. Now What?

Gerwin Klein

Australian Government
Department of Communications, Information Technology and the Arts
Australian Research Council

NICTA Members
NICTA Partners

The Team

The Team

















L4Verified

L4.Verified

l microkernel

8,700 lines of C

0 bugs*

qed

*conditions apply

Windows

An exception 06 has occured at 0028:C11B3ADC in VxD DiskTSD(03) +
00001660. This was called from 0028:C11B40C8 in VxD voltrack(04) +
00000000. It may be possible to continue normally.

* Press any key to attempt to continue.
* Press CTRL+ALT+RESET to restart your computer. You will
  lose any unsaved information in all applications.

Press any key to continue

Windows Vista™

Stunning. Breakthrough. Entertaining.

HP TouchSmart PC and Microsoft Windows Vista deliver you a PC experience designed to fit wherever life happens.

Innovative solutions brought to you by: Microsoft hp BEST BUY


the Bay
the Bay

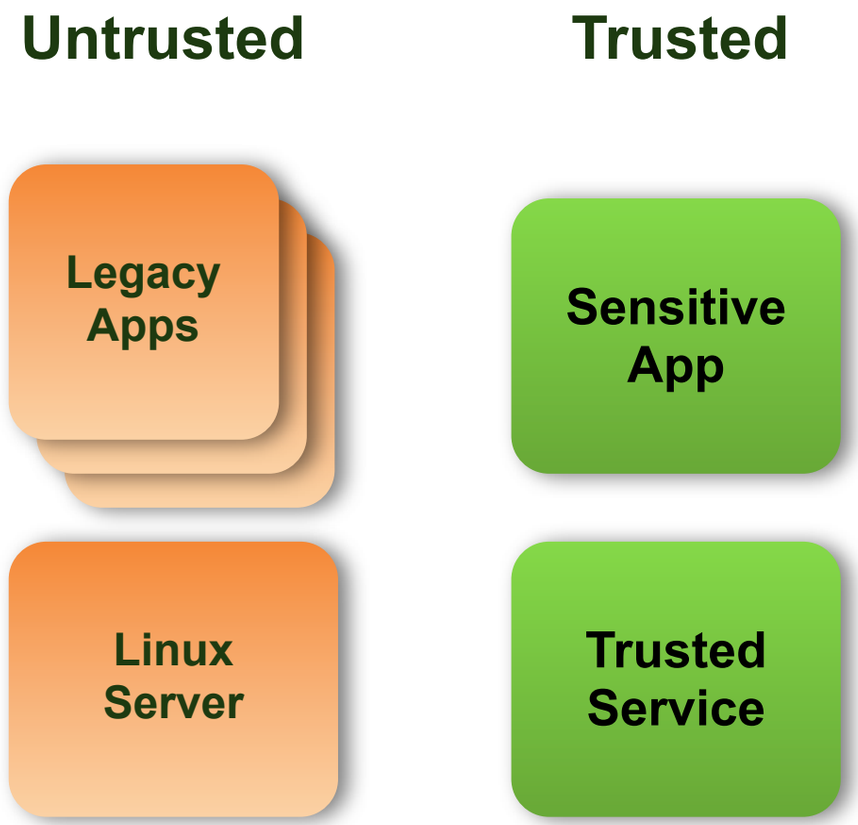
The Problem

Small Kernels

NICTA

Small trustworthy foundation

- hypervisor, microkernel, nano-kernel, virtual machine, separation kernel, exokernel ...
- High assurance components in presence of other components



seL4 API:

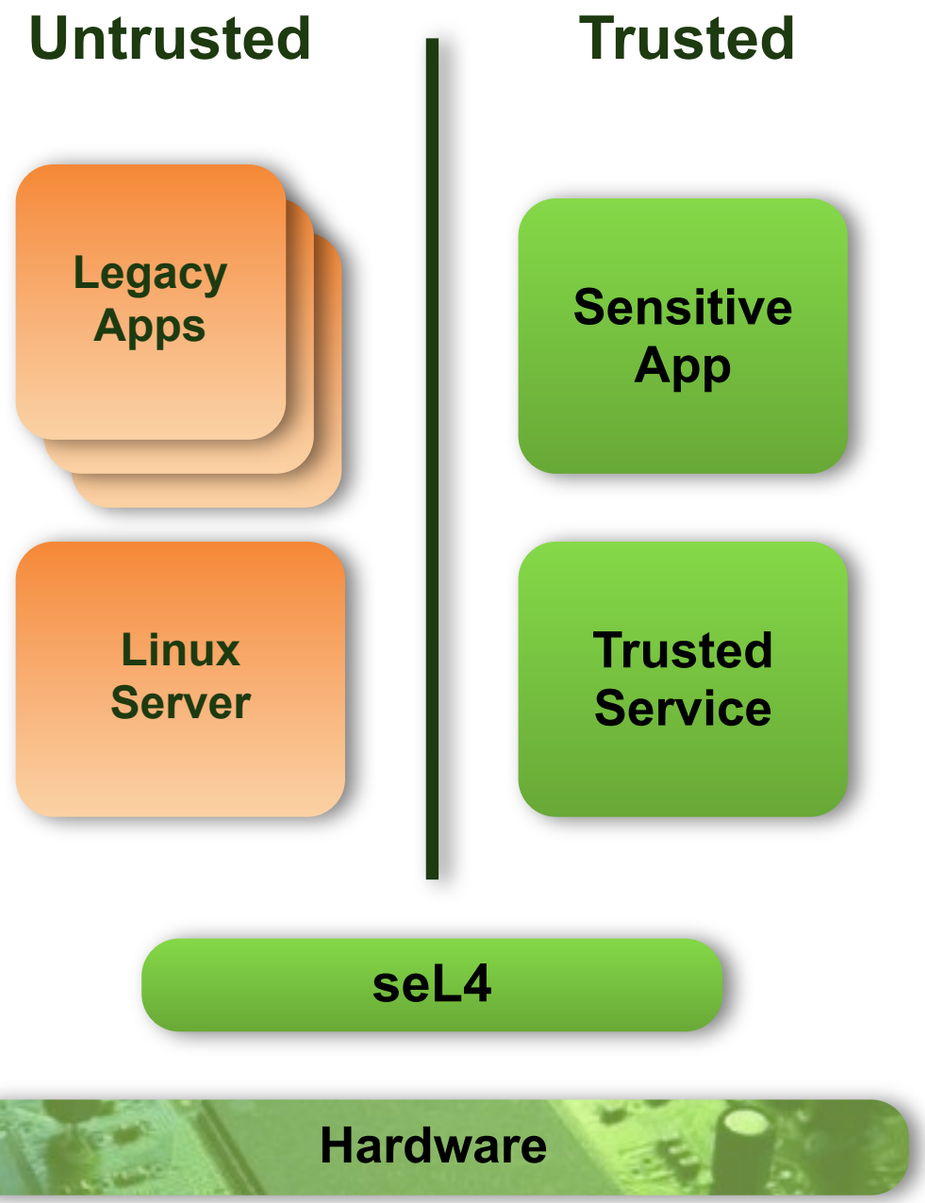
- IPC
- Threads
- VM
- IRQ
- Capabilities

Small Kernels

NICTA

Small trustworthy foundation

- hypervisor, microkernel, nano-kernel, virtual machine, separation kernel, exokernel ...
- High assurance components in presence of other components



seL4 API:

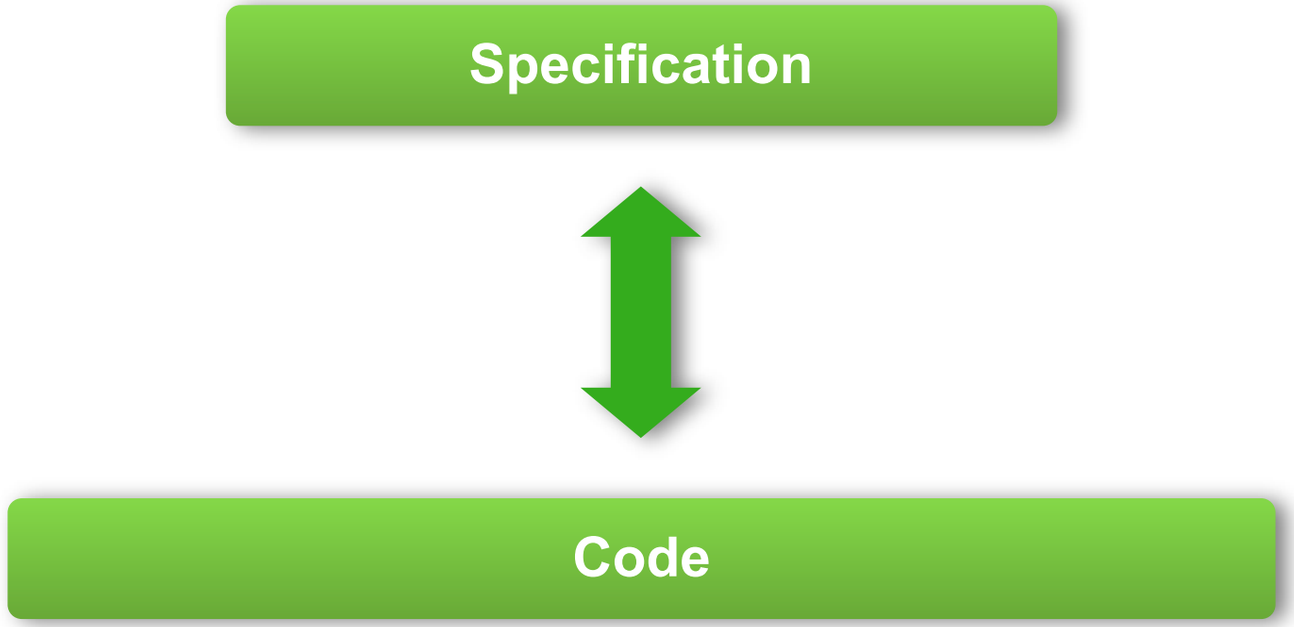
- IPC
- Threads
- VM
- IRQ
- Capabilities

The Proof

The Proof

Functional Correctness

Proof



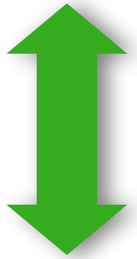
Functional Correctness

What

Proof

Specification

```
definition
  schedule :: unit s_monad where
  schedule ≡ do
    threads ← allActiveTCBs;
    thread ← select threads;
    switch_to_thread thread
  od
  OR switch_to_idle_thread
```



Code

Functional Correctness

NICTA

What

Proof

How

Specification

```
definition
  schedule :: unit s_monad where
  schedule ≡ do
    threads ← allActiveTCBs;
    thread ← select threads;
    switch_to_thread thread
  od
  OR switch_to_idle_thread
```

Implementation

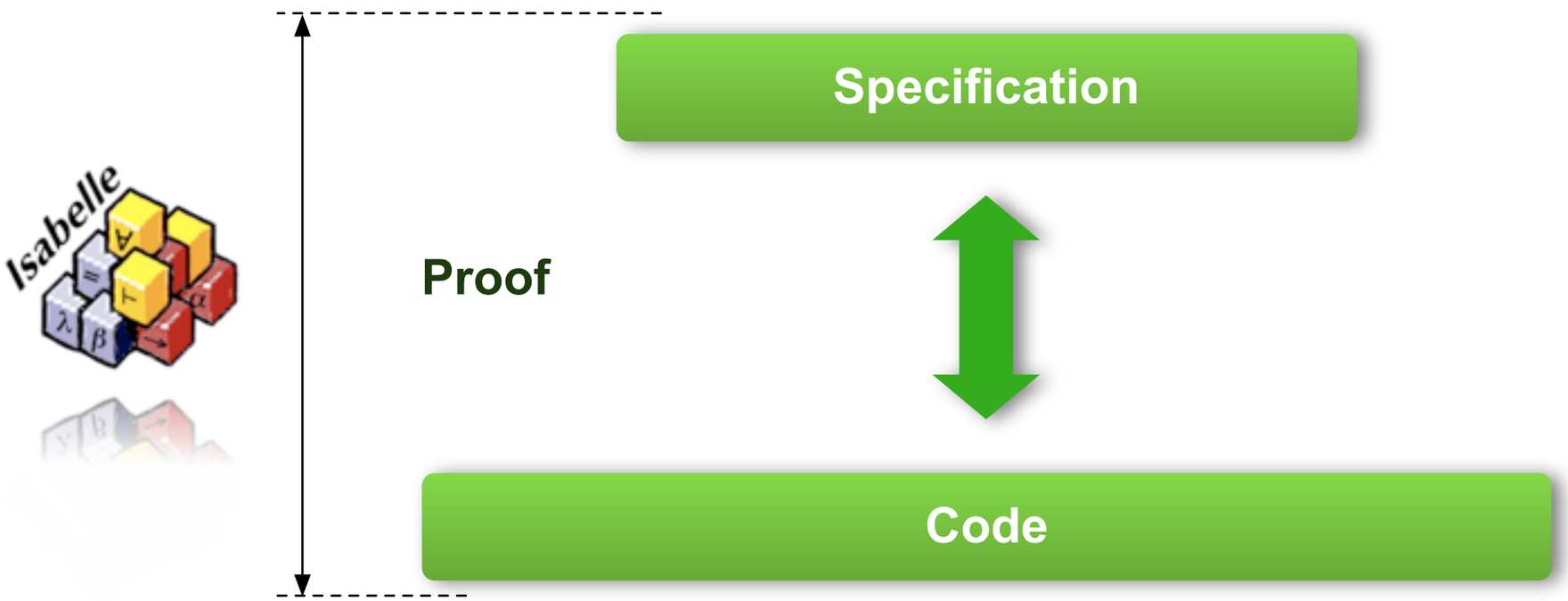
```
void
schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case (word_t)SchedulerAction_ResumeCurrentThread:
            break;

        case (word_t)SchedulerAction_ChooseNewThread:
            chooseThread();
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;

        default: /* SwitchToThread */
            switchToThread(ksSchedulerAction);
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;
    }
}

void
chooseThread(void) {
    prio_t prio;
    tcb_t *thread, *next;
```

*conditions apply



*conditions apply

NICTA

Expectation

Specification

Proof

Code

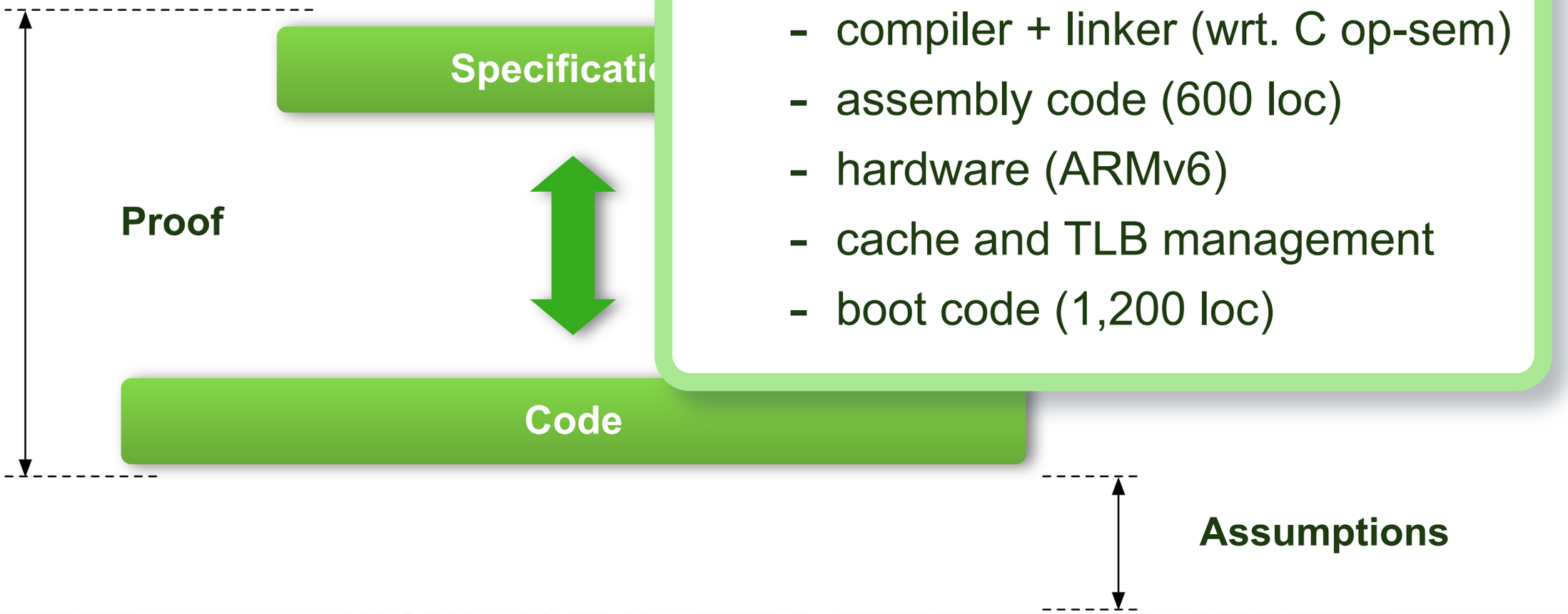
Assumptions

*conditions apply

NICTA

Assume correct:

- compiler + linker (wrt. C op-sem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

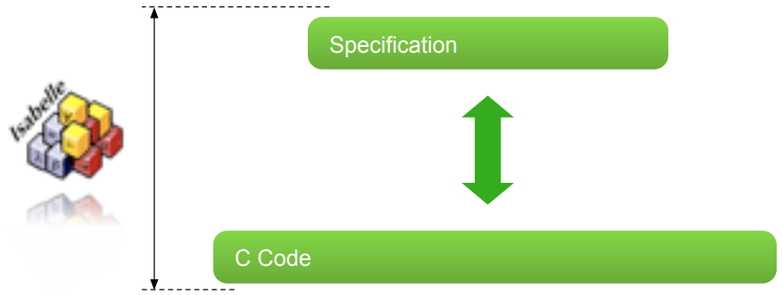


Implications

NICTA

Execution always defined:

- no null pointer de-reference
- no buffer overflows
- no code injection
- no memory leaks/out of kernel memory
- no div by zero, no undefined shift
- no undefined execution
- no infinite loops/recursion



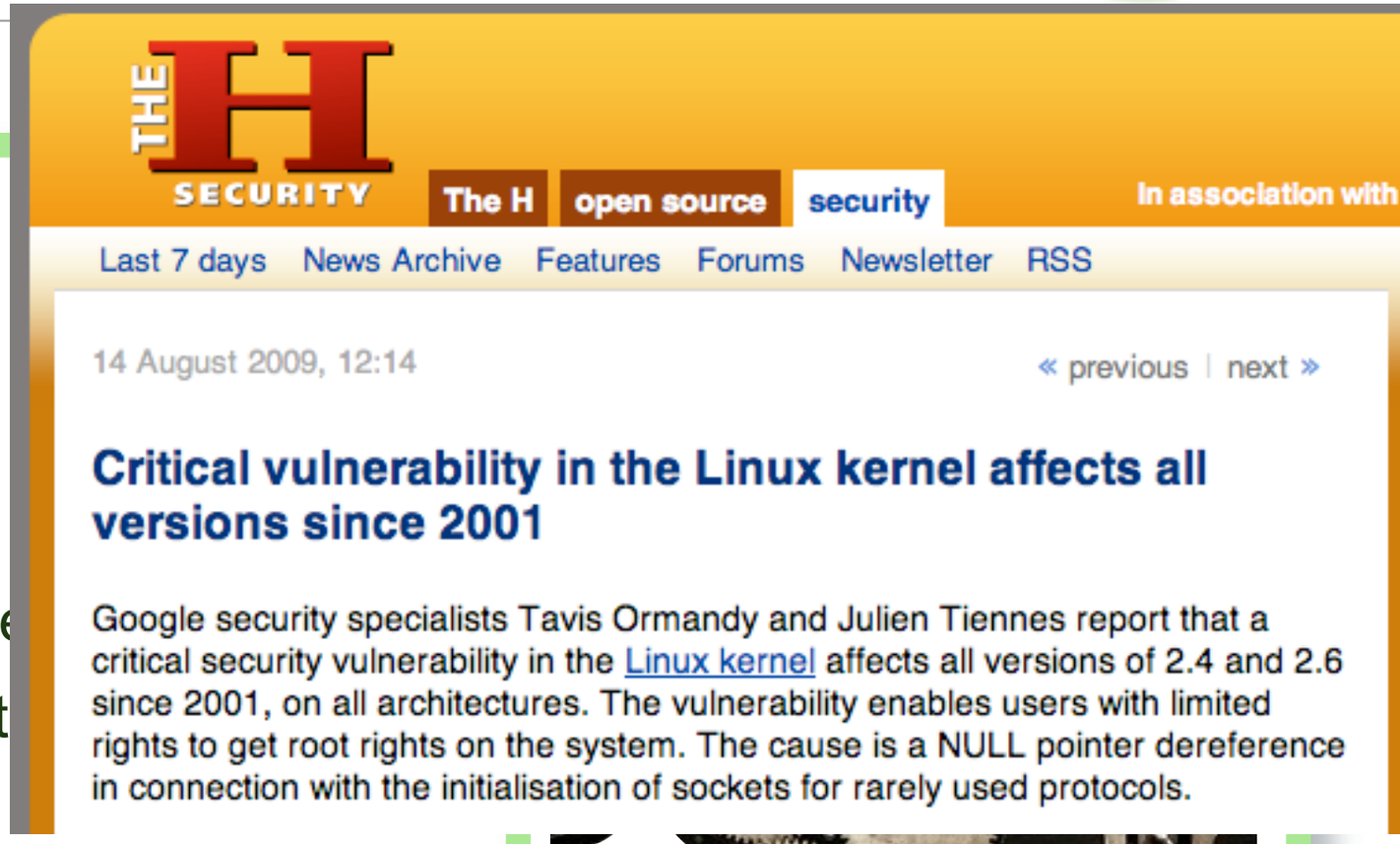
Not implied:

- “secure” (define secure)
- zero bugs from expectation to physical world
- covert channel analysis

Implications

Execution always defined:

- no null pointer de-reference
- no buffer overflows
- no code injection
- no memory leaks/out of kernel me
- no div by zero, no undefined shift
- no undefined execution
- no infinite loops/recursion



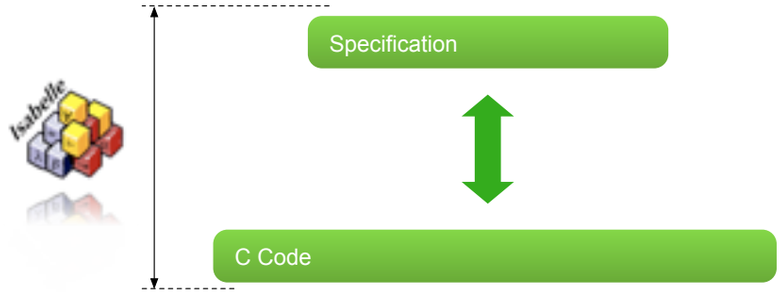
Not implied:

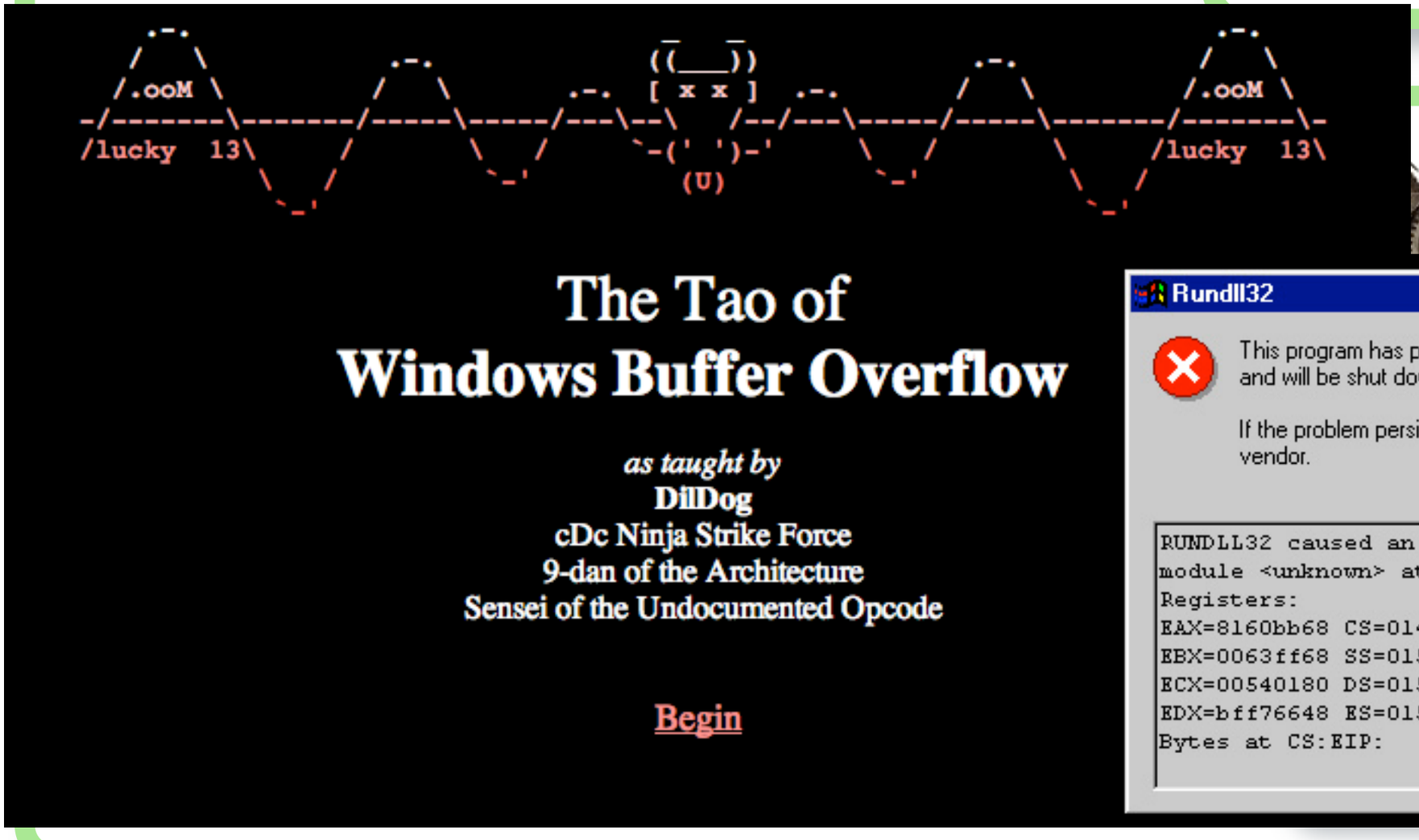
- “secure” (define secure)
- zero bugs from expectation to physical world
- covert channel analysis

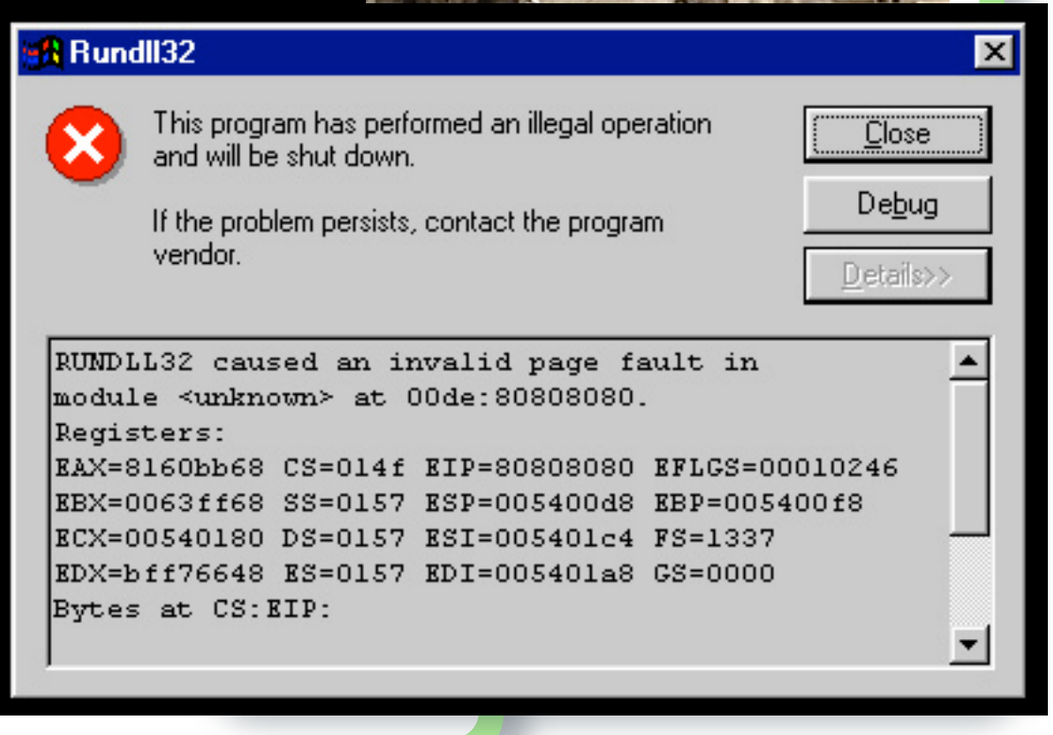
Implications

Execution always defined:

- no null pointer de-reference
- no buffer overflows





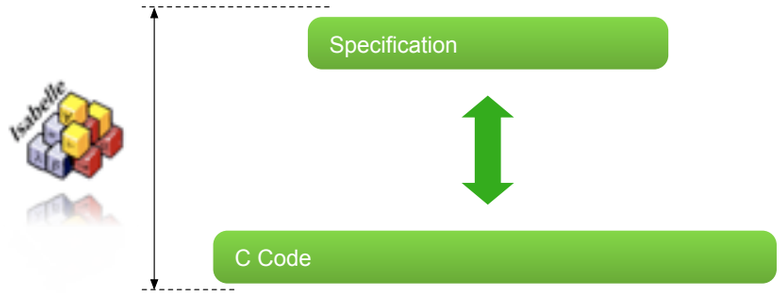


Implications

NICTA

Execution always defined:

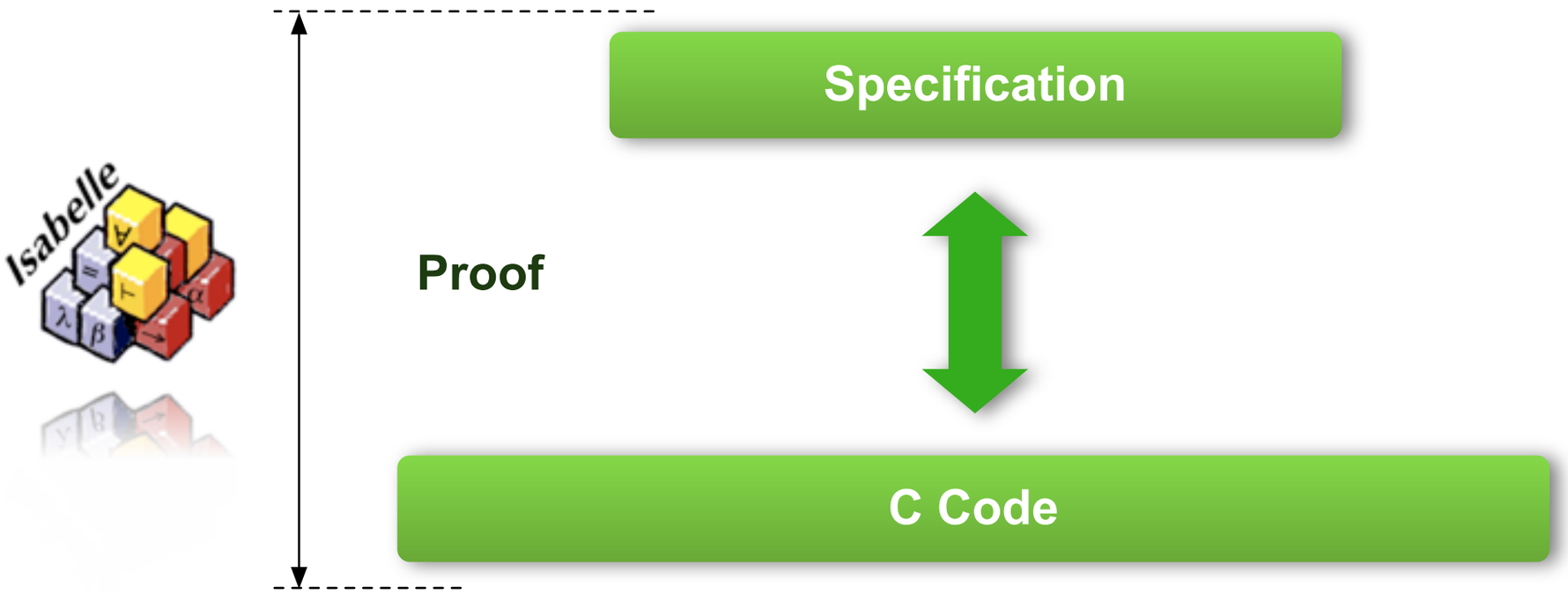
- no null pointer de-reference
- no buffer overflows
- no code injection
- no memory leaks/out of kernel memory
- no div by zero, no undefined shift
- no undefined execution
- no infinite loops/recursion



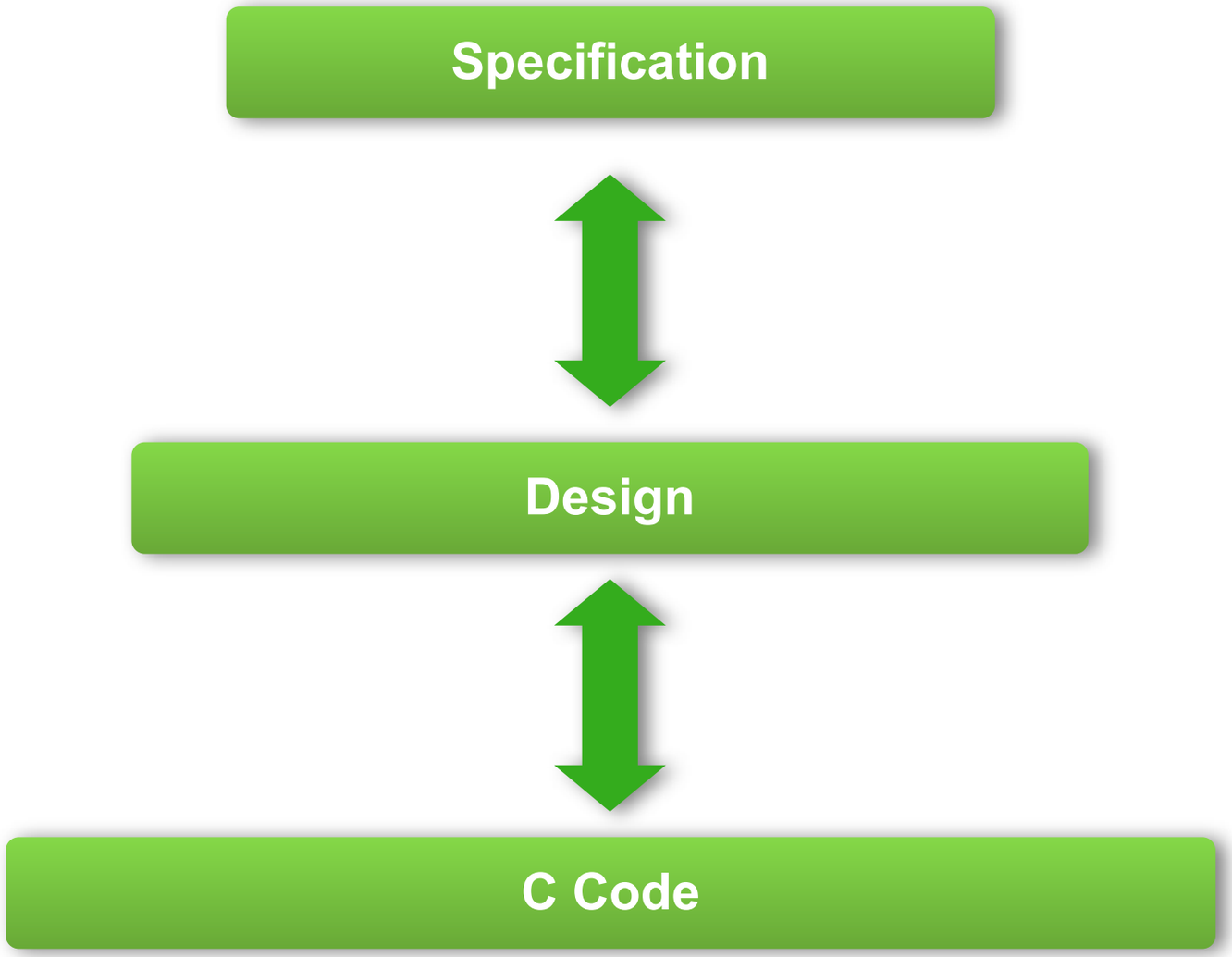
Not implied:

- “secure” (define secure)
- zero bugs from expectation to physical world
- covert channel analysis

