School of Computer Science & Engineering

Trustworthy Systems Group

Two interfaces for seL4 kernel verification: Time Protection and the Kernel–Userland Gap

Isabelle Workshop, Jul'26, Lisbon

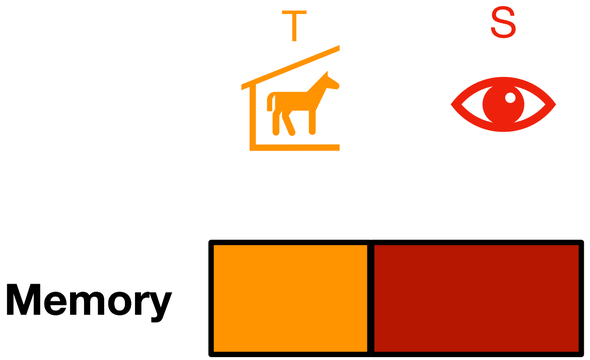
Dr Rob Sison

Senior Research Associate, UNSW Sydney

r.sison@unsw.edu.au

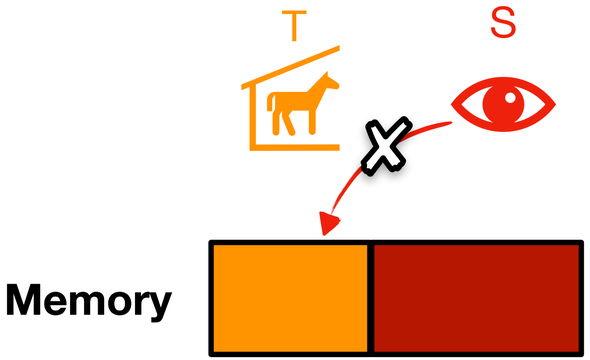
I: Time Protection

What is Time Protection?



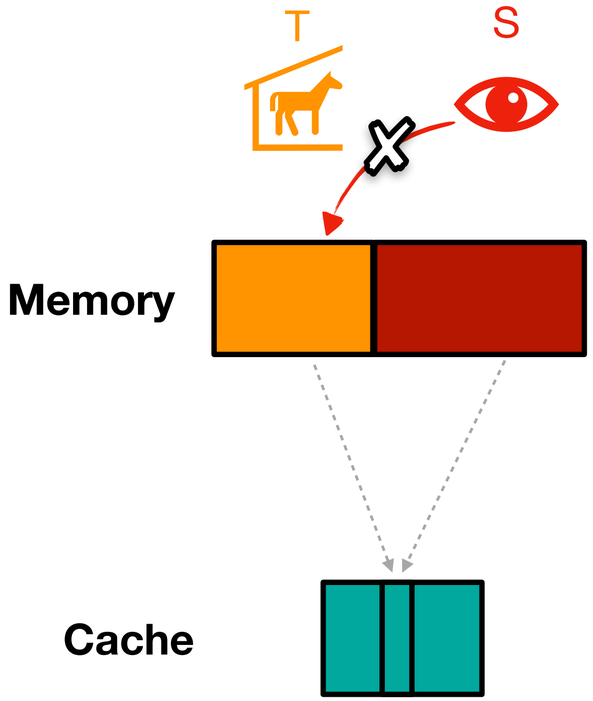
What is Time Protection?

- OSes typically implement *memory protection.*



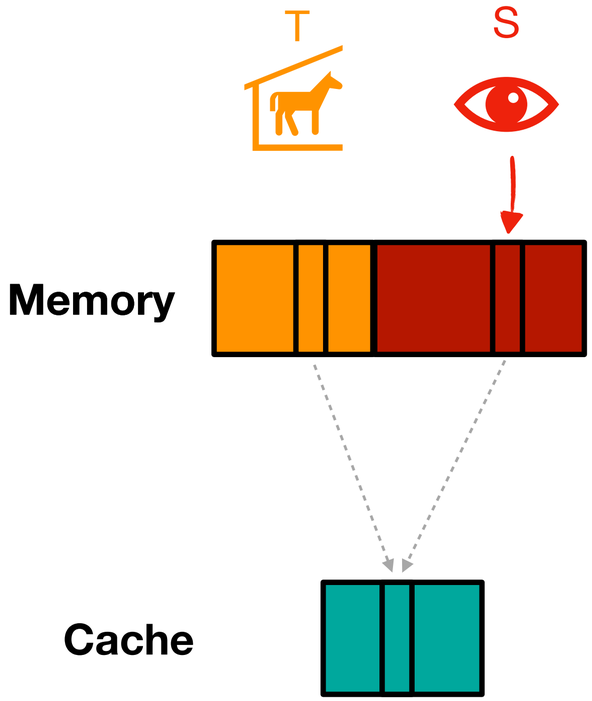
What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.



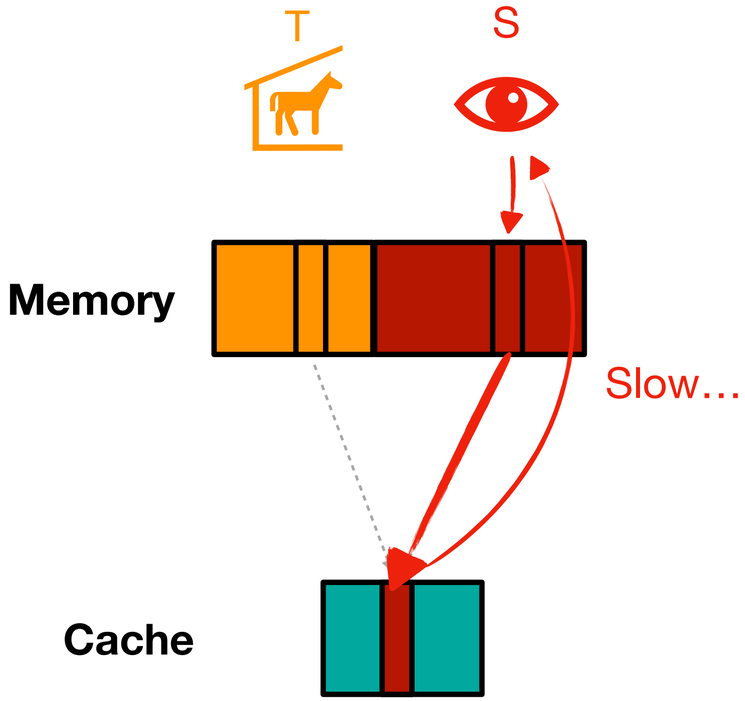
What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.



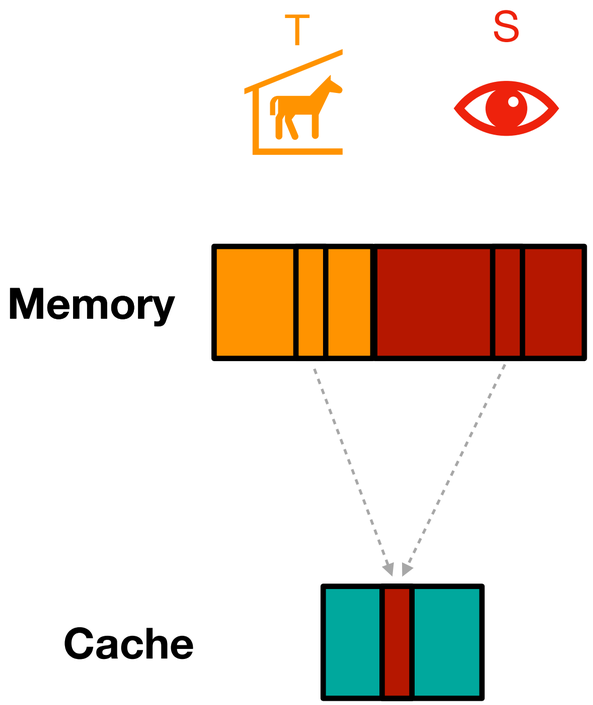
What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.



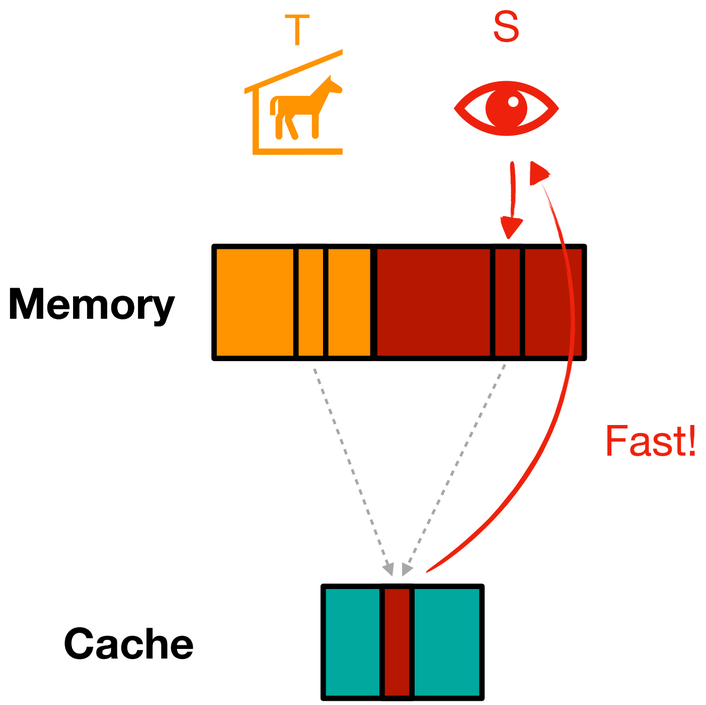
What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.



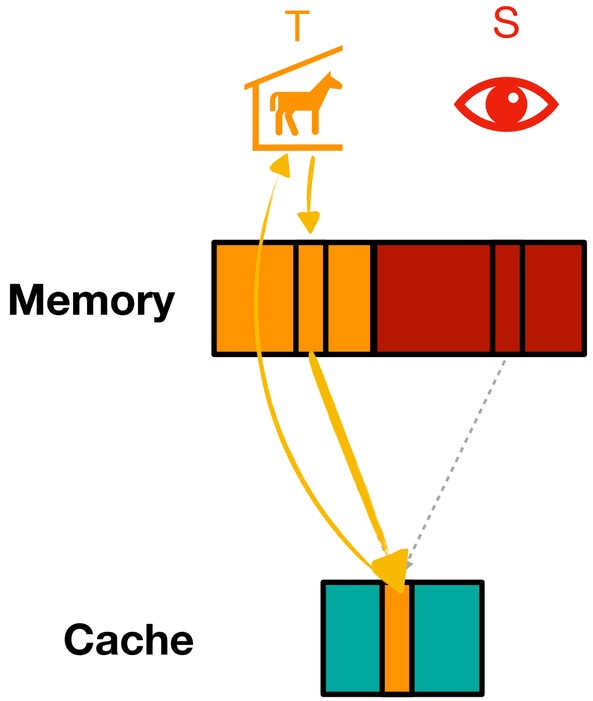
What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.



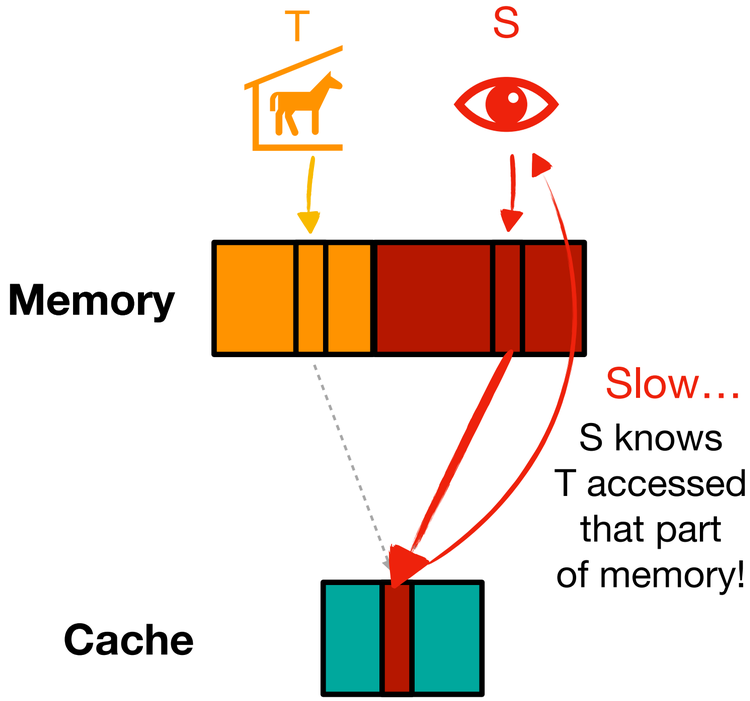
What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.



What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.

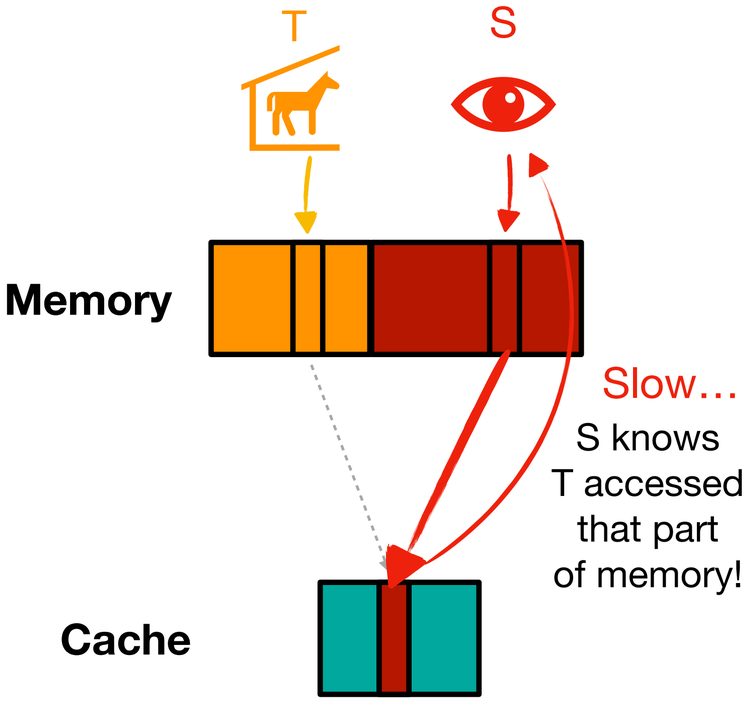


What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.

To prevent these *timing channels*,

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]



Qian Ge

Gernot Heiser

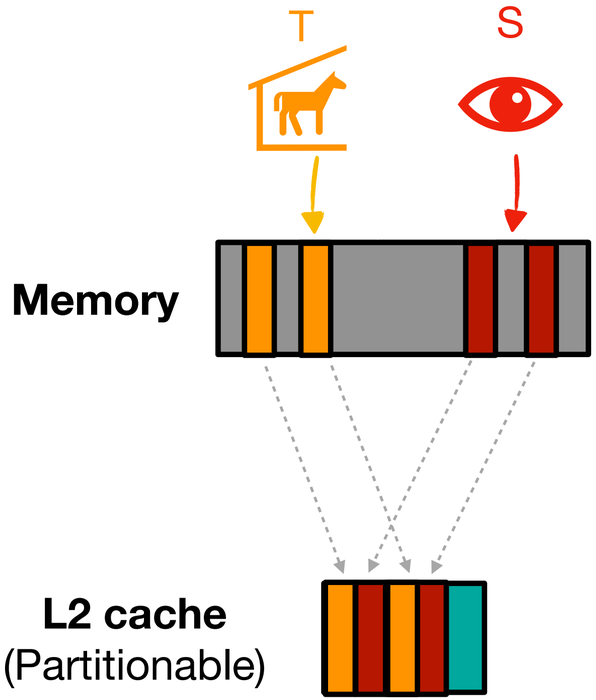
with
Yuval Yarom (U. Adelaide)
Tom Chothia (U. Birmingham)

What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.

To prevent these *timing channels*,

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches



Qian Ge

Gernot Heiser

with
Yuval Yarom (U. Adelaide)
Tom Chothia (U. Birmingham)

What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.

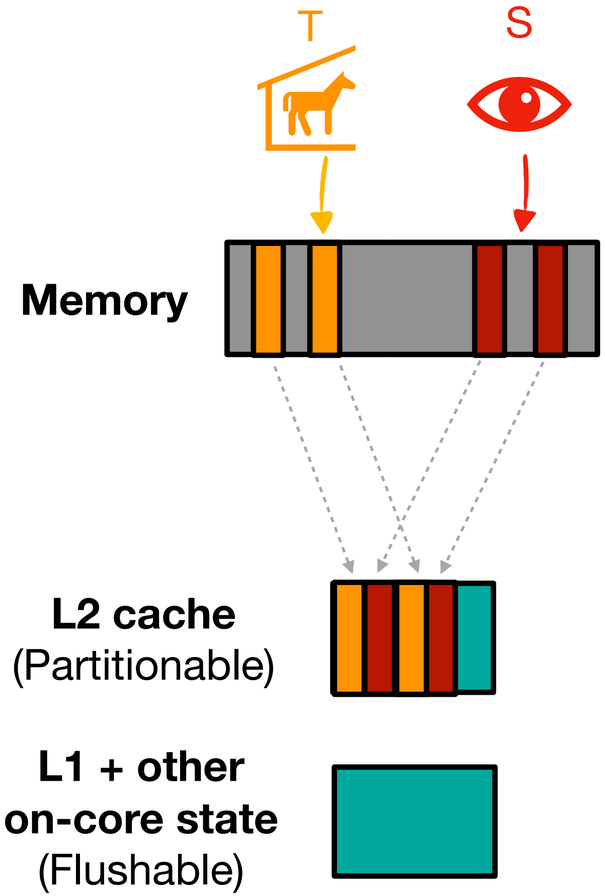
To prevent these *timing channels*,

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch

Qian Ge

Gernot Heiser

with
Yuval Yarom (U. Adelaide)
Tom Chothia (U. Birmingham)



What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.

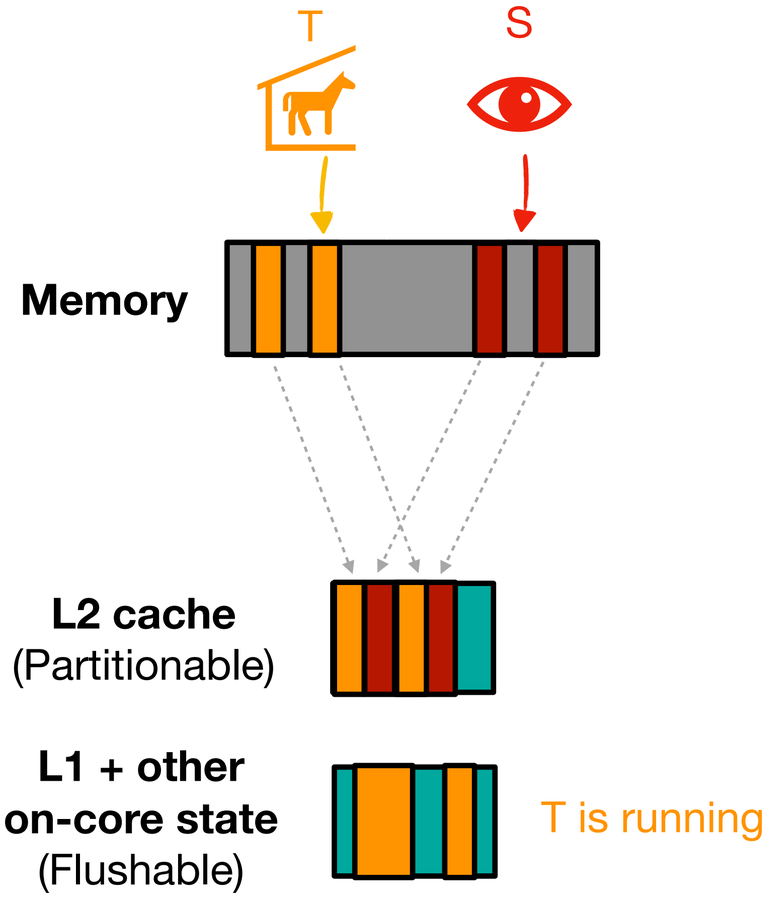
To prevent these *timing channels*,

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch

Qian Ge

Gernot Heiser

with
Yuval Yarom (U. Adelaide)
Tom Chothia (U. Birmingham)



What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.

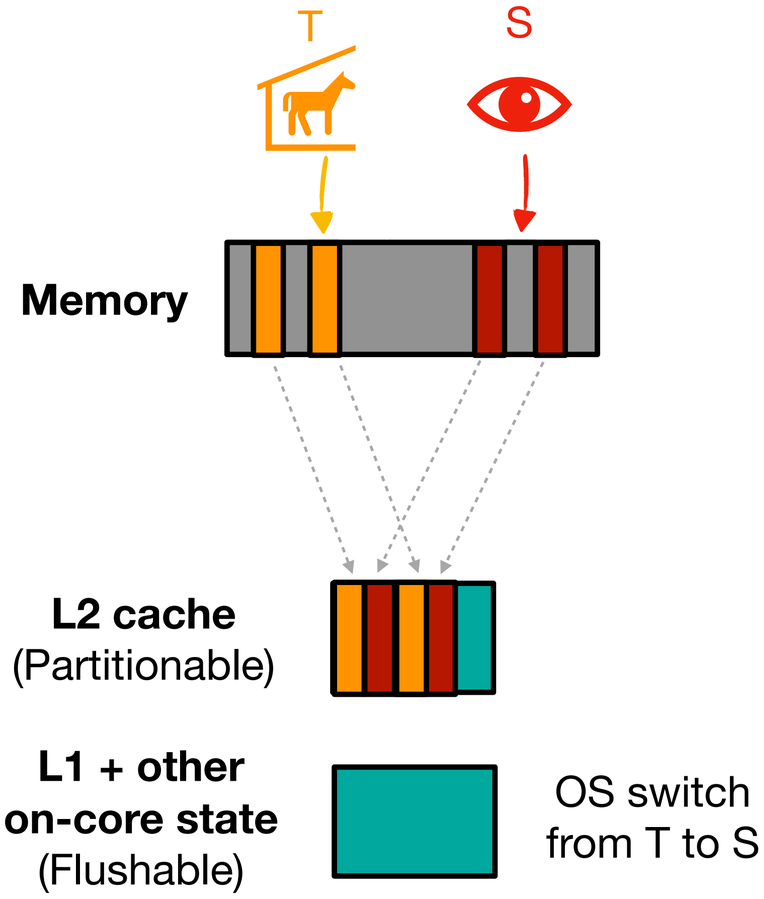
To prevent these *timing channels*,

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch

Qian Ge

Gernot Heiser

with
Yuval Yarom (U. Adelaide)
Tom Chothia (U. Birmingham)



"Flush": Write fixed content; "Pad": Wait up to fixed time.

What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.

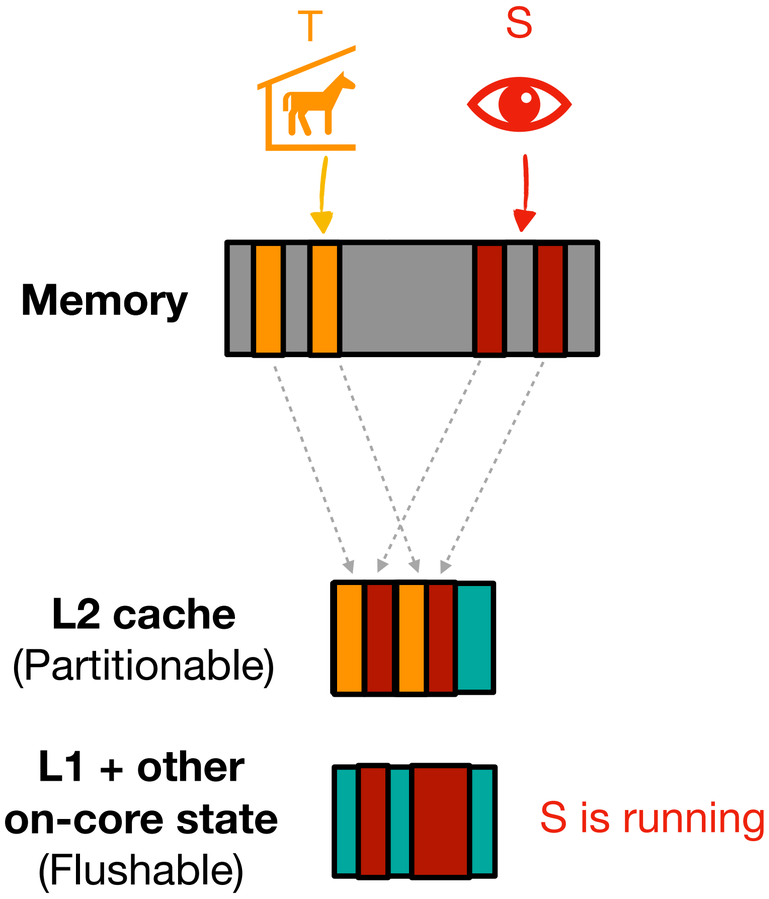
To prevent these *timing channels*,

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch

Qian Ge

Gernot Heiser

with
Yuval Yarom (U. Adelaide)
Tom Chothia (U. Birmingham)



"Flush": Write fixed content; "Pad": Wait up to fixed time.

