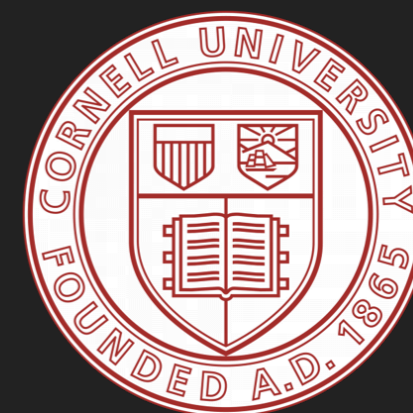


DREW ZAGIEBOYLO
G. EDWARD SUH
ANDREW C. MYERS



USING INFORMATION FLOW TO DESIGN AN ISA THAT CONTROLS TIMING CHANNELS

**WE NEED A NEW
HW-SW CONTRACT (ISA) TO
CONTROL MICROARCHITECTURAL
TIMING CHANNELS**

MICROARCHITECTURE

- ▶ *Unspecified* CPU behavior + state
- ▶ Primarily affect performance

- ▶ Caches

Valid	Address	Value
1	0xffff	0x1234
0	0xabcd	0xaaaa
1	0x10af	0xde12

- ▶ Branch Predictors

PC	Branch?
0x1234	Y
0xa3a4	N
0xdddd	Y

- ▶ Arithmetic Units, Prefetchers, Address Translators, Fill buffers, Memory Arbiters, Pipeline State, etc.

MICROARCHITECTURAL TIMING CHANNEL

- ▶ Essence of the *Meltdown* ('18) attack

CPU Cache

```
s1 = p0[s0]  
p1 = p0[0]
```


Valid	Address
0	0xffff
0	0xabcd
0	0x10af

MICROARCHITECTURAL TIMING CHANNEL

- ▶ Essence of the *Meltdown* ('18) attack

CPU Cache

```
s1 = p0[s0]  
p1 = p0[0]
```



Valid	Address
0	0xffffffff
0	0xabcd
0	0x10af

MICROARCHITECTURAL TIMING CHANNEL

- ▶ Essence of the *Meltdown* ('18) attack

CPU Cache

```
s1 = p0[s0] ←  
p1 = p0[0]
```

Valid	Address
0	0xffffffff
1	p0 + s0
0	0x10af

MICROARCHITECTURAL TIMING CHANNEL

- ▶ Essence of the *Meltdown* ('18) attack

CPU Cache

```
s1 = p0[s0]  
p1 = p0[0]
```



Valid	Address
0	0xffffffff
1	p0 + s0
0	0x10af

s0 == 0

Cache HIT!



MICROARCHITECTURAL SIDE CHANNELS



Spectre ('18)



Foreshadow ('18)



Meltdown ('18)



RIDL ('19)



Zombieload ('19)

MITIGATING MICROARCHITECTURAL SIDE CHANNELS

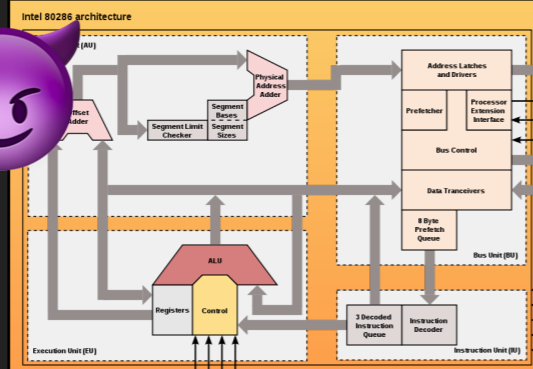
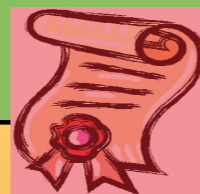
APPLICATION CODE



