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

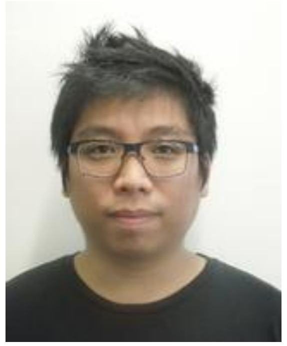
Formal Methods (FM), 7 March 2023

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

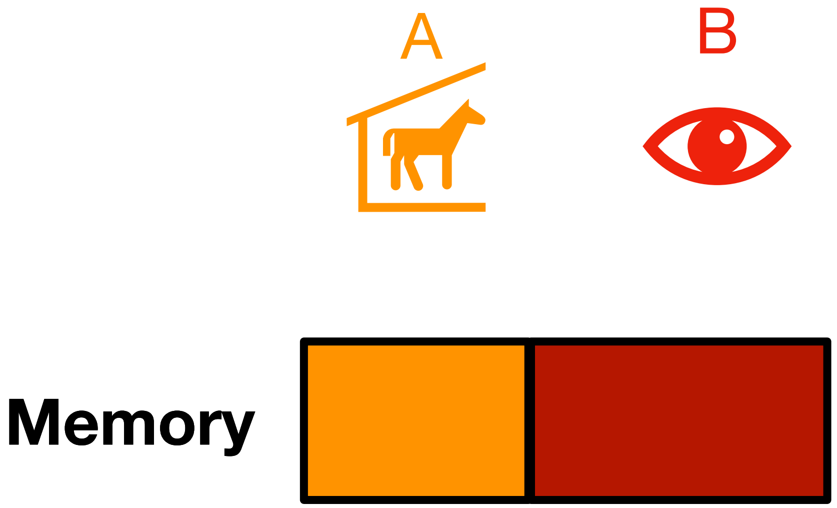
[1] The University of Melbourne, Australia

[2] UNSW Sydney, Australia

[3] Proofcraft, Sydney, Australia

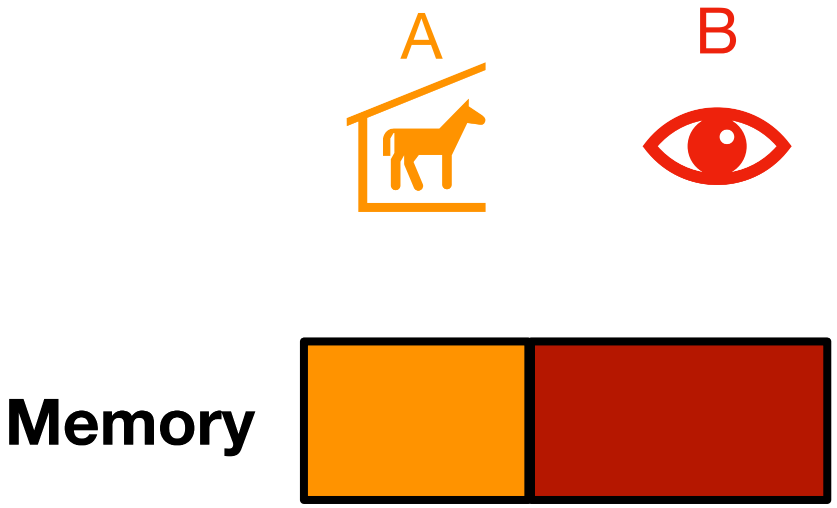


Threat scenario: *Trojan* and *spy*



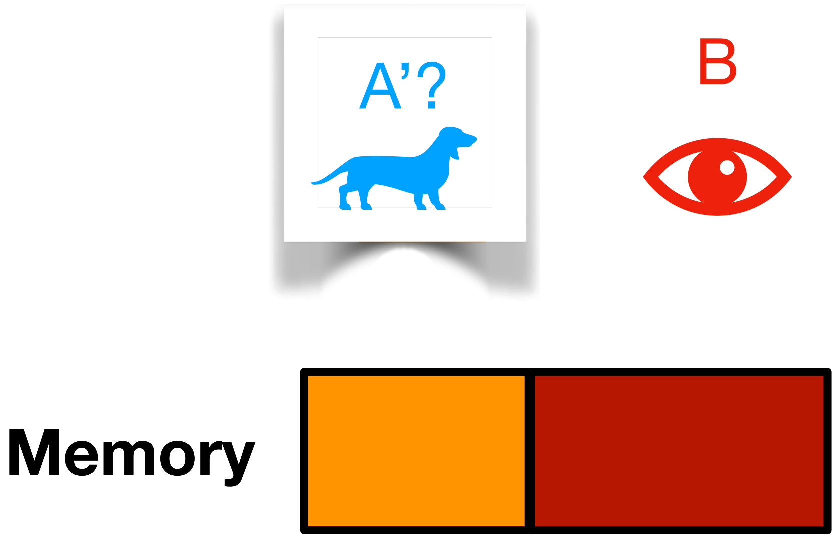
Threat scenario:
Trojan and *spy*

Covert channels



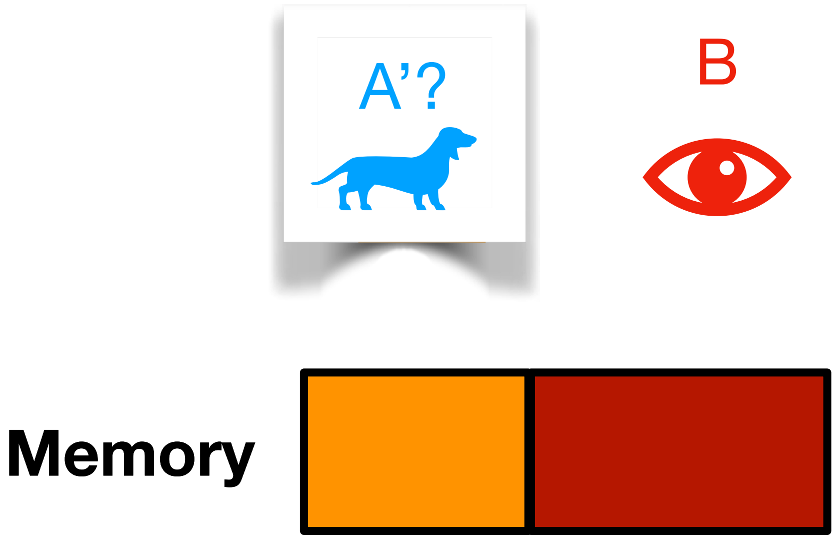
Threat scenario: *Victim/~~Trojan~~* and *spy*?

Covert channels
+
Side channels



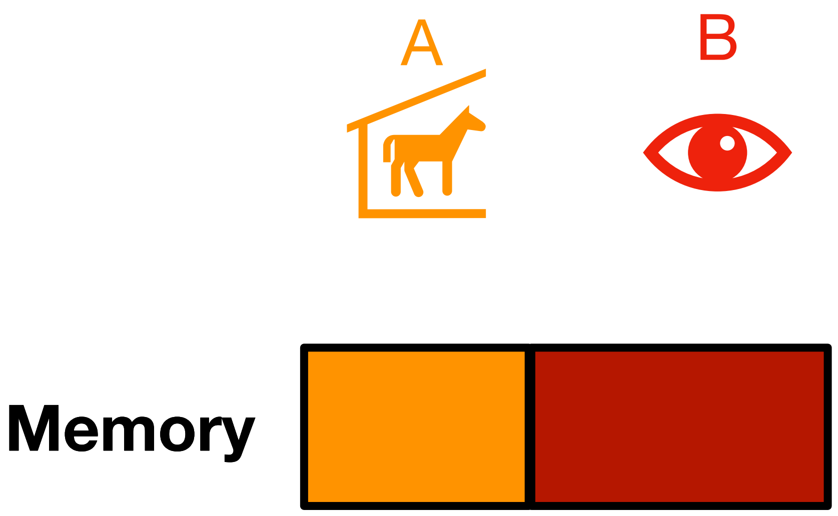
Threat scenario:
Victim/~~Trojan~~ and spy?

Covert channels ✓
+
Side channels ✓



Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels



A

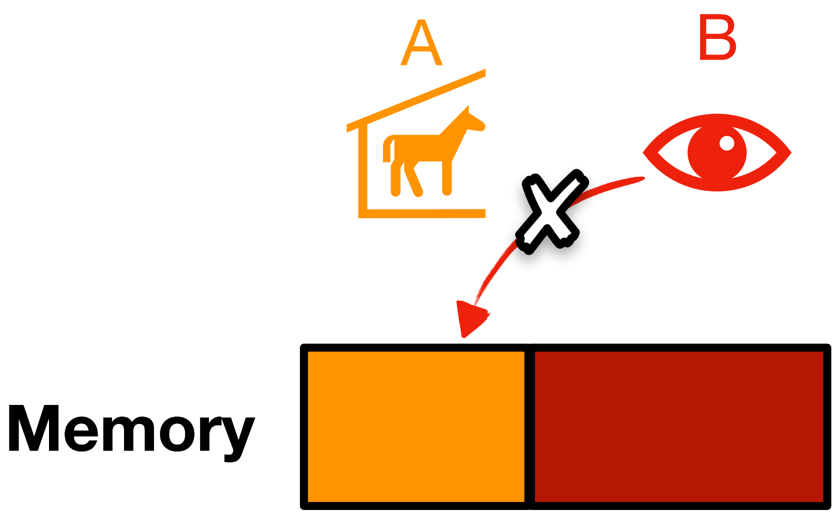
B

Memory

Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

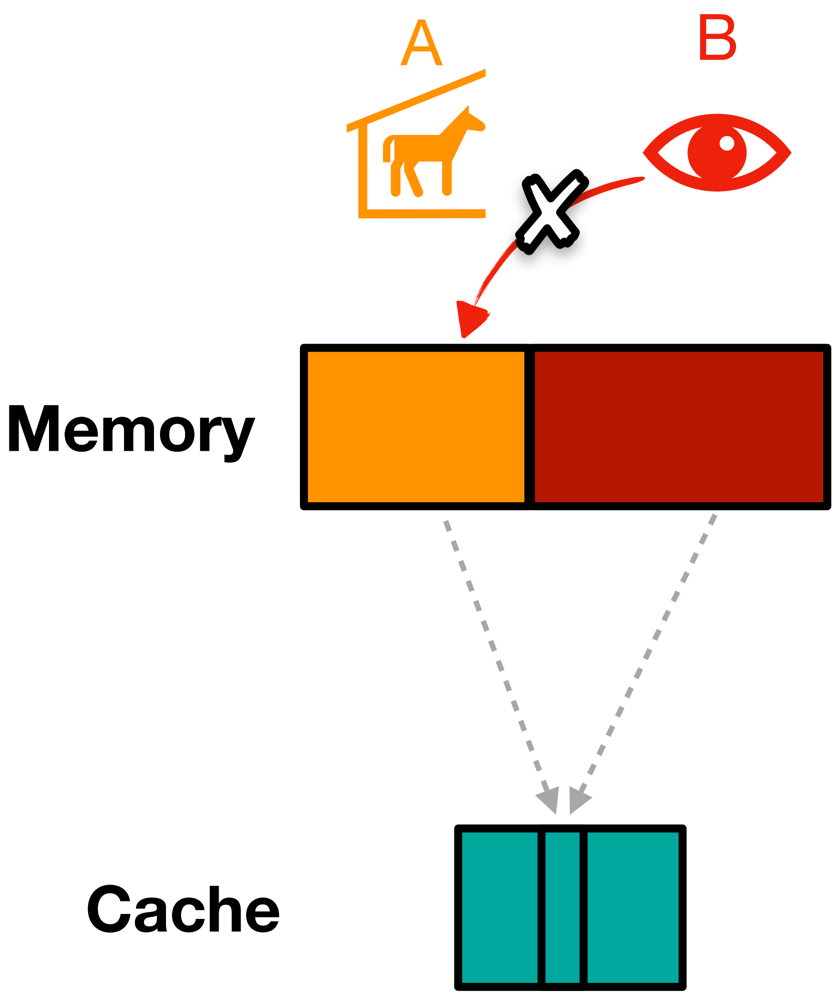
- OSes typically implement *memory protection.*



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.

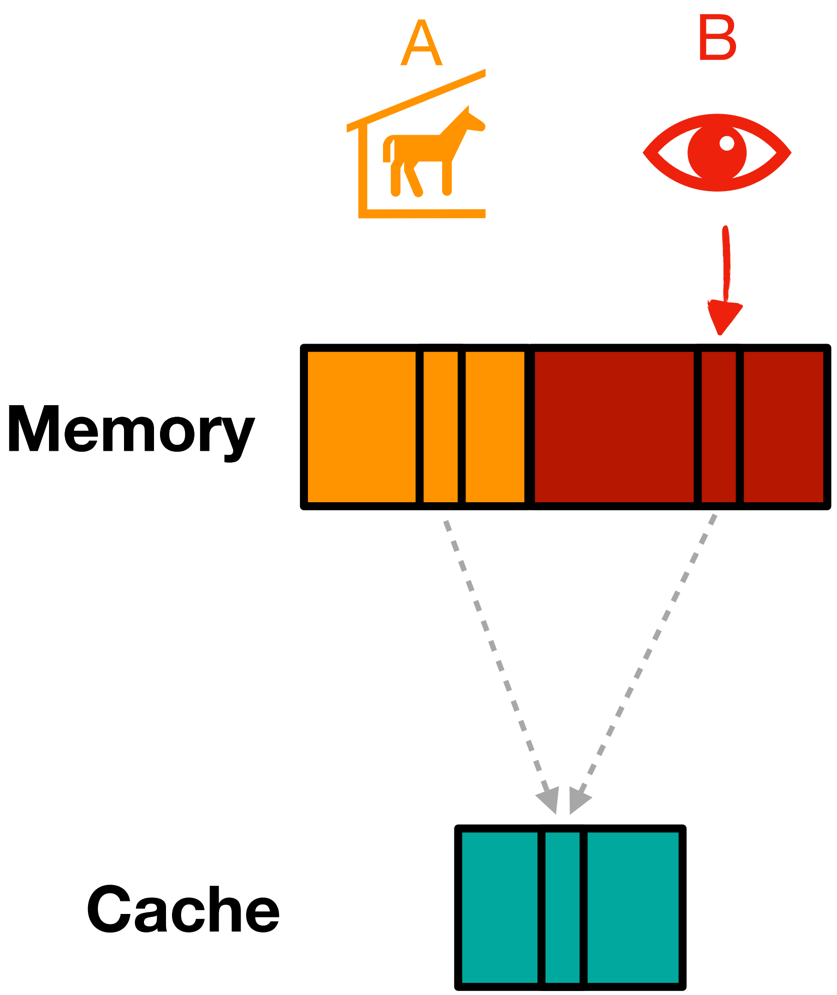


Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

- OSes typically implement *memory protection.*

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

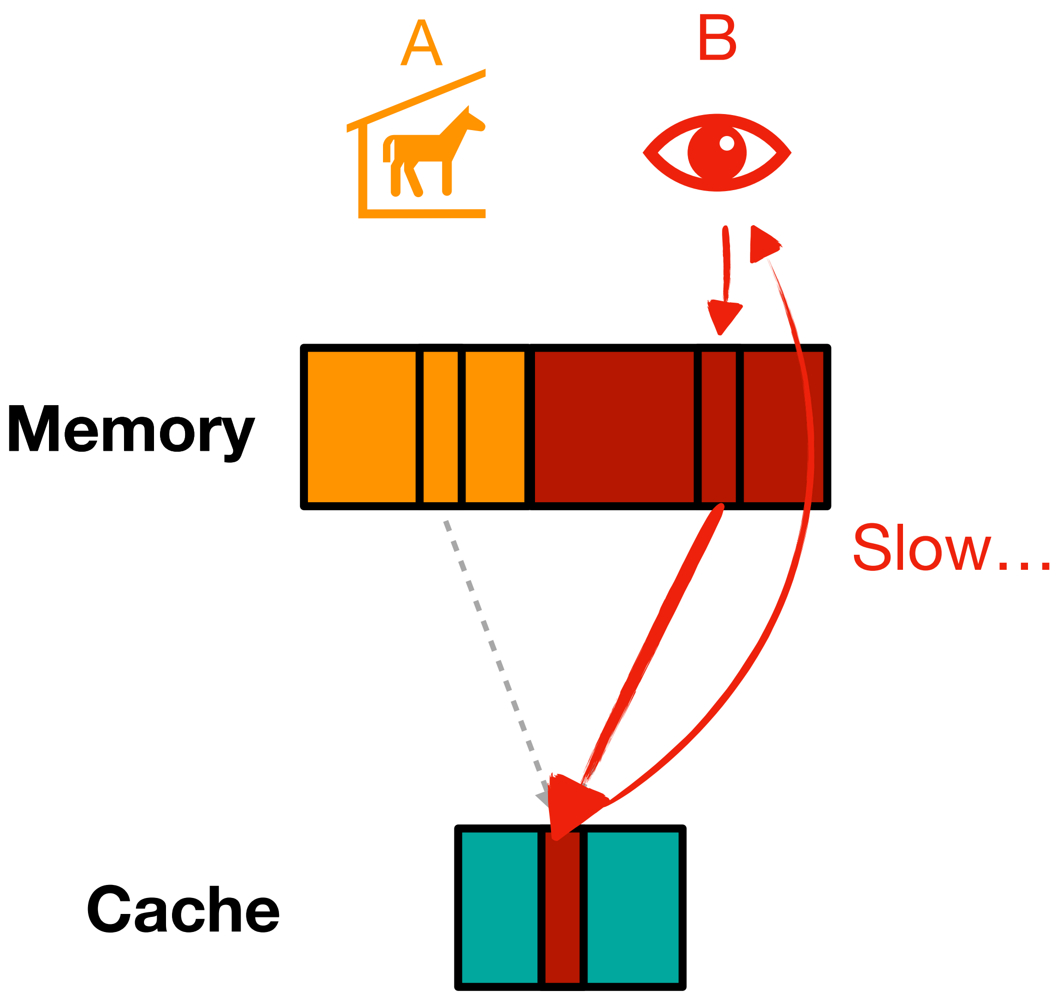


Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

- OSes typically implement *memory protection.*

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

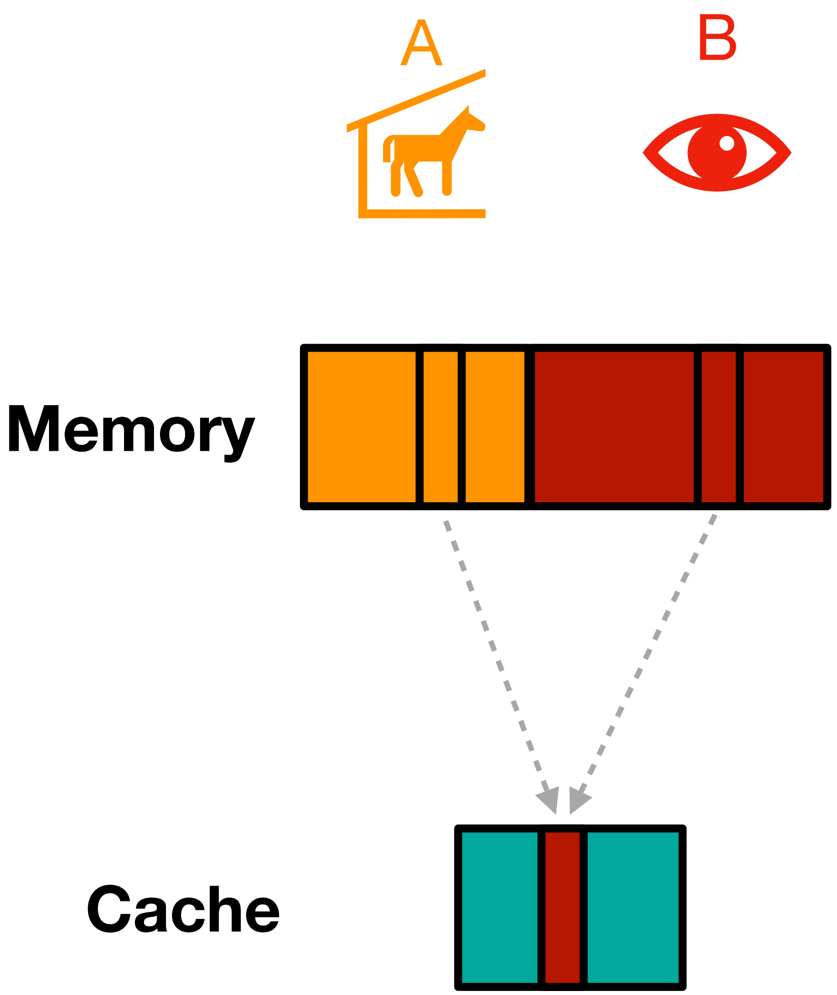


Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

- OSes typically implement *memory protection.*

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

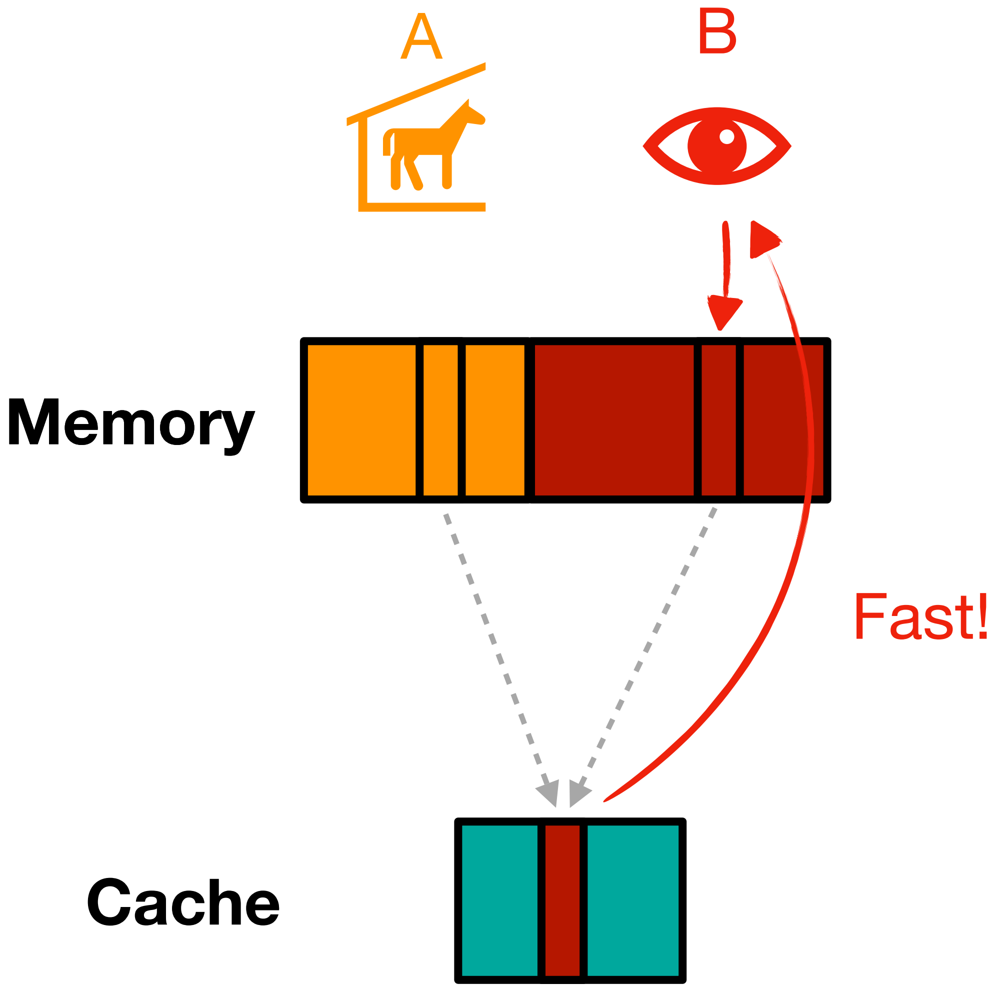


Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

- OSes typically implement *memory protection.*

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

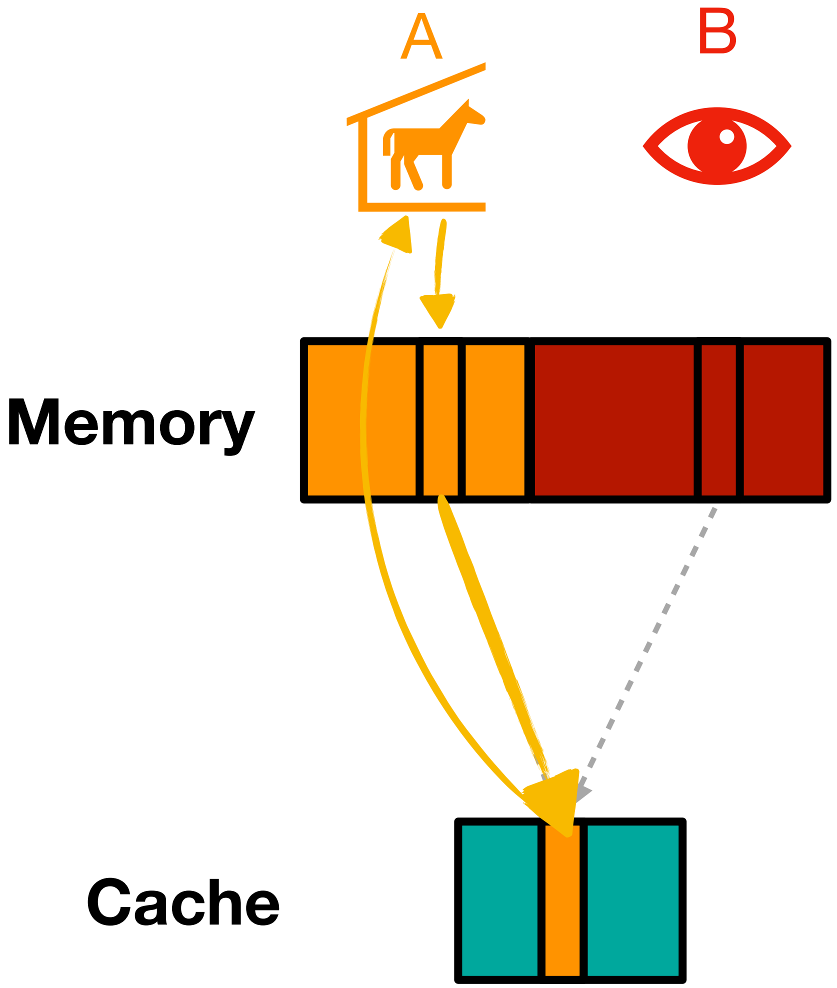


Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

- OSes typically implement *memory protection.*

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

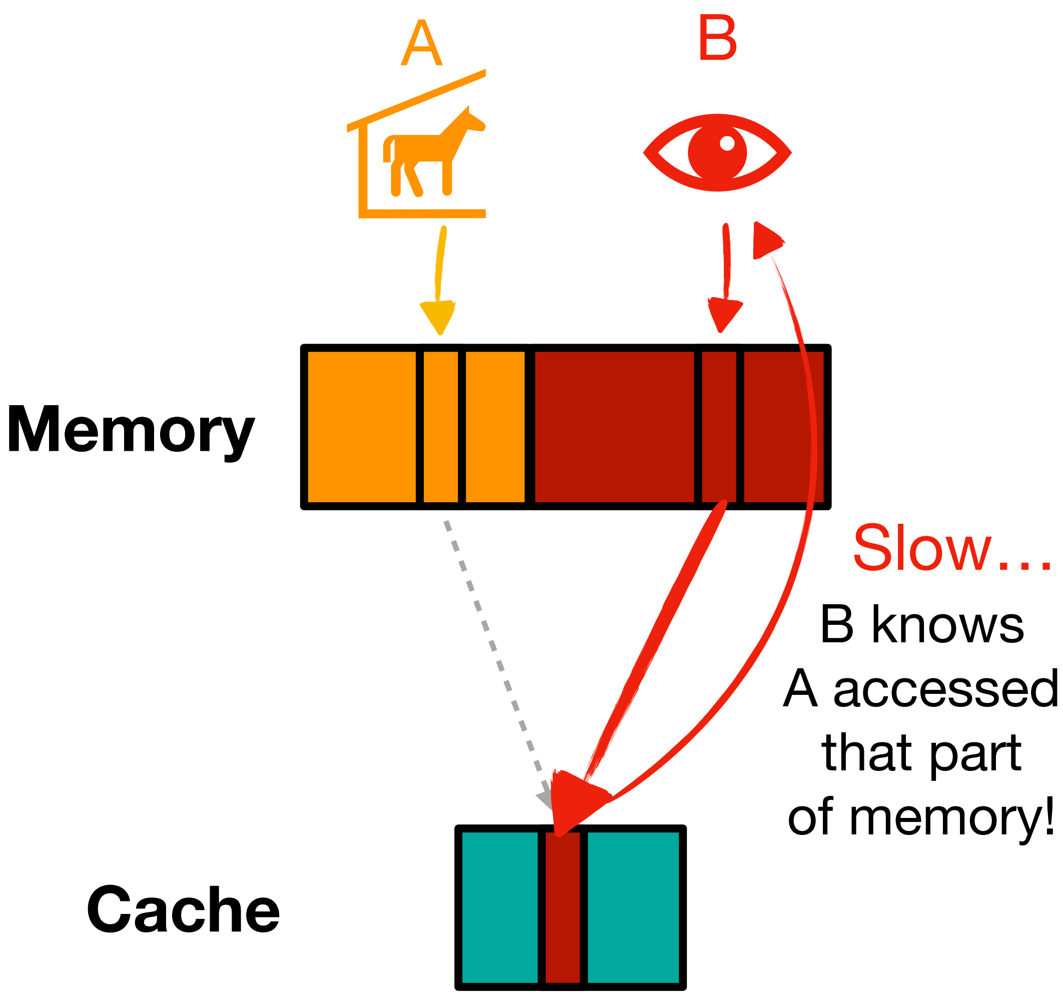


Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

- OSes typically implement *memory protection.*

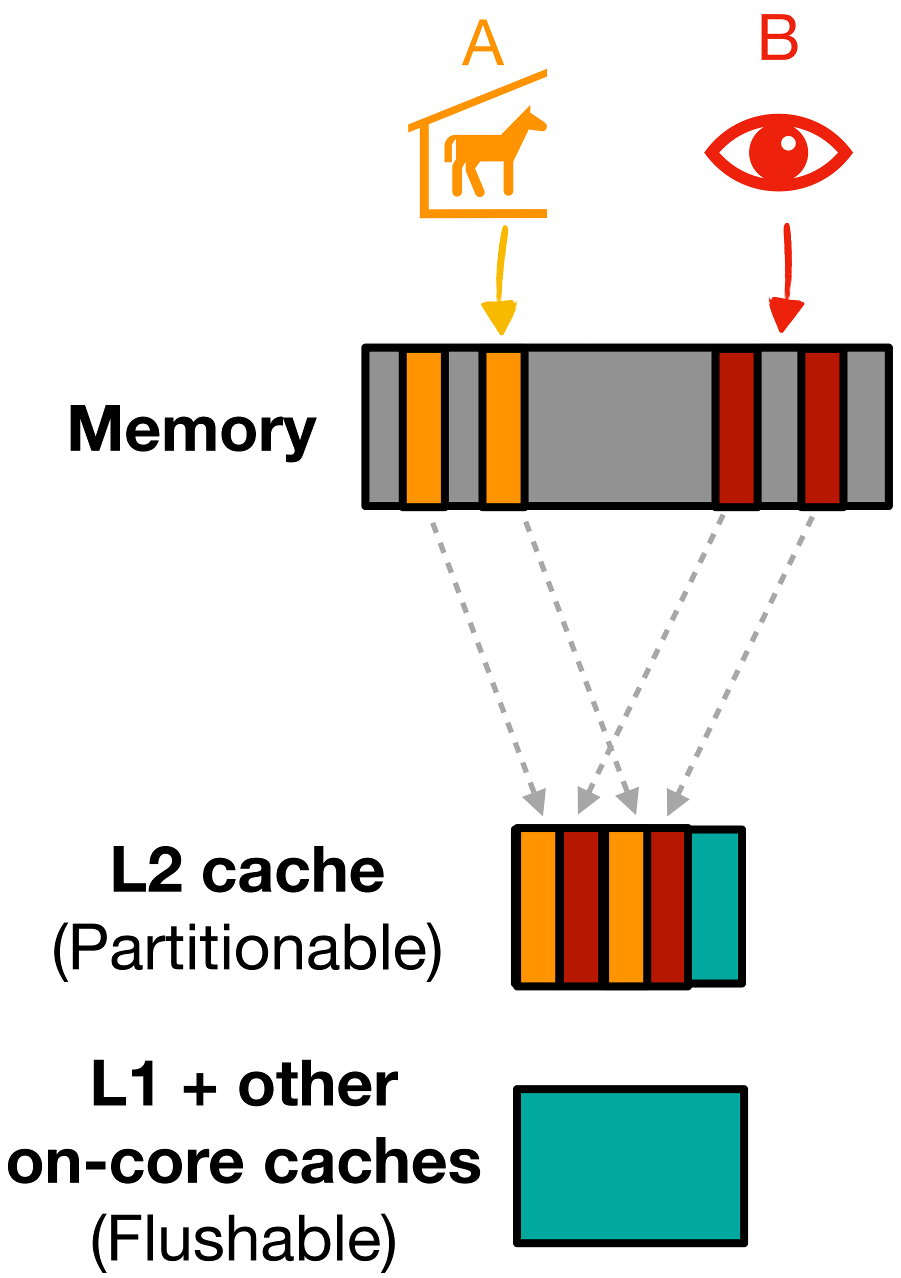
- But: Mere memory access can change the microarch. state — this affects timing.



