School of Computer Science & Engineering

Trustworthy Systems Group

Where are we with Time Protection?

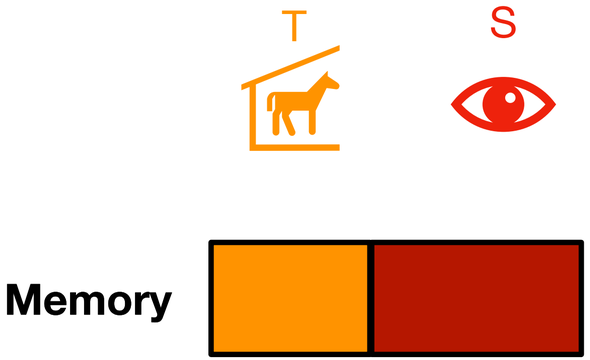
Verification and cross-domain communication update

Dr Rob Sison

Senior Research Associate, UNSW Sydney

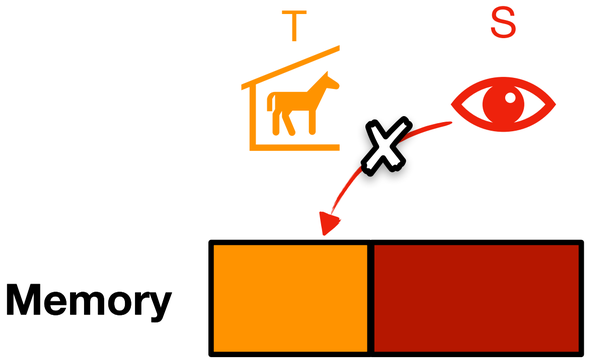
r.sison@unsw.edu.au

What is Time Protection?



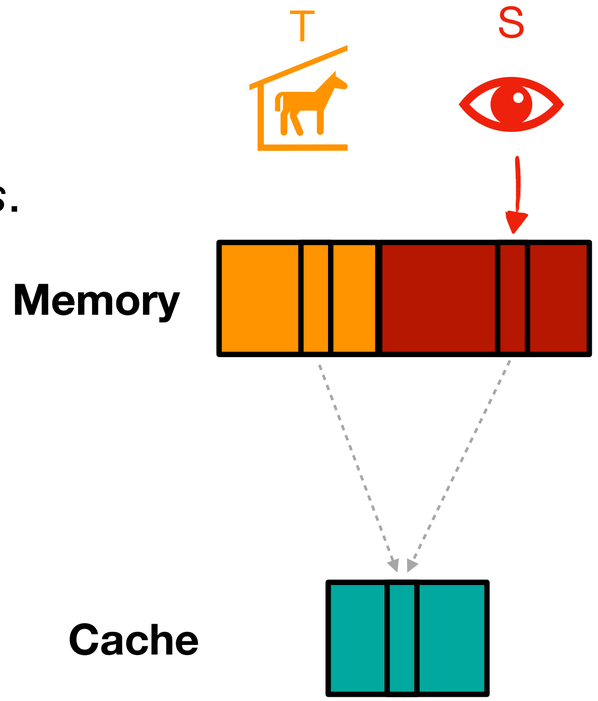
What is Time Protection?

- OSes typically implement *memory protection.*



What is Time Protection?

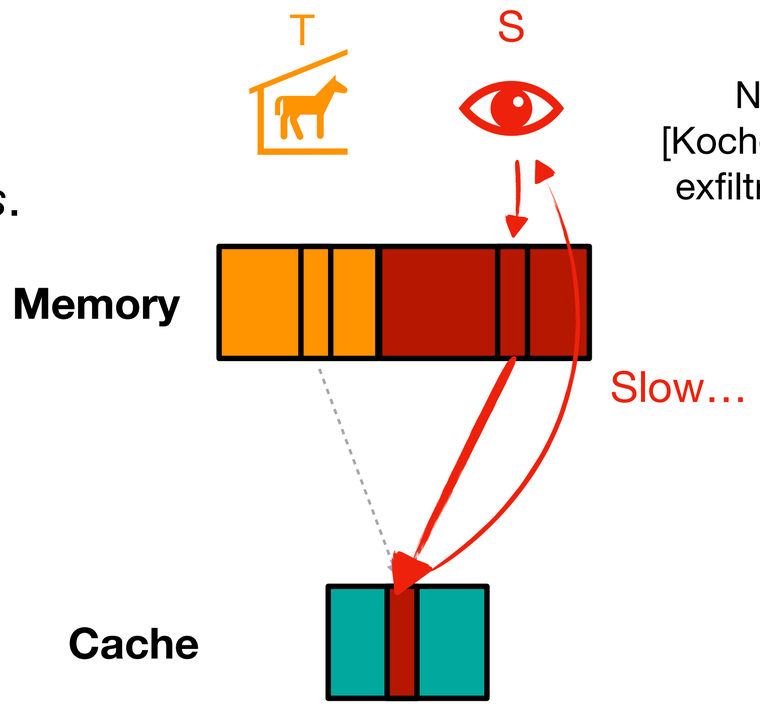
- OSes typically implement *memory protection.*
- But: Just accessing memory changes HW's microarchitectural state — this creates *timing channels*.



NB: Spectre [Kocher et al. 2017]'s exfiltration method!

What is Time Protection?

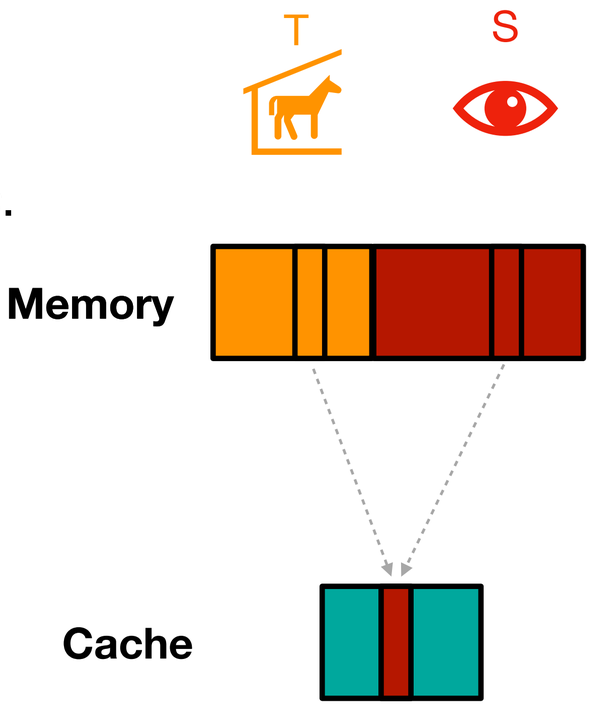
- OSes typically implement *memory protection.*
- But: Just accessing memory changes HW's microarchitectural state — this creates *timing channels*.



NB: Spectre [Kocher et al. 2017]'s exfiltration method!

What is Time Protection?

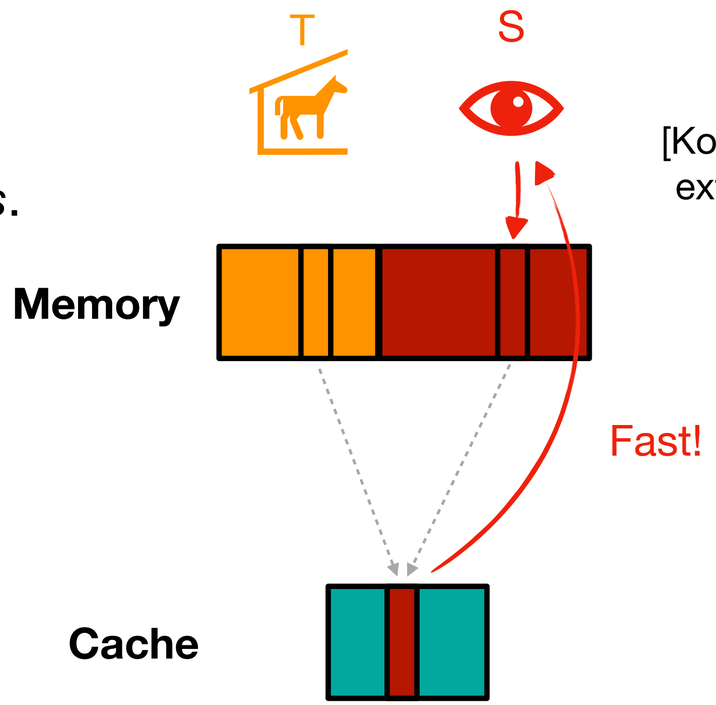
- OSes typically implement *memory protection.*
- But: Just accessing memory changes HW's microarchitectural state — this creates *timing channels*.



NB: Spectre [Kocher et al. 2017]'s exfiltration method!

What is Time Protection?

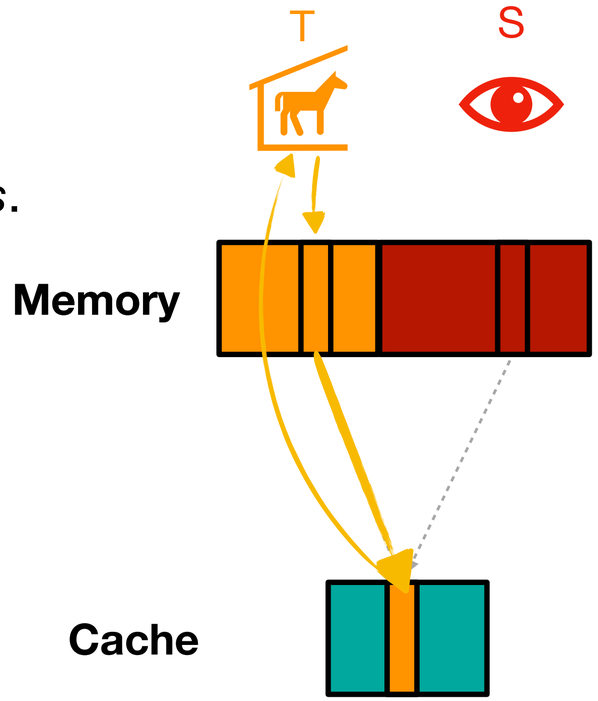
- OSes typically implement *memory protection.*
- But: Just accessing memory changes HW's microarchitectural state — this creates *timing channels*.



NB: Spectre [Kocher et al. 2017]'s exfiltration method!

What is Time Protection?

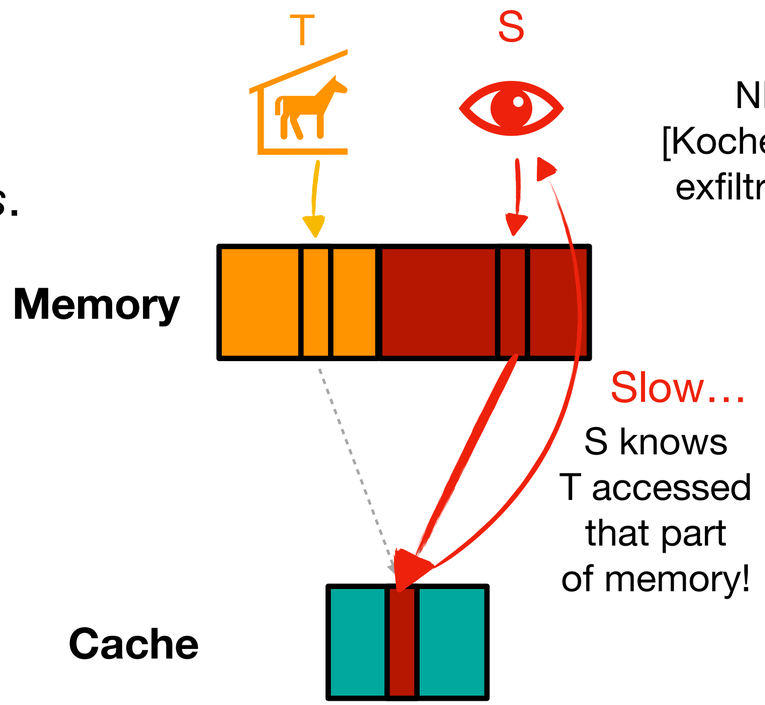
- OSes typically implement *memory protection.*
- But: Just accessing memory changes HW's microarchitectural state — this creates *timing channels*.



NB: Spectre [Kocher et al. 2017]'s exfiltration method!

What is Time Protection?

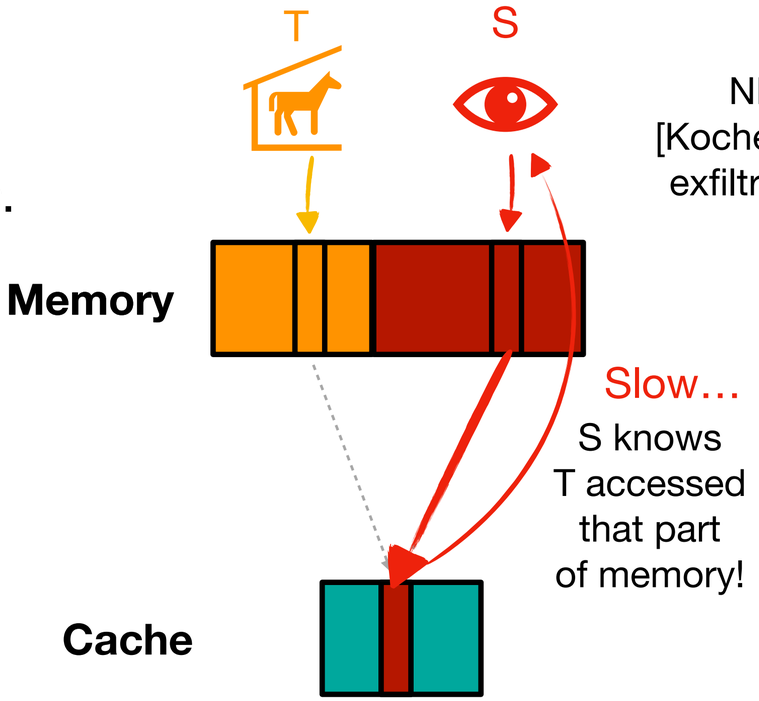
- OSes typically implement *memory protection.*
- But: Just accessing memory changes HW's microarchitectural state — this creates *timing channels*.



NB: Spectre [Kocher et al. 2017]'s exfiltration method!

What is Time Protection?

- OSes typically implement *memory protection.*
- But: Just accessing memory changes HW's microarchitectural state — this creates *timing channels*.
- To prevent timing channels, OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]



NB: Spectre [Kocher et al. 2017]'s exfiltration method!

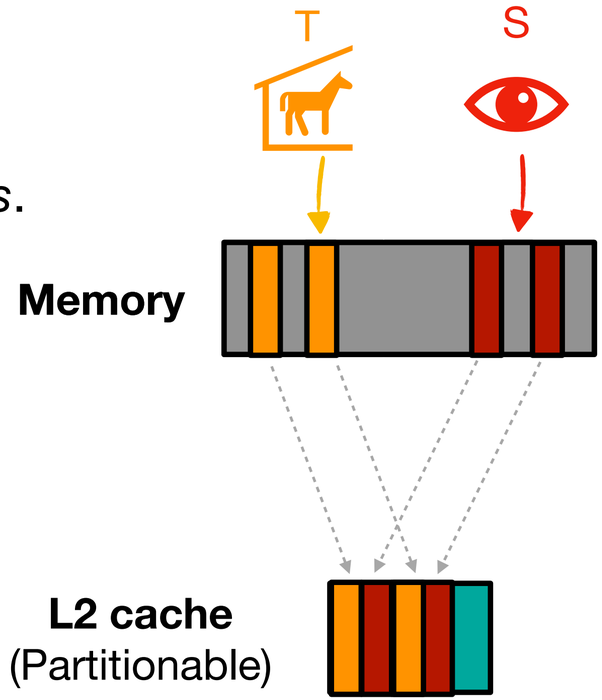
Qian Ge

Gernot Heiser

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

What is Time Protection?

- OSes typically implement *memory protection.*
- But: Just accessing memory changes HW's microarchitectural state — this creates *timing channels*.
- To prevent timing channels, OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches



NB: Spectre [Kocher et al. 2017]'s exfiltration method!

Qian Ge

Gernot Heiser

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

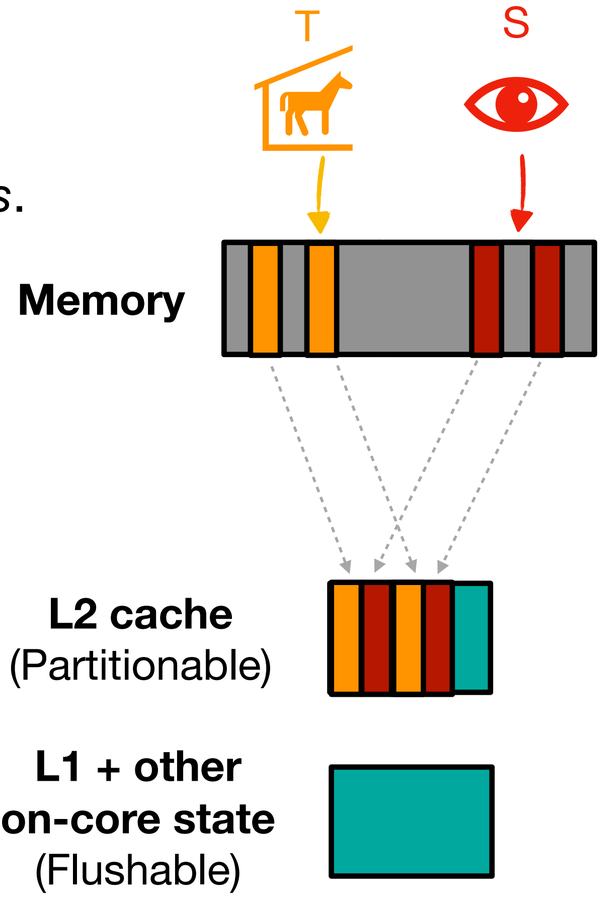
What is Time Protection?

- OSes typically implement *memory protection.*
- But: Just accessing memory changes HW's microarchitectural state — this creates *timing channels*.
- To prevent 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 domain switch

Qian Ge

Gernot Heiser

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



NB: Spectre [Kocher et al. 2017]'s exfiltration method!

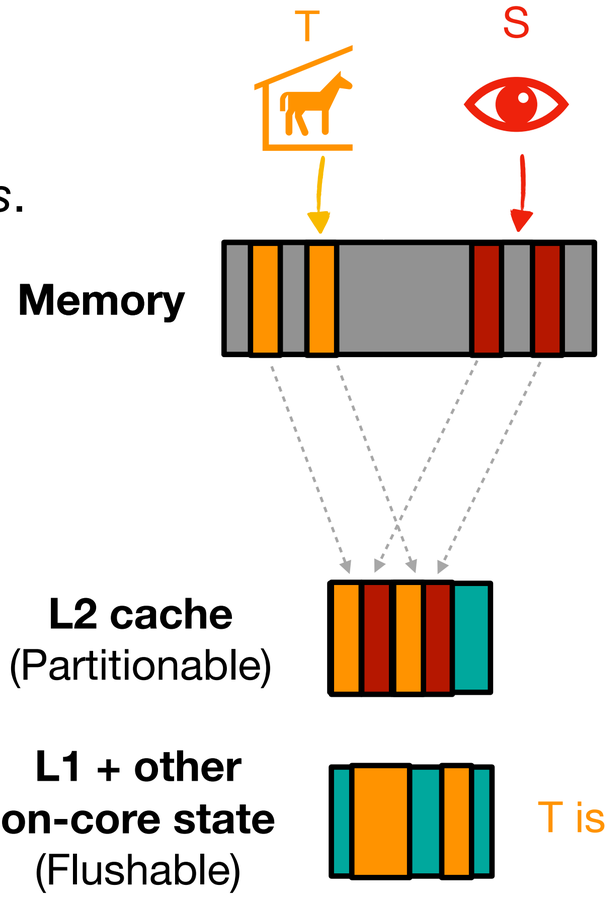
What is Time Protection?

- OSes typically implement *memory protection.*
- But: Just accessing memory changes HW's microarchitectural state — this creates *timing channels*.
- To prevent 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 domain switch

Qian Ge

Gernot Heiser

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



NB: Spectre [Kocher et al. 2017]'s exfiltration method!

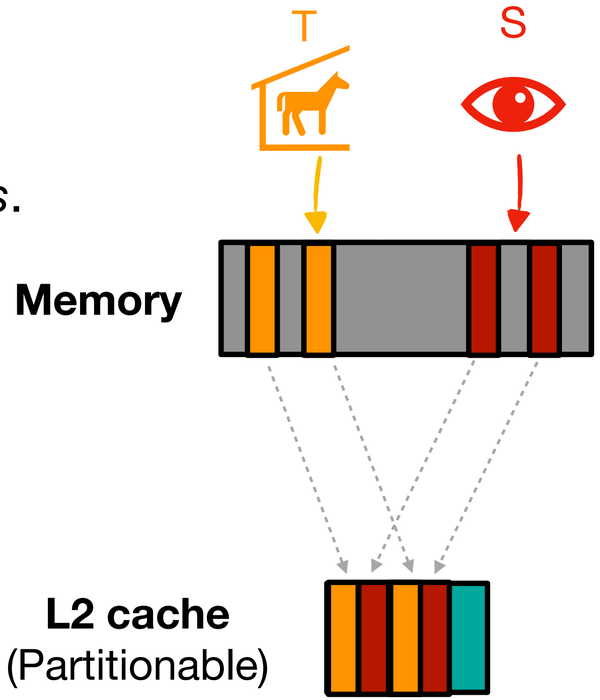
What is Time Protection?

- OSes typically implement *memory protection.*
- But: Just accessing memory changes HW's microarchitectural state — this creates *timing channels*.
- To prevent 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 domain switch

Qian Ge

Gernot Heiser

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



NB: Spectre [Kocher et al. 2017]'s exfiltration method!

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

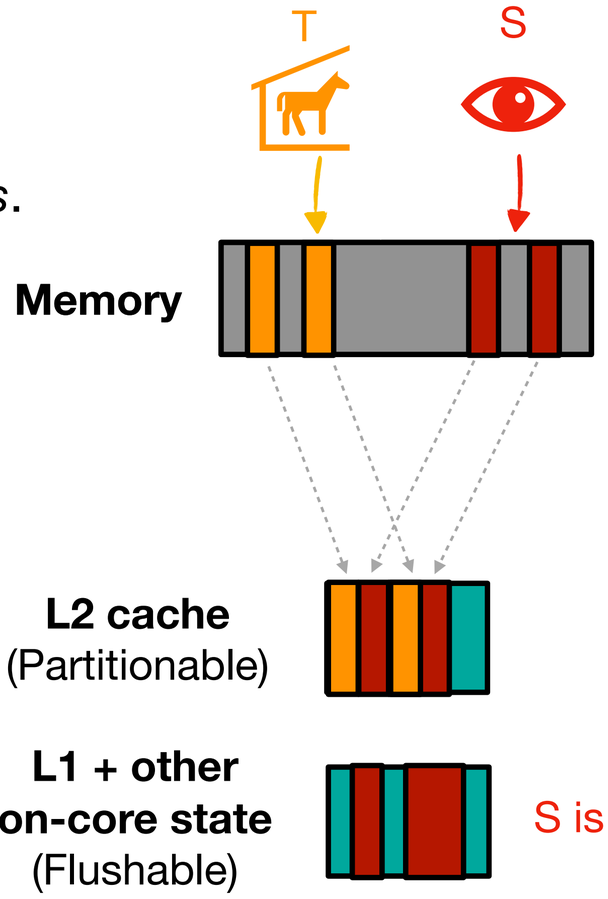
What is Time Protection?

- OSes typically implement *memory protection.*
- But: Just accessing memory changes HW's microarchitectural state — this creates *timing channels*.
- To prevent 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 domain switch

Qian Ge

Gernot Heiser

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



NB: Spectre [Kocher et al. 2017]'s exfiltration method!

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

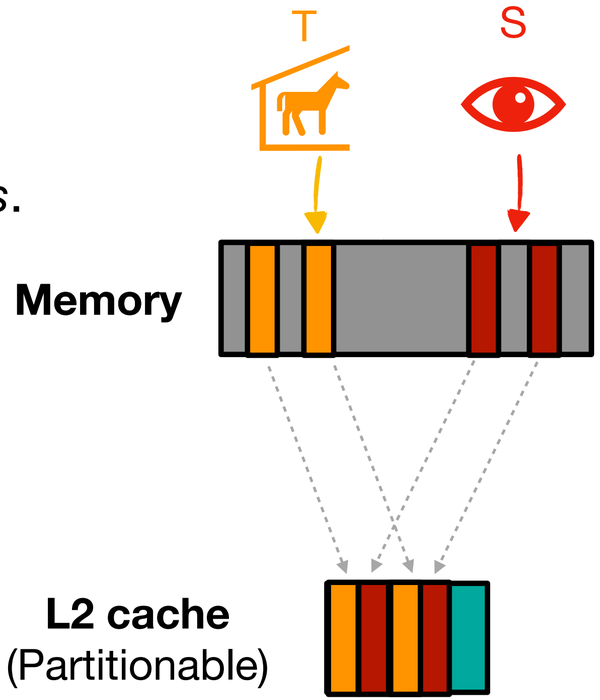
What is Time Protection?

- OSes typically implement *memory protection.*
- But: Just accessing memory changes HW's microarchitectural state — this creates *timing channels*.
- To prevent 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 domain switch

Qian Ge

Gernot Heiser

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



NB: Spectre [Kocher et al. 2017]'s exfiltration method!

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

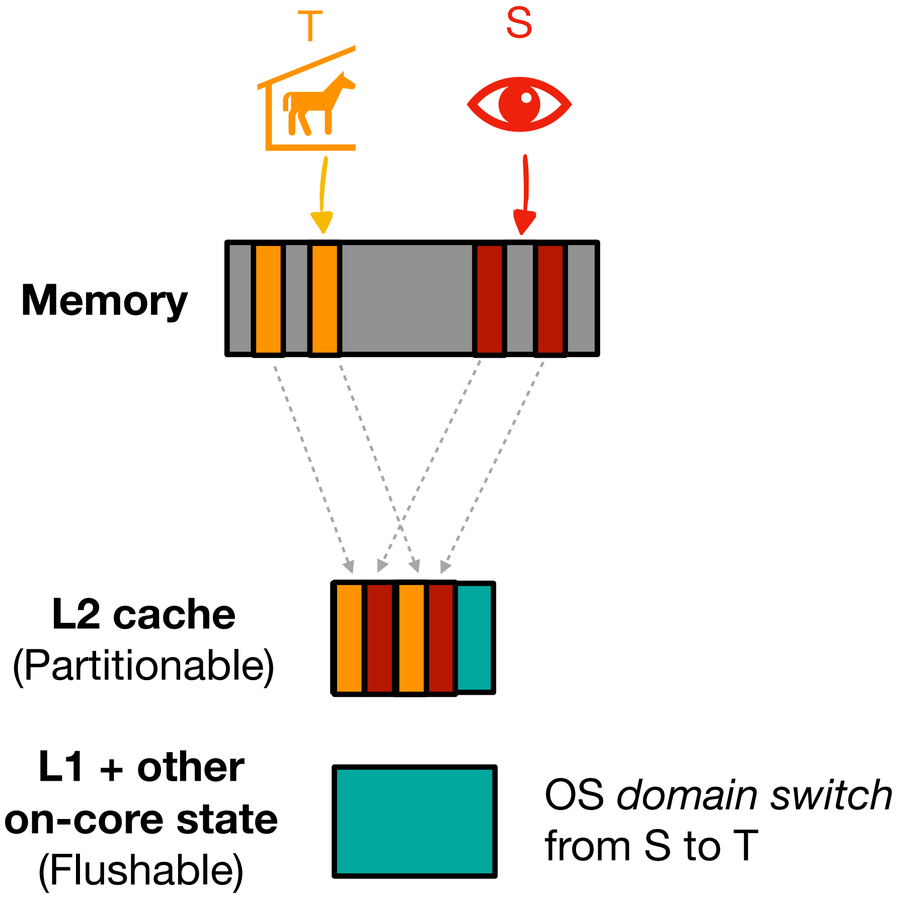
The seL4 kernel and Time Protection

- To prevent 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 domain 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.