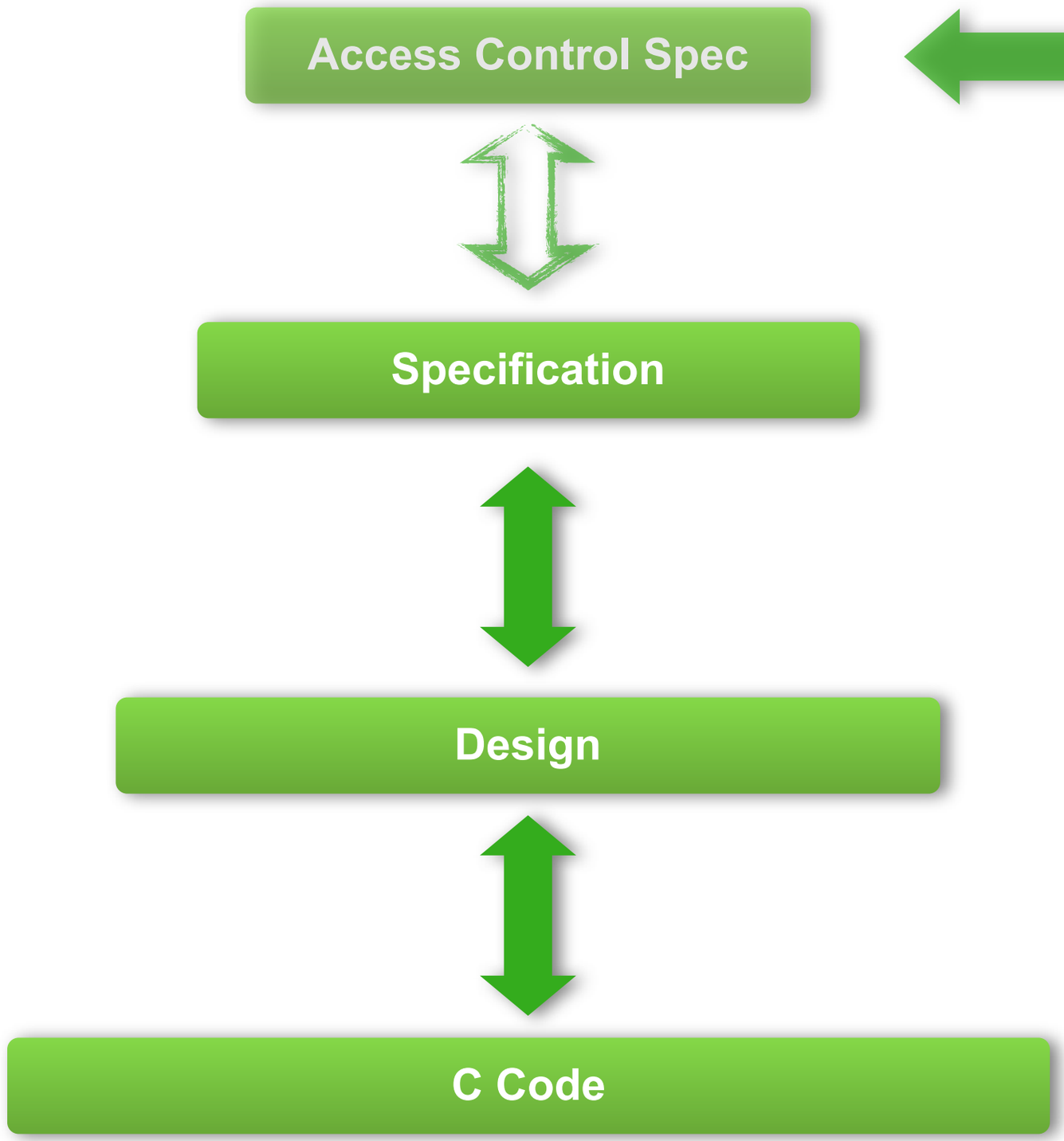
Proof Architecture



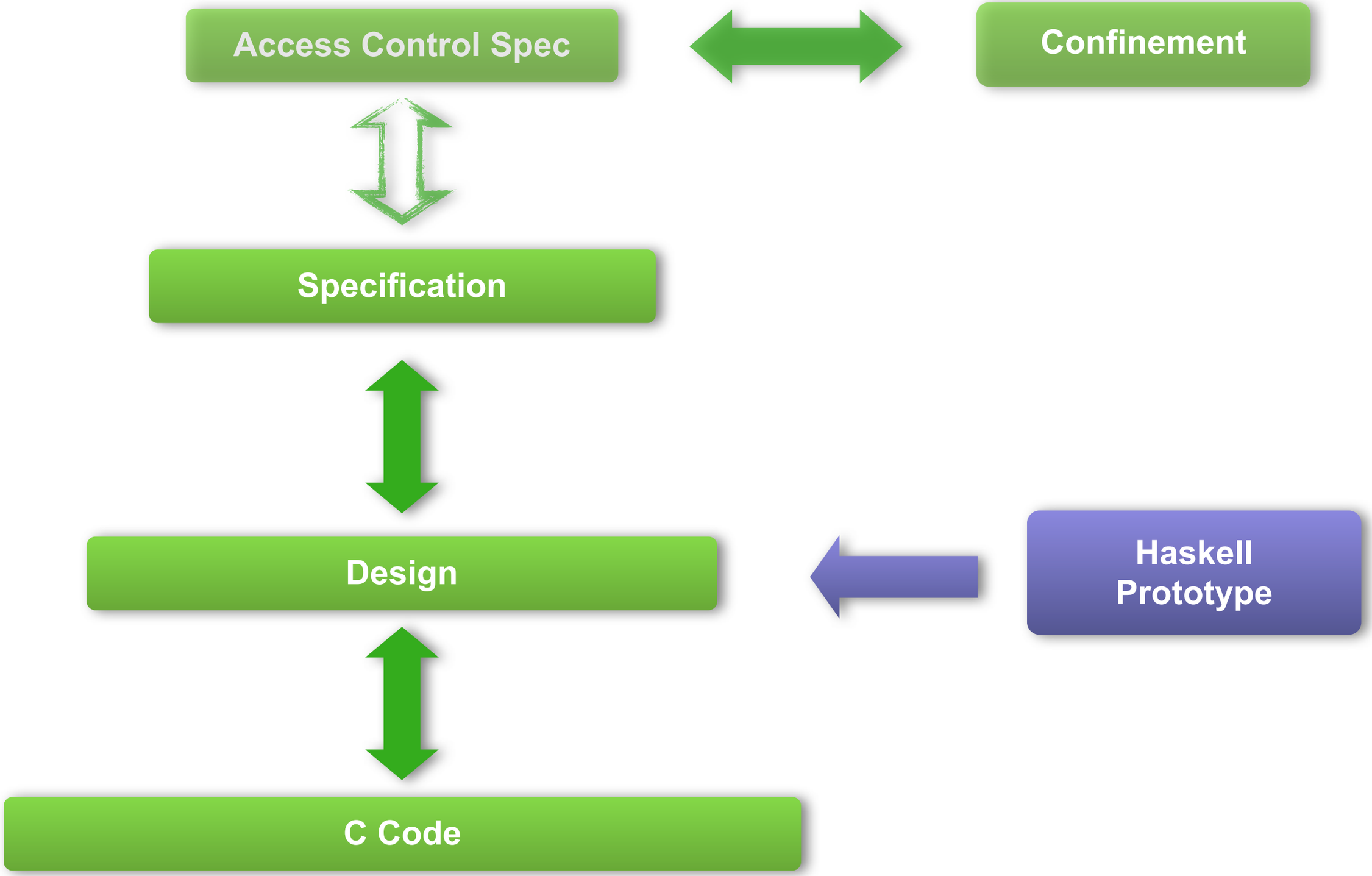
Proof Architecture



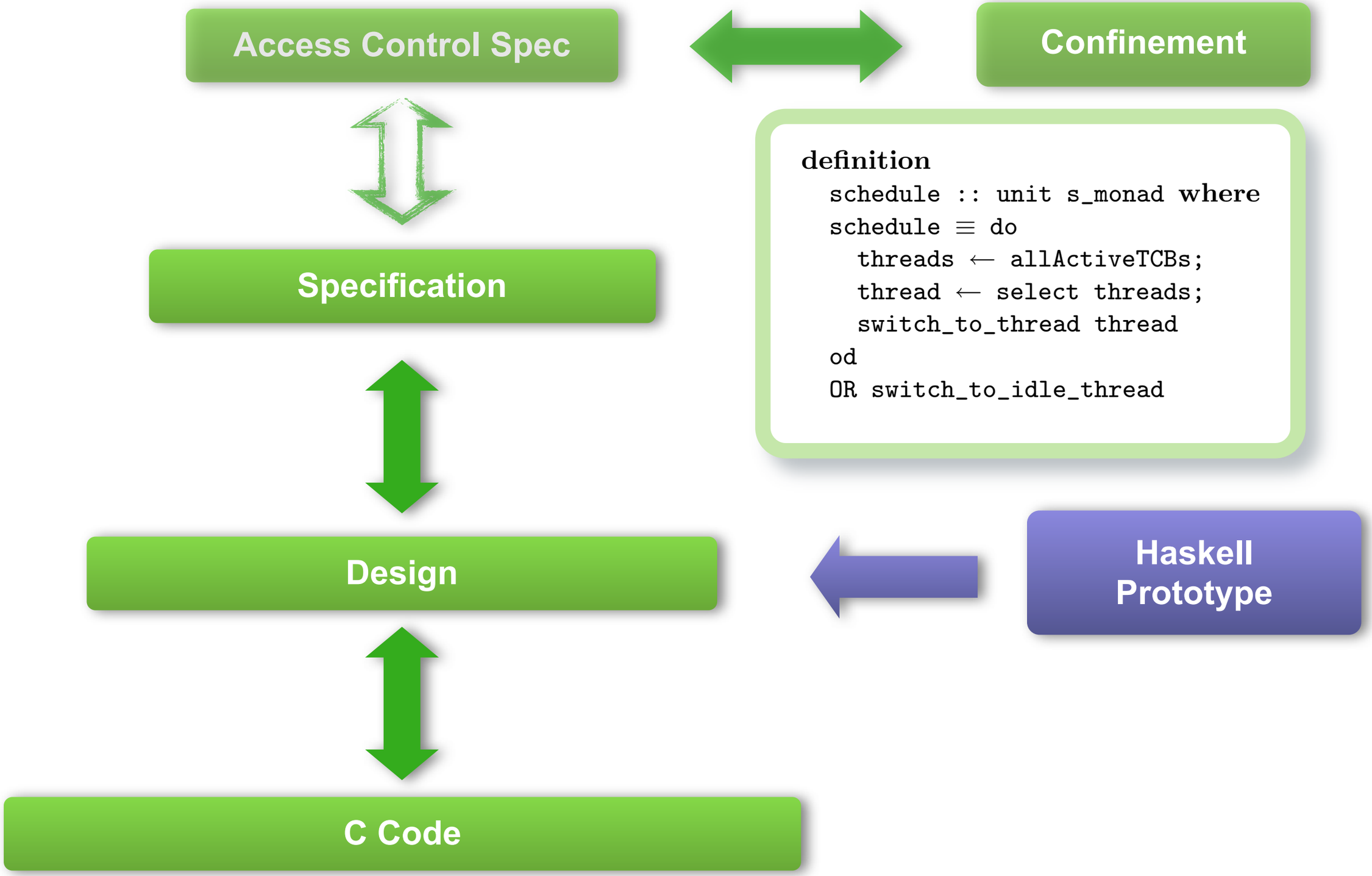
Proof Architecture



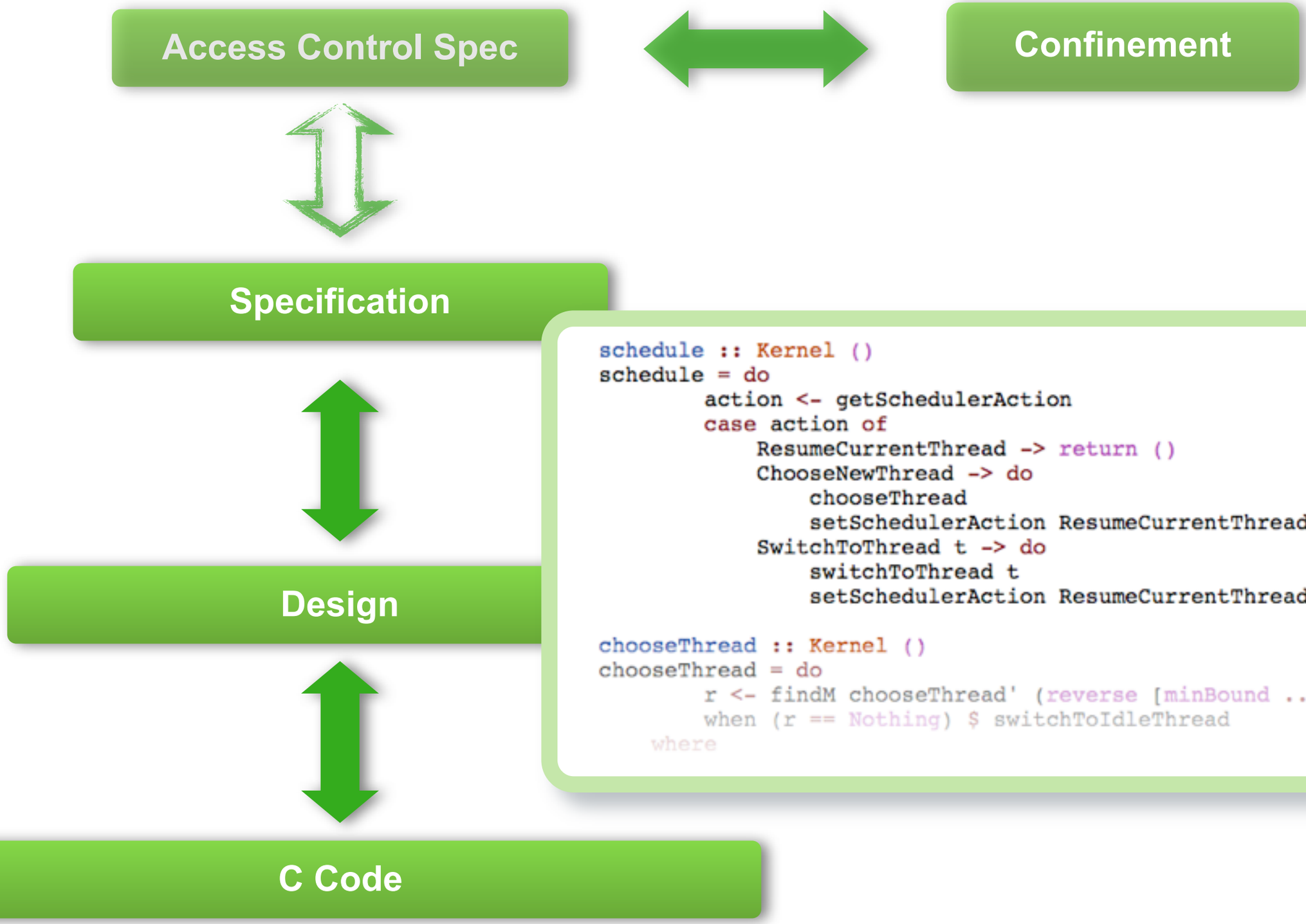
Proof Architecture



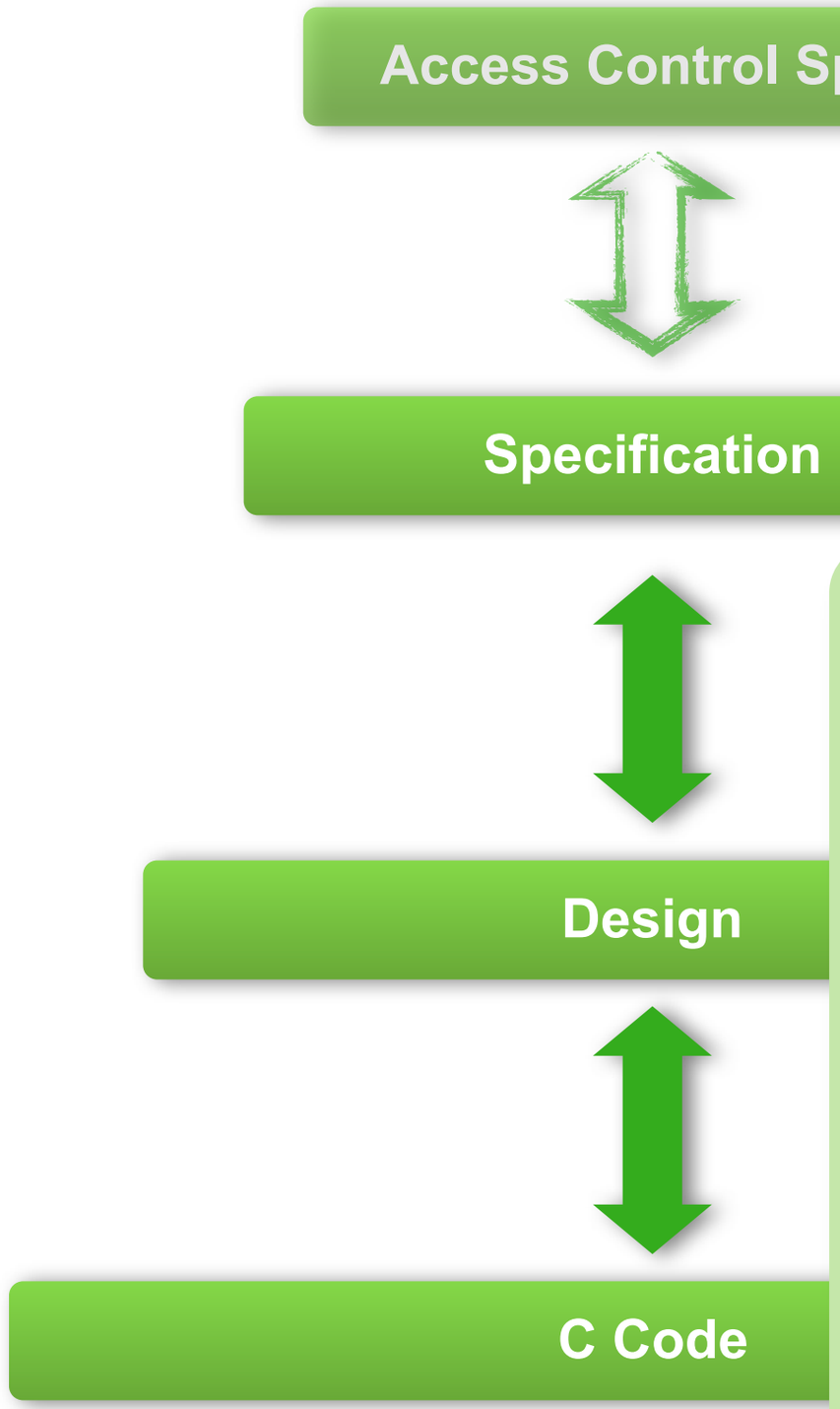
Proof Architecture



Proof Architecture



Proof Architecture

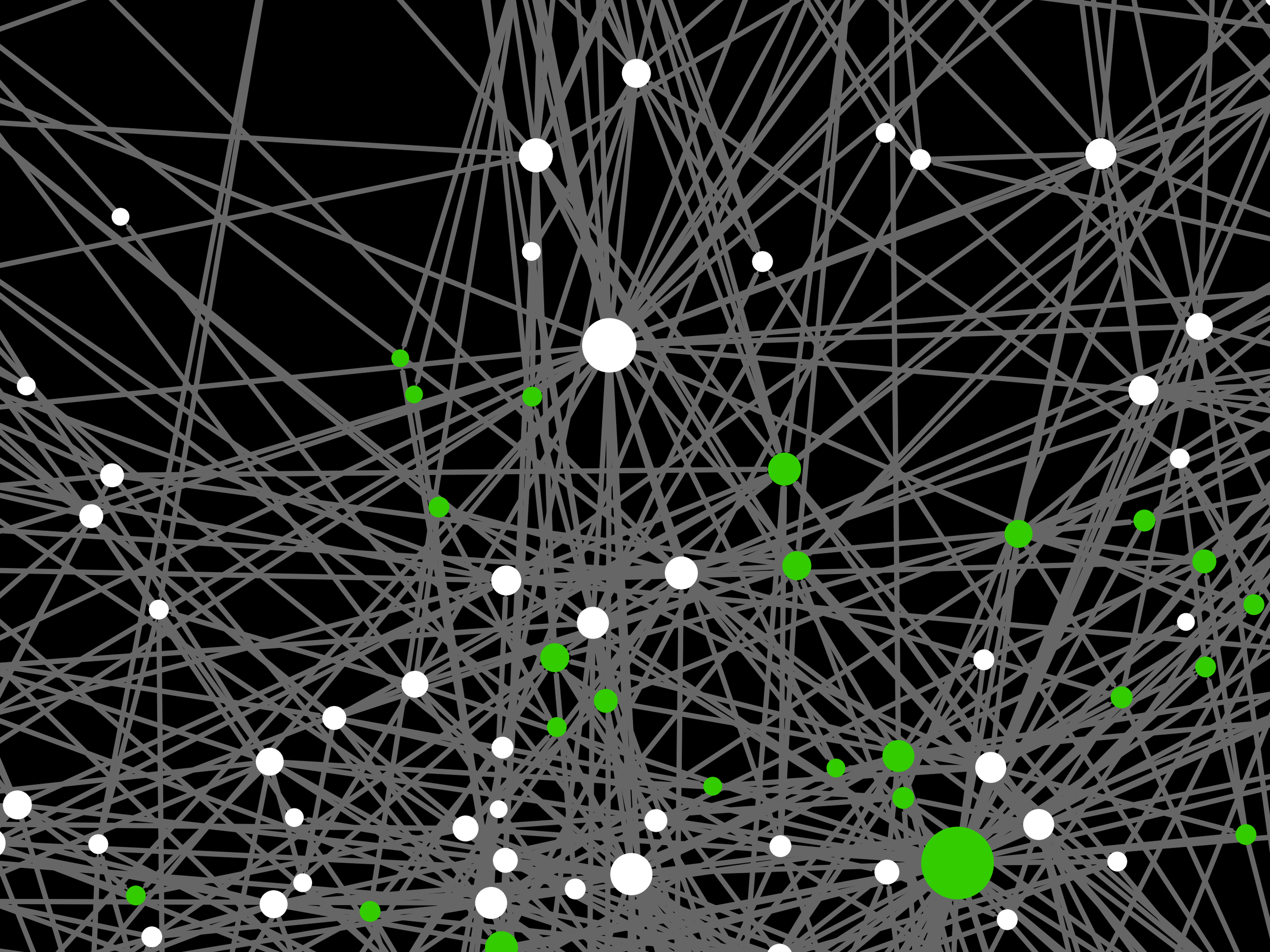



```
void
schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case (word_t)SchedulerAction_ResumeCurrentThread:
            break;

        case (word_t)SchedulerAction_ChooseNewThread:
            chooseThread();
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;

        default: /* SwitchToThread */
            switchToThread(ksSchedulerAction);
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;
    }
}

void
chooseThread(void) {
    prio_t prio;
    tcb_t *thread, *next;
```




seL4

Did you find any Bugs?

NICTA

Bugs found

- in C: 160
- in design: ~150
- in spec: ~150

460 bugs

```
void
schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case (word_t)SchedulerAction_ResumeCurrentThread:
            break;

        case (word_t)SchedulerAction_ChooseNewThread:
            chooseThread();
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;

        d

    }
}

void
chooseThr
    prio_
    tcb_t

    for(p
        f
                if(!isRunnable(thread)) {
                    next = thread->tcbSchedNext;
                    tcbSchedDequeue(thread);
                }
                else {
                    switchToThread(thread);
                    return;
                }
            }
        }
```

ead;

Effort

Haskell design	2	py
First C impl.	2	weeks
Debugging/Testing	2	months
Kernel verification	12	py
Formal frameworks	10	py
Total	25	py

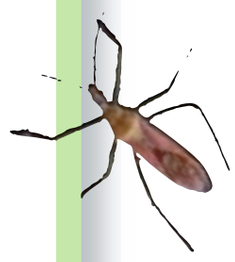
Did you find any Bugs?

NICTA

Bugs found

- in C: 160
- in design: ~150
- in spec: ~150

460 bugs



```
void
schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case (word_t)SchedulerAction_ResumeCurrentThread:
            break;

        case (word_t)SchedulerAction_ChooseNewThread:
            chooseThread();
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;
    }
}

void
chooseThr
    prio_
    tcb_t

    for(p
        f
                if(!isRunnable(thread)) {
                    next = thread->tcbSchedNext;
                    tcbSchedDequeue(thread);
                }
                else {
                    switchToThread(thread);
                    return;
                }
            }
        }
```

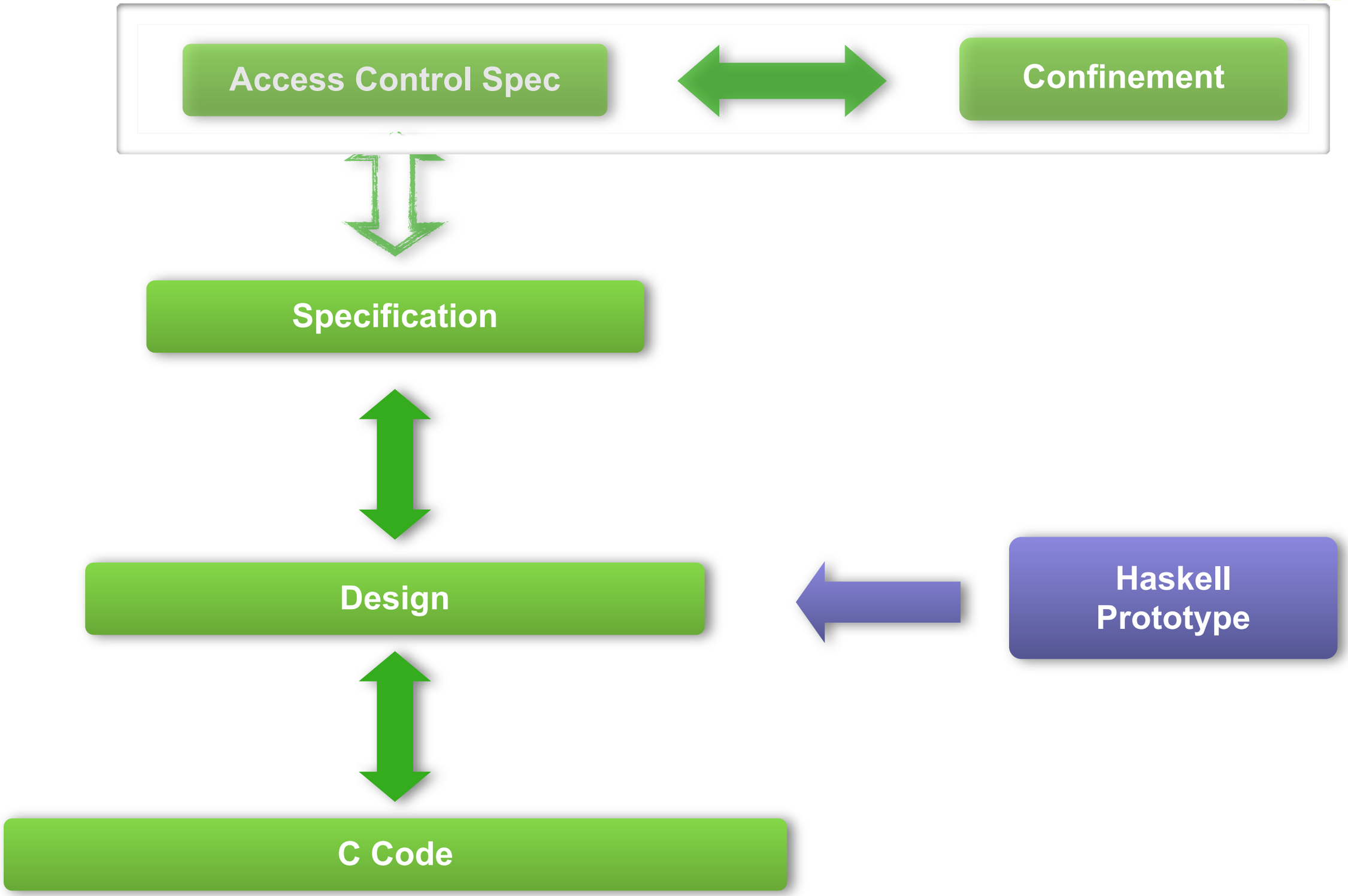
Effort

Haskell design	2	py
First C impl.	2	weeks
Debugging/Testing	2	months
Kernel verification	12	py
Formal frameworks	10	py
Total	25	py

Access Control

Access Control

Proof Architecture



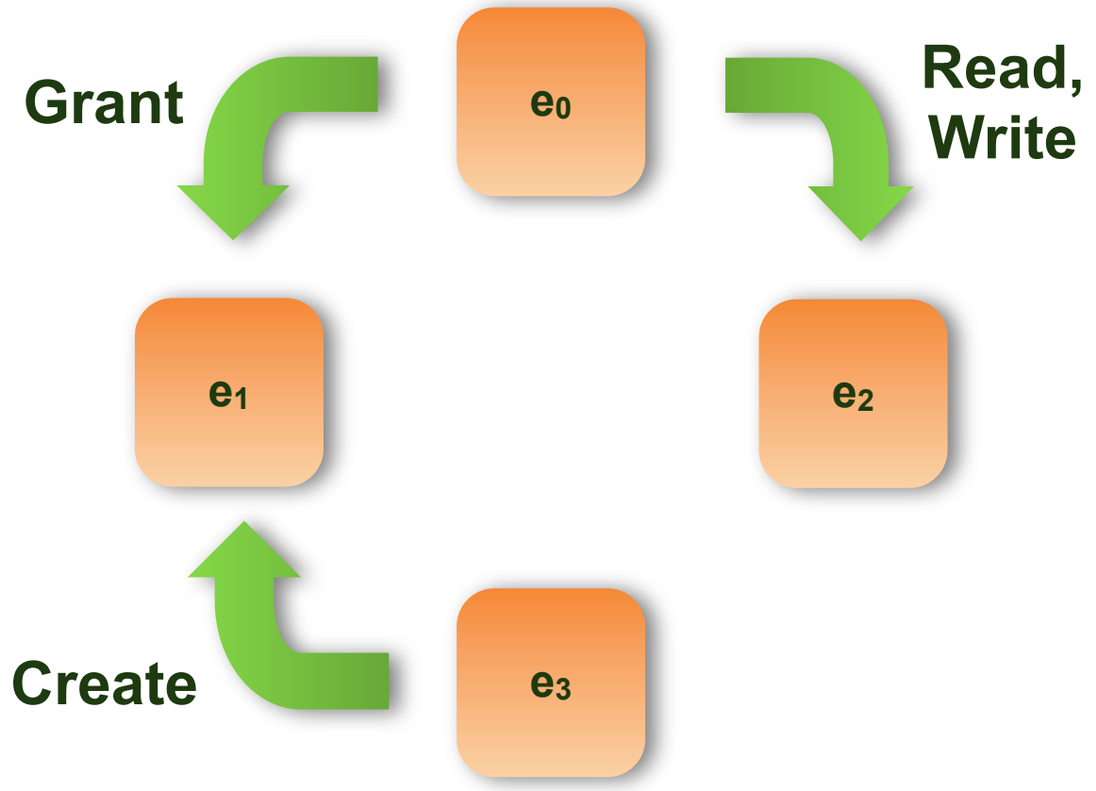
Take-Grant model

Lipton and Snyder:

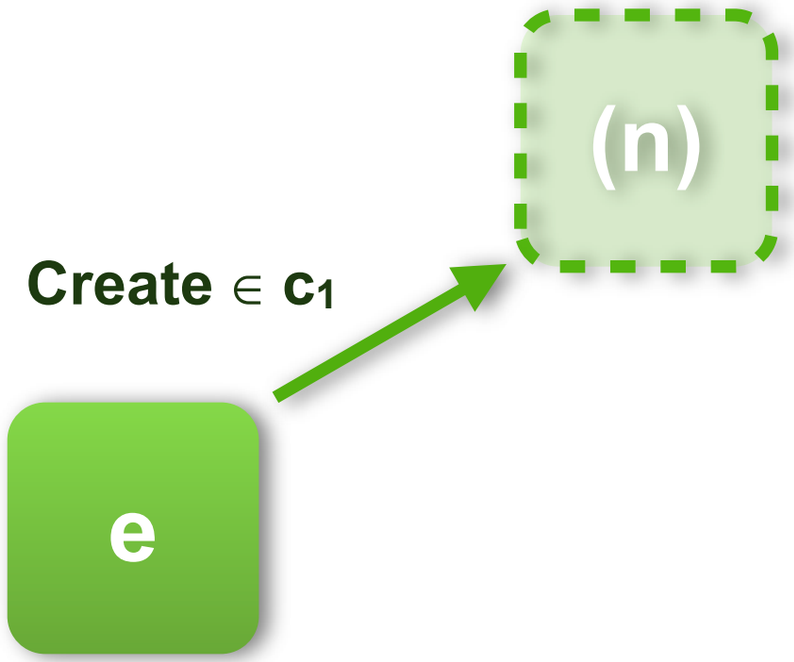
- entities represented as nodes of a graph
- capabilities represented as edges of a graph
- rights are contained in capabilities

The Rights:

Read
Write
Create
Take
Grant

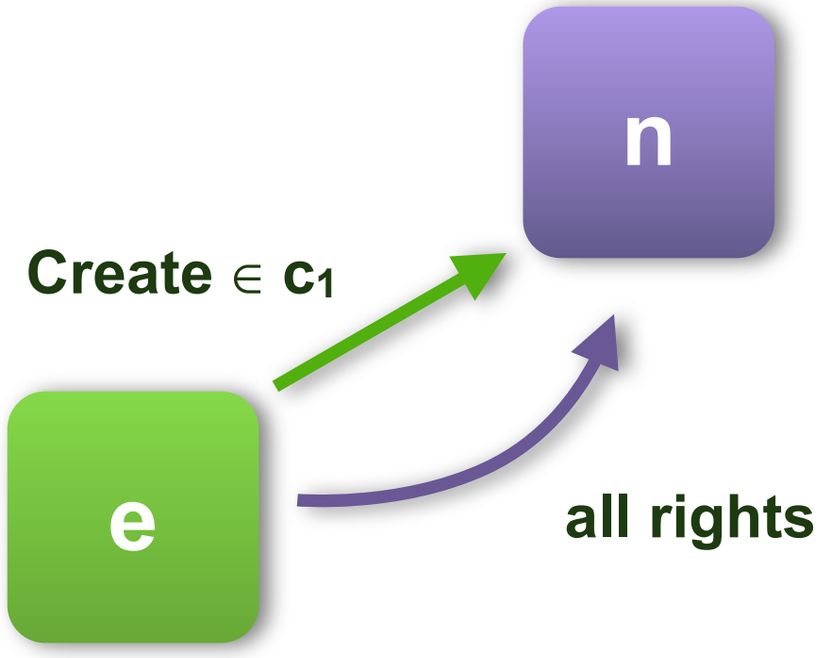


Operations - Create



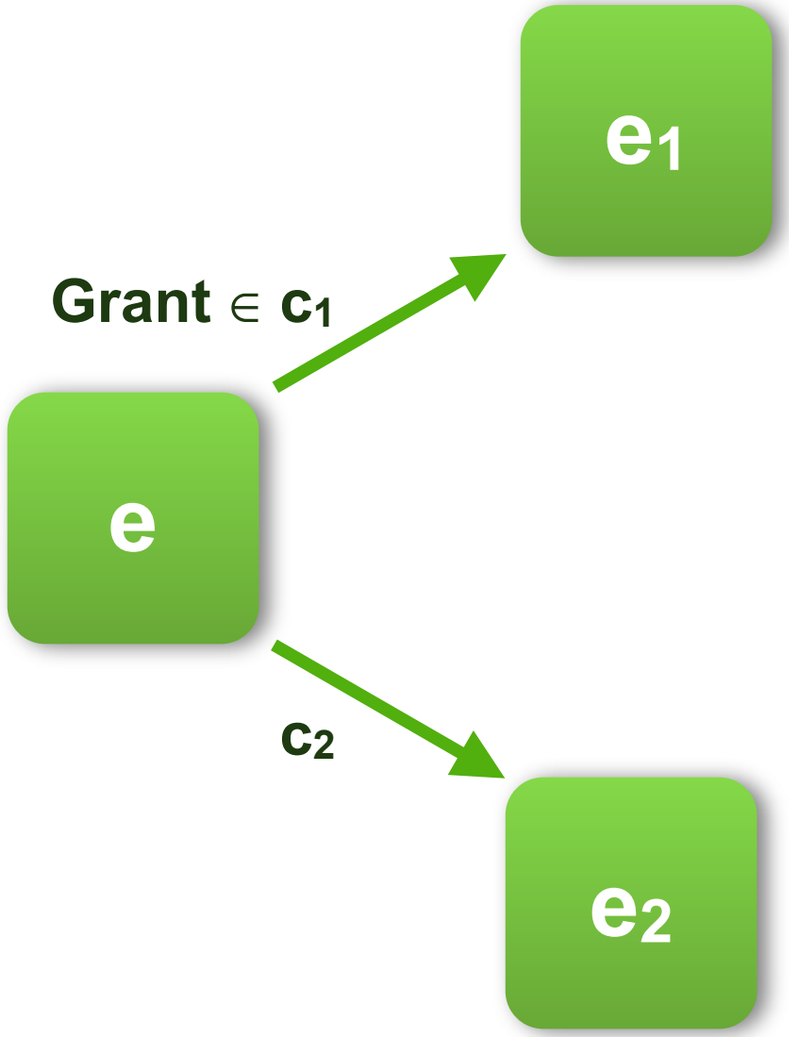
Create new entity

Operations - Create



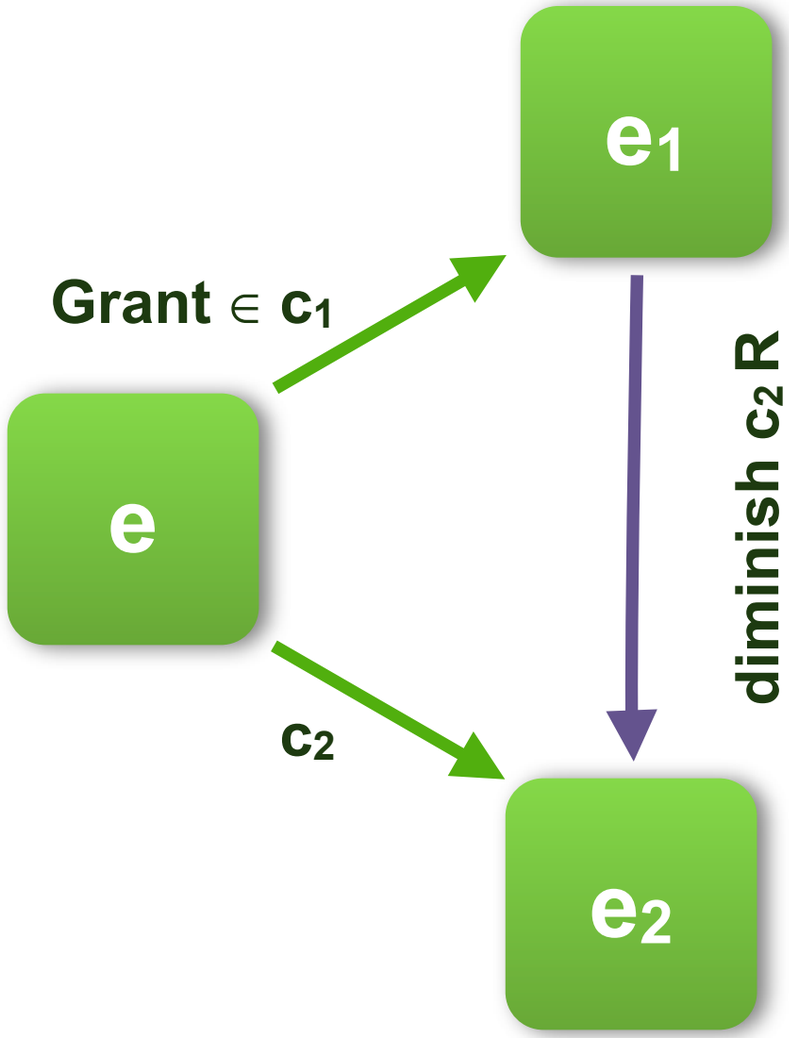
Create new entity

Operations - Grant



Grant c_2 to e_1
with mask R

Operations - Grant



Grant c_2 to e_1
with mask R

Operations - Remove/Delete

Remove capability c_2

Delete entity e_2

Operations - Remove/Delete

Remove capability c_2

Delete entity e_2

Operations - Remove/Delete

Remove capability c_2

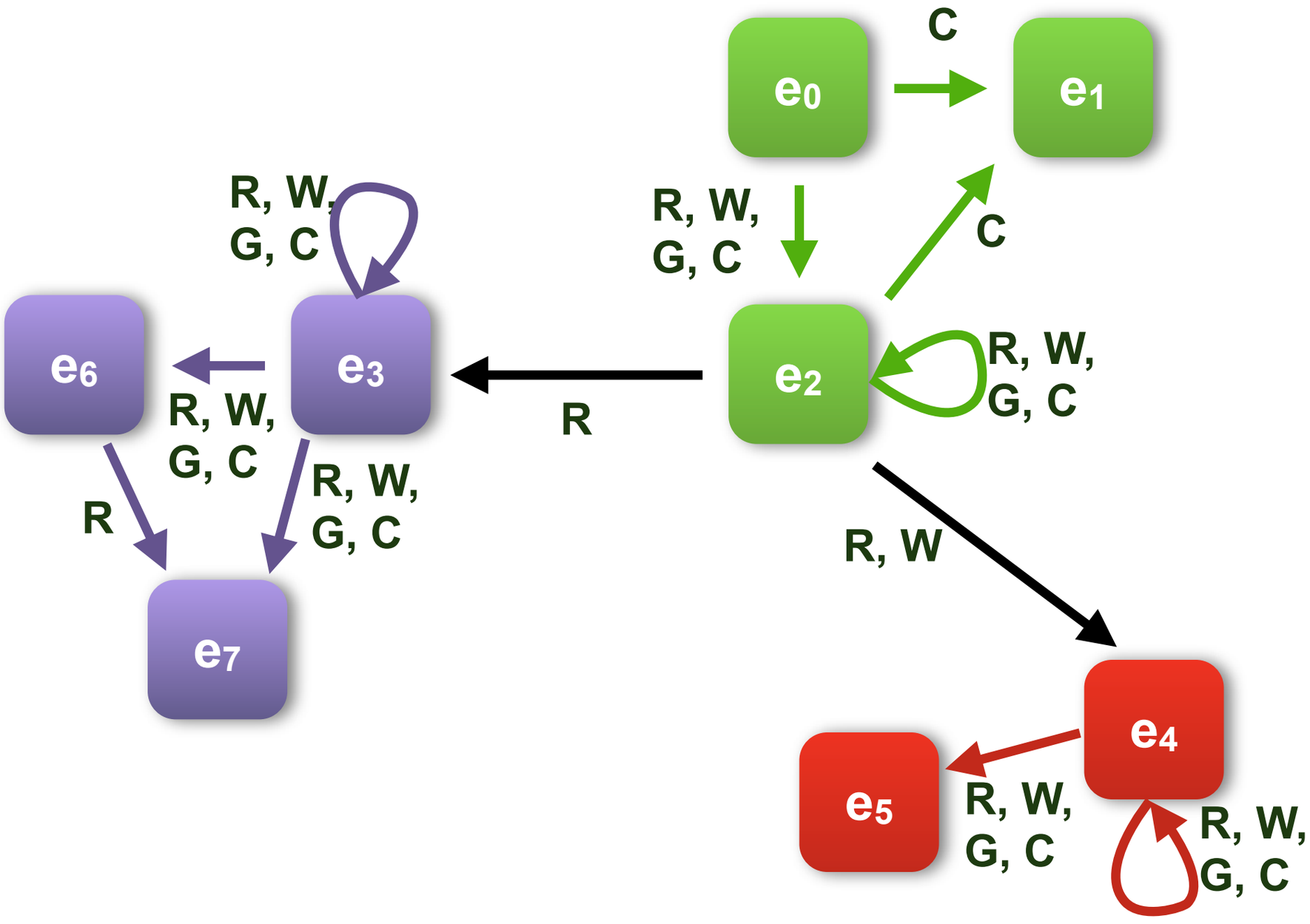
Delete entity e_2

Questions

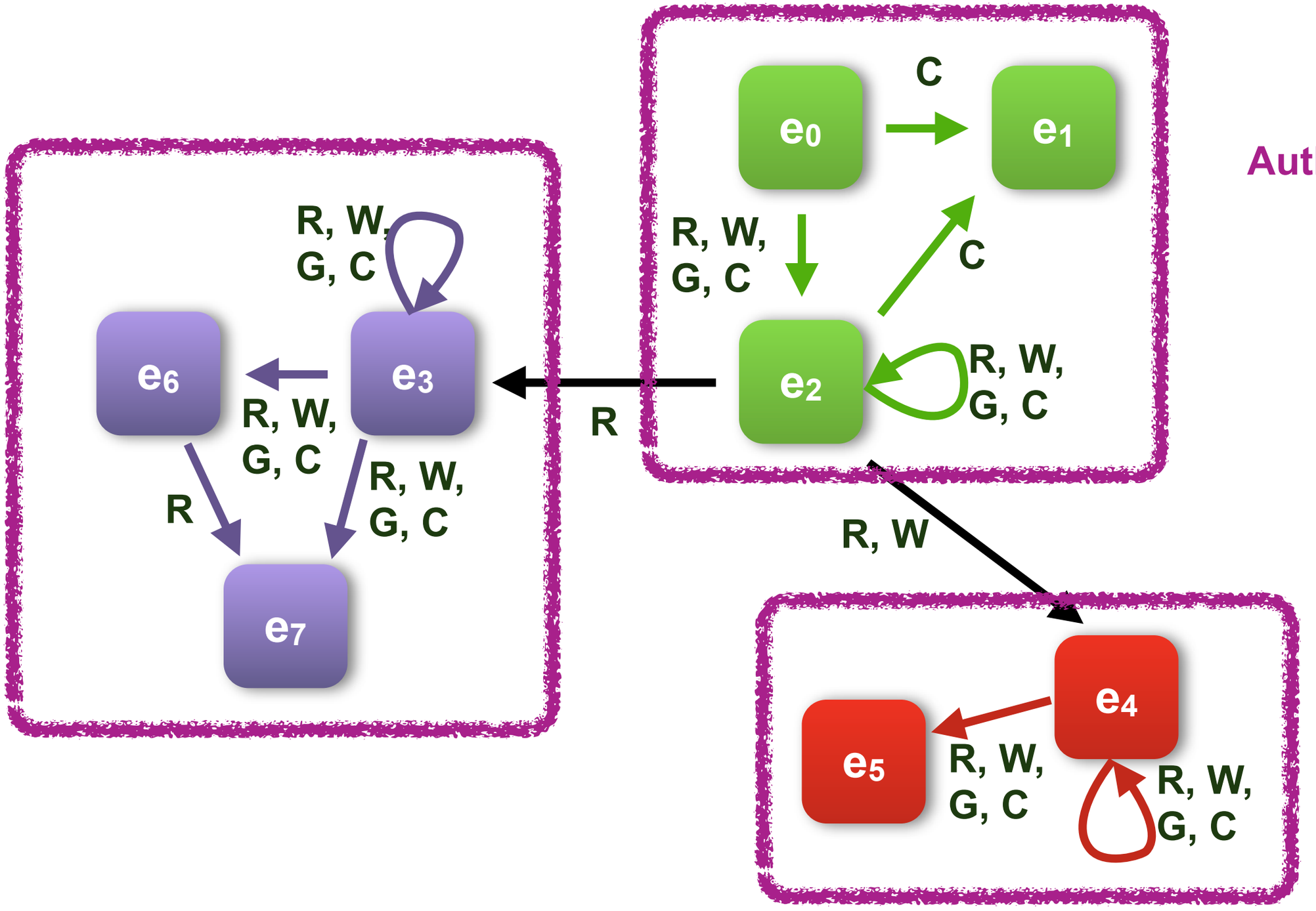
For any state in the future:

- Can **E** gain authority to do **X**?
- Can **E** gain more **authority** than it has?
- **How** much more?
- Can **information** flow from A to B?

