School of Computer Science and Engineering

Trustworthy Systems Group

Formalising device protocols for sDDF driver verification

Liam Murphy

liam.p.murphy@student.unsw.edu.au

Background

Why Drivers?

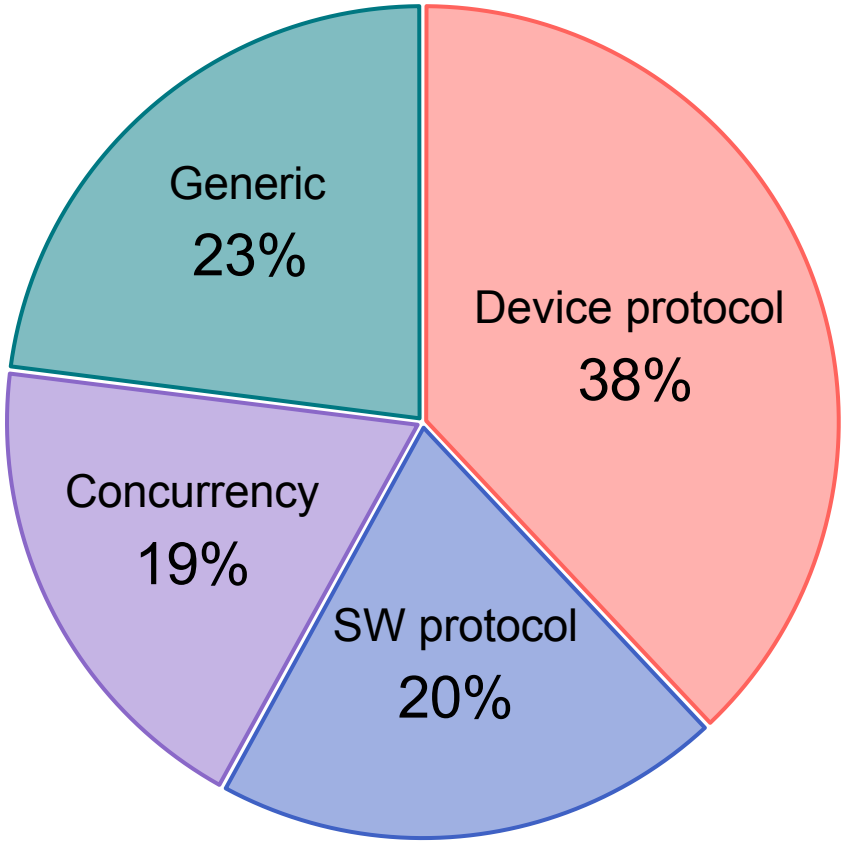
- Drivers are a frequent source of OS bugs
  - Majority of Linux CVEs from 2018–2022

Why Drivers?

- Drivers are a frequent source of OS bugs
  - Majority of Linux CVEs from 2018–2022
- Formal methods are the only reliable way to prevent this

Why Drivers?

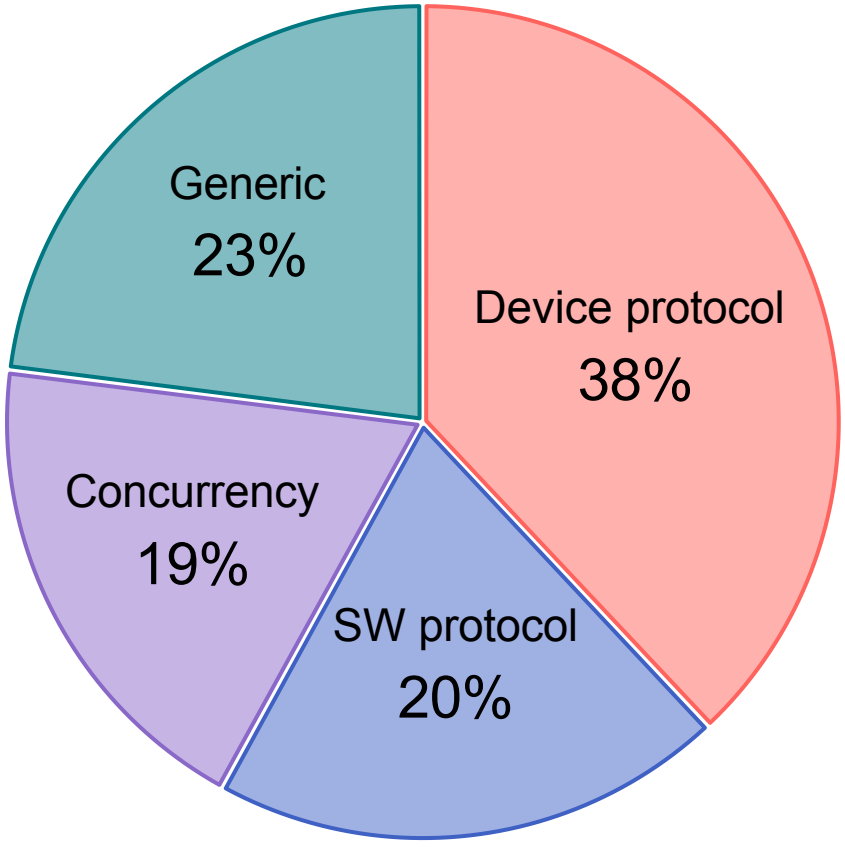
- Drivers are a frequent source of OS bugs
  - Majority of Linux CVEs from 2018–2022
- Formal methods are the only reliable way to prevent this
- Dominant cause is device-protocol violations



Causes of Linux driver bugs, 2002–2008 [Ryzhyk et al. 2009]

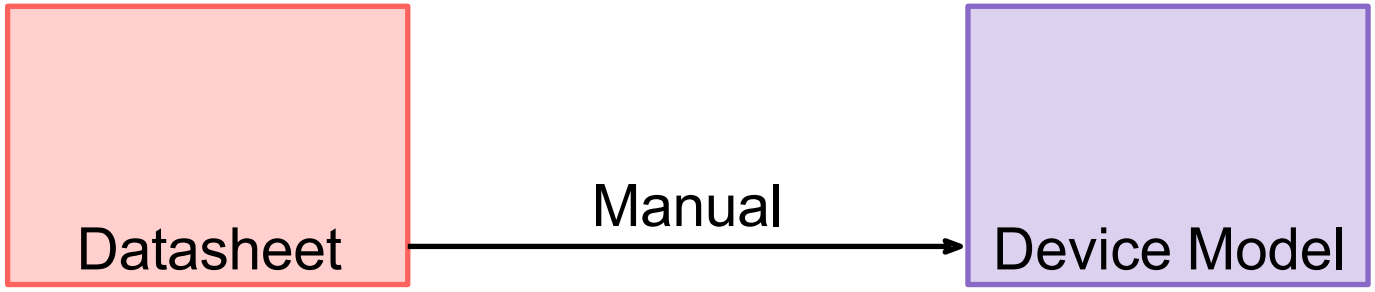
Why Drivers?

- Drivers are a frequent source of OS bugs
  - Majority of Linux CVEs from 2018–2022
- Formal methods are the only reliable way to prevent this
- Dominant cause is device-protocol violations
  - Without a formal model of the protocol, formal methods cannot prevent this

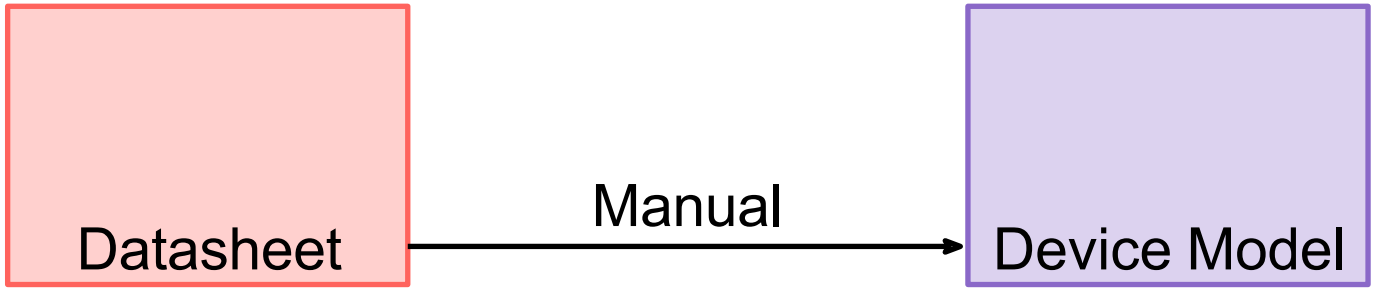


Causes of Linux driver bugs, 2002–2008 [Ryzhyk et al. 2009]

Datasheets

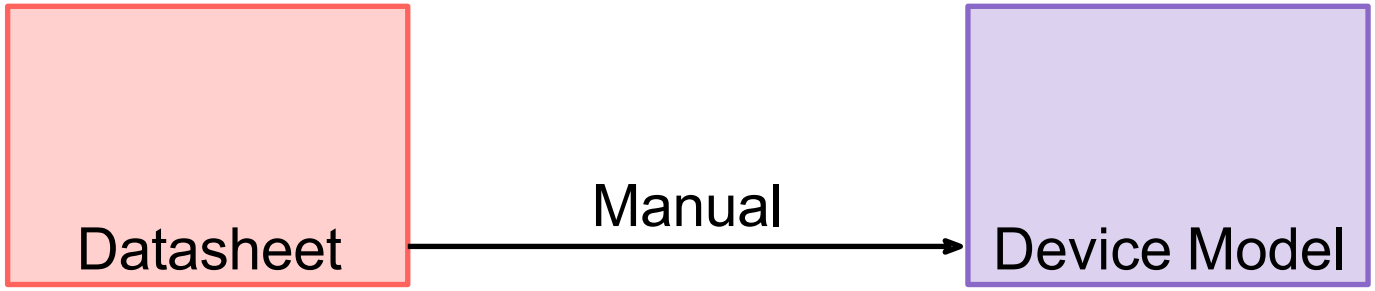


Datasheets



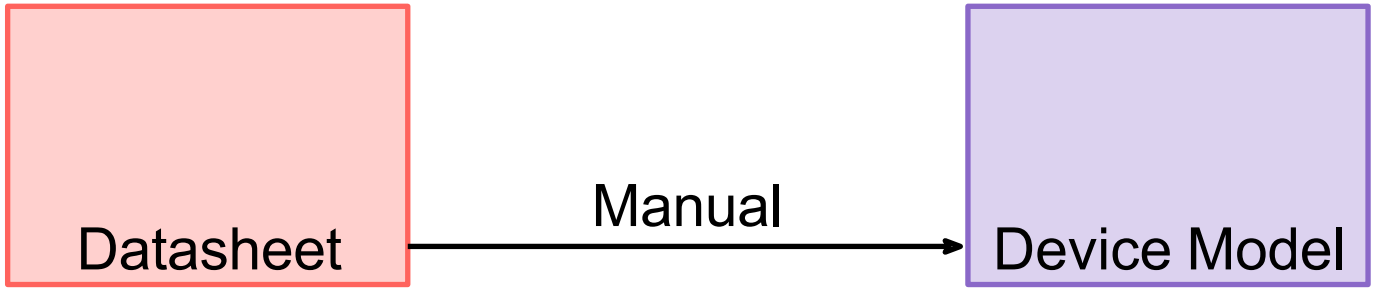
- Imprecise

Datasheets



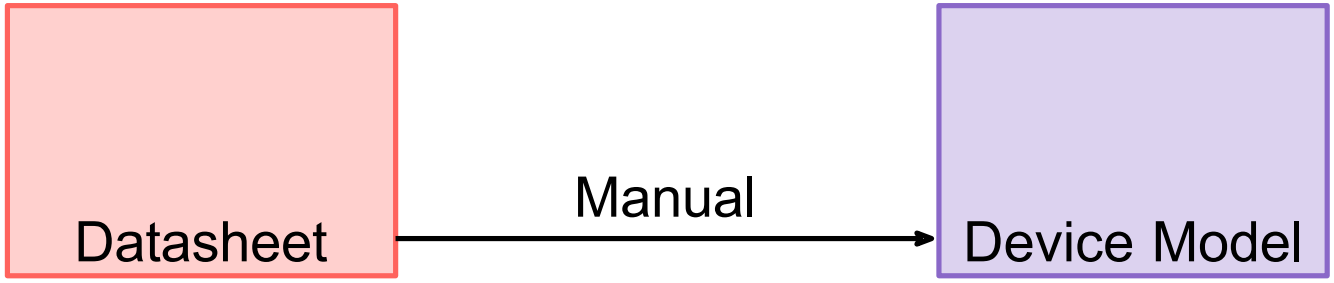
- Imprecise
- Incomplete

Datasheets



- Imprecise
- Incomplete
- Incorrect?

Datasheets



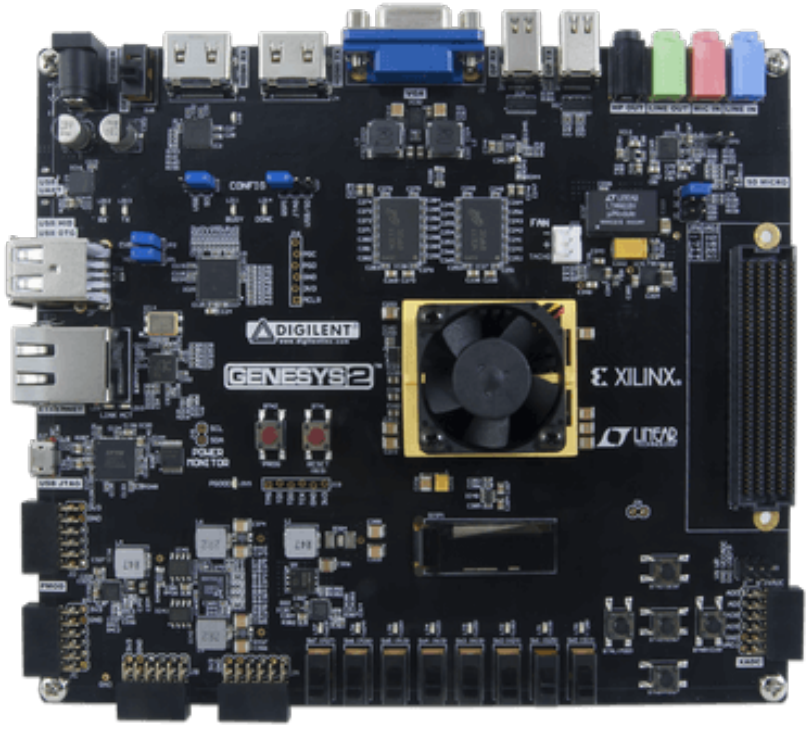
- Imprecise
- Incomplete
- Incorrect?

⟹ Unreliable!

Datasheets

Serengeti

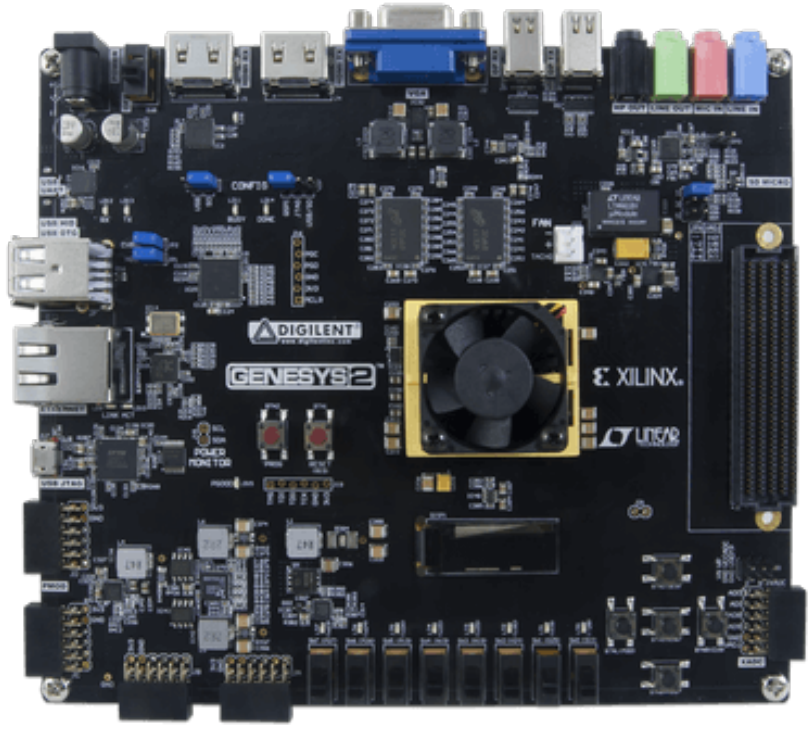
- Fork of Cheshire, an open-source RISC-V SoC from the PULP project
- Runs on common Digilent FPGA dev kits
- Several added features:



Digilent Genesys2, used to run Serengeti

Serengeti

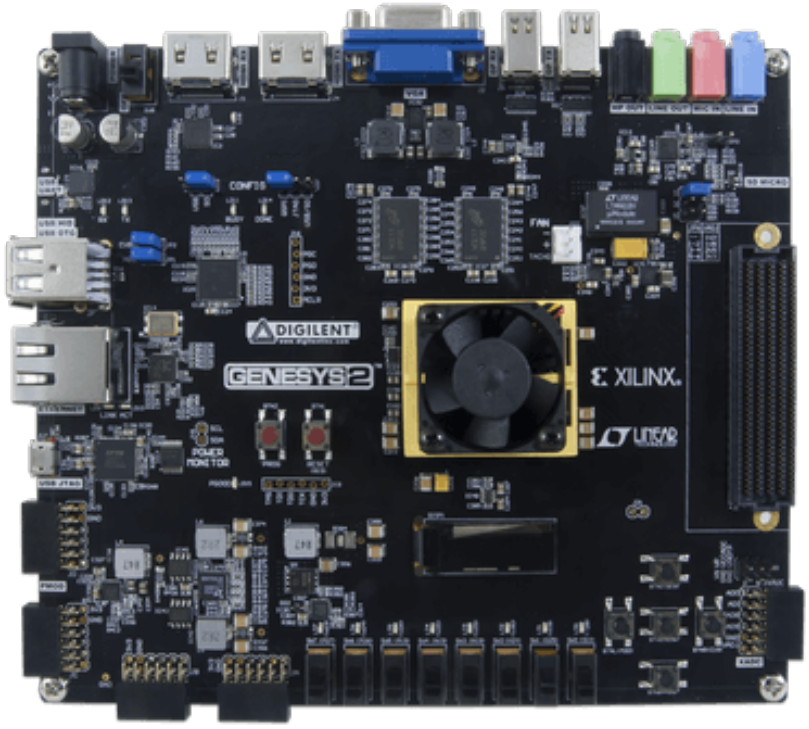
- Fork of Cheshire, an open-source RISC-V SoC from the PULP project
- Runs on common Digilent FPGA dev kits
- Several added features:
  - Time protection (fence.t, LLC partitioning)



Digilent Genesys2, used to run Serengeti

Serengeti

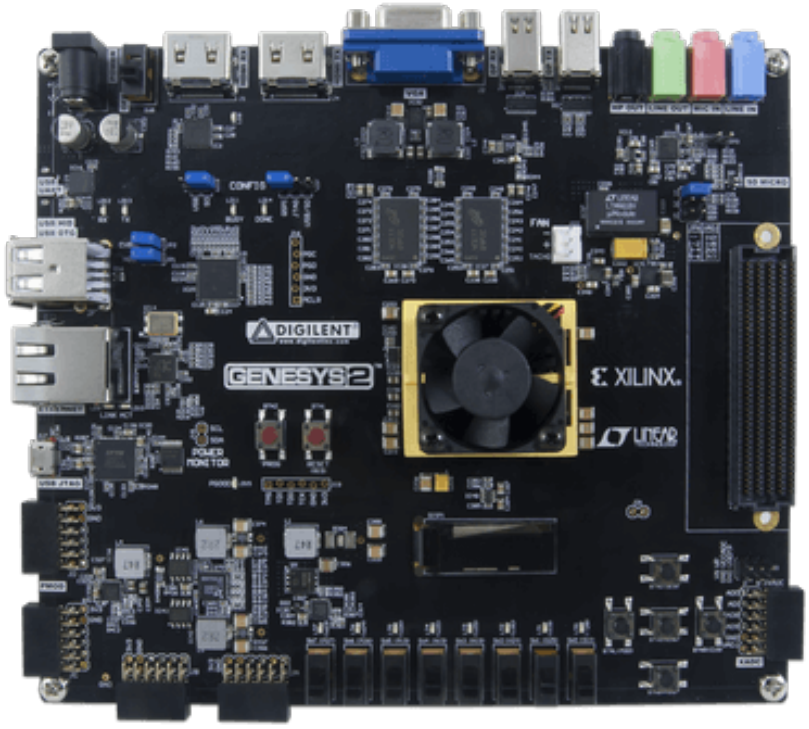
- Fork of Cheshire, an open-source RISC-V SoC from the PULP project
- Runs on common Digilent FPGA dev kits
- Several added features:
  - Time protection (fence.t, LLC partitioning)
  - High-speed Ethernet MAC (WIP)



Digilent Genesys2, used to run Serengeti

Serengeti

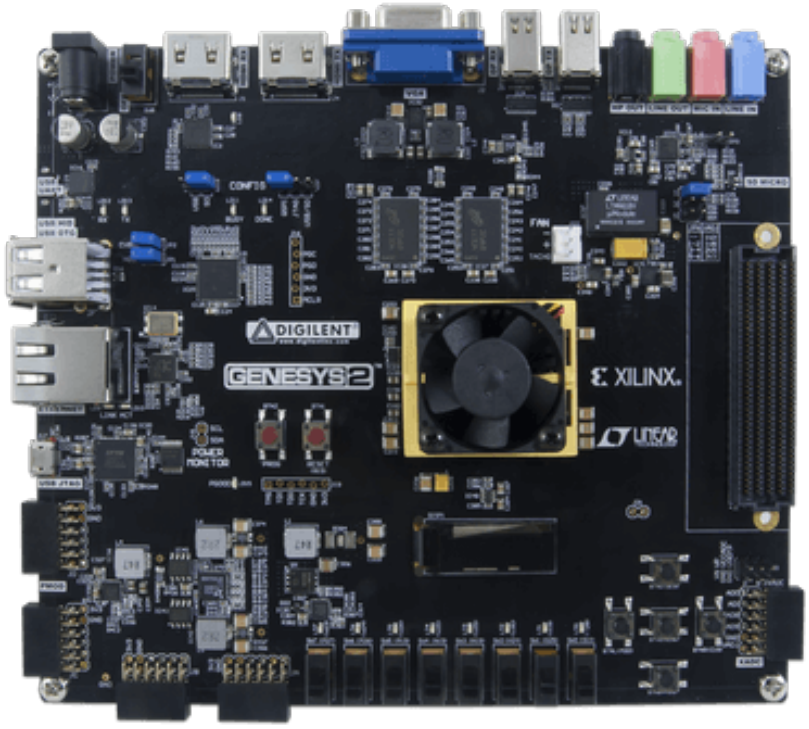
- Fork of Cheshire, an open-source RISC-V SoC from the PULP project
- Runs on common Digilent FPGA dev kits
- Several added features:
  - Time protection (fence.t, LLC partitioning)
  - High-speed Ethernet MAC (WIP)
  - Open-source root of trust + secure boot (WIP)



Digilent Genesys2, used to run Serengeti

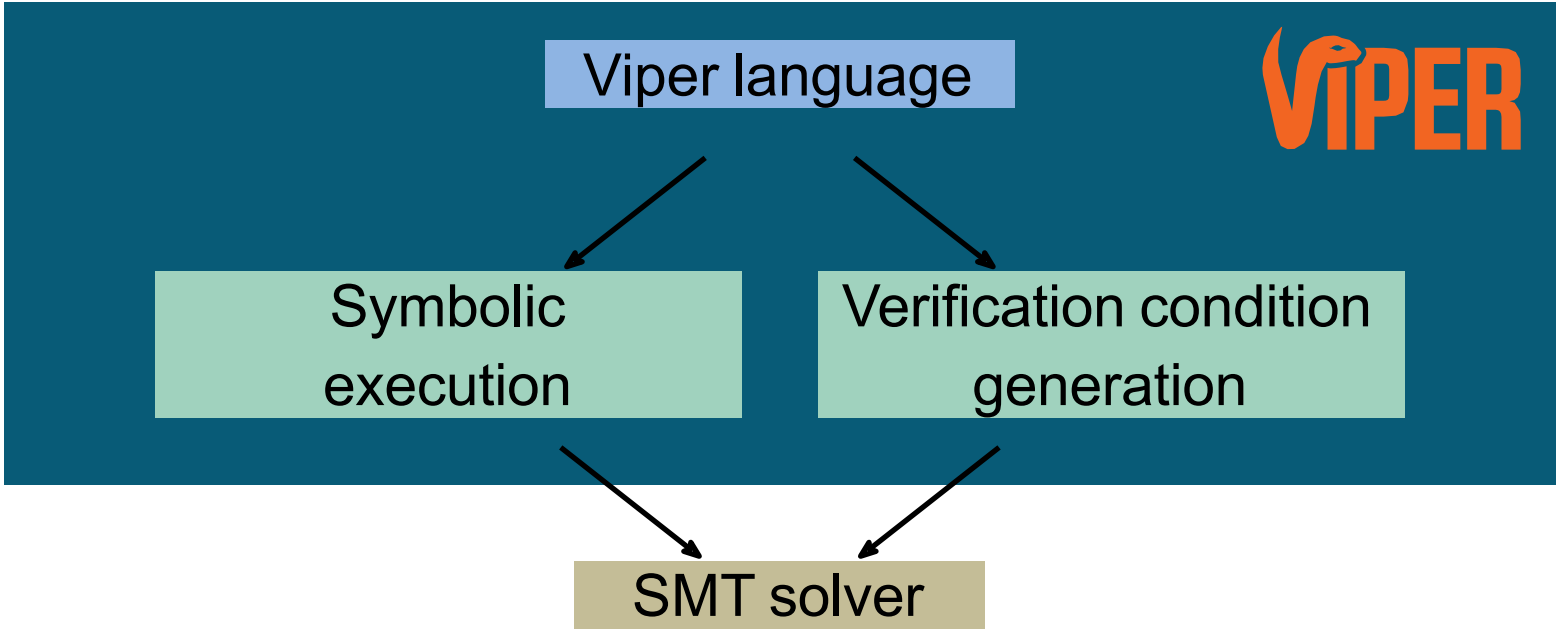
Serengeti

- Fork of Cheshire, an open-source RISC-V SoC from the PULP project
- Runs on common Digilent FPGA dev kits
- Several added features:
  - Time protection (fence.t, LLC partitioning)
  - High-speed Ethernet MAC (WIP)
  - Open-source root of trust + secure boot (WIP)
- Initial target: I^2C controller