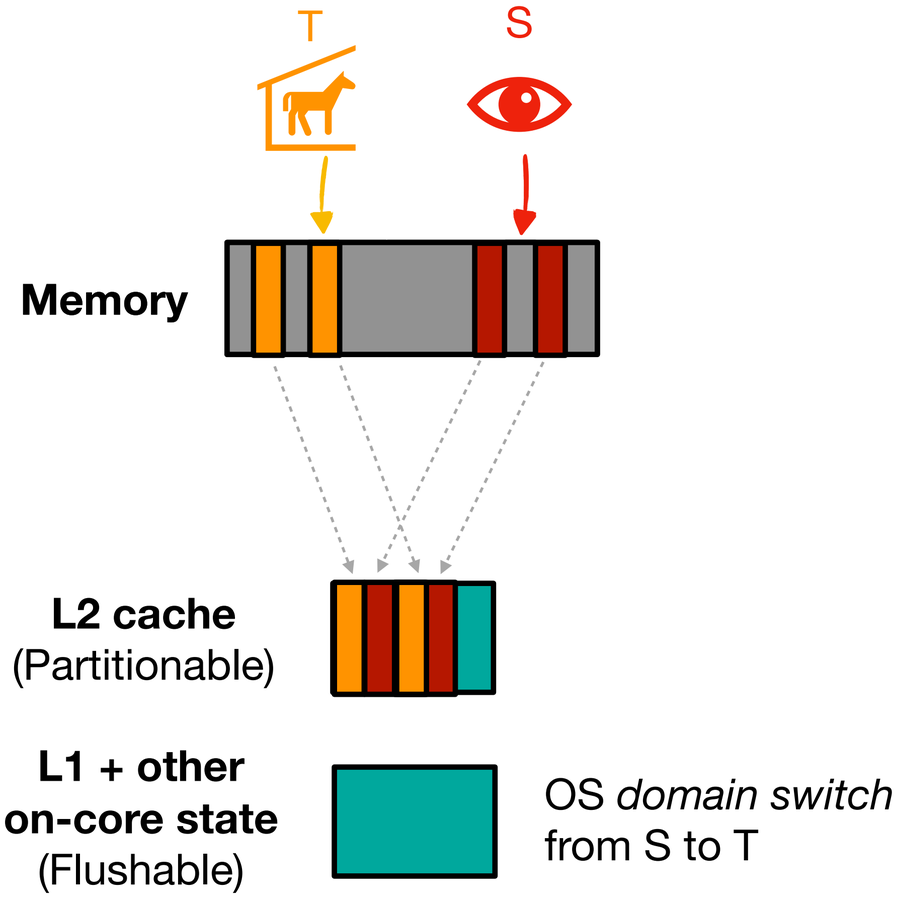
The seL4 kernel and Time Protection

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

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

- To prevent 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 domain 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])


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

- To prevent 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 domain 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])

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


Scott Buckley


Rob Sison


Nils Wistoff


Curtis Millar


Toby Murray


Gerwin Klein

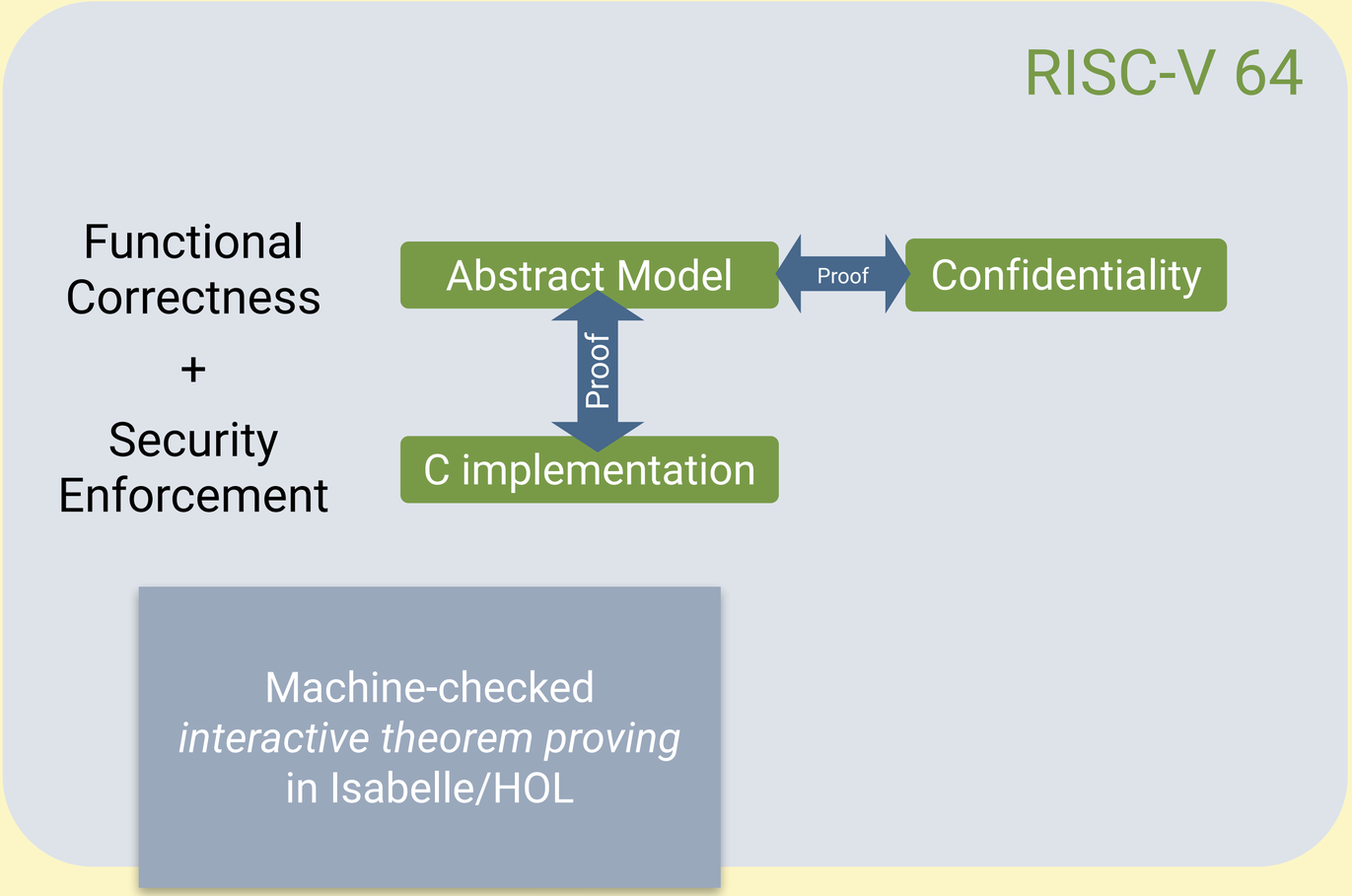

Gernot Heiser

Thanks to funding from
Australian Research Council

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

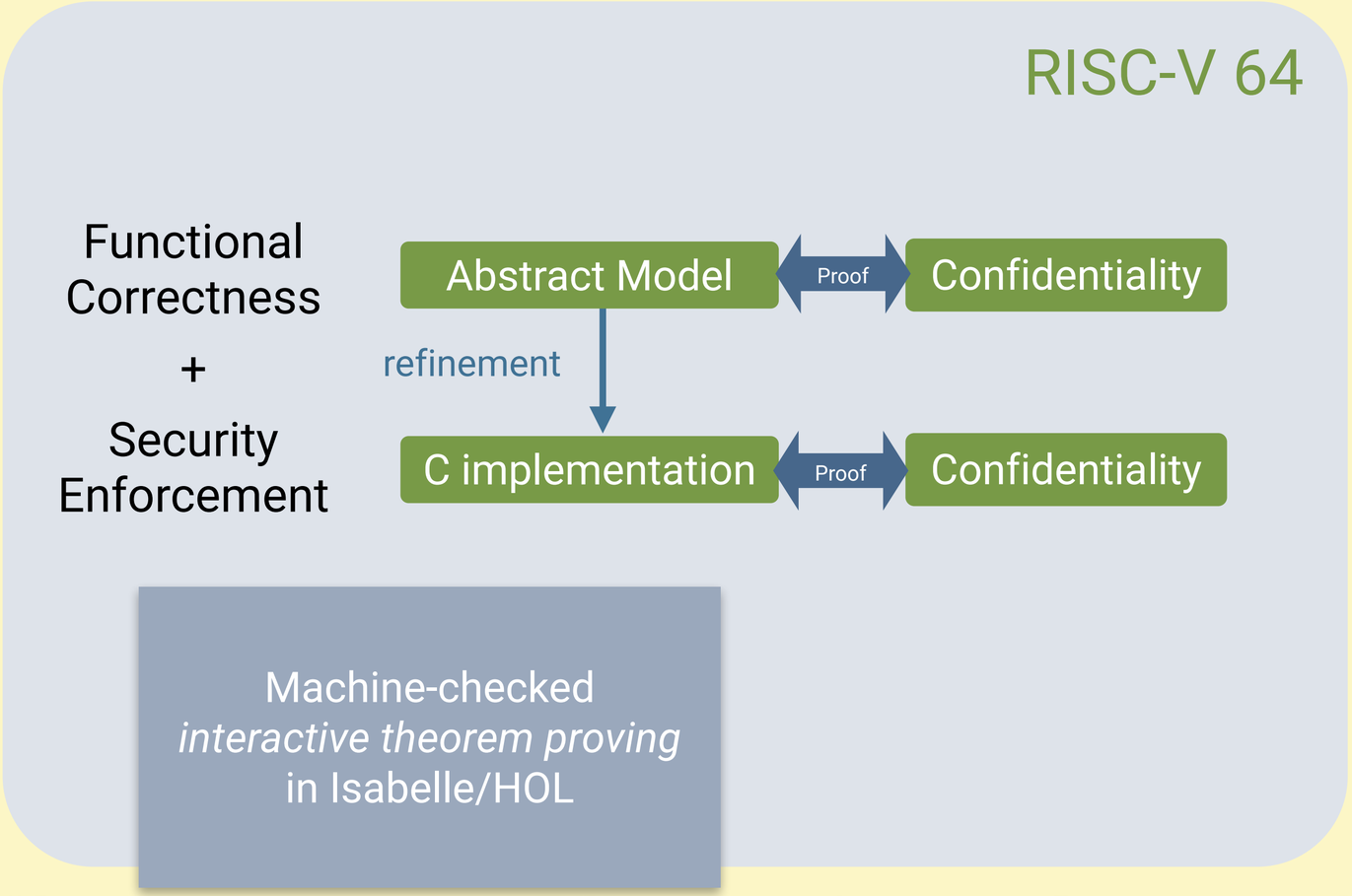
Time protection verification for seL4

The plan 2 years ago… *(presented at seL4 Summit ’24)*



Time protection verification for seL4

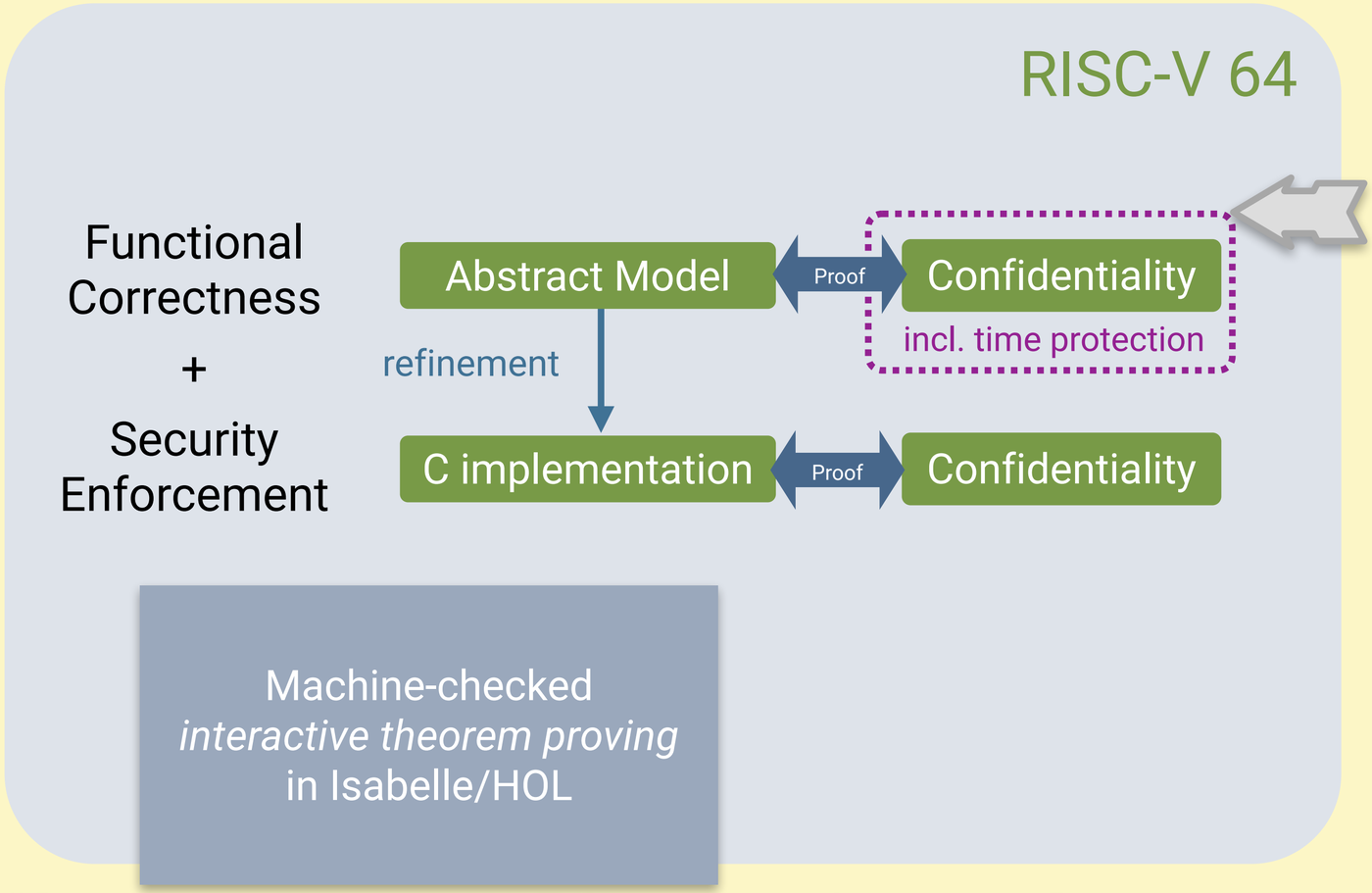
The plan 2 years ago… *(presented at seL4 Summit ’24)*



Time protection verification for seL4

The plan 2 years ago… *(presented at seL4 Summit ’24)*

Thanks to funding from
Australian Research Council

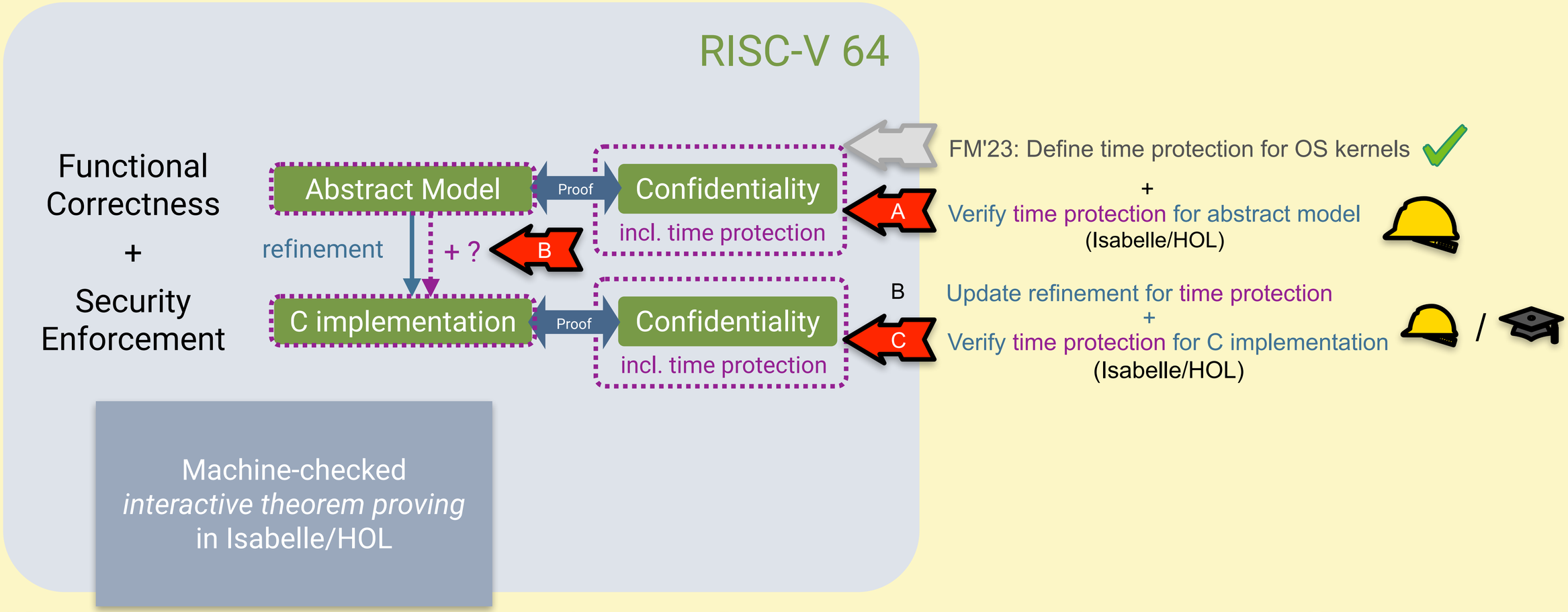


FM'23: Define time protection for OS kernels ✓

Time protection verification for seL4

The plan 2 years ago… *(presented at seL4 Summit ’24)*

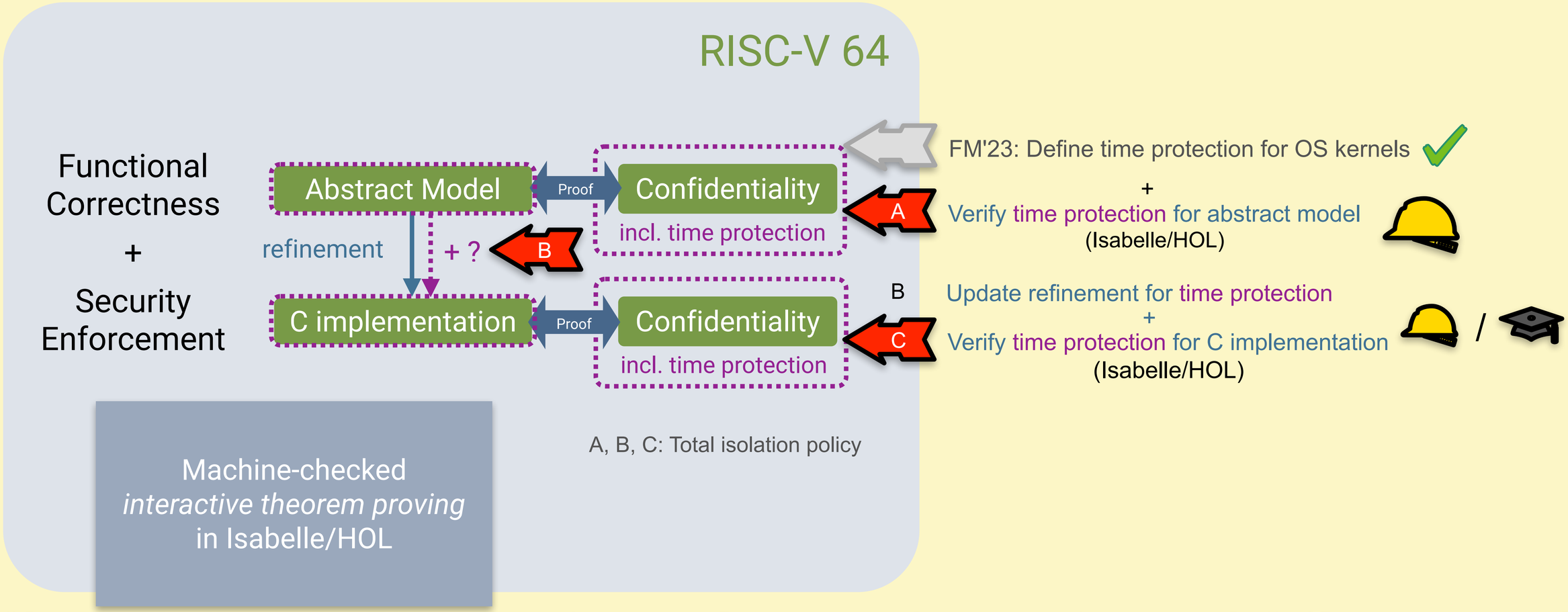
Thanks to funding from
Australian Research Council



Time protection verification for seL4

The plan 2 years ago… *(presented at seL4 Summit ’24)*

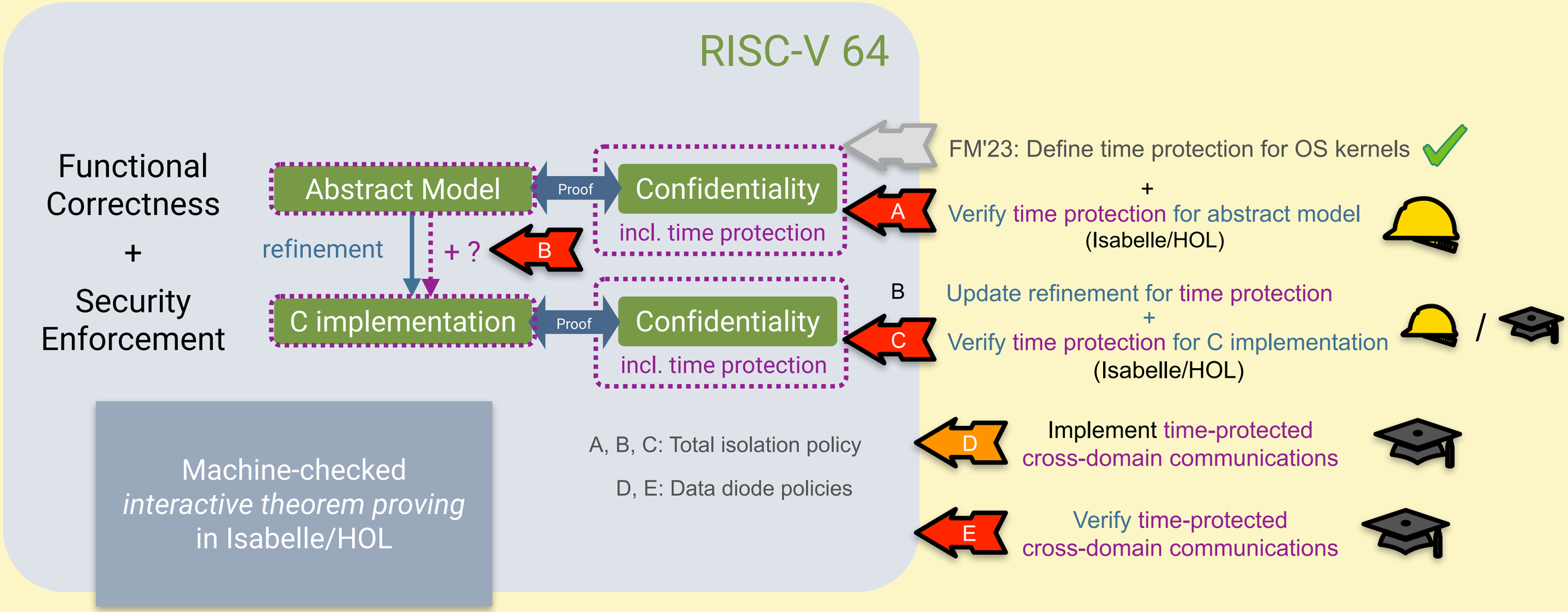
Thanks to funding from
Australian Research Council



Time protection verification for seL4

The plan 2 years ago… *(presented at seL4 Summit ’24)*

Thanks to funding from
Australian Research Council

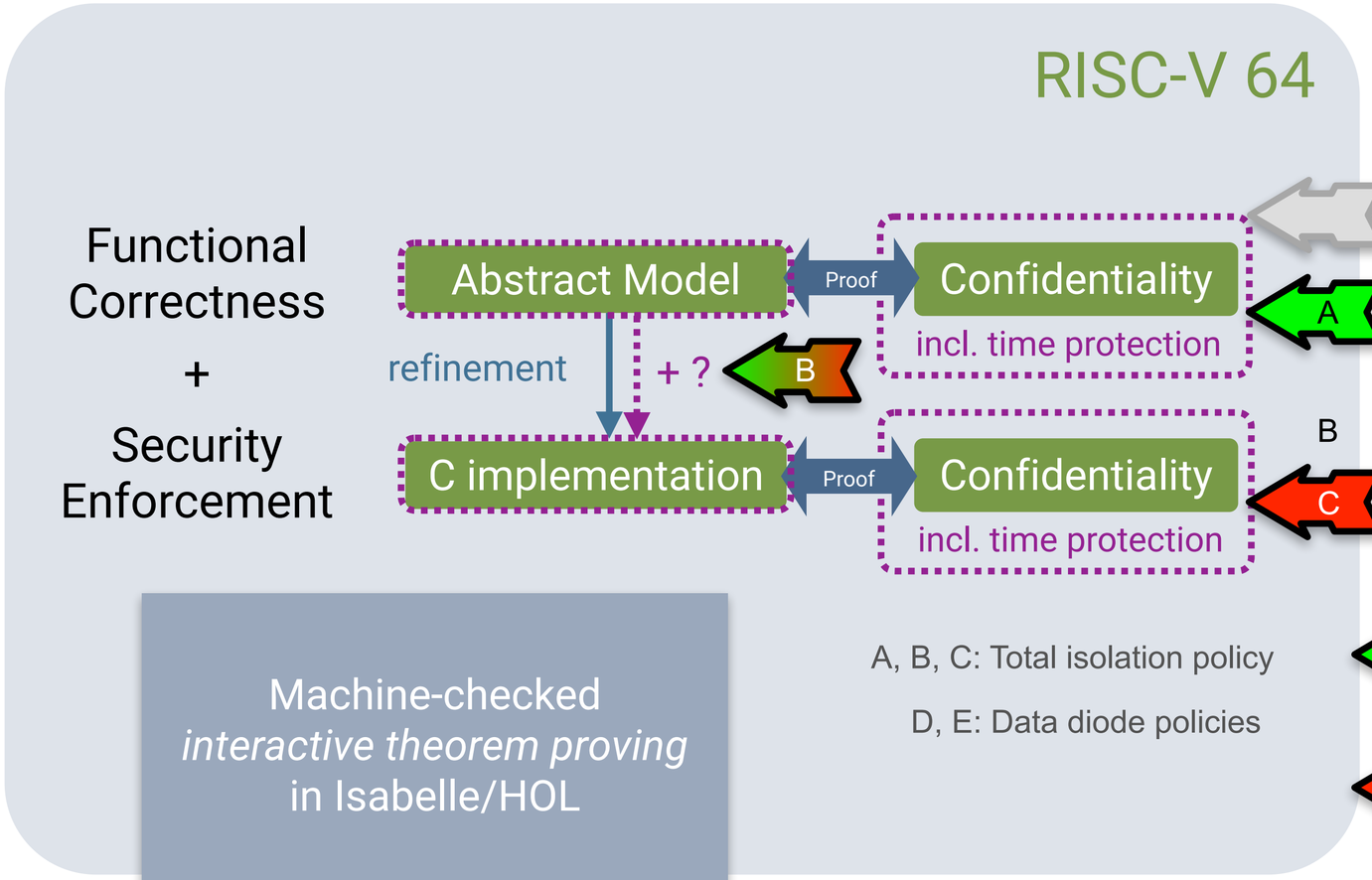


Time protection verification for seL4

The status today:

Thanks to funding from

cyberagentur



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)

D — Implement time-protected cross-domain communications ✓

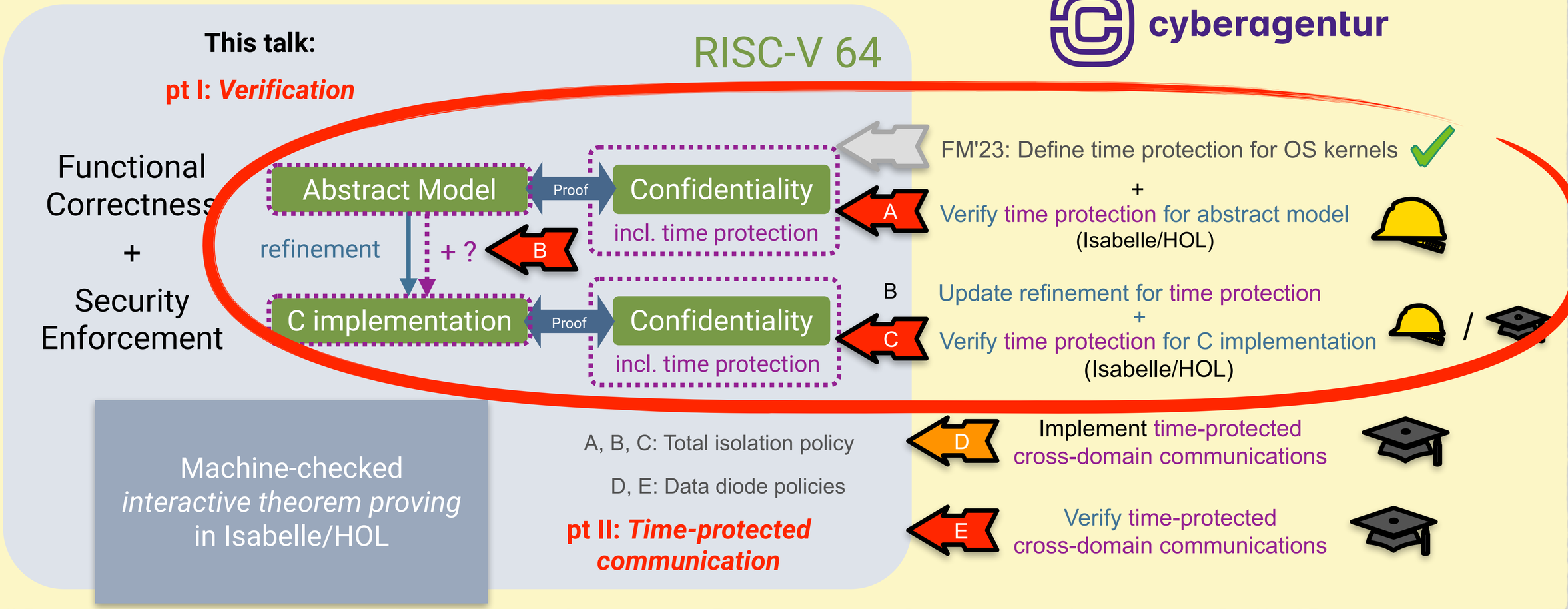
E — Verify time-protected cross-domain communications

Time protection verification for seL4

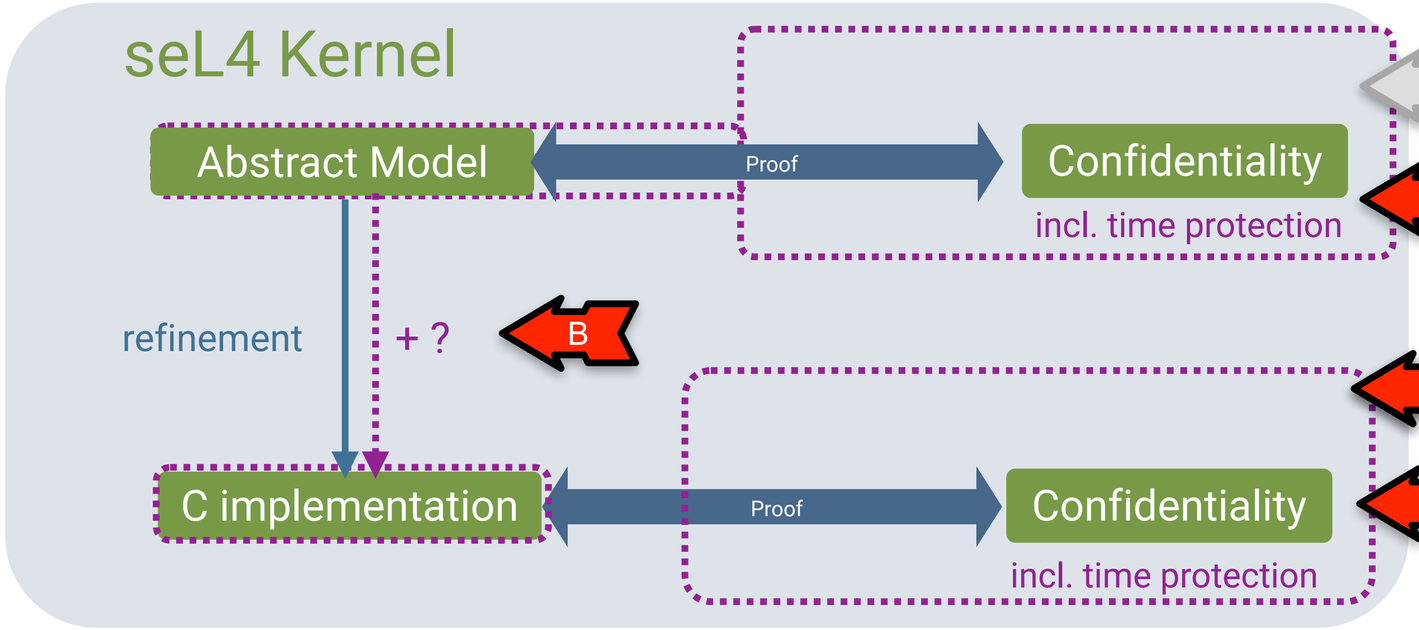
Since the plan 2 years ago…

Thanks to funding from

cyberagentur



Originally planned approach… (c. 2023-2024)



FM'23: Define time protection for OS kernels ✓

+

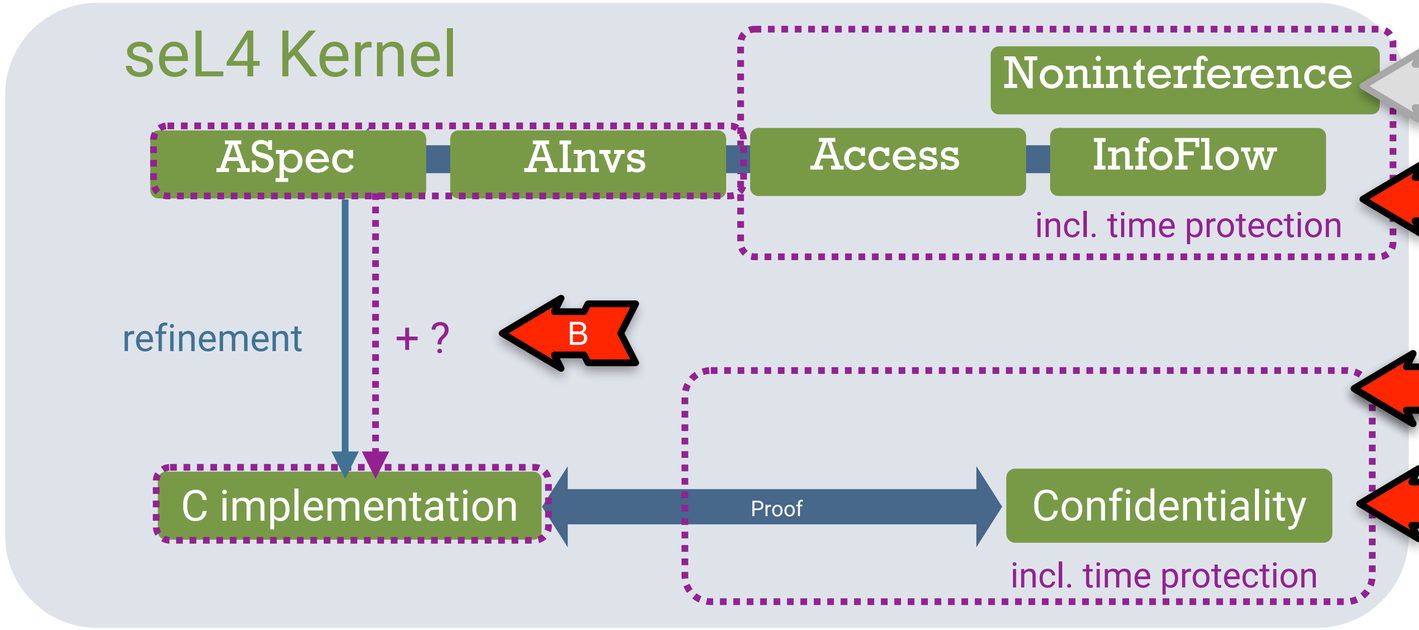
A Verify time protection for abstract model (Isabelle/HOL)

B Update refinement for time protection

+

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

Originally planned approach… (c. 2023-2024)



FM'23: Define time protection for OS kernels ✓

+

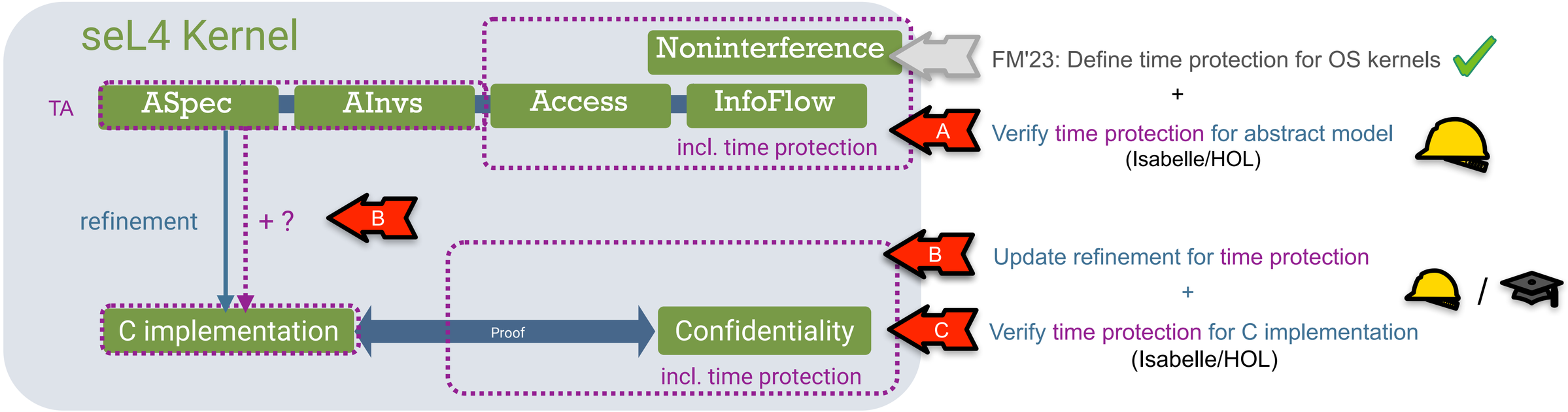
A Verify time protection for abstract model (Isabelle/HOL)

B Update refinement for time protection

+

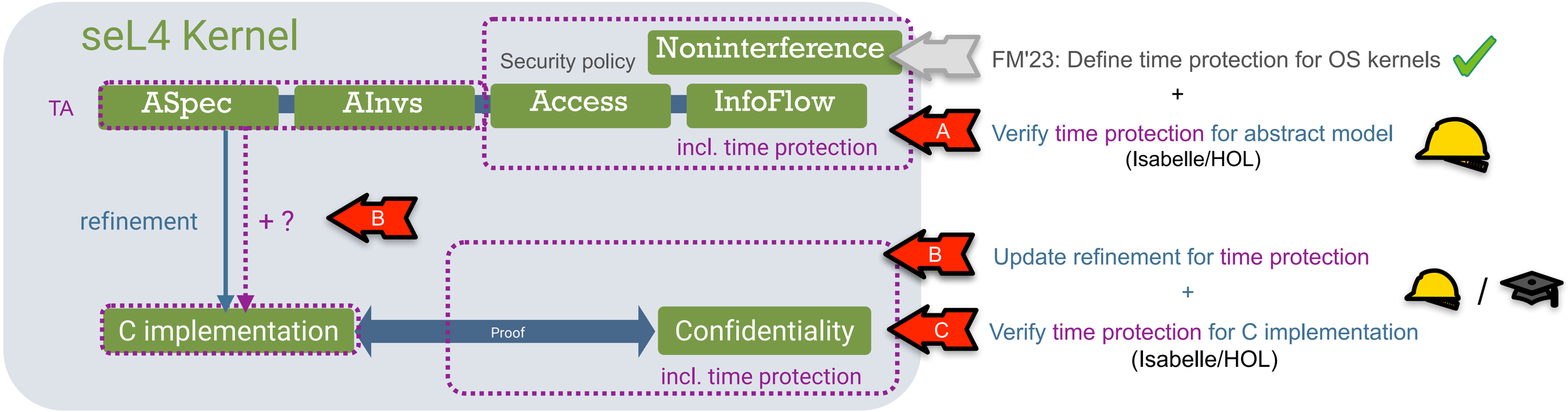
C Verify time protection for C implementation (Isabelle/HOL)

Originally planned approach… (c. 2023-2024)



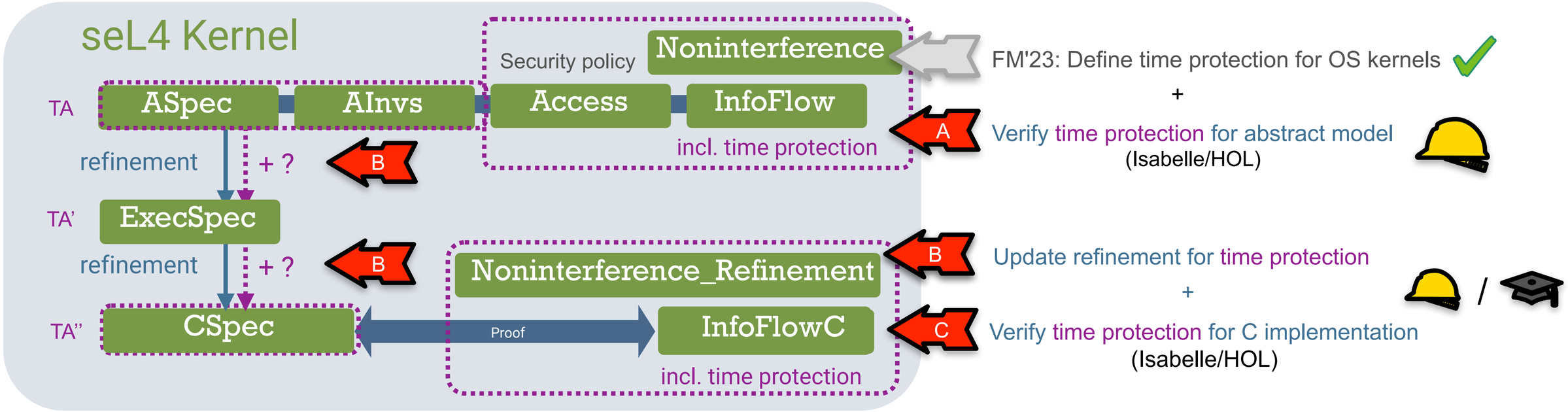
- Abstract model (**ASpec**) tracks *dynamic* touched addresses (TA) set

Originally planned approach… (c. 2023-2024)



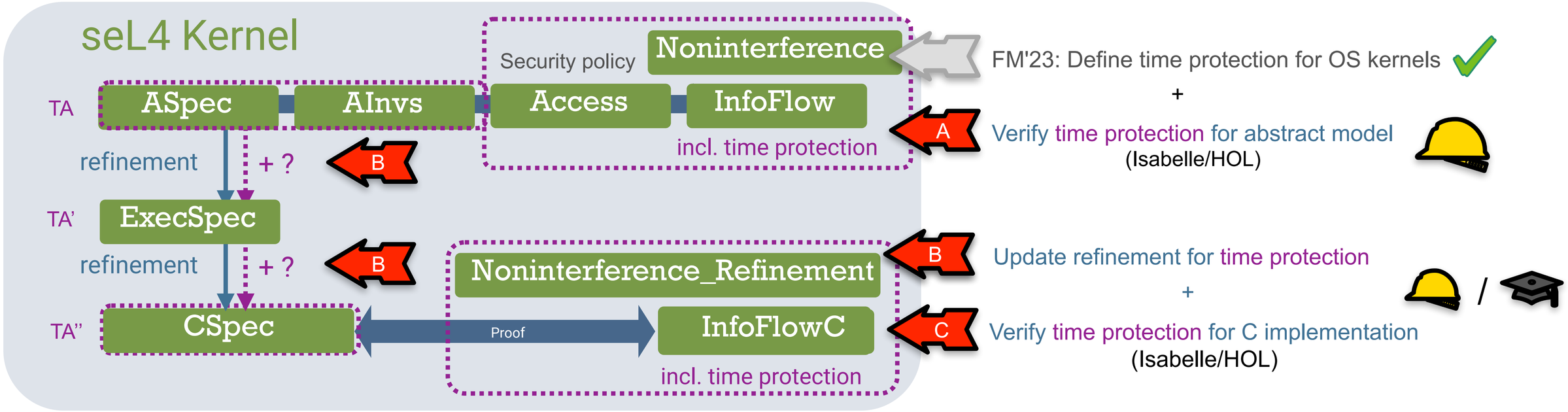
- Abstract model (**ASpec**) tracks *dynamic* touched addresses (TA) set
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (as per *security policy*)

Originally planned approach… (c. 2023-2024)



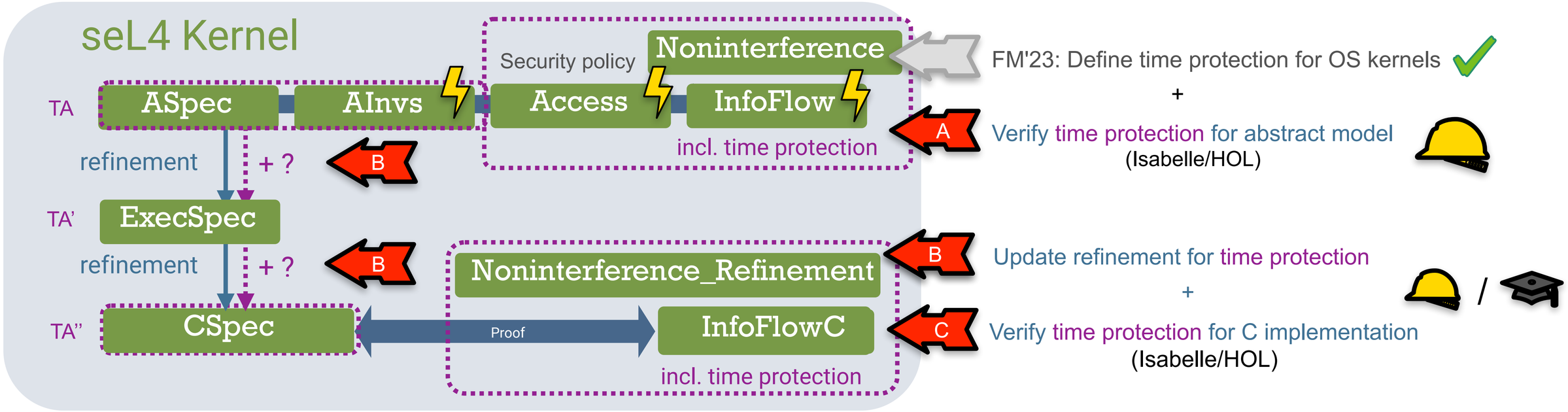
- Abstract model (**ASpec**) tracks *dynamic* touched addresses (TA) set
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (as per *security policy*)
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)

Originally planned approach… (c. 2023-2024)



- Abstract model (**ASpec**) tracks *dynamic* touched addresses (TA) set
  Many existing proofs rely on *nothing changing*
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (as per *security policy*)
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)

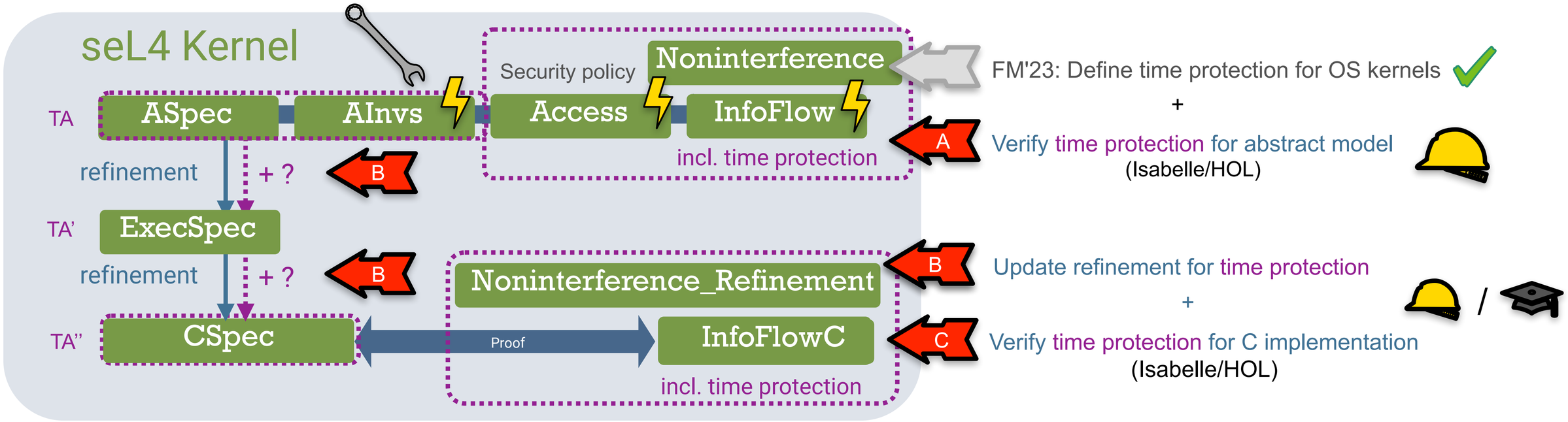
Originally planned approach… (c. 2023-2024)



- Abstract model (**ASpec**) tracks *dynamic* touched addresses (TA) set
  Many existing proofs rely on *nothing changing*
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (as per *security policy*)
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)

This changes
> *125* times

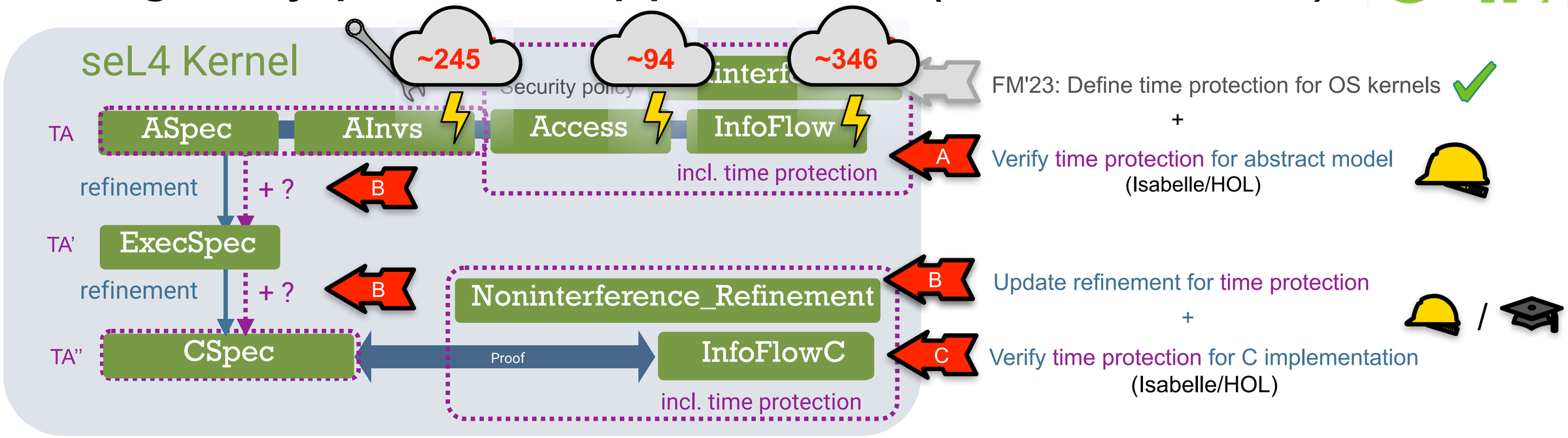
Originally planned approach… (c. 2023-2024)



- Abstract model (**ASpec**) tracks *dynamic* touched addresses (TA) set
  Many existing proofs rely on *nothing changing*
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (as per *security policy*)
  THUS, Too many breakages in **AInvs**, **Access**, **InfoFlow** & beyond…
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)

This changes
> *125* times

Originally planned approach… (c. 2023-2024)

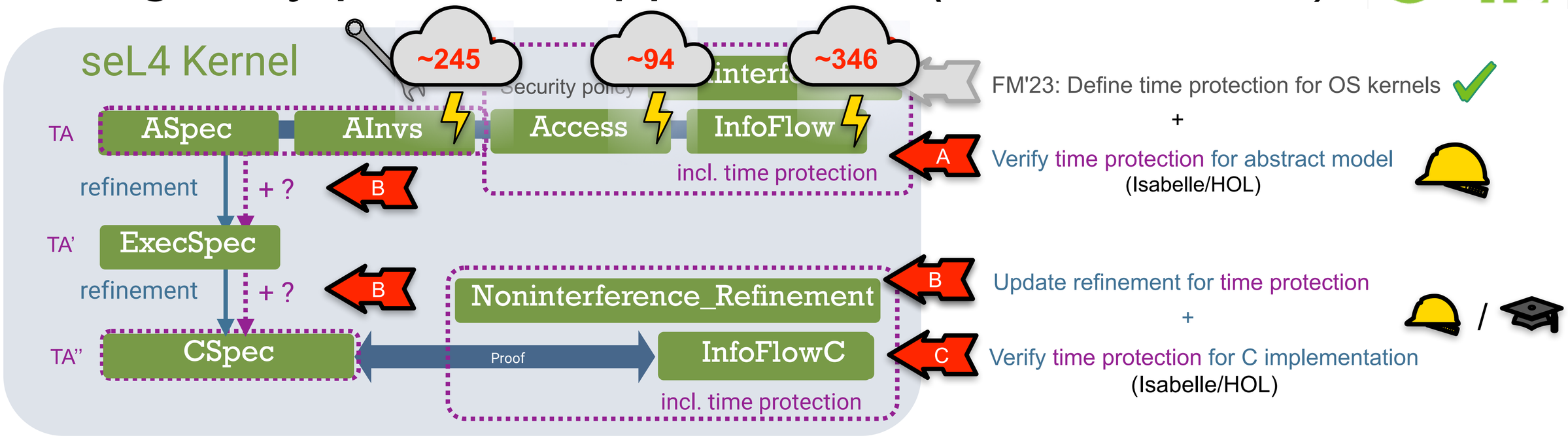


- Abstract model (**ASpec**) tracks *dynamic* touched addresses (TA) set
  Many existing proofs rely on *nothing changing*
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (as per *security policy*)
  THUS, Too many breakages in **AInvs**, **Access**, **InfoFlow** & beyond…
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)

This changes
> *125* times

> 685, *after* the ones
we already fixed

Originally planned approach… (c. 2023-2024)

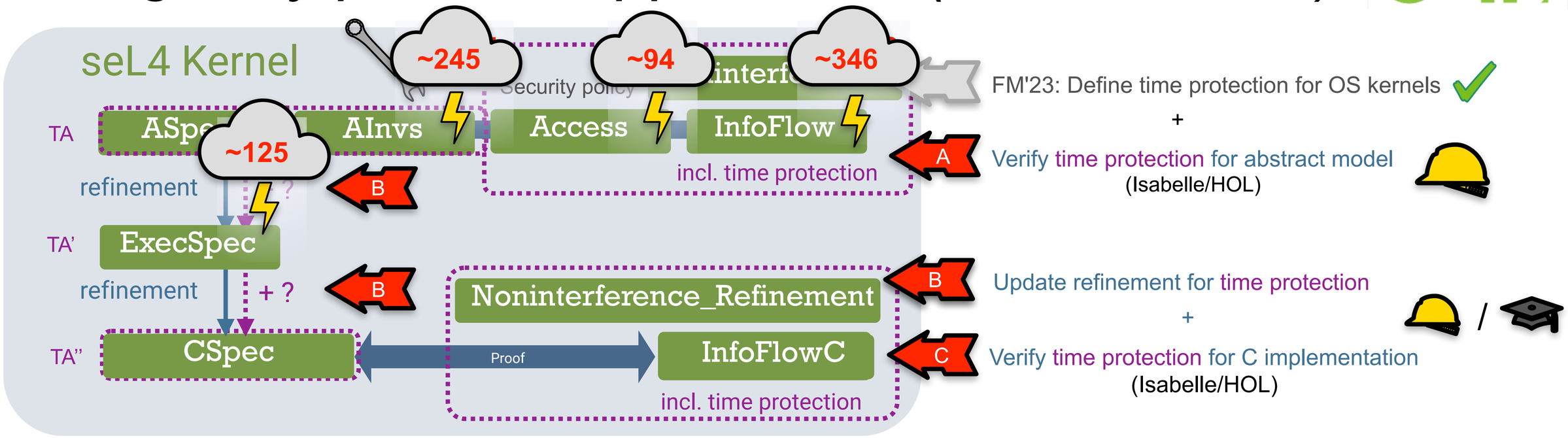


- Abstract model (**ASpec**) tracks *dynamic* touched addresses (TA) set
  Many existing proofs rely on *nothing changing*
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (as per *security policy*)
  THUS, Too many breakages in **AInvs**, **Access**, **InfoFlow** & beyond…
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)
  ExecSpec, **CSpec** access new addresses

This changes > *125* times

> 685, *after* the ones we already fixed

Originally planned approach… (c. 2023-2024)

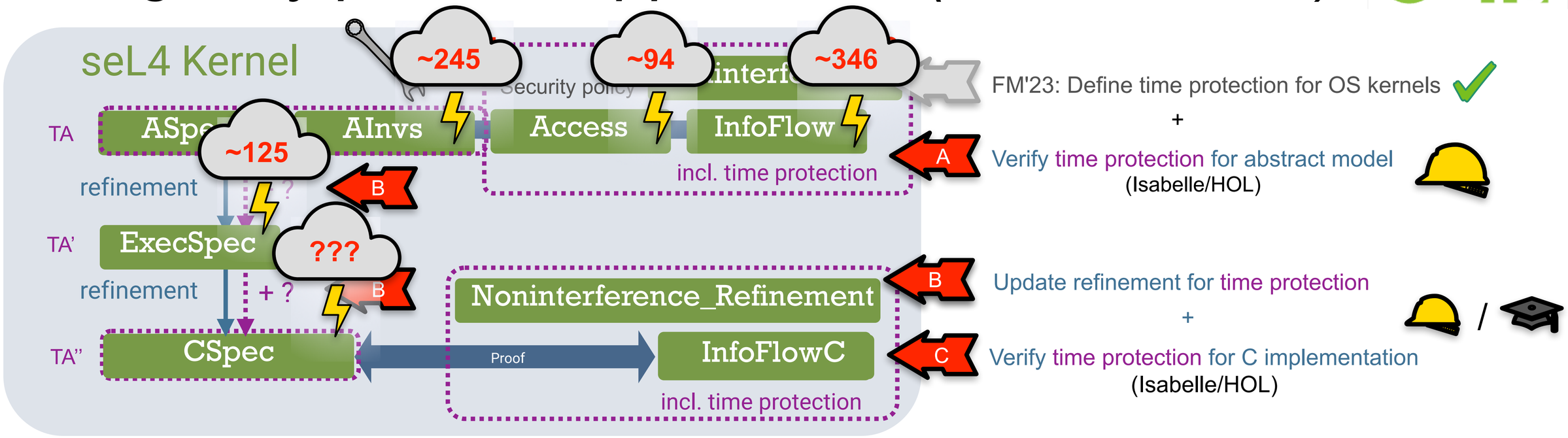


- Abstract model (**ASpec**) tracks *dynamic* touched addresses (TA) set
  Many existing proofs rely on *nothing changing*
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (as per *security policy*)
  THUS, Too many breakages in **AInvs**, **Access**, **InfoFlow** & beyond…
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)
  ExecSpec, **CSpec** access new addresses

This changes
> *125* times

> 685, *after* the ones
we already fixed

Originally planned approach… (c. 2023-2024)



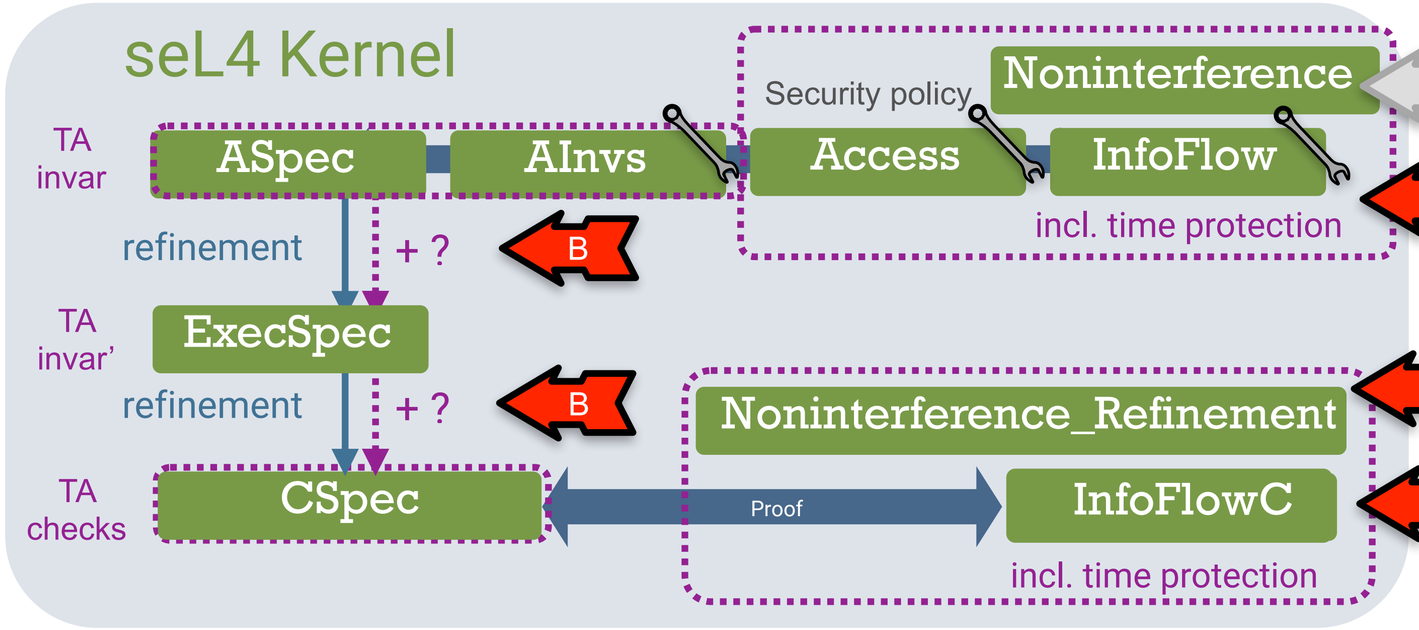
- Abstract model (**ASpec**) tracks *dynamic* touched addresses (TA) set
  Many existing proofs rely on *nothing changing*
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (as per *security policy*)
  THUS, Too many breakages in **AInvs**, **Access**, **InfoFlow** & beyond…
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)
  ExecSpec, **CSpec** access new addresses

This changes > *125* times

> 685, *after* the ones we already fixed

+ ??? at lower levels!

New approach & progress so far



FM'23: Define time protection for OS kernels ✓

+

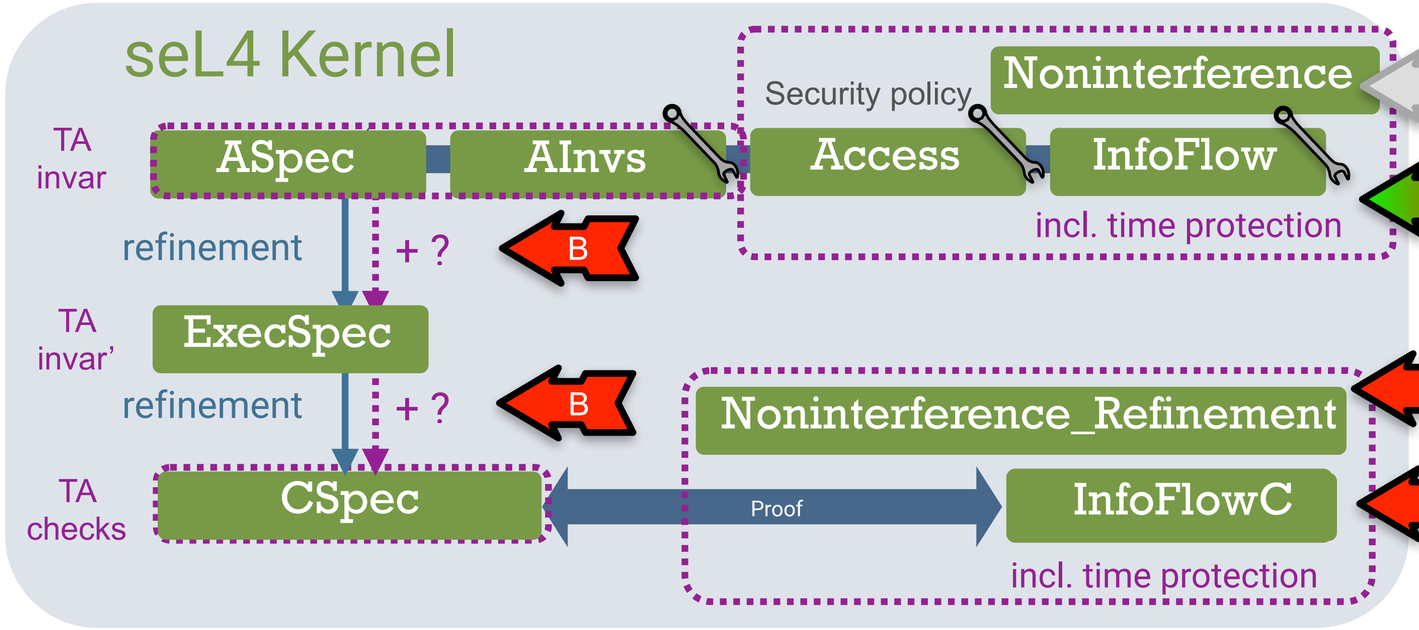
A Verify time protection for abstract model (Isabelle/HOL)

B Update refinement for time protection

+

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

New approach & progress so far



FM'23: Define time protection for OS kernels ✓

+

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

B — Update refinement for time protection

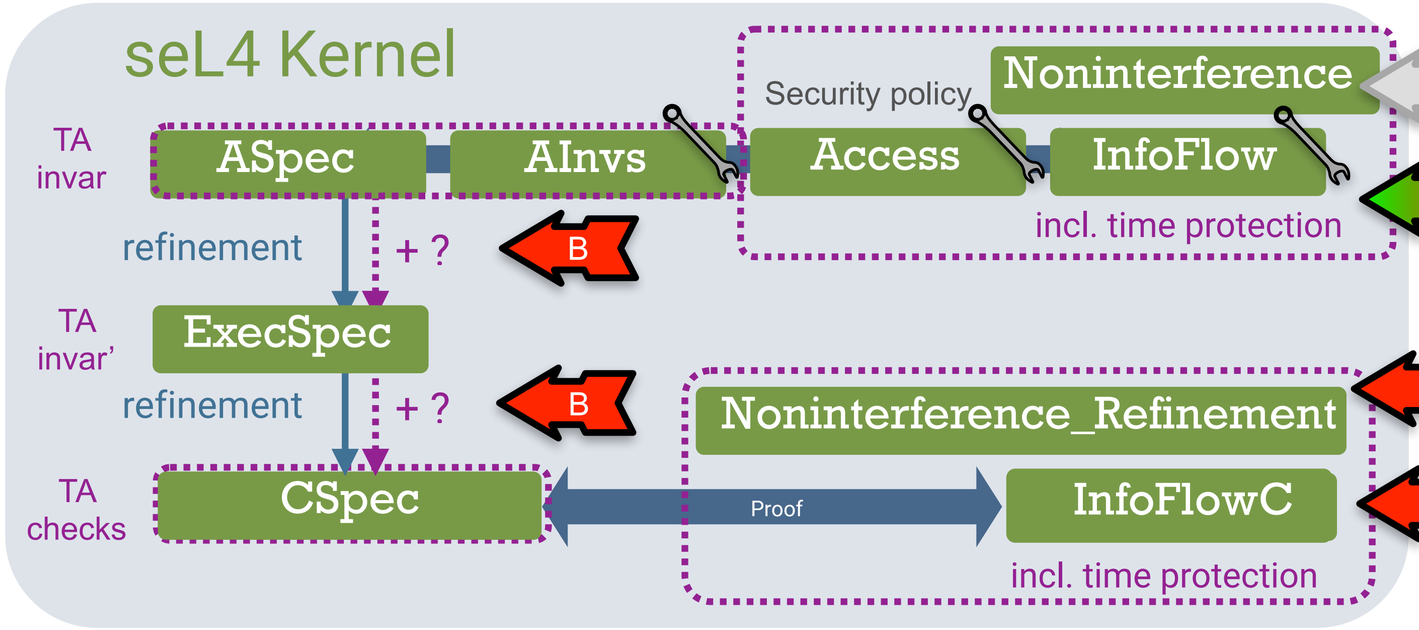
+

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

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

Sai Nair
BSc (Hons)
thesis

New approach & progress so far



FM'23: Define time protection for OS kernels ✓

+

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

B — Update refinement for time protection

+

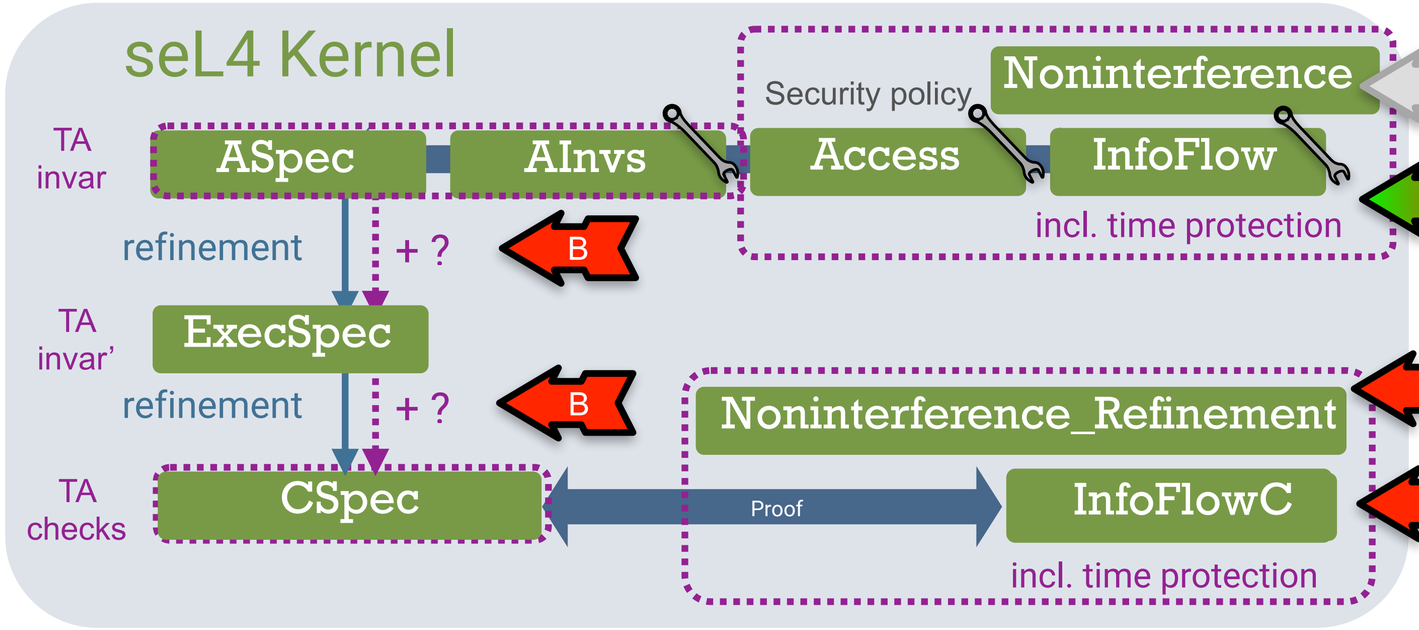
C — Verify time protection for C implementation (Isabelle/HOL)

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

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

Sai Nair
BSc (Hons)
thesis

New approach & progress so far



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)

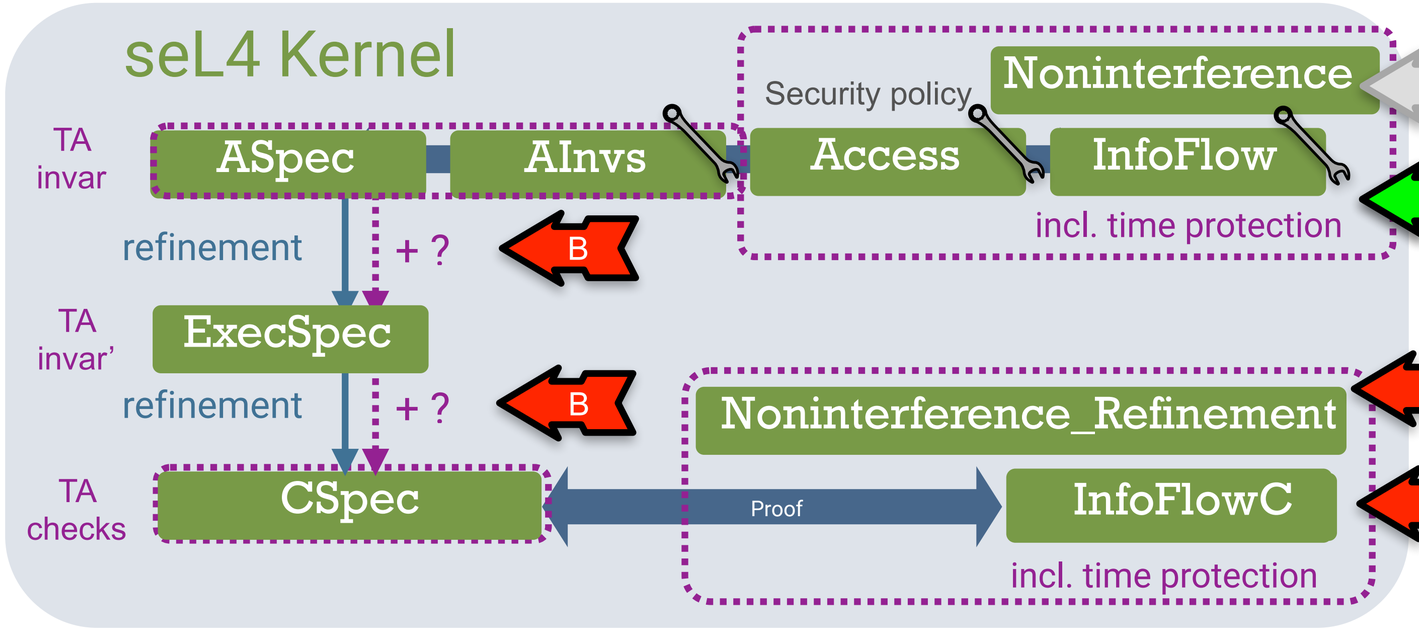
"obj refs don't point outside static *TA set"*

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

Continuing on from

Sai Nair
BSc (Hons)
thesis

New approach & progress so far



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)

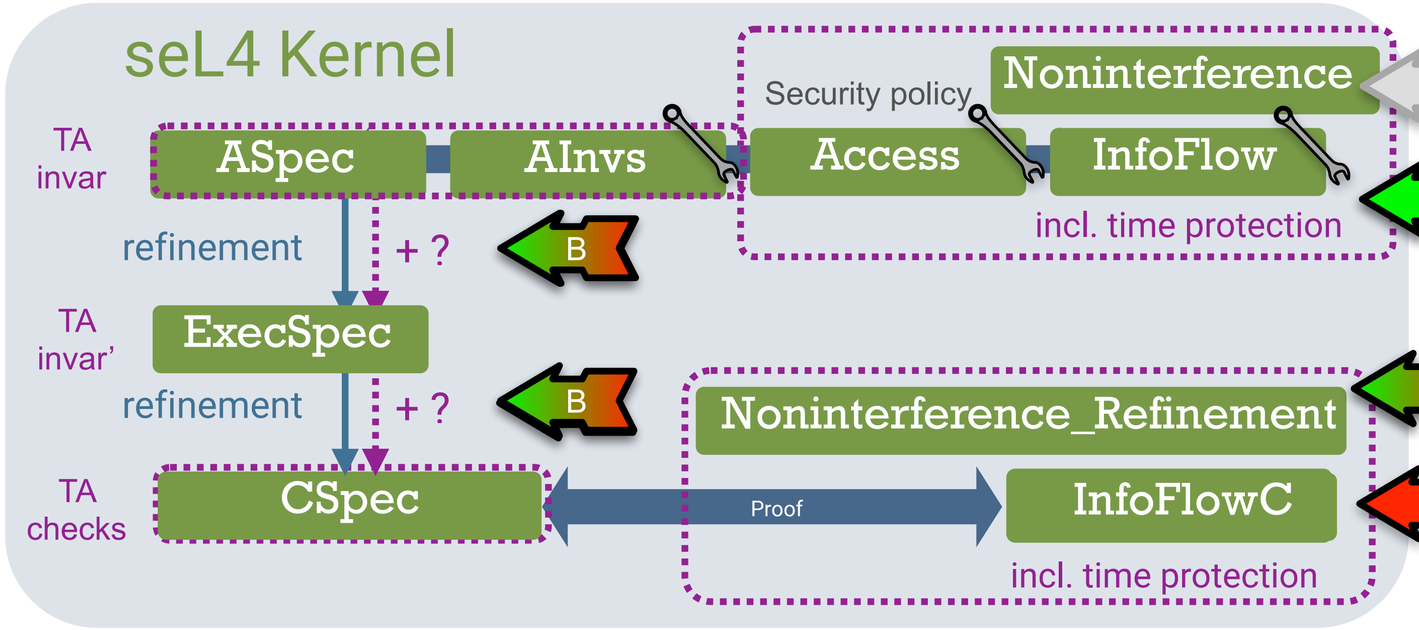
"obj refs don't point outside static *TA set"*

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

Continuing on from

Sai Nair
BSc (Hons)
thesis

New approach & progress so far



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 our time protection proofs 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

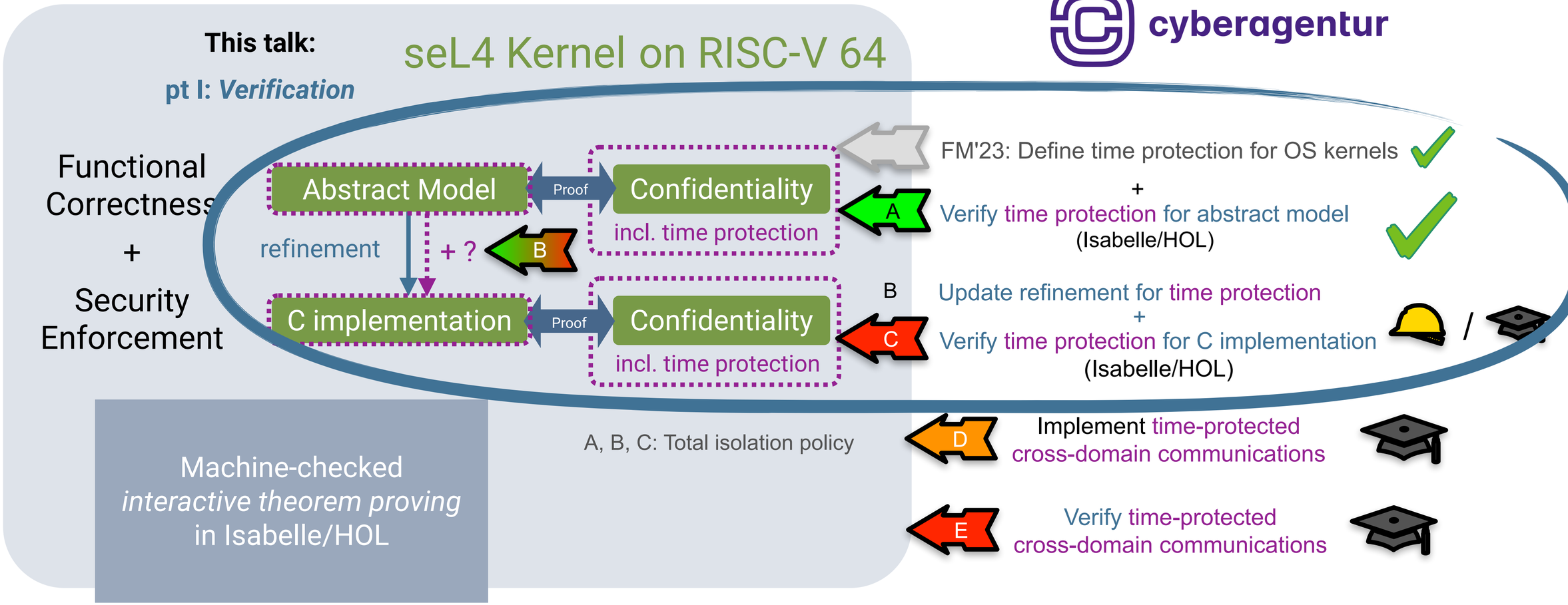
Thomas Liang
BSc (Hons)
Thesis

Time protection verification for seL4

Since the plan 2 years ago…

Thanks to funding from

cyberagentur

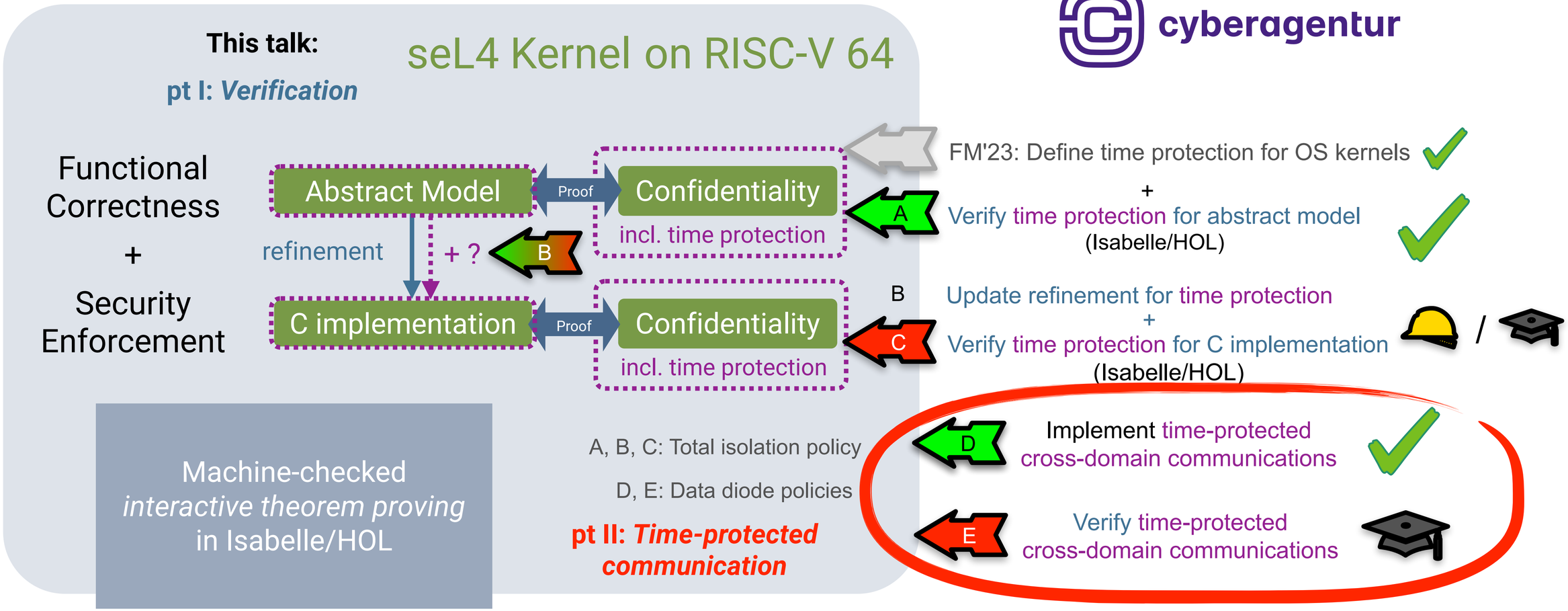


Time protection verification for seL4

The status today:

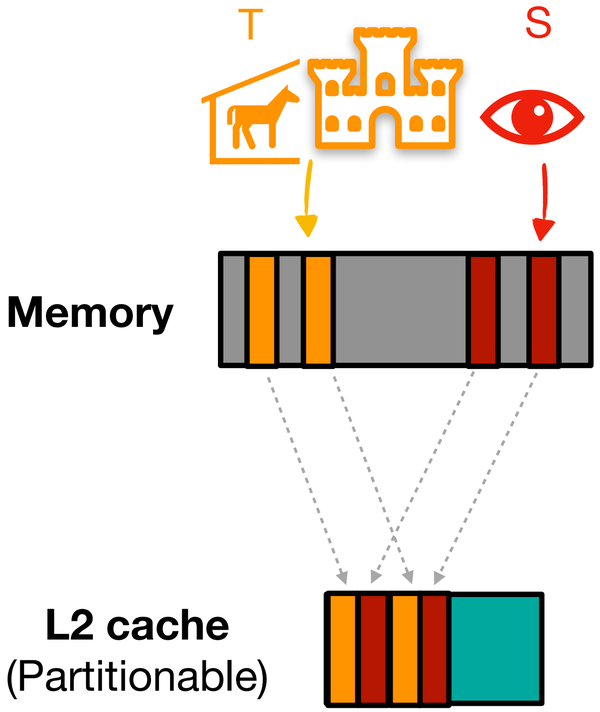
Thanks to funding from

cyberagentur



What are time-protected communications?

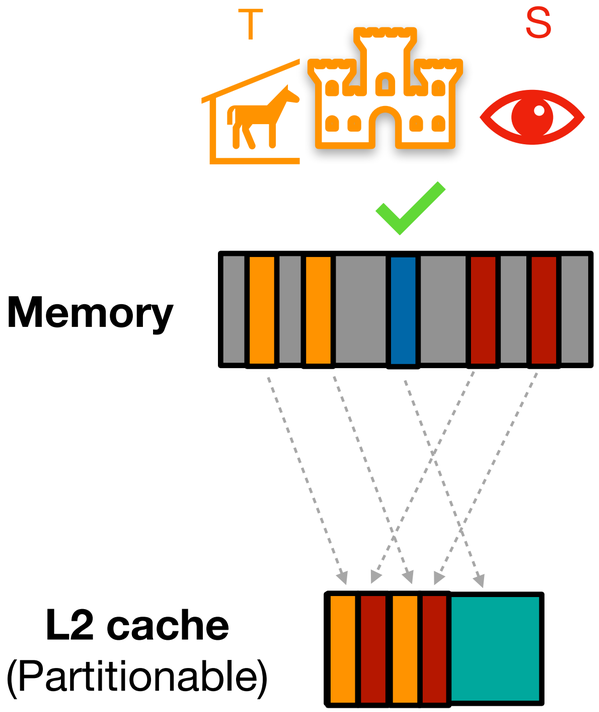
- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch



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
- What if we want a *data diode* via **shared memory** (for performant data transfer)?

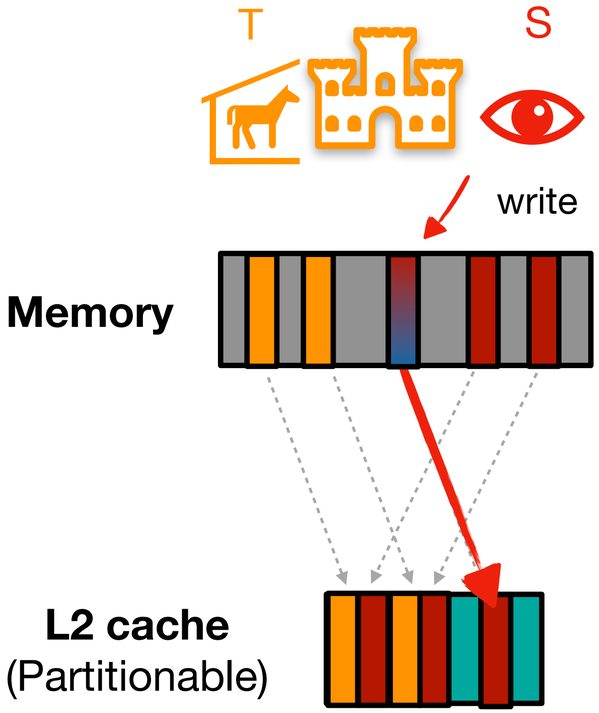
  Allowed: T <~ S, *Disallowed*: T ~/> S



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
- What if we want a *data diode* via **shared memory** (for performant data transfer)?

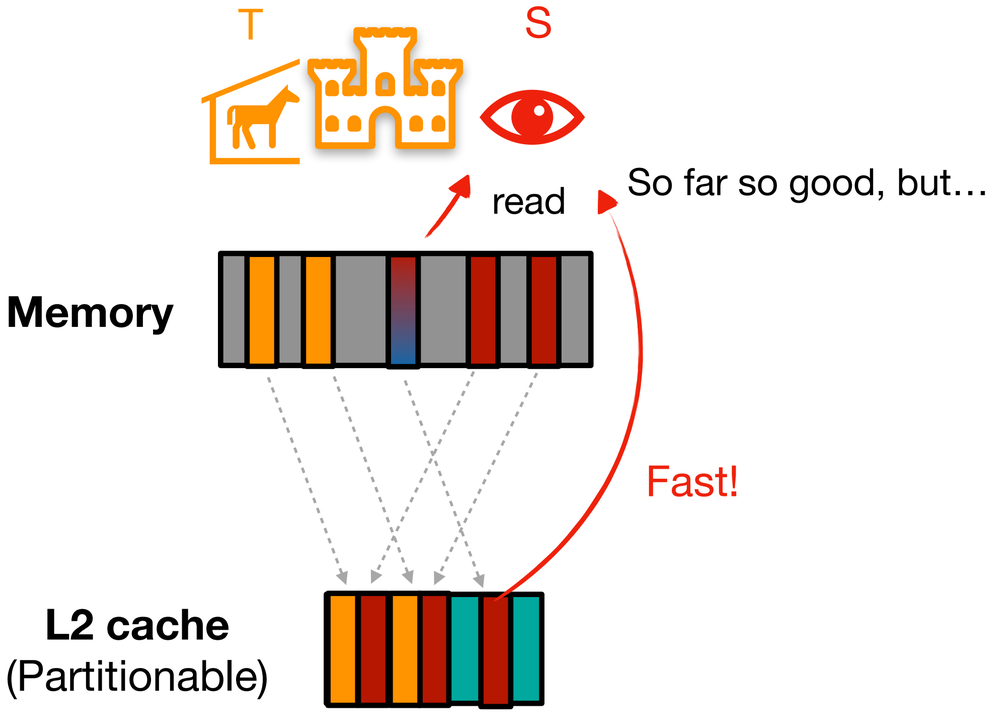
  Allowed: T <~ S, *Disallowed*: T ~/> S



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
- What if we want a *data diode* via **shared memory** (for performant data transfer)?

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

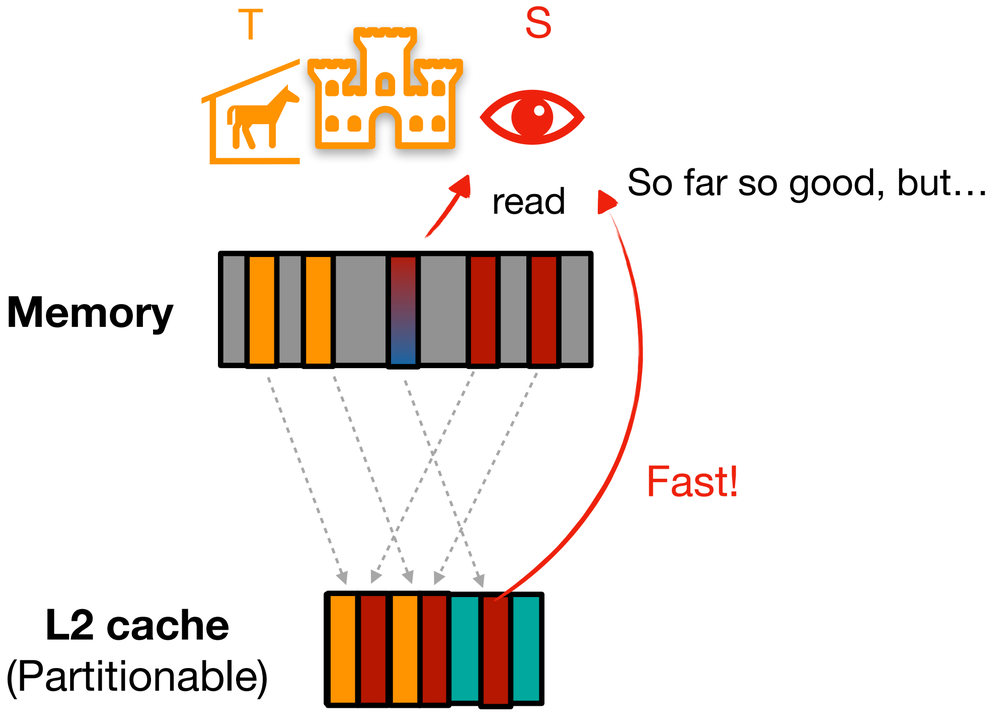


What are time-protected communications?

- 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
- What if we want a *data diode* via **shared memory** (for performant data transfer)?

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

  Mere Reads by T can evict lines from the L2:



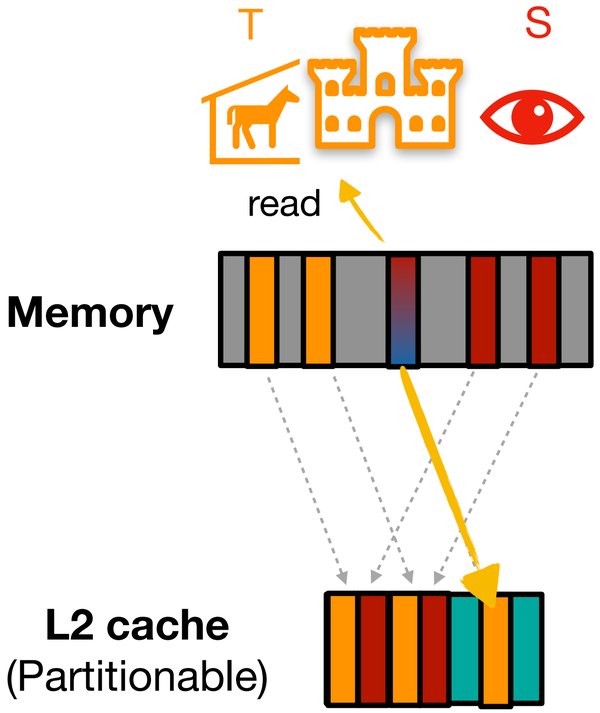
What are time-protected communications?

- 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 if we want a *data diode* via **shared memory** (for performant data transfer)?

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

  Mere Reads by T can evict lines from the L2:

  S will see a timing difference!



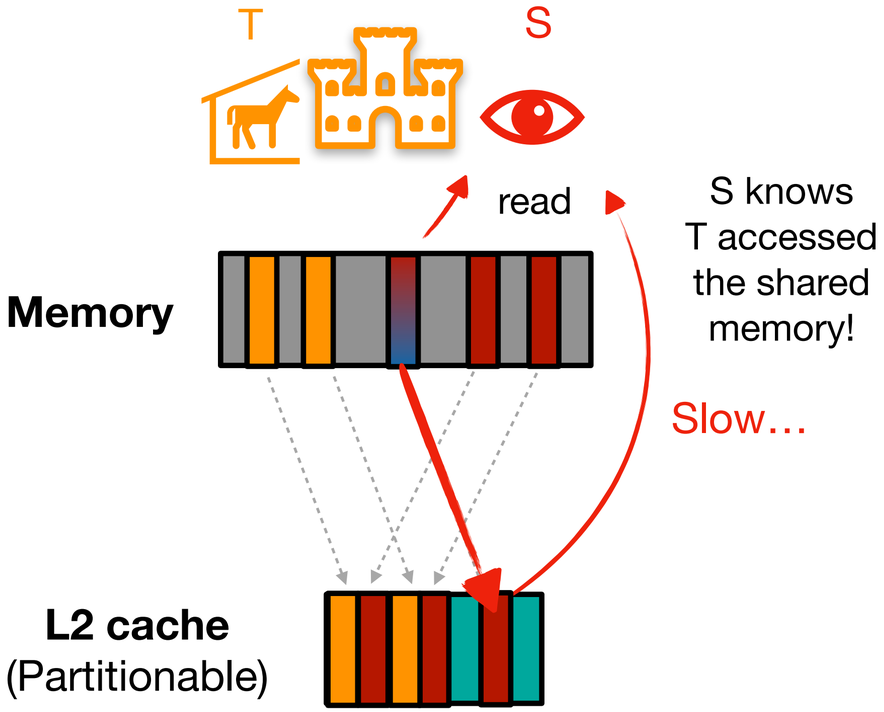
What are time-protected communications?

- 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 if we want a *data diode* via **shared memory** (for performant data transfer)?

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

  Mere Reads by T can evict lines from the L2:

  S will see a timing difference!



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 if we want a *data diode* via **shared memory** (for performant data transfer)?

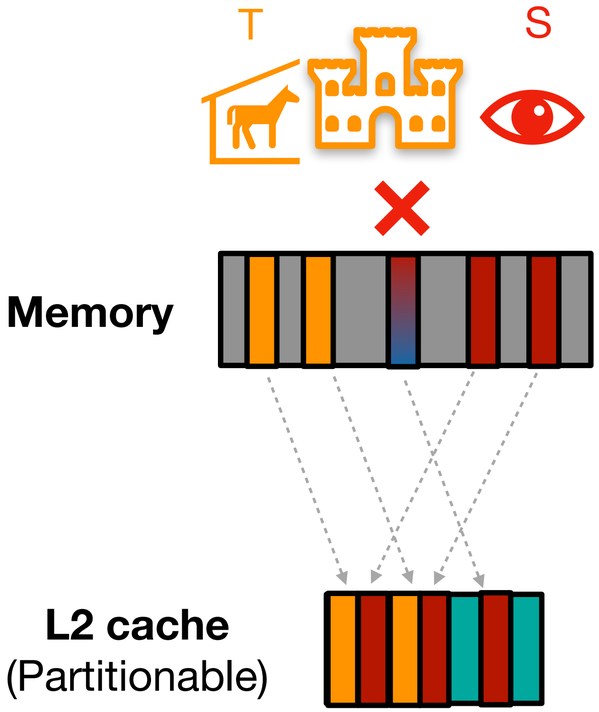
  Allowed: T <~ S, *Disallowed*: T ~/> S

  Mere Reads by T can evict lines from the L2:

  S will see a timing difference!

  Takeaways:

  - T and S cannot share memory, or



What are time-protected communications?

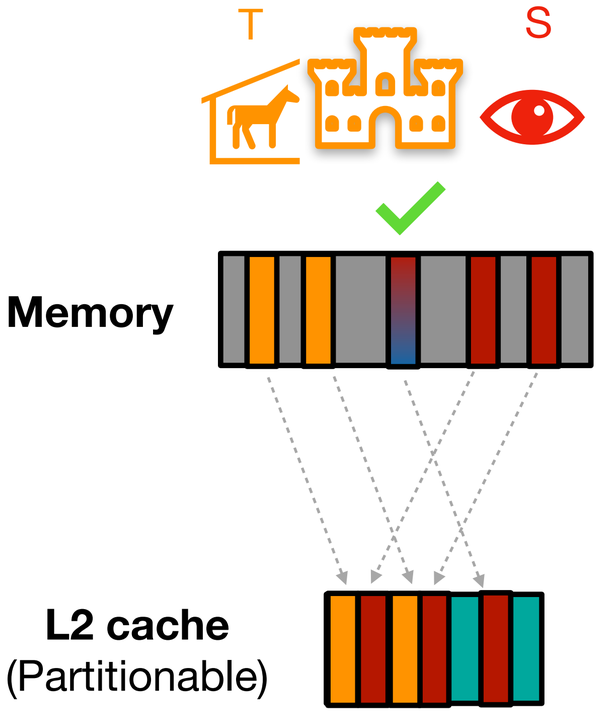
- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- What if we want a *data diode* via **shared memory** (for performant data transfer)?

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

  Mere Reads by T can evict lines from the L2:

  S will see a timing difference!

  Takeaways:
  - T and S cannot share memory, or
  - We need a *targeted flush* for the L2 cache



What are time-protected communications?

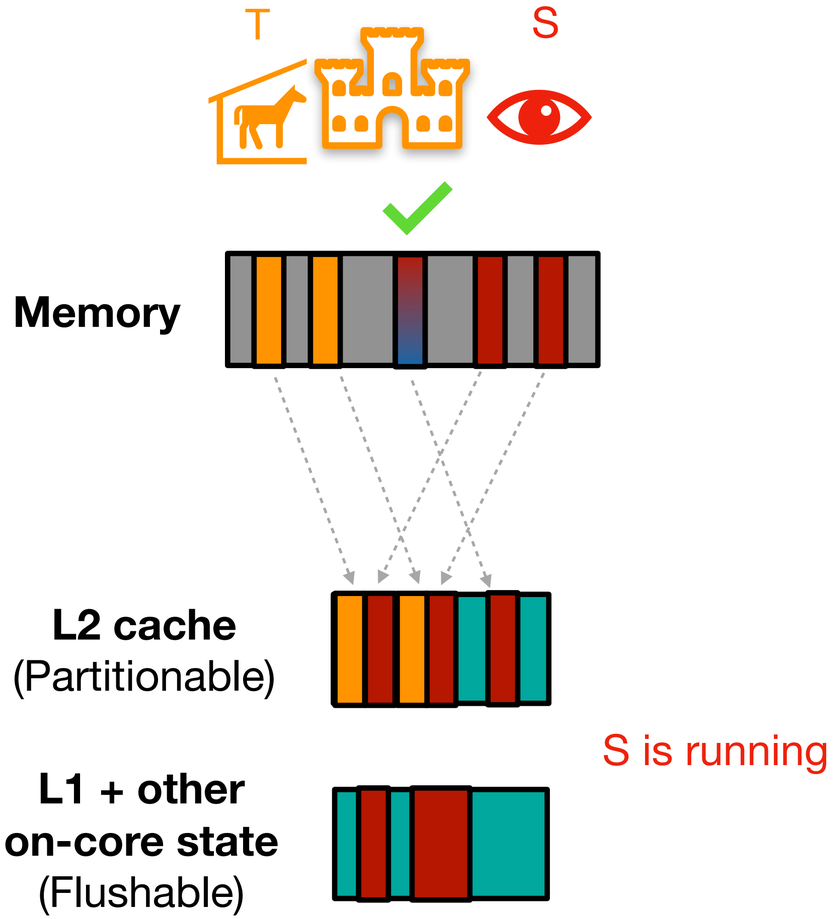
- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- What if we want a *data diode* via **shared memory** (for performant data transfer)?

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

  Mere Reads by T can evict lines from the L2:

  S will see a timing difference!

  Takeaways:
  - T and S cannot share memory, or
  - We need a *targeted flush* for the L2 cache



What are time-protected communications?

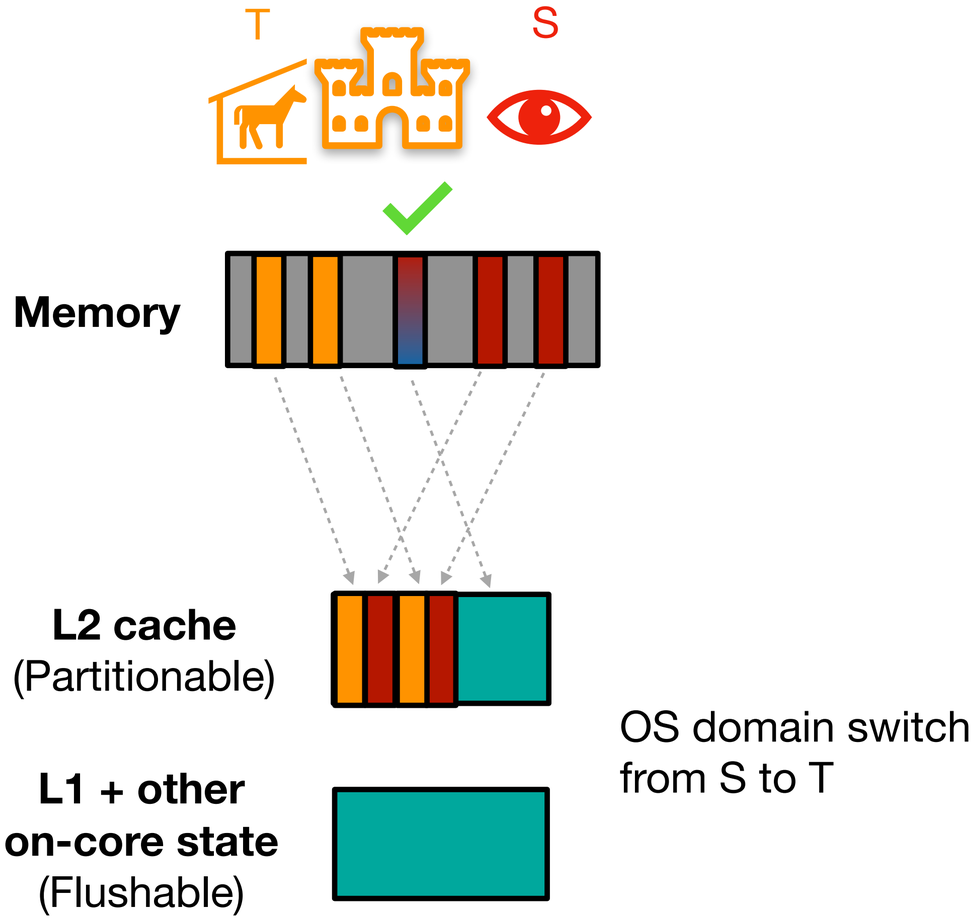
- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- What if we want a *data diode* via **shared memory** (for performant data transfer)?

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

  Mere Reads by T can evict lines from the L2:

  S will see a timing difference!

  Takeaways:
  - T and S cannot share memory, or
  - We need a *targeted flush* for the L2 cache



What are time-protected communications?

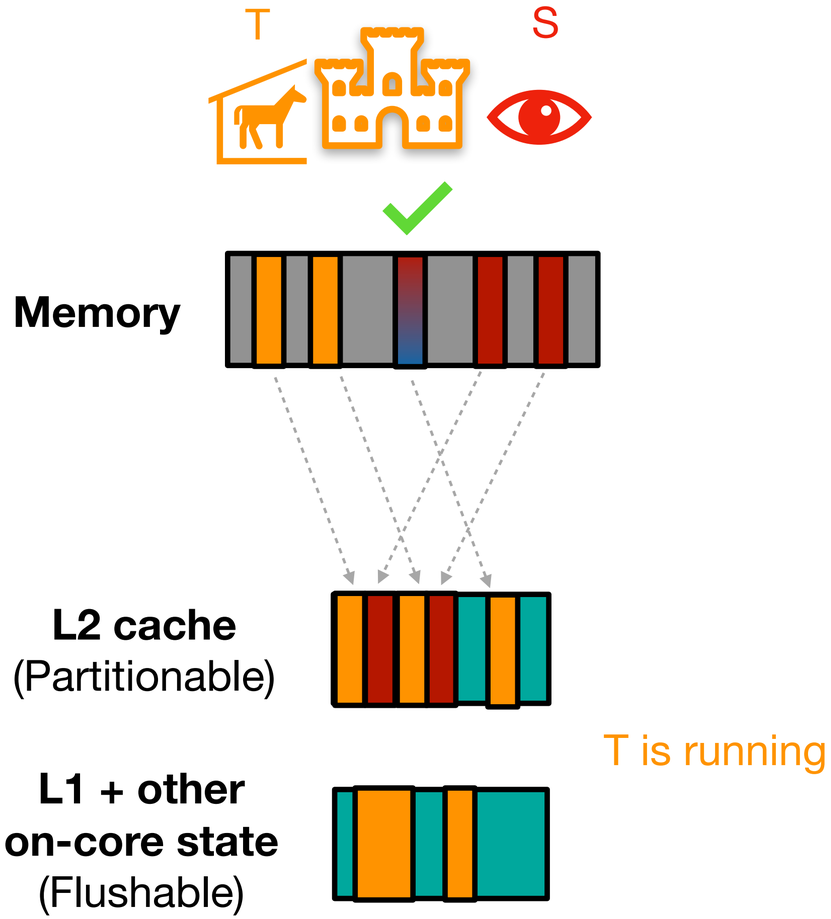
- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- What if we want a *data diode* via **shared memory** (for performant data transfer)?

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

  Mere Reads by T can evict lines from the L2:

  S will see a timing difference!

  Takeaways:
  - T and S cannot share memory, or
  - We need a *targeted flush* for the L2 cache



What are time-protected communications?

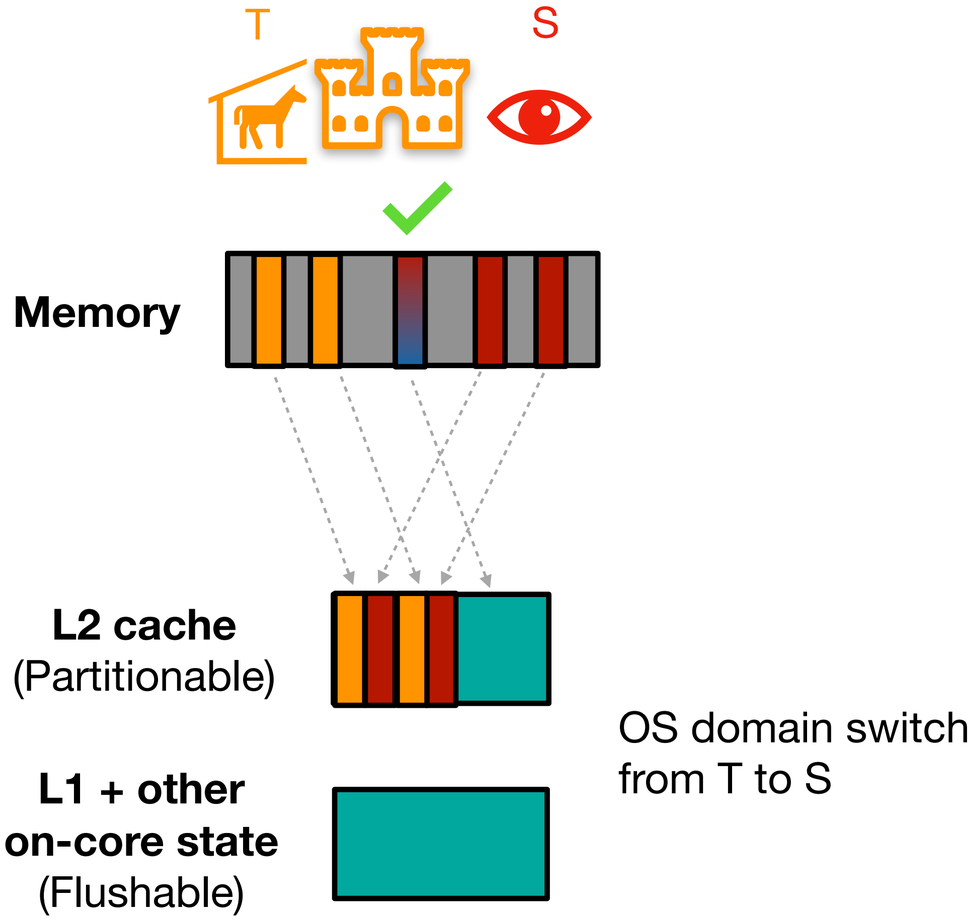
- OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- What if we want a *data diode* via **shared memory** (for performant data transfer)?

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

  Mere Reads by T can evict lines from the L2:

  S will see a timing difference!

  Takeaways:
  - T and S cannot share memory, or
  - We need a *targeted flush* for the L2 cache

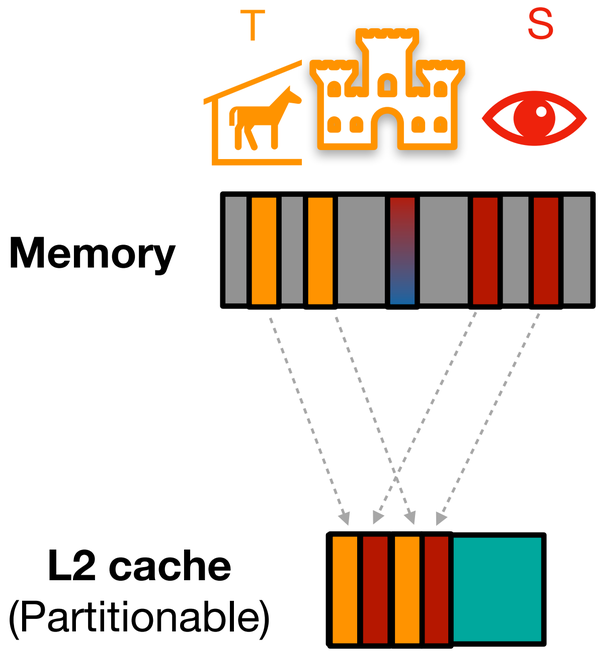


Time-protected communications in seL4

seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

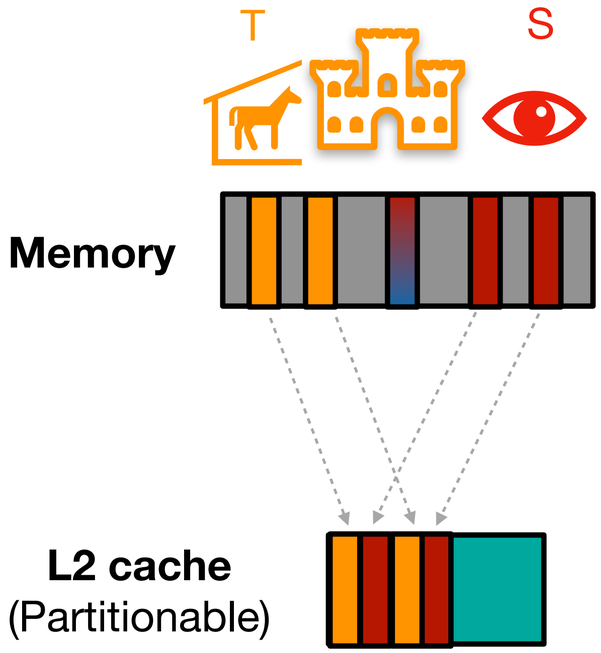
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

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

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

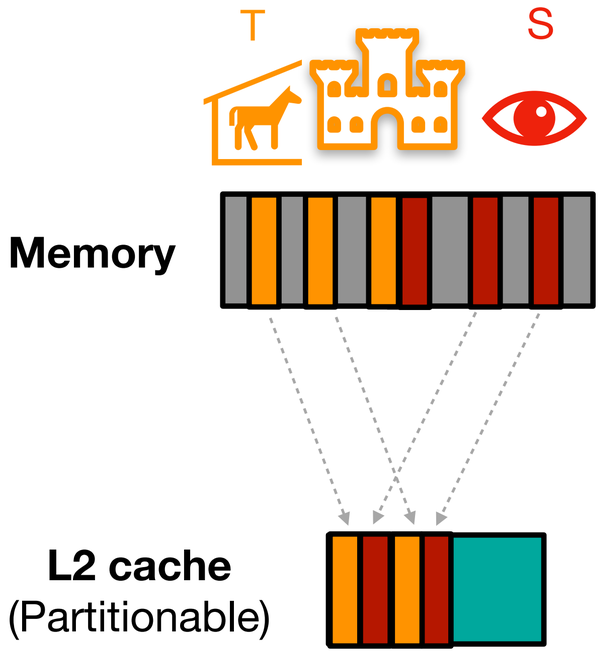
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

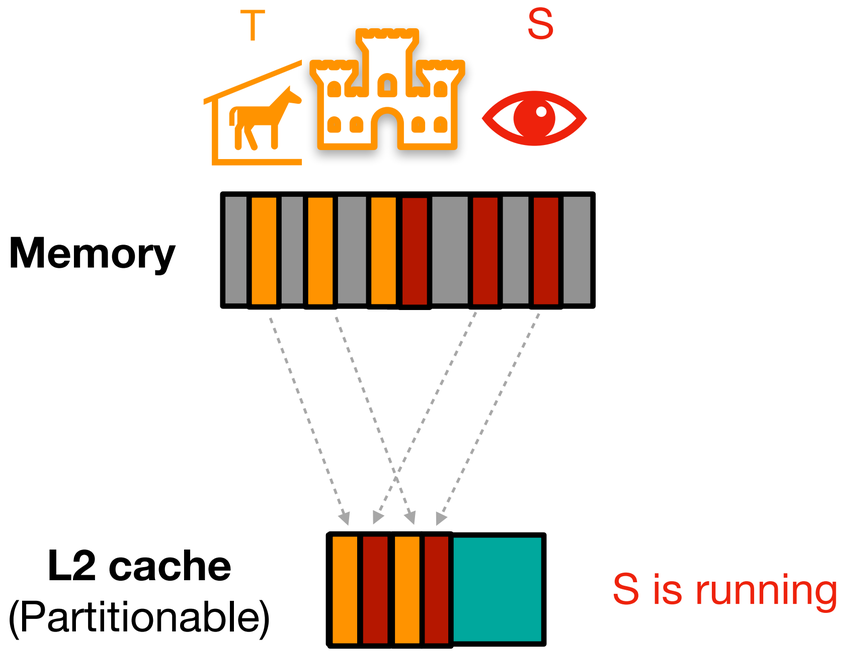
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

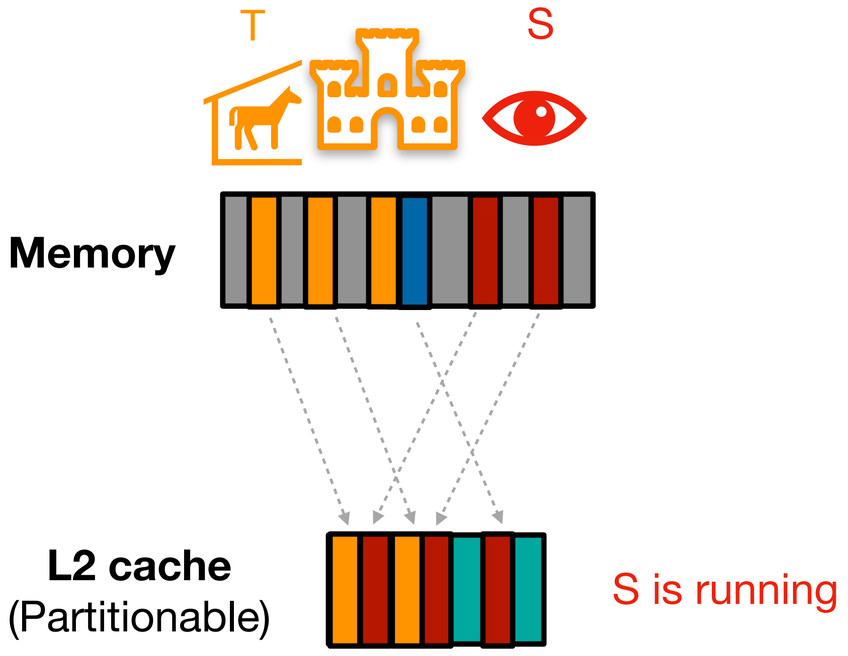
seL4 on RISC-V

D Implement time-protected cross-domain communications

E Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

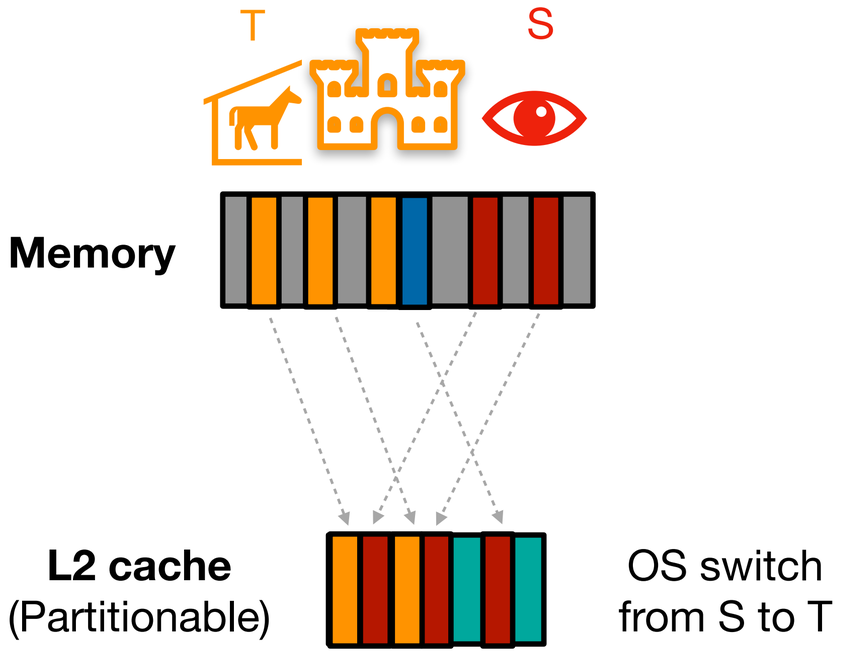
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

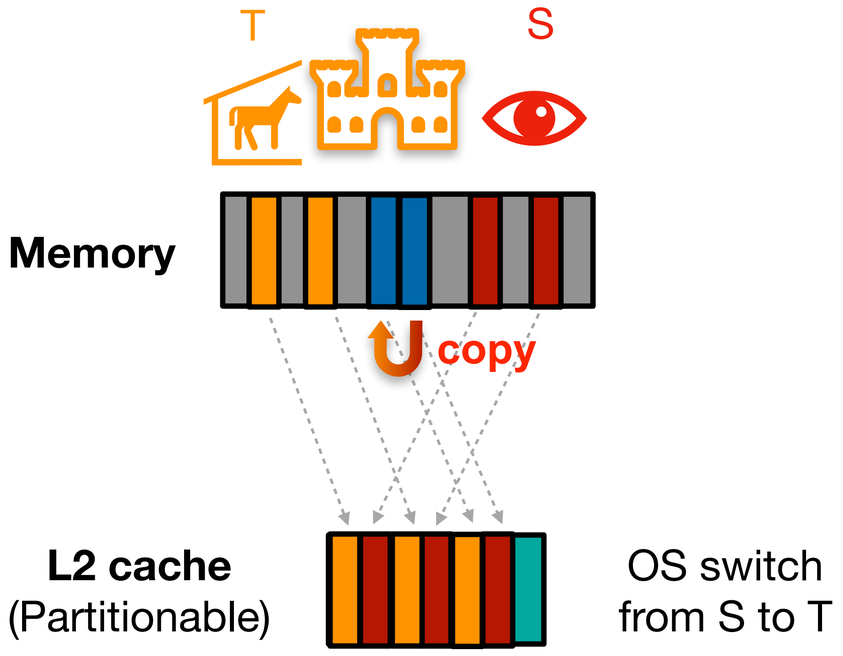
seL4 on RISC-V

D Implement time-protected cross-domain communications

E Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

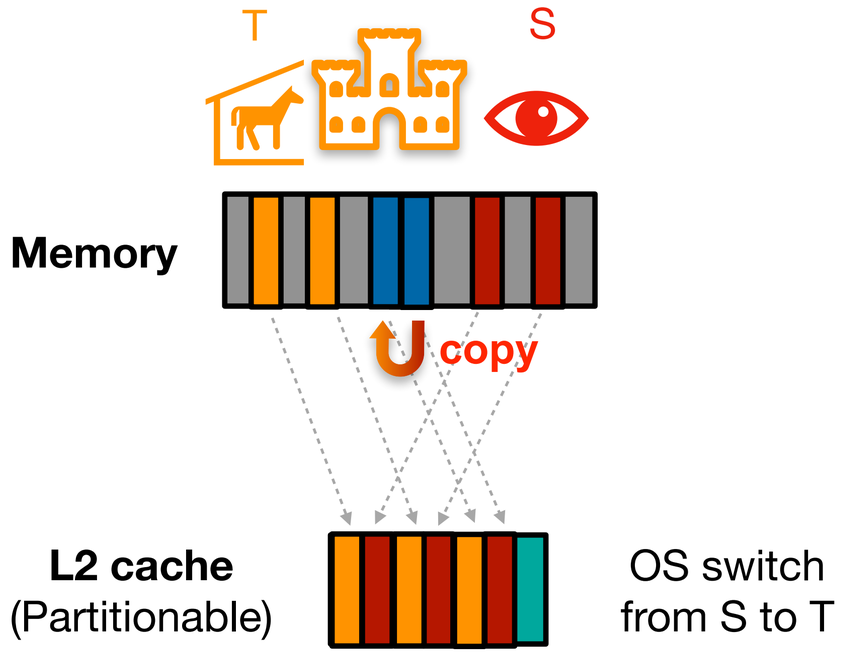
seL4 on RISC-V

D Implement time-protected cross-domain communications

E Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

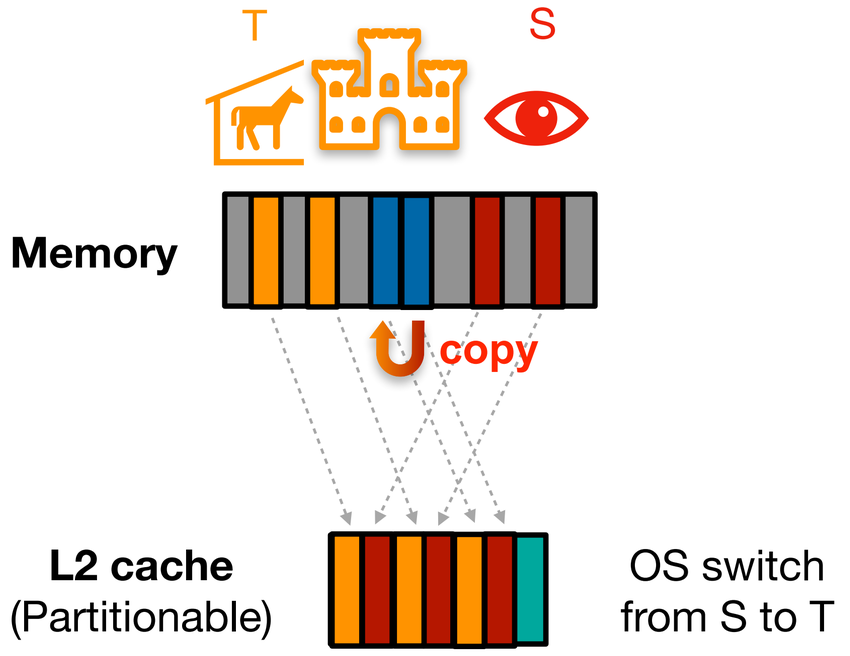
seL4 on RISC-V

D Implement time-protected cross-domain communications

E Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!

Varun Sethu
BE Thesis



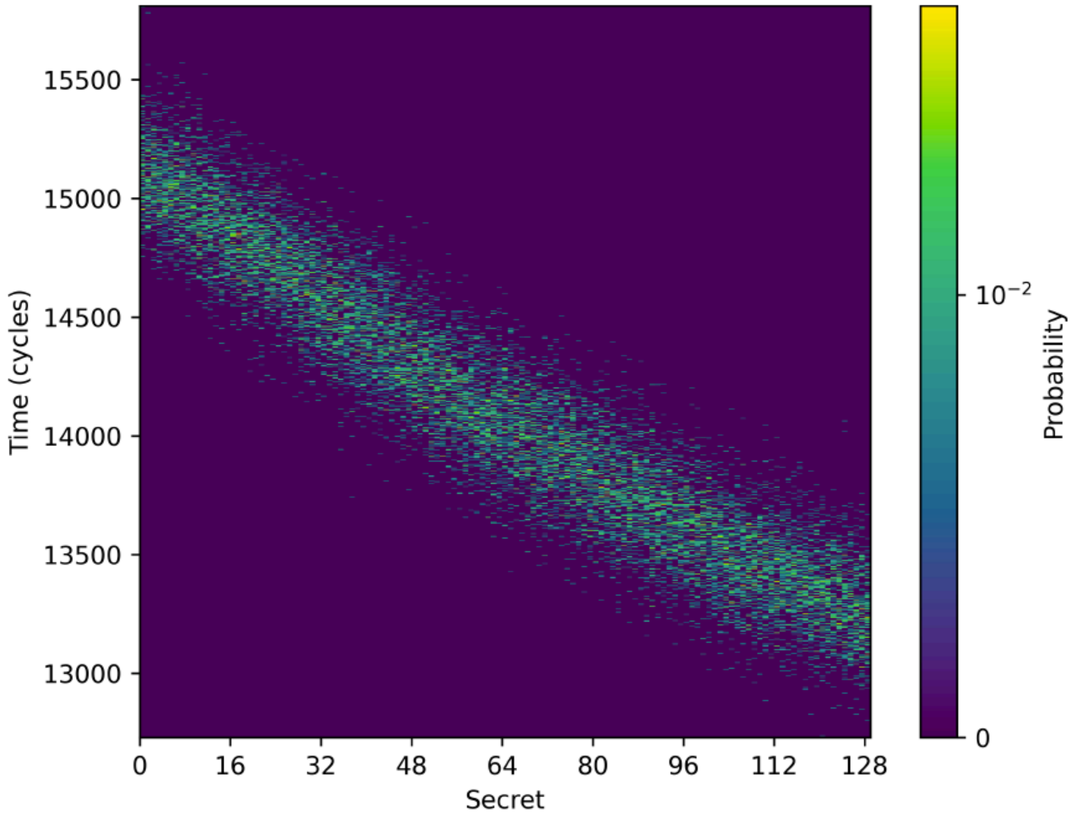
Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

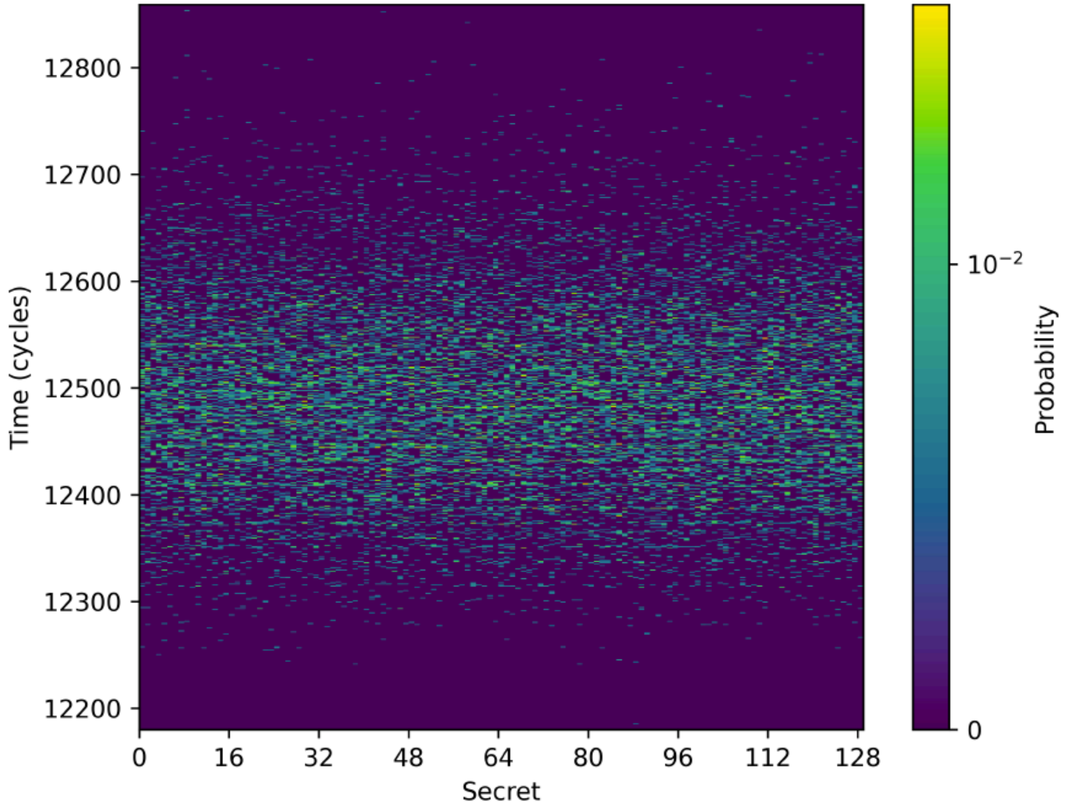
Time-protected communications in seL4

e.g. Spy probe time after Trojan reads secret # of lines

Non-time-protected
shared buffer



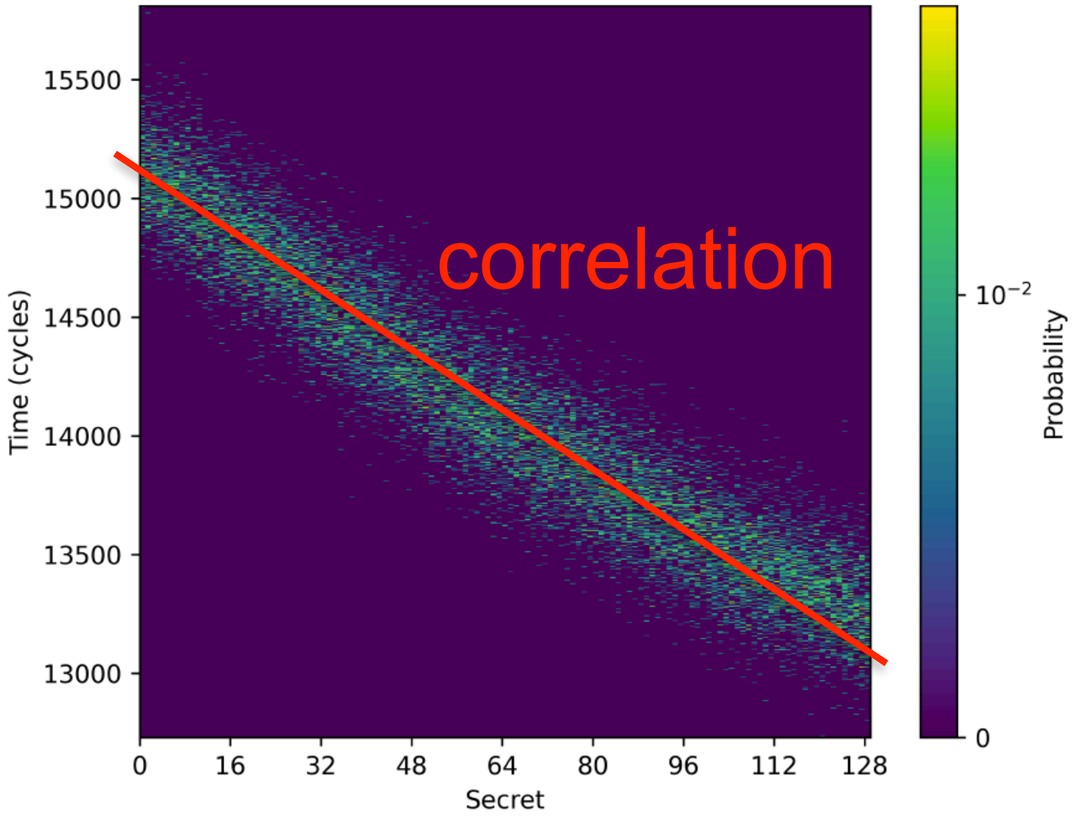
Time-protected copy between
non-shared buffers



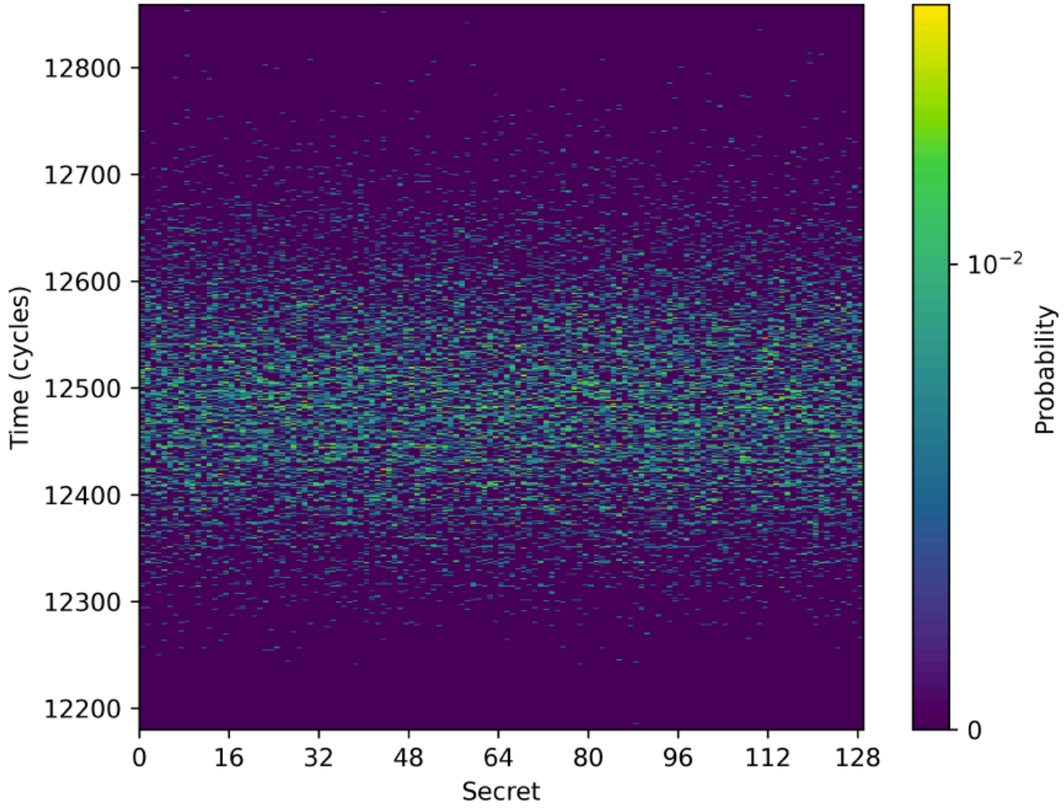
Time-protected communications in seL4

e.g. Spy probe time after Trojan reads secret # of lines

Non-time-protected
shared buffer



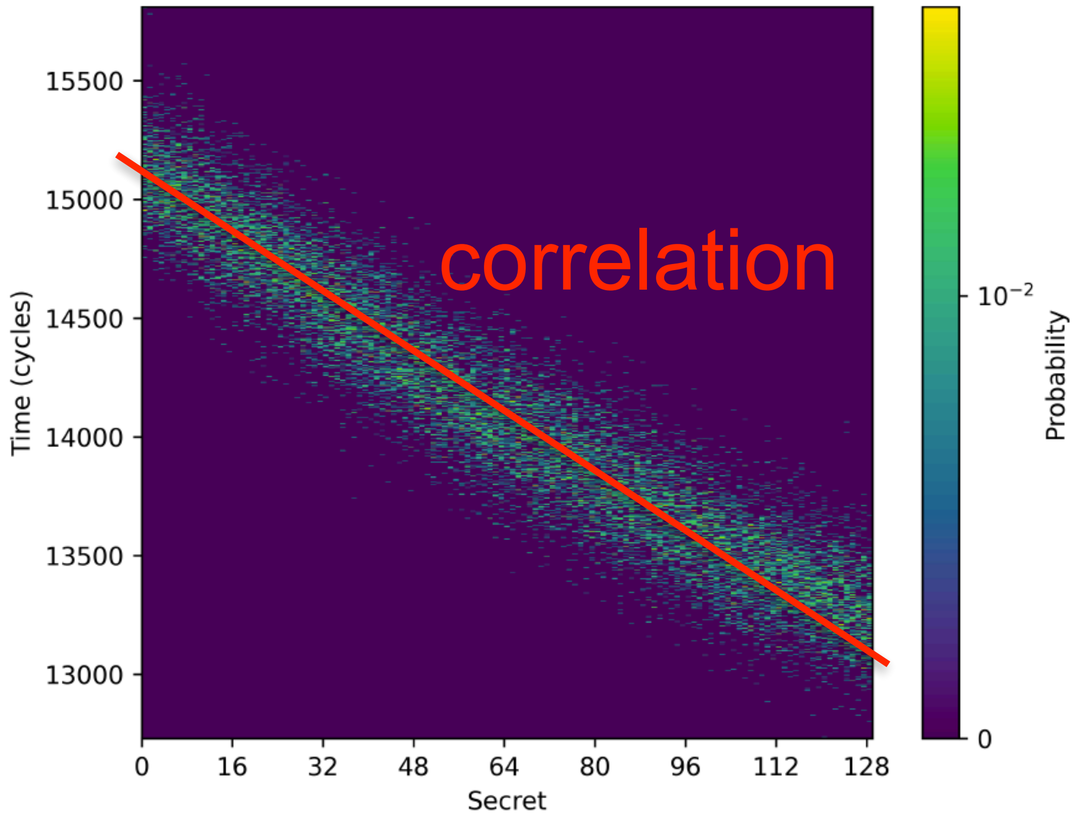
Time-protected copy between
non-shared buffers



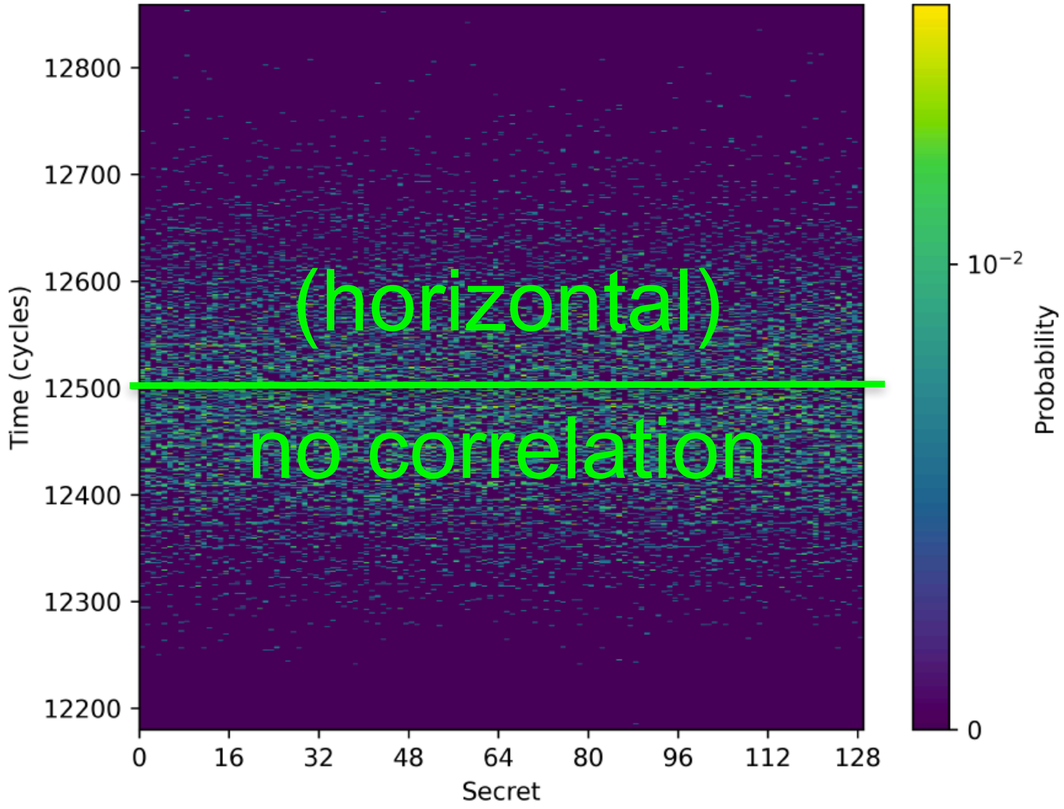
Time-protected communications in seL4

e.g. Spy probe time after Trojan reads secret # of lines

Non-time-protected
shared buffer



Time-protected copy between
non-shared buffers



Time-protected communications in seL4

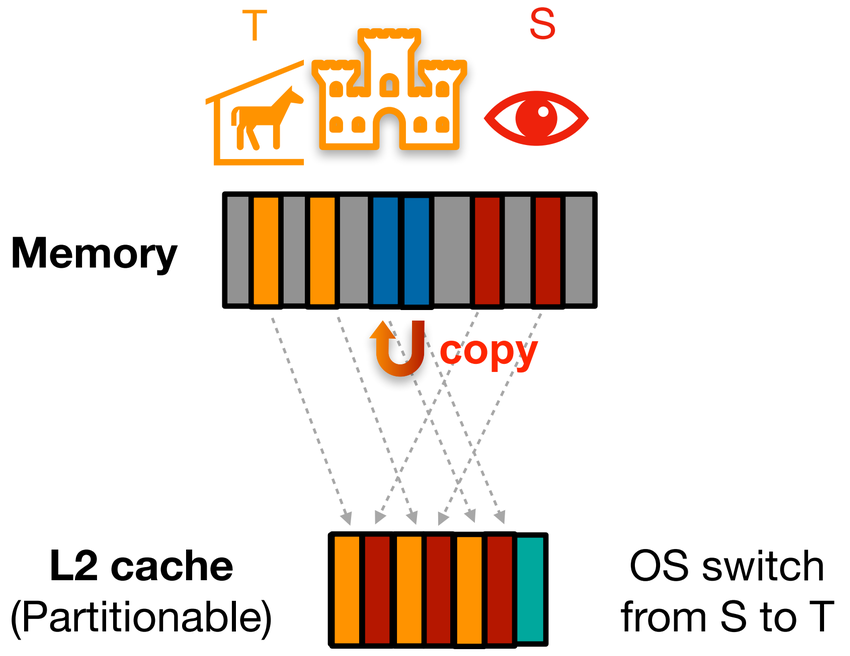
seL4 on RISC-V

D Implement time-protected cross-domain communications

E Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

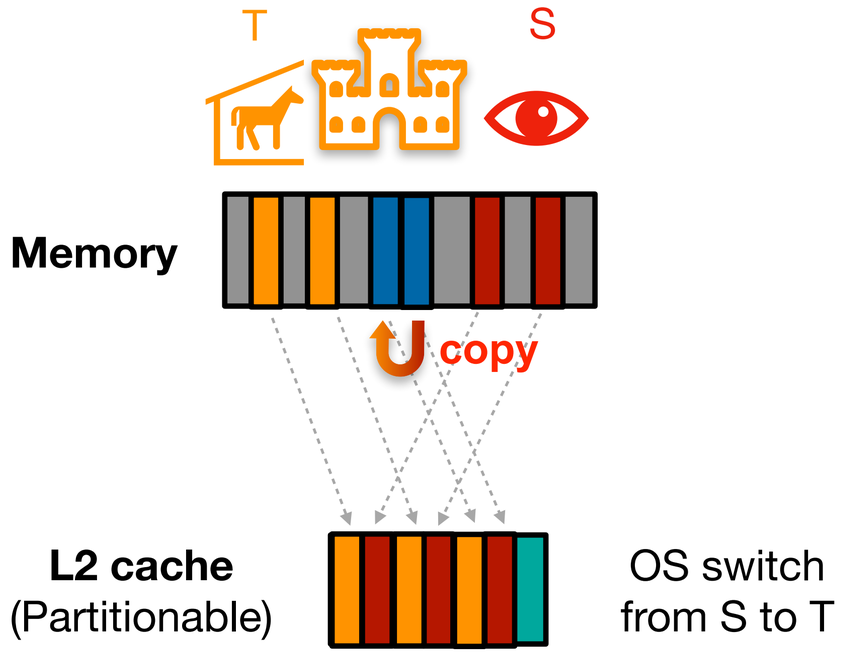
seL4 on RISC-V

D Implement time-protected cross-domain communications

E Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

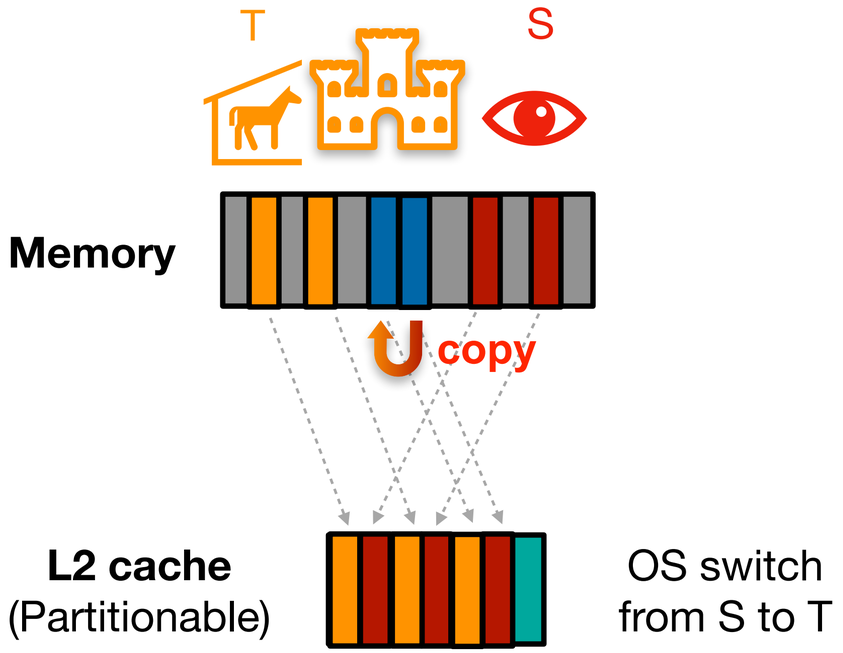
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!
    - ➡ **slow** (~19-34% overhead/page ≈ copy WCET)

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

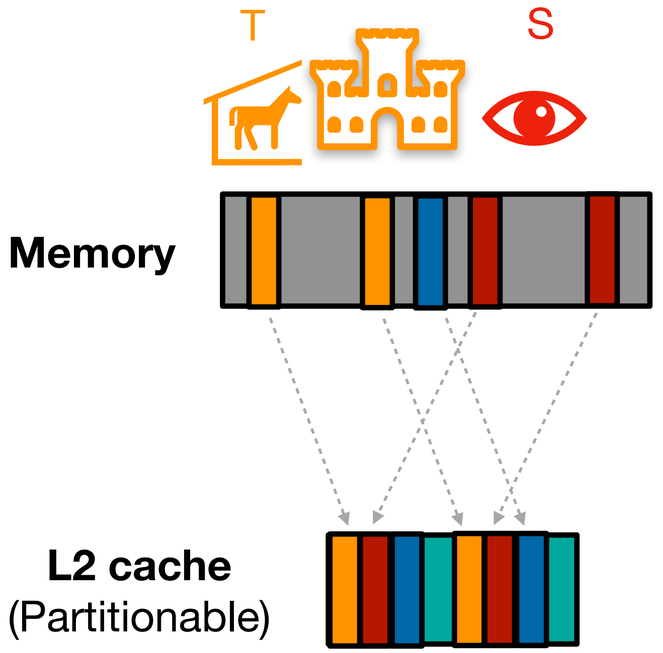
seL4 on RISC-V

D Implement time-protected cross-domain communications

E Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!
    - ➡ **slow** (~19-34% overhead/page ≈ copy WCET)
  - *Dedicate colours* for shared memory

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

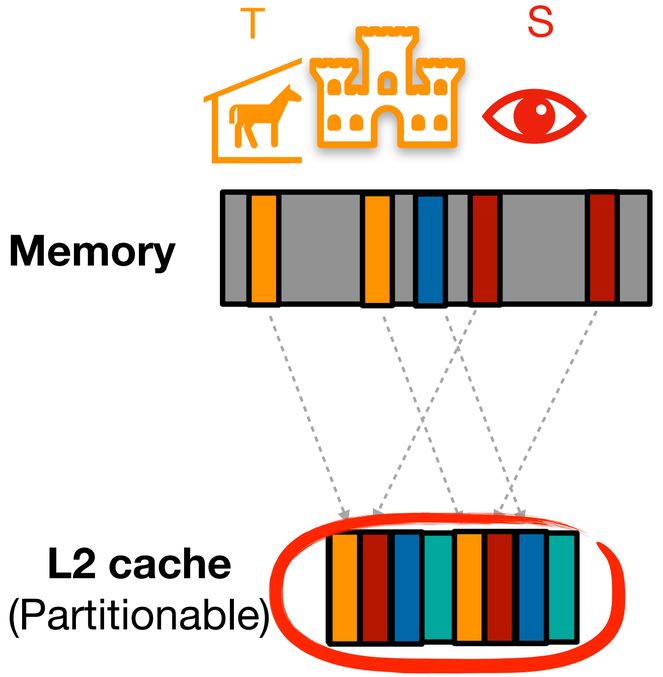
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!
    - ➡ **slow** (~19-34% overhead/page ≈ copy WCET)
  - *Dedicate colours* for shared memory
    - ➡ wastes space (e.g. only 16 colours on typical ARM boards)

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

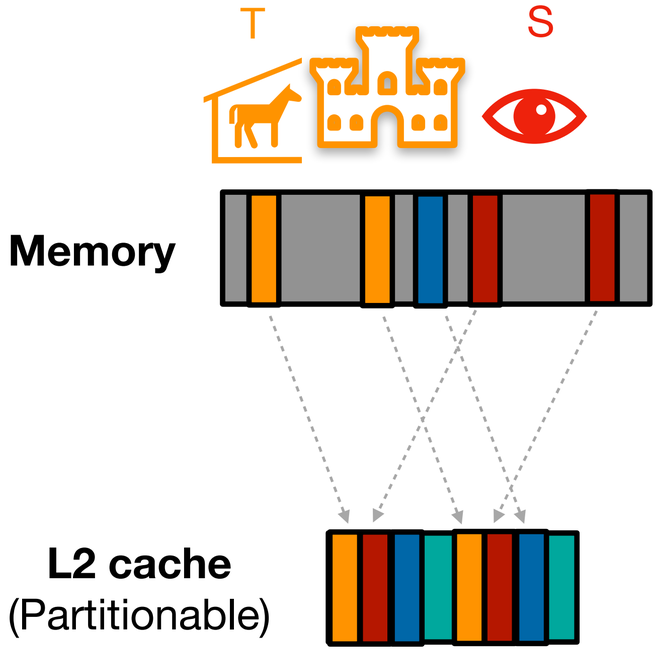
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!
    - ➡ **slow** (~19-34% overhead/page ≈ copy WCET)
  - *Dedicate colours* for shared memory
    - ➡ wastes space (e.g. only 16 colours on typical ARM boards)
    - ➡ targeted L2 flush via *prefetching,* a'la [Ge et al. 2019]

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

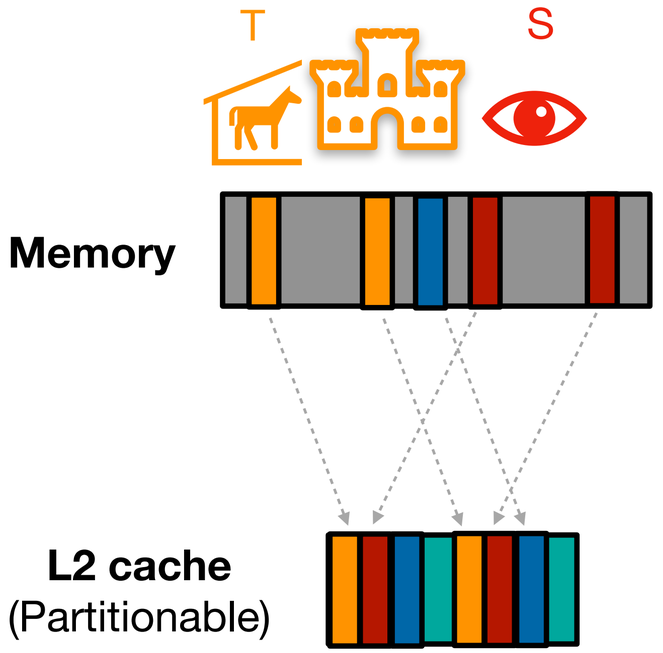
seL4 on RISC-V

D Implement time-protected cross-domain communications

E Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!
    - ➡ **slow** (~19-34% overhead/page ≈ copy WCET)
  - *Dedicate colours* for shared memory
    - ➡ wastes space (e.g. only 16 colours on typical ARM boards)
    - ➡ targeted L2 flush via *prefetching*, a'la [Ge et al. 2019]
      - still leaks if buffer > lines per cache set, due to "random" eviction!

Varun Sethu
BE Thesis



Takeaways:
- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

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 L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!
    - ➡ **slow** (~19-34% overhead/page ≈ copy WCET)
  - *Dedicate colours* for shared memory
    - ➡ wastes space (e.g. only 16 colours on typical ARM boards)
    - ➡ targeted L2 flush via *prefetching*, a'la [Ge et al. 2019]
      - still leaks if buffer > lines per cache set, due to "random" eviction!

Varun Sethu
BE Thesis

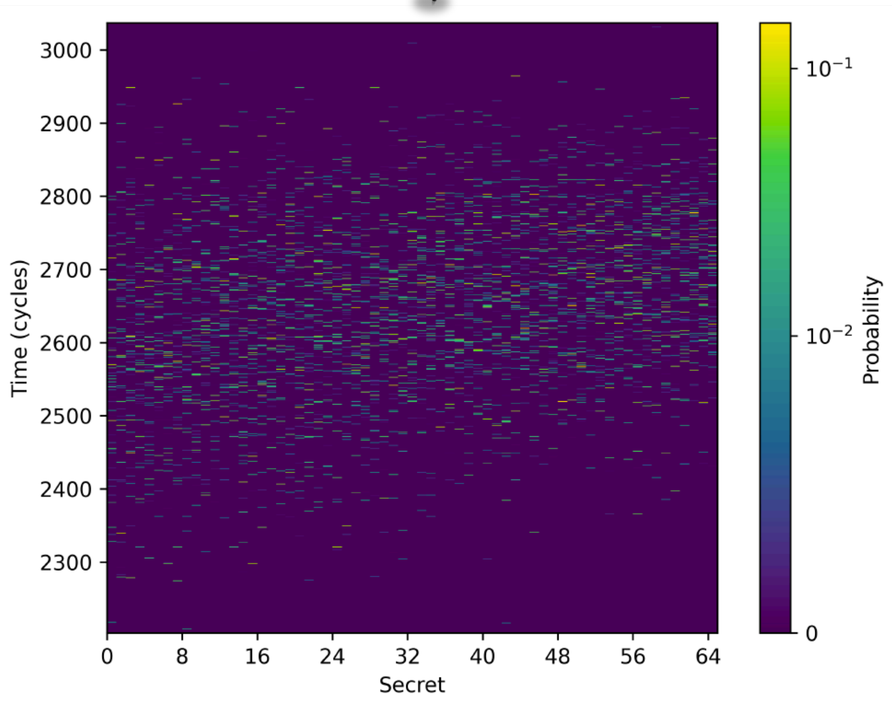


Figure 8.10: First-time use with longer eviction chains. $\mathscr{M} = 0.1179$. $\mathscr{M}_0 = 0.0023$.

Time-protected communications in seL4

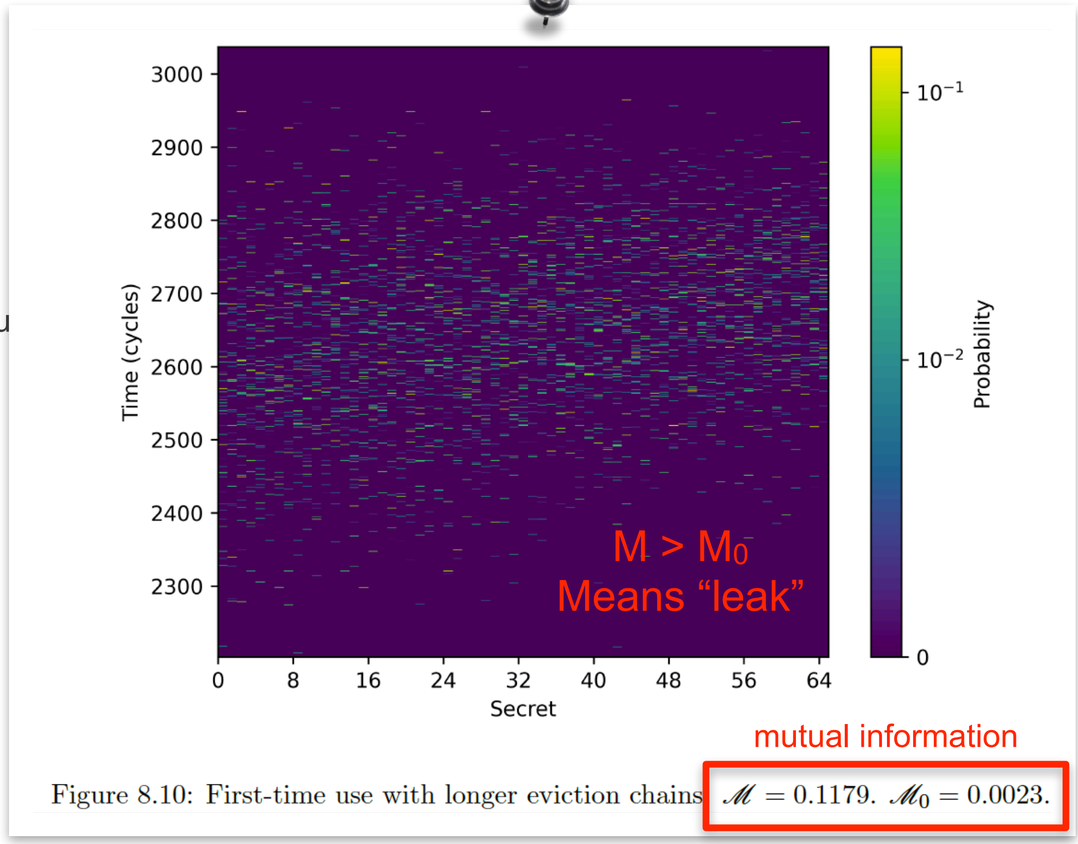
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!
    - ➡ **slow** (~19-34% overhead/page ≈ copy WCET)
  - *Dedicate colours* for shared memory
    - ➡ wastes space (e.g. only 16 colours on typical ARM boards)
    - ➡ targeted L2 flush via *prefetching*, a'la [Ge et al. 2019]
      - still leaks if buffer > lines per cache set, due to "random" eviction!

Varun Sethu
BE Thesis



Time-protected communications in seL4

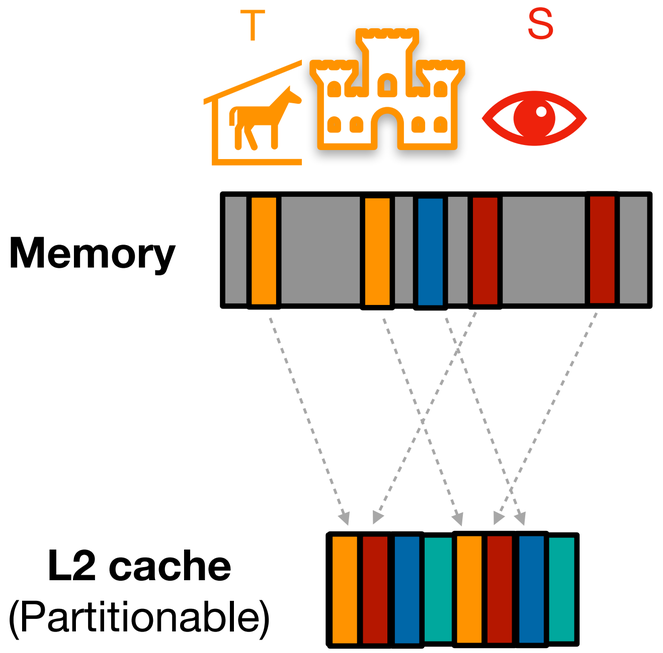
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!
    - ➡ **slow** (~19-34% overhead/page ≈ copy WCET)
  - *Dedicate colours* for shared memory
    - ➡ wastes space (e.g. only 16 colours on typical ARM boards)
    - ➡ targeted L2 flush via *prefetching,* a'la [Ge et al. 2019]
      - still leaks if buffer > lines per cache set, due to "random" eviction!

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

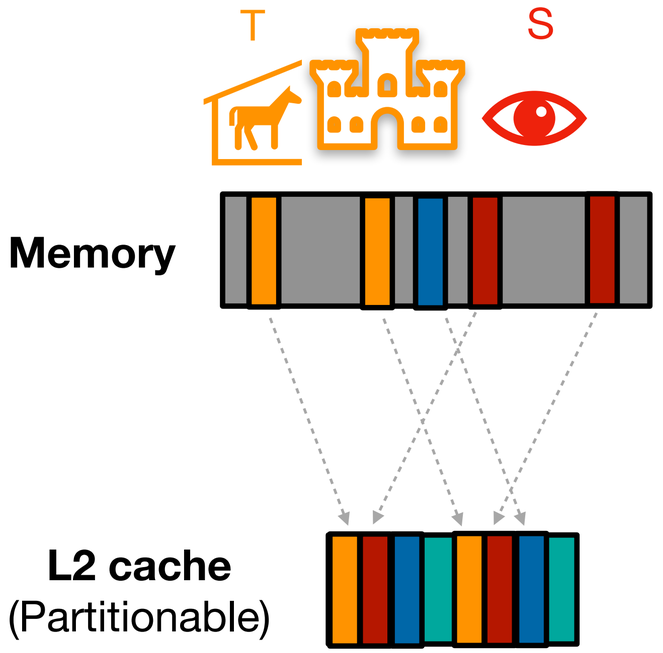
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!
    - ➡ **slow** (~19-34% overhead/page ≈ copy WCET)
  - *Dedicate colours* for shared memory
    - ➡ wastes space (e.g. only 16 colours on typical ARM boards)
    - ➡ targeted L2 flush via *prefetching,* a'la [Ge et al. 2019]
      - still leaks if buffer > lines per cache set, due to "random" eviction!
      - *infeasible to verify* w/o knowing eviction policy [Buckley et al. 2023]

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

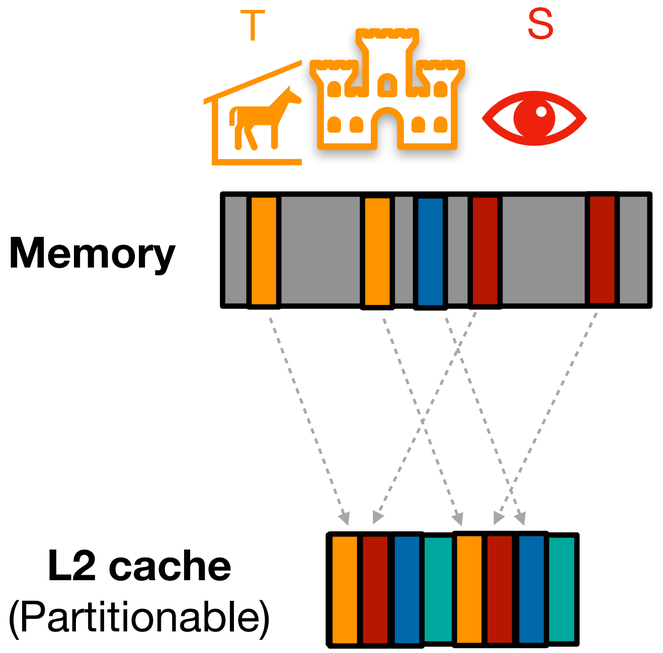
seL4 on RISC-V

D Implement time-protected cross-domain communications

E Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!
    - ➡ **slow** (~19-34% overhead/page ≈ copy WCET)
  - *Dedicate colours* for shared memory
    - ➡ wastes space (e.g. only 16 colours on typical ARM boards)
    - ➡ targeted L2 flush via *prefetching,* a'la [Ge et al. 2019]
      - still leaks if buffer > lines per cache set, due to "random" eviction!
      - *infeasible to verify* w/o knowing eviction policy [Buckley et al. 2023]

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

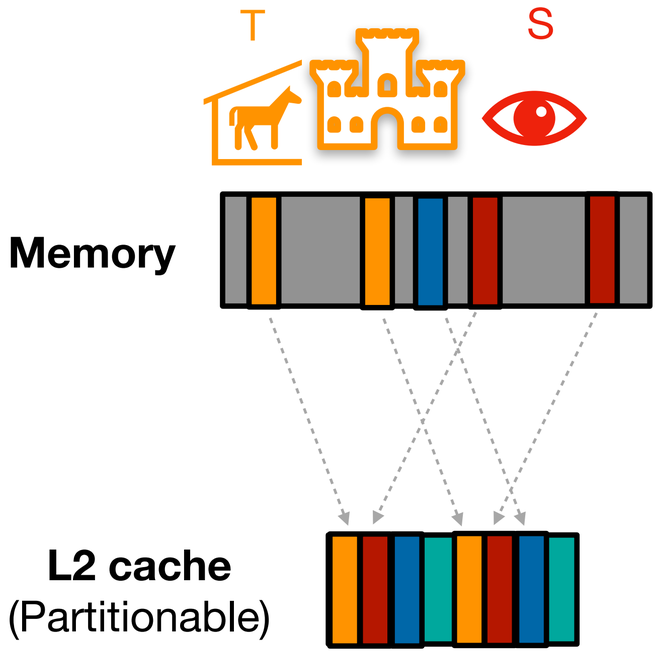
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between *non-shared memory*
    - ➡ no flush needed, eliminates channels!
    - ➡ **slow** (~19-34% overhead/page ≈ copy WCET)
  - *Dedicate colours* for shared memory
    - ➡ wastes space (e.g. only 16 colours on typical ARM boards)
    - ➡ targeted L2 flush via *prefetching,* a'la [Ge et al. 2019]
      - still leaks if buffer > lines per cache set, due to "random" eviction!
      - *infeasible to verify* w/o knowing eviction policy [Buckley et al. 2023]

Varun Sethu
BE Thesis



Takeaways:

- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

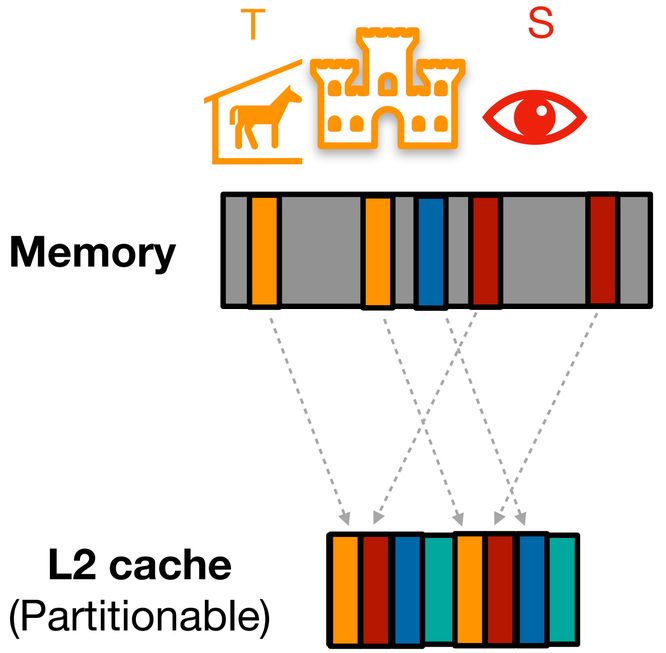
seL4 on RISC-V

D Implement time-protected cross-domain communications

E Verify time-protected cross-domain communications

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

Varun Sethu
BE Thesis



- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

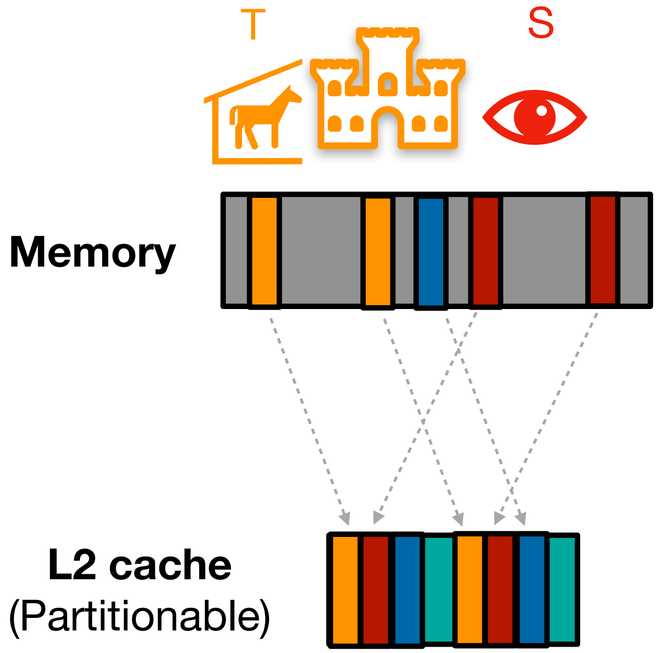
seL4 on RISC-V

D Implement time-protected cross-domain communications

E Verify time-protected cross-domain communications

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

Varun Sethu
BE Thesis



- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

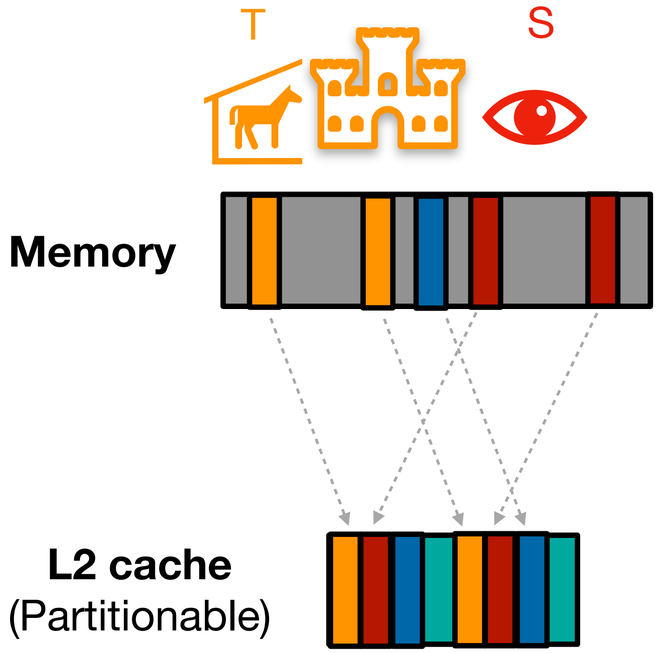
seL4 on RISC-V

D — Implement time-protected cross-domain communications

E — Verify time-protected cross-domain communications

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

Varun Sethu
BE Thesis



- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

seL4 on RISC-V

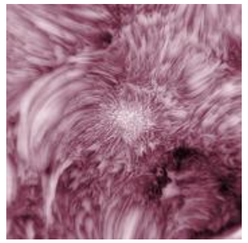
D Implement time-protected cross-domain communications ✓

E Verify time-protected cross-domain communications 🎓

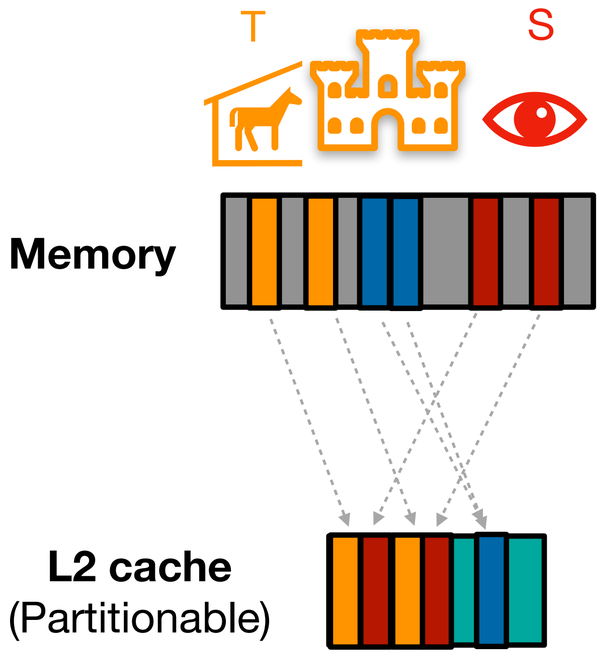
- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between non-shared memory (works, but slow)
  - Dedicate colours for shared memory (wastes space)
    - Targeted L2 "flush" via prefetching (still leaks; and *infeasible to verify* w/o knowing eviction policy)

Varun Sethu
BE Thesis

- Now we're using *dynamically partitionable last-level cache* (DPLLC) i.e. dedicated HW support on RISC-V Cheshire



Julia Vassiliki



- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

seL4 on RISC-V

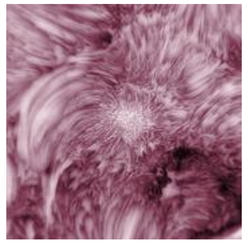
D Implement time-protected cross-domain communications ✓

E Verify time-protected cross-domain communications 🎓

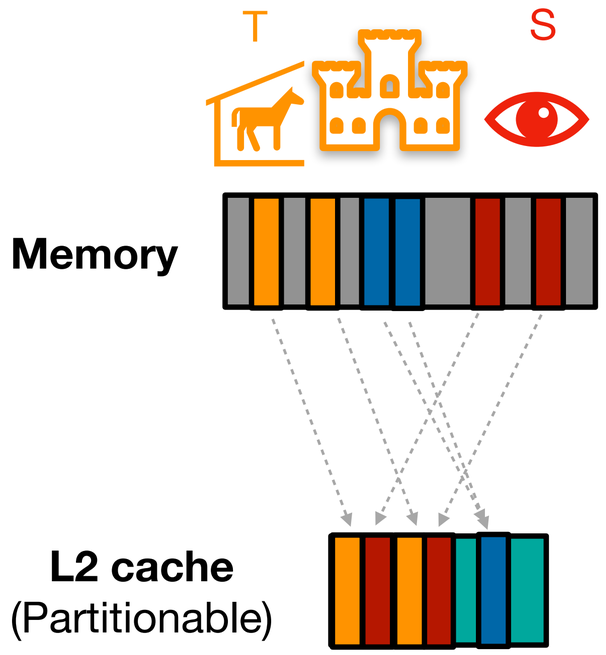
- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between non-shared memory (works, but slow)
  - Dedicate colours for shared memory (wastes space)
    - Targeted L2 "flush" via prefetching (still leaks; and *infeasible to verify* w/o knowing eviction policy)

Varun Sethu
BE Thesis

- Now we're using *dynamically partitionable last-level cache* (DPLLC) i.e. dedicated HW support on RISC-V Cheshire
  - Supports targeted flush of partitions



Julia Vassiliki



- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

seL4 on RISC-V

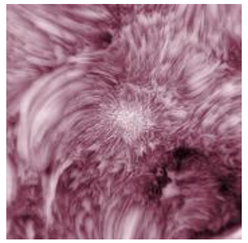
D Implement time-protected cross-domain communications ✓

E Verify time-protected cross-domain communications

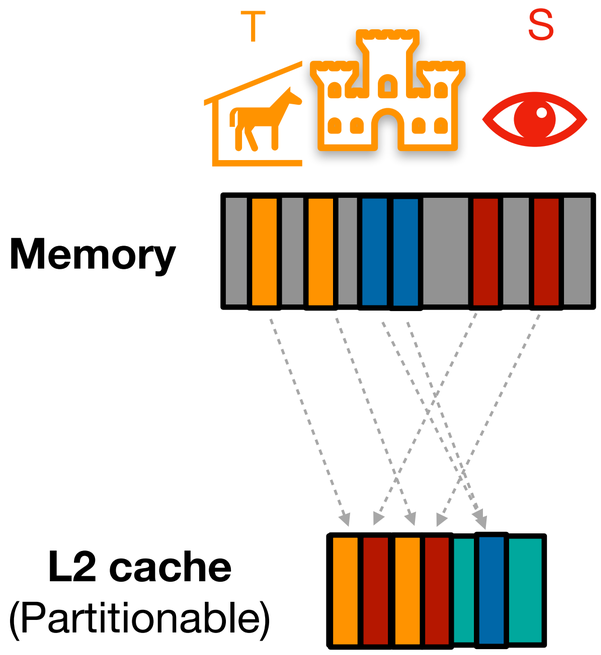
- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between non-shared memory (works, but slow)
  - Dedicate colours for shared memory (wastes space)
    - Targeted L2 "flush" via prefetching (still leaks; and *infeasible to verify* w/o knowing eviction policy)

Varun Sethu
BE Thesis

- Now we're using *dynamically partitionable last-level cache* (DPLLC) i.e. dedicated HW support on RISC-V Cheshire
  - Supports targeted flush of partitions
  - Now adding needed time padding on flush



Julia Vassiliki



- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

Time-protected communications in seL4

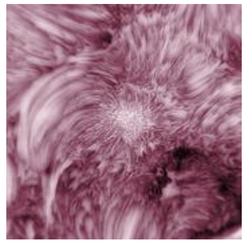
seL4 on RISC-V

D — Implement time-protected cross-domain communications ✓

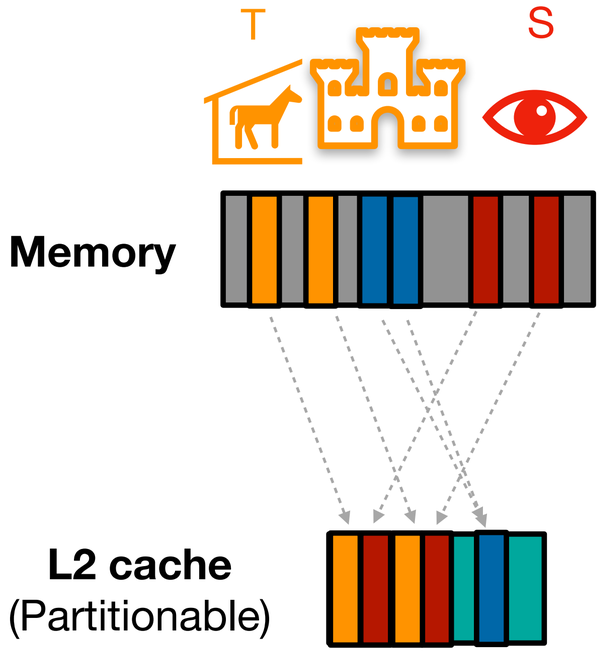
E — Verify time-protected cross-domain communications

- [Sethu 2025] comprehensively evaluated "legacy" implementation options on seL4, with L2 partitioned via *cache colouring*:
  - Copy between non-shared memory (works, but slow)
  - Dedicate colours for shared memory (wastes space)
    - Targeted L2 "flush" via prefetching (still leaks; and *infeasible to verify* w/o knowing eviction 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
  - Now adding needed time padding on flush

Varun Sethu
BE Thesis



Julia Vassiliki



- T and S cannot share memory, or
- We need a *targeted flush* for the L2 cache

What's next?

seL4 on RISC-V

Implement time-protected cross-domain communications

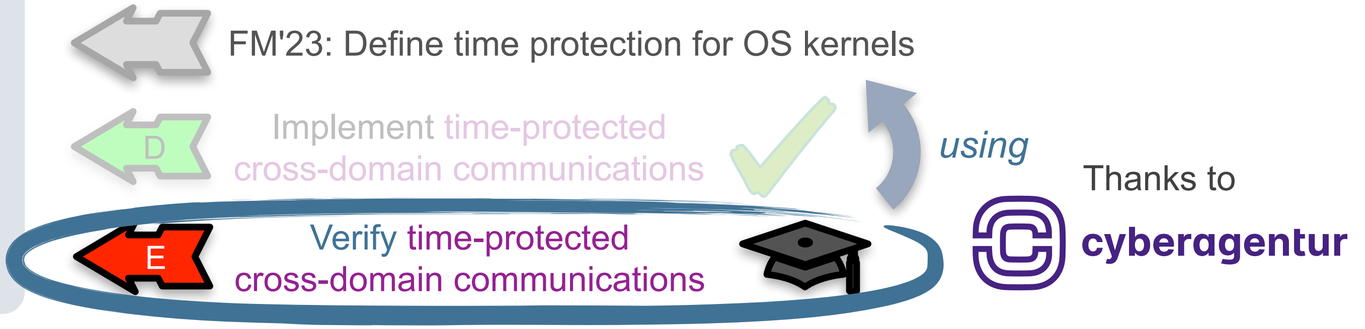
Verify time-protected cross-domain communications

Thanks to

cyberagentur

What's next? Verification

seL4 on RISC-V



What's next? Verification

seL4 on RISC-V

FM'23: Define time protection for OS kernels

D

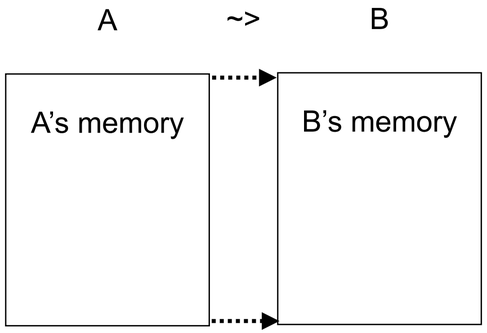
Implement time-protected cross-domain communications

using

Thanks to **cyberagentur**

E

Verify time-protected cross-domain communications



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

What's next? Verification

seL4 on RISC-V

FM'23: Define time protection for OS kernels

D

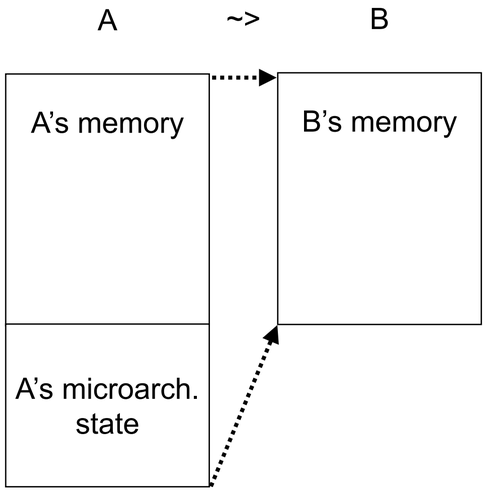
Implement time-protected cross-domain communications

using

Thanks to **cyberagentur**

E

Verify time-protected cross-domain communications



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

What's next? Verification

seL4 on RISC-V

FM'23: Define time protection for OS kernels

D

Implement time-protected cross-domain communications

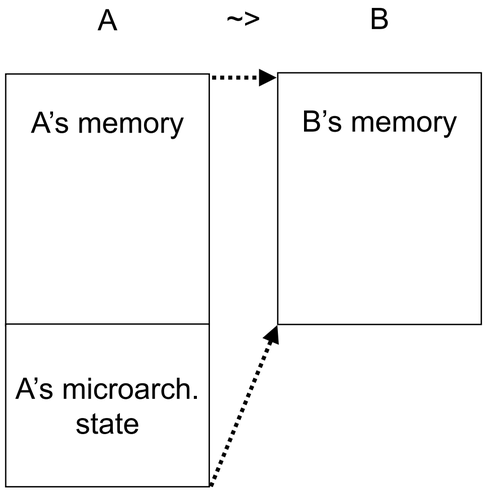
using

E

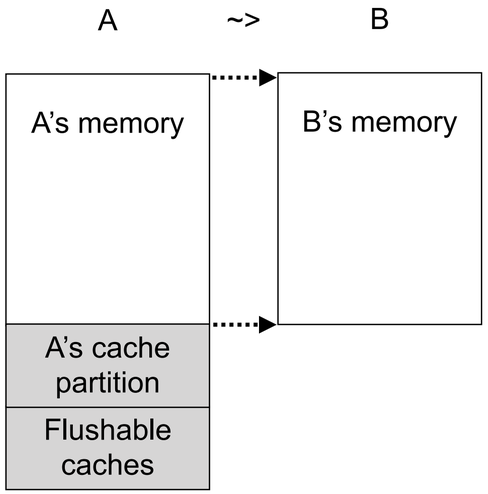
Verify time-protected cross-domain communications

Thanks to Australian Research Council funding

Thanks to cyberagentur



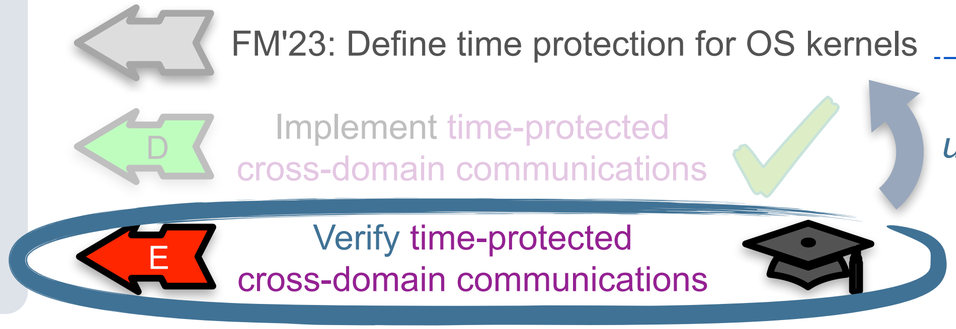
From prior seL4 infoflow proofs [Murray et al. 2012, 2013]: *"<u>all</u> or <u>nothing</u>" policies*



Need *spatial precision* to *allow some flows* but *exclude others*

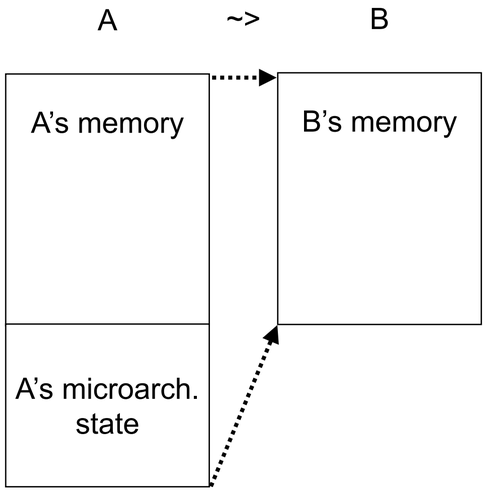
What's next? Verification

seL4 on RISC-V

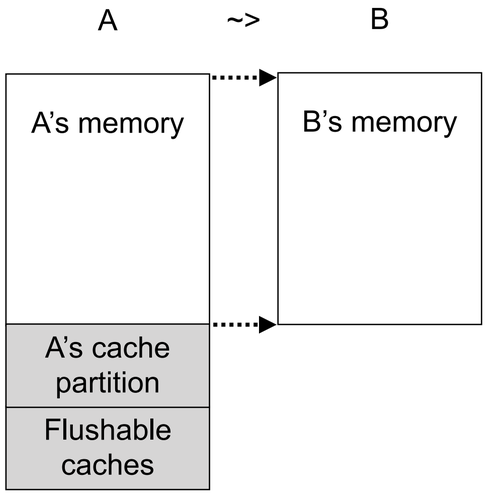


Thanks to Australian Research Council funding

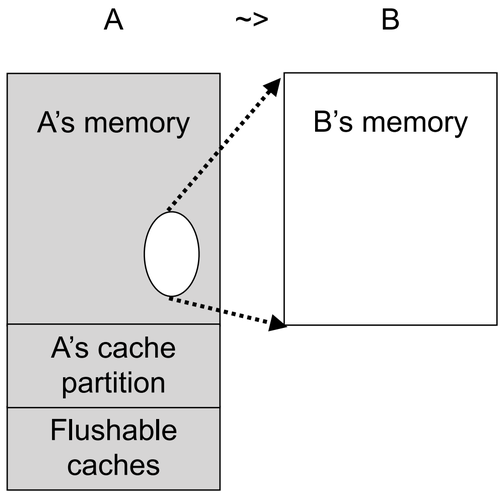
Thanks to cyberagentur



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



Need *spatial precision* to *allow some flows* but *exclude others*



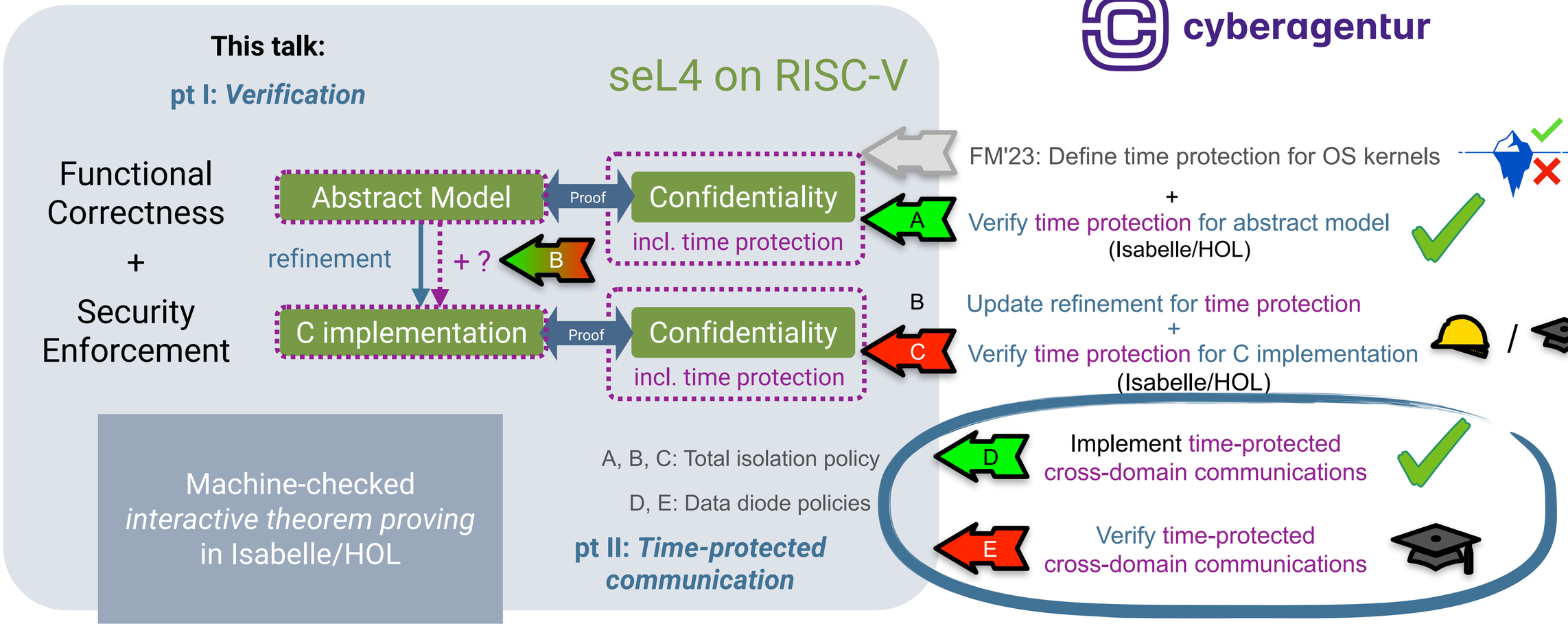
Our FM'23 paper [Sison et al. 2023] allows infoflow policies with arbitrary *spatial* and *temporal* precision

Time protection verification for seL4

The status today:

Thanks to funding from

cyberagentur



Time protection verification for seL4

The status today:

Thank you!
Q&A

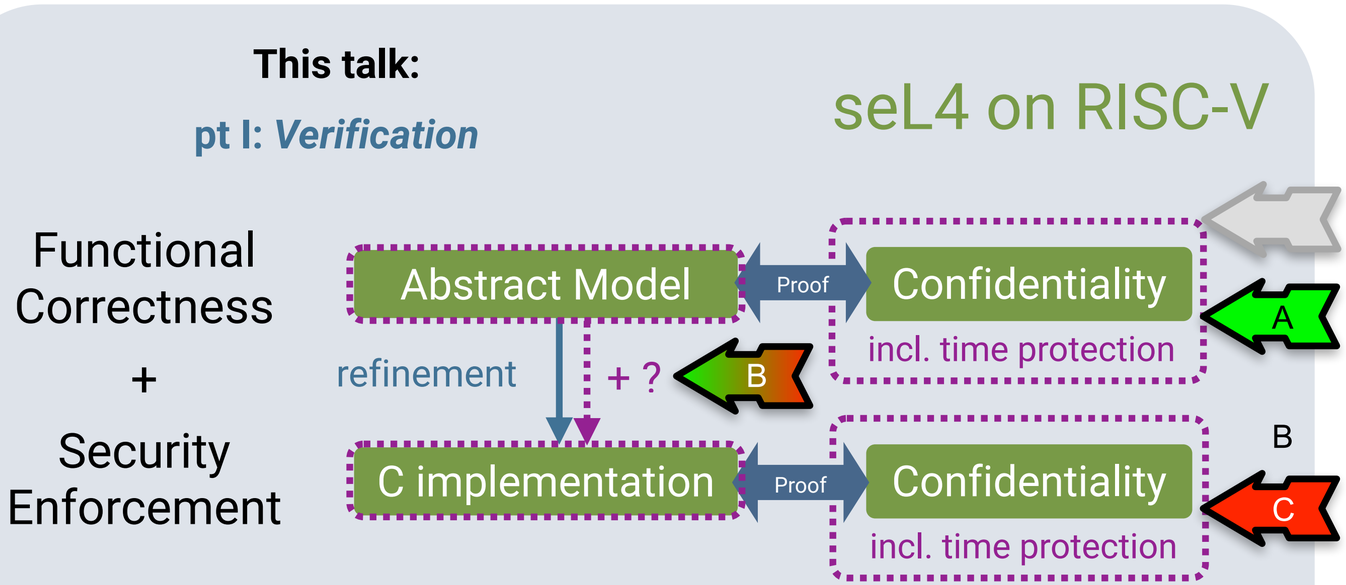
Thanks to funding from

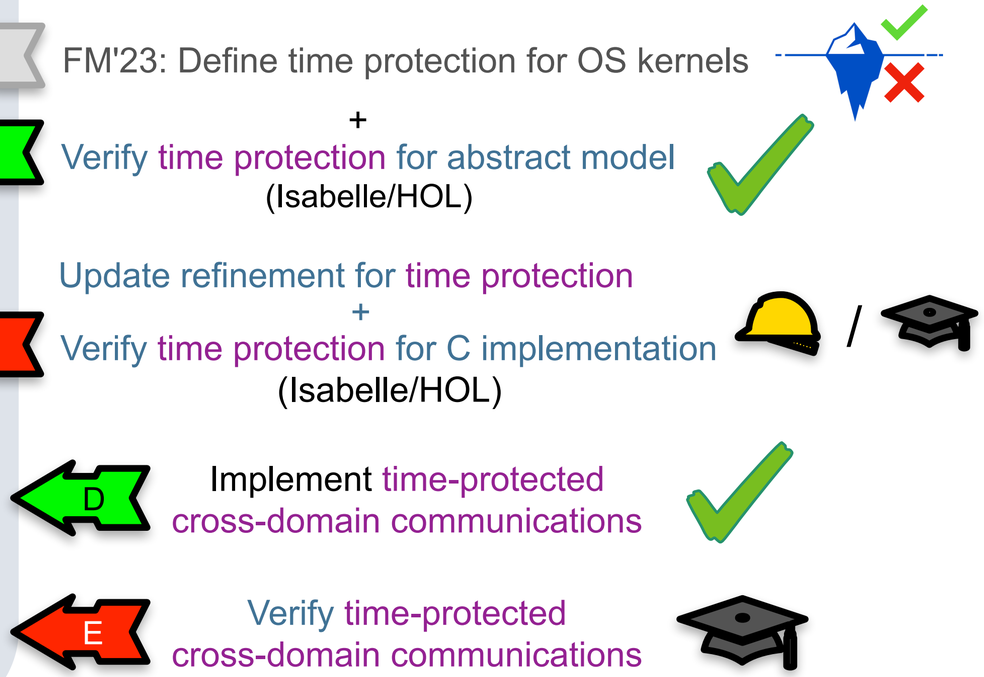
cyberagentur

This talk:

pt I: *Verification*

seL4 on RISC-V





UNSW SYDNEY