Proving confidentiality and its preservation under compilation for mixed-sensitivity concurrent programs

PhD thesis (2016-2020), UNSW

https://doi.org/fjmt

SSS^2, 25 January 2023

Dr Robert Sison

Research Fellow, The University of Melbourne

Visiting Fellow, UNSW Sydney

Proving confidentiality and its preservation under compilation for mixed-sensitivity concurrent programs

PhD thesis (2016-2020), UNSW
https://doi.org/fjmt

SSS^2, 25 January 2023

Dr Robert Sison
Research Fellow, The University of Melbourne
Visiting Fellow, UNSW Sydney

Note: *Interactive theorem proving* (Isabelle)

Proving confidentiality and its preservation under compilation for mixed-sensitivity concurrent programs

PhD thesis (2016-2020), UNSW

https://doi.org/fjmt

SSS^2, 25 January 2023

Dr Robert Sison

Research Fellow, The University of Melbourne

Visiting Fellow, UNSW Sydney

Note: *Interactive theorem proving* (Isabelle)

Proving confidentiality:

The floor is lava

Proving confidentiality:

The floor is lava

“secret files can’t touch the lava” game

Proving confidentiality:

The floor is lava

“secret files can’t
touch the lava”
game

✗

Proving confidentiality:

“secret files can’t touch the lava” game

The floor is lava

in reality:
a hotel window;
the media

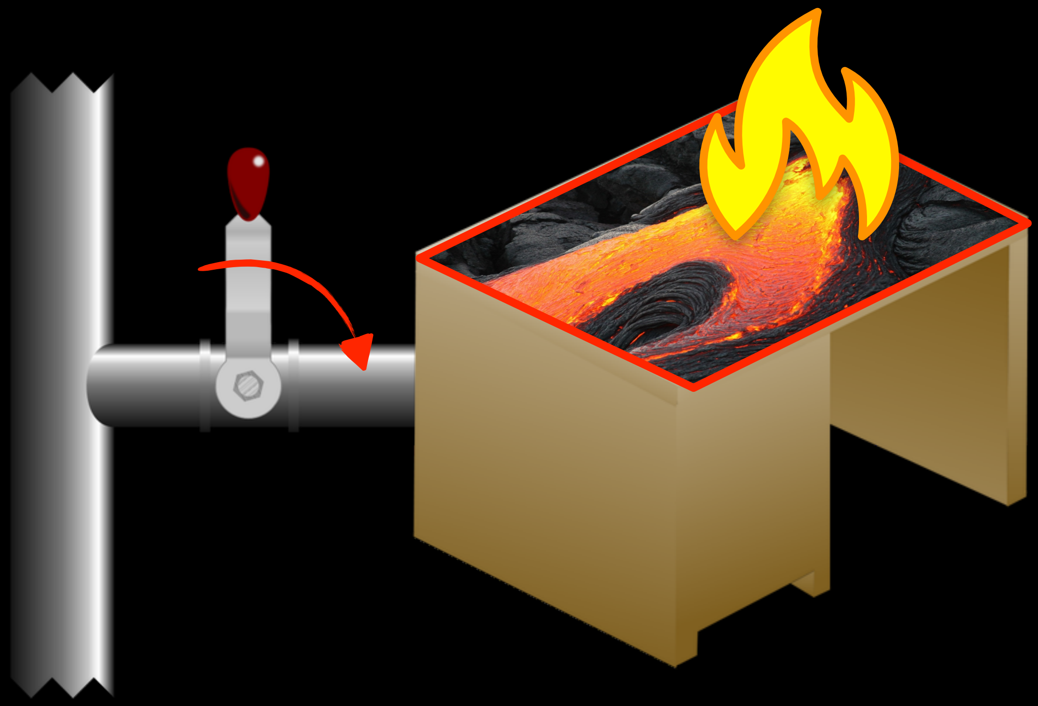
✗

Confidentiality in the face of scale

The *desk* is lava

Mixed-sensitivity reuse

“We have 2 customers and 1 desk”

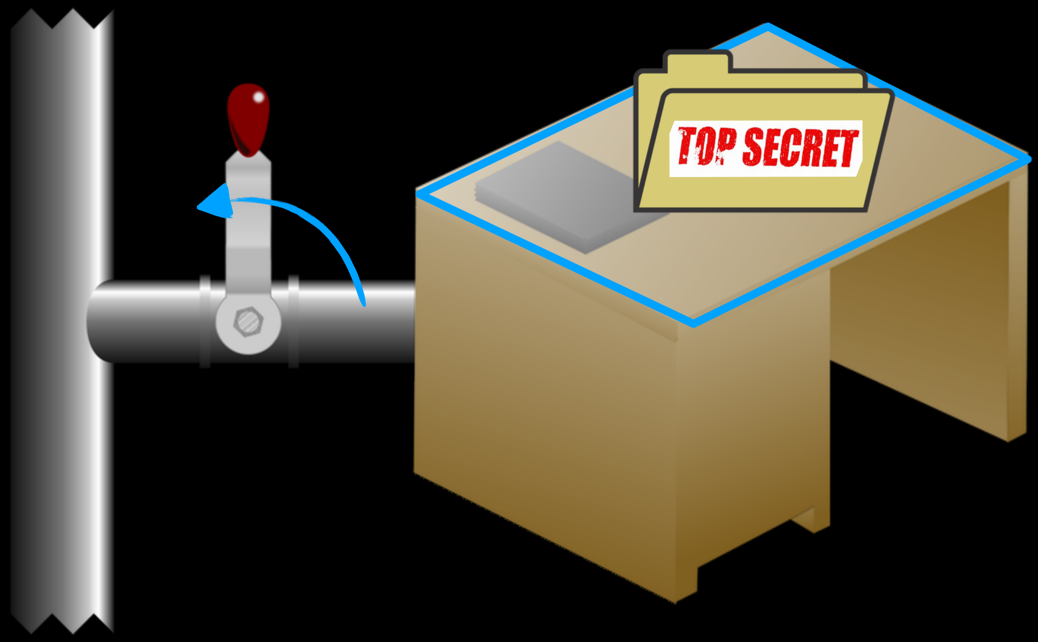


Confidentiality in the face of scale

The *desk* is lava

Mixed-sensitivity reuse

“We have 2 customers and 1 desk”



Confidentiality in the face of scale

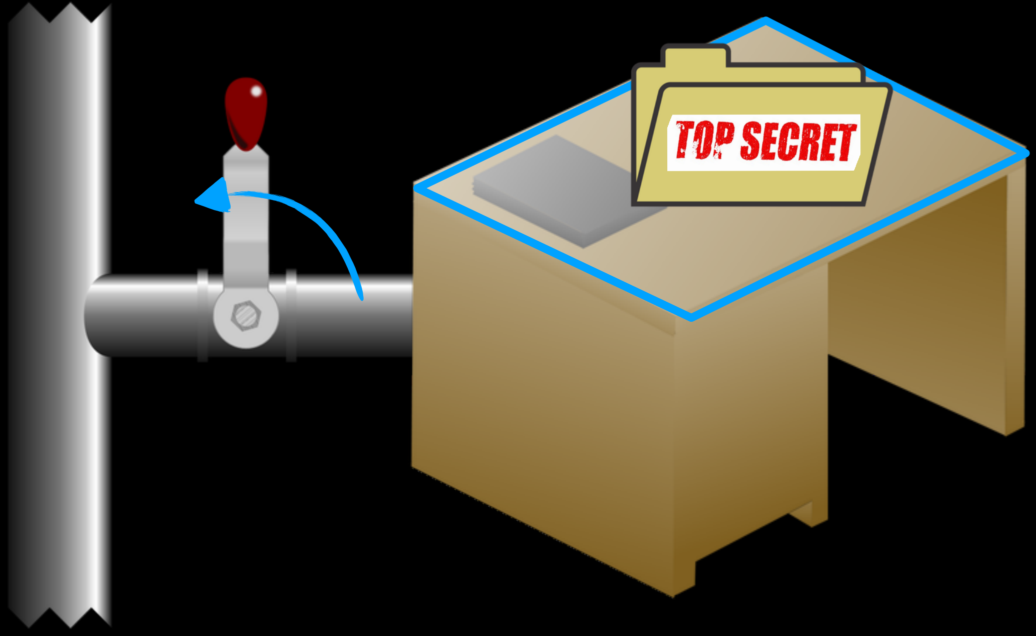
The *desk* is lava

Mixed-sensitivity reuse

“We have 2 customers and 1 desk”

Shared-memory concurrency

“15 of us work in this office”

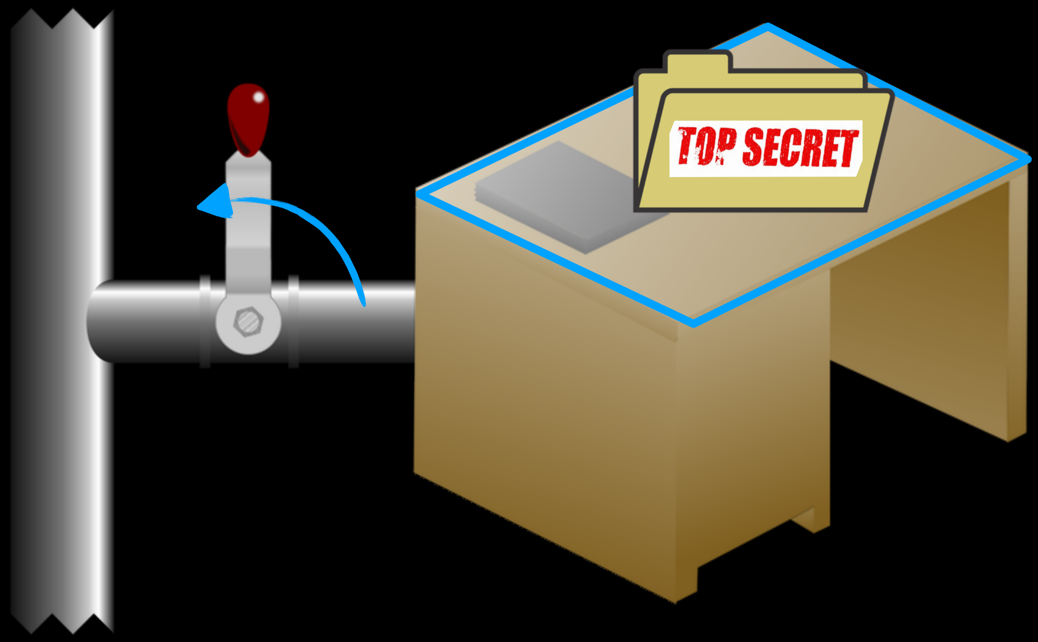


Confidentiality in the face of scale

The *desk* is lava

Mixed-sensitivity reuse

"We have 2 customers and 1 desk"



Shared-memory concurrency

"15 of us work in this office"

- Any of us might use the desk

Confidentiality in the face of scale

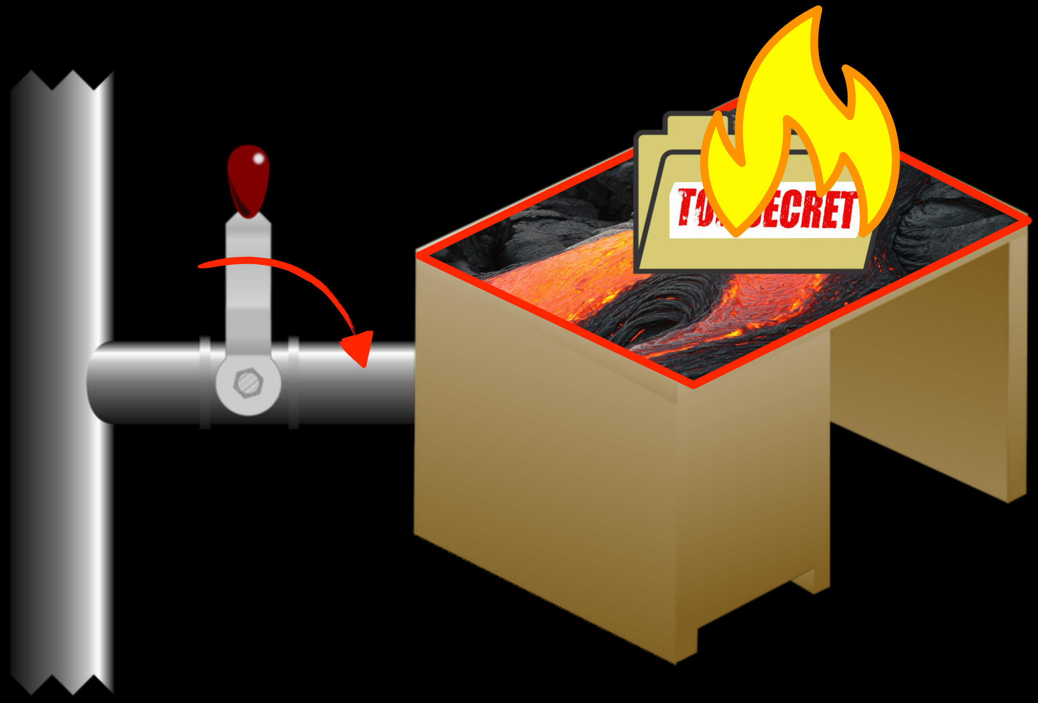
The *desk* is lava

Mixed-sensitivity reuse

“We have 2 customers and 1 desk”

Shared-memory concurrency

“15 of us work in this office”



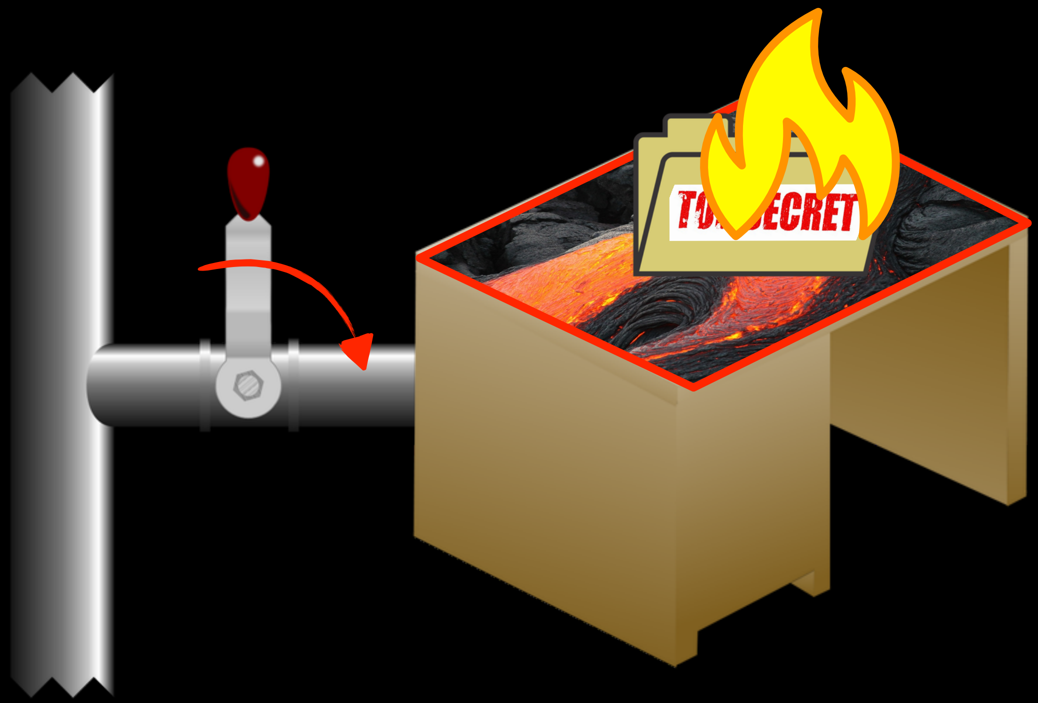
- Any of us might use the desk
- Any of us might touch the lever

Confidentiality in the face of scale

The *desk* is lava

Mixed-sensitivity reuse

“We have 2 customers and 1 desk”



Shared-memory concurrency

“15 of us work in this office”

- Any of us might use the desk
- Any of us might touch the lever

Mixed-sensitivity concurrent programs

Motivating use case

Beaumont, McCarthy, Murray
(ACSAC 2016)

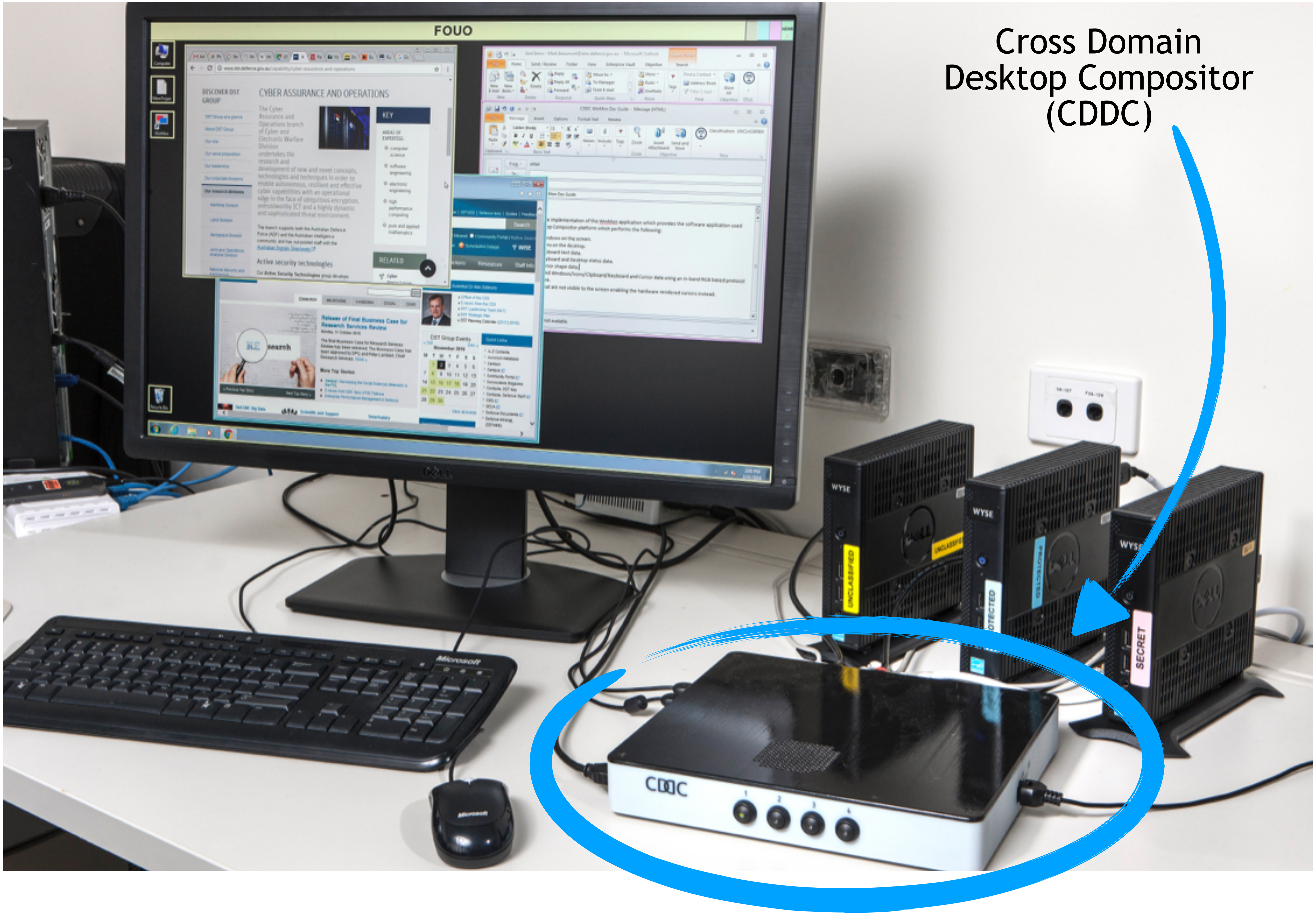
Cross Domain Desktop Compositor
(DSTG + Trustworthy Systems collaboration)

Finalist entry for 2021 Eureka Prize
(Outstanding Science in Safeguarding Australia)

Motivating use case

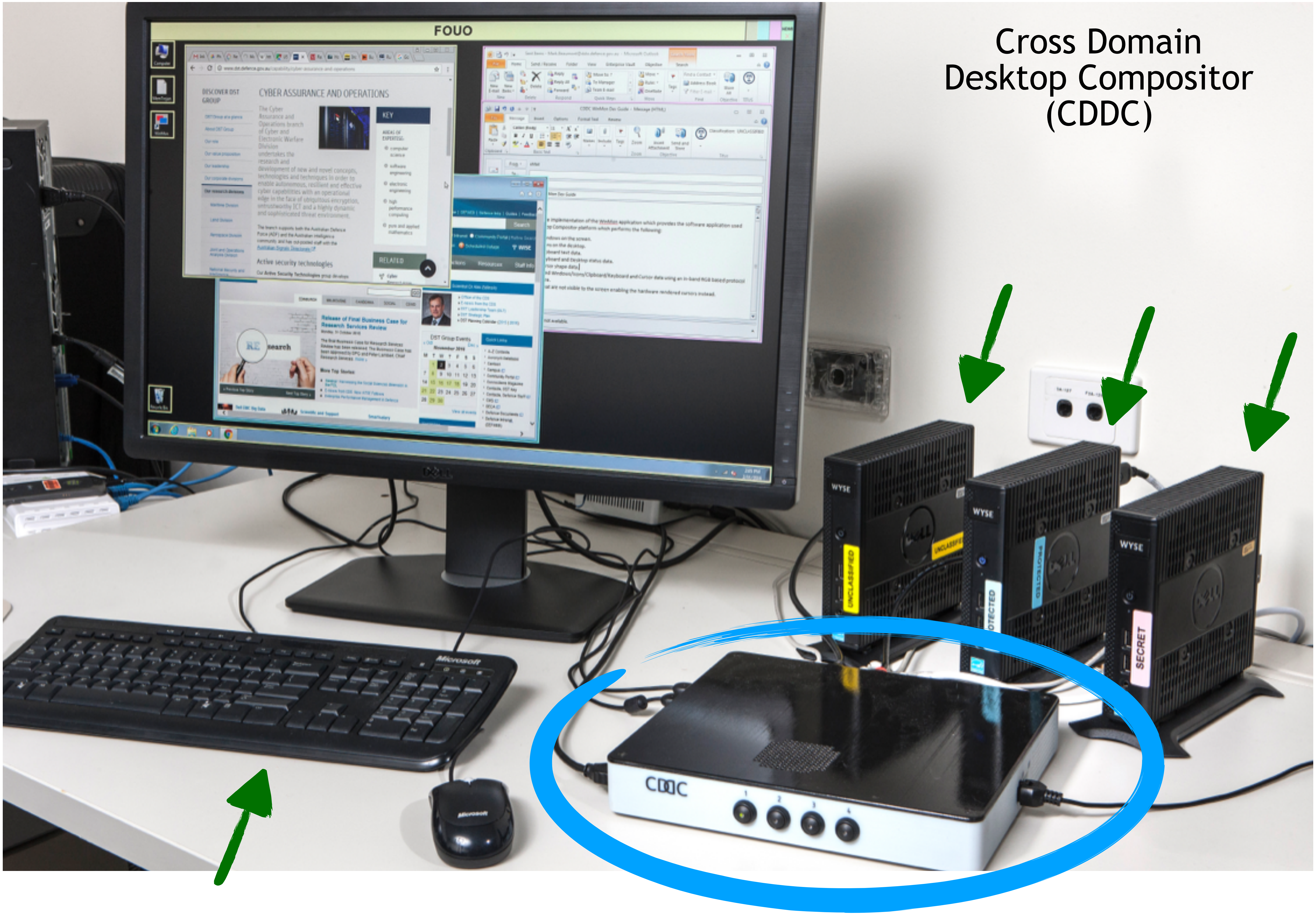
Motivating use case

TOP SECRET
PROTECTED
Unclassified
… 😑 , 💸

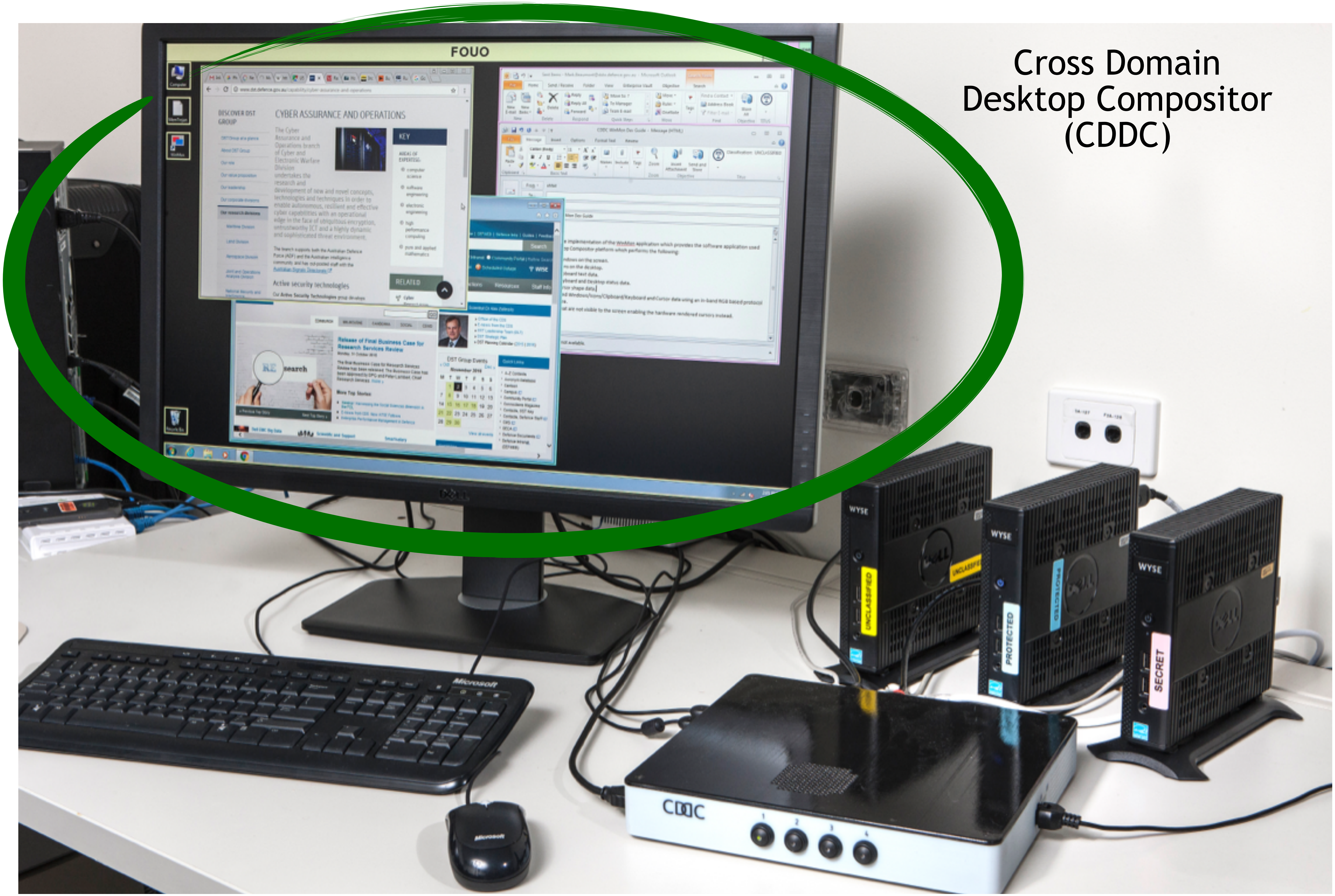


Cross Domain Desktop Compositor (CDDC)

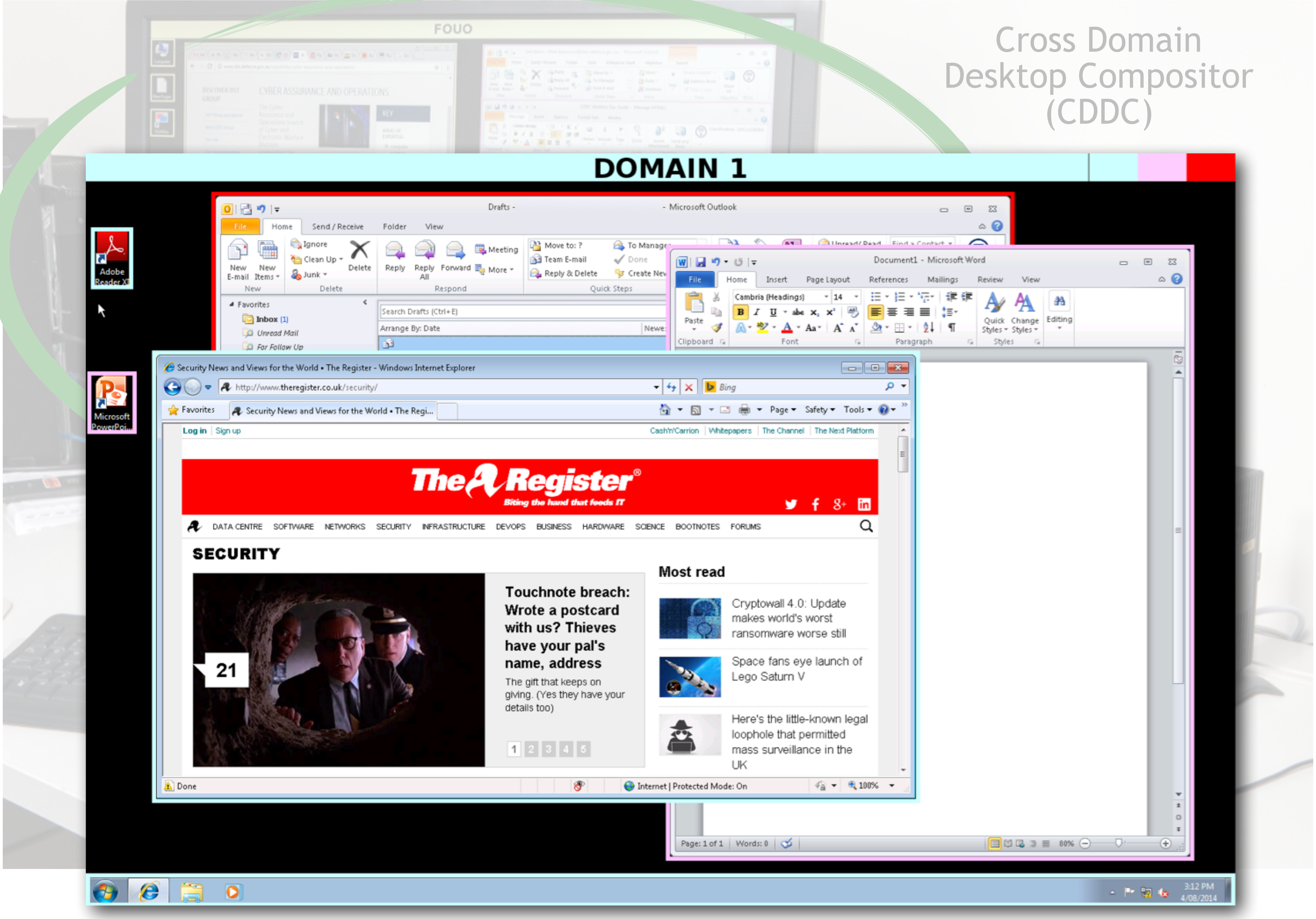
Cross Domain Desktop Compositor (CDDC)



Cross Domain Desktop Compositor (CDDC)



Cross Domain Desktop Compositor (CDDC)



Touchnote breach: Wrote a postcard with us? Thieves have your pal's name, address

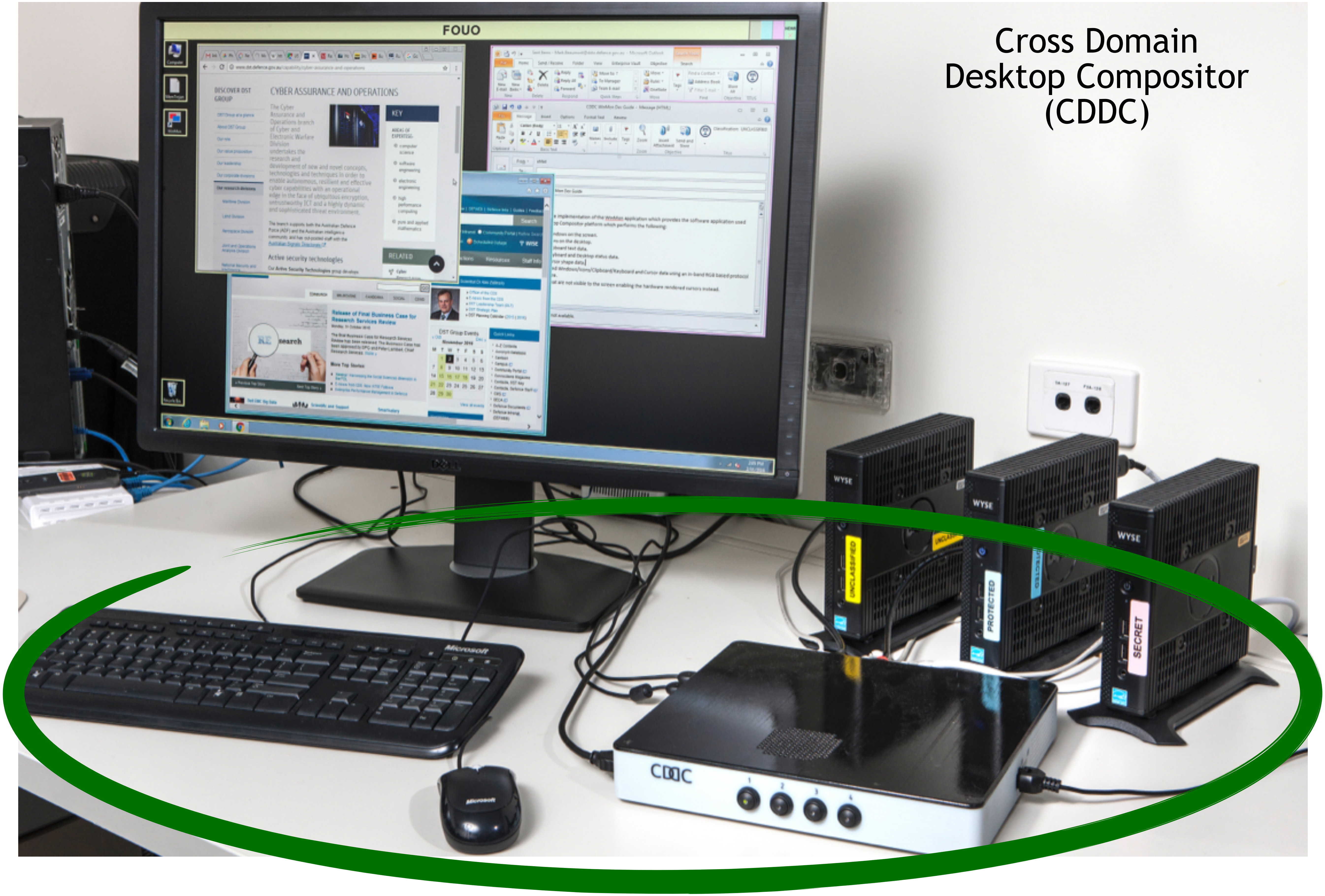
The gift that keeps on giving. (Yes they have your details too)