Threat scenario: *Trojan* and *spy*

Covert channels
+
Side channels

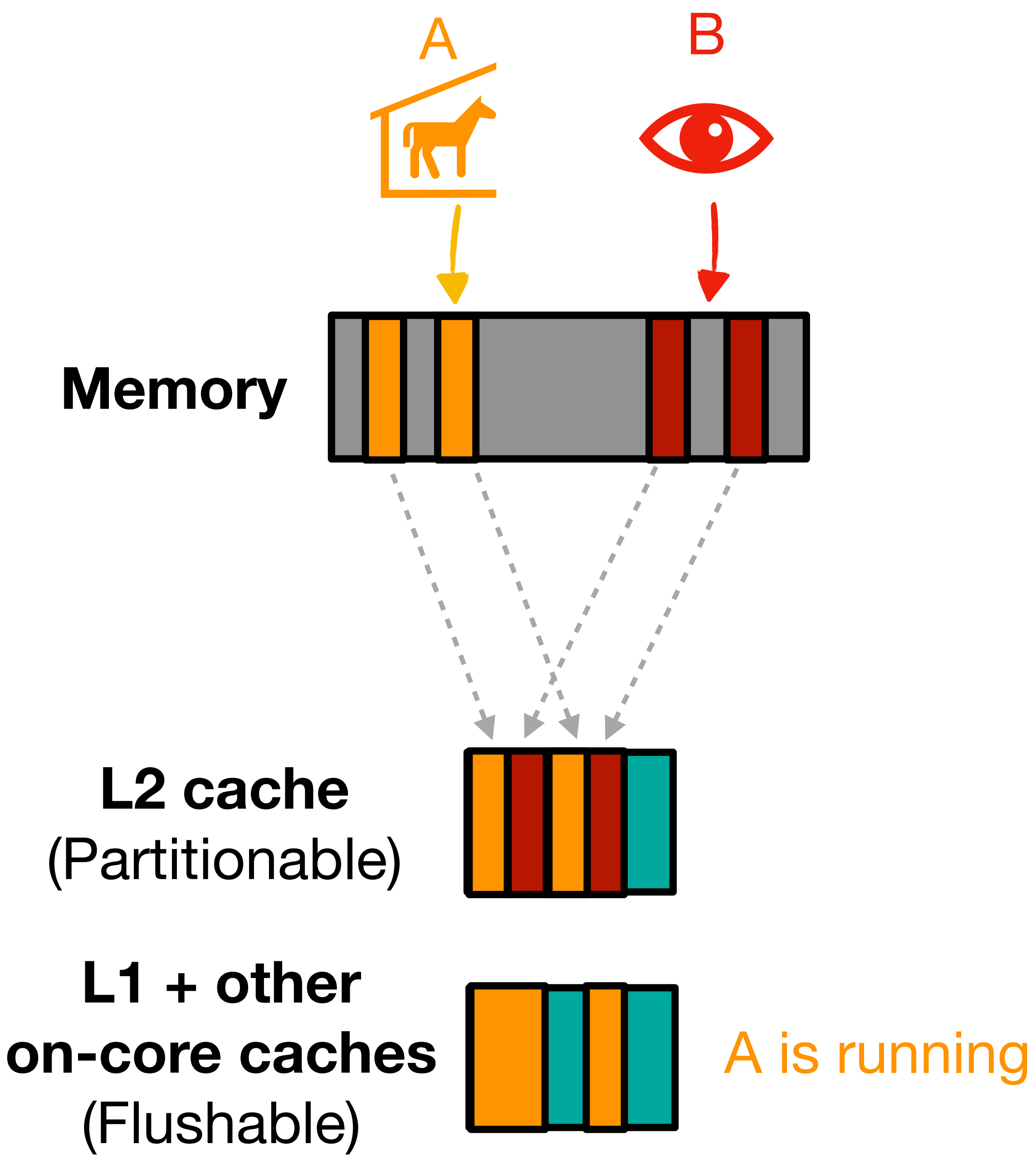
- OSes typically implement *memory protection.*
- But: Mere memory access can change the microarch. state — this affects timing.
- To prevent these *timing channels*, OSes can implement *time protection*:
  e.g. [Ge et al. 2019] for seL4 microkernel OS
  - Partition what we can
  - Flush what we can't



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

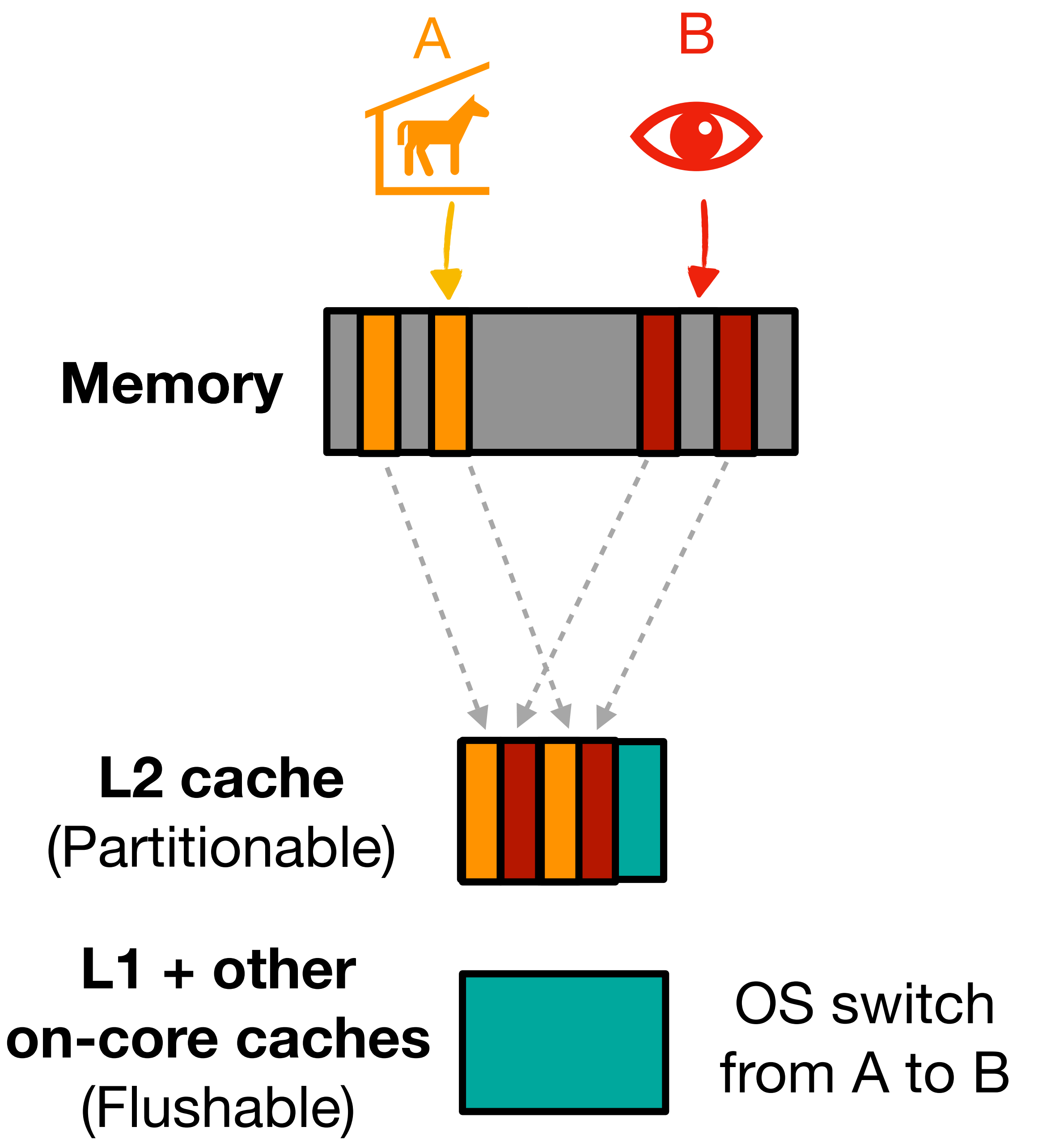
- OSes typically implement *memory protection.*

- 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

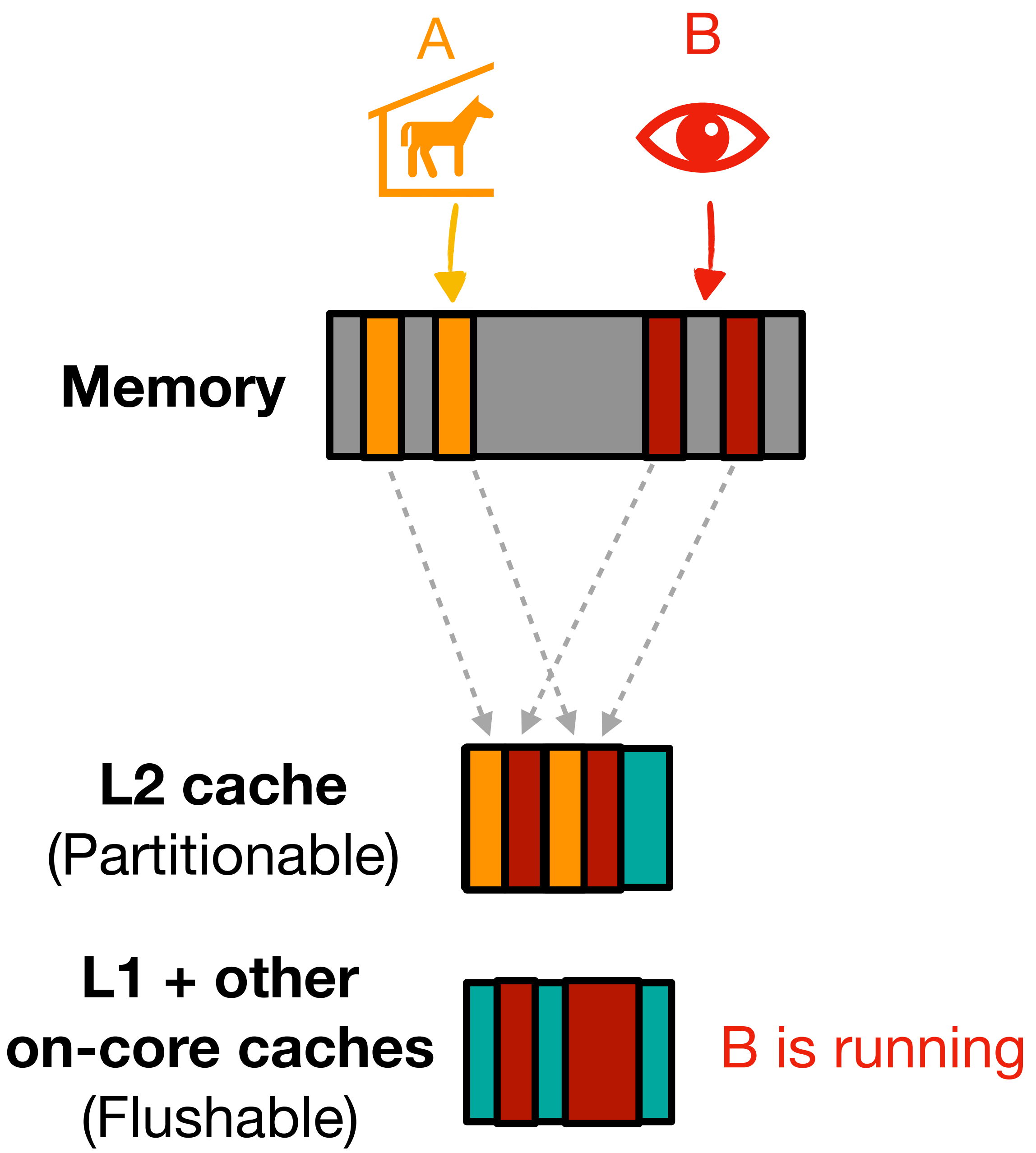


"Flush": Write fixed content; wait up to fixed time.

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

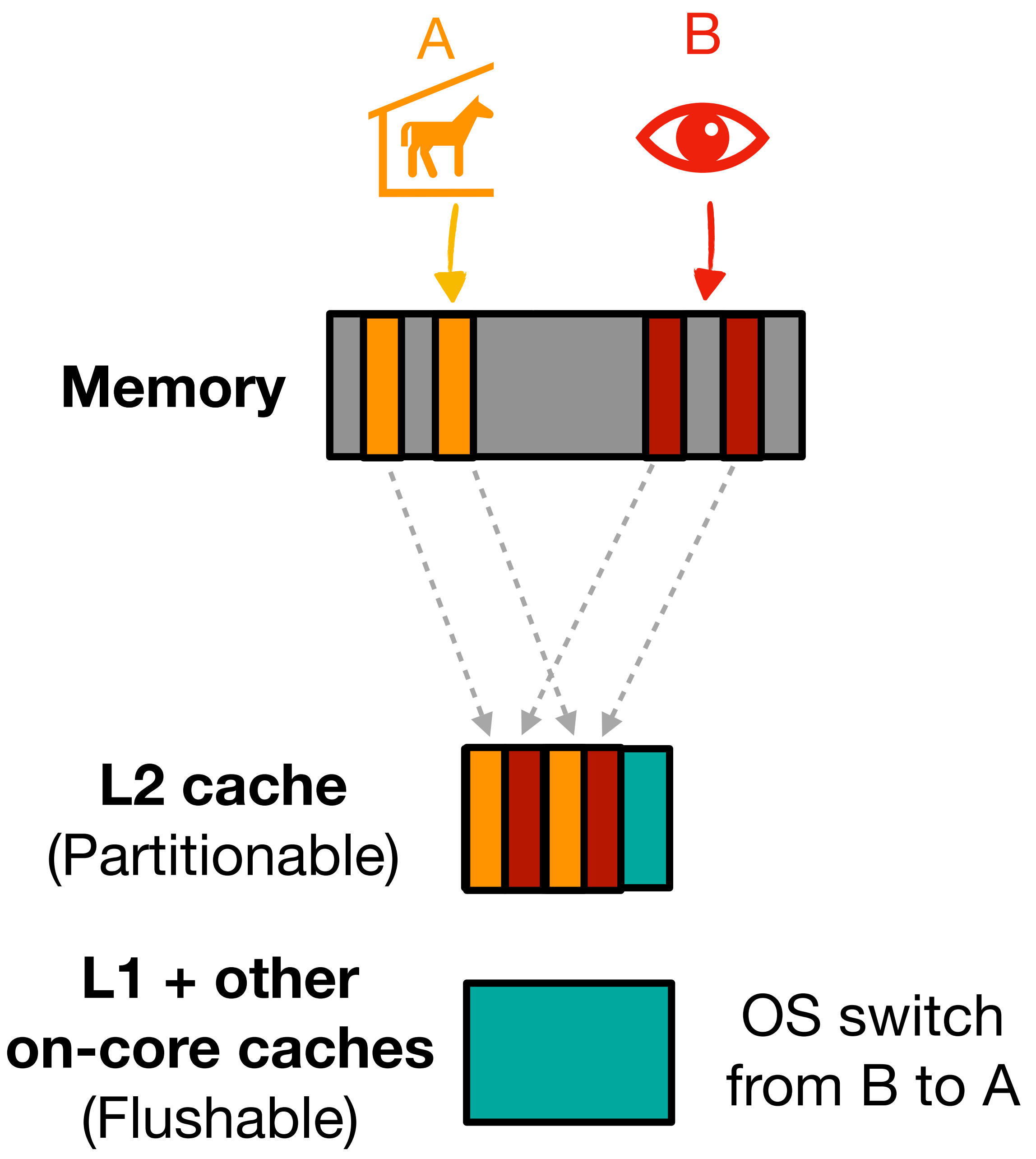


"Flush": Write fixed content; wait up to fixed time.

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



A

B

Memory

L2 cache
(Partitionable)

L1 + other on-core caches
(Flushable)

OS switch from B to A

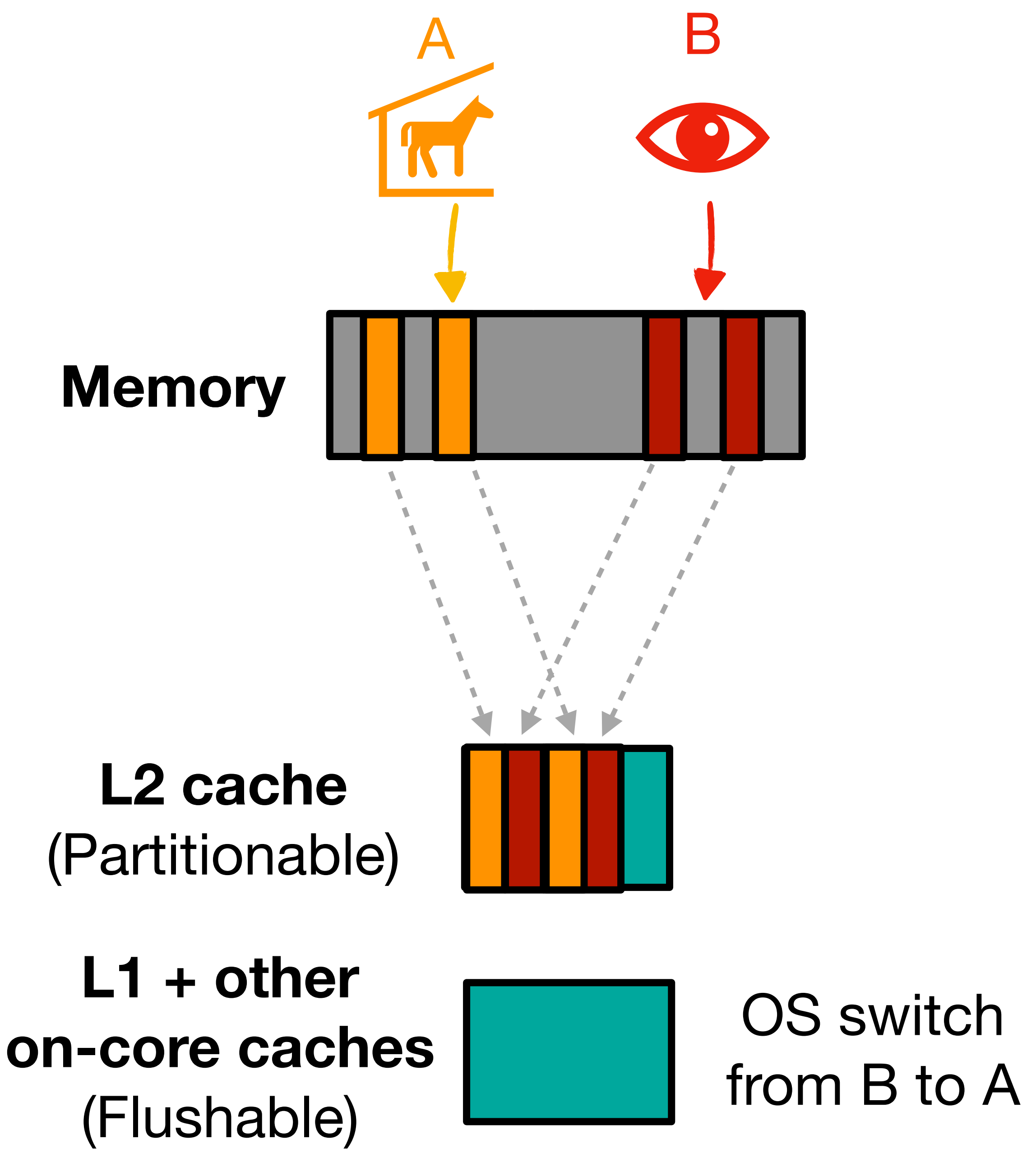
"Flush": Write fixed content; wait up to fixed time.

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



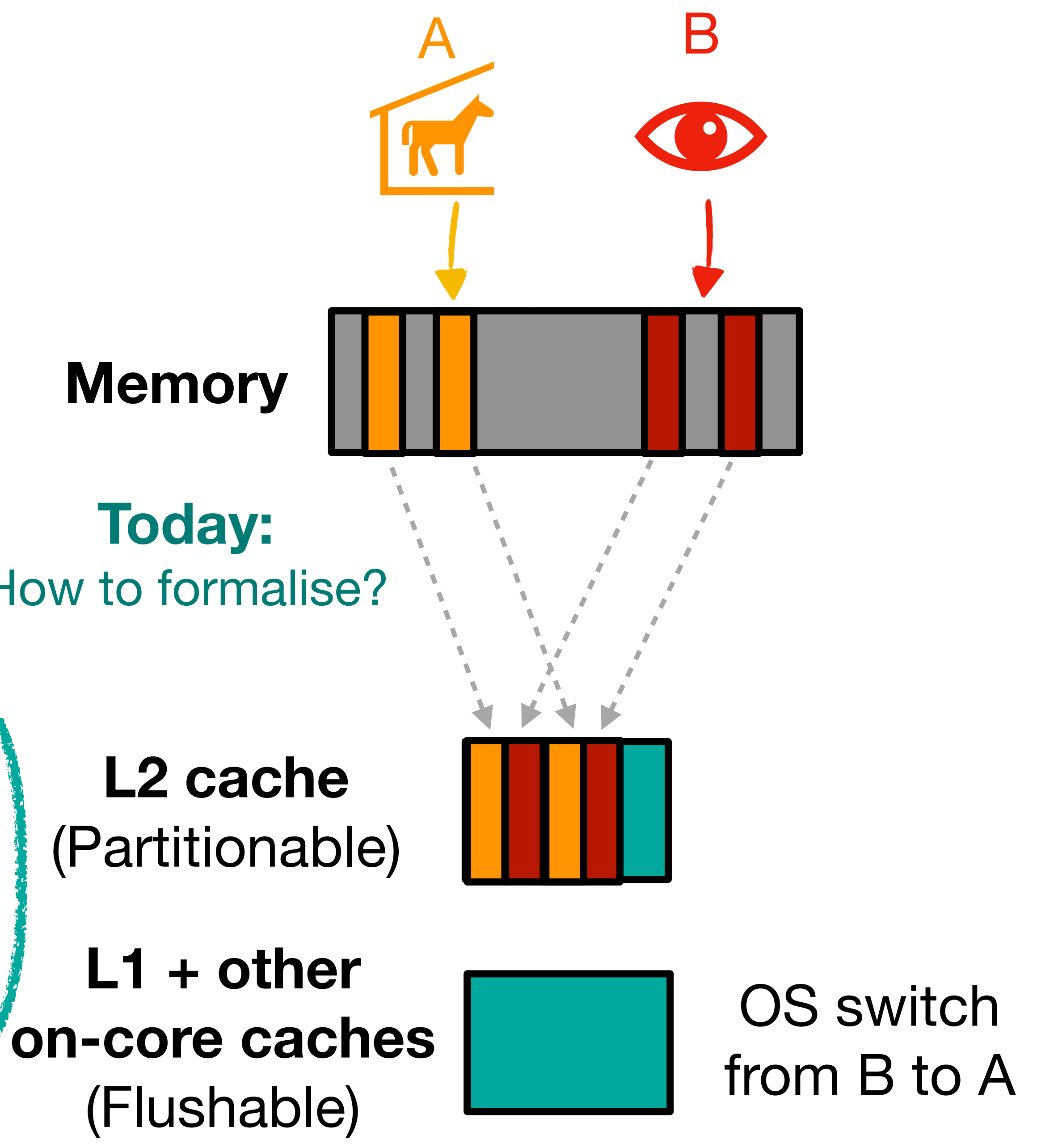
"Flush": Write fixed content; wait up to fixed time.

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



Today:
How to formalise?

"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

🐎 ➡ 👁

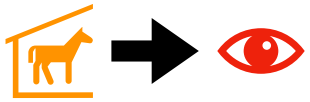
No channels!

OS 🤝 HW

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

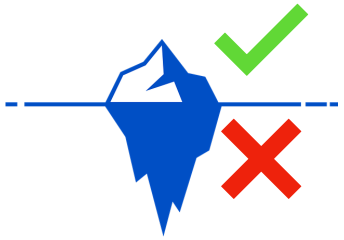
How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy



No
channels!
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.

How to formalise an OS enforces *time protection*?

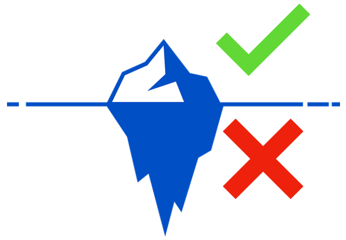
Versus threat scenario:
trojan and spy

🐴 ➡ 👁

No channels!

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:

How to formalise an OS enforces *time protection*?

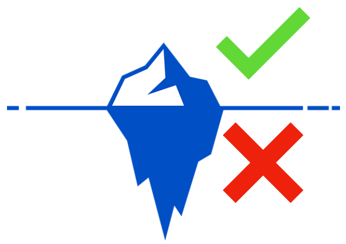
Versus threat scenario:
trojan and spy

🐴 ➡ 👁

No channels!

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*)

How to formalise an OS enforces *time protection*?

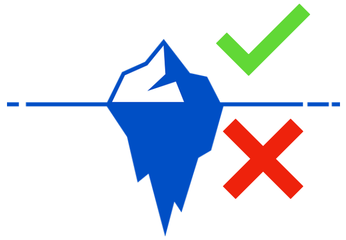
Versus threat scenario:
trojan and spy

🐎 ➡ 👁

No channels!

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*)

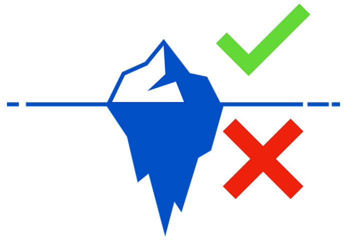
2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

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:

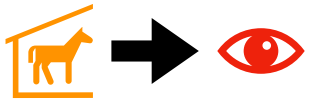
1. OS security model imposing requirements on relevant parts of OS.
(Intended for seL4, but *generic*)

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.

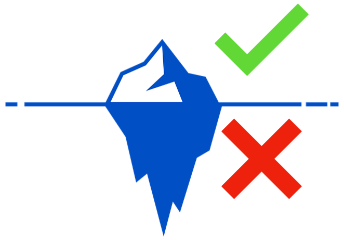
How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy



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*)

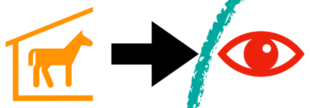
2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

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

4. Basic instantiation of OS model exercising dynamic policy.

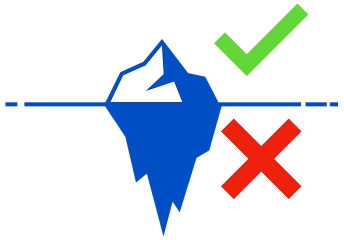
How to formalise an OS enforces *time protection*?

Versus threat scenario: trojan and spy



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:

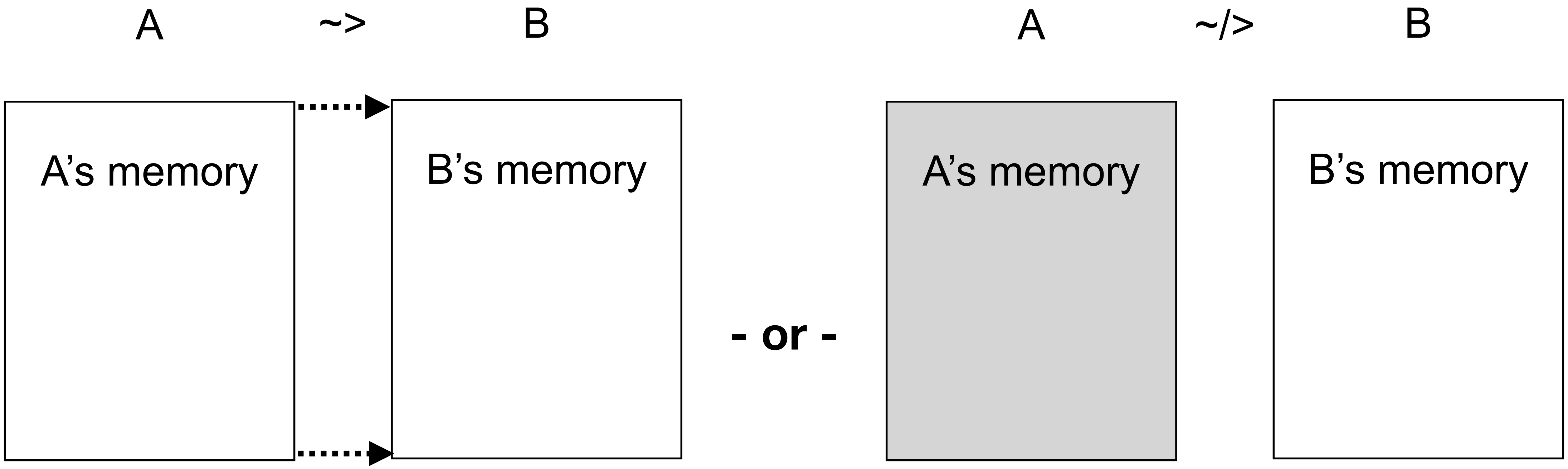
1. OS security model imposing requirements on relevant parts of OS. (Intended for seL4, but *generic*)

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.