PROGRAMMING LANGUAGES



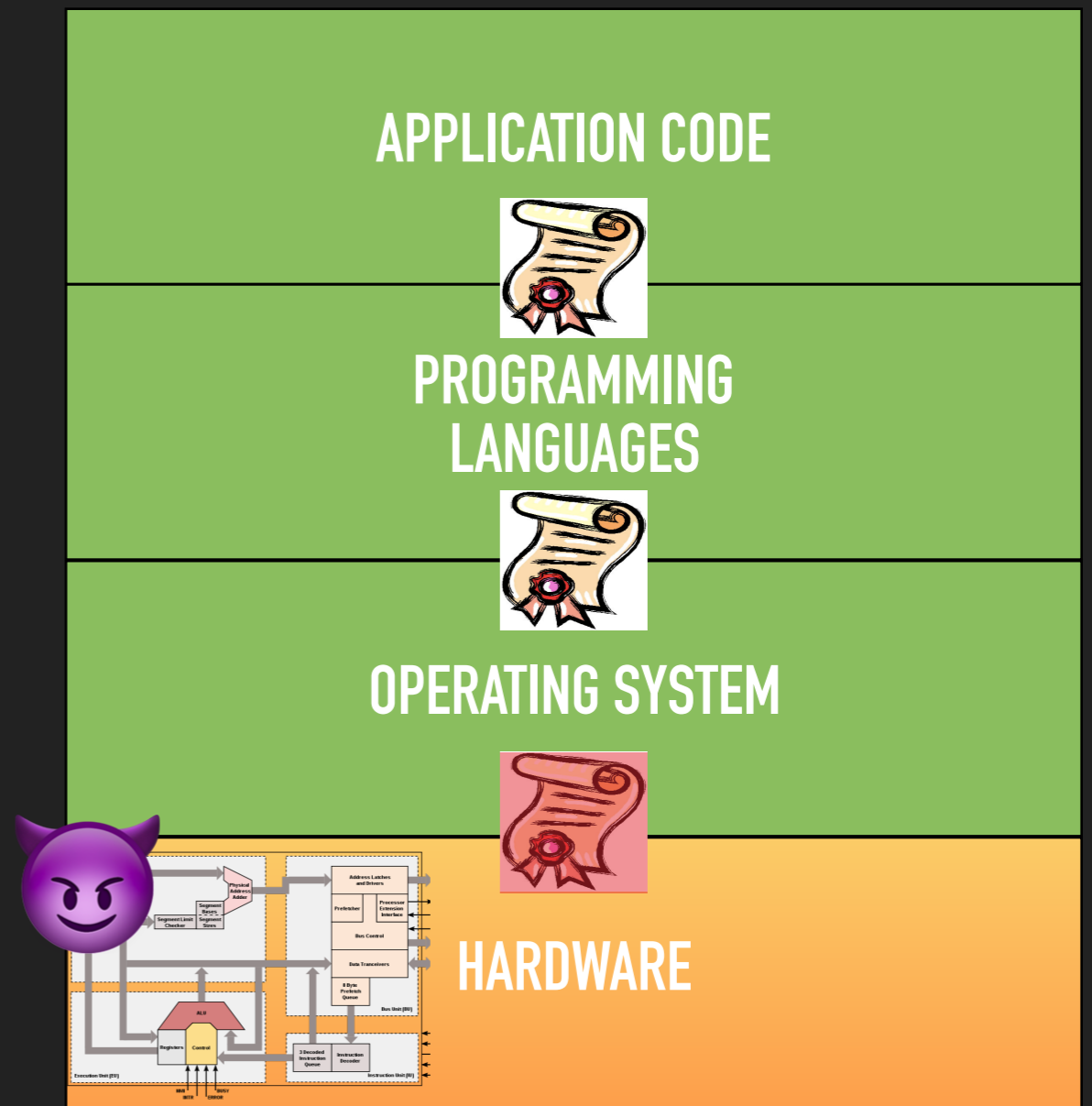
OPERATING SYSTEM



HARDWARE

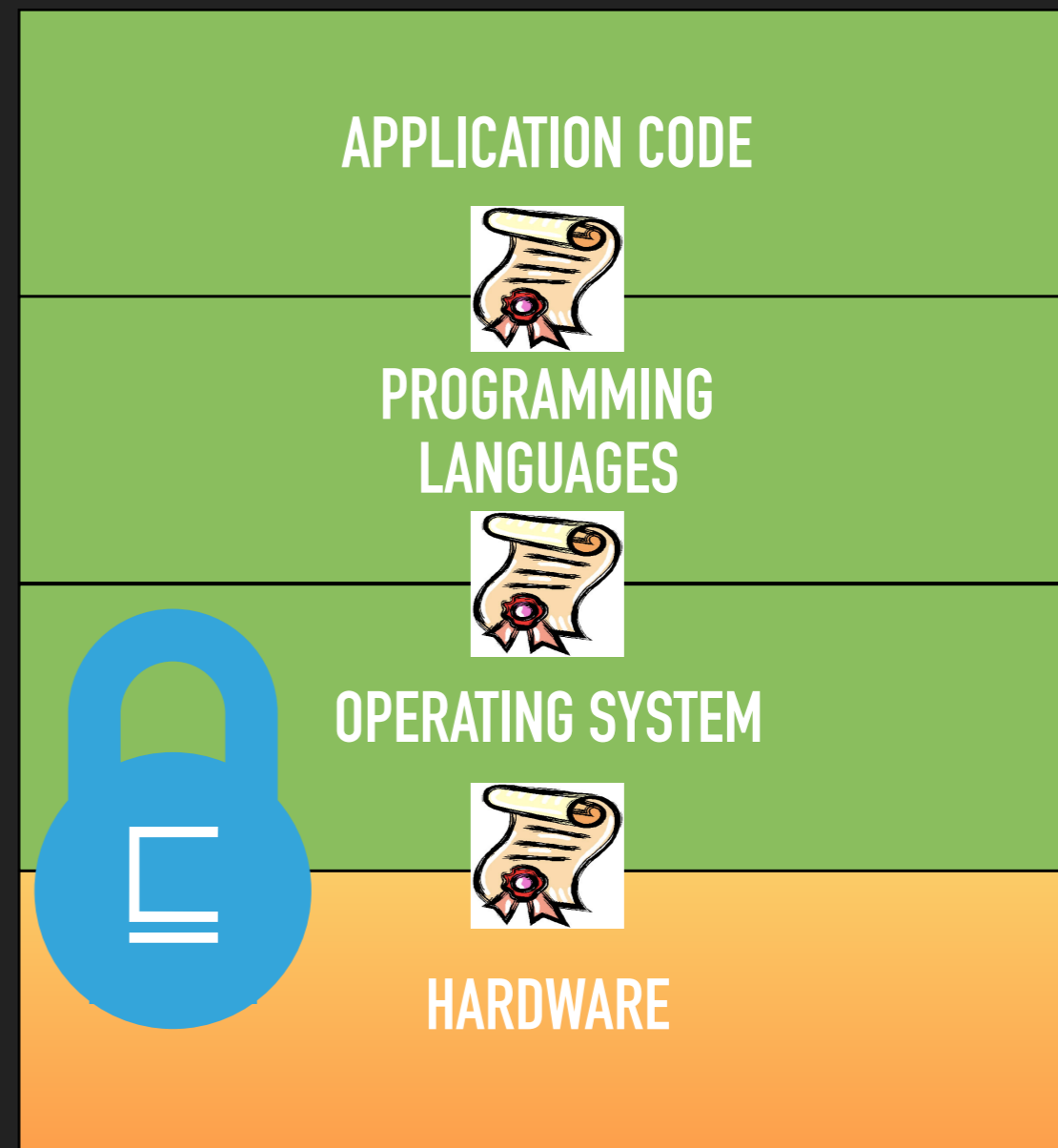
MITIGATING MICROARCHITECTURAL SIDE CHANNELS

- ▶ Microarchitectural side channels *bypass* current contracts
- ▶ Need a HW-SW co-design mitigation mechanism



THIS PAPER: INFORMATION FLOW CONTROL ISA

1. Sound and portable contract for secure SW design
 - Extension of RISC-V
 - Timing-Sensitive Noninterference
2. Guide to HW designers
 - Microarchitectural Noninterference
 - No implementation details
3. Practical ISA Features
 - Constrained Downgrading
 - Control Transfer Primitives



DYNAMIC IFC CONTRACT

- ▶ Software controls IFC labels
 - ▶ Labels are *mutable state*
 - ▶ SW must explicitly change architectural labels
- ▶ Hardware enforces policies at runtime

