**Paper:** https://doi.org/jzwj
**Artifact:** https://doi.org/jzwk

# Formalising the Prevention of Microarchitectural Timing Channels by Operating Systems

Formal Methods (FM), 7 March 2023

Robert Sison[1,2], Scott Buckley[2],
Toby Murray[1], Gerwin Klein[3,2], and Gernot Heiser[2]




**[1] The University of Melbourne, Australia**

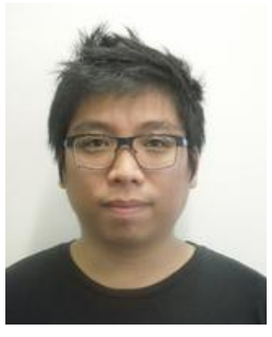
**[2] UNSW Sydney, Australia**

**[3] Proofcraft, Sydney, Australia**

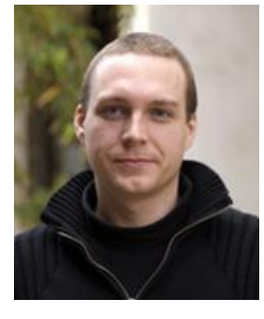
**Paper:** https://doi.org/jzwj
**Artifact:** https://doi.org/jzwk

# Formalising the Prevention of Microarchitectural Timing Channels by Operating Systems

Formal Methods (FM), 7 March 2023



Robert Sison[1,2], Scott Buckley[2],
Toby Murray[1], Gerwin Klein[3,2], and Gernot Heiser[2]



[1] The University of Melbourne, Australia

[2] UNSW Sydney, Australia

[3] Proofcraft, Sydney, Australia

# Threat scenario: *Trojan* and *spy*


# Threat scenario: *Trojan* and *spy*

Covert channels


# Threat scenario:

## *Victim/~~Trojan~~* and *spy?*

Covert channels
+
Side channels



# Threat scenario:

## *Victim/~~Trojan~~* and *spy?*

Covert channels ✓

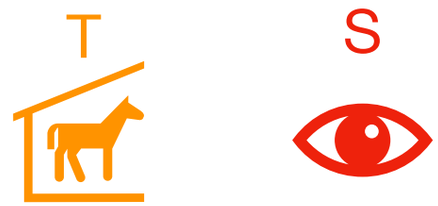
\+

Side channels ✓



# Threat scenario: *Trojan* and *spy*

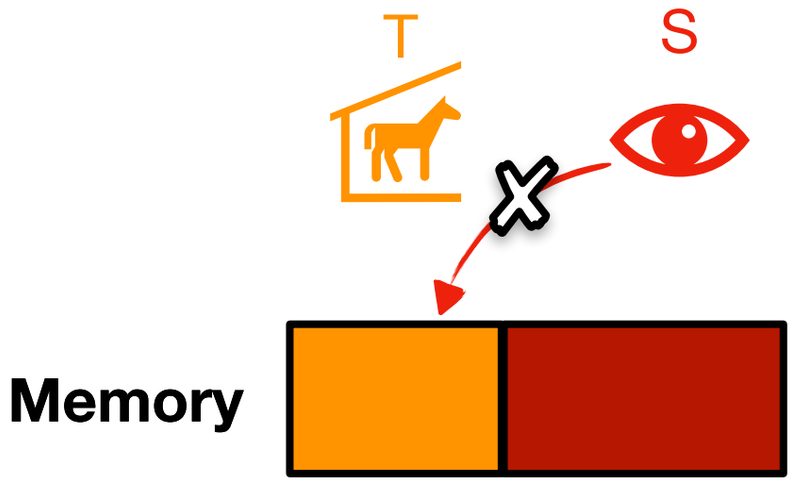
Covert channels
+
Side channels




# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

- OSes typically implement *memory protection.*

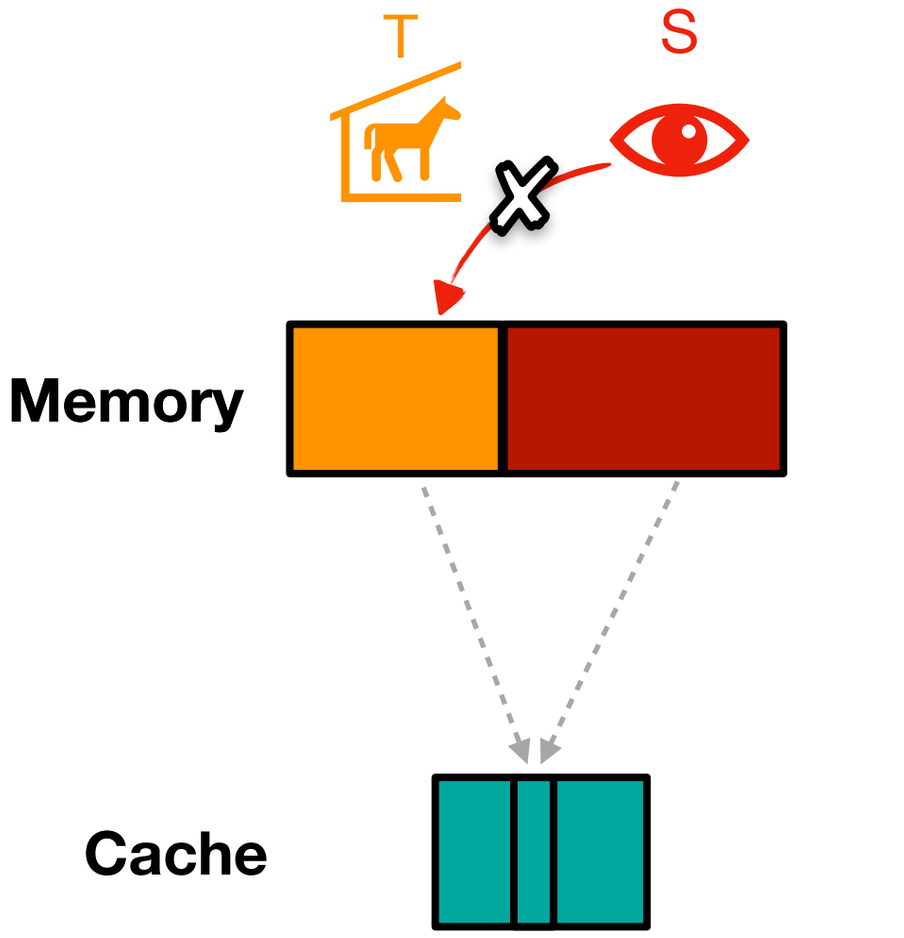


# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

- OSes typically implement *memory protection.*

- But: Mere memory access can change the microarch. state — this affects timing.

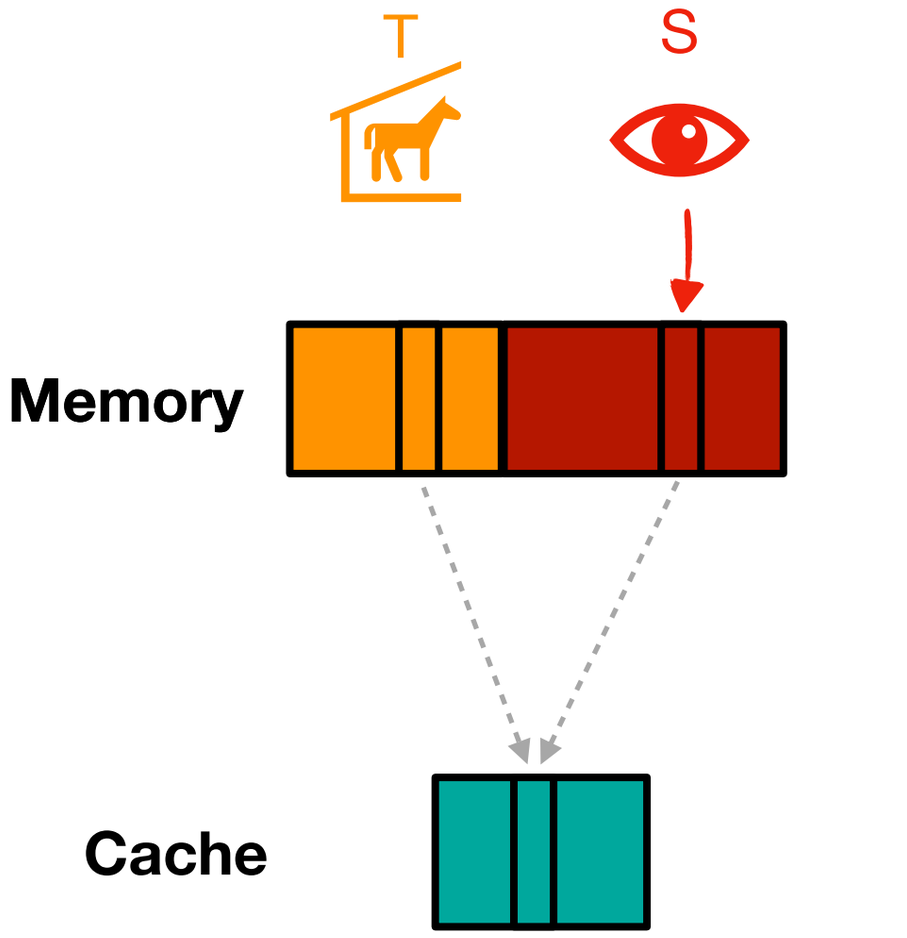


# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

- OSes typically implement *memory protection.*

- But: Mere memory access can change the microarch. state — this affects timing.

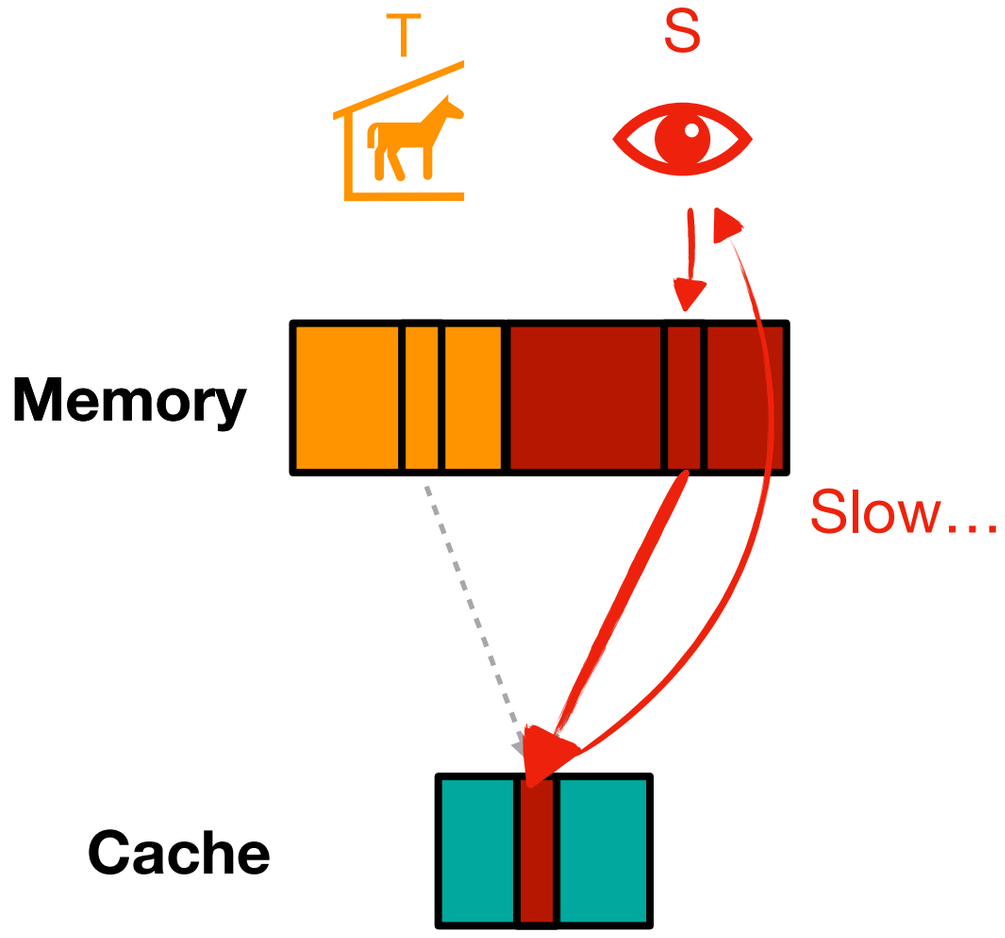


# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

- OSes typically implement *memory protection.*

- But: Mere memory access can change the microarch. state — this affects timing.

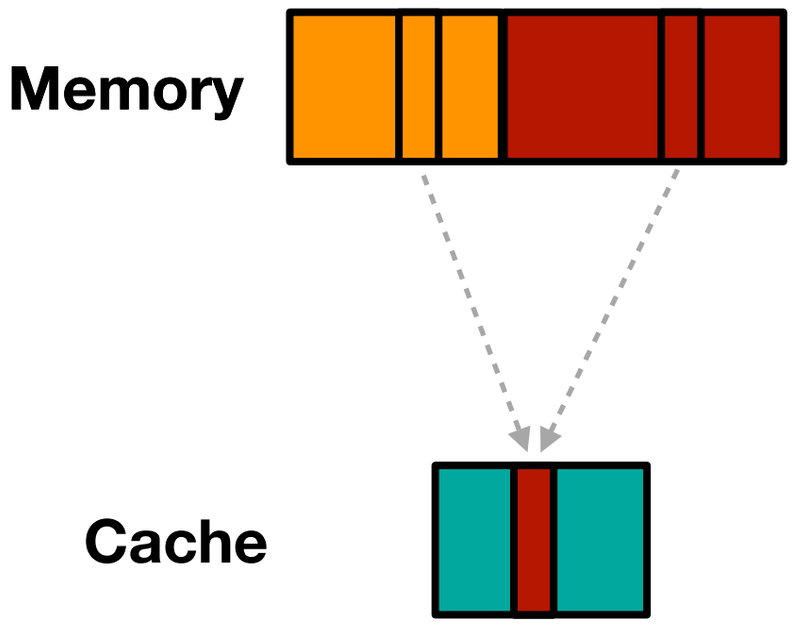


# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels


- OSes typically implement *memory protection.*

- But: <u>Mere memory access</u> can change the microarch. state — this affects timing.

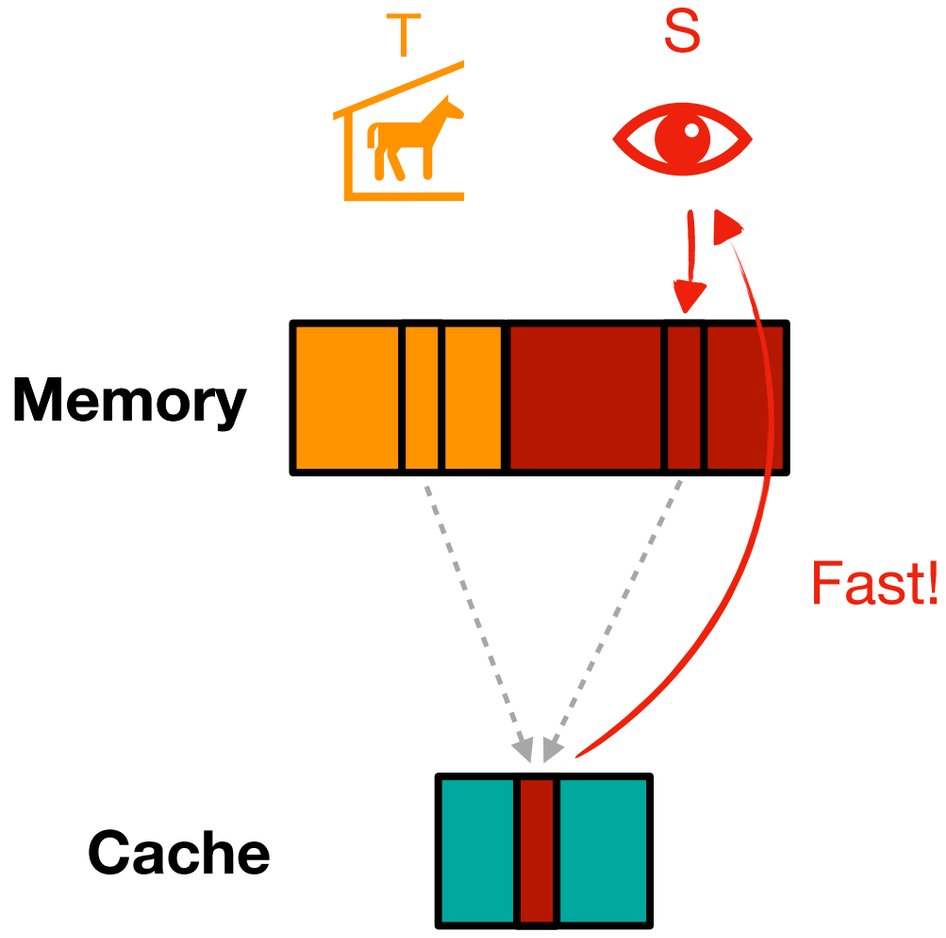


# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

- OSes typically implement *memory protection.*

- But: Mere memory access can change the microarch. state — this affects timing.

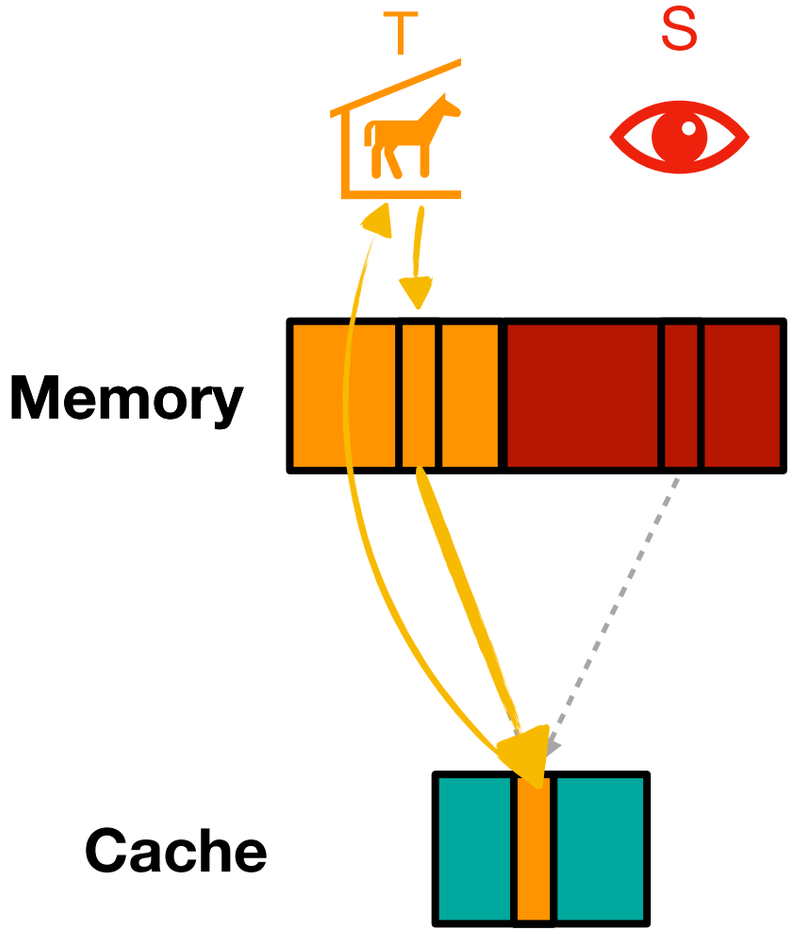


# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

- OSes typically implement *memory protection.*

- But: Mere memory access can change the microarch. state — this affects timing.

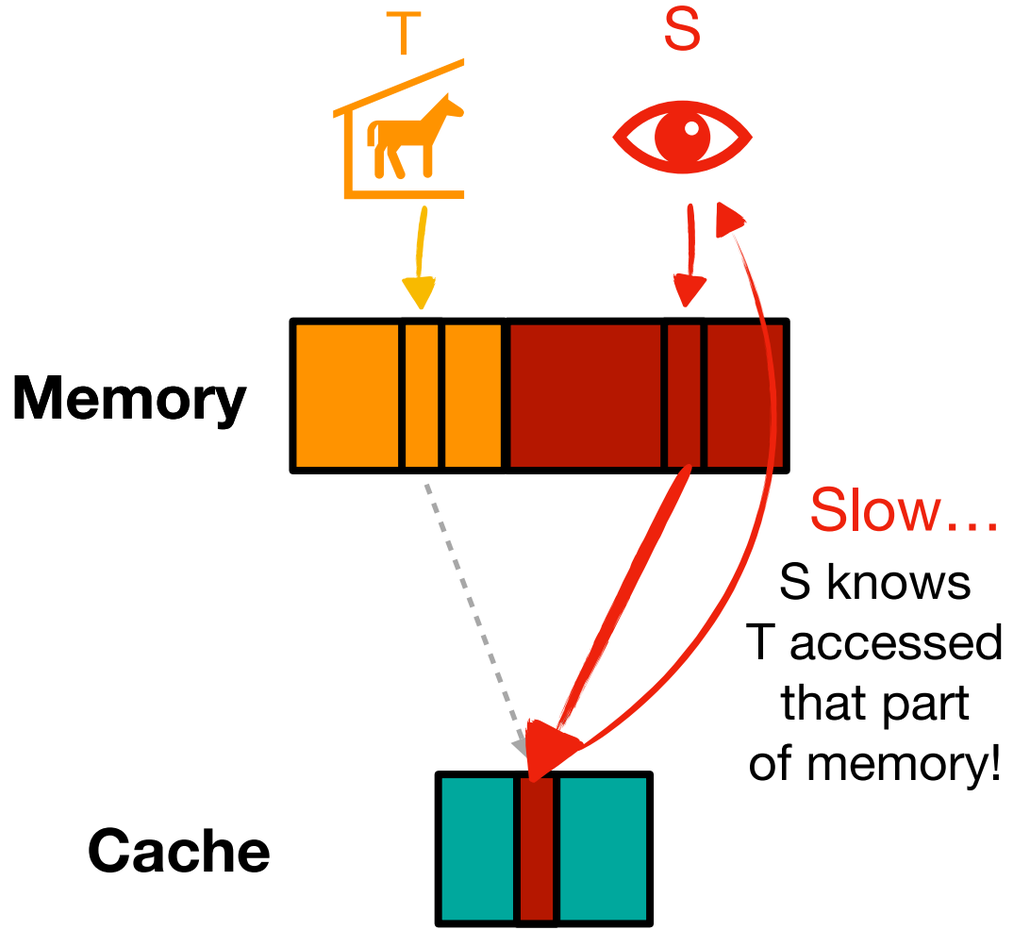


# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

- OSes typically implement *memory protection.*

- But: Mere memory access can change the microarch. state — this affects timing.



# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

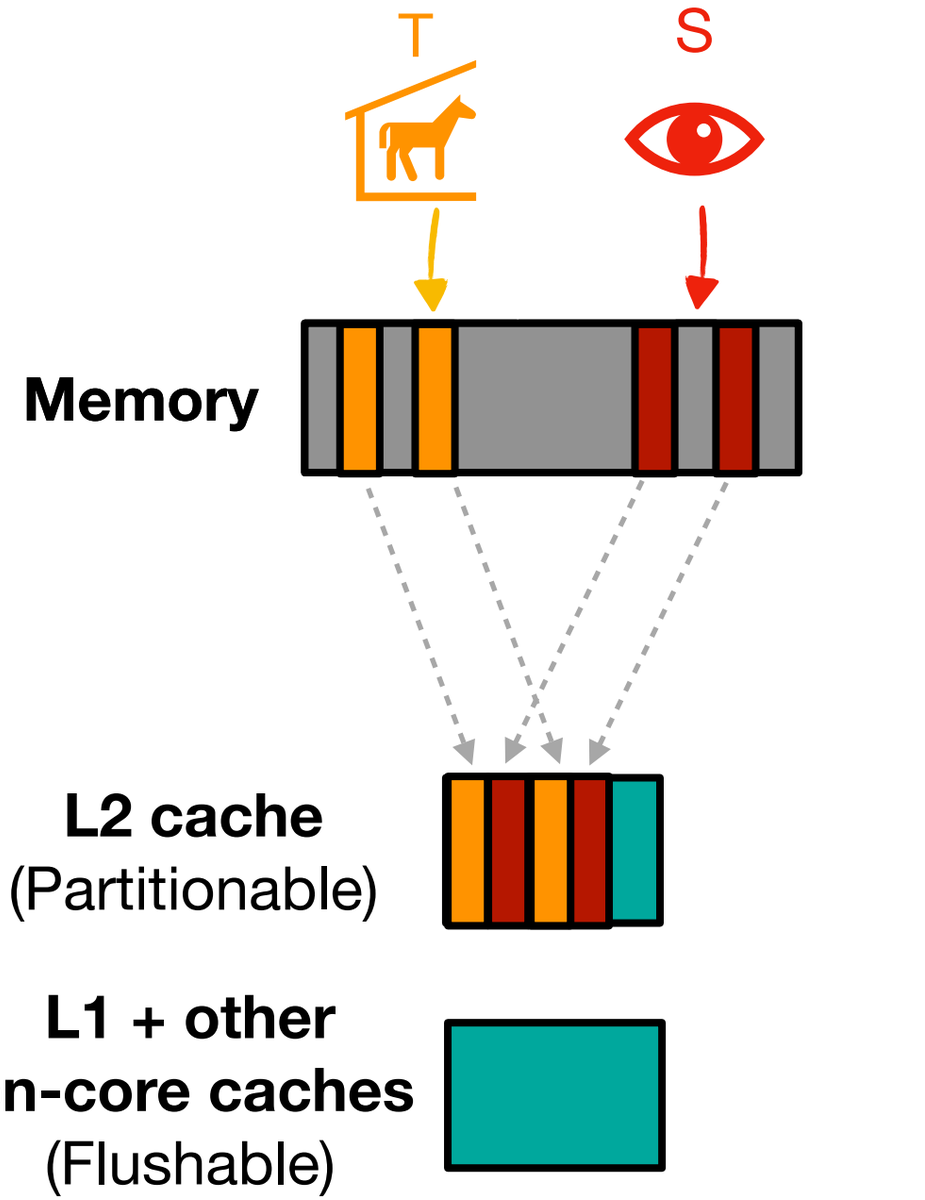
- OSes typically implement *memory protection.*

- But: Mere memory access can change the microarch. state — this affects timing.

- To prevent these *timing channels*, OSes can implement *time protection*:
  e.g. [Ge et al. 2019] for seL4 microkernel OS

  - Partition what we can

  - Flush what we can't



# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

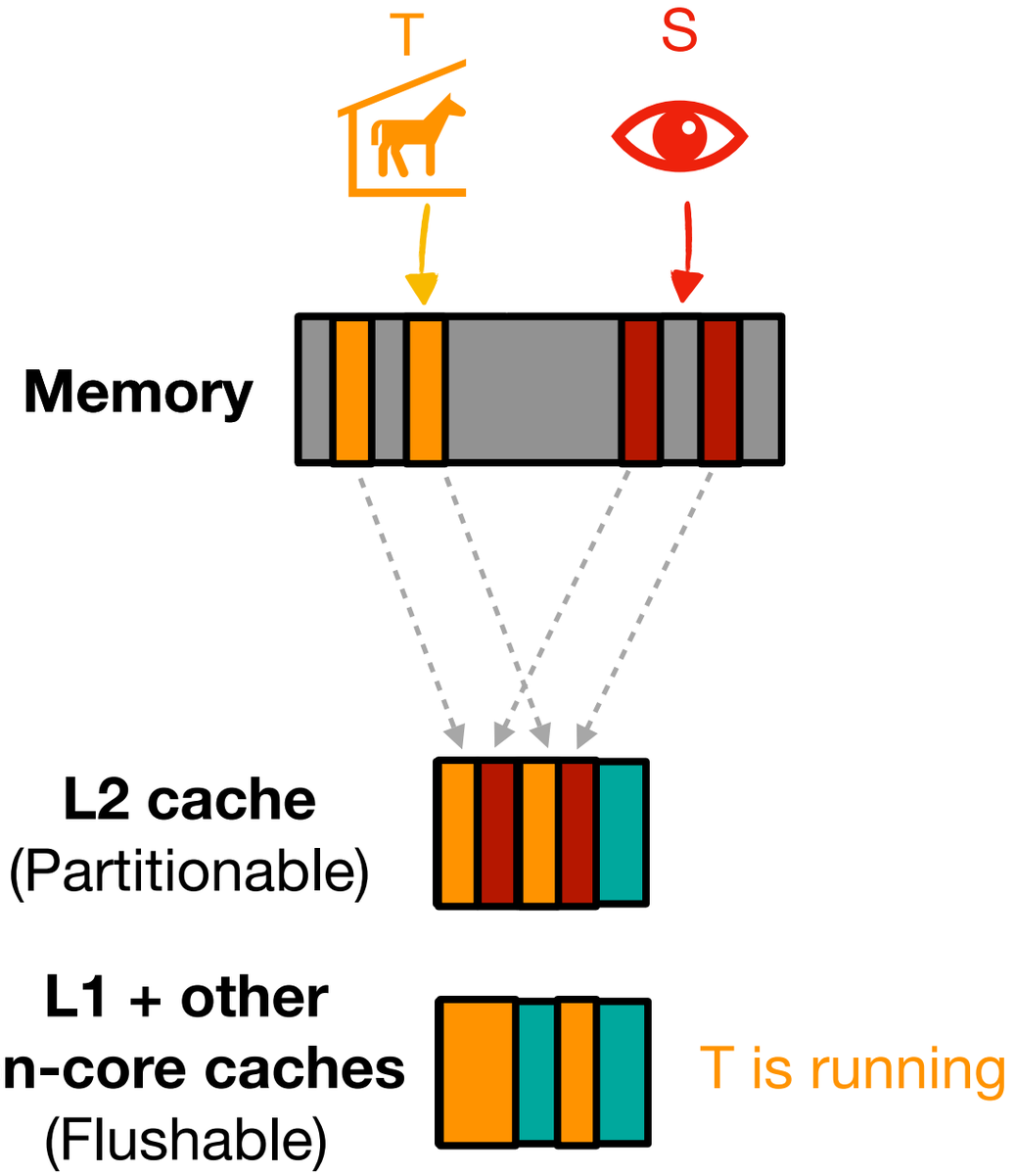
- OSes typically implement *memory protection.*

- But: Mere memory access can change the microarch. state — this affects timing.

- To prevent these *timing channels*, OSes can implement *time protection*:
  e.g. [Ge et al. 2019] for seL4 microkernel OS

  - Partition what we can

  - Flush what we can't



# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

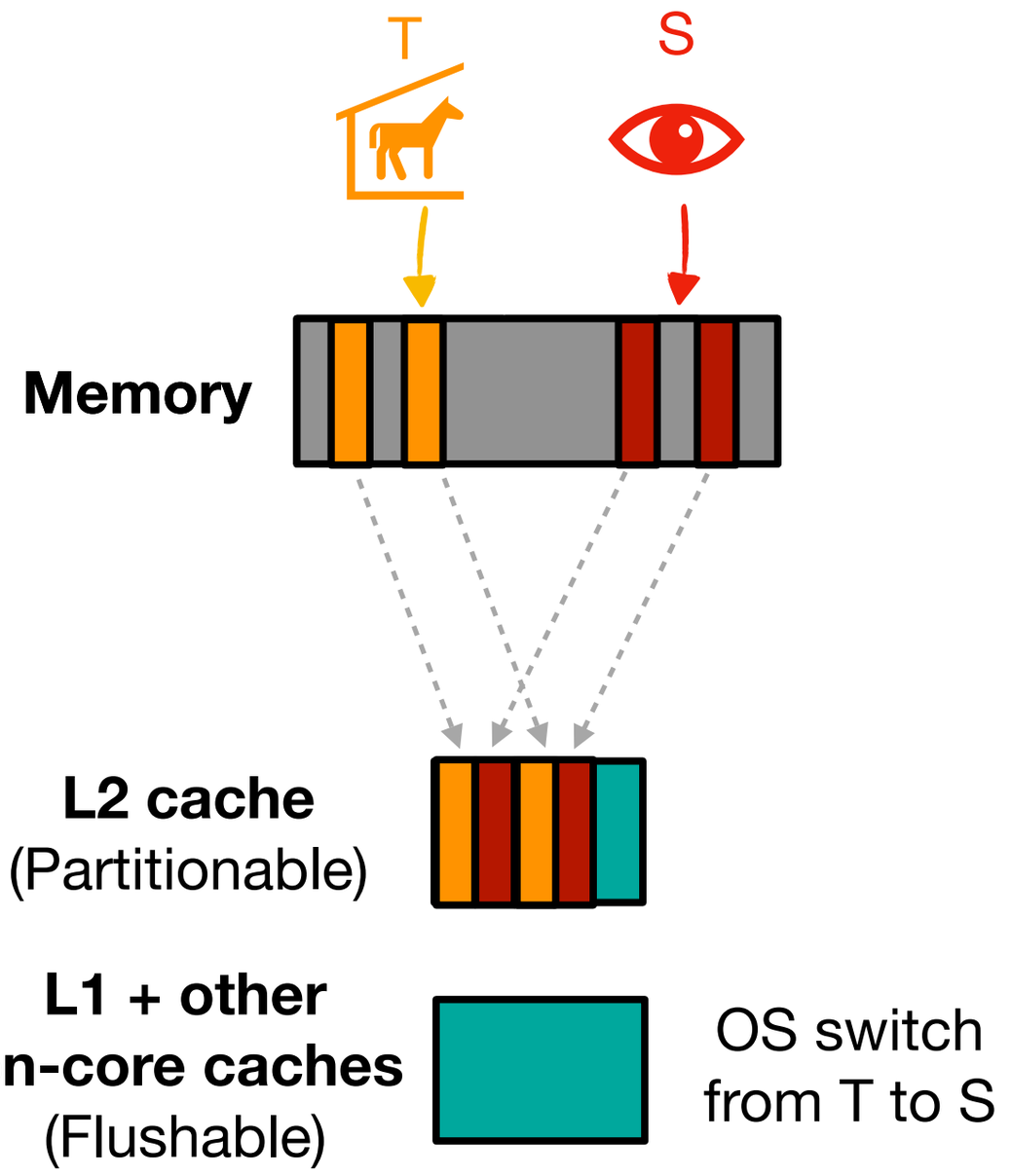
- OSes typically implement *memory protection.*

- But: Mere memory access can change the microarch. state — this affects timing.

- To prevent these *timing channels*, OSes can implement *time protection*:
  e.g. [Ge et al. 2019] for seL4 microkernel OS

  - Partition what we can

  - Flush what we can't




"Flush": Write fixed content; wait up to fixed time.

# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

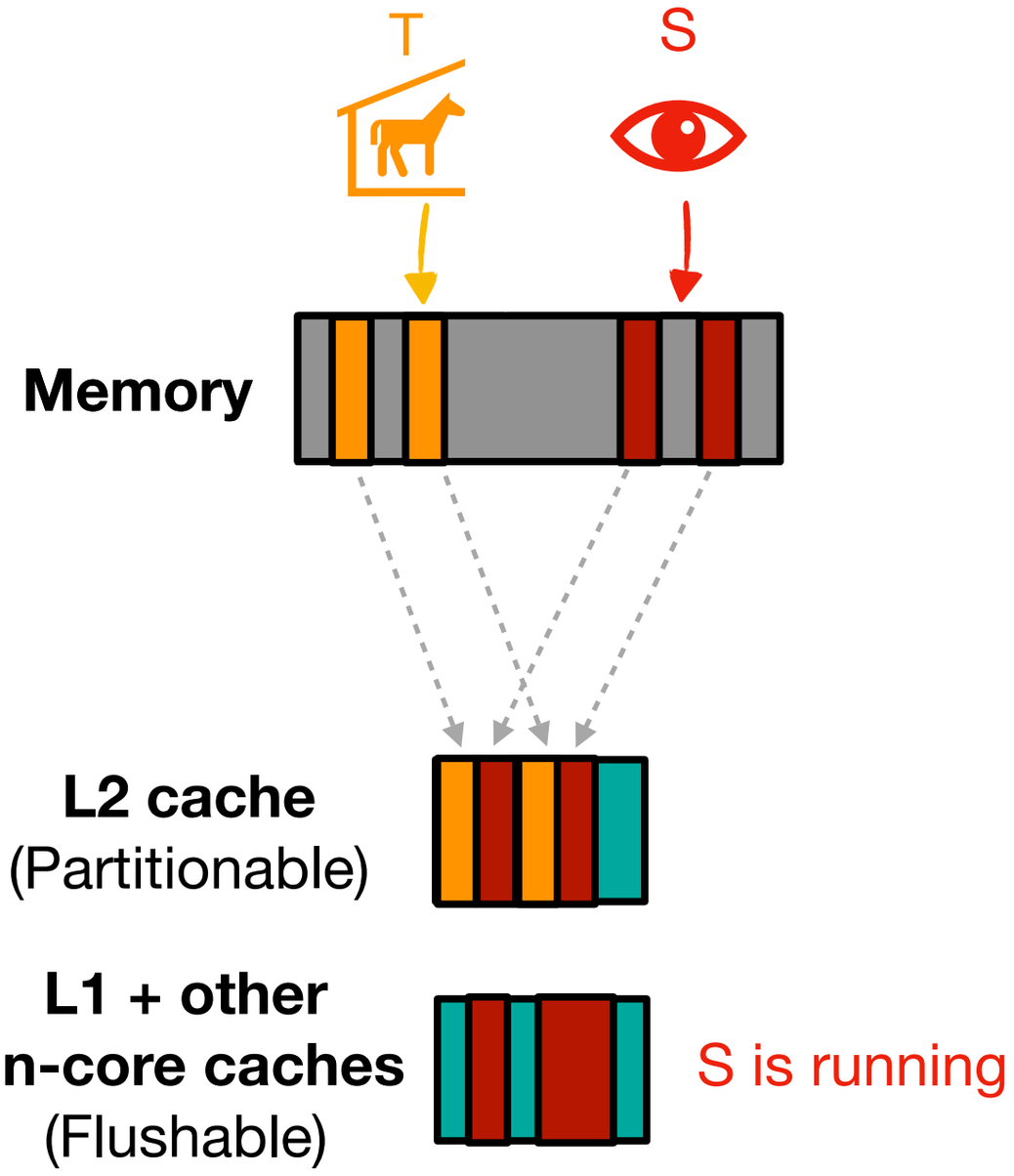
- OSes typically implement *memory protection.*

- But: Mere memory access can change the microarch. state — this affects timing.

- To prevent these *timing channels*, OSes can implement *time protection*:
  e.g. [Ge et al. 2019] for seL4 microkernel OS

  - Partition what we can

  - Flush what we can't




"Flush": Write fixed content; wait up to fixed time.

# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

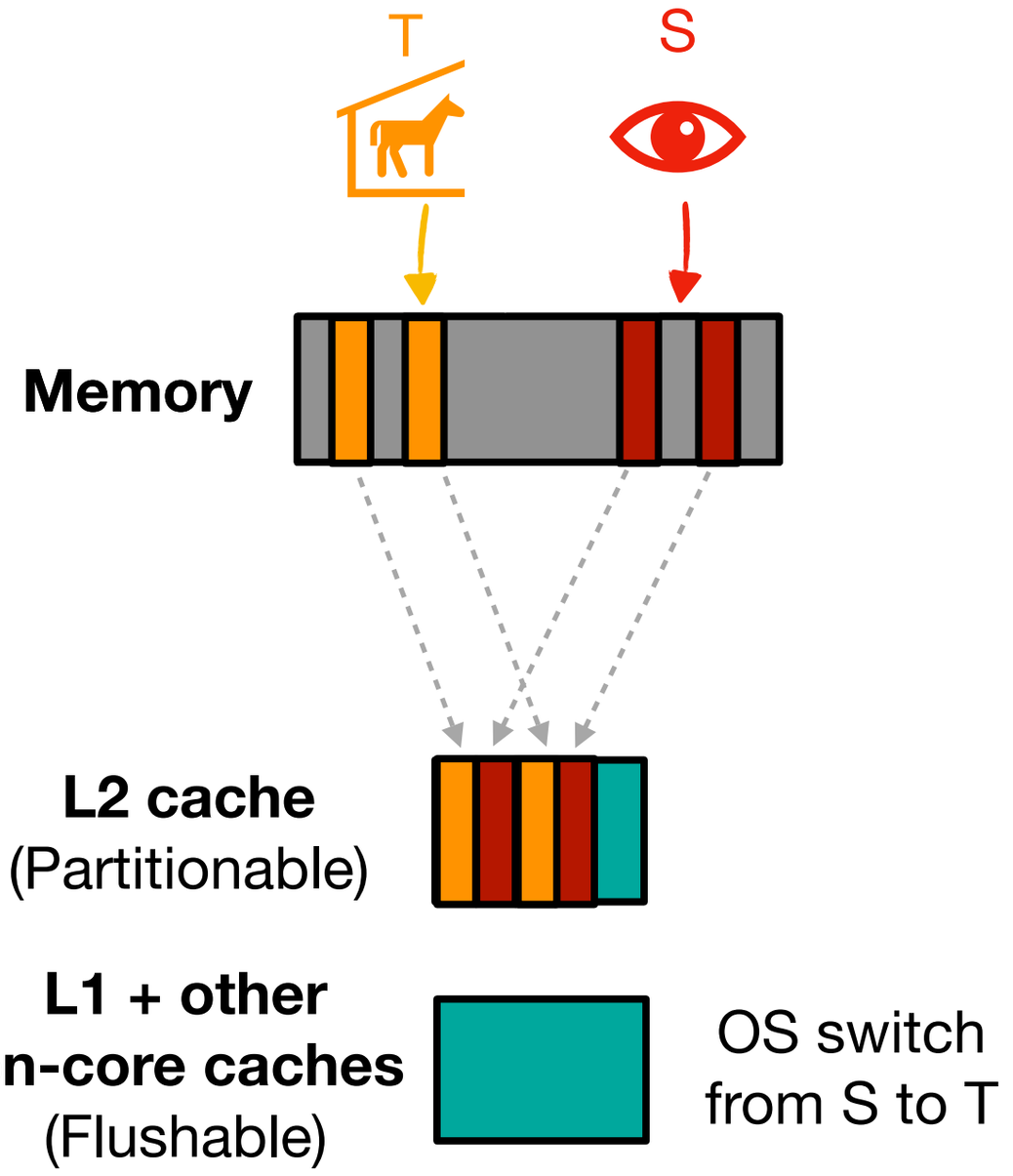
- OSes typically implement *memory protection.*

- But: Mere memory access can change the microarch. state — this affects timing.

- To prevent these *timing channels*, OSes can implement *time protection*:
  e.g. [Ge et al. 2019] for seL4 microkernel OS

  - Partition what we can

  - Flush what we can't




"Flush": Write fixed content; wait up to fixed time.

# Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

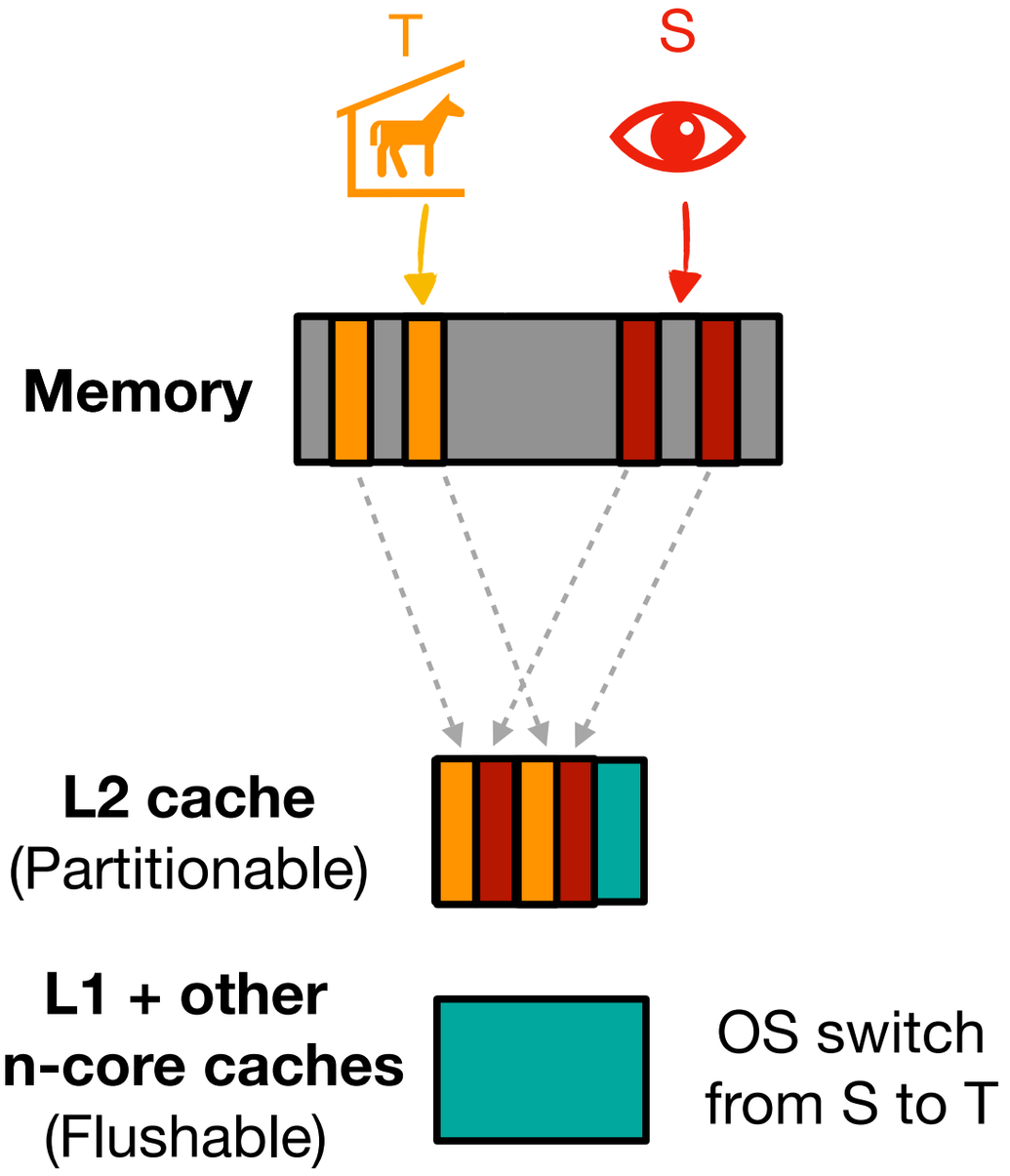
- OSes typically implement *memory protection.*

- But: Mere memory access can change the microarch. state — this affects timing.

- To prevent these *timing channels*, OSes can implement *time protection*:
  e.g. [Ge et al. 2019] for seL4 microkernel OS

  - Partition what we can
  - Flush what we can't

*HW-SW (hardware-software) contract*




"Flush": Write fixed content; wait up to fixed time.

# Threat scenario: *Trojan* and *spy*

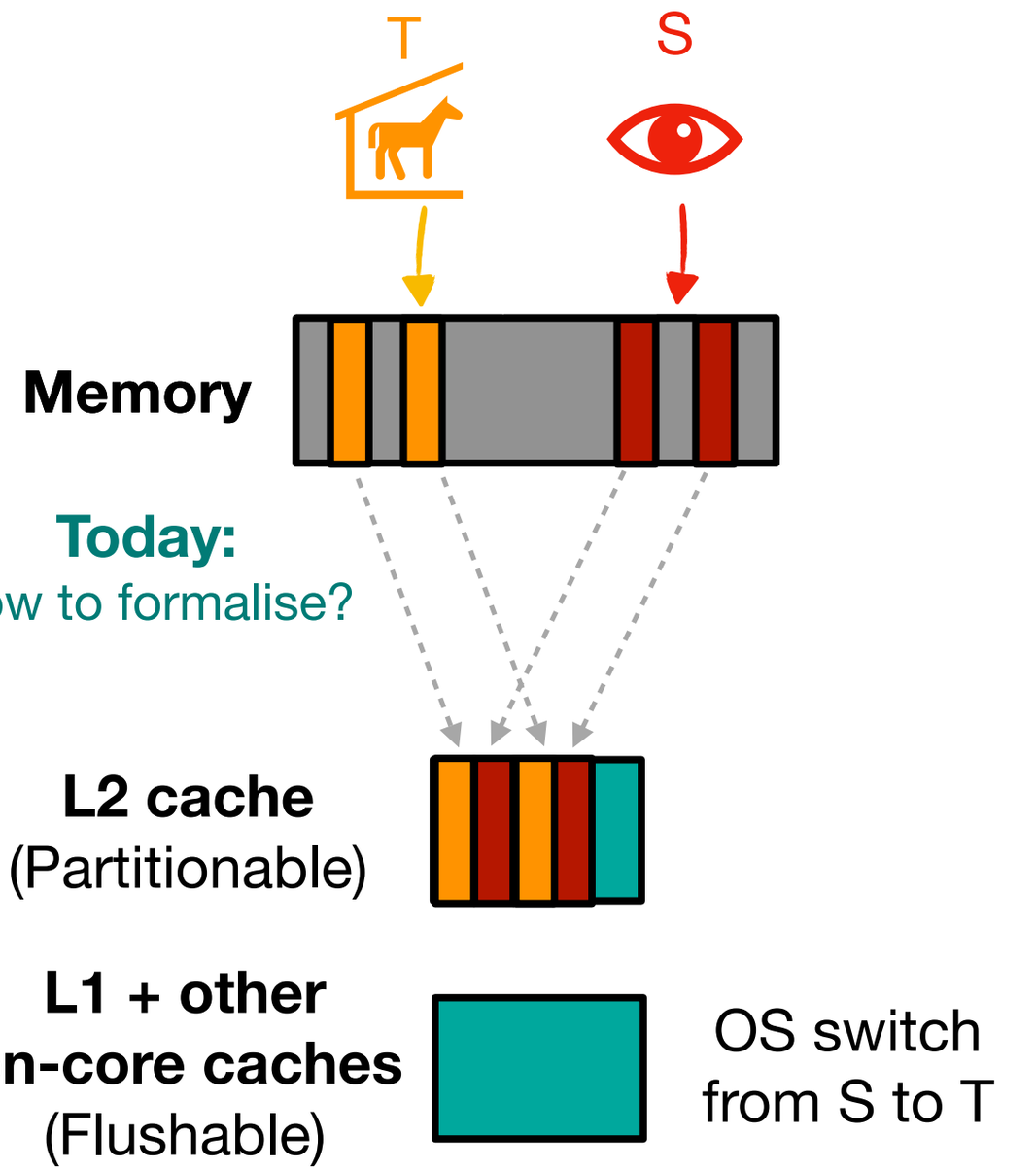
Covert channels
+
Side channels

- OSes typically implement *memory protection.*

- But: Mere memory access can change the microarch. state — this affects timing.

- To prevent these *timing channels*, OSes can implement *time protection*:
  e.g. [Ge et al. 2019] for seL4 microkernel OS
  - Partition what we can
  - Flush what we can't

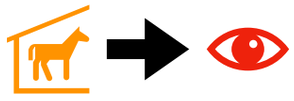
*HW-SW (hardware-software) contract*




"Flush": Write fixed content; wait up to fixed time.

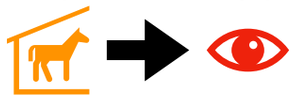
# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy



# How to formalise an OS enforces *time protection*?

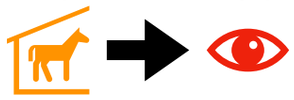
Versus threat scenario:
trojan and spy



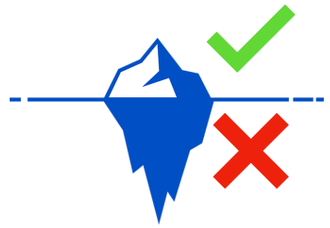

Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.

# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy



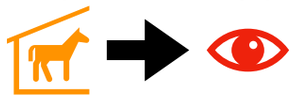

Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.



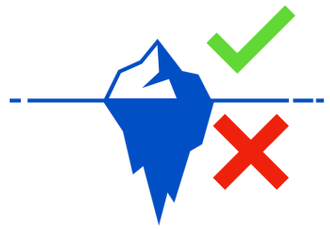
Make security property precise enough to exclude flows from covert state.

# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy




Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.

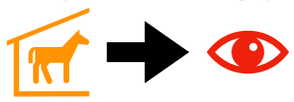


Make security property precise enough to exclude flows from covert state.

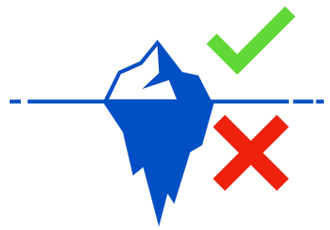

Demonstrating these principles,
we formalised in Isabelle/HOL:

# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy




Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.



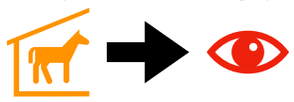
Make security property precise enough to exclude flows from covert state.


Demonstrating these principles,
we formalised in Isabelle/HOL:

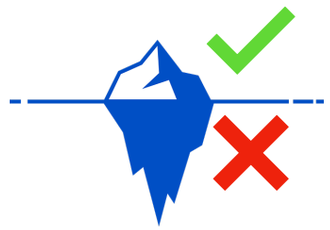
1. OS security model imposing requirements on relevant parts of OS.
(Intended for seL4, but *generic*)

# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy




Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.



Make security property precise enough to exclude flows from covert state.

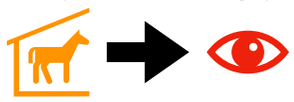

Demonstrating these principles,
we formalised in Isabelle/HOL:

1. OS security model imposing requirements on relevant parts of OS.
(Intended for seL4, but *generic*)

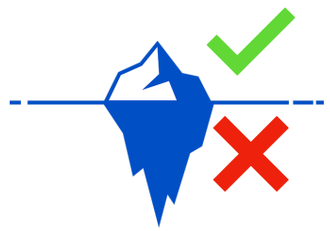
2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy




Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.



Make security property precise enough to exclude flows from covert state.


Demonstrating these principles,
we formalised in Isabelle/HOL:

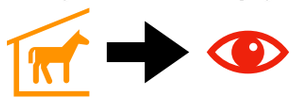
1. OS security model imposing requirements on relevant parts of OS.
(Intended for seL4, but *generic*)

2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

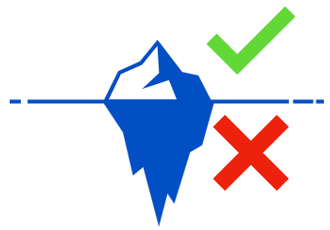
3. Proof our security property holds if OS model's requirements hold.

# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy




Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.



Make security property precise enough to exclude flows from covert state.


Demonstrating these principles,
we formalised in Isabelle/HOL:

1. OS security model imposing requirements on relevant parts of OS.
(Intended for seL4, but *generic*)

2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

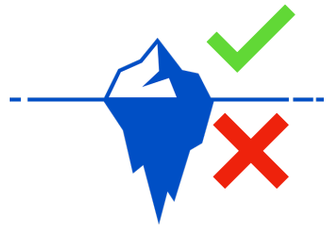
3. Proof our security property holds if OS model's requirements hold.

4. Basic instantiation of OS model exercising dynamic policy.

# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy


Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.



Make security property precise enough to exclude flows from covert state.


Demonstrating these principles,
we formalised in Isabelle/HOL:

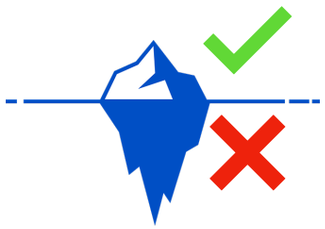
1. OS security model imposing requirements on relevant parts of OS.
(Intended for seL4, but *generic*)

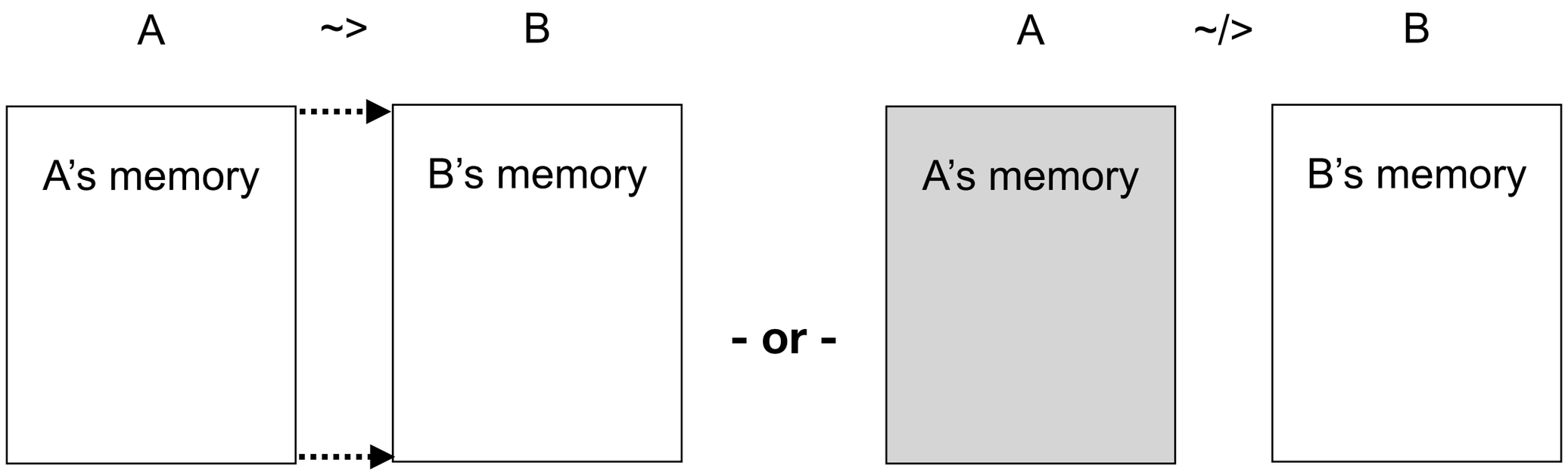
2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

3. Proof our security property holds if OS model's requirements hold.

4. Basic instantiation of OS model exercising dynamic policy.

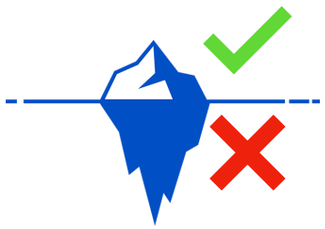
# *Overt* vs *covert* state

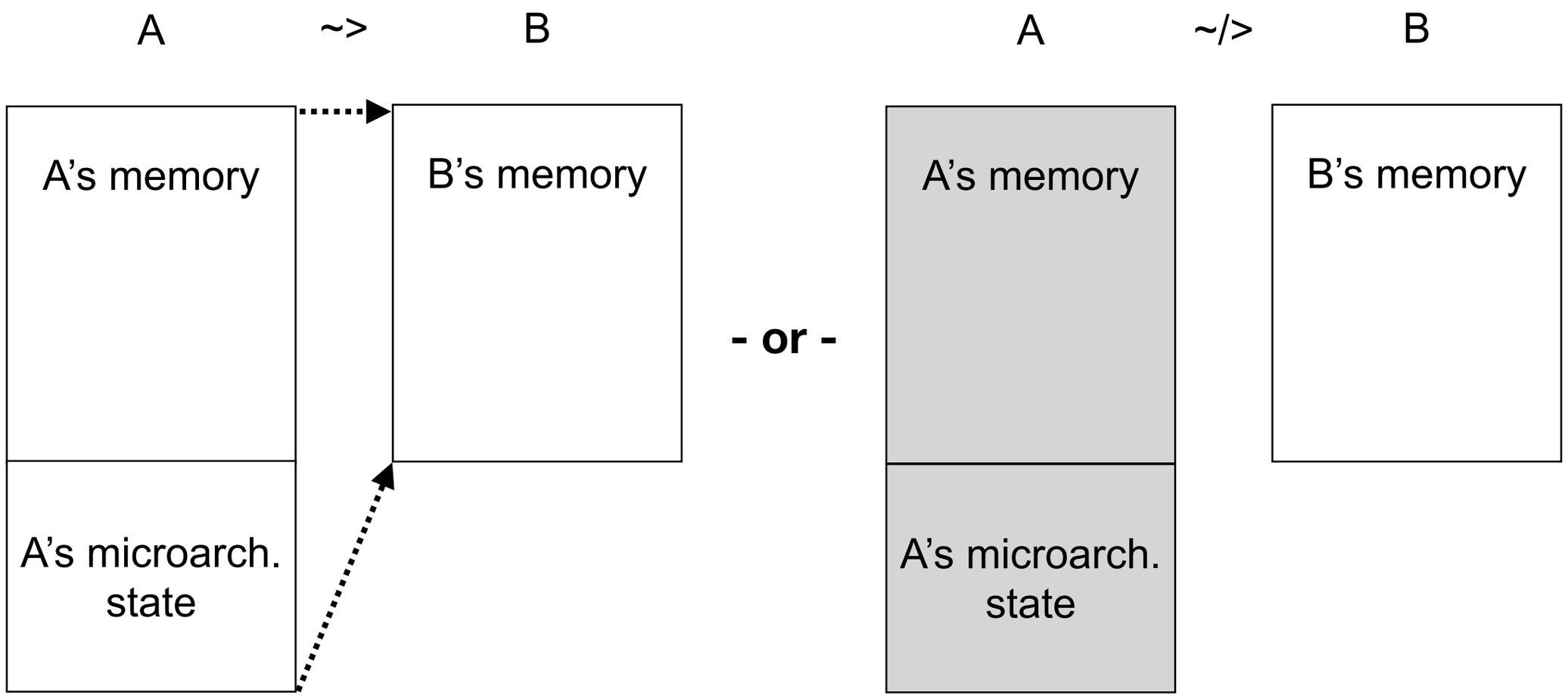





From prior seL4 infoflow proofs
[Murray et al. 2012, 2013]:
*"all or nothing" policies*

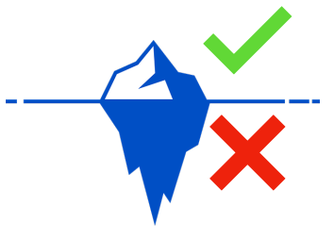
# *Overt* vs *covert* state

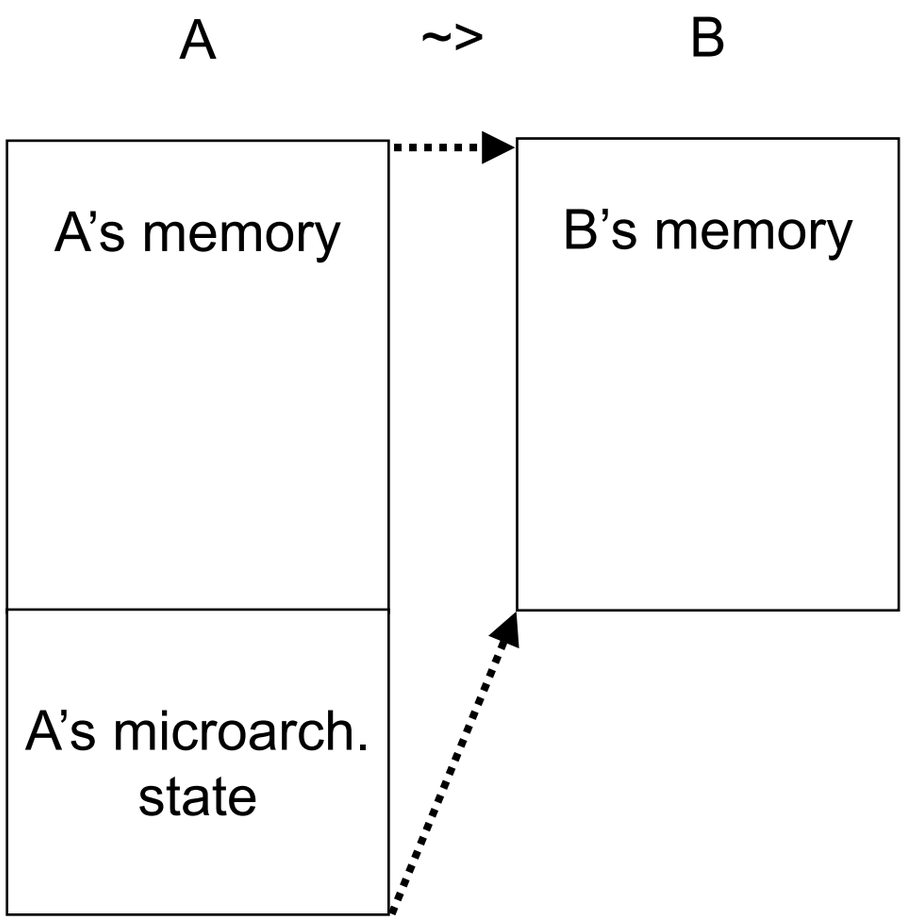




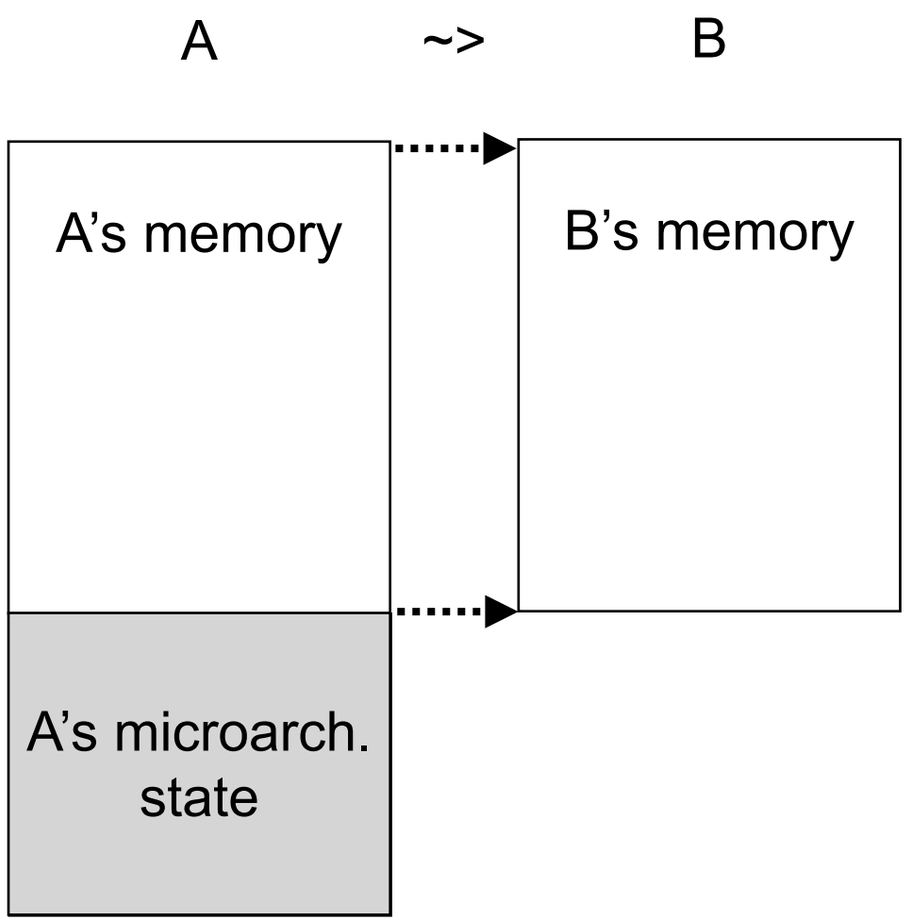

From prior seL4 infoflow proofs [Murray et al. 2012, 2013]:
*“all or nothing” policies*

# *Overt* vs *covert* state





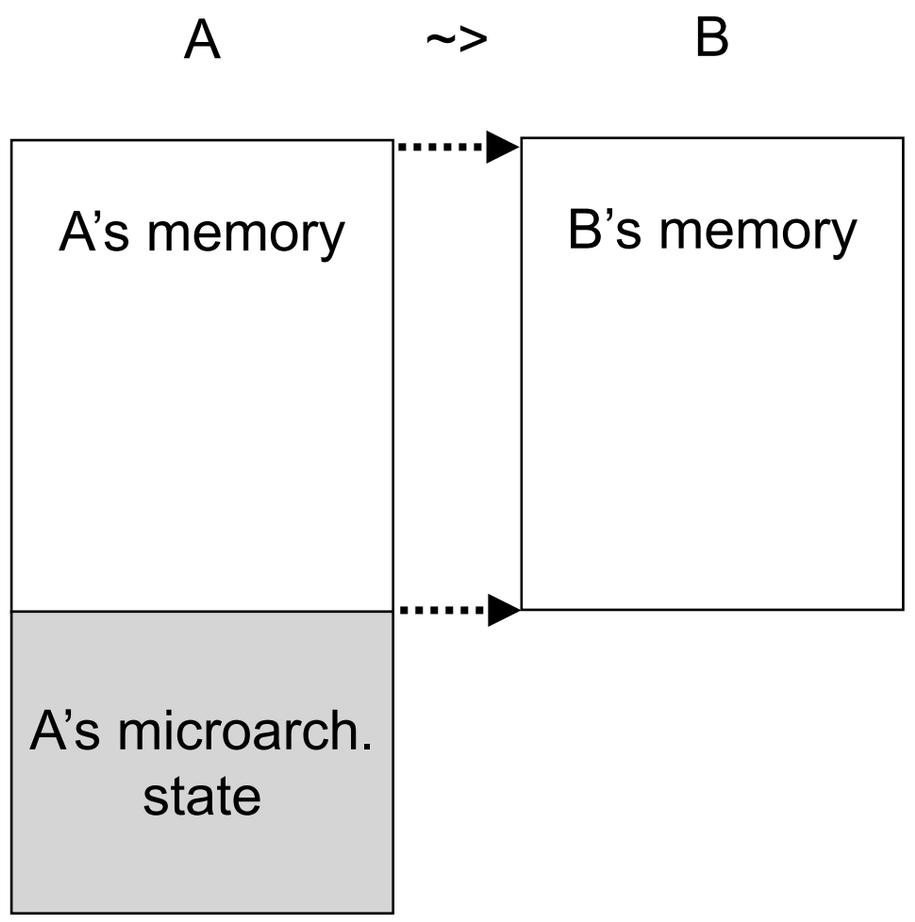

From prior seL4 infoflow proofs
[Murray et al. 2012, 2013]:
*"all or nothing" policies*




**Principle: Need policies to allow some (*overt*) flows while excluding other (*covert*) ones**

# Covert state: Partitionable vs flushable

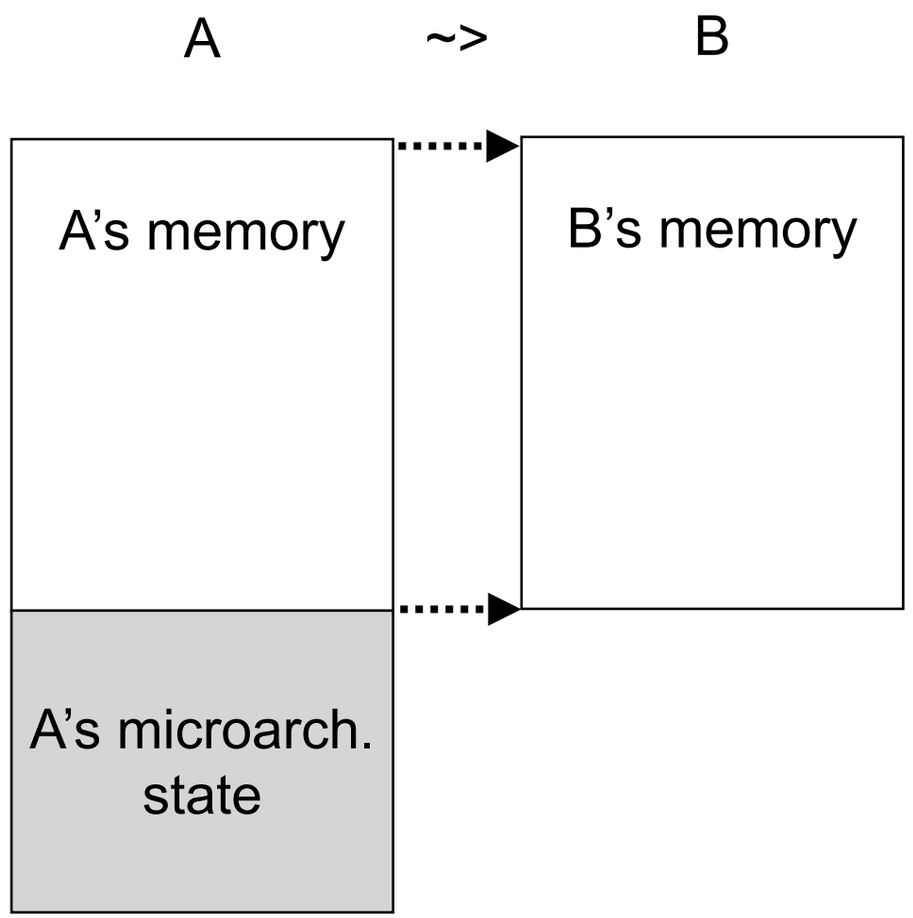

**<u>Principle</u>:**
**Model channels as *state elements***
**by their *elimination strategy***
**as per *HW-SW contract***



# Covert state: Partitionable vs flushable


**<u>Principle</u>:**
**Model channels as *state elements***
**by their *elimination strategy***
**as per *HW-SW contract***

- Strategy for OS:
  *Partition* or *flush* state; *pad* time.