Cryptowall 4.0: Update makes world's worst ransomware worse still

Space fans eye launch of Lego Saturn V

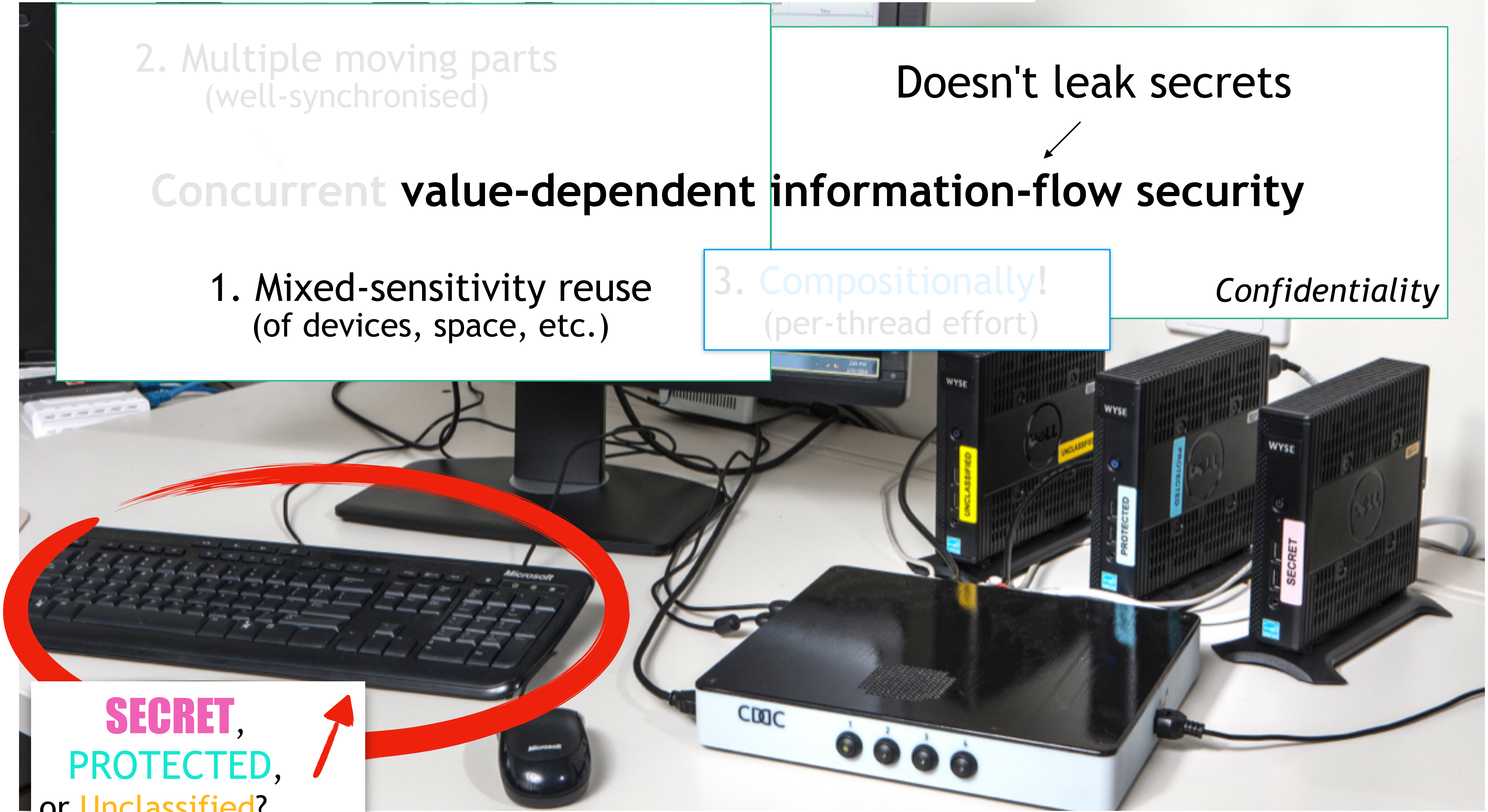
Here's the little-known legal loophole that permitted mass surveillance in the UK

Cross Domain Desktop Compositor (CDDC)



3 key challenges

Cross Domain
Desktop Compositor
(CDDC)



2. Multiple moving parts
(well-synchronised)

Doesn't leak secrets

Concurrent **value-dependent information-flow security**

1. Mixed-sensitivity reuse
(of devices, space, etc.)

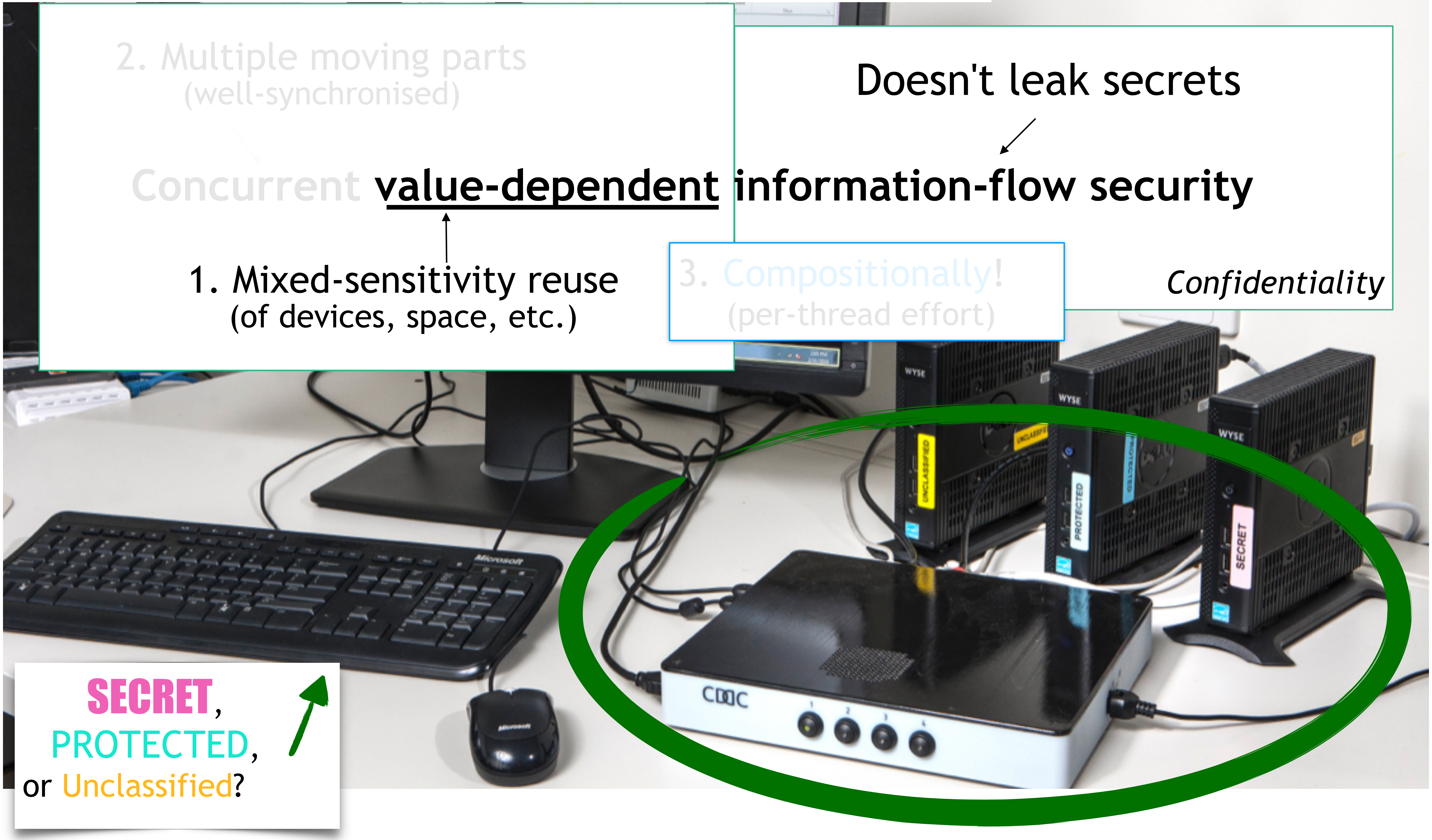
3. Compositionally!
(per-thread effort)

Confidentiality

SECRET,
PROTECTED,
or Unclassified?

3 key challenges

Cross Domain Desktop Compositor (CDDC)



2. Multiple moving parts (well-synchronised)

Doesn't leak secrets

Concurrent **value-dependent** **information-flow security**

1. Mixed-sensitivity reuse (of devices, space, etc.)

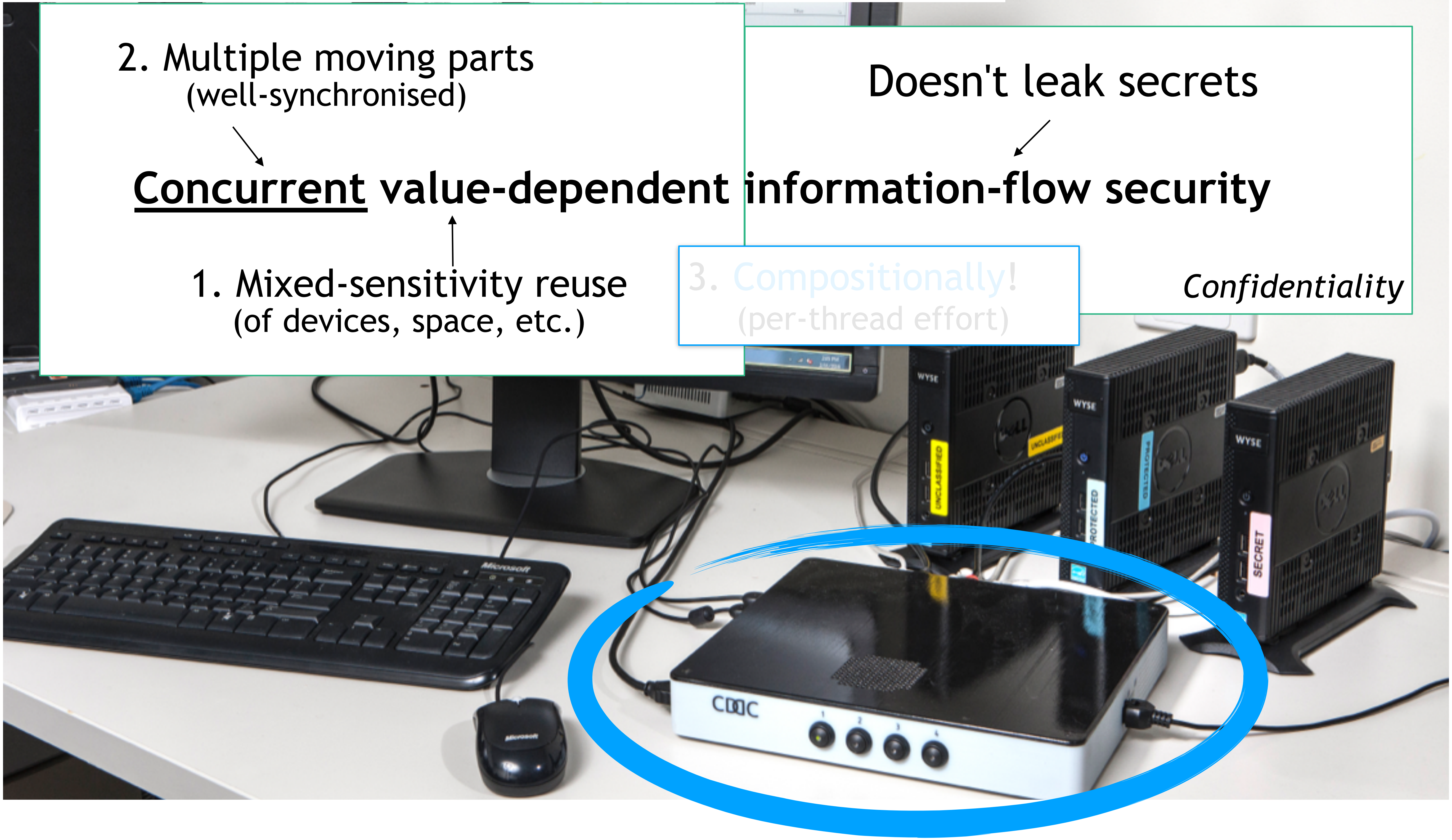
3. Compositionally! (per-thread effort)

Confidentiality

SECRET, PROTECTED, or Unclassified?

3 key challenges

Cross Domain
Desktop Compositor
(CDDC)



2. Multiple moving parts
(well-synchronised)

Doesn't leak secrets

<u>Concurrent</u> value-dependent information-flow security

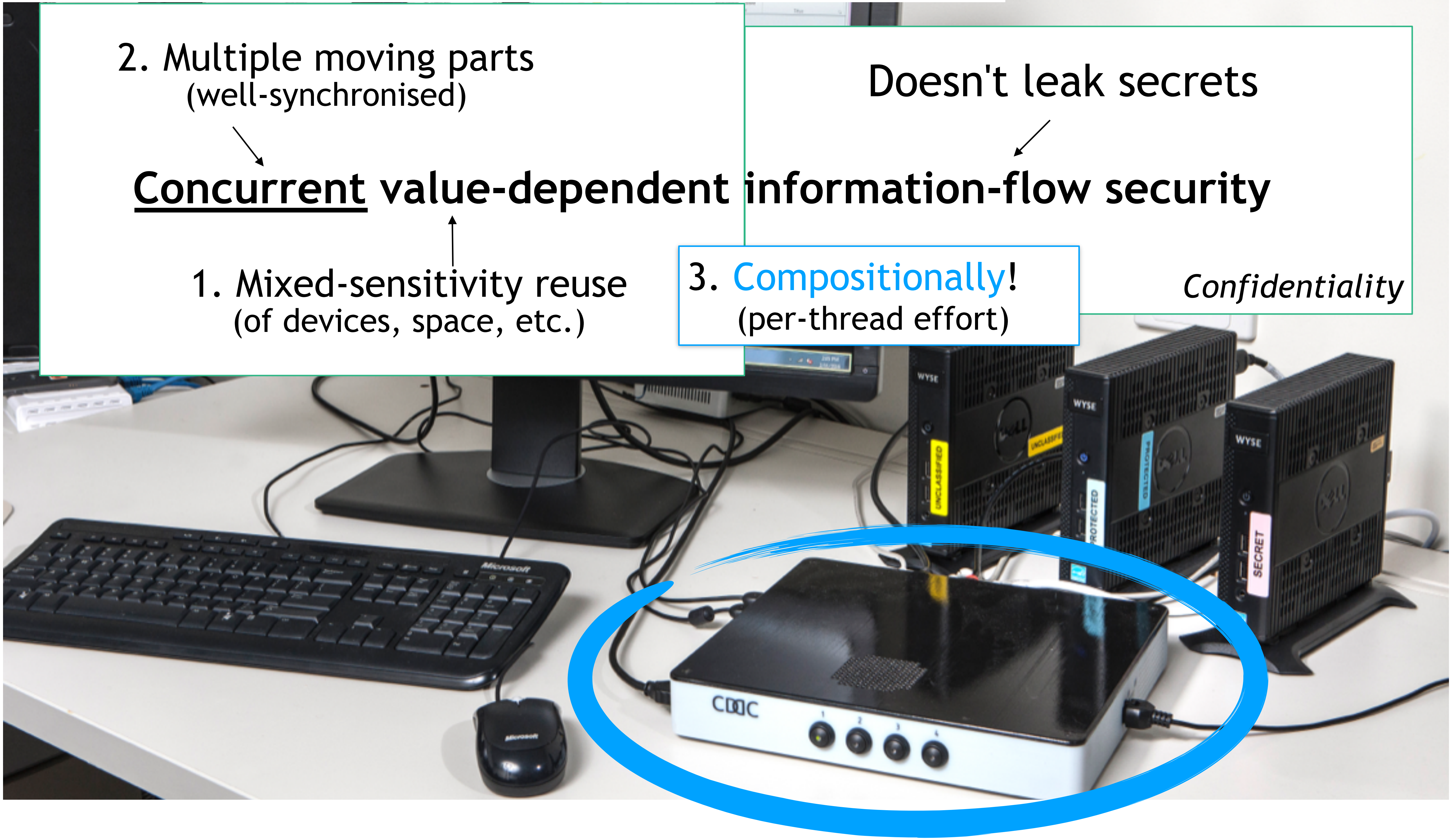
1. Mixed-sensitivity reuse
(of devices, space, etc.)

3. Compositionally!
(per-thread effort)

Confidentiality

3 key challenges

Cross Domain Desktop Compositor (CDDC)



2. Multiple moving parts
(well-synchronised)

Doesn't leak secrets

<u>Concurrent</u> value-dependent information-flow security

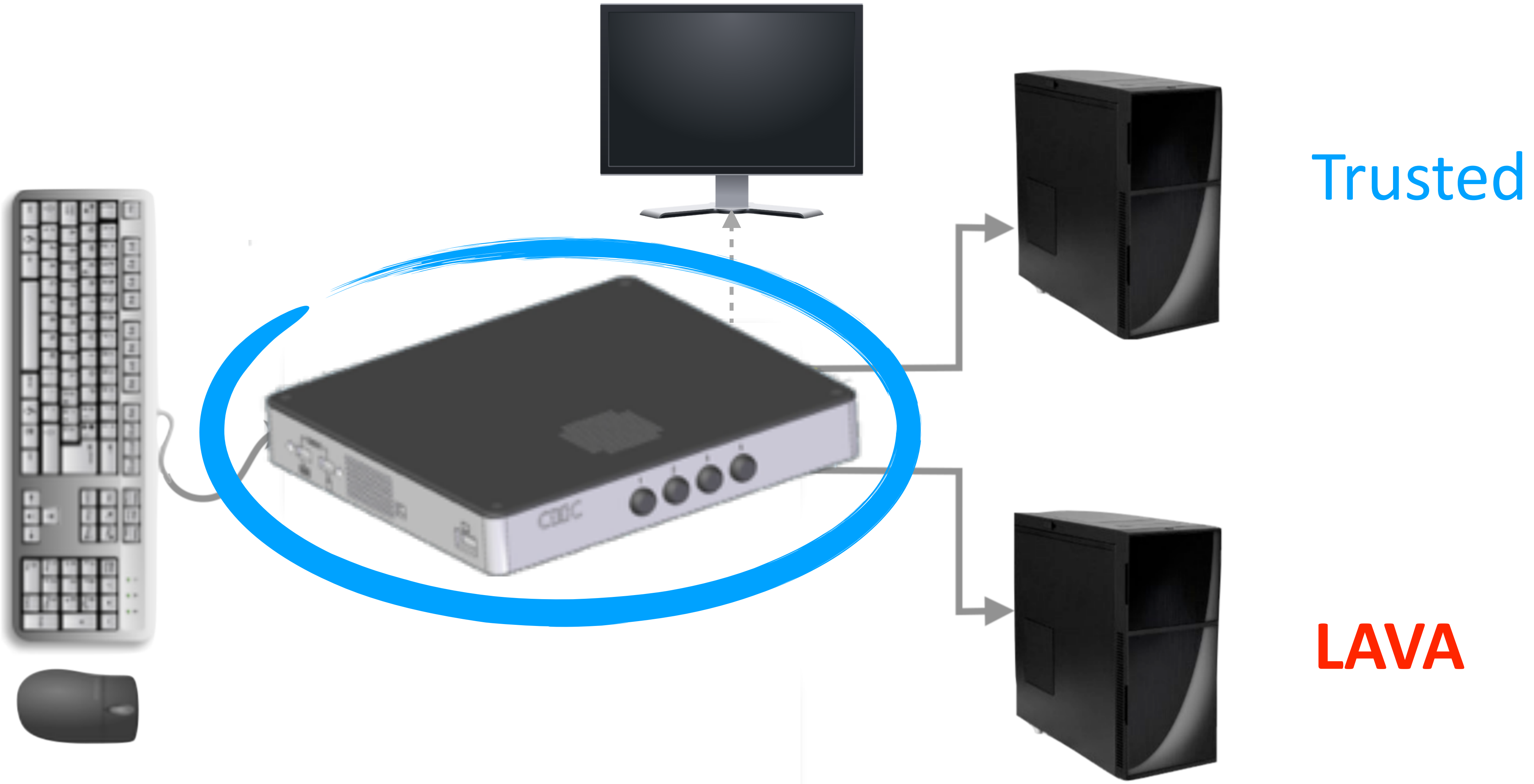
1. Mixed-sensitivity reuse
(of devices, space, etc.)

3. Compositionally!
(per-thread effort)

Confidentiality

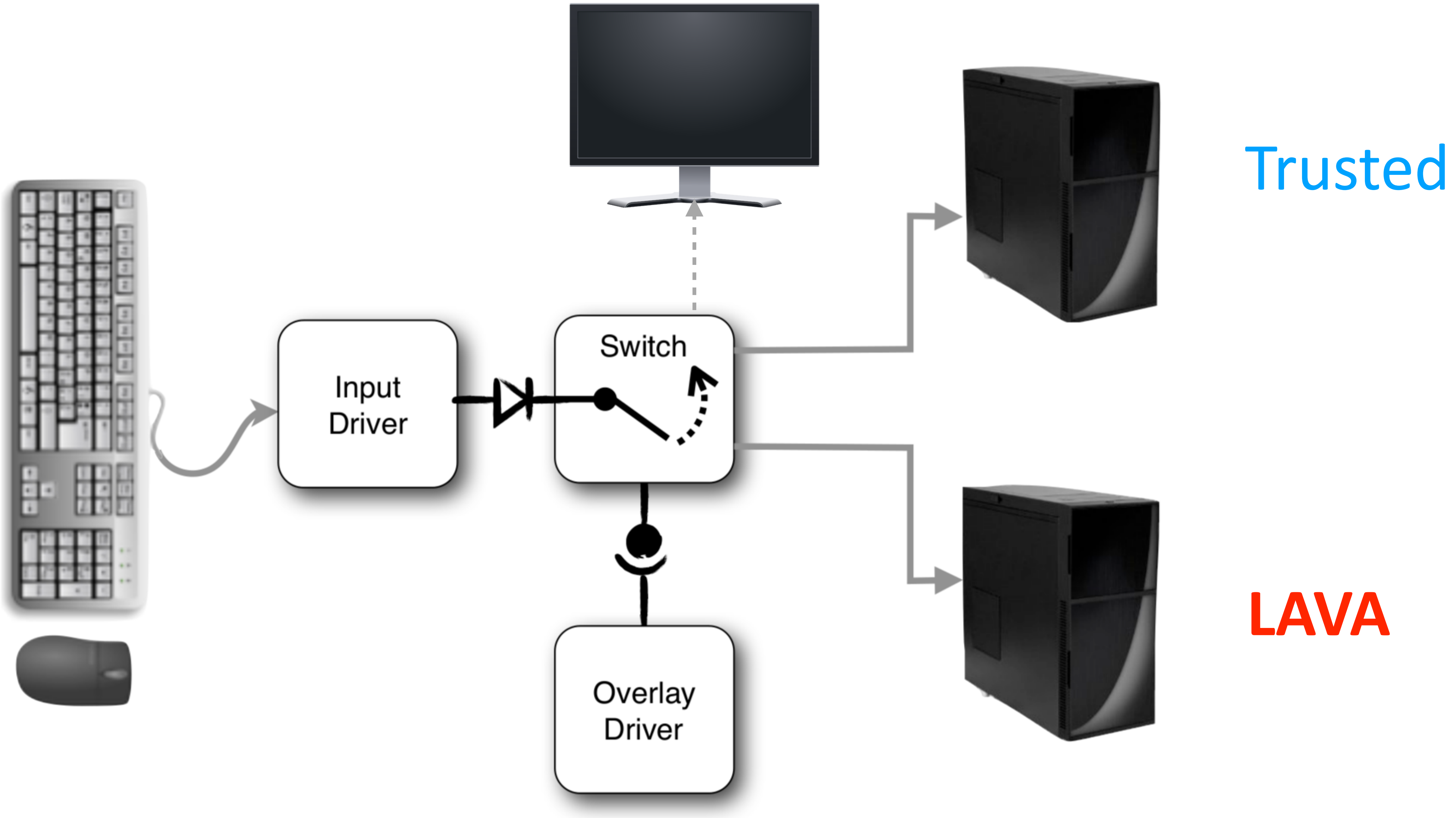
A mixed-sensitivity concurrent program

CDDC’s HID switch as software components



A mixed-sensitivity concurrent program

CDDC’s HID switch as software components

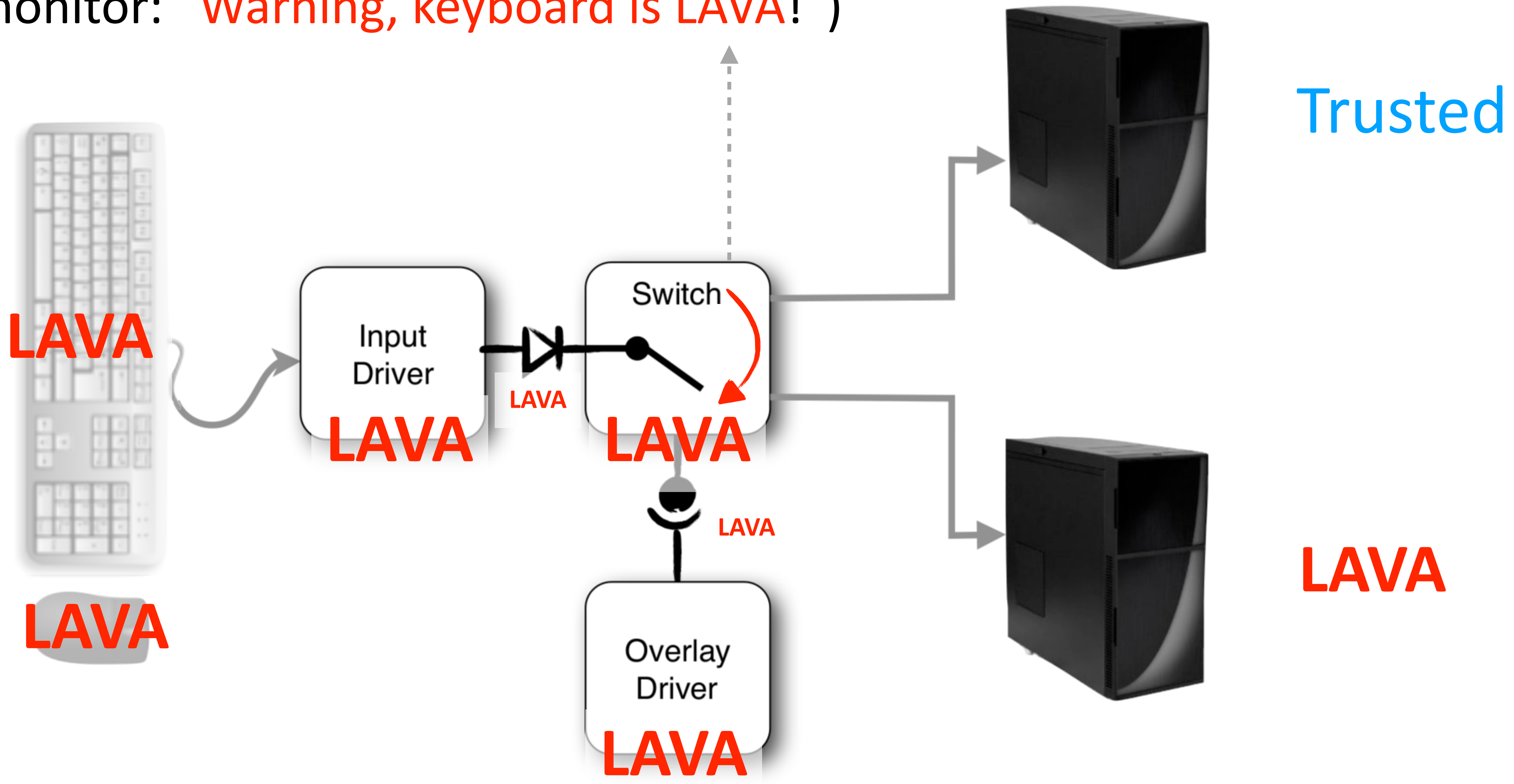


seL4 component architecture, functional schematic

A mixed-sensitivity concurrent program

CDDC’s HID switch as software components

(On video monitor: “Warning, keyboard is LAVA!”)

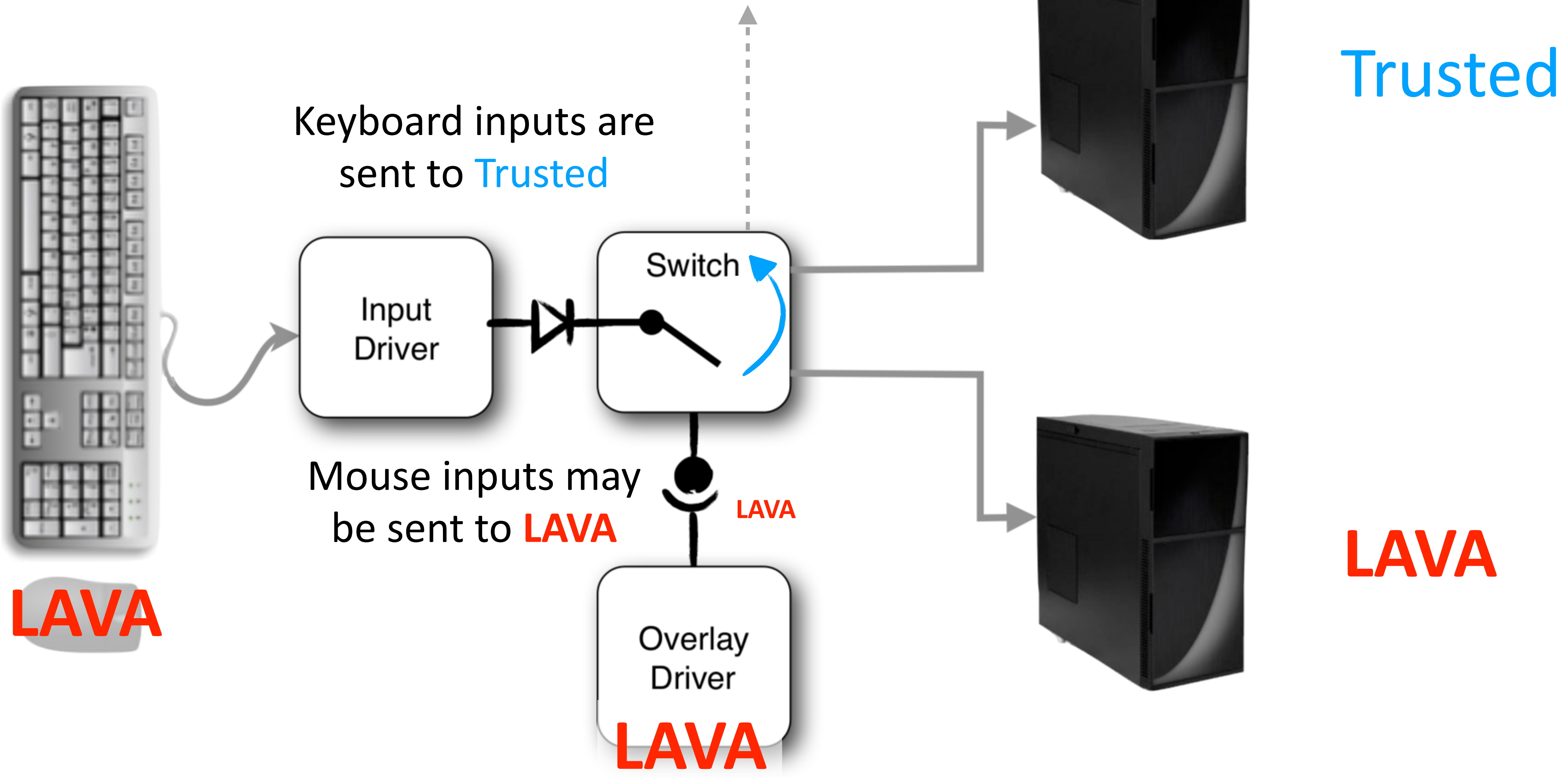


seL4 component architecture, functional schematic

A mixed-sensitivity concurrent program

CDDC’s HID switch as software components

(On video monitor: “Keyboard inputs can be secret”)



Keyboard inputs are sent to Trusted

Mouse inputs may be sent to LAVA

seL4 component architecture, functional schematic

Thesis (PhD, 2020)

That "Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs" *is feasible*.

- Program verification Chapter 4
- Compiler verification Chapter 5
- Case study: CDDC Chapter 6

- Extension to program verification Chapter 7

Thesis (PhD, 2020)

That “Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs” *is feasible*.

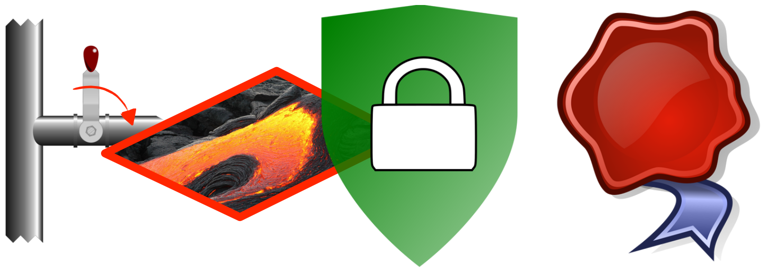
- Program verification Chapter 4
- Compiler verification Chapter 5
- Case study: CDDC Chapter 6

- Extension to program verification Chapter 7

Thesis (PhD, 2020)

That "Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs" *is feasible*.

- Program verification Chapter 4



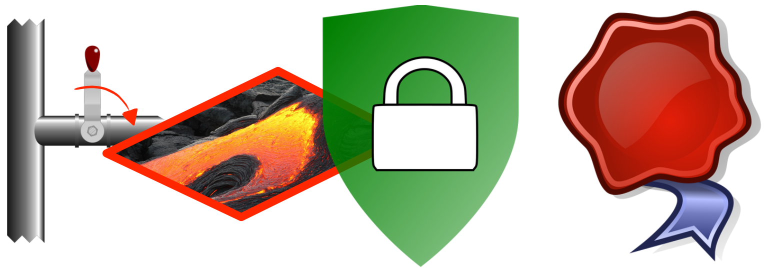
- Compiler verification Chapter 5
- Case study: CDDC Chapter 6

- Extension to program verification Chapter 7

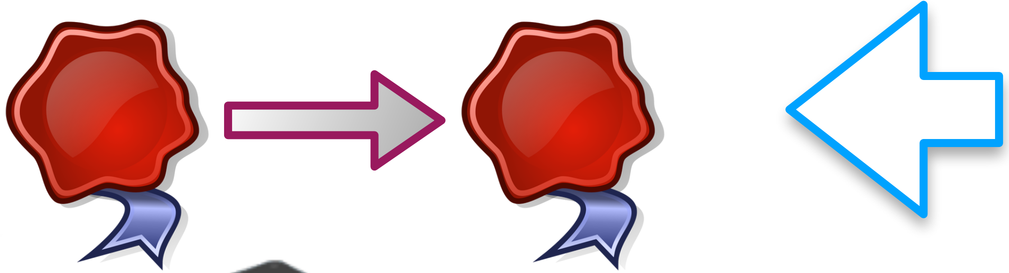
Thesis (PhD, 2020)

That “Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs” *is feasible*.

