The seL4® Verification Journey:

How Have the Challenges and Opportunities Evolved

June Andronick | Keynote of FMCAD 2022, Trento

A good journey…

…starts with a dream

Photo by mohammed alherz on Unsplash

…delivers achievements

…should offer opportunities to reflect

...and present a path to a bigger journey

Photo by Joshua Earle on Unsplash

Overview

#1
Make a dream come true:
verified, performant kernel

#2
Deliver it to the world:
true trustworthiness for critical software

#3
Keep it live:
for today and tomorrow

Photo by Xan Griffin on Unsplash

Overview

#1
Make a dream come true:
verified, performant kernel

Overview

#1
Make a dream come true:
verified, performant kernel

Opportunities:

- achieve a decades-long dream
- demonstrate FM on real systems

Photo by Xan Griffin on Unsplash

The seL4 story started as…

a research project wanting to solve a problem that was both

hard

world-changing

Formally verified microkernel.
At no more than 10% performance degradation.

Gernot Heiser, ~2004

The seL4 story started as...

a research project wanting to solve a problem that was both

hard

world-changing

Formally verified microkernel.
At no more than 10% performance degradation.

Gernot Heiser, ~2004

The seL4 story started as...

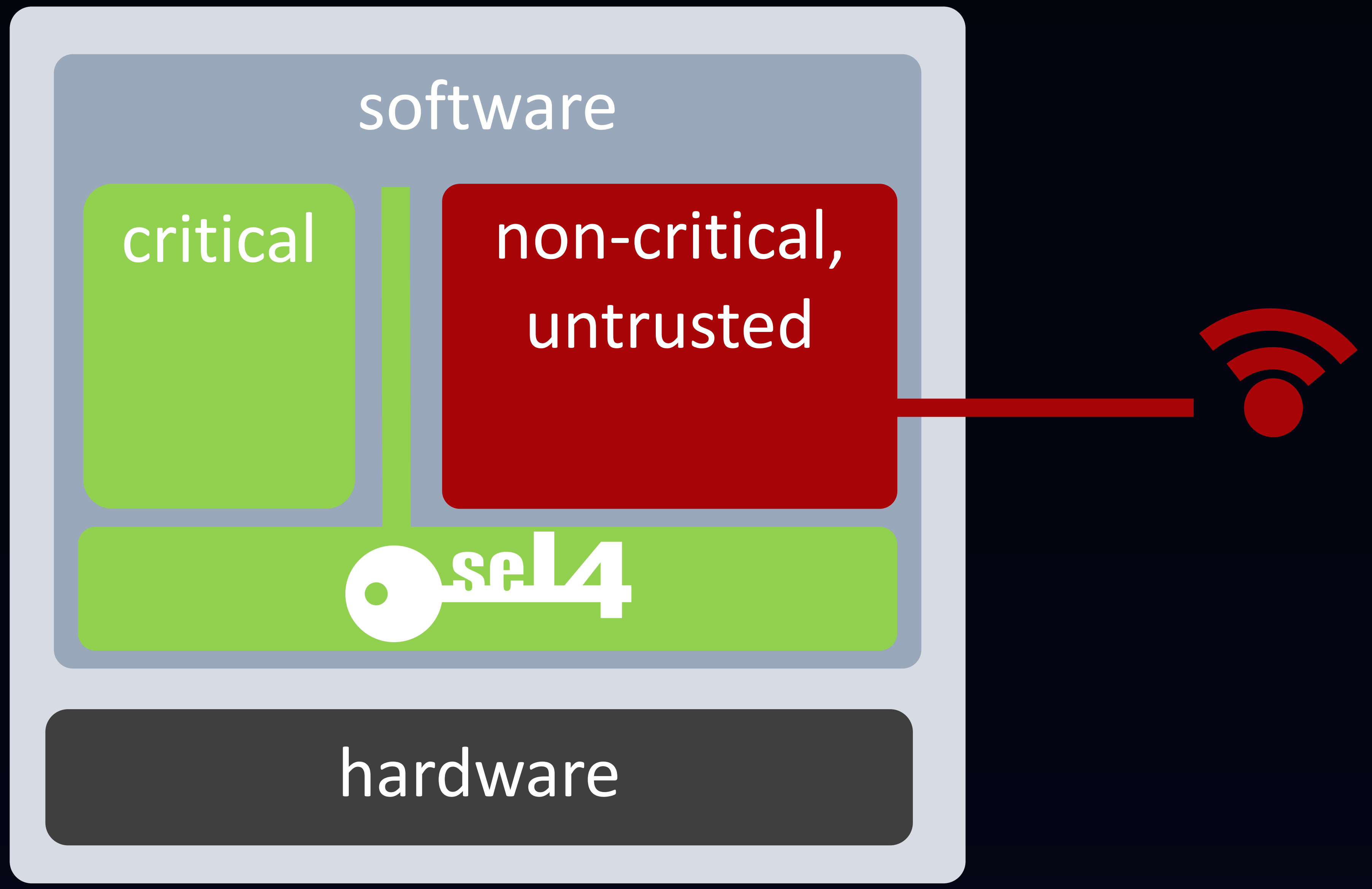
a research project wanting to solve a problem that was both

hard

world-changing

Formally verified microkernel.
At no more than 10% performance degradation.

Gernot Heiser, ~2004



The seL4 story started as…

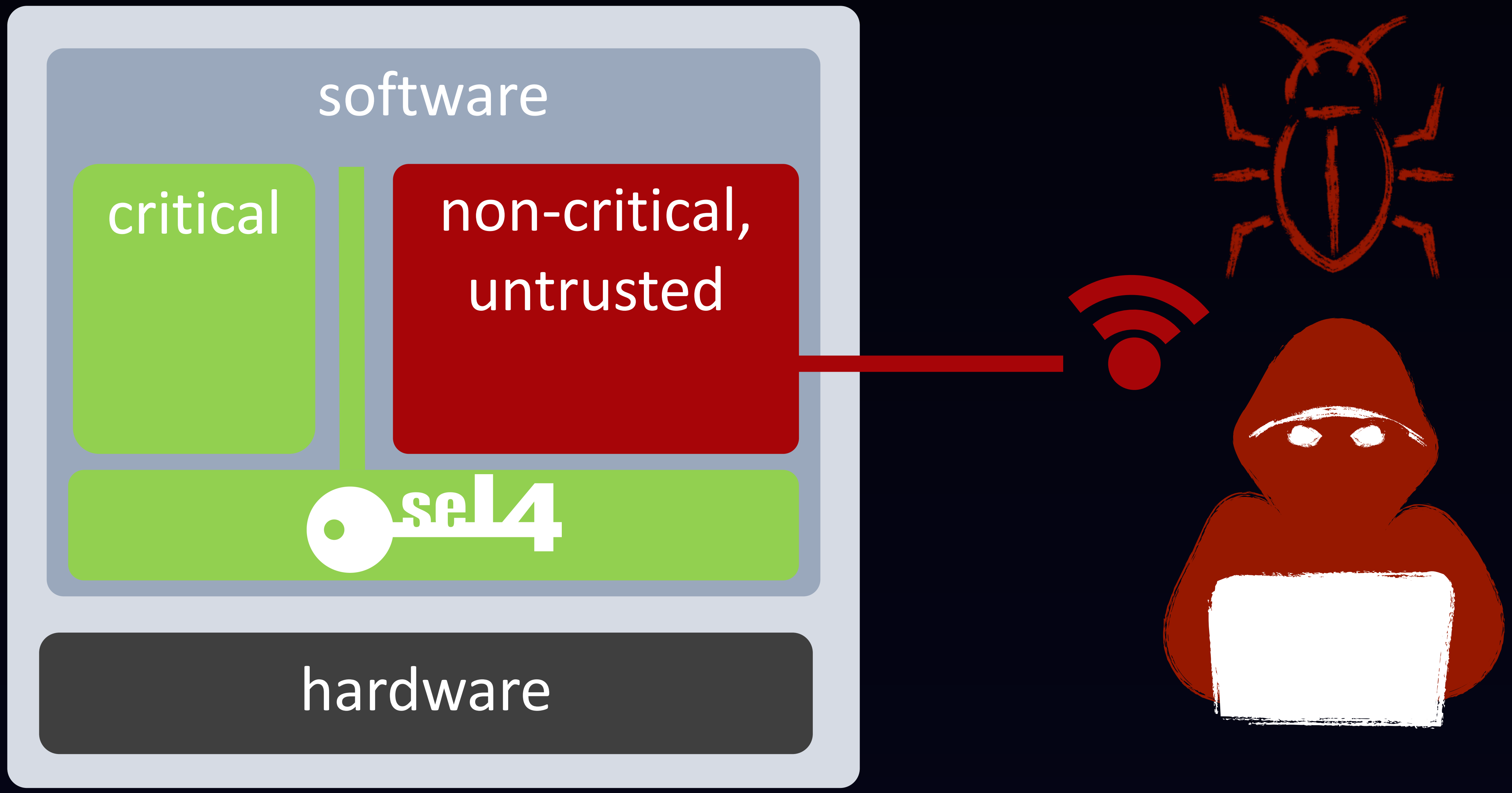
a research project wanting to solve a problem that was both

hard

world-changing

Formally verified microkernel.
At no more than 10% performance degradation.

Gernot Heiser, ~2004

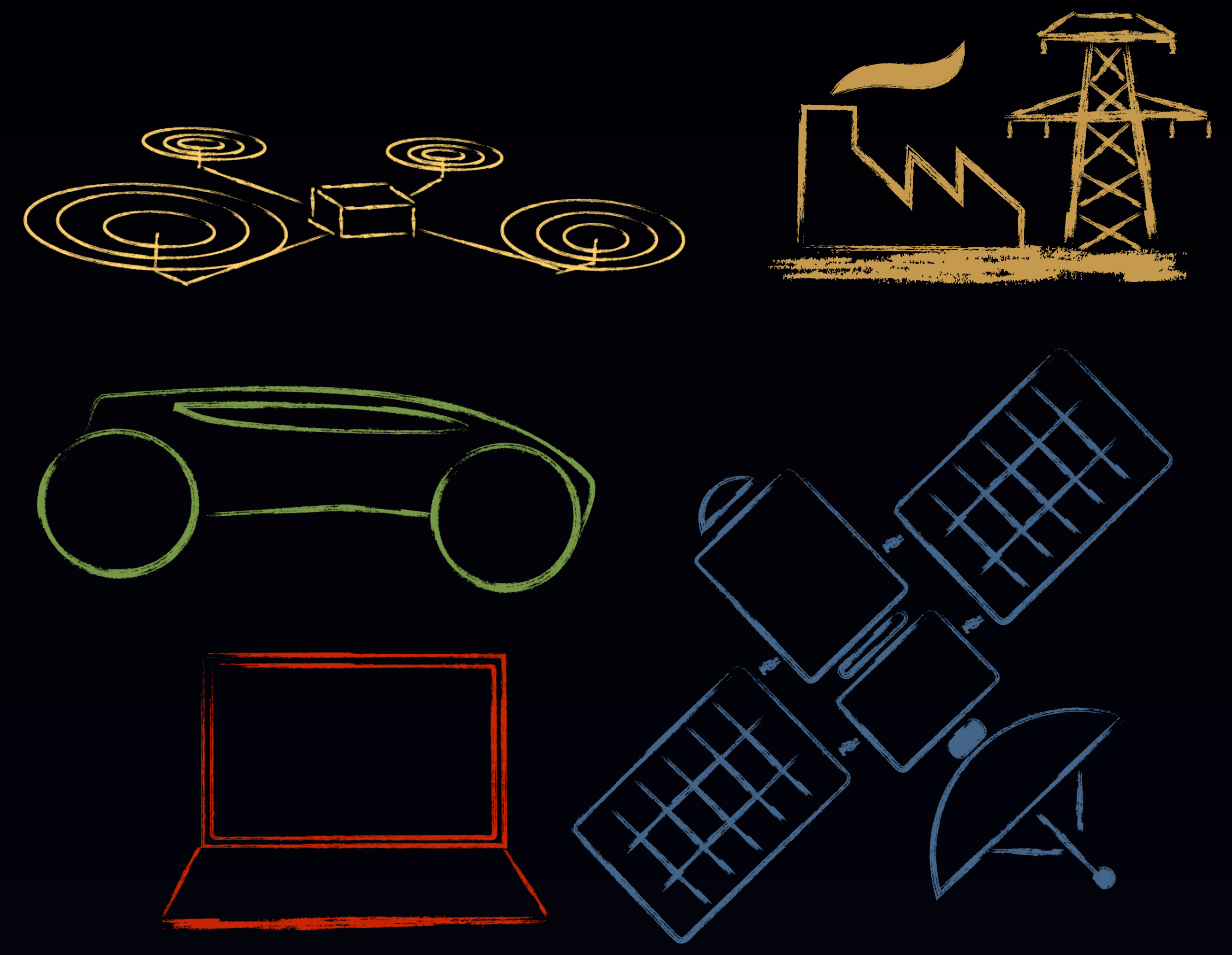


The seL4 story started as...

a research project wanting to solve a problem that was both

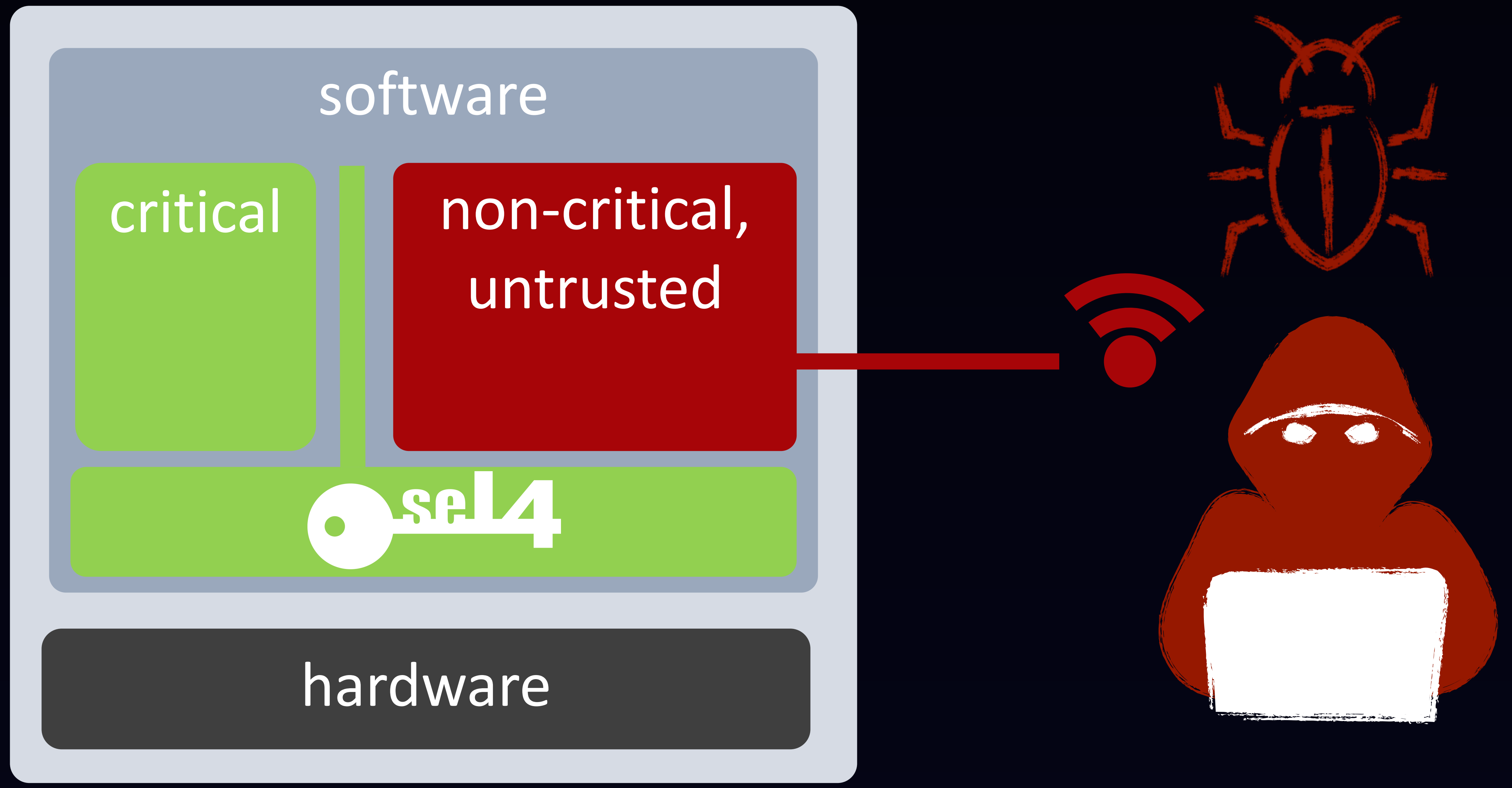
hard

world-changing



Formally verified microkernel.
At no more than 10% performance degradation.

Gernot Heiser, ~2004



The seL4 story started as...

a research project wanting to solve a problem that was both

hard

world-changing

Formally verified microkernel.
At no more than 10% performance degradation.

Gernot Heiser, ~2004

The seL4 story started as…

a research project wanting to solve a problem that was both

hard

world-changing

Formally verified microkernel.
At no more than 10% performance degradation.

Gernot Heiser, ~2004

Done.

Gerwin Klein & al, 2009

The seL4 story started as...

a research project wanting to solve a problem that was both

hard

world-changing

Formally verified microkernel.
At no more than 10% performance degradation.
Gernot Heiser, ~2004

Done.
Gerwin Klein & al, 2009

And more. And more.
Gerwin Klein & al, 2013

The seL4 journey

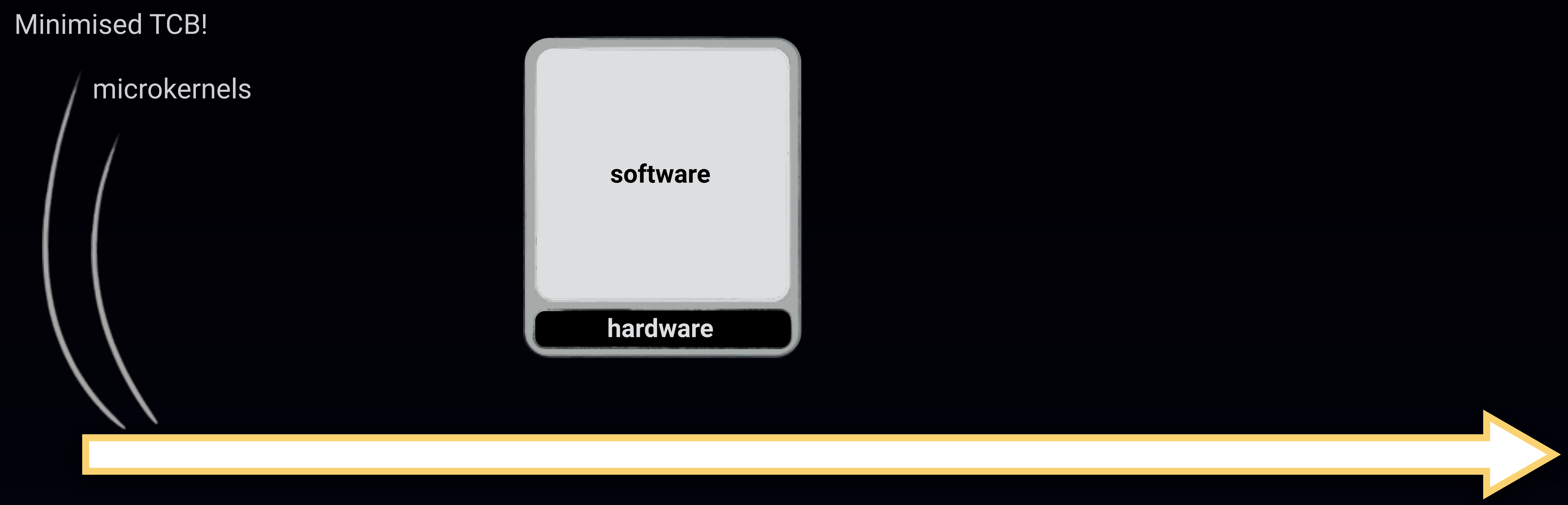
Minimised TCB!

The seL4 journey

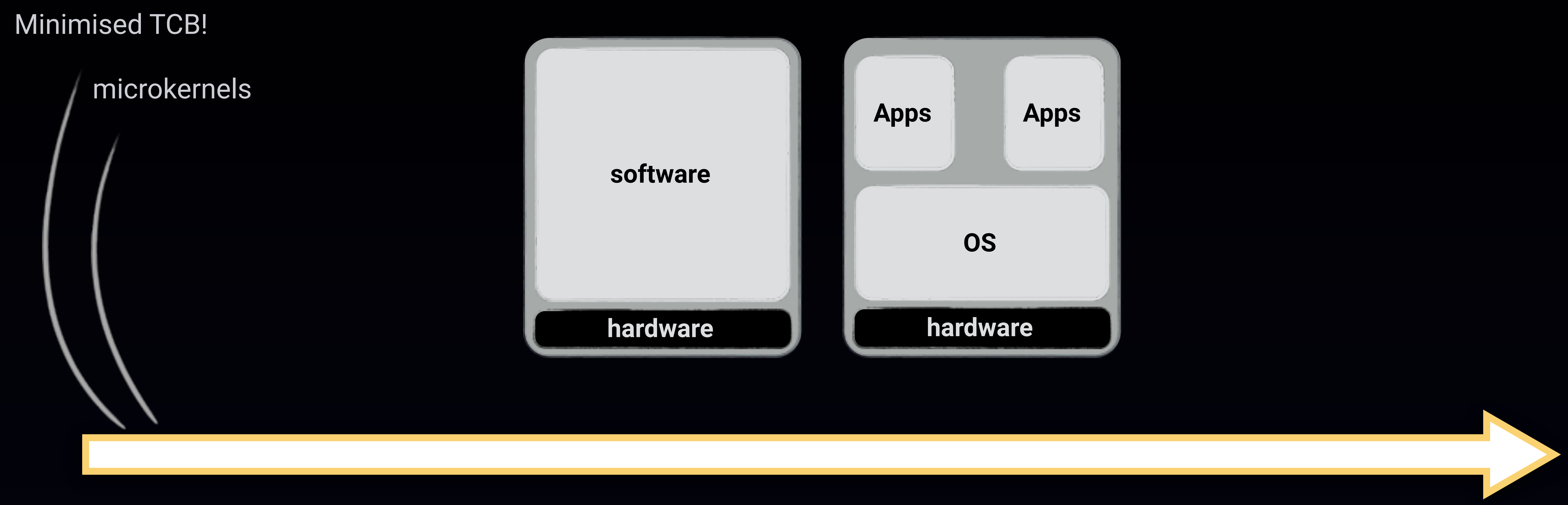
Minimised TCB!

microkernels

The seL4 journey



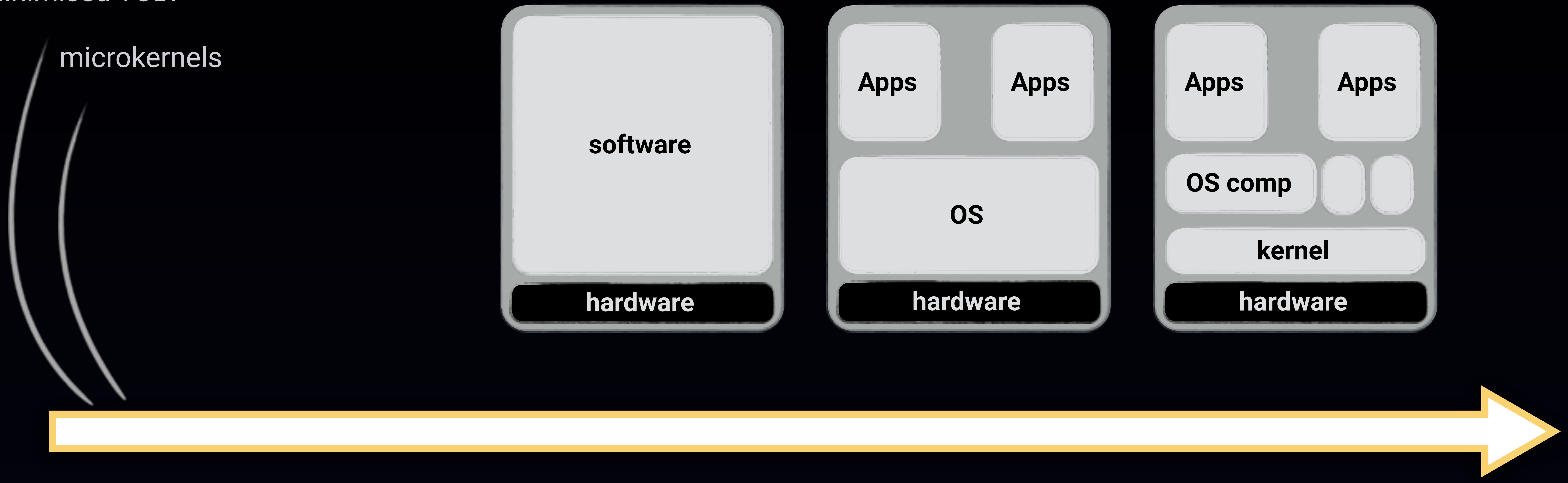
The seL4 journey



The seL4 journey

Minimised TCB!

microkernels

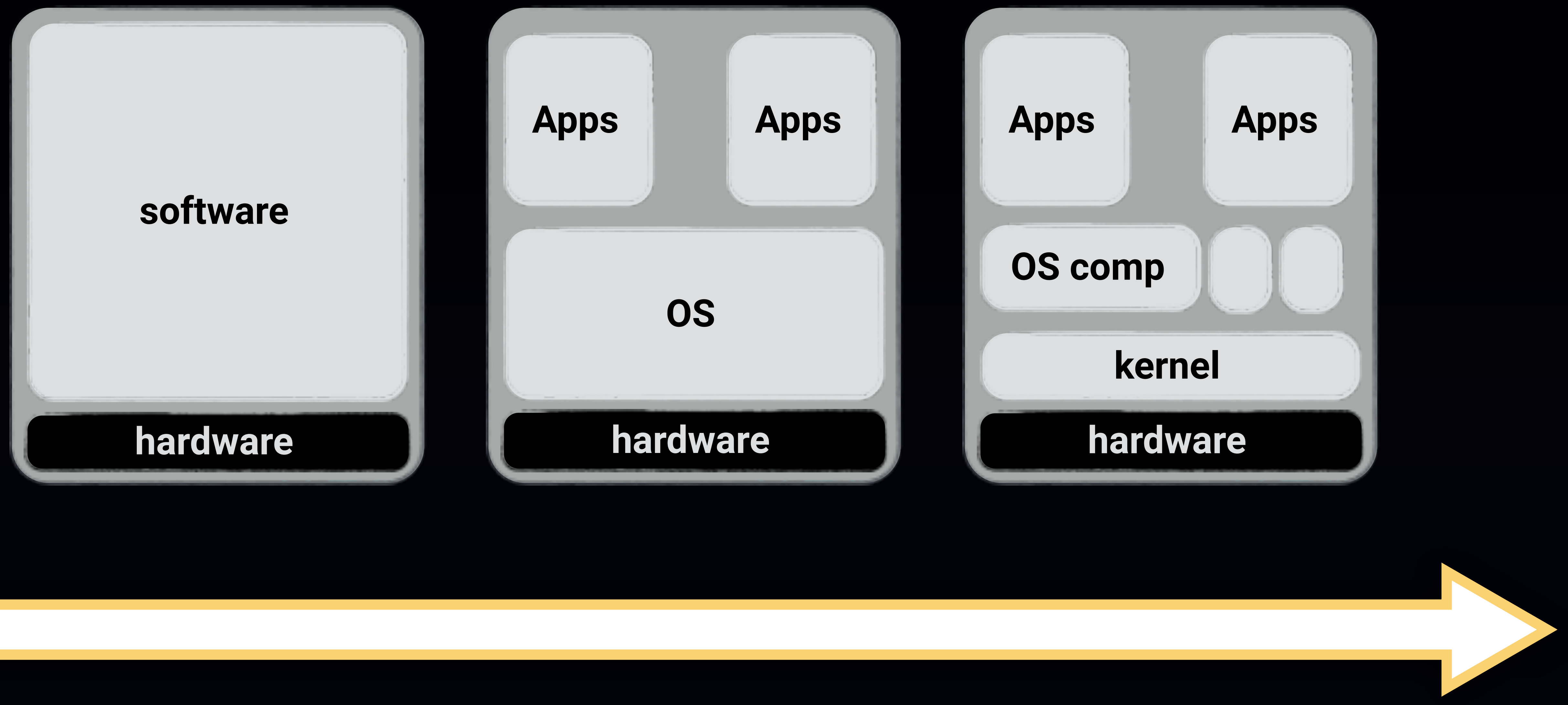


The seL4 journey

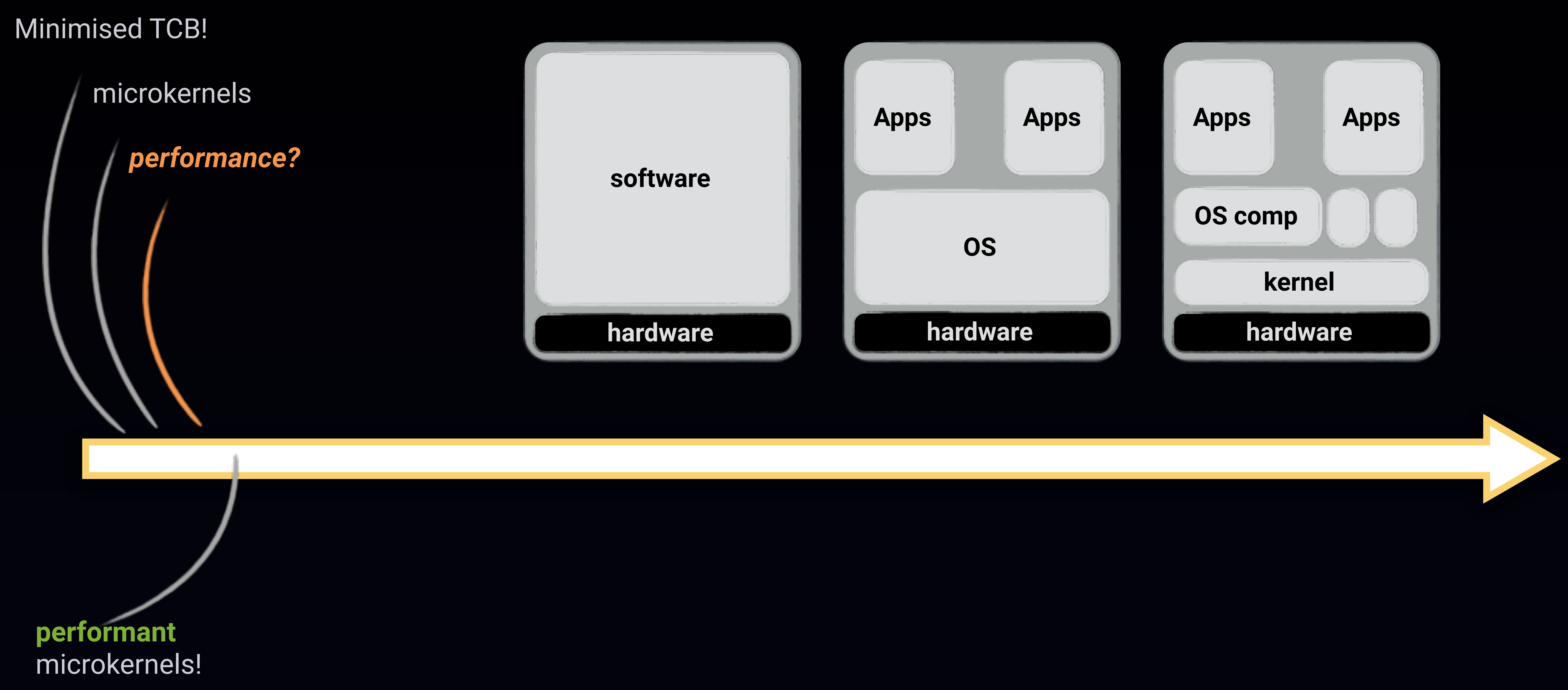
Minimised TCB!

microkernels

performance?



