“Who ya gonna call?”

Cybersecurity for the Spectre Era

Calvin Deutschbein

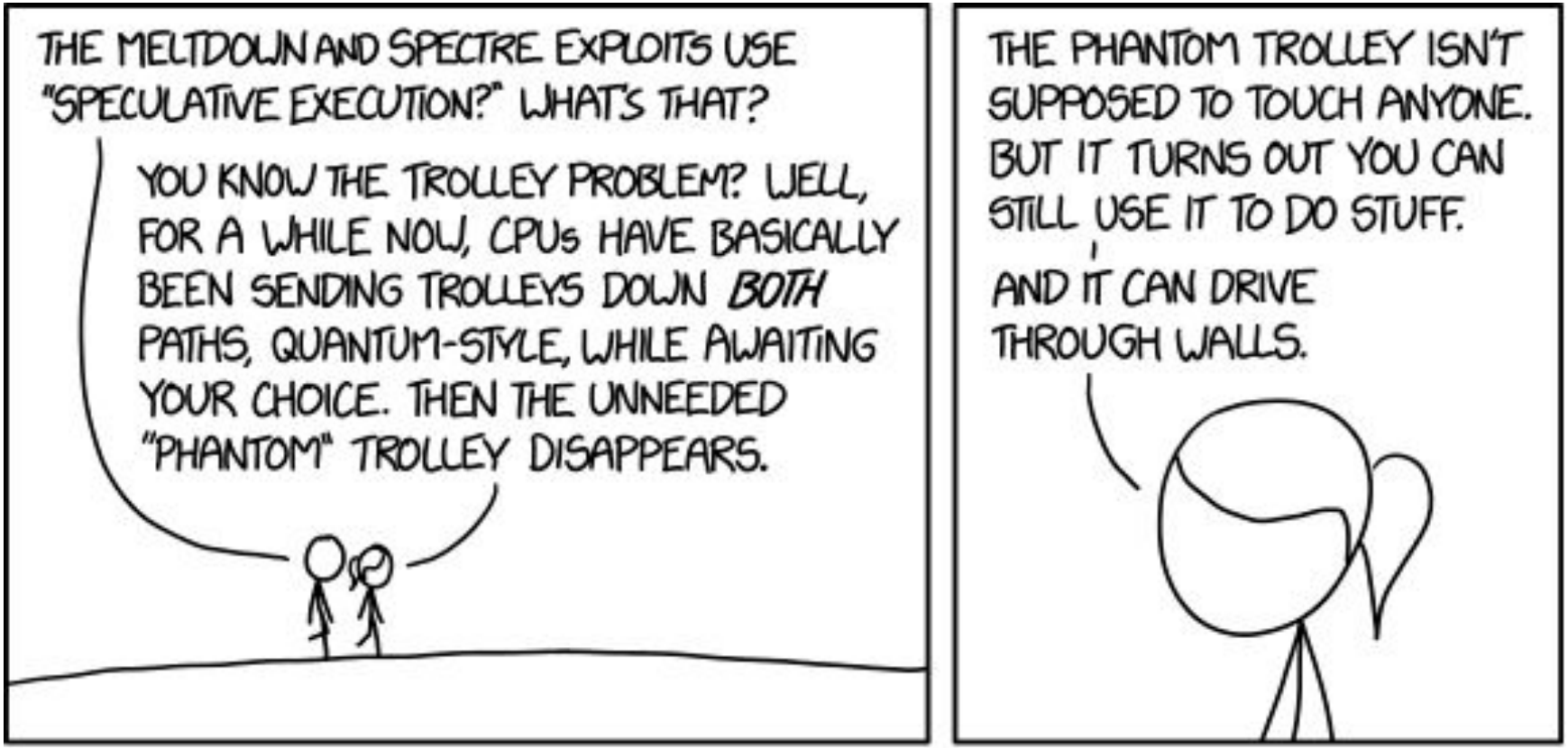
...the Spectre Era

3 Jan, 2018: Google Project Zero et al. publicly report the Spectre vulnerability.

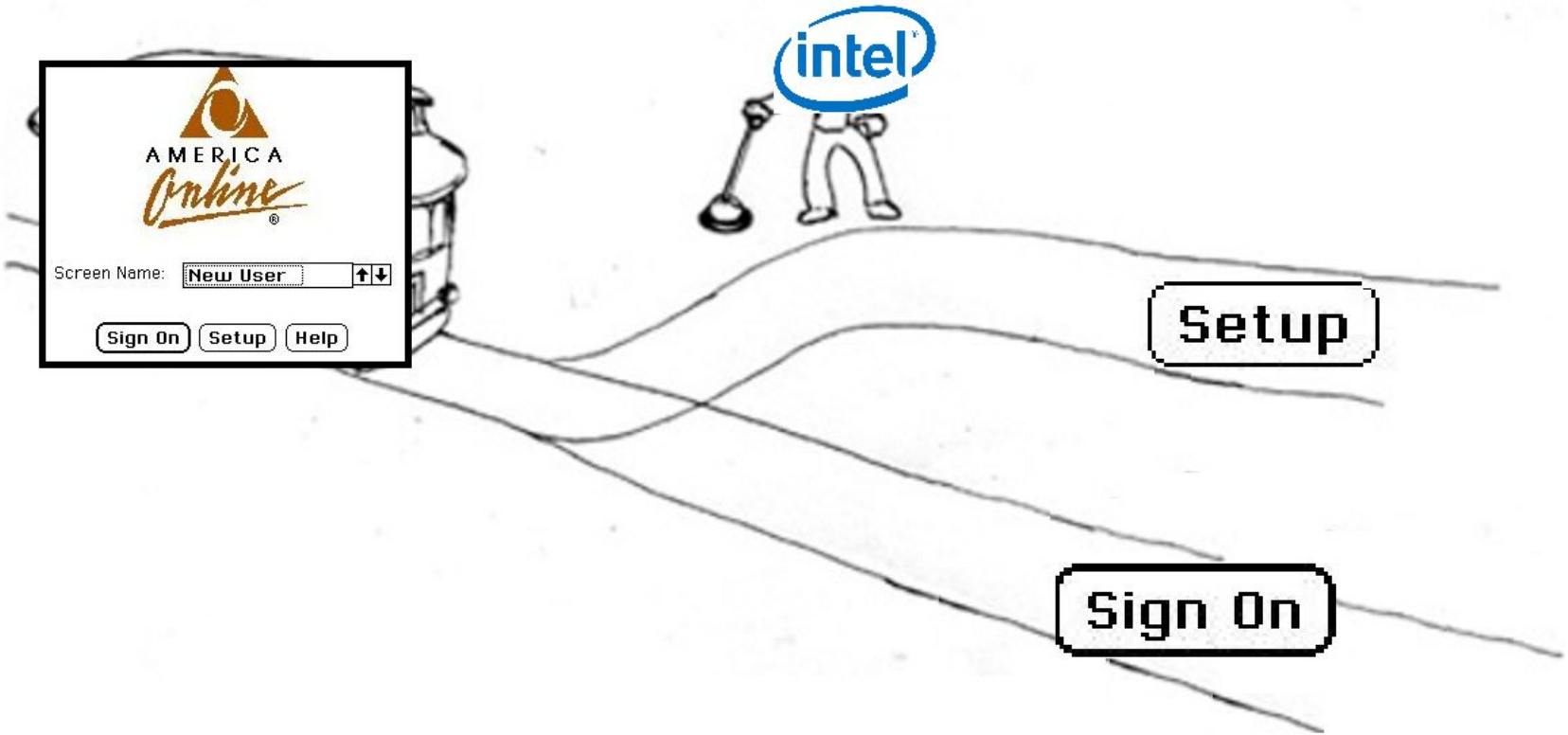
- Spectre targets **hardware** (all Intel processors since 1995)
- Spectre leaves **no traces** in traditional logs
- Spectre went **undetected for over two decades**

Learn more: meltdownattack.com/

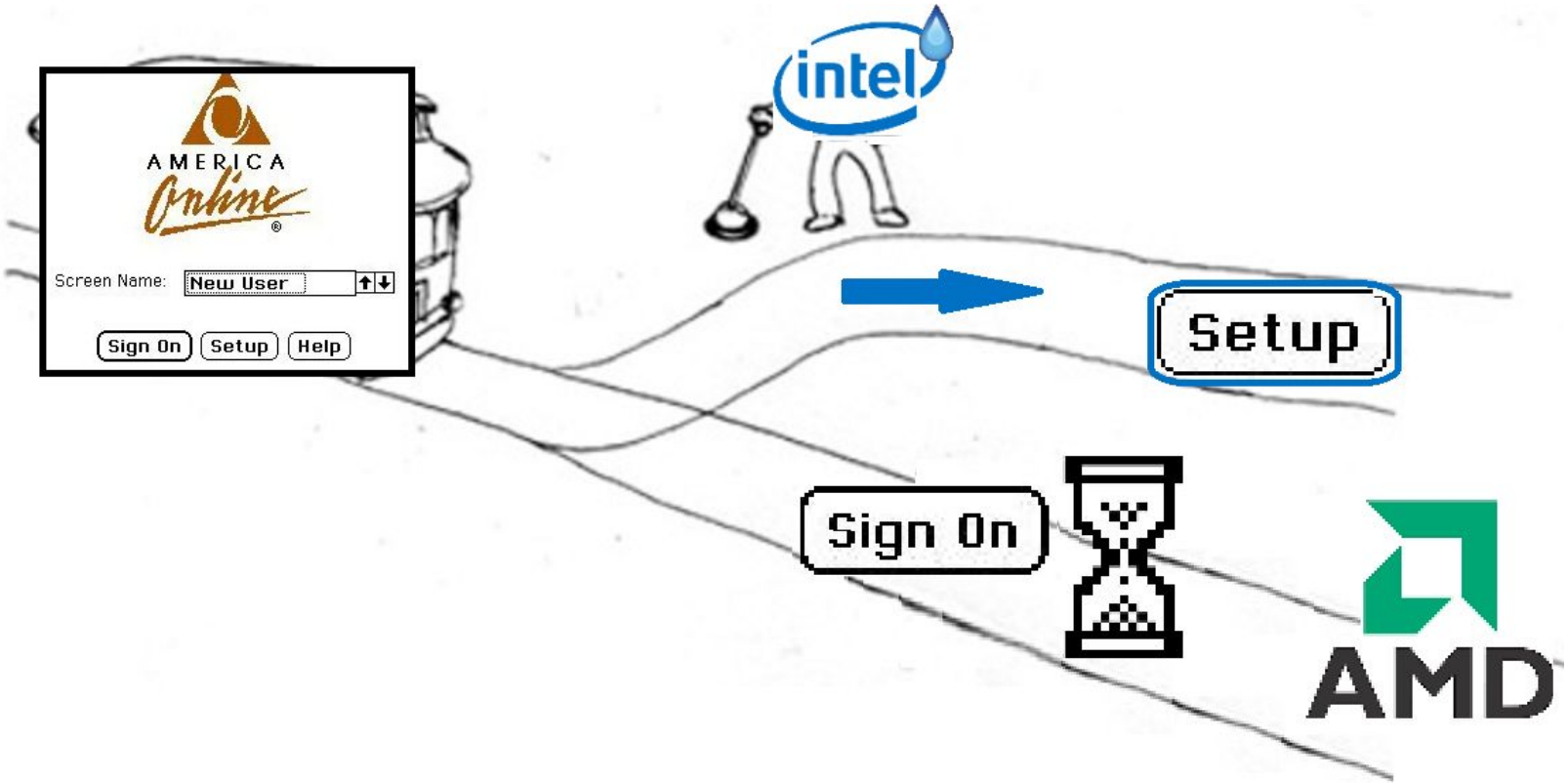
What is Spectre?