- Program verification Chapter 4



- Compiler verification Chapter 5



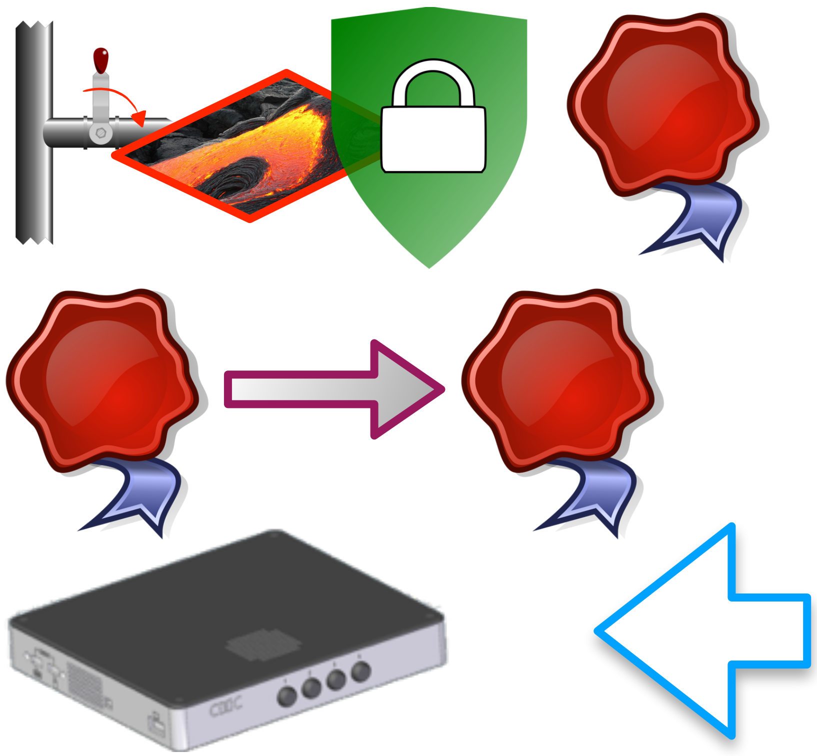
- Case study: CDDC Chapter 6

- Extension to program verification Chapter 7

Thesis (PhD, 2020)

That “Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs” *is feasible*.

- Program verification Chapter 4
- Compiler verification Chapter 5
- Case study: CDDC Chapter 6

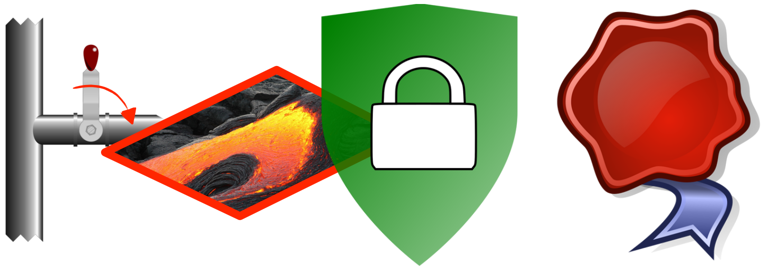


- Extension to program verification Chapter 7

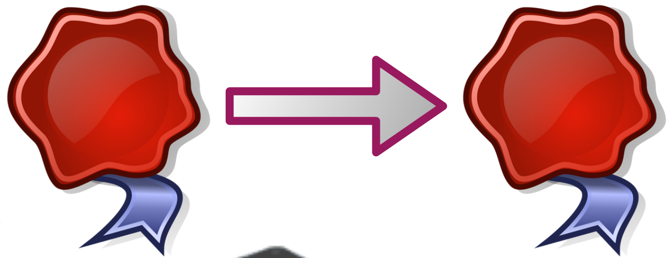
Thesis (PhD, 2020)

That "Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs" *is feasible*.

- Program verification Chapter 4



- Compiler verification Chapter 5



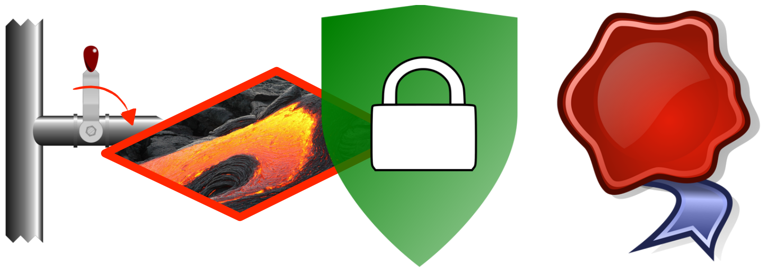
- Case study: CDDC Chapter 6

- Extension to program verification Chapter 7

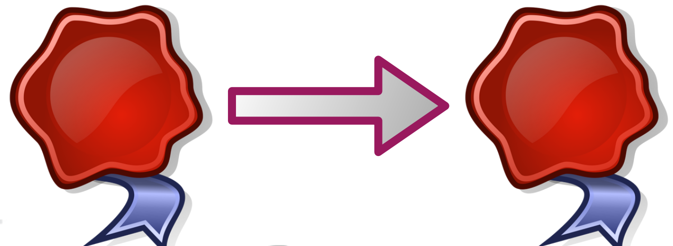
Thesis (PhD, 2020)

That “Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs” *is feasible*.

- Program verification Chapter 4



- Compiler verification Chapter 5



- Case study: CDDC Chapter 6

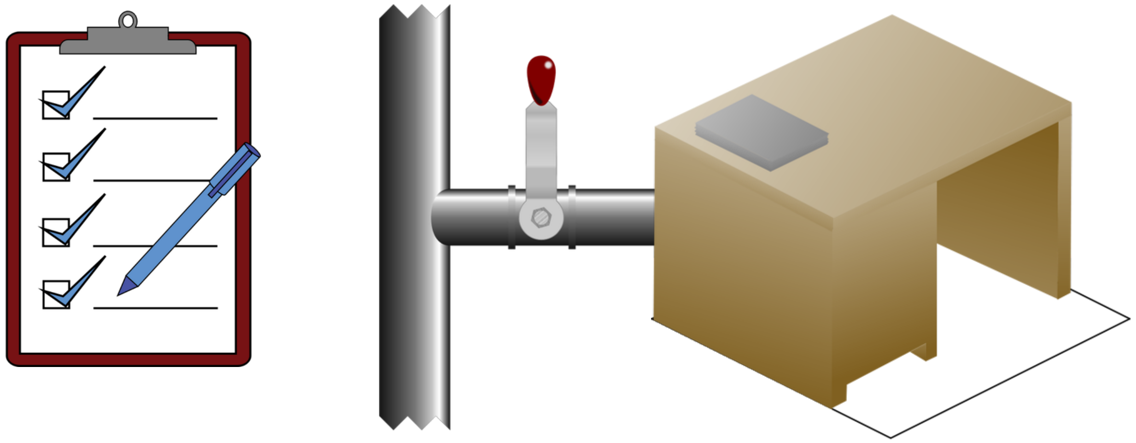
- Extension to program verification Chapter 7

Program verification: Prior work

(Murray, Sison, Pierzchalski, Rizkallah 2016)

Assume-guarantee contracts between threads

(Jones 1983 via Mantel et al. 2011)

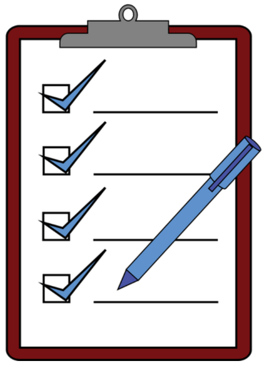


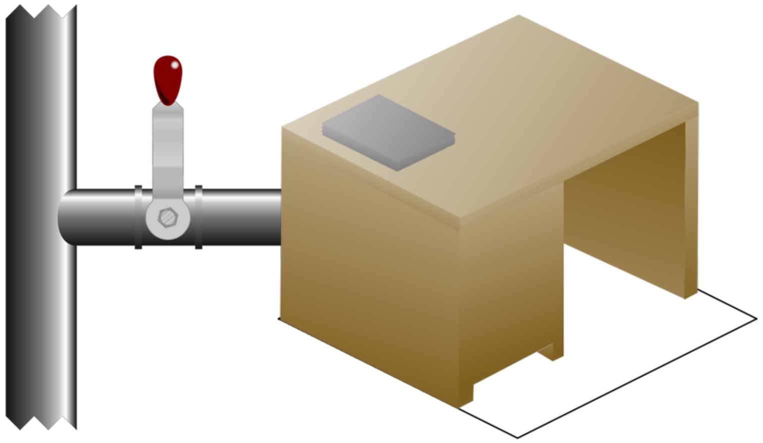
Program verification: Prior work

(Murray, Sison, Pierzchalski, Rizkallah 2016)

Assume-guarantee contracts between threads

(Jones 1983 via Mantel et al. 2011)





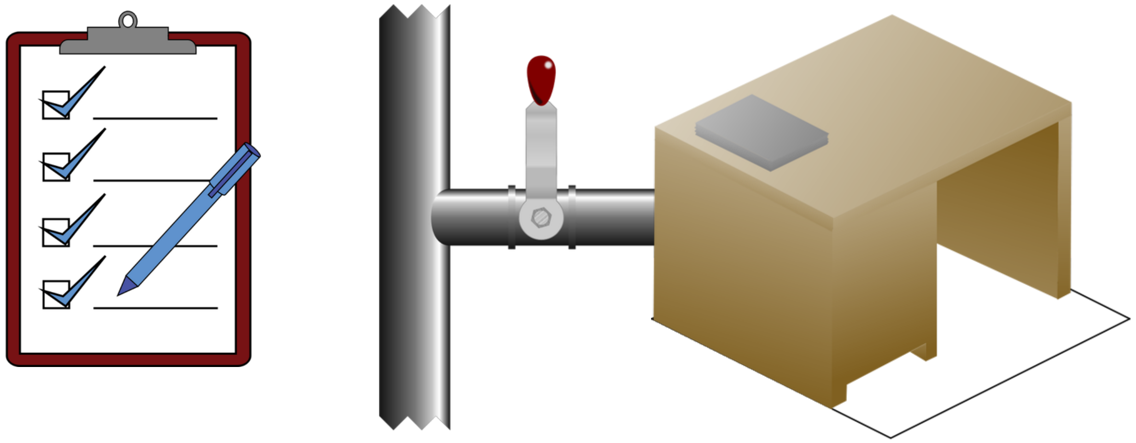
“When I’m not sitting at this desk, I *guarantee* not to touch it.”

Program verification: Prior work

(Murray, Sison, Pierzchalski, Rizkallah 2016)

Assume-guarantee contracts between threads

(Jones 1983 via Mantel et al. 2011)



“When I’m not sitting at this desk, I *guarantee* not to touch it.”

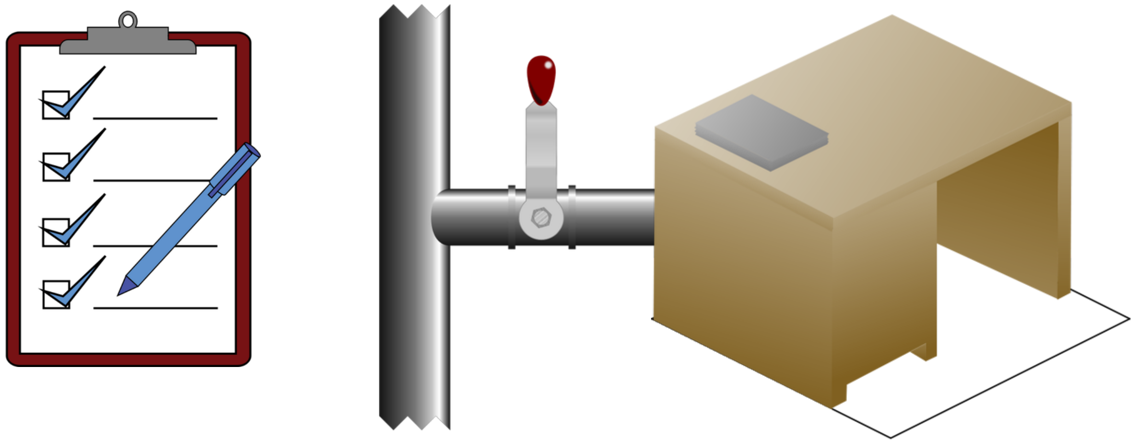
“When I’m sitting at this desk, I *assume* that nobody else will pull the lever.”

Program verification: Prior work

(Murray, Sison, Pierzchalski, Rizkallah 2016)

Assume-guarantee contracts between threads

(Jones 1983 via Mantel et al. 2011)



- Local compliance

  "I respect my guarantees"

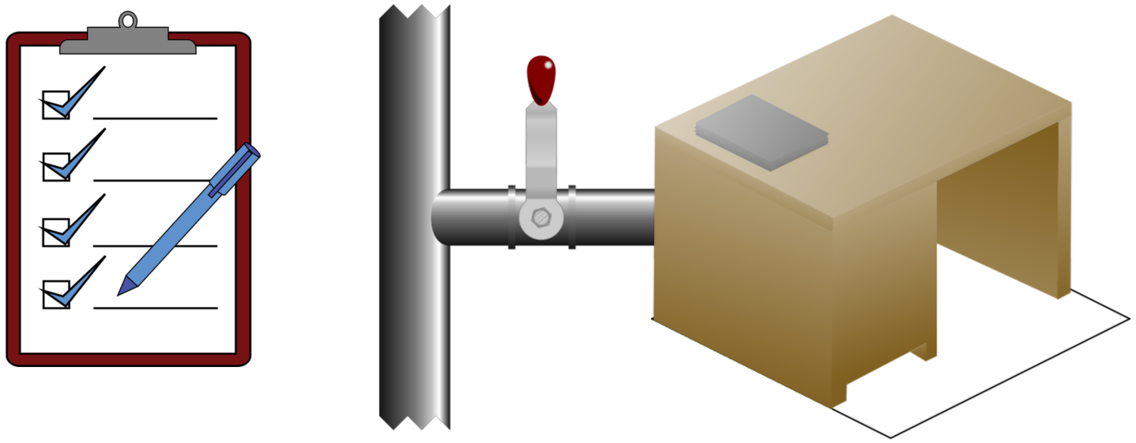
"When I'm sitting at this desk, I *assume* that nobody else will pull the lever."

Program verification: Prior work

(Murray, Sison, Pierzchalski, Rizkallah 2016)

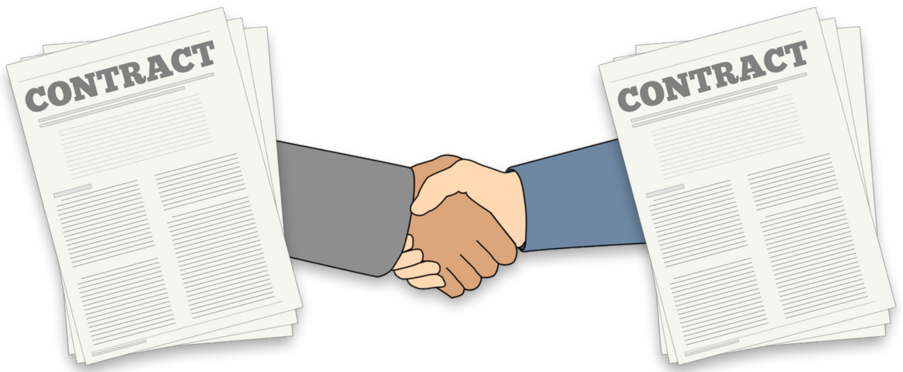
Assume-guarantee contracts between threads

(Jones 1983 via Mantel et al. 2011)



- Local compliance

  "I respect my guarantees"



- Global compatibility

  "Guarantees meet assumptions"

"When I'm sitting at this desk, I *assume* that nobody else will pull the lever."

Program verification: Prior work

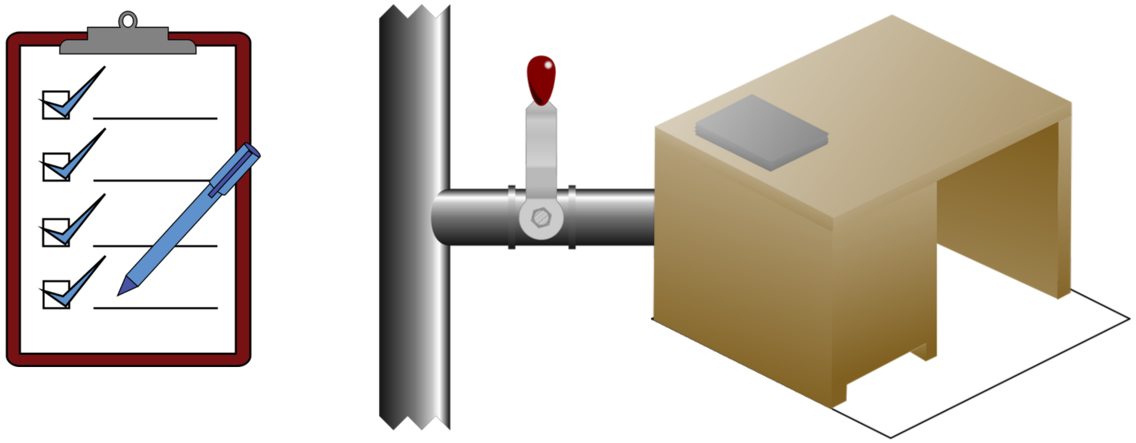
(Murray, Sison, Pierzchalski, Rizkallah 2016)

Assume-guarantee contracts between threads

(Jones 1983 via Mantel et al. 2011)

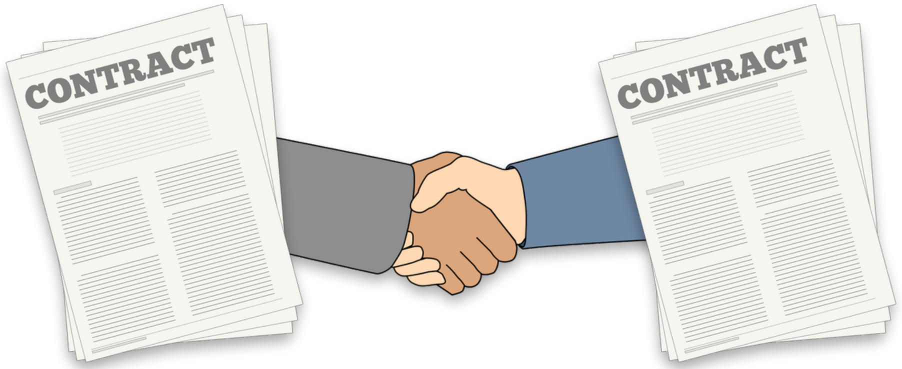
- Local compliance

  “I respect my guarantees”



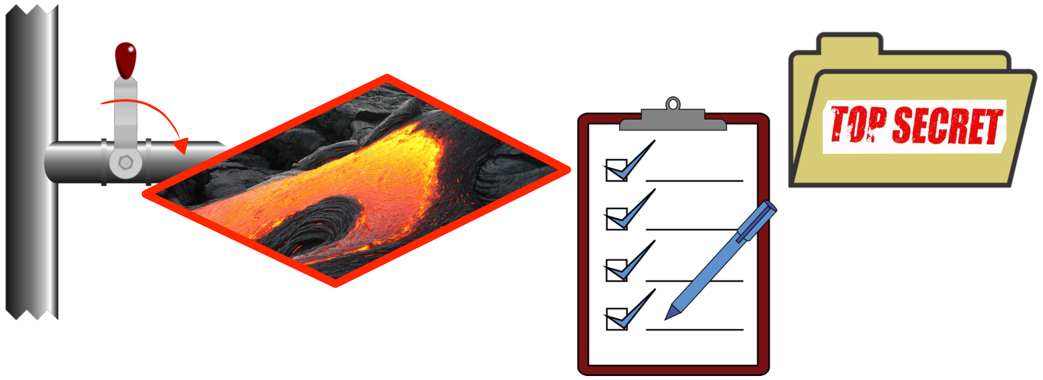
- Global compatibility

  “Guarantees meet assumptions”



- Local security

  “I win ‘floor is lava with levers’ using assumptions”

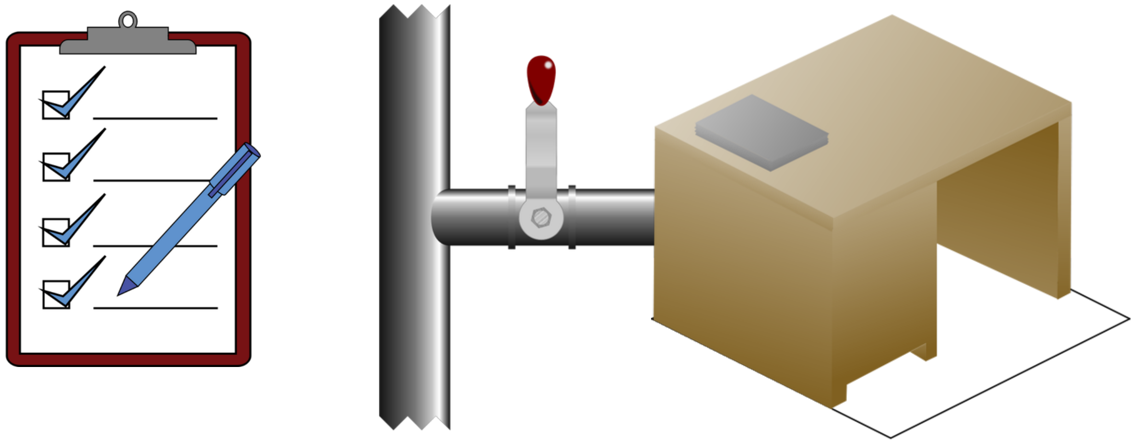


Program verification: Prior work

(Murray, Sison, Pierzchalski, Rizkallah 2016)

Assume-guarantee contracts between threads

(Jones 1983 via Mantel et al. 2011)



- Local compliance

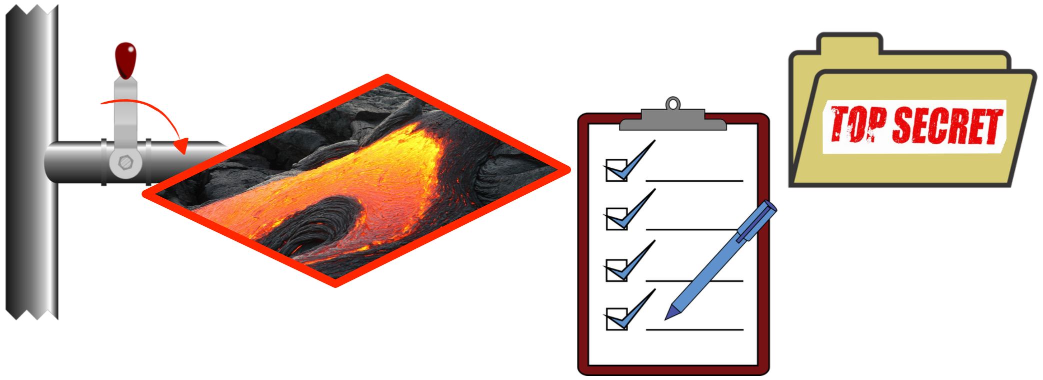
  “I respect my guarantees”

- Global compatibility

  “Guarantees meet assumptions”

- Local security

  “I win ‘floor is lava with levers’ using assumptions”



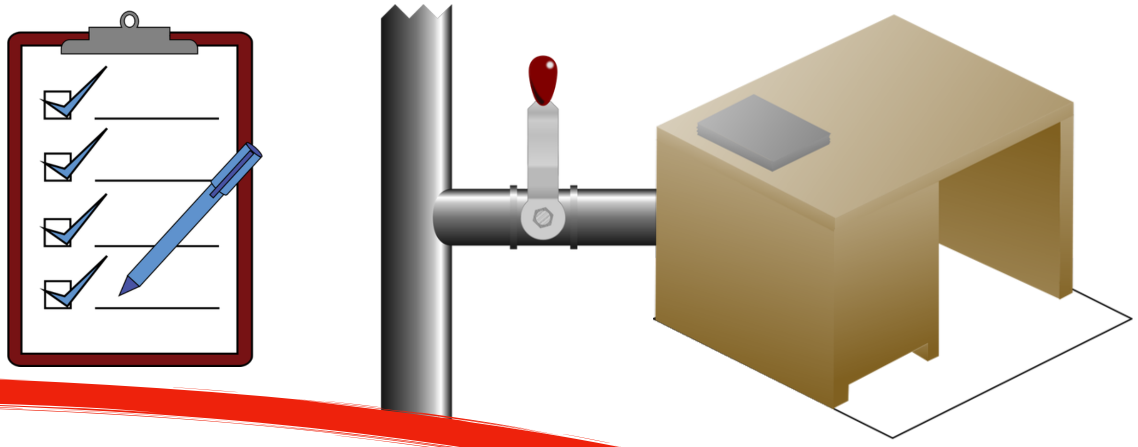
(Type systems for a generic While language)

Program verification: Prior work

(Murray, Sison, Pierzchalski, Rizkallah 2016)

Assume-guarantee contracts between threads

(Jones 1983 via Mantel et al. 2011)

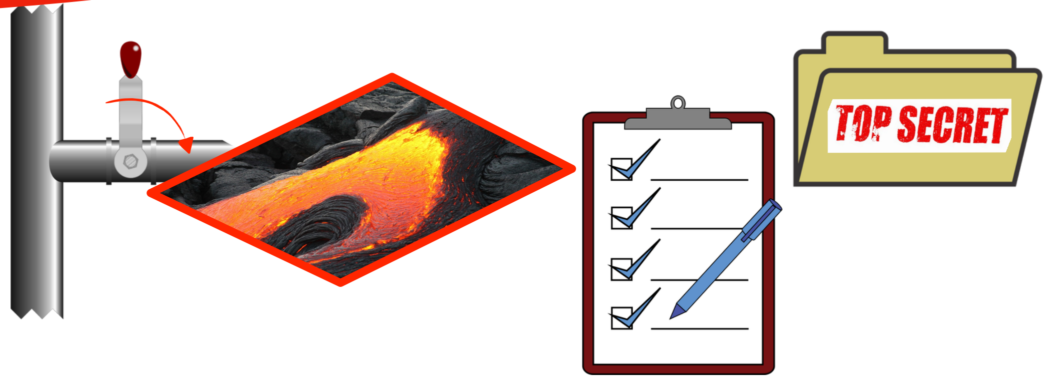


- Local compliance
  “I respect my guarantees”

- Global compatibility
  “Guarantees meet assumptions”

?

- Local security
  “I win ‘floor is lava with levers’ using assumptions”

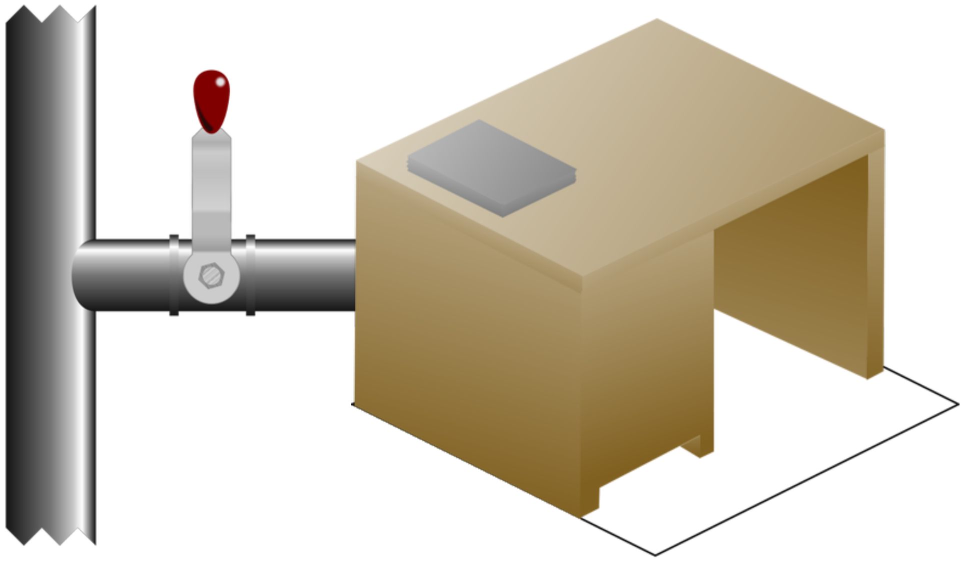


(Type systems for a generic While language)

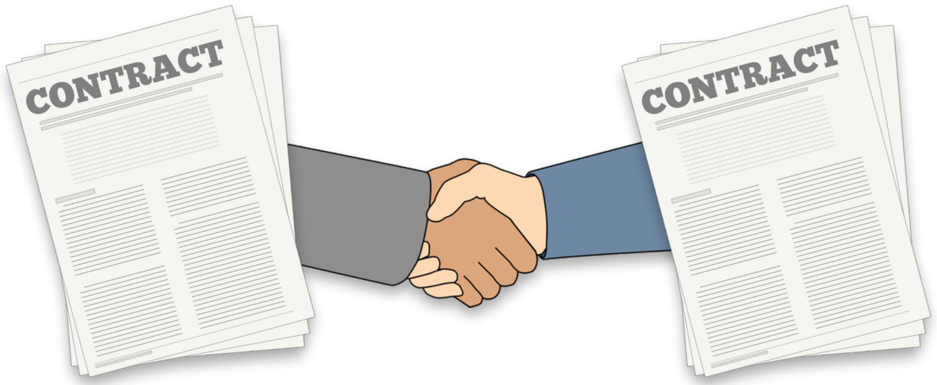
Program verification: My work

(Language designer's perspective)

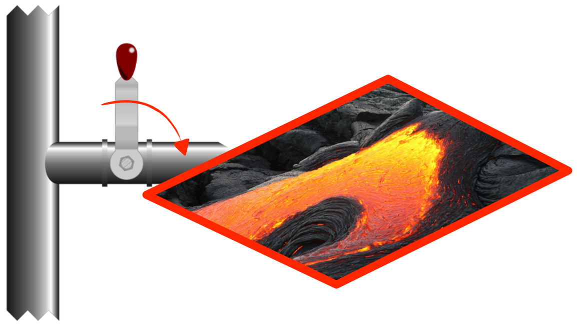
- Local compliance
  "I respect my guarantees"



- Global compatibility
  "Guarantees meet assumptions"



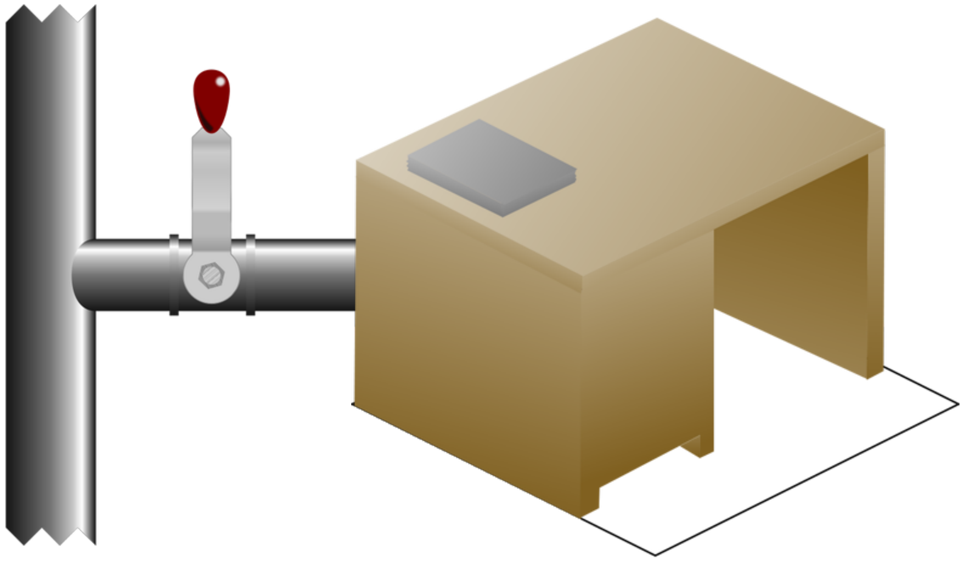
- Local security
  "I win 'floor is lava with levers' using assumptions"



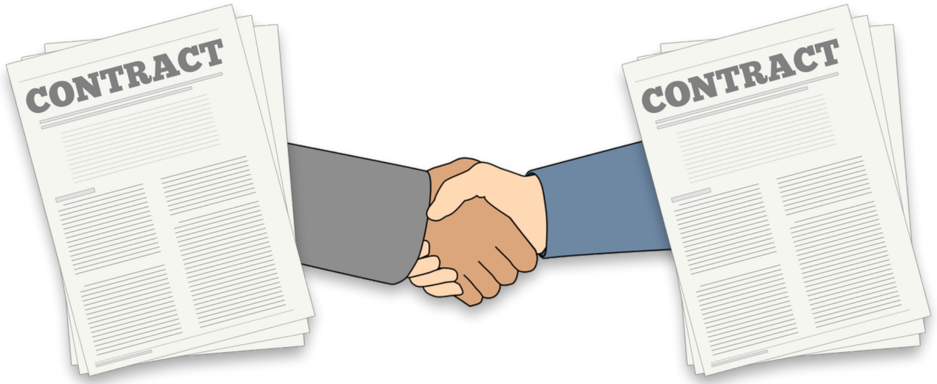
Program verification: My work

(Language designer’s perspective)

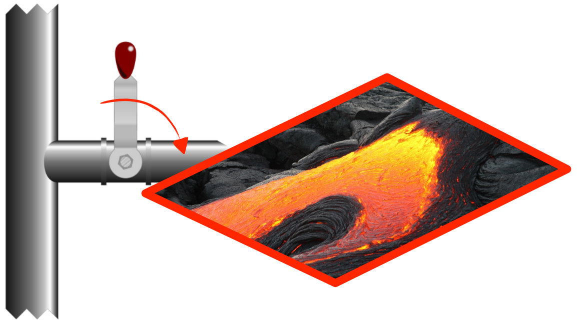
- Local compliance
  “I respect my guarantees”



- Global compatibility
  “Guarantees meet assumptions”



- Local security
  “I win ‘floor is lava with levers’ using assumptions”

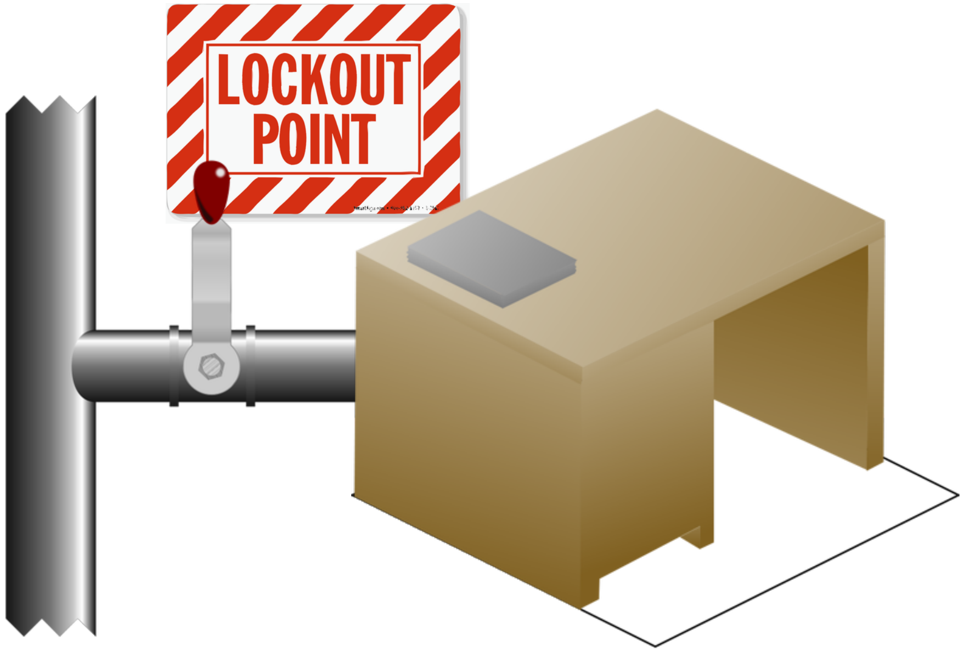


(For a generic While language with locks)

