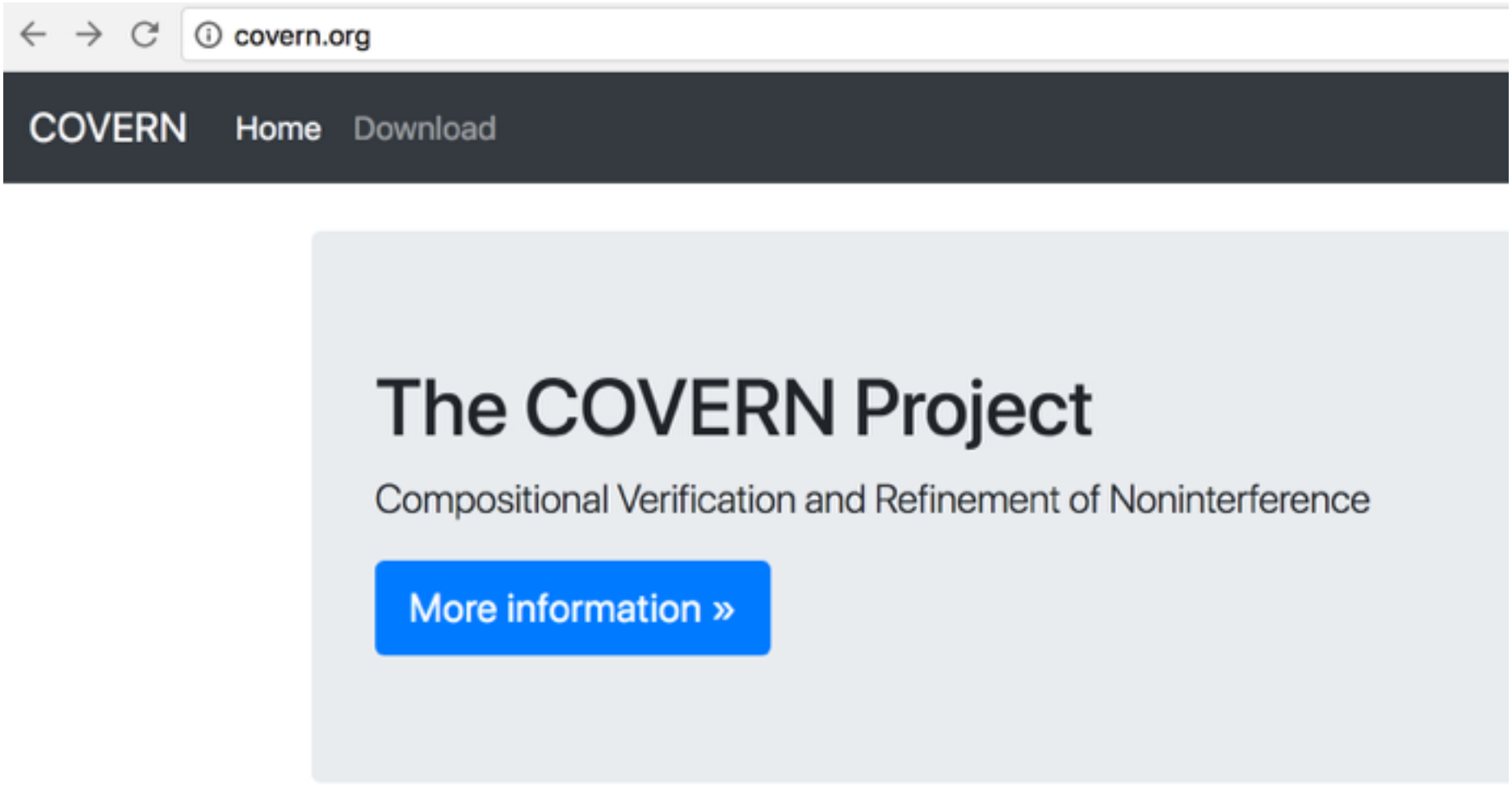
Verified Compilation of Noninterference for Shared-Memory Concurrent Programs

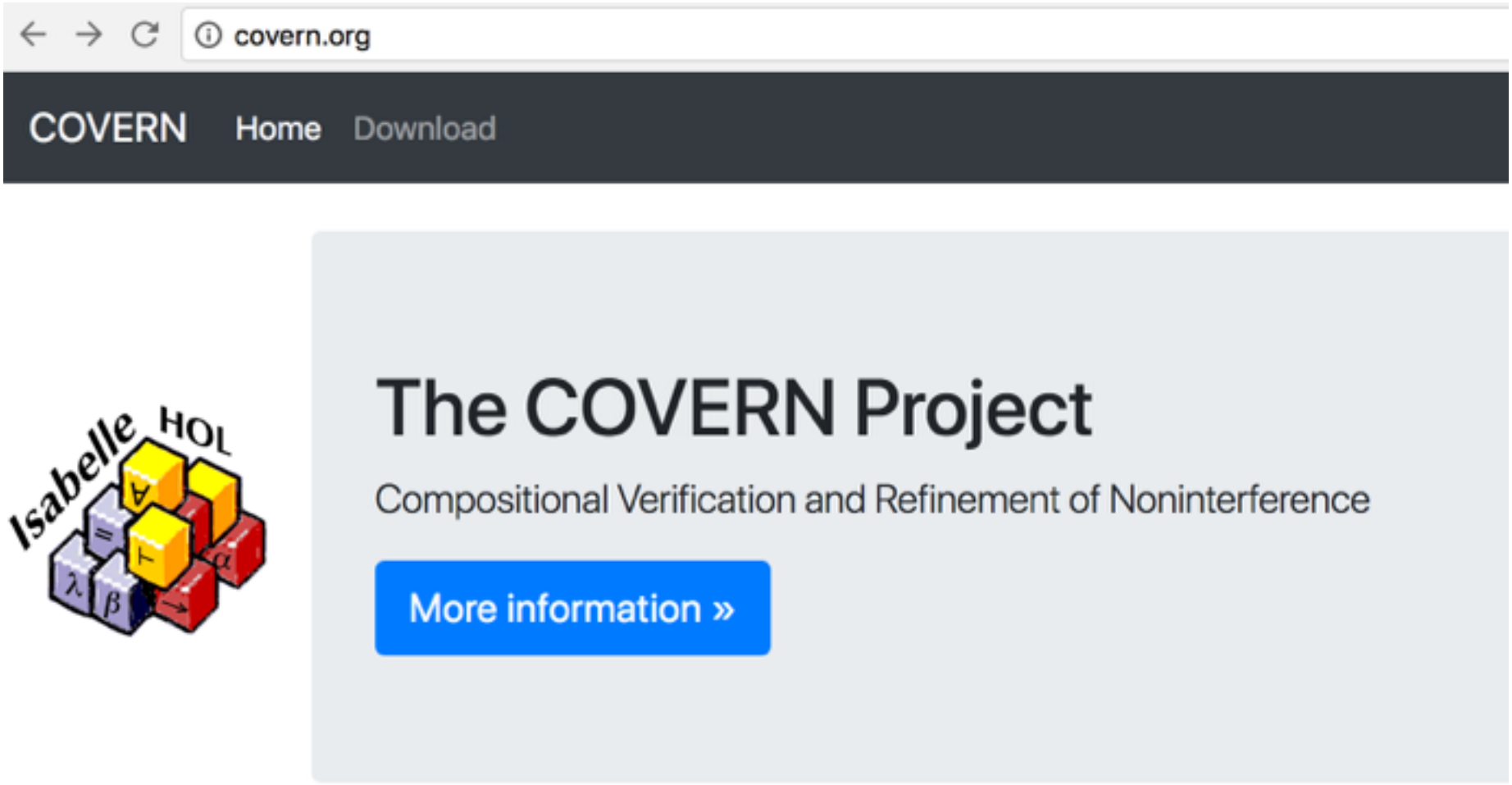
Toby Murray

University of Melbourne &
Data61 (formerly NICTA), CSIRO

http://covern.org



http://covern.org



Confidentiality (Information Flow) Proofs

Confidentiality (Information Flow) Proofs

Confidentiality (Information Flow) Proofs

CoCon

Confidentiality (Information Flow) Proofs

CoCon

CoSMed

Confidentiality (Information Flow) Proofs

CoCon

CoSMed

SAFE

Confidentiality (Information Flow) Proofs

CoCon

CoSMed

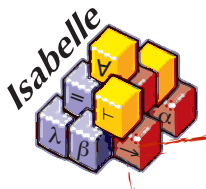
SAFE

Quark

Confidentiality (Information Flow) Proofs

CoCon

CoSMed



SEQUENTIAL

SAFE

Quark

The Cross Domain Desktop Compositor

Australian Government

Department of Defence

Defence Science and Technology Group

Mark Beaumont

CDDC

The Cross Domain Desktop Compositor

Australian Government
Department of Defence
Defence Science and Technology Group

DATA 61

CSIRO

Mark Beaumont

CDC

The Cross Domain Desktop Composit

Toby Murray

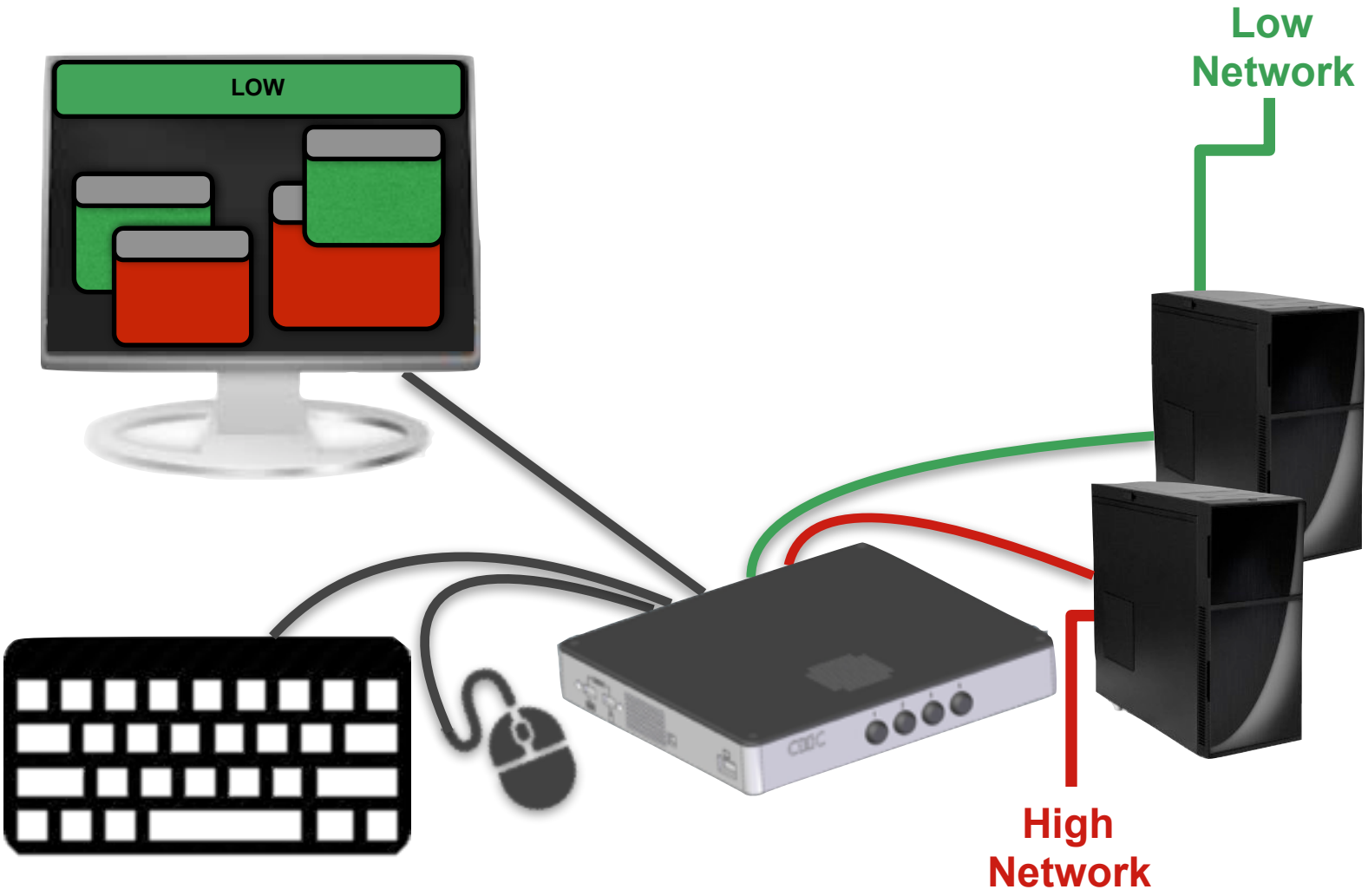
Kevin Elphinstone

Australian Government
Department of Defence
Defence Science and Technology Group

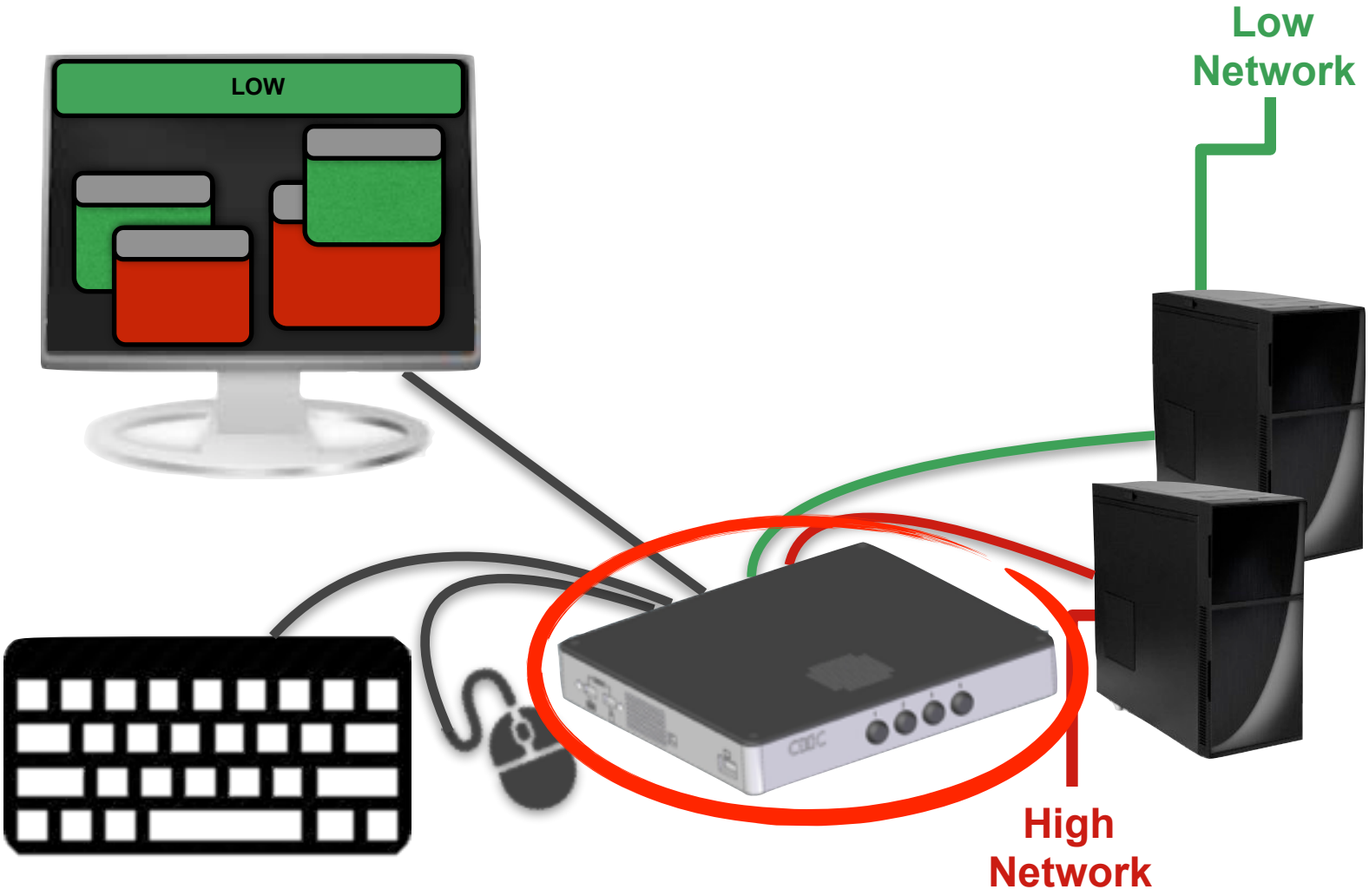
DATA 61

CSIRO

Cross Domain Desktop Compositor

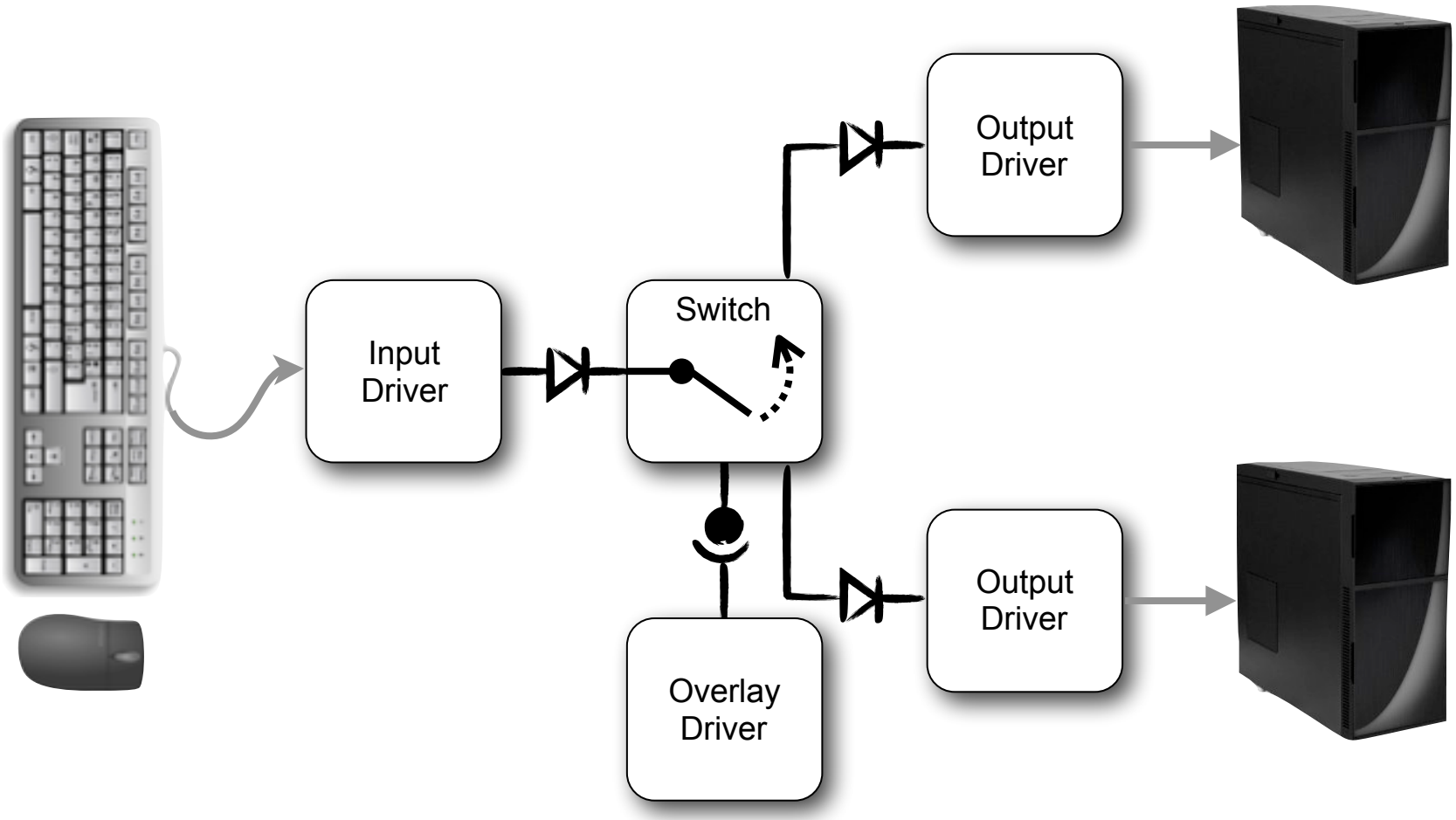


Cross Domain Desktop Compositor



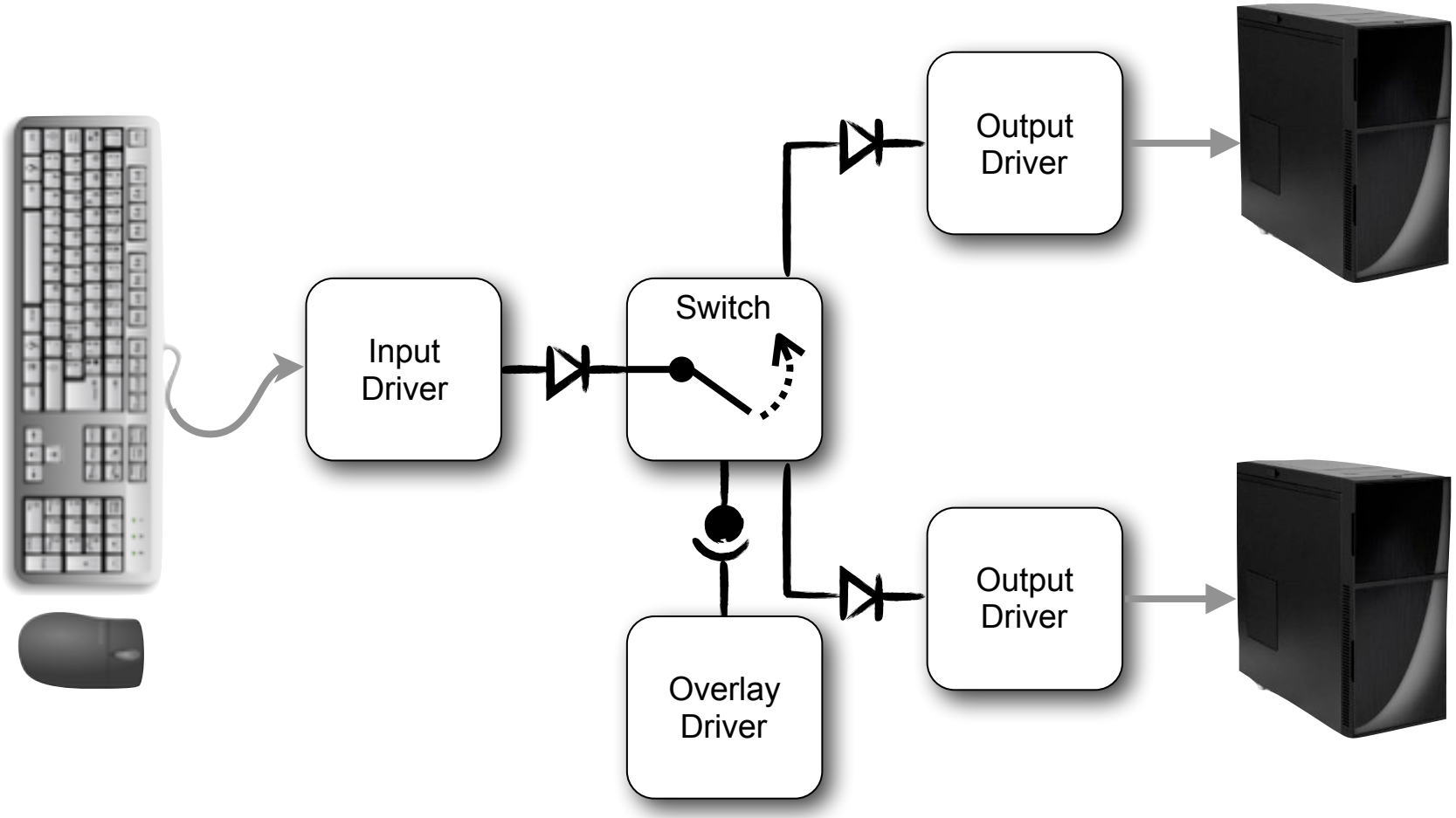
CDDC

seL4-Based Software Architecture



seL4-Based Software Architecture

(ignoring device administration and configuration, plus keyboard LED control)