The seL4 journey



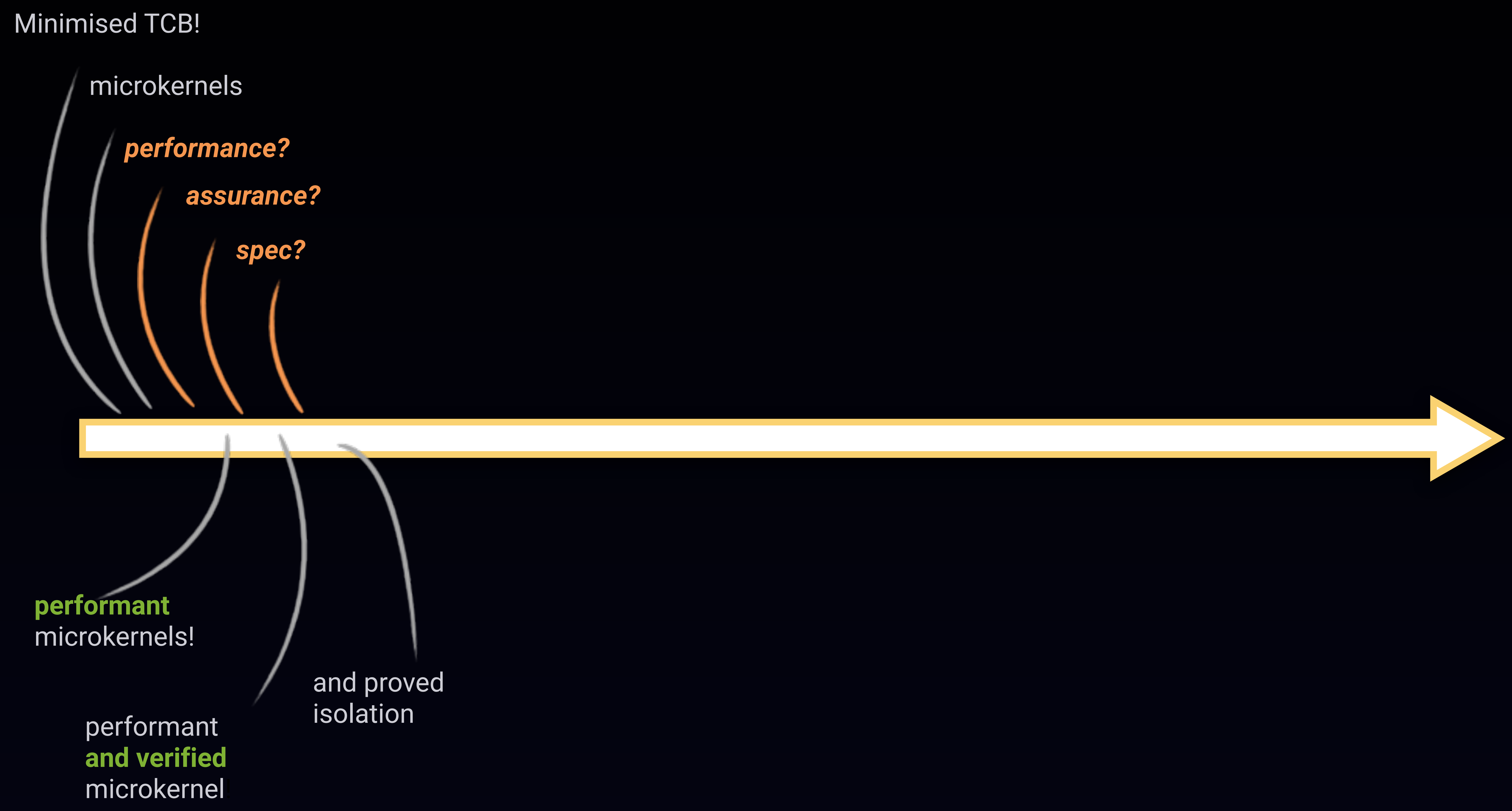
The seL4 journey

The seL4 journey

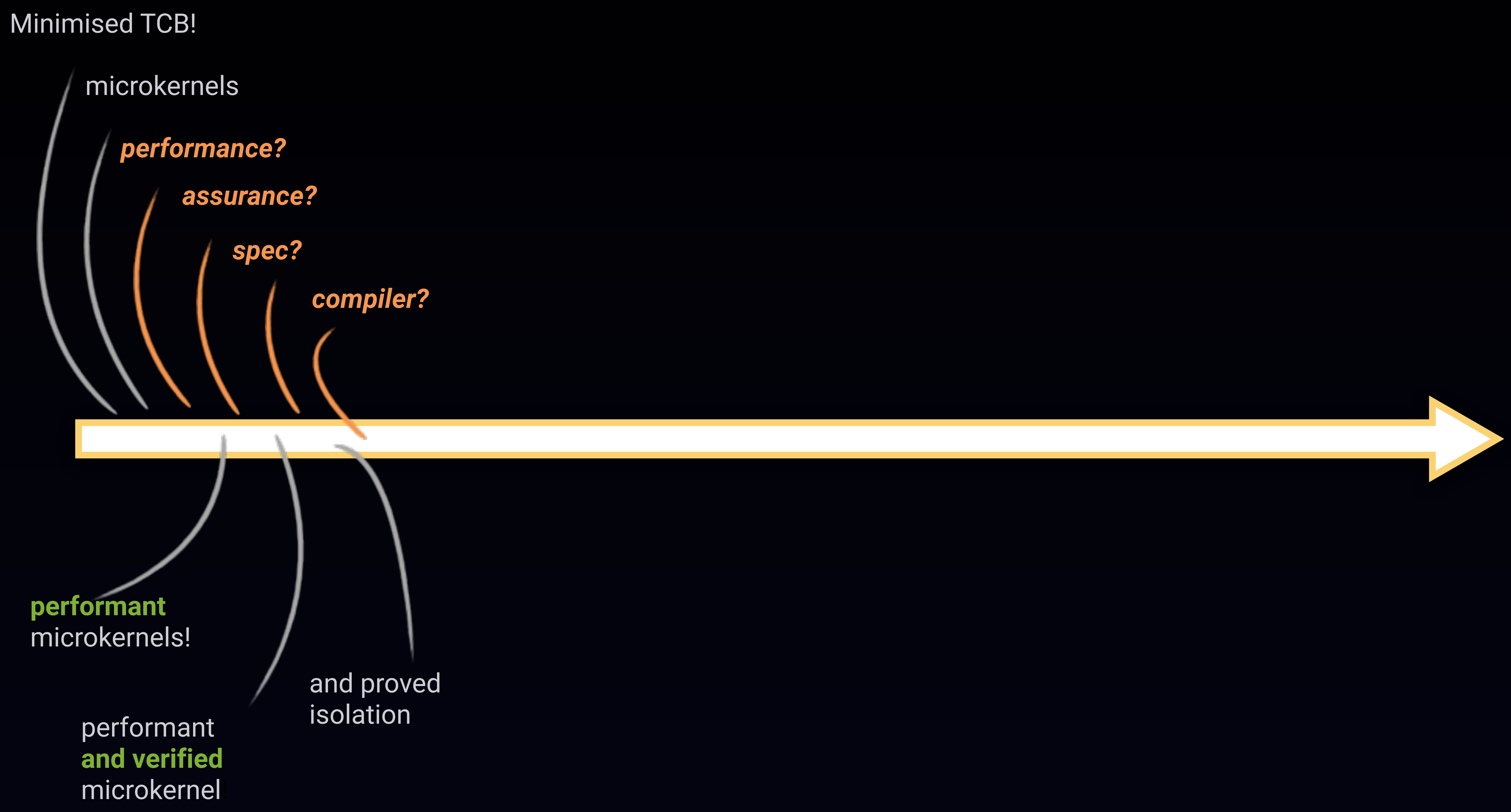
The seL4 journey

The seL4 journey

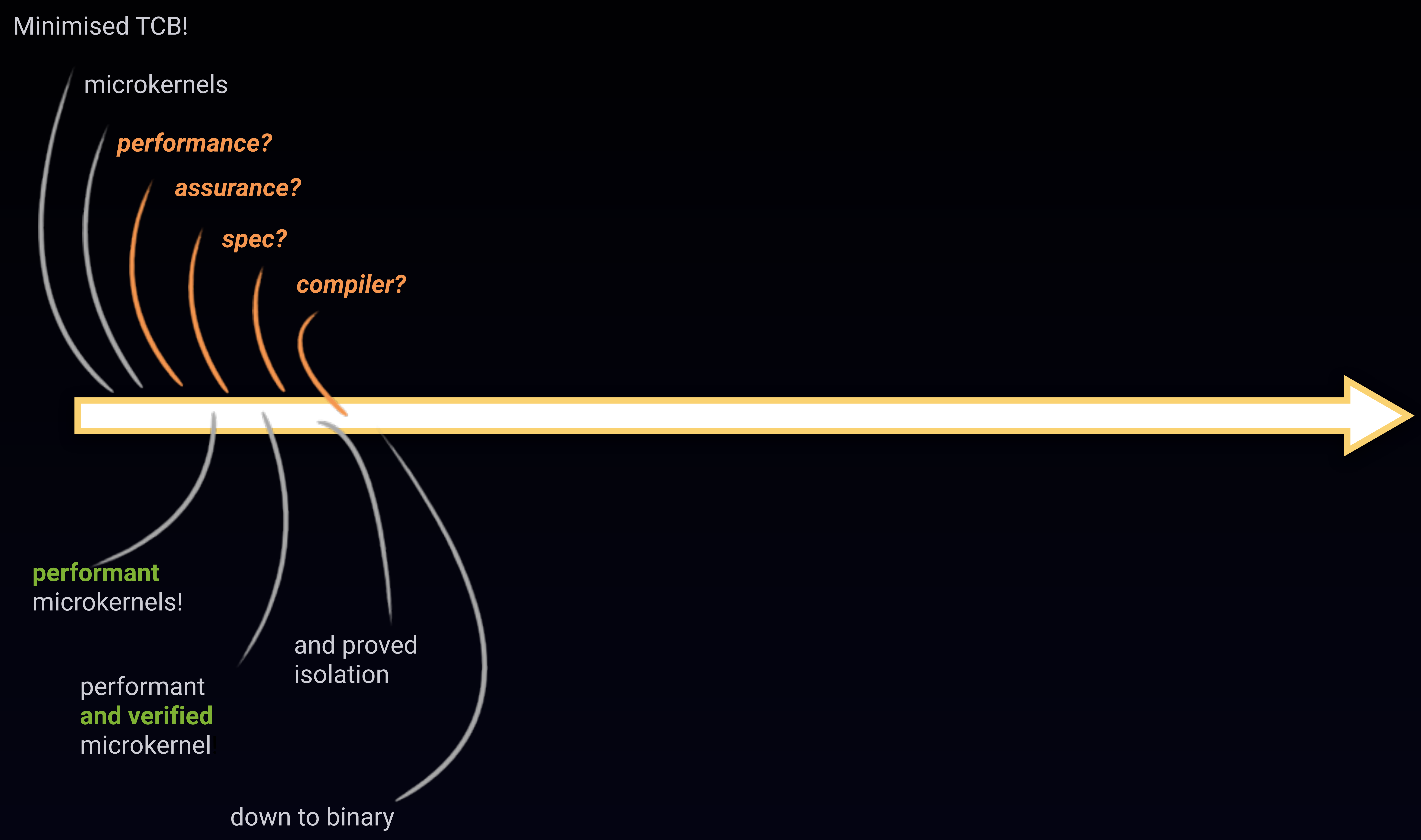
The seL4 journey



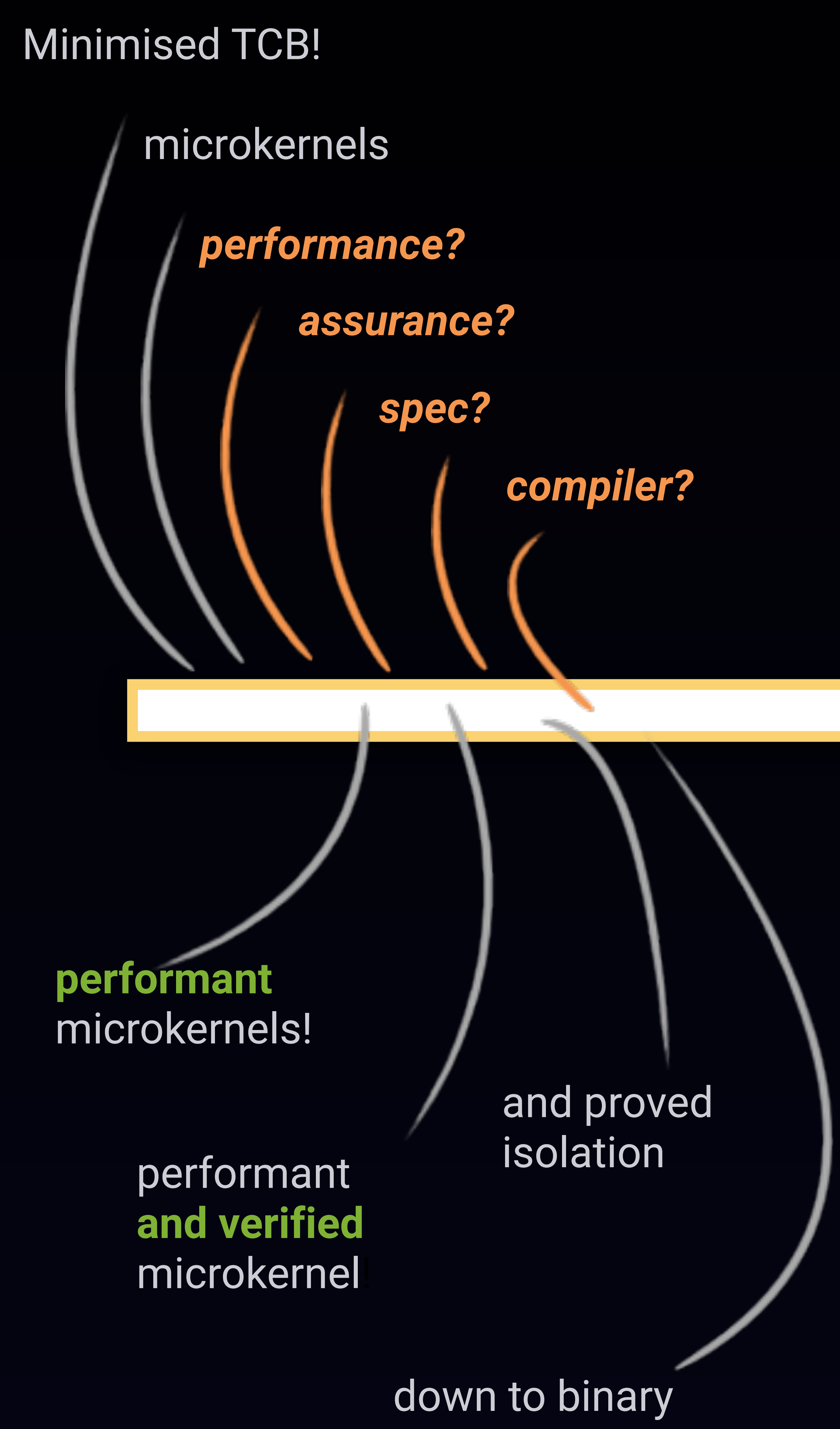
The seL4 journey



The seL4 journey



The seL4 journey



Challenges:

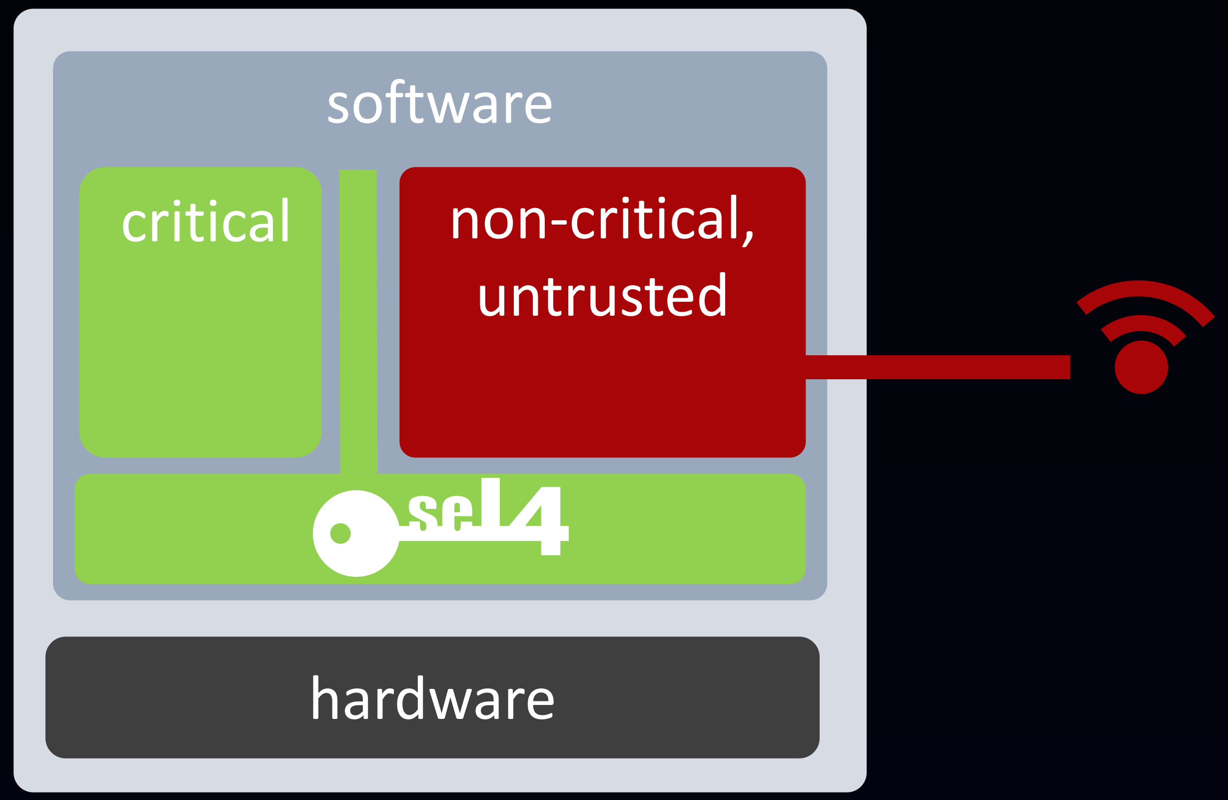
- Scale
- Thoroughness
- Performance

- Make formal verification scale to 10,000 lines of low-level code
- with proof frameworks supporting the verification of functional correctness, security properties and binary correctness
- while maintaining performance

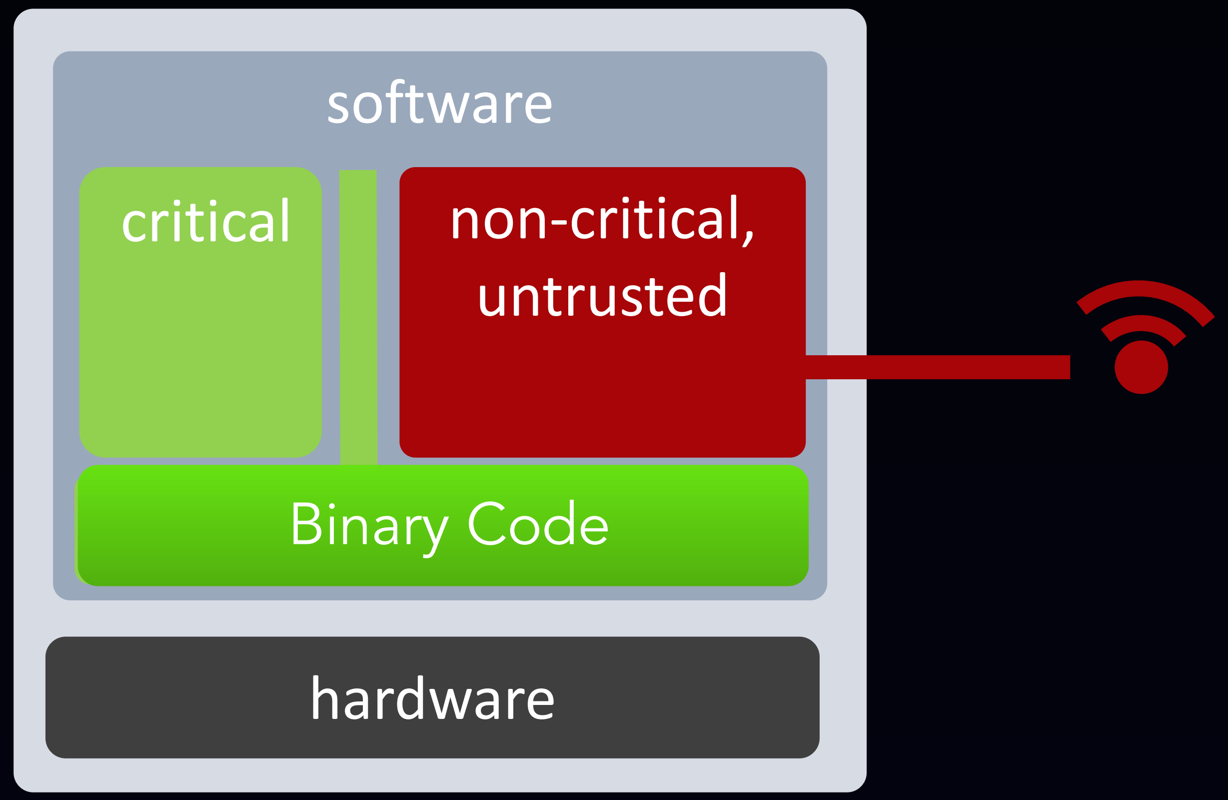
Solutions:

- Combination of foundational techniques
- Targeting machine-checked proof
- Working hand in hand with systems people

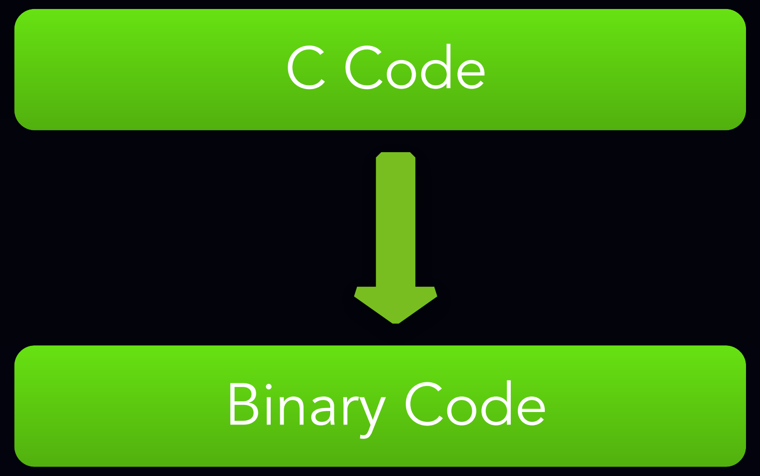
seL4 proofs' foundational techniques



seL4 proofs' foundational techniques



seL4 proofs' foundational techniques



seL4 proofs' foundational techniques

seL4 kernel call graph

C Code

void kernel_call () {…}

~10,000 LOC
>500 functions

seL4 proofs' foundational techniques

C Code → C Code Semantics

void kernel_call () {…}

kernel_call_body ≡ …

C-to-Isabelle Parser: C program → SIMPL program

(mainly) deeply embedded generic imperative language in Isabelle

seL4 proofs' foundational techniques

abstract *functional* specification

kernel_call_A ≡ …

Specification

C Code → C Code Semantics

void kernel_call () {…}

kernel_call_body ≡ …

seL4 proofs' foundational techniques

abstract *functional* specification

kernel_call_A ≡ …

Specification

state monad

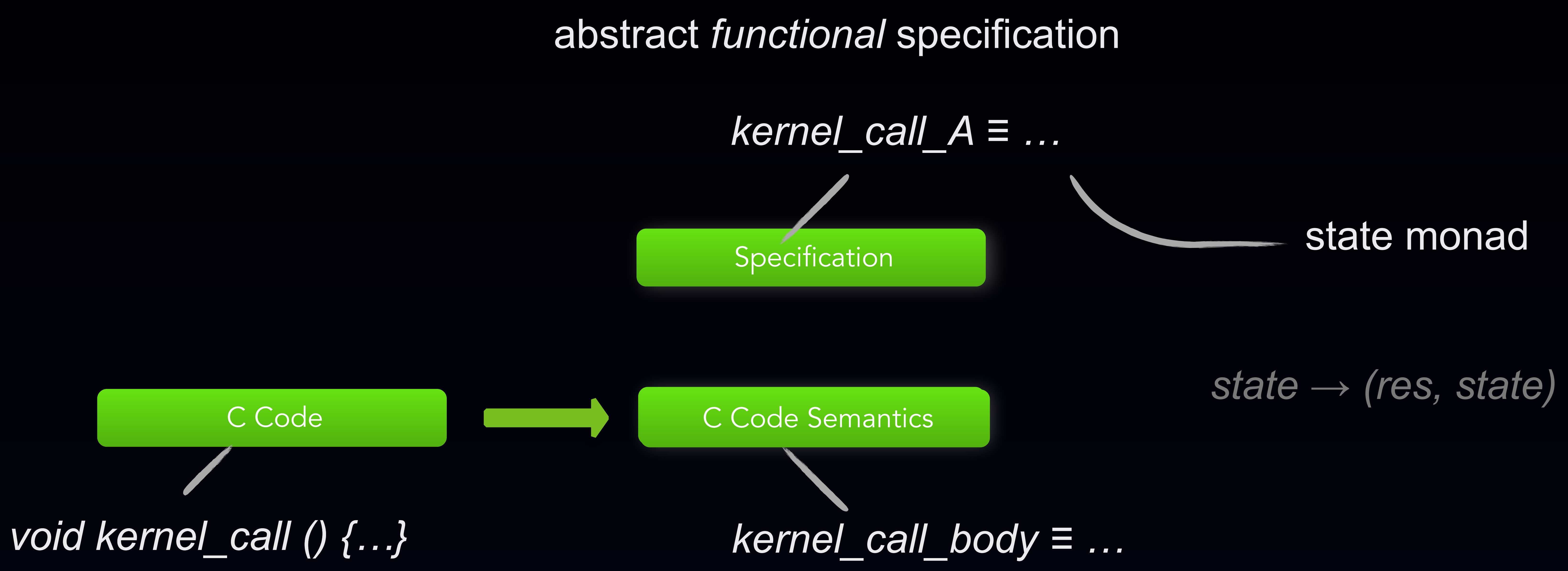
C Code

C Code Semantics

void kernel_call () {…}

kernel_call_body ≡ …

seL4 proofs' foundational techniques



seL4 proofs' foundational techniques

abstract *functional* specification

kernel_call_A ≡ …

non-deterministic
state monad

Specification

state → (res, state)