Example

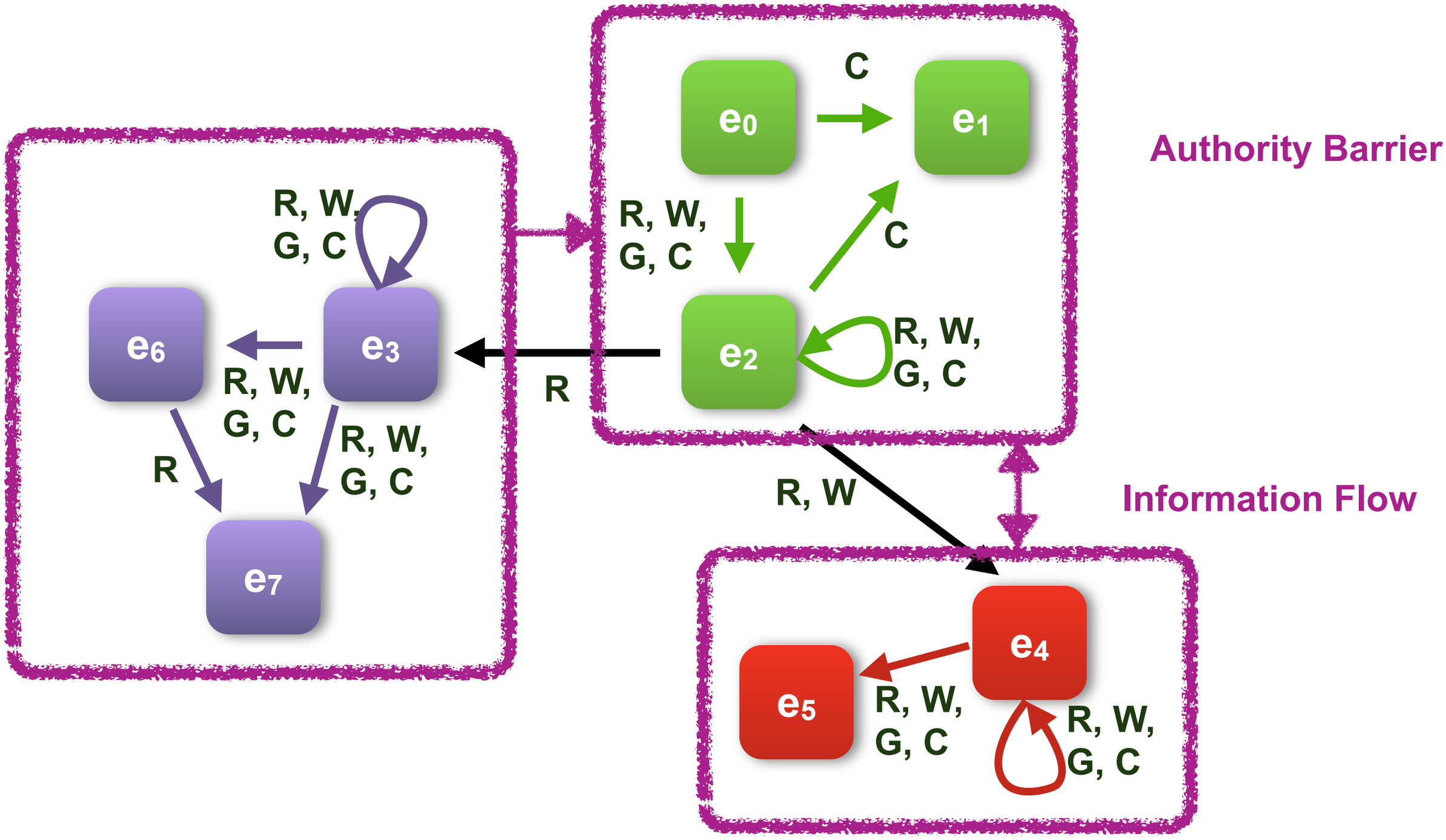

e0
C
e1
R, W,
G, C
C
R, W,
G, C
e6
e3
e2
R, W,
G, C
R
R, W,
G, C
R, W,
G, C
R
R, W
e4
e5
R, W,
G, C
R, W,
G, C
e7

Example

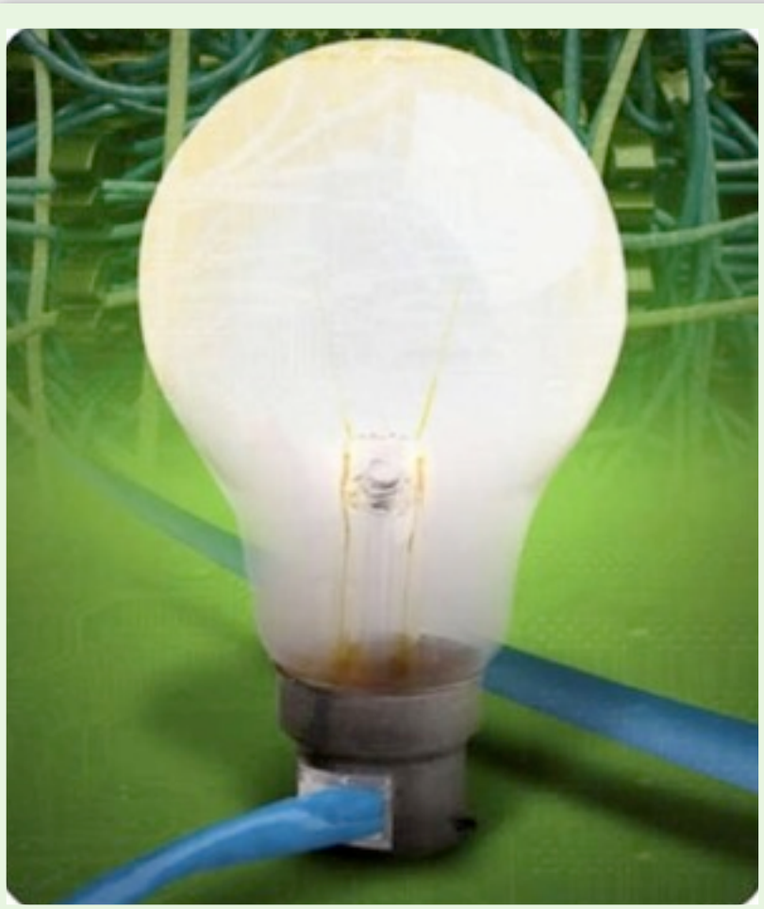


Authority Barrier

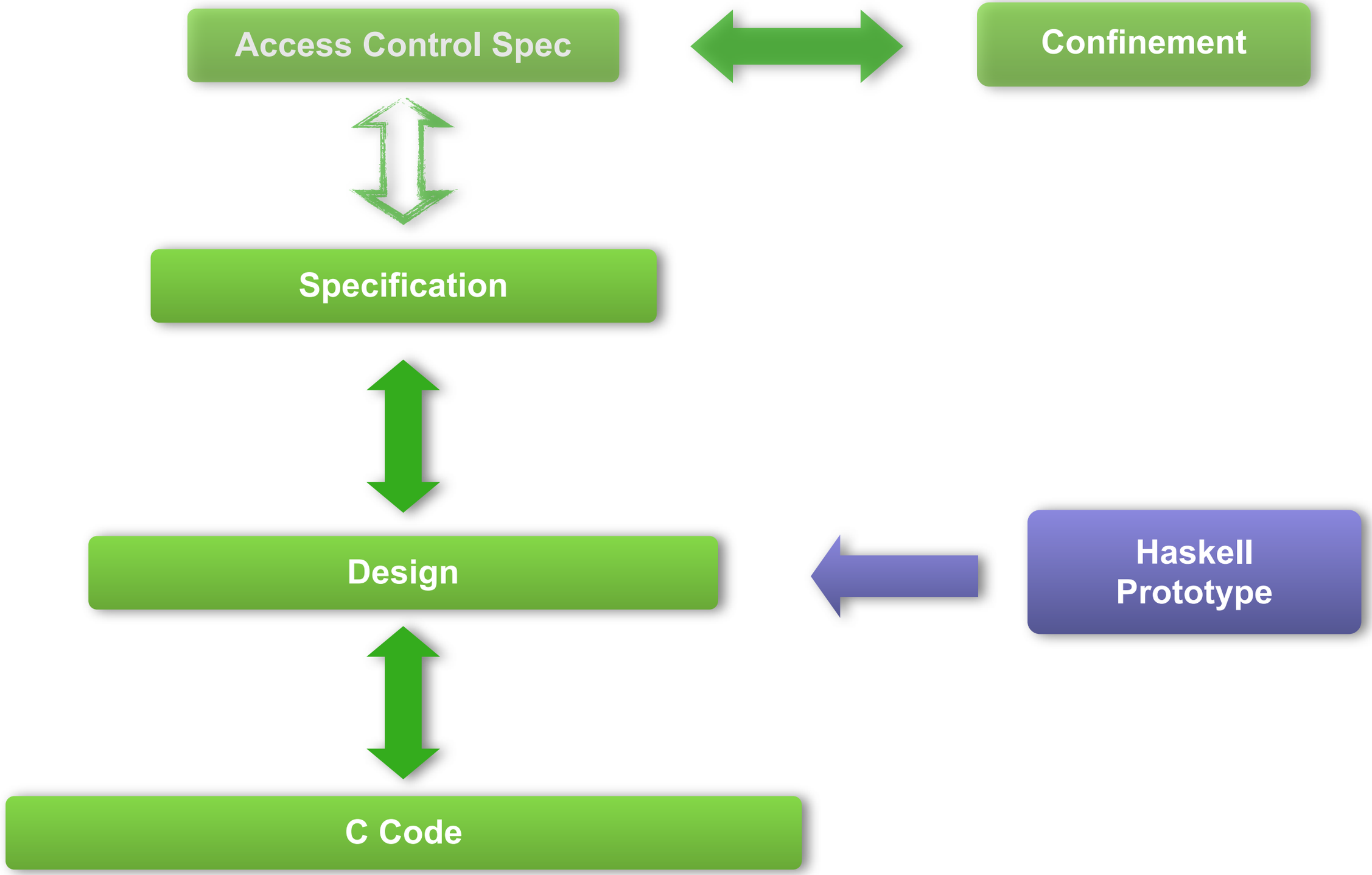
Example


e0
C
e1
R, W,
G, C
C
R, W,
G, C
e6
e3
e2
R, W,
G, C
R
R, W,
G, C
R, W,
G, C
R
R, W
e7
e4
e5
R, W,
G, C
R, W,
G, C
Authority Barrier
Information Flow

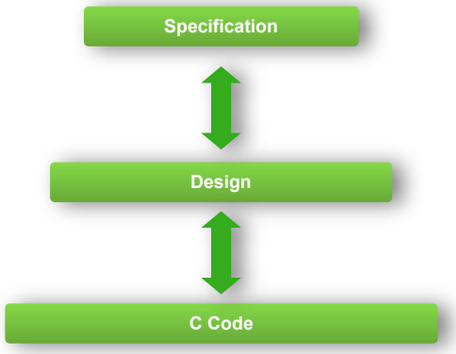
Now What?



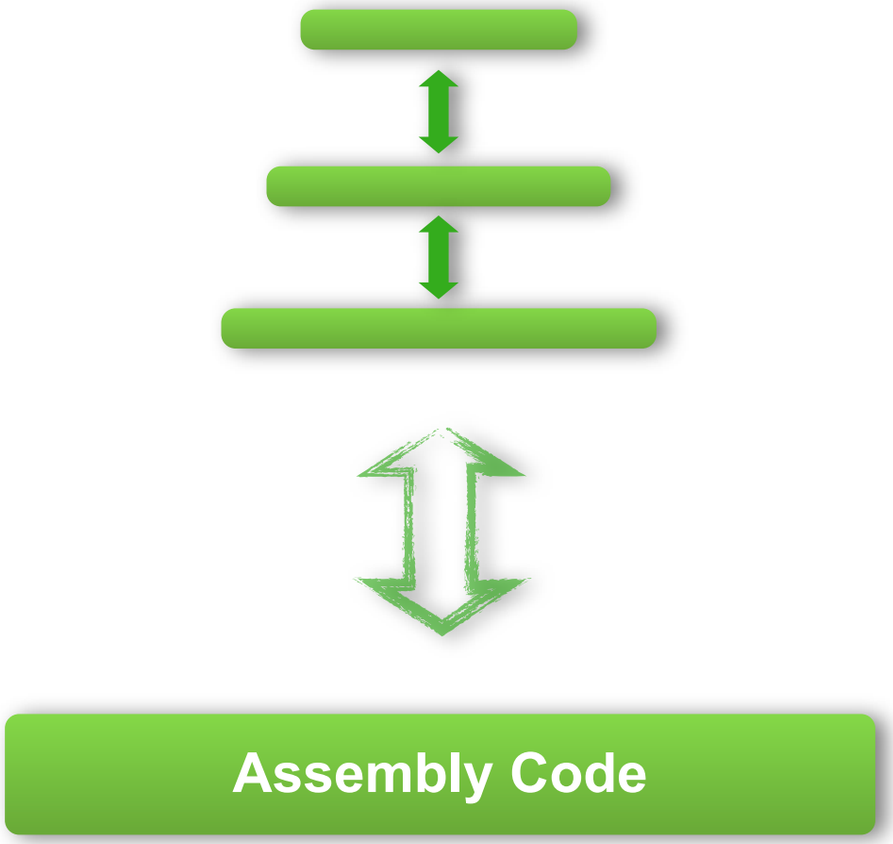
Current Proof



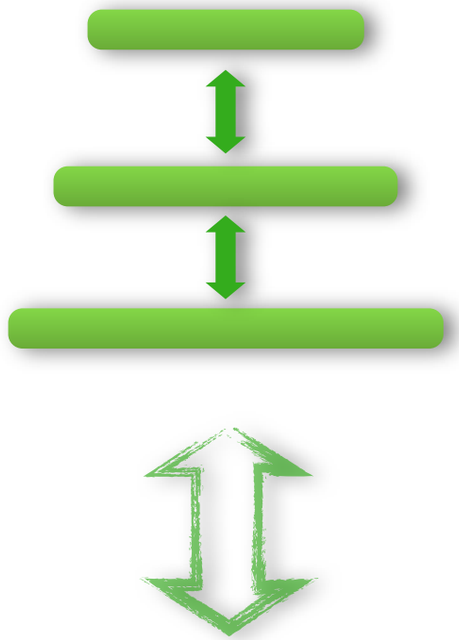
Current Proof



Even More Assurance?



Even More Assurance?



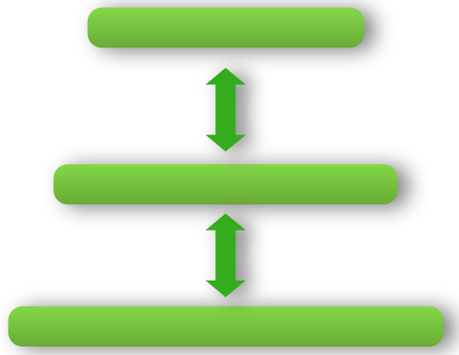
Assume correct:

- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

Assembly Code

Even More Assurance?

NICTA



Assume correct:

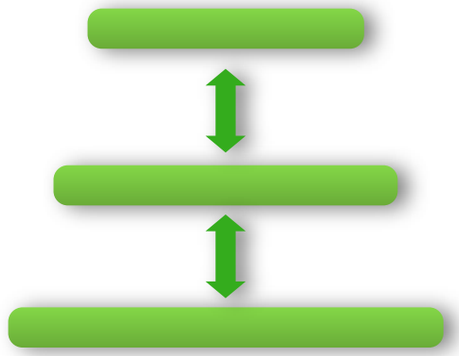
- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

Compiler Verification

CompCert

Assembly Code

Even More Assurance?



Assume correct:

- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

Compiler Verification

CompCert

Assembly Code

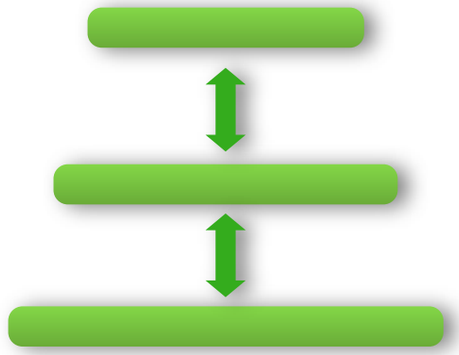
Assembly Verification

ARM model, Fox et al

Verisoft

Even More Assurance?

NICTA



Assume correct:

- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

Compiler Verification

CompCert

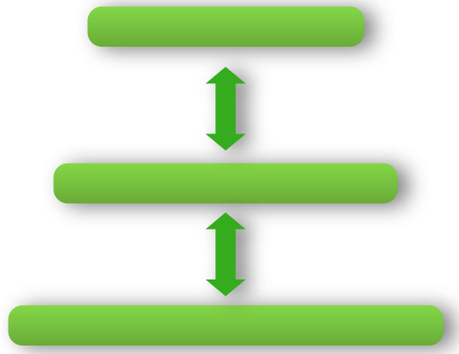
Assembly Code

Calling Conventions
Frame Conditions

Assembly Verification

ARM model, Fox et al
Verisoft

Even More Assurance?



Assume correct:

- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

Compiler Verification

CompCert

Coq

Assembly Code

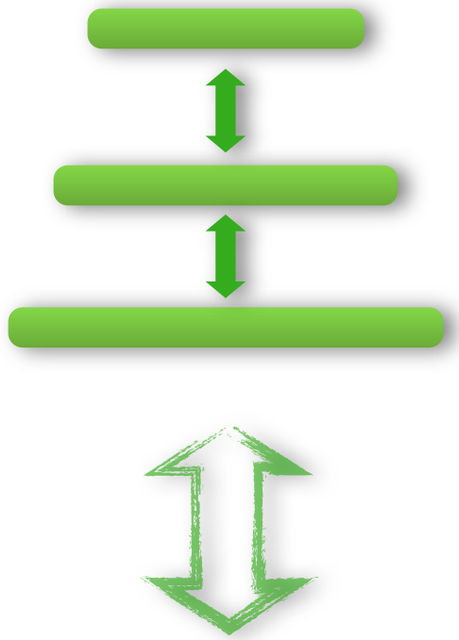
Calling Conventions
Frame Conditions

Assembly Verification

ARM model, Fox et al
Verisoft

HOL4

Even More Assurance?

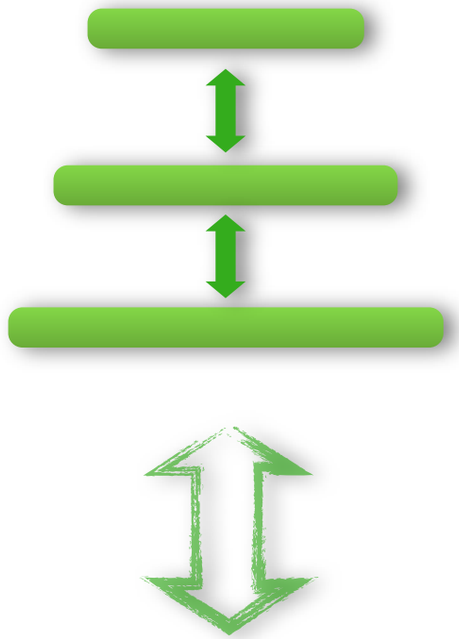


Assume correct:

- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

Assembly Code

Even More Assurance?



Assume correct:

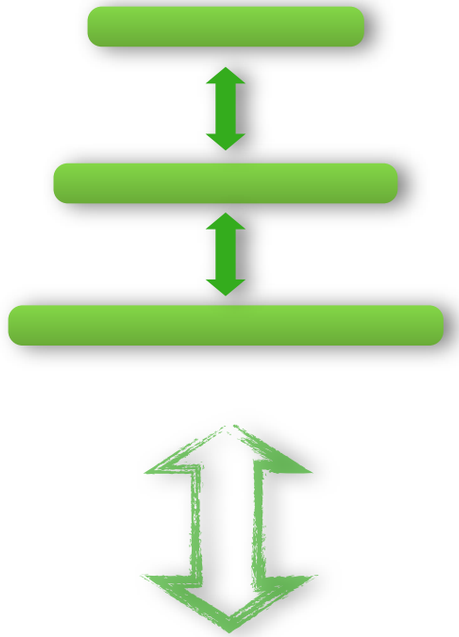
- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

Assembly Code | **Cache/TLB model**

Verisoft XT

Even More Assurance?

boot result verification



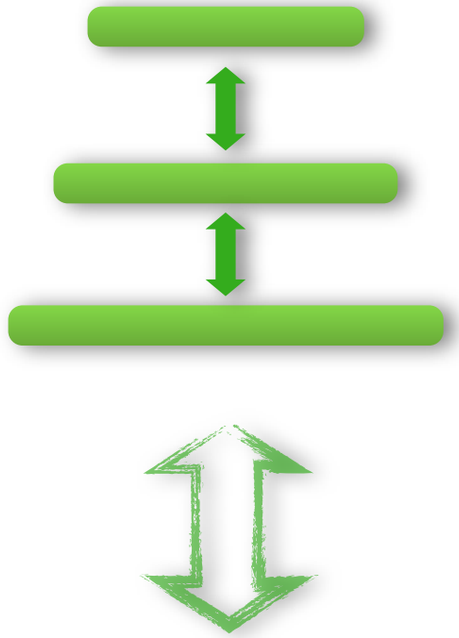
Assume correct:

- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
  - cache and TLB management
  - boot code (1,200 loc)

Assembly Code | **Cache/TLB model** | Verisoft XT

Even More Assurance?

boot result verification



Assume correct:

- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

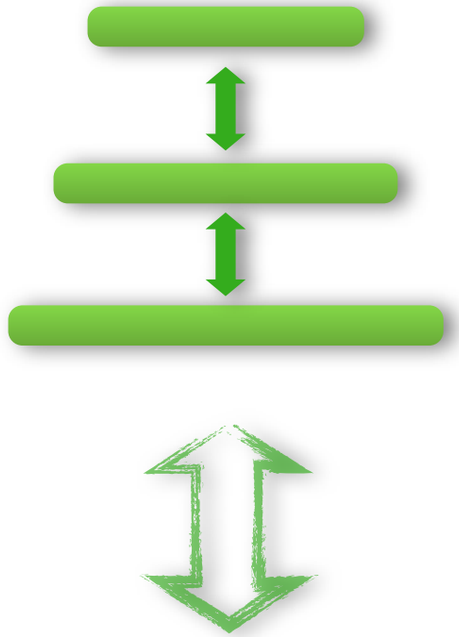
Assembly Code | **Cache/TLB model** — Verisoft XT

Hardware Verification

VHDL? — VAMP, FM9001, Intel i7

Even More Assurance?

boot result verification



Assume correct:

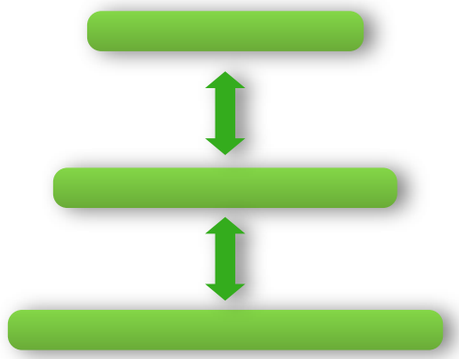
- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

Assembly Code | **Cache/TLB model** — Verisoft XT

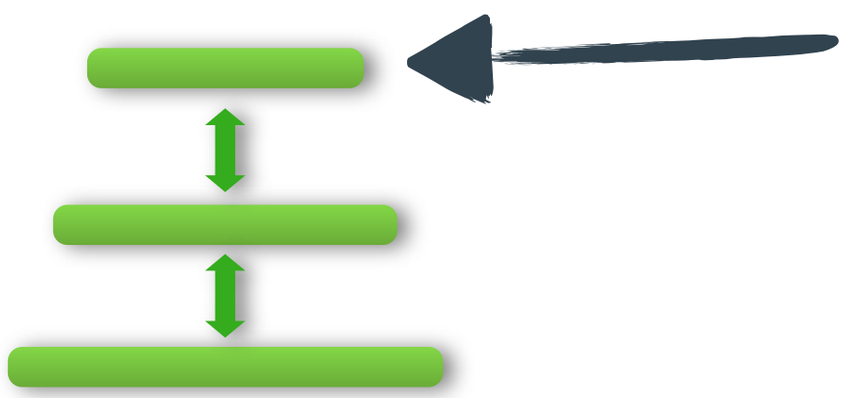
Hardware Verification

VHDL? — VAMP, FM9001, Intel i7

Systems On Top

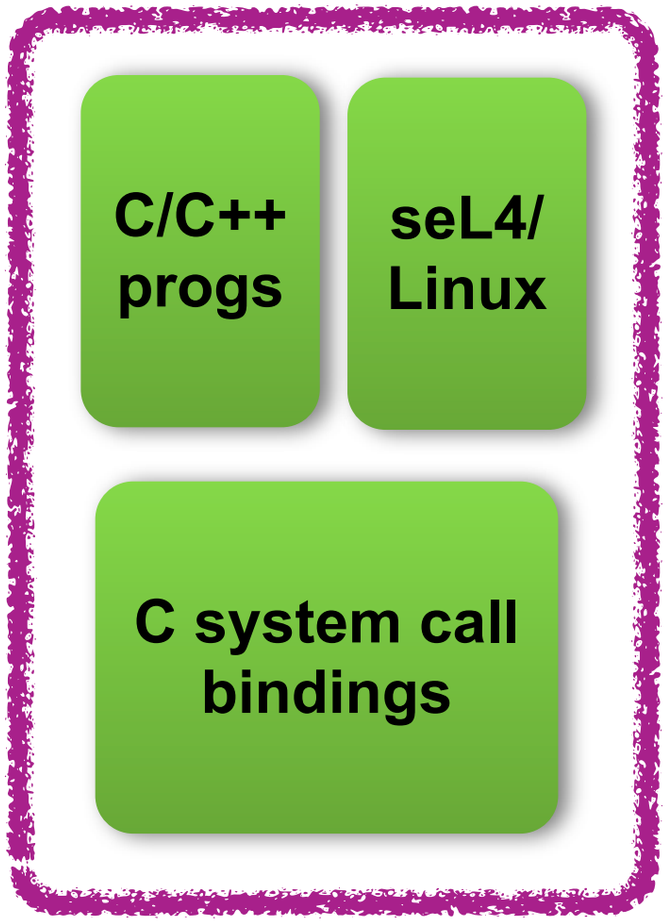


Systems On Top



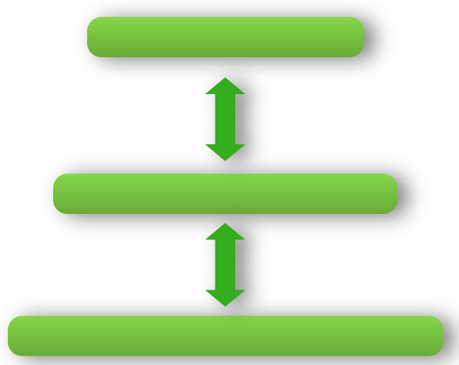
Describes Binary Interface

Systems On Top

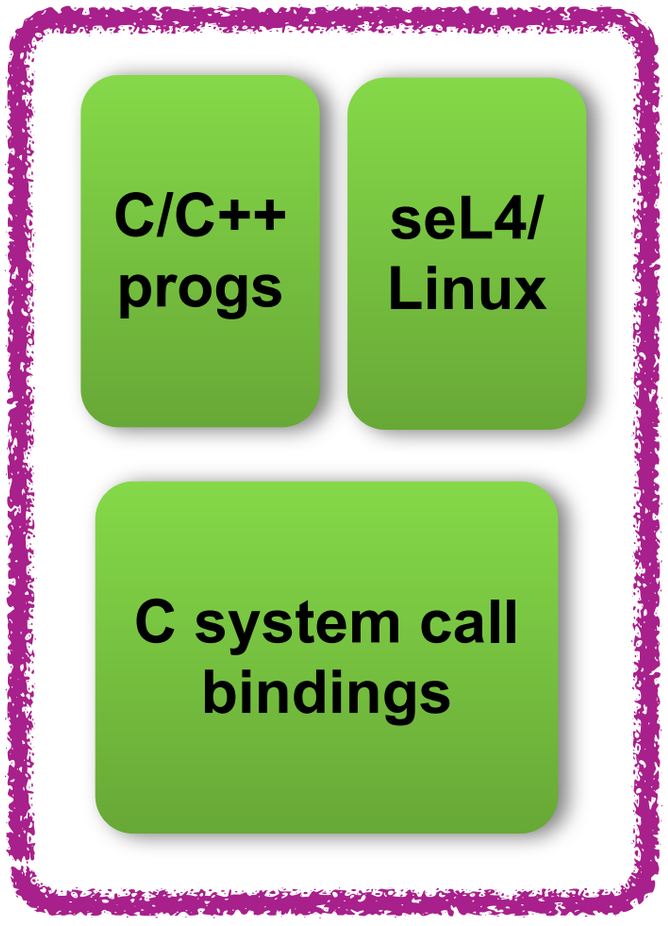


Exists:

- standard seL4 library
- used in seL4/Linux
- not hard to formally verify
- verification scheduled

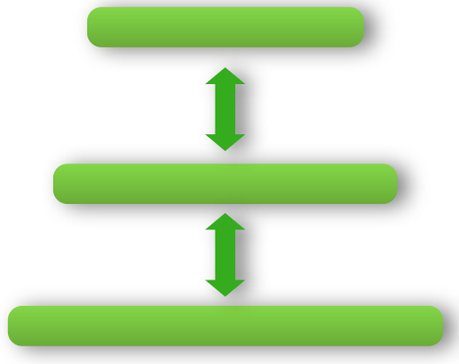


Systems On Top

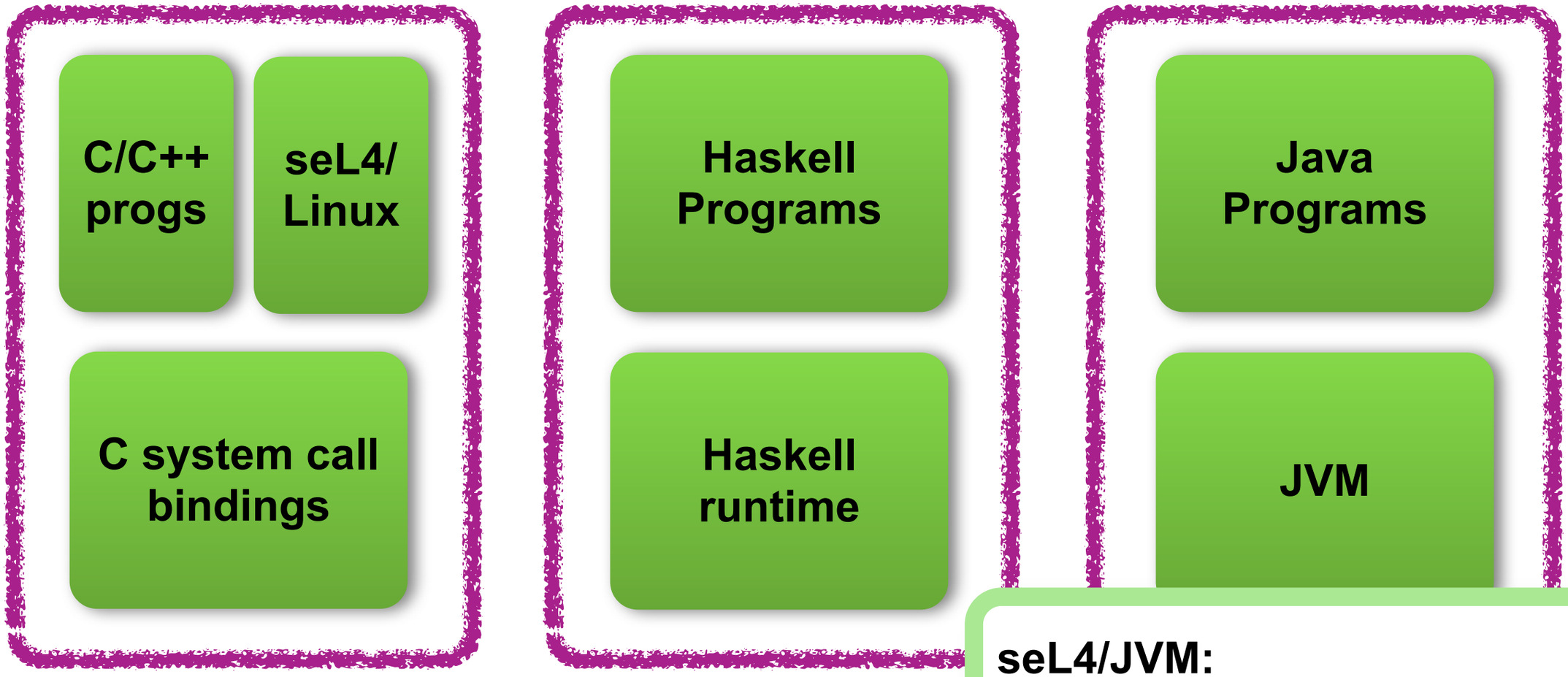


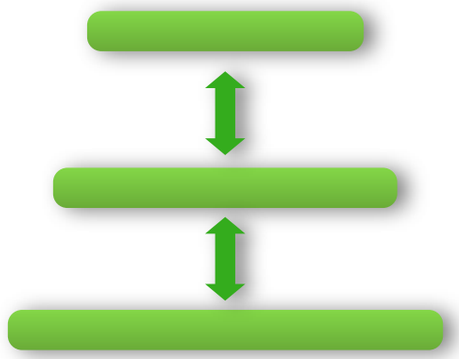
seL4/Haskell:

- early prototype Haskell runtime
- has seL4 systems call bindings
- verification hard
- runtime verification progress in HASP project @ PSU & Galois



Systems On Top

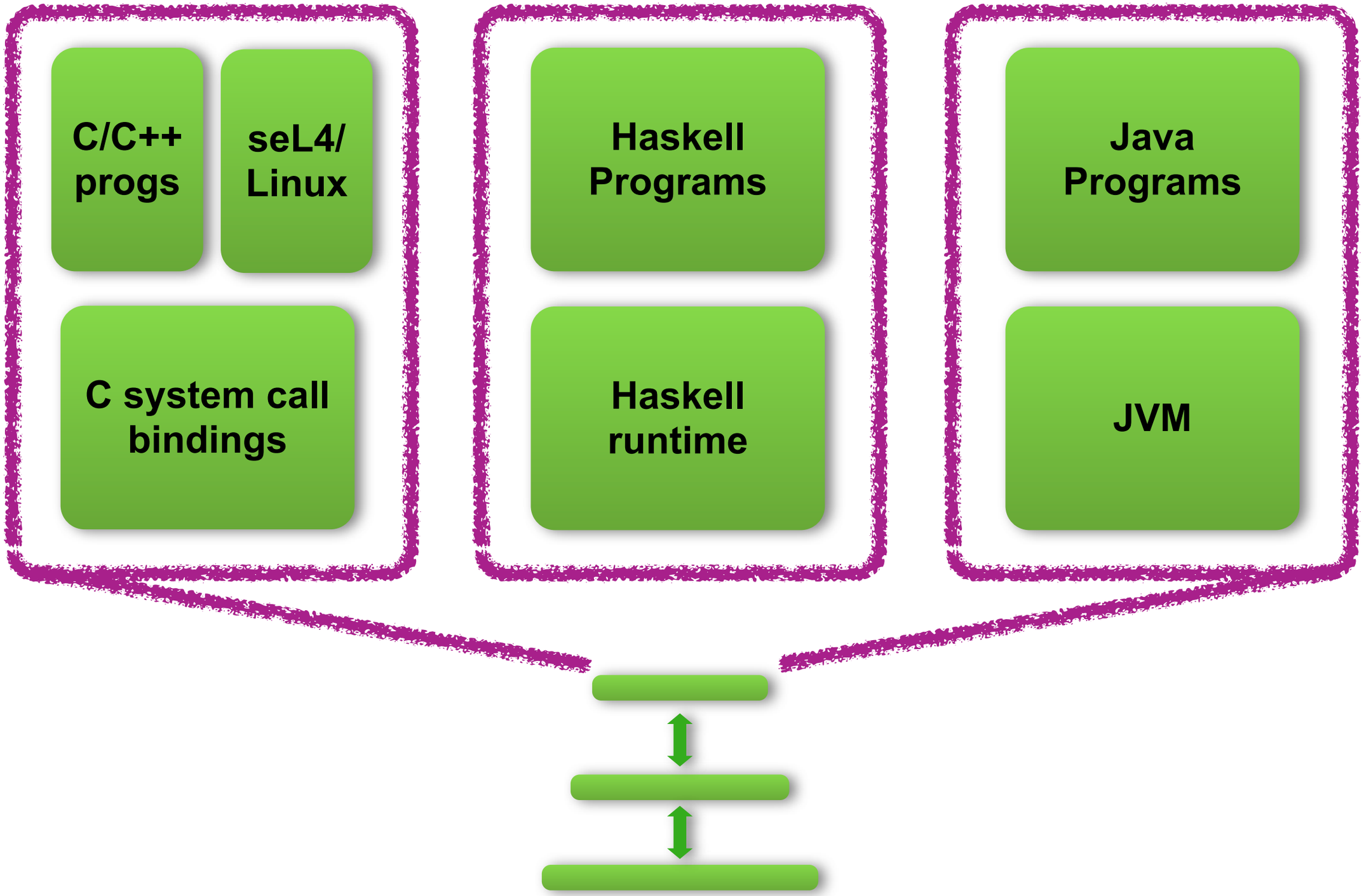




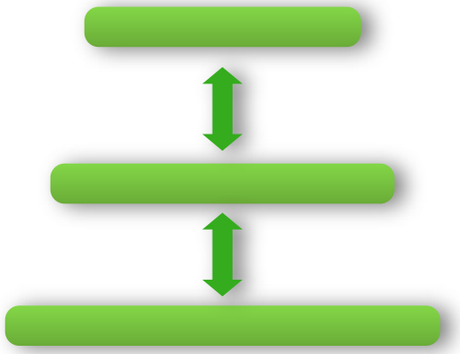
seL4/JVM:

- any takers?
- JVM extensively formalised
- widely used
- EAL7 smart card implementations exist

Systems On Top



Other Architectures

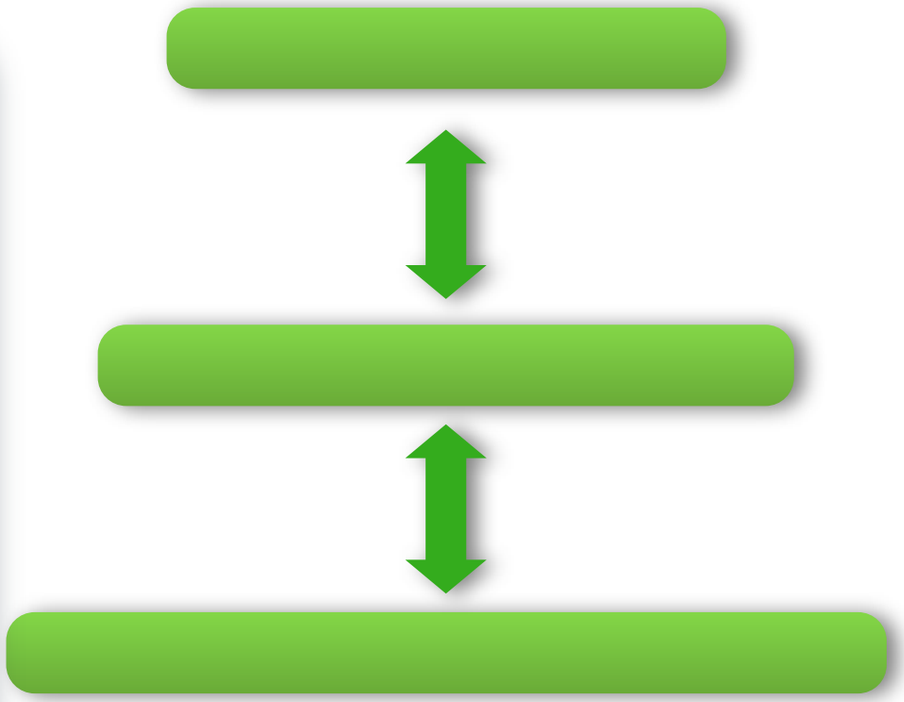


Other Architectures

seL4/x86:

- x86 version exists, supports Linux
- verification likely, not started yet

Intel 32bit



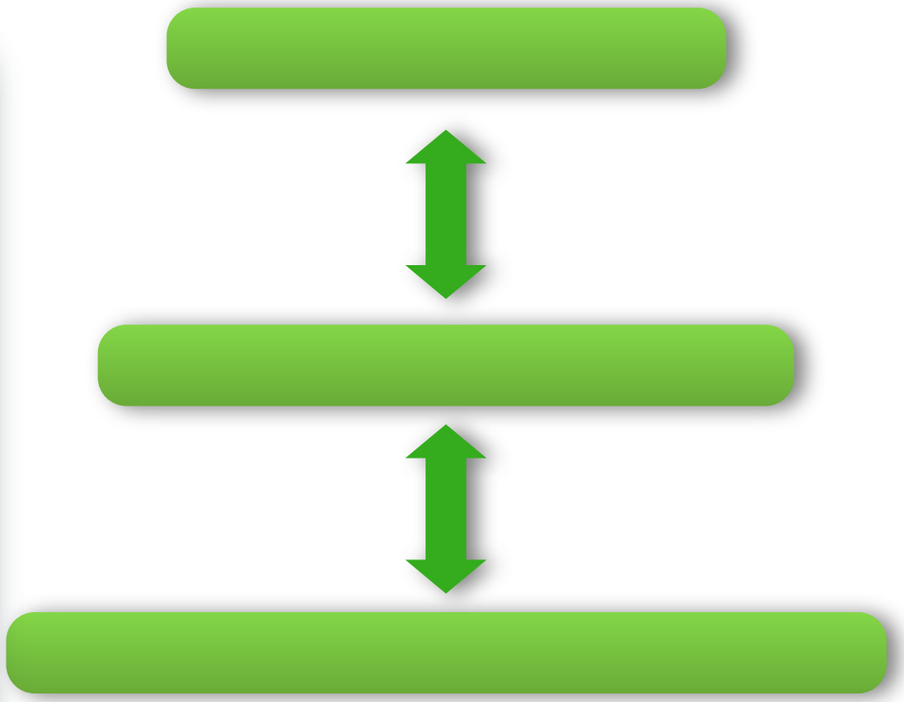
Other Architectures

seL4/x86:

- x86 version exists, supports Linux
- verification likely, not started yet

- Intel VT-d/IOMMU implemented
- enables untrusted device DMA
- verification possible

Intel 32bit + IOMMU



Other Architectures

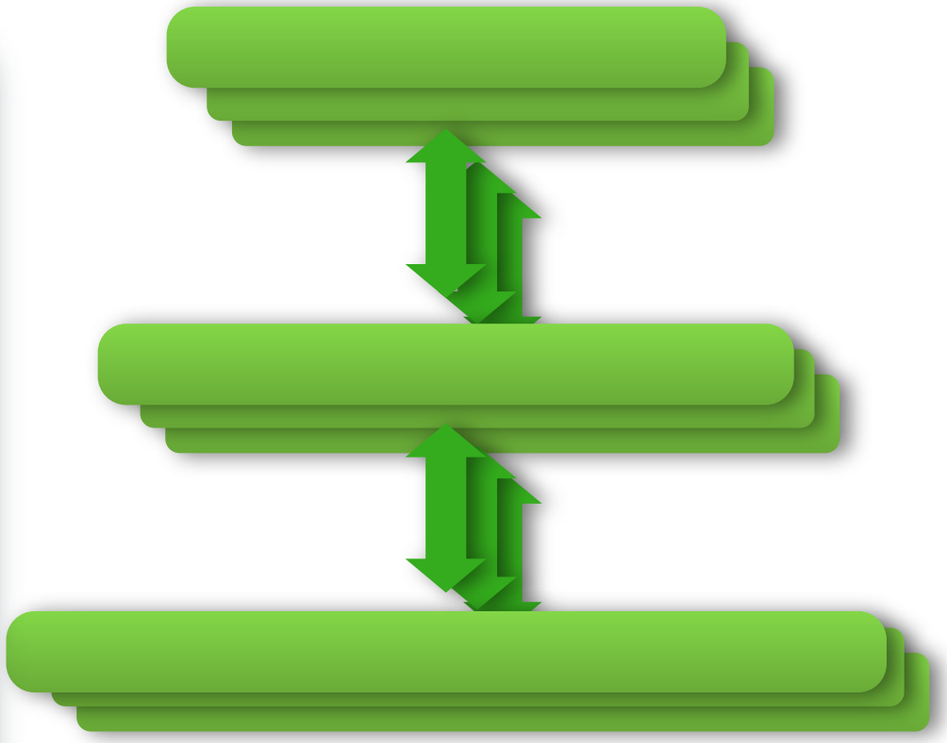
seL4/x86:

- x86 version exists, supports Linux
- verification likely, not started yet

- Intel VT-d/IOMMU implemented
- enables untrusted device DMA
- verification possible

- experimental multi processor version
- initial proofs exist

Intel 32bit + IOMMU + multi core



Other Architectures

seL4/x86:

- x86 version exists, supports Linux
- verification likely, not started yet

- Intel VT-d/IOMMU implemented
- enables untrusted device DMA
- verification possible

- experimental multi processor version
- initial proofs exist

Intel 32bit + IOMMU + multi core

New Architectures

- ca 1/3 of seL4 arch dependent
- close to ARM
  -> easy to implement and verify

Looking Forward

Looking Forward

Trustworthy Embedded Systems

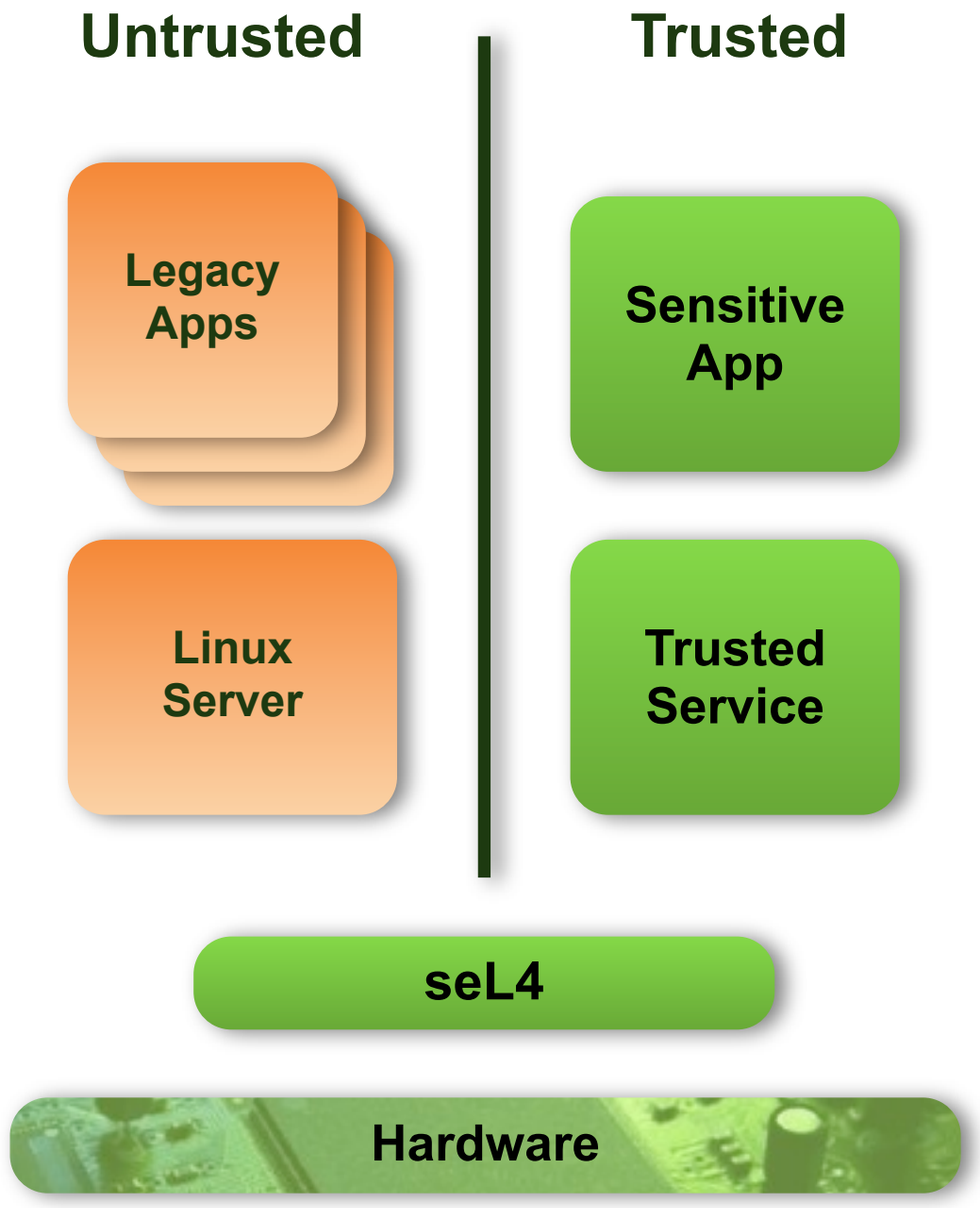
- **L4.verified:**
  functional correctness
  10,000 loc

- **Next step:**
  formal guarantees for
  > 1,000,000 loc

How?