Program verification: My work

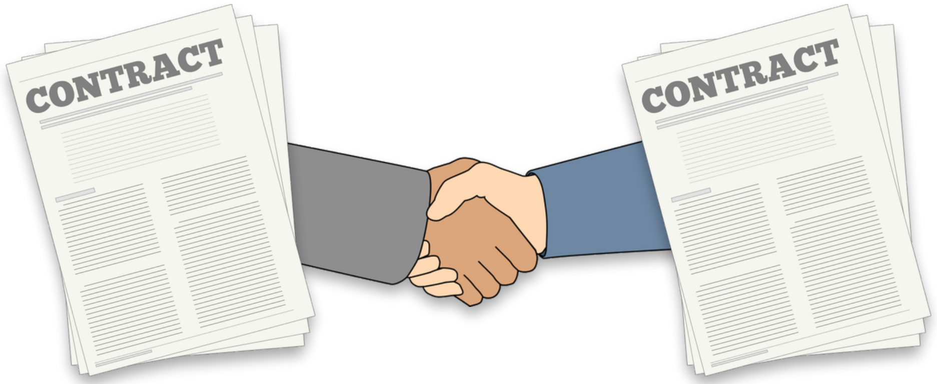
(Language designer’s perspective)

- A way to assign locks to spaces

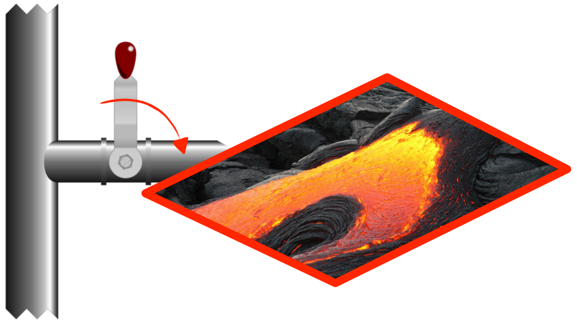


- Local compliance
  “I respect my guarantees”

- Global compatibility
  “Guarantees meet assumptions”



- Local security
  “I win ‘floor is lava with levers’ using assumptions”

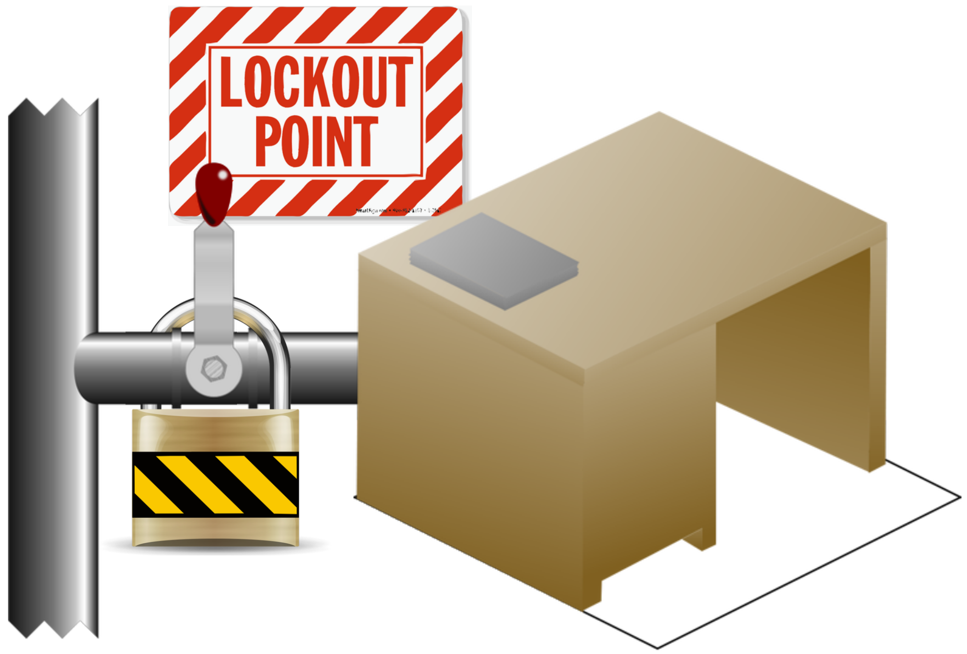


(For a generic While language with locks)

Program verification: My work

(Language designer's perspective)

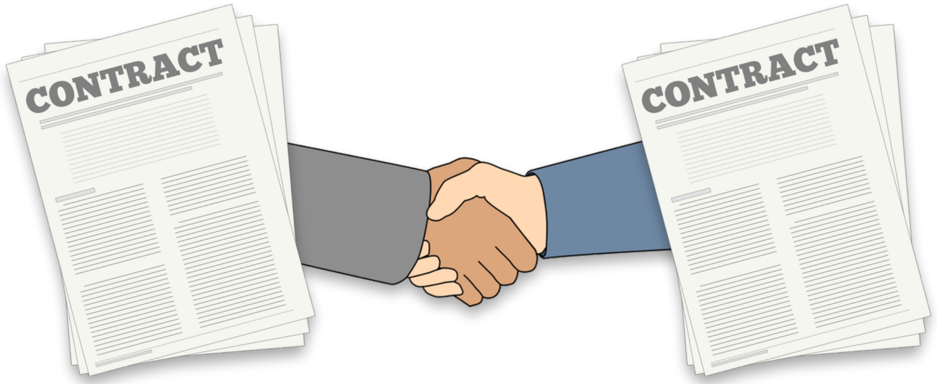
- A way to assign locks to spaces



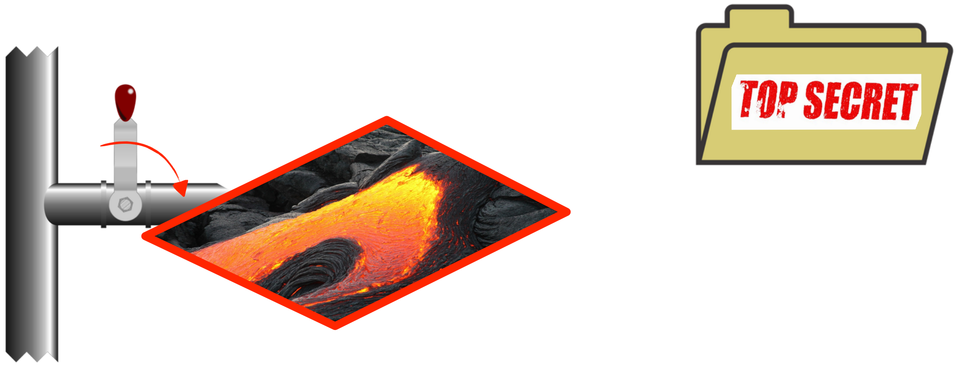
- Local compliance
  "I respect my guarantees"

- Global compatibility

  "Guarantees meet assumptions"



- Local security
  "I win 'floor is lava with levers'
  using assumptions"

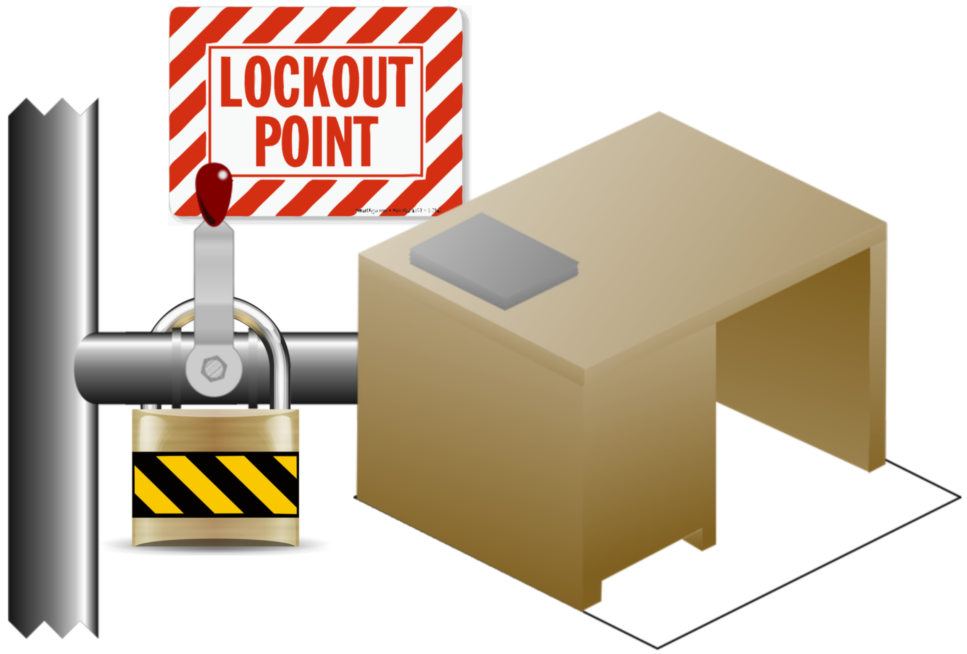


(For a generic While language with locks)

Program verification: My work

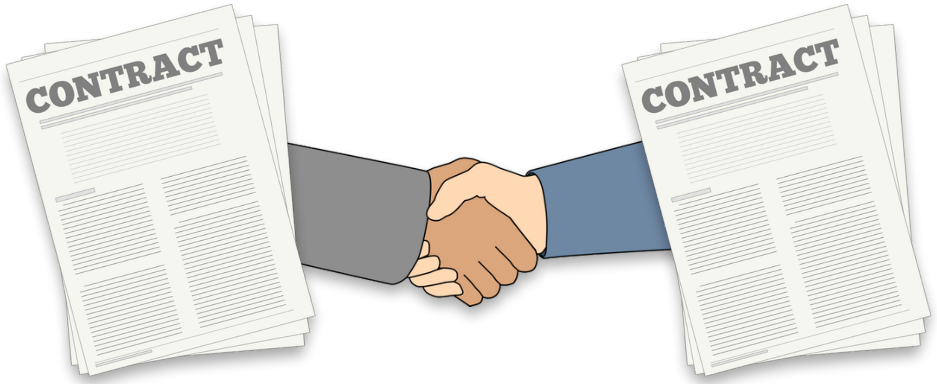
(Language designer's perspective)

- A way to assign locks to spaces

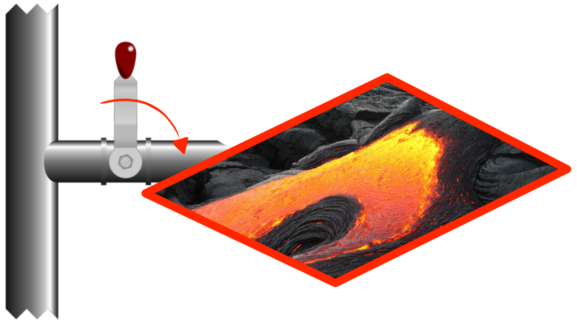


- Local compliance
  "I respect the locks"

- Global compatibility
  "Guarantees meet assumptions"



- Local security
  "I win 'floor is lava with levers' using assumptions"

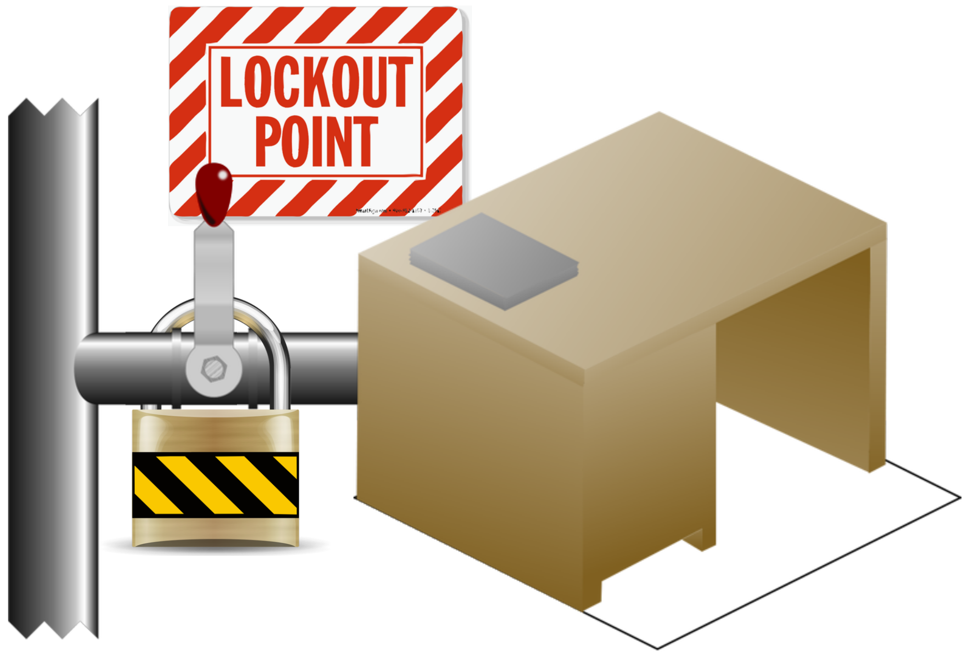


(For a generic While language with locks)

Program verification: My work

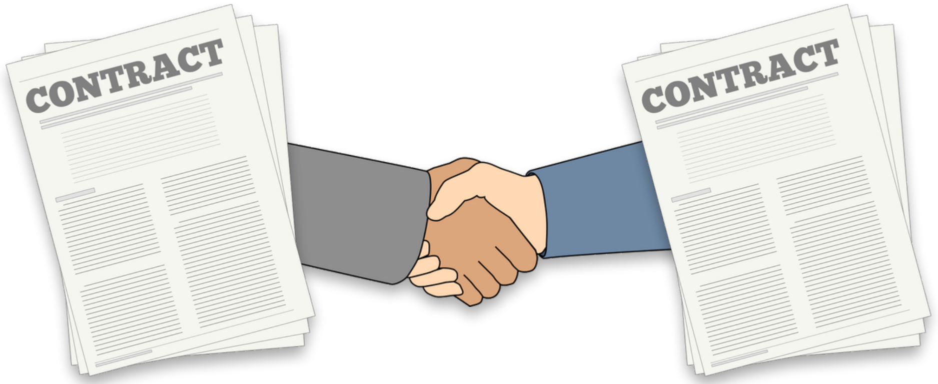
(Language designer's perspective)

- A way to assign locks to spaces



- Local compliance
  "I respect the locks"

- Global compatibility
  "Respecting locks is enough for guarantees to meet assumptions"



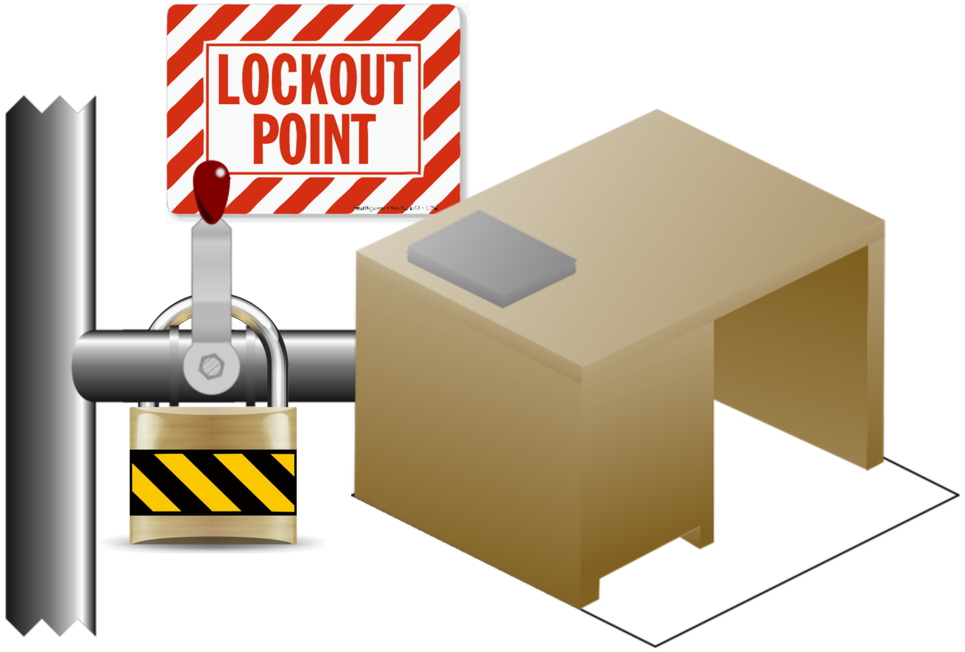
- Local security
  "I win 'floor is lava with levers' using locks"

(For a generic While language with locks)

Program verification: My work

(Language designer's perspective)

- A way to assign locks to spaces



- Local compliance
  "I respect the locks"

- ~~Global compatibility~~
  ~~"Respecting locks is enough for guarantees to meet assumptions"~~

- Local security
  "I win 'floor is lava with levers' using locks"

(For a generic While language with locks)

Program verification: My work

(Programmer's perspective)

- Assign locks to spaces

- ~~Global compatibility~~
  ~~"Respecting locks is enough for guarantees to meet assumptions"~~

- For each thread, prove:

  - Local compliance
    "I respect the locks"

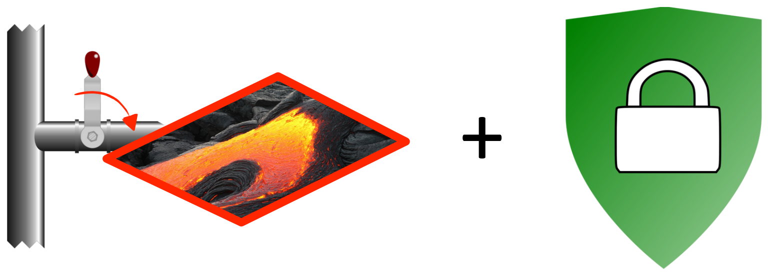
  - Local security
    "I win 'floor is lava with levers' using locks"

(Type systems provide proof method)

Thesis (PhD, 2020)

That “Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs” is feasible.

- ~~Program verification~~ Chapter 4



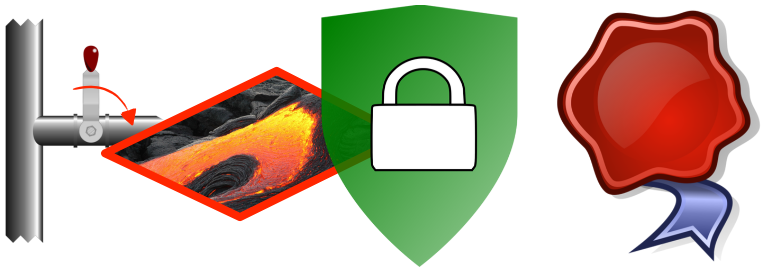
- Compiler verification Chapter 5
- Case study: CDDC Chapter 6

- Extension to program verification Chapter 7

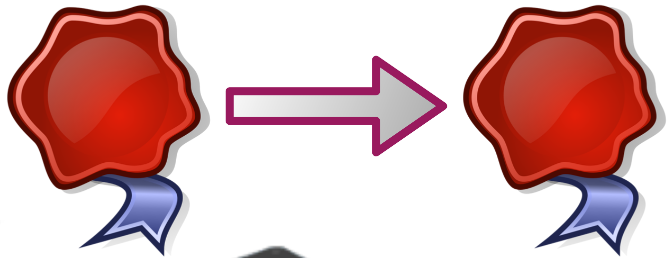
Thesis (PhD, 2020)

That “Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs” is feasible.

- ~~Program verification~~ Chapter 4



- Compiler verification Chapter 5

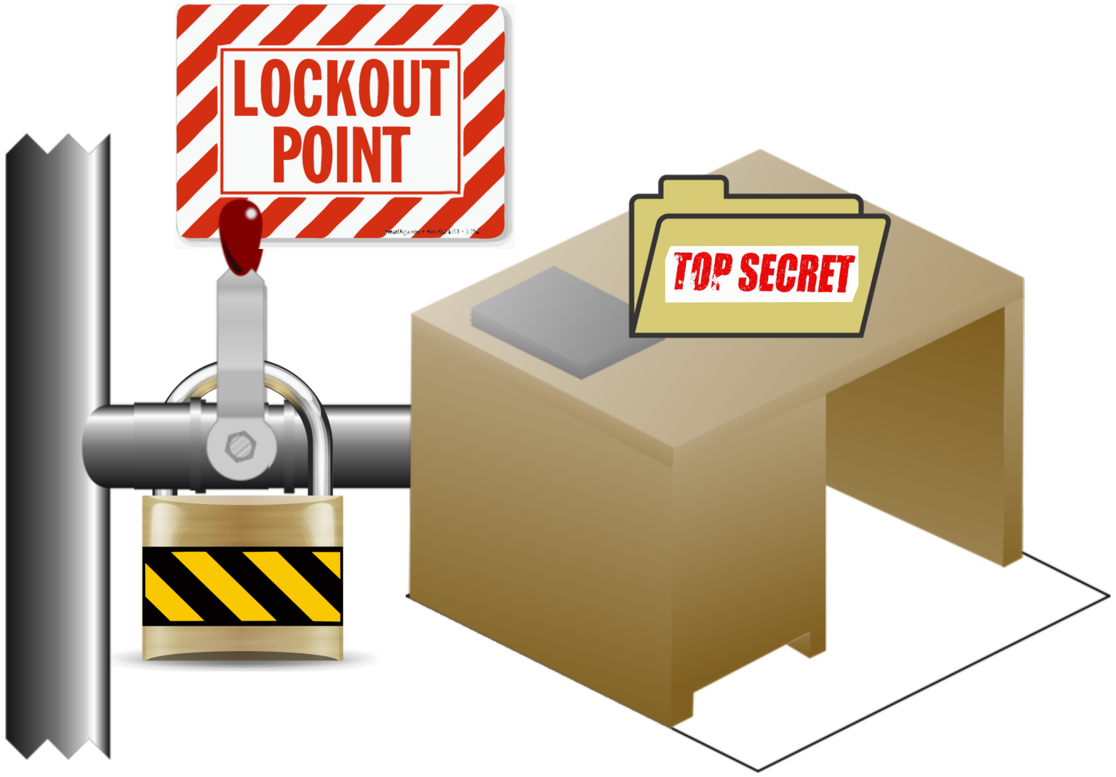


- Case study: CDDC Chapter 6

- Extension to program verification Chapter 7

Verified secure compilation

Say you've proved your mixed-sensitivity concurrent program *doesn't leak secrets…*



Verified secure compilation

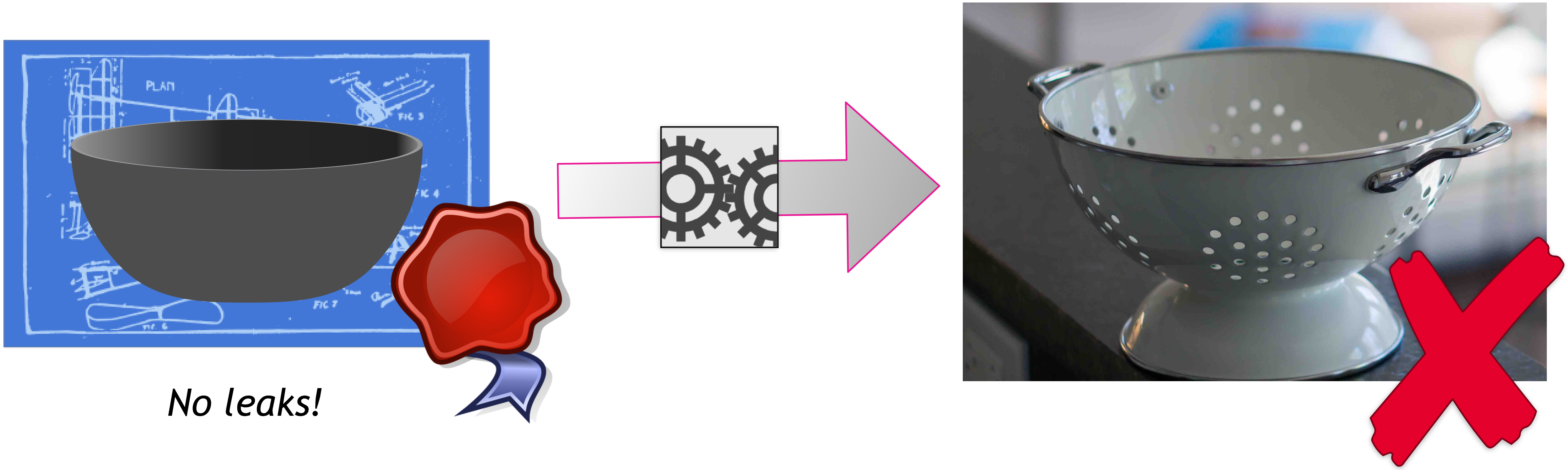
Say you've <u>proved</u> your mixed-sensitivity concurrent program *doesn't leak secrets…*

No leaks!

Verified secure compilation

Say you've <u>proved</u> your mixed-sensitivity concurrent program *doesn't leak secrets…*

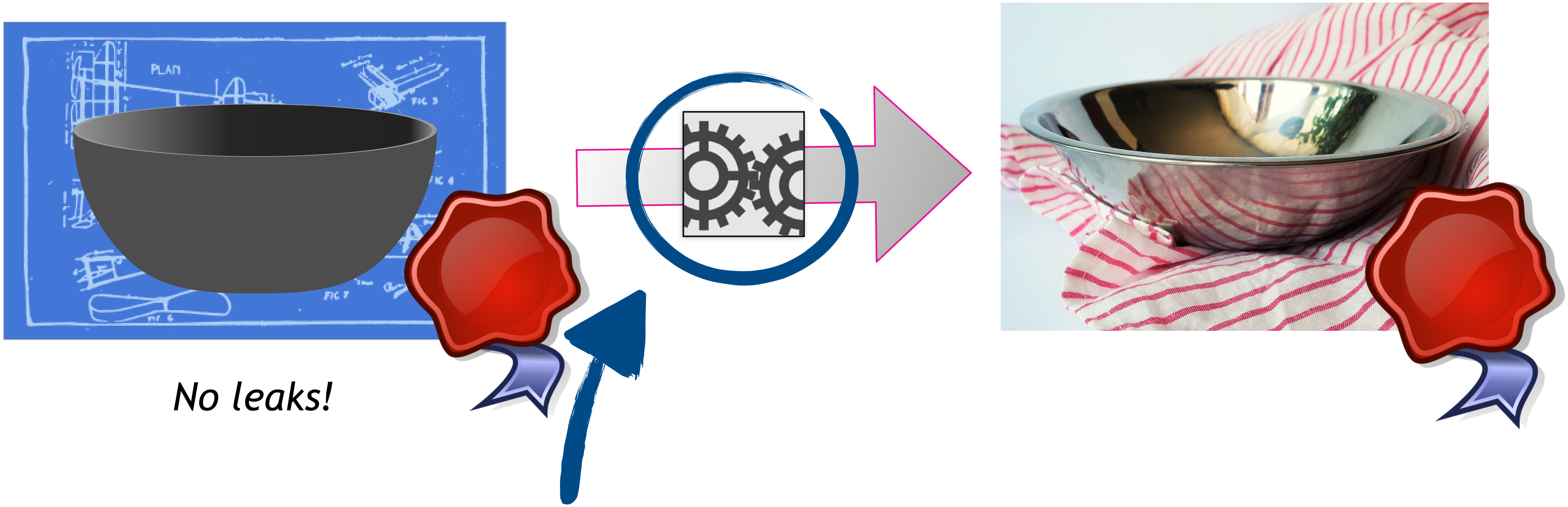
How do you know your compiler won't *introduce leaks*?



Verified secure compilation

Say you've <u>proved</u> your mixed-sensitivity concurrent program *doesn't leak secrets…*

How do you know your compiler won't *introduce leaks*?



What if your compiler <u>could be proved</u> to *preserve* it?

Verified secure compilation

Say you've proved your mixed-sensitivity concurrent program *doesn't leak secrets…*

What if your compiler could be proved to *preserve* it?

Verified secure compilation

Say you've proved your mixed-sensitivity concurrent program *doesn't leak secrets…*

What if your compiler could be proved to *preserve* it?

Here's how!

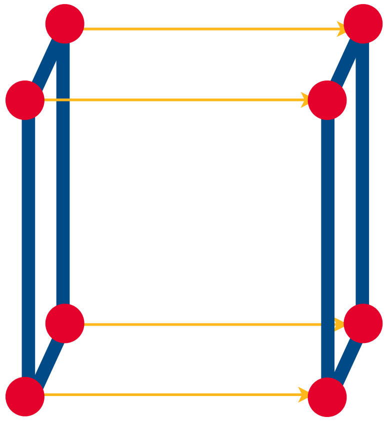
Verified secure compilation

Say you’ve proved your mixed-sensitivity concurrent program *doesn’t leak secrets…*

What if your compiler could be proved to *preserve* it?

Here's how!

Prove *confidentiality-preserving* refinement,



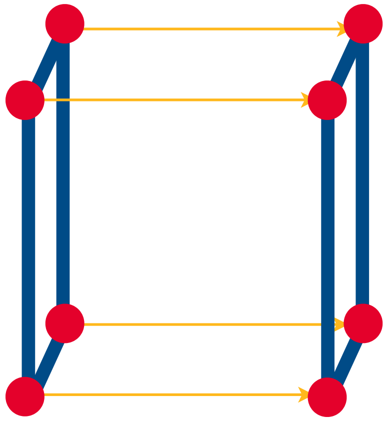
Verified secure compilation

Say you've <u>proved</u> your mixed-sensitivity concurrent program *doesn't leak secrets…*

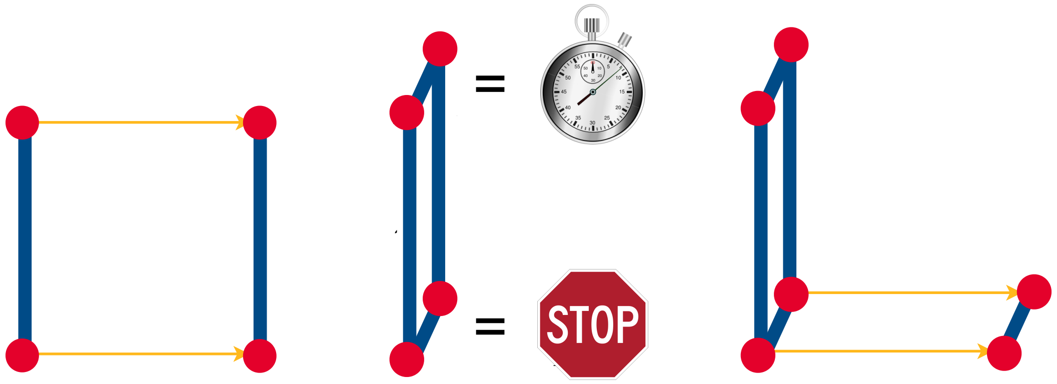
What if your compiler could be proved to *preserve* it?

Here's how!

Prove *confidentiality-preserving* refinement,



using a *decomposition principle*.