What is Time Protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.

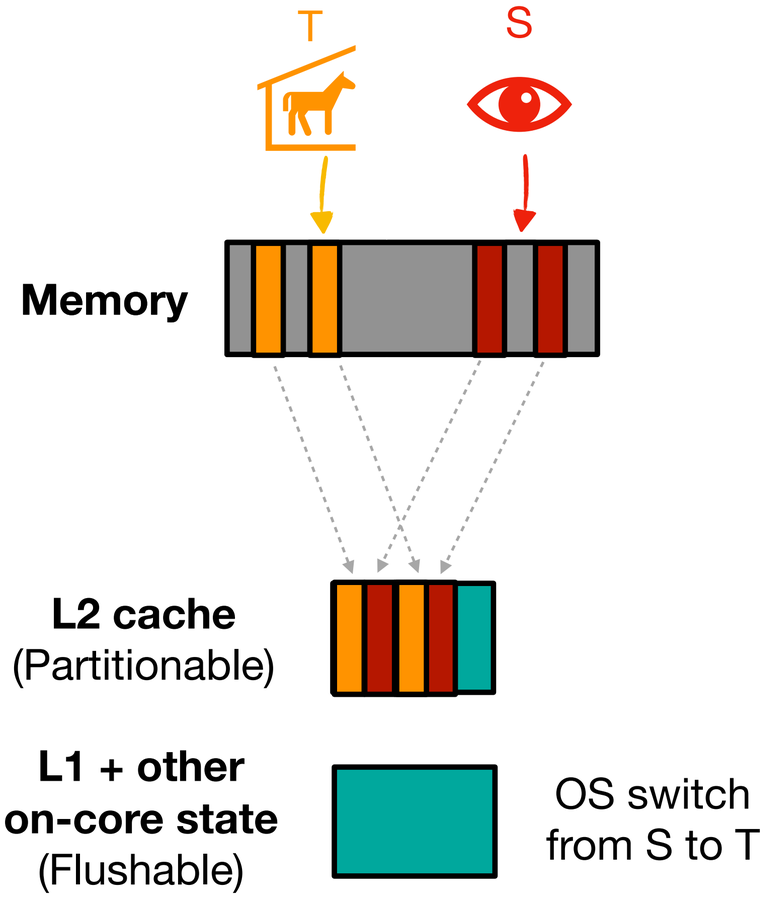
To prevent these *timing channels*,

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch

Qian Ge

Gernot Heiser

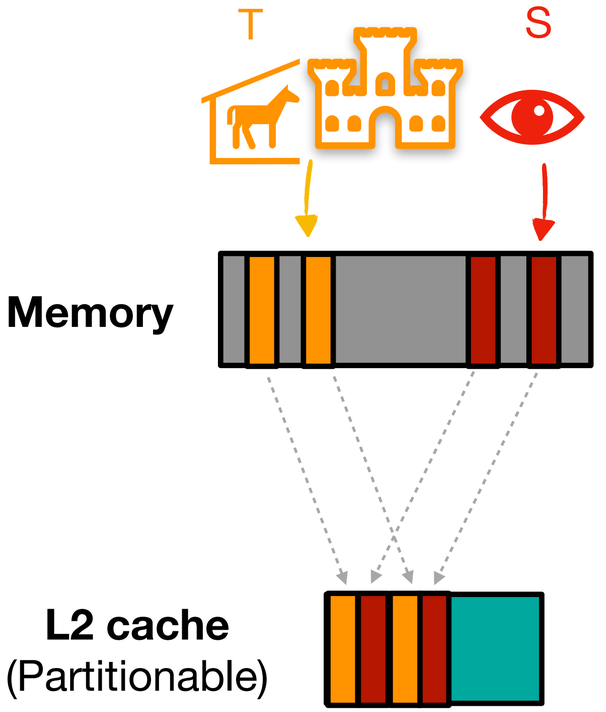
with
Yuval Yarom (U. Adelaide)
Tom Chothia (U. Birmingham)



"Flush": Write fixed content; "Pad": Wait up to fixed time.

What are time-protected communications?

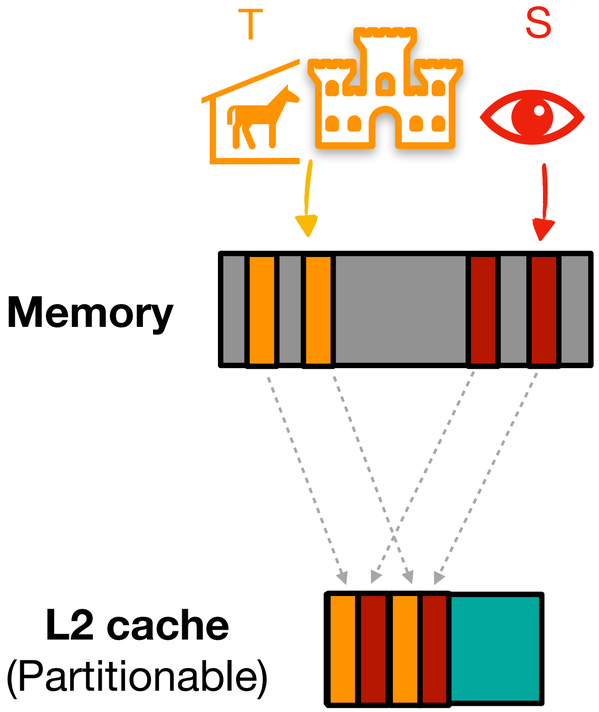
- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch



What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

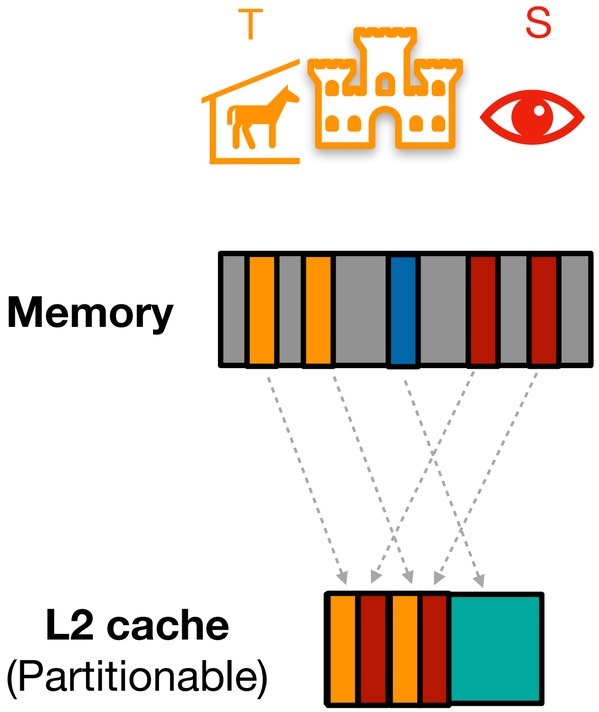
  Allowed: T <~ S, *Disallowed*: T ~/> S



What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

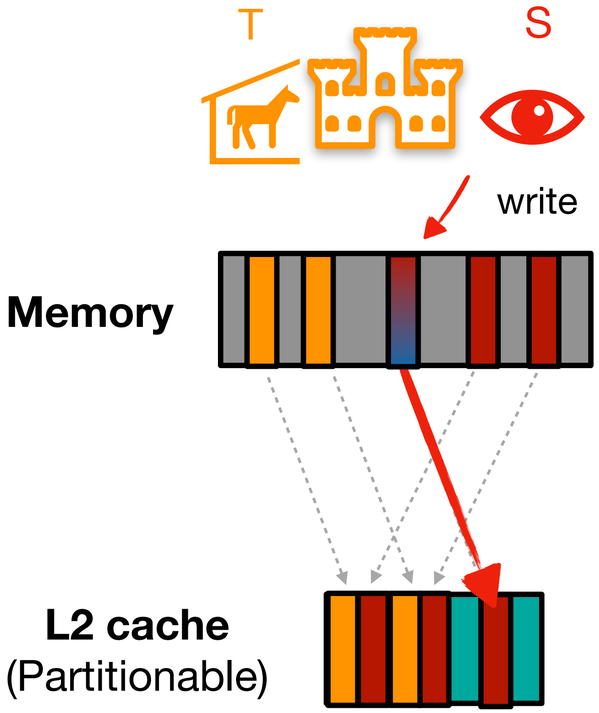
  Allowed: T <~ S, *Disallowed*: T ~/> S



What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

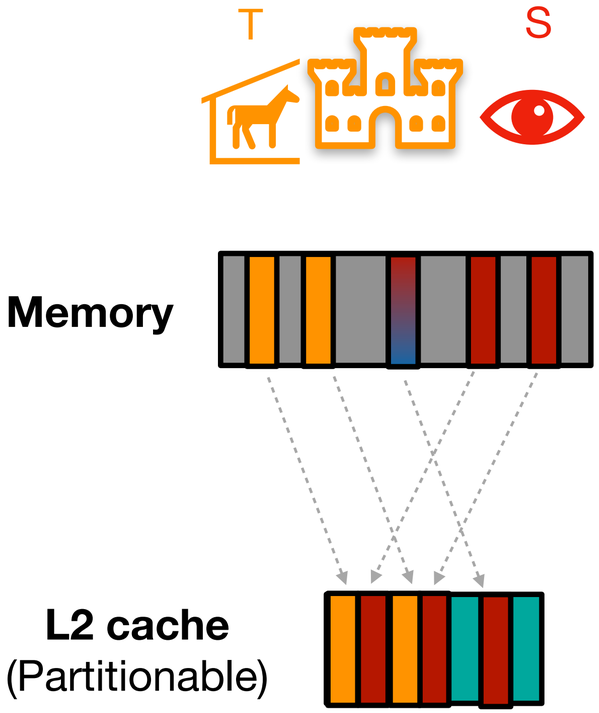
  Allowed: T <~ S, *Disallowed*: T ~/> S



What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

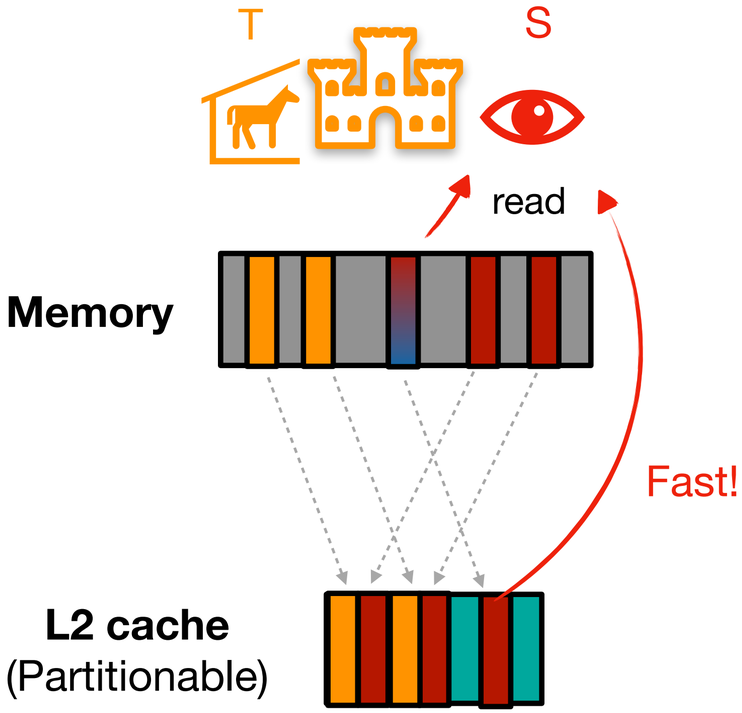
  Allowed: T <~ S, *Disallowed*: T ~/> S



What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

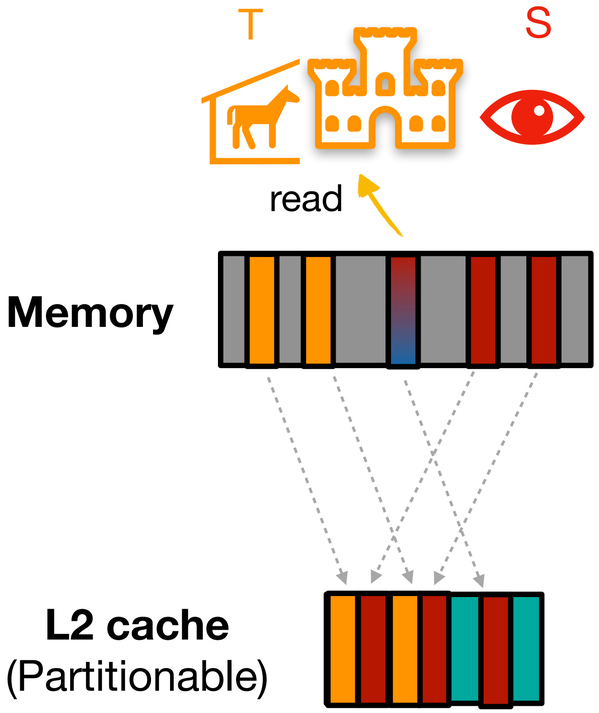
  Allowed: T <~ S, *Disallowed*: T ~/> S



What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

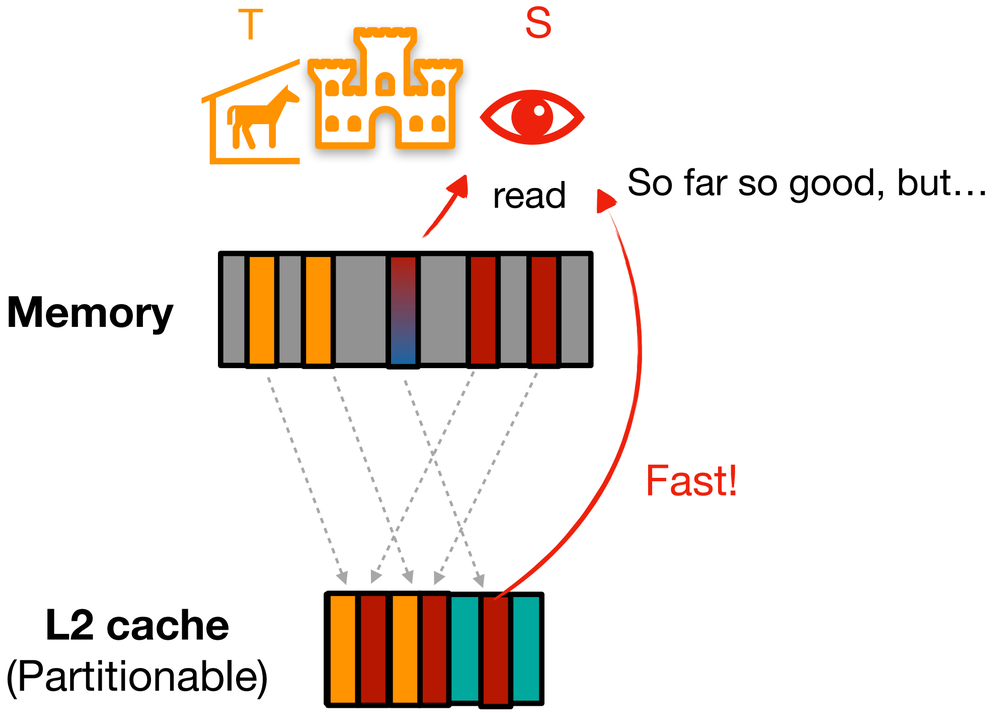
  Allowed: T <~ S, *Disallowed*: T ~/> S



What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

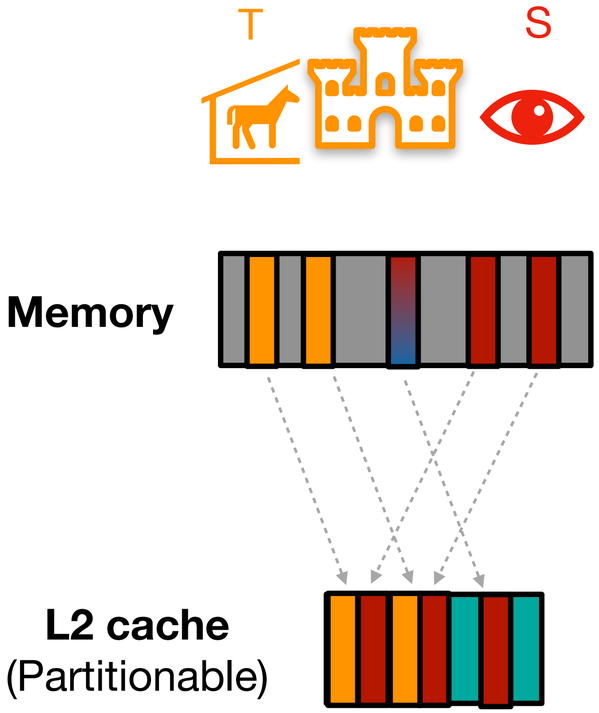


What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

S sees a timing difference on:

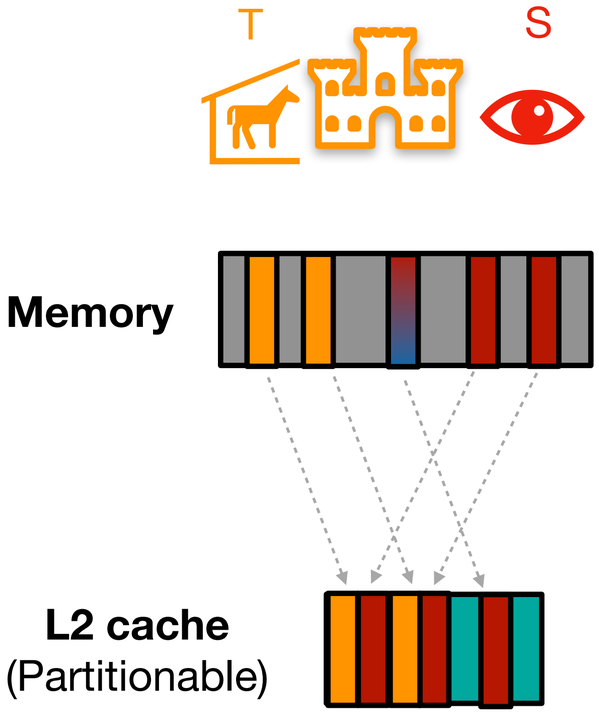


What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

  S sees a timing difference on:
  - Writes by T

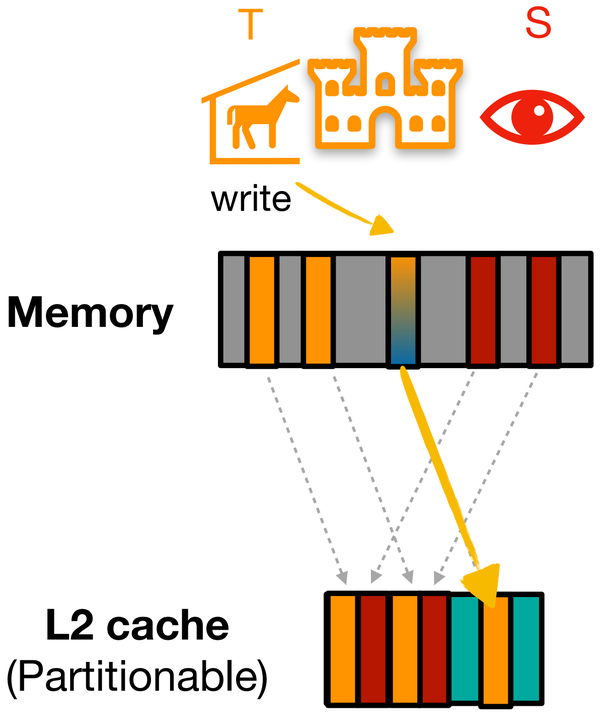


What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

  S sees a timing difference on:
  - Writes by T

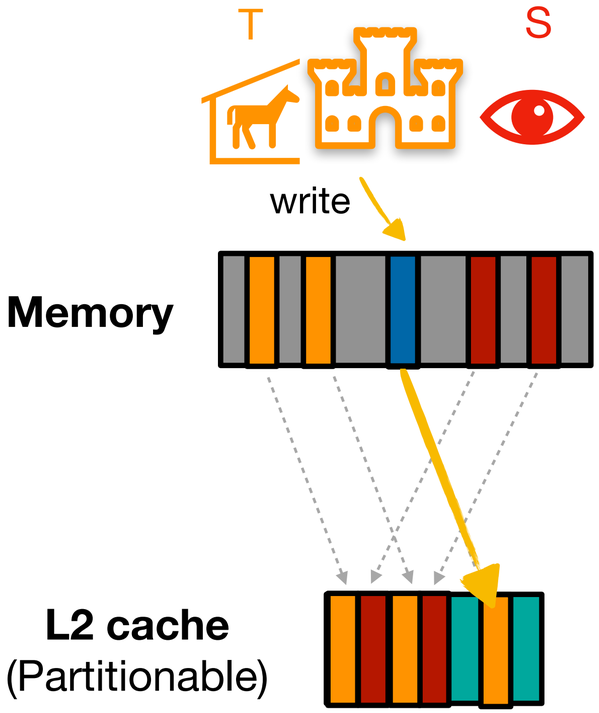


What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

  S sees a timing difference on:
  - Writes by T (even to wipe the data!)

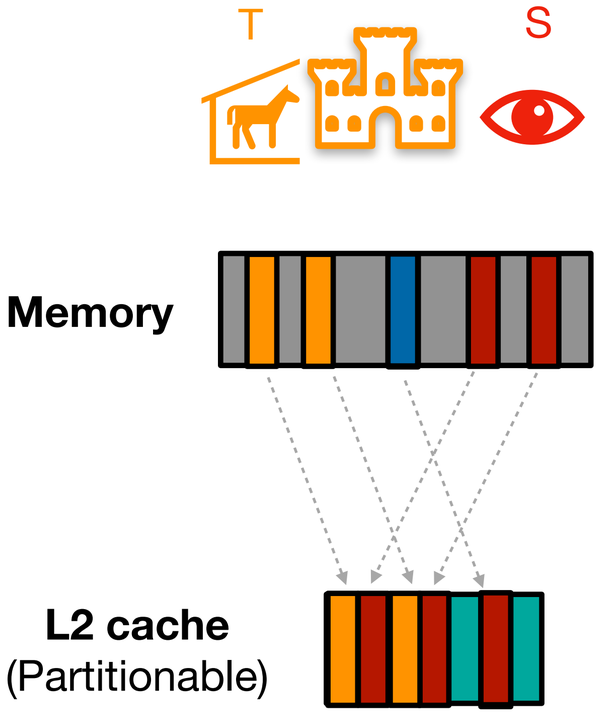


What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

  S sees a timing difference on:
  - Writes by T (even to wipe the data!)

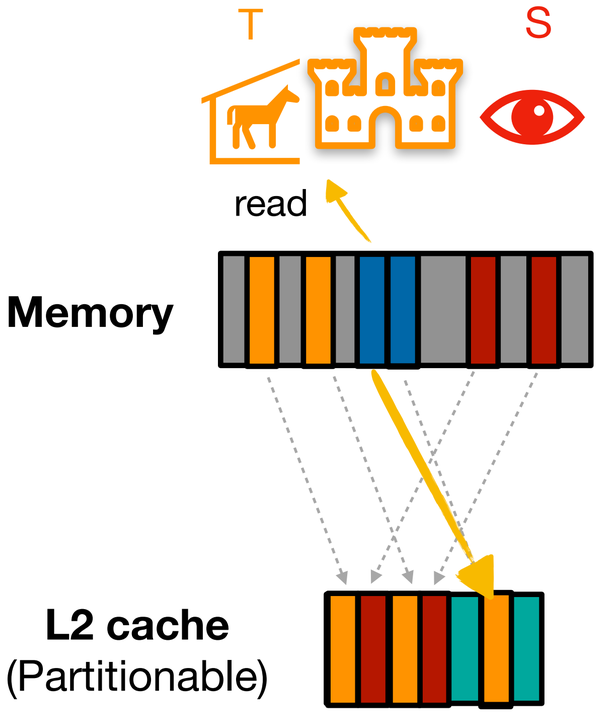


What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

  S sees a timing difference on:
  - Writes by T (even to wipe the data!)
  - Reads by T that evict lines (many per page!)

