Verifying Hardware Security Modules with Information-Preserving Refinement

Anish Athalye, M. Frans Kaashoek, and Nickolai Zeldovich

OSDI ‘22

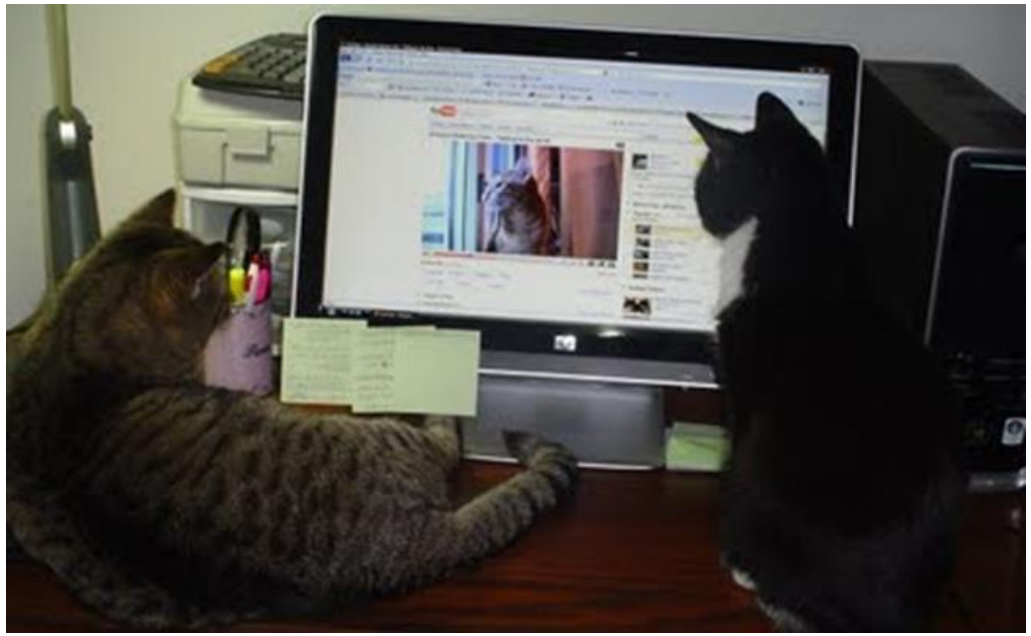
Trustworthy Systems

Trustworthy Systems

Systems that behave ‘as expected’

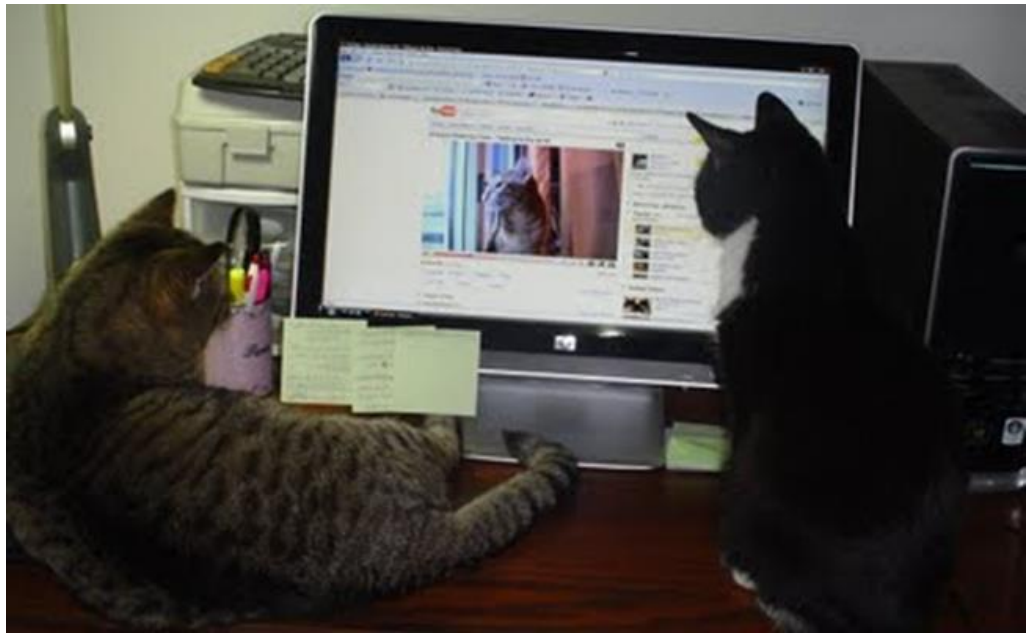
Trustworthy Systems

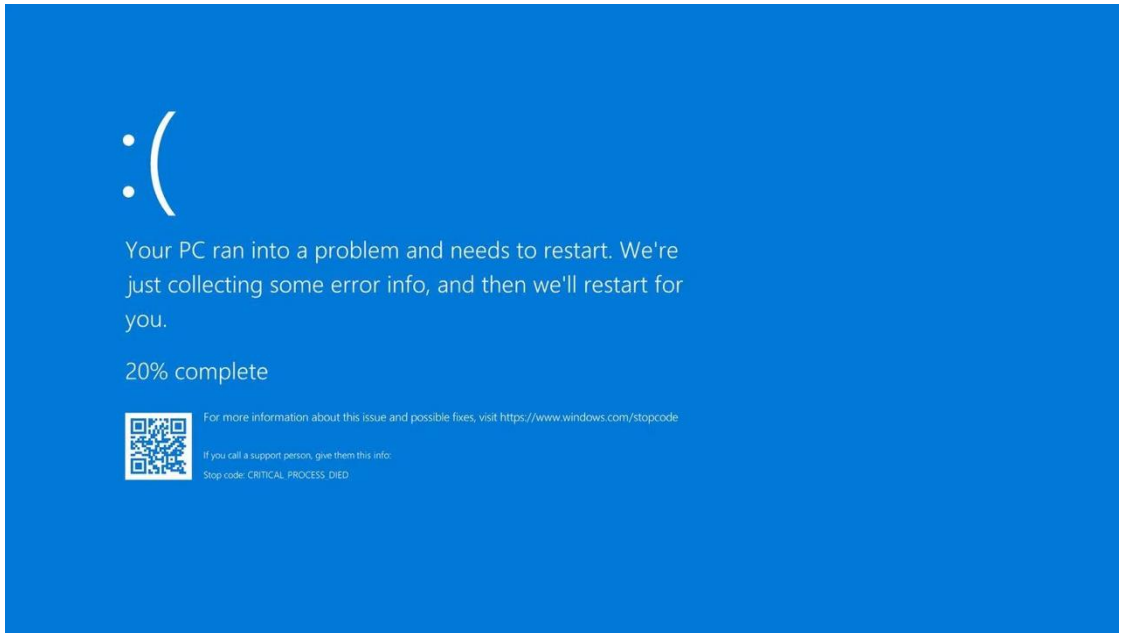
Systems that behave ‘as expected’



Trustworthy Systems

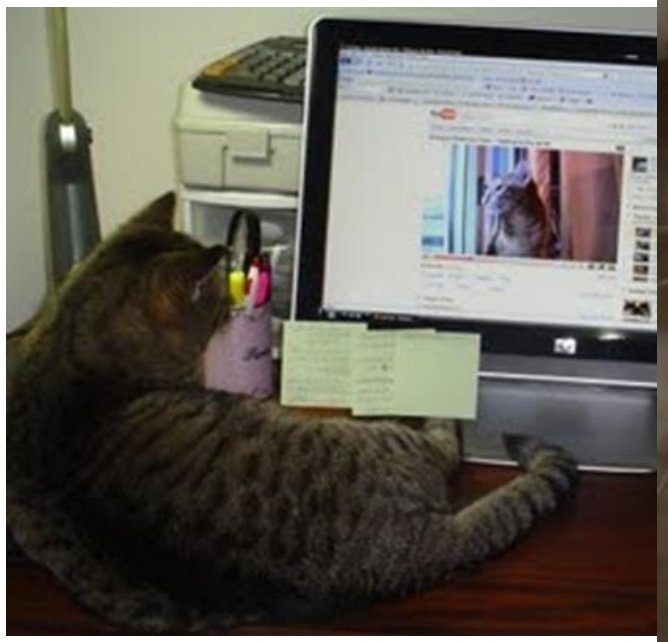
Systems that behave ‘as expected’



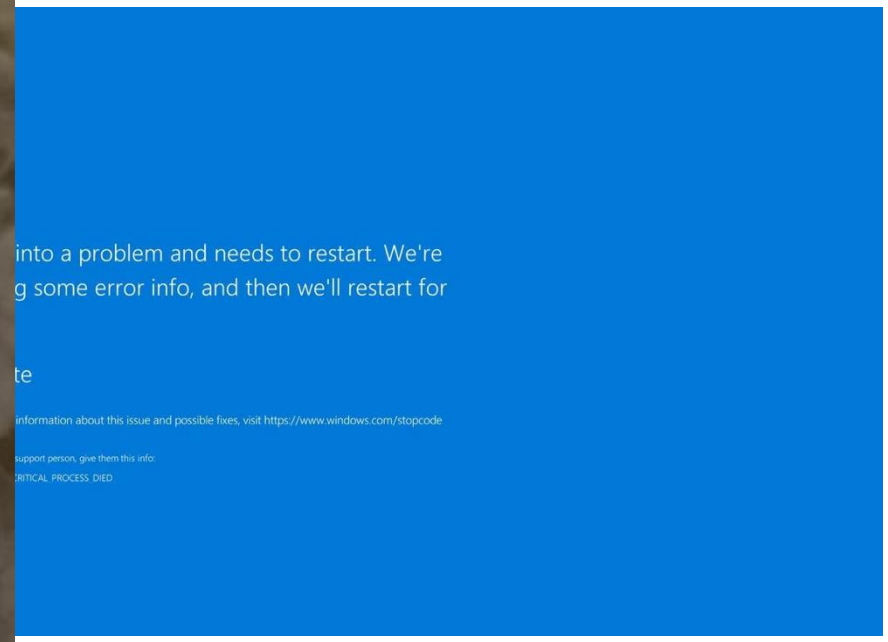


Trustworthy Systems

Systems that behave ‘as expected’

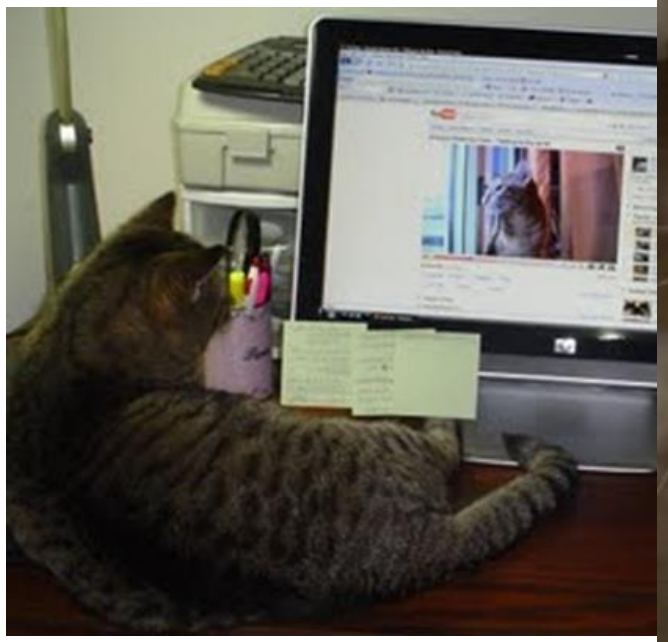




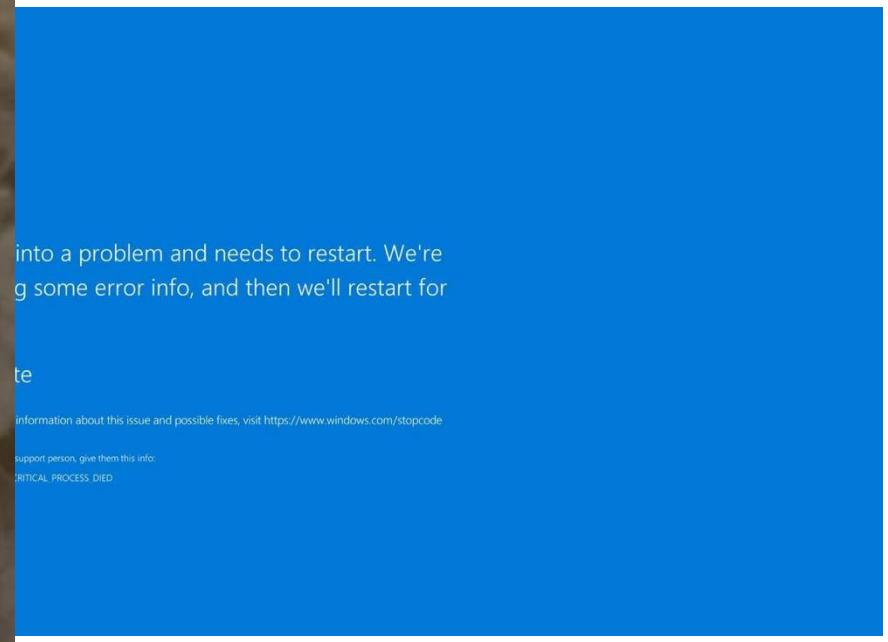


Trustworthy Systems

Systems that behave ‘as expected’







Need unambiguous specifications to reflect our ‘expectations’

Traditional System Specifications

Traditional System Specifications

Mathematical functions

Traditional System Specifications

Mathematical functions

$$\mathbf{prime}(n) \triangleq \forall \mathbf{k} \in \mathbf{N}: (\mathbf{0} < \mathbf{k} < n) \Rightarrow (\mathbf{n} \,\%\, k) \neq \mathbf{0}$$

Traditional System Specifications

Mathematical functions

$$\mathbf{prime}(n) \triangleq \forall \mathbf{k} \in \mathbf{N}: (\mathbf{0} < \mathbf{k} < n) \Rightarrow (\mathbf{n} \,\%\, k) \neq \mathbf{0}$$

```
bool prime(unsigned n)
{
    if (n == 0 || n == 1)
        return false;

    for (int i=2; i<n; ++i)
        if (n % i == 0)
           return false;

    return true;
}
```

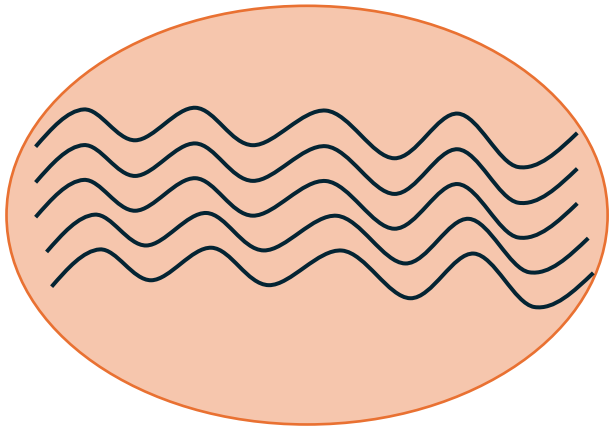
Traditional System Specifications

Traditional System Specifications

More generally, ‘allowed executions’

Traditional System Specifications

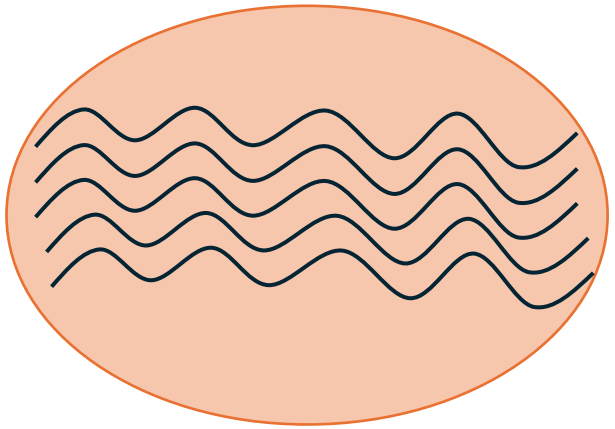
More generally, ‘allowed executions’



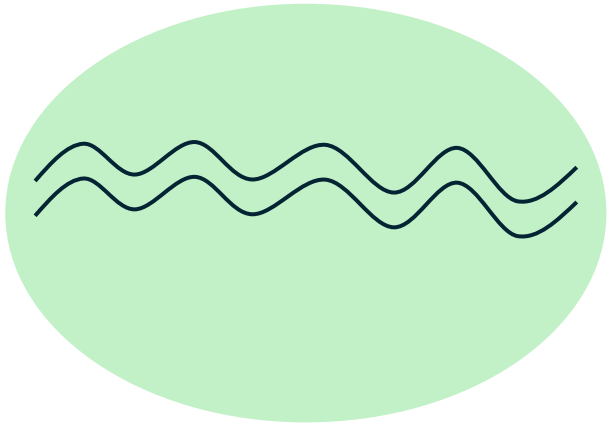
Specification

Traditional System Specifications

More generally, ‘allowed executions’



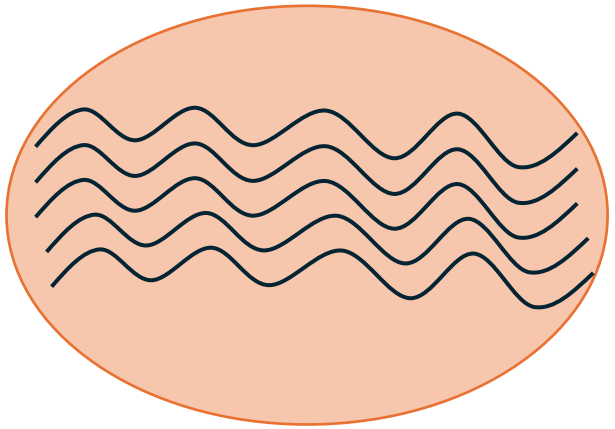
Specification



System Implementation

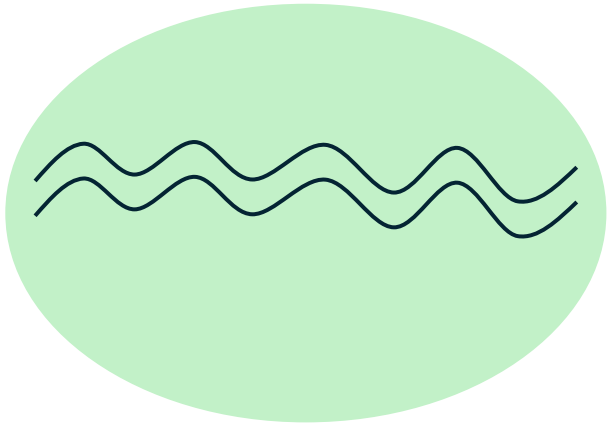
Traditional System Specifications

More generally, ‘allowed executions’



Specification

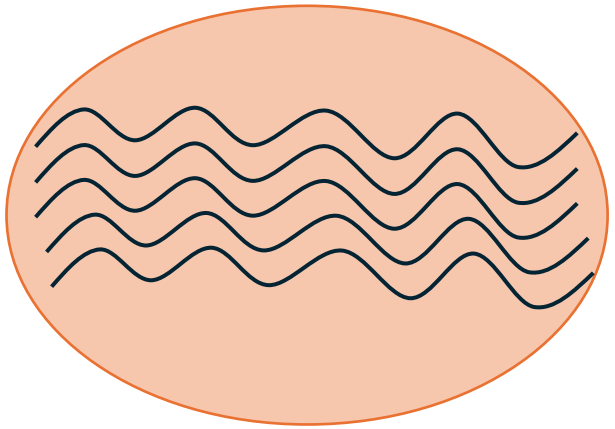
$\supseteq$



System Implementation

Traditional System Specifications

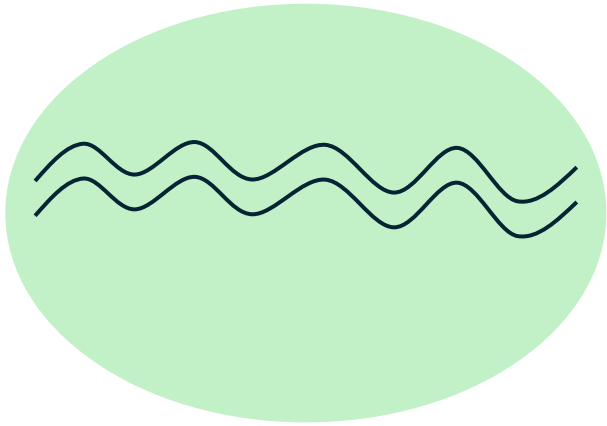
More generally, ‘allowed executions’



Specification

$\supseteq$

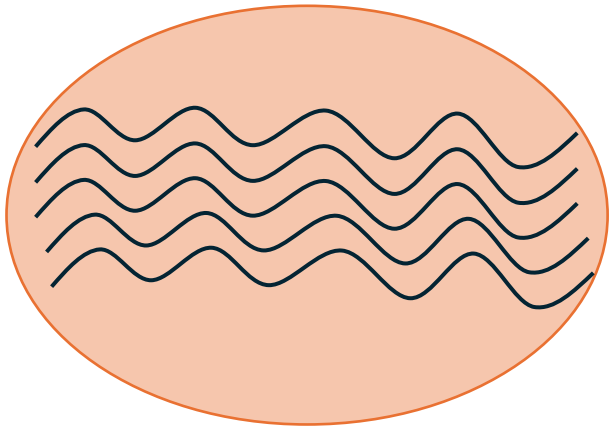
Refinement



System Implementation

Traditional System Specifications

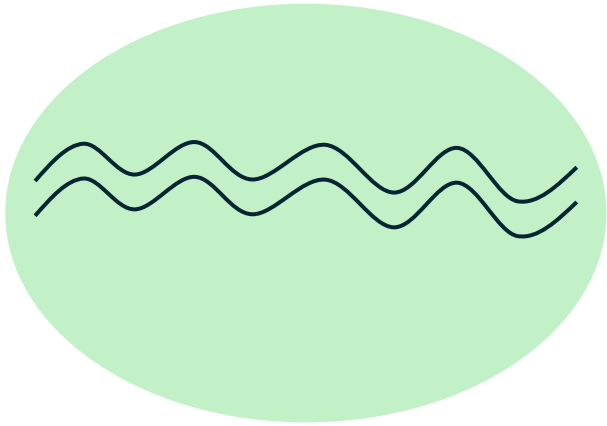
More generally, ‘allowed executions’



Specification

$\supseteq$

Refinement



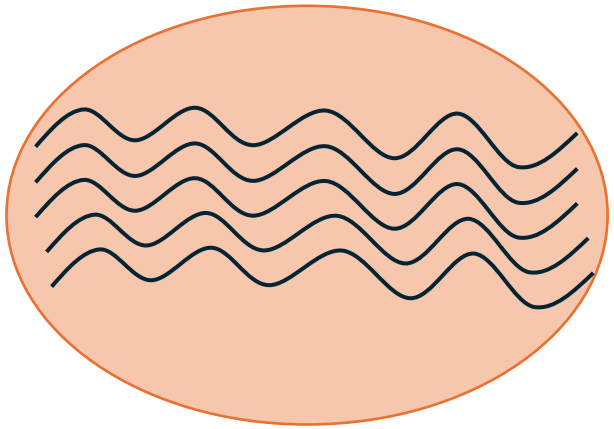
System Implementation

```
unsigned char x = 12;
{ unsigned char x = x; }
```

Here, the initialization of the second **x** has undefined behavior, because the initializer accesses the second **x** outside its lifetime (6.7.3). — *end example*]

Traditional System Specifications

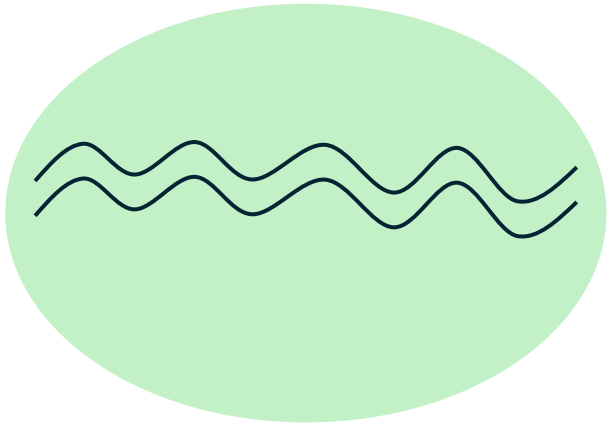
More generally, ‘allowed executions’



Specification

$\supseteq$

Refinement



System Implementation

```
unsigned char x = 12;
{ unsigned char x = x; }
```

Here, the initialization of the second **x** has undefined behavior, because the initializer accesses t
lifetime (6.7.3). — *end example*]



Traditional Specifications and Security

Traditional Specifications and Security



Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 3 (3A, 3B, 3C & 3D): System Programming Guide

NOTE: The Intel 64 and IA-32 Architectures Software Developer's Manual consists of four volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-Z*, Order Number 325383; *System Programming Guide*, Order Number 325384; *Model-Specific Registers*, Order Number 335592. Refer to all four volumes when evaluating your design needs.

Traditional Specifications and Security

Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 3 (3A, 3B, 3C & 3D): System Programming Guide

NOTE: The Intel 64 and IA-32 Architectures Software Developer's Manual consists of four volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-Z*, Order Number 325383; *System Programming Guide*, Order Number 325384; *Model-Specific Registers*, Order Number 335592. Refer to all four volumes when evaluating your design needs.



Traditional Specifications and Security





Intel® 64 and IA-32 Architectures
Software Developer's Manual

Volume 3 (3A, 3B, 3C & 3D):
System Programming Guide

NOTE: The Intel 64 and IA-32 Architectures Software Developer's Manual consists of four volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-Z*, Order Number 325383; *System Programming Guide*, Order Number 325384; *Model-Specific Registers*, Order Number 335592. Refer to all four volumes when evaluating your design needs.

Traditional Specifications and Security







Depends on secret?

Traditional Specifications and Security







Depends on secret?

No



Traditional Specifications and Security







Depends on secret?

No



Yes



Traditional Specifications and Security





Depends on secret?

No





Yes



Traditional Specifications and Security







Depends on secret?

No

Yes











Traditional Specifications and Security



NOTE: The
Basic Arch
System Pr
Refer to a



ret?

Yes



Hyperproperties





Hyperproperties

This is well known





Hyperproperties





This is well known

‘Traditional’ specifications – Trace properties

Hyperproperties





This is well known

‘Traditional’ specifications – Trace properties

All specifications – Hyperproperties

Hyperproperties





This is well known

‘Traditional’ specifications – Trace properties
All specifications – Hyperproperties

Very general

Hyperproperties





This is well known

‘Traditional’ specifications – Trace properties

All specifications – Hyperproperties

Very general

Too general…

Hyperproperties





This is well known

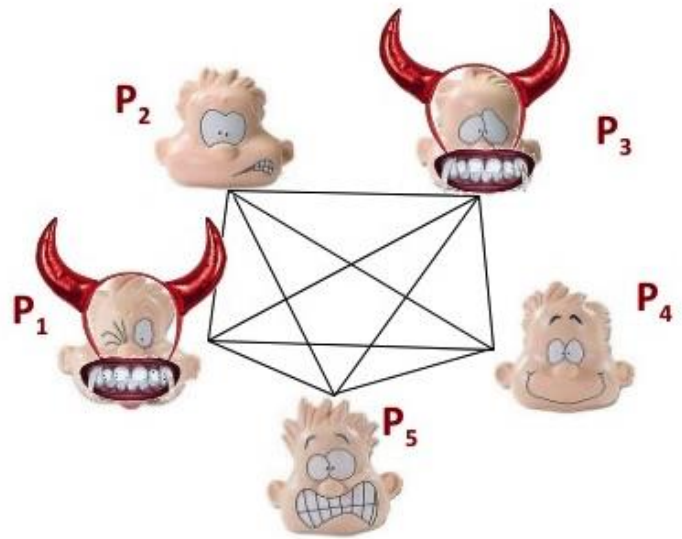
‘Traditional’ specifications – Trace properties
All specifications – Hyperproperties

Very general
Too general…

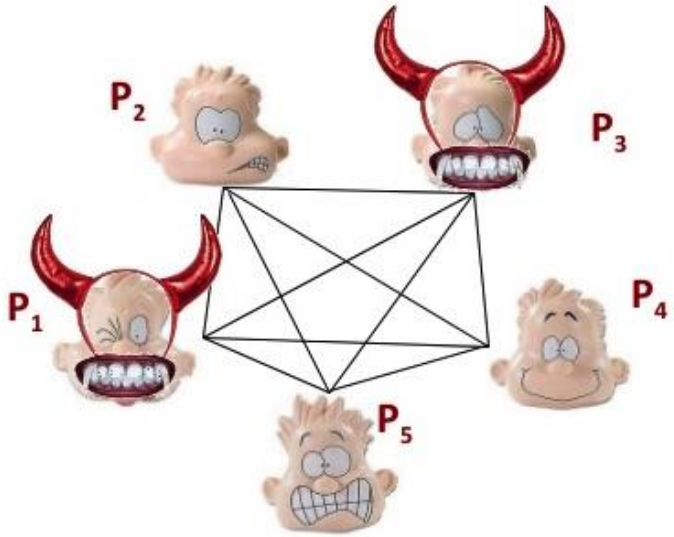
More restricted/useful security definition?

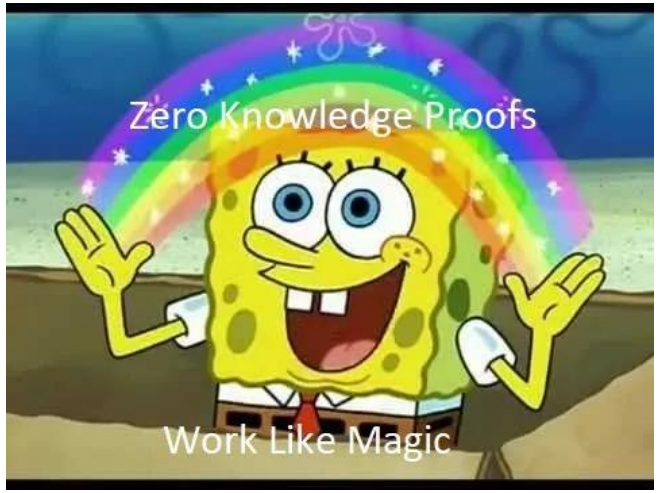
Cryptography has an approach...

Cryptography has an approach…

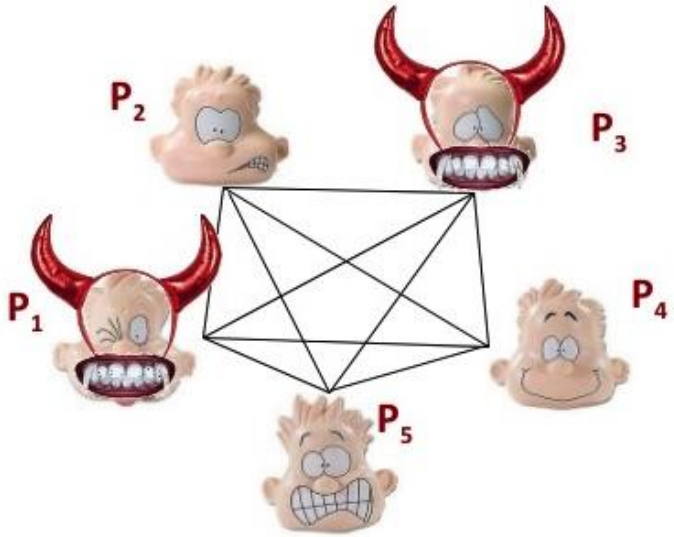


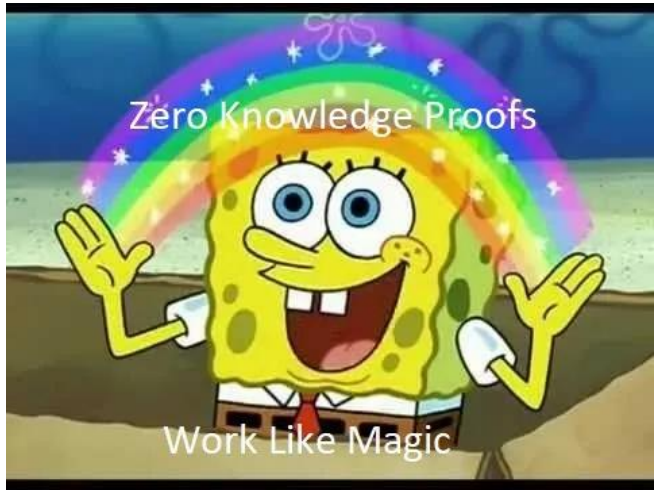
Cryptography has an approach...





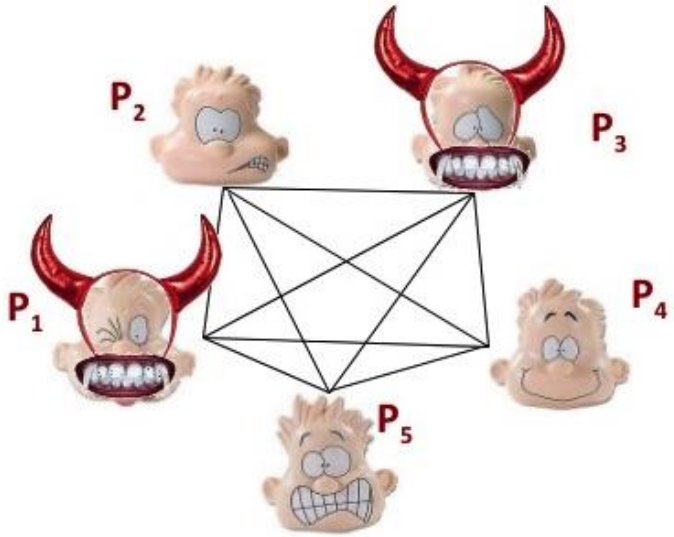
Cryptography has an approach...

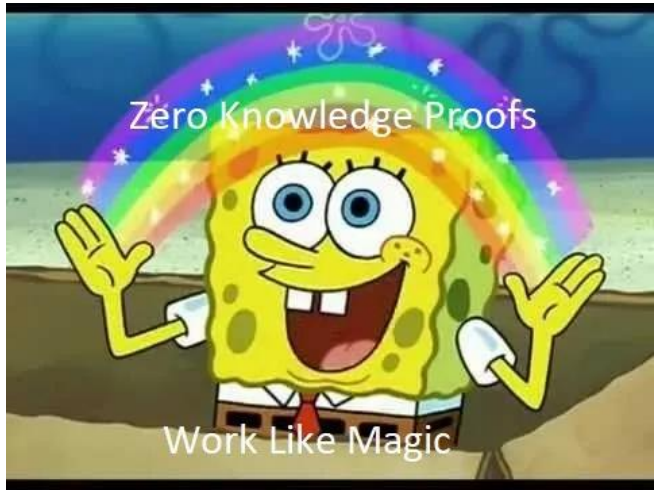




Correctness

Cryptography has an approach...

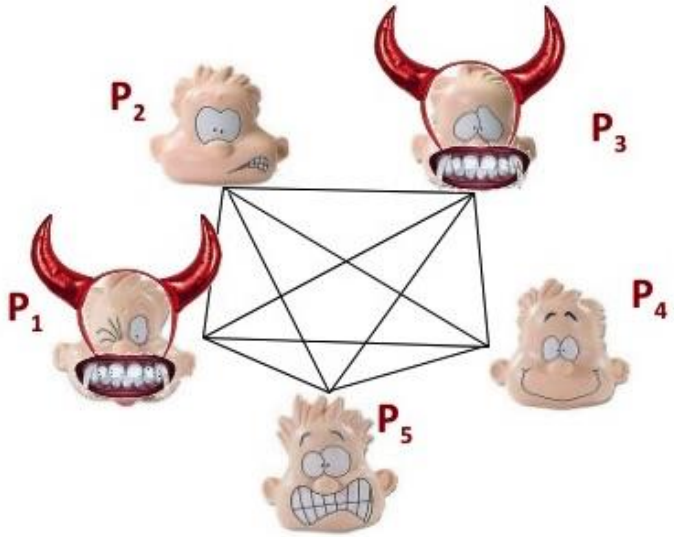


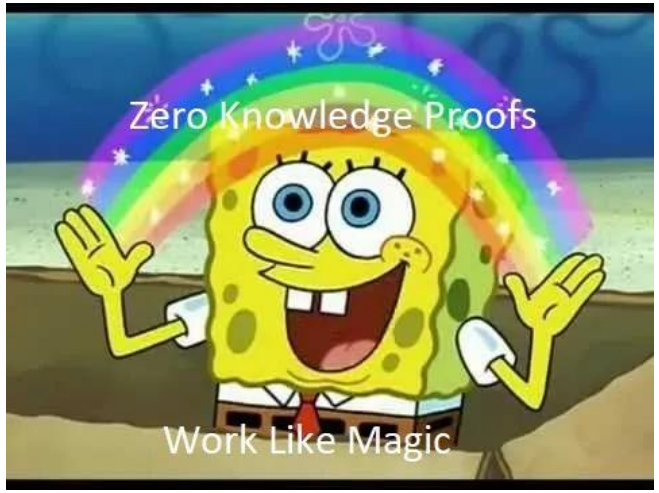


Correctness

Secrecy

Cryptography has an approach...





Correctness

Secrecy

Input independence

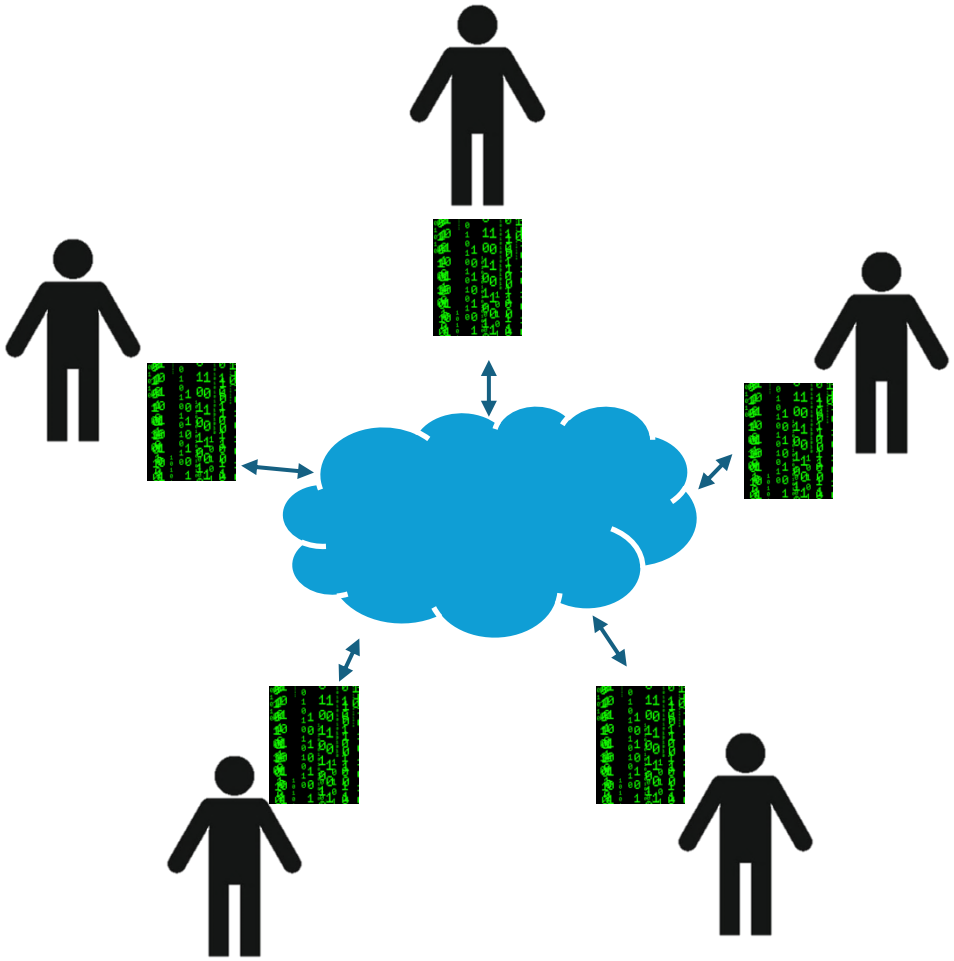
How to Spec?

How to Spec?

Real-Ideal Paradigm

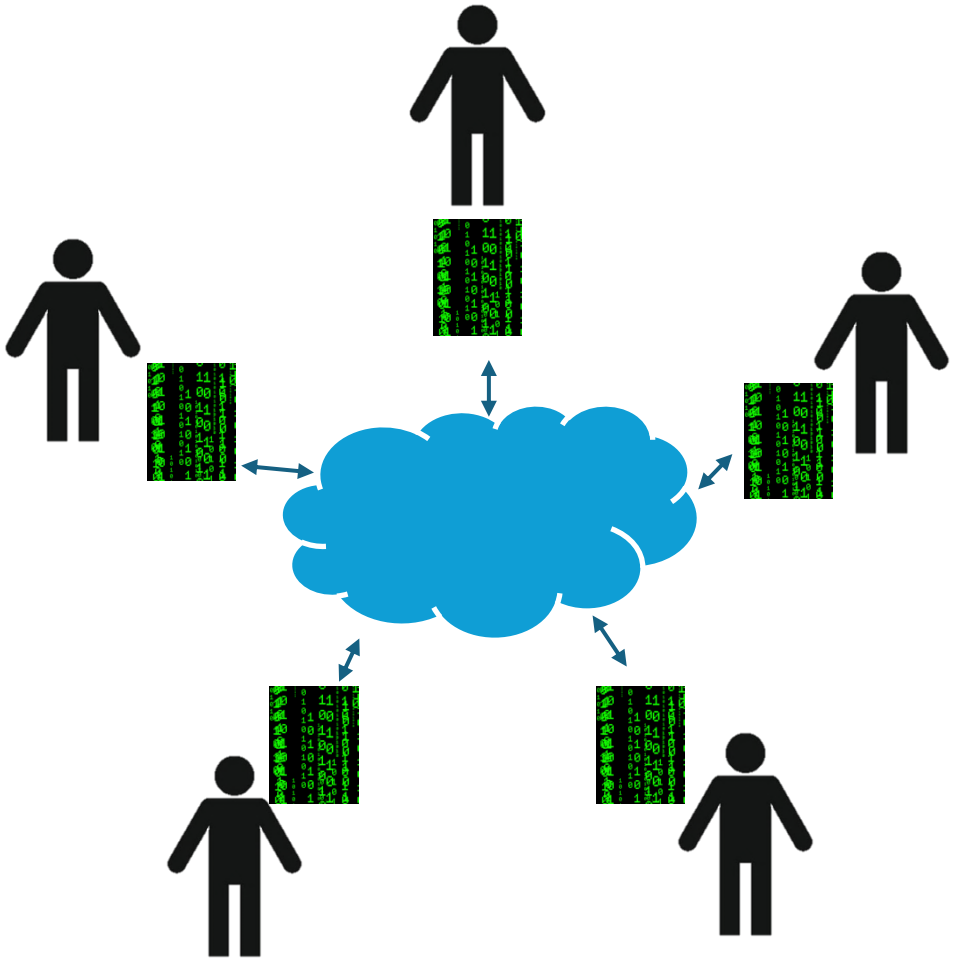
How to Spec?

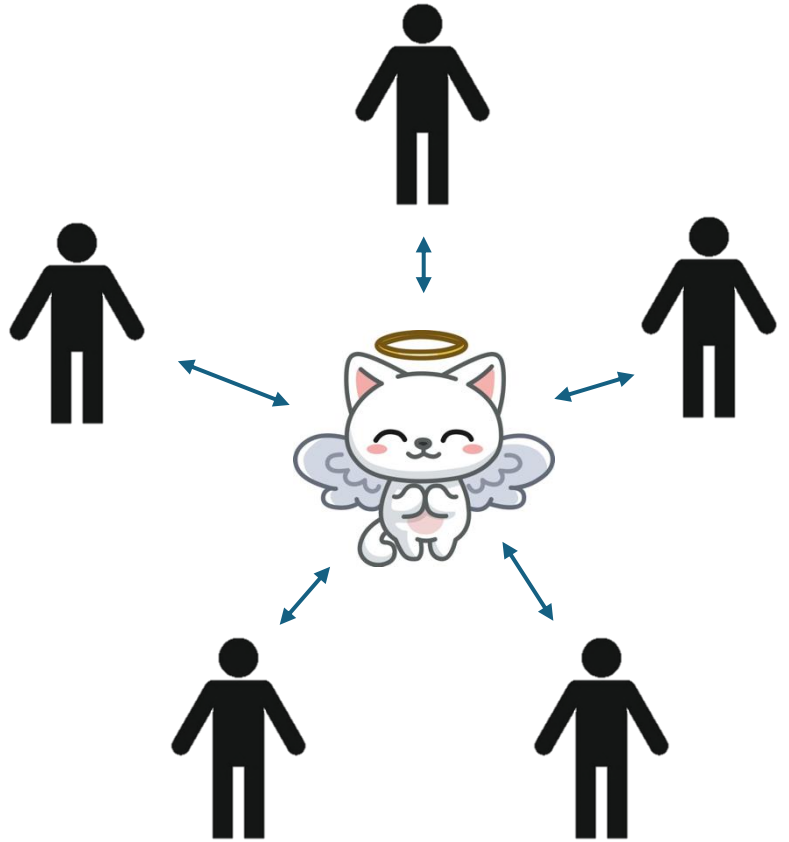
Real-Ideal Paradigm



How to Spec?

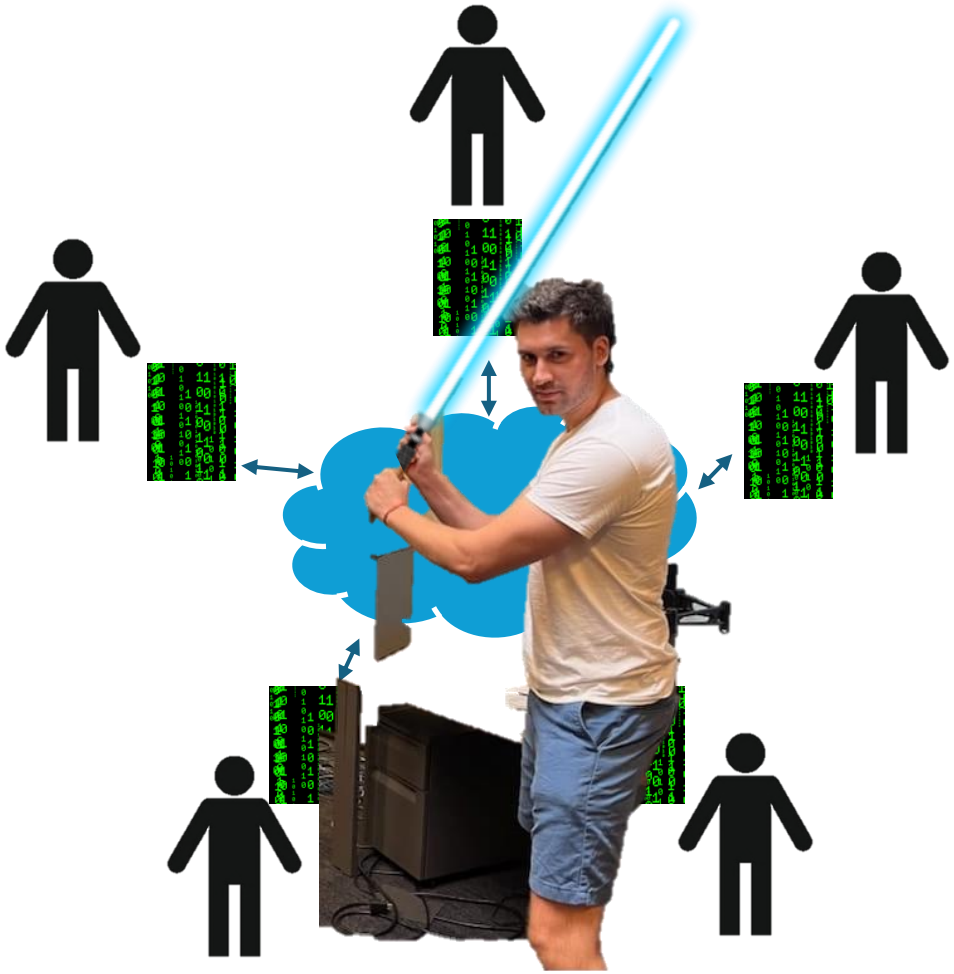
Real-Ideal Paradigm

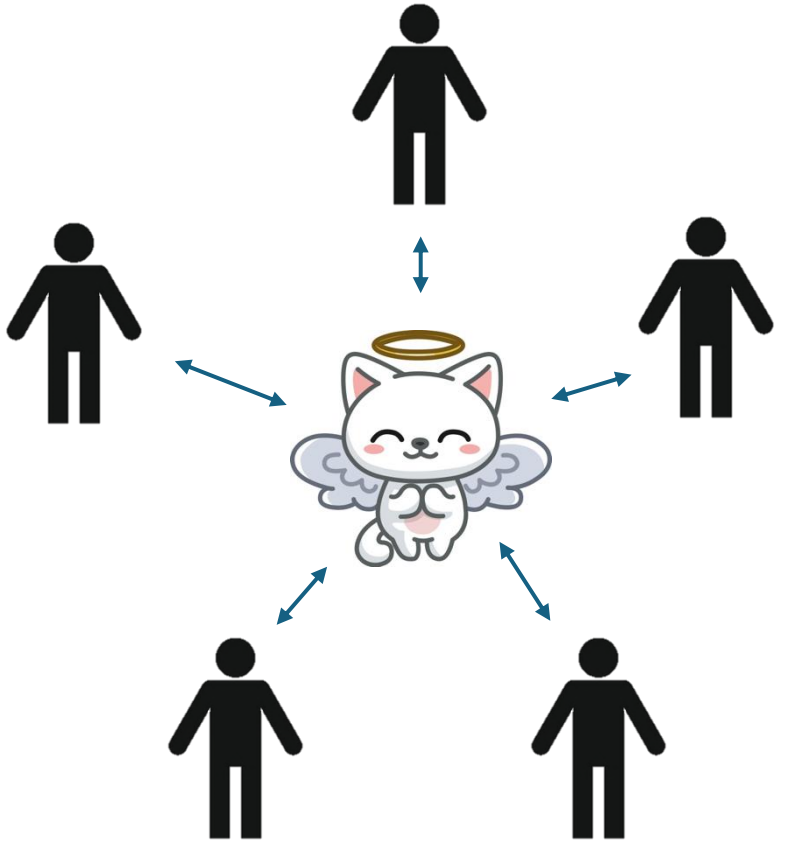




How to Spec?

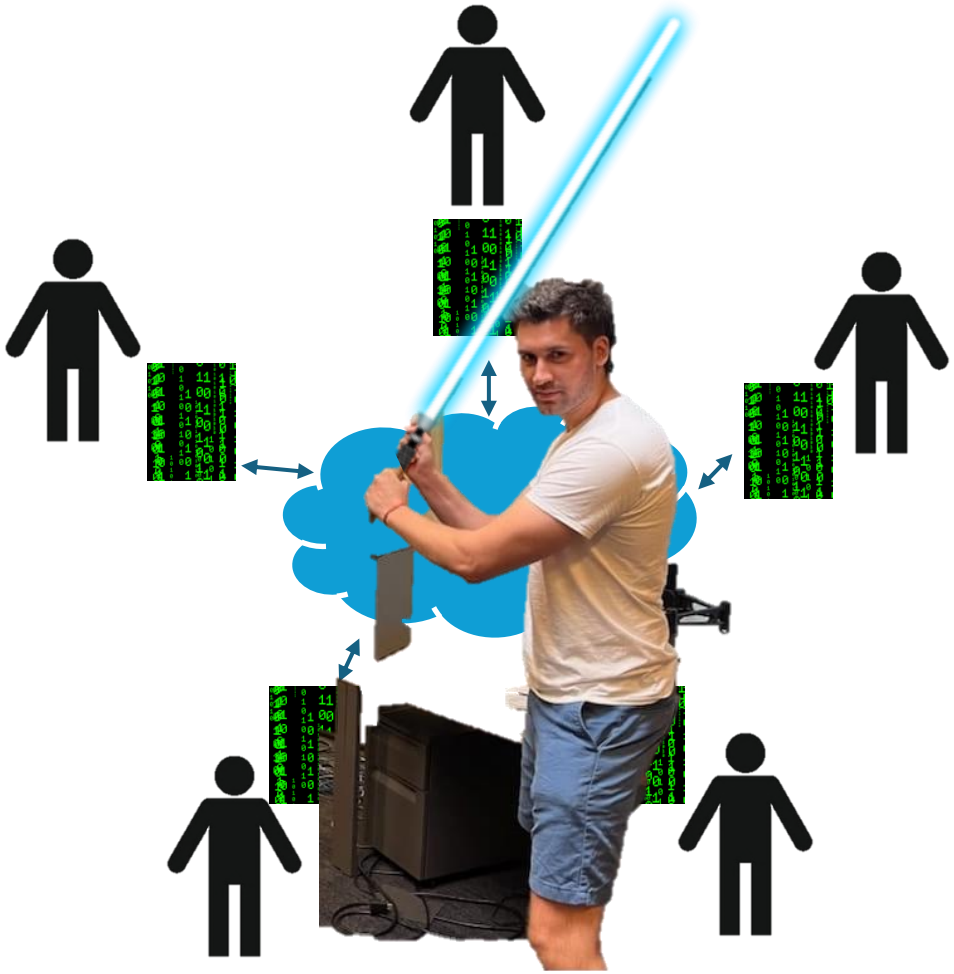
Real-Ideal Paradigm





How to Spec?

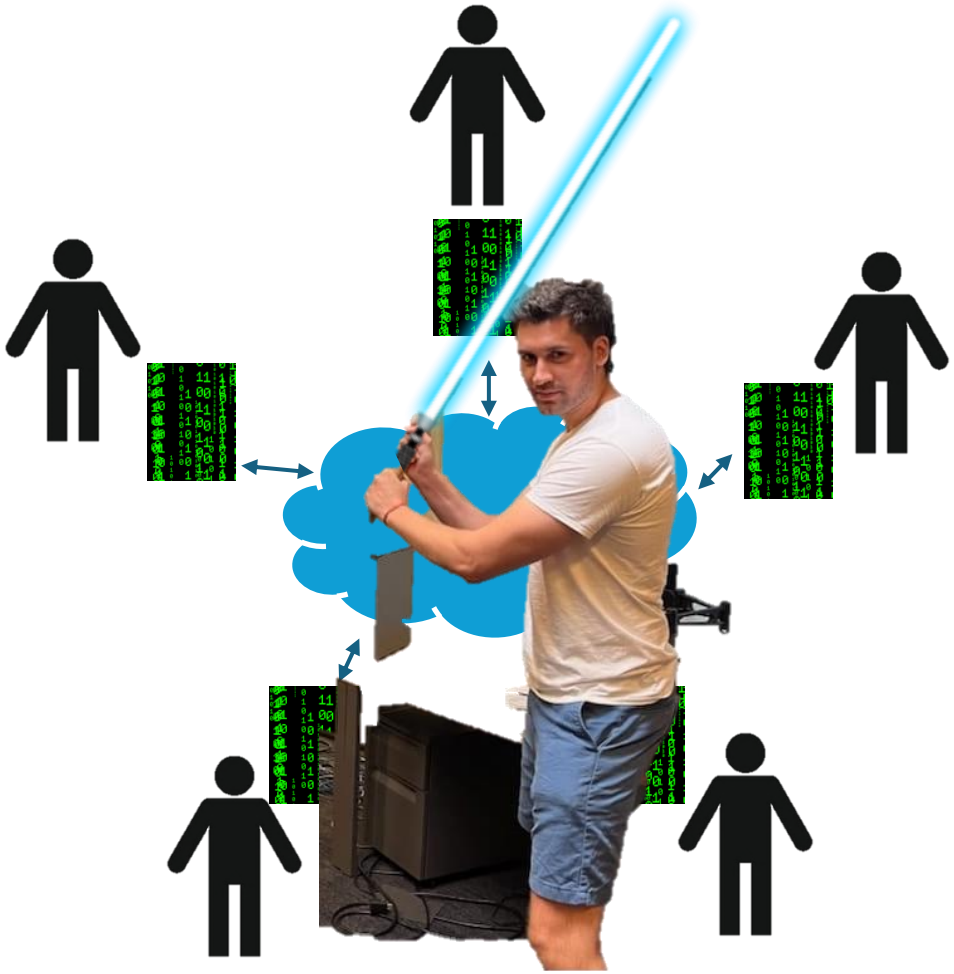
Real-Ideal Paradigm

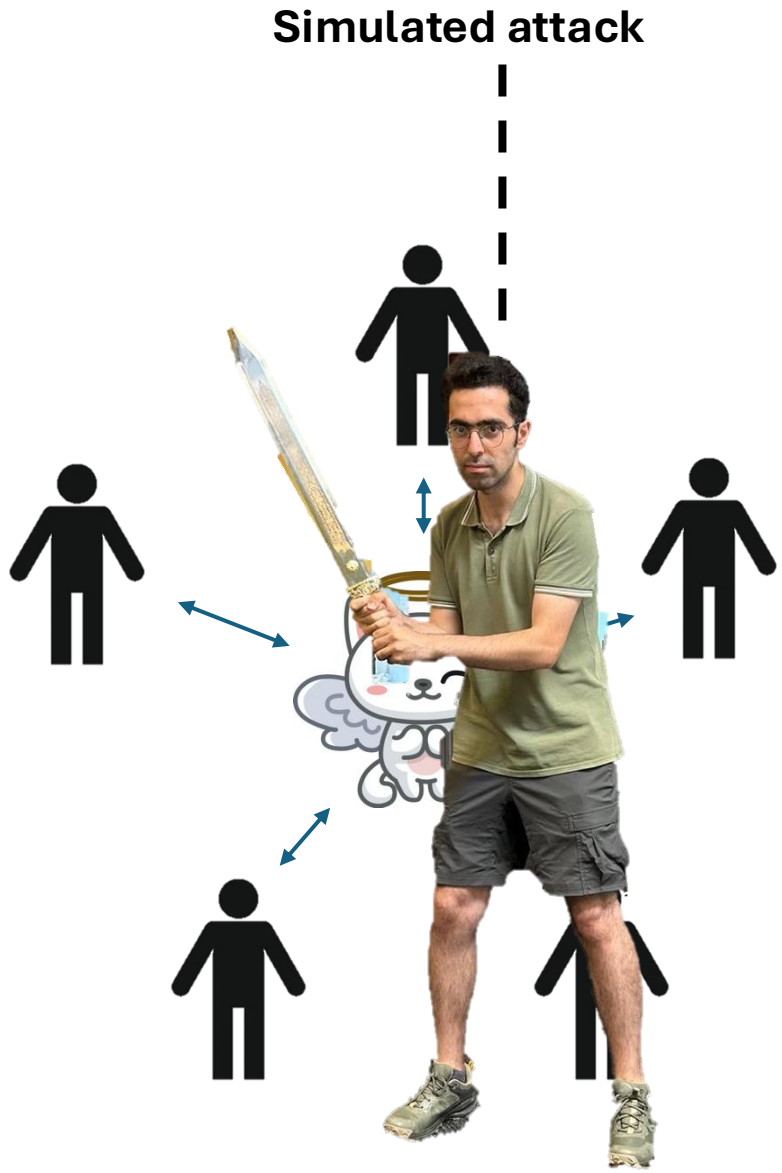




How to Spec?

Real-Ideal Paradigm





An example: Secure Communication

An example: Secure Communication

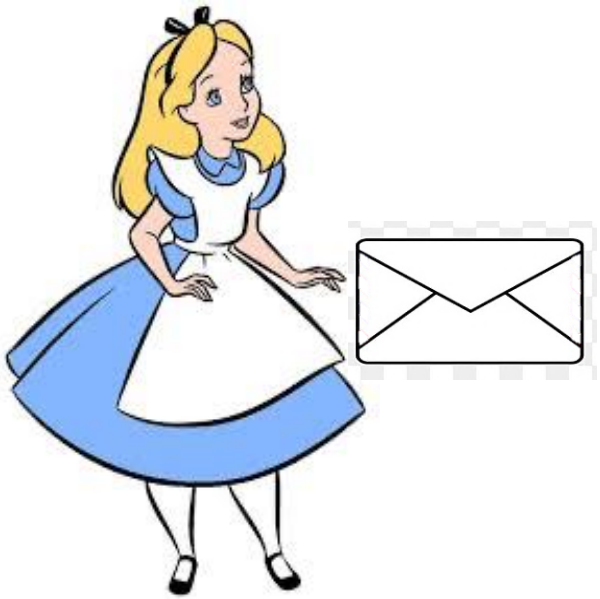


An example: Secure Communication



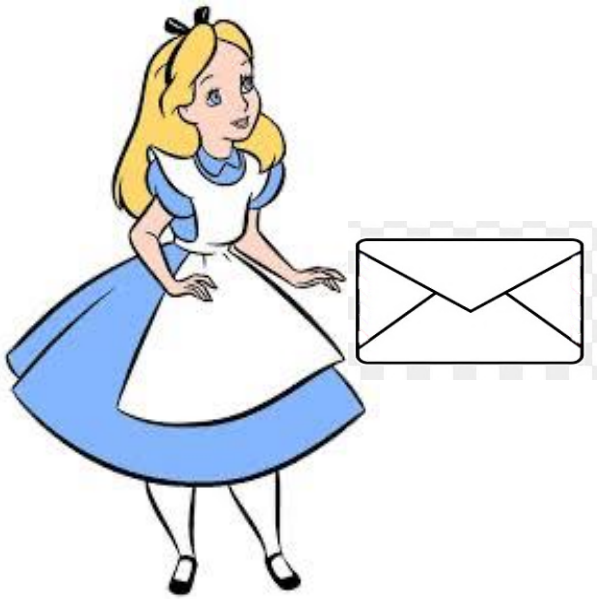


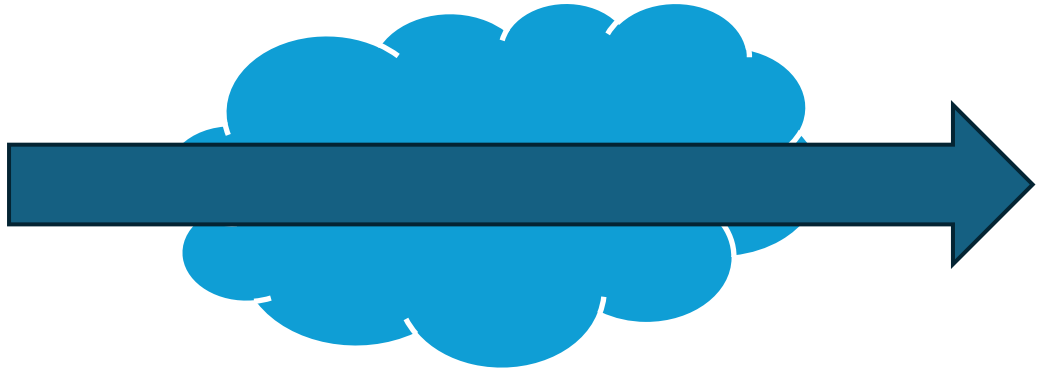
An example: Secure Communication





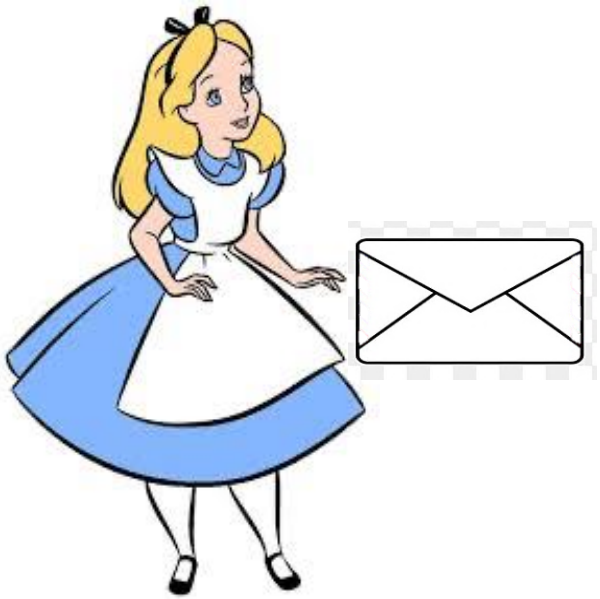
An example: Secure Communication

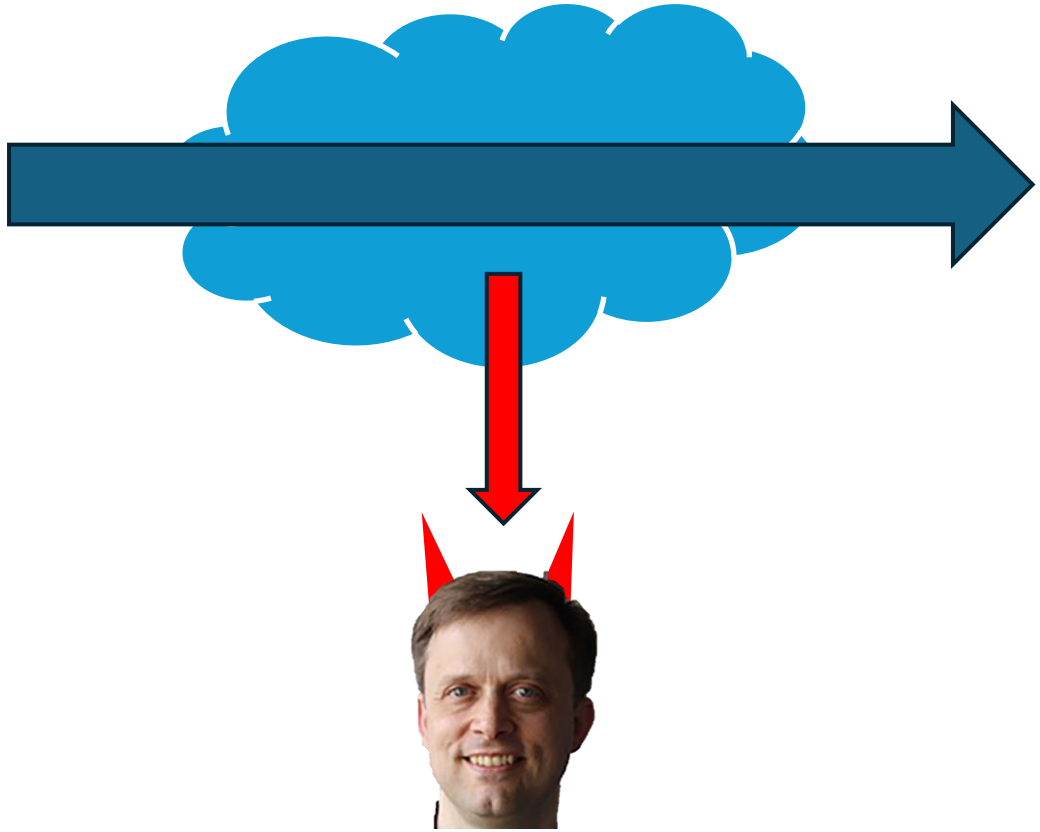






An example: Secure Communication

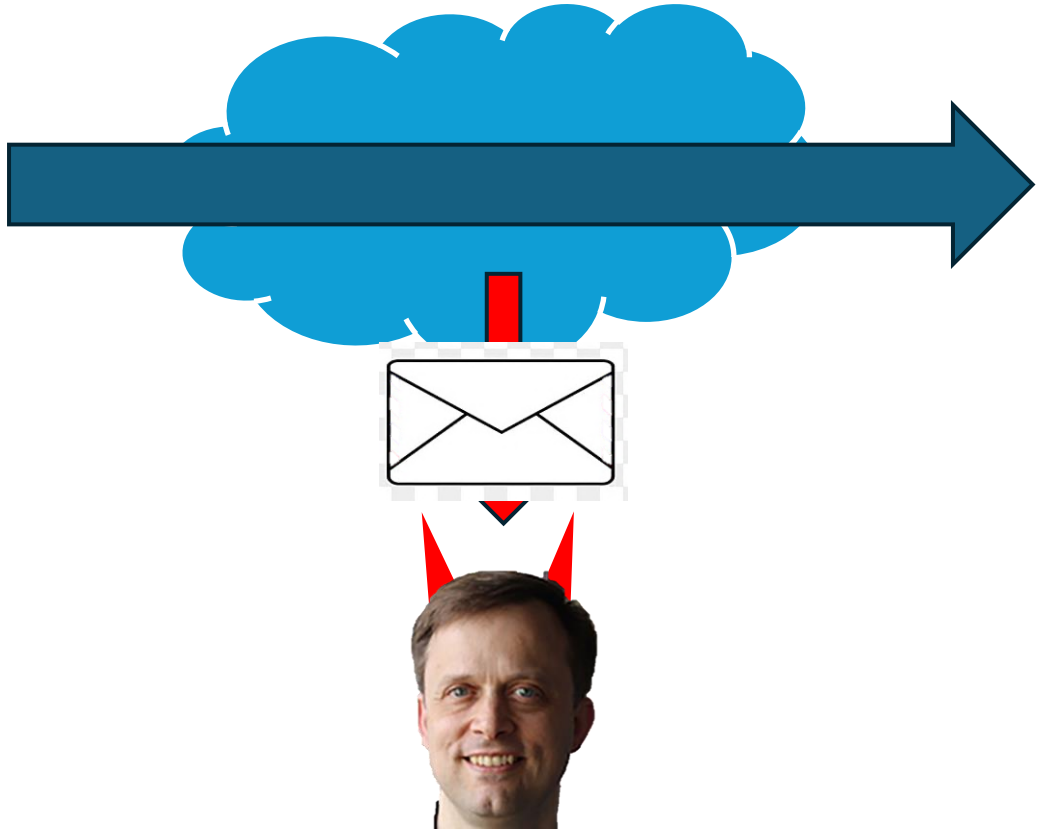


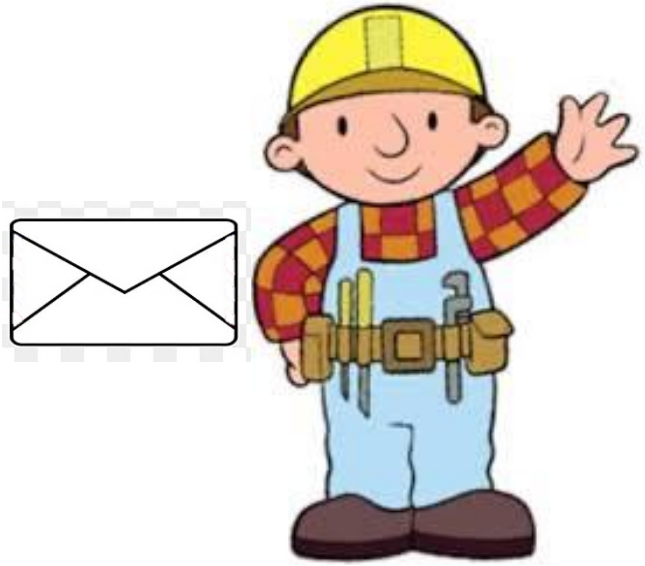




An example: Secure Communication



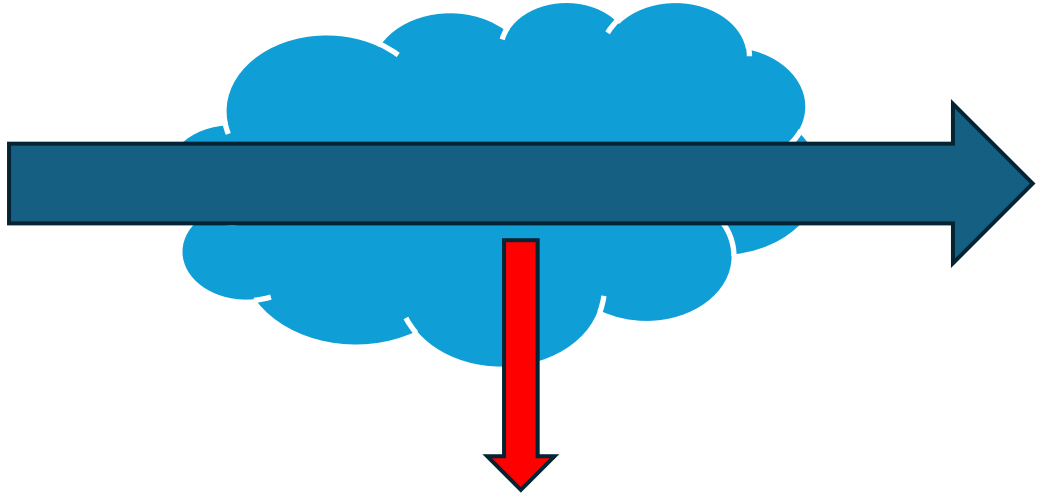




Encryption for secrecy

Encryption for secrecy



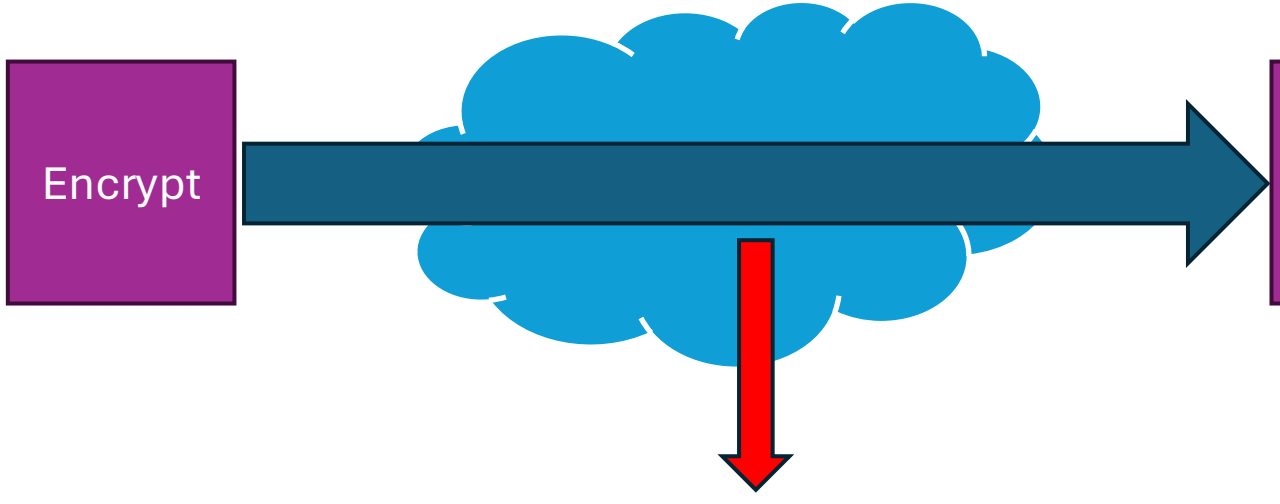






Encryption for secrecy

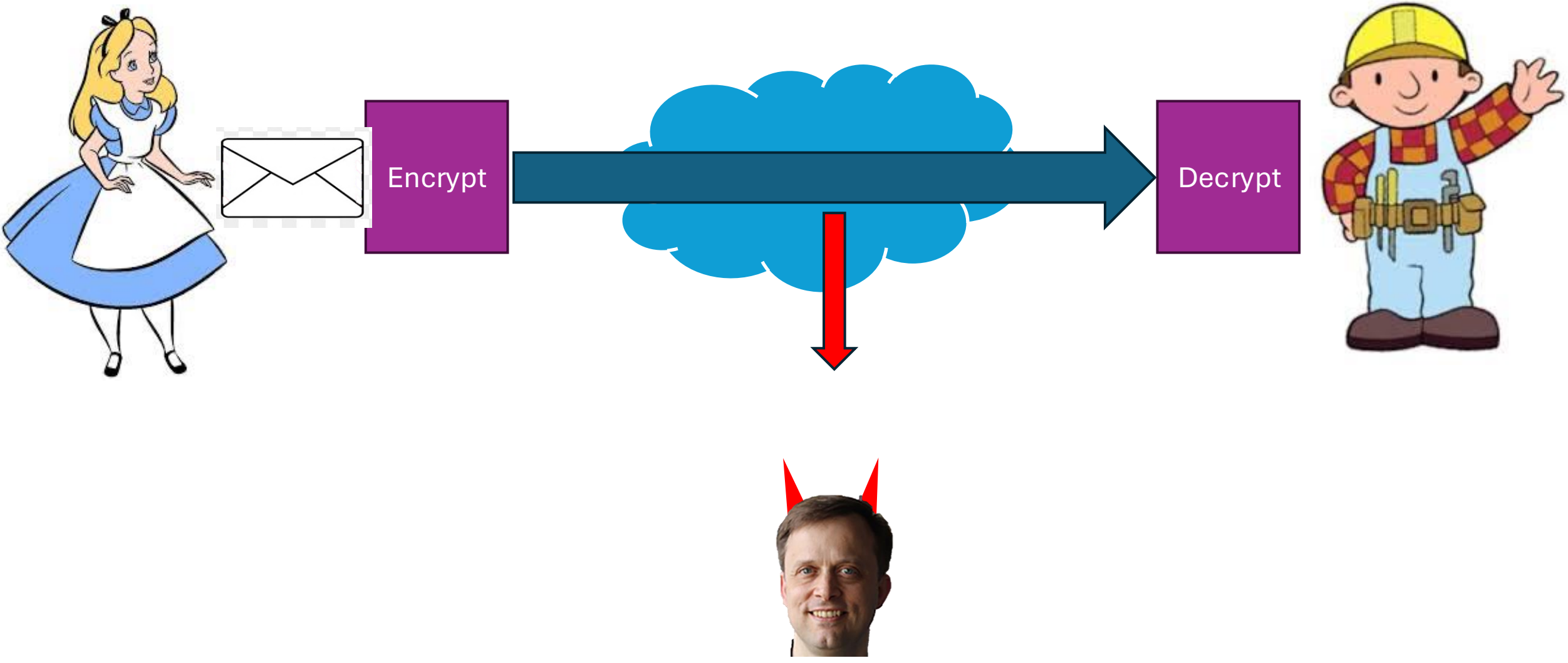






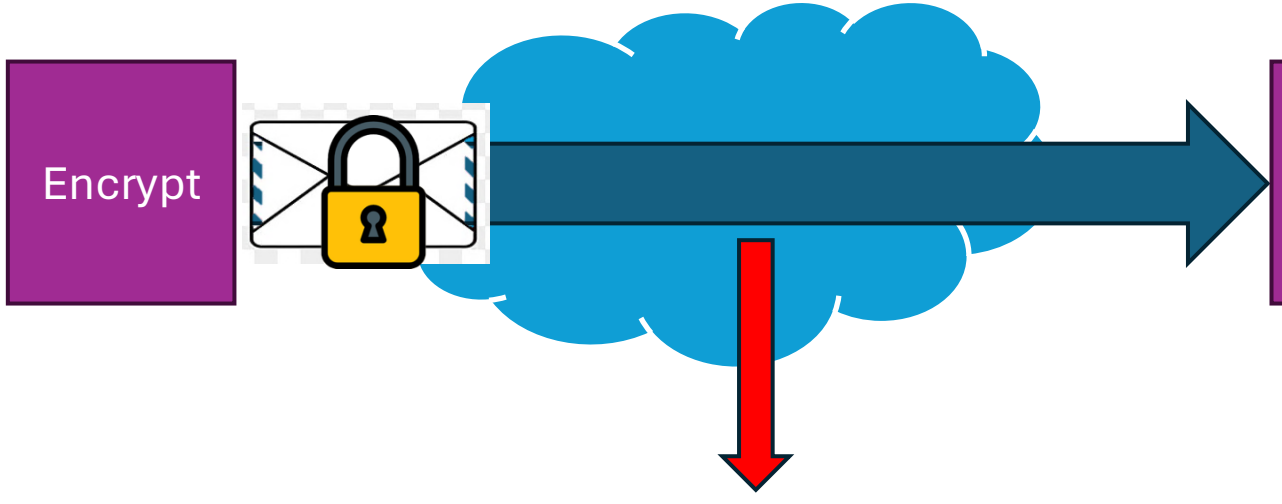


Encryption for secrecy



Encryption for secrecy



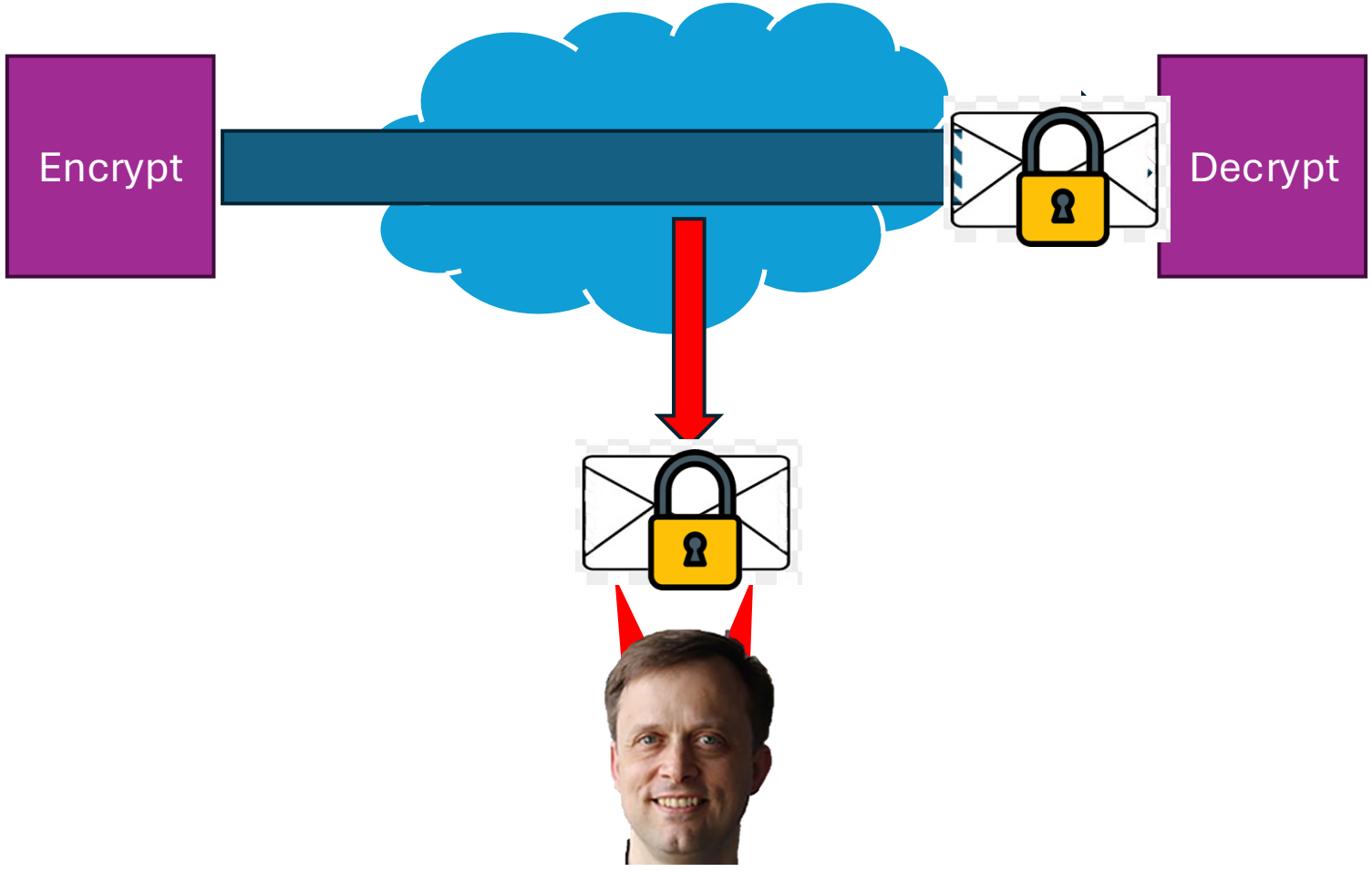






Encryption for secrecy





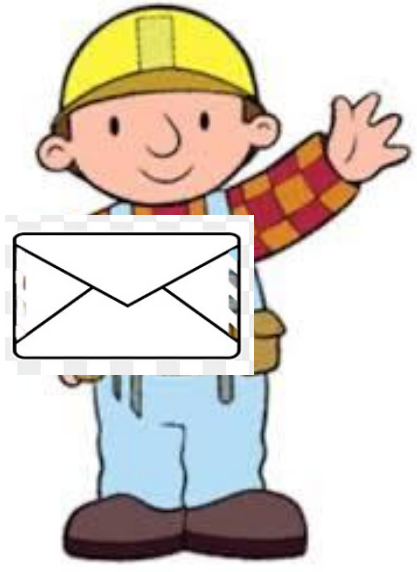


Encryption for secrecy



Encrypt

Decrypt





The Spec / Ideal world

The Spec / Ideal world



The Spec / Ideal world





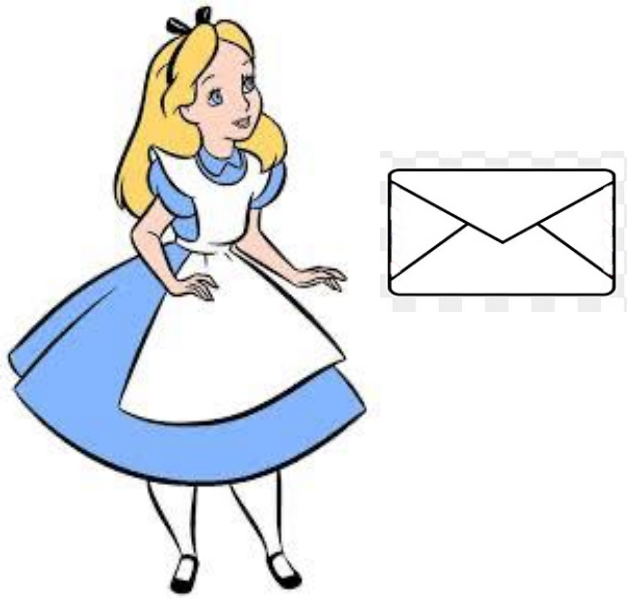
The Spec / Ideal world







The Spec / Ideal world

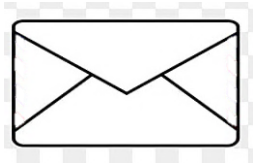






The Spec / Ideal world











The Spec / Ideal world



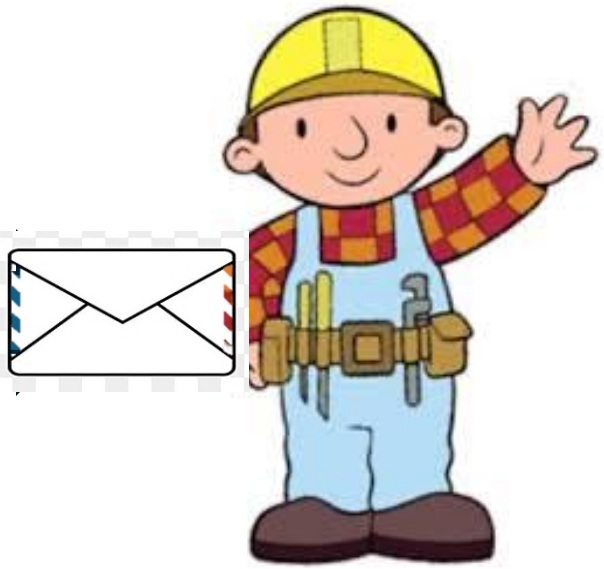




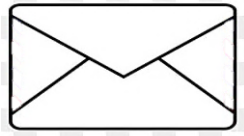
The Spec / Ideal world







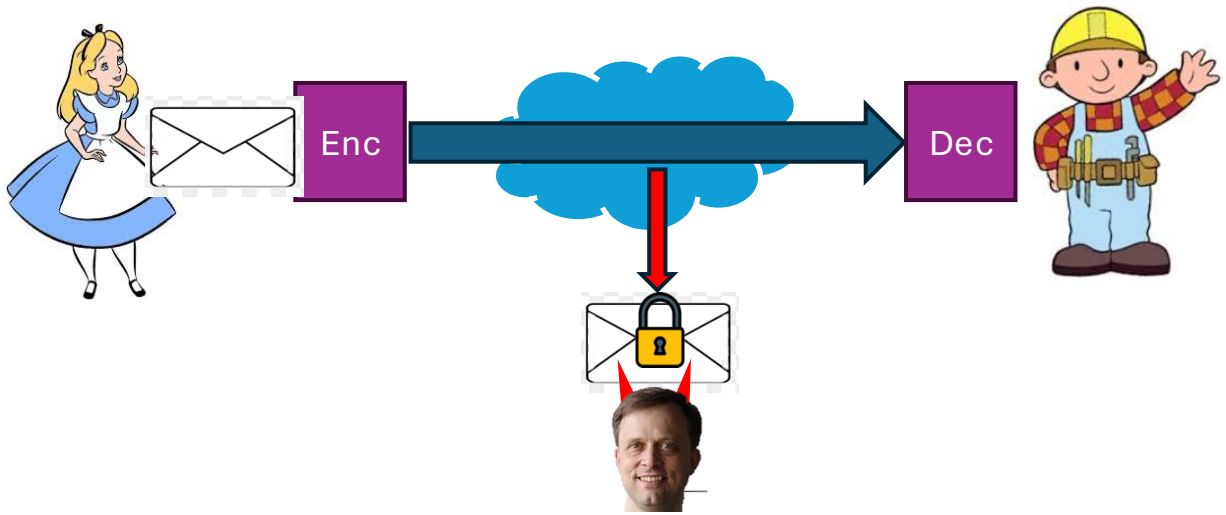


Len ()

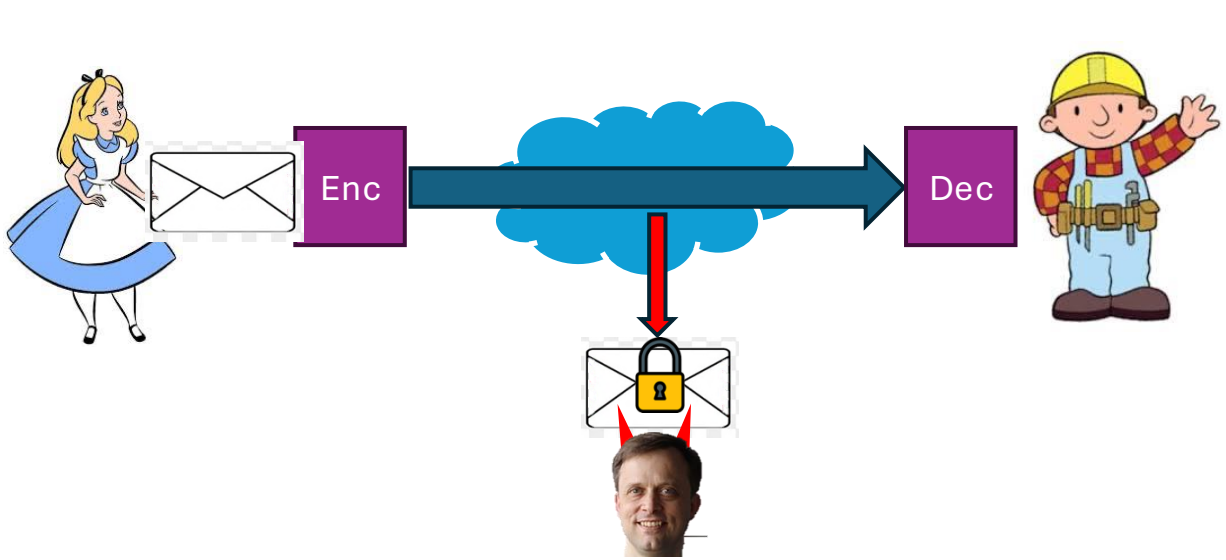


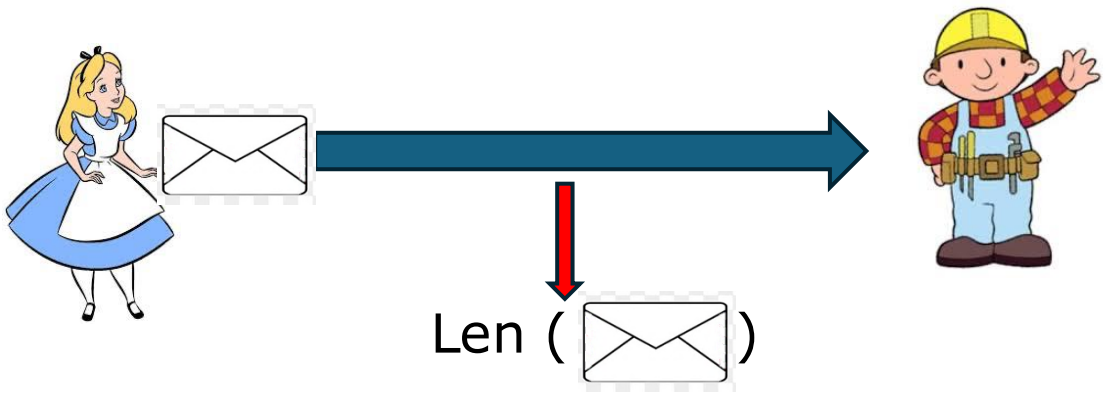
Real implements Ideal?

Real implements Ideal?

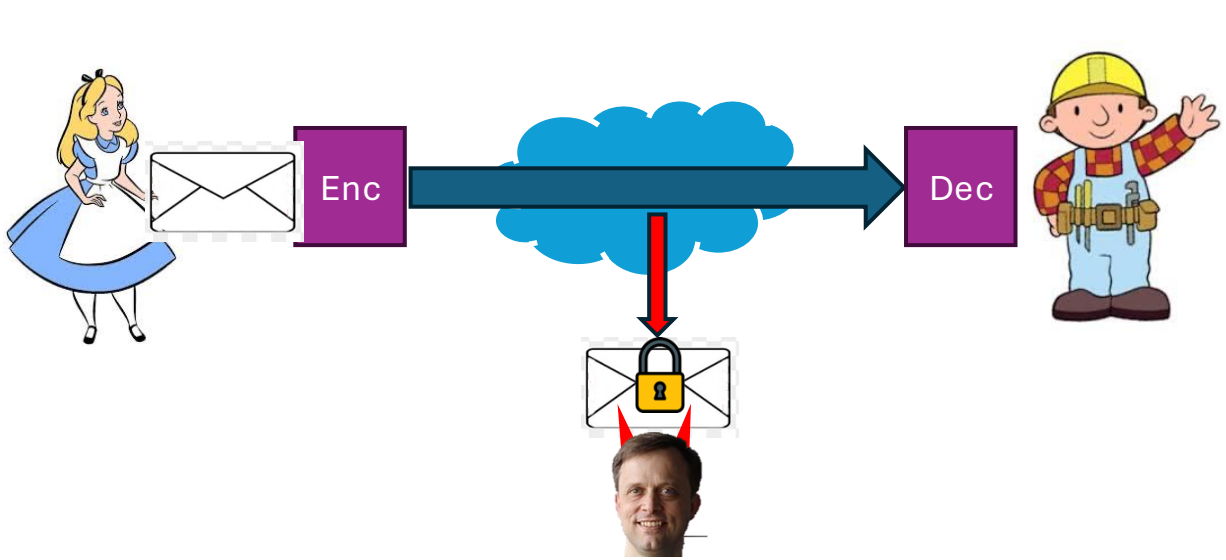


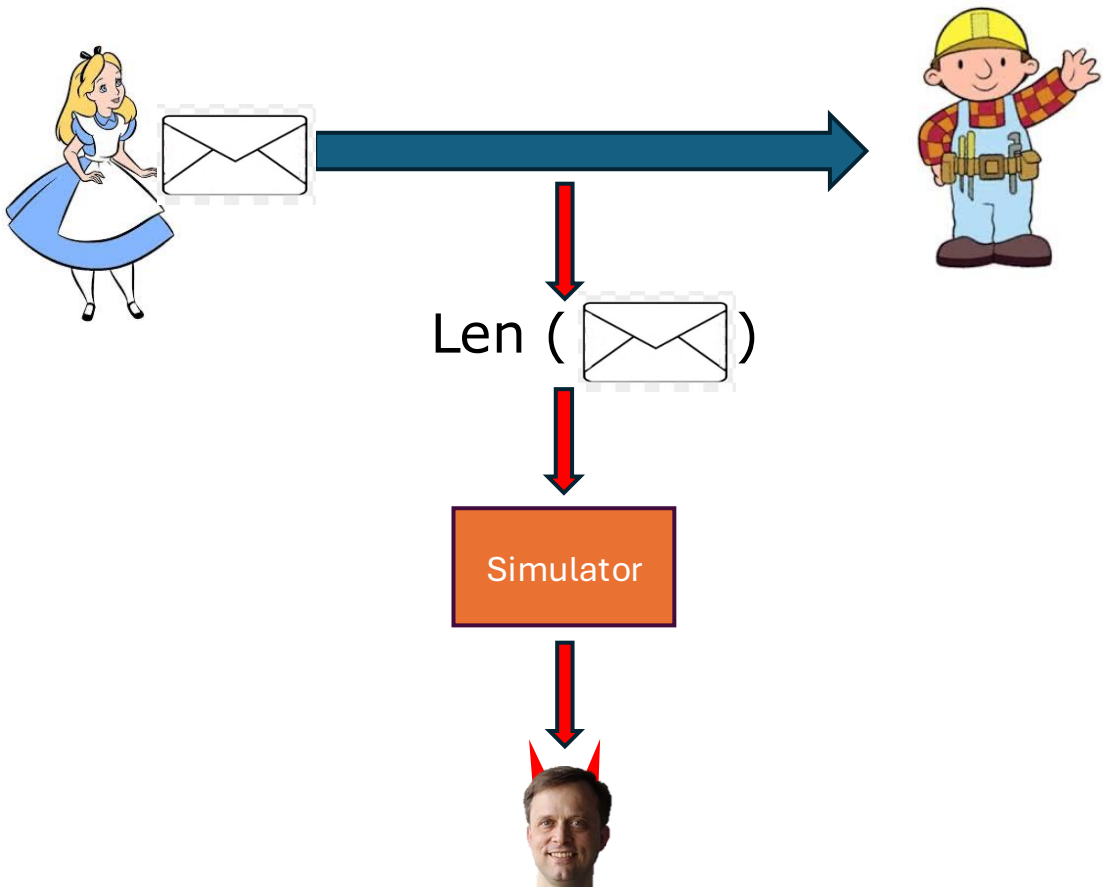
Real implements Ideal?



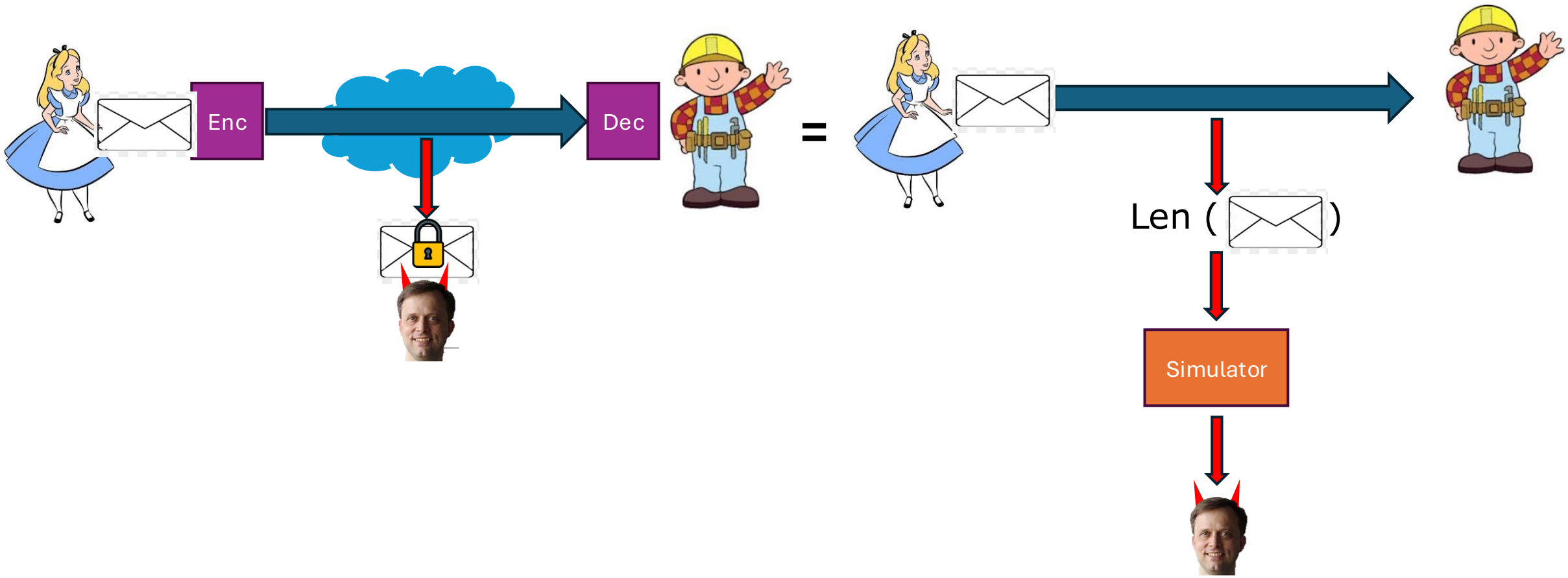


Real implements Ideal?

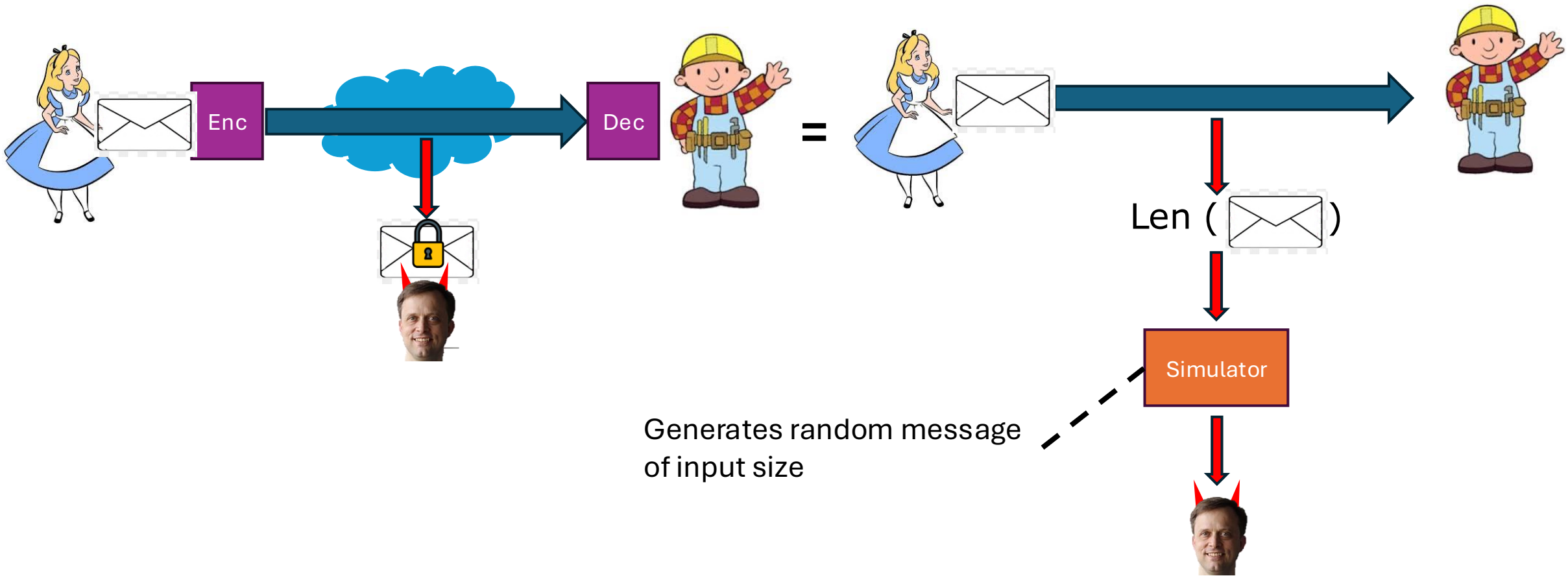




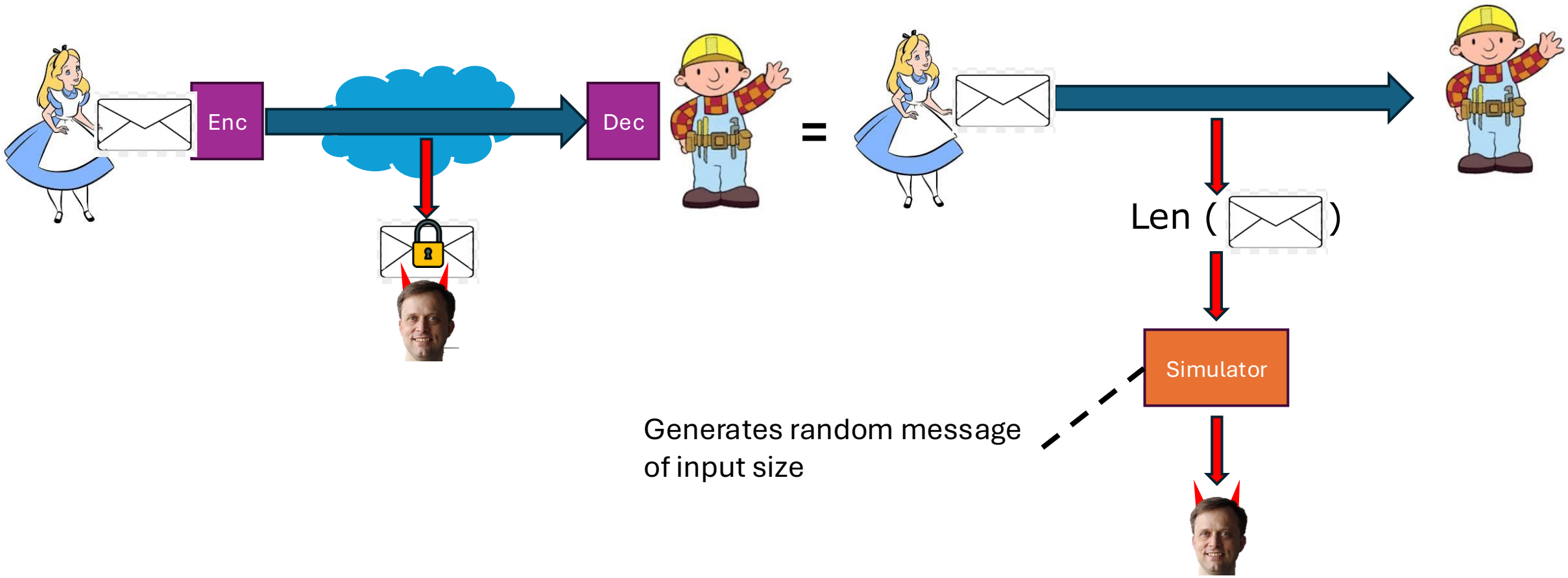
Real implements Ideal?



Real implements Ideal?

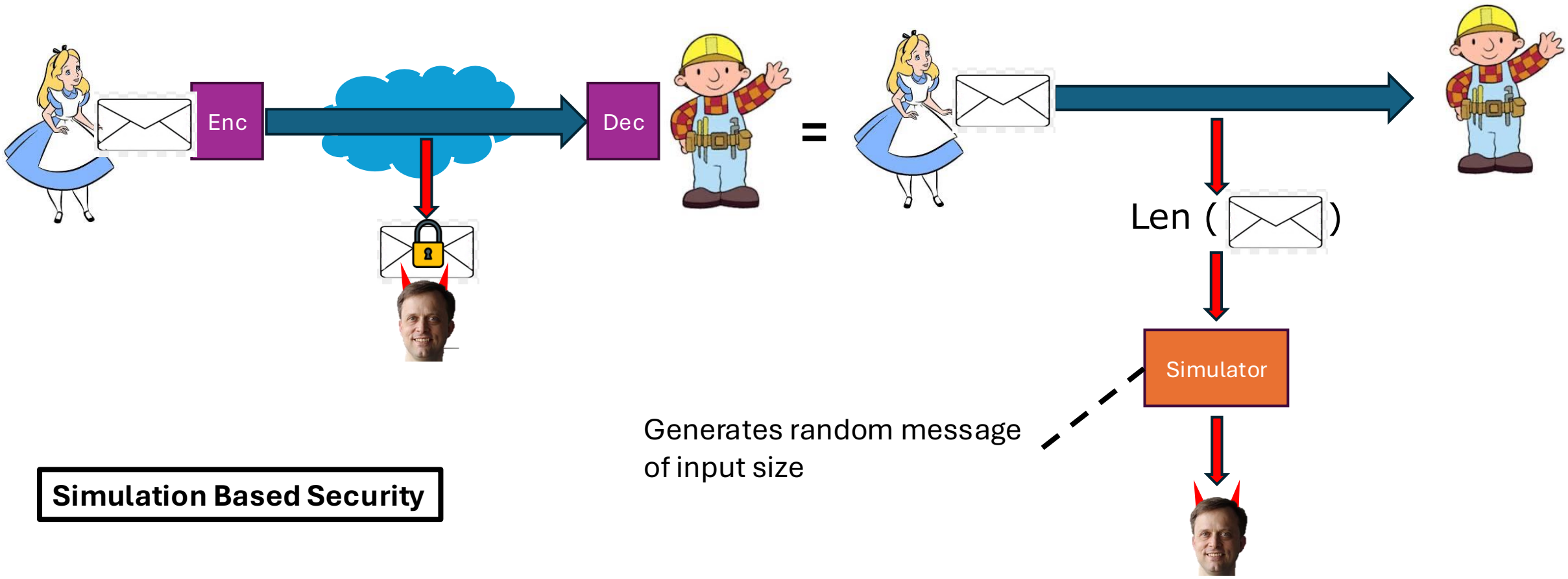


Real implements Ideal?



$$\exists Sim\colon (Sim \,||\, Ideal) = Real$$

Real implements Ideal?



Simulation Based Security

$$\exists Sim\colon (Sim \,||\, Ideal) = Real$$

This paper: Trustworthy HSMs

This paper: Trustworthy HSMs

HSM - Hardware Security Modules

This paper: Trustworthy HSMs

HSM - Hardware Security Modules

Lots of uses:

This paper: Trustworthy HSMs

HSM - Hardware Security Modules

Lots of uses:

- U2F

This paper: Trustworthy HSMs

HSM - Hardware Security Modules

Lots of uses:

- U2F
- OTP

This paper: Trustworthy HSMs

HSM - Hardware Security Modules

Lots of uses:

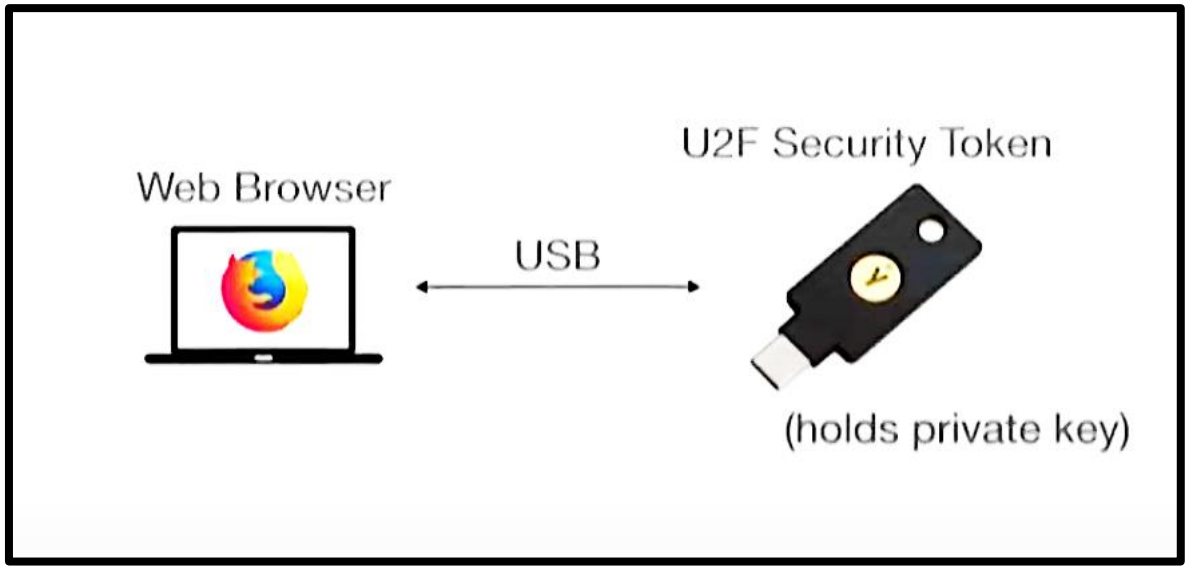
- U2F
- OTP
- Crypto

This paper: Trustworthy HSMs

HSM - Hardware Security Modules

Lots of uses:

- U2F
- OTP
- Crypto



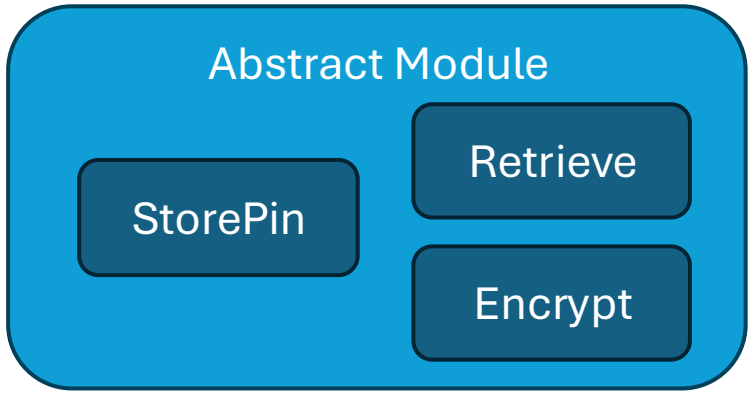
The Big Picture

The Big Picture



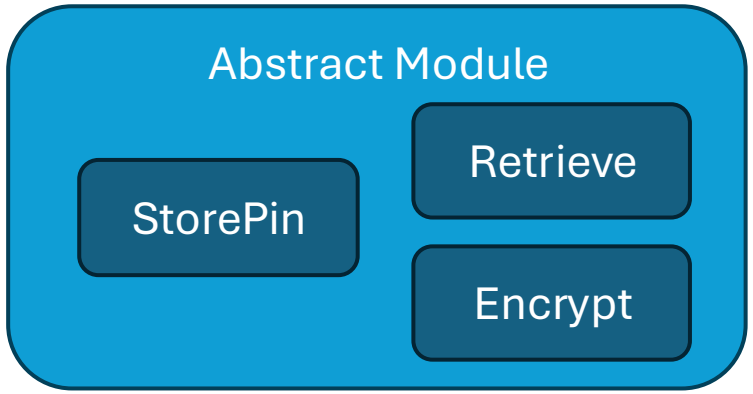
The Big Picture





The Big Picture

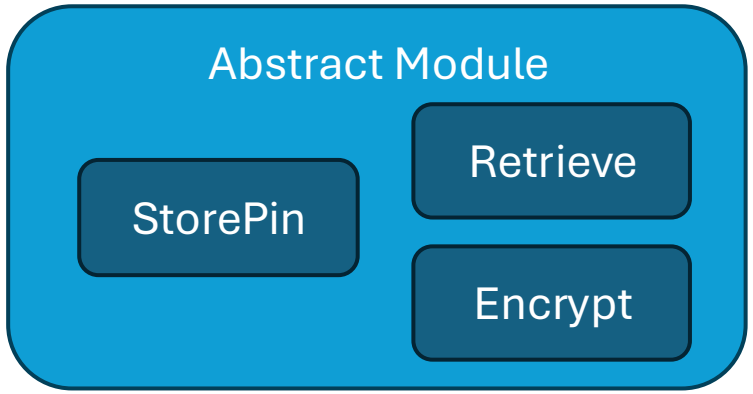




1. Correct functioning with **expected behavior**

The Big Picture



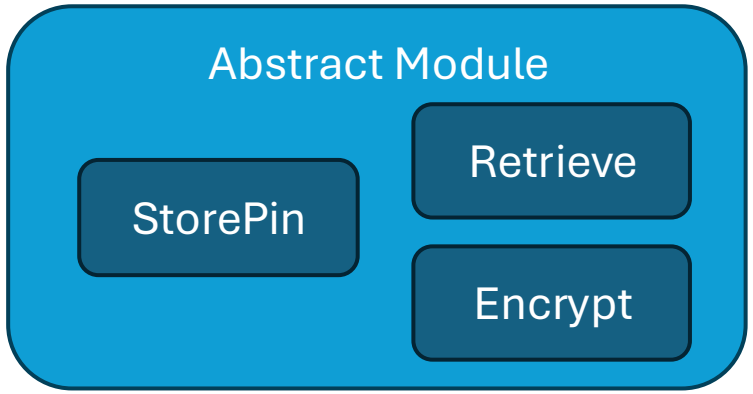


1. Correct functioning with **expected behavior**

- Use the expected protocol

The Big Picture

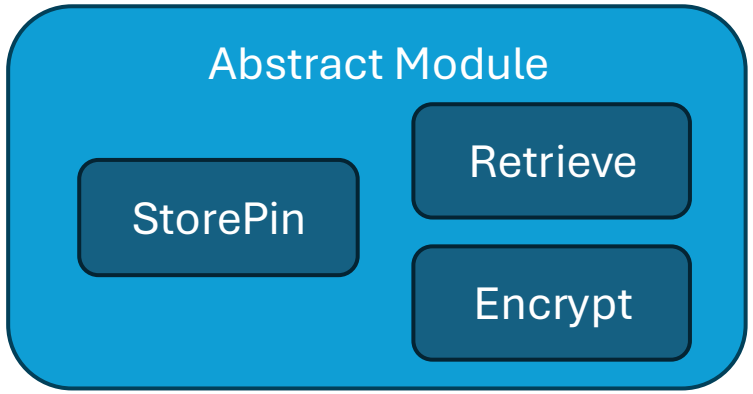




1. Correct functioning with **expected behavior**
 - Use the expected protocol
2. ‘Nothing weird’ with **unexpected behavior**

The Big Picture





1. Correct functioning with **expected behavior**
 - Use the expected protocol
2. ‘Nothing weird’ with **unexpected behavior**
- Start looking at / probing wire traces

Why not just care about correct functioning?

Why not just care about correct functioning?

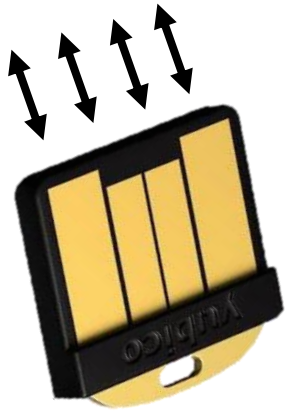


Why not just care about correct functioning?



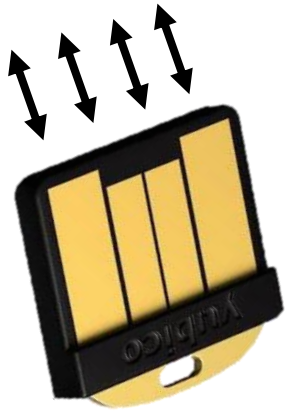
Why not just care about correct functioning?





Why not just care about correct functioning?

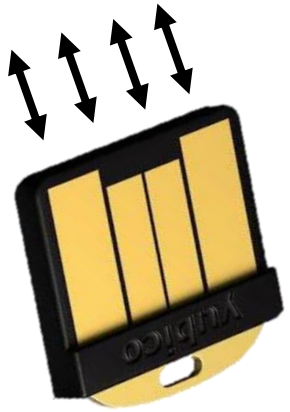


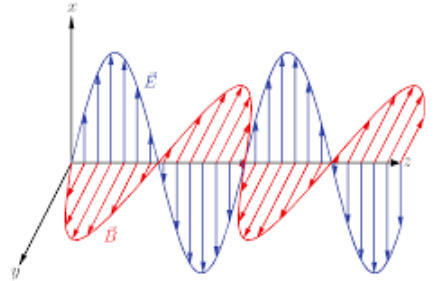


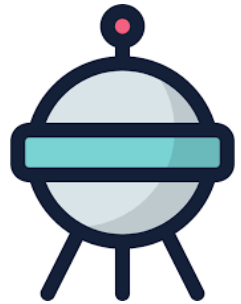
Threat Model

Why not just care about correct functioning?





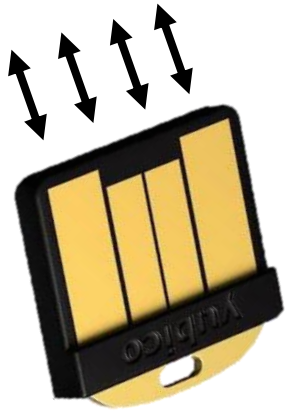


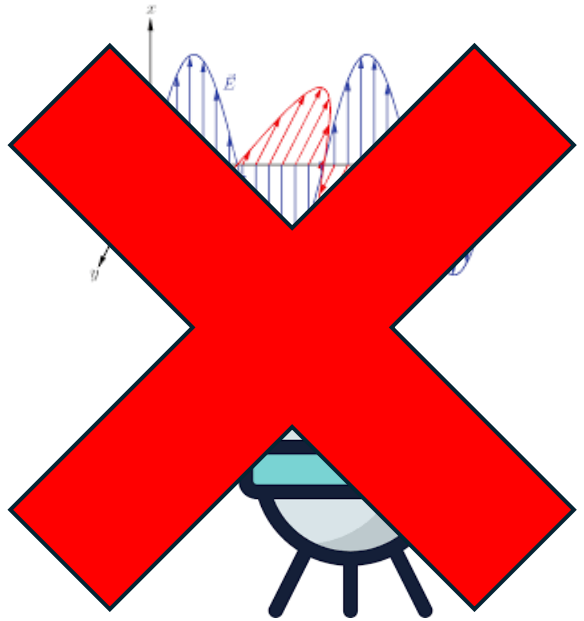


Threat Model

Why not just care about correct functioning?







Threat Model

An Example: Timing Channel

An Example: Timing Channel

```
var password

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```

An Example: Timing Channel

```
var password

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```



An Example: Timing Channel

```
var password = 1234

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```



An Example: Timing Channel

```
var password

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

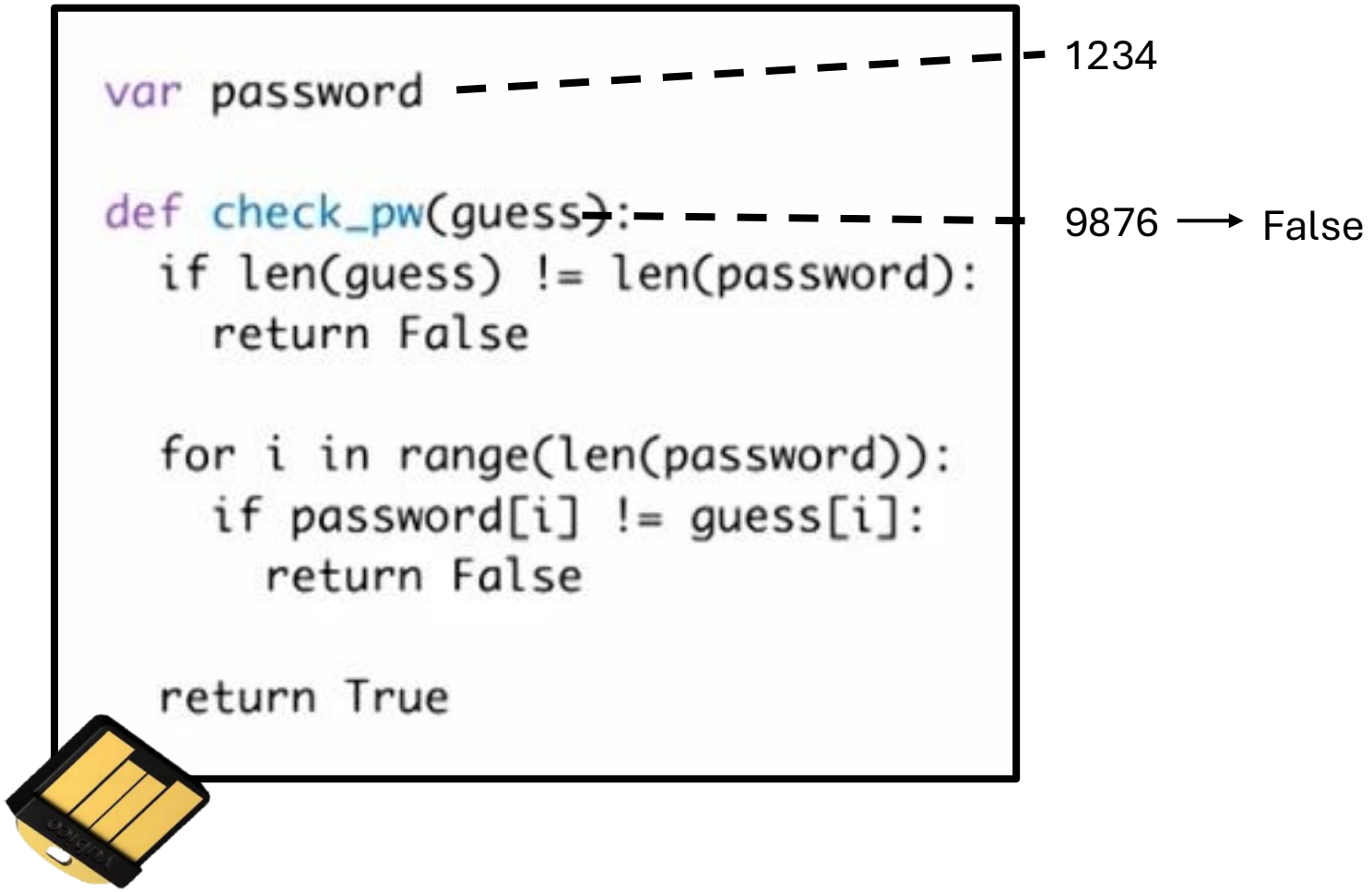
  return True
```

1234

9876



An Example: Timing Channel



An Example: Timing Channel

```
var password

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```

1234

9876 ⟶ False





An Example: Timing Channel

```
var password

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```

1234

9876 → False

1231



An Example: Timing Channel

```
var password

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```

1234

9876 → False

1231 → False



An Example: Timing Channel

```
var password

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```

1234

9876 → False

1231 → False







An Example: Timing Channel




```
var password

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```

1234

9876 → False

1231 → False







An Example: Timing Channel




```
var password

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```

1234

9876 → False

1231 → False



Module

check_pw





An Example: Timing Channel




```
var password

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```

1234

9876 → False

1231 → False



Module

check_pw

26^N



An Example: Timing Channel




```
var password

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```

1234

9876 → False

1231 → False



Module

check_pw

26^N







An Example: Timing Channel




```
var password

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```

1234

9876 → False

1231 → False



Module

check_pw

26^N





26*N

The Goal: Trustworthy HSMs

The Goal: Trustworthy HSMs

We need -

The Goal: Trustworthy HSMs

We need -



- **Functional Correctness:** Expected behavior should give expected results

The Goal: Trustworthy HSMs

We need -



- **Functional Correctness:** Expected behavior should give expected results



- **Nonleakage:** Compromise (unexpected behavior) should ‘still be fine’

The Goal: Trustworthy HSMs

We need -



- **Functional Correctness:** Expected behavior should give expected results

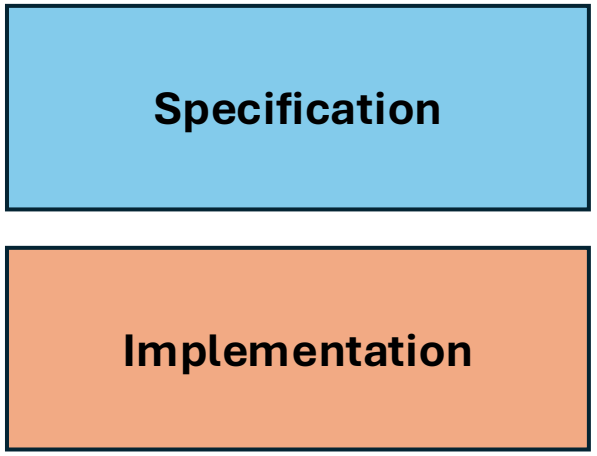


- **Nonleakage:** Compromise (unexpected behavior) should ‘still be fine’

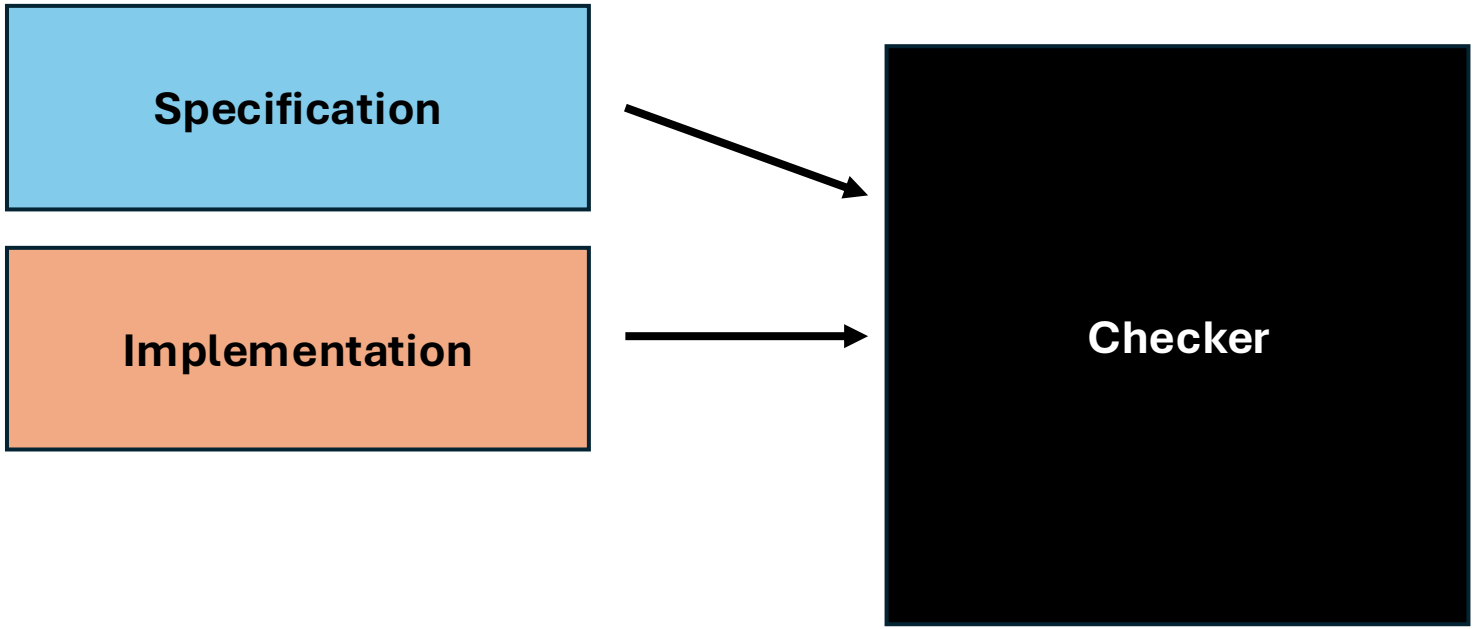
Together - **Information Preserving Refinement**

Formal Verification

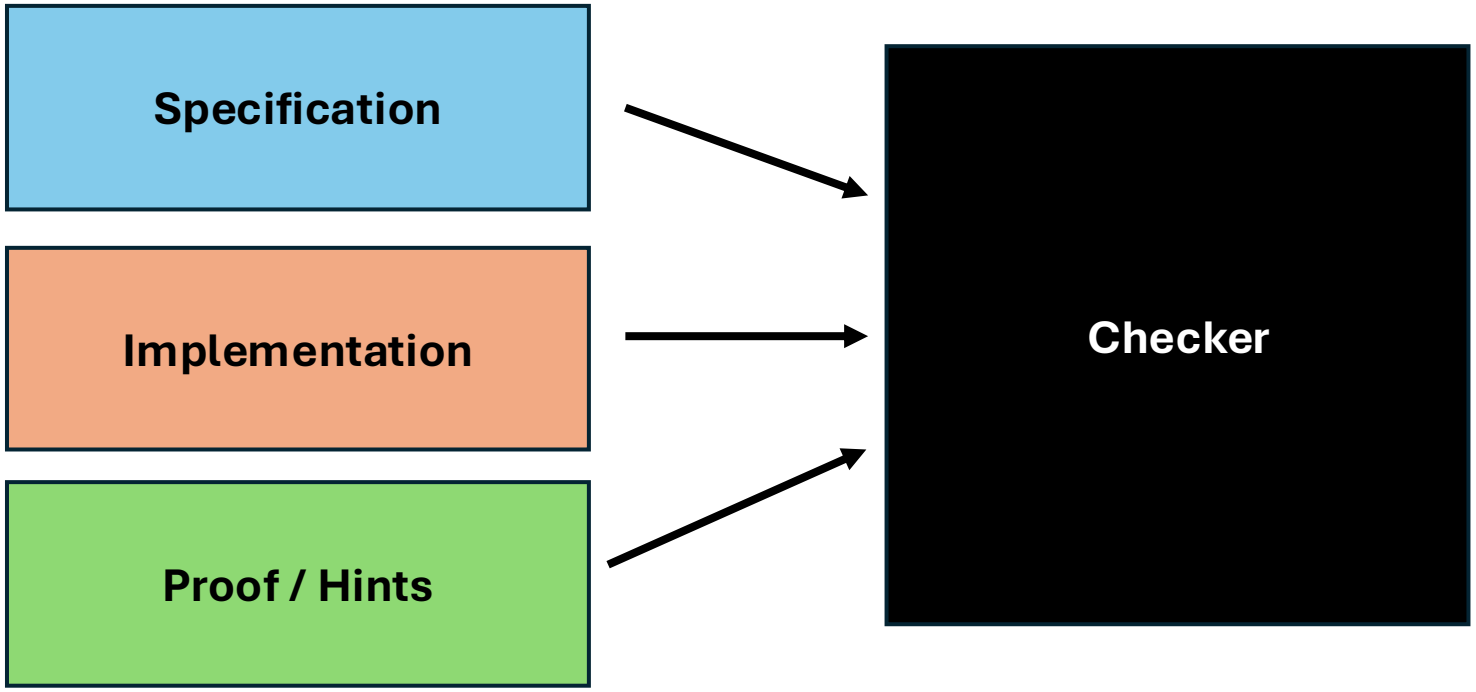
Formal Verification



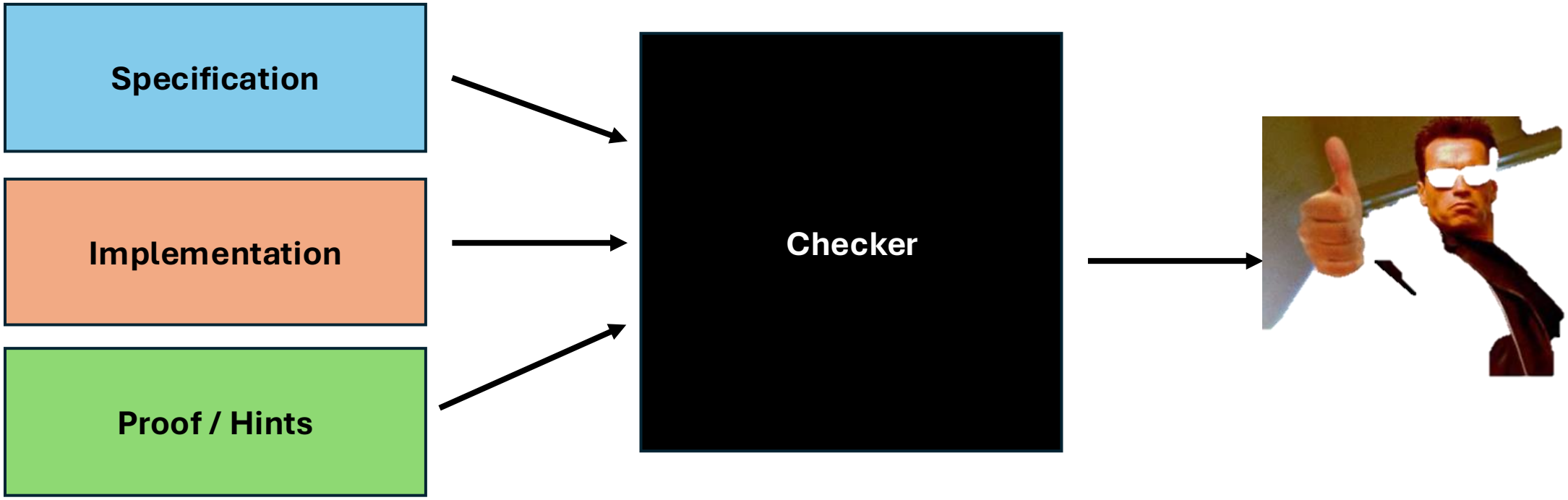
Formal Verification



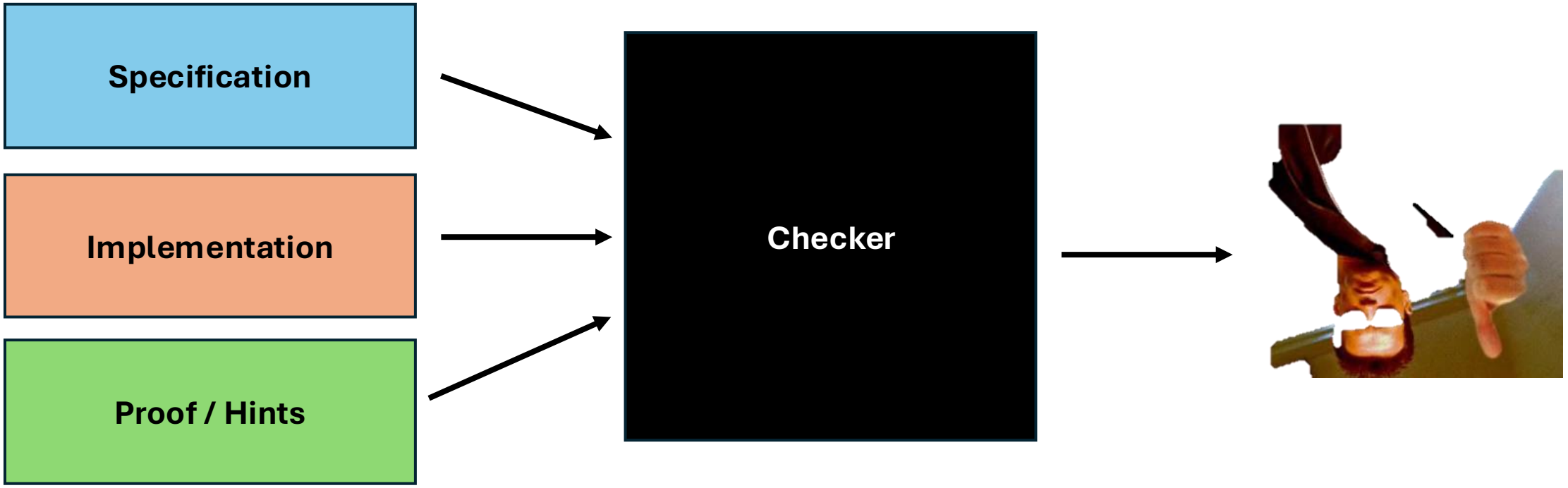
Formal Verification



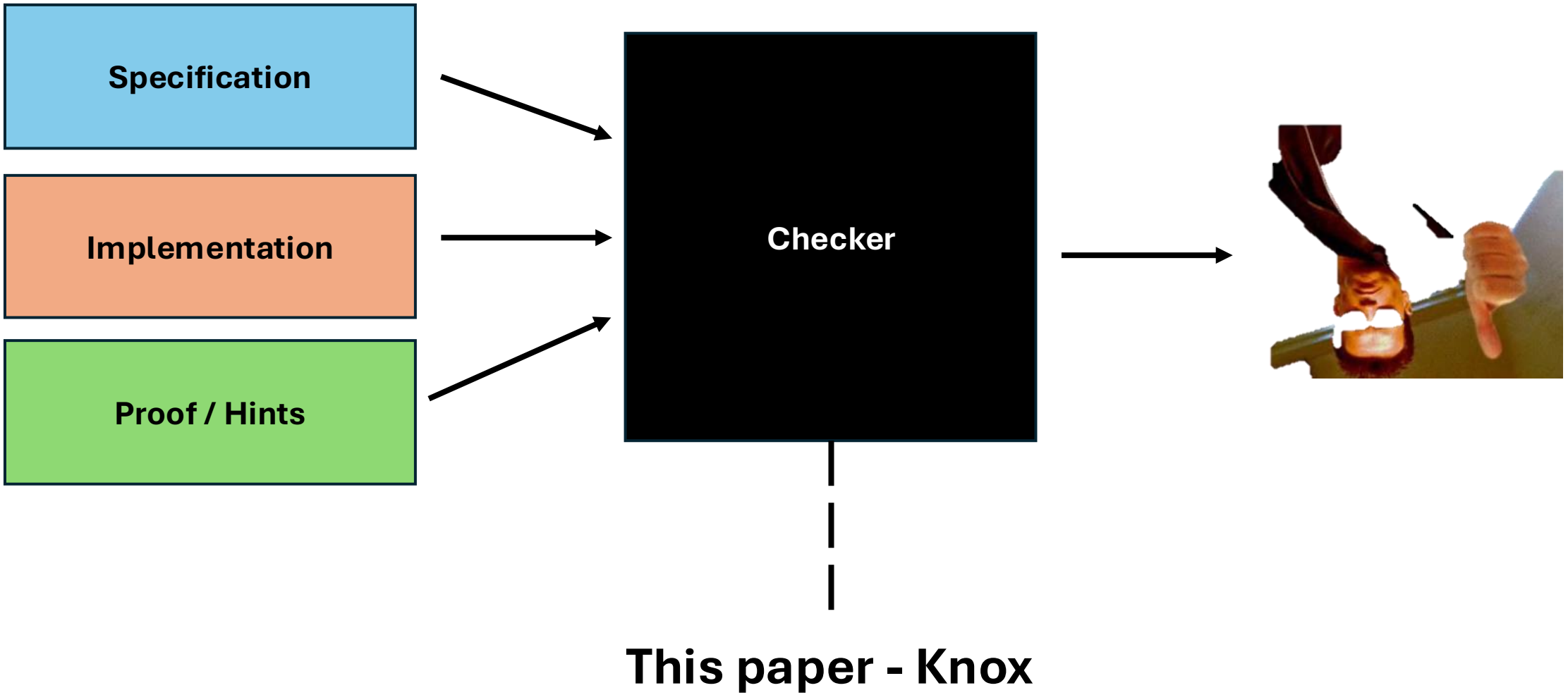
Formal Verification



Formal Verification



Formal Verification



Specifications

Specifications

High-level, Abstract

Specifications

High-level, Abstract

Python-like

Specifications

High-level, Abstract
Python-like

A PIN-protected backup HSM spec

```
var bad_guesses = 0, secret = 0, pin = 0

def store(new_secret, new_pin):
  secret = new_secret
  pin = new_pin
  bad_guesses = 0

def retrieve(guess):
  if bad_guesses >= 10:
    return 'No more guesses'
  if guess != pin:
    bad_guesses = bad_guesses + 1
    return 'Incorrect PIN'
  bad_guesses = 0
  return secret
```

Specifications

High-level, Abstract
Python-like

2 API operations

A PIN-protected backup HSM spec

```
var bad_guesses = 0, secret = 0, pin = 0

def store(new_secret, new_pin):
  secret = new_secret
  pin = new_pin
  bad_guesses = 0

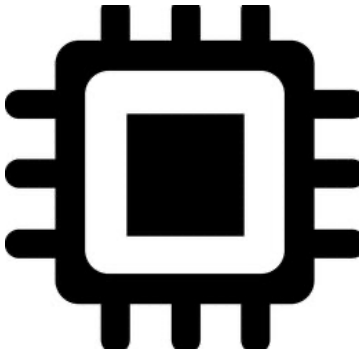
def retrieve(guess):
  if bad_guesses >= 10:
    return 'No more guesses'
  if guess != pin:
    bad_guesses = bad_guesses + 1
    return 'Incorrect PIN'
  bad_guesses = 0
  return secret
```

Implementations



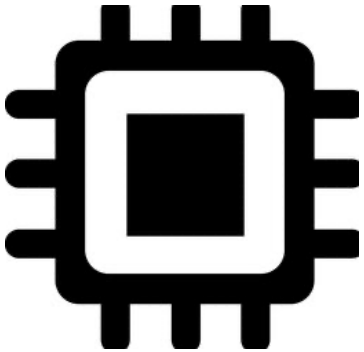
Implementations





Implementations

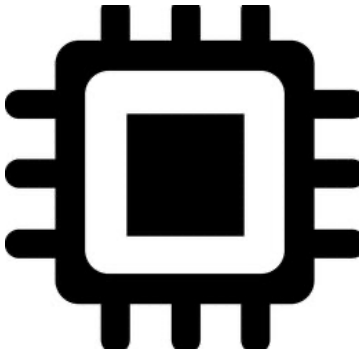




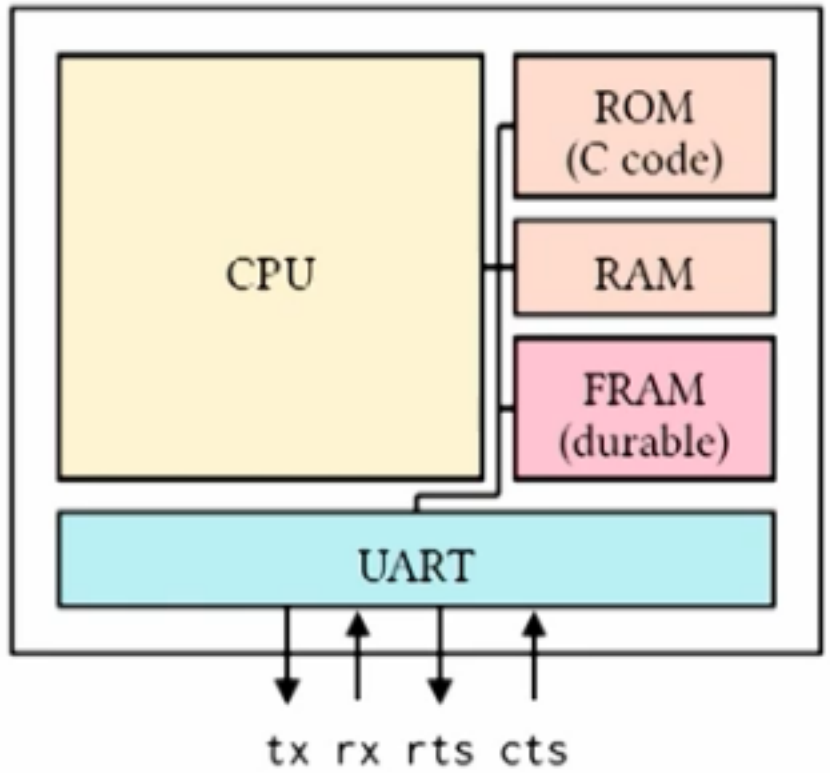


Implementations



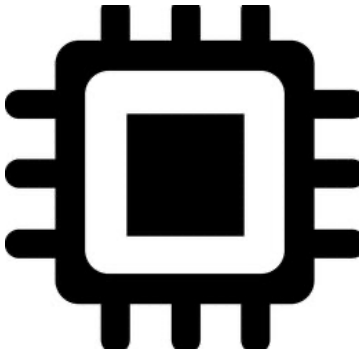






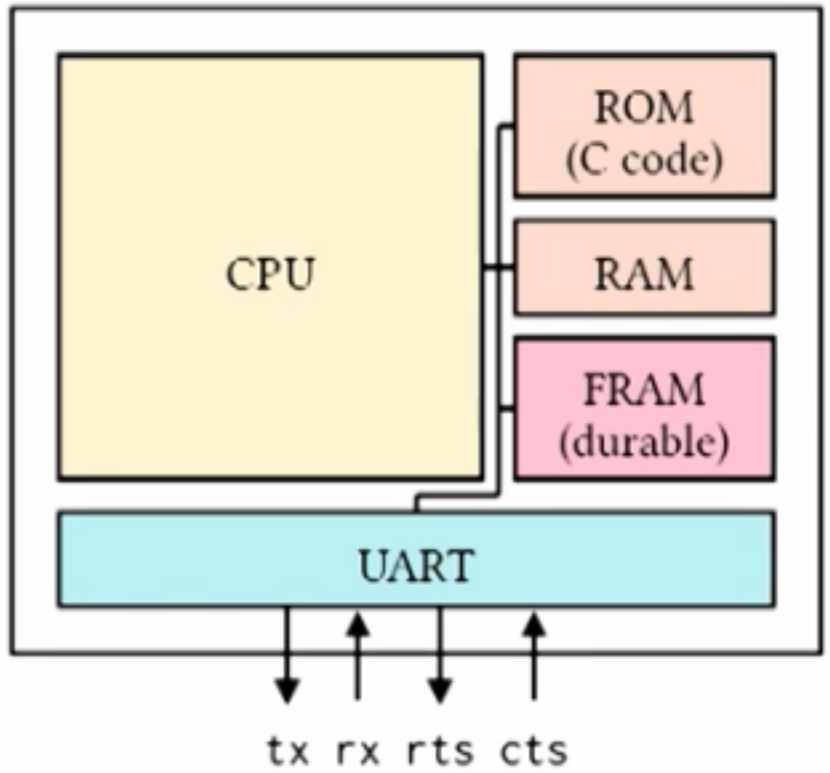
Implementations





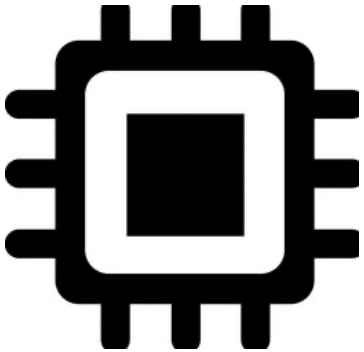


Device Interface:



Implementations

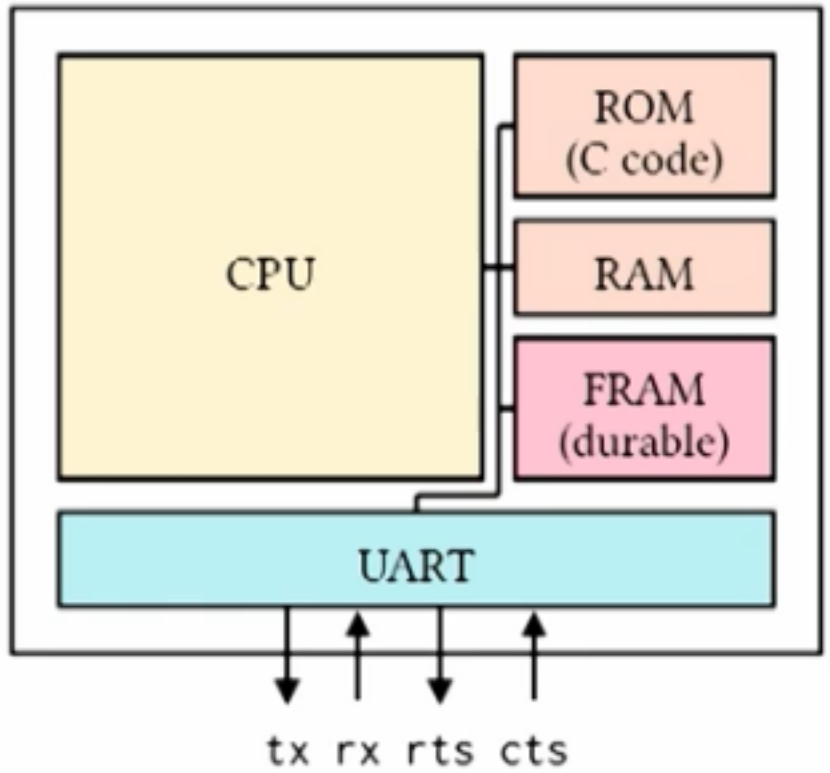






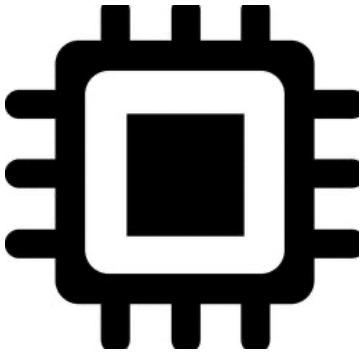
Device Interface:

- Input Wires



Implementations

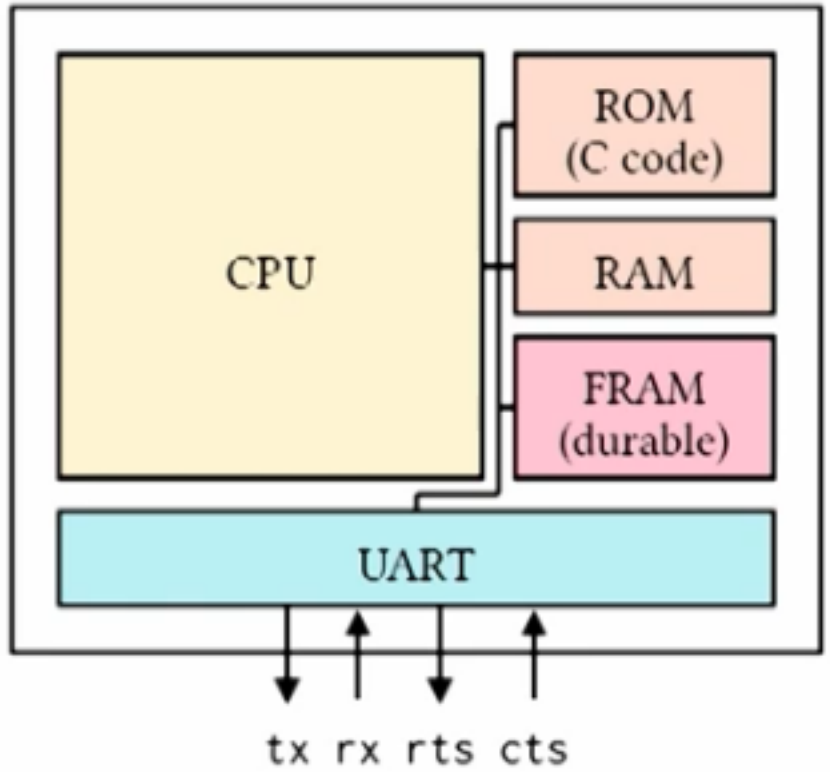






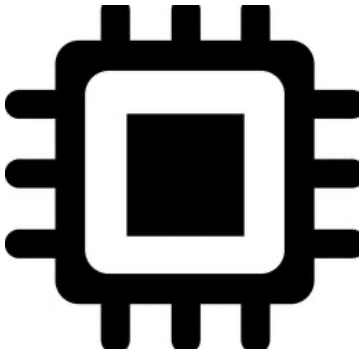
Device Interface:

- Input Wires
- Output Wires



Implementations

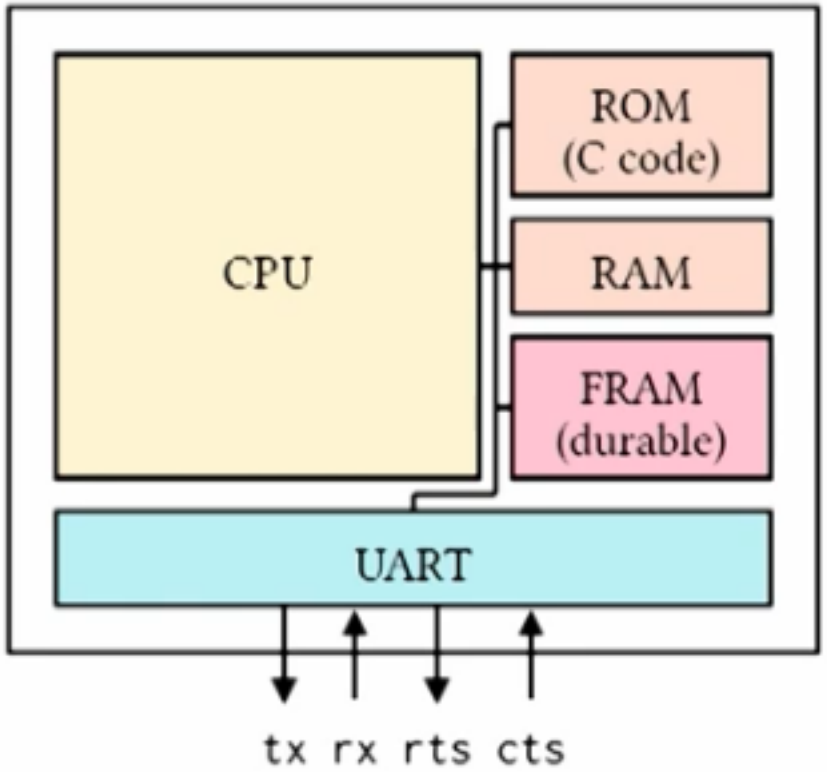






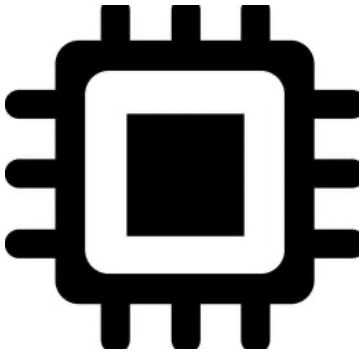
Device Interface:

- Input Wires
- Output Wires
- Clock tick



Implementations

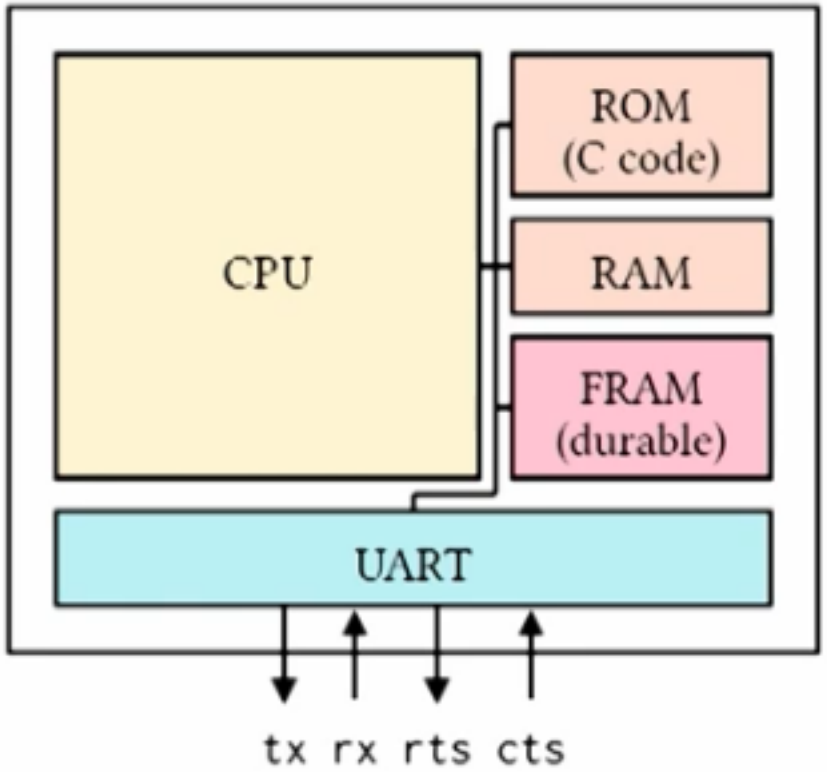






Device Interface:

- Input Wires
- Output Wires
- Clock tick
- Reset



To-do's

To-do's



To-do's



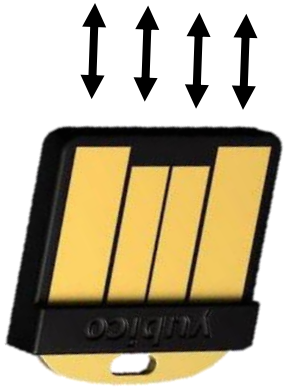


Functional Correctness



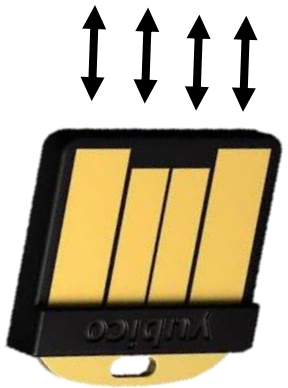
Functional Correctness

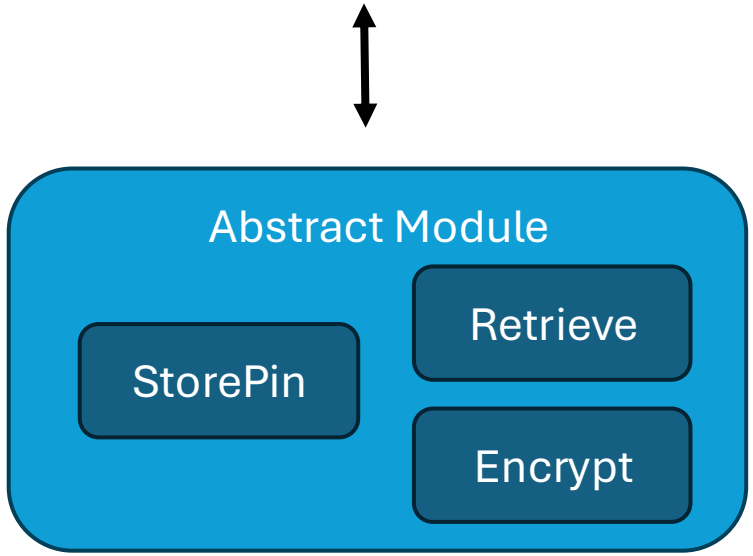




Functional Correctness

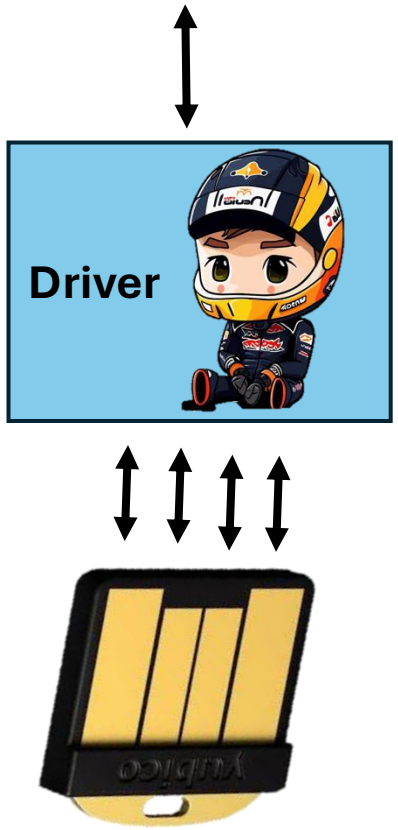


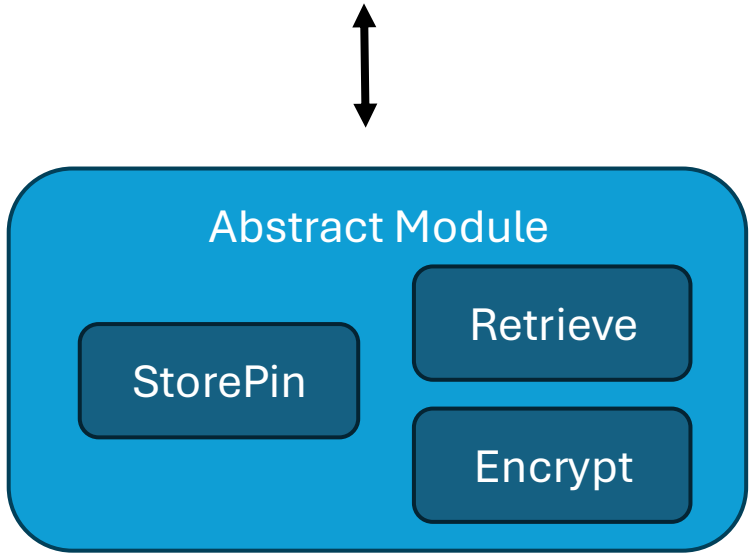




Functional Correctness

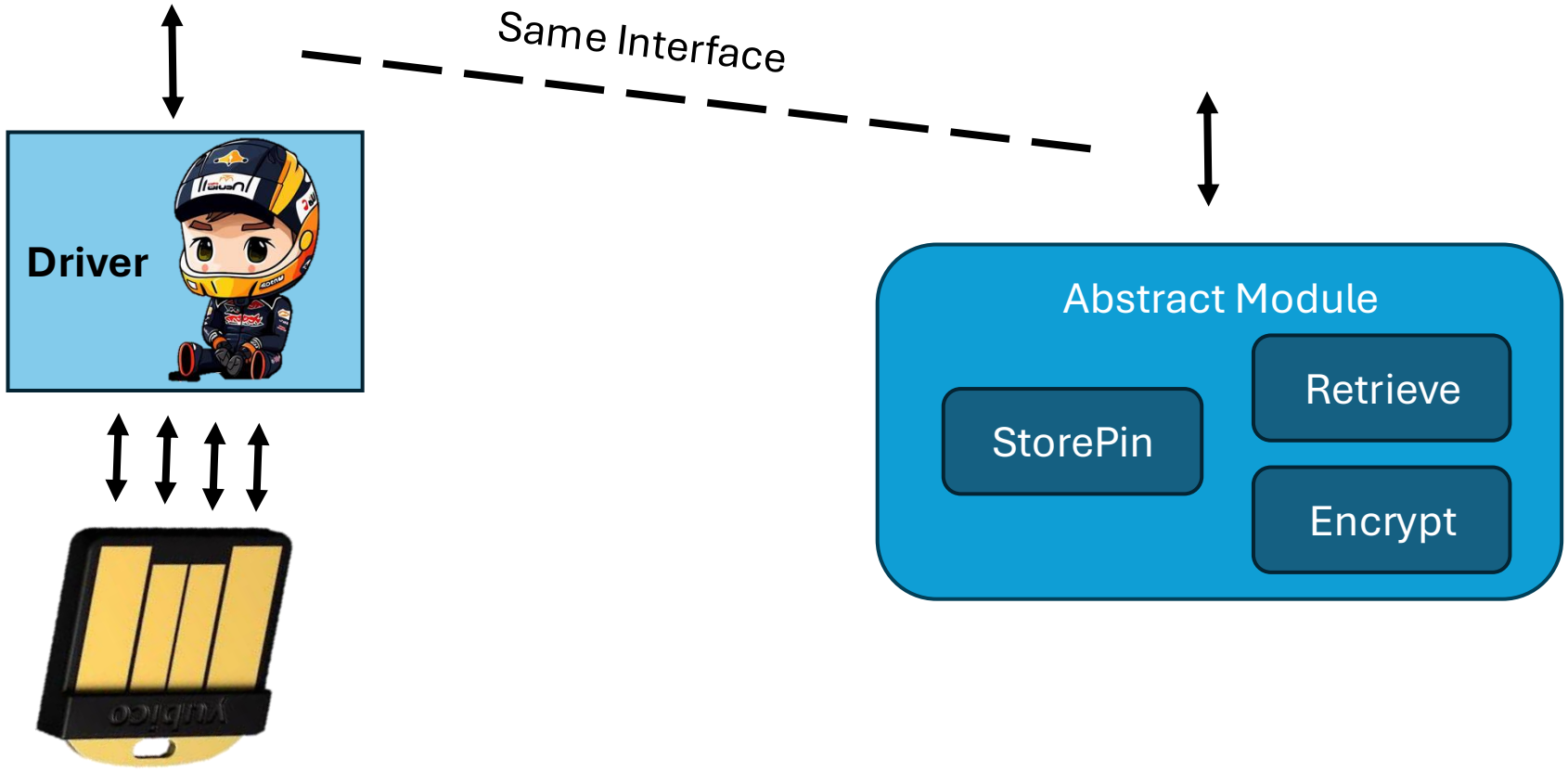






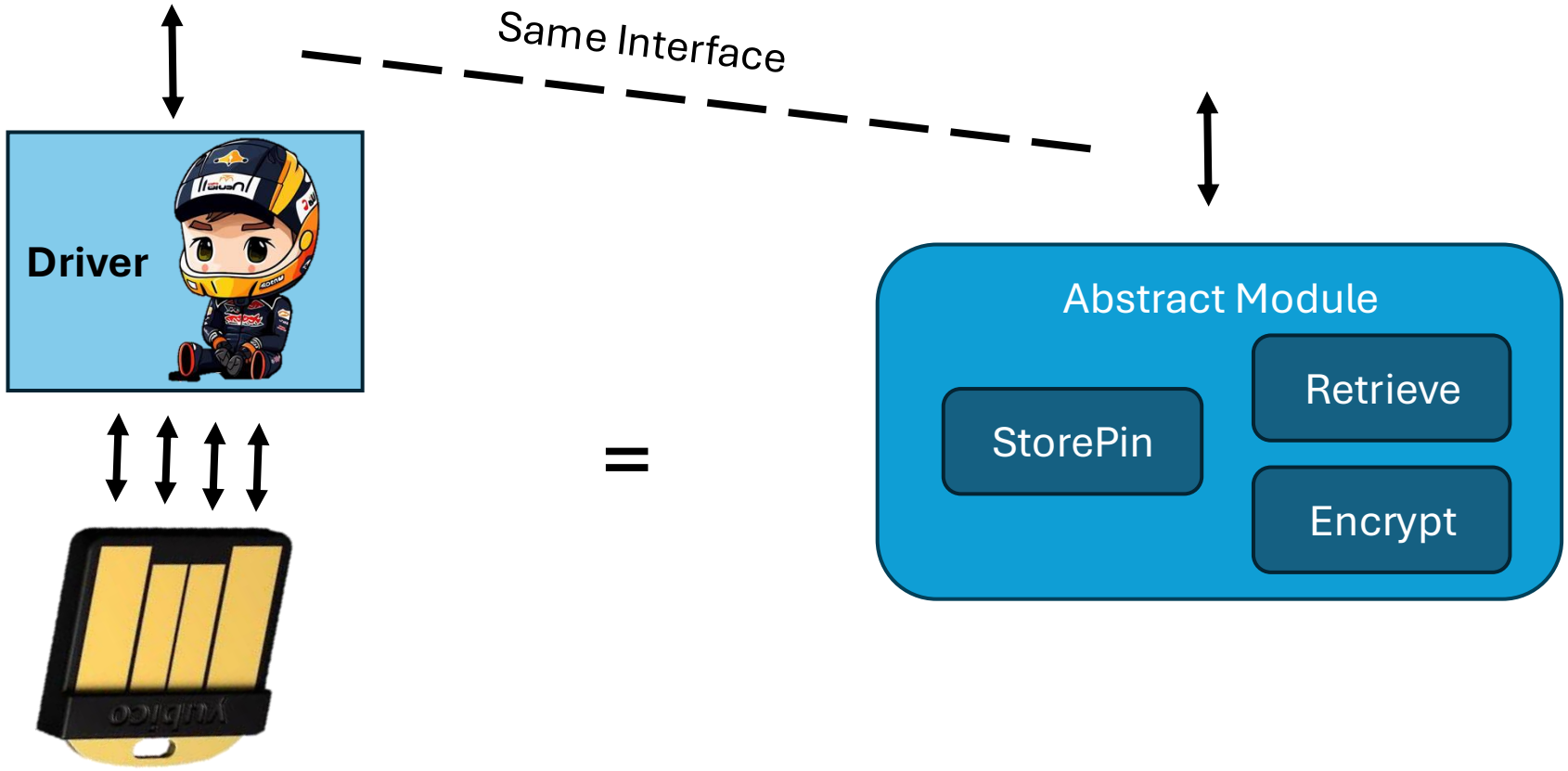
Functional Correctness





Functional Correctness





The Driver



The Driver



Expected way to use device

The Driver



Expected way to use device

Trusted - part of specification

The Driver



Expected way to use device

Trusted - part of specification

Translates spec-level operations to wire-level IO

The Driver



Expected way to use device

Trusted - part of specification

Translates spec-level operations to wire-level IO

```
(define (store secret pin)
    (send-byte #x02) ; command number
    (send-bytes pin)
    (send-bytes secret)
    (recv-byte)) ; wait for ack

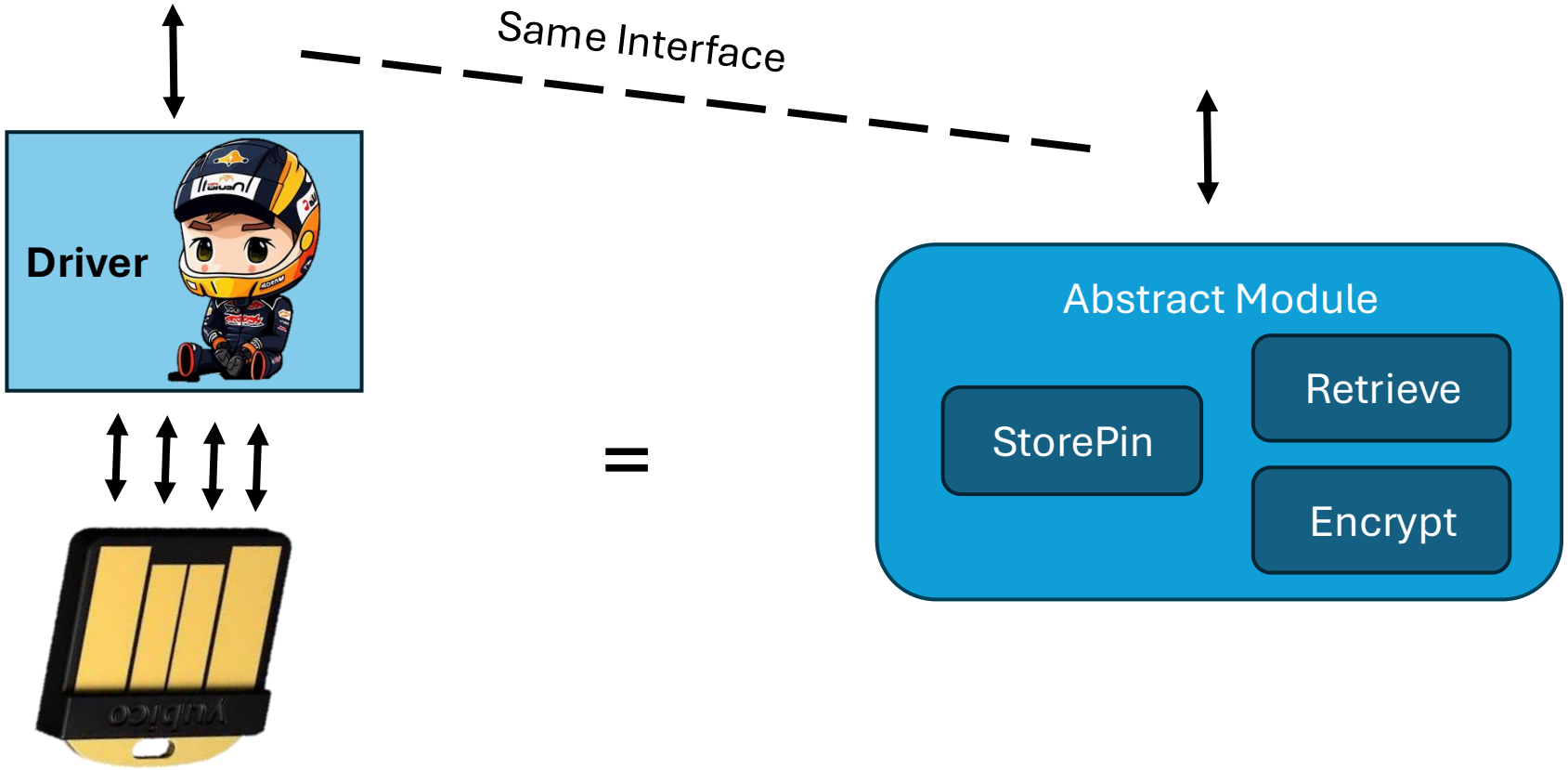
(define (wait-until-clear-to-send)
  (while (get-output 'rts))
    (tick))) ; wait a cycle

(define (send-bit bit)
  (set-input 'rx bit)
  (for ([i (in-range BAUD-RATE)])
    (tick)))

(define (send-byte byte)
  (wait-until-clear-to-send)
  (send-bit #b0) ; send start bit
  ;; send data bits
  (for ([i (in-range 8)])
    (send-bit (extract-bit byte i)))
  (send-bit #b1)) ; send stop bit
```

Functional Correctness





Functional Correctness



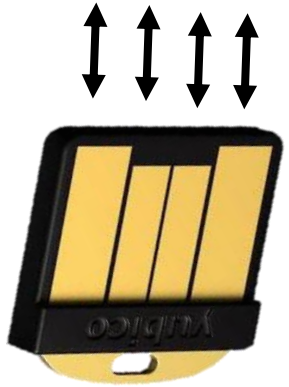
```
(define (store secret pin)
    (send-byte #x02) ; command number
    (send-bytes pin)
    (send-bytes secret)
    (recv-byte)) ; wait for ack

(define (wait-until-clear-to-send)
  (while (get-output 'rts))
    (tick))) ; wait a cycle

(define (send-bit bit)
  (set-input 'rx bit)
  (for ([i (in-range BAUD-RATE)])
    (tick)))

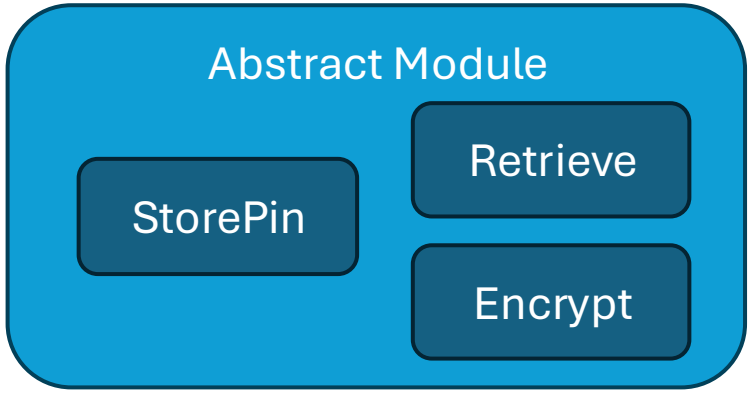
(define (send-byte byte)
  (wait-until-clear-to-send)
  (send-bit #b0) ; send start bit
  ;; send data bits
  (for ([i (in-range 8)])
    (send-bit (extract-bit byte i)))
  (send-bit #b1)) ; send stop bit
```





Same Interface

=

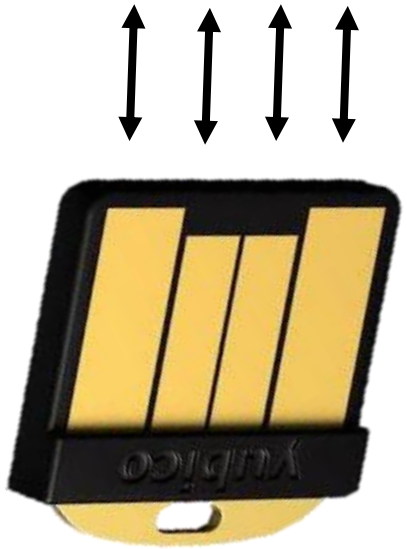


Nonleakage



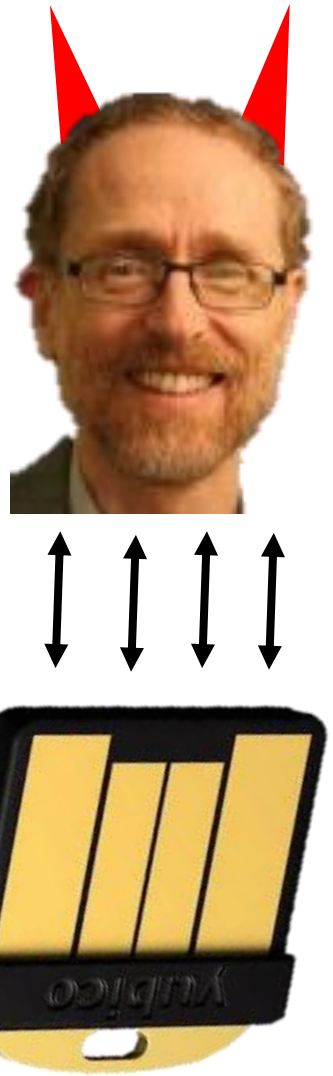
Nonleakage





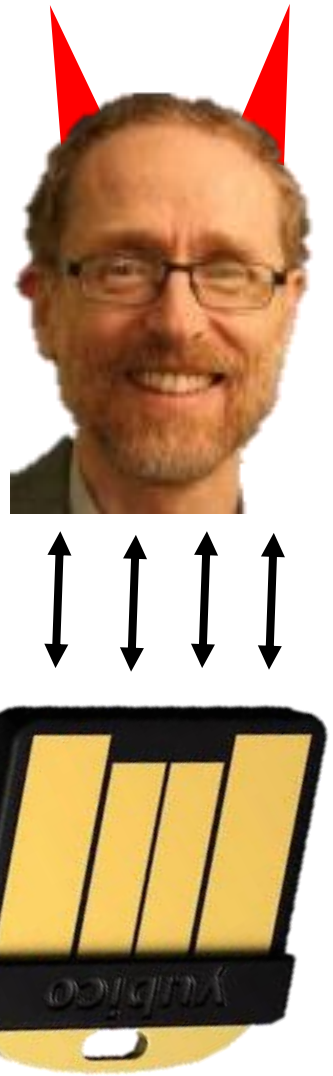
Nonleakage





Nonleakage





??

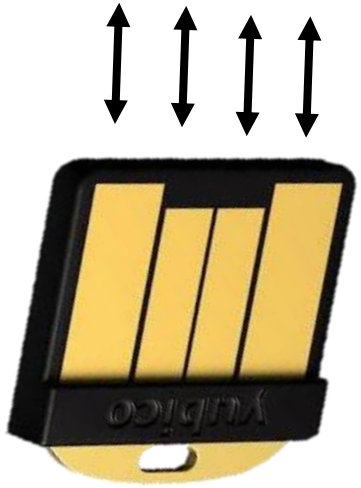
Real Ideal paradigm

Real Ideal paradigm

“Anything you can achieve using wire-level operations, you can achieve using API operations”

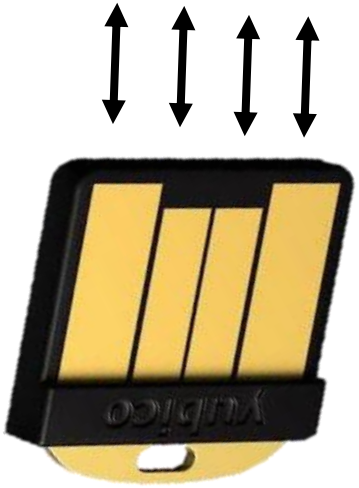
Real Ideal paradigm

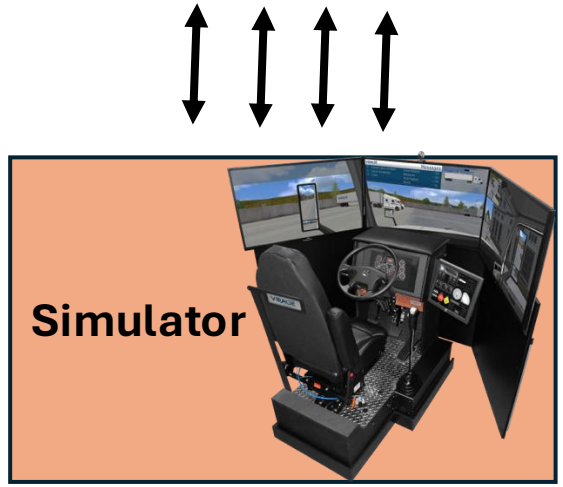
“Anything you can achieve using wire-level operations, you can achieve using API operations”



Real Ideal paradigm

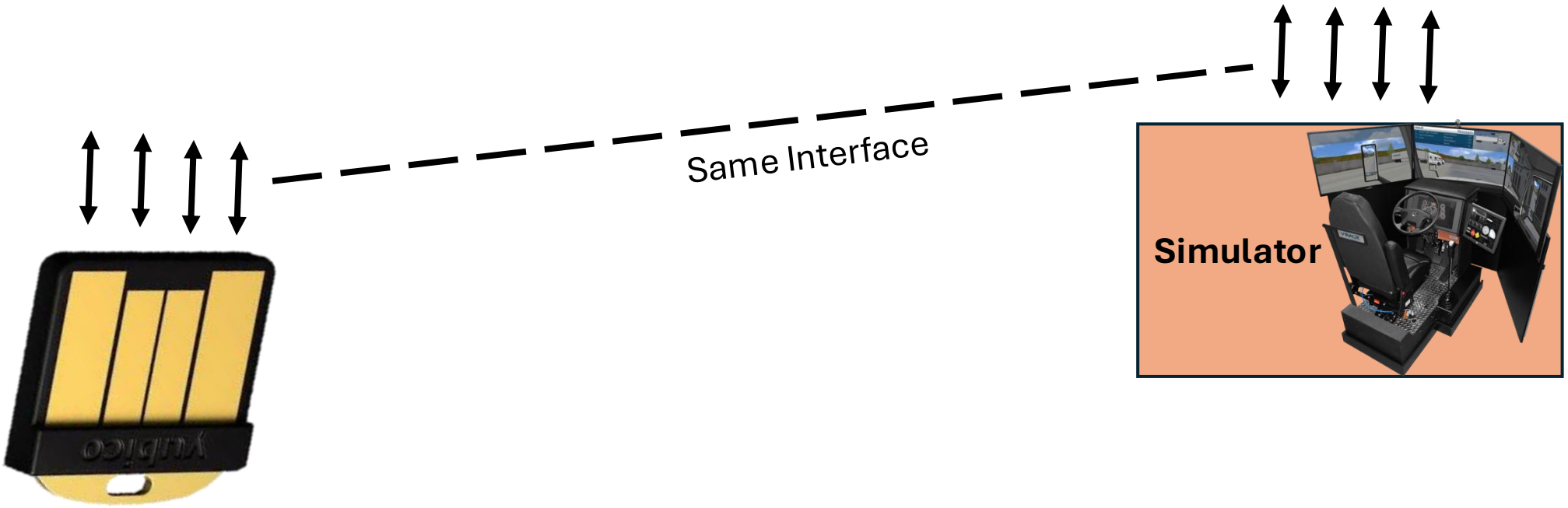
“Anything you can achieve using wire-level operations, you can achieve using API operations”





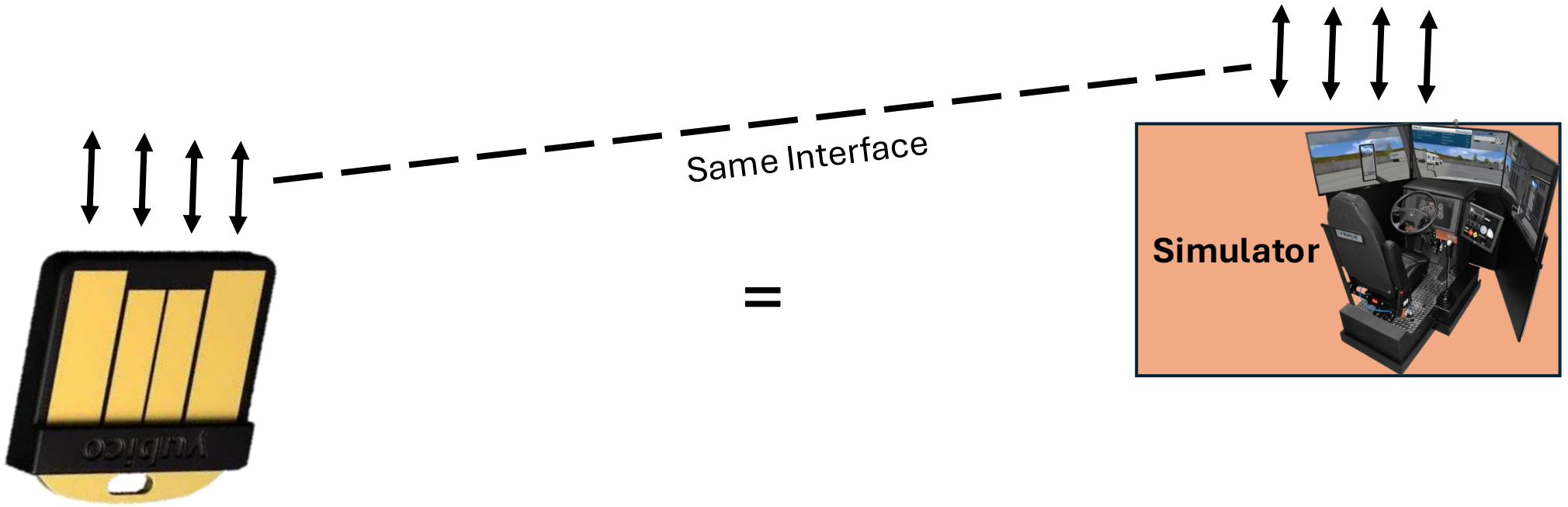
Real Ideal paradigm

“Anything you can achieve using wire-level operations, you can achieve using API operations”



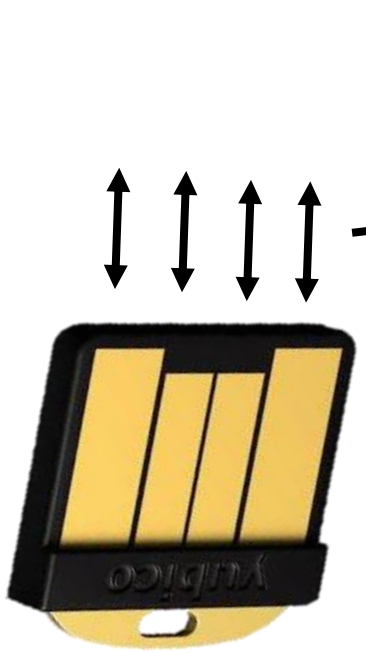
Real Ideal paradigm

“Anything you can achieve using wire-level operations, you can achieve using API operations”



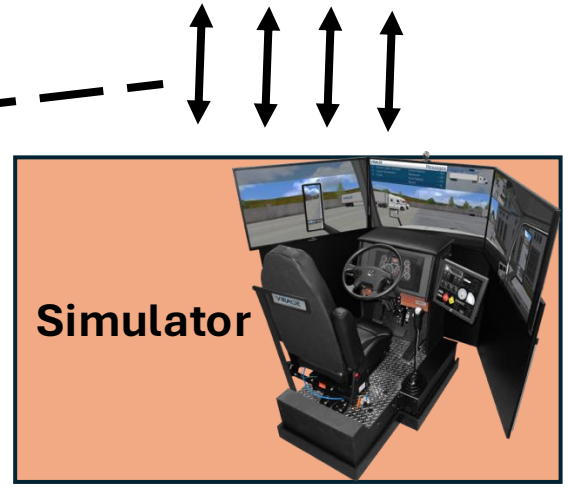
Real Ideal paradigm

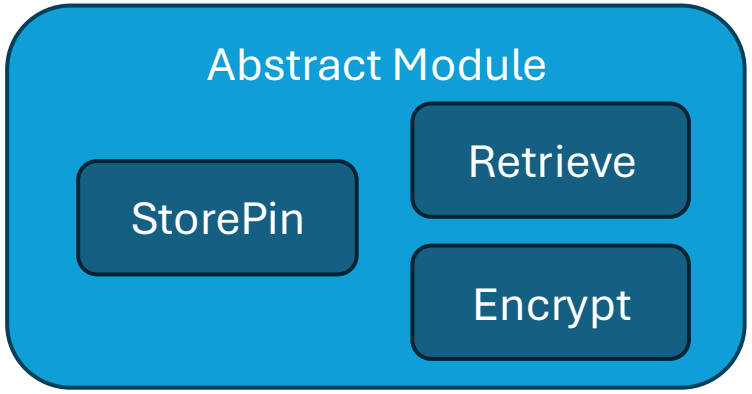
“Anything you can achieve using wire-level operations, you can achieve using API operations”



Same Interface

=





How does this help?

How does this help?

```
var password

def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```



1234

9876 → False

1231 → False





Simulator



p?

```
def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```

1234

9876 → False

1231 → False





Simulato

p?

```
def check_pw(guess):
  if len(guess) != len(password):
    return False

  for i in range(len(password)):
    if password[i] != guess[i]:
      return False

  return True
```

1234

9876 → False

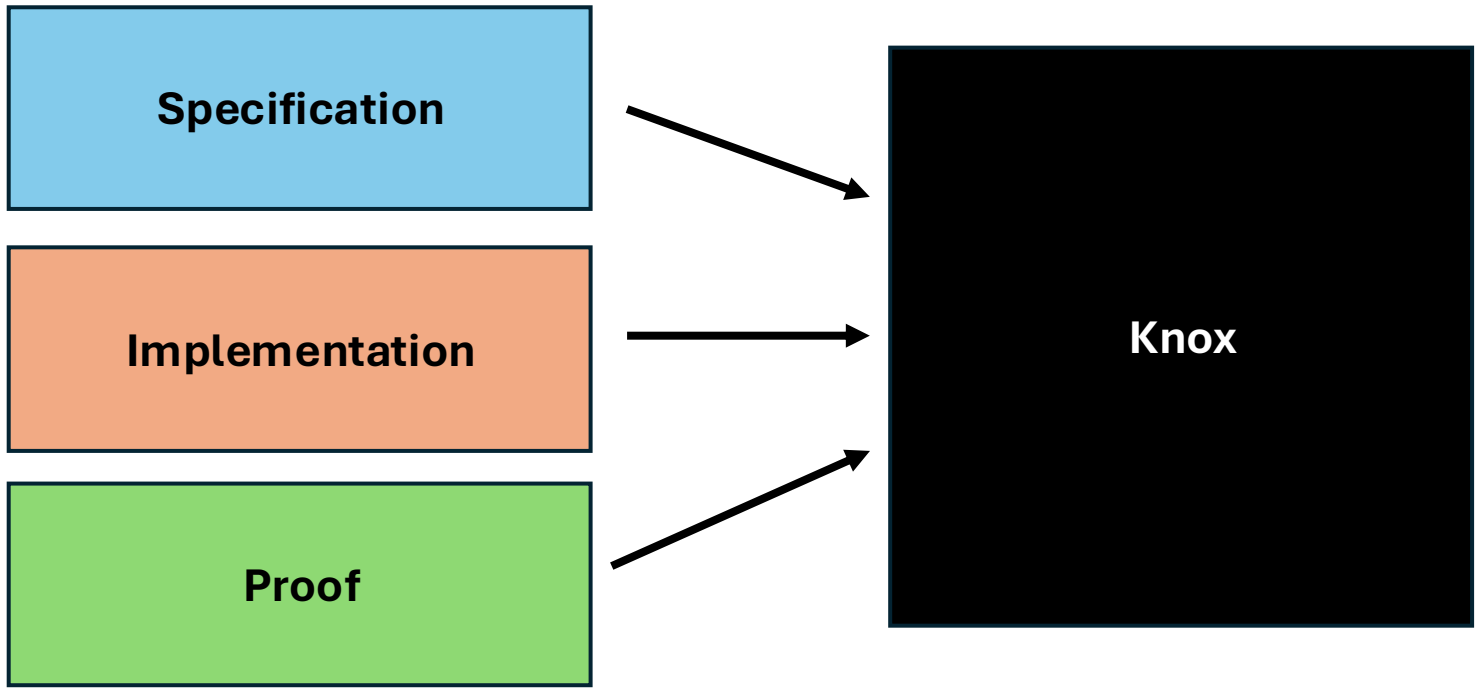
1231 → False



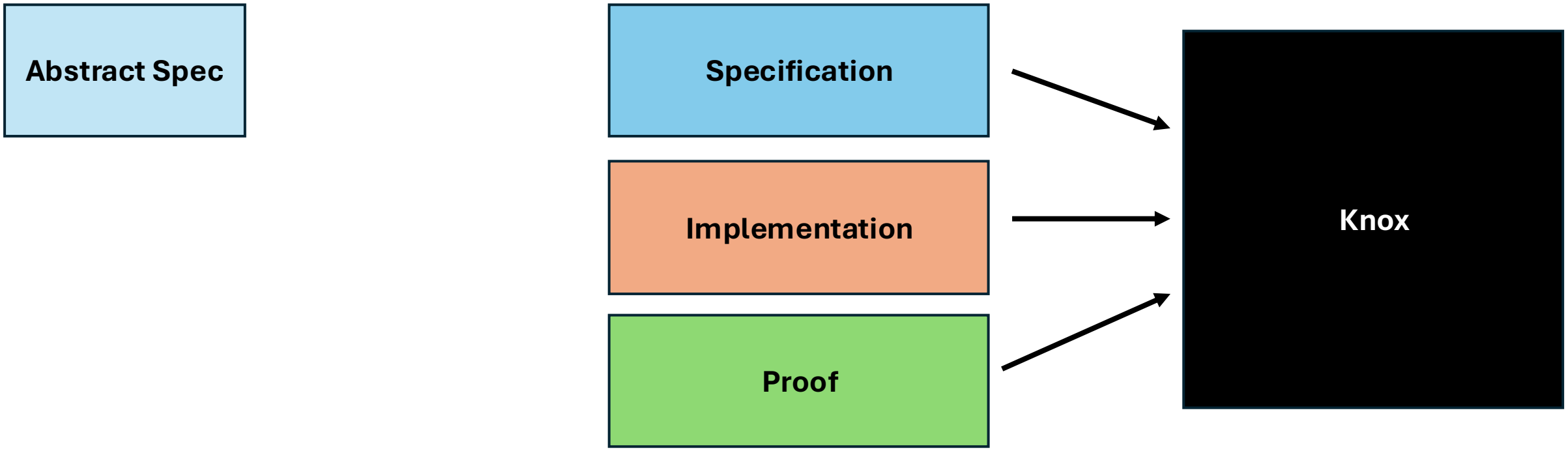


Knox

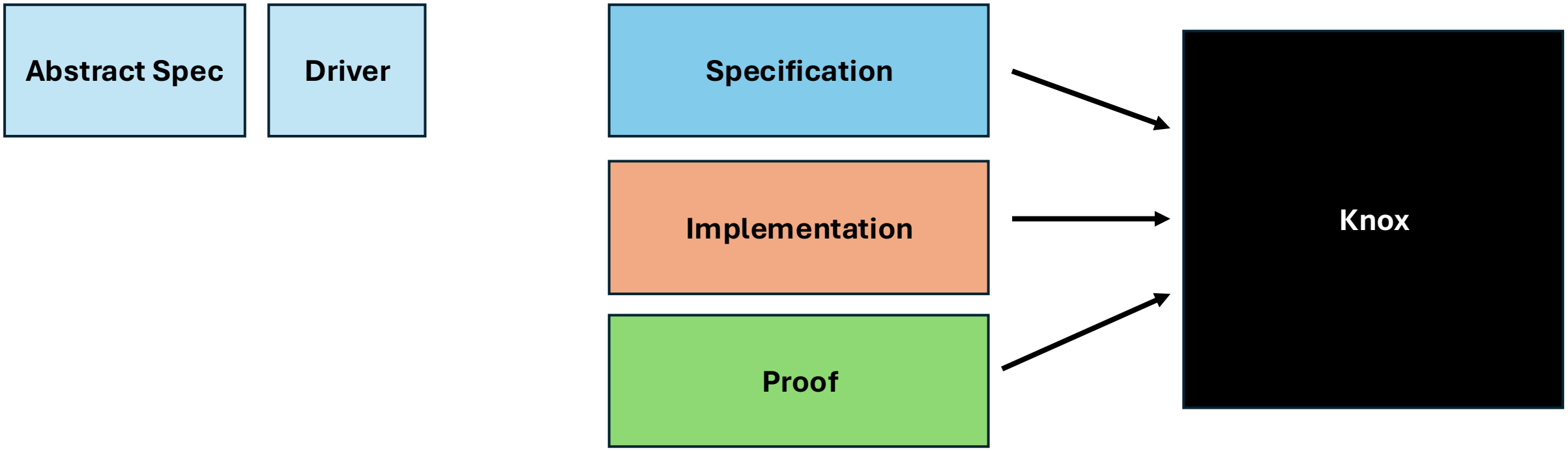
Knox



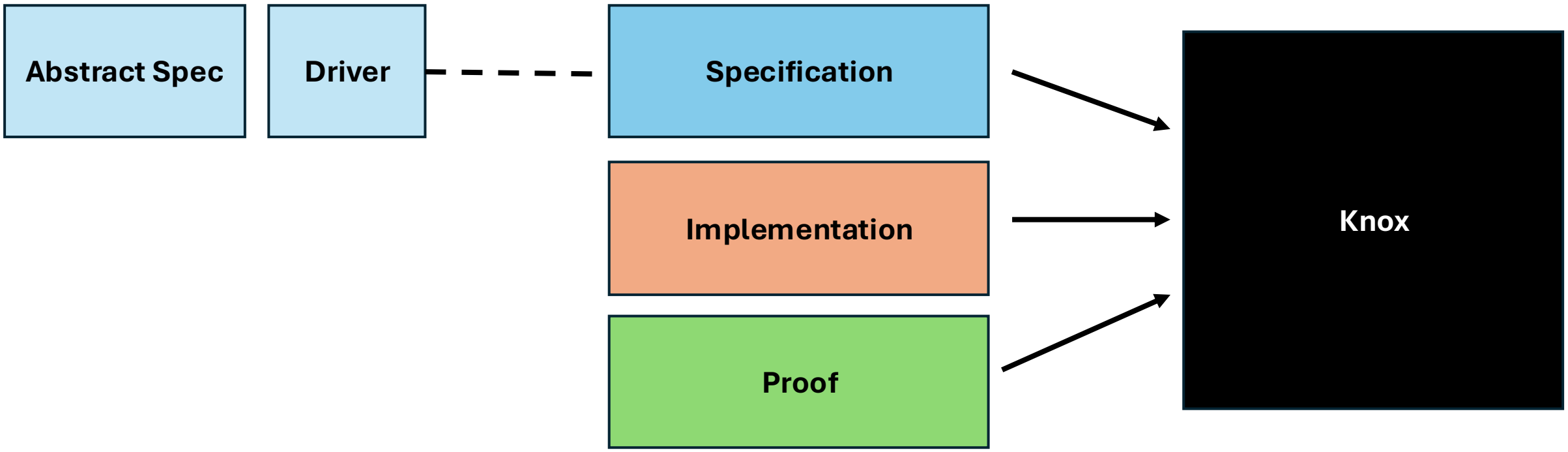
Knox



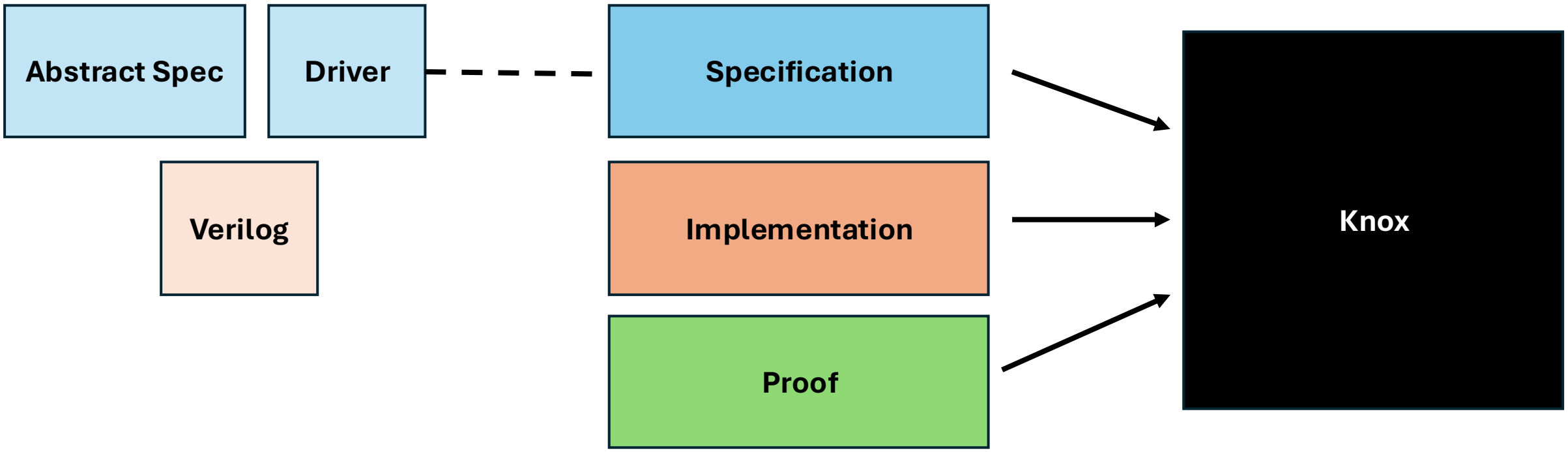
Knox


Abstract Spec
Driver
Specification
Implementation
Proof
Knox

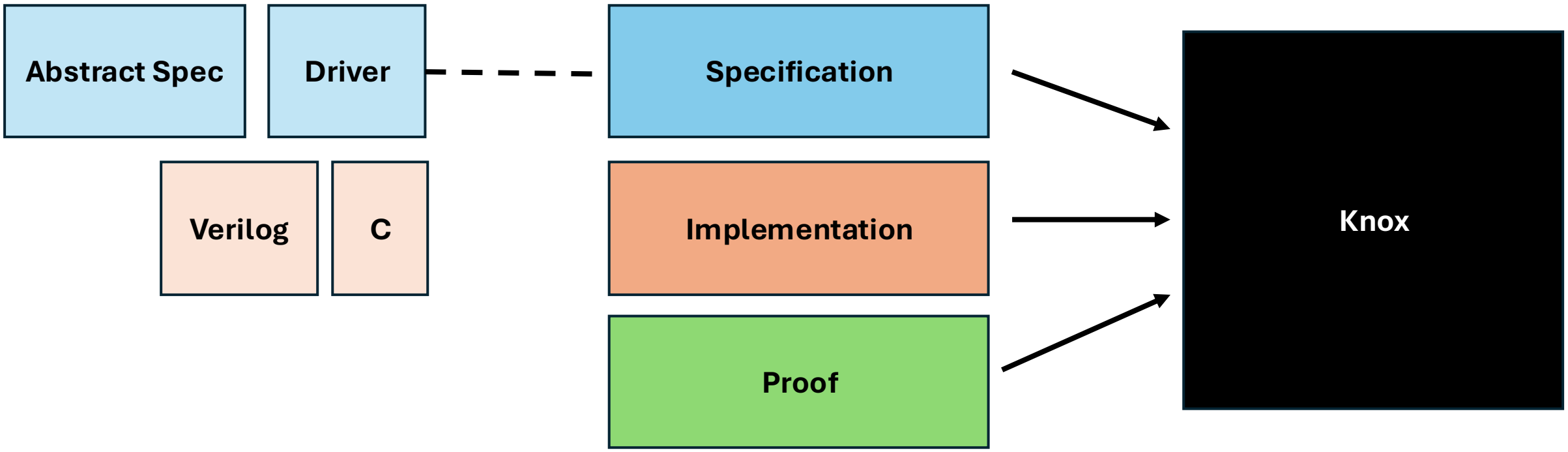
Knox


Abstract Spec
Driver
Specification
Implementation
Proof
Knox

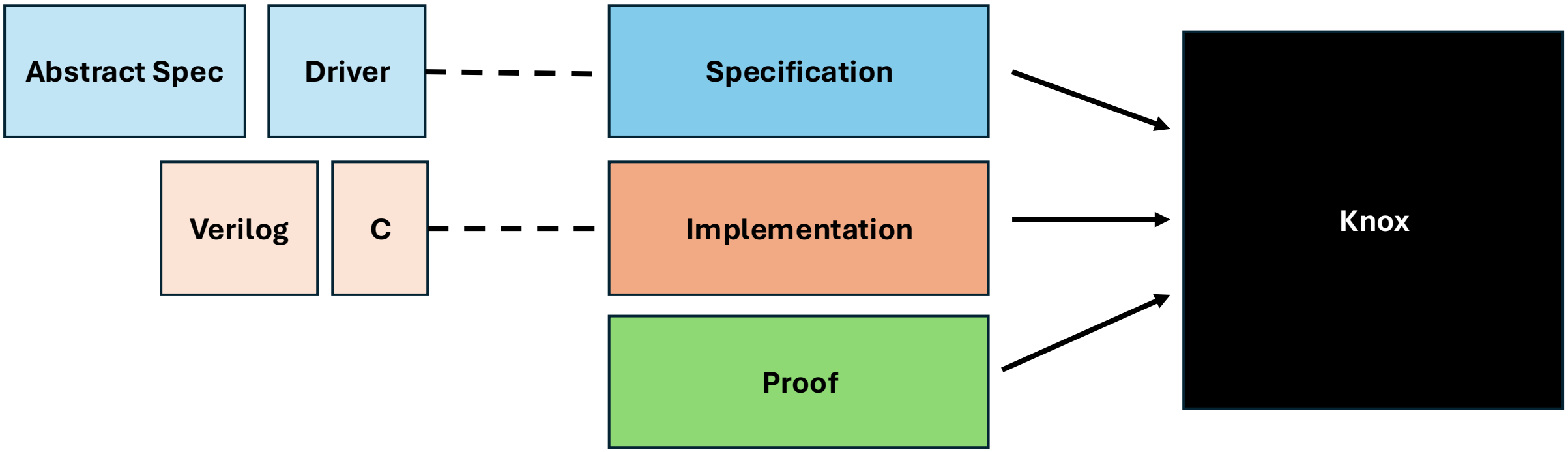
Knox


Abstract Spec
Driver
Verilog
Specification
Implementation
Proof
Knox

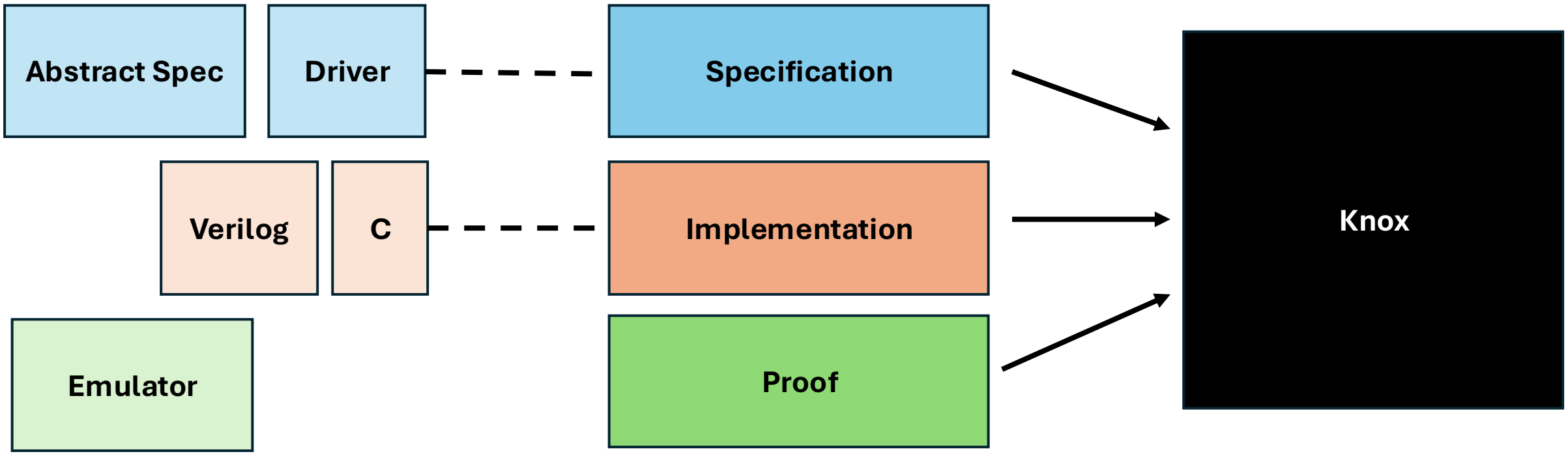
Knox


Abstract Spec
Driver
Verilog
C
Specification
Implementation
Proof
Knox

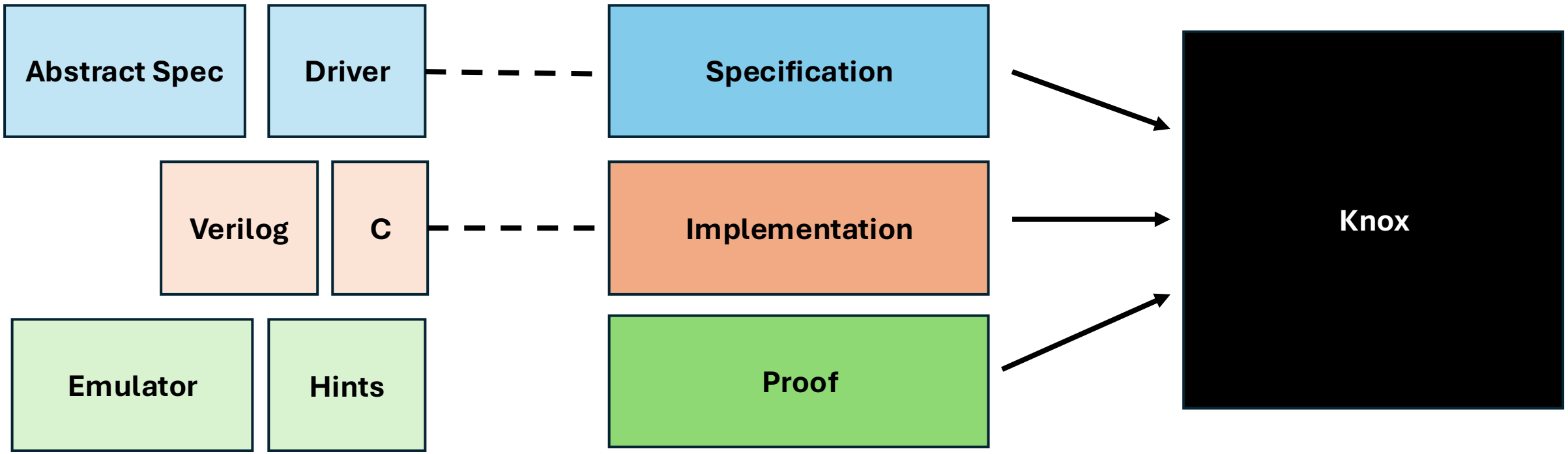
Knox


Abstract Spec
Driver
Specification
Verilog
C
Implementation
Proof
Knox

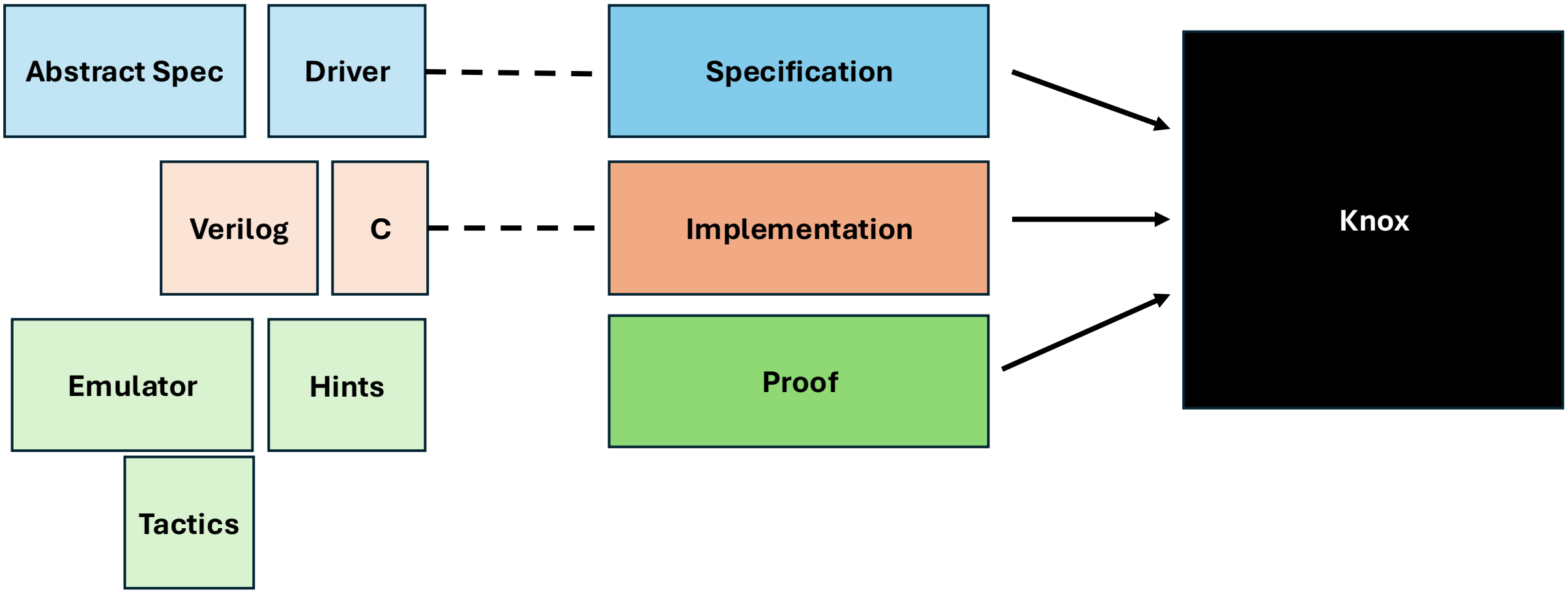
Knox


Abstract Spec
Driver
Specification
Verilog
C
Implementation
Emulator
Proof
Knox

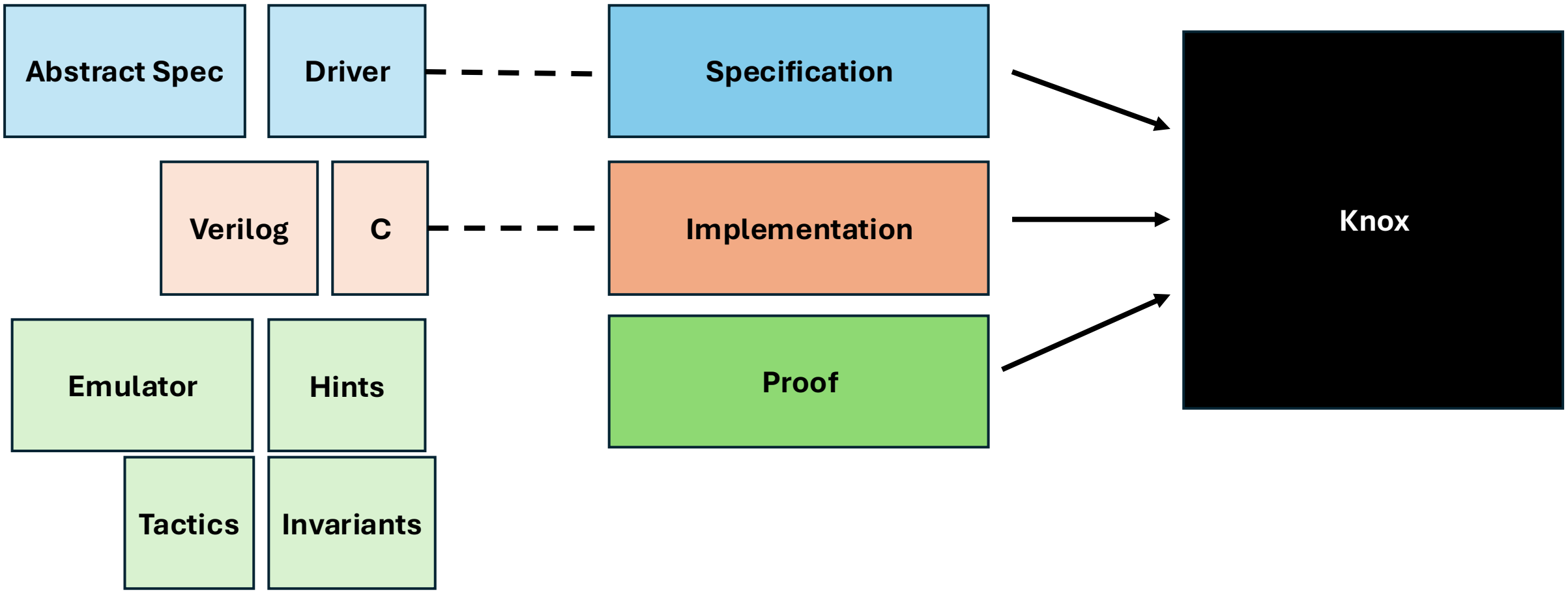
Knox


Abstract Spec
Driver
Specification
Verilog
C
Implementation
Emulator
Hints
Proof
Knox

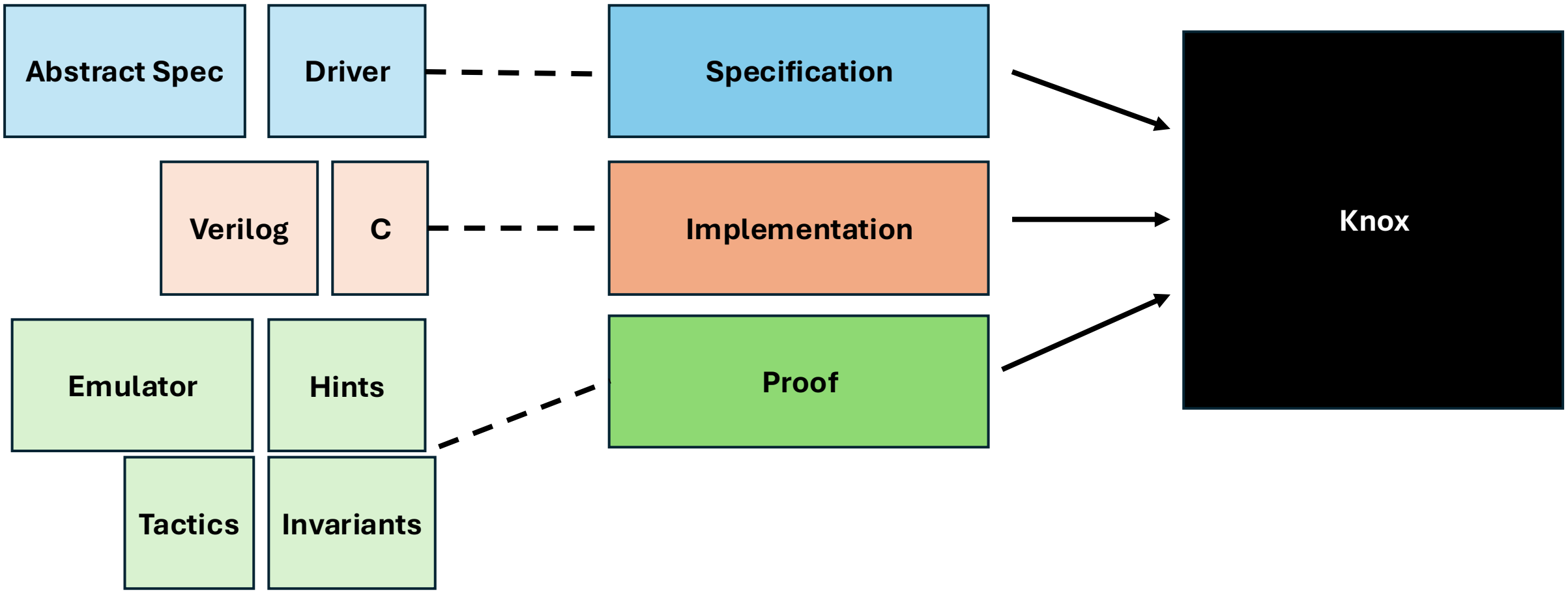
Knox


Abstract Spec
Driver
Specification
Verilog
C
Implementation
Emulator
Hints
Proof
Tactics
Knox

Knox


Abstract Spec
Driver
Specification
Verilog
C
Implementation
Emulator
Hints
Proof
Tactics
Invariants
Knox

Knox


Abstract Spec
Driver
Specification
Verilog
C
Implementation
Emulator
Hints
Proof
Tactics
Invariants
Knox

Knox

Knox

~3000 lines of code, using Rosette [PLDI ‘14]

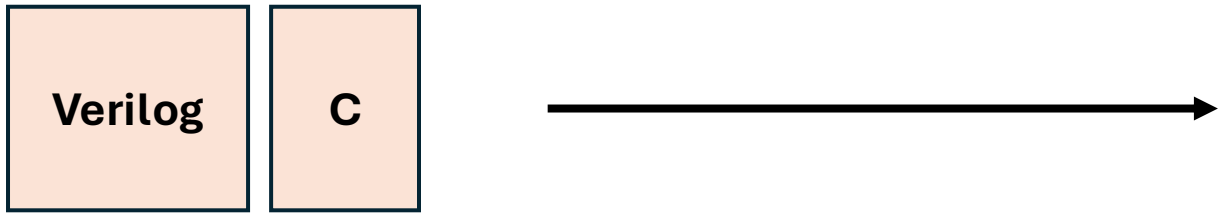
Knox

~3000 lines of code, using Rosette [PLDI ‘14]



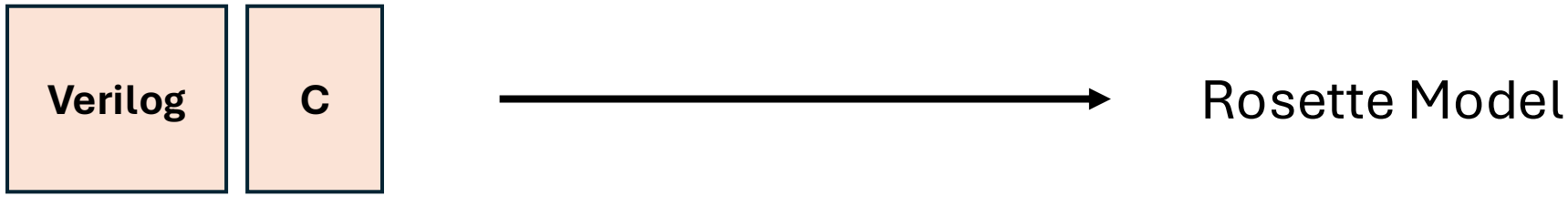
Knox

~3000 lines of code, using Rosette [PLDI ‘14]



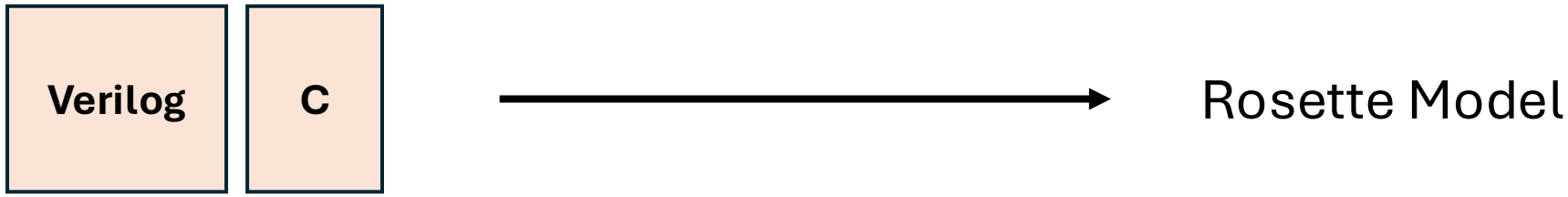
Knox

~3000 lines of code, using Rosette [PLDI ‘14]



Knox

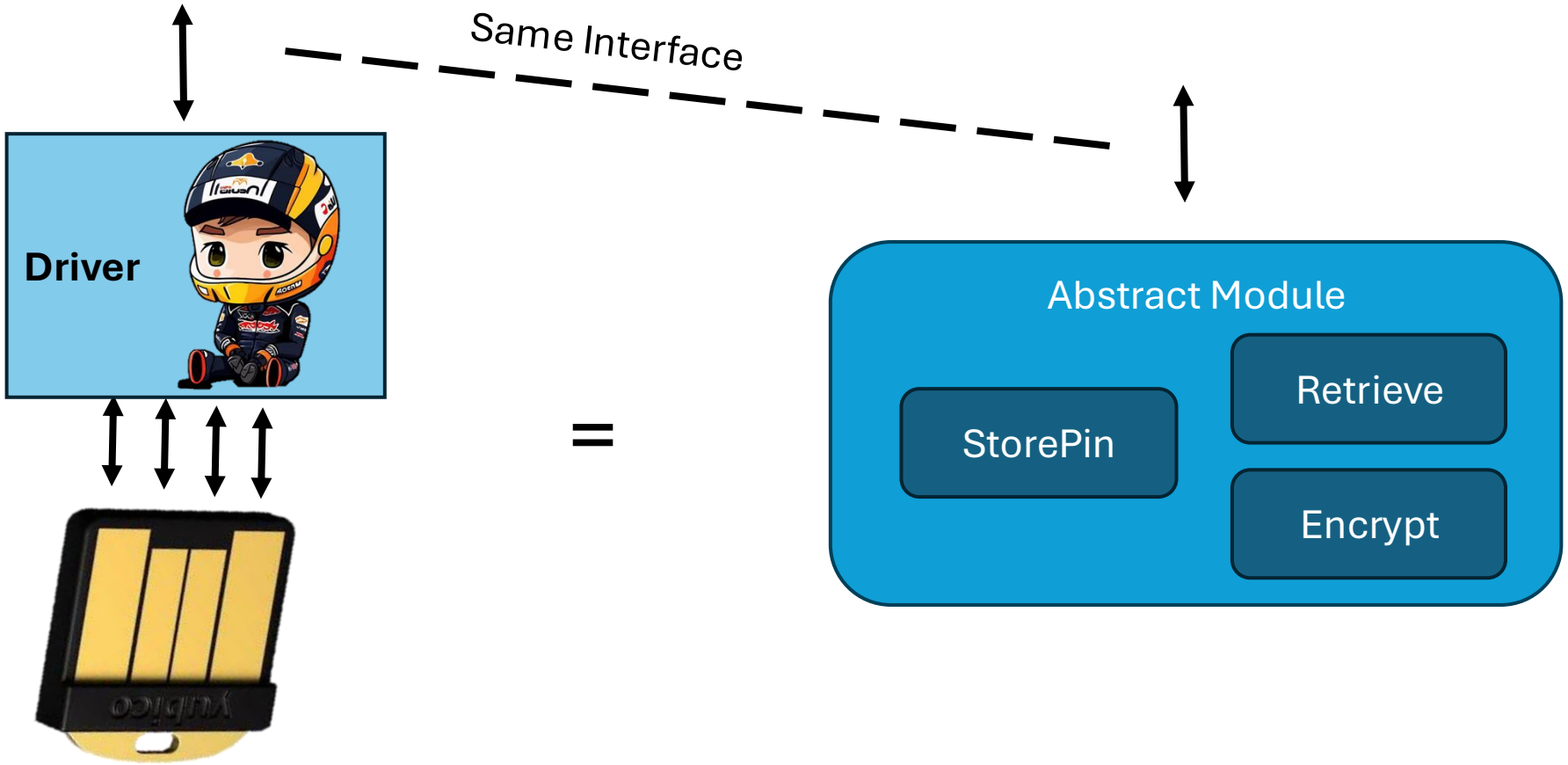
~3000 lines of code, using Rosette [PLDI ‘14]



Symbolic Execution to ‘run’ the entire circuit!

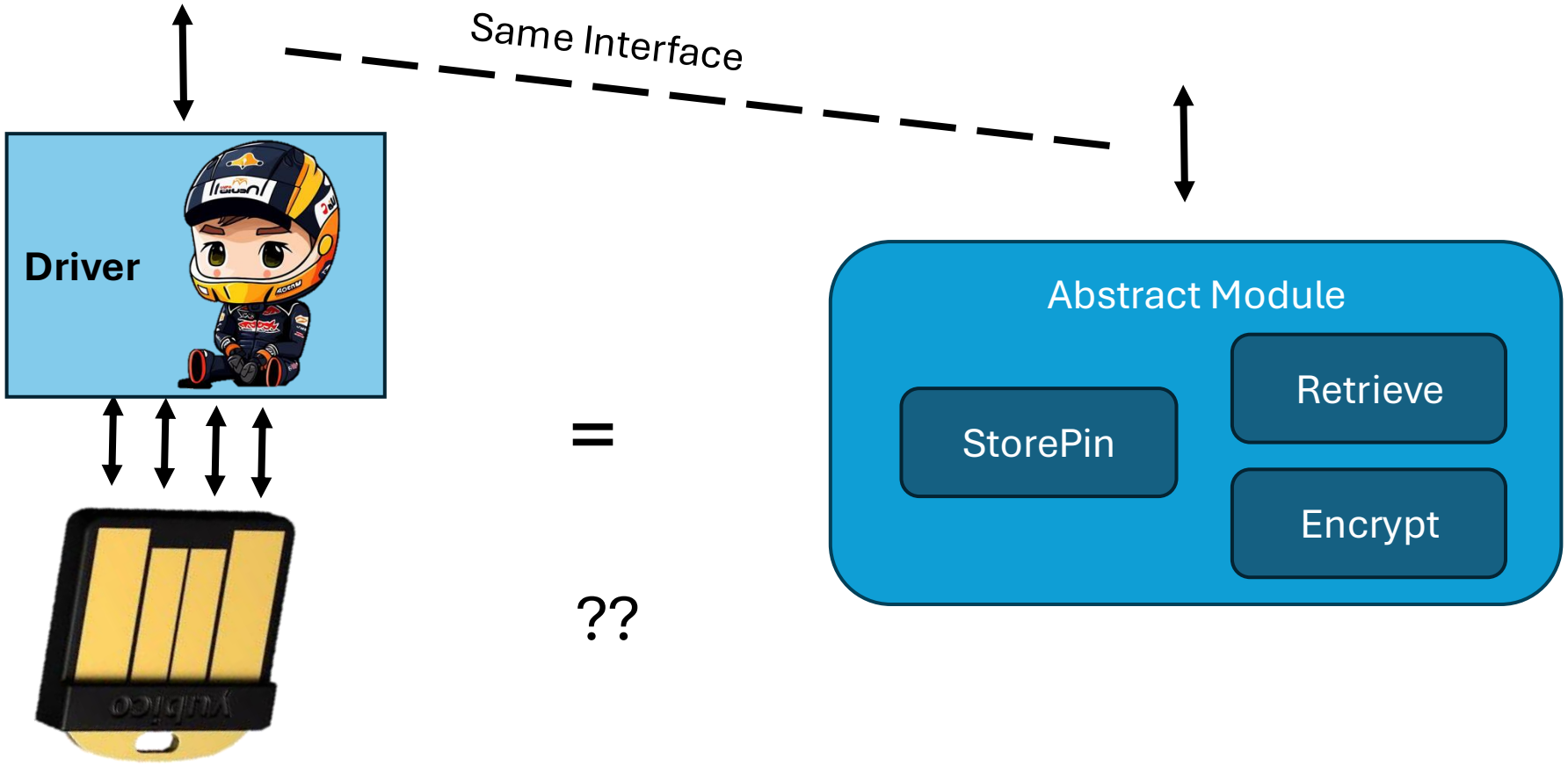
Functional Correctness





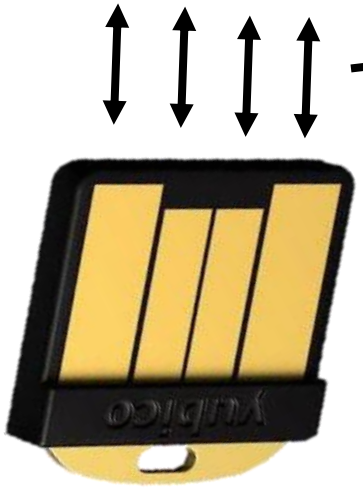
Functional Correctness





Nonleakage

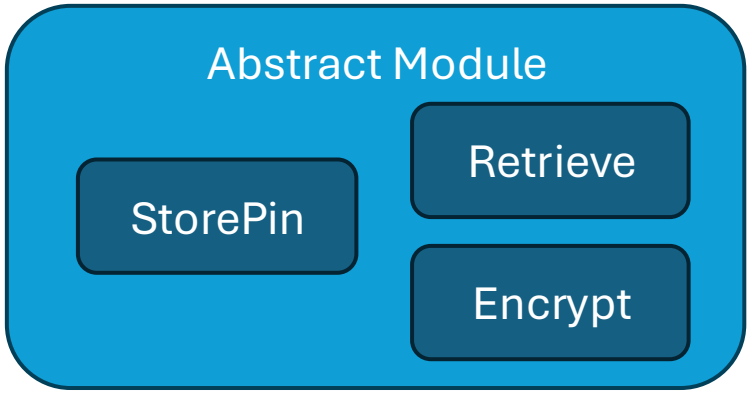




Same Interface

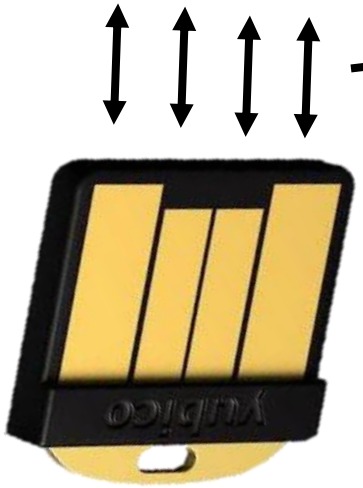
=





Nonleakage



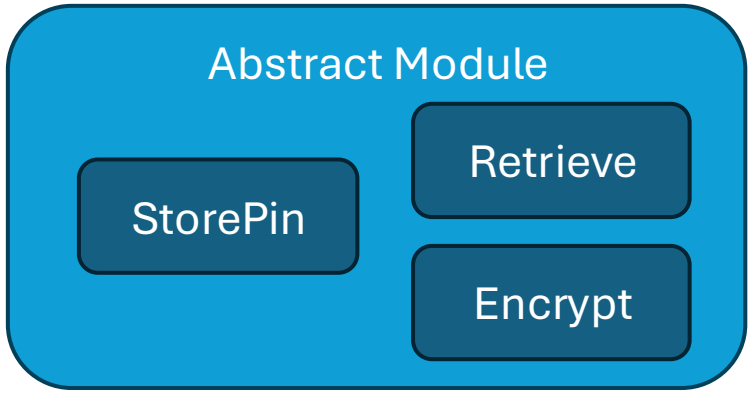


Same Interface

=

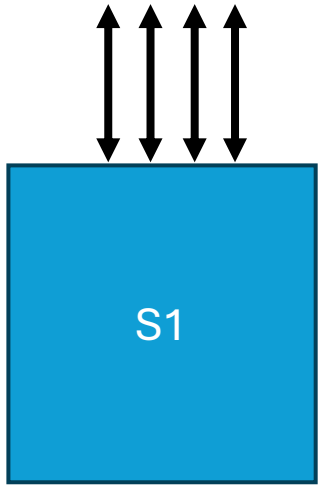
??



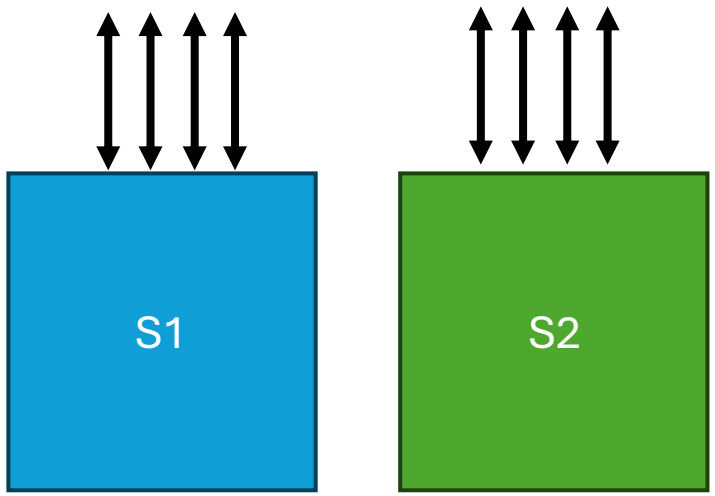


System Equivalence

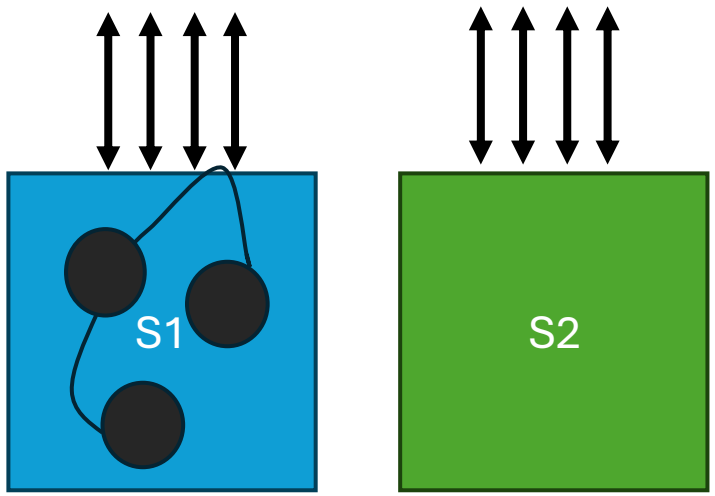
System Equivalence



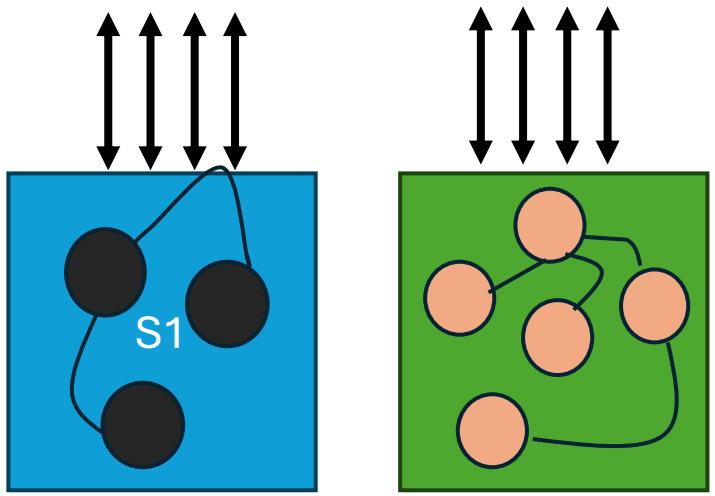
System Equivalence



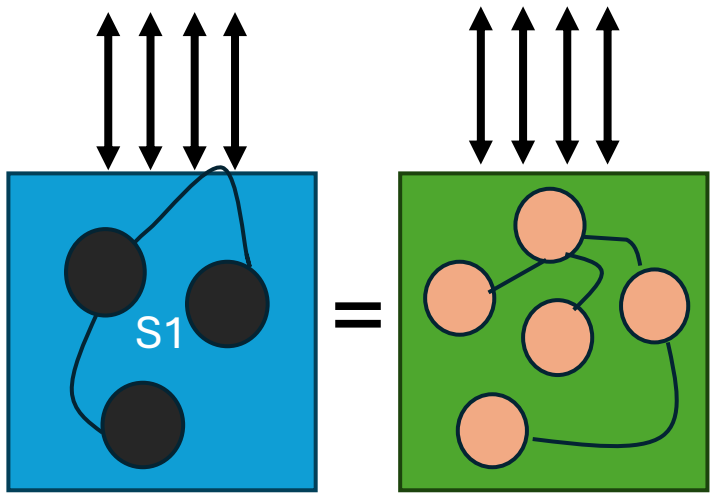
System Equivalence



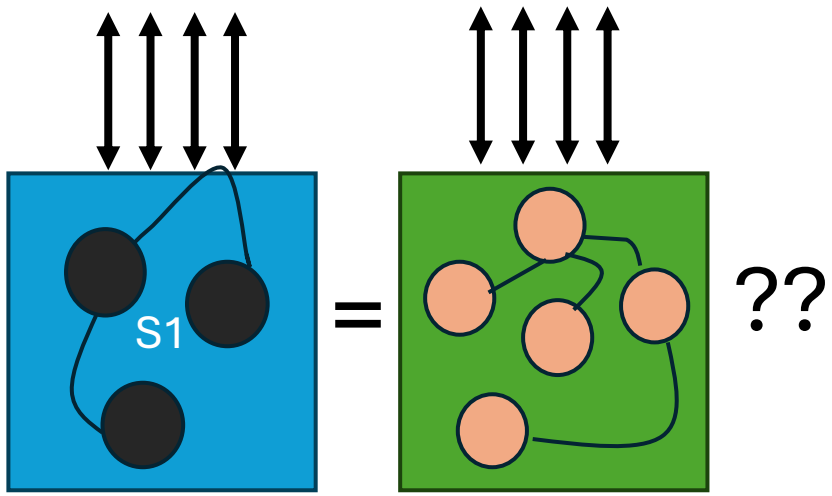
System Equivalence



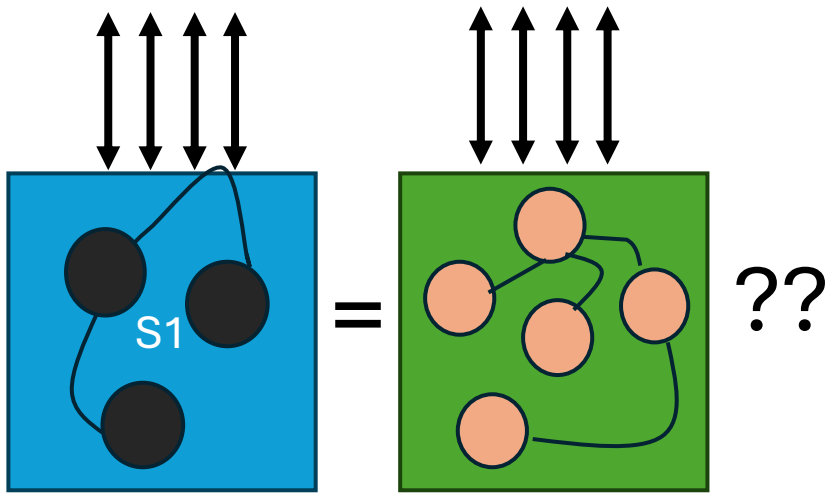
System Equivalence



System Equivalence

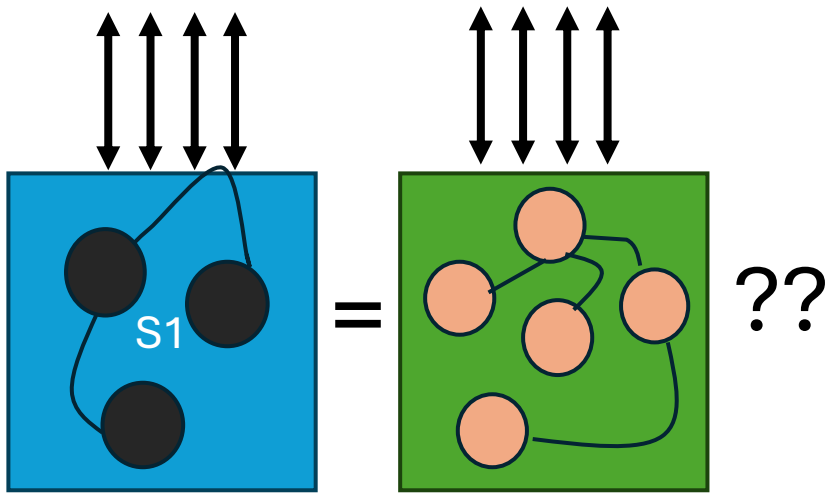


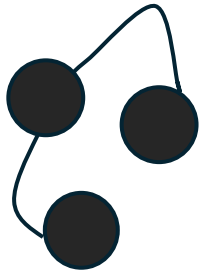
System Equivalence

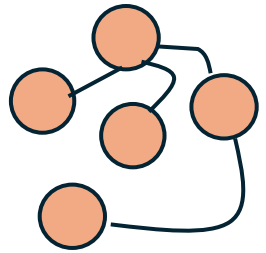


Bisimulation

System Equivalence

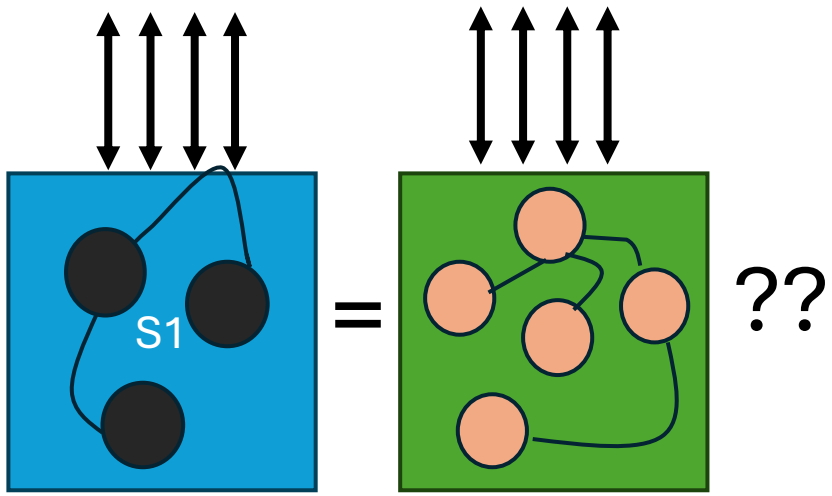


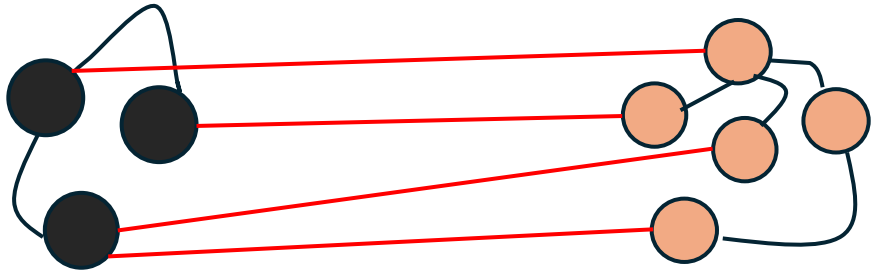




Bisimulation

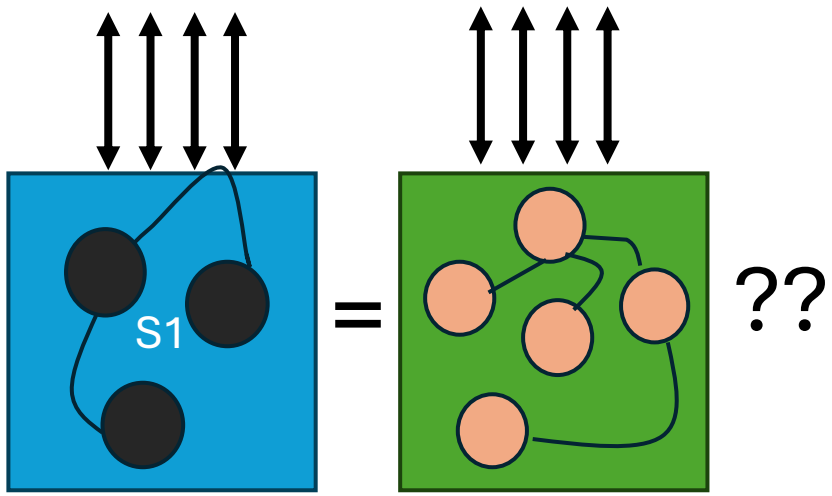
System Equivalence





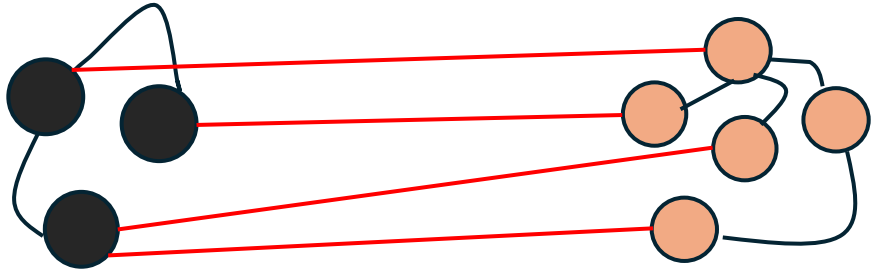
Bisimulation

System Equivalence

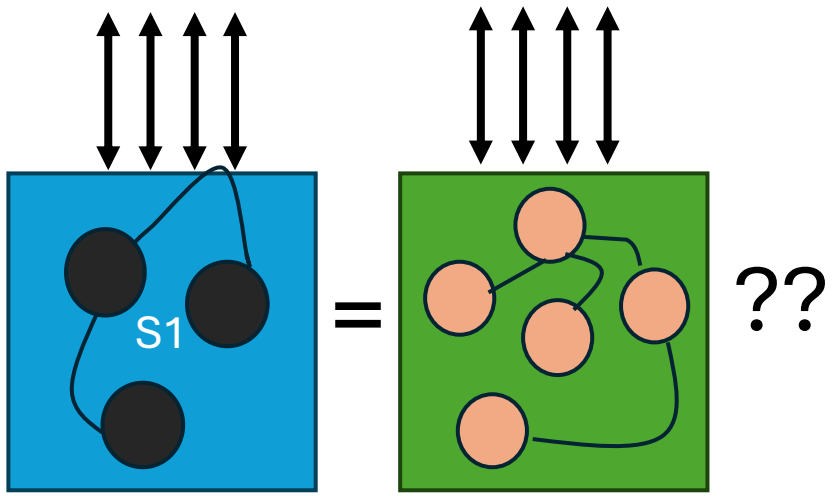


Bisimulation

Bisimulation Relation

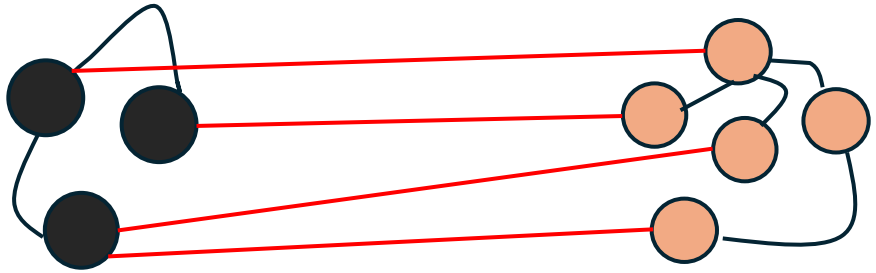


System Equivalence



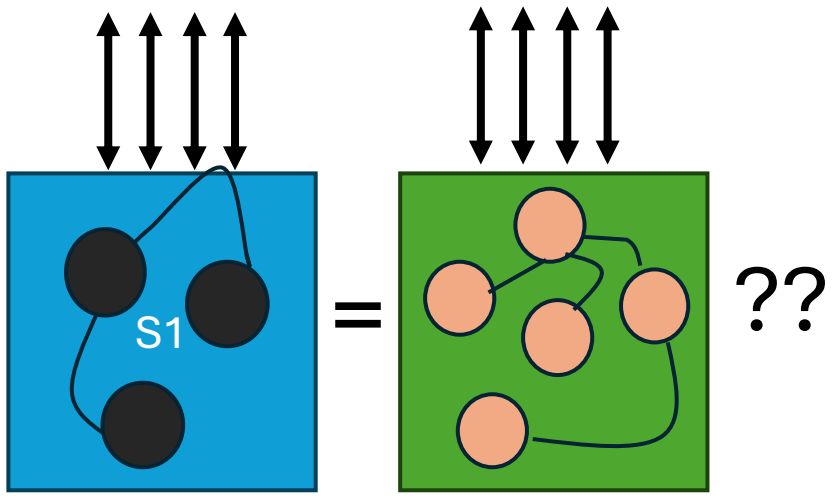
Bisimulation

Bisimulation Relation



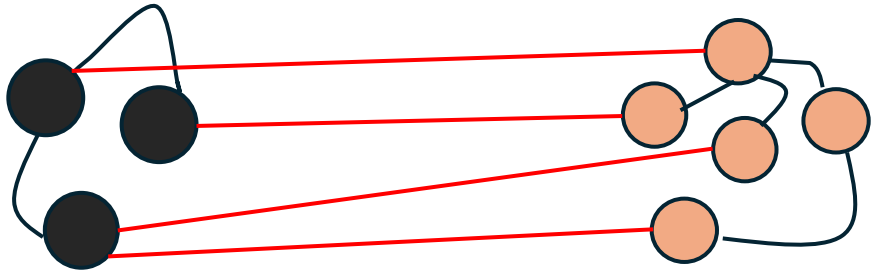
Starting states?

System Equivalence



Bisimulation

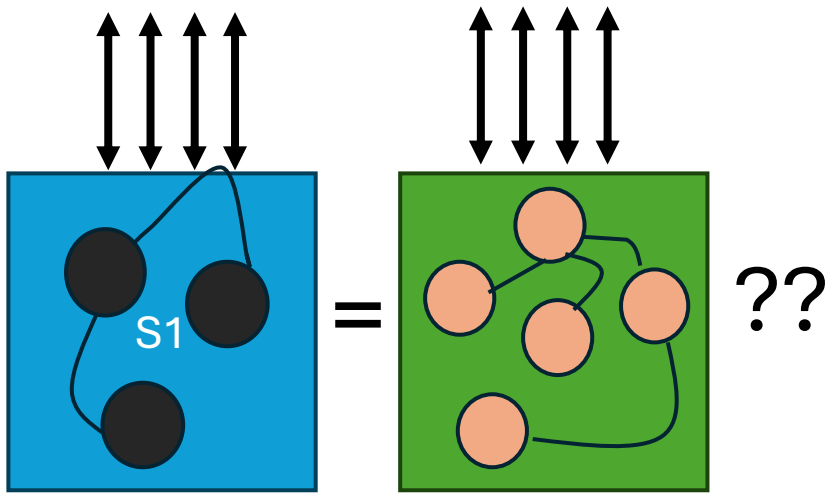
Bisimulation Relation



Starting states?

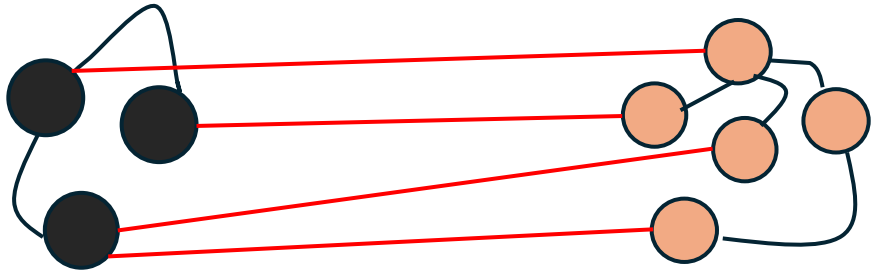
Next step?

System Equivalence



Bisimulation

Bisimulation Relation



Starting states?

Next step?

This paper - **Symbolic Execution**

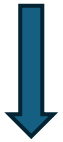
Case Studies and Evaluation

Case Studies and Evaluation

HSM	Spec		Driver	HW	SW	Proof
	core	total				
PIN backup	32	60	110	2670	190	470
PW hasher	5	150	90	3020	240	650
TOTP	10	180	80	2950	360	830

Lines of Code

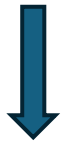
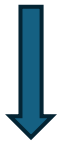
Case Studies and Evaluation



HSM	Spec		Driver	HW	SW	Proof
	core	total				
PIN backup	32	60	110	2670	190	470
PW hasher	5	150	90	3020	240	650
TOTP	10	180	80	2950	360	830

Lines of Code

Case Studies and Evaluation

HSM	Spec		Driver	HW	SW	Proof
	core	total				
PIN backup	32	60	110	2670	190	470
PW hasher	5	150	90	3020	240	650
TOTP	10	180	80	2950	360	830

Lines of Code

Case Studies and Evaluation



HSM	Spec		Driver	HW	SW	Proof
	core	total				
PIN backup	32	60	110	2670	190	470
PW hasher	5	150	90	3020	240	650
TOTP	10	180	80	2950	360	830

Lines of Code

Subtle bug

Subtle bug

Pin protected backup

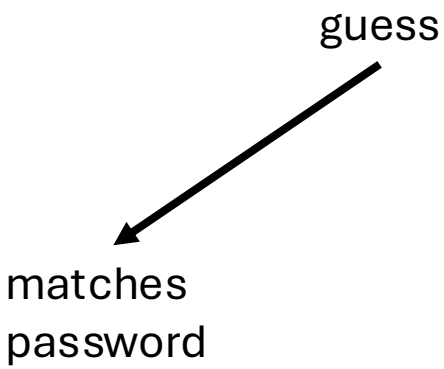
Subtle bug

guess

Pin protected backup

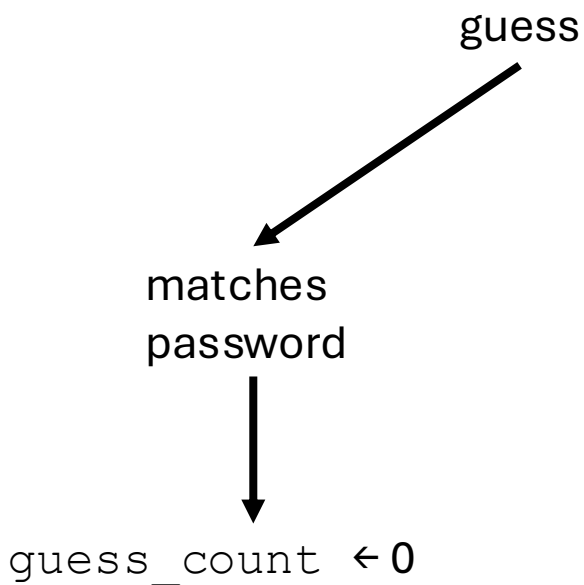
Subtle bug

Pin protected backup



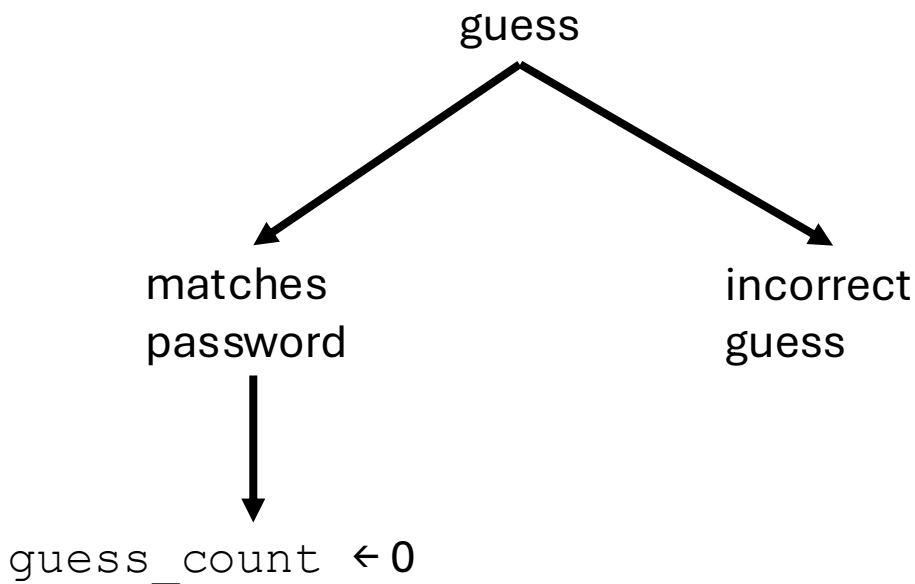
Subtle bug

Pin protected backup



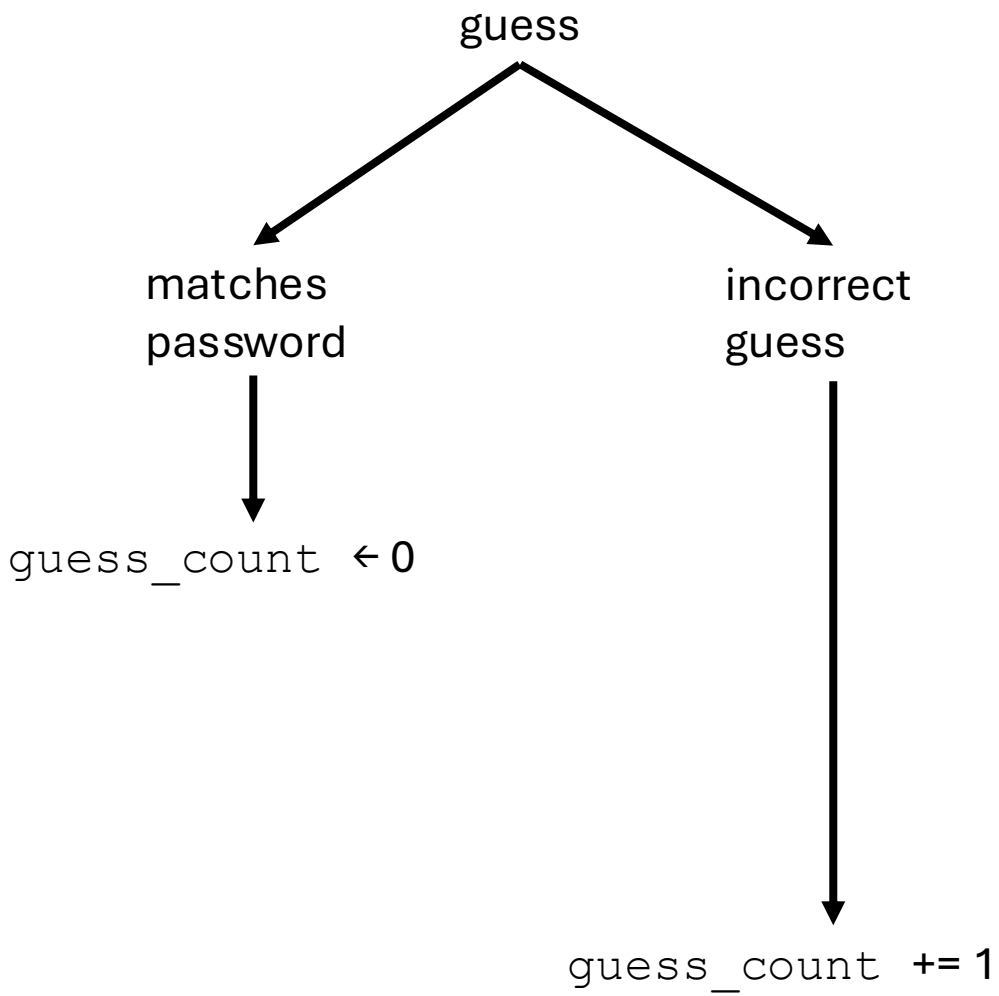
Subtle bug

Pin protected backup



Subtle bug

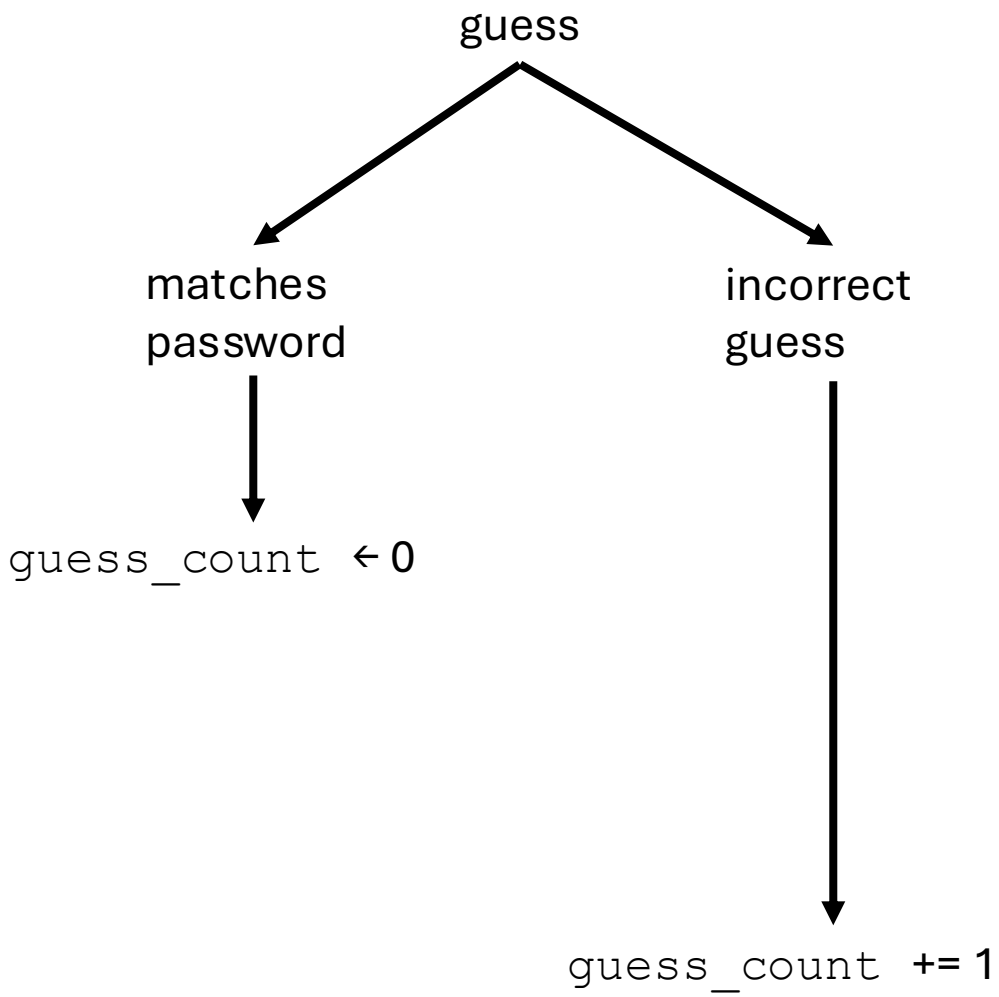
Pin protected backup



Subtle bug

Pin protected backup



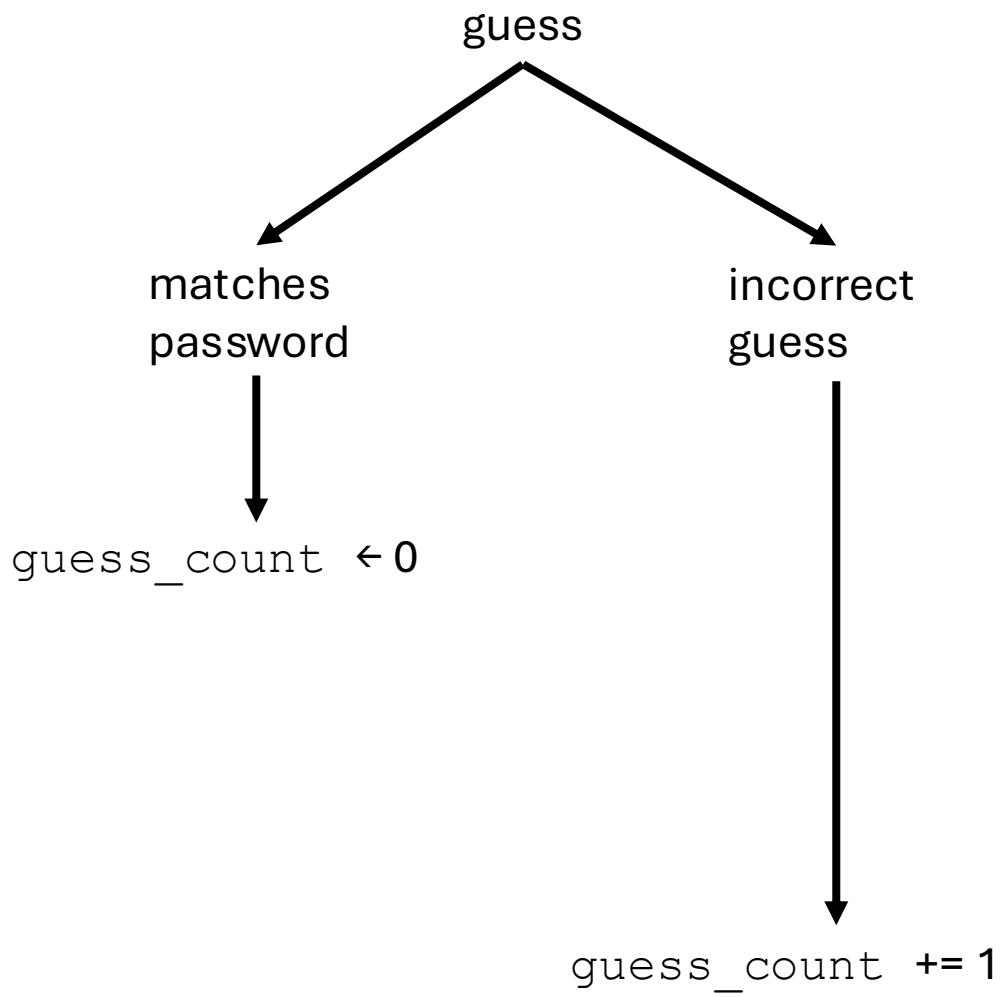


Subtle bug

Pin protected backup



guess 0

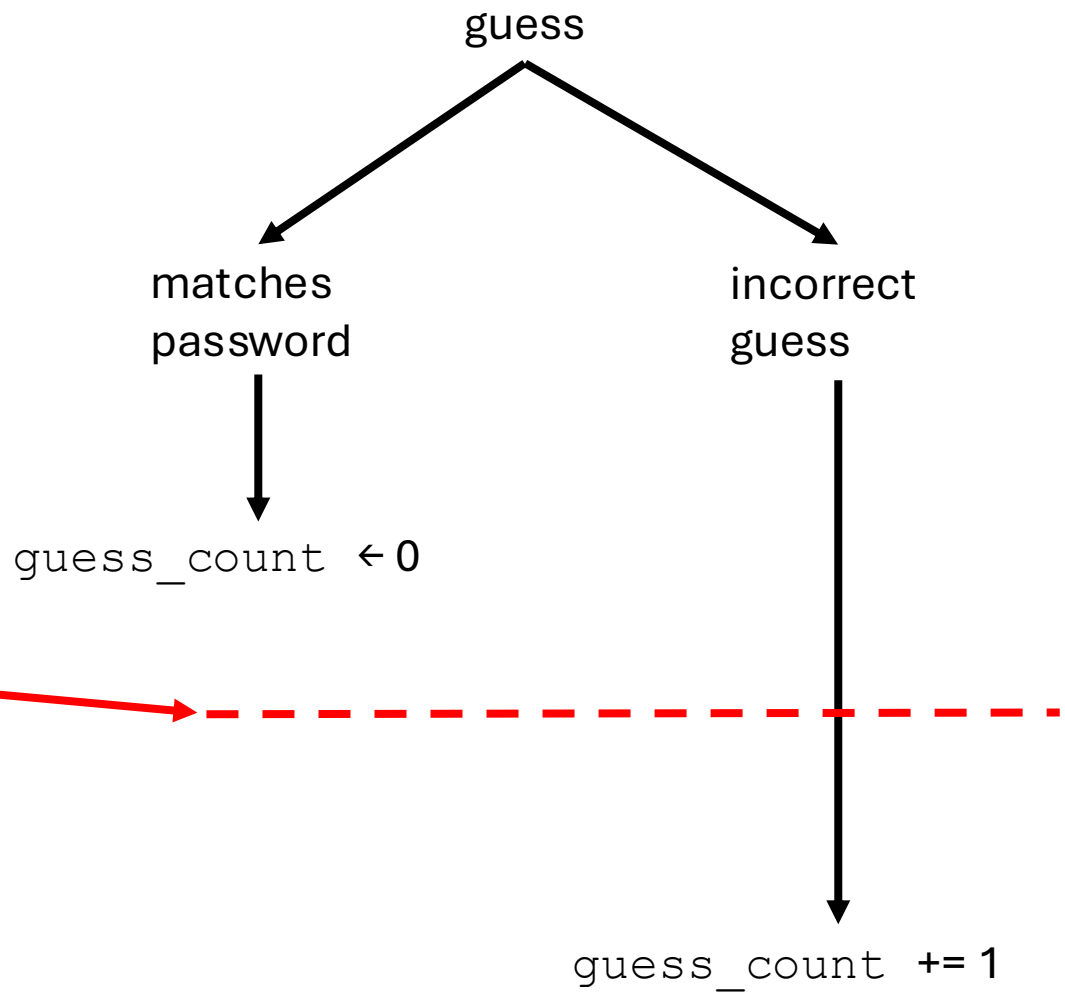


Subtle bug

Pin protected backup



guess 0
crash

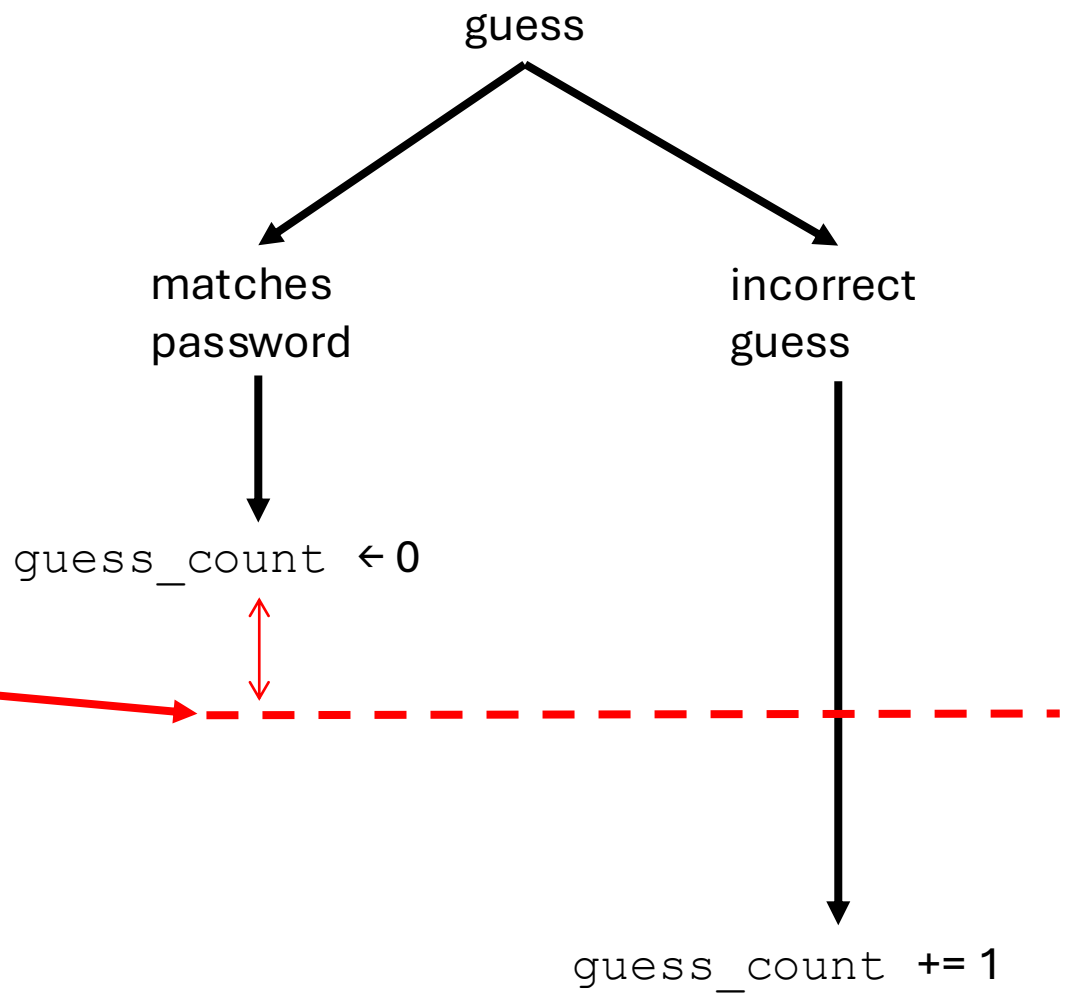


Subtle bug

Pin protected backup



guess 0
crash

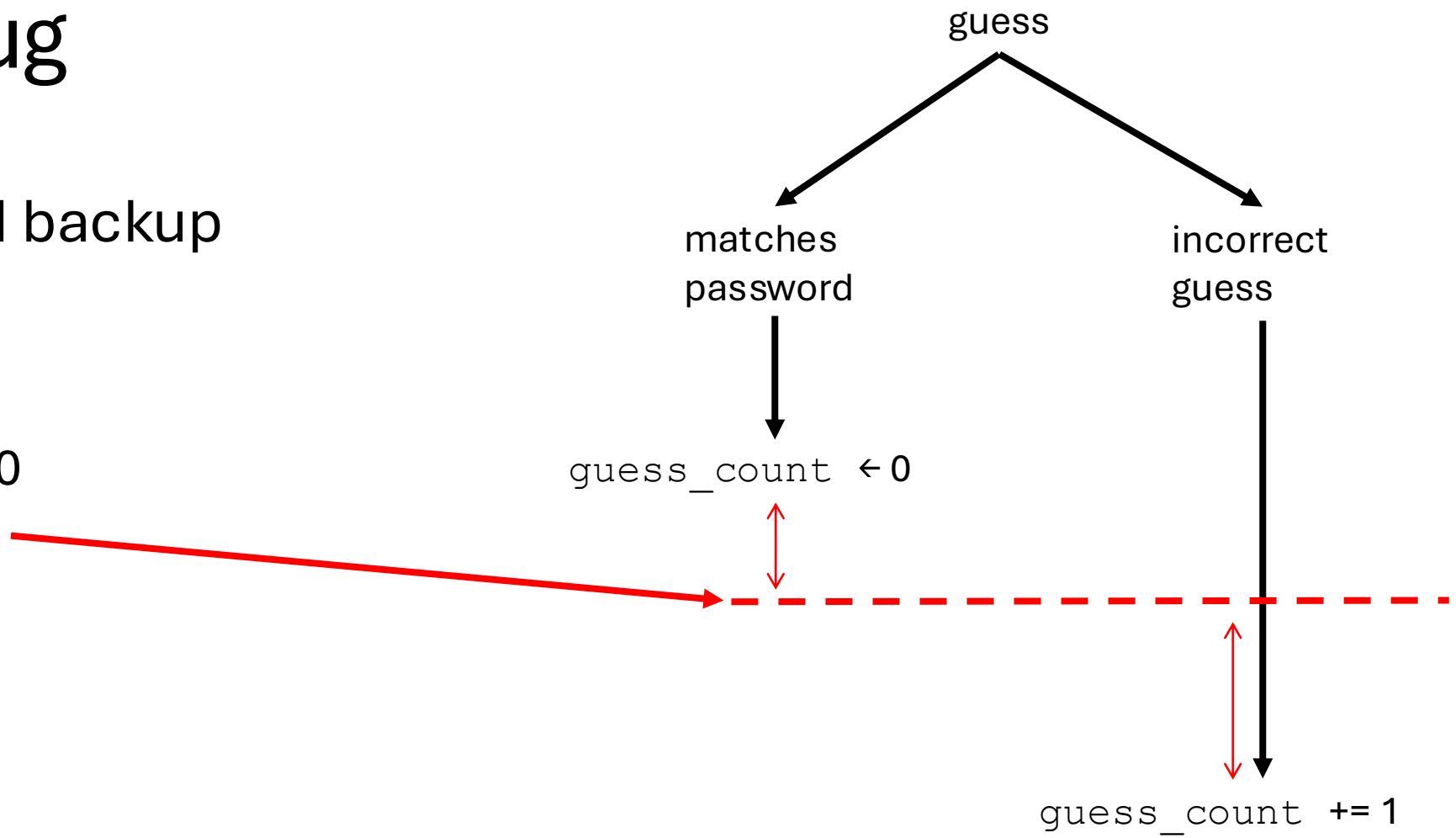


Subtle bug

Pin protected backup



guess 0
crash

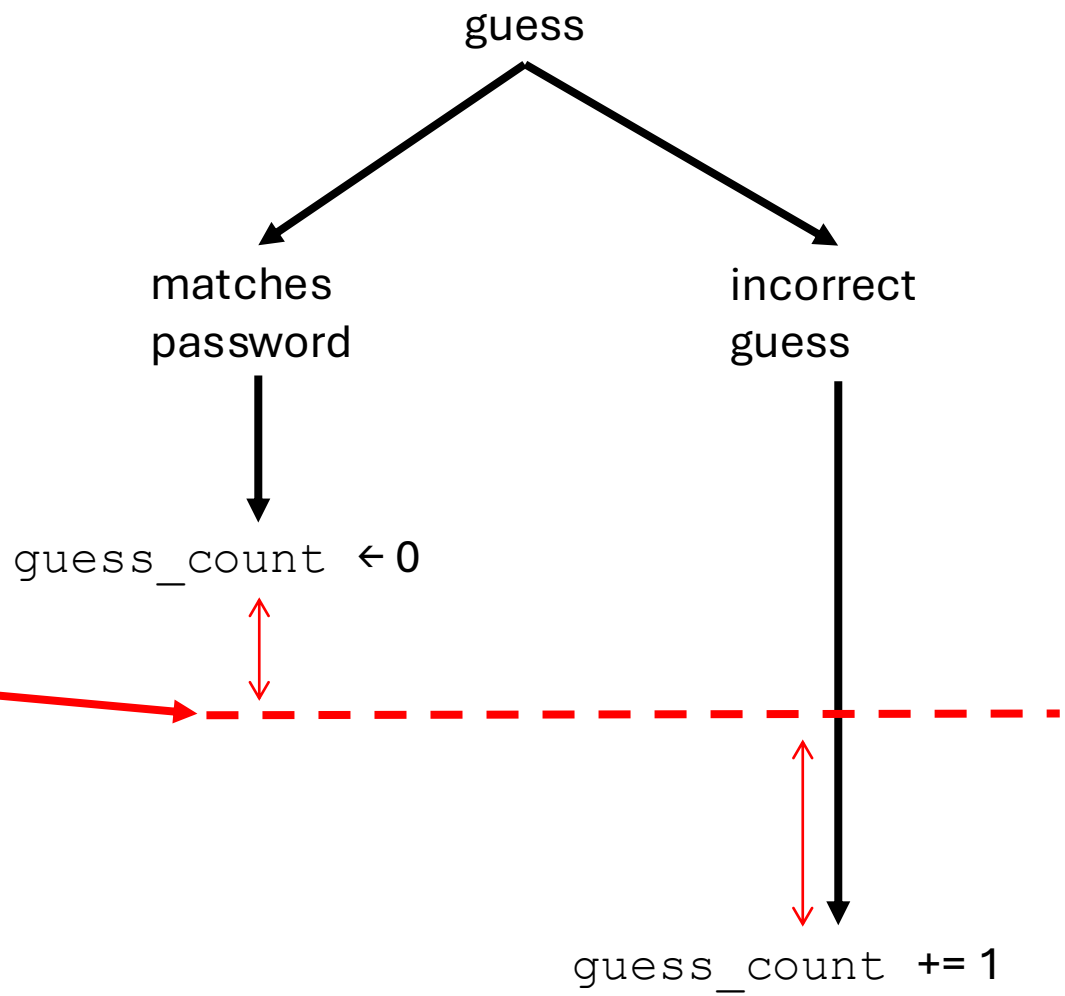


Subtle bug

Pin protected backup



guess 0
crash
guess 1

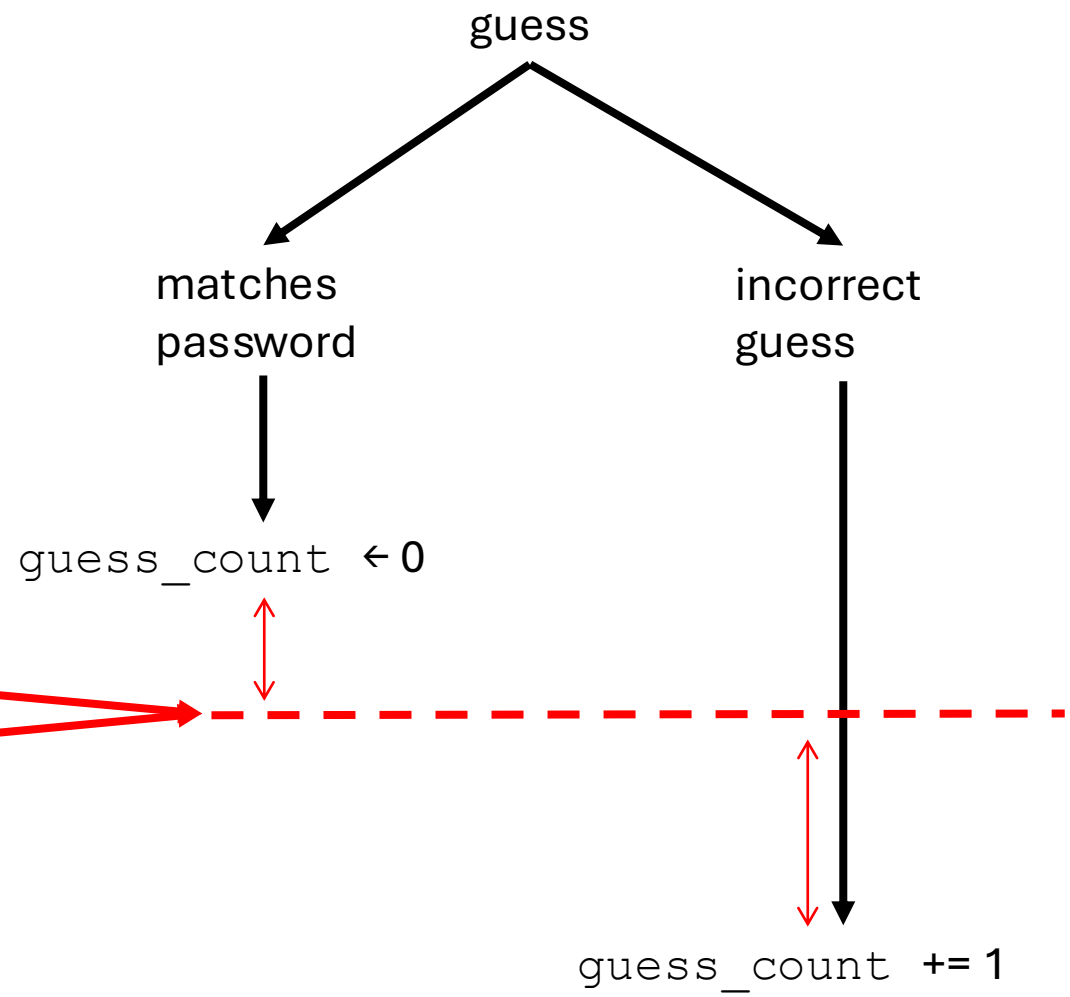


Subtle bug

Pin protected backup



guess 0
crash
guess 1
crash

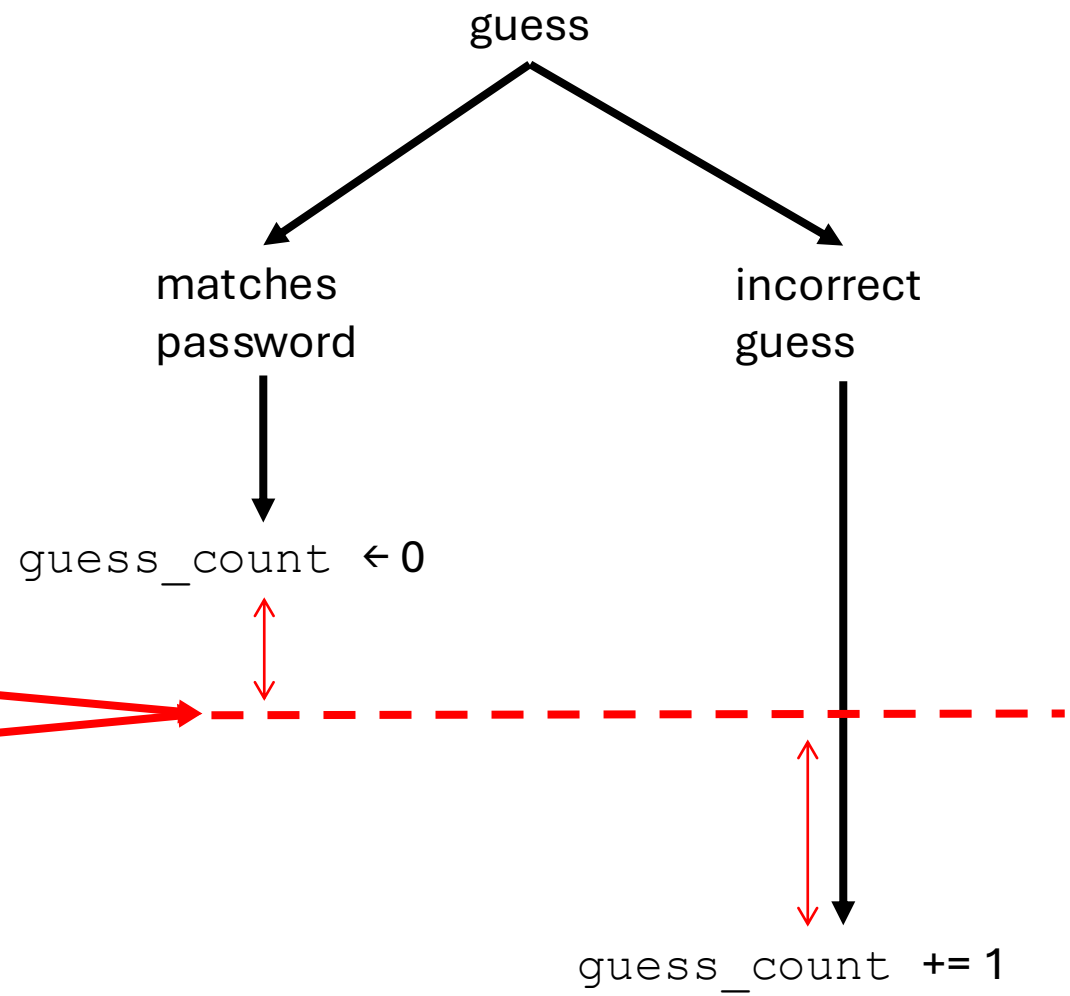


Subtle bug

Pin protected backup



guess 0
crash
guess 1
crash
...

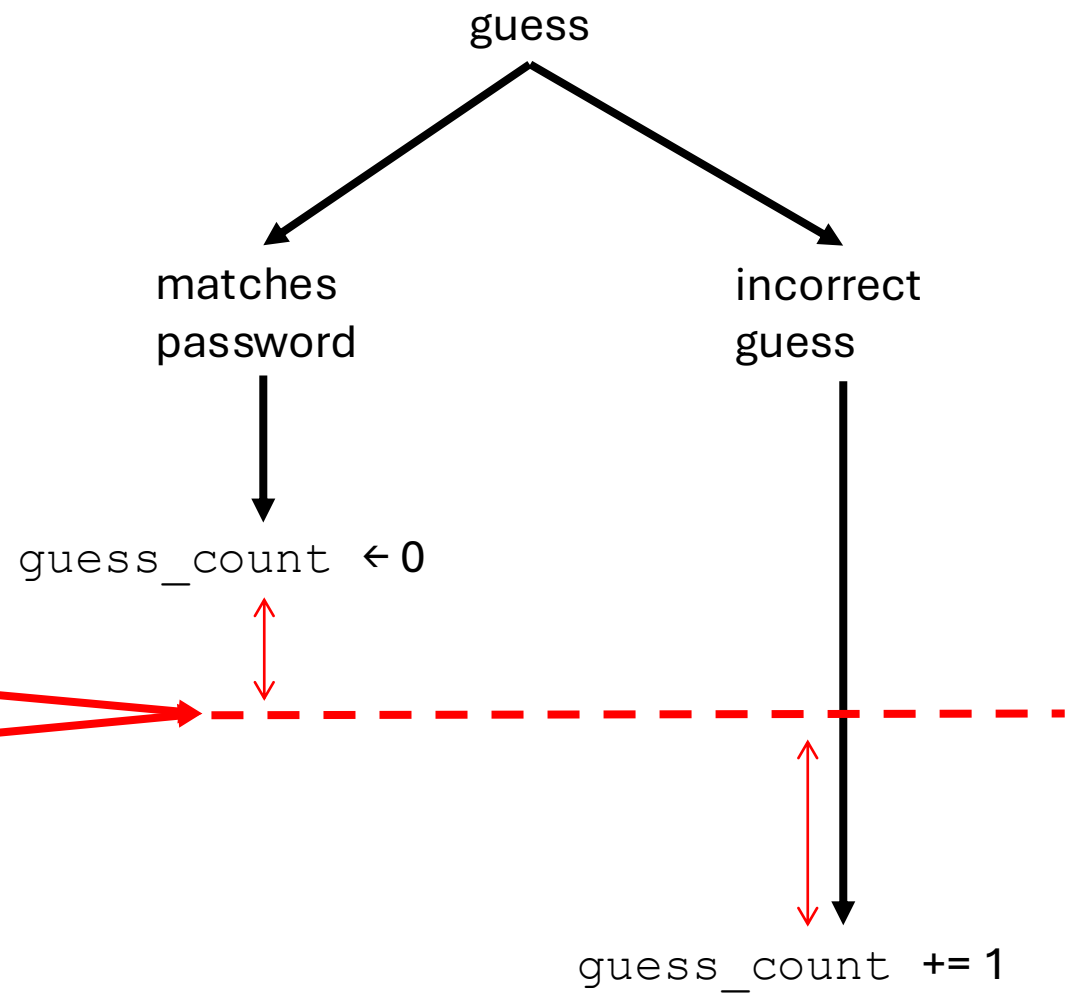


Subtle bug

Pin protected backup



guess 0
crash
guess 1
crash
...
...

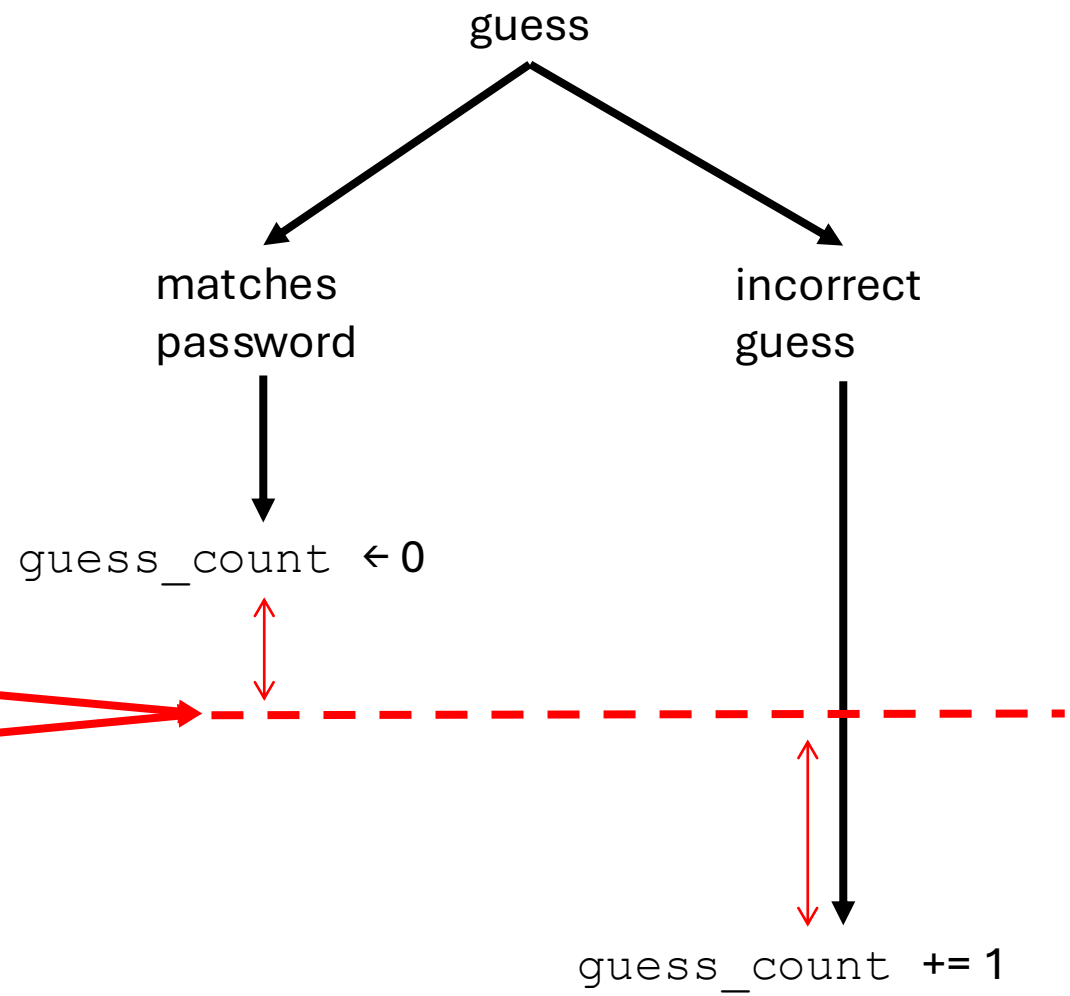


Subtle bug

Pin protected backup



guess 0
crash
guess 1
crash
...
...



Effectively:
`reset_guess_count()`

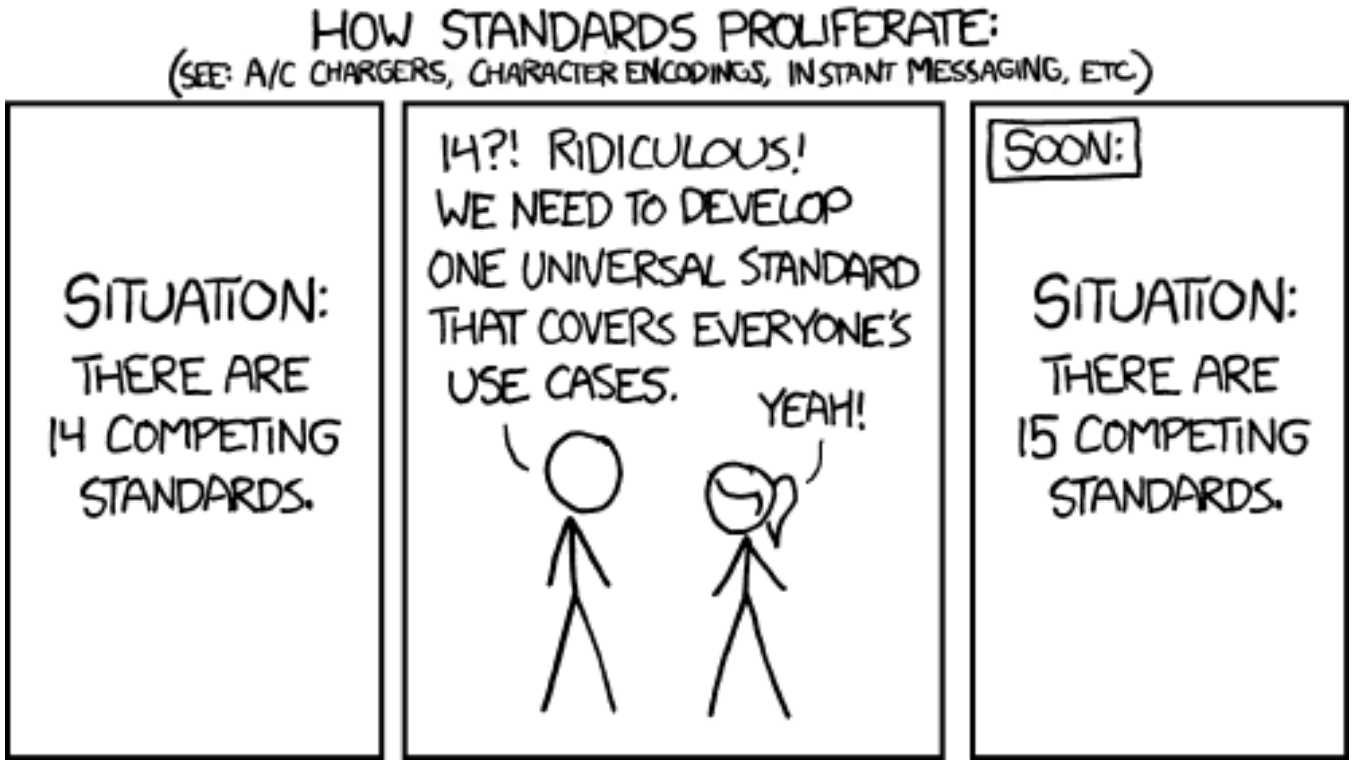
Limitations

Limitations

Inspired by simulation-based notions, **but isn’t cleanly a part of** any existing framework…

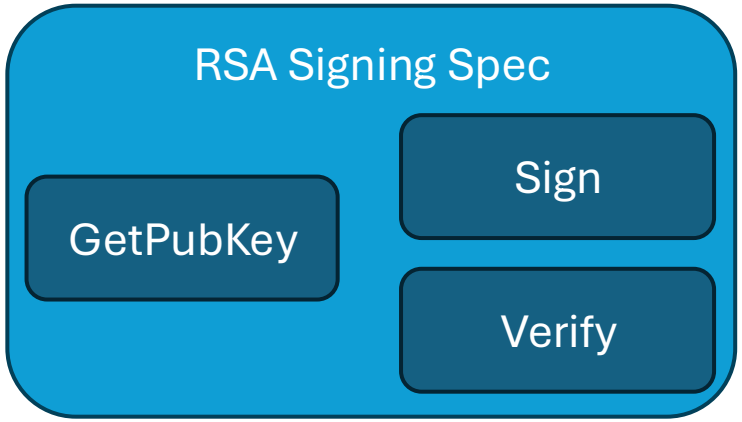
Limitations

Inspired by simulation-based notions, **but isn't cleanly a part of** any existing framework…

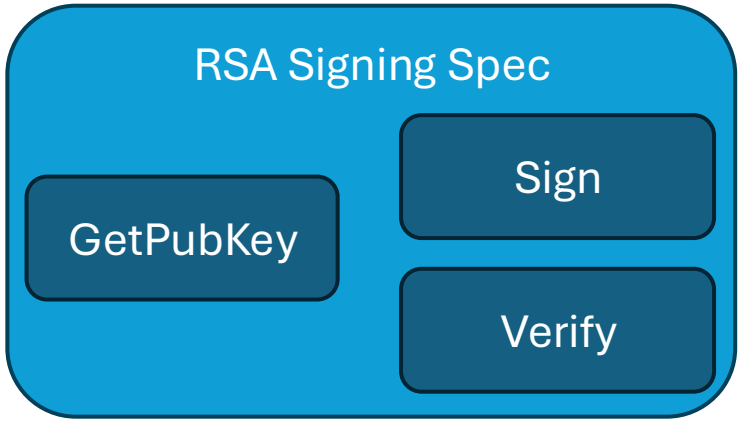


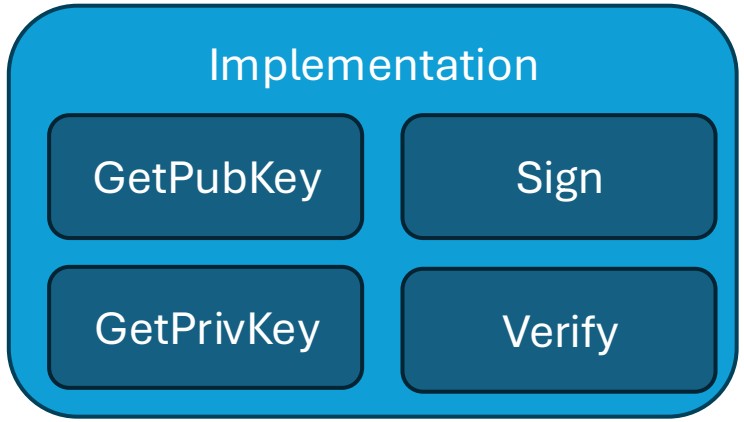
Concretely

Concretely

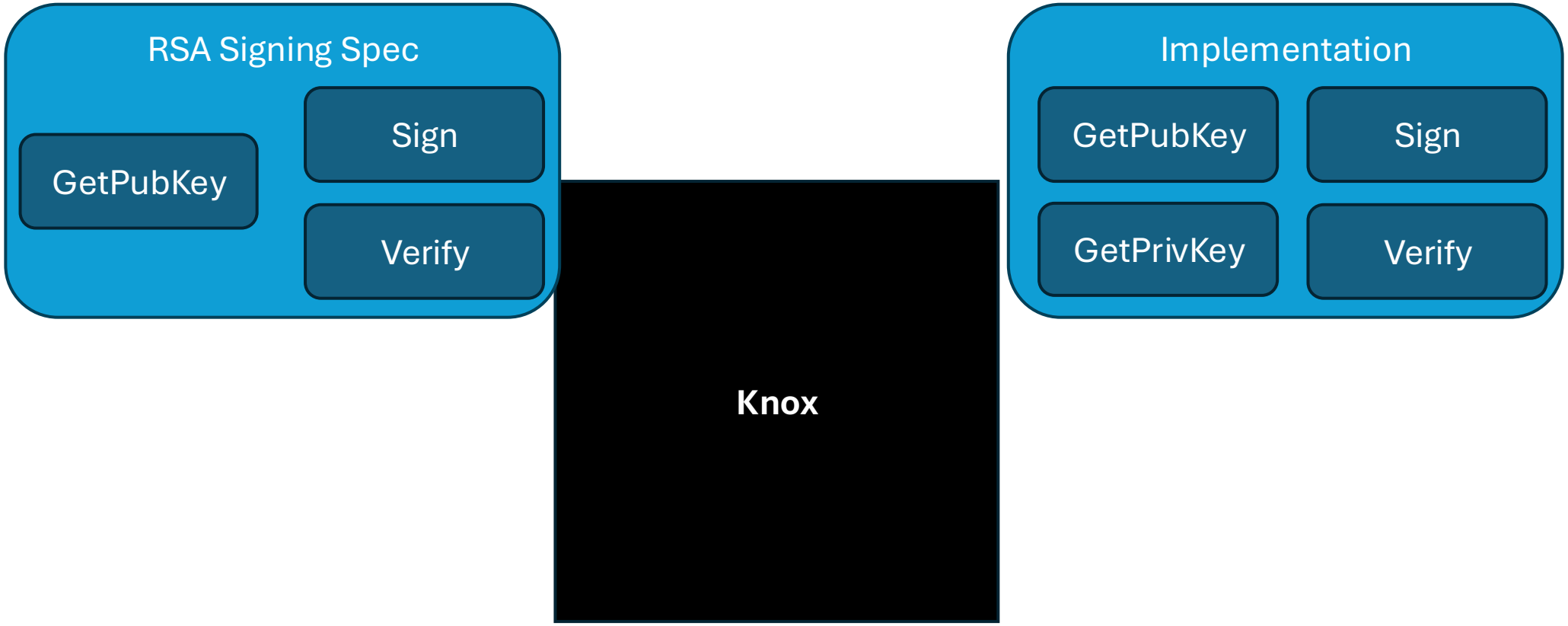


Concretely


RSA Signing Spec
GetPubKey
Sign
Verify


Implementation
GetPubKey
Sign
GetPrivKey
Verify

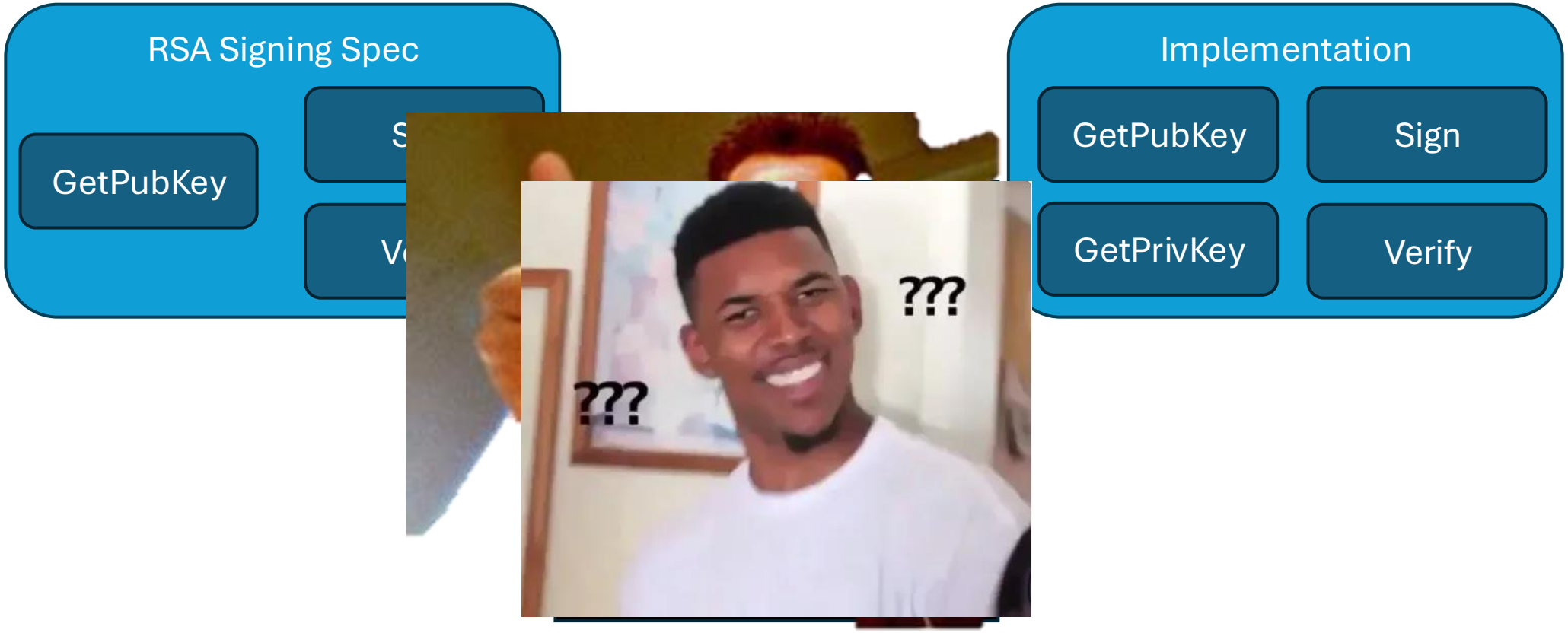
Concretely


RSA Signing Spec
GetPubKey
Sign
Verify
Knox
Implementation
GetPubKey
Sign
GetPrivKey
Verify

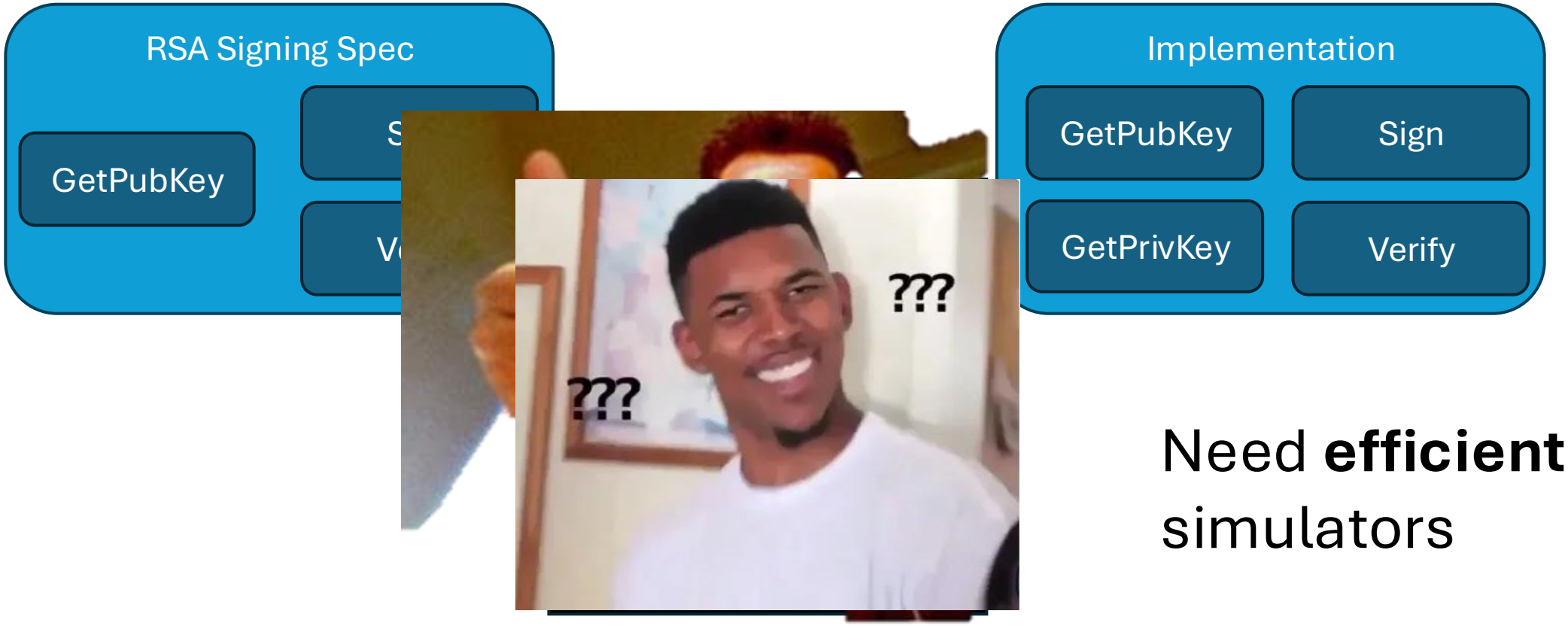
Concretely


RSA Signing Spec
GetPubKey
Implementation
GetPubKey
Sign
GetPrivKey
Verify
Knox

Concretely


RSA Signing Spec
GetPubKey
Implementation
GetPubKey
Sign
GetPrivKey
Verify
???
???

Concretely



Need **efficient** simulators

Summing it up…

Summing it up…

Goal: Trustworthy HSMs

Summing it up…

Goal: Trustworthy HSMs

- **Information Preserving Refinement** - Notion of trustworthy implementations

Summing it up…

Goal: Trustworthy HSMs

- **Information Preserving Refinement** - Notion of trustworthy implementations
- **Knox:** A system for proving IPR

Summing it up…

Goal: Trustworthy HSMs

- **Information Preserving Refinement** - Notion of trustworthy implementations
- **Knox:** A system for proving IPR
- **Case Studies:** Pin-protected backup, TOTP token, Password Hasher

Summing it up…

Goal: Trustworthy HSMs

- **Information Preserving Refinement** - Notion of trustworthy implementations
- **Knox:** A system for proving IPR
- **Case Studies:** Pin-protected backup, TOTP token, Password Hasher

Knox works in catching bugs across both hardware and software.

Verifying Hardware Security Modules with Information-Preserving Refinement

Anish Athalye, M. Frans Kaashoek, and Nickolai Zeldovich

OSDI ‘22