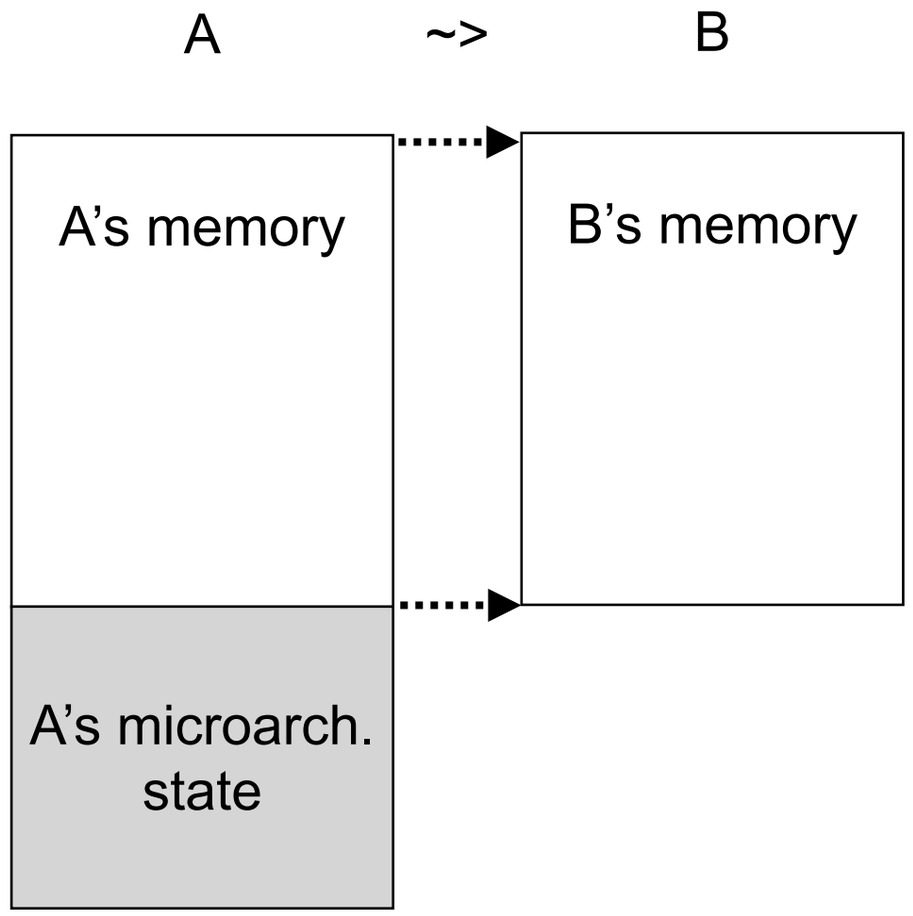


# Covert state: Partitionable vs flushable


**<u>Principle</u>:**
**Model channels as *state elements***
**by their *elimination strategy***
**as per *HW-SW contract***

- Strategy for OS:
  *Partition* or *flush* state; *pad* time.

- Relies on HW-SW contract:

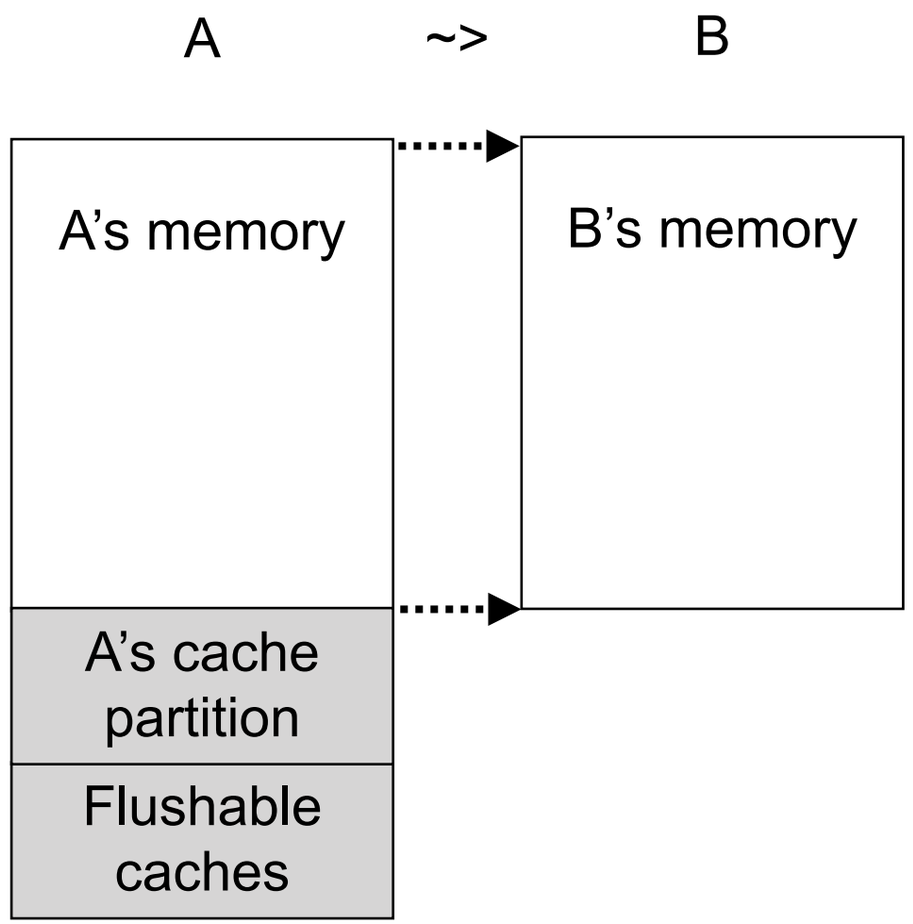


# Covert state: Partitionable vs flushable


**<u>Principle</u>:**
**Model channels as *state elements***
**by their *elimination strategy***
**as per *HW-SW contract***

- Strategy for OS:
  *Partition* or *flush* state; *pad* time.

- Relies on HW-SW contract:
  - <u>State</u>: Everything must be *partitionable* or *flushable*.

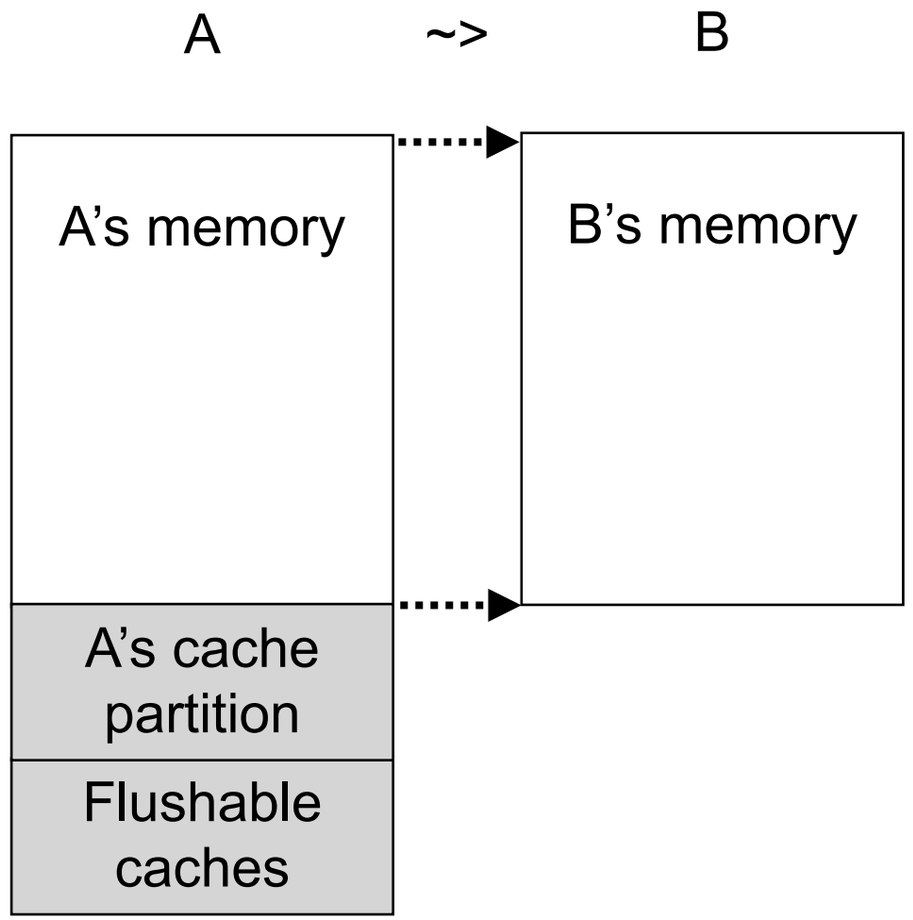


# Covert state: Partitionable vs flushable


**<u>Principle</u>:**
**Model channels as *state elements* by their *elimination strategy* as per *HW-SW contract***

- Strategy for OS:
  *Partition* or *flush* state; *pad* time.

- Relies on HW-SW contract:
  - <u>State</u>: Everything must be *partitionable* or *flushable*.
    - e.g. Off-core vs on-core caches.

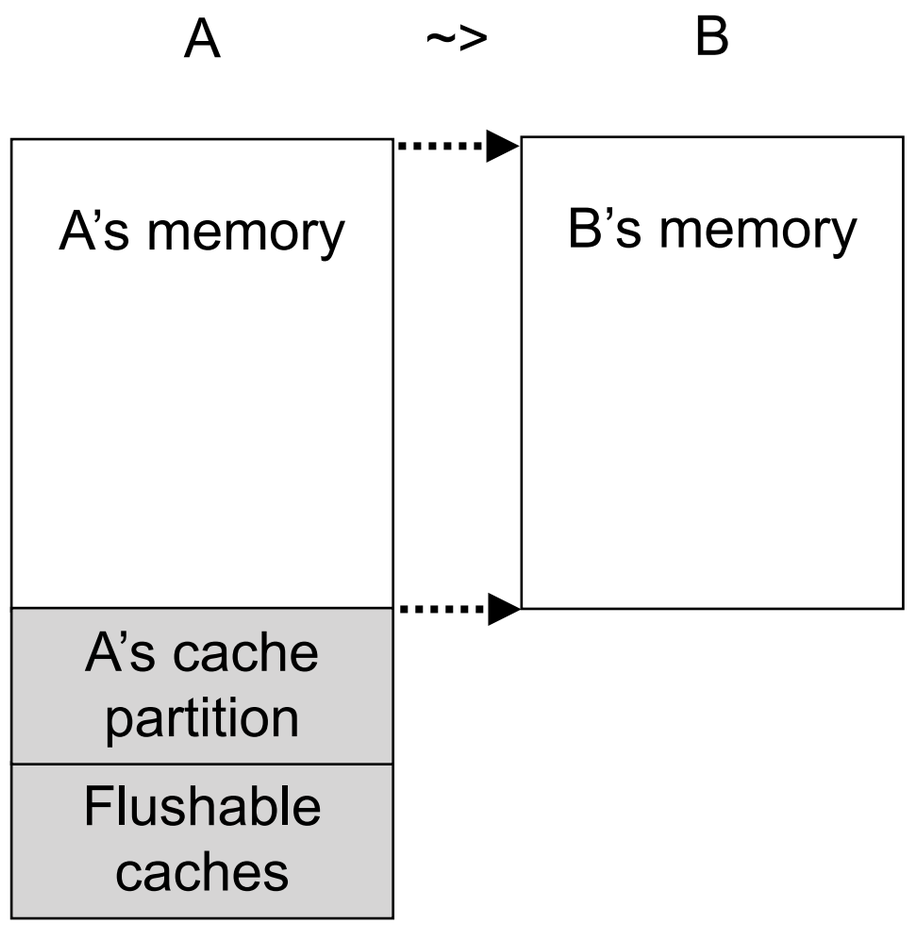


# Covert state: Partitionable vs flushable


**<u>Principle</u>:**
**Model channels as *state elements* by their *elimination strategy* as per *HW-SW contract***

- Strategy for OS:
  *Partition* or *flush* state; *pad* time.

- Relies on HW-SW contract:
  - <u>State</u>: Everything must be *partitionable* or *flushable*.
    - e.g. Off-core vs on-core caches.
    - Interrupt-generating devices (partitionable; not pictured).

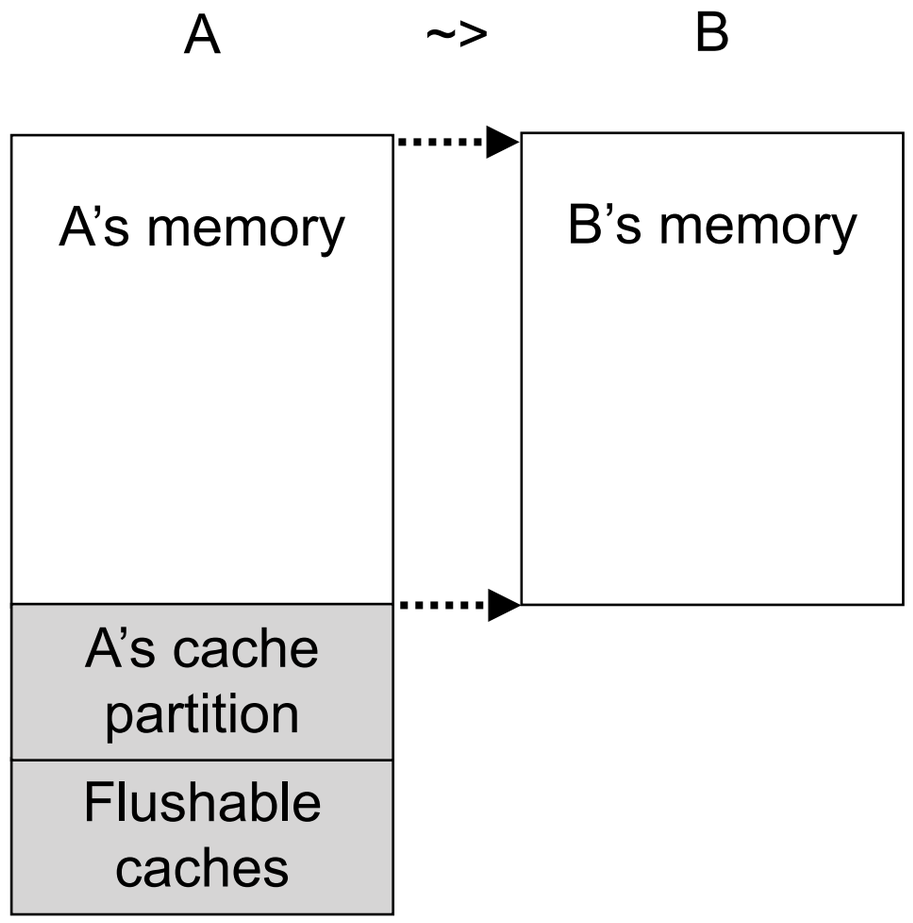


# Covert state: Partitionable vs flushable


**<u>Principle</u>:**
**Model channels as *state elements* by their *elimination strategy* as per *HW-SW contract***

- Strategy for OS:
  *Partition* or *flush* state; *pad* time.

- Relies on HW-SW contract:
  - <u>State</u>: Everything must be *partitionable* or *flushable*.
    - e.g. Off-core vs on-core caches.
    - Interrupt-generating devices (partitionable; not pictured).
  - <u>Time</u>: HW must give reliable
    - WCETs (worst-case execution times)

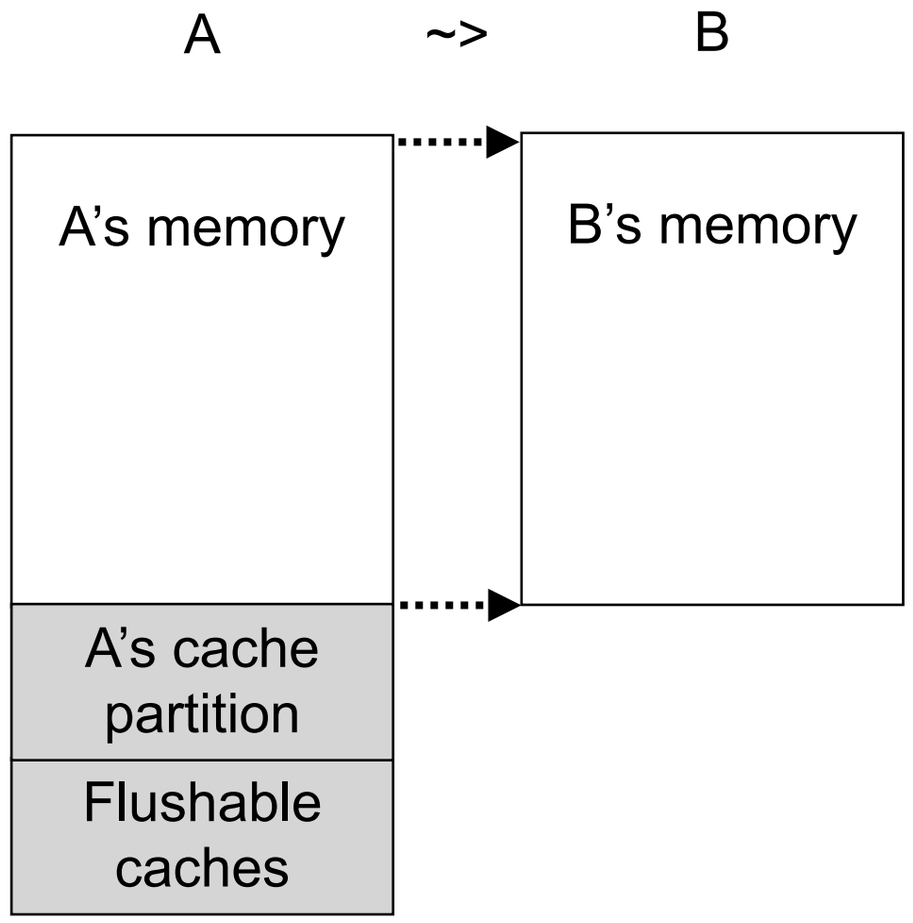


# Covert state: Partitionable vs flushable


**<u>Principle</u>:**
**Model channels as *state elements* by their *elimination strategy* as per *HW-SW contract***

- Strategy for OS:
  *Partition* or *flush* state; *pad* time.

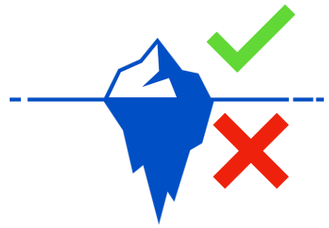
- Relies on HW-SW contract:
  - <u>State</u>: Everything must be *partitionable* or *flushable*.
    - e.g. Off-core vs on-core caches.
    - Interrupt-generating devices (partitionable; not pictured).
  - <u>Time</u>: HW must give reliable
    - WCETs (worst-case execution times)
    - method of *padding*.



# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy


Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.



Make security property precise enough to exclude flows from covert state.


Demonstrating these principles,
we formalised in Isabelle/HOL:

1. OS security model imposing requirements on relevant parts of OS.
(Intended for seL4, but *generic*)

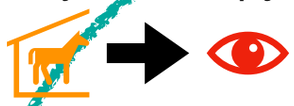
2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

3. Proof our security property holds if OS model's requirements hold.

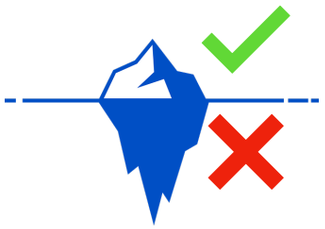
4. Basic instantiation of OS model exercising dynamic policy.

# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy




Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.



Make security property precise enough to exclude flows from covert state.


Demonstrating these principles,
we formalised in Isabelle/HOL:

1. OS security model imposing requirements on relevant parts of OS.
(Intended for seL4, but *generic*)

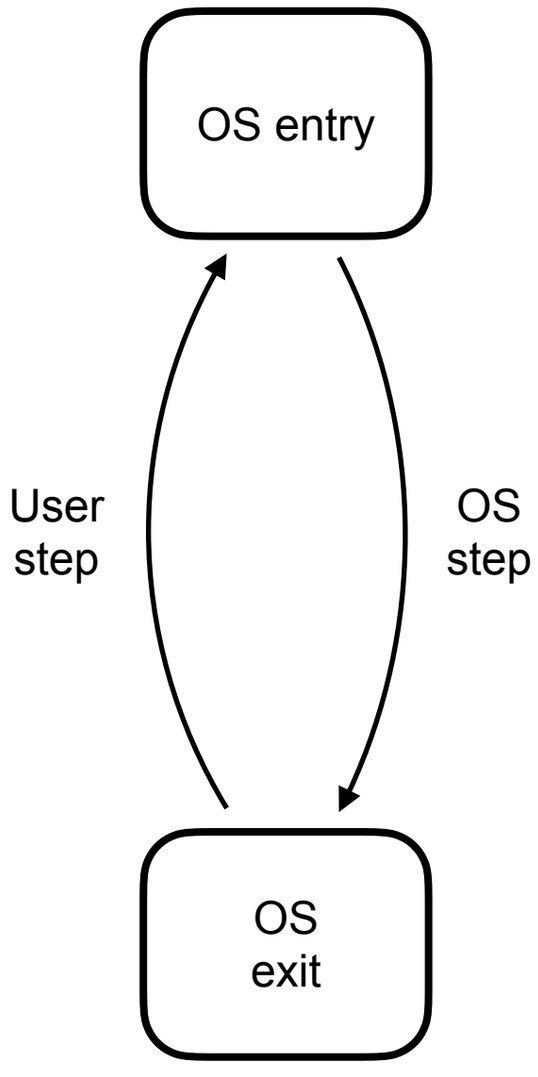
2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

3. Proof our security property holds if OS model's requirements hold.

4. Basic instantiation of OS model exercising dynamic policy.

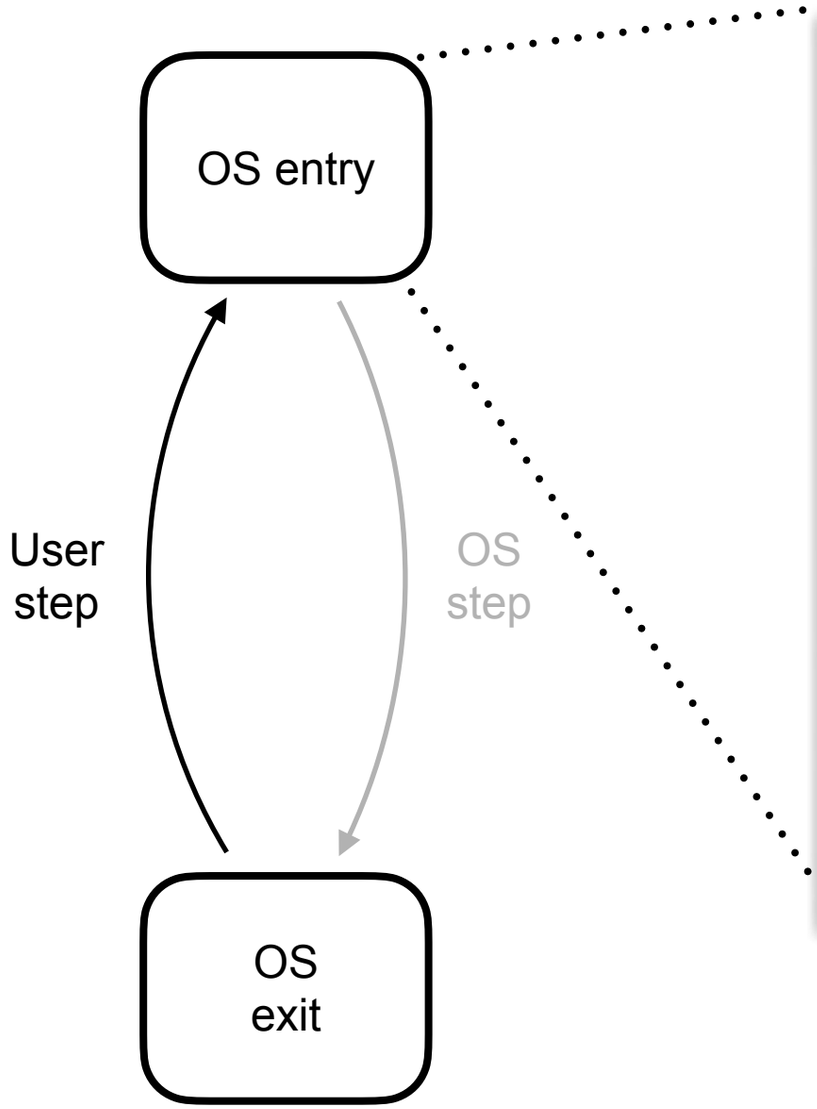
# OS security model


## Transition system



# OS security model


## Transition system




### State fields

mem :: *addr* ⇒ *int*

flst :: *addr* ⇒ bool /* Flushable microarch. */

pst :: *addr* ⇒ bool /* Partitionable microarch. */

tm :: *nat* /* Time */

dom :: *domain* /* Current domain */

devs :: *device* set /* Interrupt-generating devices */

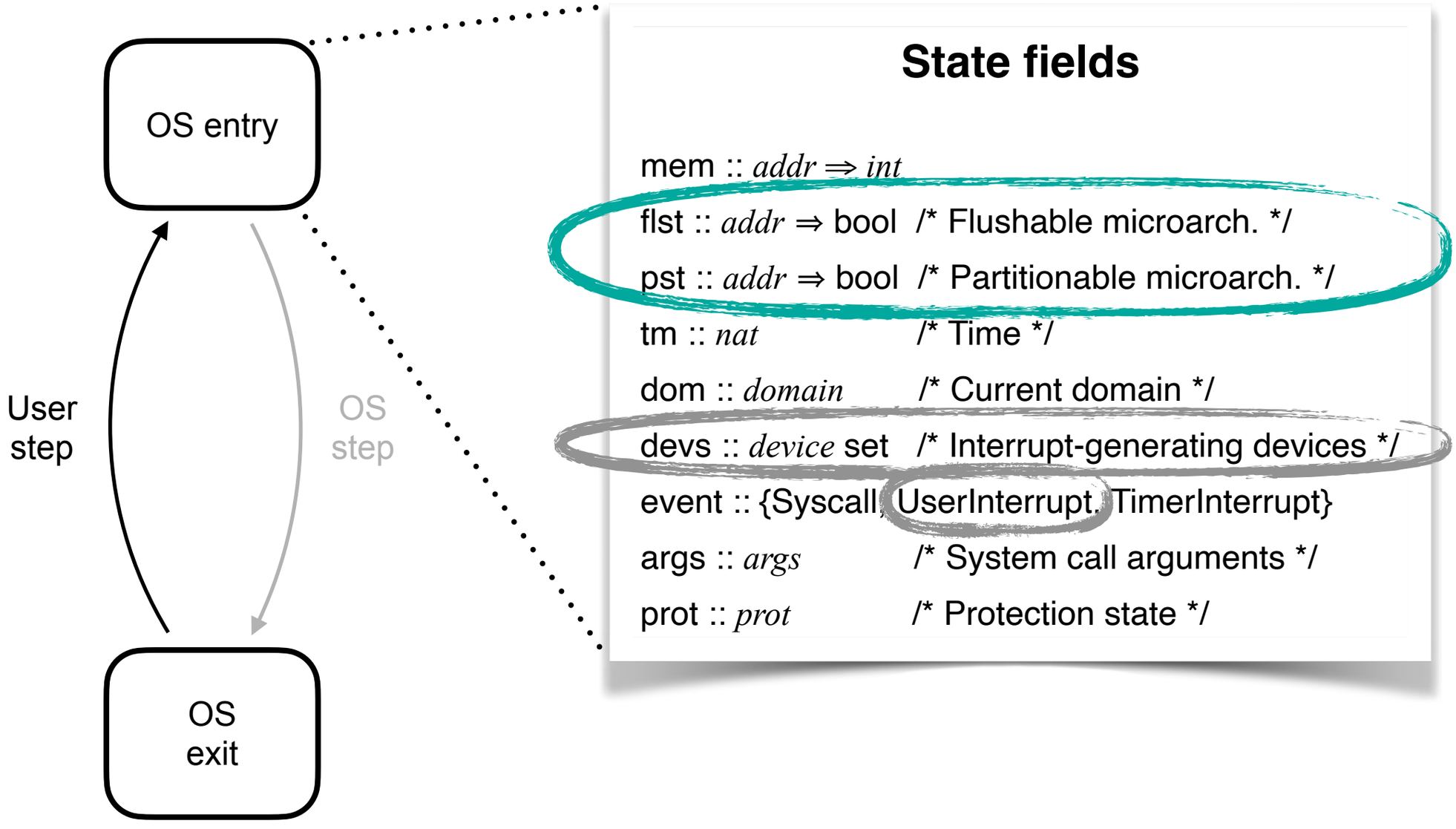
event :: {Syscall, UserInterrupt, TimerInterrupt}