Overt vs *covert* state



A ~> B

A's memory

B's memory

- or -

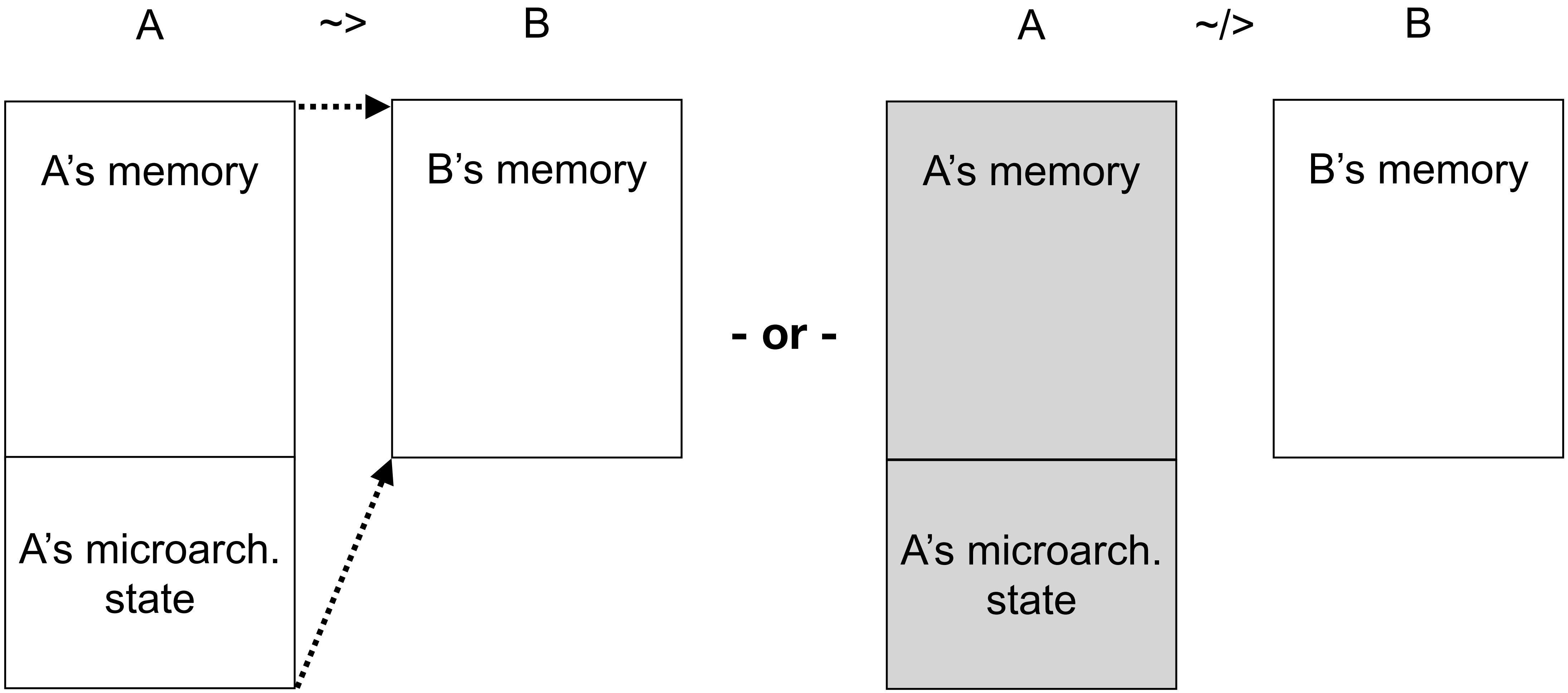
A ~/> B

A's memory

B's memory

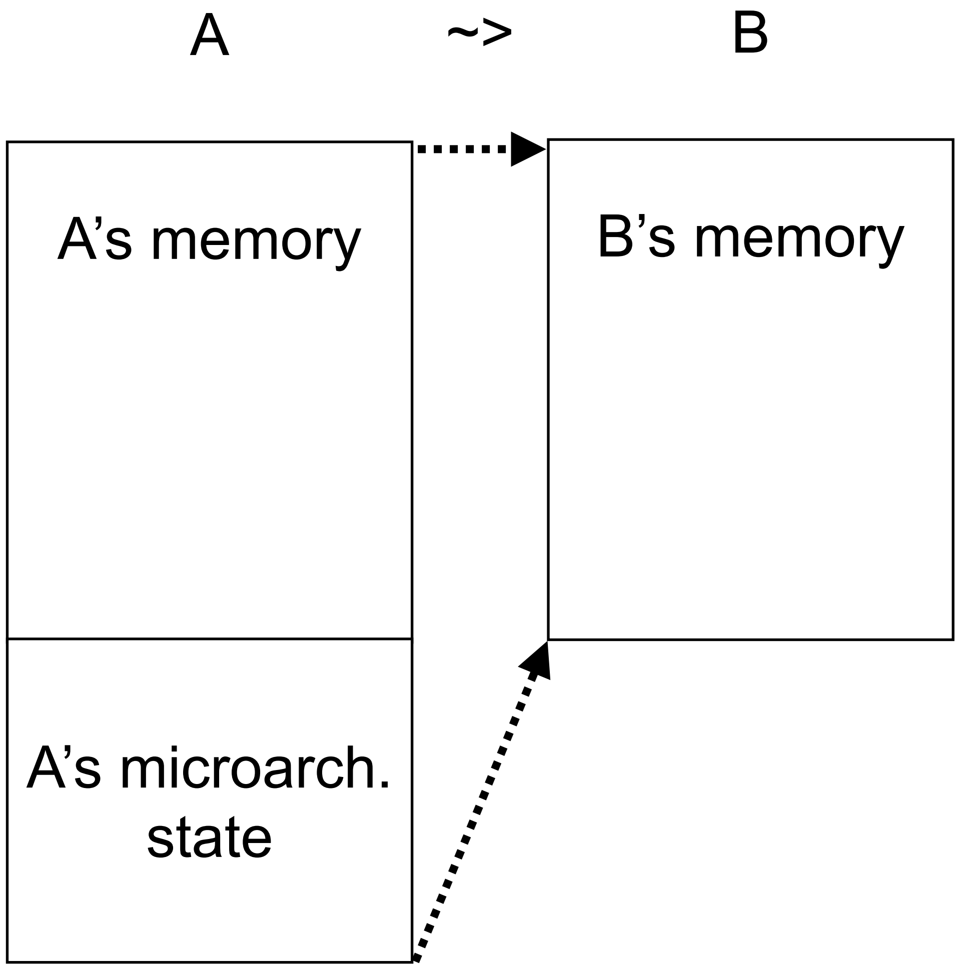
From prior seL4 infoflow proofs
[Murray et al. 2012, 2013]:
"all or nothing" policies

Overt vs *covert* state



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

Overt vs *covert* state

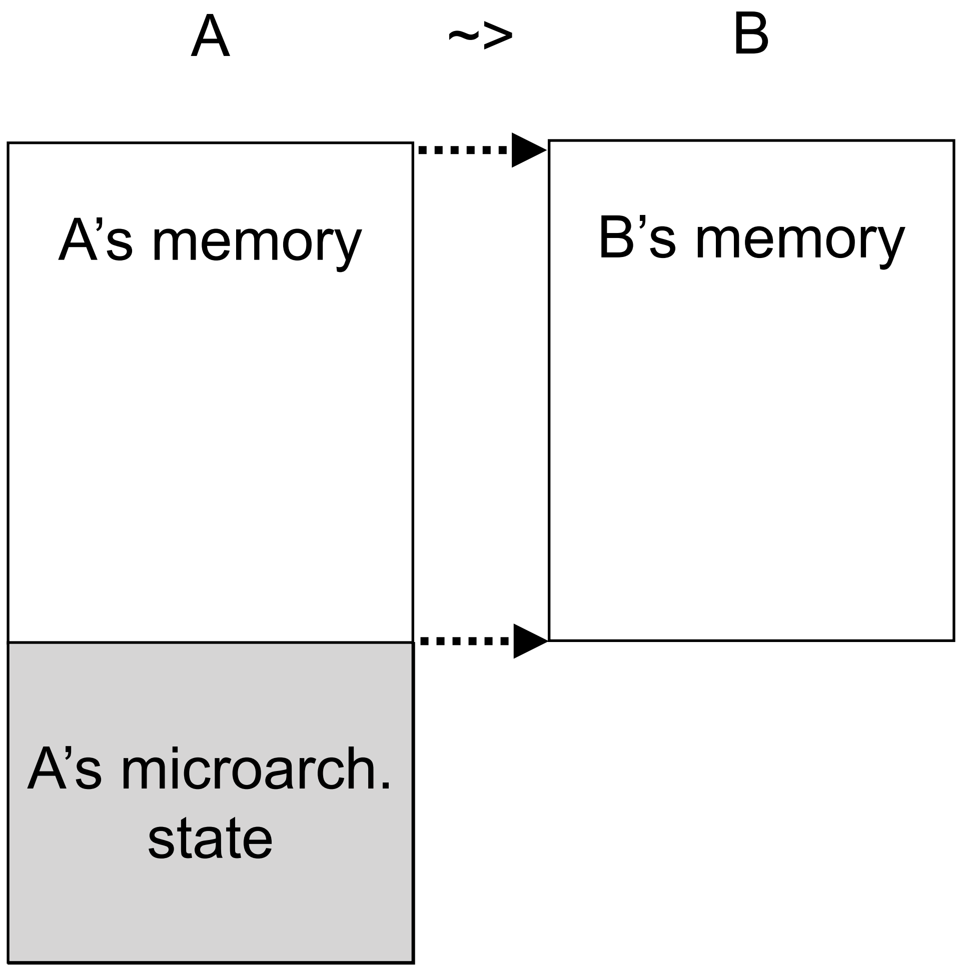

A ~> B

A's memory

B's memory

A's microarch. state

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


A ~> B

A's memory

B's memory

A's microarch. state

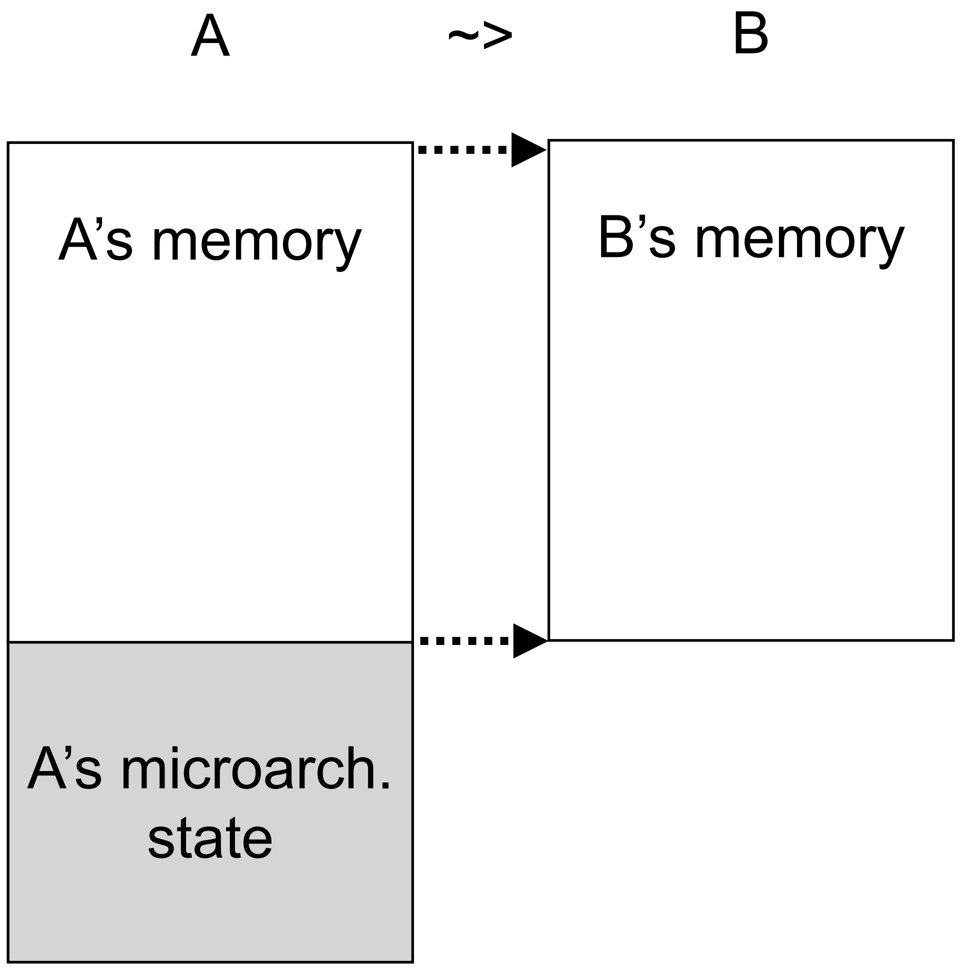
**Principle: Need policies
to allow some (*overt*) flows
while excluding other (*covert*) ones**

Covert state: Partitionable vs flushable

No channels!

OS 🤝 HW

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



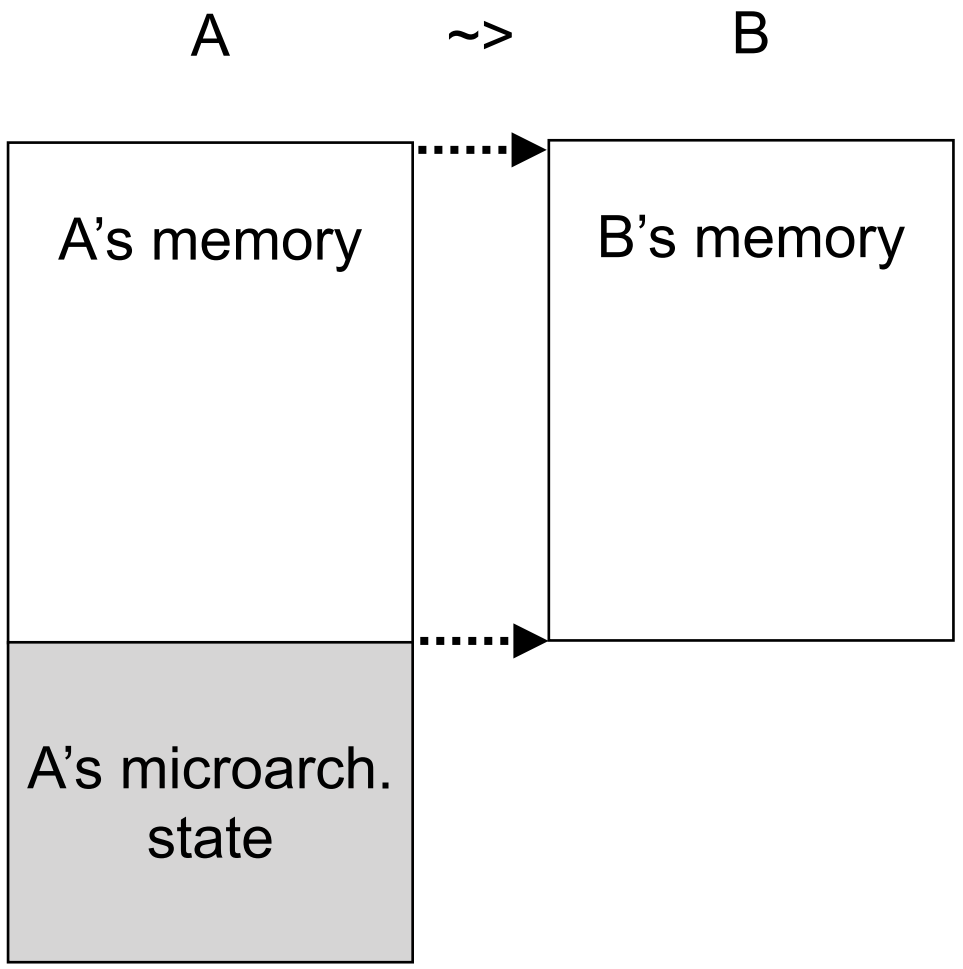
Covert state: Partitionable vs flushable

No channels!

OS 🤝 HW

<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

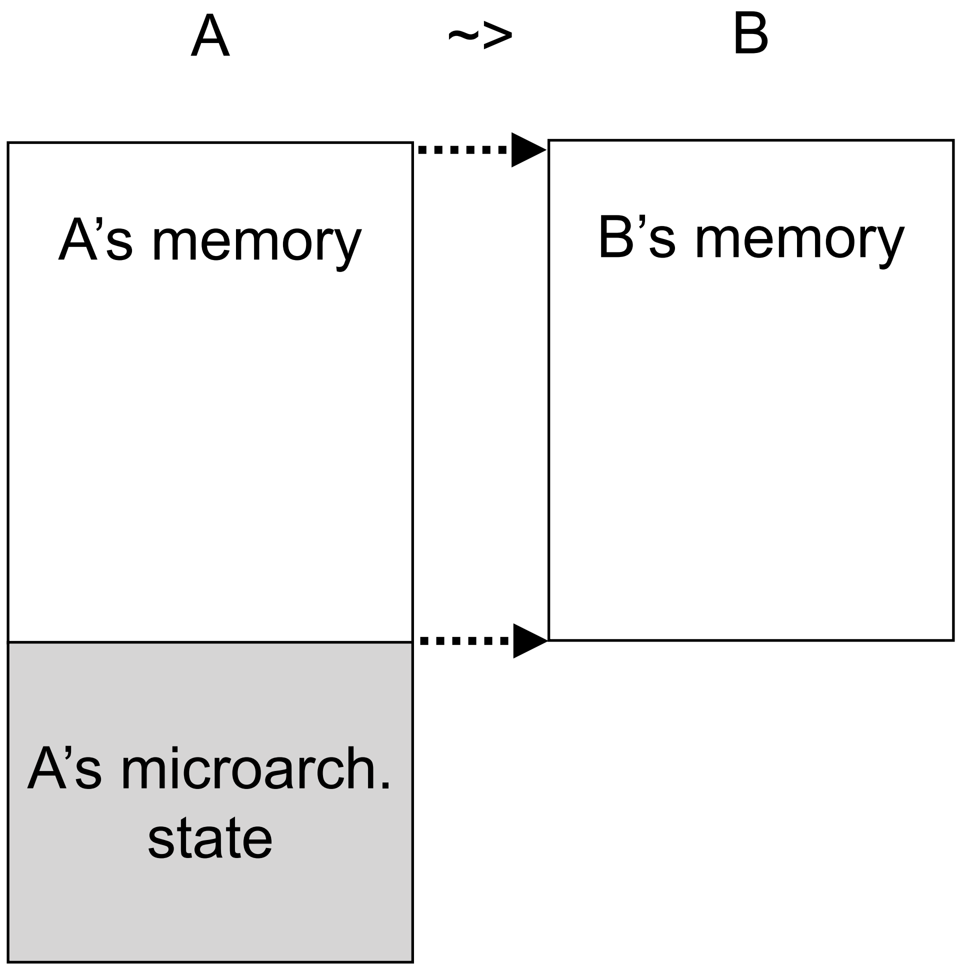
No channels!

OS 🤝 HW

<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:



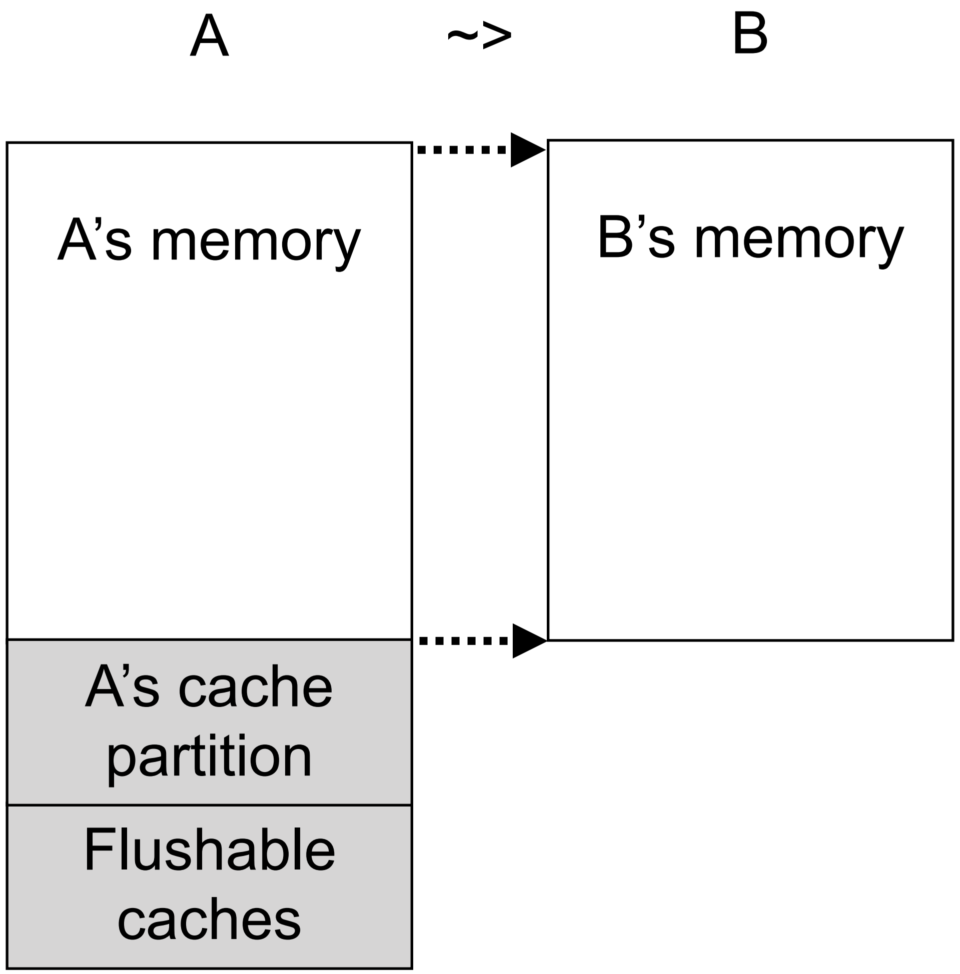
Covert state: Partitionable vs flushable

No channels!

OS 🤝 HW

<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*.



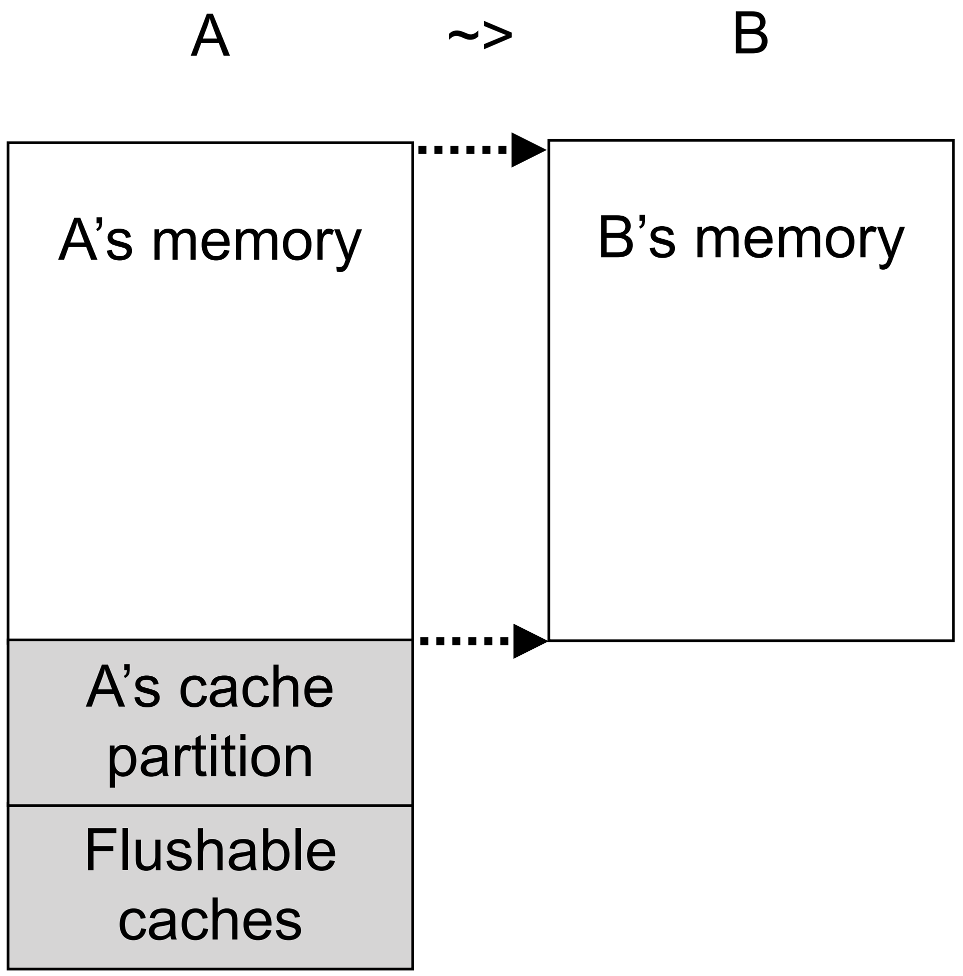
Covert state: Partitionable vs flushable

No channels!

OS 🤝 HW

<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.



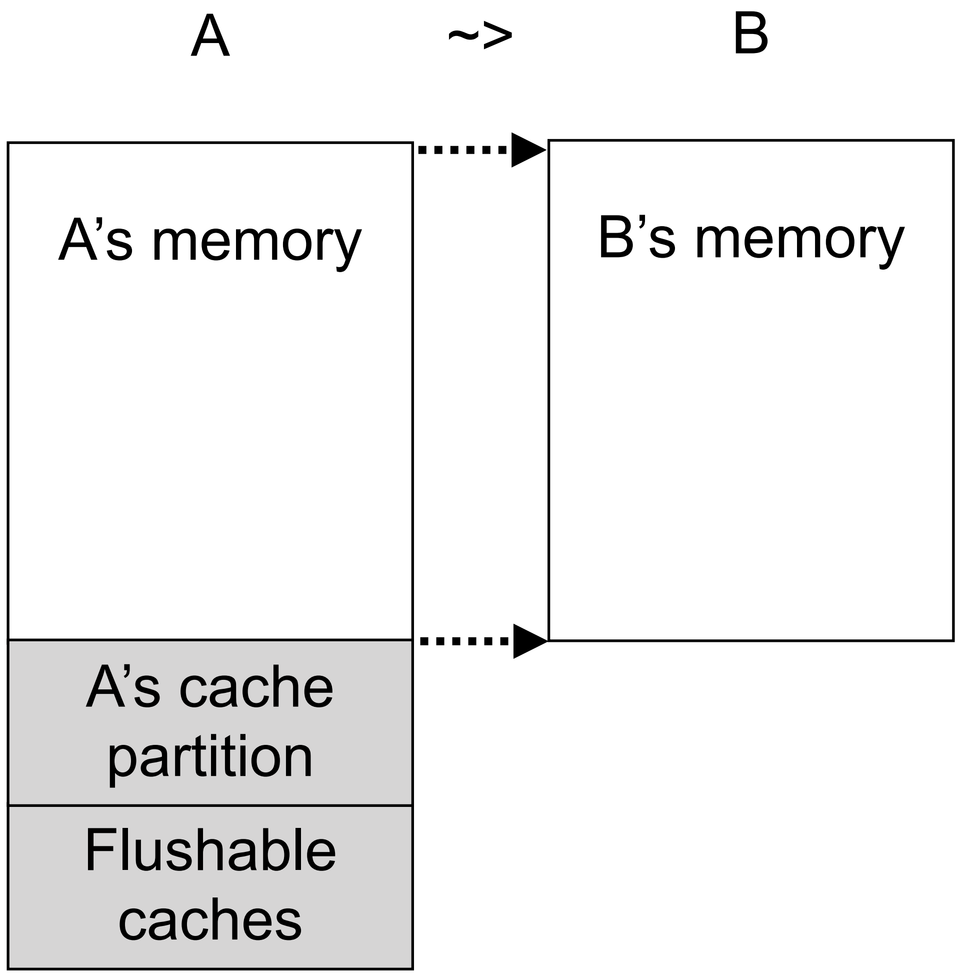
Covert state: Partitionable vs flushable

No channels!

OS 🤝 HW

<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).



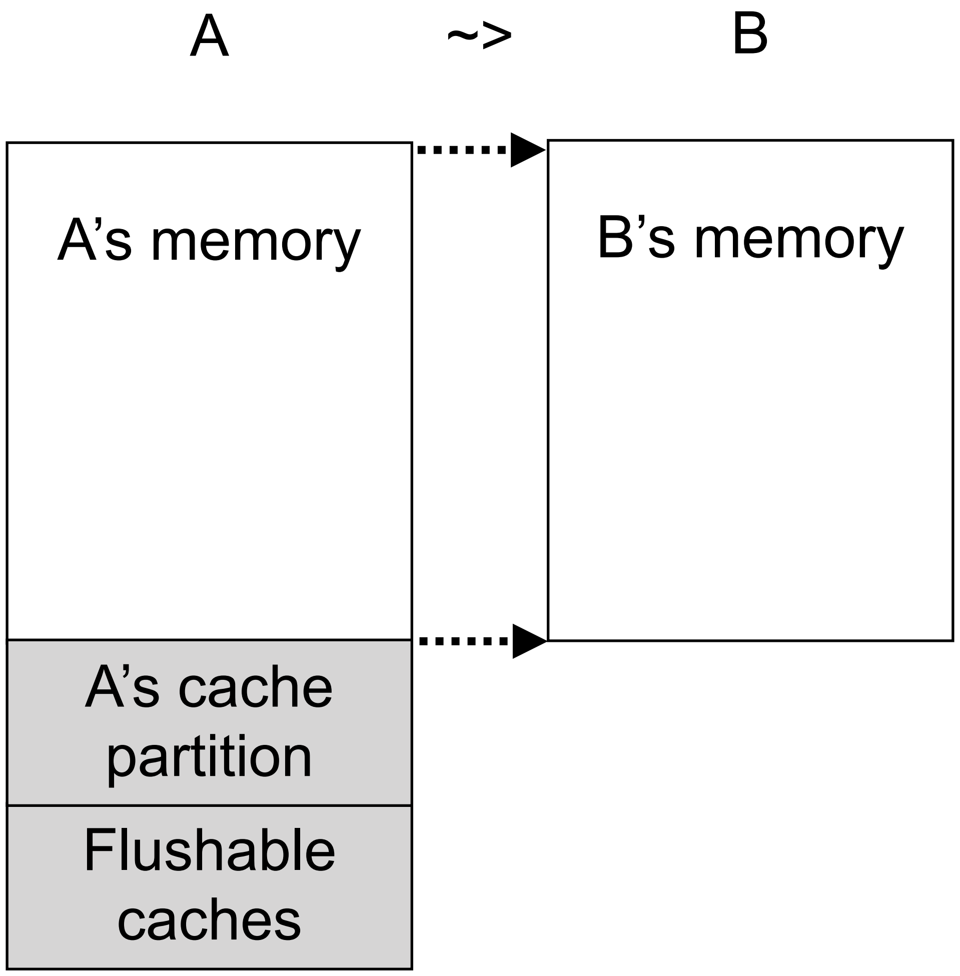
Covert state: Partitionable vs flushable

No channels!

OS 🤝 HW

<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)



A ~> B

A's memory

B's memory

A's cache partition

Flushable caches

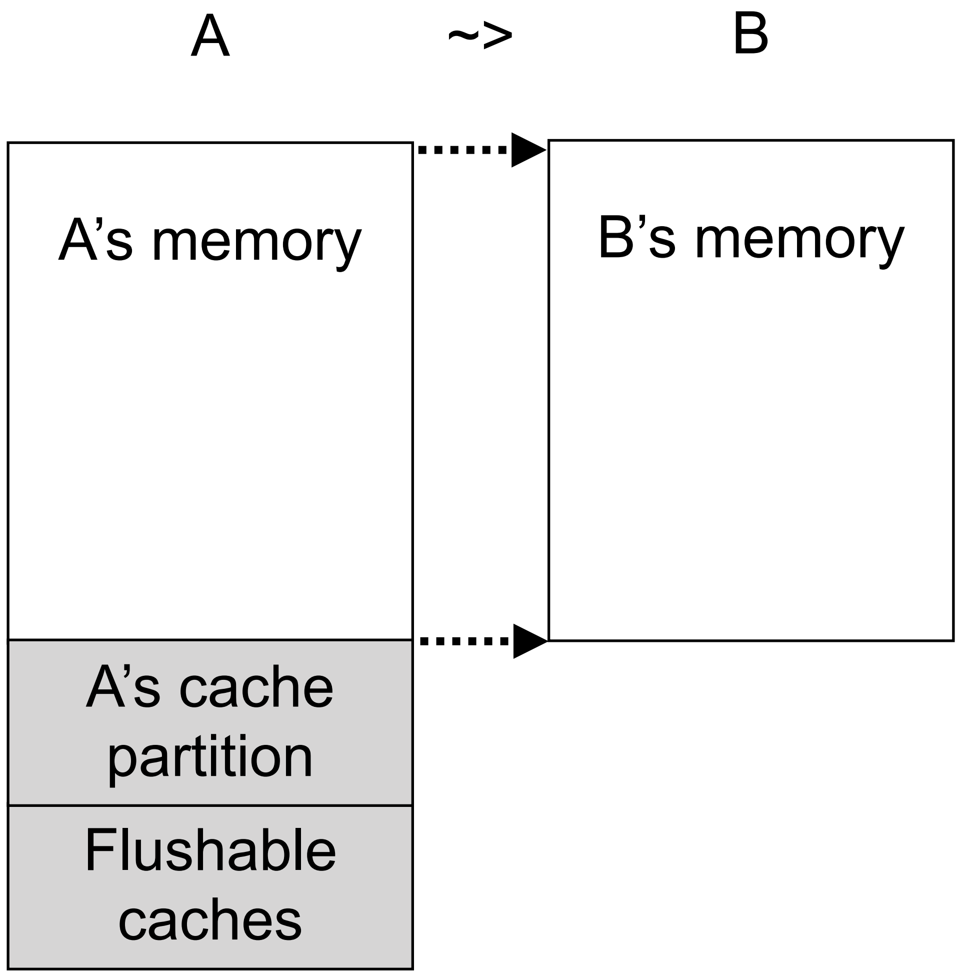
Covert state: Partitionable vs flushable

No channels!