Compiler verification: Prior work

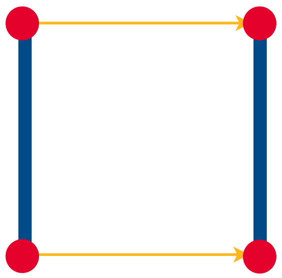
Using interactive theorem proving

- Jinja compiler (Java dialect) - verified in Isabelle/HOL
  (Klein and Nipkow, 2006)

(also)

+ JinjaThreads compiler (for multithreaded programs)
  (Lochbihler, 2010)

Note: “Usual” refinement



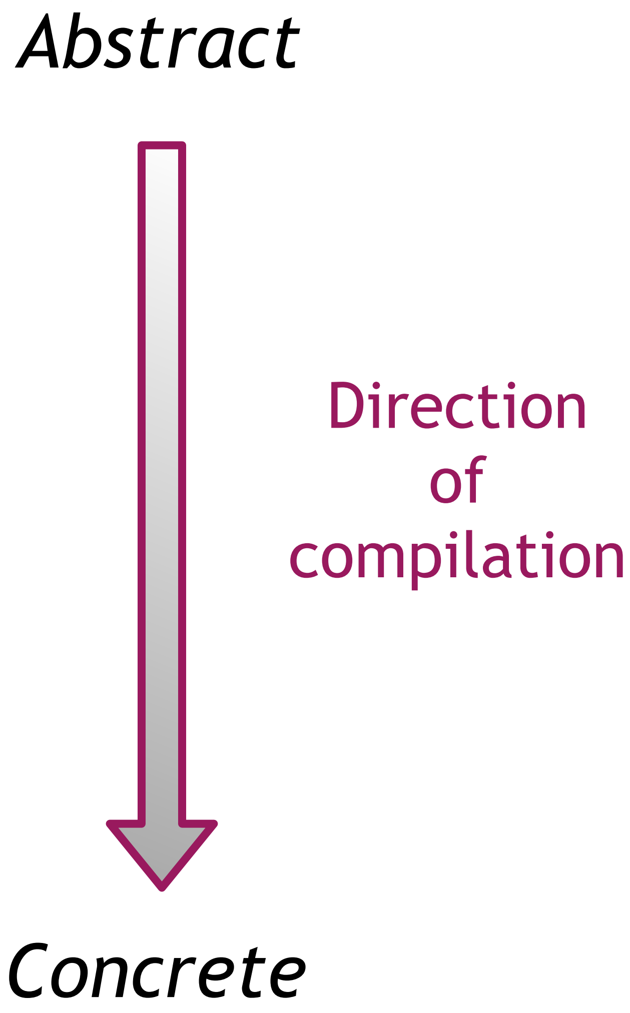
- CompCert C compiler - verified in Coq
  (Leroy, 2009)

- CakeML compiler (Standard ML dialect) - verified in HOL4
  (Kumar et al. 2014)

Compiler verification: Prior work

“Usual” refinement

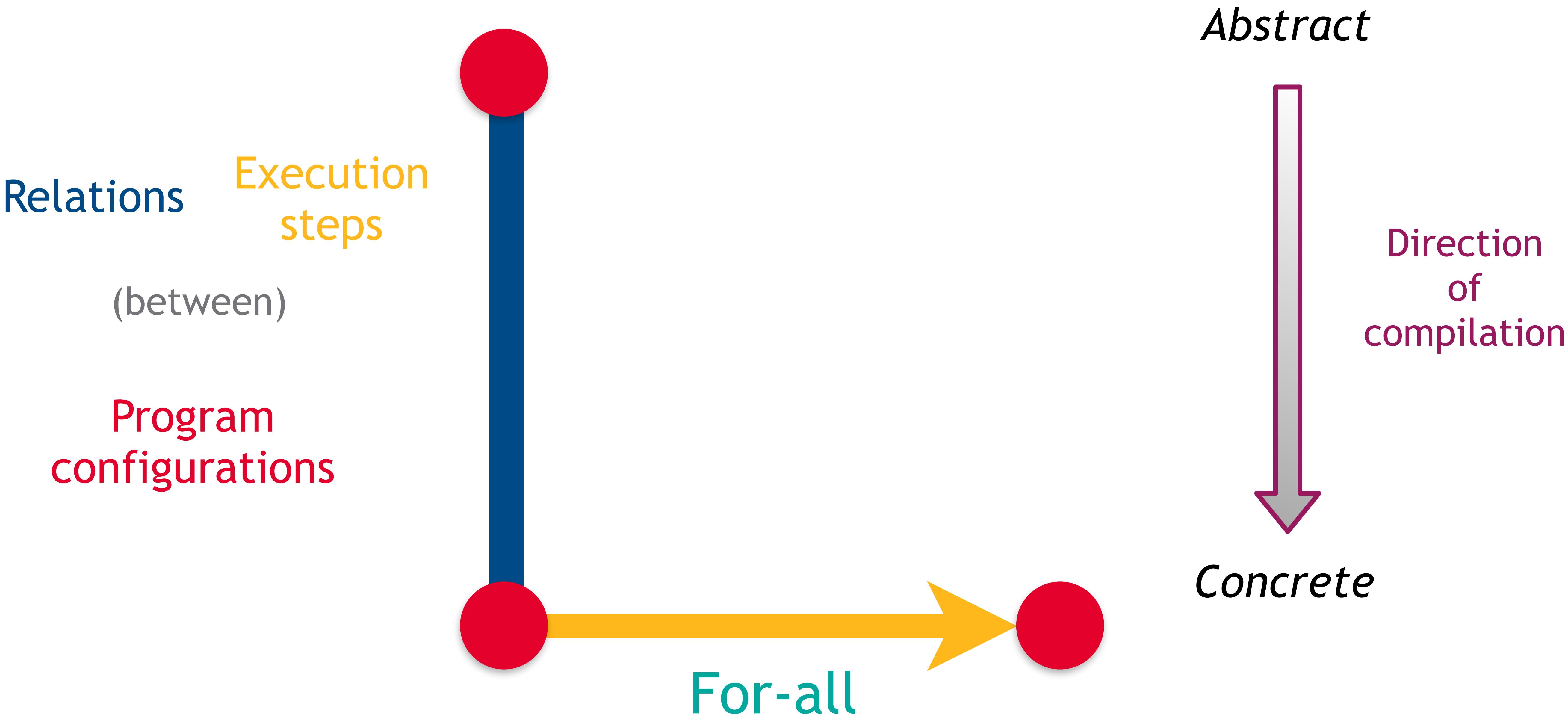
C refines A



Compiler verification: Prior work

“Usual” refinement

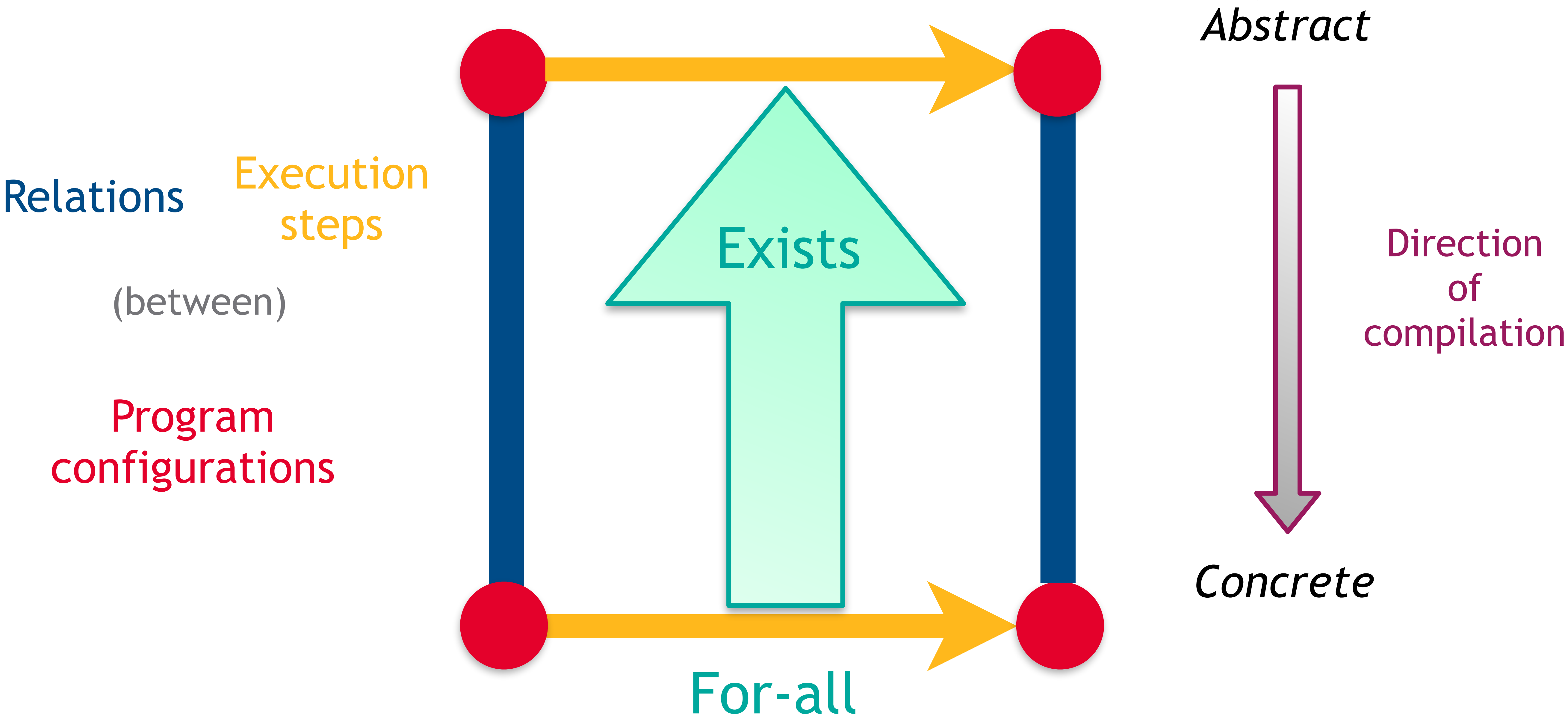
C refines A



Compiler verification: Prior work

“Usual” refinement

A simulates C ⇒ C refines A



Relations

Execution steps

(between)

Program configurations

Exists

For-all

Abstract

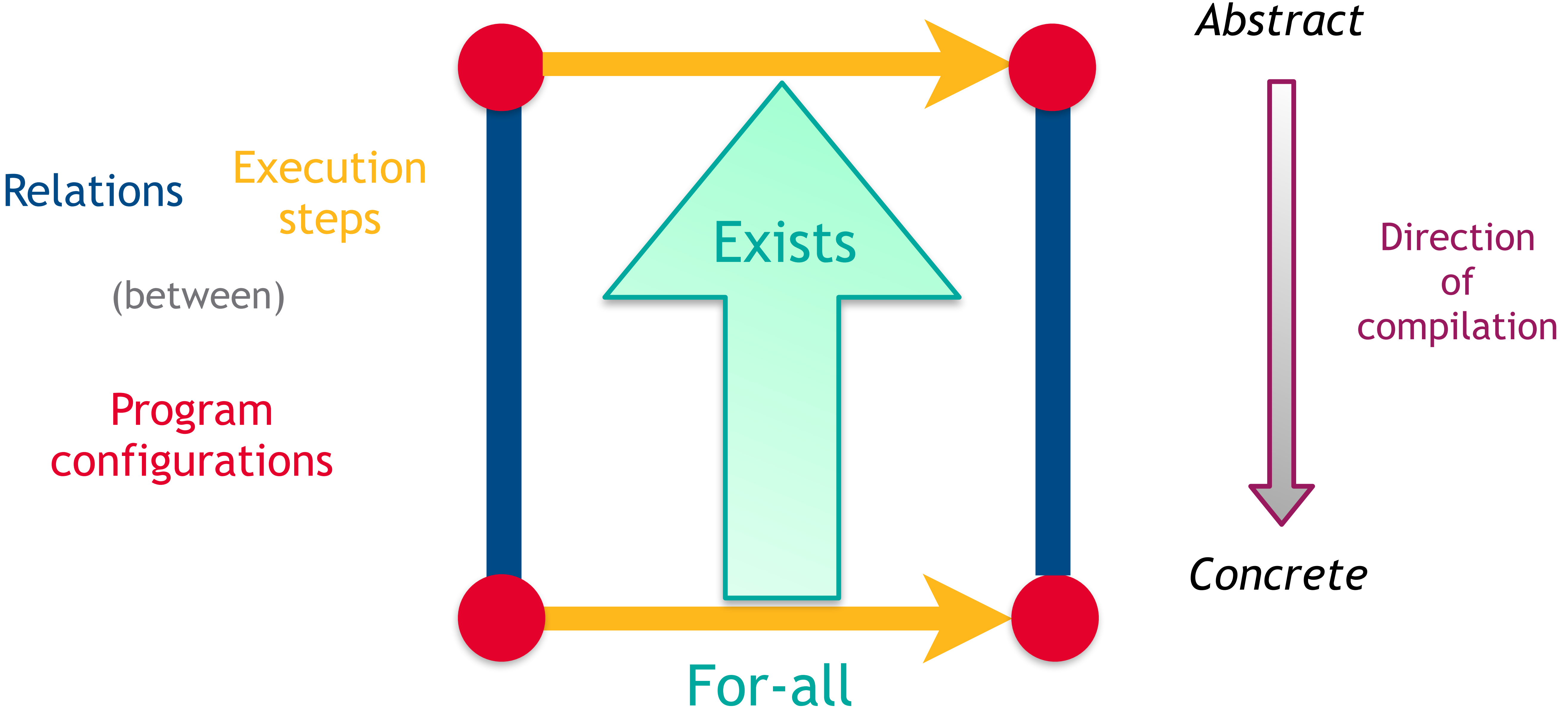
Direction of compilation

Concrete

Compiler verification: Prior work

Confidentiality-preserving refinement

(Murray, Sison, Pierzchalski, Rizkallah 2016)



Compiler verification: Prior work

Confidentiality-preserving refinement

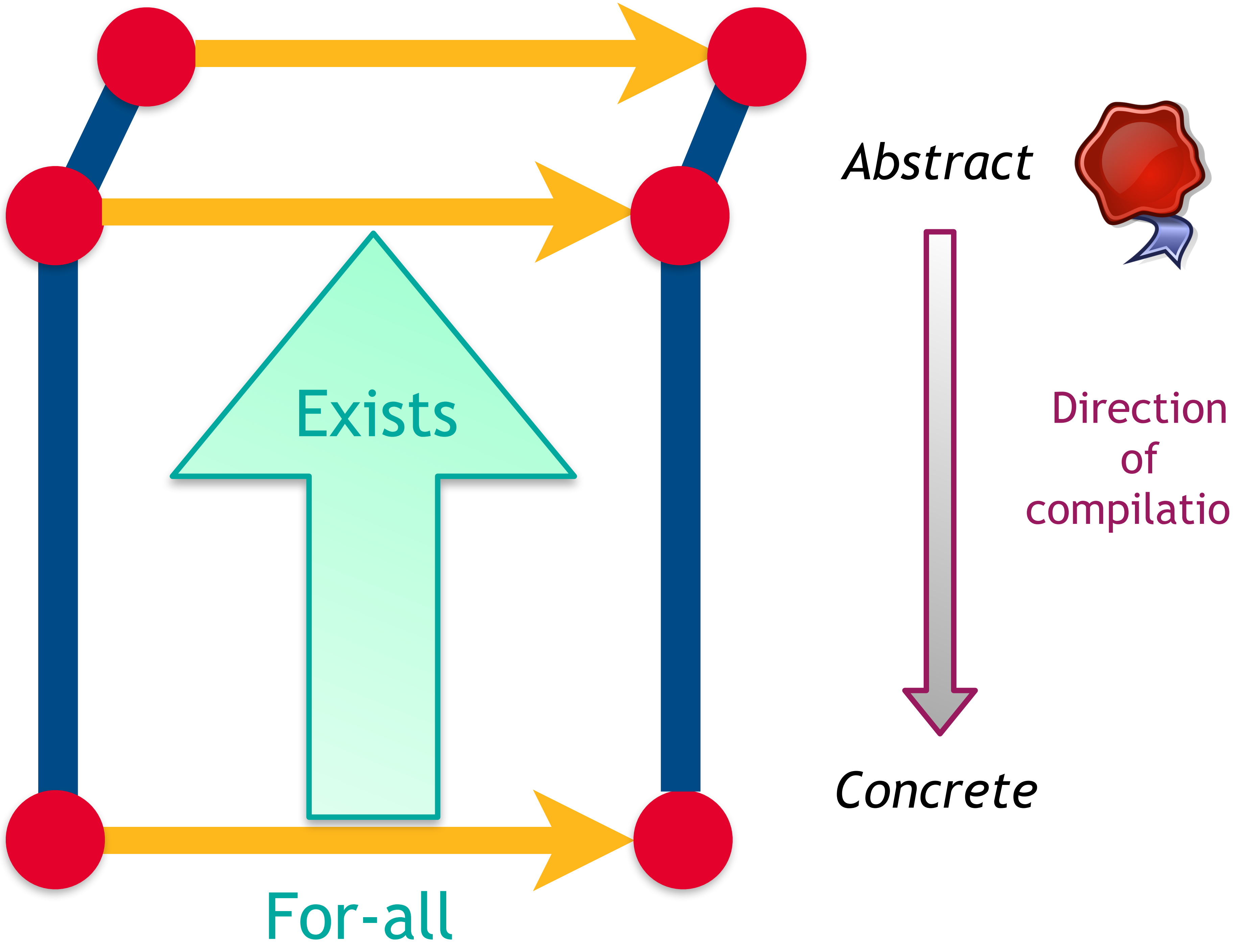
(Murray, Sison, Pierzchalski, Rizkallah 2016)

Relations

Execution steps

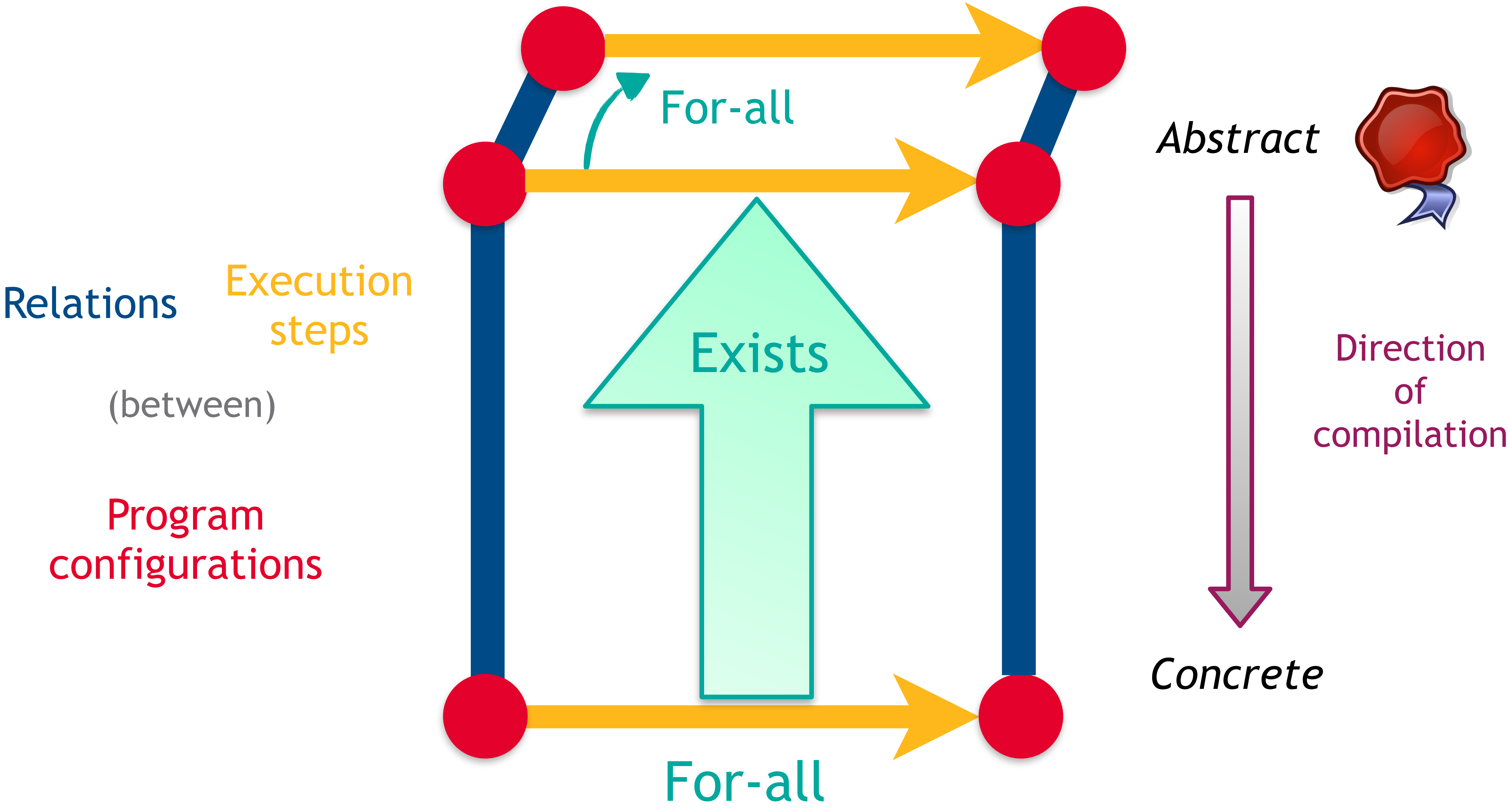
(between)

Program configurations



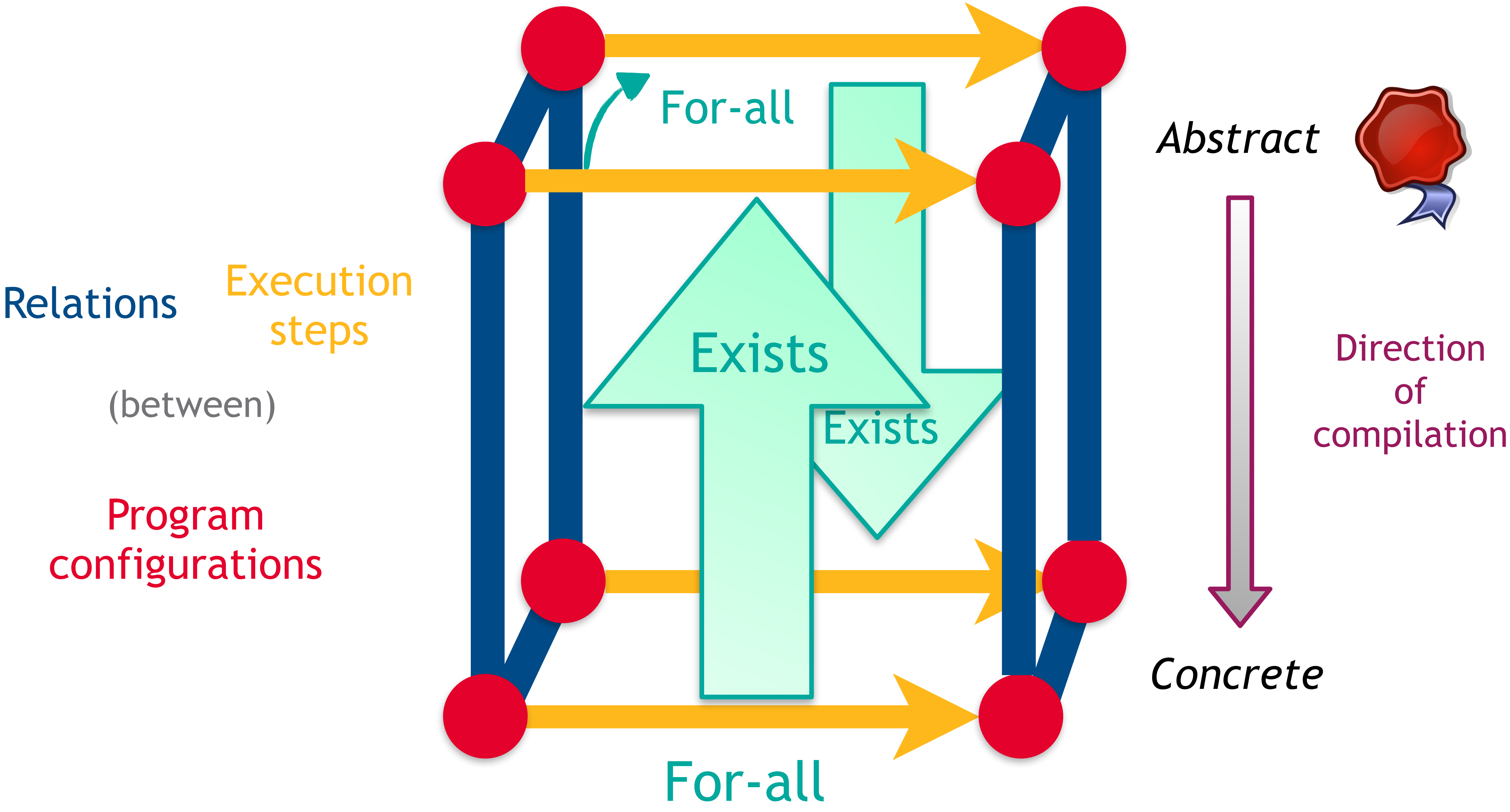
Compiler verification: Prior work

Confidentiality-preserving refinement (Murray, Sison, Pierzchalski, Rizkallah 2016)



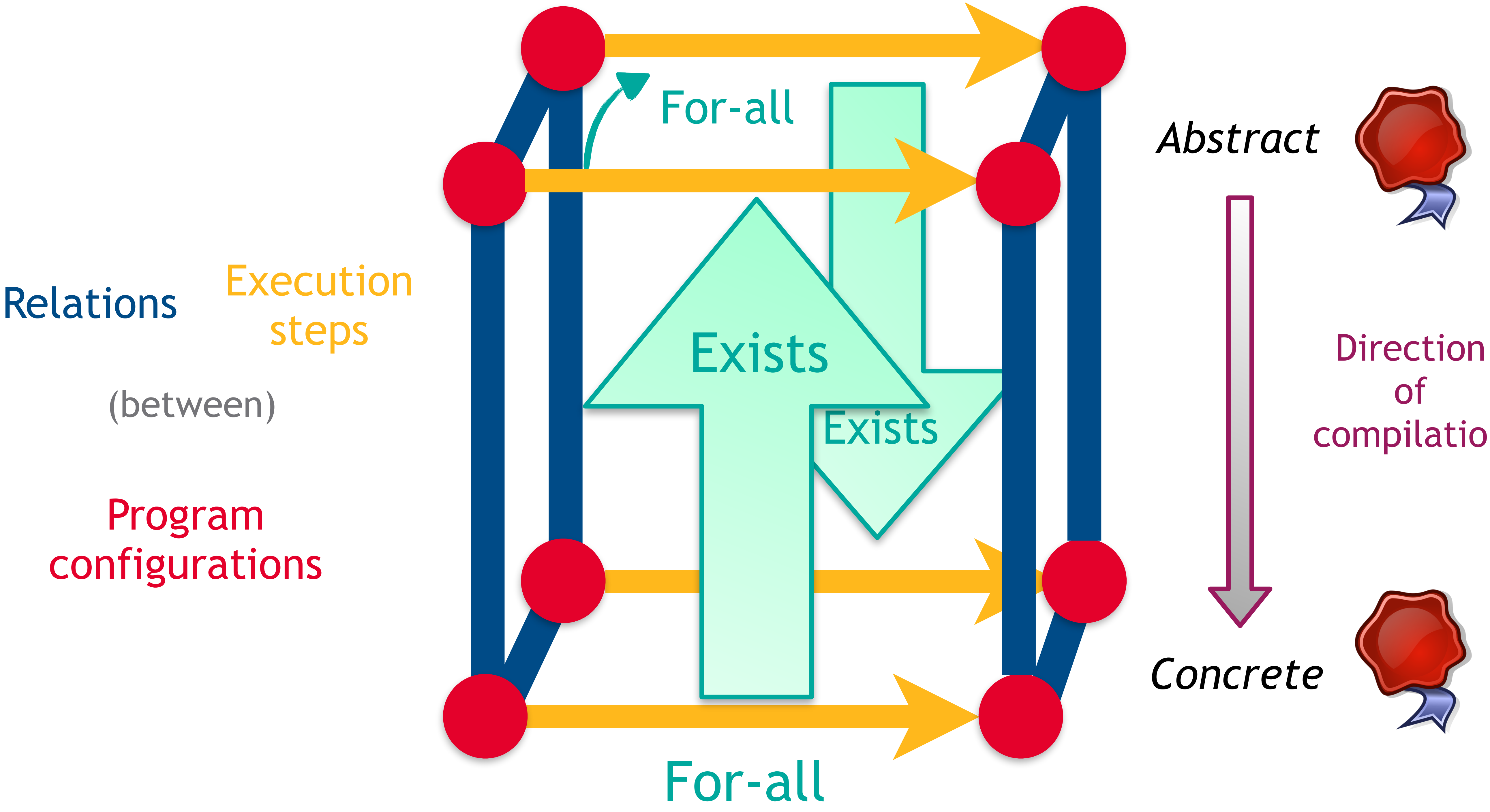
Compiler verification: Prior work

Confidentiality-preserving refinement (Murray, Sison, Pierzchalski, Rizkallah 2016)



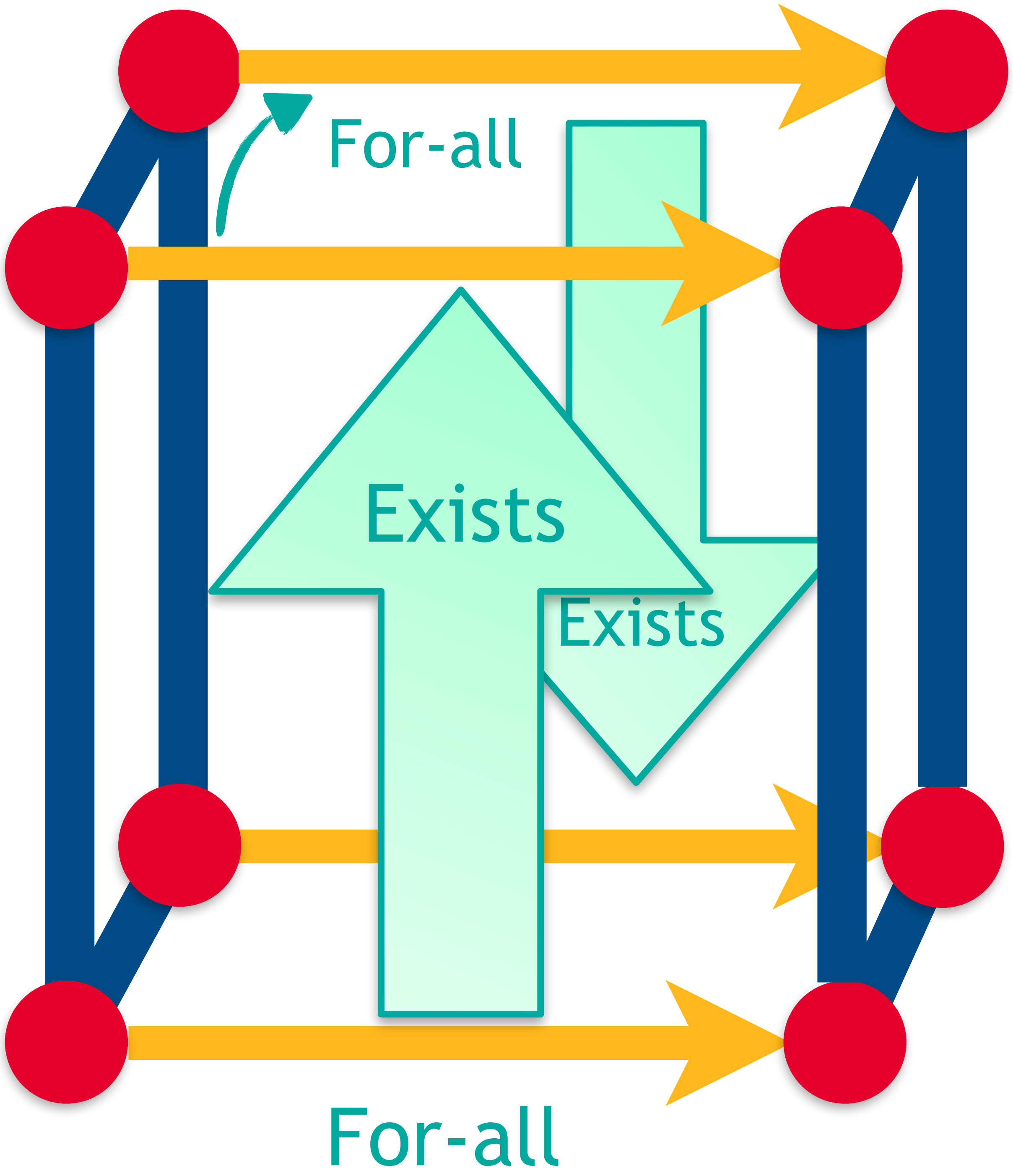
Compiler verification: Prior work

Confidentiality-preserving refinement (Murray, Sison, Pierzchalski, Rizkallah 2016)



Compiler verification: My work

(Compiler based on Tedesco et al. 2016)



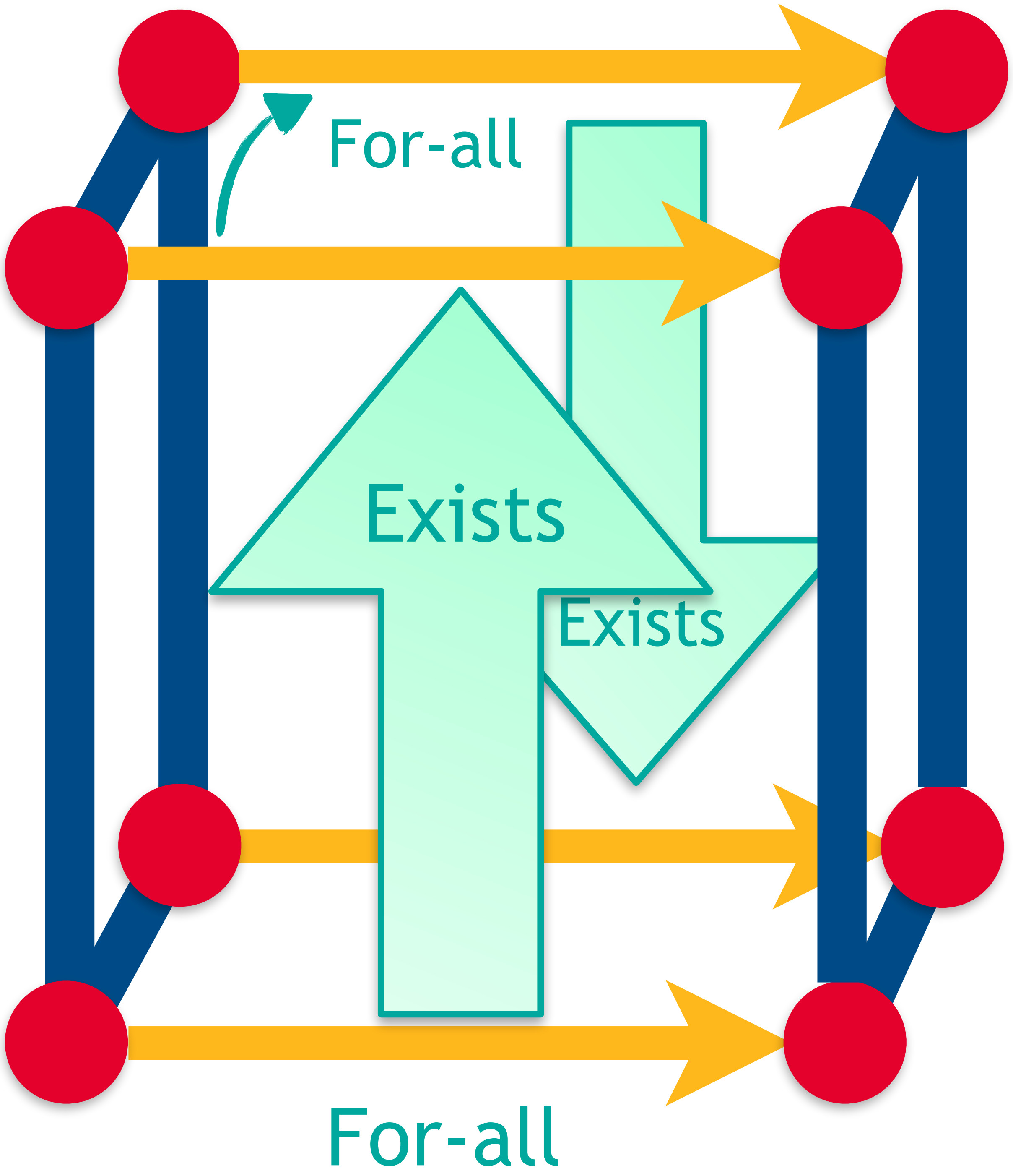
Abstract

Direction of compilation

Concrete

Compiler verification: My work

(Compiler based on Tedesco et al. 2016)



Generic

While language

with locks

Direction of compilation

Generic

RISC assembly

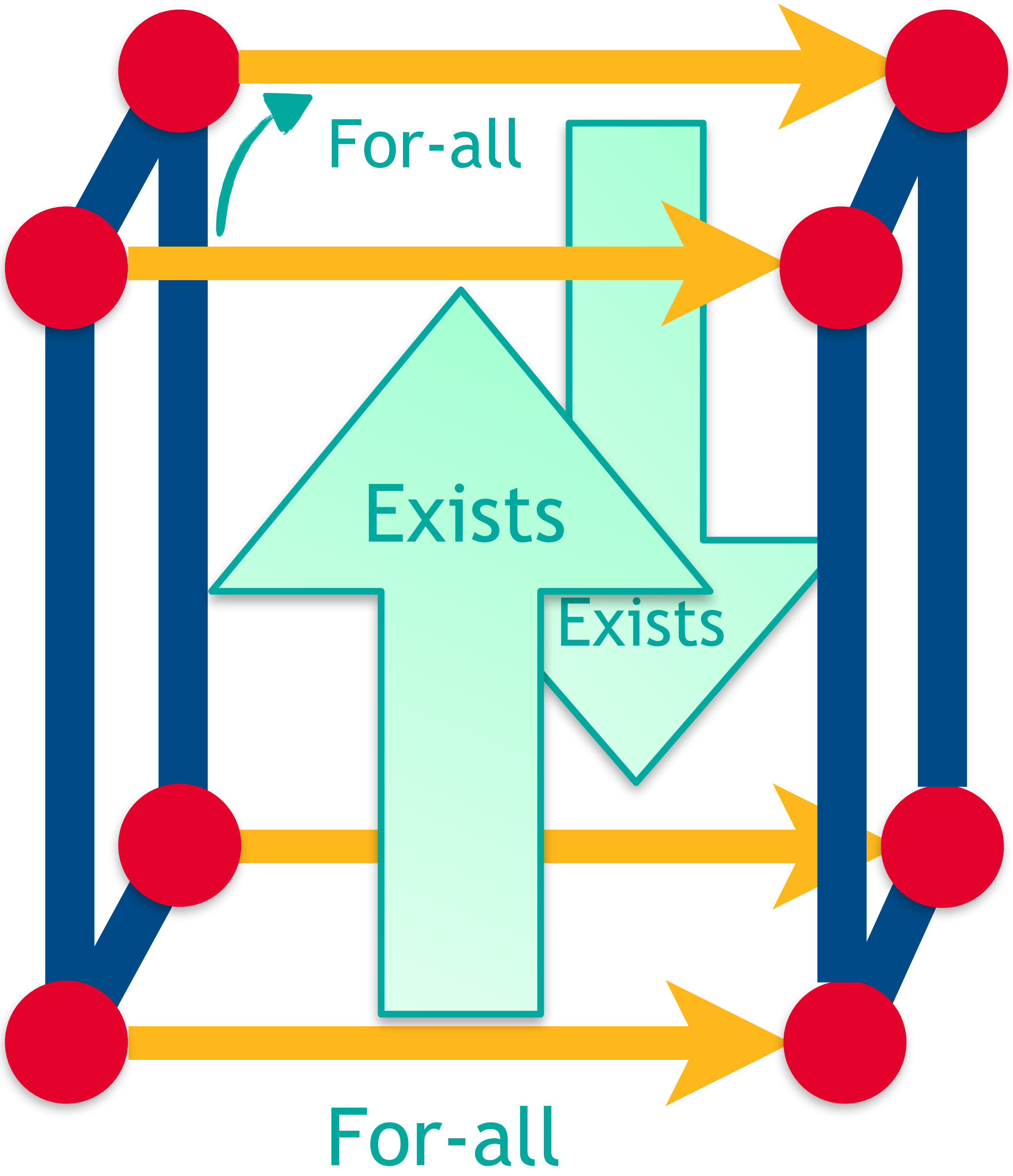
with locks

Compiler verification: My work

(Compiler based on Tedesco et al. 2016)