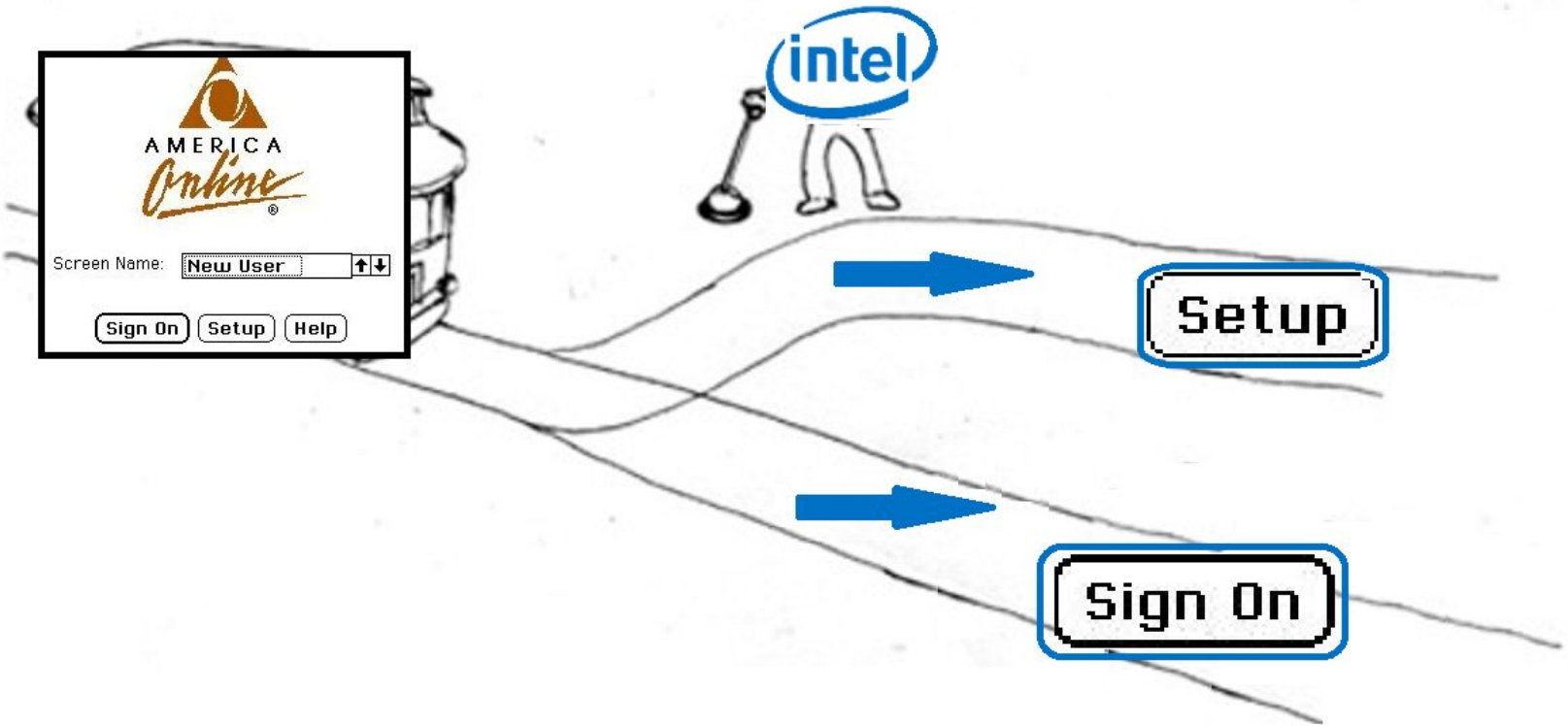
The Branch Prediction Problem in 1995



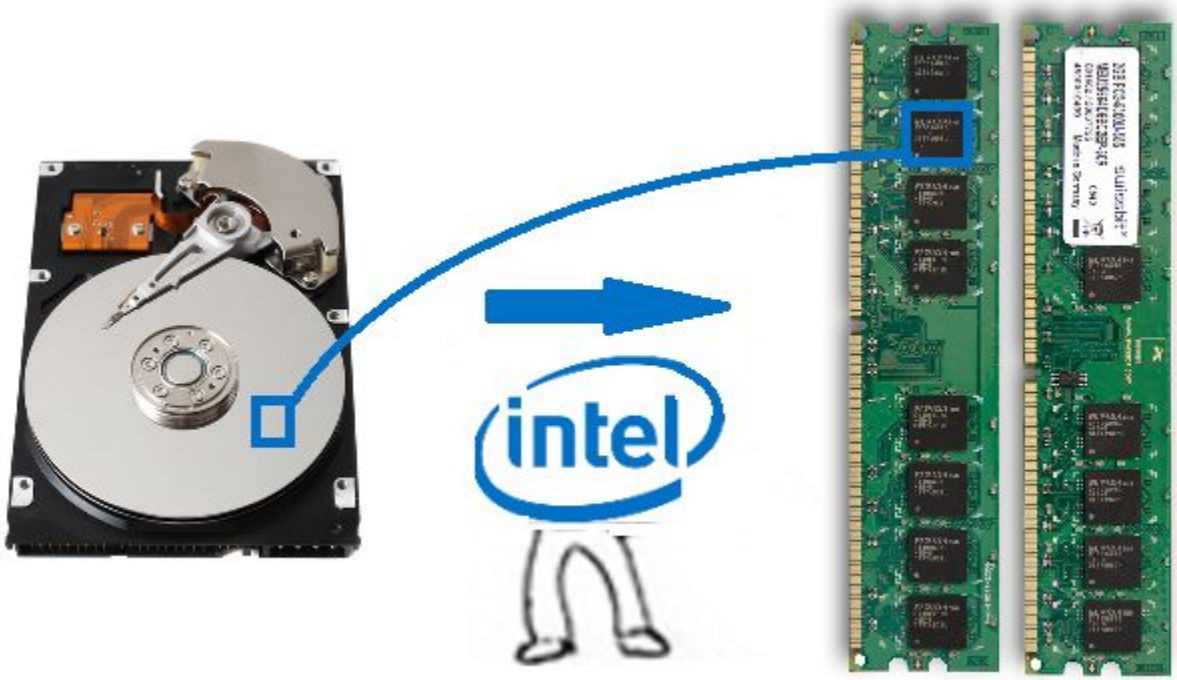
To avoid delays, x86 chips “guess” what’s next



But what if chips guess an unused branch?

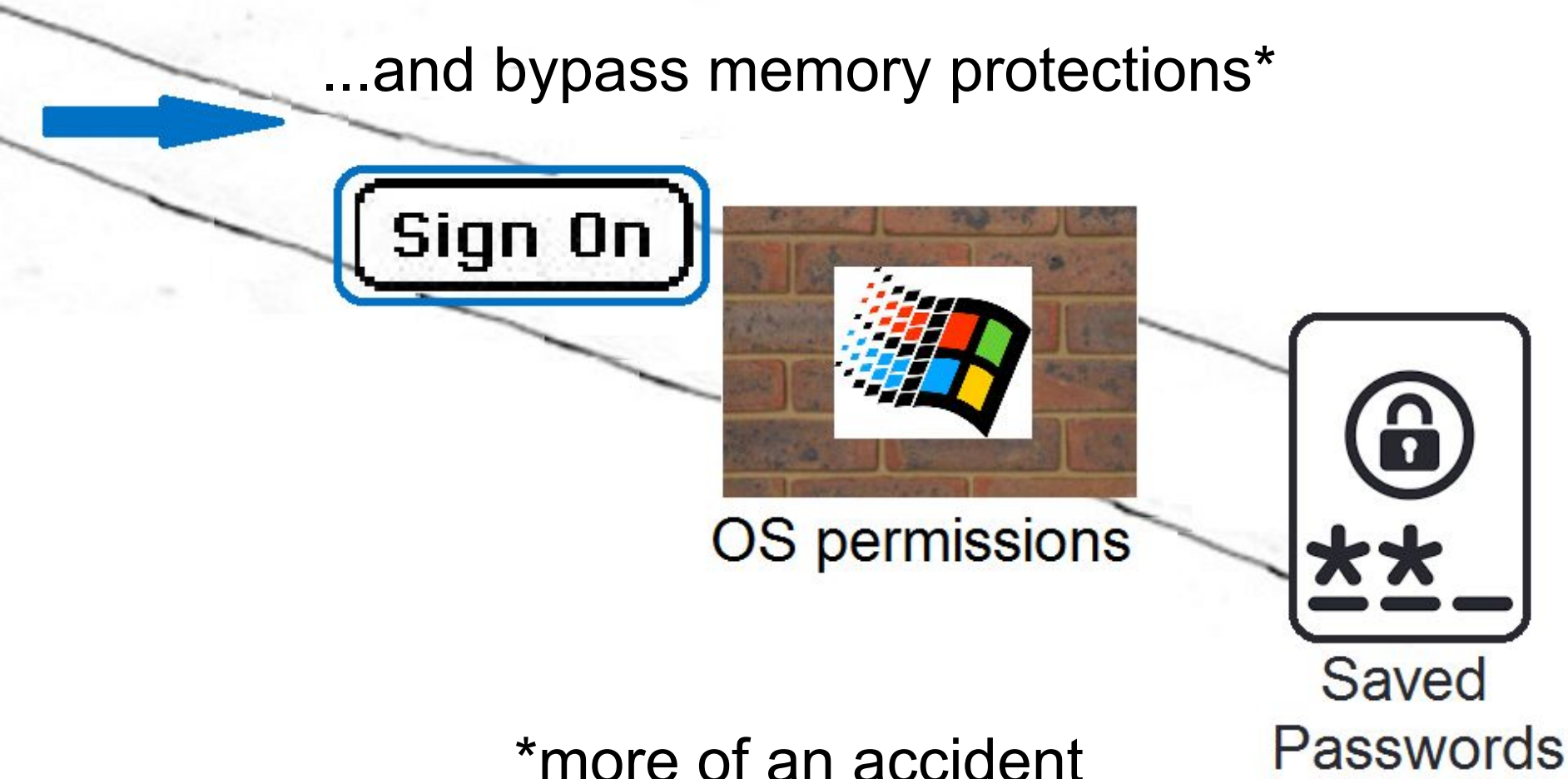


Speculative execution can access memory...*



*This is the point of speculative execution.

...and bypass memory protections*



*more of an accident

Exploiting the Vulnerability

Spectre is a **vulnerability** - it provides an entry point for an adversary.

- Spectre: “Branch misprediction may leave observable side effects”

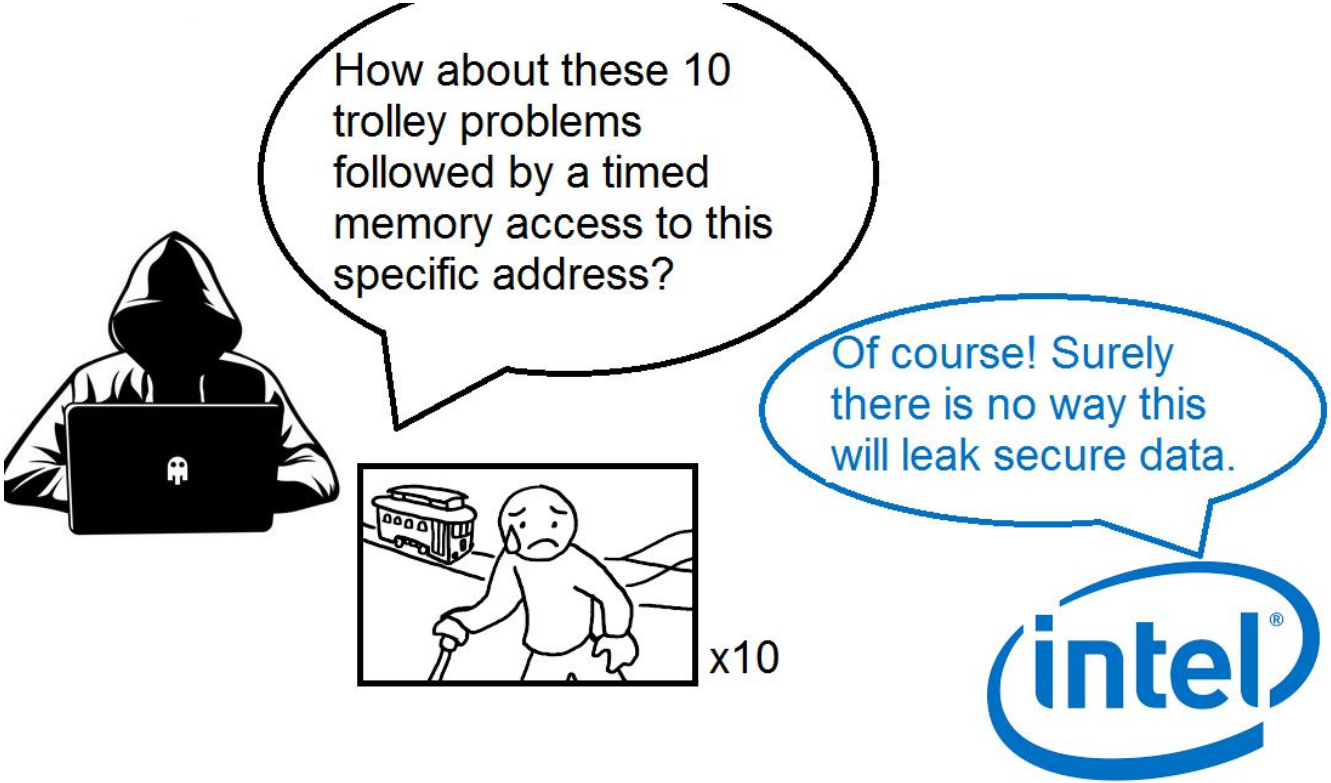
Adversaries must **exploit** the vulnerability to gain access to secure data.