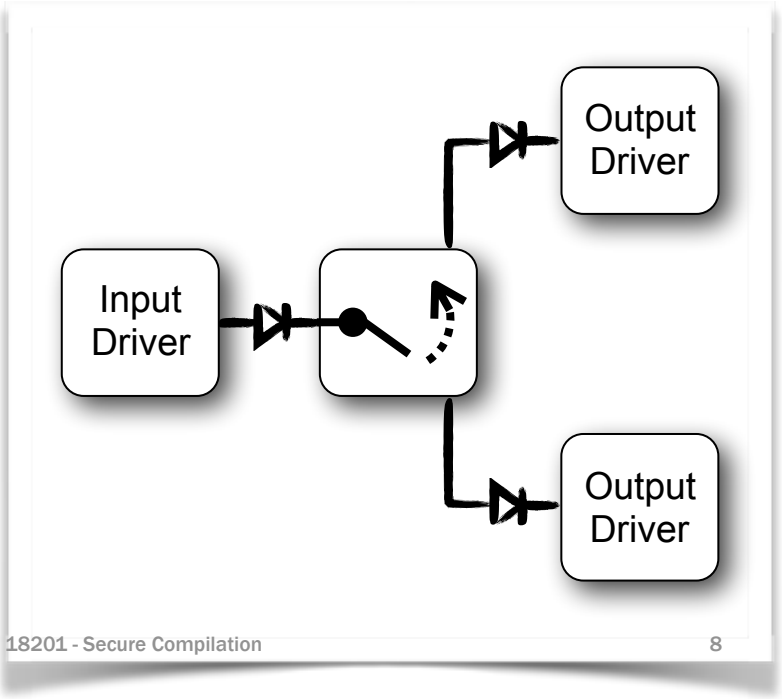
Verification Framework: Requirements

Verification Framework: Requirements

Compositional Security Property

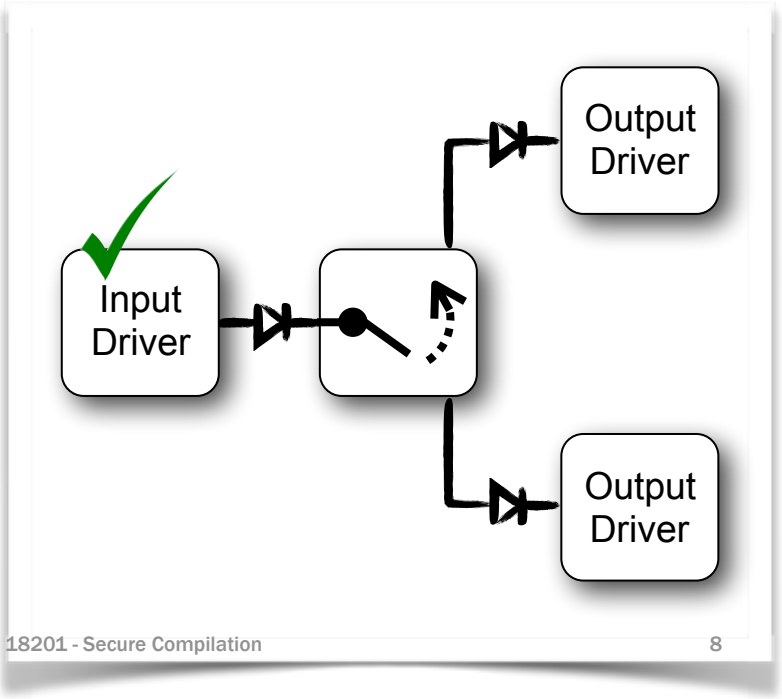
Verification Framework: Requirements

Compositional Security Property



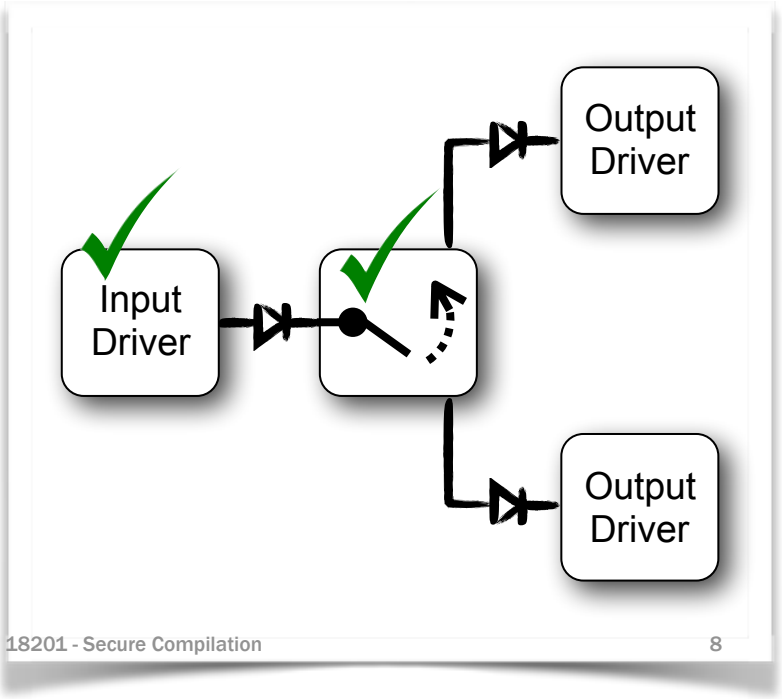
Verification Framework: Requirements

Compositional Security Property



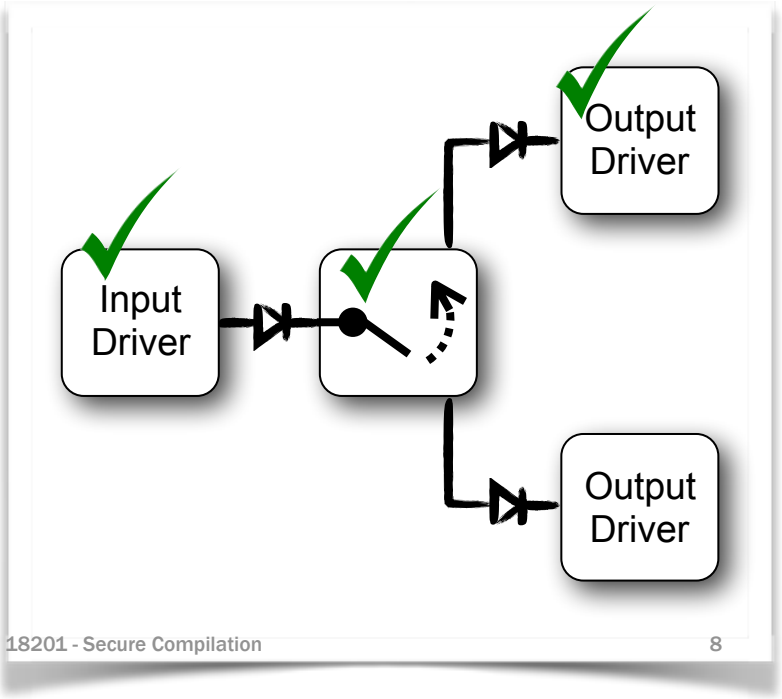
Verification Framework: Requirements

Compositional Security Property



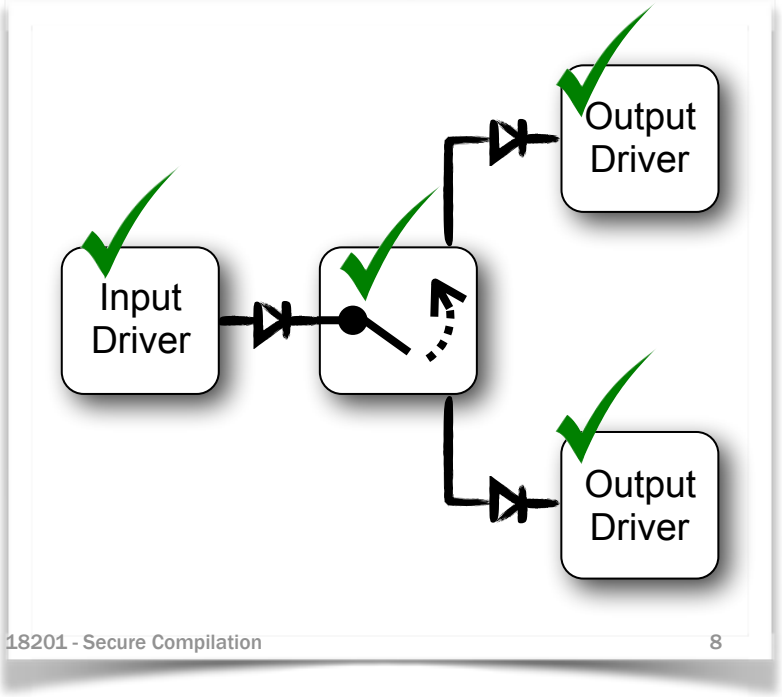
Verification Framework: Requirements

Compositional Security Property



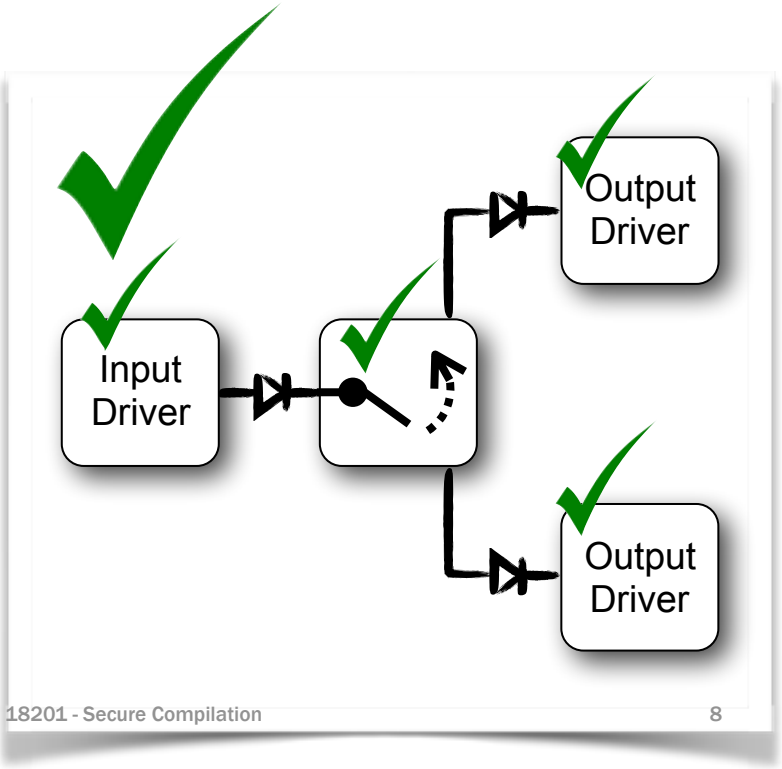
Verification Framework: Requirements

Compositional Security Property



Verification Framework: Requirements

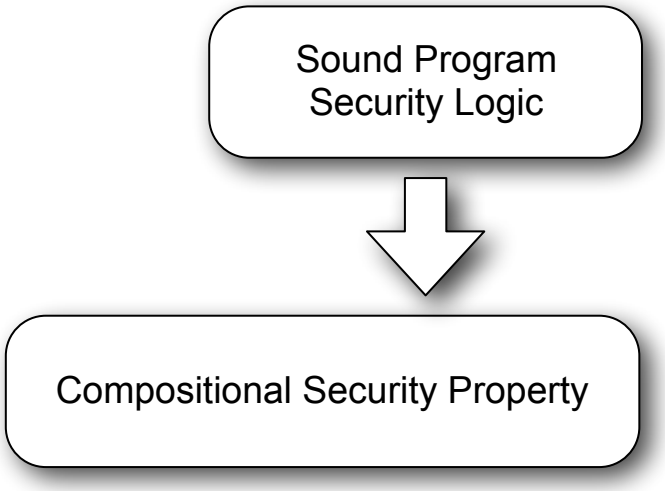
Compositional Security Property



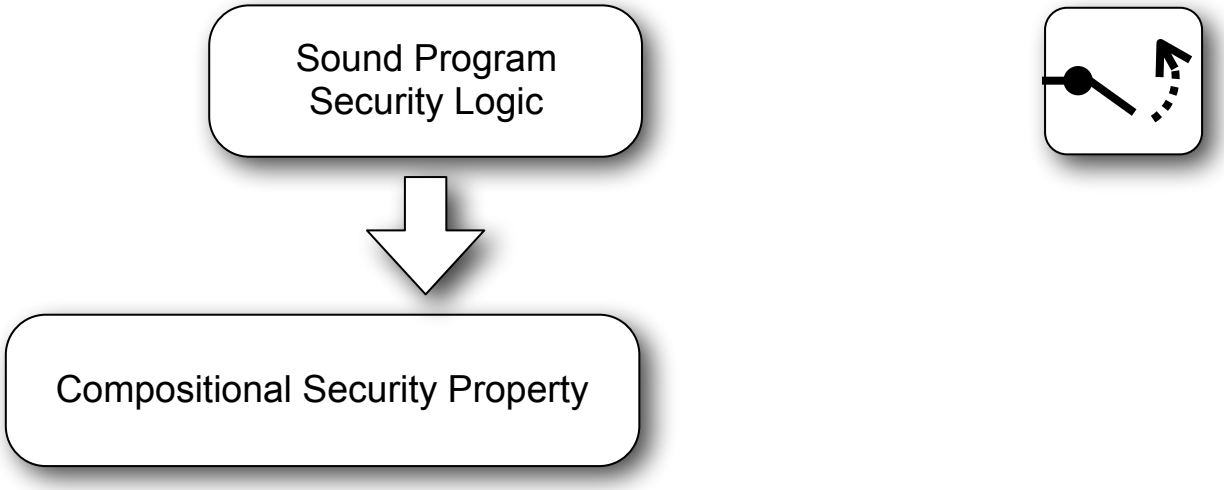
Verification Framework: Requirements

Compositional Security Property

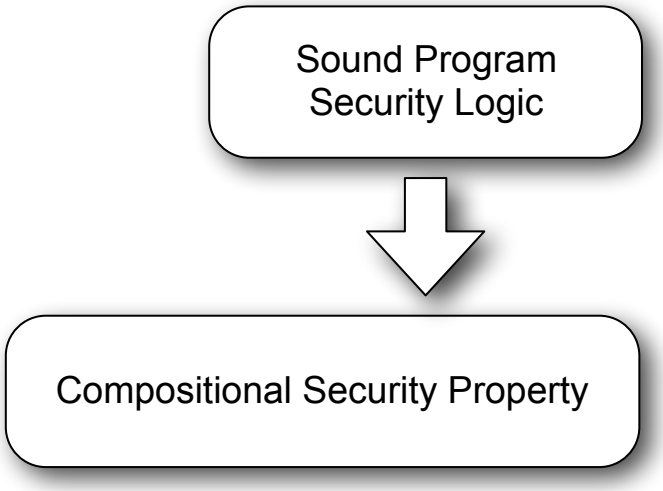
Verification Framework: Requirements



Verification Framework: Requirements

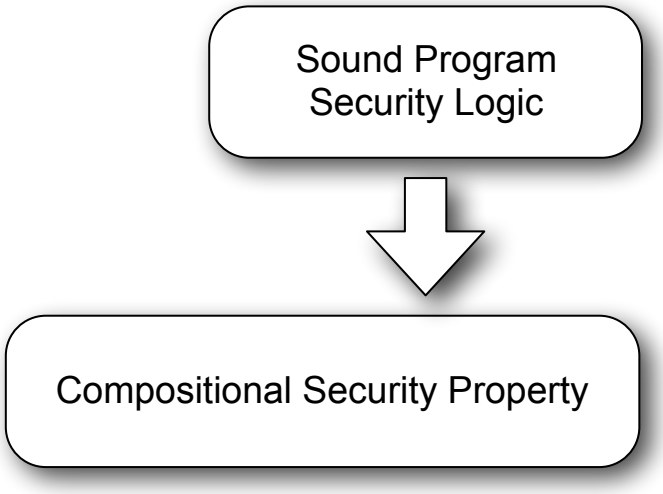


Verification Framework: Requirements



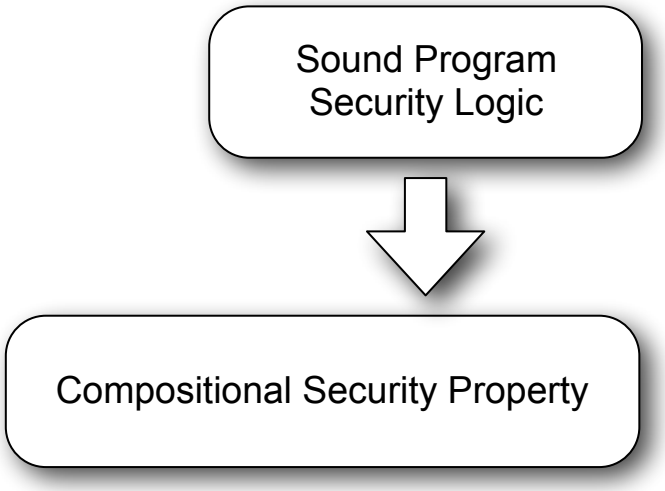

```
skip   /* control +=_m AsmNoW */;
skip   /* temp +=_m AsmNoRW */;
temp := buffer;
if control == 0 then
  low-var := temp
else
  high-var := temp
endif;
temp := 0;
skip   /* temp -=_m AsmNoRW */
```

Verification Framework: Requirements

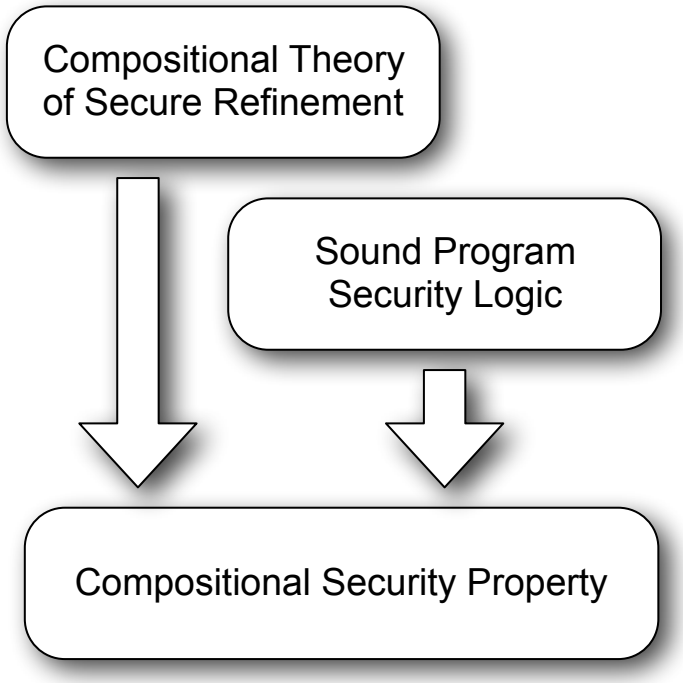



```
skip   /* control +=_m AsmNoW */;
skip   /* temp +=_m AsmNoRW */;
temp := buffer;
if control == 0 then
  low-var := temp
else
  high-var := temp
endif;
temp := 0;
skip   /* temp -=_m AsmNoRW */
```

Verification Framework: Requirements

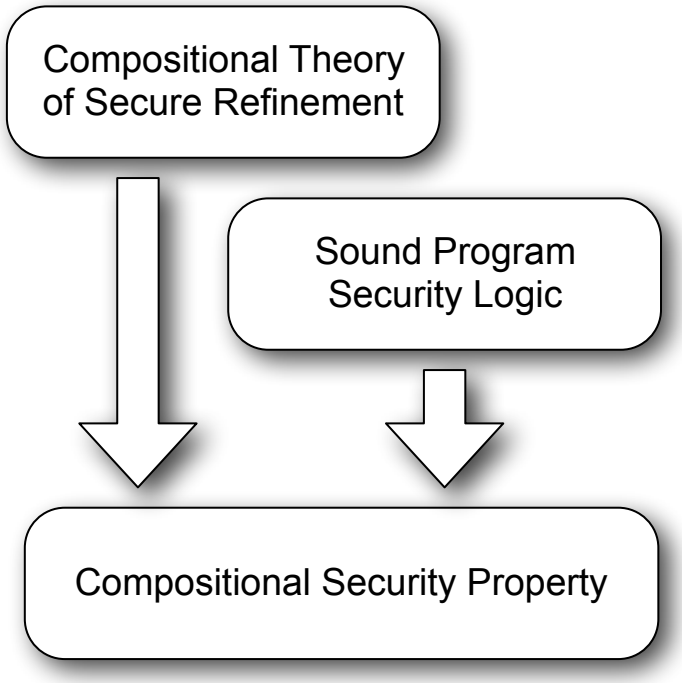


Verification Framework: Requirements

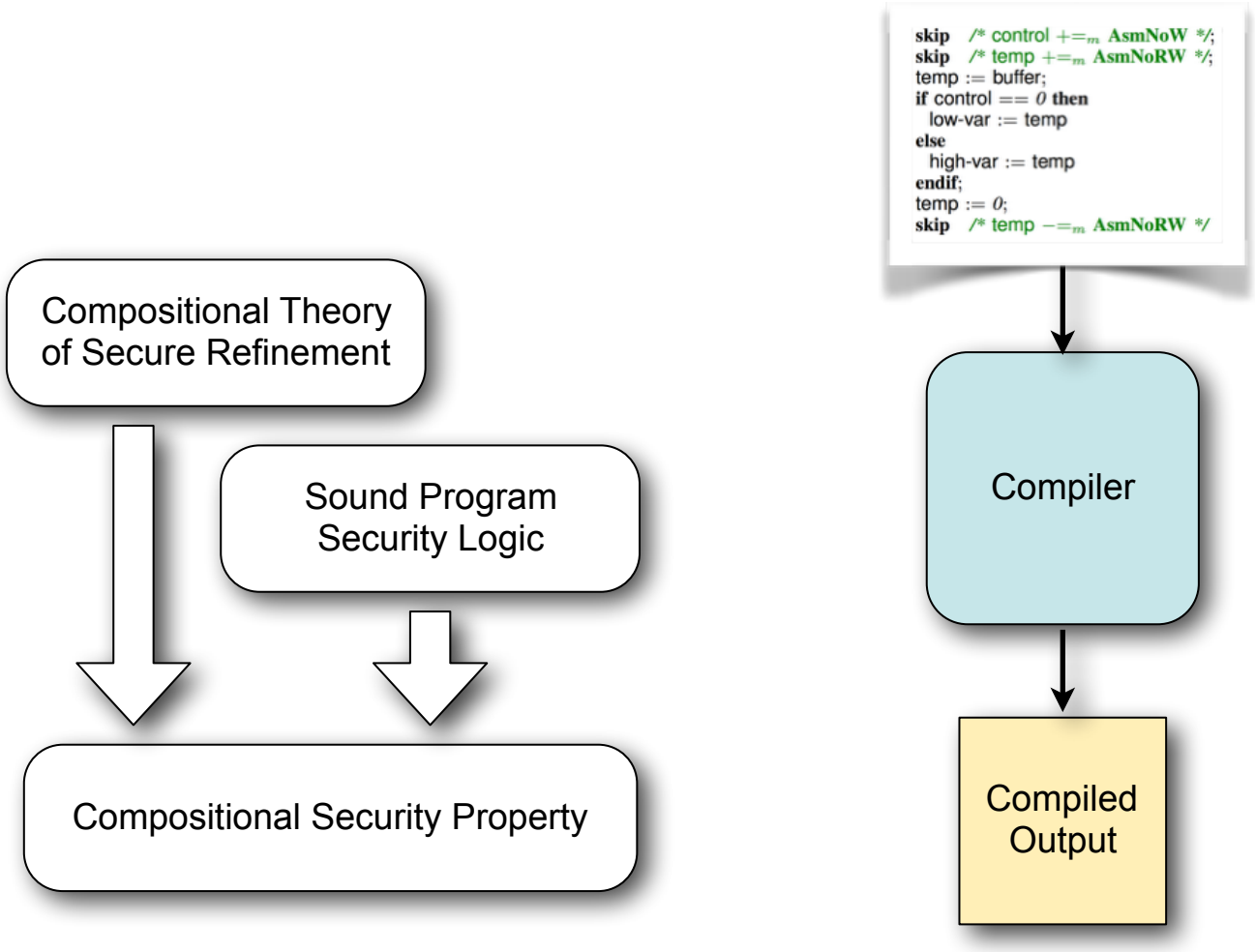


Verification Framework: Requirements

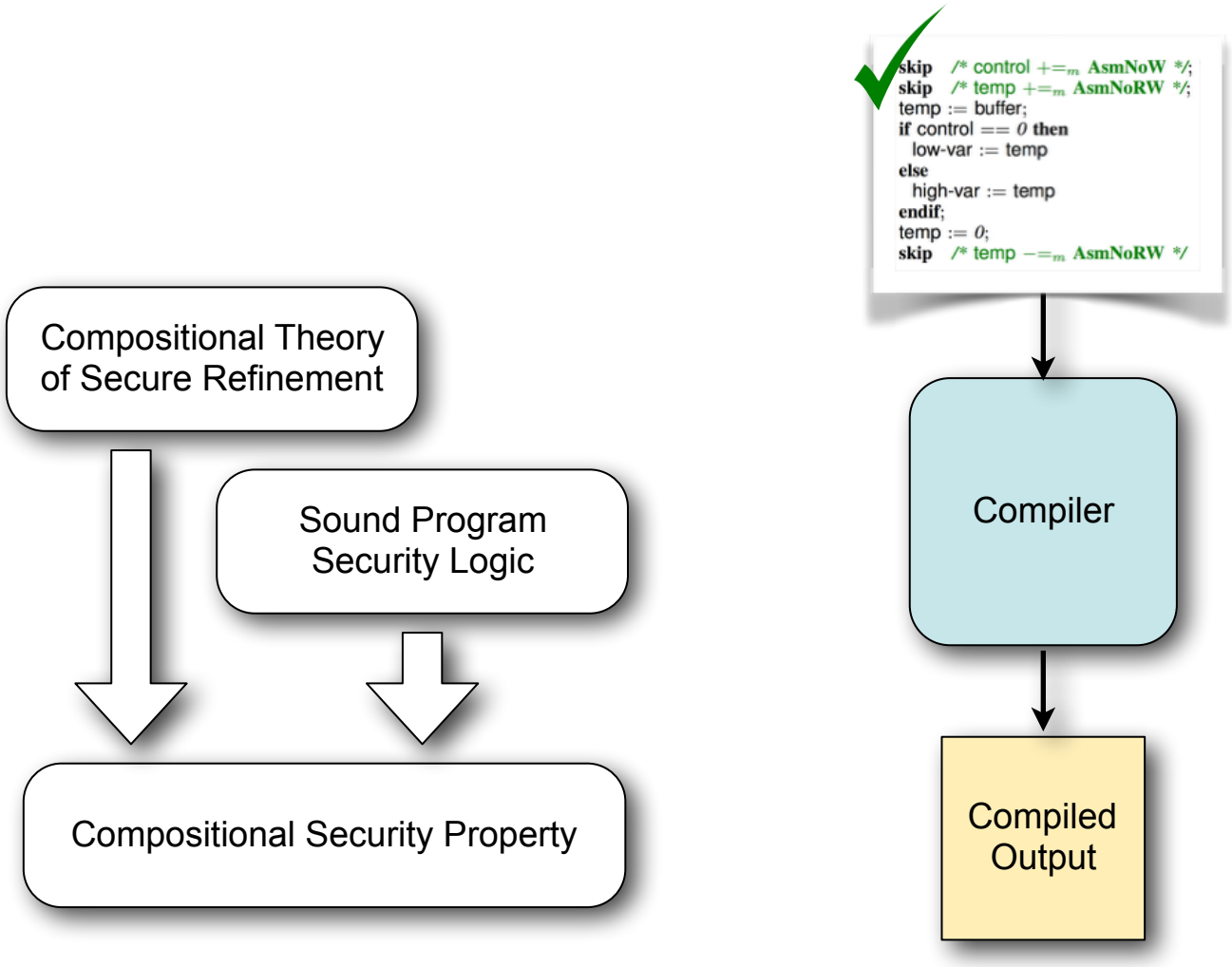
```
skip   /* control +=_m AsmNoW */;
skip   /* temp +=_m AsmNoRW */;
temp := buffer;
if control == 0 then
  low-var := temp
else
  high-var := temp
endif;
temp := 0;
skip   /* temp -=_m AsmNoRW */
```