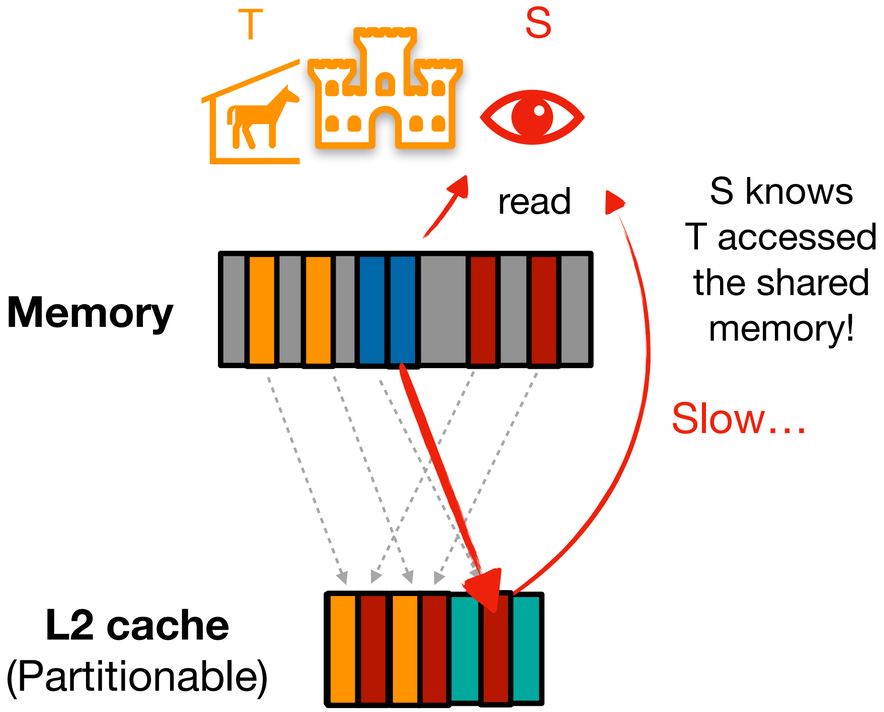


What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

  S sees a timing difference on:
  - Writes by T (even to wipe the data!)
  - Reads by T that evict lines (many per page!)



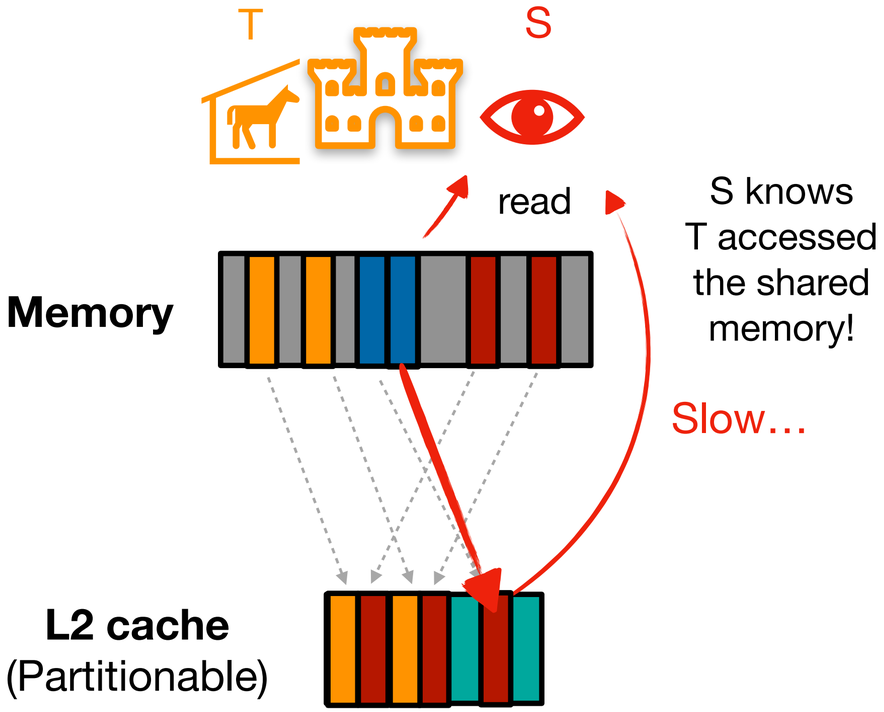
What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

  S sees a timing difference on:
  - Writes by T (even to wipe the data!)
  - Reads by T that evict lines (many per page!)

Takeaway: We need *targeted flush* for partitionable L2 caches to enforce T ~/> S while allowing T <~ S



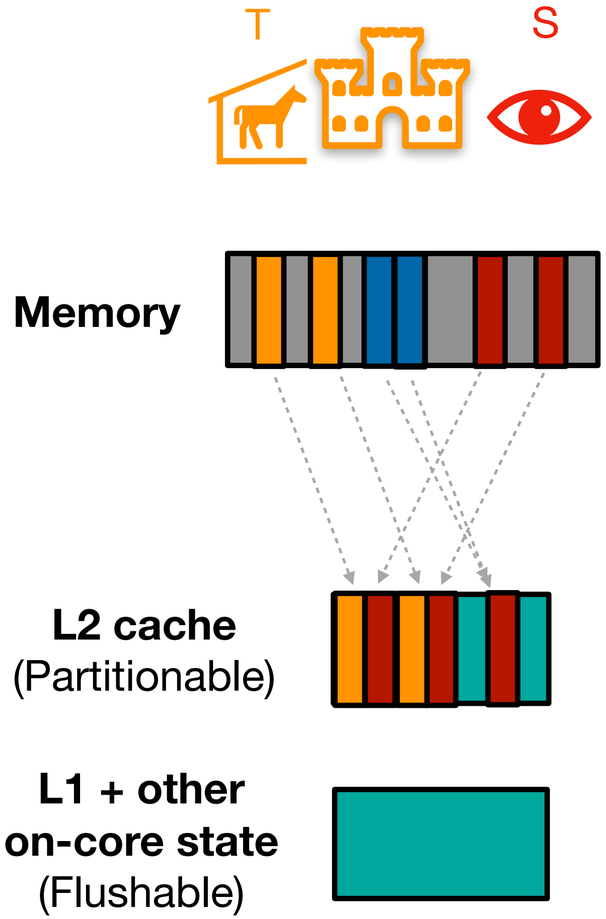
What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

  S sees a timing difference on:
  - Writes by T (even to wipe the data!)
  - Reads by T that evict lines (many per page!)

Takeaway: We need *targeted flush* for partitionable L2 caches to enforce T ~/> S while allowing T <~ S



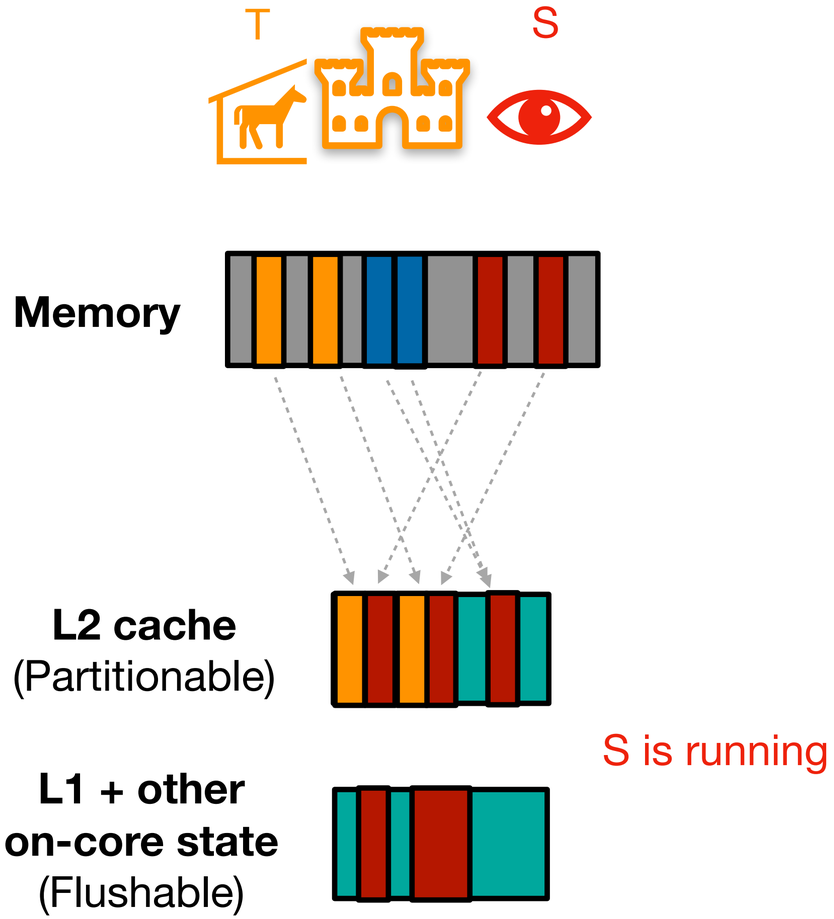
What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

  S sees a timing difference on:
  - Writes by T (even to wipe the data!)
  - Reads by T that evict lines (many per page!)

Takeaway: We need *targeted flush* for partitionable L2 caches to enforce T ~/> S while allowing T <~ S



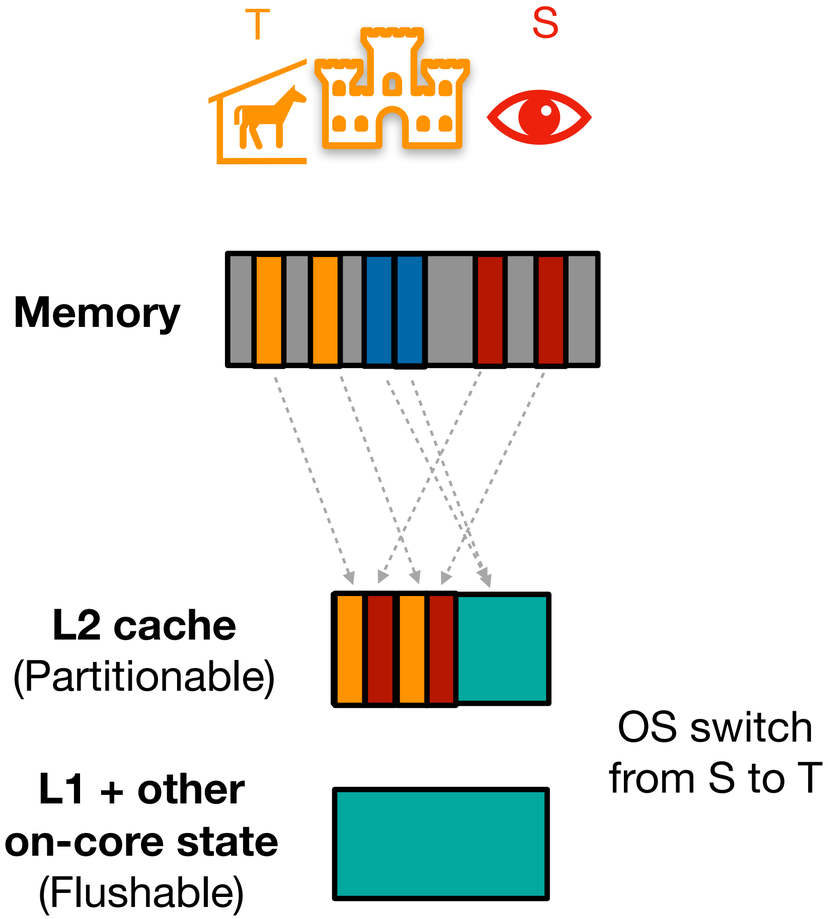
What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

  S sees a timing difference on:
  - Writes by T (even to wipe the data!)
  - Reads by T that evict lines (many per page!)

Takeaway: We need *targeted flush* for partitionable L2 caches to enforce T ~/> S while allowing T <~ S



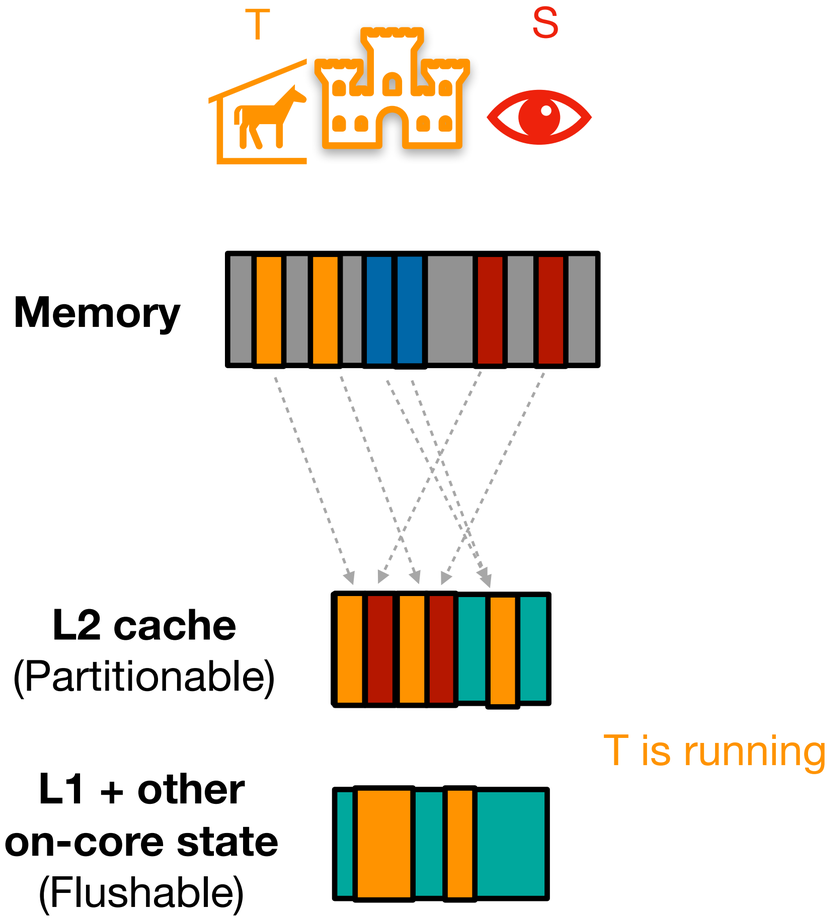
What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

  S sees a timing difference on:
  - Writes by T (even to wipe the data!)
  - Reads by T that evict lines (many per page!)

Takeaway: We need *targeted flush* for partitionable L2 caches to enforce T ~/> S while allowing T <~ S



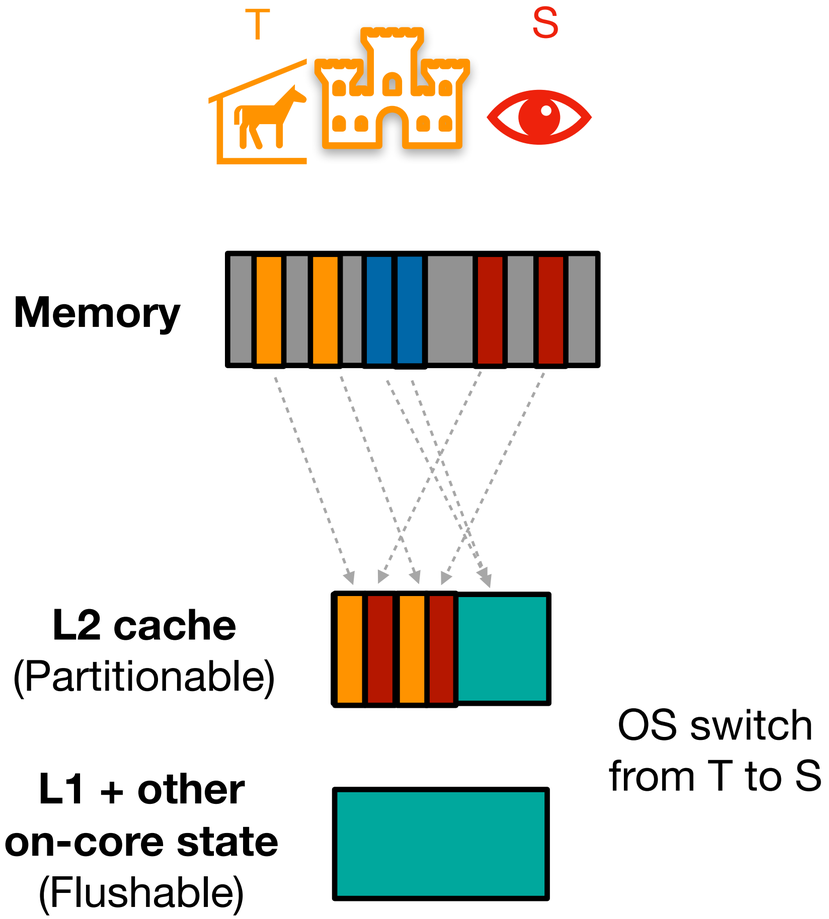
What are time-protected communications?

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- Let's now consider a *data diode* policy via a page of **shared memory**:

  Allowed: T <~ S, *Disallowed*: T ~/> S

  S sees a timing difference on:
  - Writes by T (even to wipe the data!)
  - Reads by T that evict lines (many per page!)

Takeaway: We need *targeted flush* for partitionable L2 caches to enforce T ~/> S while allowing T <~ S



The seL4 kernel and Time Protection

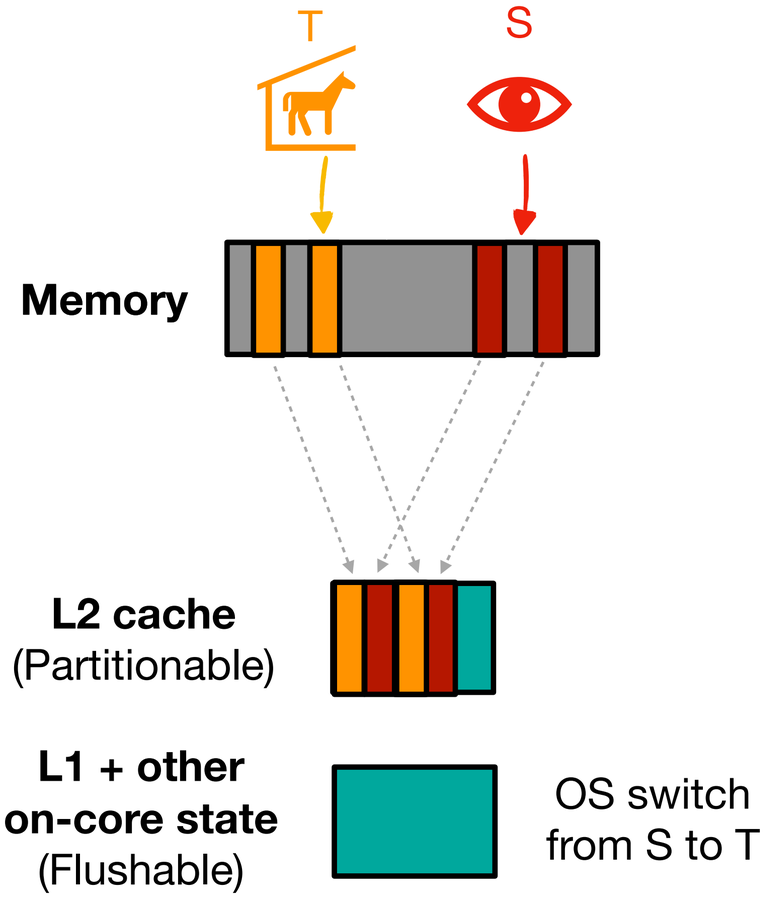
- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch

- For the seL4 OS kernel:

Qian Ge

Gernot Heiser

with
Yuval Yarom (U. Adelaide)
Tom Chothia (U. Birmingham)



"Flush": Write fixed content; "Pad": Wait up to fixed time.

The seL4 kernel and Time Protection

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch

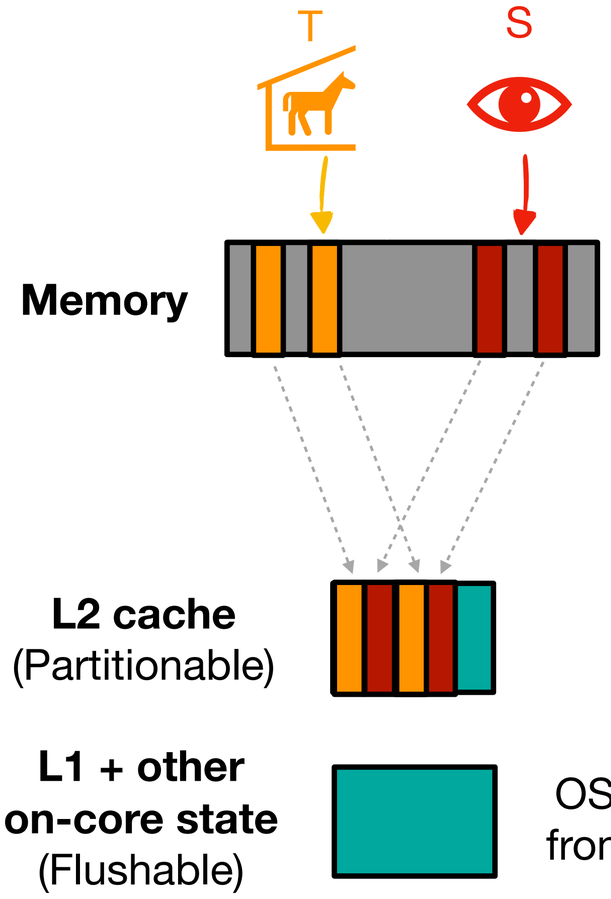
- For the seL4 OS kernel:

seL4: World's first OS kernel with correctness proof!

Qian Ge

Gernot Heiser

with
Yuval Yarom (U. Adelaide)
Tom Chothia (U. Birmingham)



"Flush": Write fixed content; "Pad": Wait up to fixed time.

The seL4 kernel and Time Protection

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch

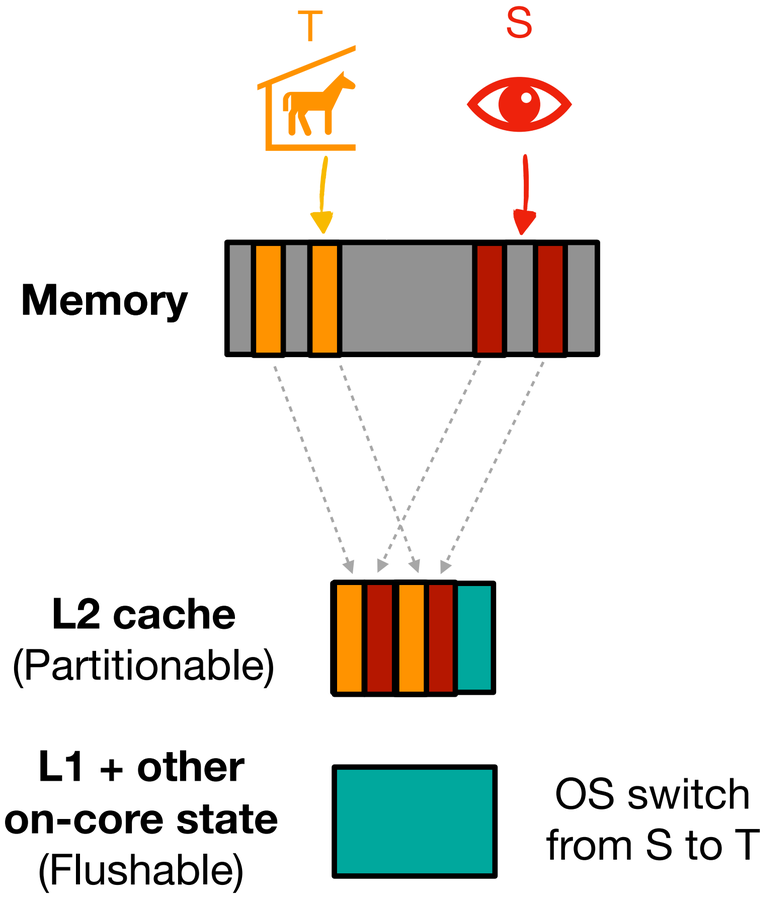
- For the seL4 OS kernel:
  - Implemented, evaluated empirically on ARM, x86
    See EuroSys: [Ge et al. 2019]

seL4: World's first OS kernel with correctness proof!

