Adapting to demand

seL4 proofs and engineering practice

Matthew Brecknell | Sep 2020

The impact of history

Microsoft Windows:

- 1981: MS-DOS
- 1985: Windows 1.0
- 1993: Windows NT 3.1
- …
- 2014: Windows 10

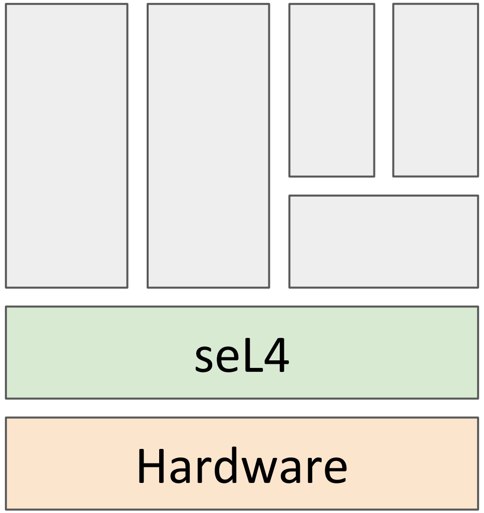
How is this possible?

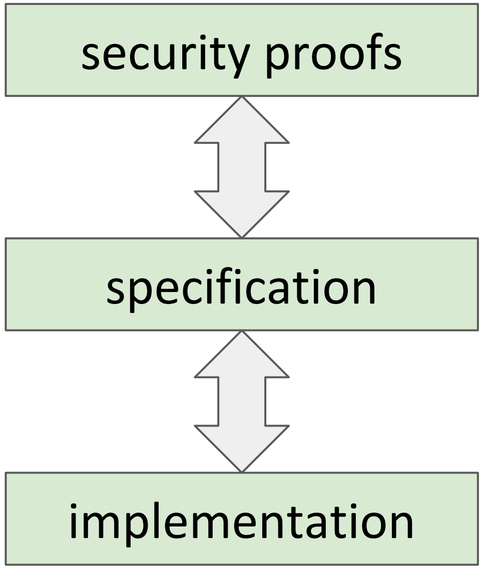
What were the challenges?

Today’s talk

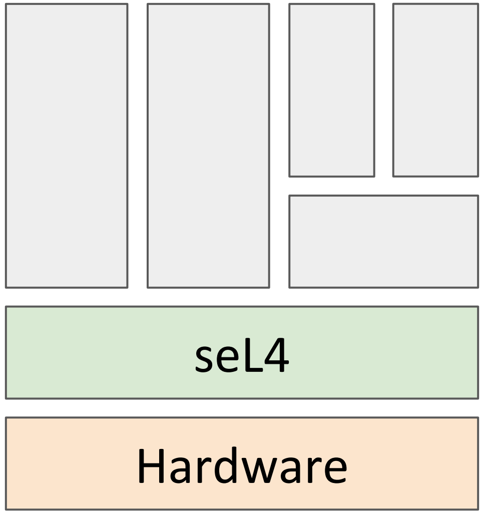
- Current challenges in seL4 proof engineering
- History and its impact
- What we’re doing with those challenges

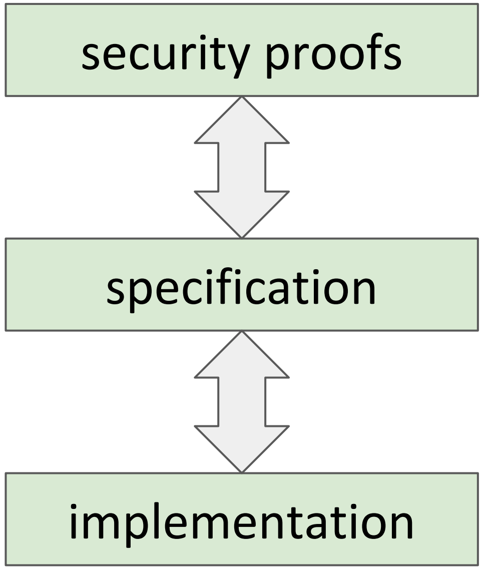
What is seL4?


seL4
Hardware

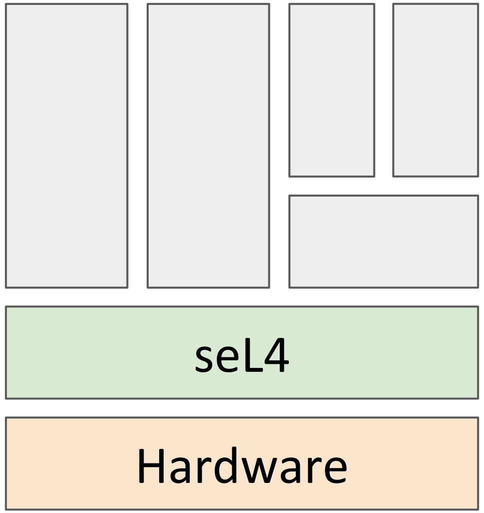

security proofs
specification
implementation

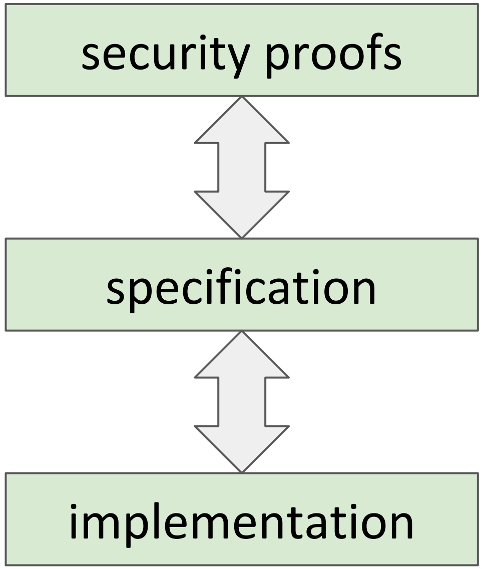
What is seL4?