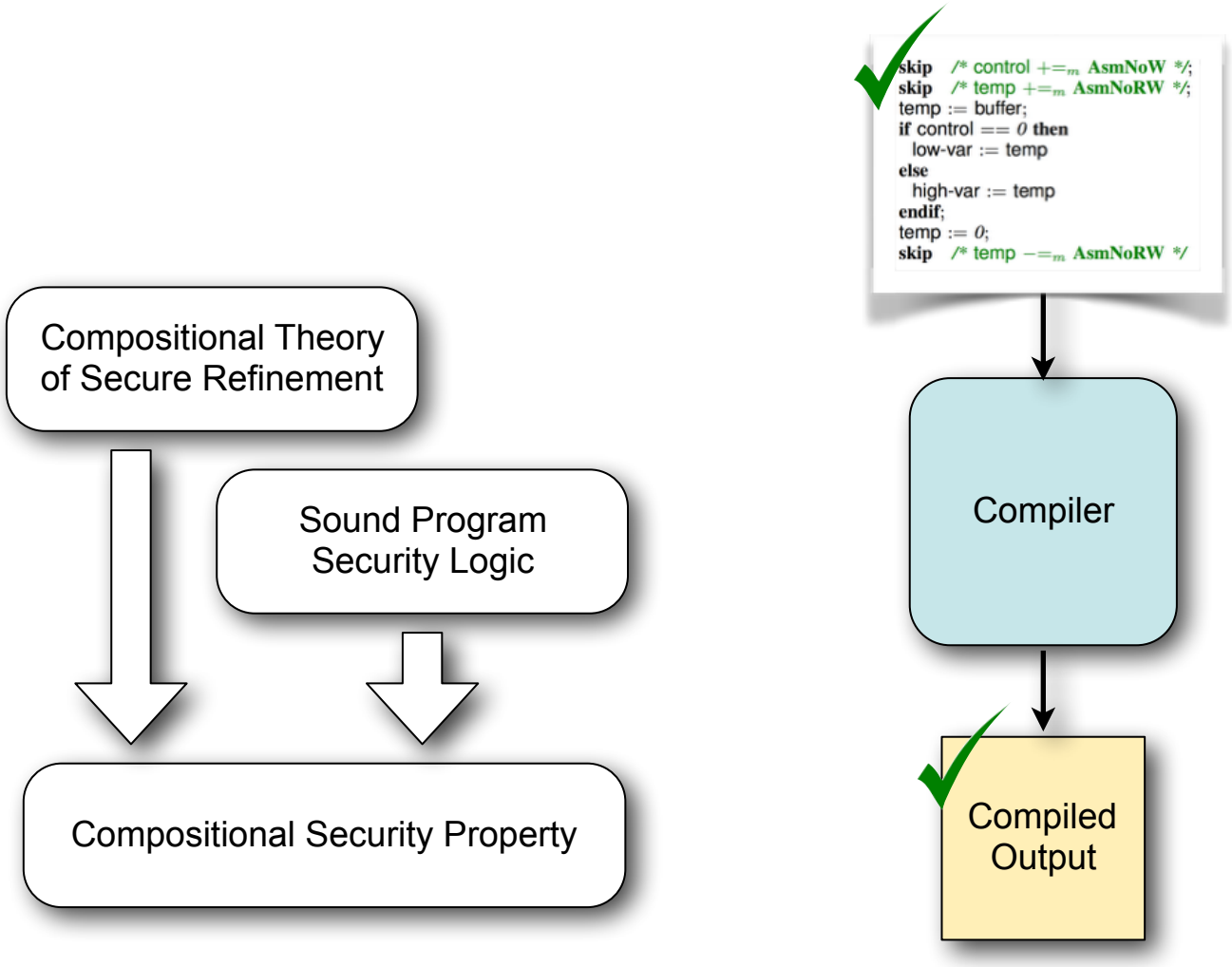
Verification Framework: Requirements



Verification Framework: Requirements

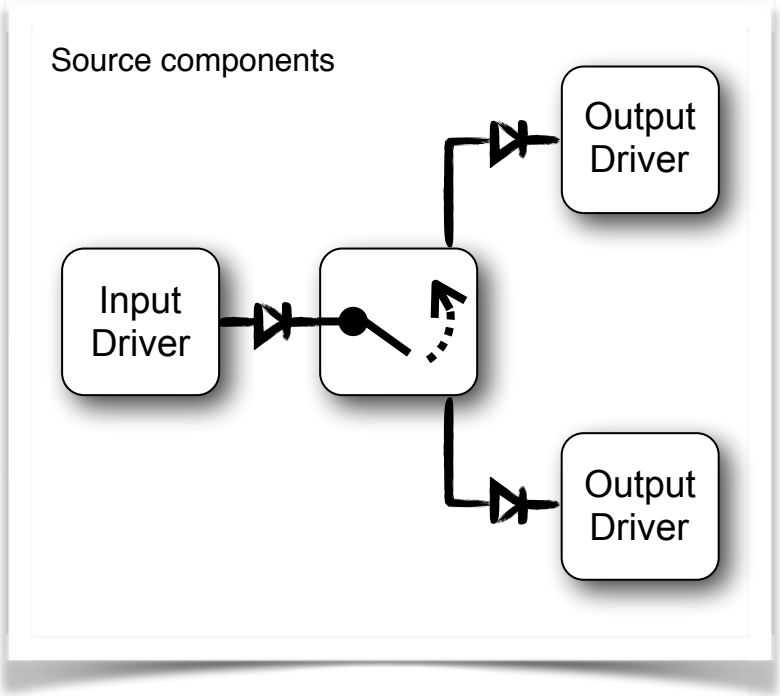


Verification Framework: Requirements

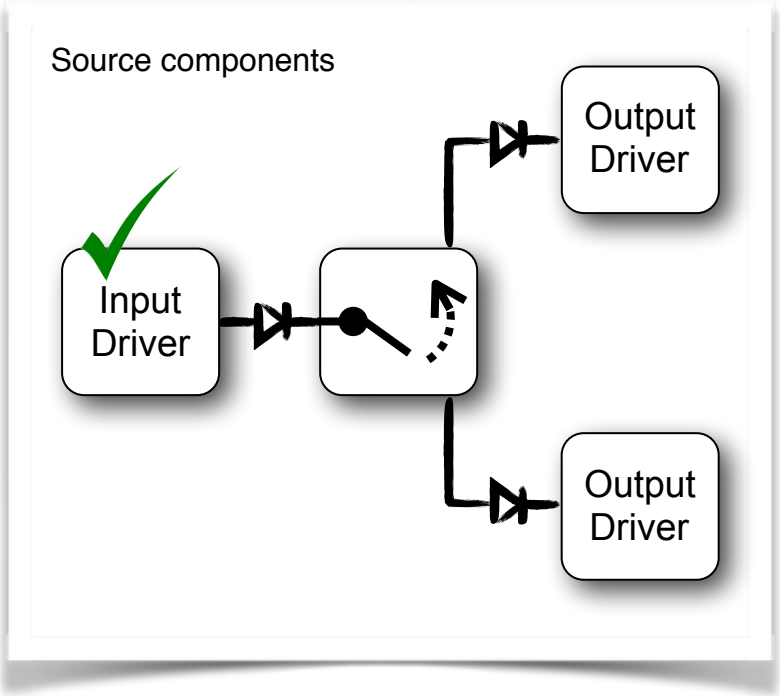


Verification Framework: Operation

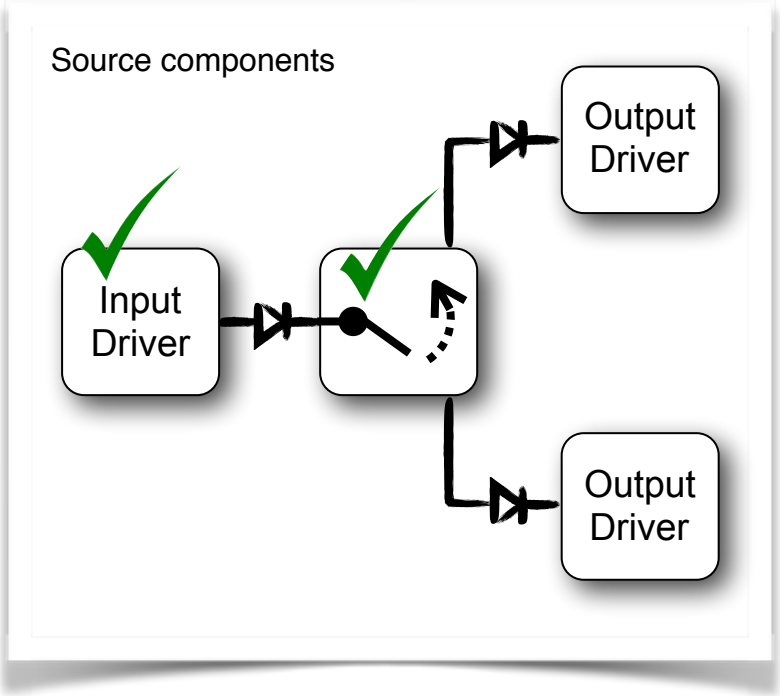
Verification Framework: Operation



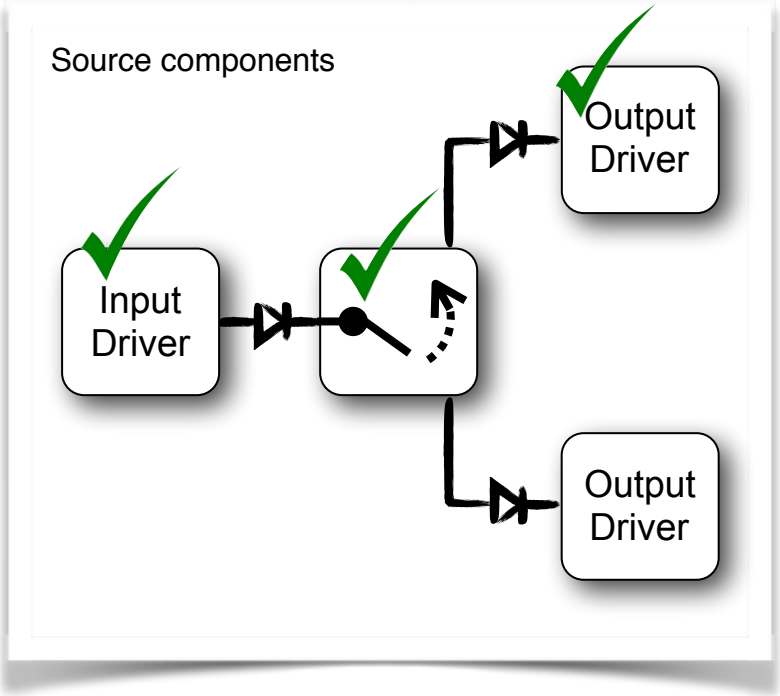
Verification Framework: Operation



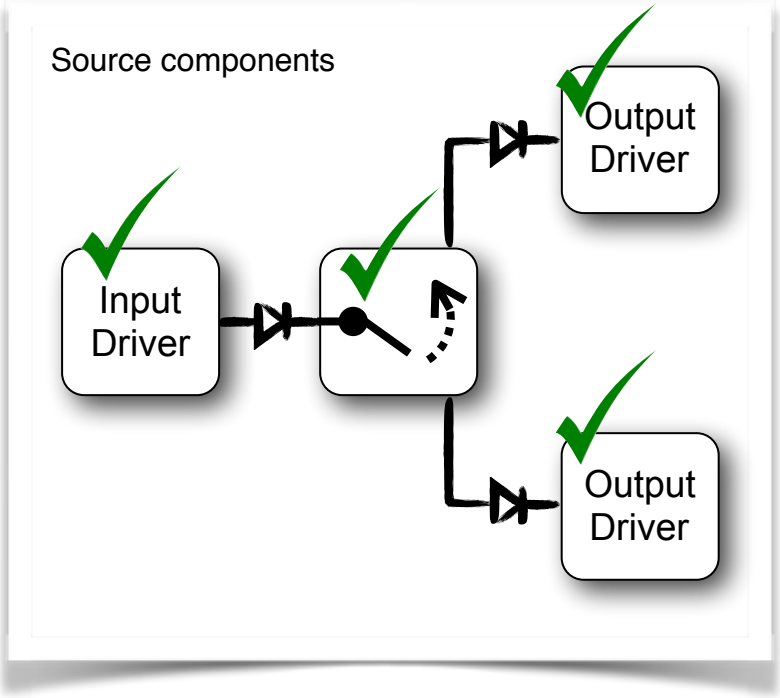
Verification Framework: Operation



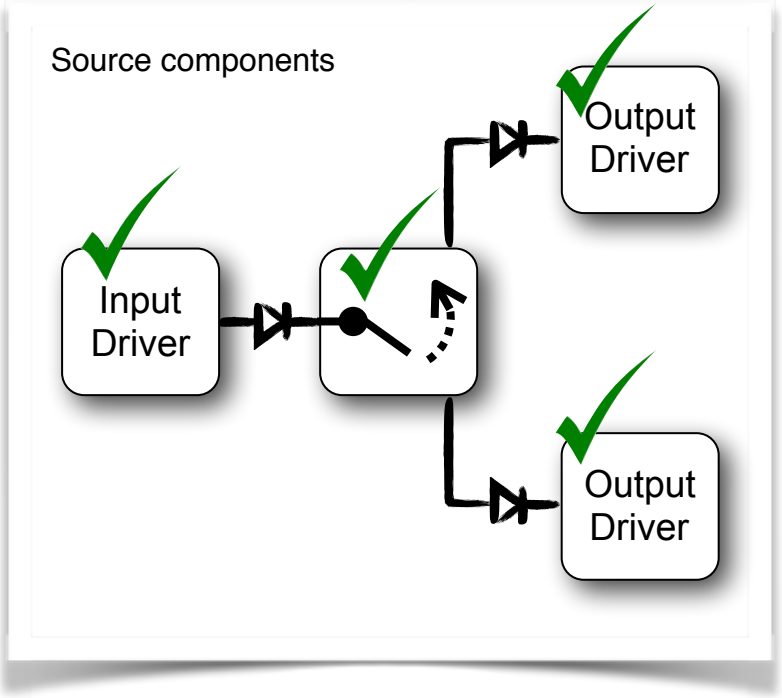
Verification Framework: Operation

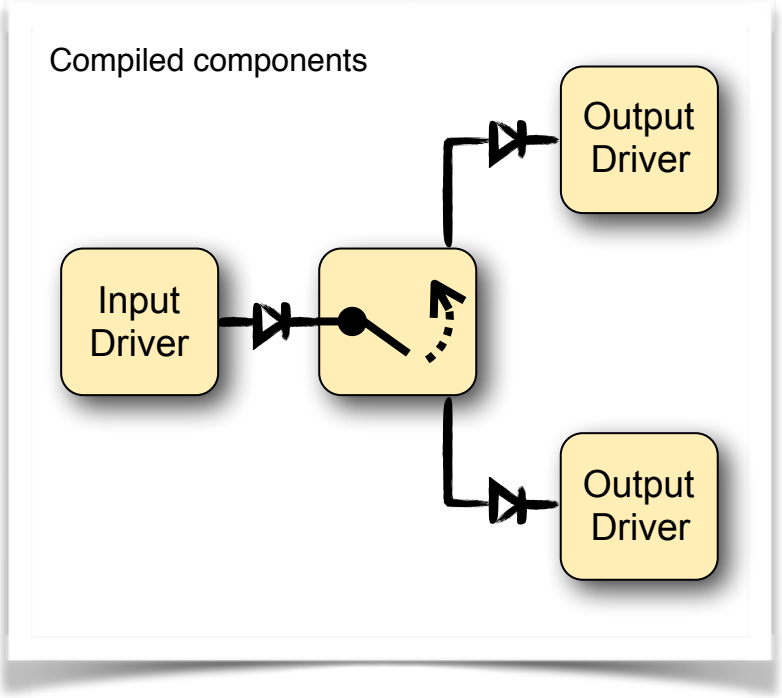


Verification Framework: Operation

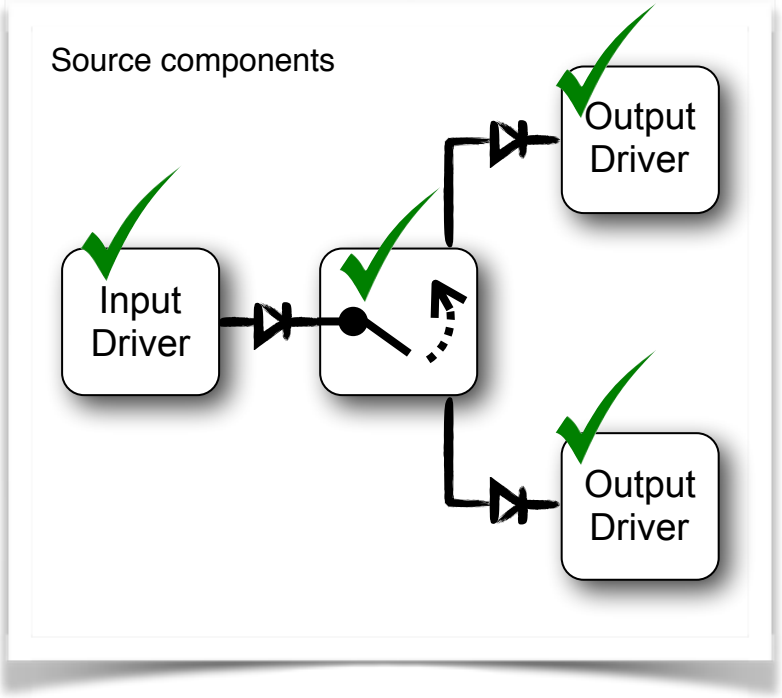


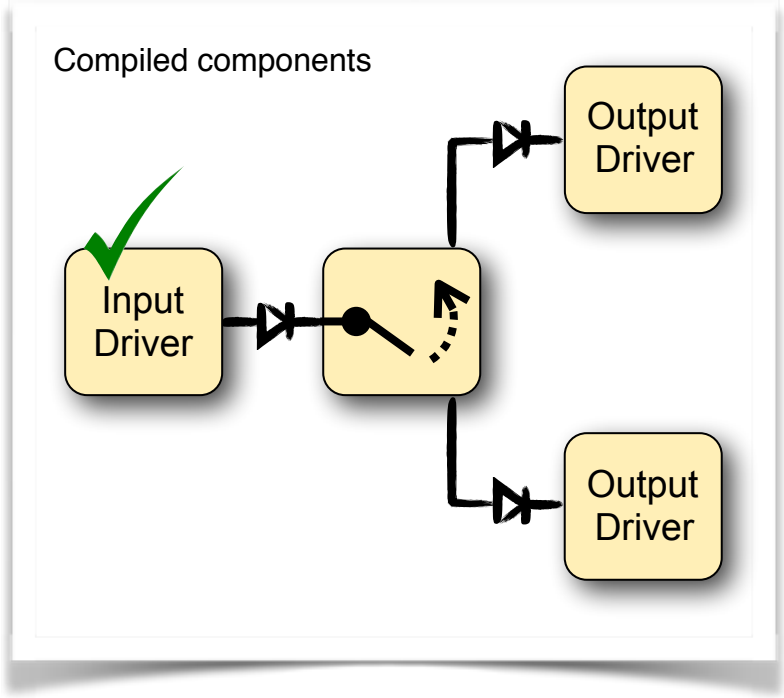
Verification Framework: Operation



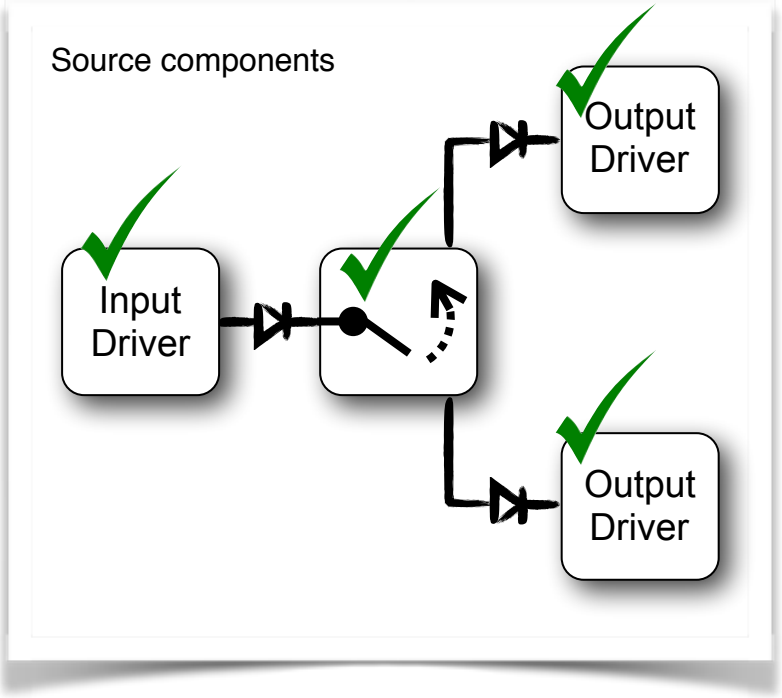


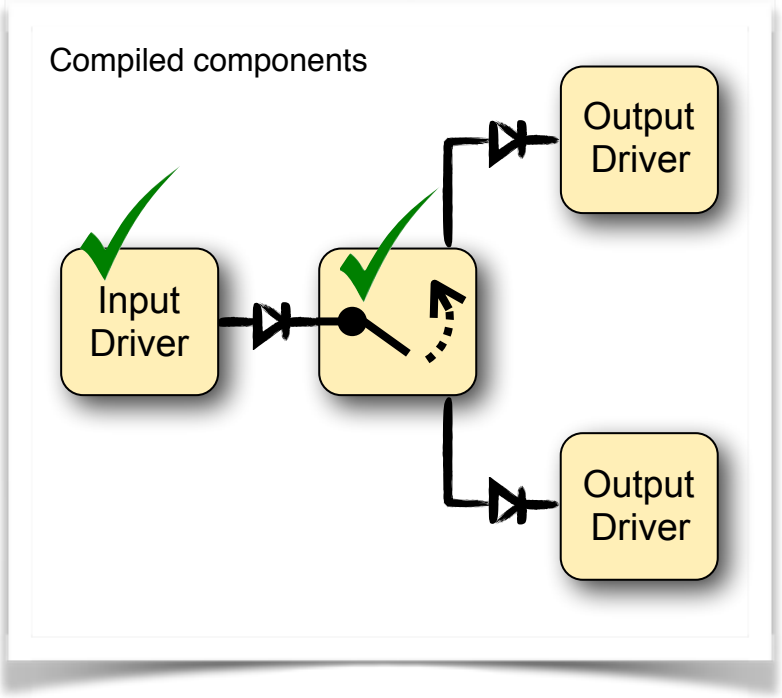
Verification Framework: Operation



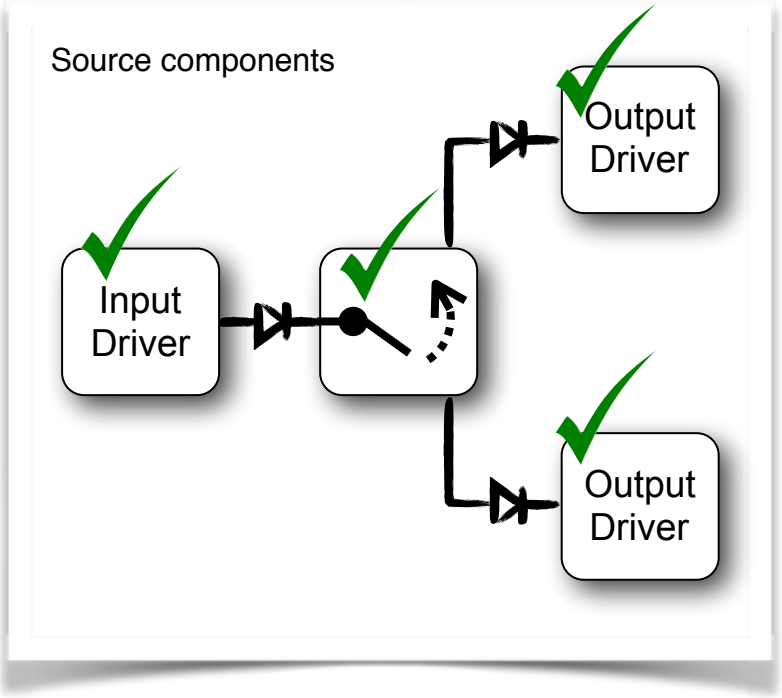


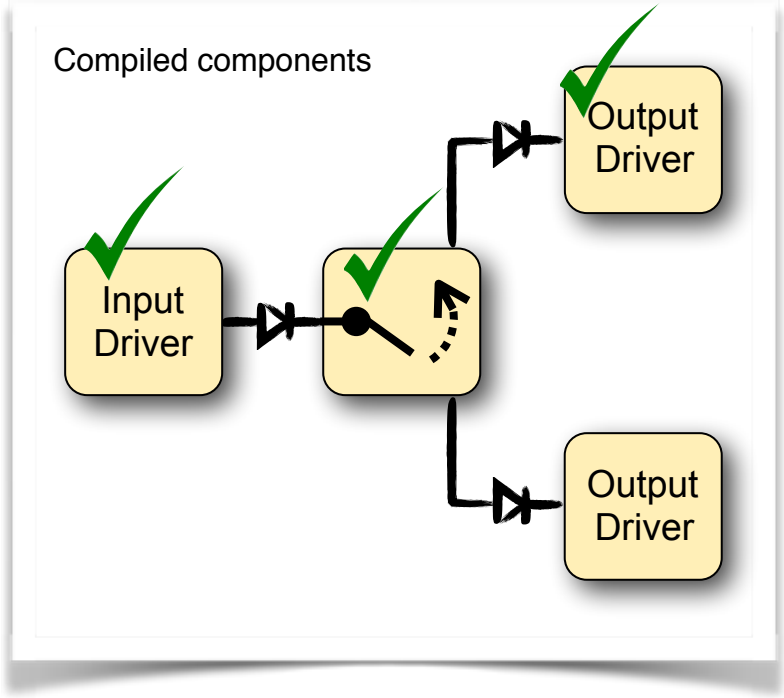
Verification Framework: Operation



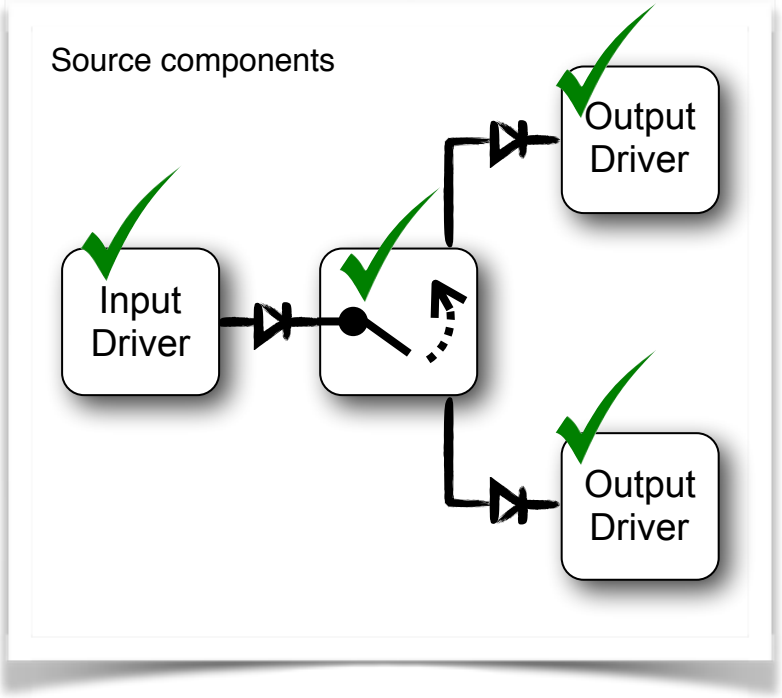


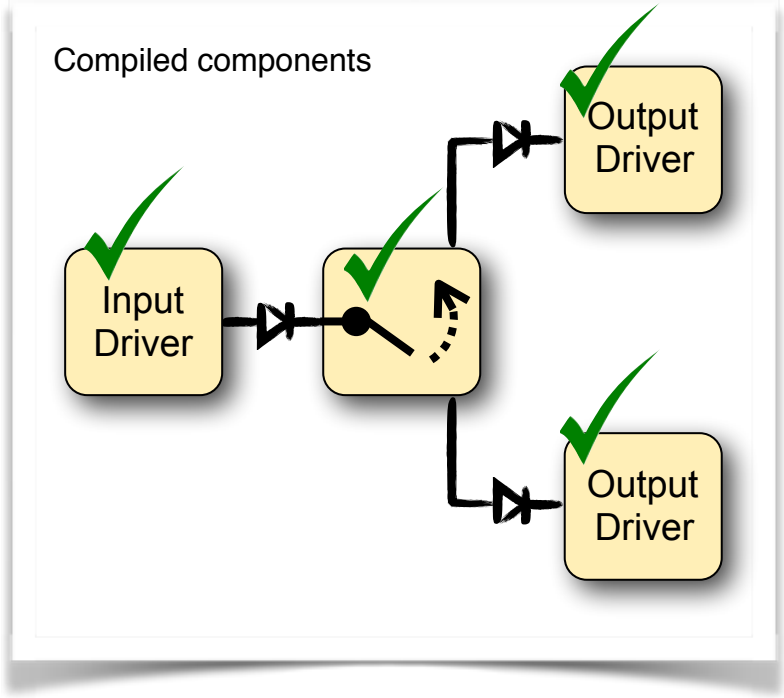
Verification Framework: Operation



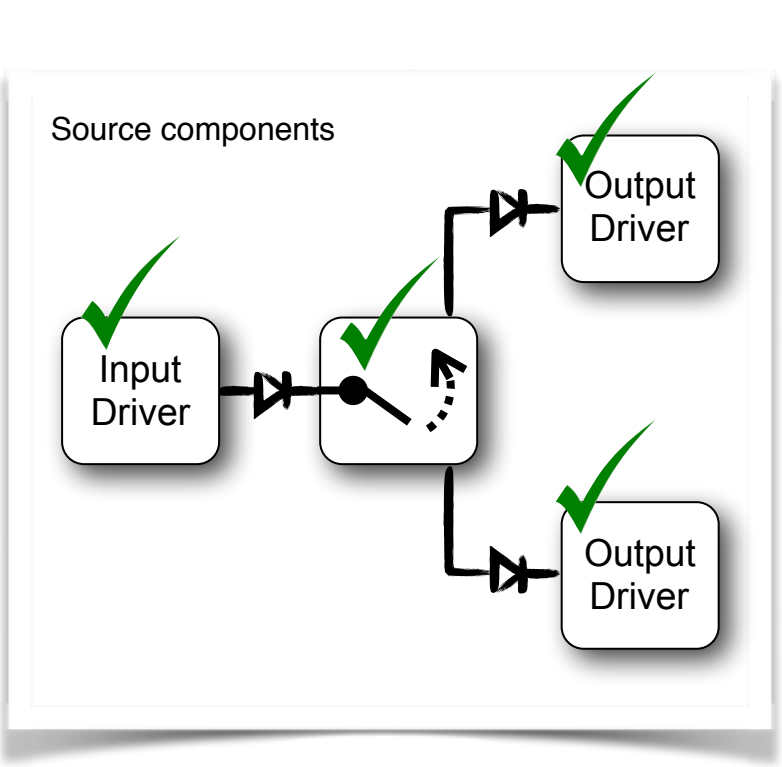


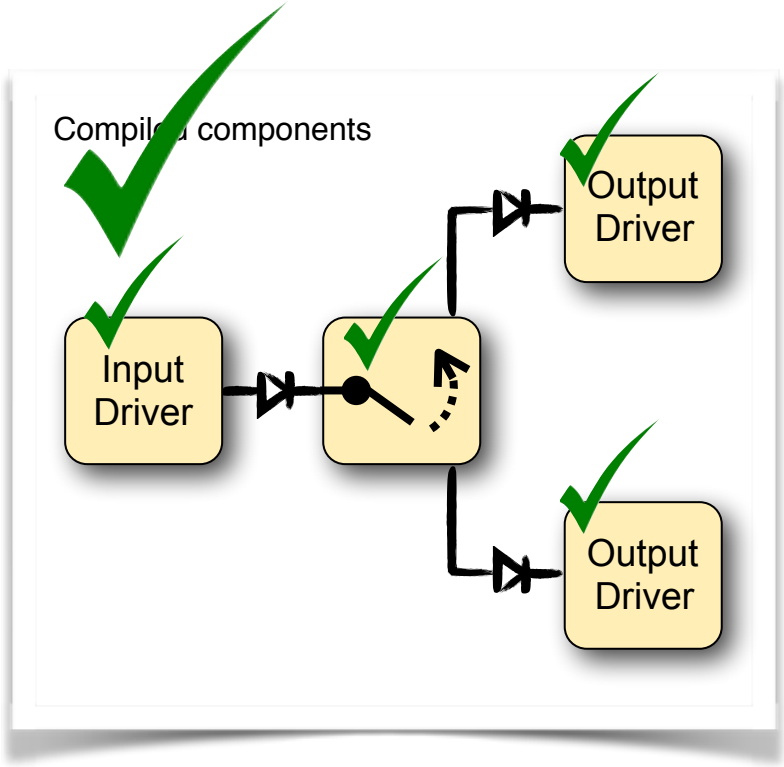
Verification Framework: Operation



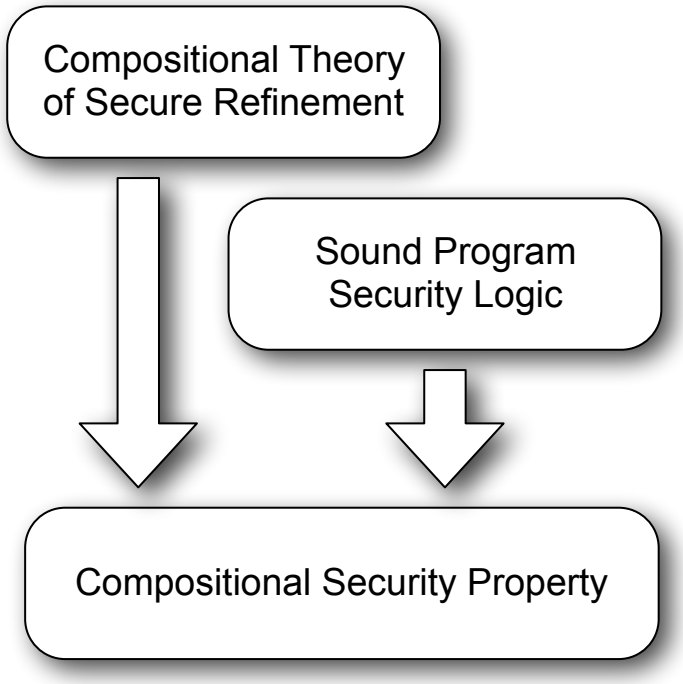


Verification Framework: Operation

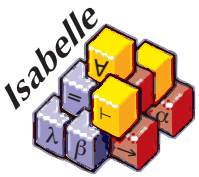


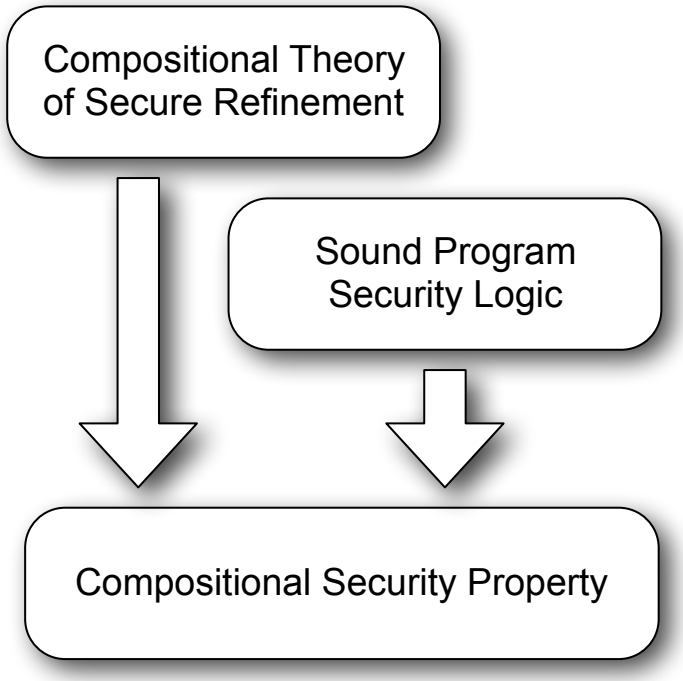


Verification Framework: Status



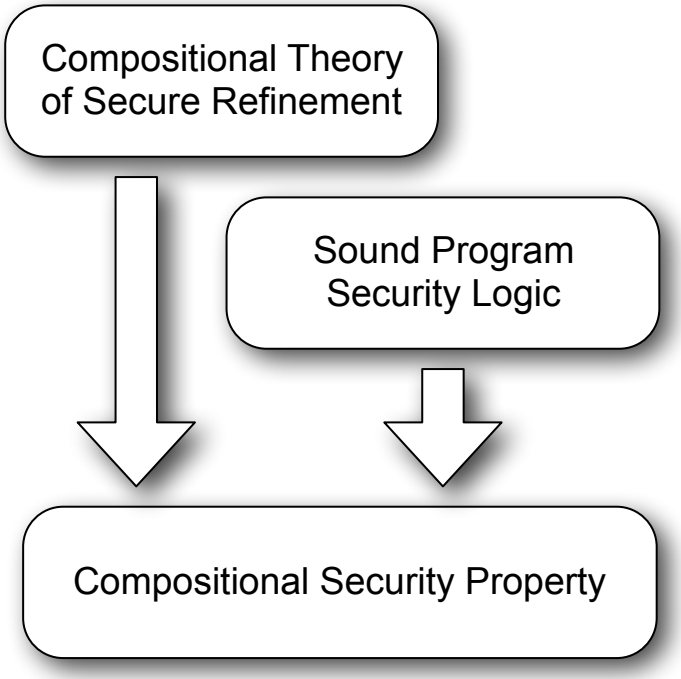
Verification Framework: Status



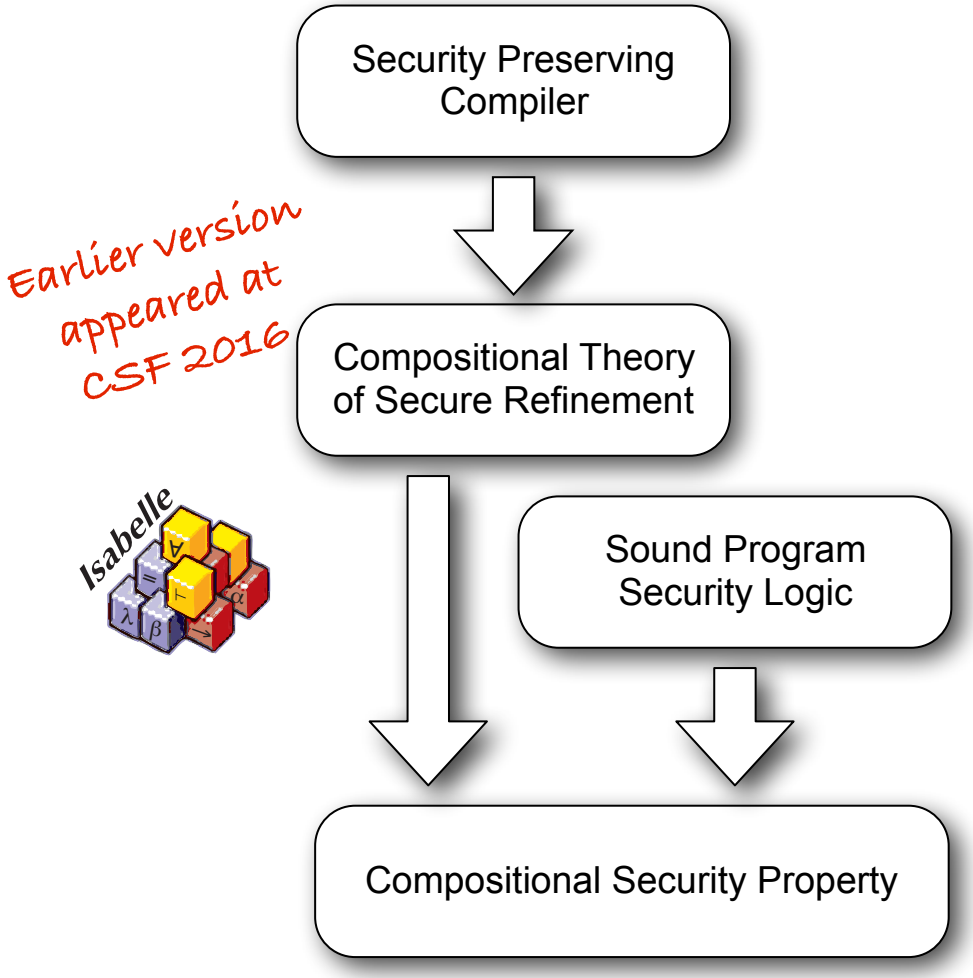


Verification Framework: Status

Earlier version appeared at CSF 2016



Verification Framework: Status



