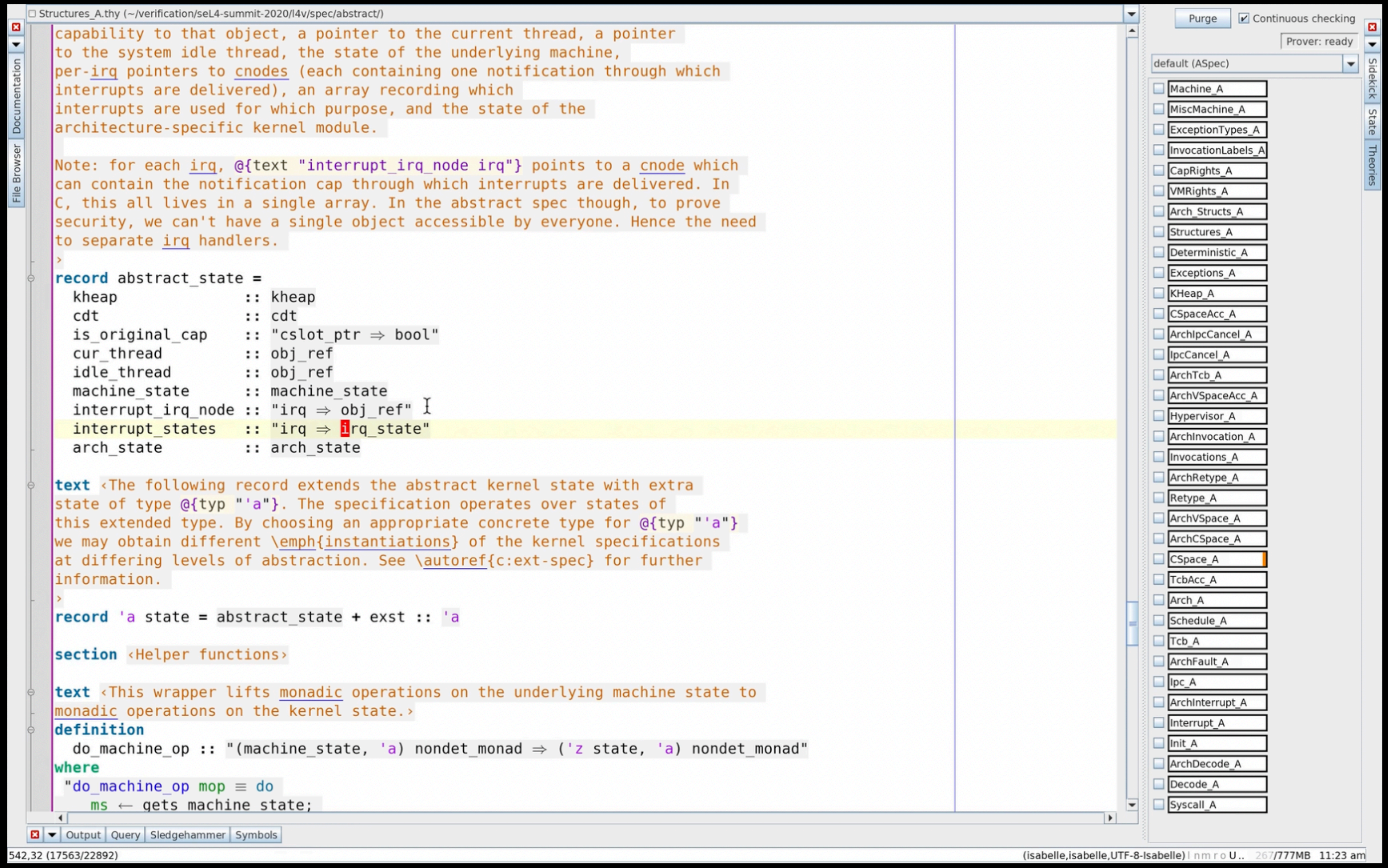
# Explaining the seL4 integrity theorems

Matt Brecknell

Kry10 Limited

seL4 Summit – Munich – October 2022




Previous

Up next

# Introduction to the seL4 proofs

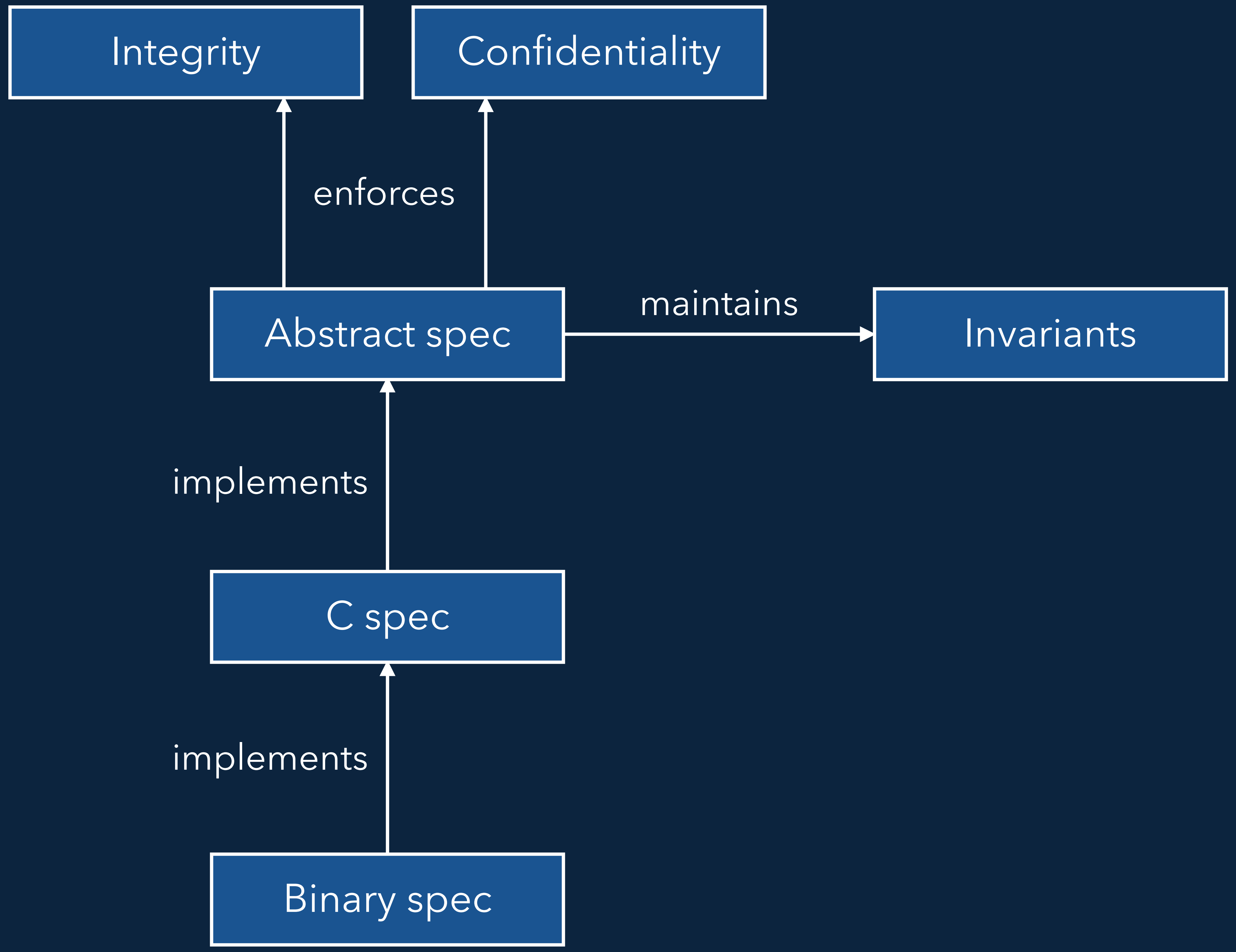
**1 year ago**

This is a guided tour of the proofs about seL4, focussing on the abstract specification and some properties we prove about it. It also has a short introduction to Isabelle/HOL, and the basic formalisms we use to construct the specification and proofs. It was a pre-recorded presentation given at the third seL4 Summit on Nov 16, 2020.

The video is based on this version of the seL4 verification manifest, which roughly corresponds to seL4-12.0.0:
https://github.com/seL4/verification-manifest/blob/c956980aa207bd8c92252ba3e642dfb393e7cd89/default.xml