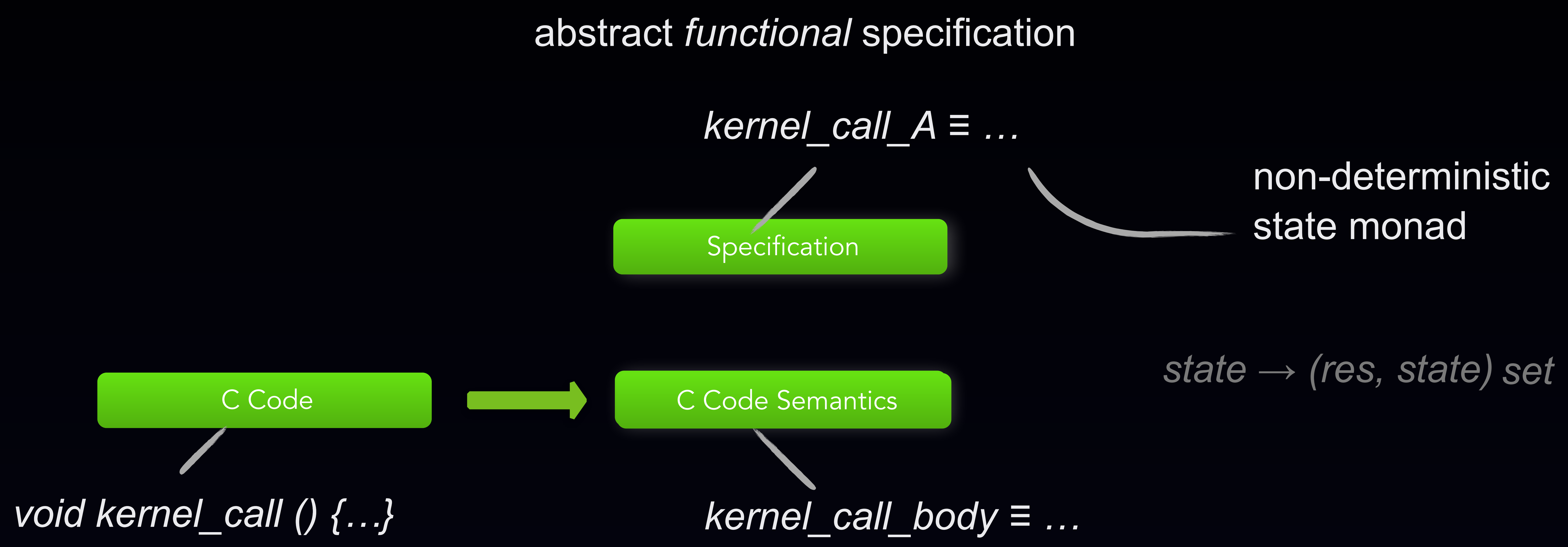
C Code

C Code Semantics

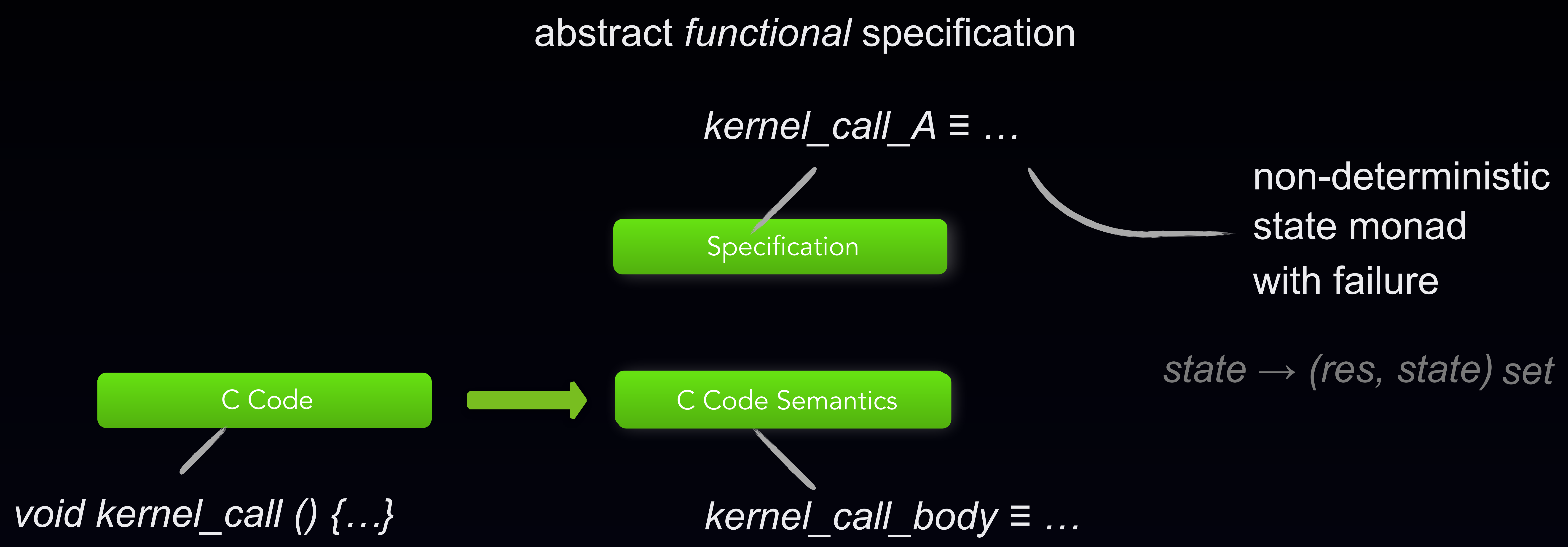
void kernel_call () {…}

kernel_call_body ≡ …

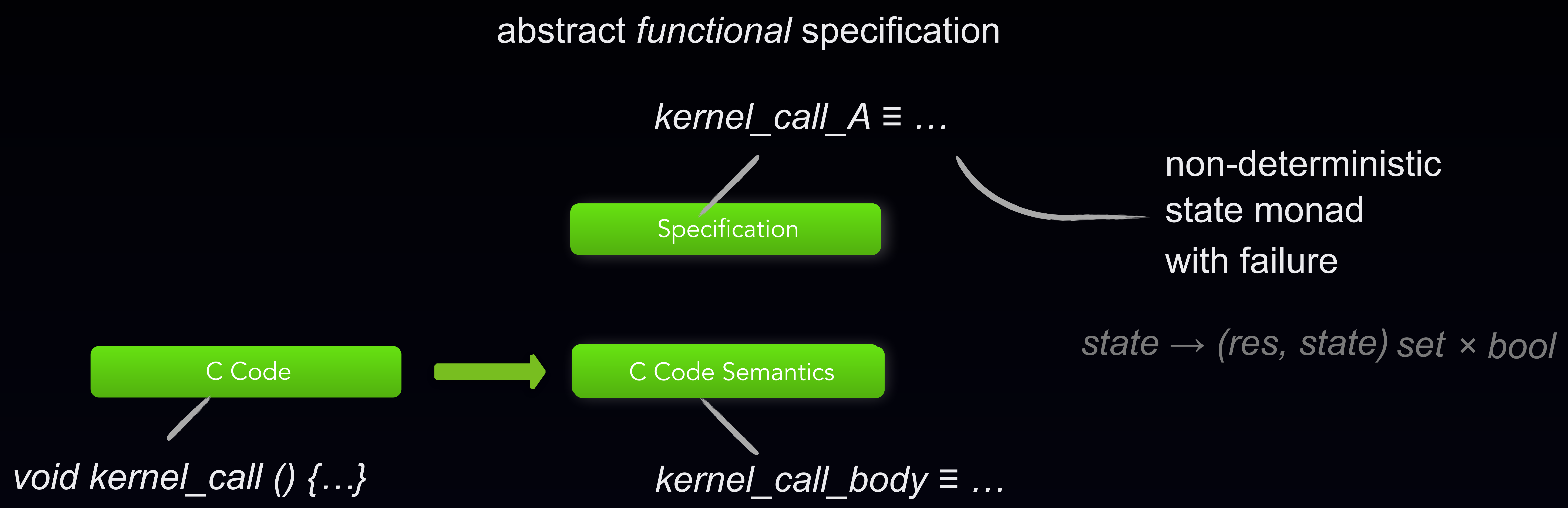
seL4 proofs' foundational techniques



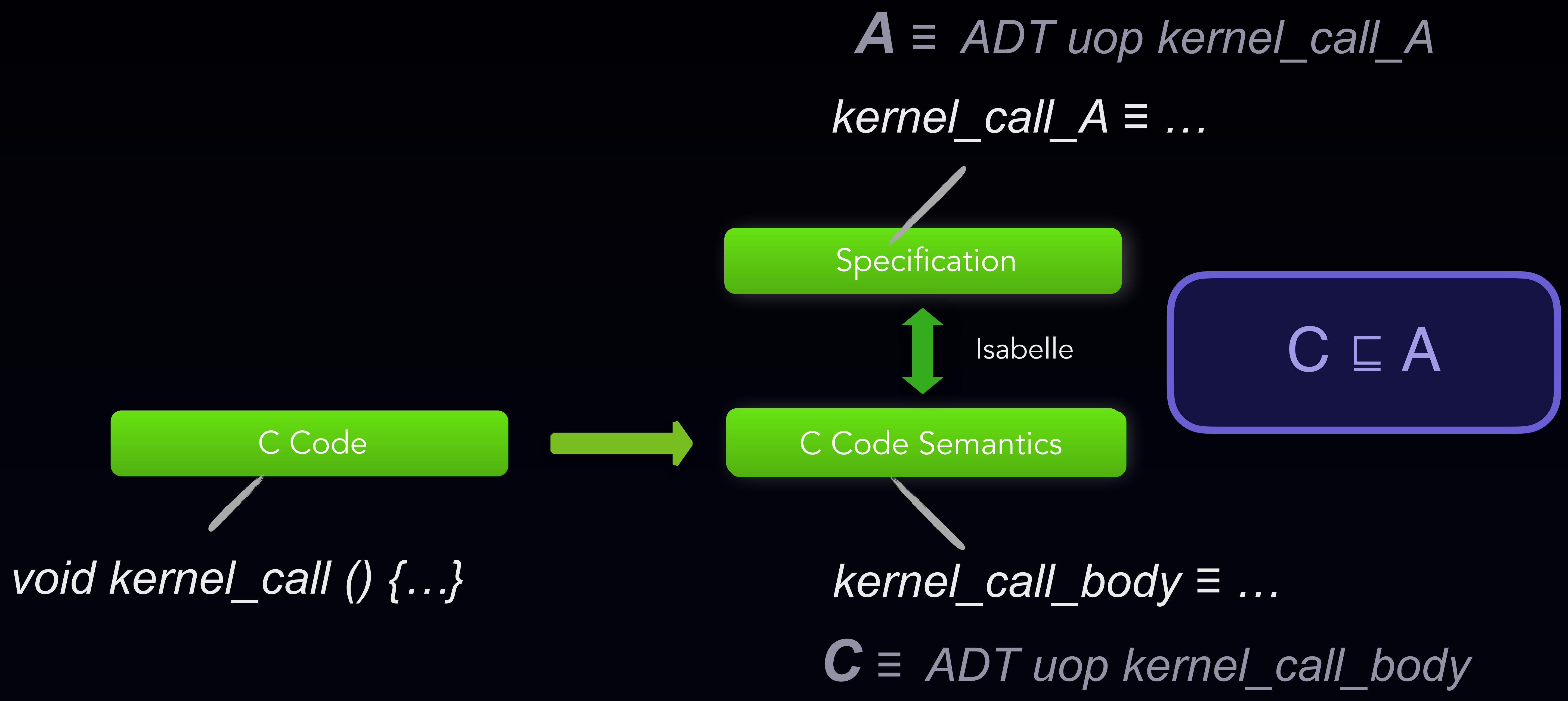
seL4 proofs’ foundational techniques



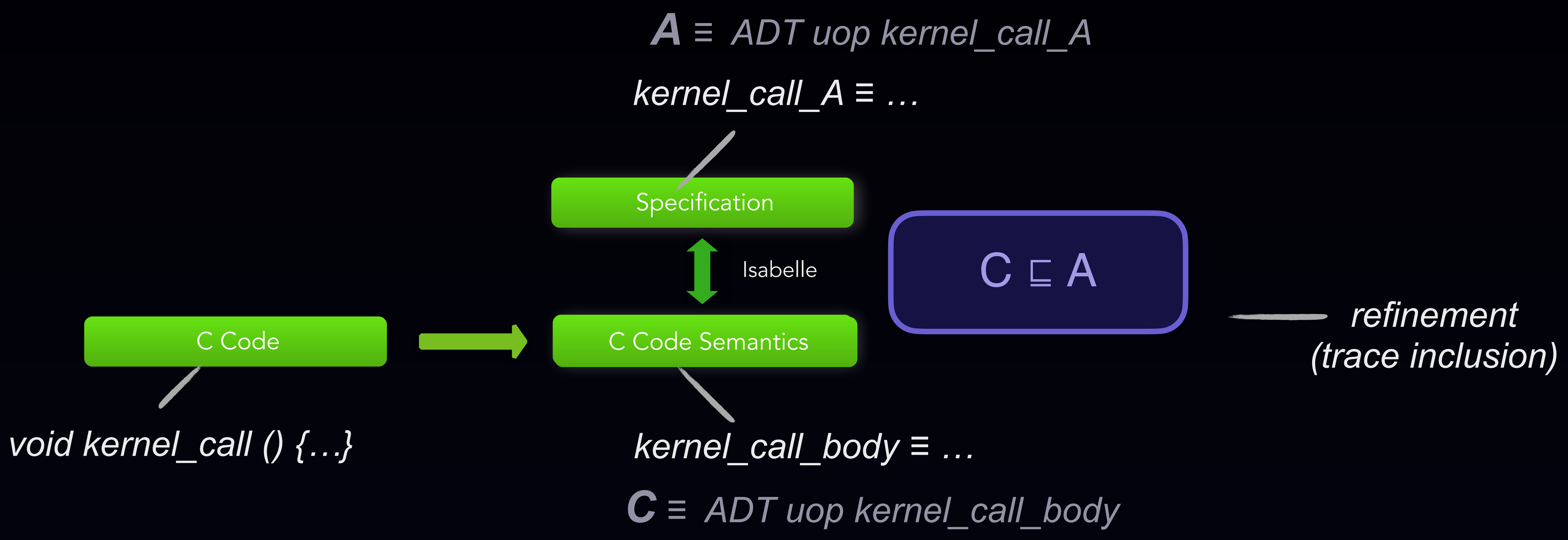
seL4 proofs' foundational techniques



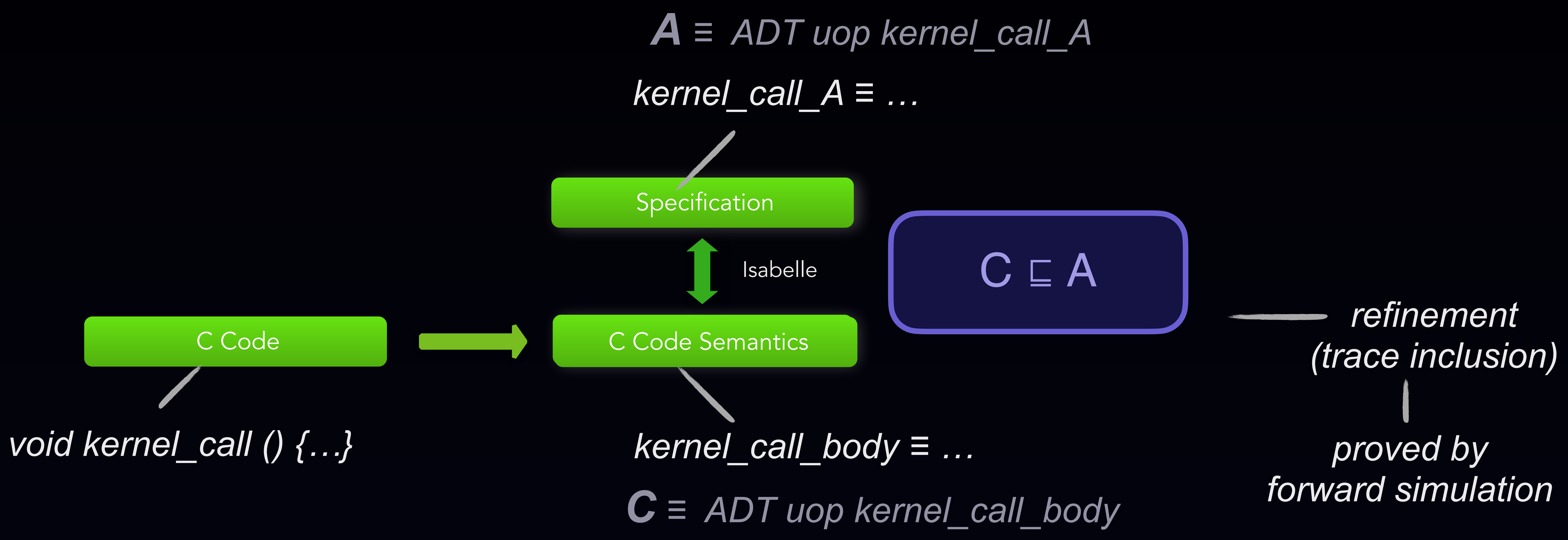
seL4 main theorem #1: Functional correctness



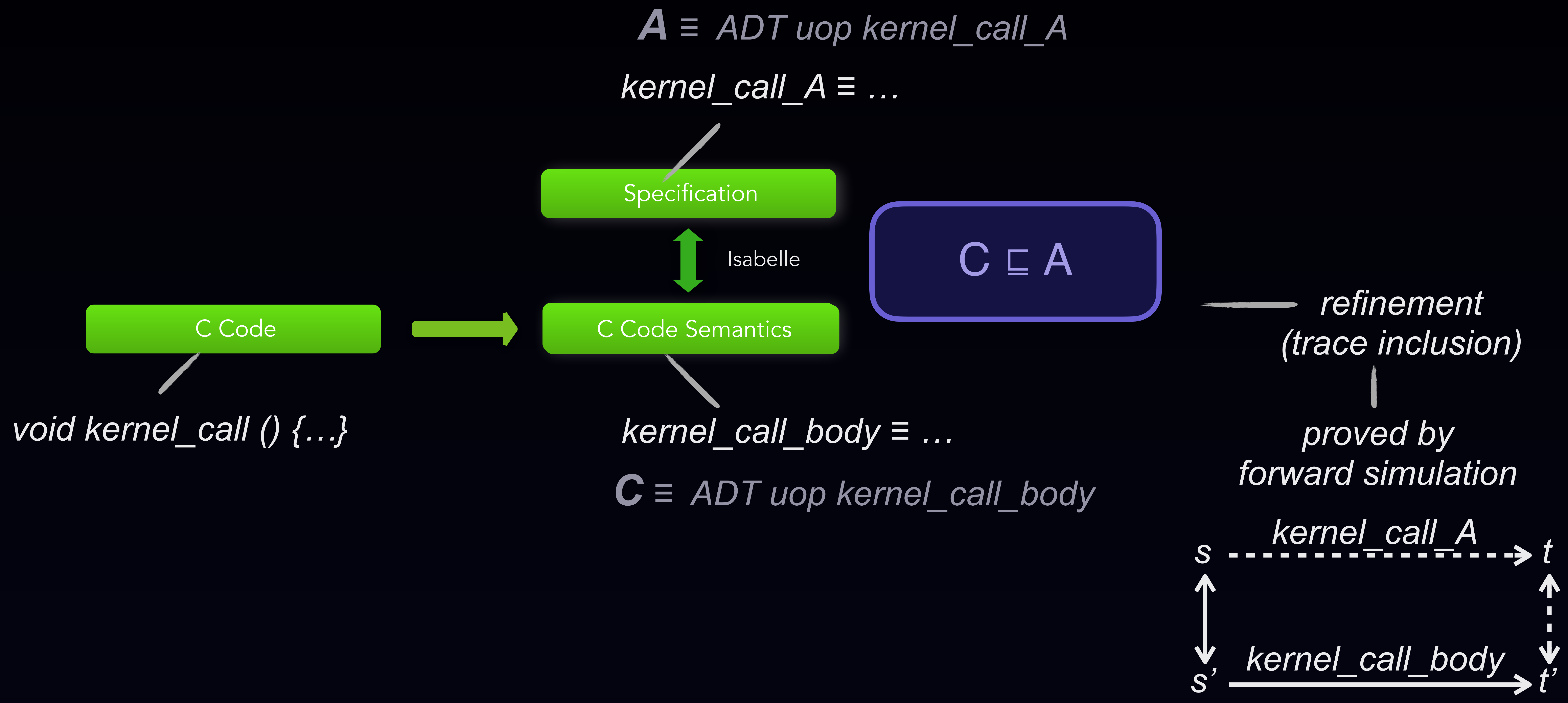
seL4 main theorem #1: Functional correctness



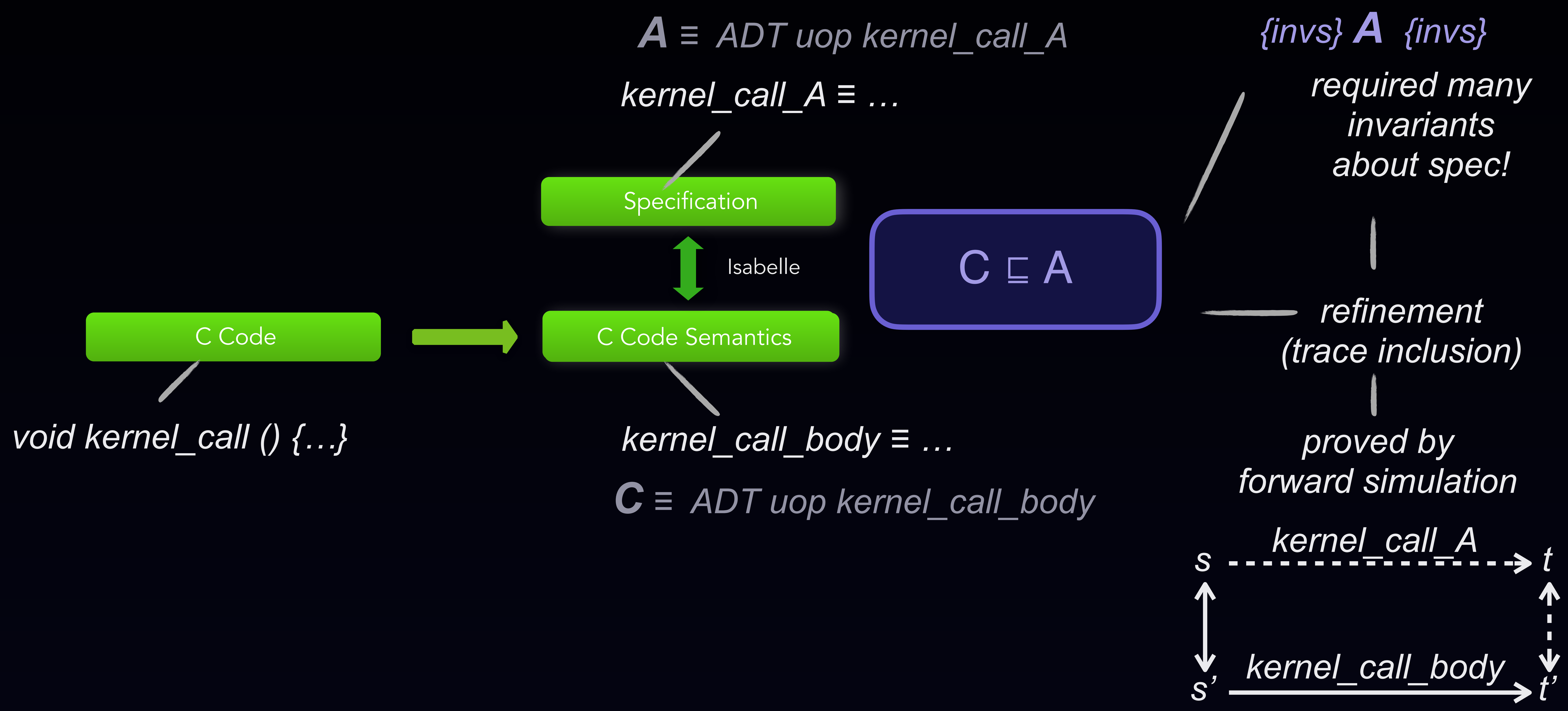
seL4 main theorem #1: Functional correctness



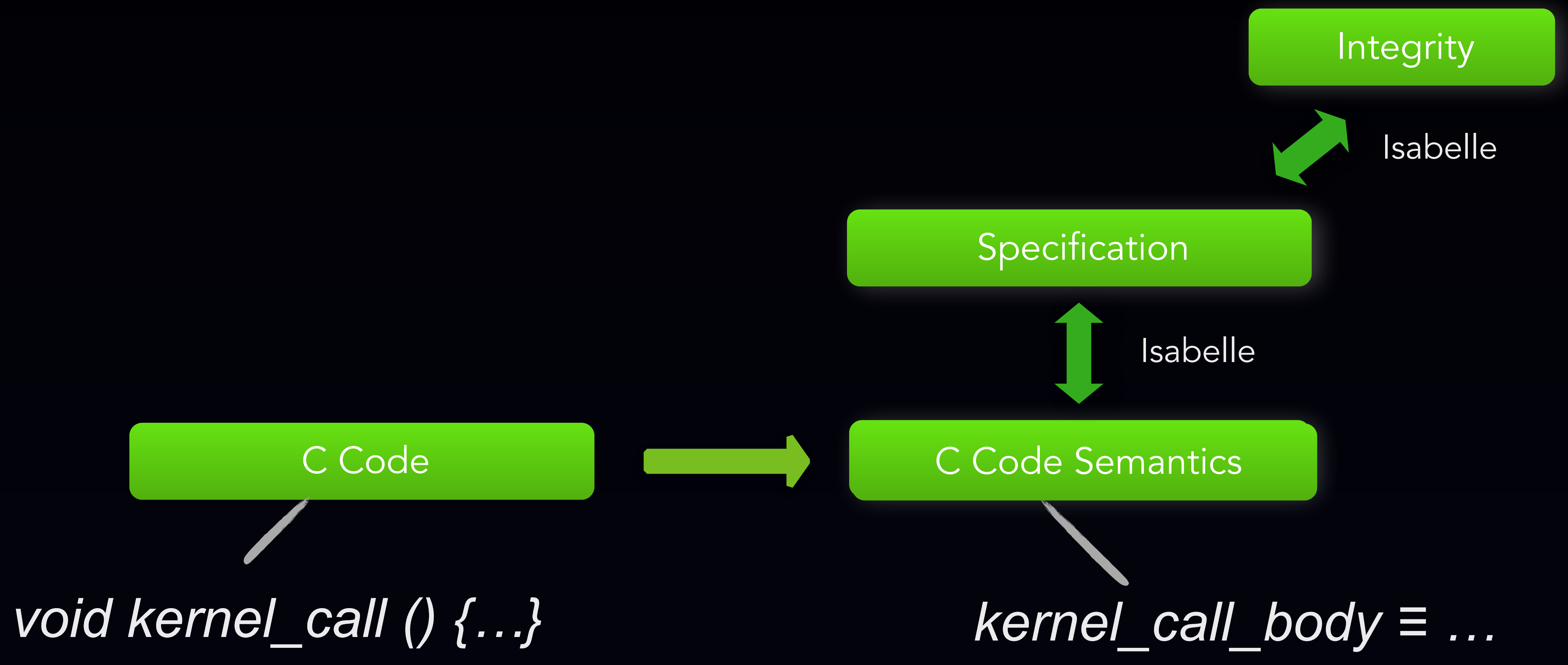
seL4 main theorem #1: Functional correctness



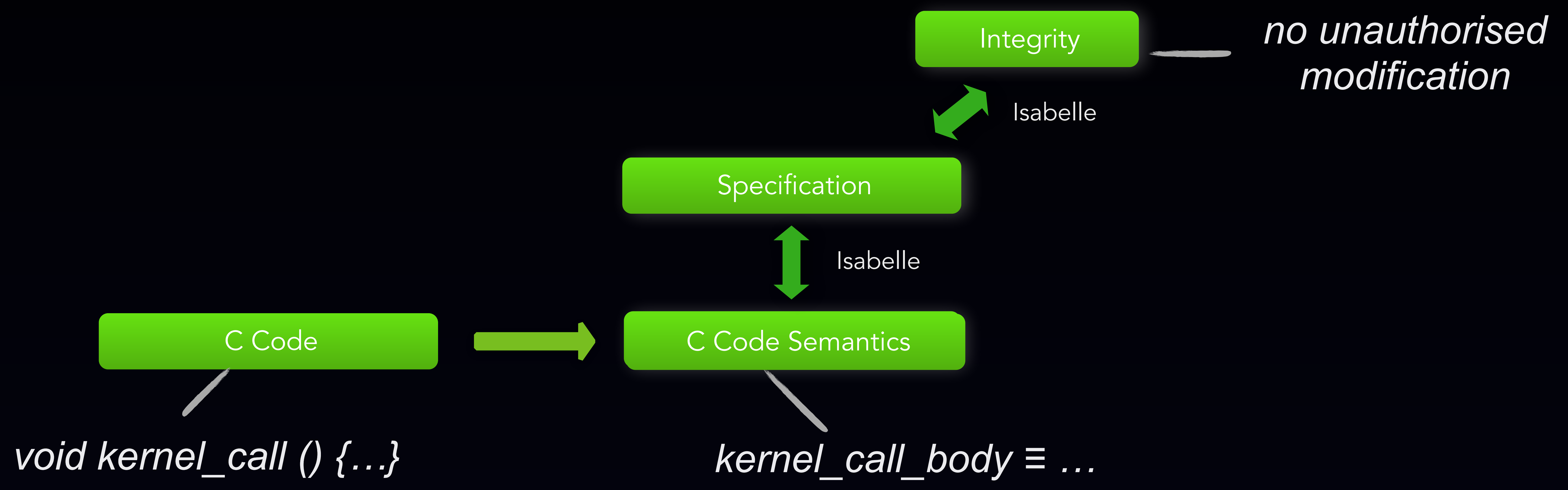
seL4 main theorem #1: Functional correctness



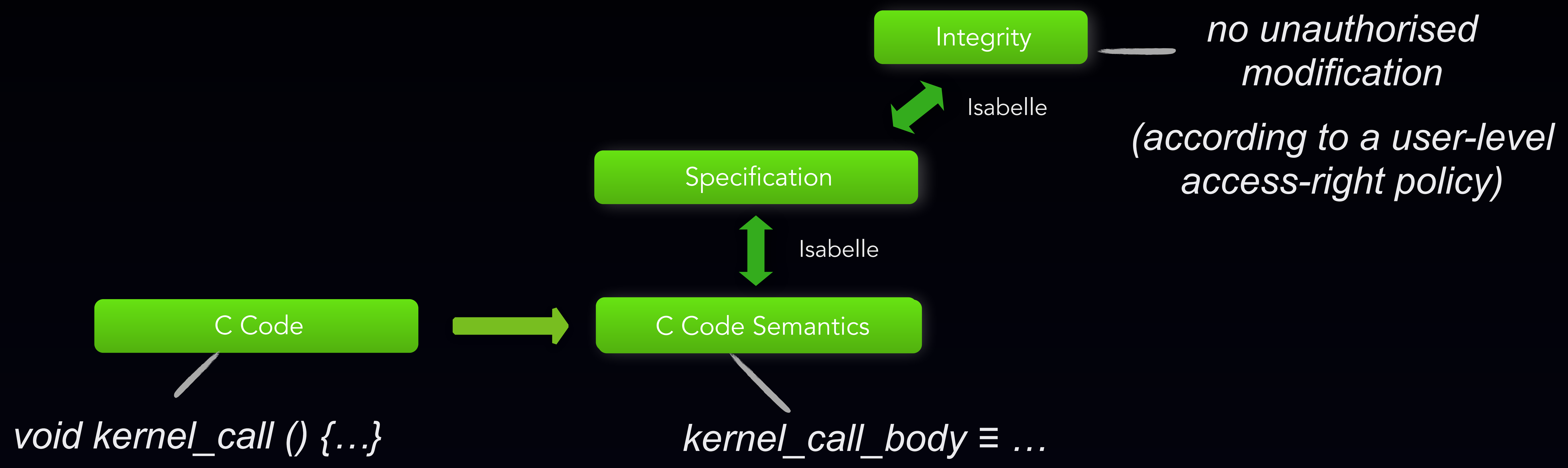
seL4 main theorem #2: integrity



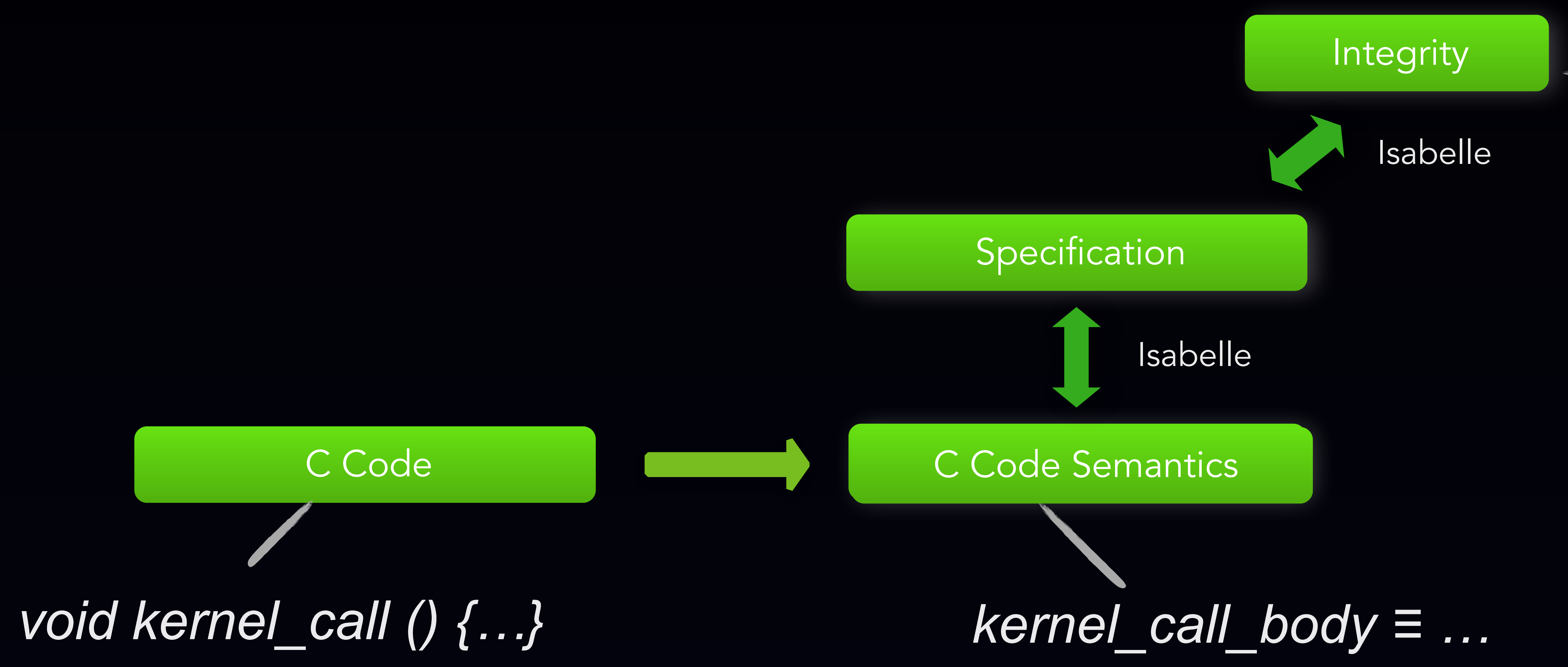
seL4 main theorem #2: integrity



seL4 main theorem #2: integrity



seL4 main theorem #2: integrity

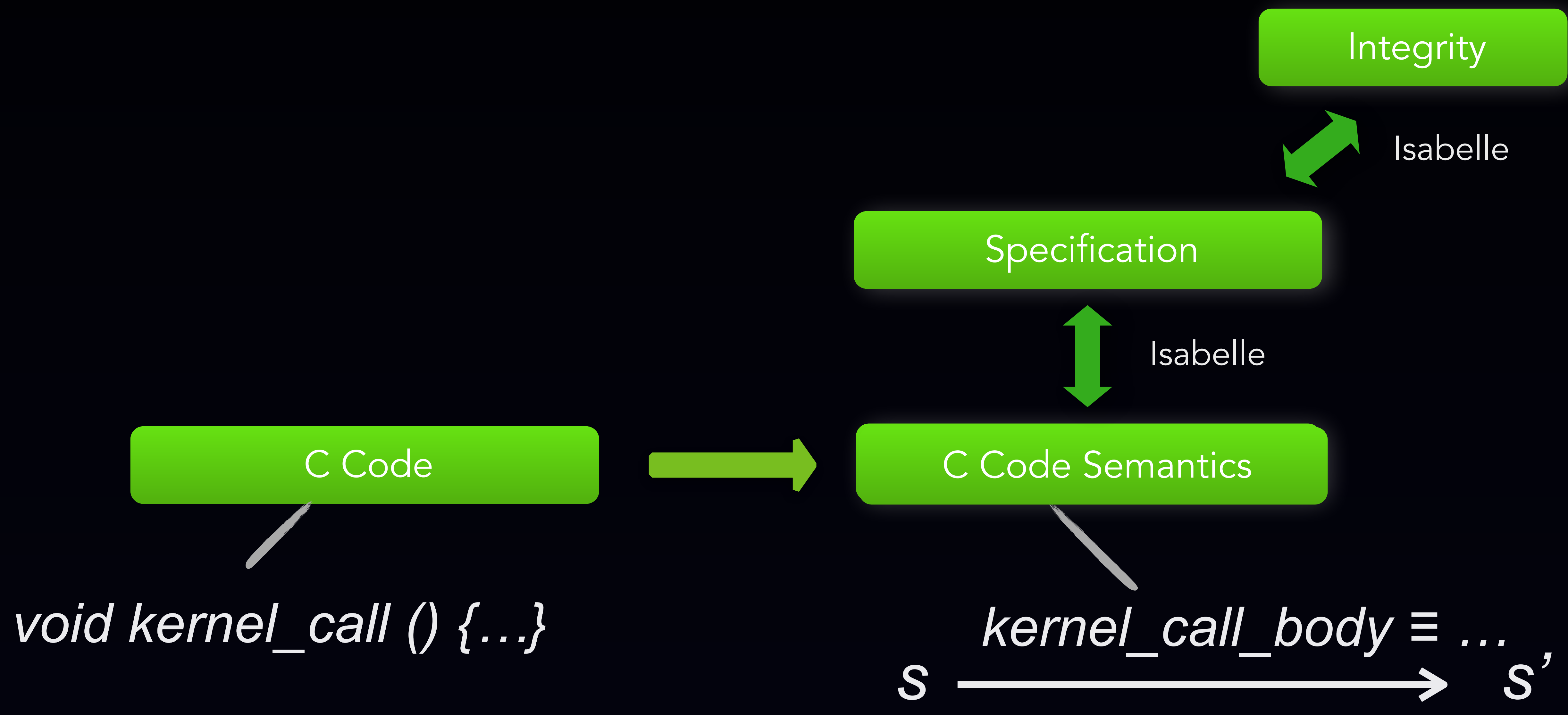


no unauthorised modification

(according to a user-level access-right policy)