Verification Framework: Status

Robert Sison

Security Preserving Compiler

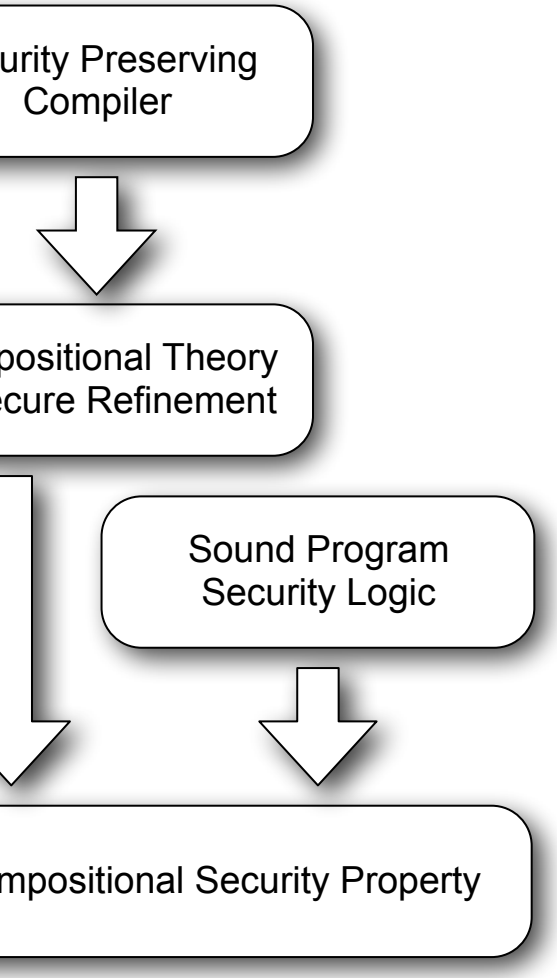
Earlier version appeared at CSF 2016

Compositional Theory of Secure Refinement

Sound Program Security Logic

Compositional Security Property

Verification Framework: Status



Earlier version appeared at CSF 2016

Robert Sison

Christine Rizkallah

Verification Framew

From simple While language to a simple RISC language

(integrated with earlier version of the logic)

Security Preserving Compiler

Earlier version appeared at CSF 2016

Compositional Theory of Secure Refinement

Sound Program Security Logic

Compositional Security Property

Robert Sison

Christine Rizkallah

Verification Framew

From simple While language to a simple RISC language

(integrated with earlier version of the logic)

Security Preserving Compiler

Earlier version appeared at CSF 2016

Compositional Theory of Secure Refinement

Sound Program Security Logic

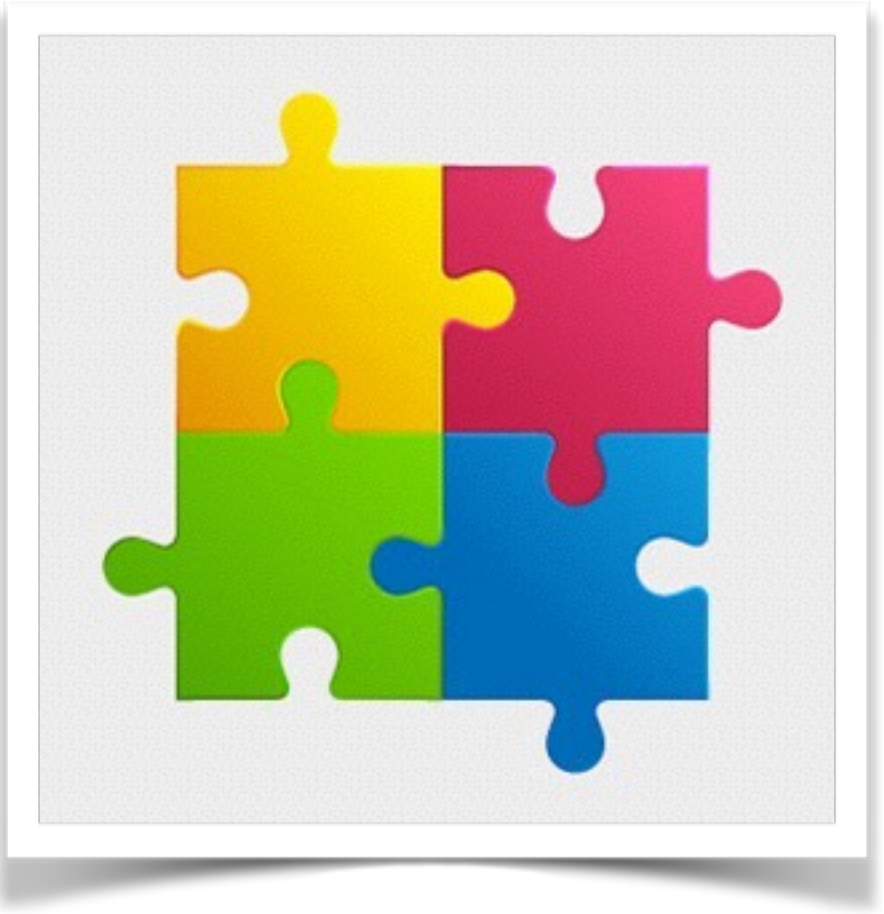
EuroS&P 2018

Compositional Security Property

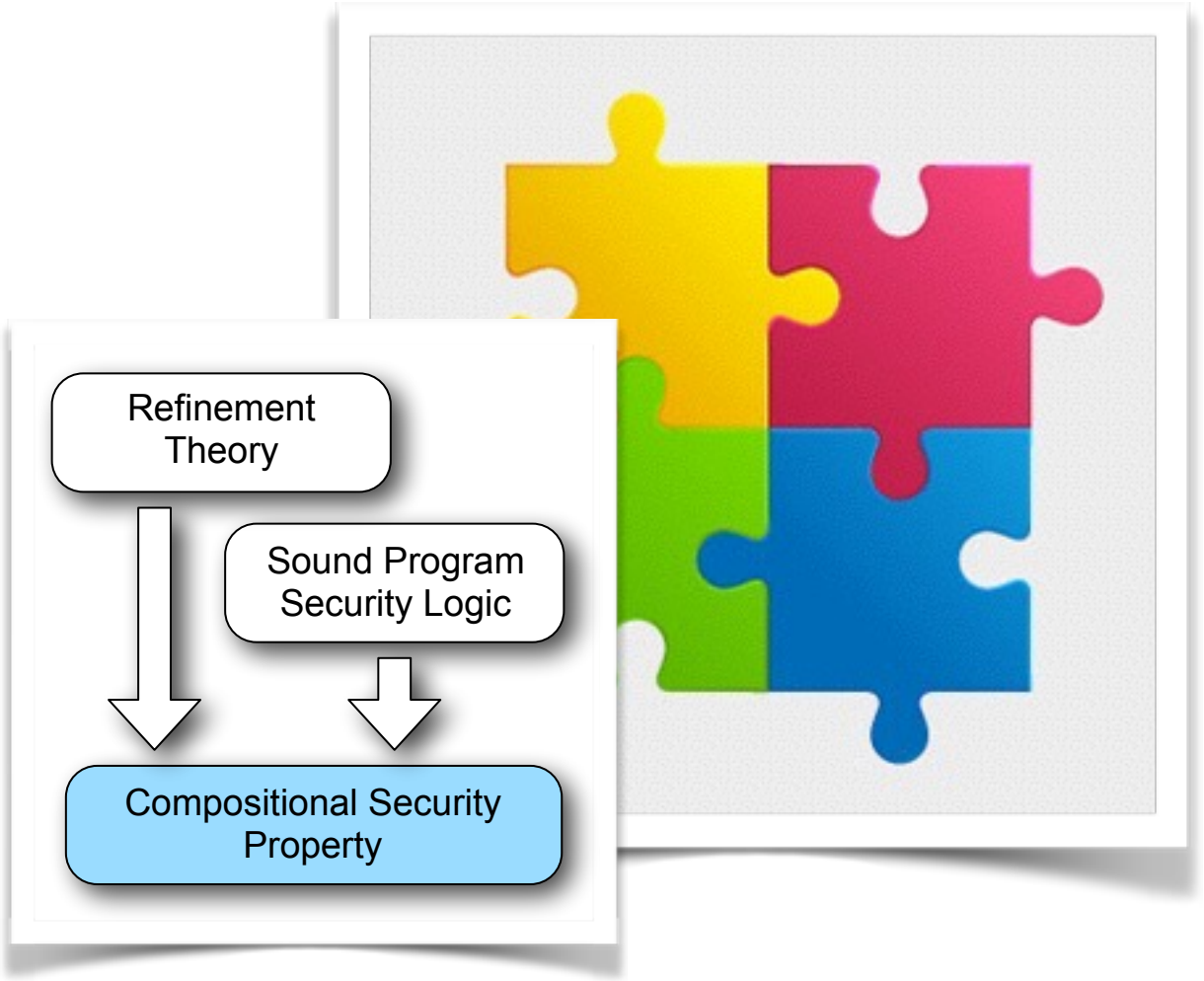
Robert Sison

Christine Rizkallah

COMPOSITIONAL SECURITY



COMPOSITIONAL SECURITY



Minimum Requirements

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

Minimum Requirements

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

Secure?

Minimum Requirements

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

Secure?

Timing-Insensitive Security:

Minimum Requirements

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

Secure?

Timing-Insensitive Security: **YES**

Minimum Requirements

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

Secure?