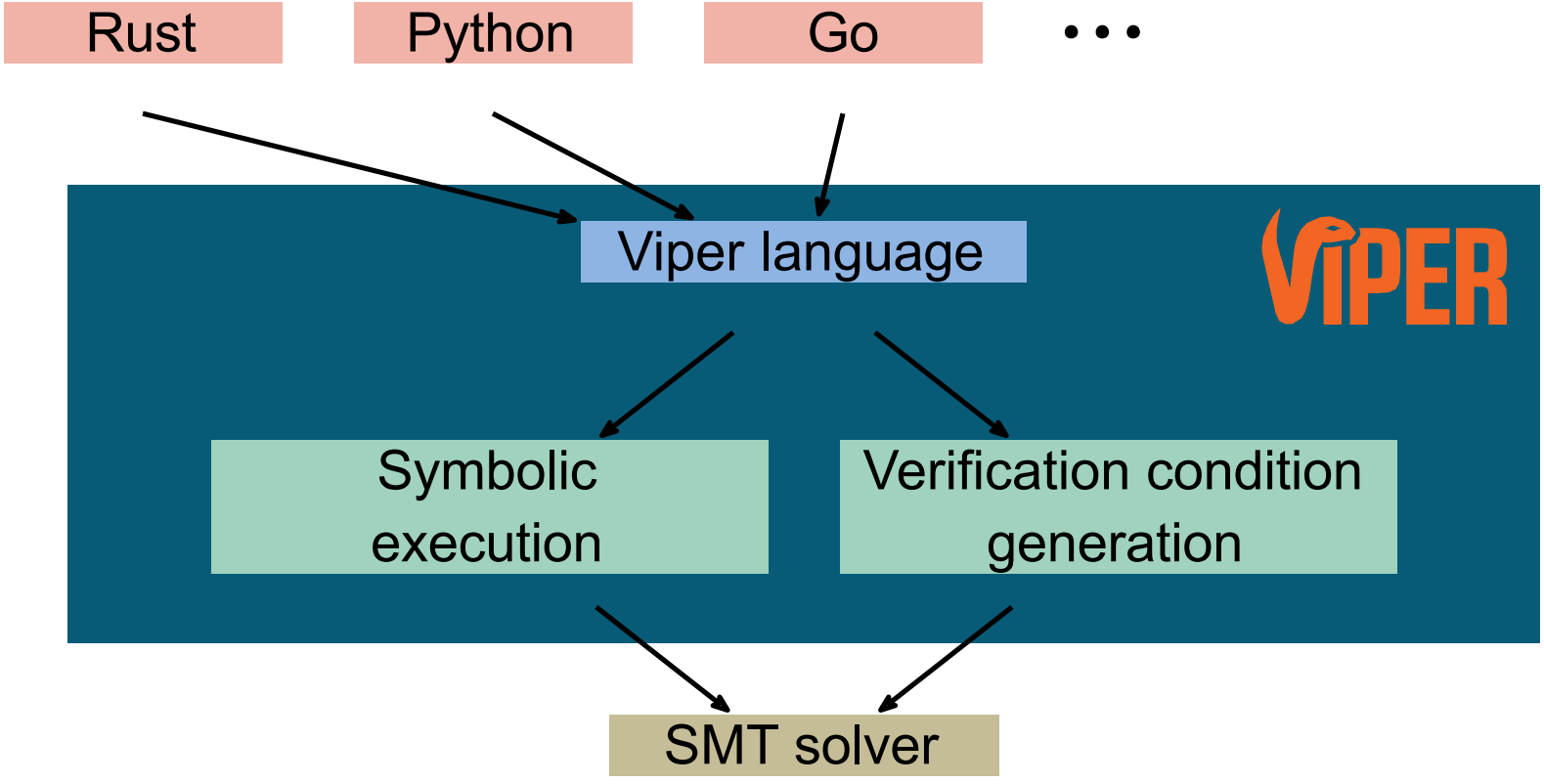


Digilent Genesys2, used to run Serengeti

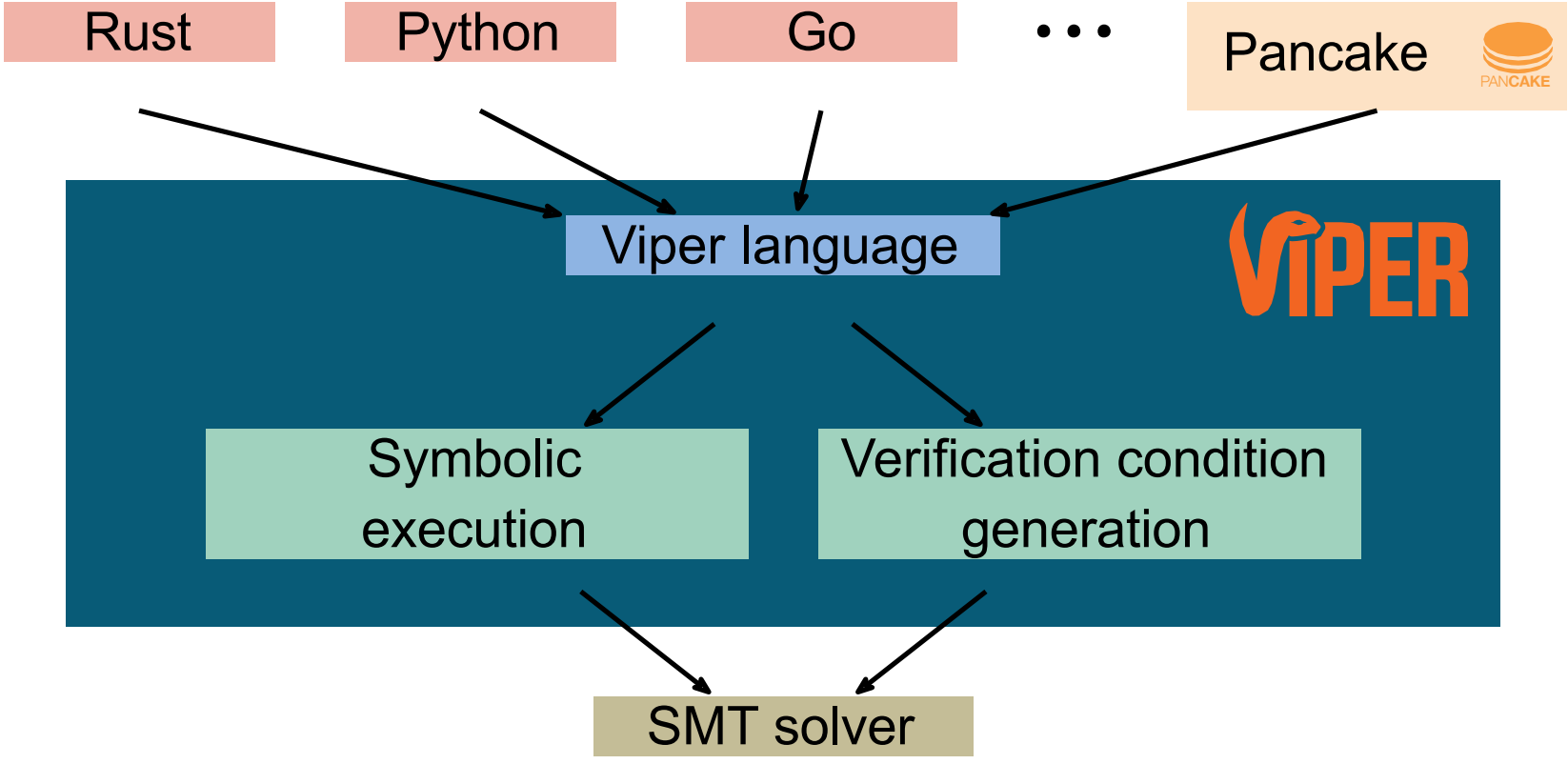
Pancake-to-Viper



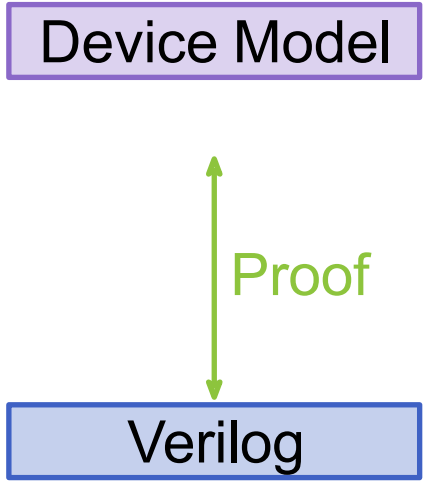
Pancake-to-Viper



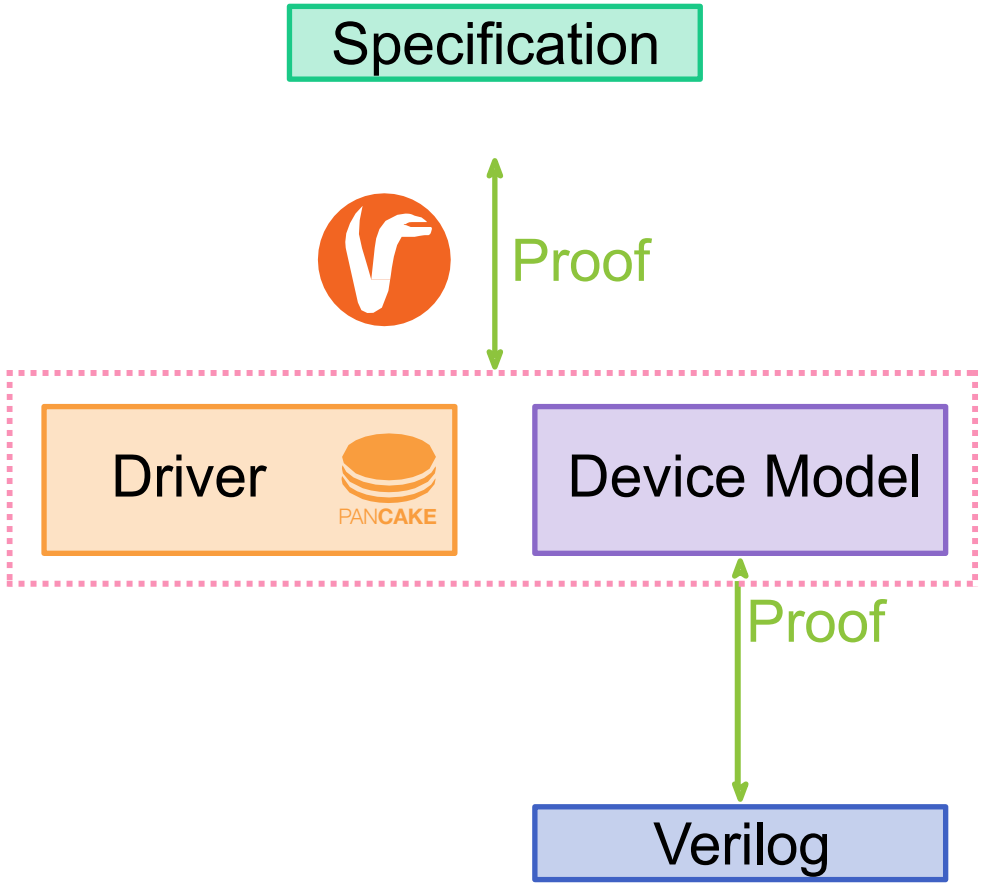
Pancake-to-Viper



Verification Flow



Verification Flow



Overview

- Hardware Verification
- Specification Format
- Driver Verification
- Status
- Future Work

Hardware Verification

Tools

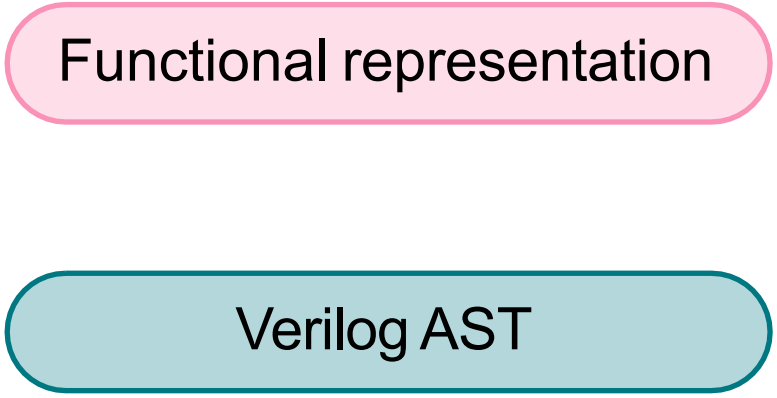
- HOL4 theorem prover
- HOL4 Verilog formalisation (Lööw):

Tools

- HOL4 theorem prover
- HOL4 Verilog formalisation (Lööw):
  - Semantics for Verilog AST

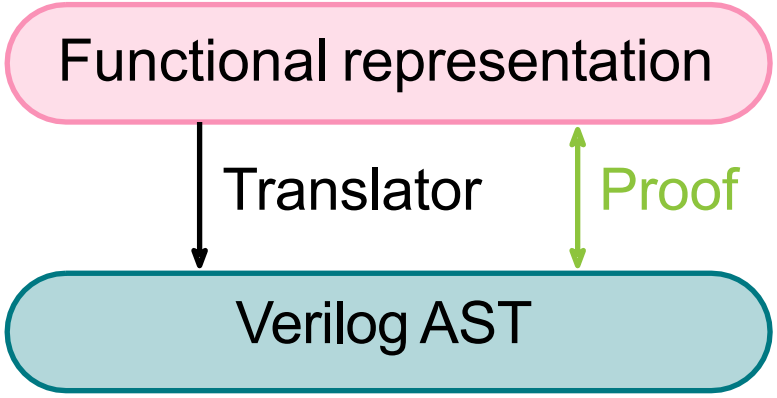
Tools

- HOL4 theorem prover
- HOL4 Verilog formalisation (Lööw):
  - Semantics for Verilog AST
  - Semantics for functional representation



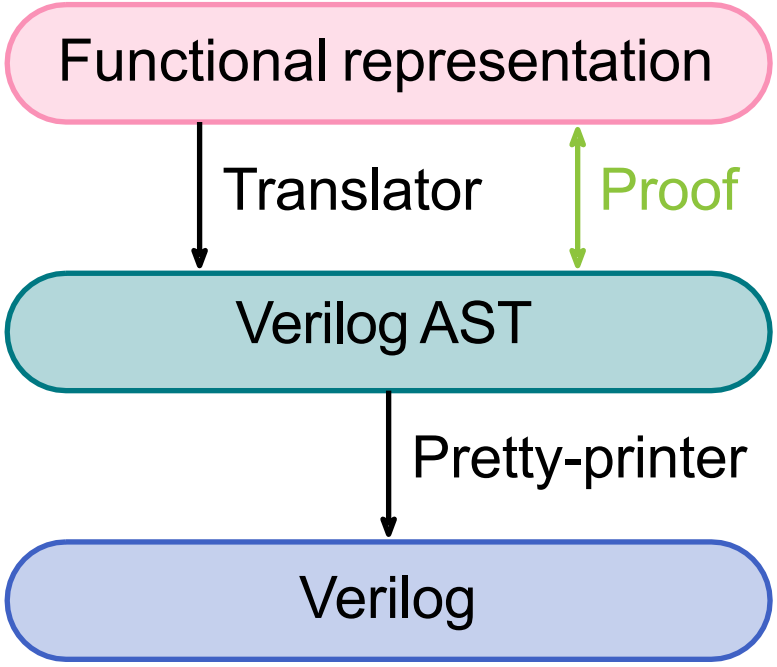
Tools

- HOL4 theorem prover
- HOL4 Verilog formalisation (Lööw):
  - Semantics for Verilog AST
  - Semantics for functional representation
  - Proof-producing translator from Verilog to AST



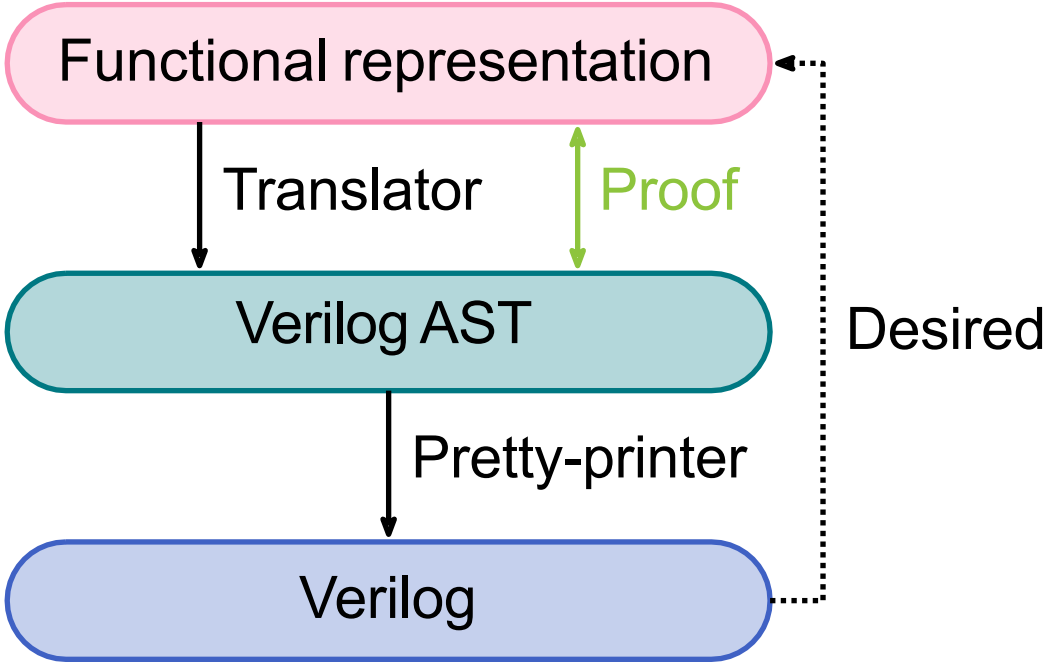
Tools

- HOL4 theorem prover
- HOL4 Verilog formalisation (Lööw):
  - Semantics for Verilog AST
  - Semantics for functional representation
  - Proof-producing translator from Verilog to AST
  - Pretty-printer for deep embedding



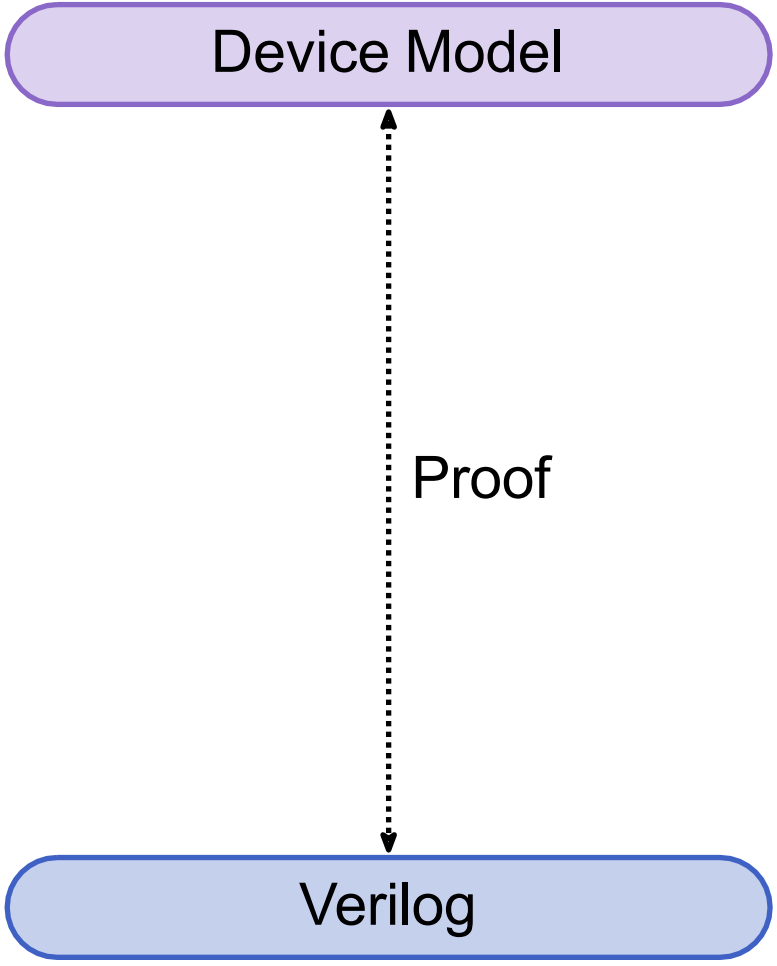
Tools

- HOL4 theorem prover
- HOL4 Verilog formalisation (Lööw):
  - Semantics for Verilog AST
  - Semantics for functional representation
  - Proof-producing translator from Verilog to AST
  - Pretty-printer for deep embedding
- Problem: need opposite direction



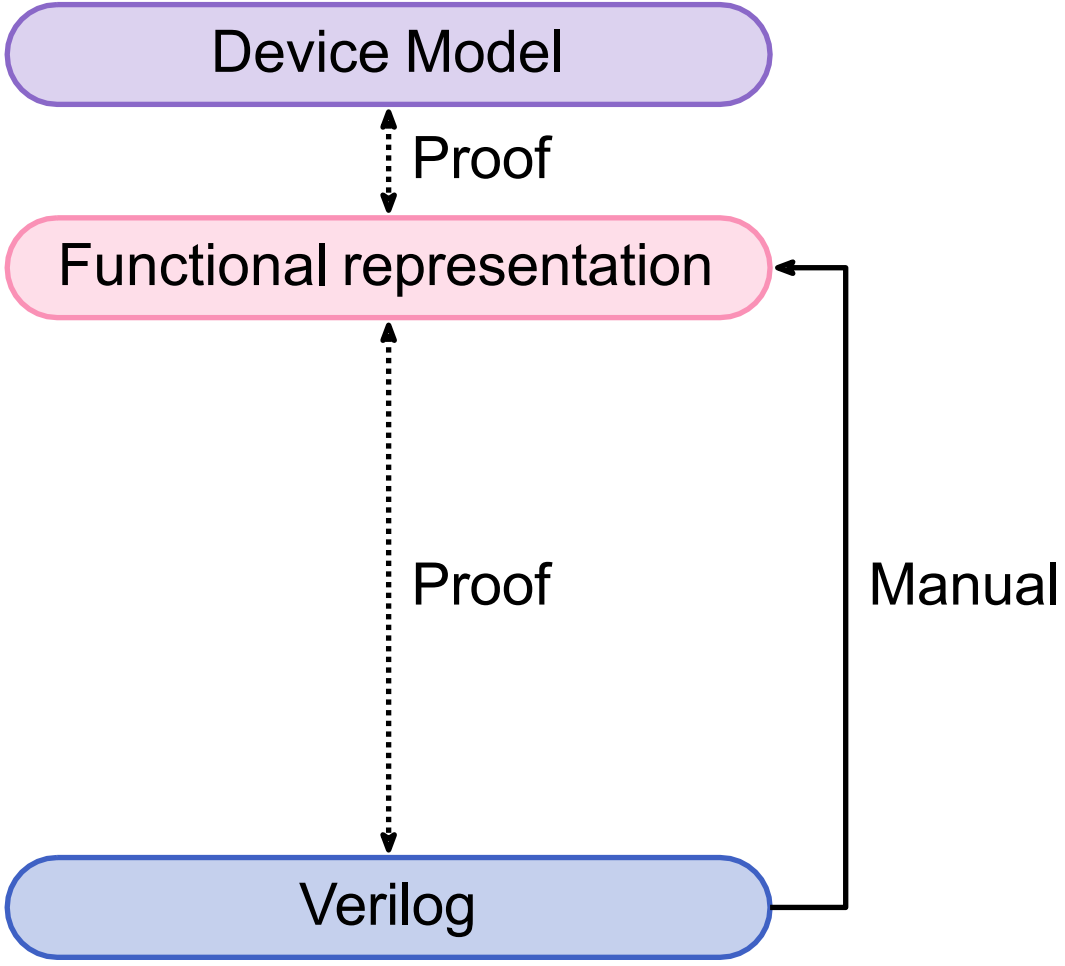
Workflow

- Workaround:



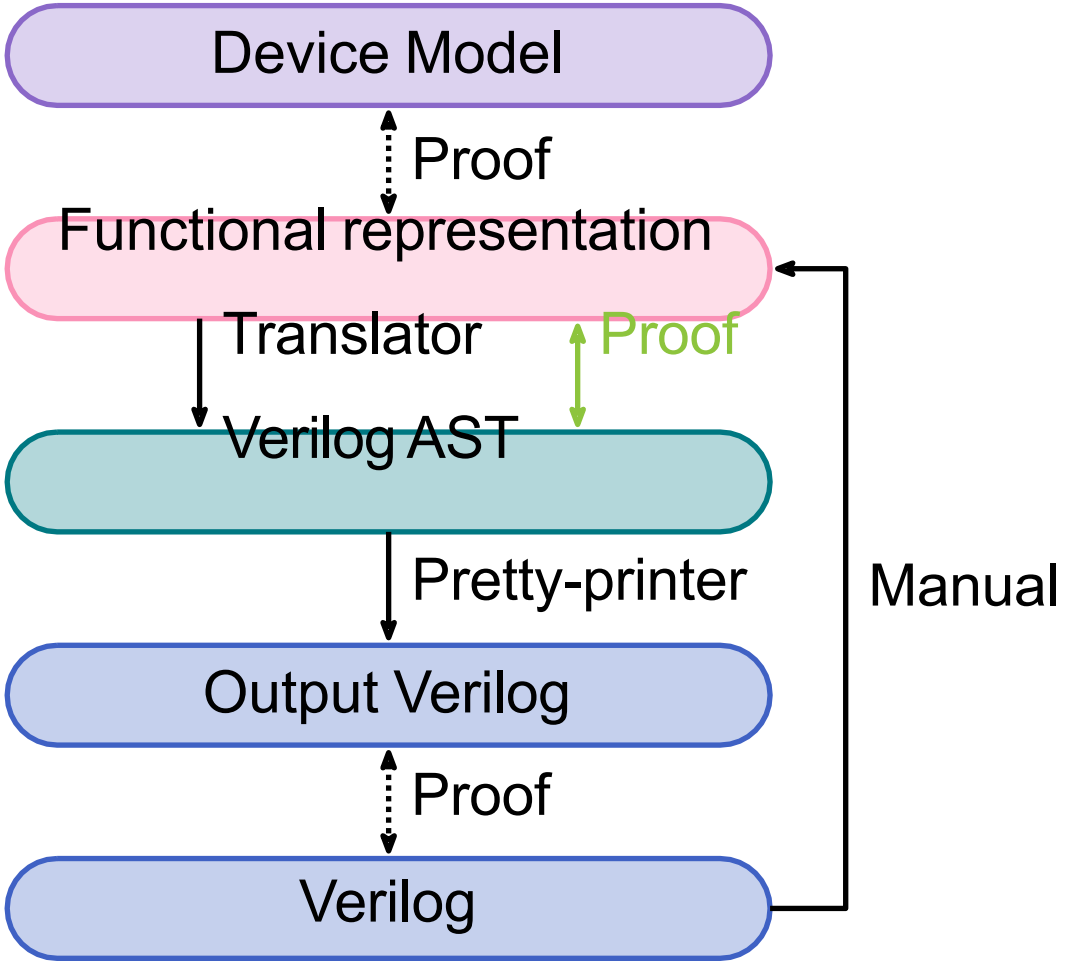
Workflow

- Workaround:
  - Manually translate Verilog to functional representation



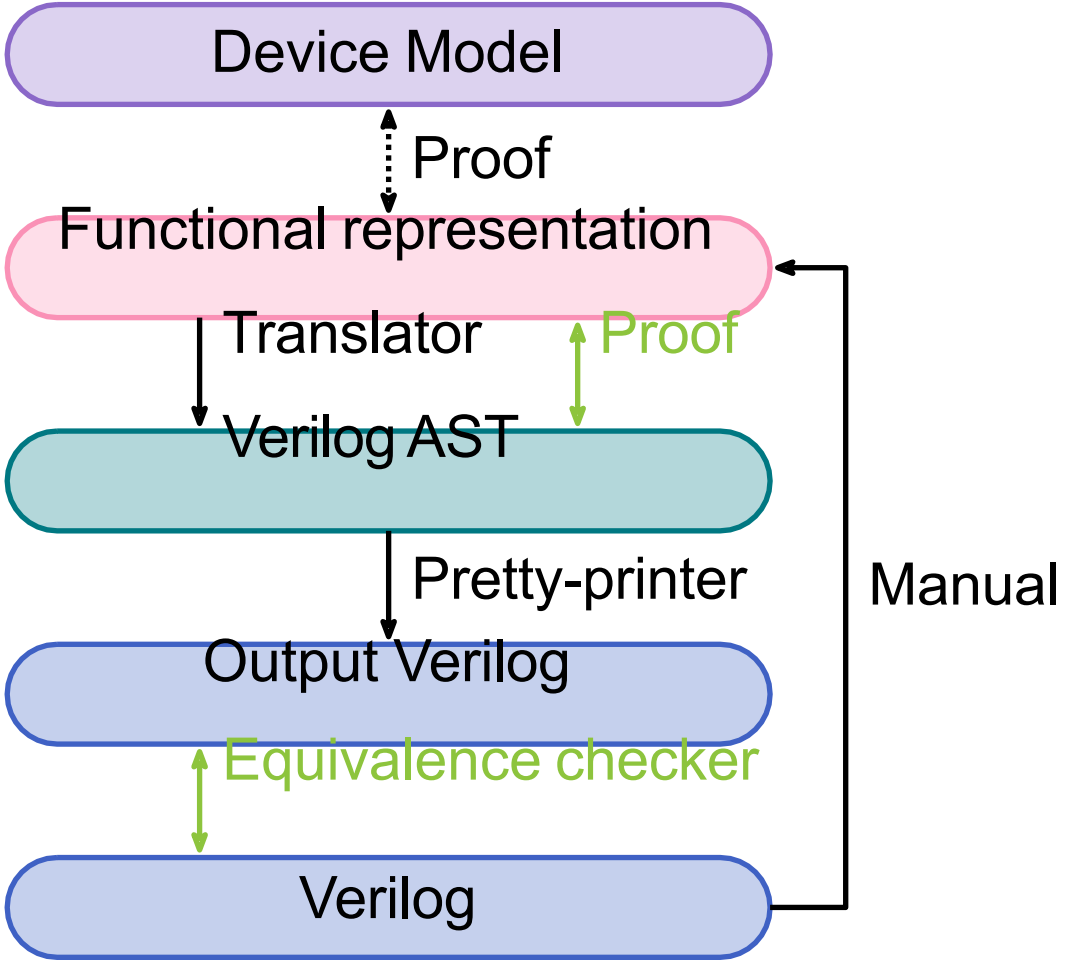
Workflow

- Workaround:
  - Manually translate Verilog to functional representation
  - Translate and export into Verilog



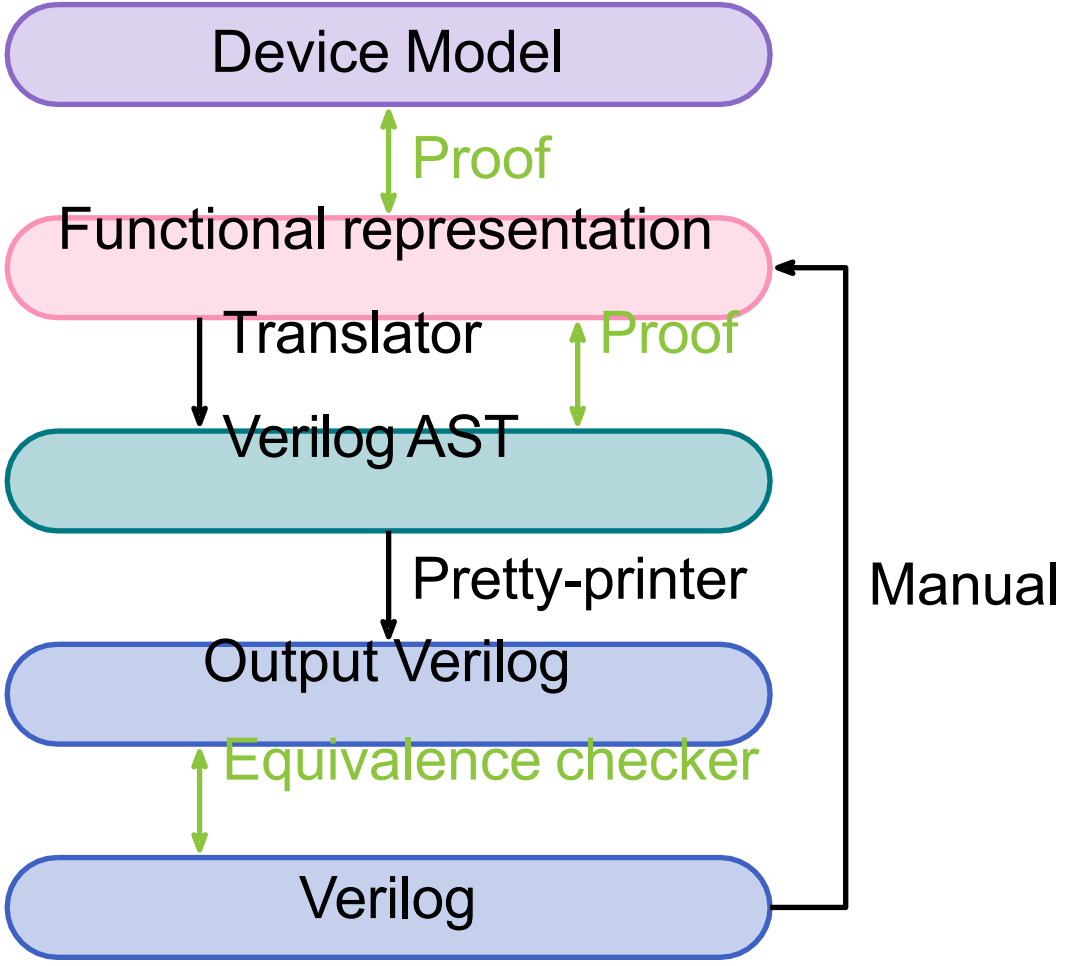
Workflow

- Workaround:
  - Manually translate Verilog to functional representation
  - Translate and export into Verilog
  - Prove equivalence to original Verilog



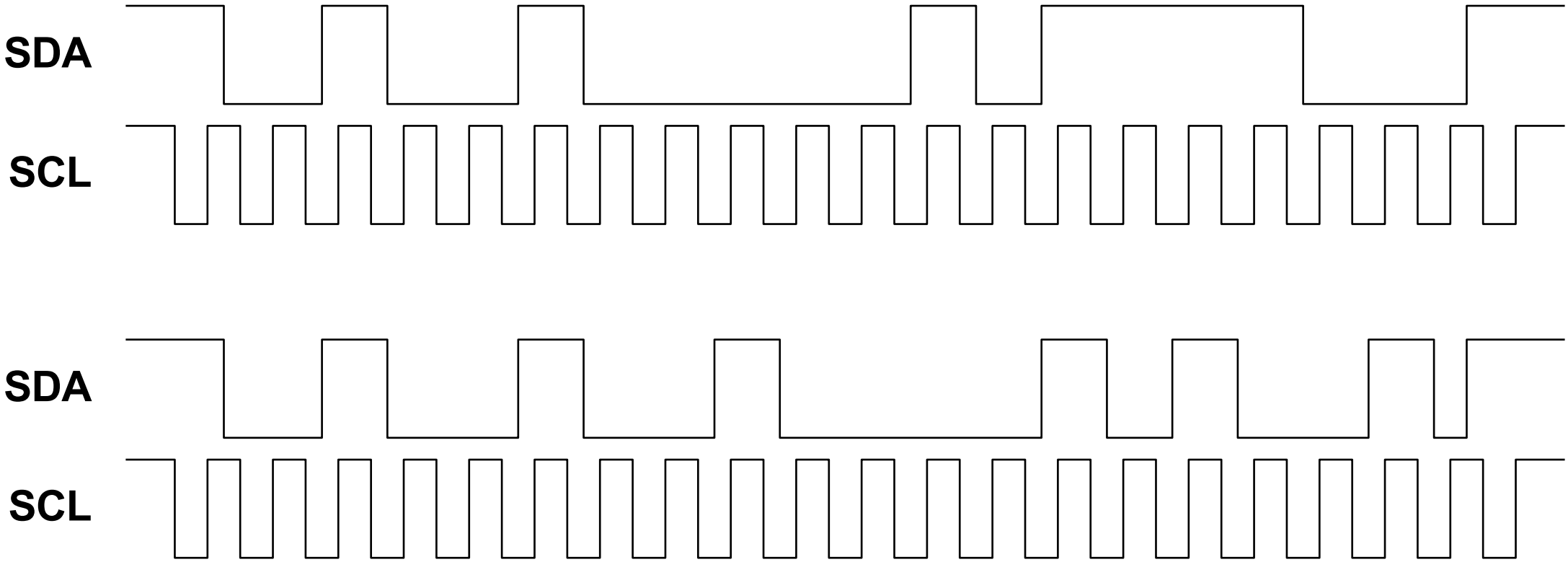
Workflow

- Workaround:
  - Manually translate Verilog to functional representation
  - Translate and export into Verilog
  - Prove equivalence to original Verilog
- Finally, prove that functional representation refines specification

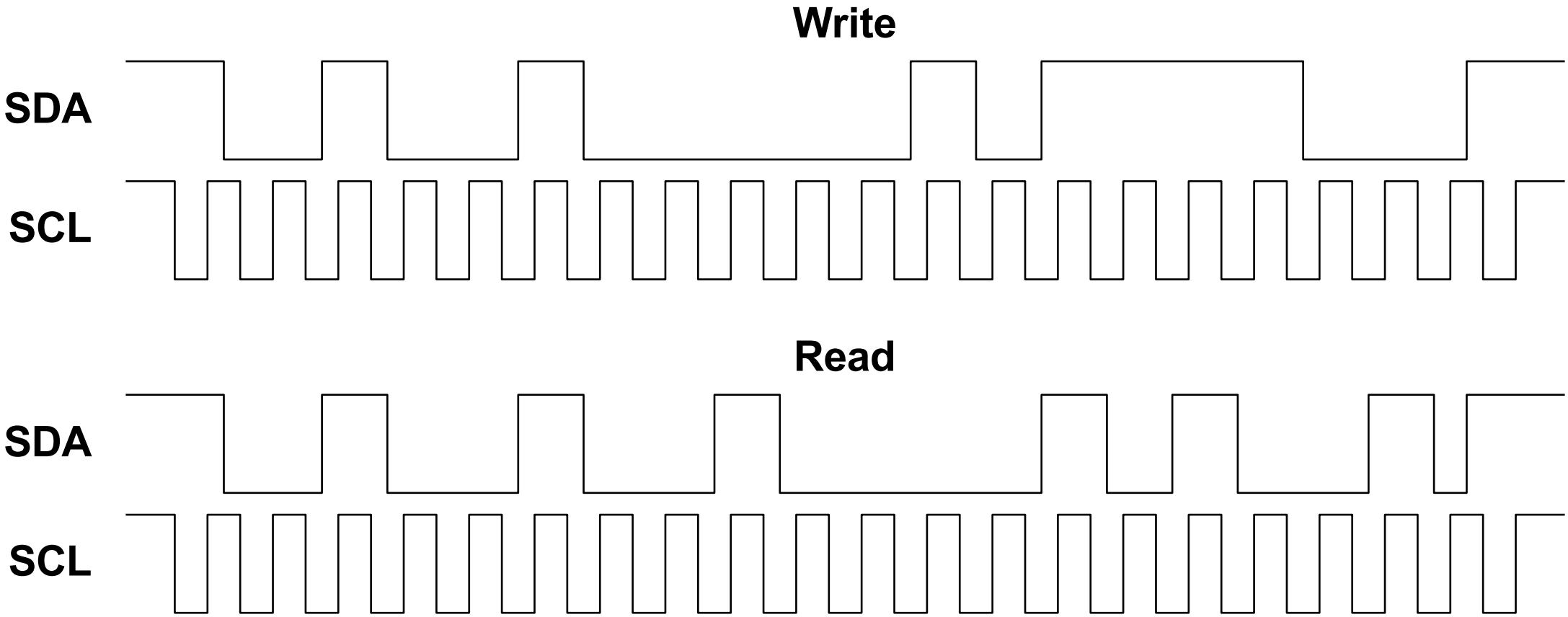


Specification Format

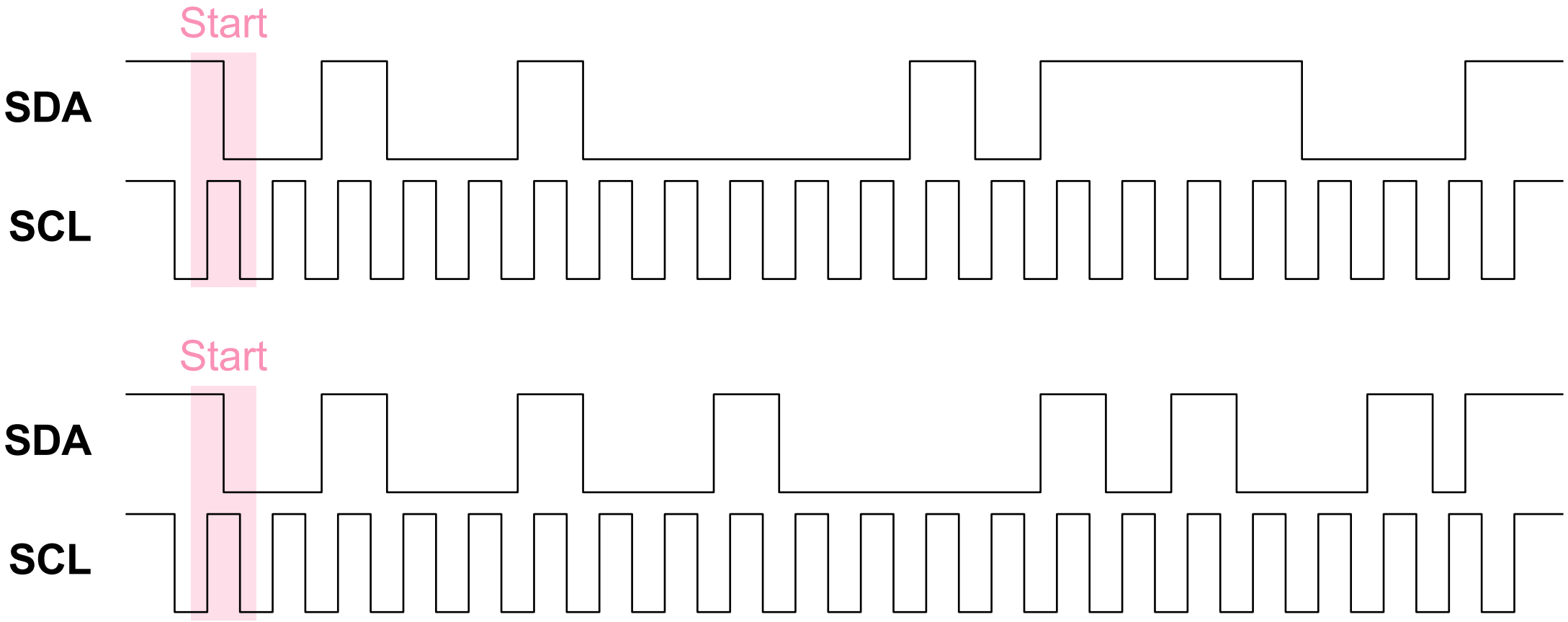
I²C



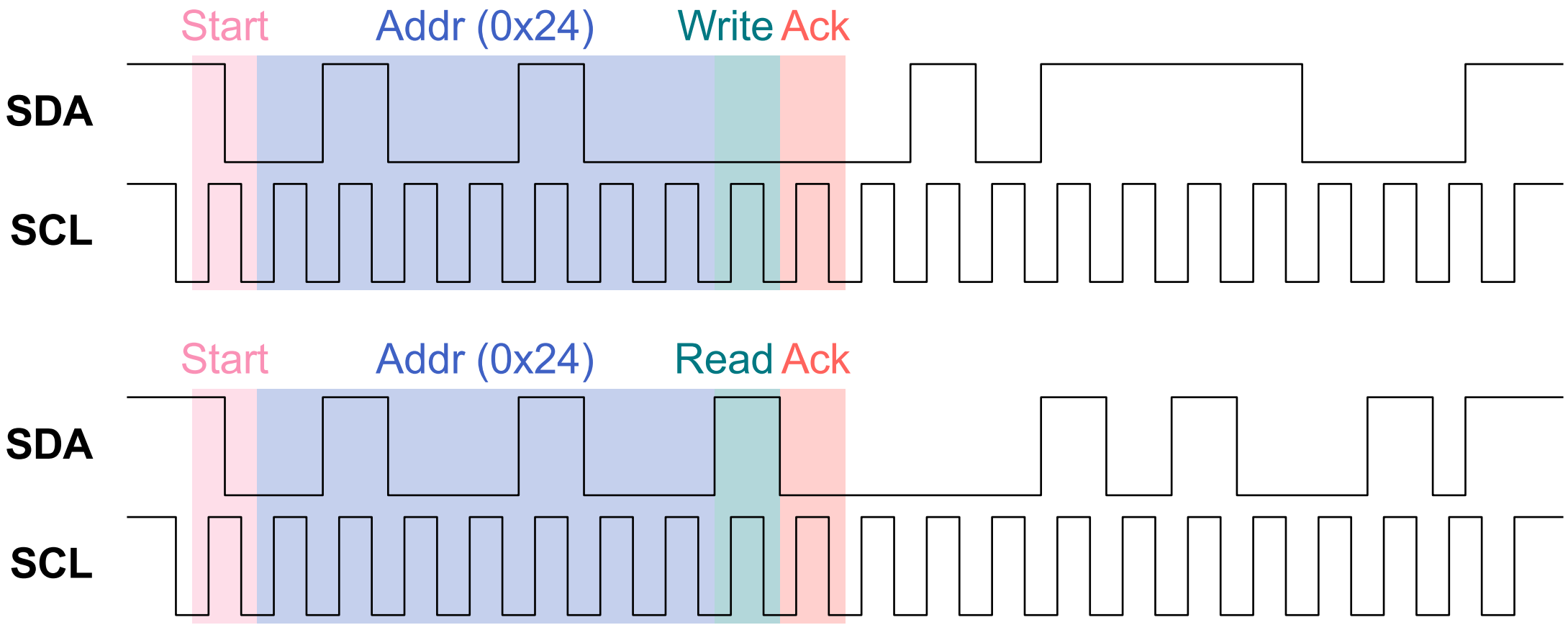
I²C



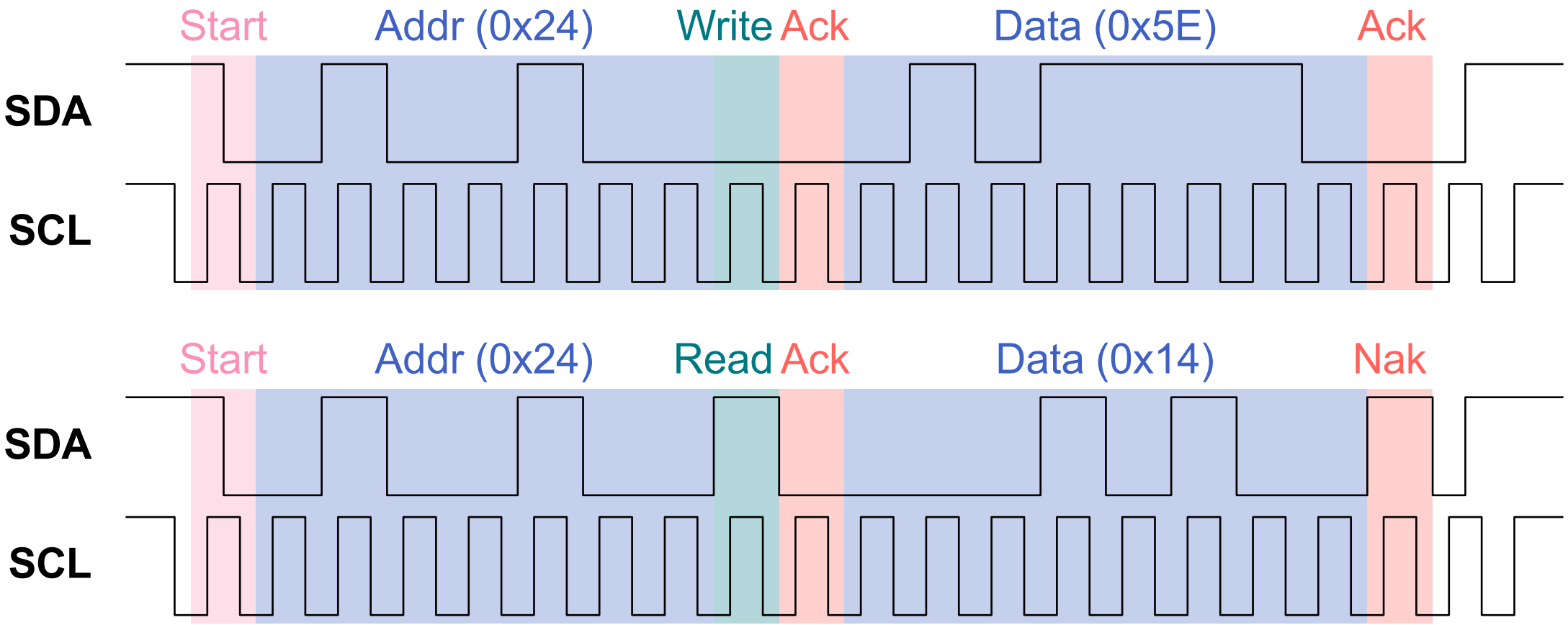
I²C



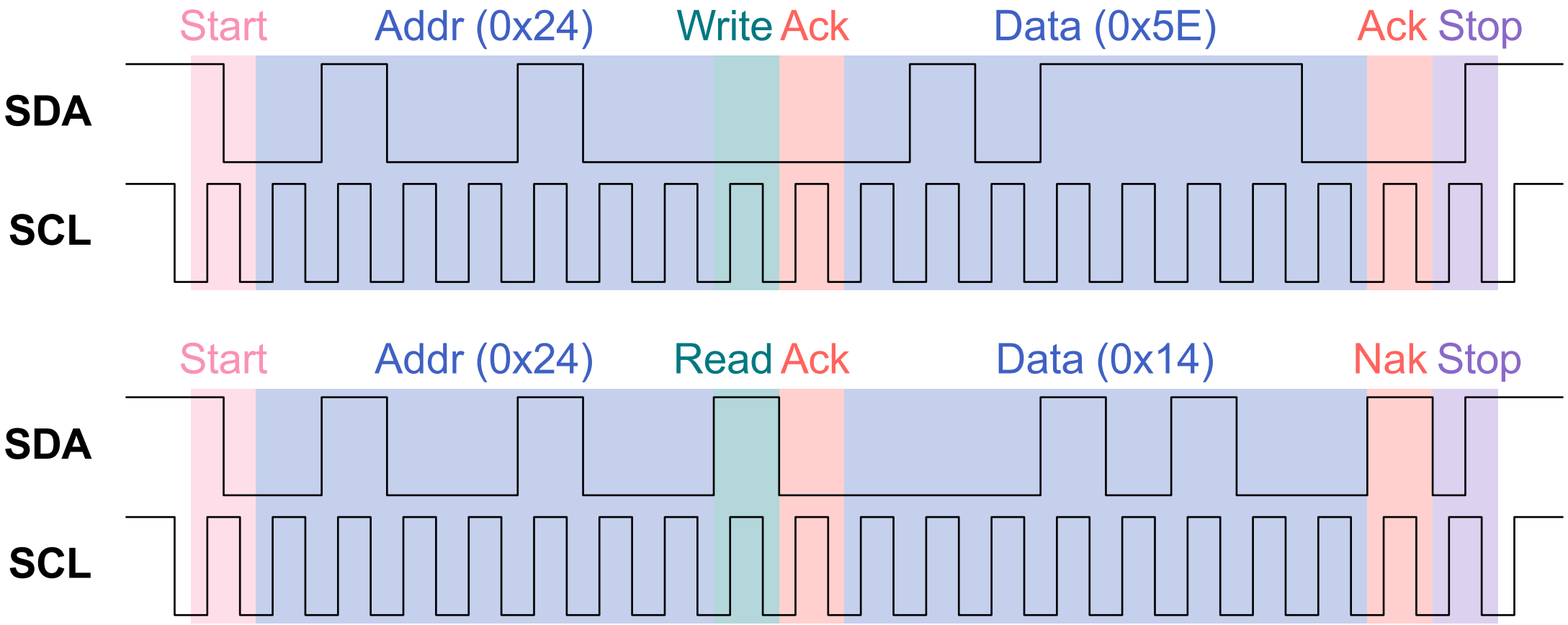
I²C


Start
Addr (0x24)
Write
Ack
SDA
SCL
Start
Addr (0x24)
Read
Ack
SDA
SCL

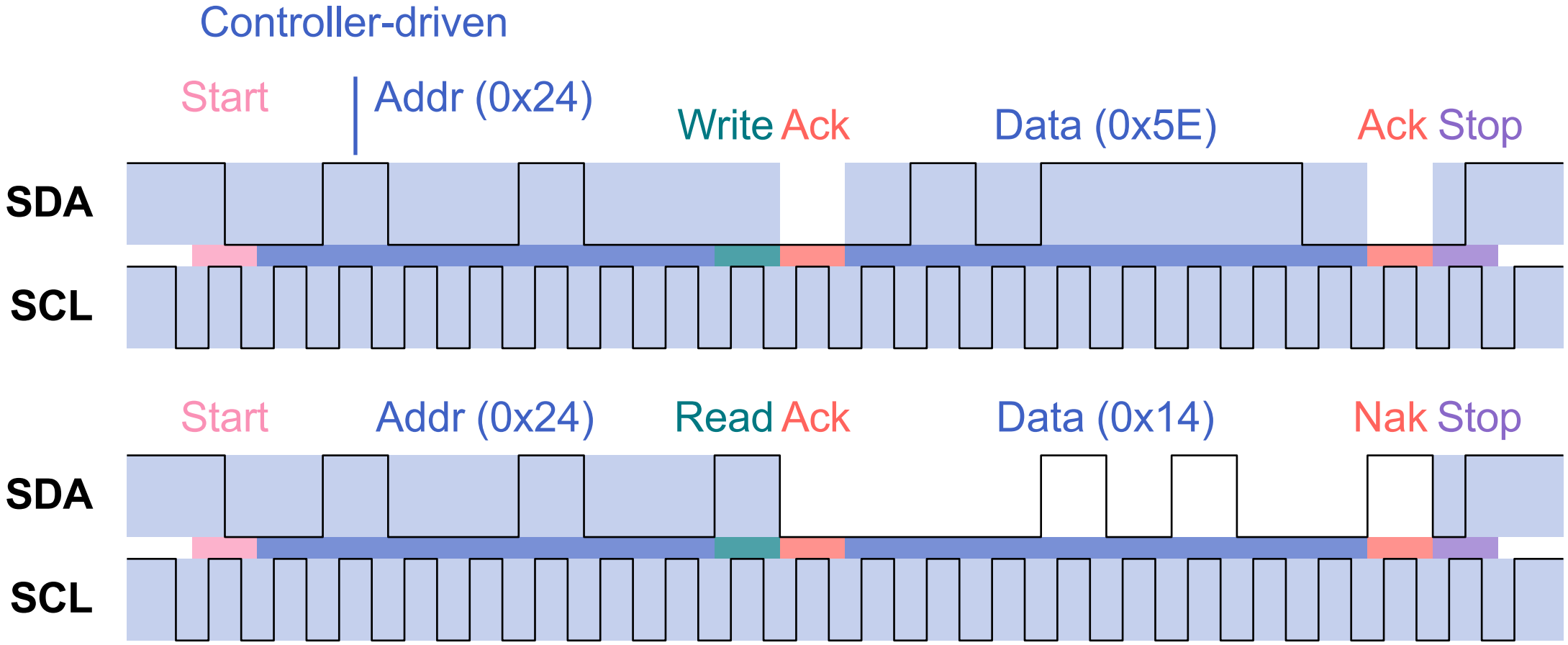
I²C


Start
Addr (0x24)
Write
Ack
Data (0x5E)
Ack
SDA
SCL
Start
Addr (0x24)
Read
Ack
Data (0x14)
Nak
SDA
SCL

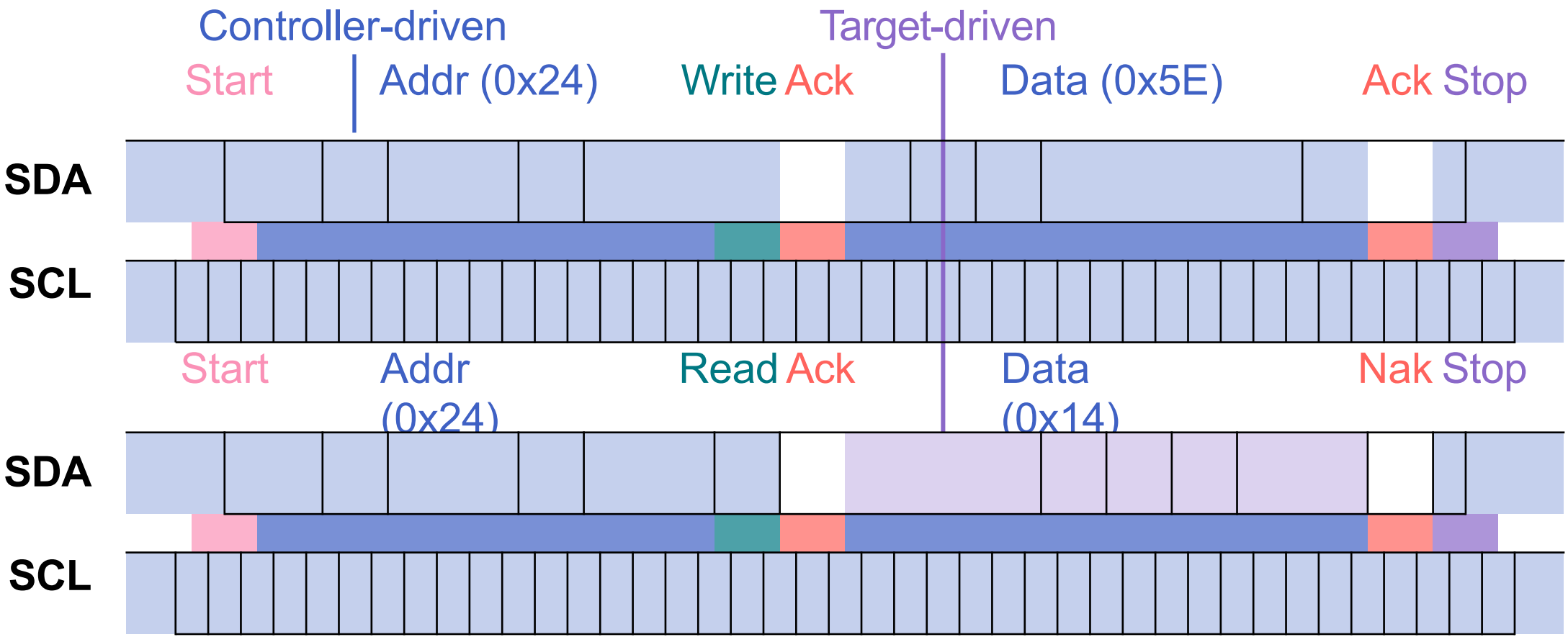
I²C


Start
Addr (0x24)
Write
Ack
Data (0x5E)
Ack
Stop
SDA
SCL
Start
Addr (0x24)
Read
Ack
Data (0x14)
Nak
Stop
SDA
SCL

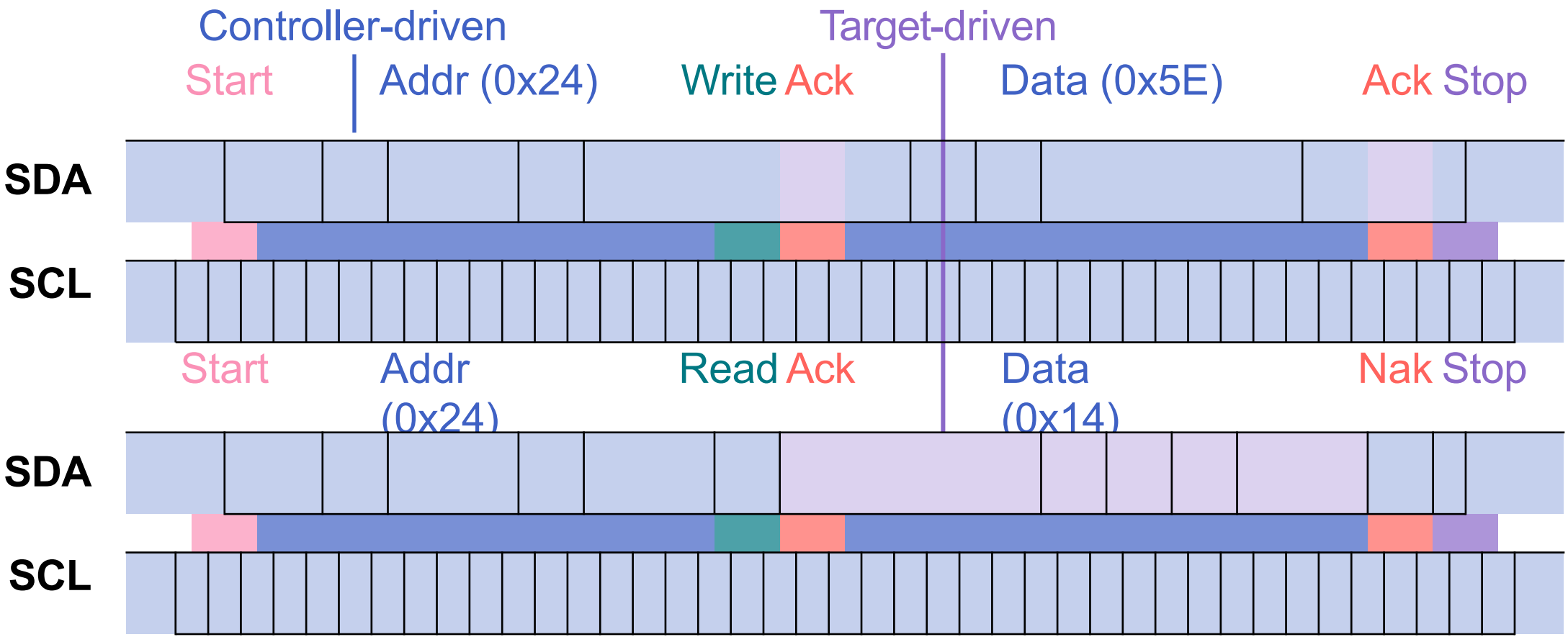
I²C


Controller-driven
Start
Addr (0x24)
Write
Ack
Data (0x5E)
Ack
Stop
SDA
SCL
Start
Addr (0x24)
Read
Ack
Data (0x14)
Nak
Stop
SDA
SCL

I²C


Controller-driven
Target-driven
Start
Addr (0x24)
Write
Ack
Data (0x5E)
Ack
Stop
SDA
SCL
Start
Addr (0x24)
Read
Ack
Data (0x14)
Nak
Stop
SDA
SCL

I²C


Controller-driven
Target-driven
Start
Addr (0x24)
Write
Ack
Data (0x5E)
Ack
Stop
SDA
SCL
Start
Addr (0x24)
Read
Ack
Data (0x14)
Nak
Stop
SDA
SCL

OpenTitan I²C

enablehost

`false`

fmt_fifo

[]

rx_fifo

[]

OpenTitan I²C

enablehost

`false`

fmt_fifo

[Wr 0x48 {start}]

rx_fifo

[]

⚡st FDATA ← Wr 0x48 {start}

OpenTitan I²C

enablehost

`false`

fmt_fifo

[Wr 0x48 {start}, Wr 0x5E {stop}]

rx_fifo

[]

⚡st FDATA ← Wr 0x5E {stop}

OpenTitan I²C

enablehost

`true`

fmt_fifo

[Wr 0x48 {start}, Wr 0x5E {stop}]

rx_fifo

[]

⚡st CTRL ← 1

OpenTitan I²C

enablehost

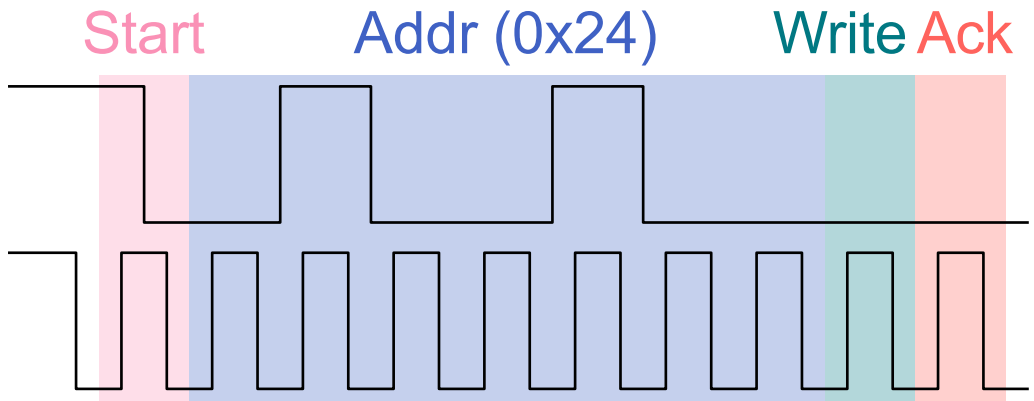
`true`

fmt_fifo

`[Wr 0x48 {start}, Wr 0x5E {stop}]`

rx_fifo

`[]`



OpenTitan I²C

enablehost

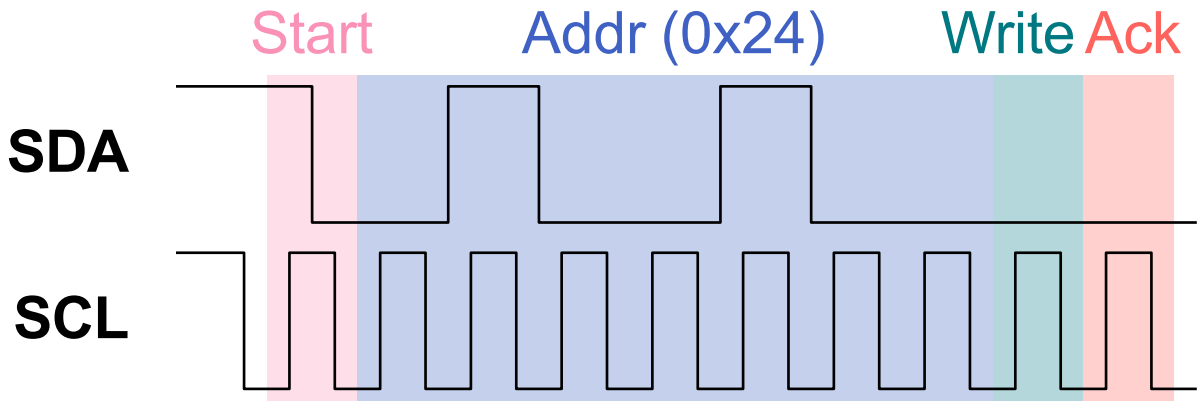
`true`

fmt_fifo

`[Wr 0x5E {stop}]`

rx_fifo

`[]`



OpenTitan I²C

enablehost

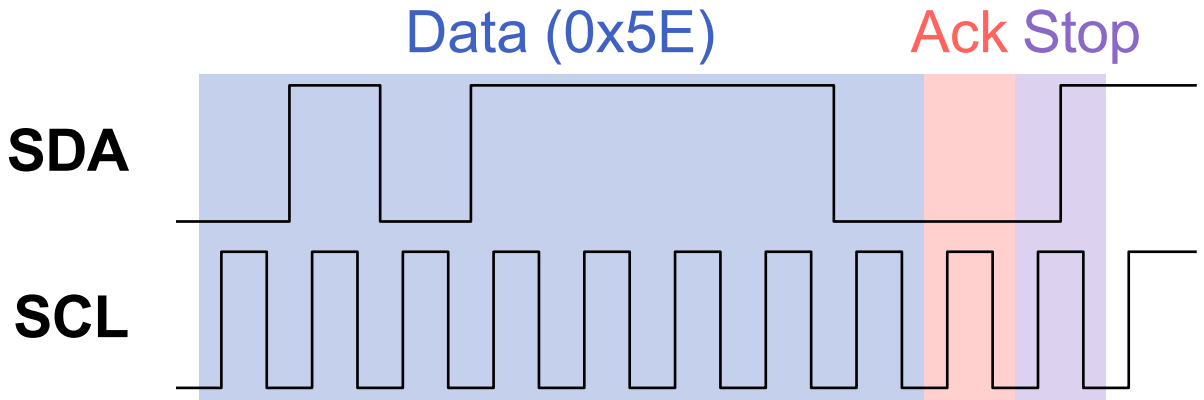
`true`

fmt_fifo

[Wr 0x5E {stop}]

rx_fifo

[]



OpenTitan I²C

enablehost

`true`

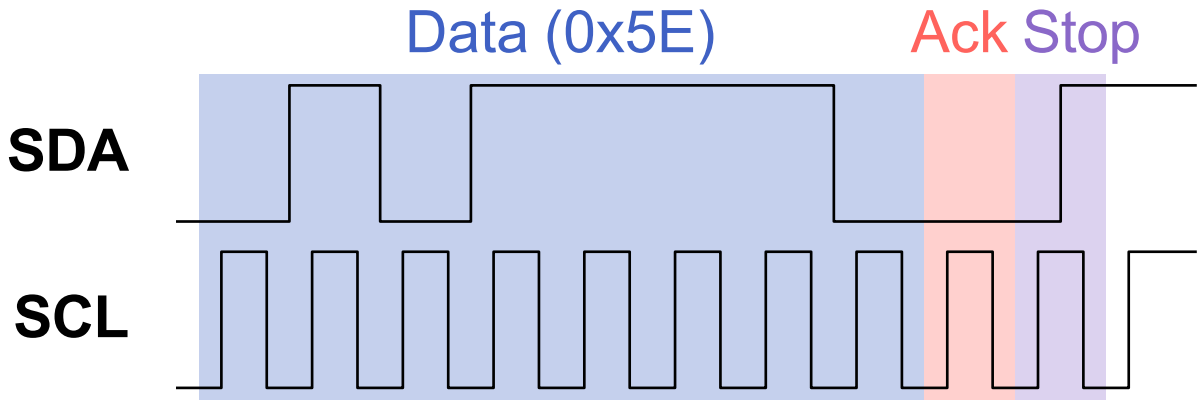
fmt_fifo

[]

rx_fifo

[]

⚡Interrupt!



OpenTitan I²C

enablehost

`true`

fmt_fifo

[Wr 0x49 {start}]

rx_fifo

[]

⚡st FDATA ← Wr 0x49 {start}

OpenTitan I²C

enablehost

`true`

fmt_fifo

[Wr 0x49 {start}, Rd 3 {stop}]

rx_fifo

[]

⚡st FDATA ← Rd 3 {stop}

OpenTitan I²C

enablehost

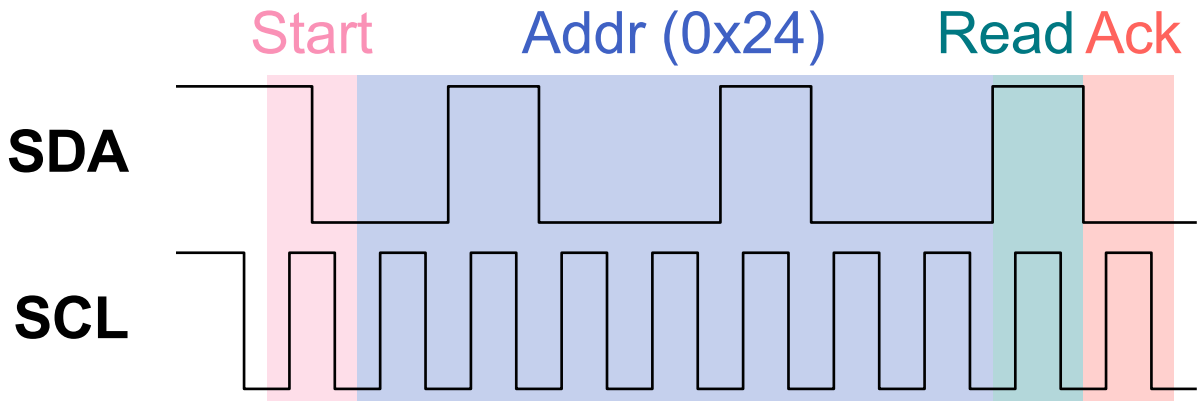
`true`

fmt_fifo

[Wr 0x49 {start}, Rd 3 {stop}]

rx_fifo

[]



OpenTitan I²C

enablehost

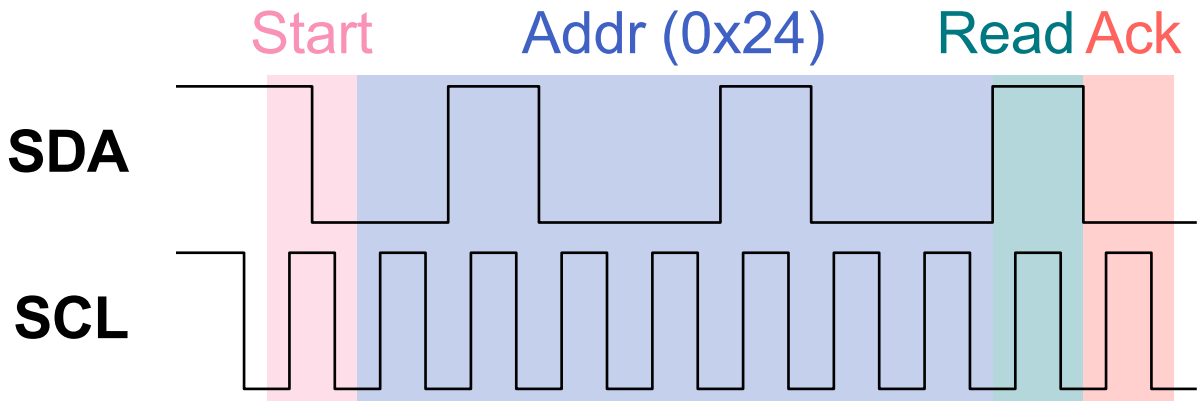
`true`

fmt_fifo

[Rd 3 {stop}]

rx_fifo

[]



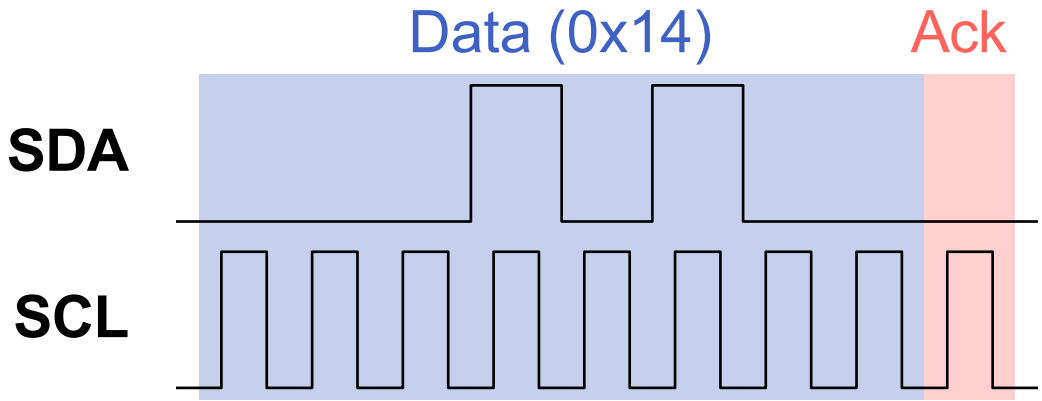
OpenTitan I²C

enablehost
`true`

fmt_fifo
[Rd 3 {stop}]

rx_fifo
[0x14]

⚡Interrupt!



OpenTitan I²C

enablehost

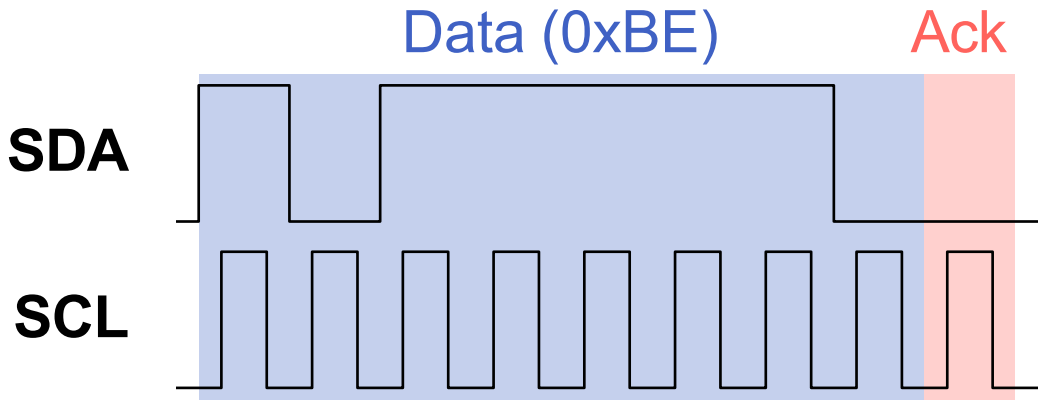
`true`

fmt_fifo

[Rd 3 {stop}]

rx_fifo

[0x14, 0xBE]



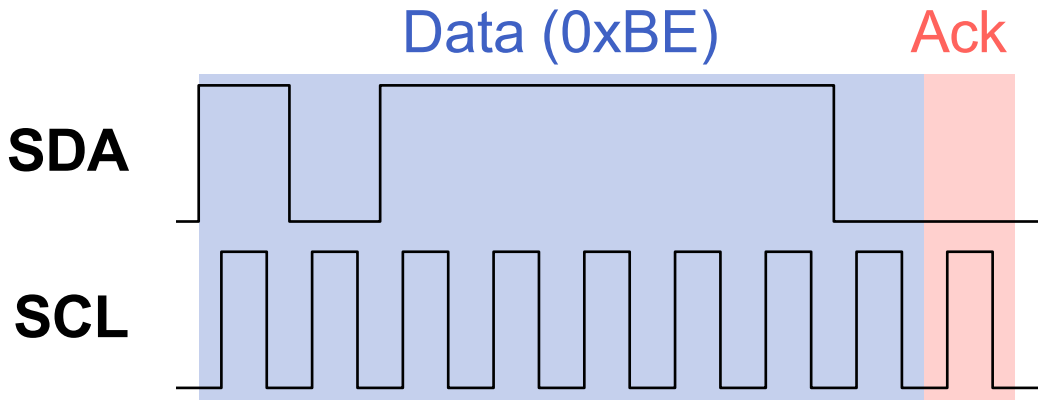
OpenTitan I²C

enablehost
`true`

fmt_fifo
[Rd 3 {stop}]

rx_fifo
[0xBE]

⚡ld RDATA → 0x14



OpenTitan I²C

enablehost

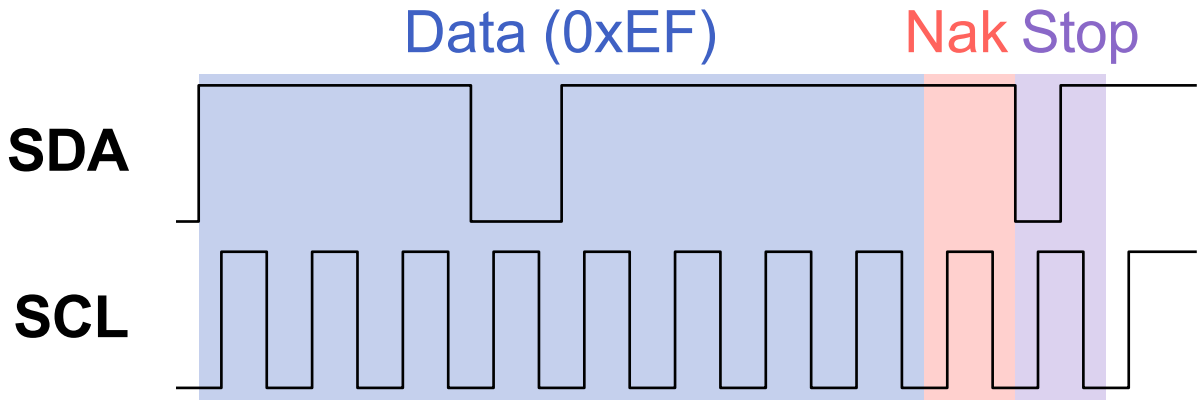
`true`

fmt_fifo

[Rd 3 {stop}]

rx_fifo

[0xBE, 0xEF]



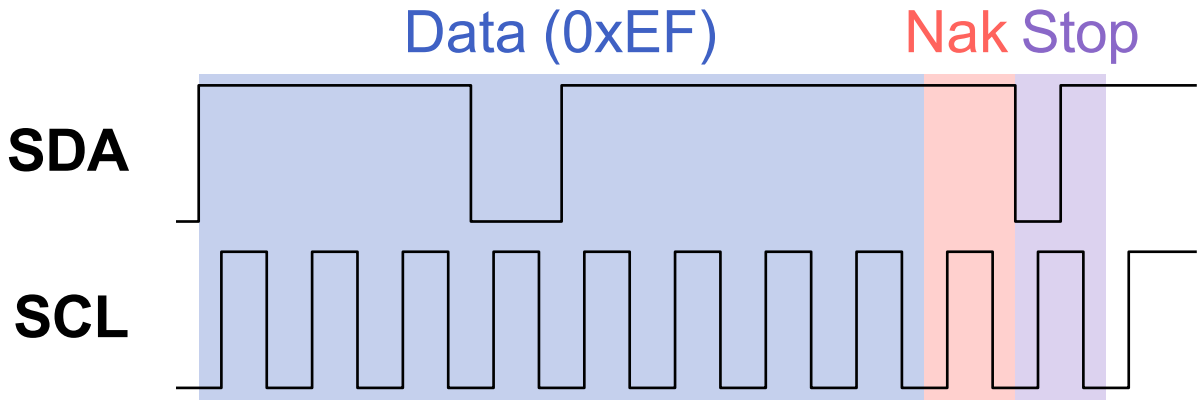
OpenTitan I²C

enablehost
`true`

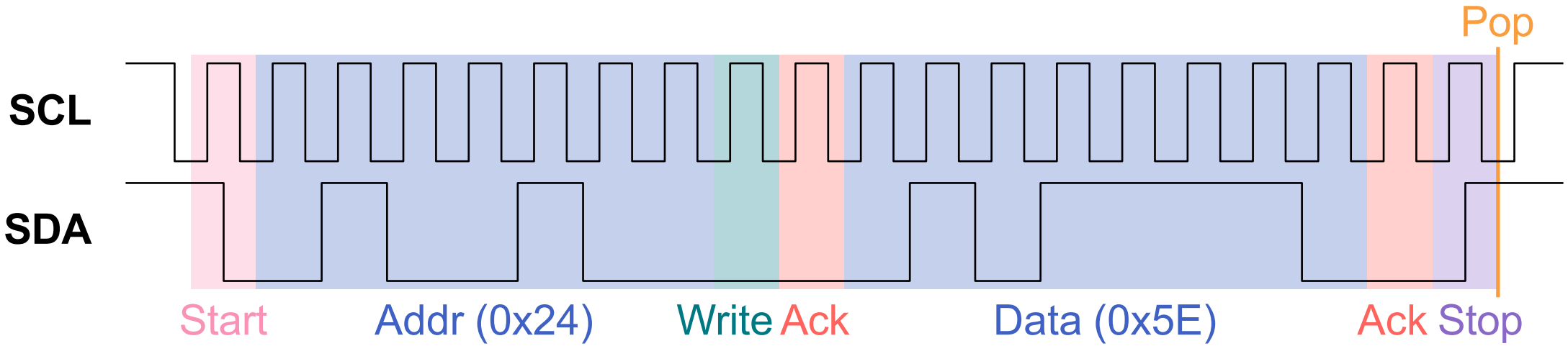
fmt_fifo
[]

rx_fifo
[0xBE, 0xEF]

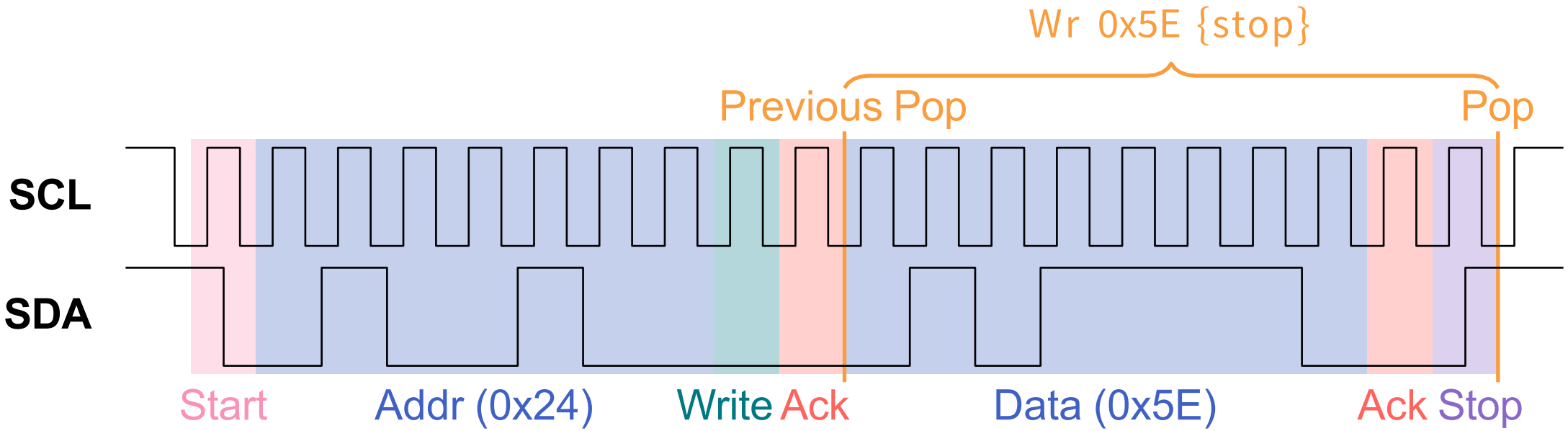
⚡Interrupt!



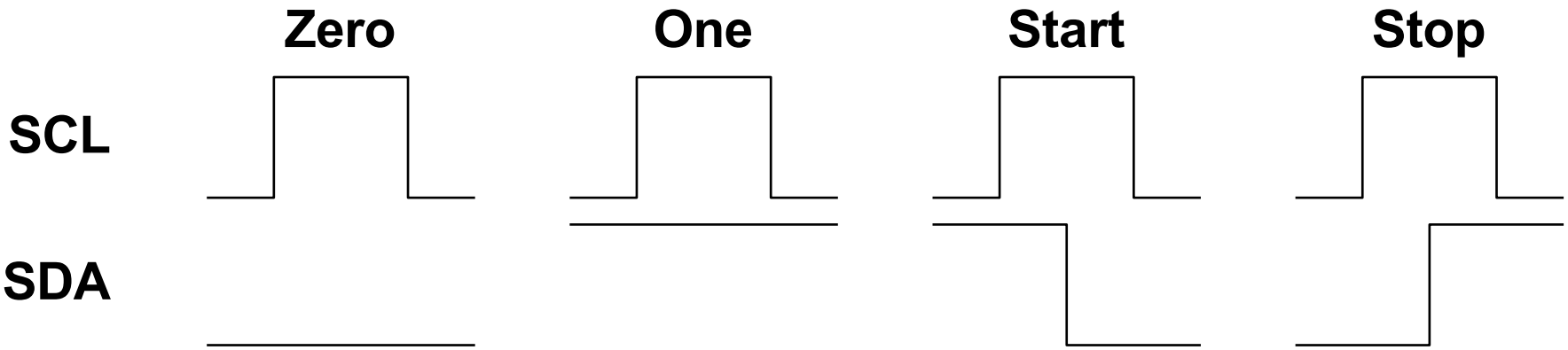
OpenTitan I²C



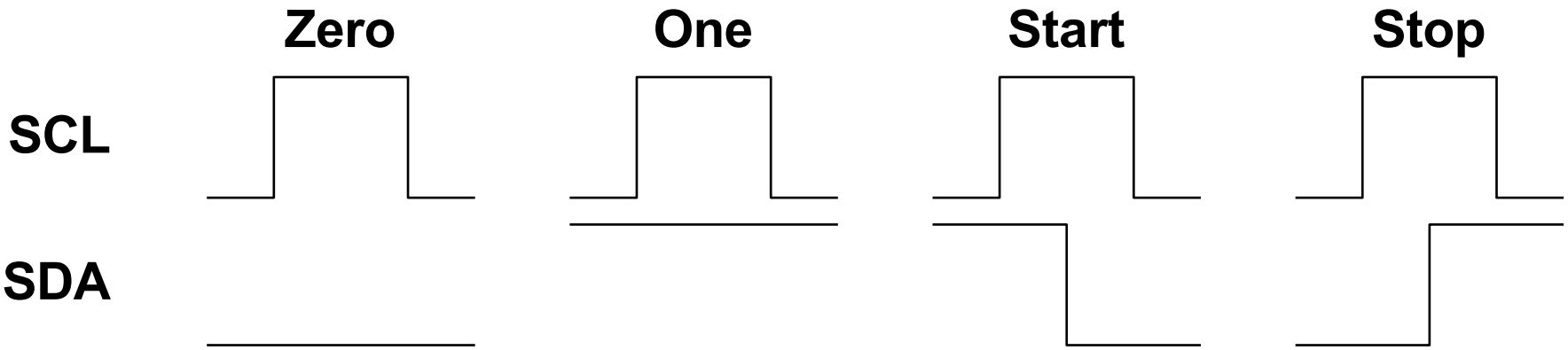
OpenTitan I²C



OpenTitan I²C

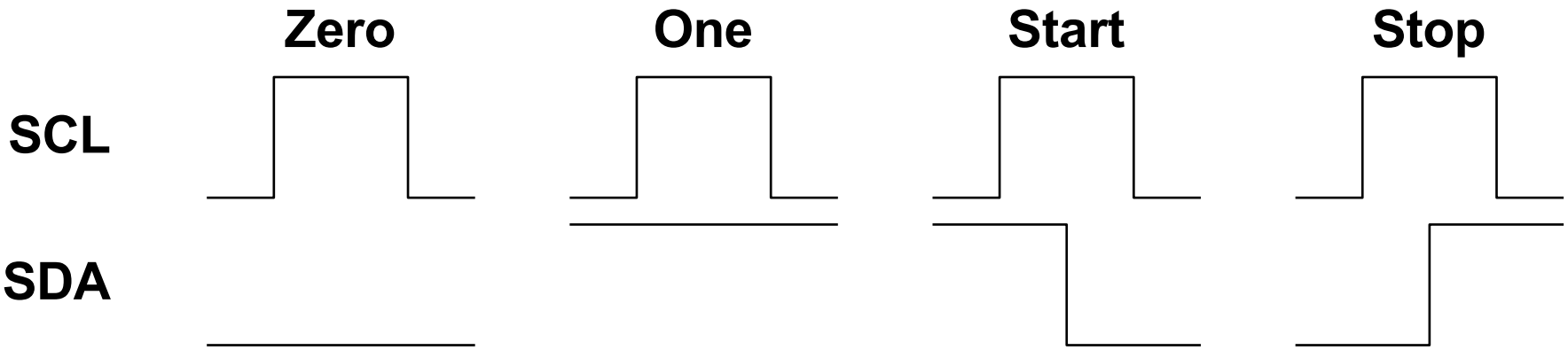


OpenTitan I²C



Wr 0x48 {start} → [Start, 0, 1, 0, 0, 1, 0, 0, 0, 0]

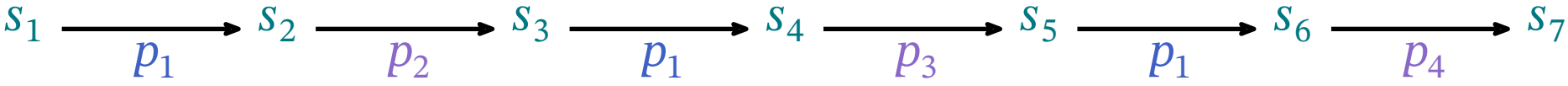
OpenTitan I²C



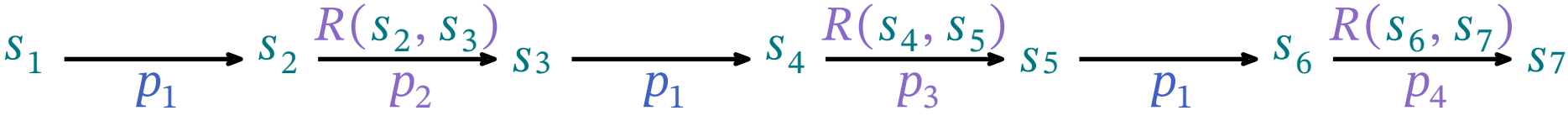
$$\text{Wr } 0x48\ \{\text{start}\} \rightarrow [\text{Start}, 0, 1, 0, 0, 1, 0, 0, 0, 0]$$

$$\text{Rd } 2\ \{\text{stop}\} \rightarrow [?,?,?,?,?,?,?,?,?, 0, ?,?,?,?,?,?,?,?, 1, \text{Stop}]$$

Rely-Guarantee



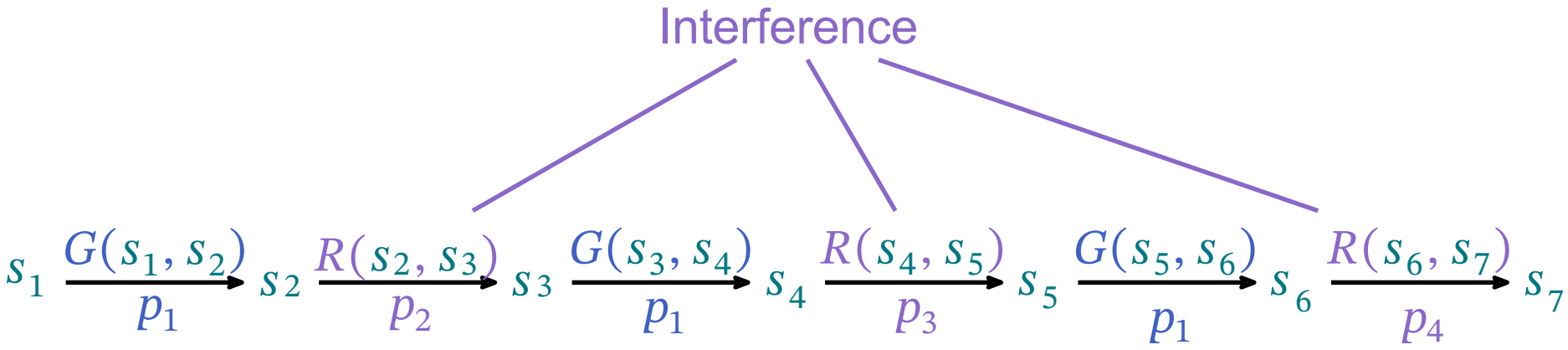
Rely-Guarantee



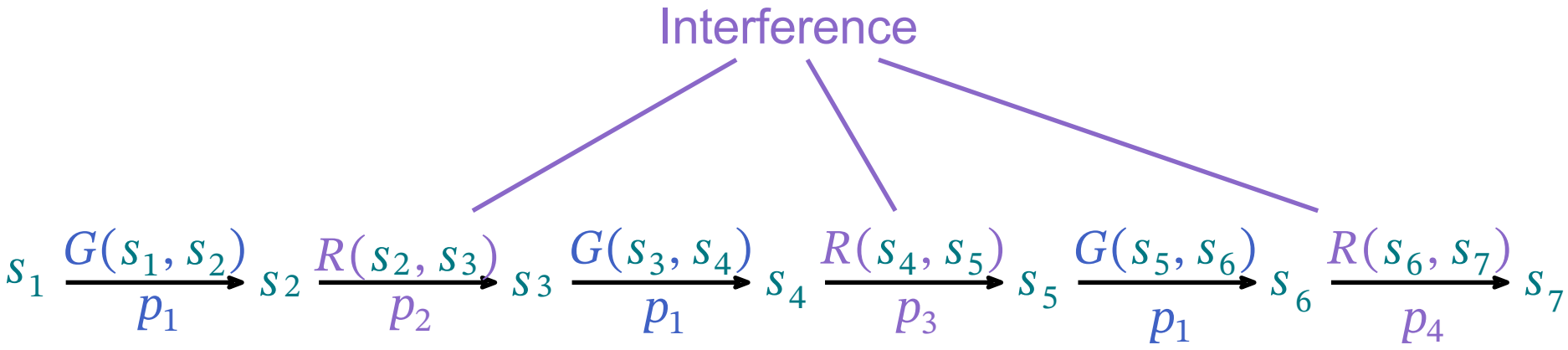
Rely-Guarantee

$$s_1 \xrightarrow[p_1]{G(s_1, s_2)} s_2 \xrightarrow[p_2]{R(s_2, s_3)} s_3 \xrightarrow[p_1]{G(s_3, s_4)} s_4 \xrightarrow[p_3]{R(s_4, s_5)} s_5 \xrightarrow[p_1]{G(s_5, s_6)} s_6 \xrightarrow[p_4]{R(s_6, s_7)} s_7$$

Rely-Guarantee

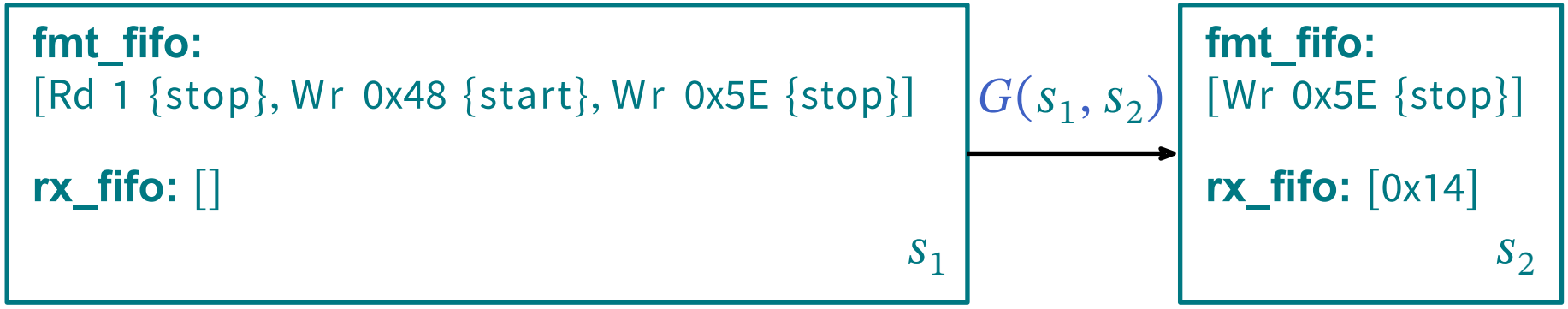


Rely-Guarantee

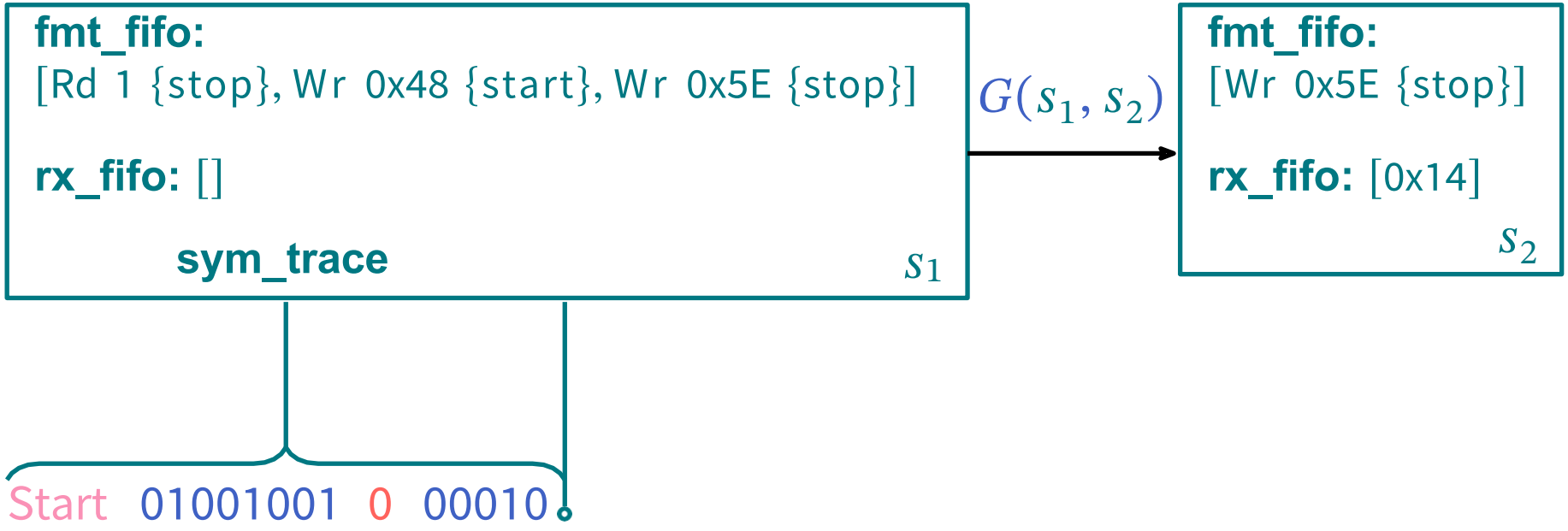


$$R(s_1, s_2) \triangleq s_2.x = s_1.x$$

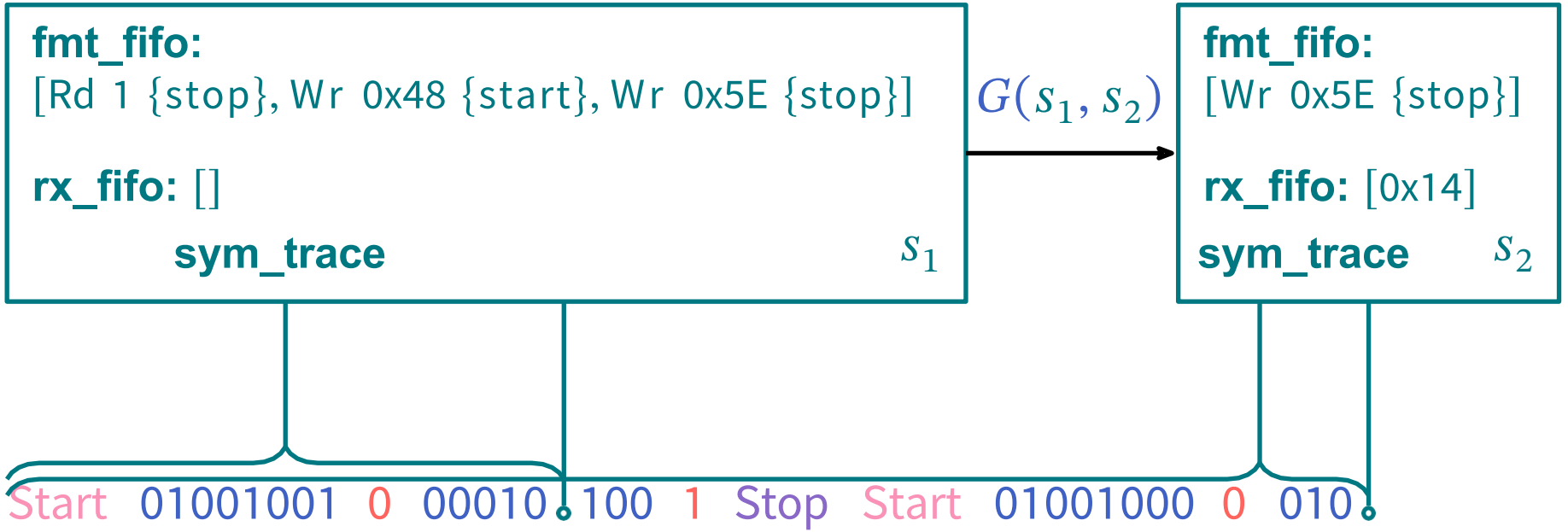
Rely-Guarantee



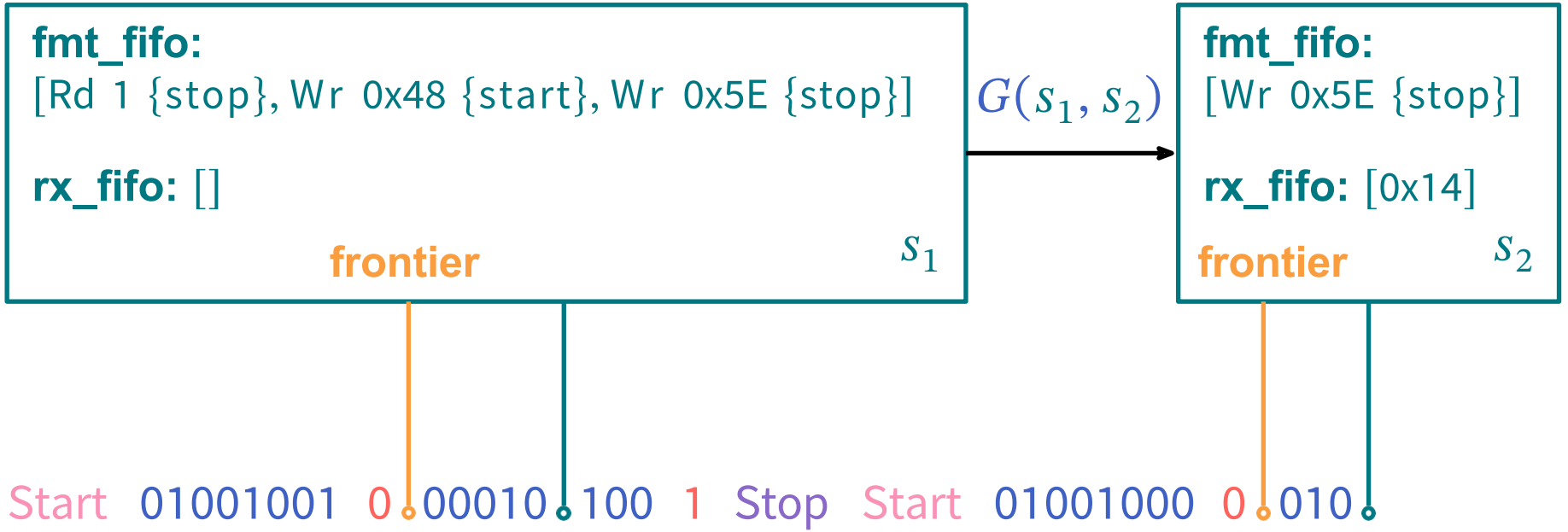
Rely-Guarantee



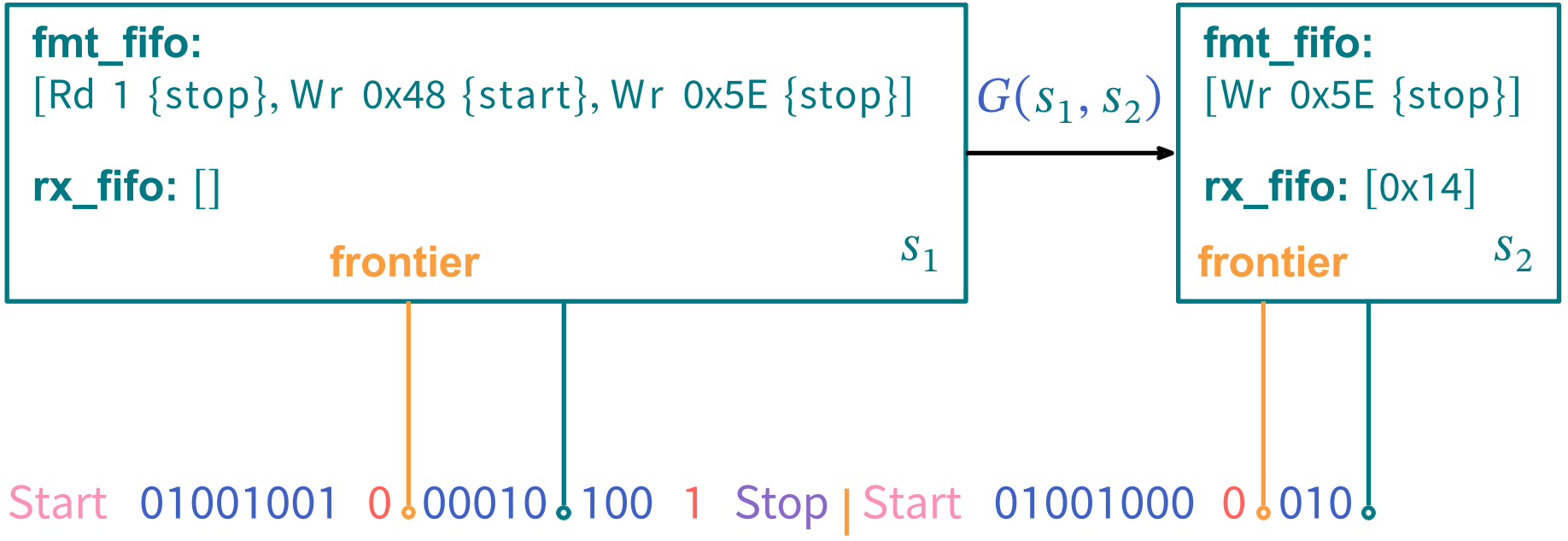
Rely-Guarantee



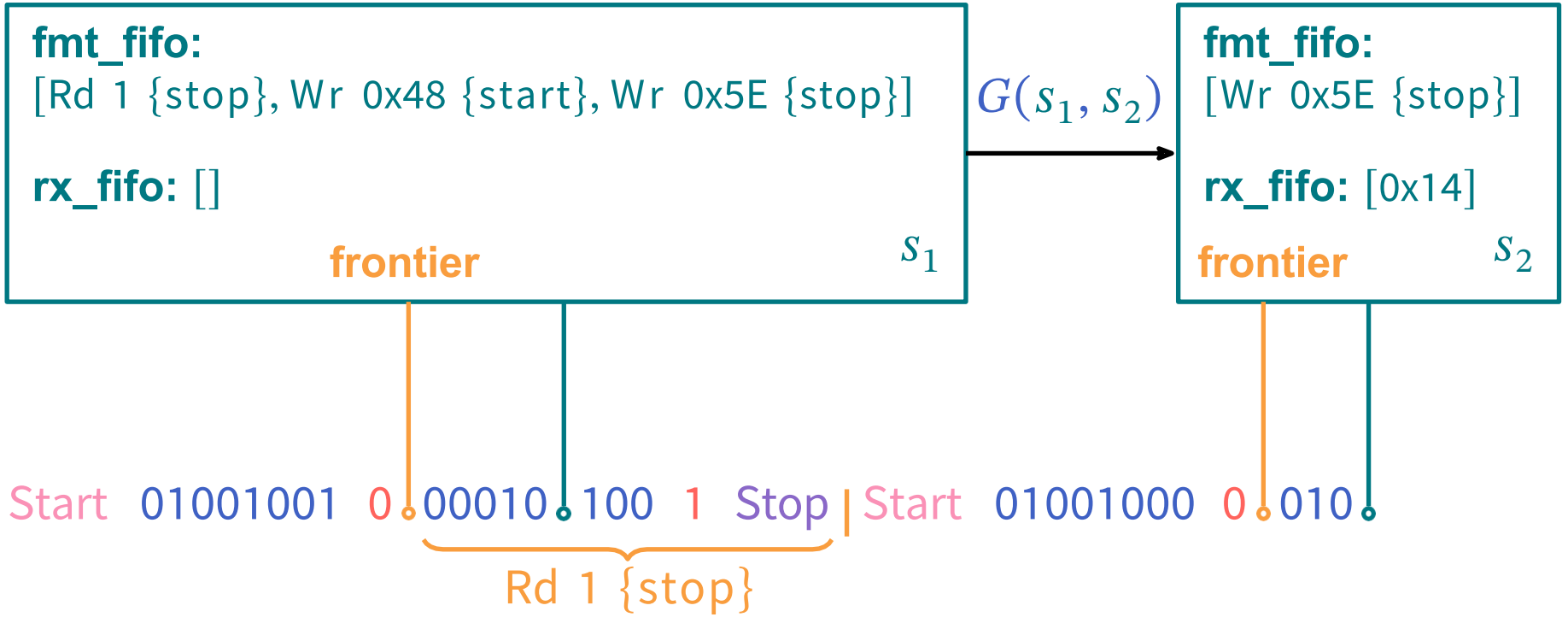
Rely-Guarantee



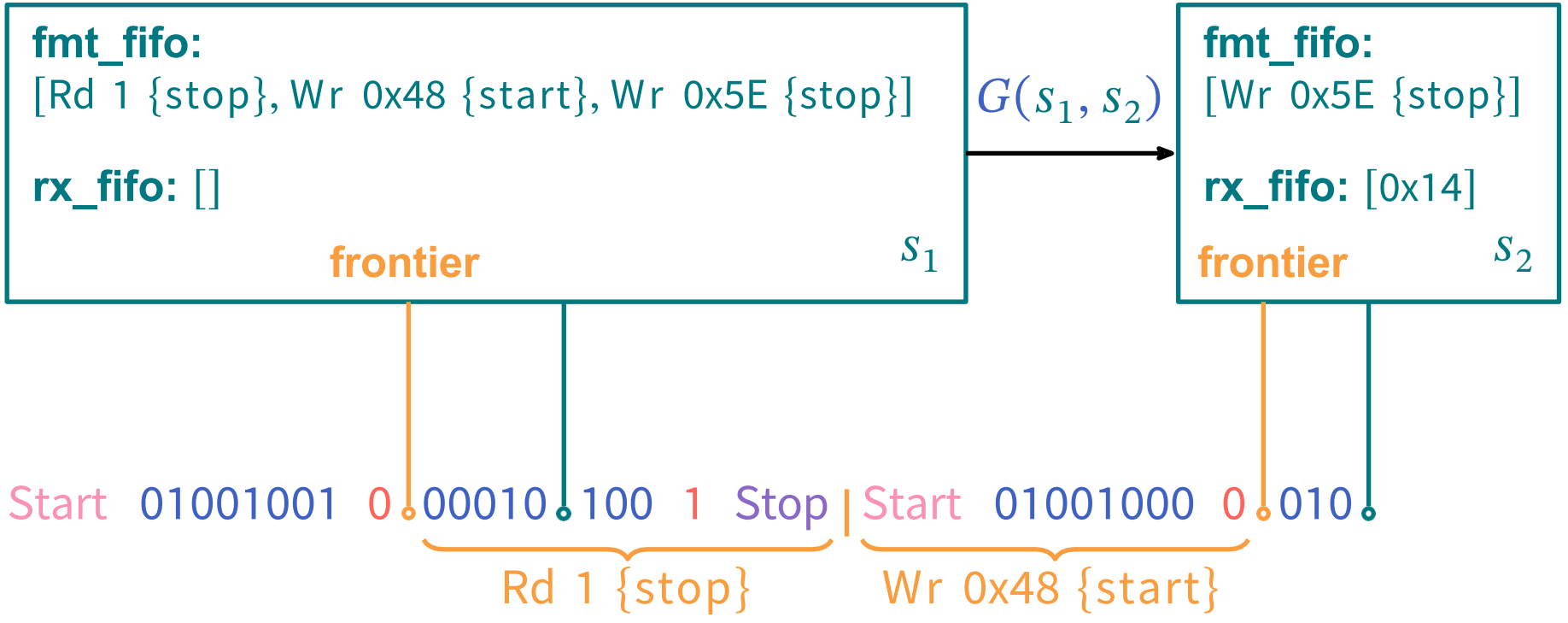
Rely-Guarantee



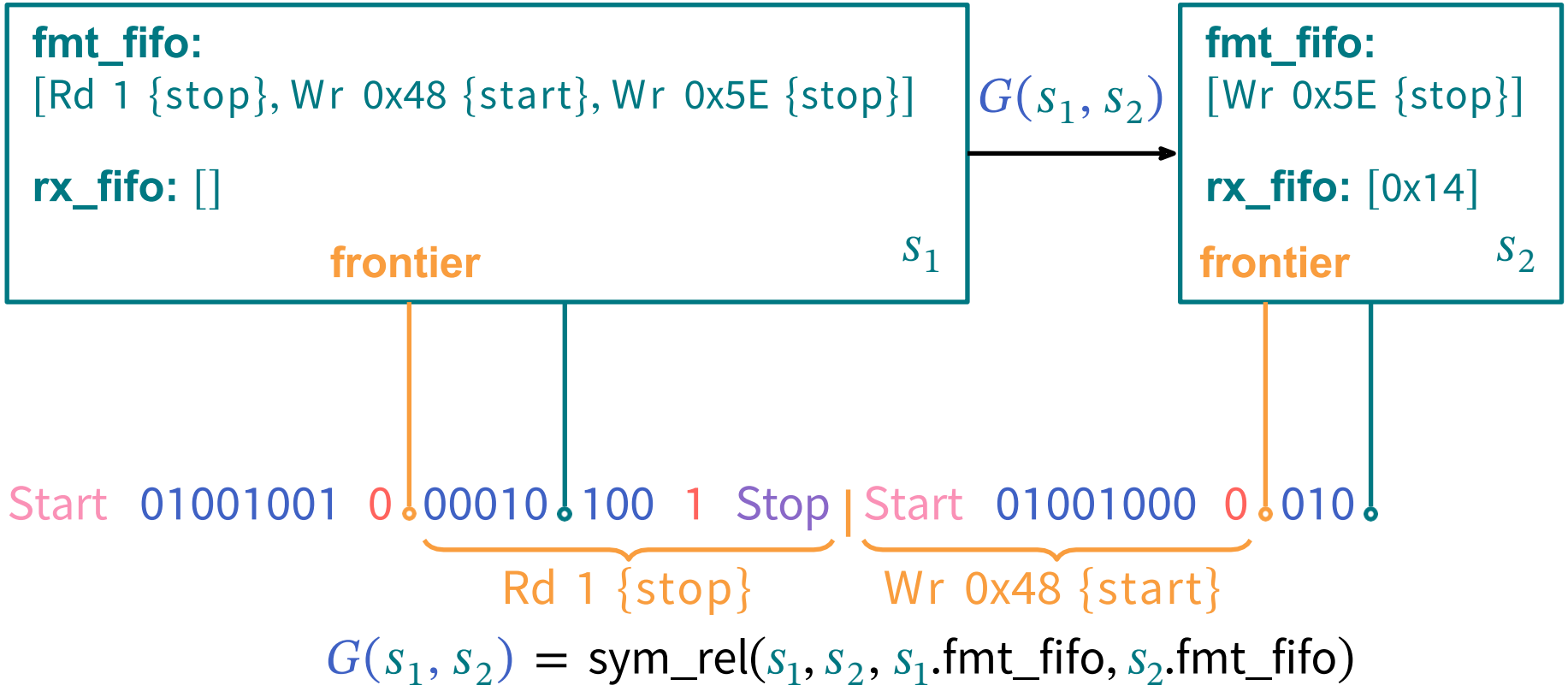
Rely-Guarantee



Rely-Guarantee



Rely-Guarantee



$$G(s_1, s_2) = \text{sym\_rel}(s_1, s_2, s_1.\text{fmt\_fifo}, s_2.\text{fmt\_fifo})$$

Rely-Guarantee

s_1 $\{|\text{fmt\_fifo}| < 64\}$

$\text{FDATA} \leftarrow \text{cmd}$

s_2 $\{\text{fmt\_fifo} = s_1.\text{fmt\_fifo} + [\text{cmd}]\}$

Driver Verification

Pancake-to-Viper

```
fun 1 mul(1 x, 1 y) {

  /@ requires y >= 0 @/
  /@ ensures retval == x * y @/
  var i = 0;
  var res = 0;
  while i < y {
    /@ invariant i <= y @/
    /@ invariant res == x * i @/
    res = res + x;
    /@ assert res == x * (i + 1) @/
    i = i + 1;
}
  return res;
}
```

```
function mul(gv:Ref, x:Int, y:Int)
  returns (retval: Int)
  requires y >= 0
  ensures retval == x * y
{ var i: Int := 0;
  var res: Int := 0;
  while (i < y)
    invariant i <= y && res == x*i
  {
    res := res + x;
    assert res == x * (i + 1);
    i := i + 1;
  }
  retval := res;
}
```

Pancake-to-Viper

```
fun 1 mul(1 x, 1 y) {

  /@ requires y >= 0 @/
  /@ ensures retval == x * y @/
  var i = 0;
  var res = 0;
  while i < y {
    /@ invariant i <= y @/
    /@ invariant res == x * i @/
    res = res + x;
    /@ assert res == x * (i + 1) @/
    i = i + 1;
}
  return res;
}
```

```
function mul(gv:Ref, x:Int, y:Int)
  returns (retval: Int)
  requires y >= 0
  ensures retval == x * y
{ var i: Int := 0;
  var res: Int := 0;
  while (i < y)
    invariant i <= y && res == x*i
  {
    res := res + x;
    assert res == x * (i + 1);
    i := i + 1;
  }
  retval := res;
}
```

Pancake-to-Viper

```
fun 1 mul(1 x, 1 y) {

  /@ requires y >= 0 @/
  /@ ensures retval == x * y @/
  var i = 0;
  var res = 0;
  while i < y {
    /@ invariant i <= y @/
    /@ invariant res == x * i @/
    res = res + x;
    /@ assert res == x * (i + 1) @/
    i = i + 1;
}
  return res;
}
```

```
function mul(gv:Ref, x:Int, y:Int)
  returns (retval: Int)
  requires y >= 0
  ensures retval == x * y
{ var i: Int := 0;
  var res: Int := 0;
  while (i < y)
    invariant i <= y && res == x*i
  {
    res := res + x;
    assert res == x * (i + 1);
    i := i + 1;
  }
  retval := res;
}
```

Pancake-to-Viper

```
/@ shared w u32 fdata[50343964] @/
/@ shared r u32 rdata[50343960] @/
!st32 50343964, x;
!ld32 x, 50343960;
```

```
store_fdata(gv, 50343964, x); x
:= load_rdata(gv, 50343960);
```

Pancake-to-Viper

```
field enablehost: Bool
field fmt_fifo: Seq[FmtEntry]
field rx_fifo: Seq[Int]
field sym_trace: Seq[Sym]
field rx_frontier: Int
field sym_frontier: Int

method foo(gv: Ref, cmd: FmtEntry) {
  // Impossible from Pancake
  gv.fmt_fifo := gv.fmt_fifo ++ [cmd];
  // Possible from Pancake
  assert |gv.fmt_fifo| >= 1
}
```

Driver Verification

```
method store_fdata(gv: Ref, addr: Int, value: Int)
  requires |gv.fmt_fifo| < 64
  ensures gv.fmt_fifo == old(gv.fmt_fifo) ++ Seq(parse(value))
```

Driver Verification

```
fun 1 write_cmd(1 cmd) {
  /@ requires |gv.fmt_fifo| < 64 @/
  /@ ensures gv.fmt_fifo == old(gv.fmt_fifo) ++ [parse(cmd)] @/

  !st32 FDATA, cmd;

  var i = 0;
  while i < 1000000 {
    i = i + 1;
  }

  return 0;
}
```

Driver Verification

s_1 { ??? }

interference

s_2 {|fmt_fifo| < 64}

FDATA ← cmd

s_3 {fmt_fifo = s_2.fmt_fifo + [cmd]}

interference

s_4 { ??? }

Driver Verification

s_1 { ??? } new precondition

interference

s_2 {|fmt_fifo| < 64}

FDATA ← cmd

s_3 {fmt_fifo = s_2.fmt_fifo + [cmd]}

interference

s_4 { ??? }

Driver Verification

s_1 { ??? } new precondition

interference

s_2 {|fmt_fifo| < 64} old precondition

FDATA ← cmd

s_3 {fmt_fifo = s_2.fmt_fifo + [cmd]} – old postcondition

interference

s_4 { ??? }

Driver Verification

s_1 { ??? } ⟵ new precondition

interference

s_2 {|fmt_fifo| < 64} ⟵ old precondition

FDATA ← cmd

s_3 {fmt_fifo = s_2.fmt_fifo + [cmd]} – old postcondition

interference

s_4 { ??? } ⟵ new postcondition

Driver Verification

s_1 { ??? } new precondition

interference ensures $R(s_1, s_2)$ (driver's R, device's G)

s_2 {|fmt_fifo| < 64} old precondition

FDATA ← cmd

s_3 {fmt_fifo = s_2.fmt_fifo + [cmd]} – old postcondition

interference ensures $R(s_3, s_4)$

s_4 { ??? } new postcondition

Driver Verification

s_1 { ??? }

$interference$ − new precondition $\wedge$ $R(s_1, s_2) \implies$ old precondition

s_2 {|fmt_fifo| < 64}

FDATA ← cmd

s_3 {fmt_fifo = s_2.fmt_fifo + [cmd]}

$interference$

s_4 { ??? }

Driver Verification

s_1 { ??? }

$interference$ – new precondition $\wedge\ R(s_1, s_2) \implies$ old precondition

s_2 {|fmt_fifo| < 64}

FDATA ← cmd

s_3 {fmt_fifo = s_2.fmt_fifo + [cmd]}

$interference$ – old postcondition $\wedge\ R(s_1, s_2) \wedge R(s_3, s_4) \implies$ new postcondition

s_4 { ??? }

Driver Verification

s_1 $\{|\text{fmt\_fifo}| < 64\}$

$interference - \text{new precondition} \wedge R(s_1, s_2) \implies \text{old precondition}$

s_2 $\{|\text{fmt\_fifo}| < 64\}$

$\texttt{FDATA} \leftarrow \text{cmd}$

s_3 $\{\text{fmt\_fifo} = s_2.\text{fmt\_fifo} + [\text{cmd}]\}$

$interference - \text{old postcondition} \wedge R(s_1, s_2) \wedge R(s_3, s_4) \implies \text{new postcondition}$

s_4 $\{ \quad ??? \quad \}$

Driver Verification

s_1 $\{|\text{fmt\_fifo}| < 64\}$

$interference - \text{new precondition} \wedge R(s_1, s_2) \implies \text{old precondition}$

s_2 $\{|\text{fmt\_fifo}| < 64\}$

$\texttt{FDATA} \leftarrow \text{cmd}$

s_3 $\{\text{fmt\_fifo} = s_2.\text{fmt\_fifo} + [\text{cmd}]\}$

$interference - \text{old postcondition} \wedge R(s_1, s_2) \wedge R(s_3, s_4) \implies \text{new postcondition}$

s_4 $\{\text{sym\_rel}(s_1, s_4, s_1.\text{fmt\_fifo} + [\text{cmd}], s_4.\text{fmt\_fifo})\}$

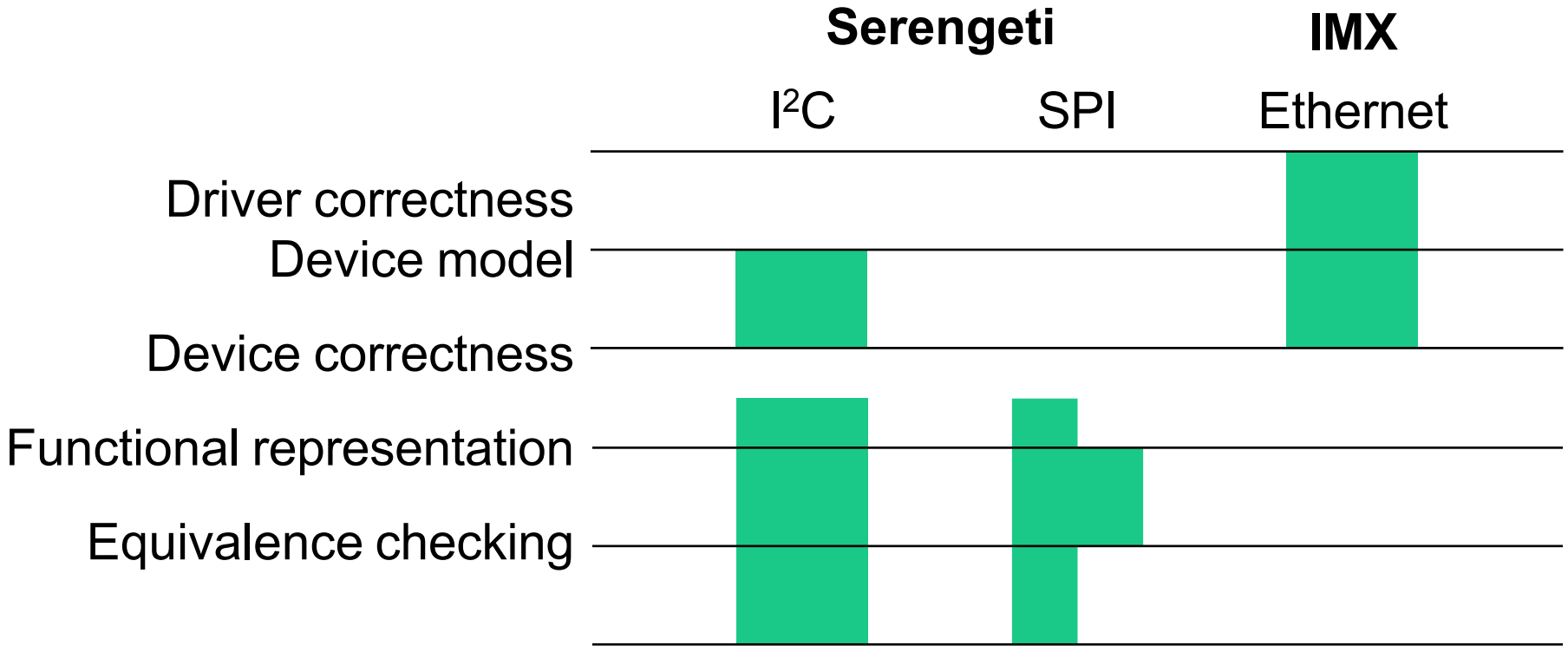
Driver Verification

s_1 $\{|\text{fmt\_fifo}| < 64\}$

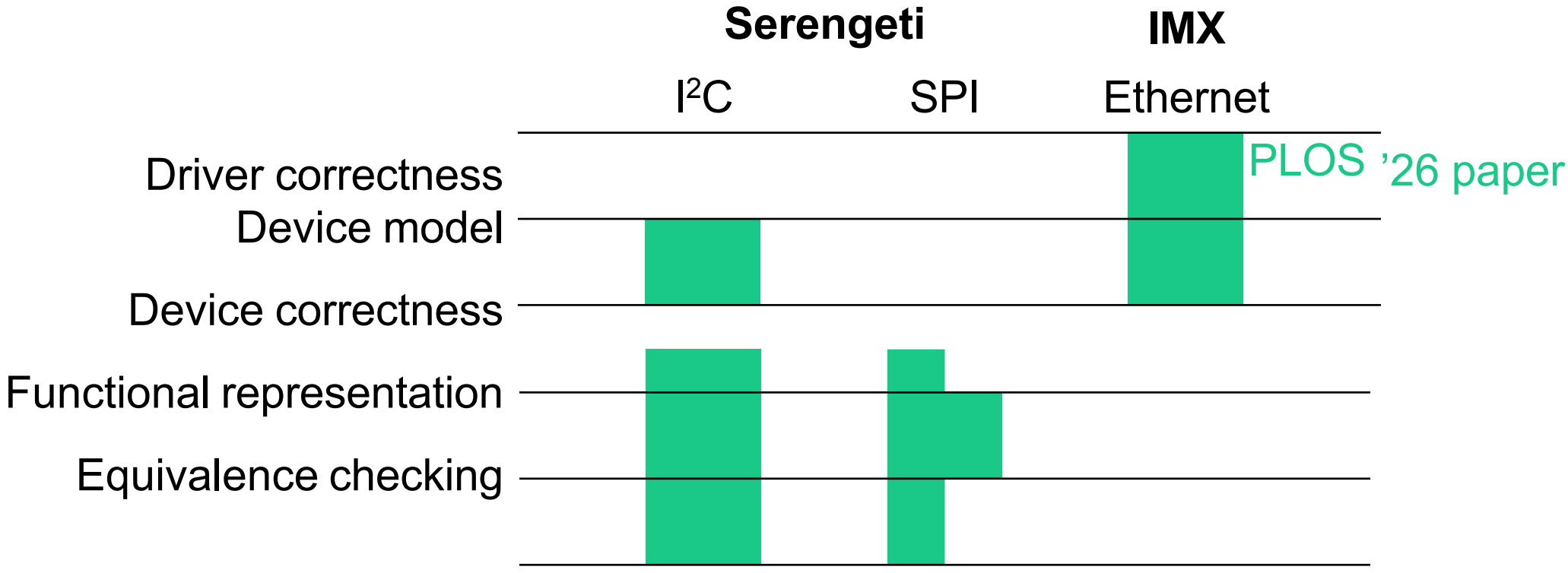
$\texttt{FDATA} \leftarrow \text{cmd}$

s_2 $\{\text{sym\_rel}(s_1, s_2, s_1.\text{fmt\_fifo} + [\text{cmd}], s_2.\text{fmt\_fifo})\}$

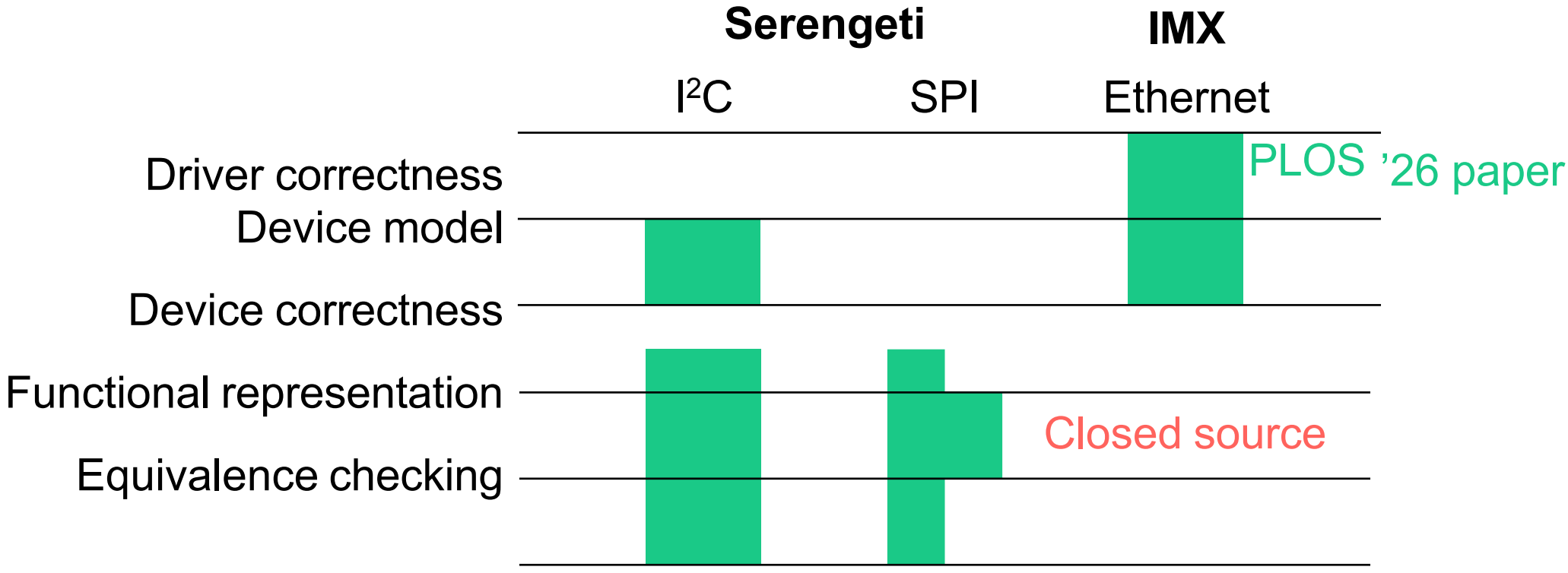
Status


Serengeti
IMX
I²C
SPI
Ethernet
Driver correctness
Device model
Device correctness
Functional representation
Equivalence checking

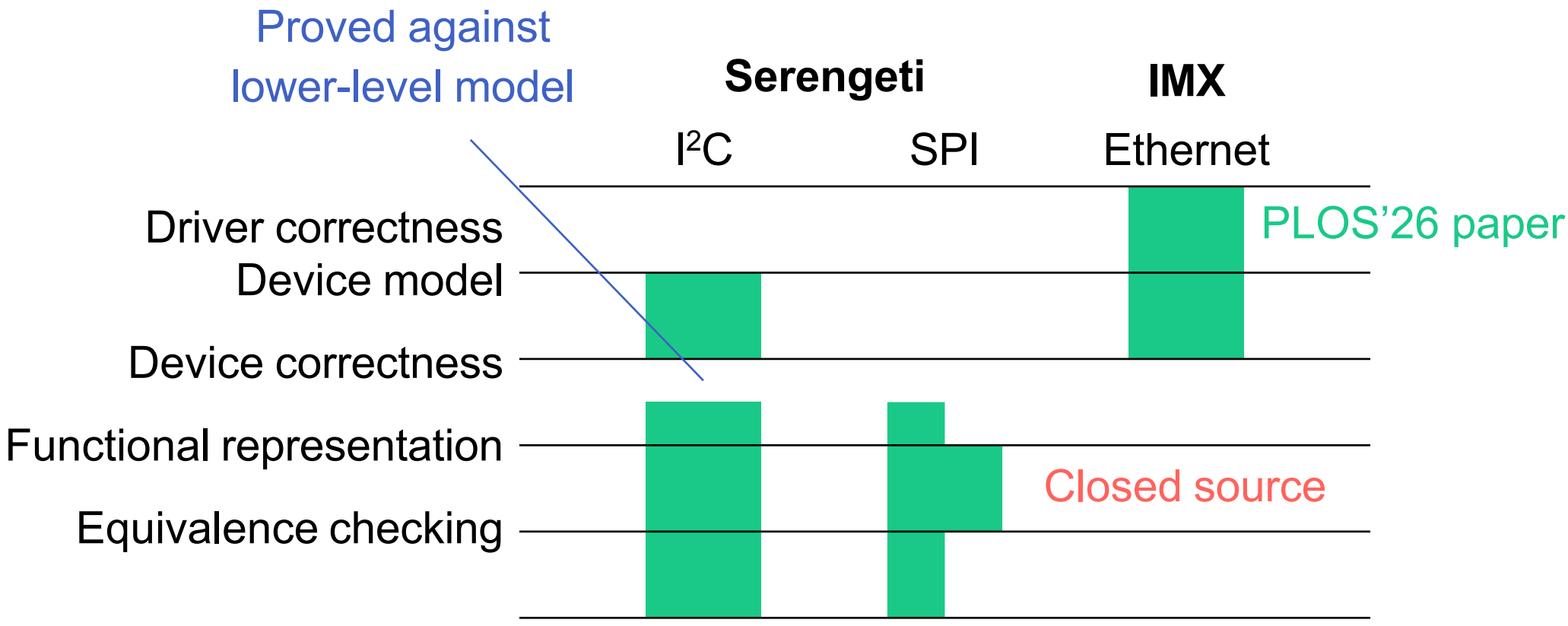
Status


Serengeti
IMX
I²C
SPI
Ethernet
Driver correctness
Device model
Device correctness
Functional representation
Equivalence checking
PLOS ’26 paper

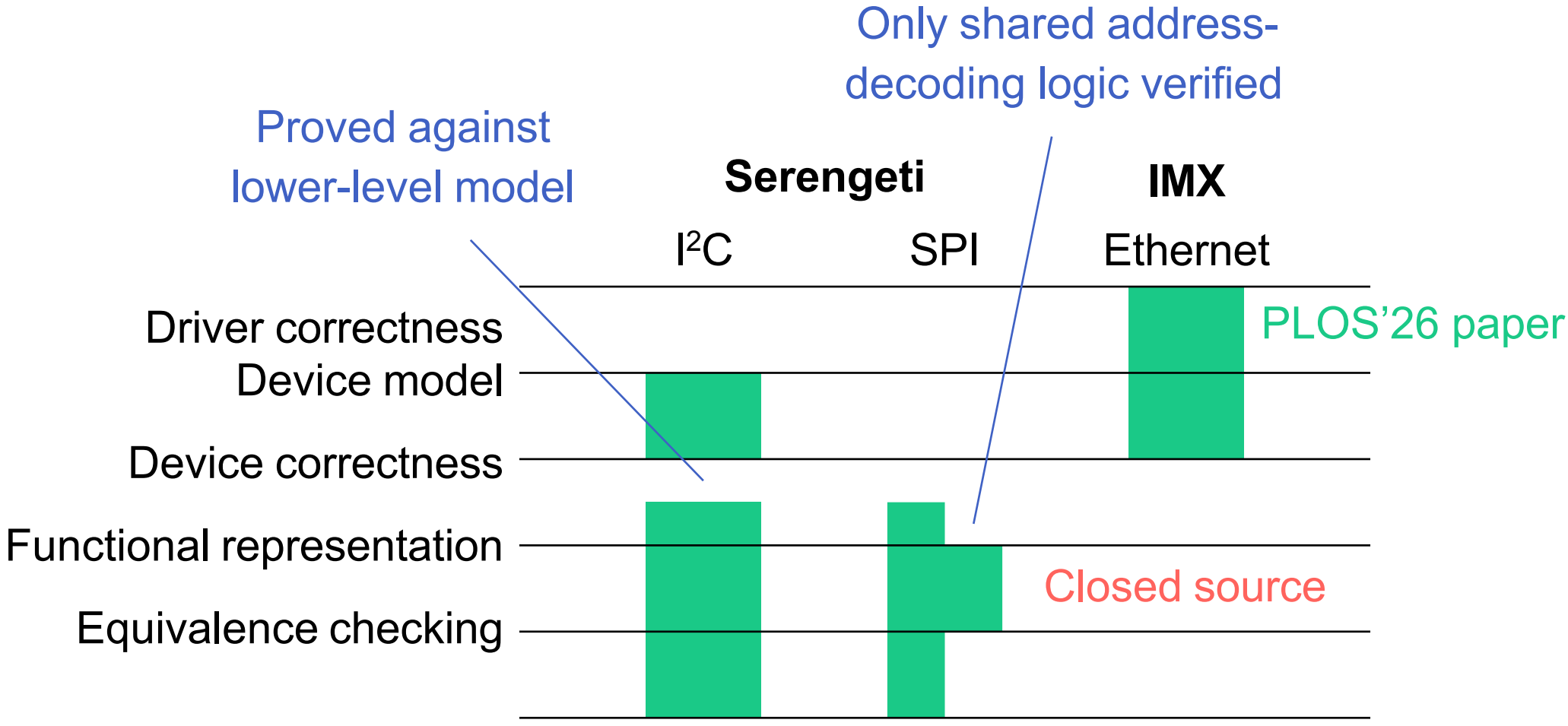
Status


Serengeti
IMX
I²C
SPI
Ethernet
Driver correctness
Device model
Device correctness
Functional representation
Equivalence checking
PLOS ’26 paper
Closed source

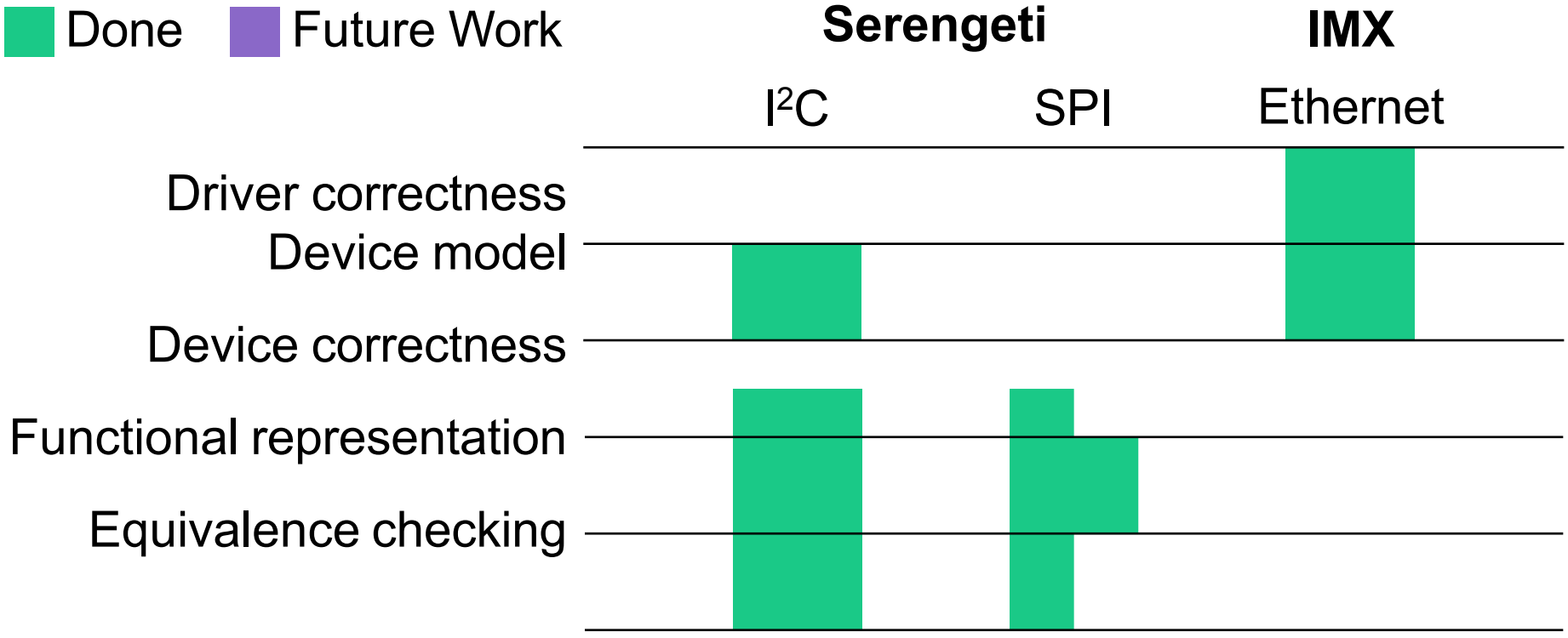
Status


Proved against
lower-level model
Serengeti
IMX
I²C
SPI
Ethernet
Driver correctness
Device model
Device correctness
Functional representation
Equivalence checking
PLOS'26 paper
Closed source

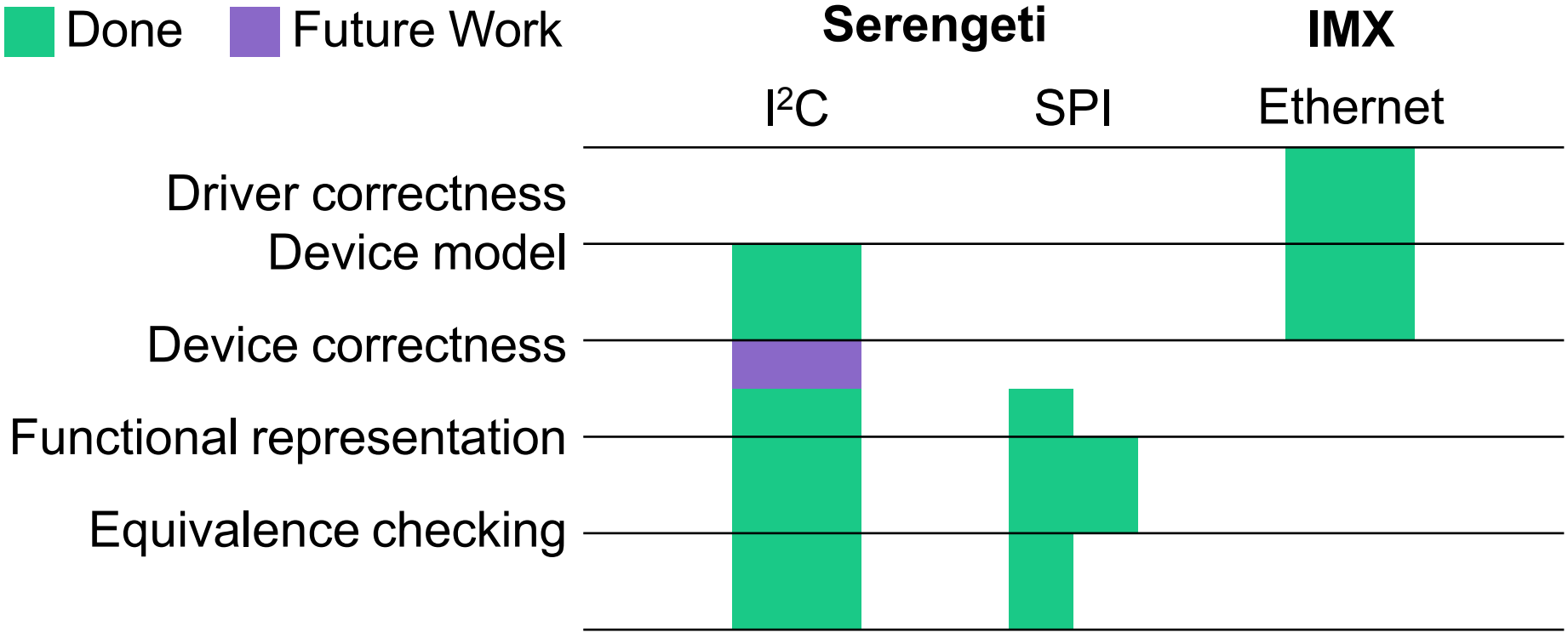
Status


Only shared address-decoding logic verified
Proved against lower-level model
Serengeti
IMX
I²C
SPI
Ethernet
Driver correctness
Device model
Device correctness
Functional representation
Equivalence checking
PLOS'26 paper
Closed source

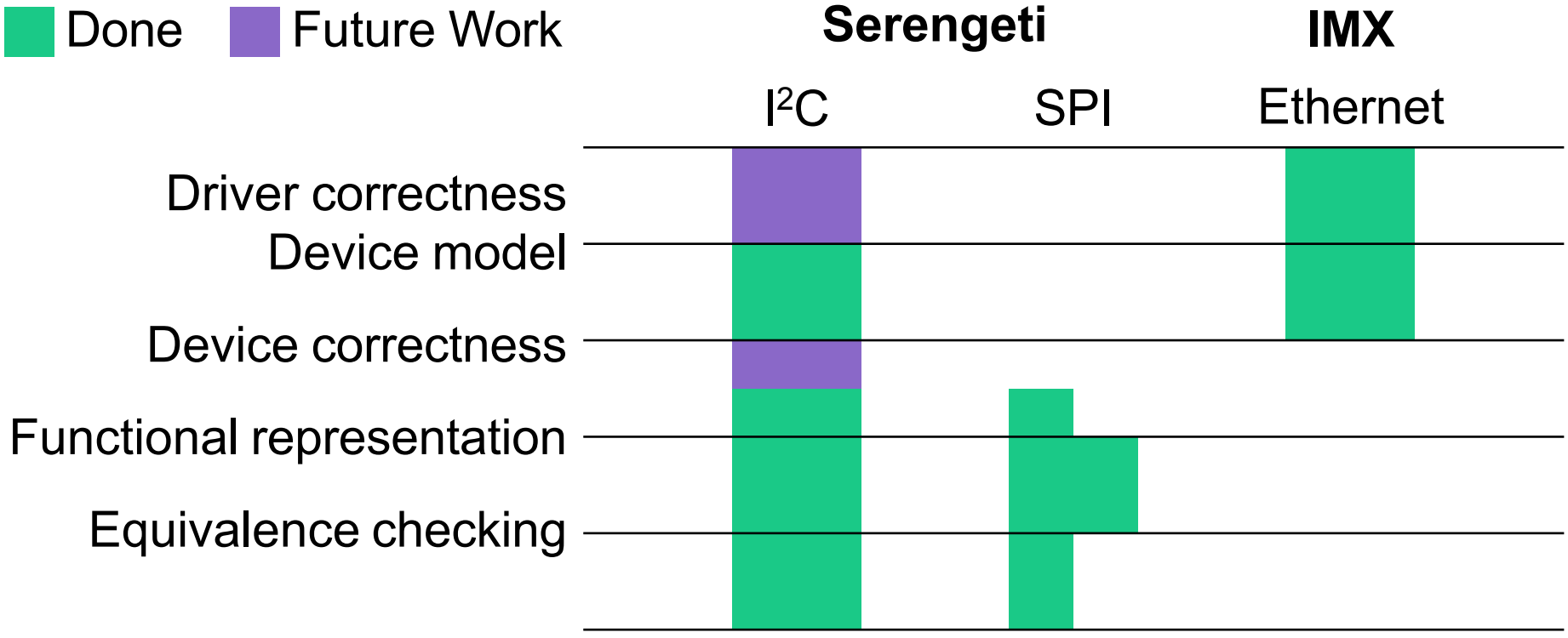
Future Work


Done
Future Work
Serengeti
IMX
I²C
SPI
Ethernet
Driver correctness
Device model
Device correctness
Functional representation
Equivalence checking

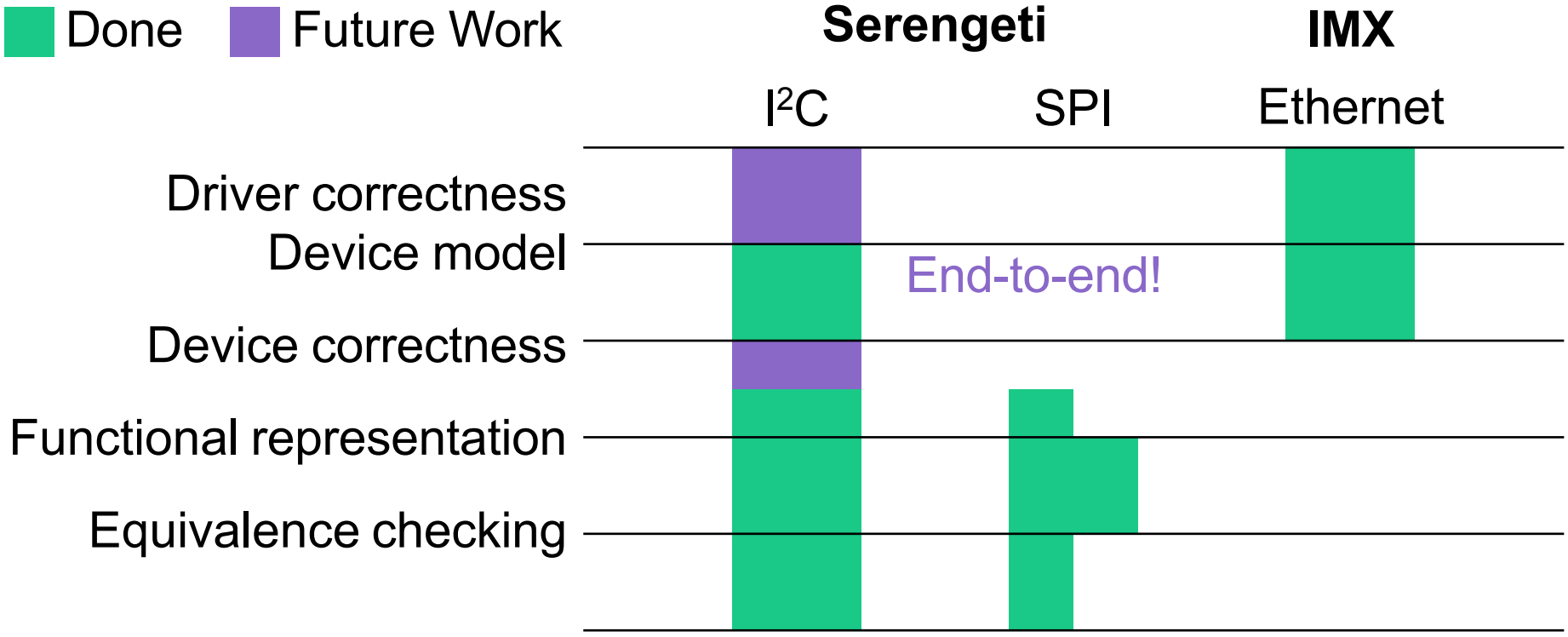
Future Work


Done
Future Work
Serengeti
IMX
I²C
SPI
Ethernet
Driver correctness
Device model
Device correctness
Functional representation
Equivalence checking

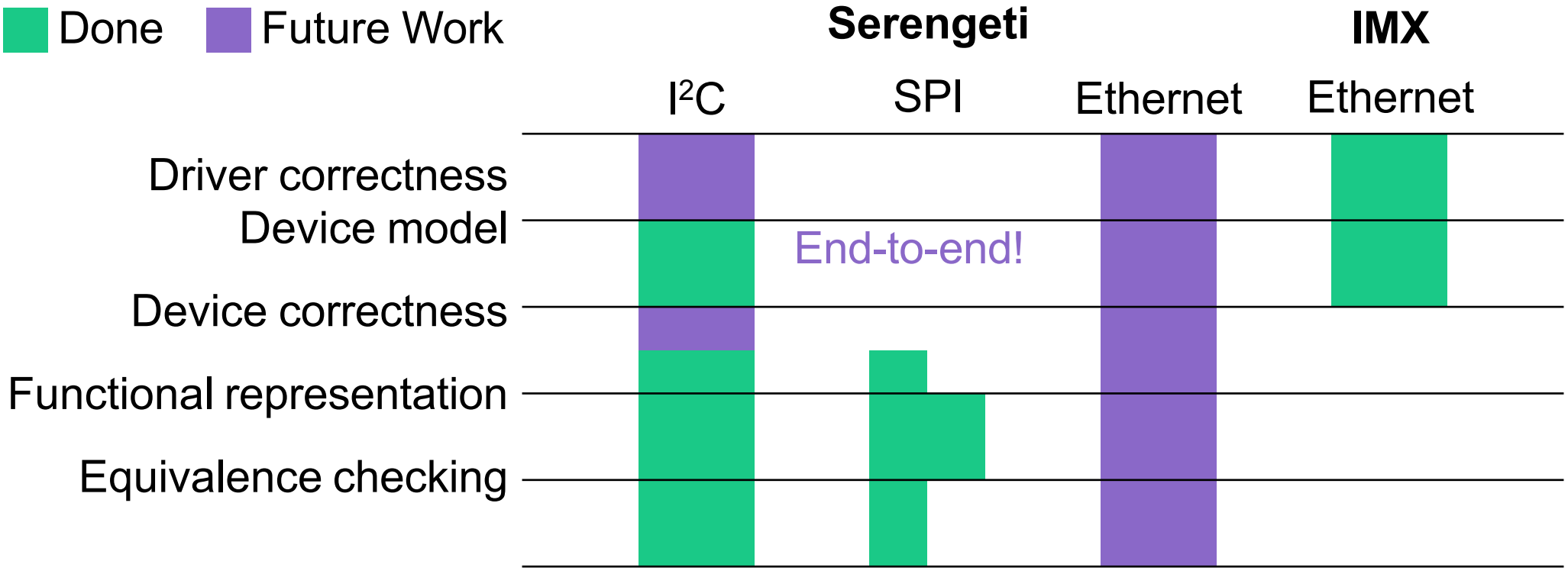
Future Work


Done
Future Work
Serengeti
IMX
I²C
SPI
Ethernet
Driver correctness
Device model
Device correctness
Functional representation
Equivalence checking

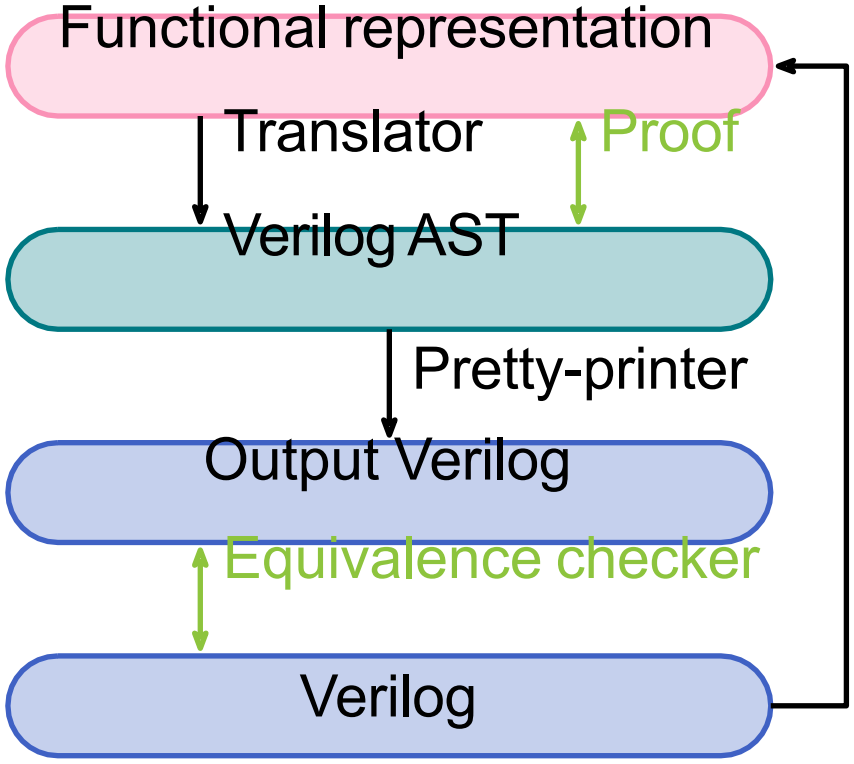
Future Work

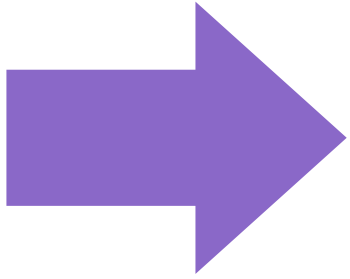

Done
Future Work
Serengeti
IMX
I²C
SPI
Ethernet
Driver correctness
Device model
Device correctness
Functional representation
Equivalence checking
End-to-end!