Qian Ge

Gernot Heiser

with
Yuval Yarom (U. Adelaide)
Tom Chothia (U. Birmingham)



"Flush": Write fixed content; "Pad": Wait up to fixed time.

The seL4 kernel and Time Protection

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch

- For the seL4 OS kernel:
  - Implemented, evaluated empirically on ARM, x86
    See EuroSys: [Ge et al. 2019]
  - Ported to RISC-V with hardware support
    See arXiv preprint: [Buckley, Sison et al. 2023]
    (HW support by [Wistoff et al. 2023])

seL4: World's first OS kernel with correctness proof!


Scott Buckley


Rob Sison


Nils Wistoff


Curtis Millar


Toby Murray


Gerwin Klein


Gernot Heiser

Thanks to funding from
Australian Research Council

"Flush": Write fixed content; "Pad": Wait up to fixed time.

The seL4 kernel and Time Protection

- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch

- For the seL4 OS kernel:
  - Implemented, evaluated empirically on ARM, x86
    See EuroSys: [Ge et al. 2019]
  - Ported to RISC-V with hardware support
    See arXiv preprint: [Buckley, Sison et al. 2023]
    (HW support by [Wistoff et al. 2023])

seL4: World's first OS kernel with correctness proof!

- **This talk (pt I):** How do we verify it works?


Scott Buckley


Rob Sison


Nils Wistoff


Curtis Millar


Toby Murray

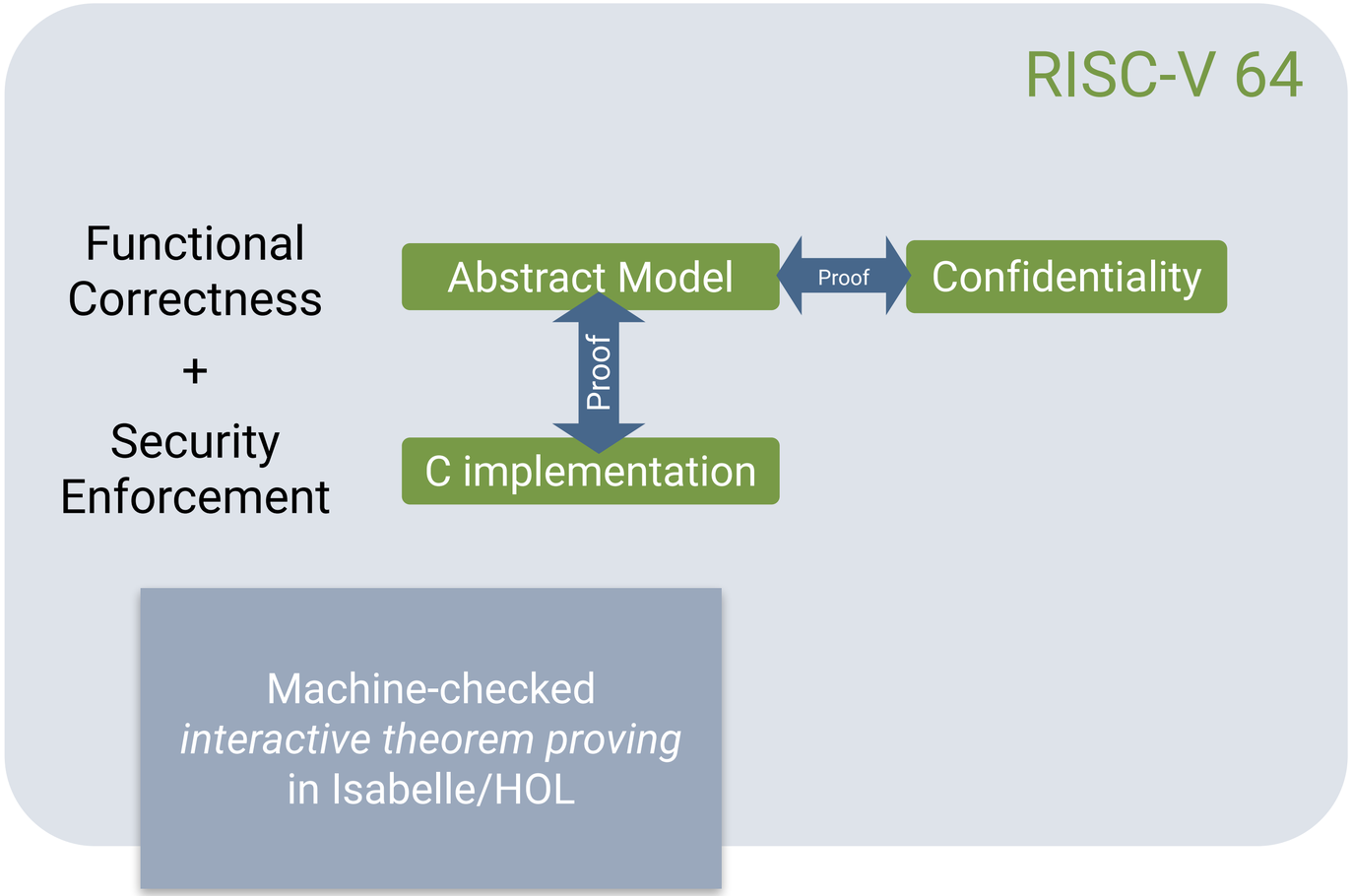

Gerwin Klein


Gernot Heiser

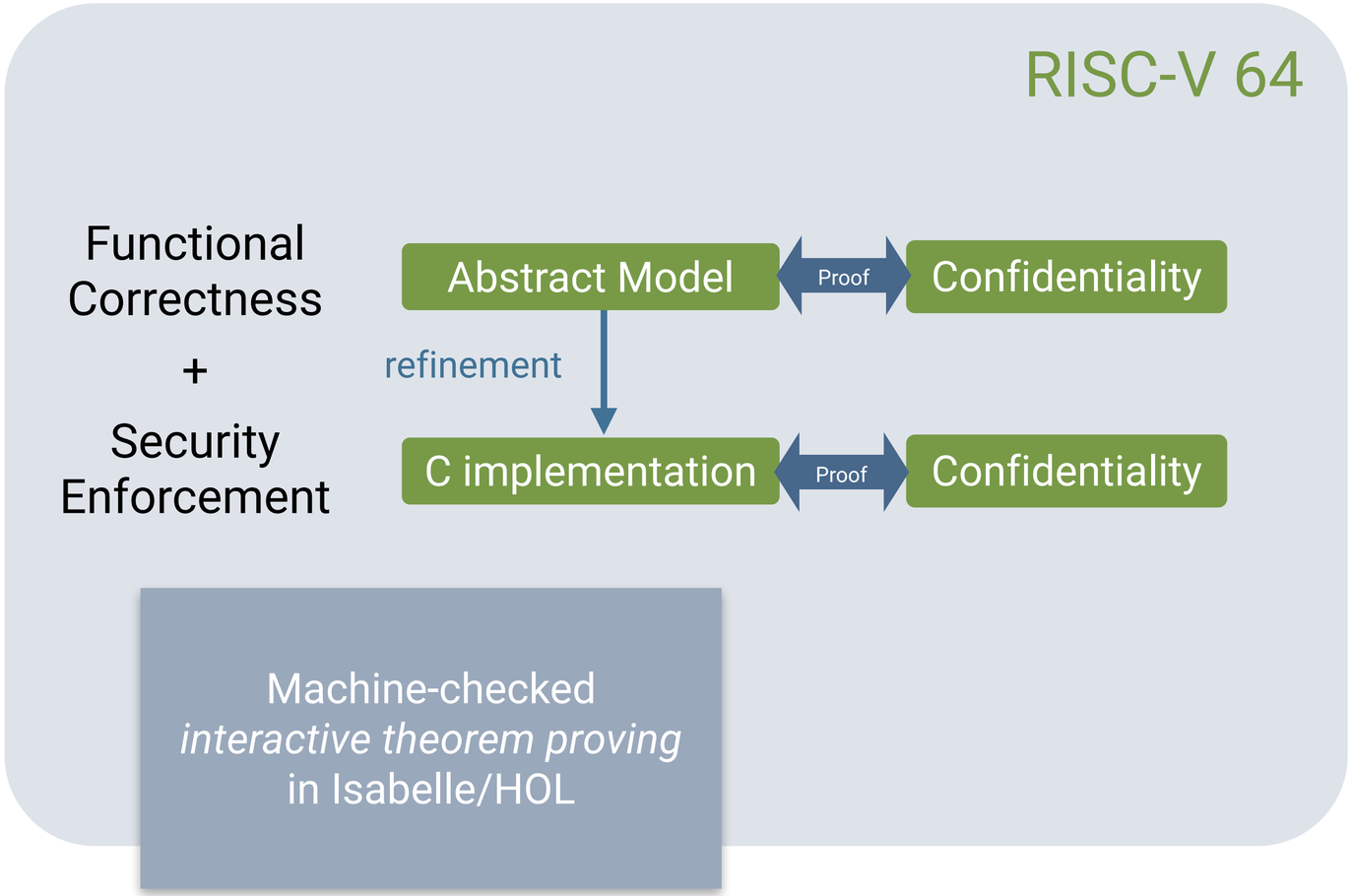
Thanks to funding from
Australian Research Council

"Flush": Write fixed content; "Pad": Wait up to fixed time.

Time protection verification for seL4



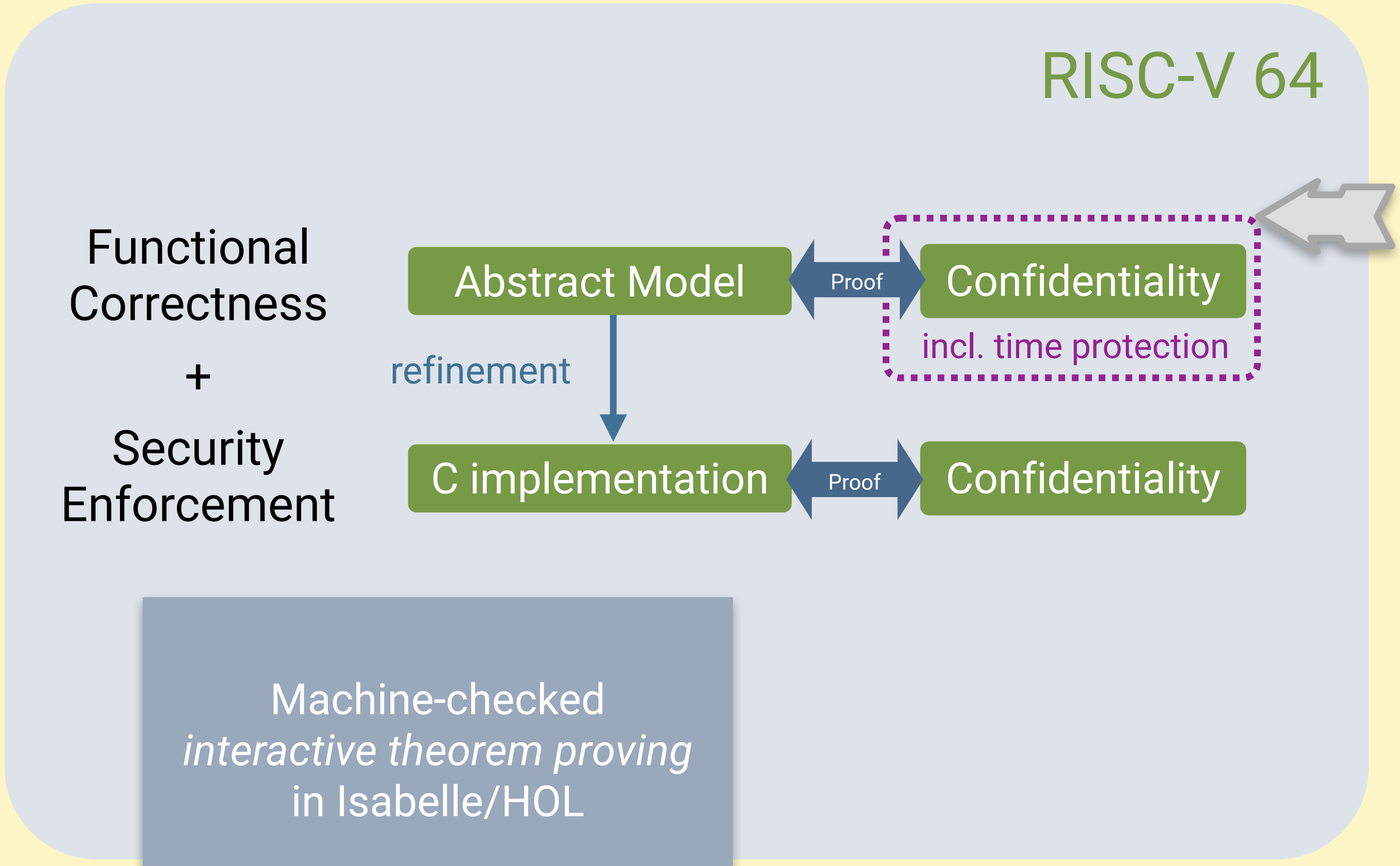
Time protection verification for seL4



Time protection verification for seL4

The plan 2 years ago…

Thanks to funding from
Australian Research Council

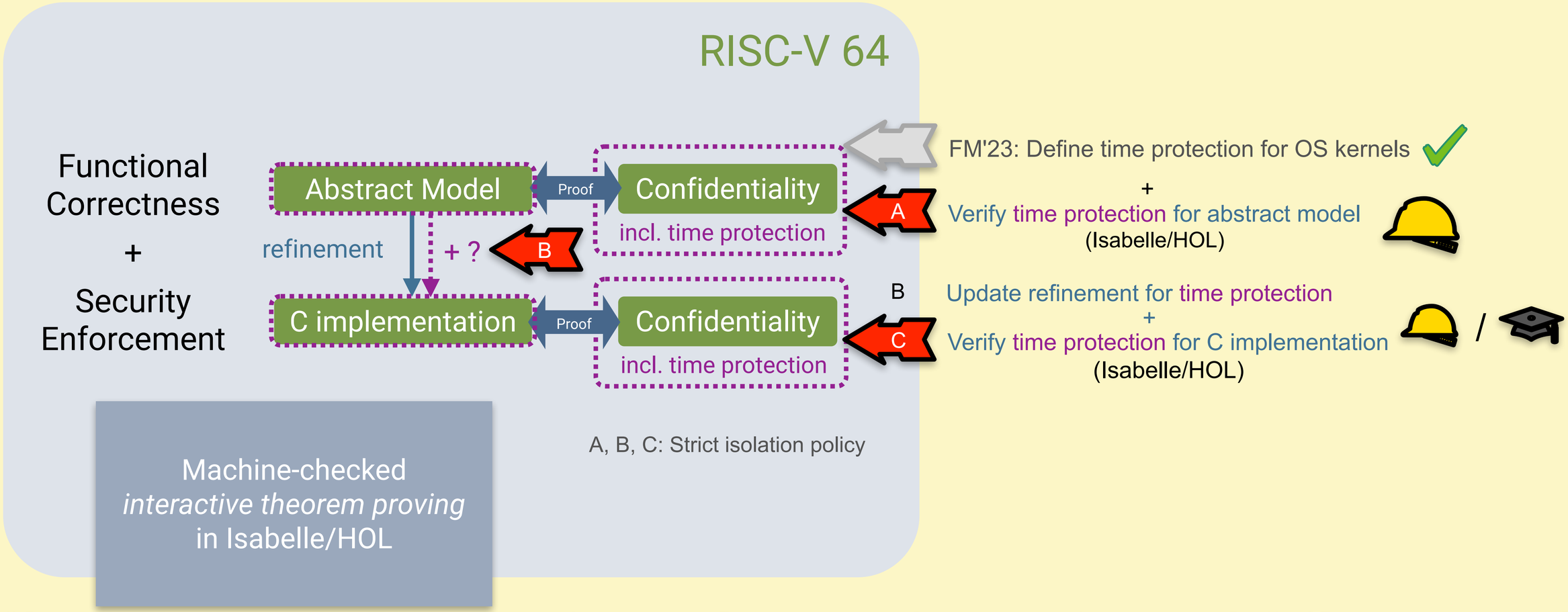


FM'23: Define time protection for OS kernels ✓

Time protection verification for seL4

The plan 2 years ago…

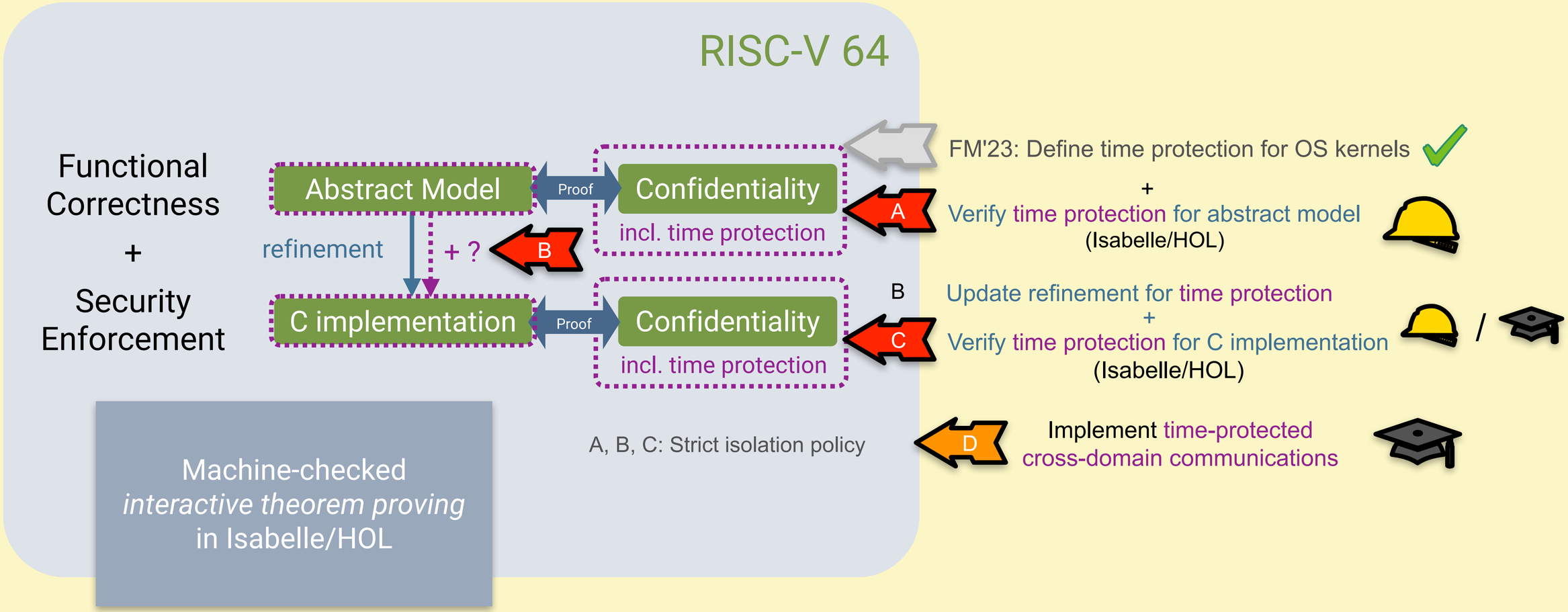
Thanks to funding from
Australian Research Council



Time protection verification for seL4

The plan 2 years ago…

Thanks to funding from
Australian Research Council

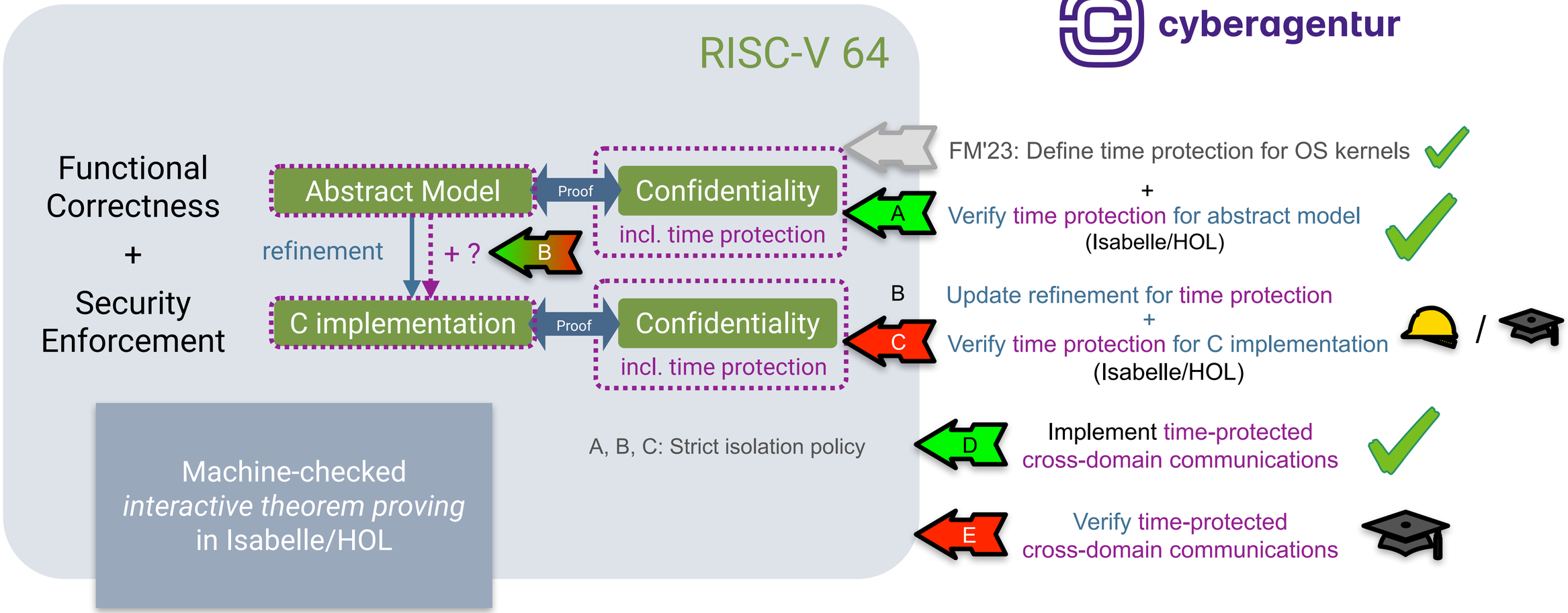


Time protection verification for seL4

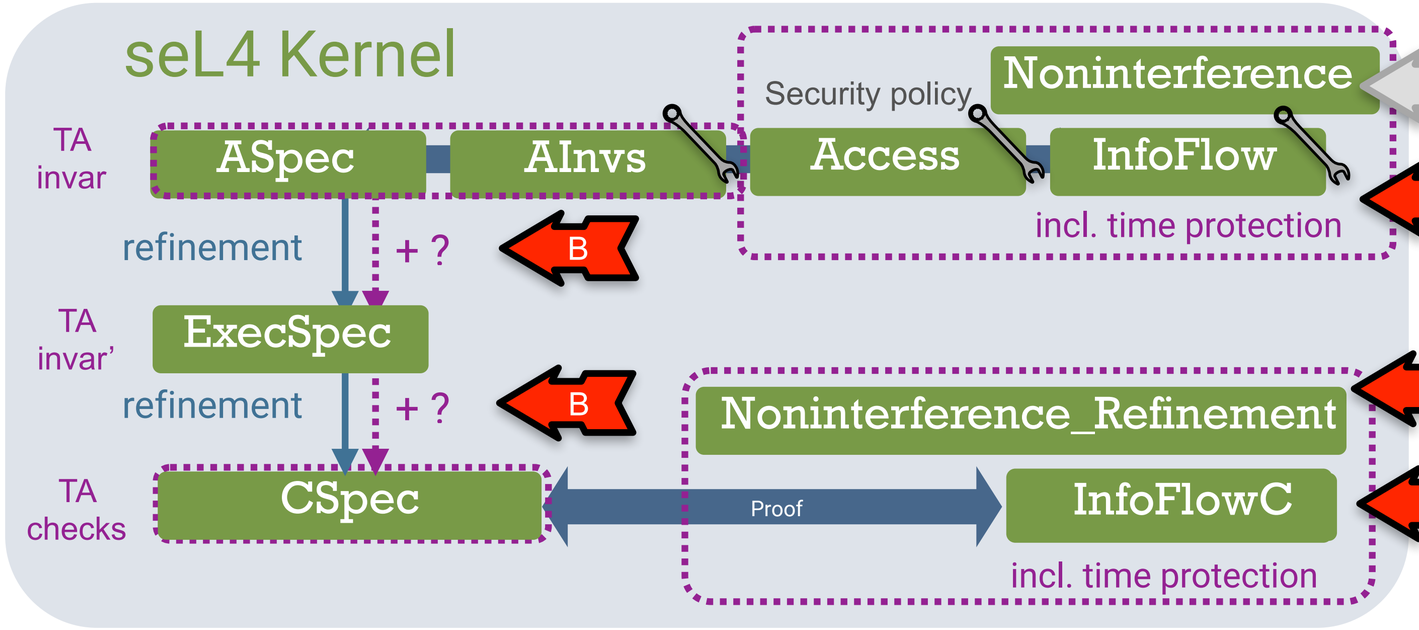
The status today:

Thanks to funding from

cyberagentur



Time protection verification for seL4



FM'23: Define time protection for OS kernels ✓

+

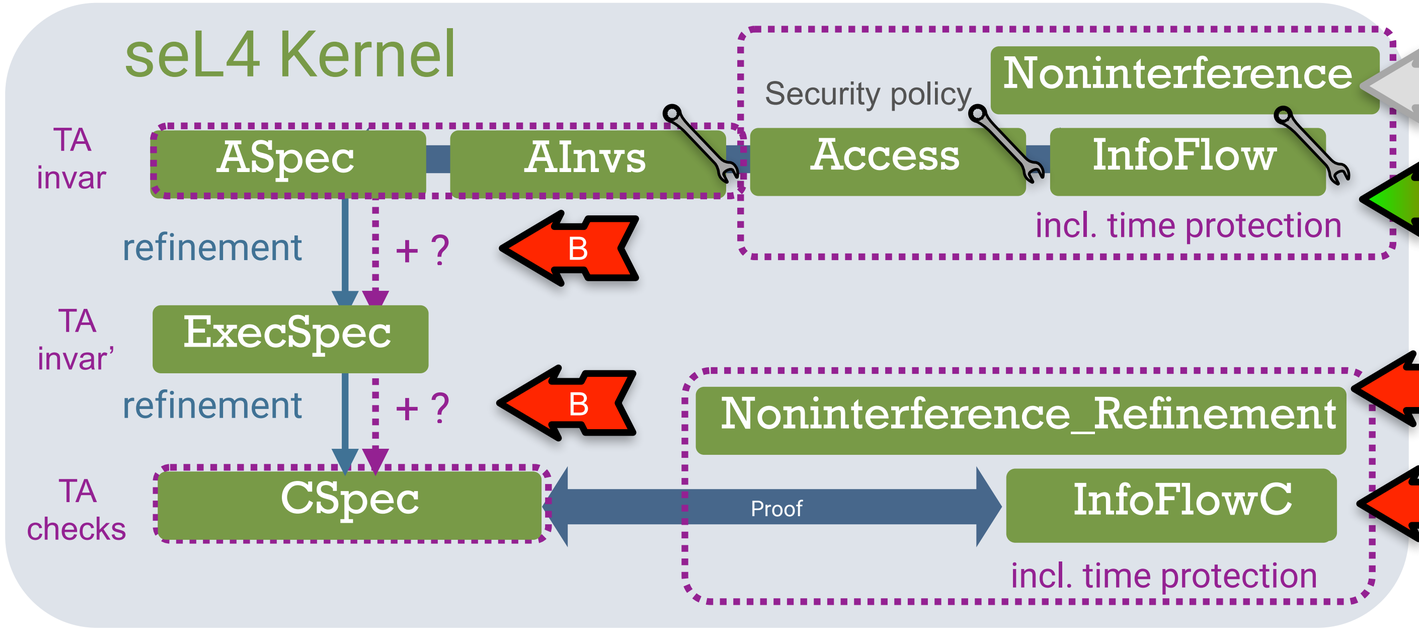
A — Verify time protection for abstract model (Isabelle/HOL)

B — Update refinement for time protection

+

C — Verify time protection for C implementation (Isabelle/HOL)

Time protection verification for seL4



FM'23: Define time protection for OS kernels ✓

+

A — Verify time protection for abstract model (Isabelle/HOL)

B — Update refinement for time protection

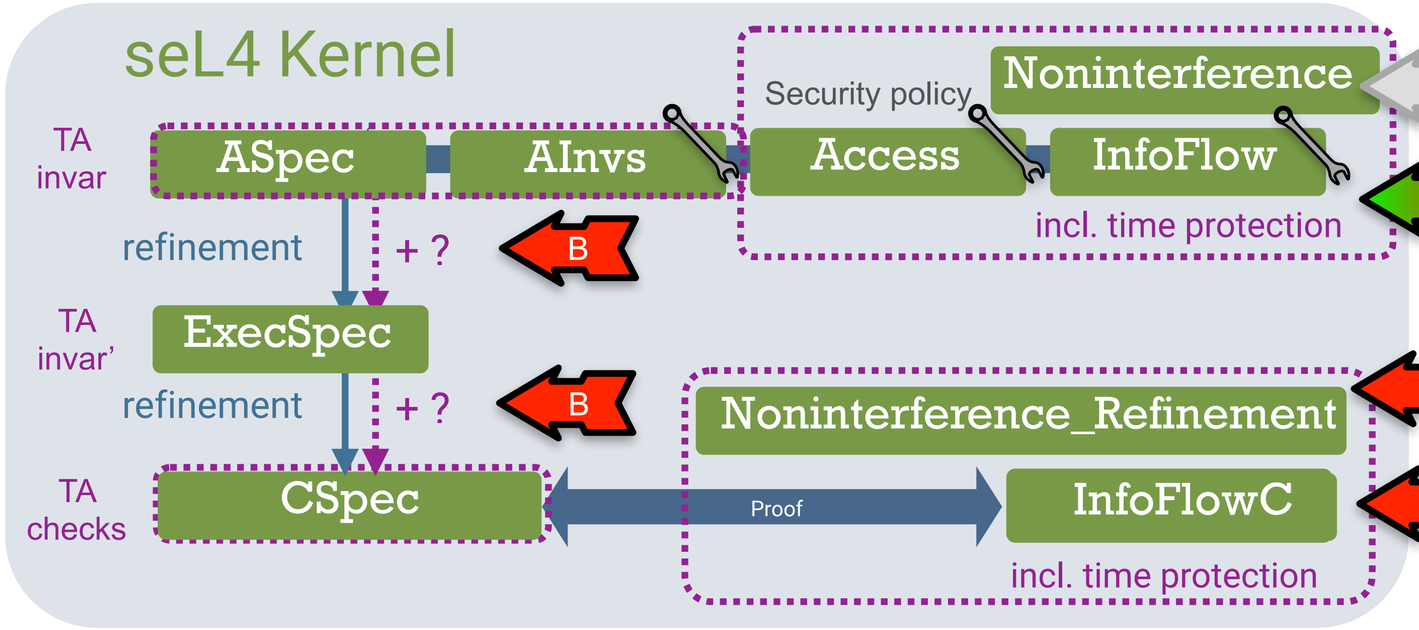
+

C — Verify time protection for C implementation (Isabelle/HOL)

- (A) Key **ASpec** property: "TA invariant"

Sai Nair
BSc(Hons)
thesis

Time protection verification for seL4



FM'23: Define time protection for OS kernels ✓

+

A — Verify time protection for abstract model (Isabelle/HOL)

B — Update refinement for time protection

+

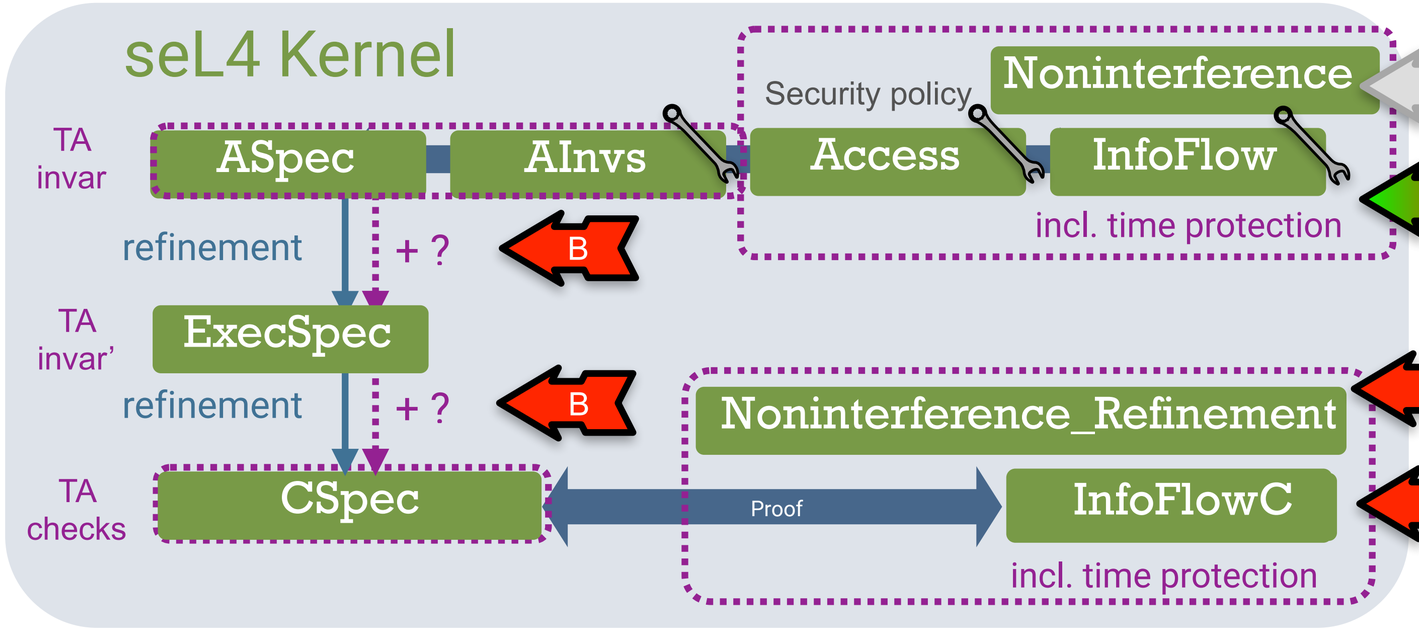
C — Verify time protection for C implementation (Isabelle/HOL)

"obj refs don't point outside static *TA set"*

- (A) Key **ASpec** property: "TA invariant"

Sai Nair
BSc(Hons)
thesis

Time protection verification for seL4



FM'23: Define time protection for OS kernels ✓

+

A — Verify time protection for abstract model (Isabelle/HOL)

B — Update refinement for time protection

+

C — Verify time protection for C implementation (Isabelle/HOL)

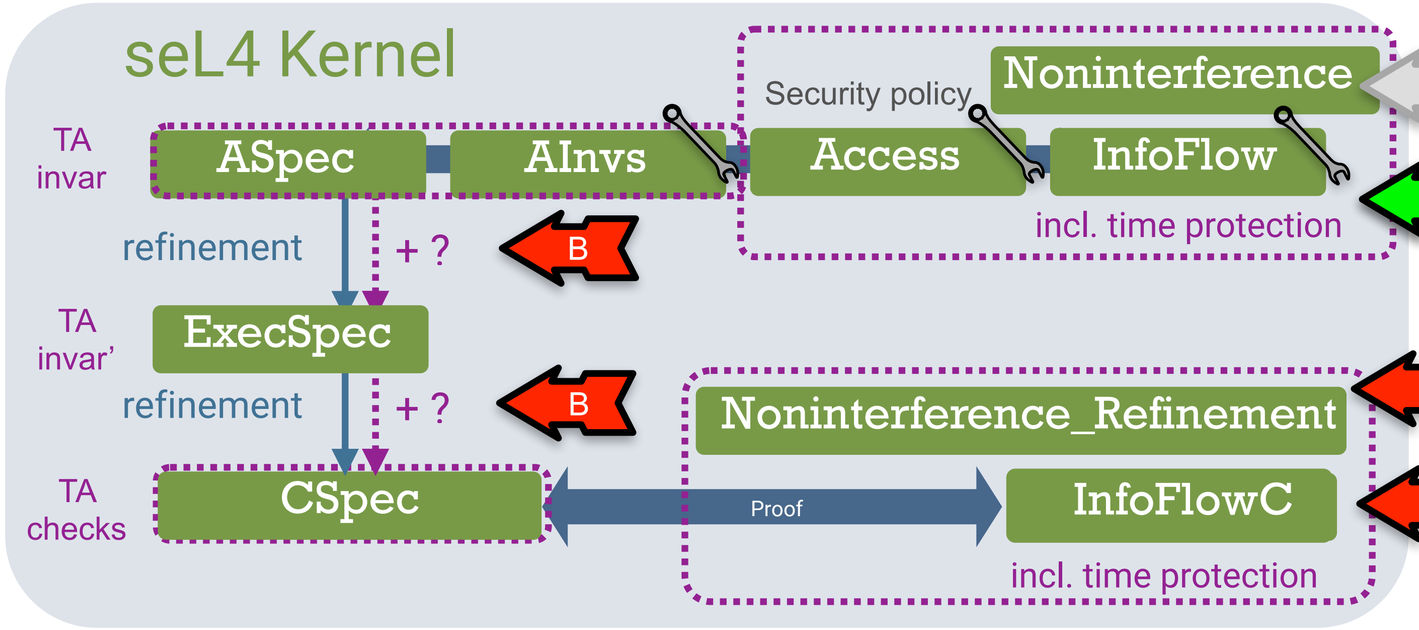
"obj refs don't point outside static *TA set"*

- (A) Key **ASpec** property: "TA invariant"
- Proof across **ASpec** in progress

Continuing on from

Sai Nair
BSc(Hons)
thesis

Time protection verification for seL4



FM'23: Define time protection for OS kernels ✓

+

A Verify time protection for abstract model (Isabelle/HOL) ✓

B Update refinement for time protection

+

C Verify time protection for C implementation (Isabelle/HOL)

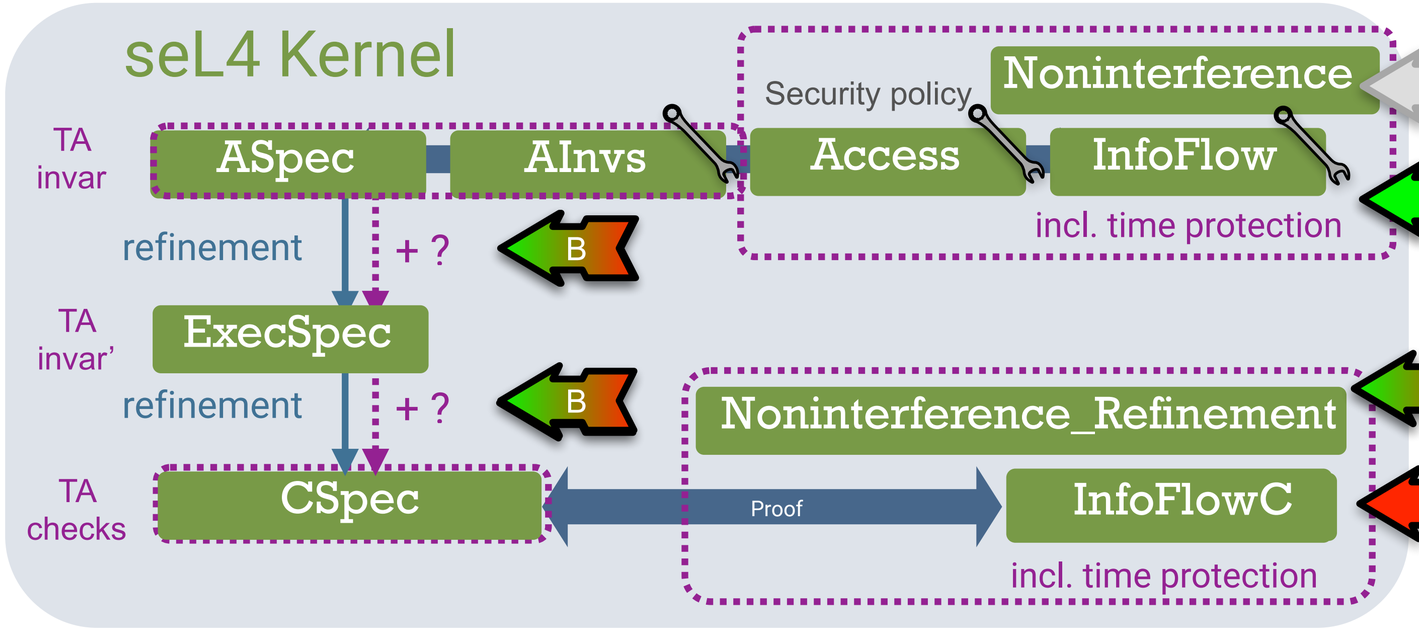
"obj refs don't point outside static *TA set"*

- (A) Key **ASpec** property: "TA invariant"
  - **ASpec** otherwise *integrated* into time protection theory (i.e. *instantiating Isabelle locales*) of [Buckley, Sison et al. 2023] ✓
- Proof across **ASpec** in progress

Continuing on from

Sai Nair
BSc(Hons)
thesis

Time protection verification for seL4



FM'23: Define time protection for OS kernels ✓

+

Verify time protection for abstract model (Isabelle/HOL) ✓

Update refinement for time protection

+

Verify time protection for C implementation (Isabelle/HOL)

"obj refs don't point outside static *TA set"*

- (A) Key **ASpec** property: "TA invariant"
  - **ASpec** otherwise *integrated* into time protection theory (i.e. *instantiating Isabelle locales*) of [Buckley, Sison et al. 2023] ✓
- (B + C) Refinement: TA invariant (via **ExecSpec**) ⟹ TA checks pass (**CSpec**)

- Proof across **ASpec** in progress

Continuing on from

Sai Nair
BSc(Hons)
thesis

Continuing on from

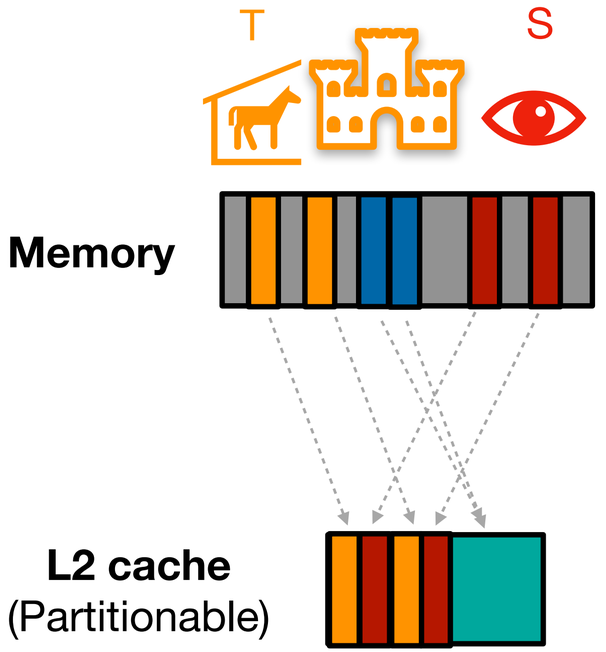
Thomas Liang
BSc (Hons)
Thesis

Time-protected communications in seL4

seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications



We need *targeted flush* for partitionable L2 caches to enforce T ~/> S while allowing T <~ S

Time-protected communications in seL4

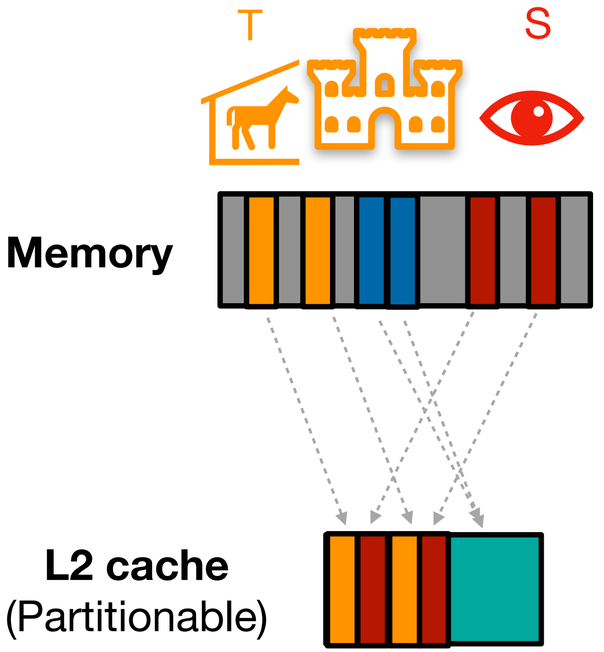
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated “legacy” implementation options on seL4, with the L2 partitioned via *cache colouring*:

Varun Sethu
BE Thesis



We need *targeted flush* for partitionable L2 caches to enforce T ~/> S while allowing T <~ S

Time-protected communications in seL4

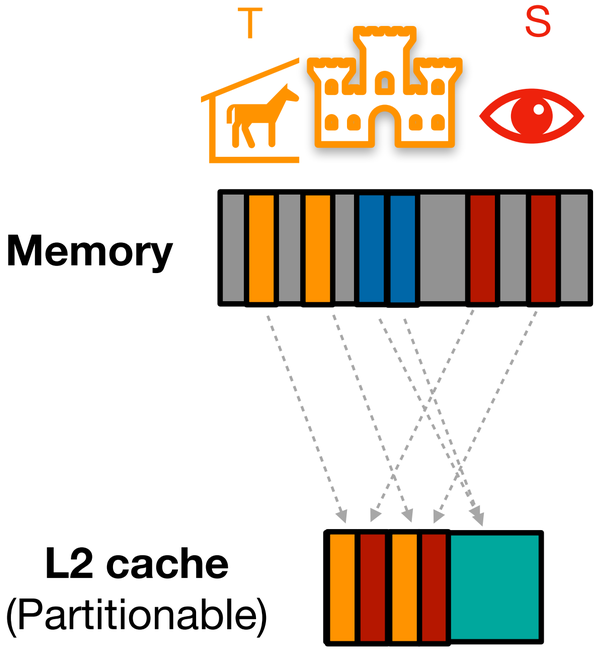
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated “legacy” implementation options on seL4, with the L2 partitioned via *cache colouring*:
  - Copy between non-shared memory (slow)
  - Dedicate colours for shared memory (wastes space)

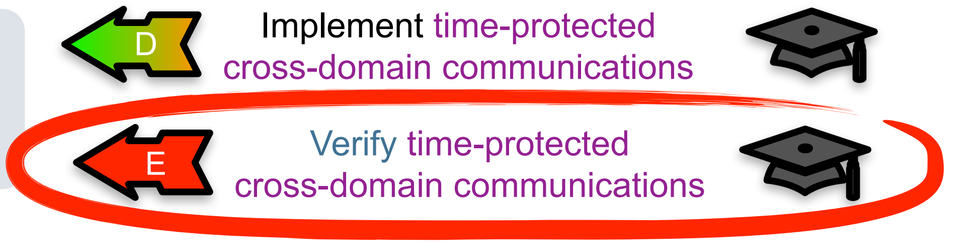
Varun Sethu
BE Thesis



We need *targeted flush* for partitionable L2 caches to enforce T ~/> S while allowing T <~ S

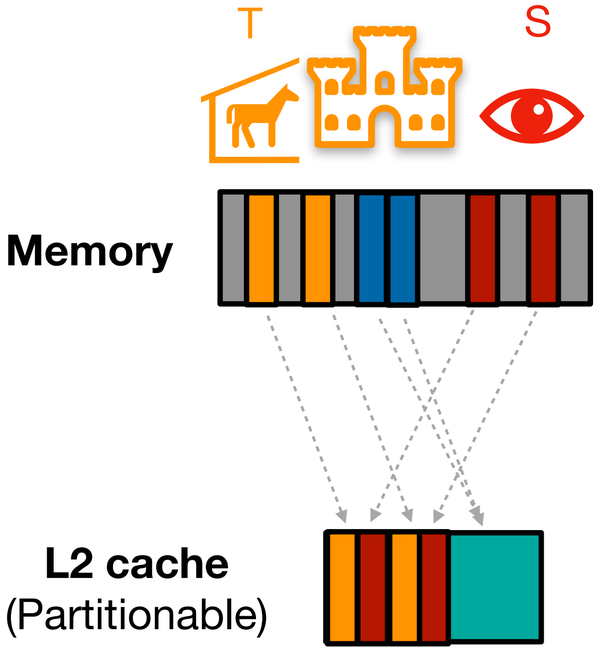
Time-protected communications in seL4

seL4 on RISC-V



- [Sethu 2025] comprehensively evaluated “legacy” implementation options on seL4, with the L2 partitioned via *cache colouring*:
  - Copy between non-shared memory (slow)
  - Dedicate colours for shared memory (wastes space)
  - Targeted L2 “flush” via prefetching (still leaks; *infeasible to verify* w/o model of replacement policy!)