Using a *decomposition principle*

(Thanks to Toby Murray)



Generic

While language

with locks

Direction of compilation

Generic

RISC assembly

with locks

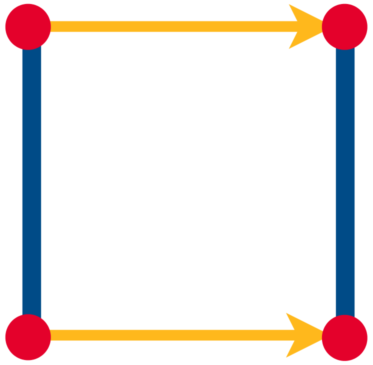
Compiler verification: My work

(Compiler based on Tedesco et al. 2016)

Using a *decomposition principle*

(Thanks to Toby Murray)

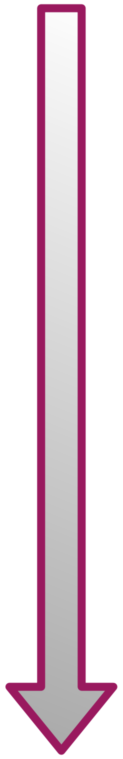
- The “usual” refinement, with no changes to locking:



Generic

While language

with locks

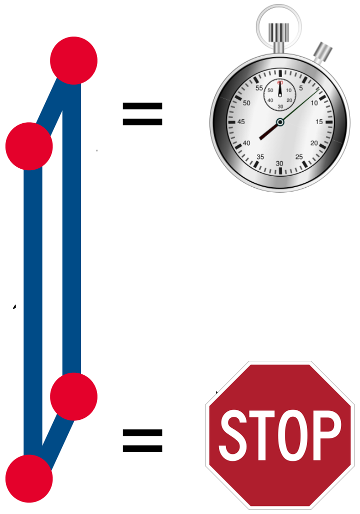


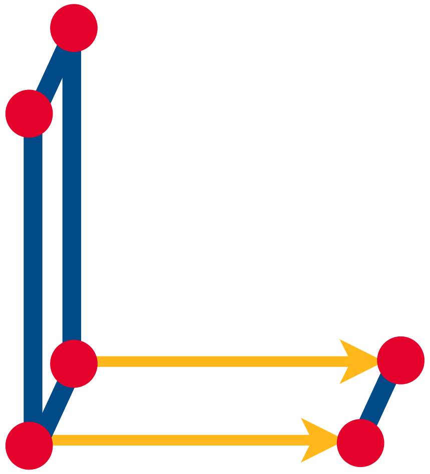
Direction of compilation

Generic

RISC assembly

with locks





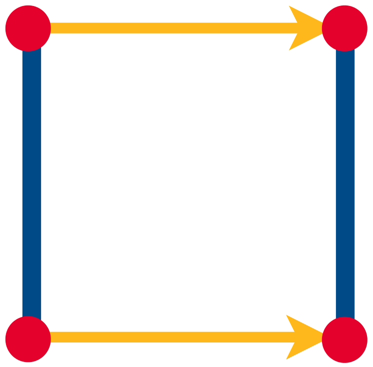
Compiler verification: My work

(Compiler based on Tedesco et al. 2016)

Using a *decomposition principle*

(Thanks to Toby Murray)

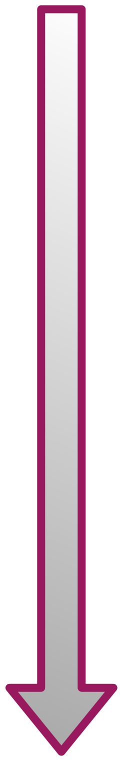
- The “usual” refinement, with no changes to locking:
  - + Using knowledge that spaces are locked



Generic

While language

with locks

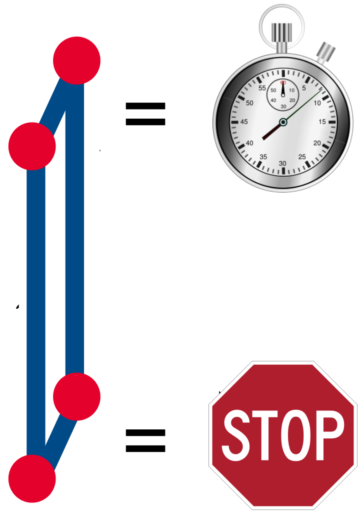


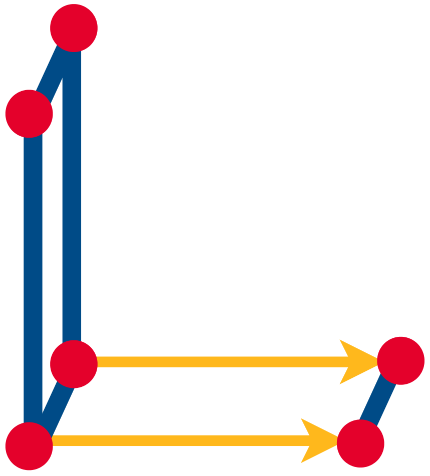
Direction of compilation

Generic

RISC assembly

with locks





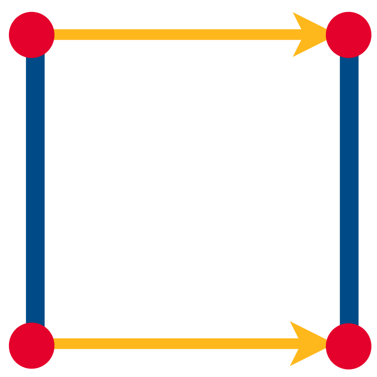
Compiler verification: My work

(Compiler based on Tedesco et al. 2016)

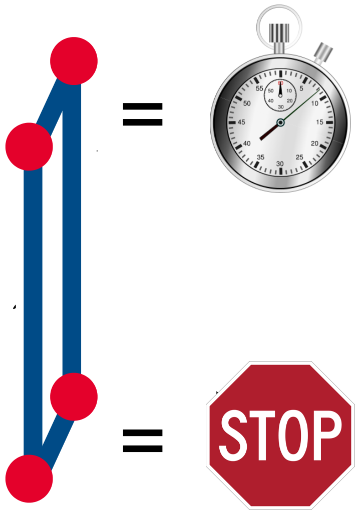
Using a *decomposition principle*

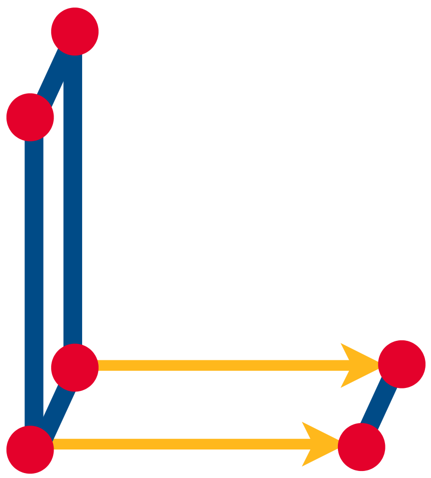
(Thanks to Toby Murray)

- The “usual” refinement, with no changes to locking:
  - + Using knowledge that spaces are locked

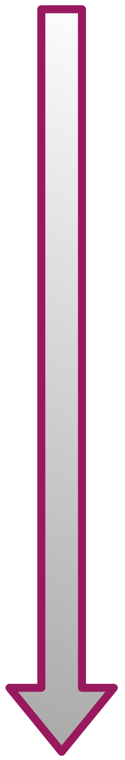


- No new *timing*, *stopping*, or *branching* based on secret information:





Generic While language with locks



Direction of compilation

Generic RISC assembly with locks

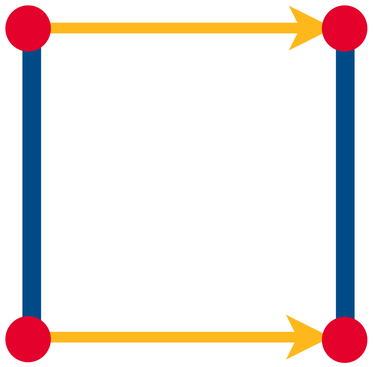
Compiler verification: My work

(Compiler based on Tedesco et al. 2016)

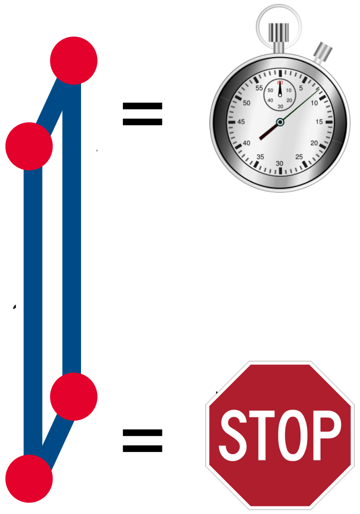
Using a *decomposition principle*

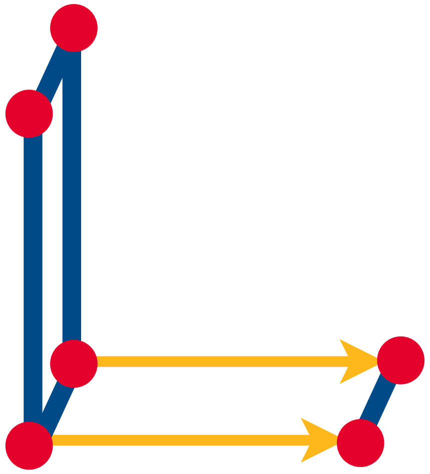
(Thanks to Toby Murray)

- The “usual” refinement, with no changes to locking:
  - + Using knowledge that spaces are locked

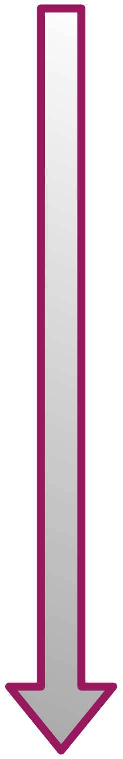


- No new *timing*, *stopping*, or *branching* based on secret information:





Generic
While language
with locks



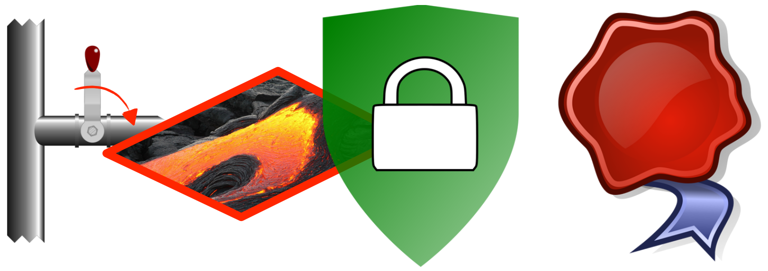
Direction of compilation

Generic
RISC assembly
with locks

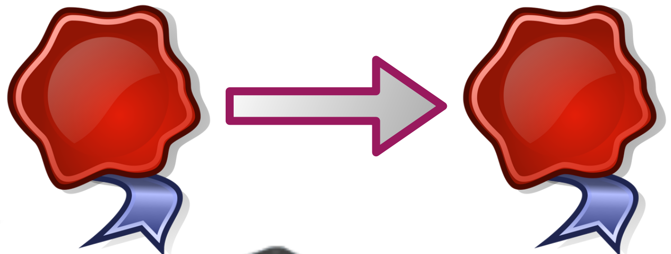
Thesis (PhD, 2020)

That “Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs” is feasible.

- ~~Program verification~~ Chapter 4



- ~~Compiler verification~~ Chapter 5



- Case study: CDDC Chapter 6

- Extension to program verification Chapter 7

Case study

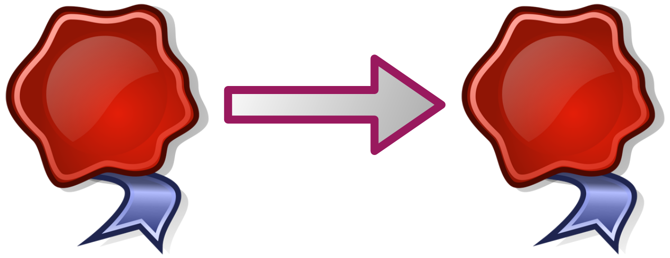
Methodology in Isabelle/HOL

Program verification

- Assign locks to spaces
- For each thread, prove:
  - Local compliance
    “I respect the locks”
  - Local security
    “I win ‘floor is lava with levers’ using locks”

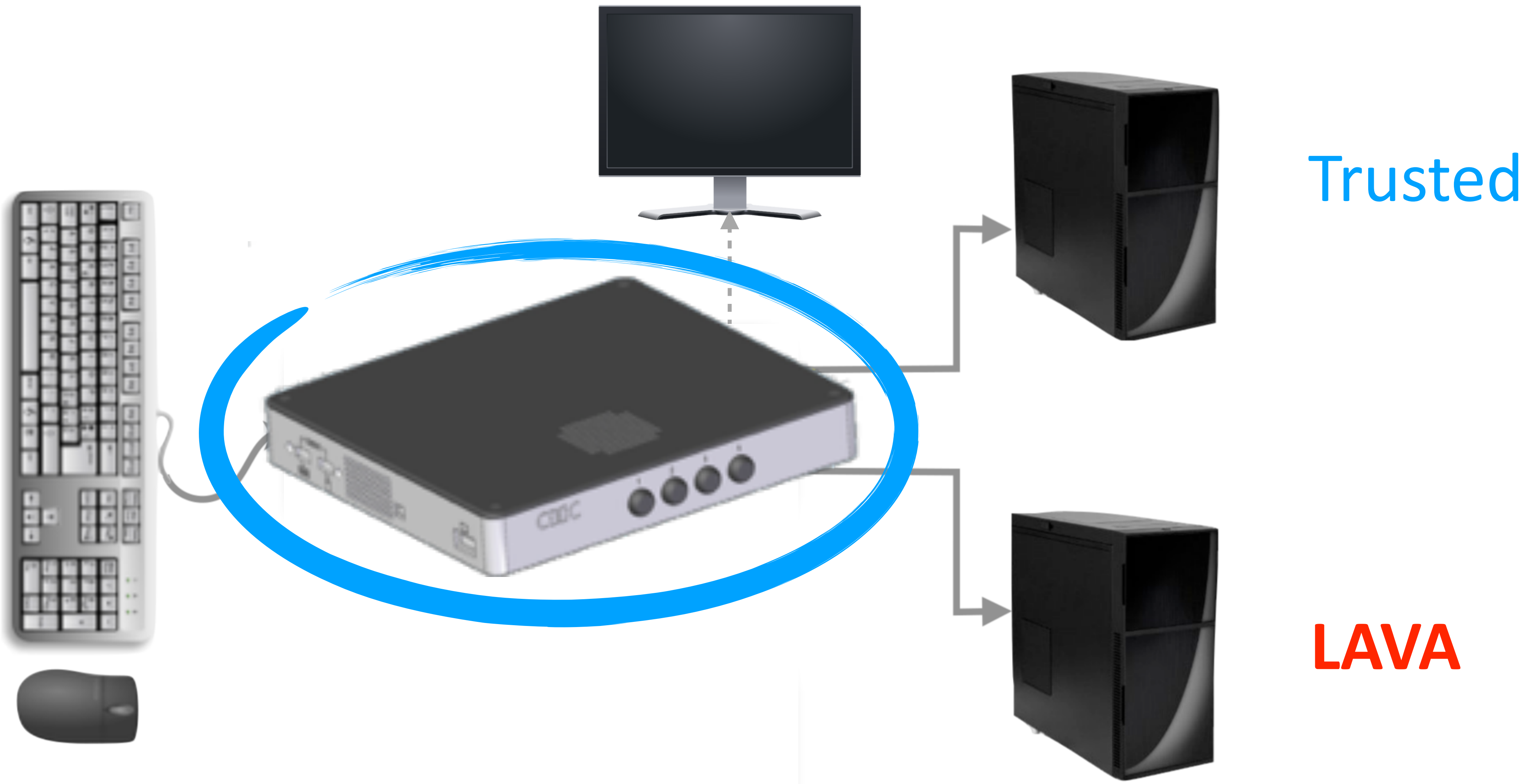
- ~~Global compatibility~~
  ~~“Respecting locks is enough for guarantees to meet assumptions”~~

Compiler application



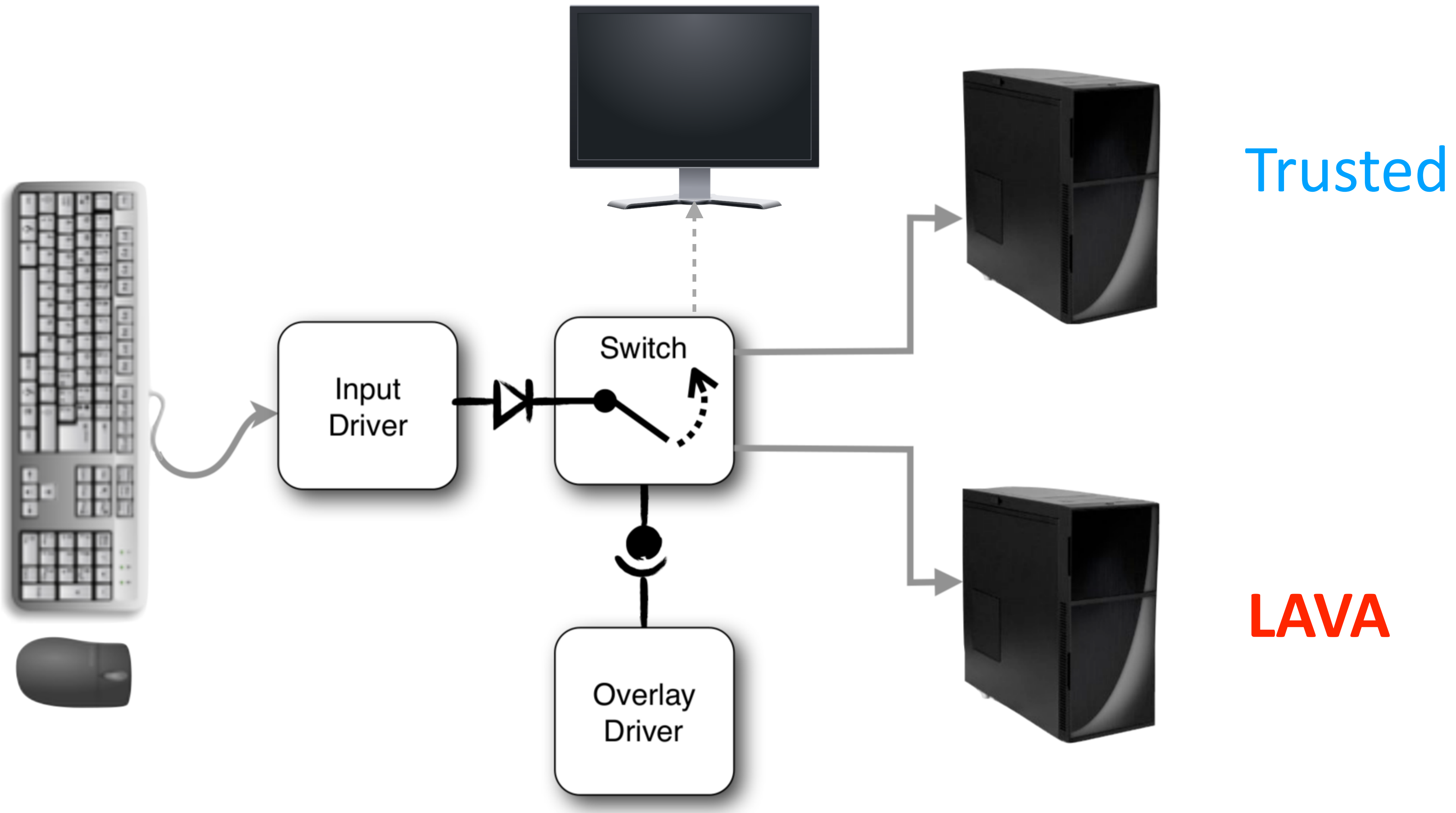
Case study

Cross Domain Desktop Compositor HID switch



Case study

Cross Domain Desktop Compositor HID switch

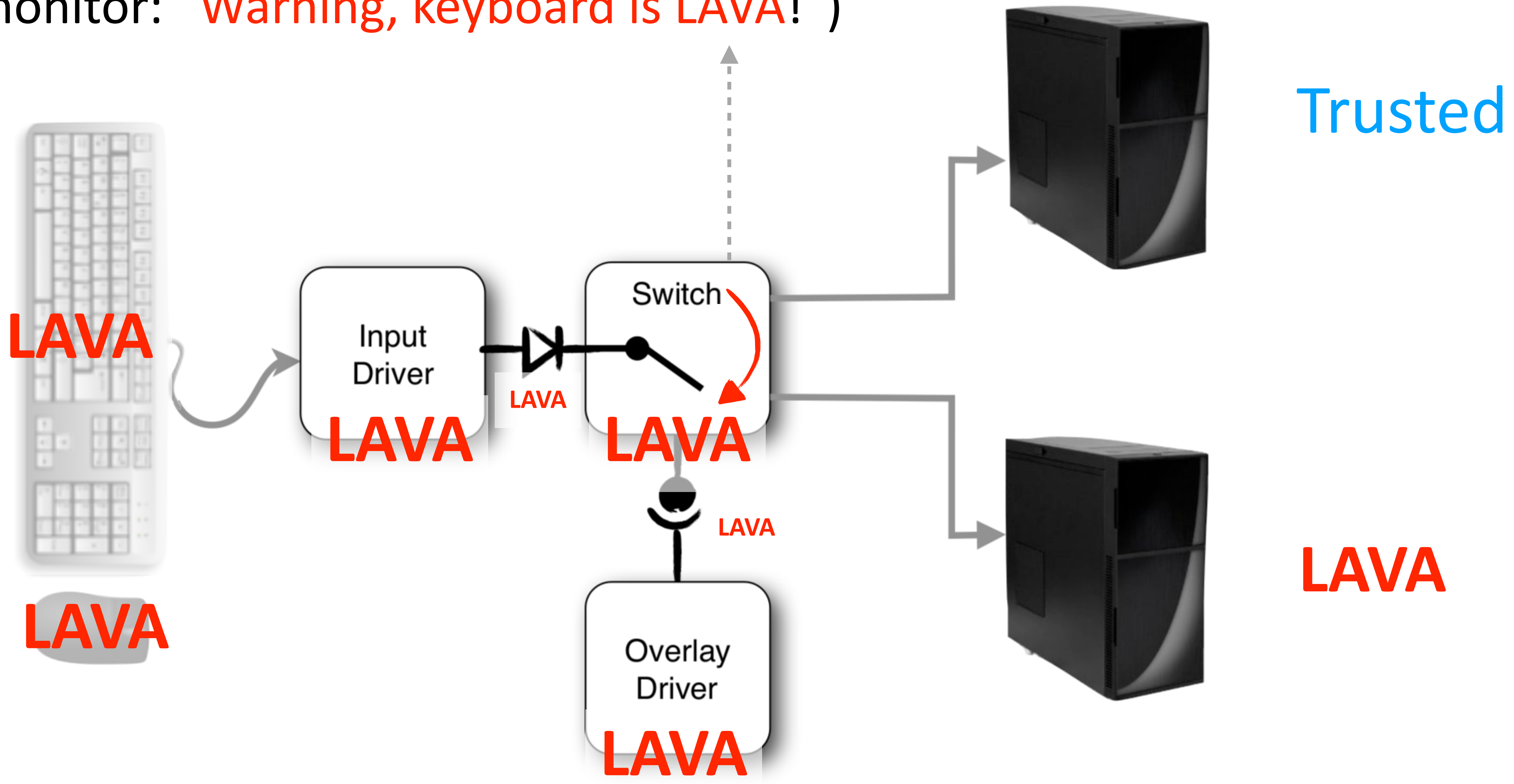


seL4 component architecture, functional schematic

Case study

Cross Domain Desktop Compositor HID switch

(On video monitor: “Warning, keyboard is LAVA!”)

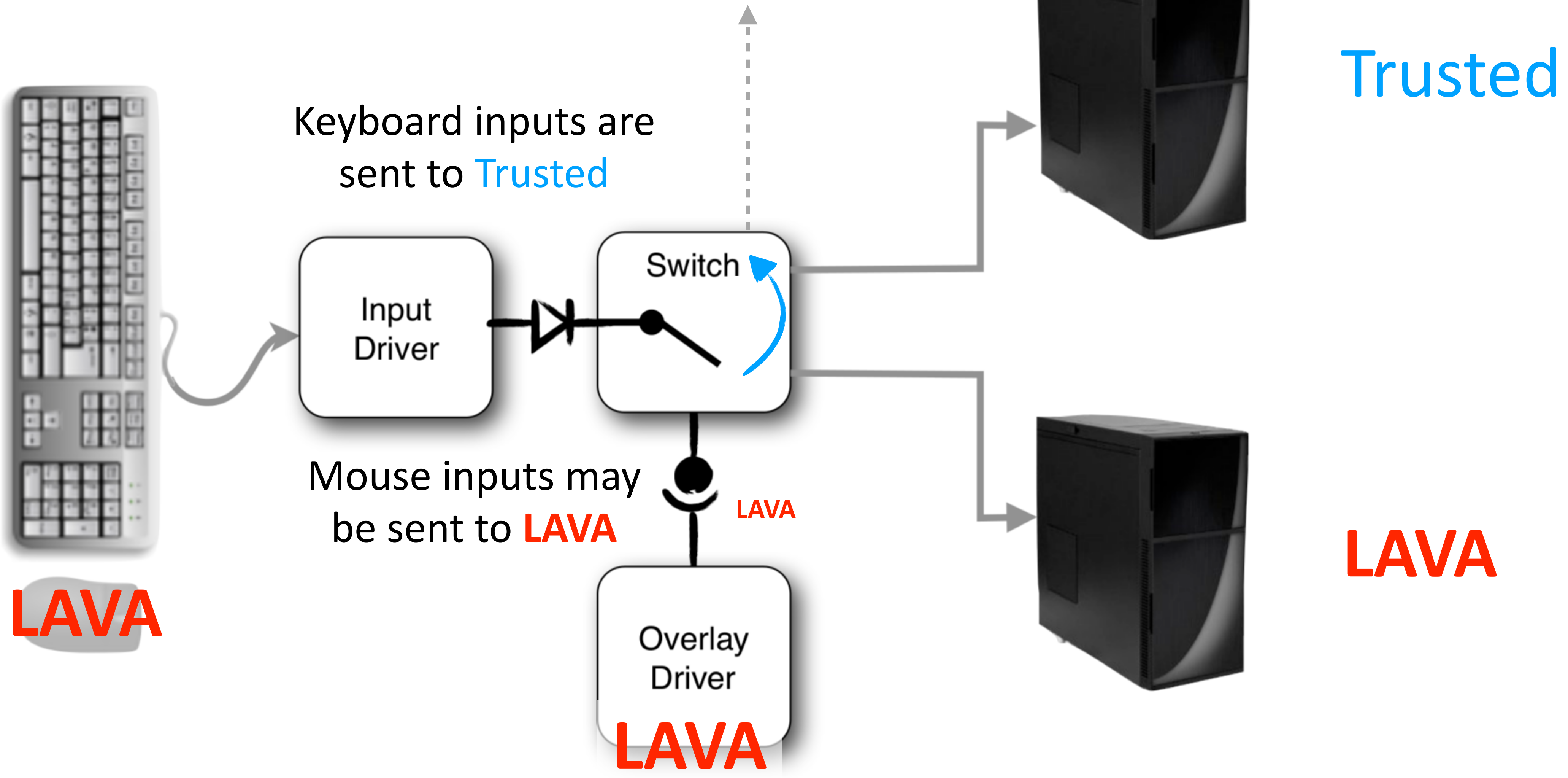


seL4 component architecture, functional schematic

Case study

Cross Domain Desktop Compositor HID switch

(On video monitor: “Keyboard inputs can be secret”)



Keyboard inputs are sent to Trusted

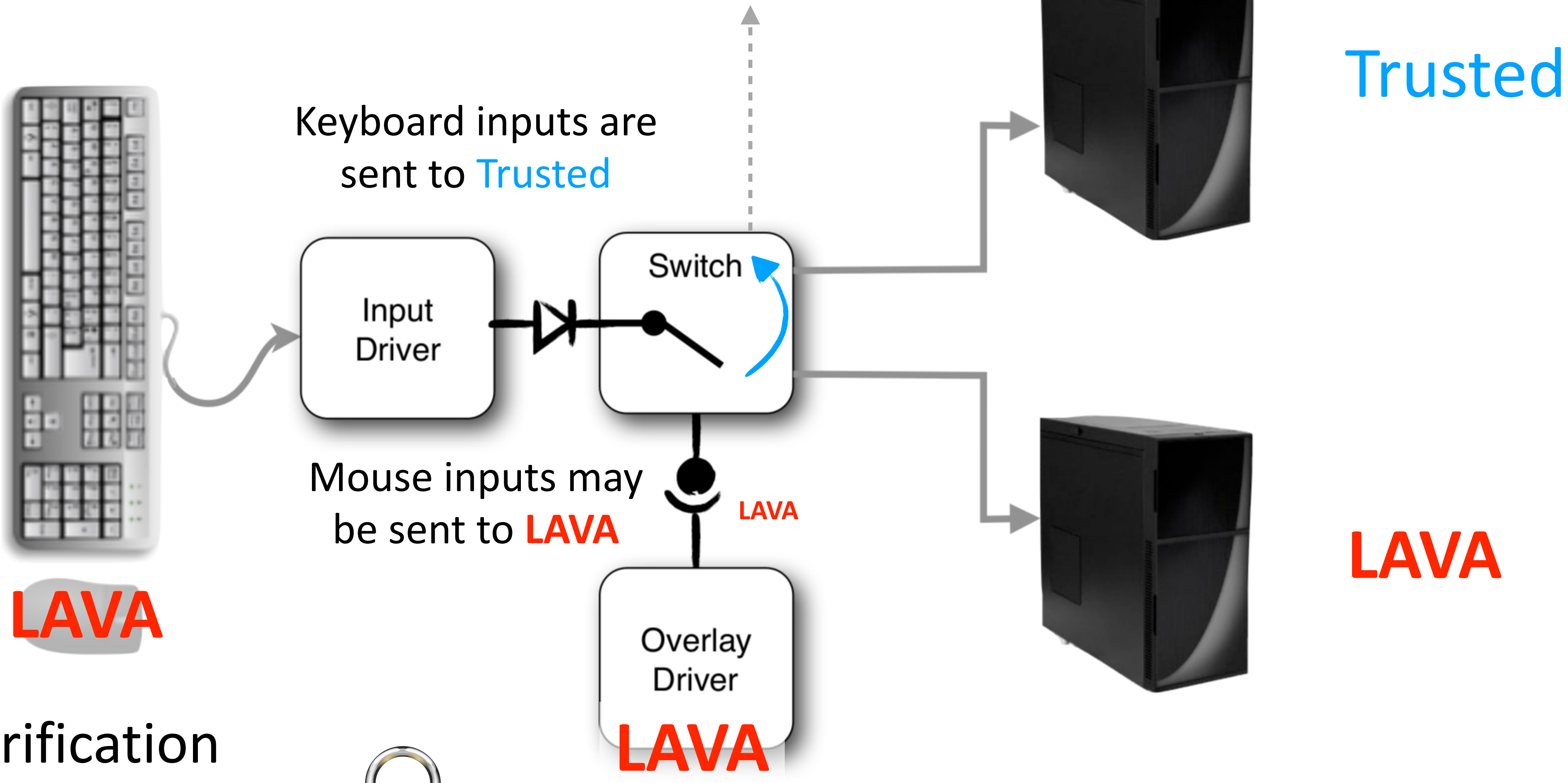
Mouse inputs may be sent to LAVA

seL4 component architecture, functional schematic

Case study

Cross Domain Desktop Compositor HID switch

(On video monitor: “Keyboard inputs can be secret”)



Keyboard inputs are sent to Trusted

Mouse inputs may be sent to LAVA

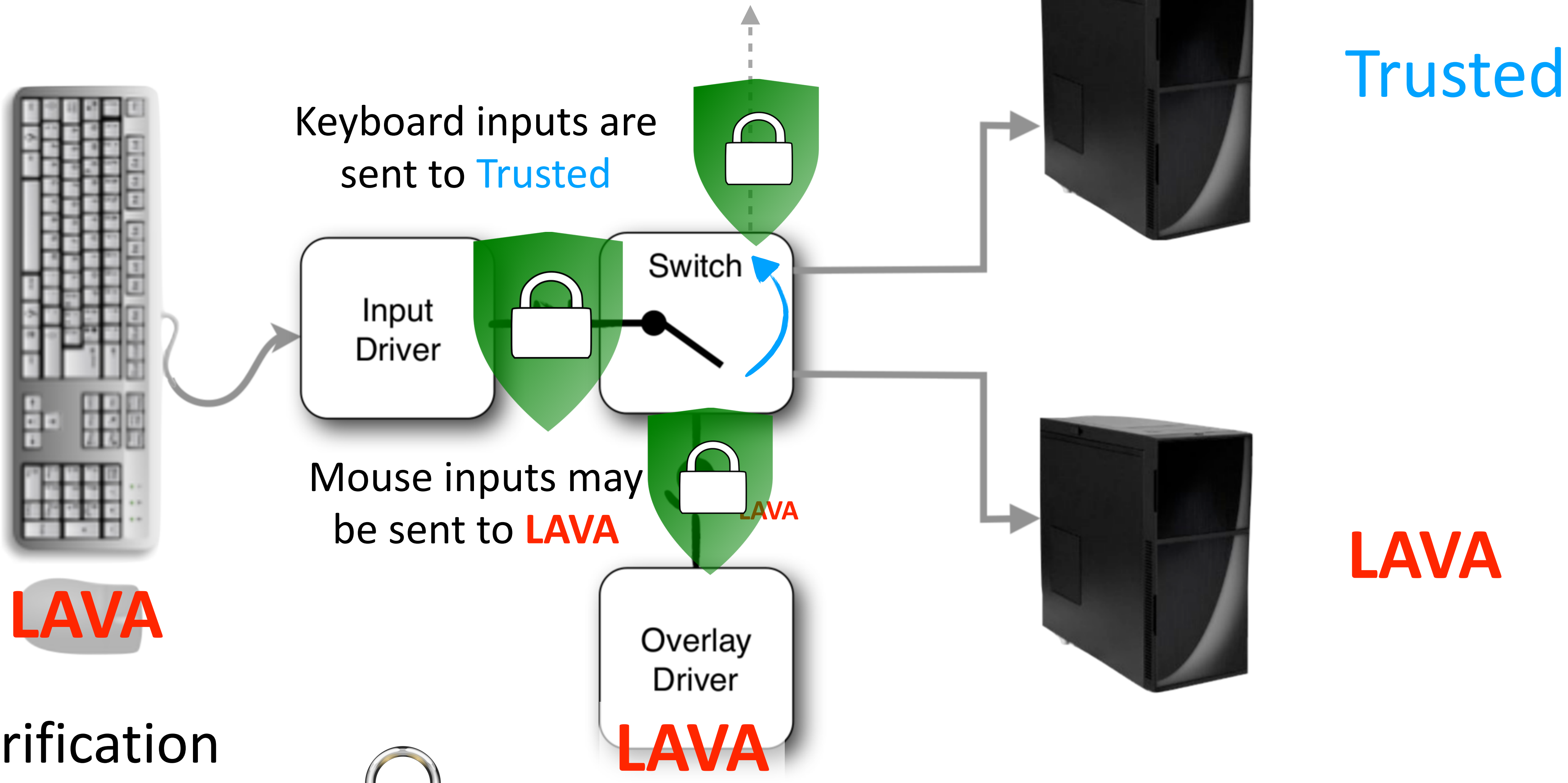
Program verification

- Assign locks to spaces

Case study

Cross Domain Desktop Compositor HID switch

(On video monitor: “Keyboard inputs can be secret”)