seL4
Hardware


security proofs
specification
implementation

What is seL4?


seL4
Hardware


security proofs
specification
implementation

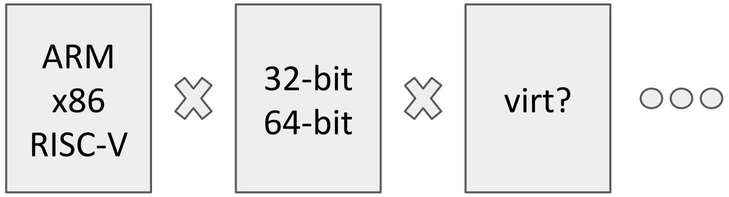
Current challenges in seL4 verification

Research and development

- Multicore
- Time
  - Integrity: mixed-criticality real time
  - Confidentiality: timing channels

Engineering practice

- Achieving agility
- Matrix problem



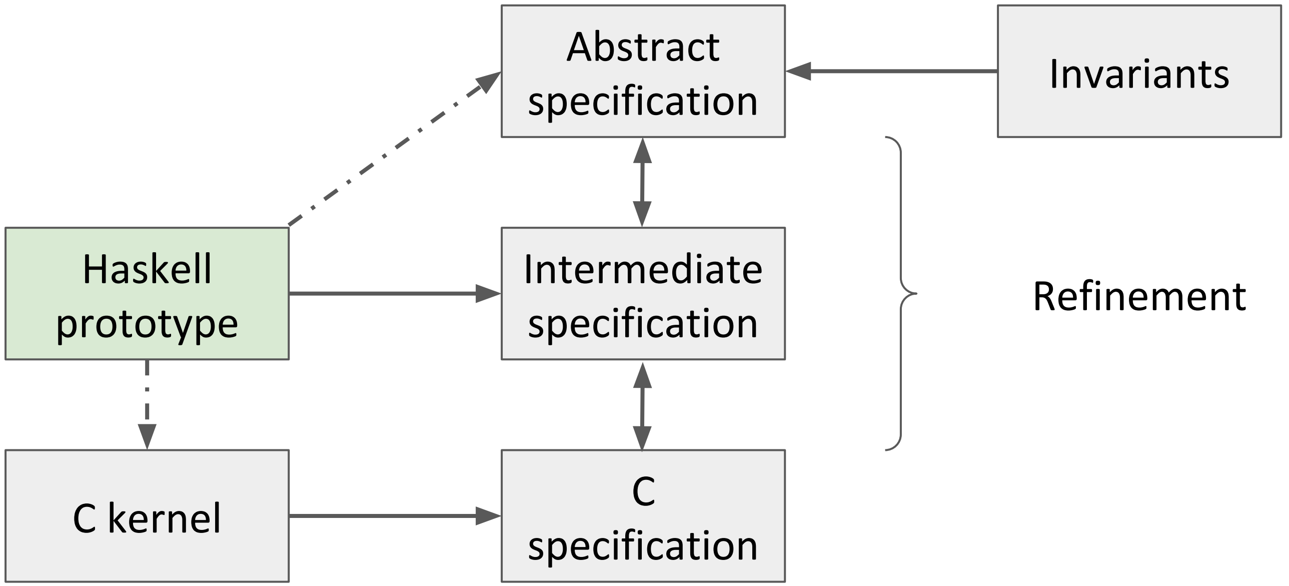
The impact of history (seL4 edition)

- Our proof architecture reflects our initial prototype.
  - Artefacts become overheads.

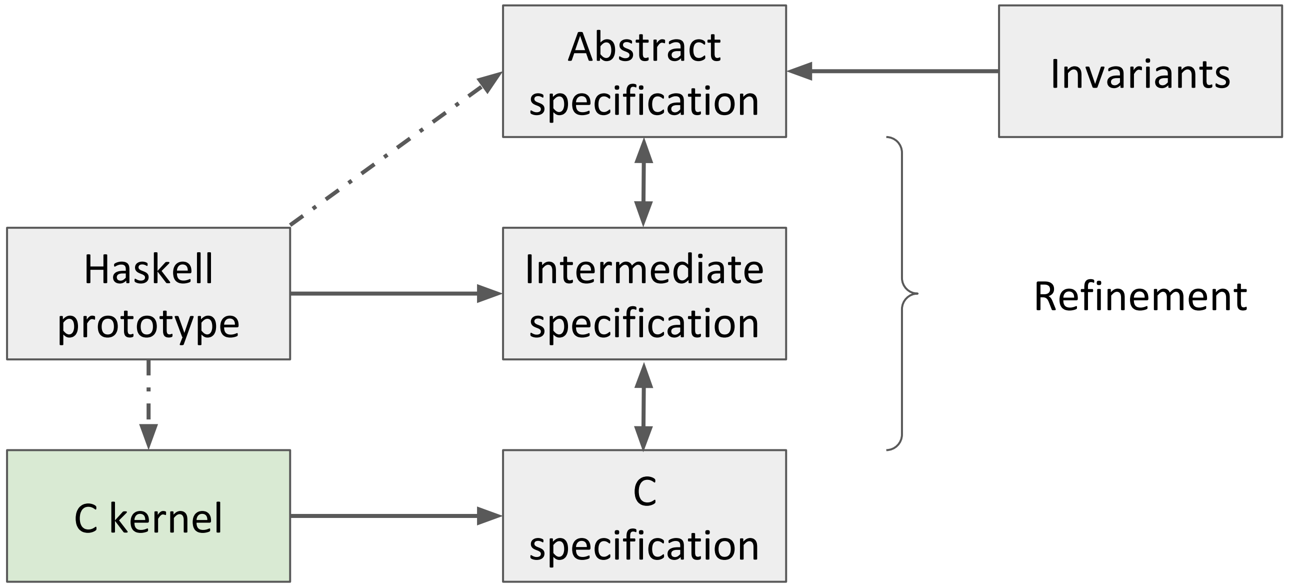
- In the beginning, we had to limit scope.
  - The priority was getting something done.

- We knew less then than we know now.

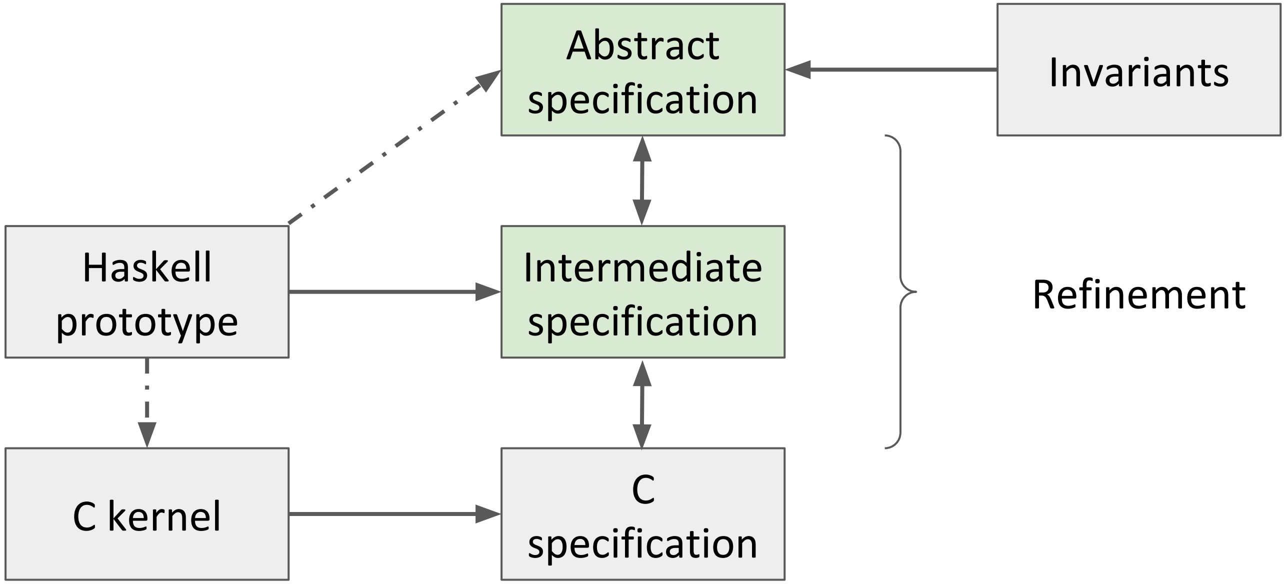
History: prototype



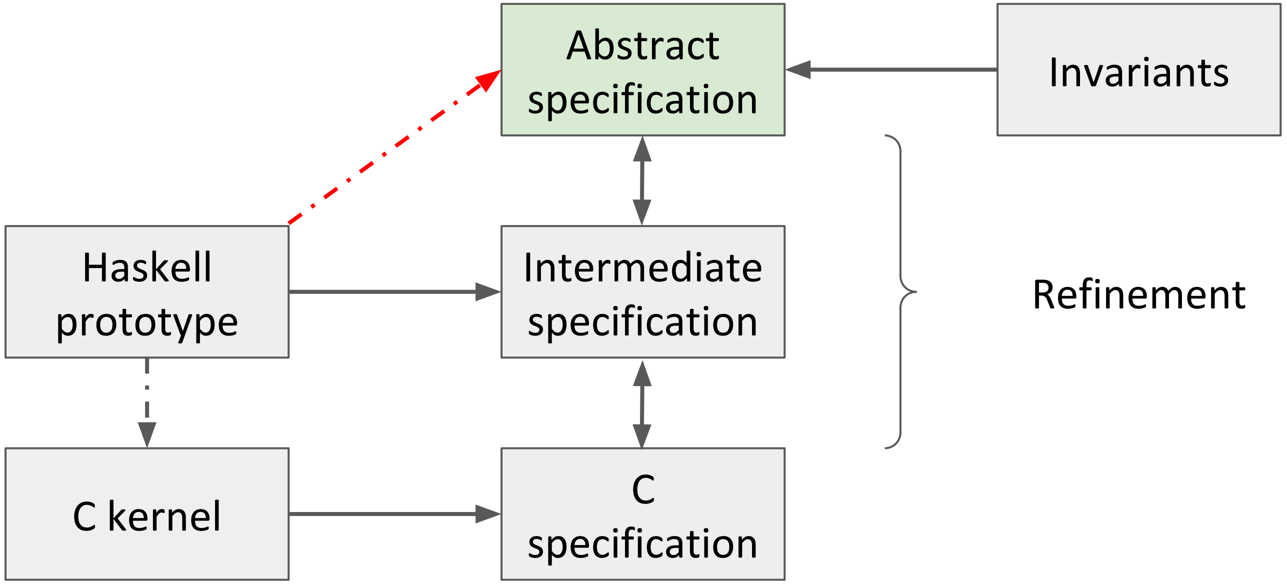
History: prototype



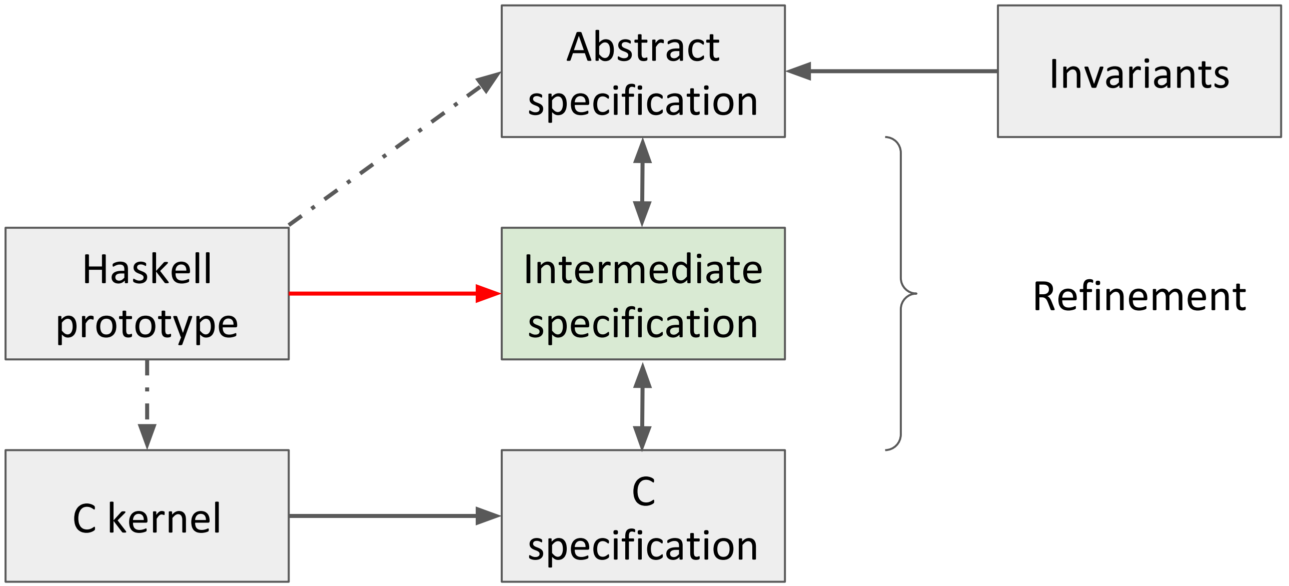
History: prototype



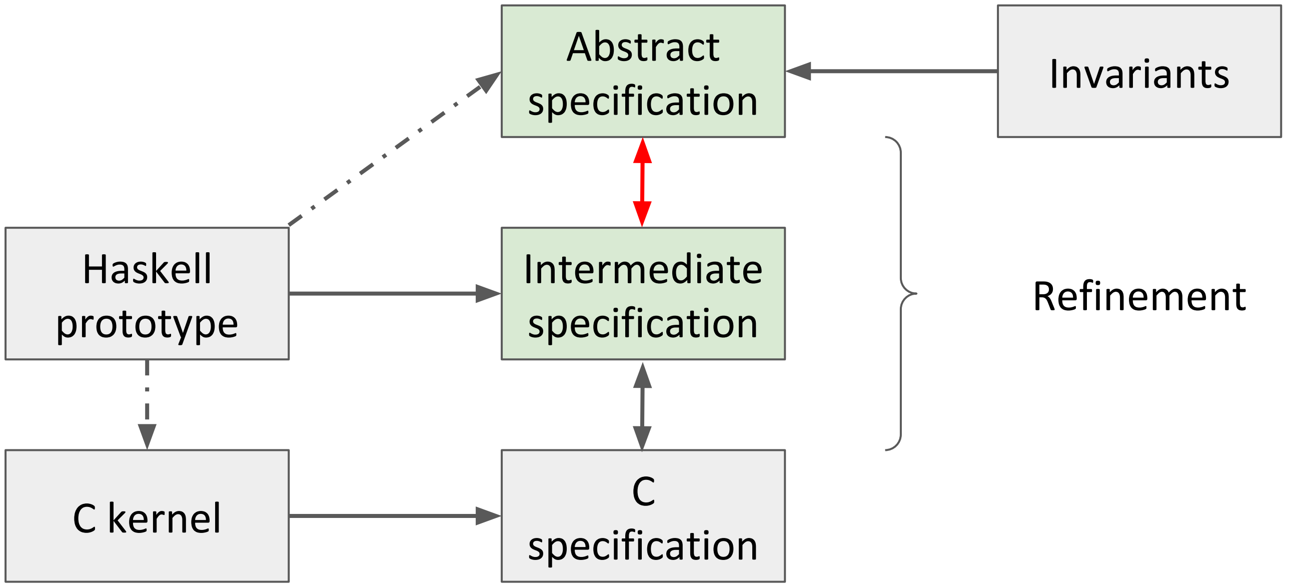
History: prototype



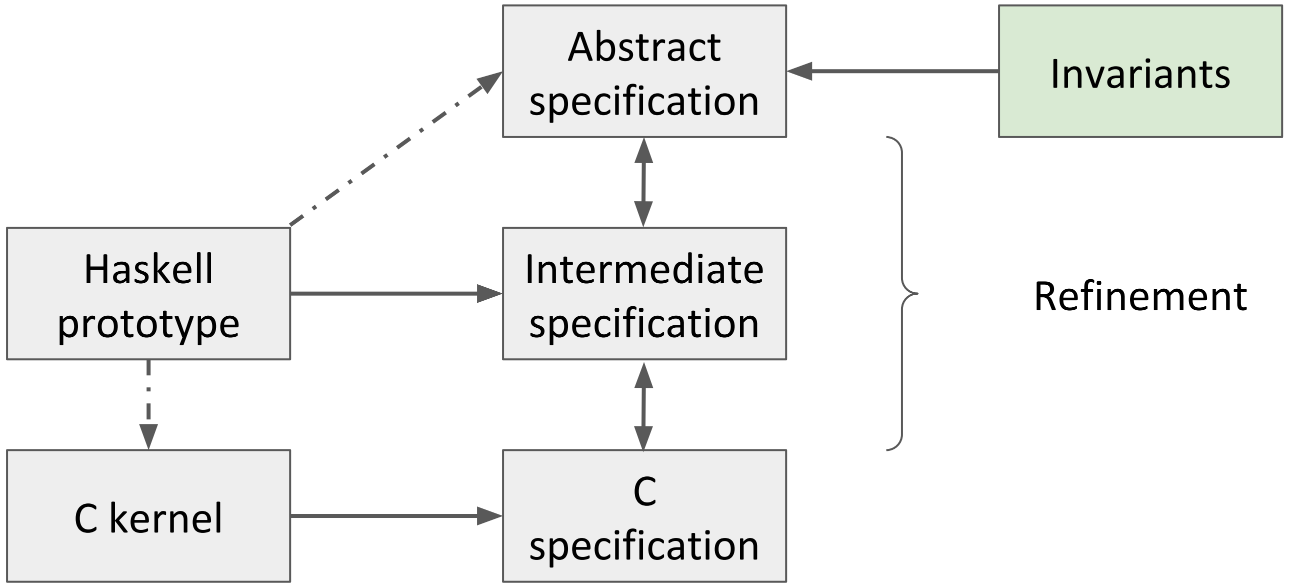
History: prototype



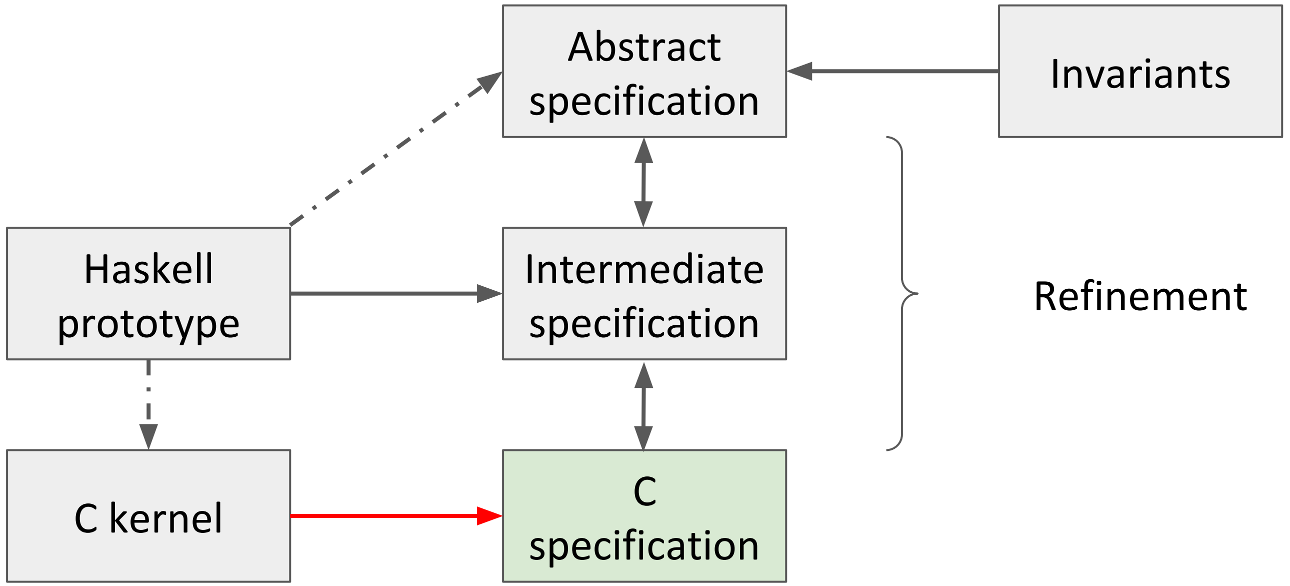
History: prototype



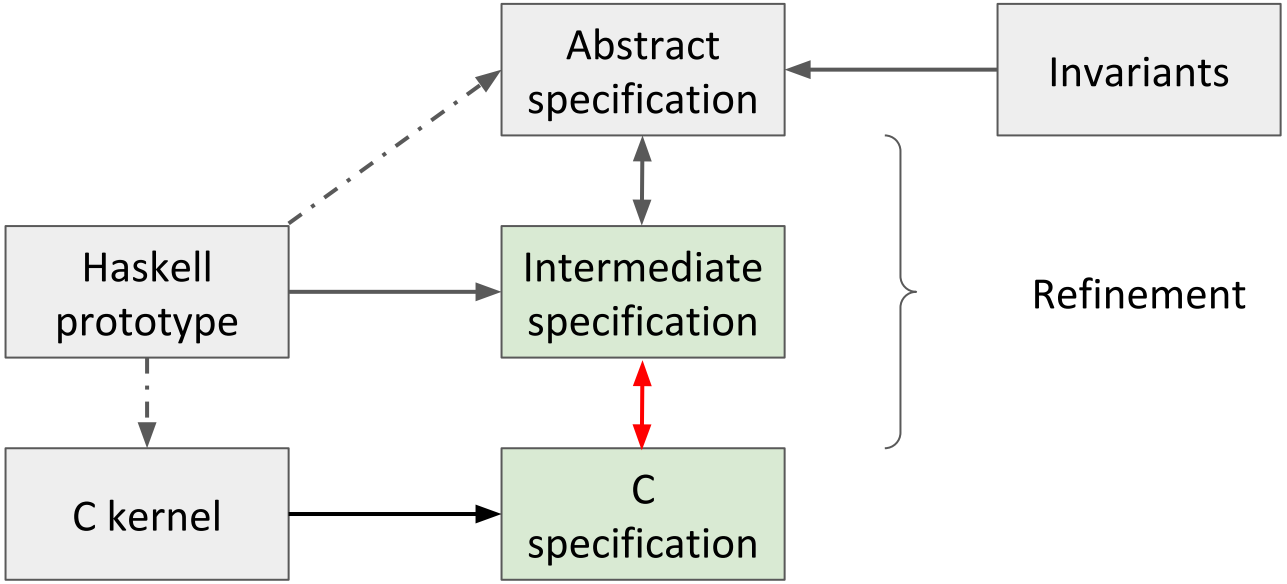
History: prototype



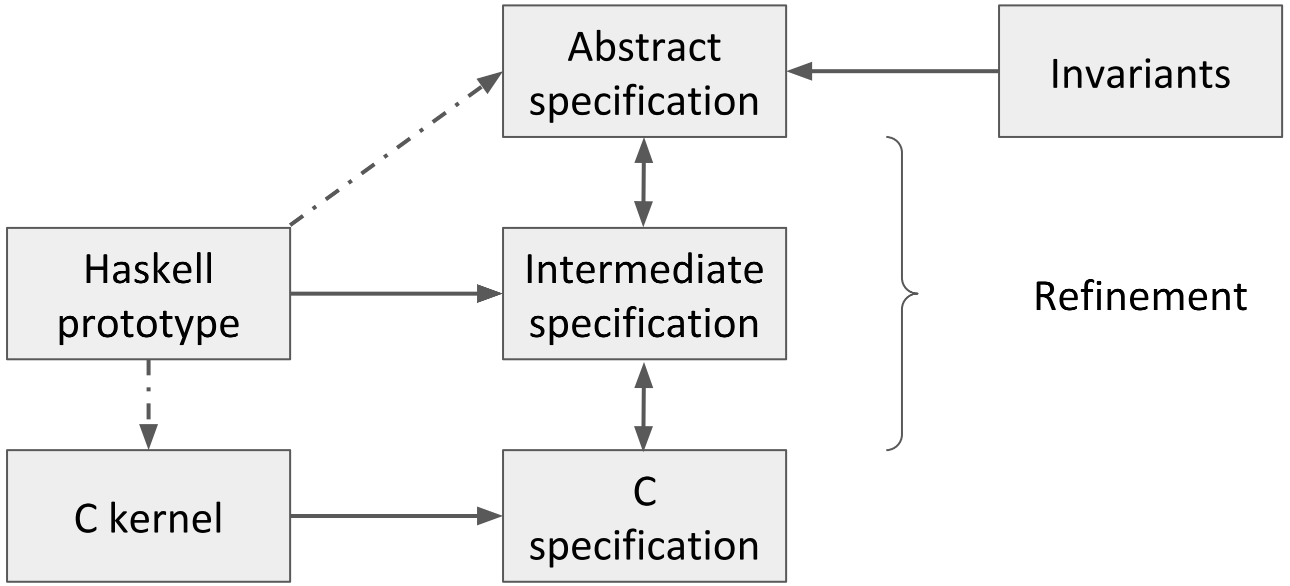
History: prototype



History: prototype



History: prototype



History: prototype

We still have the Haskell!

- But we don’t run it any more.
- Primarily a specification generator.

The upper refinement should be just a data refinement.

- But we spend time on artefacts of the prototype.

History: prototype

What to do?

- Get rid of the Haskell! Eventually.
- Increase sharing between the abstract and intermediate specifications.

History: scope

Originally:

- One instruction set architecture (ARMv7).
- One mode (32-bit).
- One platform configuration.
- One core.

History: architecture split

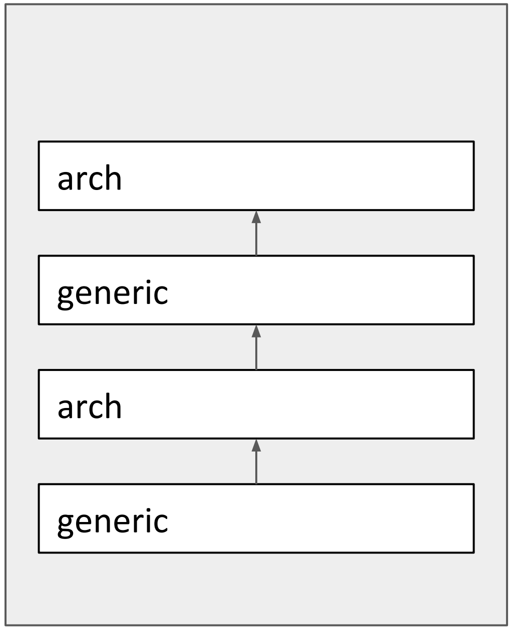
Haskell prototype
C kernel
Abstract specification
Property definitions

Conventional separation of generic and arch-specific aspects (even though only one configuration)

Virtual memory structures $\longrightarrow$ arch
Everything else (threads, IPC) $\longrightarrow$ generic

What about the proofs?

History: architecture split

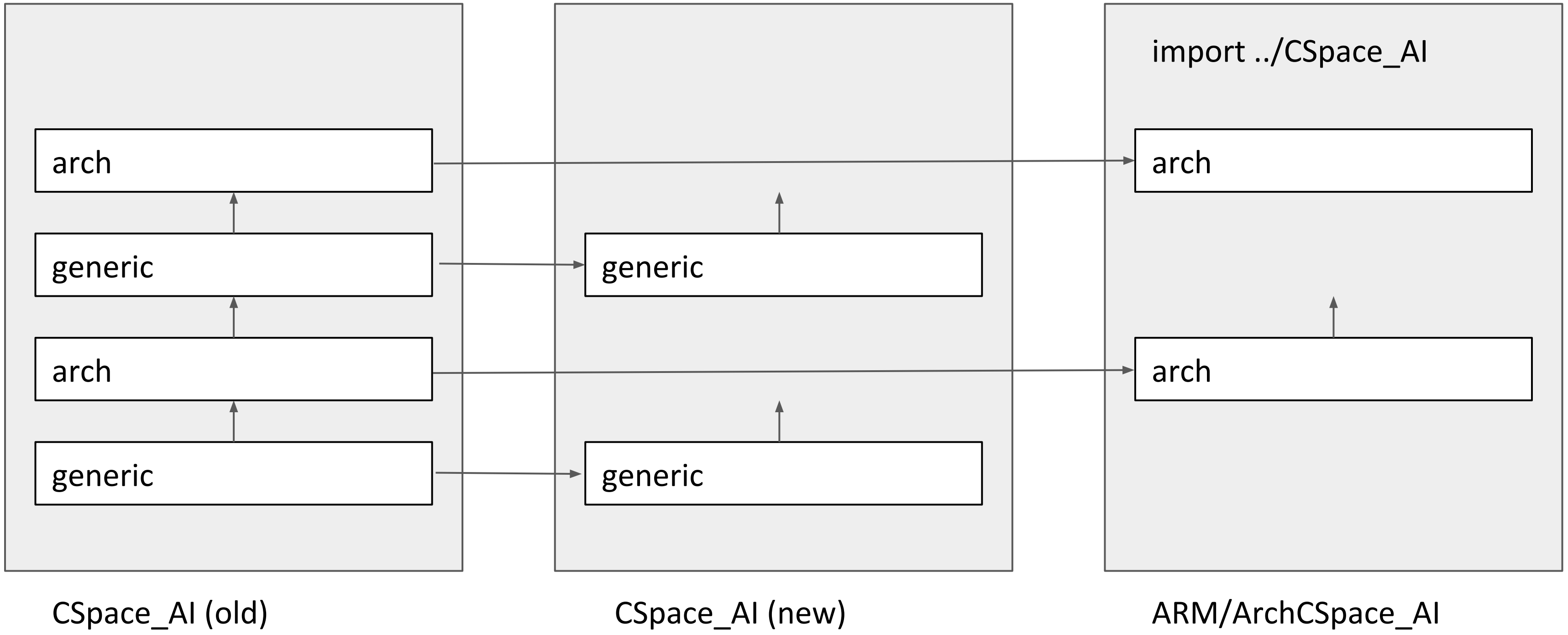


CSpace_AI (old)

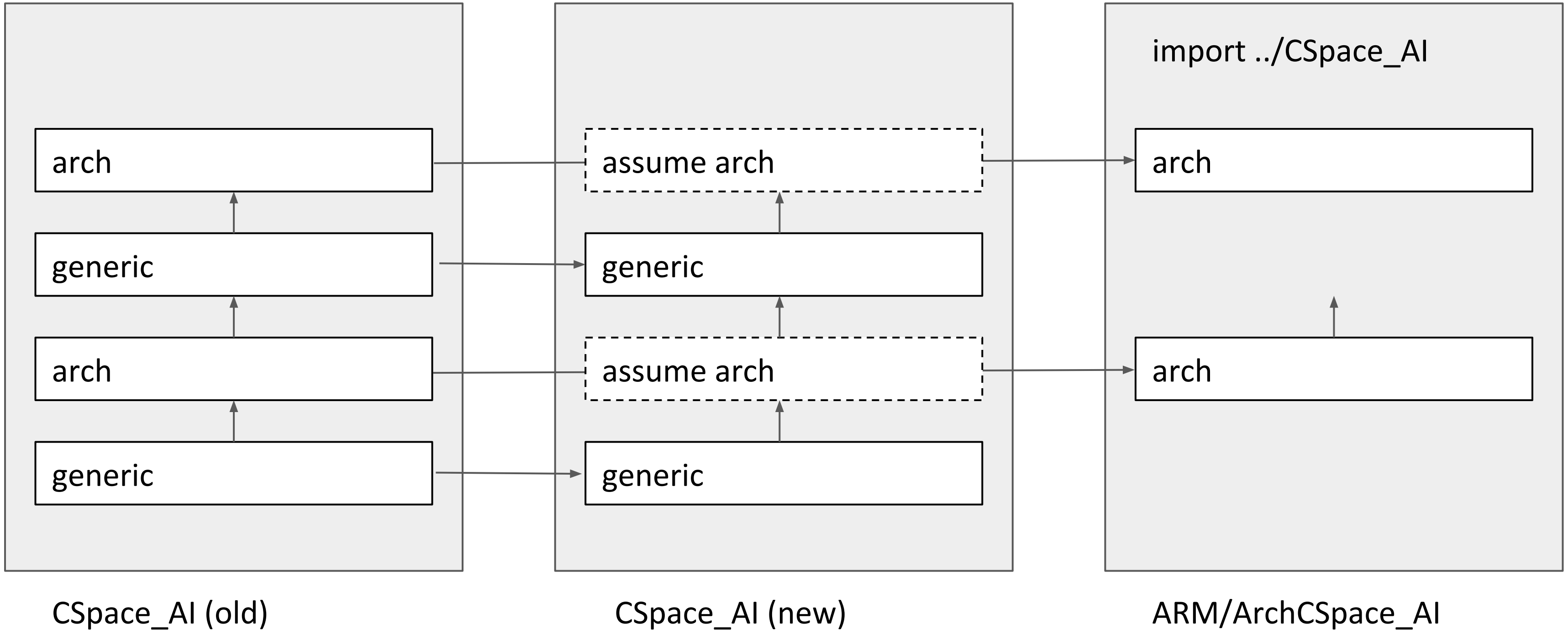
Proofs were very interdependent.

Challenge: verify more configurations (hypervisor, x86-64, RISC-V)

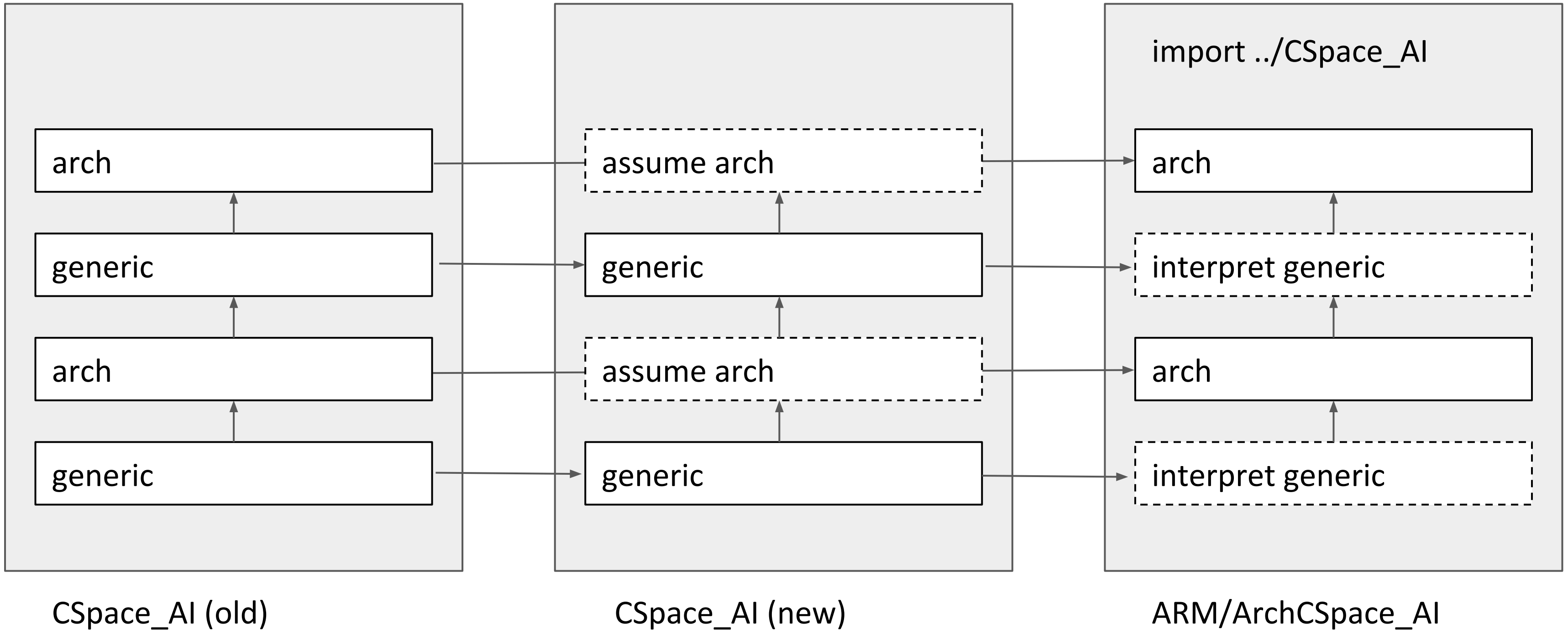
History: architecture split



History: architecture split



History: architecture split



History: architecture split

- + Lots of shared generic invariant proofs.
- + Explicit statements of what we assume about architectures.
  - - Even though we didn't look for nicer abstractions.

- - We need to check the shared proofs for each architecture.
- - Limited support for fine-grained configurability.

Configurability wish list

Easy

- Addresses of kernel base and global structures.

More challenging

- Word size: affects generic kernel structures!
- Feature switches: virtualisation, page table structures.

Impossible

- Mixed-criticality, multi-core.

Page table abstraction

Almost all arch-specific proofs are about virtual memory.

- And all VM proofs are arch-specific!

Wanted: generic theory of virtual memory mappings.

- Specialisations to ARM, x86, RISC-V.

Page table abstraction: progress

ARM

- Irregular page table structure.

RISC-V

- Perfectly regular page table structure.

RISC-V showed us how to reimagine our treatment of virtual memory mappings.

- In both code and proofs.

Page table abstraction: progress

RISC-V page table walk:

- Single recursive function.
- Not even monadic!
- Requires only a “projection” of all page table entries in the heap.
- Used in both the specification and in properties to be proved.

Page table abstraction: future

Wanted: other architectures:

+ Yes!
+ "Just" need to parameterise by a page table entry encoding.
+ Proofs about generic page table walks can be shared.
+ Makes it easier to solves problems with the architecture split.
  - But doesn't solve them directly.
+ Incremental: x86, then ARM.

Page table abstraction: future

How this helps with the architecture split:

+ Narrower and clearer interface between arch-specific and generic proofs.
+ Fewer arch-specific proofs.
- Still need to recheck all the proofs for each configuration.
+ But a clearer path to truly generic proofs.

The end goal

- + One specification, parameterised by:
  - - Page table structure.
  - - Feature options (virtualisation).
  - - Data refinement level.
- + One proof of abstract invariants.
- + One proof of the first data refinement.
- - We’ll see about the C refinement.

Conclusion

Past decisions:

+ Were right at the time.
- Have different consequences in the present.
+ But we can adapt.

We’ve made it to the point where we can have mundane software engineering concerns in formal verification!