args :: *args* /* System call arguments */

prot :: *prot* /* Protection state */

# OS security model

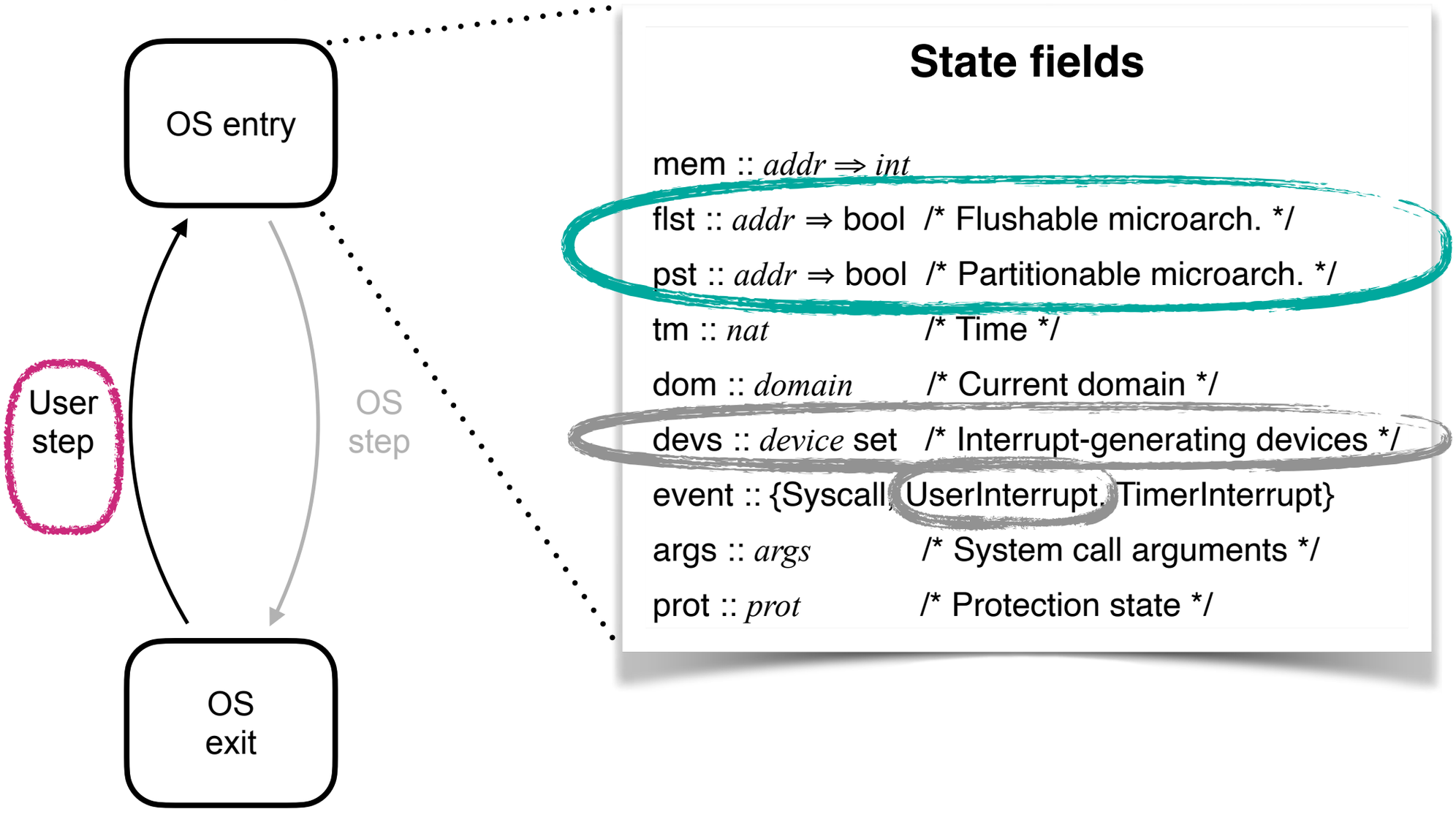

## Transition system




Microarchitecture
Devices

# OS security model

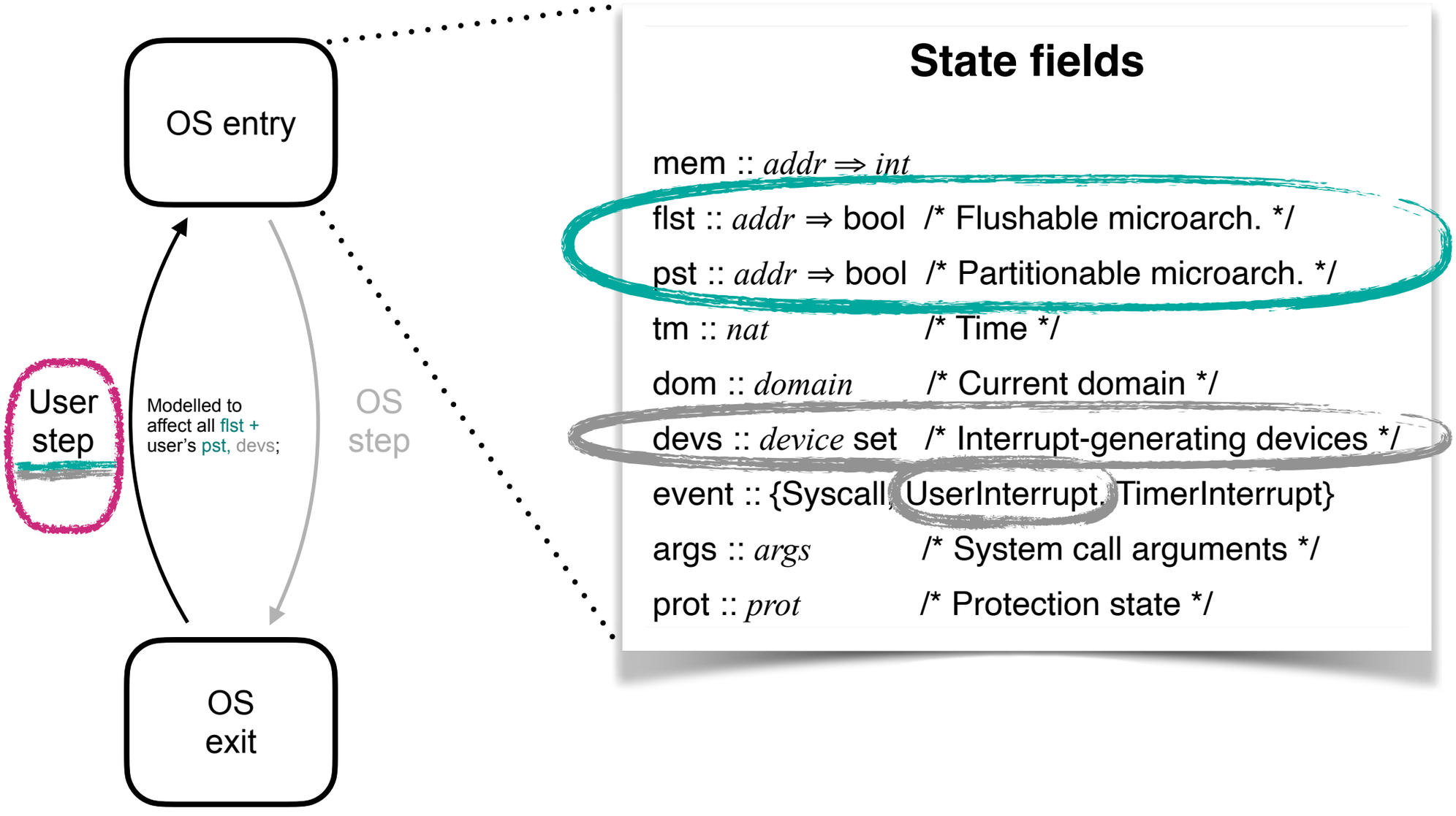

## Transition system




Microarchitecture
Devices

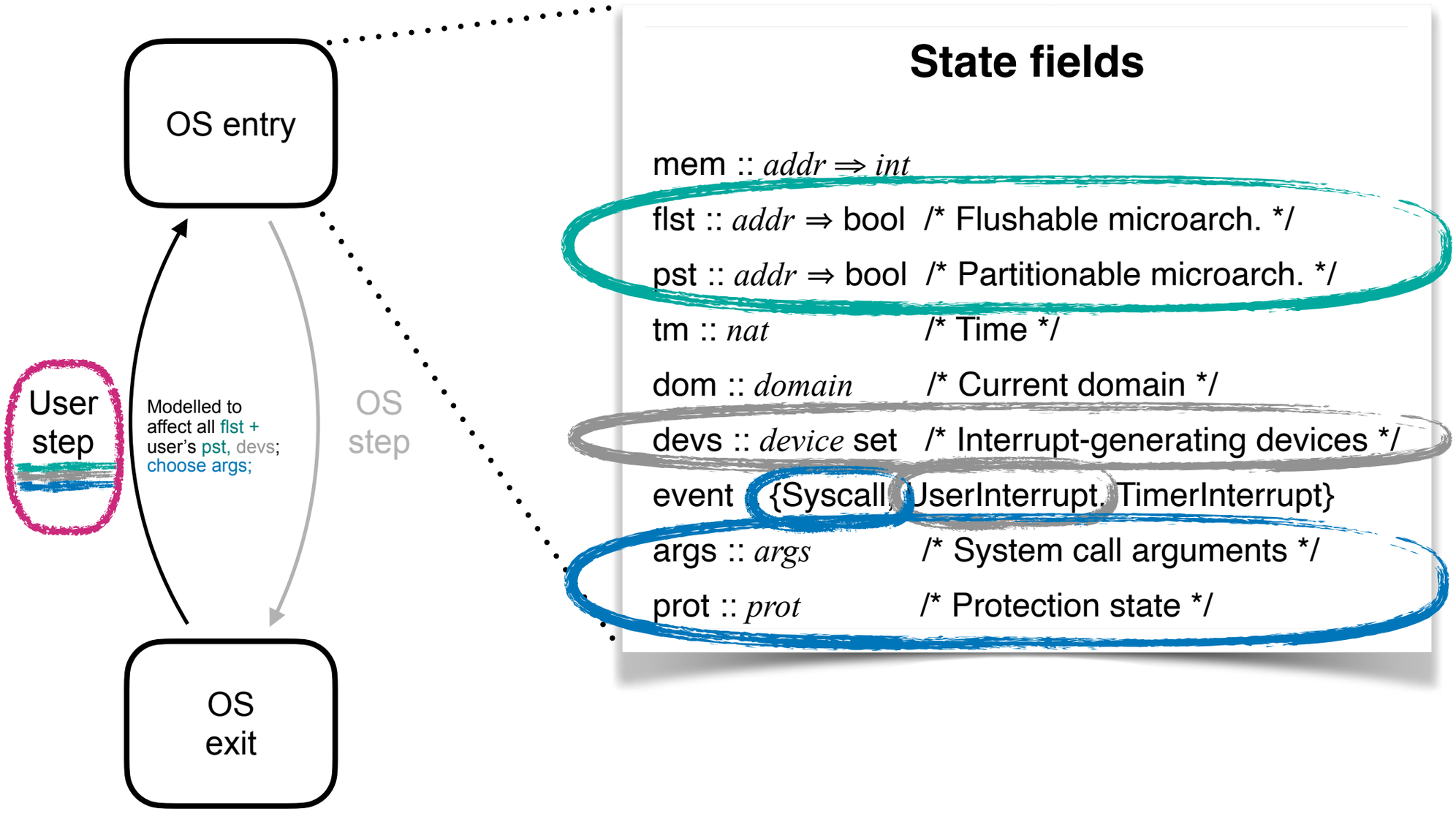
# OS security model


## Transition system



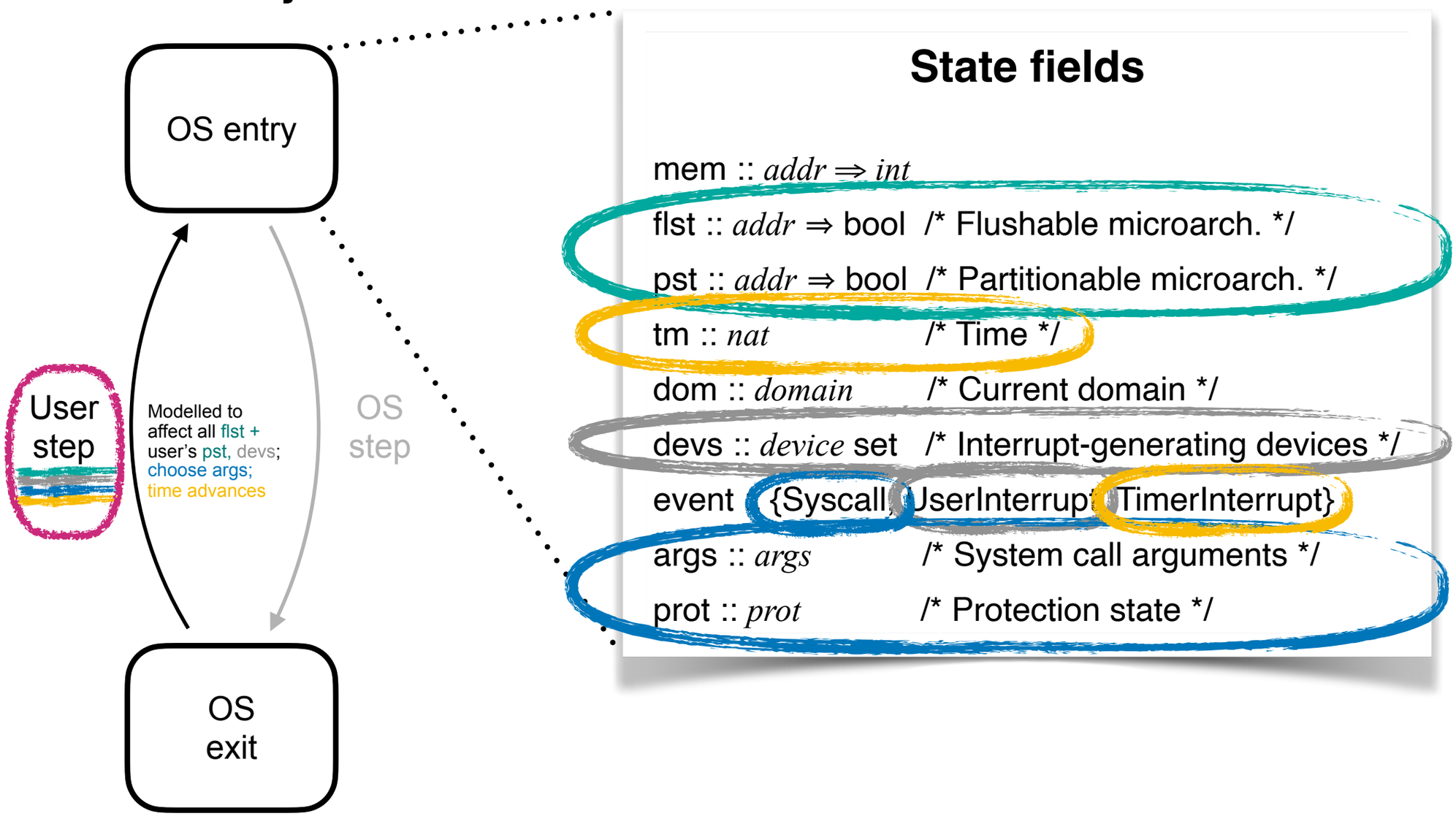
# OS security model


## Transition system



# OS security model

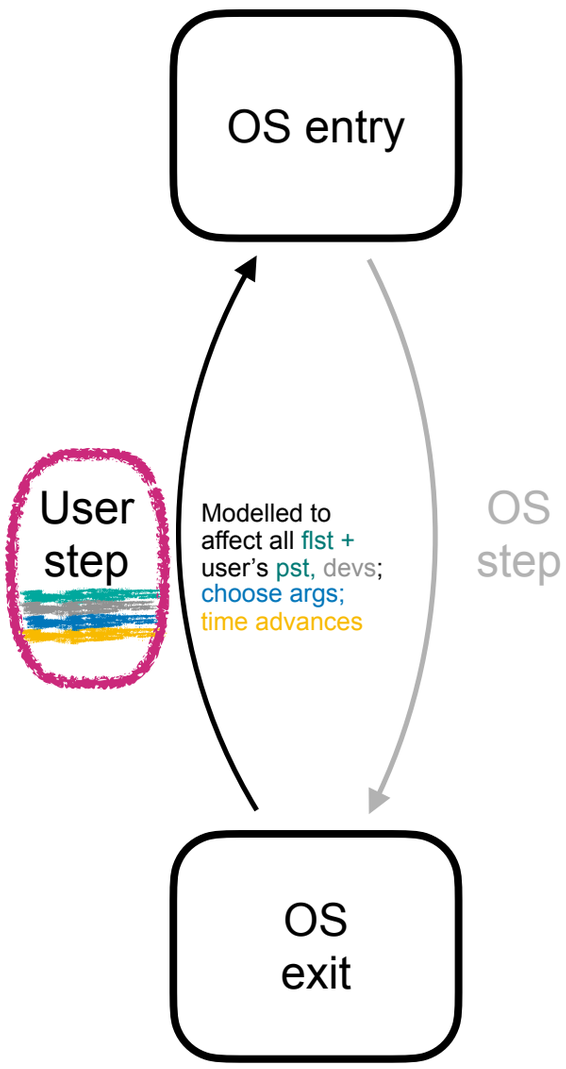

## Transition system



# OS security model


Microarchitecture
Devices
Policy-determining state
Time

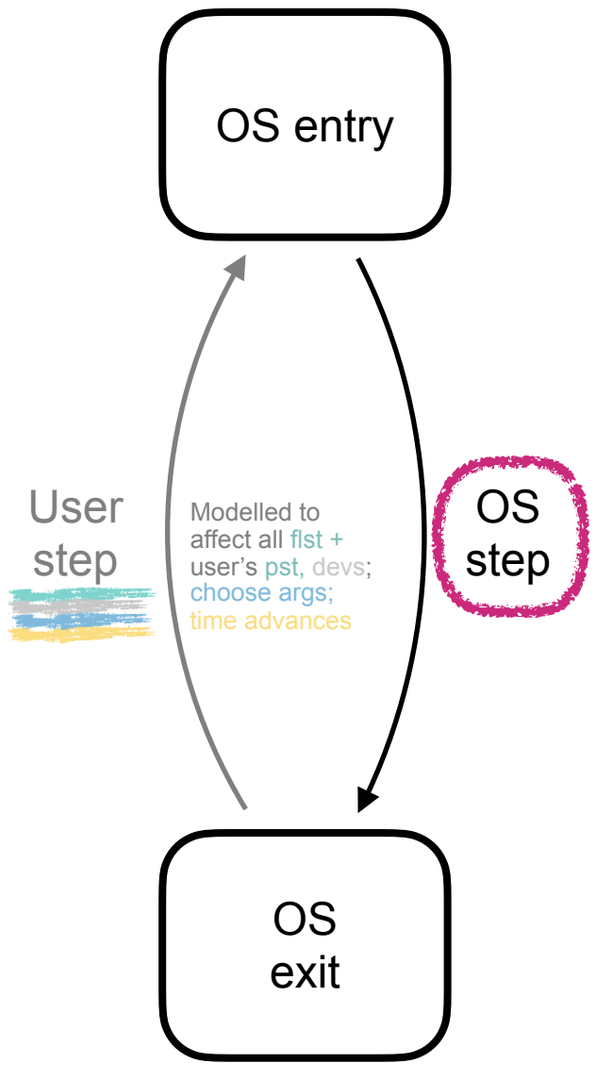
## Transition system



# OS security model


Microarchitecture
Devices
Policy-determining state
Time

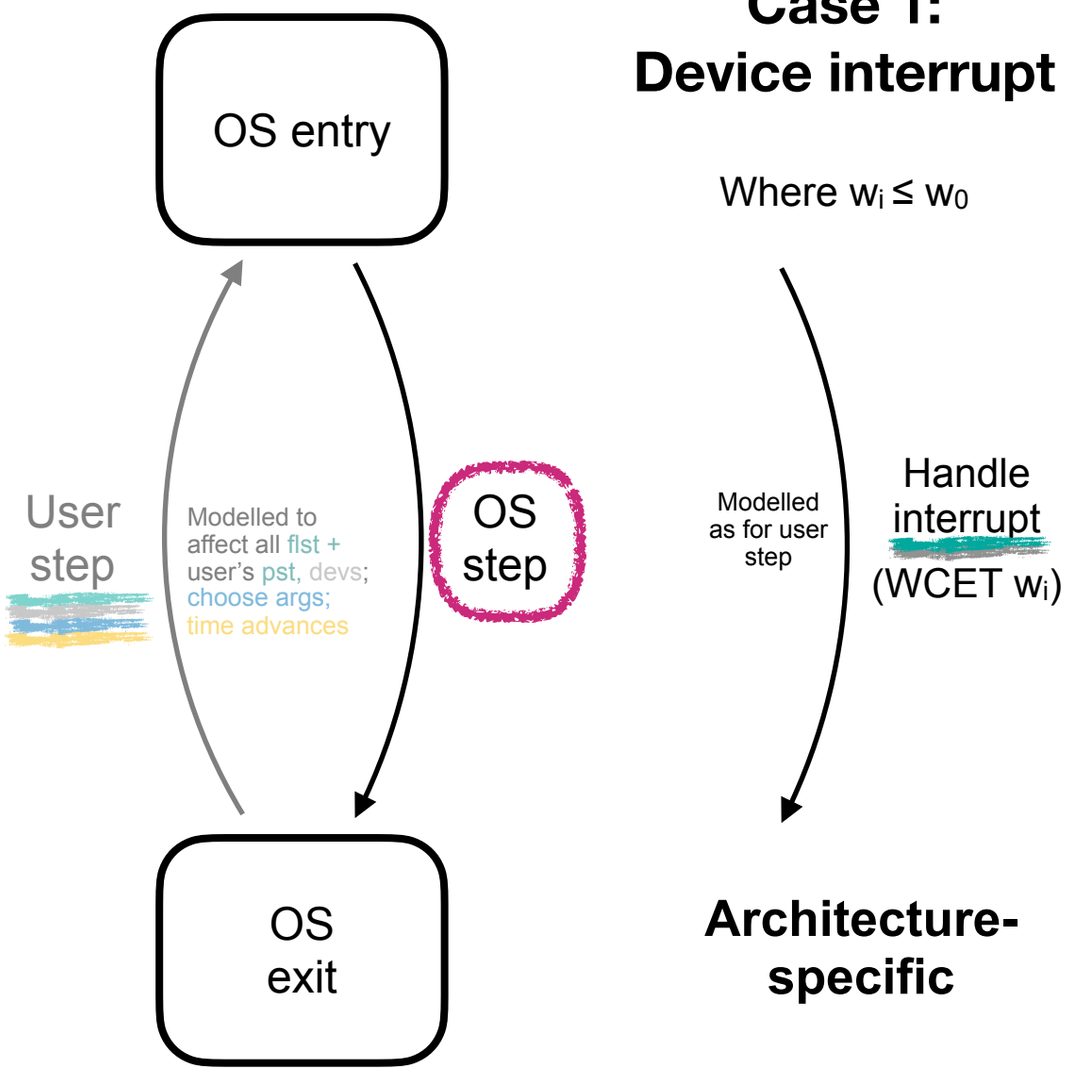
## Transition system



# OS security model


Microarchitecture
Devices
Policy-determining state
Time

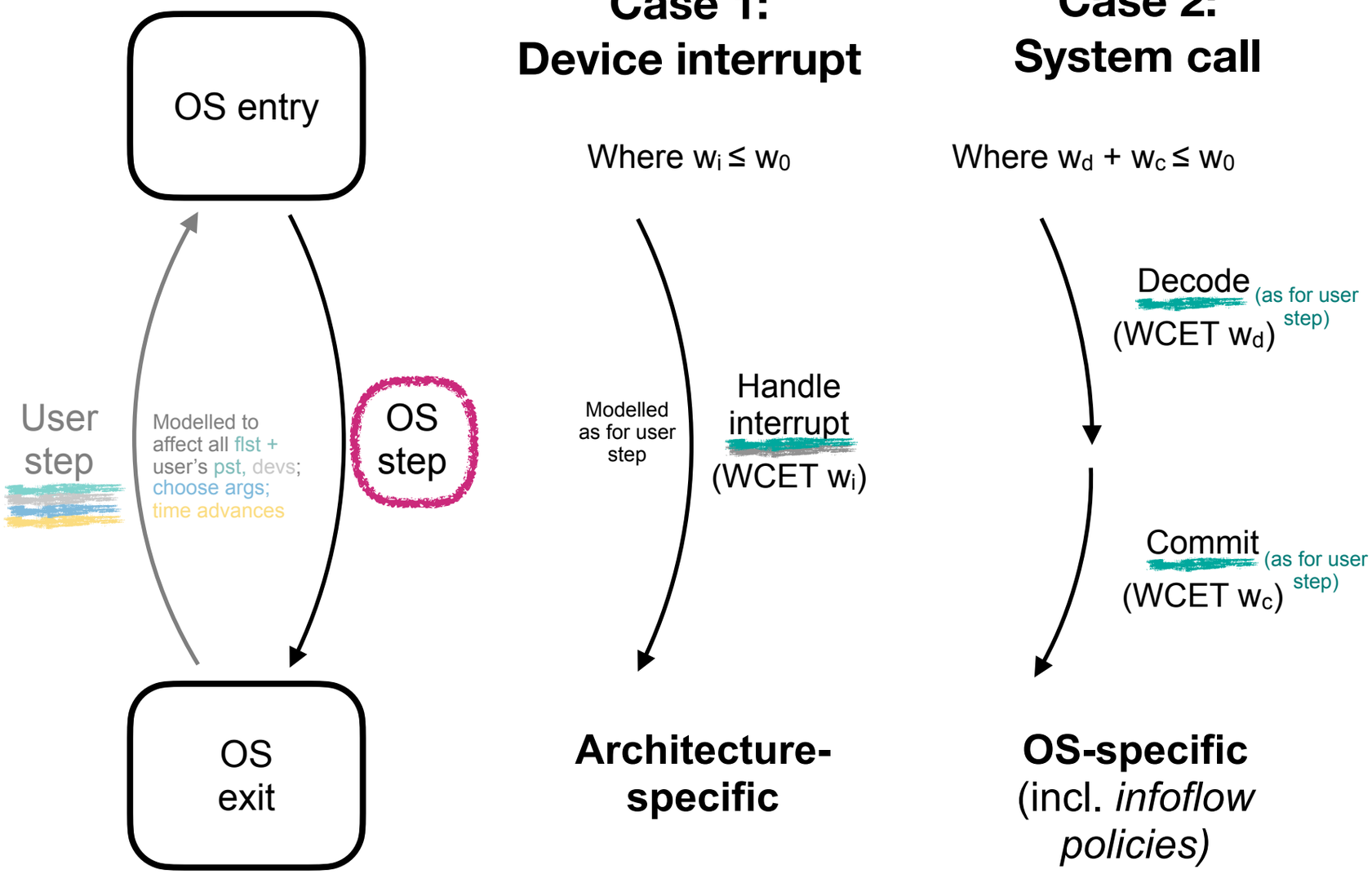
## Transition system



# OS security model


Microarchitecture
Devices
Policy-determining state
Time

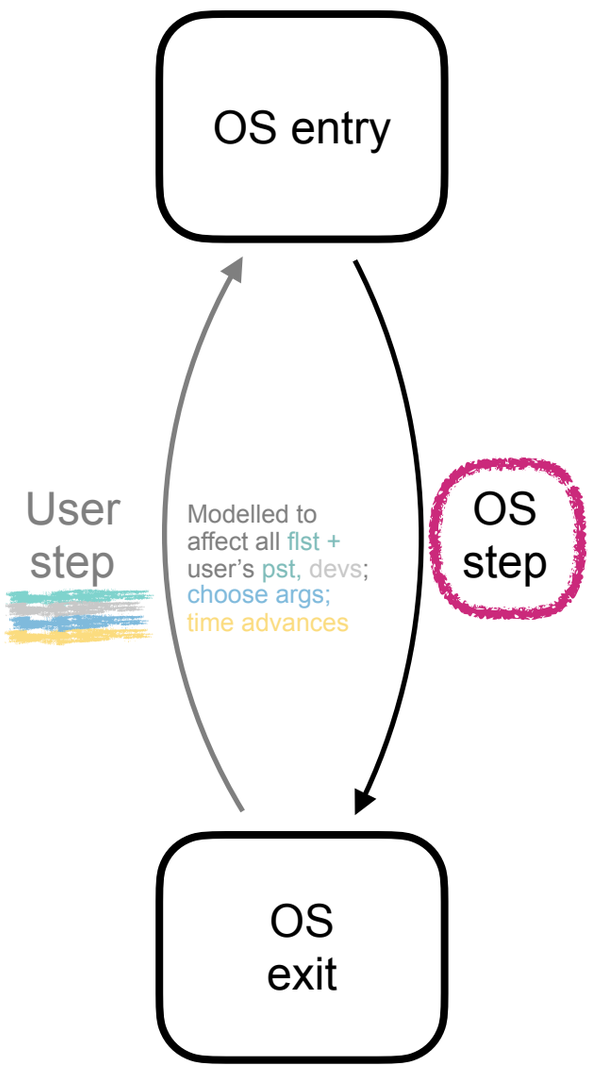
## Transition system



# OS security model

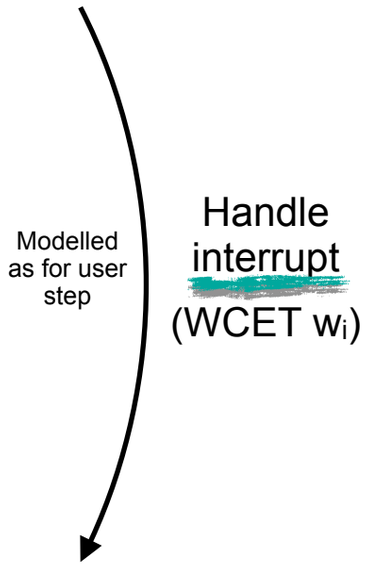

Microarchitecture
Devices
Policy-determining state
Time

## Transition system




## Case 1: Device interrupt

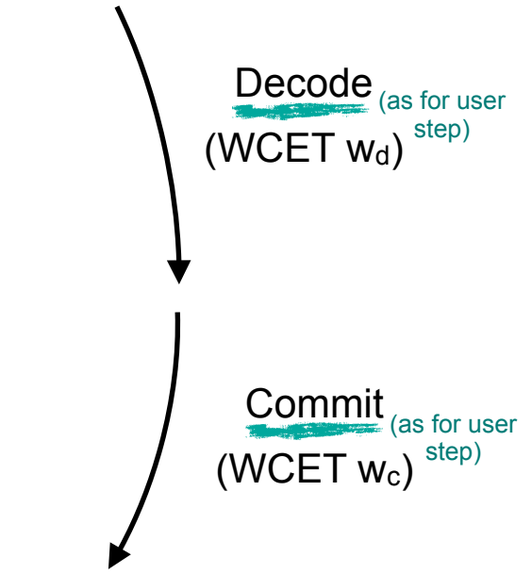
Where $w_i \leq w_0$




**Architecture-specific**

## Case 2: System call

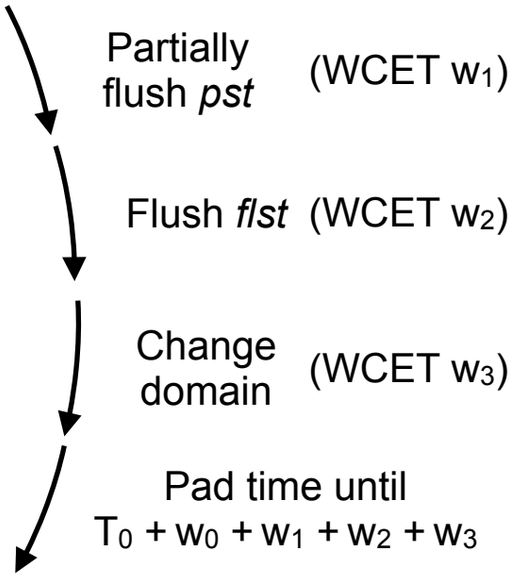
Where $w_d + w_c \leq w_0$




**OS-specific** (incl. *infoflow policies)*

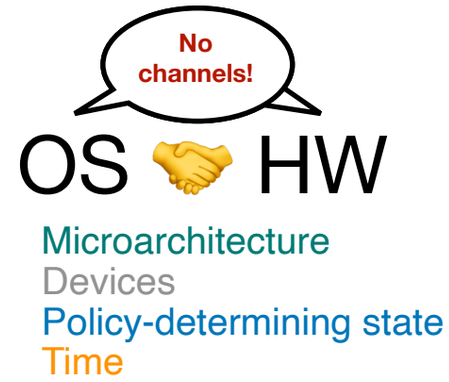
## Case 3: Domain switch

Timer interrupt delivered at (worst-case) $T_0 + w_0$

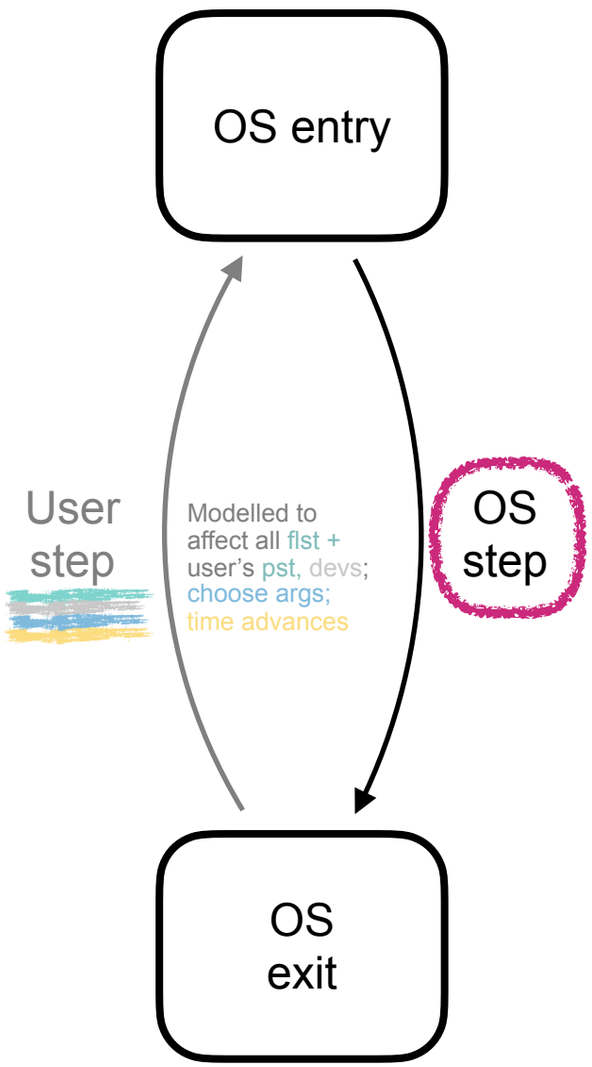



**Architecture-specific**

# OS security model

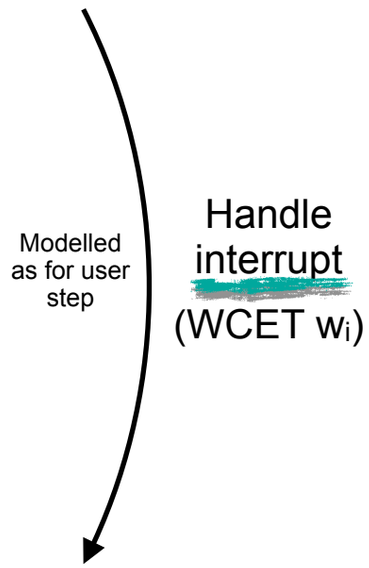



## Transition system




## Case 1: Device interrupt

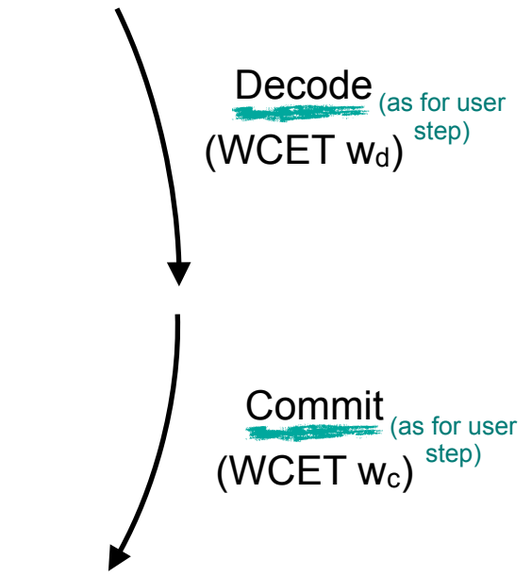
Where $w_i \leq w_0$




**Architecture-specific**

## Case 2: System call

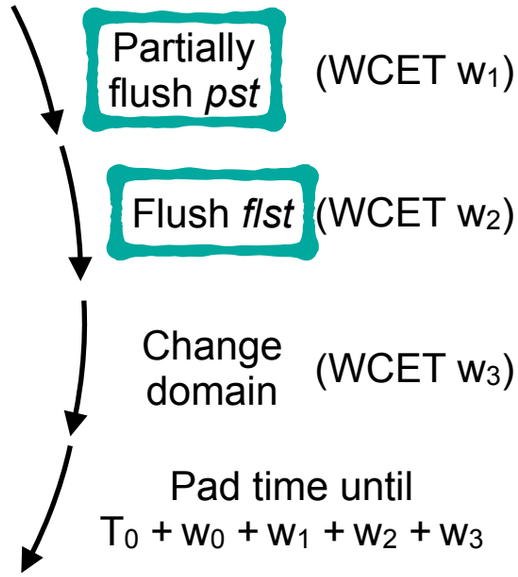
Where $w_d + w_c \leq w_0$




**OS-specific** (incl. *infoflow policies)*

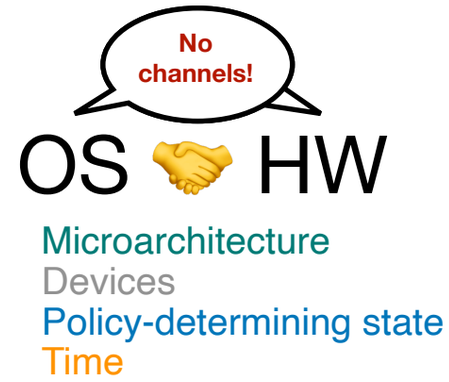
## Case 3: Domain switch

Timer interrupt delivered at (worst-case) $T_0 + w_0$

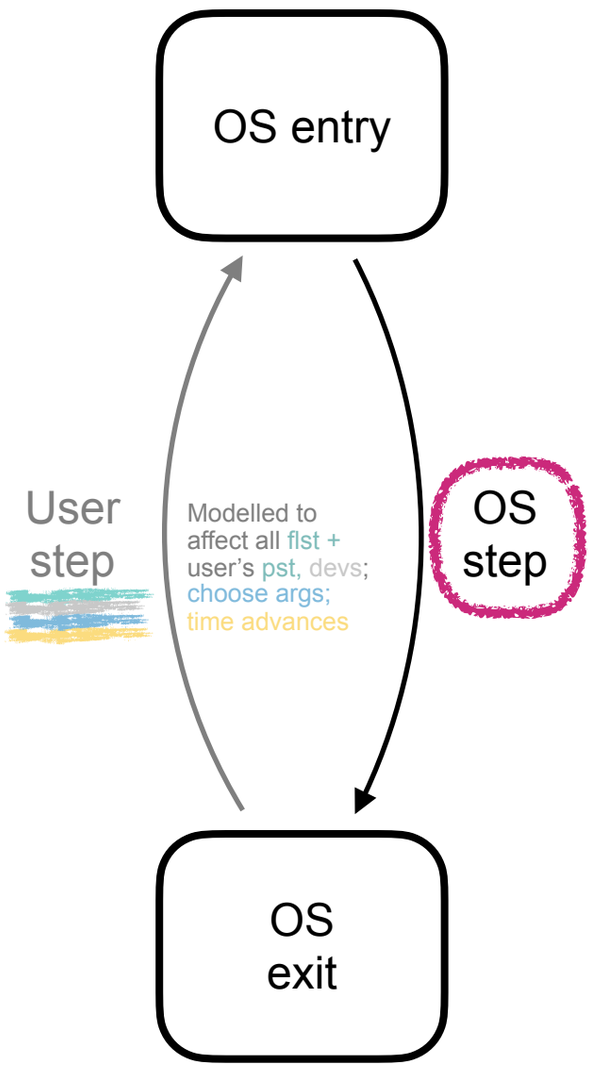



**Architecture-specific**

# OS security model

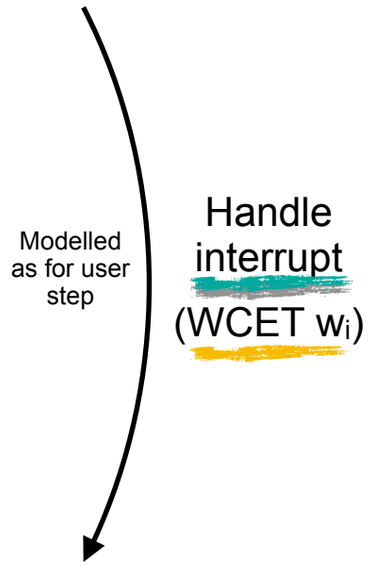



## Transition system




## Case 1: Device interrupt

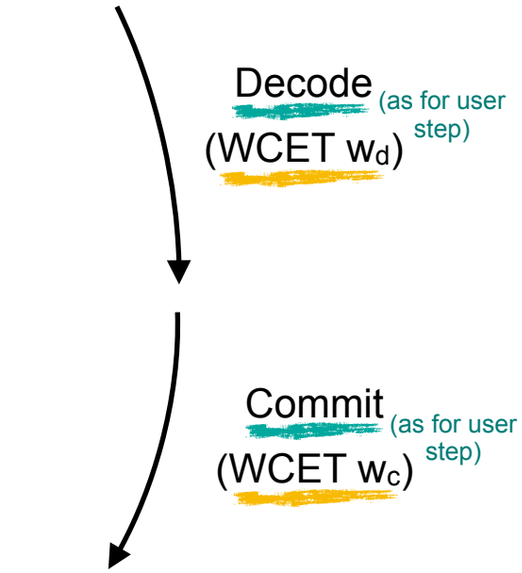
Where $w_i \leq w_0$




**Architecture-specific**

## Case 2: System call

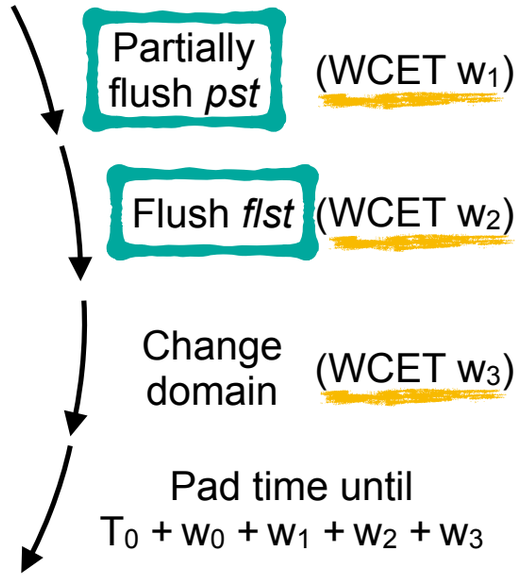
Where $w_d + w_c \leq w_0$




**OS-specific** (incl. *infoflow policies)*

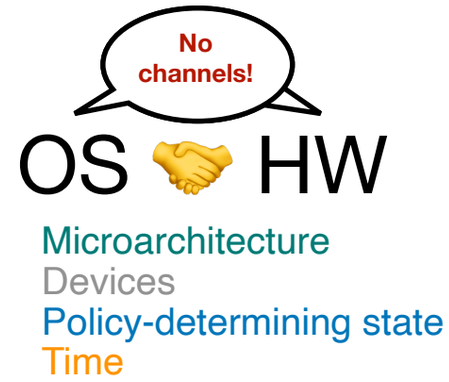
## Case 3: Domain switch

Timer interrupt delivered at (worst-case) $T_0 + w_0$

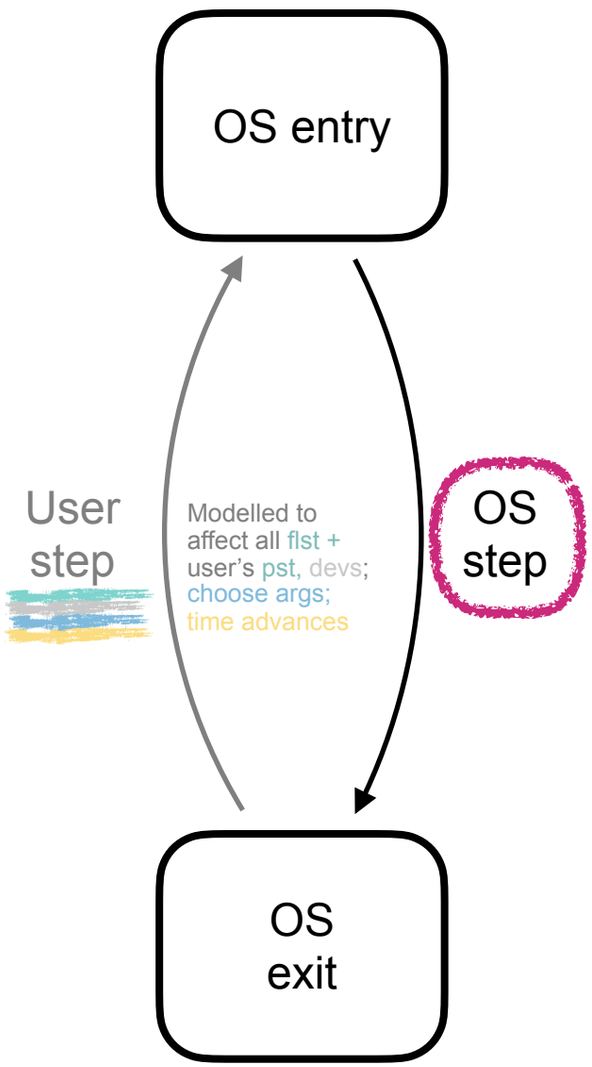



**Architecture-specific**

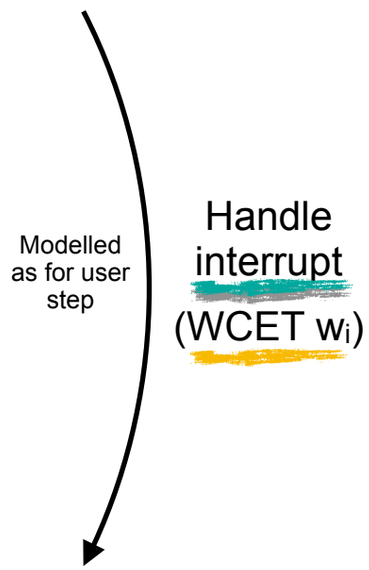
# OS security model




## Transition system

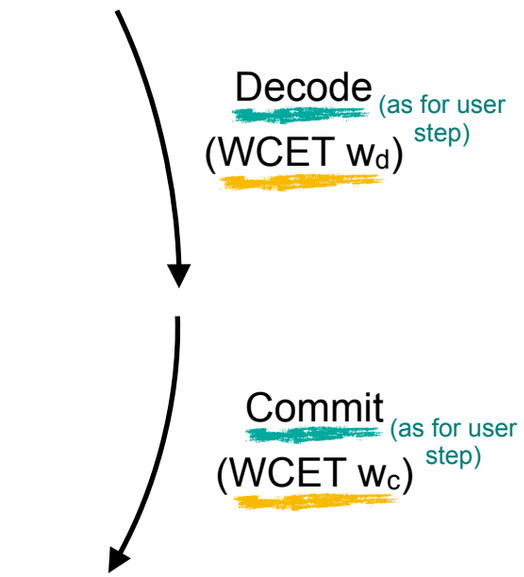



## Case 1: Device interrupt



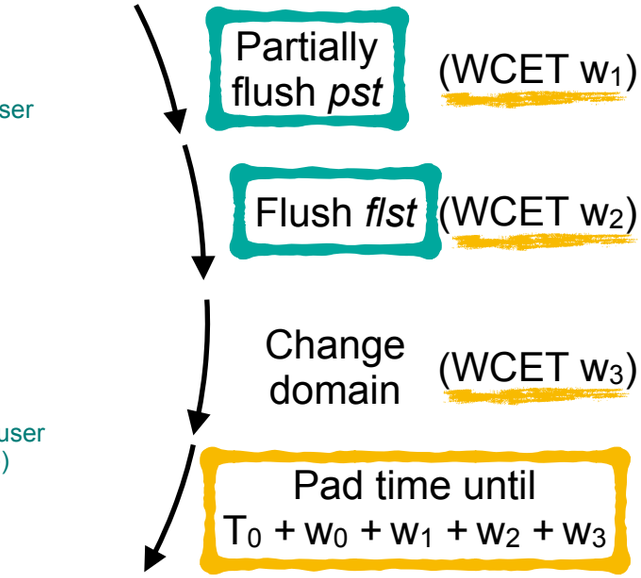

**Architecture-specific**

## Case 2: System call



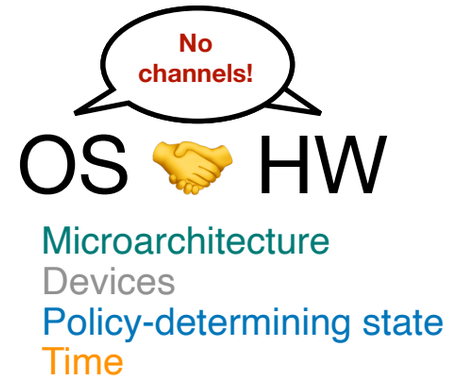

**OS-specific** (incl. *infoflow policies)*

## Case 3: Domain switch

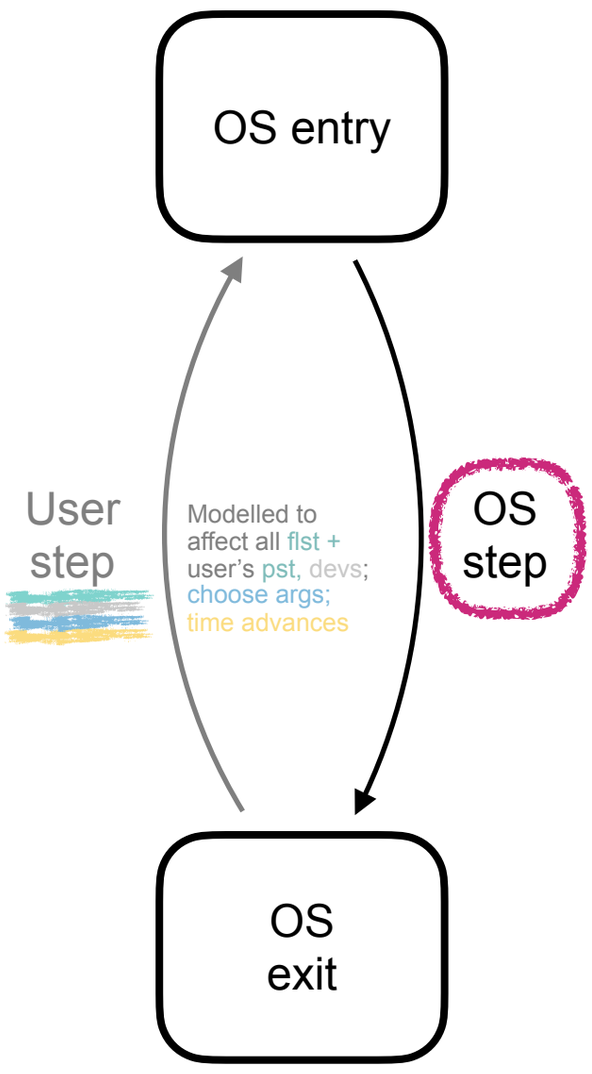



**Architecture-specific**

# OS security model

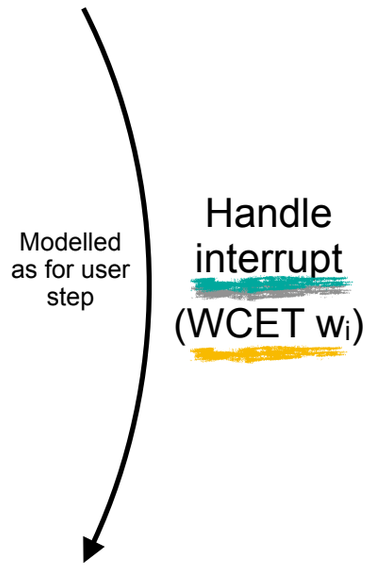



## Transition system




## Case 1: Device interrupt

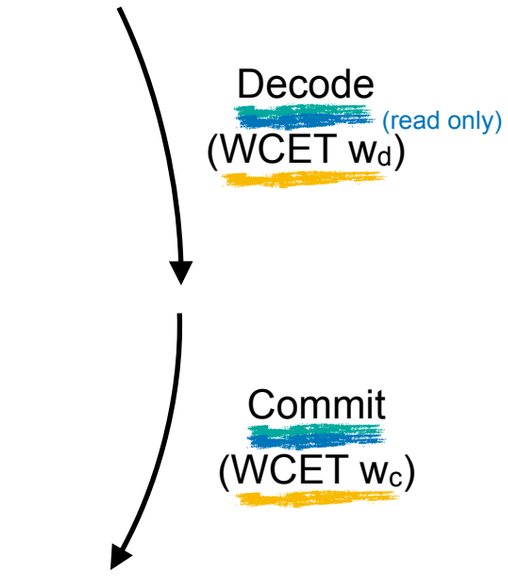
Where $w_i \leq w_0$




**Architecture-specific**

## Case 2: System call

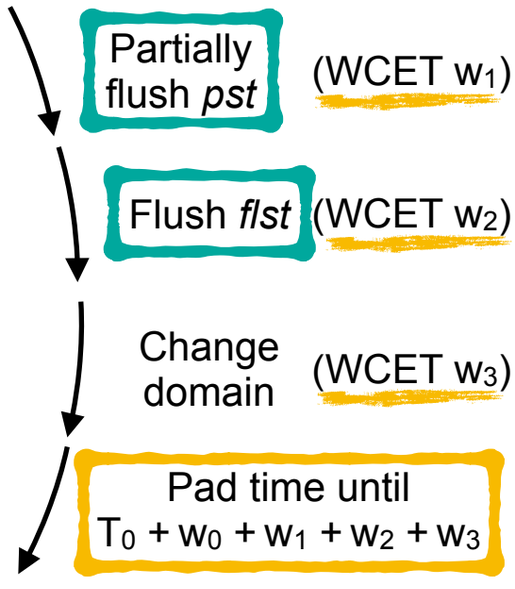
Where $w_d + w_c \leq w_0$




**OS-specific** (incl. *infoflow policies)*

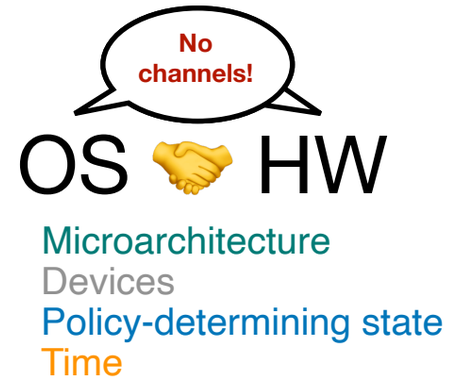
## Case 3: Domain switch

Timer interrupt delivered at (worst-case) $T_0 + w_0$

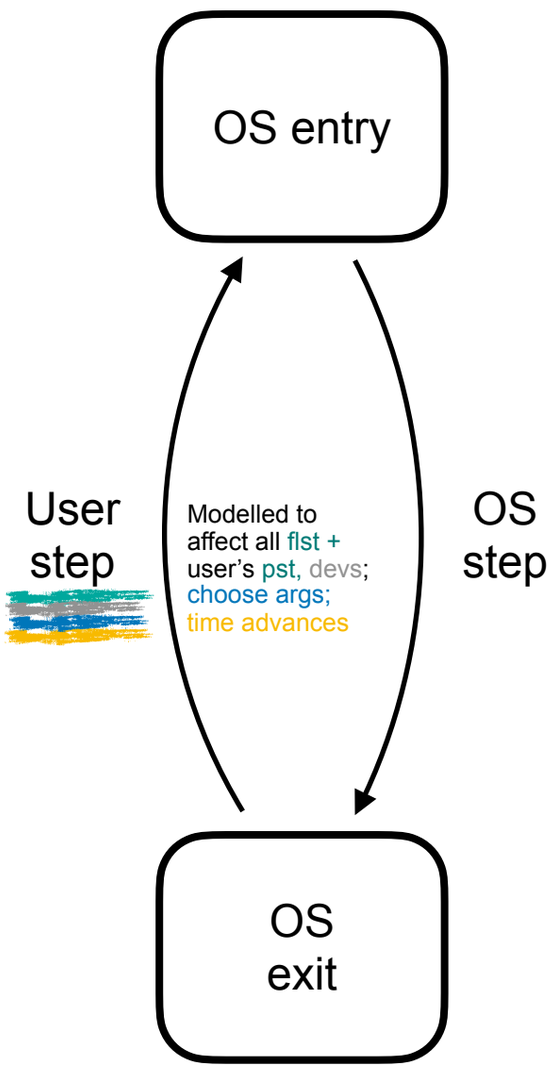



**Architecture-specific**

# Security proof approach

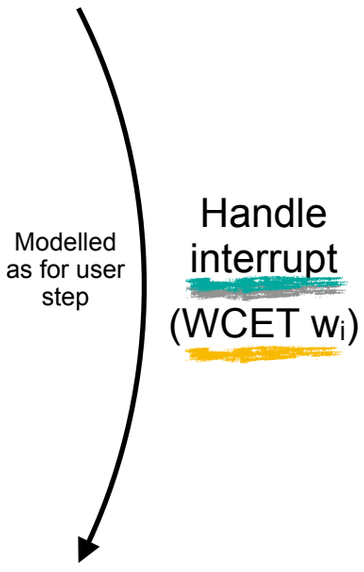



## Transition system




## Case 1: Device interrupt

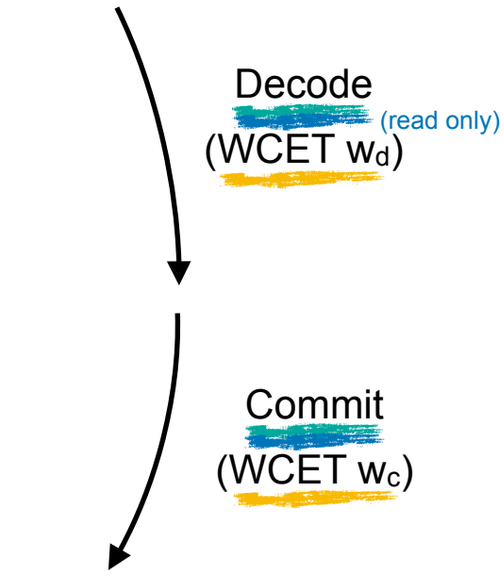
Where $w_i \leq w_0$




**Architecture-specific**

## Case 2: System call

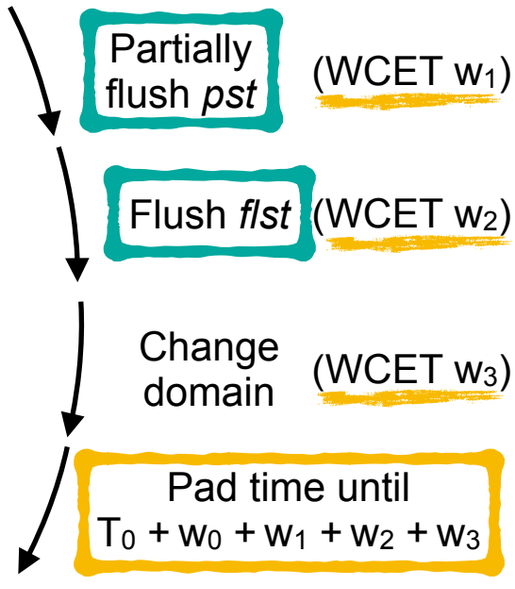
Where $w_d + w_c \leq w_0$




**OS-specific** (incl. *infoflow policies*)

## Case 3: Domain switch

Timer interrupt delivered at (worst-case) $T_0 + w_0$



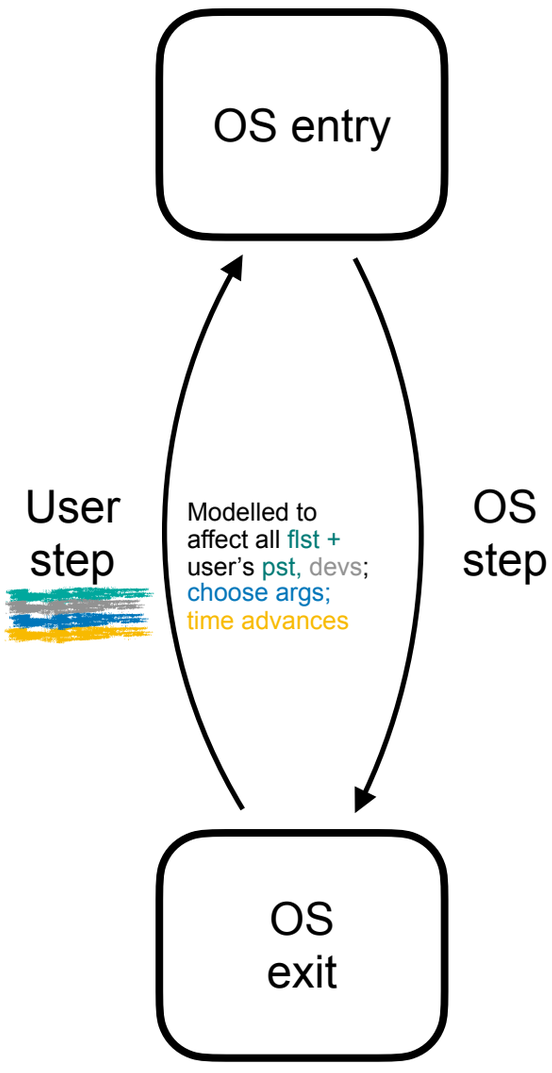

**Architecture-specific**

# Security proof approach


Microarchitecture
Devices
Policy-determining state
Time

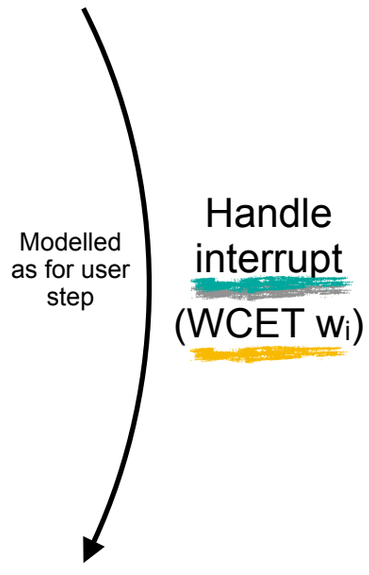
**Requirements**
(In addition to WCETs)

## Transition system




## Case 1: Device interrupt

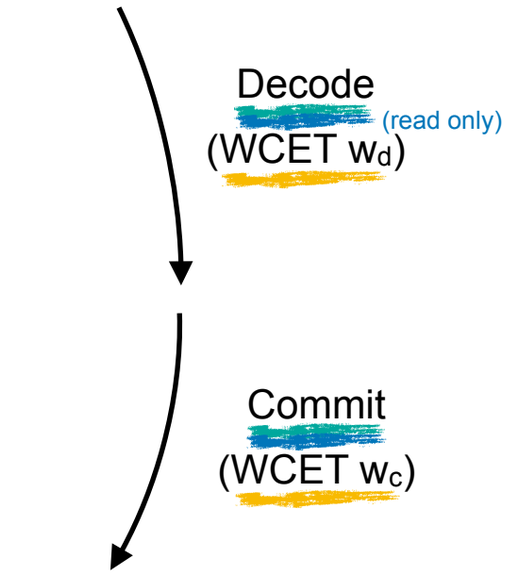
Where $w_i \leq w_0$




**Architecture-specific**

## Case 2: System call

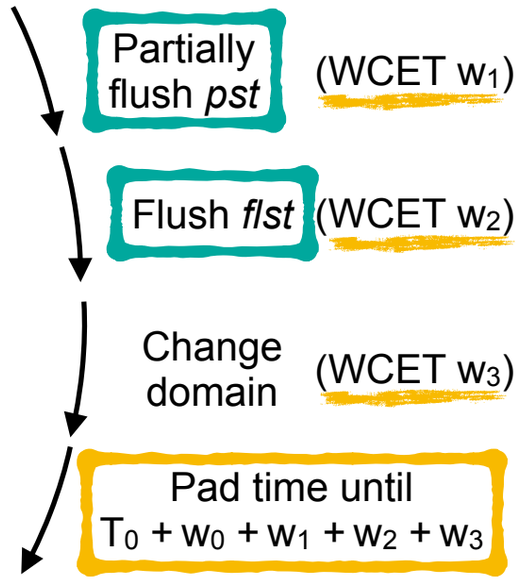
Where $w_d + w_c \leq w_0$




**OS-specific** (incl. *infoflow policies*)

## Case 3: Domain switch

Timer interrupt delivered at (worst-case) $T_0 + w_0$



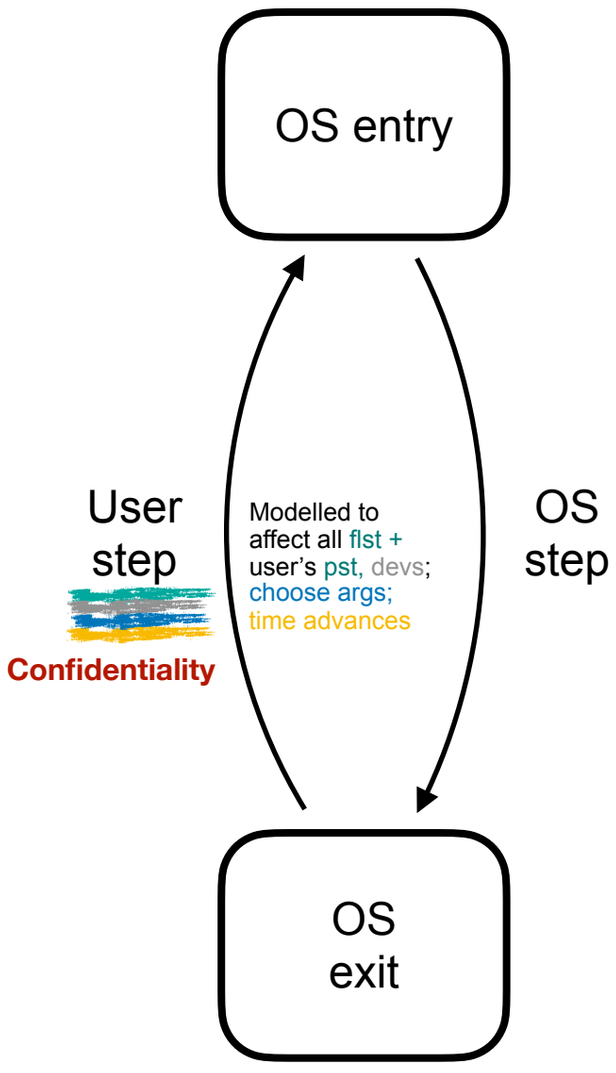

**Architecture-specific**

# Security proof approach


Microarchitecture
Devices
Policy-determining state
Time

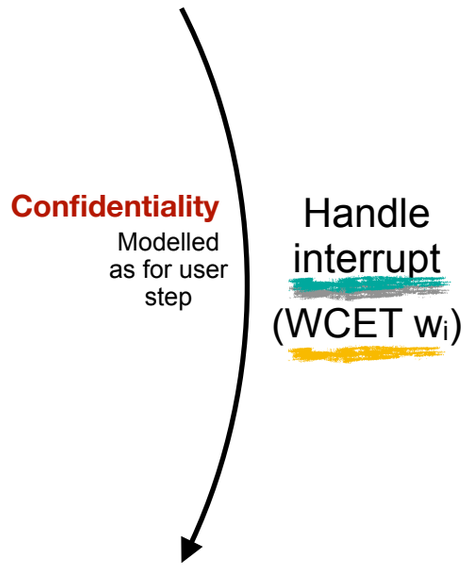
**Requirements**
(In addition to WCETs)

## Transition system




## Case 1: Device interrupt

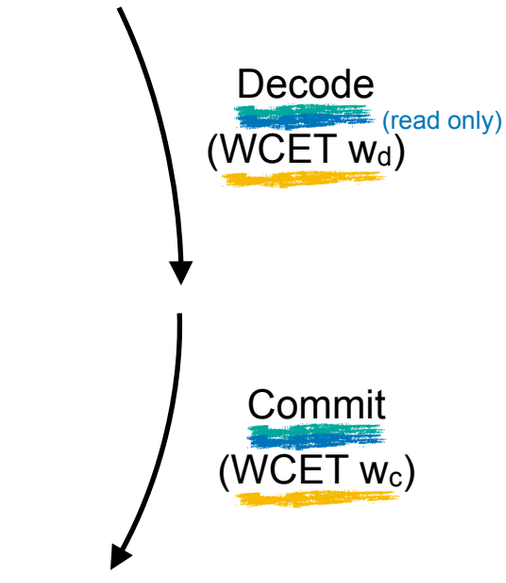
Where $w_i \leq w_0$




**Architecture-specific**

## Case 2: System call

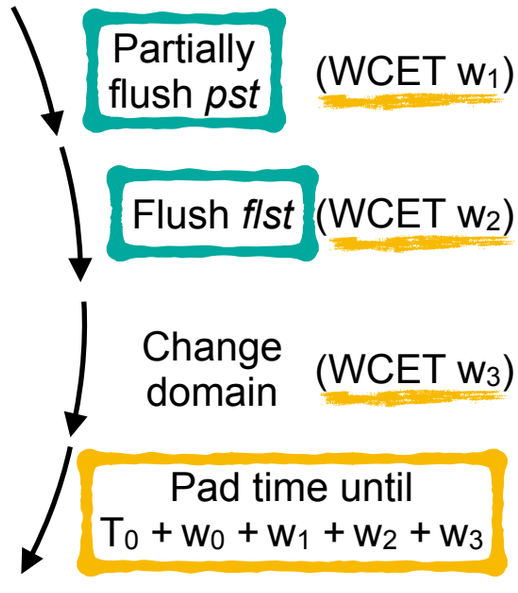
Where $w_d + w_c \leq w_0$




**OS-specific** (incl. *infoflow policies*)

## Case 3: Domain switch

Timer interrupt delivered at (worst-case) $T_0 + w_0$



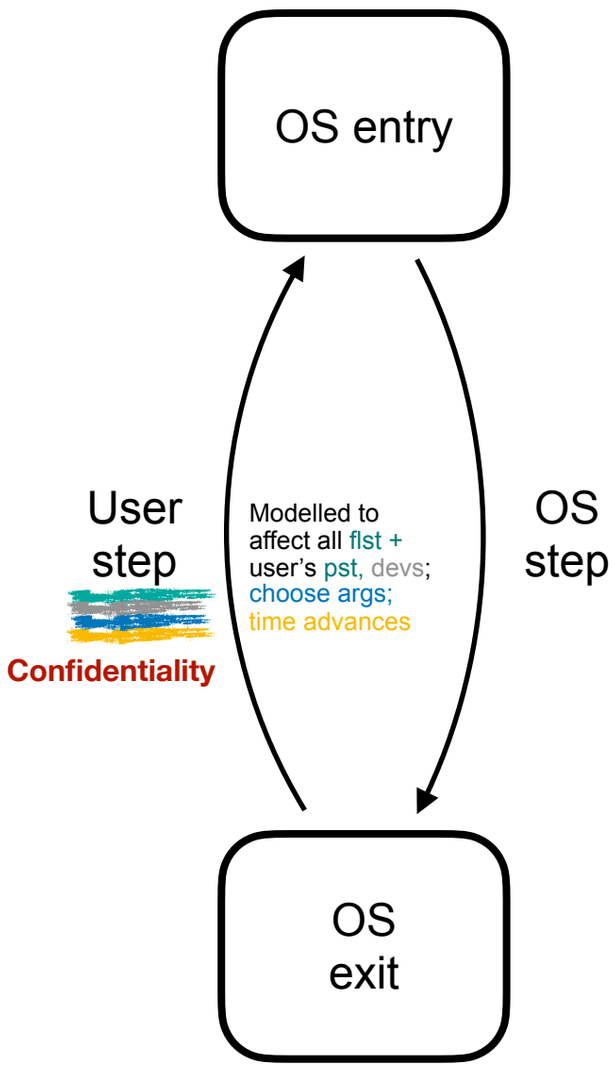

**Architecture-specific**

# Security proof approach


Microarchitecture
Devices
Policy-determining state
Time

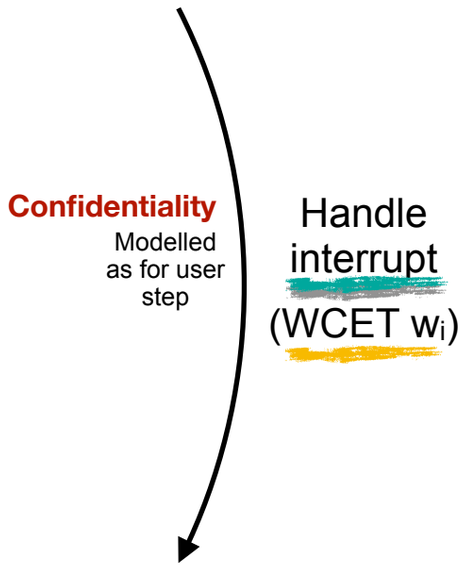
**Requirements**
(In addition to WCETs)

## Transition system




## Case 1: Device interrupt

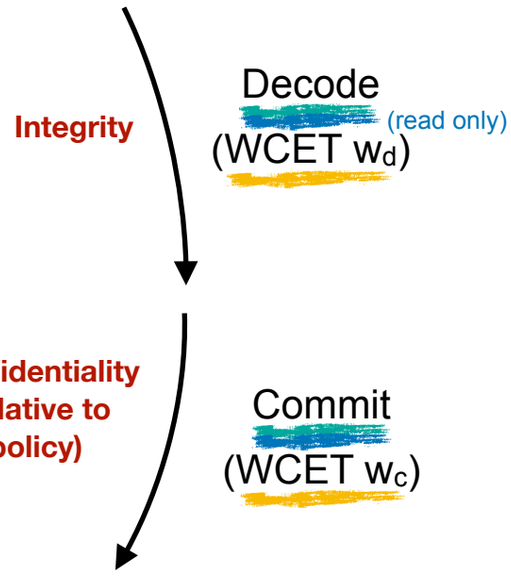
Where $w_i \leq w_0$




**Architecture-specific**

## Case 2: System call

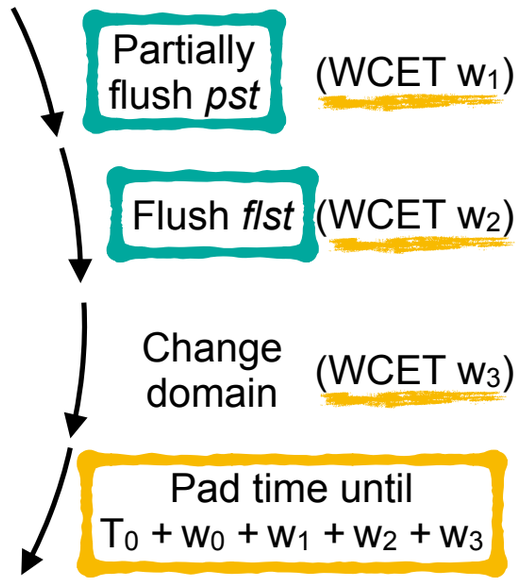
Where $w_d + w_c \leq w_0$




**OS-specific** (incl. *infoflow policies*)

## Case 3: Domain switch

Timer interrupt delivered at (worst-case) $T_0 + w_0$



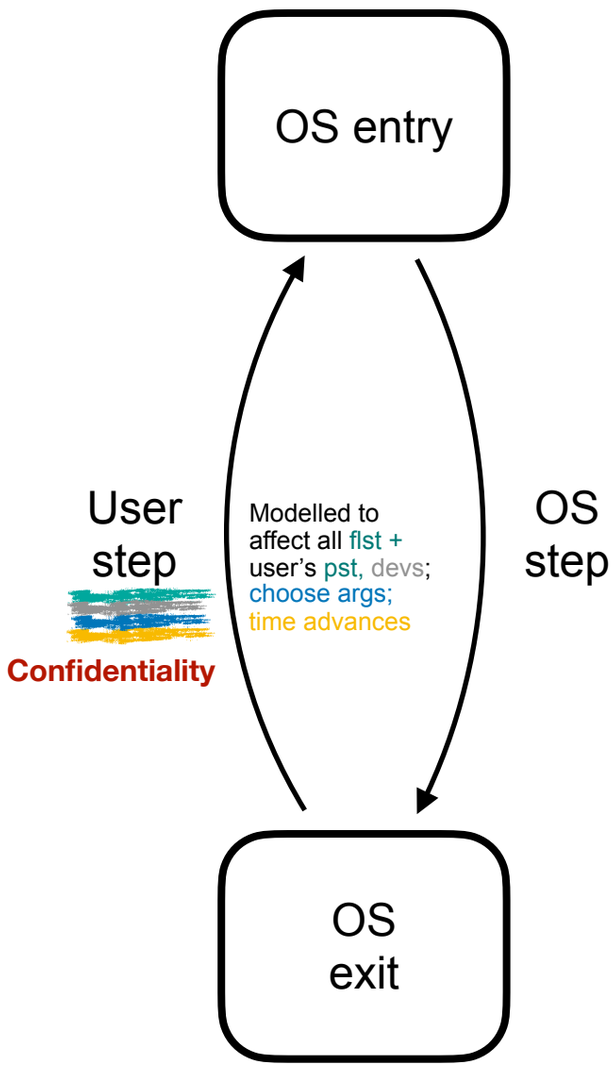

**Architecture-specific**

# Security proof approach


Microarchitecture
Devices
Policy-determining state
Time

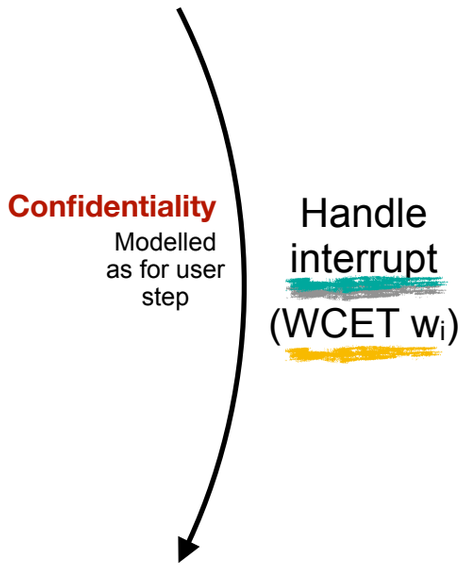
**Requirements**
(In addition to WCETs)

## Transition system




## Case 1: Device interrupt

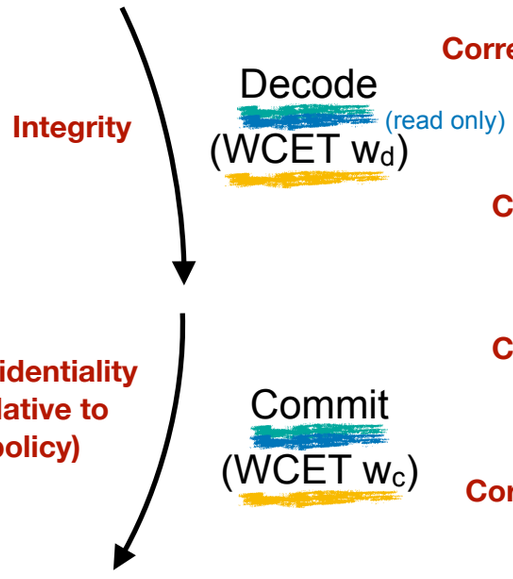
Where $w_i \leq w_0$




**Architecture-specific**

## Case 2: System call

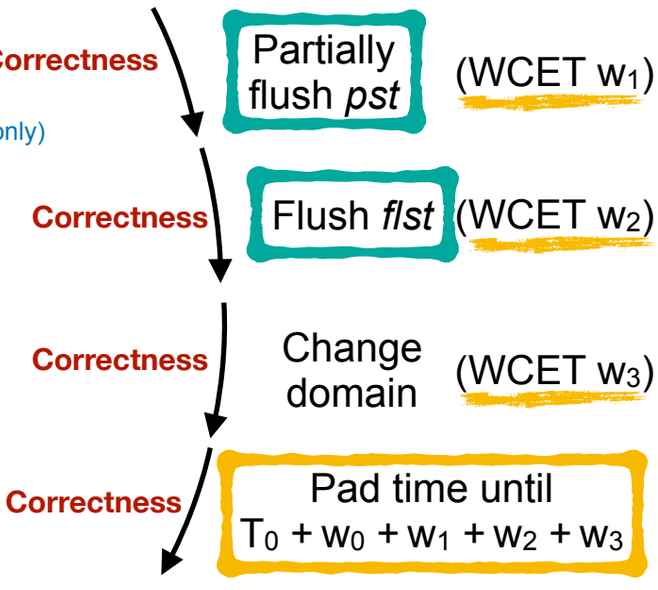
Where $w_d + w_c \leq w_0$




**OS-specific** (incl. *infoflow policies*)

## Case 3: Domain switch

Timer interrupt delivered at (worst-case) $T_0 + w_0$



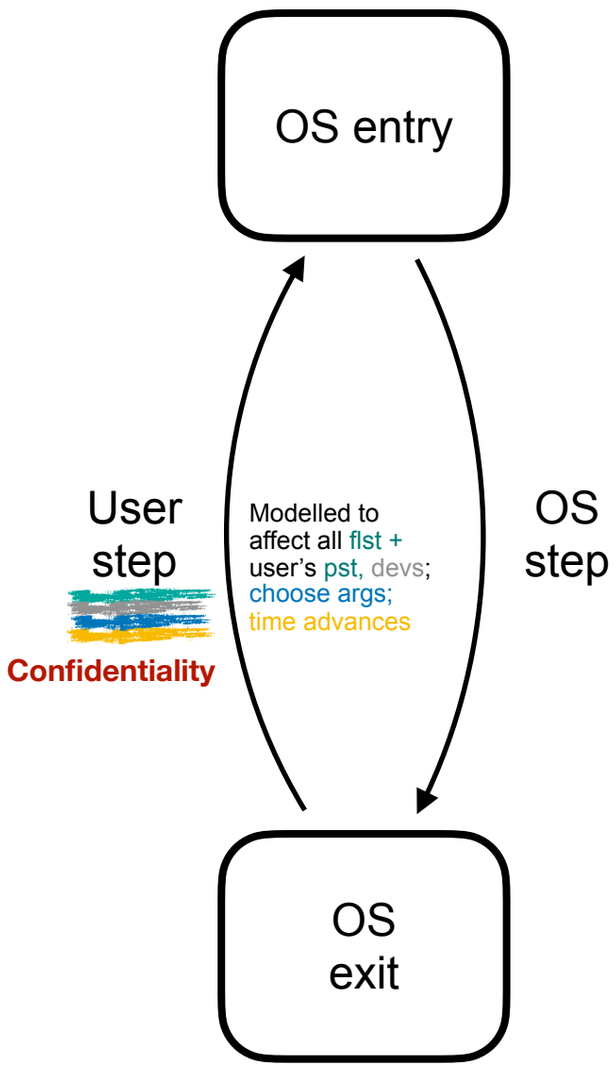

**Architecture-specific**

# Security proof approach


Microarchitecture
Devices
Policy-determining state
Time

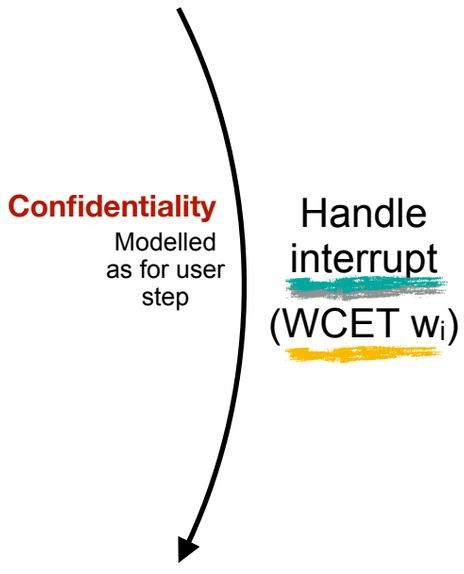
**Requirements**
(In addition to WCETs)

## Transition system




## Case 1: Device interrupt

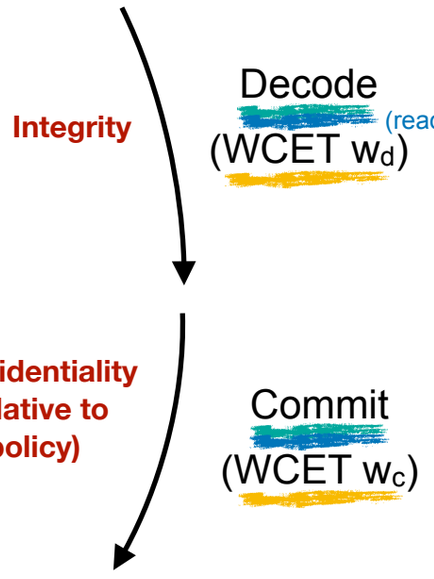
Where $w_i \leq w_0$




**Architecture-specific**

## Case 2: System call

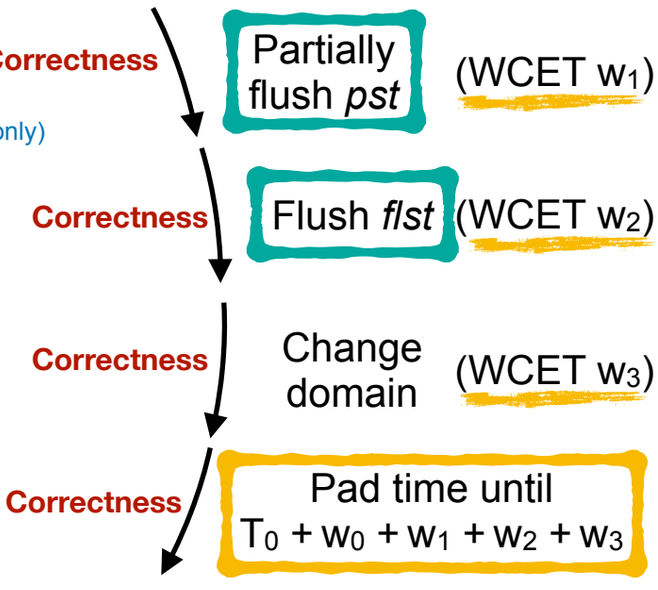
Where $w_d + w_c \leq w_0$




**OS-specific** (incl. *infoflow policies*)

## Case 3: Domain switch

Timer interrupt delivered at (worst-case) $T_0 + w_0$




**Architecture-specific**

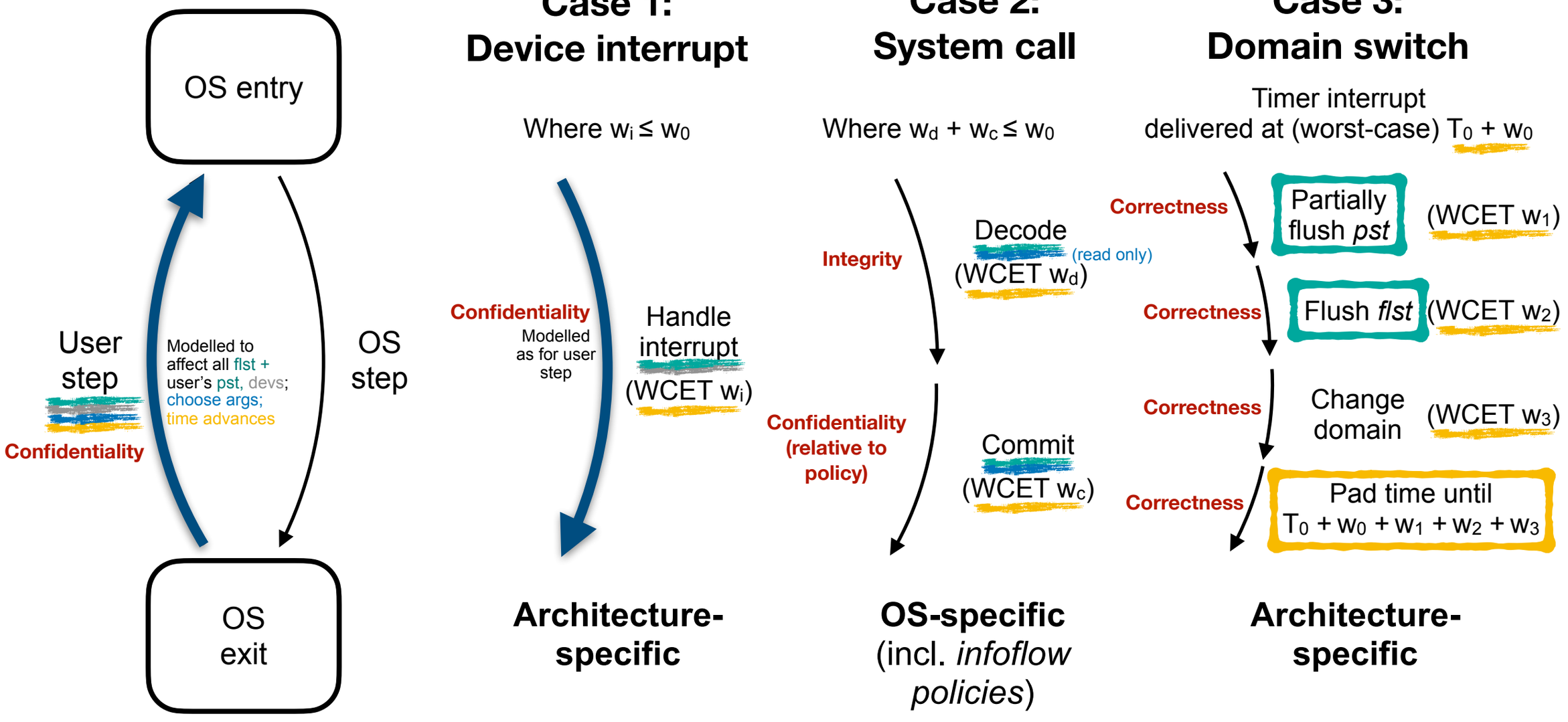
We prove: **Confidentiality property (bisimulation) step lemmas**

# Security proof approach


Microarchitecture
Devices
Policy-determining state
Time

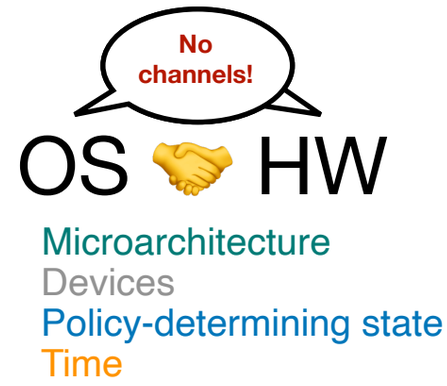
**Requirements**
(In addition to WCETs)

## Transition system



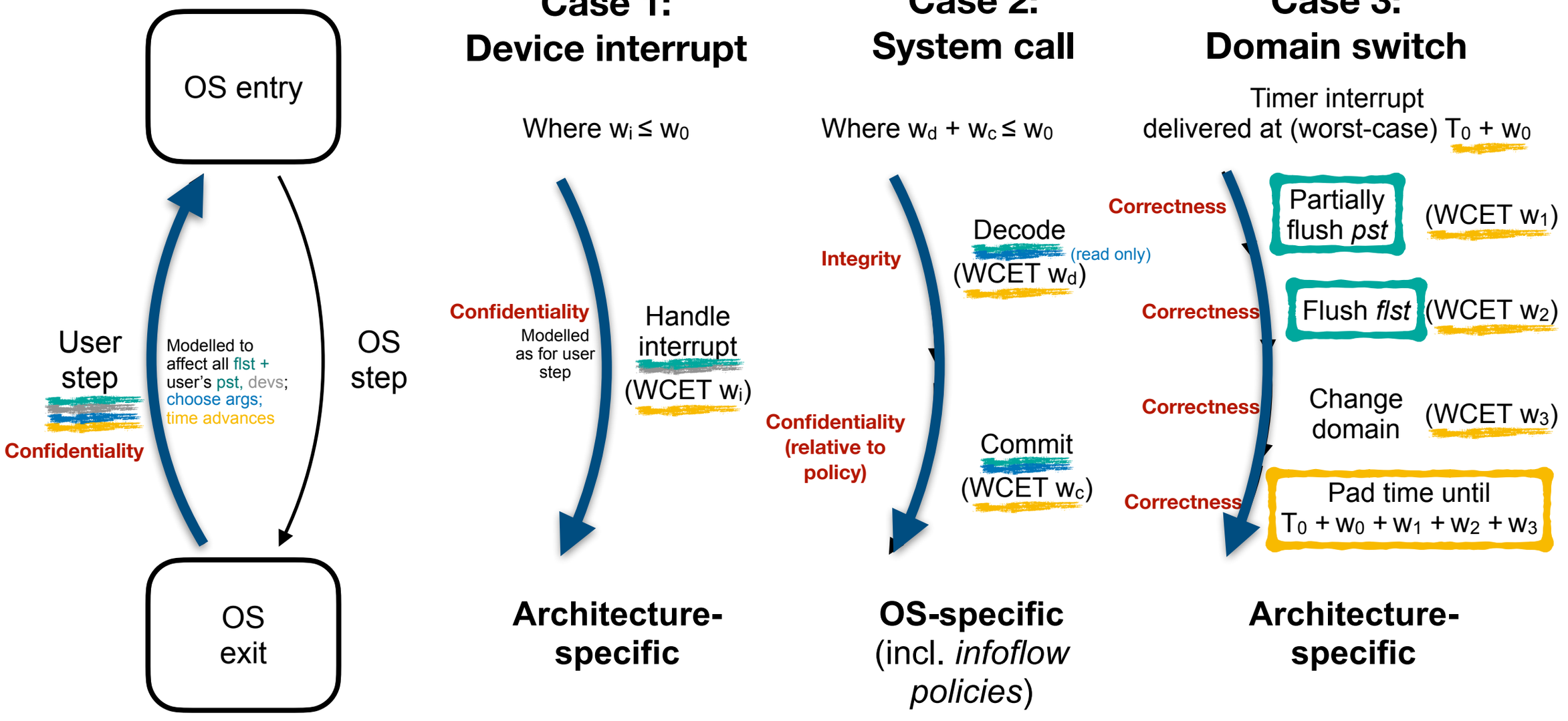

We prove: **Confidentiality property (bisimulation) step lemmas**

# Security proof approach




**Requirements**
(In addition to WCETs)

## Transition system



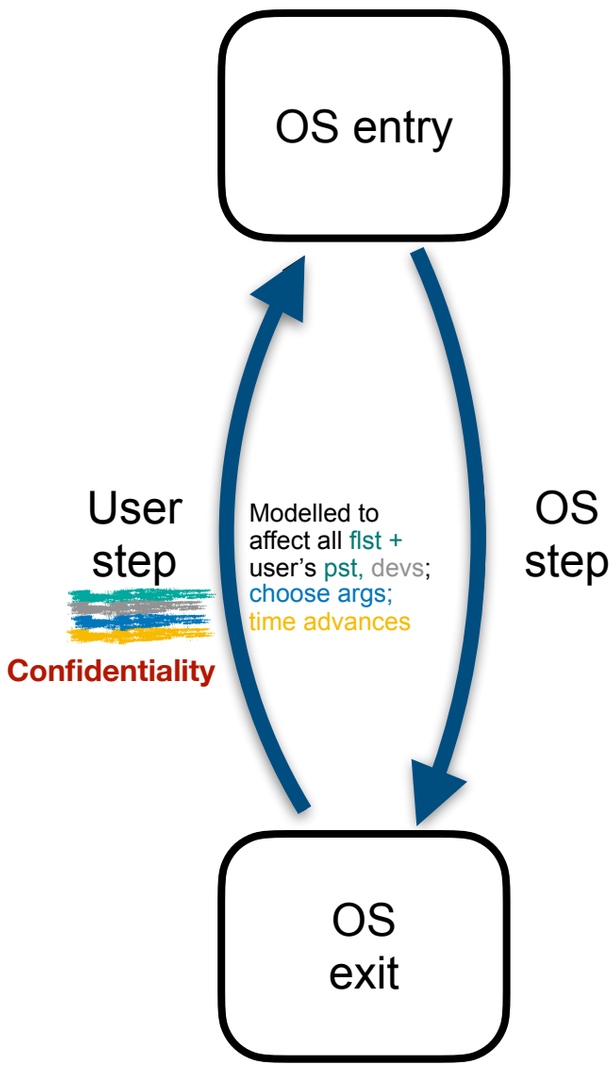

We prove: **Confidentiality property (bisimulation) step lemmas**

# Security proof approach


Microarchitecture
Devices
Policy-determining state
Time

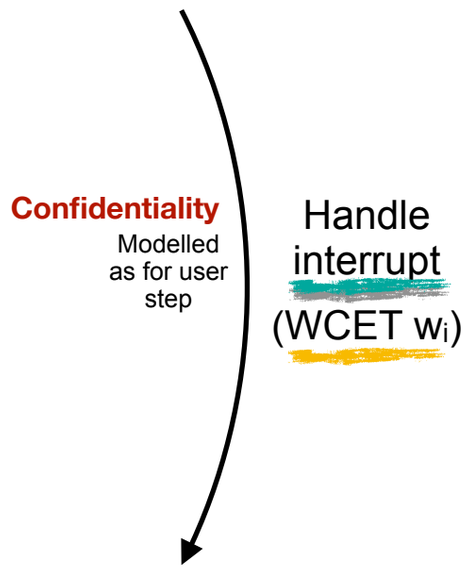
**Requirements**
(In addition to WCETs)

## Transition system




## Case 1: Device interrupt

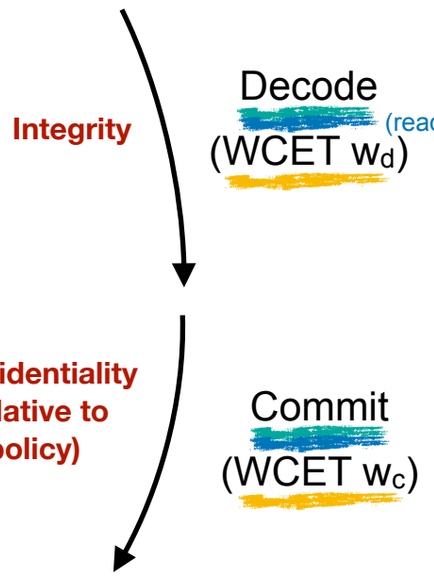
Where $w_i \leq w_0$




**Architecture-specific**

## Case 2: System call

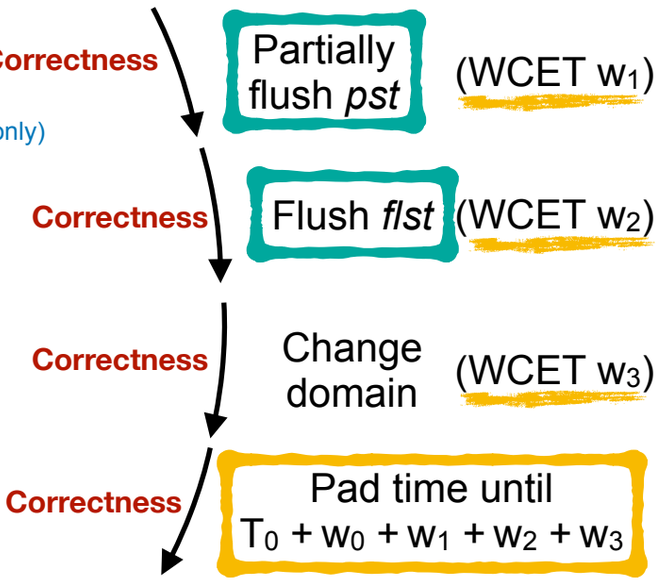
Where $w_d + w_c \leq w_0$




**OS-specific** (incl. *infoflow policies*)

## Case 3: Domain switch

Timer interrupt delivered at (worst-case) $T_0 + w_0$

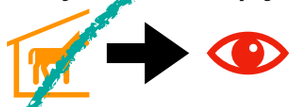



**Architecture-specific**

We prove: **Confidentiality property (bisimulation) step lemmas**

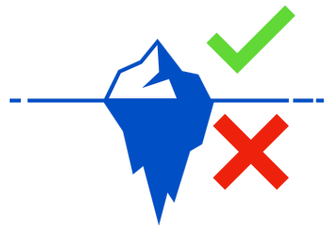
# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy




OS 🤝 HW

Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.



Make security property precise enough to exclude flows from covert state.


Demonstrating these principles,
we formalised in Isabelle/HOL:

1. OS security model imposing requirements on relevant parts of OS.
(Intended for seL4, but *generic*)

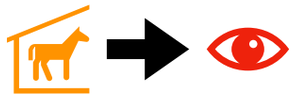
2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

3. Proof our security property holds if OS model's requirements hold.

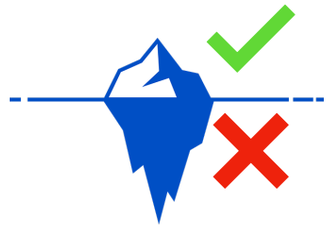
4. Basic instantiation of OS model exercising dynamic policy.

# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy




Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.



Make security property precise enough to exclude flows from covert state.


Demonstrating these principles,
we formalised in Isabelle/HOL:

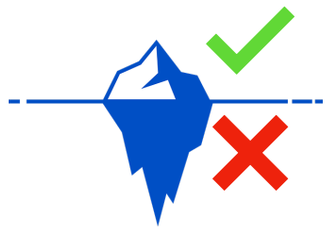
1. OS security model imposing requirements on relevant parts of OS.
(Intended for seL4, but *generic*)

2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

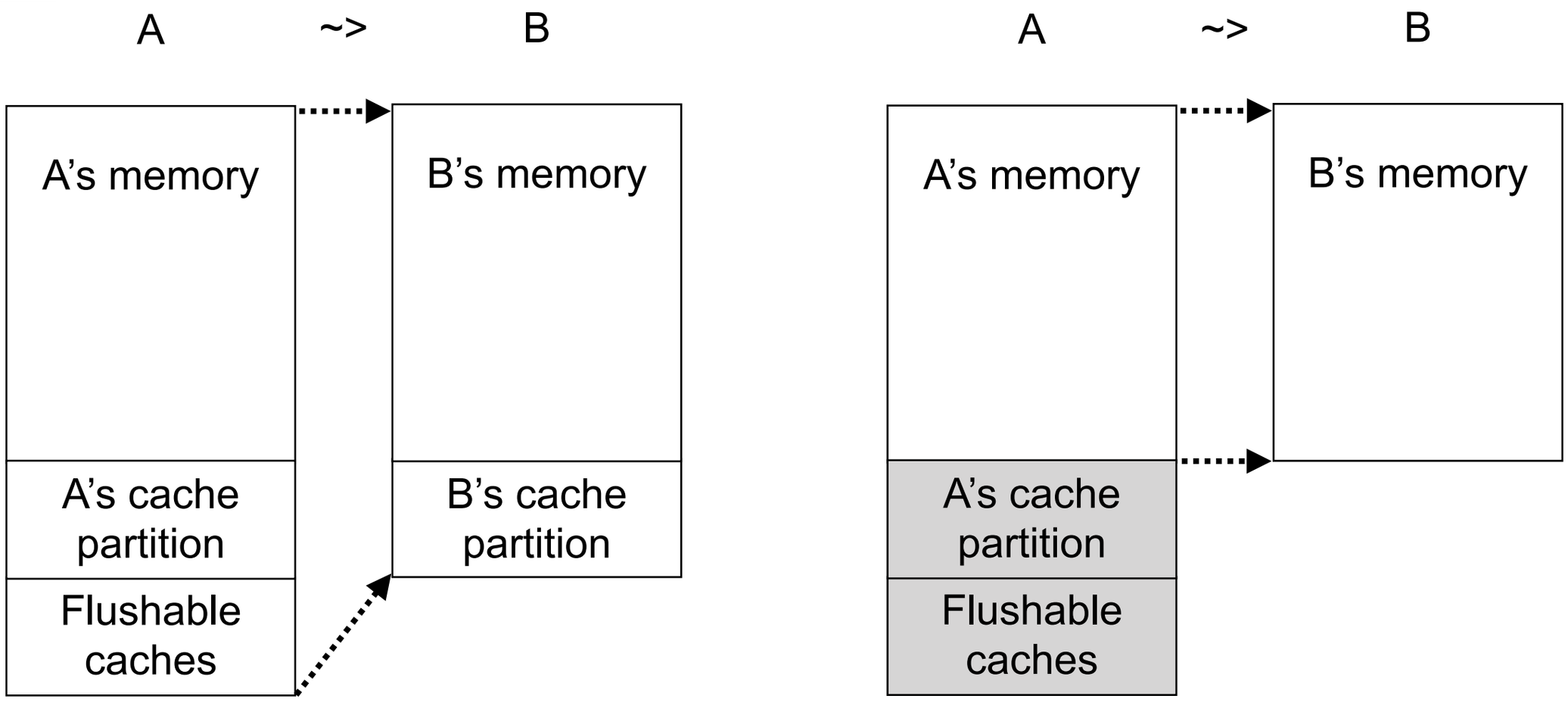
3. Proof our security property holds if OS model's requirements hold.

4. Basic instantiation of OS model exercising dynamic policy.

# OS security property



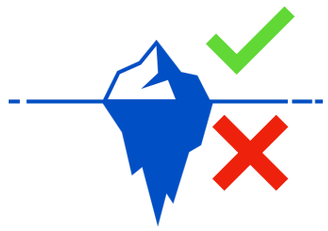
Recall:



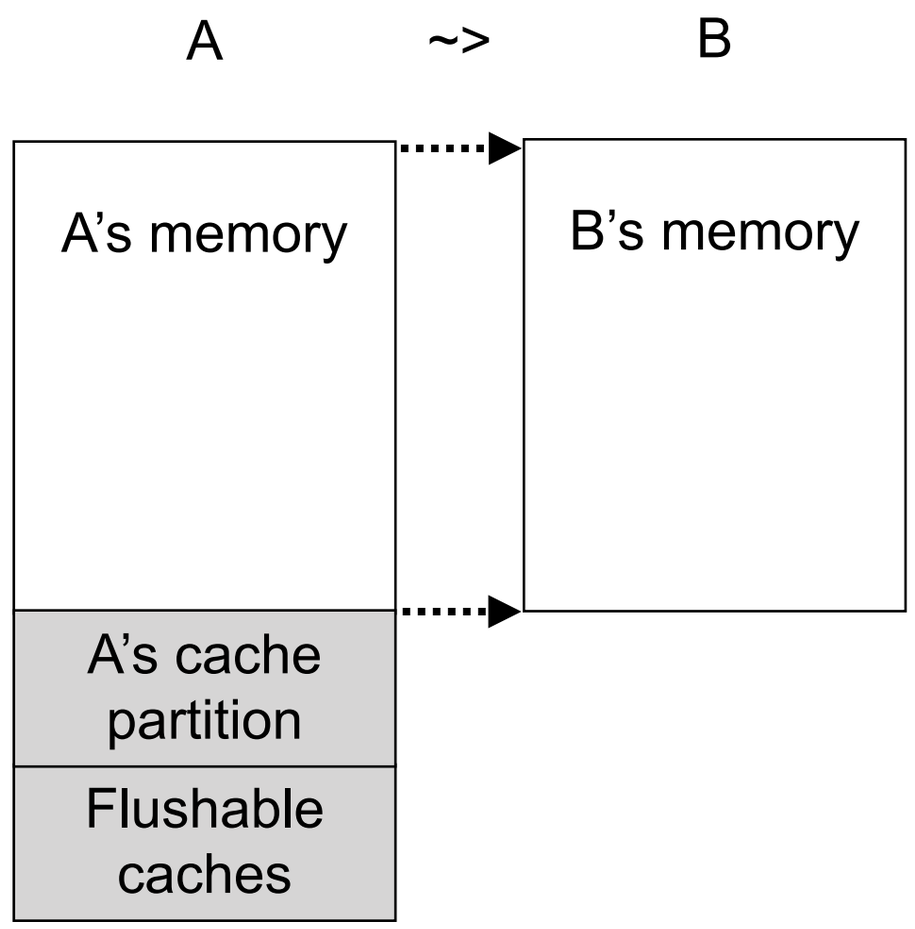

From prior seL4 infoflow proofs [Murray et al. 2012, 2013]:
*"all or nothing" policies*

**For time protection, need *spatial precision* to *allow some flows* but *exclude others***

# OS security property

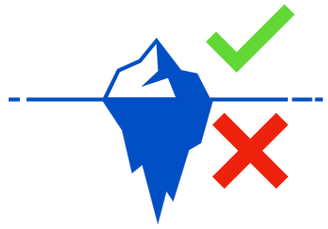


Our infoflow policies:



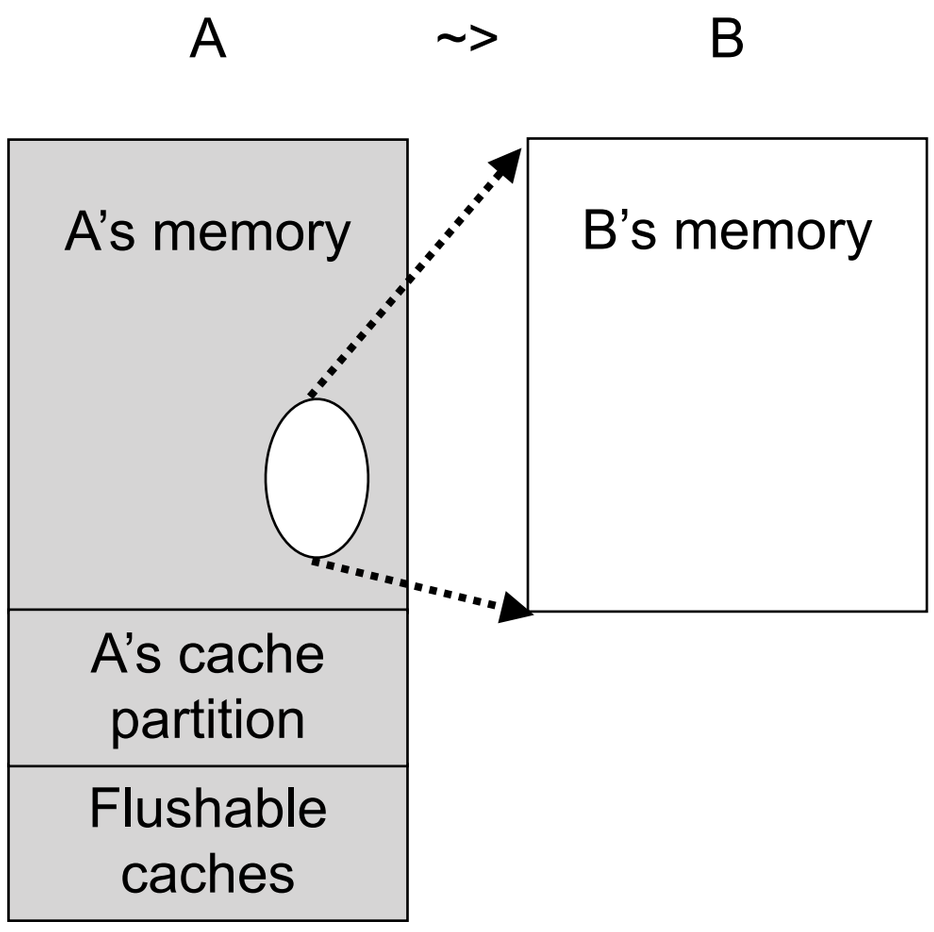

**For time protection, need *spatial precision* to *allow some flows* but *exclude others***

# OS security property



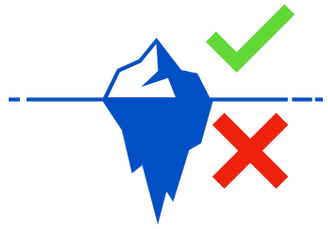
Our infoflow policies:

- *Arbitrary* spatial precision




**For time protection, need *spatial precision* to *allow some flows* but *exclude others***

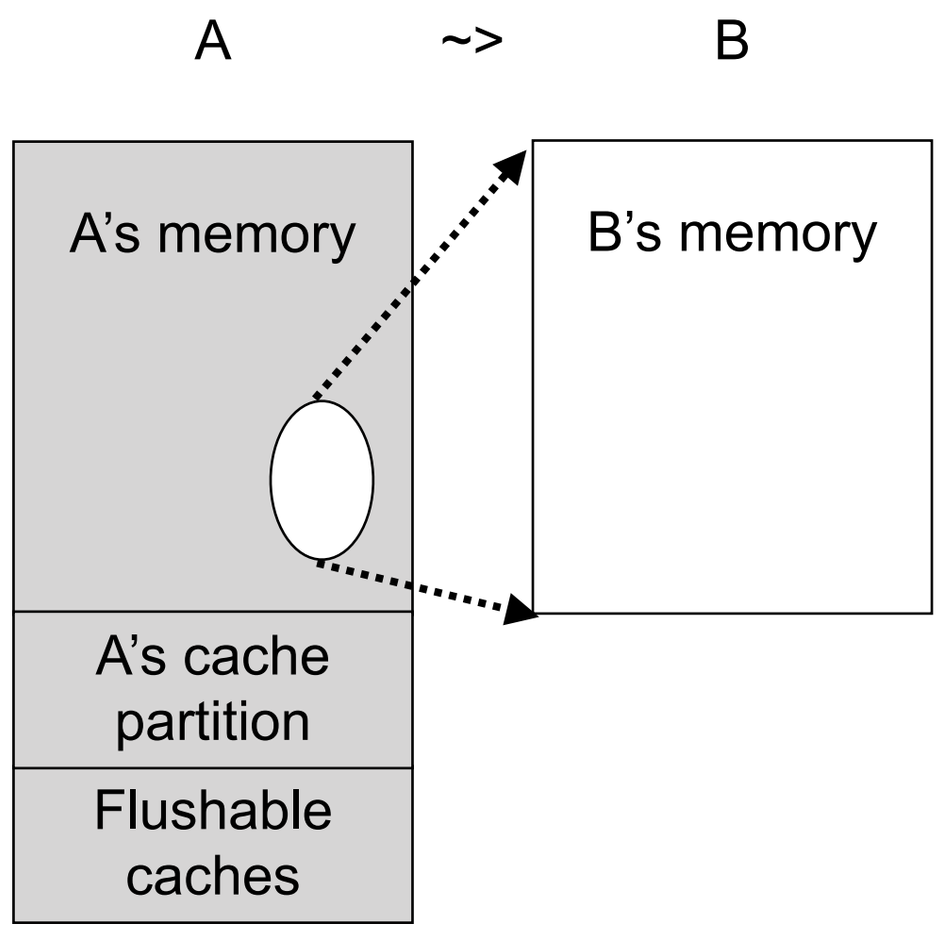
# OS security property



Our infoflow policies:

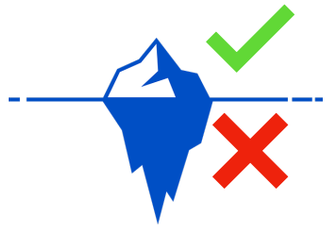
- *Arbitrary* spatial precision
- *Policy channels* specified as *state relations*: $_s\ ^{|A \underset{\sim}{\rightsquigarrow} B|}\ _t$

  If $^{|A \underset{\sim}{\rightsquigarrow} B|}$ equates part of A, then info flow is allowed from there to B.




**For time protection, need *spatial precision* to *allow some flows* but *exclude others***

# OS security property

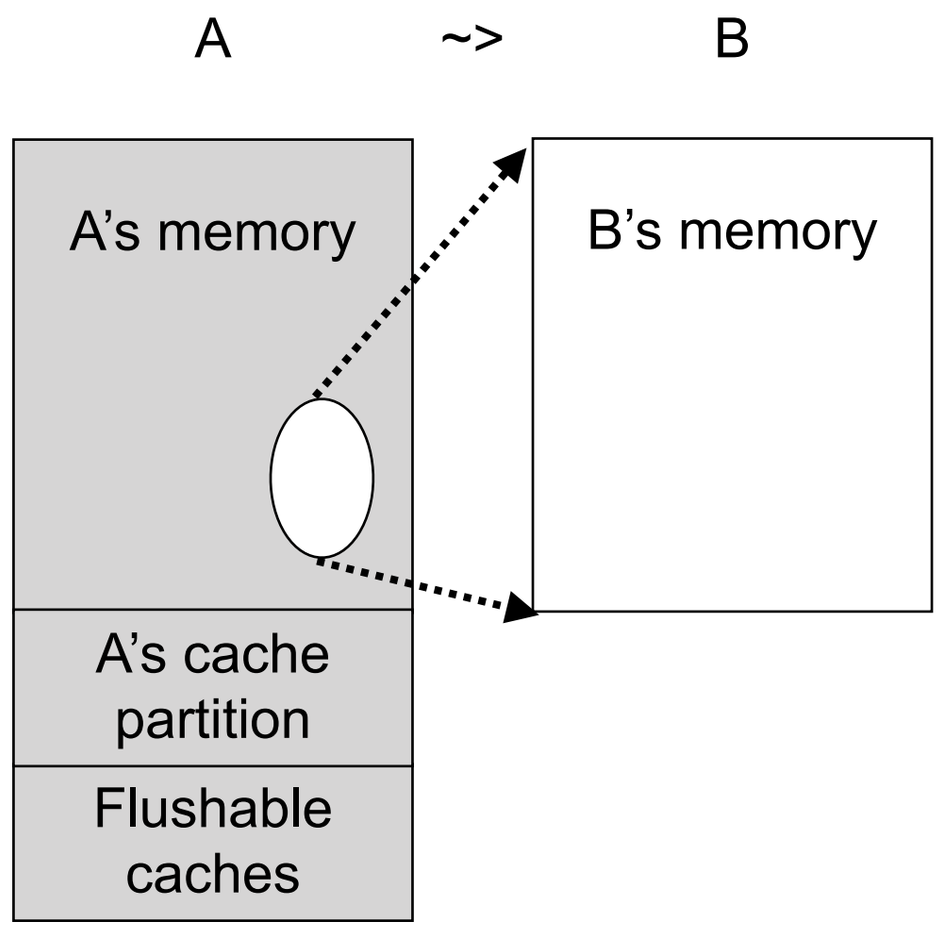


Our infoflow policies:

- *Arbitrary* spatial precision
- *Policy channels* specified as *state relations*: $_s\ ^{|A \underset{\sim}{\rightsquigarrow} B|}\ _t$

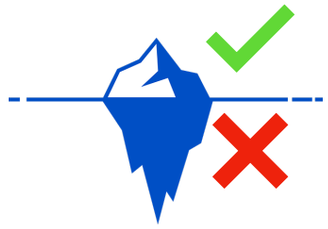
  If $^{|A \underset{\sim}{\rightsquigarrow} B|}$ equates part of A, then info flow is allowed from there to B.

- Also arbitrary *temporal precision*




**For time protection, need *spatial precision* to *allow some flows* but *exclude others***

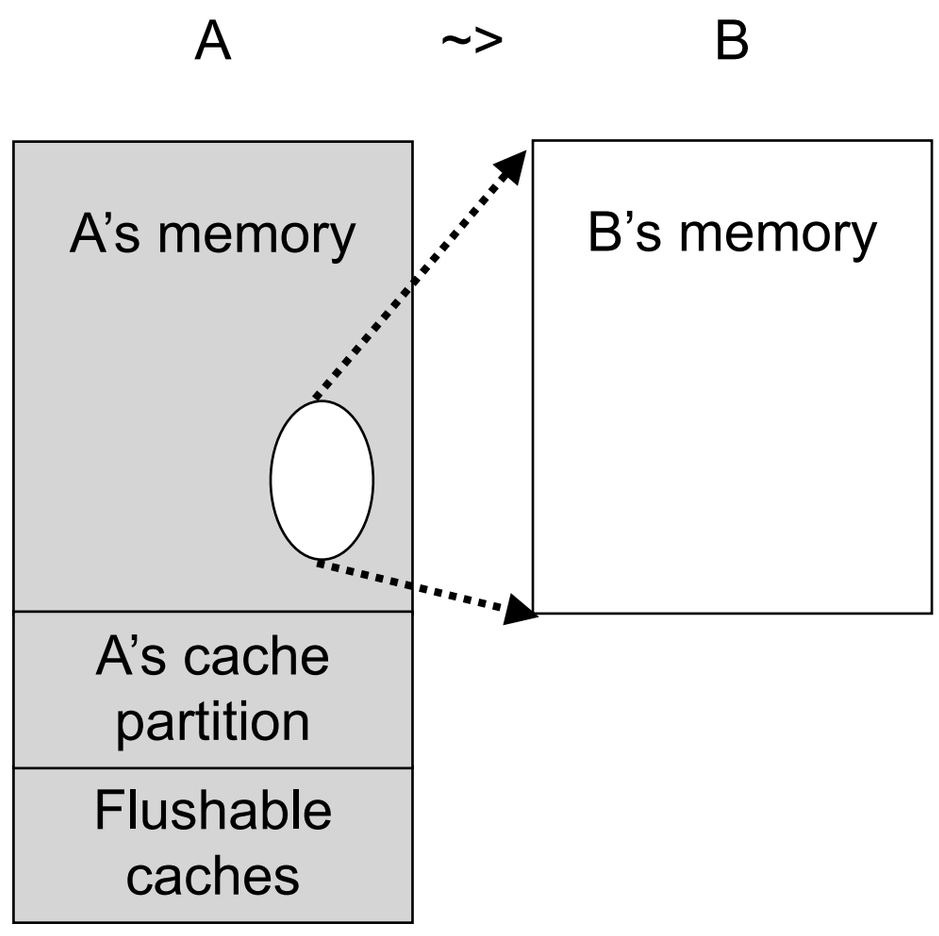
# OS security property



Our infoflow policies:

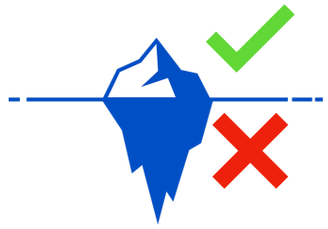
- *Arbitrary* spatial precision
- *Policy channels* specified as *state relations*: $_s\ ^{|A \underset{\sim}{\rightsquigarrow} B|}\ _t$

  If $^{|A \underset{\sim}{\rightsquigarrow} B|}$ equates part of A, then info flow is allowed from there to B.
- Also arbitrary *temporal precision*
- The *dynamicity* gives us *observer-relative* properties




**For time protection, need *spatial precision* to *allow some flows* but *exclude others***

# OS security property

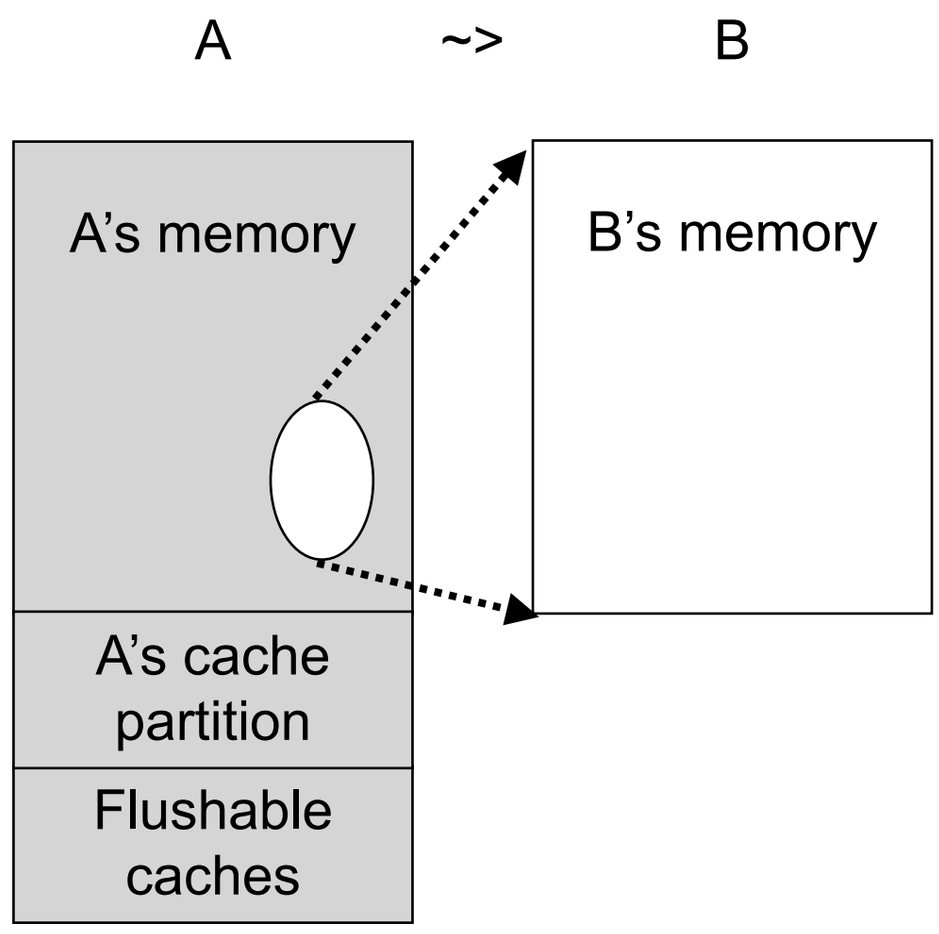


Our infoflow policies:

- *Arbitrary* spatial precision
- *Policy channels* specified as *state relations*: $_s\ ^{|A \underset{\sim}{\rightsquigarrow} B|}\ _t$

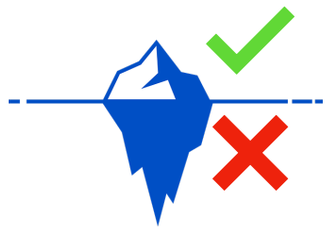
  If $^{|A \underset{\sim}{\rightsquigarrow} B|}$ equates part of A, then info flow is allowed from there to B.

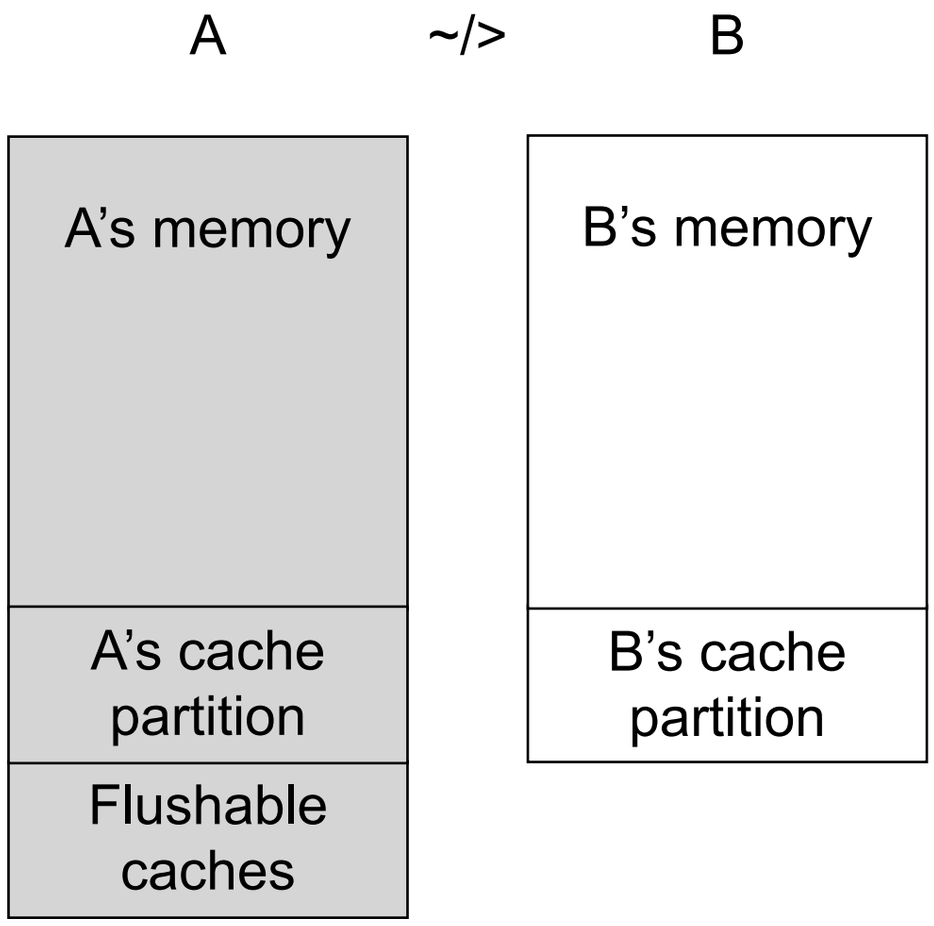
- Also arbitrary *temporal precision* ?
- The *dynamicity* gives us *observer-relative* properties



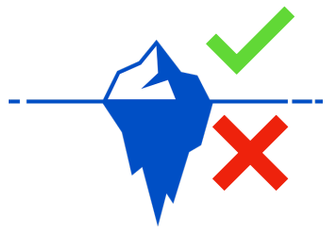

**For time protection, need *spatial precision* to *allow some flows* but *exclude others***

# Dynamic policy, observer relativity

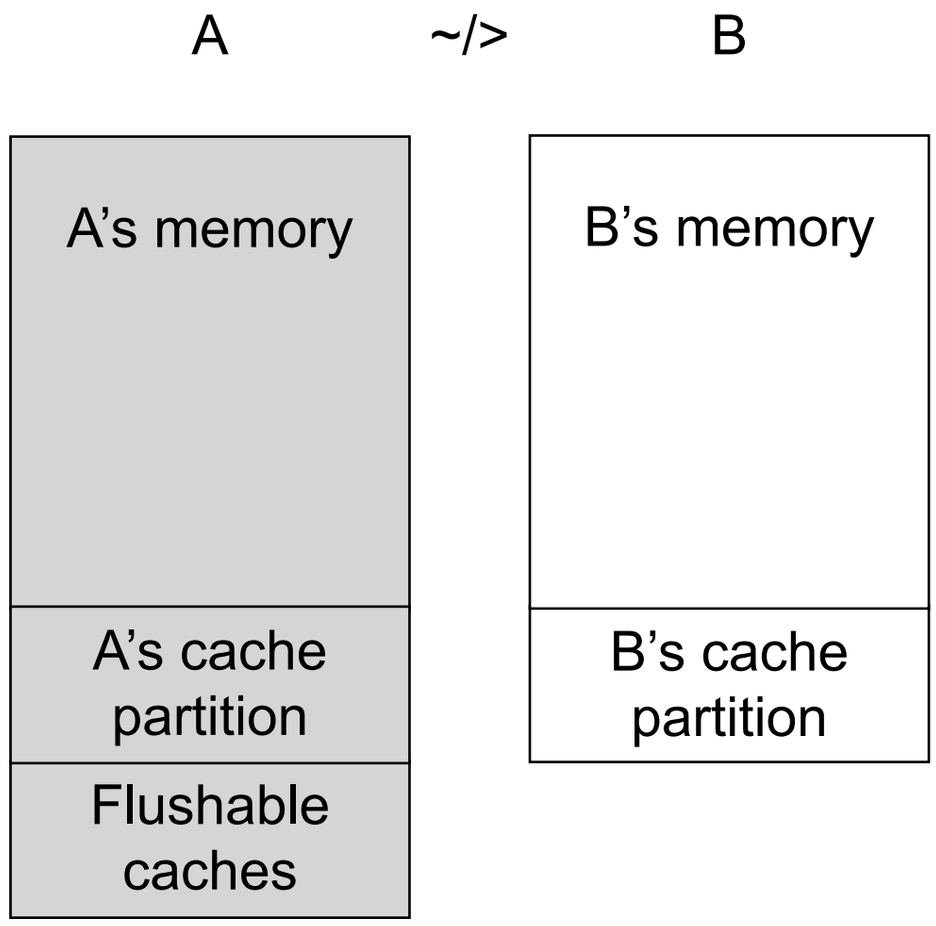




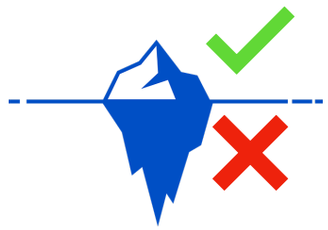
# Dynamic policy, observer relativity



Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**



# Dynamic policy, observer relativity

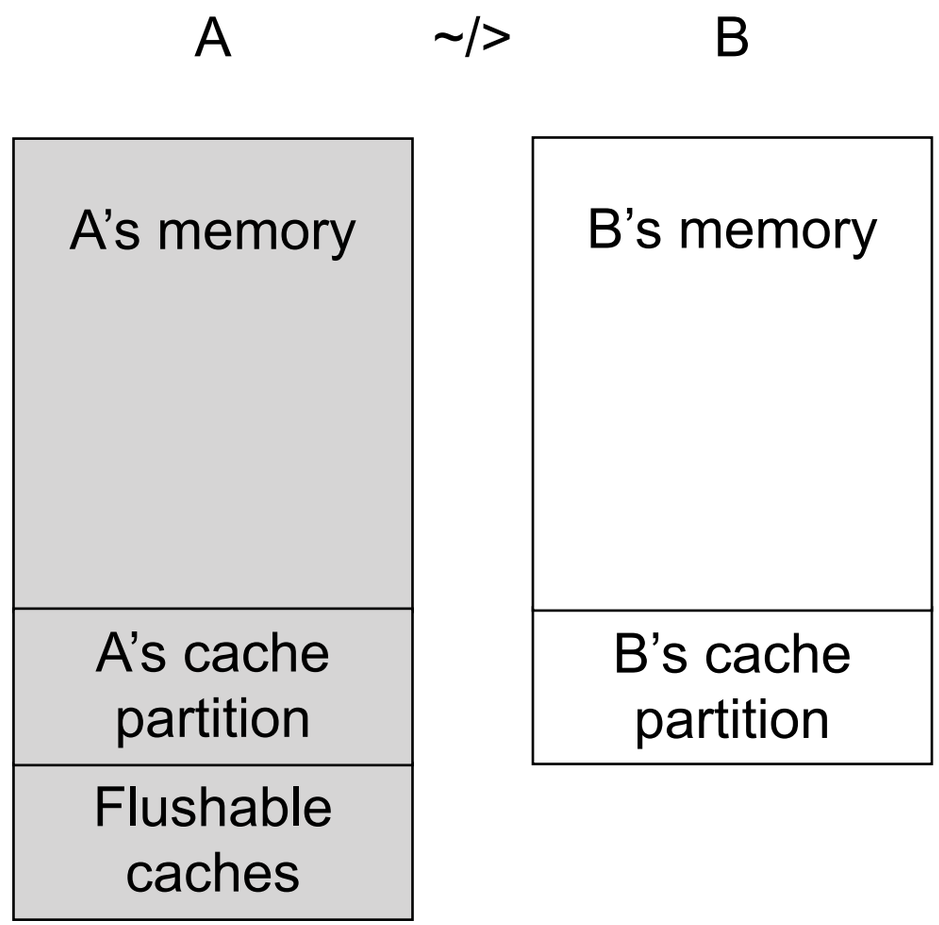


Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

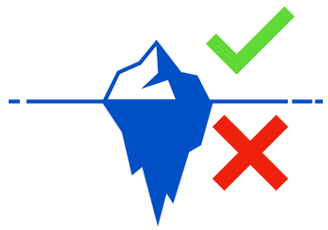
1. Dynamic policy: A ~> B ?

✓ Only when A calls

- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.



# Dynamic policy, observer relativity

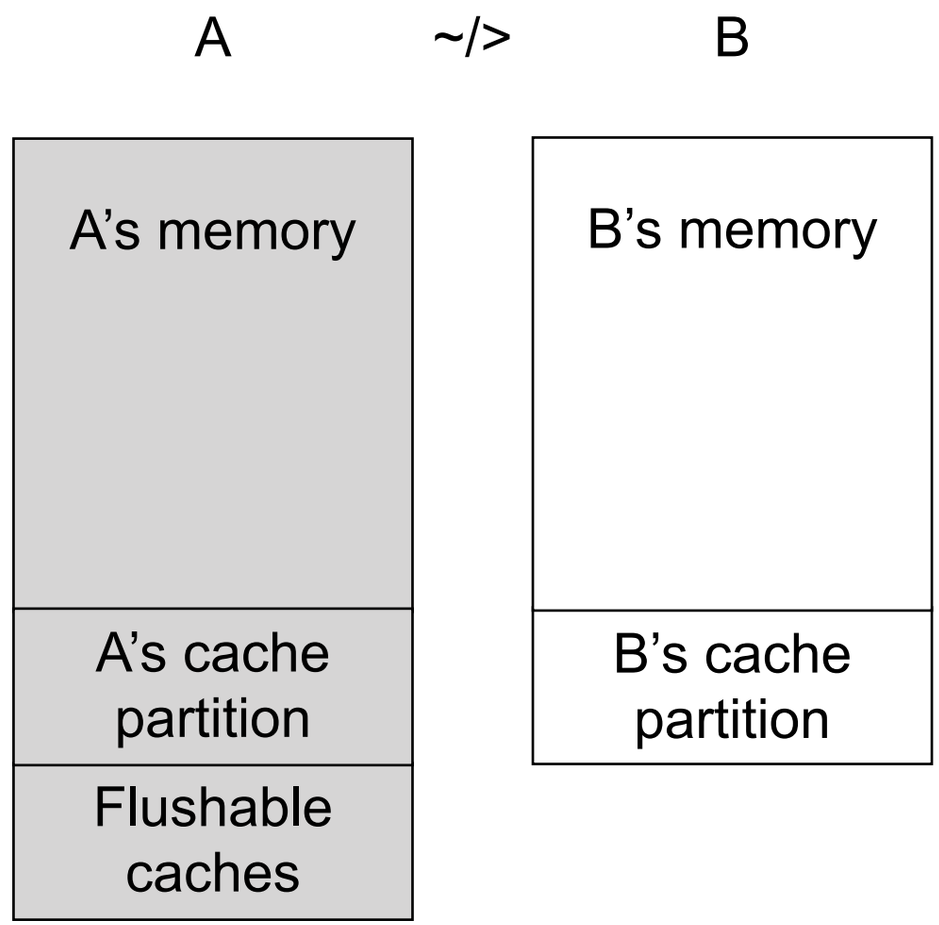


Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

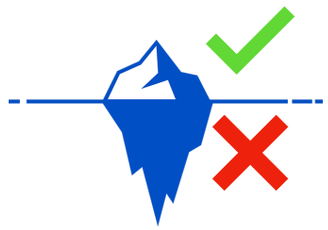
1. Dynamic policy: A ~> B ?

✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

✗ Otherwise, no channel.



# Dynamic policy, observer relativity



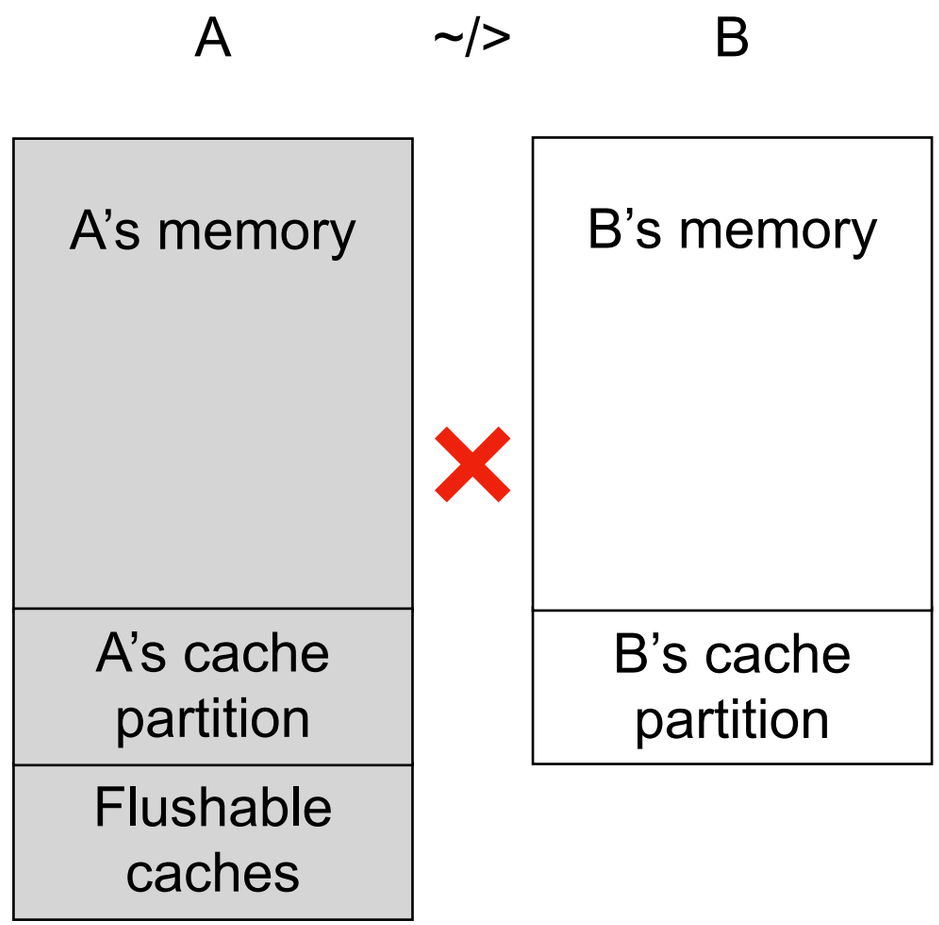
Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

1. Dynamic policy: A ~> B ?

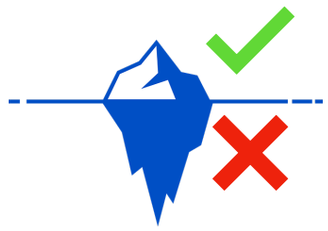
✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

✗ Otherwise, no channel.

- Example:



# Dynamic policy, observer relativity



Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

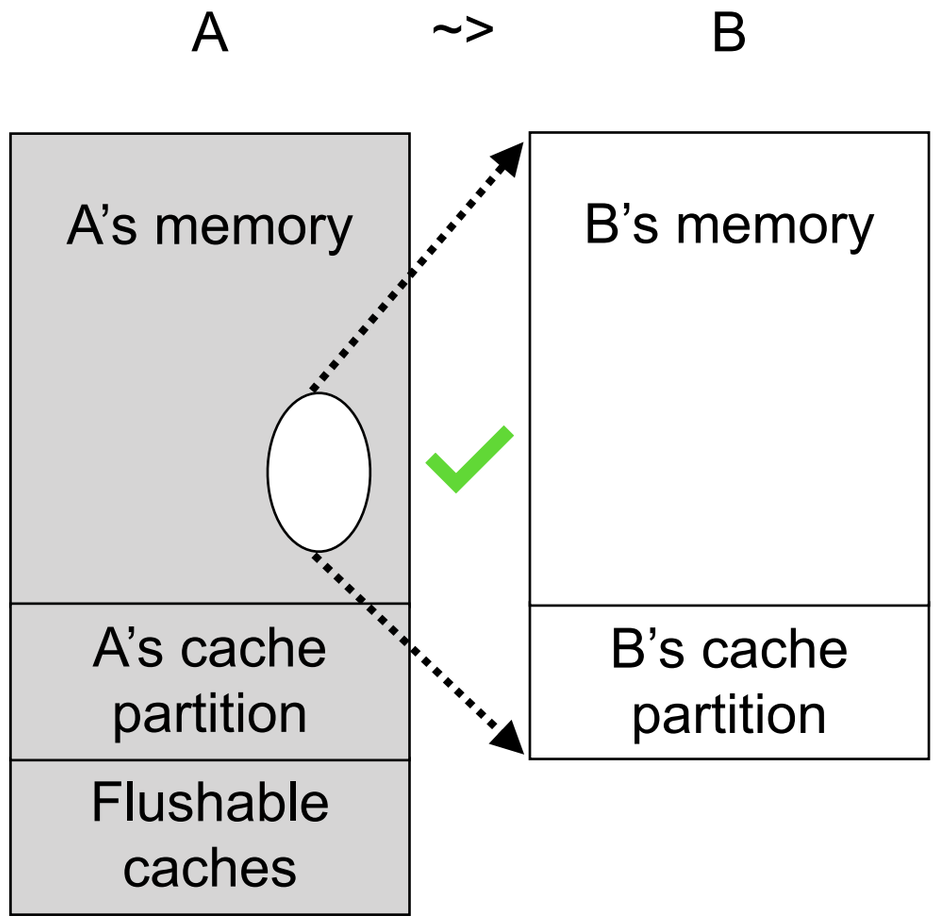
1. Dynamic policy: A ~> B ?

✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

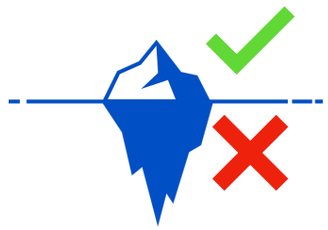
✗ Otherwise, no channel.

- Example:

1. A calls **Subscribe(B)**



# Dynamic policy, observer relativity

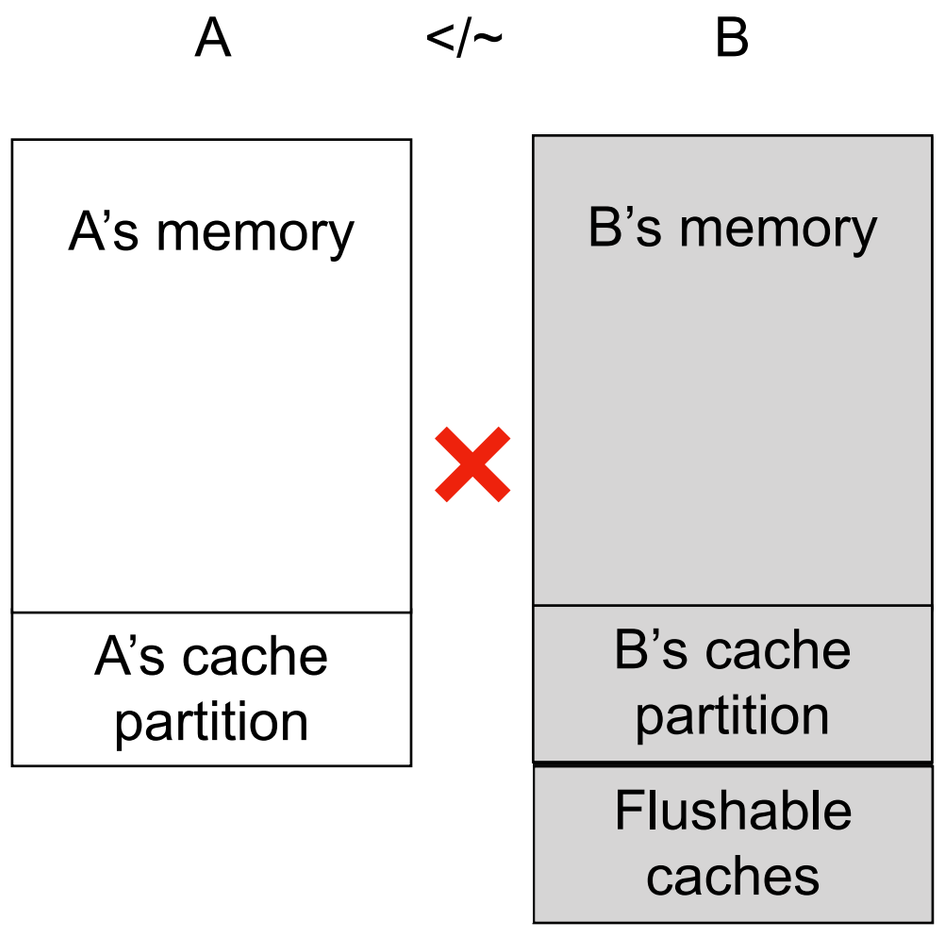


Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

1. Dynamic policy: A ~> B ?

✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

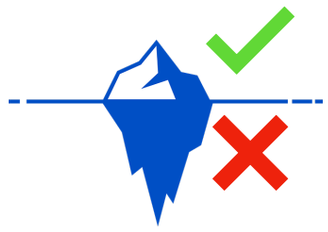
✗ Otherwise, no channel.




- Example:

1. A calls **Subscribe(B)**

# Dynamic policy, observer relativity

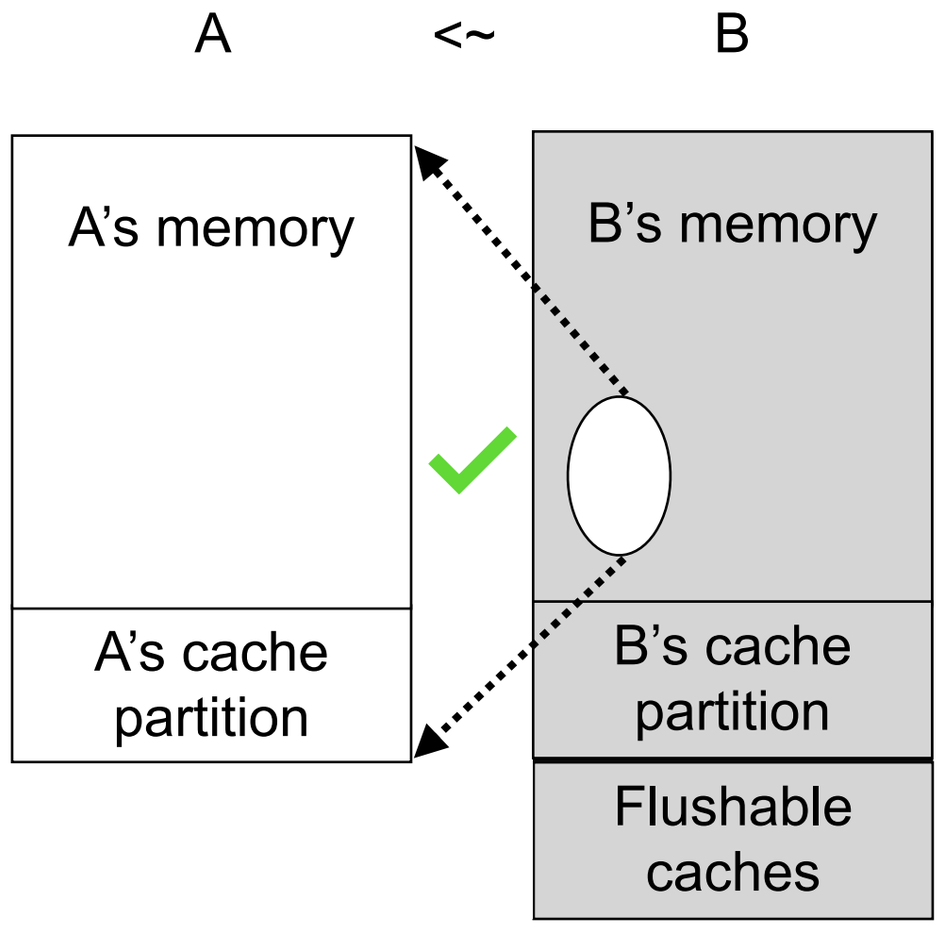


Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

1. Dynamic policy: A ~> B ?

✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

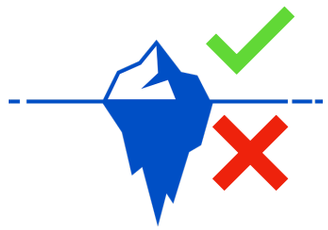
✗ Otherwise, no channel.




- Example:

1. A calls **Subscribe(B)**
2. B calls **Broadcast()**

# Dynamic policy, observer relativity

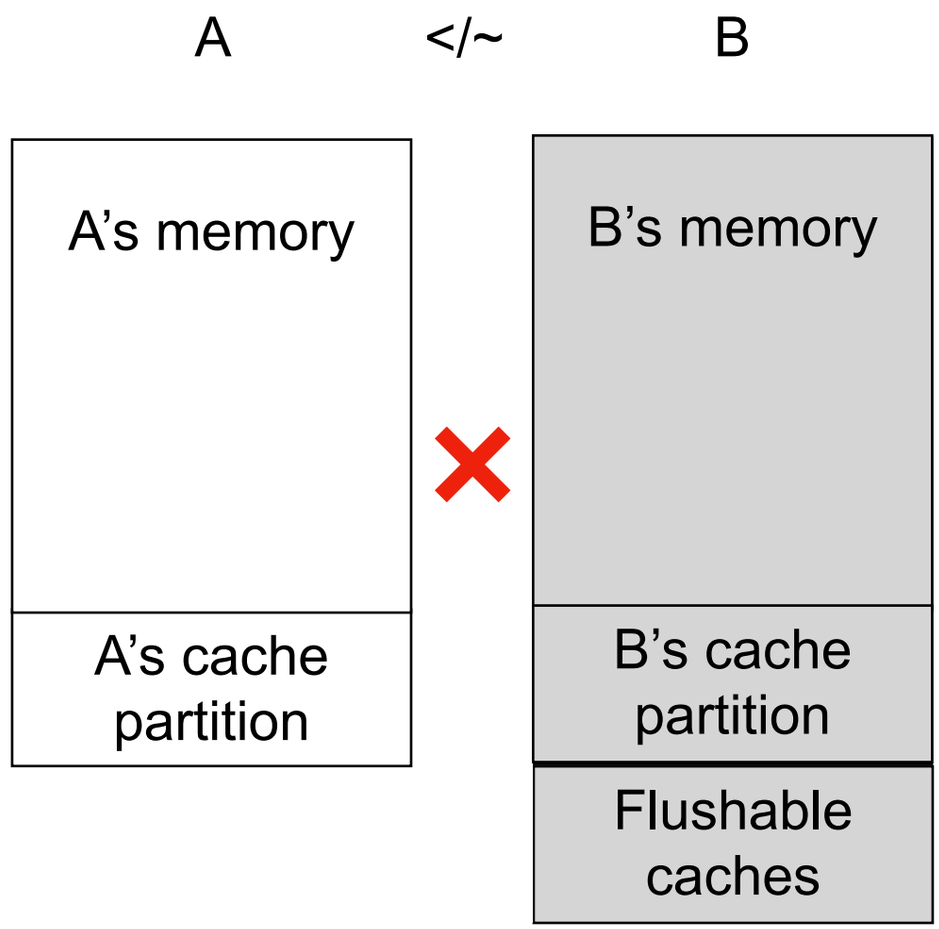


Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

1. Dynamic policy: A ~> B ?

✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

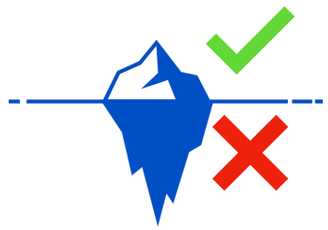
✗ Otherwise, no channel.




- Example:

1. A calls **Subscribe(B)**
2. B calls **Broadcast()**

# Dynamic policy, observer relativity

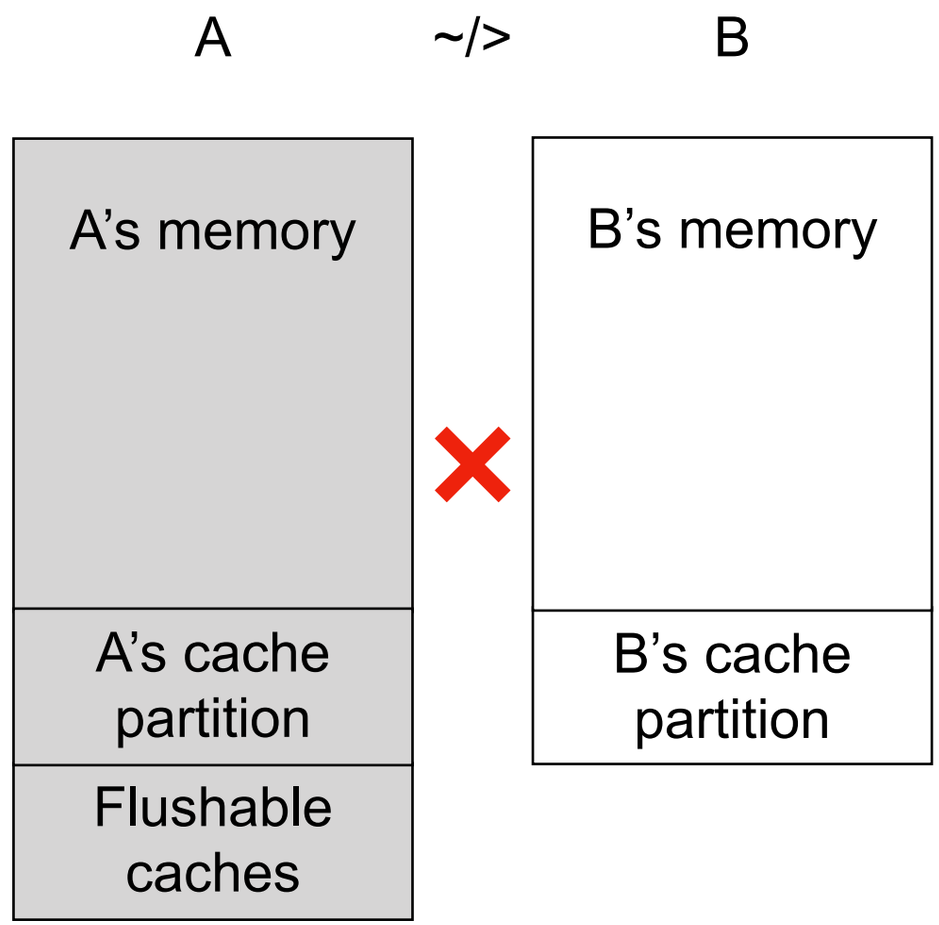


Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

1. Dynamic policy: A ~> B ?

✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

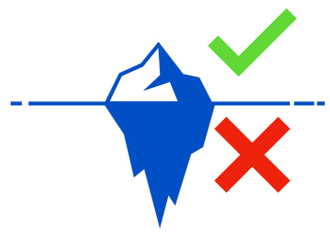
✗ Otherwise, no channel.




- Example:

1. A calls **Subscribe(B)**
2. B calls **Broadcast()**

# Dynamic policy, observer relativity



Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

1. Dynamic policy: A ~> B ?

✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

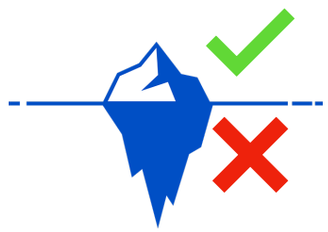
✗ Otherwise, no channel.

2. Property must be observer relative!

- Example:

    1. A calls **Subscribe(B)**
    2. B calls **Broadcast()**

# Dynamic policy, observer relativity



Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

1. Dynamic policy: A ~> B ?

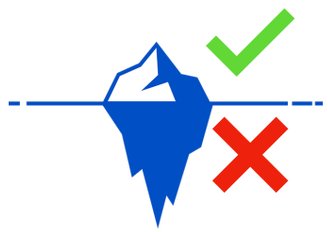
✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

✗ Otherwise, no channel.

2. Property must be observer relative!
   - If not, can't prove the (bisimulation) property for unrelated user C!
     (i.e. where A ~/> C, B ~/> C)

- Example:
  1. A calls **Subscribe(B)**
  2. B calls **Broadcast()**

# Dynamic policy, observer relativity



Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

1. Dynamic policy: A ~> B ?

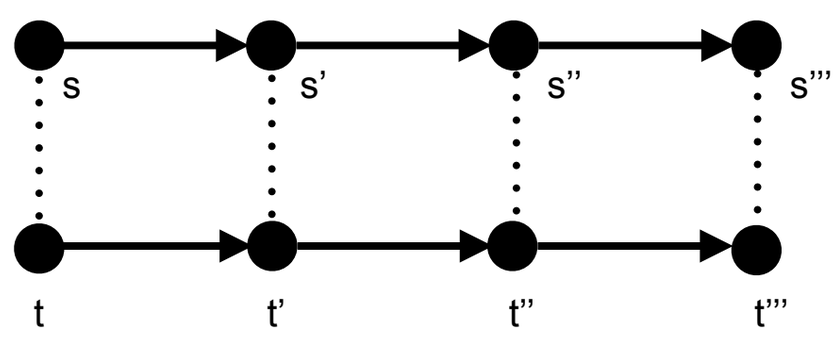
✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

✗ Otherwise, no channel.

2. Property must be observer relative!

- If not, can't prove the (bisimulation) property for unrelated user C!
  (i.e. where A ~/> C, B ~/> C)

Successful bisimulation:

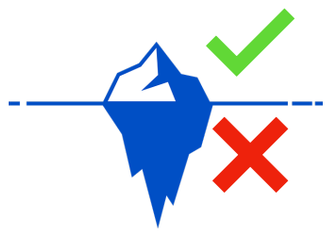



As seen by C

- Example:

1. A calls **Subscribe(B)**
2. B calls **Broadcast()**

# Dynamic policy, observer relativity



Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

1. Dynamic policy: A ~> B ?

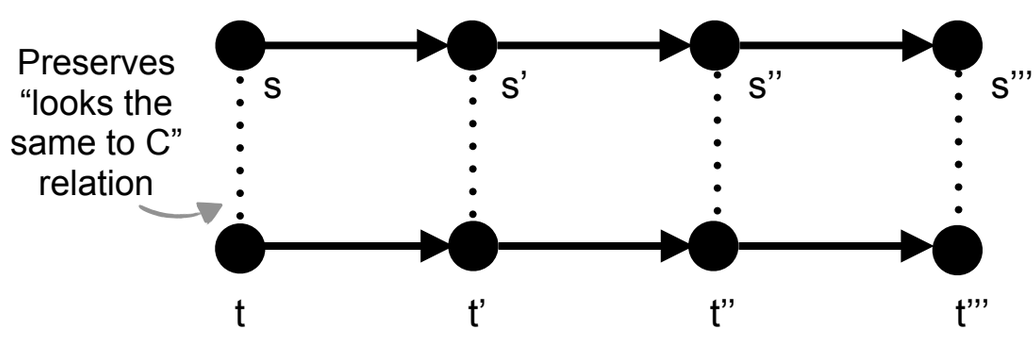
✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

✗ Otherwise, no channel.

- Example:
  1. A calls **Subscribe(B)**
  2. B calls **Broadcast()**

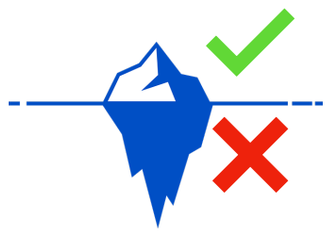
2. Property must be observer relative!
   - If not, can't prove the (bisimulation) property for unrelated user C!
     (i.e. where A ~/> C, B ~/> C)

Successful bisimulation:




As seen by C

# Dynamic policy, observer relativity



Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

1. Dynamic policy: A ~> B ?

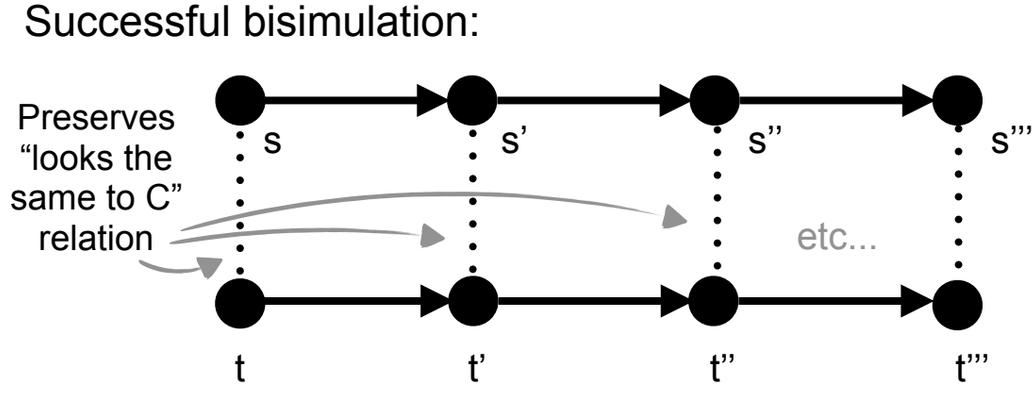
✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

✗ Otherwise, no channel.

2. Property must be observer relative!

- If not, can't prove the (bisimulation) property for unrelated user C!
  (i.e. where A ~/> C, B ~/> C)

Successful bisimulation:

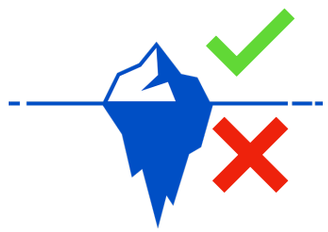



As seen by C

- Example:

1. A calls **Subscribe(B)**
2. B calls **Broadcast()**

# Dynamic policy, observer relativity



Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

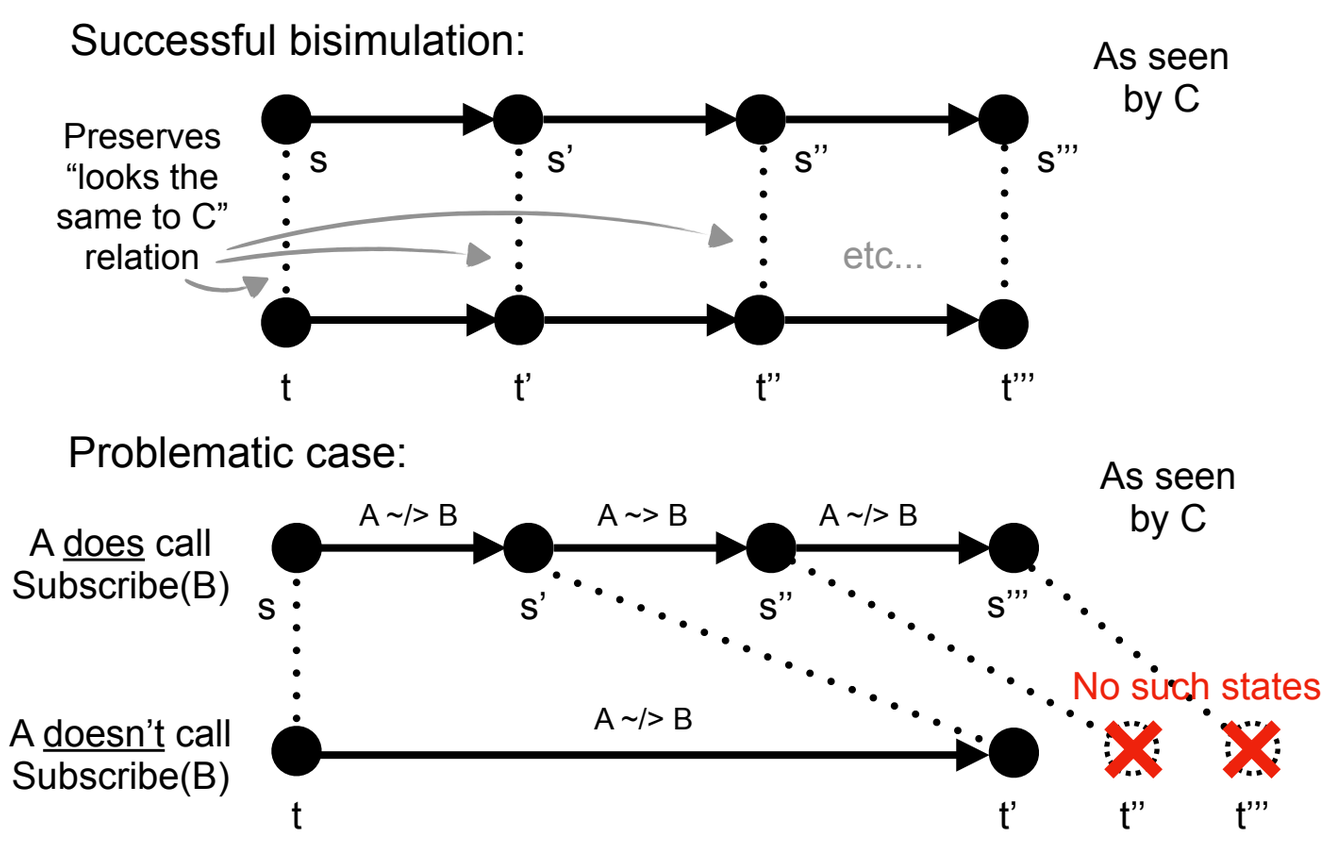
1. Dynamic policy: A ~> B ?

✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

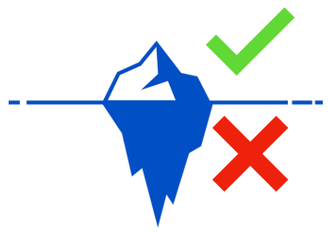
✗ Otherwise, no channel.

- Example:
  1. A calls **Subscribe(B)**
  2. B calls **Broadcast()**

2. Property must be observer relative!
   - If not, can't prove the (bisimulation) property for unrelated user C!
     (i.e. where A ~/> C, B ~/> C)



# Dynamic policy, observer relativity



Two basic system calls:
**Subscribe(*d*)**, **Broadcast()**

1. Dynamic policy: A ~> B ?

✓ Only when A calls
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

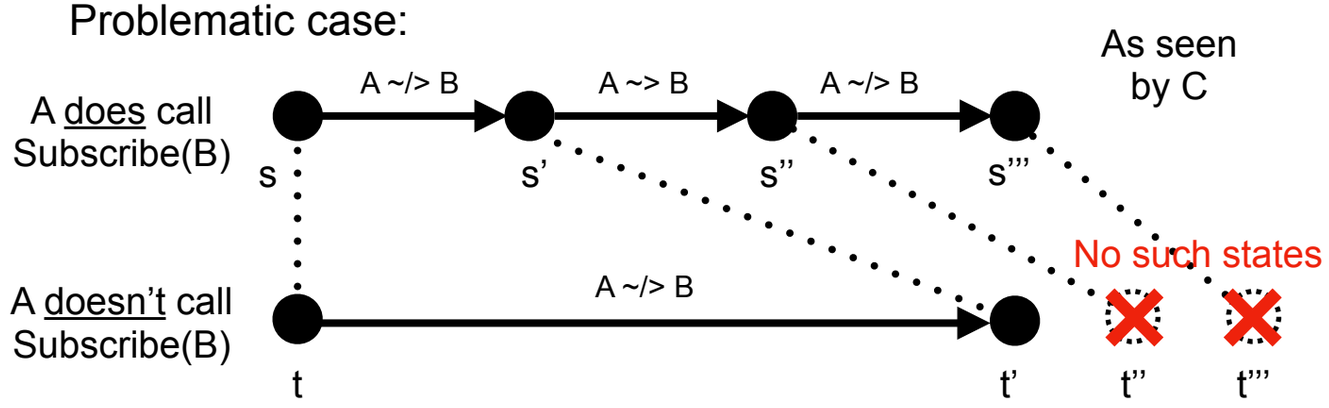
✗ Otherwise, no channel.

- Example:
  1. A calls **Subscribe(B)**
  2. B calls **Broadcast()**

2. Property must be observer relative!

- If not, can't prove the (bisimulation) property for unrelated user C!
  (i.e. where A ~/> C, B ~/> C)

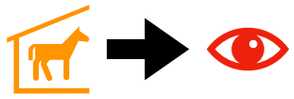
Problematic case:



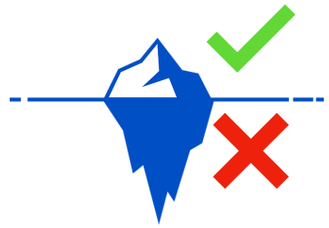

- *Solution*: C's property must treat *states* (in the state machine) as observable only whenever
  - C is running, or
  - When *d* is running, *d* ~> C.

# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy




Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.



Make security property precise enough to exclude flows from covert state.


Demonstrating these principles,
we formalised in Isabelle/HOL:

1. OS security model imposing requirements on relevant parts of OS.
(Intended for seL4, but *generic*)

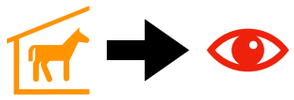
2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

3. Proof our security property holds if OS model's requirements hold.

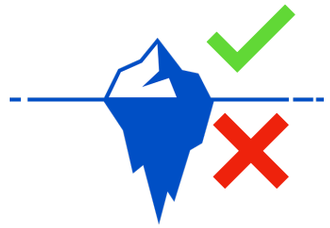
4. Basic instantiation of OS model exercising dynamic policy.

# How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy




Abstract *covert state* + *time* to reflect strategies enabled by HW:
Partition or flush state; pad time.



Make security property precise enough to exclude flows from covert state.

**Thank you!**
**Q & A**


Demonstrating these principles, we formalised in Isabelle/HOL:

**Paper:** https://doi.org/jzwj
**Artifact:** https://doi.org/jzwk

1. OS security model imposing requirements on relevant parts of OS.
(Intended for seL4, but *generic*)

2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

3. Proof our security property holds if OS model's requirements hold.

4. Basic instantiation of OS model exercising dynamic policy.