FLOATING LABELS



ENFORCING NONINTERFERENCE

- ▶ Change existing semantics of RISC-V

$$L(pc) \sqcup L(x2) \sqcup L(x3) \overset{\checkmark}{\sqsubseteq} L(x1)$$

X1 := X2 + X3

add x1, x2, x3

$$L(pc) \sqcup L(x2) \sqcup L(x3) \not\sqsubseteq L(x1)$$



NO-OP

ENFORCING NONINTERFERENCE

- ▶ Change existing semantics of RISC-V

$$L(x1) \sqcup L(x2) \sqsubseteq L(pc) \quad \checkmark$$

```
if (x1 == s2)
then PC := LOC
```

```
beq x1, x2, LOC
```

$$L(x1) \sqcup L(x2) \not\sqsubseteq L(pc) \quad \times$$

```
NO-OP
```

SIMILAR RESTRICTIONS ON OTHER INSTRUCTIONS

TIMING SENSITIVITY

- ▶ How does software control timing?
- ▶ *State that influences timing* $\sqsubseteq t$
- ▶ Language-based Timing Mitigation [Zhang et al. '12]



TIMING SENSITIVITY



(SECRETS CAN INFLUENCE TIMING)



CPU Cache

```
s1 = p0[s0]
p1 = p0[0]
```

LBL	ADDR
X	??
X	??
X	??

TIMING SENSITIVITY



(SECRETS CAN INFLUENCE TIMING)



CPU Cache

```
s1 = p0[s0]
p1 = p0[0]
```



LBL	ADDR
X	??
X	??
X	??

TIMING SENSITIVITY



(SECRETS CAN INFLUENCE TIMING)



CPU Cache

$s1 = p0[s0]$

$p1 = p0[0]$



LBL

ADDR

SEC

$p0 + s0$

X

??

X

??

TIMING SENSITIVITY



(SECRETS CAN INFLUENCE TIMING)



$s_0 == 0$
Cache HIT! 🤪

CPU Cache

```

s1 = p0[s0]
p1 = p0[0]
    
```

LBL	ADDR
SEC	$p_0 + s_0$
X	??
X	??

TIMING SENSITIVITY



CPU Cache

```
s1 = p0[s0]
p1 = p0[0]
```

A red arrow points from the right side of the code block towards the second line of code.

LBL	ADDR
X	??
X	??
X	??

TIMING SENSITIVITY



CPU Cache

```

s1 = p0[s0]
p1 = p0[0]

```



LBL	ADDR
SEC	$p0 + s0$
X	??
X	??

TIMING SENSITIVITY



```
s1 = p0[s0]  
p1 = p0[0]
```



CPU Cache

LBL

ADDR

Can't use Secret Entries!
Cache Miss!

X

??

MICROARCHITECTURAL NONINTERFERENCE

- ▶ Software controls *data* and *timing labels*
- ▶ Hardware ensures *timing* and *microarchitecture* observe noninterference

$$\forall C_1, C_2. (C_1 =_L C_2) \wedge (C_i \rightarrow C'_i)$$

$$\implies (C'_1[\mu] =_L C'_2[\mu]) \wedge (C'_1[t] =_L C'_2[t])$$

**MICROARCHITECTURAL
STATE**

INSTRUCTION DURATION

MICROARCHITECTURAL NONINTERFERENCE

$$\forall C_1, C_2. (C_1 =_L C_2) \wedge (C_i \rightarrow C'_i) \\ \implies (C'_1[\mu] =_L C'_2[\mu]) \wedge (C'_1[t] =_L C'_2[t])$$

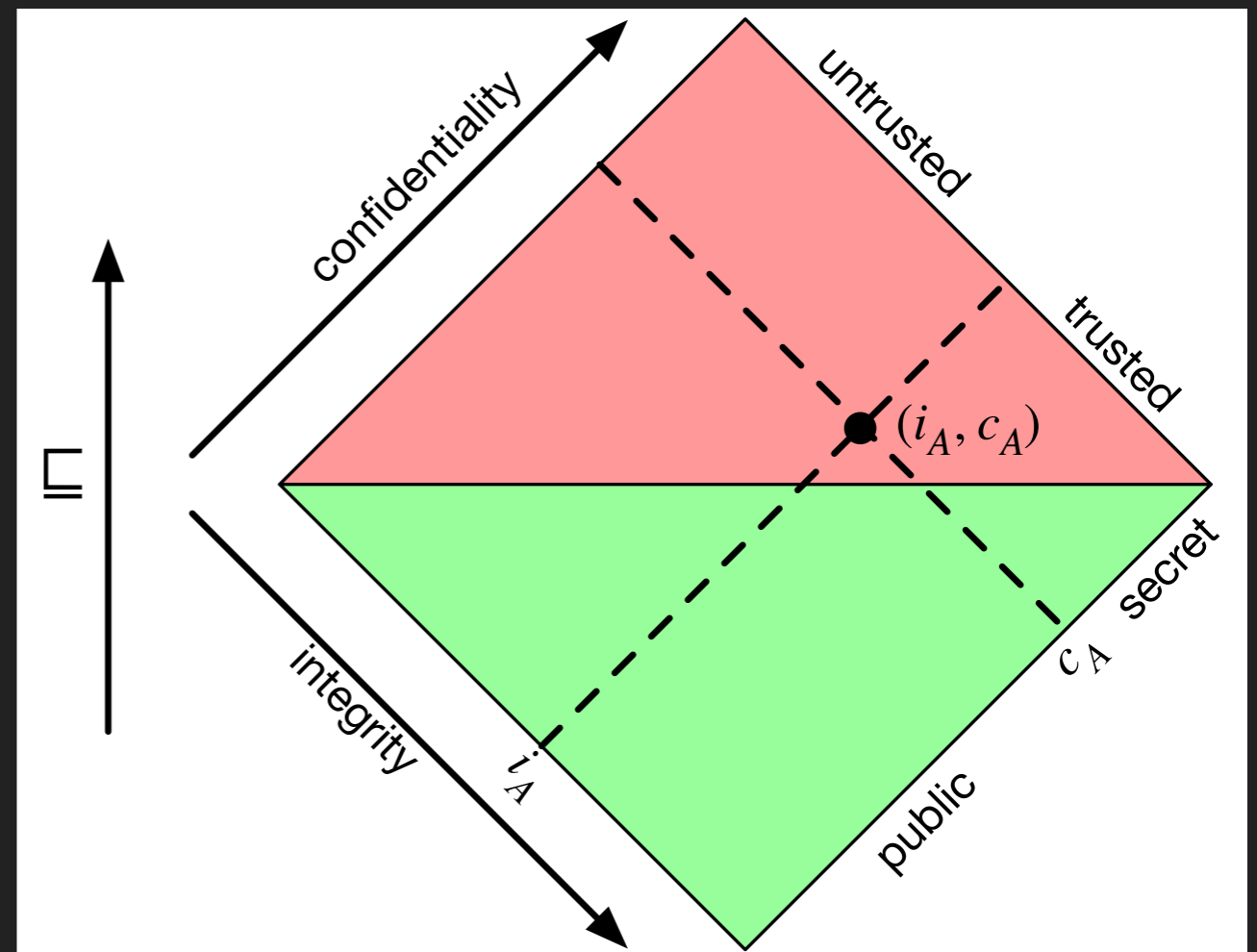
- ▶ Implementation independent
 - ▶ Allows many performance optimizations (e.g. speculation)
 - ▶ HW designer must prove implementation safe