After executing the kernel, the only modifications between the final state and the initial state must be authorised by the policy

seL4 main theorem #2: integrity

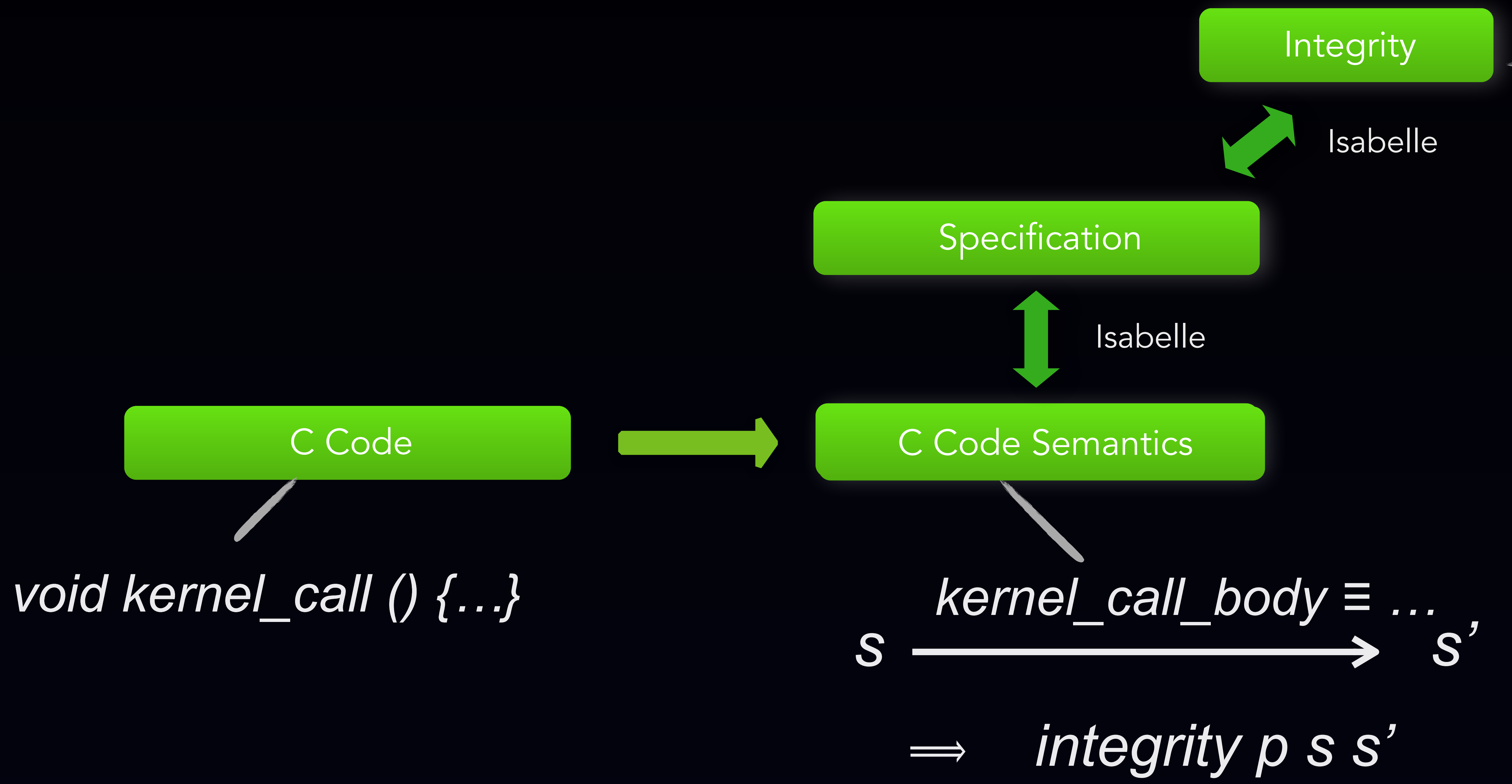


no unauthorised modification

(according to a user-level access-right policy)

After executing the kernel, the only modifications between the final state and the initial state must be authorised by the policy

seL4 main theorem #2: integrity



no unauthorised modification

(according to a user-level access-right policy)

After executing the kernel, the only modifications between the final state and the initial state must be authorised by the policy

seL4 main theorem #2: integrity

{λs. s=s0}
kernel_call_A ()
{λs. integrity p s s0}

Hoare triple

Integrity

Isabelle

Specification

Isabelle

C Code

C Code Semantics

void kernel_call () {…}

$s \xrightarrow{\textit{kernel\_call\_body} \equiv \dots} s'$

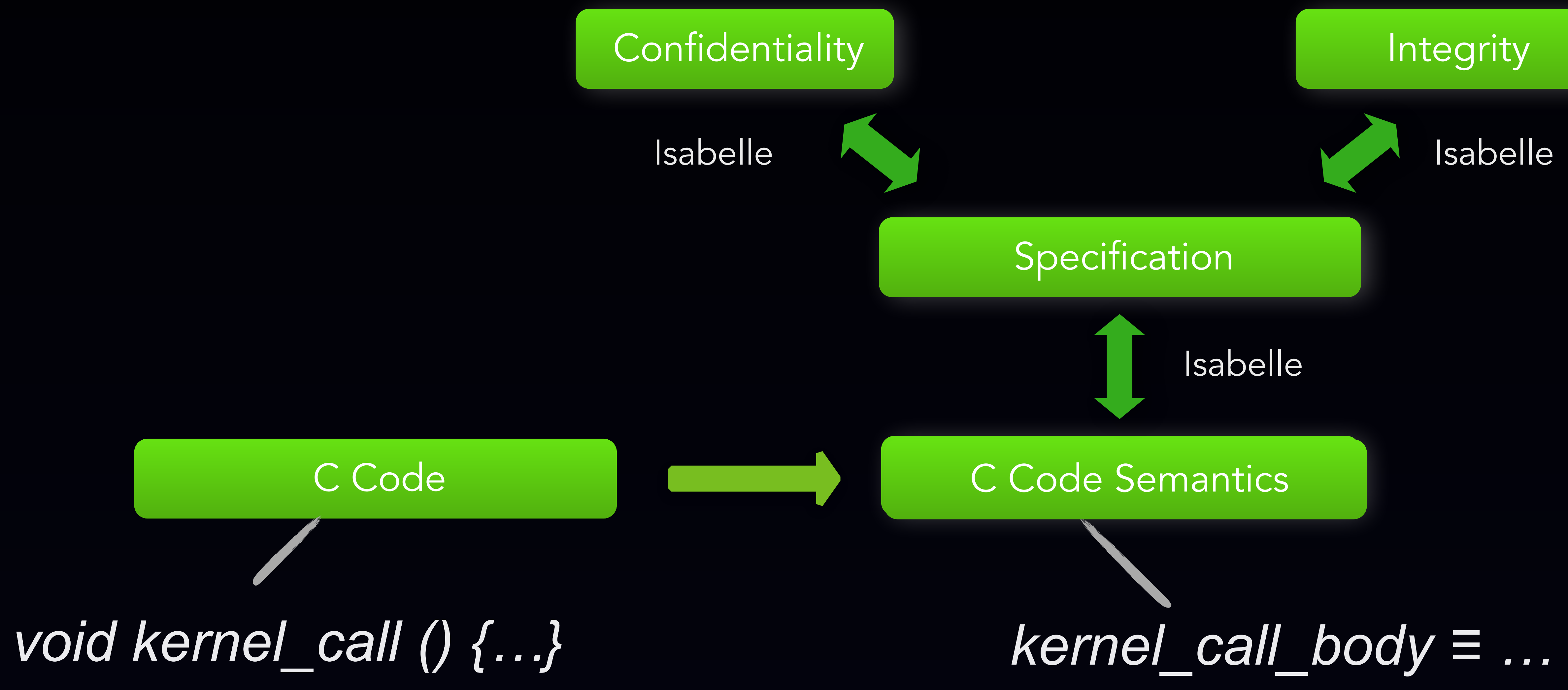
$\Longrightarrow$ *integrity p s s'*

no unauthorised modification

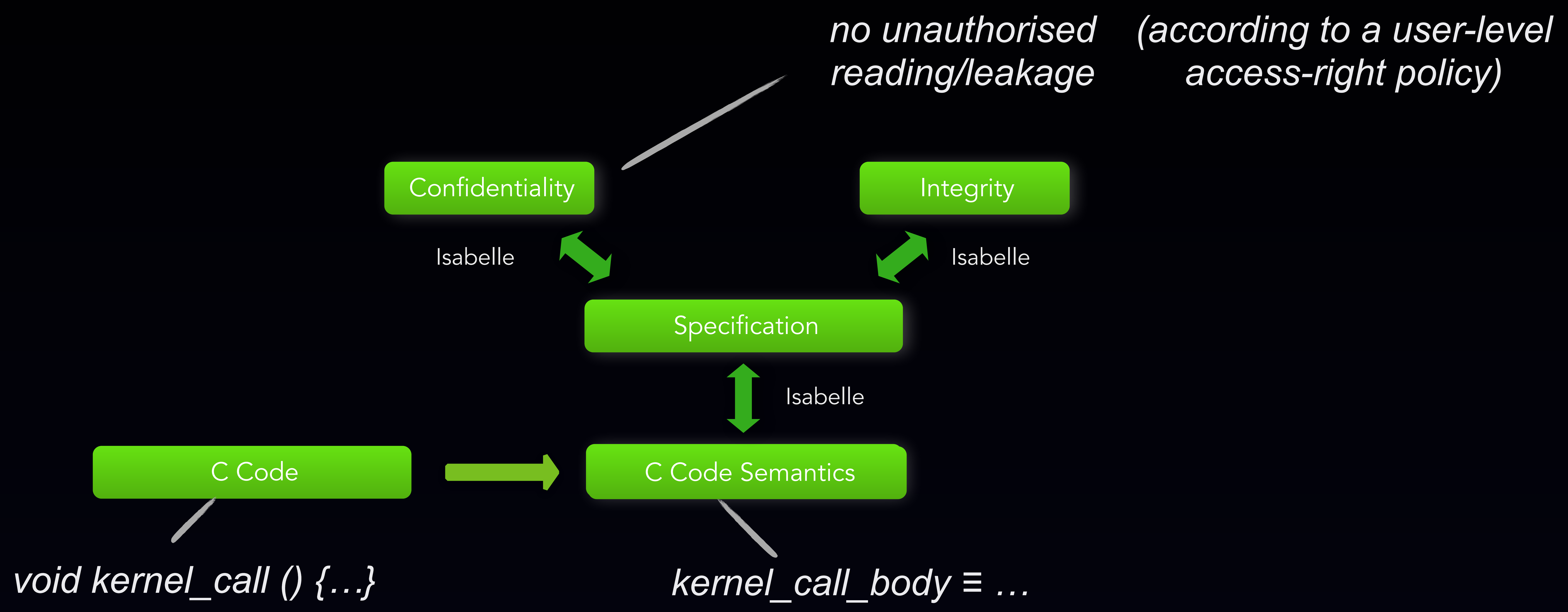
(according to a user-level access-right policy)

After executing the kernel, the only modifications between the final state and the initial state must be authorised by the policy

seL4 main theorem #3: confidentiality

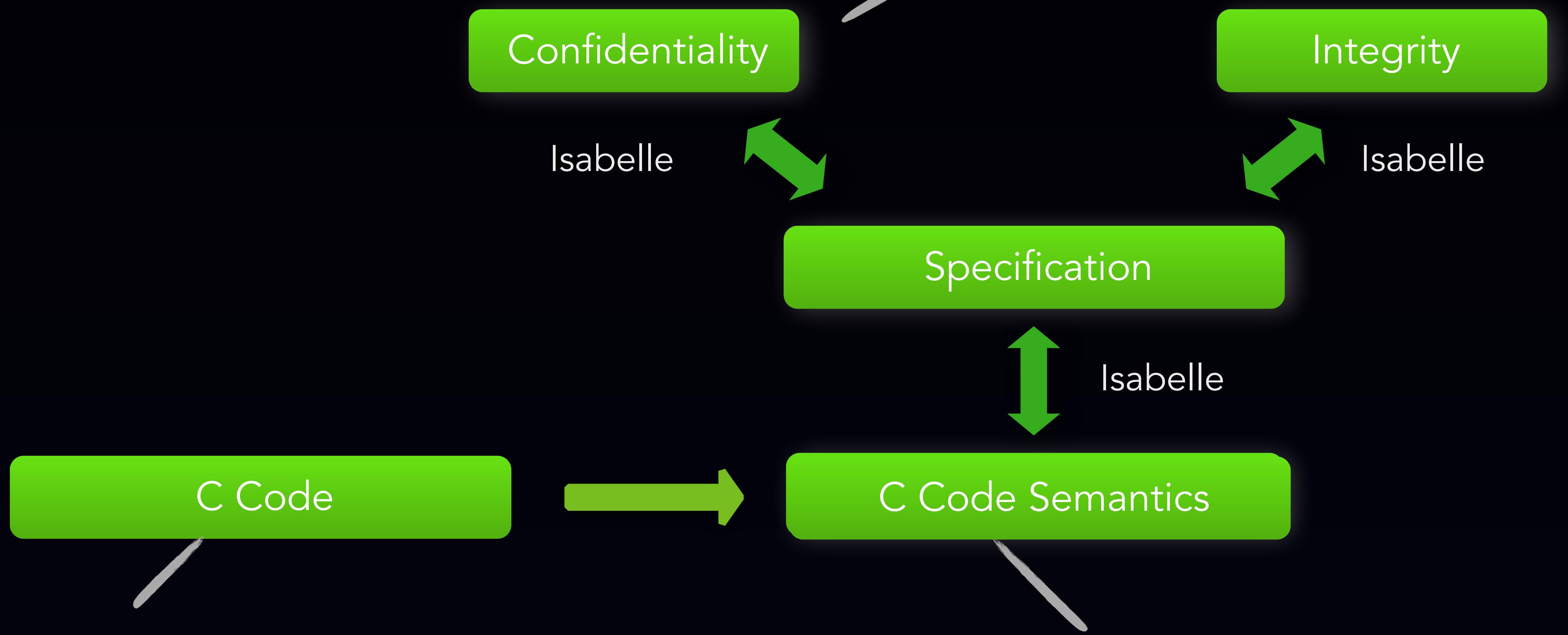


seL4 main theorem #3: confidentiality



seL4 main theorem #3: confidentiality

no unauthorised reading/leakage *(according to a user-level access-right policy)*



void kernel_call () {…}

kernel_call_body ≡ …

Non-interference:

2 executions of the kernel from states that differ only on confidential info must reach states that only differ from confidential info

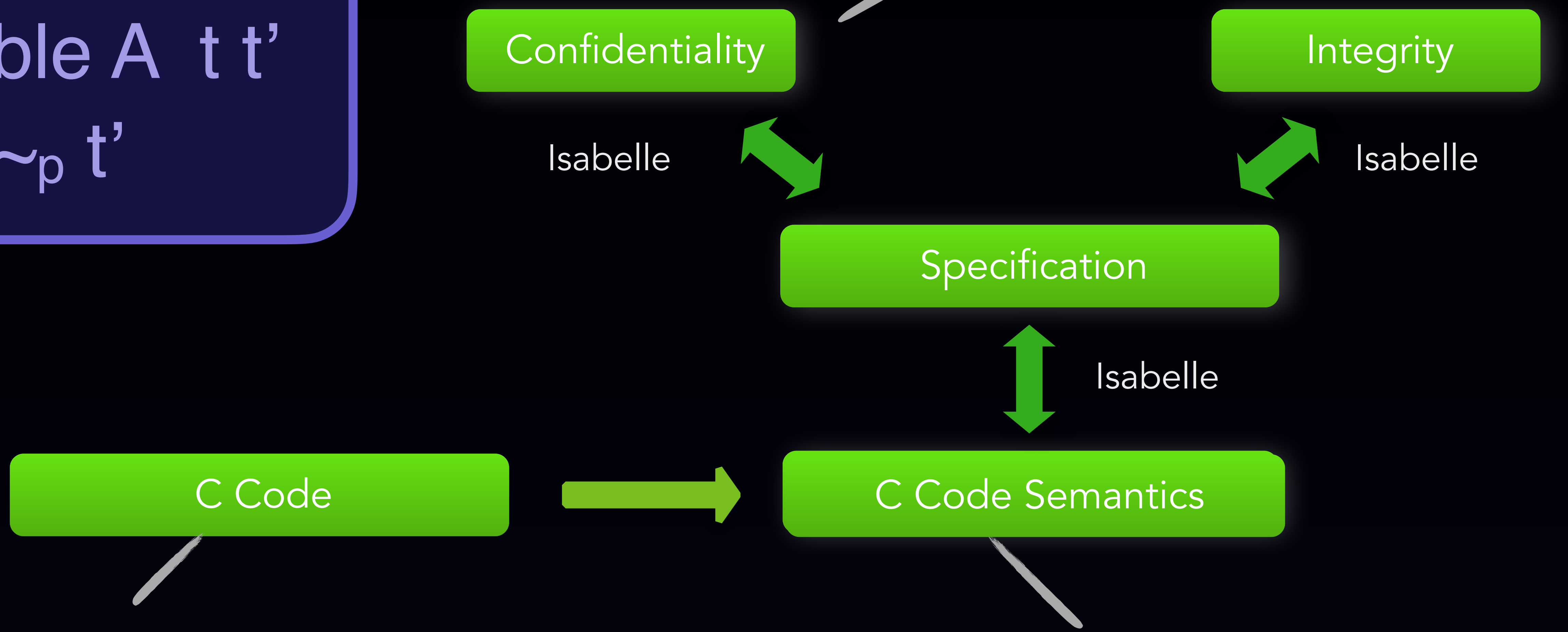
$$
\begin{array}{ccccccccc}
s & \longrightarrow & s_1 & \longrightarrow & s_2 & \longrightarrow & s_3 & \longrightarrow & \dots \\
\wr & & \wr & & \wr & & \wr & & \\
t & \longrightarrow & t_1 & \longrightarrow & t_2 & \longrightarrow & t_3 & \longrightarrow & \dots
\end{array}
$$

seL4 main theorem #3: confidentiality

$$
\begin{array}{l}
s \sim_p t \\
\wedge \quad \text{reachable } A\ s\ s' \\
\wedge \quad \text{reachable } A\ t\ t' \\
\Longrightarrow s' \sim_p t'
\end{array}
$$

2-property

no unauthorised reading/leakage *(according to a user-level access-right policy)*



void kernel_call () {…}

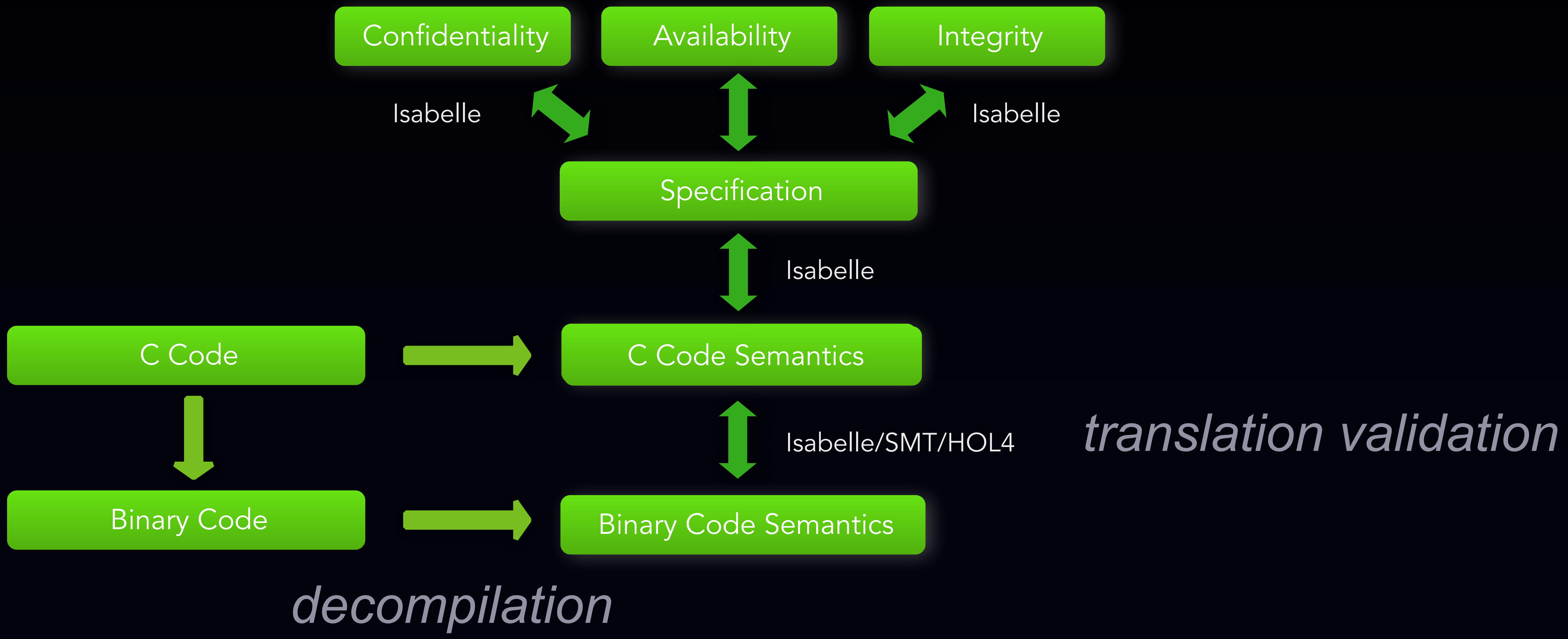
kernel_call_body ≡ …

Non-interference:

2 executions of the kernel from states that differ only on confidential info must reach states that only differ from confidential info

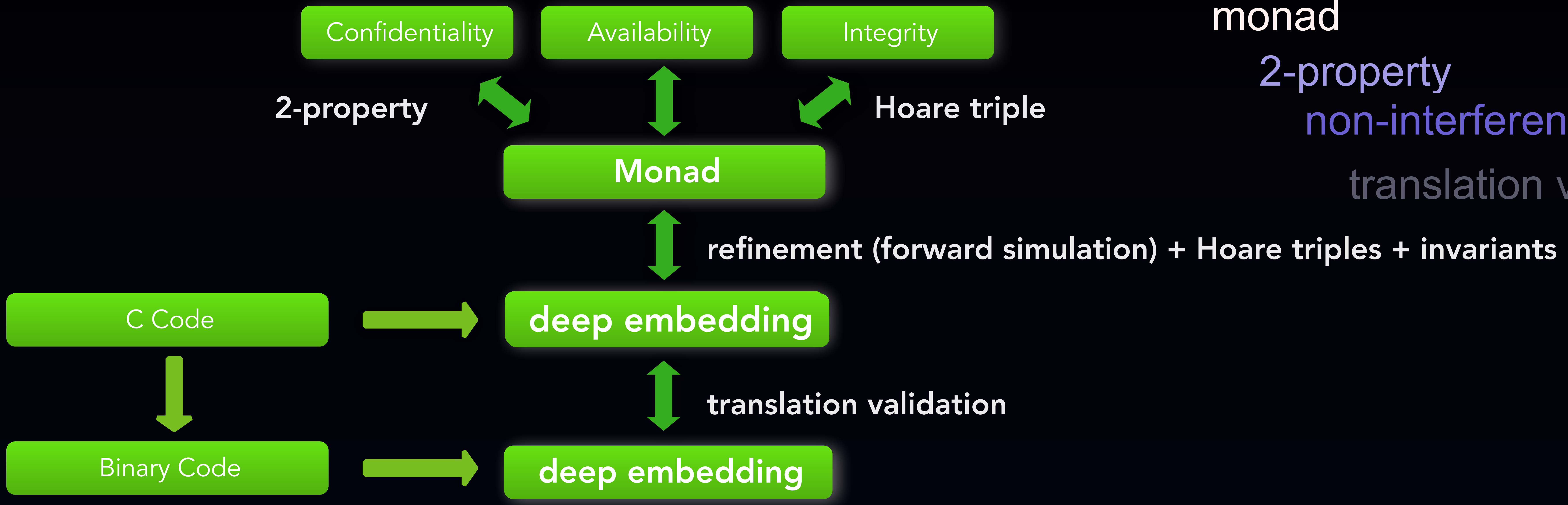
$$
\begin{array}{ccccccccc}
s & \longrightarrow & s_1 & \longrightarrow & s_2 & \longrightarrow & s_3 & \longrightarrow & \dots \\
\wr & & \wr & & \wr & & \wr & & \\
t & \longrightarrow & t_1 & \longrightarrow & t_2 & \longrightarrow & t_3 & \longrightarrow & \dots
\end{array}
$$

seL4 main theorem #4: binary verification

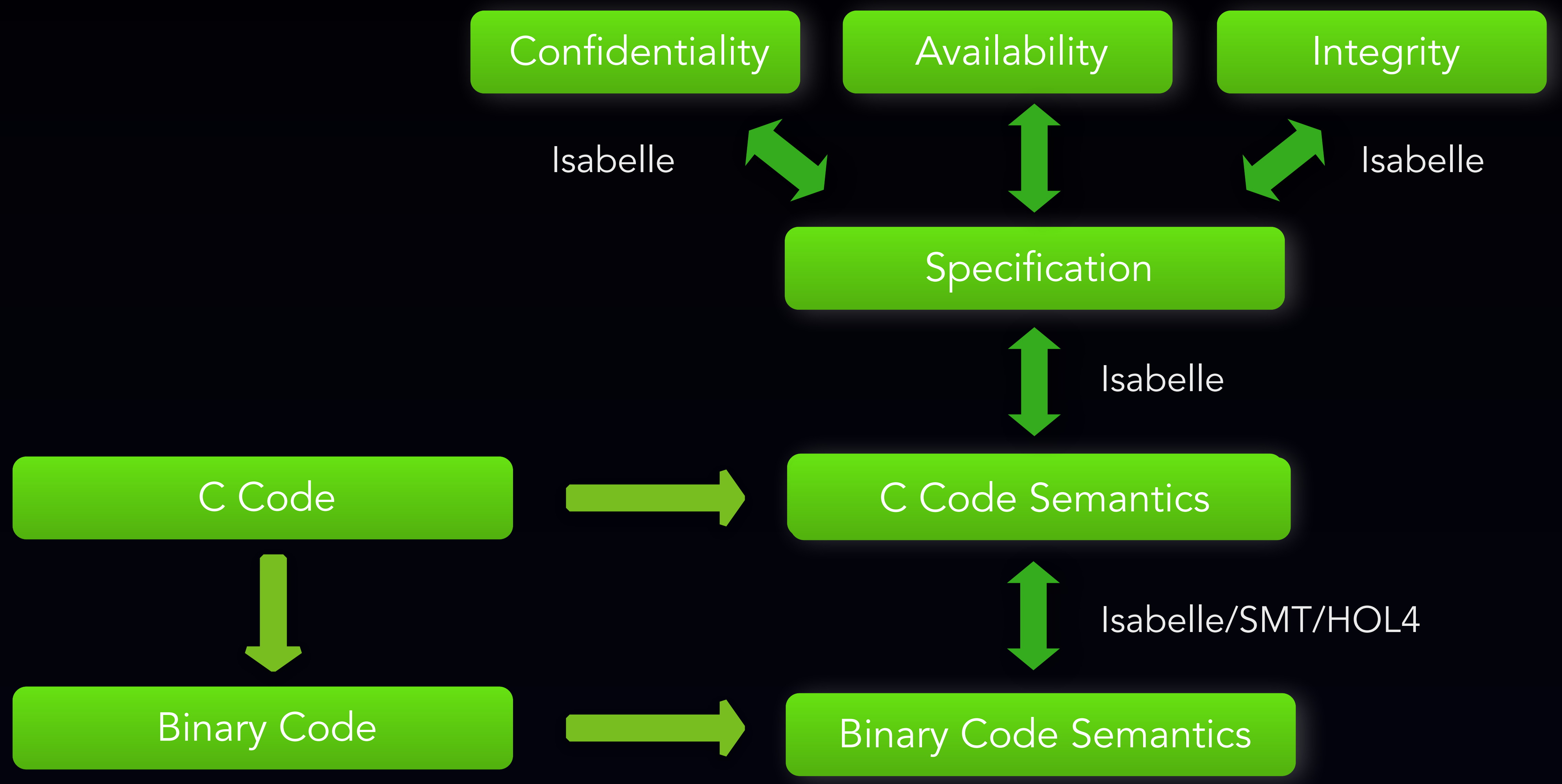


seL4 main theorems

refinement
forward simulation
Hoare triple
invariant
deep embedding
shallow embedding
monad
2-property
non-interference
translation validation



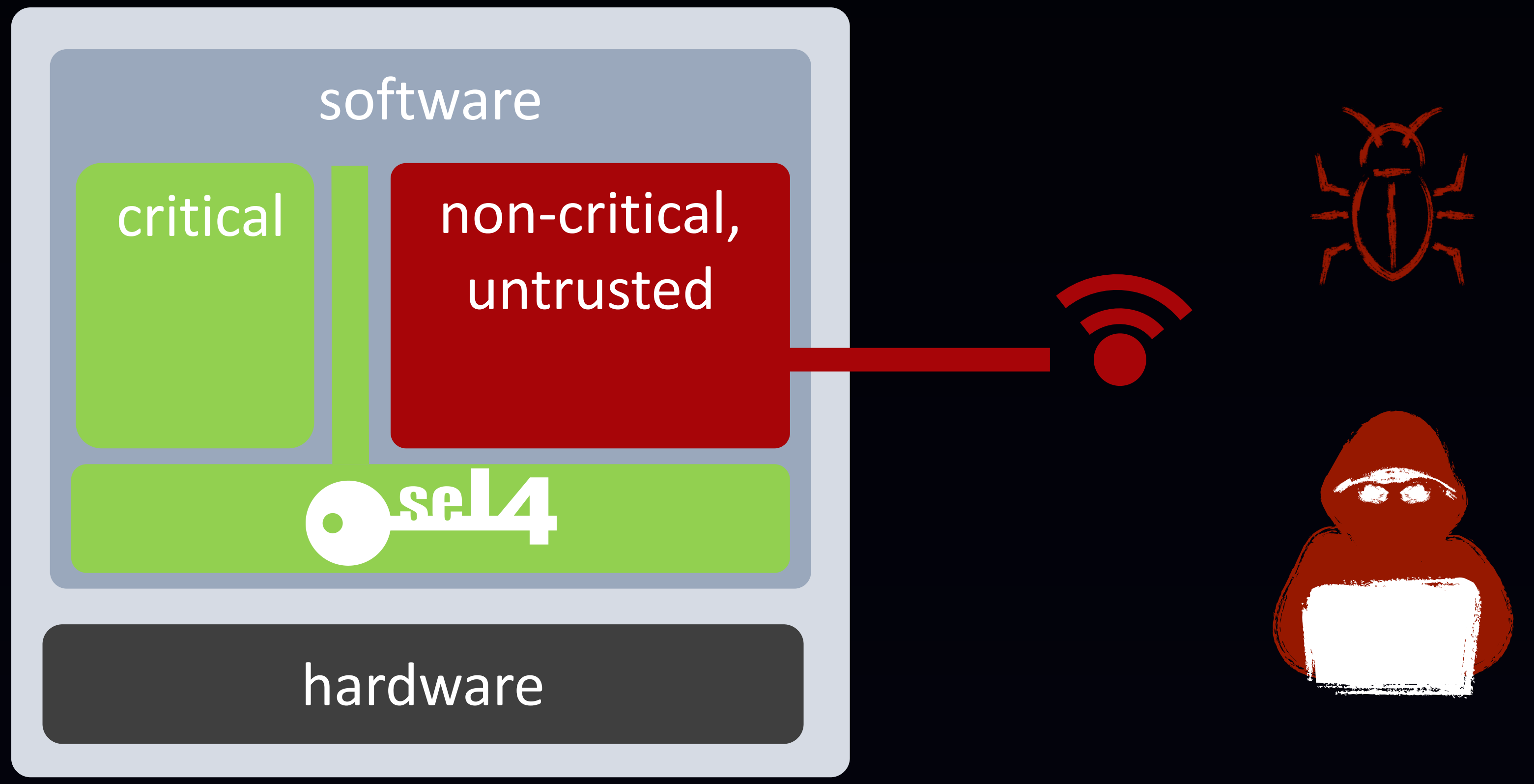
Impact



- Binary is correct w.r.t the spec and enforces isolation

Impact

▸ Binary is correct w.r.t the spec and enforces isolation



World's most comprehensive mathematical proofs of correctness and security

World's fastest microkernel

Overview

#1
Make a dream come true:
verified, performant kernel

Opportunities:

- achieve a decades-long dream
- demonstrate FM on real systems

Challenges:

- Scale
- Thoroughness
- Performance

Solutions:

- Combination of foundational techniques
- Targeting machine-checked proof
- Working hand in hand with systems people

Overview

#1
Make a dream come true:
verified, performant kernel

Success!

Opportunities:

- achieve a decades-long dream
- demonstrate FM on real systems

Challenges:

- Scale
- Thoroughness
- Performance

Solutions:

- Combination of foundational techniques
- Targeting machine-checked proof
- Working hand in hand with systems people

Overview

#2
Deliver it to the world:
true trustworthiness for critical software

Photo by Xan Griffin on Unsplash

Overview

#2
Deliver it to the world:
true trustworthiness for critical software

Opportunities:

- used in products where it matters
- set a standard

Photo by Xan Griffin on Unsplash

Setting a standard

The practical advantages of program proving
will eventually outweigh the difficulties,
in view of the increasing costs of programming error.

Tony Hoare, 1969

Setting a standard

The practical advantages of program proving
will eventually outweigh the difficulties,
in view of the increasing costs of programming error.

Tony Hoare, 1969

If the issue ever came to court,
the defense of 'state-of-the-art' practice would always prevail.

Tony Hoare, 2009

Setting a standard

The practical advantages of program proving
will eventually outweigh the difficulties,
in view of the increasing costs of programming error.

Tony Hoare, 1969

If the issue ever came to court,
the defense of 'state-of-the-art' practice would always prevail.