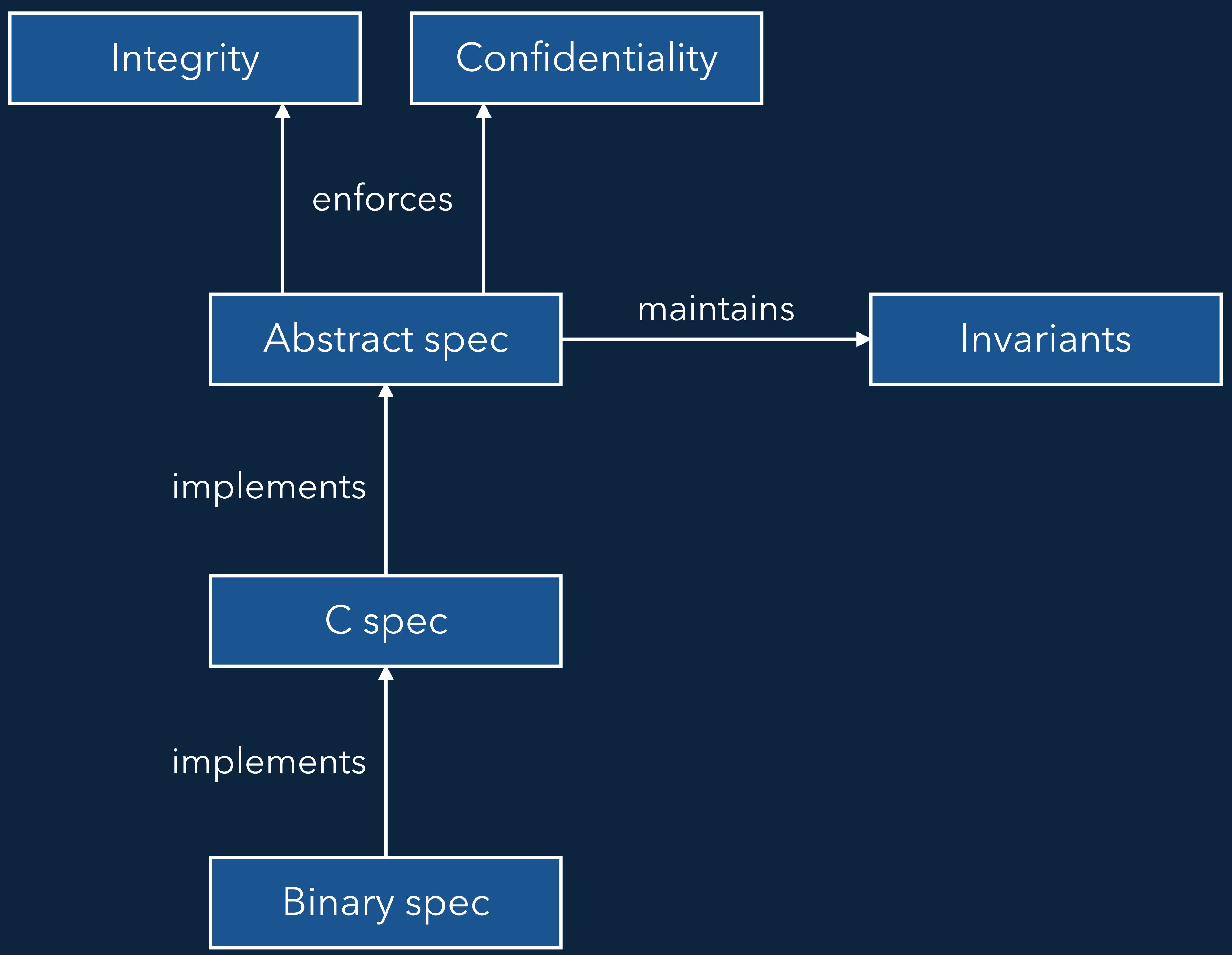
vimeo.com/mbrcknl

# Context




Invariant proofs show that the specification is internally consistent

# Context

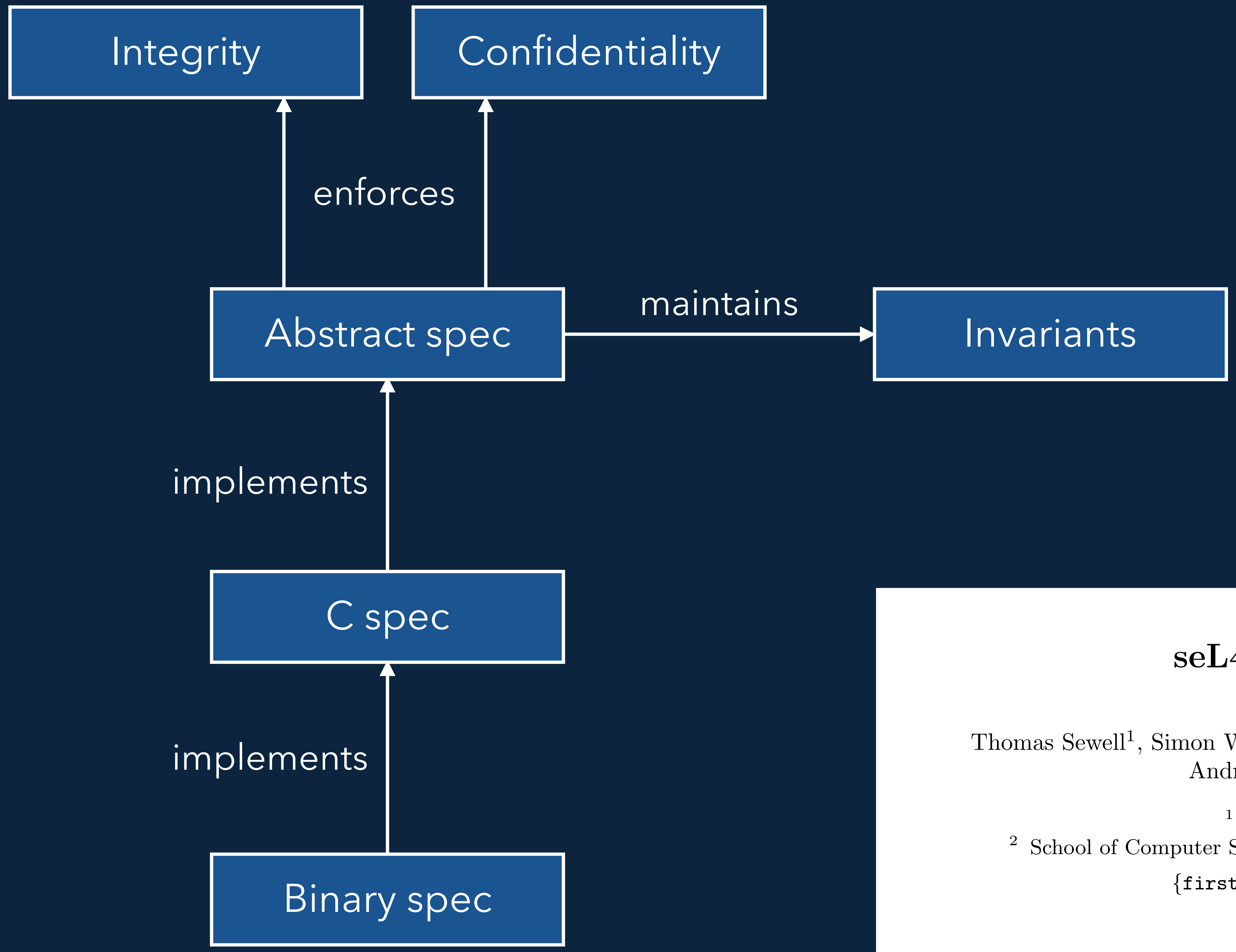



Security proofs show that seL4 enforces access control

- Integrity: for write operations
- Confidentiality: for read operations

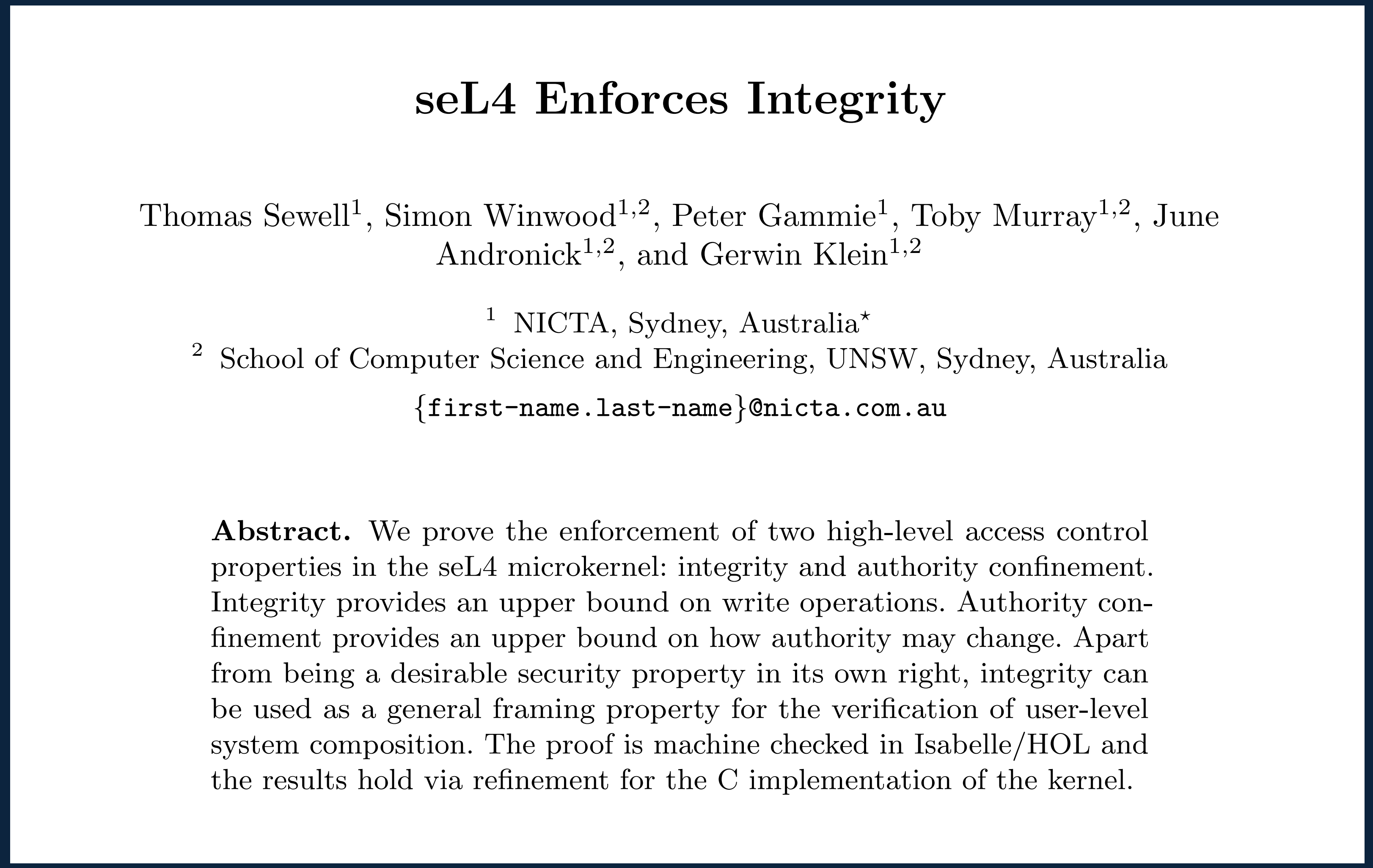
Invariant proofs show that the specification
is internally consistent

# Context




Security proofs show that seL4 enforces access control
- Integrity: for write operations
- Confidentiality: for read operations

Invariant proofs show that the specification is internally consistent



**seL4 Enforces Integrity**

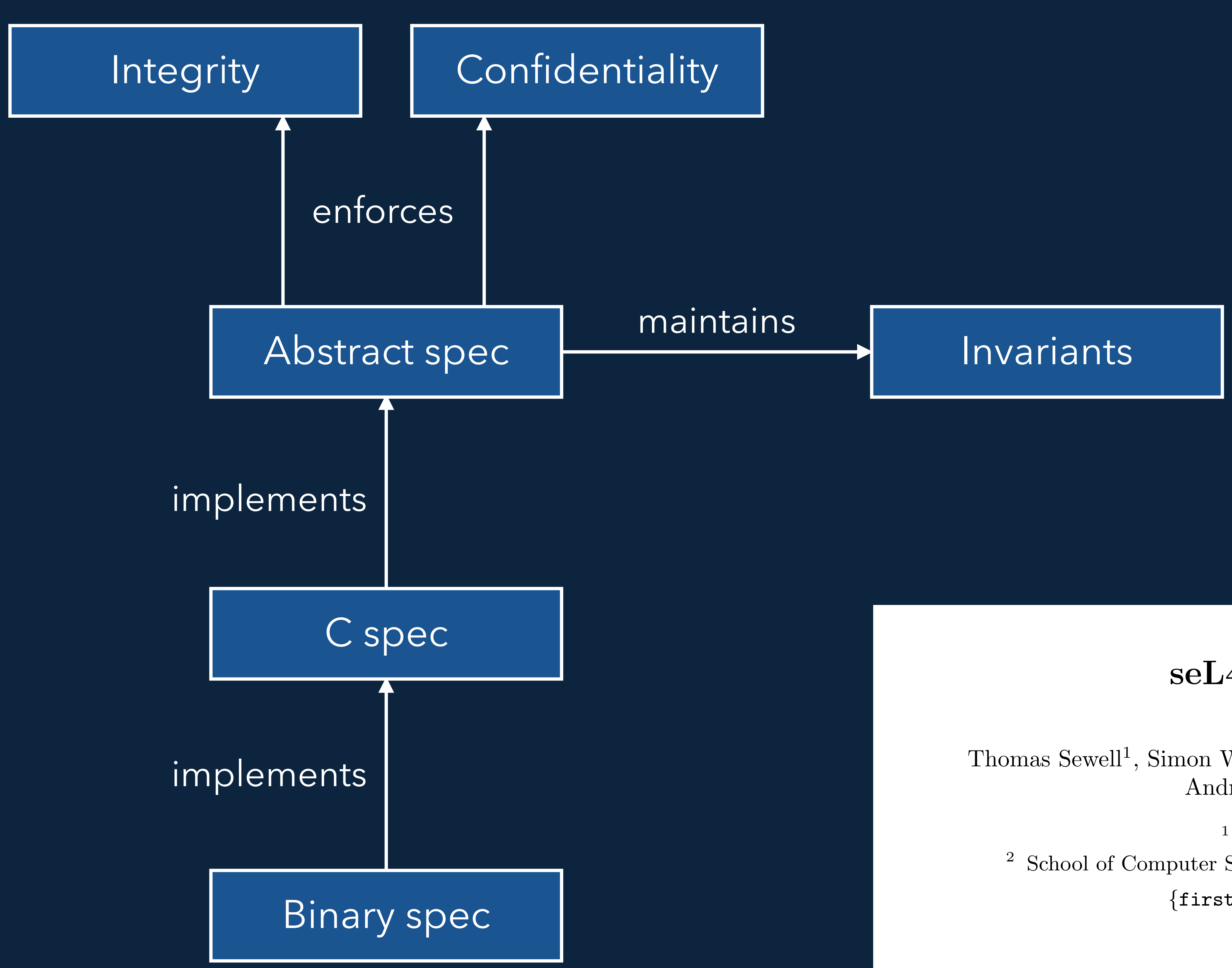
Thomas Sewell[1], Simon Winwood[1,2], Peter Gammie[1], Toby Murray[1,2], June Andronick[1,2], and Gerwin Klein[1,2]

[1] NICTA, Sydney, Australia*
[2] School of Computer Science and Engineering, UNSW, Sydney, Australia
{first-name.last-name}@nicta.com.au

**Abstract.** We prove the enforcement of two high-level access control properties in the seL4 microkernel: integrity and authority confinement. Integrity provides an upper bound on write operations. Authority confinement provides an upper bound on how authority may change. Apart from being a desirable security property in its own right, integrity can be used as a general framing property for the verification of user-level system composition. The proof is machine checked in Isabelle/HOL and the results hold via refinement for the C implementation of the kernel.

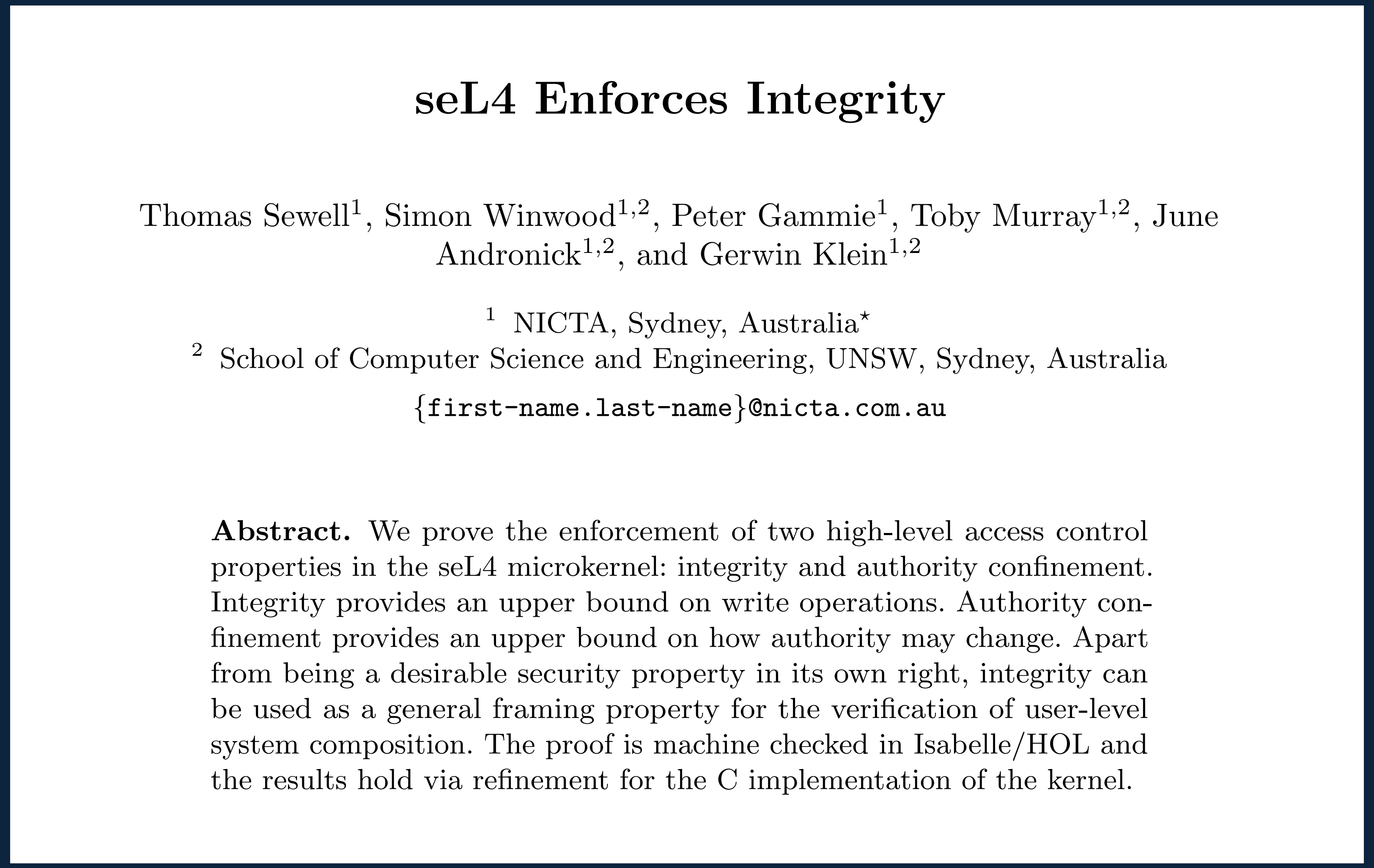
ITP 2011

# Context




Security proofs show that seL4 enforces access control
- Integrity: for write operations
- Confidentiality: for read operations

Invariant proofs show that the specification is internally consistent



**seL4 Enforces Integrity**

Thomas Sewell[1], Simon Winwood[1,2], Peter Gammie[1], Toby Murray[1,2], June Andronick[1,2], and Gerwin Klein[1,2]

[1] NICTA, Sydney, Australia*
[2] School of Computer Science and Engineering, UNSW, Sydney, Australia
{first-name.last-name}@nicta.com.au

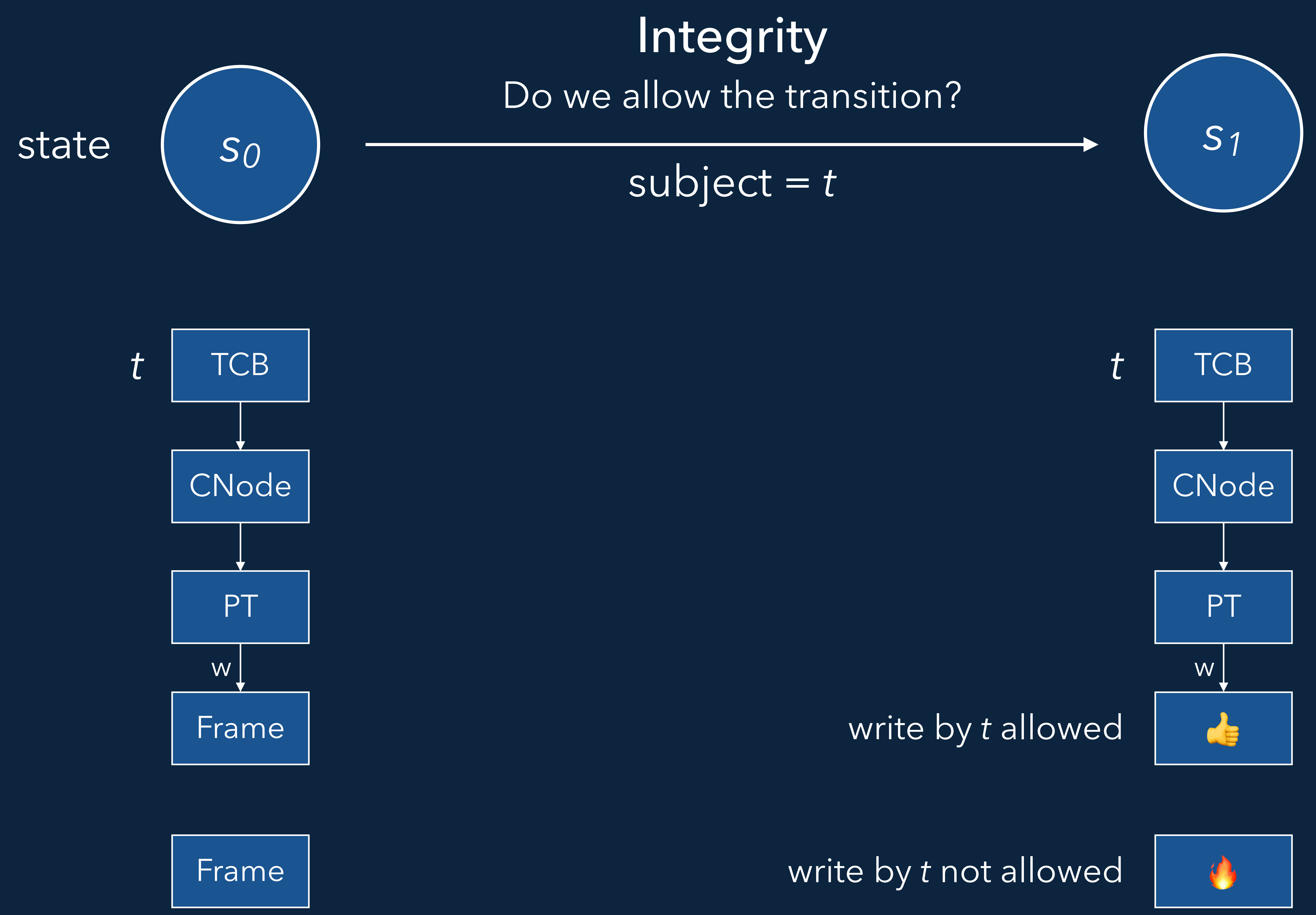
**Abstract.** We prove the enforcement of two high-level access control properties in the seL4 microkernel: integrity and authority confinement. Integrity provides an upper bound on write operations. Authority confinement provides an upper bound on how authority may change. Apart from being a desirable security property in its own right, integrity can be used as a general framing property for the verification of user-level system composition. The proof is machine checked in Isabelle/HOL and the results hold via refinement for the C implementation of the kernel.

