Secure, Precise, and Fast Floating-Point Operations on x86 Processors

Ashay Rane, Calvin Lin, Mohit Tiwari

The University of Texas at Austin

Floating-Point Math is Complicated

Specification itself is 70 pages long.

Processor manuals devote 100s of pages to describe implementation.

Demands from FP Operations

Correctness ✓

Checked using formal verifications tools like ACL2, HOL4.

Performance ✓

Ensured using wide vector units, GPUs, and FPGAs.

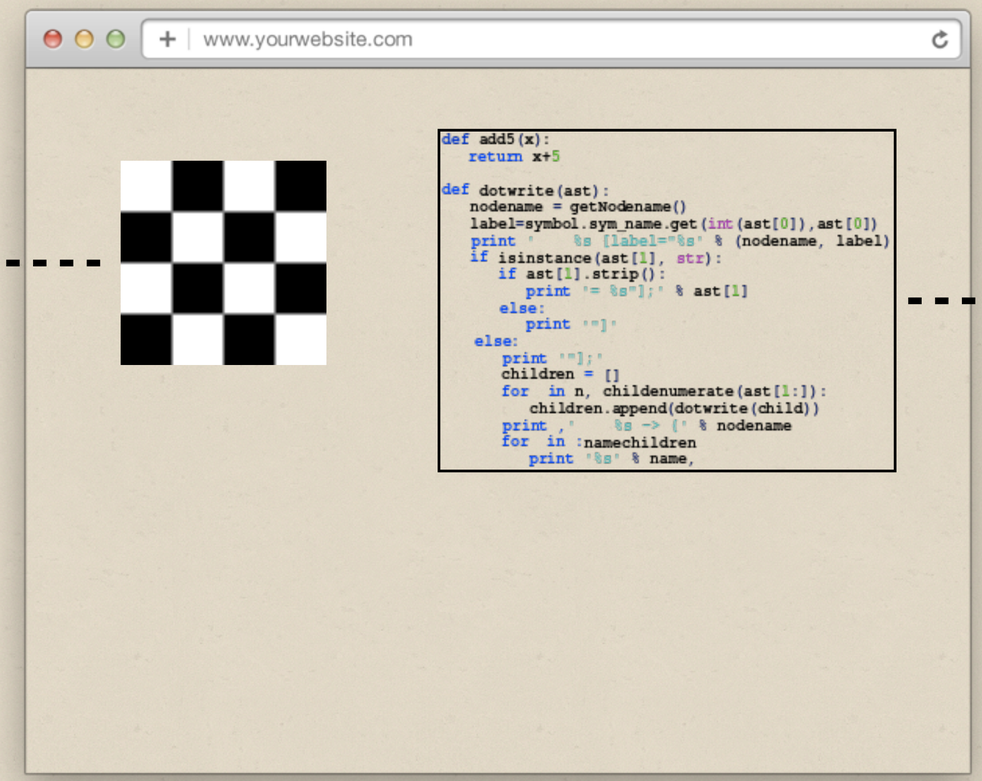
Security ✗

No strong solutions to close side channels.

Impact of Weak Security

Andrysco et al. [S&P'15] broke same-origin policy in Firefox, using timing side channel in floating-point operations.

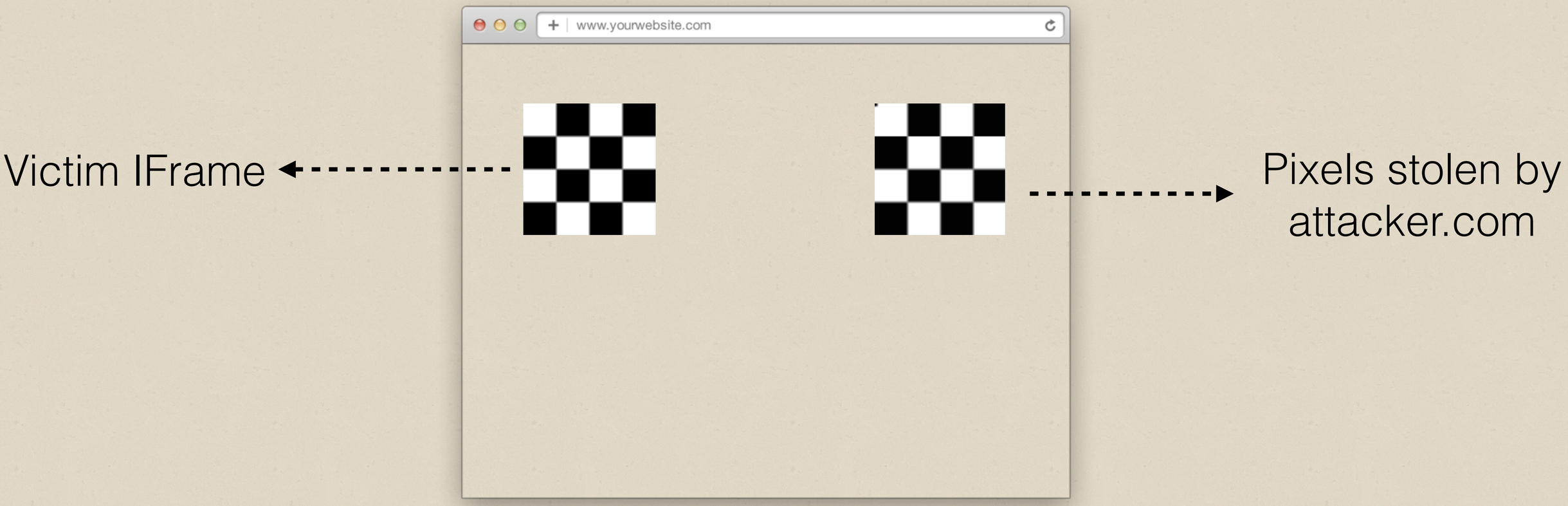
Victim IFrame



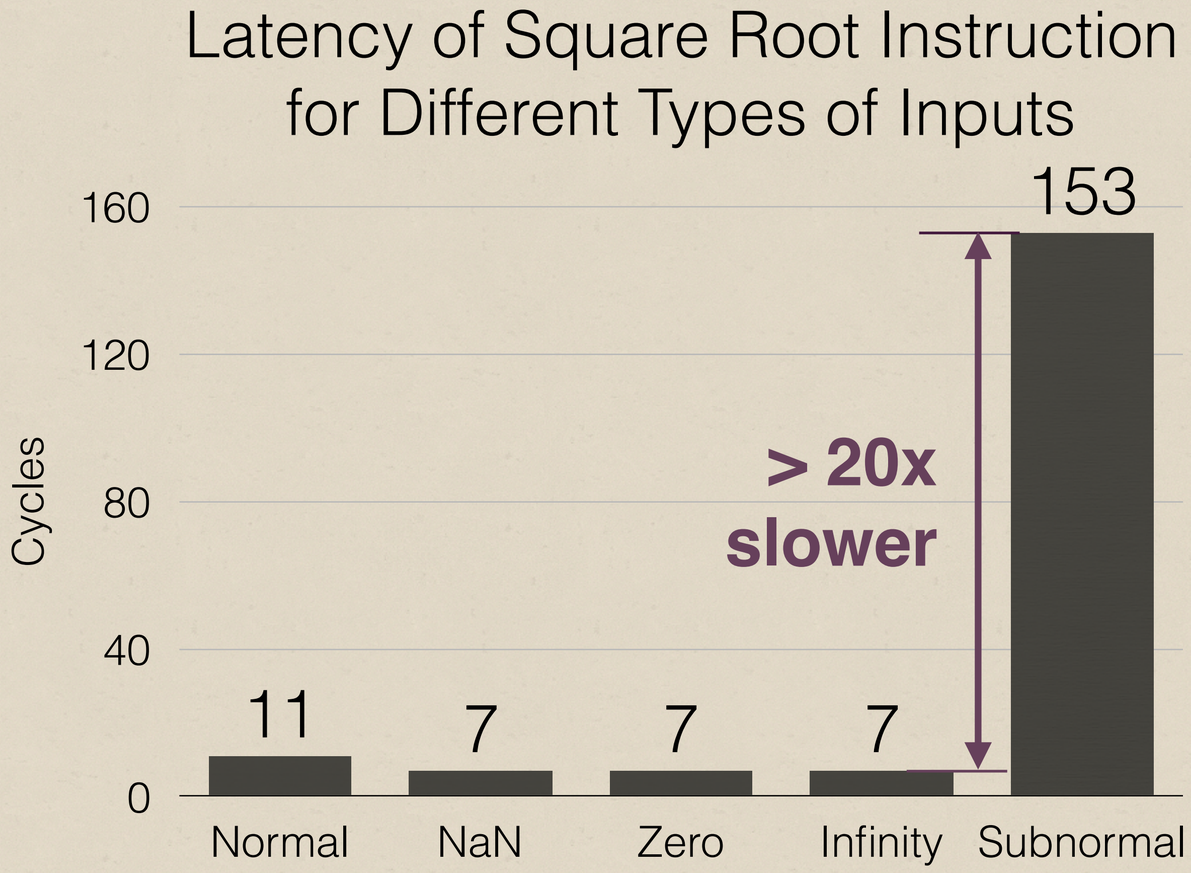
Script from attacker.com

Impact of Weak Security

Andrysco et al. [S&P'15] broke same-origin policy in Firefox, using timing side channel in floating-point operations.



Side Channel in FP Instructions



Measured on an Intel Sandy Bridge processor.

e.g. add, sub, etc.

Adversary's model:

Long latency implies Subnormal operand.

Side Channels in FP Functions

e.g. sin, cos, pow, etc.

```
float exp10f(float x) {

}
```

Side Channels in FP Functions

```
float exp10f(float x) {

  if ((int(n) >> 23 & 0xff) < 0x82) {

    if (y == 0.0f) {
      return p;
    }

  }

}
```

Language features that may leak secrets over side channels:

- `if` statements

Side Channels in FP Functions

```
float exp10f(float x) {
  float n, y = mf(x, &n);
  if ((int(n) >> 23 & 0xff) < 0x82) {

    if (y == 0.0f) {
      return p;
    }
    return exp2f(          ) * p;
  }
  return exp2(        );
}
```

Language features that may leak secrets over side channels:

- `if` statements
- function calls

Side Channels in FP Functions

```
float exp10f(float x) {
  float n, y = mf(x, &n);
  if ((int(n) >> 23 & 0xff) < 0x82) {
    float p = p10[(int) n + 7];
    if (y == 0.0f) {
      return p;
    }
    return exp2f(           ) * p;
  }
  return exp2(          );
}
```

Language features that may leak secrets over side channels:

- `if` statements
- function calls
- array accesses

Side Channels in FP Functions

```
float exp10f(float x) {
  float n, y = mf(x, &n);
  if ((int(n) >> 23 & 0xff) < 0x82) {
    float p = p10[(int) n + 7];
    if (y == 0.0f) {
      return p;
    }
    return exp2f(3.322f * y) * p;
  }
  return exp2(3.322 * x);
}
```

Language features that may leak secrets over side channels:

- `if` statements
- function calls
- array accesses
- floating-point instructions

Side Channels in FP Functions

```
float exp10f(float x) {
  float n, y = mf(x, &n);
  if ((int(n) >> 23 & 0xff) < 0x82) {
    float p = p10[(int) n + 7];
    if (y == 0.0f) {
      return p;
    }
    return exp2f(3.322f * y) * p;
  }
  return exp2(3.322 * x);
}
```

Language features that may leak secrets over side channels:

- `if` statements
- function calls
- array accesses
- floating-point instructions
- loops
- pointer dereferences

Solution: Escort

Compiler that generates

floating-point operations

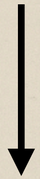
that close digital side channels

Side channels that carry information over **discrete bits**.

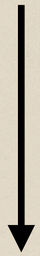
e.g. Time, Cache, Address Trace, Branch Predictor

NOT Power, EM Radiation, Heat, Sound.

Math library
source code



Escort Compiler



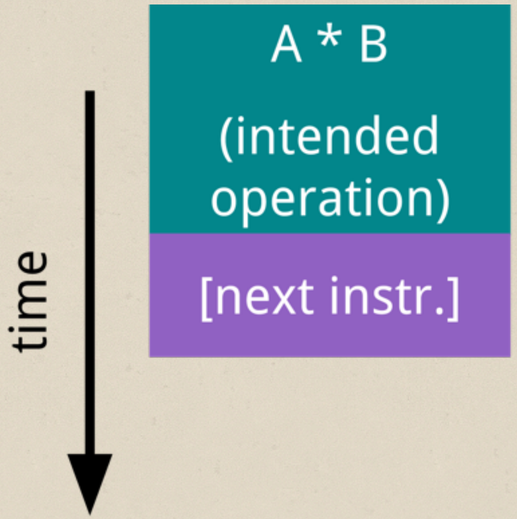
Secure math library
(shared object)

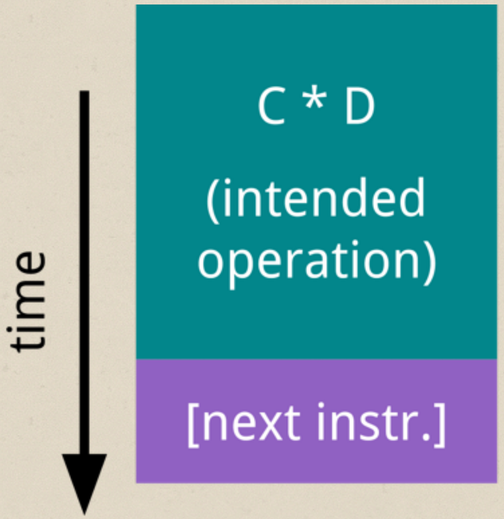
For example:
math functions in
the Musl C library.

Does not leak secrets
over digital side channels.

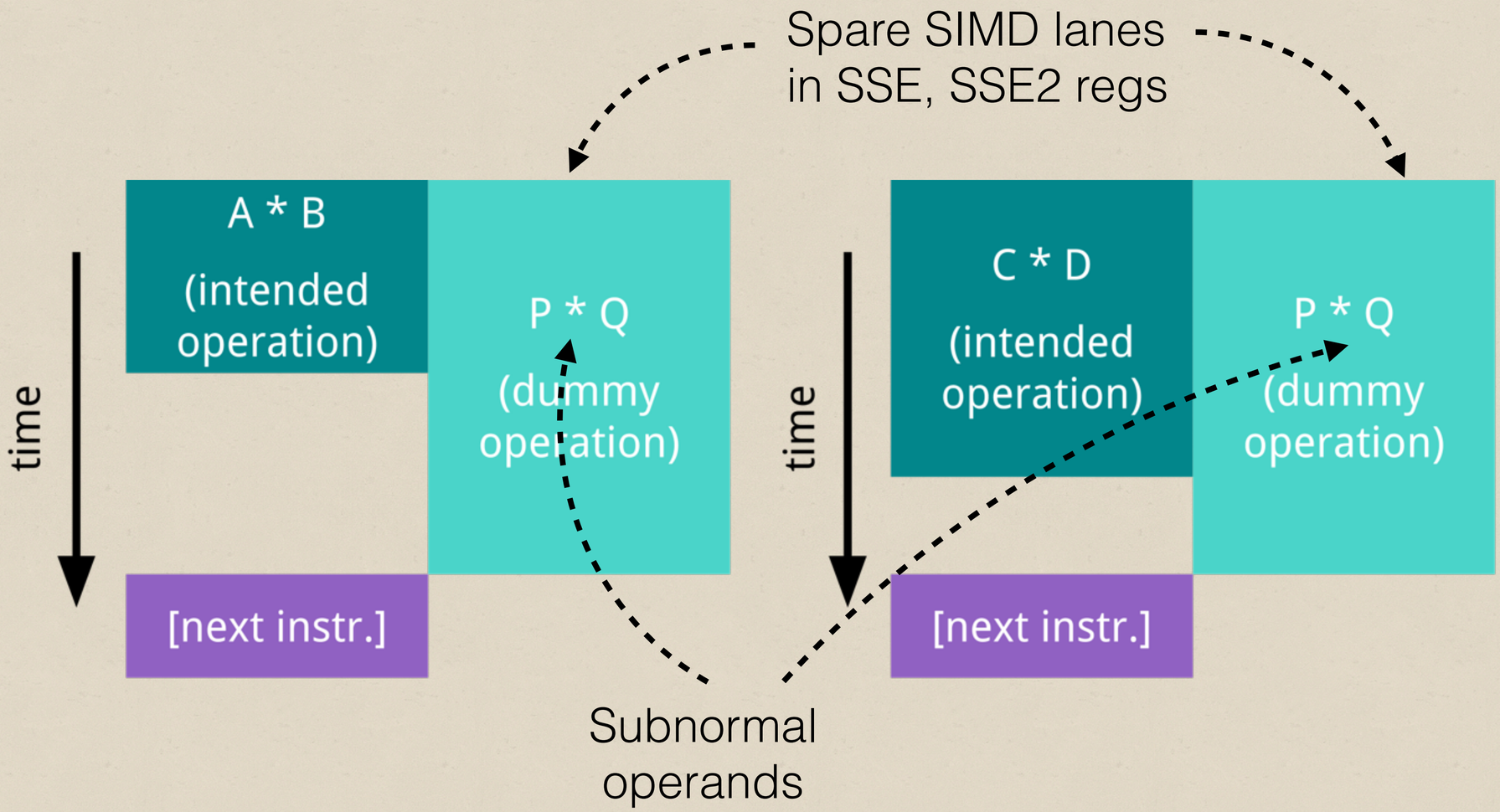
11 basic operations (add, sub, etc.) and
112 software functions (sin, cos, etc.)

Basic Operations: **Non-Secure** Execution





Basic Operations: **Secure** Execution



FP Software Functions

Floating-point software contains:

1. Forward branches (`if` statements)
2. Backward branches (loops)
3. Array Accesses
4. Function calls
5. Pointer dereferences
6. Floating-point instructions

Forward Branches

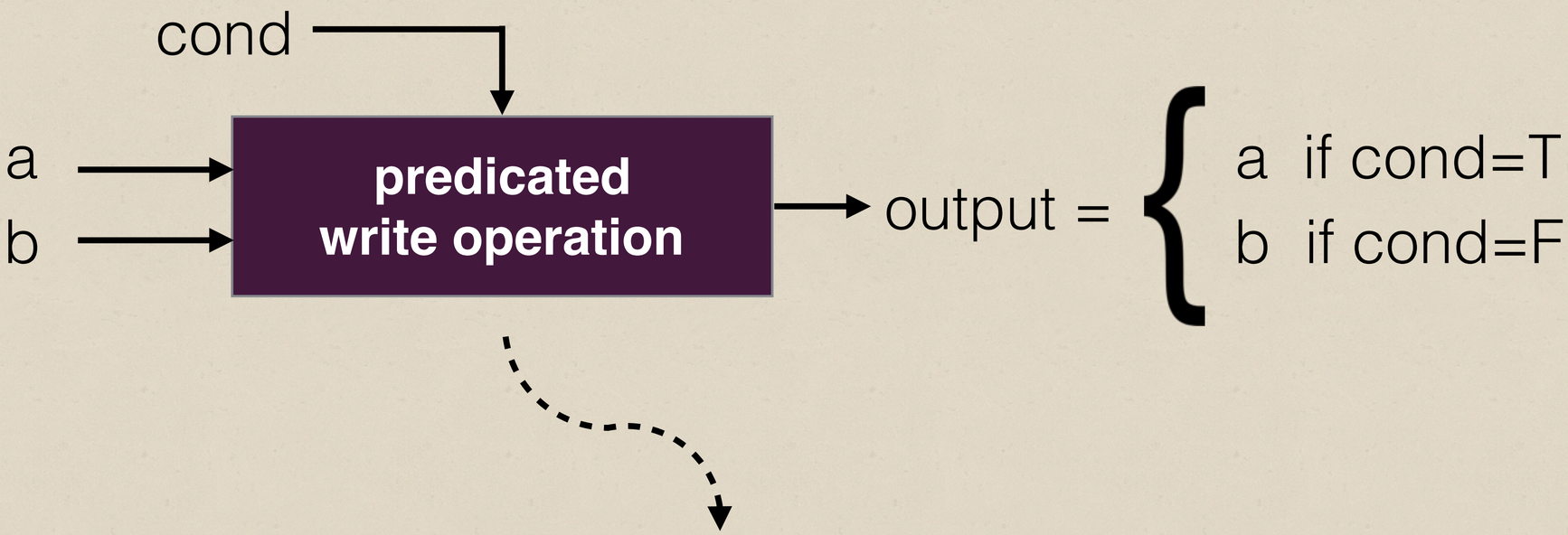
```
if (secret) {
    x = 5;
} else {
    x = 10;
}
```

⇔

```
  secret:   x = 5;
¬ secret:   x = 10;
```

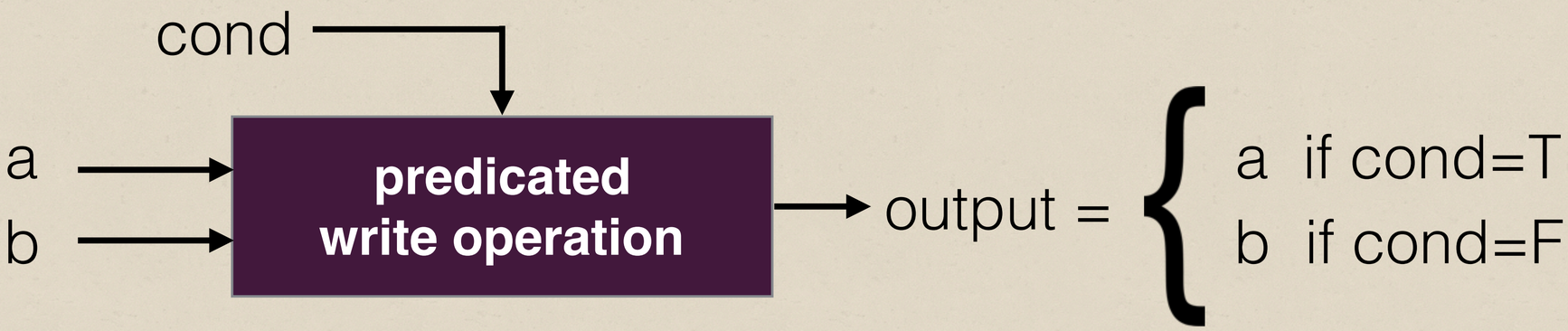
The predicates
guard assignment.

Predicated Execution




```
mov    a, output     // Set destination
test   cond, cond    // Check if non-zero
cmovz  b, output     // Conditional update
```

Predicated Execution




```
output = pred_write(cond, a, b)
```

Escort uses `pred_write` to control side effects: memory writes, function calls, and exceptions.

Forward Branches

```
   secret:  x = 5;
¬secret:  x = 10;
```

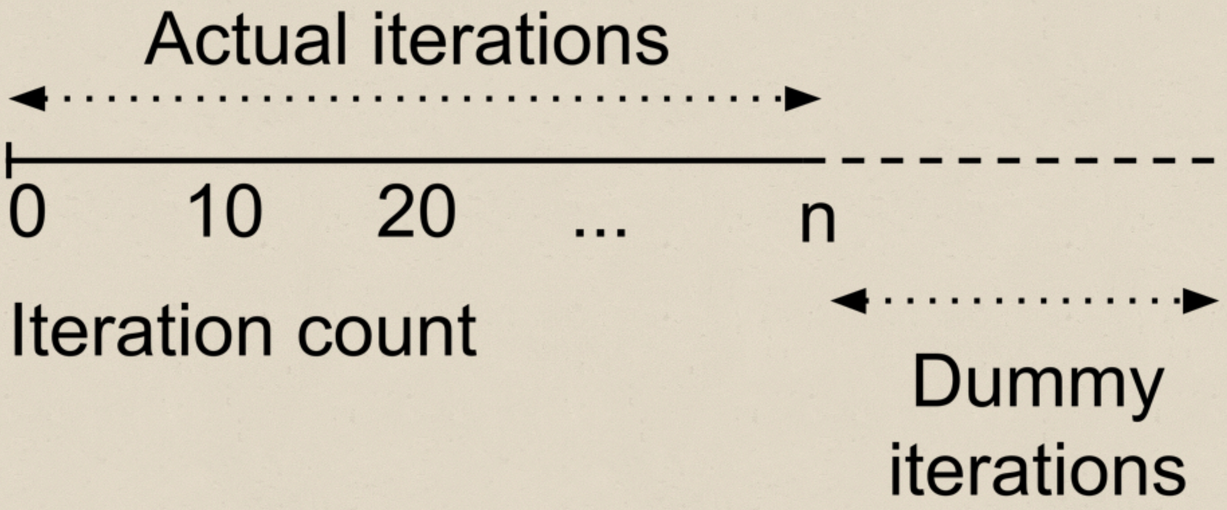
⇔

```
x = pred_write(secret, 5, x);
x = pred_write(¬secret, 10, x);
```

Backward Branches (Loops)

```
loop i :: 0 to n
  do_x;
  i = i + 1;
```

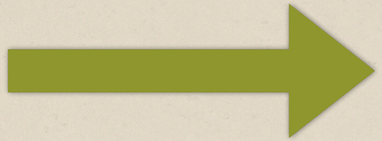
Assume 'n' is secret.



Backward Branches (Loops)

```
loop i :: 0 to n
  do_x;
  i = i + 1;
```

Escort's transformation



```
loop i' :: 0 to C

i' = i' + 1
```

Backward Branches (Loops)

```
loop i :: 0 to n
  do_x;
  i = i + 1;
```

Escort's transformation

```
loop i' :: 0 to C
  pred: do_x;
  pred: i = i + 1;
```

```
i' = i' + 1
```

Introduce predicate
for the original loop body.

Backward Branches (Loops)

```
loop i :: 0 to n
  do_x;
  i = i + 1;
```

Escort's transformation

```
pred = true;

loop i' :: 0 to C
  pred: do_x;
  pred: i = i + 1;

i' = i' + 1
```

Introduce predicate for the original loop body.

Backward Branches (Loops)

```
loop i :: 0 to n
  do_x;
  i = i + 1;
```

Escort's transformation

```
pred = true;

loop i' :: 0 to C
  pred: do_x;
  pred: i = i + 1;
  i == n: pred = false;

  i' = i' + 1
```

Introduce predicate for the original loop body.

Turn predicate OFF to run dummy iterations.

Backward Branches (Loops)

```
pred = true;

loop i' :: 0 to C
  pred: do_x;
  pred: i = i + 1;

  i == n: pred = false;
  i' = i' + 1
```

Annotated by user, for example:

```
__escort_loop_bounds(1024, 2048);
```

min max

OR

Decided automatically
using predictive mitigation [CCS'11]:
C = 1, 2, 4, 8, 16, 32, …

Array Accesses

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

```
addr := base(table) + secret
read addr
```

Adversary that can observe address, can also derive secret.

Escort's transformation

```
loop i :: 0 to n
  i == secret: result = table[i];
```

Expensive to access **entire array**, but math libraries use few tables.

Evaluation

Security

- Instructions
- Software functions
- Firefox using Escort
- Musl + Escort

Precision

- Comparing output of Escort with output of Musl

Performance

- Instructions
- Software functions

Evaluation

Security

- Instructions
- Software functions
- Firefox using Escort
- Musl + Escort
- Inference rules*

Precision

- Comparing output of Escort with output of Musl
- Escort in MinPack*

Performance

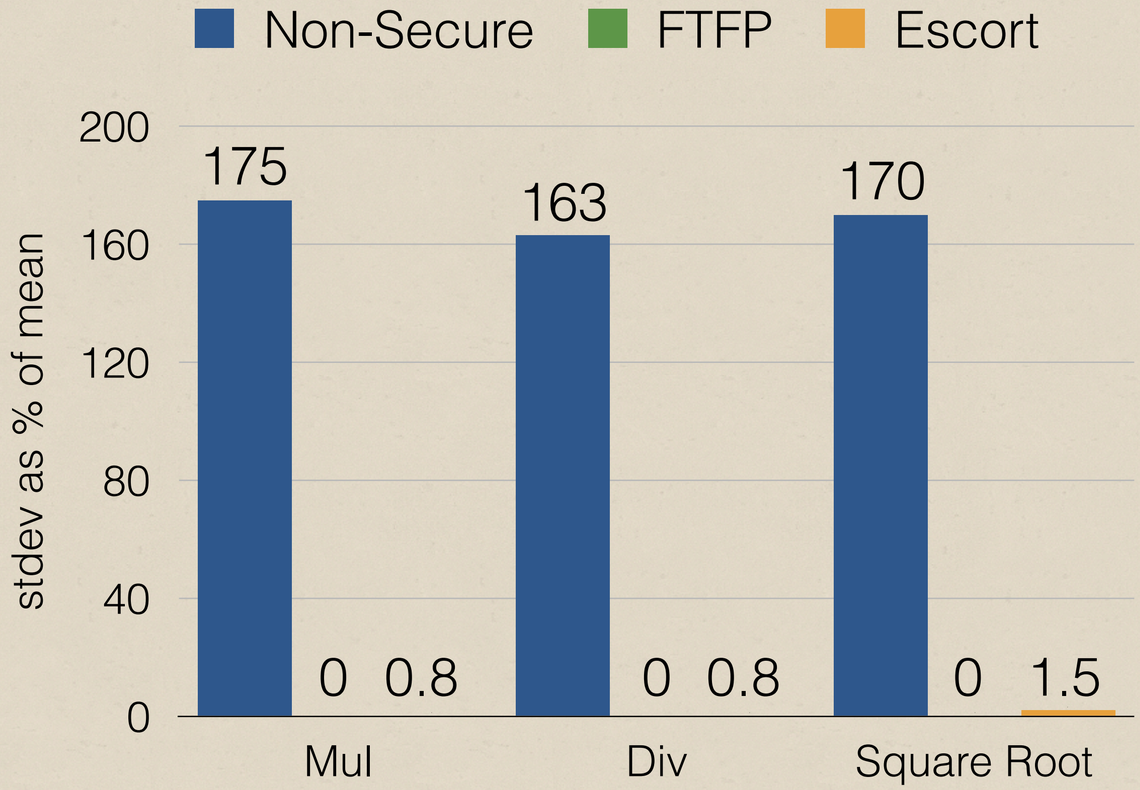
- Instructions
- Software functions
- Escort in SPEC*
- Escort in SVM*light**

* Described in the paper.

Verification of Defense

X axis: Floating-point instructions.

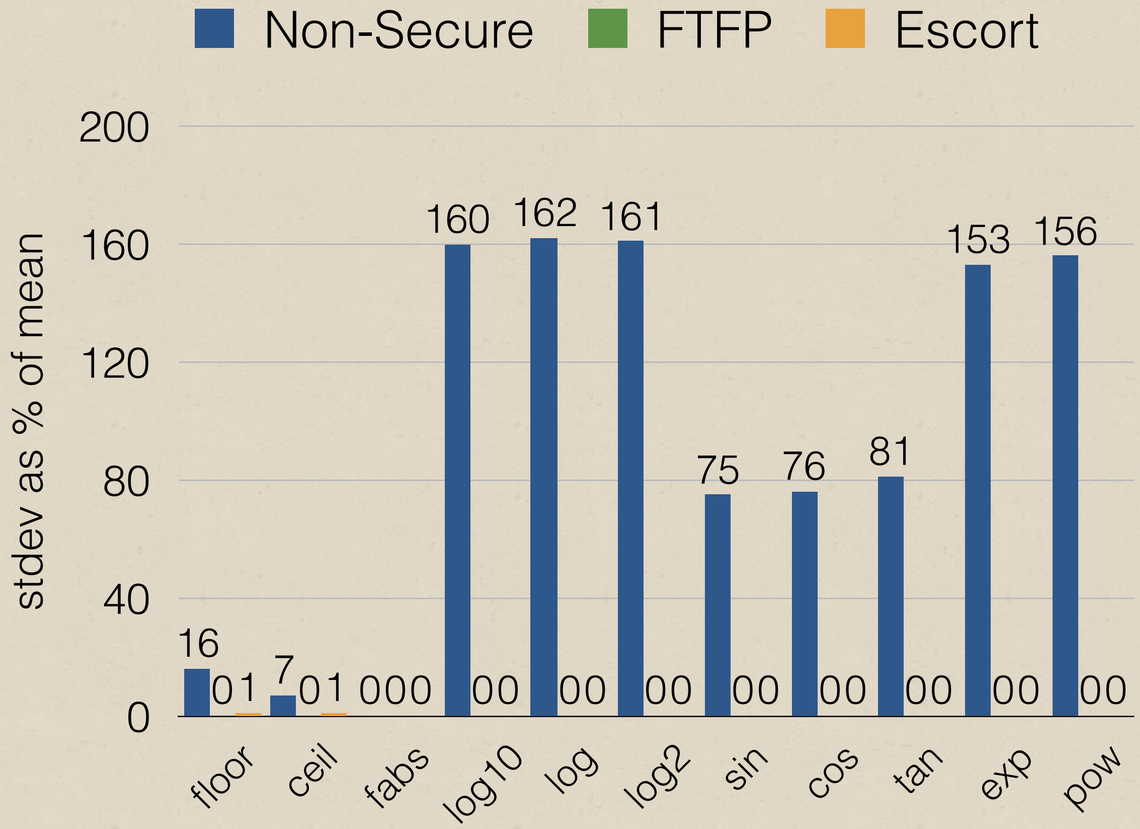
Y axis: Standard deviation of execution time as percentage of mean.



Verification of Defense

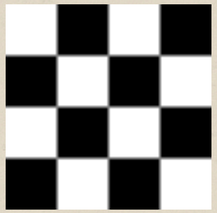
X axis: Floating-point functions.

Y axis: Standard deviation of execution time as percentage of mean.



Timing Side-Channel Defense

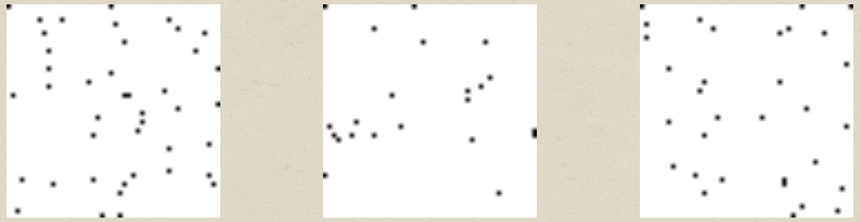
Andrysco et al. [S&P'15] broke same-origin policy in Firefox.



Contents of victim IFrame

IFrame contents recovered using timing side channel

We integrated Escort into Firefox, and re-ran the attack.



Recovered IFrames in 3 independent experiments, after using Escort in Firefox.

Address Trace Side-Channel Defense

Proof-of-concept side-channel attack on the Musl C library.

Attacker observes address trace when victim executes `expf()`, tries to guess input argument.

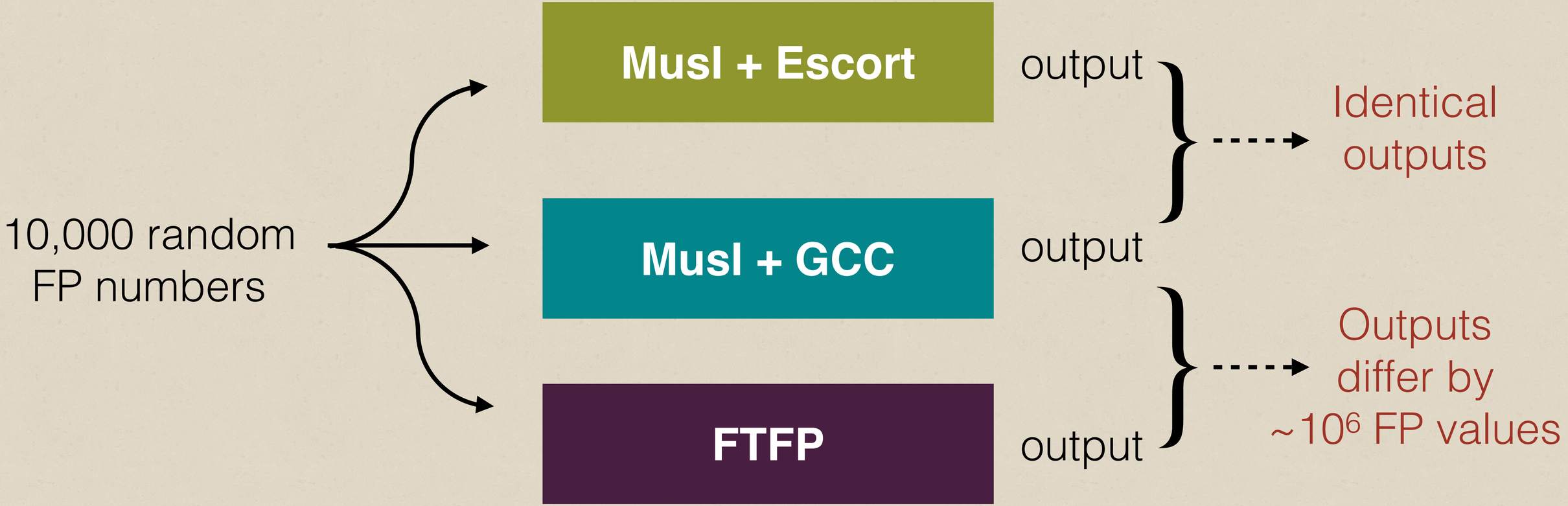
Without Escort:

Attacker correctly guesses input by observing address trace.

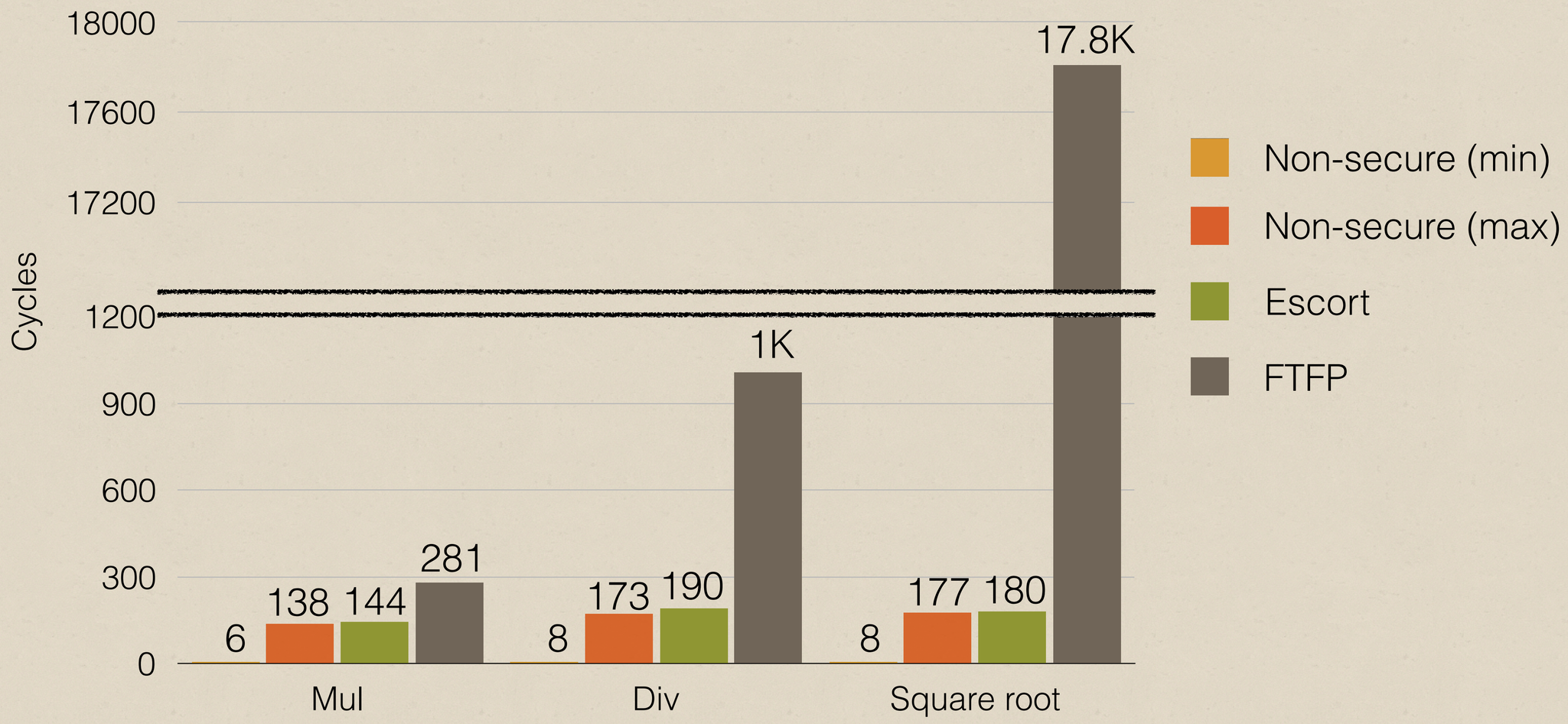
With Escort:

Attacker's guess is reduced to random chance.

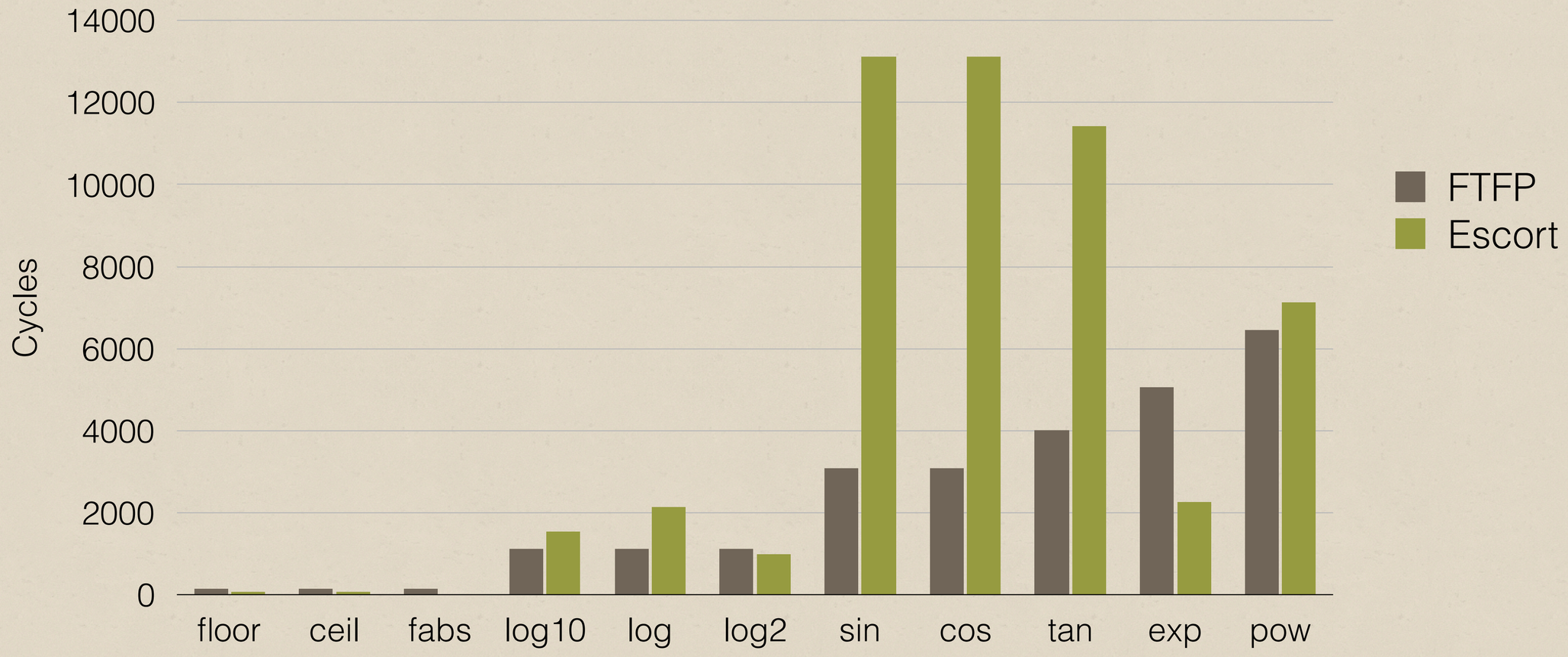
Precision Evaluation



Performance Evaluation: Basic Operations



Performance Evaluation: Higher-Level Operations



Related Work

- FTFP (libFixedTimeFixedPoint):
  - Fixed-point library that closes side channels
  - Imprecise, manually written.

- Compiler transformations by Cleemput et al.:
  - Weak security, manually written.

- Raccoon:
  - Slow, does not close timing side channel in FP operations.

There's More!

Features

- Handling pointers, exceptions.

Optimizations

- Optimization using SMT solver.

- Inference rules for static verification of Escort code. Security Evaluation

- Precision evaluation using MinPack, a non-linear solver.

Precision Evaluation

- Eliminating outliers in x86 performance tests. Performance Evaluation

- Comparison with our previous solution: Raccoon. Performance Evaluation

- End-to-end performance evaluation, on full application.

Performance Evaluation

Conclusion

- Escort is the first solution that achieves security, precision, and speed.
- Escort compiler is closely aware of the microarchitecture, and includes microarchitecture-aware transformations.
- Future work: Explore hardware support for closing side channels.