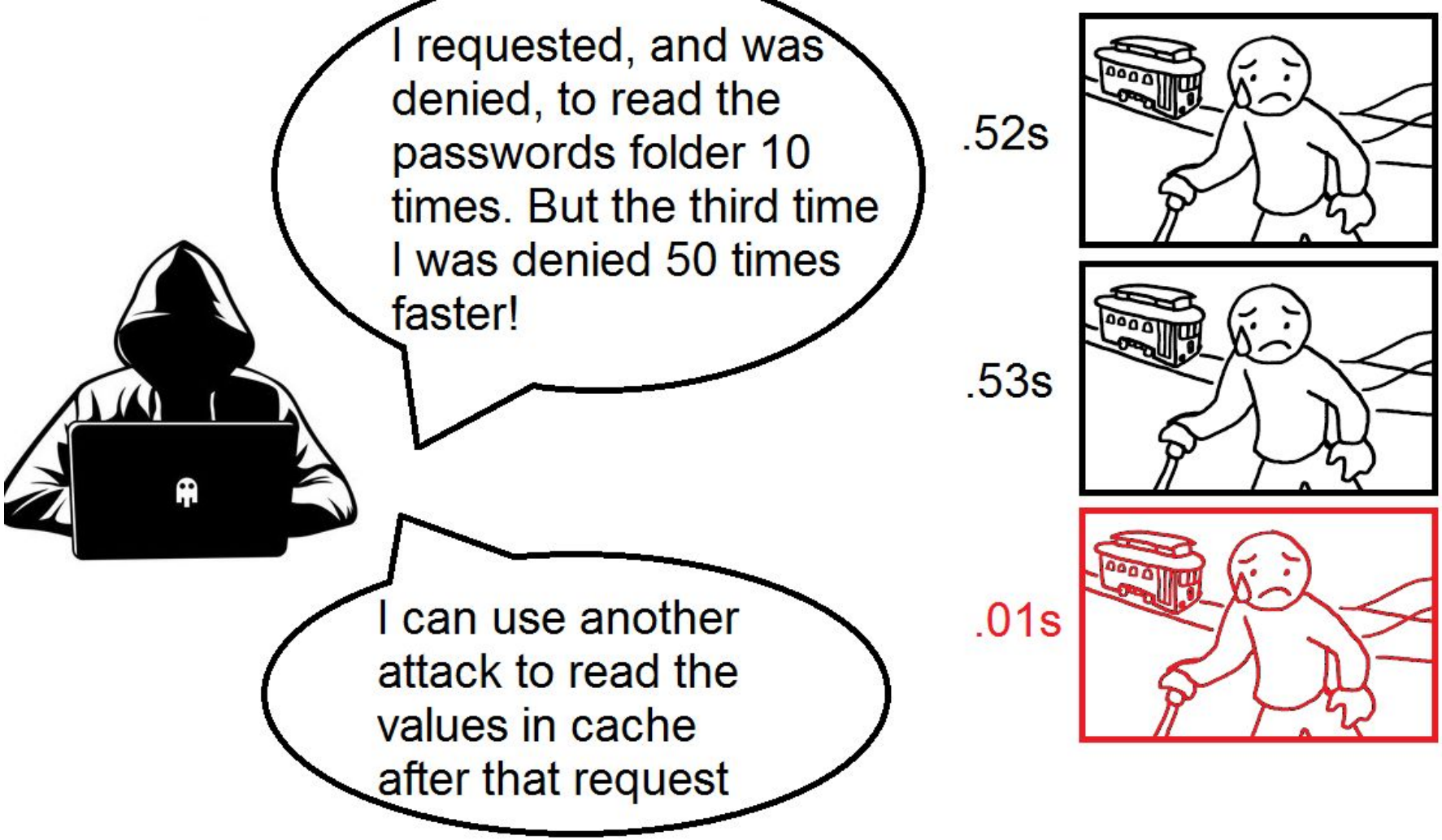
- Exploit: “Using observable side effects to access secure data.”

The mere presence of the vulnerability in on hardware not running code capable of exploiting the vulnerability will not result in a security violation.

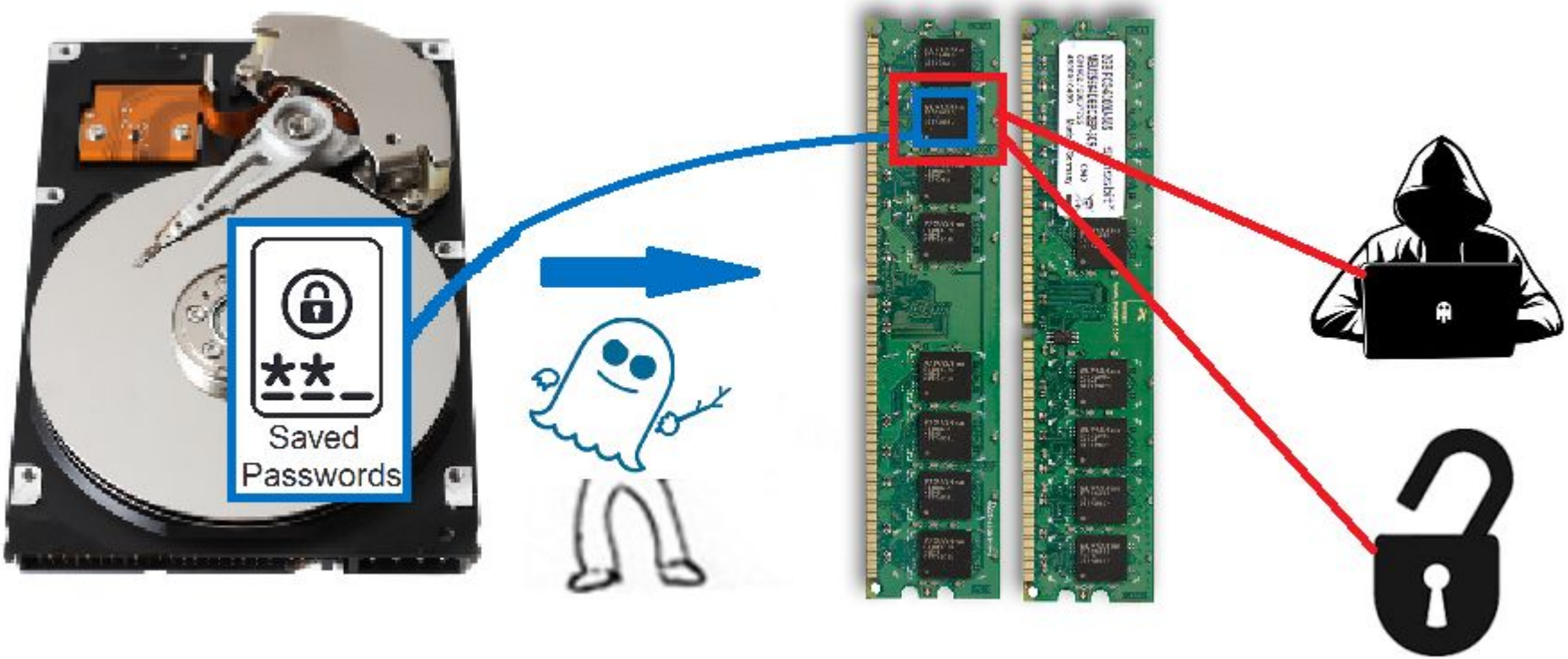
How adversaries can exploit Spectre

How adversaries can exploit Spectre



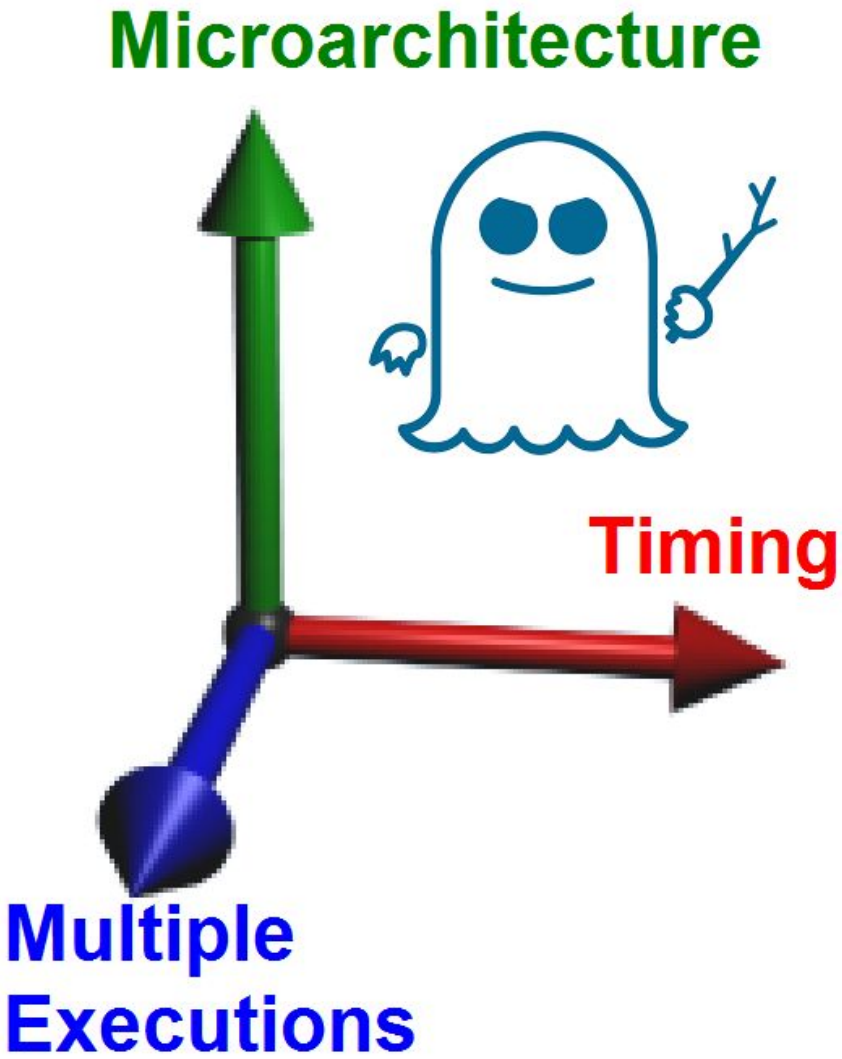

I requested, and was denied, to read the passwords folder 10 times. But the third time I was denied 50 times faster!
.52s
.53s
.01s
I can use another attack to read the values in cache after that request

Thus, an attacker gains access to secure data



Three attack surfaces:

- **Timing**
  - Timing cache hits
- **Microarchitecture**
  - Branch prediction
  - Below operating system or assembly
- **Multiple Executions**
  - Multiple runs expose timing differences

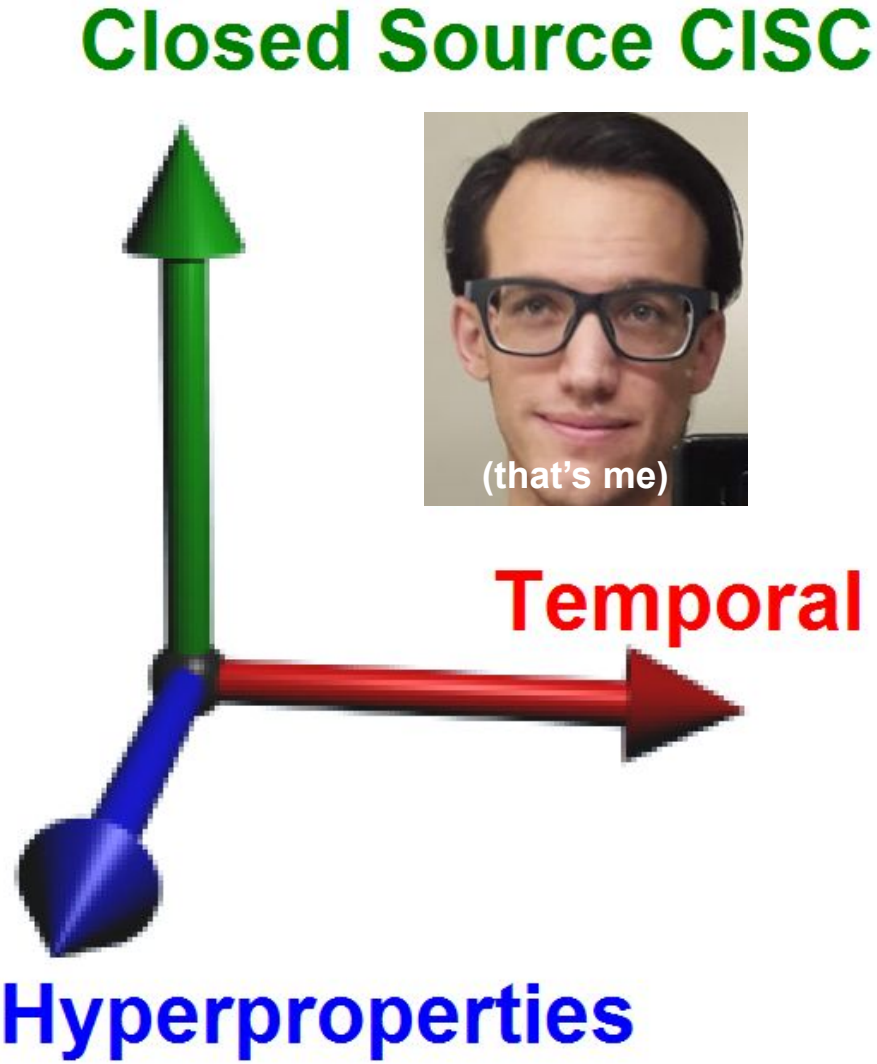


So… “Who ya gonna call?”

