School of Computer Science & Engineering

Trustworthy Systems Group

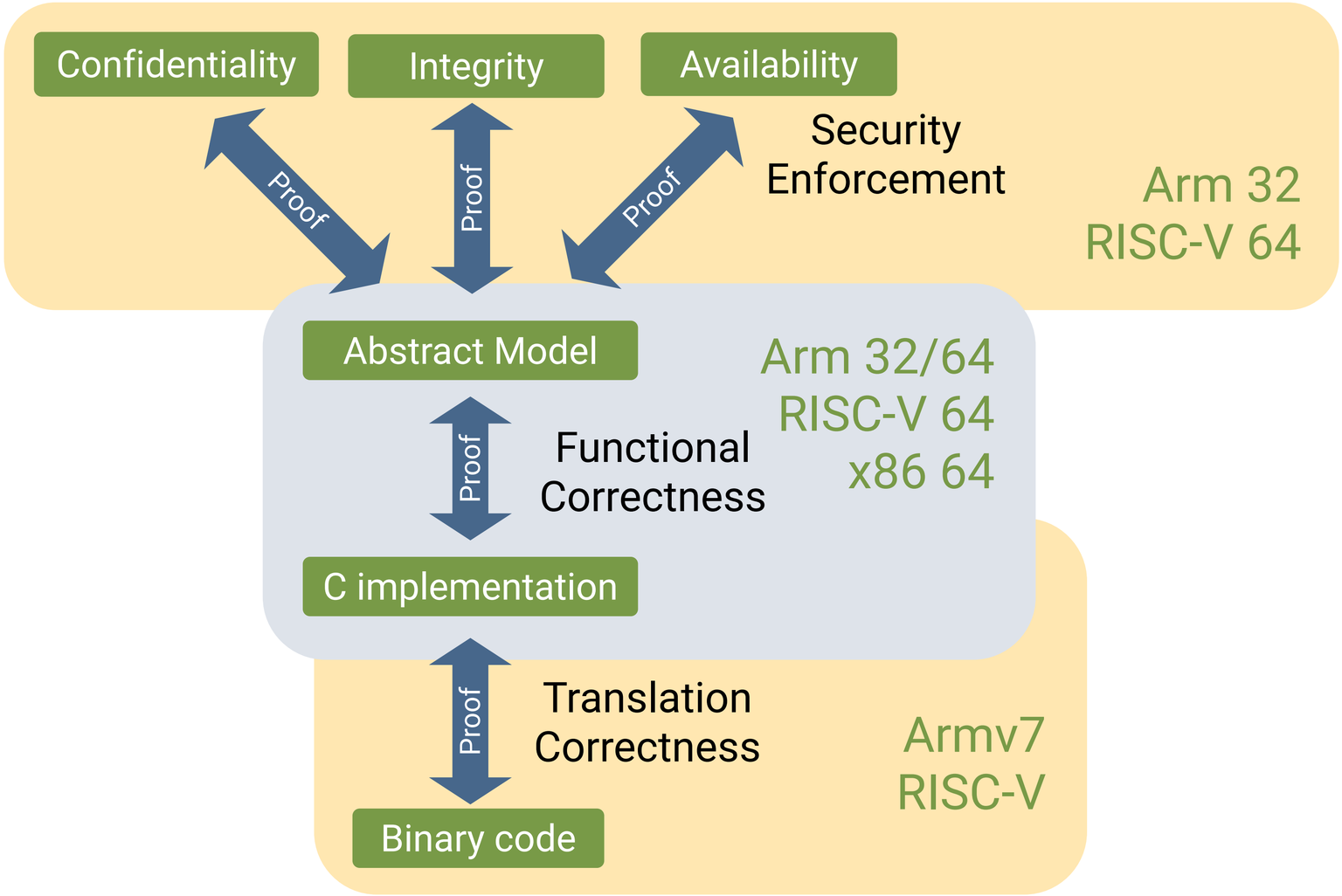
Two new frontiers for seL4's refinement proofs: Time protection and OS service verification

Dr Robert Sison

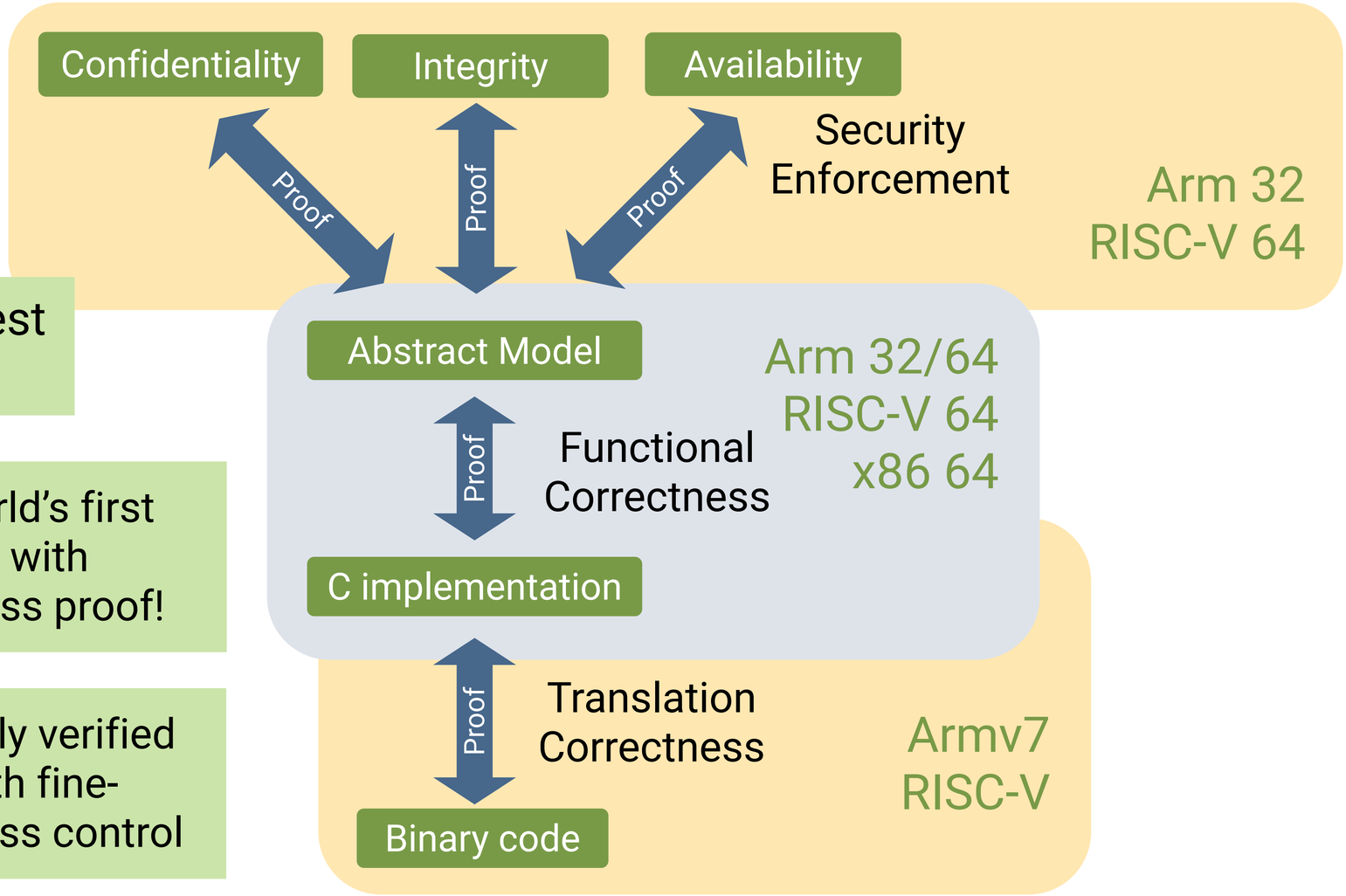
Senior Research Associate, UNSW Sydney

r.sison@unsw.edu.au

What is seL4?



What is seL4?

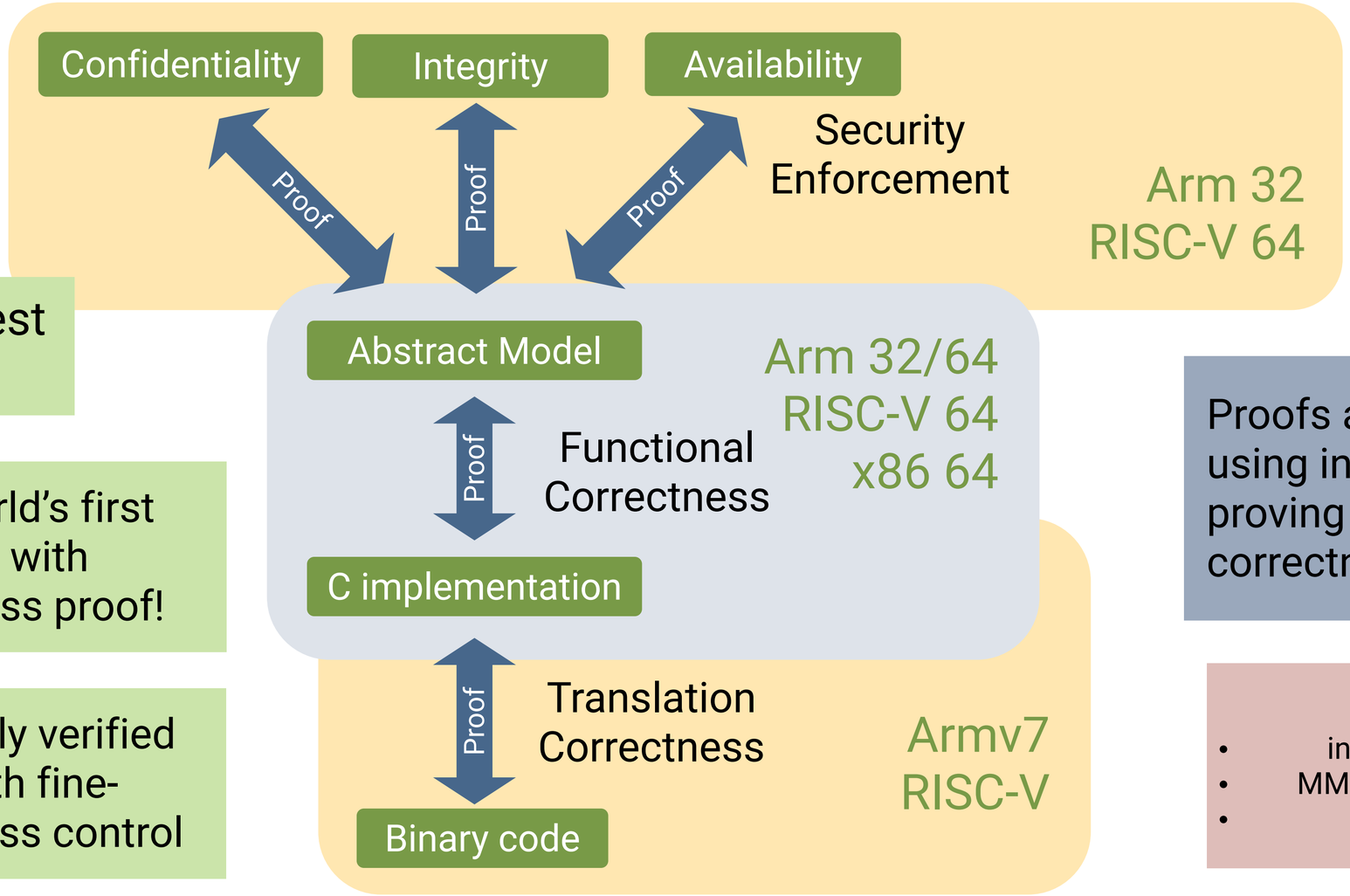


World's fastest microkernel!

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

seL4: Still only verified OS kernel with fine-grained access control

What is seL4?



World's fastest microkernel!

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

seL4: Still only verified OS kernel with fine-grained access control

Proofs are machine-checked, using interactive theorem proving (translation correctness fully automated)

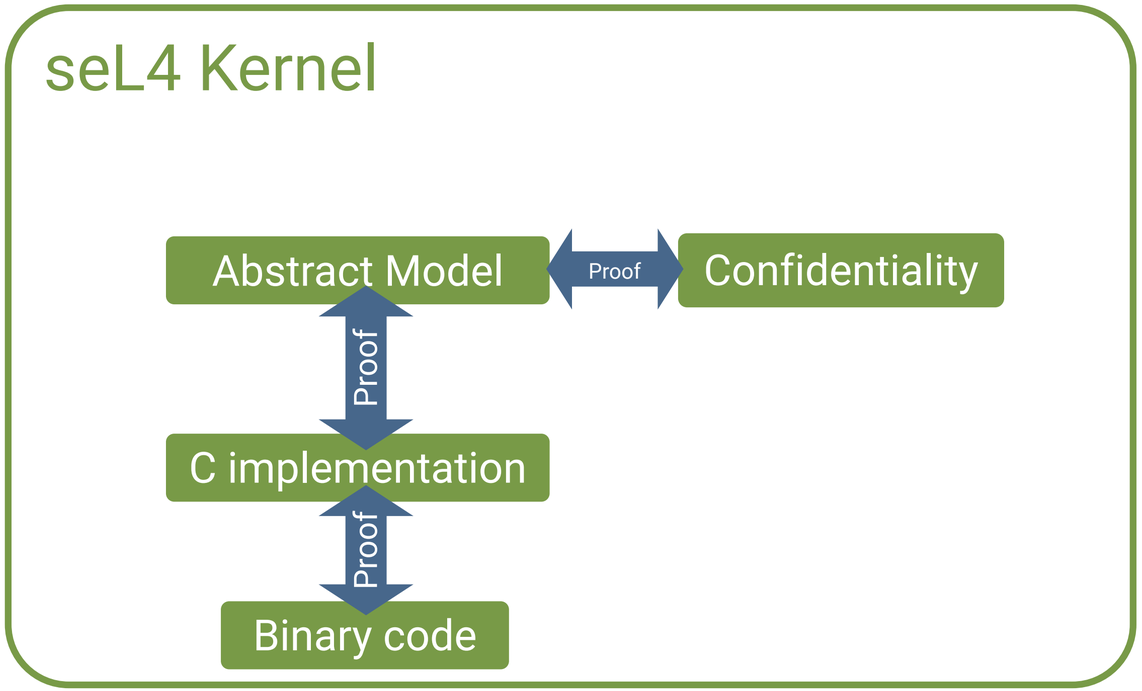
Present limitations
- initialisation code not verified
- MMU, caches modelled abstractly
- Multicore not yet verified