Varun Sethu
BE Thesis



We need *targeted flush* for partitionable L2 caches to enforce T ~/> S while allowing T <~ S

Time-protected communications in seL4

seL4 on RISC-V

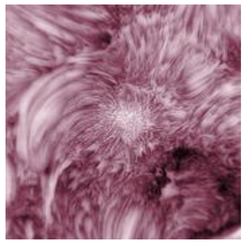
D — Implement time-protected cross-domain communications ✓

E — Verify time-protected cross-domain communications

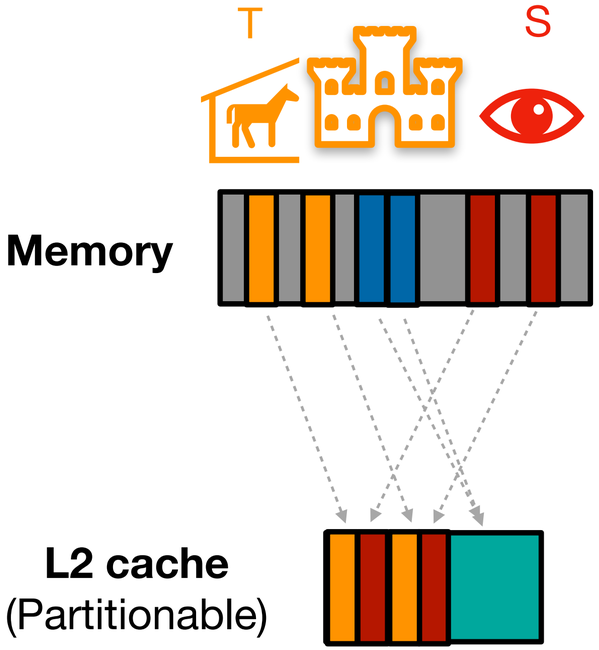
- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with the L2 partitioned via *cache colouring*:
  - Copy between non-shared memory (slow)
  - Dedicate colours for shared memory (wastes space)
  - Targeted L2 "flush" via prefetching (still leaks; *infeasible to verify* w/o model of replacement policy!)
- Now we're using *dynamically partitionable last-level cache* (DPLLC) i.e. dedicated HW support on RISC-V Cheshire

Varun Sethu
BE Thesis

Continuing on from



Julia Vassiliki



We need *targeted flush* for partitionable L2 caches to enforce T ~/> S while allowing T <~ S

Time-protected communications in seL4

seL4 on RISC-V

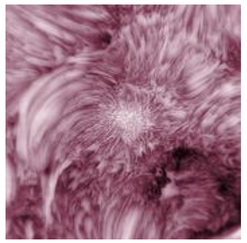
D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

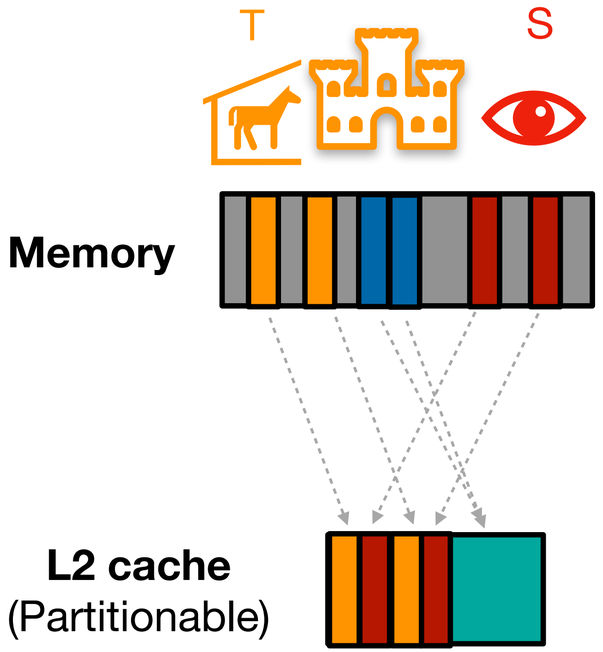
- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with the L2 partitioned via *cache colouring*:
  - Copy between non-shared memory (slow)
  - Dedicate colours for shared memory (wastes space)
  - Targeted L2 "flush" via prefetching (still leaks; *infeasible to verify* w/o model of replacement policy!)
- Now we're using *dynamically partitionable last-level cache* (DPLLC) i.e. dedicated HW support on RISC-V Cheshire
  - Supports targeted flush of partitions
  - Adding needed time padding on flush

Varun Sethu
BE Thesis

Continuing on from



Julia Vassiliki



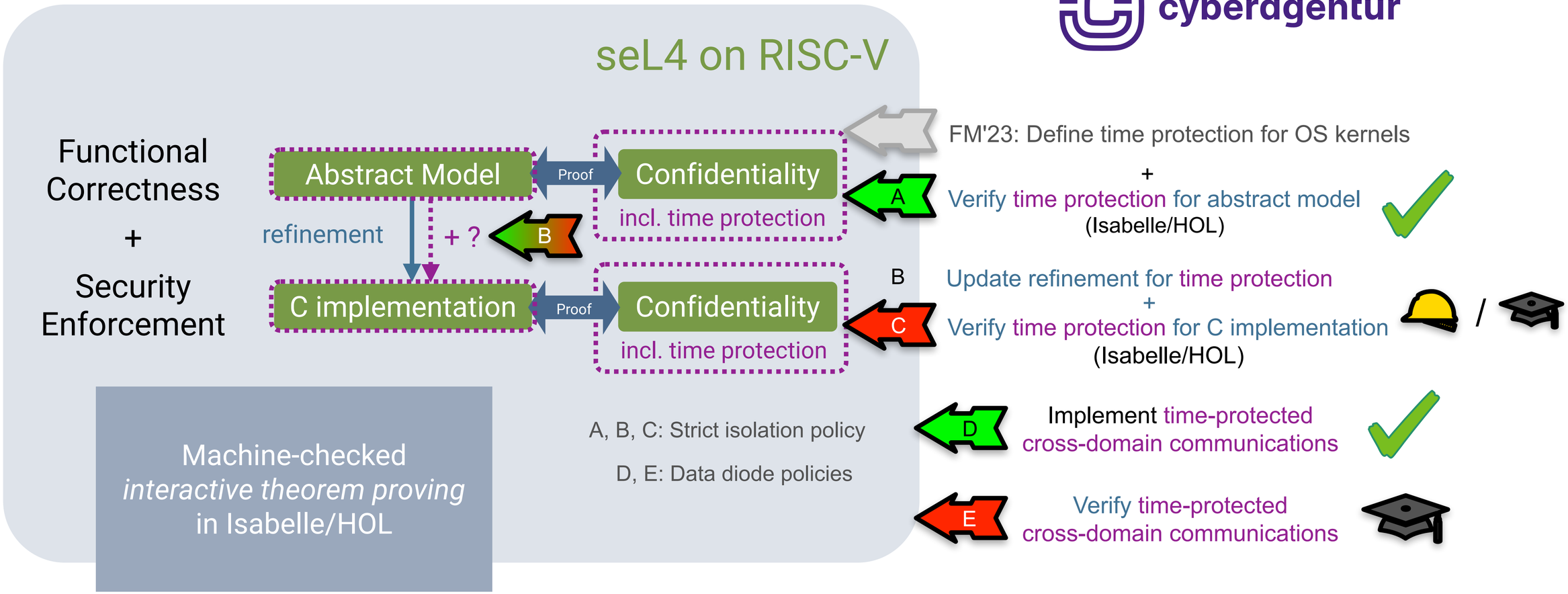
We need *targeted flush* for partitionable L2 caches to enforce T ~/> S while allowing T <~ S

Time protection verification for seL4

The status today:

Thanks to funding from

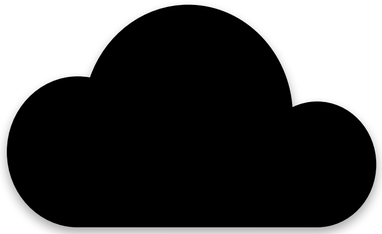
cyberagentur



II: Kernel–Userland Gap

seL4: World's first OS kernel with correctness proof!

Question:

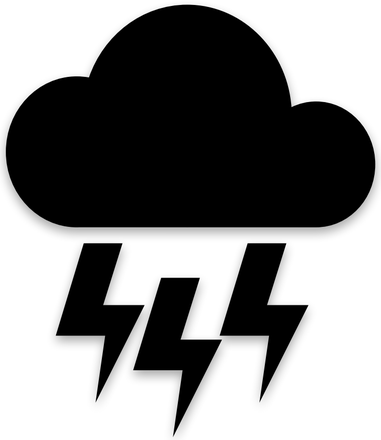


seL4: World's first OS kernel with correctness proof!

Question:

What do we mean when we say "seL4 is functionally correct"?

seL4: World's first OS kernel with correctness proof!



Question:

What do we mean when we say "seL4 is functionally correct"?

Claim:

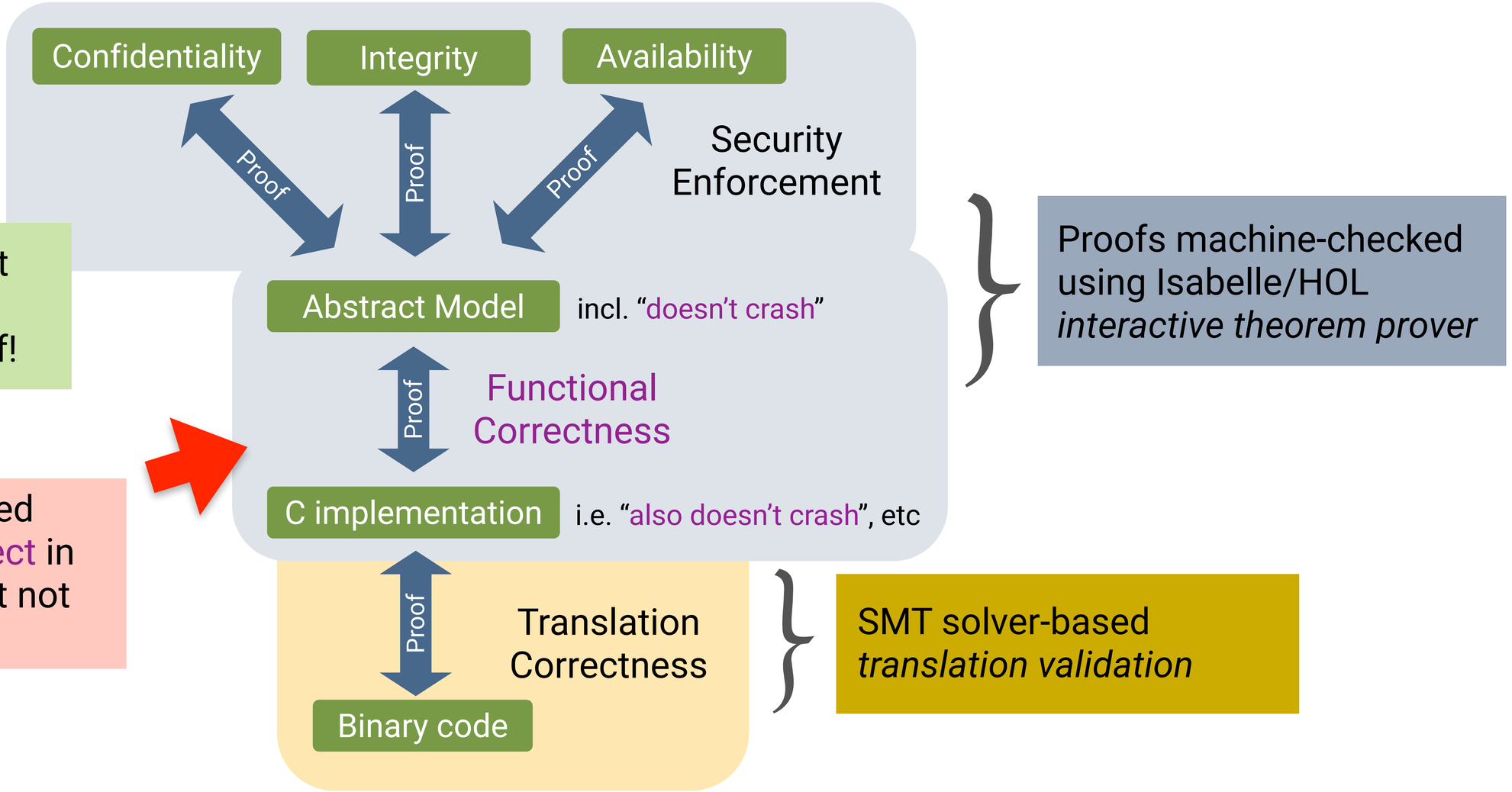
seL4 has *not* been proved functionally correct enough to verify its users!

The verified seL4 OS microkernel

seL4: World's first OS kernel with correctness proof!

But: seL4 is proved functionally correct in some senses, but not others needed

The verified seL4 OS microkernel



seL4: World’s first OS kernel with correctness proof!

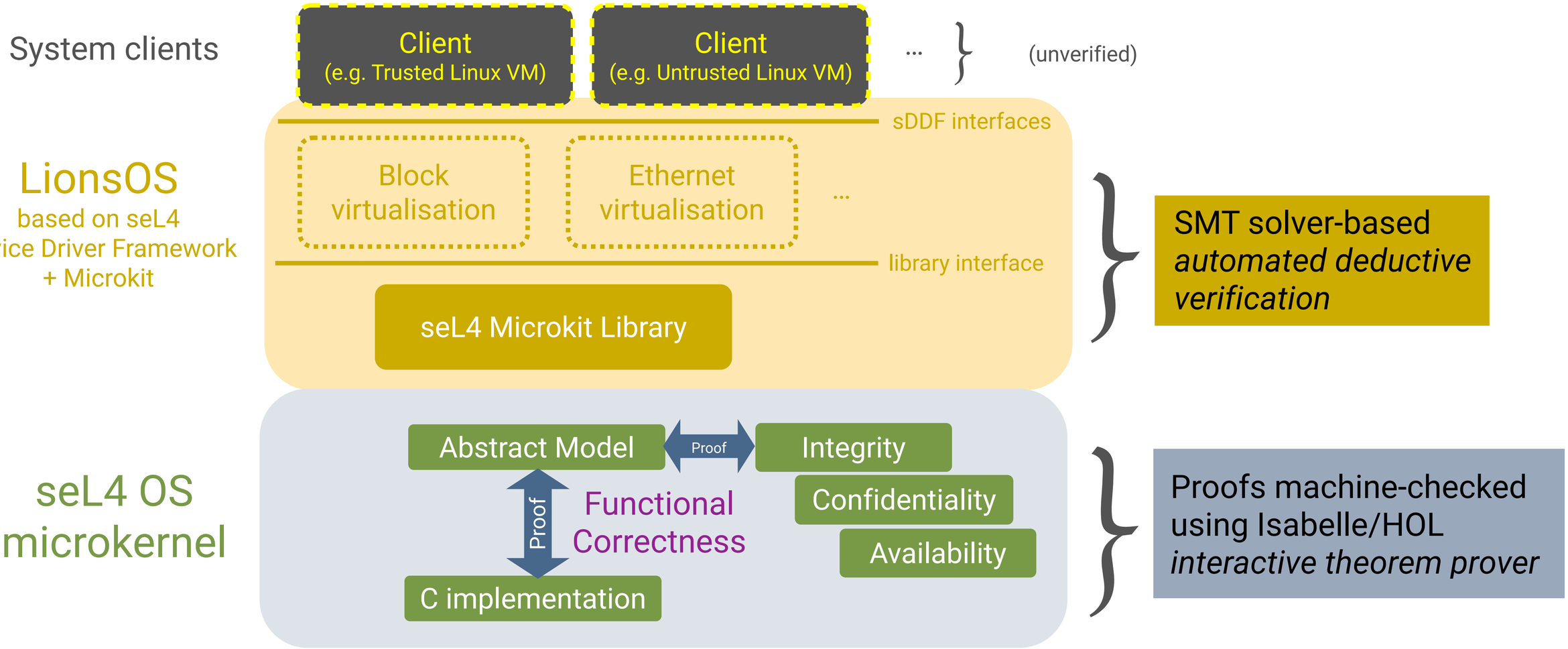
But: seL4 is proved functionally correct in some senses, but not others needed

Verifying seL4-based OSes: Vision

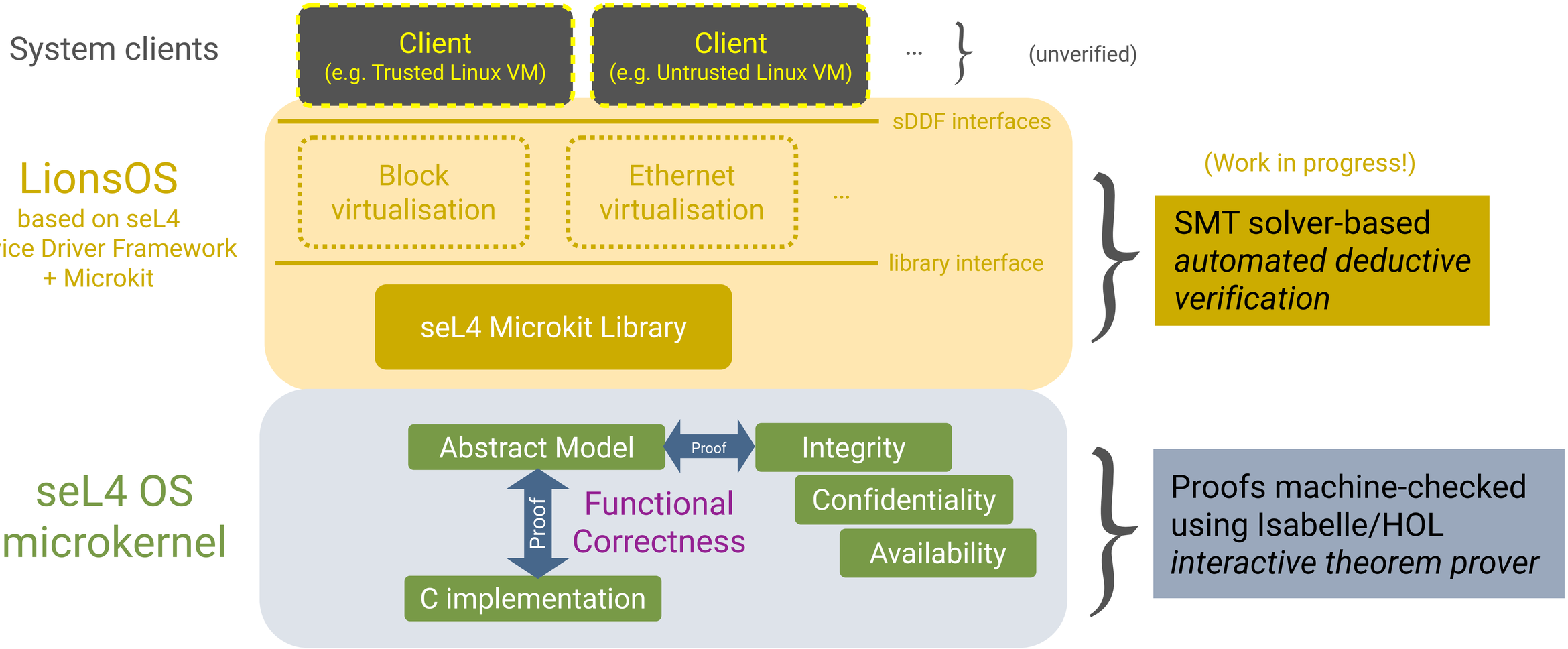
seL4 OS microkernel

Proofs machine-checked using Isabelle/HOL *interactive theorem prover*

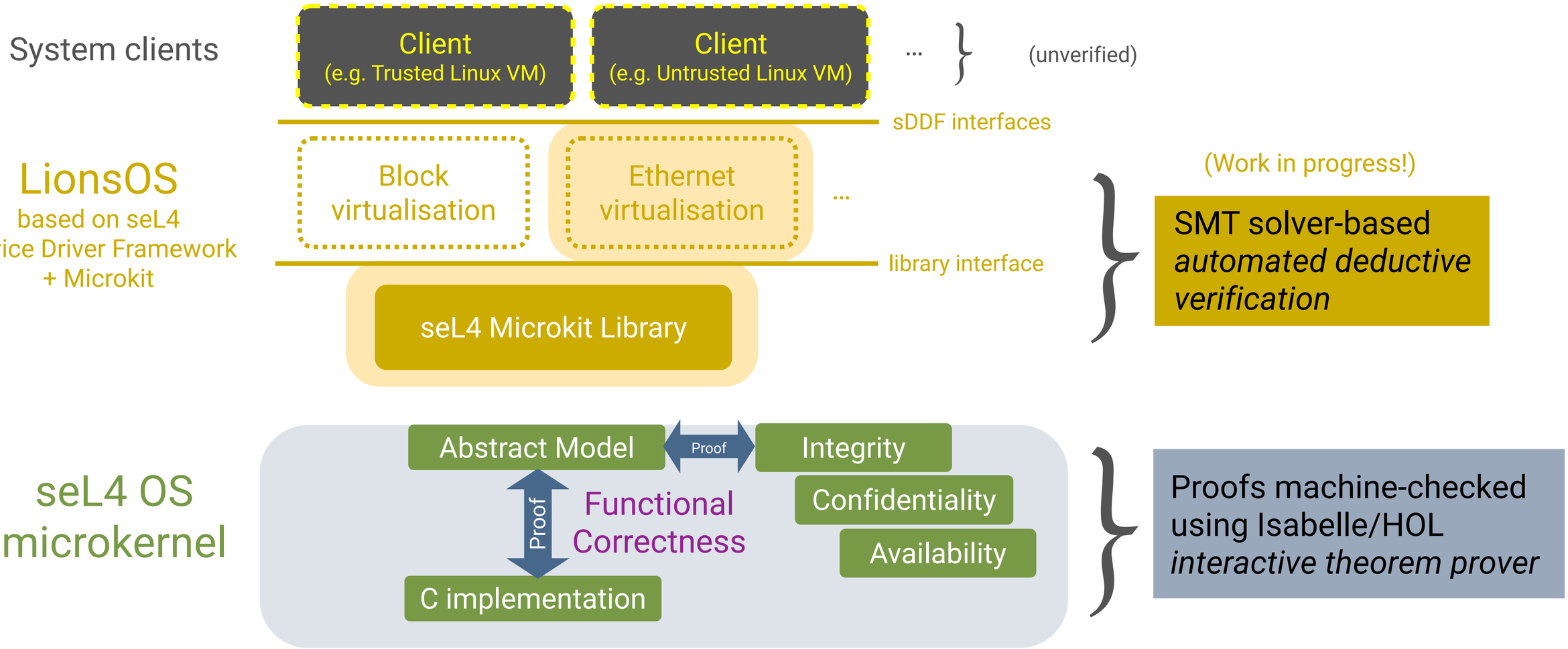
Verifying seL4-based OSes: Vision



Verifying seL4-based OSes: Vision

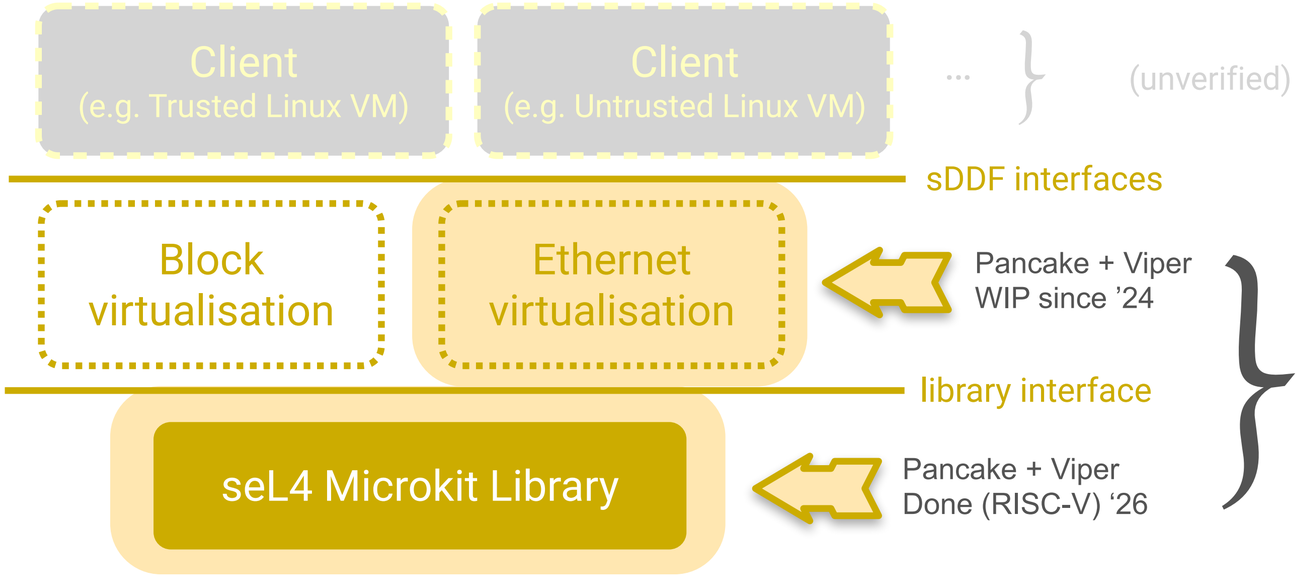


Verifying seL4-based OSes: Status



Verifying seL4-based OSes: Status

System clients

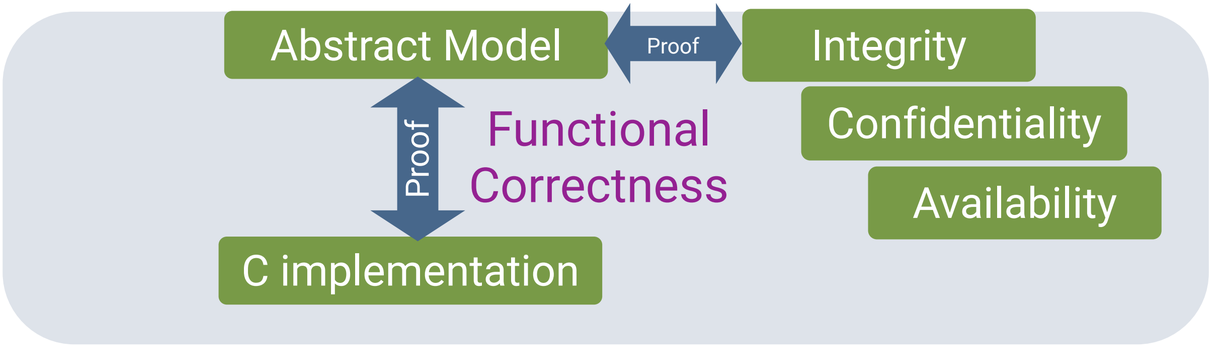


LionsOS

based on seL4
Device Driver Framework
+ Microkit

(Work in progress!)