OS 🤝 HW

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



How to formalise an OS enforces *time protection*?

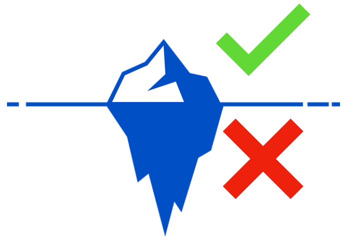
Versus threat scenario:
trojan and spy

🐴 ➡ 👁

No channels!

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*)

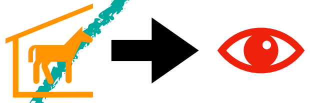
2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

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*?

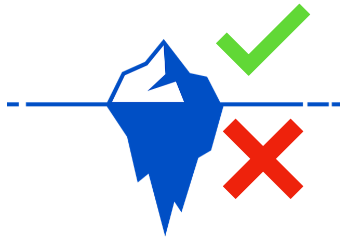
Versus threat scenario:
trojan and spy



No channels!

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*)

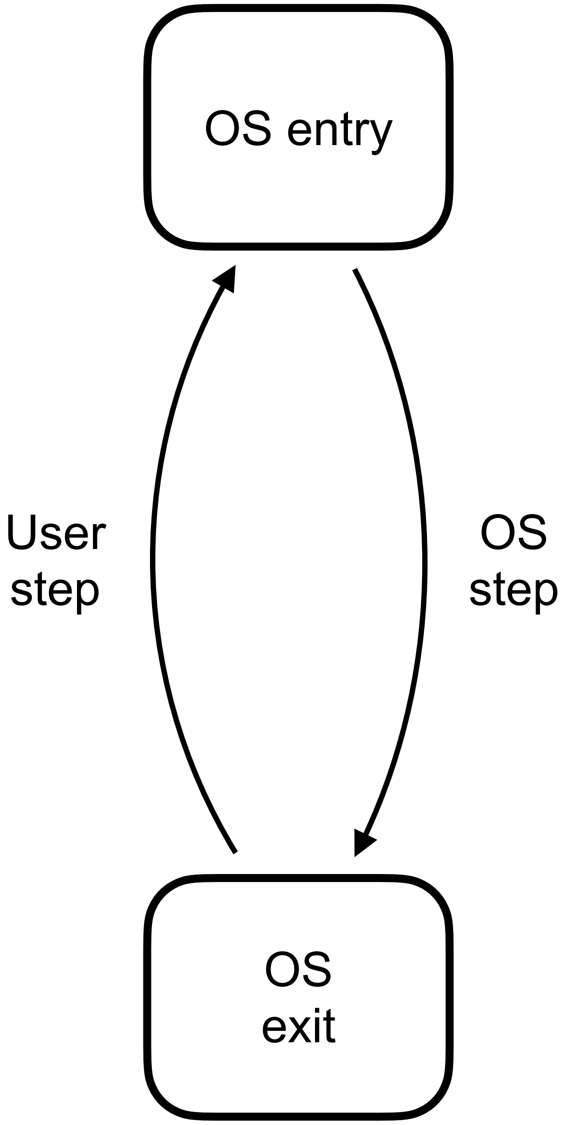
2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

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

4. Basic instantiation of OS model exercising dynamic policy.

OS security model

Transition system

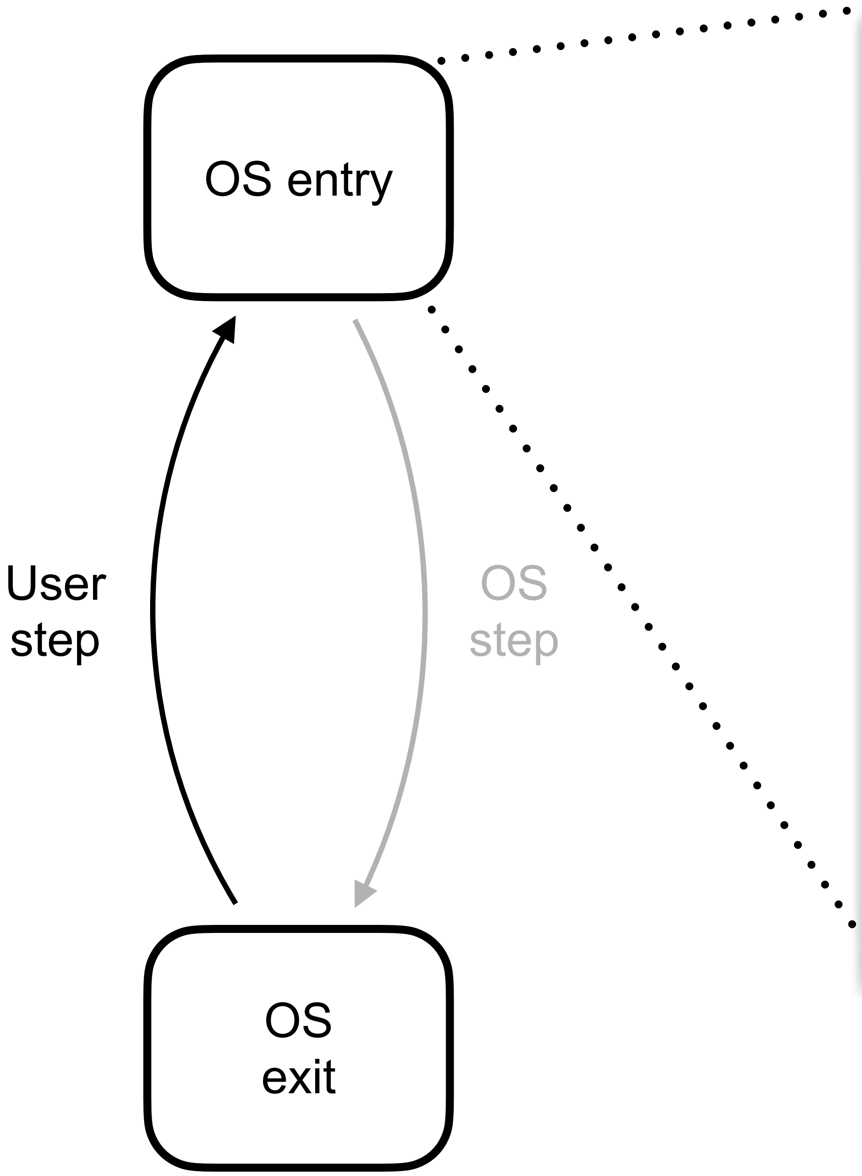


OS security model

No channels!

OS 🤝 HW

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 */

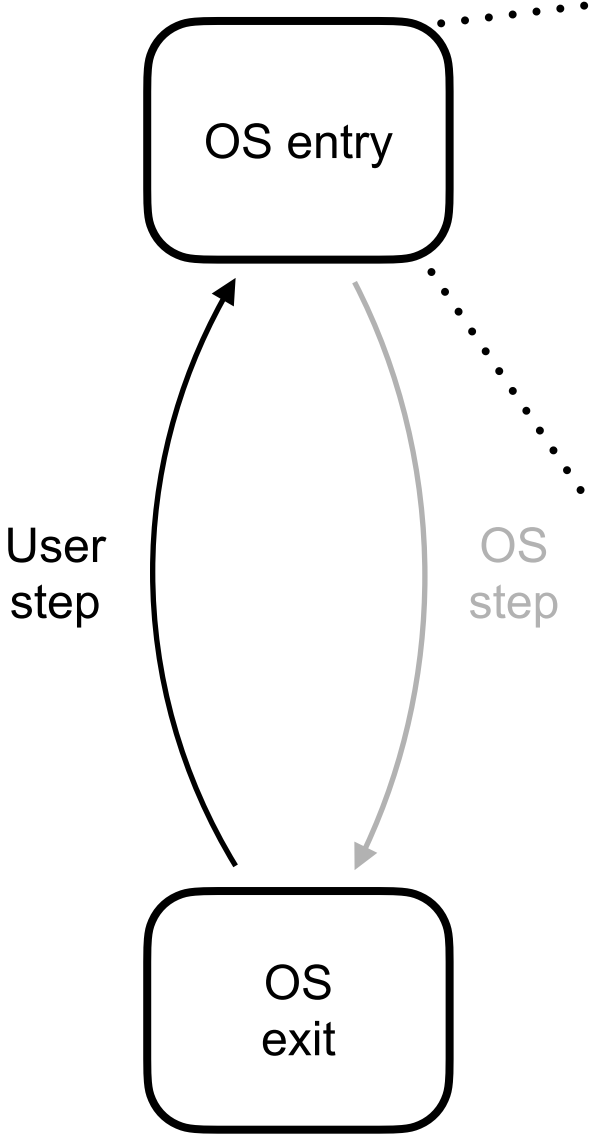
prot :: *prot* /* Protection state */

OS security model

No channels!

OS 🤝 HW

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 */
```

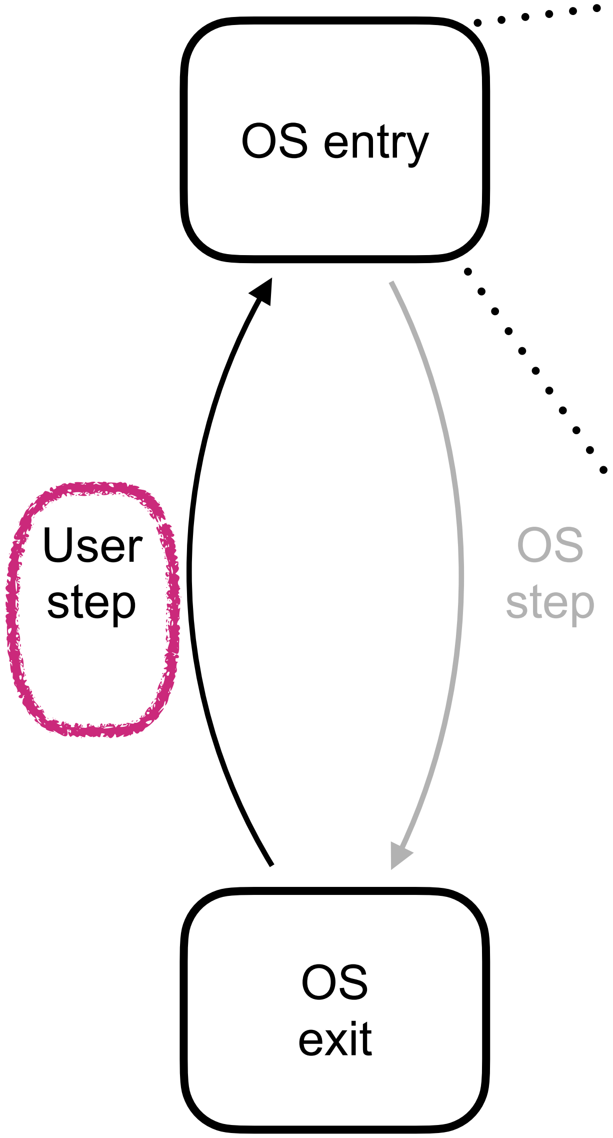
Microarchitecture
Devices

OS security model

No channels!

OS 🤝 HW

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 */

Microarchitecture

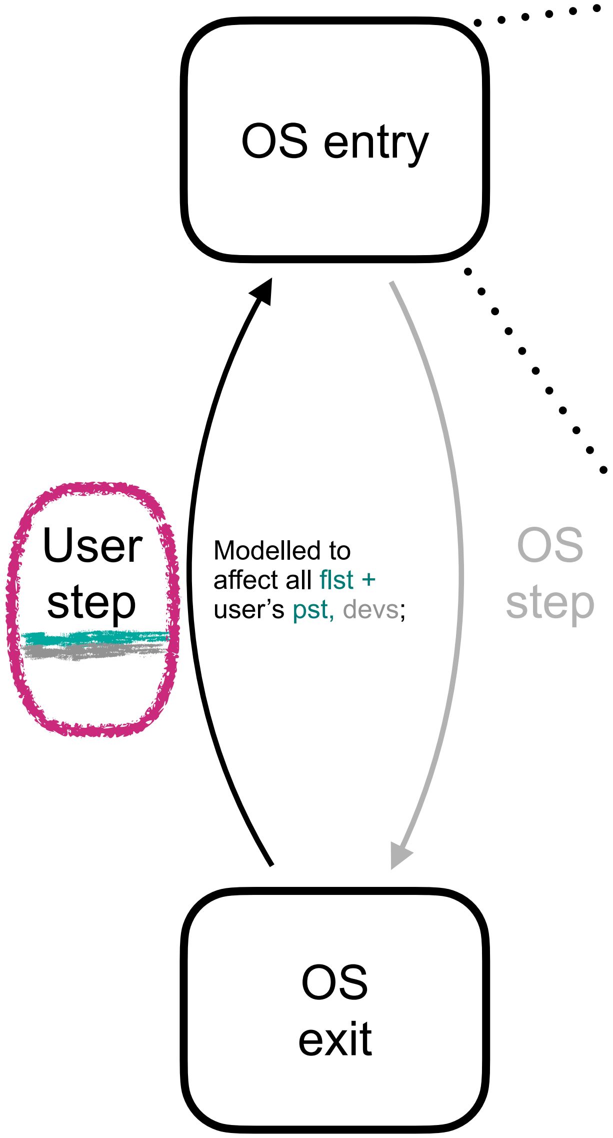
Devices

OS security model

No channels!

OS 🤝 HW

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 */
```

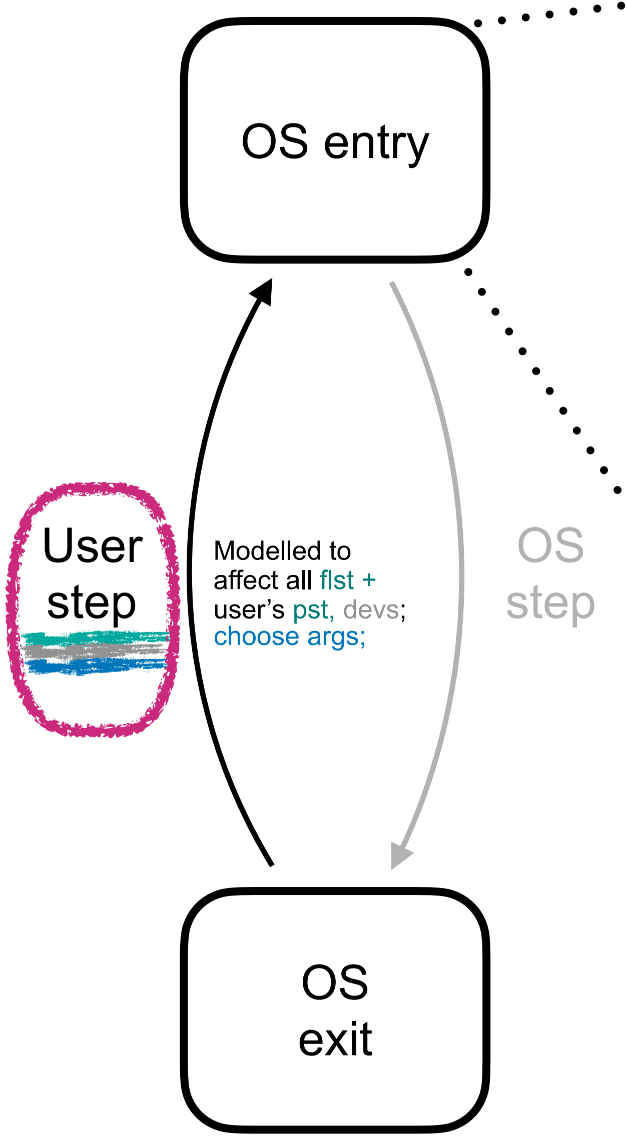
Microarchitecture
Devices

OS security model

No channels!

OS 🤝 HW

Transition system



OS entry
User step
Modelled to affect all flst + user's pst, devs; choose args;
OS step
OS exit

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 */
```

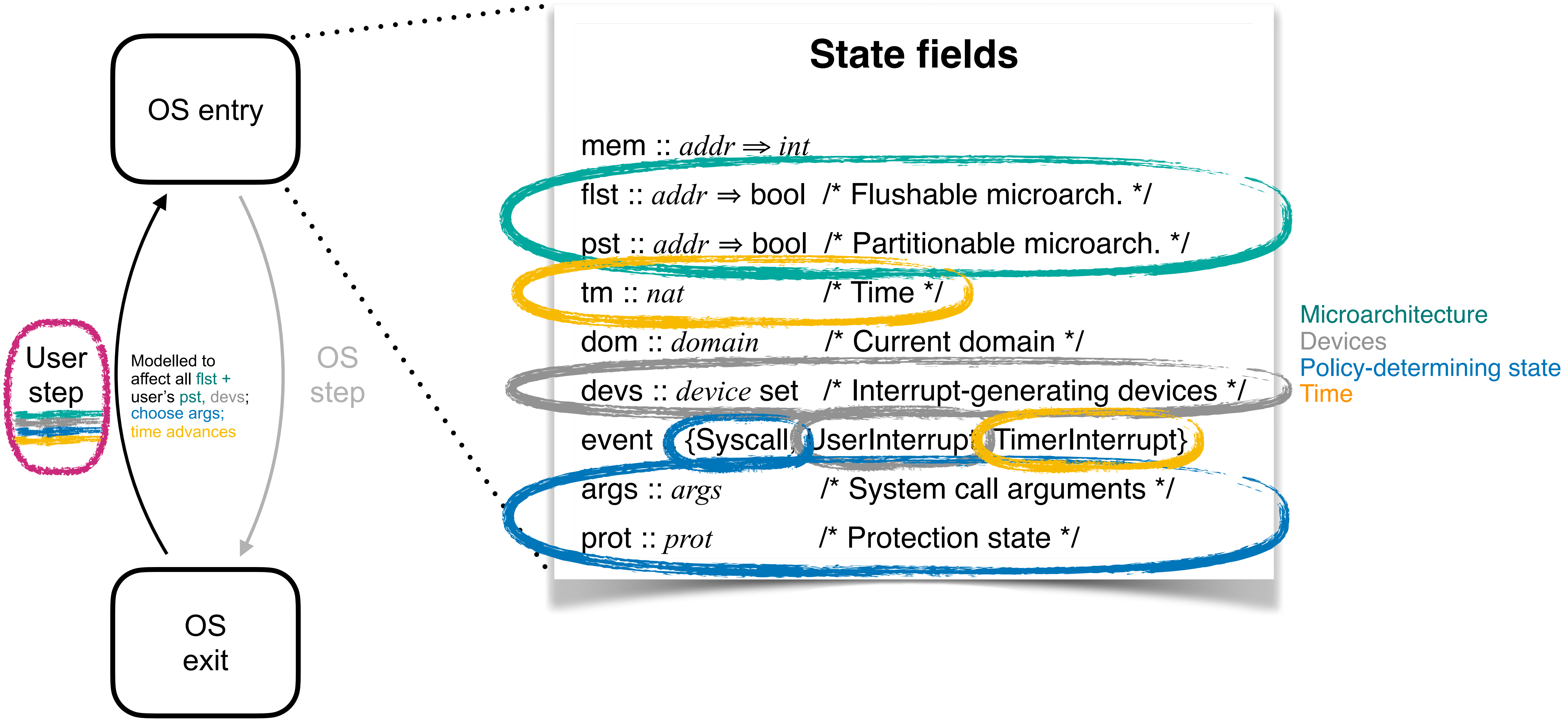
Microarchitecture
Devices
Policy-determining state

OS security model

No channels!

OS 🤝 HW

Transition system



OS entry
User step
Modelled to affect all flst + user's pst, devs; choose args; time advances
OS step
OS exit

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 */
```

Microarchitecture
Devices
Policy-determining state
Time

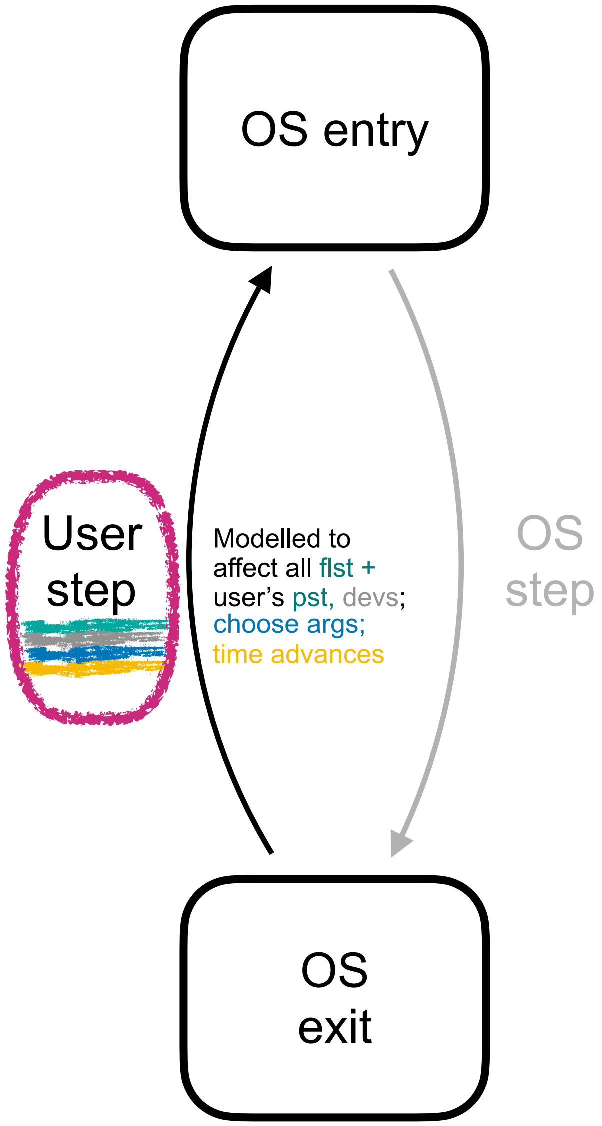
OS security model

No channels!

OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Transition system


OS entry
User step
Modelled to affect all flst + user's pst, devs; choose args; time advances
OS step
OS exit

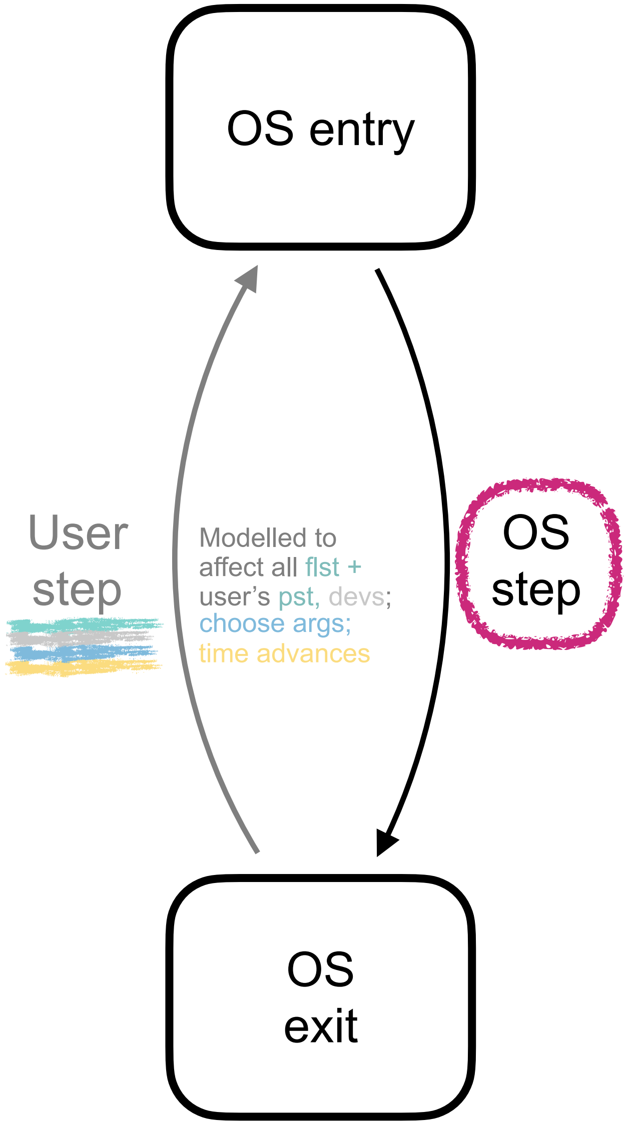
OS security model

No channels!

OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Transition system



Modelled to affect all flst + user's pst, devs; choose args; time advances

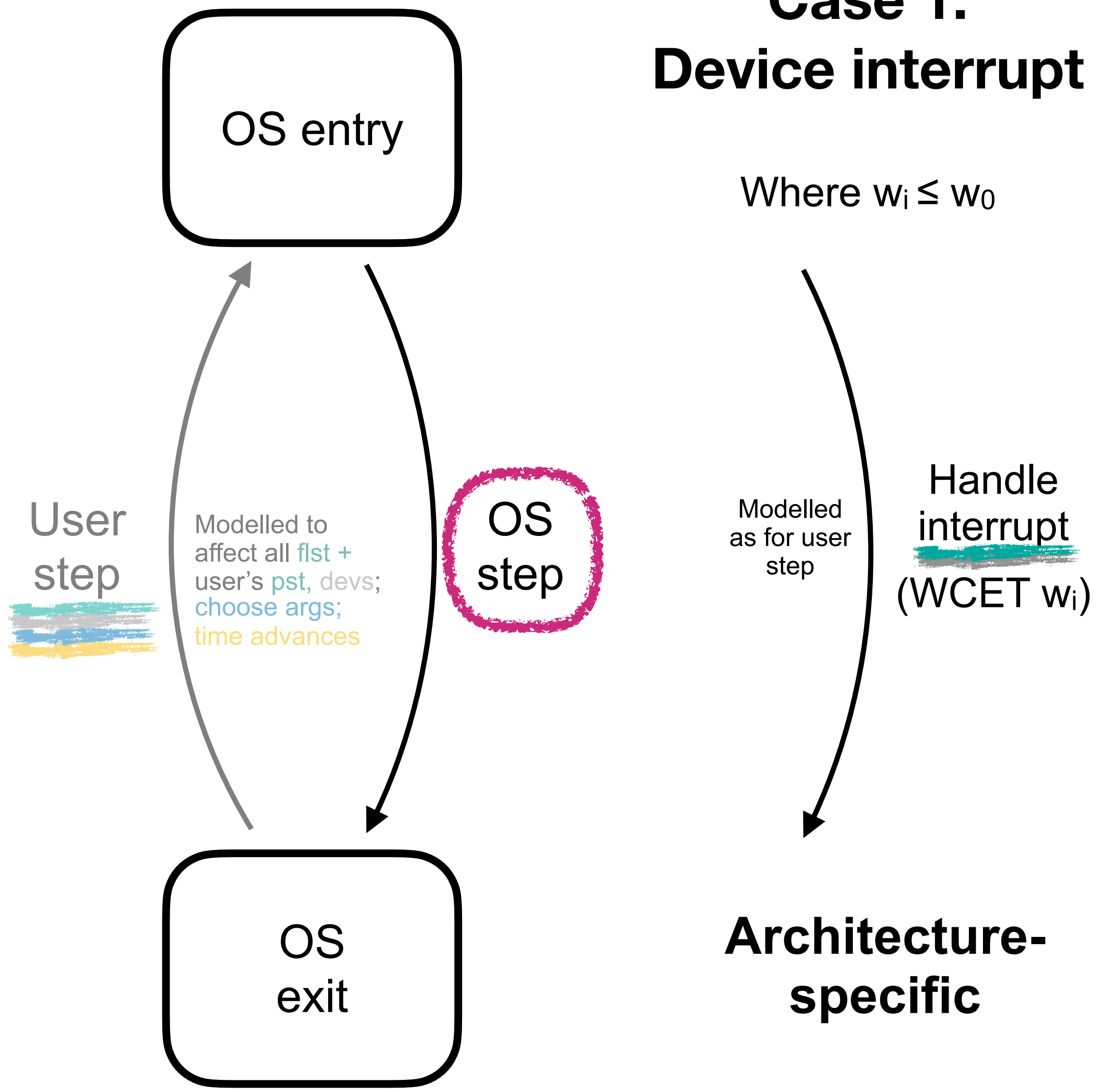
OS security model

No channels!

OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Transition system



OS entry

OS exit

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

OS step

Case 1: Device interrupt

Where $w_i \leq w_0$

Modelled as for user step

Handle interrupt (WCET w_i)

Architecture-specific

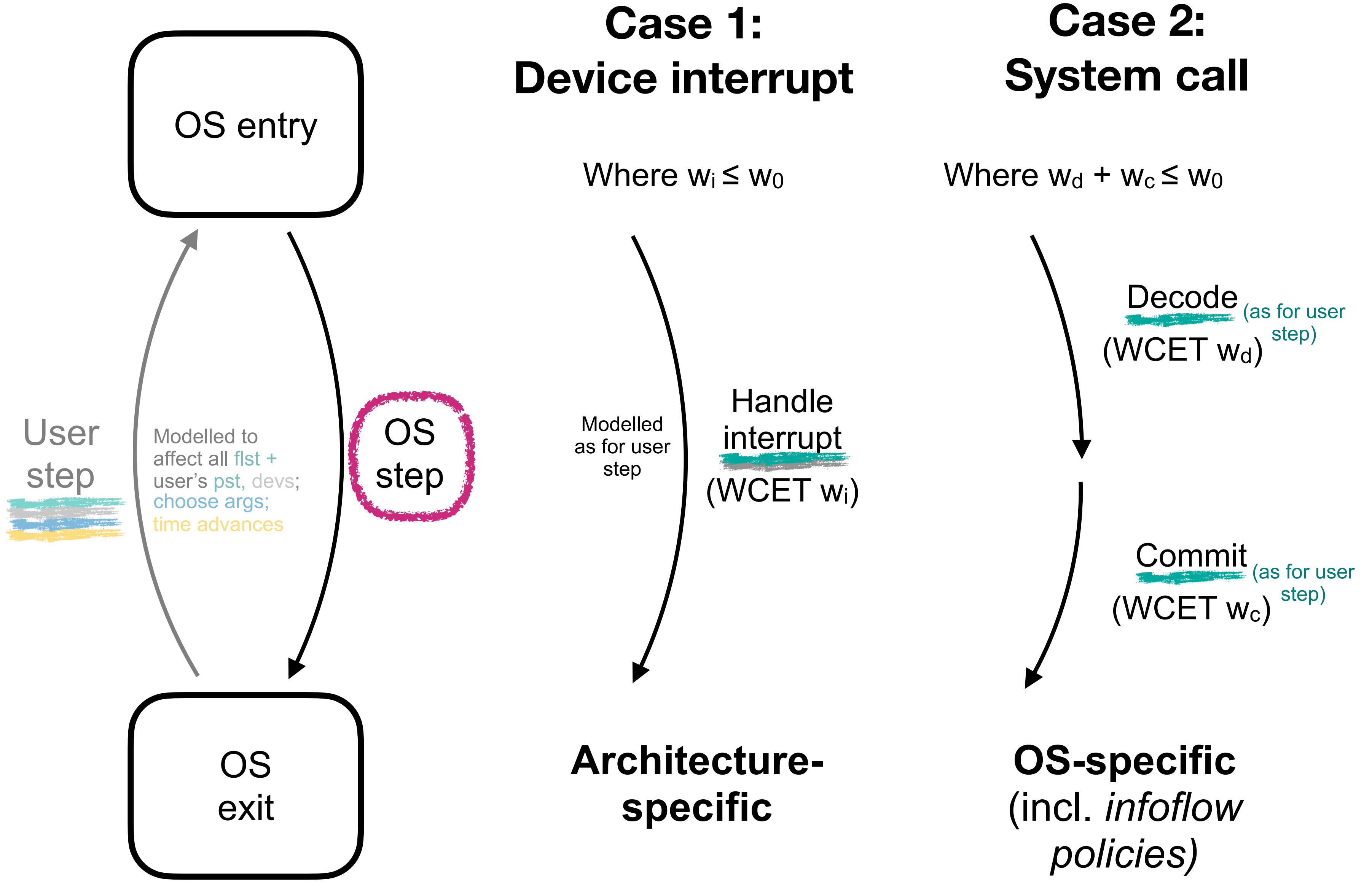
OS security model

No channels!

OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Transition system



OS entry

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

OS step

OS exit

Case 1: Device interrupt

Where $w_i \leq w_0$

Modelled as for user step

Handle interrupt (WCET w_i)

Architecture-specific

Case 2: System call

Where $w_d + w_c \leq w_0$

Decode (as for user step) (WCET w_d)

Commit (as for user step) (WCET w_c)

OS-specific (incl. *infoflow policies)*

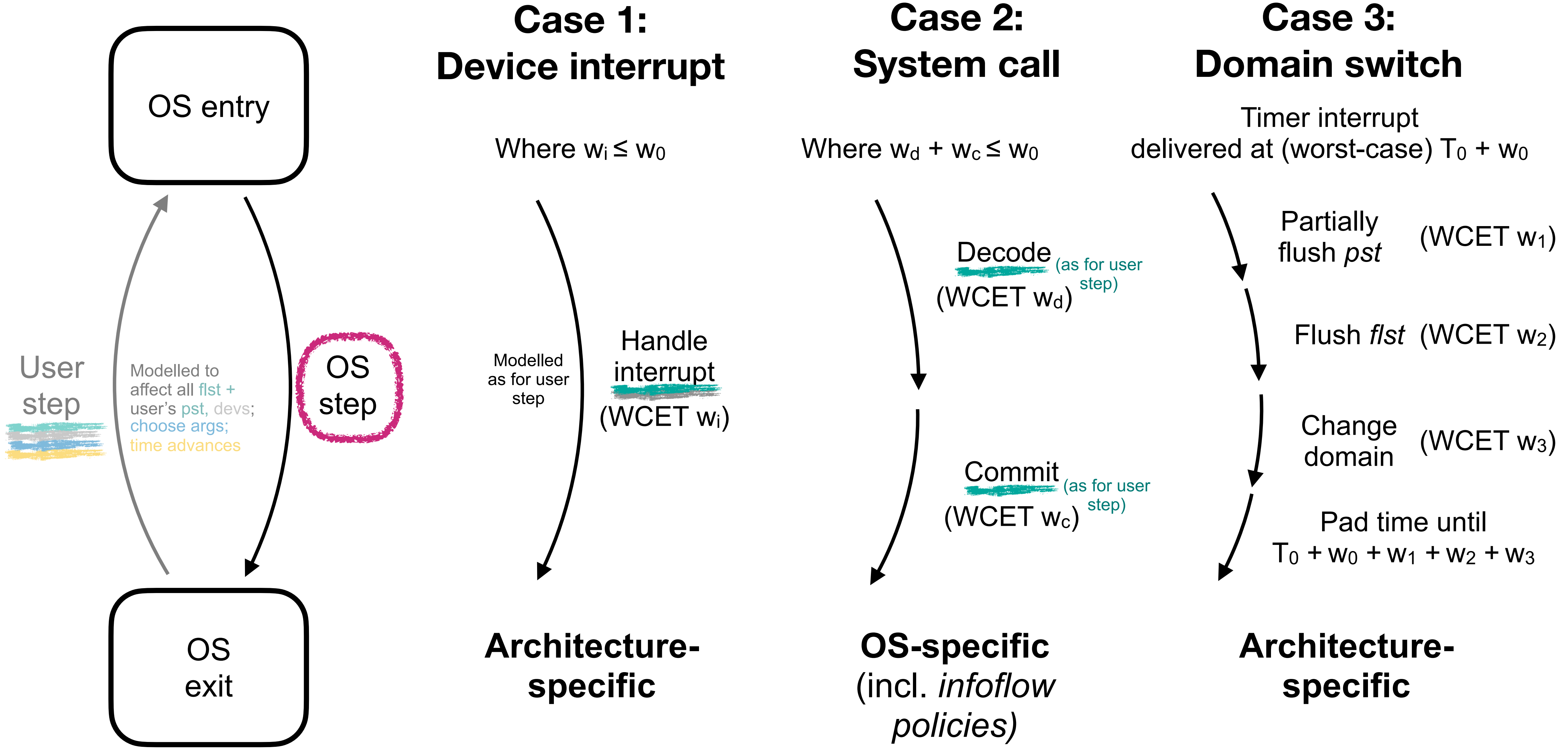
OS security model

No channels!

OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Transition system



OS entry

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

OS step

OS exit

Case 1: Device interrupt

Where $w_i \leq w_0$

Modelled as for user step

Handle interrupt (WCET w_i)

Architecture-specific

Case 2: System call

Where $w_d + w_c \leq w_0$

Decode (as for user step) (WCET w_d)

Commit (as for user step) (WCET w_c)

OS-specific (incl. *infoflow policies*)

Case 3: Domain switch

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

Partially flush *pst* (WCET w_1)

Flush *flst* (WCET w_2)

Change domain (WCET w_3)

Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

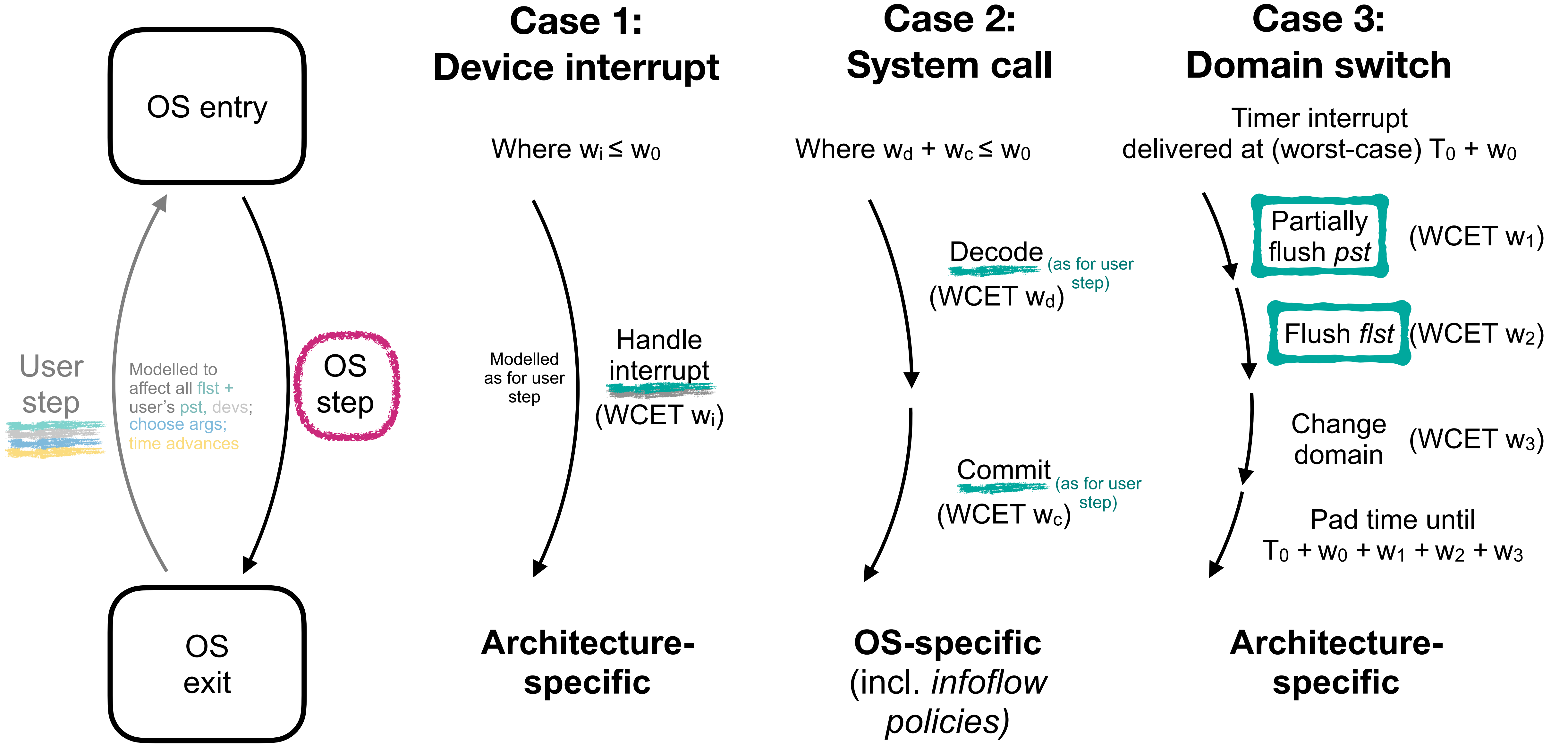
OS security model

No channels!

OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Transition system



OS entry

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

OS step

OS exit

Case 1: Device interrupt

Where $w_i \leq w_0$

Modelled as for user step

Handle interrupt (WCET w_i)

Architecture-specific

Case 2: System call

Where $w_d + w_c \leq w_0$

Decode (as for user step) (WCET w_d)

Commit (as for user step) (WCET w_c)

OS-specific (incl. *infoflow policies*)

Case 3: Domain switch

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

Partially flush *pst* (WCET w_1)

Flush *flst* (WCET w_2)

Change domain (WCET w_3)

Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

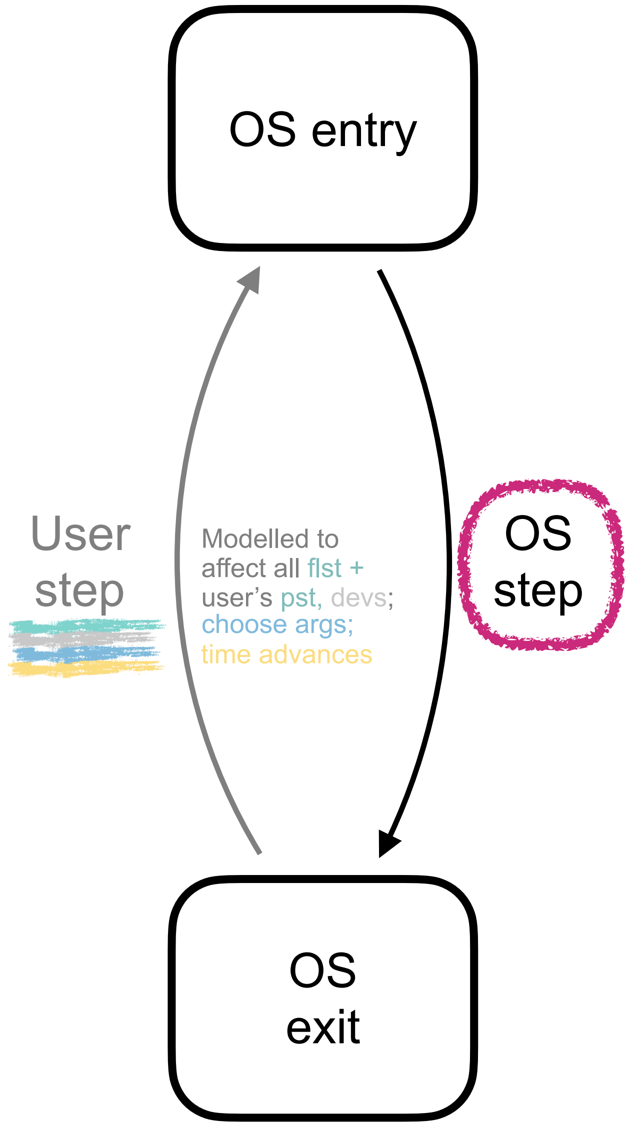
OS security model

No channels!

OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Transition system



OS entry

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

OS step

OS exit

Case 1: Device interrupt

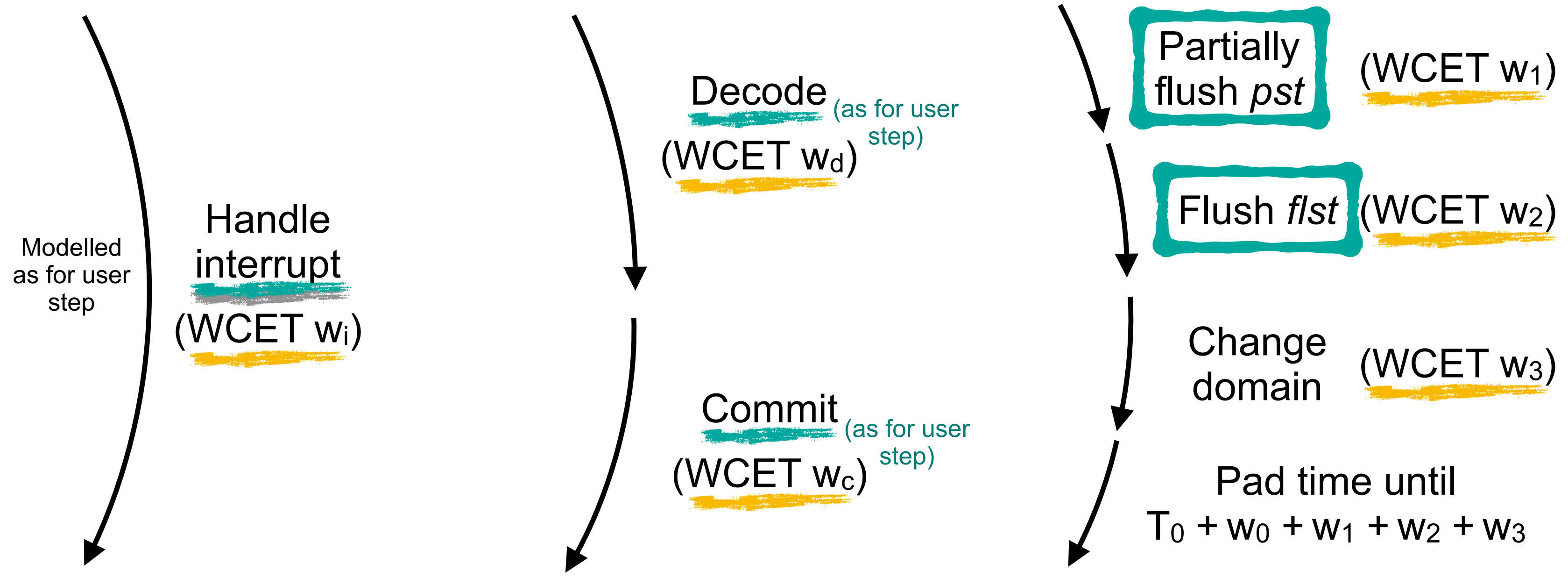
Where $w_i \leq w_0$

Case 2: System call

Where $w_d + w_c \leq w_0$

Case 3: Domain switch

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



Modelled as for user step

Handle interrupt (WCET w_i)

Decode (as for user step) (WCET w_d)

Commit (as for user step) (WCET w_c)

Partially flush *pst* (WCET w_1)

Flush *flst* (WCET w_2)

Change domain (WCET w_3)

Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

OS-specific (incl. *infoflow policies)*

Architecture-specific

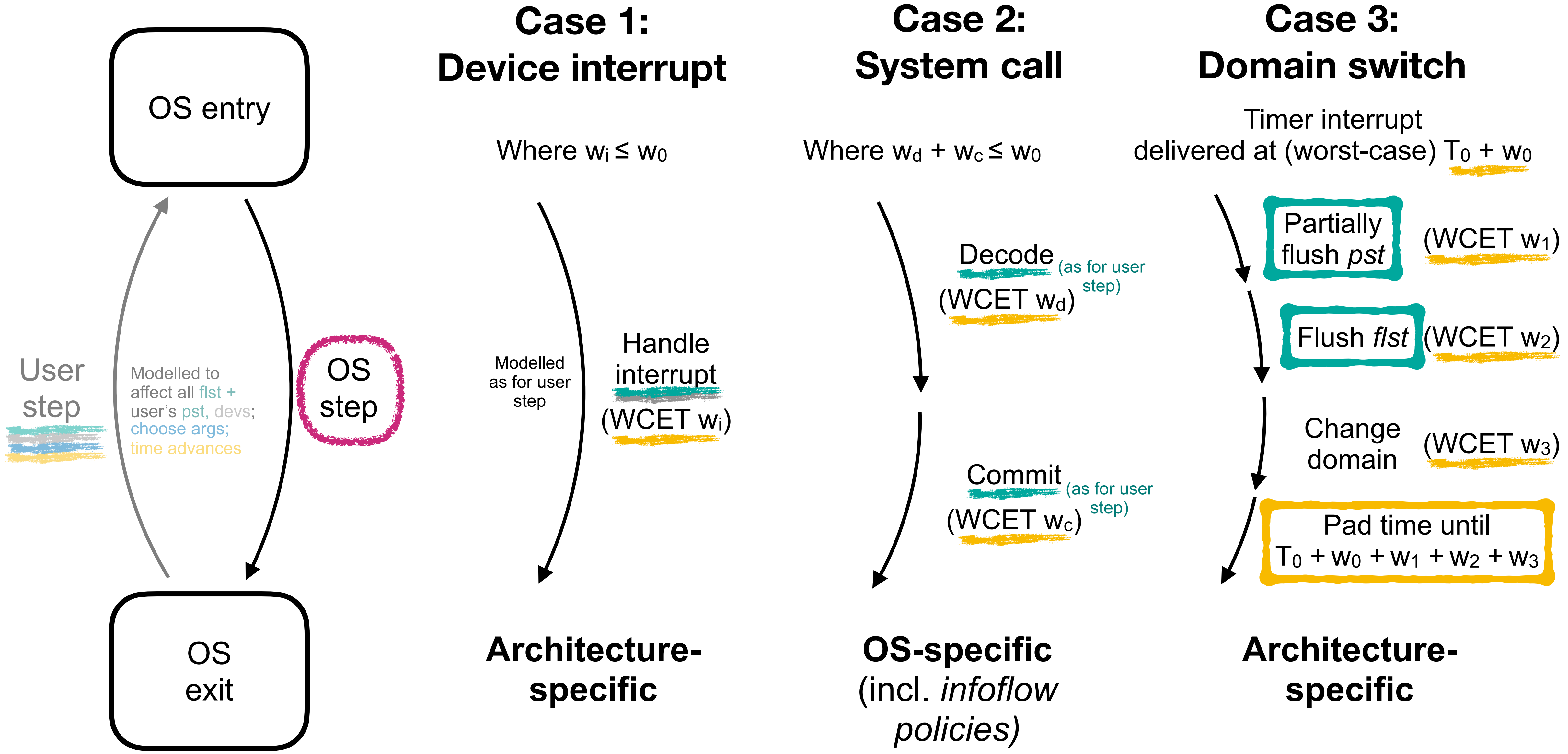
OS security model

No channels!

OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Transition system



OS entry

OS exit

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

OS step

Case 1: Device interrupt

Where $w_i \leq w_0$

Modelled as for user step

Handle interrupt

(WCET w_i)

Architecture-specific

Case 2: System call

Where $w_d + w_c \leq w_0$

Decode (as for user step)

(WCET w_d)

Commit (as for user step)

(WCET w_c)

OS-specific (incl. *infoflow policies)*

Case 3: Domain switch

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

Partially flush *pst* (WCET w_1)

Flush *flst* (WCET w_2)

Change domain (WCET w_3)

Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

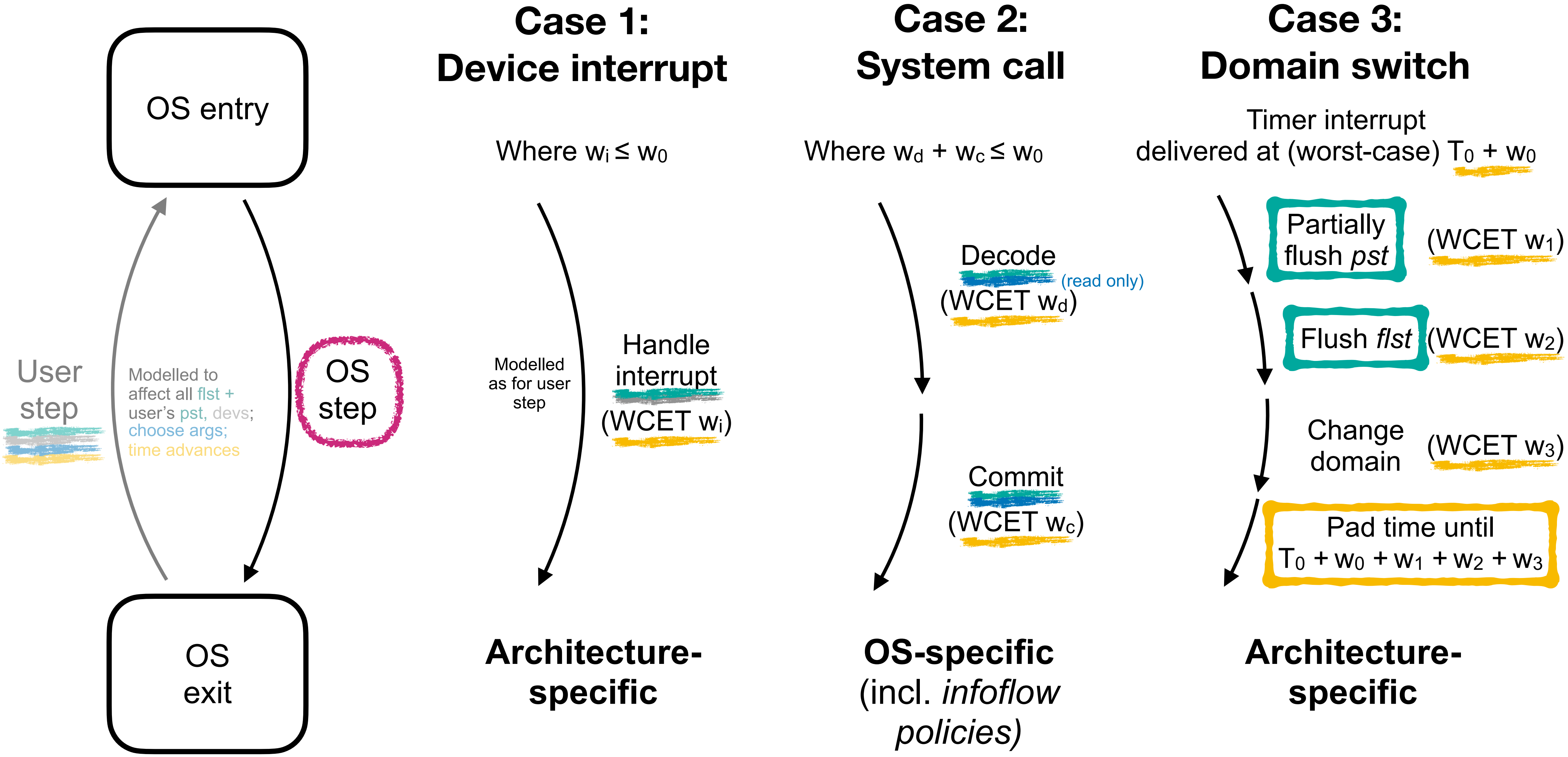
OS security model

No channels!

OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Transition system



OS entry

User step

Modelled to affect all *flst* + user's *pst*, devs; choose args; time advances

OS step

OS exit

Case 1: Device interrupt

Where $w_i \leq w_0$

Modelled as for user step

Handle interrupt ($\mathrm{WCET}\ w_i$)

Architecture-specific

Case 2: System call

Where $w_d + w_c \leq w_0$

Decode ($\mathrm{WCET}\ w_d$) (read only)

Commit ($\mathrm{WCET}\ w_c$)

OS-specific (incl. *infoflow policies)*

Case 3: Domain switch

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

Partially flush *pst* ($\mathrm{WCET}\ w_1$)

Flush *flst* ($\mathrm{WCET}\ w_2$)

Change domain ($\mathrm{WCET}\ w_3$)

Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

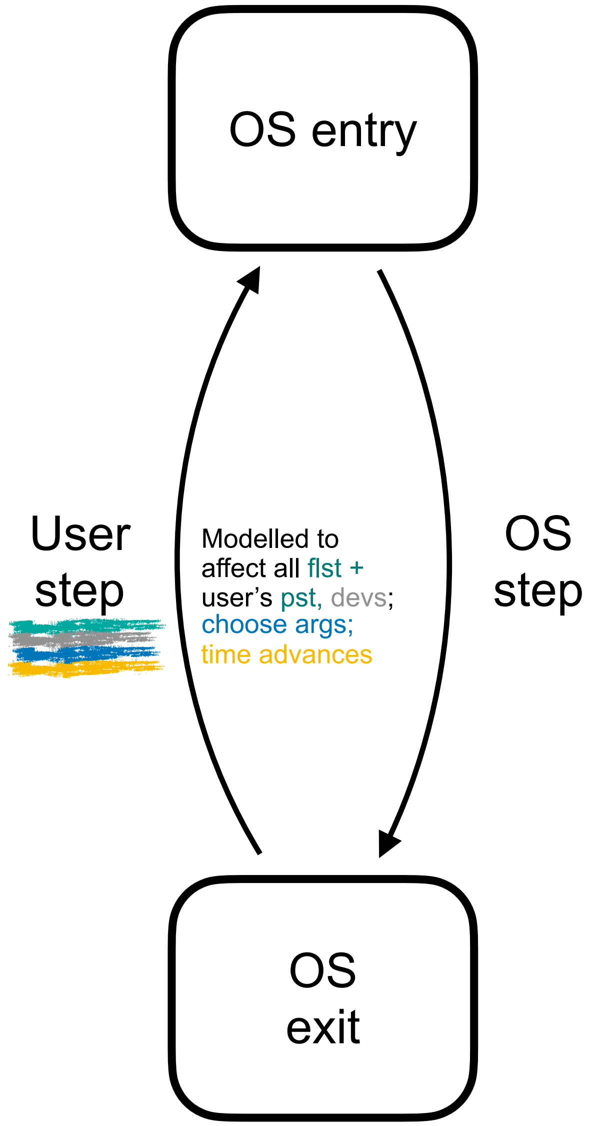
Security proof approach

No channels!

OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Transition system



OS entry

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

OS step

OS exit

Case 1: Device interrupt

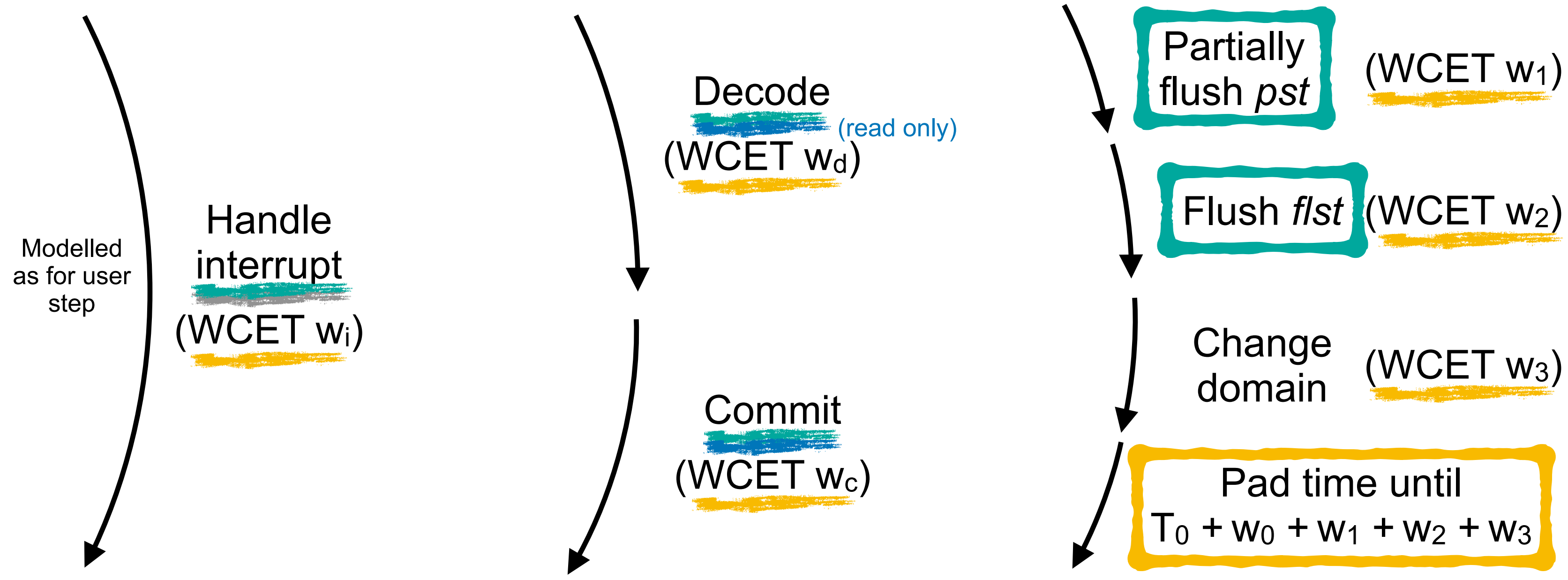
Case 2: System call

Case 3: Domain switch

Where $w_i \leq w_0$

Where $w_d + w_c \leq w_0$

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



Modelled as for user step

Handle interrupt ($\text{WCET } w_i$)

Decode (read only) ($\text{WCET } w_d$)

Commit ($\text{WCET } w_c$)

Partially flush *pst* ($\text{WCET } w_1$)

Flush *flst* ($\text{WCET } w_2$)

Change domain ($\text{WCET } w_3$)

Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

OS-specific (incl. *infoflow policies*)

Architecture-specific

Security proof approach

No channels!

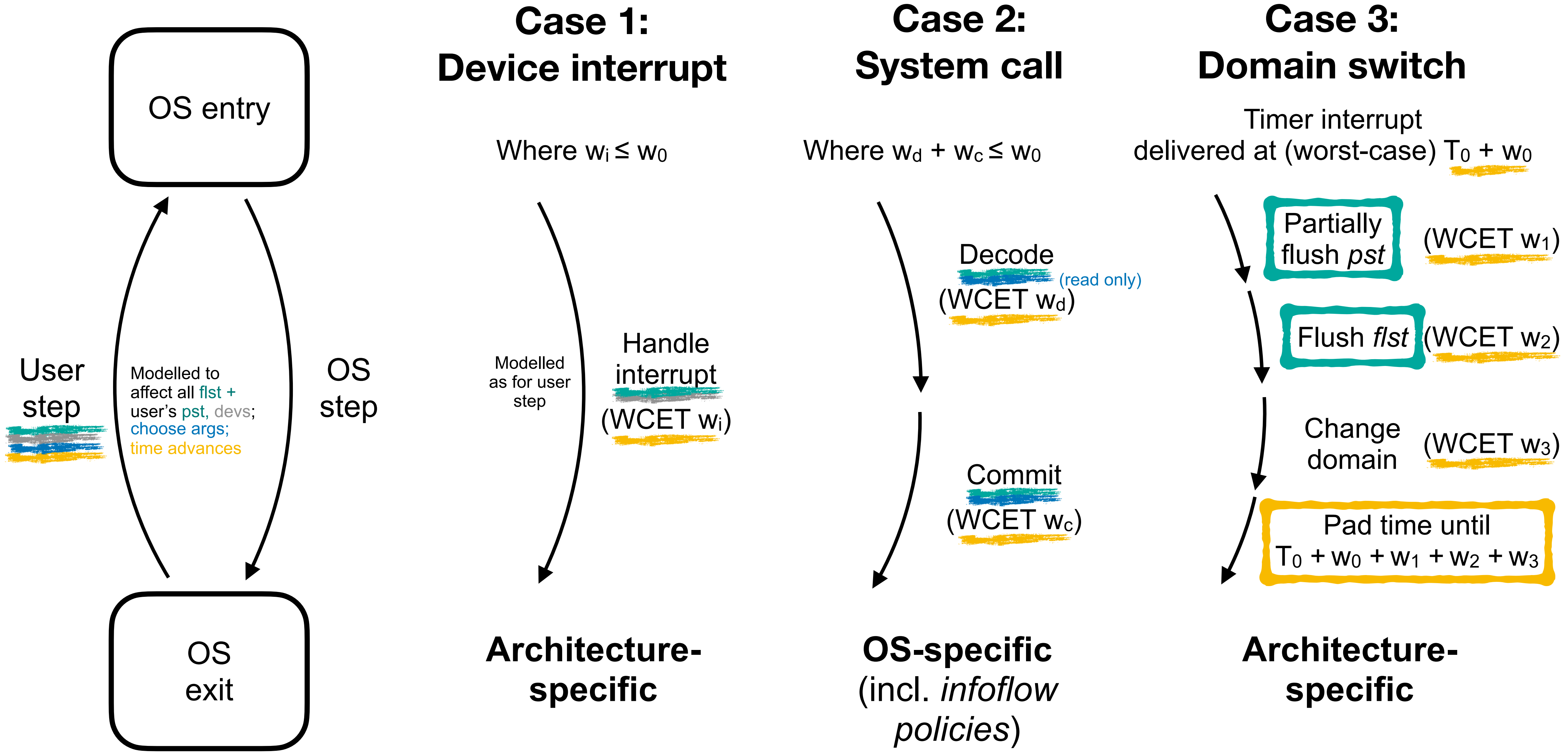
OS 🤝 HW

Requirements

(In addition to WCETs)

Microarchitecture
Devices
Policy-determining state
Time

Transition system



OS entry

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

OS step

OS exit

Case 1: Device interrupt

Where $w_i \leq w_0$

Modelled as for user step

Handle interrupt (WCET w_i)

Architecture-specific

Case 2: System call

Where $w_d + w_c \leq w_0$

Decode (WCET w_d)

(read only)

Commit (WCET w_c)

OS-specific (incl. *infoflow policies*)

Case 3: Domain switch

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

Partially flush *pst* (WCET w_1)

Flush *flst* (WCET w_2)

Change domain (WCET w_3)

Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

Security proof approach

No channels!