Mining Behavior

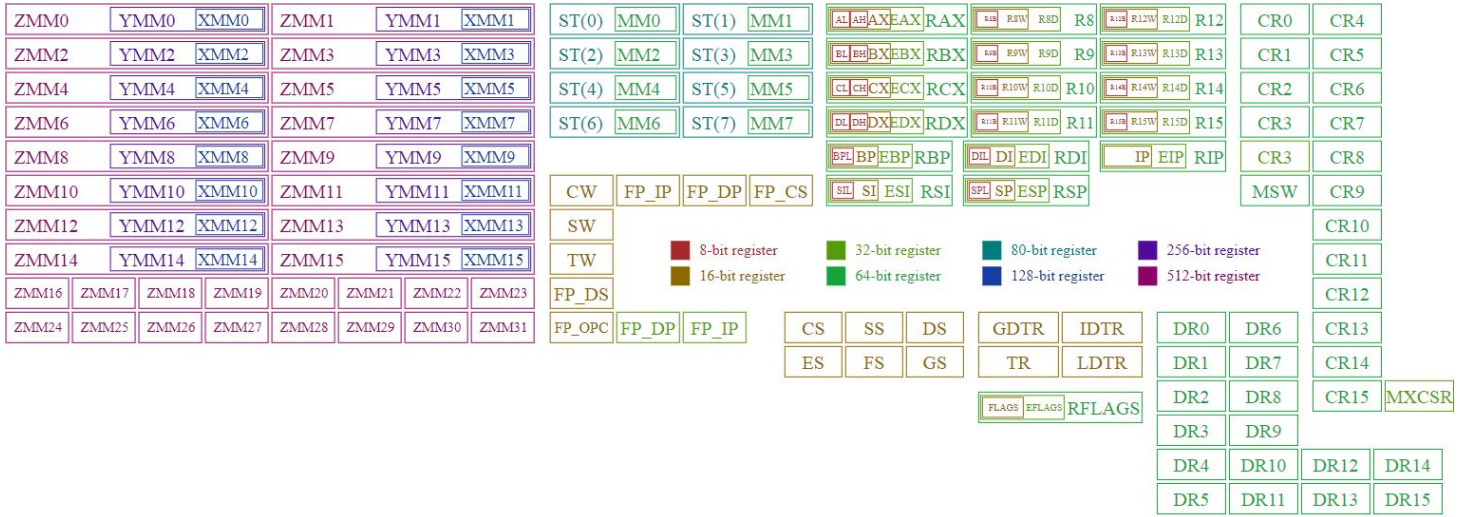
My research shows the technique of **specification mining** can find:

- **Temporal** properties, for timing
- **Closed source CISC** architecture properties, for microarchitecture
- **Hyperproperties**, properties over multiple traces of execution



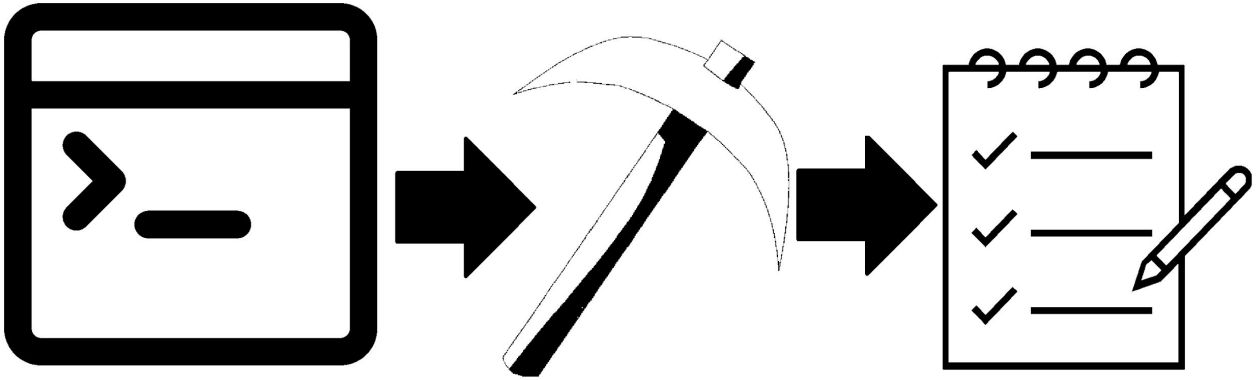
Defining Secure Behavior

For x86-64, no specification exists - so we create one.



Specification Mining

- Miners accept as input **traces of execution**.
  - For example, the debug output of an x86-64 processor booting Linux.
- Miners find **properties** that hold over the traces.
  - For example, “if reset is active, then the privilege level is supervisor”.
  - RESET==0 ⇒ CURRENT_PRIVILEGE_LEVEL==0
- Miners contain powerful **inference engines** for high performance.



Undine: Mining Temporal Properties

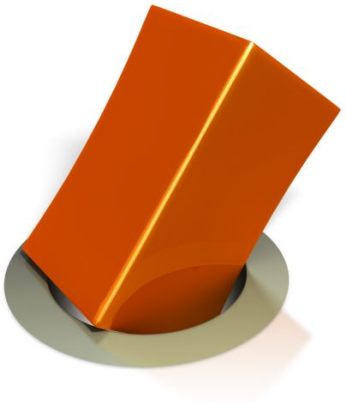
Can **linear temporal logic** properties that model secure behavior be discovered using specification mining?

A library of typed templates for my miner, Undine, enable it to find security temporal properties, including properties using **G** (Globally) or **X** (Next) operators.

Difficulties Finding Security Properties

Too Many Properties

Properties Not Security Related



Do Not Capture Semantic Info

Without separate events there are many properties

Sample Trace

```
reg_a==1
reg_b==1
reg_c==0
reg_d==0
reg_a==reg_b
reg_c==reg_d
```

Mined 30 G(x → y)

```
reg_a==1 → reg_b==1
…
reg_a==1 → reg_c==reg_d
…
reg_c==reg_d → reg_a==reg_b
```

Templates Refine to Security Properties

Sample Trace

```
reg_a==1
reg_b==1
reg_c==0
reg_d==0
reg_a==reg_b
reg_c==reg_d
```

Mined 8 G(R → R-R)

```
reg_a==1 → reg_a==reg_b
reg_a==1 → reg_c==reg_d
reg_b==1 → reg_a==reg_b
…
reg_f==0 → reg_c==reg_d
```

Register Roles Refine Further

Sample Trace

```
reg_a==1
reg_b==1
reg_c==reg_d
```

Mined 2 G(R → R-R)

```
reg_a==1 → reg_c==reg_d
reg_b==1 → reg_c==reg_d
```

Register Slices Uncover Semantic Meaning

Sample Trace

```
reg_a==7
#tick
reg_a==3
#tick
reg_a==5
...
```

Mining G(a)

<no properties>

Register Slices Uncover Semantic Meaning

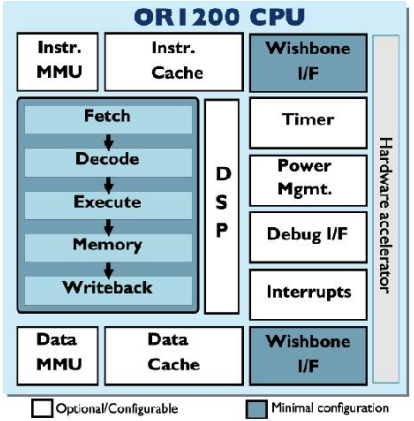
Sample Trace

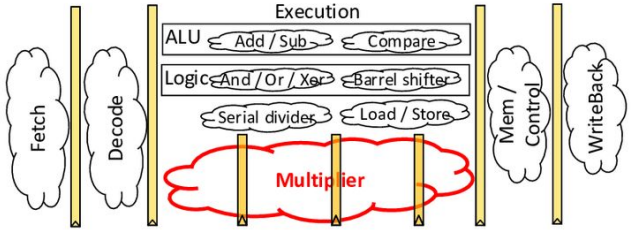
```
reg_a[0]==1
reg_a[1]==1
#tick
reg_a[0]==1
reg_a[1]==1
#tick
reg_a[0]==1
reg_a[1]==0
```

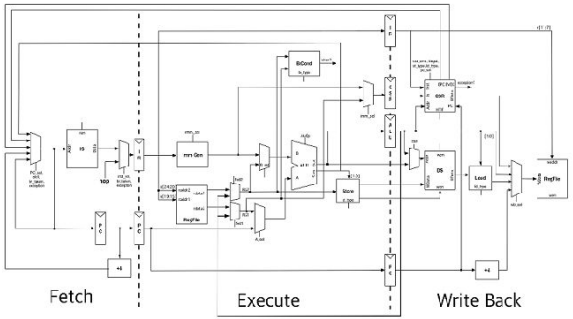
Mining G(a)

```
reg_a[0]==1
```

Tested on 3 Processors







OR1200 mor1kx RISC-V

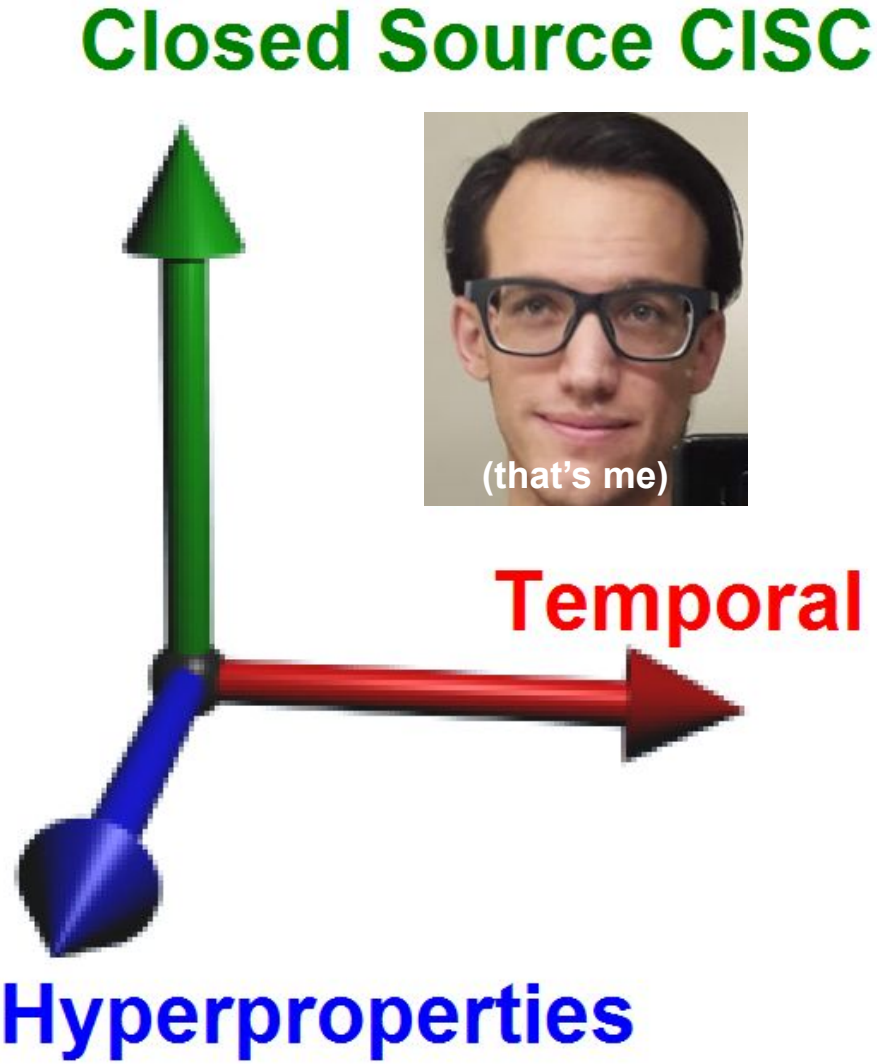
Undine: Mining Temporal Properties

Undine can discover linear temporal logic security properties such as those related to correct initialization of a system using a library of typed templates.

Mining Behavior

My research shows the technique of **specification mining** can find:

- ***Temporal*** *properties, for timing*
- **Closed source CISC** architecture properties, for microarchitecture
- **Hyperproperties**, properties over multiple traces of execution

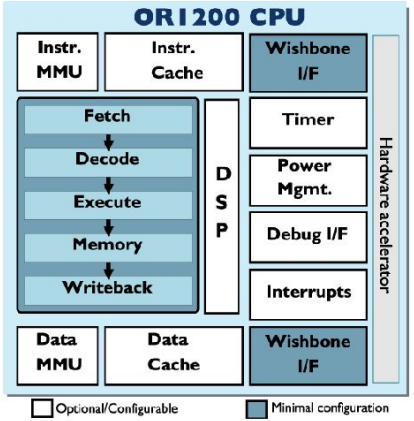


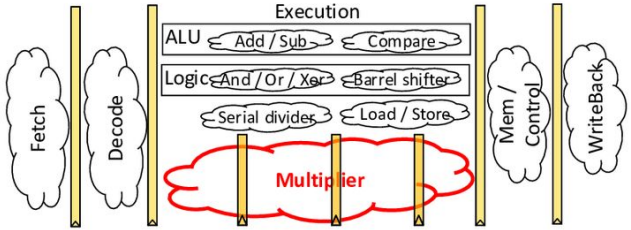
Astarte: Mining Closed Source CISC

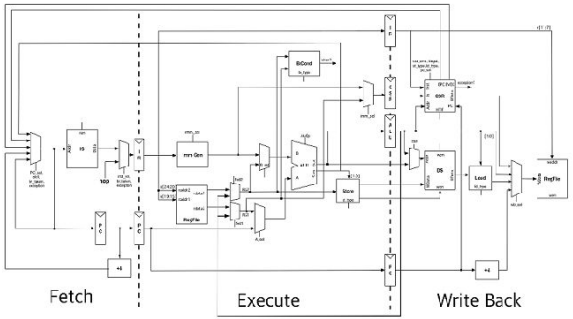
How can properties that model secure behavior of **closed source complex instruction set computer (CISC)** designs be discovered using specification mining?

Mining for control signals in the design then mining preconditioned on those control signals yields security properties of the design.

Recall: Undine Tested on 3 Processors

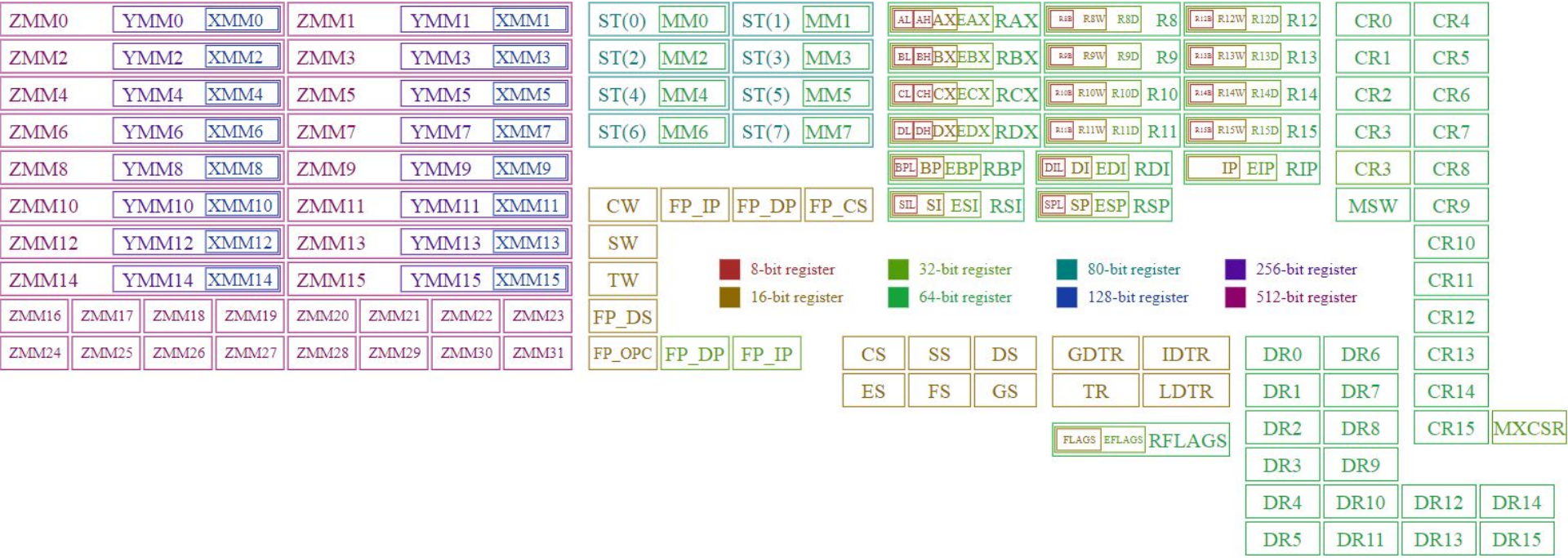




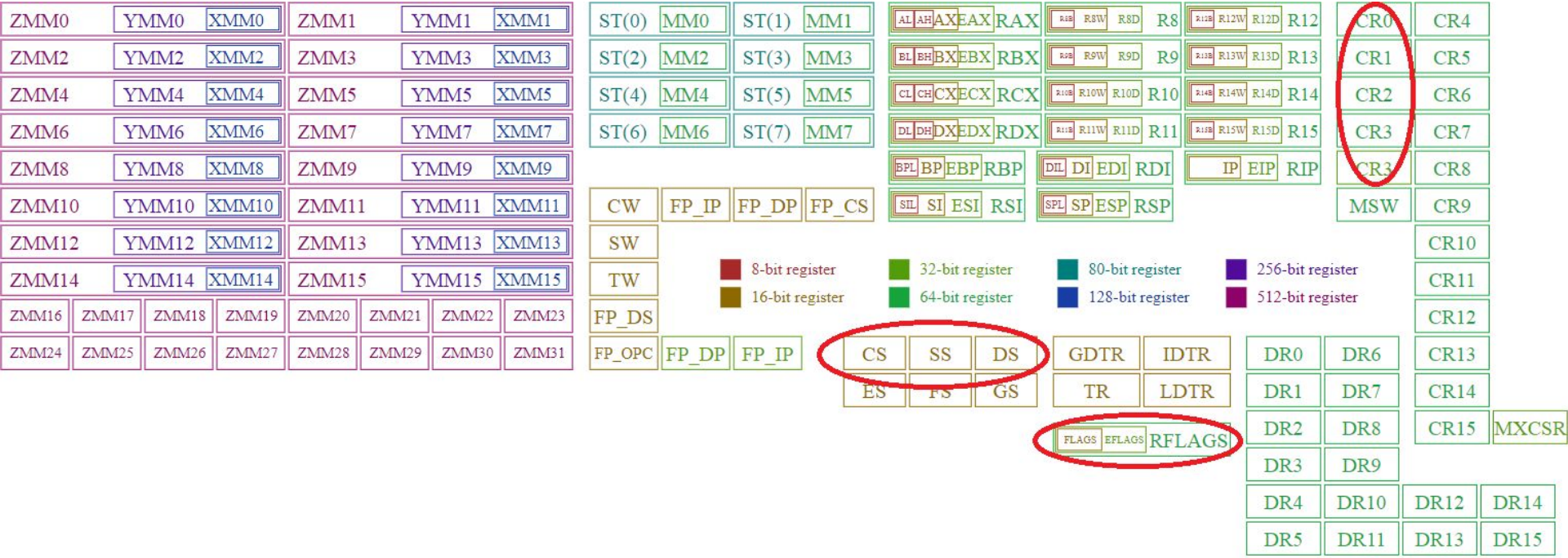


OR1200 mor1kx RISC-V

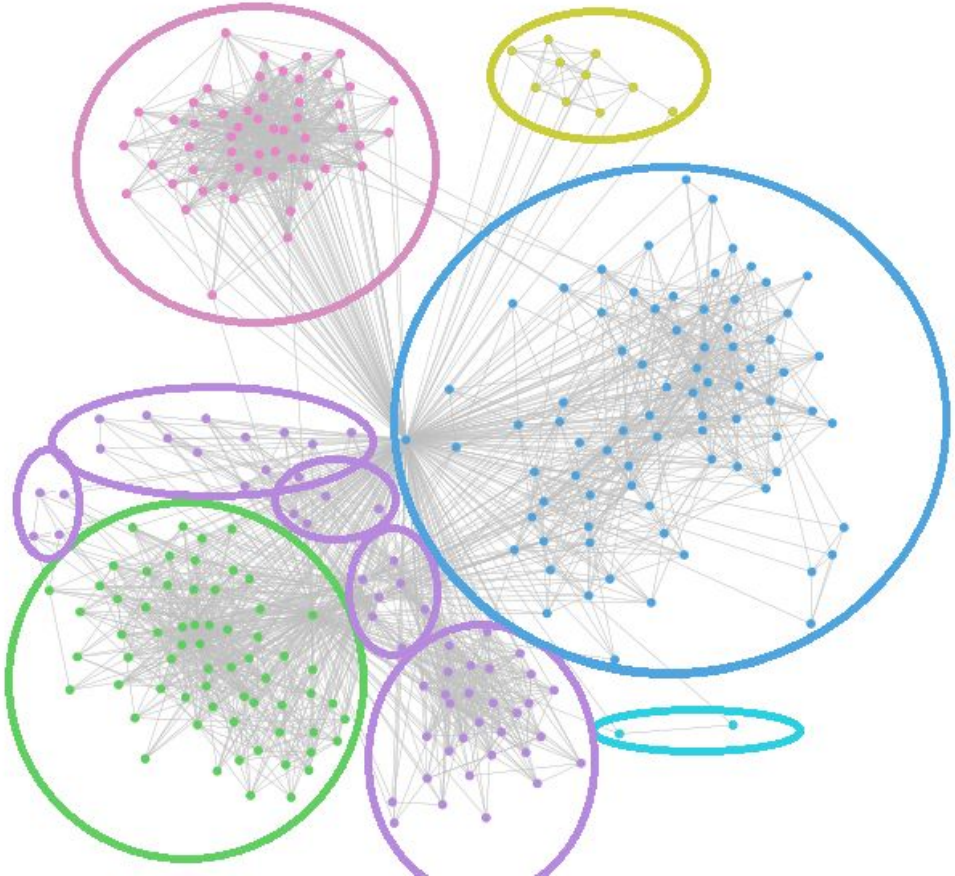
All were Open Source and RISC! x86 is neither!



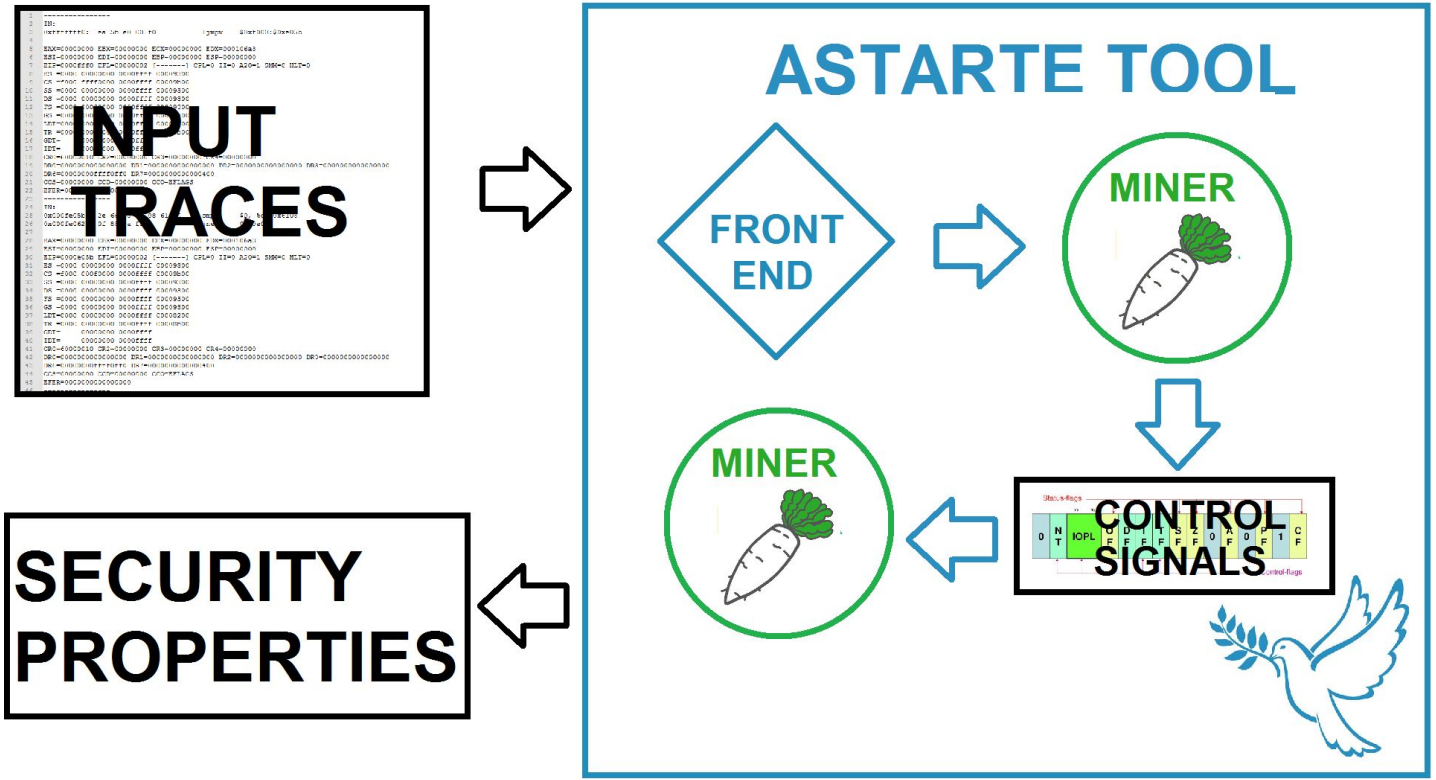
The x86 specification has many control signals...



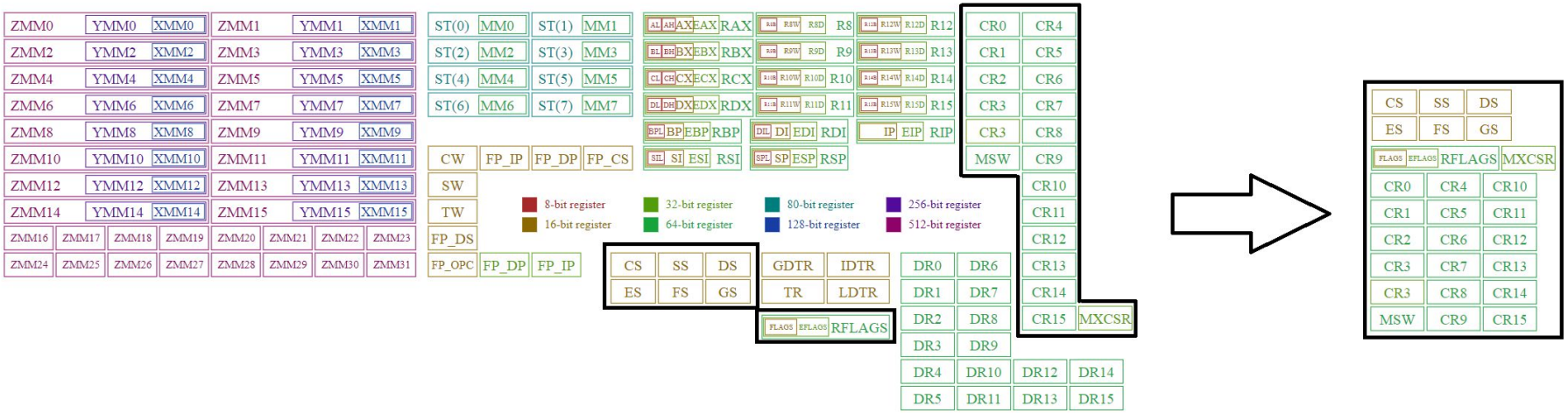
Control Signals Partition the Space



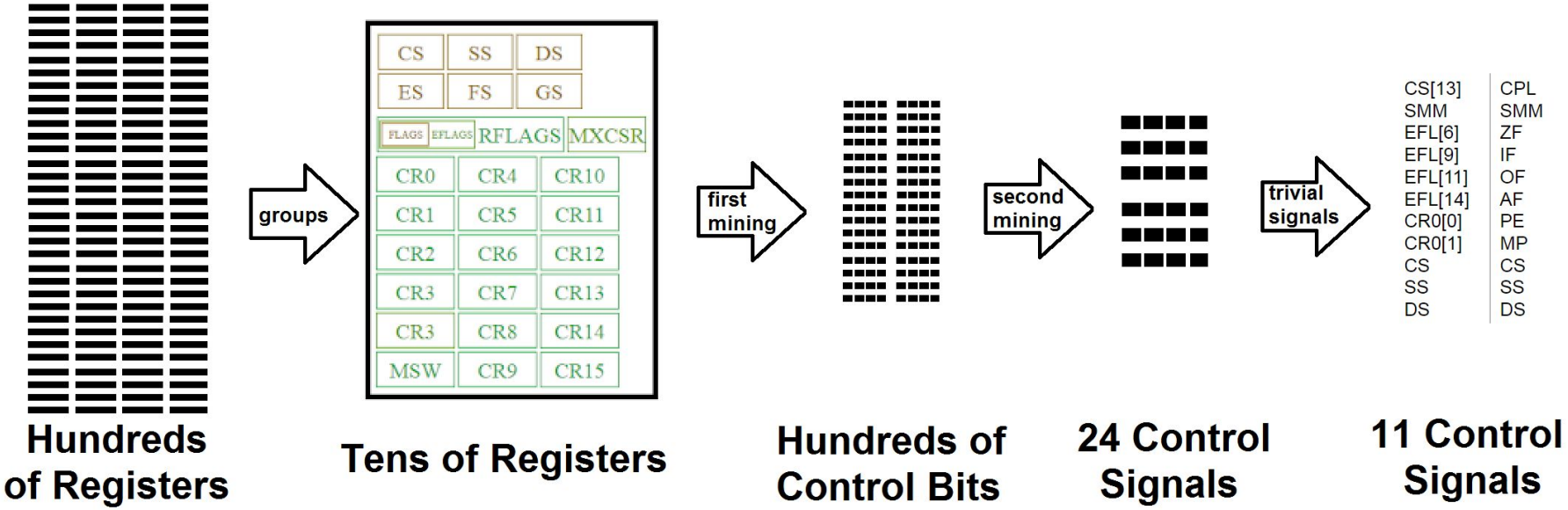
So I created a tool to find properties using signals.



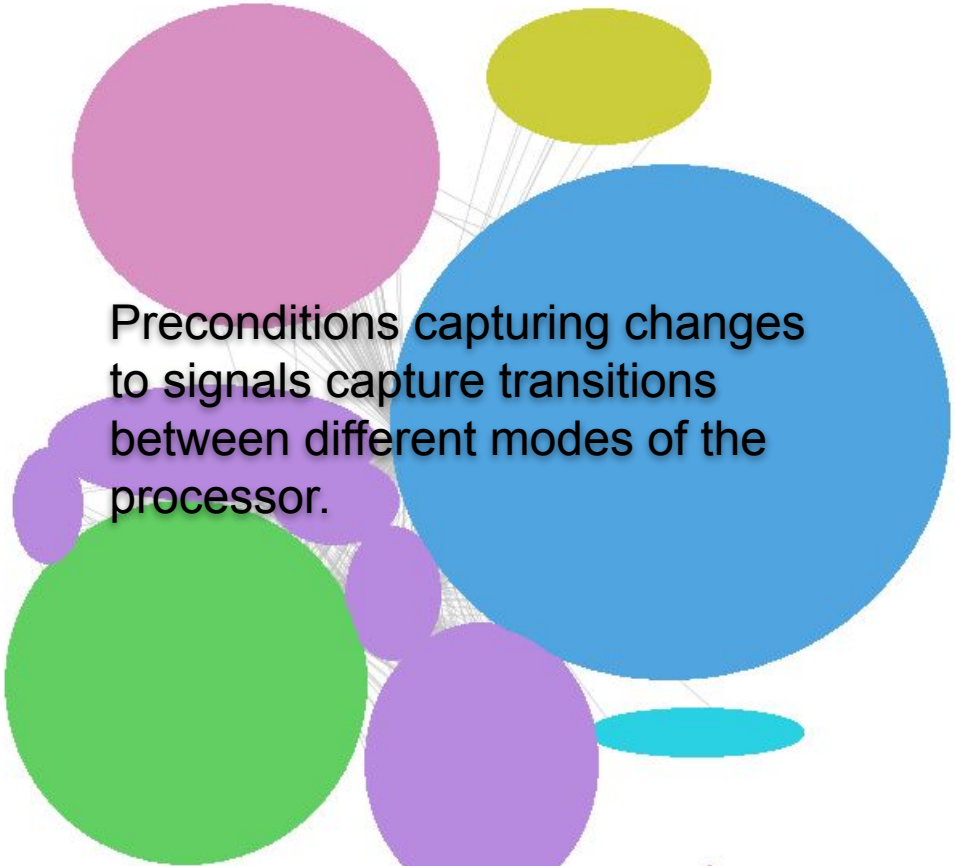
Front End: Registers Placed in Groups



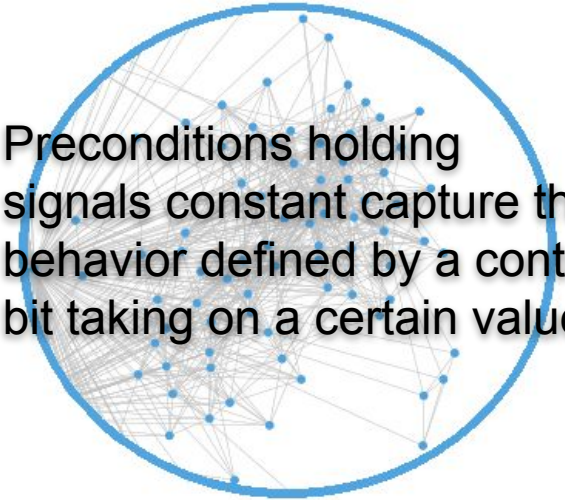
Property Refinement



Control Signals Partition the Space



Preconditions capturing changes to signals capture transitions between different modes of the processor.



Preconditions holding signals constant capture the behavior defined by a control bit taking on a certain value.

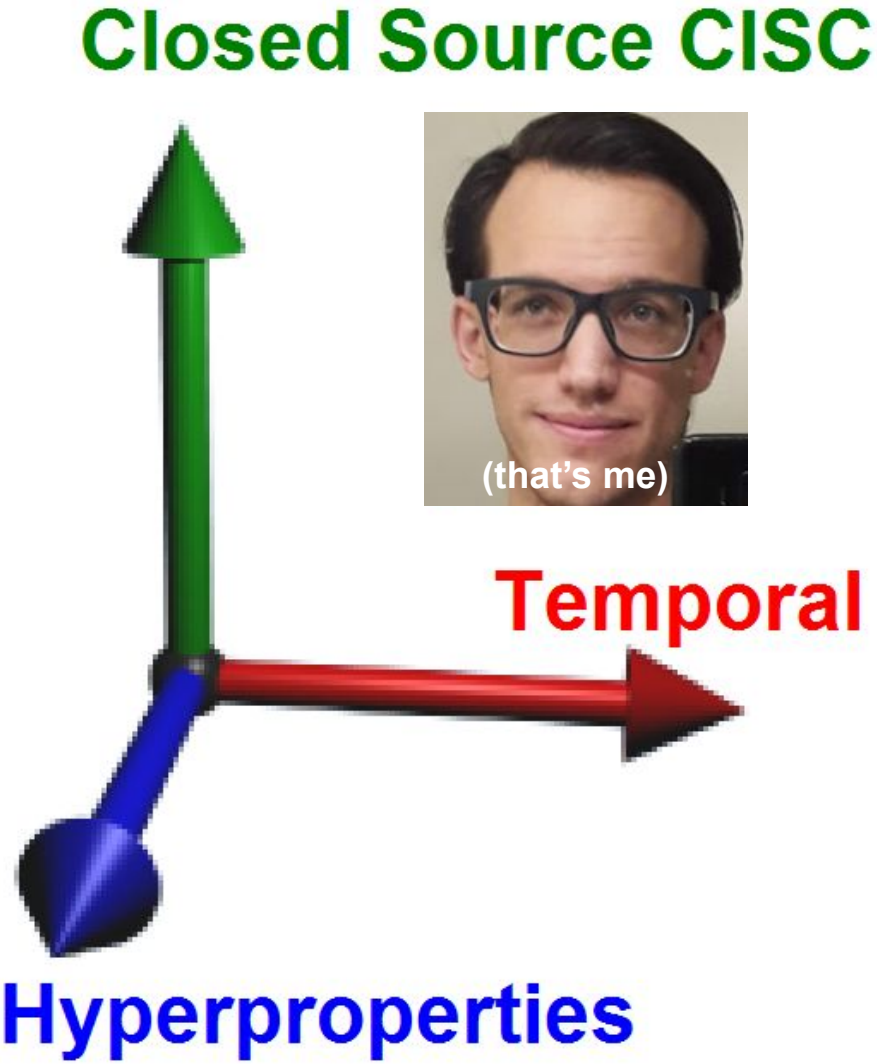
Astarte: Mining Closed Source CISC

Specification mining can discover security properties preconditioned on control signals in closed source CISC designs.

Mining Behavior

My research shows the technique of **specification mining** can find:

- **Temporal** properties, for timing
- ***Closed source CISC*** *architecture properties, for microarchitecture*
- **Hyperproperties**, properties over multiple traces of execution

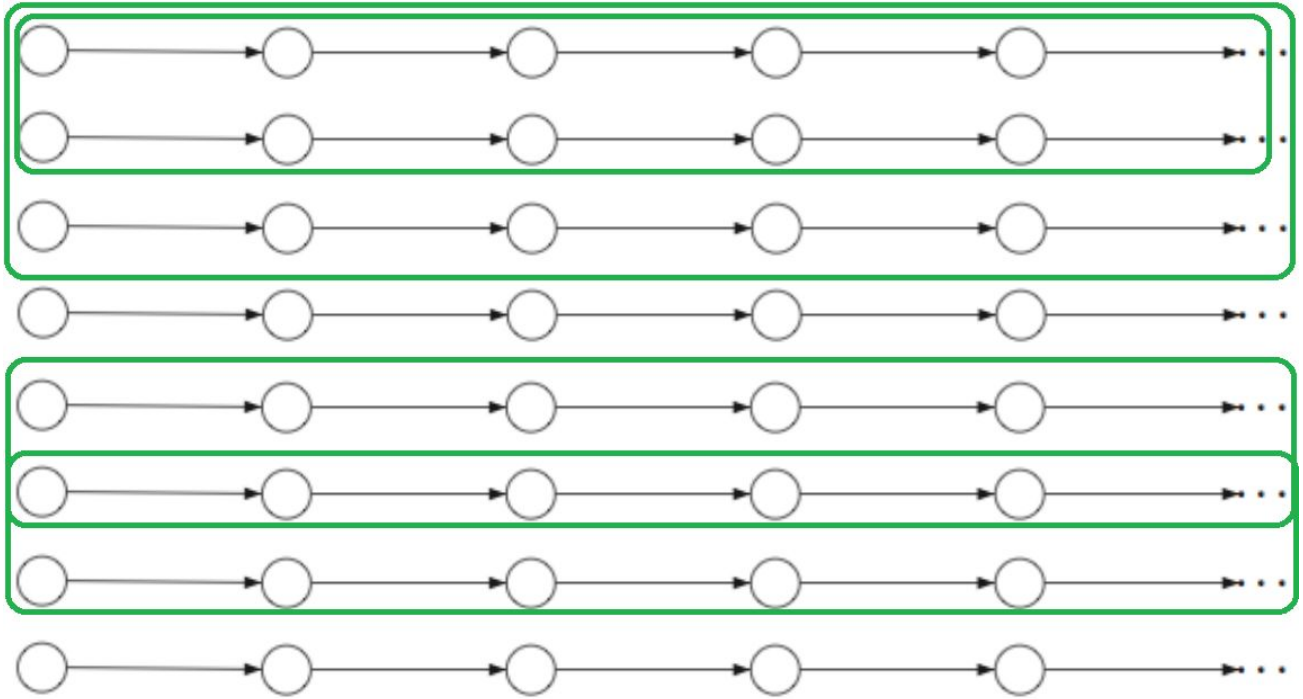


Isadora: Mining Hyperproperties (Current work)

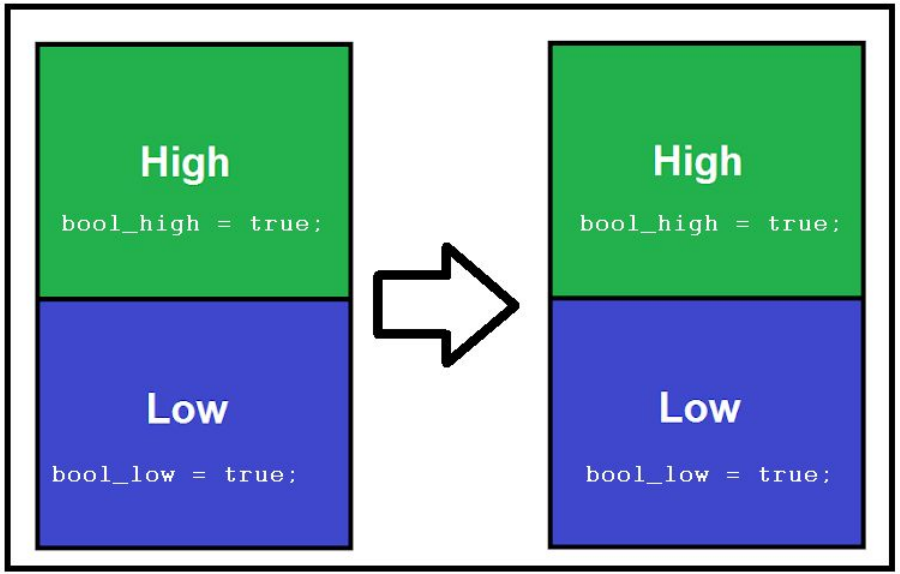
How can **hyperproperties** that model secure behavior of designs be discovered using specification mining?

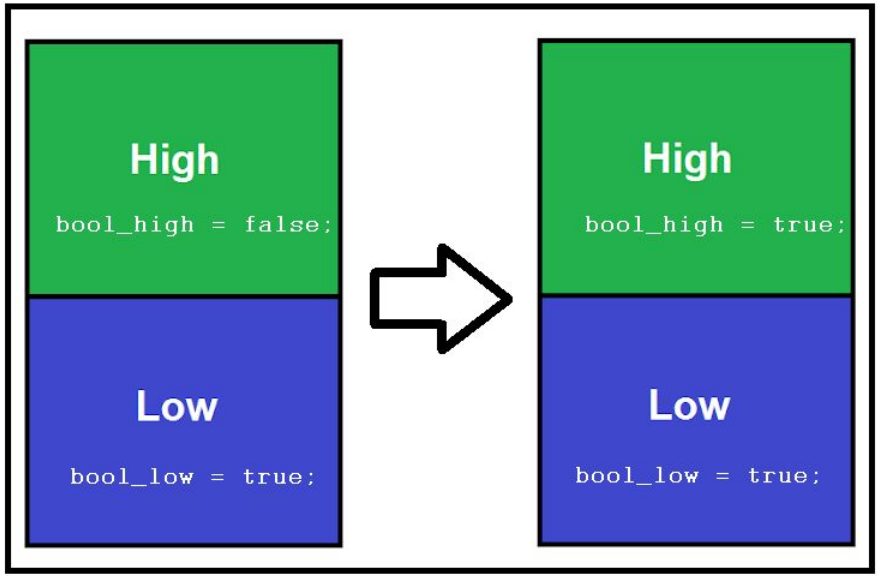
Hyperproperties

Sets of Sets of Traces, or Sets of Properties



Example: *GMNI* (Noninterference)



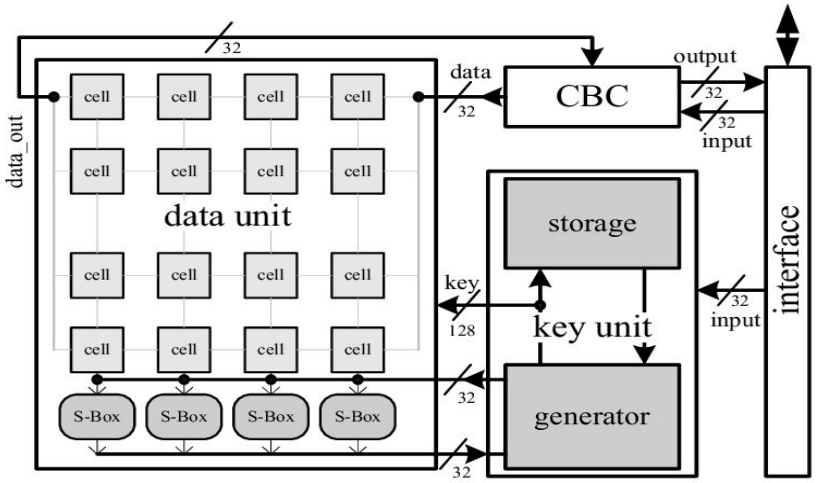


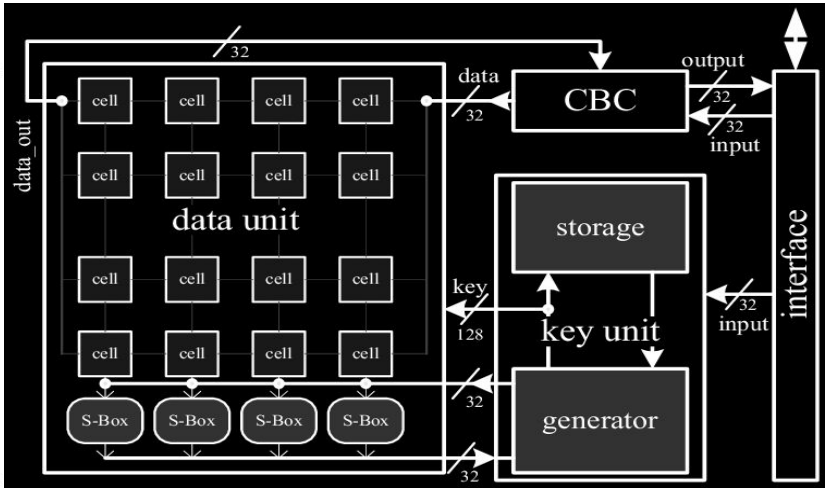
“High” could be OS, “Low” could be adversary

Instrumentation

To find hyperproperties, use Information Flow Tracking (IFT) instrumentation.

- IFT creates a shadow register for all design registers to track information flow
- *GMNI* is an information flow hyperproperty





Problem Statement

How can Information Flow Tracking (IFT) and specification mining determine

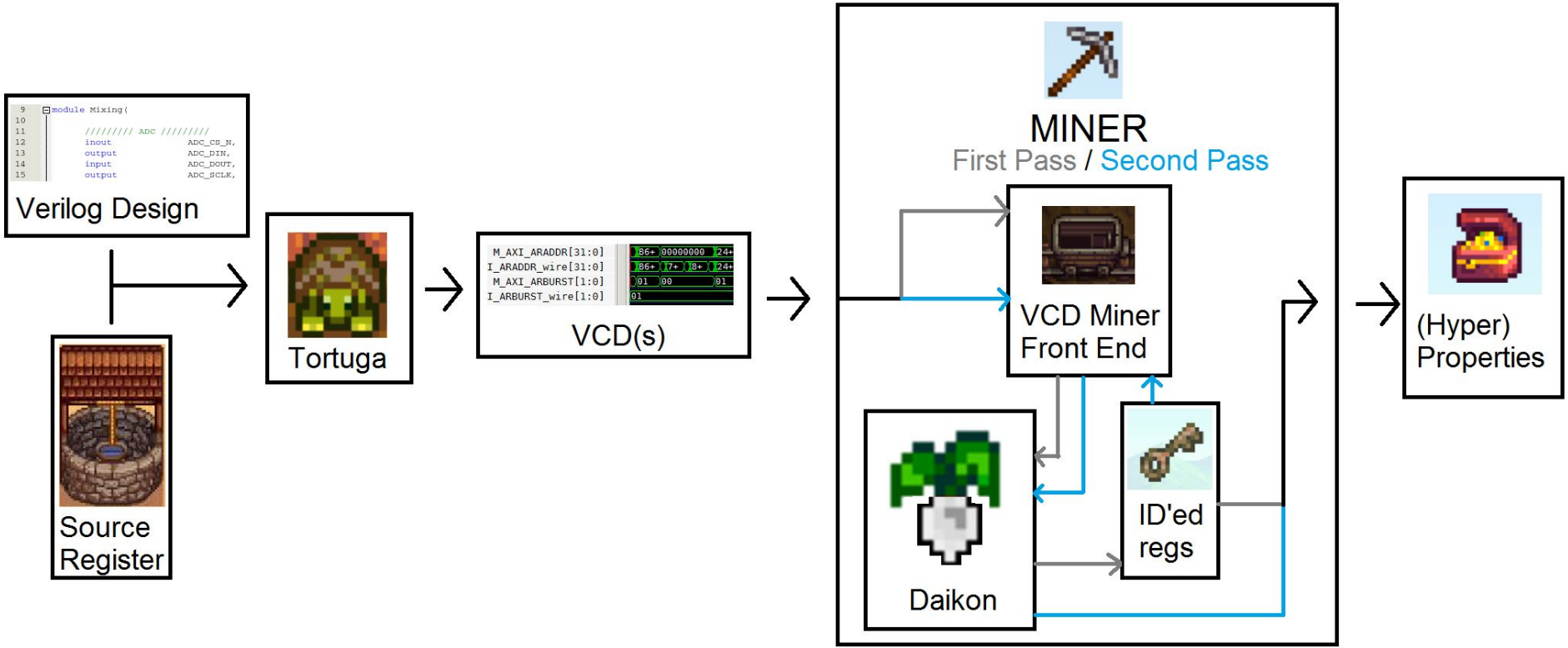
where and **when**

interference occurs in a design from any arbitrary source?

Tracking Information Flow

- Given a source, registers can be in one of three categories:
  - Always a sink: source = => sink (“flows to”)
  - Never a sink: source =/=> sink (“does not flow to”)
  - Conditionally a sink source =/=> sink UNLESS <boolean expression>

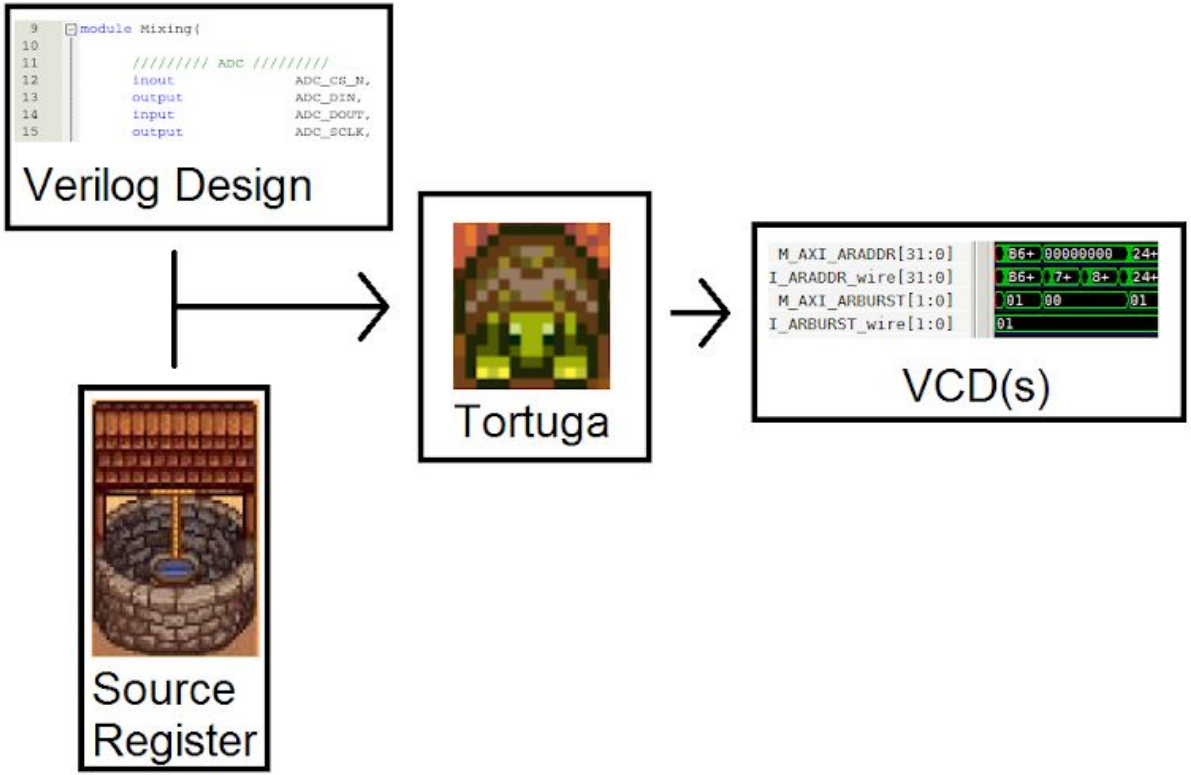
Research Technique Sketch



Trace Detail

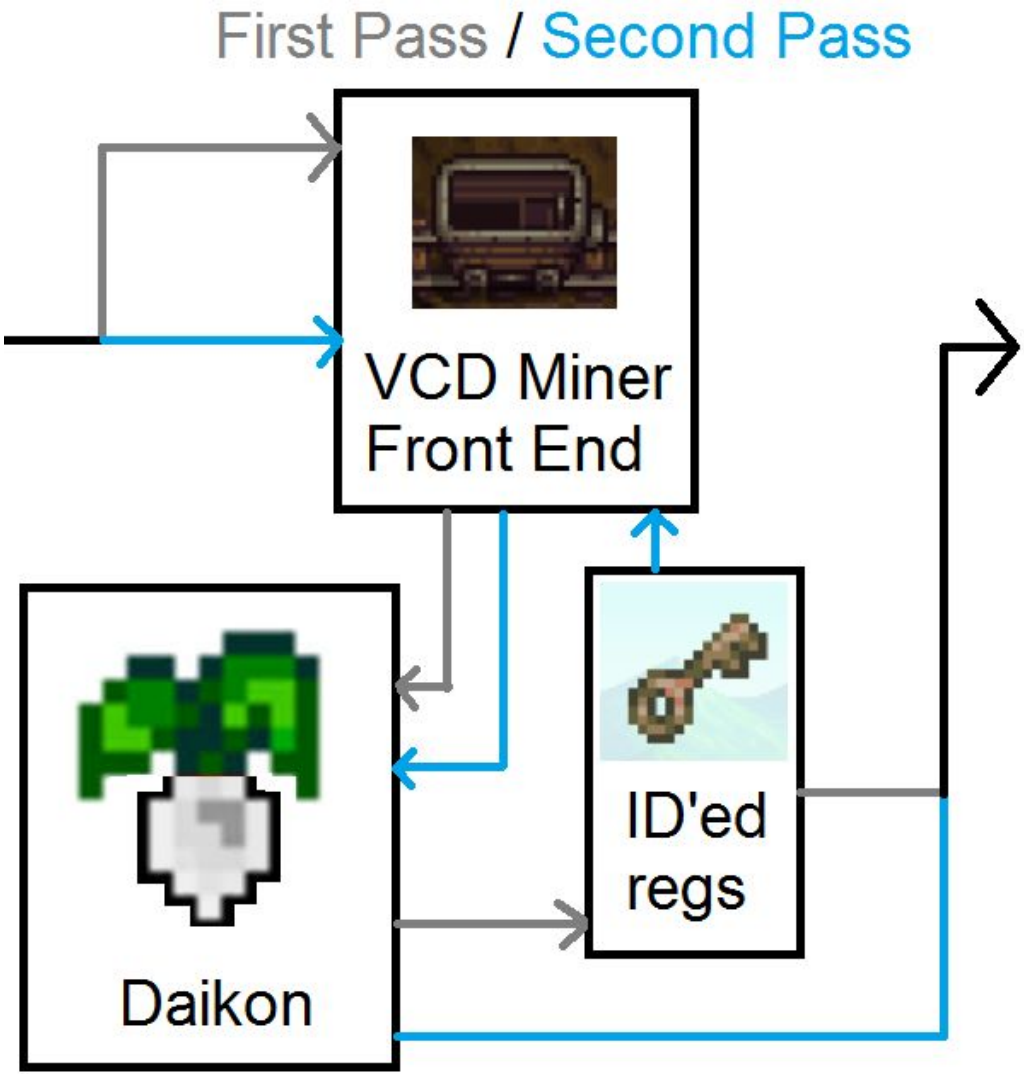
More than traces!

1. Specify Source
2. Generate Trace and IFT
3. Look at relevant regs

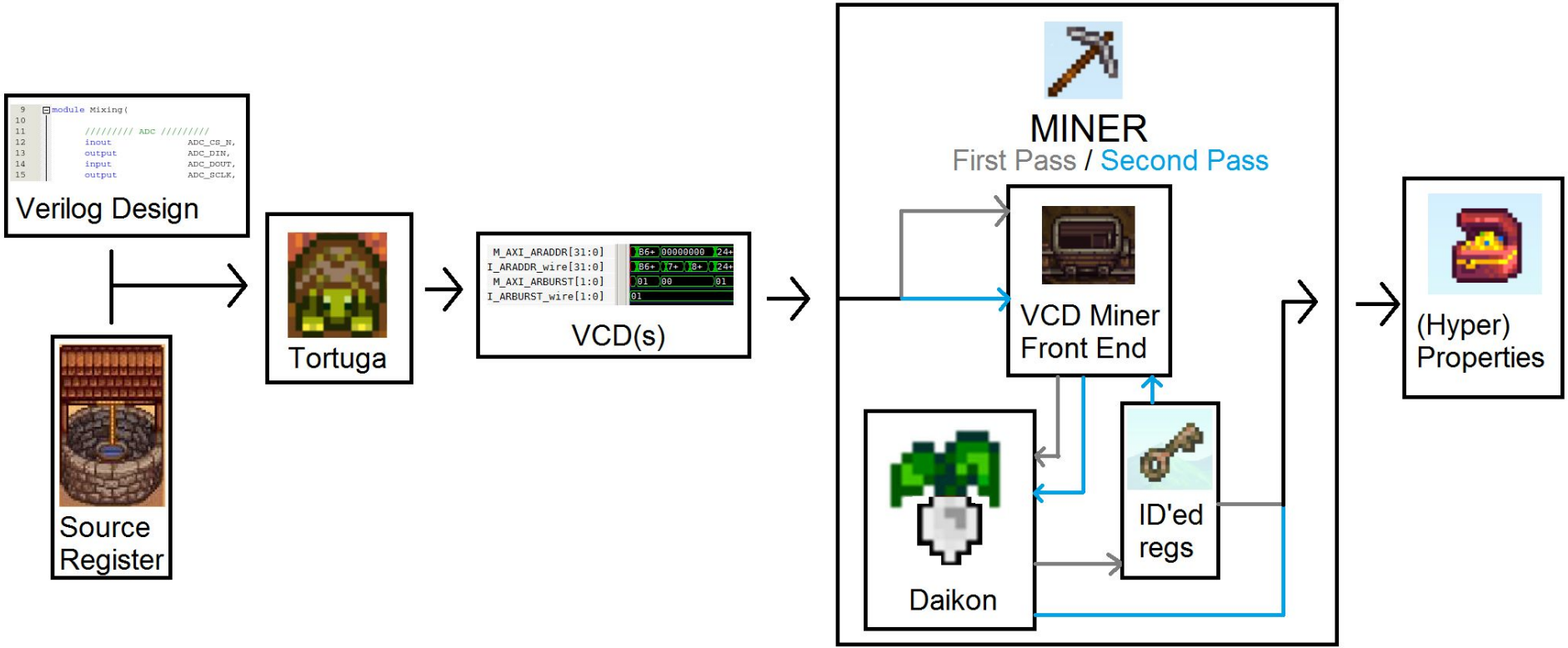


Miner Detail

1. Input Traces
2. Run Miner
3. Get Output
4. Flag interesting shadow_*
   a. shadow_* is IFT state
5. (Re-)Run Miner
6. Output Information Flow
   a. “Always, never, maybe”



Research Technique Sketch



Mining in Practice

- Test using *write-address* register
  - Always sink 003 regs
  - Never sink 189 regs
  - Conditional sink 037 regs
- Secondary mining passes can determine conditions under which the 37 conditional sinks are affected by the source register

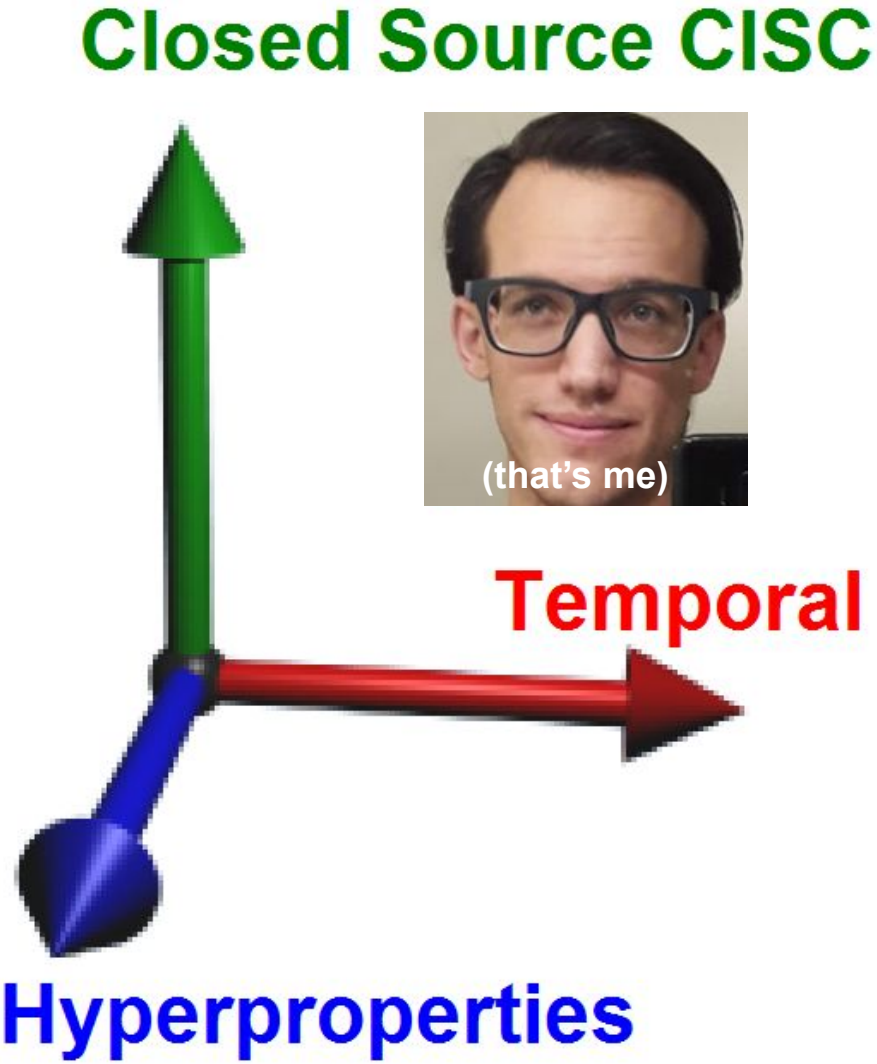
Isadora: Mining Hyperproperties (Current work)

Hyperproperties that model secure behavior of designs be discovered using specification mining along with Information Flow Tracking (IFT).

Mining Behavior

My research shows the technique of **specification mining** can find:

- **Temporal** properties, such as correct initialization
- **Closed source CISC** architecture properties, such those over x86-64
- ***Hyperproperties**, properties over multiple traces of execution*



“Who ya gonna call?”

Cybersecurity for the Spectre Era

Any Questions?