IFC SECURITY GUARANTEES

Security Guarantee	Requires Static Enforcement	Requires Compiler Changes
TS-Noninterference	Y	Y
TS-Nonmalleability	N	Y
Legacy Isolation	N	N

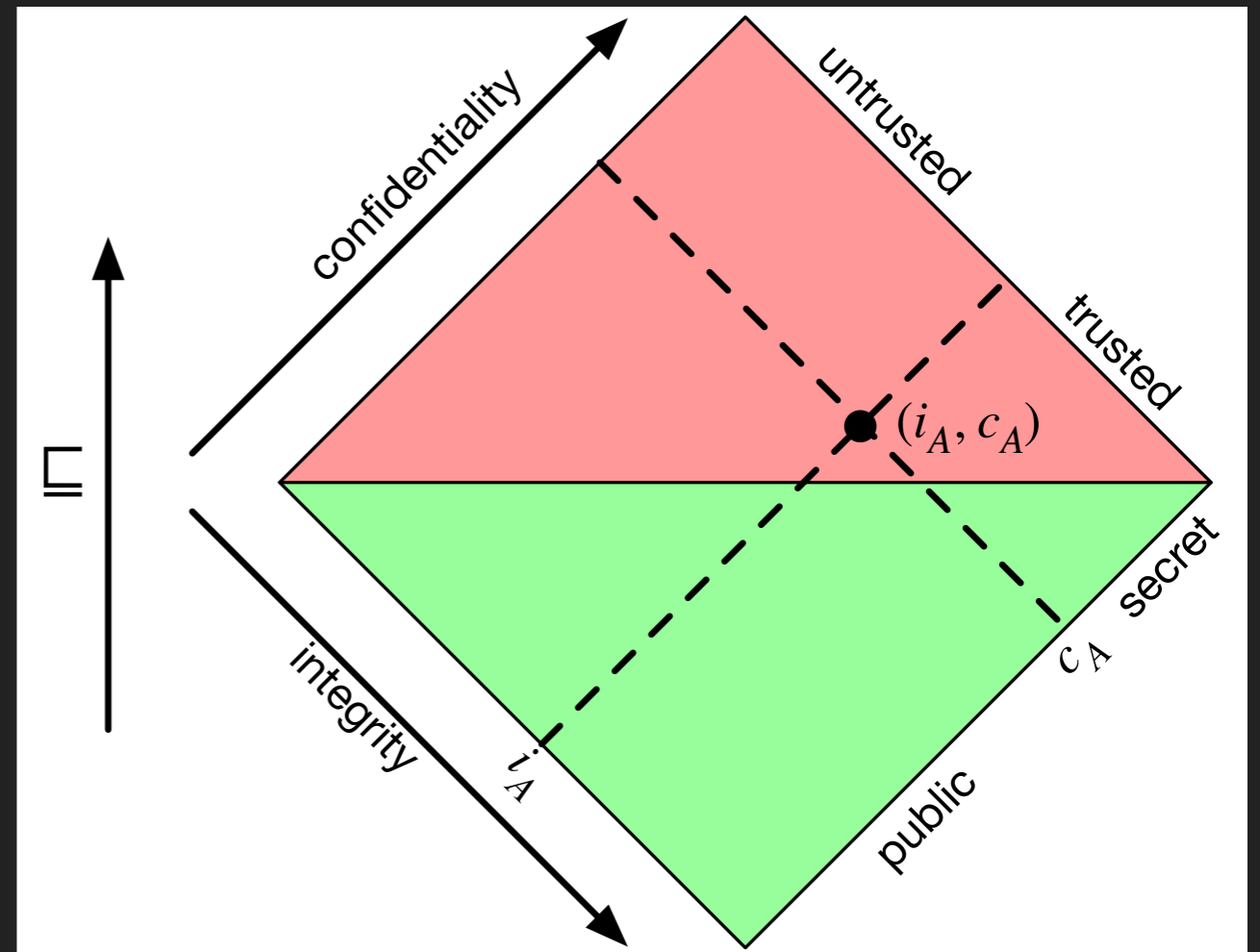
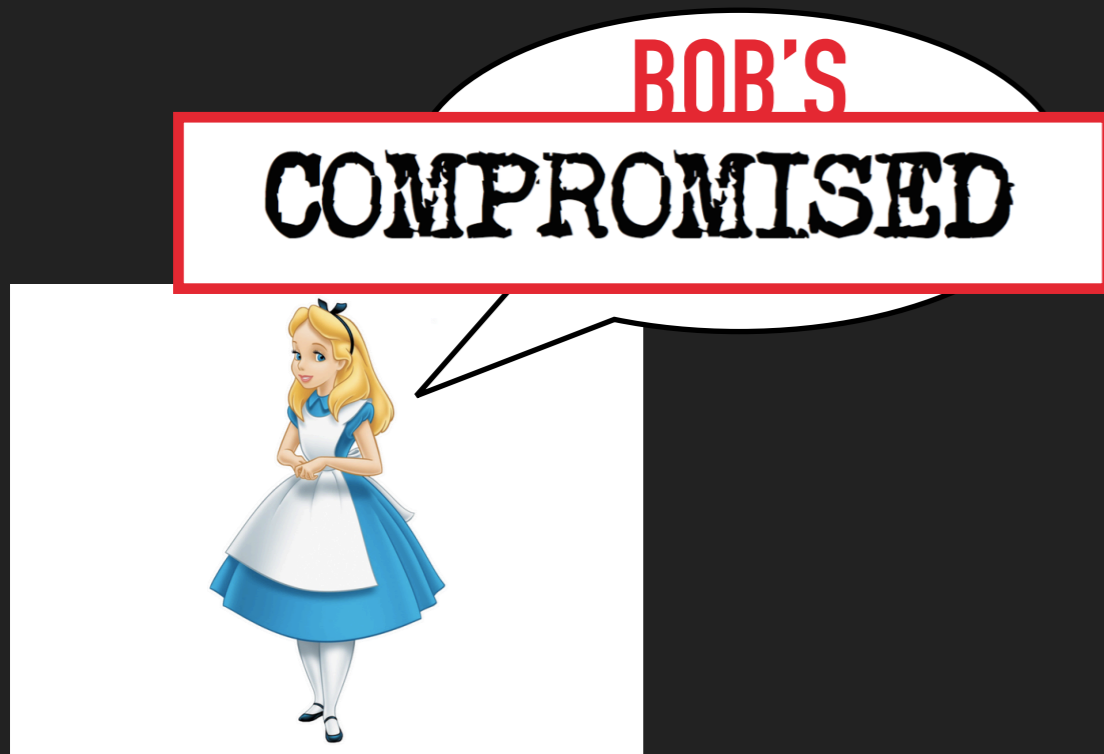
CONSTRAINING DOWNGRADING

- ▶ Downgrades violate noninterference
 - ▶ *Declassification, Endorsement*
- ▶ *Nonmalleable Information Flow* [Cecchetti et al. '17]
 - ▶ *Robust Declassification + Transparent Endorsement*



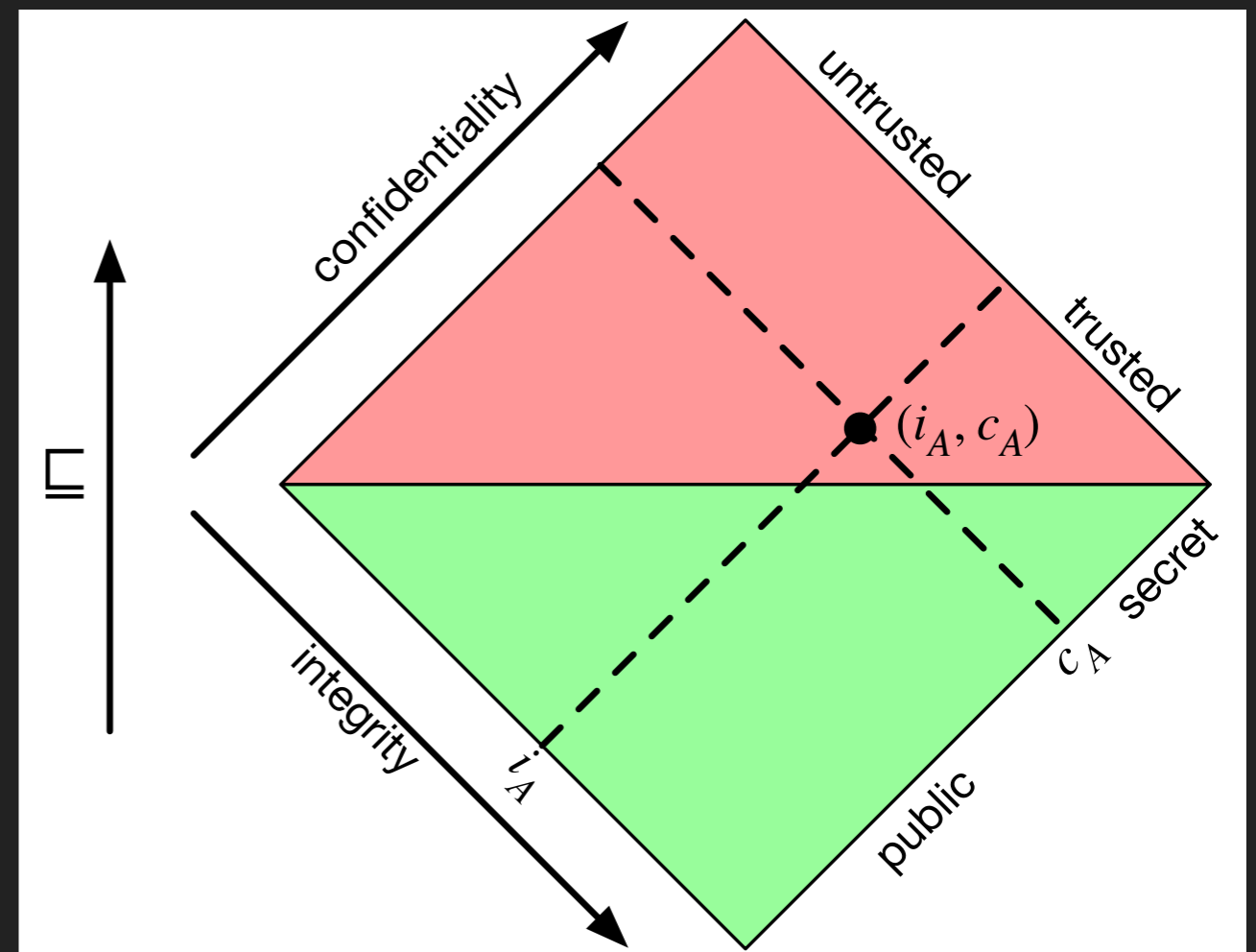
CONSTRAINING DOWNGRADING

- ▶ Key Idea: *Compromised* data may not be downgraded
- ▶ Compromised = More **secret** than **trustworthy**