Timing-Insensitive Security: **YES**

Timing-Sensitive Security:

Minimum Requirements

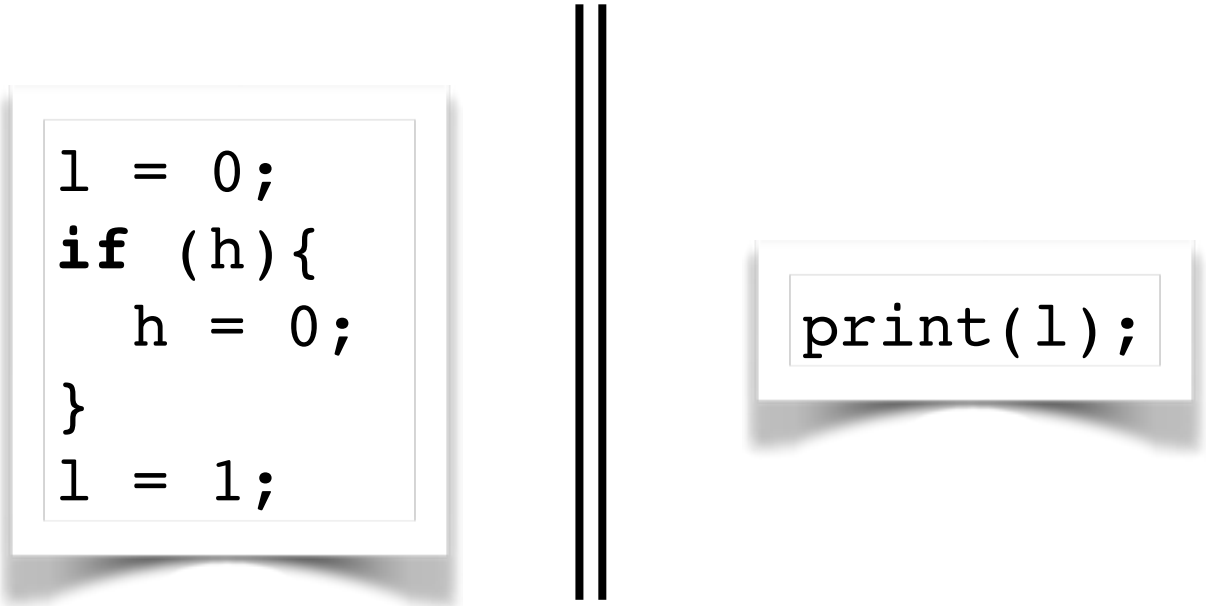
```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

Secure?

Timing-Insensitive Security: **YES**

Timing-Sensitive Security: **NO**

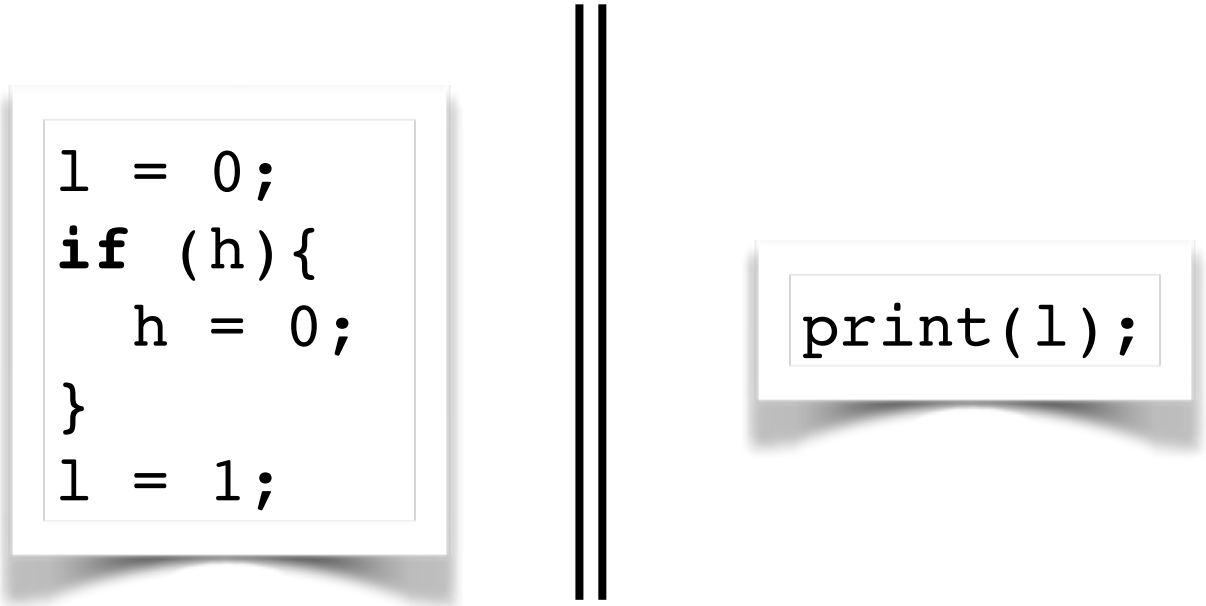
Minimum Requirements



Secure?

Timing-Insensitive Security:

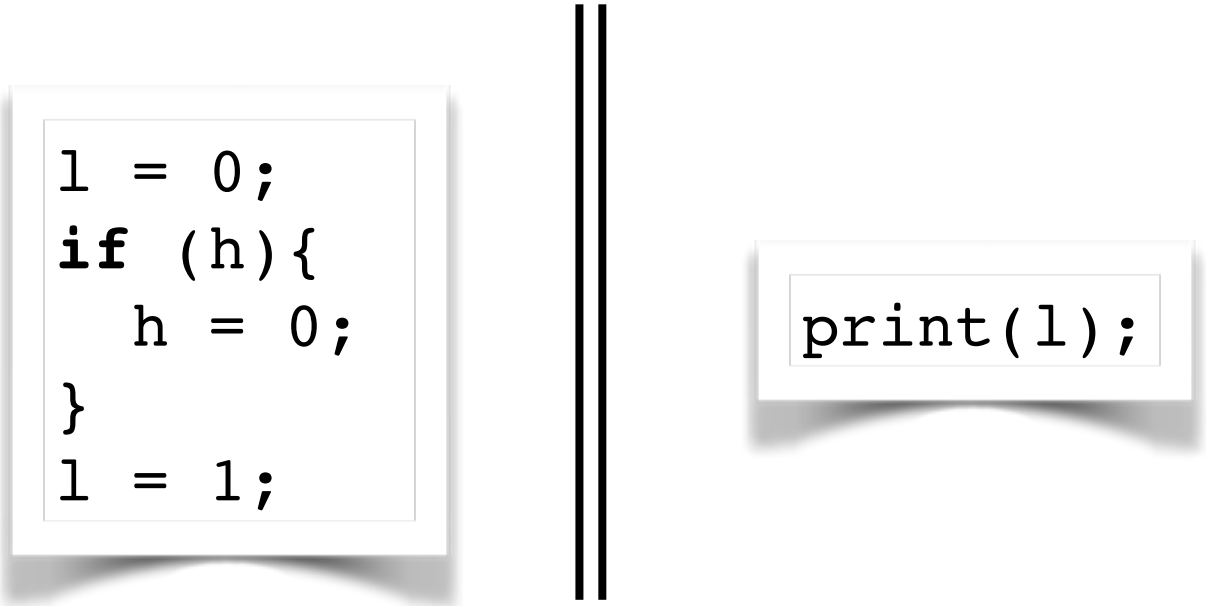
Minimum Requirements



Secure?

Timing-Insensitive Security: **NO**

Minimum Requirements

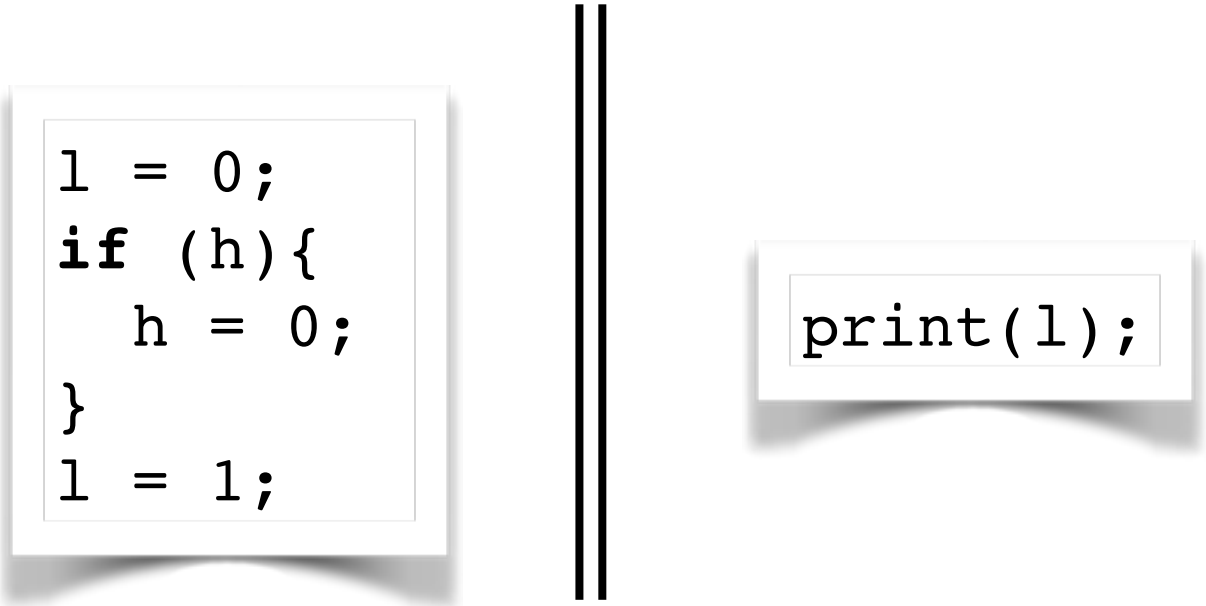


Secure?

Timing-Insensitive Security: **NO**

Timing-Sensitive Security:

Minimum Requirements

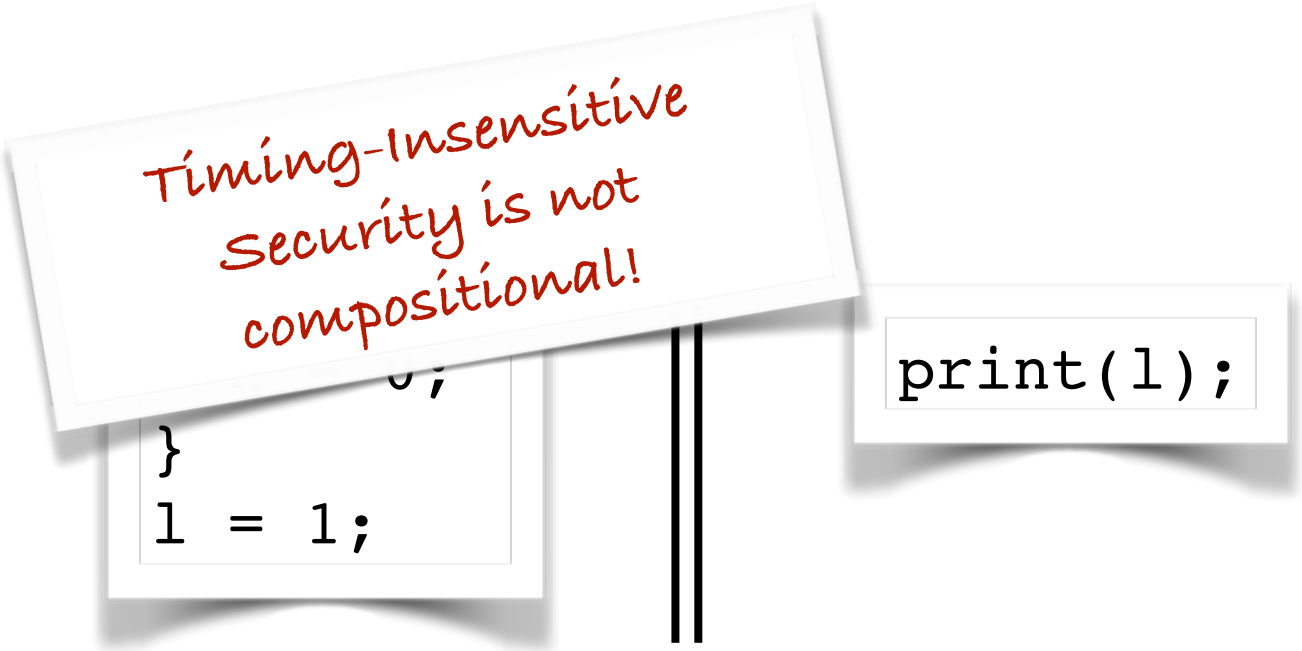


Secure?

Timing-Insensitive Security: **NO**

Timing-Sensitive Security: **NO** (trivially)

Minimum Requirements

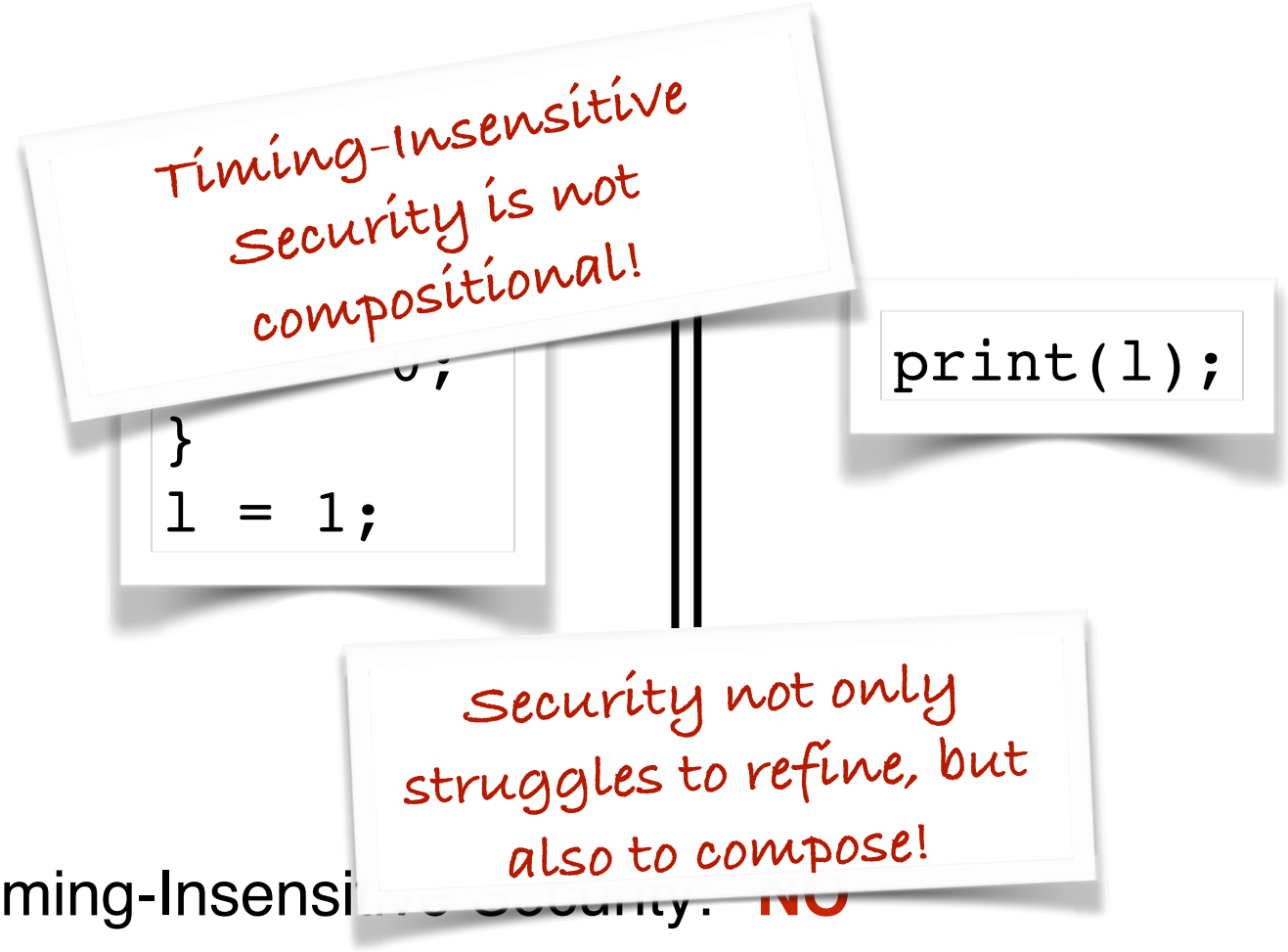


Secure?

Timing-Insensitive Security: **NO**

Timing-Sensitive Security: **NO** (trivially)

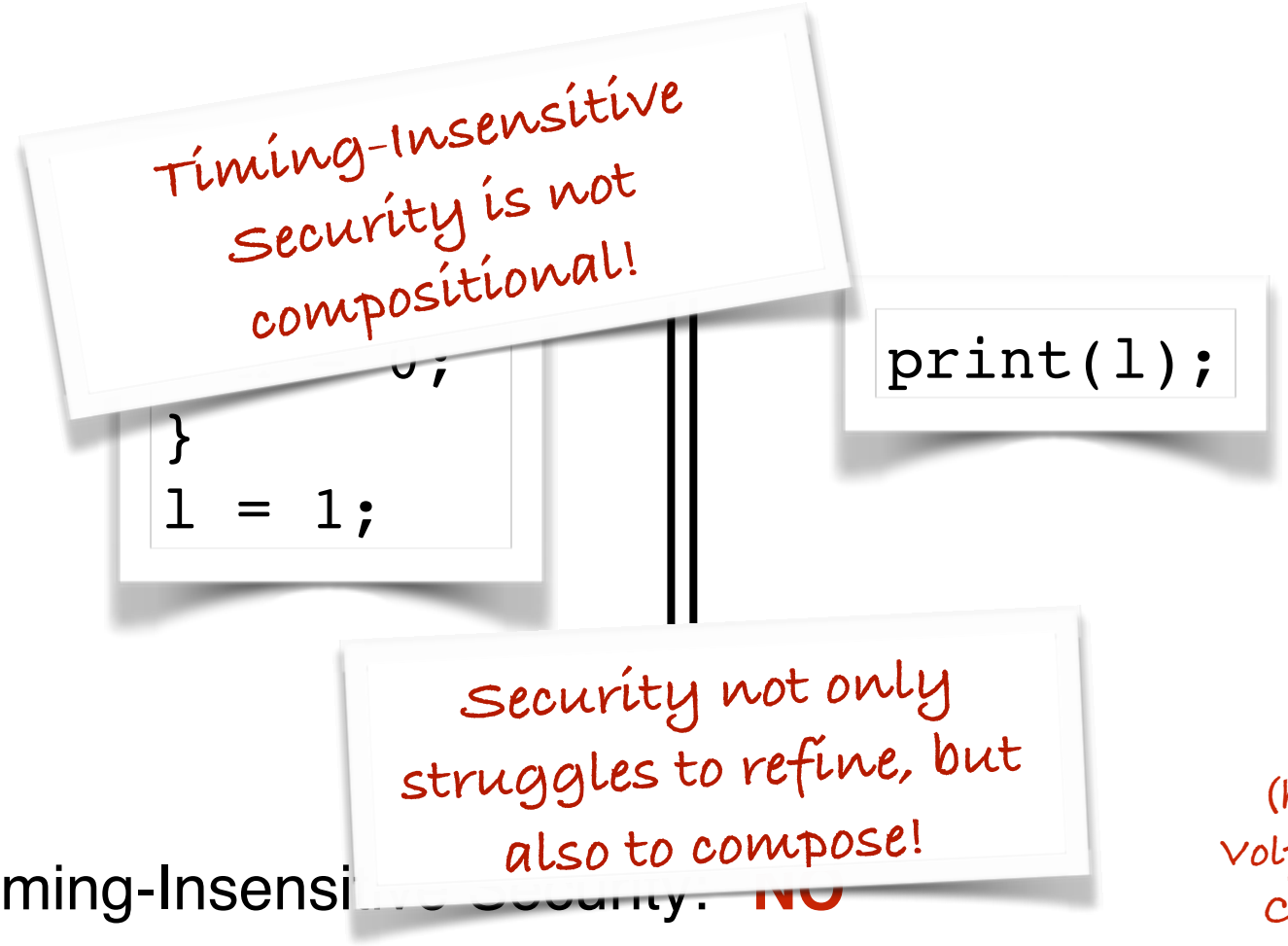
Minimum Requirements



Timing-Insensi… Security: **NO**

Timing-Sensitive Security: **NO** (trivially)

Minimum Requirements

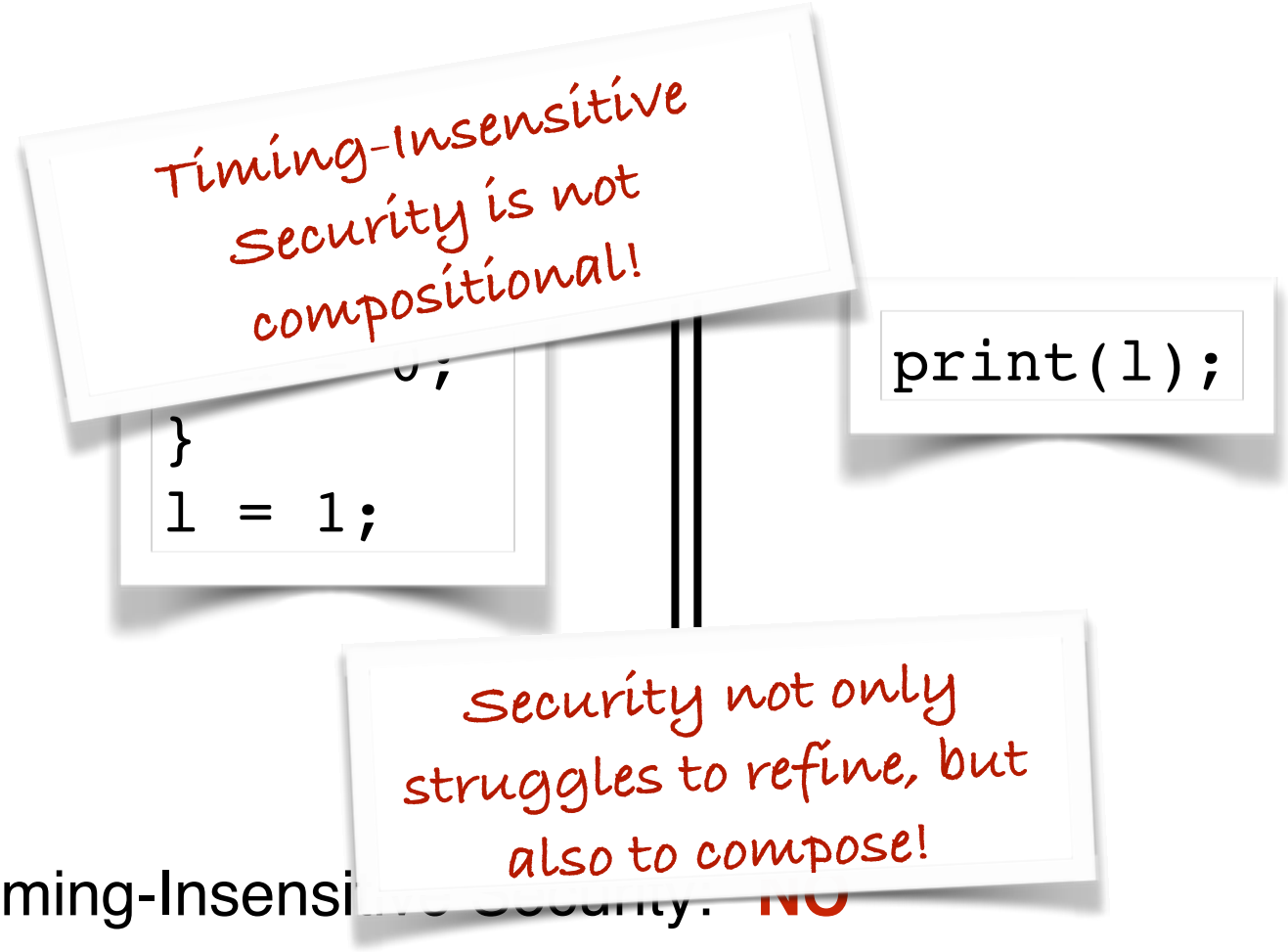


(known since Volpano & Smith, CSFW 1998)

Timing-Insensi… Security: **NO**

Timing-Sensitive Security: **NO** (trivially)

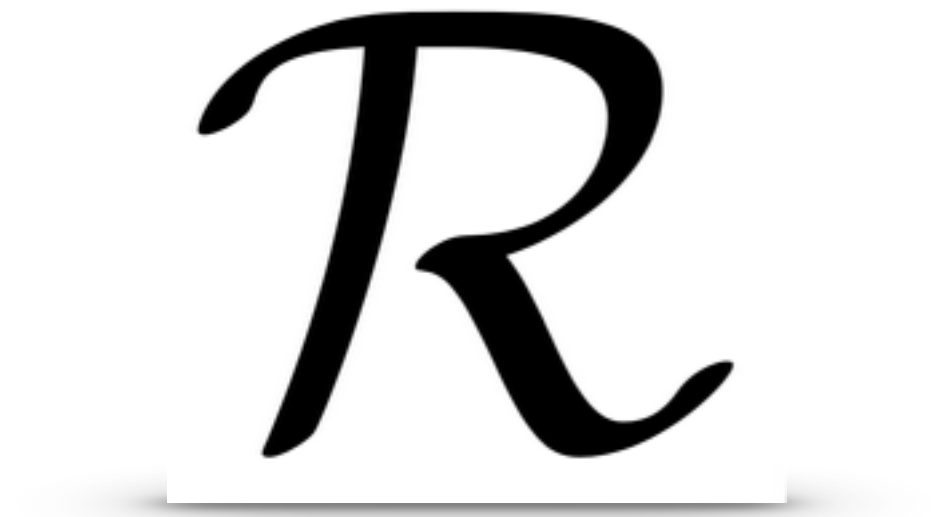
Minimum Requirements



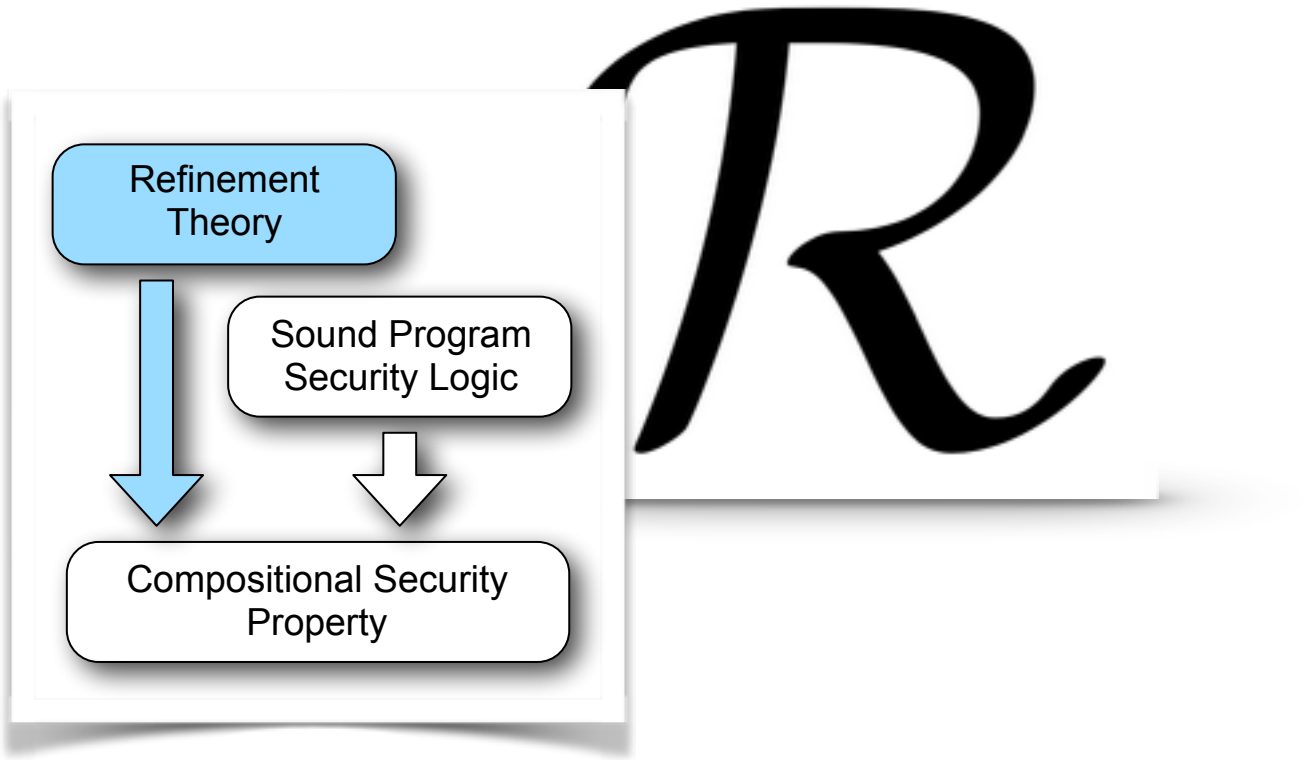
Timing-Insensi… Security: **NO**

Timing-Sensitive Security: **NO** (trivially)

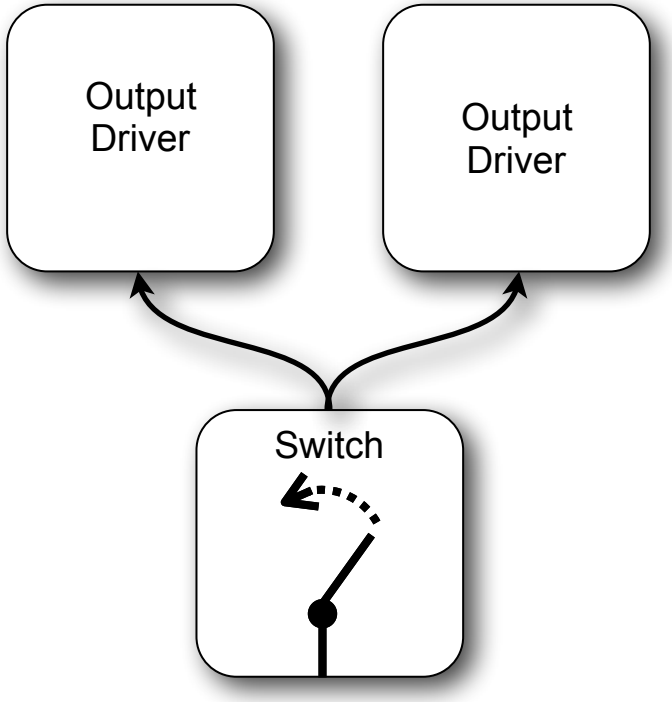
COMPOSITIONAL REFINEMENT



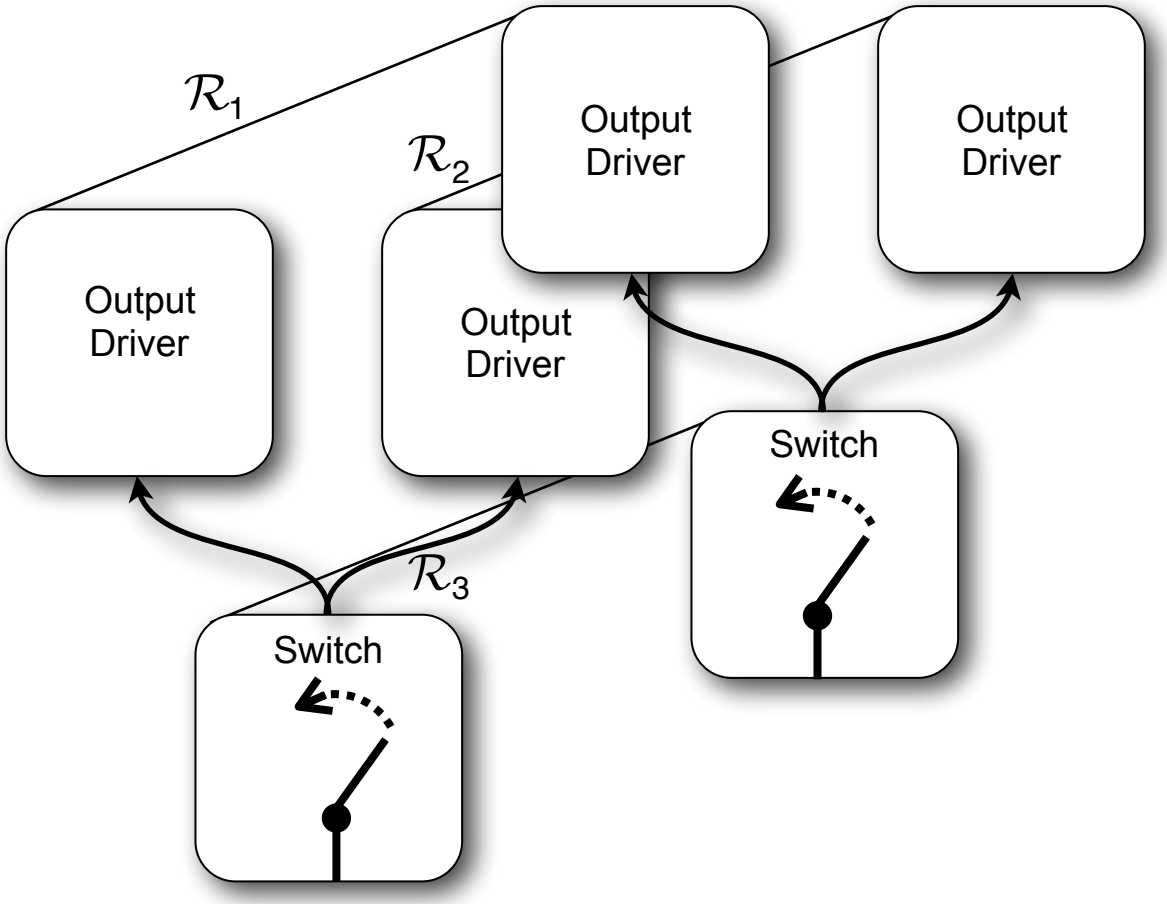
COMPOSITIONAL REFINEMENT



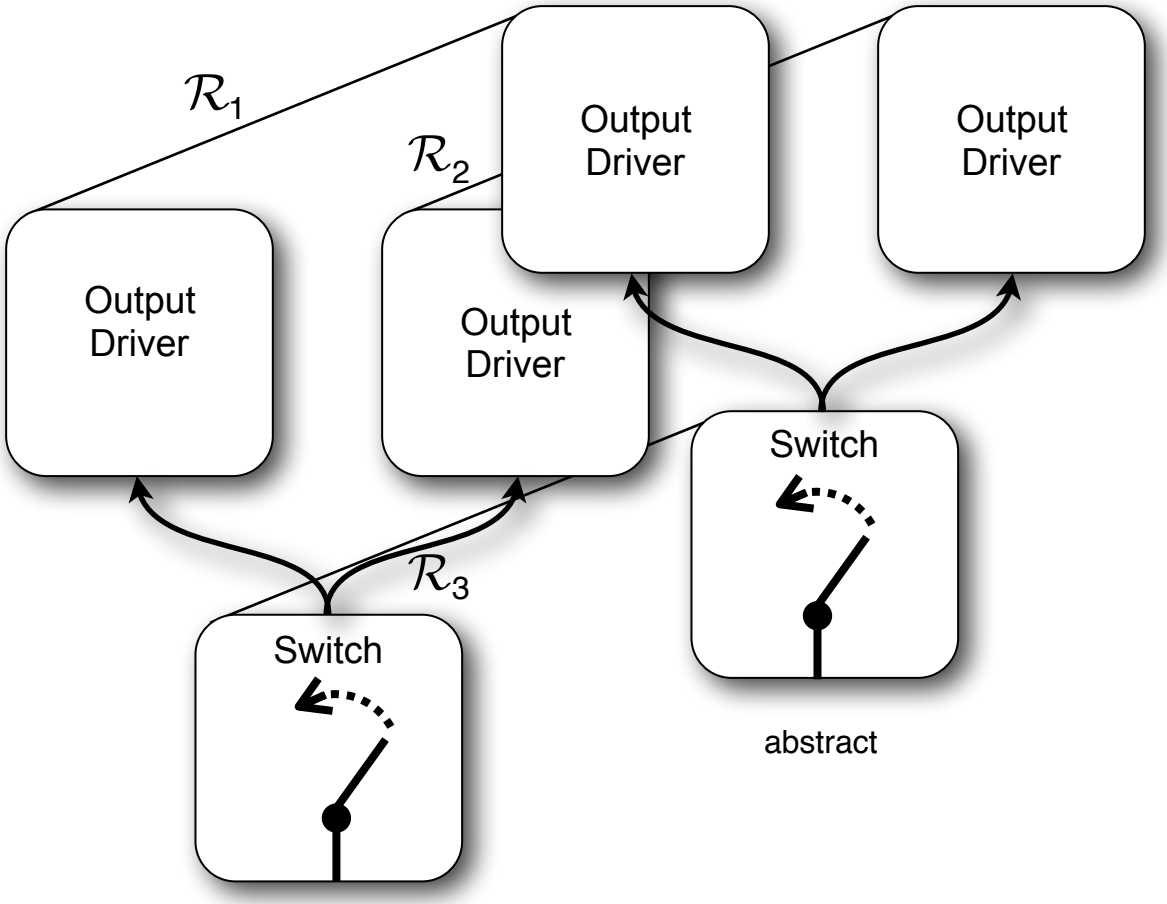
Secure Componentwise Refinement



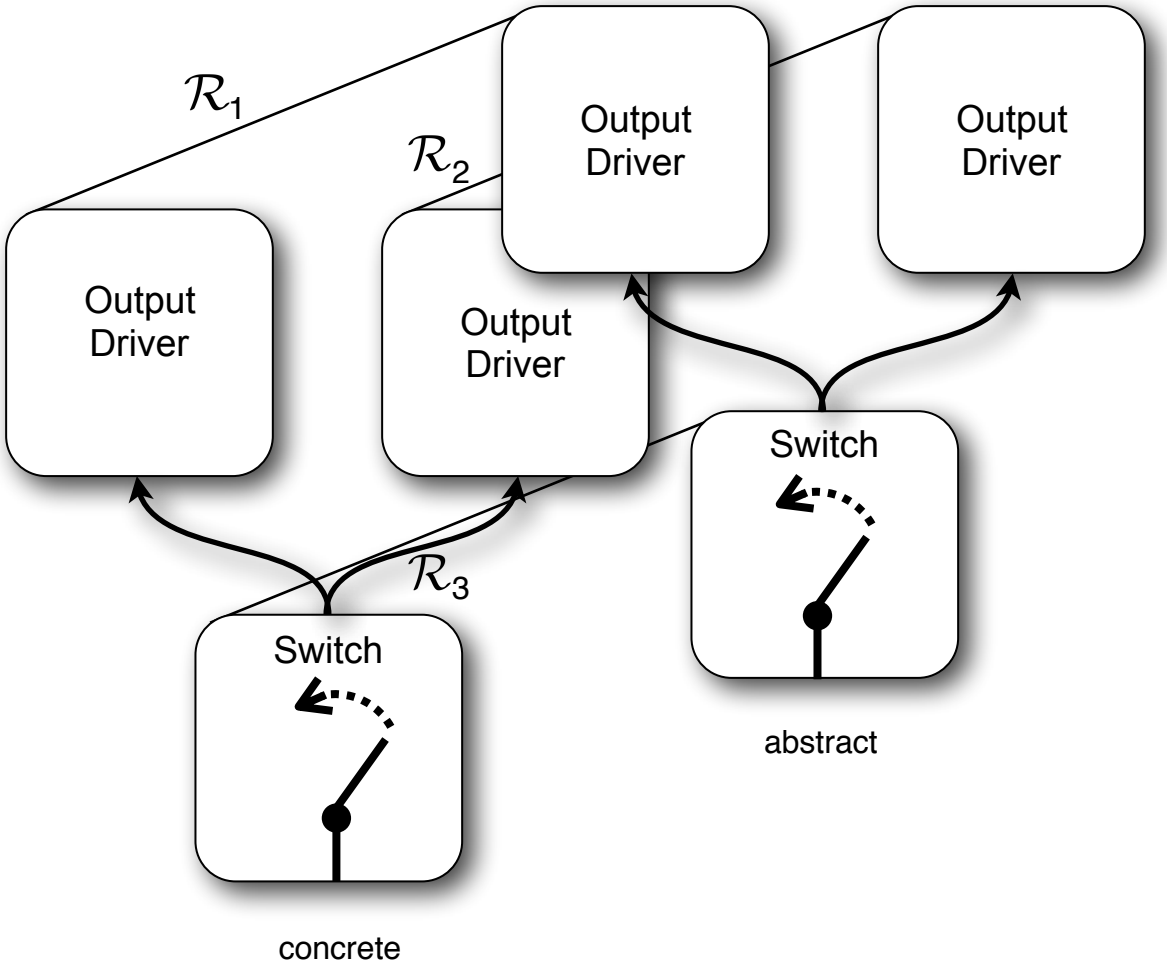
Secure Componentwise Refinement



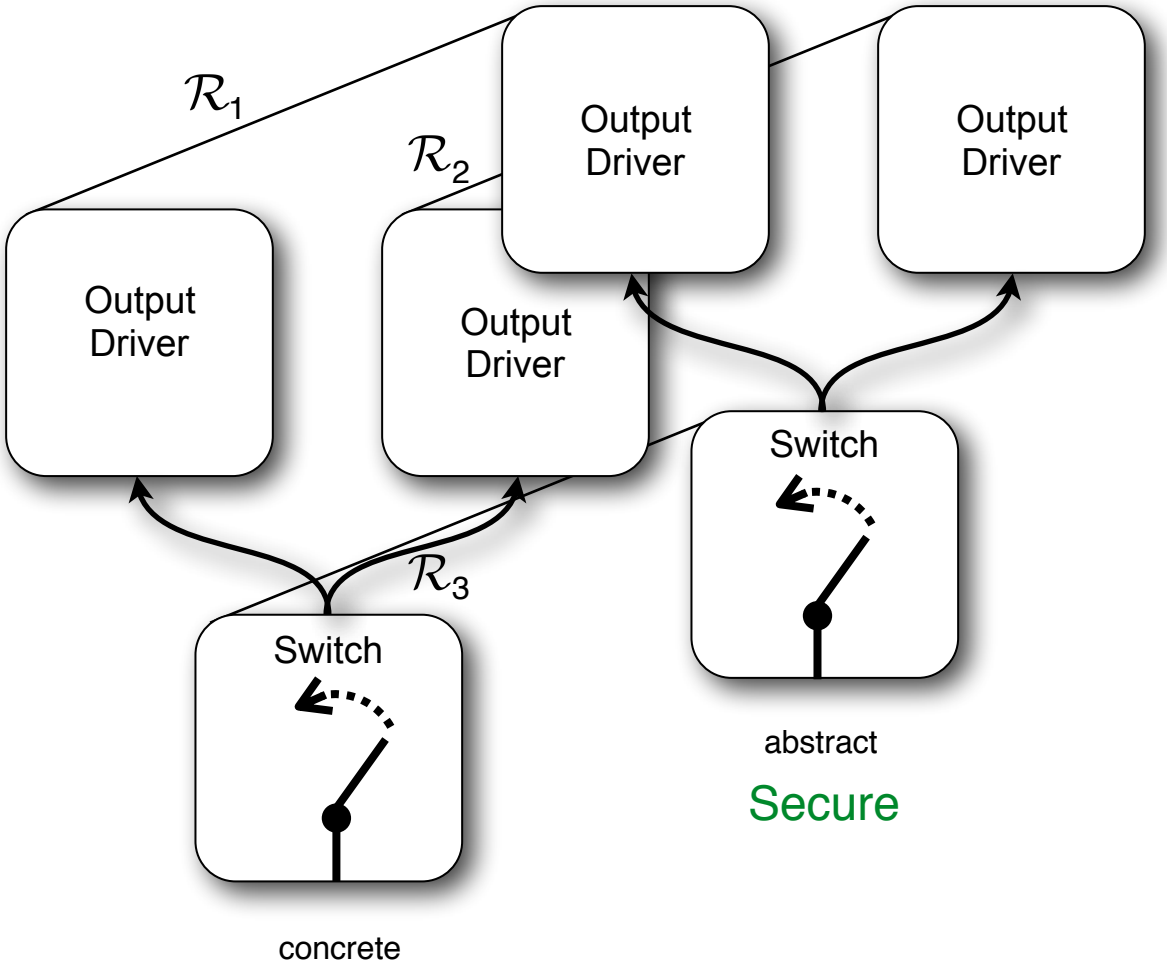
Secure Componentwise Refinement



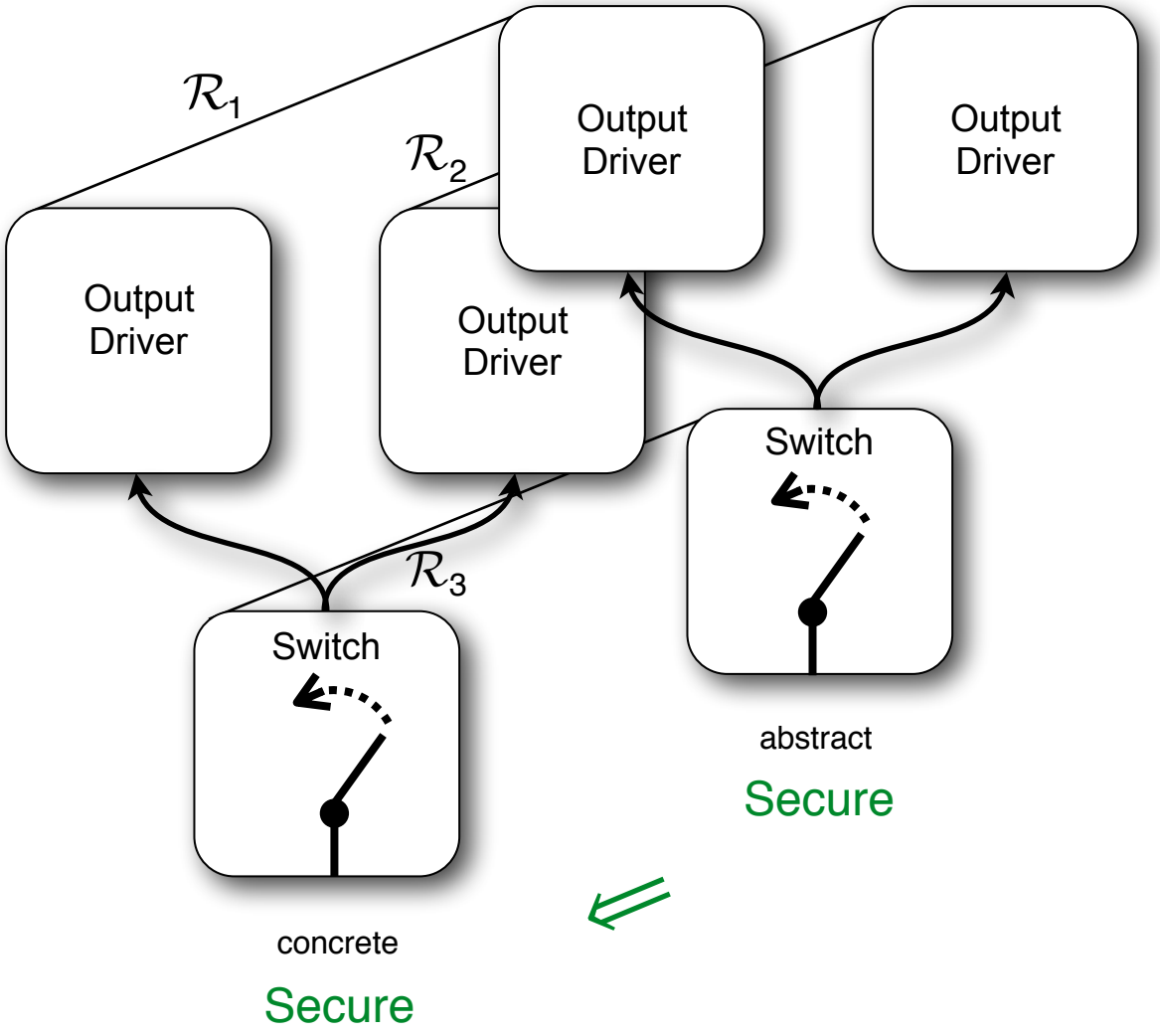
Secure Componentwise Refinement



Secure Componentwise Refinement



Secure Componentwise Refinement



Component Refinement