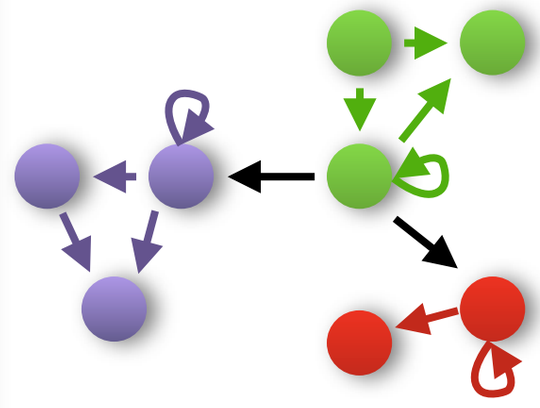
Exploit:

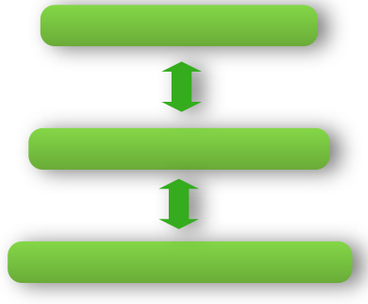
- seL4 isolation
- verified properties
- MILS architectures



Challenges

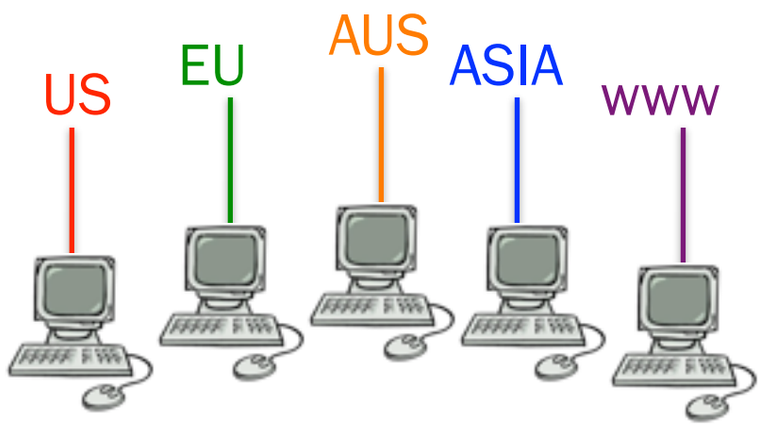
- Find right architecture
- Security analysis
  - identify trusted components
  - ideally take-grant style
  - behaviour of trusted components
- Code-level theorem in the end
  - connect to kernel proof
  - ideally prove trusted component only





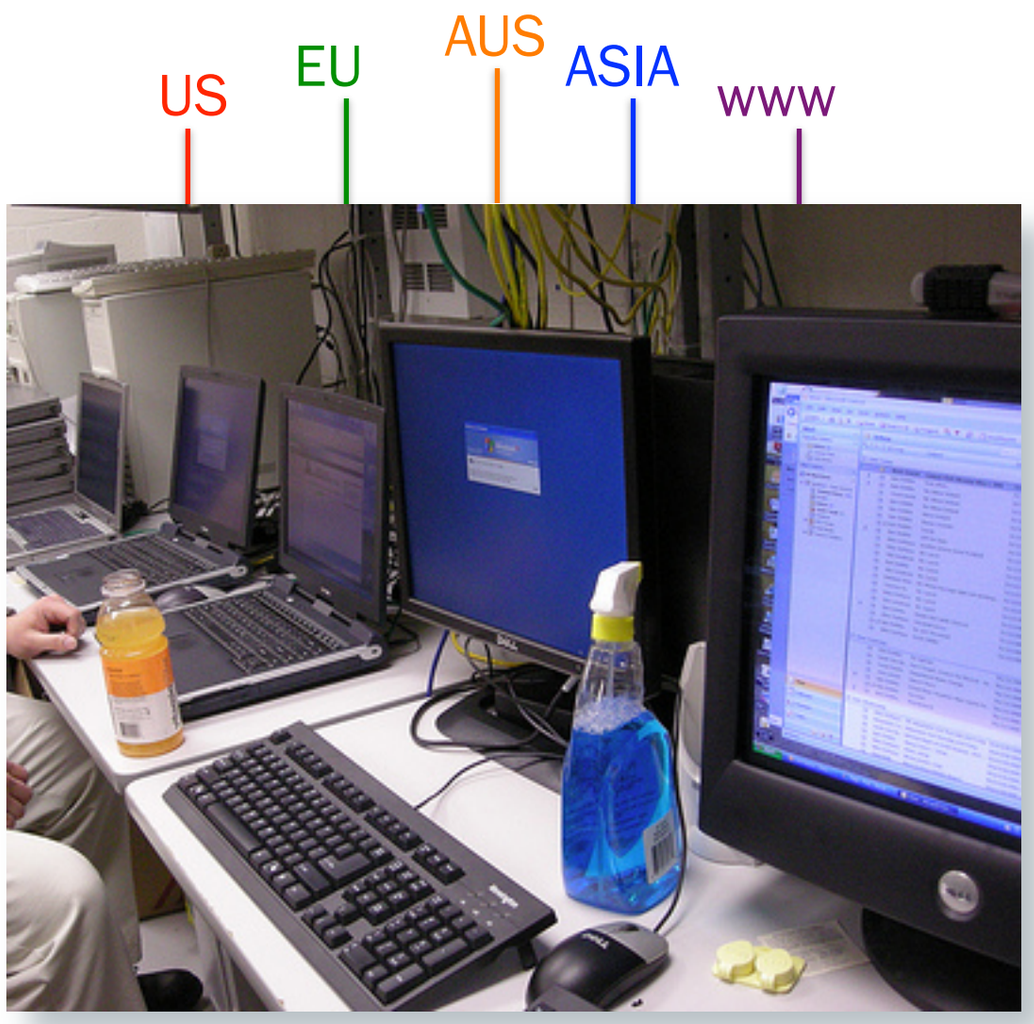
Example System

- Scenario:



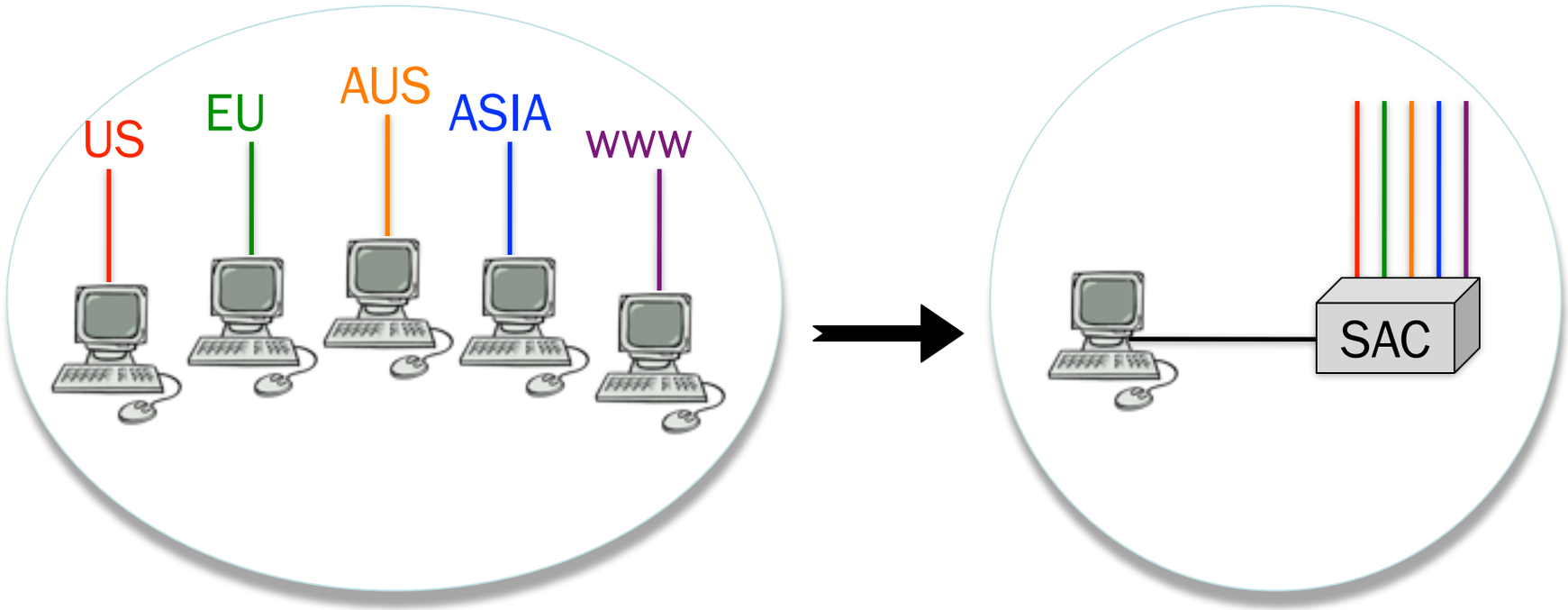
Example System

- Scenario:



Example System

- Multilevel Secure Access Device



Example System

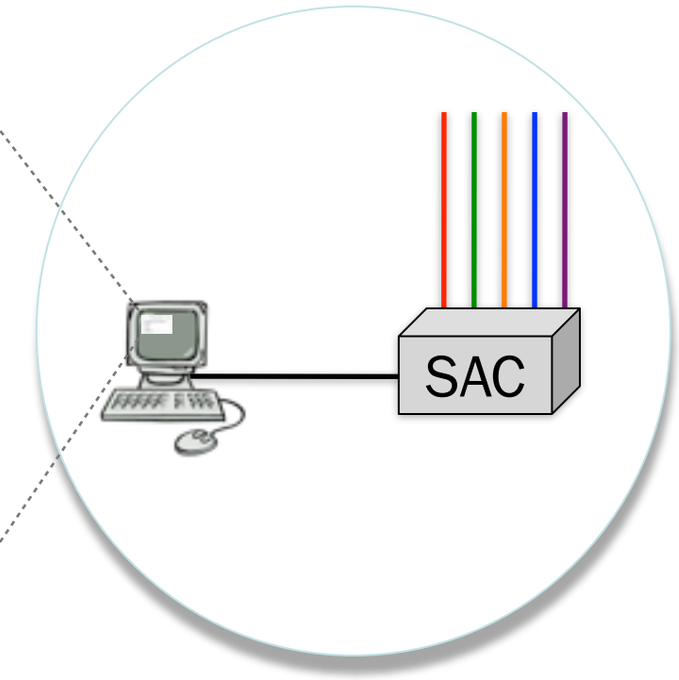
- Multilevel Secure Access Device

Secure Access Controller Login as: hotspy007
Logout

Currently selected connection: **No Active Connection**

Switch to Network:

- US
- EU
- AUS
- ASIA
- WWW



Example System

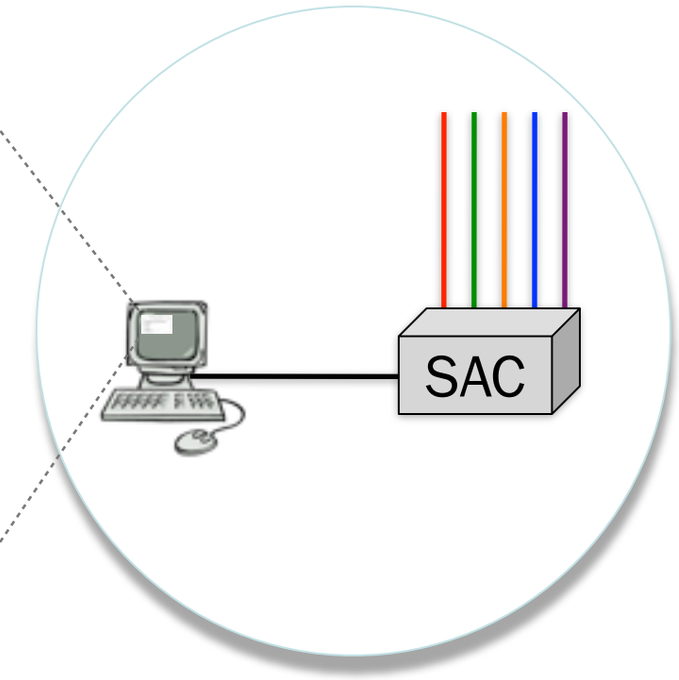
- Multilevel Secure Access Device

Secure Access Controller Login as: hotspy007
Logout

Currently selected connection: **No Active Connection**

Switch to Network:

- US
- EU
- AUS
- ASIA
- WWW



Example System

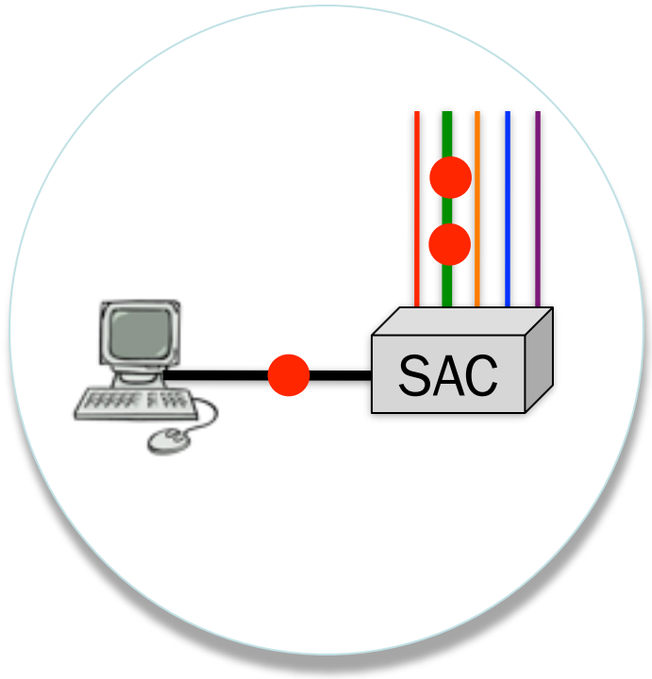
- Multilevel Secure Access Device

Secure Access Controller Login as: hotspy007
Logout

Currently selected connection: **No Active Connection**

Switch to Network:

- US
- EU
- AUS
- ASIA
- WWW



Example System

- Multilevel Secure Access Device

Secure Access Controller

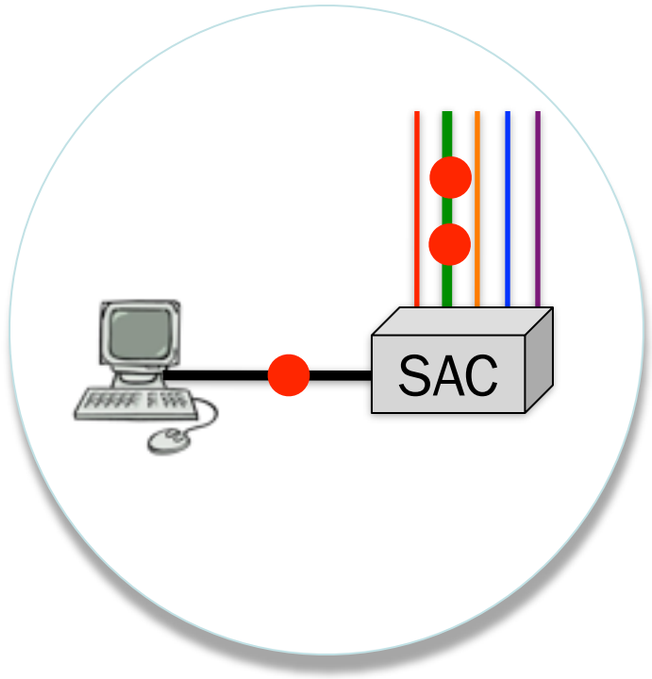
Login as: hotspy007
Logout

The Network has been successfully switched to **EU**

Currently selected connection: **EU**

Switch to Network:

- US
- EU
- AUS
- ASIA
- WWW



Example System

- Multilevel Secure Access Device

Secure Access Controller

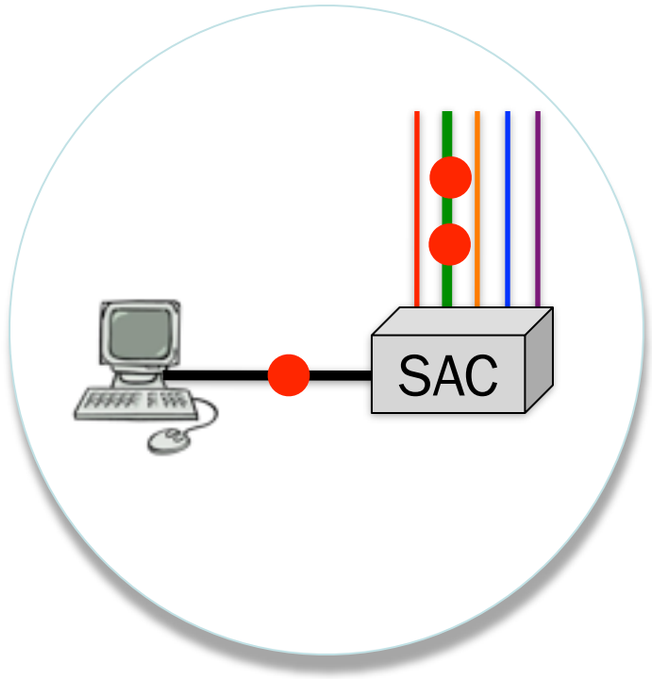
Login as: hotspy007
Logout

The Network has been successfully switched to **EU**

Currently selected connection: **EU**

Switch to Network:

- US
- EU
- AUS
- ASIA
- WWW



Example System

- Multilevel Secure Access Device

Secure Access Controller

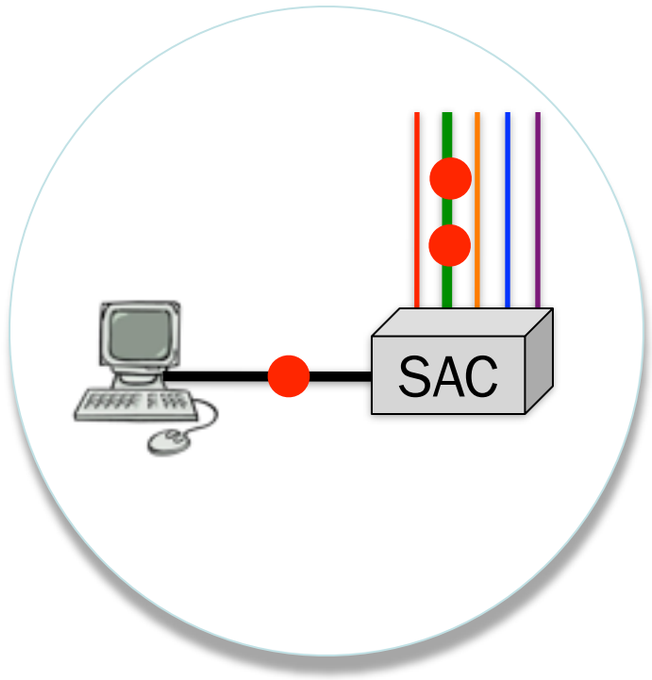
Login as: hotspy007
Logout

The Network has been successfully switched to **EU**

Currently selected connection: **EU**

Switch to Network:

- US
- EU
- AUS
- ASIA
- WWW



Example System

- Multilevel Secure Access Device

Secure Access Controller

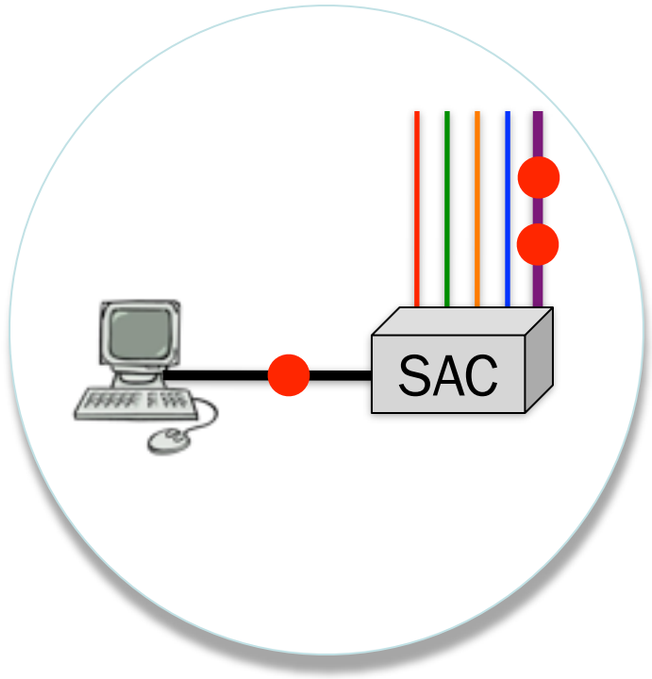
Login as: hotspy007
Logout

The Network has been successfully switched to **WWW**

Currently selected connection: **WWW**

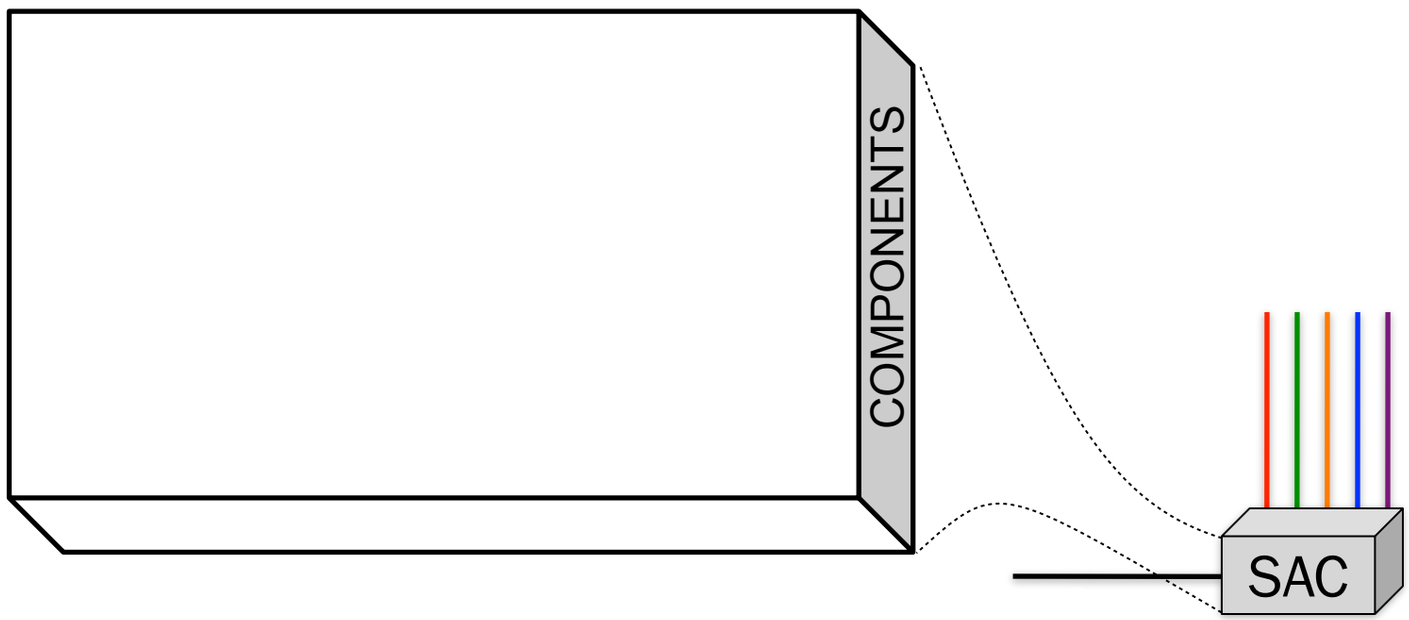
Switch to Network:

- US
- EU
- AUS
- ASIA
- WWW



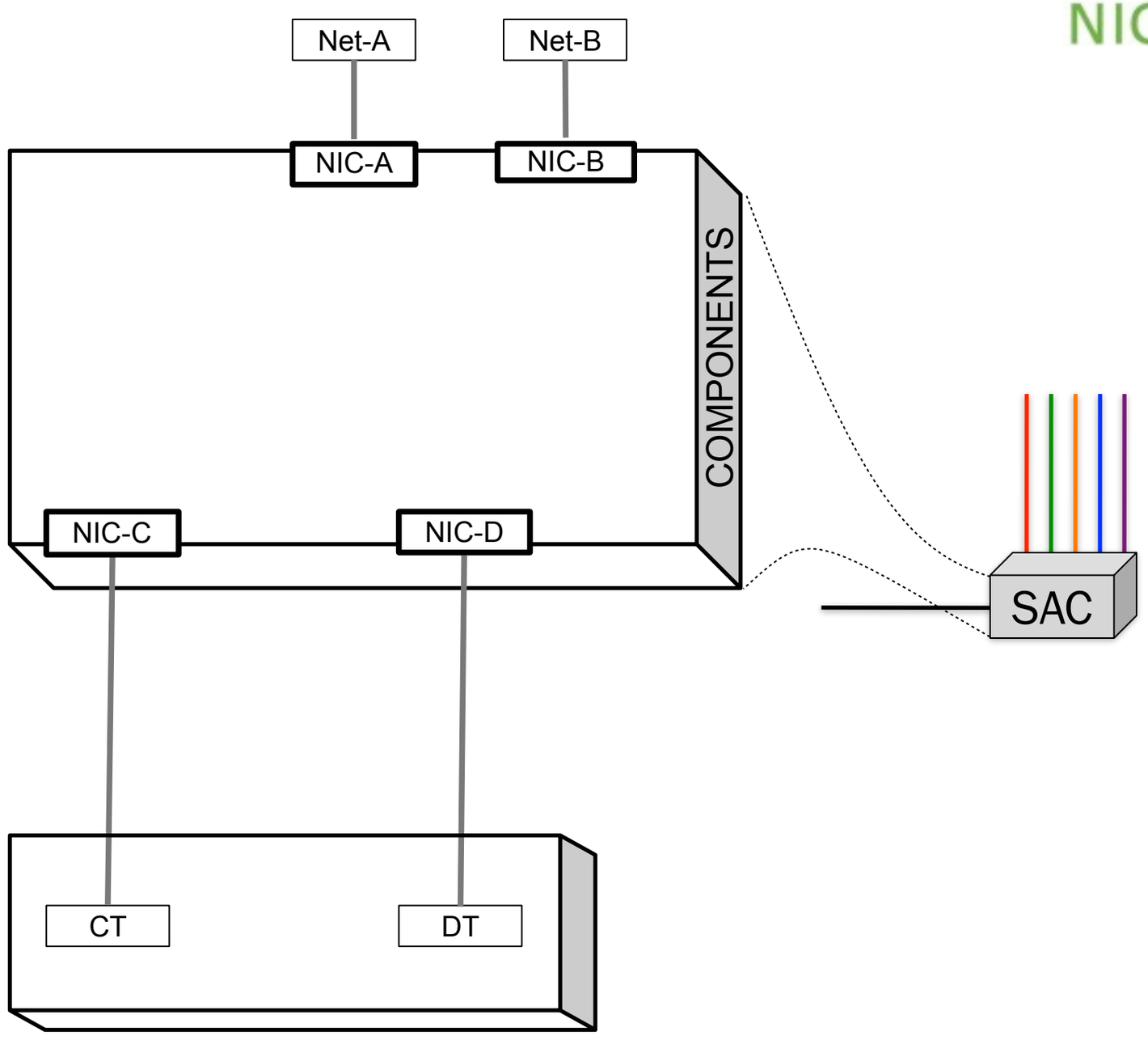
SAC System

Components



SAC System

Components

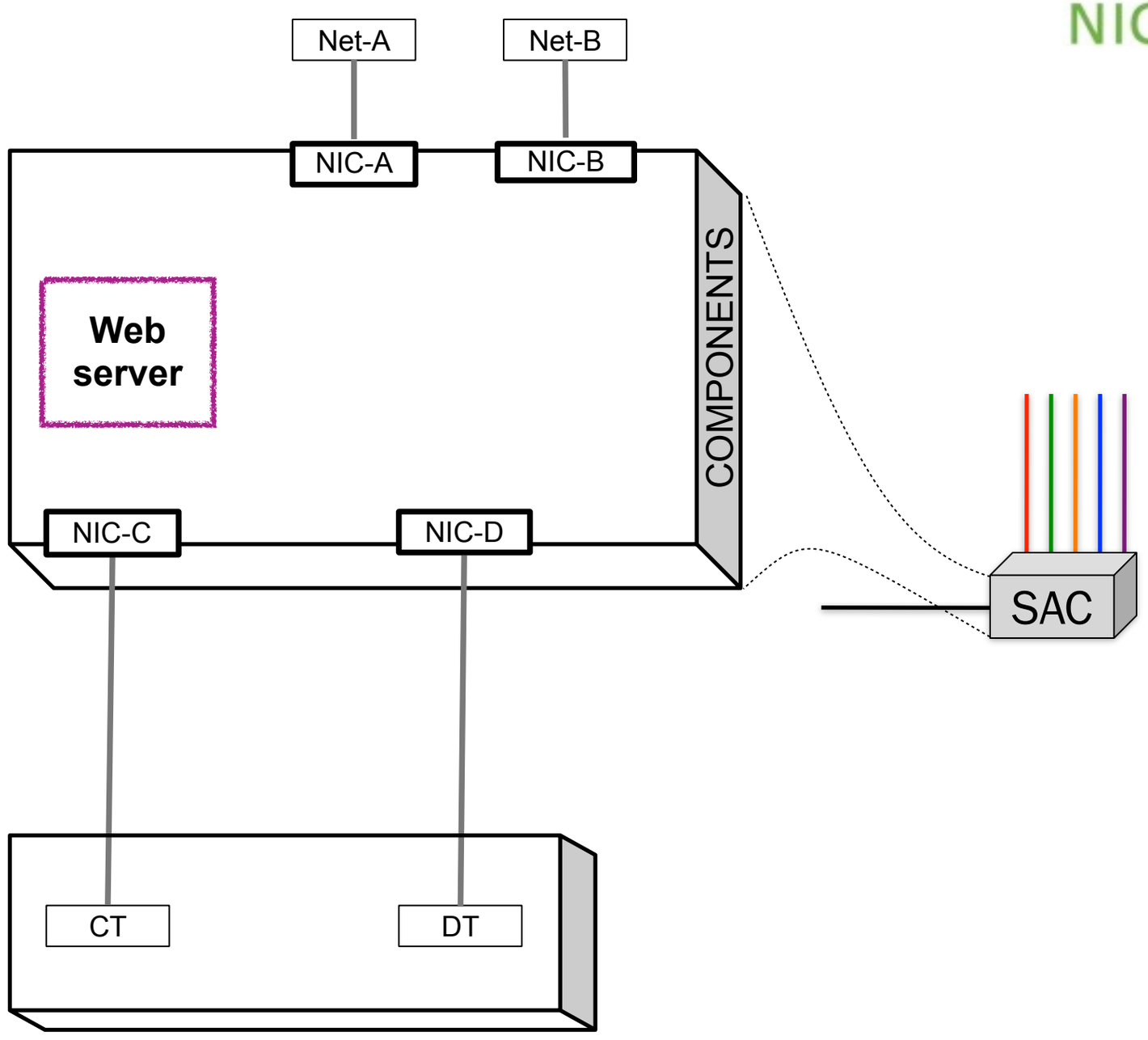


Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B

NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

SAC System

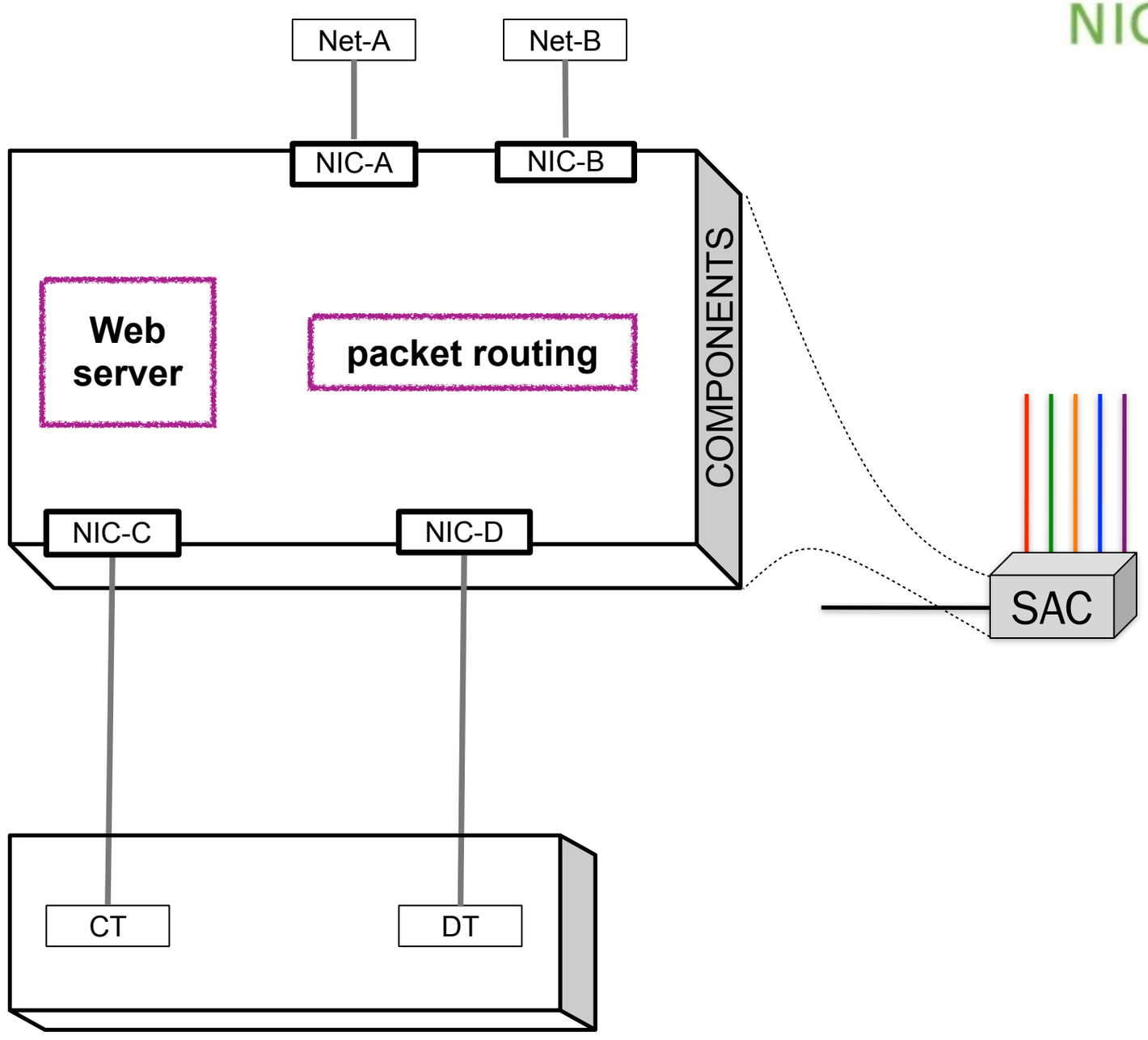
Components



Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

SAC System

Components

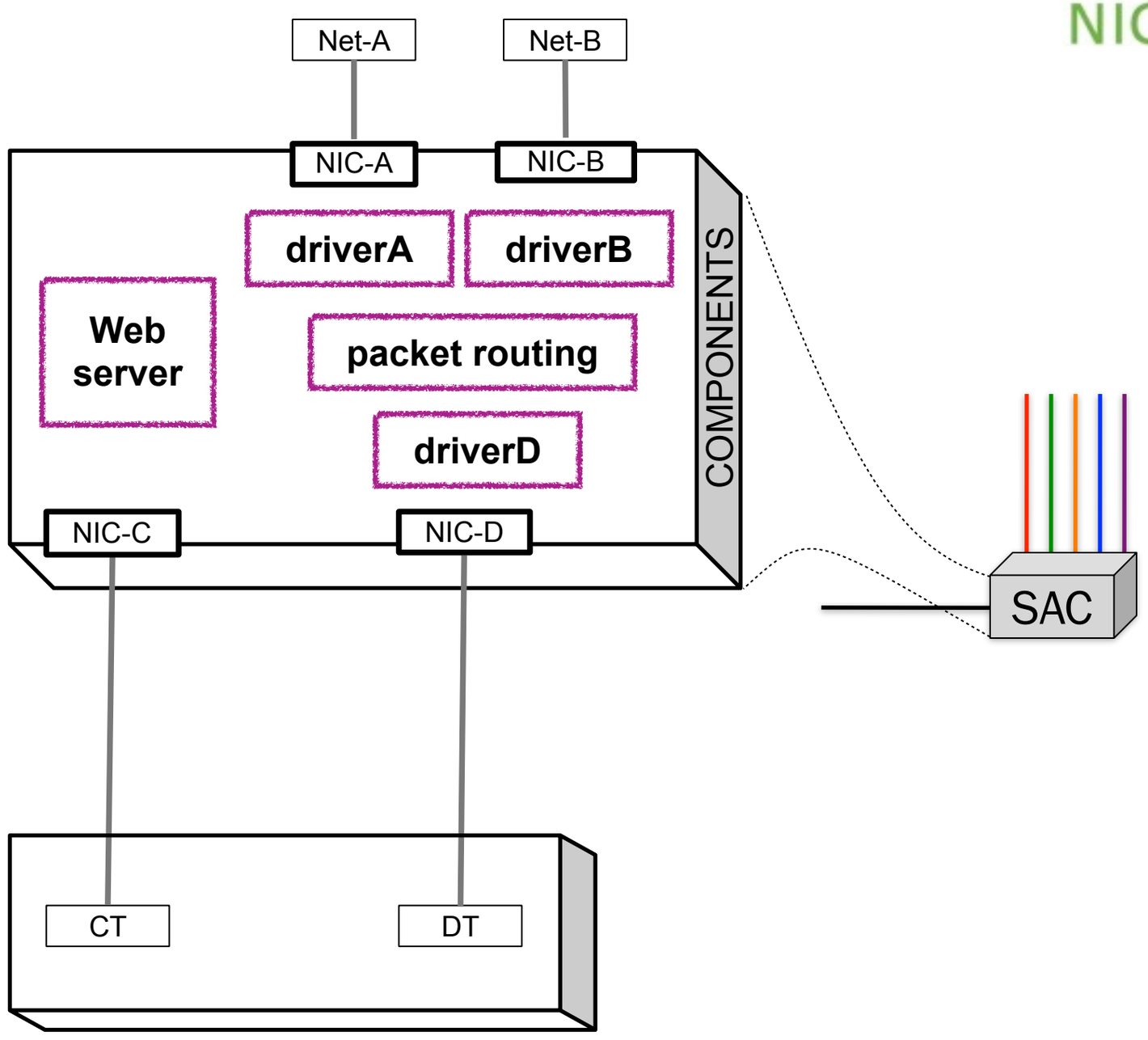


Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B

NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

SAC System

Components



Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B

NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

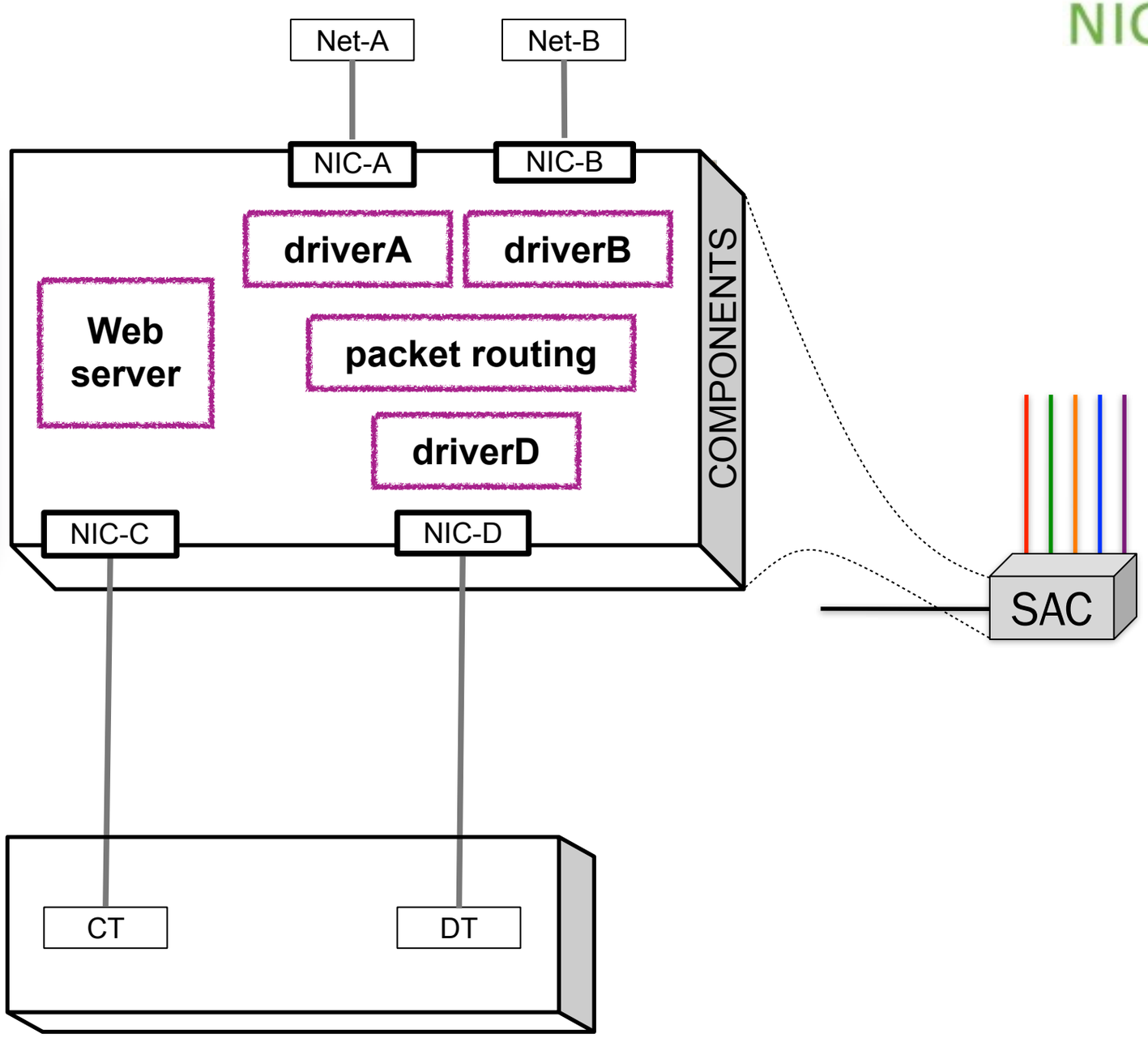
SAC System

Desired Property

No information flow
between providers **A** and **B**

through SAC

even if they collaborate



Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

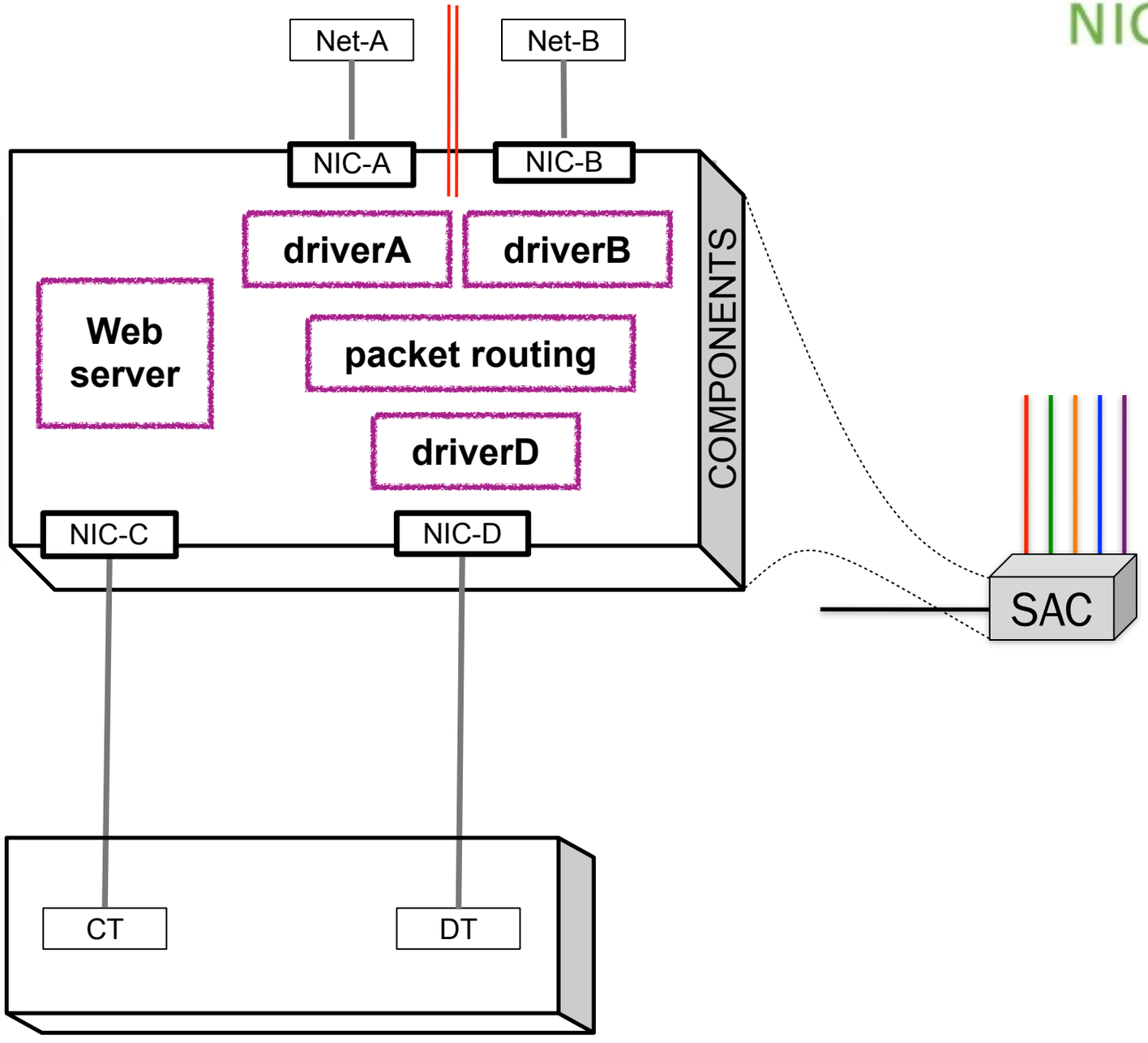
SAC System

Desired Property

No information flow
between providers **A** and **B**

through SAC

even if they collaborate



Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

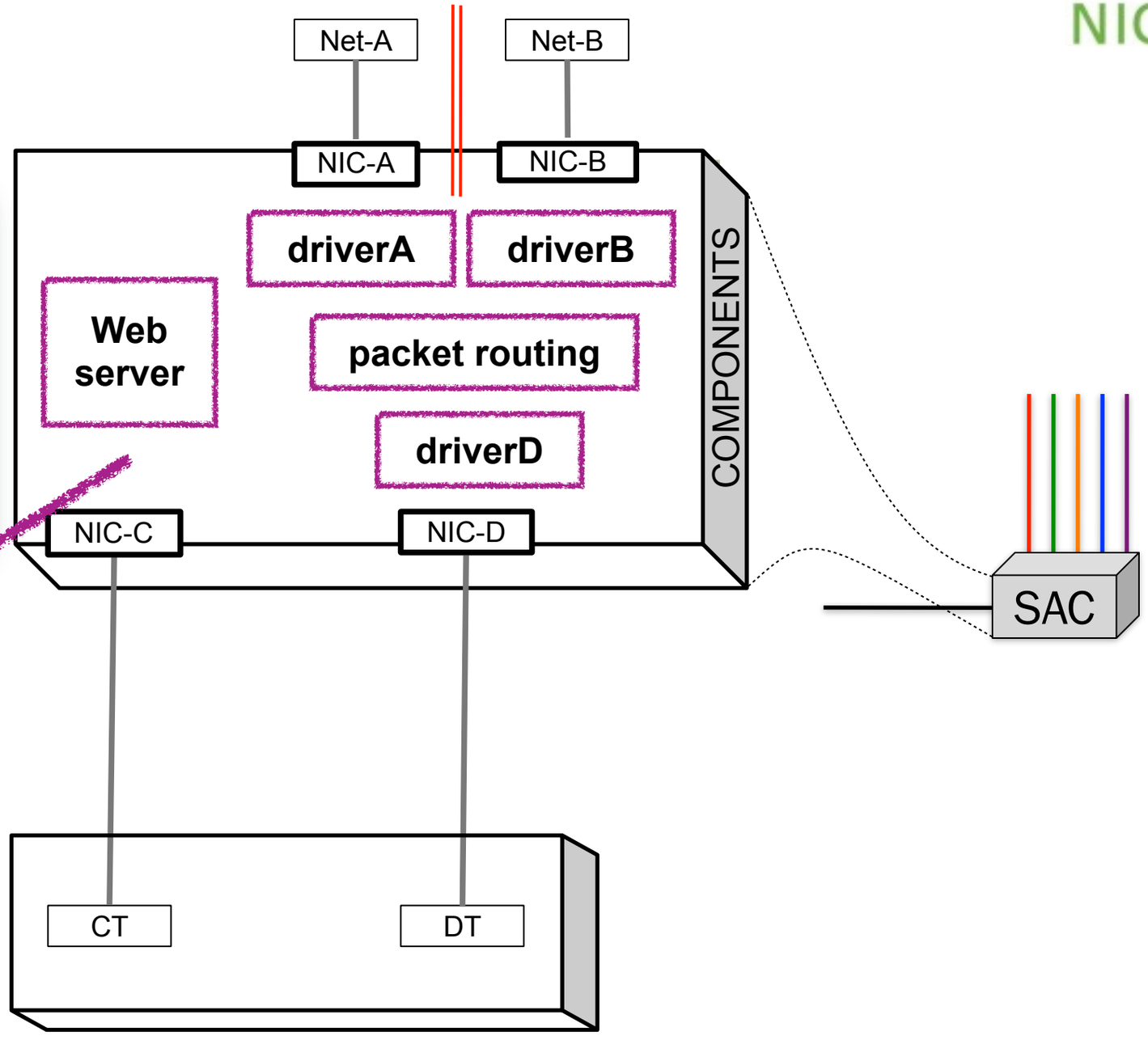
SAC System

Desired Property

No information flow
between providers **A** and **B**

through SAC

even if they collaborate



Proving all this correct?

Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B

NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

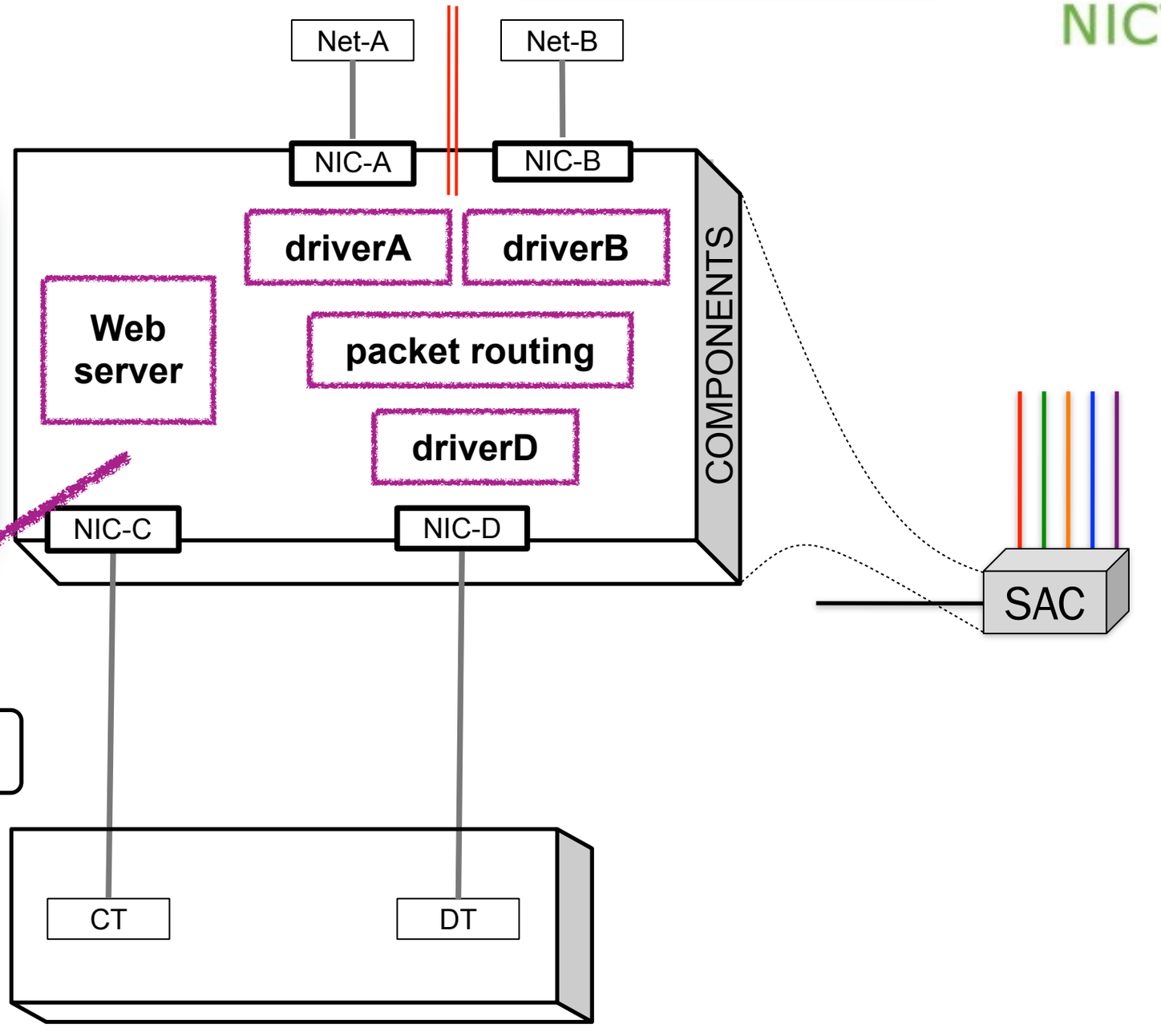
SAC System

Desired Property

No information flow
between providers **A** and **B**

through SAC

even if they collaborate



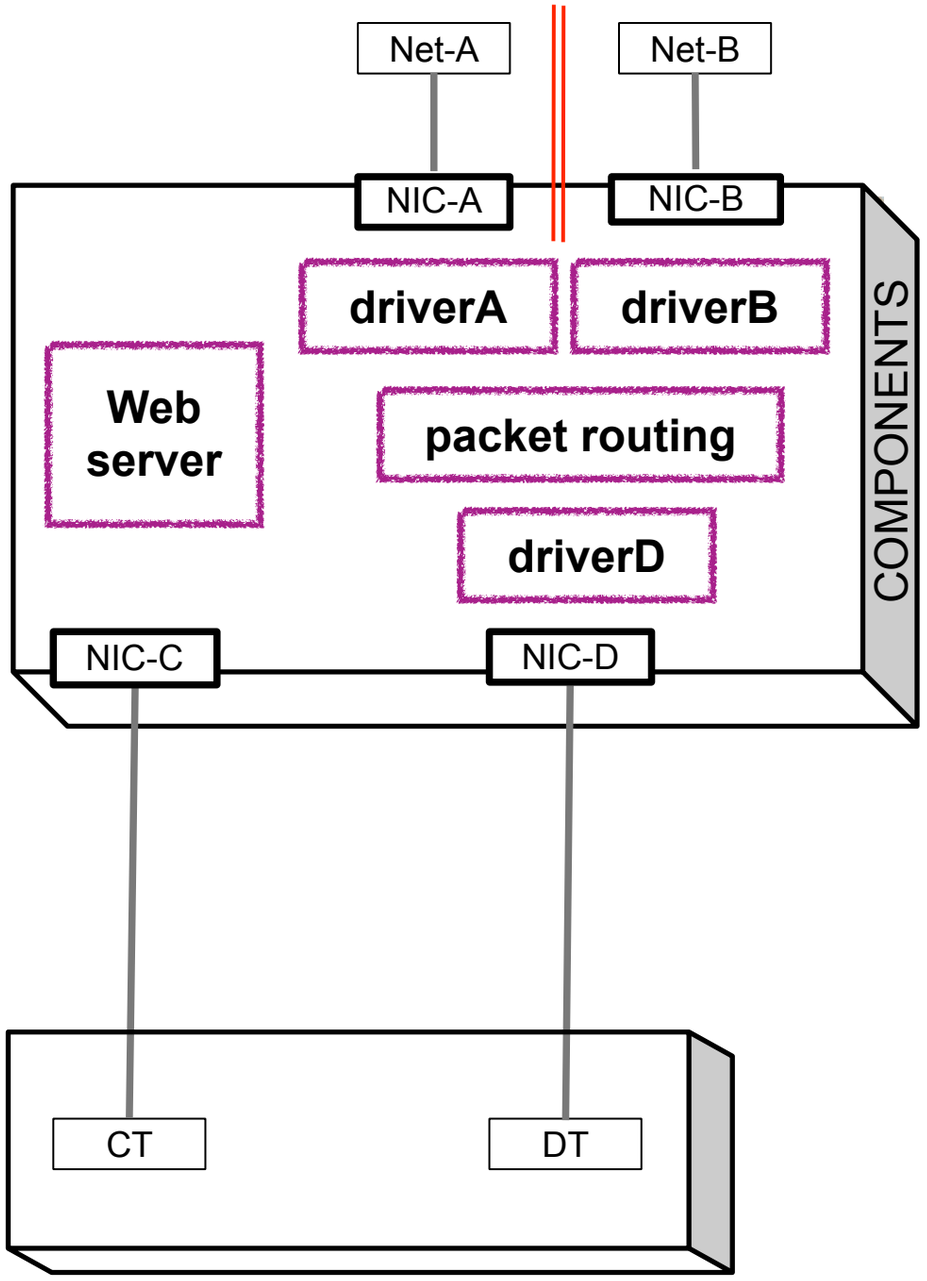
Proving all this correct?

Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B

NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

SAC System

Design

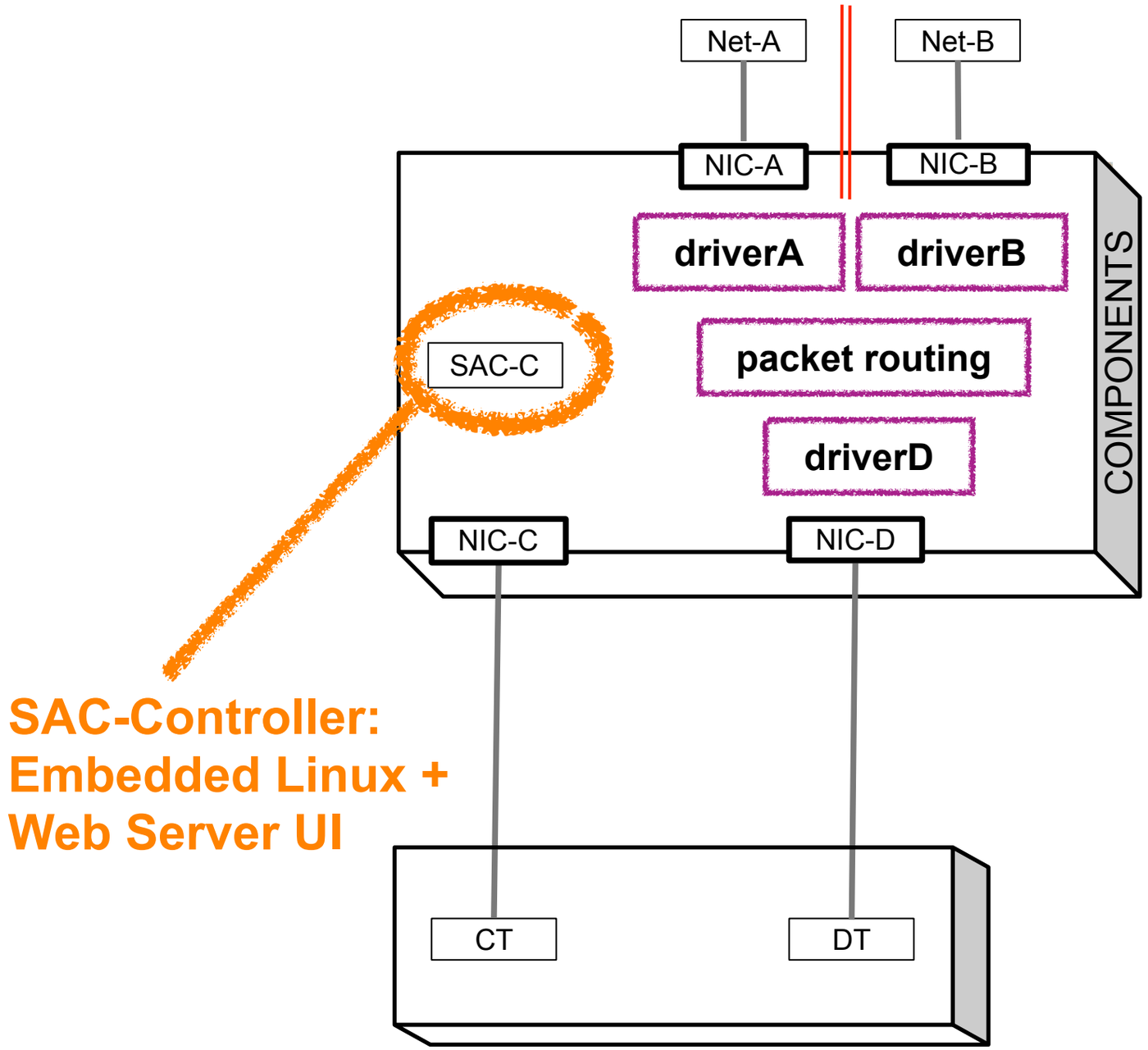


Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B

NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

SAC System

Design

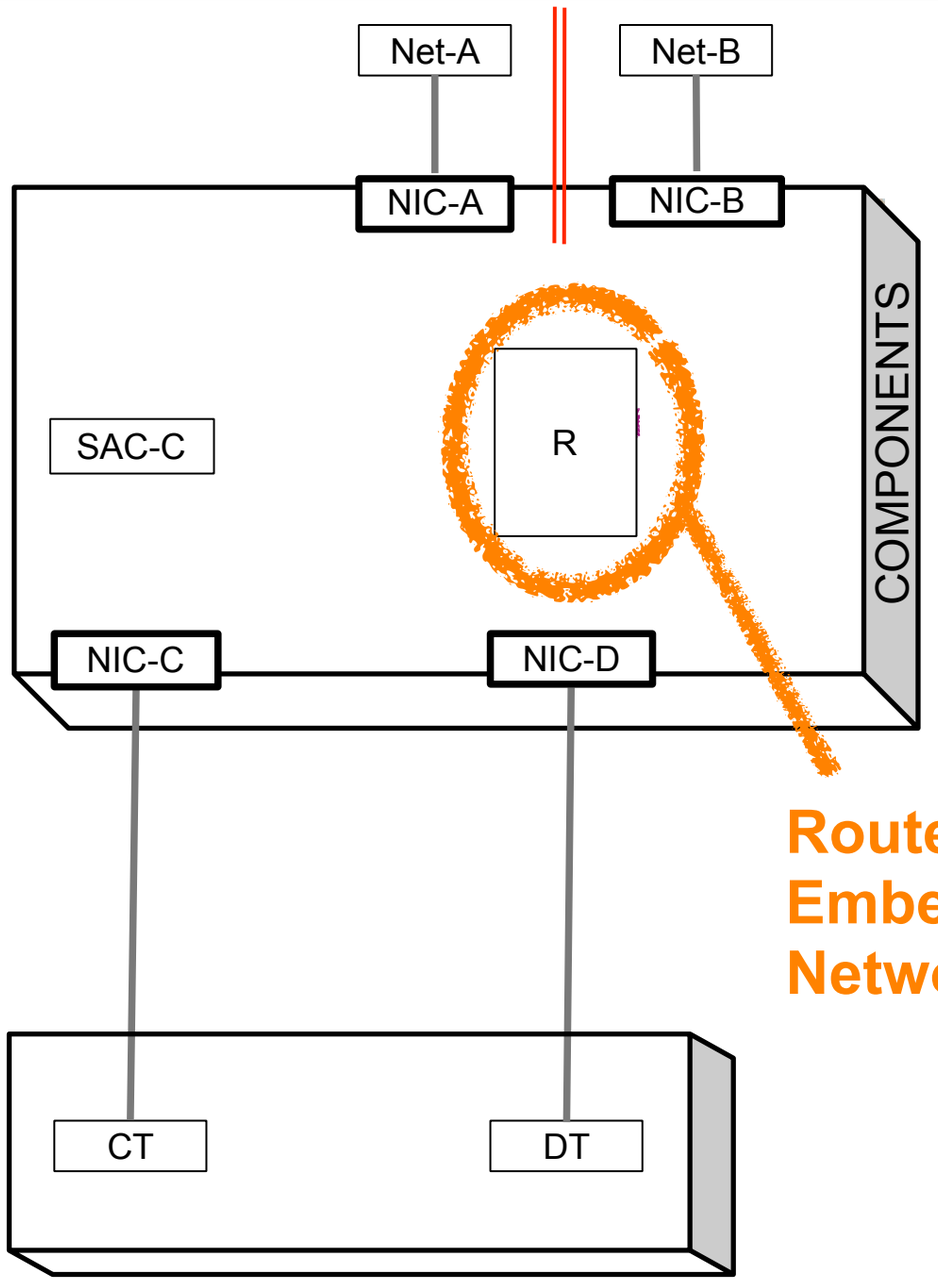


Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B

NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

SAC System

Design



Router:
Embedded Linux/
Network Routing + Drivers

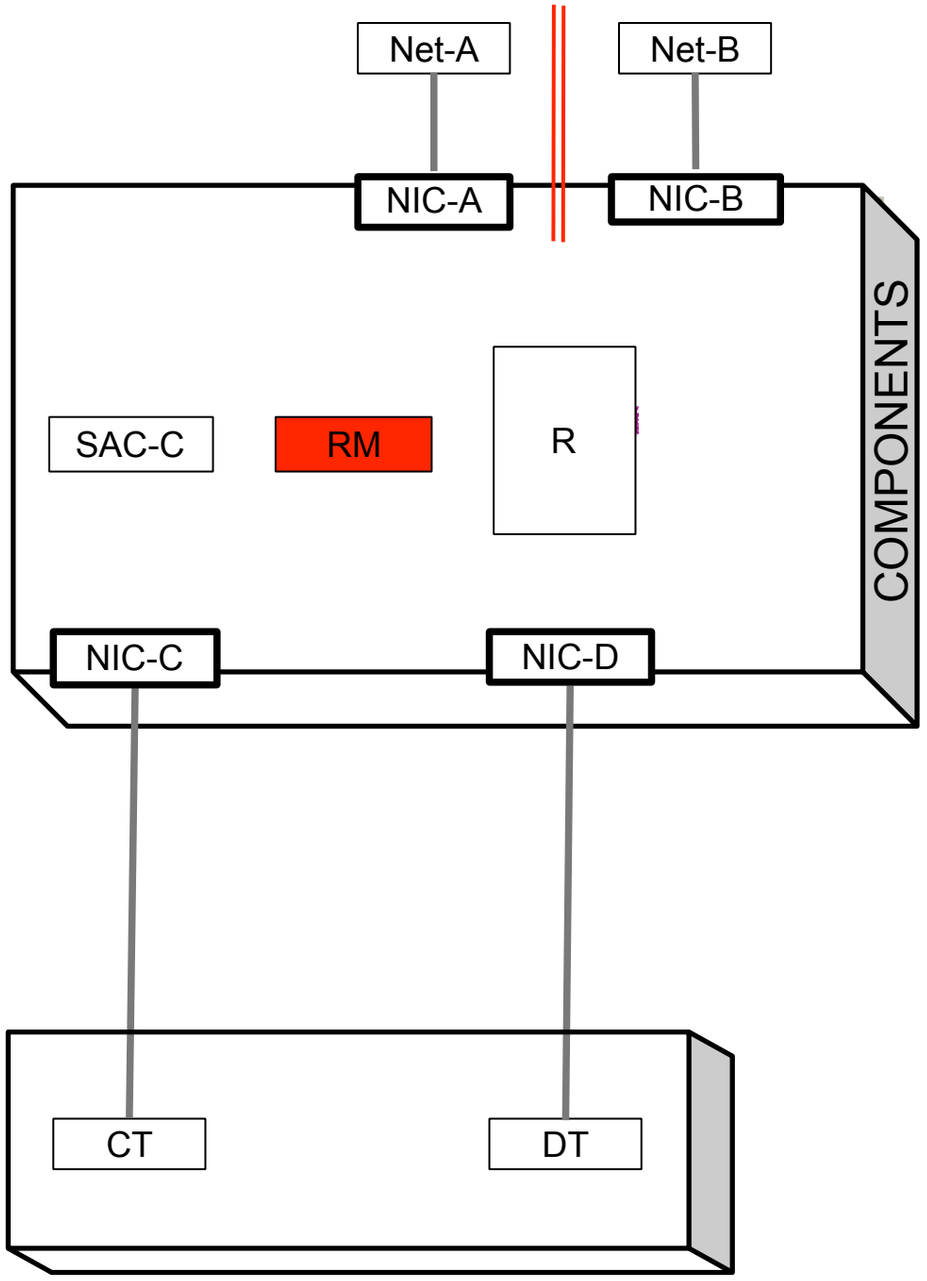
Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B

NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

R= Router
RM = Router Manager
SAC-C = SAC Controller

SAC System

Design



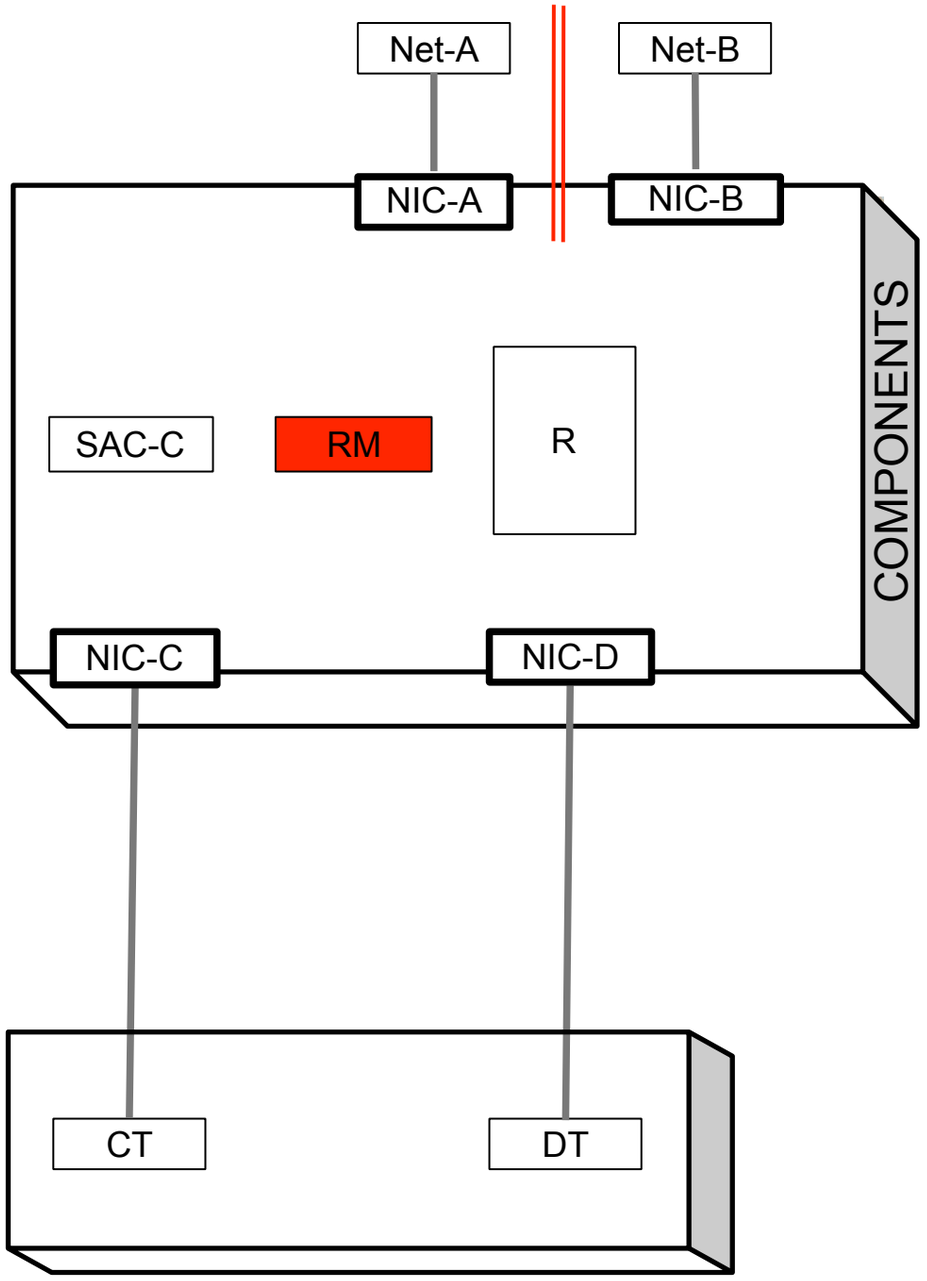
Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B

NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

R= Router
RM = Router Manager
SAC-C = SAC Controller

SAC System

Design



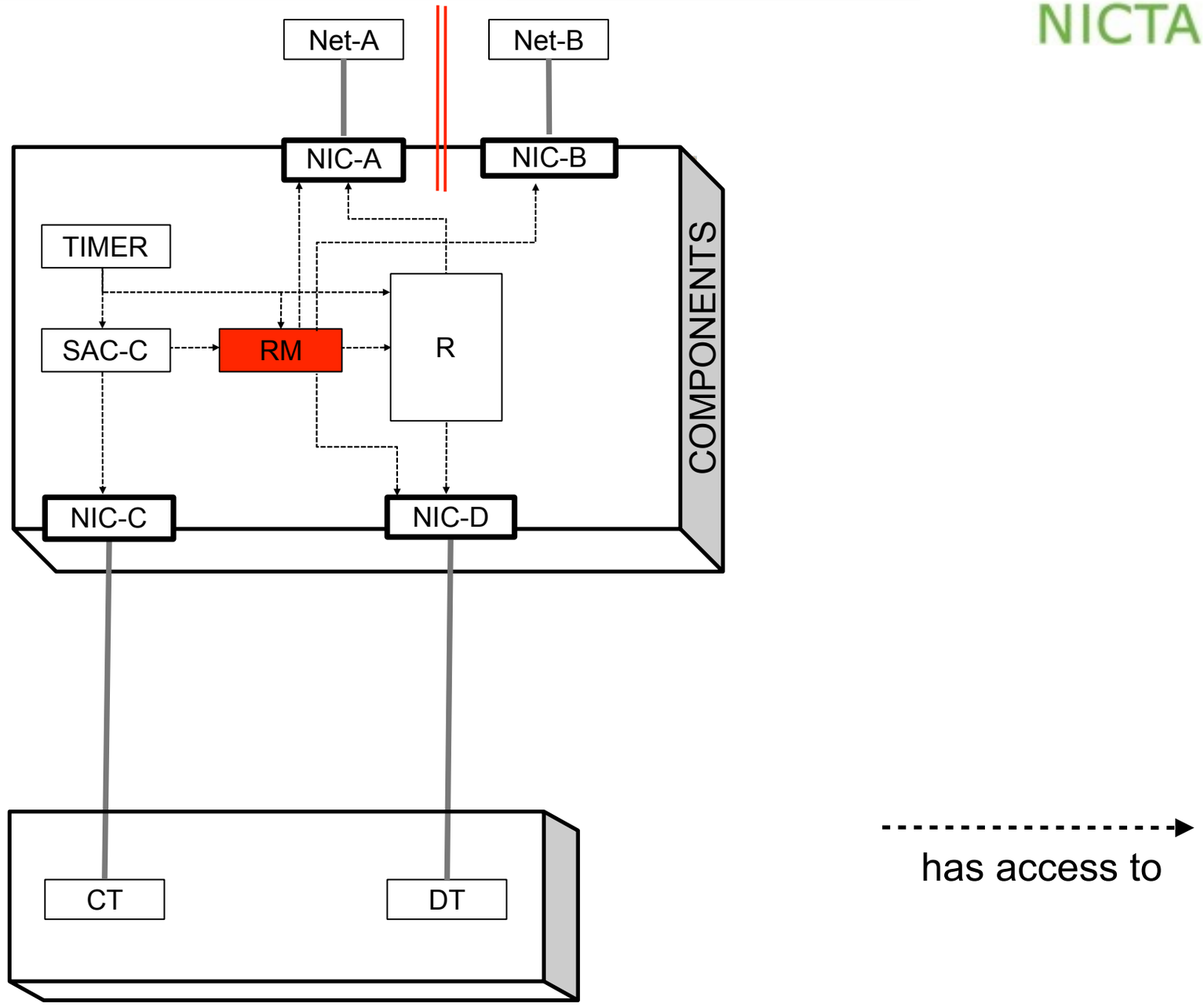
Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B

NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

R= Router
RM = Router Manager
SAC-C = SAC Controller

SAC System

Design



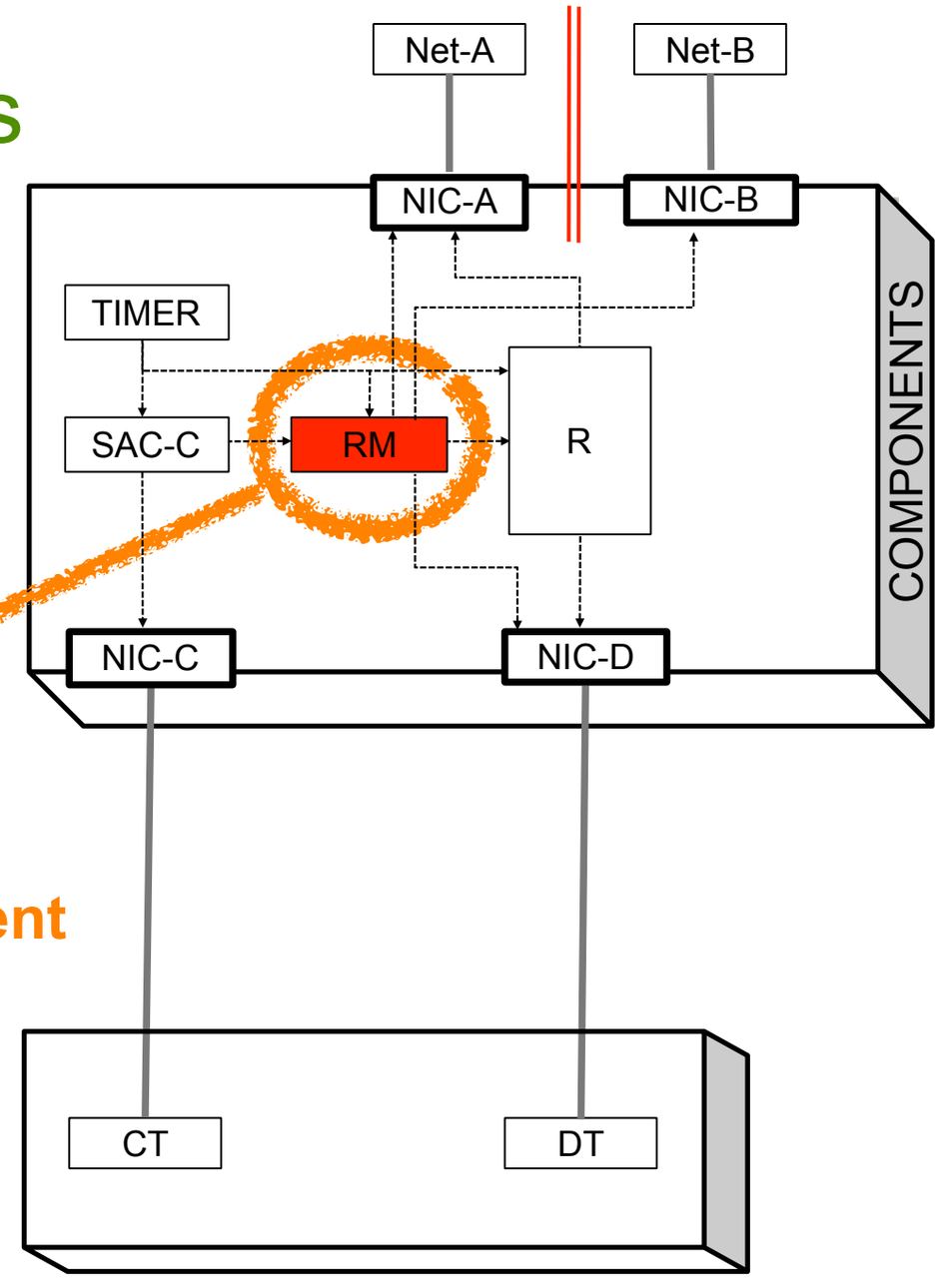
Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B

NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

R= Router
RM = Router Manager
SAC-C = SAC Controller

SAC System

Trusted Components



Router Manager:
< 2kloc
only trusted component

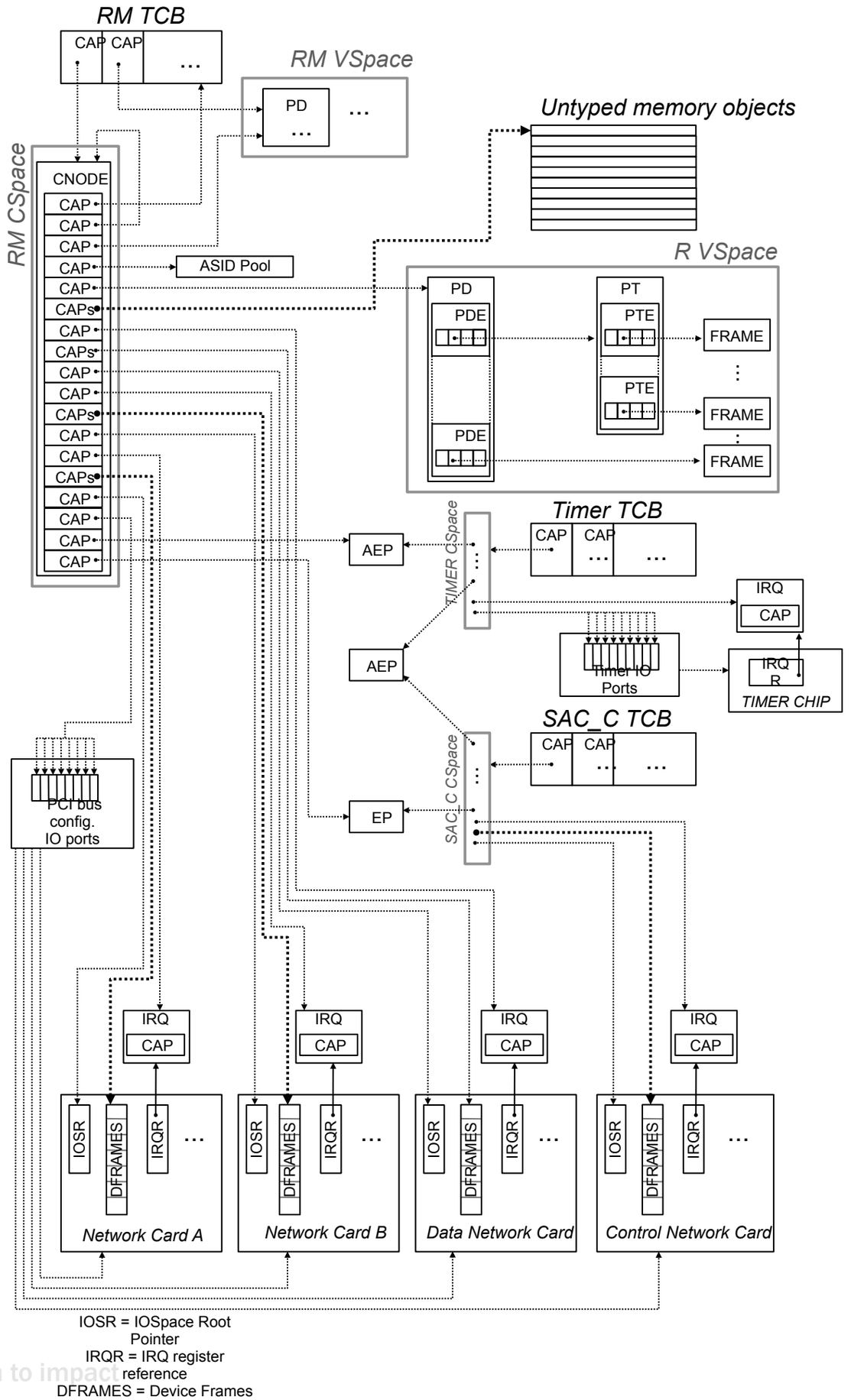
has access to

Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B

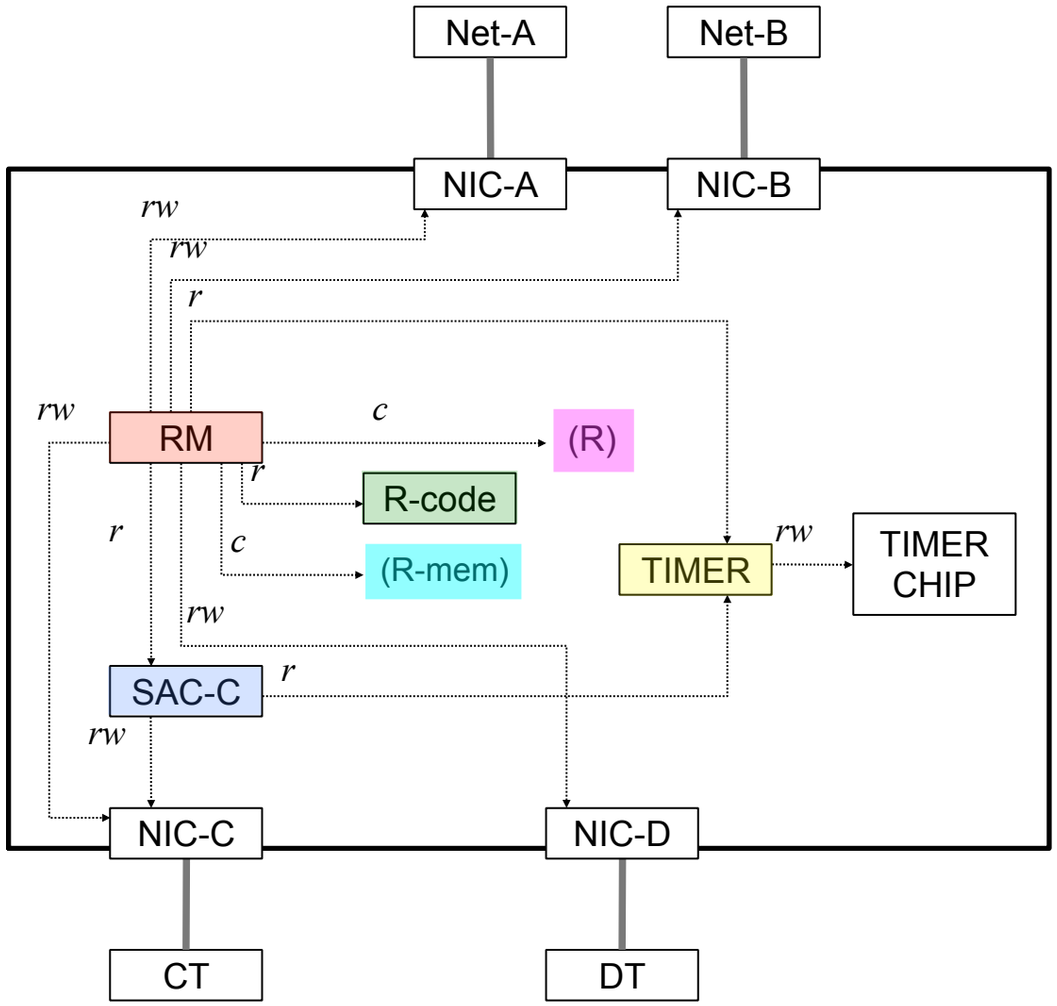
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

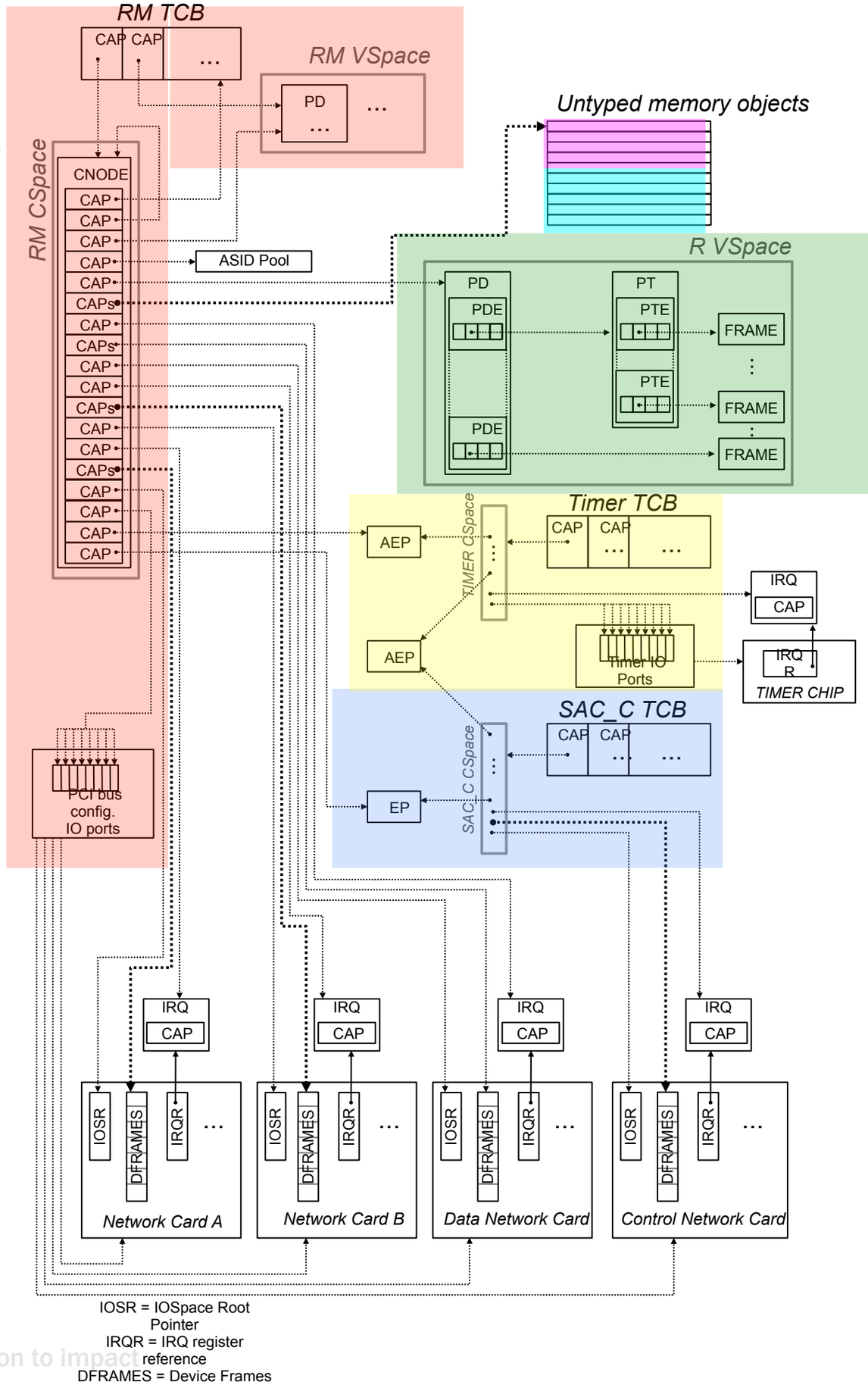
R= Router
RM = Router Manager
SAC-C = SAC Controller

Low-Level Design

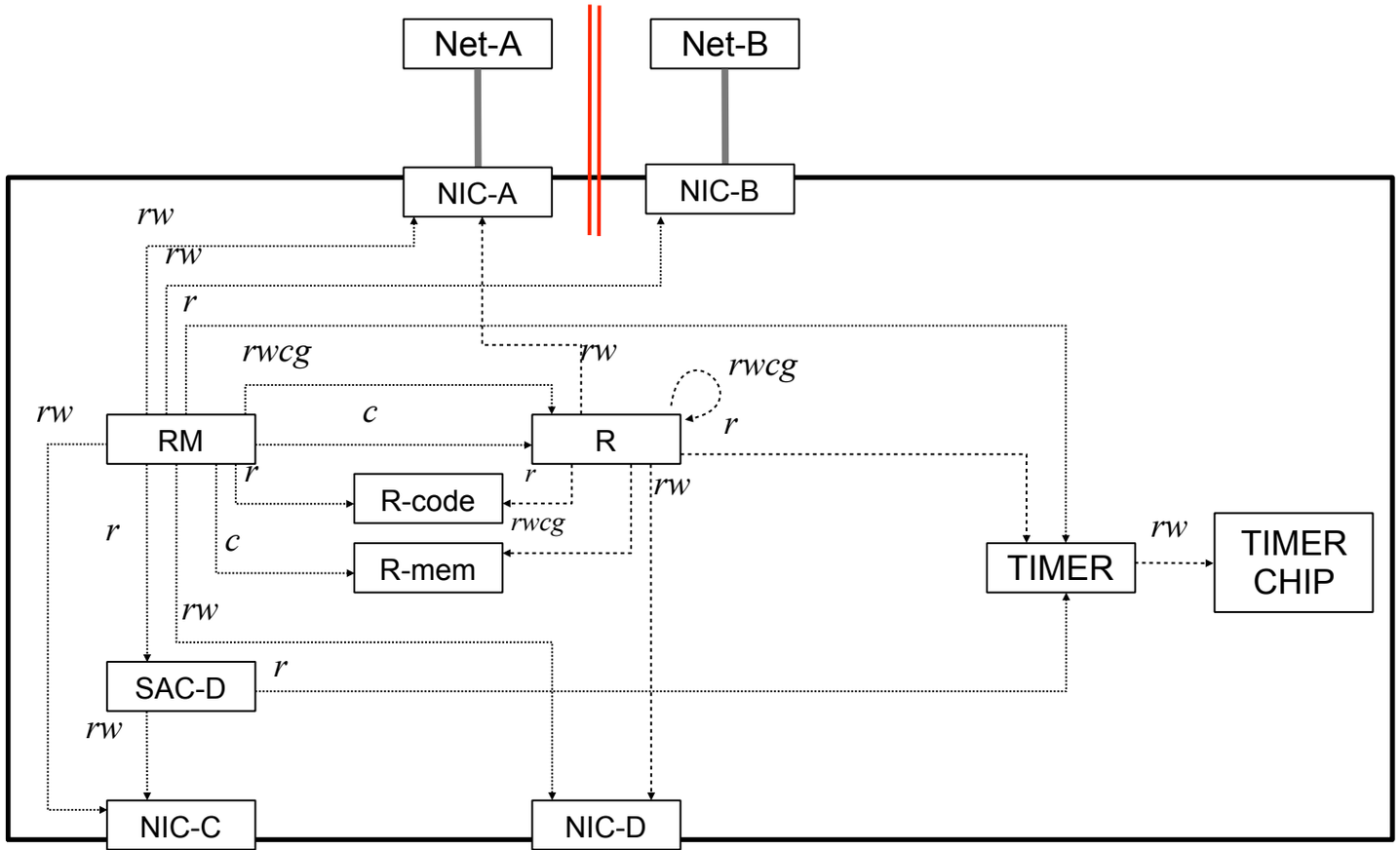


Abstraction





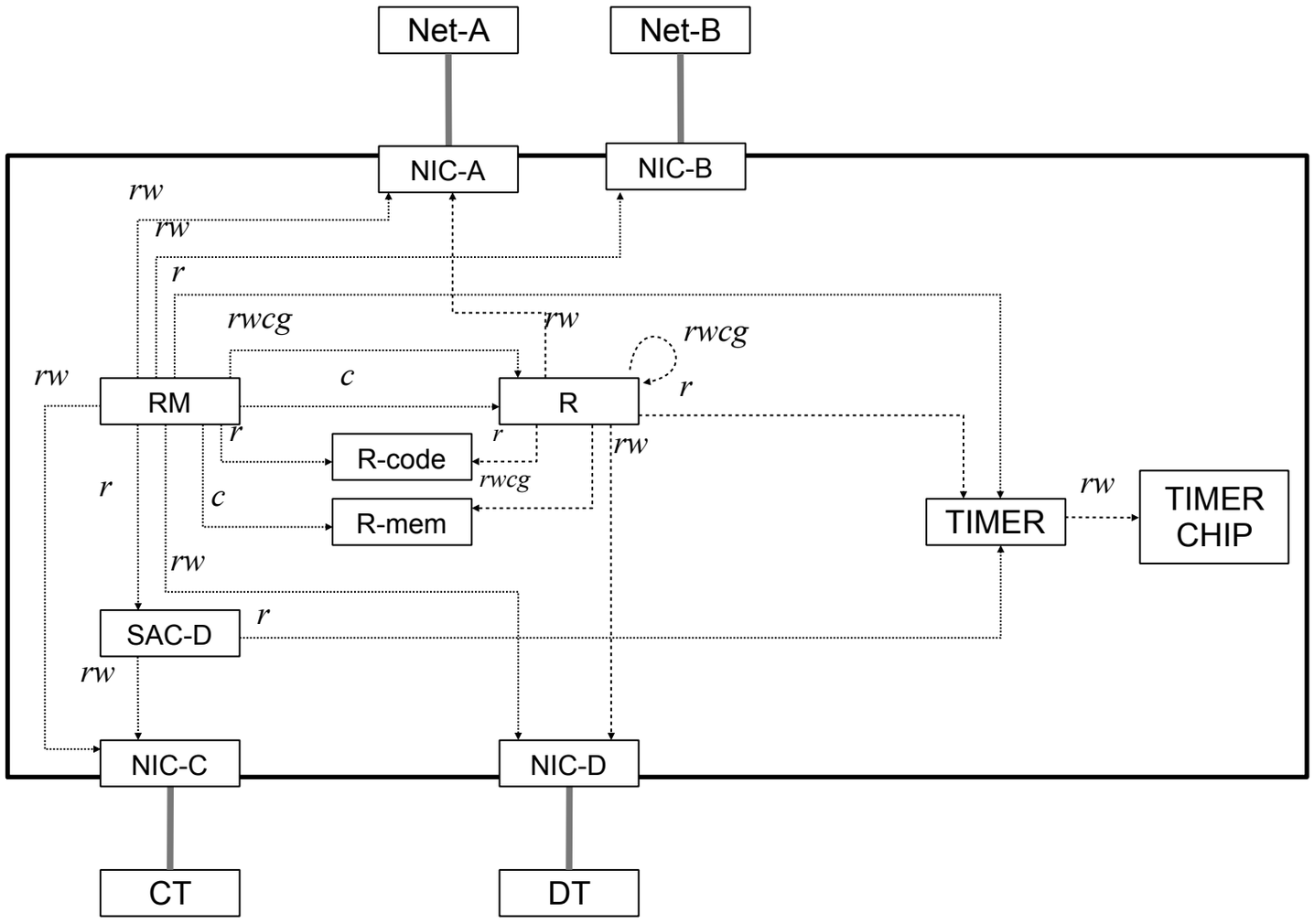
Security Goal



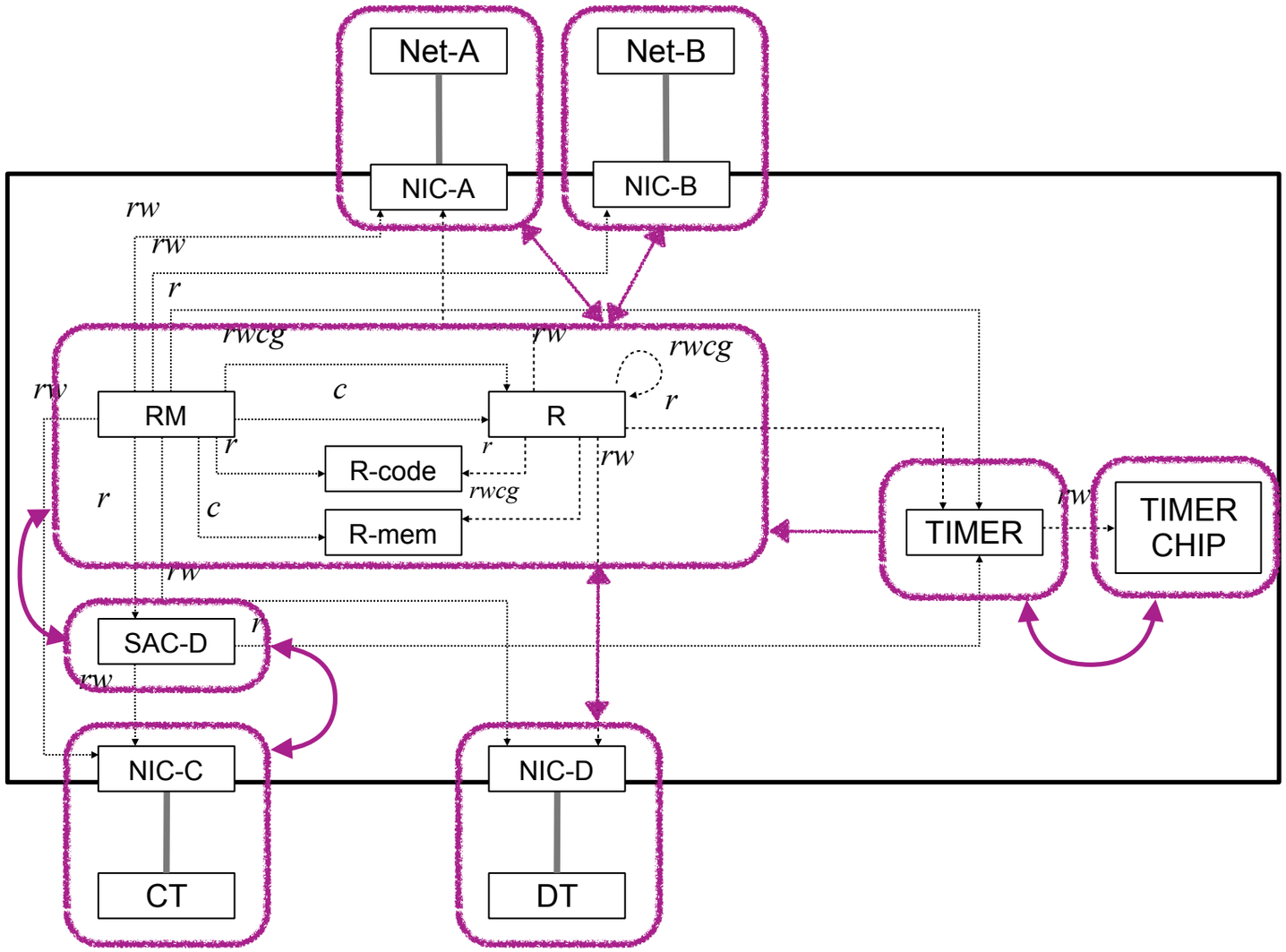
Goal: No information flowing between providers A and B

Assumption: Info flow through front-end terminal is trusted

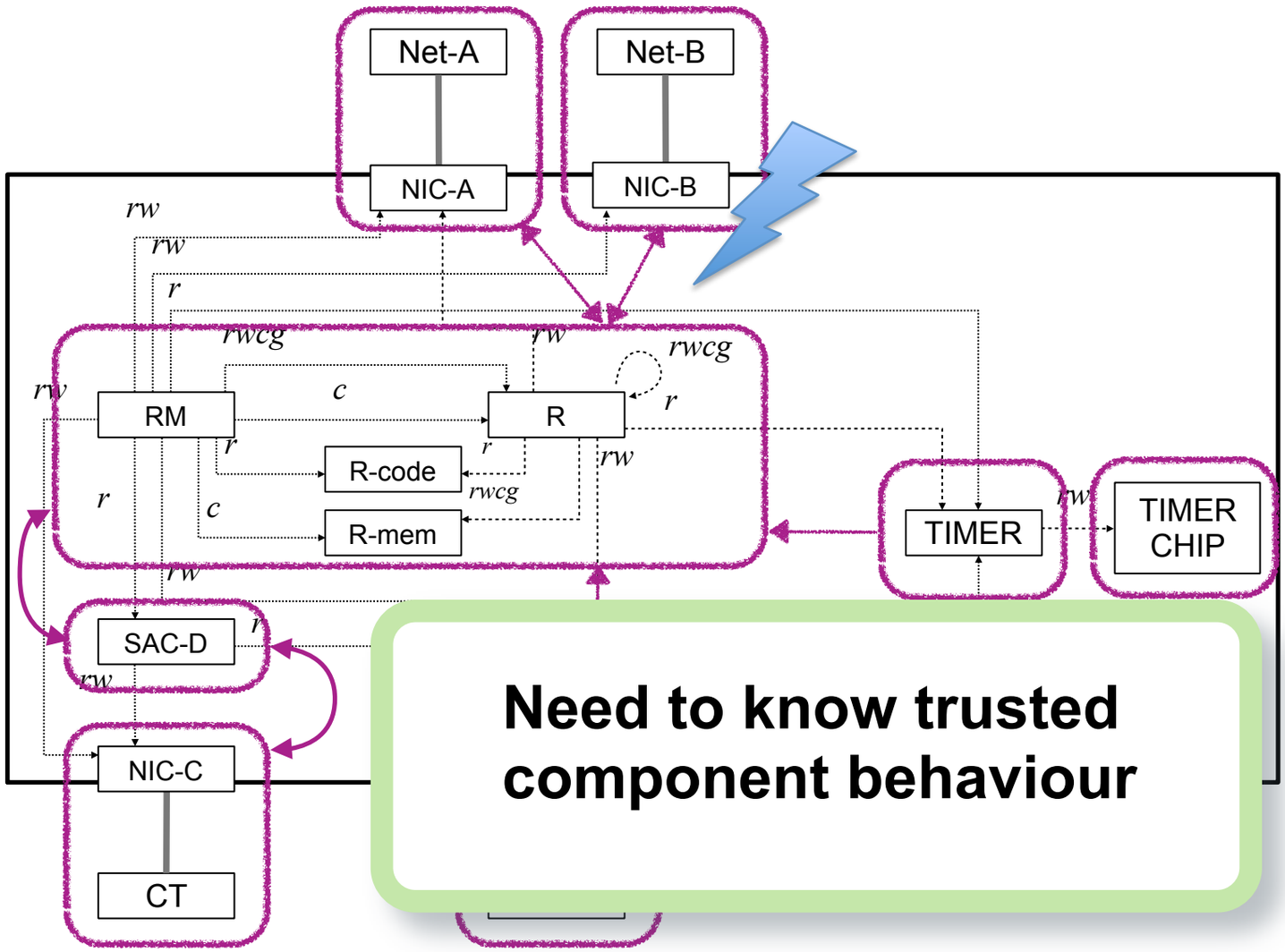
Plain Take-Grant Analysis



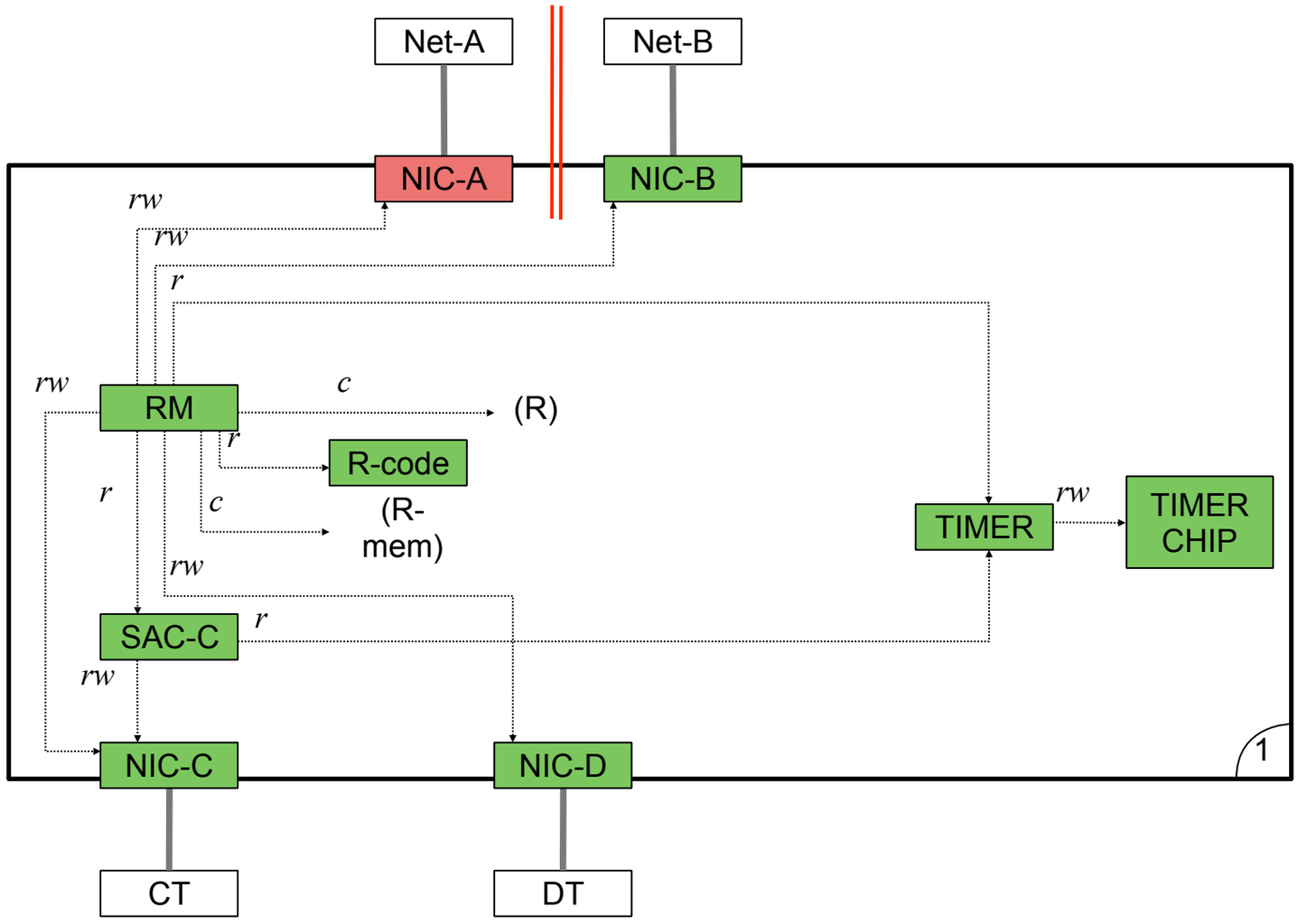
Plain Take-Grant Analysis



Plain Take-Grant Analysis



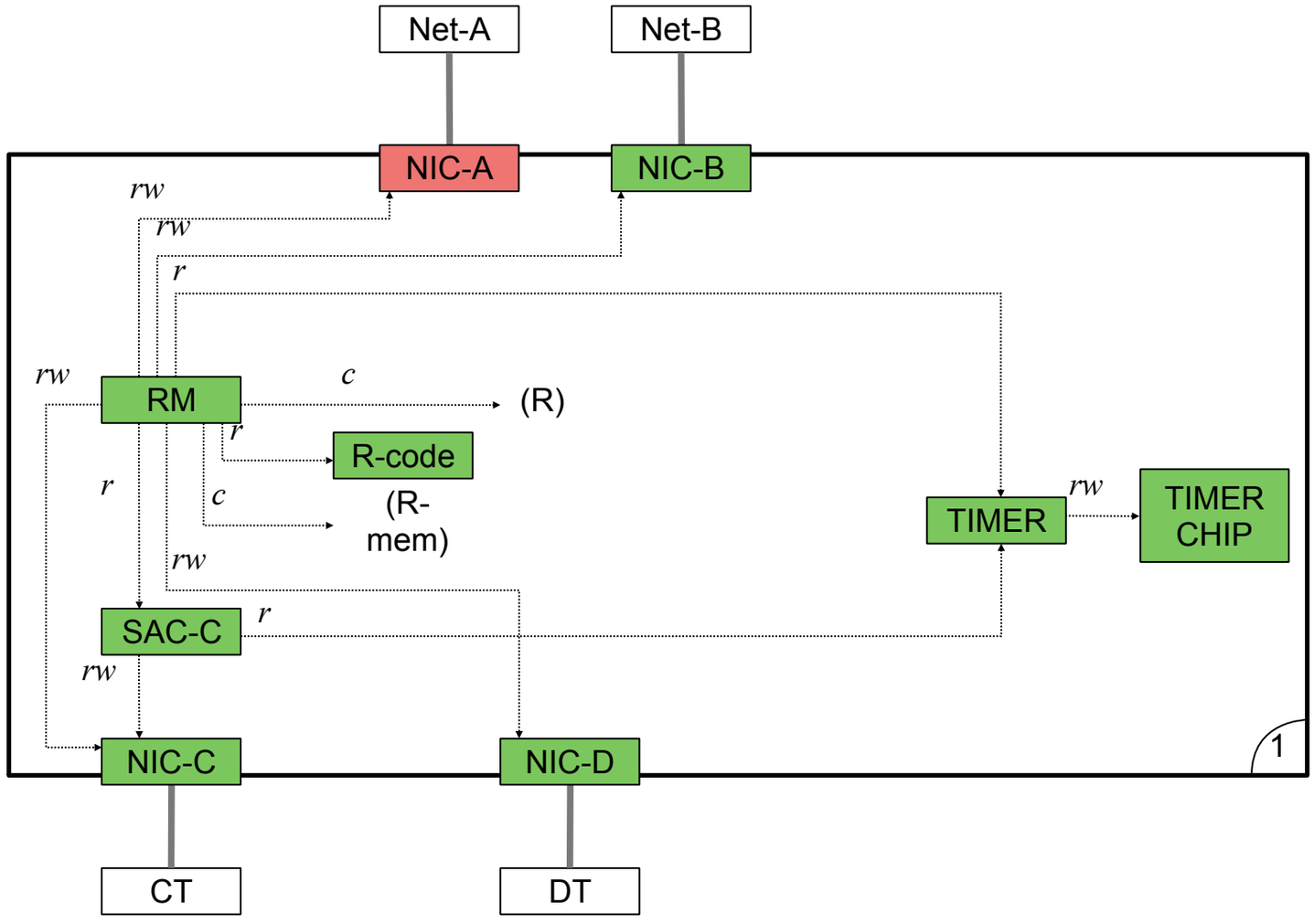
Security Goal



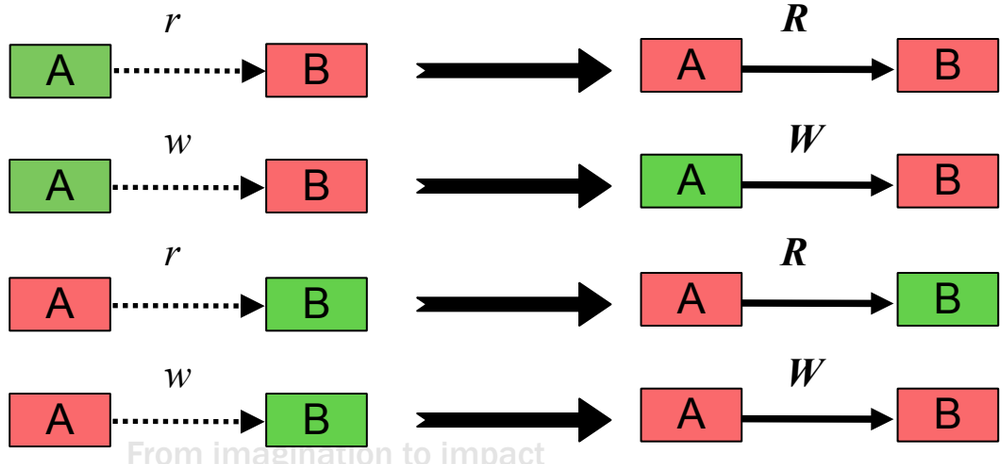
Approach:

- label-based security
  - tag as ‘contaminated’ if may contain data from Net-A
  - NIC-A always contaminated
- Goal: prove NIC-B always ‘not contaminated’

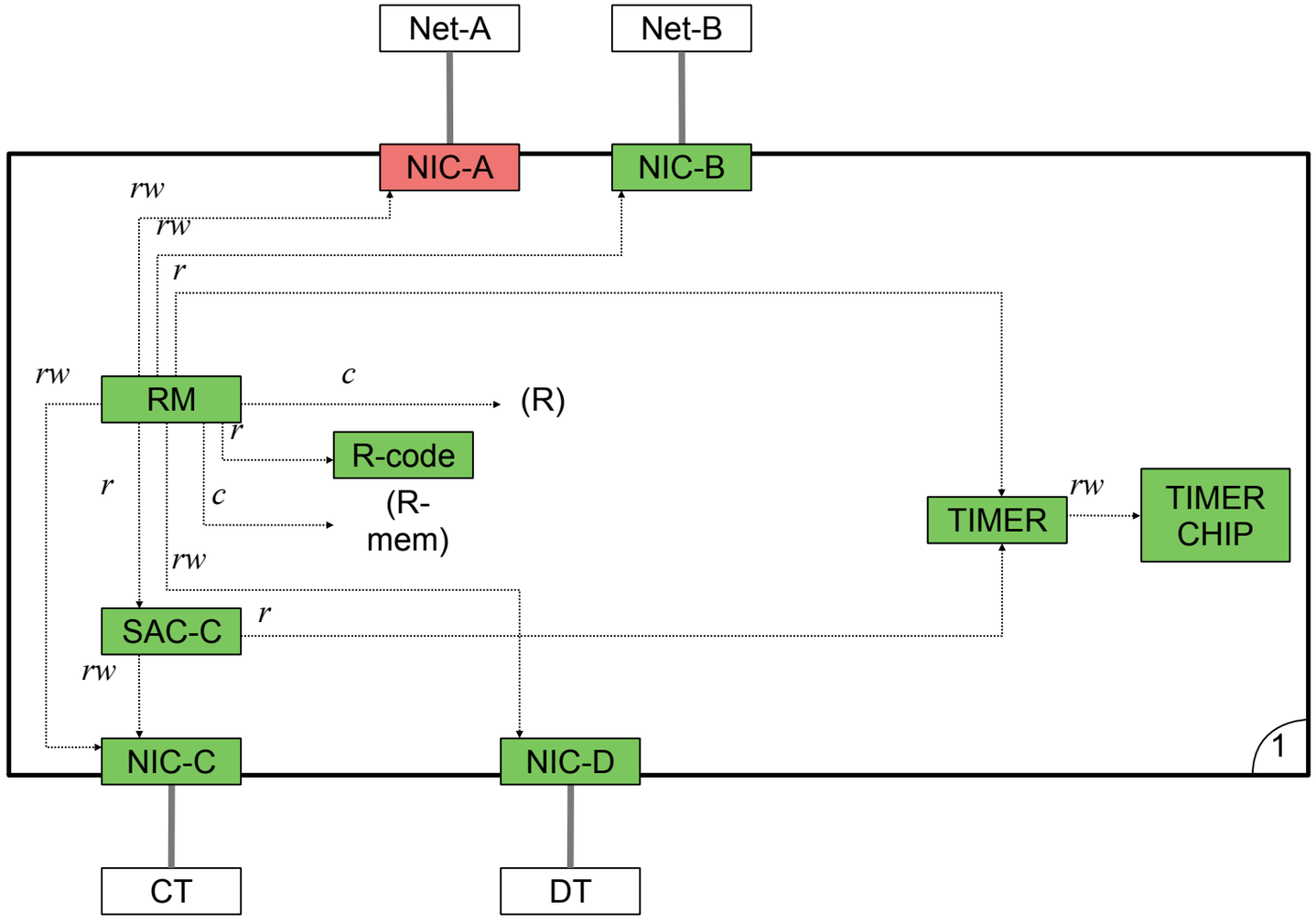
Security Analysis

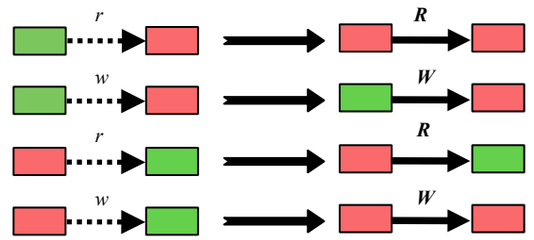


Rules:



Life Cycle

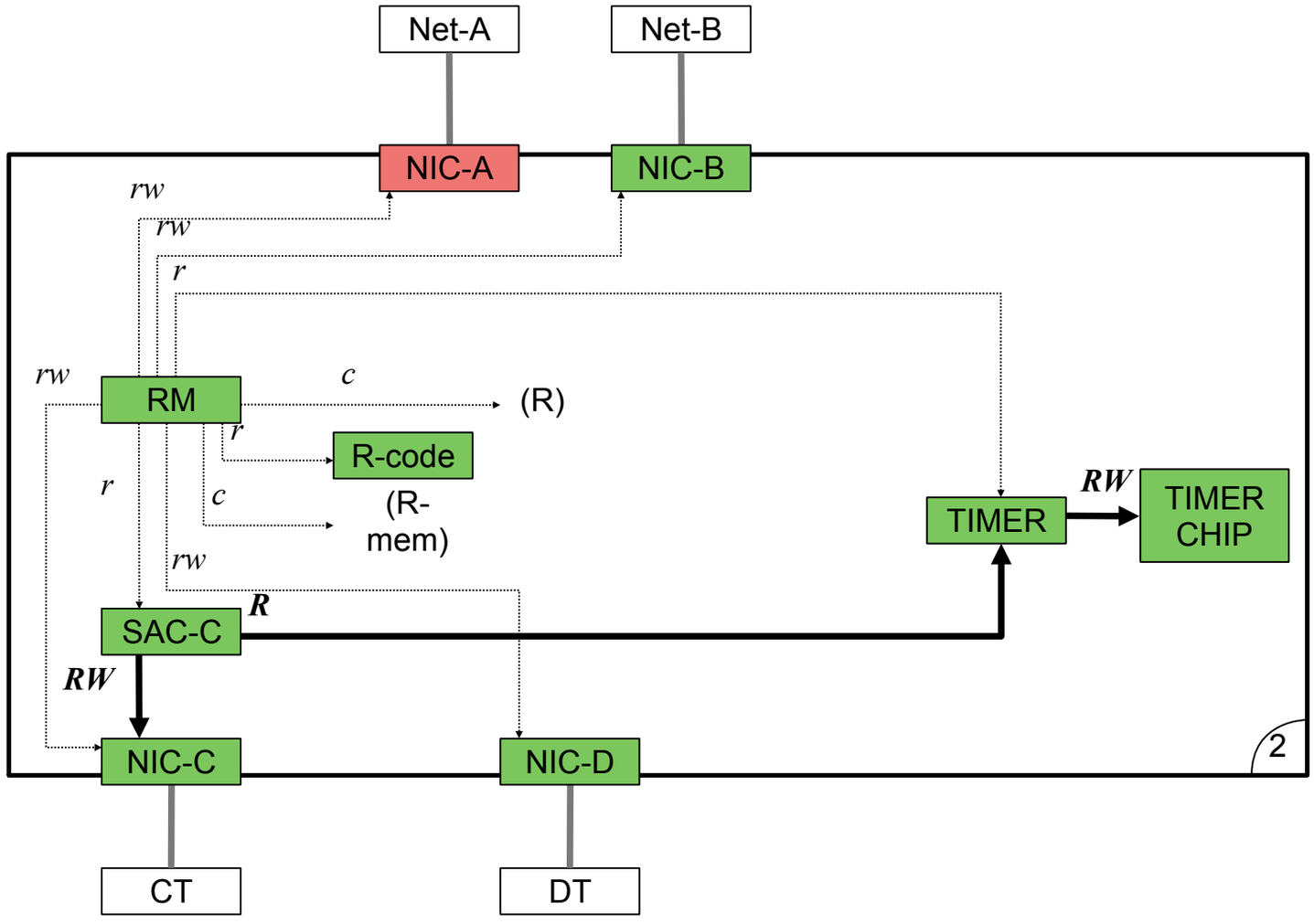


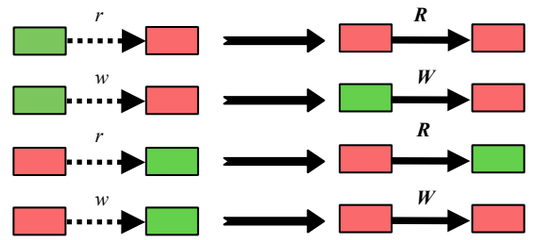


Life Cycle

CT authenticates with SAC-C

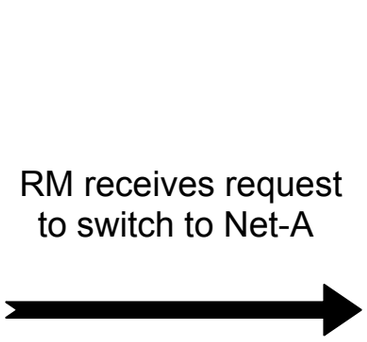
CT sends a request to switch to Net-A

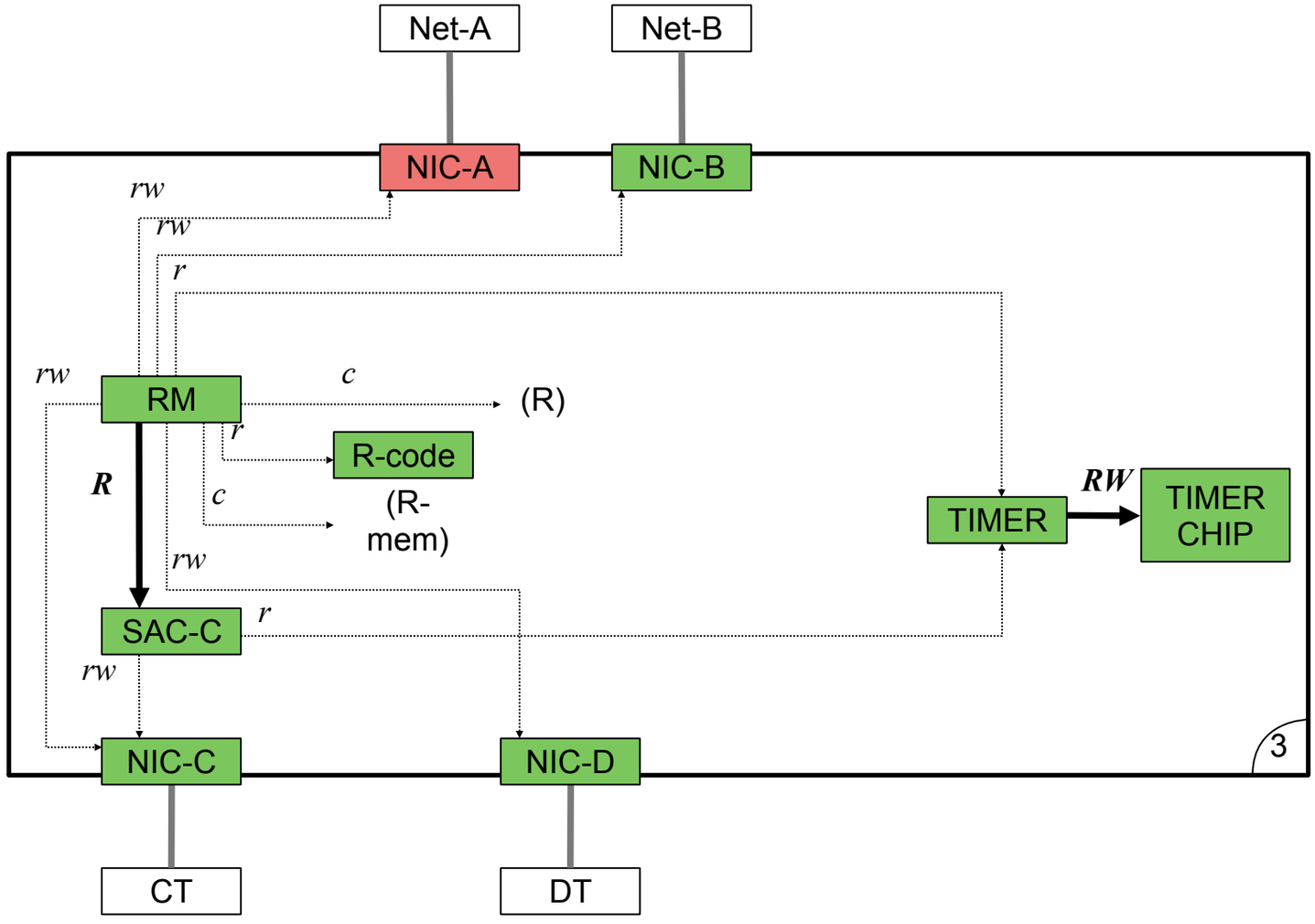


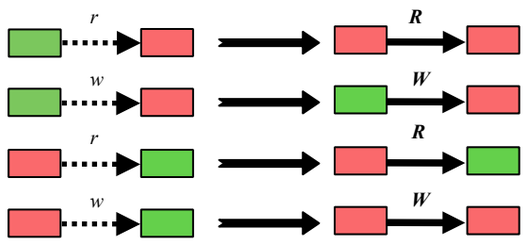


Life Cycle

RM receives request to switch to Net-A



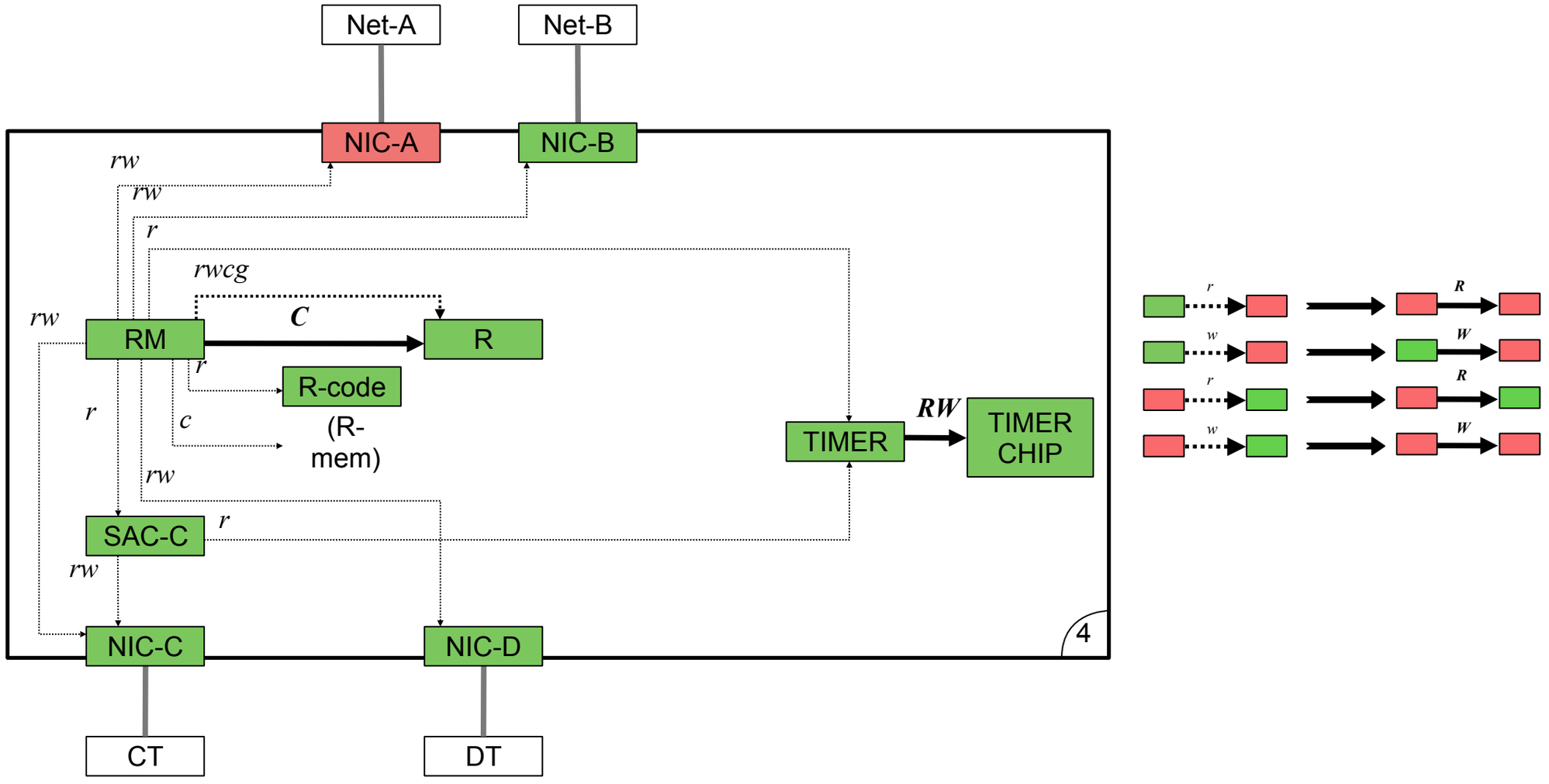




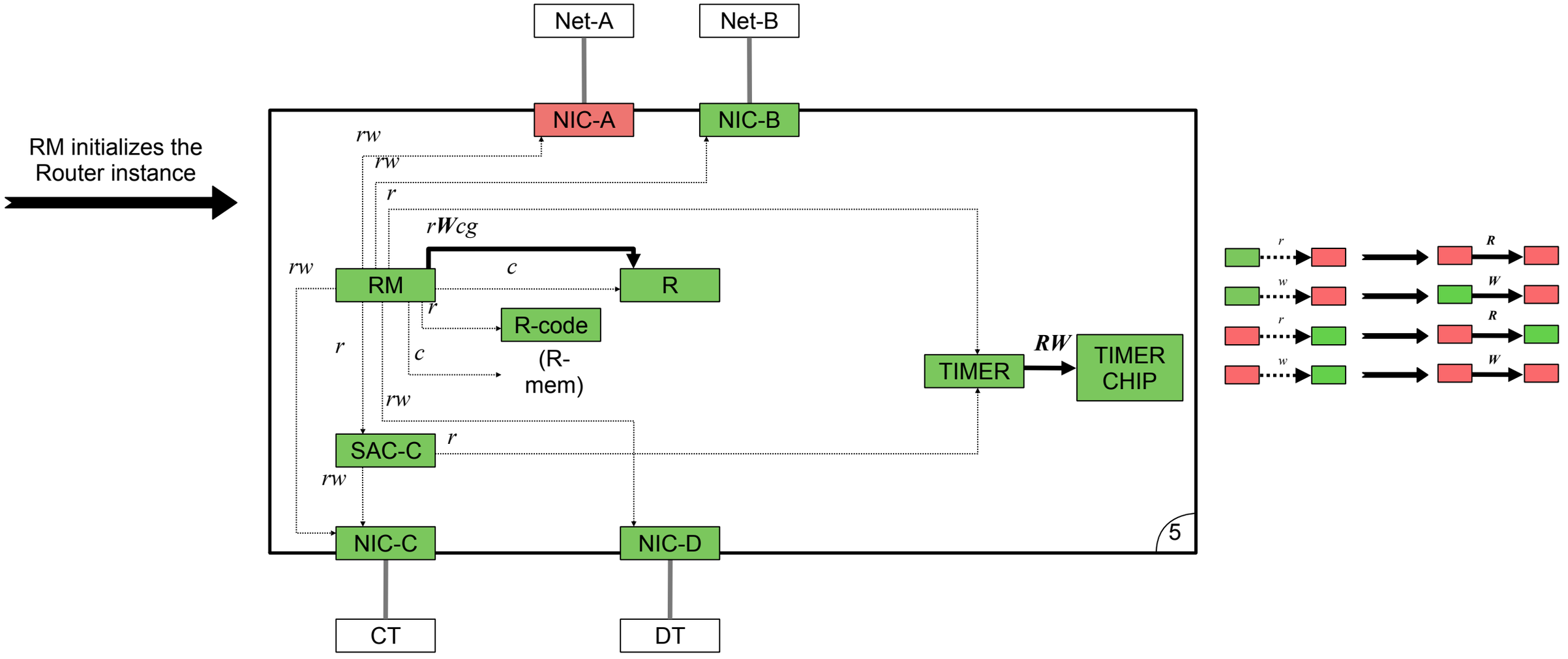
Life Cycle

RM creates a new Router instance R

(and gets full rights to the newly created object)



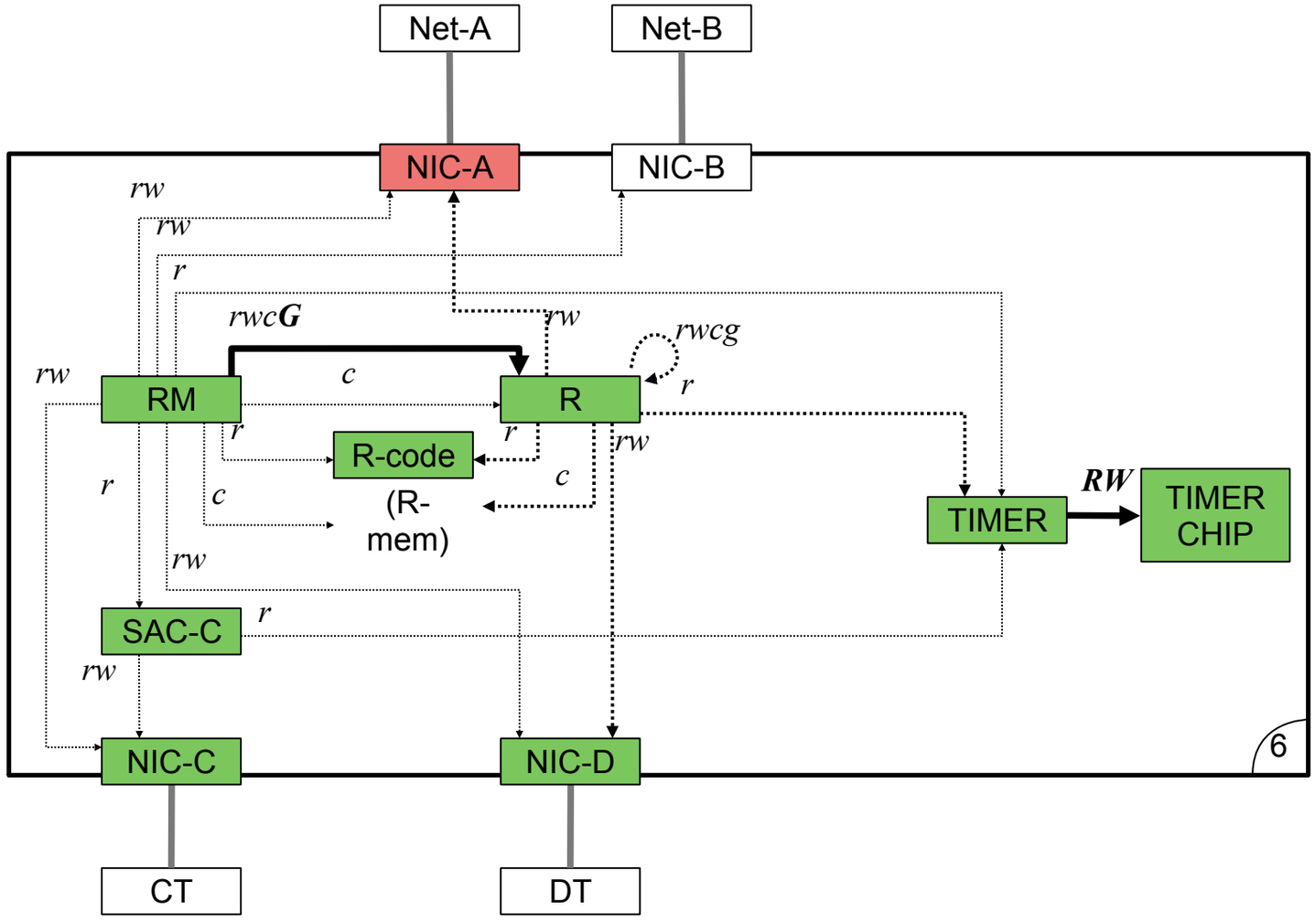
Life Cycle

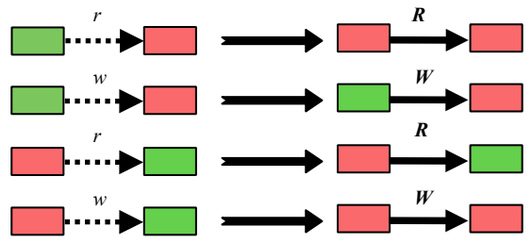


Life Cycle

RM grants to R its rights to NIC-A, NIC-D,

R-mem, R-code TIMER, and itself

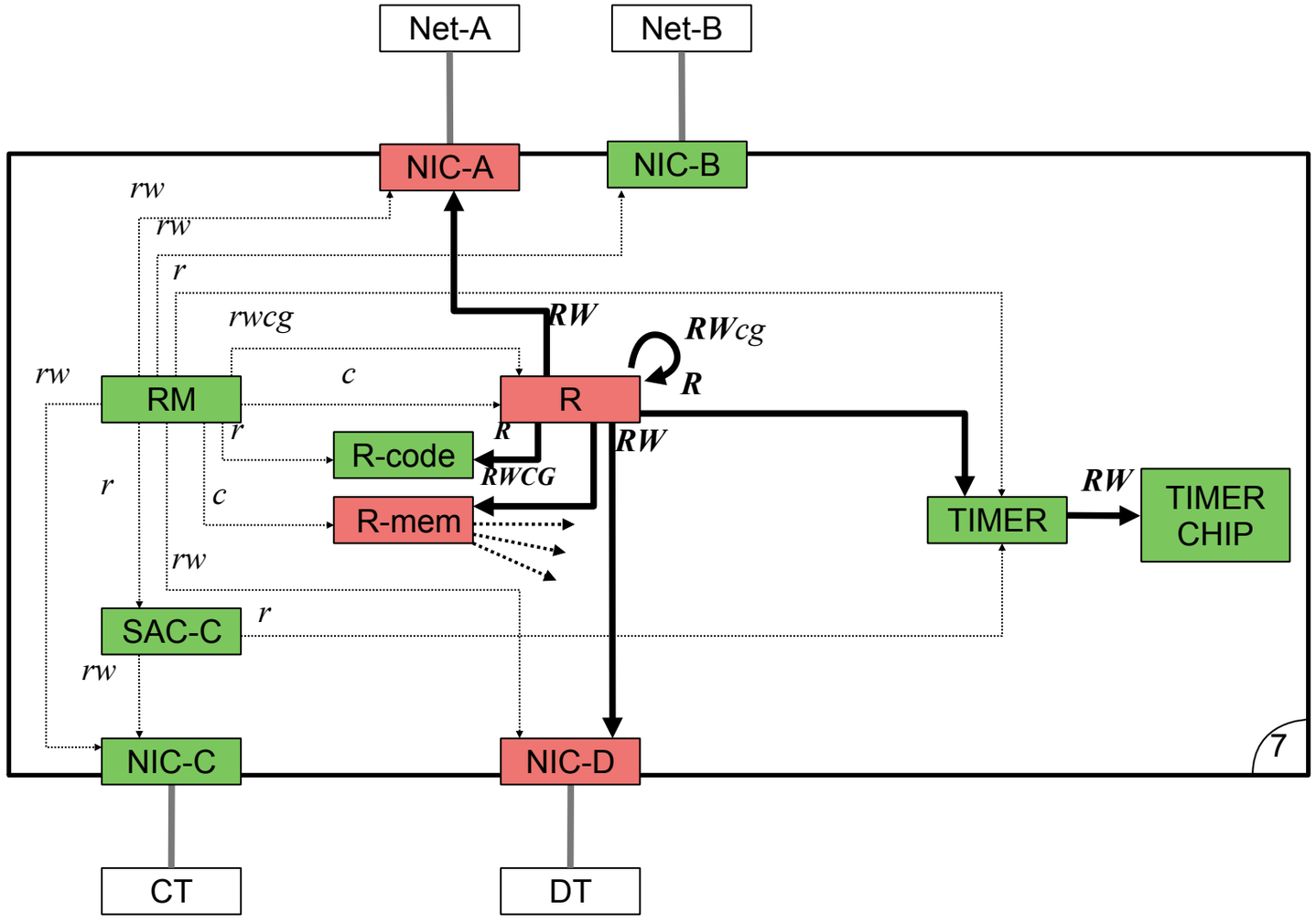


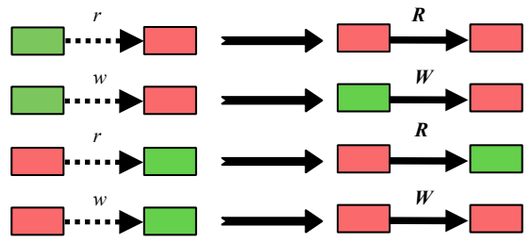


Life Cycle

DT starts to communicate with Net-A, through R.

This may imply the creation of new objects using R-mem and granting caps to them.

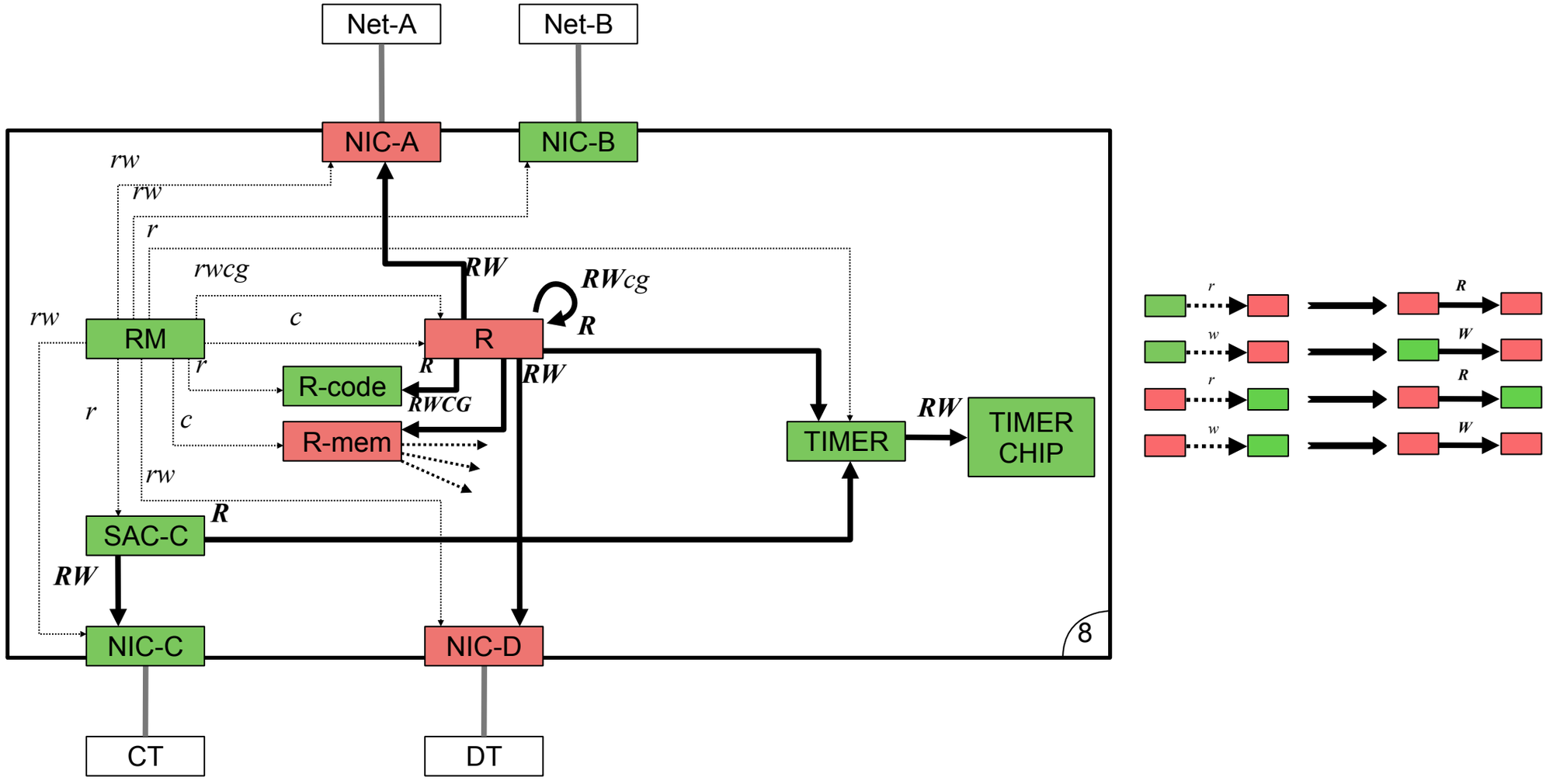




Life Cycle

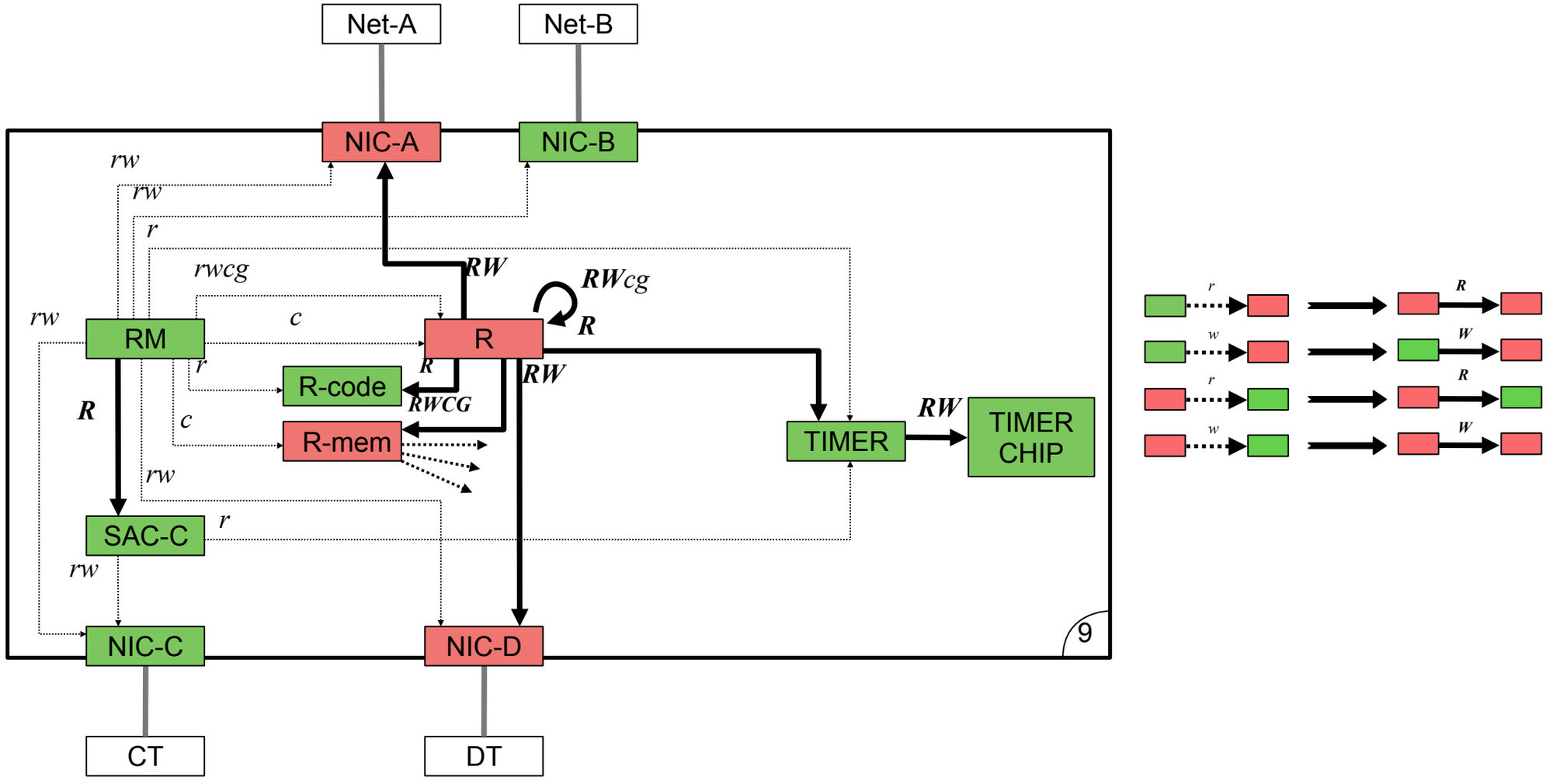
CT sends a request to switch to Net-B

(while DT still communicates with Net-A through R)



Life Cycle

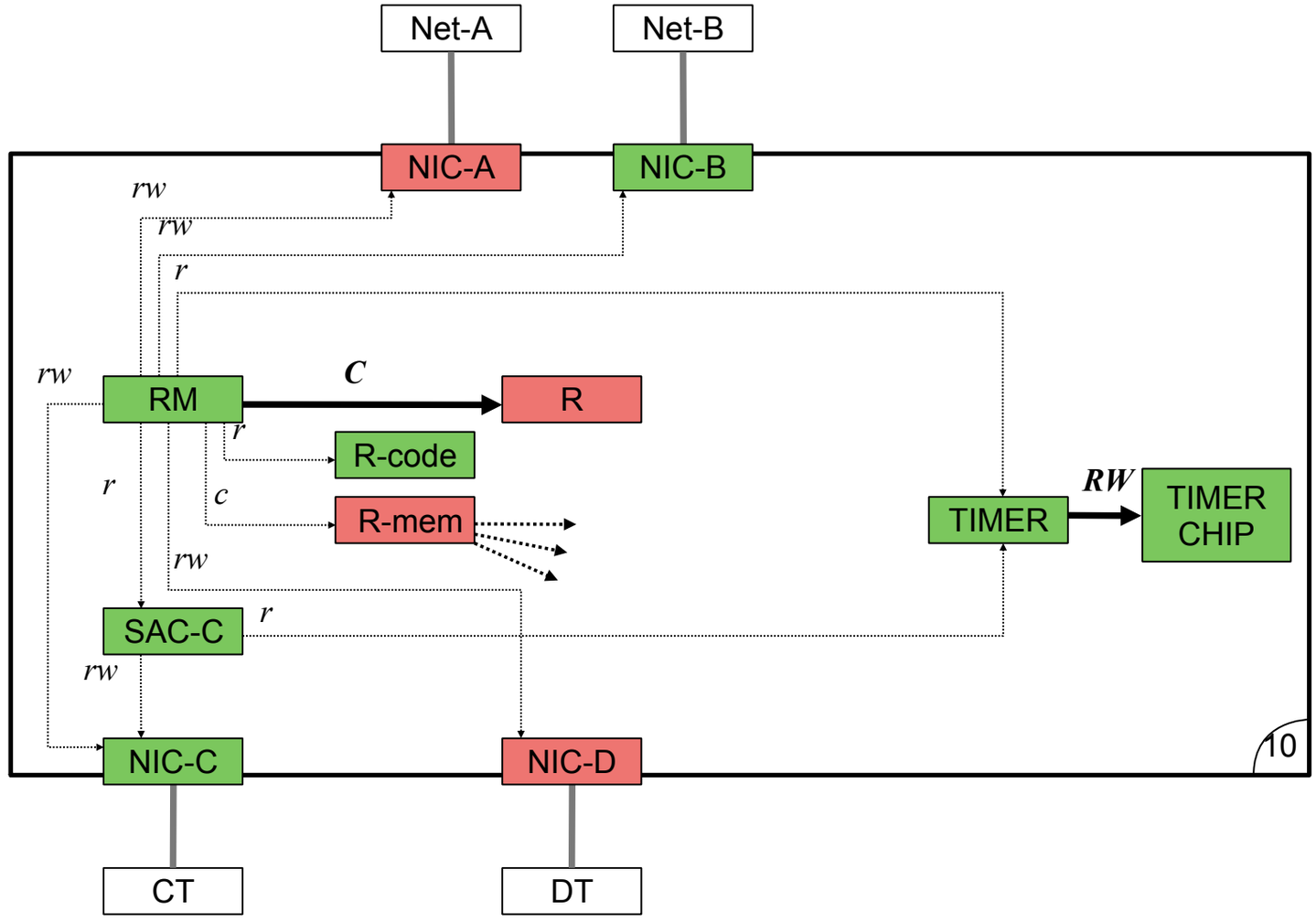
RM receives request to switch to Net-B

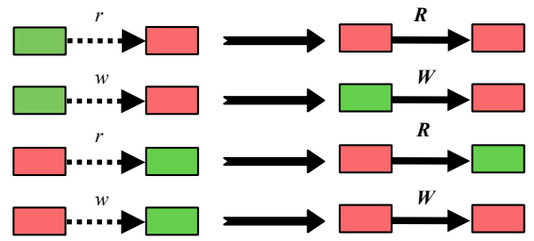


Life Cycle

RM revokes all caps given to R

(using the cap used to create R)

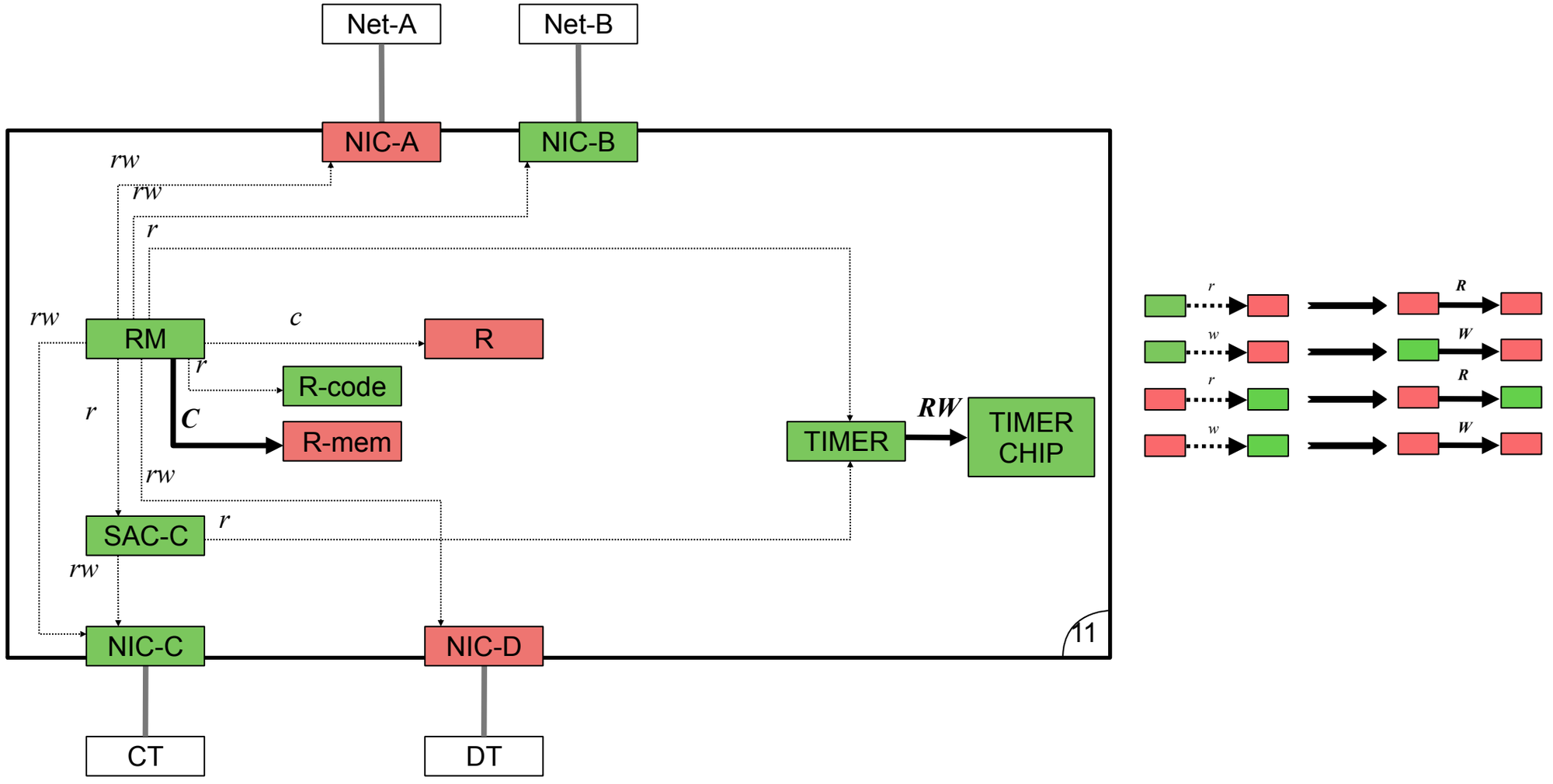




Life Cycle

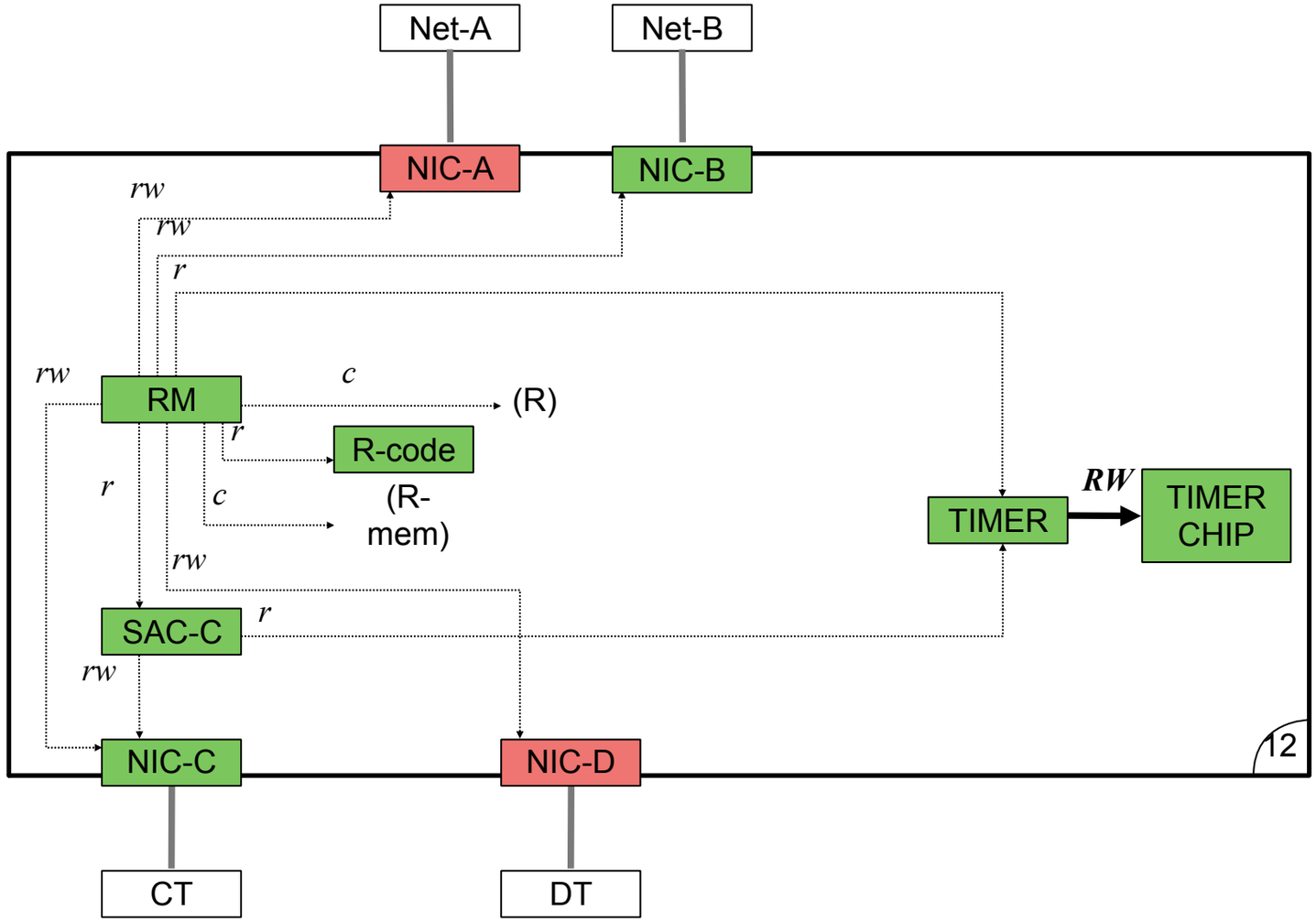
RM revokes caps of R-mem

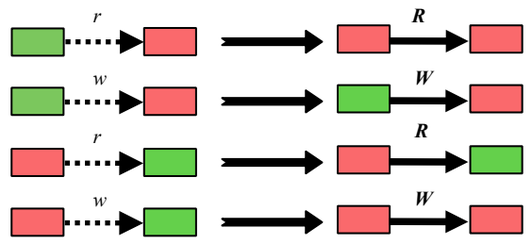
(using create cap to R-mem)