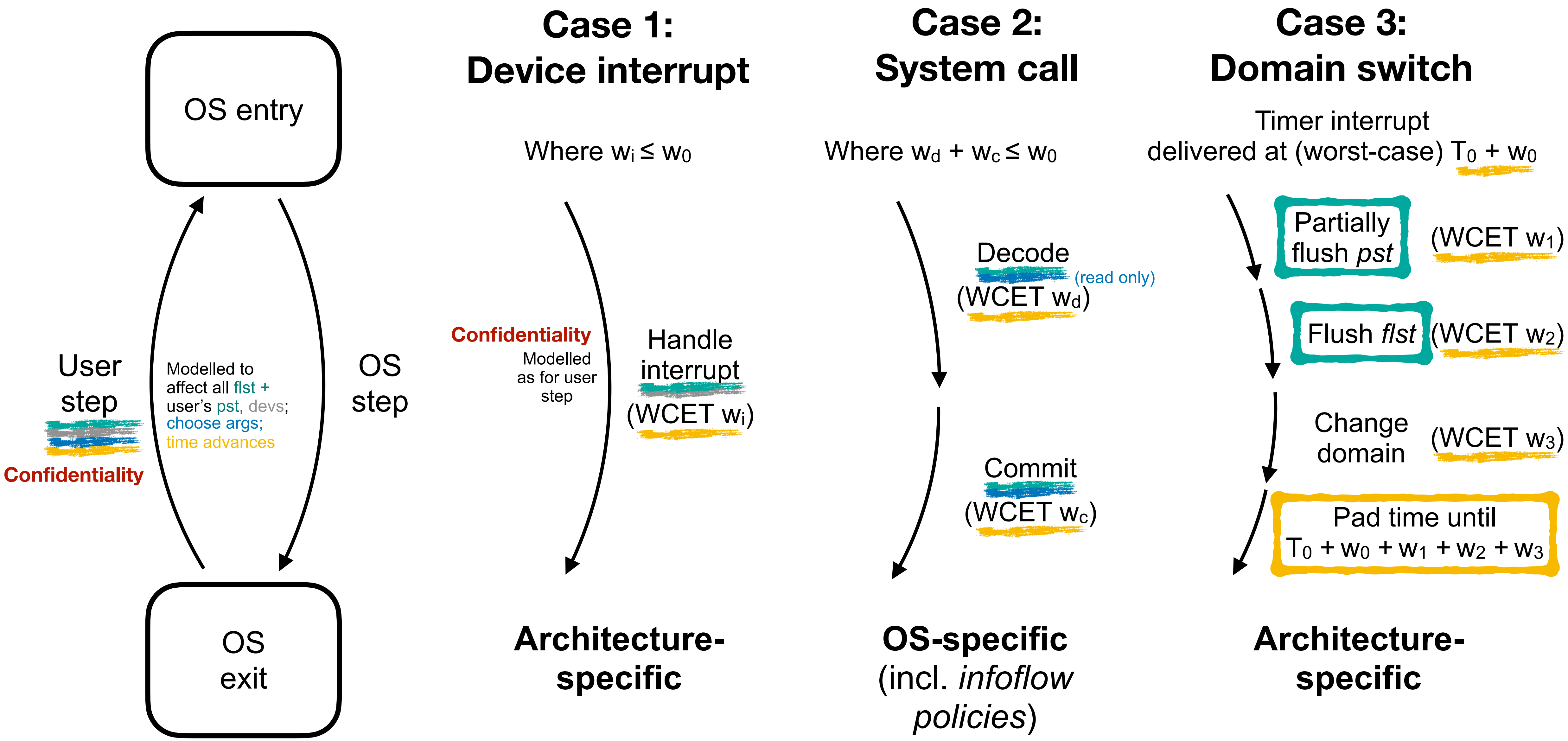
OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Requirements

(In addition to WCETs)

Transition system



OS entry

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

Confidentiality

OS step

OS exit

Case 1: Device interrupt

Where $w_i \leq w_0$

Confidentiality

Modelled as for user step

Handle interrupt (WCET w_i)

Architecture-specific

Case 2: System call

Where $w_d + w_c \leq w_0$

Decode (read only) (WCET w_d)

Commit (WCET w_c)

OS-specific (incl. *infoflow policies*)

Case 3: Domain switch

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

Partially flush *pst* (WCET w_1)

Flush *flst* (WCET w_2)

Change domain (WCET w_3)

Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

Security proof approach

No channels!

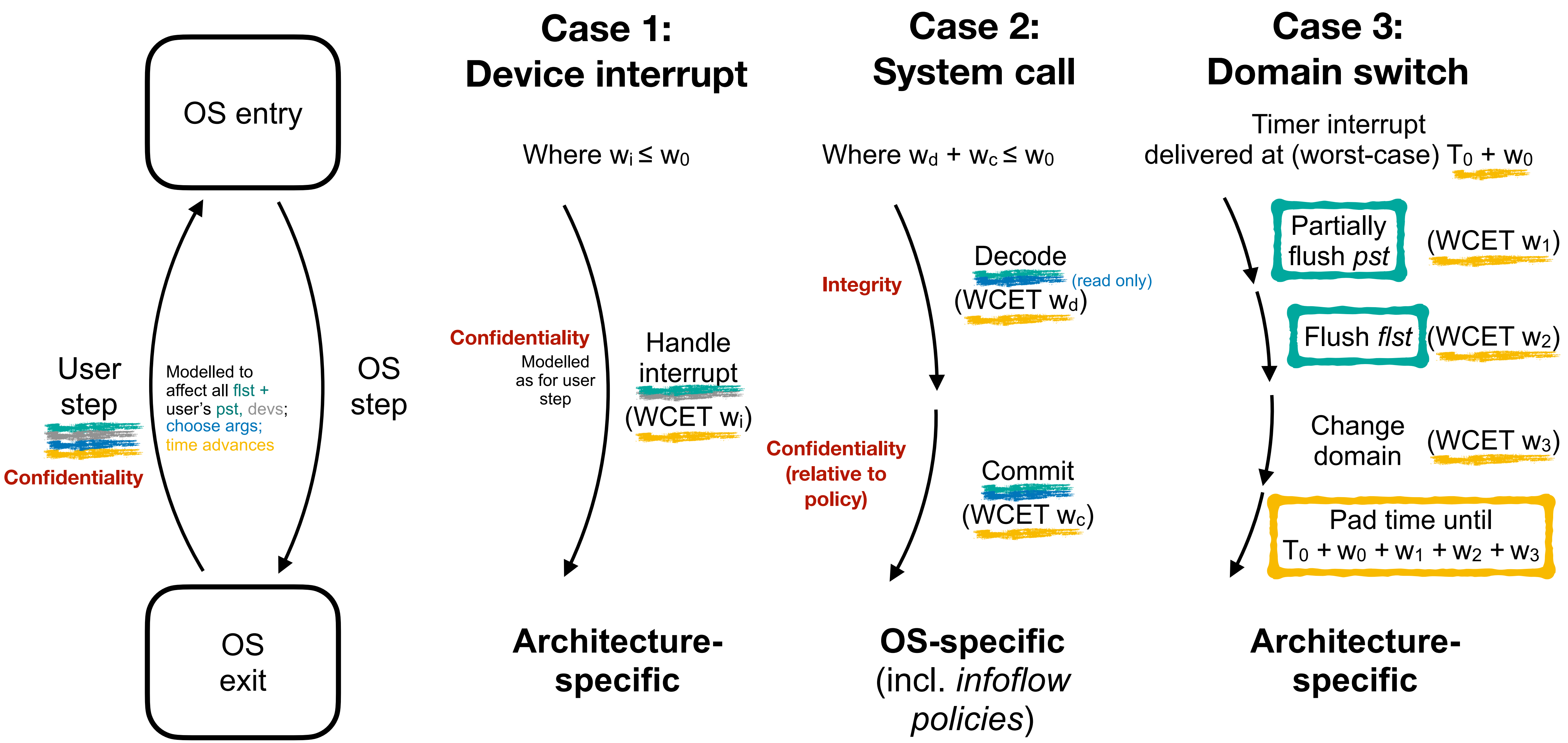
OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Requirements

(In addition to WCETs)

Transition system



OS entry

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

Confidentiality

OS step

OS exit

Case 1: Device interrupt

Where $w_i \leq w_0$

Confidentiality

Modelled as for user step

Handle interrupt (WCET w_i)

Architecture-specific

Case 2: System call

Where $w_d + w_c \leq w_0$

Integrity

Decode (read only) (WCET w_d)

Confidentiality (relative to policy)

Commit (WCET w_c)

OS-specific (incl. *infoflow policies*)

Case 3: Domain switch

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

Partially flush *pst* (WCET w_1)

Flush *flst* (WCET w_2)

Change domain (WCET w_3)

Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

Security proof approach

No channels!

OS 🤝 HW

Microarchitecture

Devices

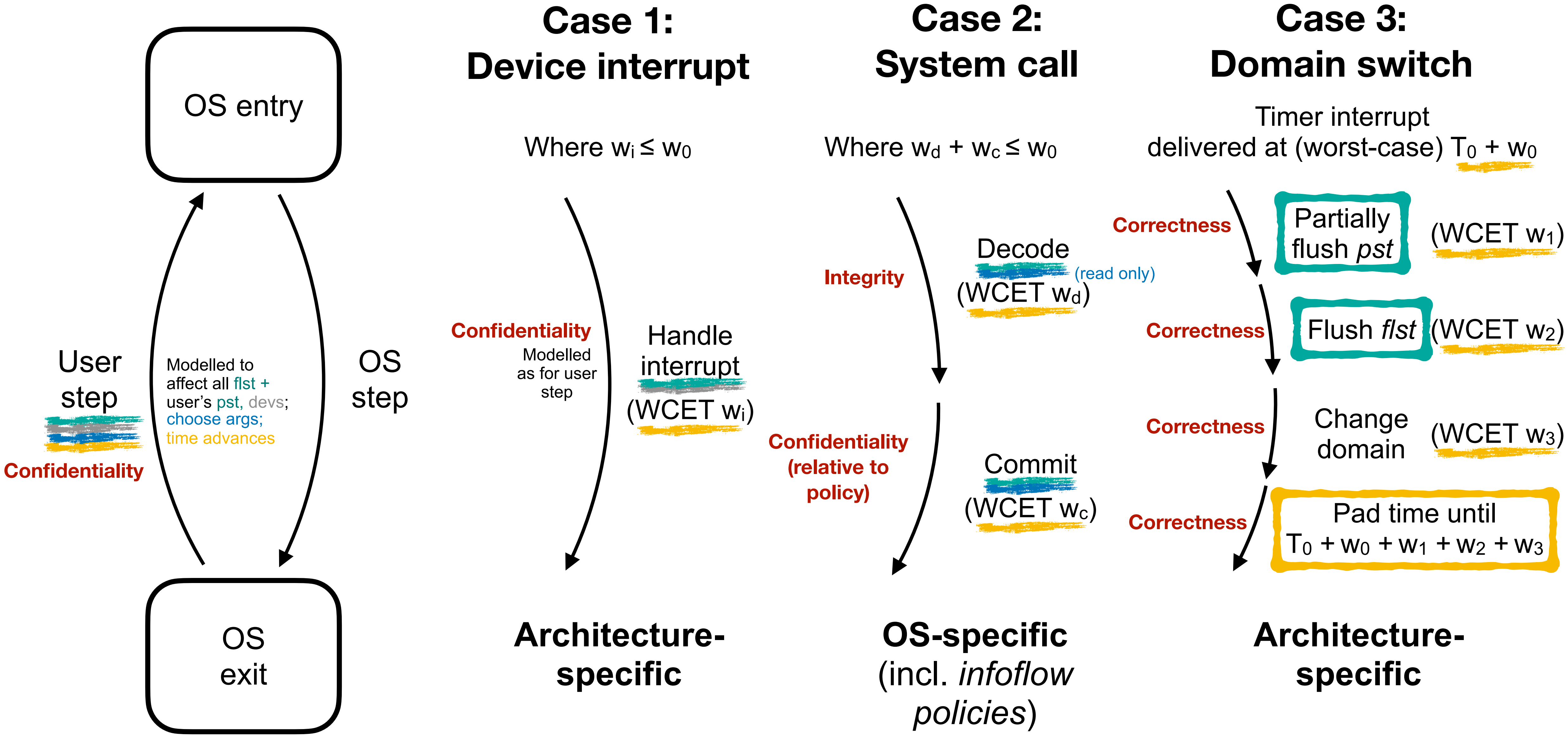
Policy-determining state

Time

Requirements

(In addition to WCETs)

Transition system



OS entry

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

Confidentiality

OS step

OS exit

Case 1: Device interrupt

Where $w_i \leq w_0$

Confidentiality

Modelled as for user step

Handle interrupt (WCET w_i)

Architecture-specific

Case 2: System call

Where $w_d + w_c \leq w_0$

Integrity

Decode (WCET w_d) (read only)

Confidentiality (relative to policy)

Commit (WCET w_c)

OS-specific (incl. *infoflow policies*)

Case 3: Domain switch

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

Correctness

Partially flush *pst* (WCET w_1)

Correctness

Flush *flst* (WCET w_2)

Correctness

Change domain (WCET w_3)

Correctness

Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

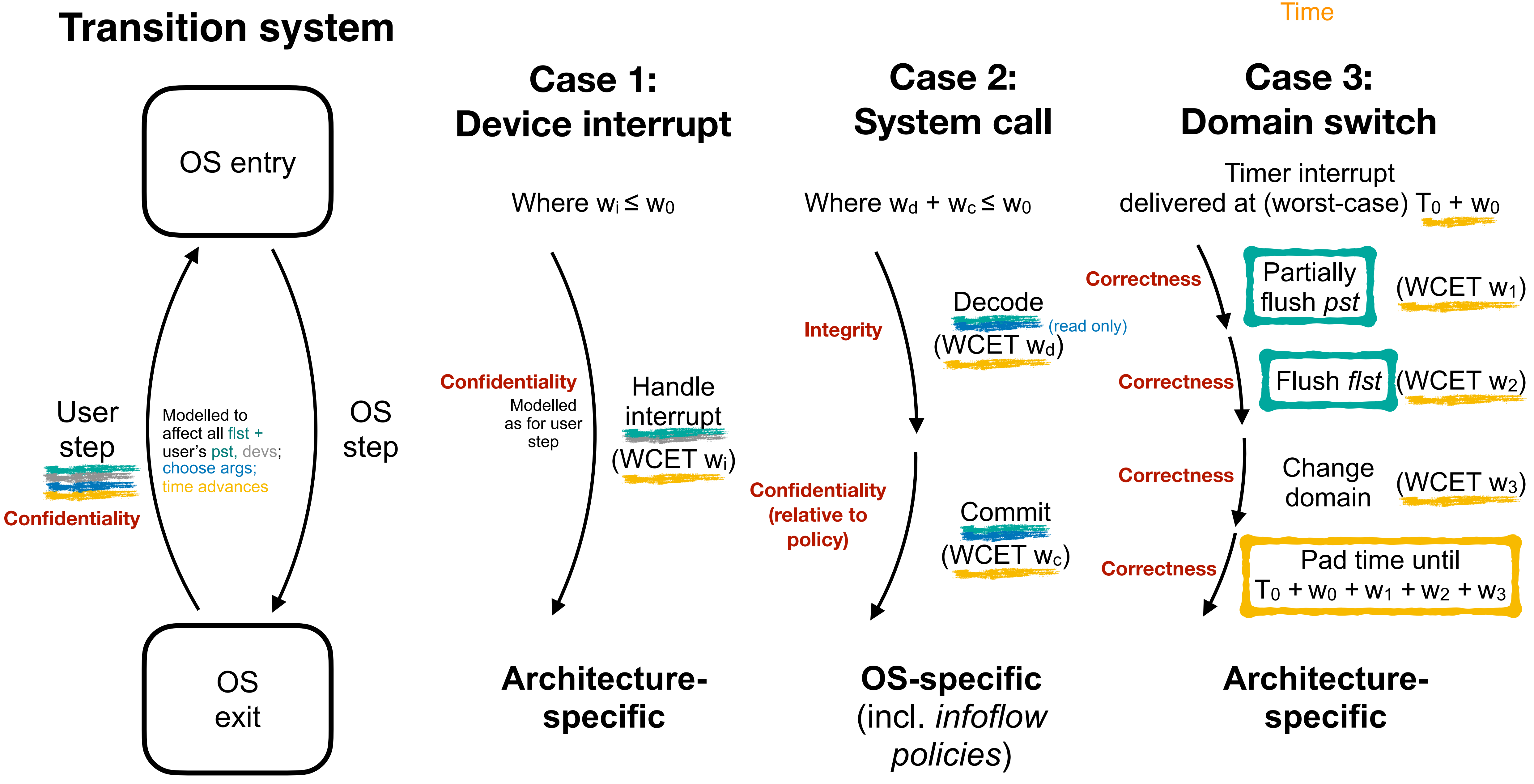
Security proof approach

No channels!

OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Requirements
(In addition to WCETs)



Transition system

OS entry

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

Confidentiality

OS step

OS exit

Case 1: Device interrupt

Where $w_i \leq w_0$

Confidentiality

Modelled as for user step

Handle interrupt (WCET w_i)

Architecture-specific

Case 2: System call

Where $w_d + w_c \leq w_0$

Integrity

Decode (read only) (WCET w_d)

Confidentiality (relative to policy)

Commit (WCET w_c)

OS-specific (incl. *infoflow policies*)

Case 3: Domain switch

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

Correctness

Partially flush *pst* (WCET w_1)

Correctness

Flush *flst* (WCET w_2)

Correctness

Change domain (WCET w_3)

Correctness

Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

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

Security proof approach

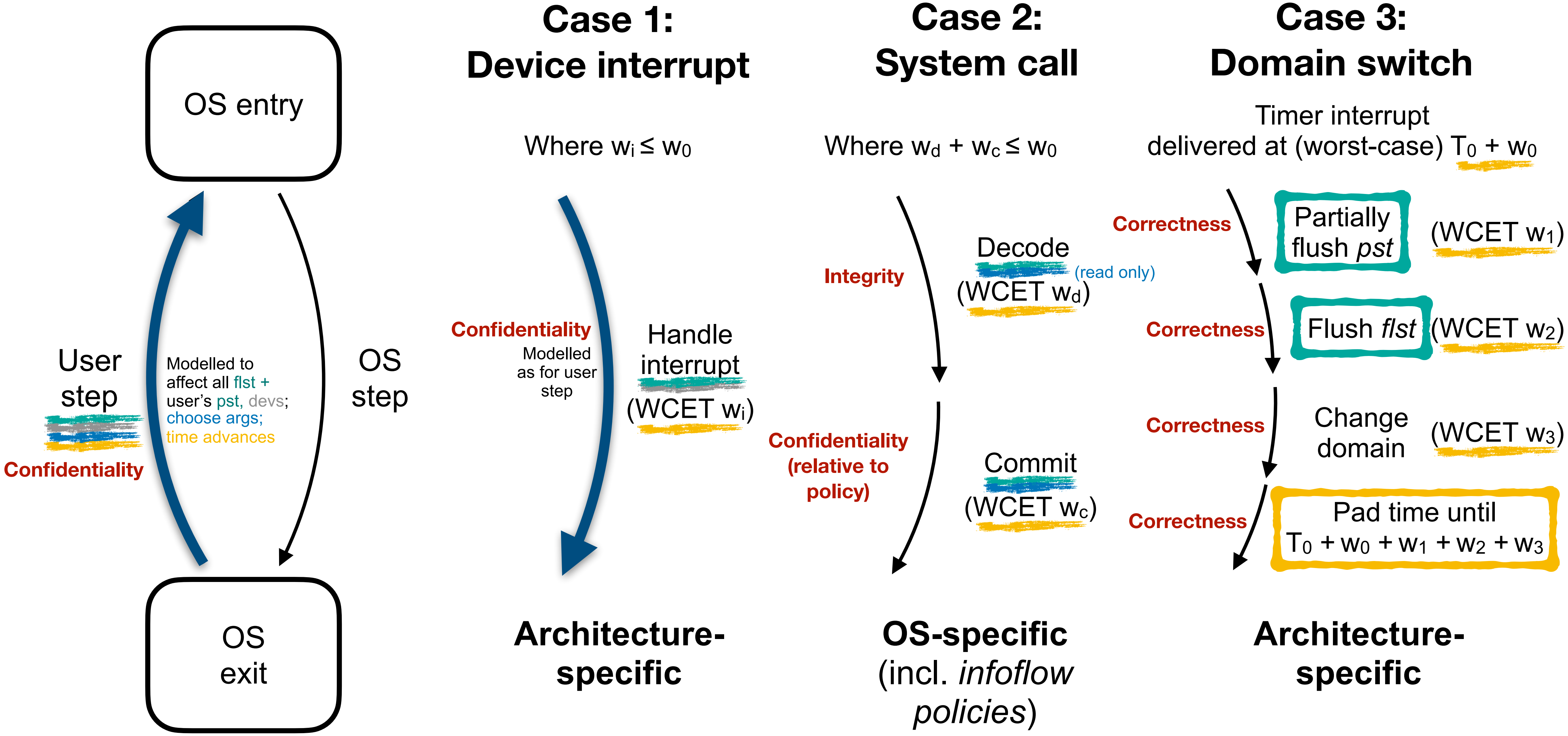
No channels!

OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Requirements
(In addition to WCETs)

Transition system



OS entry

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

Confidentiality

OS step

OS exit

Case 1: Device interrupt

Where $w_i \leq w_0$

Confidentiality

Modelled as for user step

Handle interrupt (WCET w_i)

Architecture-specific

Case 2: System call

Where $w_d + w_c \leq w_0$

Integrity

Decode (read only) (WCET w_d)

Confidentiality (relative to policy)

Commit (WCET w_c)

OS-specific (incl. *infoflow policies*)

Case 3: Domain switch

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

Correctness

Partially flush *pst* (WCET w_1)

Correctness

Flush *flst* (WCET w_2)

Correctness

Change domain (WCET w_3)

Correctness

Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

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

Security proof approach

No channels!

OS 🤝 HW

Microarchitecture

Devices

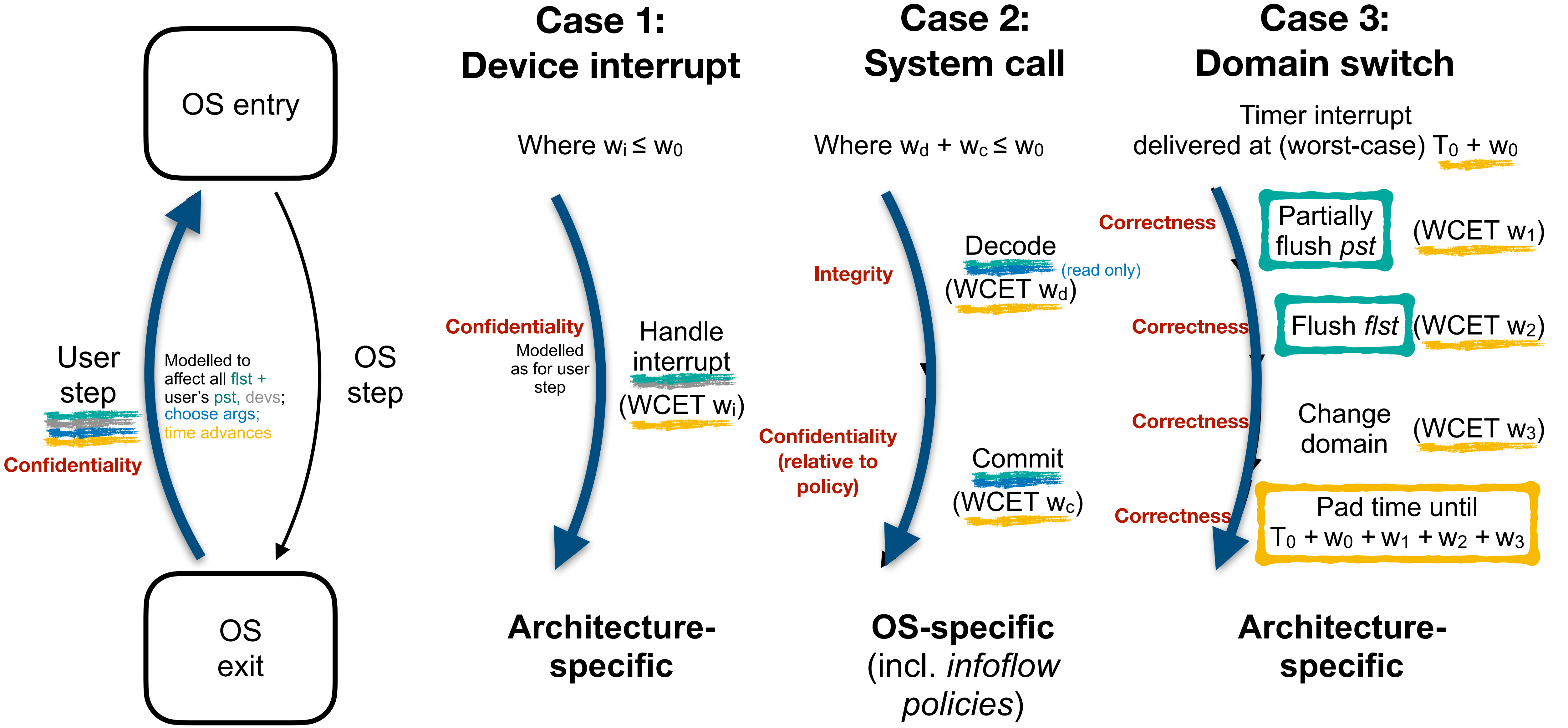
Policy-determining state

Time

Requirements

(In addition to WCETs)

Transition system



OS entry

User step

Modelled to affect all flst + user's pst, devs; choose args; time advances

Confidentiality

OS step

OS exit

Case 1: Device interrupt

Where $w_i \leq w_0$

Confidentiality

Modelled as for user step

Handle interrupt (WCET w_i)

Architecture-specific

Case 2: System call

Where $w_d + w_c \leq w_0$

Integrity

Decode (read only) (WCET w_d)

Confidentiality (relative to policy)

Commit (WCET w_c)

OS-specific (incl. *infoflow policies*)

Case 3: Domain switch

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

Correctness

Partially flush *pst* (WCET w_1)

Correctness

Flush *flst* (WCET w_2)

Correctness

Change domain (WCET w_3)

Correctness

Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

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

Security proof approach

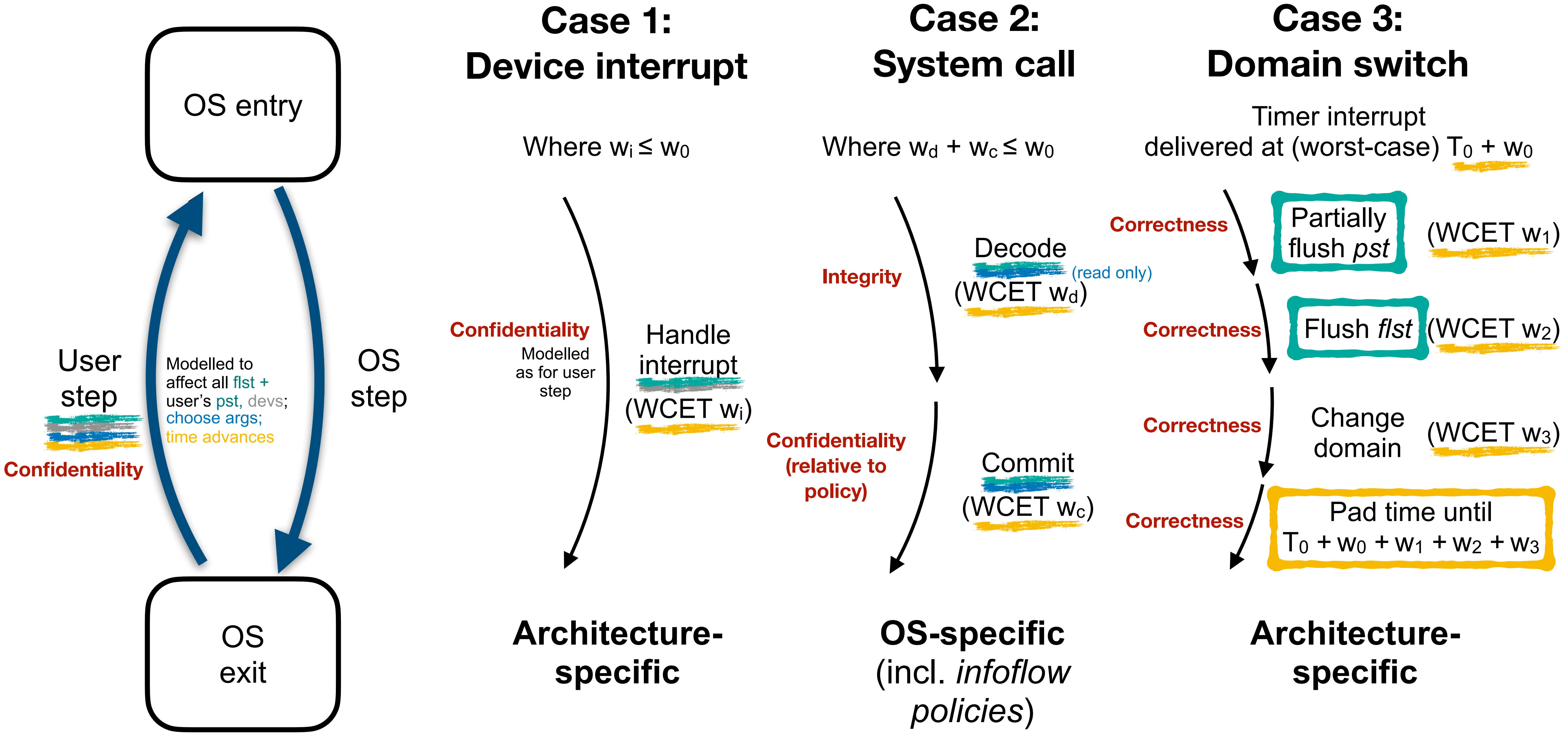
No channels!

OS 🤝 HW

Microarchitecture
Devices
Policy-determining state
Time

Requirements
(In addition to WCETs)

Transition system



OS entry

User step

Confidentiality

Modelled to affect all flst + user's pst, devs; choose args; time advances

OS step

OS exit

Case 1: Device interrupt

Where $w_i \leq w_0$

Confidentiality

Modelled as for user step

Handle interrupt

(WCET w_i)

Architecture-specific

Case 2: System call

Where $w_d + w_c \leq w_0$

Integrity

Decode

(read only)

(WCET w_d)

Confidentiality (relative to policy)

Commit

(WCET w_c)

OS-specific (incl. *infoflow policies*)

Case 3: Domain switch

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

Correctness

Partially flush *pst*

(WCET w_1)

Correctness

Flush *flst*

(WCET w_2)

Correctness

Change domain

(WCET w_3)

Correctness

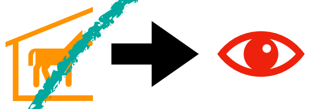
Pad time until $T_0 + w_0 + w_1 + w_2 + w_3$

Architecture-specific

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

How to formalise an OS enforces *time protection*?

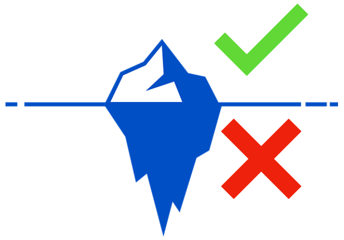
Versus threat scenario:
trojan and spy



No channels!

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*)

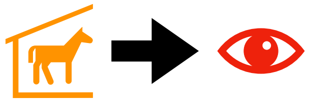
2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

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*?

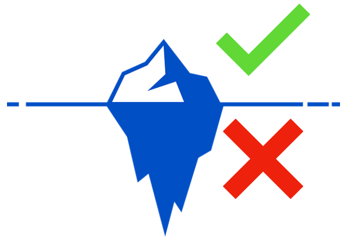
Versus threat scenario:
trojan and spy



No channels!

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:

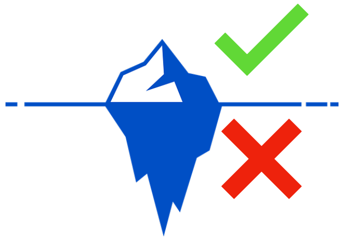
1. OS security model imposing requirements on relevant parts of OS.
(Intended for seL4, but *generic*)

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

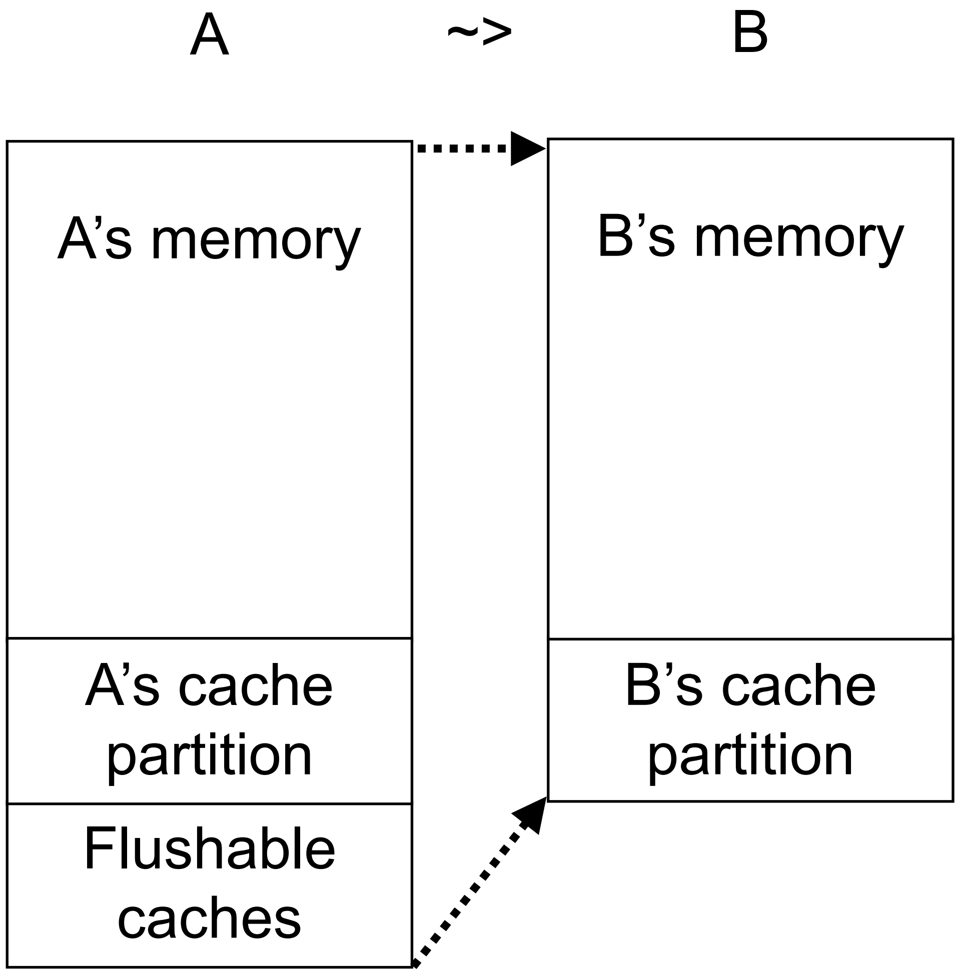
3. Proof our security property holds if OS model's requirements hold.

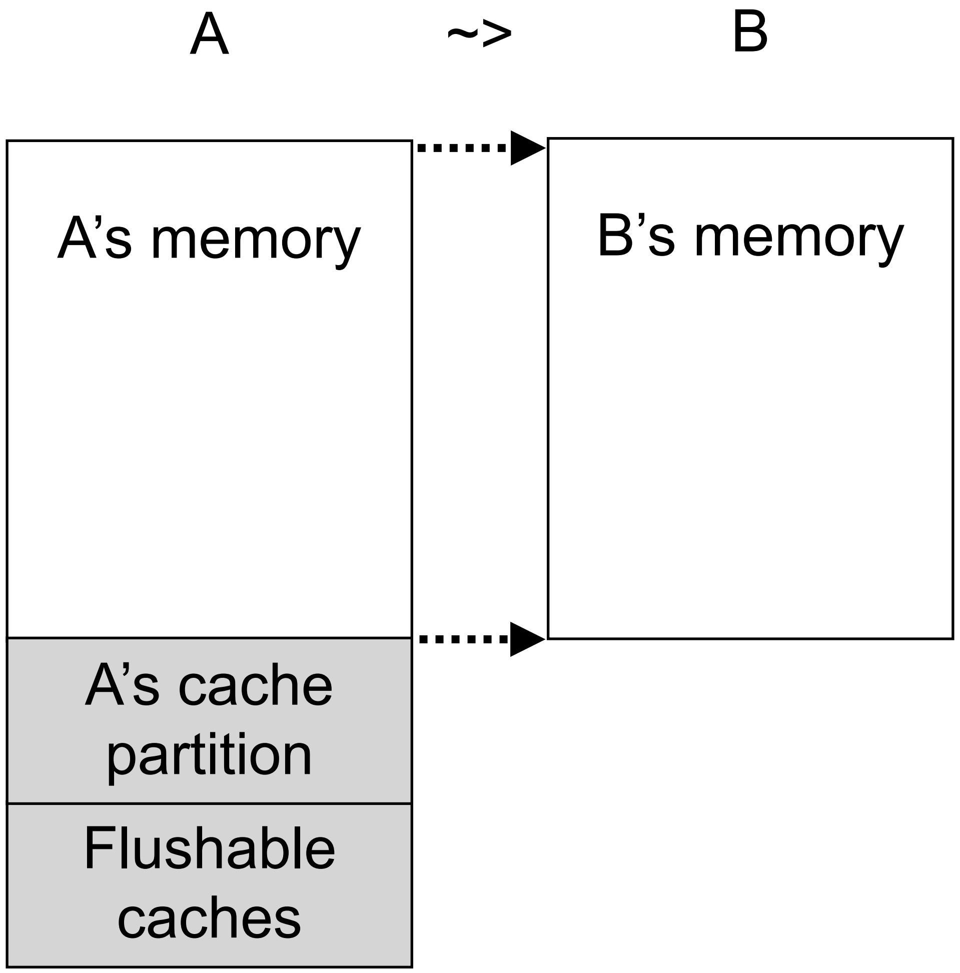
4. Basic instantiation of OS model exercising dynamic policy.

OS security property



Recall:



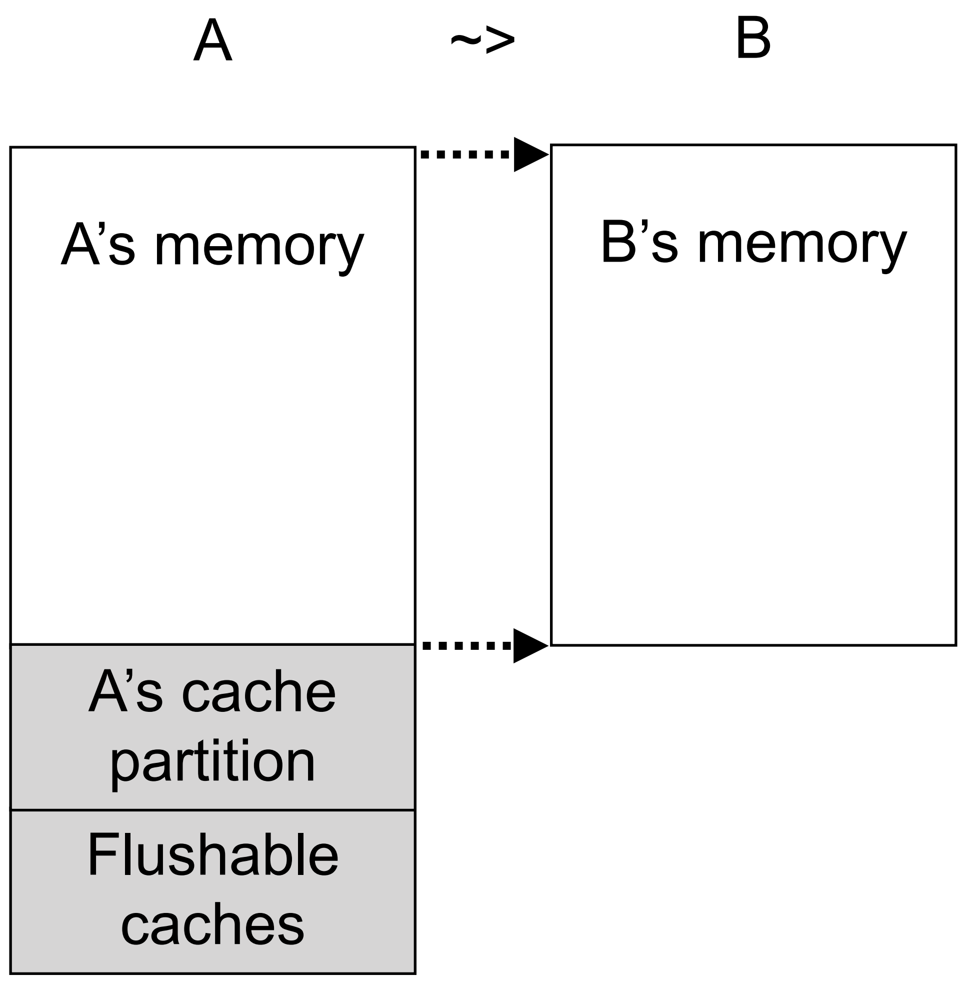


From prior seL4 infoflow proofs [Murray et al. 2012, 2013]: *"all or nothing" policies*

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

OS security property

Our infoflow policies:

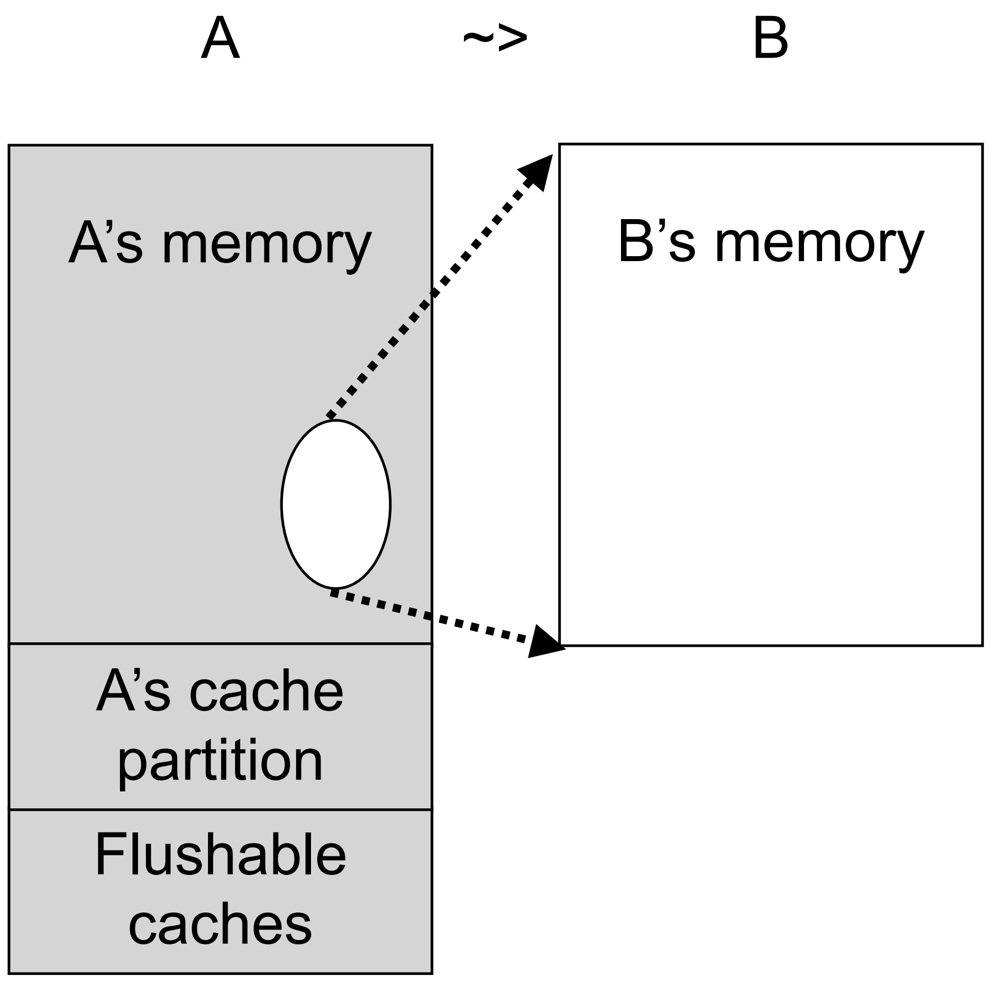


For time protection, need *spatial precision* to *allow some flows* but *exclude others*

OS security property

Our infoflow policies:

- *Arbitrary* spatial precision



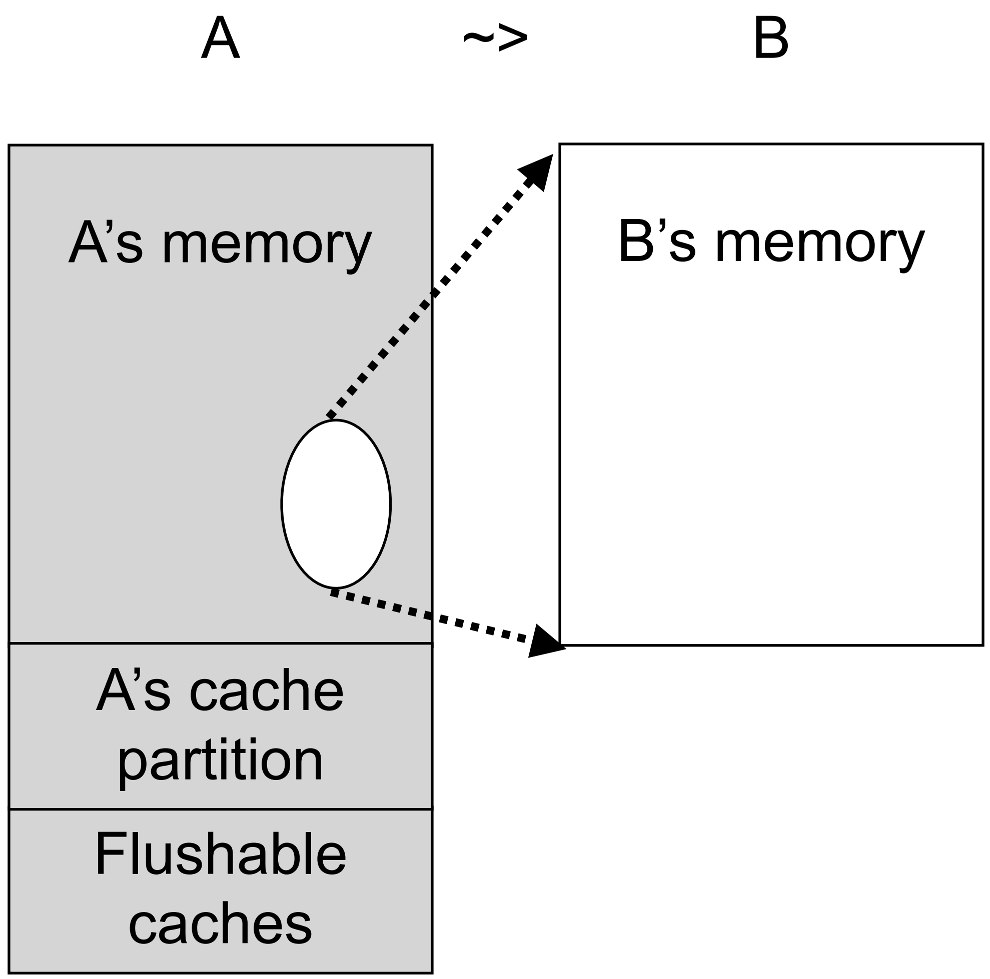
For time protection, need *spatial precision* to *allow some flows* but *exclude others*

OS security property

Our infoflow policies:

- *Arbitrary* spatial precision
- *Policy channels* specified as *state relations*: $s \overset{|A \rightsquigarrow B|}{\sim} t$

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



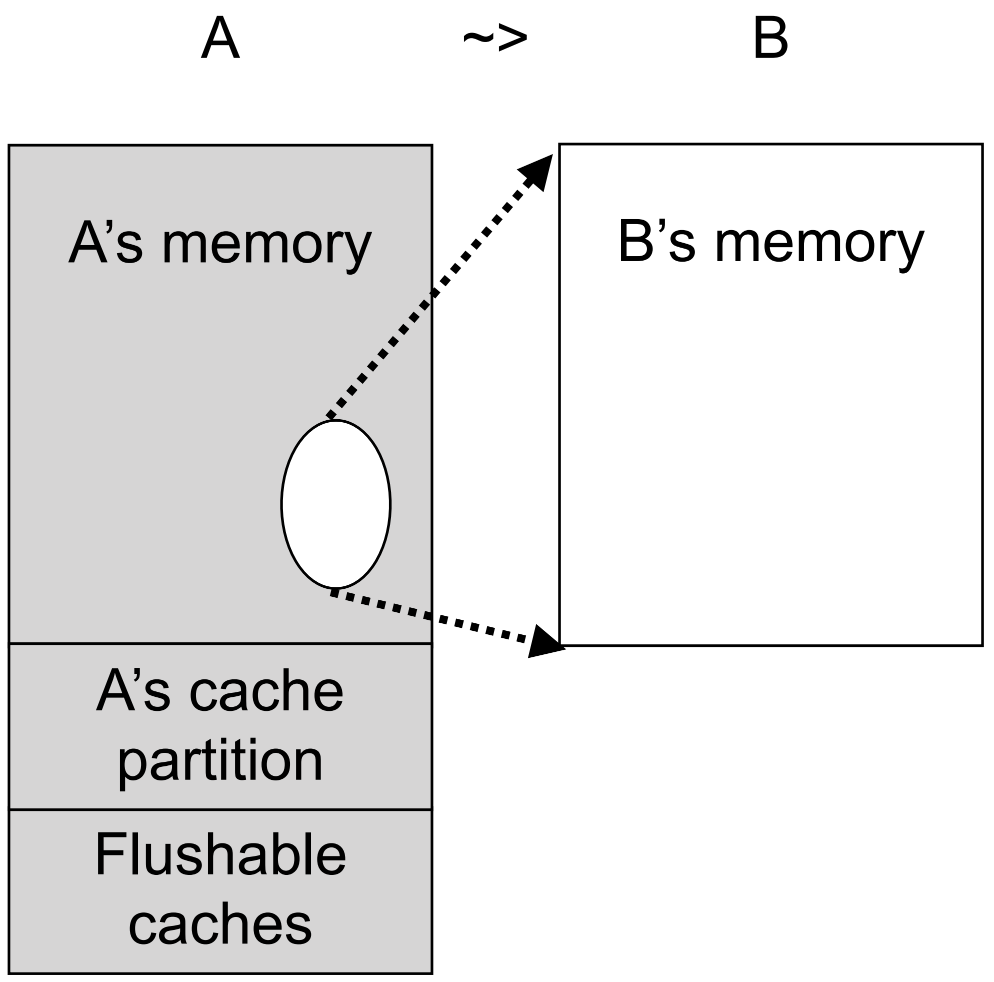
For time protection, need *spatial precision* to *allow some flows* but *exclude others*

OS security property

Our infoflow policies:

- *Arbitrary* spatial precision
- *Policy channels* specified as *state relations*: $s \overset{|A \rightsquigarrow B|}{\sim} t$

  If $\overset{|A \rightsquigarrow B|}{\sim}$ equates part of A, then info flow is allowed from there to B.
- Also arbitrary *temporal precision*



A ~> B

A's memory

B's memory

A's cache partition

Flushable caches

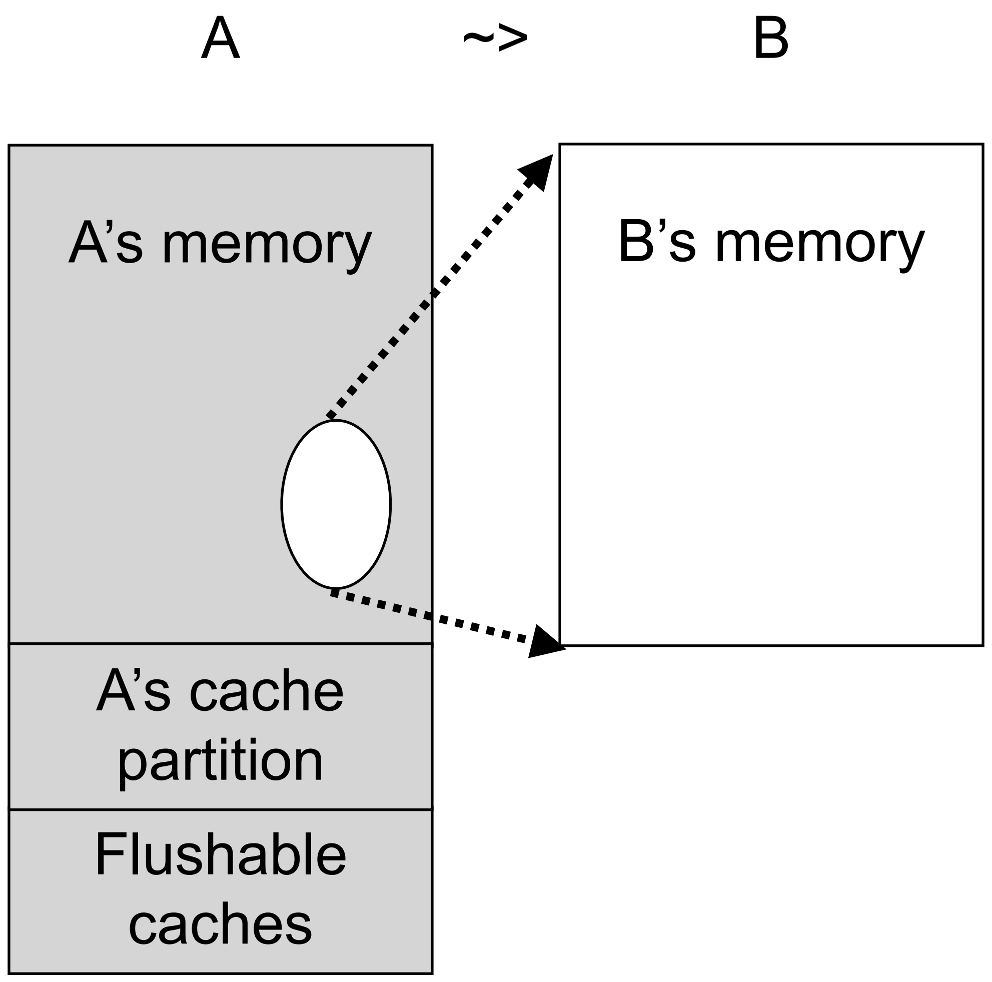
For time protection, need *spatial precision* to *allow some flows* but *exclude others*

OS security property

Our infoflow policies:

- *Arbitrary* spatial precision
- *Policy channels* specified as *state relations*: $s \overset{|A \rightsquigarrow B|}{\sim} t$

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



A ~> B

A's memory

B's memory

A's cache partition

Flushable caches

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

OS security property

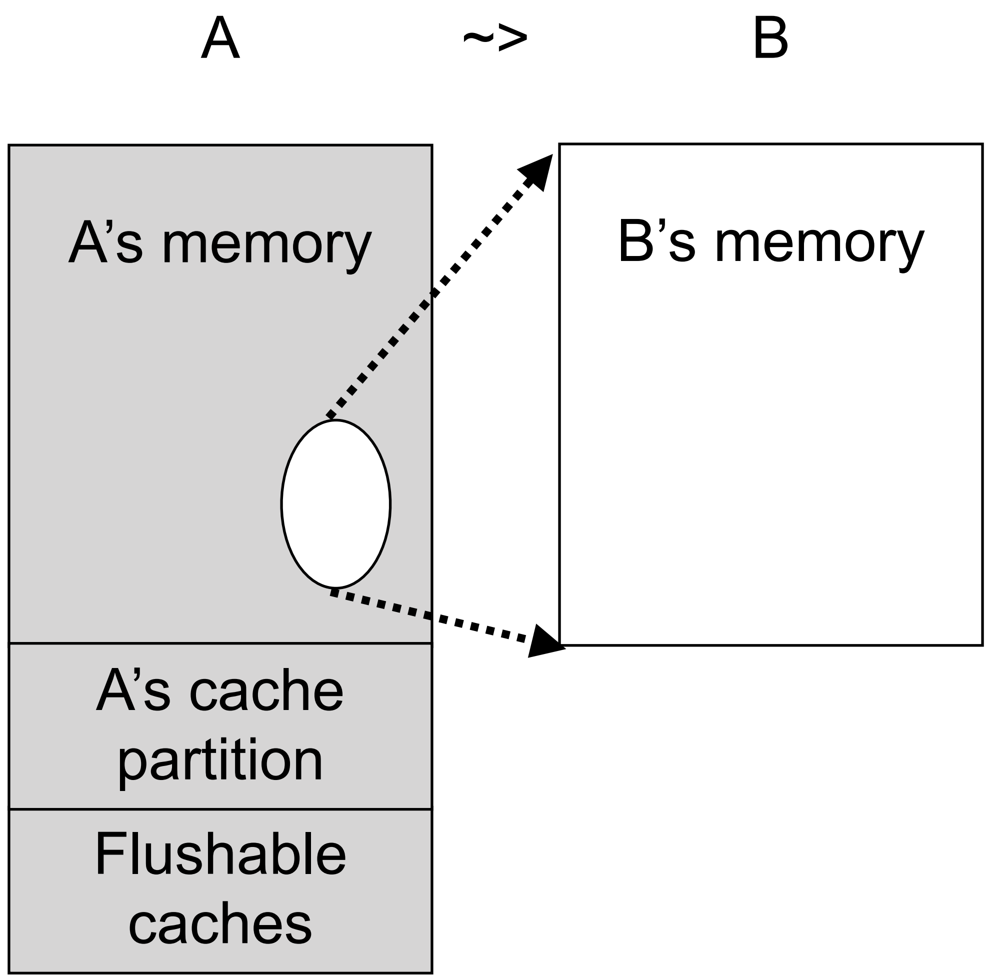
Our infoflow policies:

- *Arbitrary* spatial precision
- *Policy channels* specified as *state relations*: $s \overset{|A \rightsquigarrow B|}{\sim} t$

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

?

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



A ~> B

A's memory

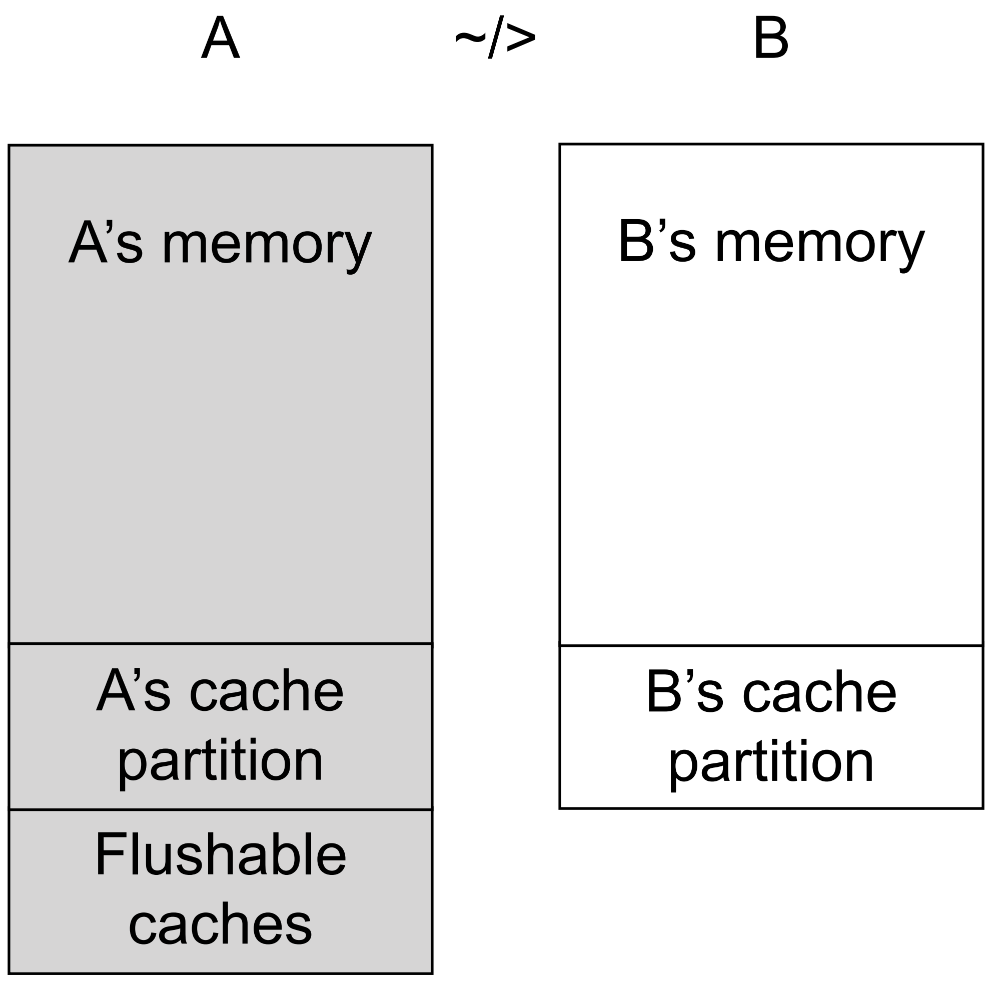
B's memory

A's cache partition

Flushable caches

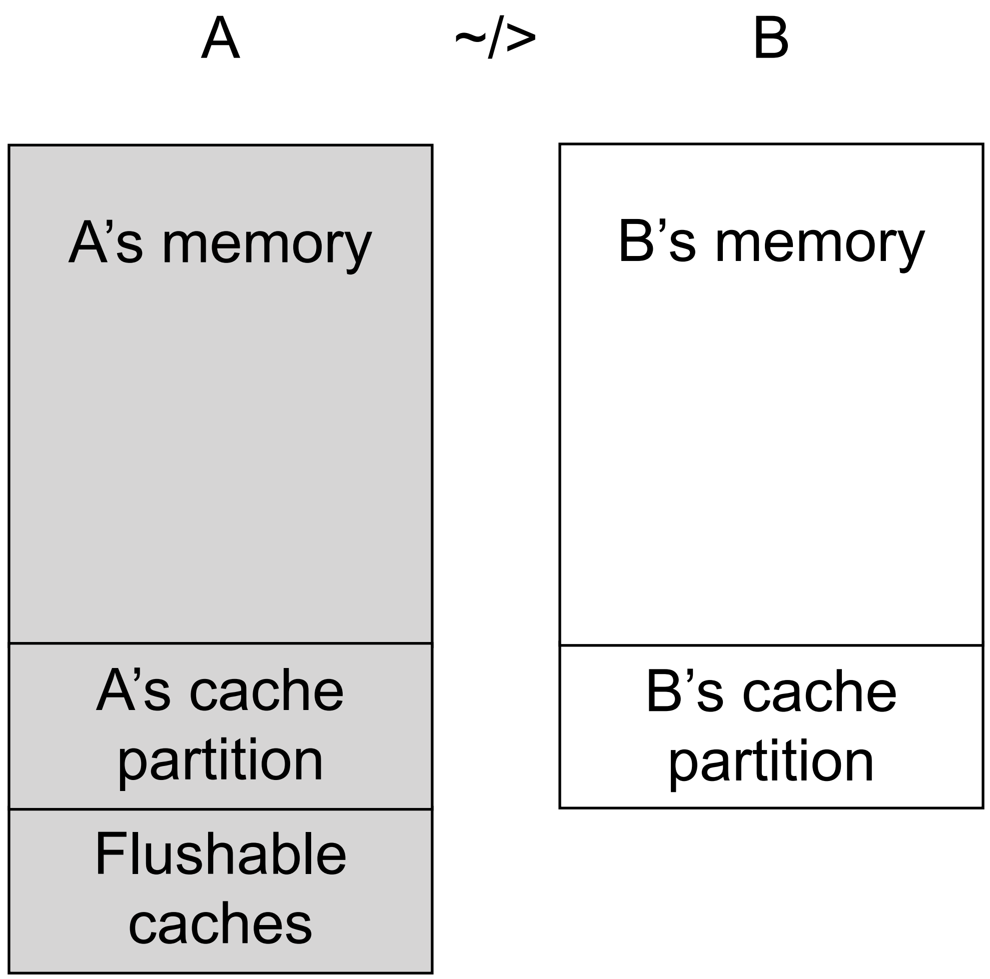
For time protection, need *spatial precision* to *allow some flows* but *exclude others*

Dynamic policy, observer relativity



Dynamic policy, observer relativity

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



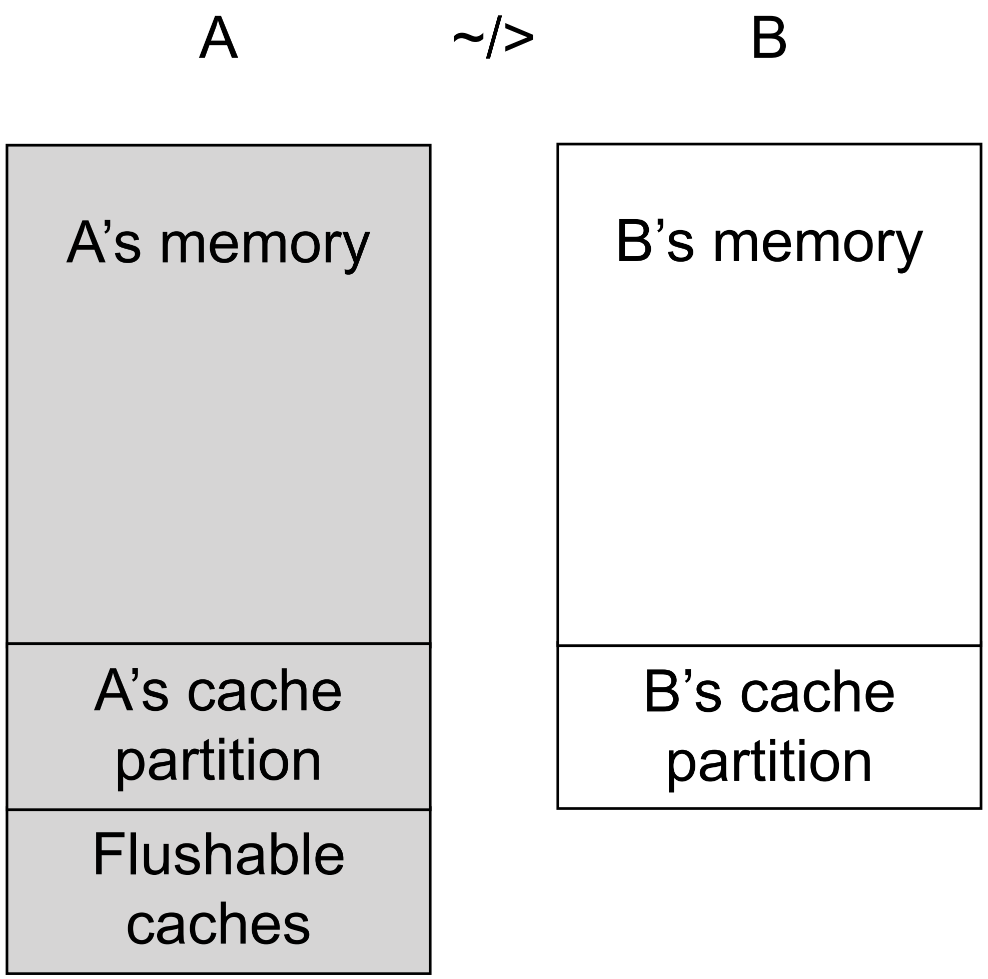
Dynamic policy, observer relativity

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

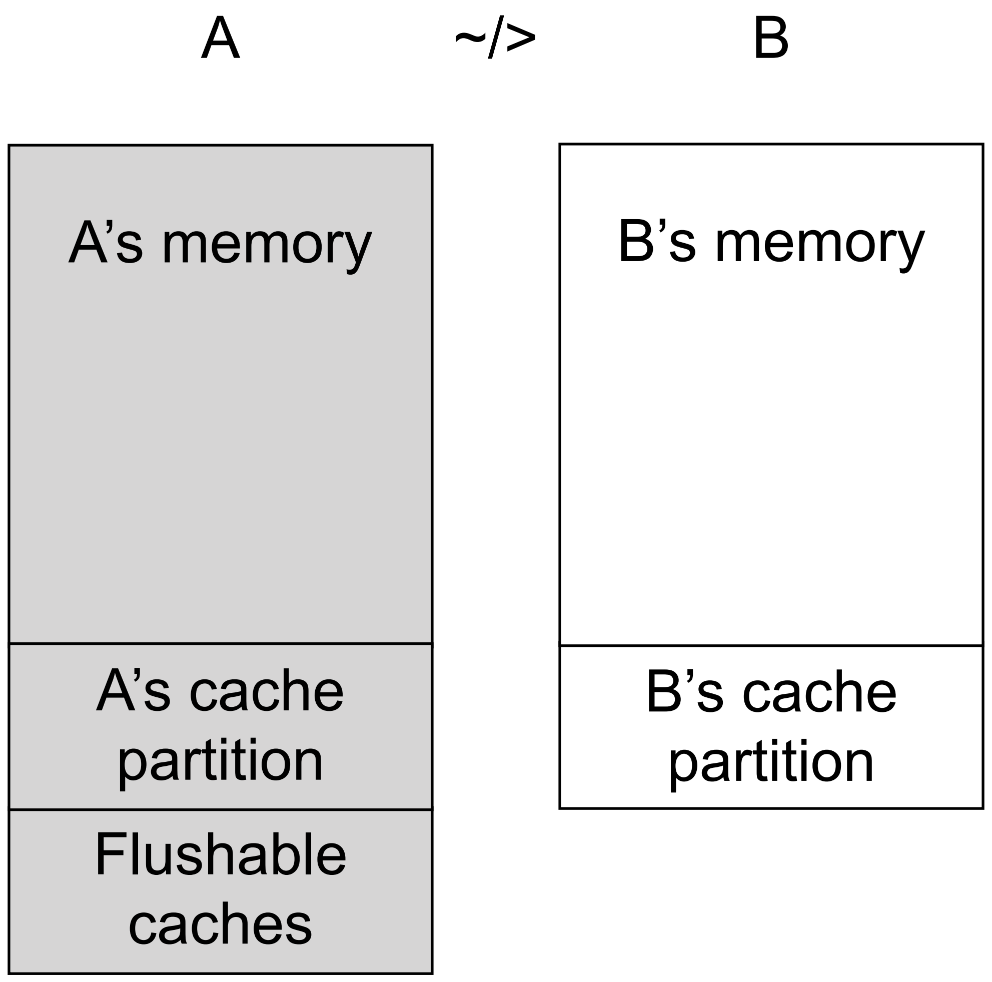
Two basic system calls:
Subscribe(*d***)**, **Broadcast()**

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()**

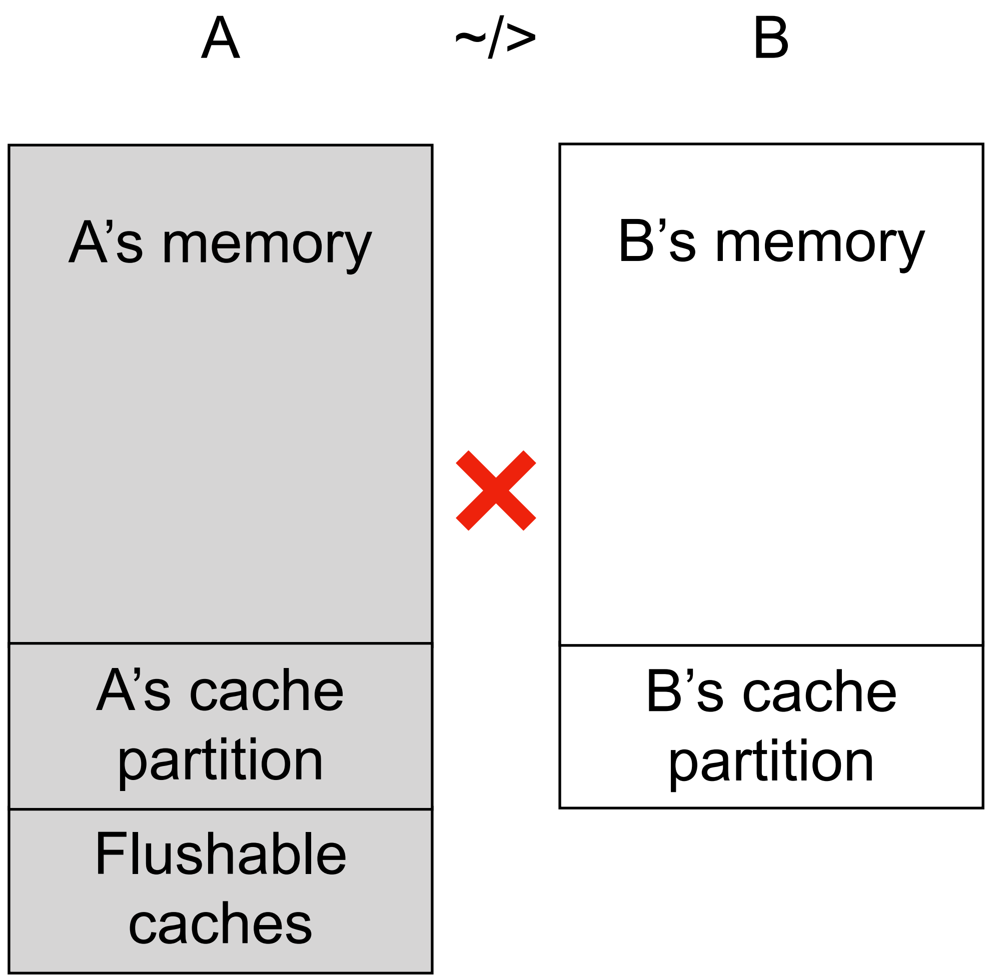
1. Dynamic policy: A ~> B ?

✓ 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 ?

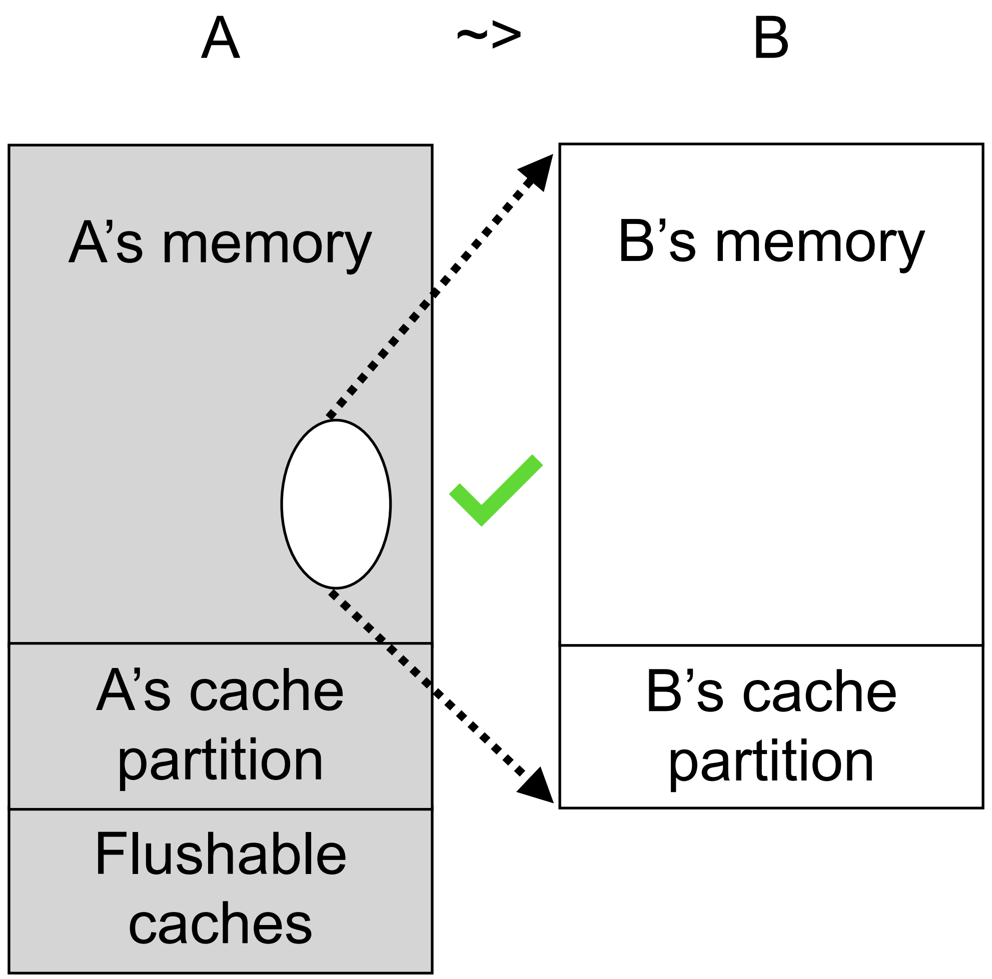
✓ Only when A calls

- **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 ?

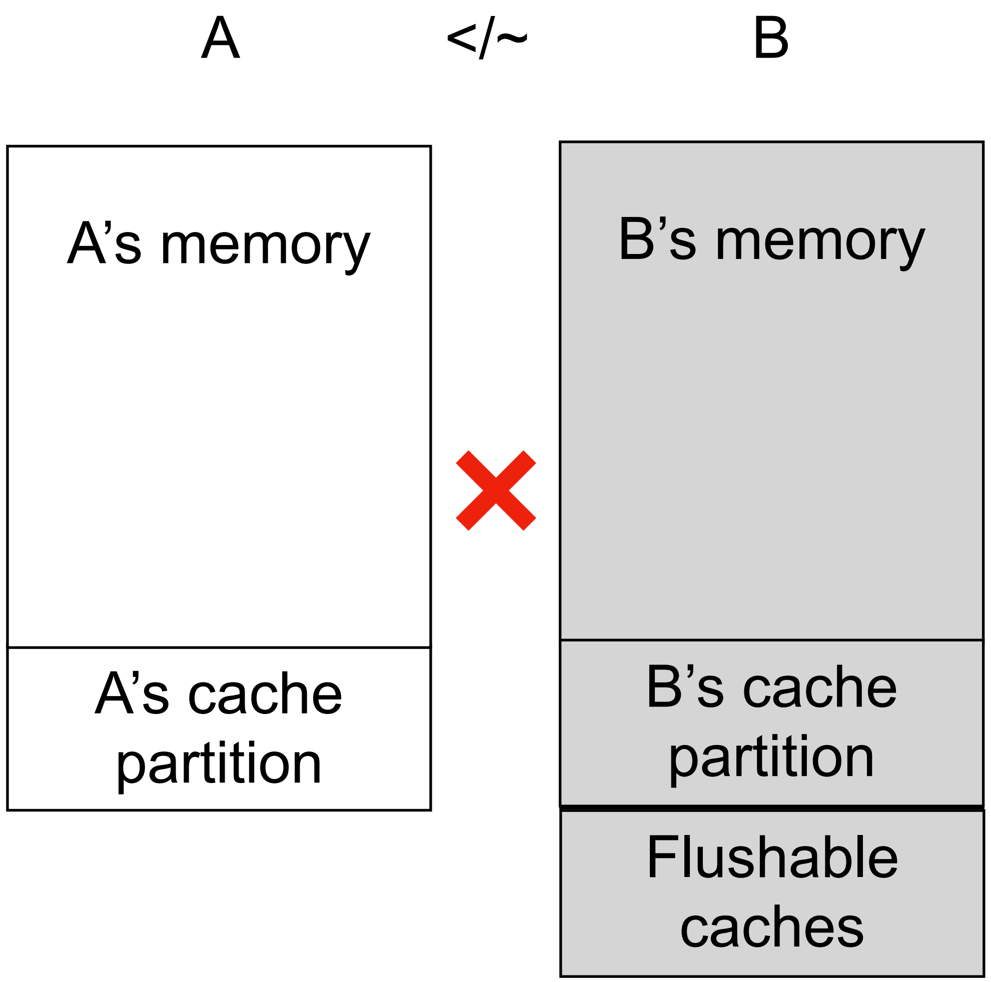
✓ Only when A calls

- **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 ?

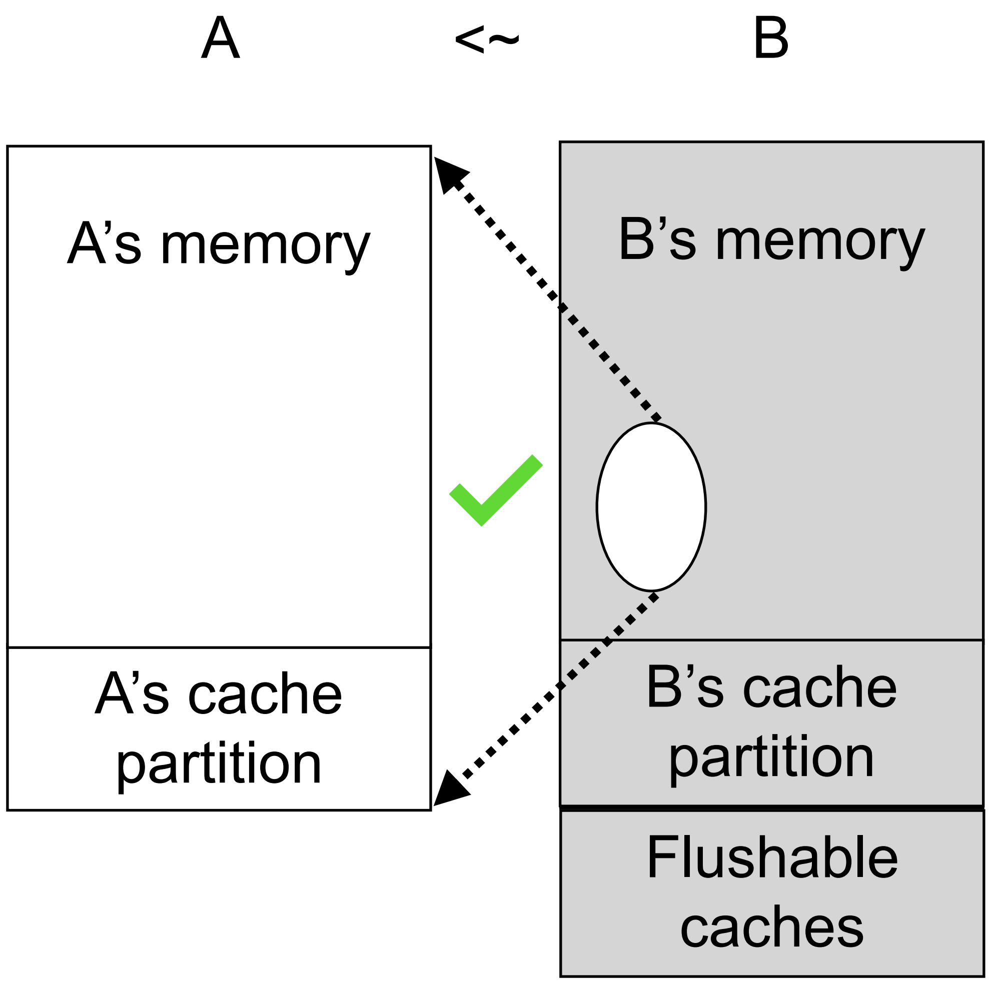
✓ Only when A calls

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

✗ Otherwise, no channel.

- Example:

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



A <~ B

A's memory

A's cache partition

✓

B's memory

B's cache partition

Flushable caches

Dynamic policy, observer relativity

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

1. Dynamic policy: A ~> B ?

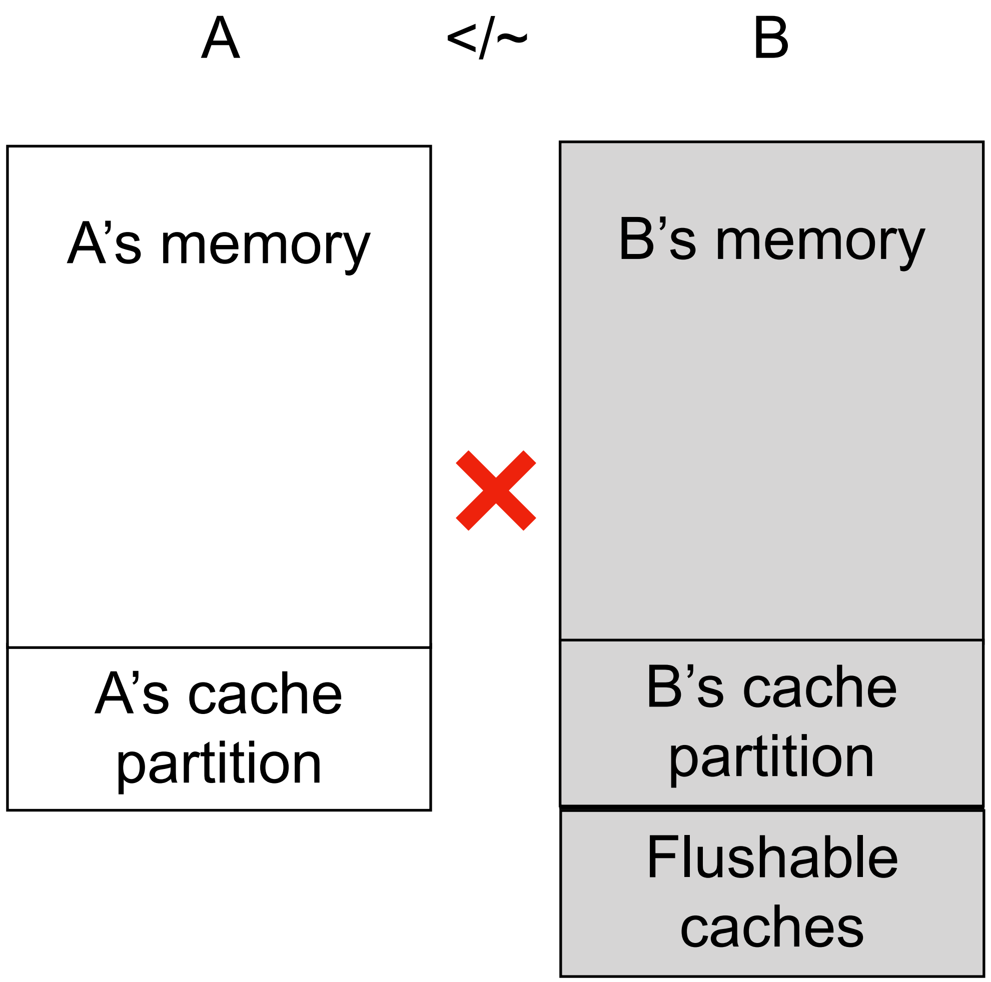
✓ Only when A calls

- **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 ?

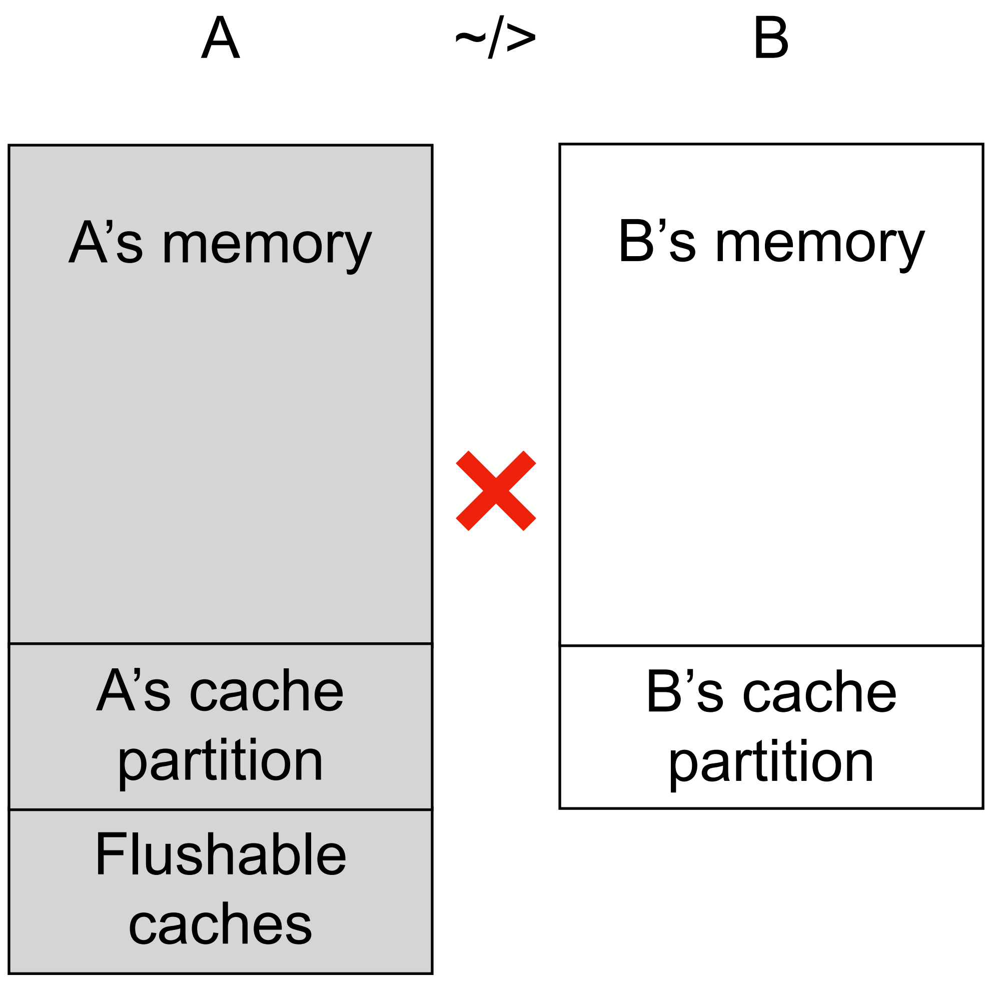
✓ Only when A calls

- **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()**

2. Property must be observer relative!

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!

Dynamic policy, observer relativity

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

1. Dynamic policy: A ~> B ?

✓ Only when A calls

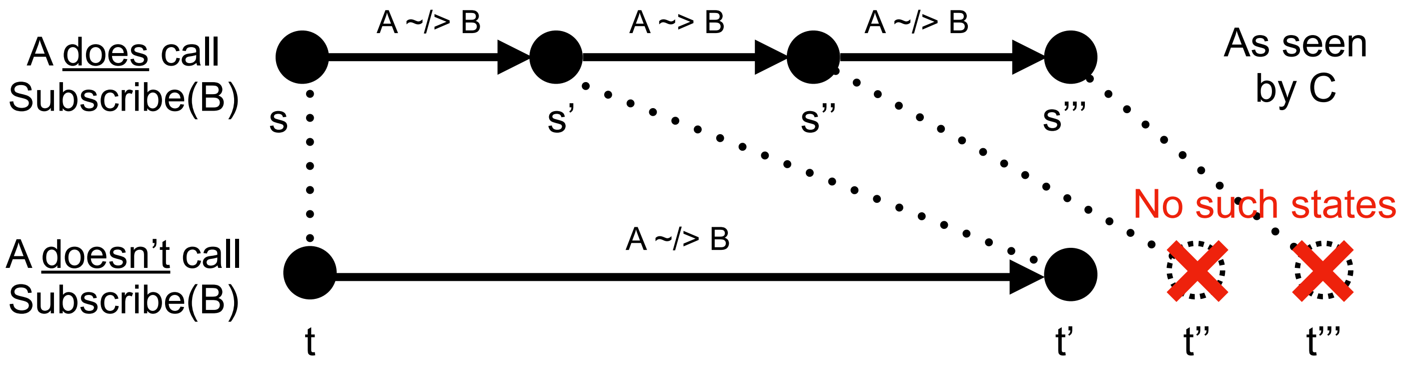
- **Subscribe(**B**)**, or
- **Broadcast()** 1st time after B called **Subscribe(**A**)**.

✗ 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!



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**)**.

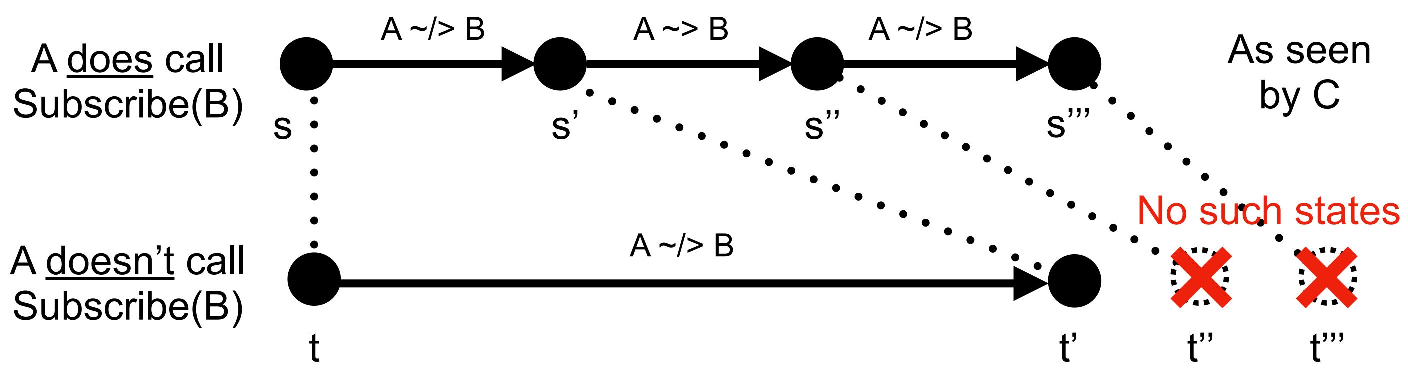
✗ Otherwise, no channel.

- Example:

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

2. Property must be observer relative!

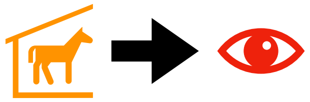
- If not, can't prove the (bisimulation) property for unrelated user C!



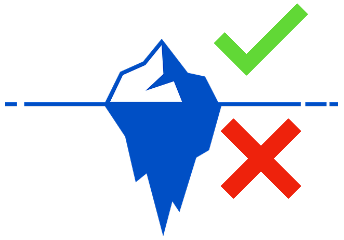
- *Solution*: C's property must treat *states* (in the state machine) as observable only whenever
  - C is running, or
  - When *d* is running, *d* ~> C.

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*)

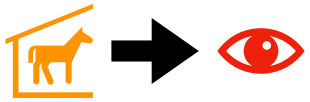
2. OS security property that is dynamic; this makes it observer relative.
(Improving on seL4's of [Murray et al. 2012])

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

4. Basic instantiation of OS model exercising dynamic policy.

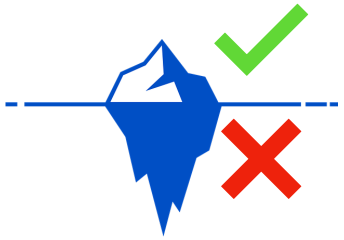
How to formalise an OS enforces *time protection*?

Versus threat scenario:
trojan and spy



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.

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.