Keyboard inputs are sent to Trusted

Mouse inputs may be sent to LAVA

Trusted

LAVA

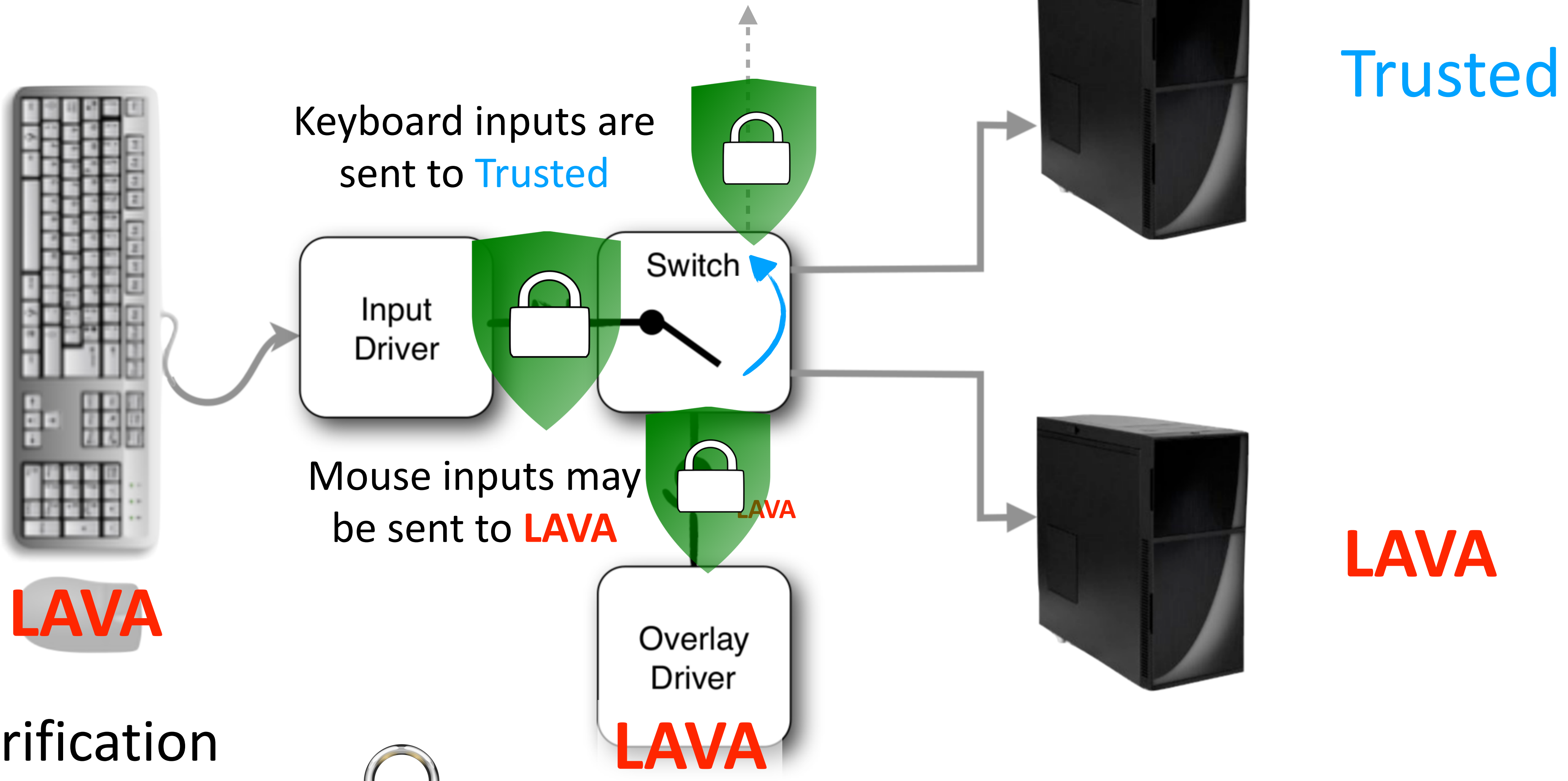
Program verification

- Assign locks to spaces

Case study

Cross Domain Desktop Compositor HID switch

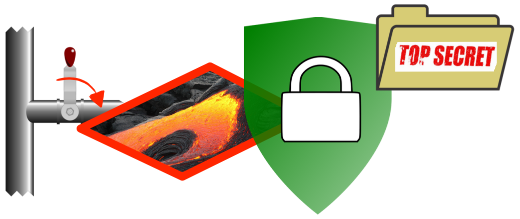
(On video monitor: “Keyboard inputs can be secret”)



Keyboard inputs are sent to Trusted
Input Driver
Switch
Trusted
Mouse inputs may be sent to LAVA
LAVA
LAVA
Overlay Driver
LAVA
LAVA

Program verification

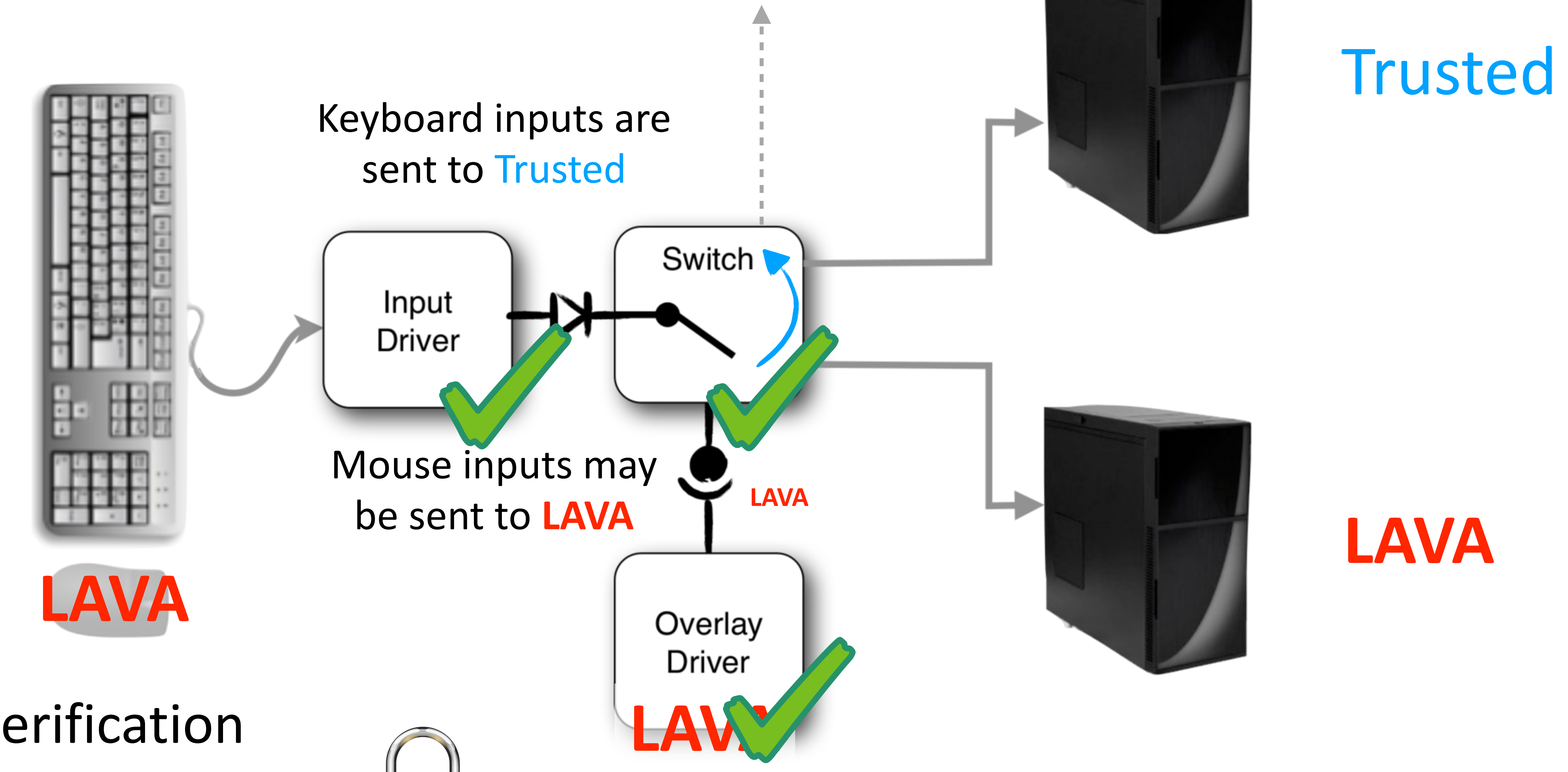
- Assign locks to spaces
- For each thread, prove:
  - Local compliance
    “I respect the locks”
  - Local security
    “I win ‘floor is lava with levers and locks’”



Case study

Cross Domain Desktop Compositor HID switch

(On video monitor: “Keyboard inputs can be secret”)



Keyboard inputs are sent to Trusted

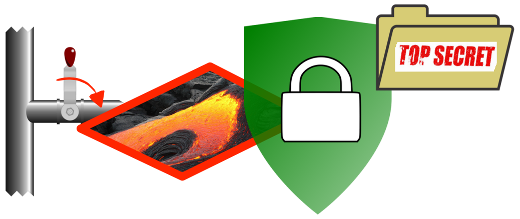
Mouse inputs may be sent to LAVA

Trusted

LAVA

Program verification

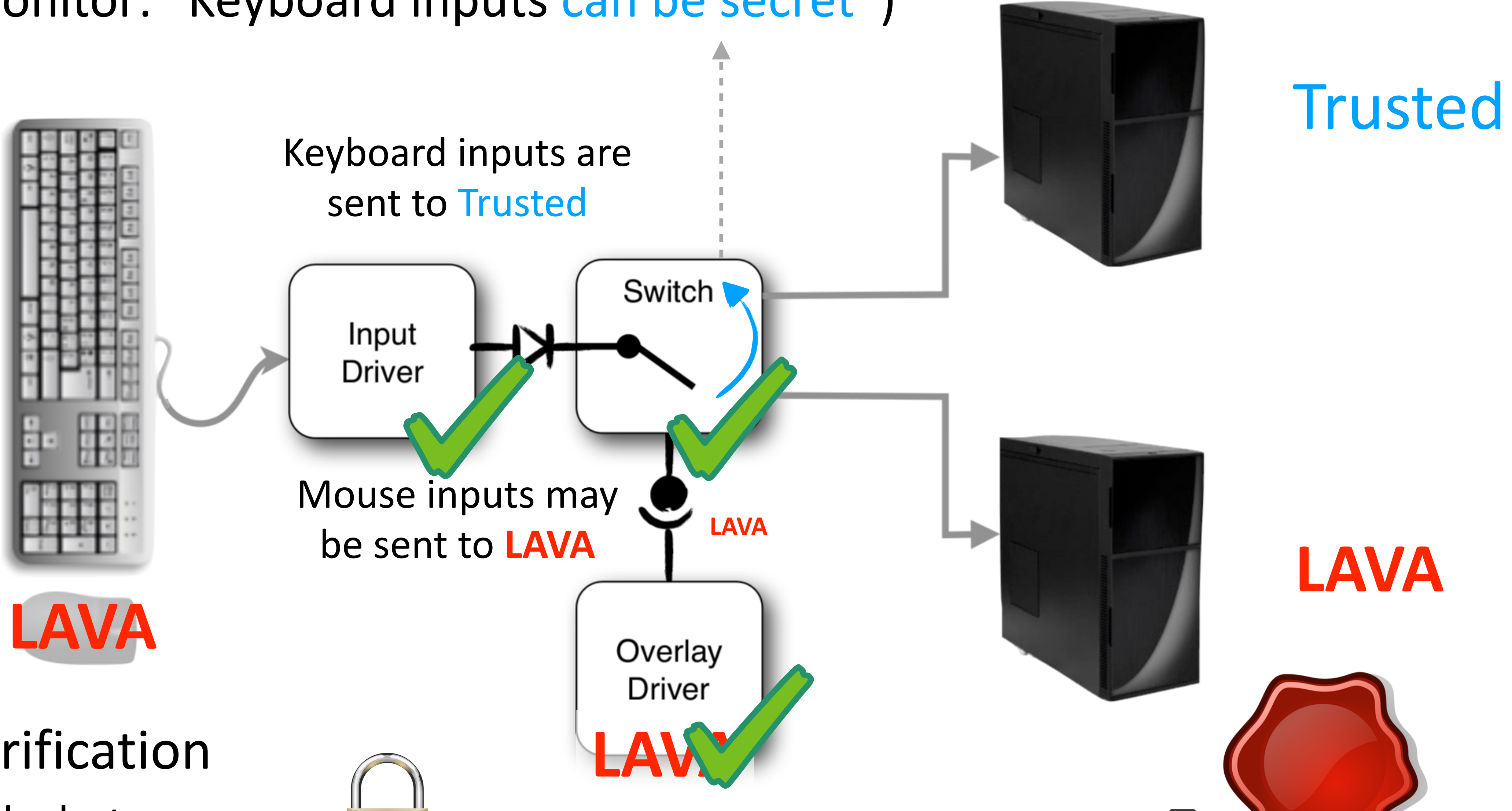
- Assign locks to spaces
- For each thread, prove:
  - Local compliance
    “I respect the locks”
  - Local security
    “I win ‘floor is lava with levers and locks’”



Case study

Cross Domain Desktop Compositor HID switch

(On video monitor: “Keyboard inputs can be secret”)



Keyboard inputs are sent to Trusted

Mouse inputs may be sent to LAVA

Input Driver

Switch

Overlay Driver

LAVA

Trusted

LAVA

LAVA

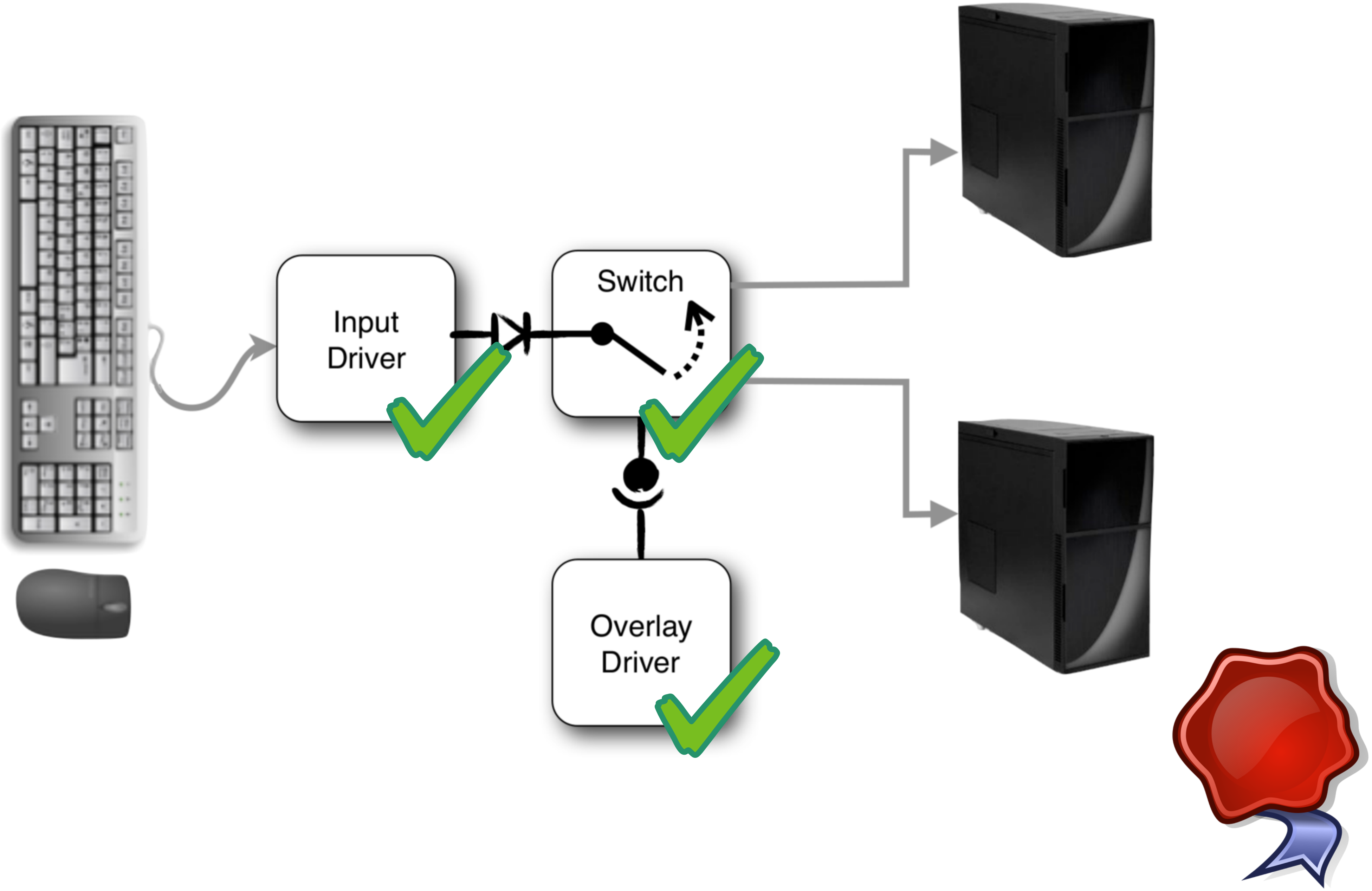
LAVA

Program verification

- Assign locks to spaces
- For each thread, prove:
  - Local compliance
    “I respect the locks”
  - Local security
    “I win ‘floor is lava with levers and locks’”

Case study

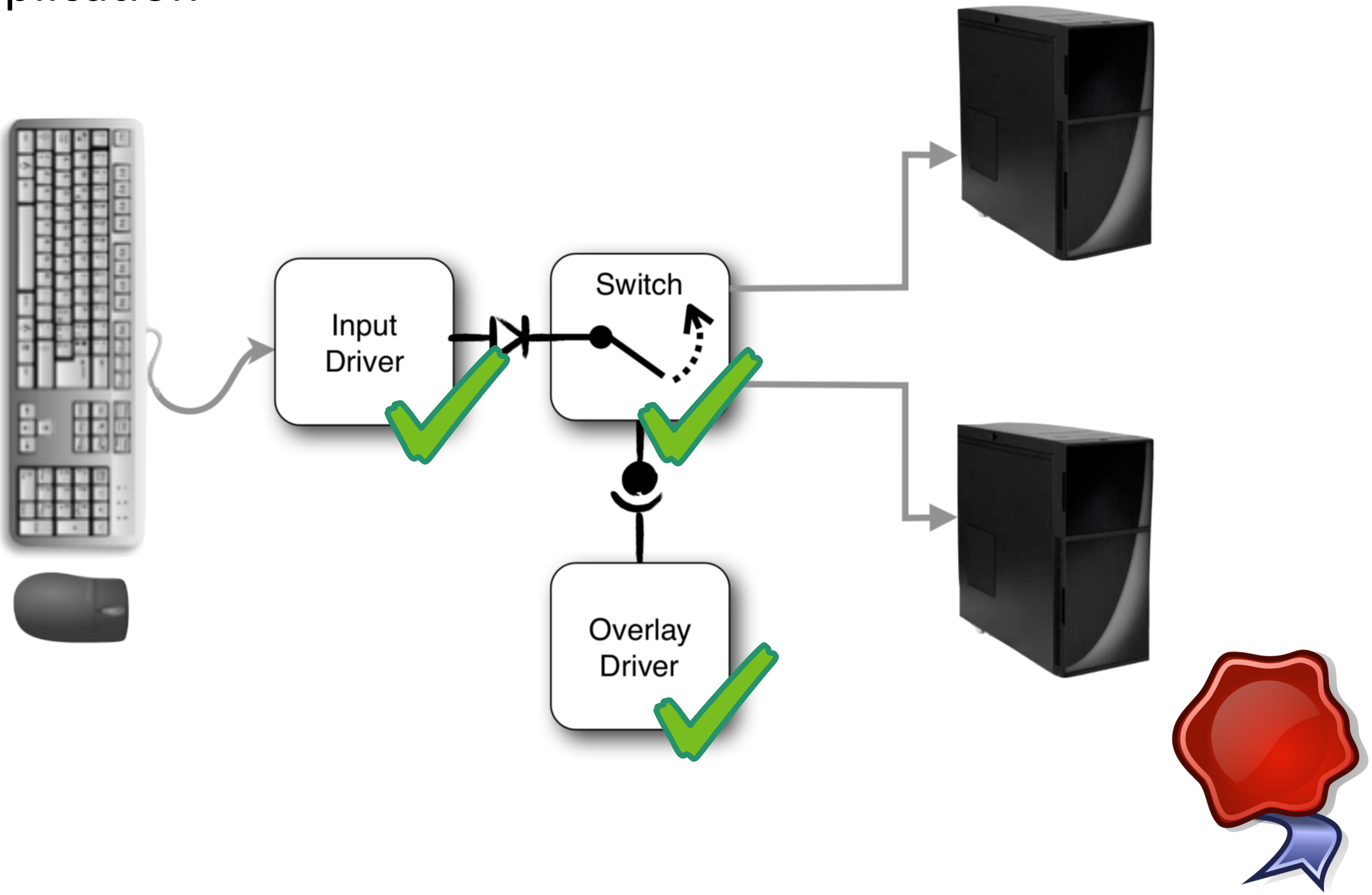
Cross Domain Desktop Compositor HID switch



Case study

Cross Domain Desktop Compositor HID switch

Compiler application

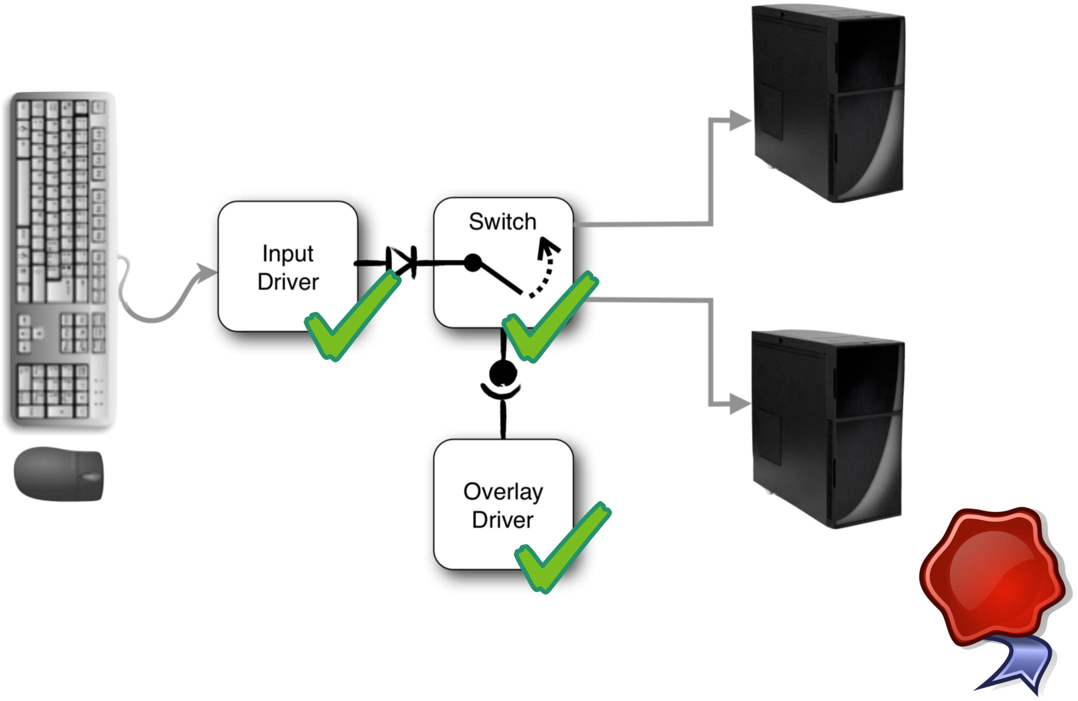


Case study

Cross Domain Desktop Compositor HID switch

Compiler application

While language with locks



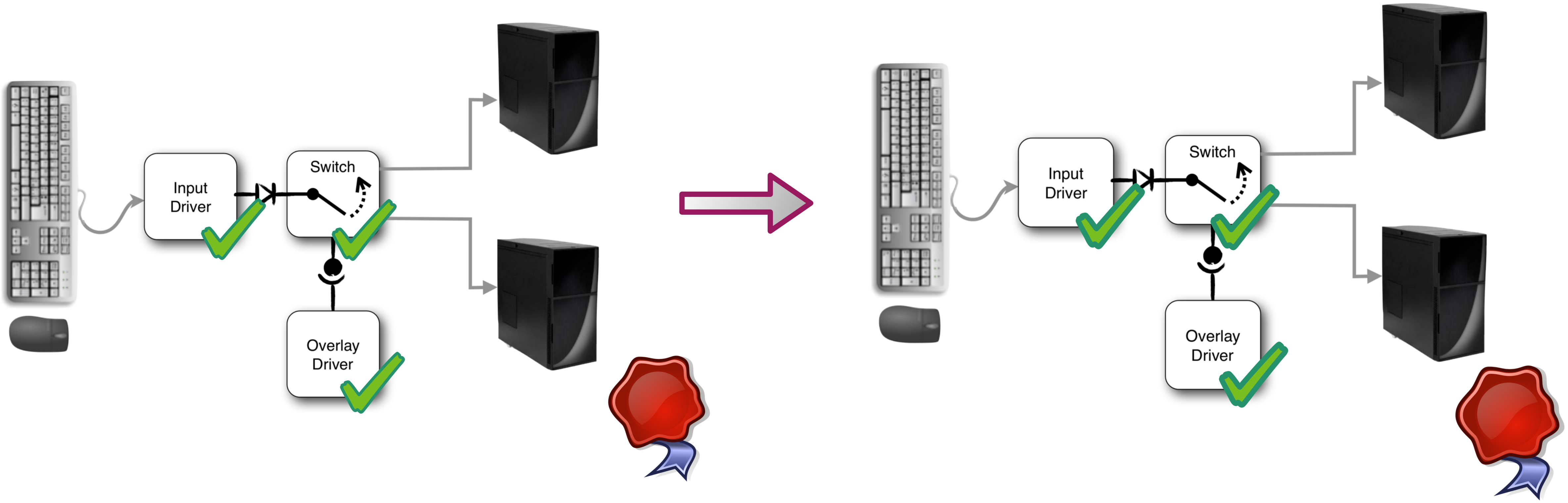
Case study

Cross Domain Desktop Compositor HID switch

Compiler application

While language with locks

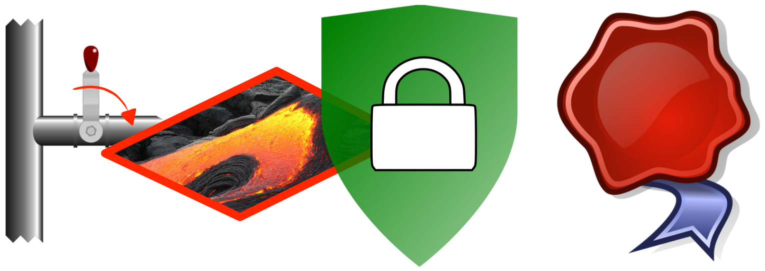
Generic RISC assembly with locks



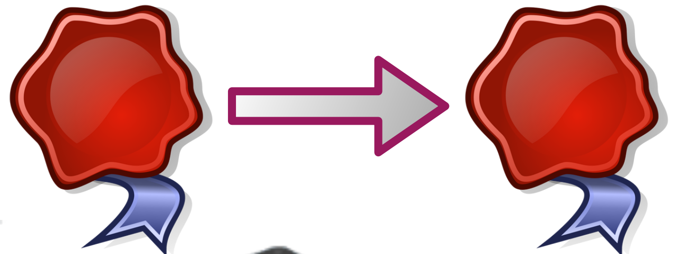
Thesis (PhD, 2020)

That “Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs” is feasible.

- **Program verification** Chapter 4



- **Compiler verification** Chapter 5



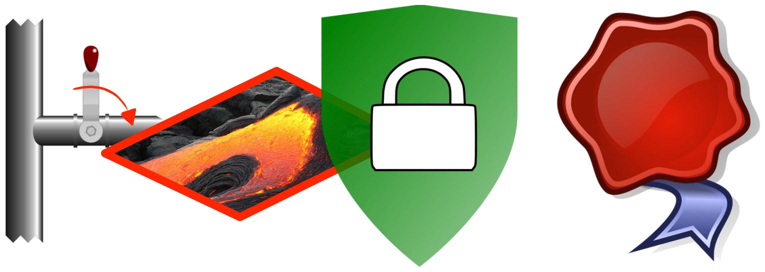
- **Case study: CDDC** Chapter 6

- Extension to program verification: Chapter 7

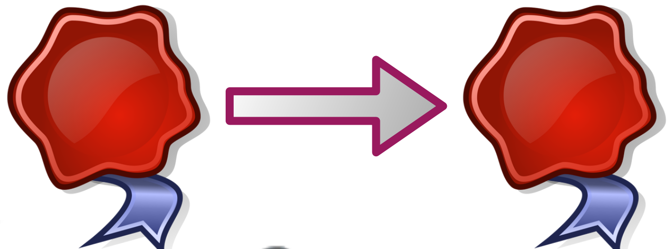
Thesis (PhD, 2020)

That “Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs” is feasible.

- **Program verification** Chapter 4



- **Compiler verification** Chapter 5

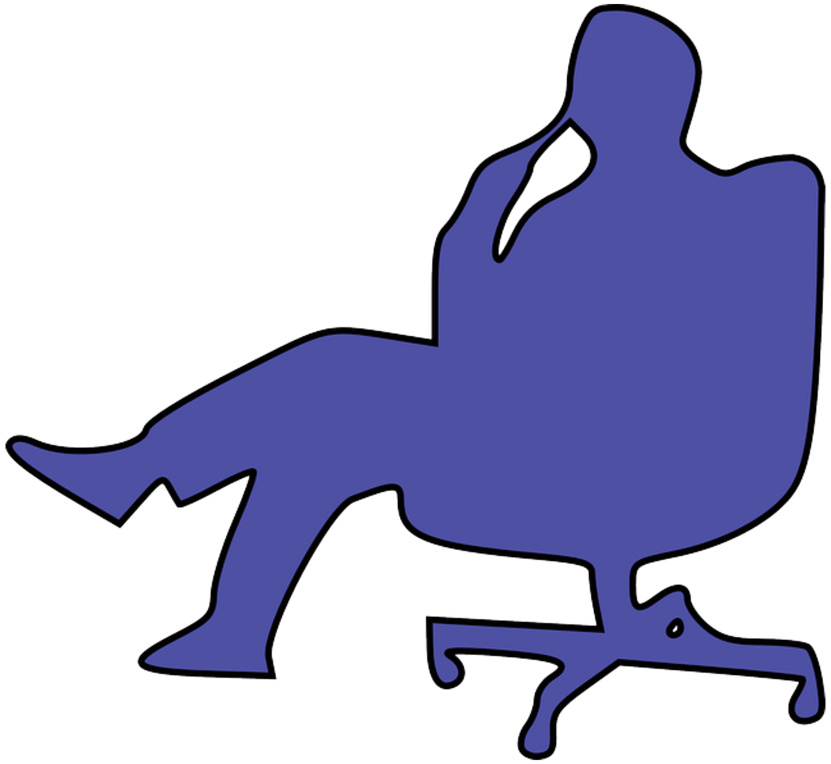


- **Case study: CDDC** Chapter 6

- Extension to program verification: Allow conditional branching on secrets Chapter 7

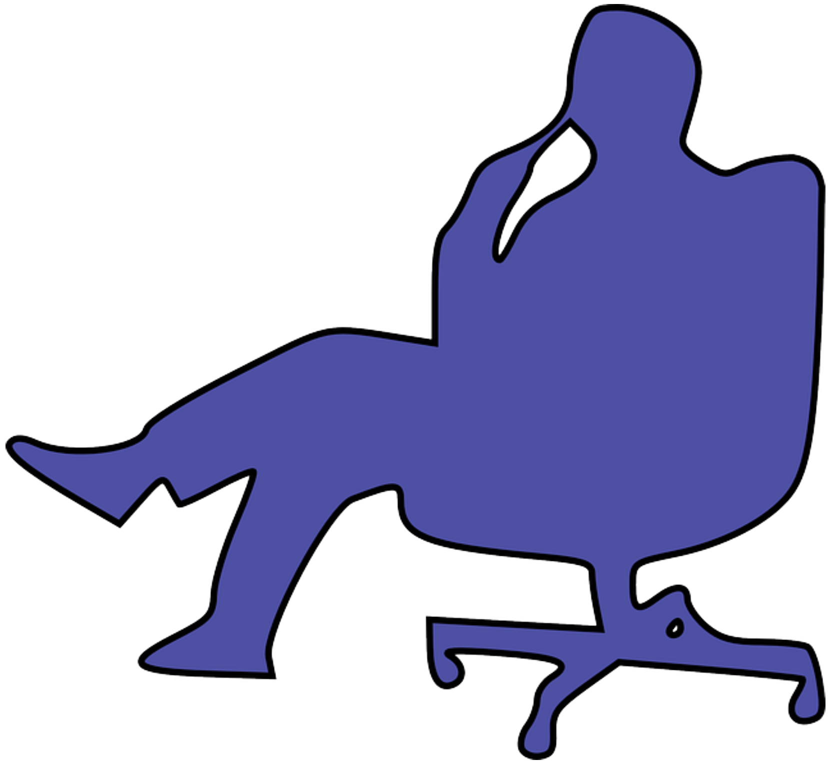
Explicit flow

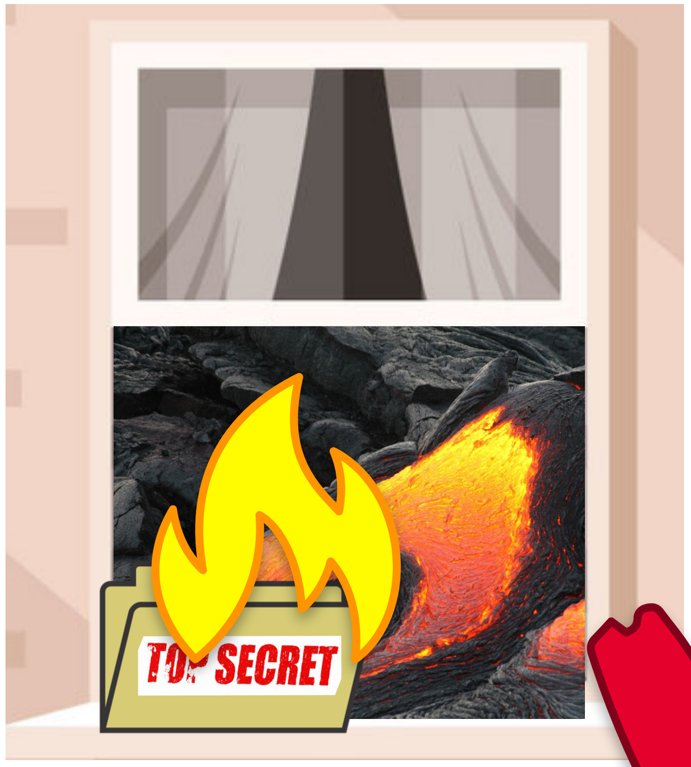
Explicit flow



output := secret

Explicit flow





output := secret

Analysis (rightly) rejects this!