```
skip   /* control +=_m AsmNoW */;
skip   /* temp +=_m AsmNoRW */;
temp := buffer;
if control == 0 then
  low-var := temp
else
  high-var := temp
endif;
temp := 0;
skip   /* temp -=_m AsmNoRW */
```

```
skip   /* control_C +=_m AsmNoW */;
skip   /* temp_C +=_m AsmNoRW */;
temp_C := buffer_C;
reg_C := control_C;
if reg_C == 0 then
  low-var_C := temp_C
else
  high-var_C := temp_C
endif;
temp_C := 0;
skip   /* temp_C -=_m AsmNoRW */
```

Component Refinement

Original Program

```
skip   /* control +=_m AsmNoW */;
skip   /* temp +=_m AsmNoRW */;
temp := buffer;
if control == 0 then
  low-var := temp
else
  high-var := temp
endif;
temp := 0;
skip   /* temp -=_m AsmNoRW */
```

```
skip   /* control_C +=_m AsmNoW */;
skip   /* temp_C +=_m AsmNoRW */;
temp_C := buffer_C;
reg_C := control_C;
if reg_C == 0 then
  low-var_C := temp_C
else
  high-var_C := temp_C
endif;
temp_C := 0;
skip   /* temp_C -=_m AsmNoRW */
```

Component Refinement

Original Program

```
skip   /* control +=_m AsmNoW */;
skip   /* temp +=_m AsmNoRW */;
temp := buffer;
if control == 0 then
  low-var := temp
else
  high-var := temp
endif;
temp := 0;
skip   /* temp -=_m AsmNoRW */
```

A Trivial Refinement

```
skip   /* control_C +=_m AsmNoW */;
skip   /* temp_C +=_m AsmNoRW */;
temp_C := buffer_C;
reg_C := control_C;
if reg_C == 0 then
  low-var_C := temp_C
else
  high-var_C := temp_C
endif;
temp_C := 0;
skip   /* temp_C -=_m AsmNoRW */
```

Component Refinement

concrete variables implementing abstract ones

Original Program

```
skip  /* control +=_m AsmNoW */;
skip  /* temp +=_m AsmNoRW */;
temp := buffer;
if control == 0 then
  low-var := temp
else
  high-var := temp
endif;
temp := 0;
skip  /* temp -=_m AsmNoRW */
```

A Trivial Refinement

```
skip  /* control_C +=_m AsmNoW */;
skip  /* temp_C +=_m AsmNoRW */;
temp_C := buffer_C;
reg_C := control_C;
if reg_C == 0 then
  low-var_C := temp_C
else
  high-var_C := temp_C
endif;
temp_C := 0;
skip  /* temp_C -=_m AsmNoRW */
```

Component Refinement

Original Program

```
skip   /* control +=_m AsmNoW */;
skip   /* temp +=_m AsmNoRW */;
temp := buffer;
if control == 0 then
  low-var := temp
else
  high-var := temp
endif;
temp := 0;
skip   /* temp -=_m AsmNoRW */
```

concrete variables implementing abstract ones

concrete variables with no abstract counterpart

A Trivial Refinement

```
skip   /* control_C +=_m AsmNoW */;
skip   /* temp_C +=_m AsmNoRW */;
temp_C := buffer_C;
reg_C := control_C;
if reg_C == 0 then
  low-var_C := temp_C
else
  high-var_C := temp_C
endif;
temp_C := 0;
skip   /* temp_C -=_m AsmNoRW */
```

Component Refinement

Original Program