ITP 2011

Does integrity apply to dynamic systems?

# Concepts

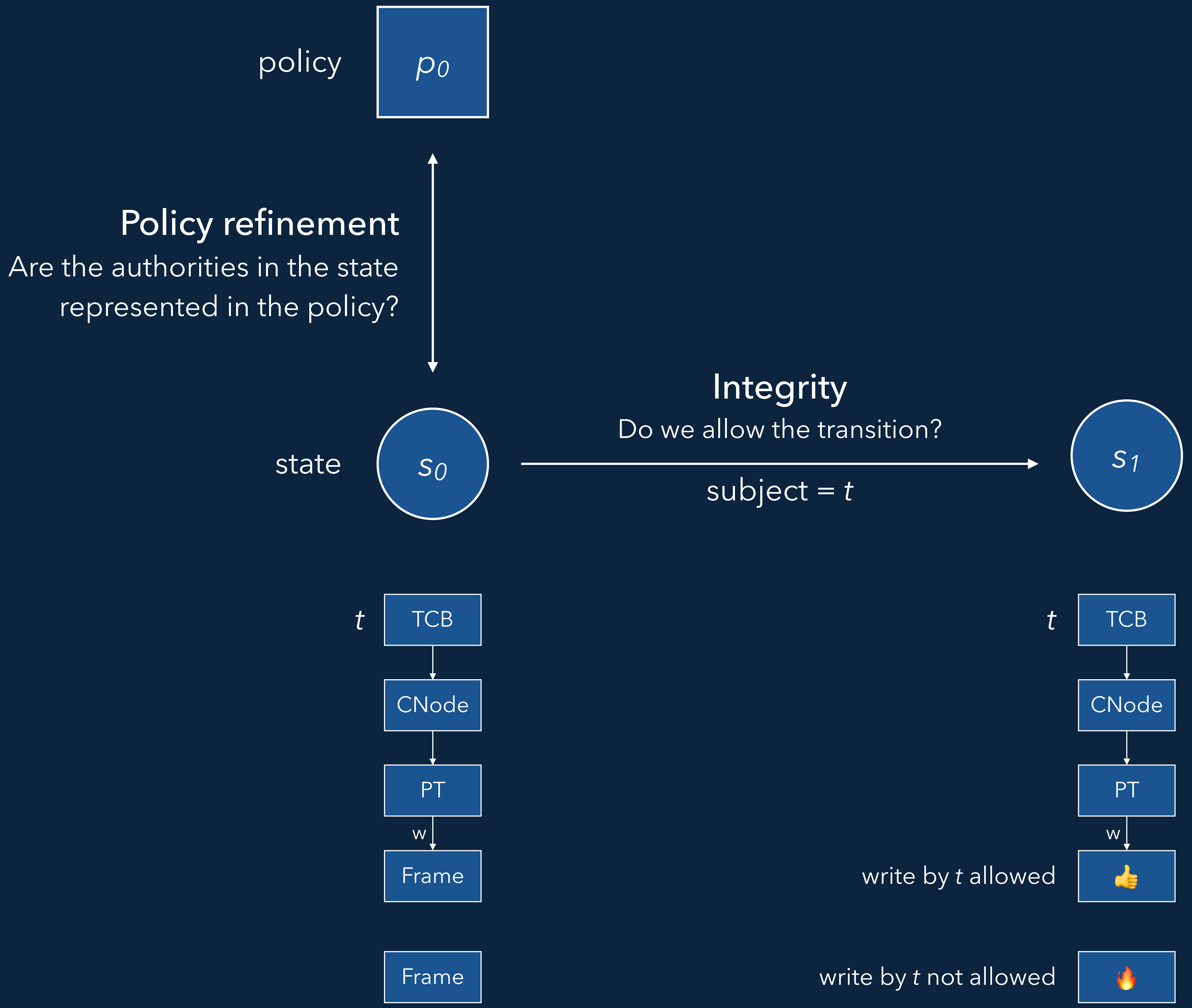
# Concepts




Look at an individual state to determine the authority held by the subject.

Look at a pair of states to determine whether a change may be allowed for the subject, given its authority.

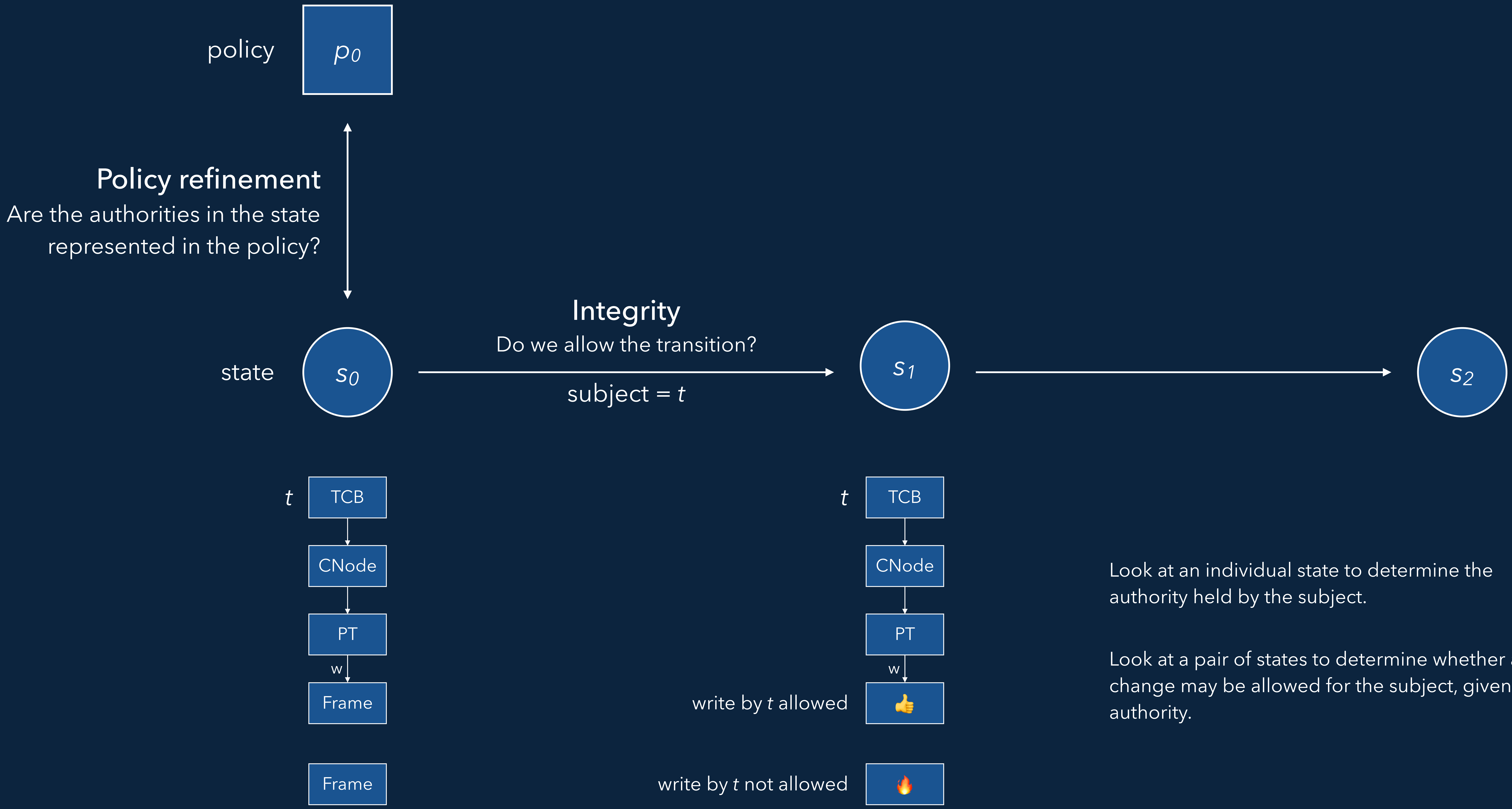
# Concepts




Look at an individual state to determine the authority held by the subject.

Look at a pair of states to determine whether a change may be allowed for the subject, given its authority.

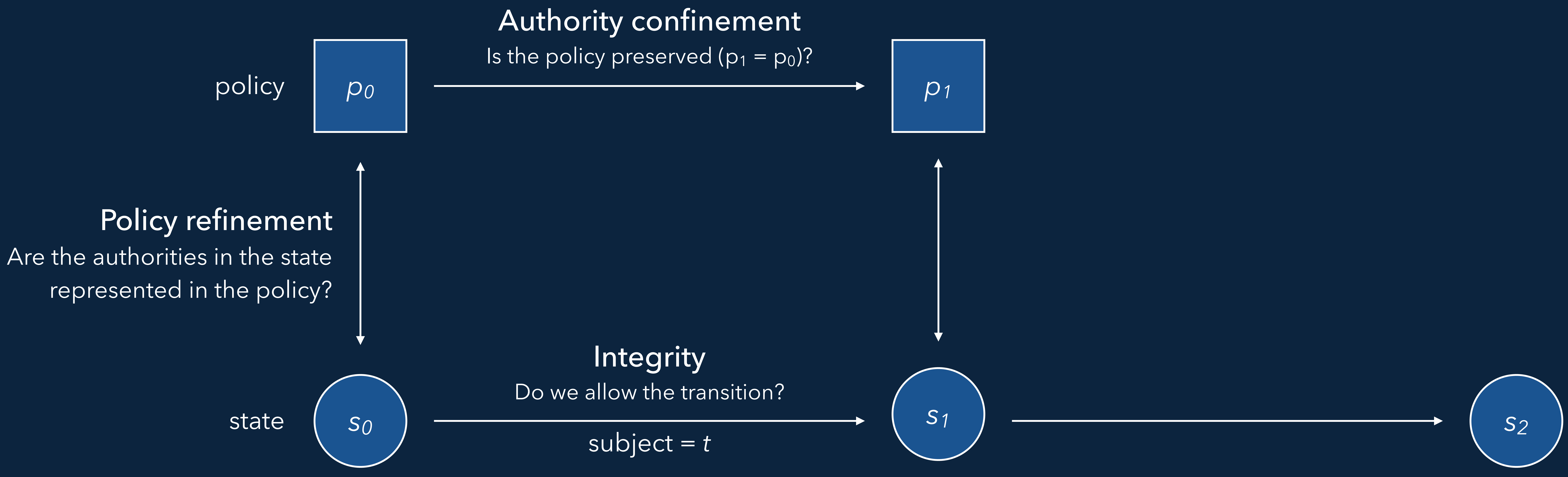
# Concepts

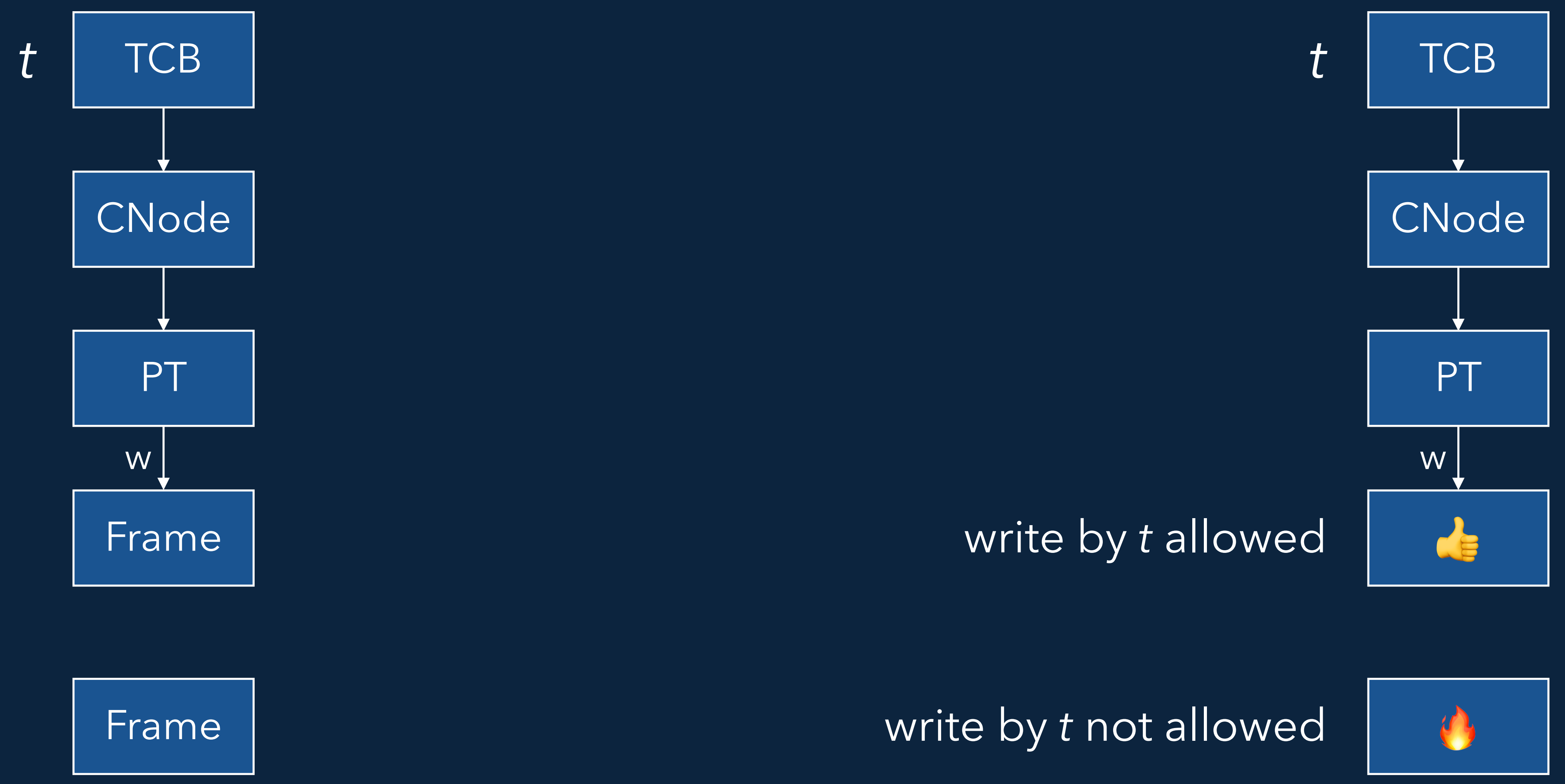



Look at an individual state to determine the authority held by the subject.

Look at a pair of states to determine whether a change may be allowed for the subject, given its authority.

# Concepts




Look at an individual state to determine the authority held by the subject.

Look at a pair of states to determine whether a change may be allowed for the subject, given its authority.

# The process of theorem proving

1. State definitions
   - Definitions give names to expressions, functions, predicates, relations

2. Prove theorems
   - Theorems are also logical expressions with names
   - But they require proofs

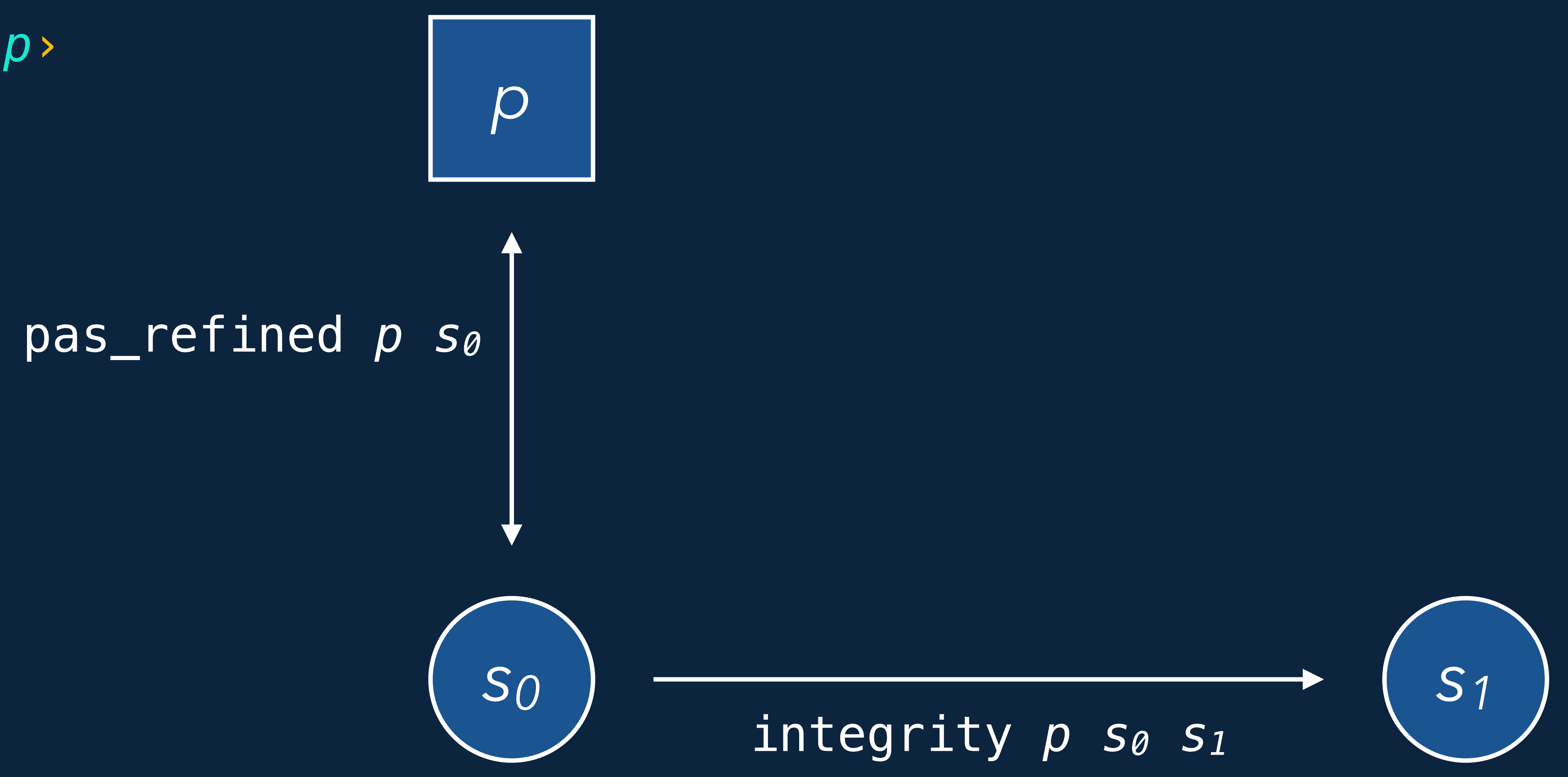
# The process of theorem proving

1. State definitions

- Definitions give names to expressions, functions, predicates, relations

```
— ‹True iff all authorities in state s are represented in policy p›
definition pas_refined p s ≡ …

— ‹True iff the change between states s0 and s1
   is authorised for the current subject by policy p›
definition integrity p s0 s1 ≡ …
```




2. Prove theorems

- Theorems are also logical expressions with names
- But they require proofs

# The process of theorem proving

1. State definitions

- Definitions give names to expressions, functions, predicates, relations

```
— ‹True iff all authorities in state s are represented in policy p›
definition pas_refined p s ≡ …

— ‹True iff the change between states s0 and s1
   is authorised for the current subject by policy p›
definition integrity p s0 s1 ≡ …
```


2. Prove theorems

- Theorems are also logical expressions with names
- But they require proofs

```
theorem kernel_integrity:
  — ‹If the subject calls the kernel in a state s0 where pas_refined p s0 is True,
     then the kernel exits in a state s1 where integrity p s0 s1 is True›

theorem auth_confinement:
  — ‹If the subject calls the kernel in a state s0 where pas_refined p s0 is True,
     then the kernel exits in a state s1 where pas_refined p s1 is True›
```

# Summary

## How to show integrity

1. Define an access control policy
   a. Identify components, i.e. label system resources
   b. Define an authority graph, i.e. arrows between components

2. Show policy refinement for the current state
   a. Show that protection state maps onto the authority graph
   b. Show well-formedness for the subject

3. The theorems establish that
   a. State changes initiated by the subject are bounded by the policy
   b. The policy is maintained for the subject

4. For static systems
   - Use a tool to check well-formedness, and a trustworthy loader

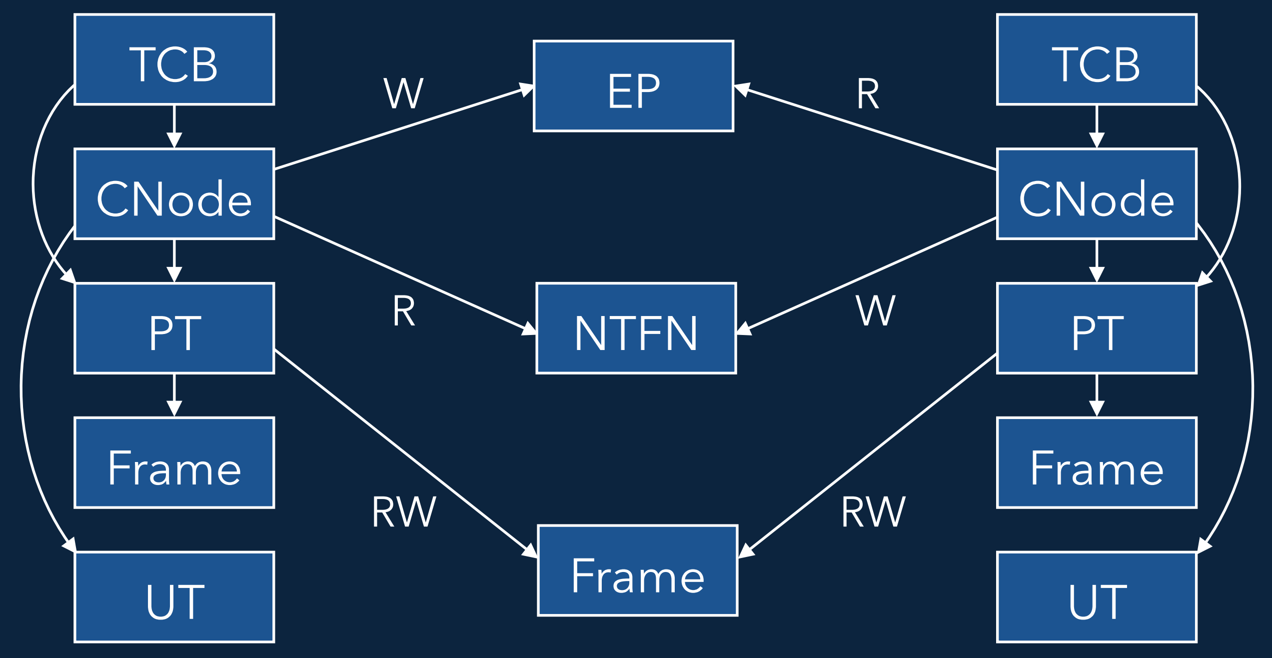
5. For dynamic systems
   - Prove that trusted components establish well-formed policies for their subordinates

# 1. Define an access control policy

## a. Define components

- Draw labelled boxes around resources
  - Usually, groups threads with all their private resources
  - Separate shared resources from their owners

```
pasObjectAbs :: obj_ref ⇒ 'label
```

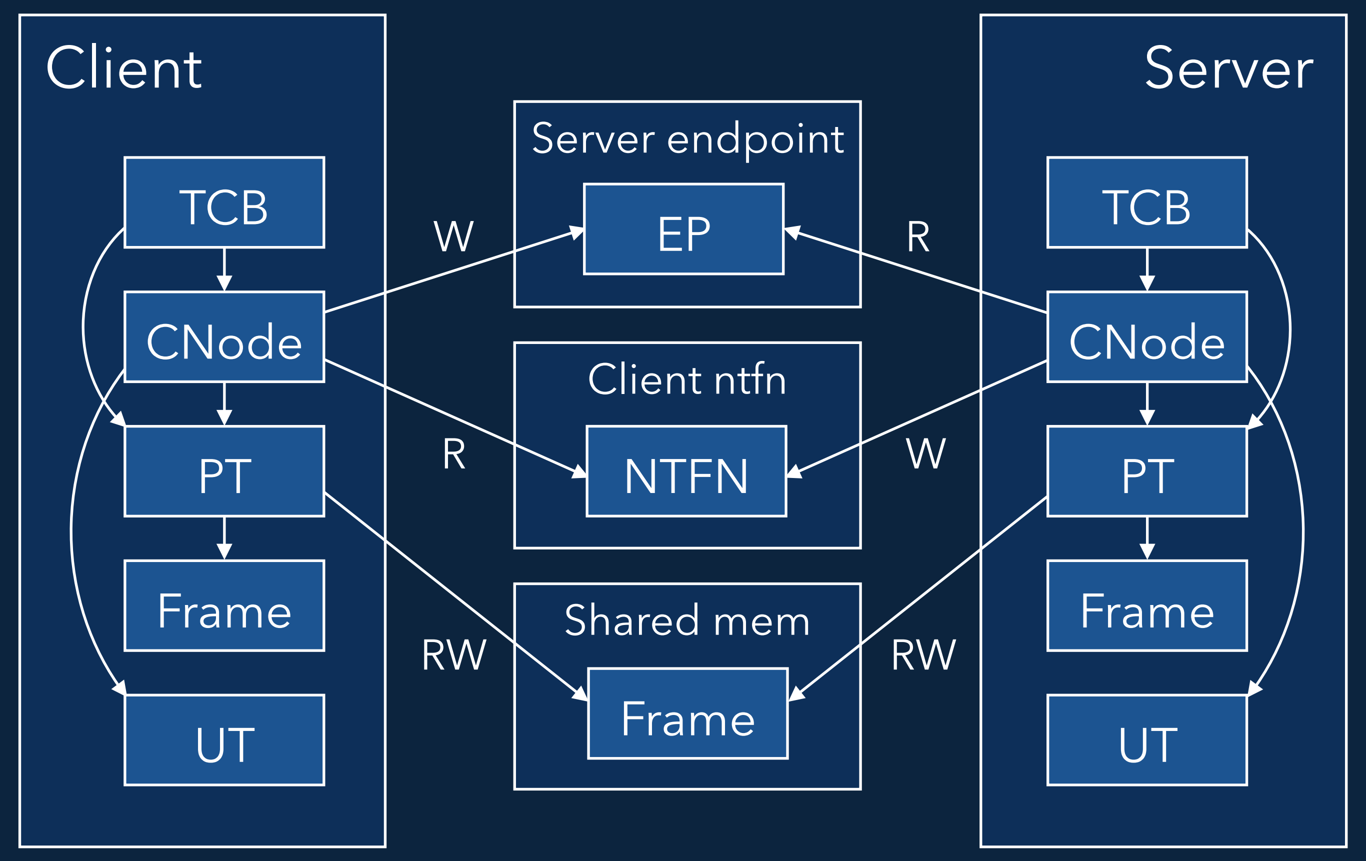


# 1. Define an access control policy

## a. Define components

- Draw labelled boxes around resources
  - Usually, groups threads with all their private resources
  - Separate shared resources from their owners

```
pasObjectAbs :: obj_ref ⇒ 'label
```

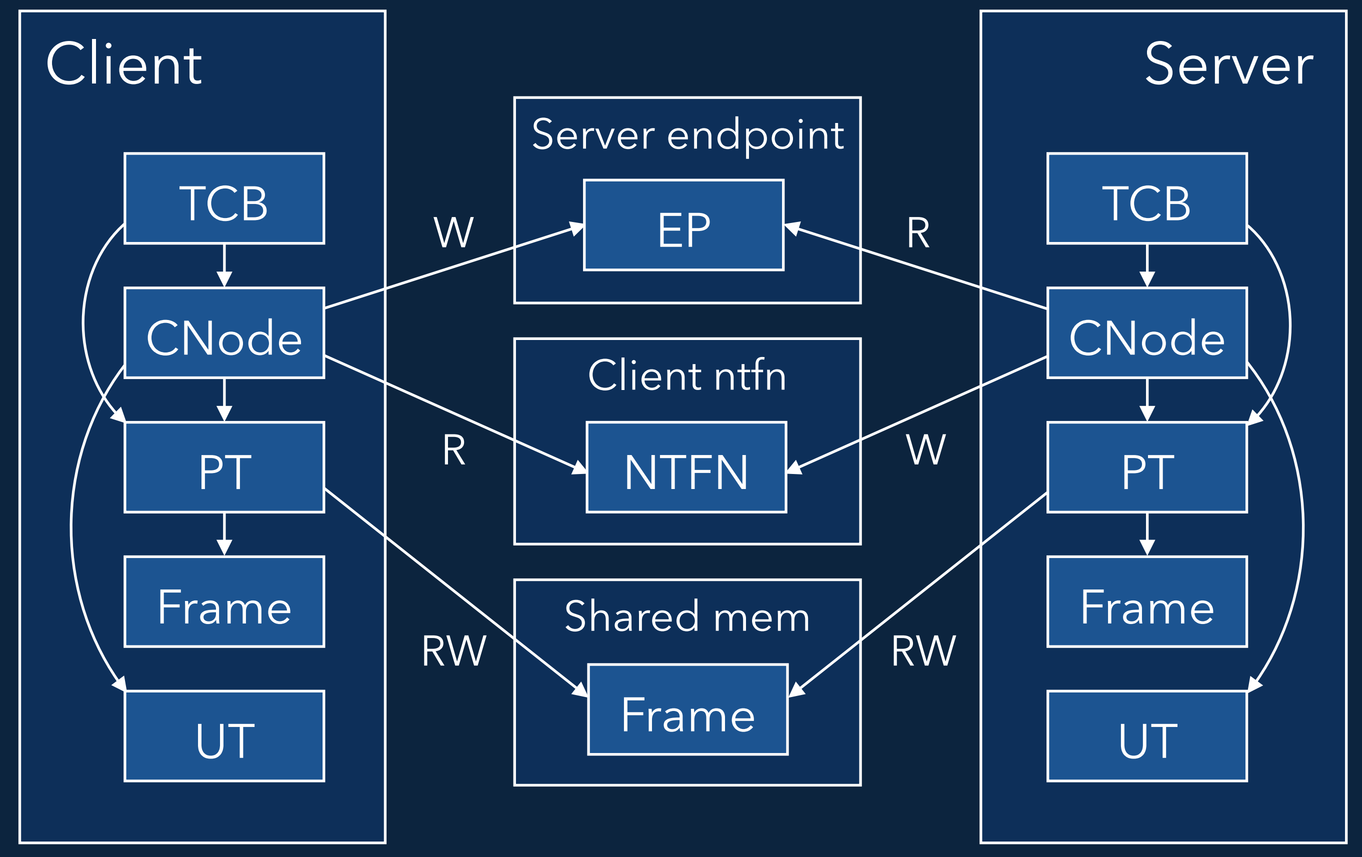


# 1. Define an access control policy

## a. Define components

- Draw labelled boxes around resources
  - Usually, groups threads with all their private resources
  - Separate shared resources from their owners

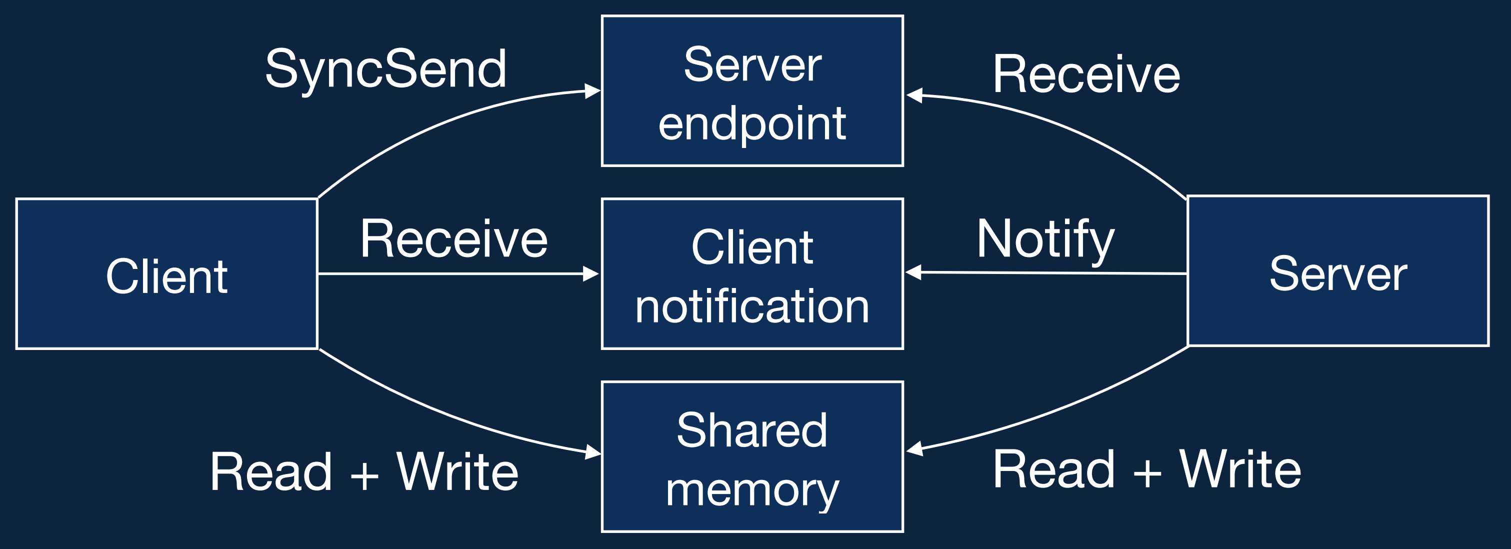
```
pasObjectAbs :: obj_ref ⇒ 'label
```




## b. Define an authority graph

- Arrows between components, labelled with authority types

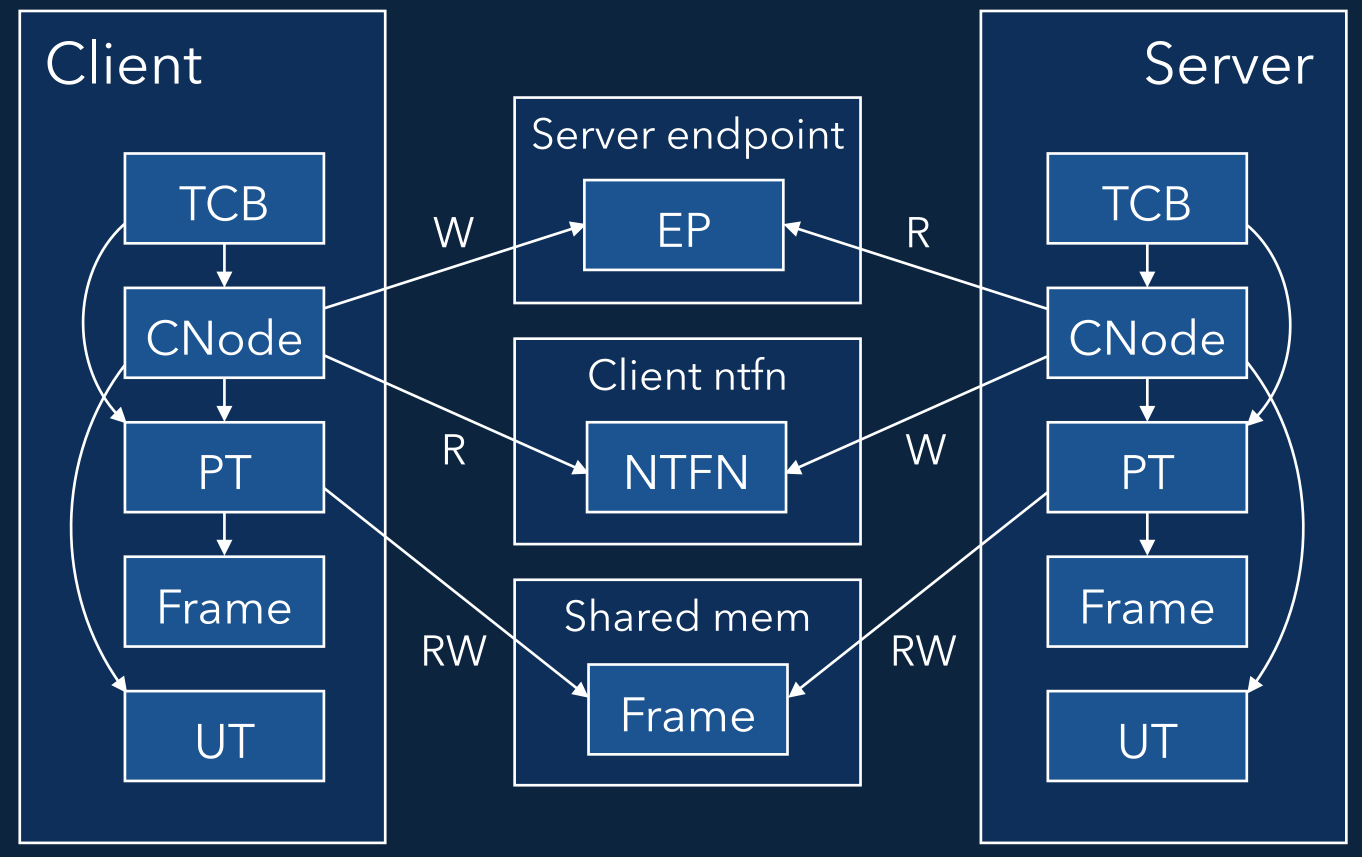
```
pasPolicy :: ('label × auth × 'label) set
```



# 1. Define an access control policy

## a. Define components

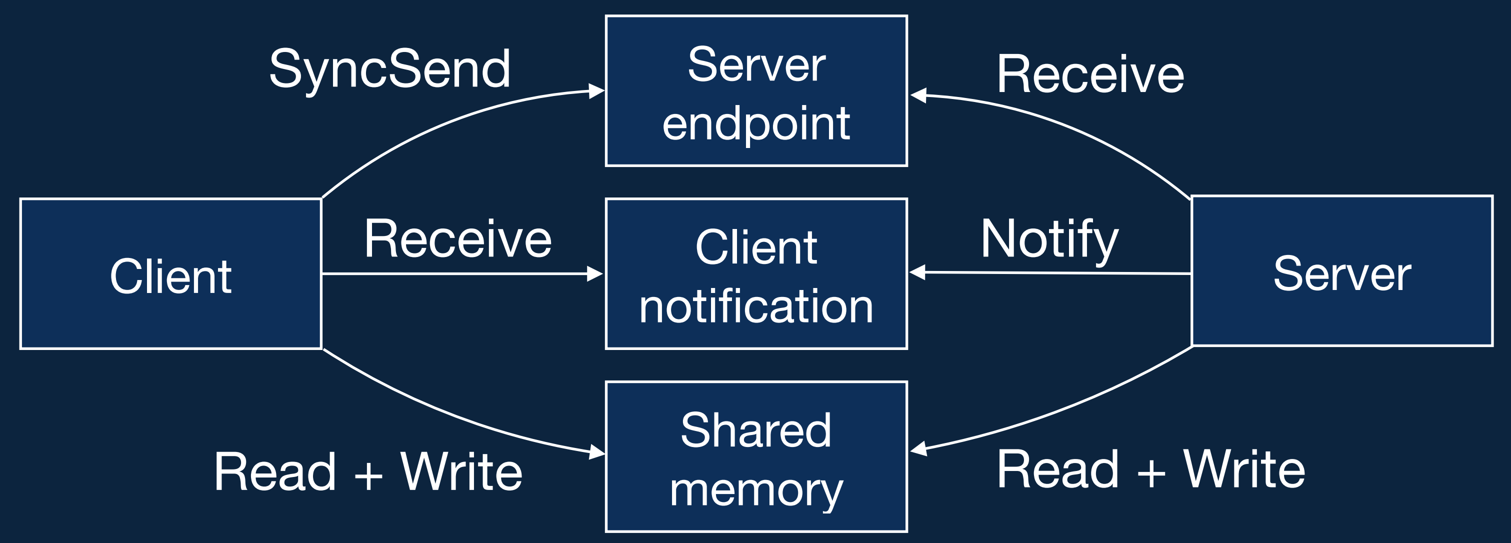
- Draw labelled boxes around resources
  - Usually, groups threads with all their private resources
  - Separate shared resources from their owners




```
pasObjectAbs :: obj_ref ⇒ 'label
```

## b. Define an authority graph

- Arrows between components, labelled with authority types




```
pasPolicy :: ('label × auth × 'label) set
```

```
datatype auth = SyncSend
              | Notify
              | Receive
              | Grant
              | Reset
              | Call
              | Reply
              | DeleteDerived
              | Read
              | Write
              | Control
```

endpoints and notifications (SyncSend, Notify, Receive, Grant, Reset)

protected procedure calls (Call, Reply, DeleteDerived)

frame contents (Read, Write)

TCBs, CNodes, page tables, IRQs, untyped memory (Control)

## Summary

# How to show integrity

✓ 1. Define an access control policy
   a. Identify components, i.e. label system resources
   b. Define an authority graph, i.e. arrows between components

2. Show policy refinement for the current state
   a. Show that protection state maps onto the authority graph
   b. Show well-formedness for the subject

3. The theorems establish that
   a. State changes initiated by the subject are bounded by the policy
   b. The policy is maintained for the subject

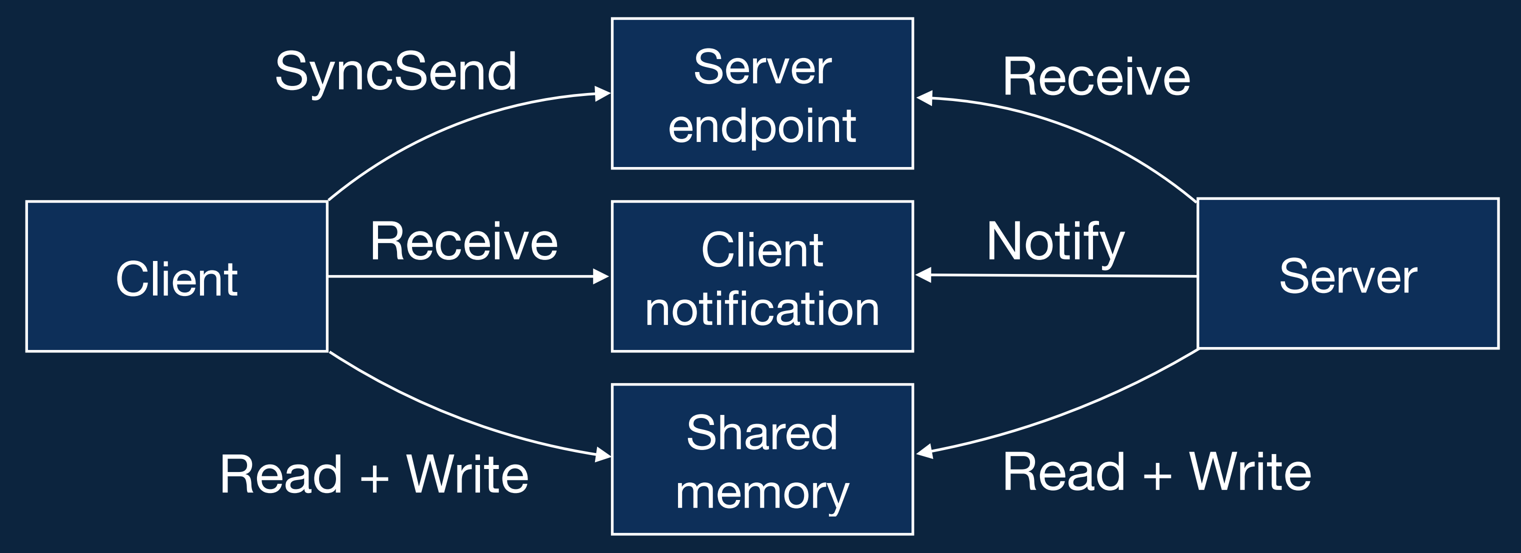
4. For static systems
   - Use a tool to check well-formedness, and a trustworthy loader

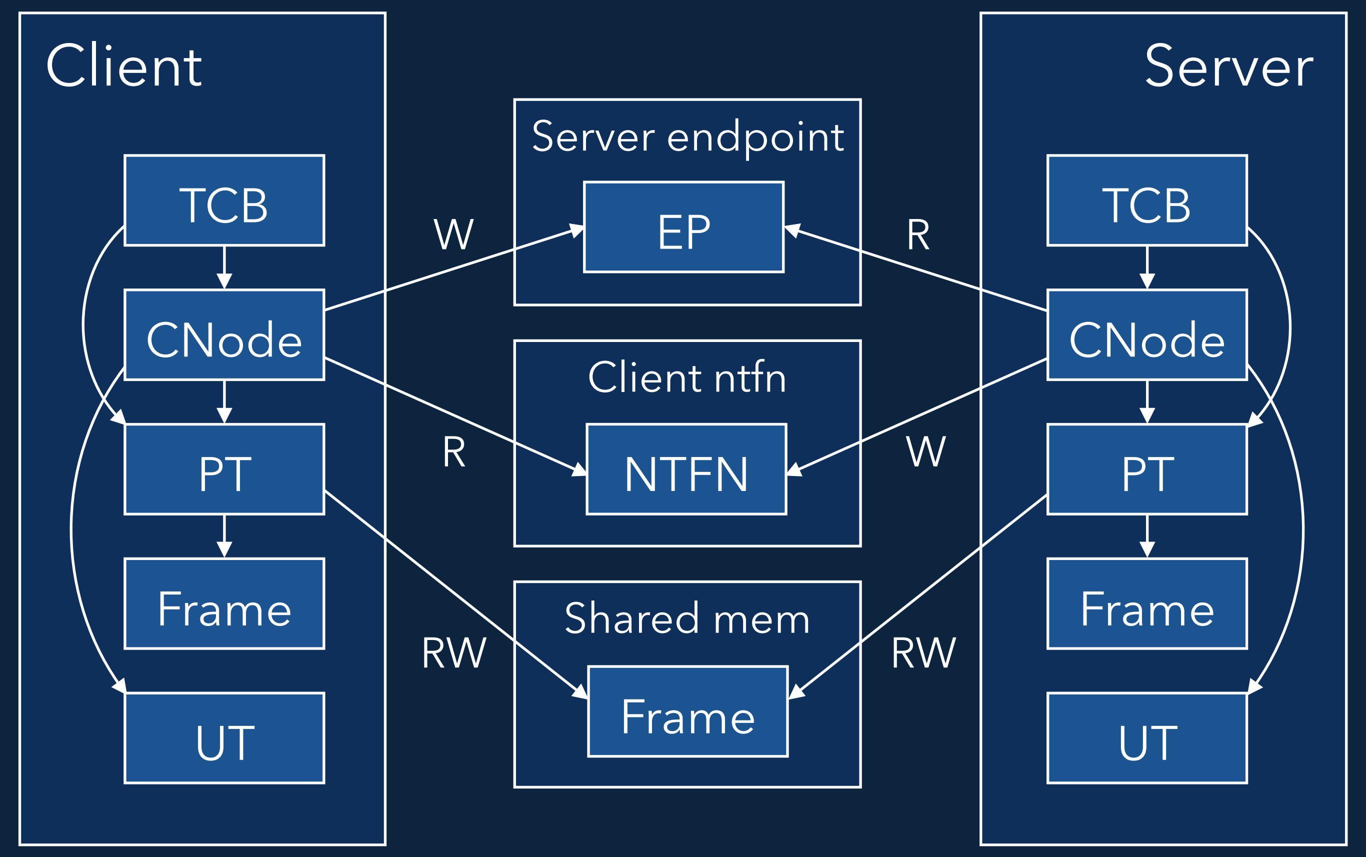
5. For dynamic systems
   - Prove that trusted components establish well-formed policies for their subordinates

# 2. Show policy refinement

a. Show that protection state maps onto the authority graph

- Every authority inherent in the state must be represented in the policy
- `pas_refined` covers all the ways authority can present

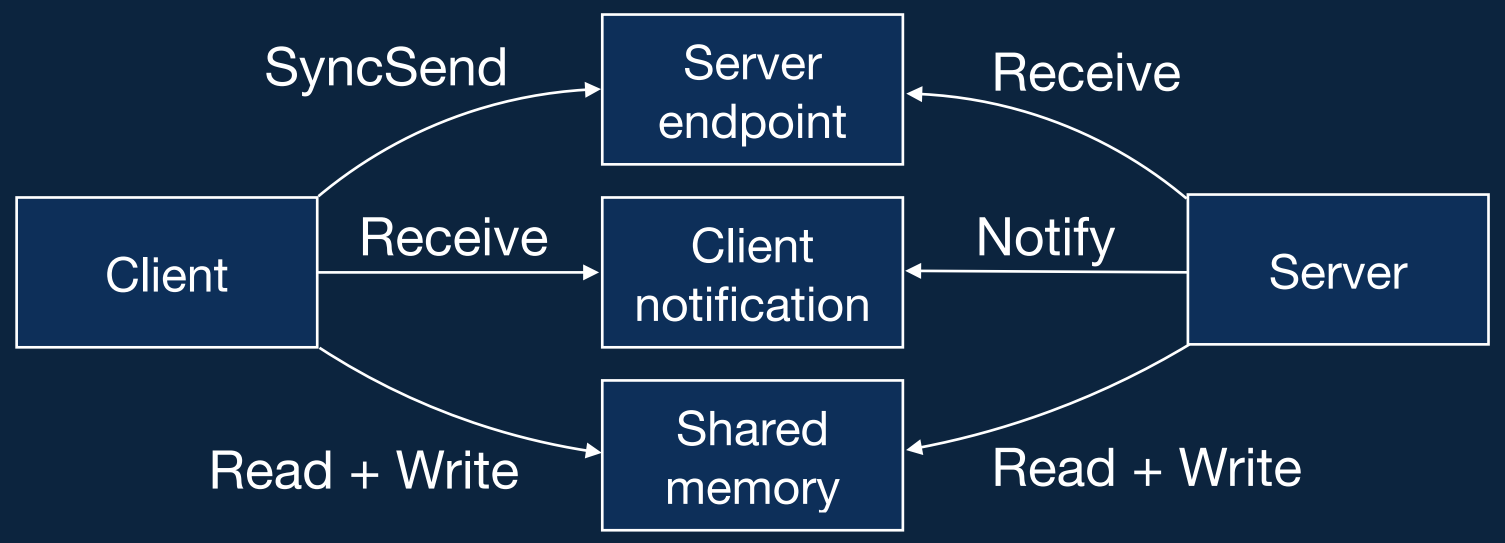


# 2. Show policy refinement

## a. Show that protection state maps onto the authority graph

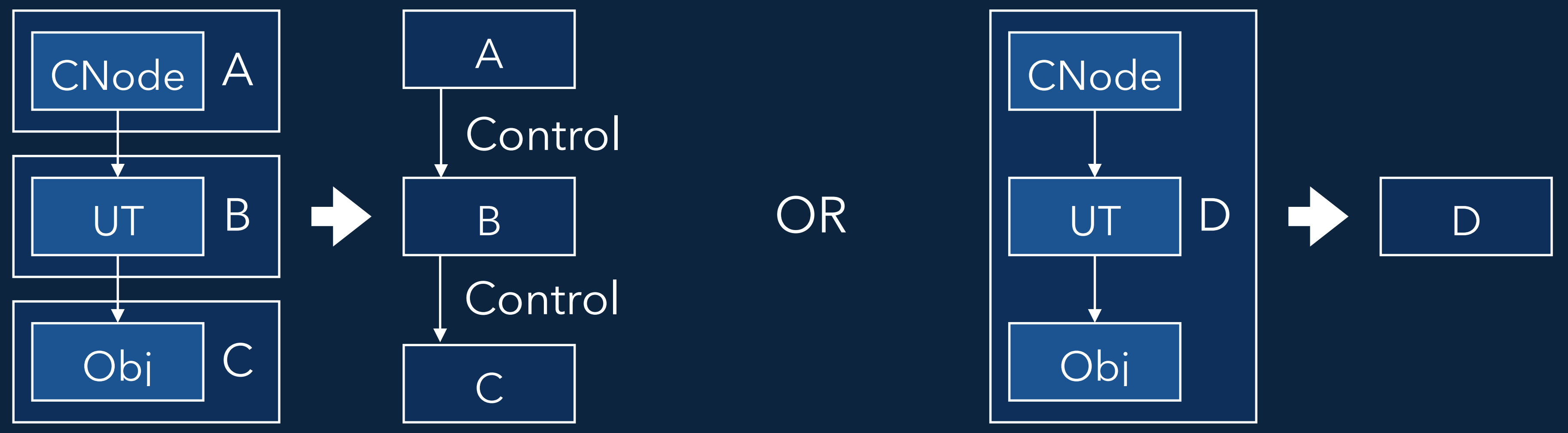
- Every authority inherent in the state must be represented in the policy
- `pas_refined` covers all the ways authority can present

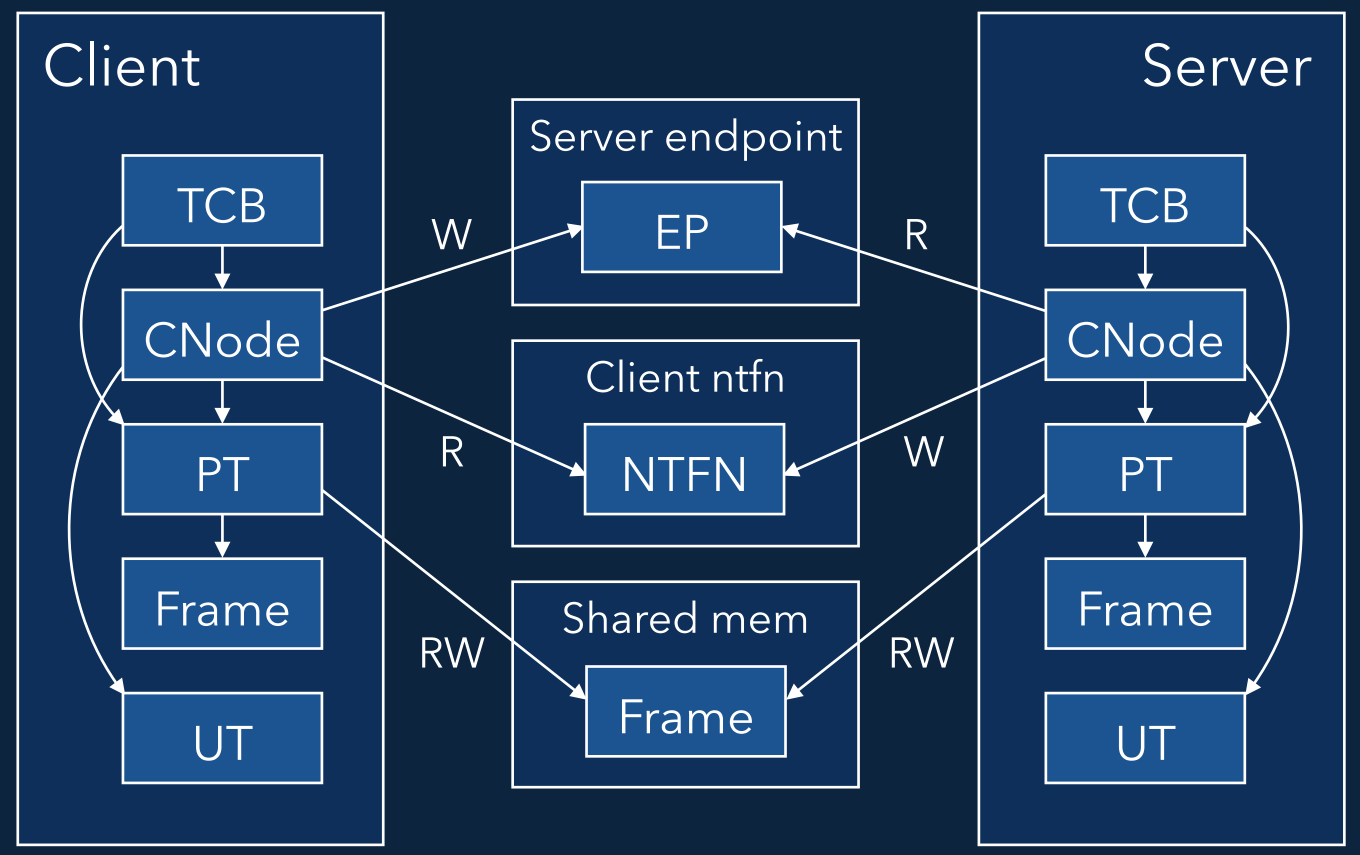



### Examples

- If a TCB has a capability to a CNode, then the TCB's component has Control over the CNode's component


- If a CNode has a capability to untyped memory, then the CNode's component has Control over the untyped memory's component, and also the components of all objects allocated from the untyped memory.

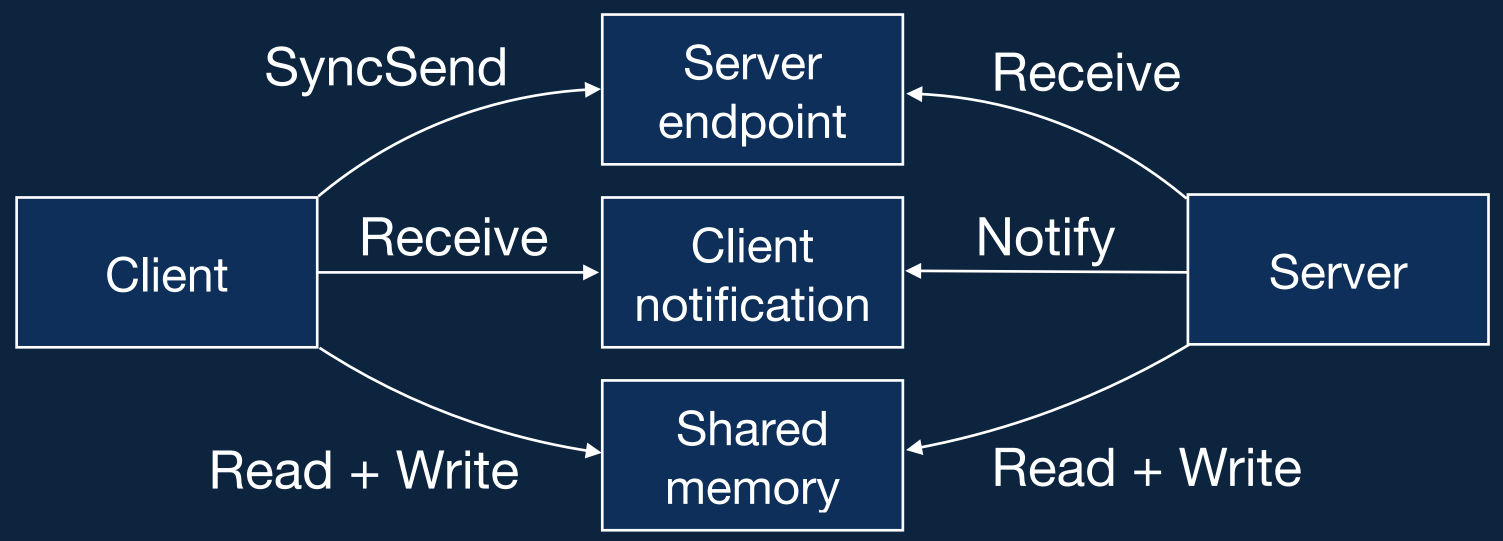


# 2. Show policy refinement

## a. Show that protection state maps onto the authority graph

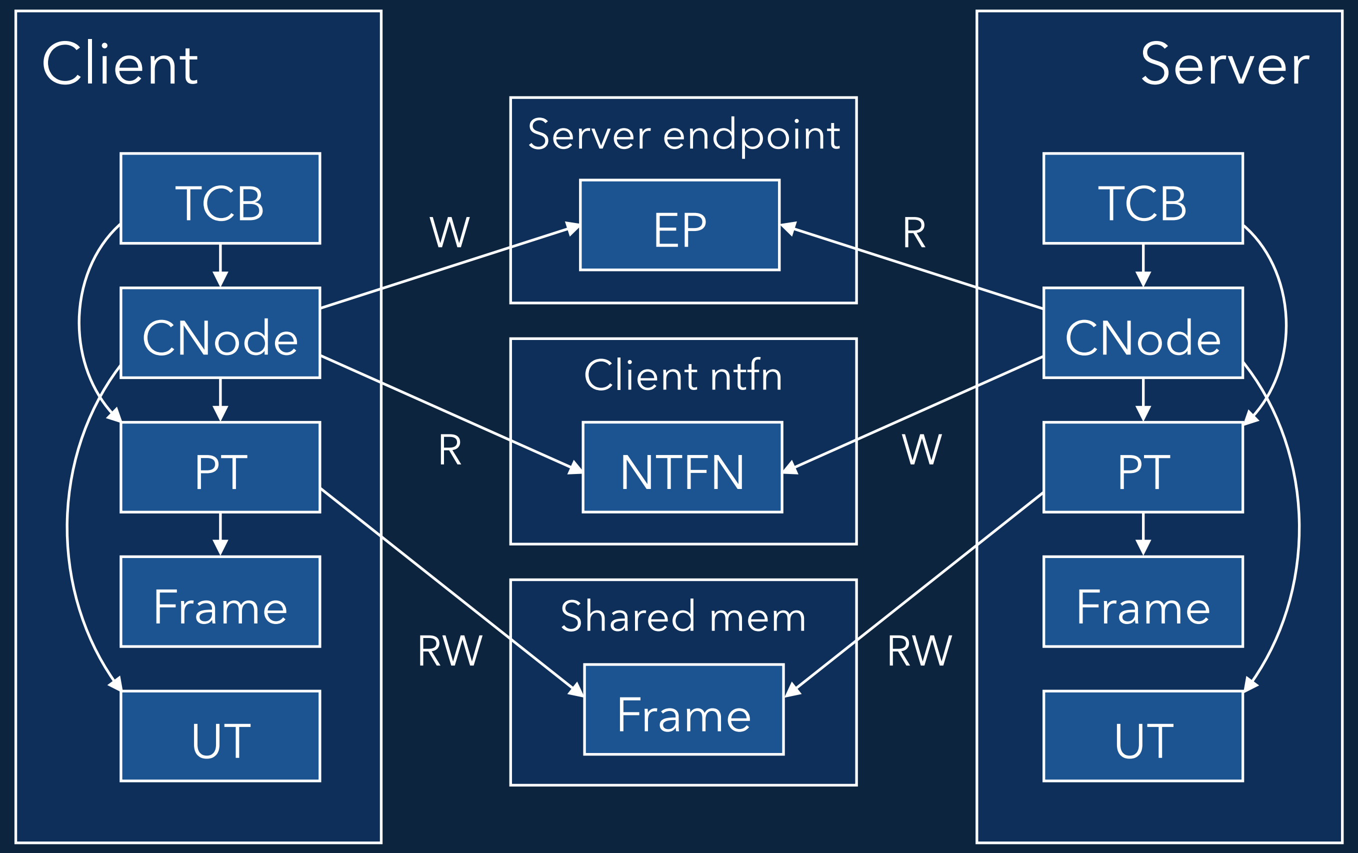
- Every authority inherent in the state must be represented in the policy
- `pas_refined` covers all the ways authority can present



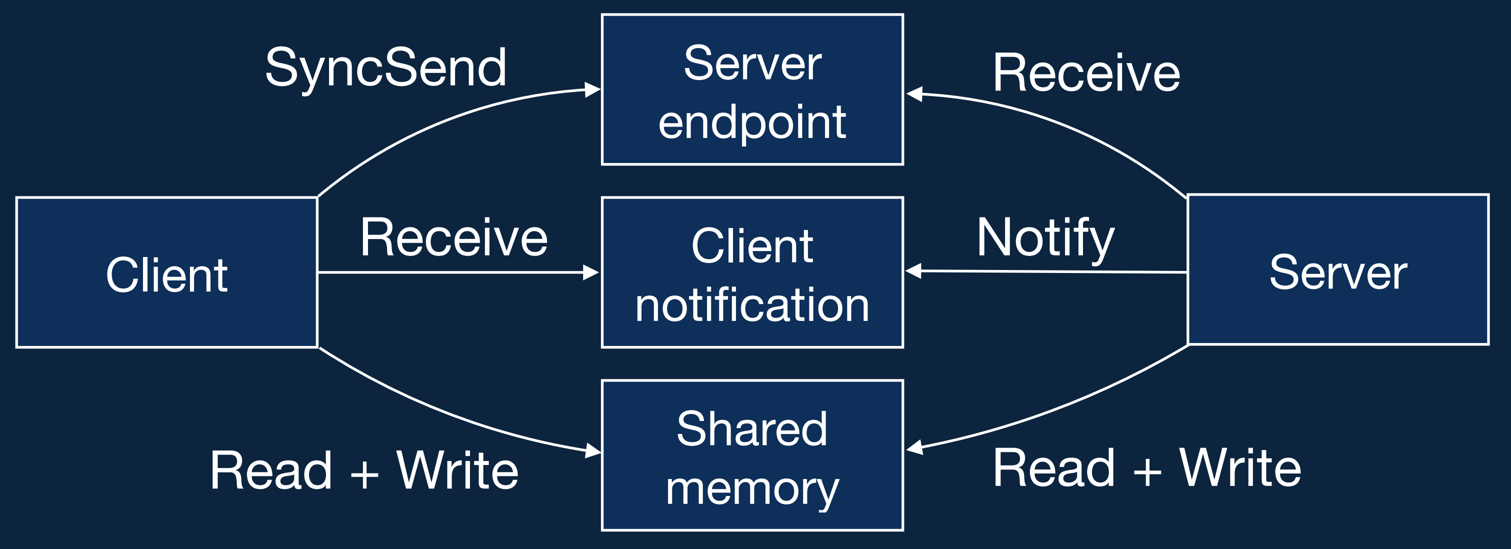

### Examples

- If a page table has a write-enabled mapping for a frame, then the page table's component has Write authority to the Frame's component


- If a TCB is blocked sending on an endpoint, then the TCB's component has SyncSend authority to the TCB's component

# 2. Show policy refinement



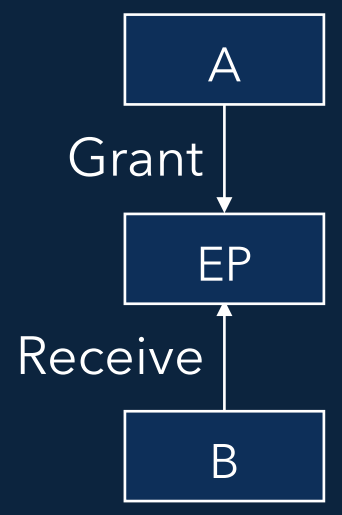

## b. Show that the policy is well-formed for the subject

- `pas_refined` imposes extra conditions called “well-formedness conditions”
- These conditions simplify the model by restricting it to sensible system designs

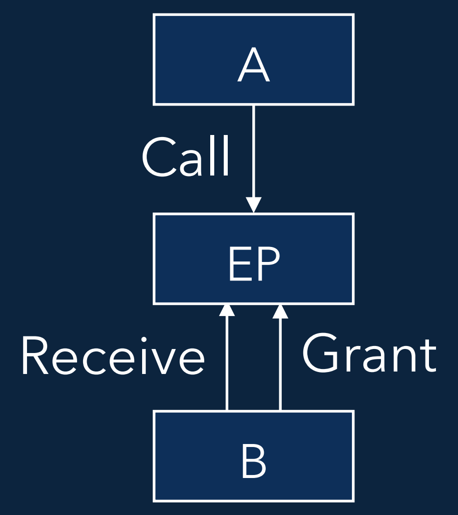
### The important conditions

- Grant authority requires mutual Control

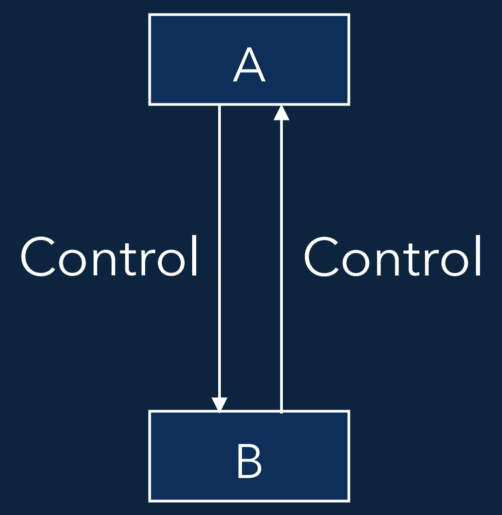
If a policy has either




or

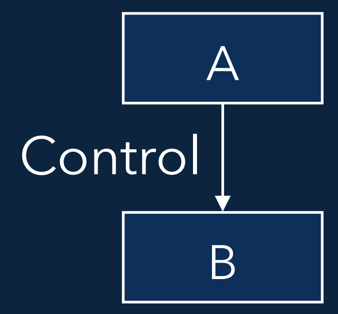



then it must also have




- The subject cannot have Control over another component

If a policy has

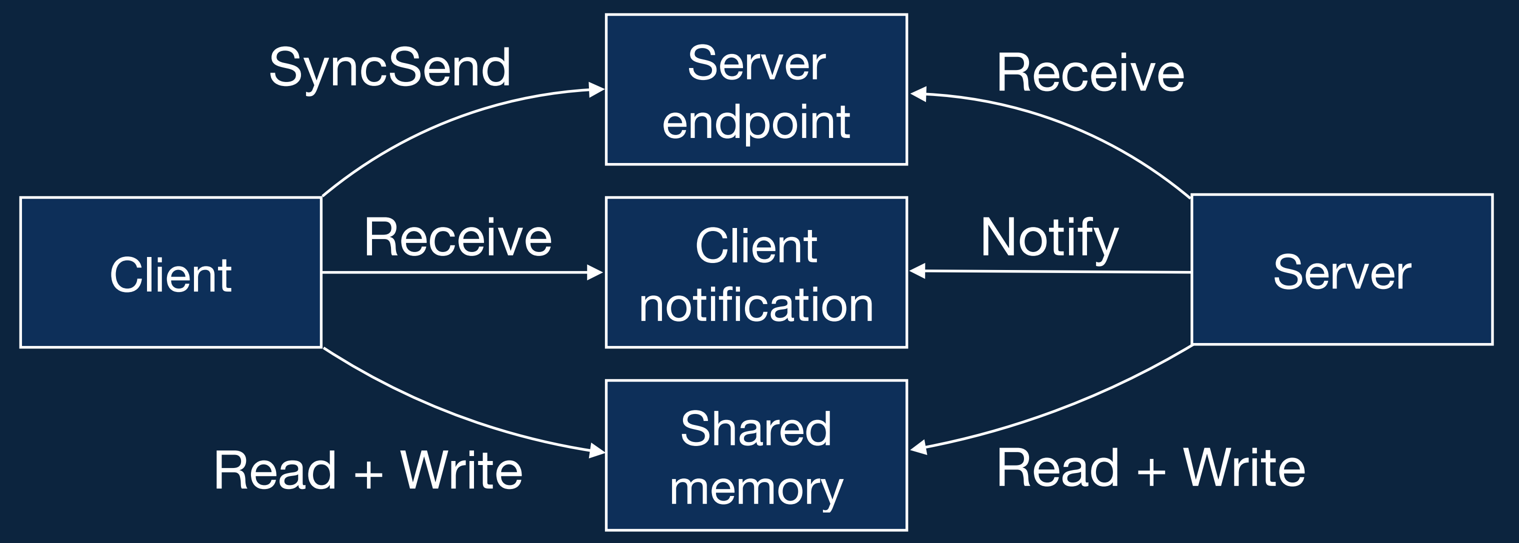



then A cannot be the subject

# 2. Show policy refinement

## b. Show that the policy is well-formed for the subject

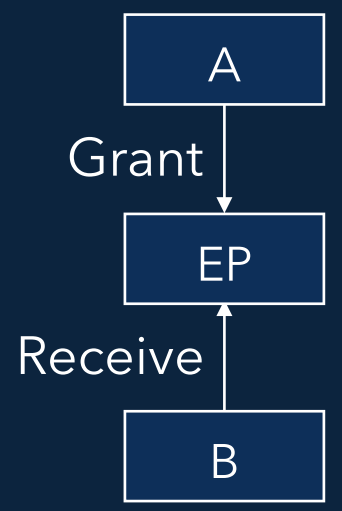
- `pas_refined` imposes extra conditions called "well-formedness conditions"
- These conditions simplify the model by restricting it to sensible system designs



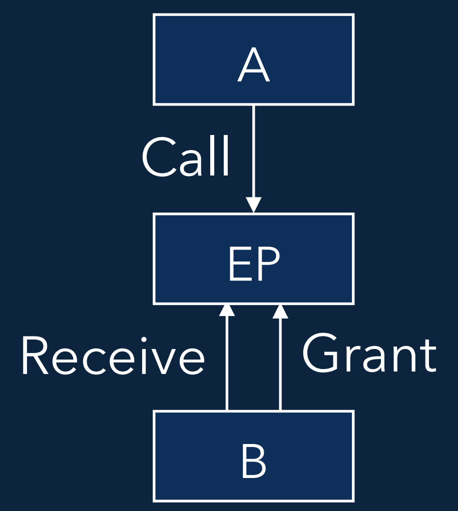

### The important conditions

- Grant authority requires mutual Control

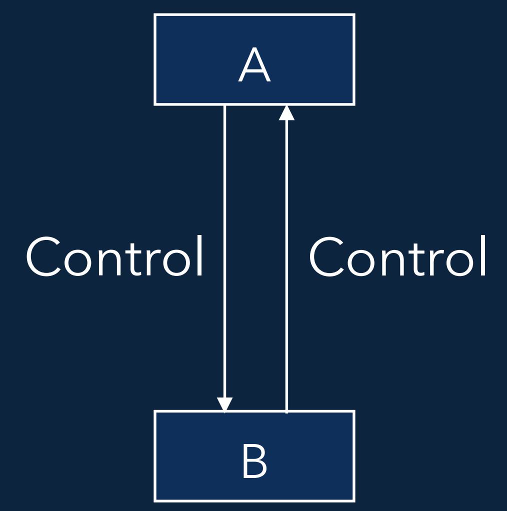
If a policy has either




or

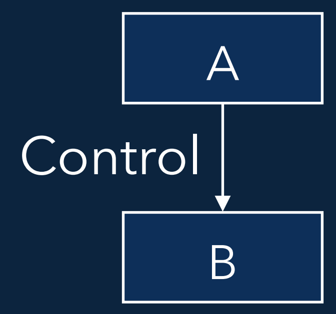



then it must also have




- The subject cannot have Control over another component

If a policy has




then A cannot be the subject

### Policies are subjective

- A policy identifies the component taking the current action

### Policy refinement is subjective

- Changing the subject may affect policy well-formedness

## Summary

# How to show integrity

✓ 1. Define an access control policy
   a. Identify components, i.e. label system resources
   b. Define an authority graph, i.e. arrows between components

✓ 2. Show policy refinement for the current state
   a. Show that protection state maps onto the authority graph
   b. Show well-formedness for the subject

3. The theorems establish that
   a. State changes initiated by the subject are bounded by the policy
   b. The policy is maintained for the subject

4. For static systems
   - Use a tool to check well-formedness, and a trustworthy loader

5. For dynamic systems
   - Prove that trusted components establish well-formed policies for their subordinates

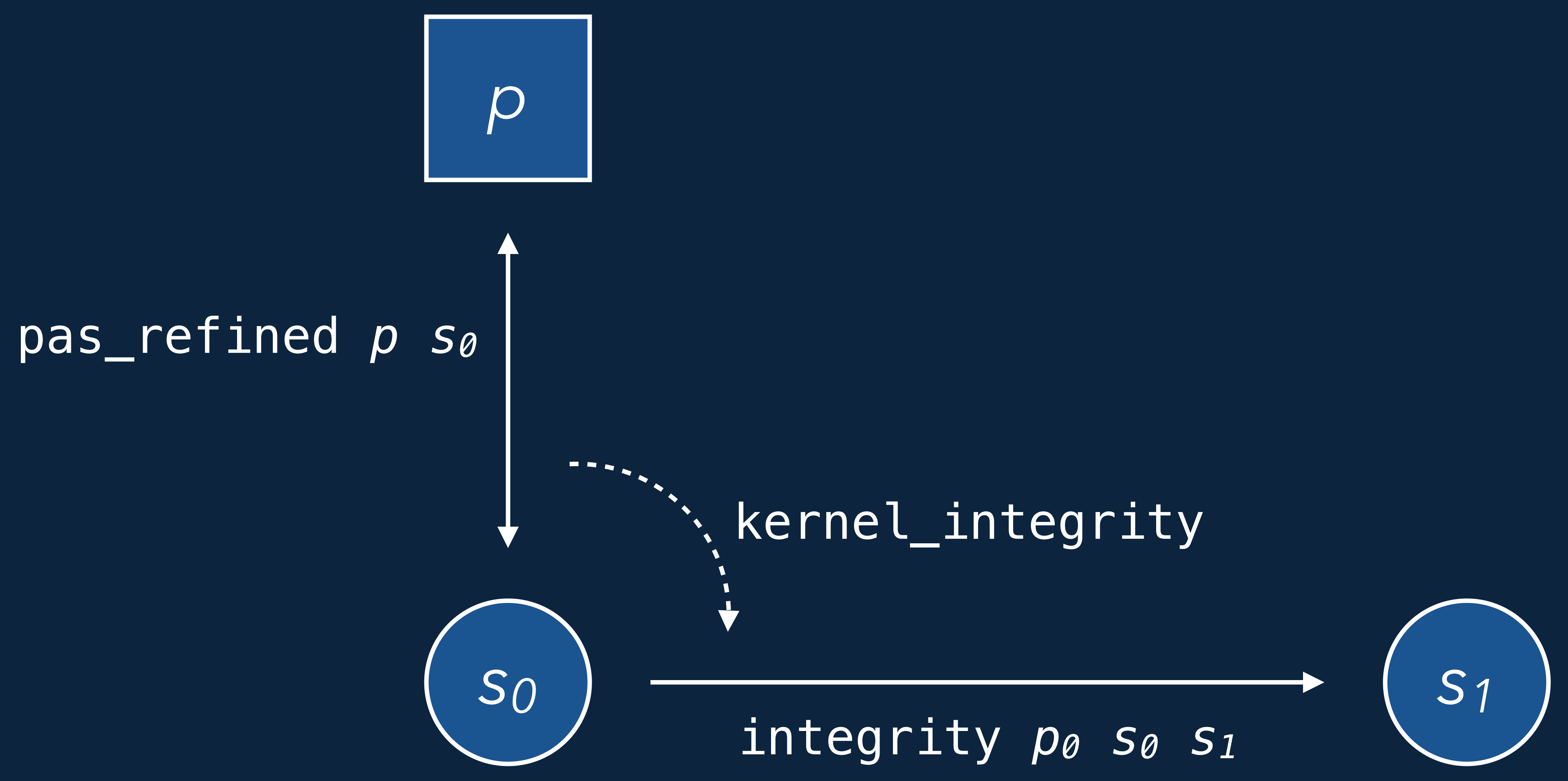
# 3. Theorems

- If a state refines a policy, and the policy is well-formed for the subject, then from that state…

## a. Integrity

- any transition will respect the policy

```
theorem kernel_integrity:
  — ‹If the subject calls the kernel
     in a state s₀ where pas_refined p s₀ is True,
     then the kernel exits in a state s₁
     where integrity p s₀ s₁ is True›
```




### Examples of changes permitted by integrity

- Frame contents may change if the subject has Write access to the frame's component
- A thread may be restarted if it's blocked receiving on an endpoint and the subject has SyncSend to the endpoint's component

# 3. Theorems

- If a state refines a policy, and the policy is well-formed for the subject, then from that state…

## a. Integrity

- any transition will respect the policy

```
theorem kernel_integrity:
  — ‹If the subject calls the kernel
     in a state s₀ where pas_refined p s₀ is True,
     then the kernel exits in a state s₁
     where integrity p s₀ s₁ is True›
```

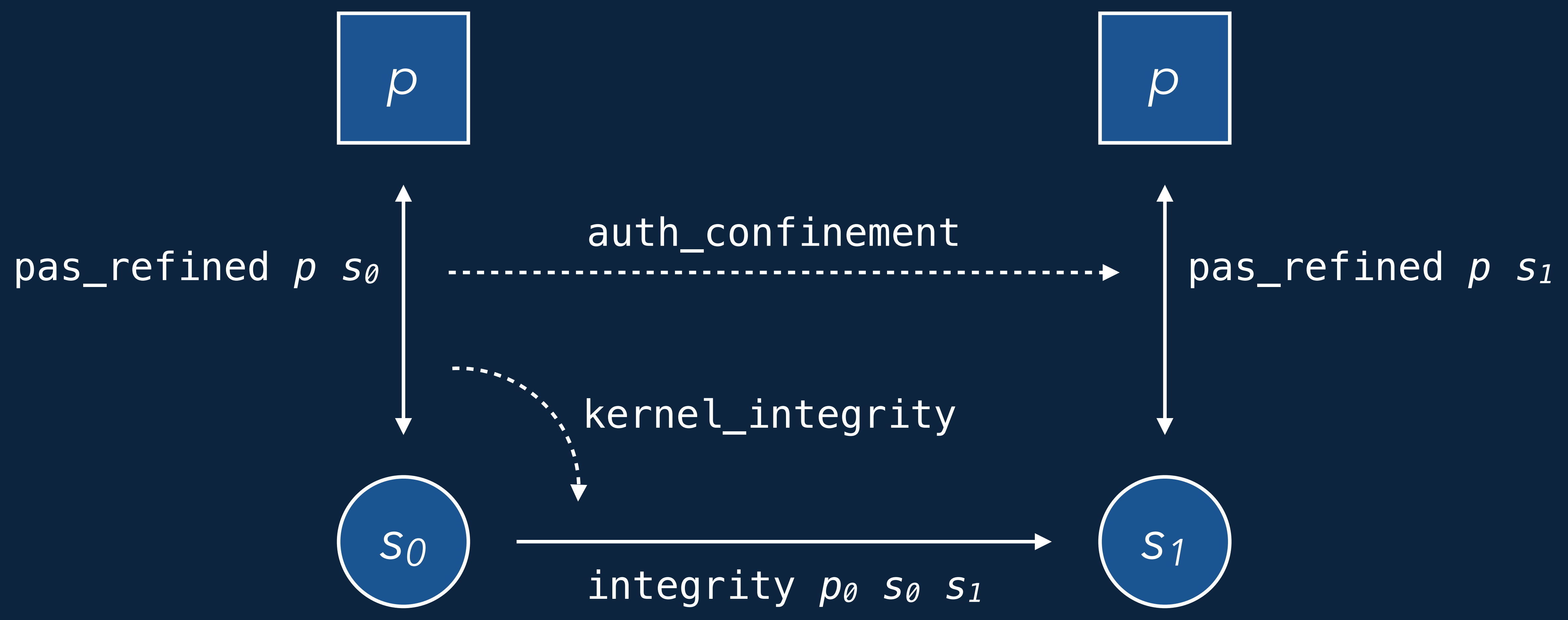
### Examples of changes permitted by integrity

- Frame contents may change if the subject has Write access to the frame's component
- A thread may be restarted if it's blocked receiving on an endpoint and the subject has SyncSend to the endpoint's component

## b. Authority confinement

- any transition will maintain the policy

```
theorem auth_confinement:
  — ‹If the subject calls the kernel
     in a state s₀ where pas_refined p s₀ is True,
     then the kernel exits in a state s₁
     where pas_refined p s₁ is True›
```



# 3. Theorems

- If a state refines a policy, and the policy is well-formed for the subject, then from that state…

## a. Integrity

- any transition will respect the policy

```
theorem kernel_integrity:
  — ‹If the subject calls the kernel
     in a state s0 where pas_refined p s0 is True,
     then the kernel exits in a state s1
     where integrity p s0 s1 is True›
```

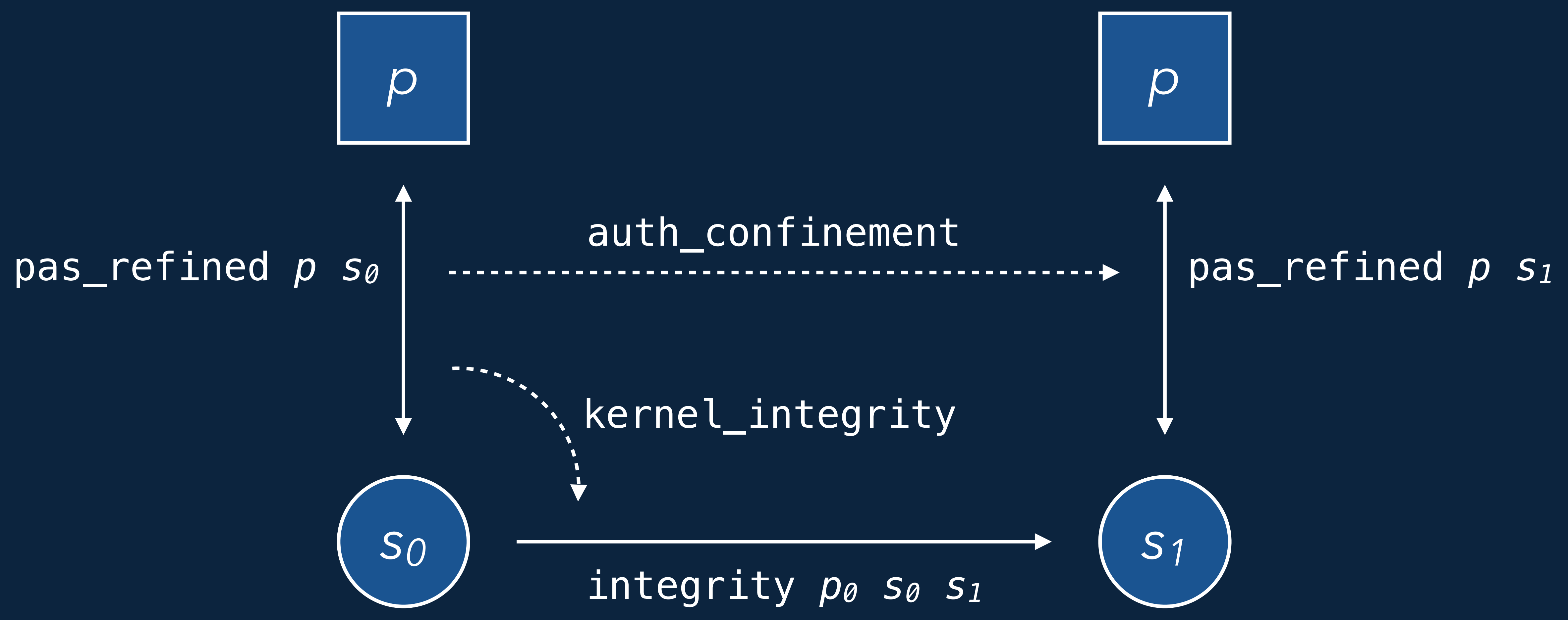
### Examples of changes permitted by integrity

- Frame contents may change if the subject has Write access to the frame's component
- A thread may be restarted if it's blocked receiving on an endpoint and the subject has SyncSend to the endpoint's component

## b. Authority confinement

- any transition will maintain the policy

```
theorem auth_confinement:
  — ‹If the subject calls the kernel
     in a state s0 where pas_refined p s0 is True,
     then the kernel exits in a state s1
     where pas_refined p s1 is True›
```




### Theorems are subjective

- They require that the current thread belongs to the subject
- The changes allowed by integrity depend on the subject

# Summary

## How to show integrity

✓ 1. Define an access control policy
   a. Identify components, i.e. label system resources
   b. Define an authority graph, i.e. arrows between components

✓ 2. Show policy refinement for the current state
   a. Show that protection state maps onto the authority graph
   b. Show well-formedness for the subject

✓ 3. The theorems establish that
   a. State changes initiated by the subject are bounded by the policy
   b. The policy is maintained for the subject

4. For static systems
   - Use a tool to check well-formedness, and a trustworthy loader

5. For dynamic systems
   - Prove that trusted components establish well-formed policies for their subordinates

# Subjectivity

- The component currently taking an action is called the "subject"

## Policies are subjective

- Every policy identifies one of its components as the current subject

## Policy refinement is subjective

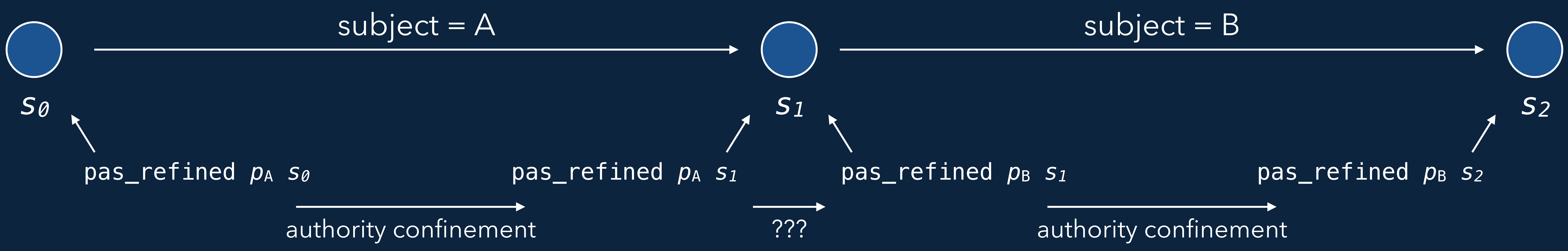
- The well-formedness of a policy depends on the choice of subject
- The subject may not have Control over another component

## The theorems are subjective

- The current thread must belong to the current subject
- Changes permitted by integrity depend on the subject

## Switching subjects requires switching policies

- What gives us the right to do that?



# 4. Static Systems

## Constraints

- No component has Control over another component
  - No authority to redistribute resources

## Payoff

- Without Control, policy well-formedness is no longer subjective
  - Therefore, policy switches are free!
- If policy refinement holds for the initial state, then it holds always

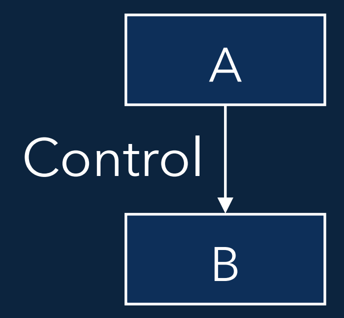
## To ensure integrity

- Use a system build tool that generates capDL
  - It should check well-formedness for all components
- Use a verified capDL loader

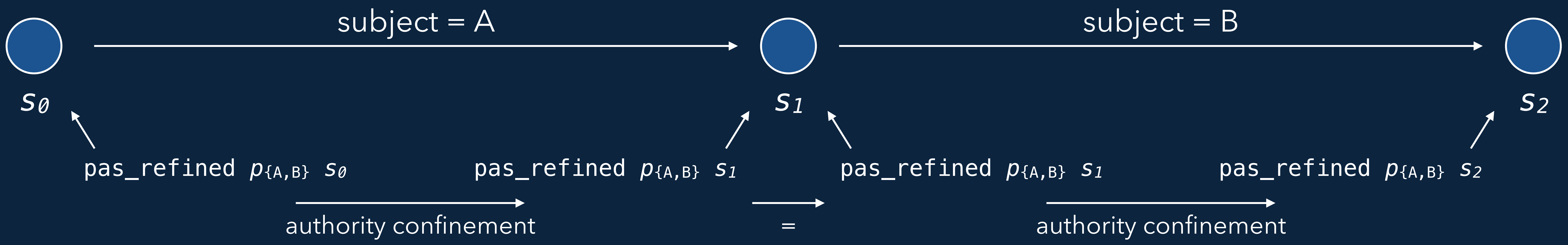
## Subjectivity of well-formedness

- The subject cannot have Control over another component

If a policy has



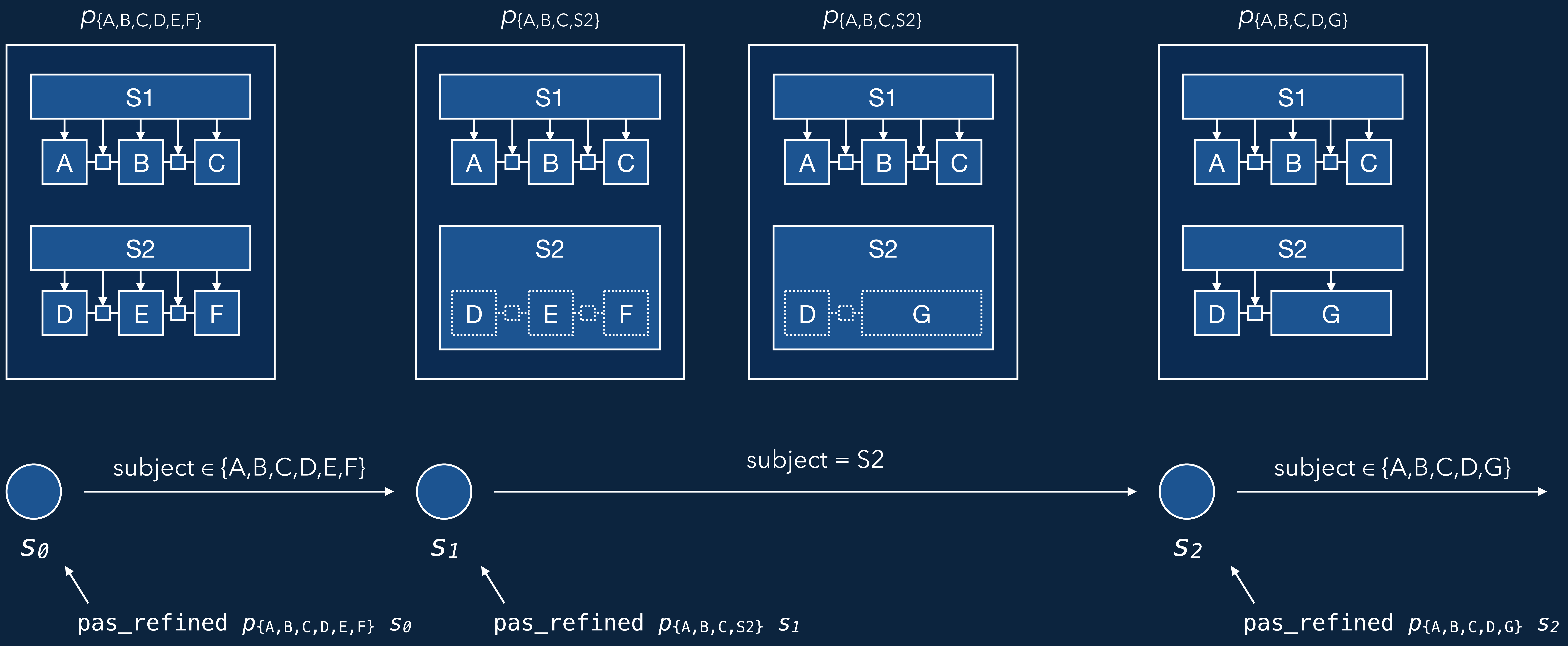

then A cannot be the subject



# 5. Dynamic Systems

Resources may be reconfigured by a trusted component

- A trusted component may have Control over its subordinates
  - To treat it as subject, we need to redraw its boundary around its subordinates
  - Switching away from a trusted component requires proof that it establishes a new well-formed policy



# Summary

## How to show integrity

✓ 1. Define an access control policy
   a. Identify components, i.e. label system resources
   b. Define an authority graph, i.e. arrows between components

✓ 2. Show policy refinement for the current state
   a. Show that protection state maps onto the authority graph
   b. Show well-formedness for the subject

✓ 3. The theorems establish that
   a. State changes initiated by the subject are bounded by the policy
   b. The policy is maintained for the subject

✓ 4. For static systems
   - Use a tool to check well-formedness, and a trustworthy loader

✓ 5. For dynamic systems
   - Prove that trusted components establish well-formed policies for their subordinates