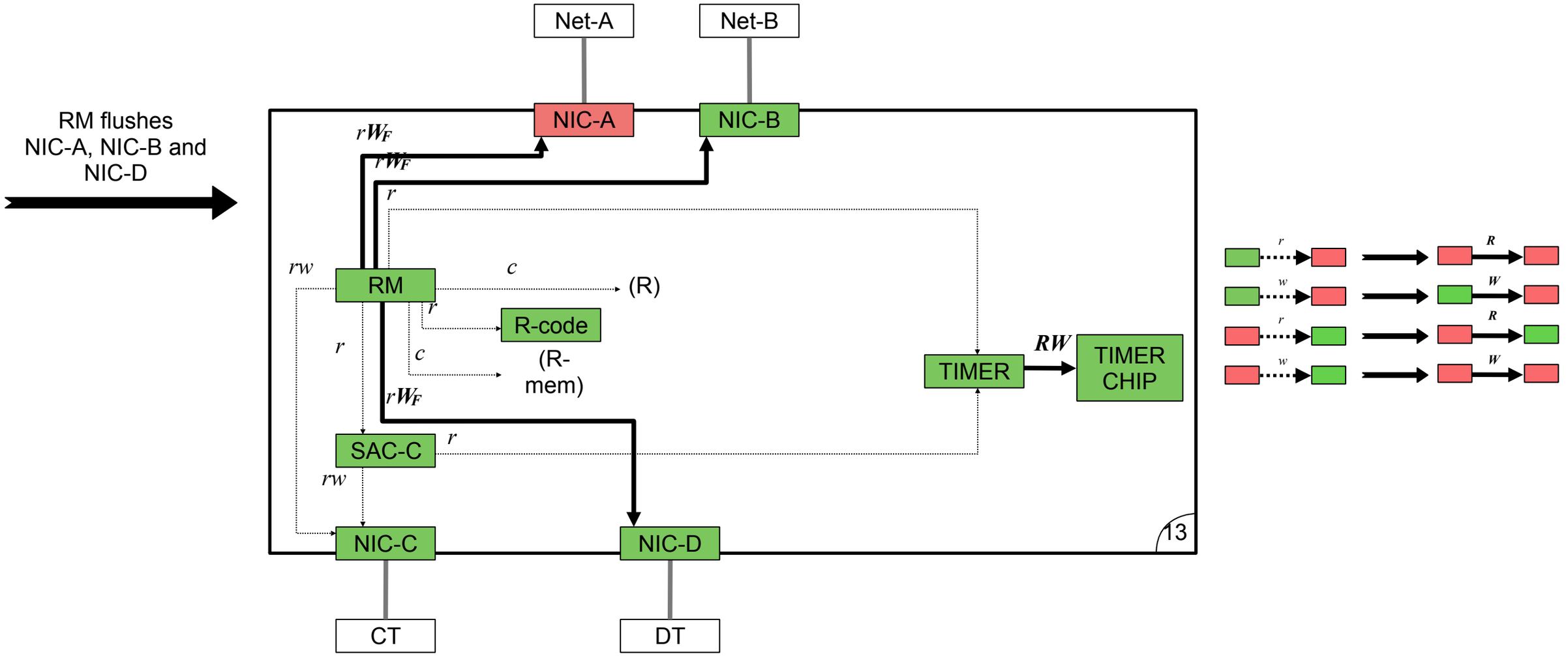
Life Cycle

RM deletes R and R-mem





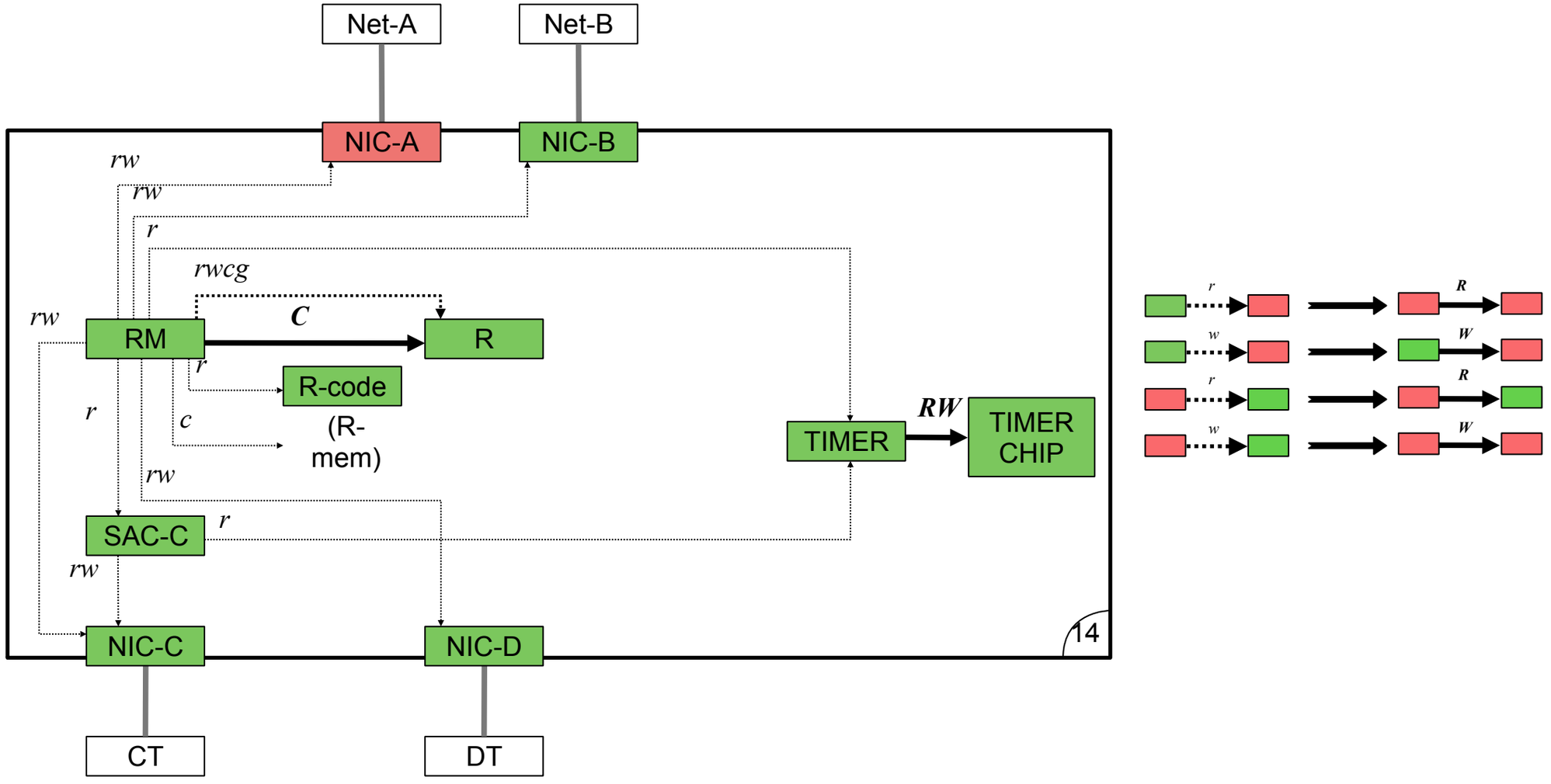
Life Cycle



Life Cycle

RM creates a new Router instance R

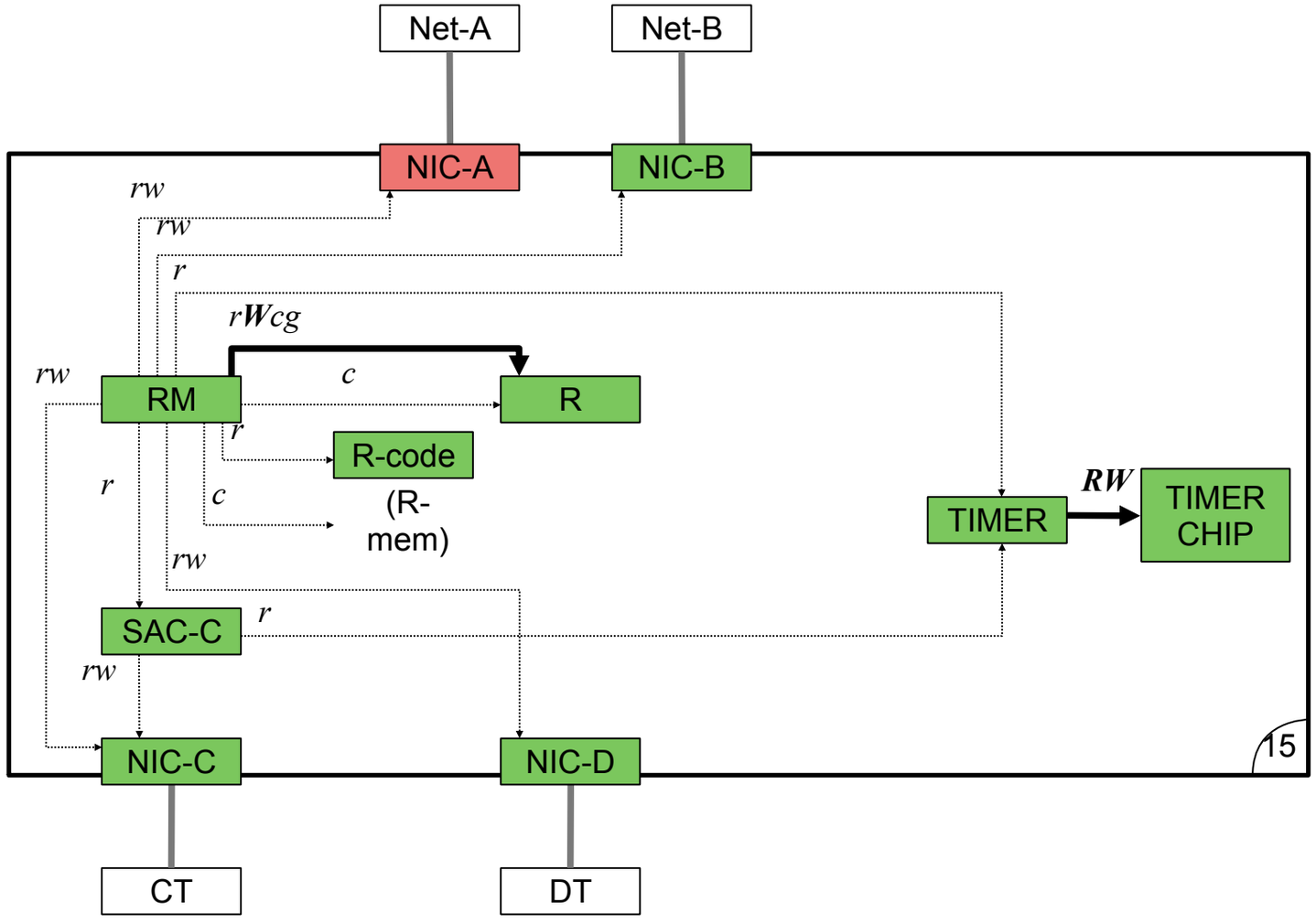
(and gets full rights to the newly created object)

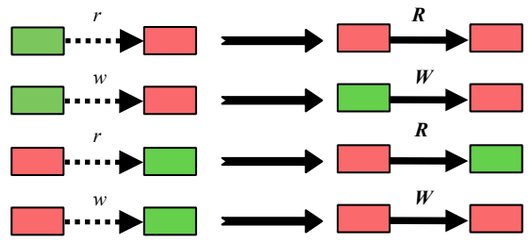


Life Cycle

RM grants to R its rights to NIC-B, NIC-D,

R-mem, R-code, TIMER and itself

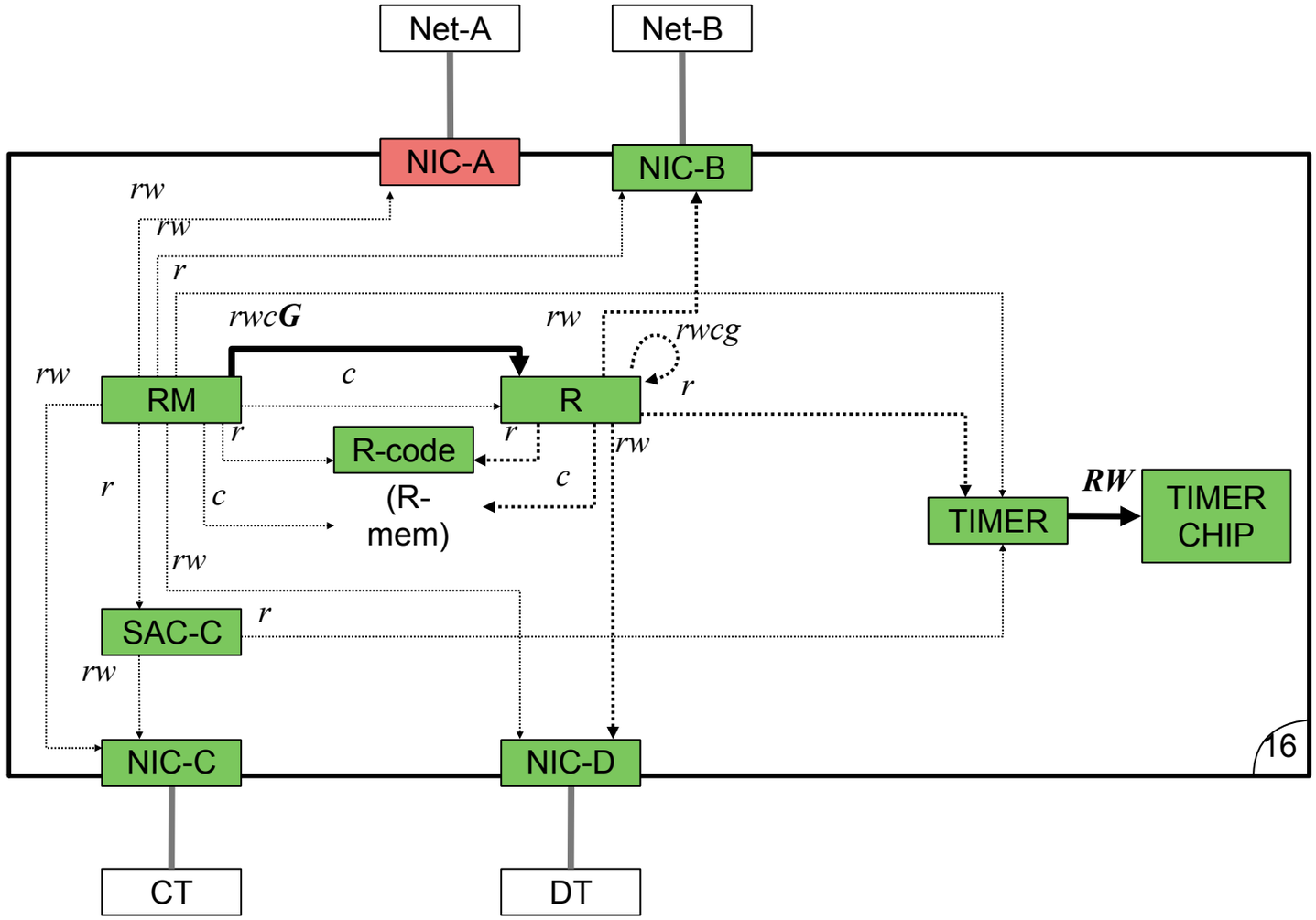


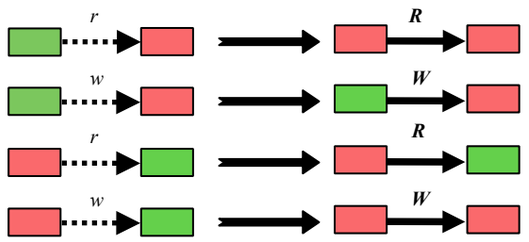


Life Cycle

RM grants to R its rights to NIC-B, NIC-D,

R-mem, R-code, TIMER and itself

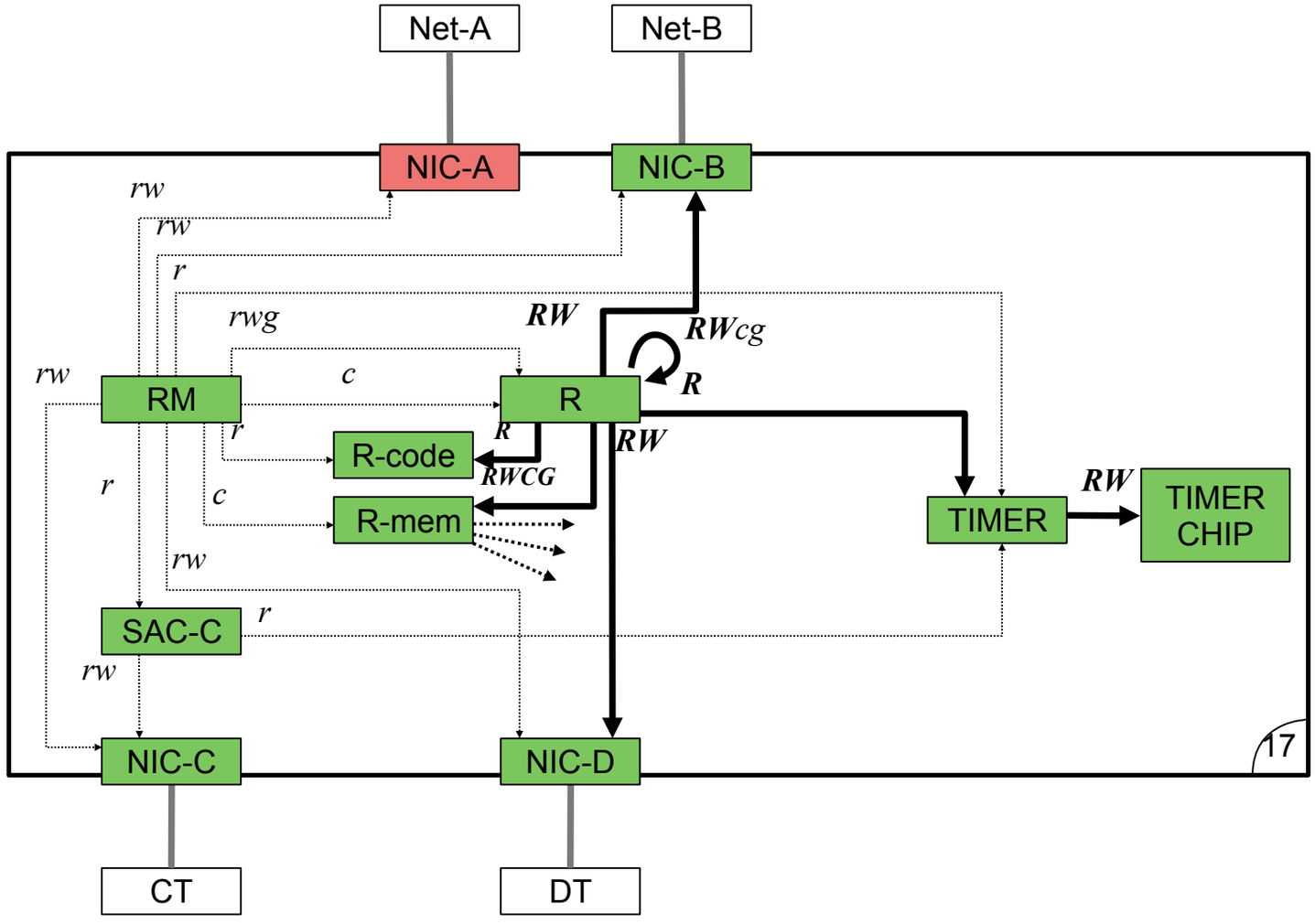


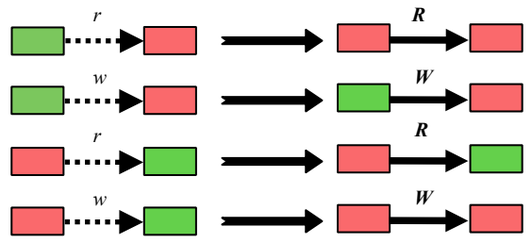


Life Cycle

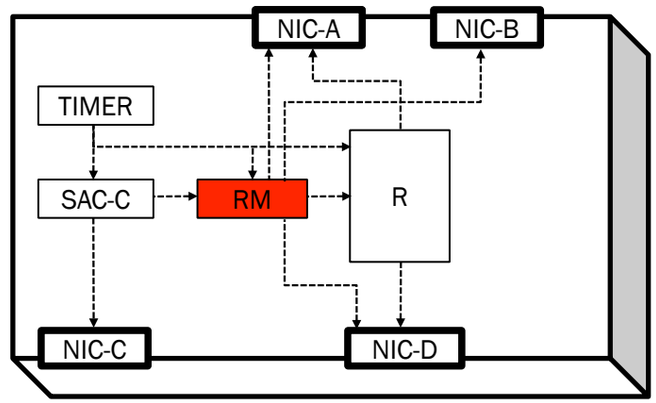
DT starts to communicate with Net-B, through R.

This may imply the creation of new objects using R-mem and granting some caps to them.



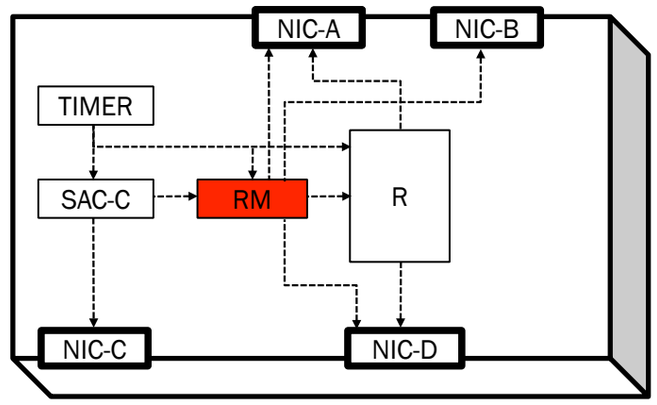


So far



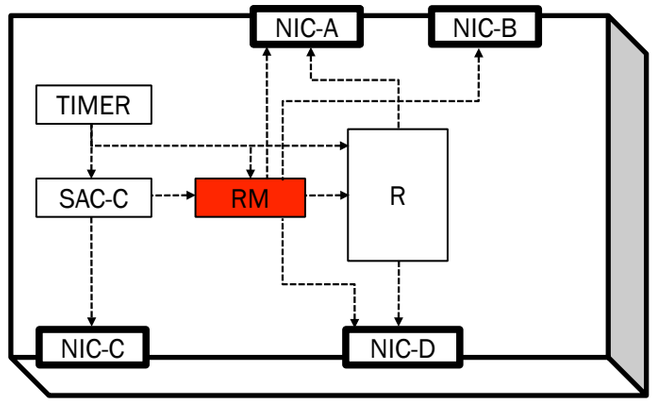
So far

- Can build systems with
  - large untrusted components
  - plus few small, trusted components
  - trusted = needs behaviour spec

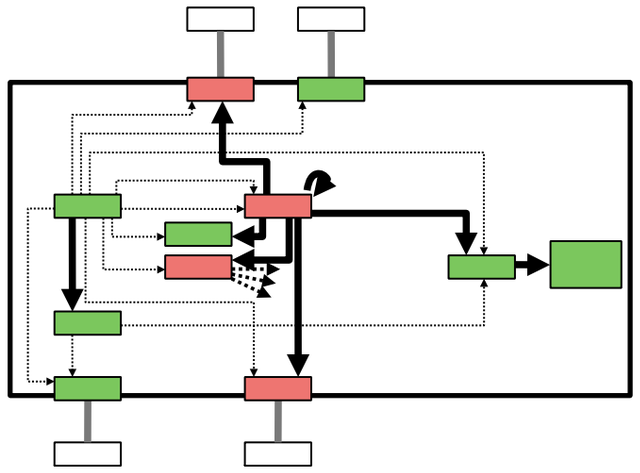


So far

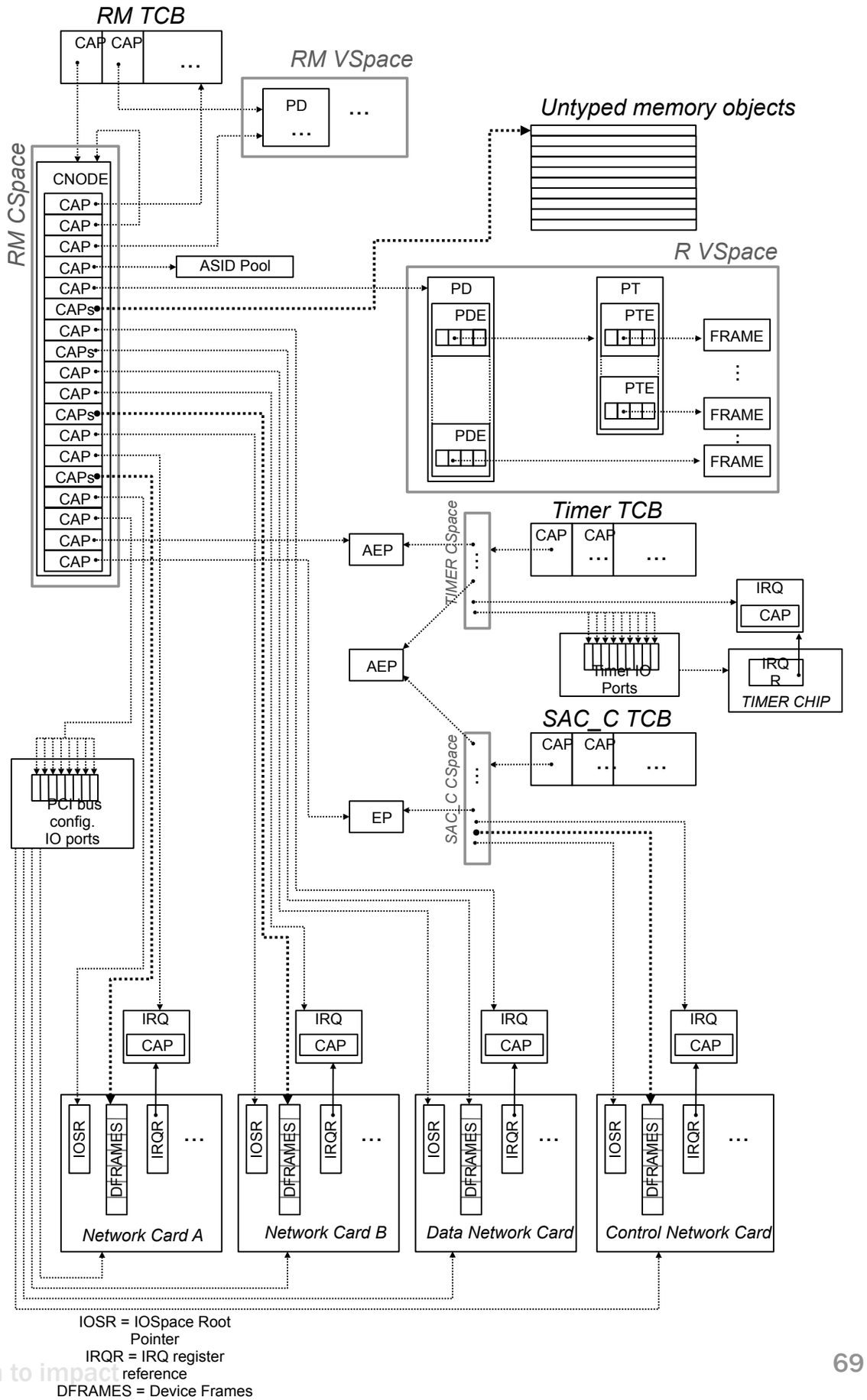
- Can build systems with
  - large untrusted components
  - plus few small, trusted components
  - trusted = needs behaviour spec



- Use take-grant to model security
  - can simulate system
  - modelling already finds bugs
  - high-level proof in Isabelle/HOL or SPIN
  - includes behaviour of trusted component

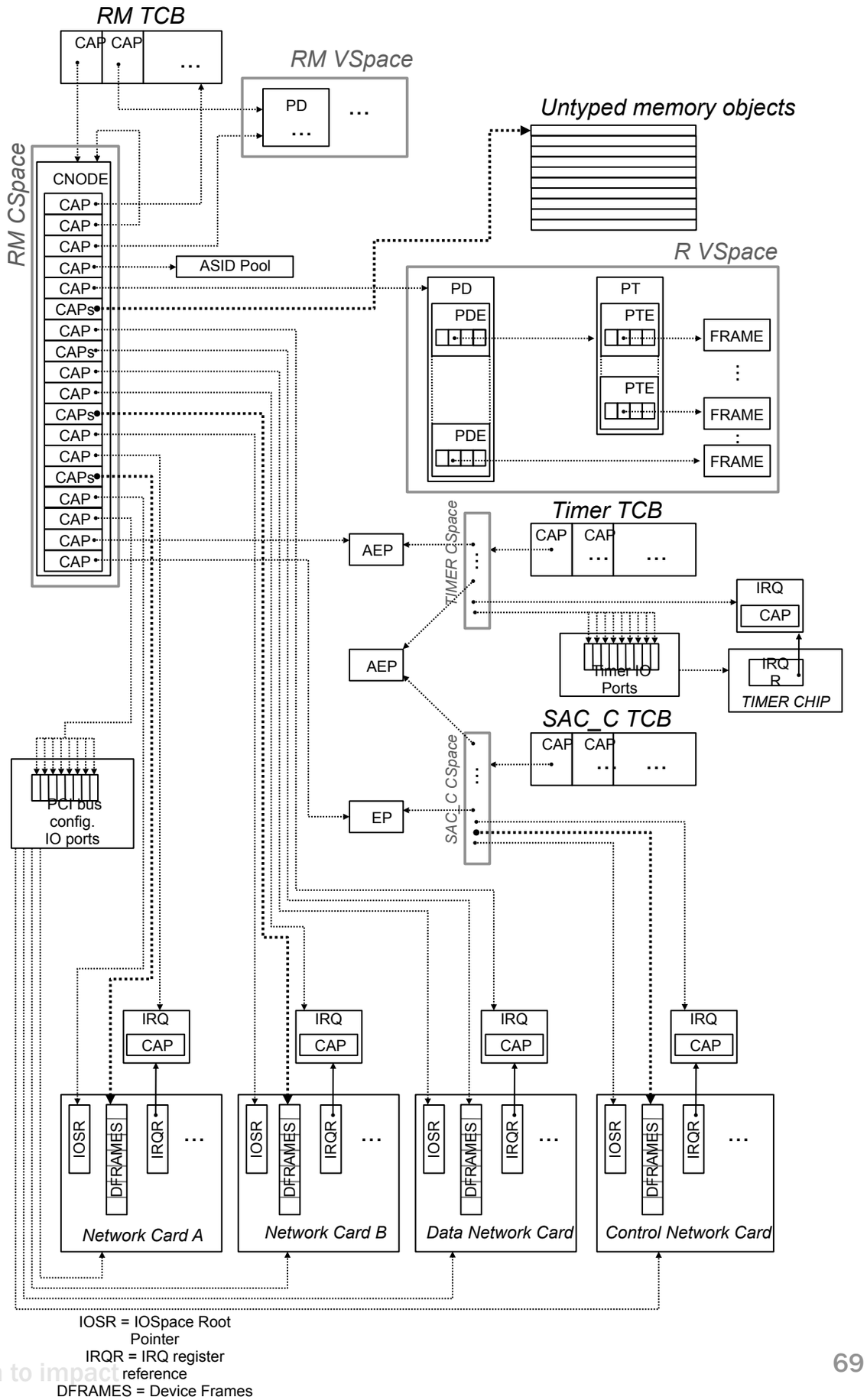


Future



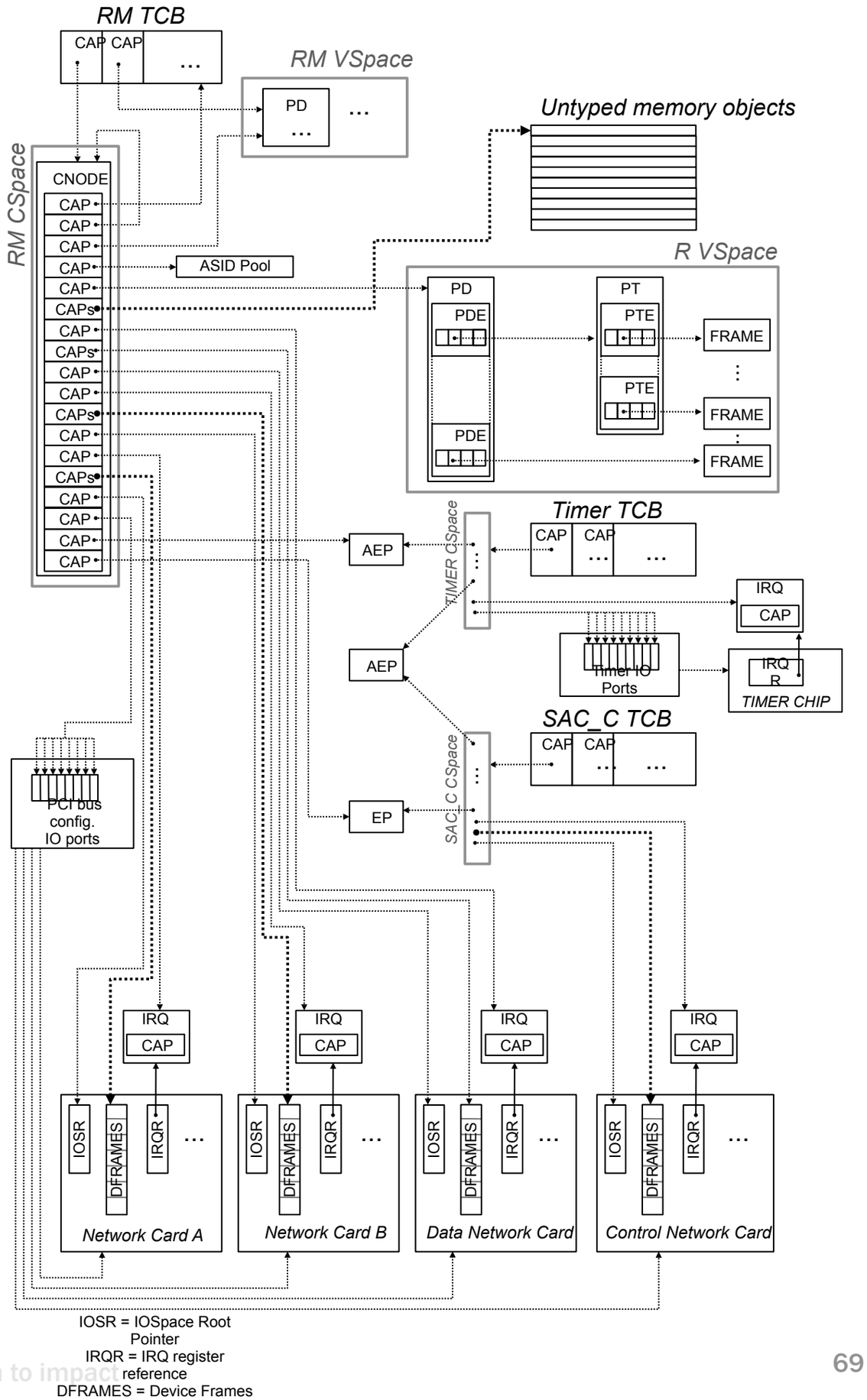
Future

- Need to verify low-level design

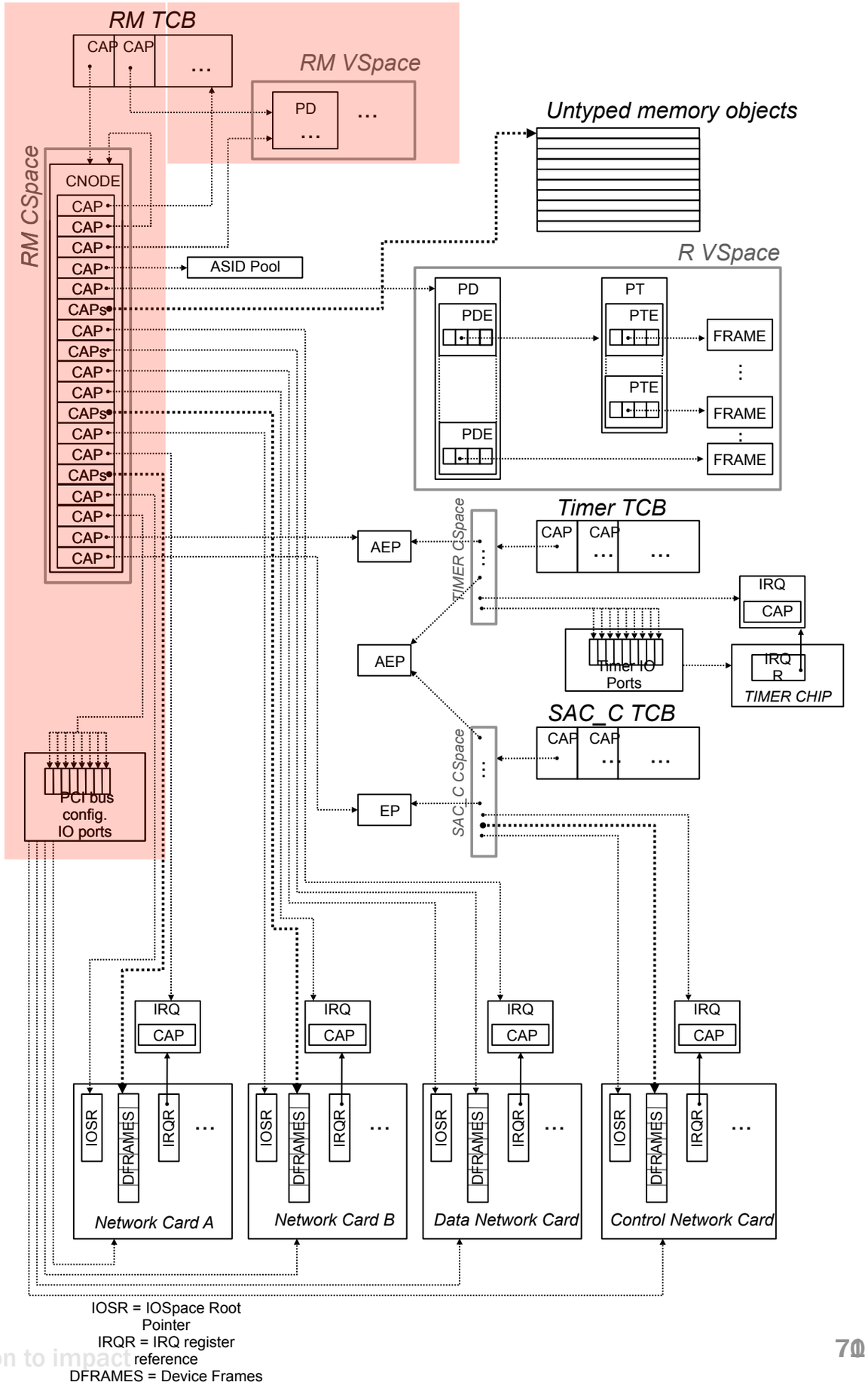


Future

- Need to verify low-level design
- Building tool-chain for:
  - describing cap layout (capDL)
  - generating booter
  - generating booter proof
  - abstraction to take-grant

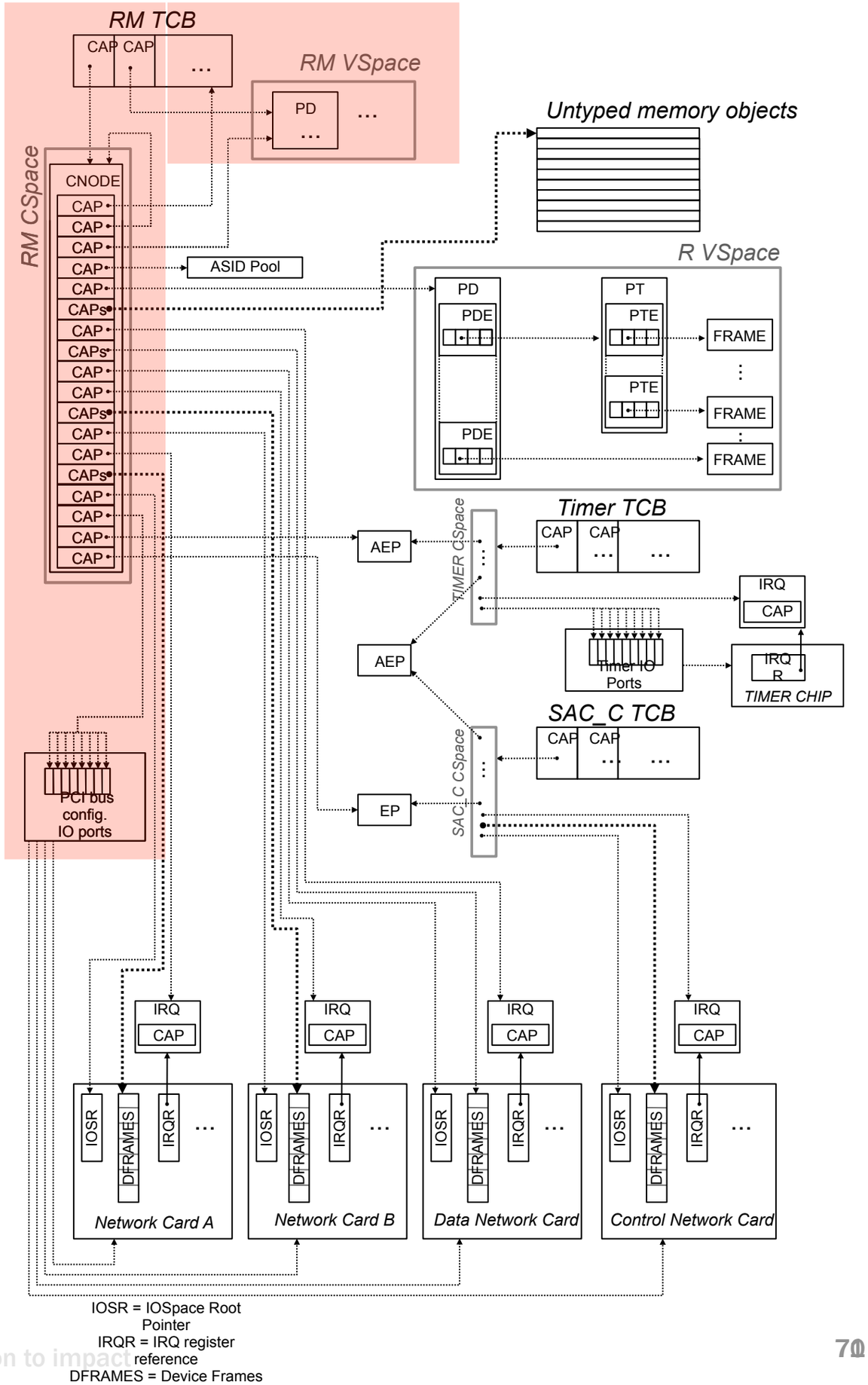


More Future



More Future

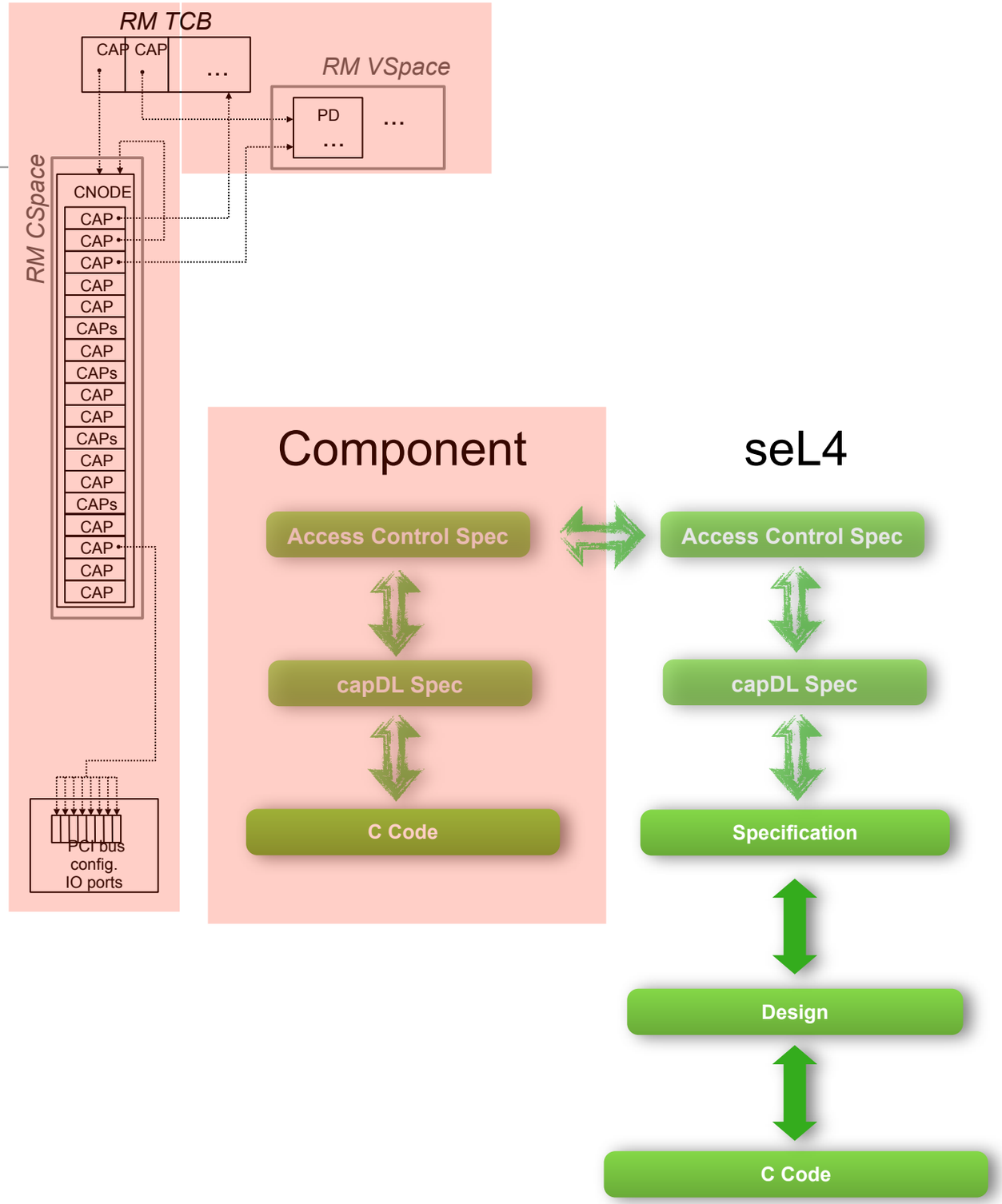
- Verify
  Trusted Component



More Future

- Verify
  Trusted Component

- Refine to C:
  - interface with kernel
  - use most abstract level possible
  - make sure sec property preserved by refinement



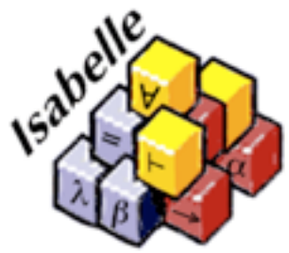
Summary

Summary

Summary

Formal proof all the way from spec to C.

- **200kloc** handwritten, machine-checked proof
- **~460** bugs (160 in C)
- Verification on **code**, **design**, and **spec**
- Systems with **trusted components**
- **The future: formal proof for large systems down to code**



Formal Code Verification up to 10kloc:

It works.
It's feasible.
It's fun.

NICTA

Thank You

NICTA

Thank You