```
skip   /* control +=_m AsmNoW */;
skip   /* temp +=_m AsmNoRW */;
temp := buffer;
if control == 0 then
  low-var := temp
else
  high-var := temp
endif;
temp := 0;
skip   /* temp -=_m AsmNoRW */
```

A Trivial Refinement

```
skip   /* control_C +=_m AsmNoW */;
skip   /* temp_C +=_m AsmNoRW */;
temp_C := buffer_C;
reg_C := control_C;
if reg_C == 0 then
  low-var_C := temp_C
else
  high-var_C := temp_C
endif;
temp_C := 0;
skip   /* temp_C -=_m AsmNoRW */
```

Component Refinement

Original Program

```
skip   /* control +=_m AsmNoW */;
skip   /* temp +=_m AsmNoRW */;
temp := buffer;
if control == 0 then
  low-var := temp
else
  high-var := temp
endif;
temp := 0;
skip   /* temp -=_m AsmNoRW */
```

A Trivial Refinement

```
skip   /* control_C +=_m AsmNoW */;
skip   /* temp_C +=_m AsmNoRW */;
temp_C := buffer_C;
reg_C := control_C;
if reg_C == 0 then
  low-var_C := temp_C
else
  high-var_C := temp_C
endif;
temp_C := 0;
skip   /* temp_C -=_m AsmNoRW */
```

Component Refinement

Original Program

```
skip   /* control +=_m AsmNoW */;
skip   /* temp +=_m AsmNoRW */;
temp := buffer;
if control == 0 then
  low-var := temp
else
  high-var := temp
endif;
temp := 0;
skip   /* temp -=_m AsmNoRW */
```

A Trivial Refinement

```
skip   /* control_C +=_m AsmNoW */;
skip   /* temp_C +=_m AsmNoRW */;
temp_C := buffer_C;
reg_C := control_C;
if reg_C == 0 then
  low-var_C := temp_C
else
  high-var_C := temp_C
endif;
temp_C := 0;
skip   /* temp_C -=_m AsmNoRW */
```

Abstract Program Security: Bisimulation

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

Abstract Program Security: Bisimulation

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

Abstract Program Security: Bisimulation

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

$\mathcal{B}$

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

Abstract Program Security: Bisimulation

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

$\mathcal{B}$

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

Important:

$\mathcal{B}$ must guarantee that the two memories are always Low-equivalent

Abstract Program Security: Bisimulation

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

$\mathcal{B}$

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

Important:

$\mathcal{B}$ must guarantee that the two memories are always Low-equivalent

Abstract Program Security: Bisimulation

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

$\mathcal{B}$

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

Important:

$\mathcal{B}$ must guarantee that the two memories are always Low-equivalent

Abstract Program Security: Bisimulation

```
if (h) {
     h = 0;
     l = 1;
} else {
     skip;
     l = 1;
}
```

$\mathcal{B}$

```
if (h) {
     h = 0;
     l = 1;
} else {
     skip;
     l = 1;
}
```

Important:

$\mathcal{B}$ must guarantee that the two memories are always Low-equivalent

Abstract Program Security: Bisimulation

```
if (h) {
     h = 0;
     l = 1;
} else {
     skip;
     l = 1;
}
```

$\mathcal{B}$

```
if (h) {
     h = 0;
     l = 1;
} else {
     skip;
     l = 1;
}
```

(the soundness proof for the security logic constructs an appropriate $\mathcal{B}$)

Important:

$\mathcal{B}$ must guarantee that the two memories are always Low-equivalent

Refining High Conditionals

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

Refining High Conditionals

A Secure Program

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

Refining High Conditionals

A Secure Program

branches need to do the same Low things at the same time

```
if (h) {
     h = 0;
     l = 1;
} else {
     skip;
     l = 1;
}
```

Refining High Conditionals

branches need to do the same Low things at the same time

A Secure Program

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

A Trivial Refinement

```
if (h) {
    h = 0;
    l = 1;
} else {
    l = 1;
}
```

Refining High Conditionals

branches need to do the same Low things at the same time

A Secure Program

```
if (h) {
     h = 0;
     l = 1;
} else {
     skip;
     l = 1;
}
```

A Trivial Refinement

```
if (h) {
     h = 0;
     l = 1;
} else {
     l = 1;
}
```

insecure

Refining High Conditionals

branches need to do the same Low things at the same time

A Secure Program

```
if (h) {
    h = 0;
    l = 1;
} else {
    skip;
    l = 1;
}
```

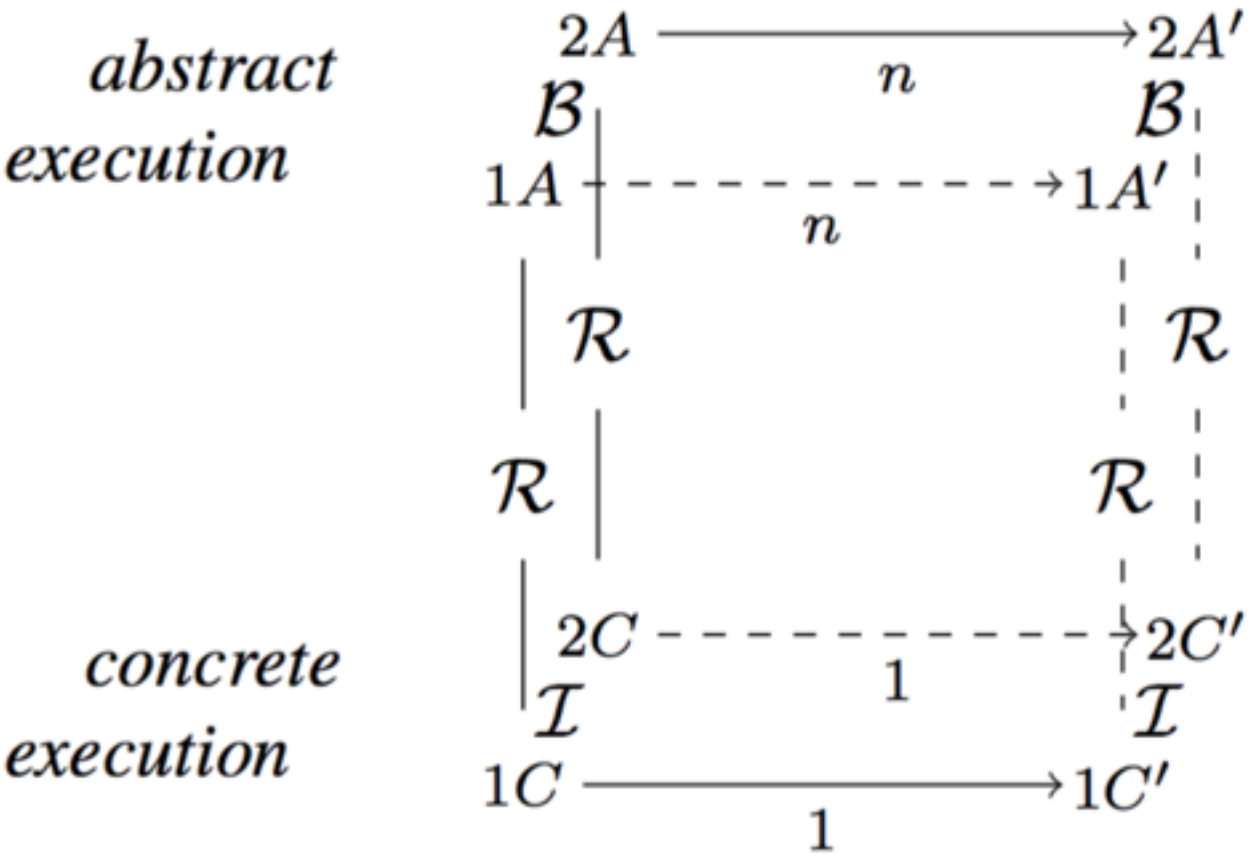
A Trivial Refinement

```
if (h) {
    h = 0;
    l = 1;
} else {
    l = 1;
}
```

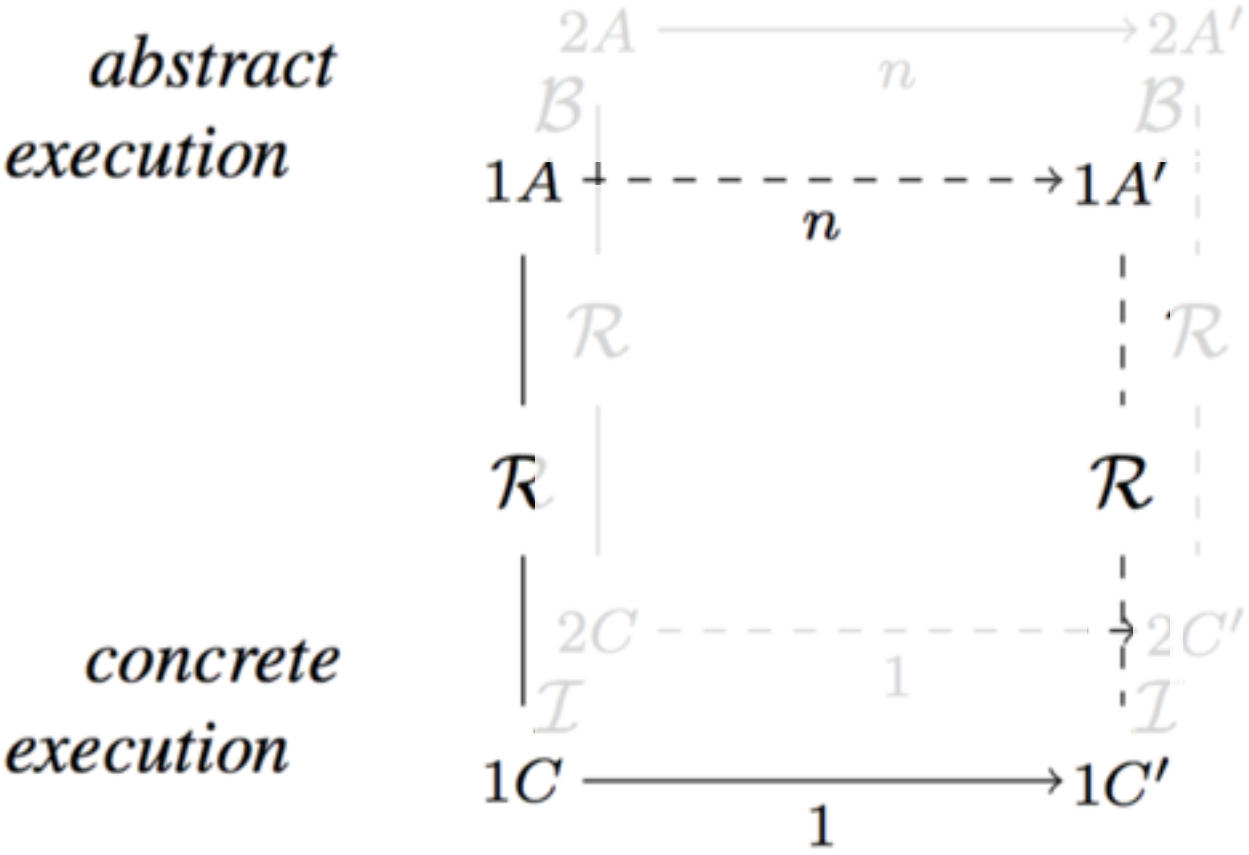
insecure

Need to make sure that the concrete executions stay in sync.

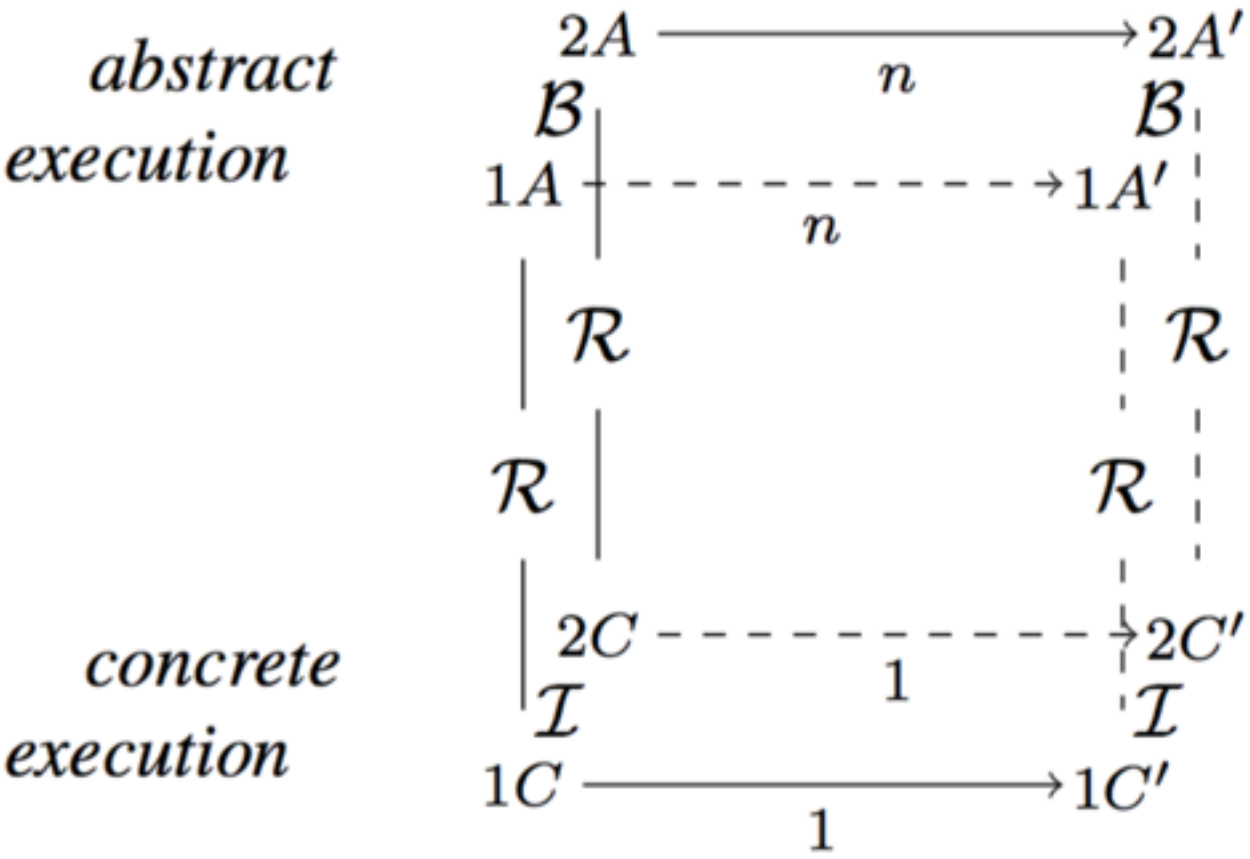
Secure Refinement via Coupling



Secure Refinement via Coupling



Secure Refinement via Coupling



Coupling Invariant: $\mathcal{I}$

```
skip   /* h +=_m AsmNoW */;
skip   /* y +=_m AsmNoW */;
skip   /* z +=_m AsmNoW */;
y := 0;
z := 0;
x := y;
if h != 0 then
  x := y
else
  x := y + z
endif
```

Coupling Invariant: $\mathcal{I}$

```
skip   /* h_C +=_m AsmNoW */;
skip   /* y_C +=_m AsmNoW */;
skip   /* z_C +=_m AsmNoW */;
y_C := 0;
z_C := 0;
x_C := y_C;
reg3_C := h_C;
if reg3_C != 0 then
  skip;
  skip;
  reg0_C := y_C;
  x_C := reg0_C
else
  reg1_C := y_C;
  reg2_C := z_C;
  reg0_C := reg1_C + reg2_C;
  x_C := reg0_C
endif
```

Coupling Invariant: $\mathcal{I}$

```
skip   /* h_C +=_m AsmNoW */;
skip   /* y_C +=_m AsmNoW */;
skip   /* z_C +=_m AsmNoW */;
y_C := 0;
z_C := 0;
x_C := y_C;
reg3_C := h_C;
if reg3_C != 0 then
  skip;
  skip;
  reg0_C := y_C;
  x_C := reg0_C
else
  reg1_C := y_C;
  reg2_C := z_C;
  reg0_C := reg1_C + reg2_C;
  x_C := reg0_C
endif
```

```
skip   /* h_C +=_m AsmNoW */;
skip   /* y_C +=_m AsmNoW */;
skip   /* z_C +=_m AsmNoW */;
y_C := 0;
z_C := 0;
x_C := y_C;
reg3_C := h_C;
if reg3_C != 0 then
  skip;
  skip;
  reg0_C := y_C;
  x_C := reg0_C
else
  reg1_C := y_C;
  reg2_C := z_C;
  reg0_C := reg1_C + reg2_C;
  x_C := reg0_C
endif
```

Coupling Invariant: $\mathcal{I}$

```
skip  /* h_C +=_m AsmNoW */;
skip  /* y_C +=_m AsmNoW */;
skip  /* z_C +=_m AsmNoW */;
y_C := 0;
z_C := 0;
x_C := y_C;
reg3_C := h_C;
if reg3_C != 0 then
  skip;
  skip;
  reg0_C := y_C;
  x_C := reg0_C
else
  reg1_C := y_C;
  reg2_C := z_C;
  reg0_C := reg1_C + reg2_C;
  x_C := reg0_C
endif
```

```
skip  /* h_C +=_m AsmNoW */;
skip  /* y_C +=_m AsmNoW */;
skip  /* z_C +=_m AsmNoW */;
y_C := 0;
z_C := 0;
x_C := y_C;
reg3_C := h_C;
if reg3_C != 0 then
  skip;
  skip;
  reg0_C := y_C;
  x_C := reg0_C
else
  reg1_C := y_C;
  reg2_C := z_C;
  reg0_C := reg1_C + reg2_C;
  x_C := reg0_C
endif
```

Coupling Invariant: $\mathcal{I}$

```
skip  /* h_C +=_m AsmNoW */;
skip  /* y_C +=_m AsmNoW */;
skip  /* z_C +=_m AsmNoW */;
y_C := 0;
z_C := 0;
x_C := y_C;
reg3_C := h_C;
if reg3_C != 0 then
  skip;
  skip;
  reg0_C := y_C;
  x_C := reg0_C
else
  reg1_C := y_C;
  reg2_C := z_C;
  reg0_C := reg1_C + reg2_C;
  x_C := reg0_C
endif
```

```
skip  /* h_C +=_m AsmNoW */;
skip  /* y_C +=_m AsmNoW */;
skip  /* z_C +=_m AsmNoW */;
y_C := 0;
z_C := 0;
x_C := y_C;
reg3_C := h_C;
if reg3_C != 0 then
  skip;
  skip;
  reg0_C := y_C;
  x_C := reg0_C
else
  reg1_C := y_C;
  reg2_C := z_C;
  reg0_C := reg1_C + reg2_C;
  x_C := reg0_C
endif
```

Coupling Invariant: $\mathcal{I}$

```
skip   /* h_C +=_m AsmNoW */;
skip   /* y_C +=_m AsmNoW */;
skip   /* z_C +=_m AsmNoW */;
y_C := 0;
z_C := 0;
x_C := y_C;
reg3_C := h_C;
if reg3_C != 0 then
  skip;
  skip;
  reg0_C := y_C;
  x_C := reg0_C
else
  reg1_C := y_C;
  reg2_C := z_C;
  reg0_C := reg1_C + reg2_C;
  x_C := reg0_C
endif
```

```
skip   /* h_C +=_m AsmNoW */;
skip   /* y_C +=_m AsmNoW */;
skip   /* z_C +=_m AsmNoW */;
y_C := 0;
z_C := 0;
x_C := y_C;
reg3_C := h_C;
if reg3_C != 0 then
  skip;
  skip;
  reg0_C := y_C;
  x_C := reg0_C
else
  reg1_C := y_C;
  reg2_C := z_C;
  reg0_C := reg1_C + reg2_C;
  x_C := reg0_C
endif
```

Coupling Invariant: $\mathcal{I}$

```
skip  /* h_C +=_m AsmNoW */;
skip  /* y_C +=_m AsmNoW */;
skip  /* z_C +=_m AsmNoW */;
y_C := 0;
z_C := 0;
x_C := y_C;
reg3_C := h_C;
if reg3_C != 0 then
  skip;
  skip;
  reg0_C := y_C;
  x_C := reg0_C
else
  reg1_C := y_C;
  reg2_C := z_C;
  reg0_C := reg1_C + reg2_C;
  x_C := reg0_C
endif
```

```
skip  /* h_C +=_m AsmNoW */;
skip  /* y_C +=_m AsmNoW */;
skip  /* z_C +=_m AsmNoW */;
y_C := 0;
z_C := 0;
x_C := y_C;
reg3_C := h_C;
if reg3_C != 0 then
  reg0_C := reg1_C + reg2_C;
  x_C := reg0_C
endif
```

Important:

$\mathcal{I}$ need not talk at all about memory contents being secure

Concurrency-Aware Refinement

$$x := y + z$$

Concurrency-Aware Refinement

$$x := y + z$$

Source language semantics evaluates y + z atomically

Concurrency-Aware Refinement

$$x := y + z$$

Source language semantics evaluates y + z atomically

Imagine we compile to a simple stack machine:

```
PUSH y;
PUSH z;
ADD;
POP x;
```

Concurrency-Aware Refinement

$$x := y + z$$

Source language semantics evaluates y + z atomically

Imagine we compile to a simple stack machine:

```
PUSH y;
PUSH z;
ADD;
POP x;
```

This compilation is valid only if no other thread concurrently modifies y and z

Concurrency-Aware Refinement

$$x := y + z$$

Source language semantics evaluates y + z atomically

Imagine we compile to a simple stack machine:

```
PUSH y;
PUSH z;
ADD;
POP x;
```

This compilation is valid only if no other thread concurrently modifies y and z

Idea: require y and z to be assumes not-writable by others

Concurrency-Aware Refinement

$$x := y + z$$

Source language semantics evaluates y + z atomically

Imagine we compile to a simple stack machine:

```
PUSH y;
PUSH z;
ADD;
POP x;
```

This compilation is valid only if no other thread concurrently modifies y and z

Idea: require y and z to be assumes not-writable by others

i.e. that the refinement relation is preserved by potential actions of other threads

Concurrency-Aware Refinement

$$x := y + z$$

Source language semantics evaluates y + z atomically

Imagine we compile to a simple stack machine:

```
PUSH y;
PUSH z;
ADD;
POP x;
```

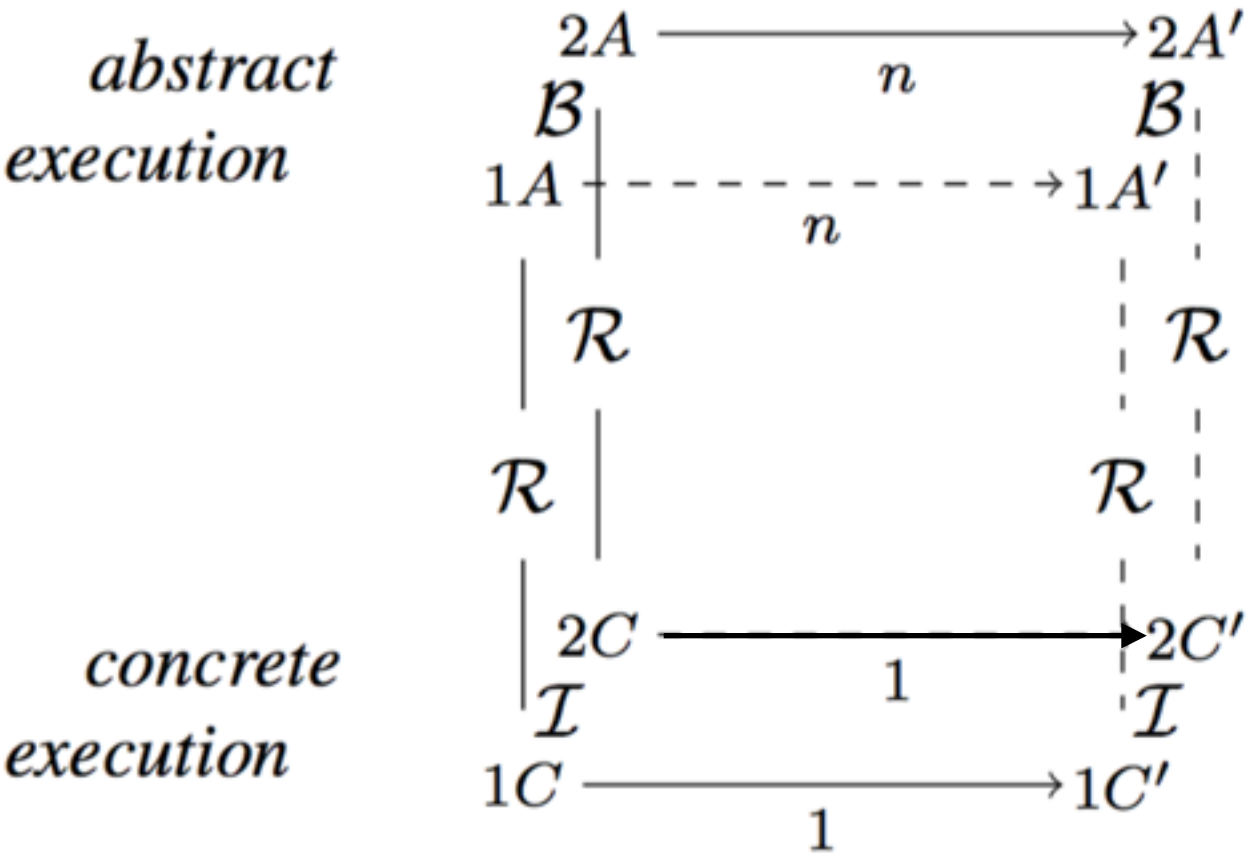
This compilation is valid only if no other thread concurrently modifies y and z

Idea: require y and z to be assumes not-writable by others

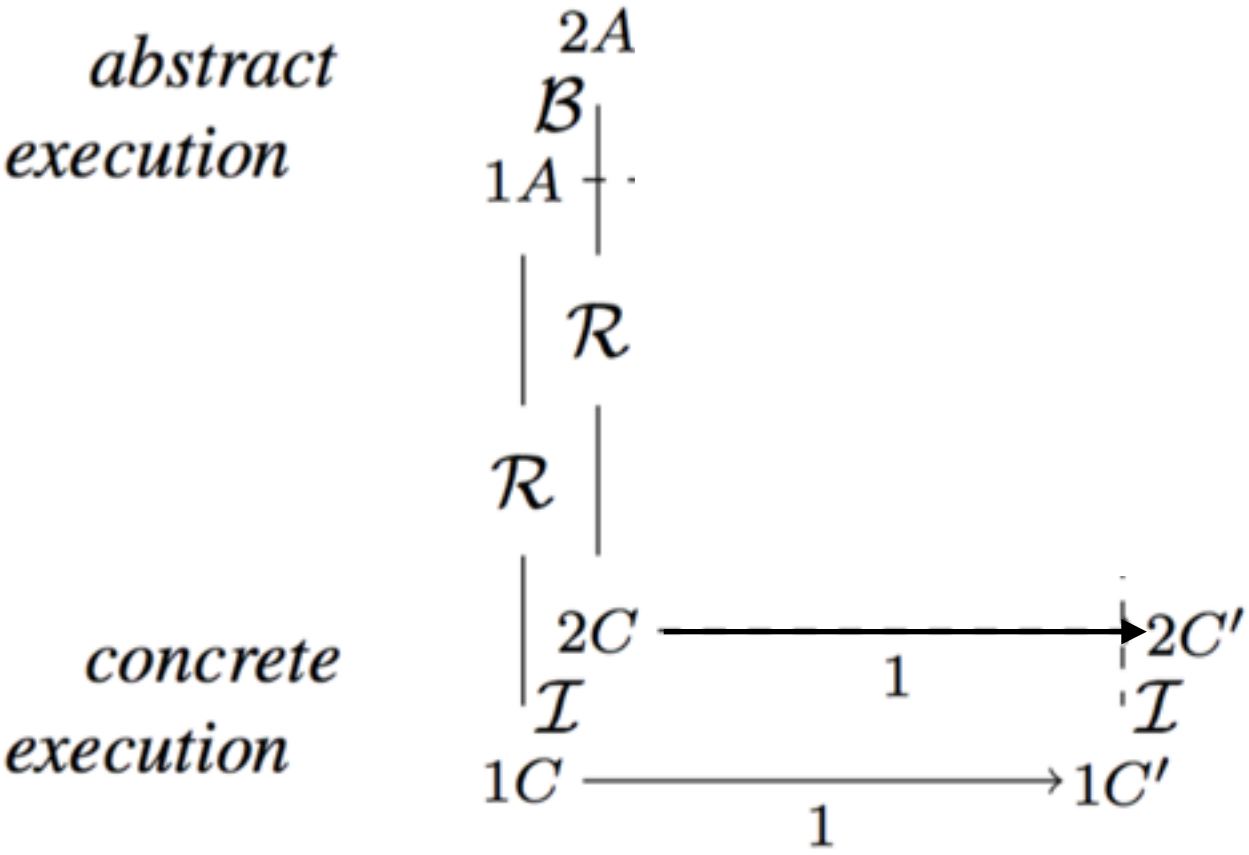
(see Liang et al., TOPLAS 2014)

i.e. that the refinement relation is preserved by potential actions of other threads

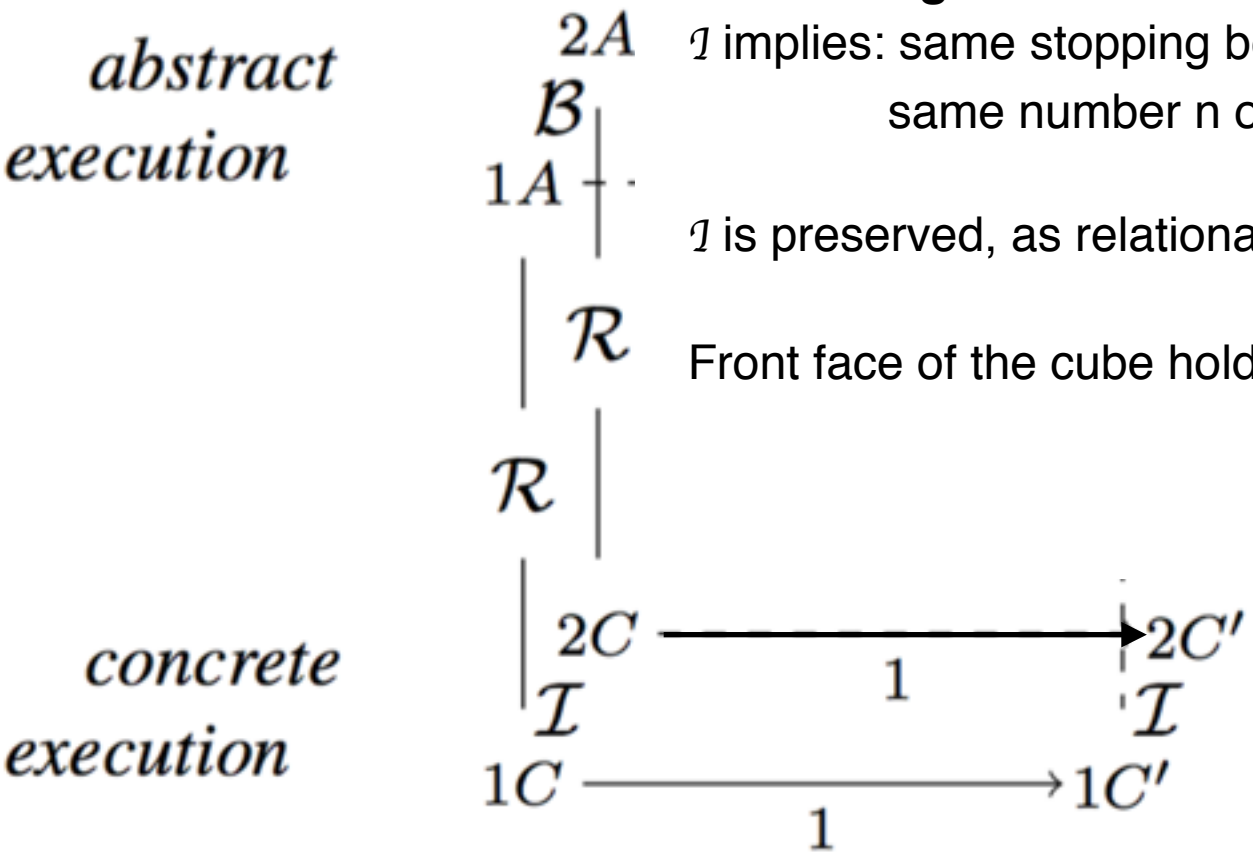
Leveraging Determinism



Leveraging Determinism



Leveraging Determinism



Proof Obligations:

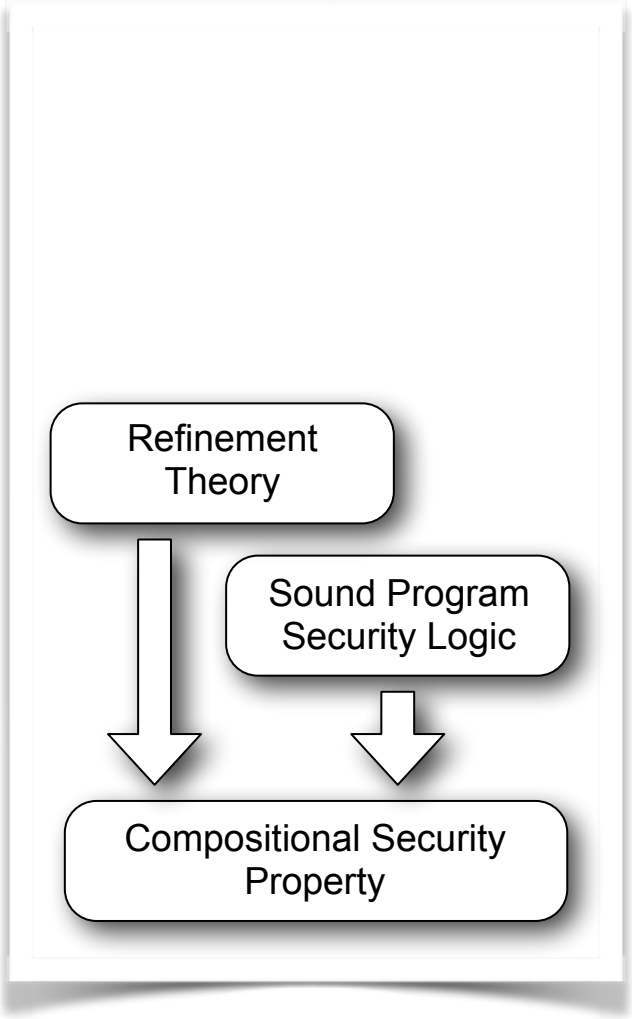
$\mathcal{I}$ implies: same stopping behaviour
same number n of abstract steps

$\mathcal{I}$ is preserved, as relational invariant

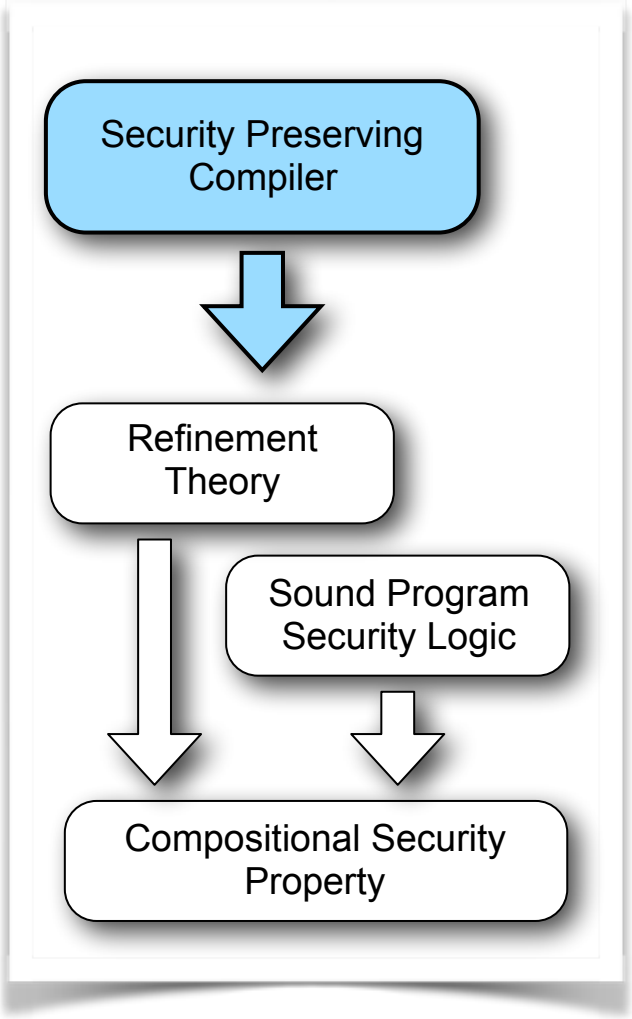
Front face of the cube holds

SECURITY-PRESERVING COMPILATION

SECURITY-PRESERVING COMPILATION



SECURITY-PRESERVING COMPILATION

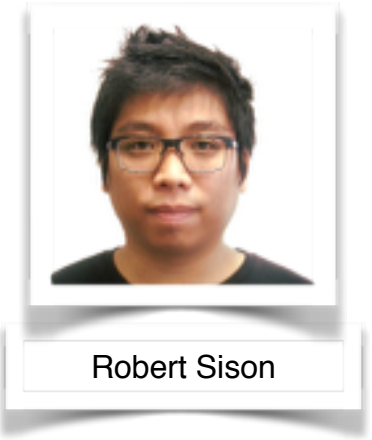


Case Study: Compilation to Idealised RISC

instr ::= **lock** l
| **release** l
| **load** reg x
| **store** x reg
| **Jump** label
| **Jz** label
| **Noop**
| **MoveK** reg w
| **Mov** reg reg
| **Binop** op reg reg

Case Study: Compilation to Idealised RISC

instr ::= **lock** l
release l
load reg x
store x reg
Jump label
Jz label
Noop
MoveK reg w
Mov reg reg
Binop op reg reg


Robert Sison

Current Status

- Proof finished for source programs that are secure and never branch on High data
  - This is the simple case where the compiler doesn't have to do anything except compile the program properly to preserve noninterference
- Sufficient for the CDDC model we have verified
- First example compiler for shared-memory concurrent programs proved to preserve noninterference
- Next steps: High conditionals, weak memory, …

CONCLUSION

CONCLUSION

Moving from verified secure kernels to verified secure systems:

CONCLUSION

Moving from verified secure kernels to verified secure systems:

- concurrency ~> compositionality ~>
  timing-sensitivity, assume-guarantee reasoning
- dynamic policies (value-dependence)
- compositional secure refinement ~>
  coupling invariants for preservation of timing-sensitive security

CONCLUSION

Moving from verified secure kernels to verified secure systems:

- concurrency ~> compositionality ~>
  timing-sensitivity, assume-guarantee reasoning
- dynamic policies (value-dependence)
- compositional secure refinement ~>
  coupling invariants for preservation of timing-sensitive security

Still lots more to be done.

Thank You


toby.murray@unimelb.edu.au


http://people.eng.unimelb.edu.au/tobym

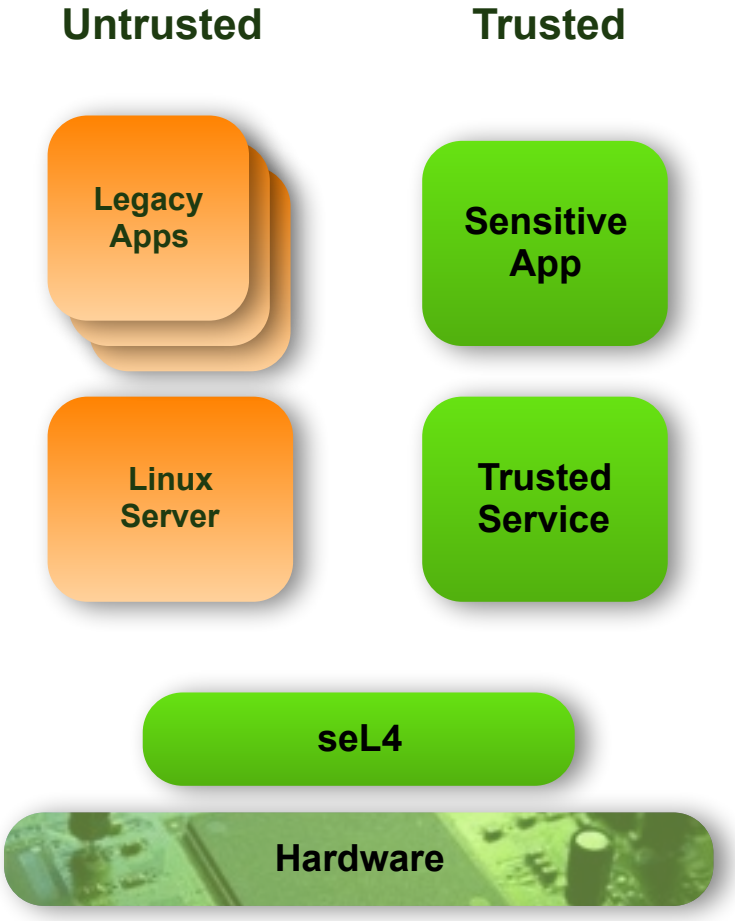

@tobycmurray

What about … ?

and various other **intra-device** timing channels?

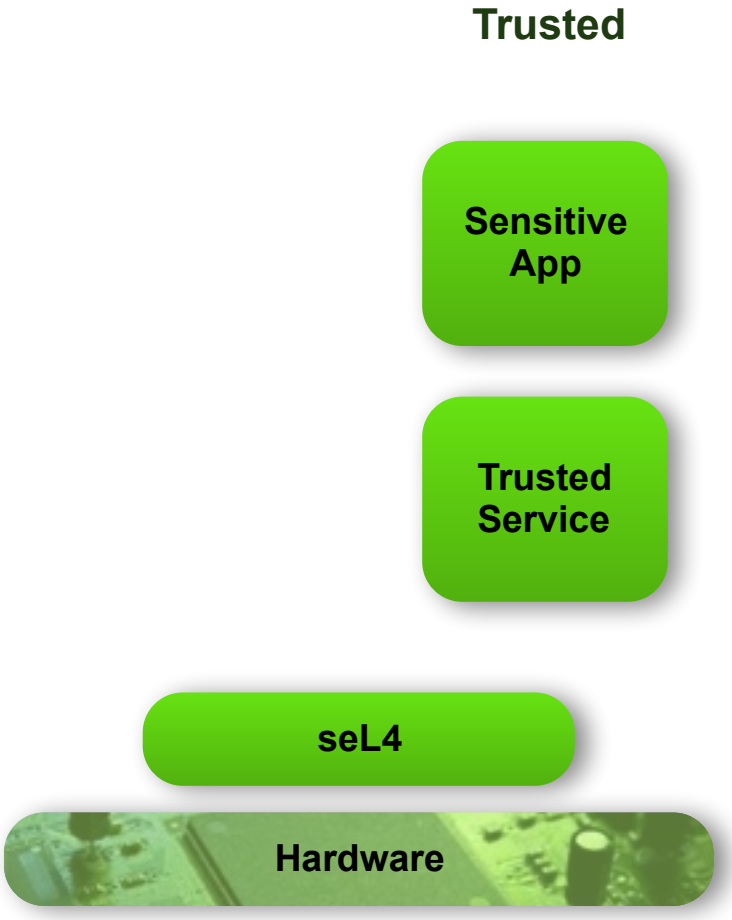
What about … ?

and various other **intra-device** timing channels?



What about … ?

and various other **intra-device** timing channels?



Minimum Requirements

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

```
print(l);
```

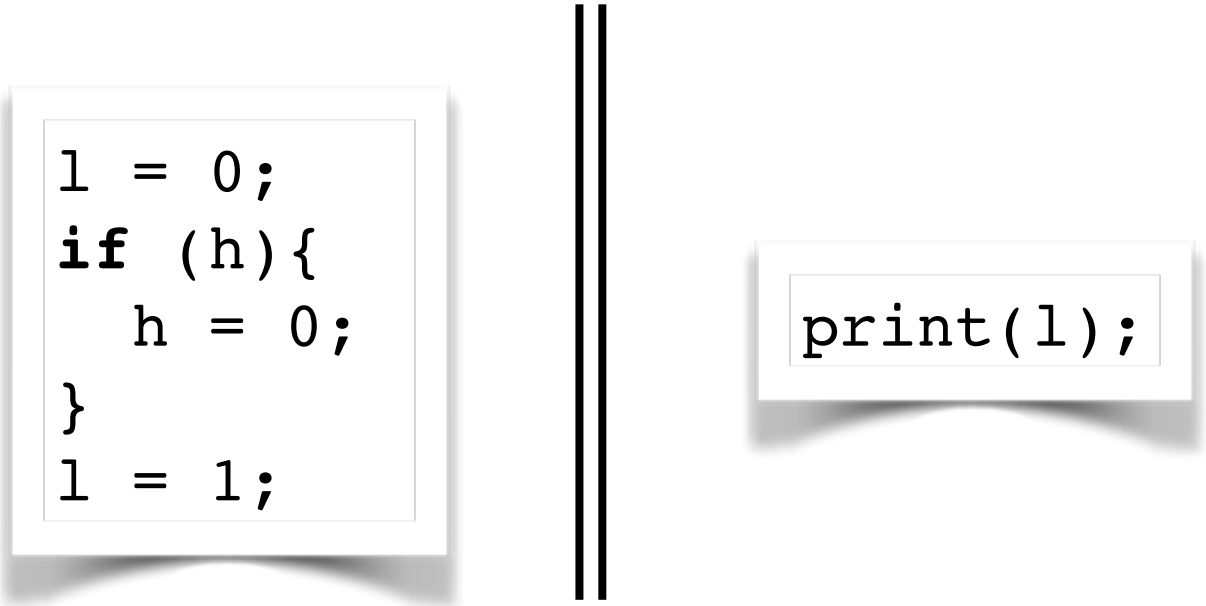
Minimum Requirements

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

```
print(l);
```

Secure?

Minimum Requirements



Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 0)

pc

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

≈

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 0)

pc

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

≡

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 0)

pc

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

≡

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 0)

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

pc

≈

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 0)

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

pc

≈

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 0)

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

pc

≈

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 0)

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

pc

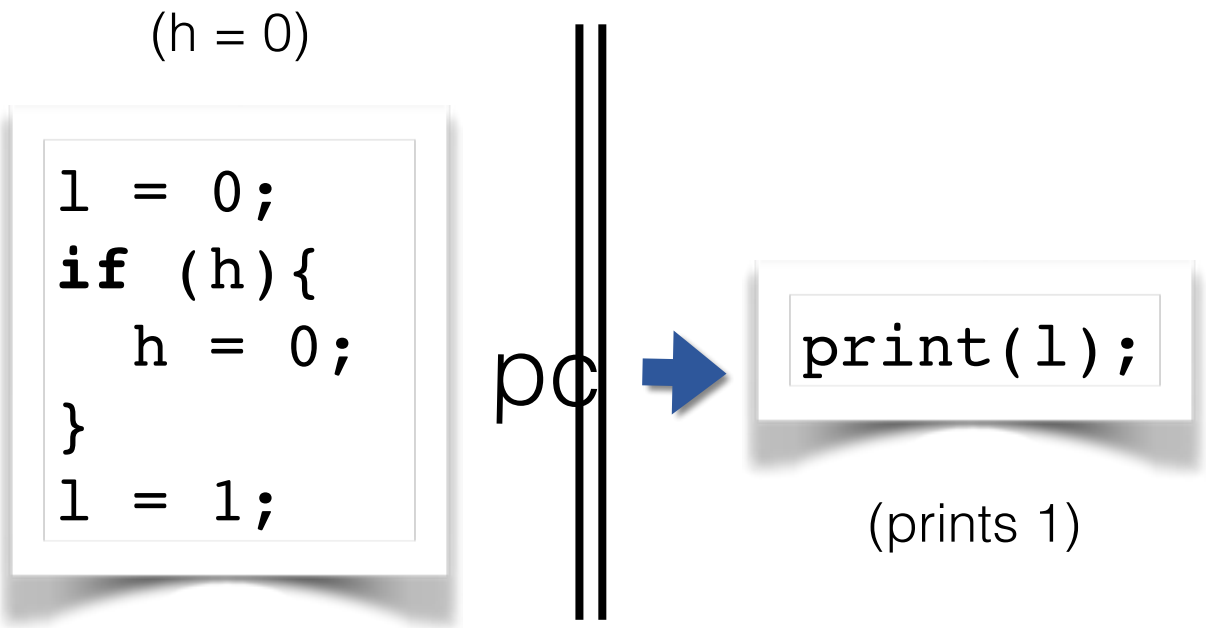
≈

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements



Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 1)

pc

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

≈

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 1)

pc

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

≈

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 1)

pc

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

≡

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 1)

pc

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

≈

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 1)

pc

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

≈

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 1)

pc

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

≈

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements

(h = 1)

```
l = 0;
if (h){
  h = 0;
}
l = 1;
```

pc

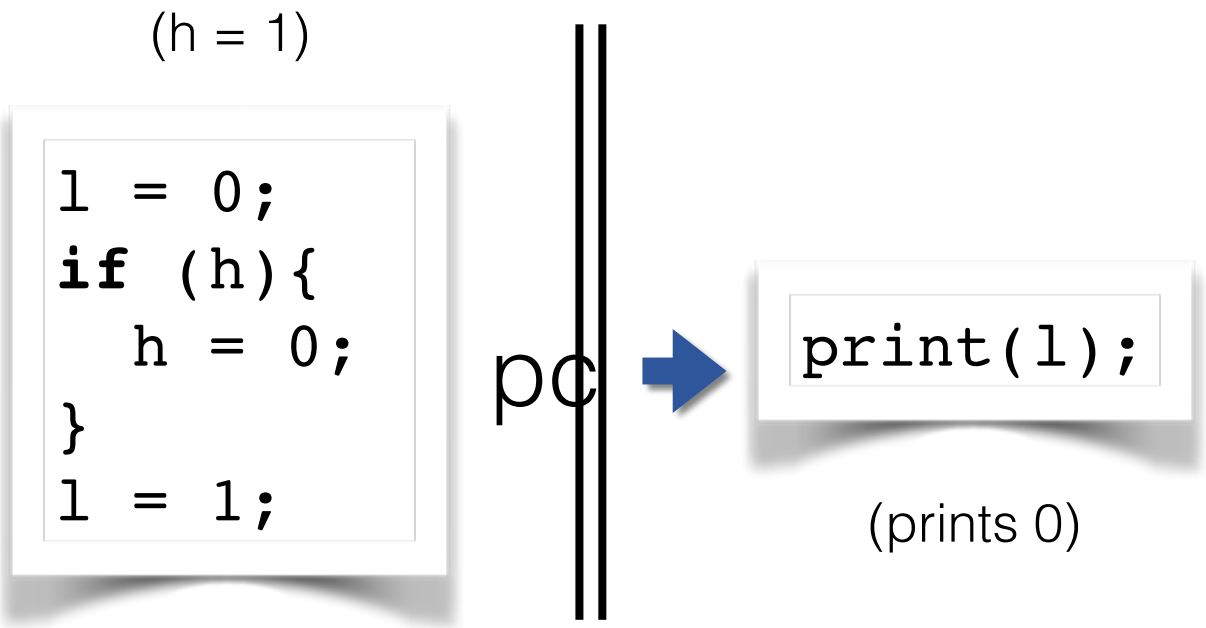
||

```
print(l);
```

Secure?

Timing-Insensitive Security:

Minimum Requirements



Secure?

Timing-Insensitive Security: