

Broad-Based Side-Channel Defenses for Modern Processor Architectures

Ashay Rane

Advisors: Calvin Lin and Mohit Tiwari

The University of Texas at Austin

theguardian

Meltdown and Spectre: 'worst ever' CPU bugs affect virtually all computers

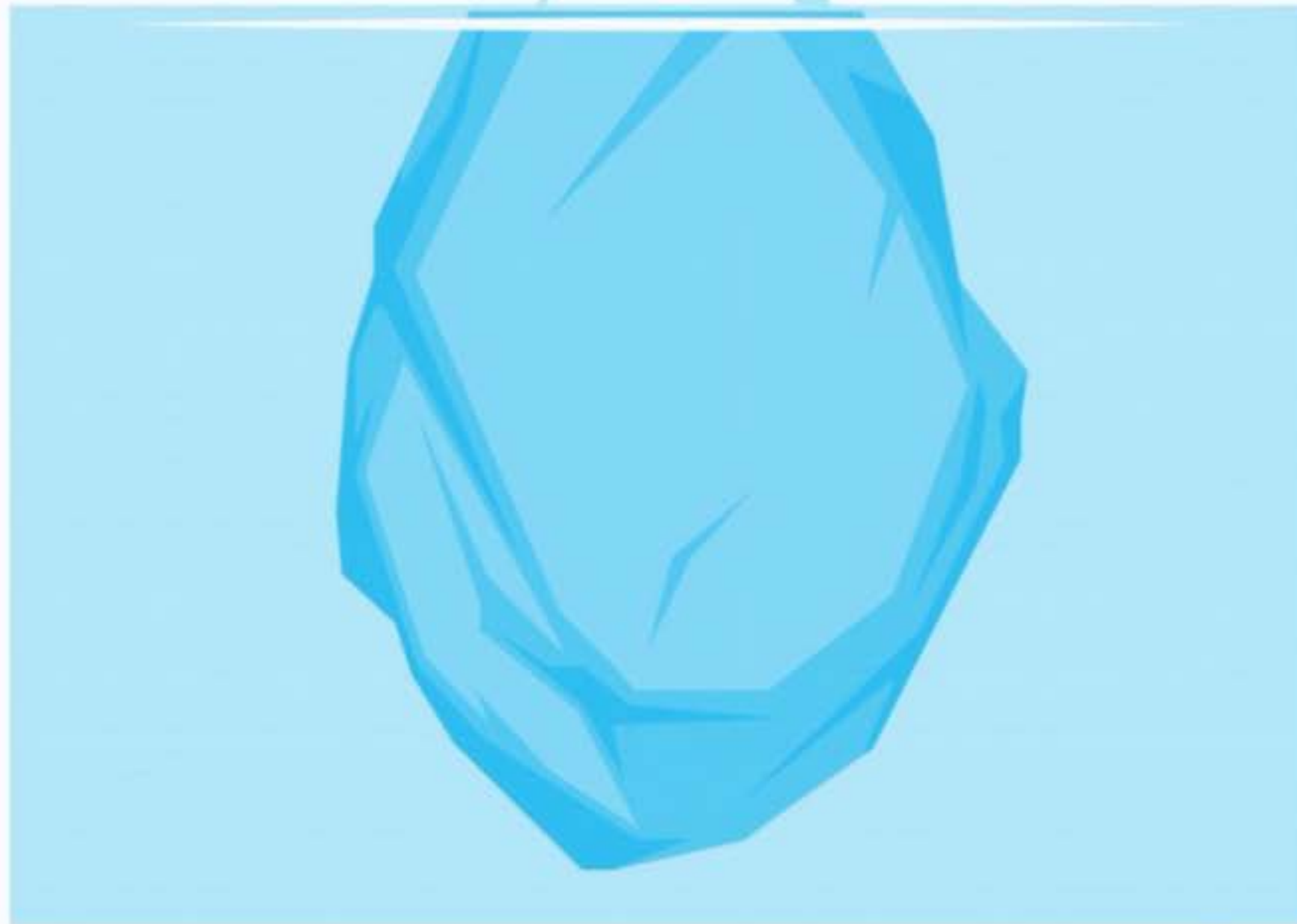
The Washington Post

Huge security flaws revealed – and tech companies can barely keep up

**BUSINESS
INSIDER**

chip maker faces 35 lawsuits over the attacks

Spectre,
Meltdown, ...



Spectre,
Meltdown, ...

A diagram of an iceberg floating in water. The water is represented by a light blue rectangular area. The iceberg is a darker blue, jagged shape. The top part of the iceberg is above the water line, and the much larger bottom part is submerged. The text 'Side Channels' is written in the center of the submerged part, and 'Spectre, Meltdown, ...' is written above the water line.

Side
Channels

Private Information in Various Applications



Cryptography



Web Browsers



Machine Learning

Private Information in Various Applications



Cryptography



Web Browsers



Machine Learning

We want to prevent leakage of private information

Techniques to Maintain Privacy of Information

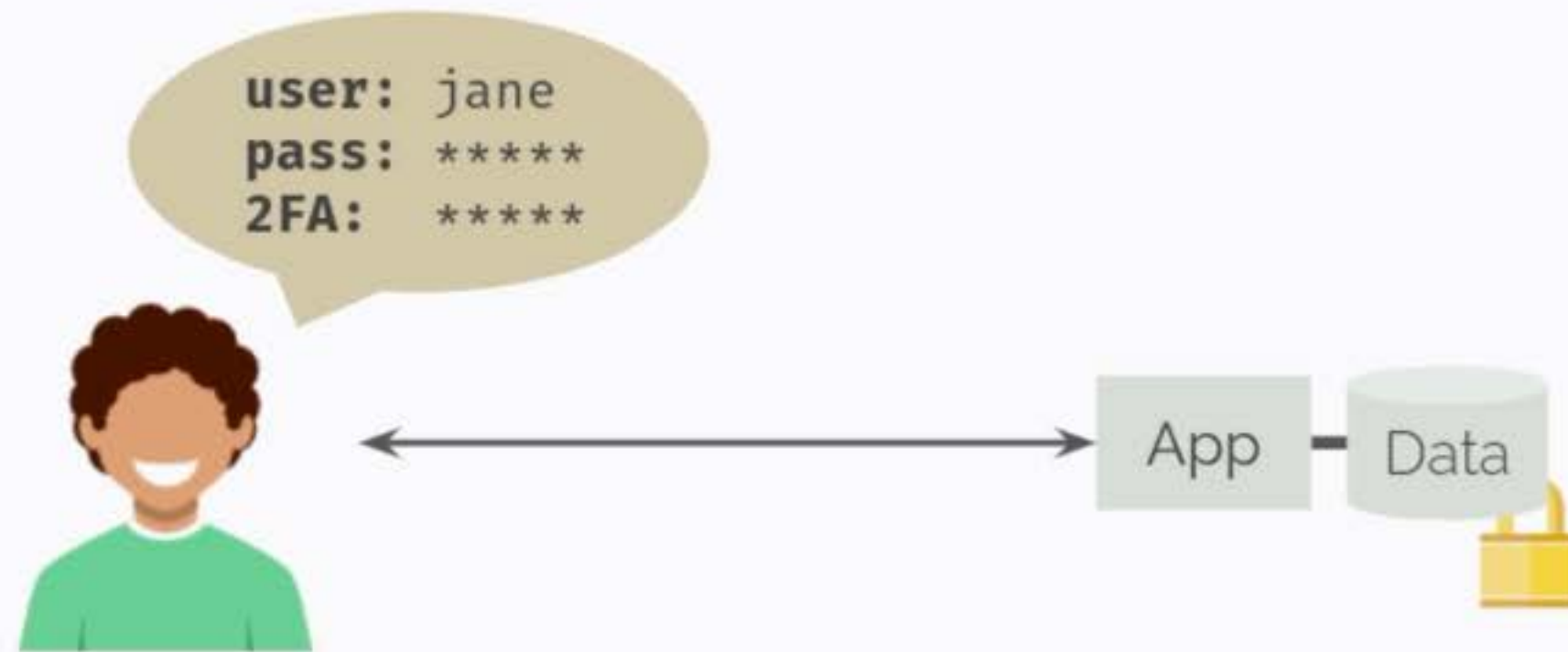


Techniques to Maintain Privacy of Information



Encryption

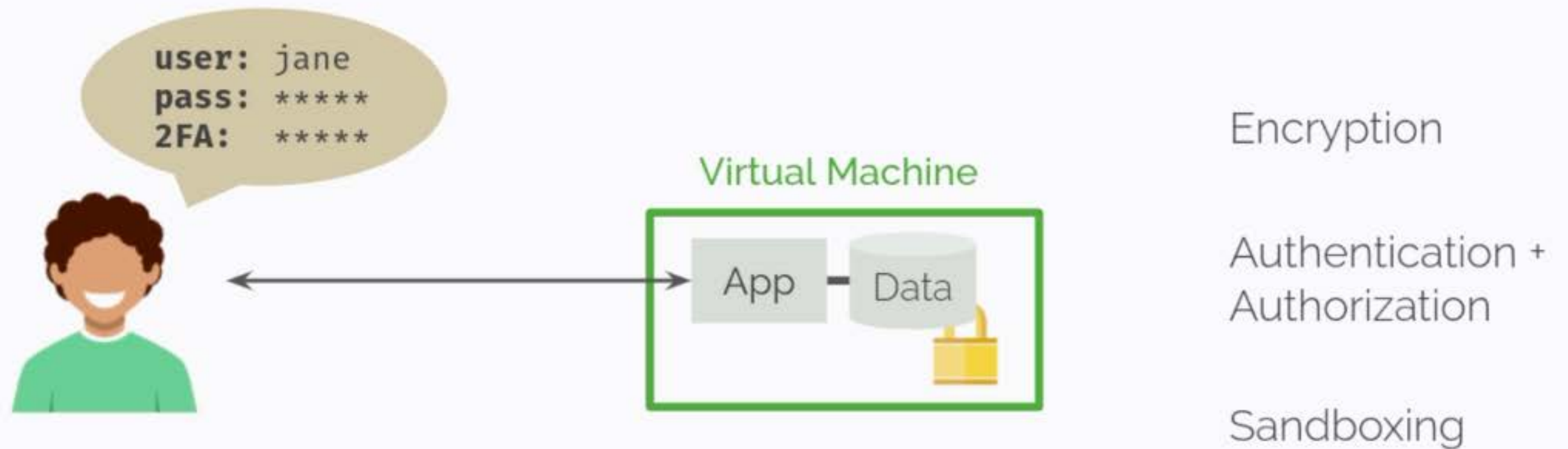
Techniques to Maintain Privacy of Information



Encryption

Authentication +
Authorization

Techniques to Maintain Privacy of Information





Virtual Machine



Side Channels

Encryption

Authentication +
Authorization

Sandboxing

Example Scenario

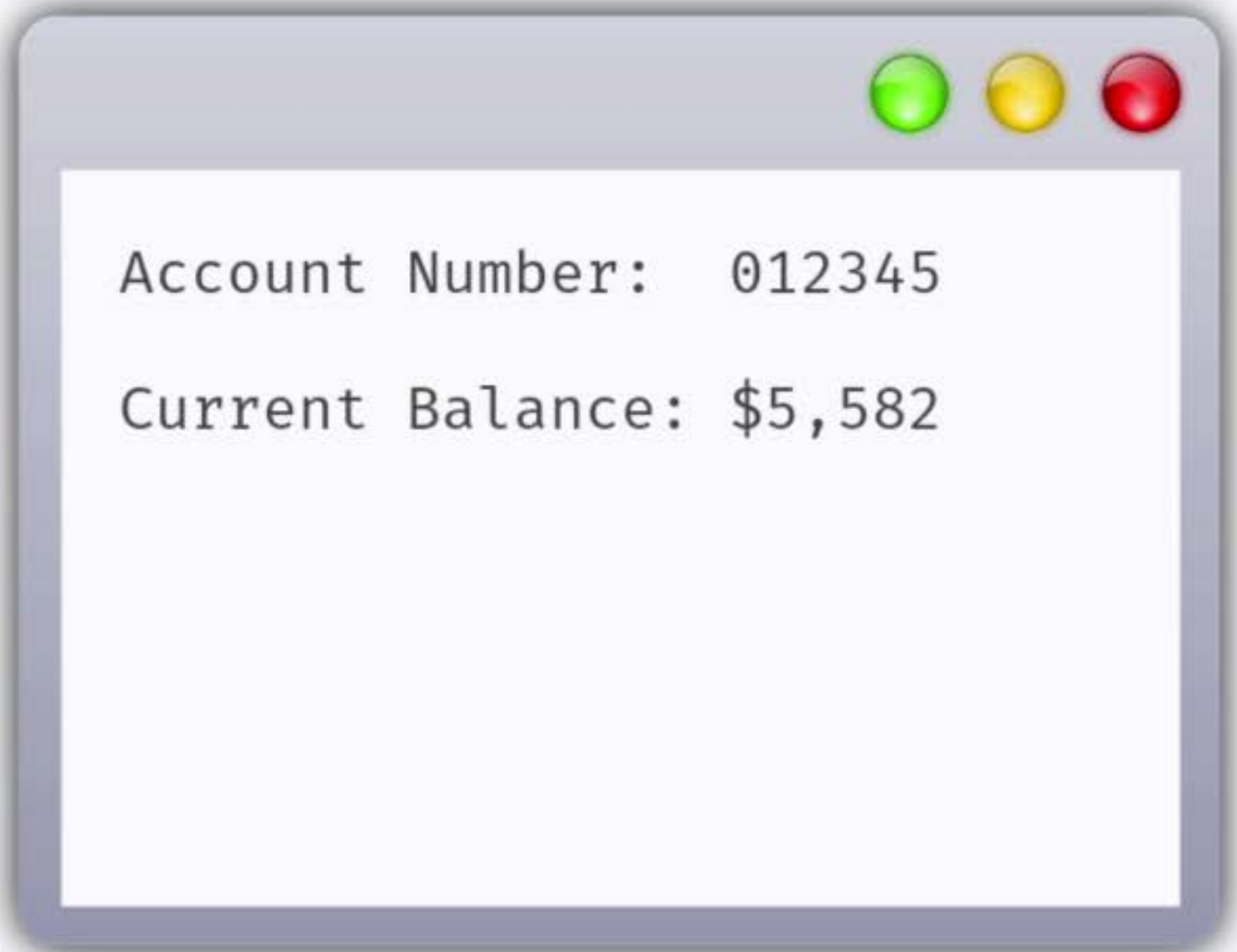


bank-stmt.pdf

Example Scenario



bank-stmt.pdf



Account Number: 012345

Current Balance: \$5,582

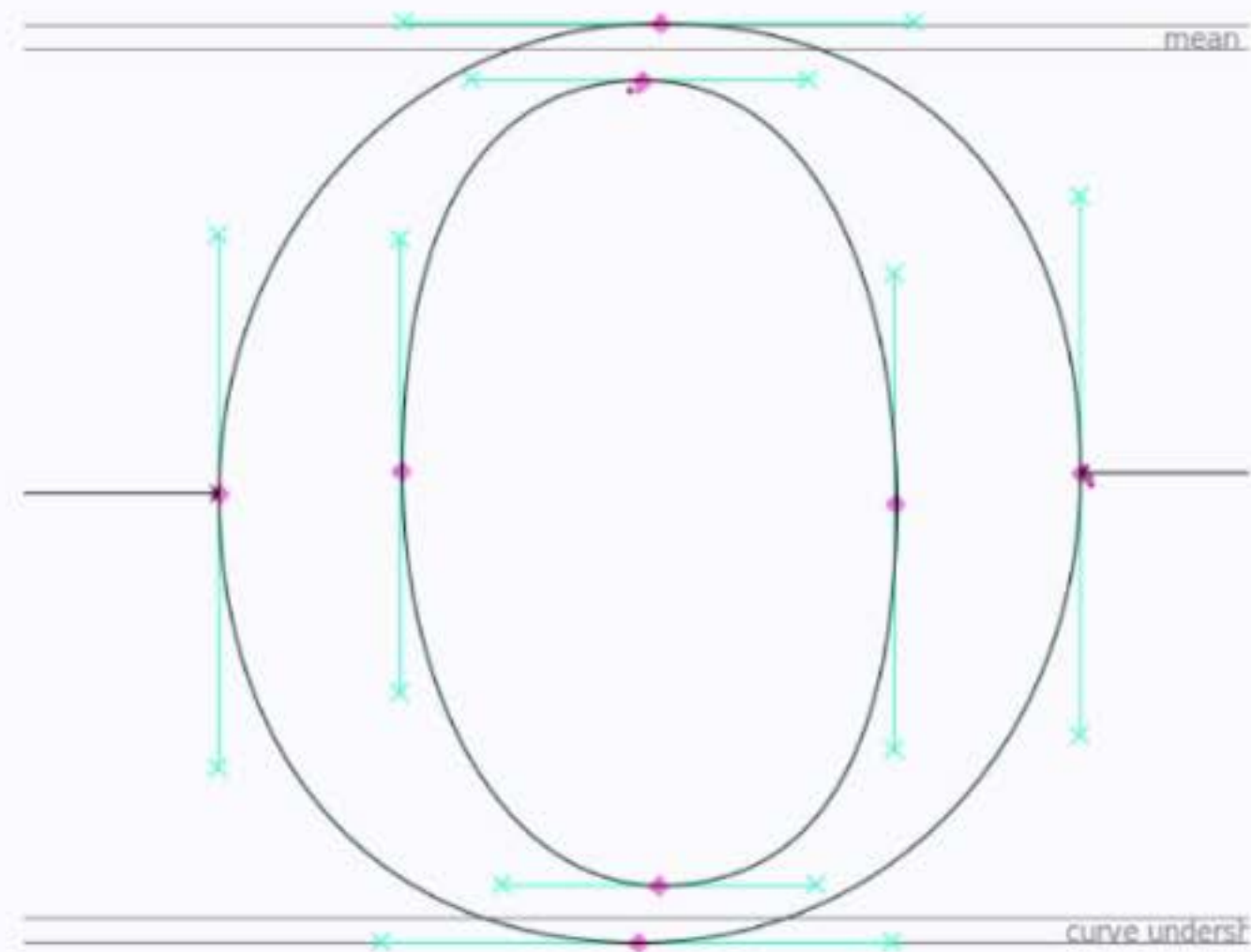
Rendering Characters Using Lines and Curves

O

Rendering Characters Using Lines and Curves

O

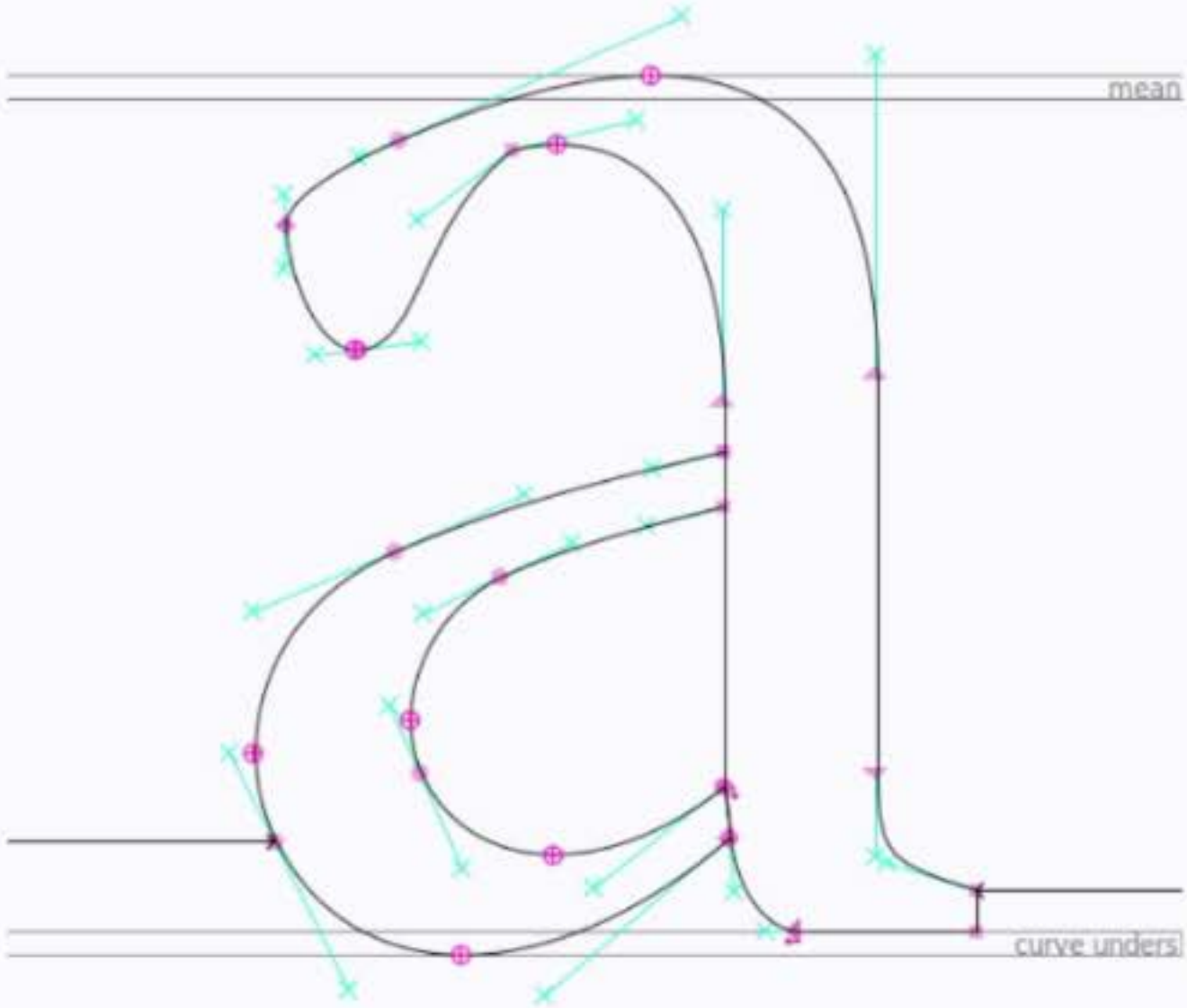
Rendered using
lines and curves



Rendering Characters Using Lines and Curves

a

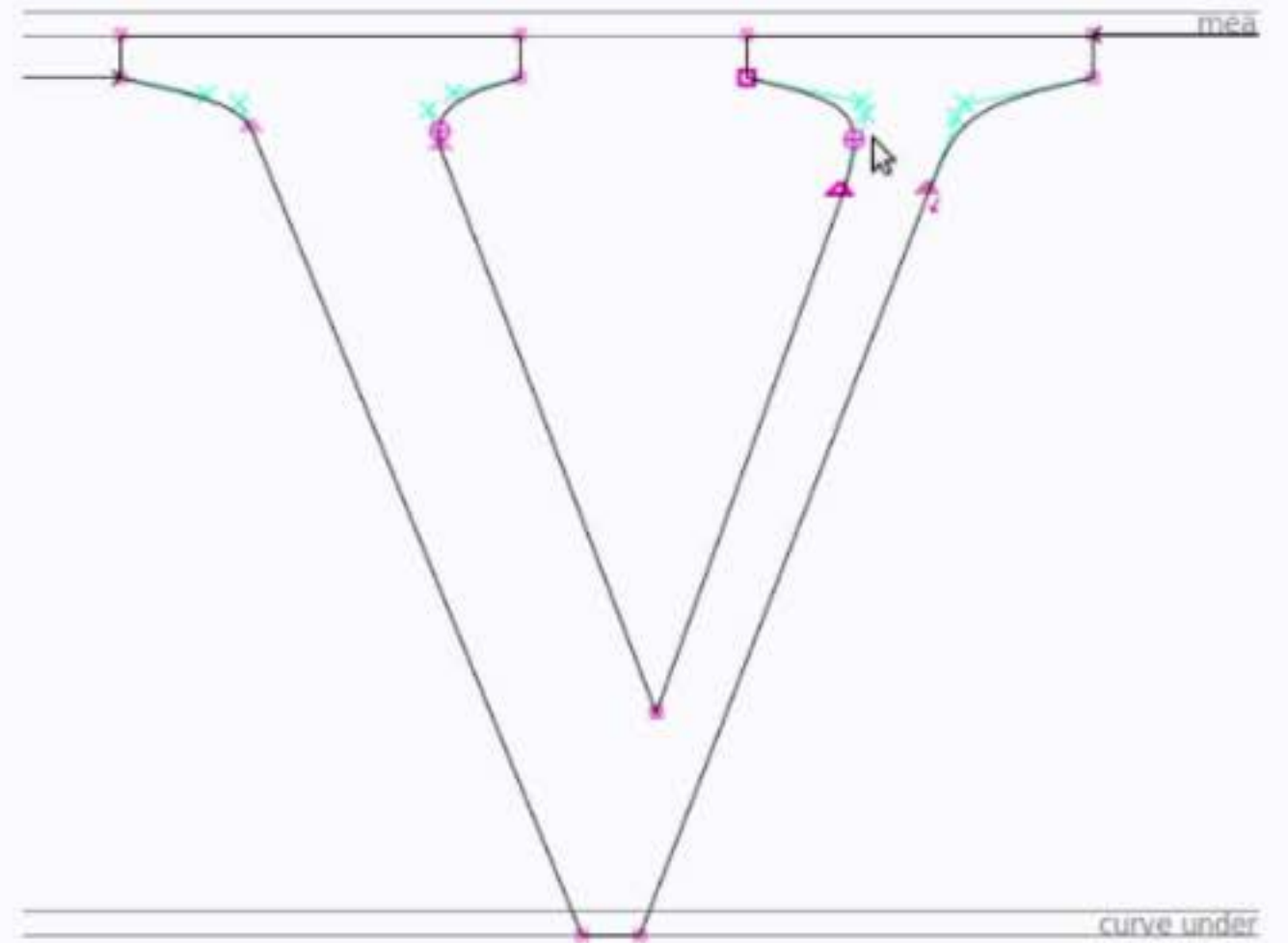
Rendered using
lines and curves



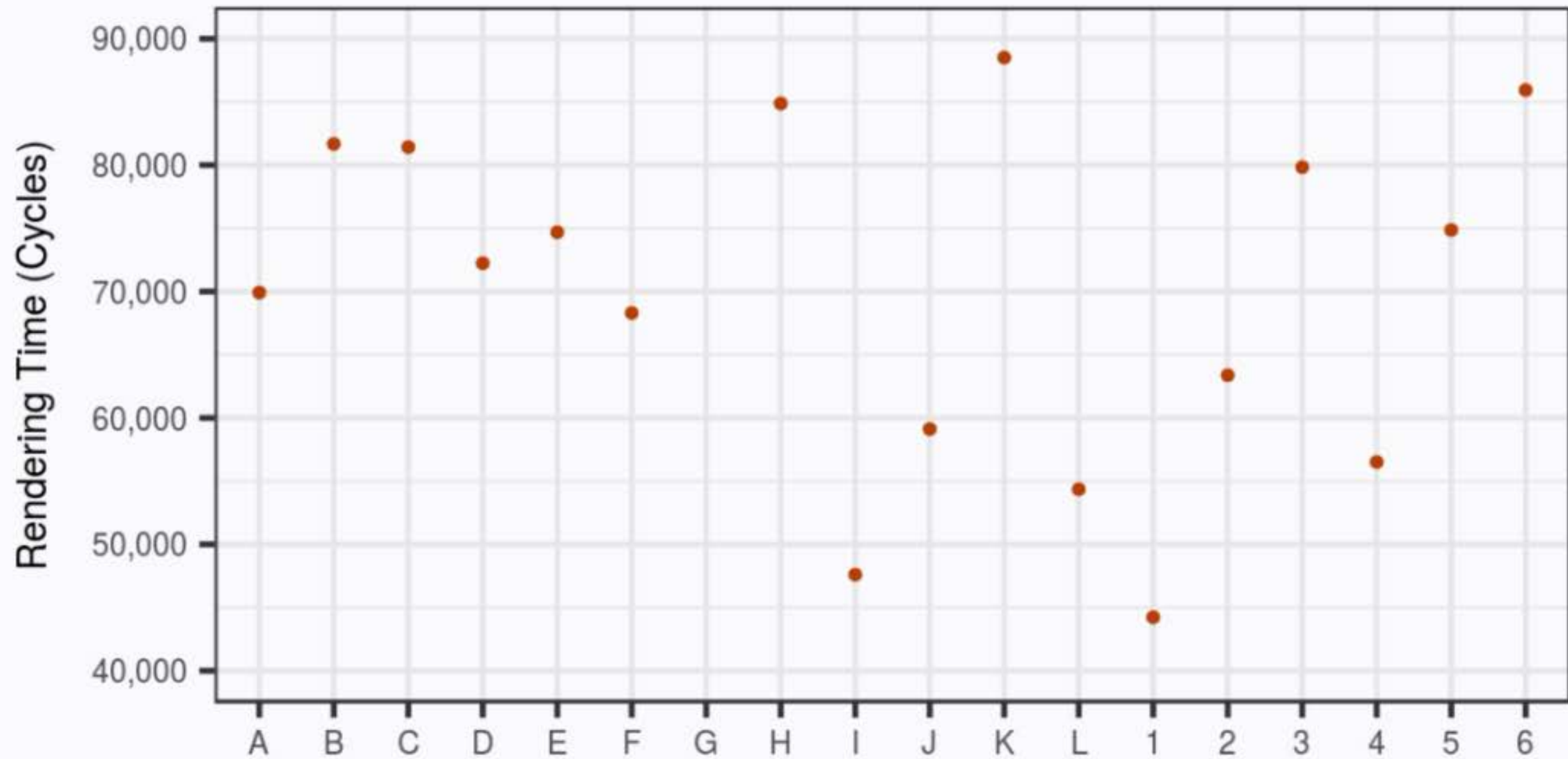
Rendering Characters Using Lines and Curves



Rendered using
lines and curves



Execution Time for Rendering Characters



Proof of Concept on a Font Renderer

Original Text:

hello world my social security number is 123 45 6789

Proof of Concept on a Font Renderer

Original Text:

hello world my social security number is 123 45 6789

Recovered Text:

Proof of Concept on a Font Renderer

Original Text:

hello world my social security number is 123 45 6789

Recovered Text:

wello would my socqal secuqtk kumweu it 1r3 45 6789

Proof of Concept on a Font Renderer

Original Text:

hello world my social security number is 123 45 6789

Recovered Text:

wello would my socqal secuqt kumweu it 1r3 45 6789

41 out of **52** characters correctly guessed

Attacker can measure **execution time**
to steal sensitive document contents

Real-World Attack on FreeType Renderer

[Xu et al., Oakland-2015]

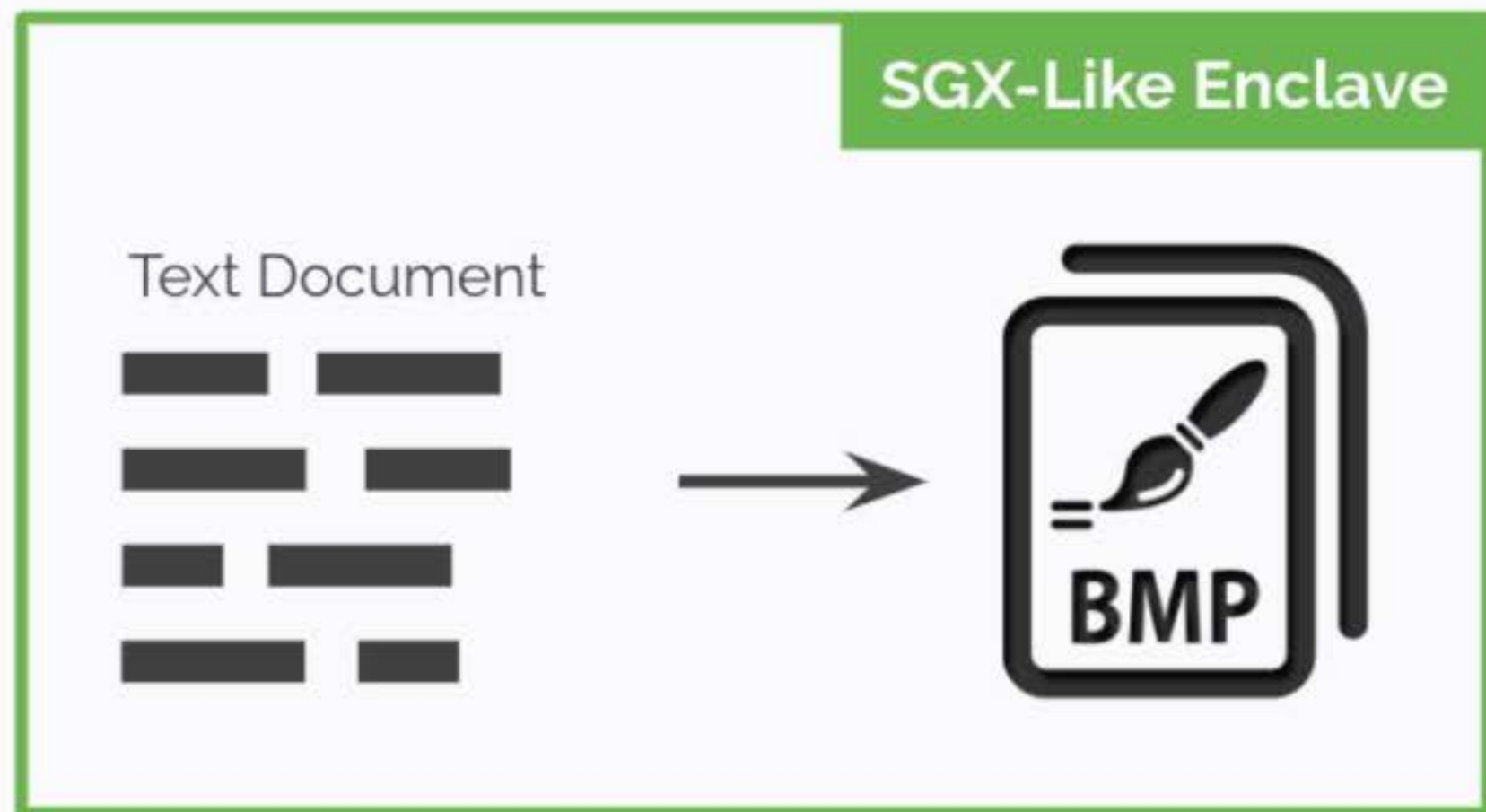
Application converts document into an image

Text Document



Real-World Attack on FreeType Renderer

[Xu et al., Oakland-2015]

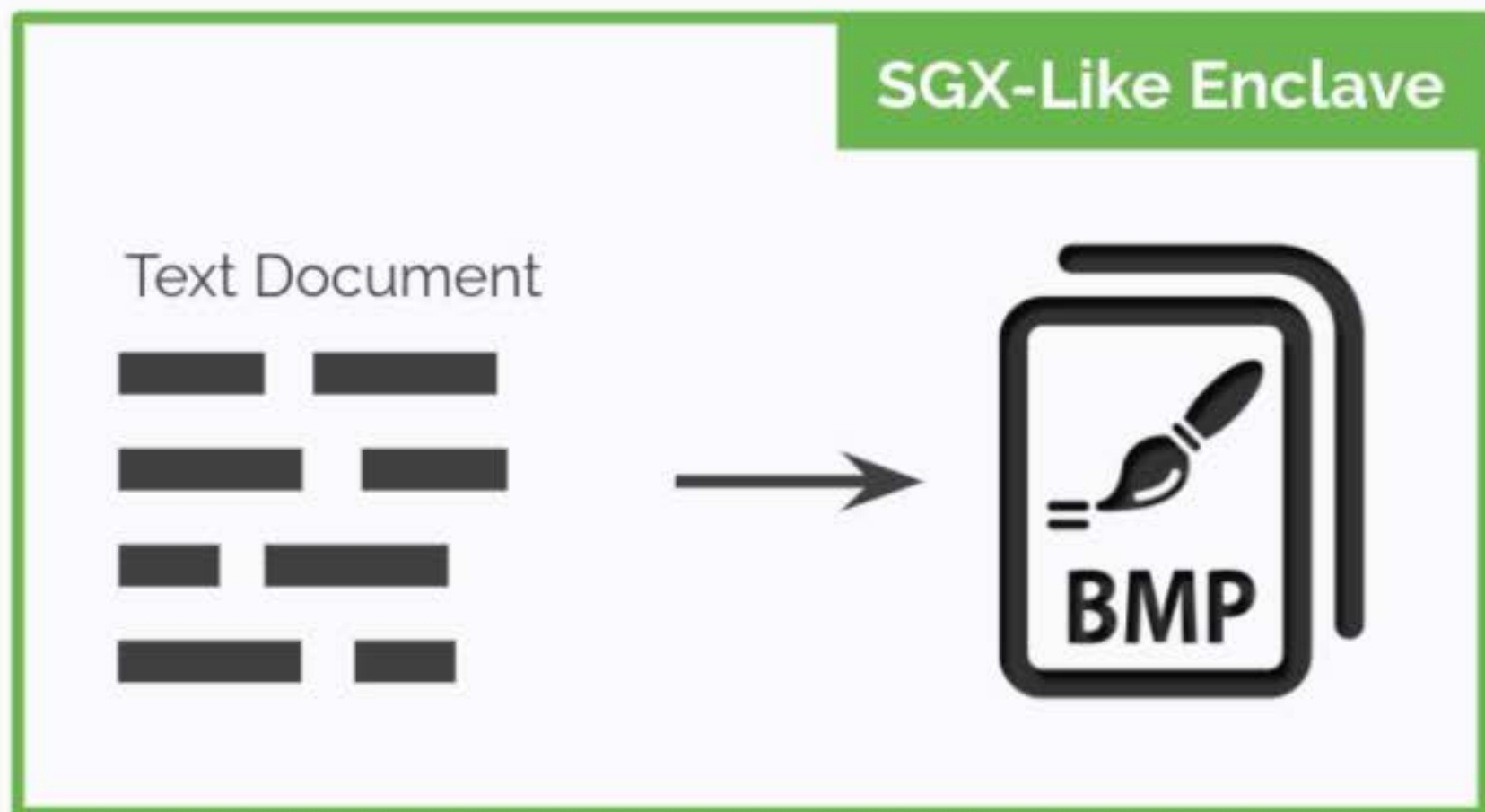


Application converts document into an image

Application runs inside an SGX-like enclave

Real-World Attack on FreeType Renderer

[Xu et al., Oakland-2015]



Malicious Operating System

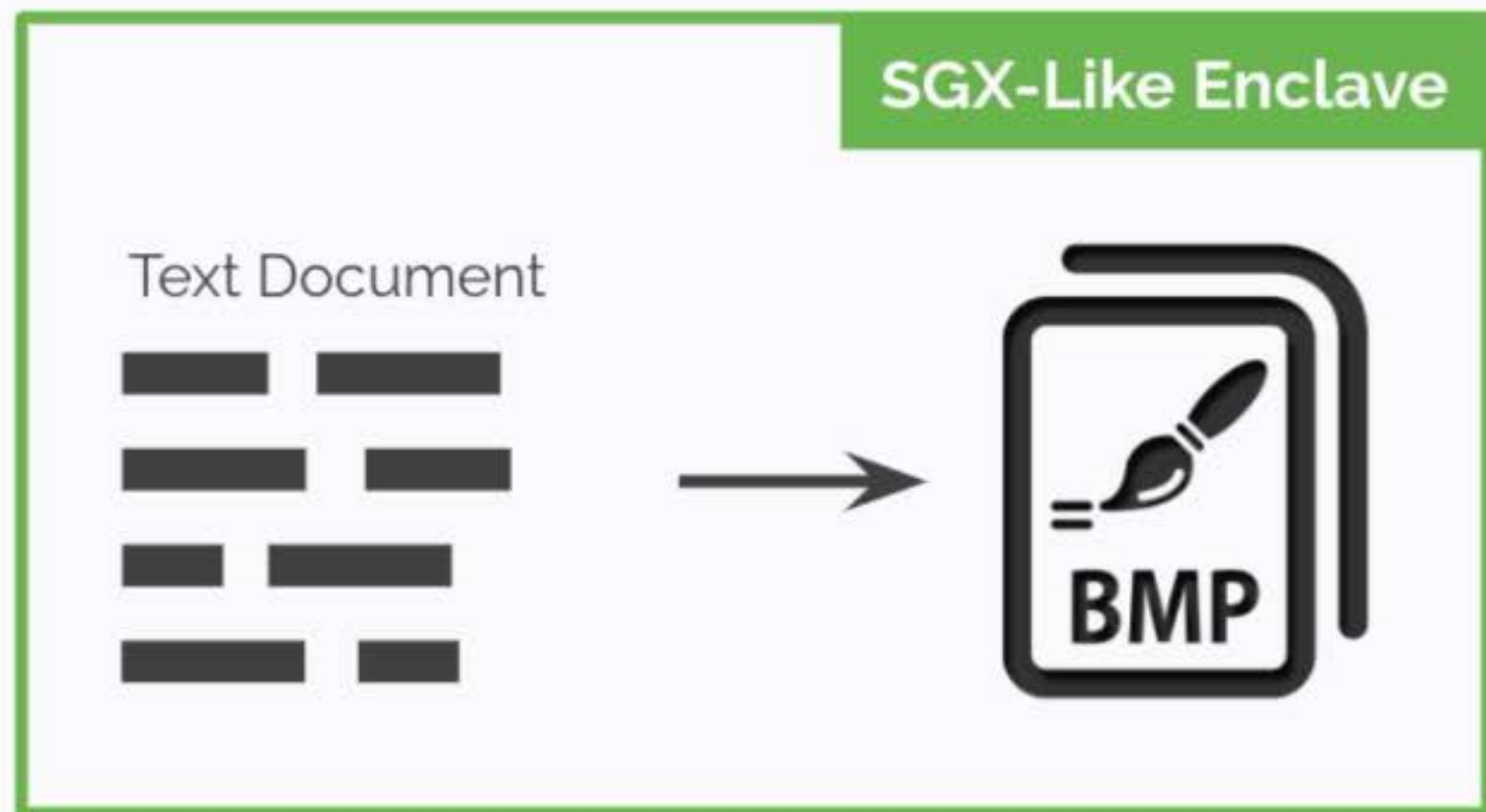
Application converts document into an image

Application runs inside an SGX-like enclave

Malicious OS observes page faults

Real-World Attack on FreeType Renderer

[Xu et al., Oakland-2015]



Malicious Operating System

Application converts document into an image

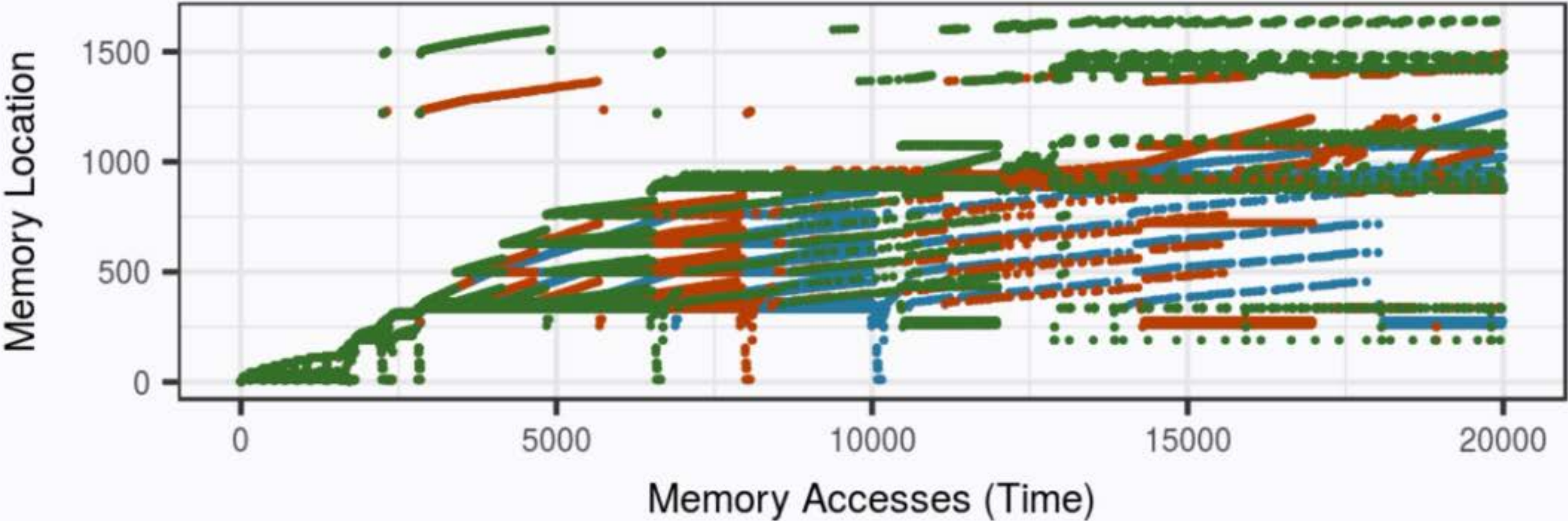
Application runs inside an SGX-like enclave

Malicious OS observes page faults

100% text recovered by OS

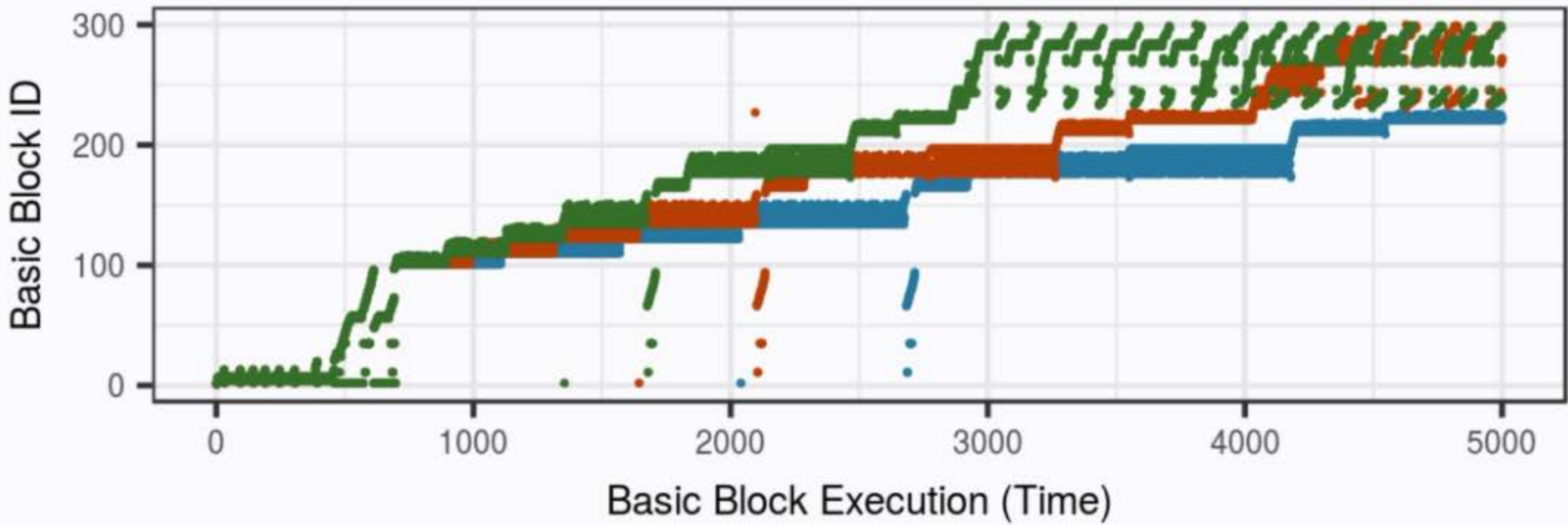
Memory Address Trace While Rendering Characters

Rendered Character: ■ X ■ Y ■ Z



Instruction Trace While Rendering Characters

Rendered Character: ■ X ■ Y ■ Z



Information May Leak Through **Many Side Channels**

Application Program

e.g. execution time

Instruction Set Arch

e.g. page faults

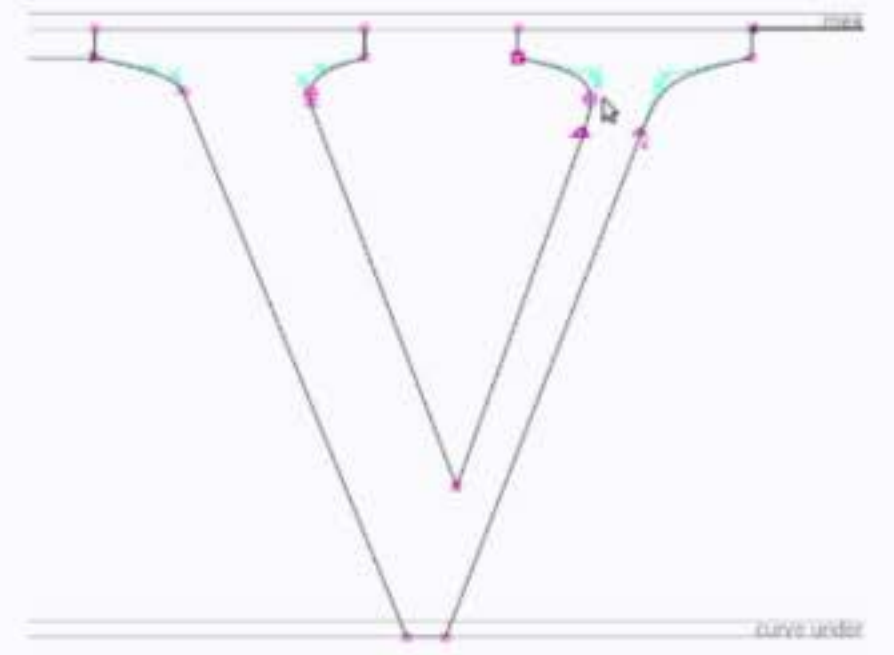
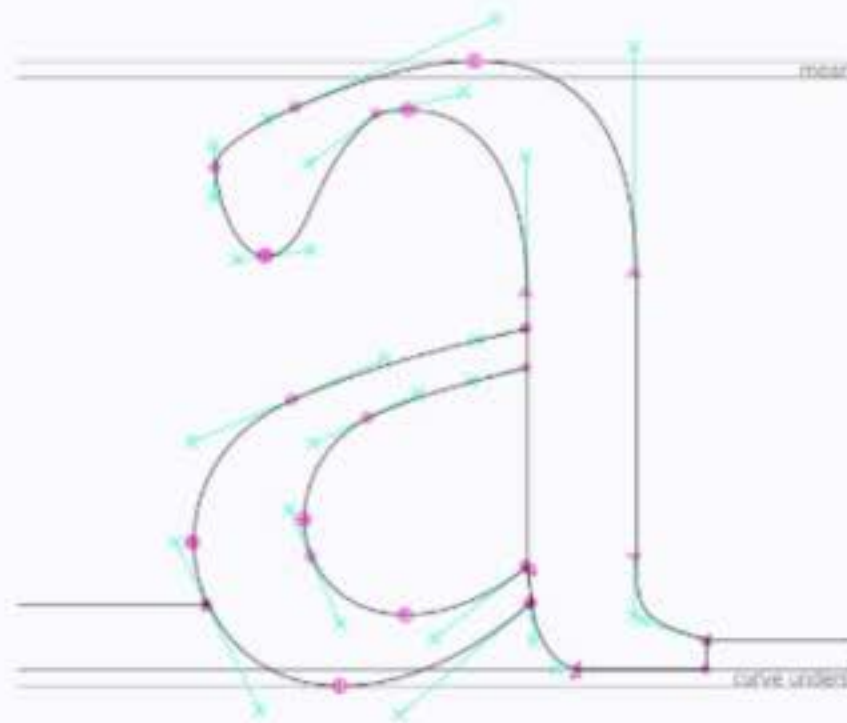
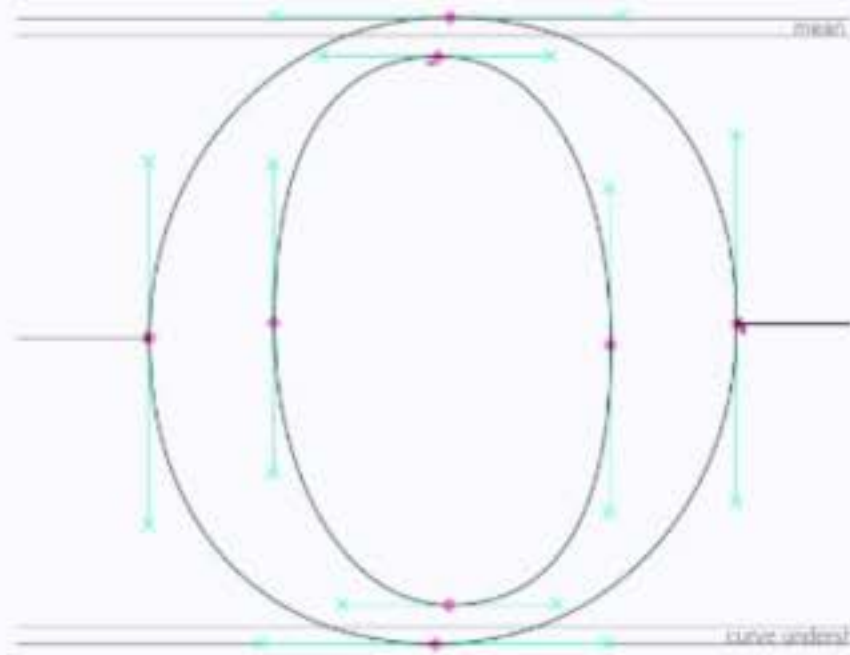
Microarchitecture

e.g. branch predictor, cache, PC, DRAM addresses

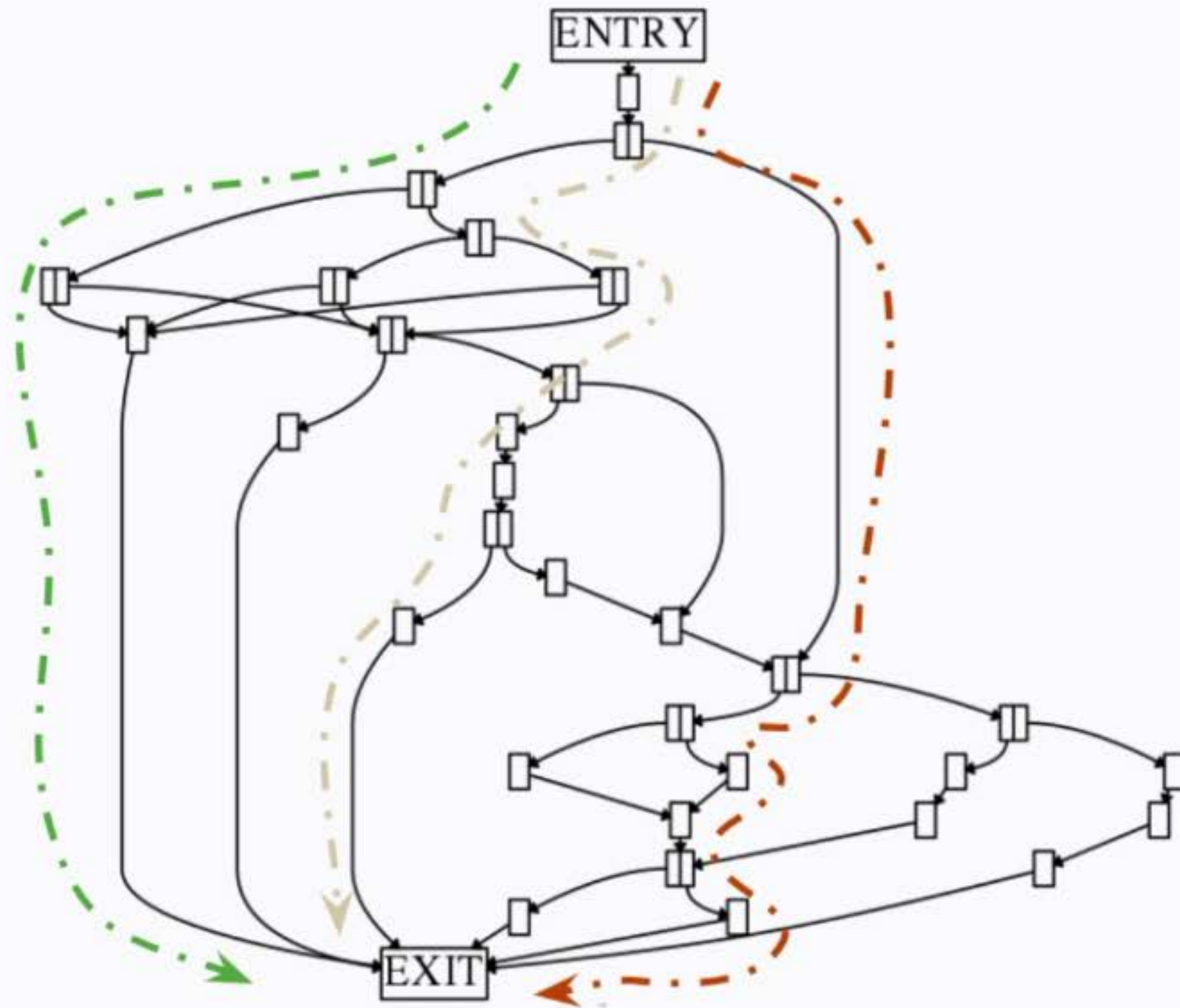
Physical Hardware

e.g. power consumption, EM radiation

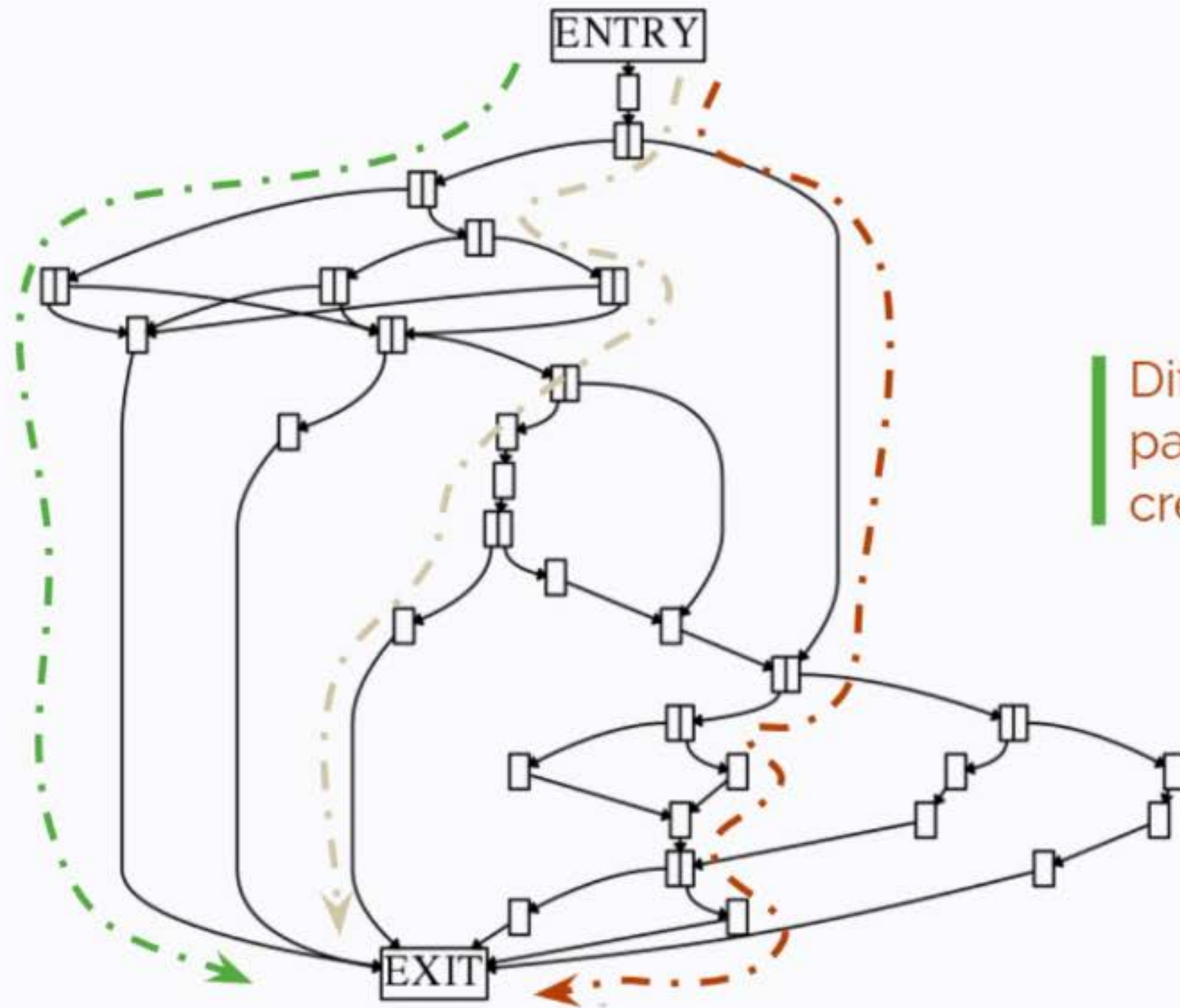
What is the **Core Vulnerability**?



Control Flow Graph



Control Flow Graph



Different input values execute different paths, thus causing variations, which create side channels.

Prior Side Channel Defenses

Focus on symptoms, thus providing point solutions

Application Program

e.g. execution time

[ISCA12], [ASPLOS15], [CHES00],
[ICISC03], [ICISC05], [ICISC10]

Instruction Set Arch

e.g. page faults

Microarchitecture

e.g. branch predictor, cache, PC, DRAM addresses

Physical Hardware

e.g. power consumption, EM radiation

Prior Side Channel Defenses

Focus on symptoms, thus providing point solutions

Application Program

e.g. execution time

Instruction Set Arch

e.g. page faults

Microarchitecture

e.g. branch predictor, **cache**, PC, DRAM addresses

[ISCA07], [ISCA08], [HPCA09], [INDSS15], [CCS13a]

Physical Hardware

e.g. power consumption, EM radiation

Prior Side Channel Defenses

Focus on symptoms, thus providing point solutions

Application Program

e.g. execution time

Instruction Set Arch

e.g. page faults

Microarchitecture

e.g. branch predictor, cache, PC, DRAM addresses

[ISCA13], [CCS13b], [CCS13c], [ASIACRYPT11]

Physical Hardware

e.g. power consumption, EM radiation

Drawbacks of Point Solutions

Drawbacks of Point Solutions

1. **Focused on the symptoms not the root cause**
Requires a completely redesigned solution for every side channel

Drawbacks of Point Solutions

1. **Focused on the symptoms not the root cause**
Requires a completely redesigned solution for every side channel
2. **Difficult to ensure end-to-end or comprehensive security**
One point solution may negate the security guarantees of another

Drawbacks of Point Solutions

1. **Focused on the symptoms not the root cause**
Requires a completely redesigned solution for every side channel
2. **Difficult to ensure end-to-end or comprehensive security**
One point solution may negate the security guarantees of another
3. **Require disabling of optimizations in the compiler and thus, require redesigning the processor for each side channel**
Since optimizations may break security guarantees

Drawbacks of Point Solutions

GhostRider [ASPLOS-15]

Original Program

```
if (secret == 0) {  
  x <- load ptr_1  
  y <- load ptr_2  
} else {  
  z <- load ptr_3  
}
```

Drawbacks of Point Solutions

GhostRider [ASPLOS-15]

Original Program

```
if (secret == 0) {  
  x <- load ptr_1  
  y <- load ptr_2  
} else {  
  z <- load ptr_3  
}
```

Ensure equal
load instructions
of each path



Transformed Code

```
if (secret == 0) {  
  x <- load ptr_1  
  y <- load ptr_2  
} else {  
  z <- load ptr_3  
  d <- load dummy  
}
```

Drawbacks of Point Solutions

GhostRider [ASPLOS-15]

Original Program

```
if (secret == 0) {  
  x <- load ptr_1  
  y <- load ptr_2  
} else {  
  z <- load ptr_3  
}
```

Ensure equal
load instructions
of each path



Dead Code
Elimination

Transformed Code

```
if (secret == 0) {  
  x <- load ptr_1  
  y <- load ptr_2  
} else {  
  z <- load ptr_3  
d <- load dummy  
}
```

Drawbacks of Point Solutions

GhostRider [ASPLOS-15]

Optimizing compilers **may break the security guarantee**

Original Program

```
if (secret == 0) {  
  x <- load ptr_1  
  y <- load ptr_2  
} else {  
  z <- load ptr_3  
}
```

Ensure equal
load instructions
of each path



Dead Code
Elimination

Transformed Code

```
if (secret == 0) {  
  x <- load ptr_1  
  y <- load ptr_2  
} else {  
  z <- load ptr_3  
d <- load dummy  
}
```

Drawbacks of Point Solutions

GhostRider [ASPLOS-15]

Original Program

```
if (secret == 0) {  
  x <- load ptr_1  
  y <- load ptr_2  
} else {  
  z <- load ptr_3  
}
```

Ensure equal
load instructions
of each path



Transformed Code

```
if (secret == 0) {  
  x <- load ptr_1 Cache hit  
  y <- load ptr_2 Cache hit  
} else {  
  z <- load ptr_3 Cache hit  
  d <- load dummy  
}
```

Drawbacks of Point Solutions

GhostRider [ASPLOS-15]

Original Program

```
if (secret == 0) {  
  x <- load ptr_1  
  y <- load ptr_2  
} else {  
  z <- load ptr_3  
}
```

Ensure equal
load instructions
of each path



Transformed Code

```
if (secret == 0) {  
  x <- load ptr_1 Cache hit  
  y <- load ptr_2 Cache hit  
} else {  
  z <- load ptr_3 Cache hit  
  d <- load dummy Miss !  
}
```

Drawbacks of Point Solutions

GhostRider [ASPLOS-15]

Caches and prefetchers **may break the security guarantee**

Original Program

```
if (secret == 0) {  
  x <- load ptr_1  
  y <- load ptr_2  
} else {  
  z <- load ptr_3  
}
```

Ensure equal
load instructions
of each path

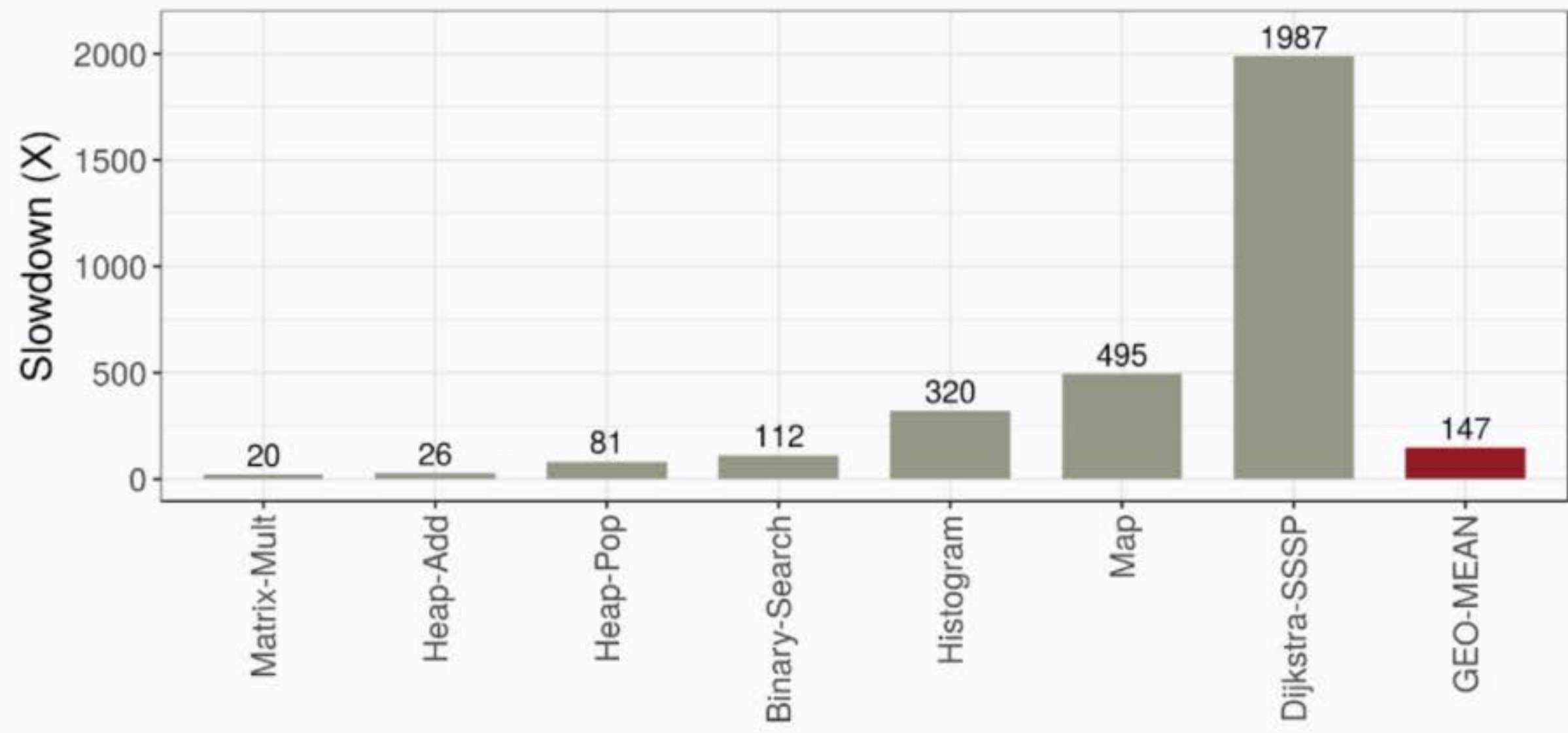


Transformed Code

```
if (secret == 0) {  
  x <- load ptr_1 Cache hit  
  y <- load ptr_2 Cache hit  
} else {  
  z <- load ptr_3 Cache hit  
  d <- load dummy Miss !  
}
```

Performance Impact of Using Point Solution

Disabled optimizations result in significant performance overhead



Prior Side Channel Defenses

Prior Side Channel Defenses

Are **point solutions**, since they **focus on symptoms** and not the root cause, and they may not compose well.

Prior Side Channel Defenses

Are **point solutions**, since they **focus on symptoms** and not the root cause, and they may not compose well.

Many require **redesigned hardware**, since the solution is forced to disable optimizations in compiler and microarch.

Prior Side Channel Defenses

Are **point solutions**, since they **focus on symptoms** and not the root cause, and they may not compose well.

Many require **redesigned hardware**, since the solution is forced to disable optimizations in compiler and microarch.

Many are **inflexible** because they **cannot be tailored to the program** or to portions of the program.

My Solutions

Closes a Broad Class
of Side Channels

My Solutions

Closes a Broad Class
of Side Channels

Executes on Modern
Microprocessors

My Solutions

Closes a Broad Class
of Side Channels

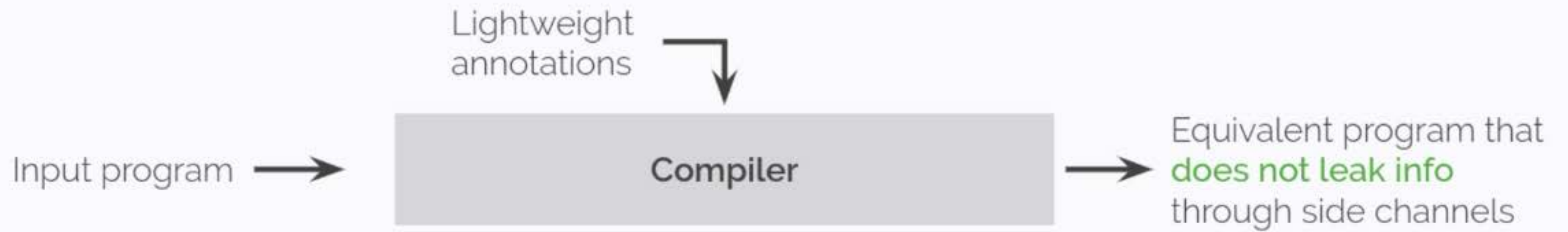
Executes on Modern
Microprocessors

Protects a Diverse
Set of Applications

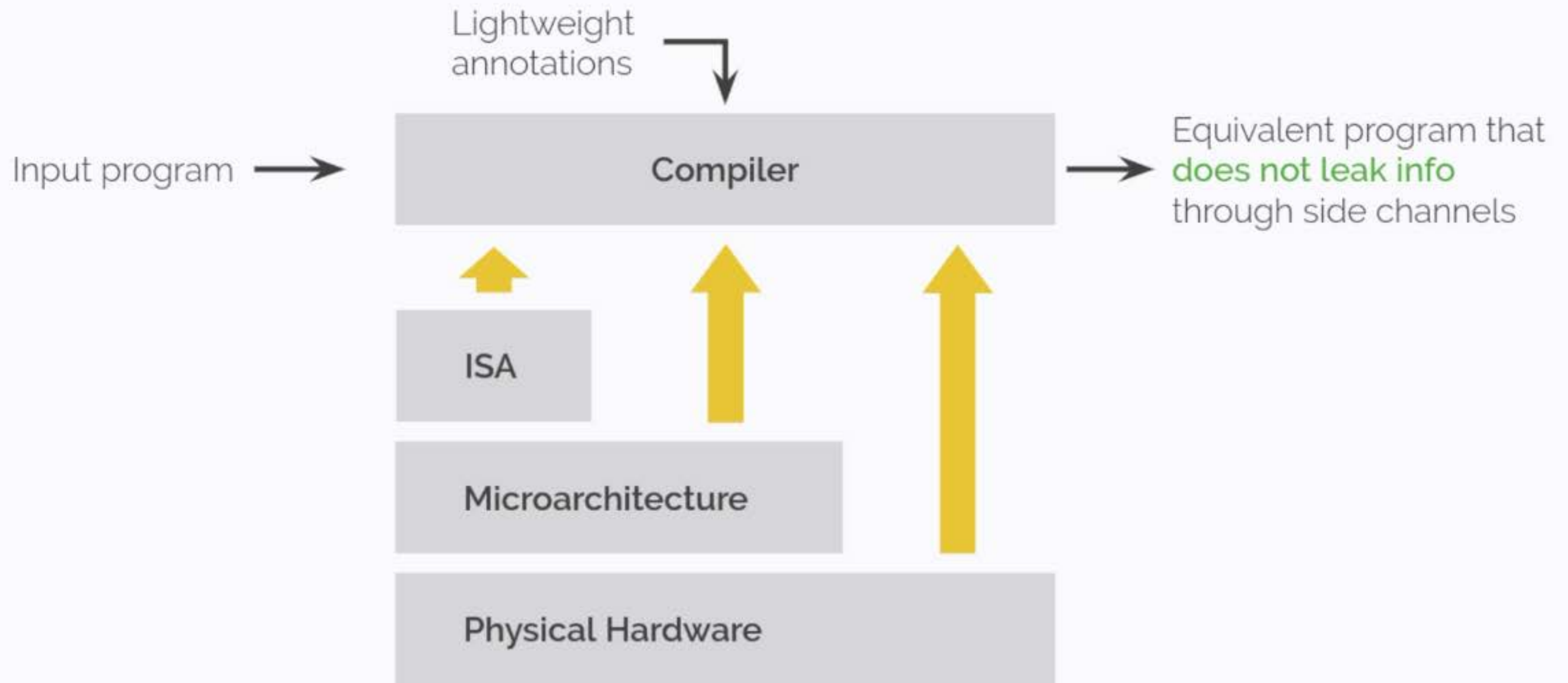
My Research **Contributions**



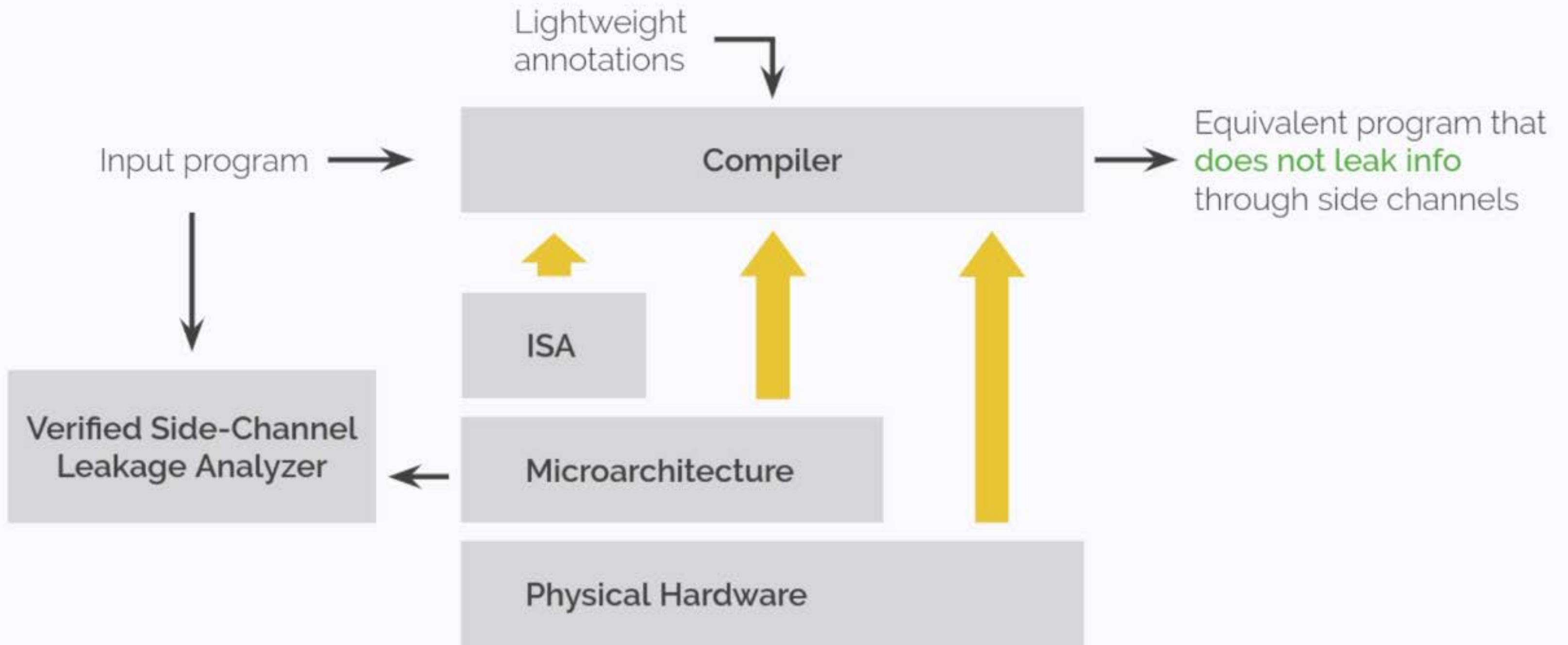
My Research **Contributions**



My Research **Contributions**



My Research **Contributions**



My Research **Contributions**

Raccoon

USENIX Security
Symposium 2015

Escort

USENIX Security
Symposium 2016

Vale

USENIX Security
Symposium 2017

Vantage

Work In Progress

My Research **Contributions**

Raccoon

USENIX Security
Symposium 2015

Escort

USENIX Security
Symposium 2016

Vale

USENIX Security
Symposium 2017

Vantage

Work In Progress

Compilers for closing all
digital side channels

My Research **Contributions**

Raccoon

USENIX Security
Symposium 2015

Compilers for closing all
digital side channels

Escort

USENIX Security
Symposium 2016

e.g. cache, address trace,
branch predictor, TLB, etc.

Vale

USENIX Security
Symposium 2017

Vantage

Work In Progress

My Research **Contributions**

Raccoon

USENIX Security
Symposium 2015

Escort

USENIX Security
Symposium 2016

Vale

USENIX Security
Symposium 2017

Verified side channel
leakage analyzer

Vantage

Work In Progress

My Research **Contributions**

Raccoon

USENIX Security
Symposium 2015

Escort

USENIX Security
Symposium 2016

Vale

USENIX Security
Symposium 2017

Vantage

Work In Progress

Compiler that mitigates **power side channel** attacks
in diverse programs and microarchitectures

Outline

- ▶ Our Solution's Design

Outline

- ▶ Our Solution's Design
- ▷ Core Principles that Enable Generalization

Outline

- ▶ Our Solution's Design
- ▷ Core Principles that Enable Generalization
- ▷ Performance Comparison

Outline

- ▶ Our Solution's Design
- ▷ Core Principles that Enable Generalization
- ▷ Performance Comparison
- ▷ Future Work

Key Insight Behind Our Solutions

Key Insight Behind Our Solutions

A broad range of side channels arise due to **variations in source-level behavior**.

Key Insight Behind Our Solutions

A broad range of side channels arise due to **variations in source-level behavior**.

- **Branch predictor side channel** is caused by **program path**
- **Memory trace channel** is caused by **pointer dereferences** and **program path**
- **Instruction count** is caused by **program path**

Key Insight Behind Our Solutions

Source-Level Behavior



causes

Different Side Channels

Key Insight Behind Our Solutions

Control Flow and Data Flow



causes

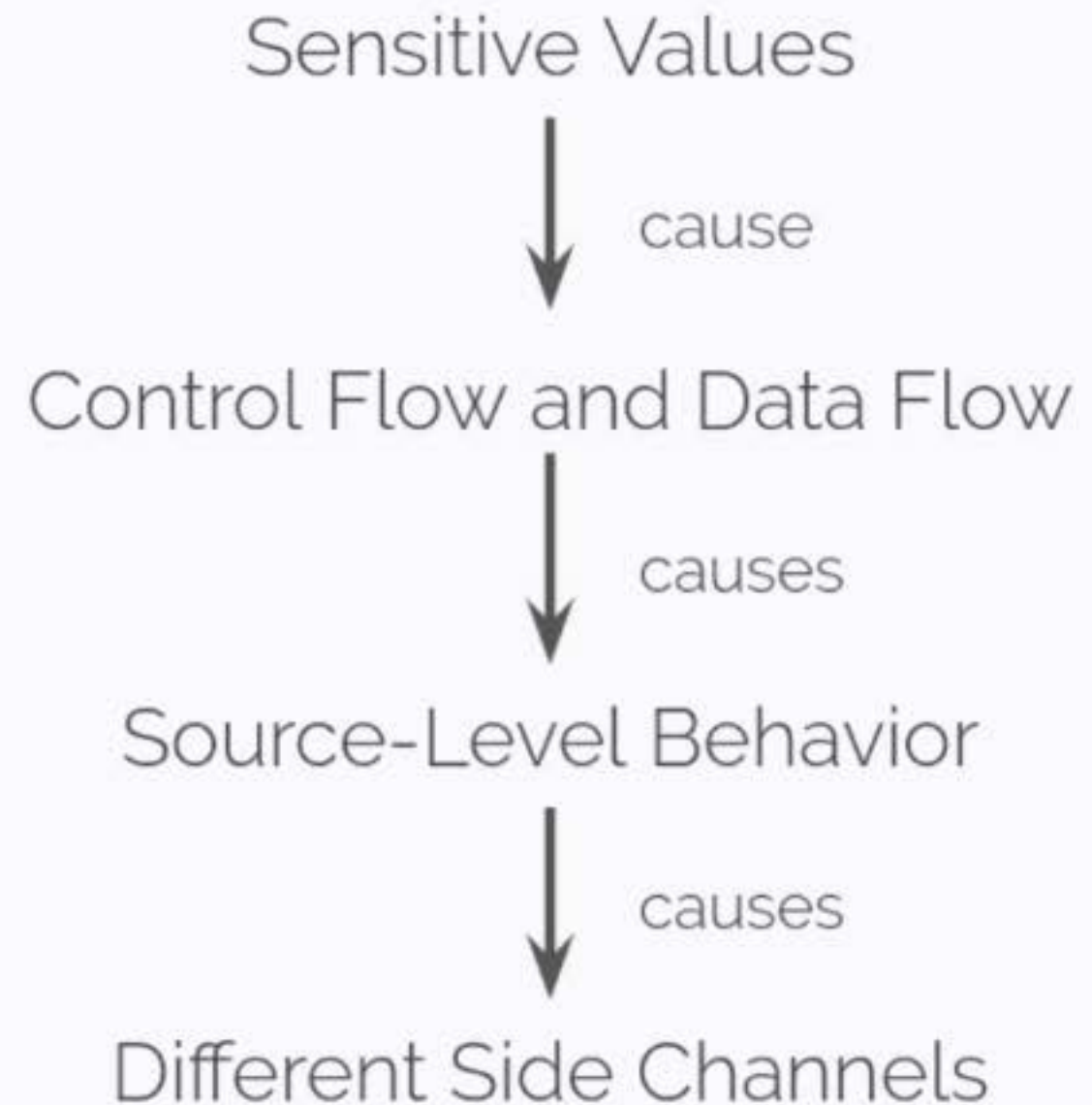
Source-Level Behavior



causes

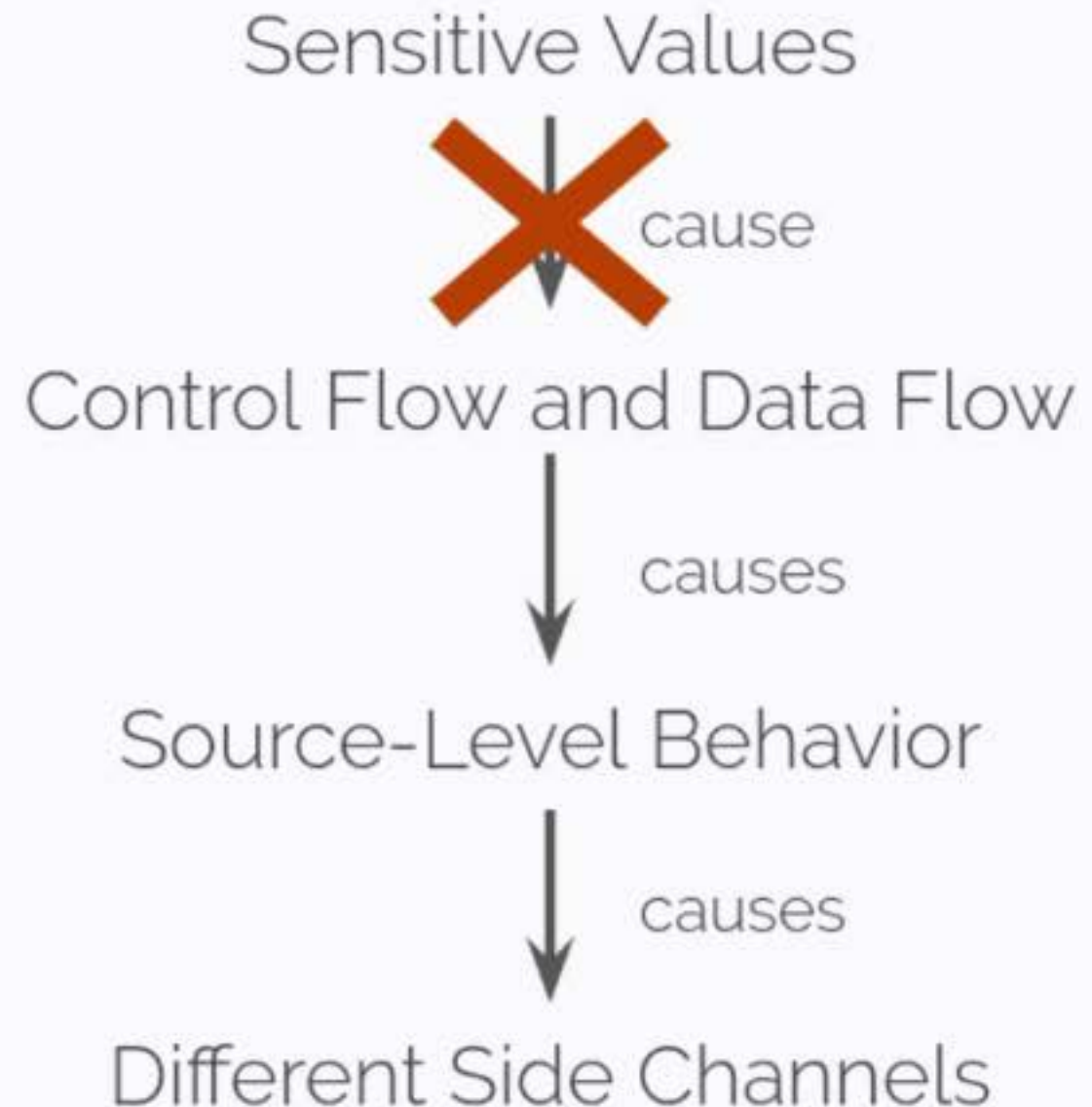
Different Side Channels

Key Insight Behind Our Solutions



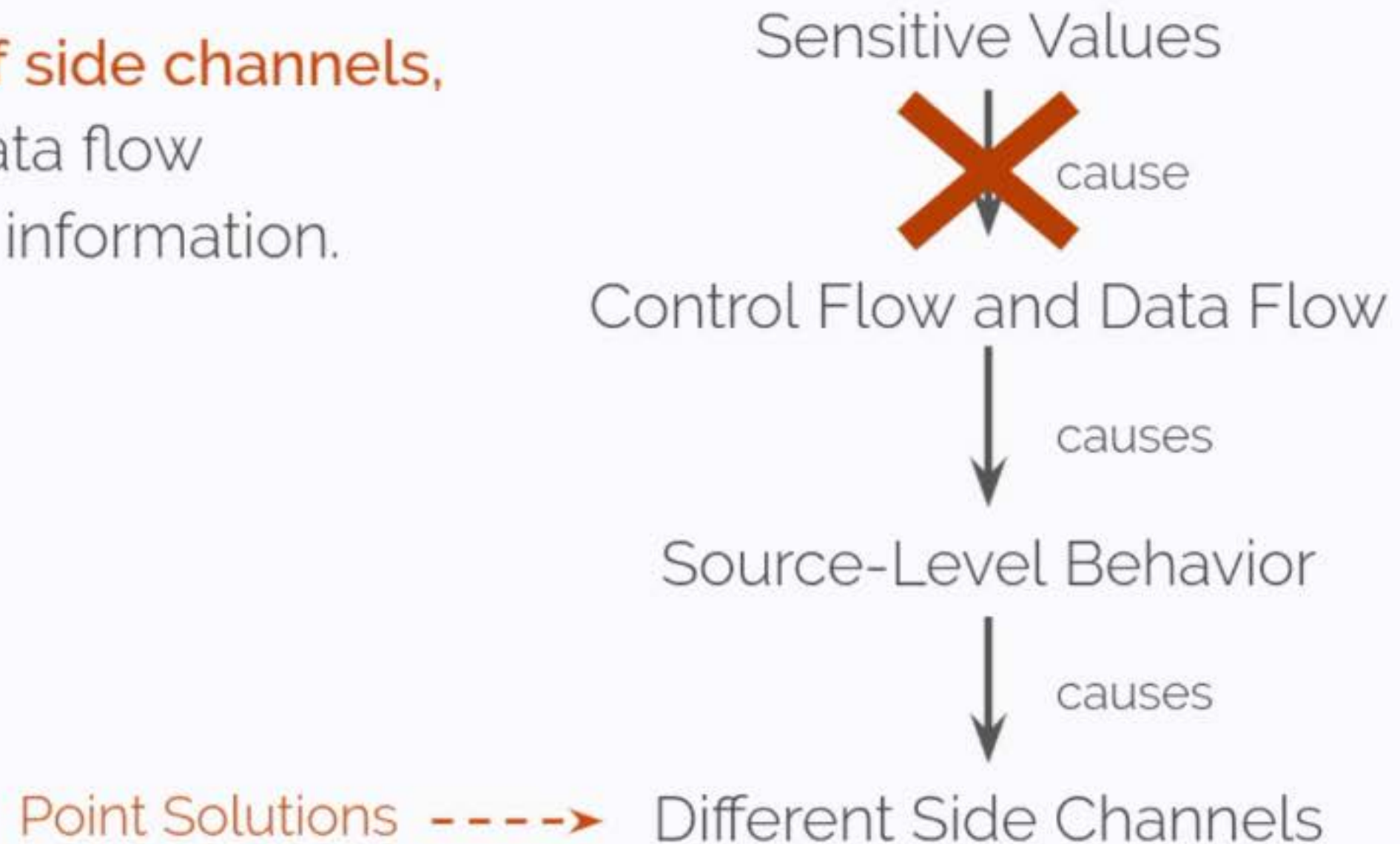
Key Insight Behind Our Solutions

To close a broad class of side channels, make control flow and data flow independent of sensitive information.



Key Insight Behind Our Solutions

To close a broad class of side channels, make control flow and data flow independent of sensitive information.



Solution: **Execute All Paths**

```
if (secret_bit == 1) {  
    z = (msg * z * z) mod n;  
} else {  
    z = (z * z) mod n;  
}
```

Solution: **Execute All Paths**

```
if (secret_bit == 1) {  
→ z = (msg * z * z) mod n;  
} else {  
    z = (z * z) mod n;  
}
```

Adversary sees `secret_bit = 1`

Solution: **Execute All Paths**


```
if (secret_bit == 1) {  
    z = (msg * z * z) mod n;  
} else {  
    → z = (z * z) mod n;  
}
```

Adversary sees `secret_bit = 1`
and `secret_bit != 1`.

Predication to Execute All Paths

Original Program

```
if (secret_bit == 1) {  
    z = (msg * z * z) mod n;  
} else {  
    z = (z * z) mod n;  
}
```



Predication to Execute All Paths

Original Program

```
if (secret_bit == 1) {  
    z = (msg * z * z) mod n;  
} else {  
    z = (z * z) mod n;  
}
```



Transformed Program

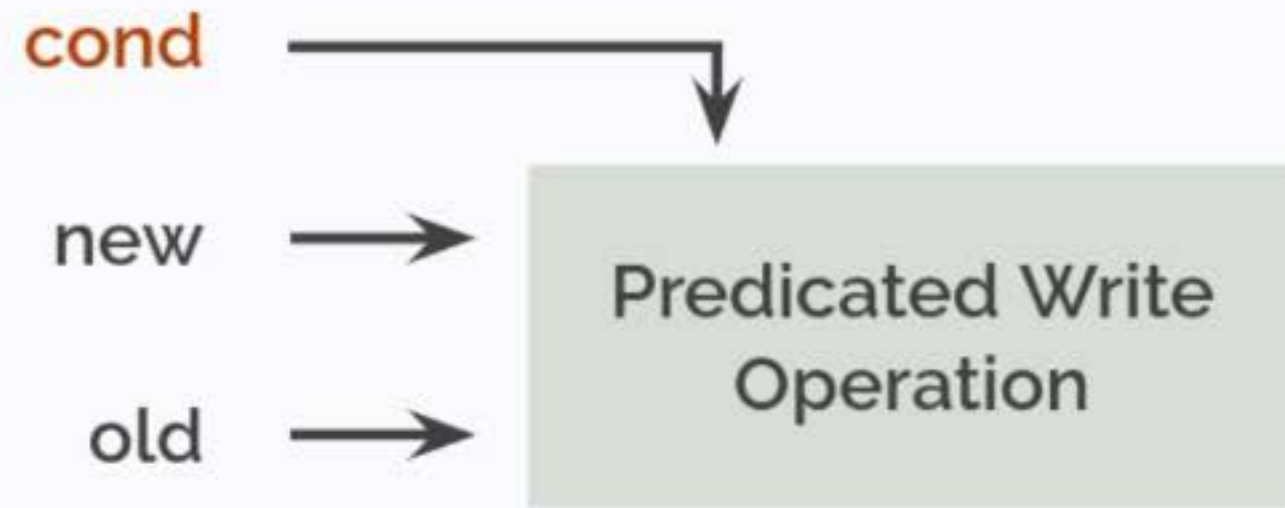
```
p = (secret_bit == 1)  
p : z = (msg * z * z) mod n;  
¬p: z = (z * z) mod n;
```

Key Building Block: **Software Predication**

Key Building Block: **Software Predication**



Key Building Block: **Software Predication**



Key Building Block: **Software Predication**



Key Building Block: **Software Predication**



Implementation in x64 assembly:

```
mov    old -> output    // Set destination  
test   cond, cond      // Check if non-zero  
cmovz new -> output    // Conditional update  
test   0, 0           // Overwrite flags
```

Also implemented using ARM v7, ARM v8, and RISC-V assembly instructions.

Key Building Block: **Software Predication**

1. Straight-line control flow

```
mov    old -> output    // Set destination
test   cond, cond       // Check if non-zero
cmovz  new -> output    // Conditional update
test   0, 0             // Overwrite flags
```

Key Building Block: **Software Predication**

1. Straight-line control flow
2. All data in registers;
no pointer dereferences

```
mov    old -> output    // Set destination
test   cond, cond       // Check if non-zero
cmovz  new -> output    // Conditional update
test   0, 0            // Overwrite flags
```


Key Building Block: **Software Predication**

1. Straight-line control flow
2. All data in registers;
no pointer dereferences
3. Fixed execution time

```
mov    old -> output    // Set destination
test   cond, cond      // Check if non-zero
cmovz new -> output    // Conditional update
test   0, 0            // Overwrite flags
```

Software Predication to Execute All Paths

Original Program

```
if (secret_bit == 1) {  
    z = (msg * z * z) mod n  
} else {  
    z = (z * z) mod n  
}
```



Transformed Program

```
p = (secret_bit == 1)  
z1 = (msg * z * z) mod n  
z2 = (z * z) mod n  
  
z = pred_write(p, z1, z)  
z = pred_write(!p, z2, z)
```

Software Predication to Execute All Paths

Original Program

```
if (secret_bit == 1) {  
    z = (msg * z * z) mod n  
} else {  
    z = (z * z) mod n  
}
```



Transformed Program

```
p = (secret_bit == 1)  
z1 = (msg * z * z) mod n  
z2 = (z * z) mod n  
  
z = pred_write(p, z1, z)  
z = pred_write(!p, z2, z)
```

Software Predication to Execute All Paths

Original Program

```
if (secret_bit == 1) {  
    z = (msg * z * z) mod n  
} else {  
    z = (z * z) mod n  
}
```



Transformed Program

```
p = (secret_bit == 1)  
z1 = (msg * z * z) mod n  
z2 = (z * z) mod n  
  
z = pred_write(p, z1, z)  
z = pred_write(!p, z2, z)
```

Software Predication to Execute All Paths

Original Program

```
if (secret_bit == 1) {  
    z = (msg * z * z) mod n  
} else {  
    z = (z * z) mod n  
}
```



Transformed Program

```
p = (secret_bit == 1)  
z1 = (msg * z * z) mod n  
z2 = (z * z) mod n  
  
z = pred_write(p, z1, z)  
z = pred_write(!p, z2, z)
```

Software Predication to Execute All Paths

Original Program

```
if (secret_bit == 1) {  
    z = (msg * z * z) mod n  
} else {  
    z = (z * z) mod n  
}
```



Transformed Program

```
p = (secret_bit == 1)  
z1 = (msg * z * z) mod n  
z2 = (z * z) mod n  
  
z = pred_write(p, z1, z)  
z = pred_write(!p, z2, z)
```

But Predication **May Crash the Program**

Original Program

```
v = 0;  
if (secret) {  
    v = 10;  
    y = x / v;  
}
```

But Predication **May Crash the Program**

Original Program

```
v = 0;  
if (secret) {  
    v = 10;  
    y = x / v;  
}
```



Transformed Program

```
v = 0;  
v = pred_write(secret, 10, v);  
y = pred_write(secret, x/v, y);
```


But Predication **May Crash the Program**

Original Program

```
v = 0;  
if (secret) {  
    v = 10;  
    y = x / v;  
}
```



Transformed Program

```
v = 0;  
v = pred_write(secret, 10, v);  
y = pred_write(secret, x/v, y);
```

But Predication **May Crash the Program**

Original Program

```
v = 0;  
if (secret) {  
    v = 10;  
    y = x / v;  
}
```



Transformed Program

```
v = 0;  
v = pred_write(secret, 10, v);  
y = pred_write(secret, x/v, y);
```

If `secret` is false, `v` is not updated, hence `v` remains 0.

But Predication **May Crash the Program**

Original Program

```
v = 0;  
if (secret) {  
    v = 10;  
    y = x / v;  
}
```



Transformed Program

```
v = 0;  
v = pred_write(secret, 10, v);  
y = pred_write(secret, x/v, y);
```

If `secret` is false, `v` is not updated, hence `v` remains 0.

Division by zero exception causes program to terminate.

But Predication **May Crash the Program**

Original Program

```
v = 0;  
if (secret) {  
    v = 10;  
    y = x / v;  
}
```



Transformed Program

```
v = 0;  
v = pred_write(secret, 10, v);  
y = pred_write(secret, x/v, y);
```



```
v = 0;  
v = pred_write(secret, 10, v);  
t = pred_write(v == 0, 1, v);  
y = pred_write(secret, x/t, y);
```

But Predication **May Crash the Program**

Original Program

```
v = 0;  
if (secret) {  
    v = 10;  
    y = x / v;  
}
```



Transformed Program

```
v = 0;  
v = pred_write(secret, 10, v);  
y = pred_write(secret, x/v, y);
```



```
v = 0;  
v = pred_write(secret, 10, v);  
t = pred_write(v == 0, 1, v);  
y = pred_write(secret, x/t, y);
```

Our solution masks exceptions by covertly changing divisor value.

But Predication **May Crash the Program**

Original Program

```
v = 0;  
if (secret) {  
    v = 10;  
    y = x / v;  
}
```



Transformed Program

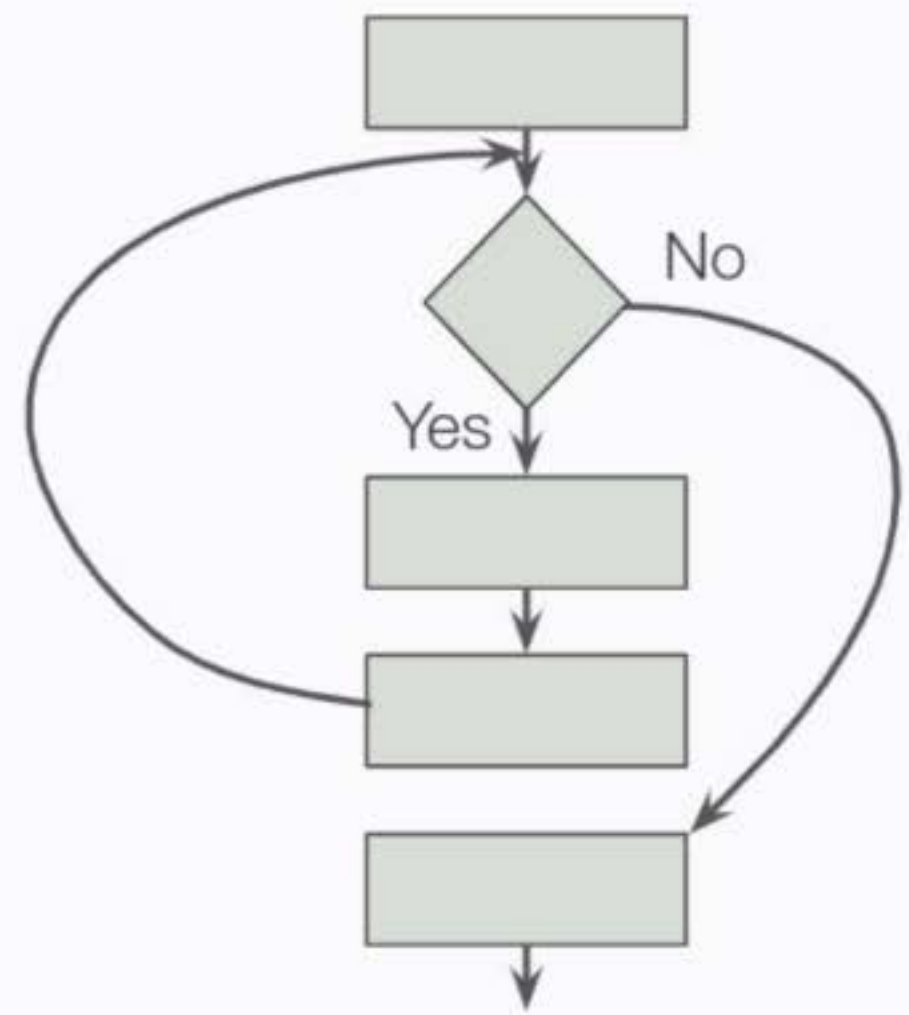
```
v = 0;  
v = pred_write(secret, 10, v);  
y = pred_write(secret, x/v, y);
```



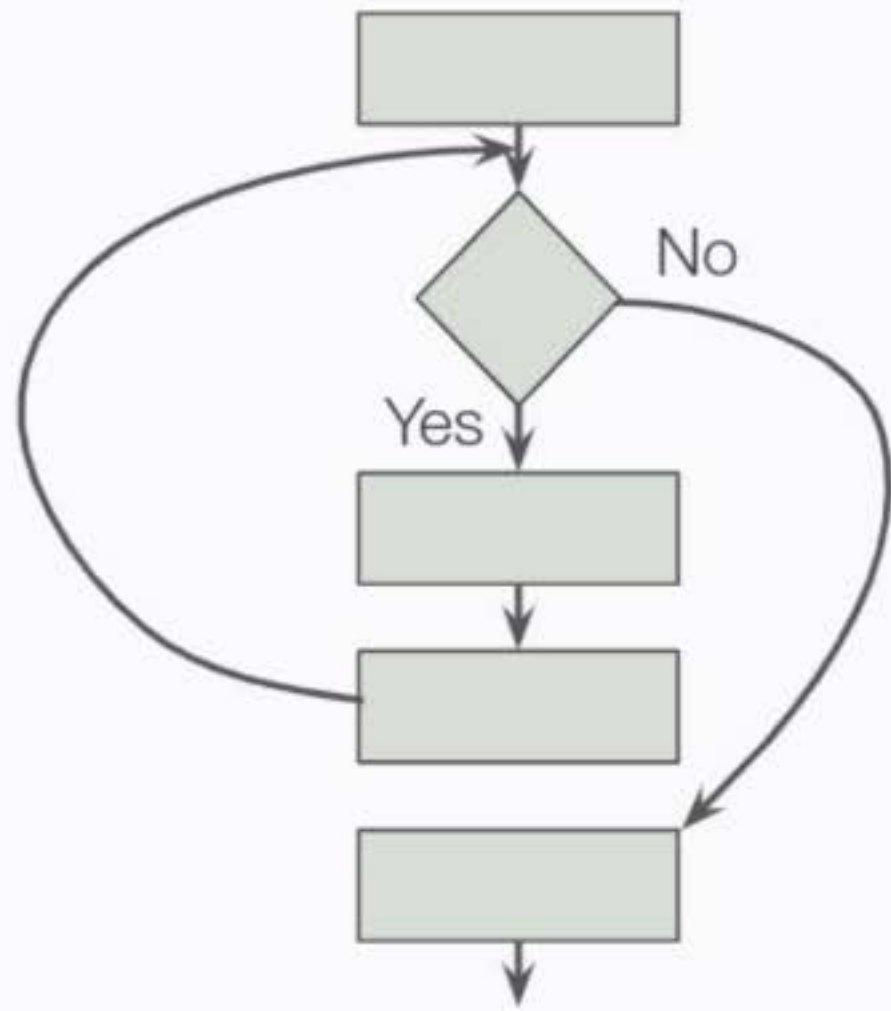
```
v = 0;  
v = pred_write(secret, 10, v);  
t = pred_write(v == 0, 1, v);  
y = pred_write(secret, x/t, y);
```

Our solution assumes that the pre-transformation program does not throw arch exceptions.

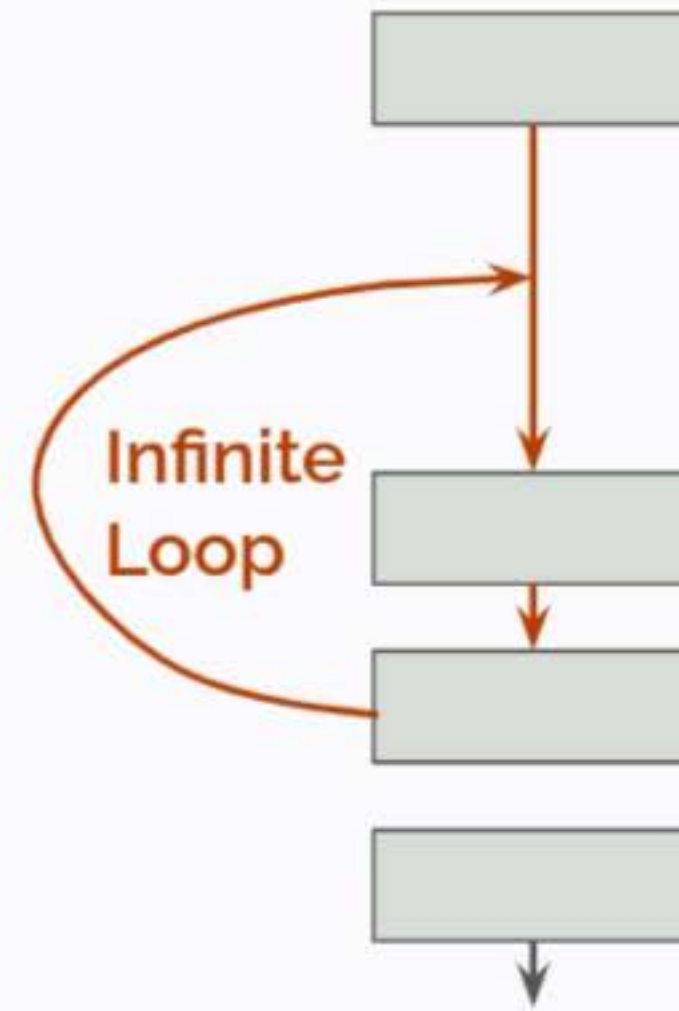
But Predication **May Cause Infinite Loops**



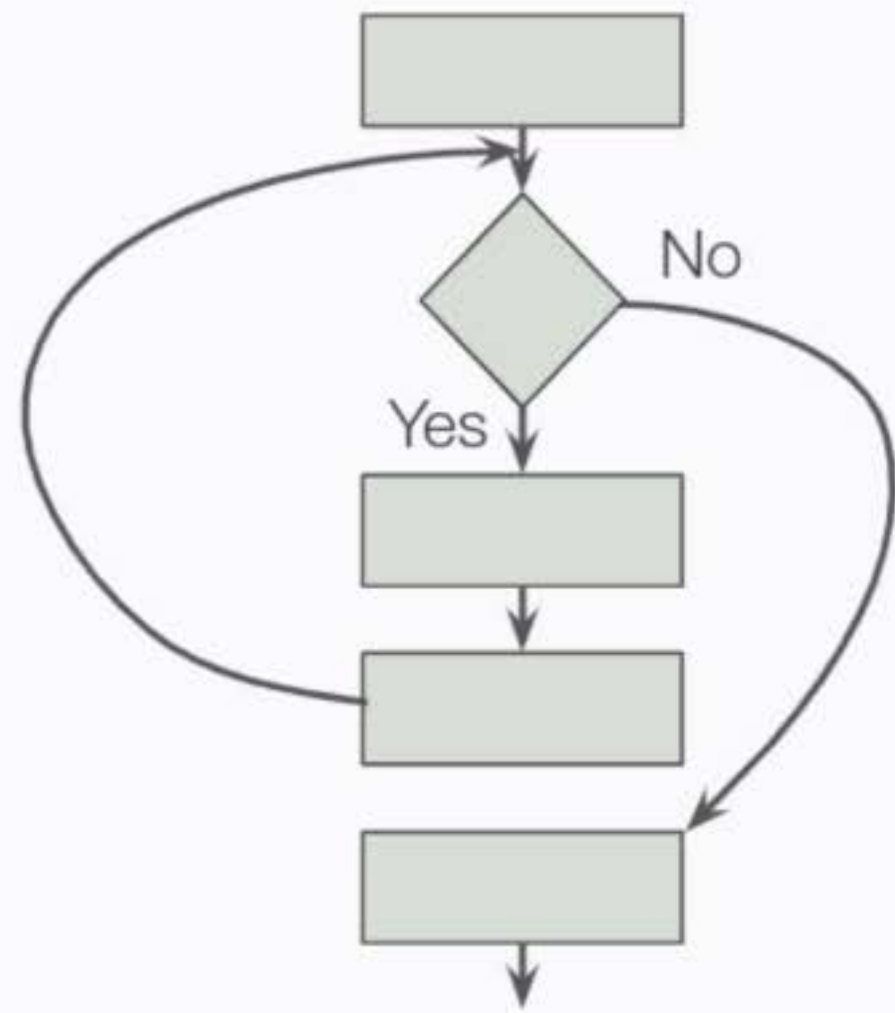
But Predication **May Cause Infinite Loops**



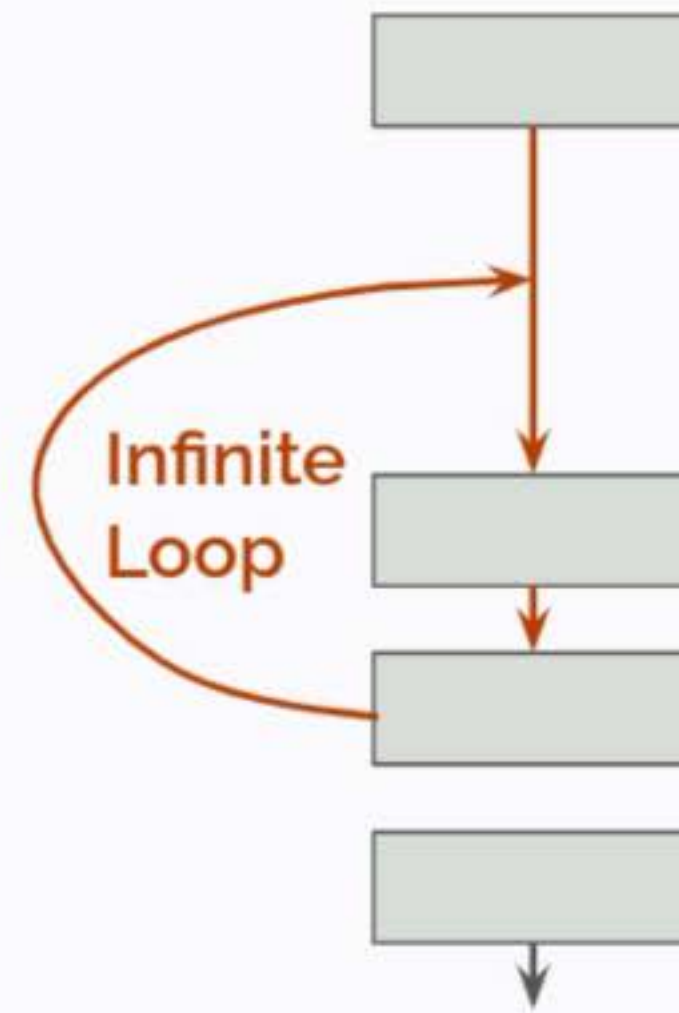
Incorrect code transformation



But Predication **May Cause Infinite Loops**



Incorrect code transformation




Loops require a different transformation.

Transforming Loops

```
loop i :: 0 to n  
  x = x * y;  
  i = i + 1;
```

Transforming Loops

```
loop i :: 0 to n  
  x = x * y;  
  i = i + 1;
```



Assume **n** is secret.

Transformation should hide the number of executed iterations.

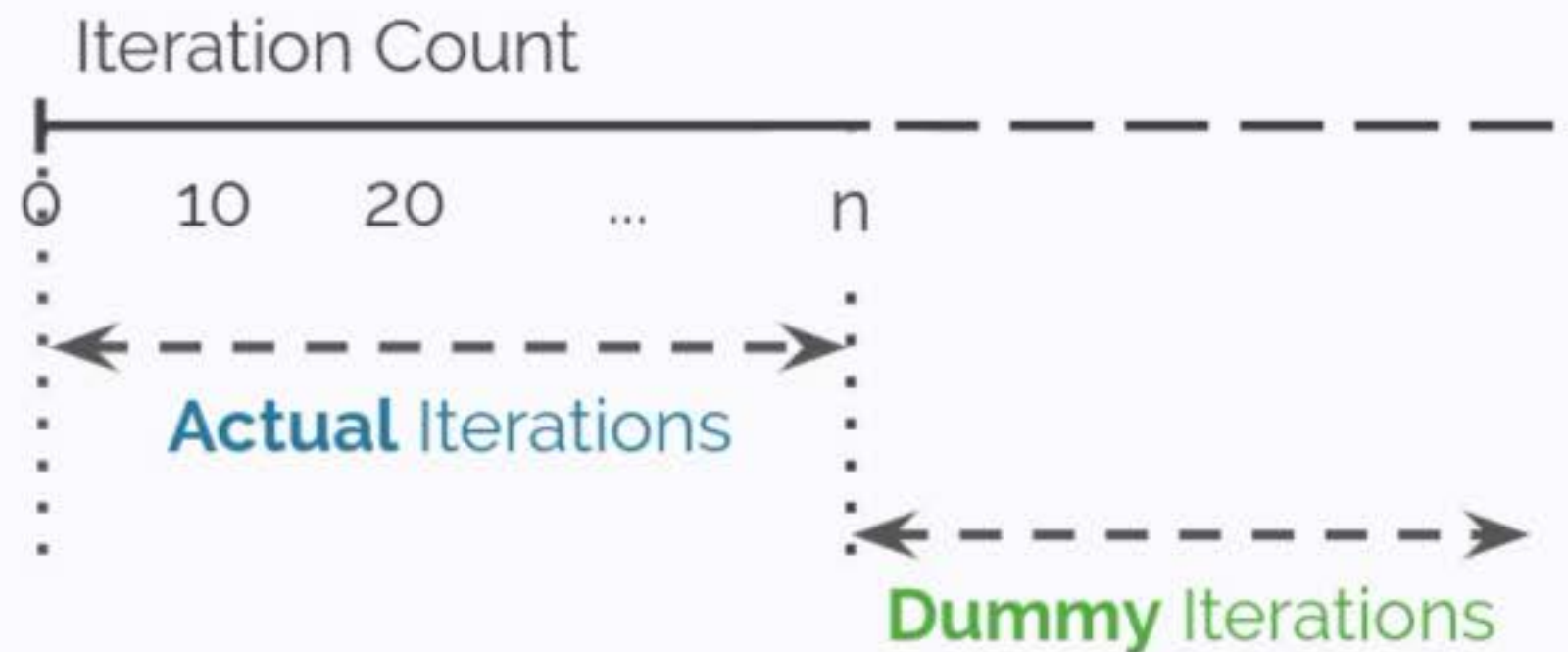
Transforming Loops

```
loop i :: 0 to n  
  x = x * y;  
  i = i + 1;
```

Assume **n** is secret.

Transformation should hide the number of executed iterations.

Our Solution's Approach



Transforming Loops

Original Program

```
loop i :: 0 to n  
  x = x * y;  
  i = i + 1;
```



Transforming Loops

Original Program

```
loop i :: 0 to n  
  x = x * y;  
  i = i + 1;
```



Transformed Program

```
i = 0  
  
loop ctr :: 0 to C  
  
ctr = ctr + 1;
```

Transforming Loops

Original Program

```
loop i :: 0 to n  
  x = x * y;  
  i = i + 1;
```



Transformed Program

```
i = 0  
  
loop ctr :: 0 to C  
  p: x = x * y;  
  p: i = i + 1;  
  
  ctr = ctr + 1;
```

New predicate
for loop body



Transforming Loops

Original Program

```
loop i :: 0 to n  
  x = x * y;  
  i = i + 1;
```



Transformed Program

```
i = 0  
p = TRUE  
loop ctr :: 0 to C  
  p: x = x * y;  
  p: i = i + 1;  
  
ctr = ctr + 1;
```


Transforming Loops

Original Program

```
loop i :: 0 to n
  x = x * y;
  i = i + 1;
```



Transformed Program

```
i = 0
p = TRUE
loop ctr :: 0 to C
  p: x = x * y;
  p: i = i + 1;
  Turn predicate OFF to
  run dummy iterations.

  i == n: p = FALSE;
  ctr = ctr + 1;
```

Transforming Loops

Original Program

```
loop i :: 0 to n
  x = x * y;
  i = i + 1;
```



Transformed Program

```
i = 0
p = TRUE
loop ctr :: 0 to C
  p: x = x * y;
  p: i = i + 1;
  i == n: p = FALSE;
  ctr = ctr + 1;
```

Annotated by user, for example:
__loop_count(1024)



Variations In Program Behavior

Control Flow

Data Flow

Variations in **Data Flow**

```
result = table[secret];
```

Variations in **Data Flow**

```
result = table[secret];
```



```
addr <- base(table) + secret  
result <- read addr
```

Variations in **Data Flow**

```
result = table[secret];
```



```
addr <- base(table) + secret  
result <- read addr
```

An adversary that can observe address, can also derive secret.

```
secret = addr - base(table)
```

Eliminating Variations in **Data Flow**

Solution #1: Array Streaming

Accesses the entire array to read one element of the array.

Expensive to access entire array, but vector instructions, caches, and prefetchers reduce latency.

Eliminating Variations in **Data Flow**

Solution #1: Array Streaming

Accesses the entire array to read one element of the array.

Expensive to access entire array, but vector instructions, caches, and prefetchers reduce latency.

Solution #2: Software ORAM

Software version of Path ORAM [CCS'13], which shuffles memory to hide location of data.

Variations In Program Behavior

Control Flow

Data Flow

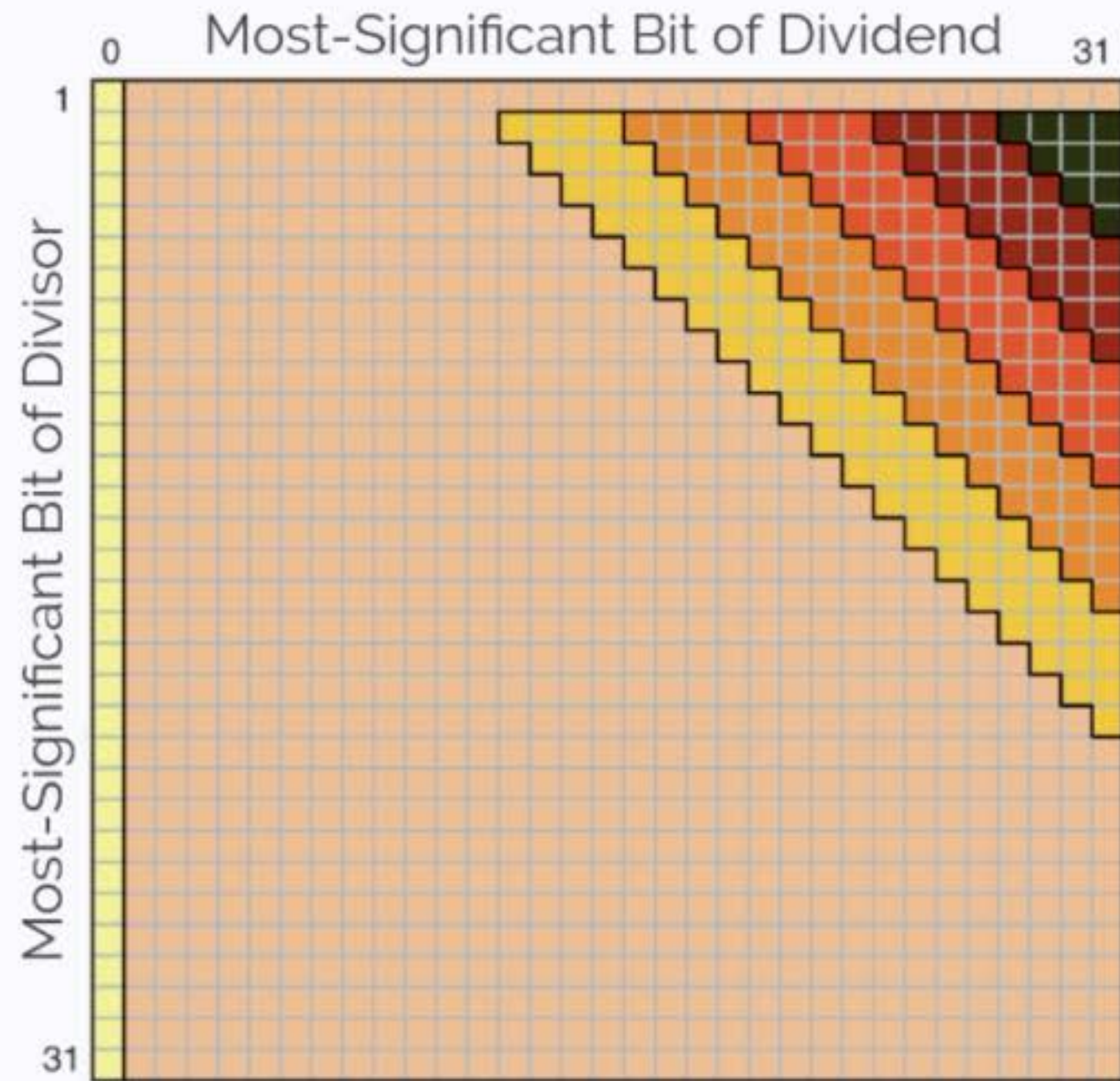
Variations In Program Behavior

Control Flow

Data Flow

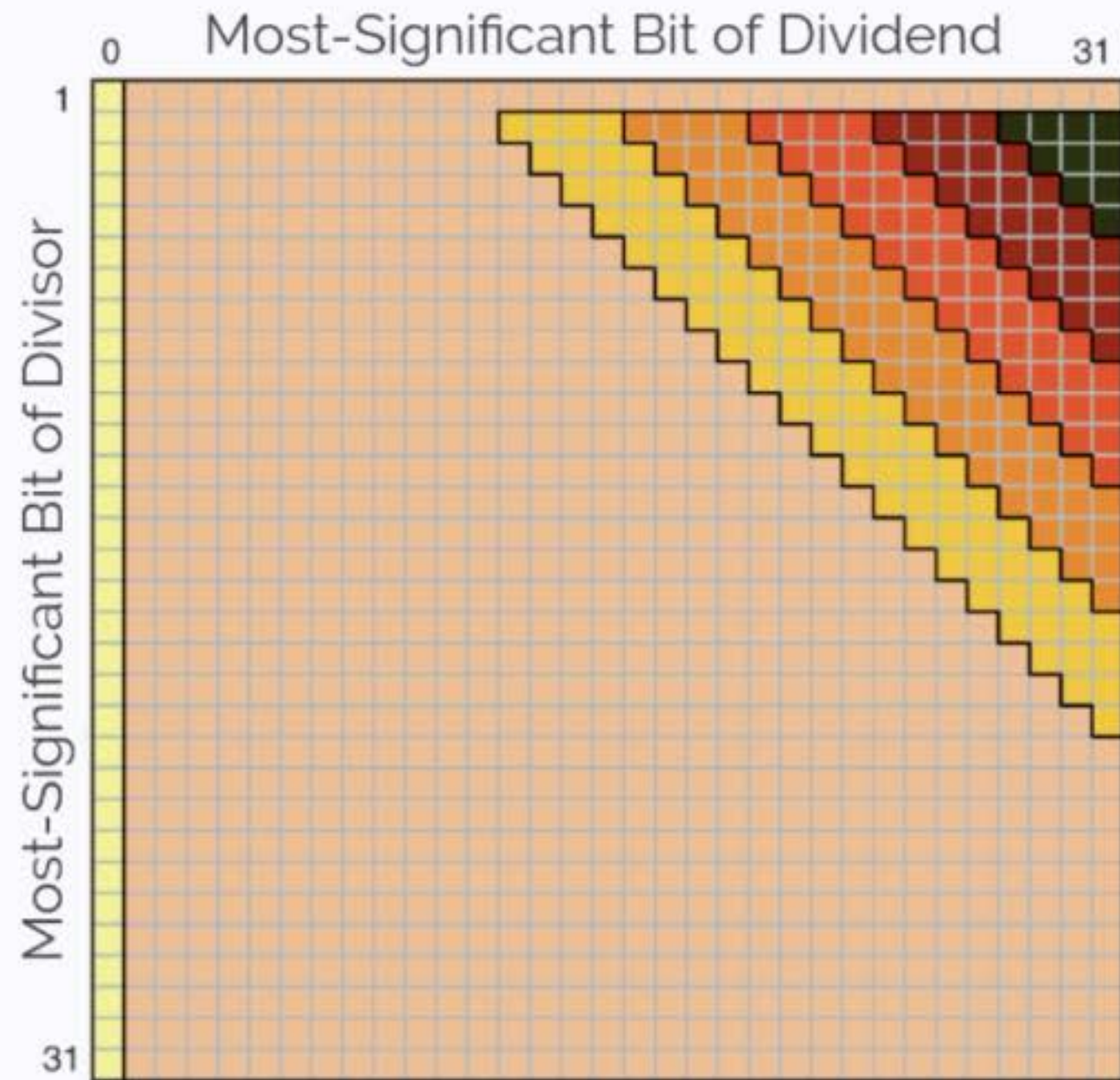
ISA Instructions

Example #1: Latency of Integer Division



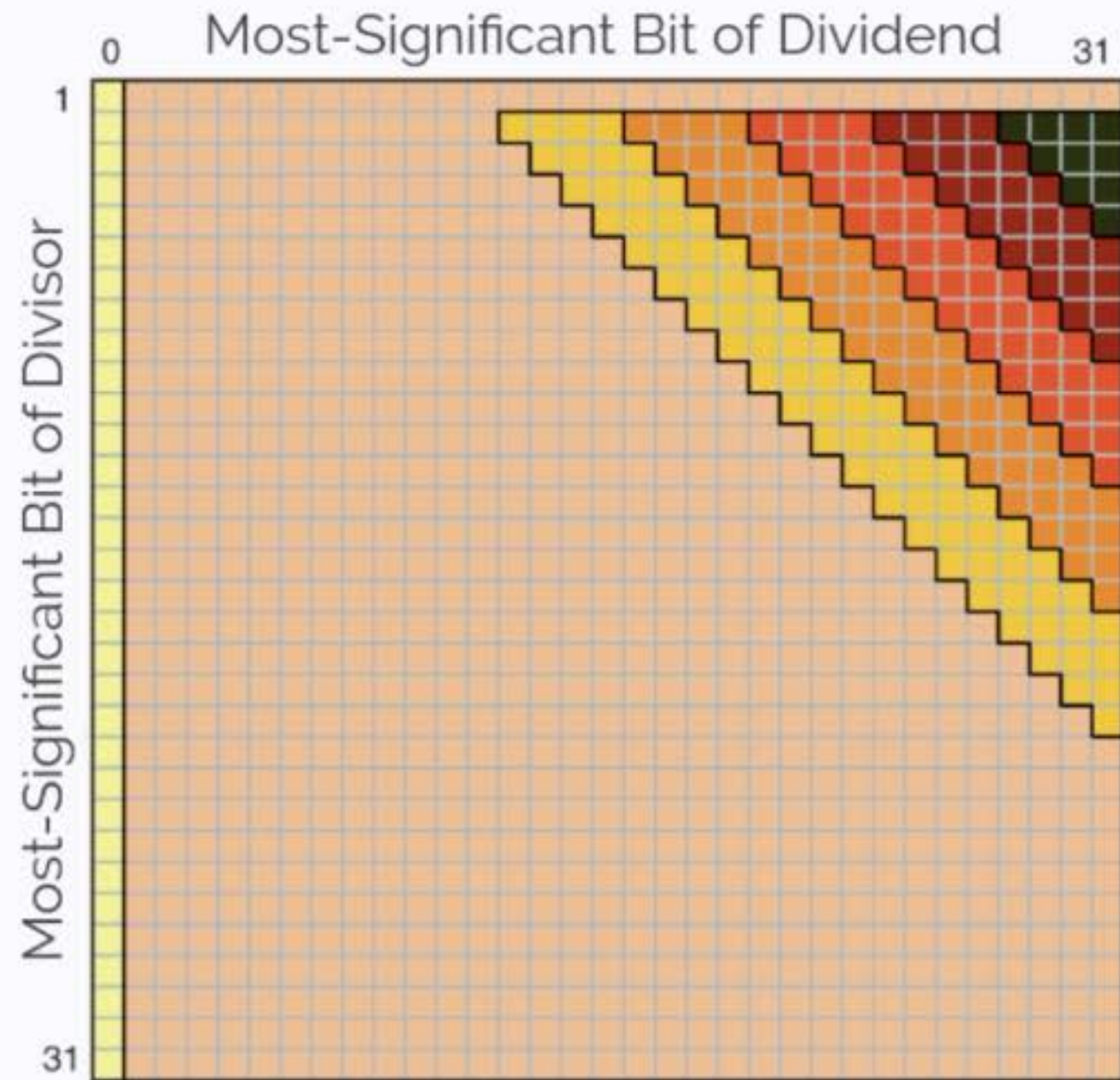
Cleemput et al., "Compiler Mitigations for Time Attacks on Modern x86 Processors", ACM TACO 2012.

Example #1: Latency of Integer Division



Colors indicate different execution times

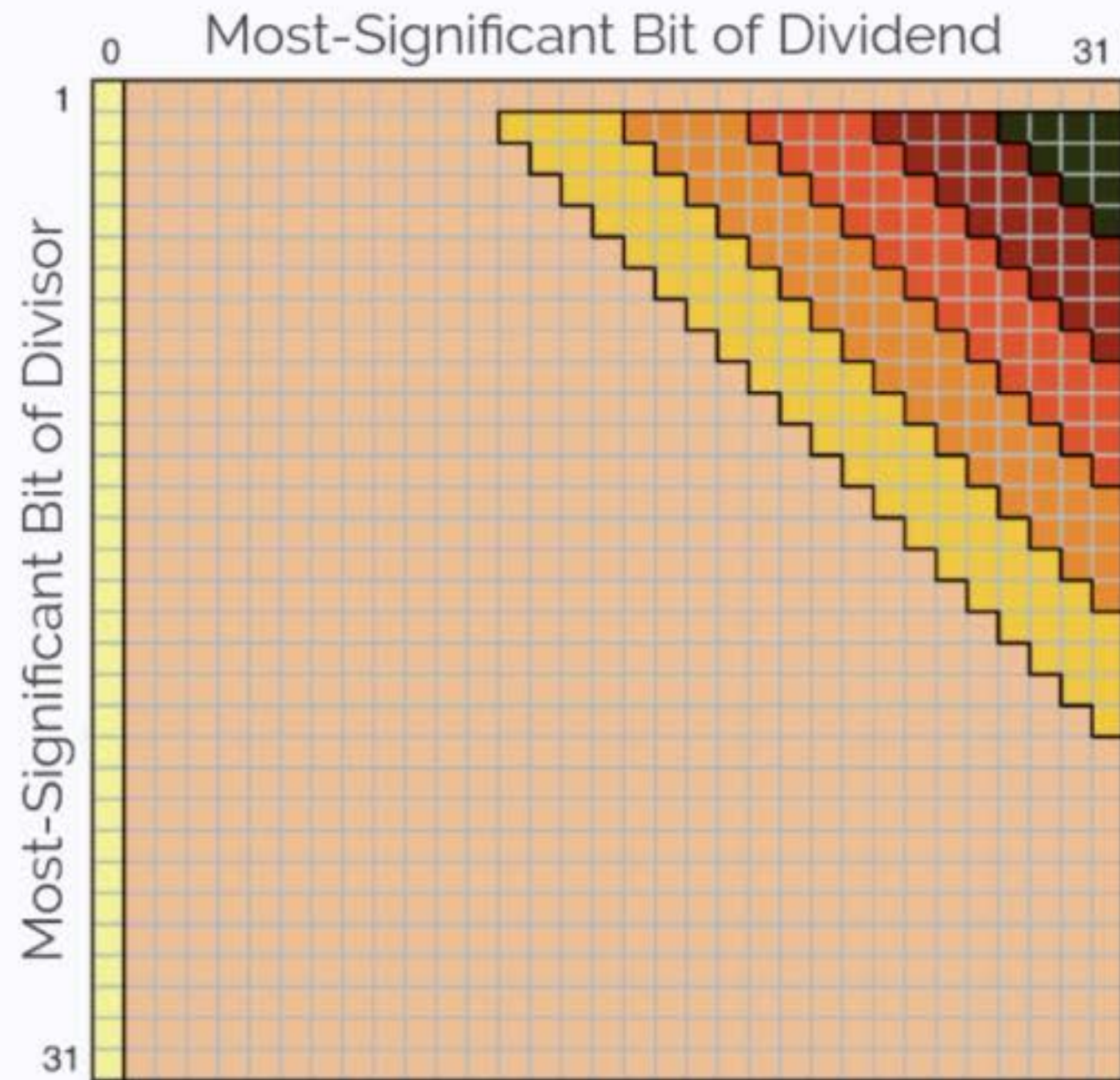
Example #1: Latency of Integer Division



Colors indicate different execution times

Execution time depends on the operand values, creating timing side channel

Example #1: Latency of Integer Division



Colors indicate different execution times

Execution time depends on the operand values, creating timing side channel

Solution: Rewrite division using bitwise arithmetic

Example #2: Power Consumption of Comparisons

C Program

```
result = (secret != 3)
```

Assembly Program

```
cmp    src , #3  
movw   dst <- #0  
movne  dst <- #1
```

Example #2: Power Consumption of Comparisons

C Program

```
result = (secret != 3)
```

Assembly Program

```
cmp    src , #3  
movw  dst <- #0  
movne dst <- #1
```

Compares `secret` with the literal constant 3

Example #2: Power Consumption of Comparisons

C Program

```
result = (secret != 3)
```

Assembly Program

```
cmp    src , #3  
movw   dst <- #0  
movne  dst <- #1
```

Preemptively copies literal 0 into the destination

Example #2: Power Consumption of Comparisons

C Program

```
result = (secret != 3)
```

Assembly Program

```
cmp    src , #3  
movw  dst <- #0  
movne dst <- #1
```

Conditionally copies literal 1 into the destination

Example #2: Power Consumption of Comparisons

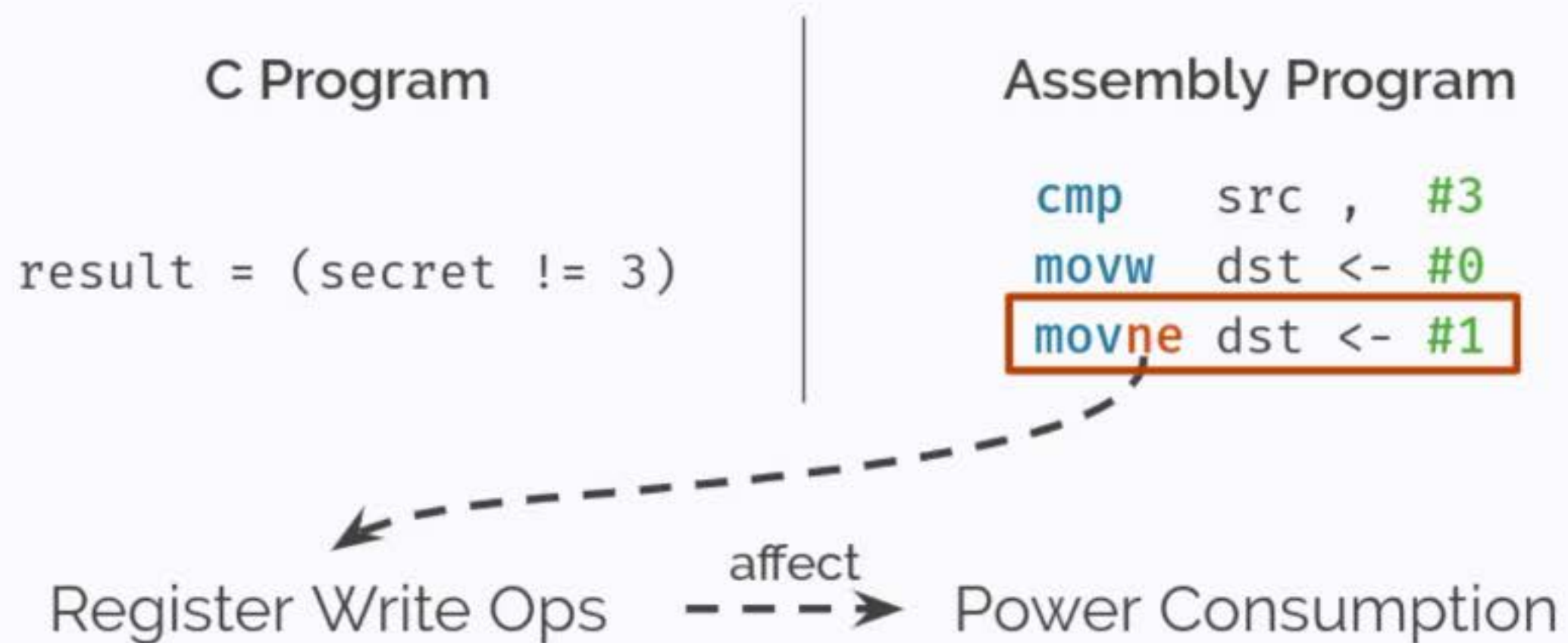
C Program

```
result = (secret != 3)
```

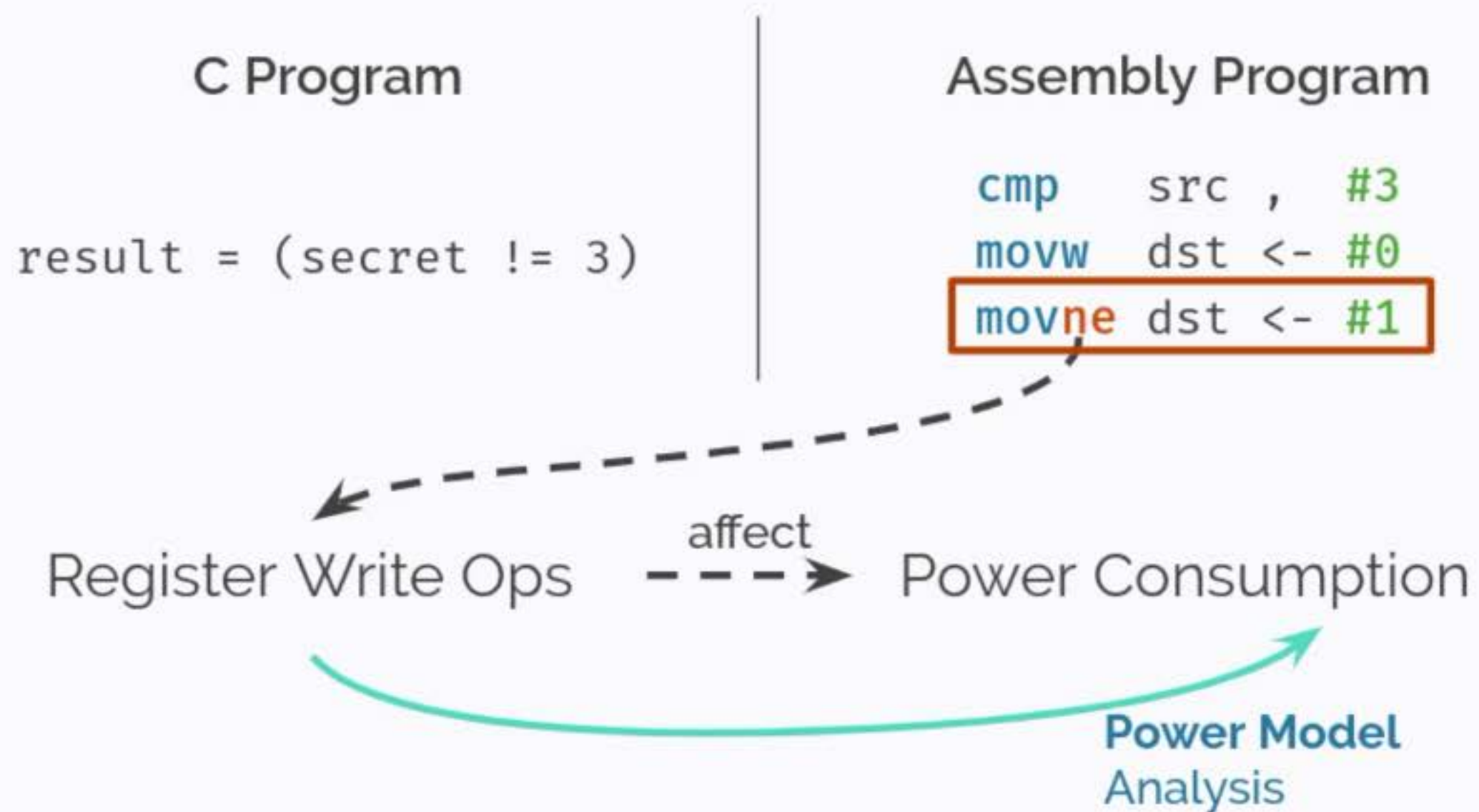
Assembly Program

```
cmp    src , #3  
movw   dst <- #0  
movne  dst <- #1
```

Example #2: Power Consumption of Comparisons



Example #2: Power Consumption of Comparisons



Analysis of **Power Models**

**Open Source
Power Model**

for example, McPAT

Analysis of **Power Models**

**Open Source
Power Model**

for example, McPAT

We apply **backward slicing** to identify the microarchitectural metrics that affect dynamic power consumption.

Analysis of **Power Models**

Open Source
Power Model

for example, McPAT

Closed Source
Power Model

for example, Intel RAPL

We apply **backward slicing** to identify the microarchitectural metrics that affect dynamic power consumption.

Analysis of **Power Models**

**Open Source
Power Model**

for example, McPAT

We apply **backward slicing** to identify the microarchitectural metrics that affect dynamic power consumption.

**Closed Source
Power Model**

for example, Intel RAPL

We fit a **regression model** to the RAPL power model, and we identify the inputs that have non-zero regression coefficients.

Superoptimizers for Alternate Code Sequences

Original
Assembly Program

```
cmp    src1,src2  
movw   dst <- #0  
movne  dst <- #1
```



Transformed
Assembly Program

```
sub    tmp <- src1,src2  
sub    dst <- src2,src1  
orr    dst <- dst , tmp  
lsr    dst <- dst , #31
```

Outline

- ▷ Our Solution's Design
- ▶ Core Principles that Enable Generalization
- ▷ Performance Comparison
- ▷ Future Work

Mapping between Instructions and Leakage

LLVM
Instructions



Side-Channel
Information Leakage

Mapping between Instructions and Leakage

LLVM
Instructions



Side-Channel
Information Leakage

```
div numerator denominator
```

```
numerator denominator
```

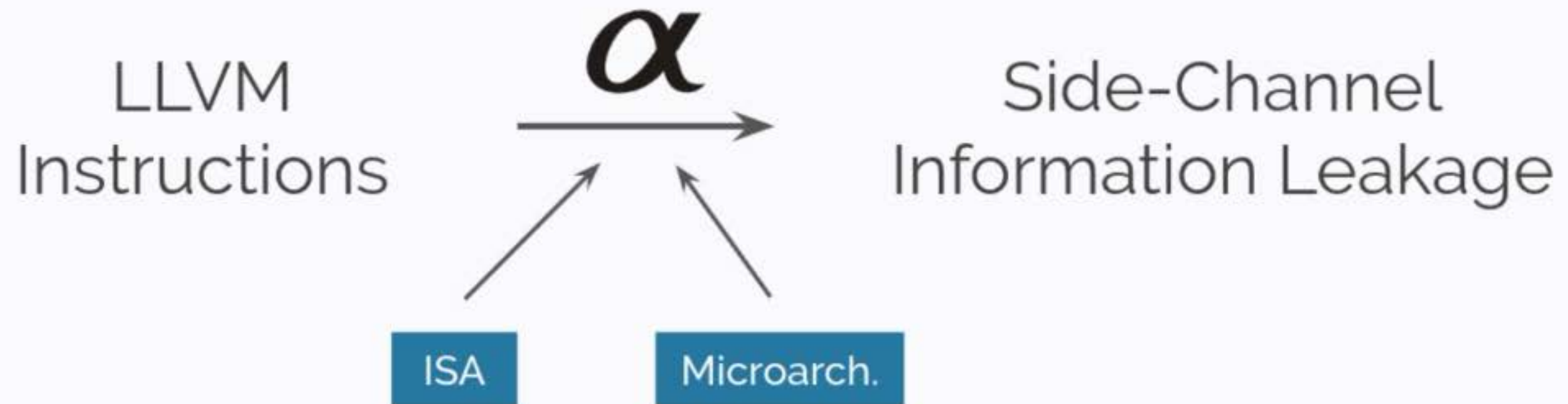
Mapping between Instructions and Leakage



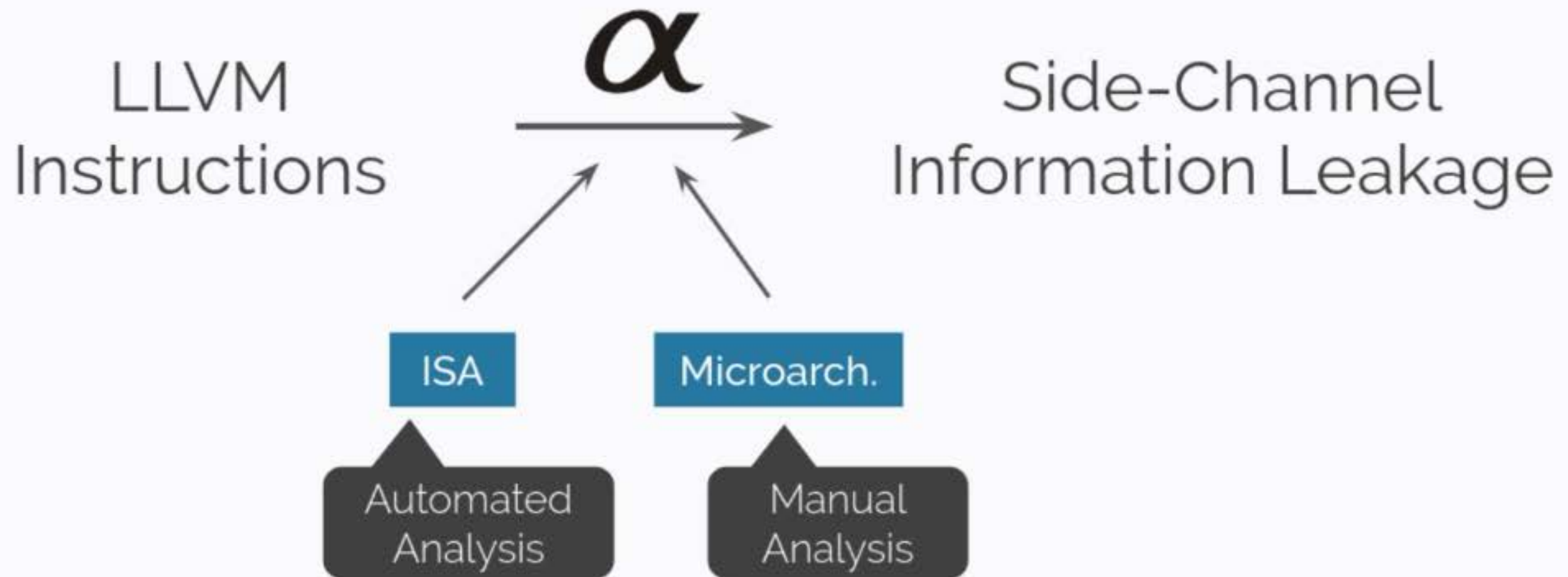
`div numerator denominator`

`numerator denominator`

Mapping between Instructions and Leakage



Mapping between Instructions and Leakage







f : $\mathbf{I} \rightarrow \mathbf{O}$
computation secret input data output results

α ↓

l : $\mathbf{I} \rightarrow \mathbf{S}$
information leakage secret input data side channel observations



Original Program

$$f(x)$$

Initial Approach to Close Side Channel

Original Program

$f(x)$



Initial Approach to Close Side Channel

Original Program

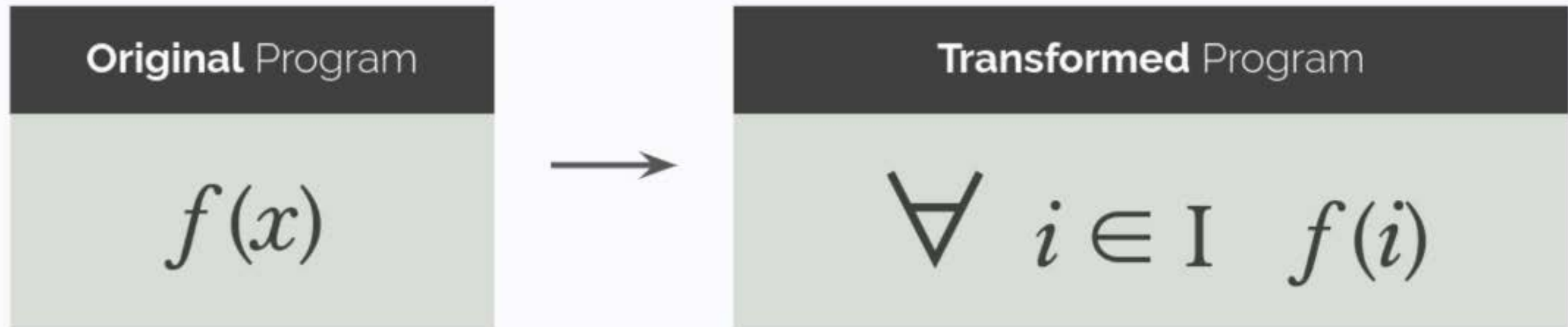
$f(x)$



Transformed Program

$\forall i \in I \quad f(i)$

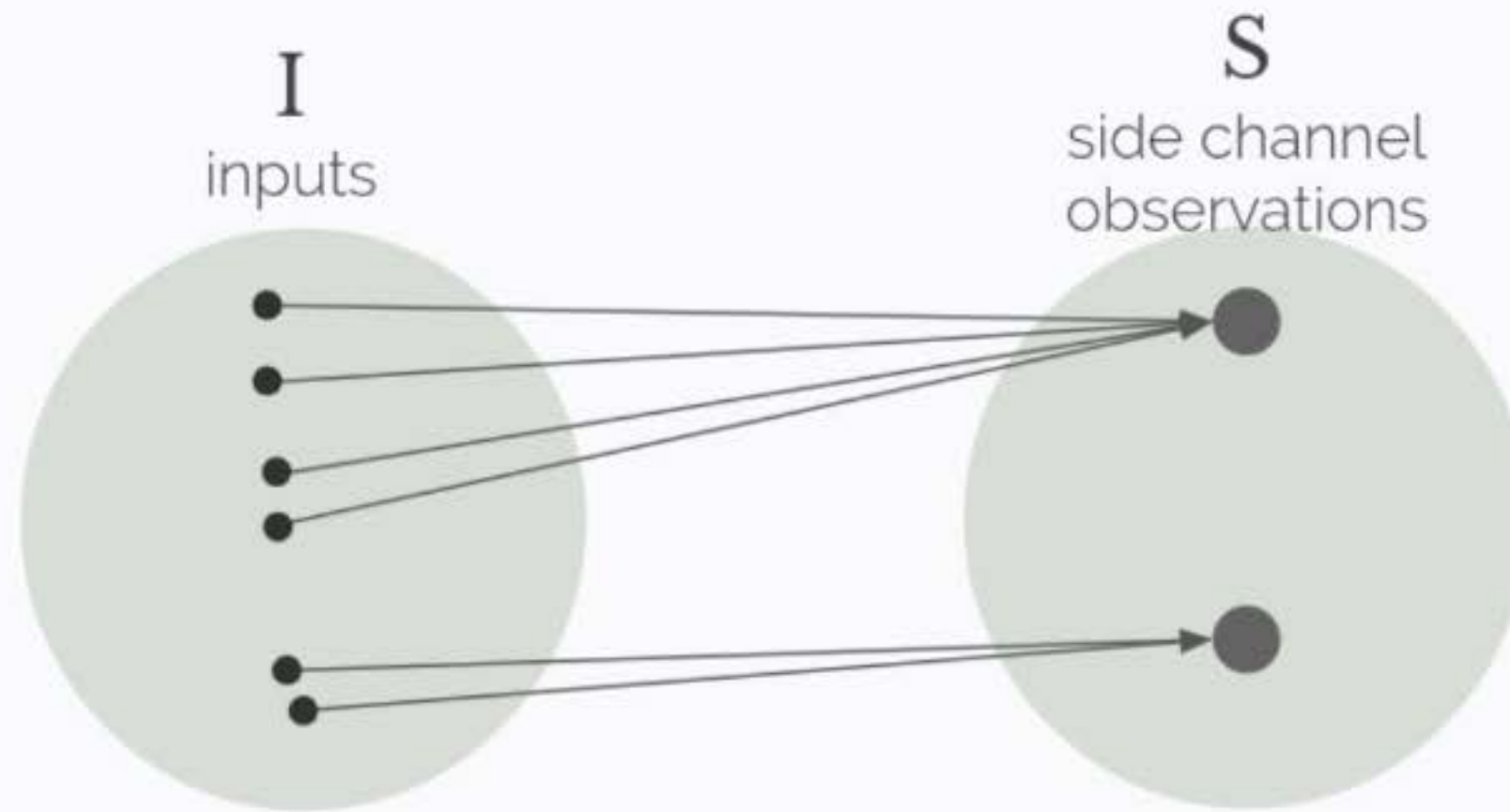
Initial Approach to Close Side Channel



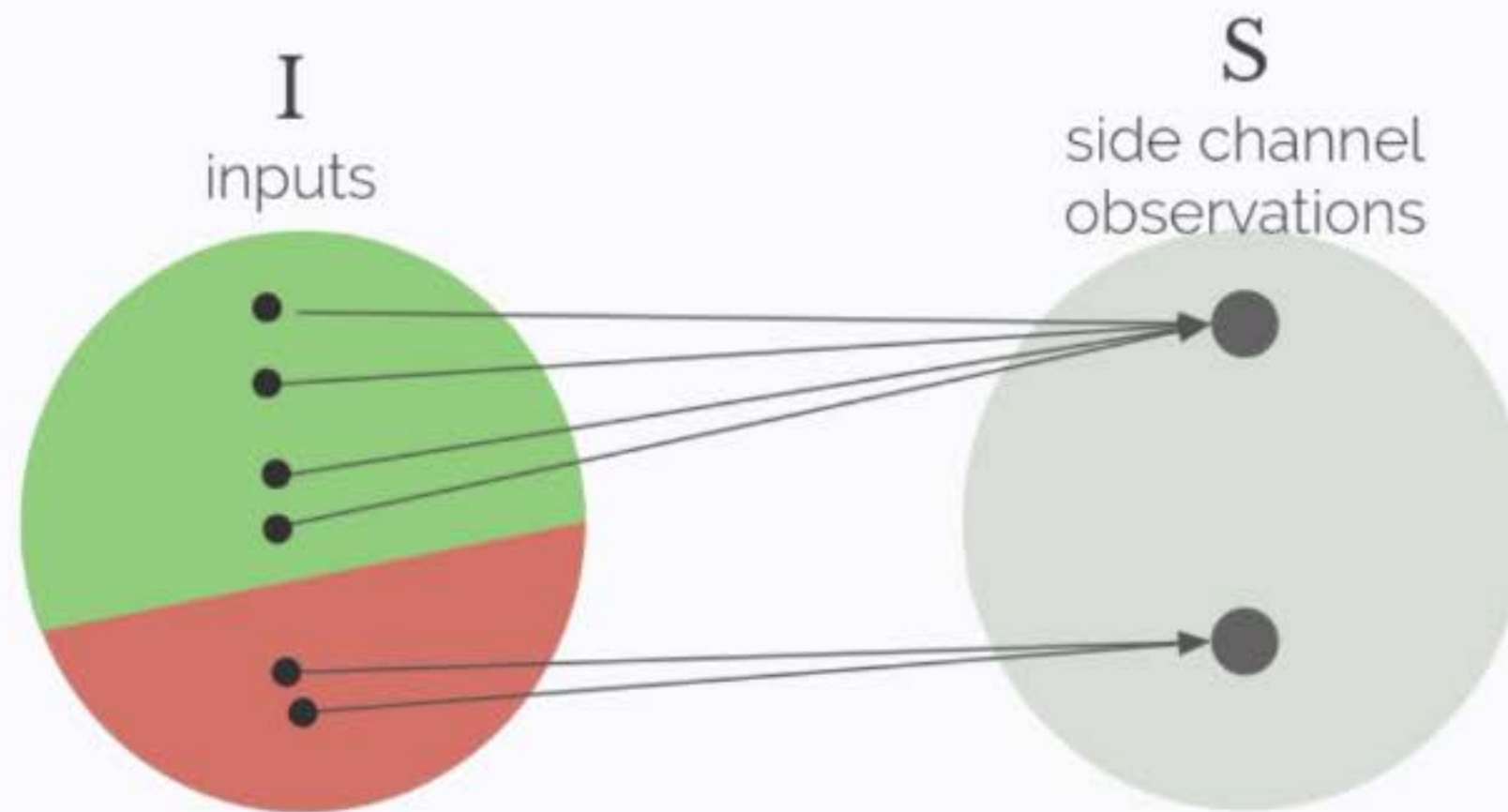
Inefficient if set of all possible inputs is large.

Approach #2: Slight Optimization

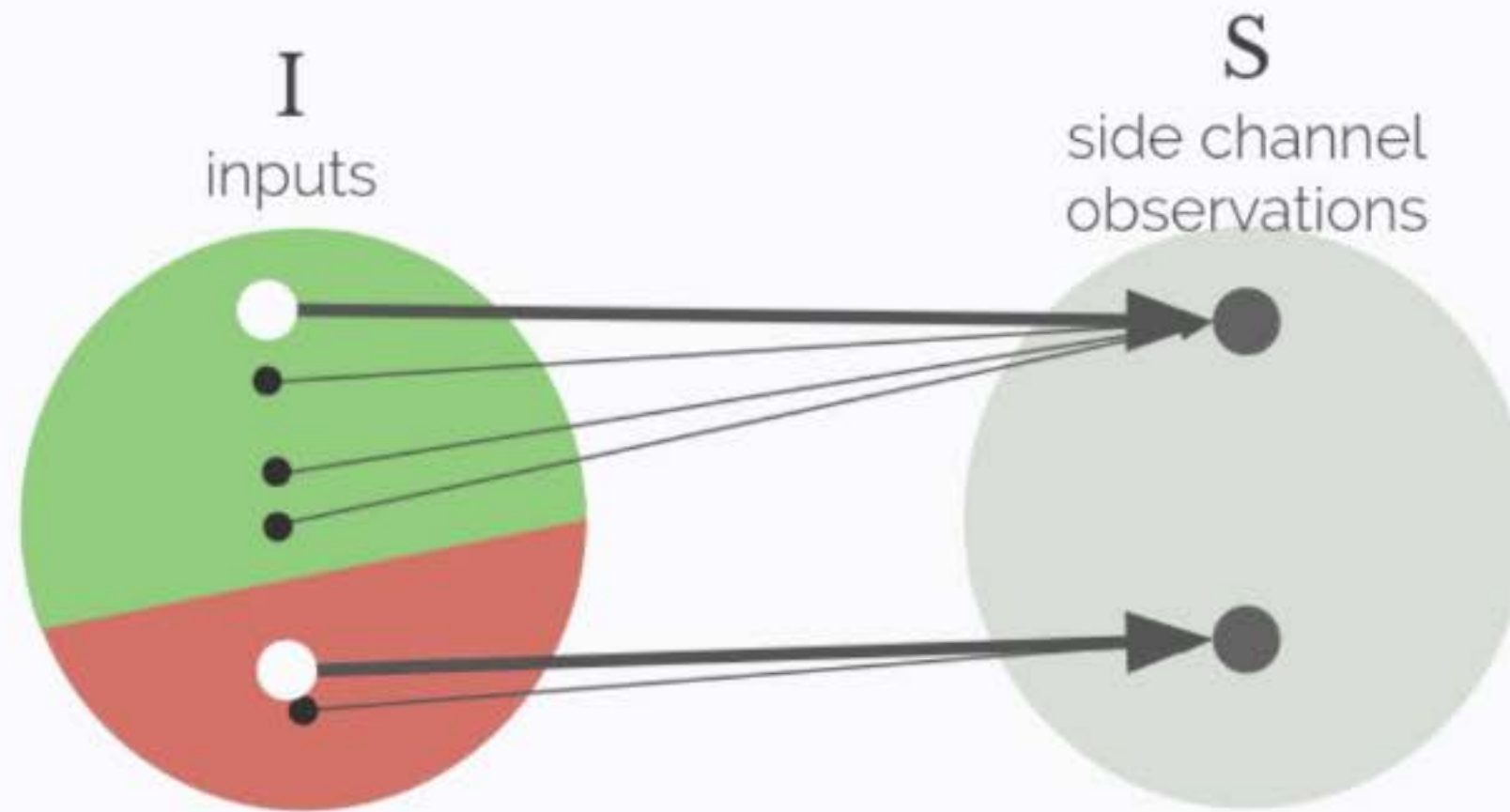
Approach #2: Slight Optimization



Approach #2: Slight Optimization



Approach #2: Slight Optimization



Use one representative input from each partition.

Approach #2: Slight Optimization

Original Program

$$f(x)$$



Transformed Program

~~$$\forall i \in I \quad f(i)$$~~

$$\forall i \in I/\sim \quad f(i)$$

Approach #3: Further Optimization

$$f : I \rightarrow O, \quad l : I \rightarrow S$$

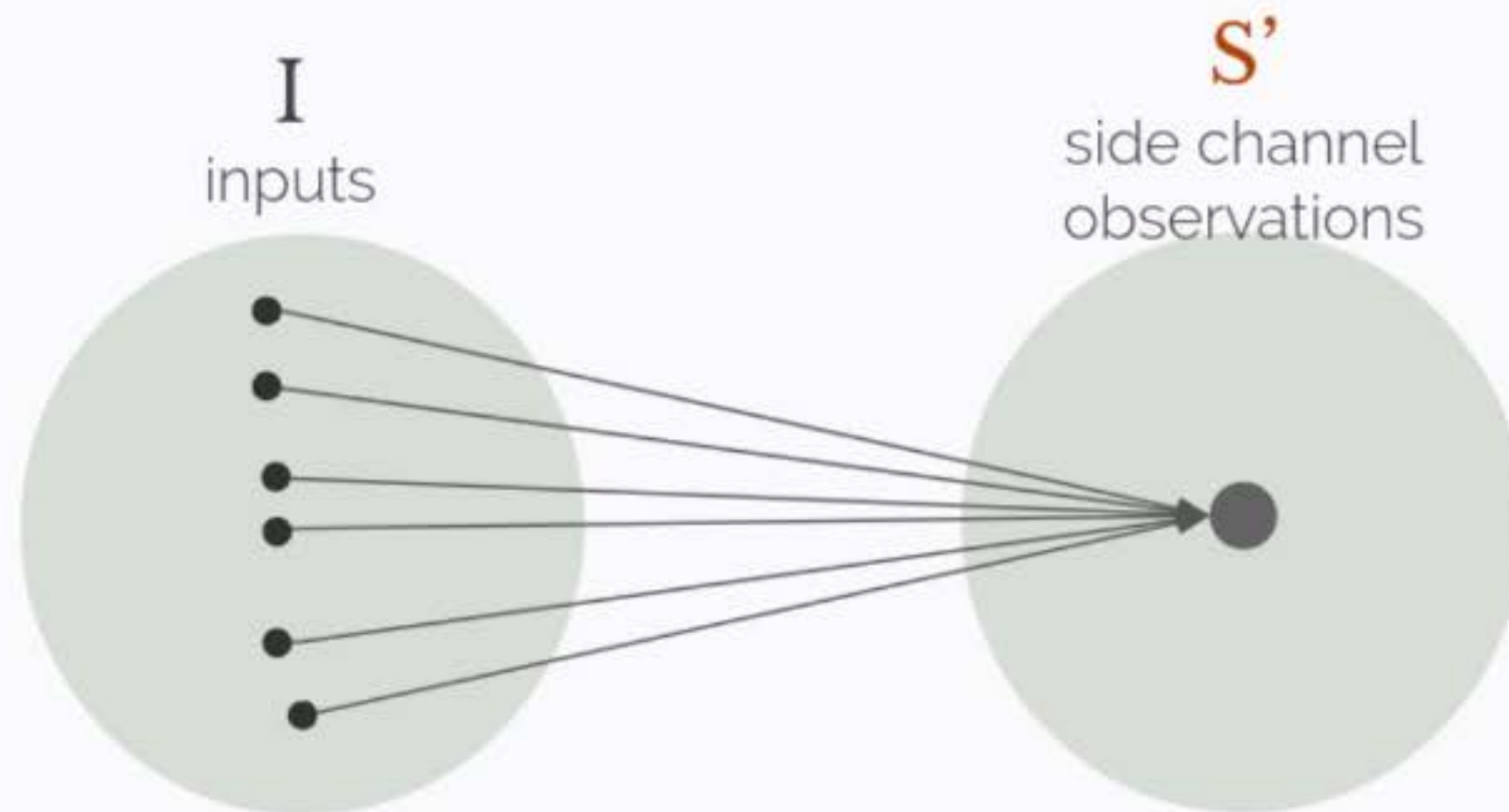
Approach #3: Further Optimization

$$f : I \rightarrow O,$$

$$l : I \rightarrow S$$

$$f' : I \rightarrow O,$$

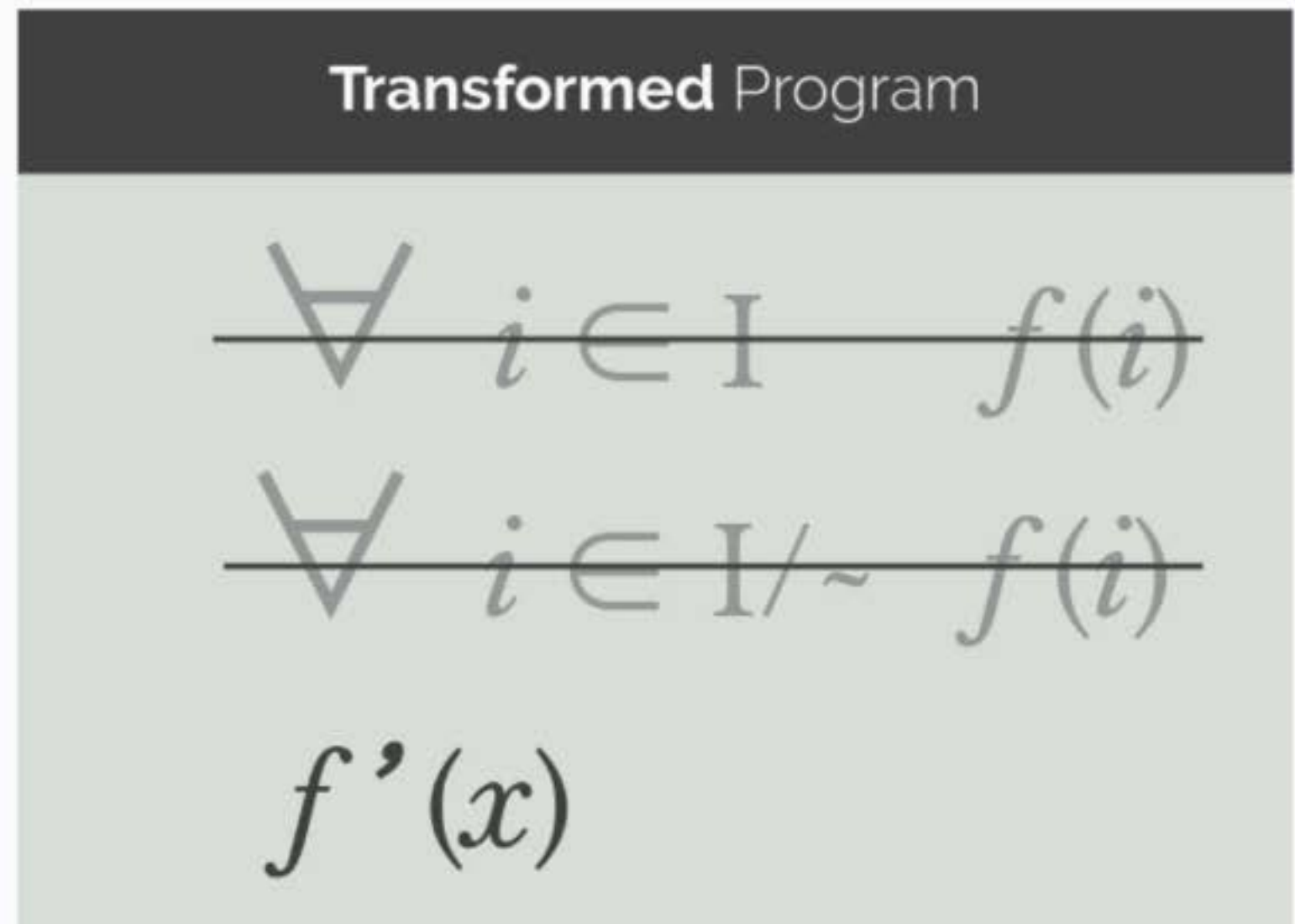
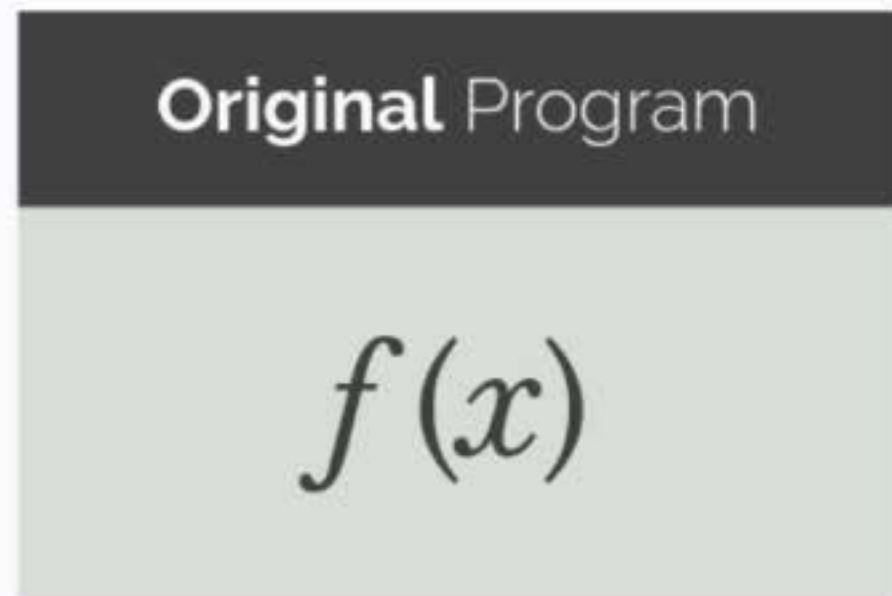
$$l' : I \rightarrow S' \quad \text{with } |S'| = 1$$



Approach #3: Further Optimization

$$f' : I \rightarrow O,$$

$$l' : I \rightarrow S' \quad \text{with } |S'| = 1$$



Implementation

Close a Broad Class
of Side Channels

Execute on Modern
Microprocessors

Protect a Diverse
Set of Applications

Digital Side Channels

e.g. address trace, cache,
branch predictor, etc.

Implementation

Close a Broad Class
of Side Channels

Execute on Modern
Microprocessors

Protect a Diverse
Set of Applications

Digital Side Channels and
some **Non-Digital** Side Channels

e.g. instruction-level
power consumption

Implementation

Close a Broad Class
of Side Channels

Execute on Modern
Microprocessors

Protect a Diverse
Set of Applications

Microarchitectural Implementations of
x64, **32-bit ARM**, and **64-bit ARM** ISAs

Implementation

Close a Broad Class
of Side Channels

Execute on Modern
Microprocessors

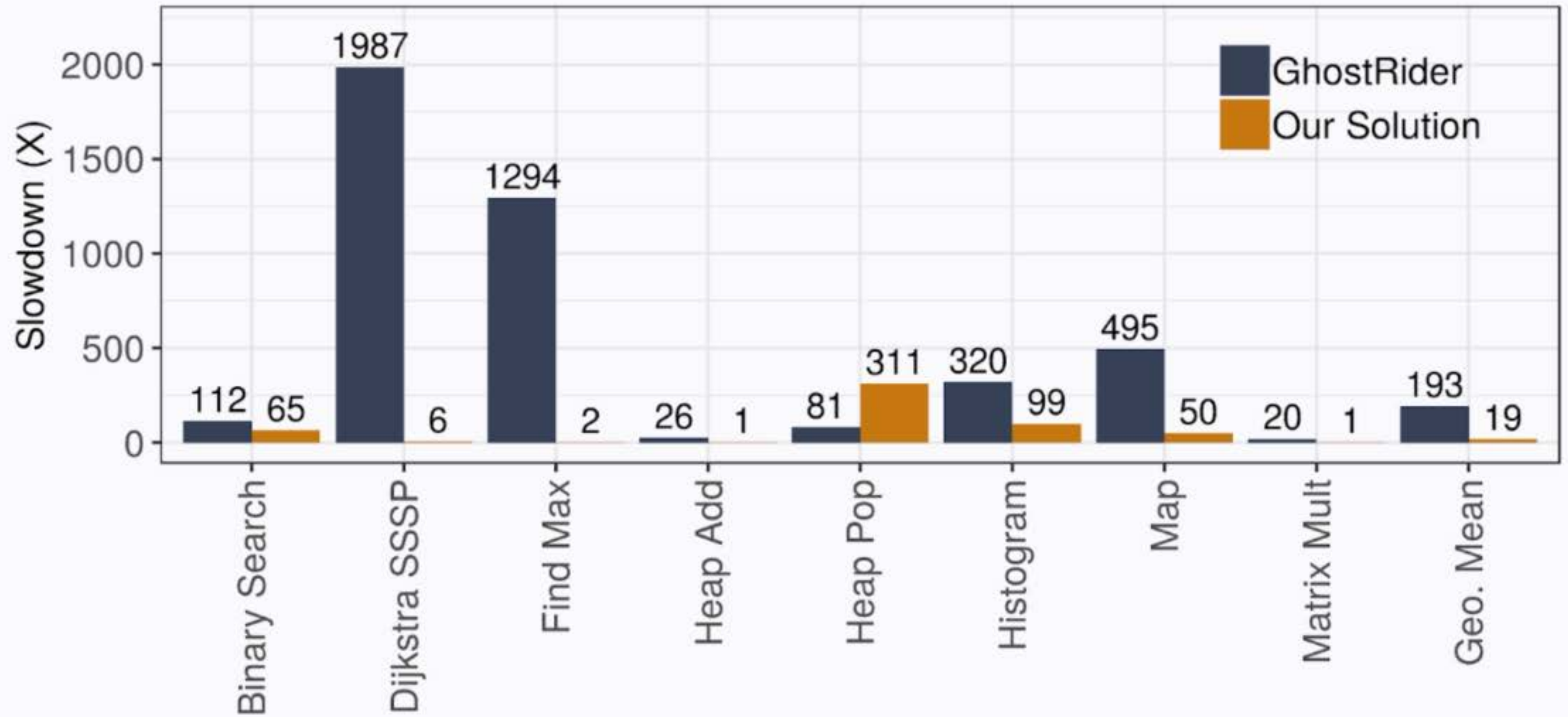
**Protect a Diverse
Set of Applications**

Not just Cryptographic Implementations, but also
Graph Kernels, Machine-Learning Libraries, etc.

Outline

- ▷ Our Solution's Design
- ▷ Core Principles that Enable Generalization
- ▶ Performance Comparison
- ▷ Future Work

Comparison With **GhostRider**



Evaluation Programs

Cryptography

- Lattice Crypto
- Curve-25519
- Poly-1305

GhostRider

- Binary Search
- Heap Add
- Matrix Mult

Graph Kernels

- Top-K Search
- Bellman Ford
- PageRank

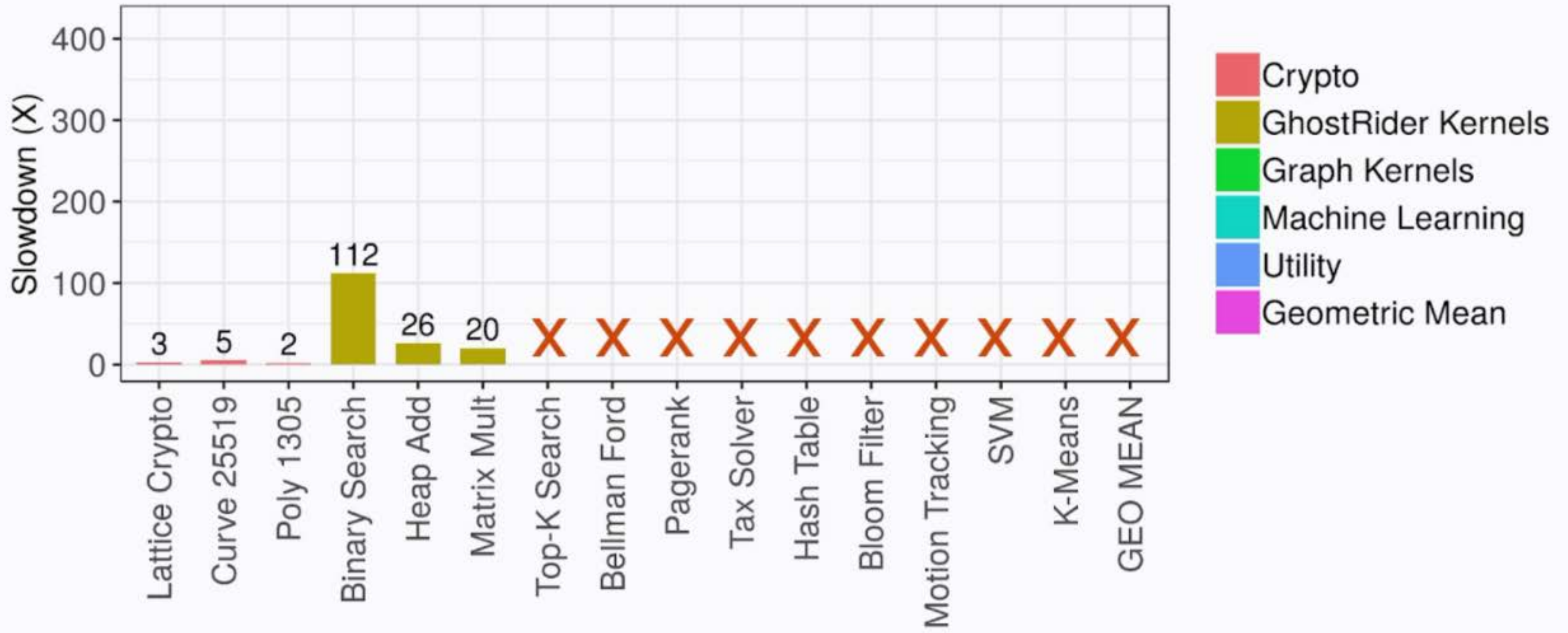
Machine Learning

- Motion Tracking
- SVM Classifier

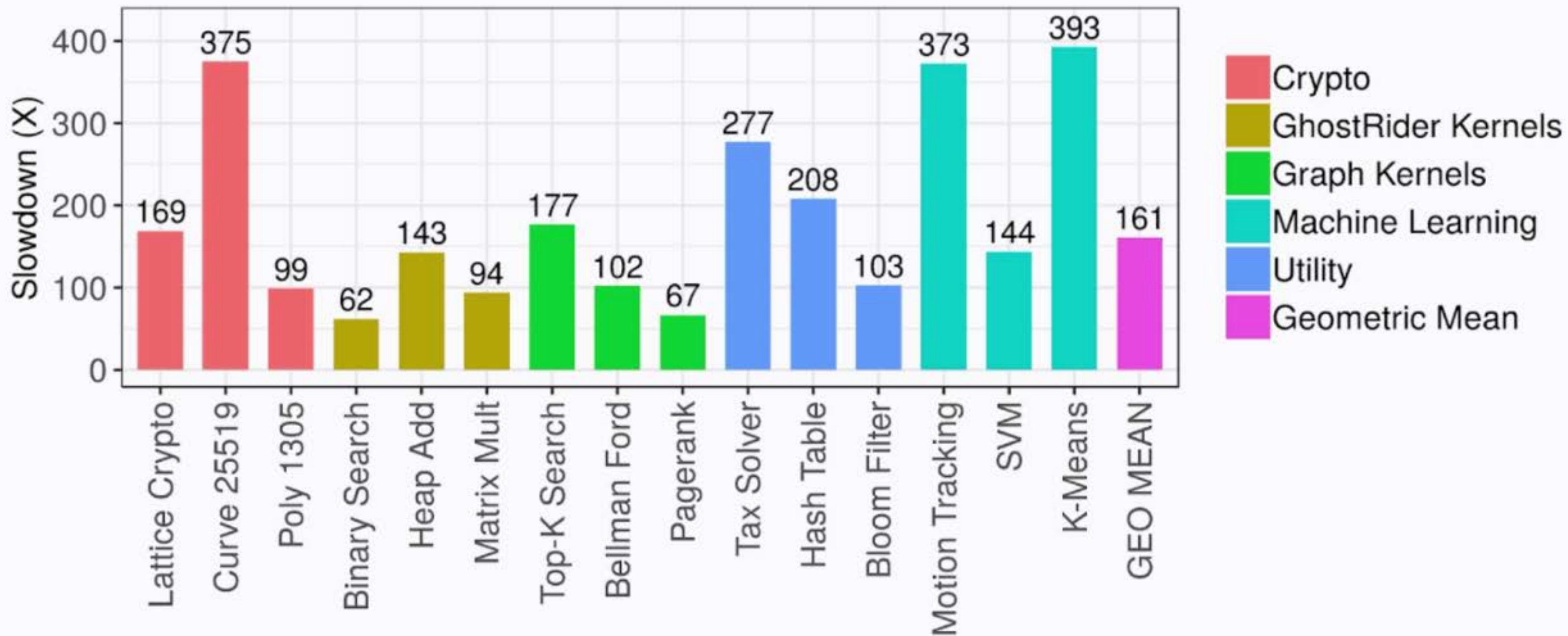
Utility Programs

- Font Renderer
- Hash Table
- Bloom Filter

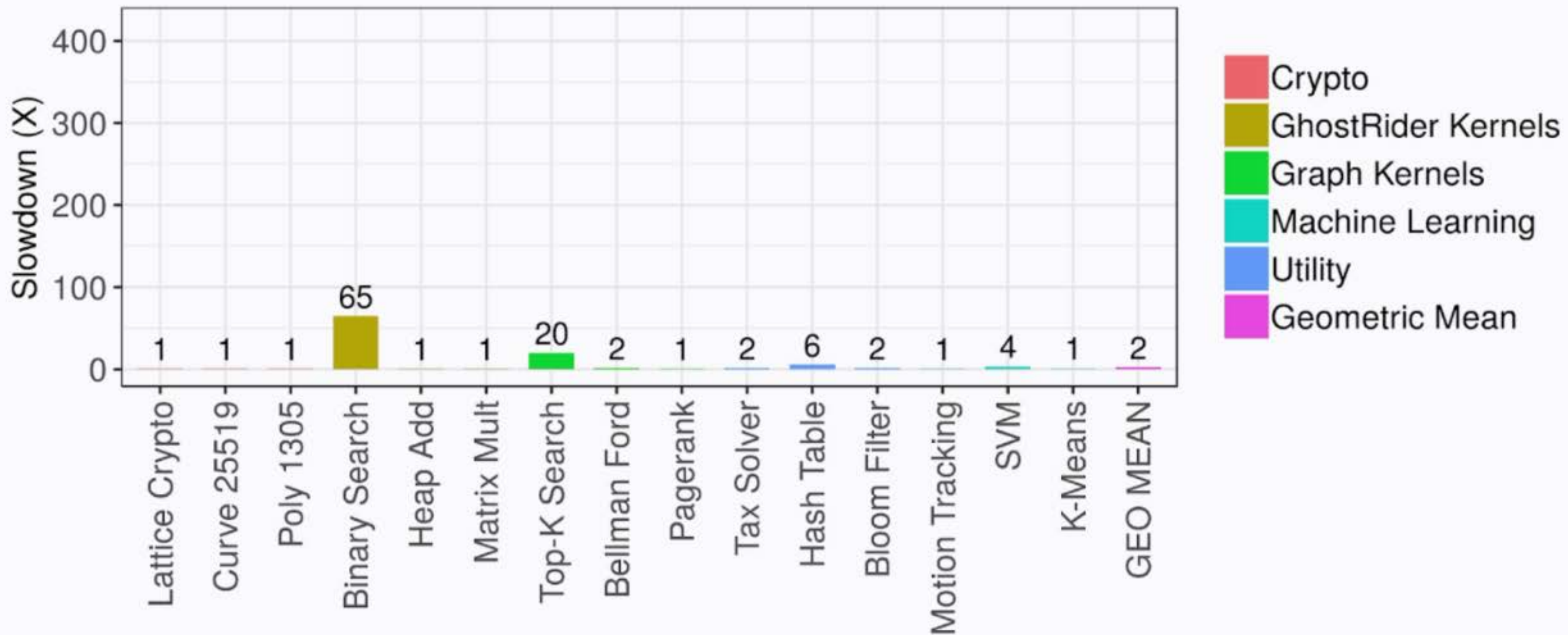
GhostRider Evaluation



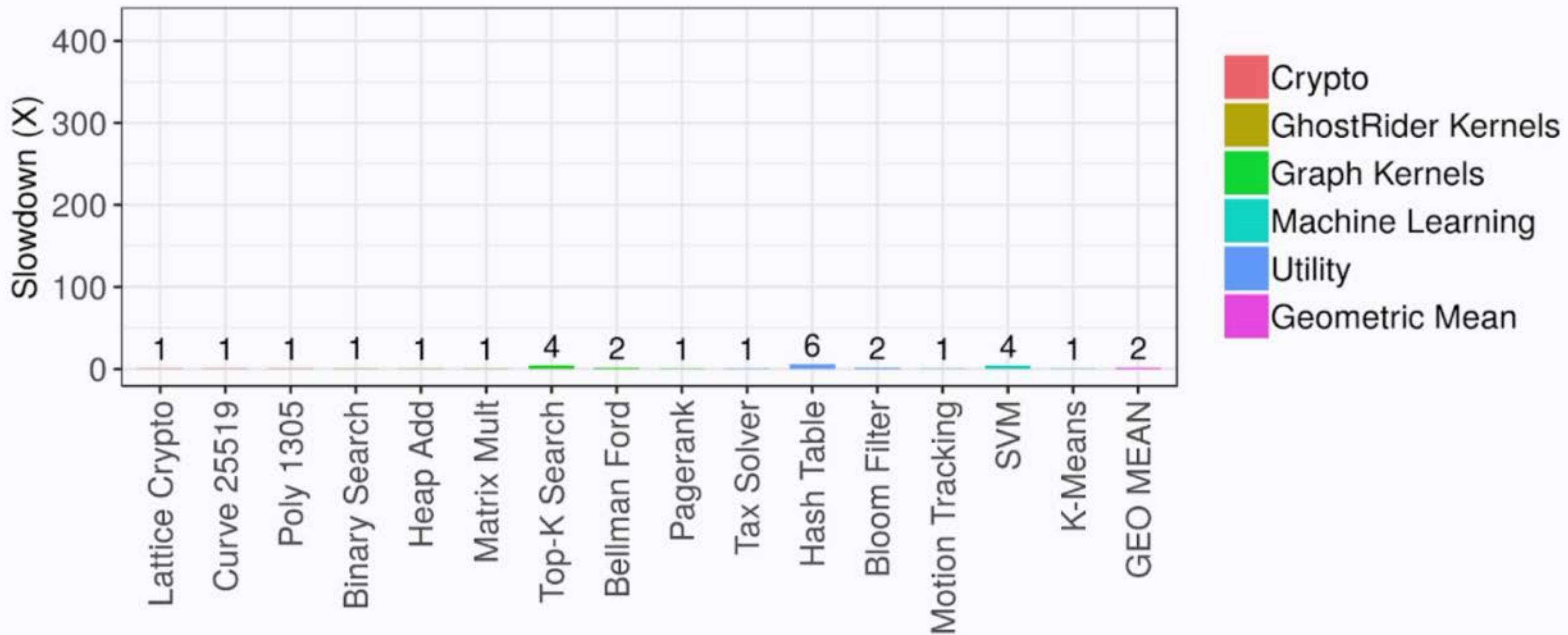
Hardware-Only Solution Evaluation



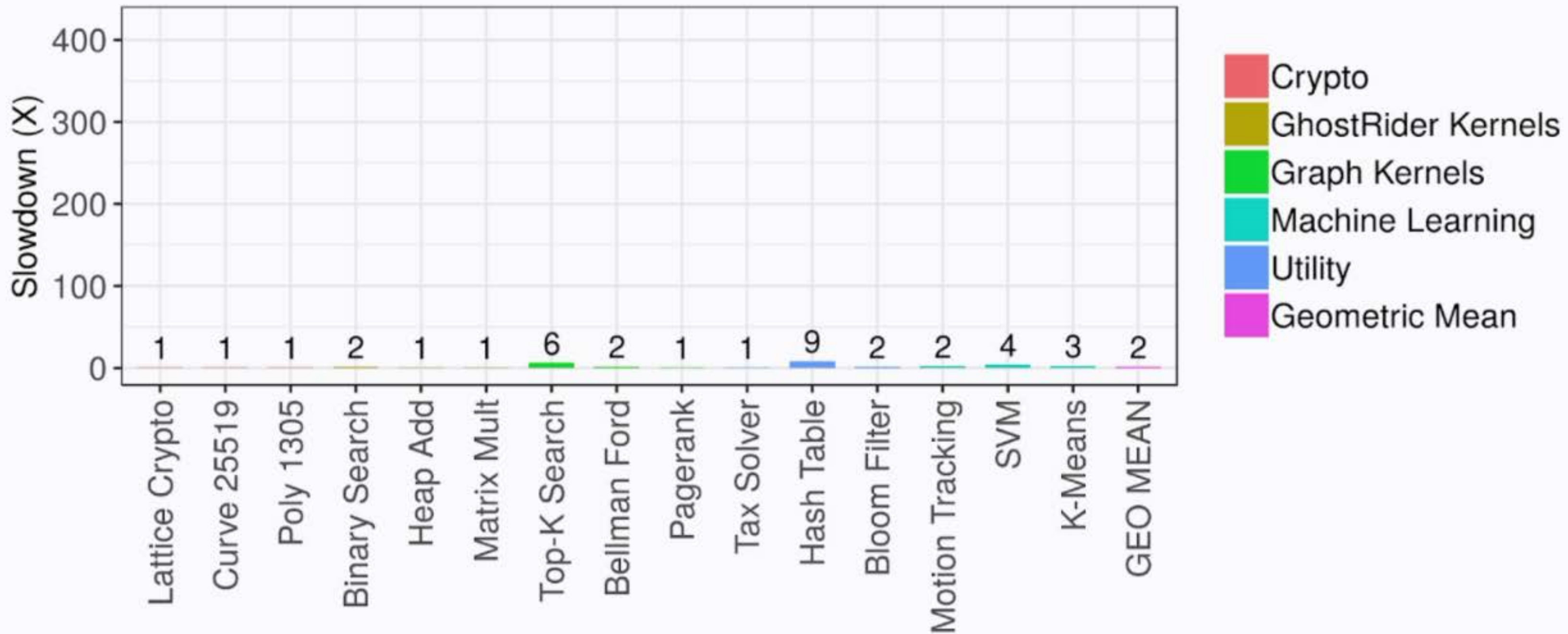
Our Solution: All Digital Side Channels



Our Solution: **Timing** Side Channel



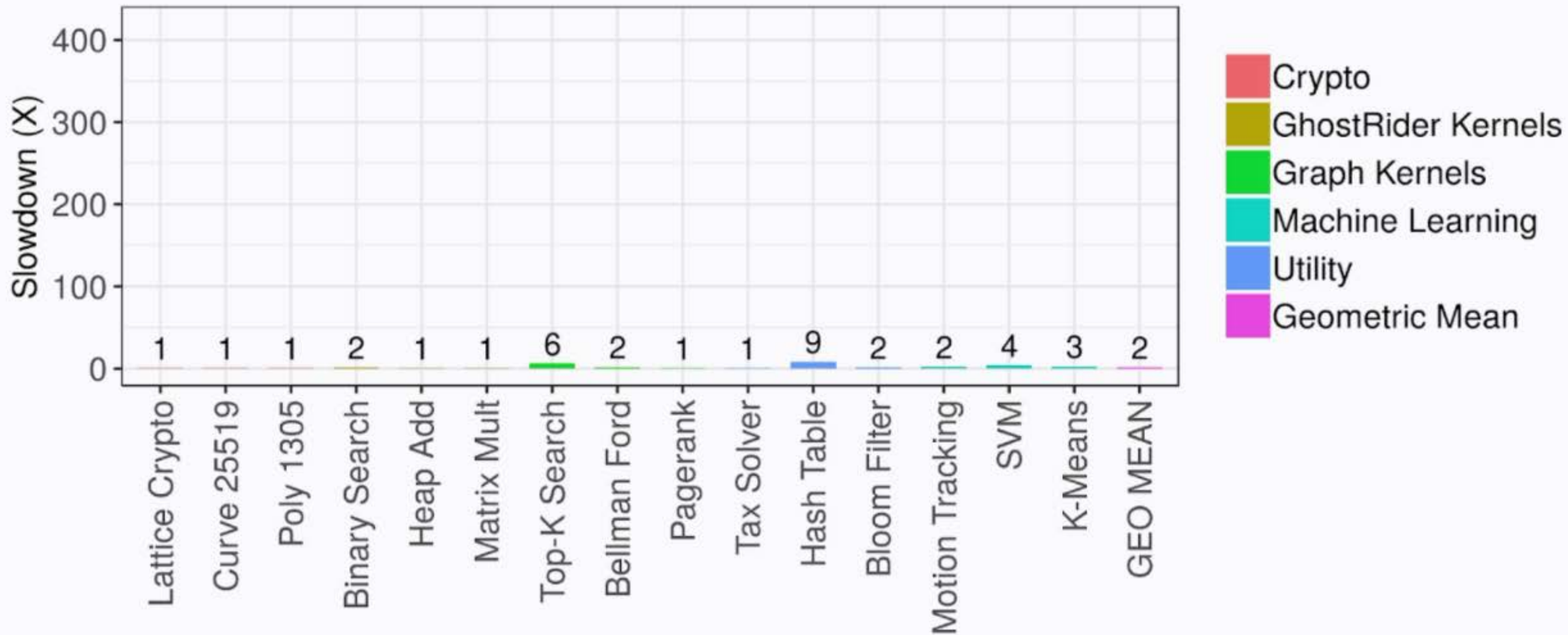
Our Solution: **Microarch. Power** Side Channel



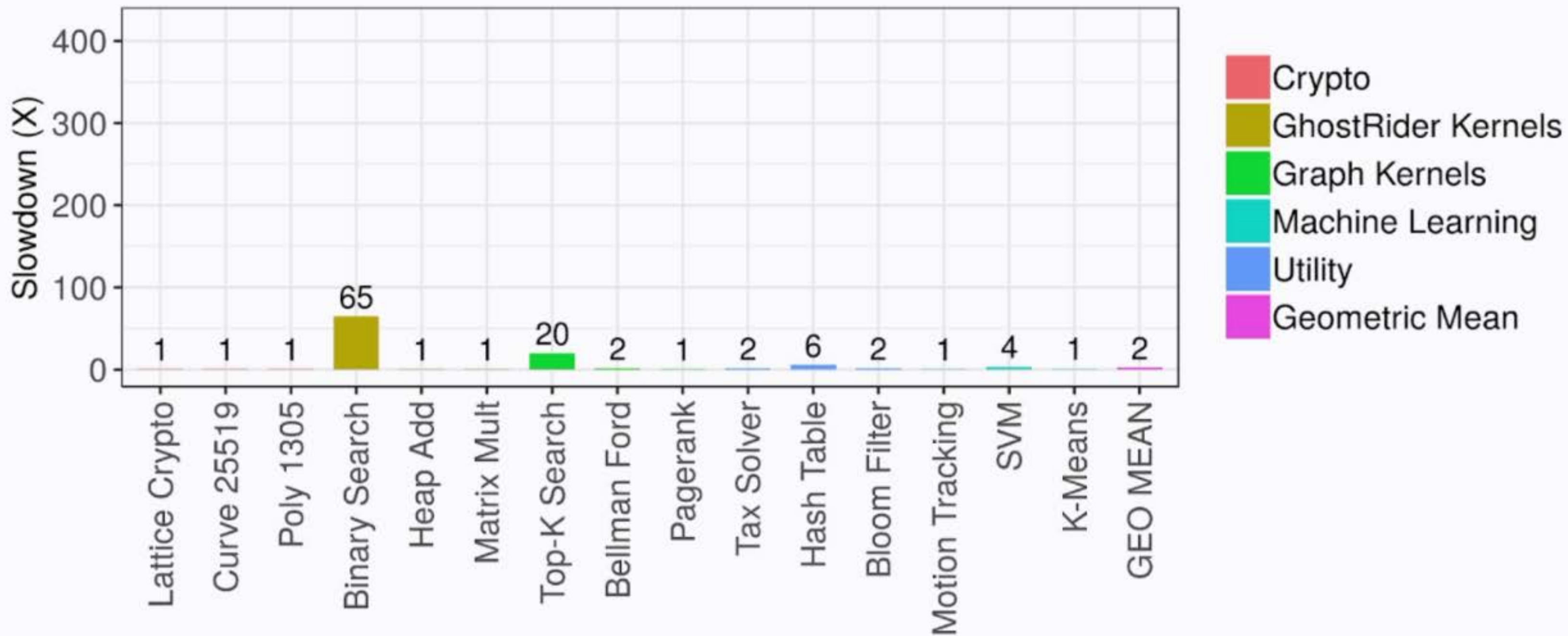
Outline

- ▷ Our Solution's Design
- ▷ Core Principles that Enable Generalization
- ▷ Performance Comparison
- ▶ Future Work

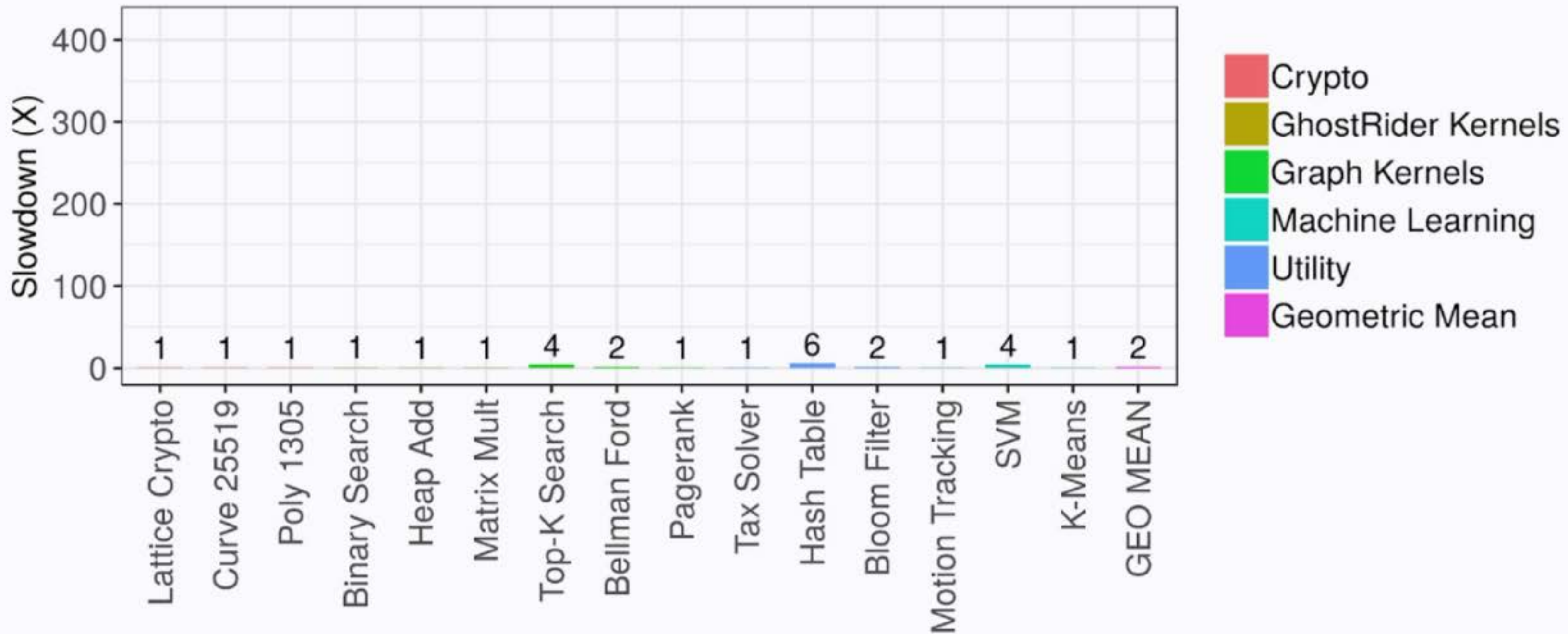
Our Solution: **Microarch. Power** Side Channel



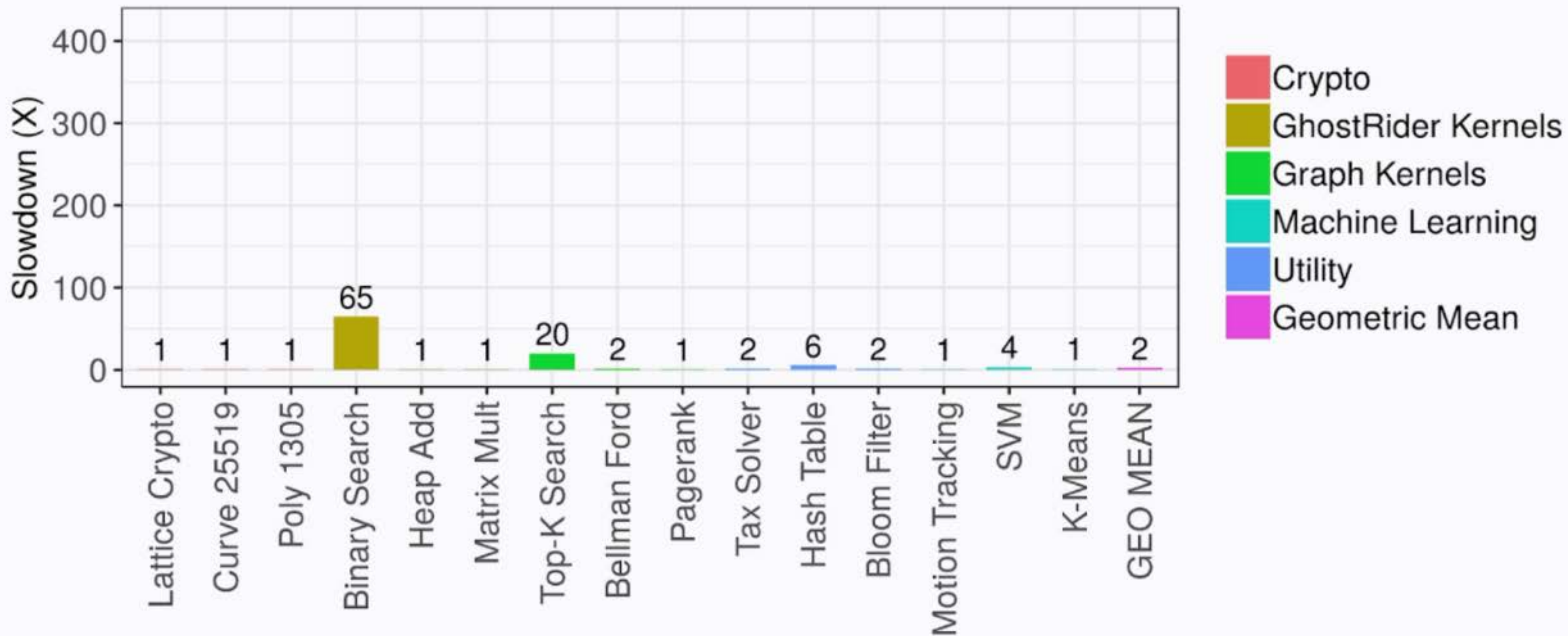
Our Solution: All Digital Side Channels



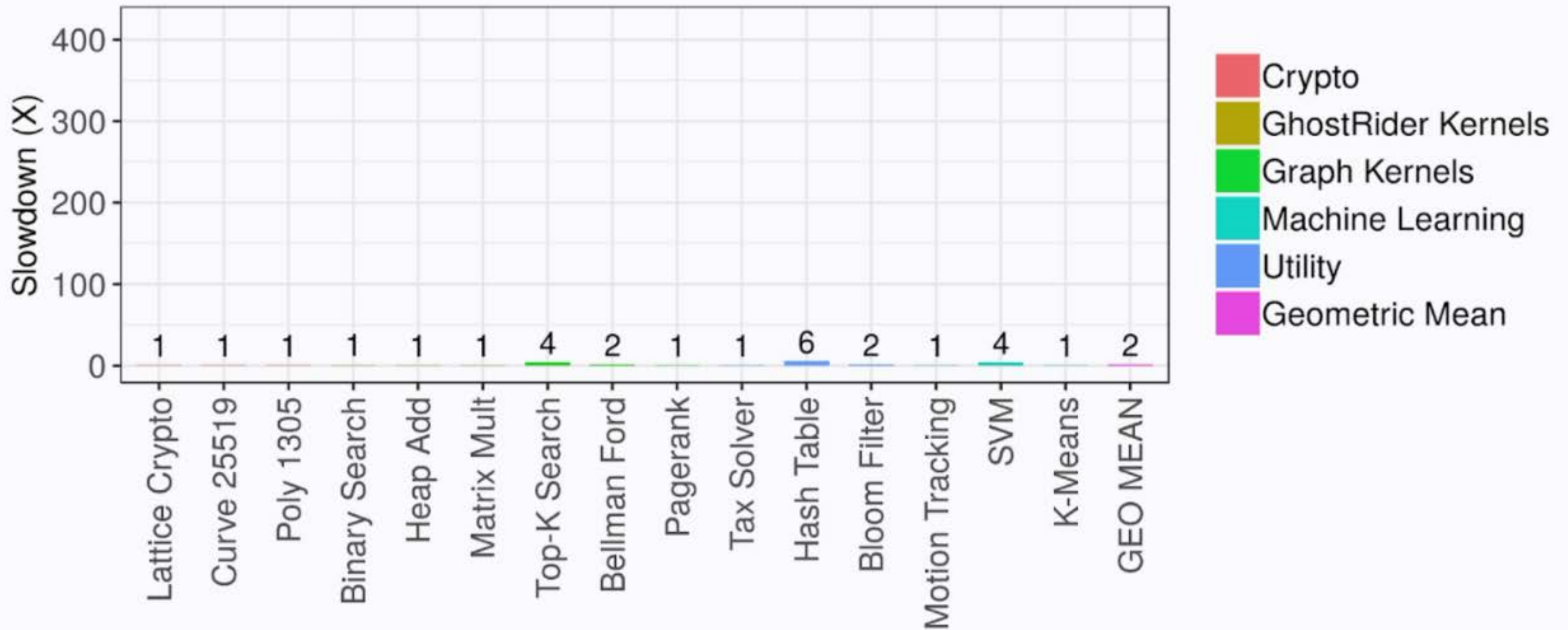
Our Solution: **Timing** Side Channel



Our Solution: All Digital Side Channels



Our Solution: **Timing** Side Channel



Outline

- ▷ Our Solution's Design
- ▷ Core Principles that Enable Generalization
- ▷ Performance Comparison
- ▶ Future Work

Future Work in Side-Channel Defenses

Automation

Synthesizing program transformations

Precision

Integrating better models of information leakage

Performance

Aggressive compiler optimizations and modest microarch. modifications





such that f' does not leak information through side channels

Synthesizing Side-Channel Defenses



such that f' does not leak information through side channels

Synthesizing Side-Channel Defenses



such that f' does not leak information through side channels

Goal: Adapt research in superoptimizers and program synthesis

Synthesis for **Stronger Guarantees**

Microarchitectural Specification



Domain-Specific Language
for Compiler Transformations



Executable Code for Program Transformations

Synthesis for **Stronger Guarantees**

Microarchitectural Specification



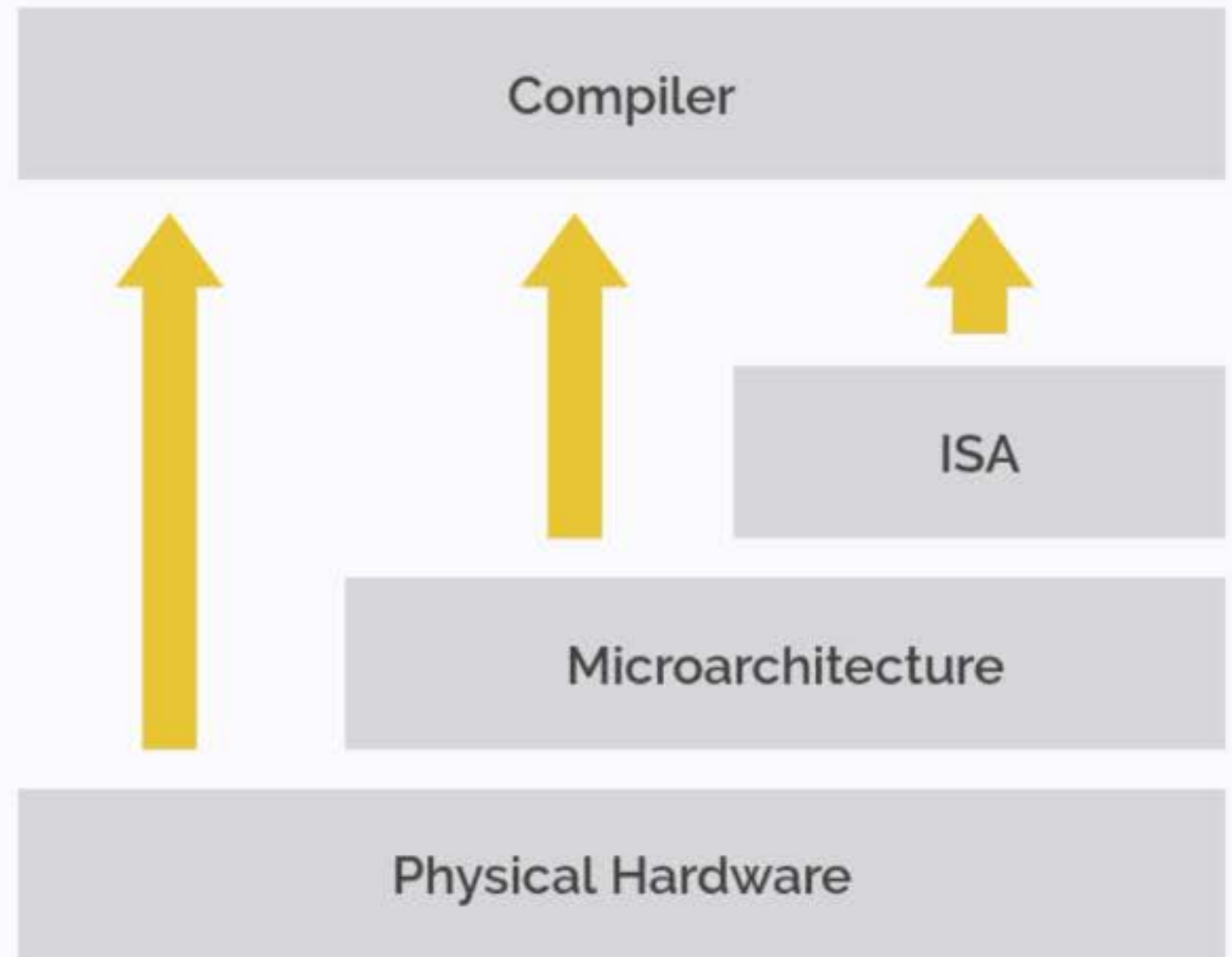
Domain-Specific Language
for Compiler Transformations



+ Type Analysis
+ Solver-Based Verification

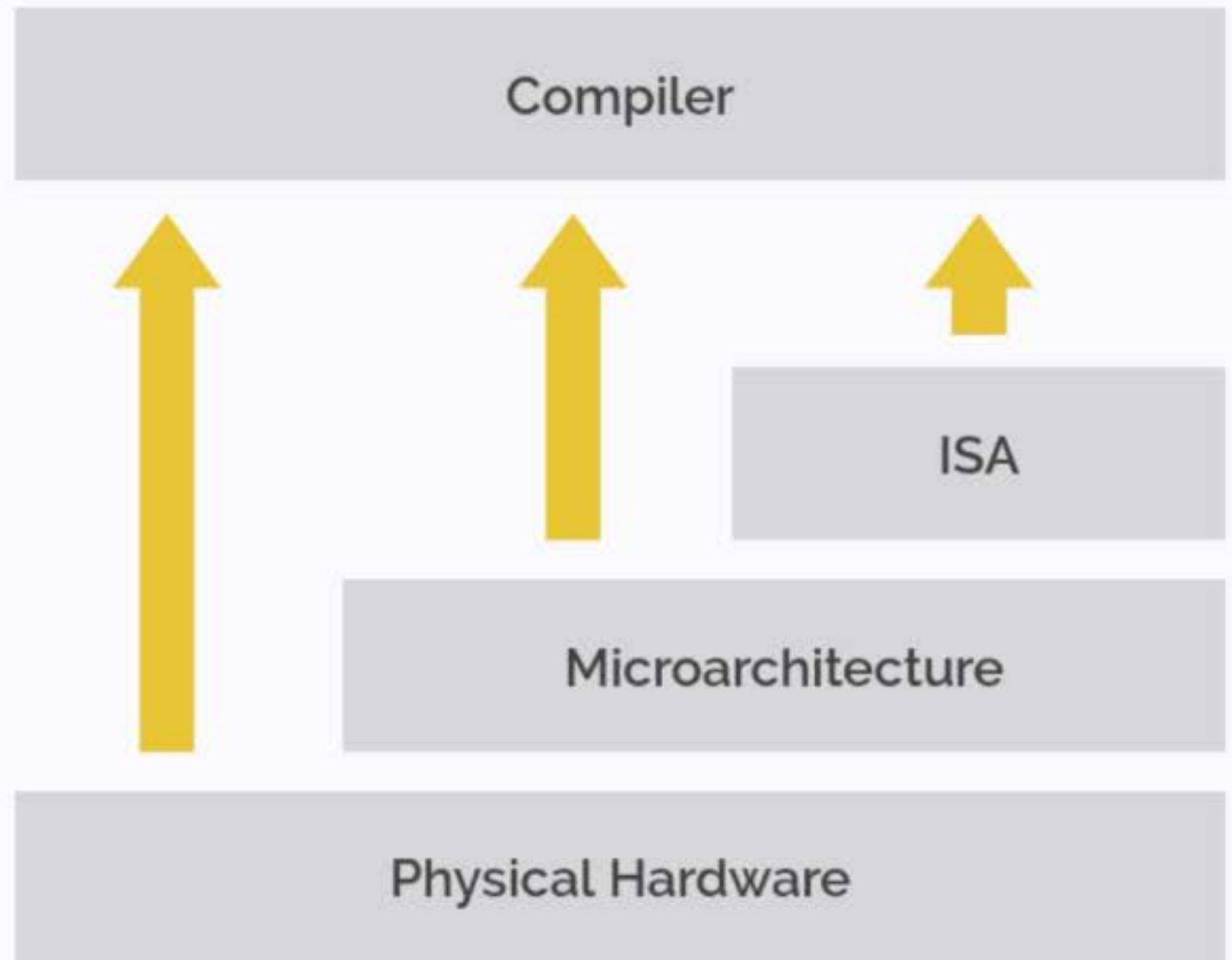
Executable Code for Program Transformations

Precision of Side-Channel Defenses



Precision of Side-Channel Defenses

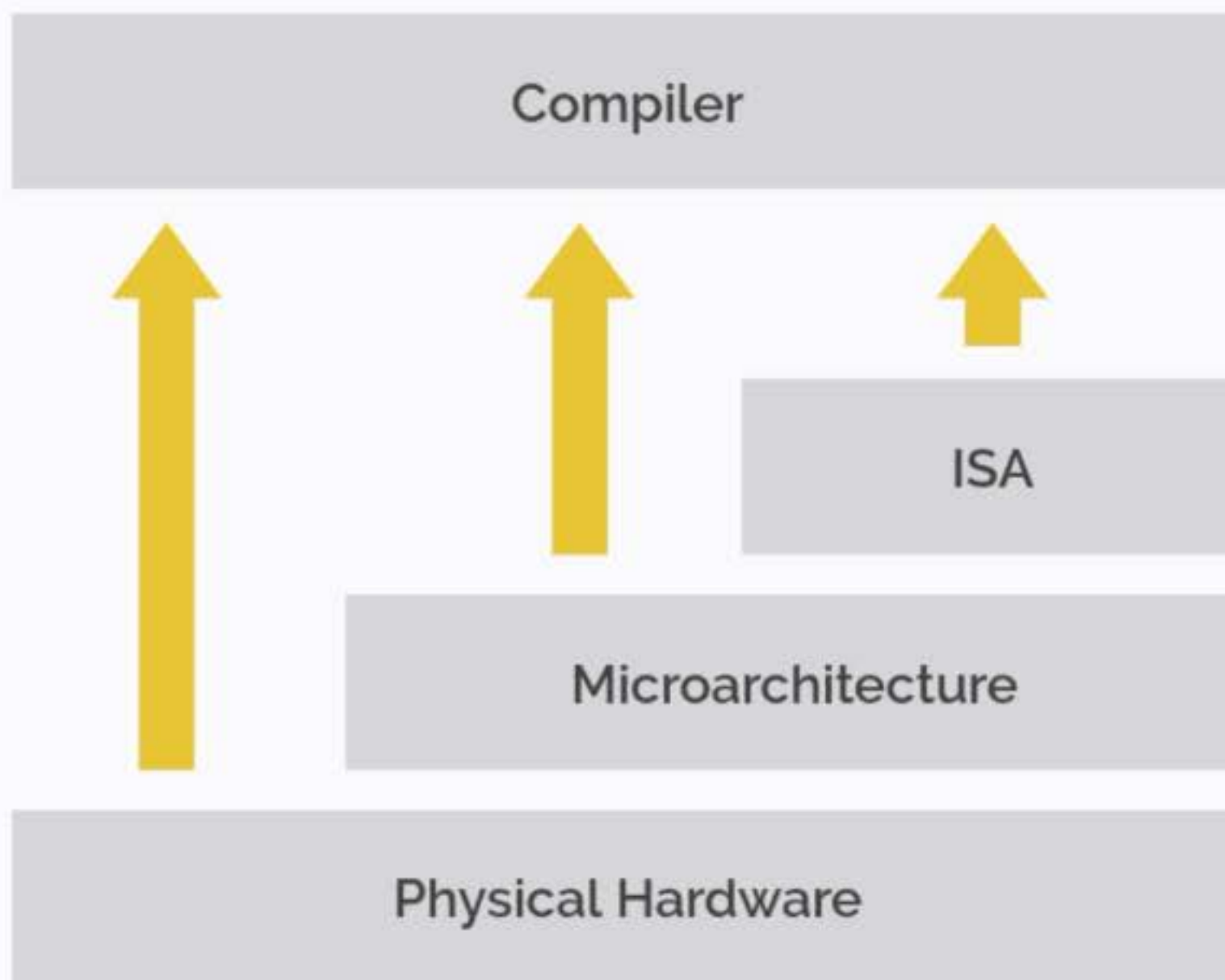
We need to tell compilers about potential side channels.



Precision of Side-Channel Defenses

We need to tell compilers about potential side channels.

Our current approach is an **ad-hoc mix** of program analysis, statistics, and manual inspection.

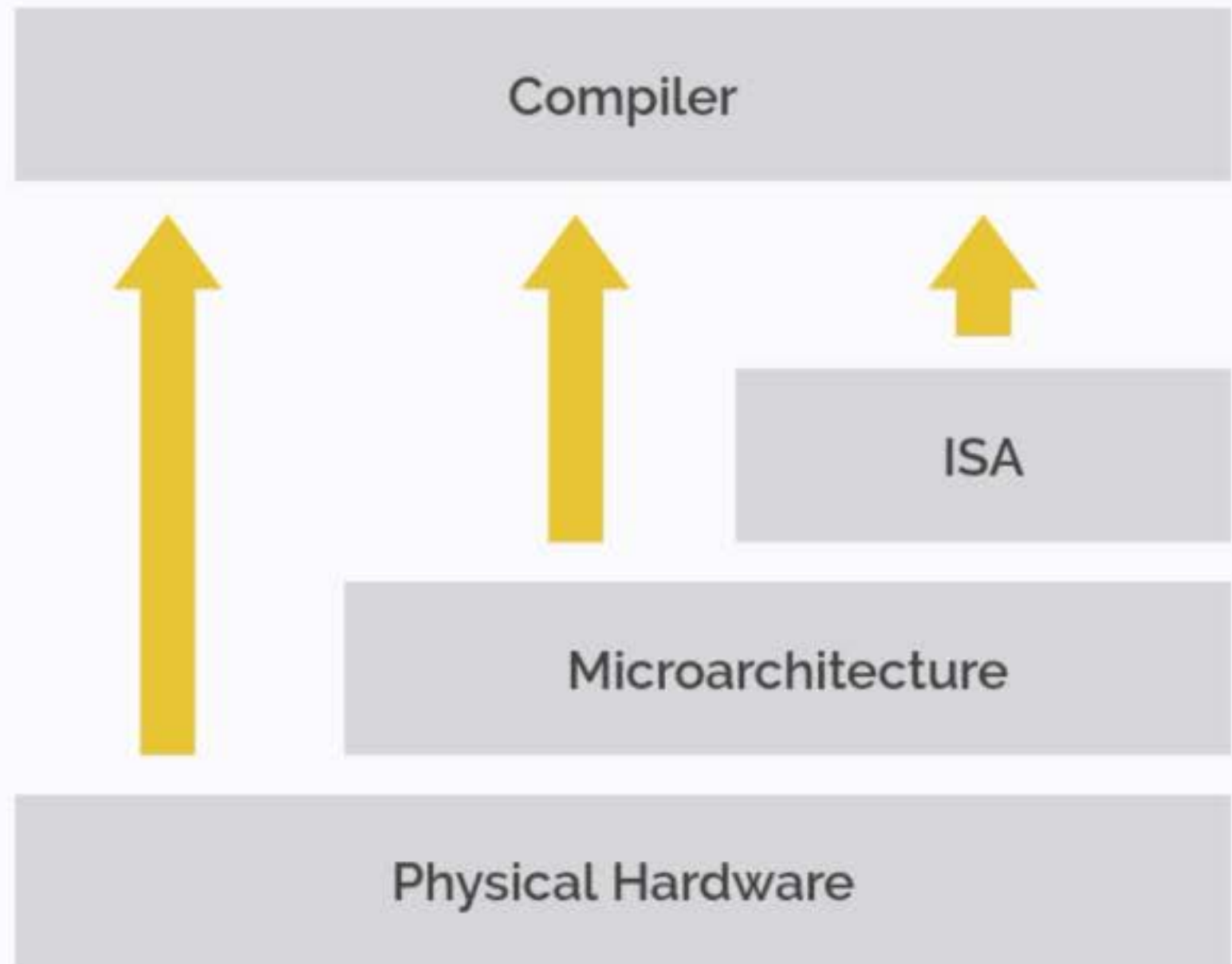


Precision of Side-Channel Defenses

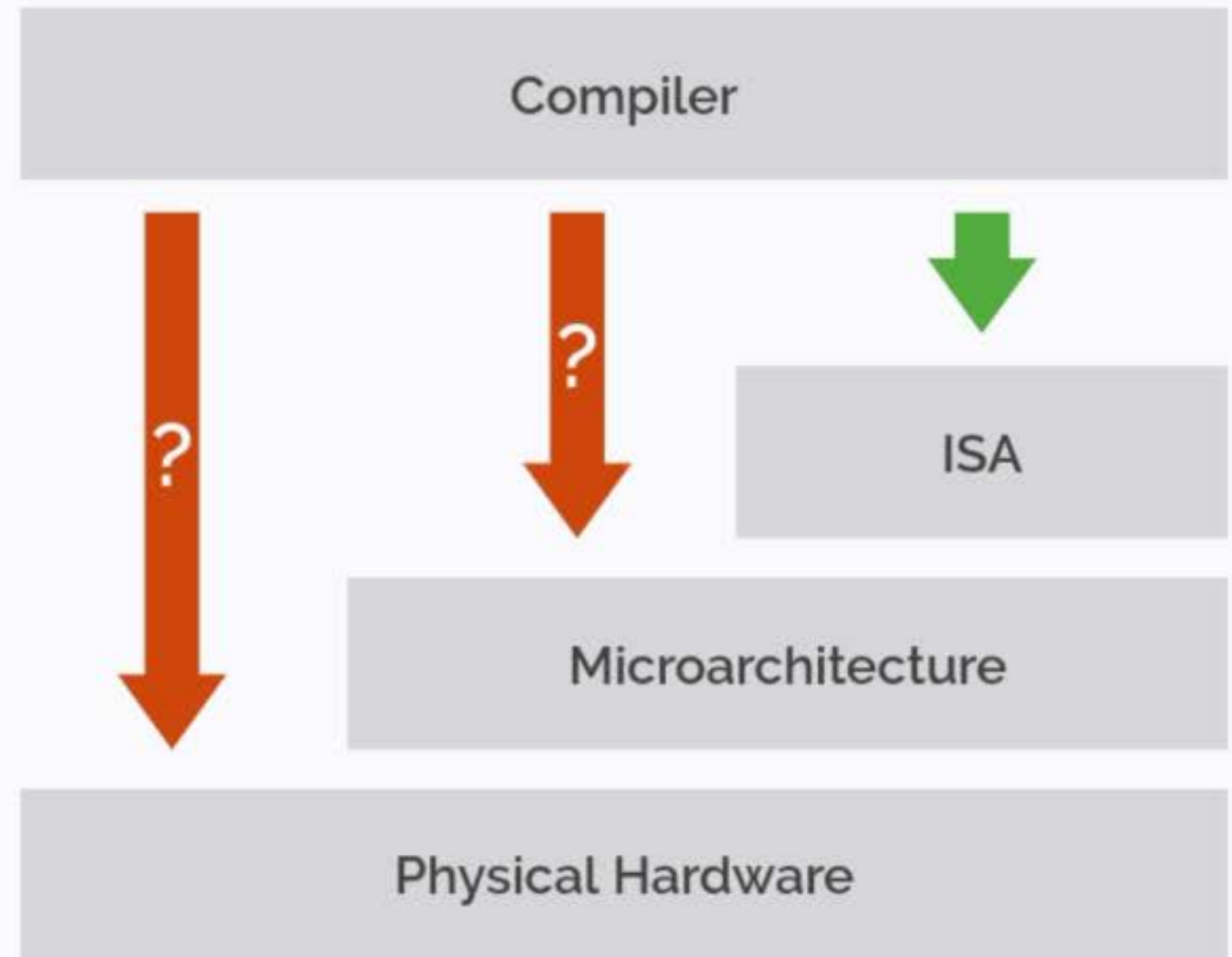
We need to tell compilers about potential side channels.

Our current approach is an **ad-hoc mix** of program analysis, statistics, and manual inspection.

Goal: **Precise** abstractions of underlying layers.

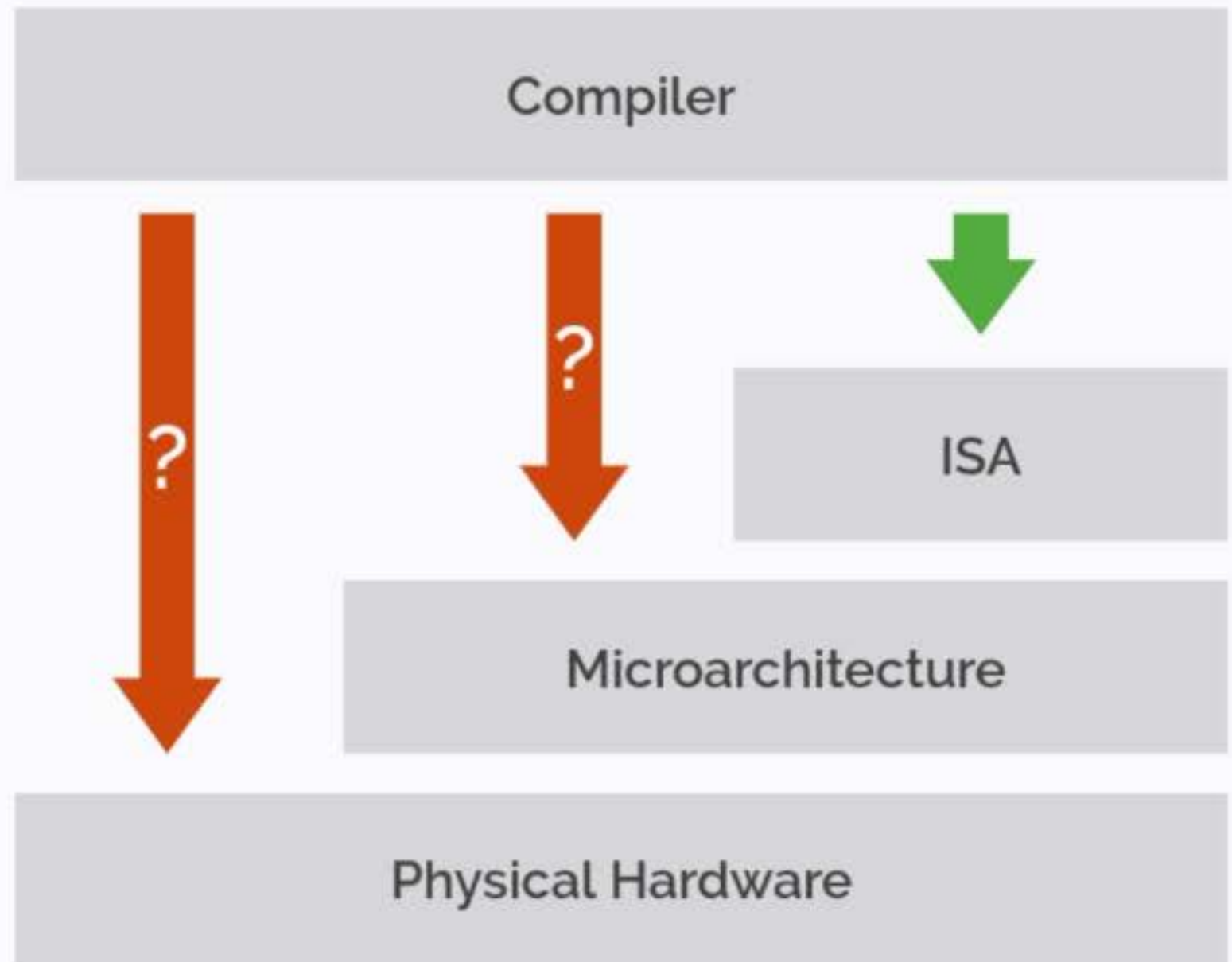


Performance of Side-Channel Defenses



Performance of Side-Channel Defenses

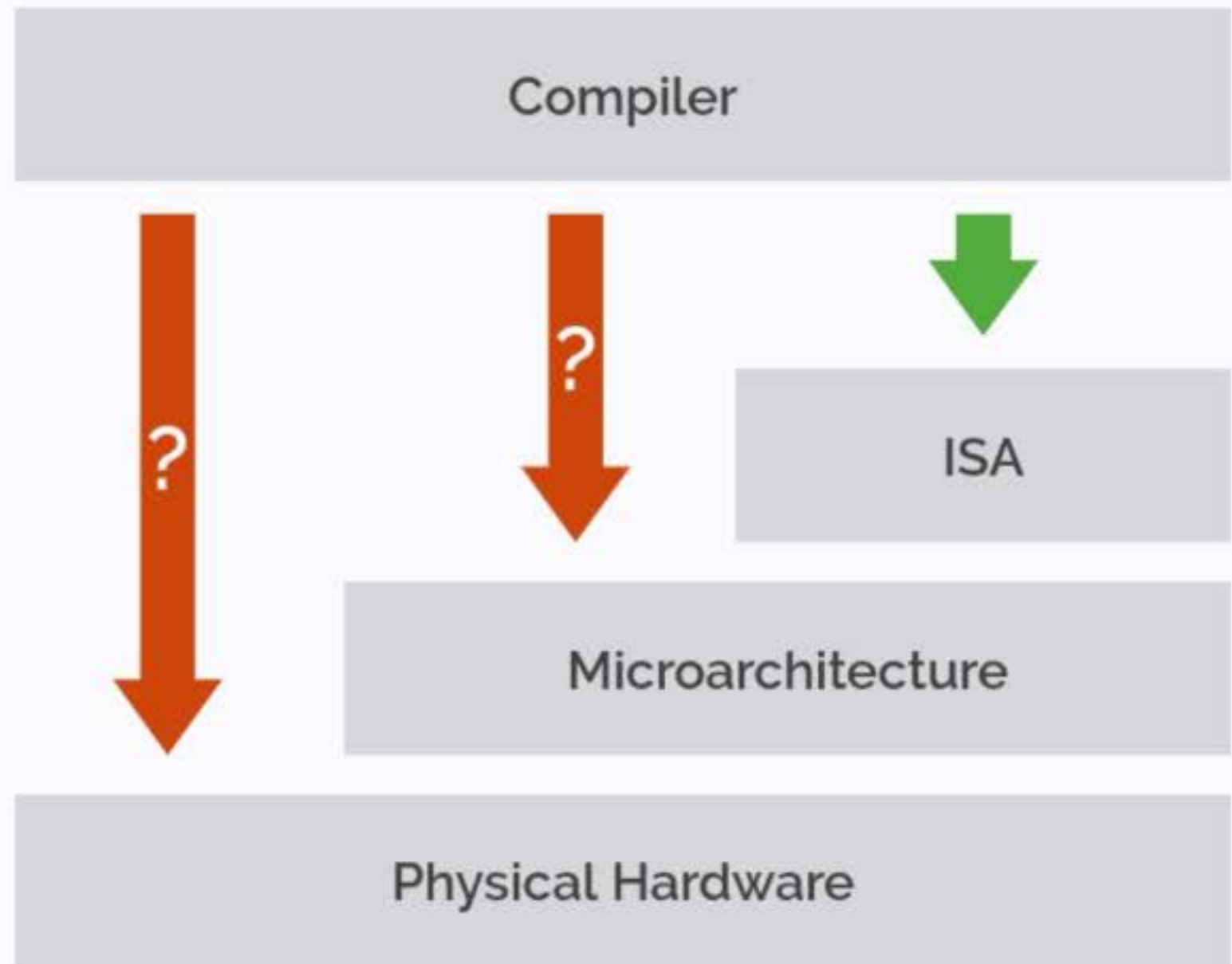
Compilers currently are at the mercy of the ISA.



Performance of Side-Channel Defenses

Compilers currently are at the mercy of the ISA.

We need more control of the microarchitecture and hardware.

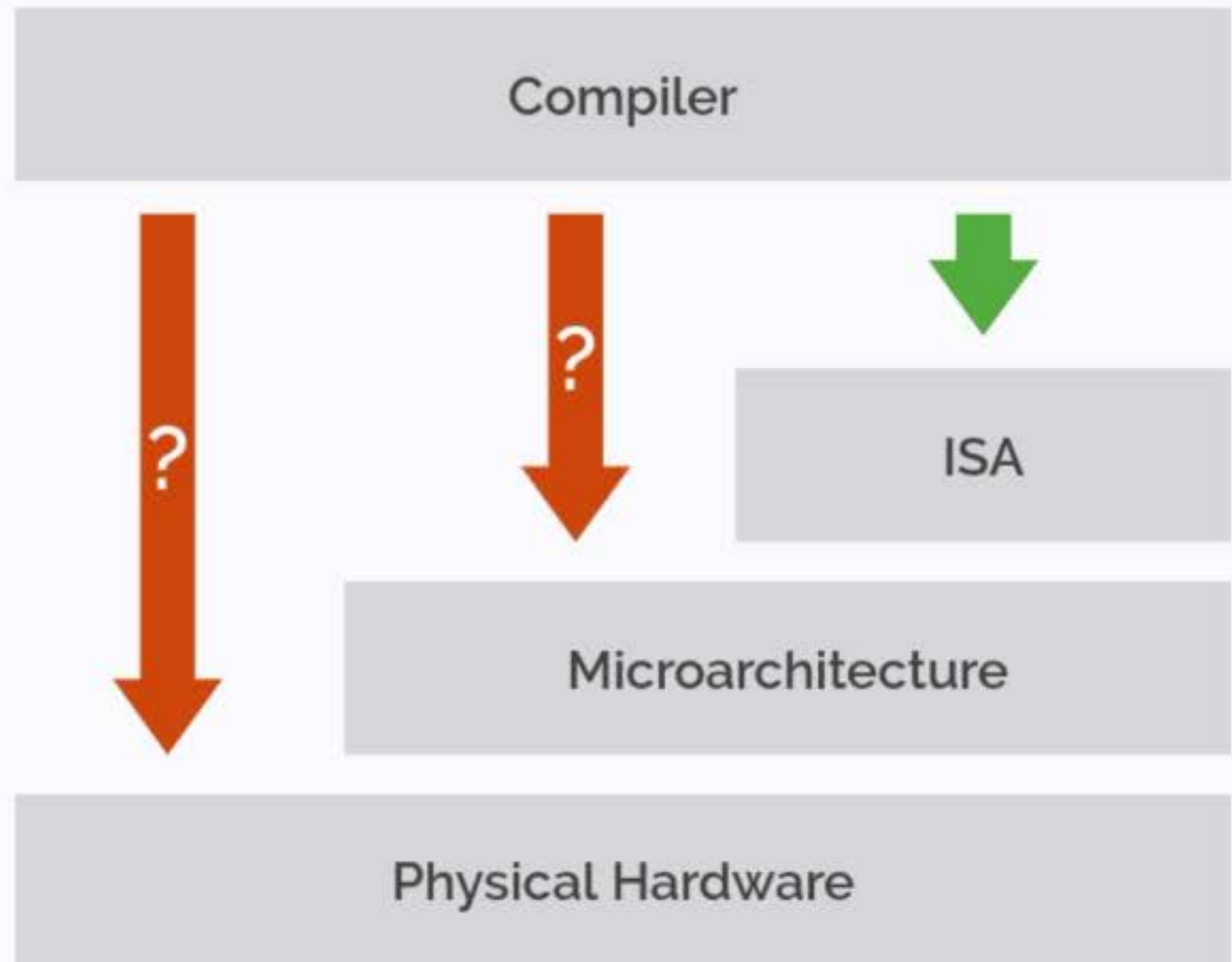


Performance of Side-Channel Defenses

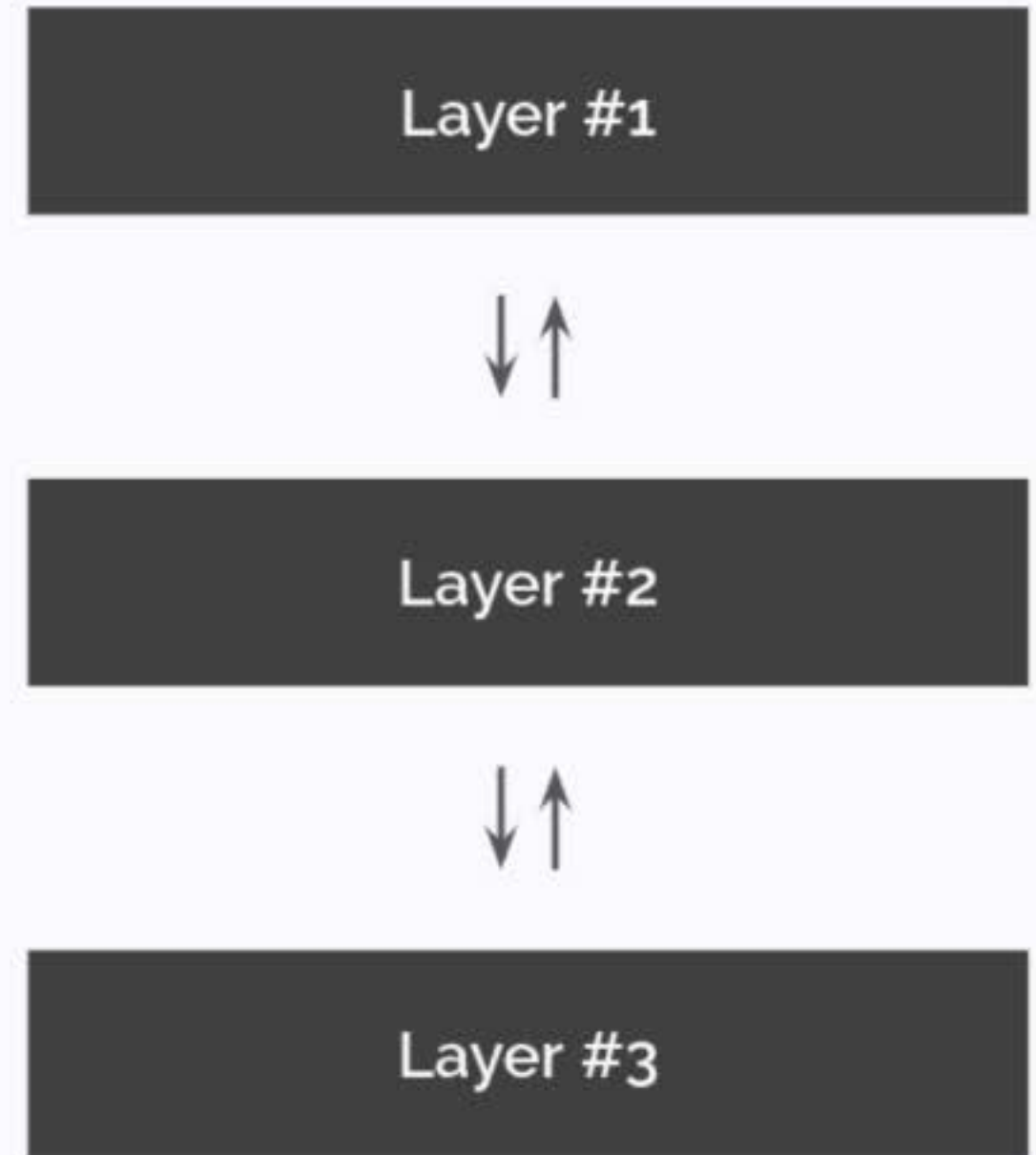
Compilers currently are at the mercy of the ISA.

We need more control of the microarchitecture and hardware.

Goal: Broaden the definition of the ISA beyond just a functional interface.

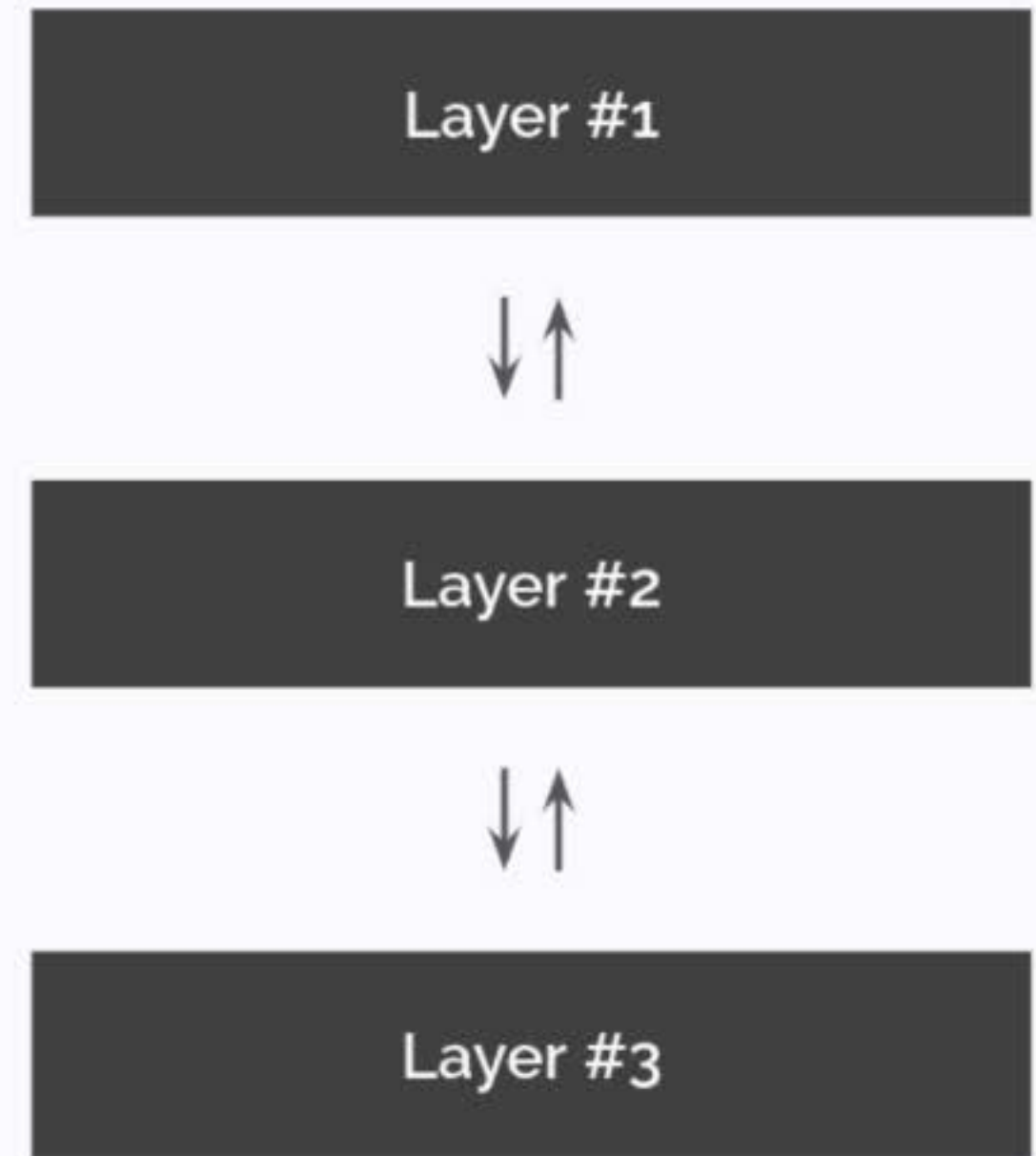


Layers of Abstraction as a Liability



Layers of Abstraction as a Liability

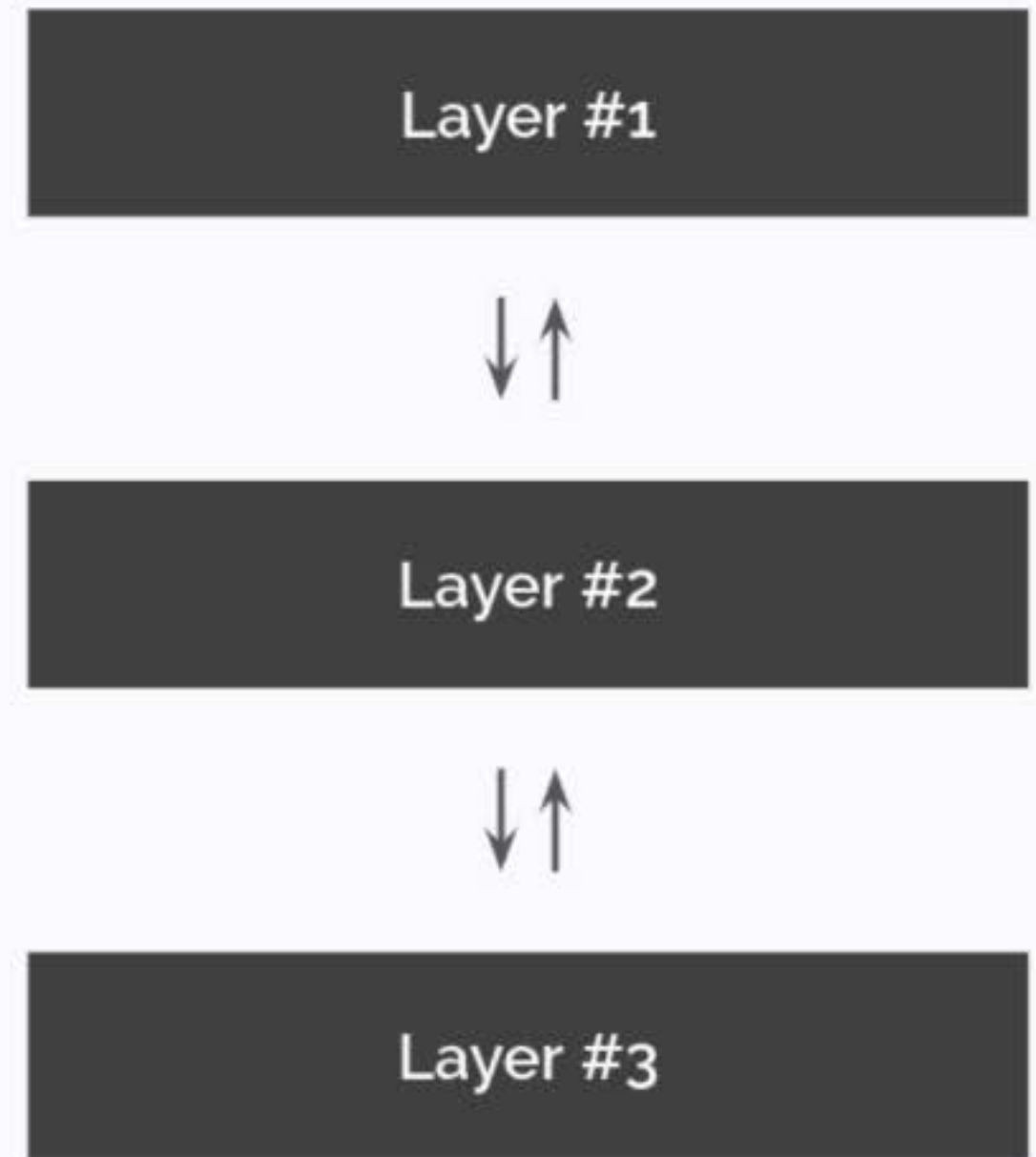
Debugging for security sometimes requires knowing a little about the implementation.



Layers of Abstraction as a Liability

Debugging for security sometimes requires knowing a little about the implementation.

But abstractions explicitly disable peeking into the implementation!

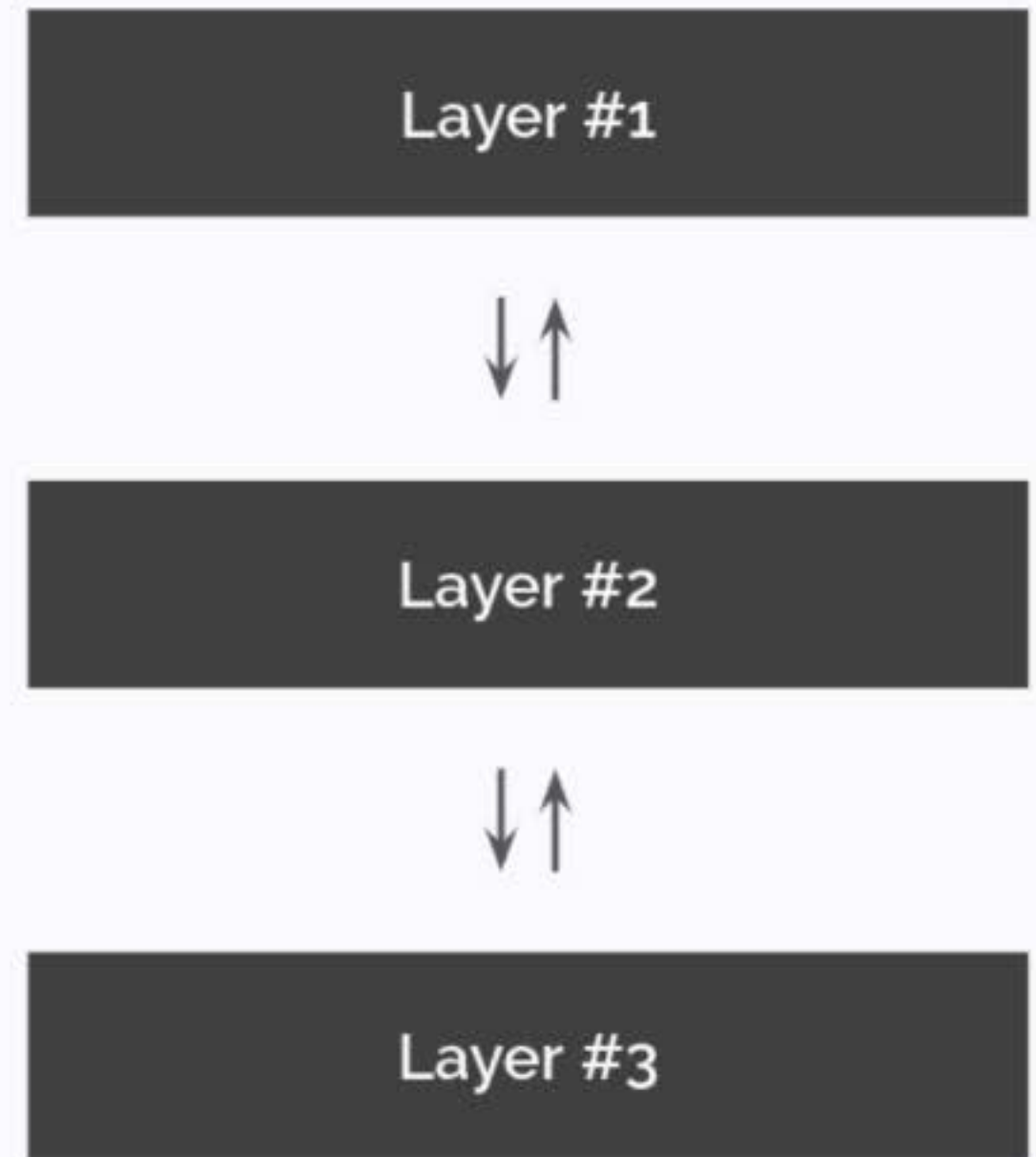


Layers of Abstraction as a Liability

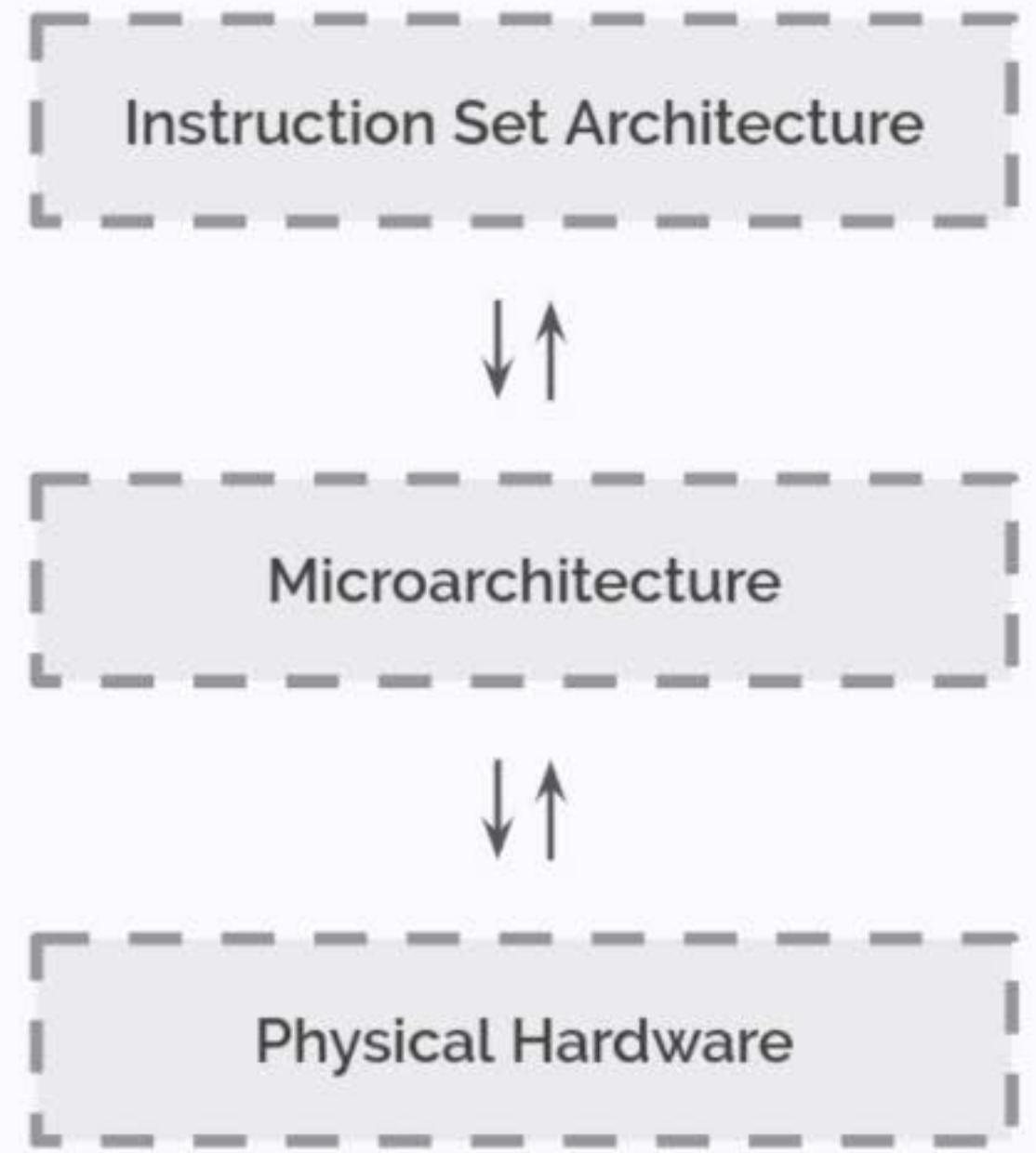
Debugging for security sometimes requires knowing a little about the implementation.

But abstractions explicitly disable peeking into the implementation!

This problem affects performance debugging as well.



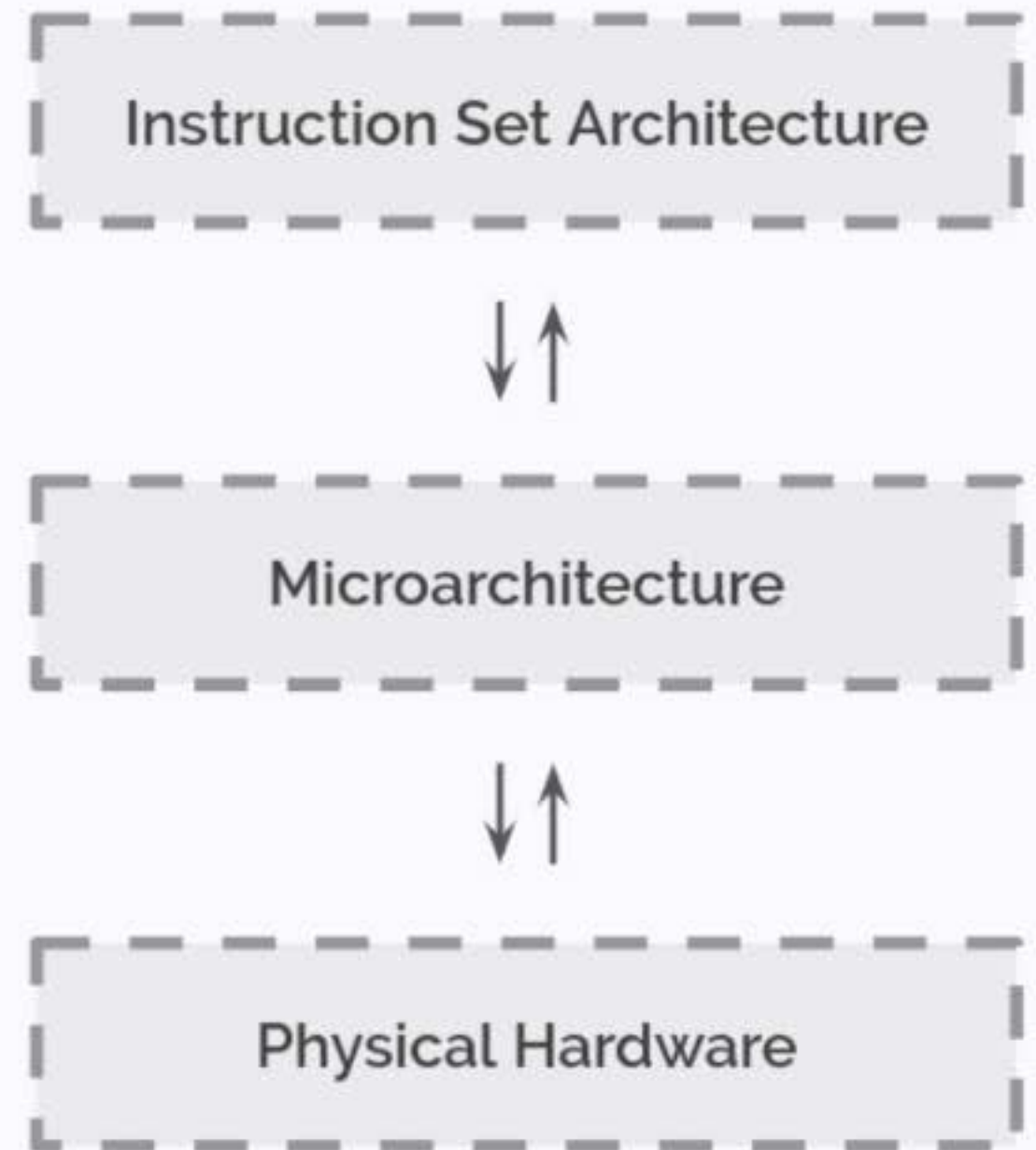
Peeking Inside the Hardware Implementation



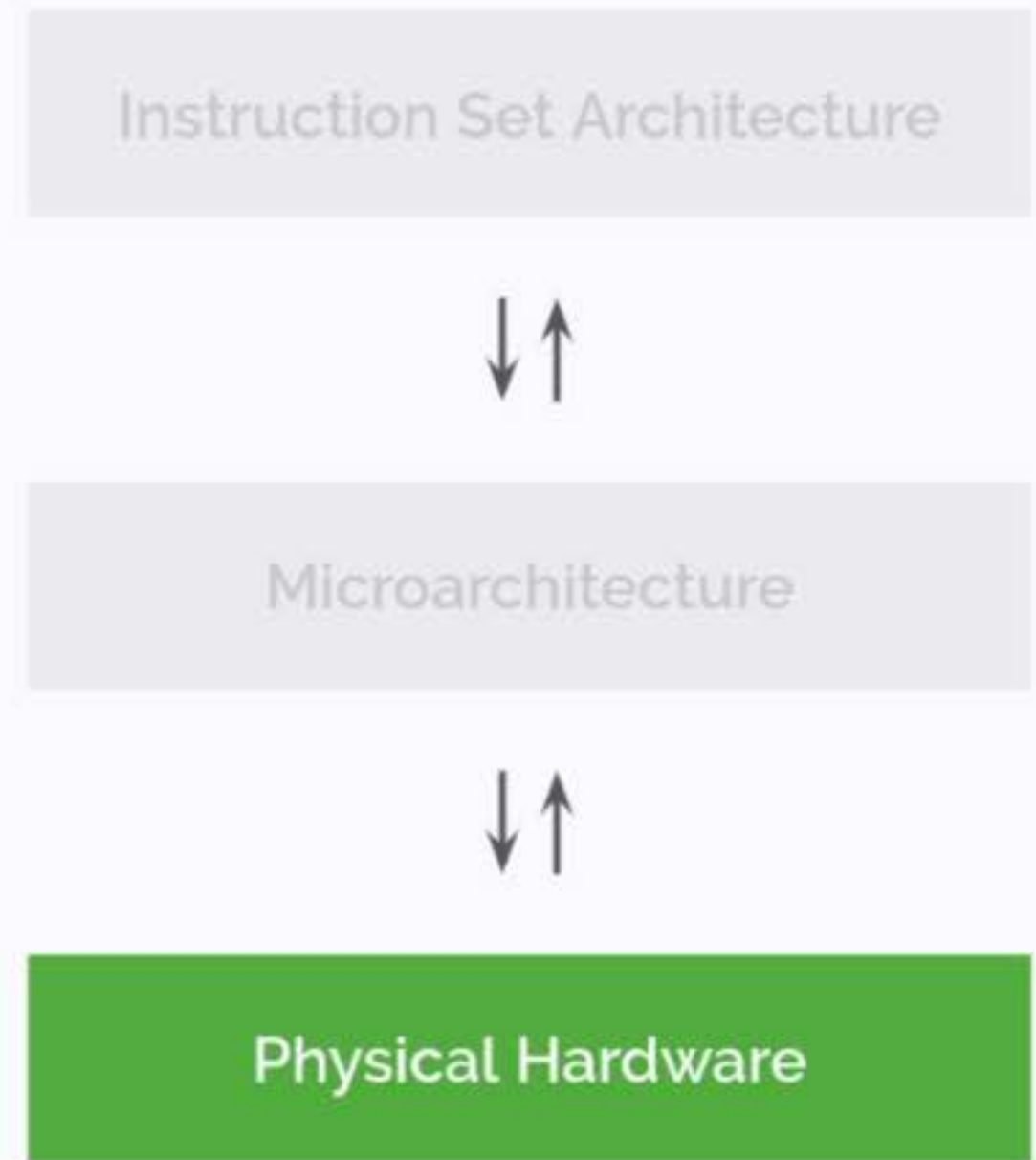
Peeking Inside the Hardware Implementation

Can assist in:

- Discovering Denial-of-Service attacks
- Locating Confused-Deputy problems
- Tuning performance and energy consumption



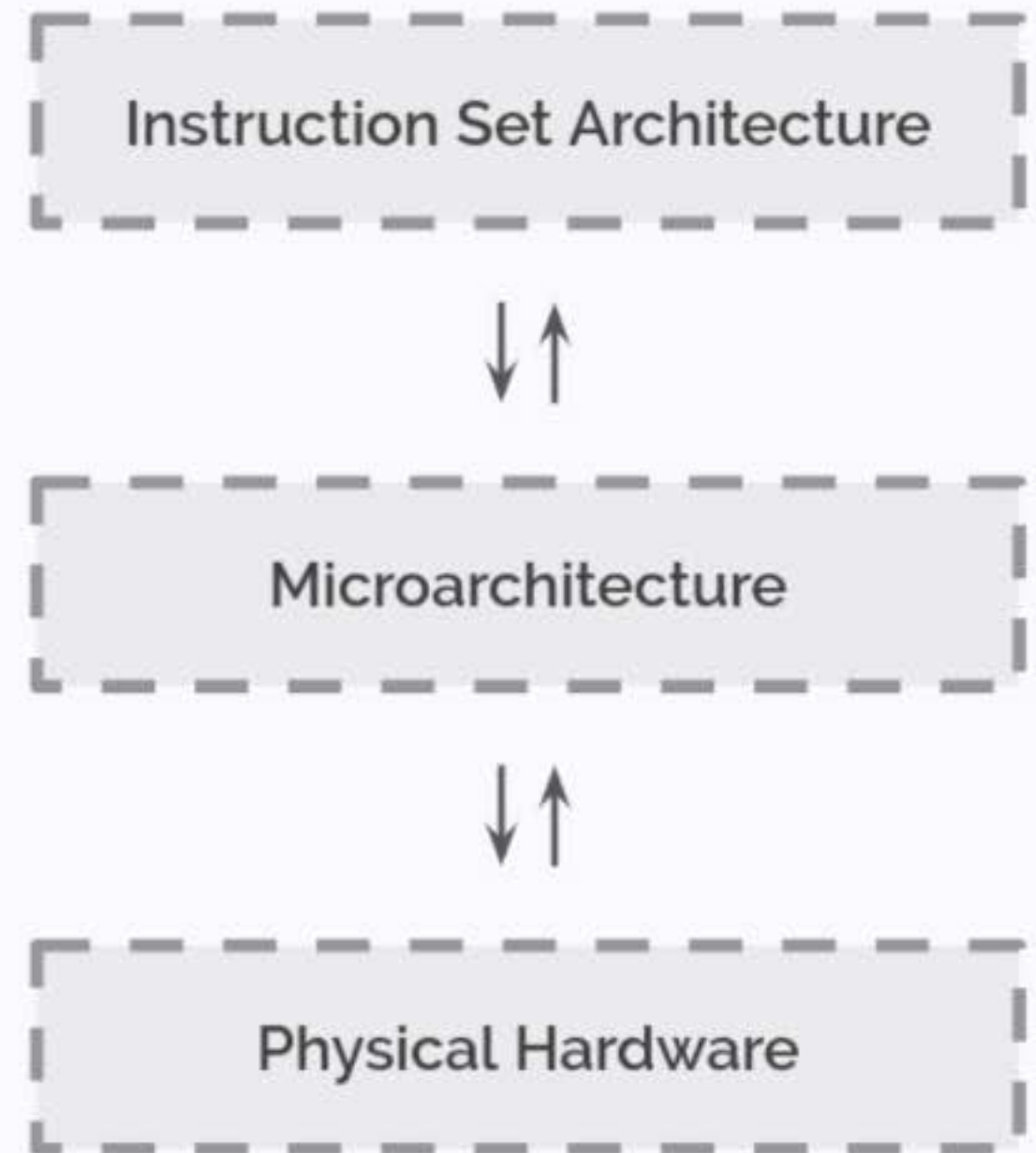
Program Analysis for Hardware Design



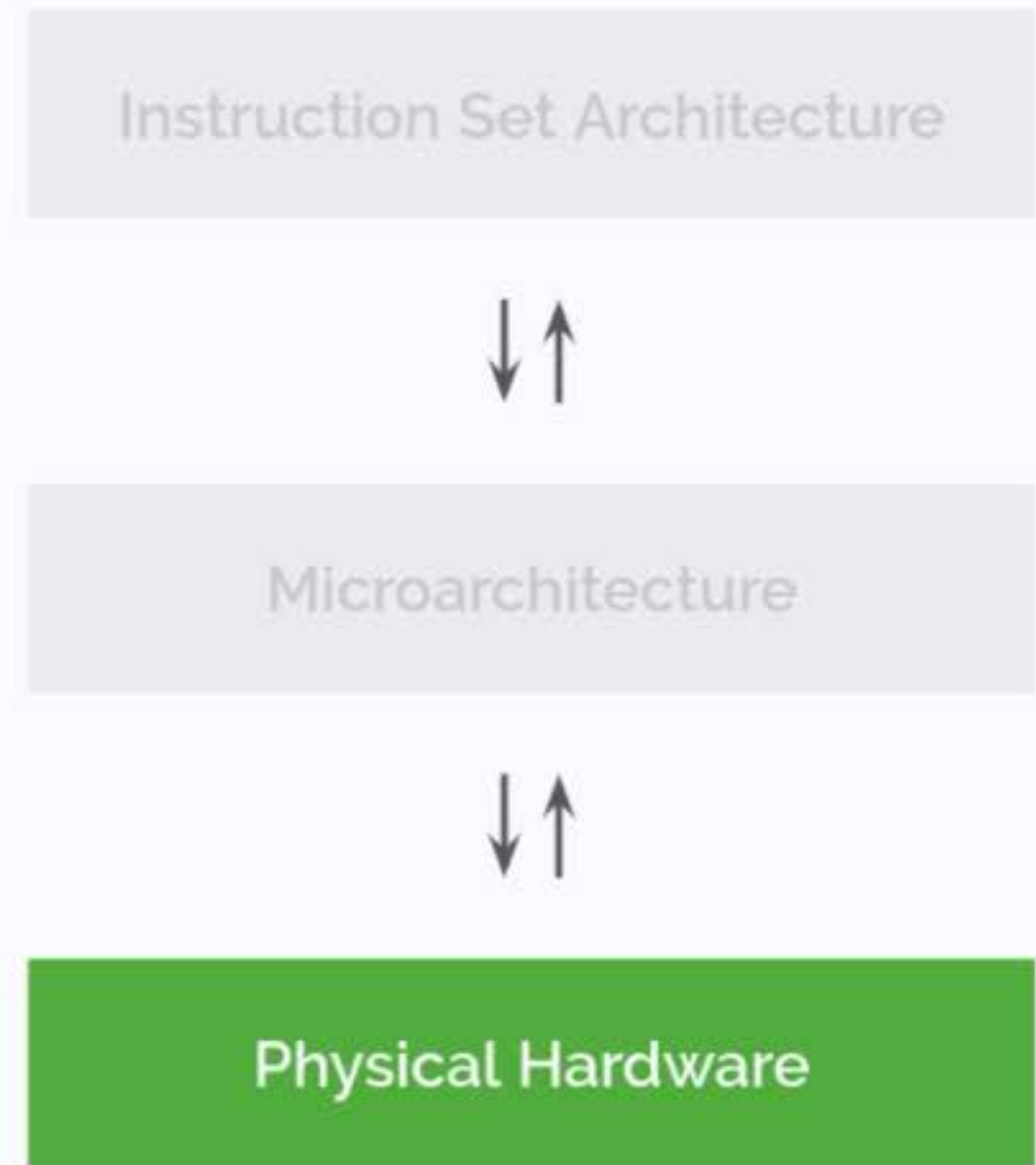
Peeking Inside the Hardware Implementation

Can assist in:

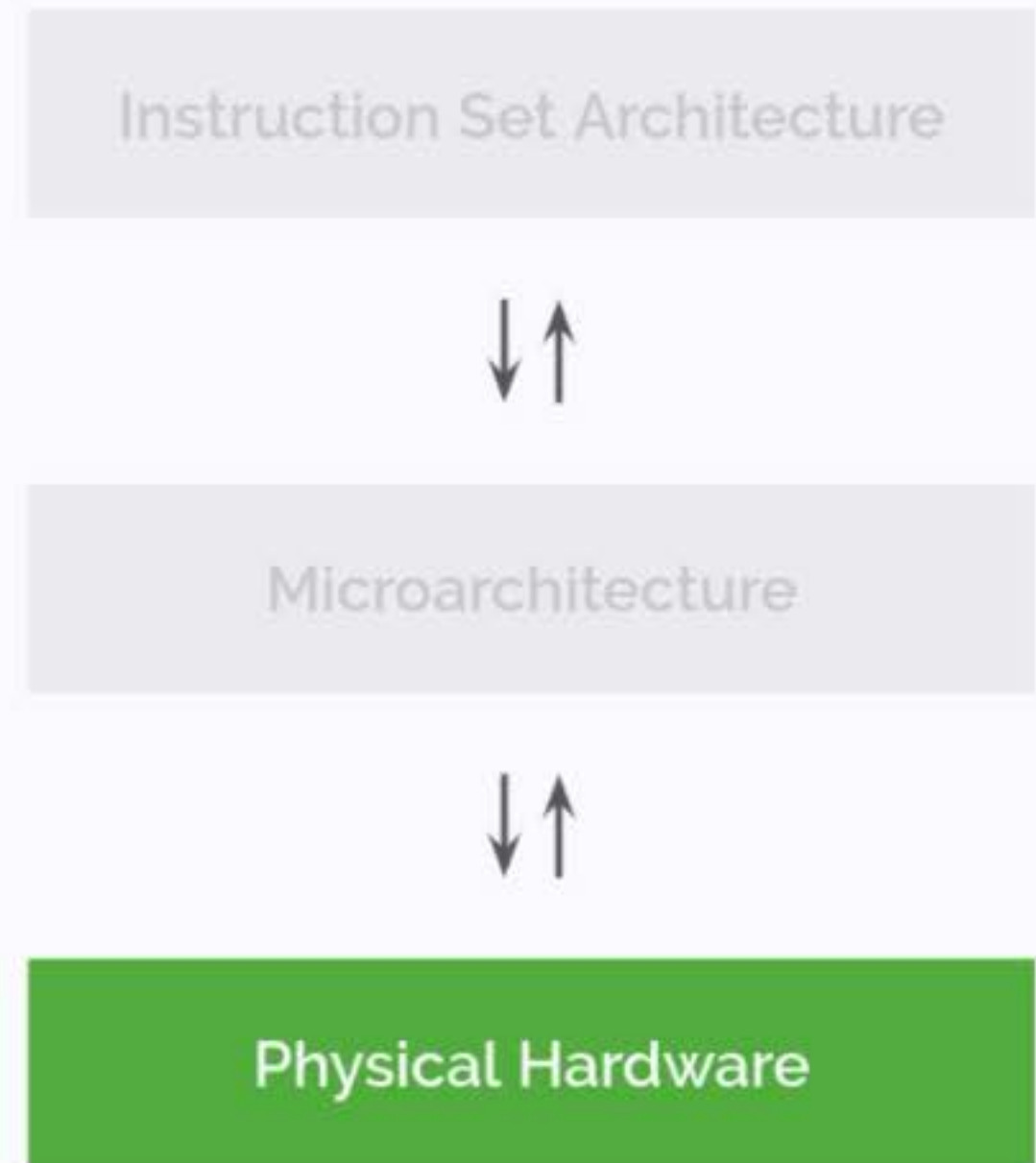
- Discovering Denial-of-Service attacks
- Locating Confused-Deputy problems
- Tuning performance and energy consumption



Program Analysis for Hardware Design



Program Analysis for Hardware Design



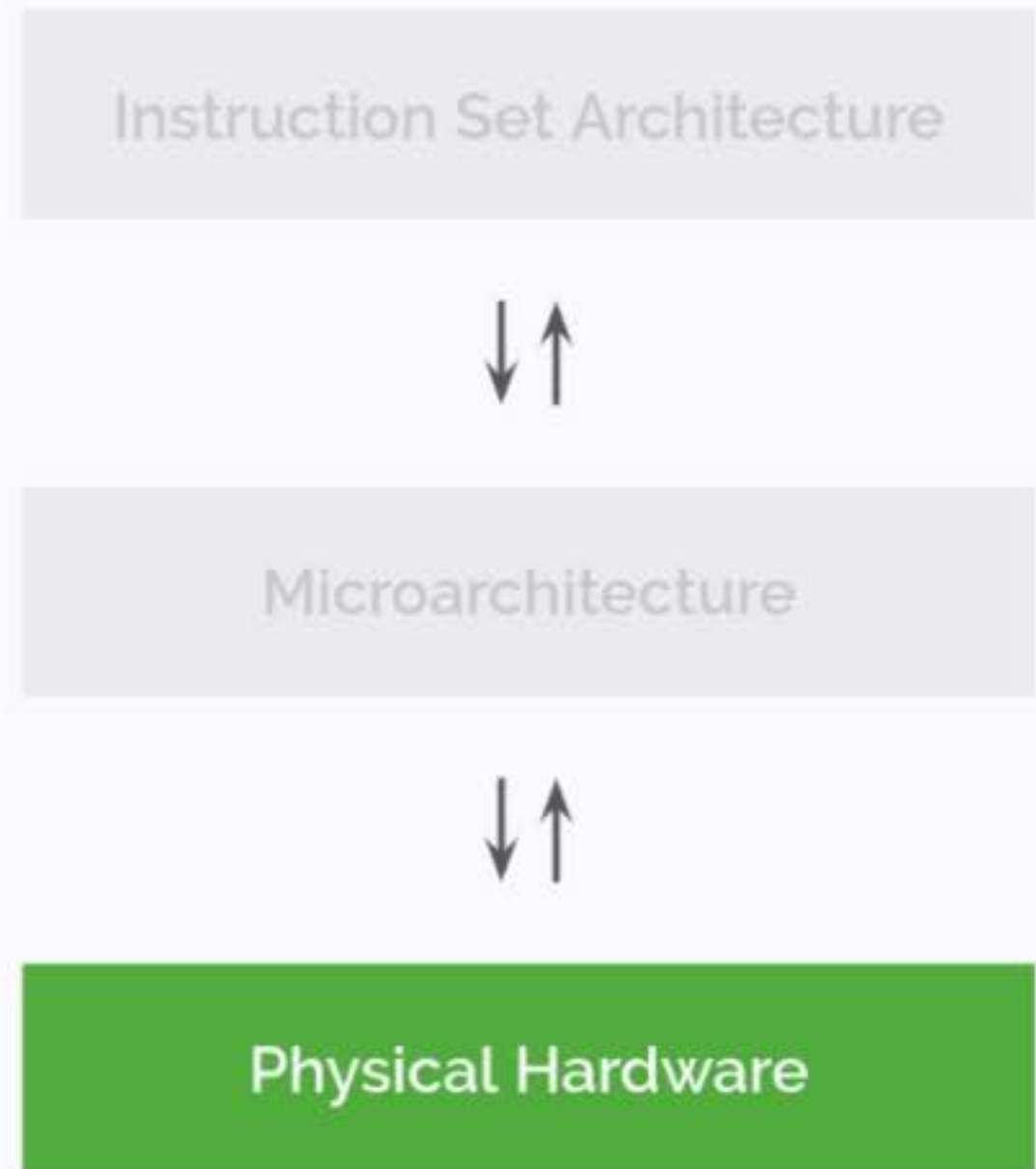
Functional programming for hardware design [e.g. Clash, Floh, SHard, and FLaSH]

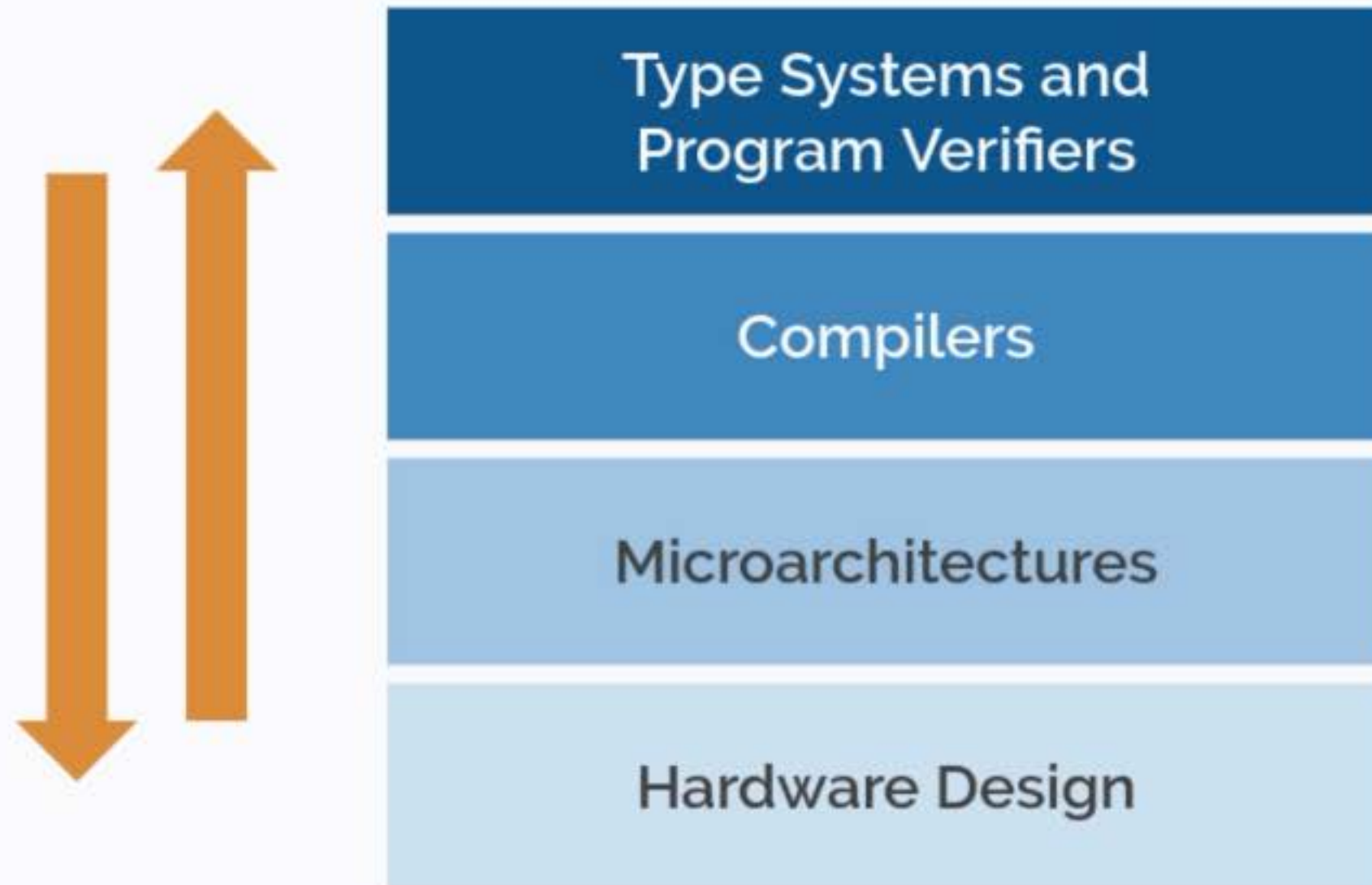
Program Analysis for Hardware Design

Research Questions:

- How can we quantify security?
- How can we transform programs for energy efficiency?

Functional programming for hardware design [e.g. Clash, Floh, SHard, and FLaSH]





Thanks to My Collaborators and Sponsors

Calvin Lin, PhD Advisor, UT Austin

Mohit Tiwari, PhD Advisor, UT Austin

K Rustan M. Leino, Amazon

Chris Hawblitzel, Microsoft Research

Laure Thompson, Cornell University

Joshua Eversmann, Civitas Learning

Greg McDonald, HBK Investments

Bryan Parno, Carnegie Mellon University

Jacob R. Lorch, Microsoft Research

Srinath Setty, Microsoft Research

Manos Kapritsos, University of Michigan

Barry Bond, Microsoft Research

Raymond Chee, Carnegie Mellon University

Varun Adiga and **Kasra Sadeghi**, UT Austin