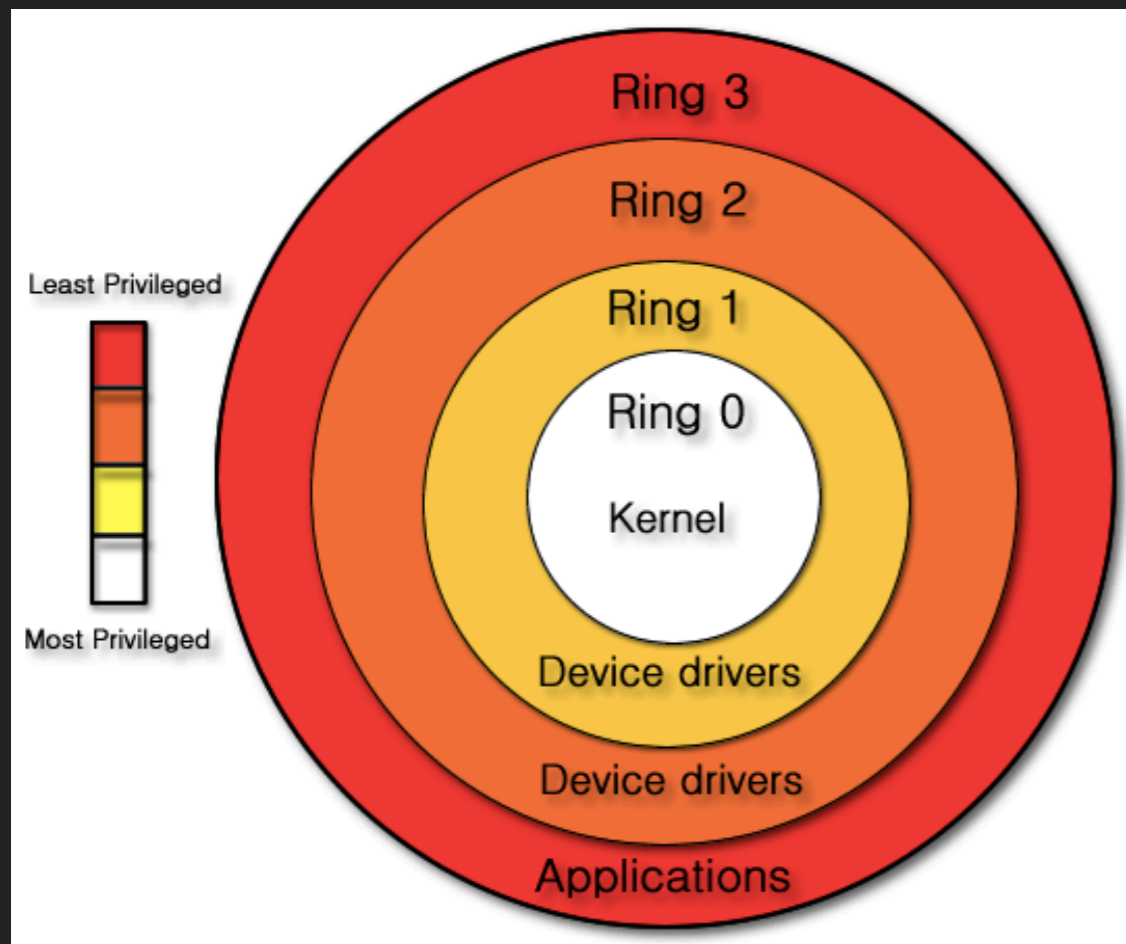


CONSTRAINING DOWNGRADING

- ▶ *Compromised* = More **secret** than **trustworthy**
- ▶ Novel contribution:
 - ▶ More general re-label restrictions in dynamic label setting
 - ▶ Must keep pc uncompromised



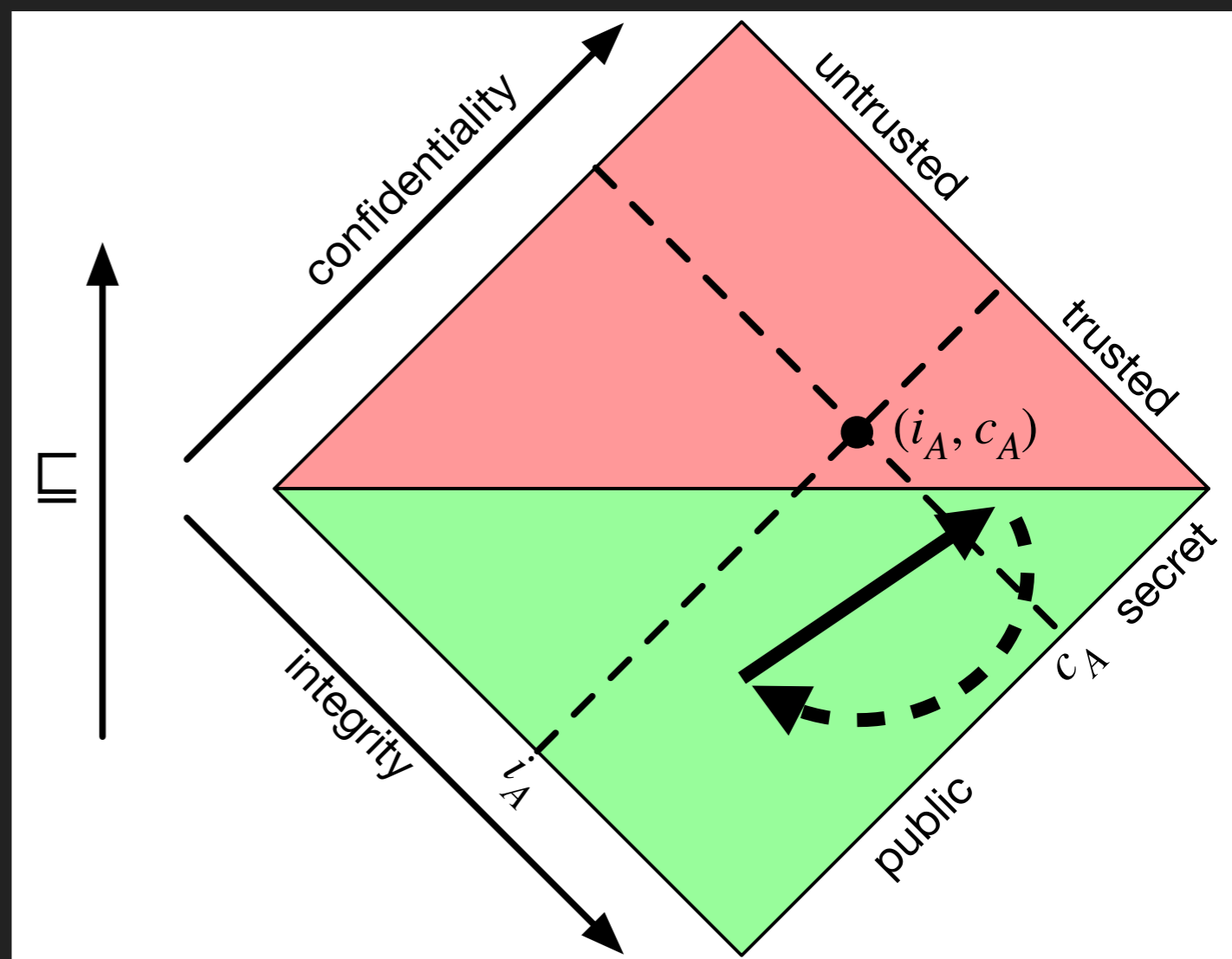
CONTROL TRANSFER



- ▶ Required for sharing HW across security domains
- ▶ System Calls and other HW primitives provide support
- ▶ *Protection Rings* are an instance of a Lattice
- ▶ **Call Gates**
 - ▶ Generalized control transfer for a lattice policy

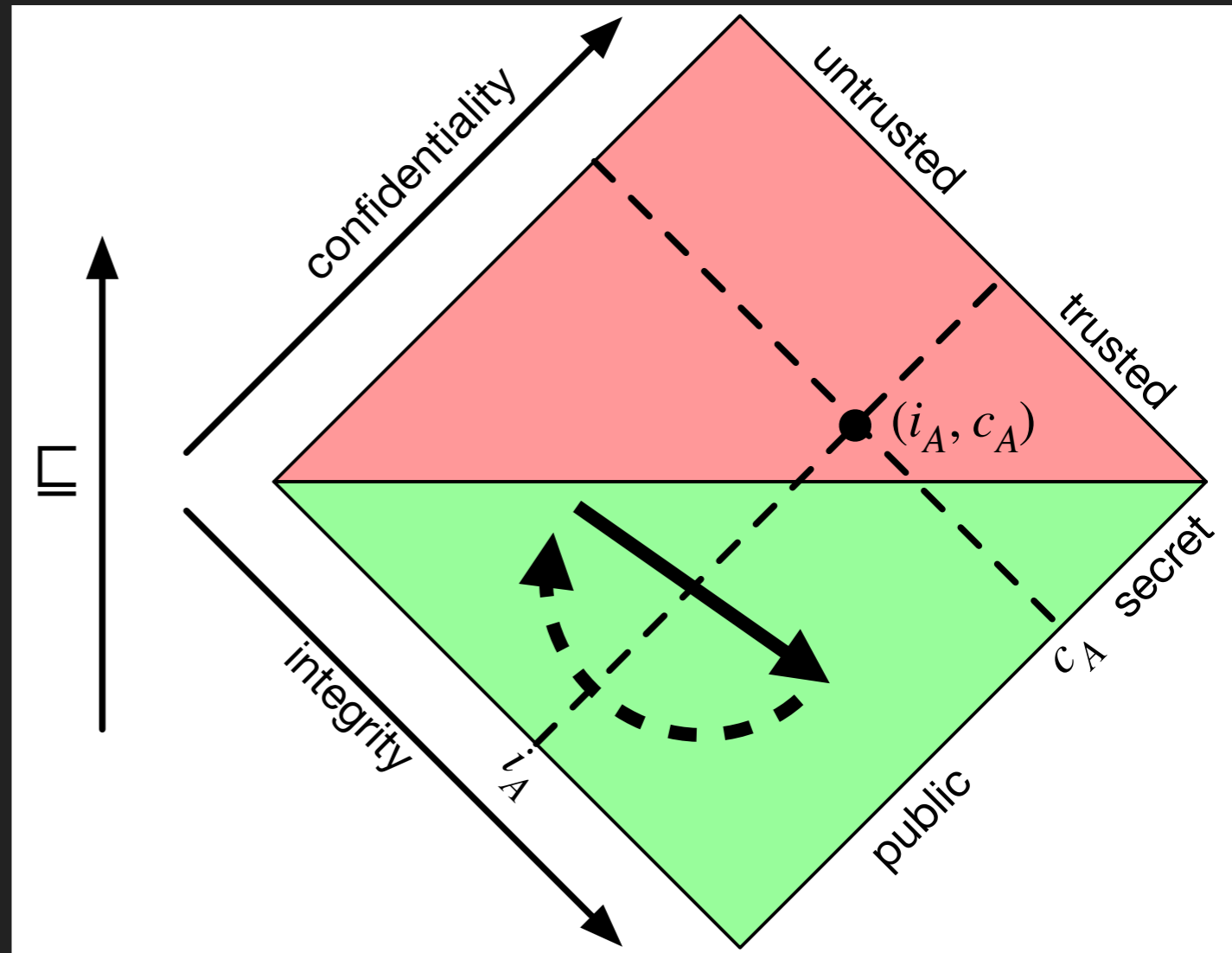
CALL GATES

- ▶ Upcalls
 - ▶ Establish return conditions a priori
 - ▶ Primitive for *black-box timing mitigation* [Zhang et al. '12]
 - ▶ Primitive for sandboxing



CALL GATES

- ▶ Downcalls
- ▶ Analogous to system calls
- ▶ Pre-register entry points



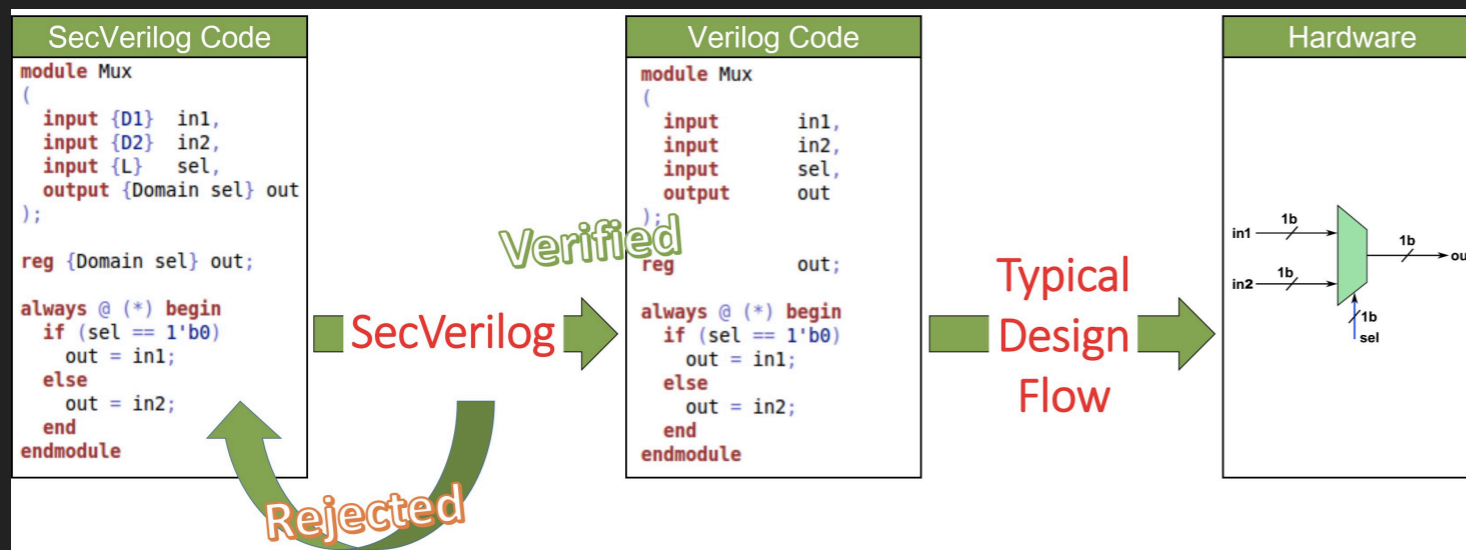
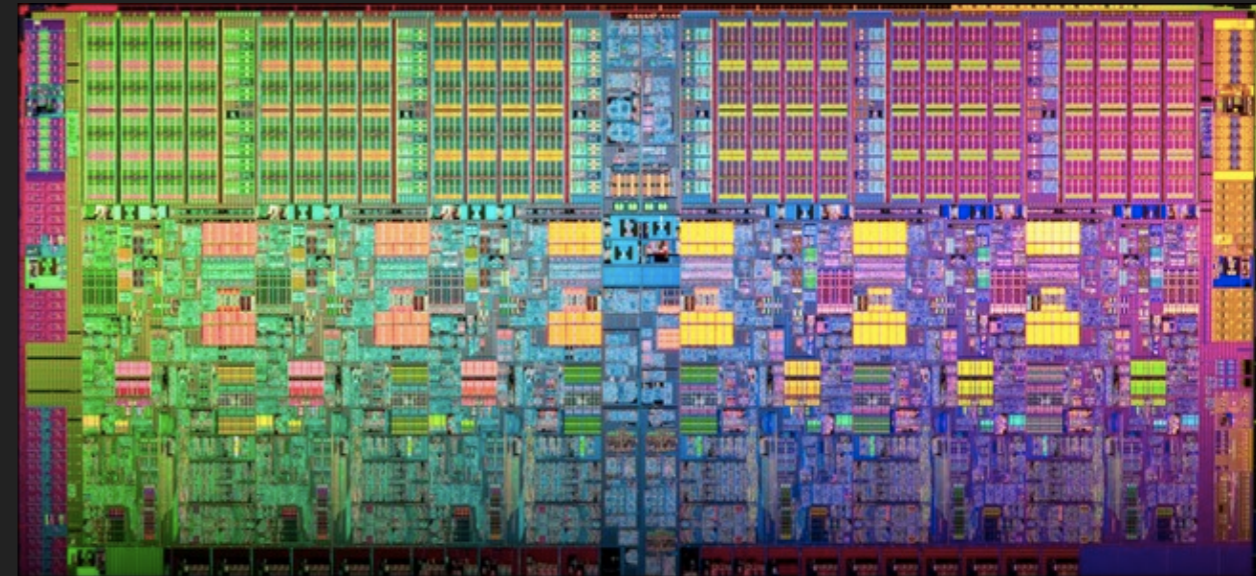
**WE PROVIDE OTHER CONSTRAINTS ON CALL GATES
TO PROVE NONMALLEABILITY**

MORE TECHNICAL DETAILS IN PAPER / TECHNICAL REPORT

- ▶ Attacker model, low equivalence relation, more explicit restrictions and *full proofs* of
Timing-Sensitive Noninterference
TS-Nonmalleable Information Flow
- ▶ Discussion on Exception Handling + Asynchrony
- ▶ Example Programs + Call Gate Usage

FUTURE WORK

- ▶ Model multicore and concurrency ISA operations



- ▶ How do we *verify* that HW satisfies *Microarchitectural NI*?
- ▶ Existing tools not expressive enough (SecVerilog '12, '19)

CONCLUSION

- ▶ General and portable contract for secure SW design
 - ▶ **Microarchitectural NI** Security Condition
Guides Secure HW Development
- ▶ Proof of **Timing-Sensitive Noninterference** and **Timing-Sensitive Nonmalleability**
- ▶ Practical primitives for downgrading and control transfer
 - ▶ Nonmalleability w/ Dynamic Labels
 - ▶ Call Gates provide Generalized Control Transfer

THANK YOU!