Two new frontiers for seL4 refinement

Functional Correctness

+

Security Enforcement

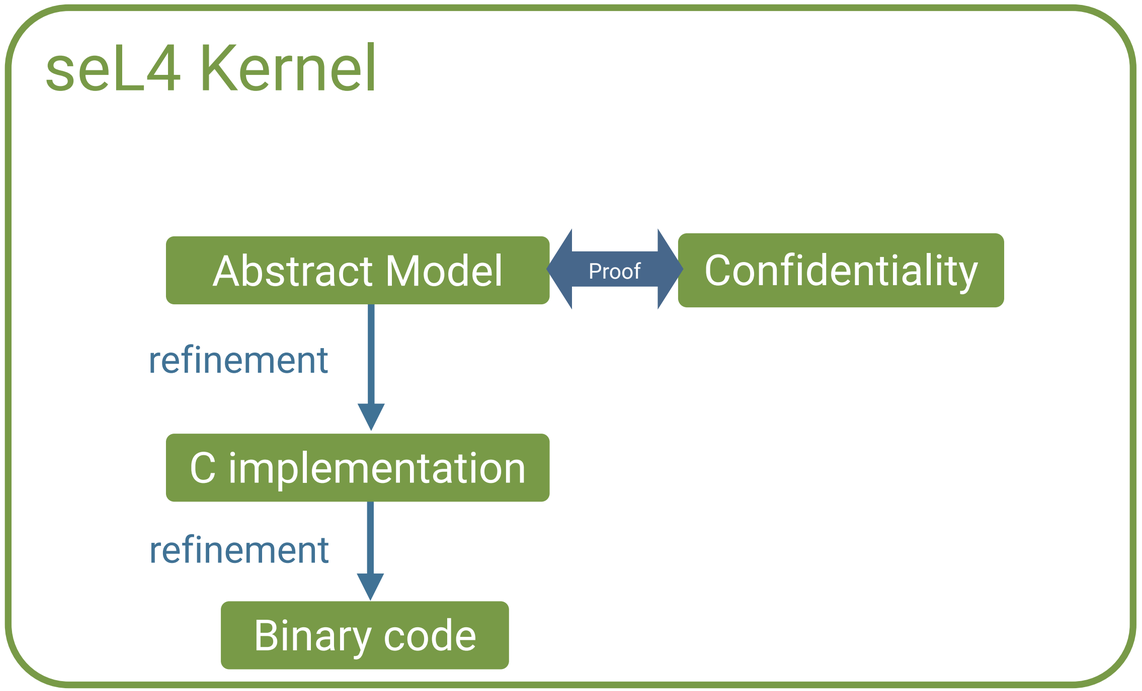


Two new frontiers for seL4 refinement

Functional Correctness

+

Security Enforcement

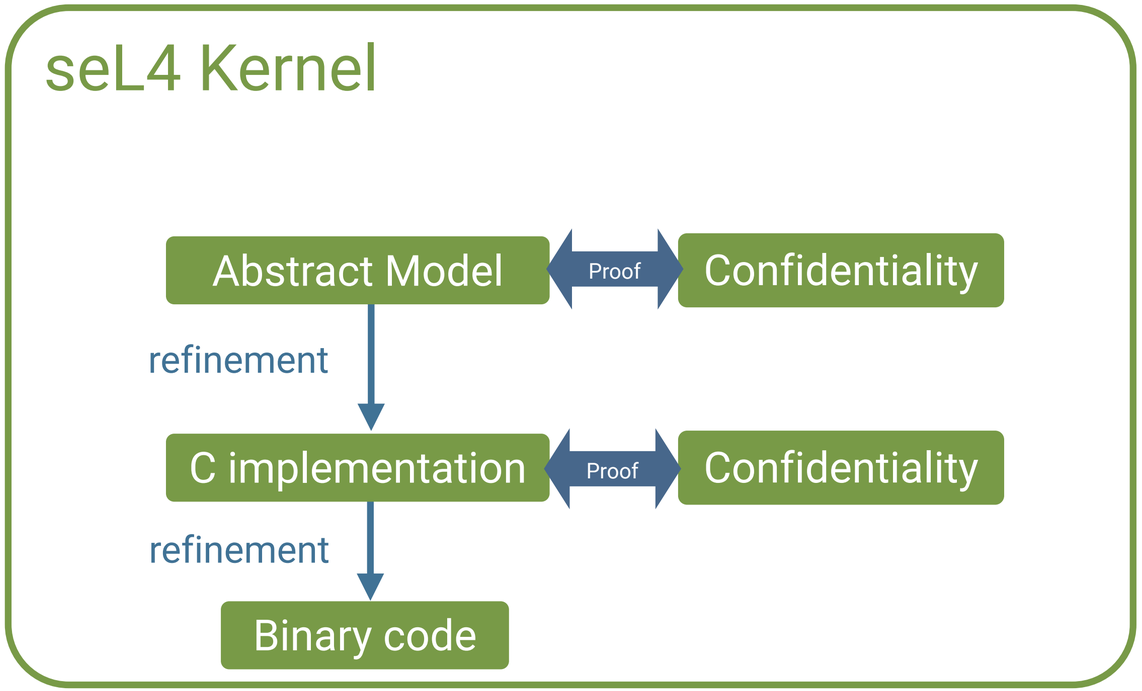


Two new frontiers for seL4 refinement

Functional Correctness

+

Security Enforcement

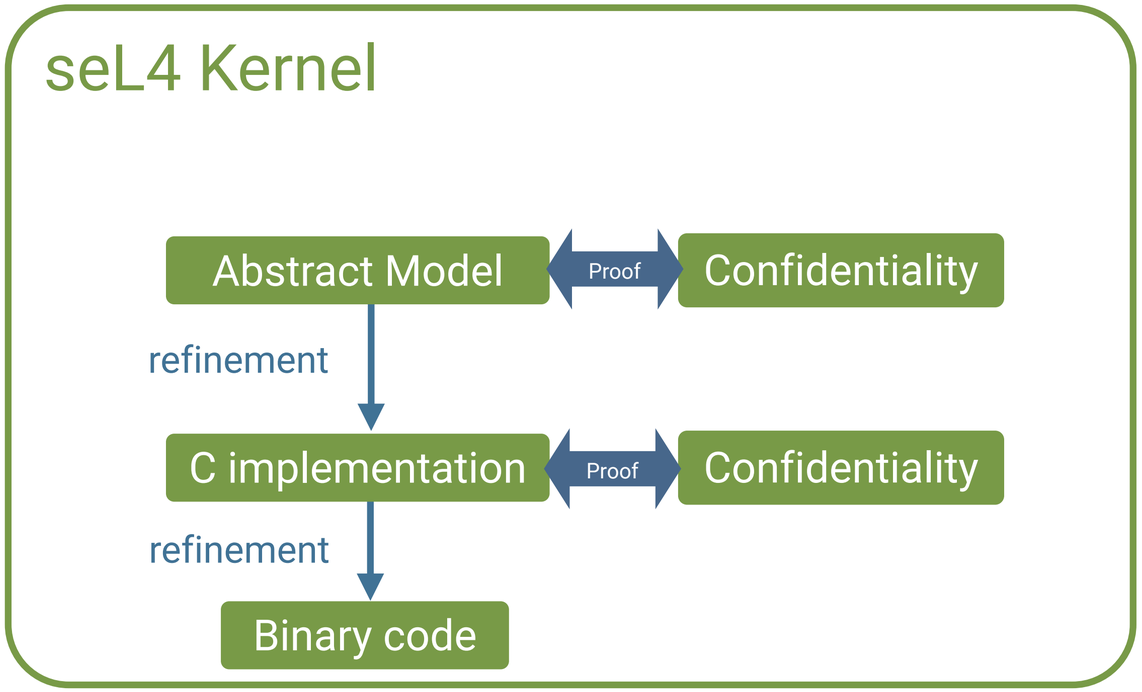


Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness of OS services

Security Enforcement

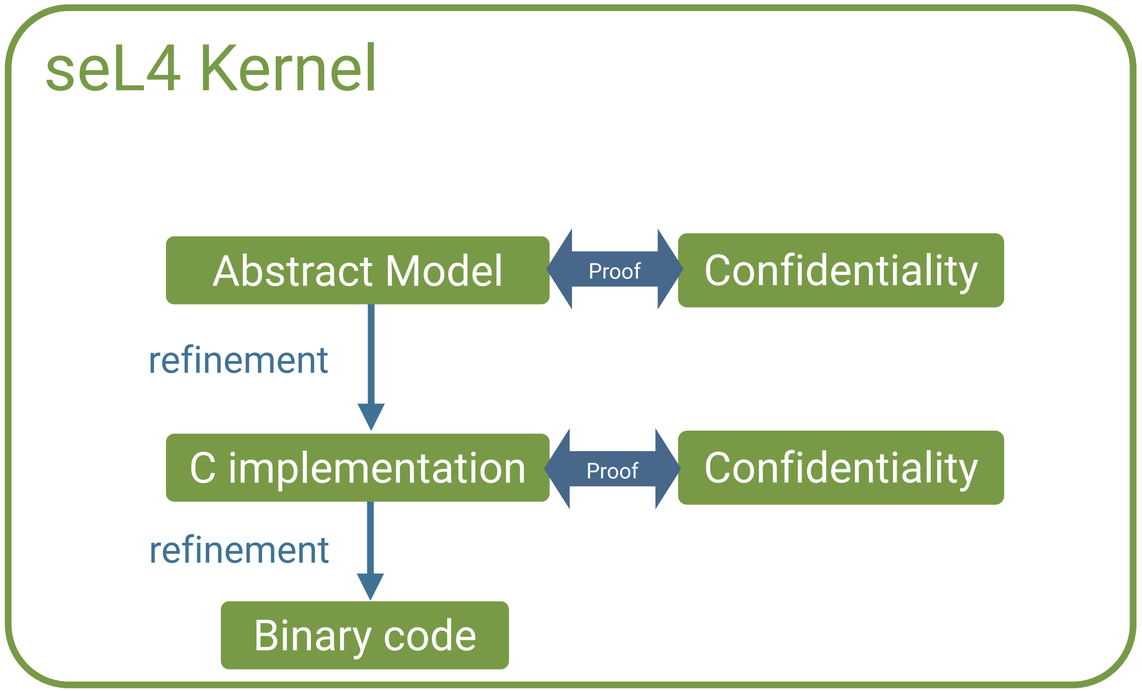


Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness of OS services

Security Enforcement

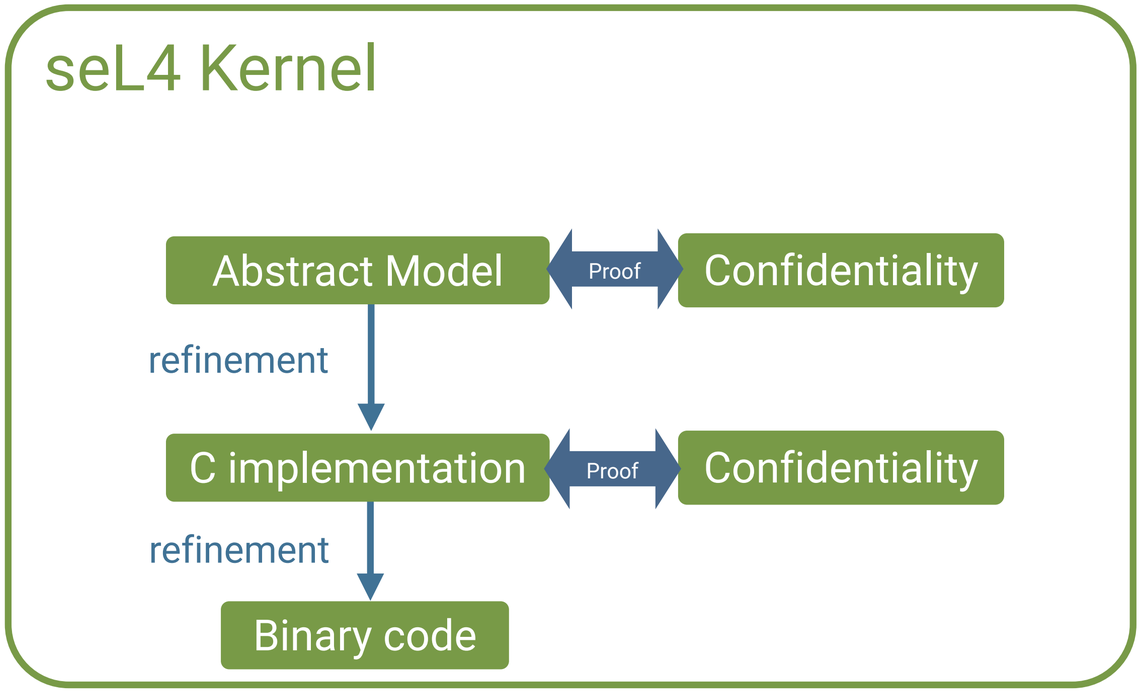


Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness of OS services

Security Enforcement

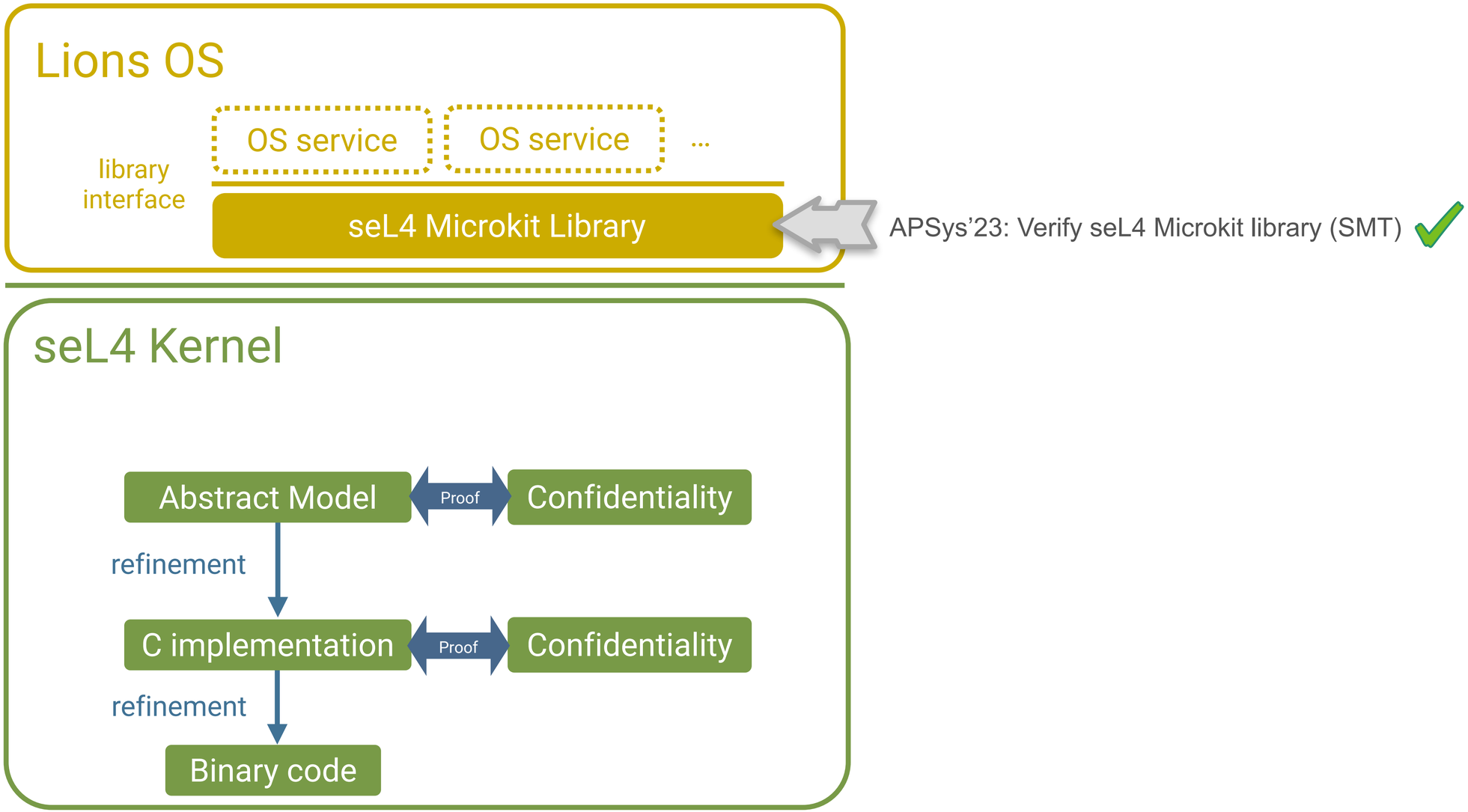


Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness of OS services

Security Enforcement

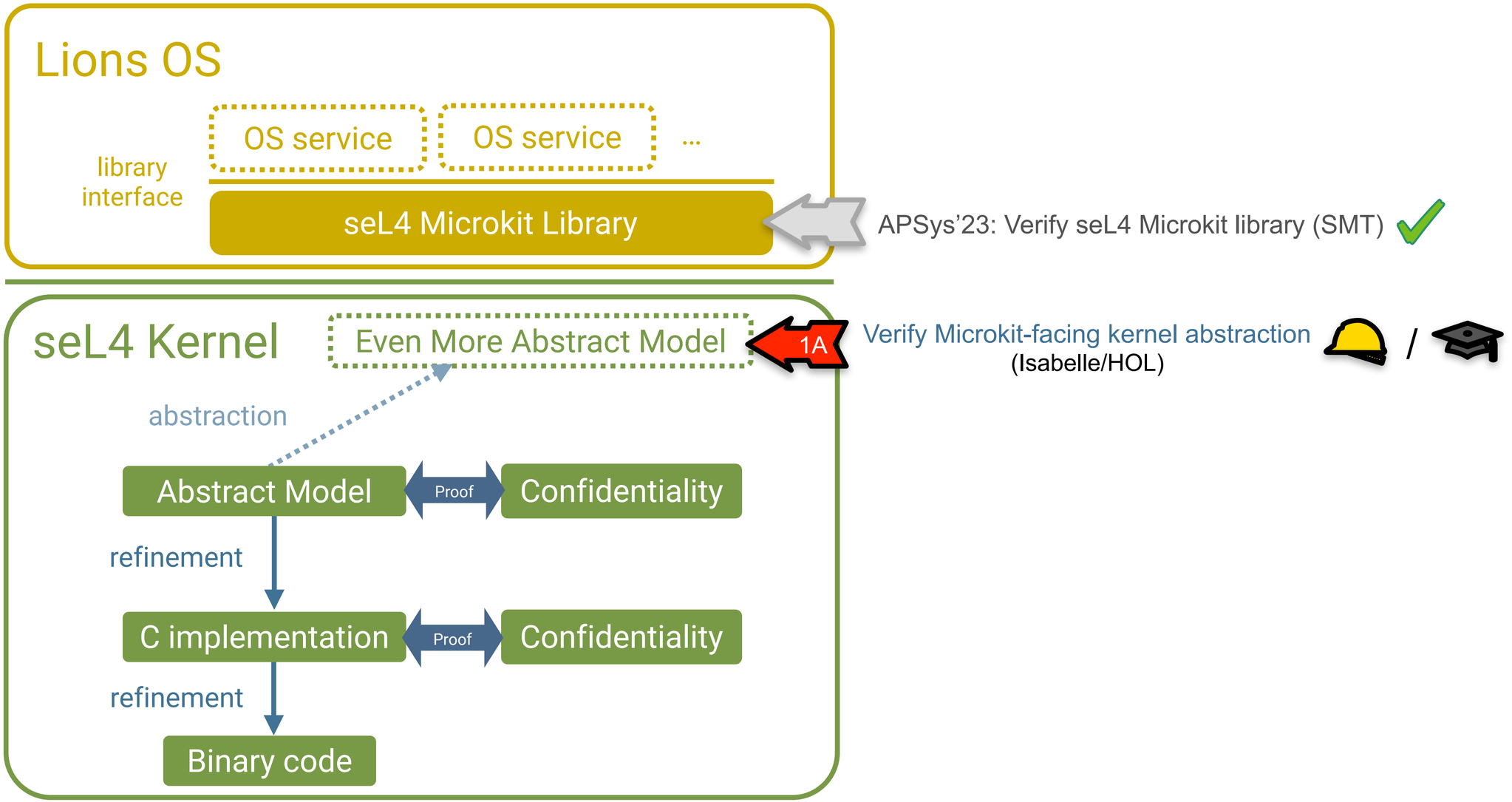


Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness of OS services

Security Enforcement

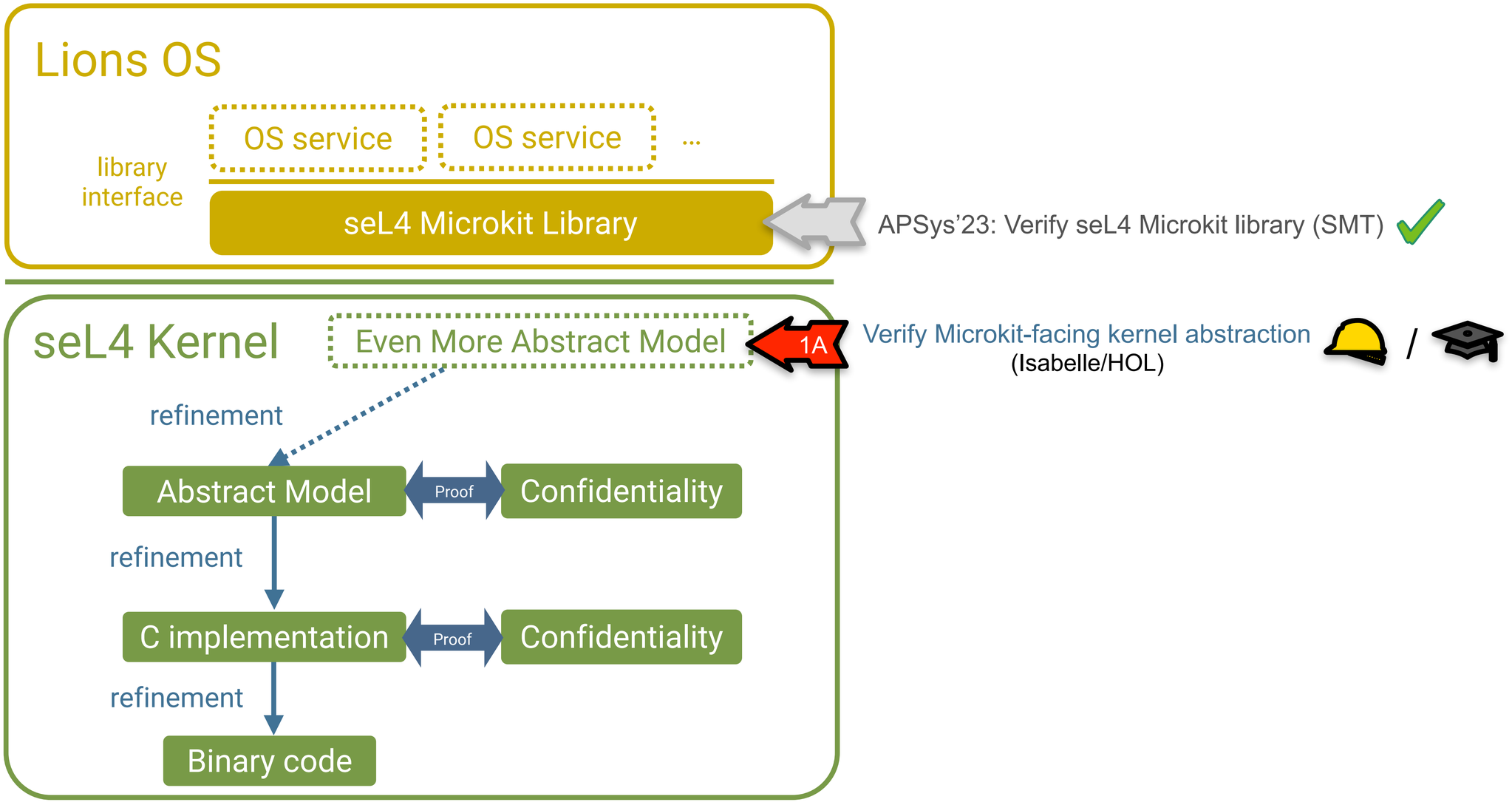


Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness of OS services

Security Enforcement

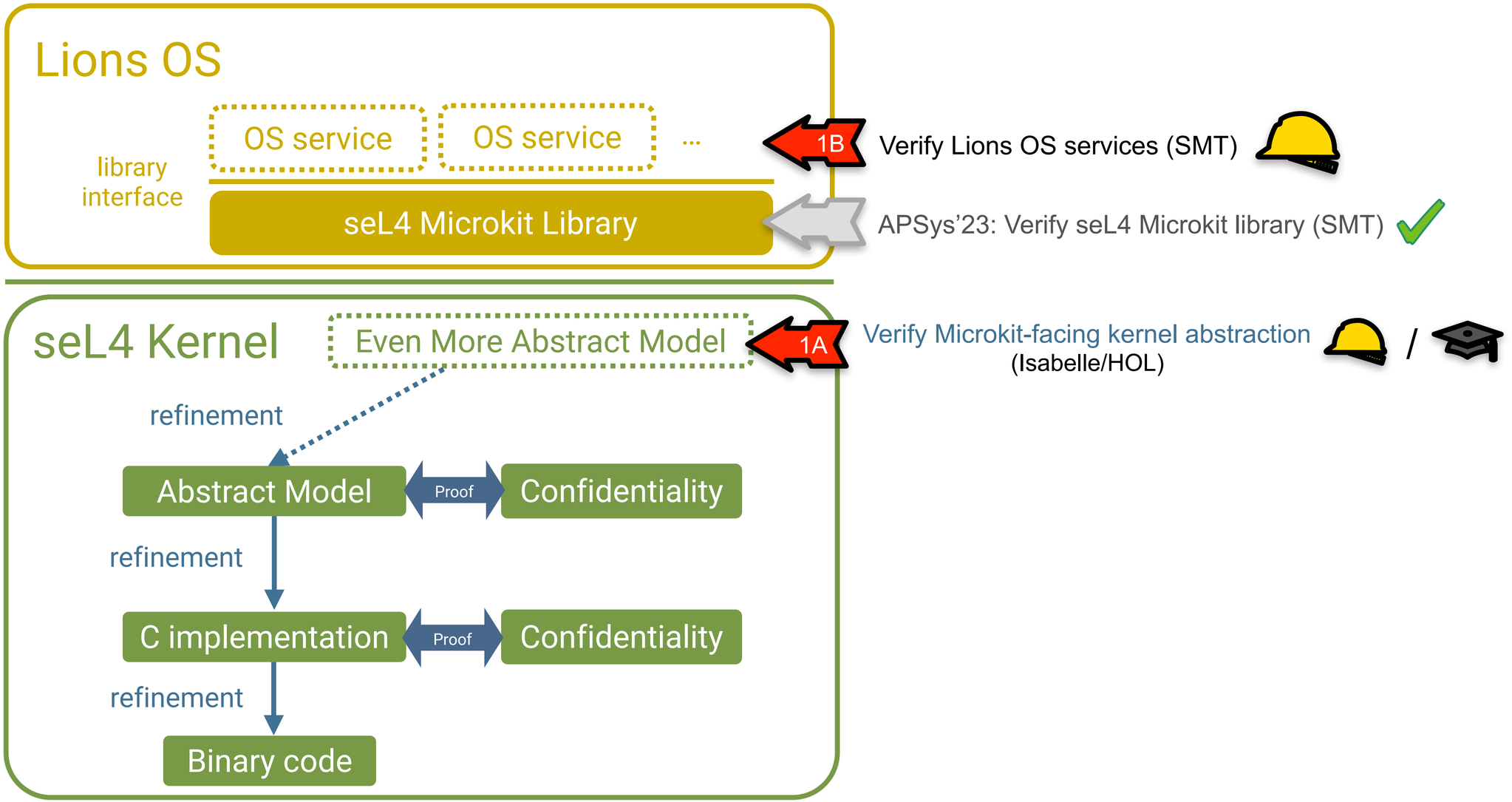


Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness of OS services

Security Enforcement

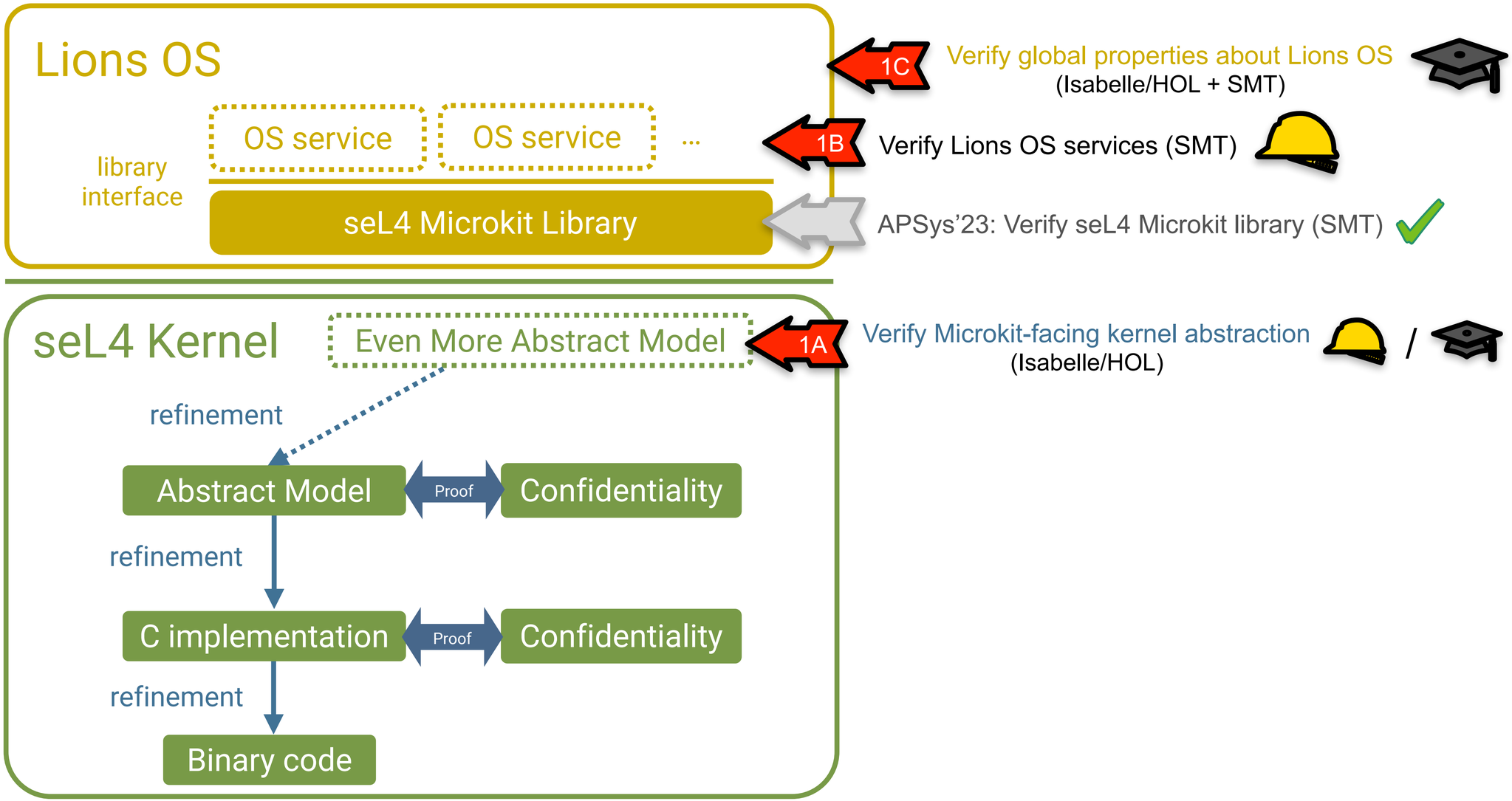


Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness of OS services

Security Enforcement



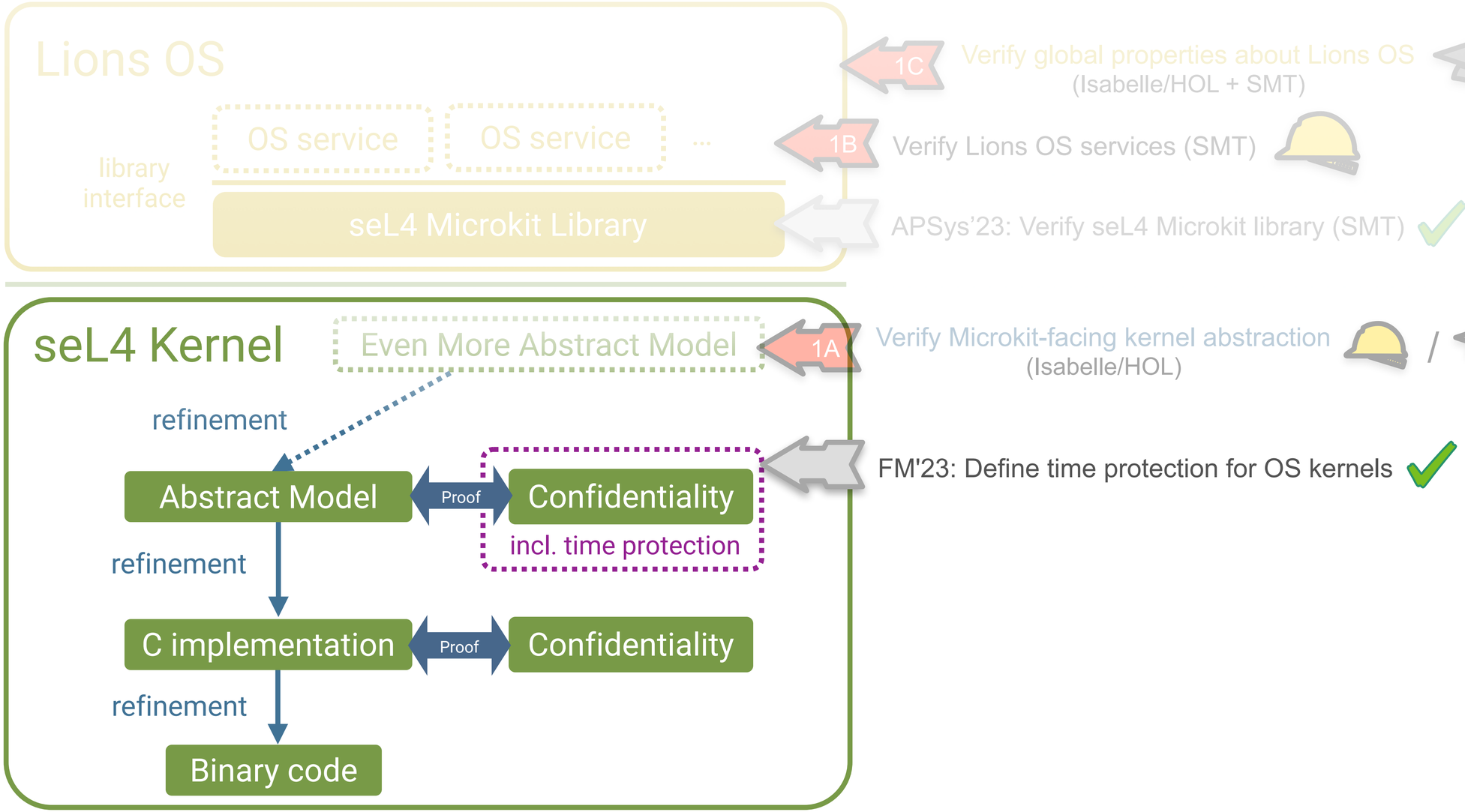
Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness of OS services

Frontier #2:

Security Enforcement of time protection



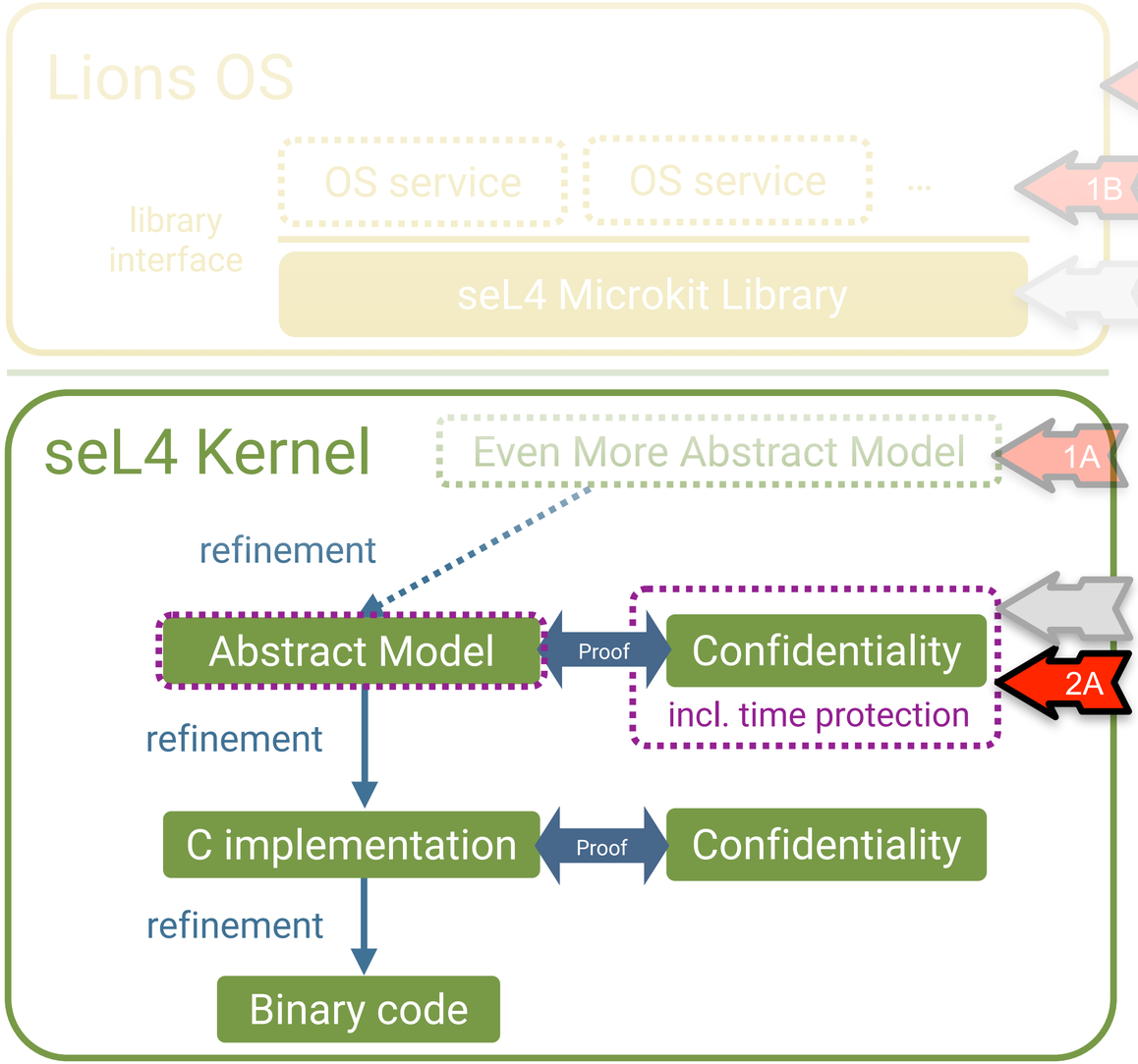
Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness of OS services

Frontier #2:

Security Enforcement of time protection



1C Verify global properties about Lions OS (Isabelle/HOL + SMT)

1B Verify Lions OS services (SMT)

APSys'23: Verify seL4 Microkit library (SMT)

1A Verify Microkit-facing kernel abstraction (Isabelle/HOL) /

FM'23: Define time protection for OS kernels +

2A Verify time protection for abstract model (Isabelle/HOL)

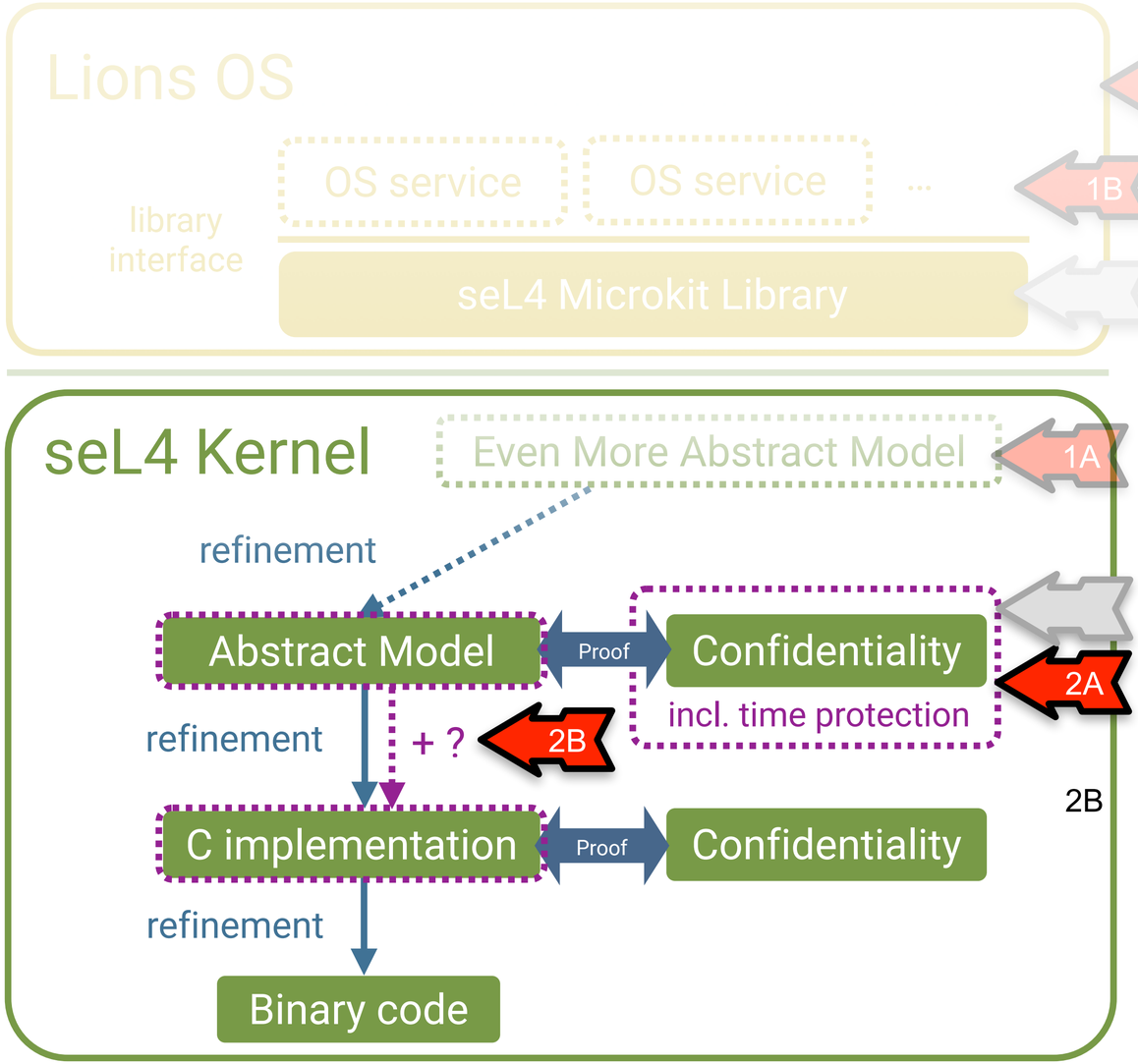
Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness of OS services

Frontier #2:

Security Enforcement of time protection



1C Verify global properties about Lions OS (Isabelle/HOL + SMT)

1B Verify Lions OS services (SMT)

APSys'23: Verify seL4 Microkit library (SMT) ✓

1A Verify Microkit-facing kernel abstraction (Isabelle/HOL) /

FM'23: Define time protection for OS kernels ✓
+
2A Verify time protection for abstract model (Isabelle/HOL)

2B Update refinement for time protection

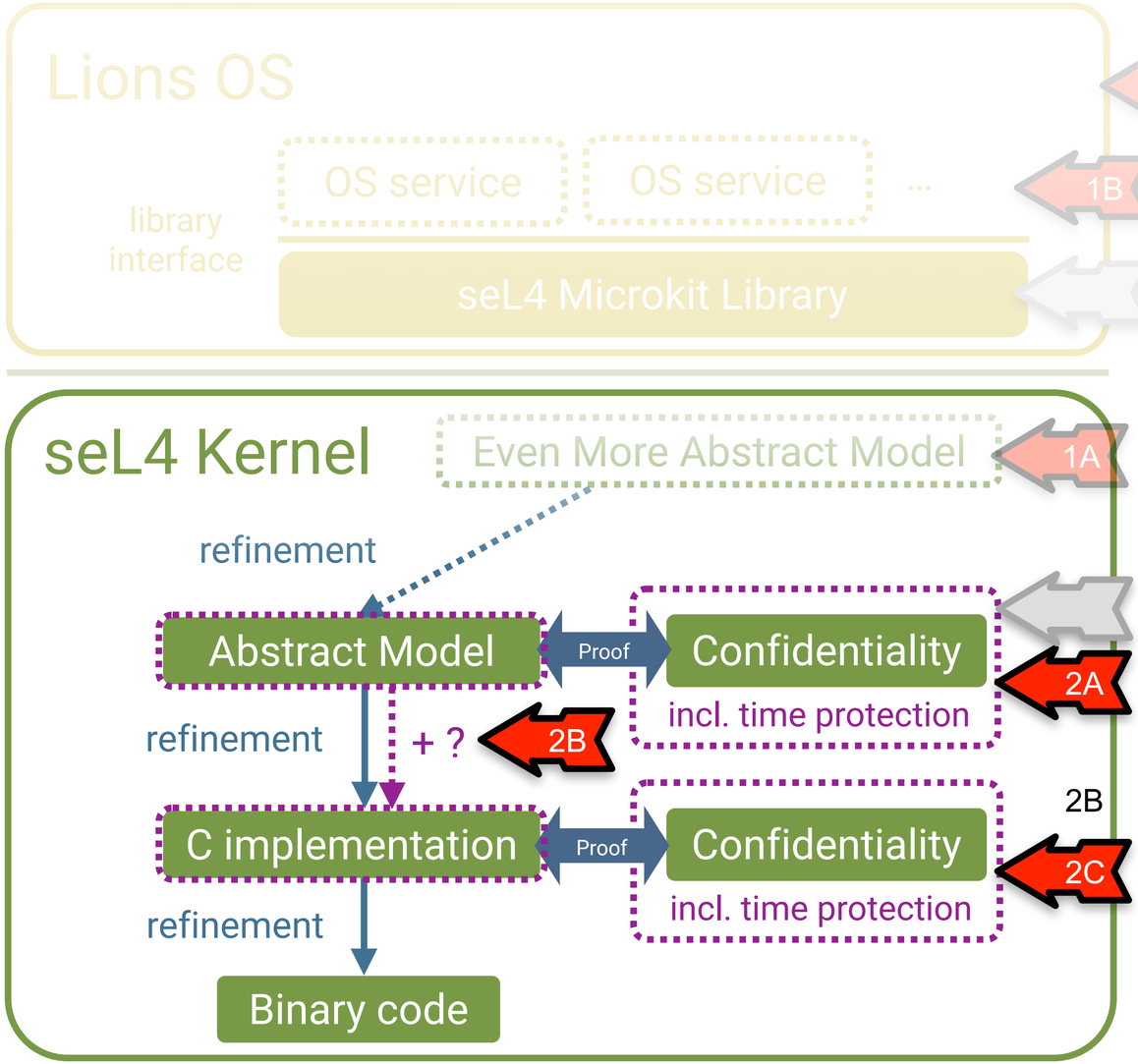
Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness of OS services

Frontier #2:

Security Enforcement of time protection



1C Verify global properties about Lions OS (Isabelle/HOL + SMT)

1B Verify Lions OS services (SMT)

APSys'23: Verify seL4 Microkit library (SMT)

1A Verify Microkit-facing kernel abstraction (Isabelle/HOL)

FM'23: Define time protection for OS kernels
+
2A Verify time protection for abstract model (Isabelle/HOL)

2B Update refinement for time protection
+
2C Verify time protection for C implementation (Isabelle/HOL)

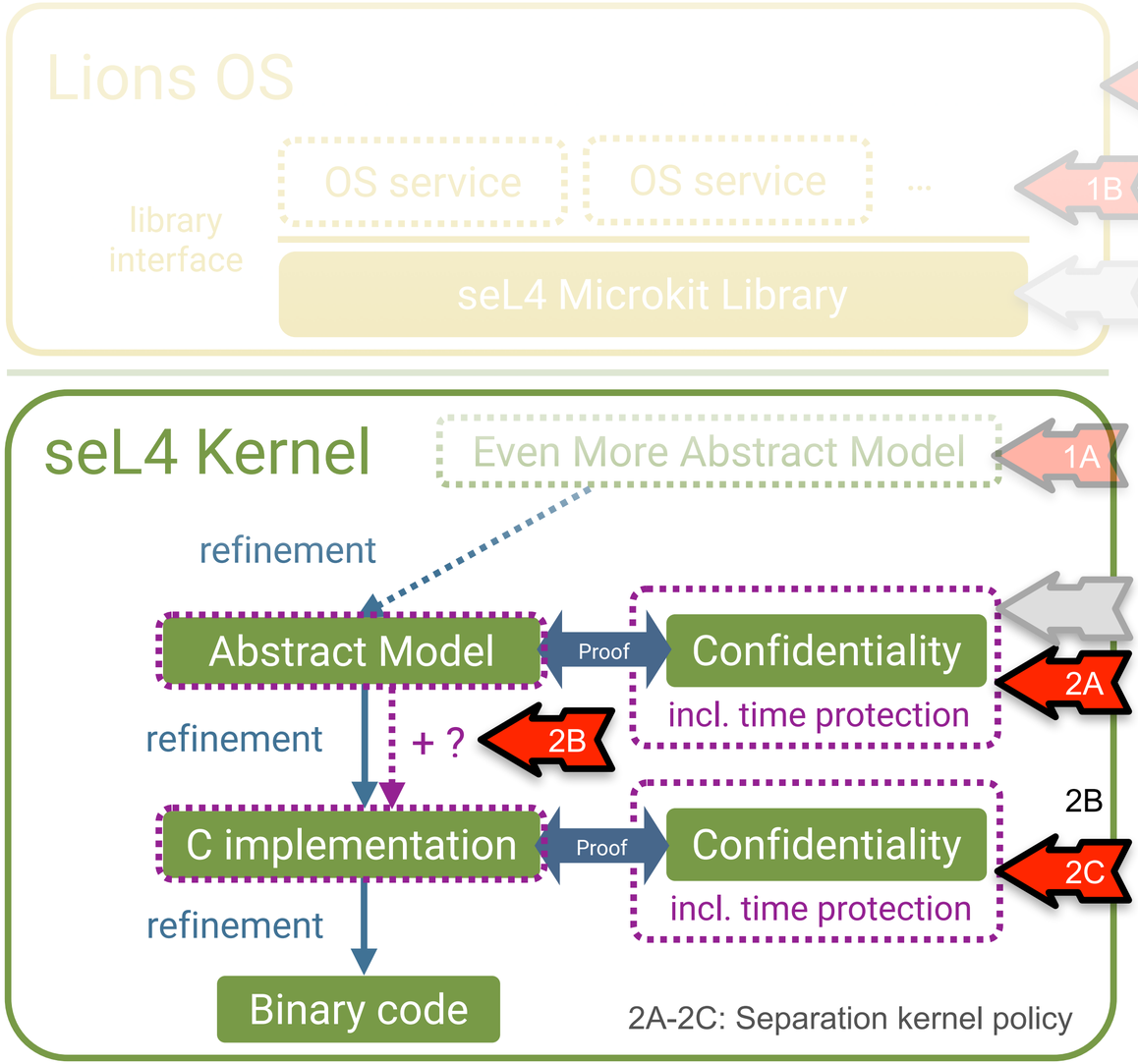
Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness of OS services

Frontier #2:

Security Enforcement of time protection



1C Verify global properties about Lions OS (Isabelle/HOL + SMT)

1B Verify Lions OS services (SMT)

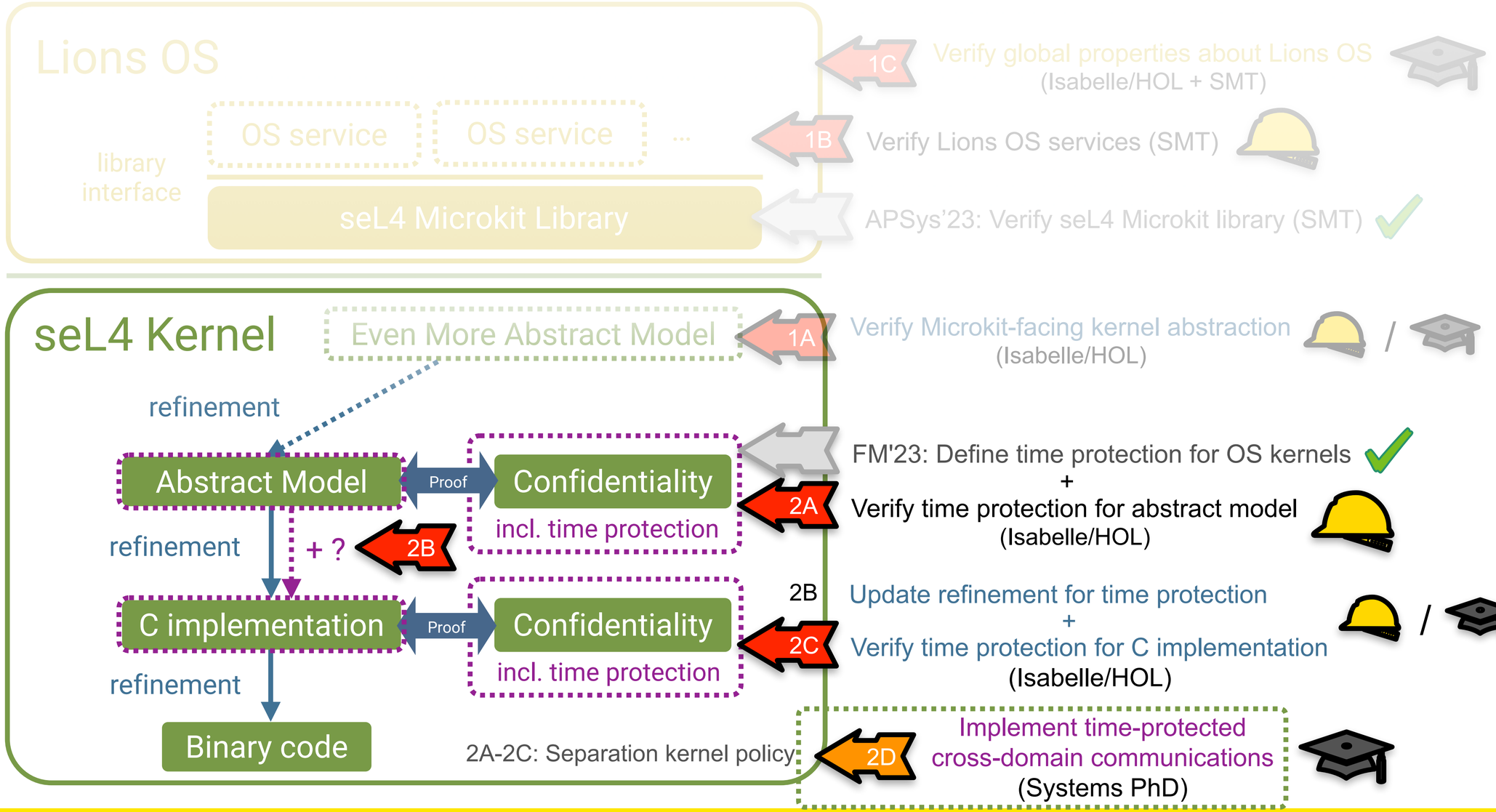
APSys'23: Verify seL4 Microkit library (SMT) ✓

1A Verify Microkit-facing kernel abstraction (Isabelle/HOL) /

FM'23: Define time protection for OS kernels ✓
+
2A Verify time protection for abstract model (Isabelle/HOL)

2B Update refinement for time protection
+
2C Verify time protection for C implementation (Isabelle/HOL) /

Two new frontiers for seL4 refinement



Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness

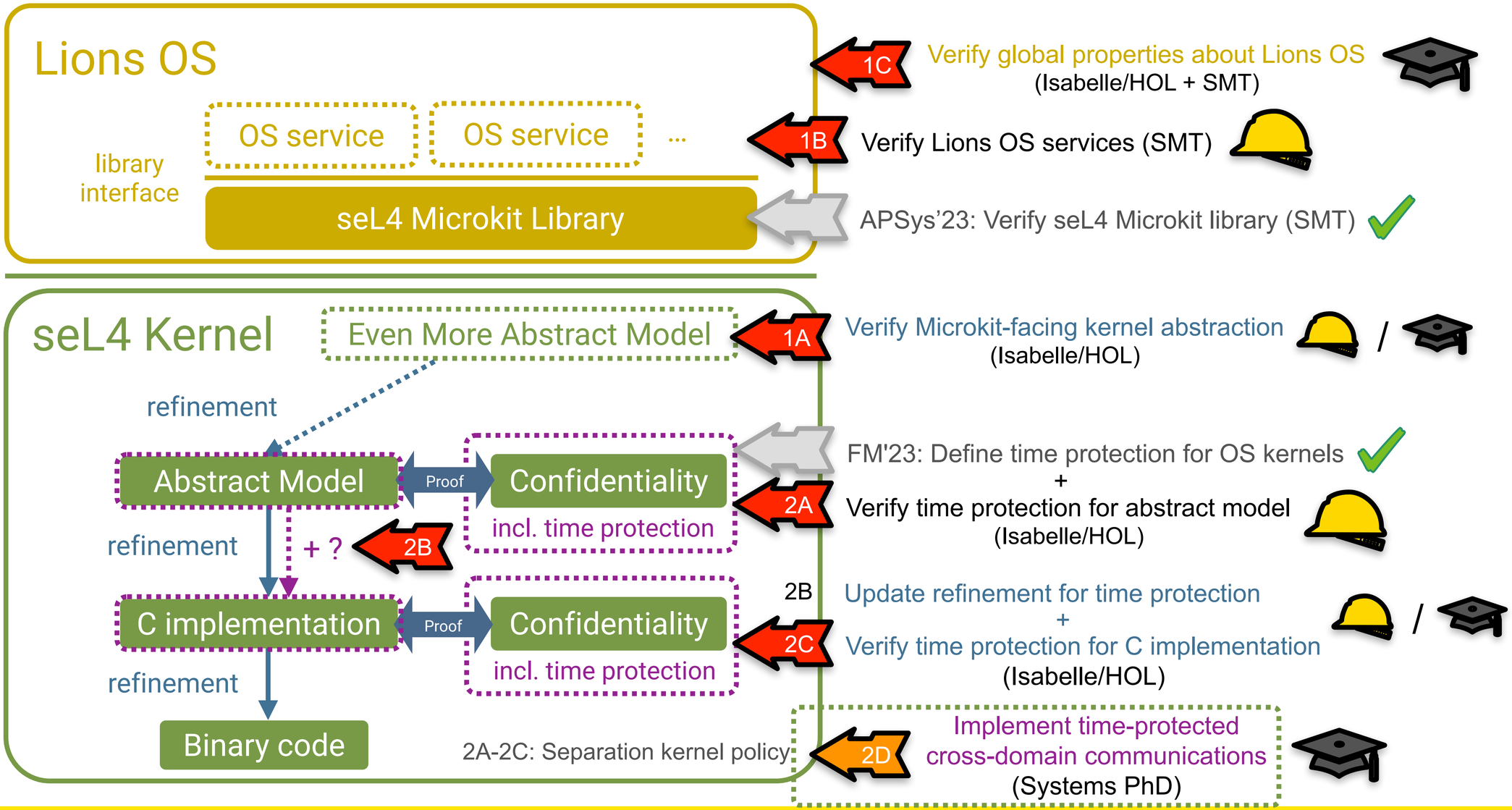
of OS services

syscall interface

Frontier #2:

Security Enforcement

of time protection



Two new frontiers for seL4 refinement

Frontier #1:

Functional Correctness

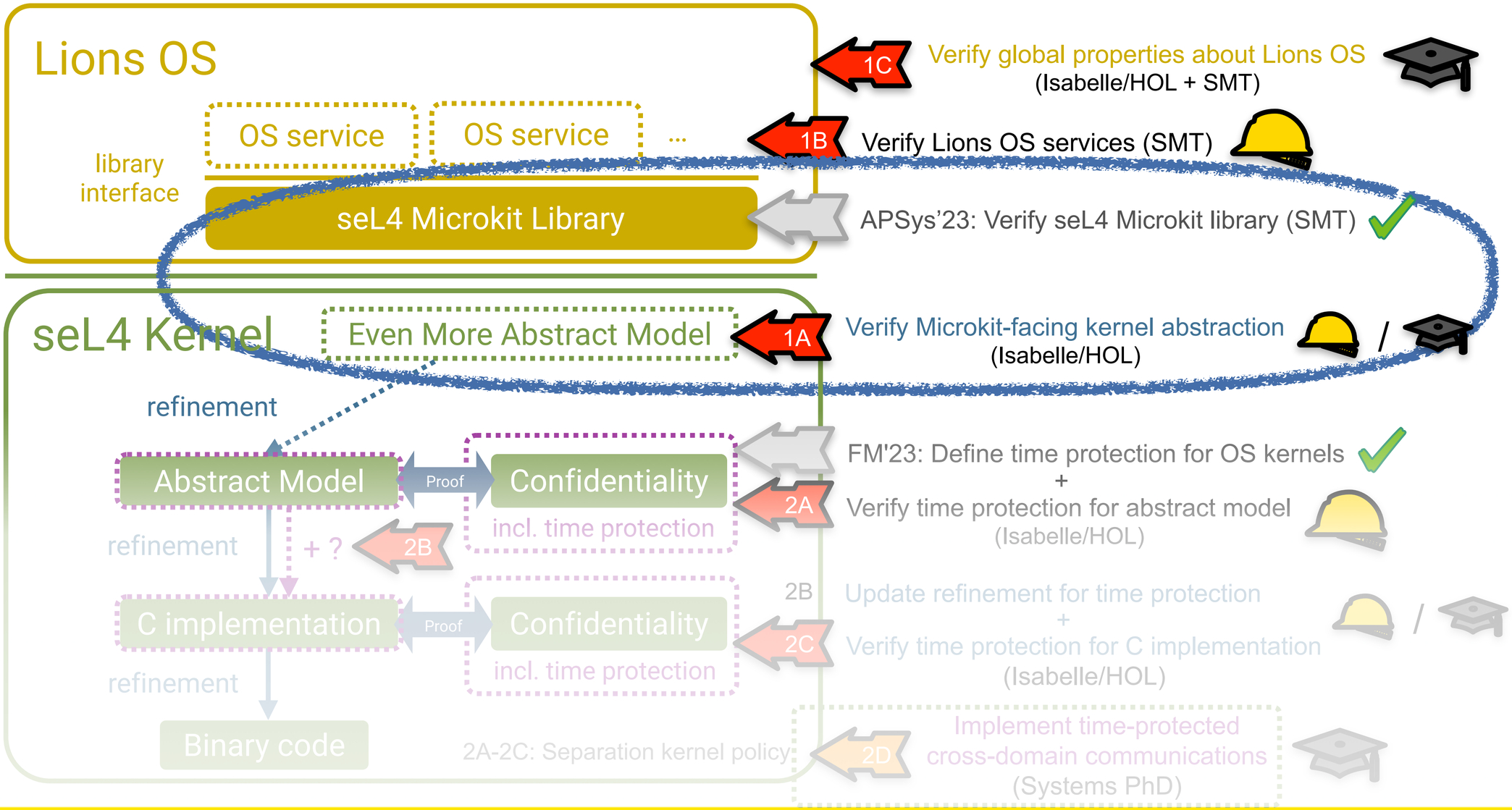
of OS services

syscall interface

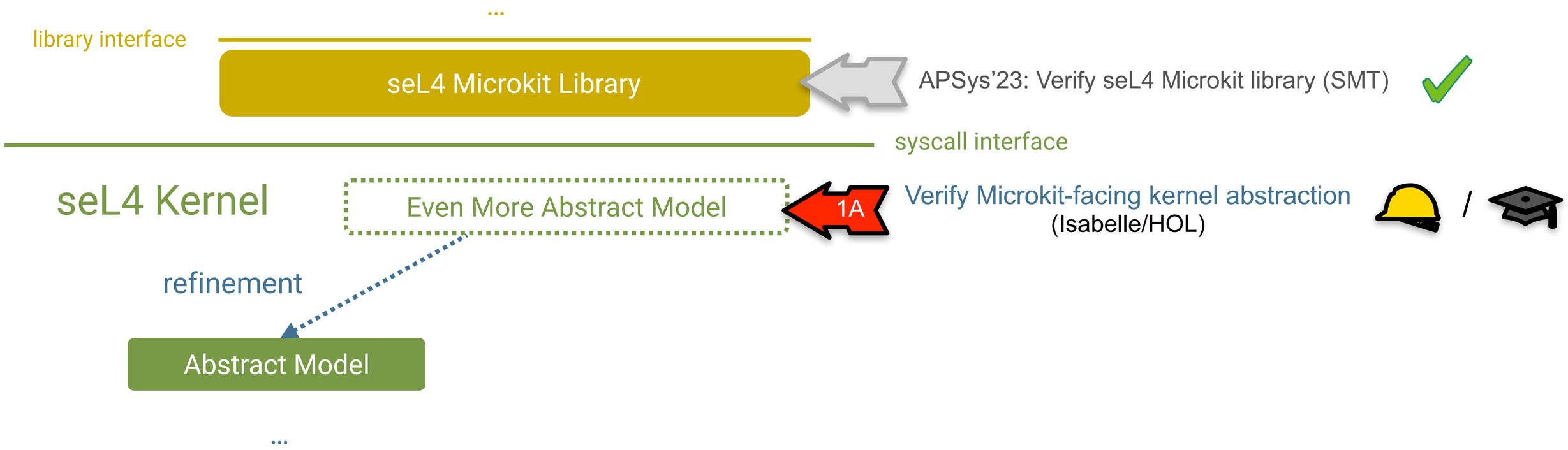
Frontier #2:

Security Enforcement

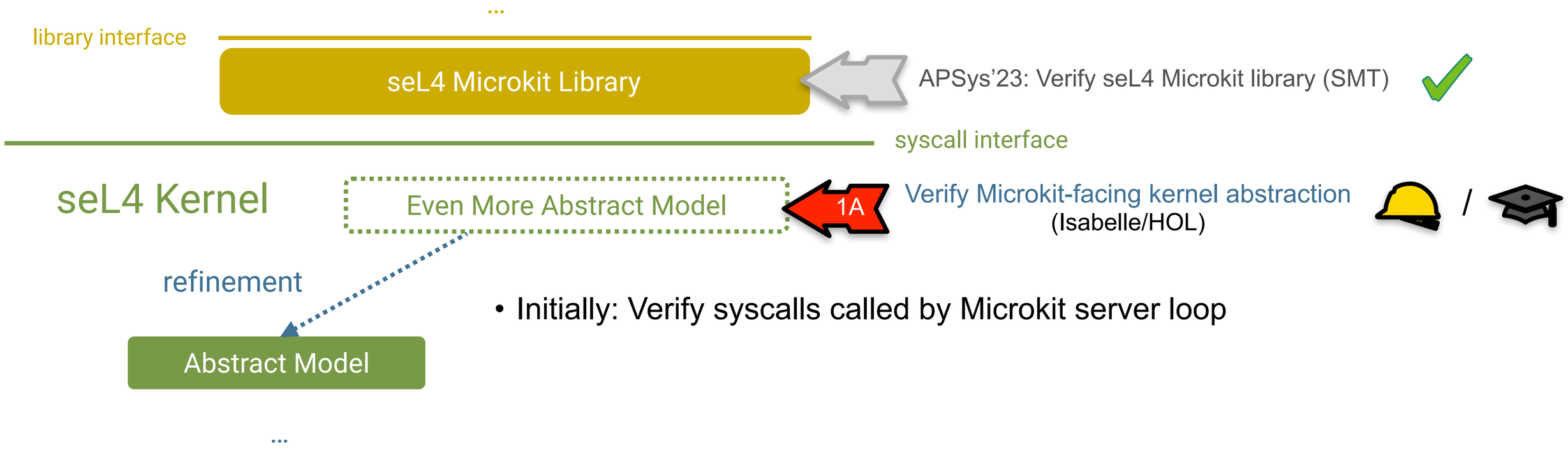
of time protection



Proving an OS kernel implements its syscall interface

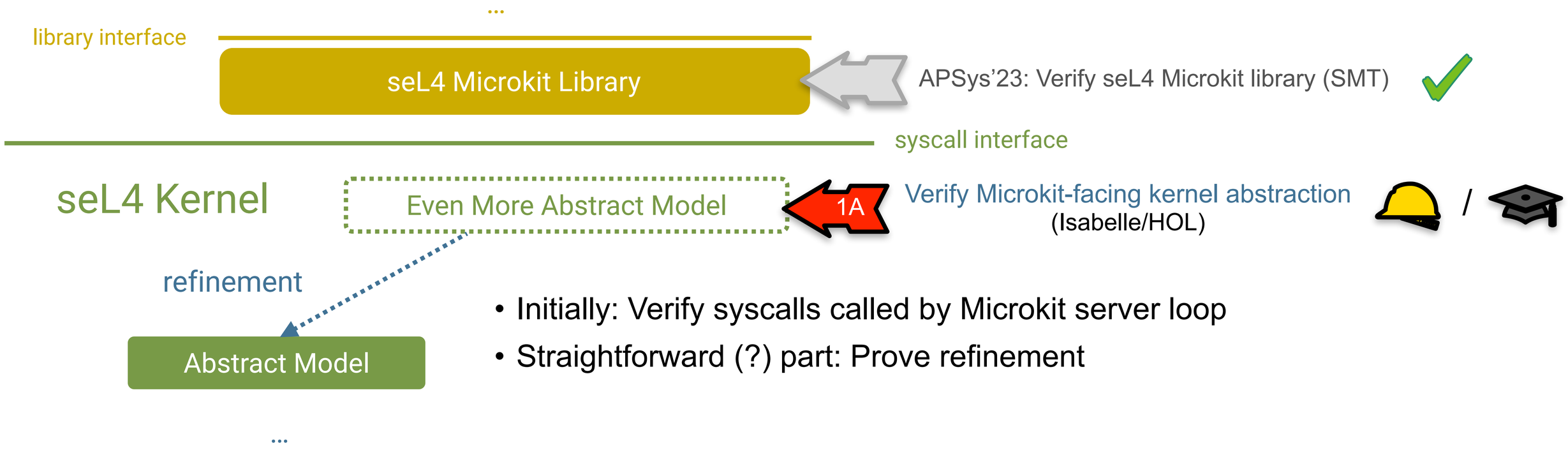


Proving an OS kernel implements its syscall interface



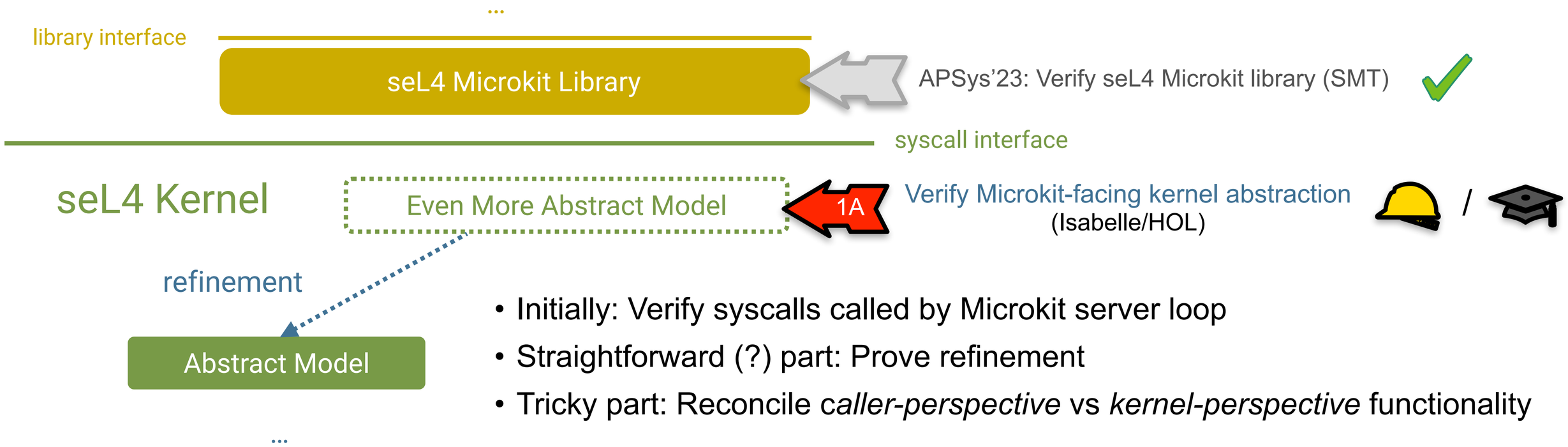
- Initially: Verify syscalls called by Microkit server loop

Proving an OS kernel implements its syscall interface



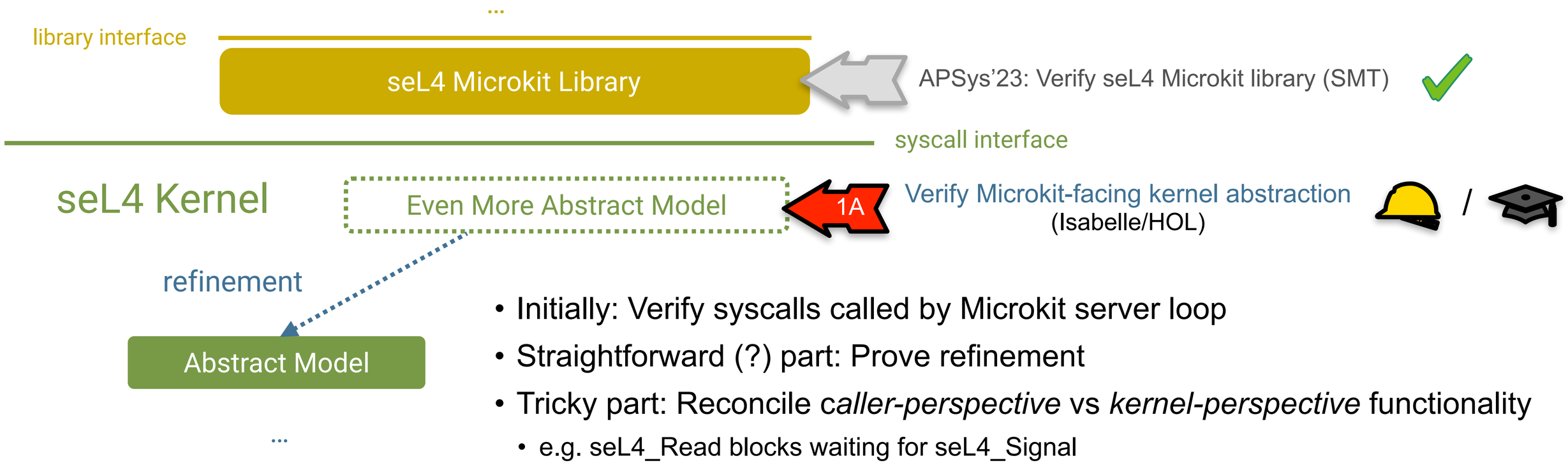
- Initially: Verify syscalls called by Microkit server loop
- Straightforward (?) part: Prove refinement

Proving an OS kernel implements its syscall interface



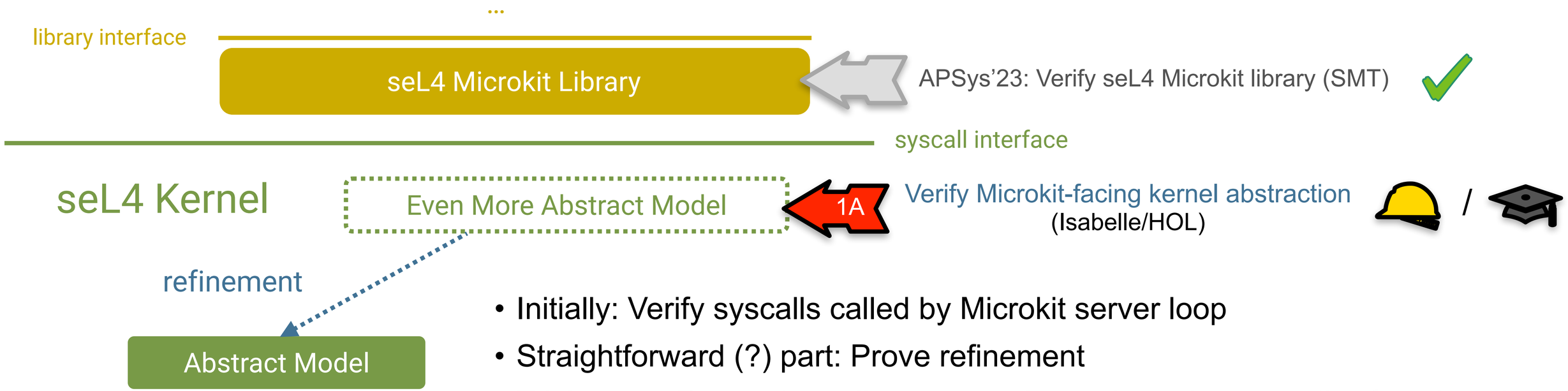
- Initially: Verify syscalls called by Microkit server loop
- Straightforward (?) part: Prove refinement
- Tricky part: Reconcile *caller-perspective* vs *kernel-perspective* functionality

Proving an OS kernel implements its syscall interface



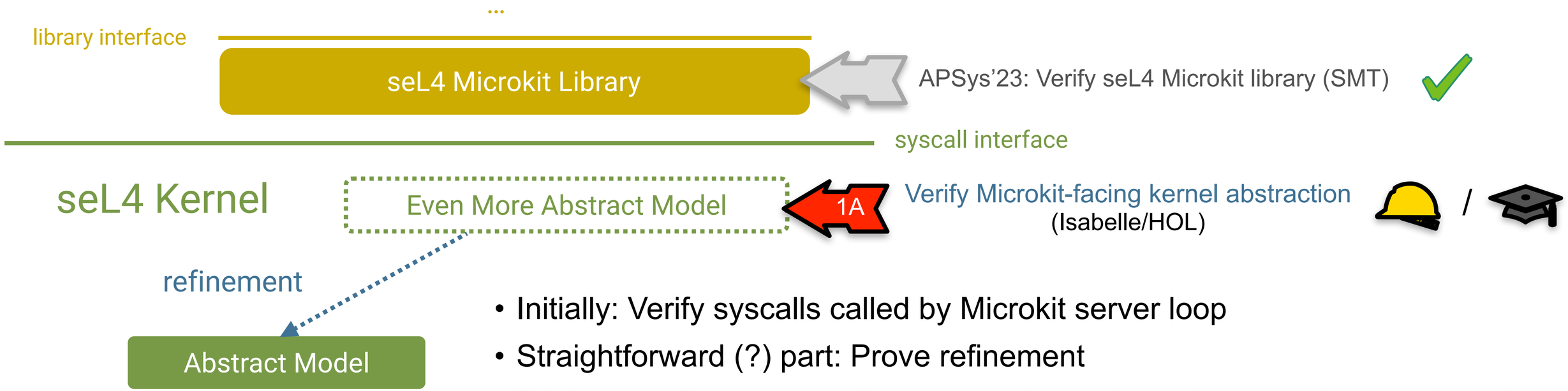
- Initially: Verify syscalls called by Microkit server loop
- Straightforward (?) part: Prove refinement
- Tricky part: Reconcile *caller-perspective* vs *kernel-perspective* functionality
  - e.g. seL4_Read blocks waiting for seL4_Signal

Proving an OS kernel implements its syscall interface



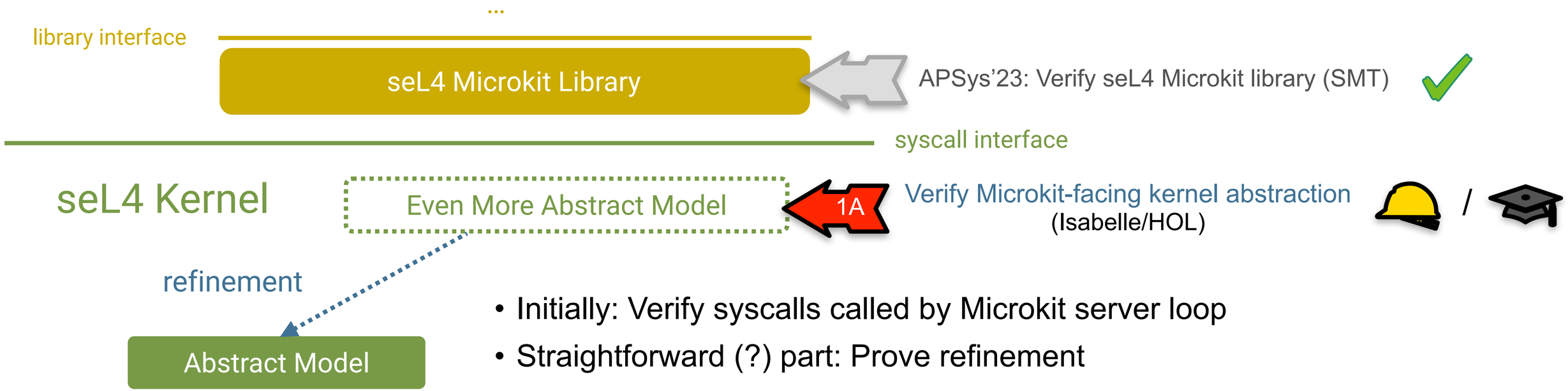
- Initially: Verify syscalls called by Microkit server loop
- Straightforward (?) part: Prove refinement
- Tricky part: Reconcile *caller-perspective* vs *kernel-perspective* functionality
  - e.g. seL4_Read blocks waiting for seL4_Signal
  - Kernel: I return to *different* user than caller

Proving an OS kernel implements its syscall interface



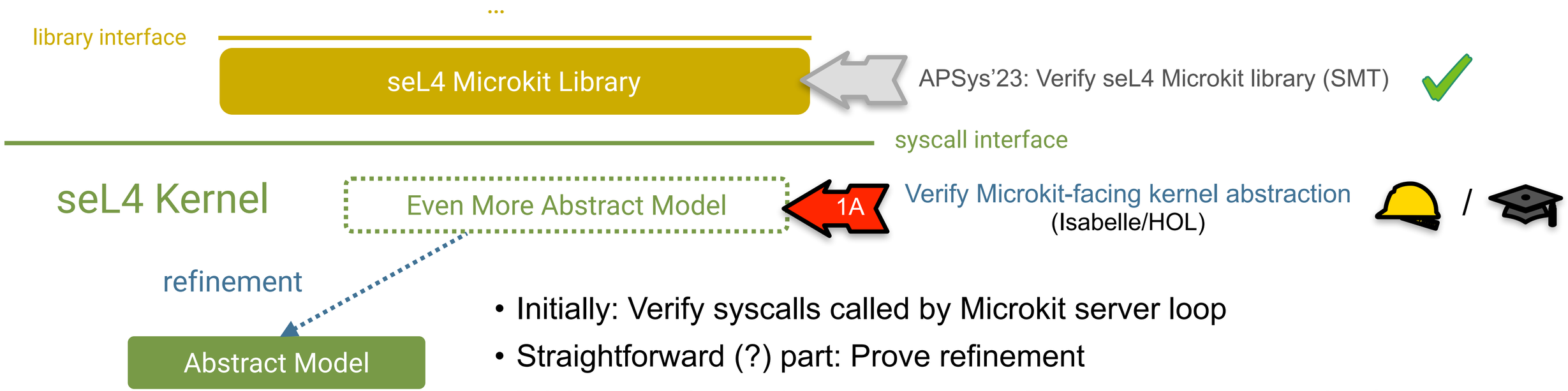
- Initially: Verify syscalls called by Microkit server loop
- Straightforward (?) part: Prove refinement
- Tricky part: Reconcile *caller-perspective* vs *kernel-perspective* functionality
  - e.g. seL4_Read blocks waiting for seL4_Signal
  - Kernel: I return to *different* user than caller
  - Caller: I wake up when signal arrives

Proving an OS kernel implements its syscall interface



- Initially: Verify syscalls called by Microkit server loop
- Straightforward (?) part: Prove refinement
- Tricky part: Reconcile *caller-perspective* vs *kernel-perspective* functionality
  - e.g. seL4_Read blocks waiting for seL4_Signal
  - Kernel: I return to *different* user than caller
  - Caller: I wake up when signal arrives
- I don't think any OS verification has handled this case before

Proving an OS kernel implements its syscall interface

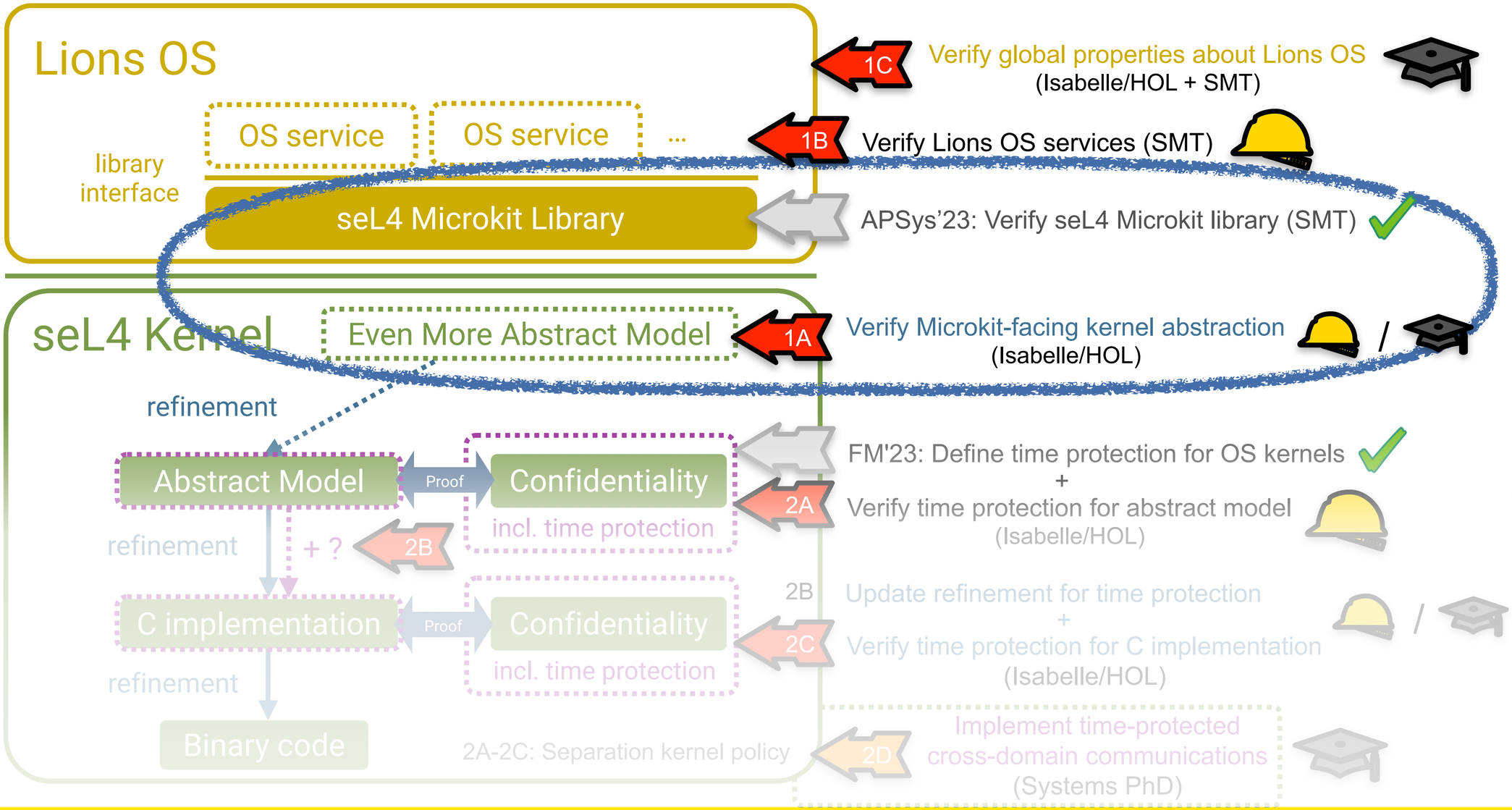


- Initially: Verify syscalls called by Microkit server loop
- Straightforward (?) part: Prove refinement
- Tricky part: Reconcile *caller-perspective* vs *kernel-perspective* functionality
  - e.g. seL4_Read blocks waiting for seL4_Signal
  - Kernel: I return to *different* user than caller
  - Caller: I wake up when signal arrives
- I don't think any OS verification has handled this case before
  - cf. CertiKOS ESOP'20 - blocks on IO, not another user

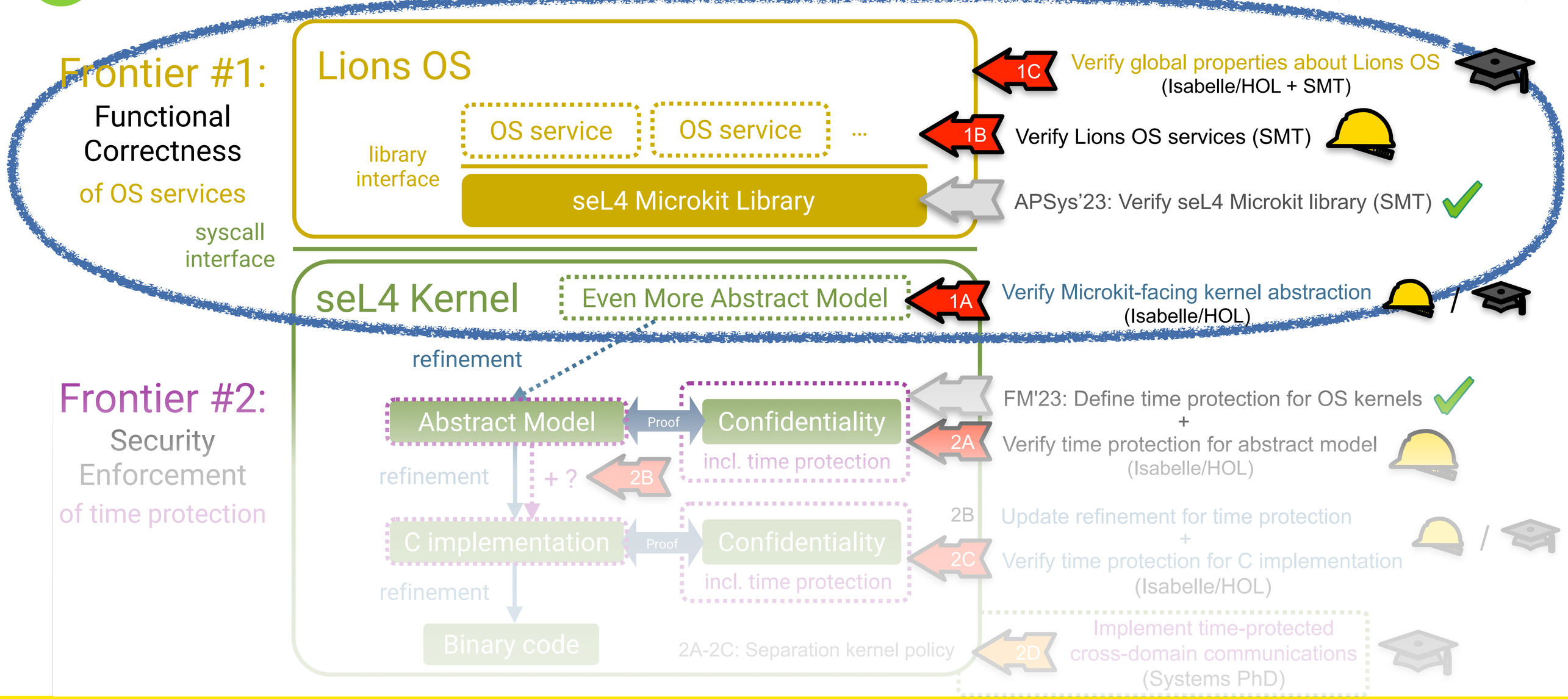
Two new frontiers for seL4 refinement

Frontier #1:
Functional Correctness
of OS services

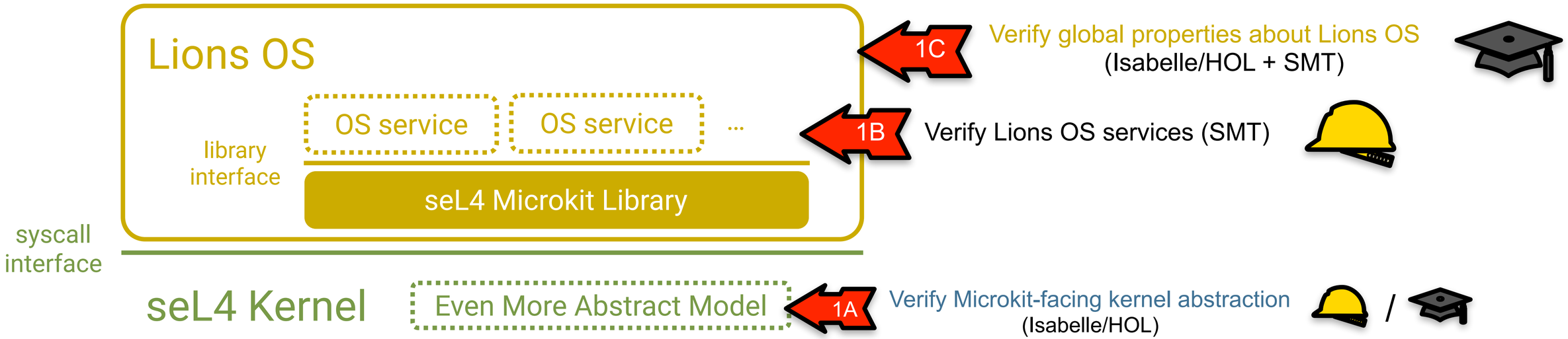
Frontier #2:
Security Enforcement
of time protection



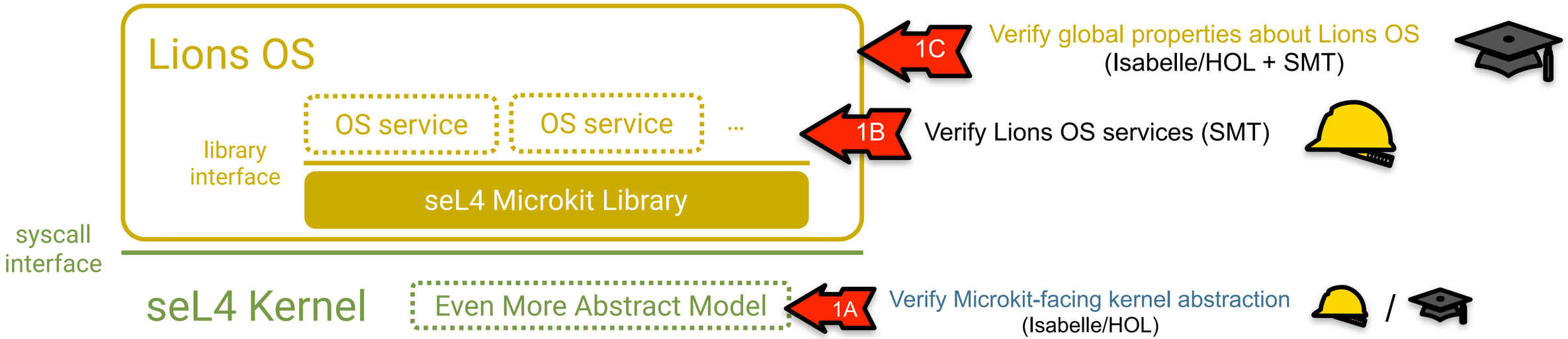
Two new frontiers for seL4 refinement



Proving OS services provide functional + security properties

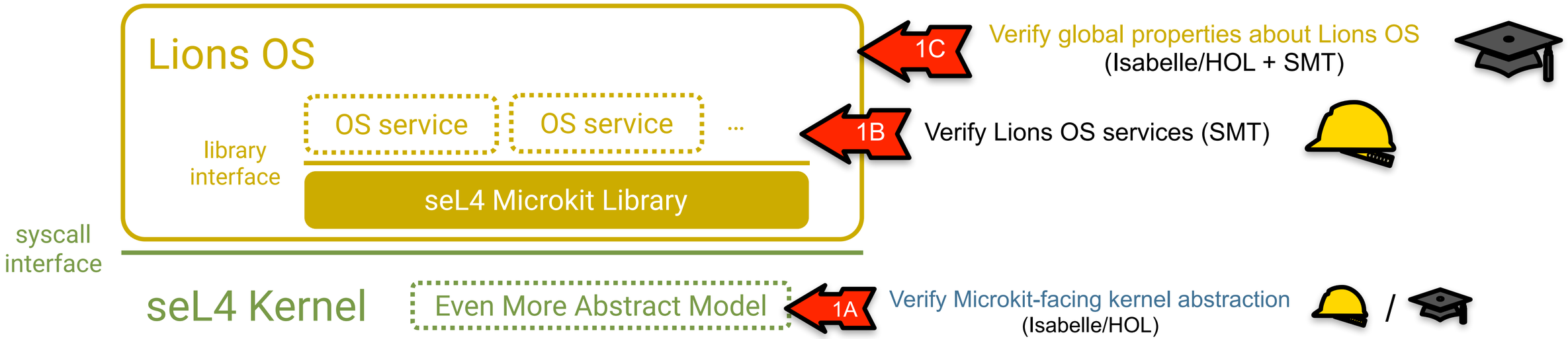


Proving OS services provide functional + security properties



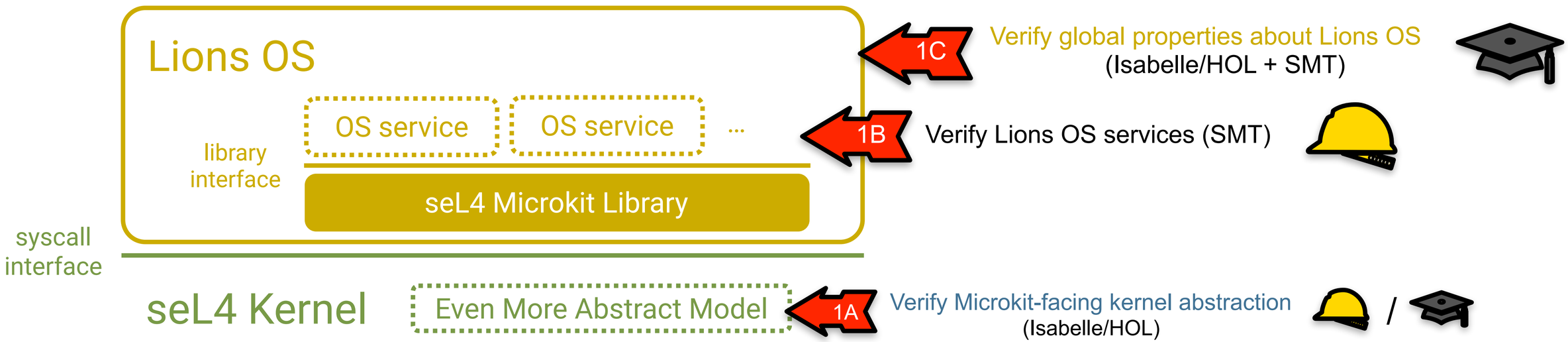
- Initially: Verify functional, safety properties (local + global)

Proving OS services provide functional + security properties



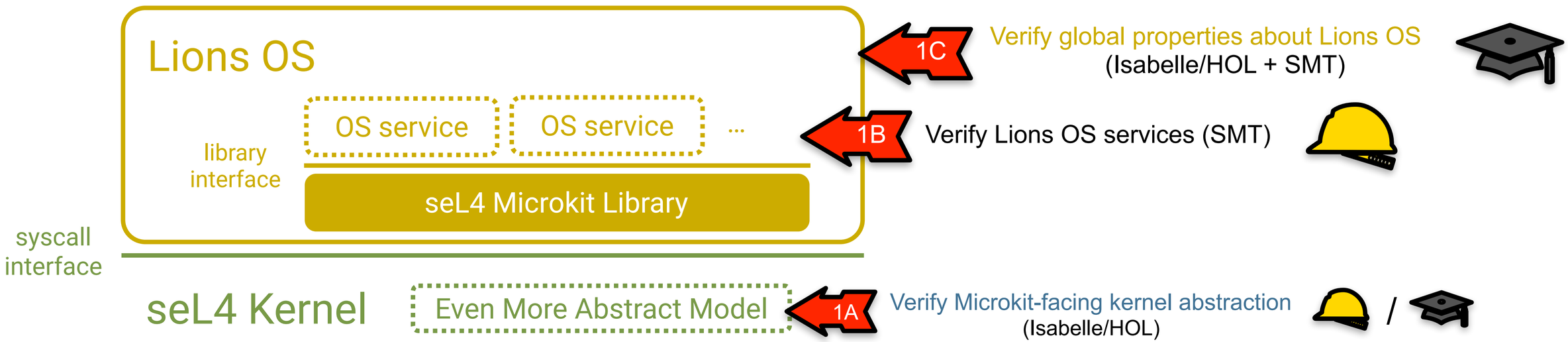
- Initially: Verify functional, safety properties (local + global)
- Further challenge: Define + verify *compositional* security properties (local => global)

Proving OS services provide functional + security properties



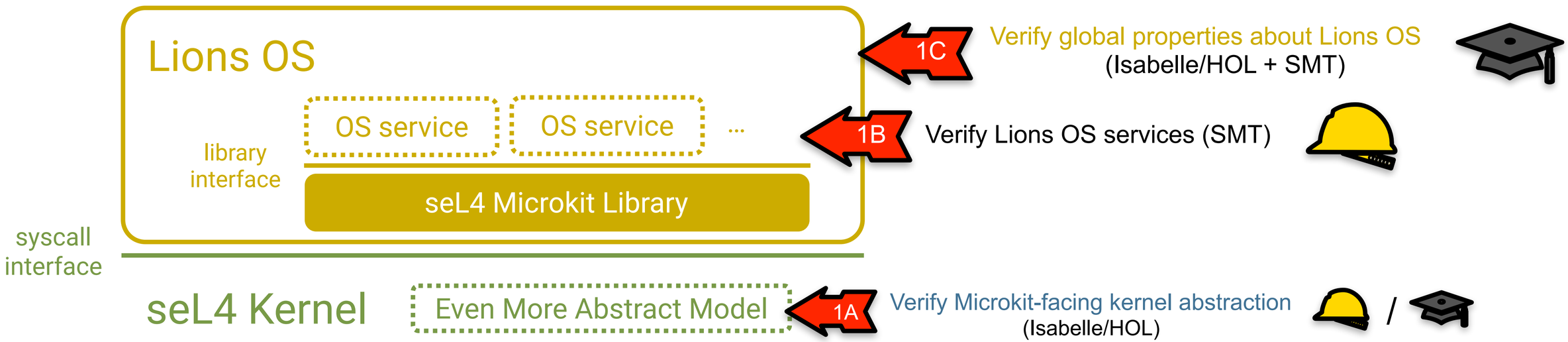
- Initially: Verify functional, safety properties (local + global)
- Further challenge: Define + verify *compositional* security properties (local => global)
- Guiding questions for all phases:
  - What properties are useful to Lions OS' users?

Proving OS services provide functional + security properties



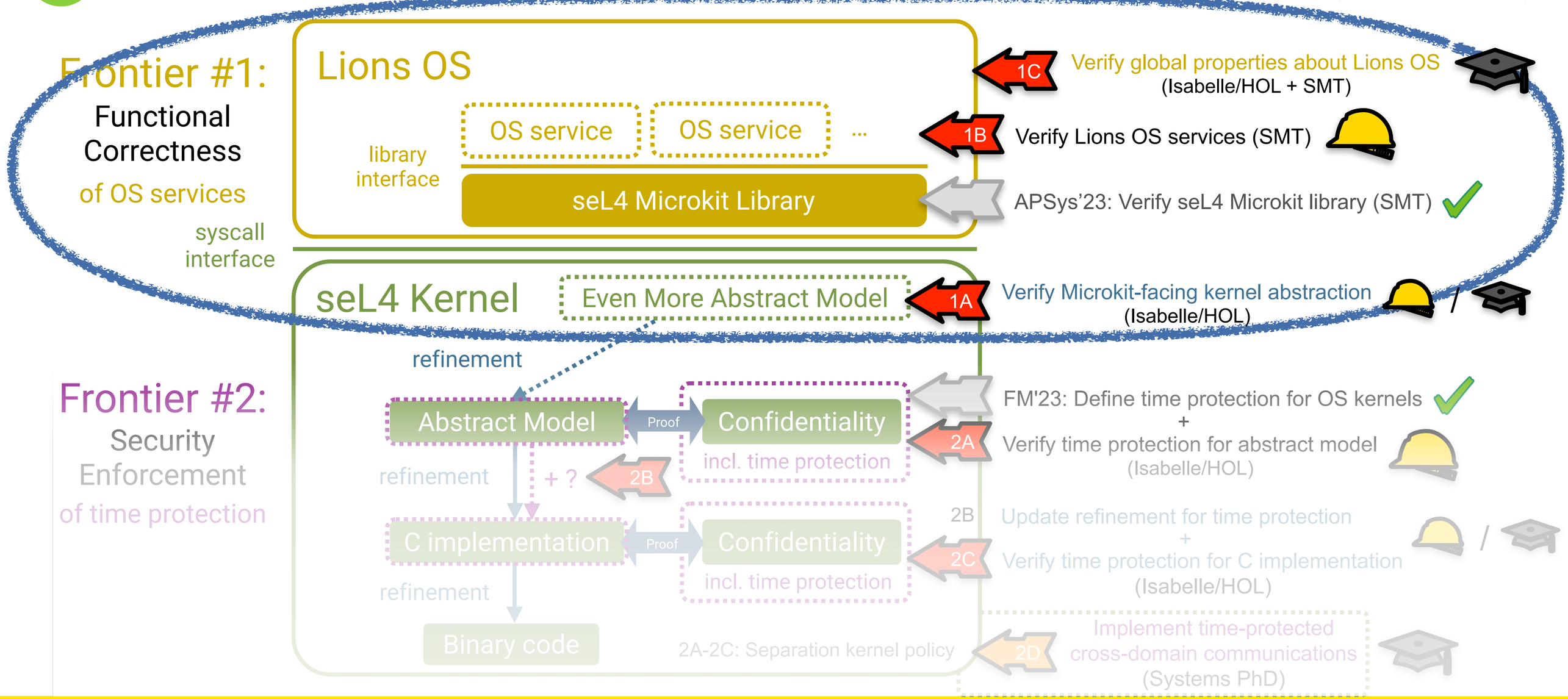
- Initially: Verify functional, safety properties (local + global)
- Further challenge: Define + verify *compositional* security properties (local => global)
- Guiding questions for all phases:
  - What properties are useful to Lions OS' users?
  - What stays amenable to SMT solving?

Proving OS services provide functional + security properties

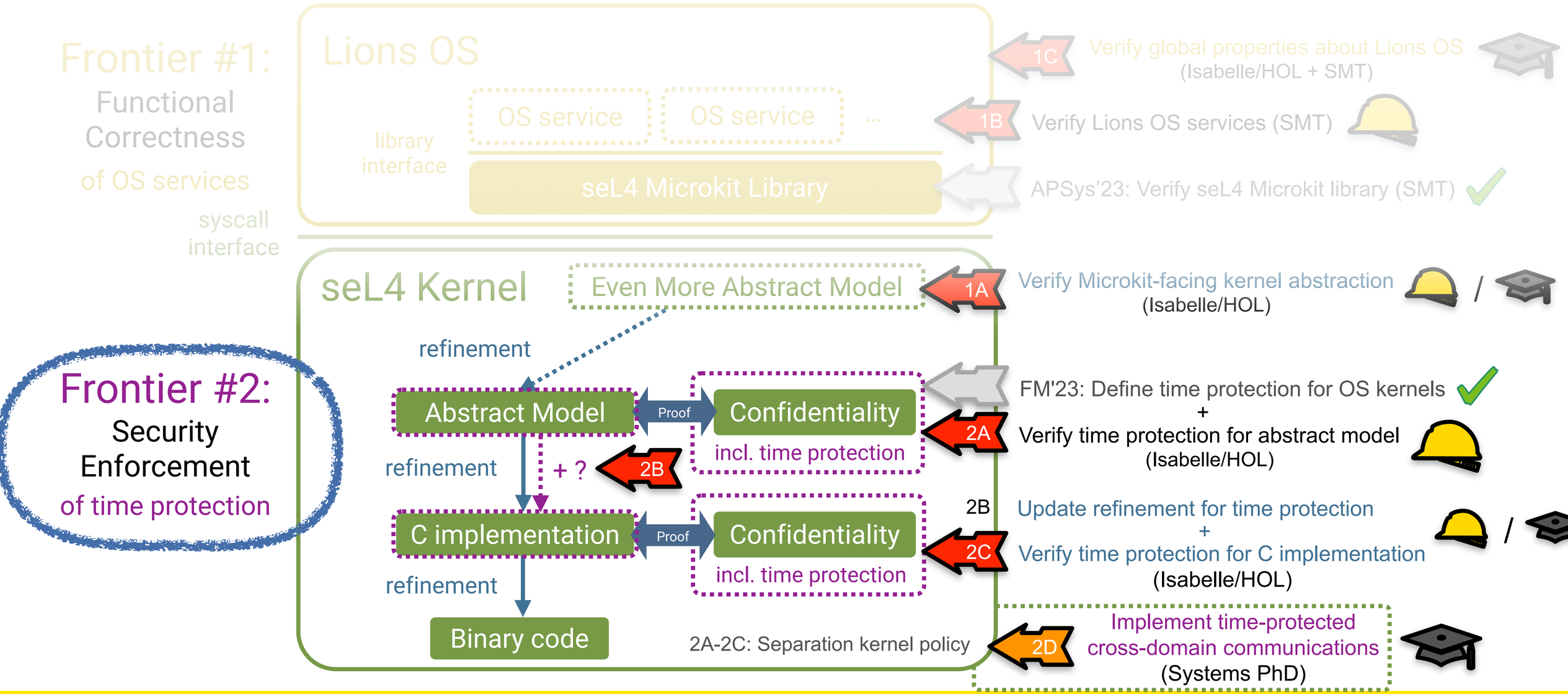


- Initially: Verify functional, safety properties (local + global)
- Further challenge: Define + verify *compositional* security properties (local => global)
- Guiding questions for all phases:
  - What properties are useful to Lions OS' users?
  - What stays amenable to SMT solving?
- May demand further proofs of syscall properties (i.e. additions to 1A)

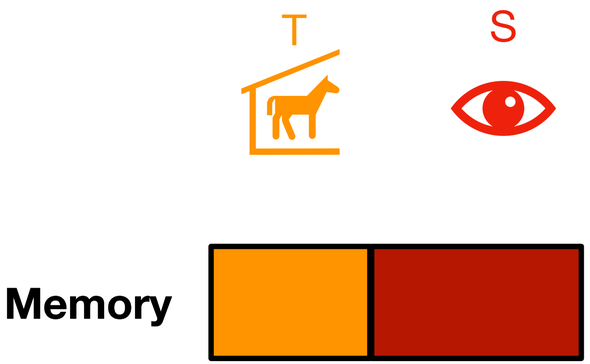
Two new frontiers for seL4 refinement



Two new frontiers for seL4 refinement

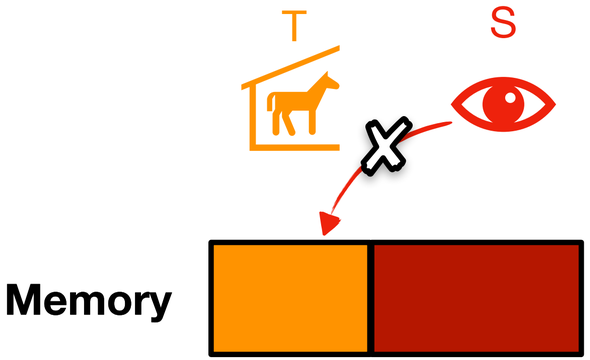


What is time protection?



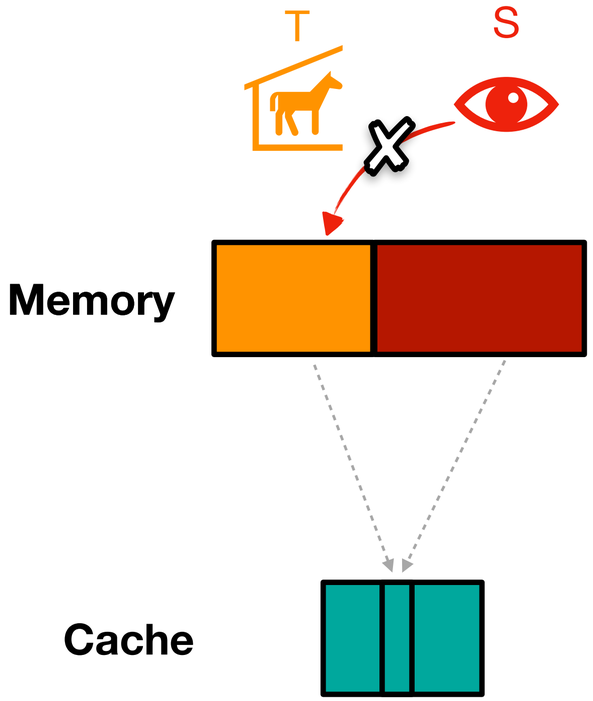
What is time protection?

- OSes typically implement *memory protection.*



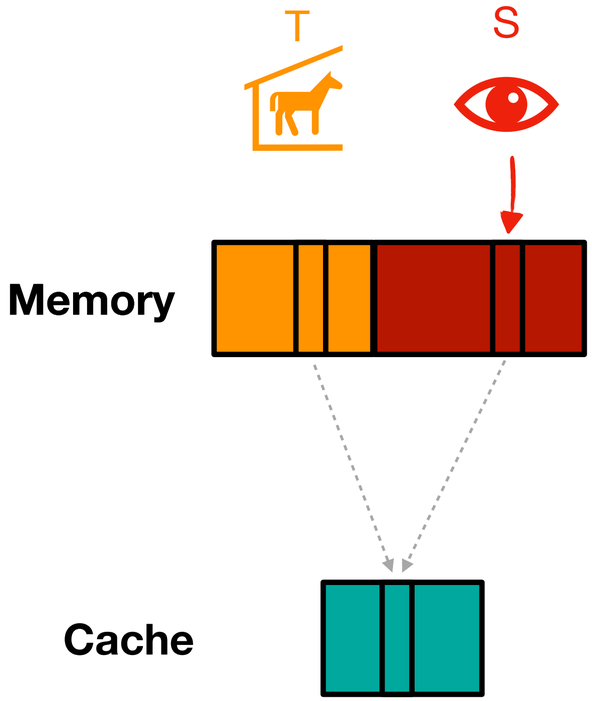
What is time protection?

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



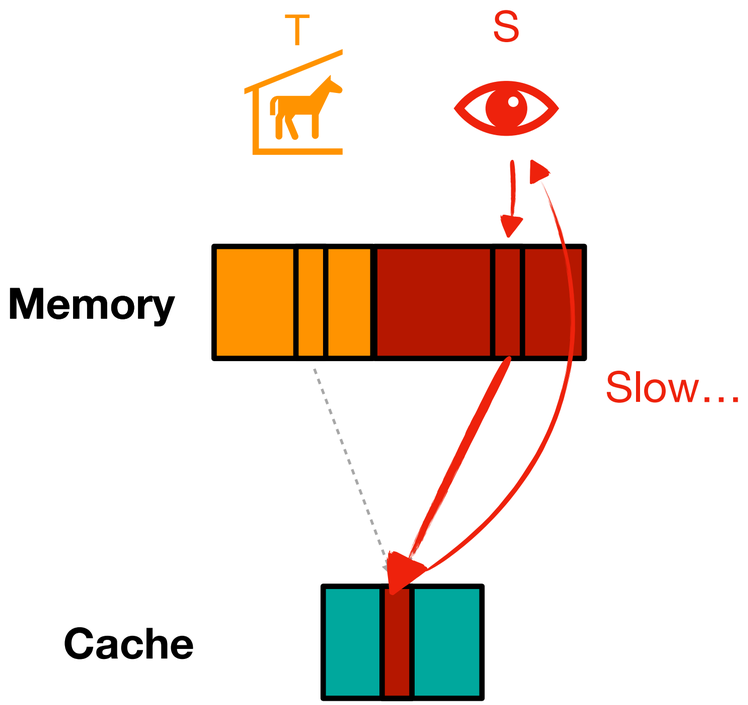
What is time protection?

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



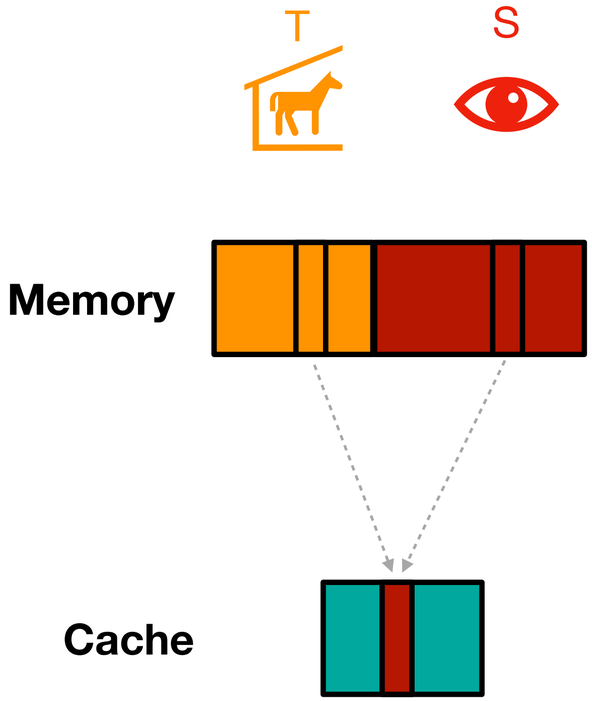
What is time protection?

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



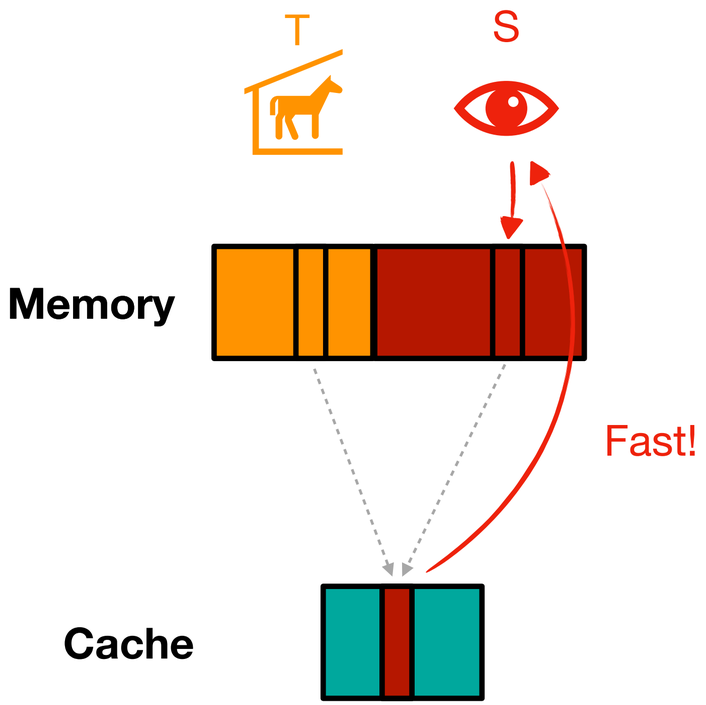
What is time protection?

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



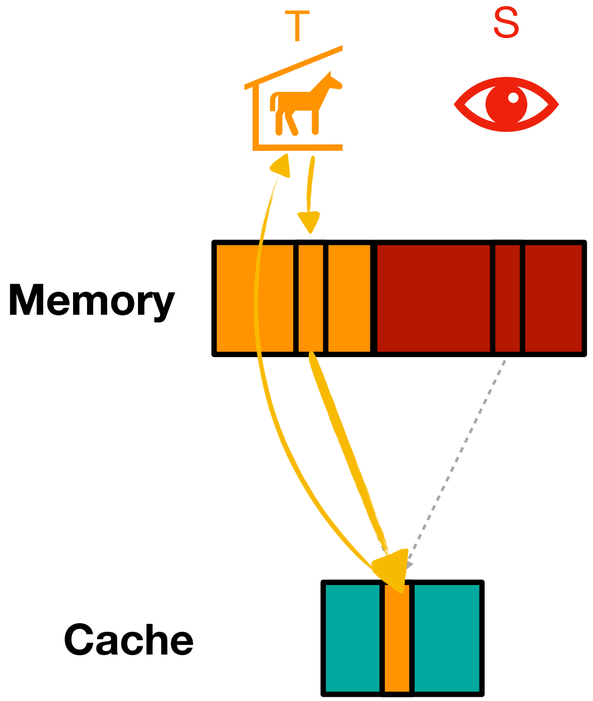
What is time protection?

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



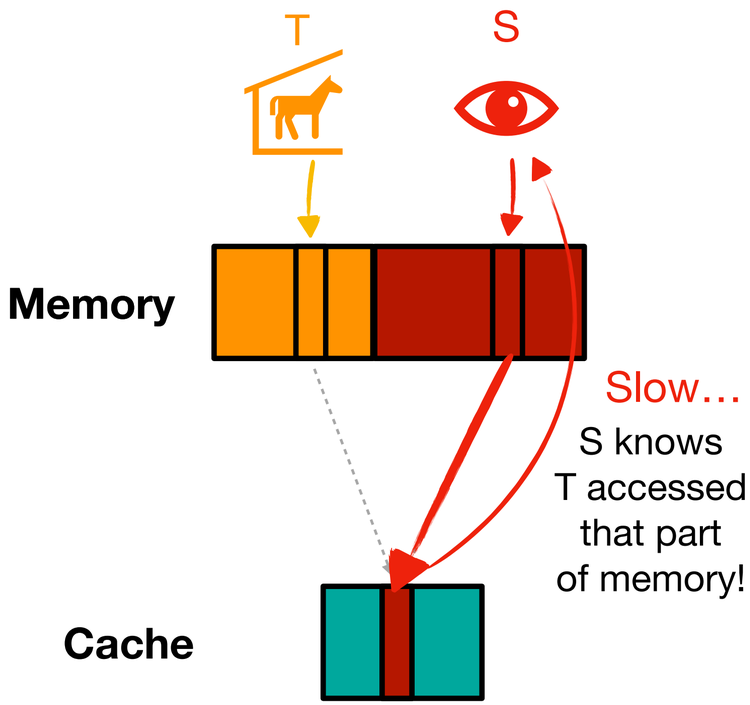
What is time protection?

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



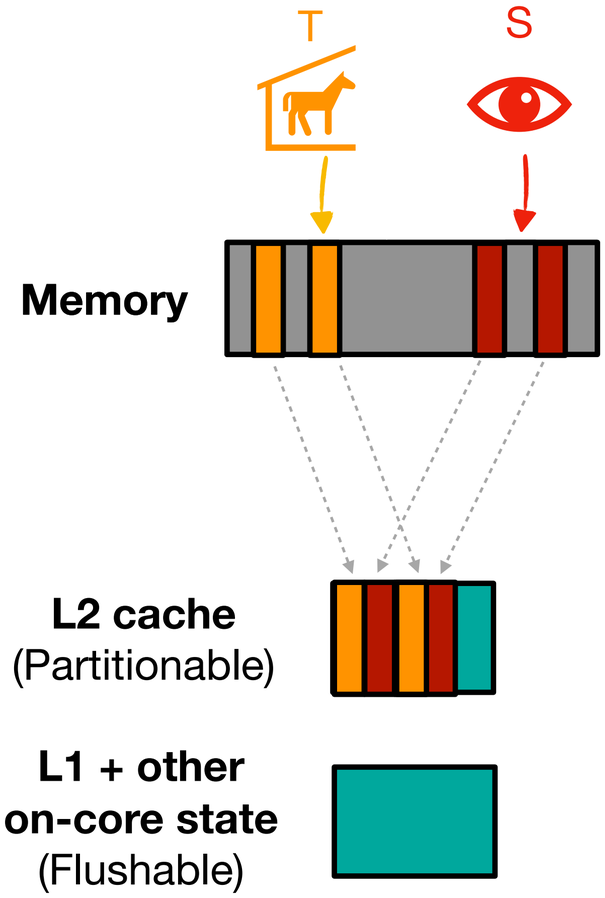
What is time protection?

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



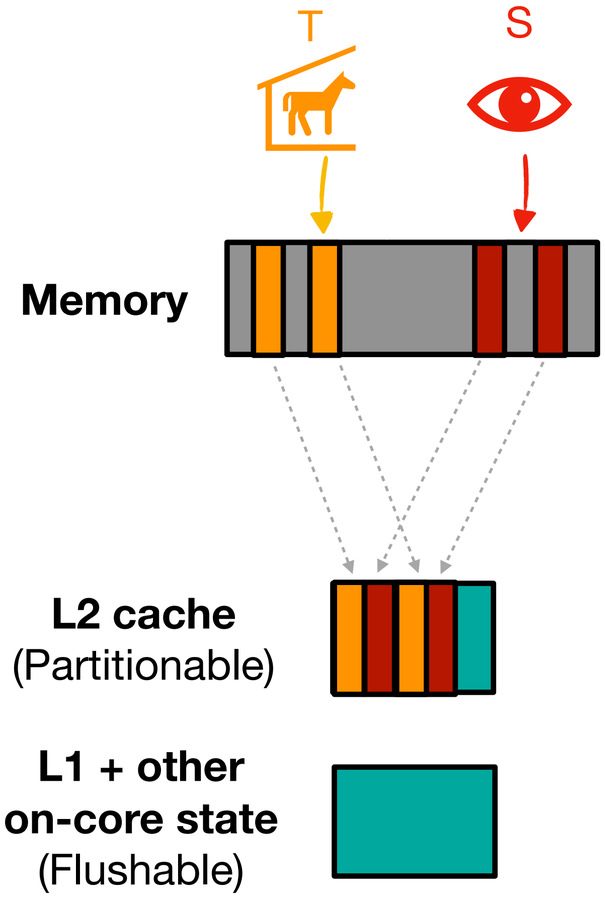
What is time protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.
- To prevent these *timing channels*, OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches



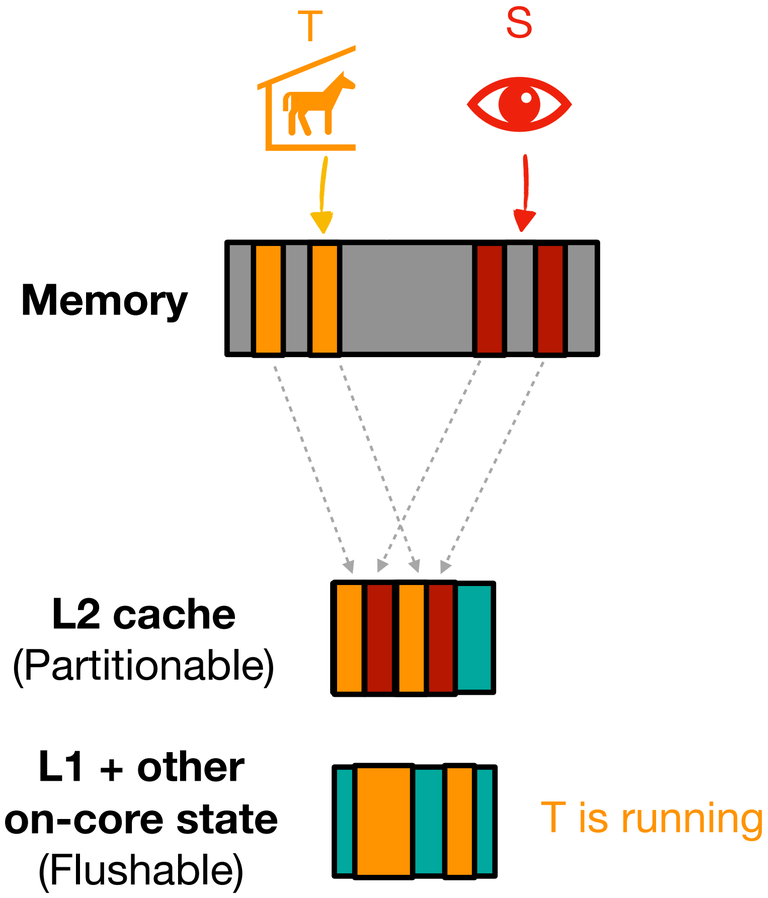
What is time protection?

- 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



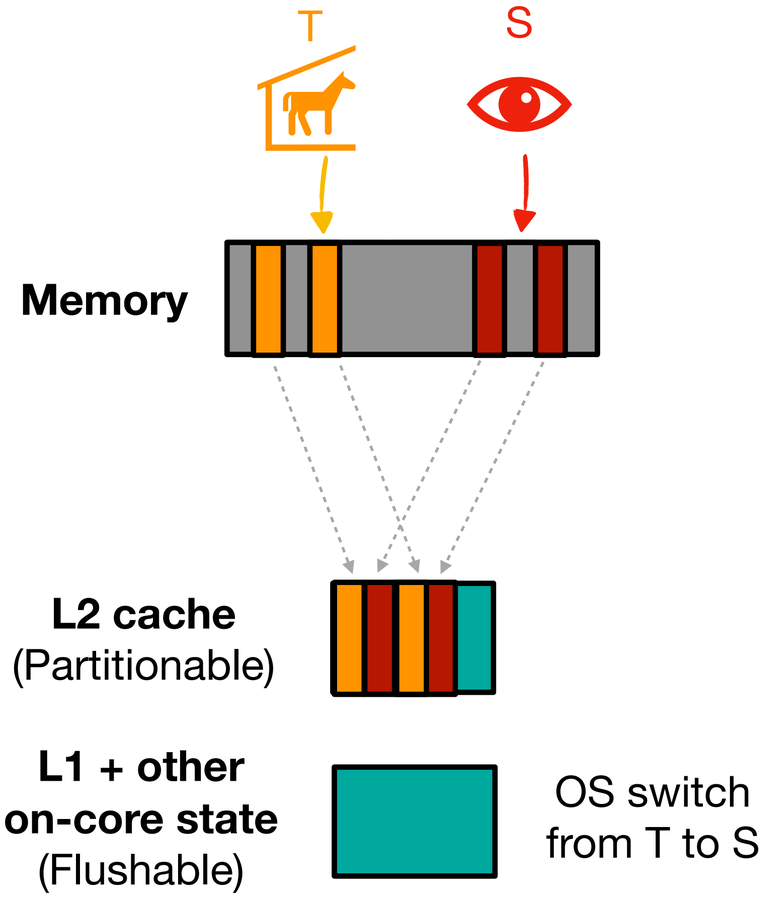
What is time protection?

- 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



What is time protection?

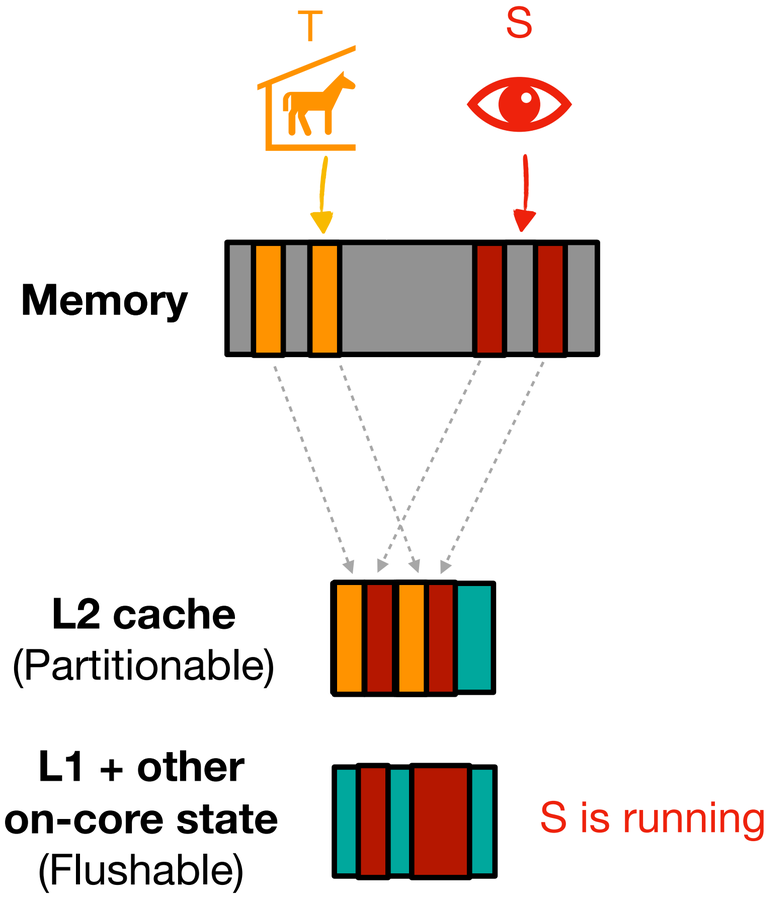
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.
- To prevent these *timing channels*, OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch



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

What is time protection?

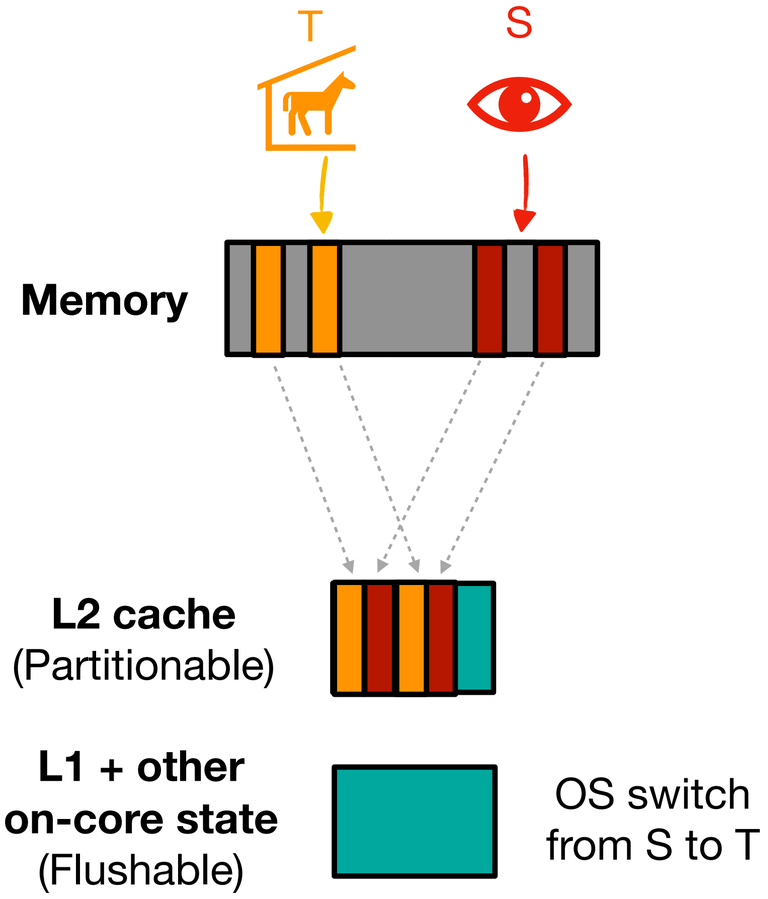
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.
- To prevent these *timing channels*, OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch



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

What is time protection?

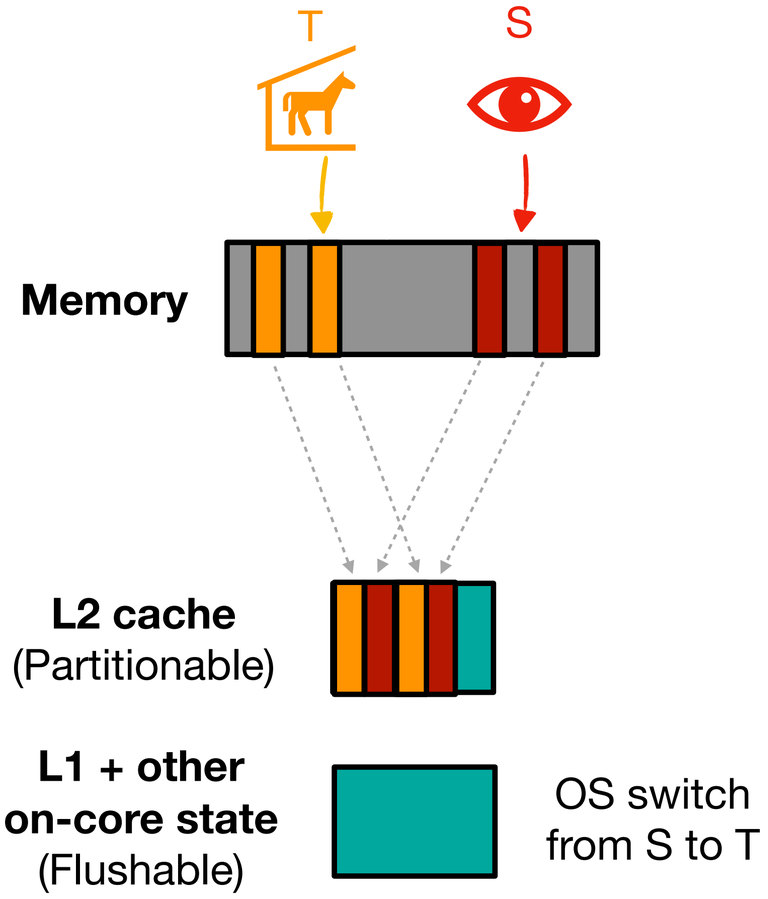
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.
- To prevent these *timing channels*, OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch



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

What is time protection?

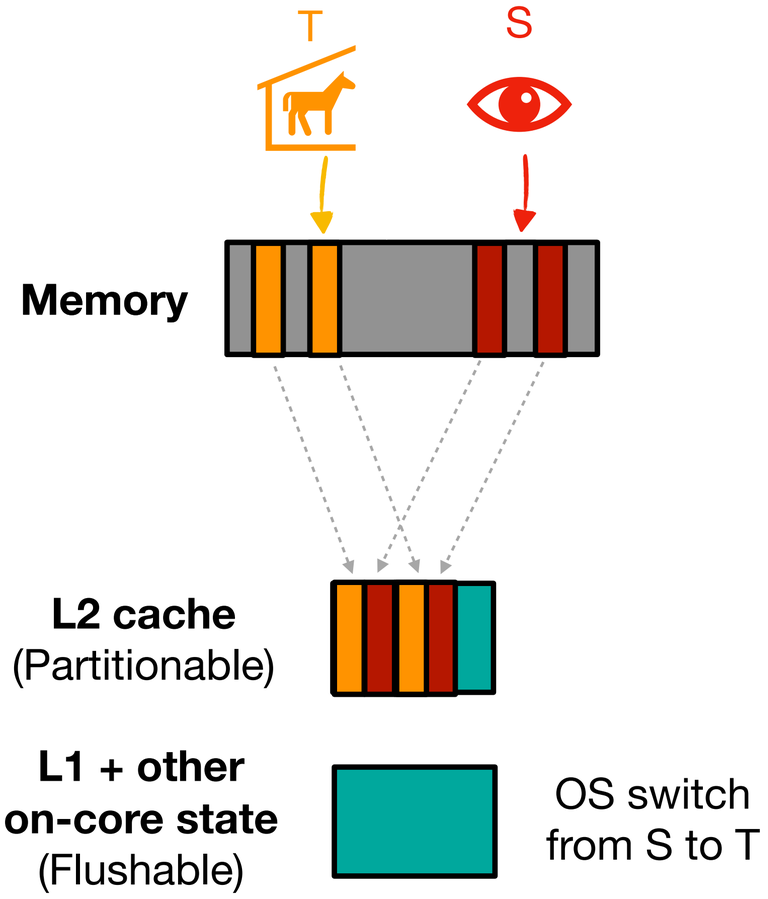
- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.
- To prevent these *timing channels*, OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- seL4 OS kernel's enforcement of time protection:
  - Implemented, evaluated empirically on ARM, x86
    See EuroSys: [Ge et al. 2019]



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

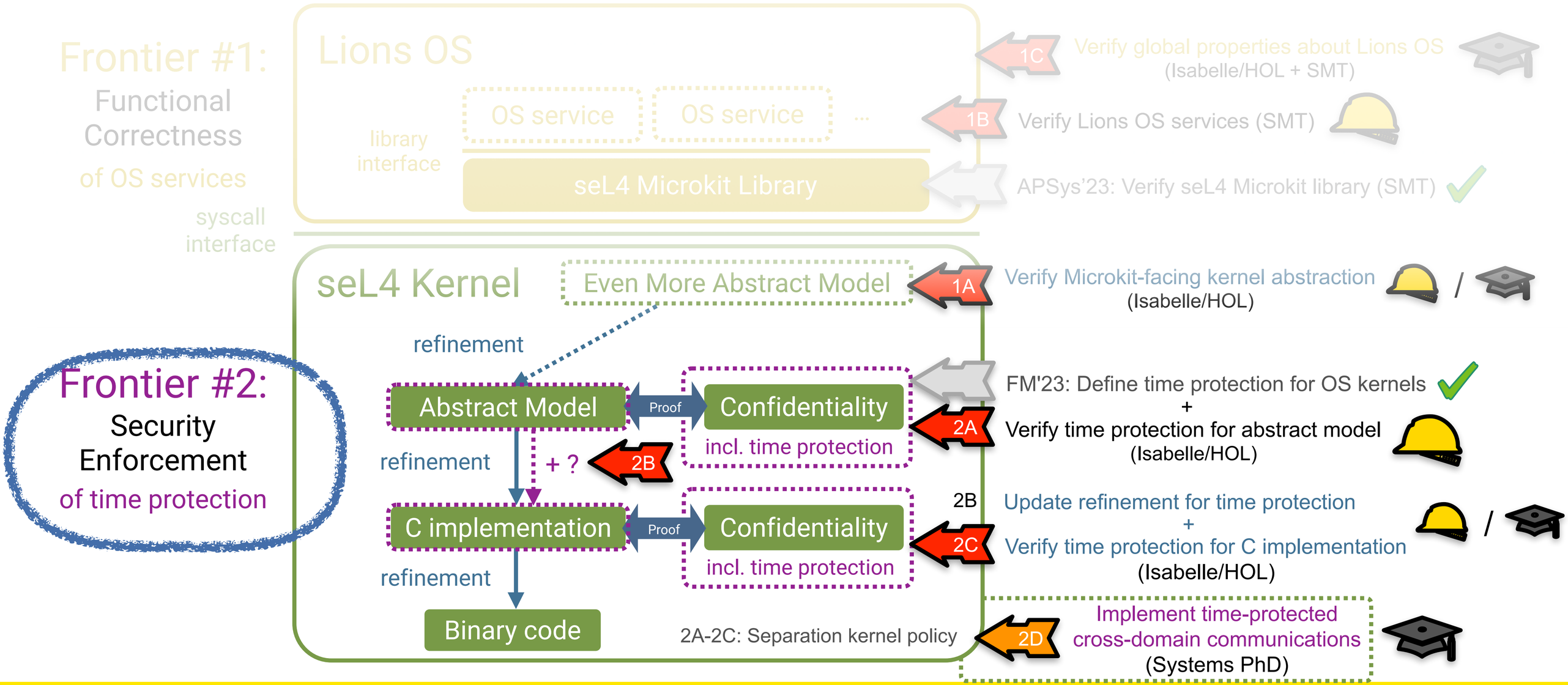
What is time protection?

- OSes typically implement *memory protection.*
- But: Mere memory access changes microarchitectural state — this affects timing.
- To prevent these *timing channels*, OSes can implement *time protection*:
  See EuroSys: [Ge et al. 2019]
  - *Partition* off-core memory caches
  - *Flush* on-core and non-architected state and *pad* time on context switch
- seL4 OS kernel's enforcement of time protection:
  - 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: [Wistoff et al. 2023]

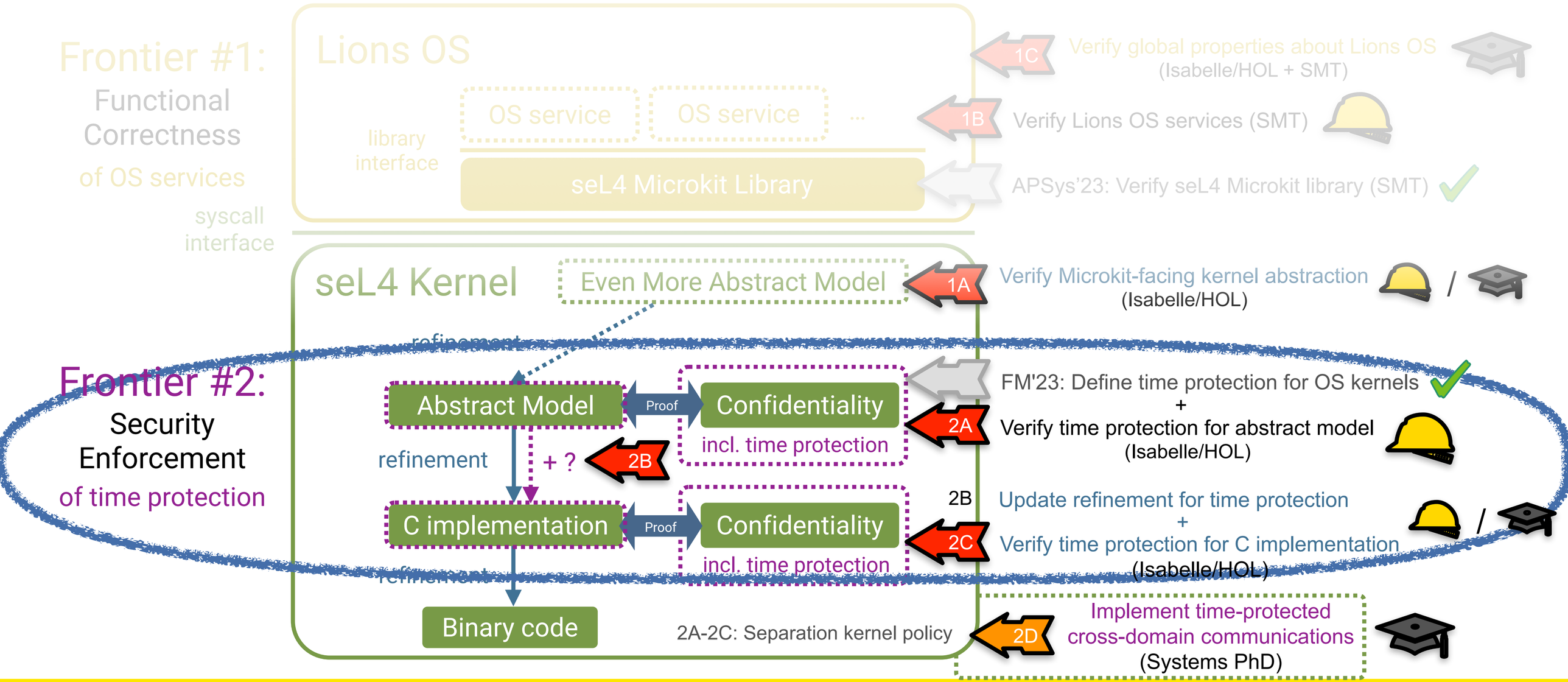


"Flush": Write fixed content; wait up to fixed time.

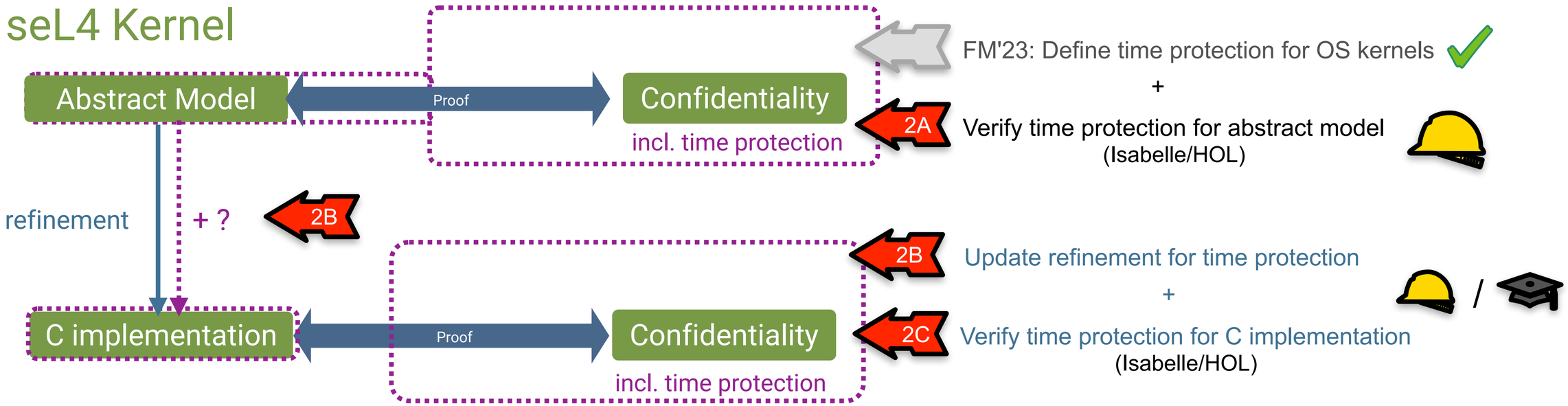
Two new frontiers for seL4 refinement



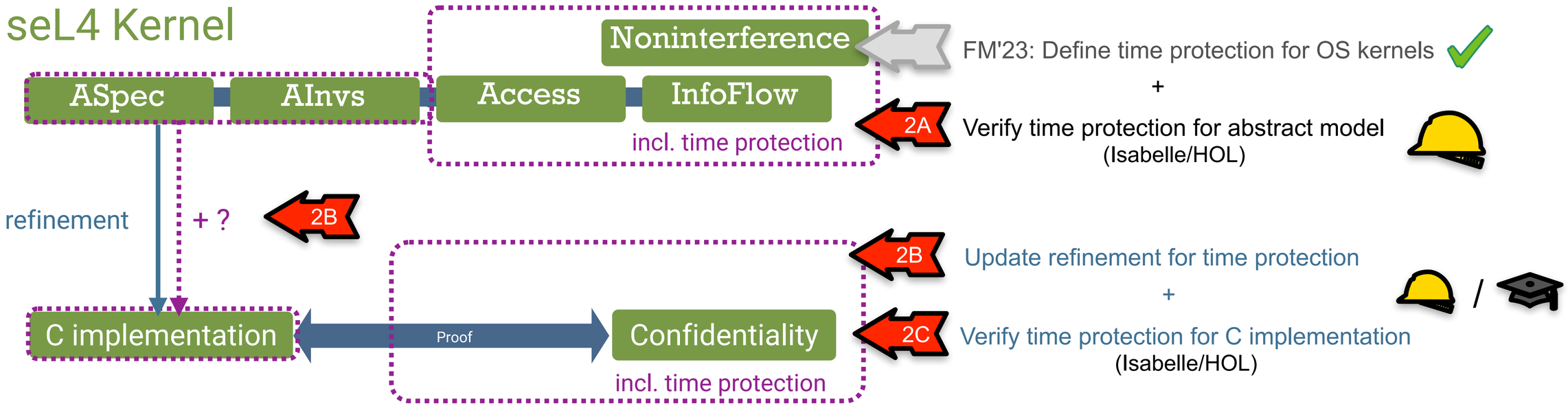
Two new frontiers for seL4 refinement



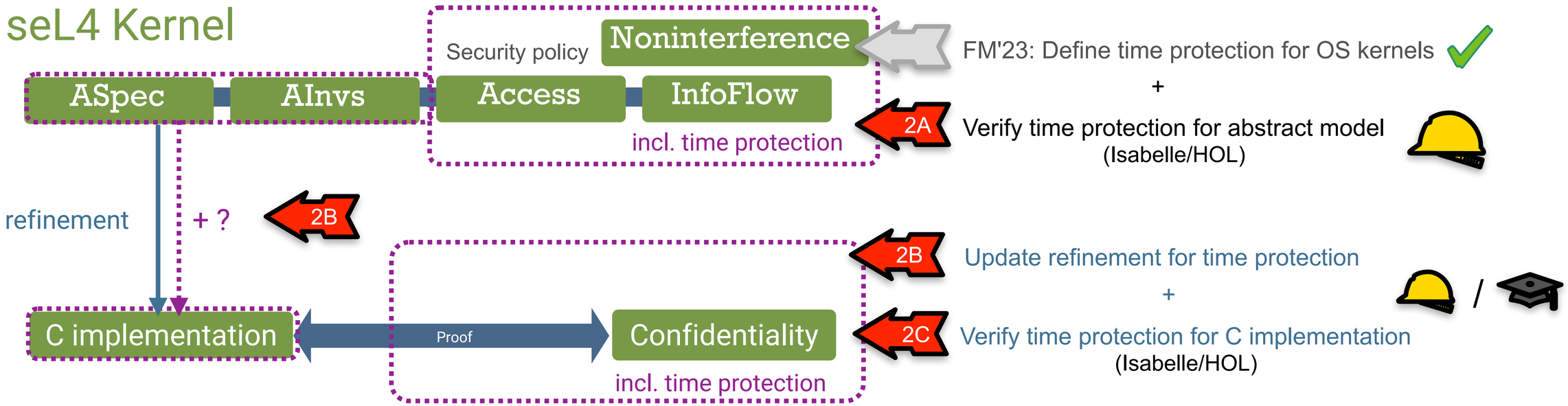
Proving seL4 implements time protection



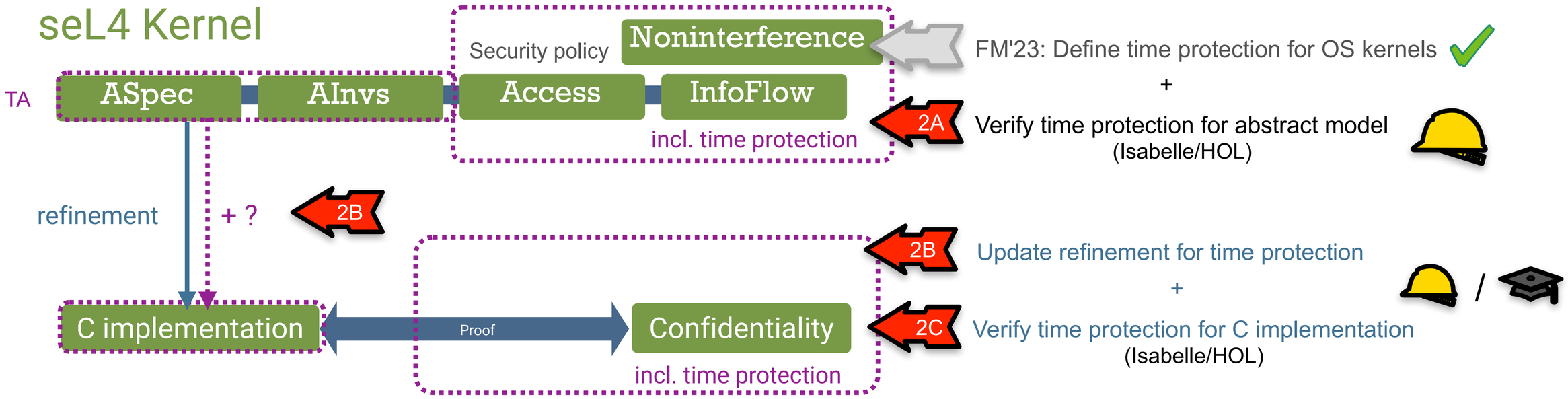
Proving seL4 implements time protection



Proving seL4 implements time protection

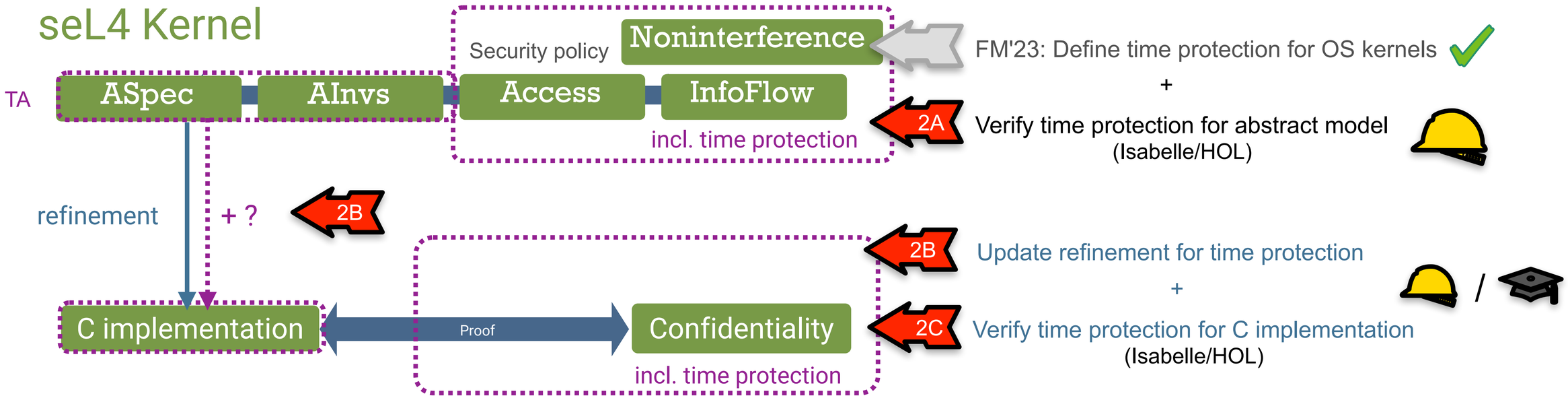


Proving seL4 implements time protection



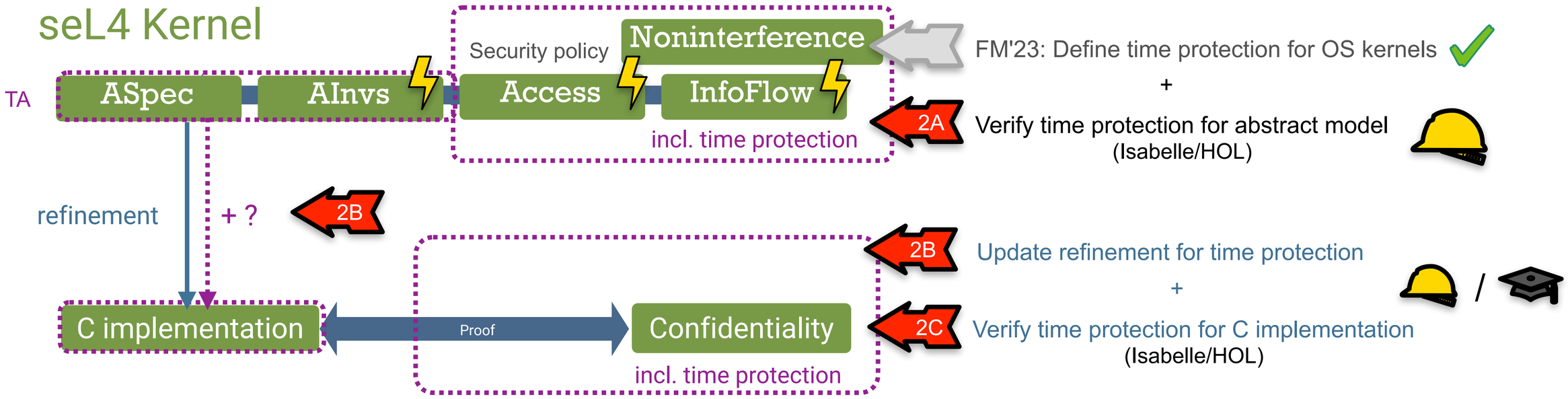
- Abstract model (**ASpec**) checks *touched addresses* (TA) set

Proving seL4 implements time protection



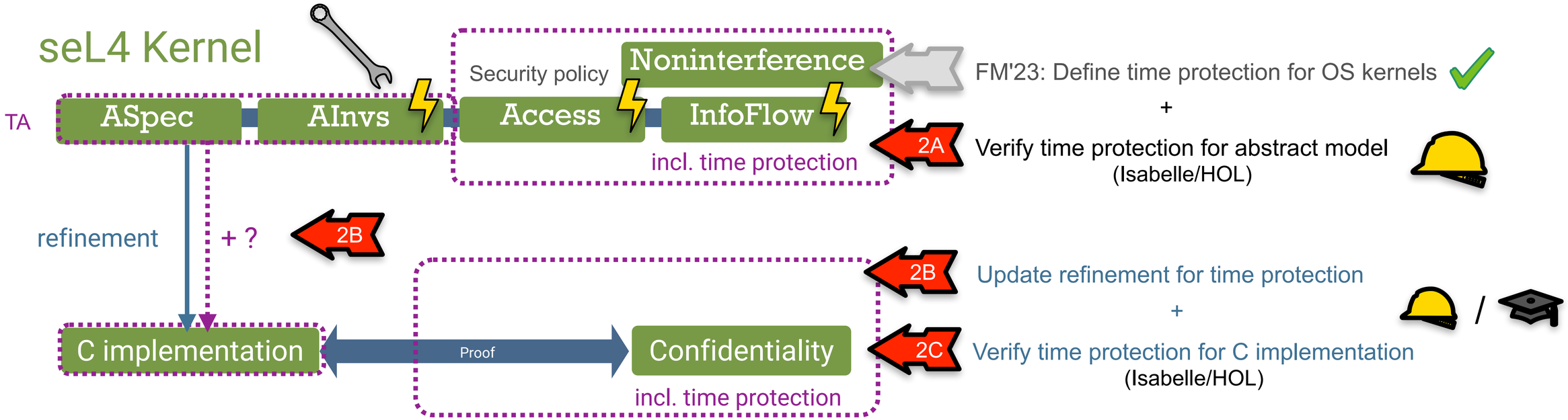
- Abstract model (**ASpec**) checks *touched addresses* (TA) set
- (2A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)

Proving seL4 implements time protection



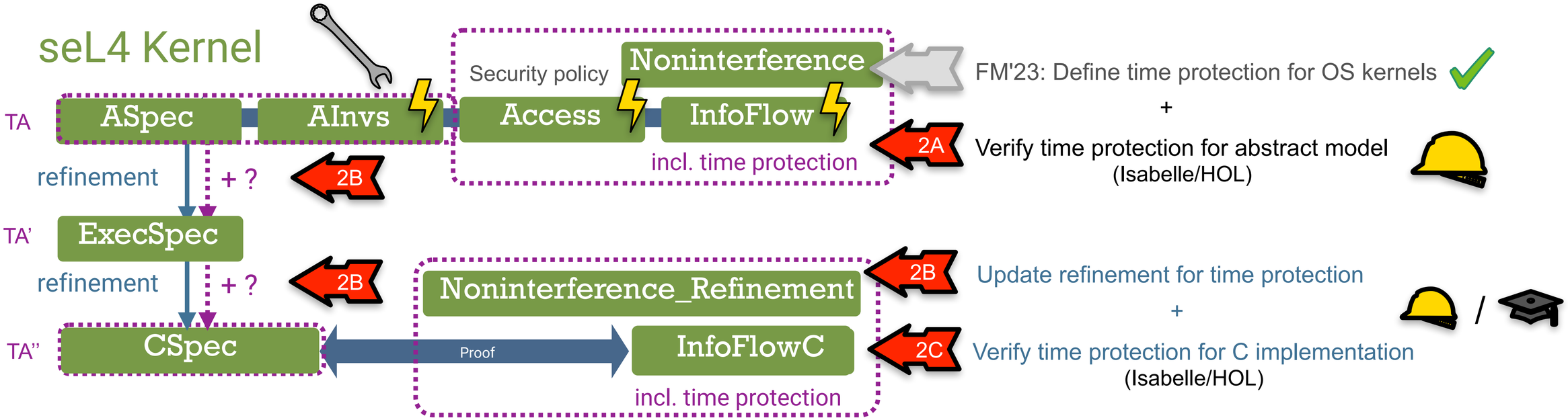
- Abstract model (**ASpec**) checks *touched addresses* (TA) set
- (2A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs

Proving seL4 implements time protection



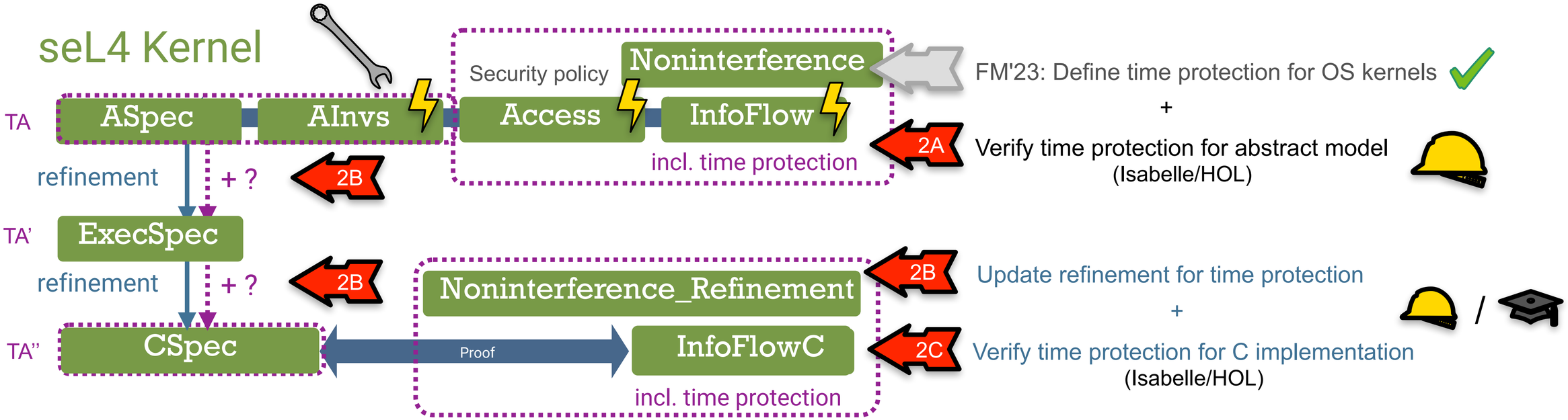
- Abstract model (**ASpec**) checks *touched addresses* (TA) set
- (2A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs

Proving seL4 implements time protection



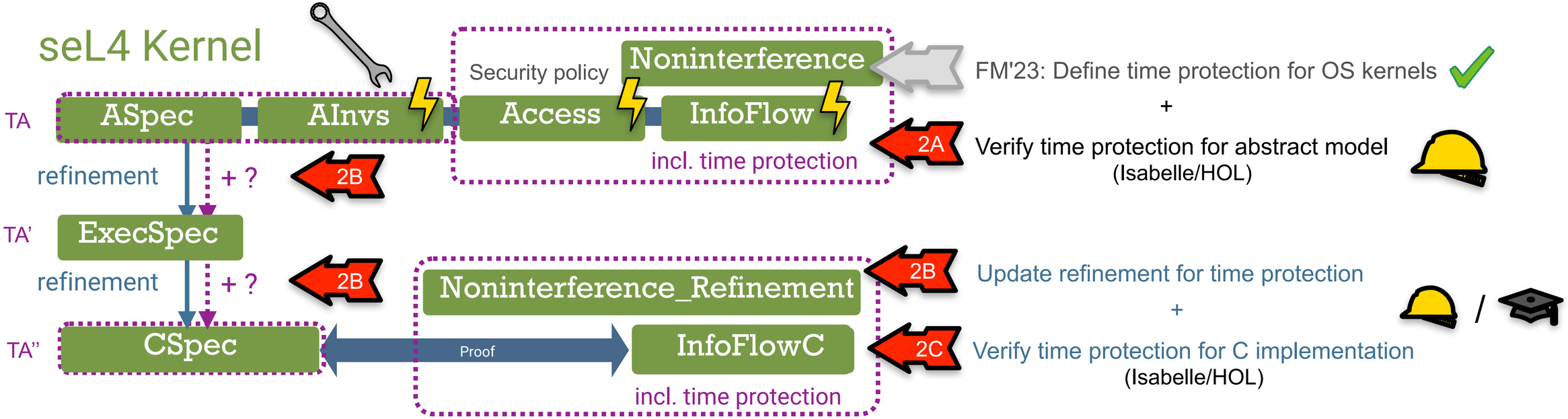
- Abstract model (**ASpec**) checks *touched addresses* (TA) set
- (2A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs

Proving seL4 implements time protection



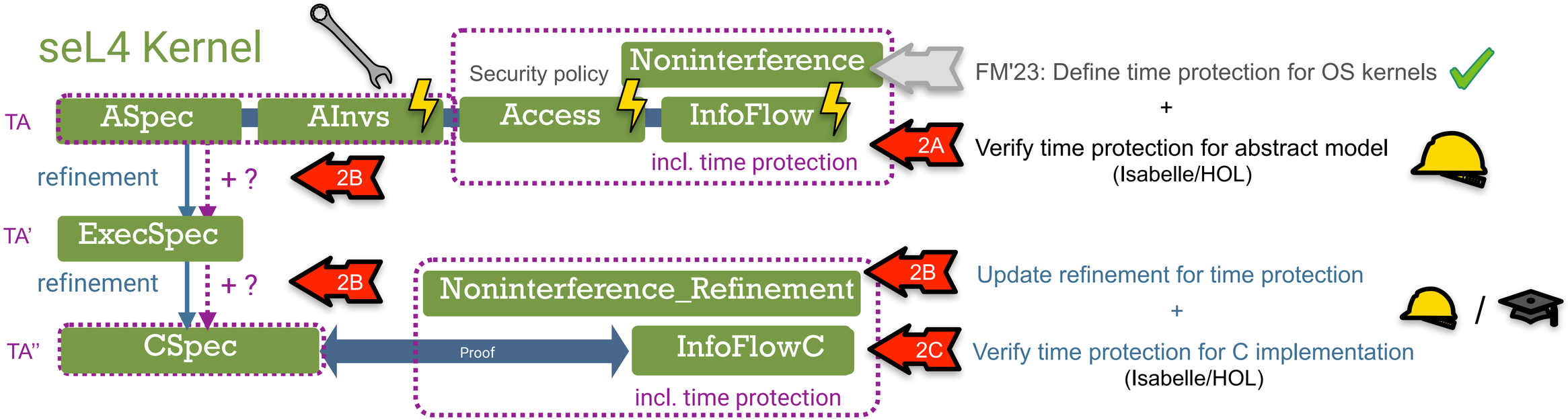
- Abstract model (**ASpec**) checks *touched addresses* (TA) set
- (2A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs
- (2B + 2C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)

Proving seL4 implements time protection



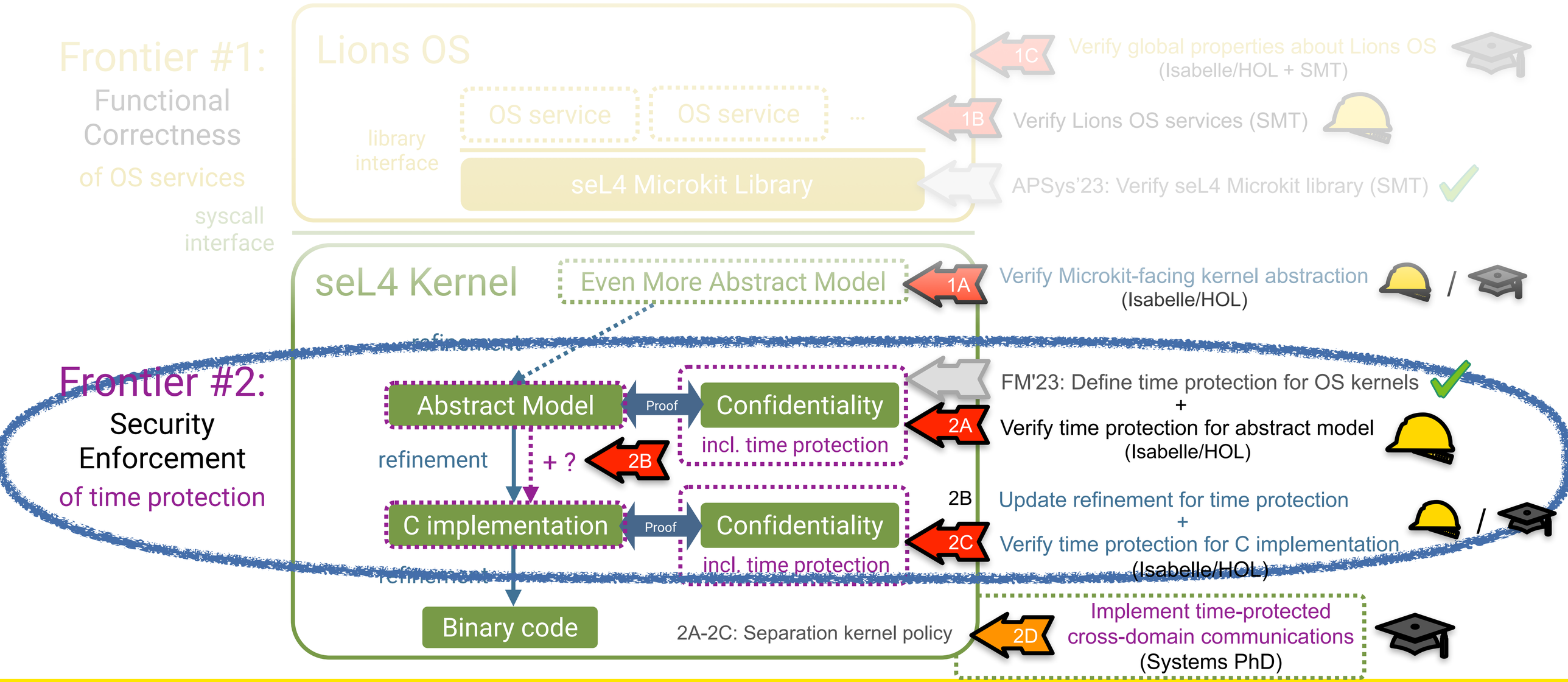
- Abstract model (**ASpec**) checks *touched addresses* (TA) set
- (2A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs
- (2B + 2C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)
  - Tricky part #1: Refinement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

Proving seL4 implements time protection

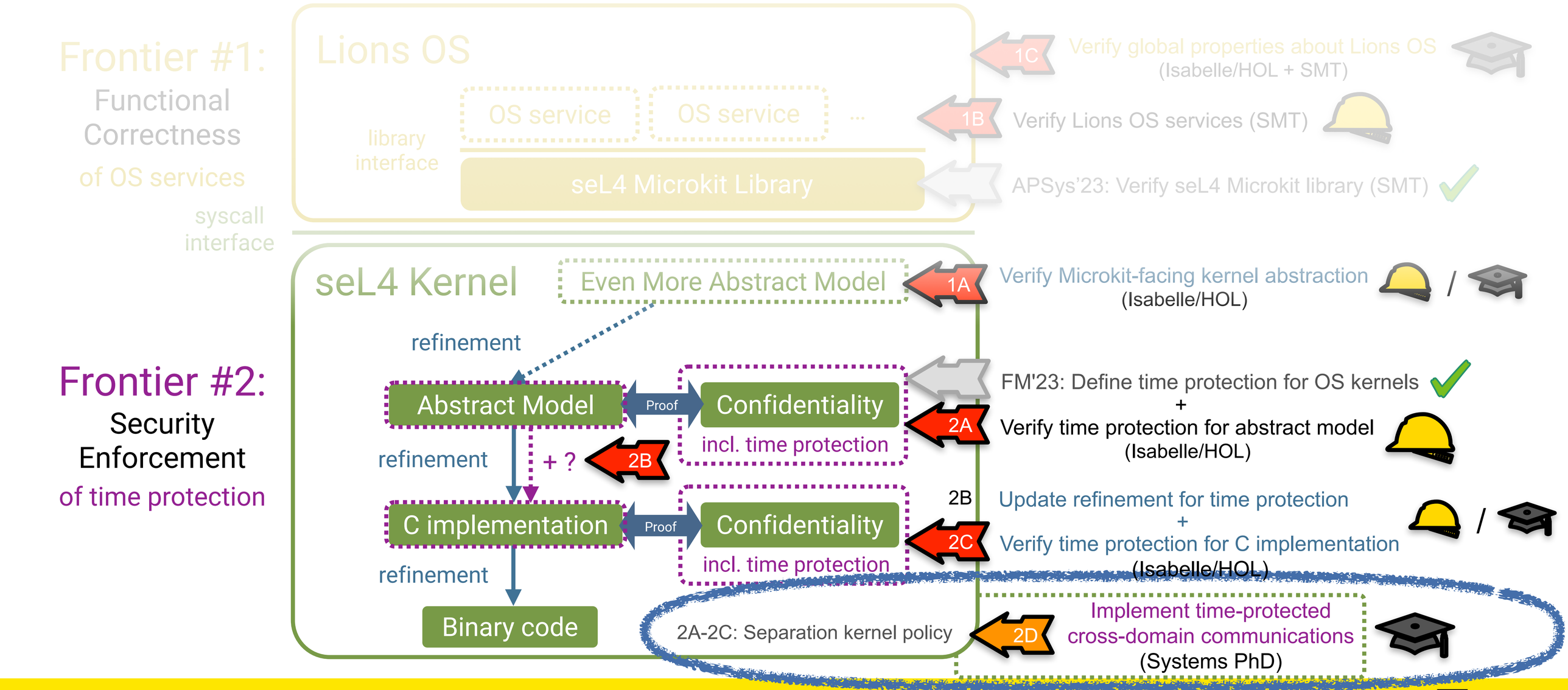


- Abstract model (**ASpec**) checks *touched addresses* (TA) set
- (2A) Key **ASpec** property: TA ⊆ *domain's addresses* (according to *security policy*)
  - Also: **AInvs**, **Access**, **InfoFlow** need repairs
- (2B + 2C) Refinement: TA'' ⊆ TA' ⊆ TA ? (via **ExecSpec**)
  - Tricky part #1: Refinement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

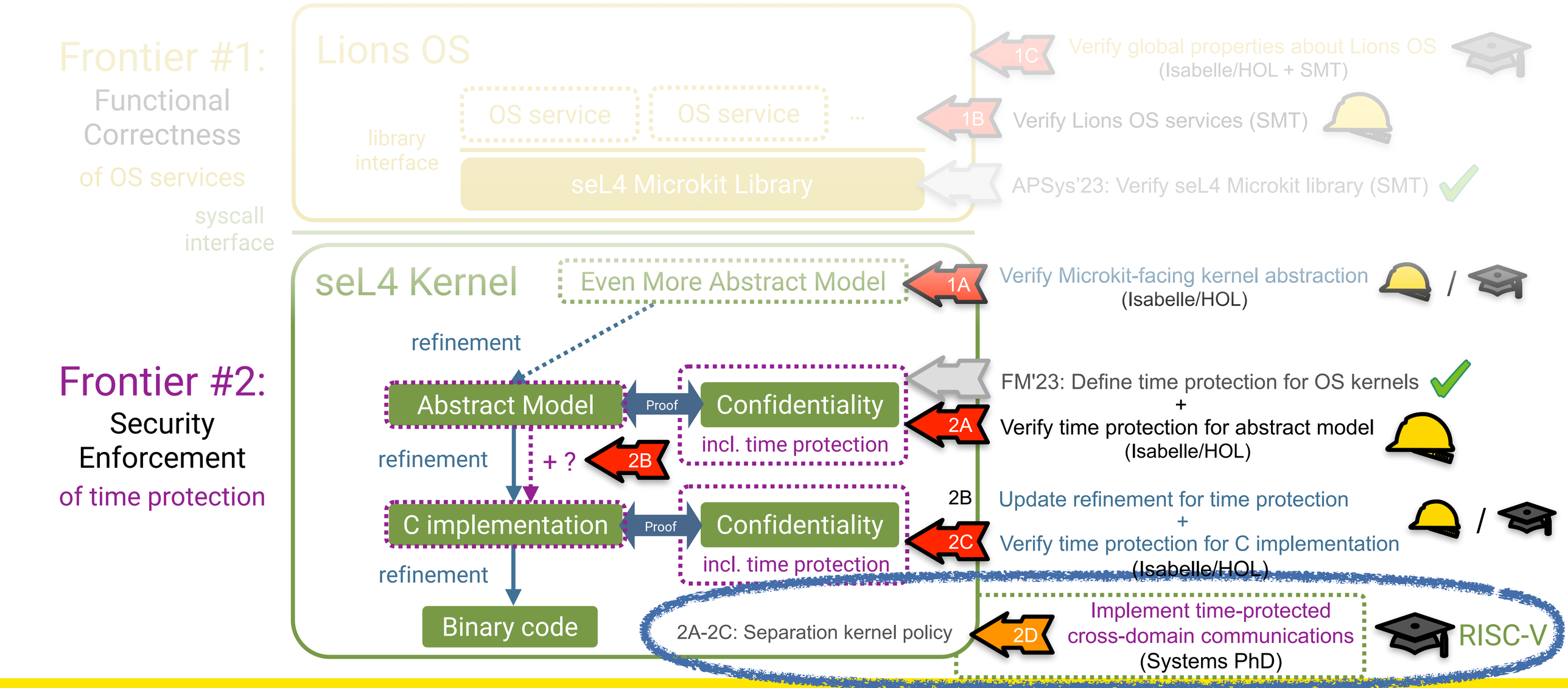
Two new frontiers for seL4 refinement



Two new frontiers for seL4 refinement



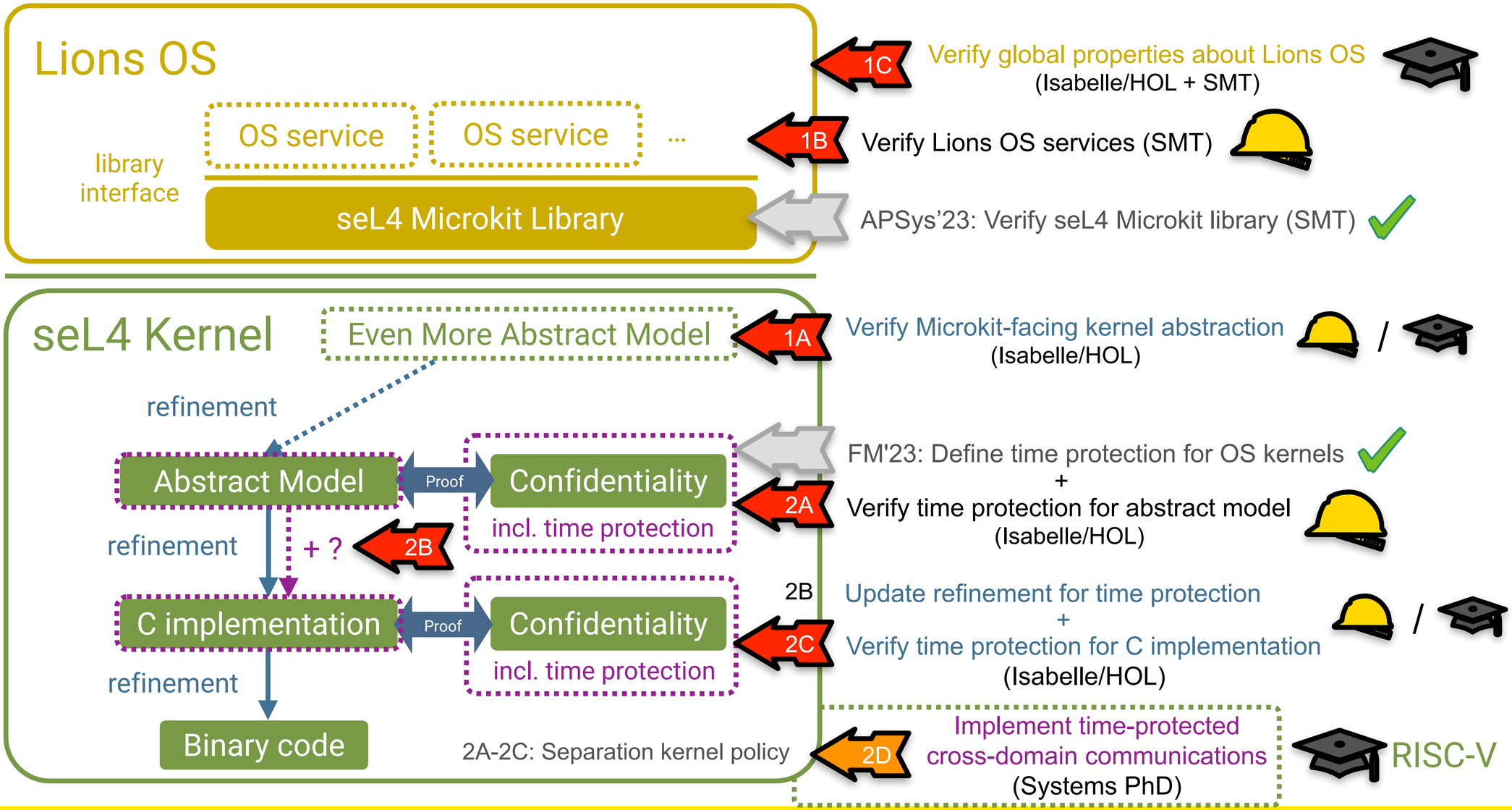
Two new frontiers for seL4 refinement



Two new frontiers for seL4 refinement

Frontier #1:
Functional Correctness
of OS services

Frontier #2:
Security Enforcement
of time protection



Trustworthy Systems @ CSE, UNSW Sydney

Thank you! Questions?

Frontier #1:
Functional Correctness
of OS services

Frontier #2:
Security Enforcement
of time protection