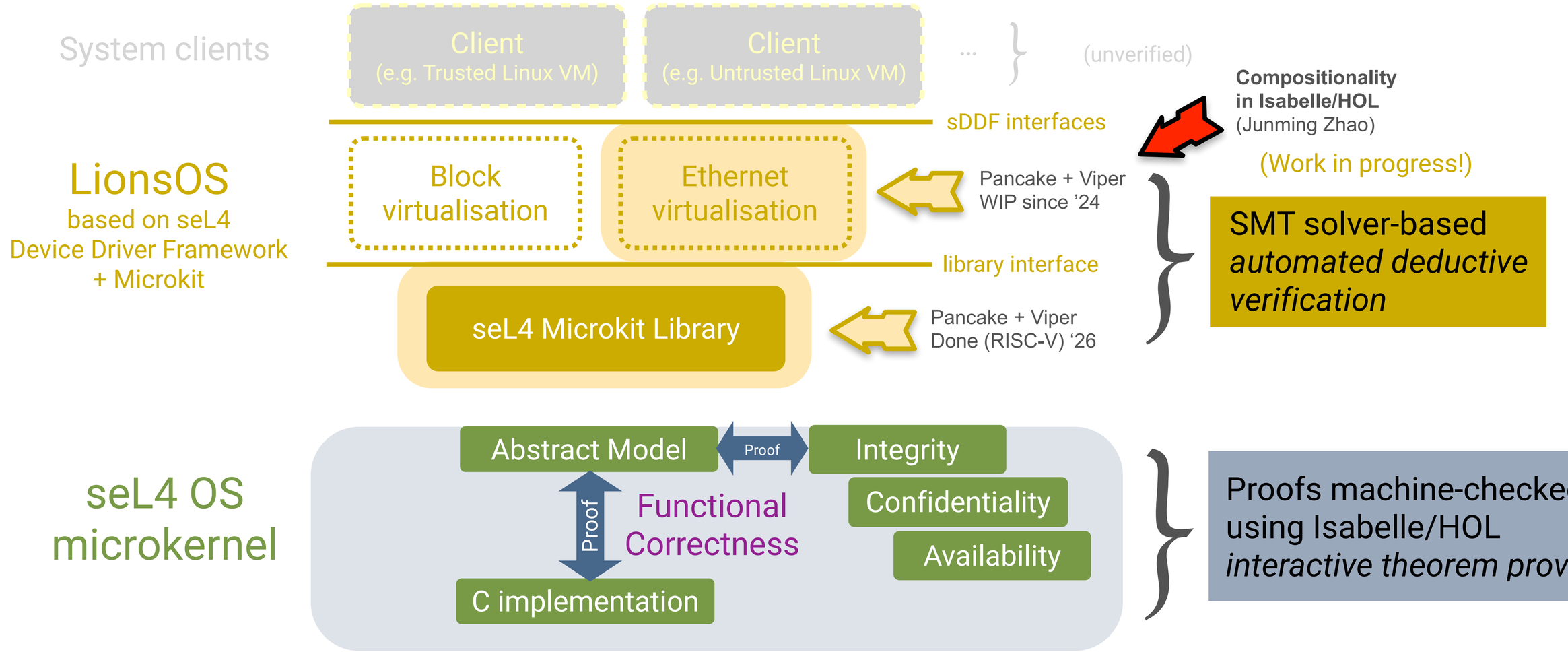
SMT solver-based
automated deductive verification



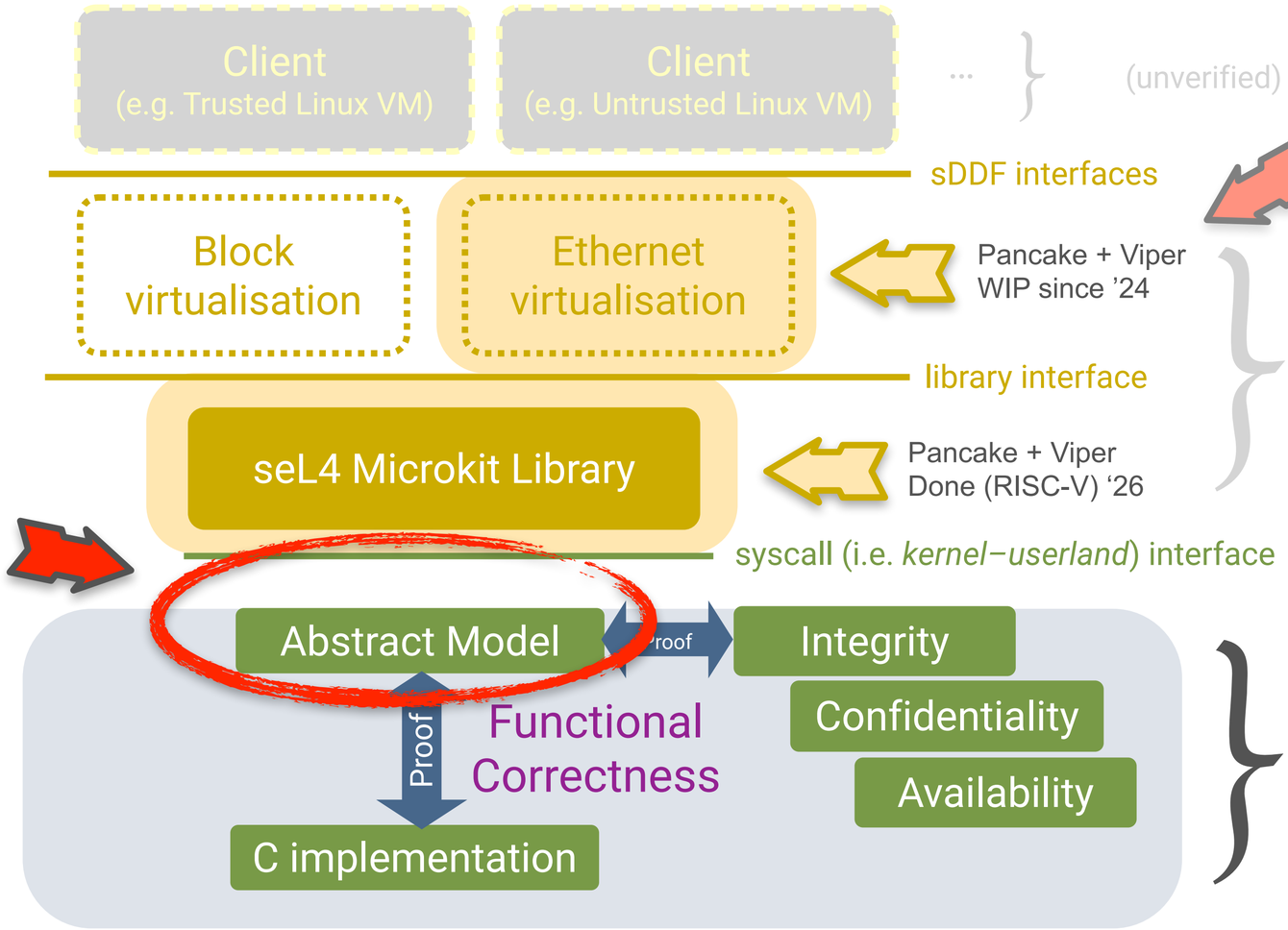
seL4 OS microkernel

Proofs machine-checked
using Isabelle/HOL
interactive theorem prover

Verifying seL4-based OSes: Status



Verifying seL4-based OSes: Status



Compositionality in Isabelle/HOL
(Junming Zhao)

(Work in progress!)

SMT solver-based *automated deductive verification*

LionsOS
based on seL4
Device Driver Framework
+ Microkit

This talk (pt II):
Kernel–Userland Gap

seL4 OS microkernel

Proofs machine-checked using Isabelle/HOL *interactive theorem prover*

Microkit assumes seL4's syscalls work

```
static void handler_loop(void) {
    …

        if (have_reply) {
            …
        } else {


            tag = seL4_Recv(INPUT_CAP, &badge, REPLY_CAP);


        }

    …
}
```

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

Microkit assumes seL4's syscalls work

```
static void handler_loop(void) {
    …
        {{ R }}                          Precondition
        if (have_reply) {
            {{ R && … }} … {{ S }}
        } else {
            {{ R && ! have_reply && ! microkit_have_signal }}


            tag = seL4_Recv(INPUT_CAP, &badge, REPLY_CAP);


            {{ S }}
        }
        {{ S }}
    …                                    Postcondition
}
```

Hoare triples:
{{ Pre }} f {{ Post }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

Microkit assumes seL4's syscalls work

```
static void handler_loop(void) {
    …
        {{ R }}                                   Precondition
        if (have_reply) {
            {{ R && … }} … {{ S }}
        } else {
            {{ R && ! have_reply && ! microkit_have_signal }}
            // if R && … => P, assume
Precond.    {{ P }}
            tag = seL4_Recv(INPUT_CAP, &badge, REPLY_CAP);
            {{ Q }}
            // if Q => S, conclude
Postcond.   {{ S }}
        }
        {{ S }}
    …                                             Postcondition
}
```

Hoare triples:
{{ Pre }} f {{ Post }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

Microkit assumes seL4's syscalls work

```
static void handler_loop(void) {
    …
    {{ R }}                                   Precondition
    if (have_reply) {
        {{ R && … }} … {{ S }}
    } else {
        {{ R && ! have_reply && ! microkit_have_signal }}
        // if R && … => P, assume
        {{ P }}
        tag = seL4_Recv(INPUT_CAP, &badge, REPLY_CAP);
        {{ Q }}
        // if Q => S, conclude
        {{ S }}
    }
    {{ S }} // done
    …                                         Postcondition
}
```

Precond.

Postcond.

Hoare triples:
{{ Pre }} f {{ Post }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

Microkit assumes seL4's syscalls work

But is it true?

Hoare triples:
{{ Pre }} f {{ Post }}

seL4 Microkit Library

```
static void handler_loop(void) {
    …
        {{ R }}                                   Precondition
        if (have_reply) {
            {{ R && … }} … {{ S }}
        } else {
            {{ R && ! have_reply && ! microkit_have_signal }}
Precond.    // if R && … => P, assume
            {{ P }}
            tag = seL4_Recv(INPUT_CAP, &badge, REPLY_CAP);
            {{ Q }}
Postcond.   // if Q => S, conclude
            {{ S }}
?       }
        {{ S }} // done
    …                                             Postcondition
}
```

syscall (i.e. *kernel–userland*) interface

Microkit's assumptions need verifying!

```
static void handler_loop(void) { …
        {{ R && ! have_reply && ! microkit_have_signal }}
        // if R && … => P, assume
        {{ P }}
        tag = seL4_Recv(INPUT_CAP, &badge, REPLY_CAP);
        {{ Q }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

seL4 OS Microkernel Abstract Model

Microkit's assumptions need verifying!

```
static void handler_loop(void) { …
        {{ R && ! have_reply && ! microkit_have_signal }}
        // if R && … => P, assume
        {{ P }}
        tag = seL4_Recv(INPUT_CAP, &badge, REPLY_CAP);
        {{ Q }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

```
definition call_kernel :: "event ⇒ (unit, …) s_monad" where

  call_kernel ev ≡ do
      handle_event ev …;
      schedule;
      activate_thread
  od
```

seL4 OS Microkernel Abstract Model

Microkit's assumptions need verifying!

```
static void handler_loop(void) { …
        {{ R && ! have_reply && ! microkit_have_signal }}
        // if R && … => P, assume
        {{ P }}
        tag = seL4_Recv(INPUT_CAP, &badge, REPLY_CAP);
        {{ Q }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

```
definition call_kernel :: "event ⇒ (unit, …) s_monad" where
 {{ P }}
 call_kernel ev ≡ do
      handle_event ev …;
      schedule;
      activate_thread
 od
 {{ Q }} ?
```

seL4 OS Microkernel Abstract Model

Microkit's assumptions need verifying!

```
static void handler_loop(void) { …
        {{ R && ! have_reply && ! microkit_have_signal }}
        // if R && … => P, assume
        {{ P }}
        tag = seL4_Recv(INPUT_CAP, &badge, REPLY_CAP);
        {{ Q }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

```
definition call_kernel :: "event ⇒ (unit, …) s_monad" where
 {{ P }}                {{ invs }}
 call_kernel ev ≡ do
     handle_event ev …;
     schedule;
     activate_thread
 od
 {{ Q }} ?              {{ invs }}   incl. "doesn't crash"
```

seL4 OS Microkernel Abstract Model

Microkit's assumptions need verifying!

```
static void handler_loop(void) { …
        {{ R && ! have_reply && ! microkit_have_signal }}
        // if R && … => P, assume
        {{ P }}
        tag = seL4_Recv(INPUT_CAP, &badge, REPLY_CAP);
        {{ Q }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

```
definition call_kernel :: "event ⇒ (unit, …) s_monad" where
 {{ P }}
 call_kernel ev ≡ do
      handle_event ev …;
      schedule;
      activate_thread
 od
 {{ Q }} ?
```

Question:

Is proving a Hoare triple over one call_kernel entry always enough?

seL4 OS Microkernel Abstract Model

Microkit's assumptions need verifying!

```
static void handler_loop(void) { …
        {{ R && ! have_reply && ! microkit_have_signal }}
        // if R && … => P, assume
        {{ P }}
        tag = seL4_Recv(INPUT_CAP, &badge, REPLY_CAP);
        {{ Q }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

```
definition call_kernel :: "event ⇒ (unit, …) s_monad" where
 {{ P }}
 call_kernel ev ≡ do
      handle_event ev …;
      schedule;
      activate_thread
 od
 {{ Q }} ?
```

Question:

Is proving a Hoare triple over one call_kernel entry always enough?

No.

seL4 OS Microkernel Abstract Model

System calls can block

Process A

```
static void handler_loop(void) { …
        {{ R && … }}
        // if R && … => P, assume
        {{ PA }}
        tag = seL4_Recv(…);
        {{ QA }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

Blocking seL4_Recv call

```
{{ PA }} => {{ P' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
            schedule;
            activate_thread
        od
{{ QA }} ✗ {{ Q' && … }}
```

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

seL4 OS Microkernel Abstract Model

System calls can block

Process A

```
static void handler_loop(void) { …
        {{ R && … }}
        // if R && … => P, assume
        {{ PA }}
        tag = seL4_Recv(…);
        {{ QA }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

Process B

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

Blocking seL4_Recv call

```
{{ PA }} => {{ P' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
            schedule;
            activate_thread // B
        od
        {{ Q' && … }}
```

seL4 OS Microkernel Abstract Model

System calls can block

Process A

```
static void handler_loop(void) { …
        {{ R && … }}
        // if R && … => P, assume
        {{ PA }}
        tag = seL4_Recv(…);
        {{ QA }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

Process B

Blocking seL4_Recv call

```
{{ PA }} => {{ P' && … }}
            call_kernel ev ≡ do
                handle_event ev …;
                schedule;
                activate_thread // B
            od
            {{ Q' && … }}
```

→ **Unblocking** seL4_Signal call

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

seL4 OS Microkernel Abstract Model

System calls can block

Process A

```
static void handler_loop(void) { …
        {{ R && … }}
        // if R && … => P, assume
        {{ PA }}
        tag = seL4_Recv(…);
        {{ QA }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

Process B

```
static inline void
microkit_notify(microkit_channel ch) { …
    {{ PB }}
    seL4_Signal(… + ch);
    {{ QB }}
}
```

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

Blocking seL4_Recv call

```
{{ PA }} => {{ P' && … }}
            call_kernel ev ≡ do
                handle_event ev …;
                schedule;
                activate_thread // B
            od
            {{ Q' && … }}
```

→ **Unblocking** seL4_Signal call

seL4 OS Microkernel Abstract Model

System calls can block

Verifying these is an unsolved problem*

Process A

```
static void handler_loop(void) { …
        {{ R && … }}
        // if R && … => P, assume
        {{ PA }}
        tag = seL4_Recv(…);
        {{ QA }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

Process B

```
static inline void
microkit_notify(microkit_channel ch) { …
    {{ PB }}
    seL4_Signal(… + ch);
    {{ QB }}
}
```

seL4 Microkit Library

* to the best of our knowledge, and when they block on other processes (not just I/O!)

syscall (i.e. *kernel–userland*) interface

Blocking seL4_Recv call

```
{{ PA }} => {{ P' && … }}
            call_kernel ev ≡ do
                handle_event ev …;
                schedule;
                activate_thread // B
            od
            {{ Q' && … }}
```

→ **Unblocking** seL4_Signal call

seL4 OS Microkernel Abstract Model

System calls can block

Verifying these is an unsolved problem*

Process A

```
static void handler_loop(void) { …
        {{ R && … }}
        // if R && … => P, assume
        {{ PA }}
        tag = seL4_Recv(…);
        {{ QA }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

Process B

```
static inline void
microkit_notify(microkit_channel ch) { …
    {{ PB }}
    seL4_Signal(… + ch);
    {{ QB }}
}
```

seL4 Microkit Library

* to the best of our knowledge, and when they block on other processes (not just I/O!)

syscall (i.e. *kernel–userland*) interface

Blocking seL4_Recv call

```
{{ PA }} => {{ P' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
            schedule;
            activate_thread // B
        od
        {{ Q' && … }}
```

Unblocking seL4_Signal call

```
{{ PB }} => {{ P'' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
            schedule;
            activate_thread
        od
        {{ Q'' && … }}
```

seL4 OS Microkernel Abstract Model

System calls can block

Verifying these is an unsolved problem*

Process A

```
static void handler_loop(void) { …
        {{ R && … }}
        // if R && … => P, assume
        {{ PA }}
        tag = seL4_Recv(…);
        {{ QA }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

Process B

```
static inline void
microkit_notify(microkit_channel ch) { …
    {{ PB }}
    seL4_Signal(… + ch);
    {{ QB }}
}
```

seL4 Microkit Library

* to the best of our knowledge, and when they block on other processes (not just I/O!)

syscall (i.e. *kernel–userland*) interface

Blocking seL4_Recv call

```
{{ PA }} => {{ P' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
            schedule;
            activate_thread // B
        od
        {{ Q' && … }}
```

Unblocking seL4_Signal call

```
{{ PB }} => {{ P'' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
            schedule;
            activate_thread // A
        od
        {{ Q'' && … }} => {{ QA }}
```

seL4 OS Microkernel Abstract Model

System calls can block

Verifying these is an unsolved problem*

Process A

```
static void handler_loop(void) { …
        {{ R && … }}
        // if R && … => P, assume
        {{ PA }}
        tag = seL4_Recv(…);
        {{ QA }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

Process B

```
static inline void
microkit_notify(microkit_channel ch) { …
    {{ PB }}
    seL4_Signal(… + ch);
    {{ QB }}
}
```

seL4 Microkit Library

* to the best of our knowledge, and when they block on other processes (not just I/O!)

syscall (i.e. *kernel–userland*) interface

Blocking seL4_Recv call

```
{{ PA }} => {{ P' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
            schedule;
            activate_thread // B
        od
        {{ Q' && … }}
```

Unblocking seL4_Signal call

```
{{ PB }} => {{ P'' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
            schedule;
            activate_thread // A
        od
        {{ Q'' && … }} => {{ QA }}
```

seL4 OS Microkernel Abstract Model

System calls can block

Verifying these is an unsolved problem*

Process A

```
static void handler_loop(void) { …
        {{ R && … }}
        // if R && … => P, assume
        {{ PA }}
        tag = seL4_Recv(…);
        {{ QA }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

Process B

```
static inline void
microkit_notify(microkit_channel ch) { …
    {{ PB }}
    seL4_Signal(… + ch);
    {{ QB }}
}
```

seL4 Microkit Library

* to the best of our knowledge, and when they block on other processes (not just I/O!)

syscall (i.e. *kernel–userland*) interface

Blocking seL4_Recv call

```
{{ PA }} => {{ P' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
          ? schedule;
            activate_thread // X
        od
        {{ Q' && … }}
```

Unblocking seL4_Signal call

```
{{ PB }} => {{ P'' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
            schedule;
            activate_thread // A
        od
{{ Q'' && … }} => {{ QA }}
```

seL4 OS Microkernel Abstract Model

System calls can block

Verifying these is an unsolved problem*

Process A

```
static void handler_loop(void) { …
        {{ R && … }}
        // if R && … => P, assume
        {{ PA }}
        tag = seL4_Recv(…);
        {{ QA }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

Process B

```
static inline void
microkit_notify(microkit_channel ch) { …
    {{ PB }}
    seL4_Signal(… + ch);
    {{ QB }}
}
```

seL4 Microkit Library

* to the best of our knowledge, and when they block on other processes (not just I/O!)

syscall (i.e. *kernel–userland*) interface

Blocking seL4_Recv call

```
{{ PA }} => {{ P' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
          ? schedule;
            activate_thread // X
        od
        {{ Q' && … }}
```

X, Y, Z … B

Unblocking seL4_Signal call

```
{{ PB }} => {{ P'' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
            schedule;
            activate_thread // A
        od
{{ Q'' && … }} => {{ QA }}
```

seL4 OS Microkernel Abstract Model

System calls can block

Verifying these is an unsolved problem*

Process A

```
static void handler_loop(void) { …
        {{ R && … }}
        // if R && … => P, assume
        {{ PA }}
        tag = seL4_Recv(…);
        {{ QA }}
        // if Q => S, conclude
        {{ S }}
    …
}
```

Process B

```
static inline void
microkit_notify(microkit_channel ch) { …
    {{ PB }}
    seL4_Signal(… + ch);
    {{ QB }}
}
```

seL4 Microkit Library

* to the best of our knowledge, and when they block on other processes (not just I/O!)

syscall (i.e. *kernel–userland*) interface

Blocking seL4_Recv call

```
{{ PA }} => {{ P' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
          ? schedule;
            activate_thread // X
        od
        {{ Q' && … }}
```

X, Y, Z … B

Irrelevant calls (IRQs too!)

Unblocking seL4_Signal call

```
{{ PB }} => {{ P'' && … }}
        call_kernel ev ≡ do
            handle_event ev …;
            schedule;
            activate_thread // A
        od
{{ Q'' && … }} => {{ QA }}
```

seL4 OS Microkernel Abstract Model

Towards kernel–userland functional proofs

Thanks to
cyberagentur
FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

{{ P }}
call_kernel
{{ Q }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

{{ P' }}
call_kernel
{{ Q' }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

seL4 OS Microkernel Abstract Model

Towards kernel–userland functional proofs

- It helps Microkit's assumptions are on *per-process* projections of kernel state:

Read:
ks_of = "kernel state of"

Thanks to

cyberagentur

FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

{{ λs. P (ks_of A s) }}
call_kernel
{{ λs. Q (ks_of A s) }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

{{ λs. P' (ks_of B s) }}
call_kernel
{{ λs. Q' (ks_of B s) }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

seL4 OS Microkernel Abstract Model

```
ks_of p s ≡ (
    thread_cnode p s,
    bound_notification p s,
    mapped_writable p s,
    not_writable_others p s,
    recv_oracle s (bound_notification p s) (thread_cnode p s),
    λch. ntfn_status s (thread_cnode p s) ch
)
```

Towards kernel–userland functional proofs

- It helps Microkit's assumptions are on *per-process* projections of kernel state:

Read:
ks_of = "kernel state of"

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

{{ λs. P (ks_of A s) }}
call_kernel
{{ λs. Q (ks_of A s) }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

{{ λs. P' (ks_of B s) }}
call_kernel
{{ λs. Q' (ks_of B s) }}

Thanks to
cyberagentur
FORESIGHT INSTITUTE

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

seL4 OS Microkernel Abstract Model

- On Microkit systems,
no X ≠ Y should possess the *capabilities*
to change ***most parts*** of (ks_of Y s).

ks_of p s ≡ (
  thread_cnode p s,
  bound_notification p s, } **cap state, thread init!**
  mapped_writable p s,
  not_writable_others p s, } **memory mappings!**
  recv_oracle s (bound_notification p s) (thread_cnode p s),
  λch. ntfn_status s (thread_cnode p s) ch
)

Towards kernel–userland functional proofs

- It helps Microkit's assumptions are on *per-process* projections of kernel state:

Read:
ks_of = "kernel state of"

Thanks to

cyberagentur FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

{{ λs. P (ks_of A s) }}
call_kernel
{{ λs. Q (ks_of A s) }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

{{ λs. P' (ks_of B s) }}
call_kernel
{{ λs. Q' (ks_of B s) }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

seL4 OS Microkernel Abstract Model

- On Microkit systems, no X ≠ Y should possess the *capabilities* to change ***most parts*** of (ks_of Y s).

- But we still have to prove that *the assumed changes happen*.

ks_of p s ≡ ⦇
thread_cnode p s,
bound_notification p s, } **cap state, thread init!**
mapped_writable p s,
not_writable_others p s, } **memory mappings!**
recv_oracle s (bound_notification p s) (thread_cnode p s),
λch. ntfn_status s (thread_cnode p s) ch
⦈

What we're doing

- Defined state projections

Read:
ks_of = "kernel state of"

Thanks to
cyberagentur
FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
call_kernel
{{ λs. Q (ks_of A s) }}

{{ λs. P' (ks_of B s) }}
call_kernel
{{ λs. Q' (ks_of B s) }}

seL4 OS Microkernel Abstract Model

Microkit-facing state ks_of p s ≡

- thread_cnode p s,
- bound_notification p s, } **cap state, thread init!**
- mapped_writable p s,
- not_writable_others p s, } **memory mappings!**
- recv_oracle s (bound_notification p s) (thread_cnode p s),
- λch. ntfn_status s (thread_cnode p s) ch

≡

What we're doing

- Defined state projections

Read:
ks_of = "kernel state of"

Thanks to
cyberagentur
FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

{{ λs. P (ks_of A s) }}
call_kernel
{{ λs. Q (ks_of A s) }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

{{ λs. P' (ks_of B s) }}
call_kernel
{{ λs. Q' (ks_of B s) }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

seL4 OS Microkernel Abstract Model

Microkit-facing state ks_of p s ≡

Abstract Model state

- thread_cnode p s,
- bound_notification p s,
- mapped_writable p s,
- not_writable_others p s,
- recv_oracle s (bound_notification p s) (thread_cnode p s),
- λch. ntfn_status s (thread_cnode p s) ch

} **cap state, thread init!**

} **memory mappings!**

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to
cyberagentur
FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

{{ λs. P (ks_of A s) }}
call_kernel
{{ λs. Q (ks_of A s) }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

{{ λs. P' (ks_of B s) }}
call_kernel
{{ λs. Q' (ks_of B s) }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

seL4 OS Microkernel Abstract Model

Abstract Model state

```
ks_of p s ≡ (
    thread_cnode p s,
    bound_notification p s,
    mapped_writable p s,
    not_writable_others p s,
    recv_oracle s (bound_notification p s) (thread_cnode p s),
    λch. ntfn_status s (thread_cnode p s) ch
)
```

cap state, thread init!

memory mappings!

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to
cyberagentur
FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

seL4 OS Microkernel Abstract Model

{{ λs. P (ks_of A s) }}
call_kernel
{{ λs. Q (ks_of A s) }}

{{ λs. P' (ks_of B s) }}
call_kernel
{{ λs. Q' (ks_of B s) }}

```
call_kernel ev ≡ do
    handle_event ev …;
    schedule;
    activate_thread
od
```

Abstract Model state

```
ks_of p s ≡ ⦇
    thread_cnode p s,
    bound_notification p s,
    mapped_writable p s,
    not_writable_others p s,
    recv_oracle s (bound_notification p s) (thread_cnode p s),
    λch. ntfn_status s (thread_cnode p s) ch
⦈
```

cap state, thread init!

memory mappings!

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to
cyberagentur
FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

seL4 OS Microkernel Abstract Model

{{ λs. P (ks_of A s) }}
call_kernel
{{ λs. Q (ks_of A s) }}

{{ λs. P' (ks_of B s) }}
call_kernel
{{ λs. Q' (ks_of B s) }}

```
call_kernel ev ≡ do
    handle_event ev …;
    schedule;
    activate_thread
od
```

we'll come back to this

Abstract Model state

```
ks_of p s ≡ (
    thread_cnode p s,
    bound_notification p s,
    mapped_writable p s,
    not_writable_others p s,
    recv_oracle s (bound_notification p s) (thread_cnode p s),
    λch. ntfn_status s (thread_cnode p s) ch
)
```

cap state, thread init!

memory mappings!

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to
cyberagentur
FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

seL4 OS Microkernel Abstract Model

```
call_kernel ev ≡ do
    handle_event ev …;
    schedule;
    activate_thread
od
```

} we'll come back to this

Abstract Model state

```
ks_of p s ≡ (|
    thread_cnode p s,
    bound_notification p s,
    mapped_writable p s,
    not_writable_others p s,
    recv_oracle s (bound_notification p s) (thread_cnode p s),
    λch. ntfn_status s (thread_cnode p s) ch
|)
```

} cap state, thread init!

} memory mappings!

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to
cyberagentur
FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

seL4 OS Microkernel Abstract Model

```
call_kernel ev ≡ do
    handle_event ev …;
    schedule;
    activate_thread
od
```

} we'll come back to this

Abstract Model state

```
ks_of p s ≡ (
    thread_cnode p s,
    bound_notification p s,
    mapped_writable p s,
    not_writable_others p s,
    recv_oracle s (bound_notification p s) (thread_cnode p s),
    λch. ntfn_status s (thread_cnode p s) ch
)
```

} cap state, thread init!

} memory mappings!

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to
cyberagentur
FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"
✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

Blocking
✓ **Unblocking**

seL4 OS Microkernel Abstract Model

```
call_kernel ev ≡ do
    handle_event ev …;
    schedule;
    activate_thread
od
```

} we'll come back to this

Abstract Model state

```
ks_of p s ≡ (
    thread_cnode p s,
    bound_notification p s,
    mapped_writable p s,
    not_writable_others p s,
    recv_oracle s (bound_notification p s) (thread_cnode p s),
    λch. ntfn_status s (thread_cnode p s) ch
)
```

} **cap state, thread init!**

} **memory mappings!**

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to
cyberagentur
FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"
✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

Blocking
✓ **Unblocking**

seL4 OS Microkernel Abstract Model

What we've proved:

- More precise cap lookup lemmas

```
ks_of p s ≡ (
    thread_cnode p s,
    bound_notification p s,                    } cap state, thread init!
    mapped_writable p s,
    not_writable_others p s,                   } memory mappings!
    recv_oracle s (bound_notification p s) (thread_cnode p s),
    λch. ntfn_status s (thread_cnode p s) ch
)
```

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to
cyberagentur
FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"
✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

Blocking
✓ **Unblocking**

seL4 OS Microkernel Abstract Model

What we've proved:

- More precise cap lookup lemmas
- e.g. handle_recv, *as assumed*:
  - **does not modify** most ks_of fields

```
ks_of p s ≡ (|
    thread_cnode p s,
    bound_notification p s,          } cap state, thread init!
    mapped_writable p s,
    not_writable_others p s,         } memory mappings!
    recv_oracle s (bound_notification p s) (thread_cnode p s),
    λch. ntfn_status s (thread_cnode p s) ch
|)
```

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to
cyberagentur
FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"
✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

Blocking
✓ **Unblocking**

seL4 OS Microkernel Abstract Model

What we've proved:

- More precise cap lookup lemmas
- e.g. handle_recv, *as assumed*:
  - **does not modify** most ks_of fields
  - returns recv_oracle-"predicted" badge in "Nonblocking" case
  - updates ntfn_status in Blocking case

```
ks_of p s ≡ (
    thread_cnode p s,
    bound_notification p s,          } cap state, thread init!
    mapped_writable p s,
    not_writable_others p s,         } memory mappings!
    recv_oracle s (bound_notification p s) (thread_cnode p s),
    λch. ntfn_status s (thread_cnode p s) ch
)
```

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to

cyberagentur

FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"
✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

Blocking
✓ **Unblocking**

seL4 OS Microkernel Abstract Model

What we've proved:

- More precise cap lookup lemmas
- e.g. handle_recv, *as assumed*:
  - **does not modify** most ks_of fields
  - returns recv_oracle-"predicted" badge in "Nonblocking" case
  - updates ntfn_status in Blocking case
- Lifting helpers for state projections

ks_of p s ≡ ⦇
thread_cnode p s,
bound_notification p s, } **cap state, thread init!**
mapped_writable p s,
not_writable_others p s, } **memory mappings!**
recv_oracle s (bound_notification p s) (thread_cnode p s),
λch. ntfn_status s (thread_cnode p s) ch
⦈

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to

cyberagentur

FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"

✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

✓ Blocking

✓ **Unblocking**

seL4 OS Microkernel Abstract Model

- Proving & generalising their composition

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to

cyberagentur

FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"

✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

✓ **Blocking** **Unblocking**

seL4 OS Microkernel Abstract Model

- Proving & generalising their composition

{{ λs. P s }}
seL4_SomeCall
{{ λs. Q s }}

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to

cyberagentur

FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"
✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

✓ Blocking
Unblocking

seL4 OS Microkernel Abstract Model

- Proving & generalising their composition

{{ λs. P s }}
seL4_SomeCall
{{ λs. Q s }}

Blocking A

Unblocking B

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to

cyberagentur

FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"

✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

Blocking

✓ **Unblocking**

seL4 OS Microkernel Abstract Model

- Proving & generalising their composition

{{ λs. P s }}
seL4_SomeCall
{{ λs. Q s }}

Blocking A

C, D, E … X, Y, Z

Irrelevant

Unblocking B

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to

cyberagentur

FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"
✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

~~Blocking~~
✓ **Unblocking**

seL4 OS Microkernel Abstract Model

- Proving & generalising their composition

{{ λs. P s }}
seL4_SomeCall
{{ λs. Q s }}

→ {{ λs. P (ks_of A s) }}
Blocking A

C, D, E … X, Y, Z
Irrelevant

Unblocking B

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to

cyberagentur

FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"

✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

~~Blocking~~

✓ **Unblocking**

seL4 OS Microkernel Abstract Model

- Proving & generalising their composition

{{ λs. P s }}
seL4_SomeCall
{{ λs. Q s }}

→

{{ λs. P (ks_of A s) }}
Blocking A
{{ λs. R (ks_of A s) &&
P' (ks_of B s) }}

C, D, E … X, Y, Z
Irrelevant

Unblocking B

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to

cyberagentur

FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"

✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

~~Blocking~~

✓ **Unblocking**

seL4 OS Microkernel Abstract Model

- Proving & generalising their composition

{{ λs. P s }}
seL4_SomeCall
{{ λs. Q s }}

→ {{ λs. P (ks_of A s) }}
Blocking A
{{ λs. R (ks_of A s) &&
P' (ks_of B s) }}

C, D, E … X, Y, Z
Irrelevant

{{ λs. R (ks_of A s) && P' (ks_of B s) }}
Unblocking B

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to

cyberagentur

FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"

✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

~~Blocking~~

✓ **Unblocking**

seL4 OS Microkernel Abstract Model

- Proving & generalising their composition

{{ λs. P s }}
seL4_SomeCall
{{ λs. Q s }}

→ {{ λs. P (ks_of A s) }}
Blocking A
{{ λs. R (ks_of A s) &&
P' (ks_of B s) }}

{{ λs. R (ks_of A s) && P' (ks_of B s) }}
Unblocking B

Note: If we can't prove these irrelevant for wellformed Microkit systems, then the seL4 kernel is incorrect!

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to

cyberagentur

FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"
✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

Blocking
✓ **Unblocking**

seL4 OS Microkernel Abstract Model

- Proving & generalising their composition

{{ λs. P s }}
seL4_SomeCall
{{ λs. Q s }}

→ {{ λs. P (ks_of A s) }}
Blocking A
{{ λs. R (ks_of A s) &&
P' (ks_of B s) }}

Note: If we can't prove these irrelevant for wellformed Microkit systems, then the seL4 kernel is incorrect!

{{ λs. R (ks_of A s) && P' (ks_of B s) }}
Unblocking B
{{ λs. Q (ks_of A s) && Q' (ks_of B s) }}

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to
cyberagentur
FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"
✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

~~Blocking~~
✓ **Unblocking**

seL4 OS Microkernel Abstract Model

- Proving & generalising their composition

Note: If we can't prove these irrelevant for wellformed Microkit systems, then the seL4 kernel is incorrect!

{{ λs. P s }} → {{ λs. P (ks_of A s) }}

Blocking A

seL4_SomeCall {{ λs. R (ks_of A s) && P' (ks_of B s) }}

C, D, E … X, Y, Z
Irrelevant

{{ λs. R (ks_of A s) && P' (ks_of B s) }}

Unblocking B

{{ λs. Q s }} ← {{ λs. Q (ks_of A s) && Q' (ks_of B s) }}

What we're doing

- Defined state projections
- Verifying Hoare triples on calls

Read:
ks_of = "kernel state of"

Thanks to

cyberagentur

FORESIGHT INSTITUTE

Process A

{{ λs. P s }} seL4_Recv {{ λs. Q s }}

Process B

{{ λs. P' s }} seL4_Signal {{ λs. Q' s }}

seL4 Microkit Library

syscall (i.e. *kernel–userland*) interface

{{ λs. P (ks_of A s) }}
handle_recv
{{ λs. Q (ks_of A s) }}

✓ "Nonblocking"

✓ **Blocking**

{{ λs. P' (ks_of B s) }}
send_signal
{{ λs. Q' (ks_of B s) }}

Blocking

✓ **Unblocking**

seL4 OS Microkernel Abstract Model

- Proving & generalising their composition

{{ λs. P s }} → {{ λs. P (ks_of A s) }}

Blocking A

seL4_SomeCall {{ λs. R (ks_of A s) && P' (ks_of B s) }}

C, D, E … X, Y, Z
Irrelevant

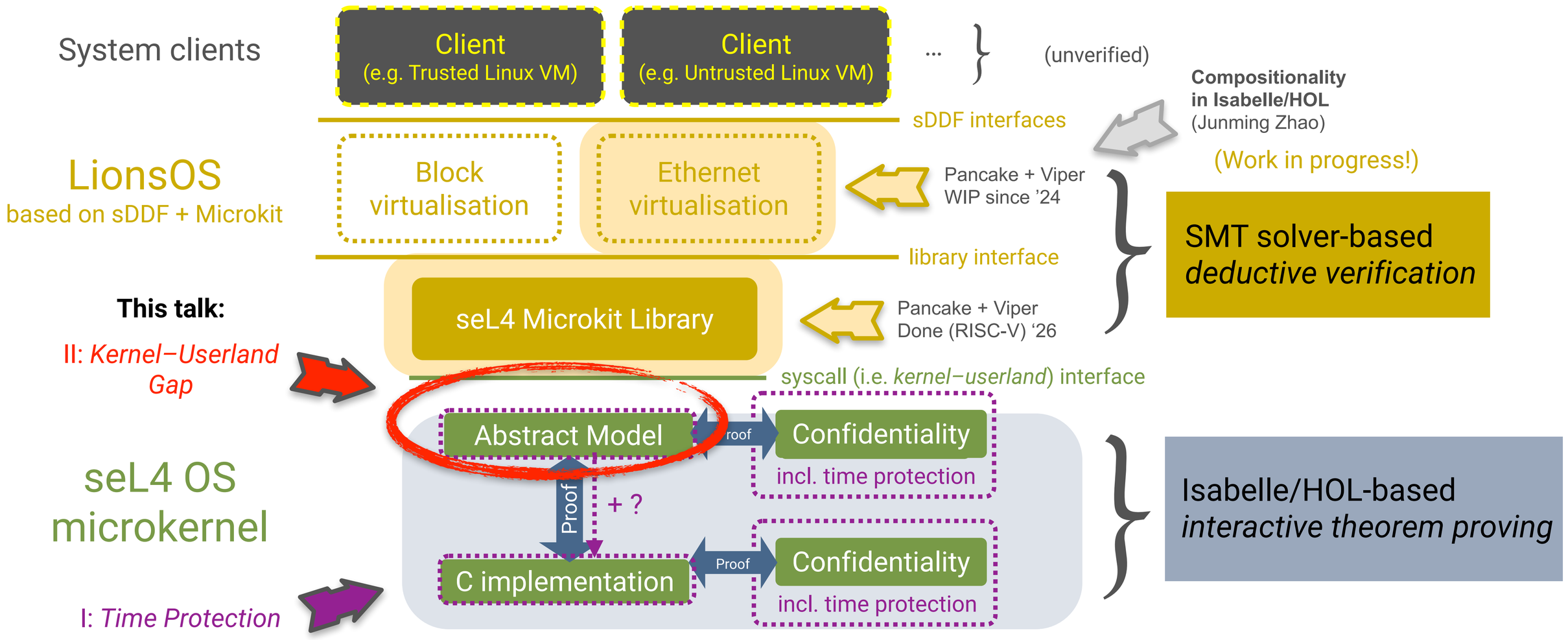
{{ λs. R (ks_of A s) && P' (ks_of B s) }}

Unblocking B

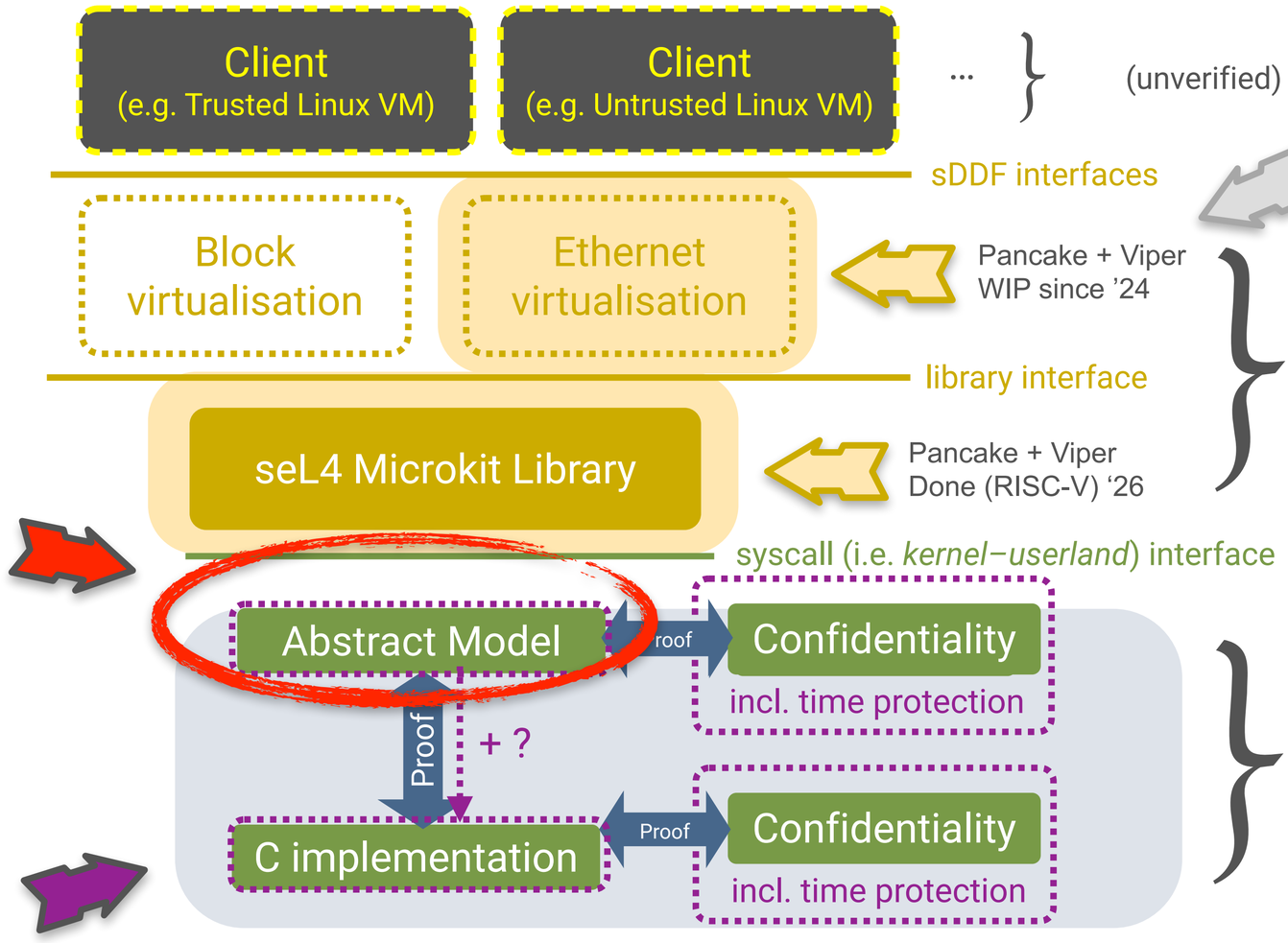
{{ λs. Q s }} ← {{ λs. Q (ks_of A s) && Q' (ks_of B s) }}

Note: If we can't prove these irrelevant for wellformed Microkit systems, then the seL4 kernel is incorrect!

Verifying seL4-based OSes: Status



Verifying seL4-based OSes: Status



Compositionality in Isabelle/HOL
(Junming Zhao)

(Work in progress!)

SMT solver-based
deductive verification

Thank you!
Q & A

Isabelle/HOL-based
interactive theorem proving