Dangers of conditional branching on secrets

Implicit flow #1: "storage" leak

if (secret)

then

...do stuff, then...

output := 0

Dangers of conditional branching on secrets

Implicit flow #1: “storage” leak

if (secret)

then

…do stuff, then…

output := 0

else

…do other stuff, then…

output := 1

Dangers of conditional branching on secrets

Implicit flow #1: “storage” leak

if (secret)

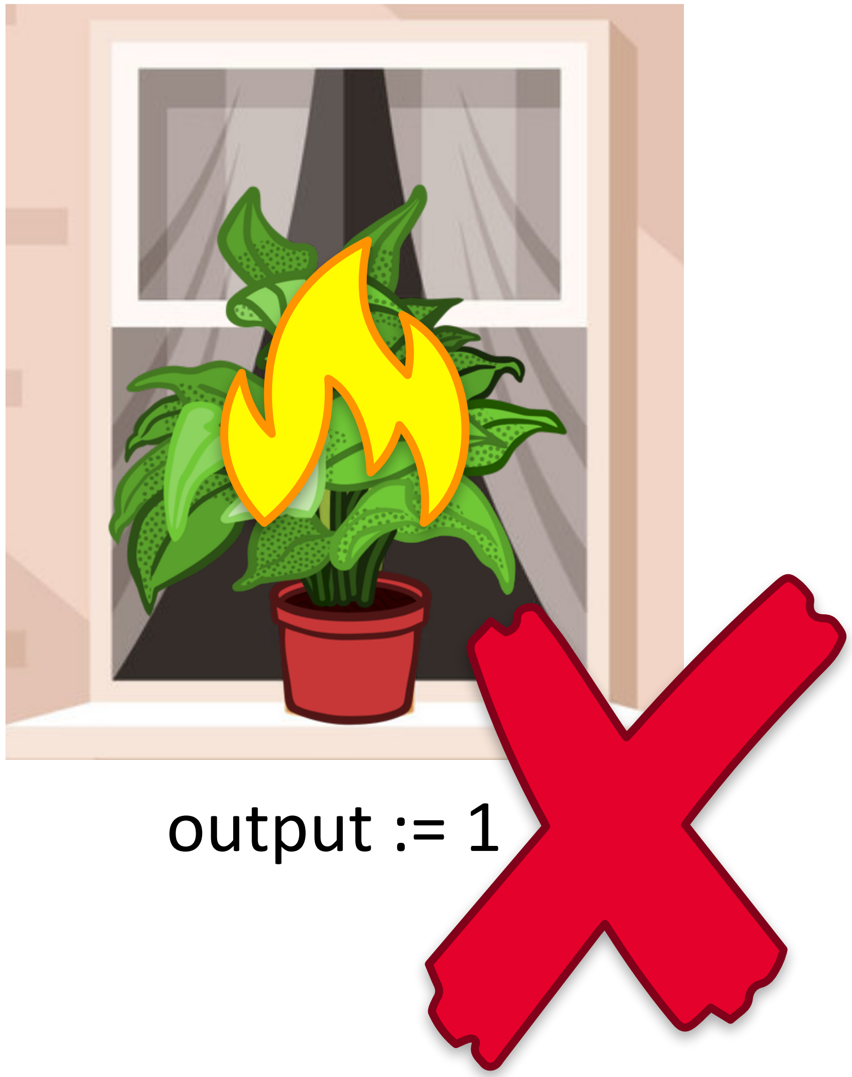
then

…do stuff, then…

output := 0

else

…do other stuff, then…



output := 1

Also (rightly) rejected

Dangers of conditional branching on secrets

Implicit flow #1: “storage” leak

if (secret)

then

…do stuff, then…

output := 0

else

…do other stuff, then…

output := 0

?

Dangers of conditional branching on secrets

Implicit flow #1: “storage” leak

if (secret)

then

…do stuff, then…

output := 0

else

…do other stuff, then…

output := 0

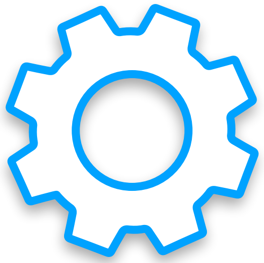
?

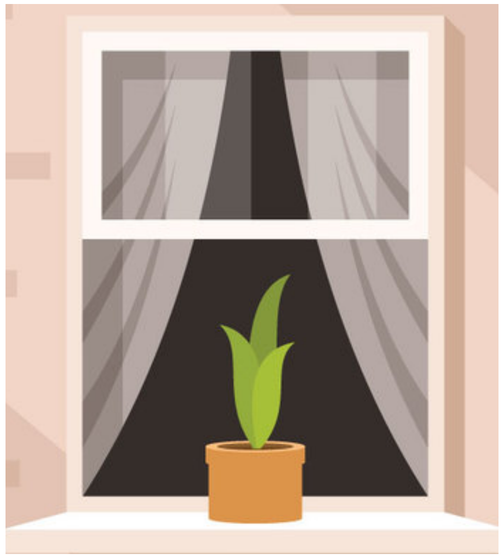
Is this safe?

Conditional branching on secrets

Implicit flow #2: timing leak

if (secret) then *…do stuff, then…*





(3pm)

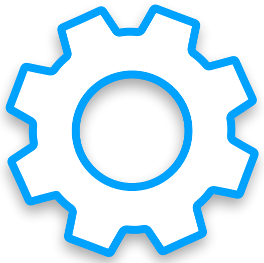
Conditional branching on secrets

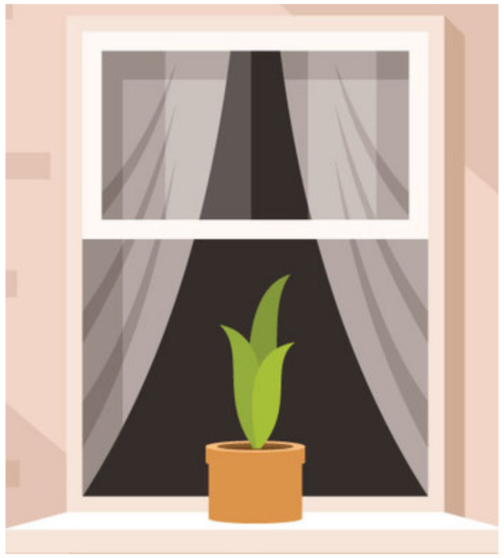
Implicit flow #2: timing leak

if (secret)

then

...do stuff, then...

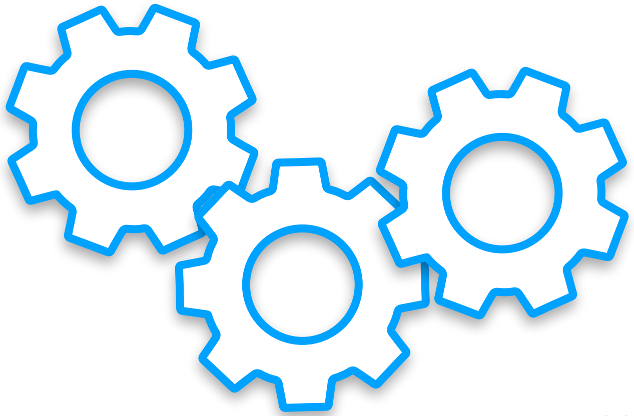




(3pm)

else

...do other stuff, then...



...

(9pm)

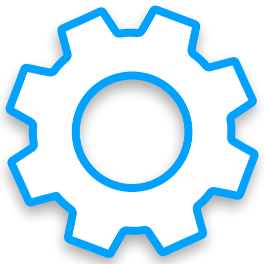
Conditional branching on secrets

Implicit flow #2: timing leak

if (secret)

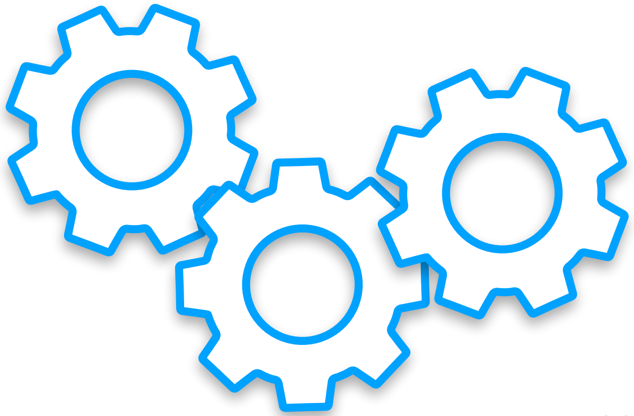
then

...do stuff, then...

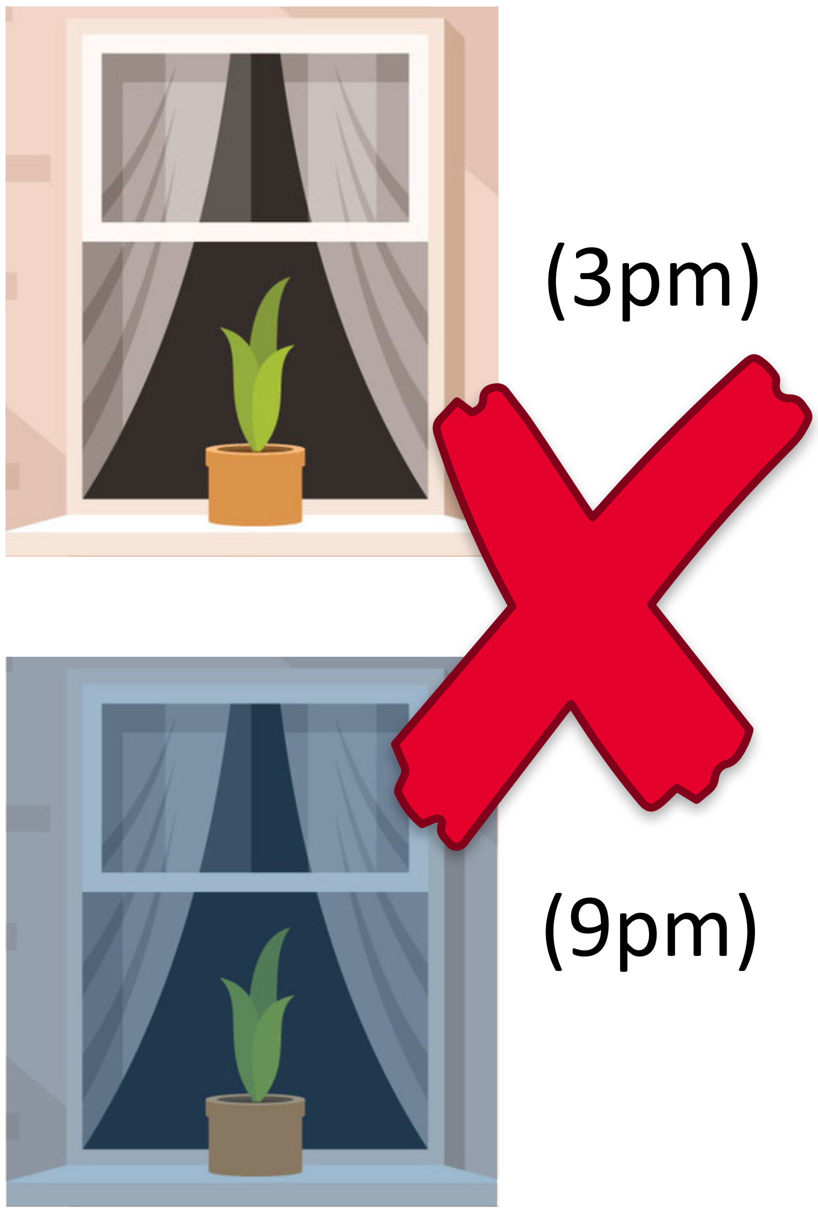


else

...do other stuff, then...



...

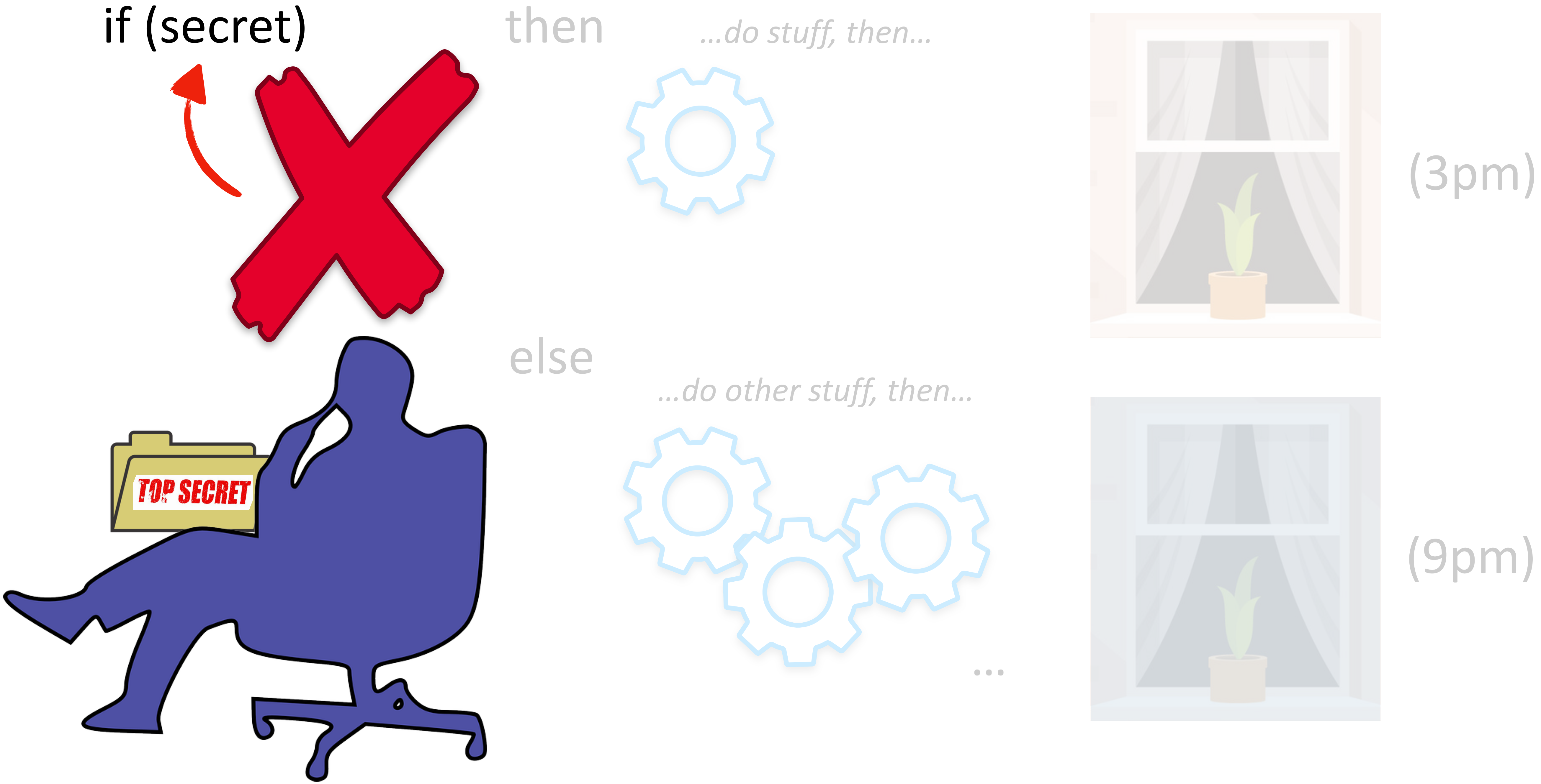


(3pm)

(9pm)

Conditional branching on secrets

Disallowed in previous Chps 4-6 of thesis!



if (secret)

then

...do stuff, then...

(3pm)

else

...do other stuff, then...

...

(9pm)

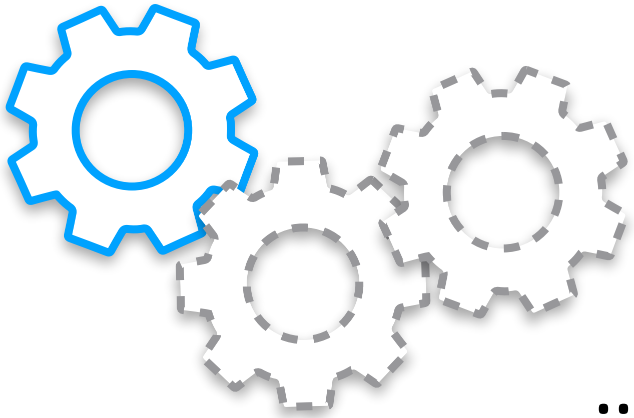
Conditional branching on secrets

Allowed by Chp 7 extension to type system

if (secret)

then

...do stuff, wait..., then...

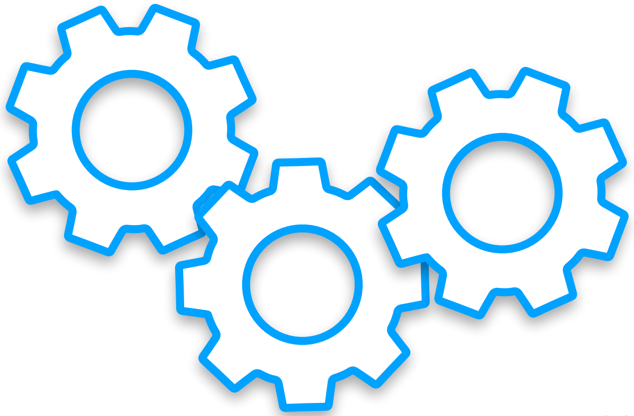


...

(9pm)

else

...do other stuff, then...



...

(9pm)

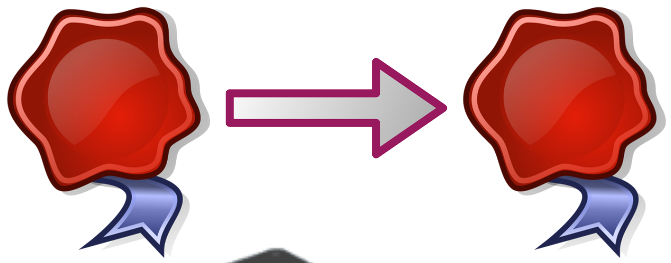
Thesis (PhD, 2020)

https://doi.org/fjmt

That “Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs” is feasible.

- **Program verification** Chapter 4

- **Compiler verification** Chapter 5



- **Case study: CDDC** Chapter 6

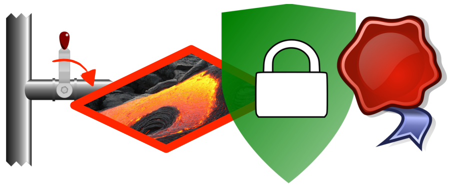
- Extension to program verification: Chapter 7
  Allow conditional branching on secrets

Thesis (PhD, 2020) and publications

https://doi.org/fjmt www.robs-cse.com

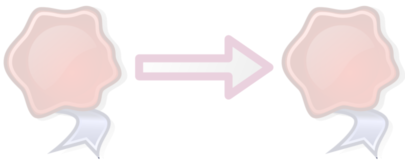
That "Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs" is feasible.

- **Program verification**



- **Case study: CDDC**

- **Compiler verification**

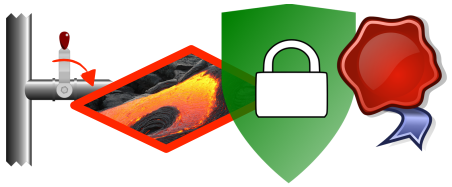


Thesis (PhD, 2020) and publications

https://doi.org/fjmt www.robs-cse.com

That "Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs" is feasible.

- **Program verification**

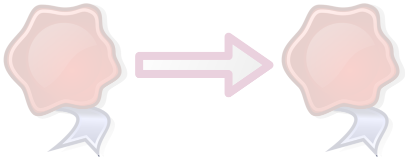


Joint work with
Toby Murray
(Uni Melbourne)

- **Case study: CDDC**

  - Murray, Sison & Engelhardt (EuroS&P 2018)

- Compiler verification



Thesis (PhD, 2020) and publications

https://doi.org/fjmt www.robs-cse.com

That “Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs” is feasible.

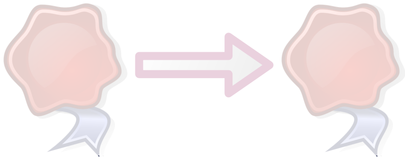
- **Program verification**

Joint work with
Toby Murray
(Uni Melbourne)

- **Case study: CDDC**

  - Murray, Sison & Engelhardt (EuroS&P 2018)
  - Part of Eureka Prize 2021 finalist entry
    (w/ Beaumont et al. ACSAC 2016)

- **Compiler verification**

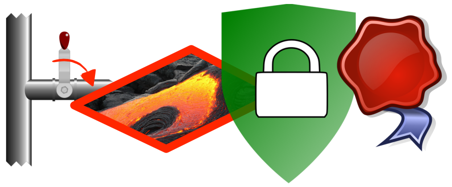


Thesis (PhD, 2020) and publications

https://doi.org/fjmt www.robs-cse.com

That “Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs” is feasible.

- **Program verification**

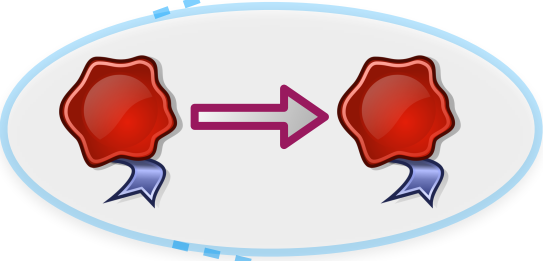


Joint work with
Toby Murray
(Uni Melbourne)

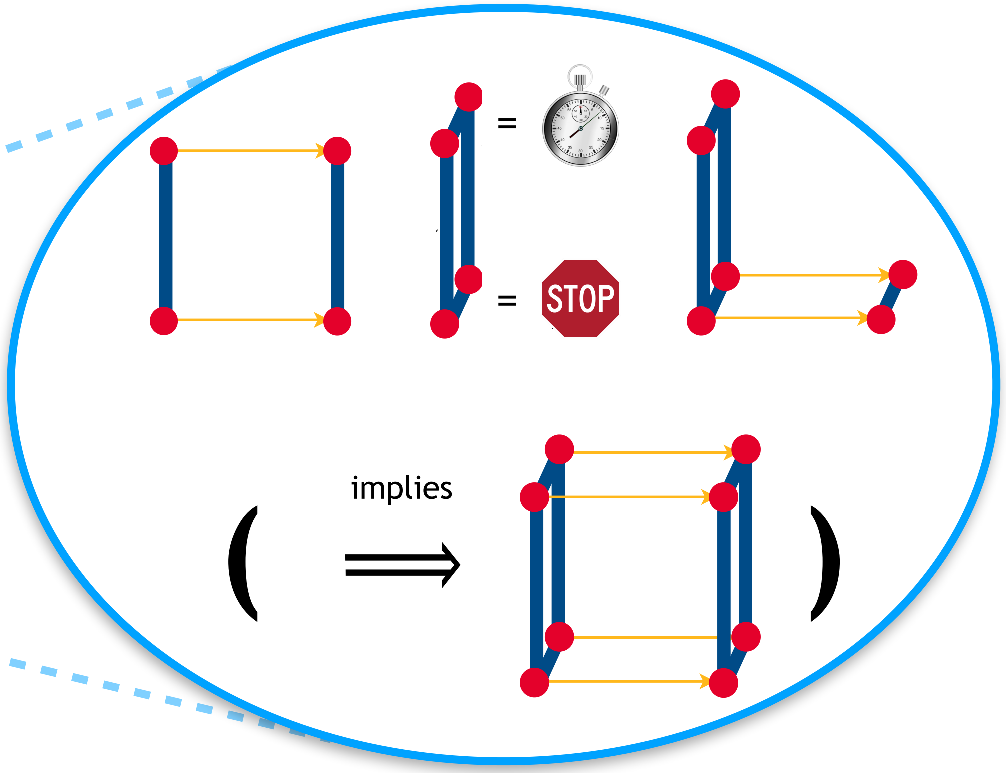
- **Case study: CDDC**

  - Murray, Sison & Engelhardt (EuroS&P 2018)
  - Part of Eureka Prize 2021 finalist entry (w/ Beaumont et al. ACSAC 2016)

- **Compiler verification**



decomposition principle



Thesis (PhD, 2020) and publications

https://doi.org/fjmt www.robs-cse.com

That "Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs" is feasible.

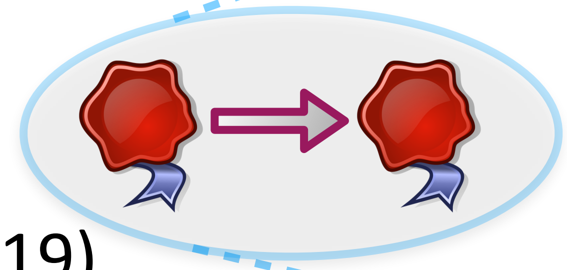
Joint work with
Toby Murray
(Uni Melbourne)

- **Program verification**

- **Case study: CDDC**

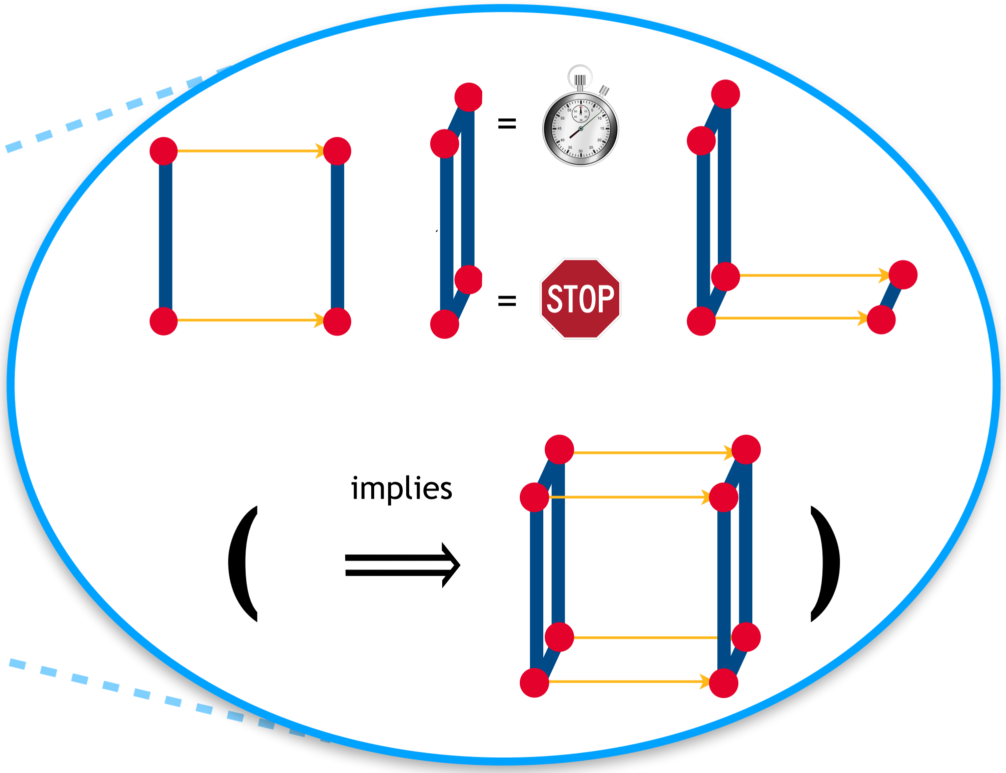
  - Murray, Sison & Engelhardt (EuroS&P 2018)
  - Part of Eureka Prize 2021 finalist entry (w/ Beaumont et al. ACSAC 2016)

- **Compiler verification**



  - Sison & Murray (ITP 2019)

decomposition principle



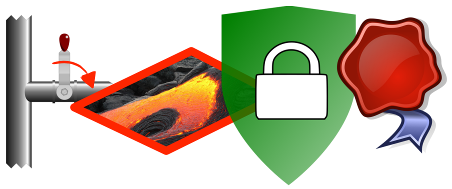
Thesis (PhD, 2020) and publications

https://doi.org/fjmt www.robs-cse.com

That "Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs" is feasible.

Joint work with
Toby Murray
(Uni Melbourne)

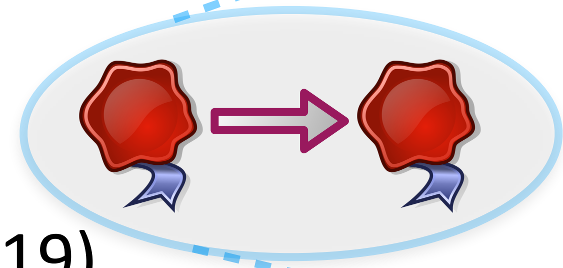
- **Program verification**



- **Case study: CDDC**

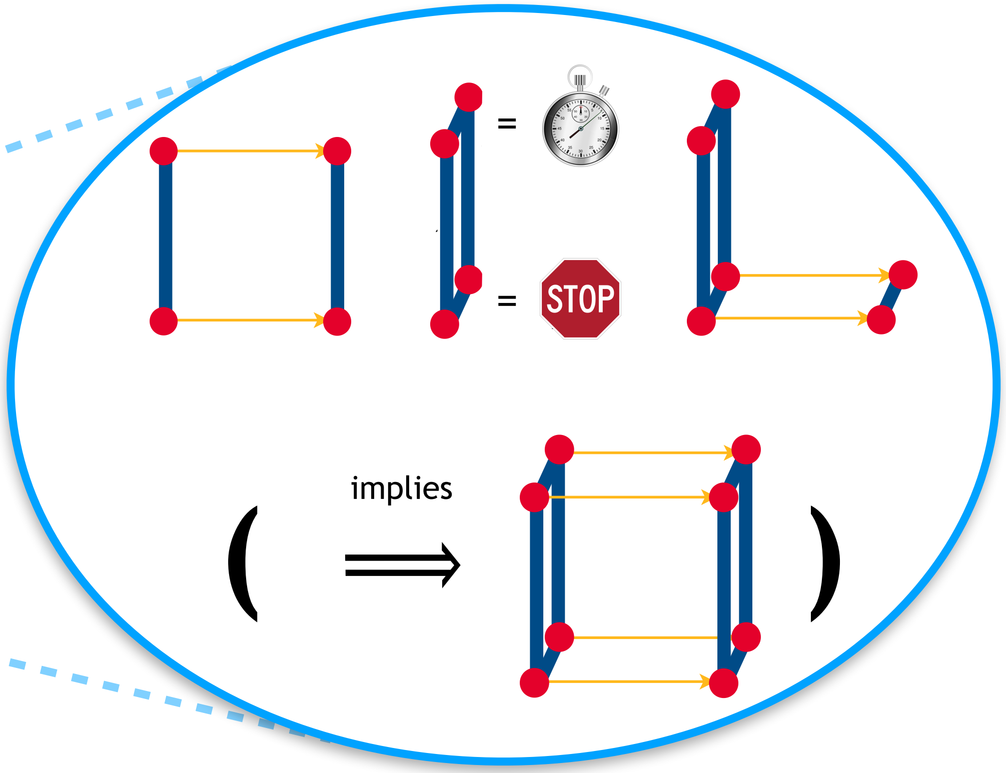
  - Murray, Sison & Engelhardt (EuroS&P 2018)
  - Part of Eureka Prize 2021 finalist entry (w/ Beaumont et al. ACSAC 2016)

- **Compiler verification**



  - Sison & Murray (ITP 2019)
  - J. Funct. Programming vol. 31, 2021

decomposition principle



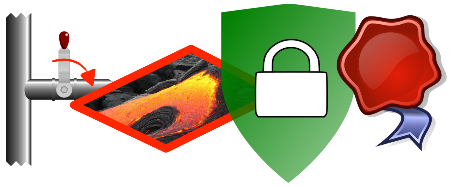
Thesis (PhD, 2020) and publications

https://doi.org/fjmt www.robs-cse.com

Thank you!
Q & A

That "Proving Confidentiality and Its Preservation Under Compilation for Mixed-Sensitivity Concurrent Programs" is feasible.

- **Program verification**

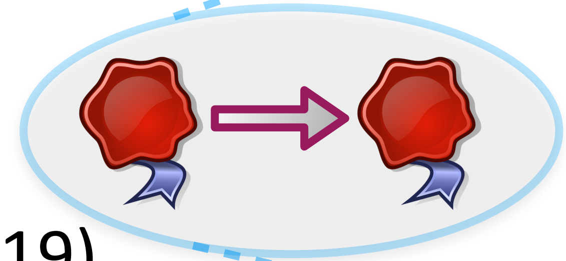


Joint work with
Toby Murray
(Uni Melbourne)

- **Case study: CDDC**

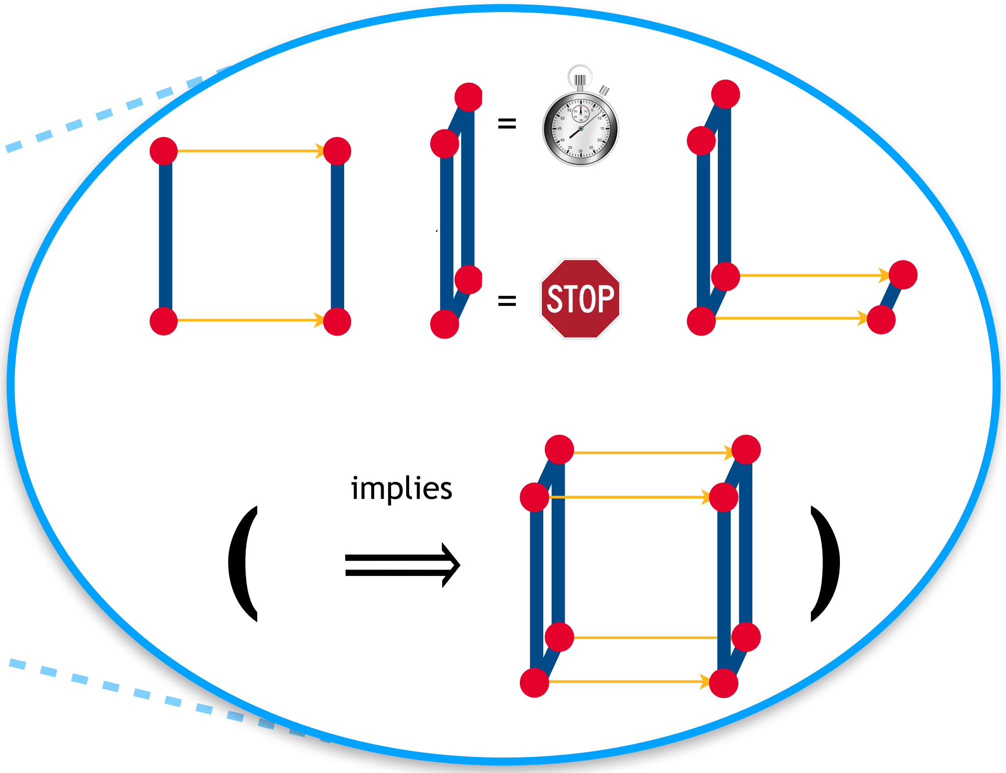
  - Murray, Sison & Engelhardt (EuroS&P 2018)
  - Part of Eureka Prize 2021 finalist entry (w/ Beaumont et al. ACSAC 2016)

- **Compiler verification**



  - Sison & Murray (ITP 2019)
  - J. Funct. Programming vol. 31, 2021

decomposition principle


=
=
STOP
implies