School of Computer Science & Engineering

Trustworthy Systems Group

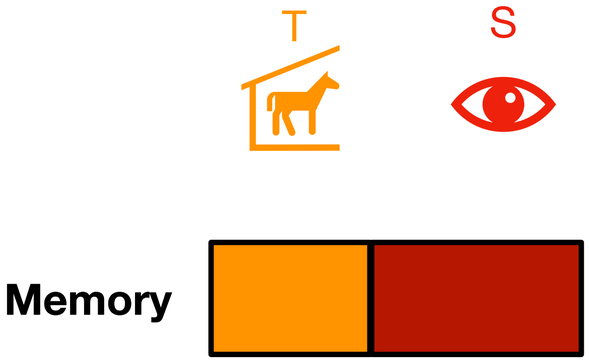
Clearing roadblocks to Time Protection verification for seL4

Dr Rob Sison

Senior Research Associate, UNSW Sydney

r.sison@unsw.edu.au

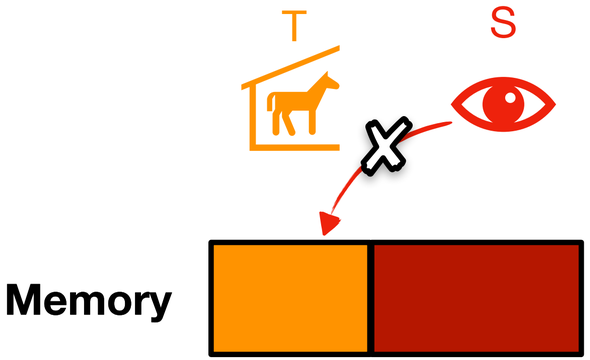
What is Time Protection?



© Rob Sison 2026, CC BY 4.0

What is Time Protection?

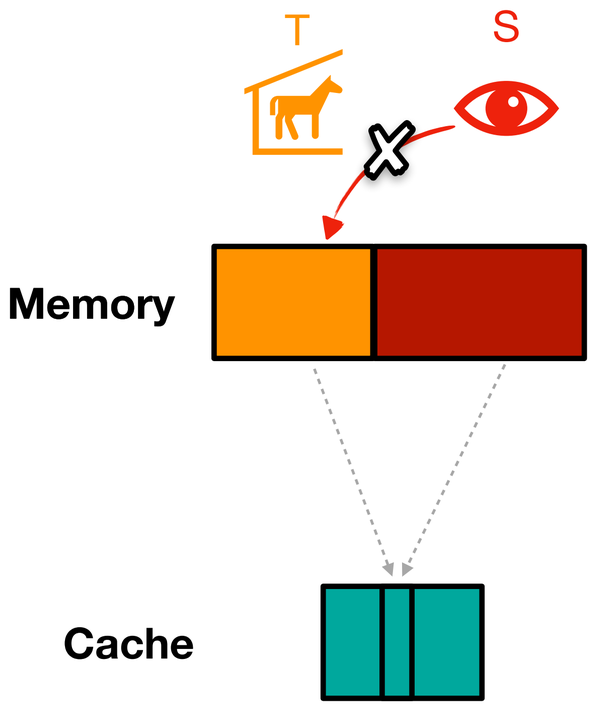
- OSes typically implement *memory protection.*



© Rob Sison 2026, CC BY 4.0

What is Time Protection?

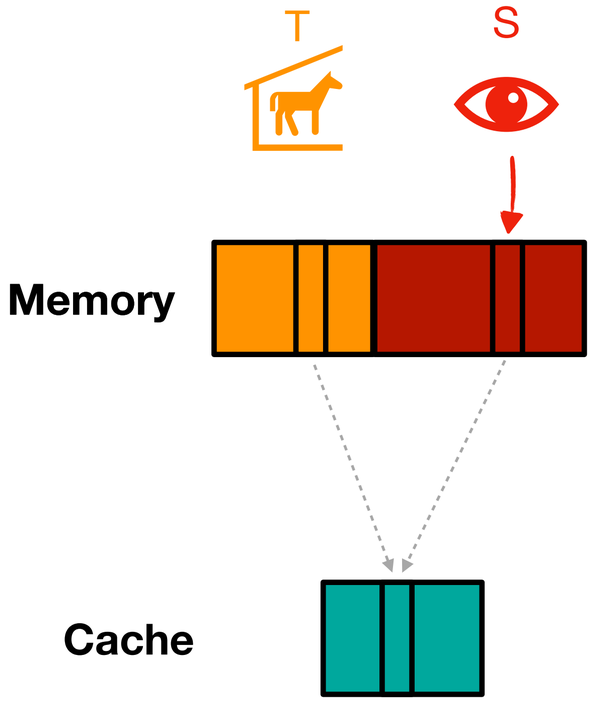
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.



© Rob Sison 2026, CC BY 4.0

What is Time Protection?

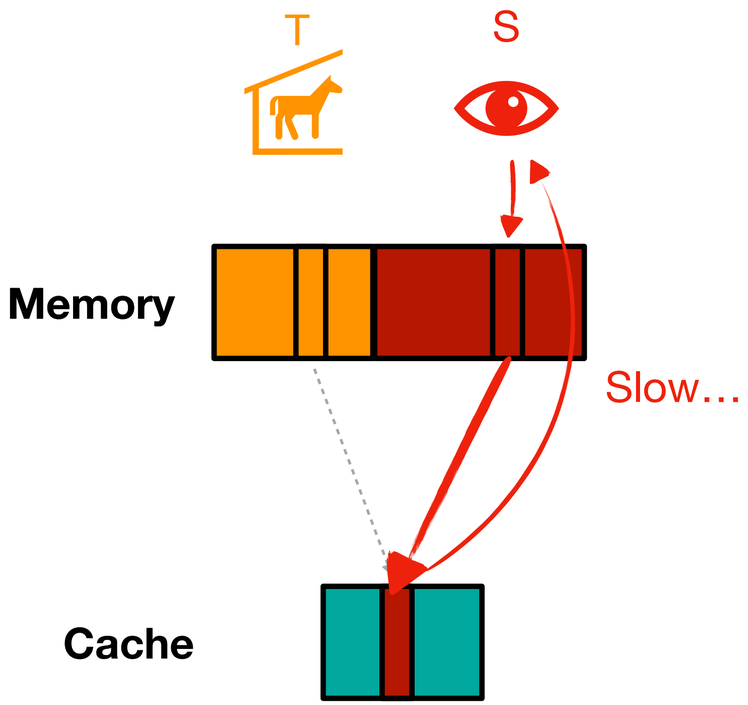
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.



© Rob Sison 2026, CC BY 4.0

What is Time Protection?

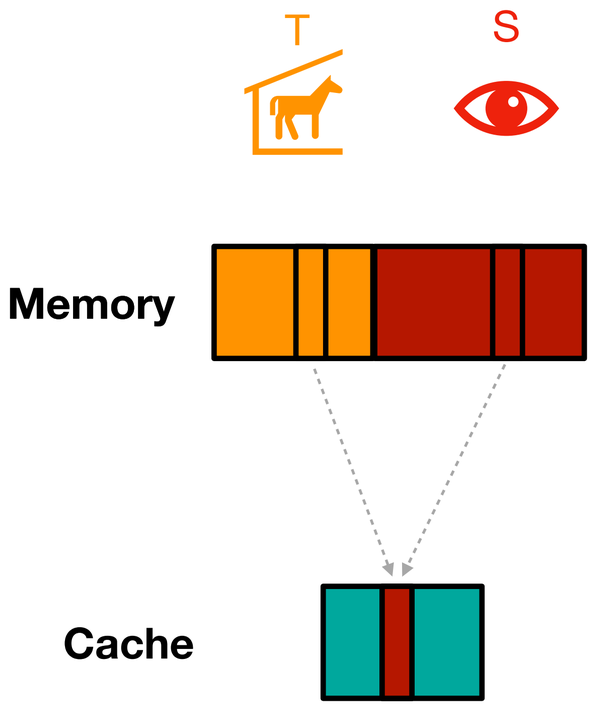
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.



© Rob Sison 2026, CC BY 4.0

What is Time Protection?

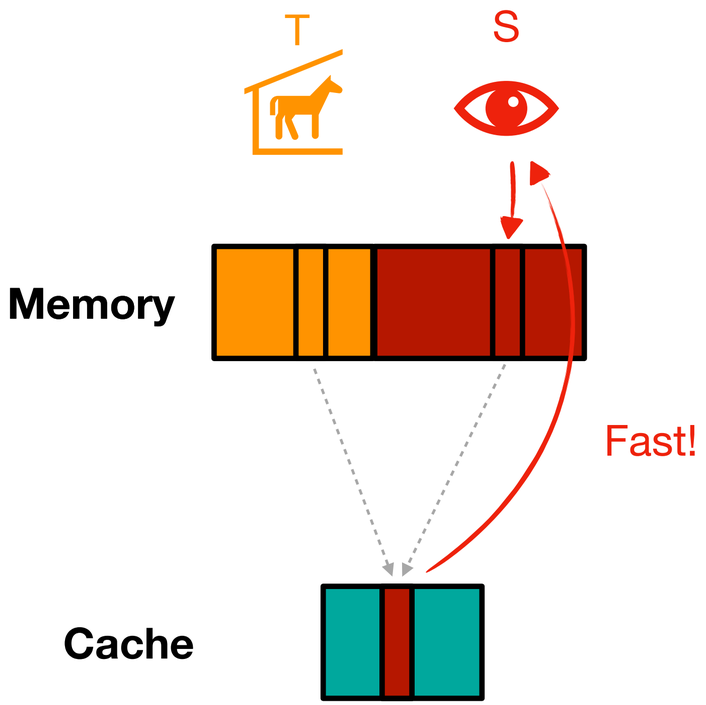
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.



© Rob Sison 2026, CC BY 4.0

What is Time Protection?

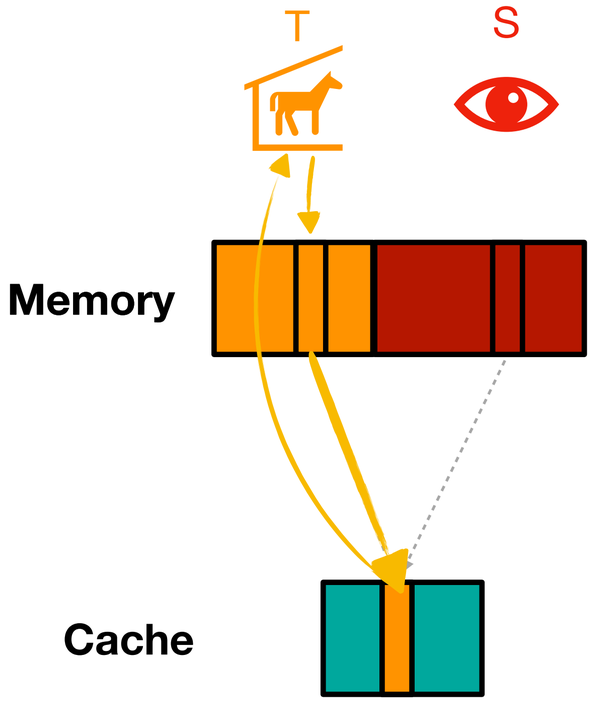
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.



© Rob Sison 2026, CC BY 4.0

What is Time Protection?

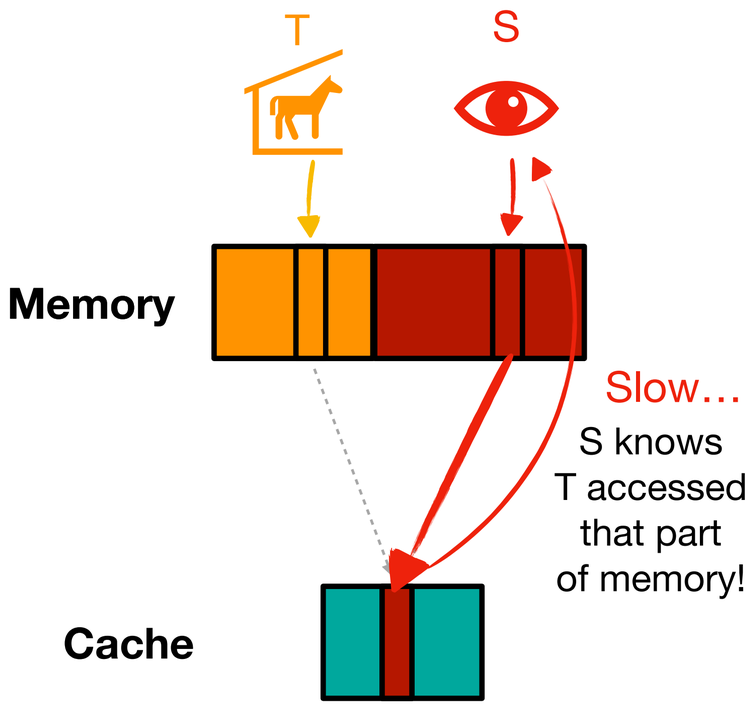
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.



© Rob Sison 2026, CC BY 4.0

What is Time Protection?

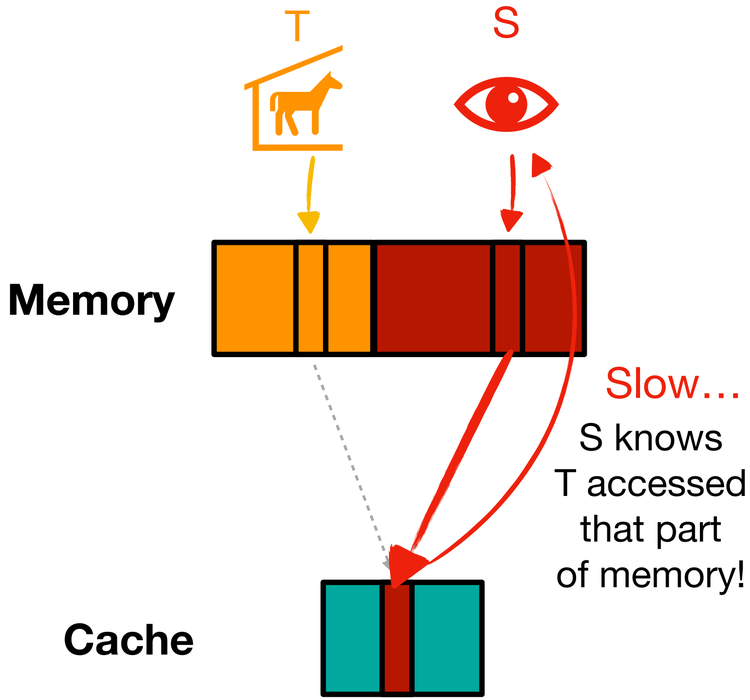
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.



© Rob Sison 2026, CC BY 4.0

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

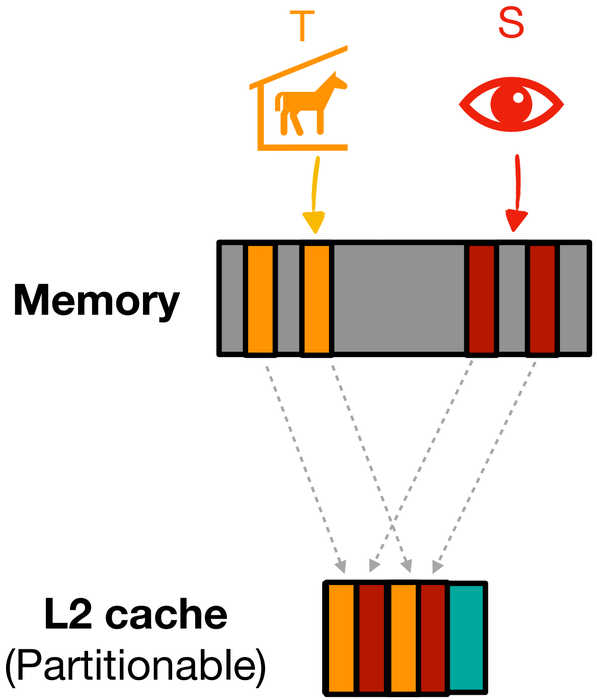
with
Yuval Yarom (U. Adelaide)
Tom Chothia (U. Birmingham)

© Rob Sison 2026, CC BY 4.0

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)

© Rob Sison 2026, CC BY 4.0

What is Time Protection?

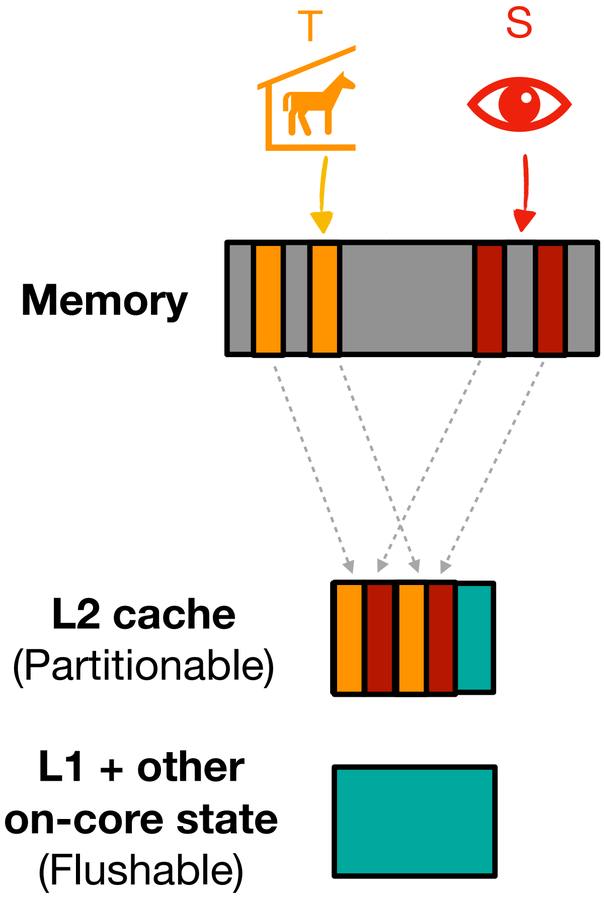
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.

- 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)



© Rob Sison 2026, CC BY 4.0

What is Time Protection?

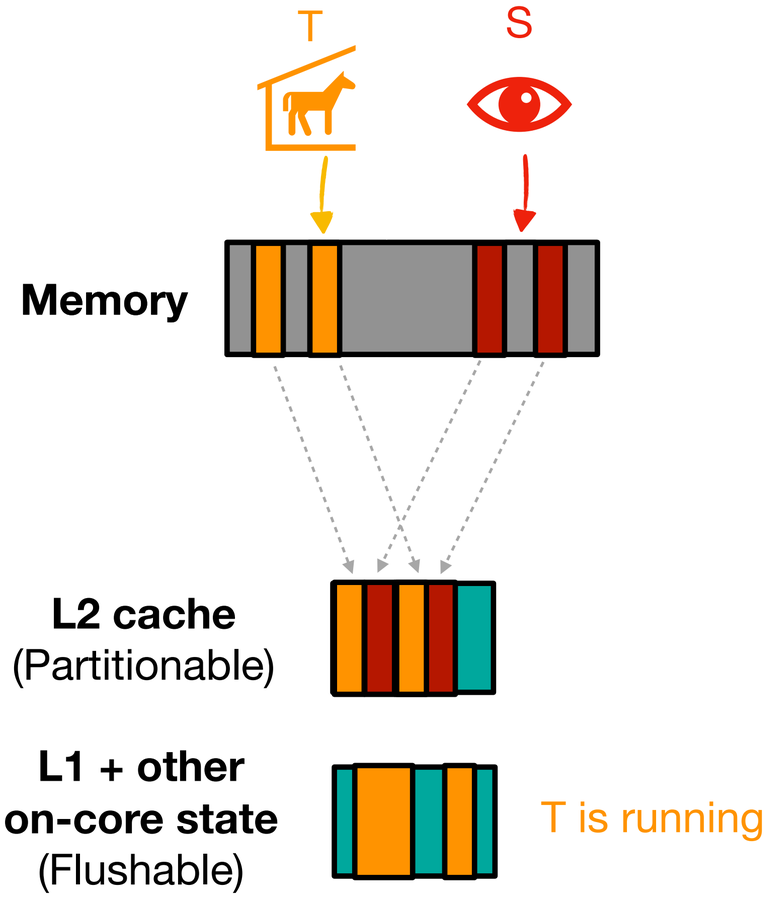
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.

- 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)



© Rob Sison 2026, CC BY 4.0

What is Time Protection?

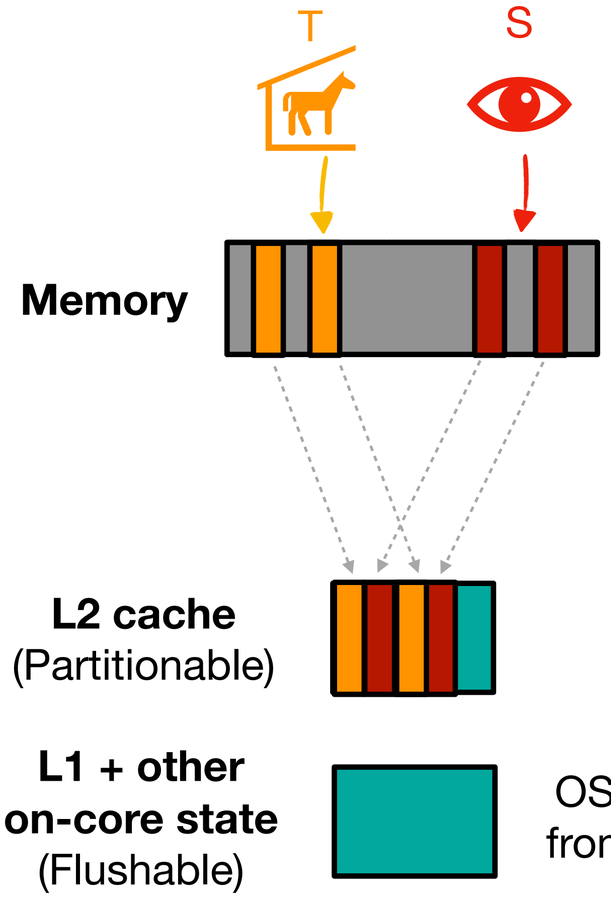
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.

- 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.

© Rob Sison 2026, CC BY 4.0

What is Time Protection?

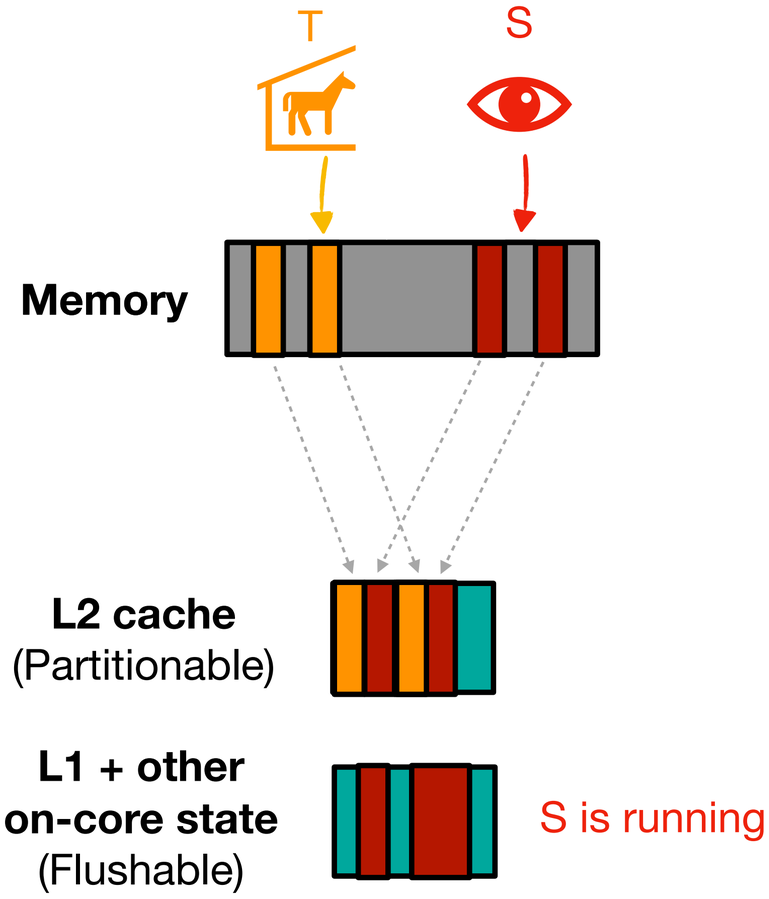
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.

- 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.

© Rob Sison 2026, CC BY 4.0

What is Time Protection?

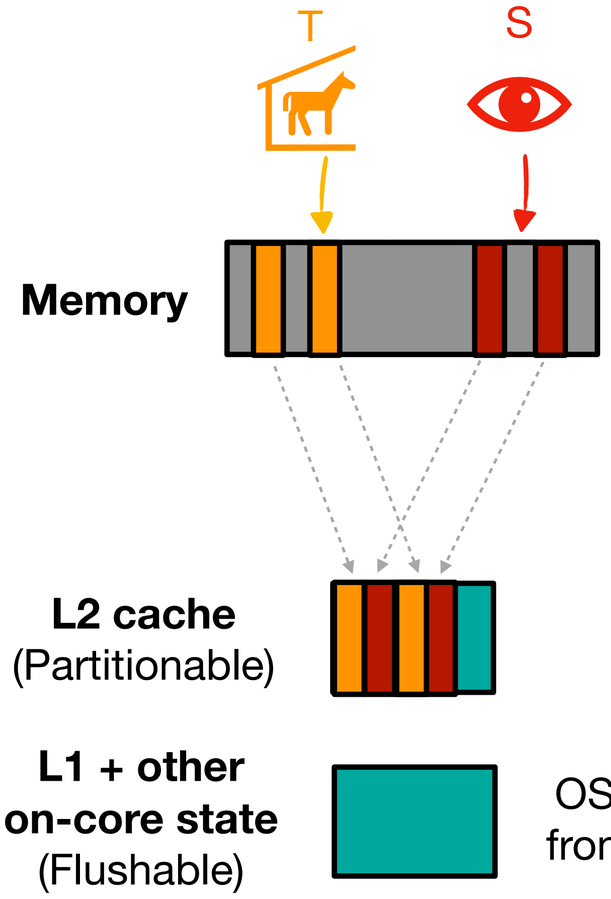
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.

- 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.

© Rob Sison 2026, CC BY 4.0

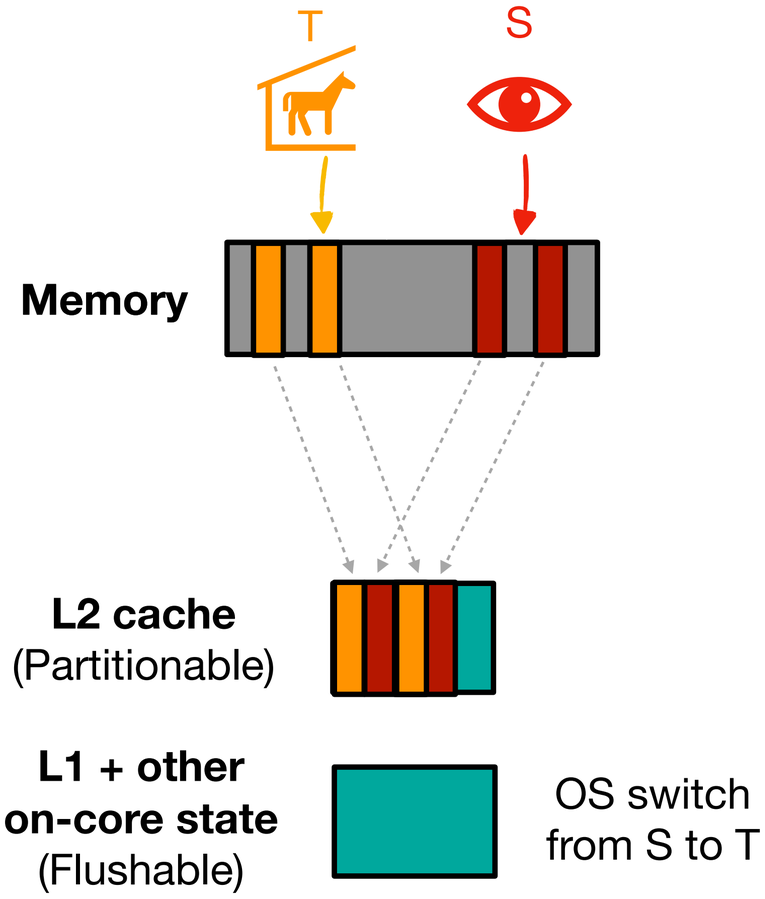
The seL4 kernel and Time Protection

- 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
- 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.

© Rob Sison 2026, CC BY 4.0

The seL4 kernel and Time Protection

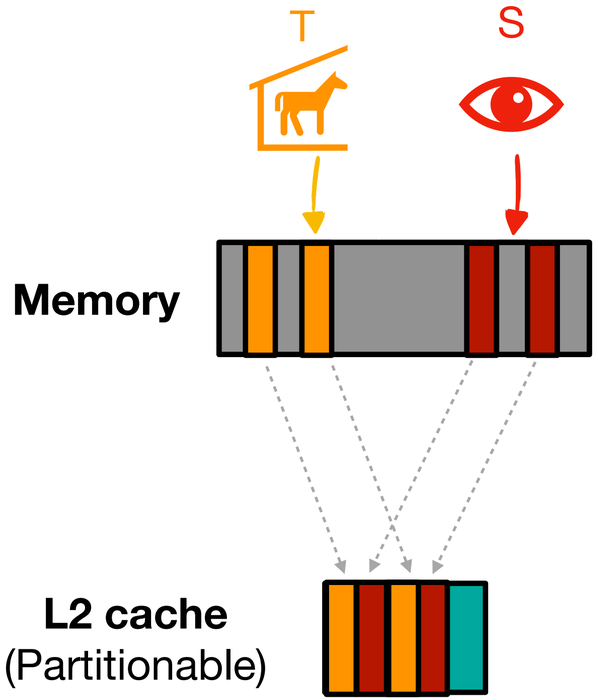
- To prevent these *timing channels*, OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- For the seL4 OS kernel:

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.

© Rob Sison 2026, CC BY 4.0

The seL4 kernel and Time Protection

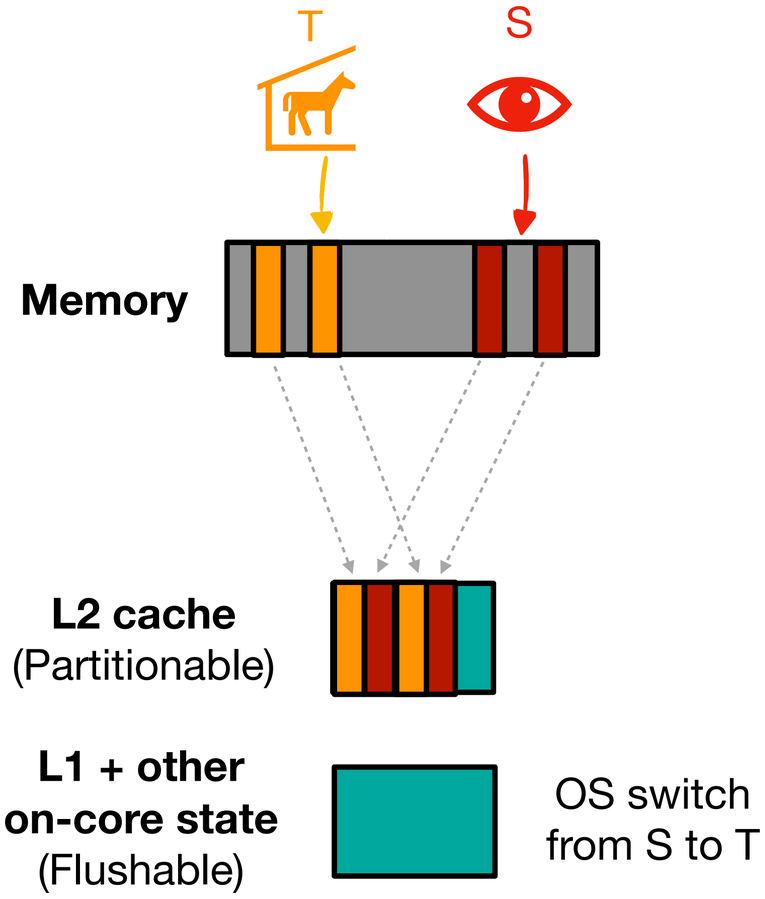
- To prevent these *timing channels*, OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- For the seL4 OS kernel:
  - Implemented, evaluated empirically on ARM, x86
    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.

© Rob Sison 2026, CC BY 4.0

The seL4 kernel and Time Protection

- 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

- 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.

© Rob Sison 2026, CC BY 4.0

The seL4 kernel and Time Protection

- 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

- 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:** 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.

© Rob Sison 2026, CC BY 4.0

How is seL4 verified?

- **This talk:** How do we verify Time Protection on seL4 works?
  - 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!

Thanks to funding from Australian Research Council

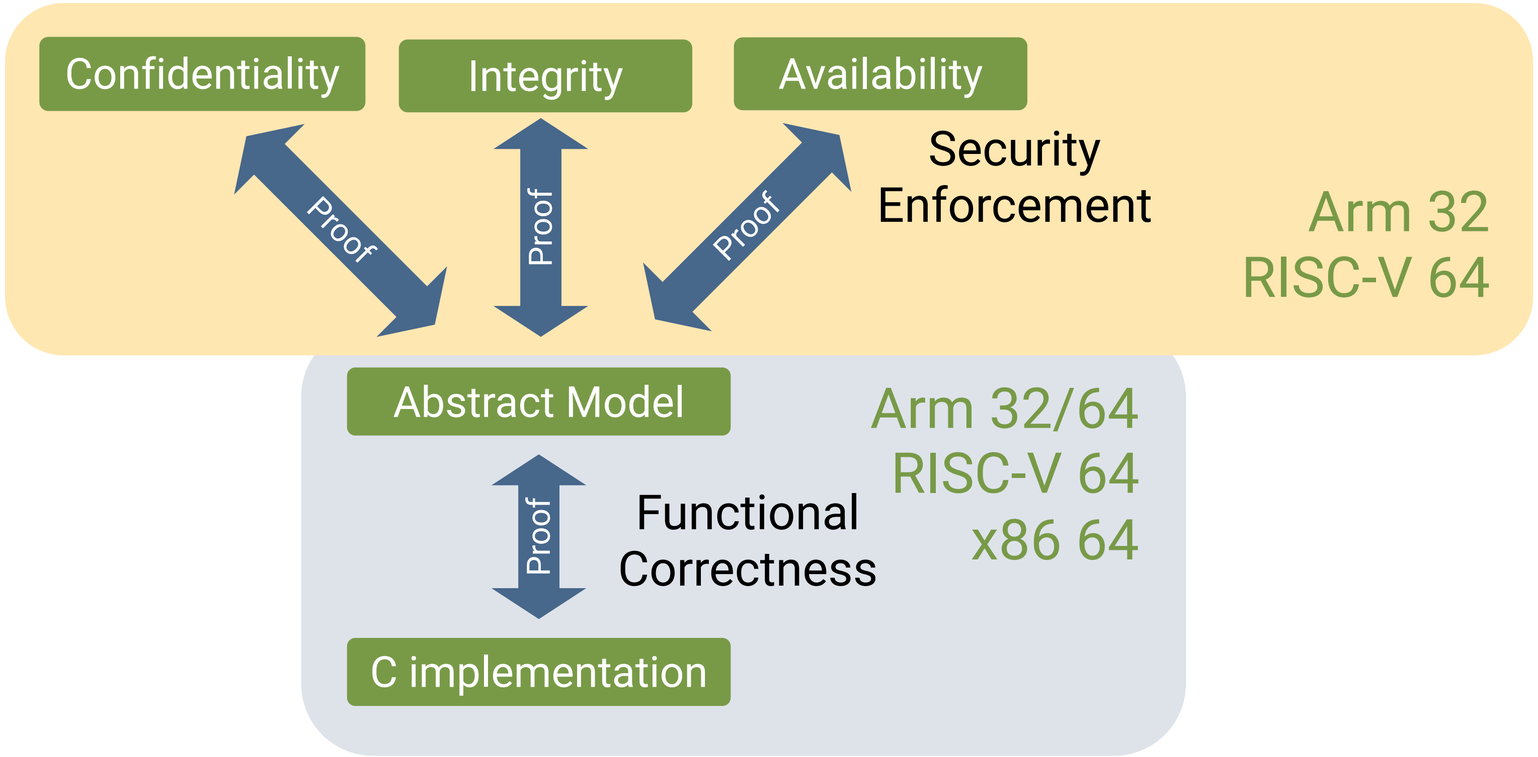
© Rob Sison 2026, CC BY 4.0

How is seL4 verified?

- **This talk:** How do we verify Time Protection on seL4 works?
  - 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])

Thanks to funding from Australian Research Council

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



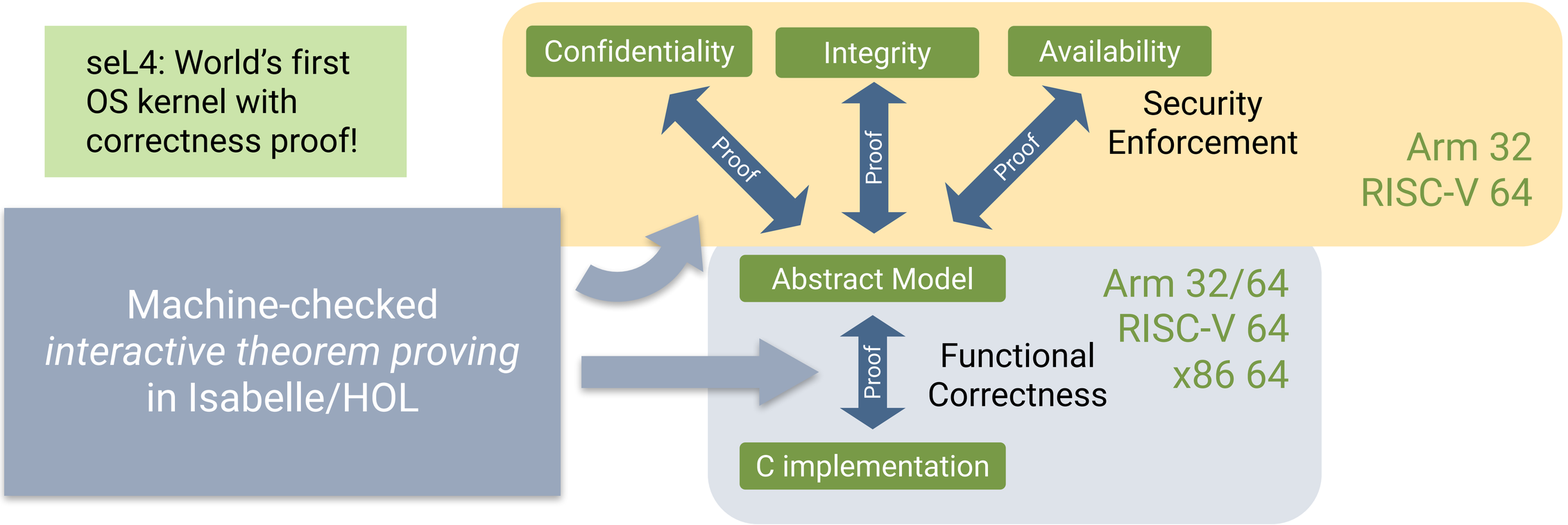
© Rob Sison 2026, CC BY 4.0

How is seL4 verified?

- **This talk:** How do we verify Time Protection on seL4 works?
  - 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])

Thanks to funding from Australian Research Council

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



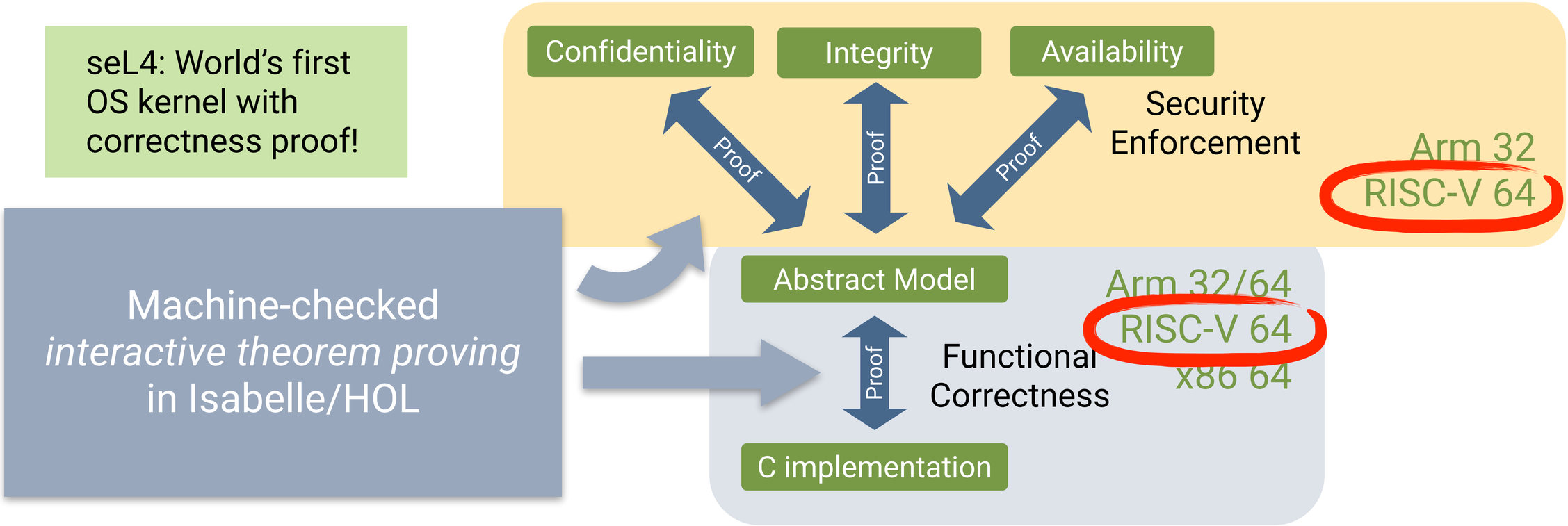
© Rob Sison 2026, CC BY 4.0

How is seL4 verified?

- **This talk:** How do we verify Time Protection on seL4 works?
  - 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])

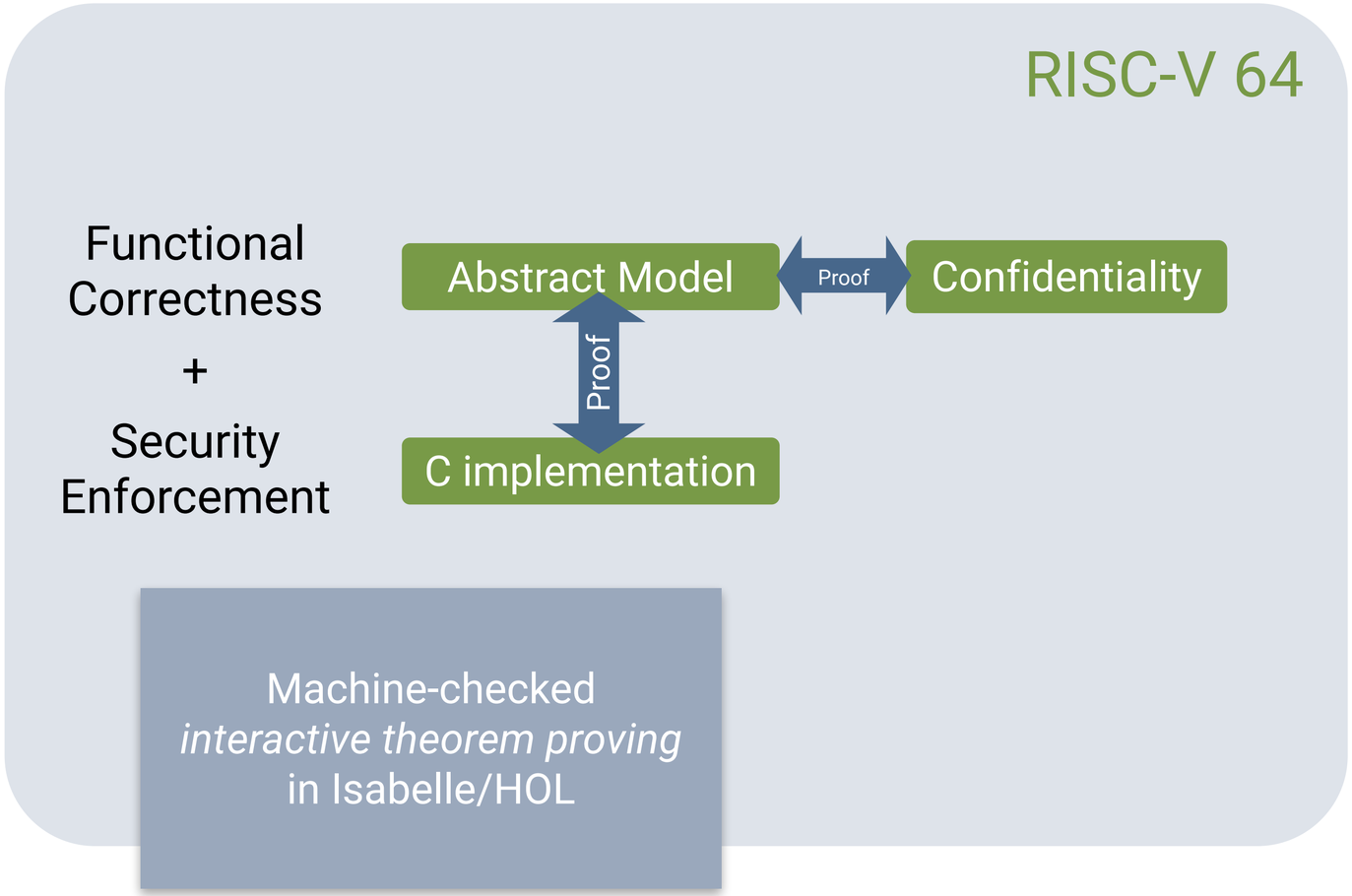
Thanks to funding from Australian Research Council

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



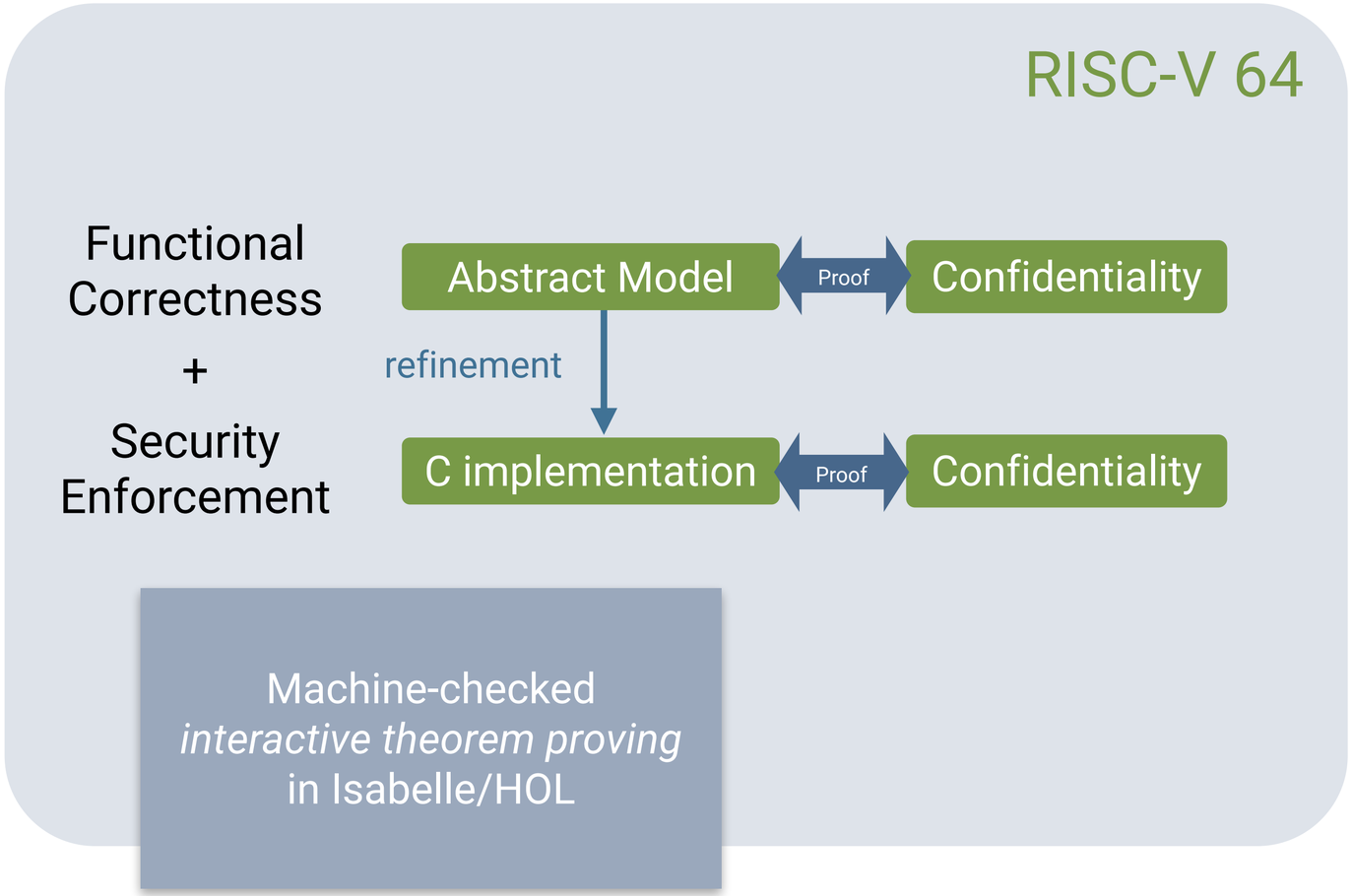
© Rob Sison 2026, CC BY 4.0

Time protection verification for seL4



© Rob Sison 2026, CC BY 4.0

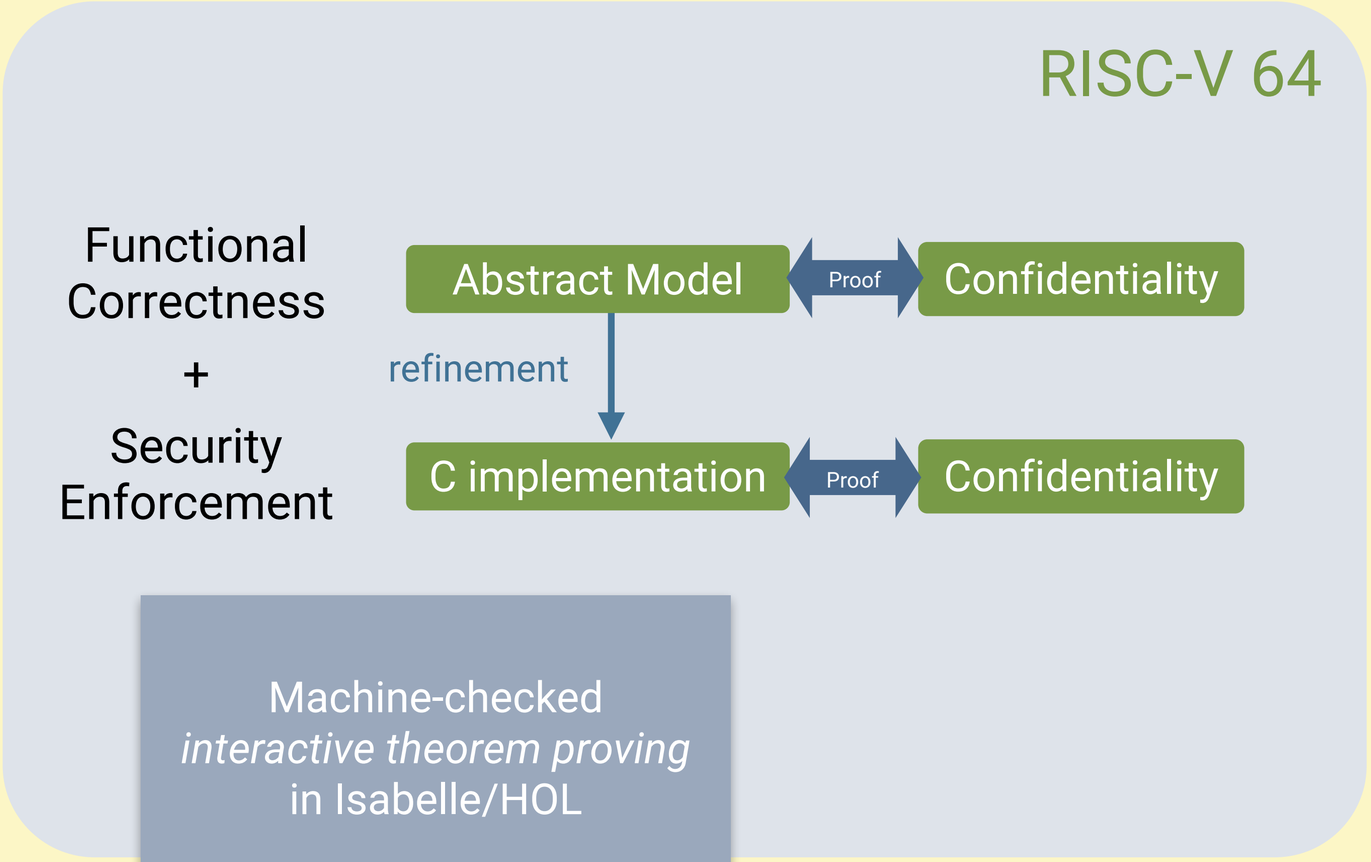
Time protection verification for seL4



© Rob Sison 2026, CC BY 4.0

Time protection verification for seL4

The plan 2 years ago… *(presented at FM-OZ'24)*

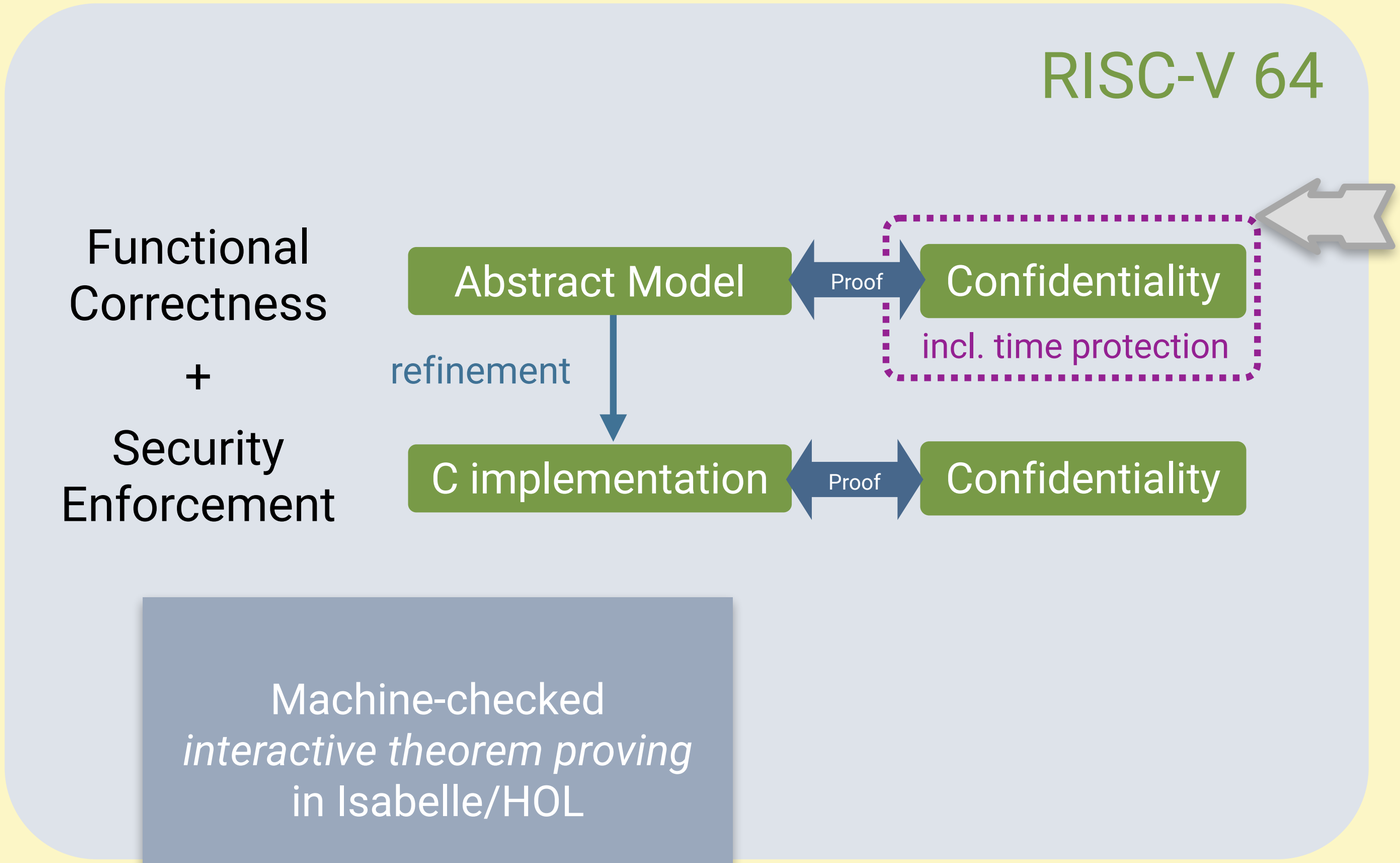


© Rob Sison 2026, CC BY 4.0

Time protection verification for seL4

The plan 2 years ago… *(presented at FM-OZ'24)*

Thanks to funding from
Australian Research Council



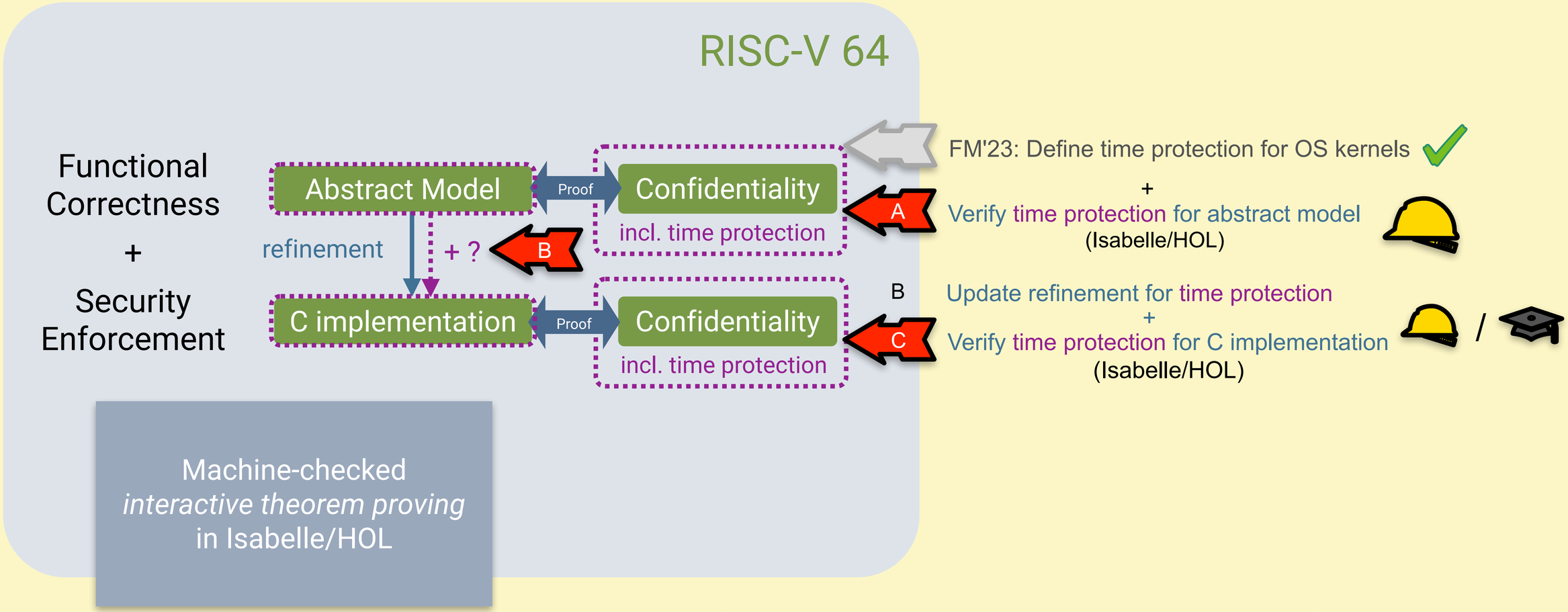
FM'23: Define time protection for OS kernels ✓

© Rob Sison 2026, CC BY 4.0

Time protection verification for seL4

The plan 2 years ago… *(presented at FM-OZ'24)*

Thanks to funding from
Australian Research Council

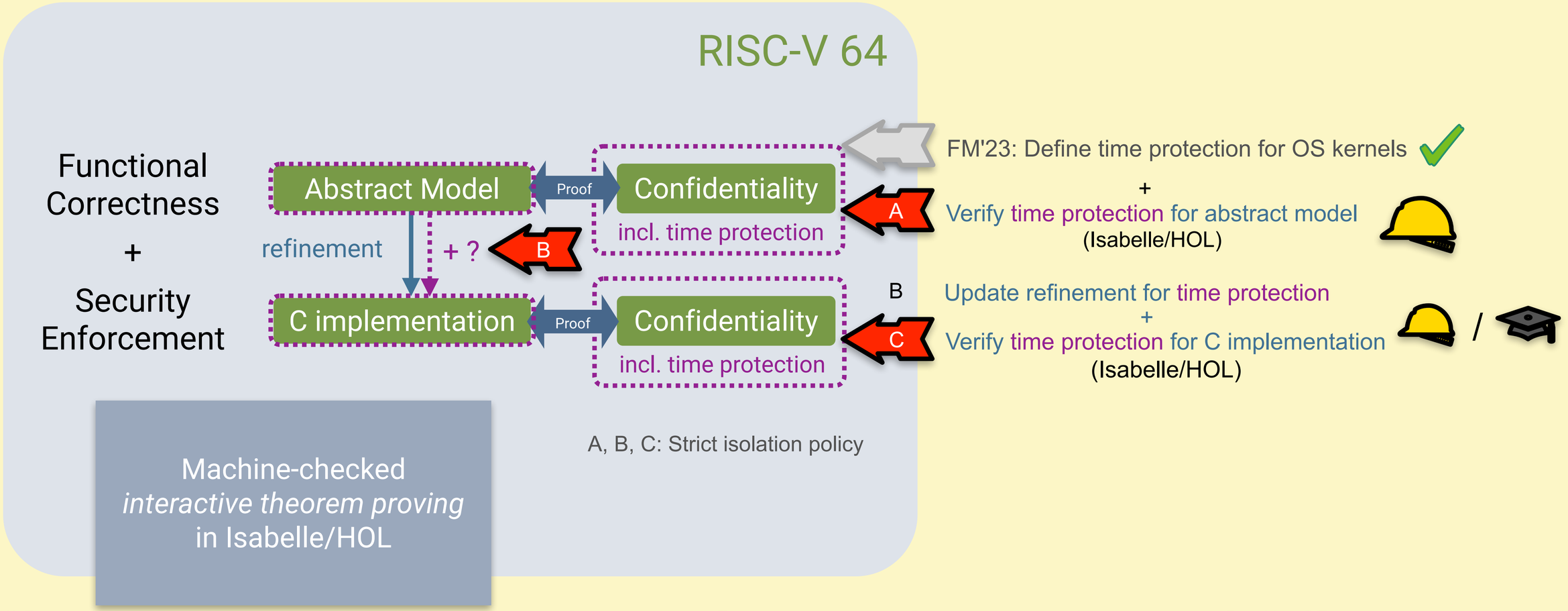


© Rob Sison 2026, CC BY 4.0

Time protection verification for seL4

The plan 2 years ago… *(presented at FM-OZ'24)*

Thanks to funding from
Australian Research Council

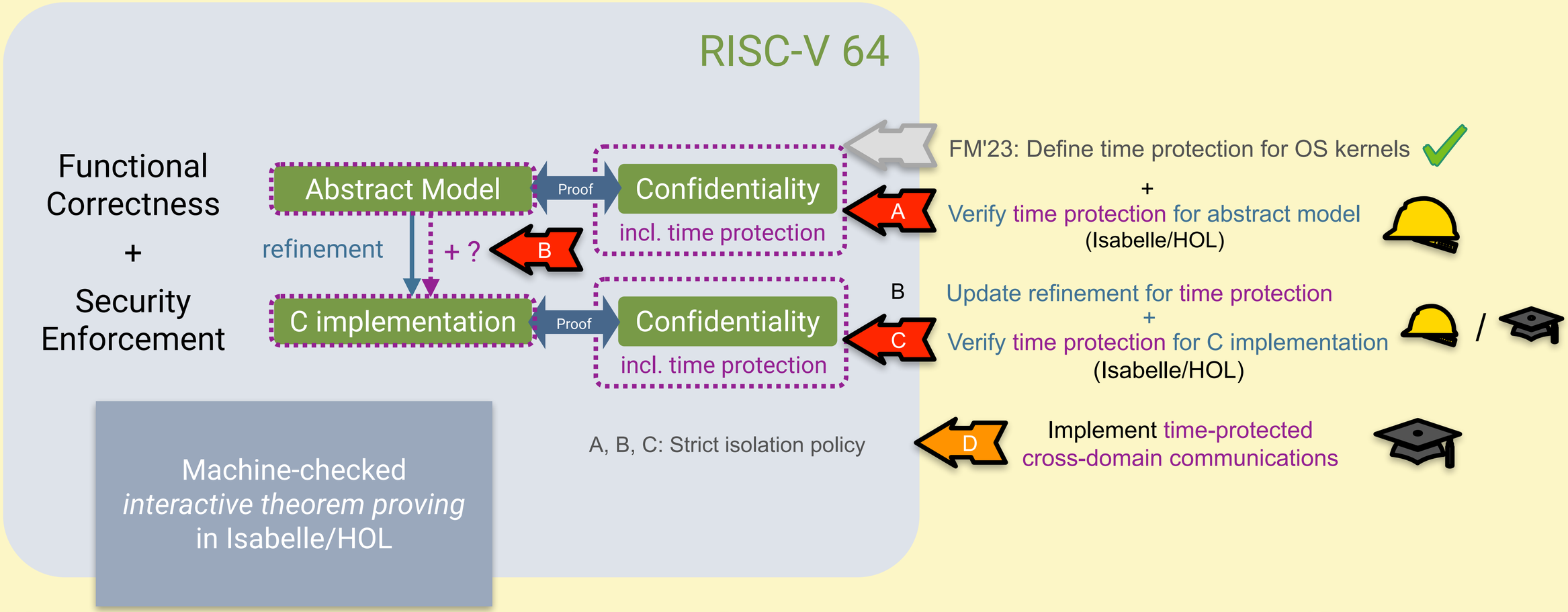


© Rob Sison 2026, CC BY 4.0

Time protection verification for seL4

The plan 2 years ago… *(presented at FM-OZ'24)*

Thanks to funding from
Australian Research Council

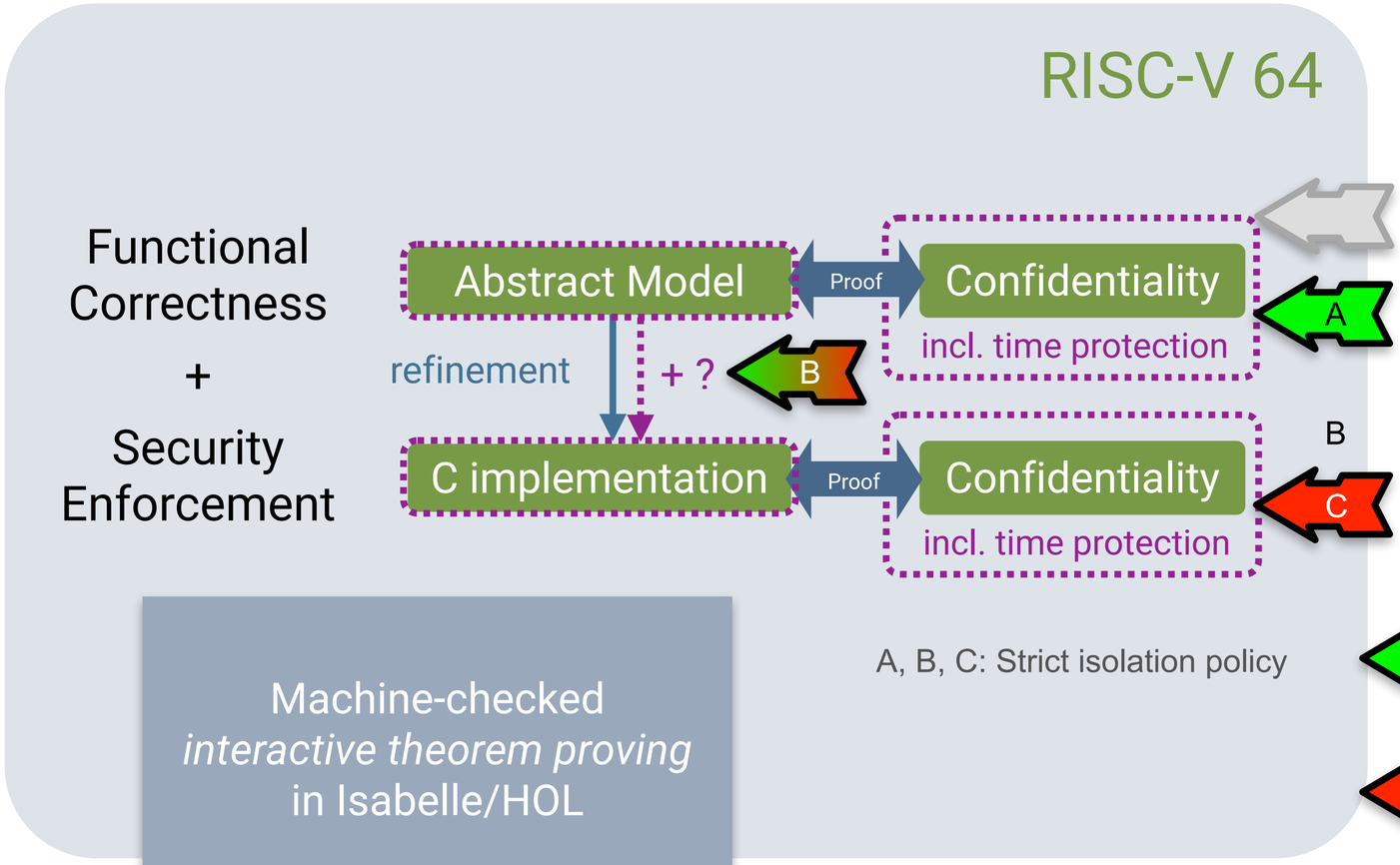


© Rob Sison 2026, CC BY 4.0

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

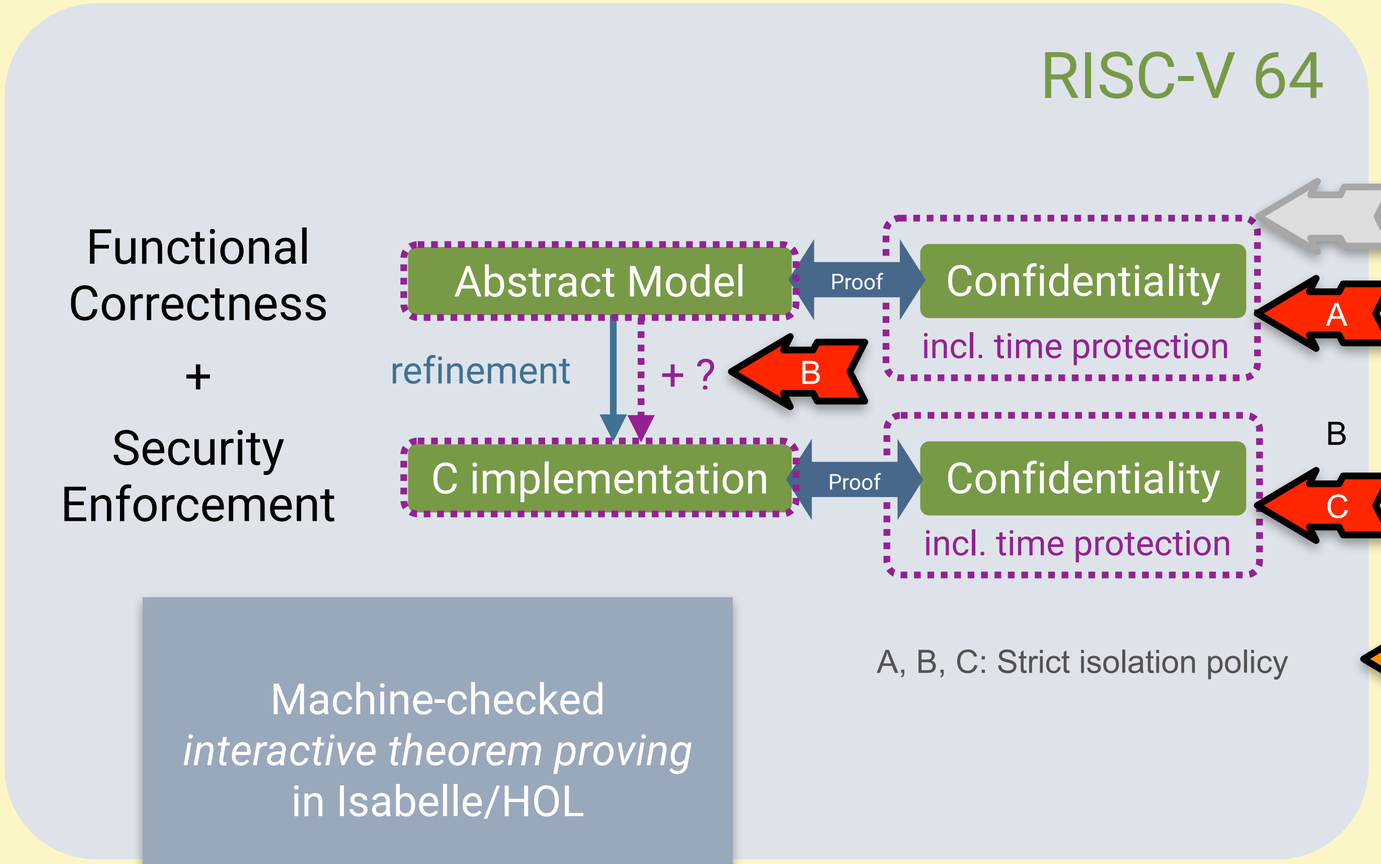
© Rob Sison 2026, CC BY 4.0

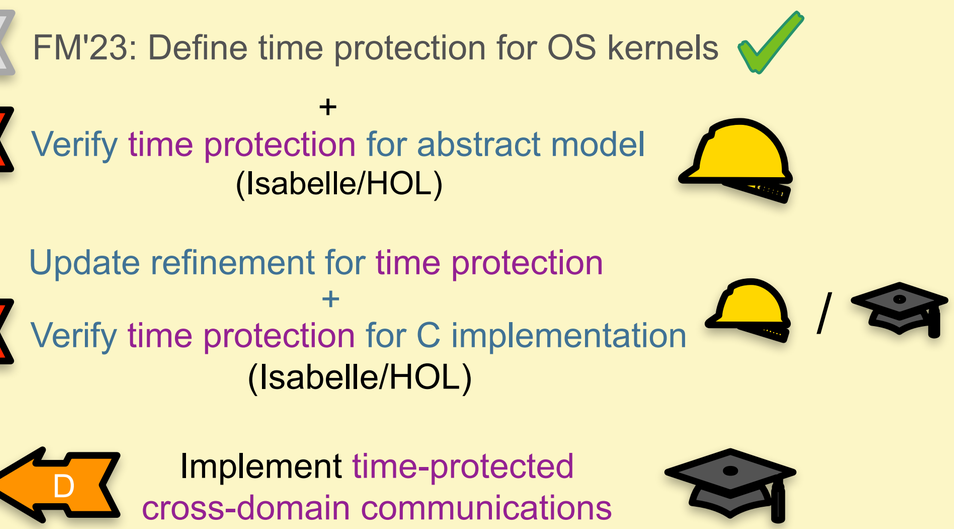
Time protection verification for seL4

Since the plan 2 years ago…

Thanks to funding from

cyberagentur





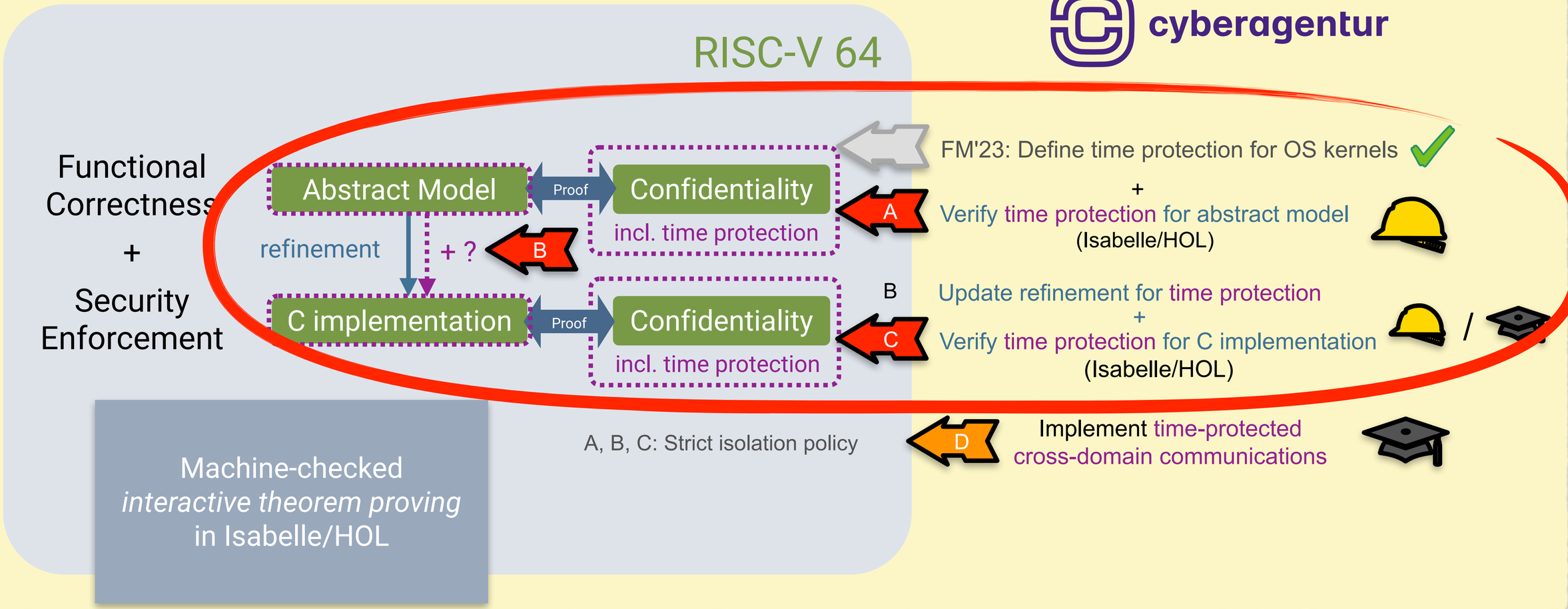
© Rob Sison 2026, CC BY 4.0

Time protection verification for seL4

Since the plan 2 years ago…

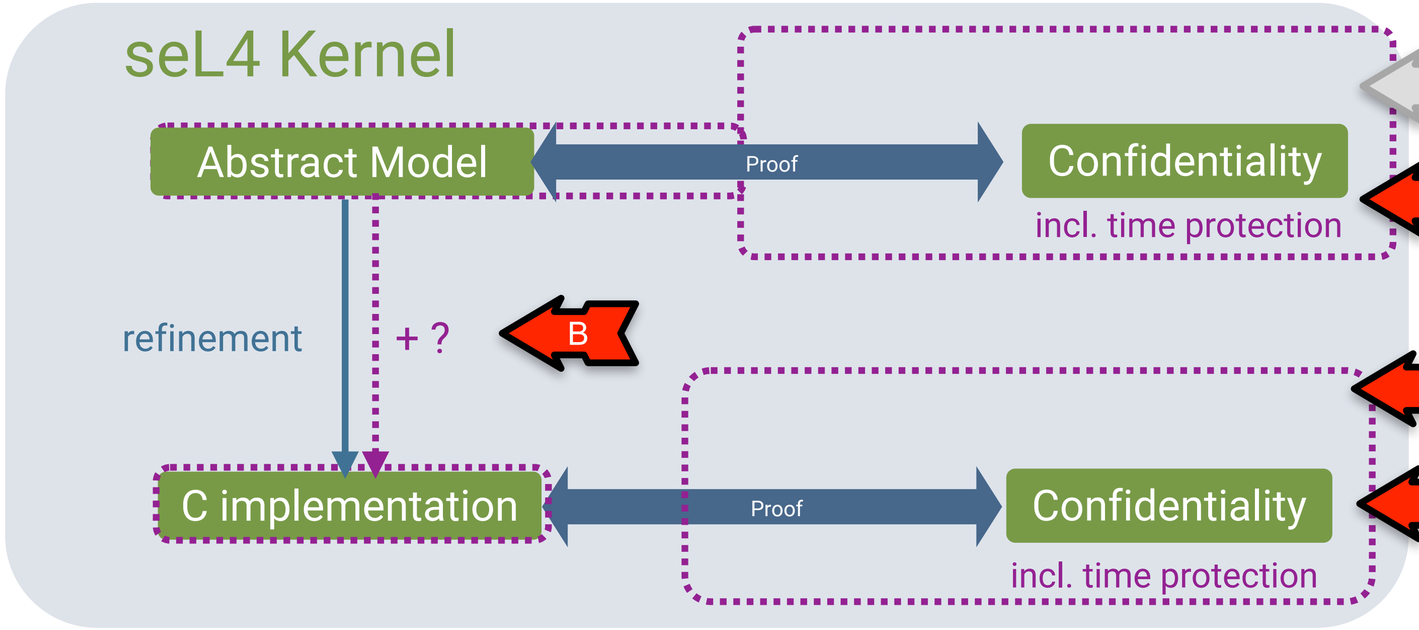
Thanks to funding from

cyberagentur



© Rob Sison 2026, CC BY 4.0

Originally planned approach… (FM-OZ'24)



FM'23: Define time protection for OS kernels ✓

+

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

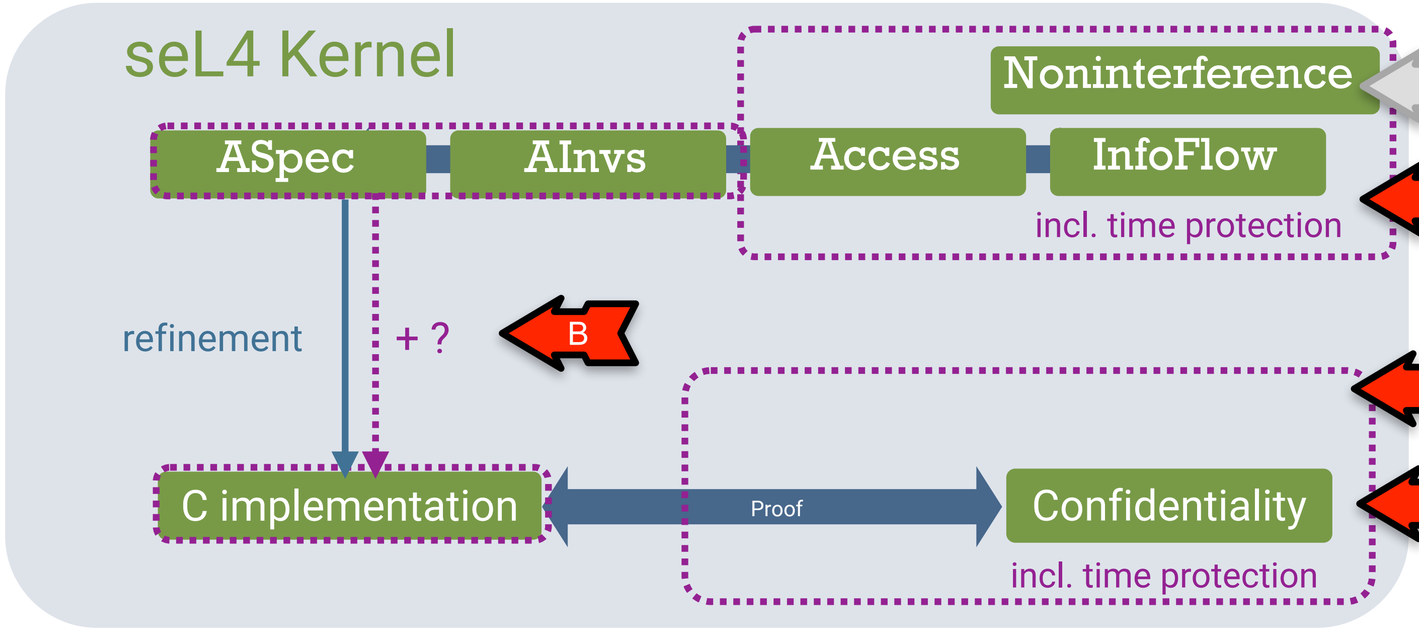
B Update refinement for time protection

+

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

© Rob Sison 2026, CC BY 4.0

Originally planned approach… (FM-OZ'24)



FM'23: Define time protection for OS kernels ✓

+

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

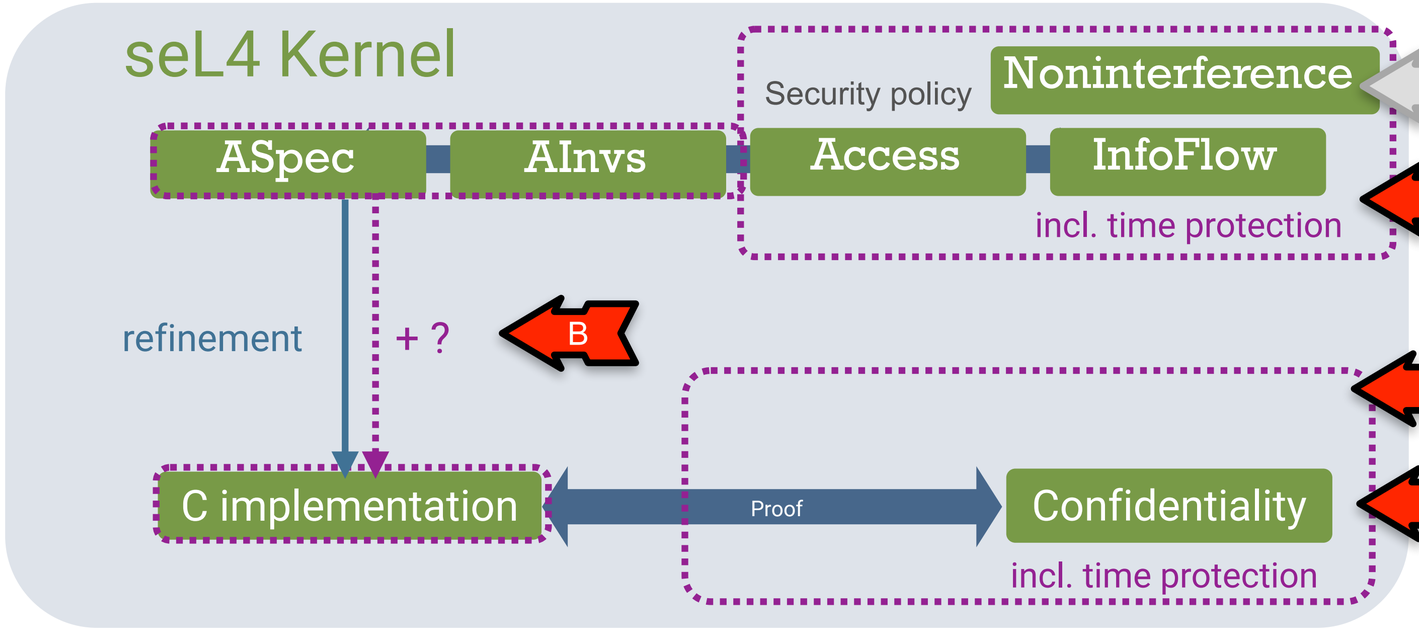
B Update refinement for time protection

+

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

© Rob Sison 2026, CC BY 4.0

Originally planned approach… (FM-OZ'24)



FM'23: Define time protection for OS kernels ✓

+

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

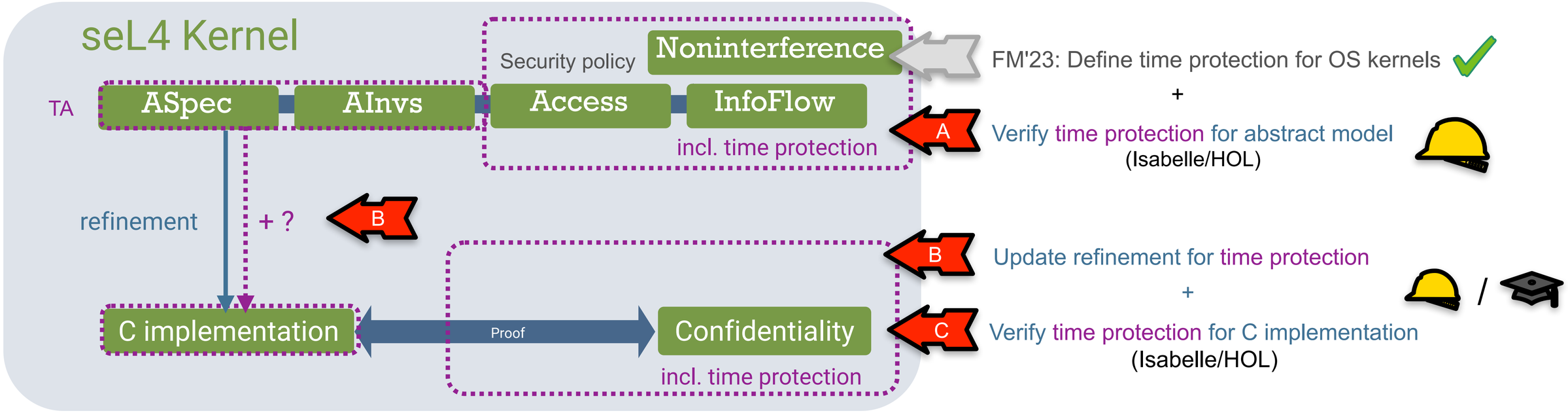
B Update refinement for time protection

+

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

© Rob Sison 2026, CC BY 4.0

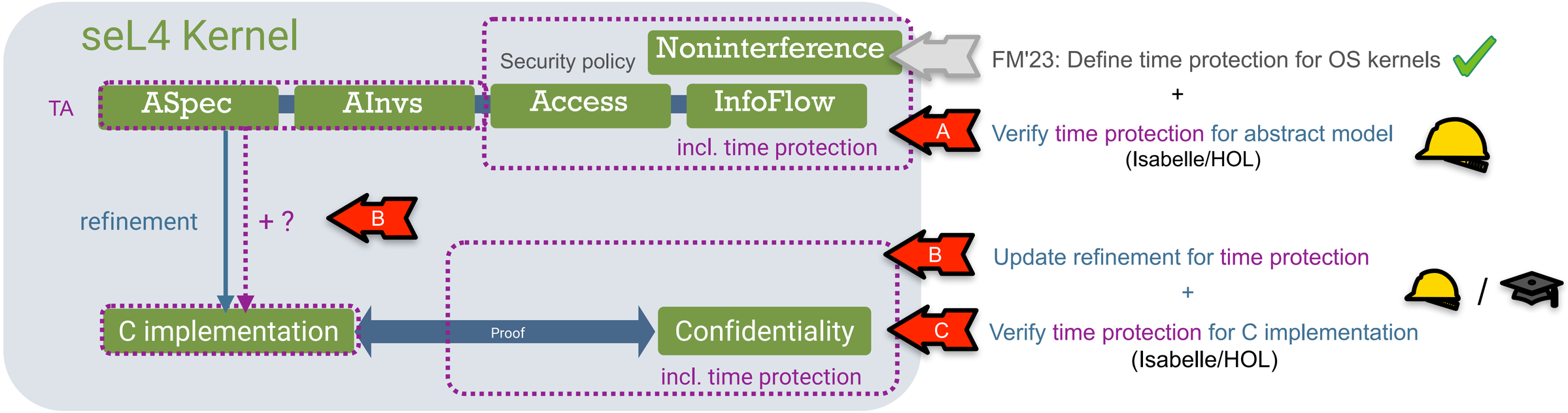
Originally planned approach… (FM-OZ'24)



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

© Rob Sison 2026, CC BY 4.0

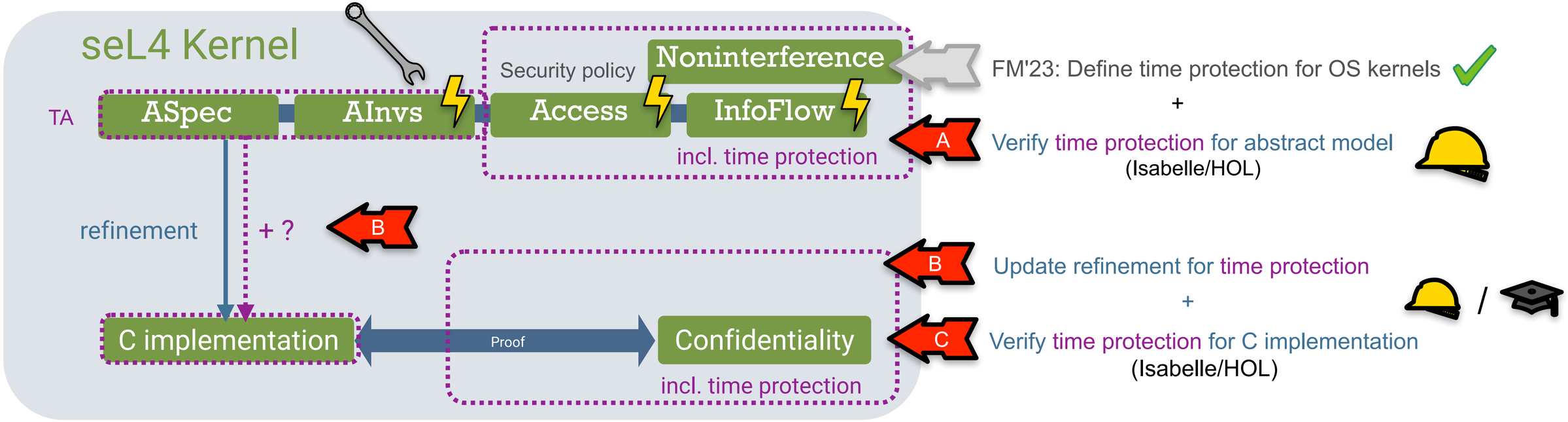
Originally planned approach… (FM-OZ'24)



- Abstract model (**ASpec**) checks dynamic *touched addresses* (TA) set
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)

© Rob Sison 2026, CC BY 4.0

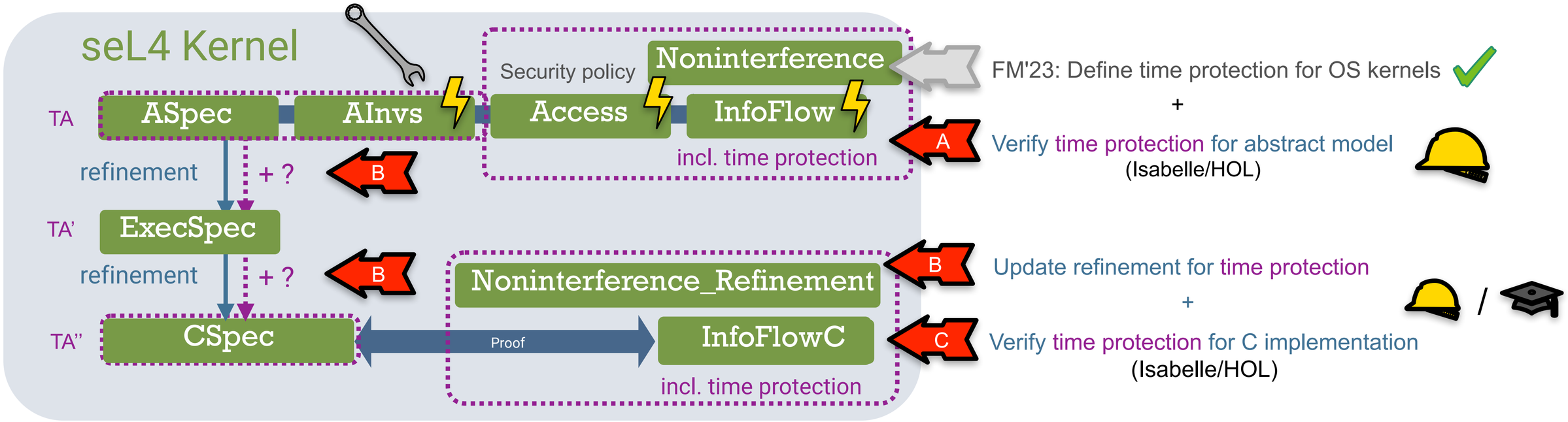
Originally planned approach… (FM-OZ'24)



- Abstract model (**ASpec**) checks dynamic *touched addresses* (TA) set
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs

© Rob Sison 2026, CC BY 4.0

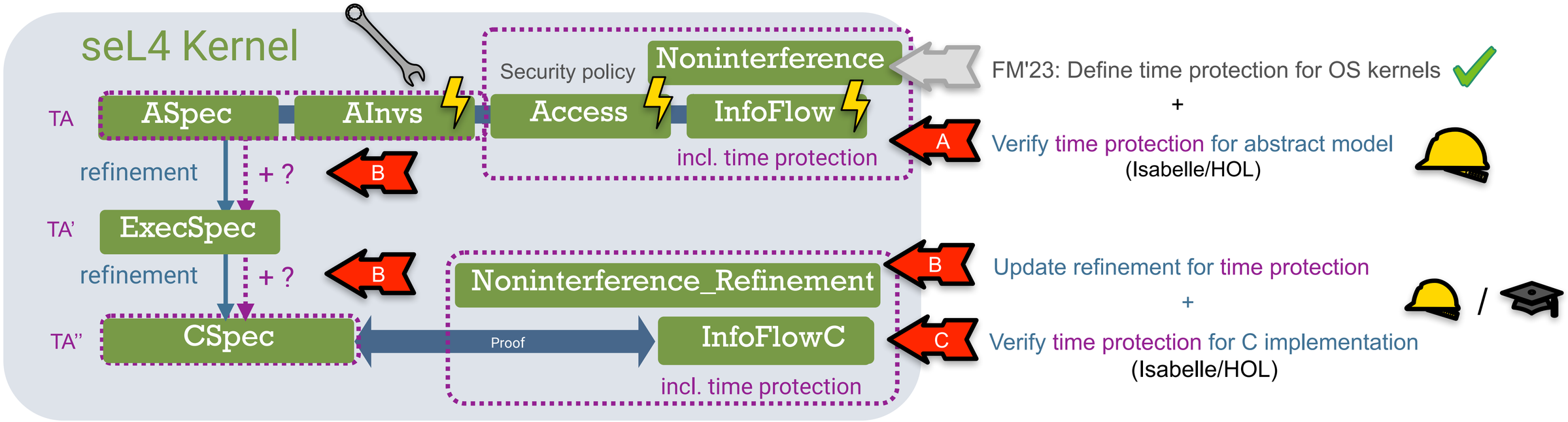
Originally planned approach… (FM-OZ'24)



- Abstract model (**ASpec**) checks dynamic *touched addresses* (TA) set
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)

© Rob Sison 2026, CC BY 4.0

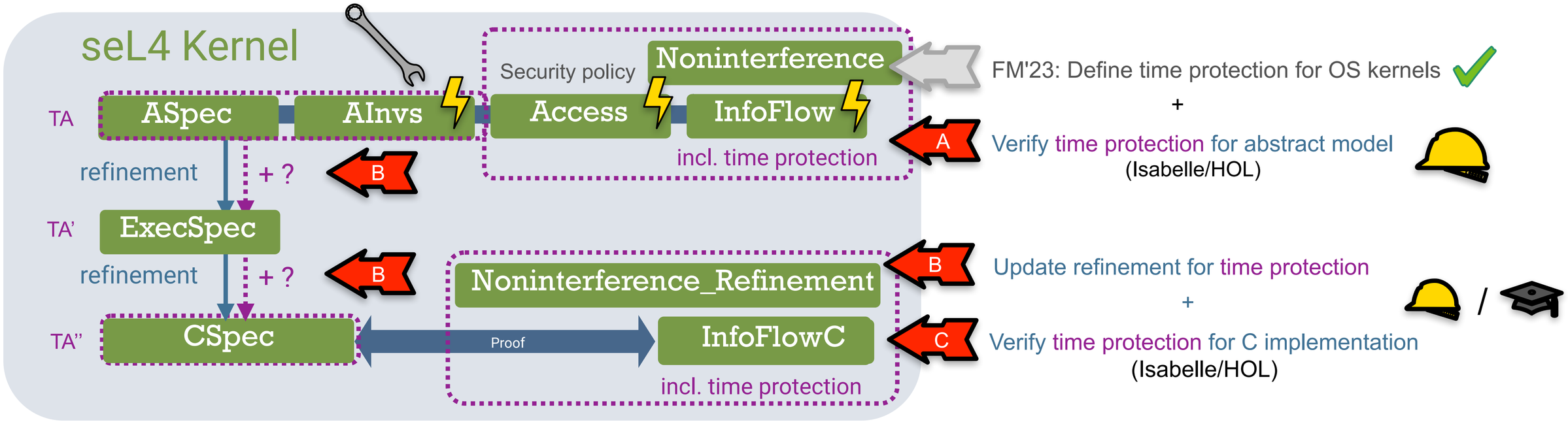
Originally planned approach… (FM-OZ'24)



- Abstract model (**ASpec**) checks dynamic *touched addresses* (TA) set
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

© Rob Sison 2026, CC BY 4.0

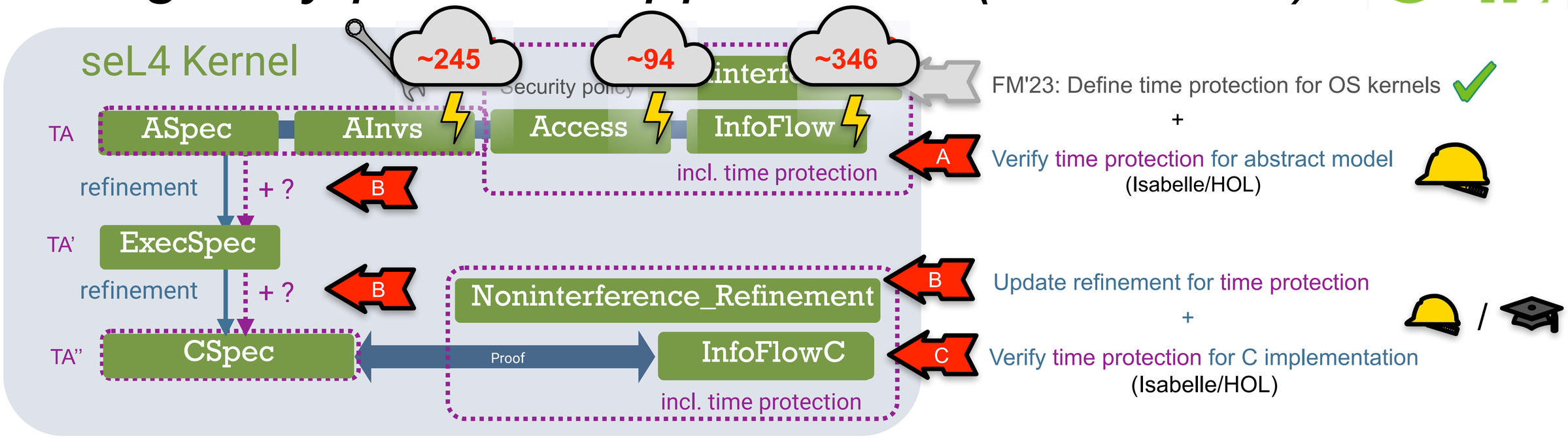
Originally planned approach… (FM-OZ'24)



- Abstract model (**ASpec**) checks dynamic *touched addresses* (TA) set
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs *How* many breakages?
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

© Rob Sison 2026, CC BY 4.0

Originally planned approach… (FM-OZ'24)

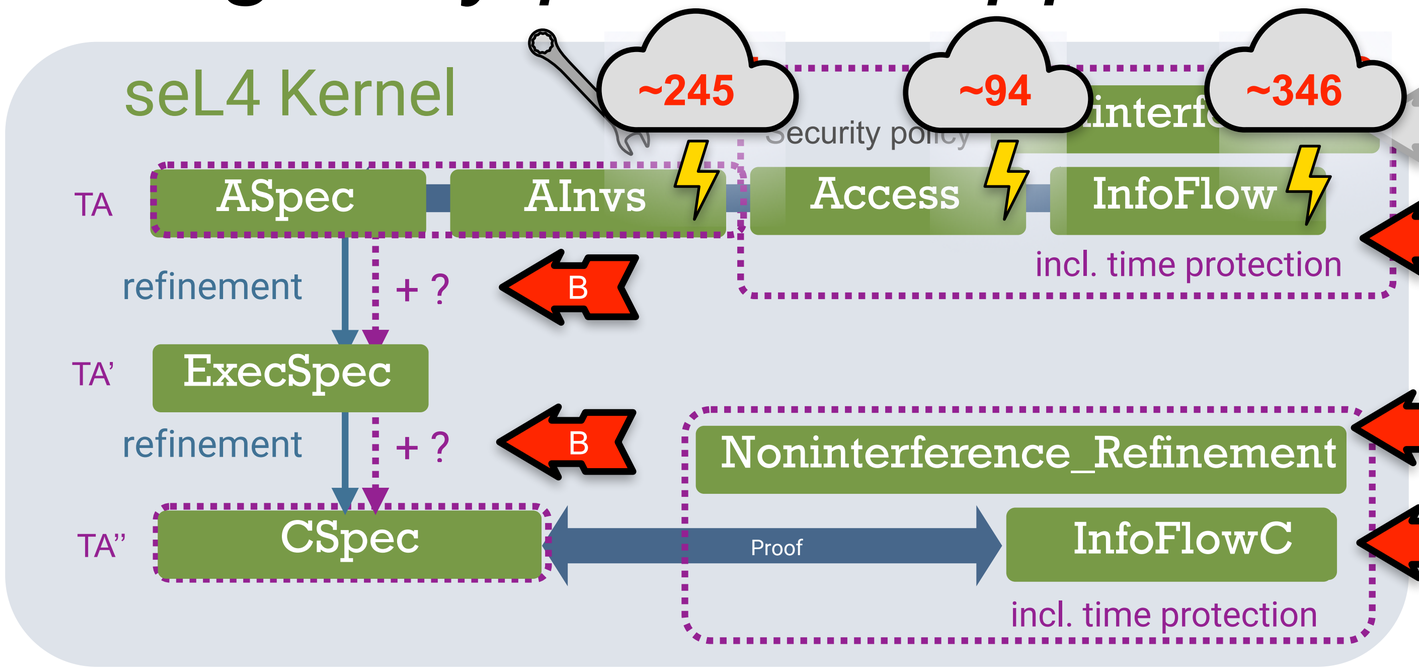


- Abstract model (**ASpec**) checks dynamic *touched addresses* (TA) set
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

How many breakages? → > 685, *after* the ones we already fixed

© Rob Sison 2026, CC BY 4.0

Originally planned approach… (FM-OZ'24)



FM'23: Define time protection for OS kernels ✓

+

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

B — Update refinement for time protection

+

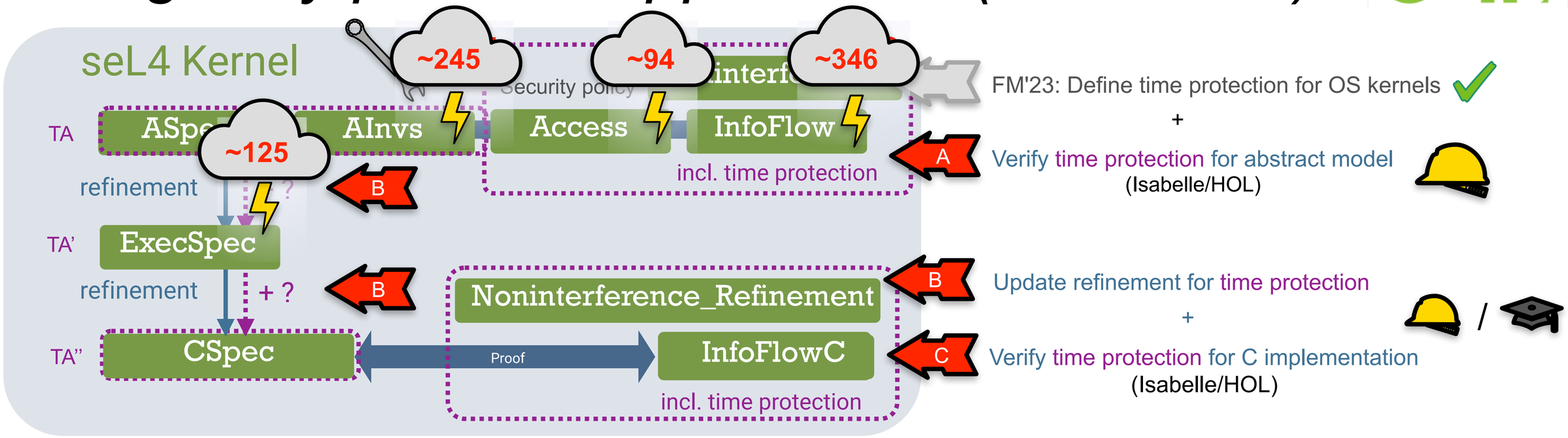
C — Verify time protection for C implementation (Isabelle/HOL)

- Abstract model (**ASpec**) checks dynamic *touched addresses* (TA) set → *125* times
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

How many breakages? → > 685, *after* the ones we already fixed

© Rob Sison 2026, CC BY 4.0

Originally planned approach… (FM-OZ'24)

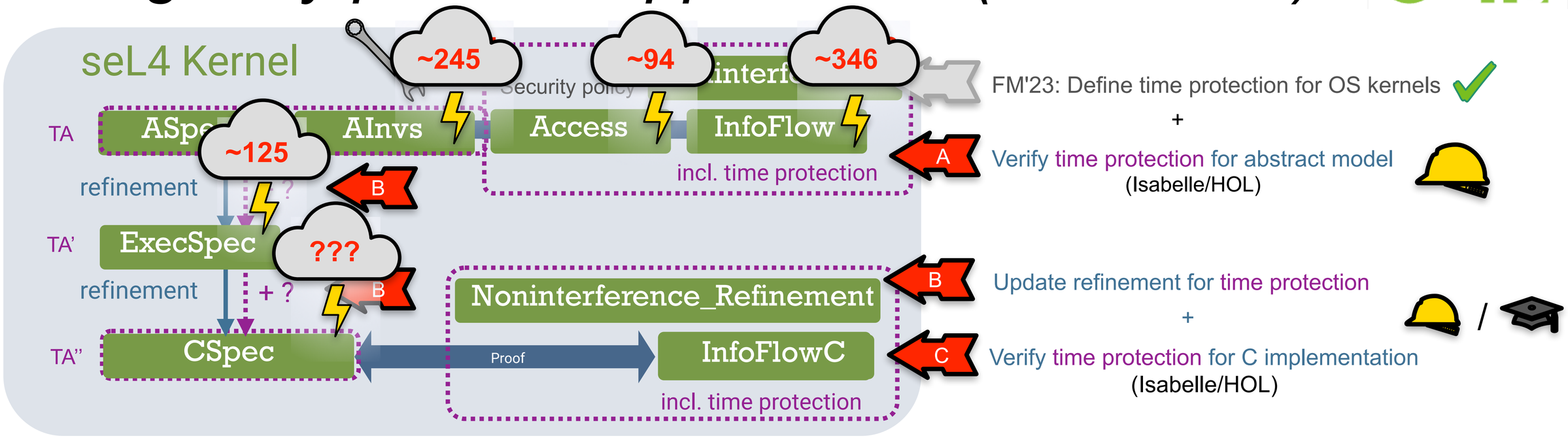


- Abstract model (**ASpec**) checks dynamic *touched addresses* (TA) set → *125* times
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

How many breakages? → > 685, *after* the ones we already fixed

© Rob Sison 2026, CC BY 4.0

Originally planned approach… (FM-OZ'24)

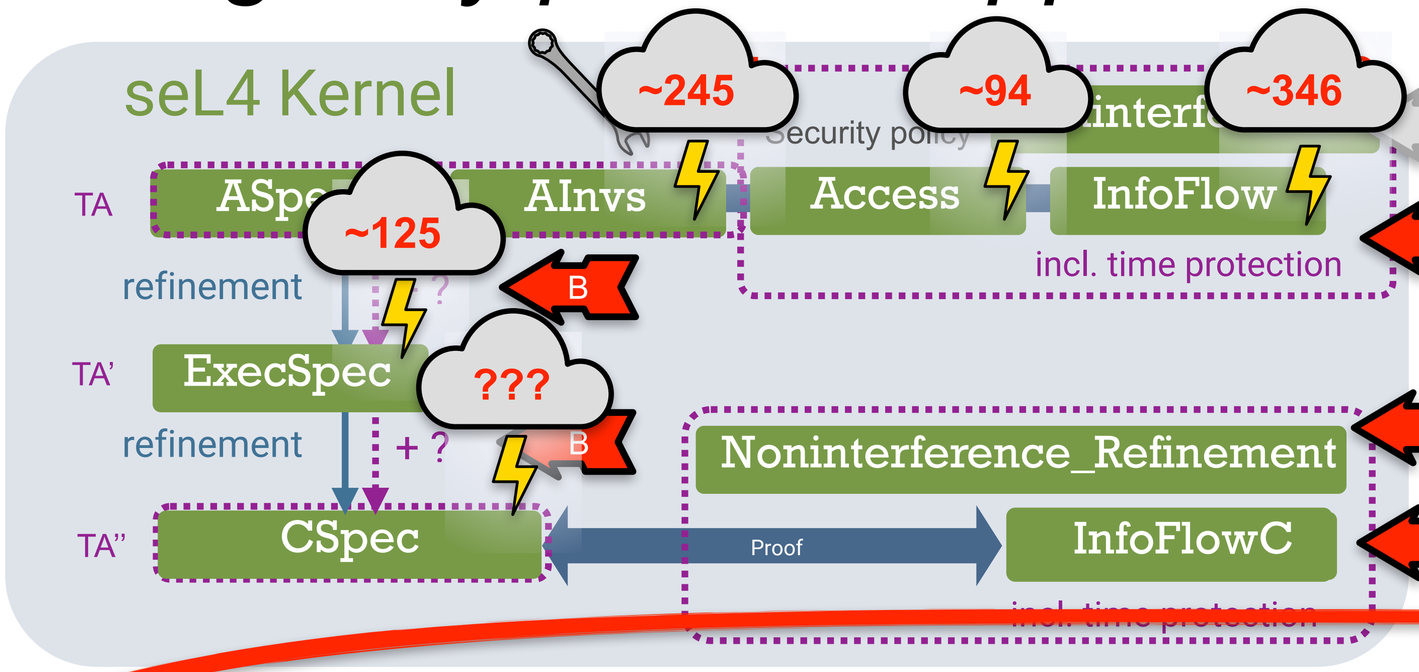


- Abstract model (**ASpec**) checks dynamic *touched addresses* (TA) set → *125* times
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

How many breakages? → > 685, *after* the ones we already fixed

© Rob Sison 2026, CC BY 4.0

Originally planned approach… (FM-OZ'24)



FM'23: Define time protection for OS kernels ✓

+

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

B — Update refinement for time protection

+

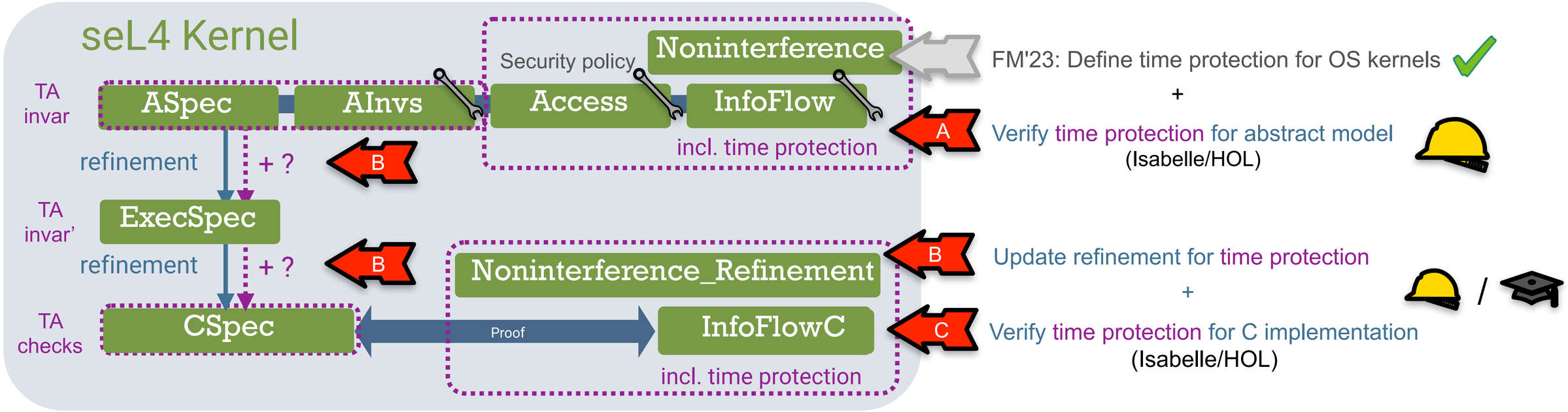
C — Verify time protection for C implementation (Isabelle/HOL)

- Abstract model (**ASpec**) checks dynamic *touched addresses* (TA) set → *125* times
- (A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs
- (B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)
  - Tricky part #1: Refinement to CSpec accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

How many breakages? → > 685, *after* the ones we already fixed

© Rob Sison 2026, CC BY 4.0

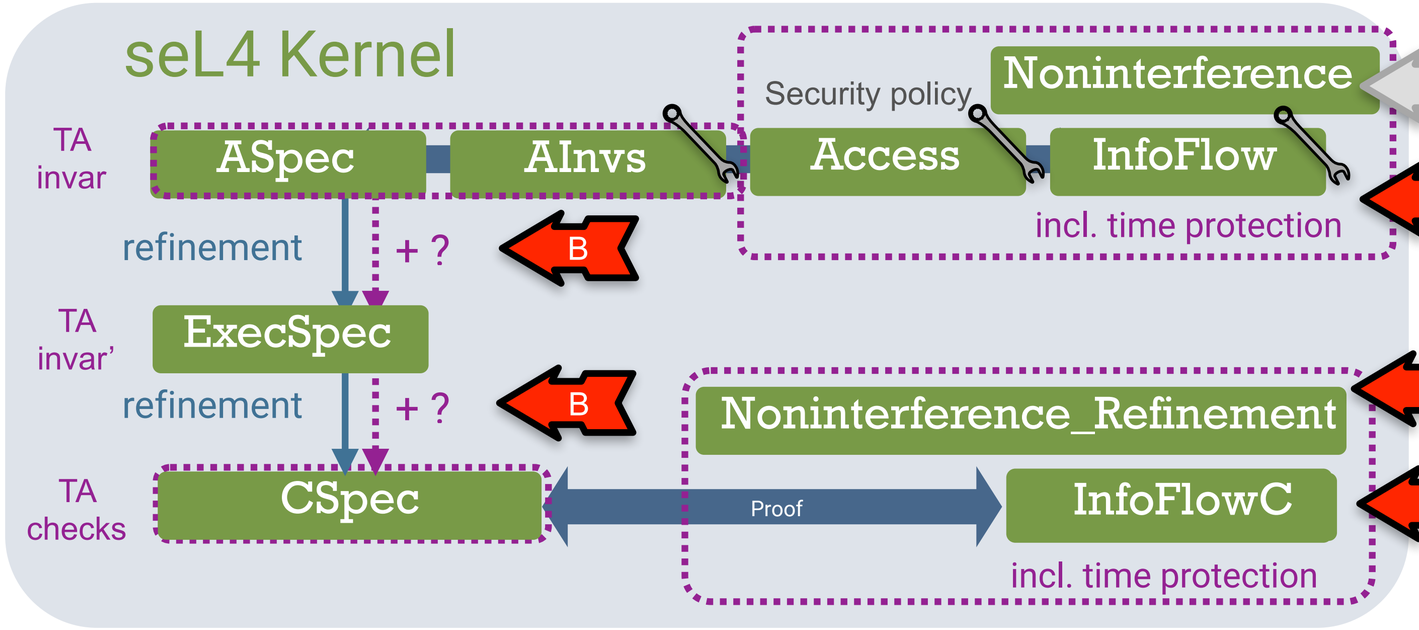
New approach



- ~~Abstract model (**ASpec**) checks dynamic *touched addresses* (TA) set~~
- ~~(A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)~~
  - ~~Also: **AInvs**, **Access**, **InfoFlow** need repairs~~
- ~~(B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)~~
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

© Rob Sison 2026, CC BY 4.0

New approach



FM'23: Define time protection for OS kernels ✓

+

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

B — Update refinement for time protection

+

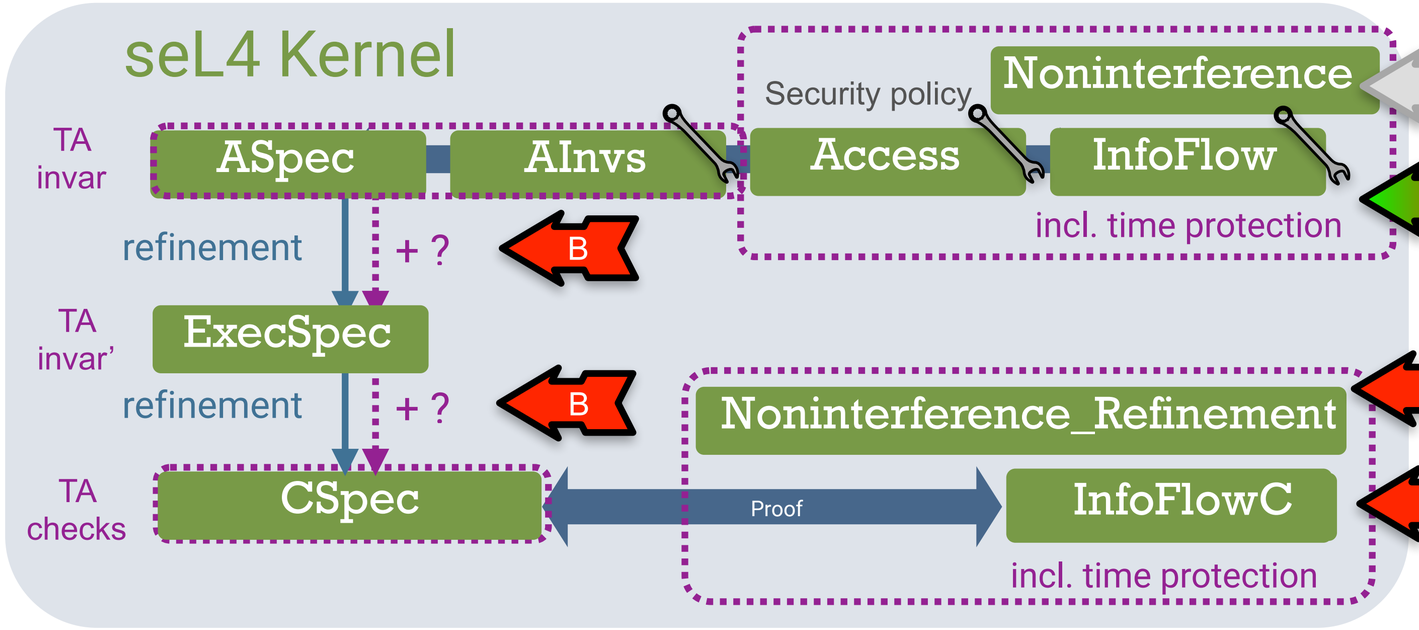
C — Verify time protection for C implementation (Isabelle/HOL)

- ~~Abstract model (**ASpec**) checks dynamic *touched addresses* (TA) set~~
- ~~(A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)~~
  - ~~Also: **AInvs**, **Access**, **InfoFlow** need repairs~~
- ~~(B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)~~
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

Sai Nair
BSc(Hons)
thesis

© Rob Sison 2026, CC BY 4.0

New approach



FM'23: Define time protection for OS kernels ✓

+

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

B — Update refinement for time protection

+

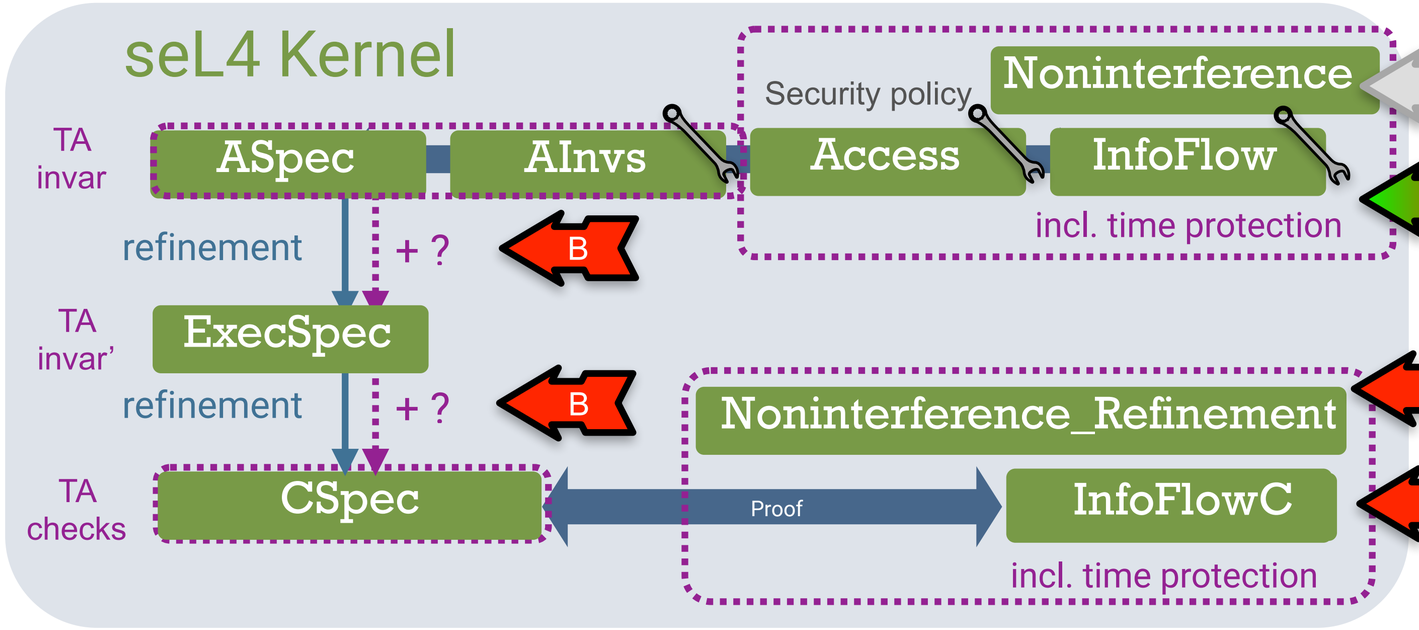
C — Verify time protection for C implementation (Isabelle/HOL)

- ~~Abstract model (**ASpec**) checks dynamic *touched addresses* (TA) set~~
- (A) Key **ASpec** property: "TA invariant"
  - ~~Also: **AInvs**, **Access**, **InfoFlow** need repairs~~
- ~~(B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)~~
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

Sai Nair
BSc(Hons)
thesis

© Rob Sison 2026, CC BY 4.0

New approach



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)

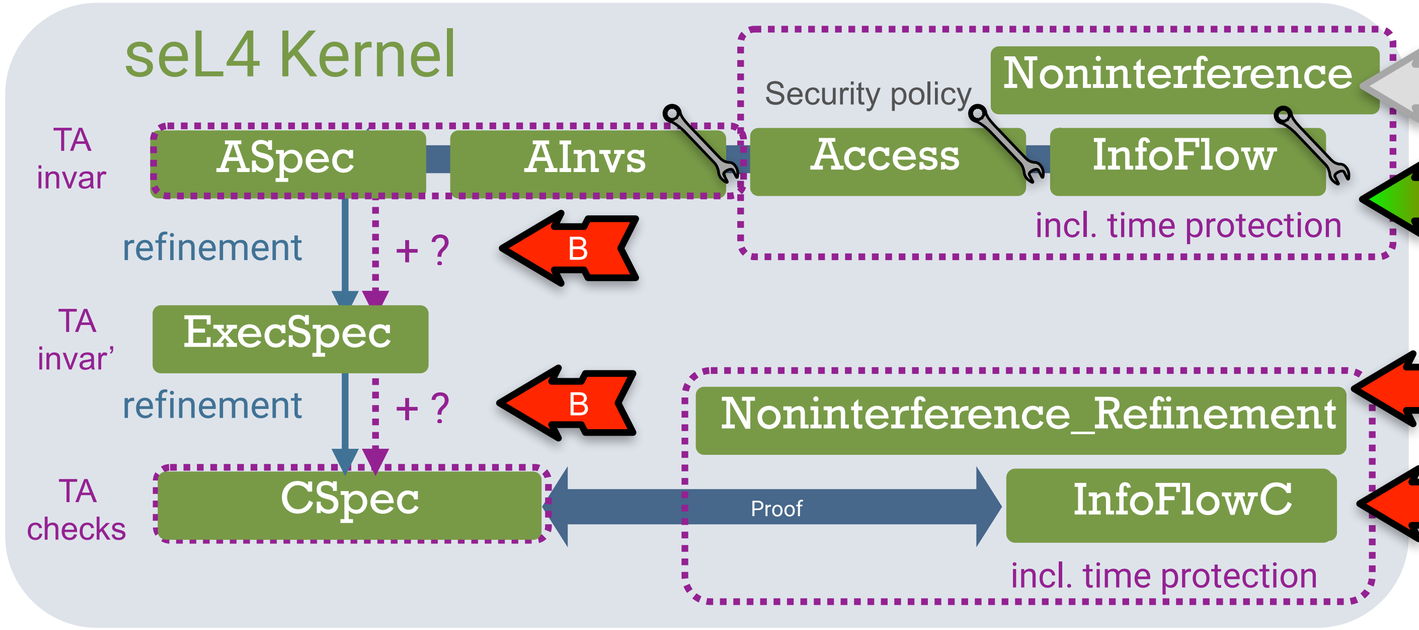
"obj refs don't point outside static *TA set"*

- (A) Key **ASpec** property: "TA invariant"
  - ~~Also: **AInvs**, **Access**, **InfoFlow** need repairs~~
- ~~(B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)~~
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

Sai Nair
BSc(Hons)
thesis

© Rob Sison 2026, CC BY 4.0

New approach



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"
  - ~~Also: **AInvs**, **Access**, **InfoFlow** need repairs~~
- ~~(B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)~~
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

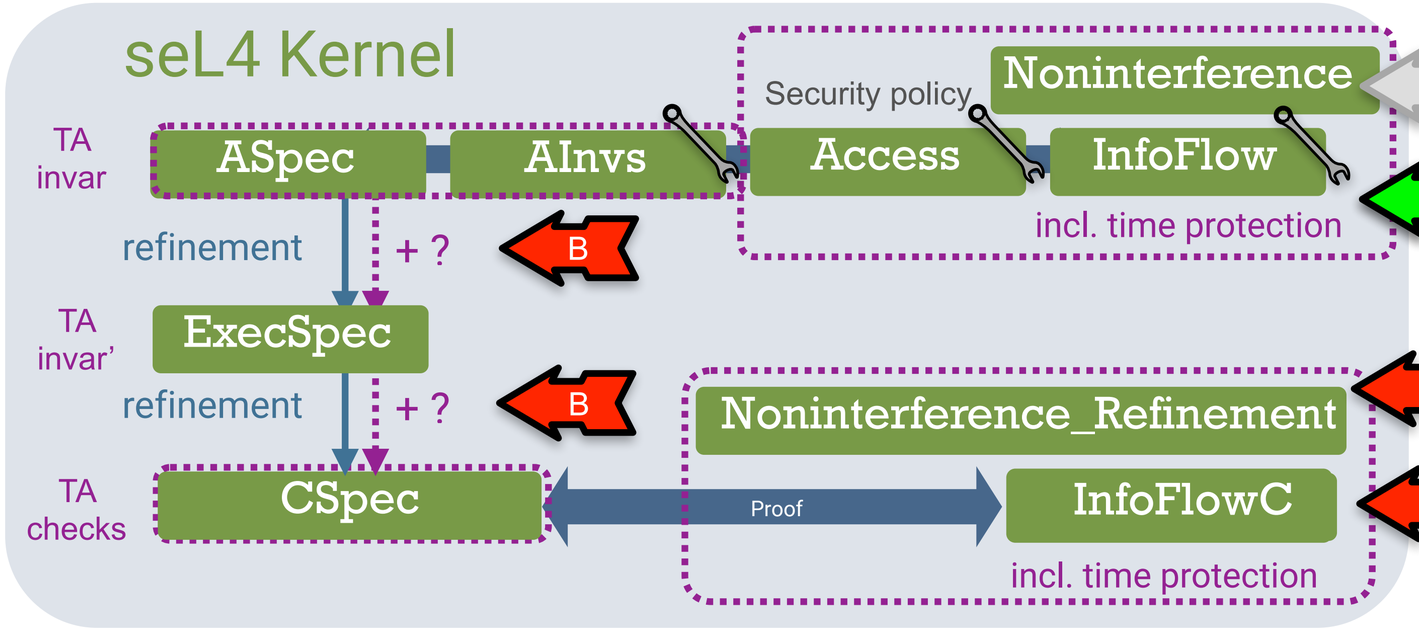
- Proof across **ASpec** in progress

Continuing on from

Sai Nair
BSc(Hons)
thesis

© Rob Sison 2026, CC BY 4.0

New approach



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"*

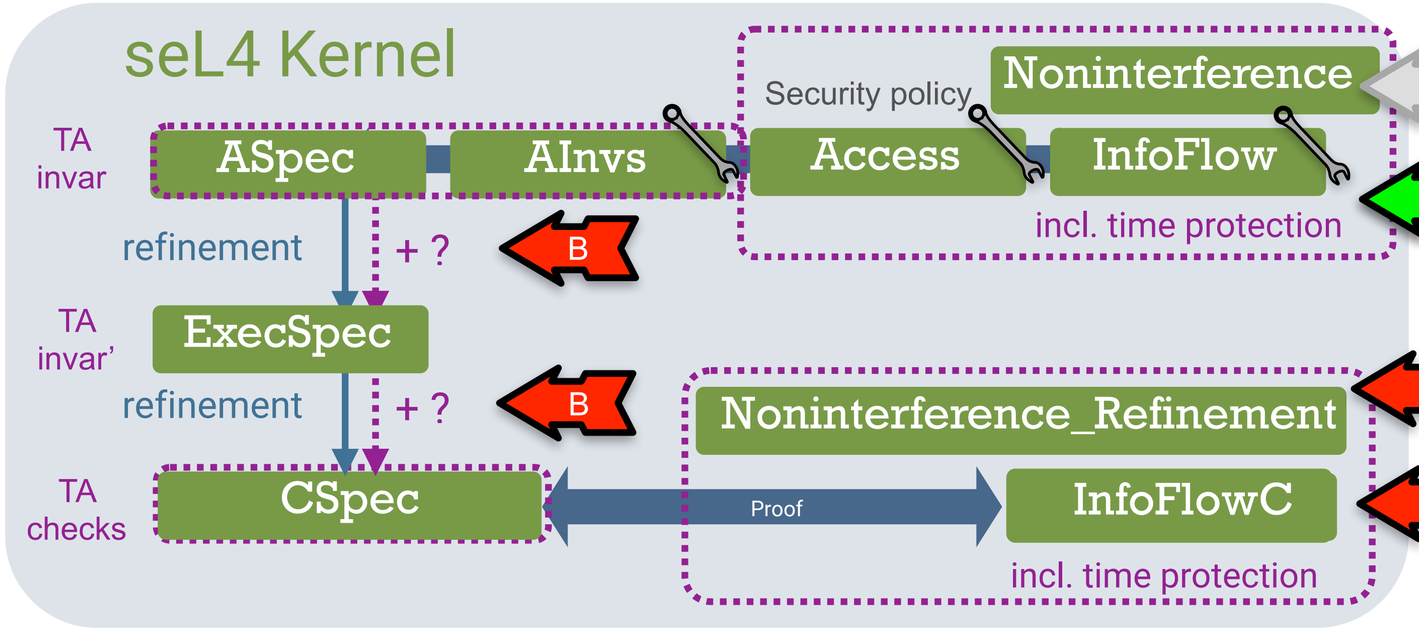
- (A) Key **ASpec** property: "TA invariant"
  - **ASpec** otherwise *integrated* into time protection theory ✓
- ~~(B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)~~
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

- Proof across **ASpec** in progress

Continuing on from Sai Nair BSc(Hons) thesis

© Rob Sison 2026, CC BY 4.0

New approach



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 ✓
- ~~(B + C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)~~
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

- Proof across **ASpec** in progress

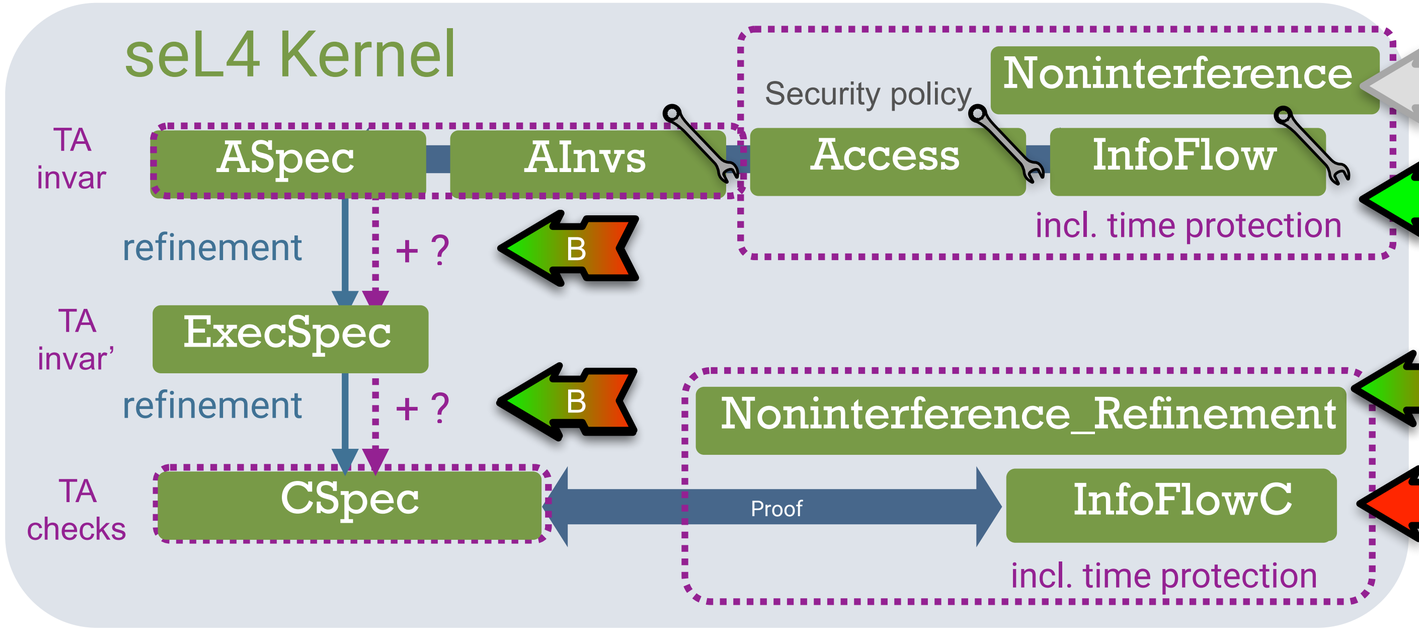
Continuing on from

Sai Nair
BSc(Hons)
thesis

Thomas Liang
BSc (Hons)
Thesis

© Rob Sison 2026, CC BY 4.0

New approach



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 ✓
- (B + C) Refinement: TA invariant (via **ExecSpec**) ⟹ TA checks pass (**CSpec**)
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level
- Proof across **ASpec** in progress

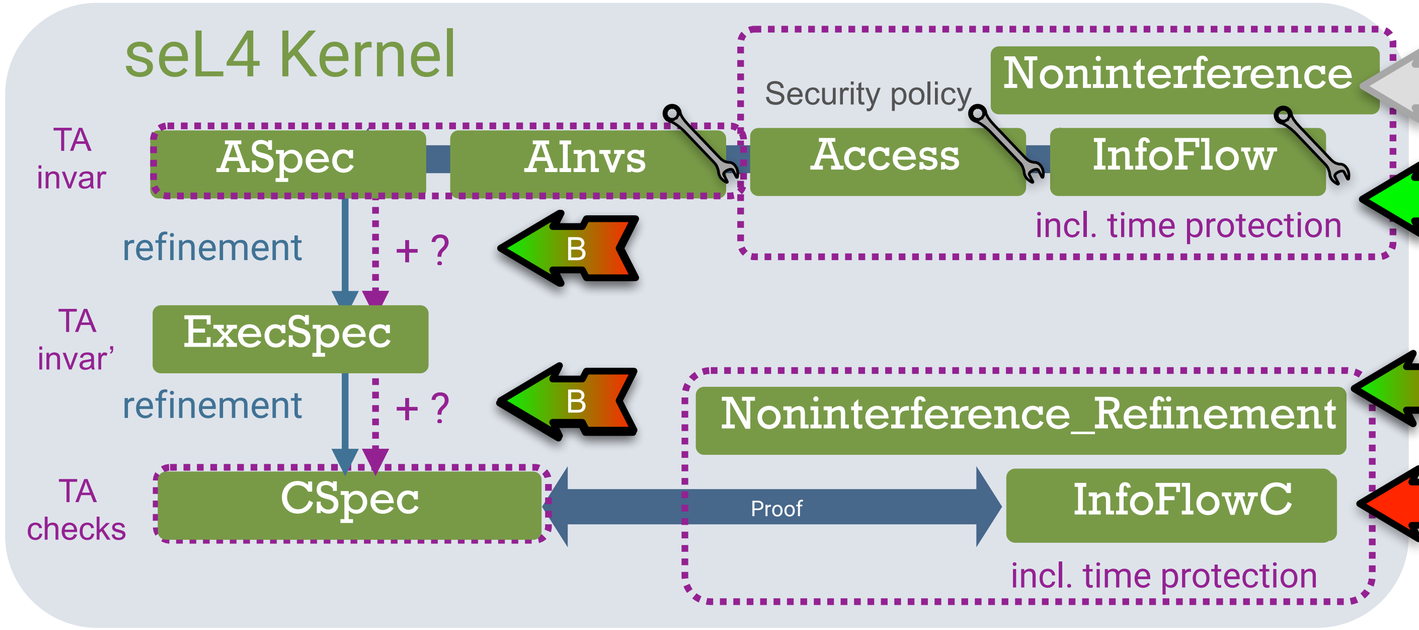
Continuing on from

Sai Nair
BSc(Hons)
thesis

Thomas Liang
BSc (Hons)
Thesis

© Rob Sison 2026, CC BY 4.0

What's next?



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 ✓
- (B + C) Refinement: TA invariant (via **ExecSpec**) ⟹ TA checks pass (**CSpec**)
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

- Proof across **ASpec** in progress

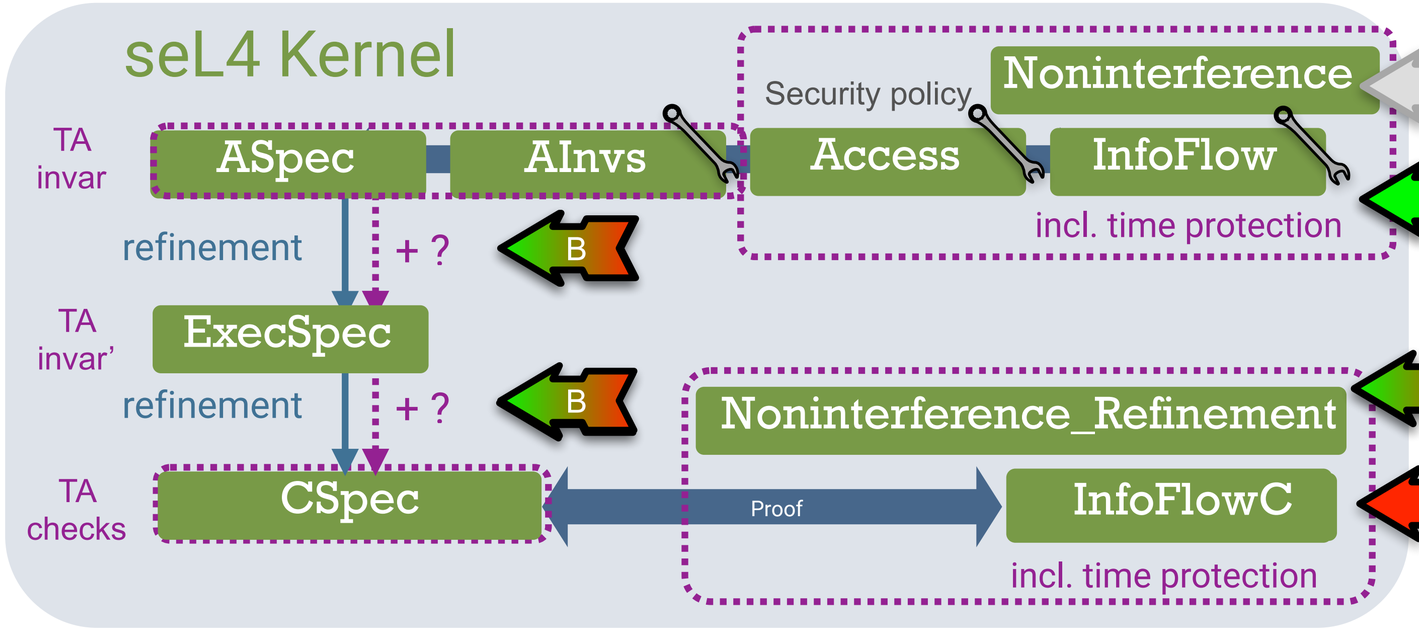
Continuing on from

Sai Nair
BSc(Hons)
thesis

Thomas Liang
BSc (Hons)
Thesis

© Rob Sison 2026, CC BY 4.0

What's next? Refinement



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 ✓
- (B + C) Refinement: TA invariant (via **ExecSpec**) ⟹ TA checks pass (**CSpec**)
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

- Proof across **ASpec** in progress

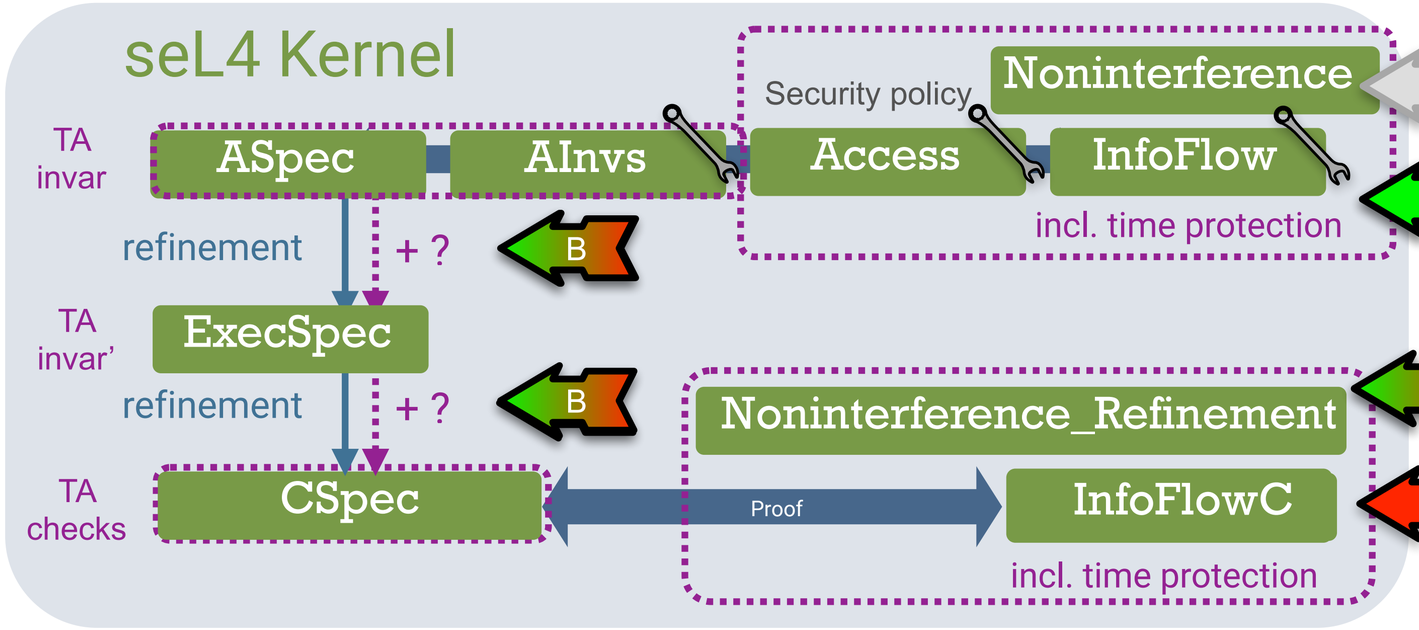
Continuing on from

Sai Nair
BSc(Hons)
thesis

Thomas Liang
BSc (Hons)
Thesis

© Rob Sison 2026, CC BY 4.0

What's next? Refinement



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 ✓
- (B + C) Refinement: TA invariant (via **ExecSpec**) ⟹ TA checks pass (**CSpec**)
  - Tricky part #1: Refinement to **CSpec** accesses new addresses
  - Tricky part #2: Making TA checks scalable at **CSpec** level

- Proof across **ASpec** in progress

Continuing on from

Sai Nair
BSc(Hons)
thesis

Continuing on from

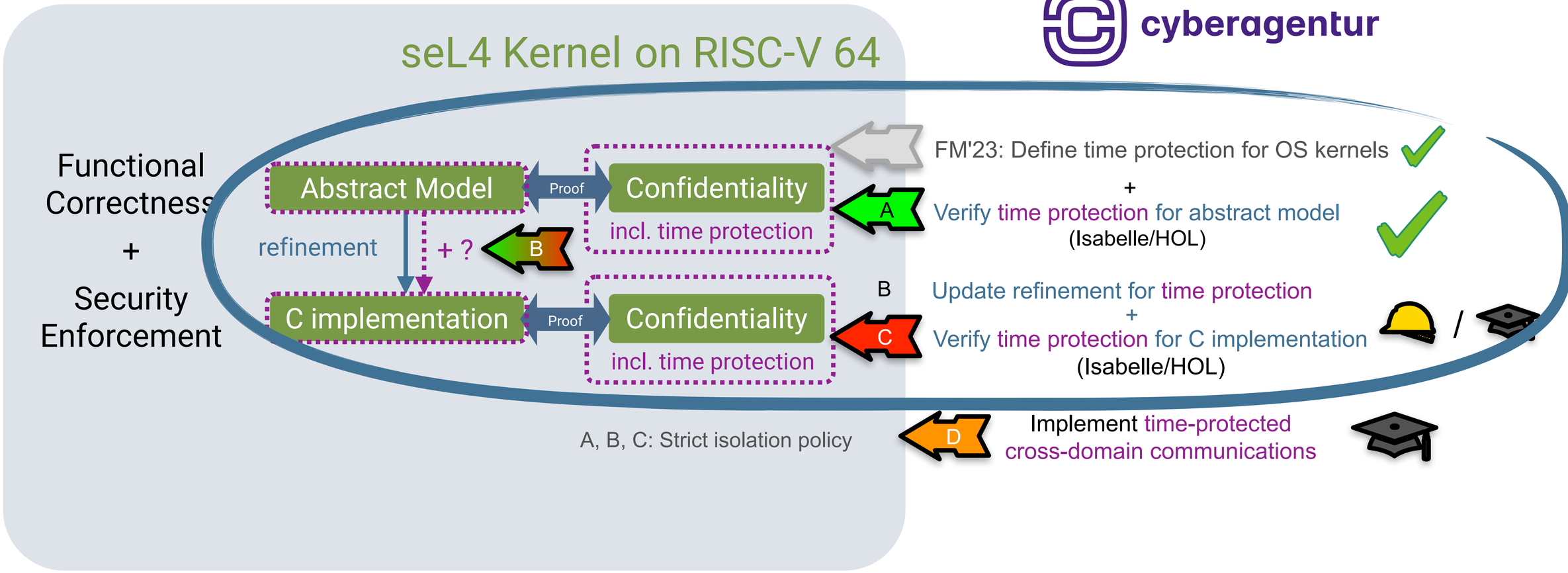
Thomas Liang
BSc (Hons)
Thesis

© Rob Sison 2026, CC BY 4.0

Time protection verification for seL4

Since the plan 2 years ago…

Thanks to funding from cyberagentur

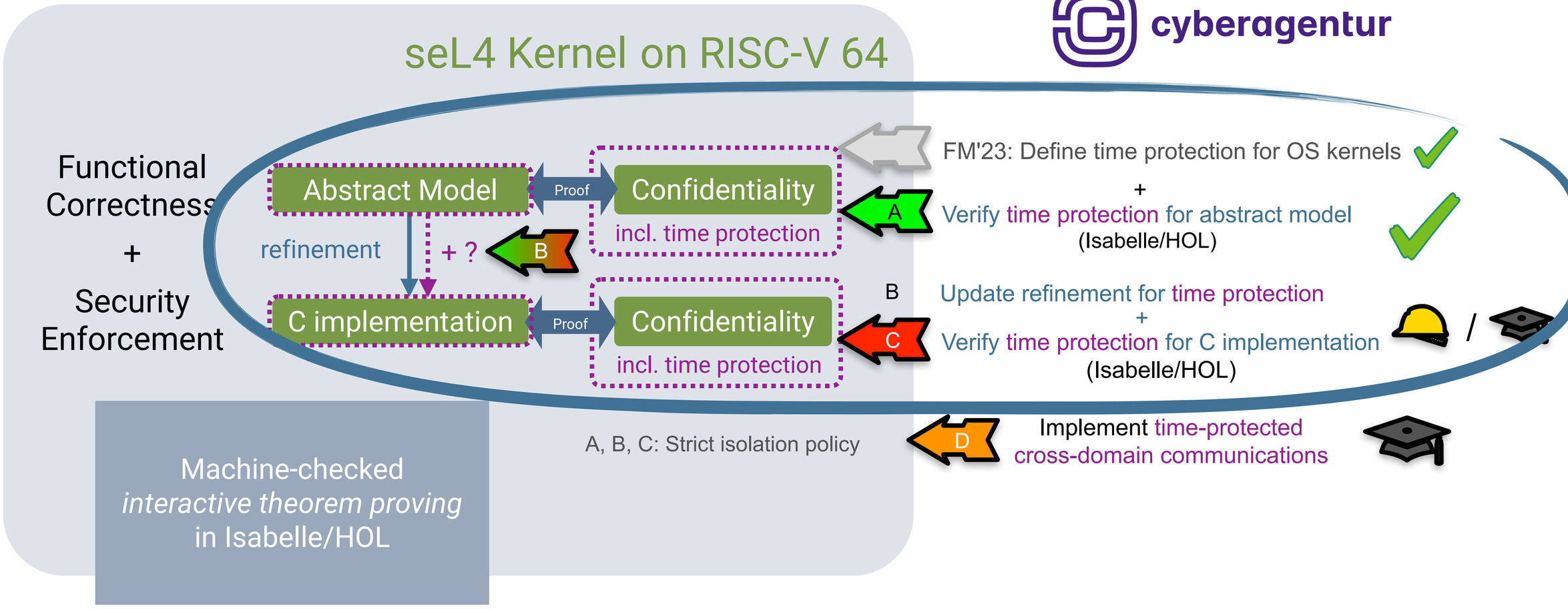


© Rob Sison 2026, CC BY 4.0

Time protection verification for seL4

Since the plan 2 years ago…

Thanks to funding from cyberagentur



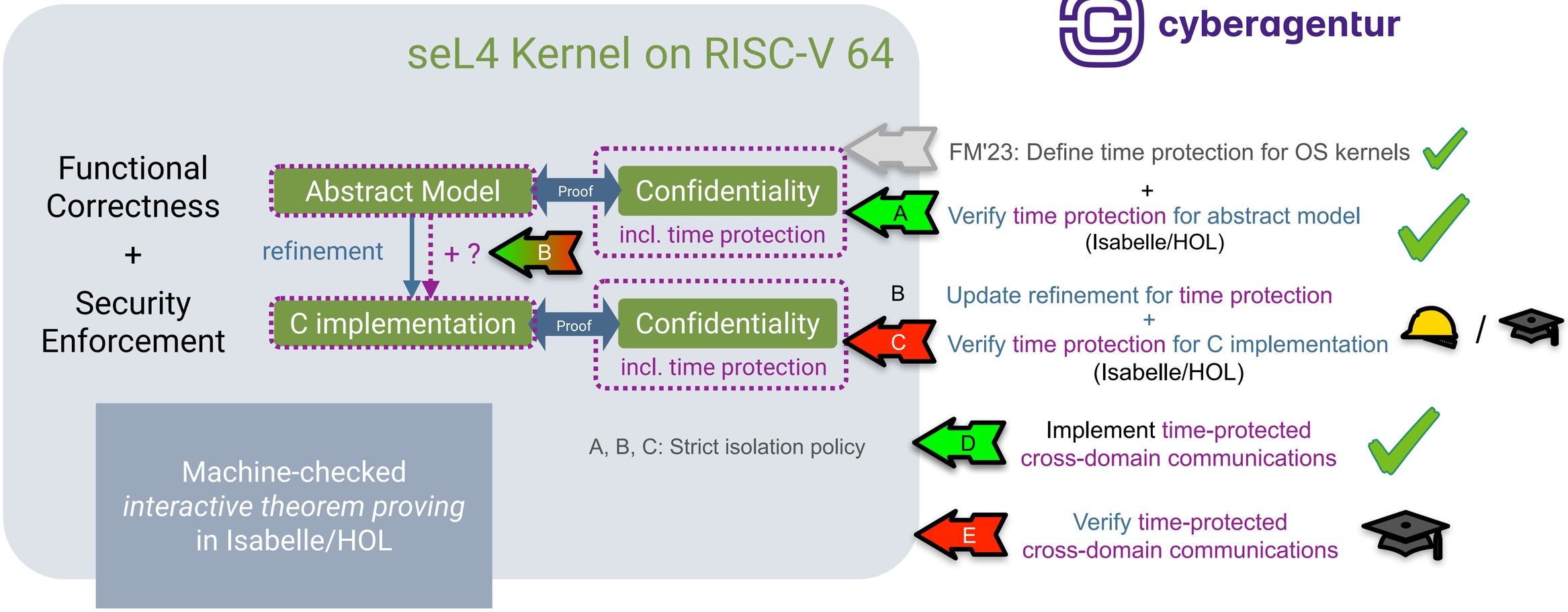
© Rob Sison 2026, CC BY 4.0

Time protection verification for seL4

The status today:

Thanks to funding from

cyberagentur



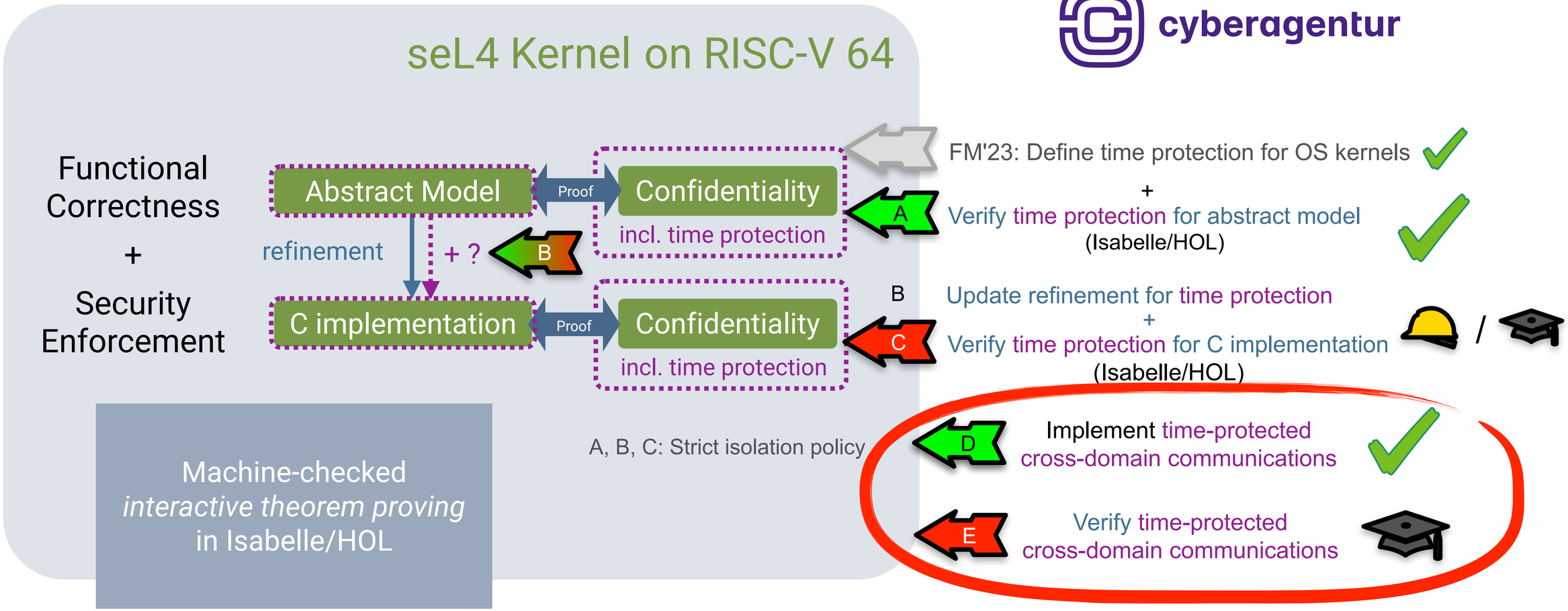
© Rob Sison 2026, CC BY 4.0

Time protection verification for seL4

The status today:

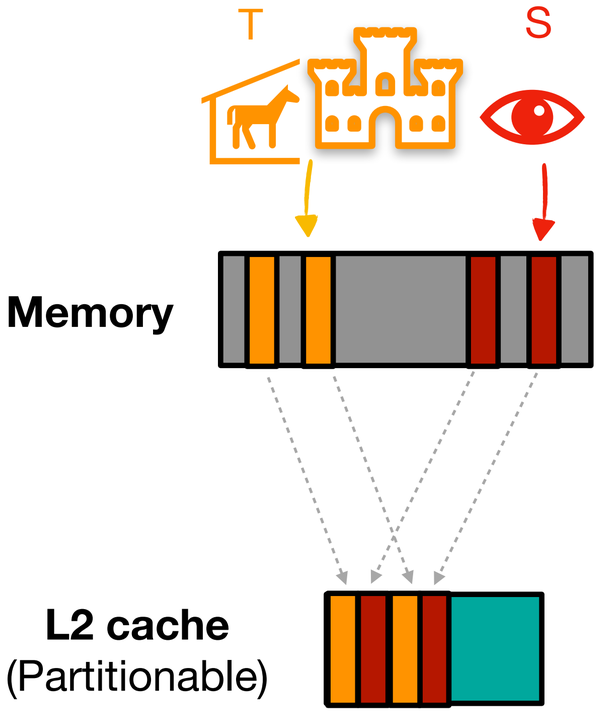
Thanks to funding from

cyberagentur



© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

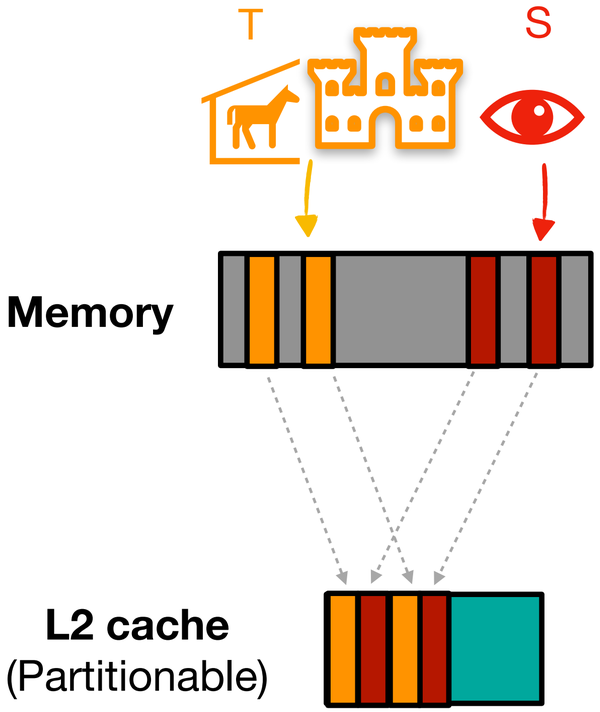


© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B



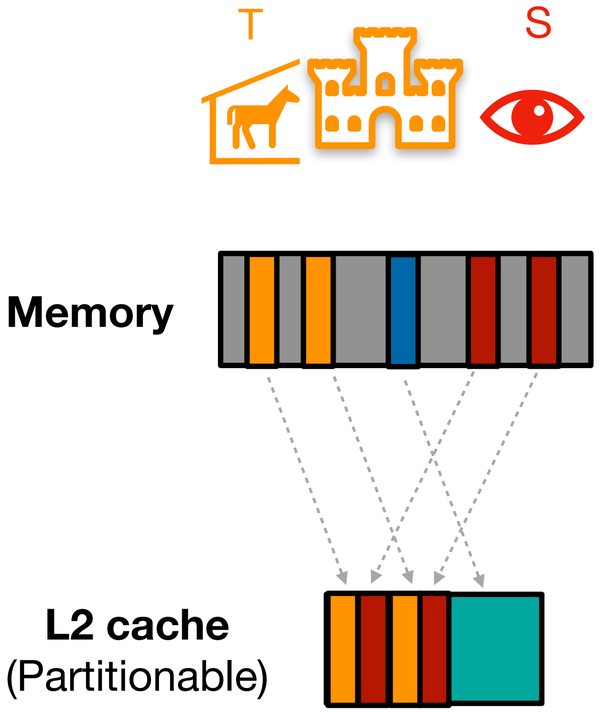
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:



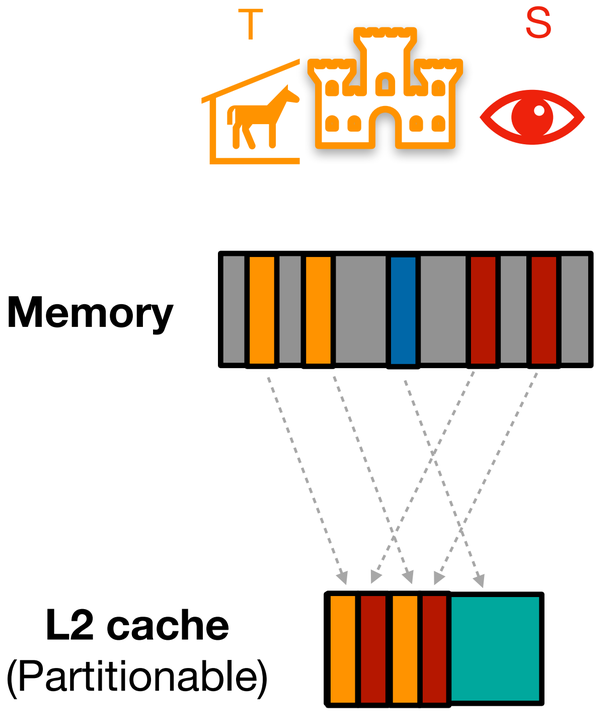
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S



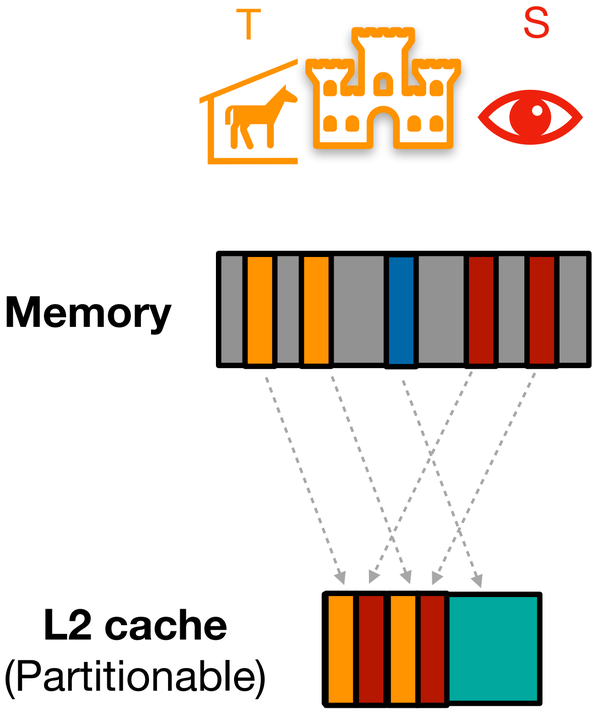
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S



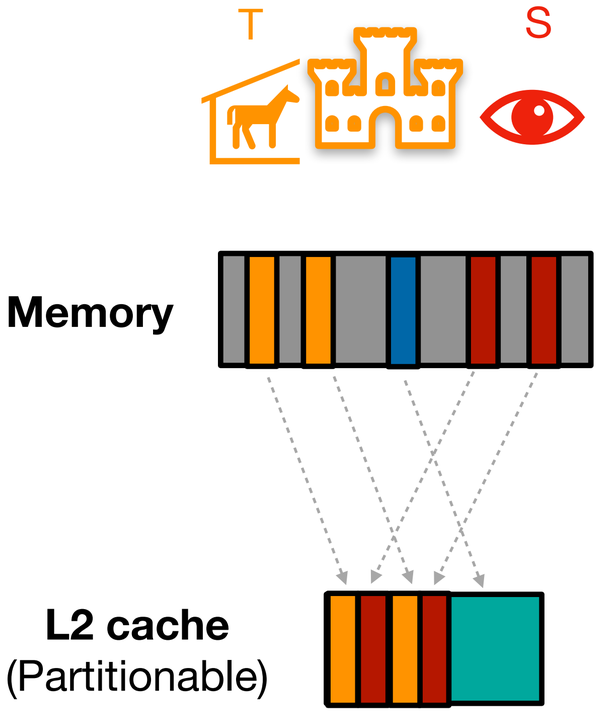
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?



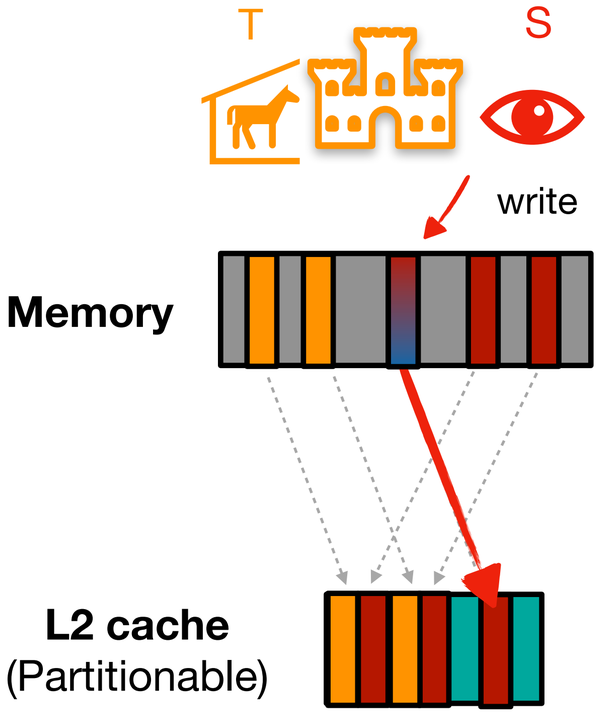
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?



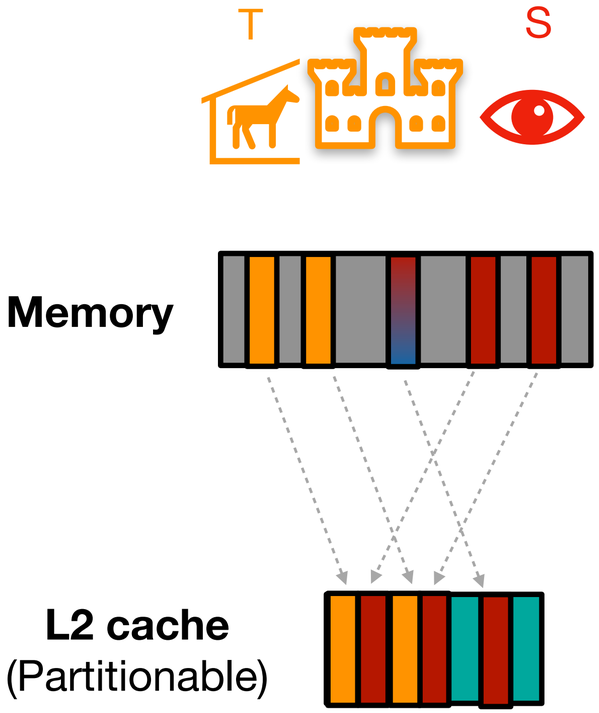
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?



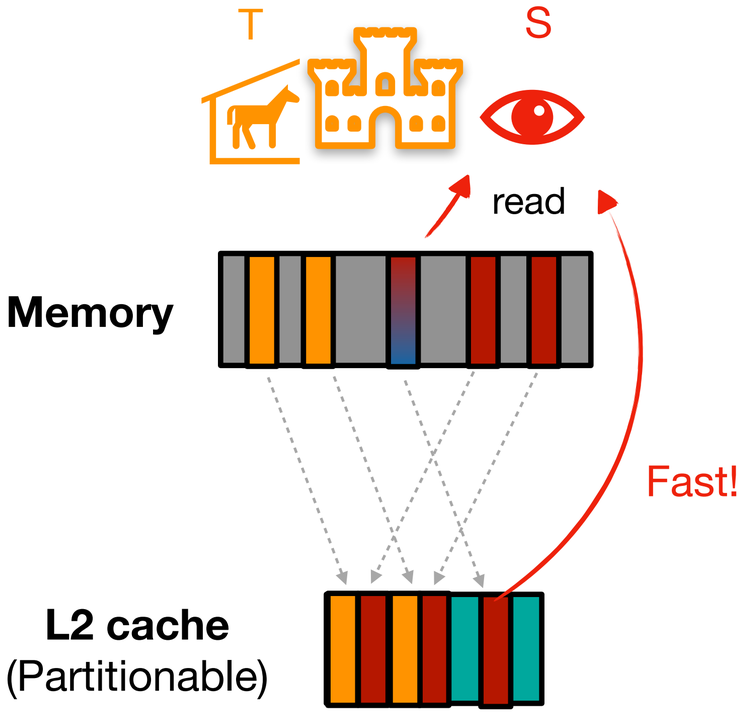
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?



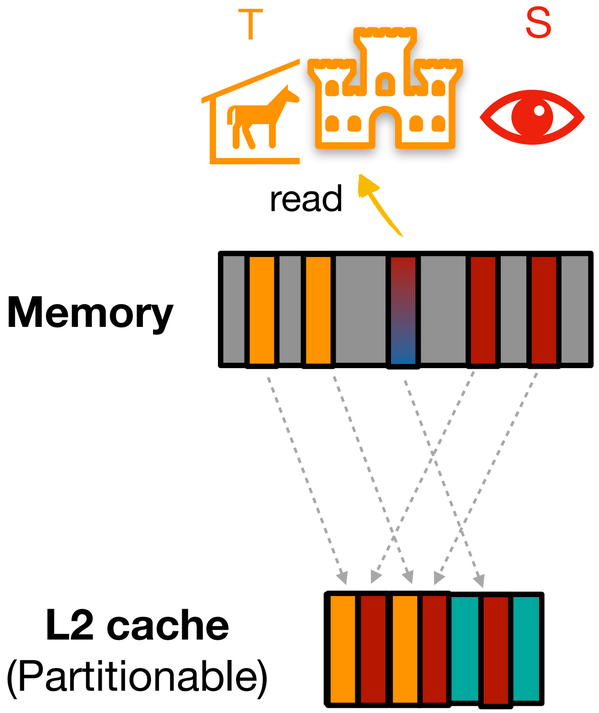
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?



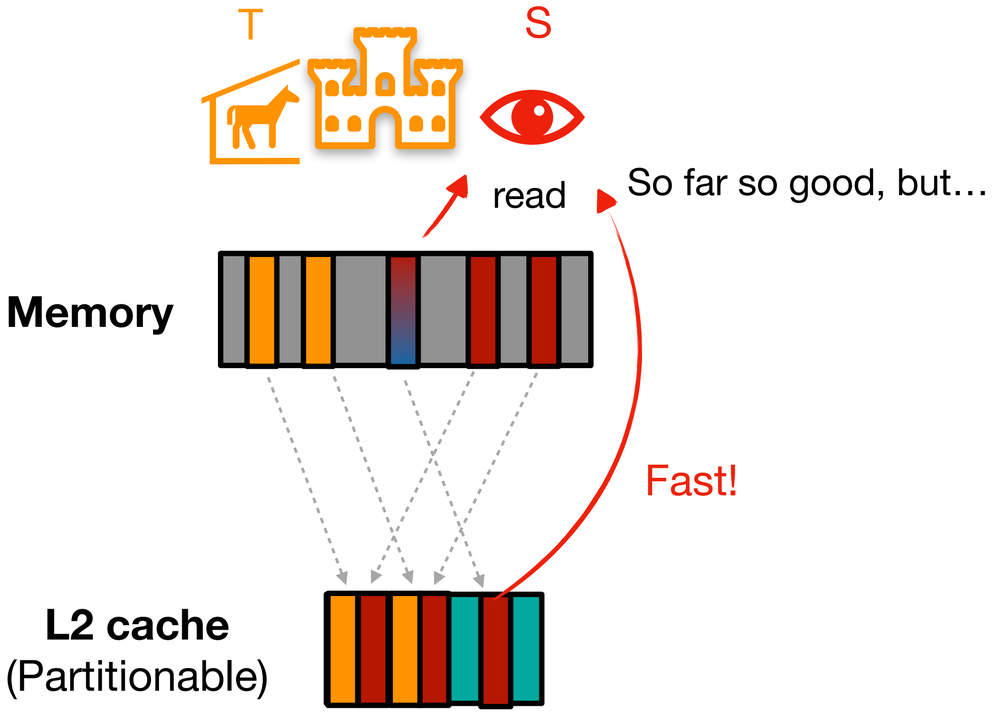
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?



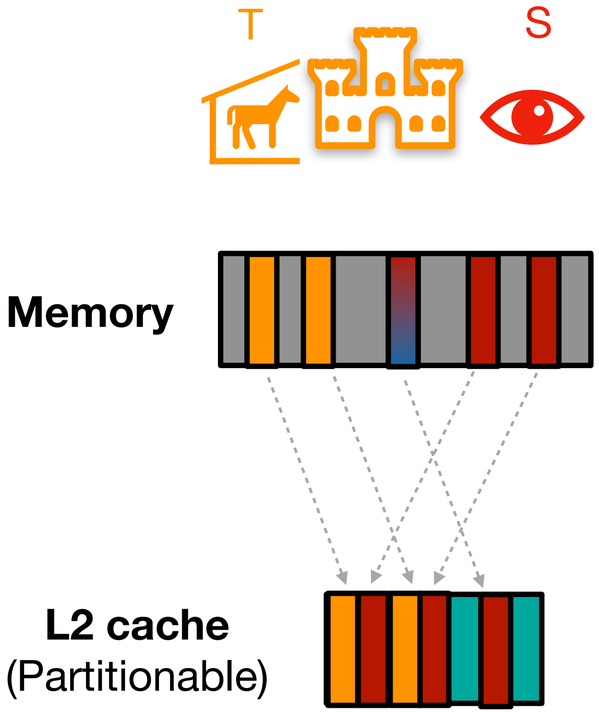
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?
  S sees a timing difference on:



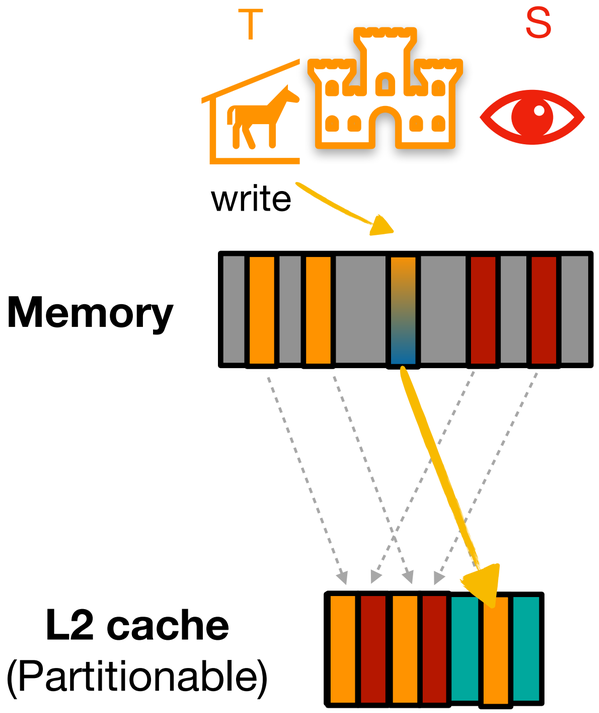
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?
  S sees a timing difference on:
  - Any writes by T



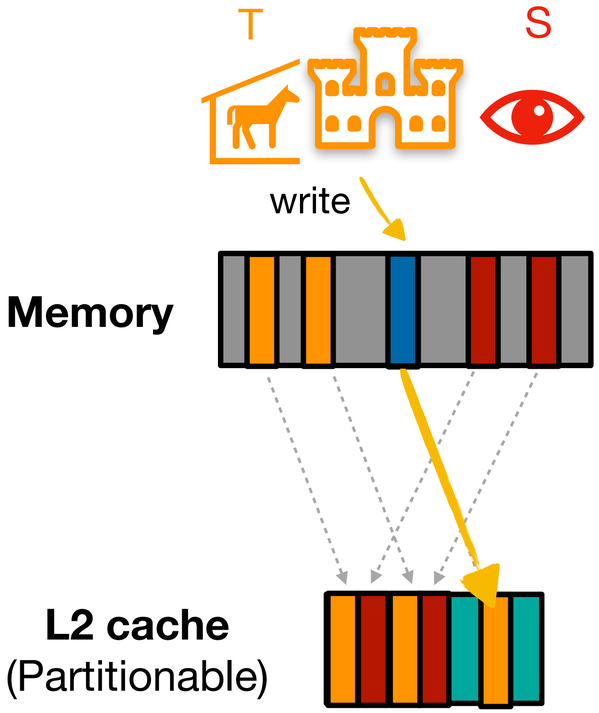
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?
  S sees a timing difference on:
  - Any writes by T (even to wipe the data!)



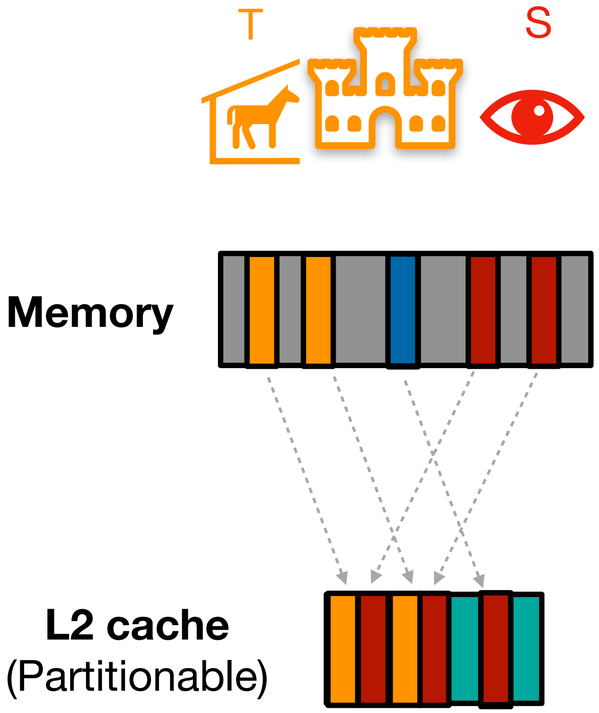
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?
  S sees a timing difference on:
  - Any writes by T (even to wipe the data!)
  - Any reads by T that evict lines (many per page!)



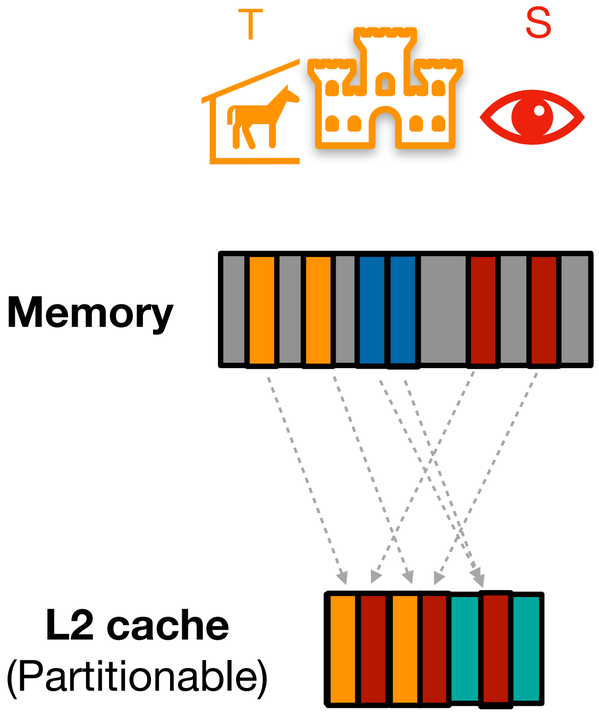
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?
  S sees a timing difference on:
  - Any writes by T (even to wipe the data!)
  - Any reads by T that evict lines (many per page!)



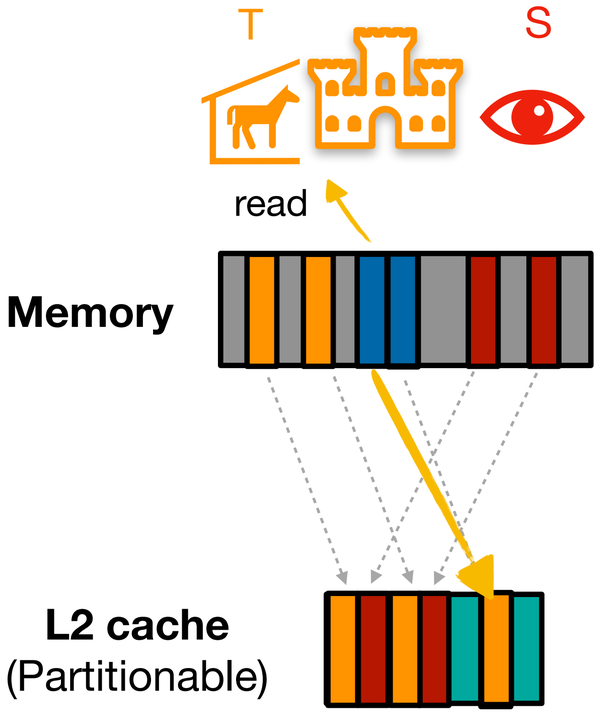
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?
  S sees a timing difference on:
  - Any writes by T (even to wipe the data!)
  - Any reads by T that evict lines (many per page!)



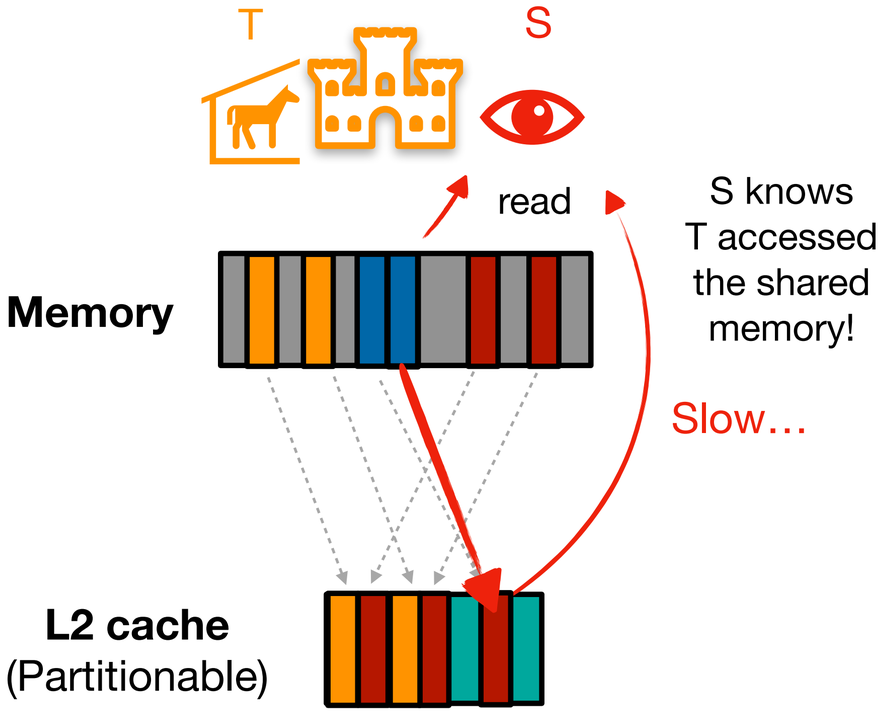
© Rob Sison 2026, CC BY 4.0

What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?
  S sees a timing difference on:
  - Any writes by T (even to wipe the data!)
  - Any reads by T that evict lines (many per page!)



© Rob Sison 2026, CC BY 4.0

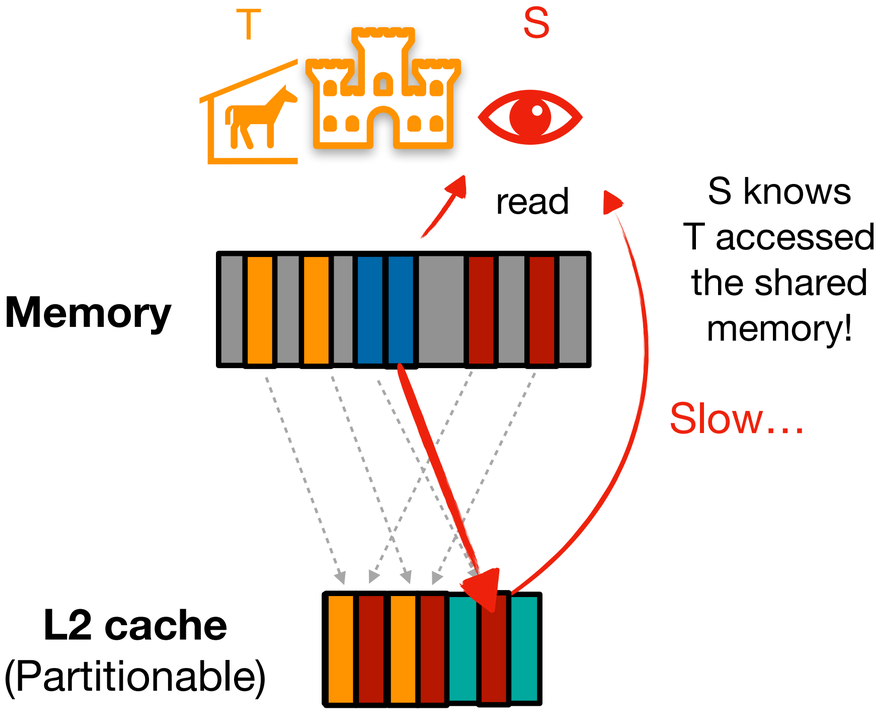
What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?
  S sees a timing difference on:
  - Any writes by T (even to wipe the data!)
  - Any reads by T that evict lines (many per page!)

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



© Rob Sison 2026, CC BY 4.0

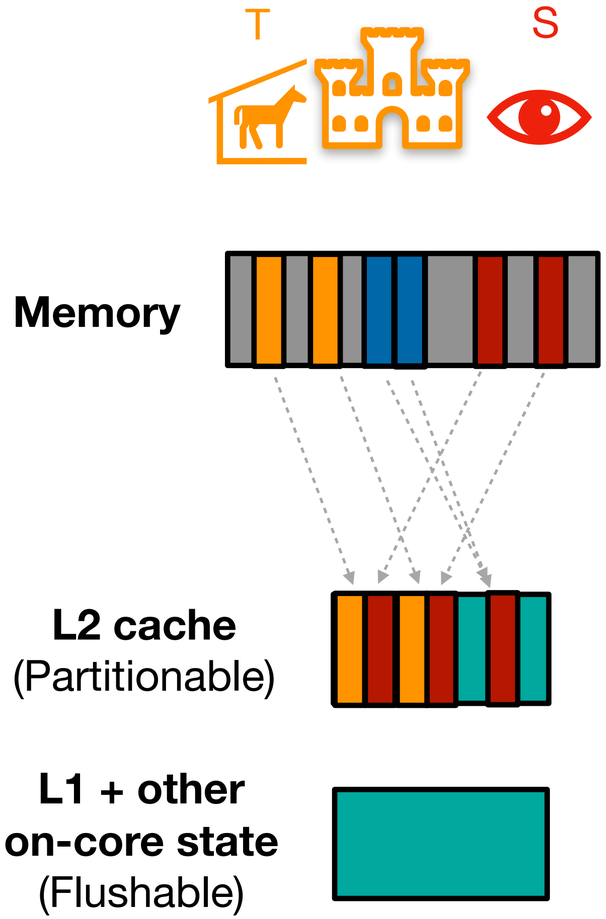
What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?
  S sees a timing difference on:
  - Any writes by T (even to wipe the data!)
  - Any reads by T that evict lines (many per page!)

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



© Rob Sison 2026, CC BY 4.0

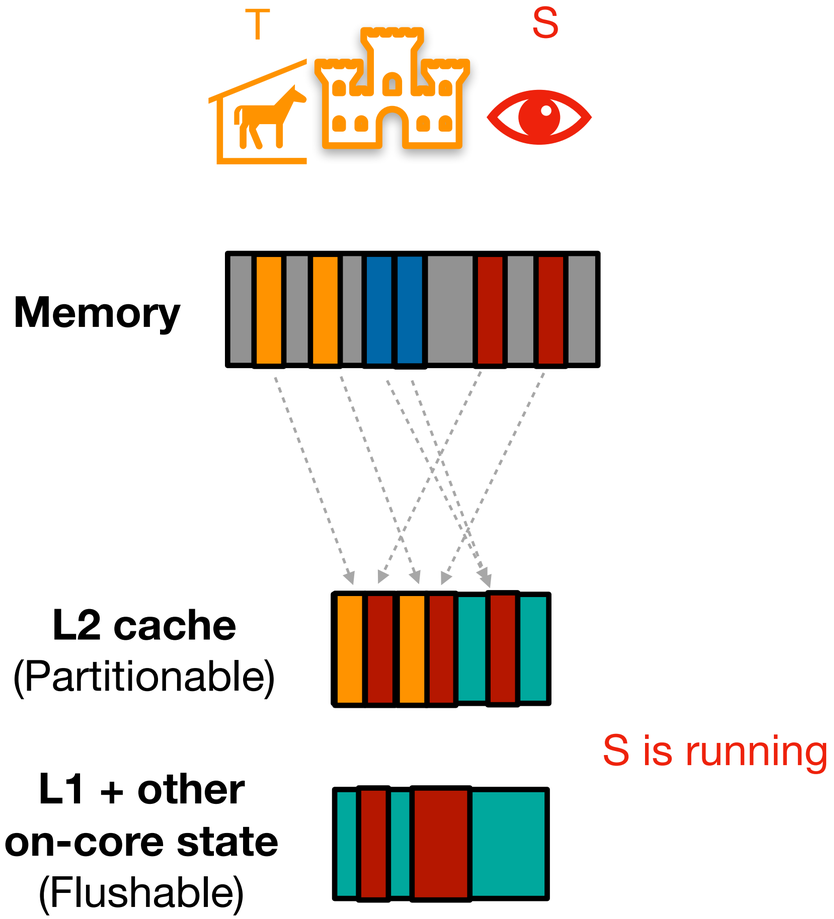
What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?
  S sees a timing difference on:
  - Any writes by T (even to wipe the data!)
  - Any reads by T that evict lines (many per page!)

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



© Rob Sison 2026, CC BY 4.0

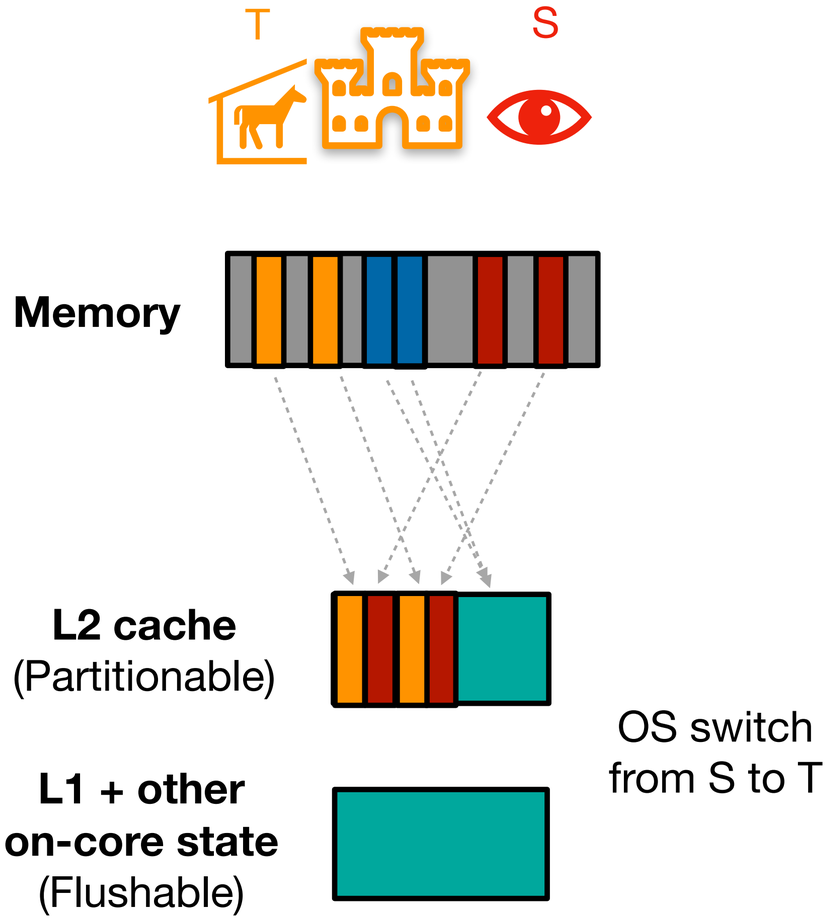
What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?
  S sees a timing difference on:
  - Any writes by T (even to wipe the data!)
  - Any reads by T that evict lines (many per page!)

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



© Rob Sison 2026, CC BY 4.0

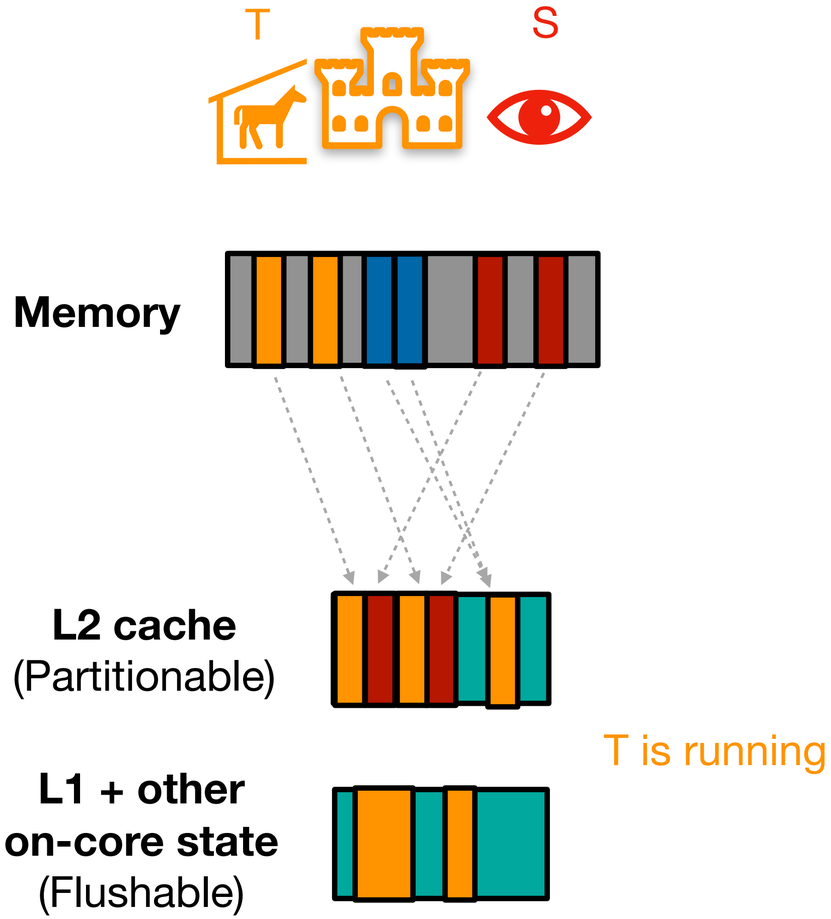
What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?
  S sees a timing difference on:
  - Any writes by T (even to wipe the data!)
  - Any reads by T that evict lines (many per page!)

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



© Rob Sison 2026, CC BY 4.0

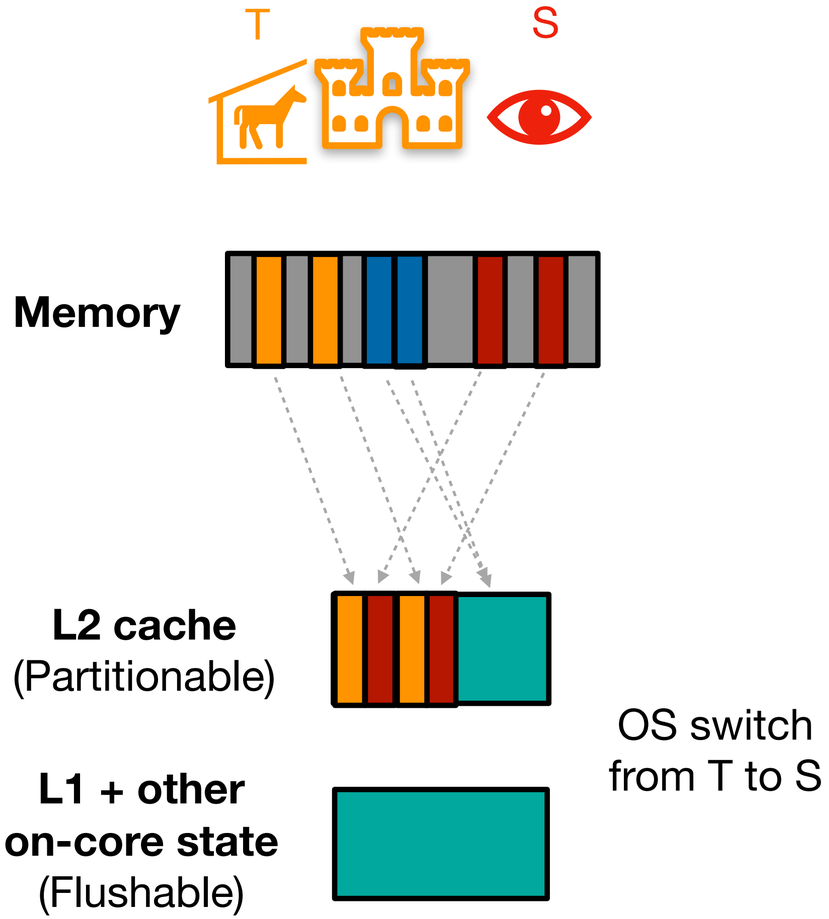
What are time-protected communications?

- So far, we've discussed verifying time protection in seL4 with *strict isolation* policies,

  i.e. ∀A B. A ~/> B

- Let's consider a *data diode* policy on infoflow via a page of **shared memory**:
  - One direction allowed: T <~ S
  - Other direction *disallowed*: T ~/> S
- Can we do it with a partitionable L2 cache?
  S sees a timing difference on:
  - Any writes by T (even to wipe the data!)
  - Any reads by T that evict lines (many per page!)

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



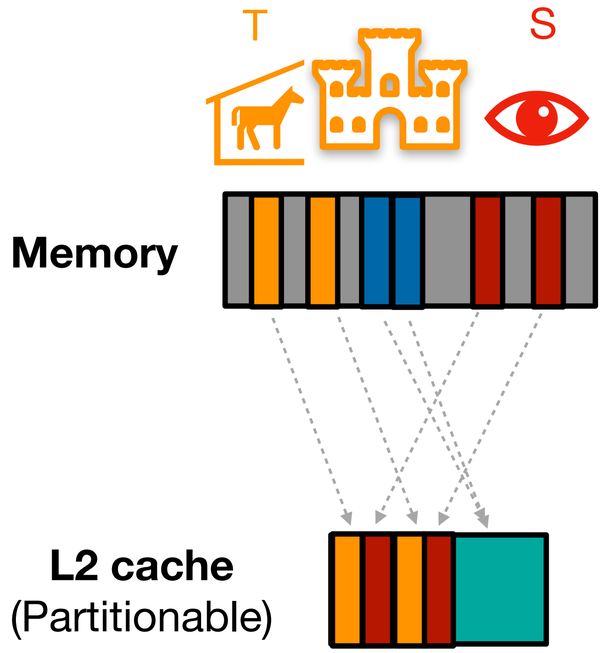
© Rob Sison 2026, CC BY 4.0

Time-protected communications in seL4

seL4 on RISC-V

D

Implement time-protected cross-domain communications



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

© Rob Sison 2026, CC BY 4.0

Time-protected communications in seL4

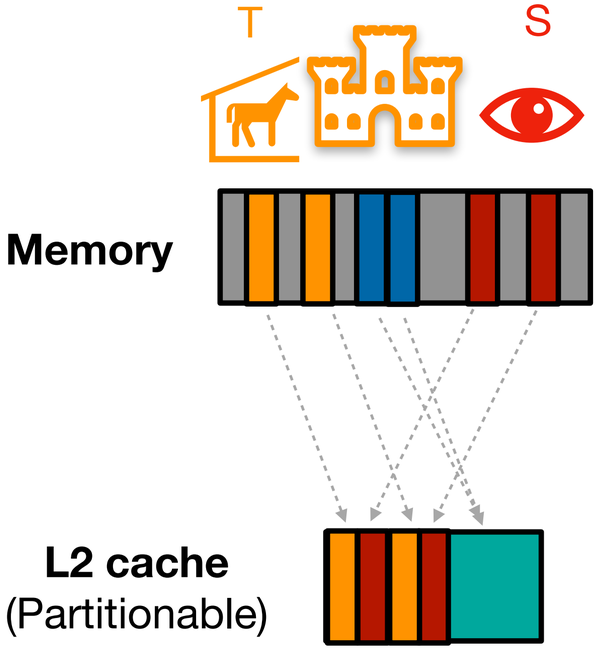
seL4 on RISC-V

D

Implement 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

© Rob Sison 2026, CC BY 4.0

Time-protected communications in seL4

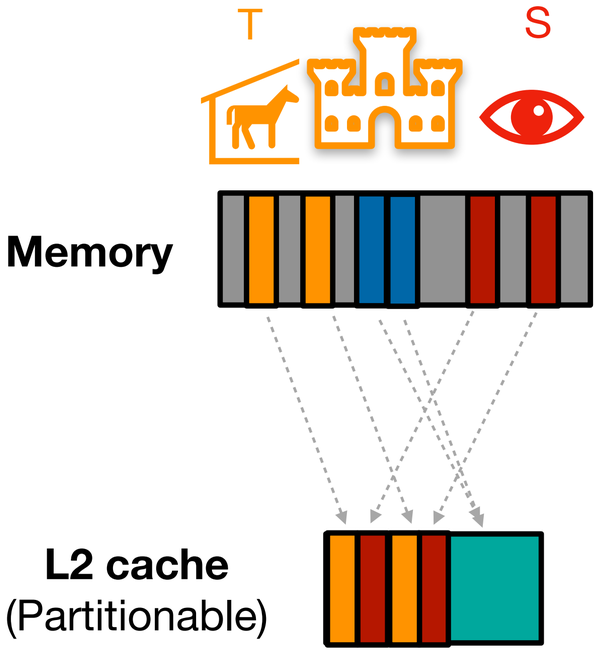
seL4 on RISC-V

D

Implement 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)

Varun Sethu
BE Thesis



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

© Rob Sison 2026, CC BY 4.0

Time-protected communications in seL4

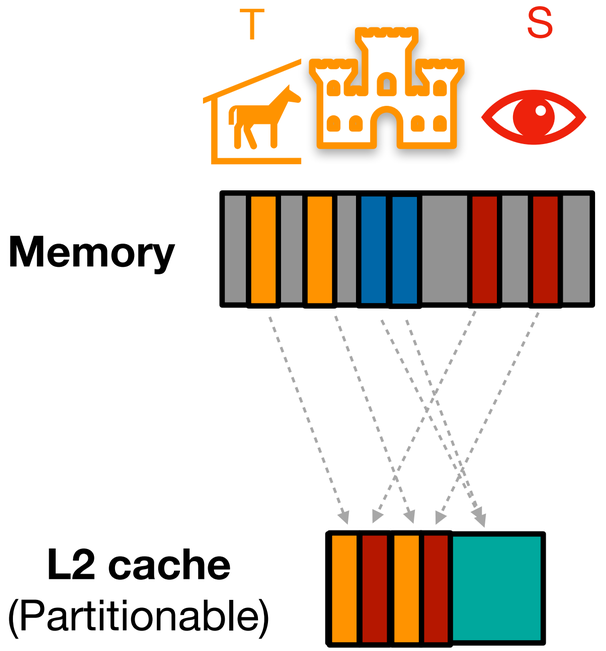
seL4 on RISC-V

D

Implement 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

© Rob Sison 2026, CC BY 4.0

Time-protected communications in seL4

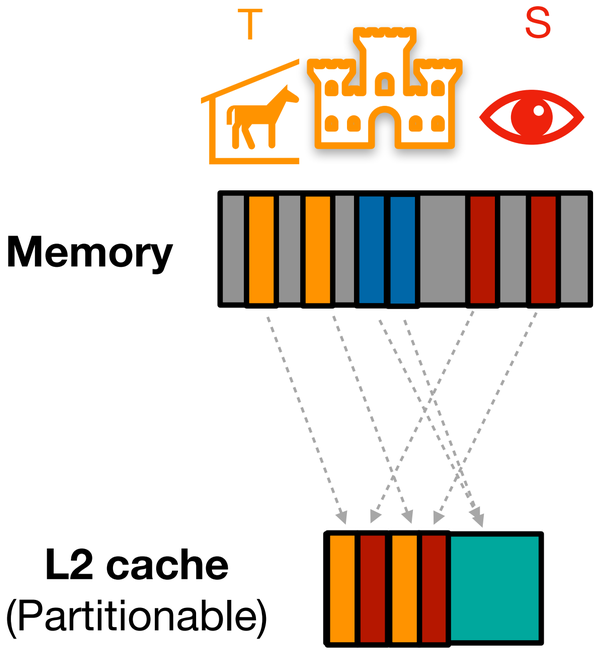
seL4 on RISC-V

D

Implement 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 “flush” of L2 via prefetching (still leaks)

Varun Sethu
BE Thesis



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

© Rob Sison 2026, CC BY 4.0

Time-protected communications in seL4

seL4 on RISC-V

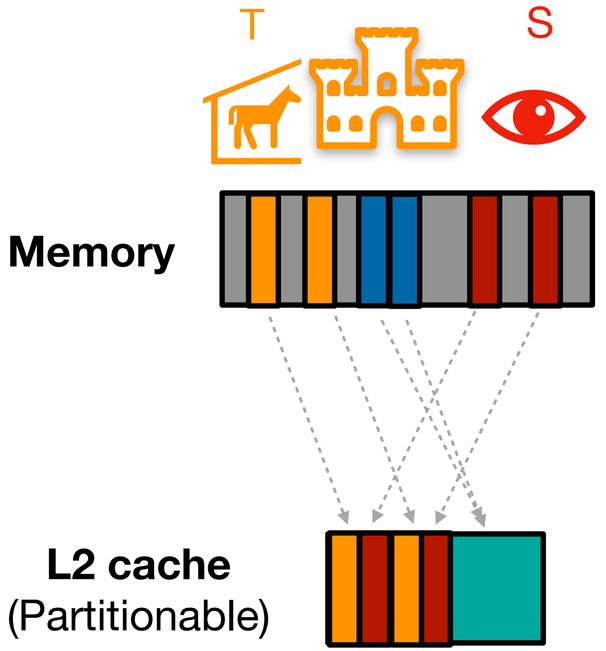
D

Implement 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 “flush” of L2 via prefetching (still leaks)

Varun Sethu
BE Thesis

Continuing on from



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

© Rob Sison 2026, CC BY 4.0

Time-protected communications in seL4

seL4 on RISC-V

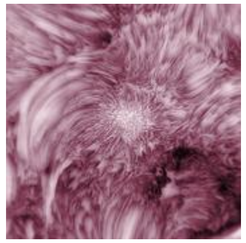
D

Implement time-protected cross-domain communications

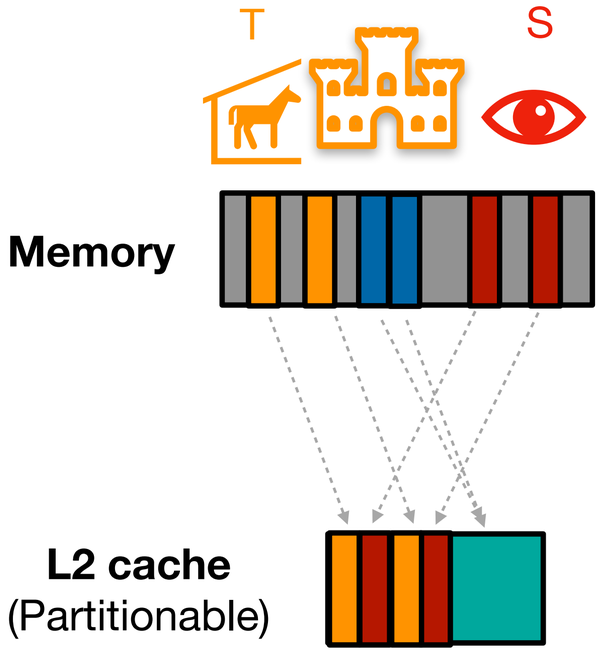
- [Sethu 2025] comprehensively evaluated “legacy” implementation options on seL4, with the L2 partitioned via *cache colouring*:
  - Copy between non-shared memory (slow)
  - Dedicate colours for shared memory (wastes space)
  - Targeted “flush” of L2 via prefetching (still leaks)
- Now we're using *dynamically 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

© Rob Sison 2026, CC BY 4.0

Time-protected communications in seL4

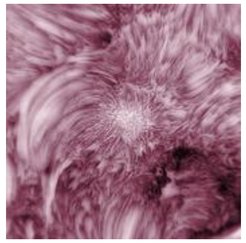
seL4 on RISC-V

Implement time-protected cross-domain communications

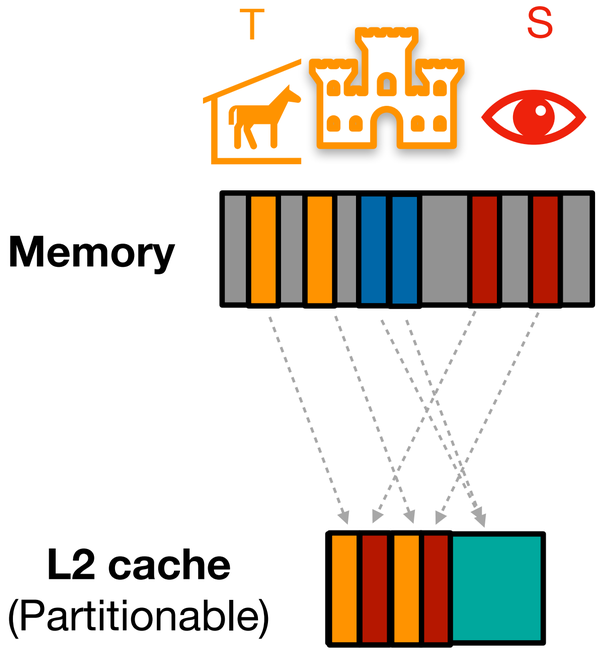
- [Sethu 2025] comprehensively evaluated “legacy” implementation options on seL4, with the L2 partitioned via *cache colouring*:
  - Copy between non-shared memory (slow)
  - Dedicate colours for shared memory (wastes space)
  - Targeted “flush” of L2 via prefetching (still leaks)
- Now we're using *dynamically partitionable last-level cache* (DPLLC) i.e. dedicated HW support on RISC-V Cheshire
  - Supports targeted flush of partitions

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

© Rob Sison 2026, CC BY 4.0

Time-protected communications in seL4

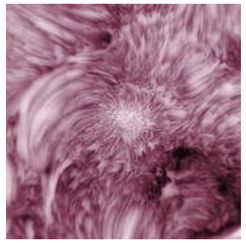
seL4 on RISC-V

Implement time-protected cross-domain communications

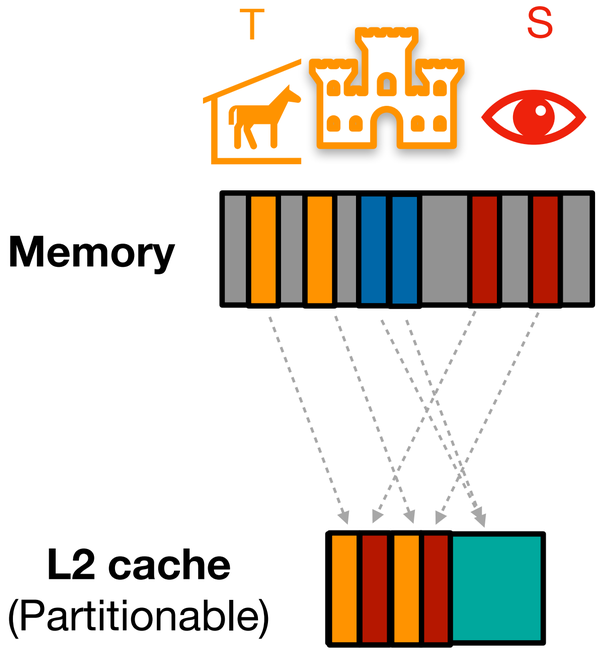
- [Sethu 2025] comprehensively evaluated “legacy” implementation options on seL4, with the L2 partitioned via *cache colouring*:
  - Copy between non-shared memory (slow)
  - Dedicate colours for shared memory (wastes space)
  - Targeted “flush” of L2 via prefetching (still leaks)
- Now we're using *dynamically 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

© Rob Sison 2026, CC BY 4.0

Verifying seL4 time-protected communications

seL4 on RISC-V

FM'23: Define time protection for OS kernels

Implement time-protected cross-domain communications

Verify time-protected cross-domain communications

© Rob Sison 2026, CC BY 4.0

Verifying seL4 time-protected communications

seL4 on RISC-V

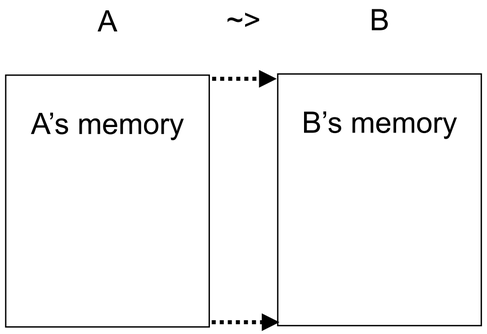
FM'23: Define time protection for OS kernels

D

Implement time-protected cross-domain communications

E

Verify time-protected cross-domain communications



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

© Rob Sison 2026, CC BY 4.0

Verifying seL4 time-protected communications

seL4 on RISC-V

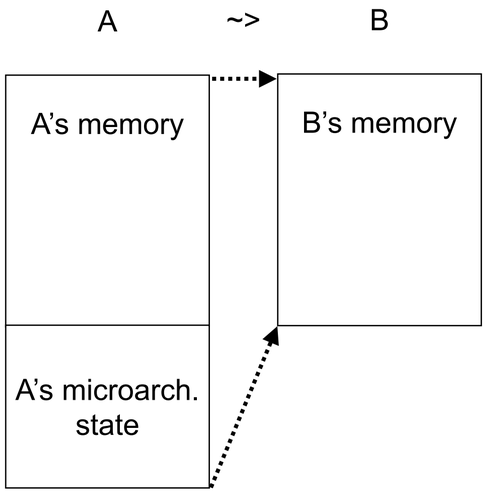
FM'23: Define time protection for OS kernels

D

Implement time-protected cross-domain communications

E

Verify time-protected cross-domain communications



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

© Rob Sison 2026, CC BY 4.0

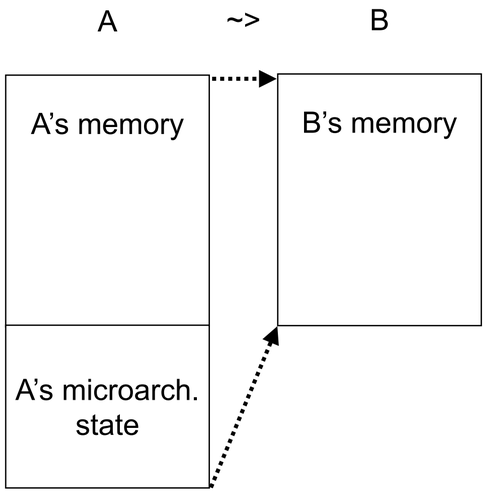
Verifying seL4 time-protected communications

seL4 on RISC-V

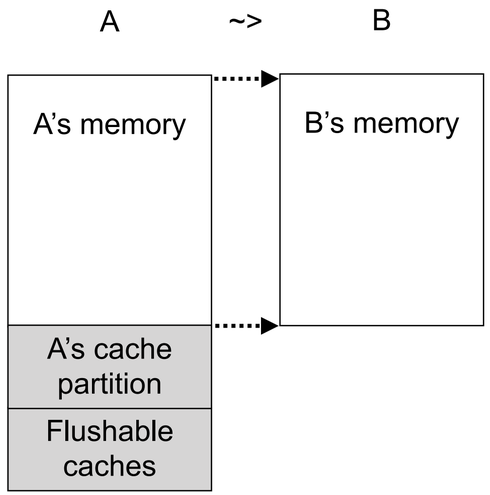
FM'23: Define time protection for OS kernels

D: Implement time-protected cross-domain communications

E: Verify time-protected cross-domain communications



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



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

© Rob Sison 2026, CC BY 4.0

Verifying seL4 time-protected communications

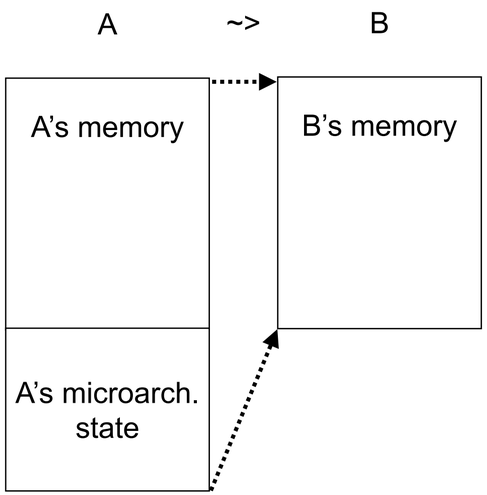
seL4 on RISC-V

FM'23: Define time protection for OS kernels

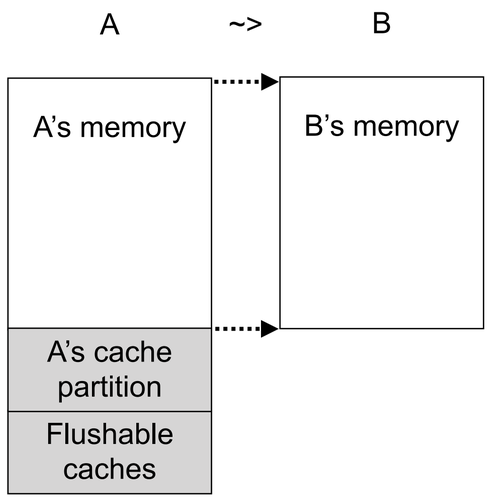
D Implement time-protected cross-domain communications

E Verify time-protected cross-domain communications

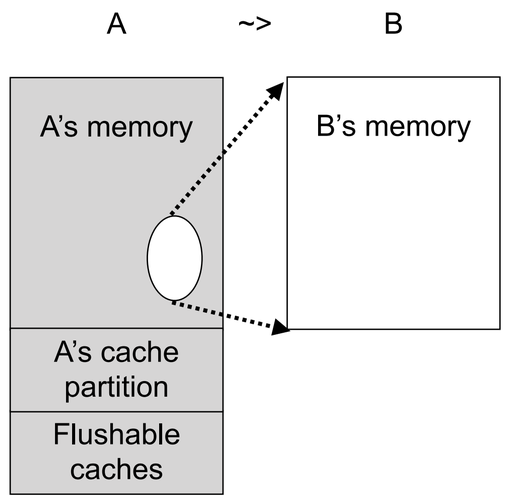
Thanks to Australian Research Council funding



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

© Rob Sison 2026, CC BY 4.0

Verifying seL4 time-protected communications

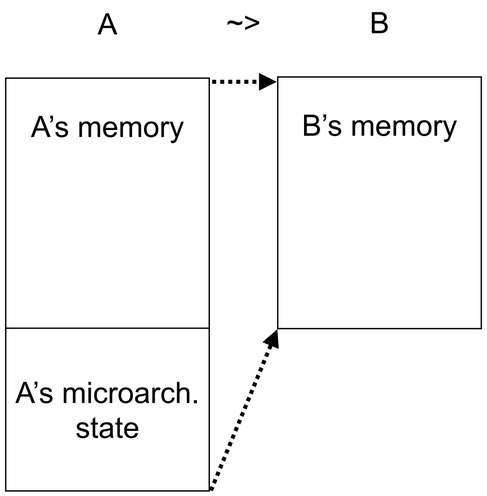
seL4 on RISC-V

FM'23: Define time protection for OS kernels

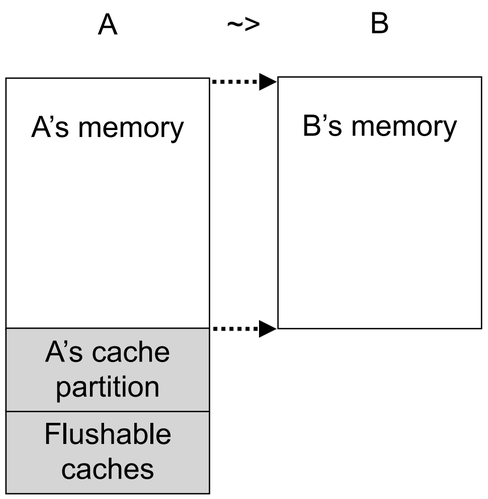
D: Implement time-protected cross-domain communications

E: Verify time-protected cross-domain communications

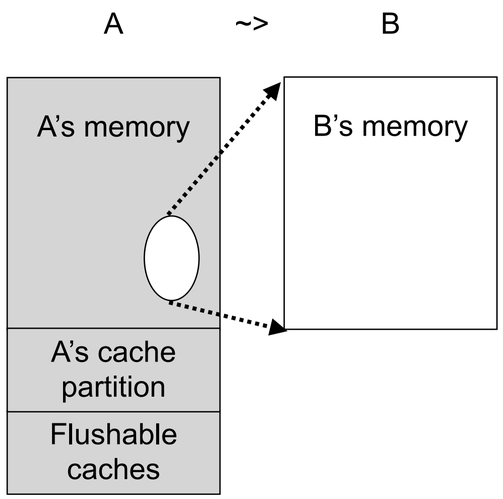
Thanks to funding from cyberagentur



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

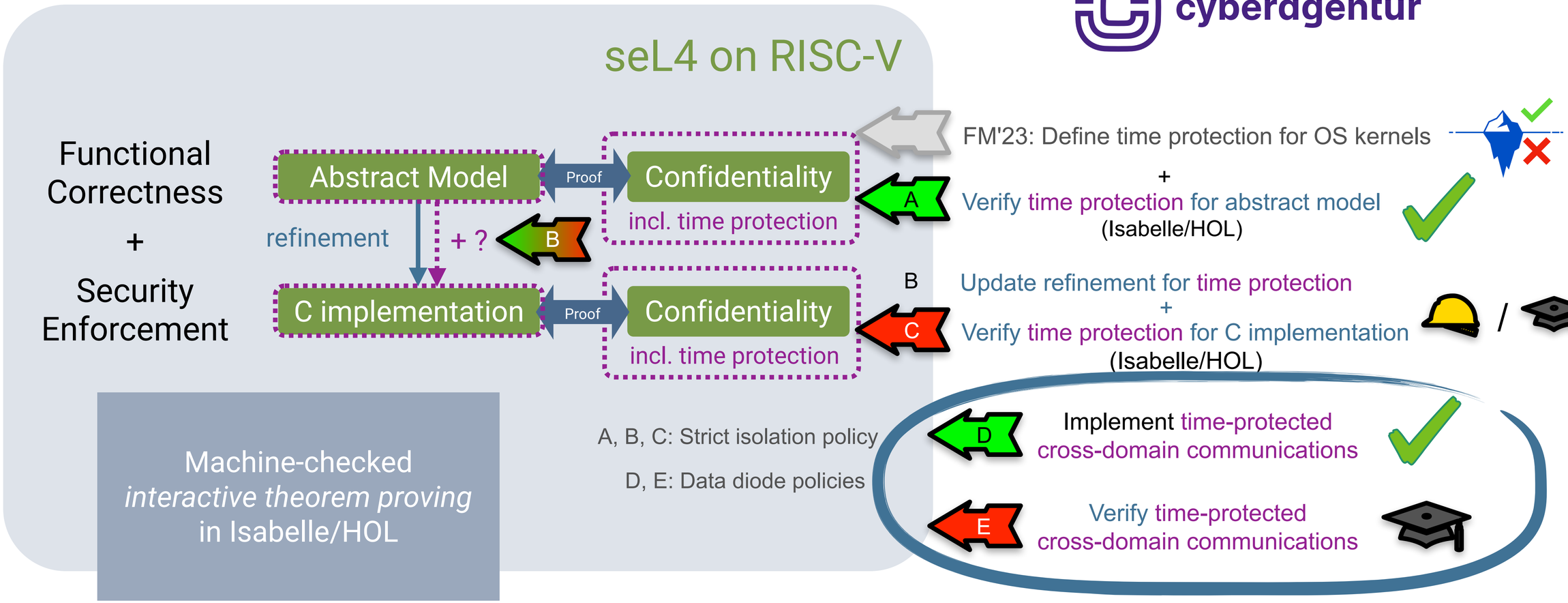
© Rob Sison 2026, CC BY 4.0

Time protection verification for seL4

The status today:

Thanks to funding from

cyberagentur



© Rob Sison 2026, CC BY 4.0

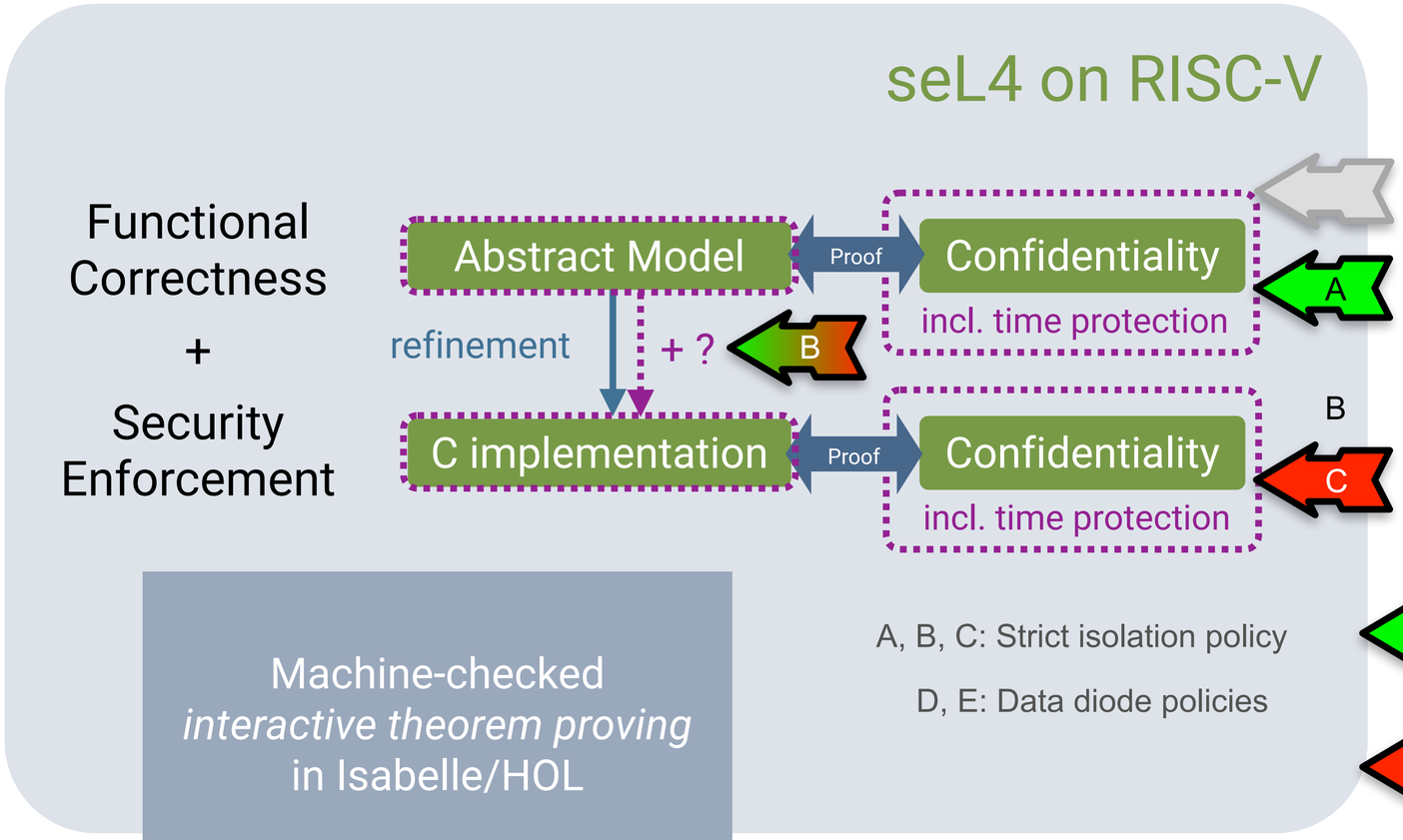
Time protection verification for seL4

The status today:

Thank you!
Q&A

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

© Rob Sison 2026, CC BY 4.0