Future Work


Done
Future Work
Serengeti
IMX
I²C
SPI
Ethernet
Ethernet
Driver correctness
Device model
Device correctness
Functional representation
Equivalence checking
End-to-end!

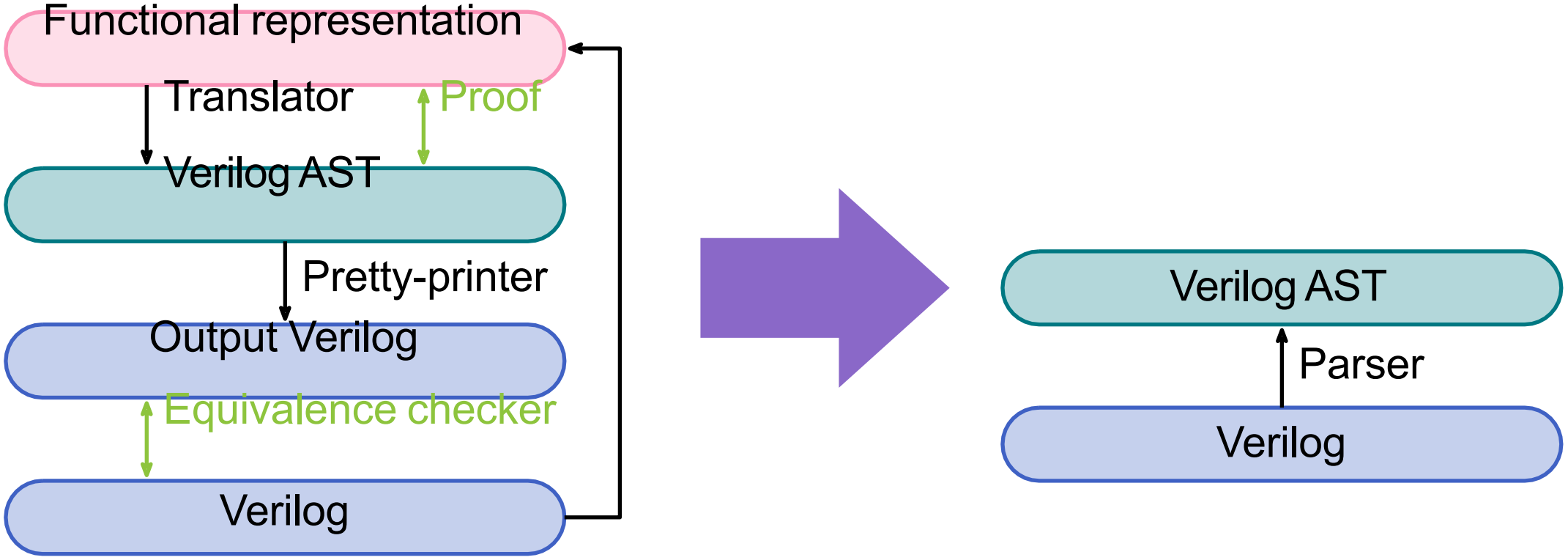
Future Work


Functional representation
Translator
Proof
Verilog AST
Pretty-printer
Output Verilog
Equivalence checker
Verilog

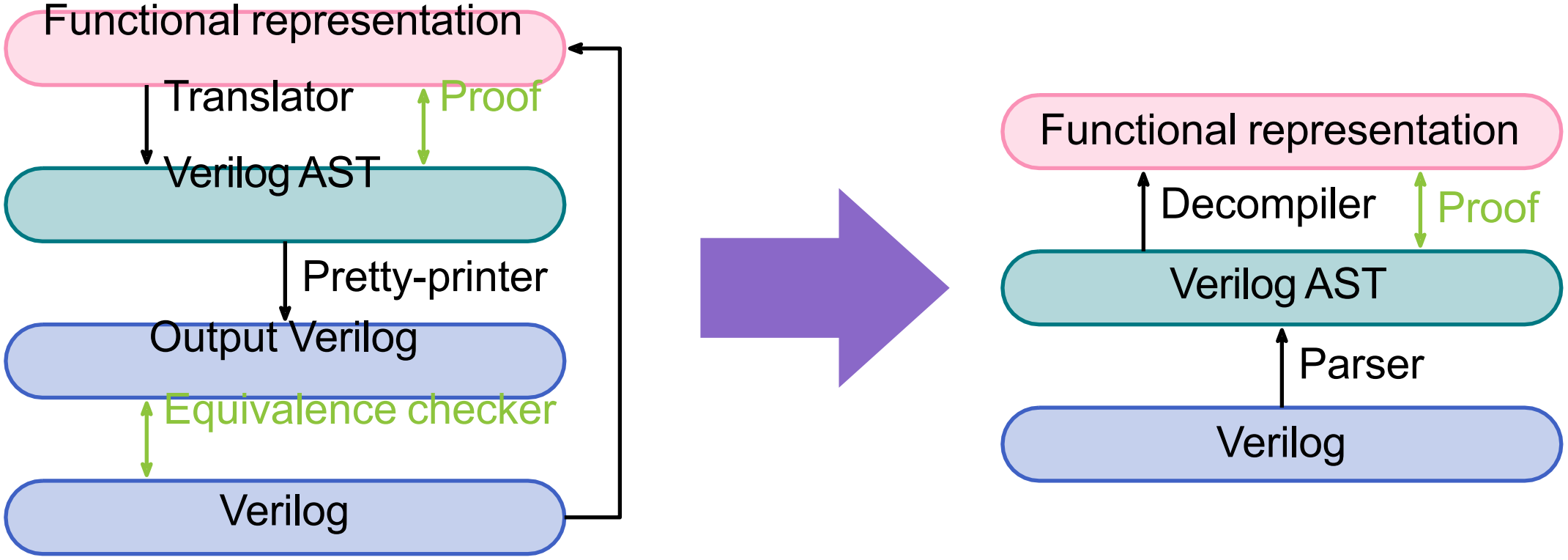



Verilog

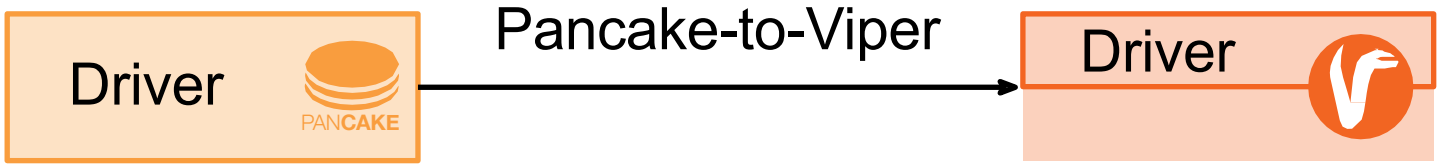
Future Work


Functional representation
Translator
Proof
Verilog AST
Pretty-printer
Output Verilog
Equivalence checker
Verilog
Verilog AST
Parser
Verilog

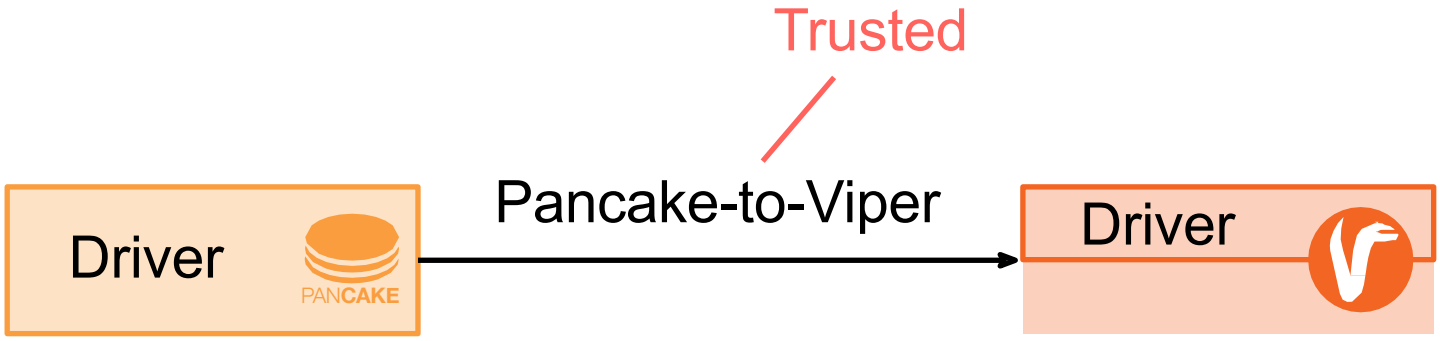
Future Work


Functional representation
Translator
Proof
Verilog AST
Pretty-printer
Output Verilog
Equivalence checker
Verilog
Functional representation
Decompiler
Proof
Verilog AST
Parser
Verilog

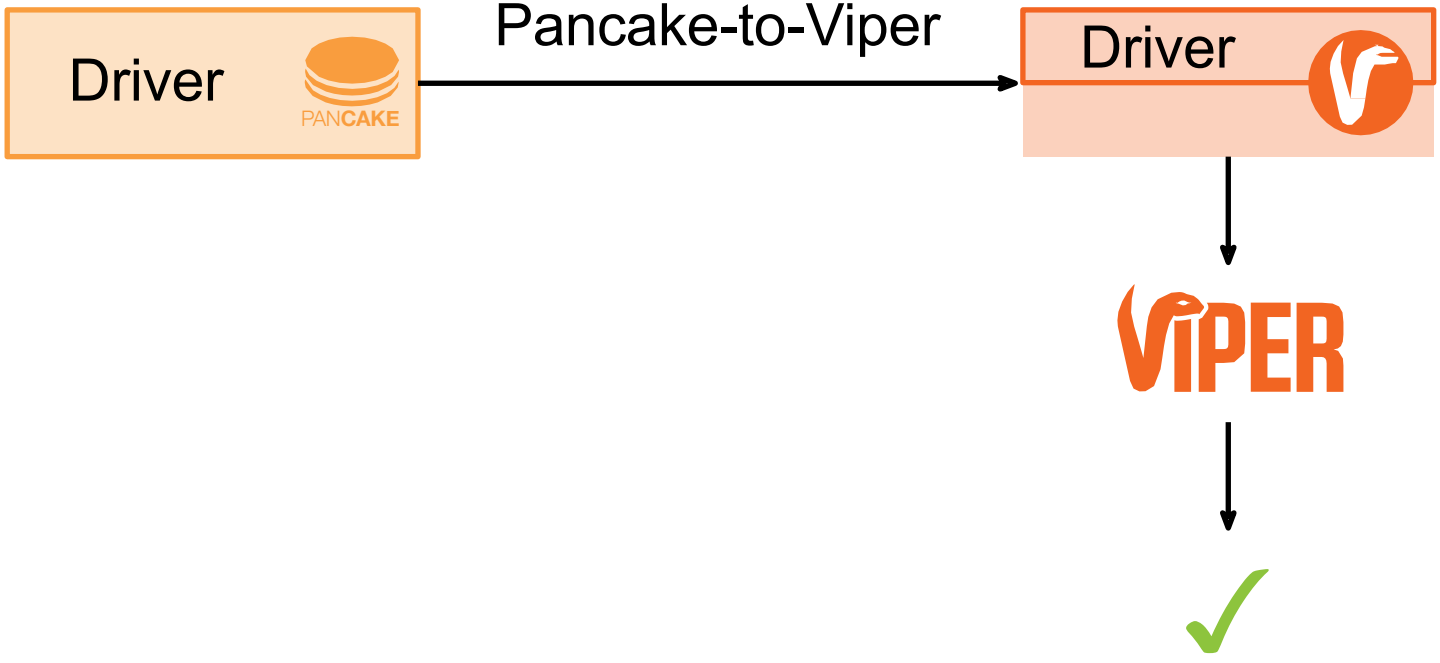
Future Work



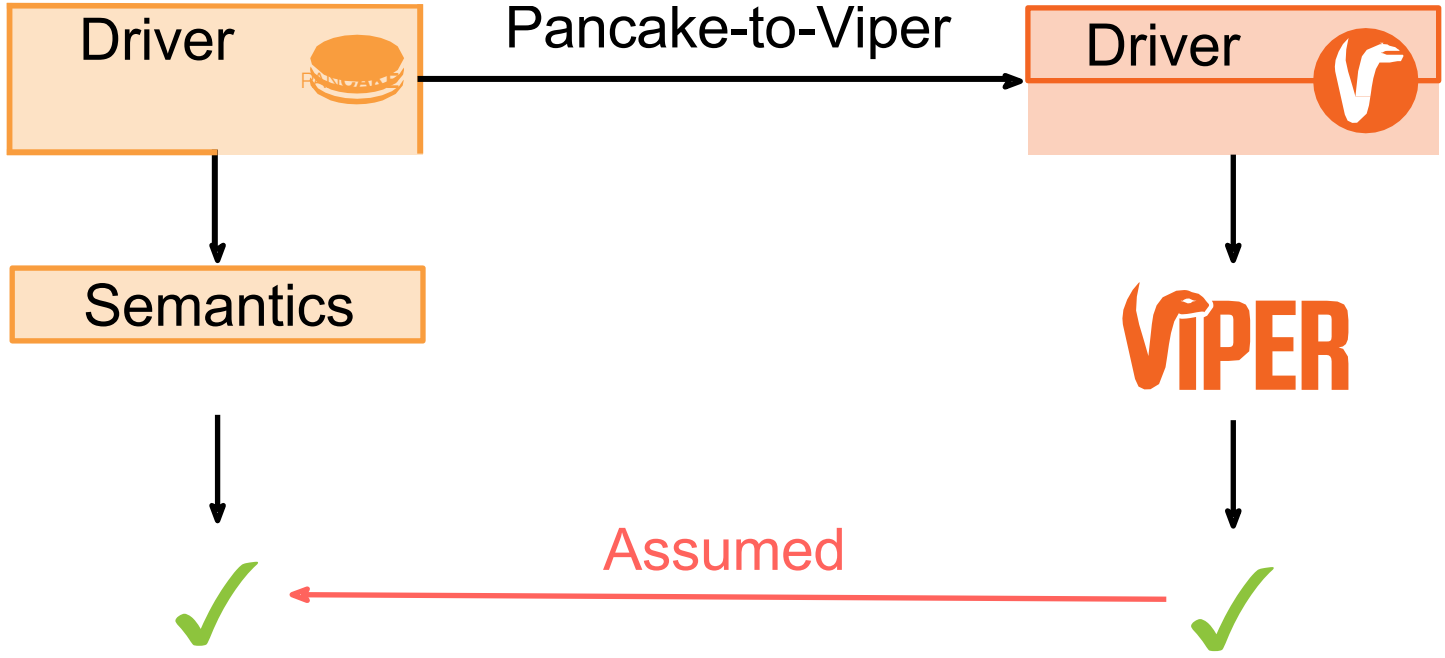
Future Work



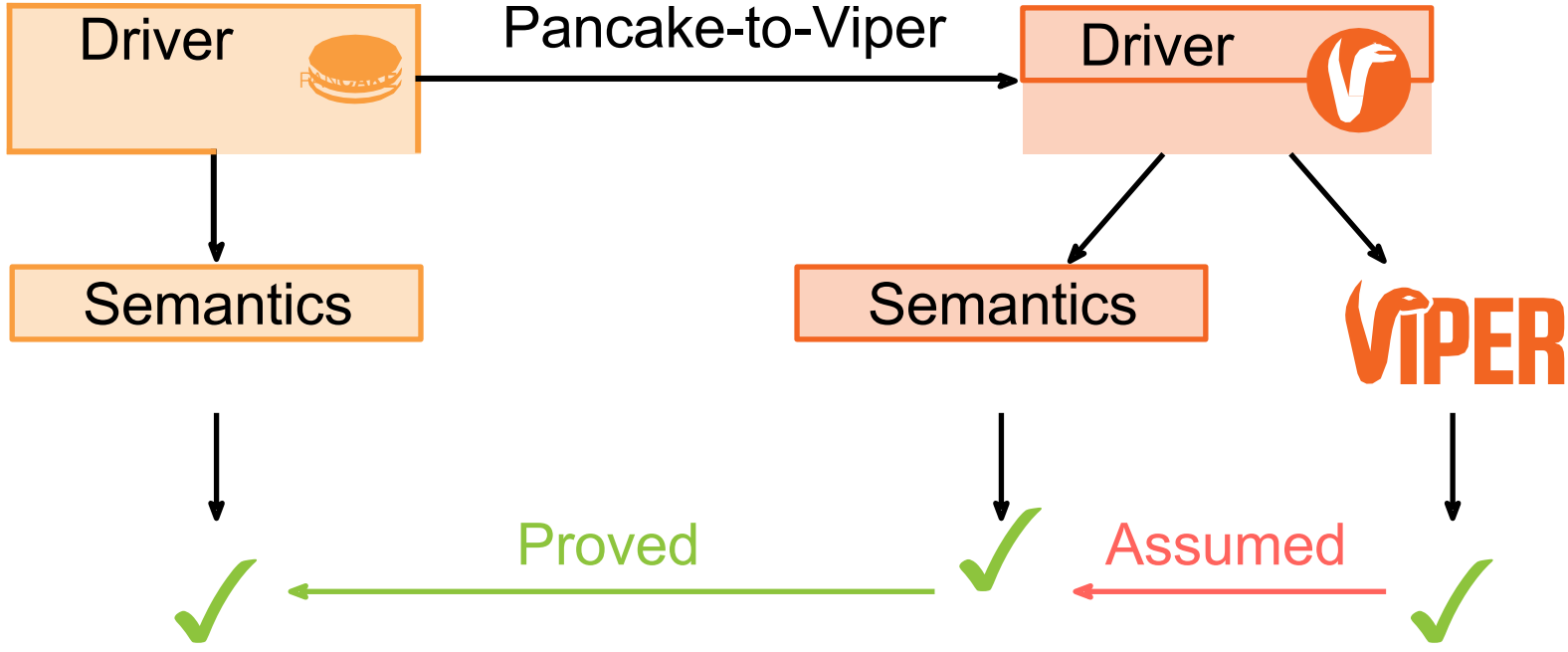
Future Work



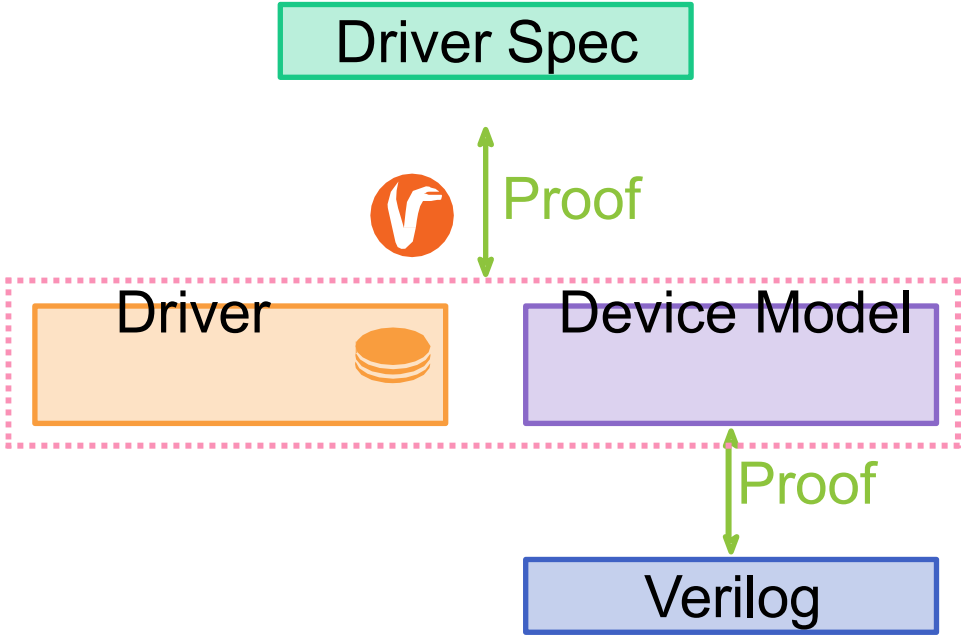
Future Work



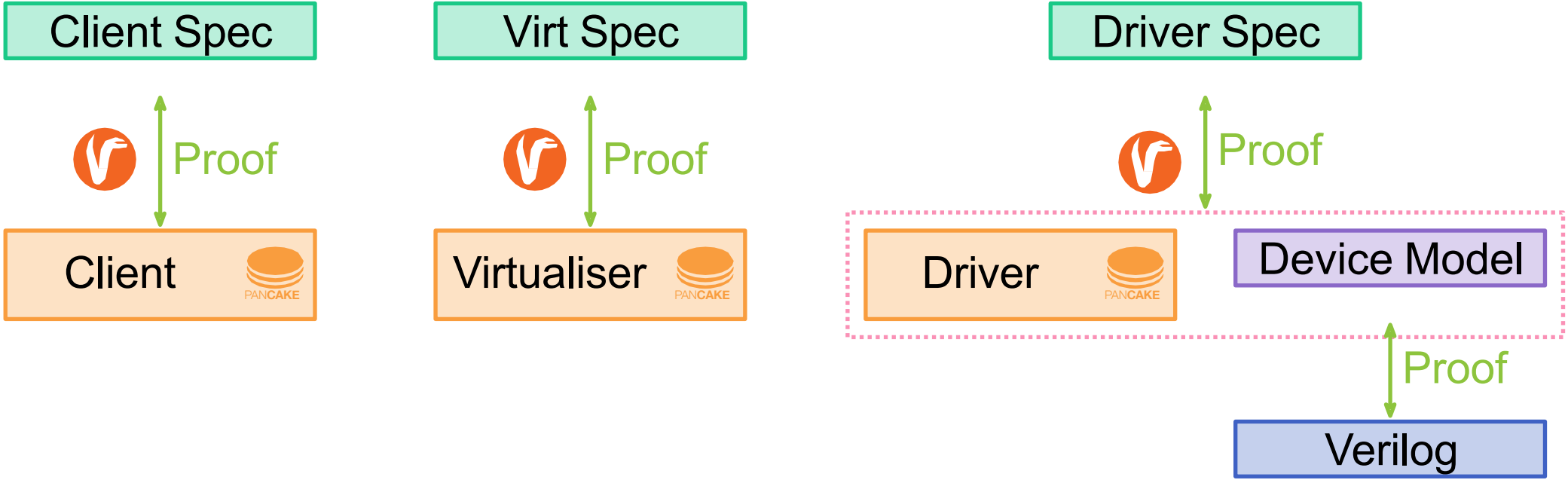
Future Work


Driver
Pancake-to-Viper
Driver
Semantics
Semantics
VIPER
Proved
Assumed

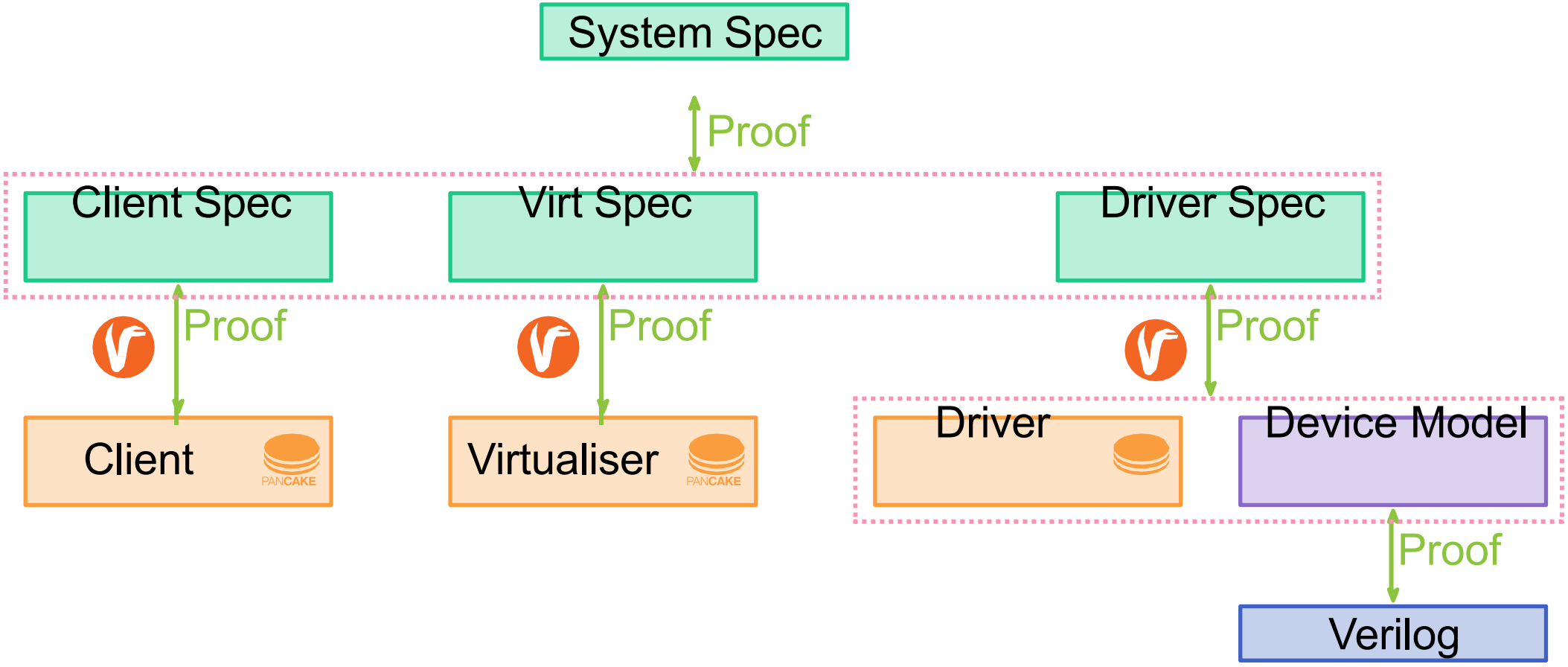
Future Work



Future Work


Client Spec
Proof
Client
PANCAKE
Virt Spec
Proof
Virtualiser
PANCAKE
Driver Spec
Proof
Driver
PANCAKE
Device Model
Proof
Verilog

Future Work


System Spec
Proof
Client Spec
Virt Spec
Driver Spec
Proof
Proof
Proof
Client
PANCAKE
Virtualiser
PANCAKE
Driver
Device Model
Proof
Verilog

Questions?