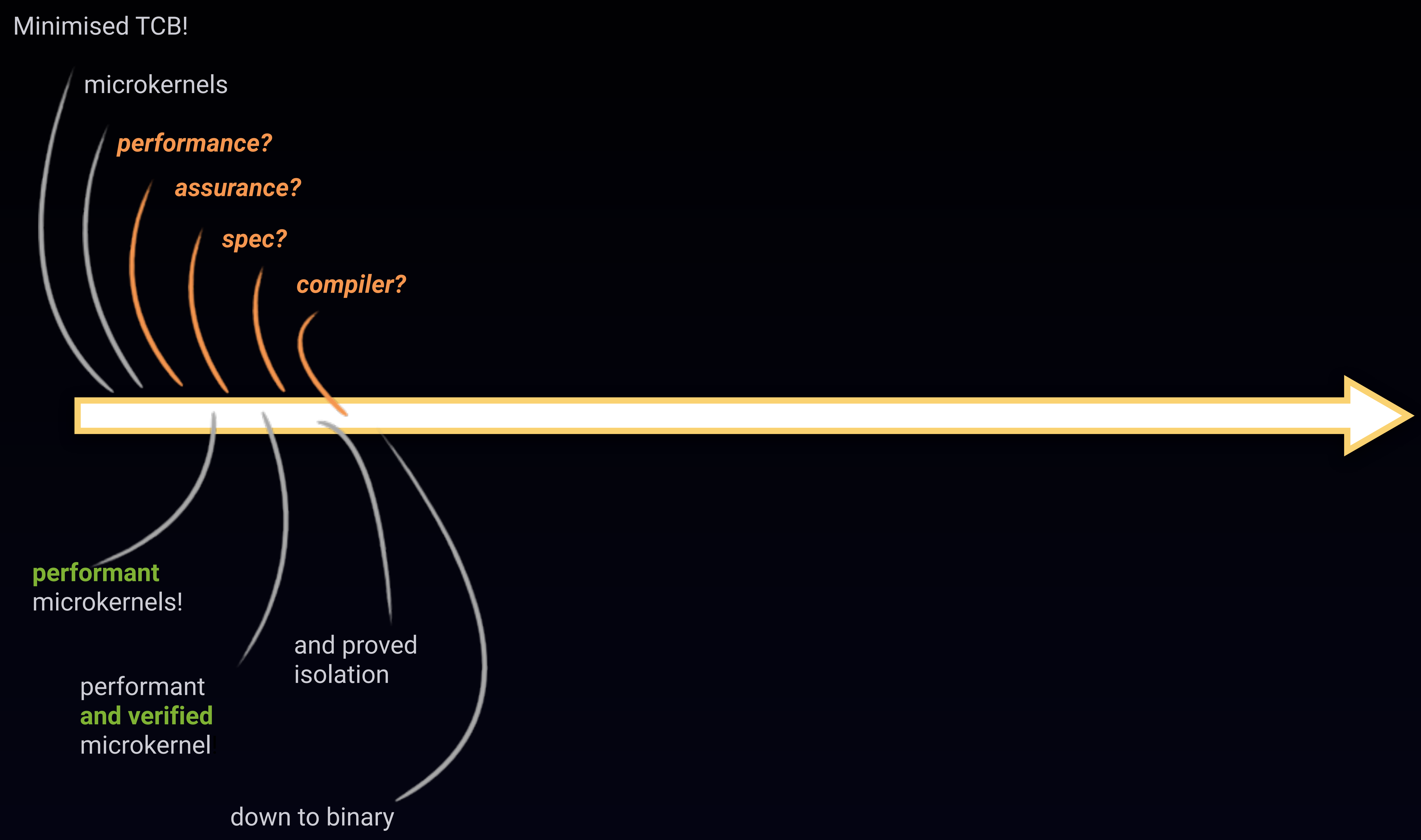
Tony Hoare, 2009

When total absence of error is a requirement
(e.g., in aircraft control software or operating system security),
failure to verify will be treated legally as negligence,
as in other branches of engineering.

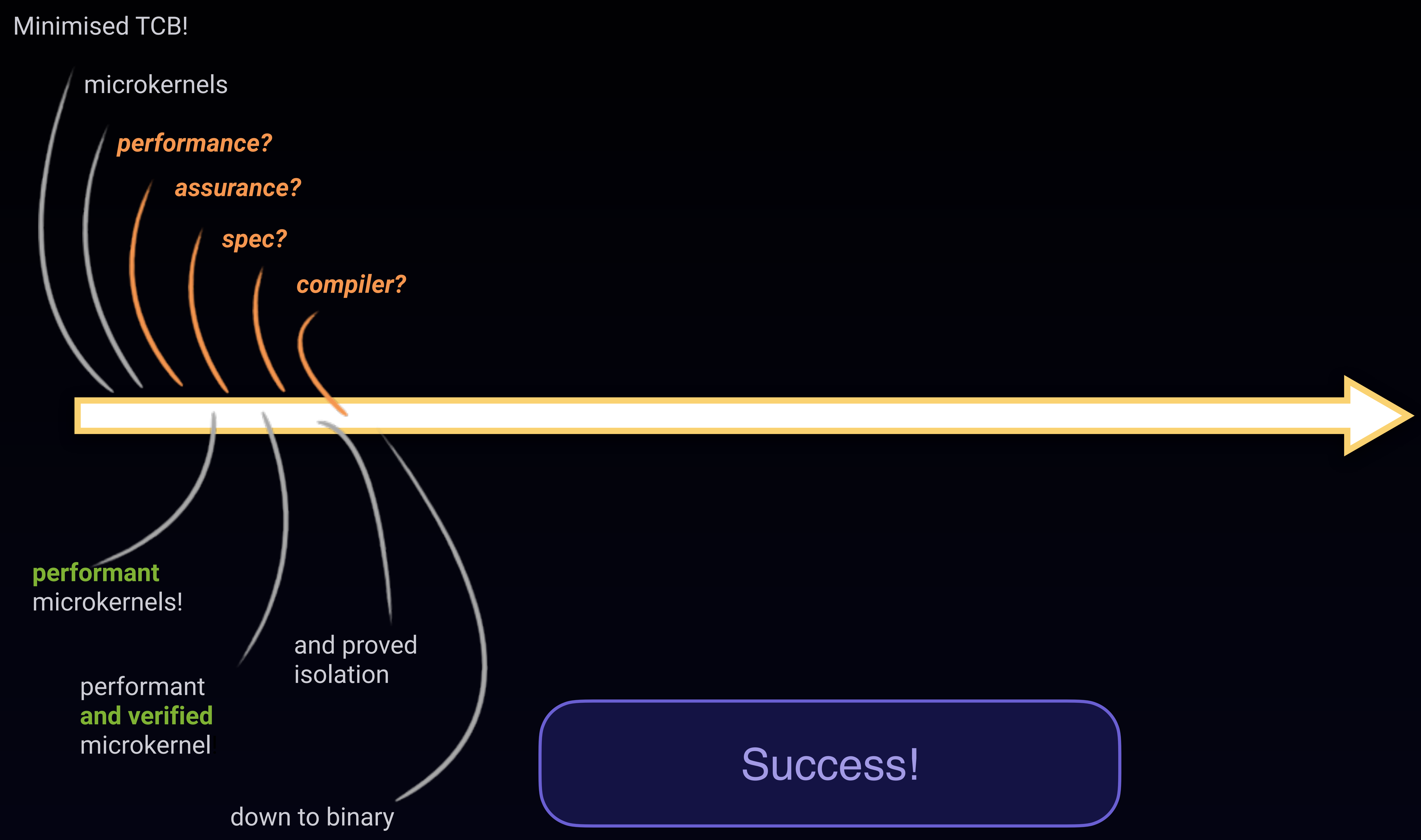
But this cannot happen until there is wide-ranging evidence of feasibility, cost, and tool support
of experimental verification of realistic applications.

The sel4 microkernel is just the sort of demonstration that convinces.

The seL4 journey

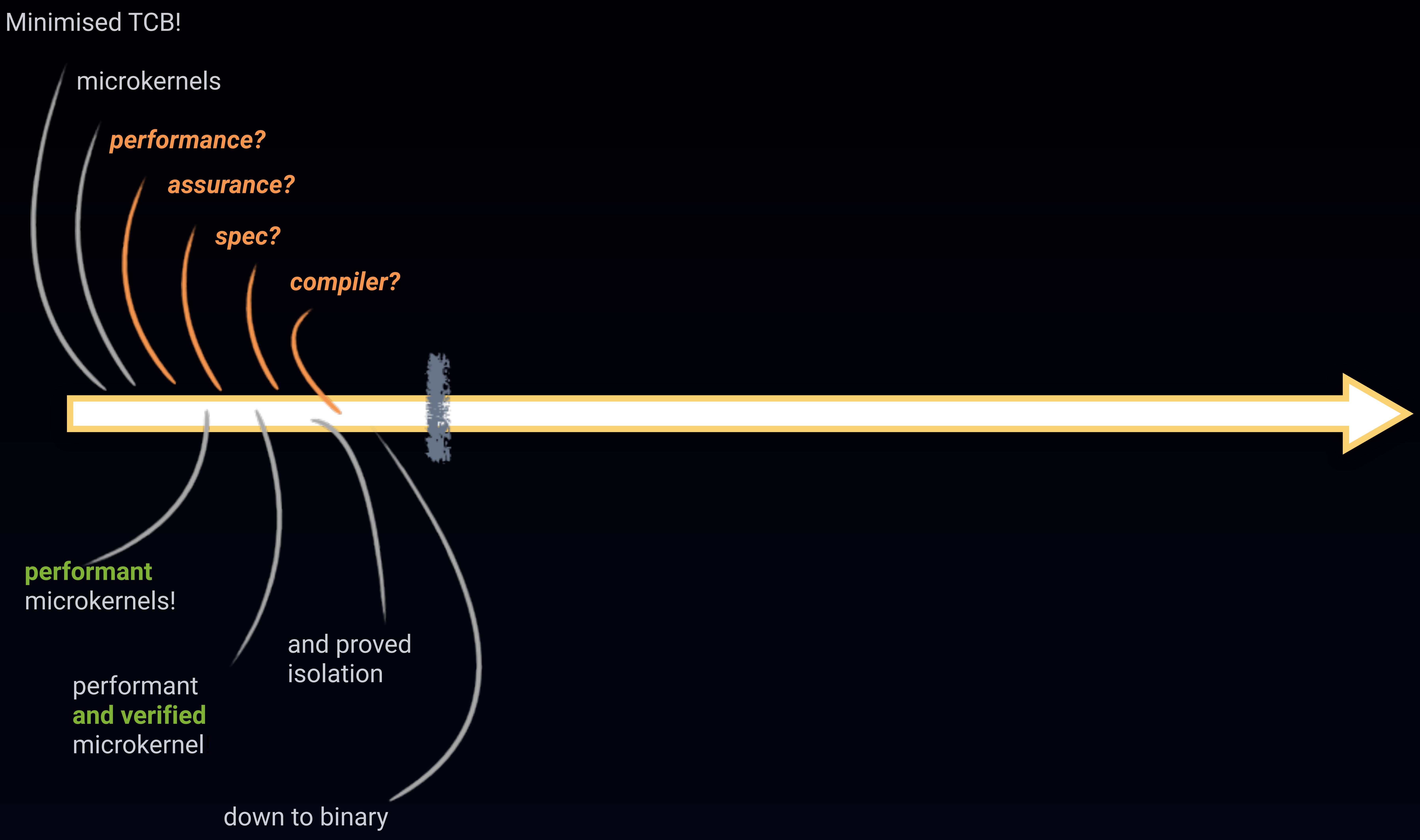


The seL4 journey

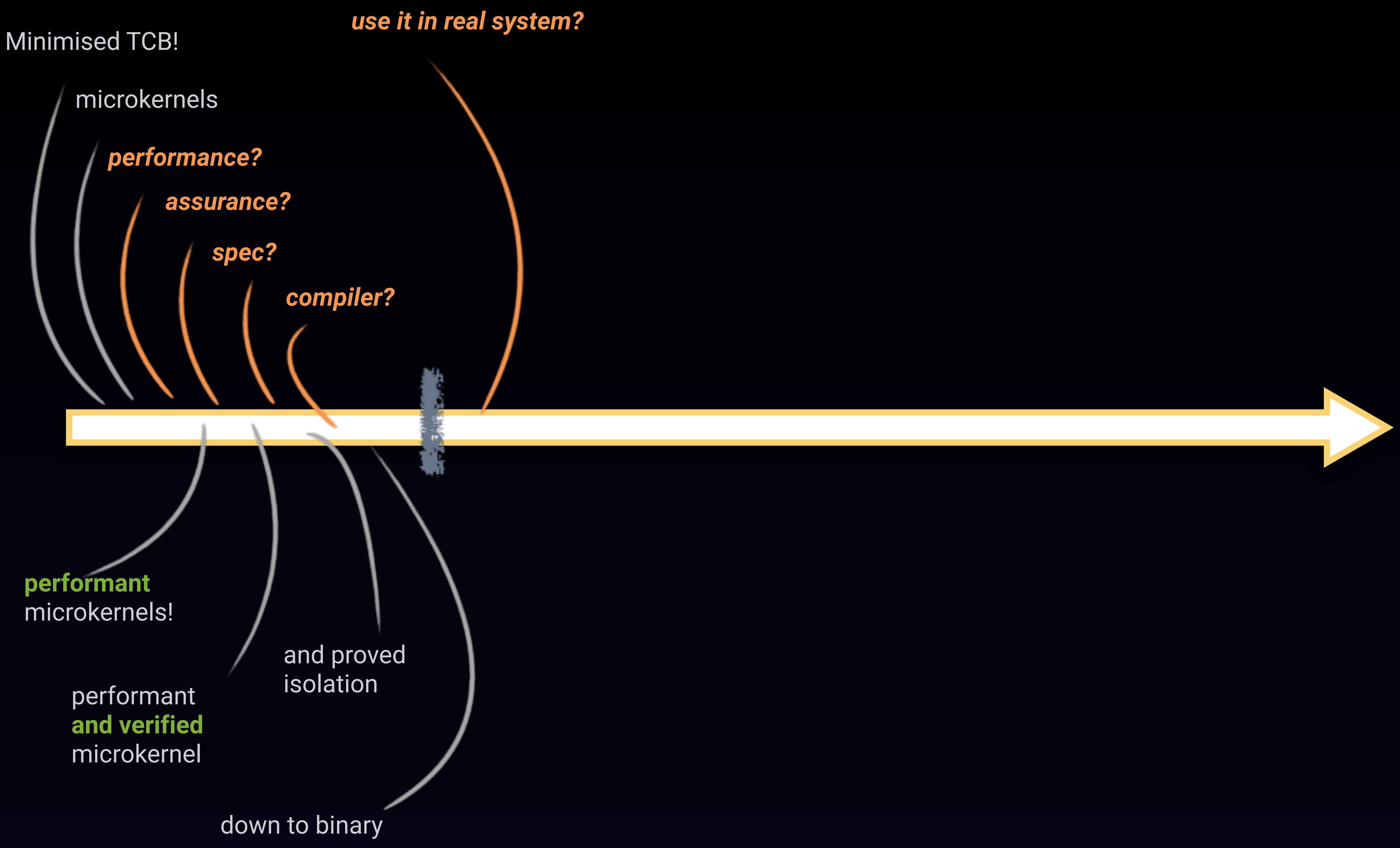


Success!

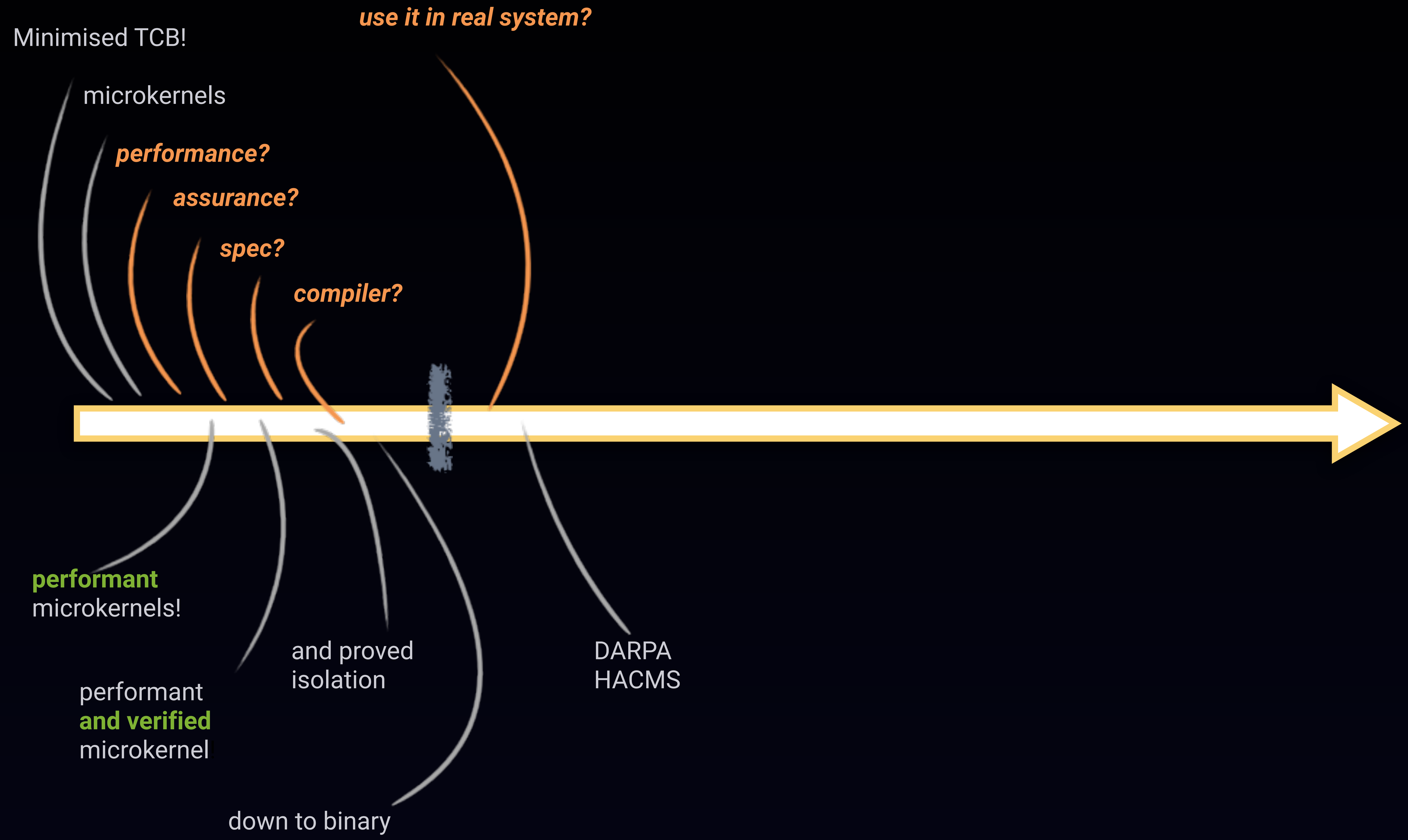
The seL4 journey



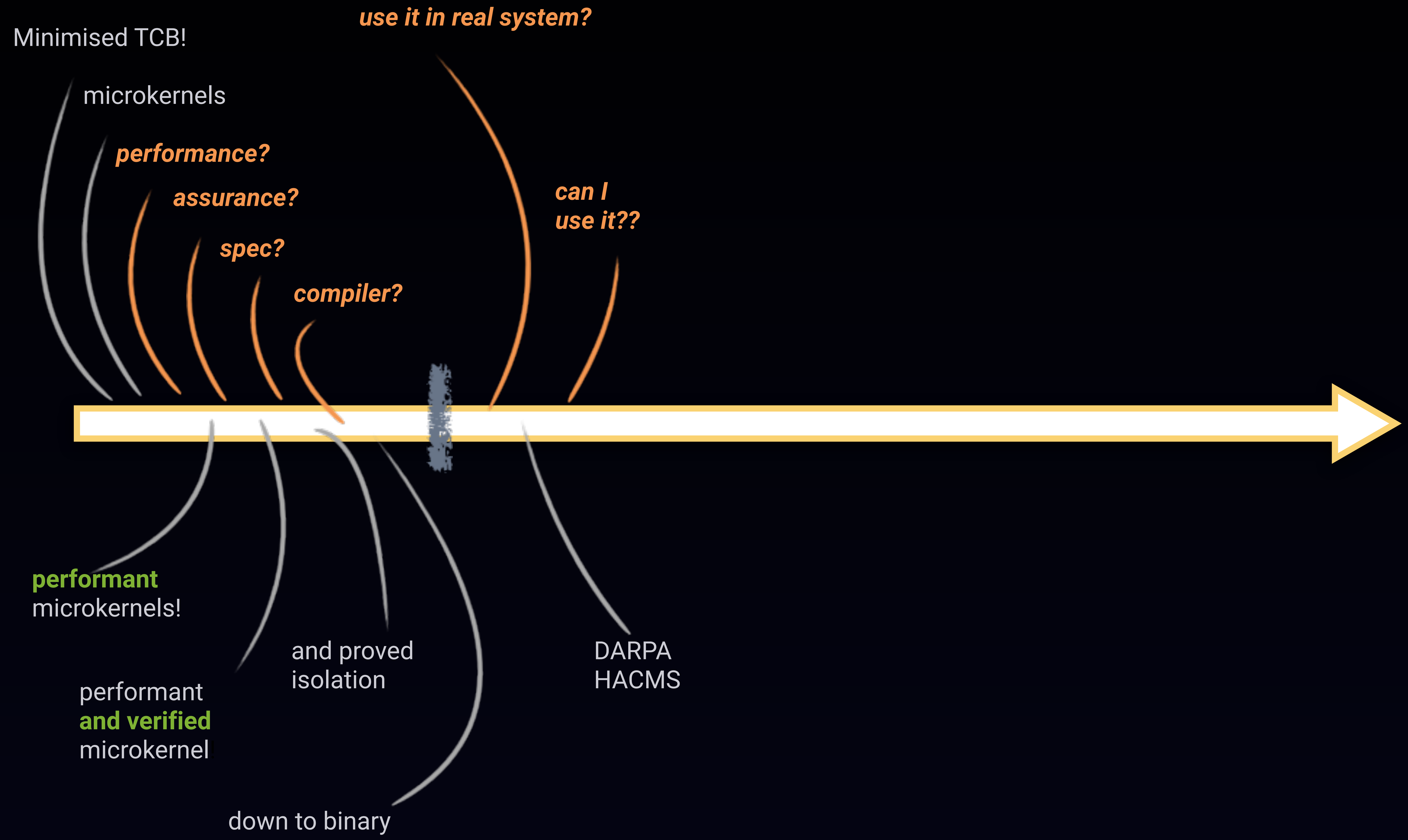
The seL4 journey



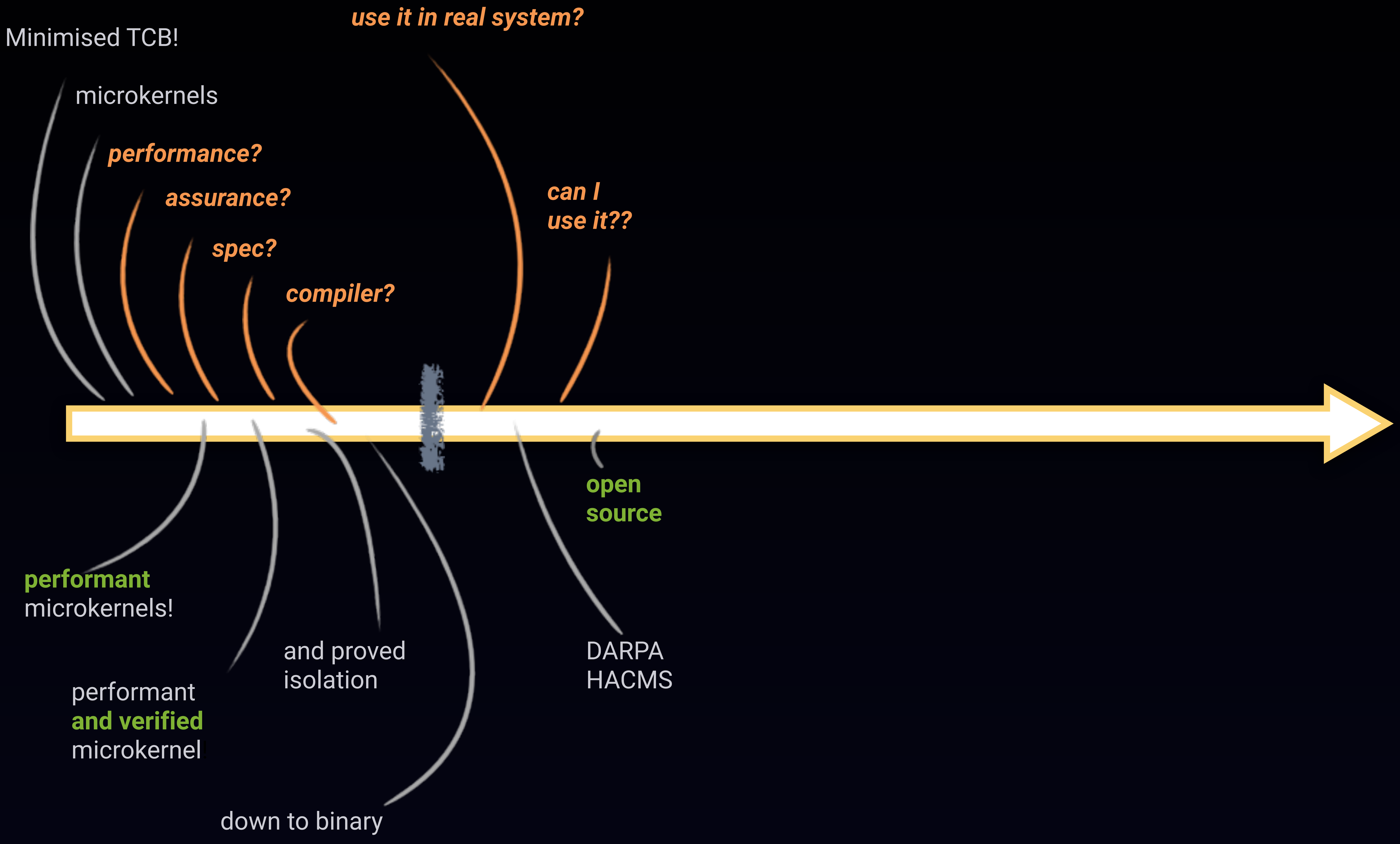
The seL4 journey



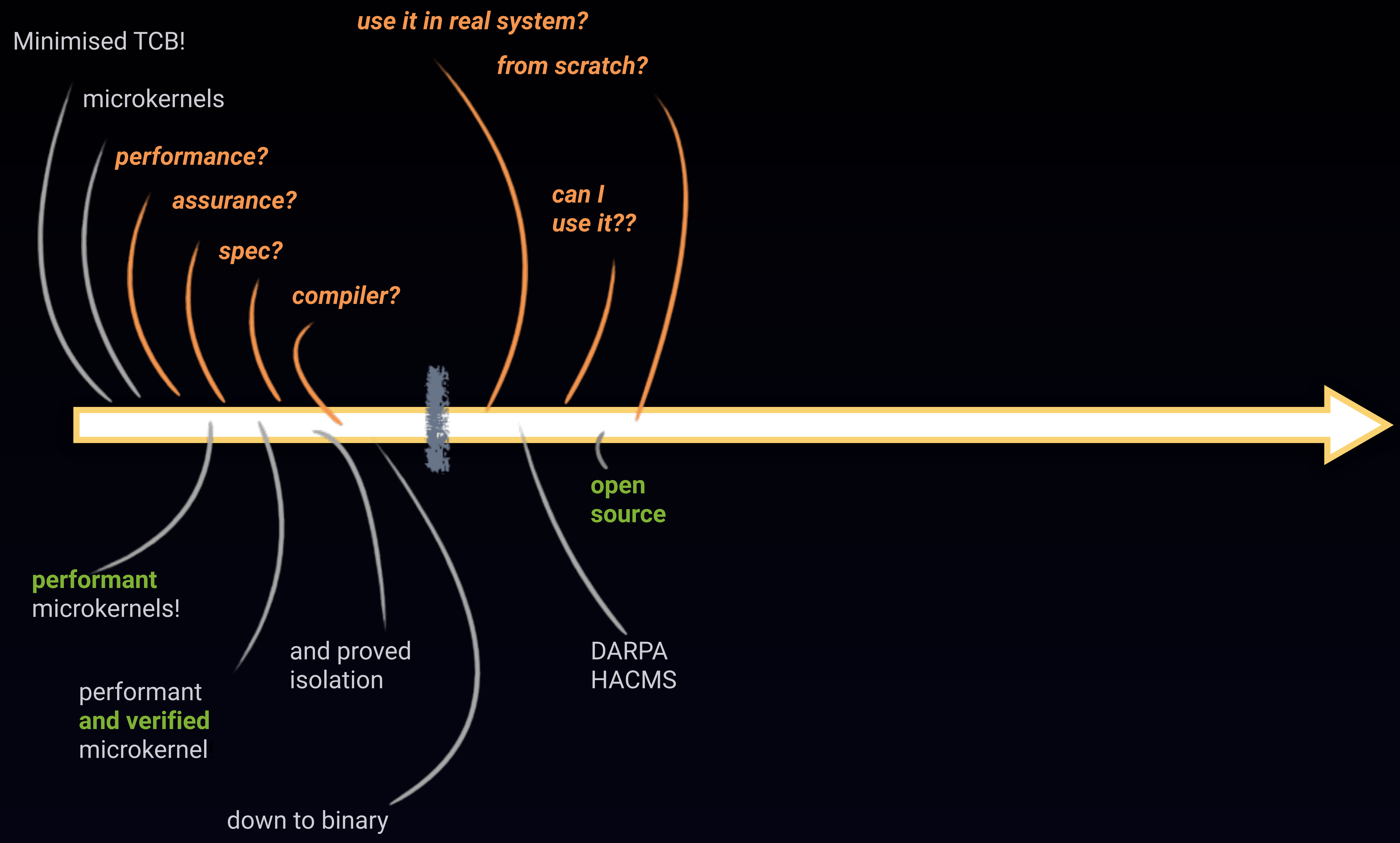
The seL4 journey



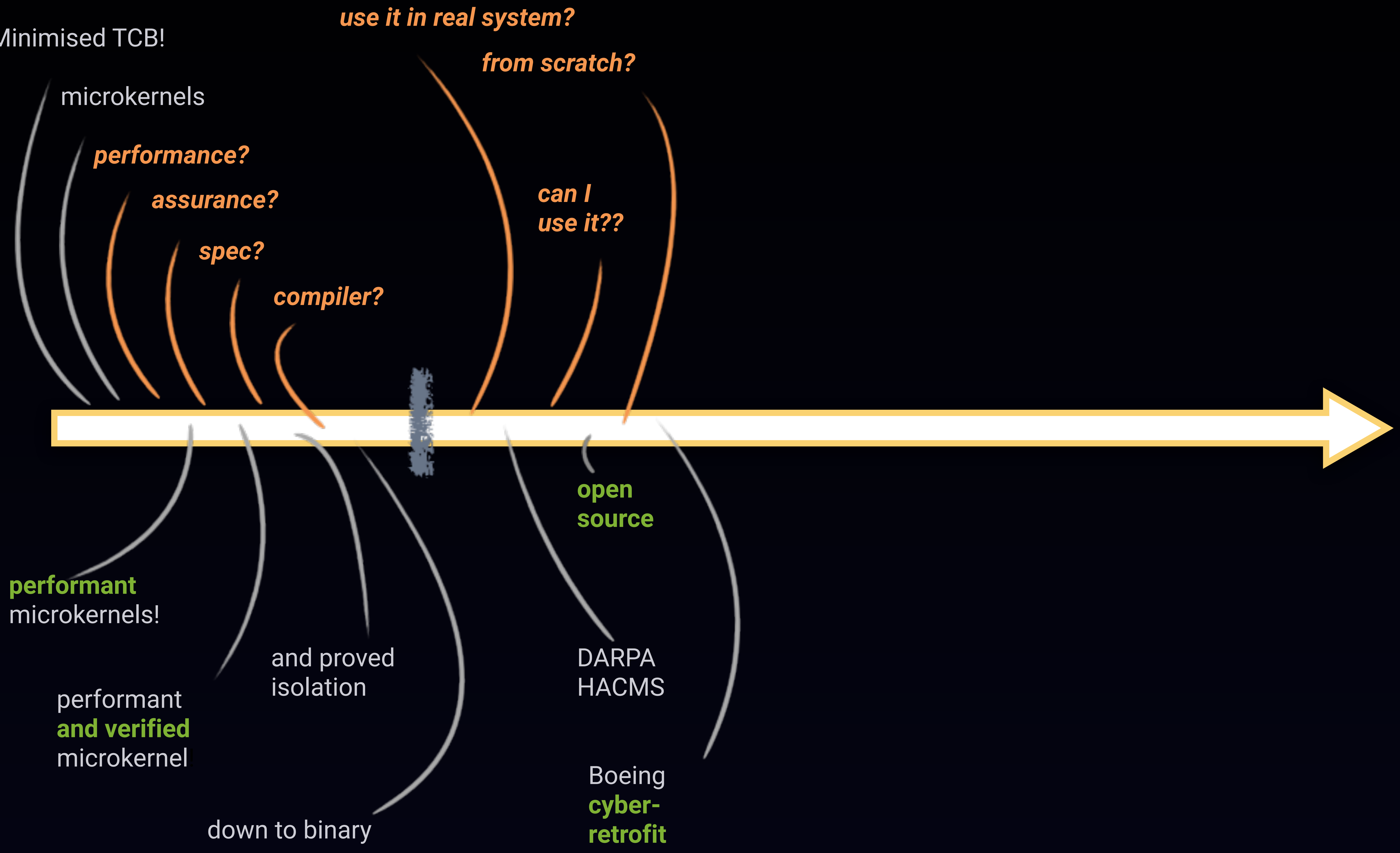
The seL4 journey



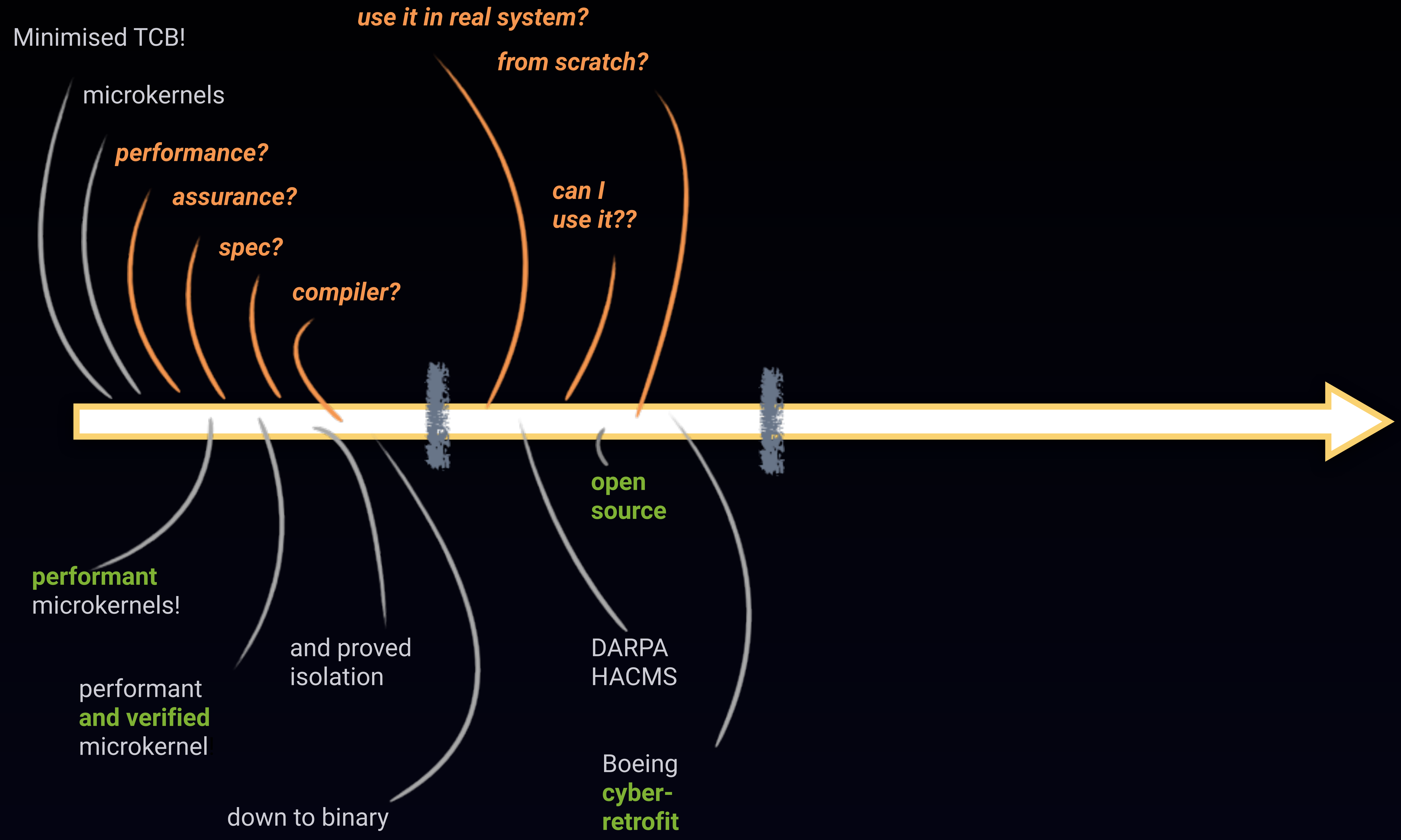
The seL4 journey



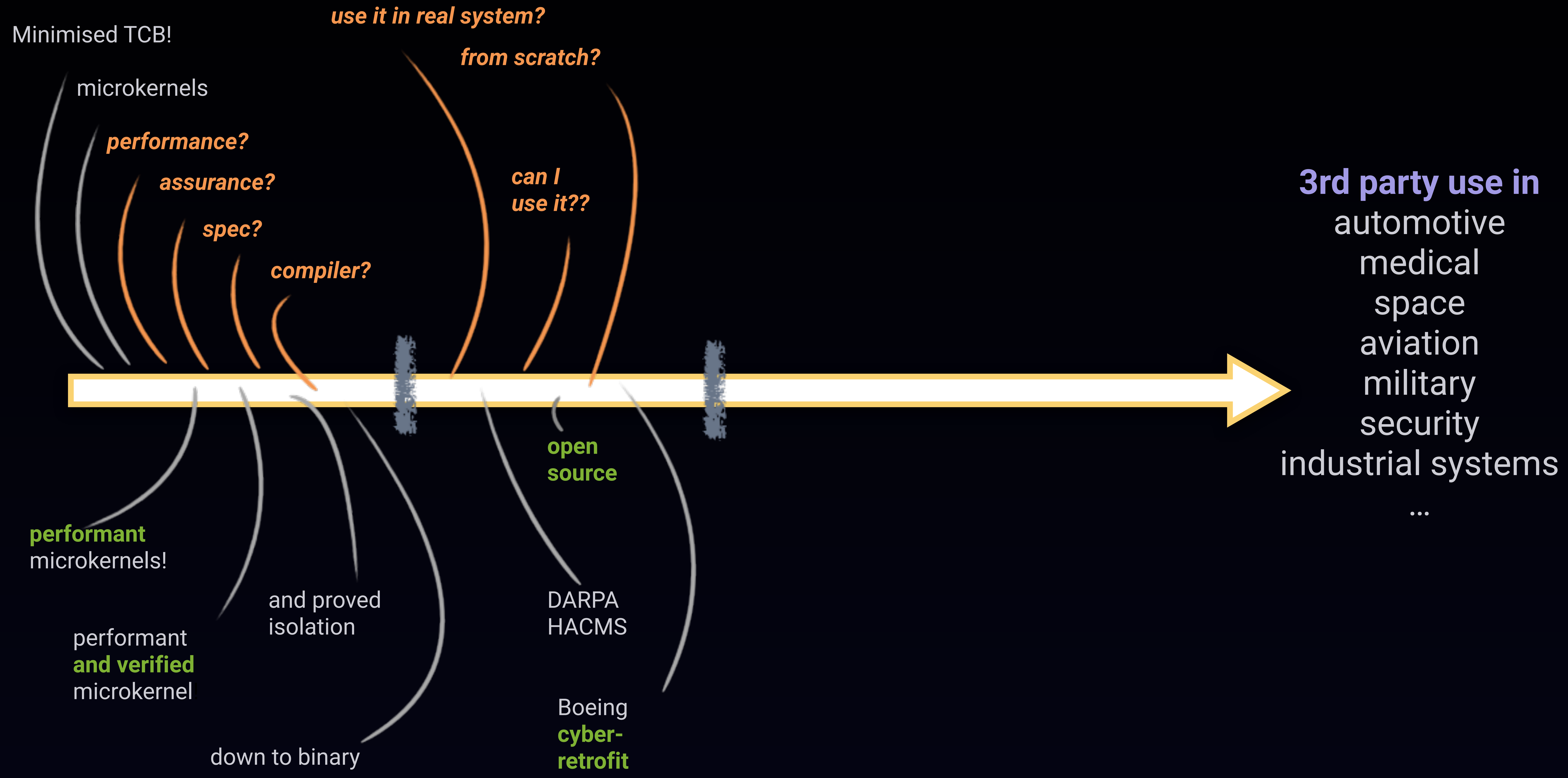
The seL4 journey



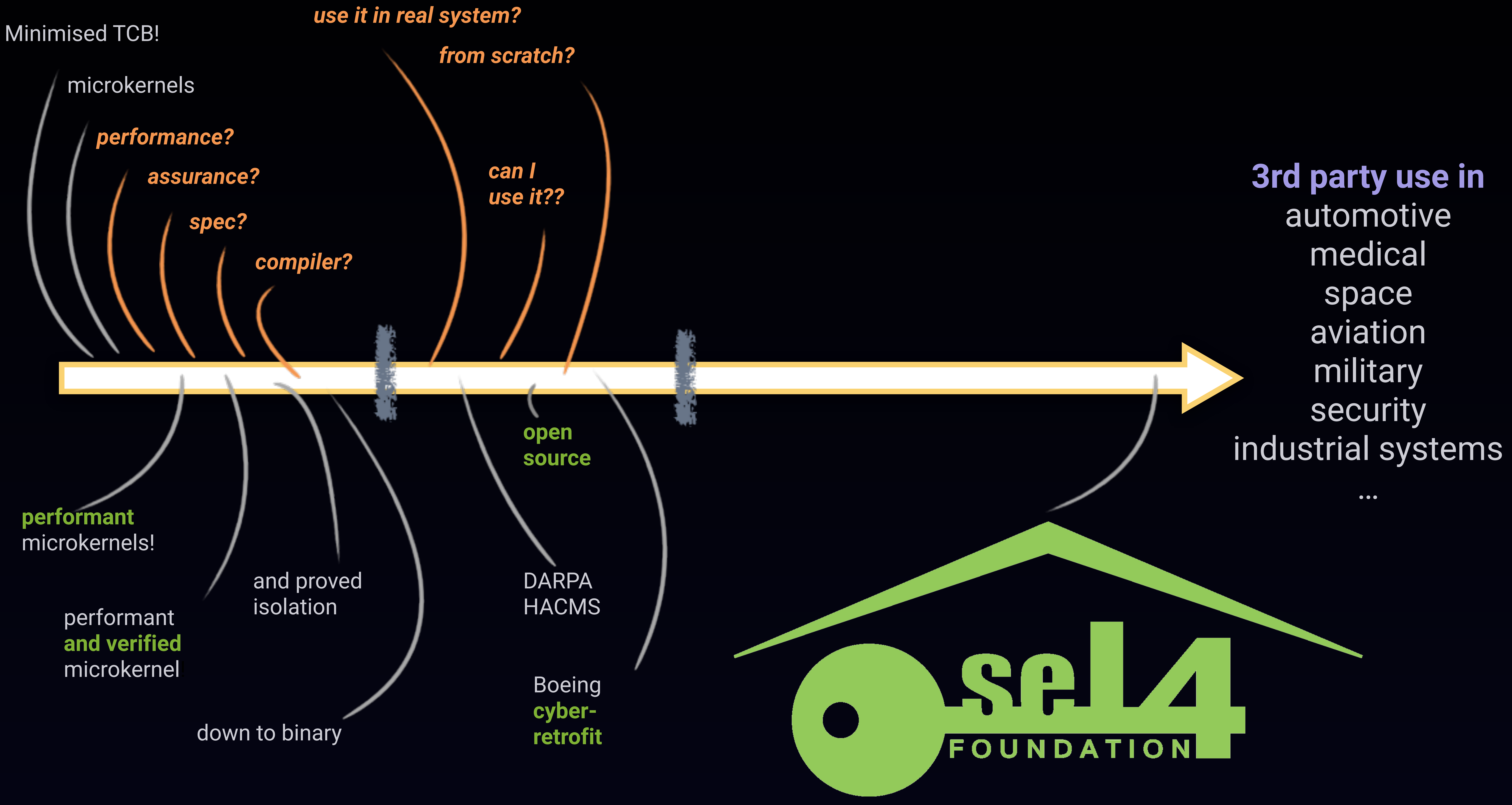
The seL4 journey



The seL4 journey



The seL4 journey



The seL4 journey

- Membership: 26
  - Premium Members: 6
  - General Members: 15
  - Associate Members: 5

3rd party use in
automotive
medical
space
aviation
military
security
industrial systems
…

General Members

Adventium Labs

Cog Systems Inc
Founding member
Endorsed Service Provider

DornerWorks Ltd
Founding member
Endorsed Service Provider

Ghost Locomotion Inc
Founding member

Google LLC

Kry10 Limited
Endorsed Service Provider

Lotus Cars

Penten Pty Ltd

Proofcraft Pty Ltd
Endorsed Service Provider

Raytheon Technologies

SpacemiT

Technology I

Xcalibyte

LatticeX

Premium Members

HENSOLDT Cyber GmbH
Founding member
Endorsed Service Provider

Horizon Robotics

Jump Trading

Li Auto Inc

NIO

UNSW Sydney
Founding member
Endorsed Service Provider

Associate Members

ETH Zurich

Kansas State University

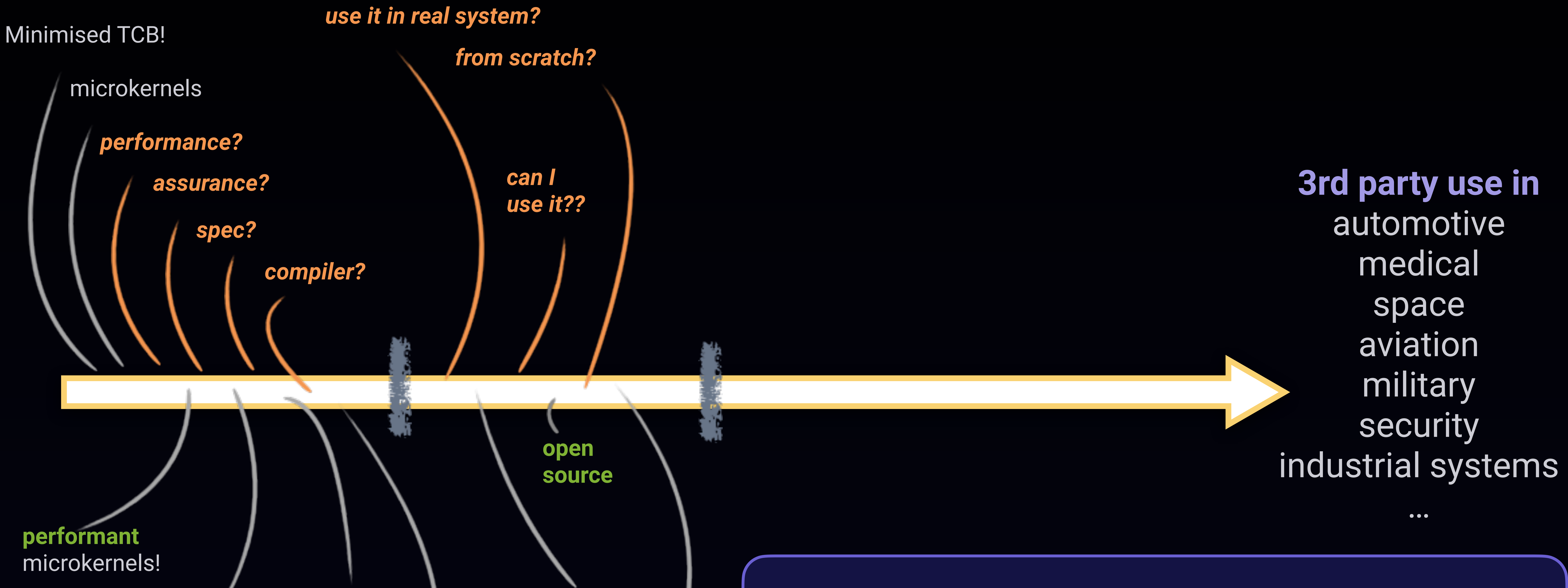
NCSC

RISC-V International

TU Munich

seL4 FOUNDATION

The seL4 journey

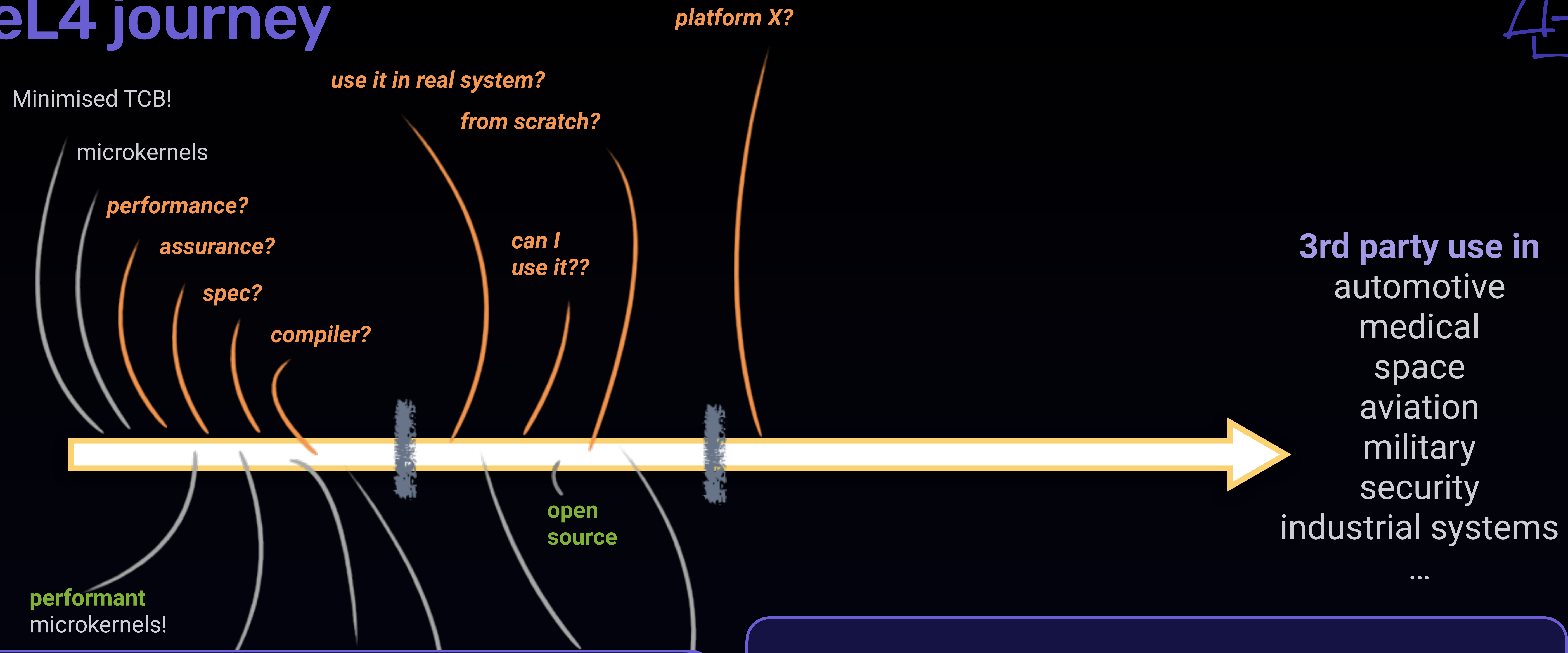


Success drives code evolution,
code evolution requires proof evolution

Challenges:

- port verification to new platforms
- port verification to new features

The seL4 journey



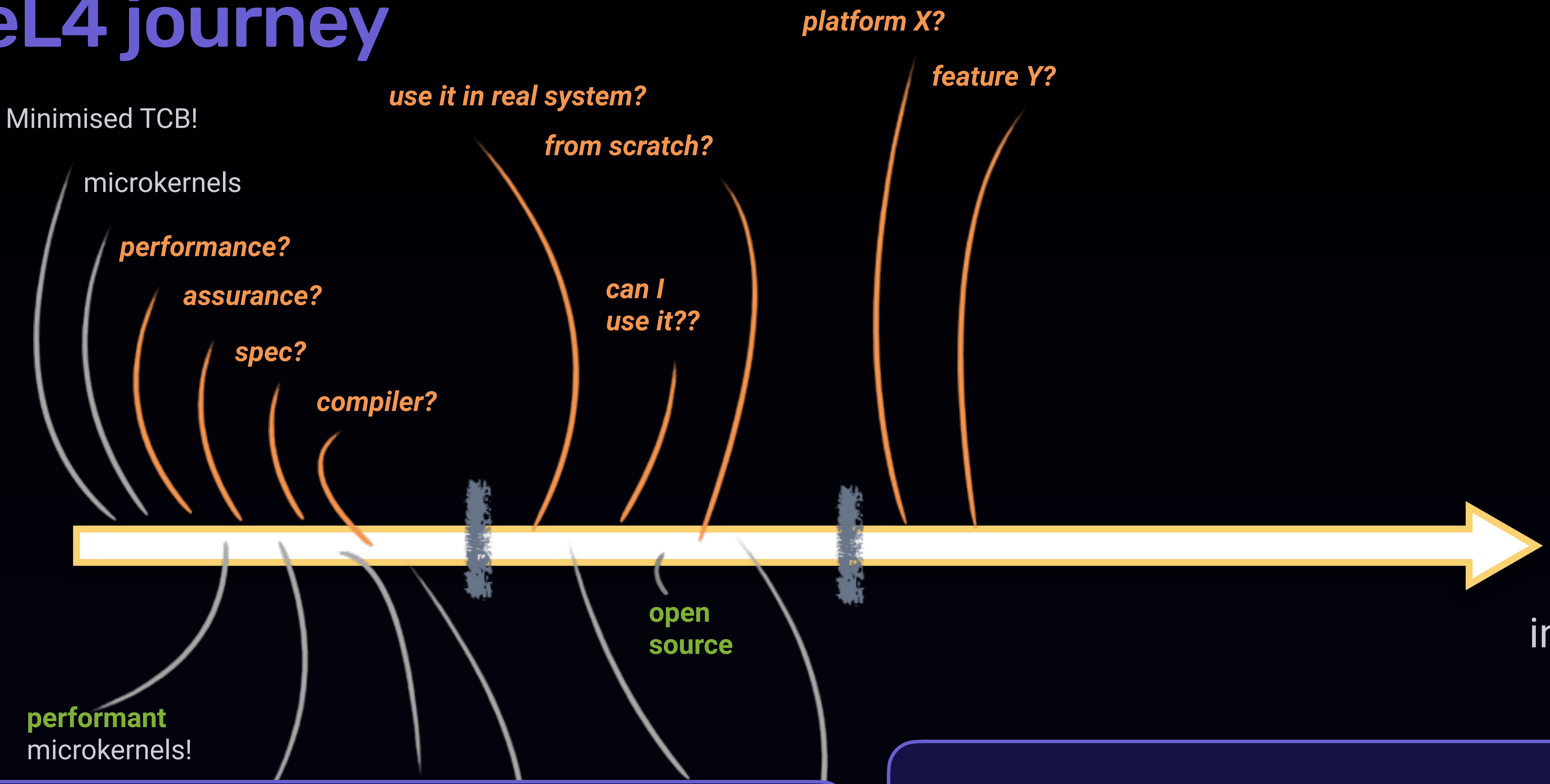
Success drives code evolution,

code evolution requires proof evolution

Challenges:

- port verification to new platforms
- port verification to new features

The seL4 journey



3rd party use in
automotive
medical
space
aviation
military
security
industrial systems
...

Success drives code evolution,
code evolution requires proof evolution

Challenges:

- port verification to new platforms
- port verification to new features

I want it all. And I want it now.

Photo by Nathan Dumlao on Unsplash

I want seL4 verified “with X on Y”

(It’s usually what we don’t have in stock :)

seL4-vanilla

seL4-MCS

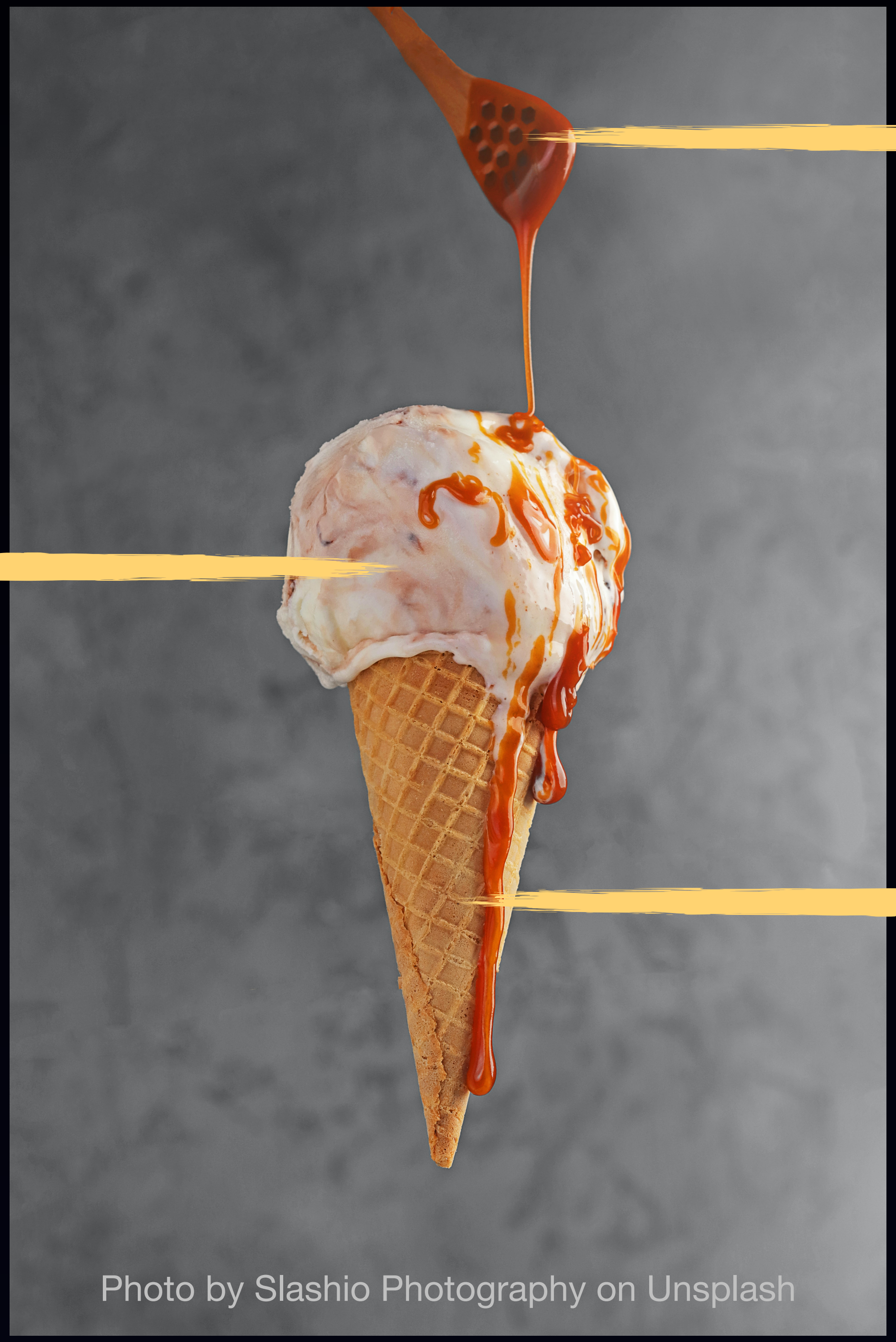


Photo by Slashio Photography on Unsplash

HYP

...

ARM

32b

64b

RISC-V

32b

64b

x86

32b

64b

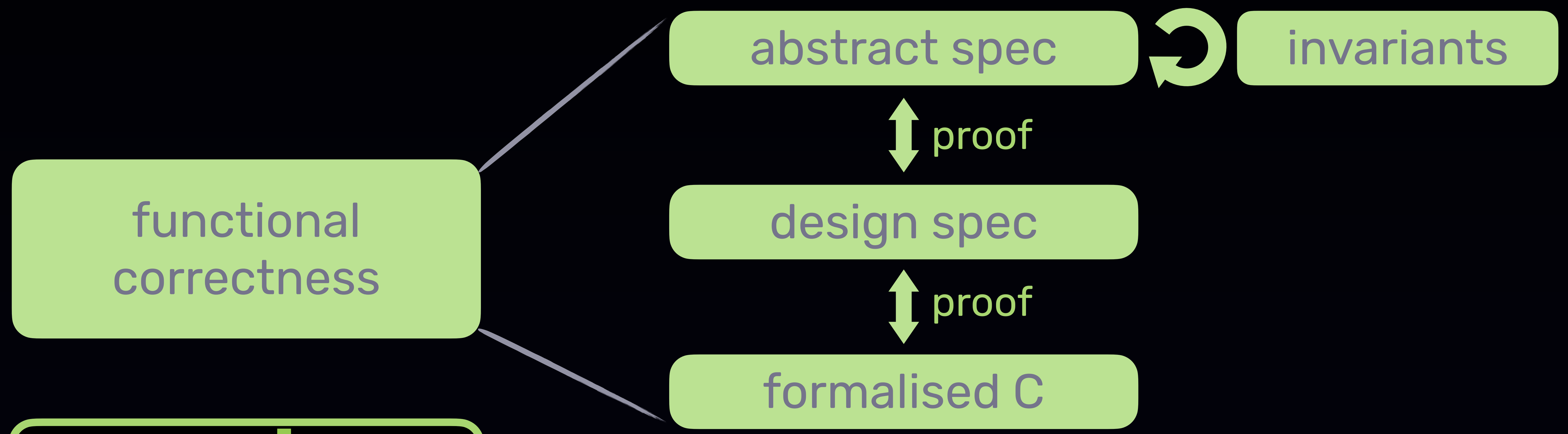
MCS = Mixed-Criticality Systems

UNICORE

MULTICORE

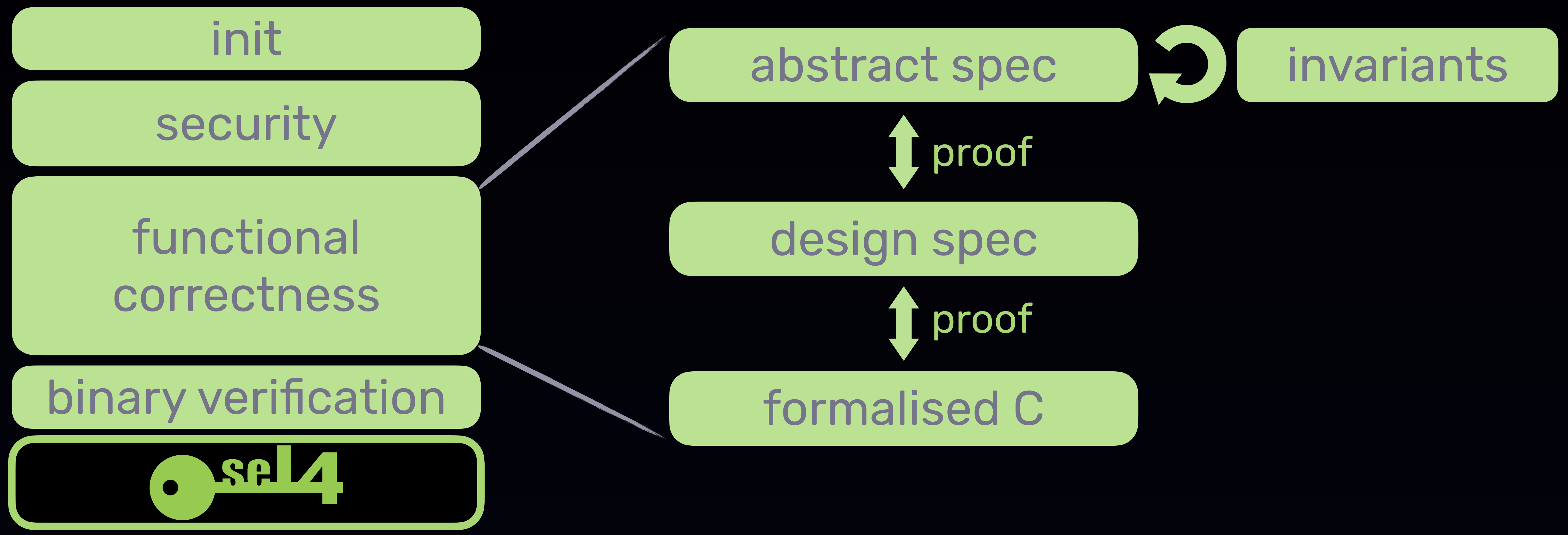
Started as...

"The" seL4 Theorem



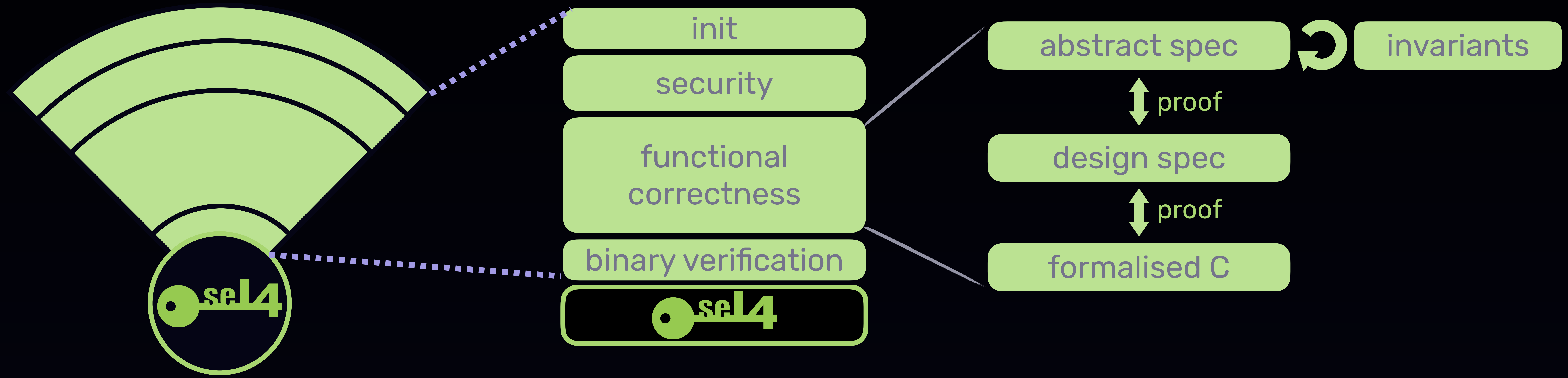
Then...

“The” seL4 Theorem(s)



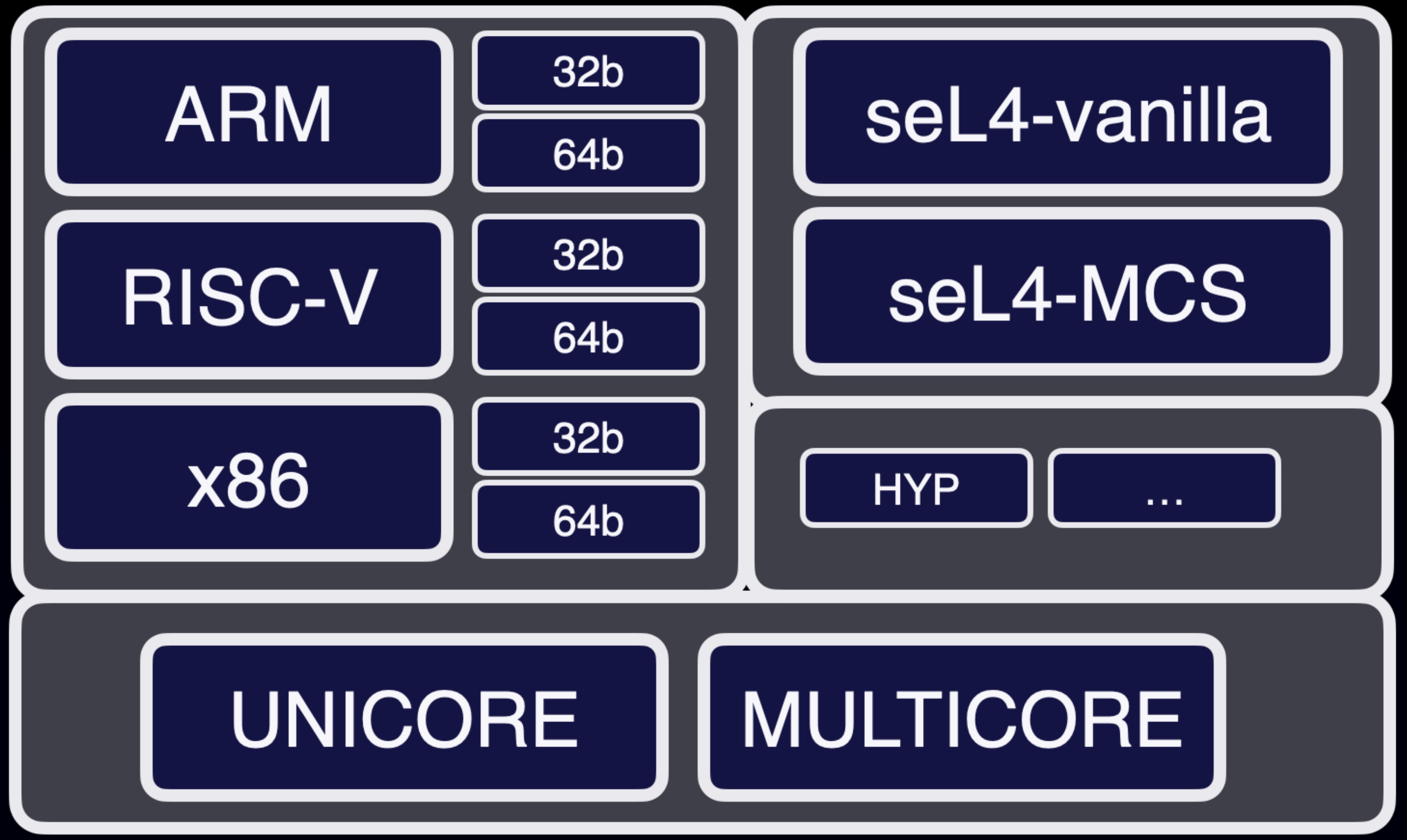
Then...

"The" seL4 Theorem(s)



Then…

“The” seL4 Theorem(s)



different configs

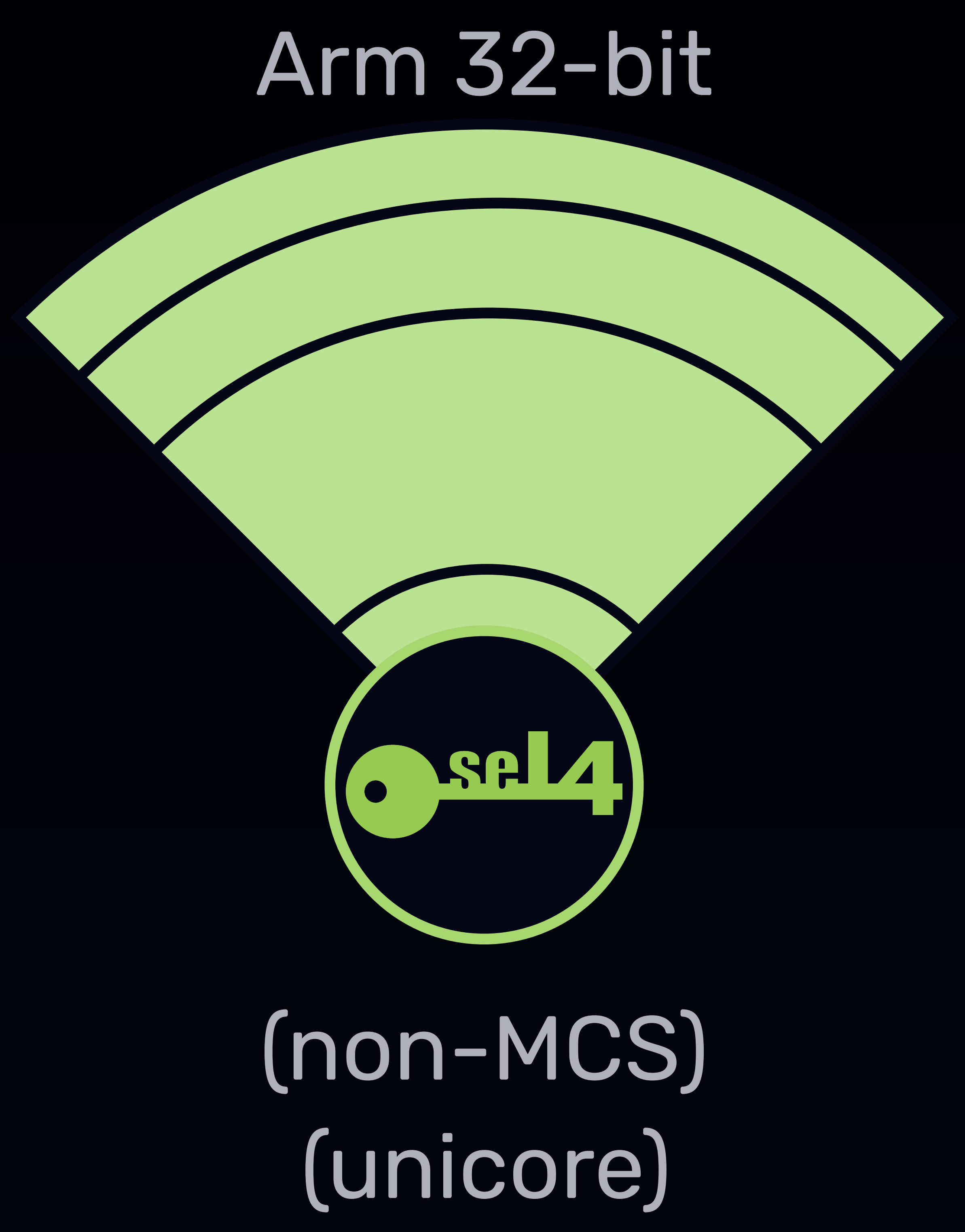
Arm 32-bit
(non-MCS)
(unicore)

different levels

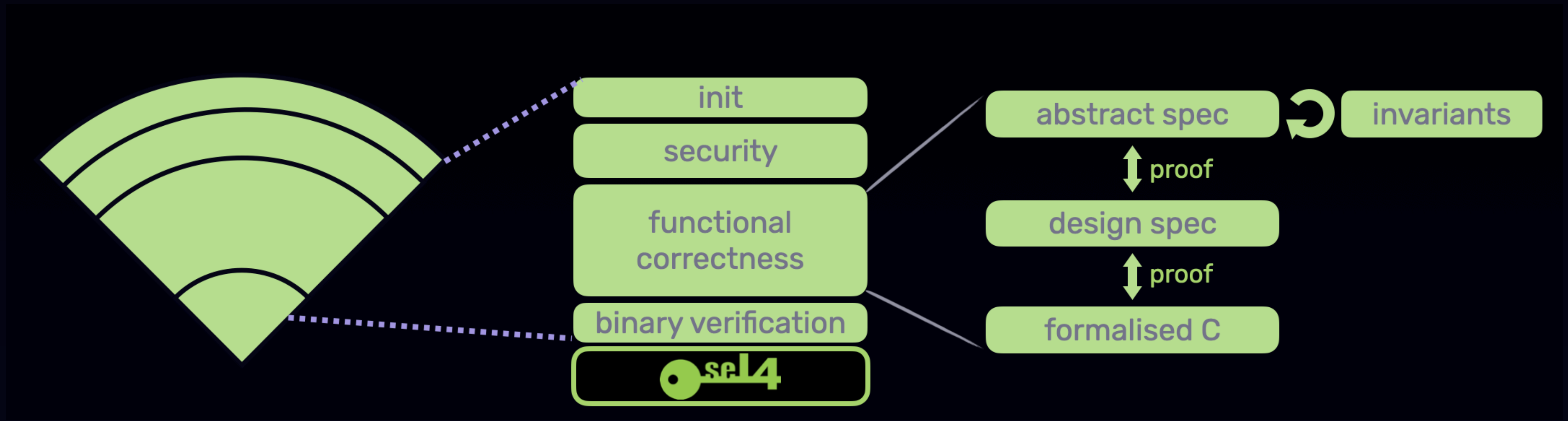
seL4’s formal proofs evolve
with new architectures

seL4’s formal proofs evolve
with new features

Started as...



Arm 32-bit
seL4
(non-MCS)
(unicore)



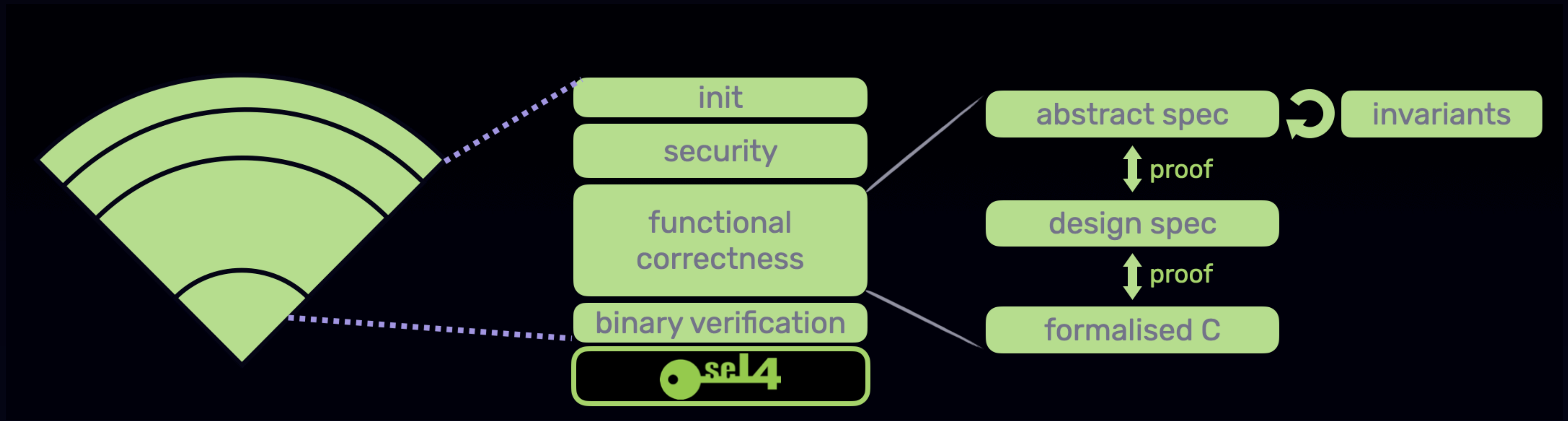
init
security
functional correctness
binary verification
seL4
abstract spec
invariants
proof
design spec
proof
formalised C

Started as...

Arm 32-bit

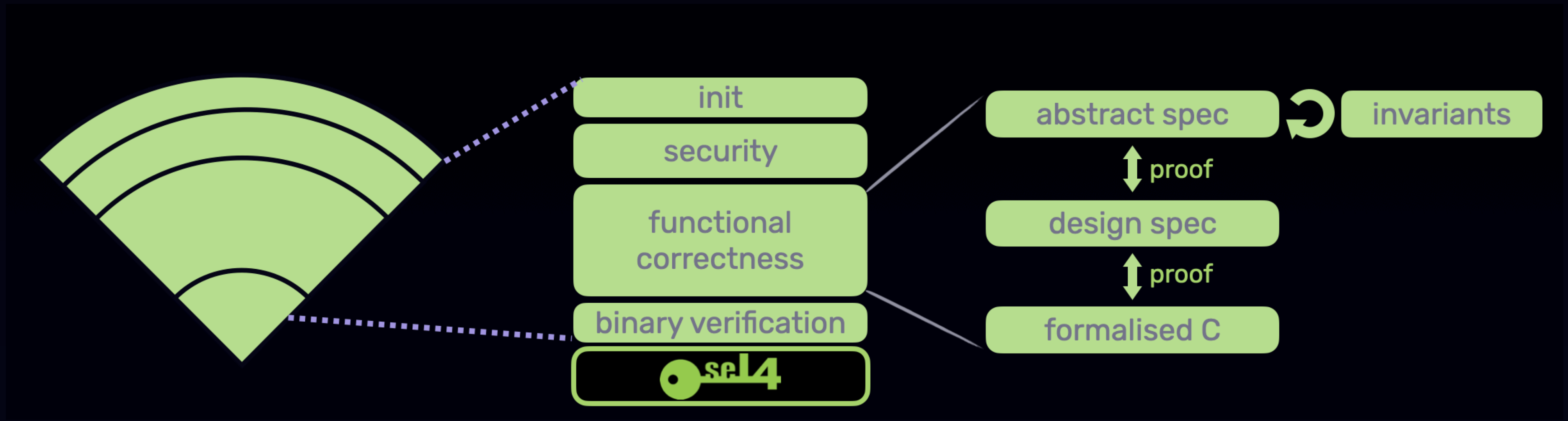
(non-MCS)
(unicore)

- 👍 AOARD, DARPA
- 👍 US Army
- 👍 NICTA

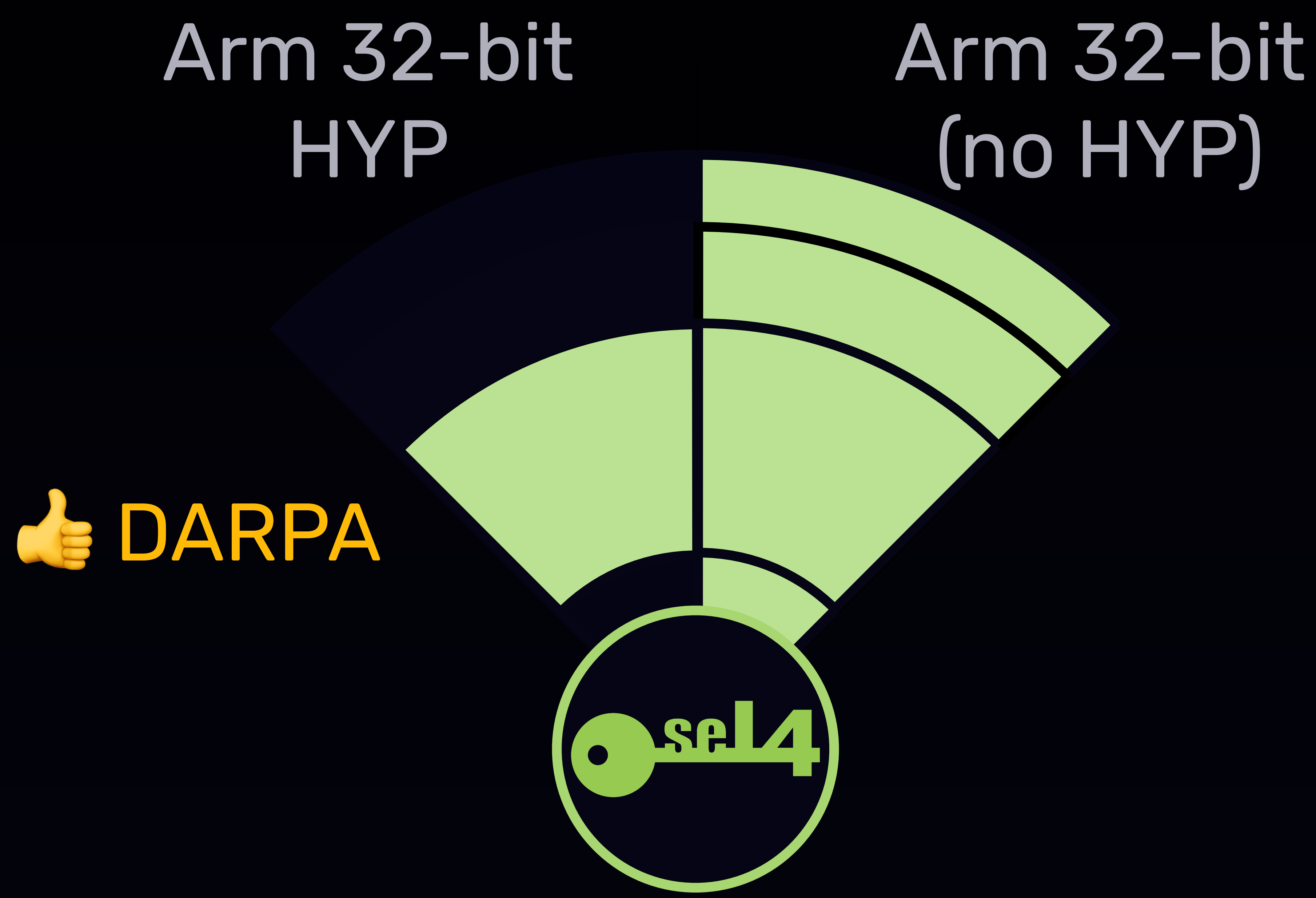


Then...

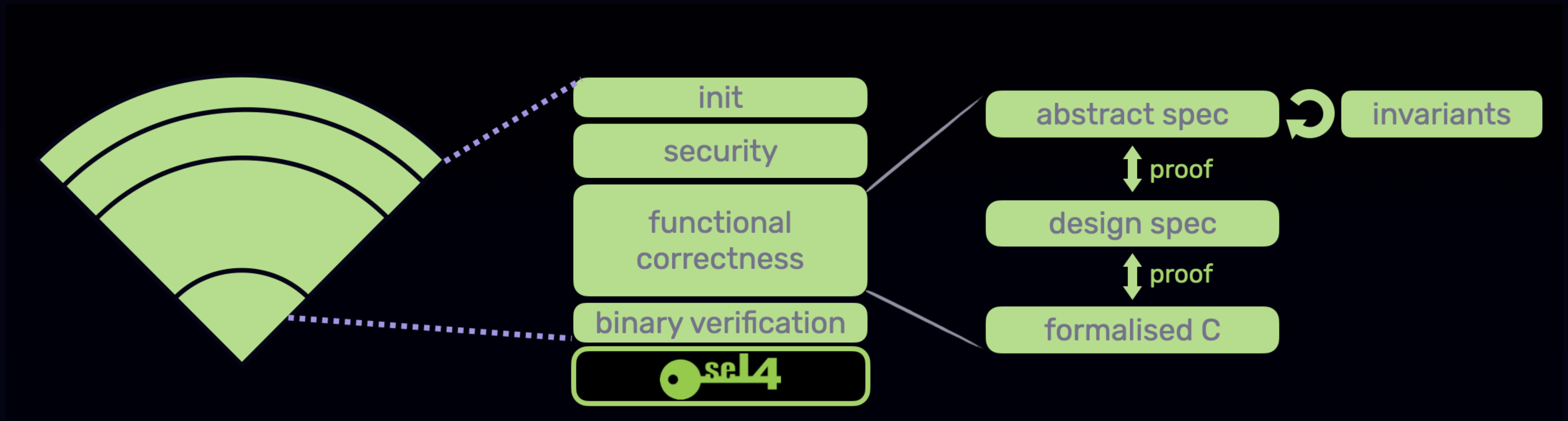

Arm 32-bit
seL4


init
security
functional correctness
binary verification
seL4
abstract spec
invariants
proof
design spec
proof
formalised C

Then...

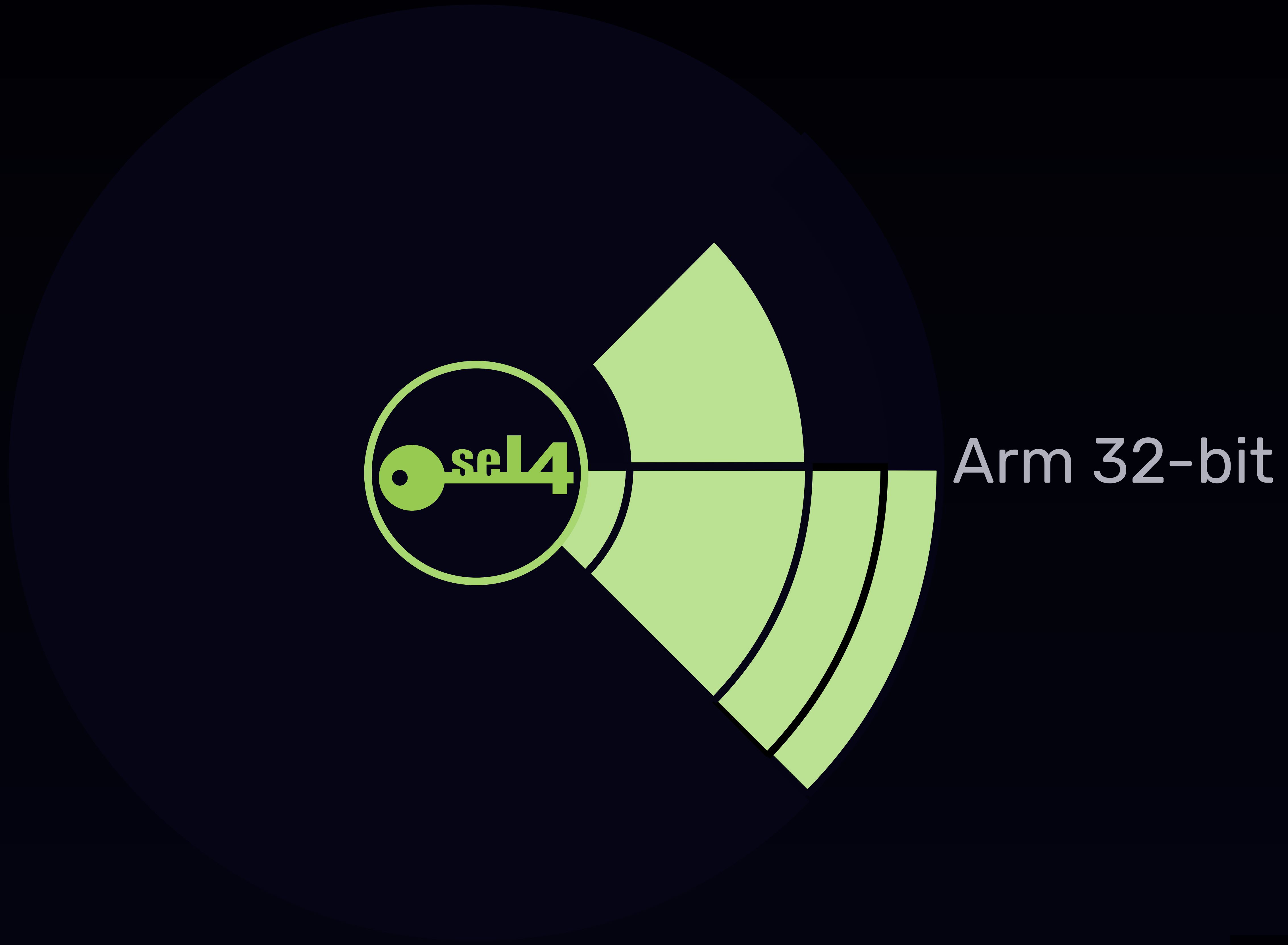


Arm 32-bit
HYP
Arm 32-bit
(no HYP)
👍 DARPA
seL4

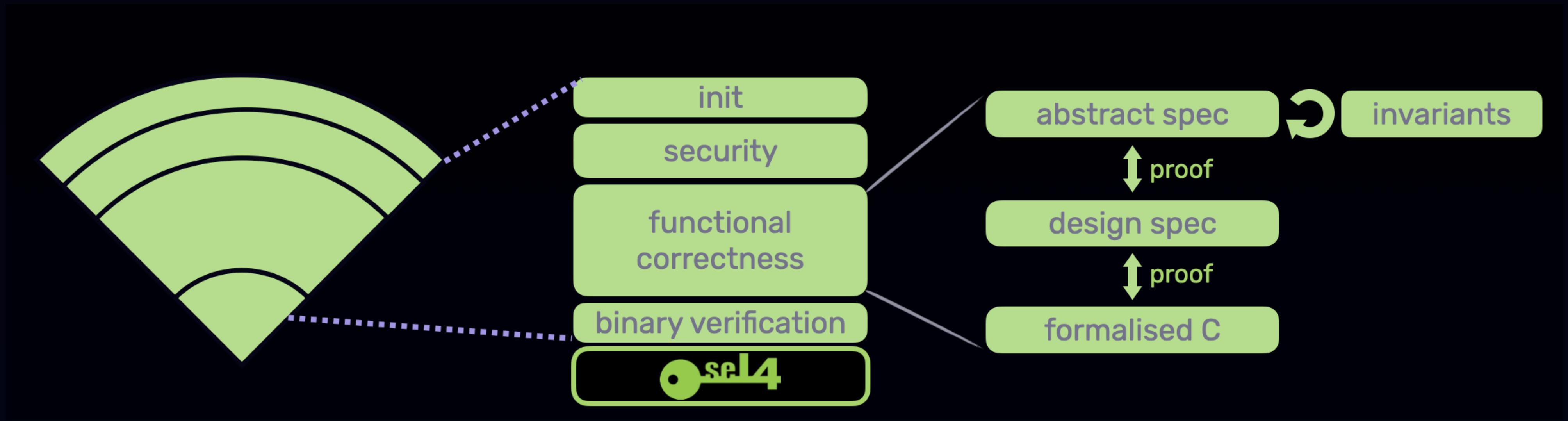


init
security
functional correctness
binary verification
seL4
abstract spec
invariants
proof
design spec
proof
formalised C

Then...

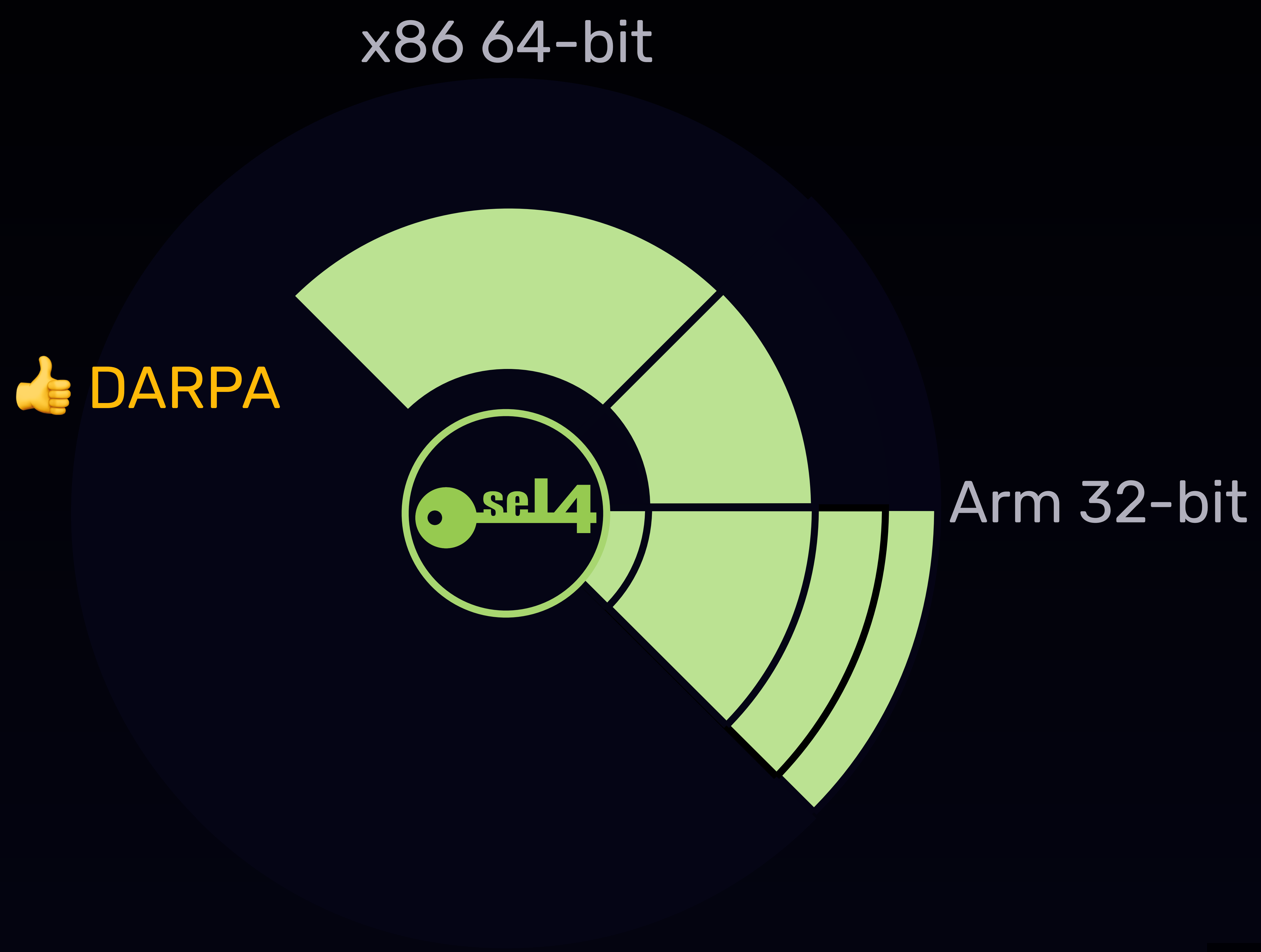


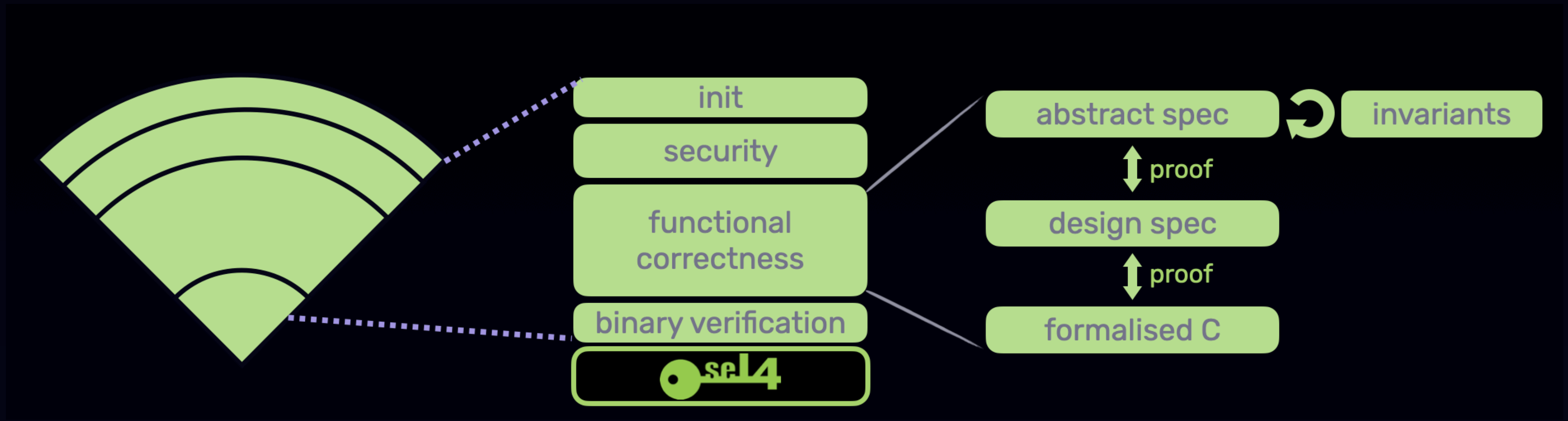
seL4
Arm 32-bit



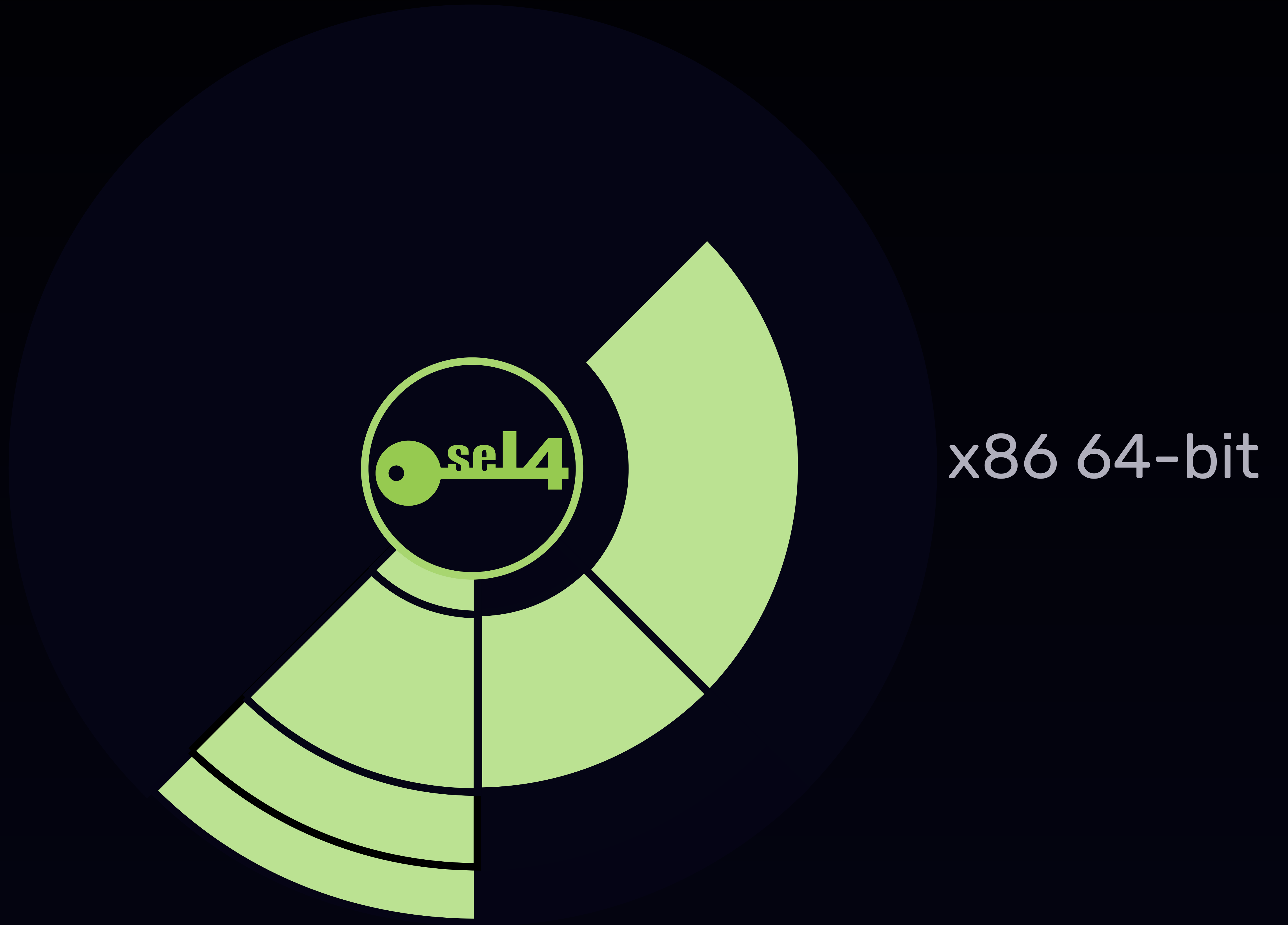
init
security
functional correctness
binary verification
seL4
abstract spec
invariants
proof
design spec
proof
formalised C

Then...

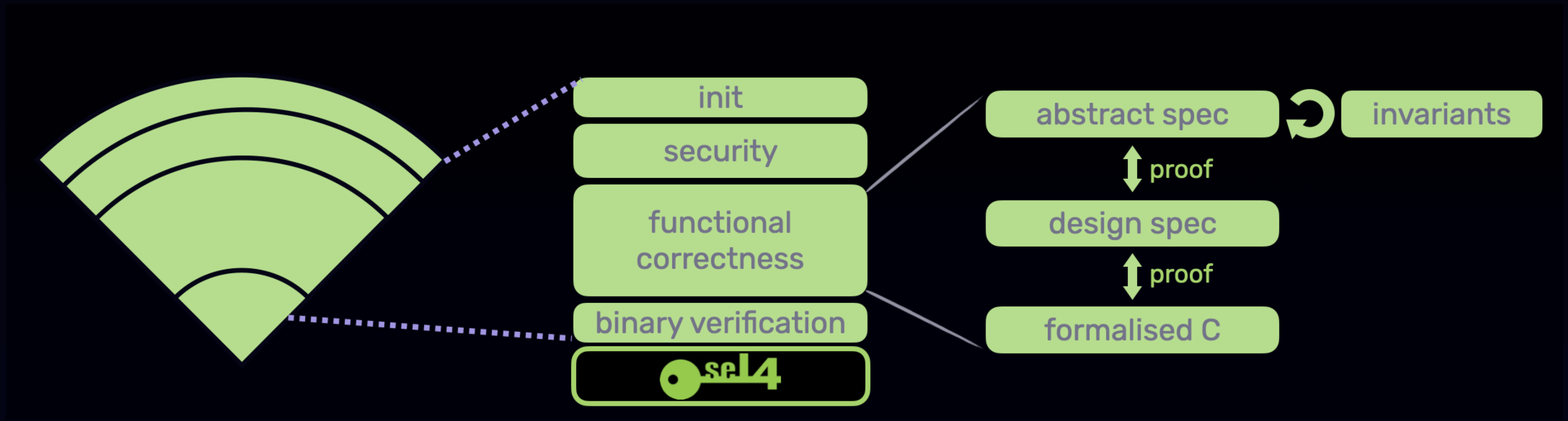

x86 64-bit
👍 DARPA
seL4
Arm 32-bit


init
security
functional correctness
binary verification
seL4
abstract spec
invariants
proof
design spec
proof
formalised C

Then...

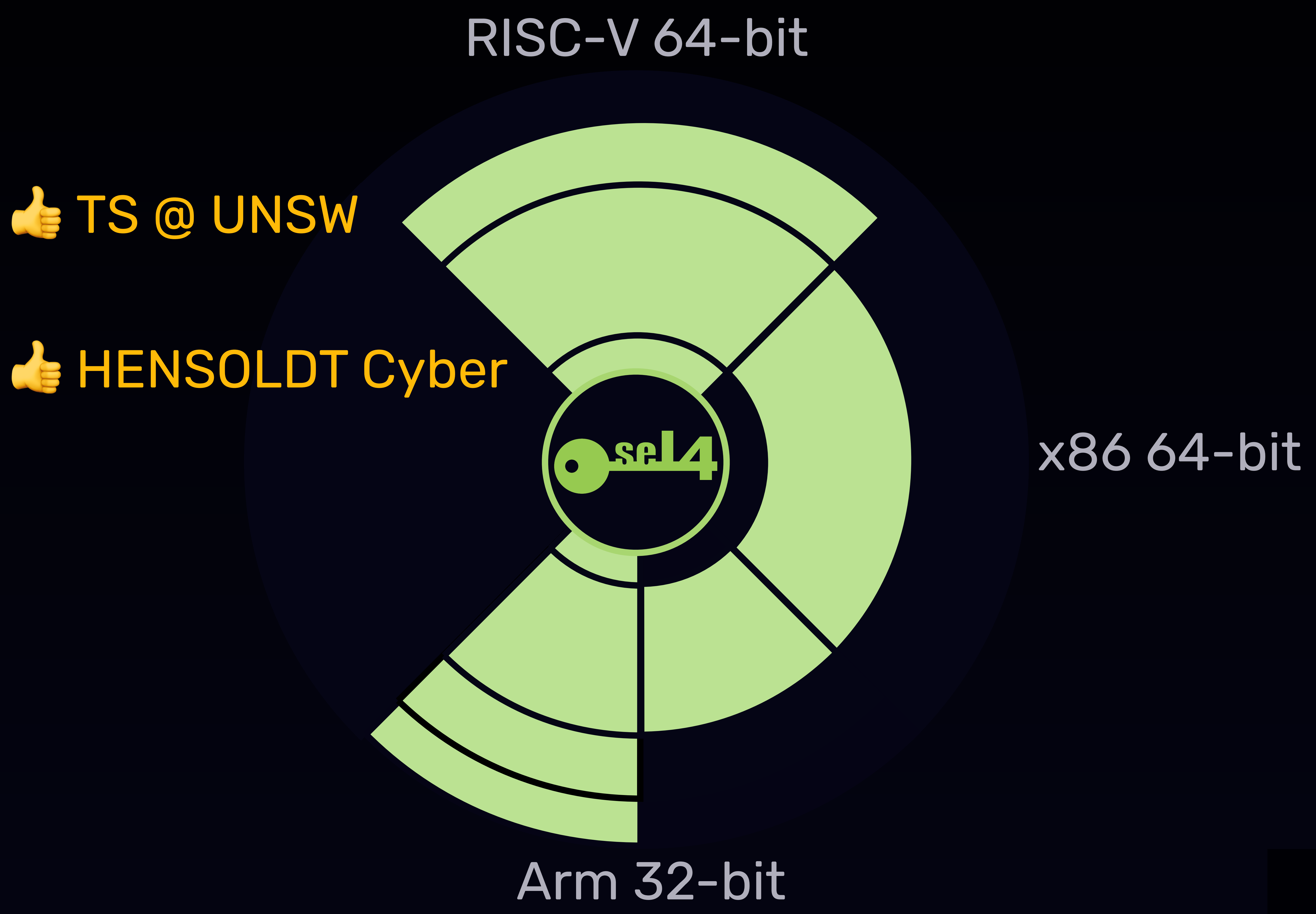


x86 64-bit
Arm 32-bit

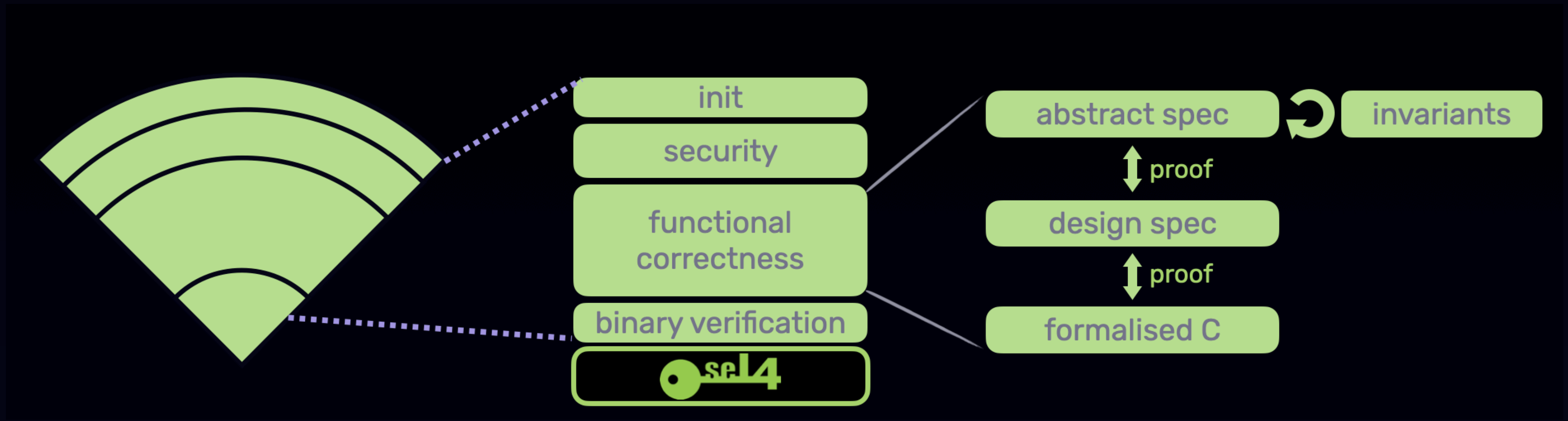


init
security
functional correctness
binary verification
abstract spec
invariants
proof
design spec
proof
formalised C

Then...

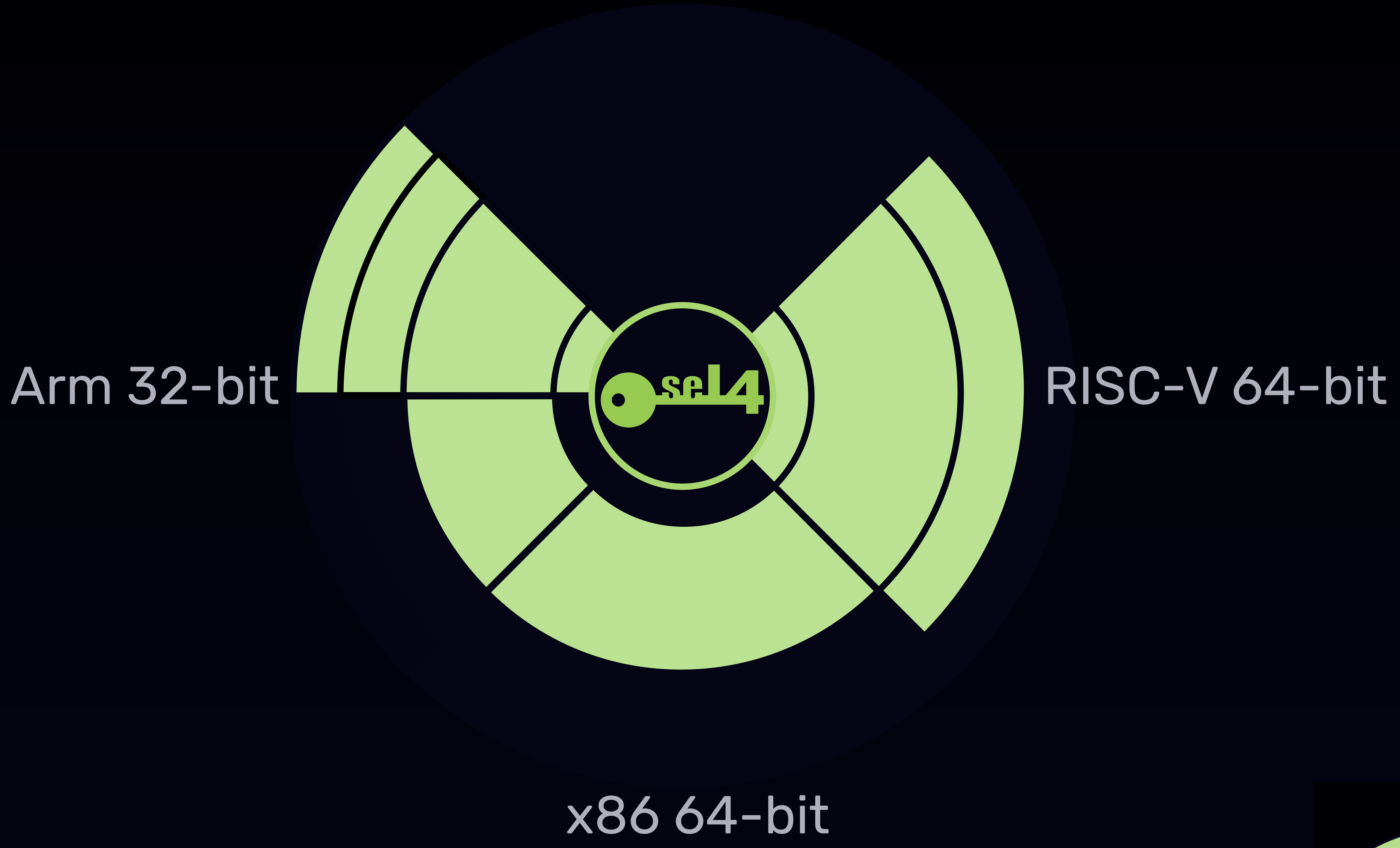


RISC-V 64-bit
👍 TS @ UNSW
👍 HENSOLDT Cyber
seL4
x86 64-bit
Arm 32-bit

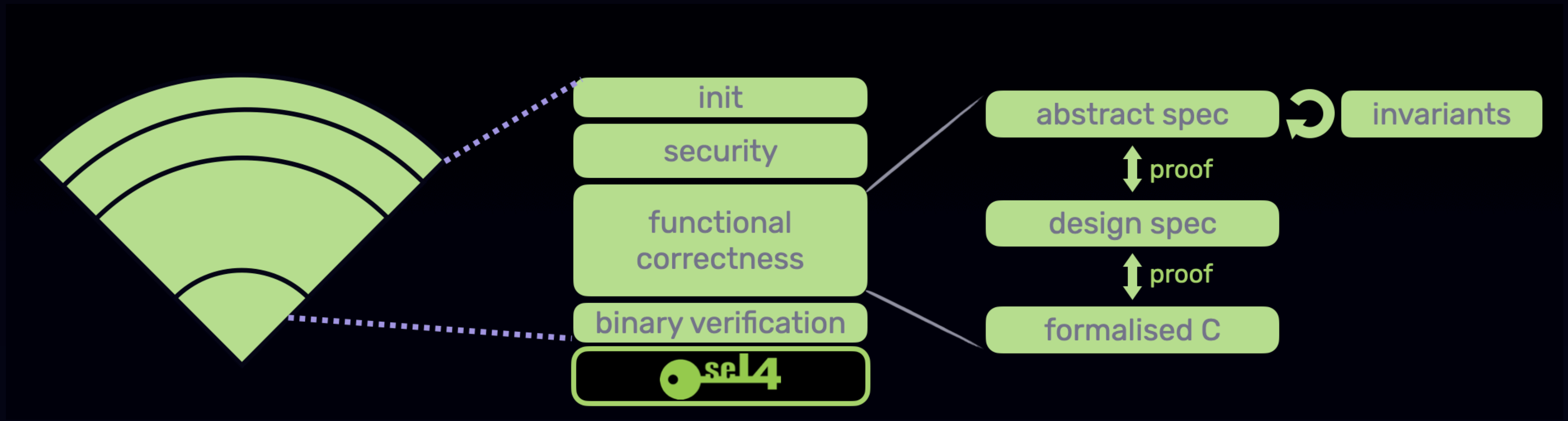


init
security
functional correctness
binary verification
seL4
abstract spec
invariants
proof
design spec
proof
formalised C

Then…

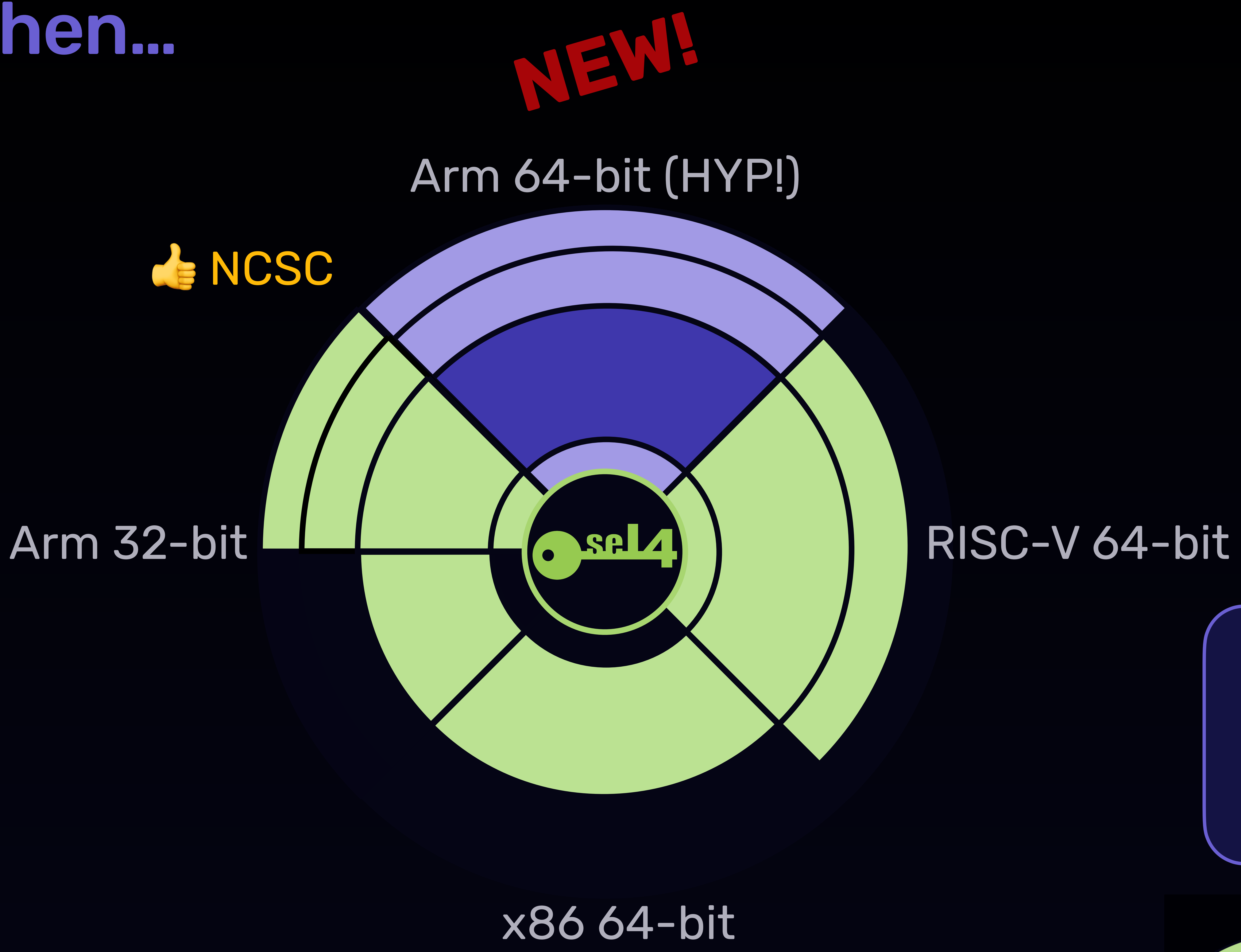


Arm 32-bit
seL4
RISC-V 64-bit
x86 64-bit



init
security
functional correctness
binary verification
seL4
abstract spec
invariants
proof
design spec
proof
formalised C

Then...

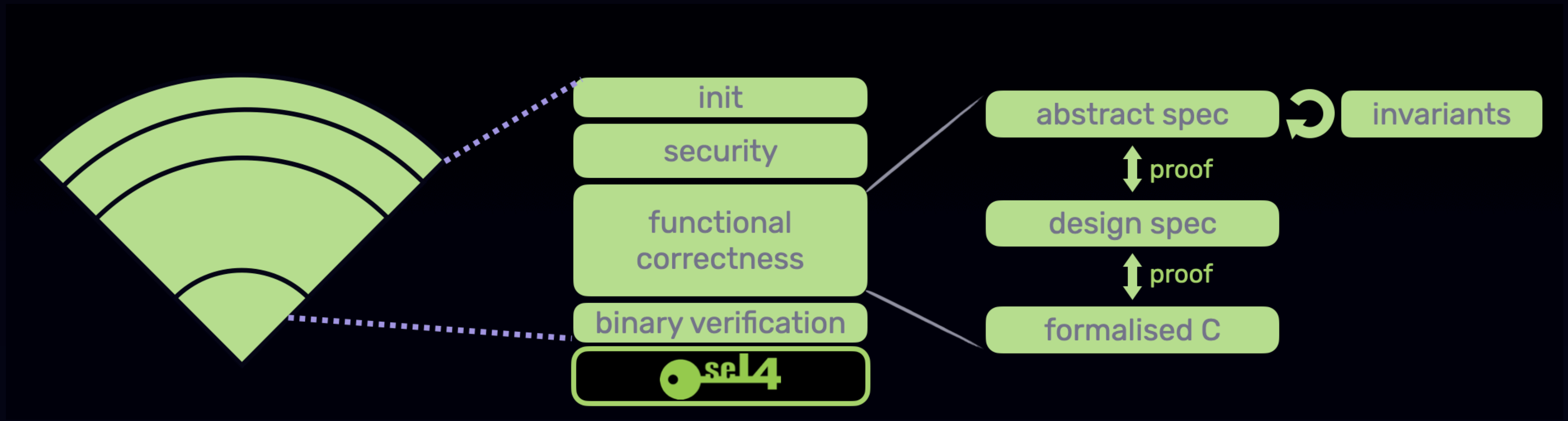


seL4 proofs

Done
Ongoing
Future

(non-MCS, unicore)

seL4's formal proofs evolve with new architectures



seL4’s formal proofs evolve
with new architectures

seL4’s formal proofs evolve
with new features

The proofs have evolved with new features over the years

Two examples:

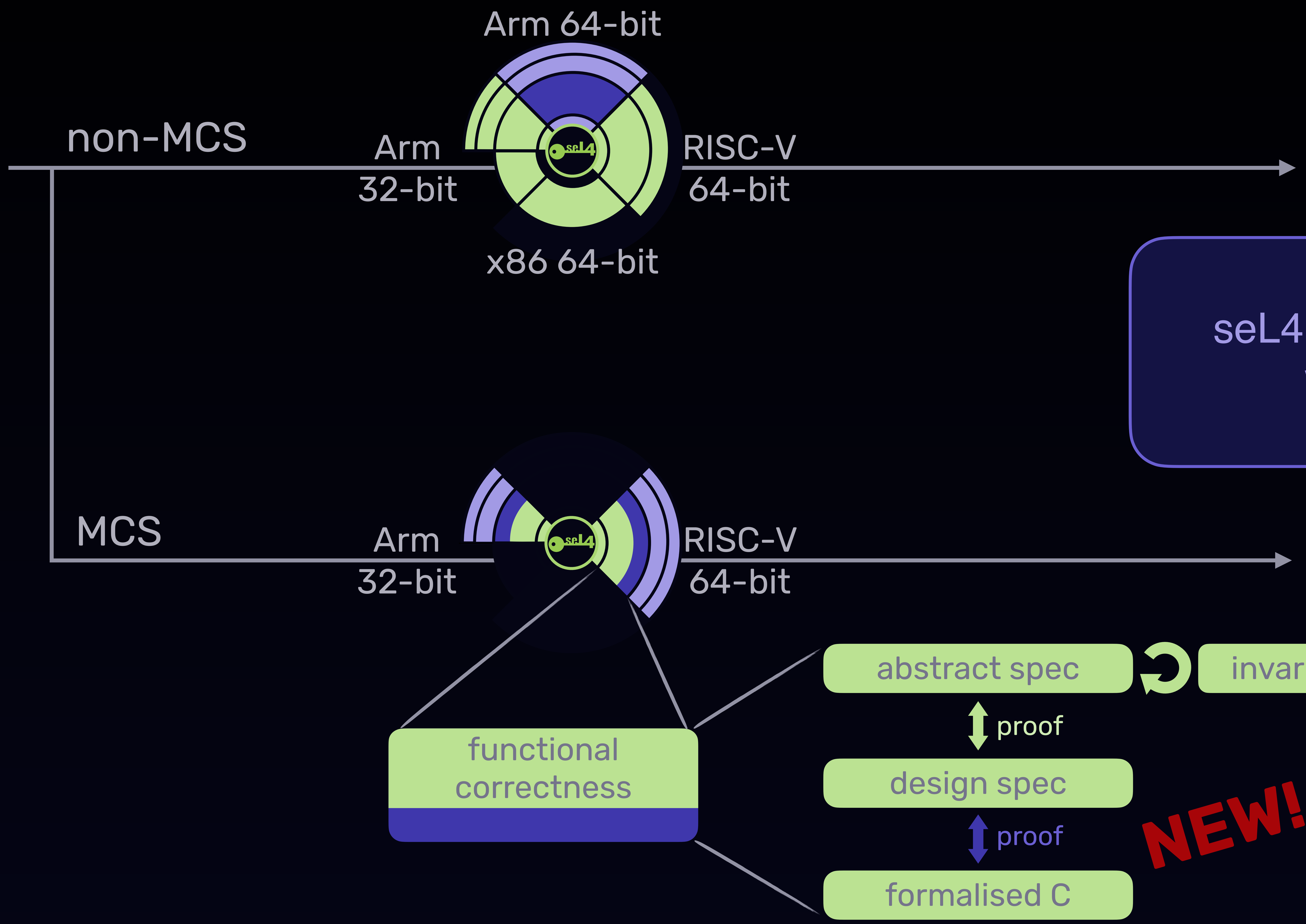
- bound notification endpoints
- bitfield scheduler optimisation

MCS is different:

- Mixed-Criticality Systems
- time as a resource
- large, invasive change

Big Feature: Mixed-Criticality Systems

Verification of multiple configs in parallel



seL4's formal proofs evolve with new features

👍 HENSOLDT Cyber
👍 DHS
👍 seL4 Foundation

Overview

#2
Deliver it to the world:
true trustworthiness for critical software

Opportunities:

- used in products where it matters
- set a standard

Challenges:

- port verification to new platforms
- port verification to new features

More challenges:

- millions lines of proofs
- duplication

Some solutions

Some solutions

Abstraction, Parametricity, Modularity

- Example: split proof into arch-specific and generic part
  - Generic part is a parametric module
  - Has been effective, but used only for part of proof
  - More of this in development
- Example: parametric page table structures in seL4/RISC-V
  - Regular structure
  - Much faster proof completion

Overview

#3
Keep it live:
for today and tomorrow

Challenges

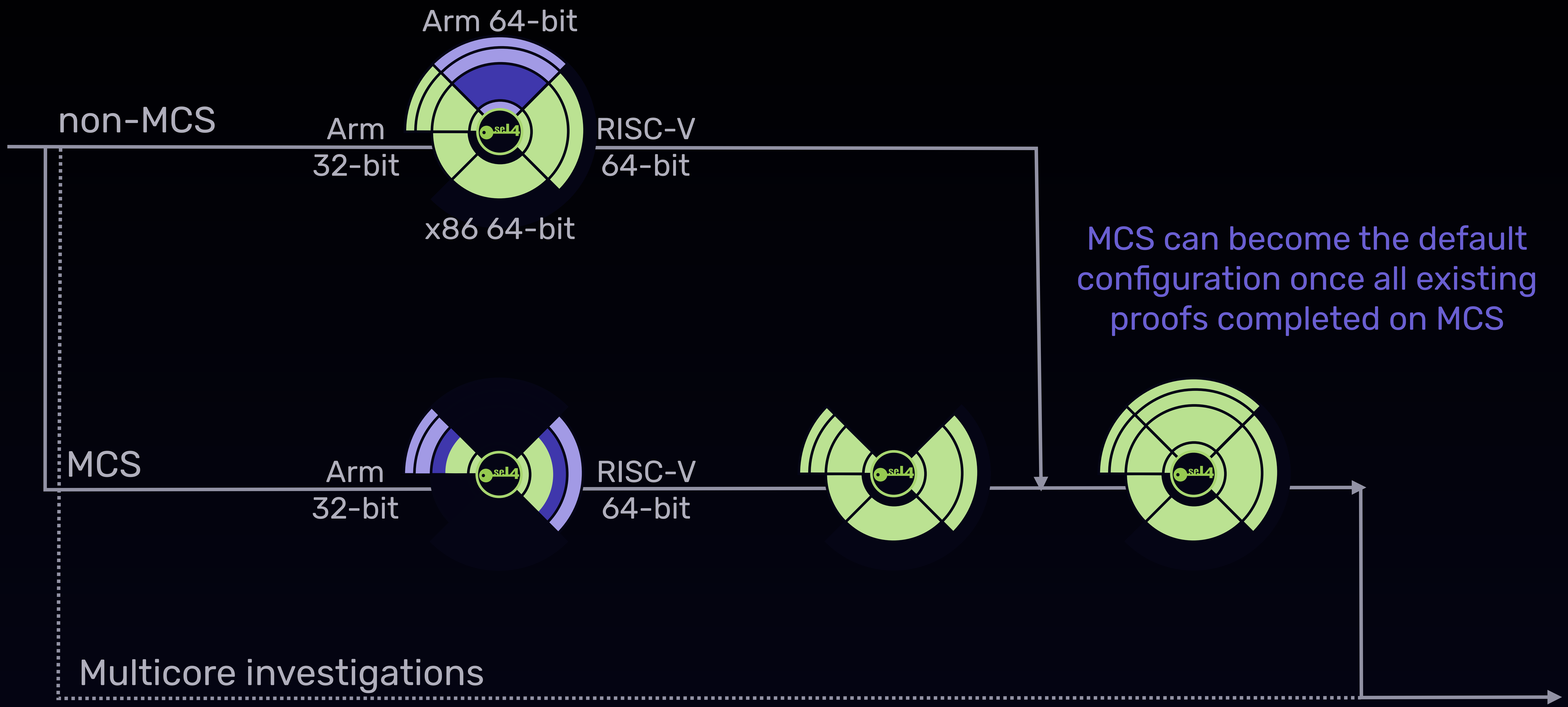
Challenges:

- maintenance
- tech debt from deadlines
- application of improvements blocked

Solutions:

- robustness, automation, proof engineering
- consolidation

Roadmap



Eventually, seL4 verified on multicore, with unicore as an instance

Opportunities to Reflect

What would we have done differently, now that we know?

Opportunities to Reflect

What would we have done differently, now that we know?

Probably not much...

I want it all. And I want it now.

“Doing arch-split too early would have killed the project”

Photo by Jim Tegman on Unsplash

“Things could have been done differently *if* we had sorted out the right solution already”

I want it all. And I want it now.

“a trade-off, everything is”

“Doing arch-split too early would have killed the project”


Photo by Jim Tegman on Unsplash

“Things could have been done differently *if* we had sorted out the right solution already”

#1
Make a dream come true:
verified, performant kernel

#2
Deliver it to the world:
true trustworthiness for critical software

#3
Keep it live:
for today and tomorrow

Conclusion

Path to a bigger journey

seL4's formal proofs were a breakthrough in formal software verification

Success creates interest, interest pushes evolution

Formal proofs must evolve as the code evolves

Proofcraft is committed to keep this evolution alive

Proofcraft

https://proofcraft.systems