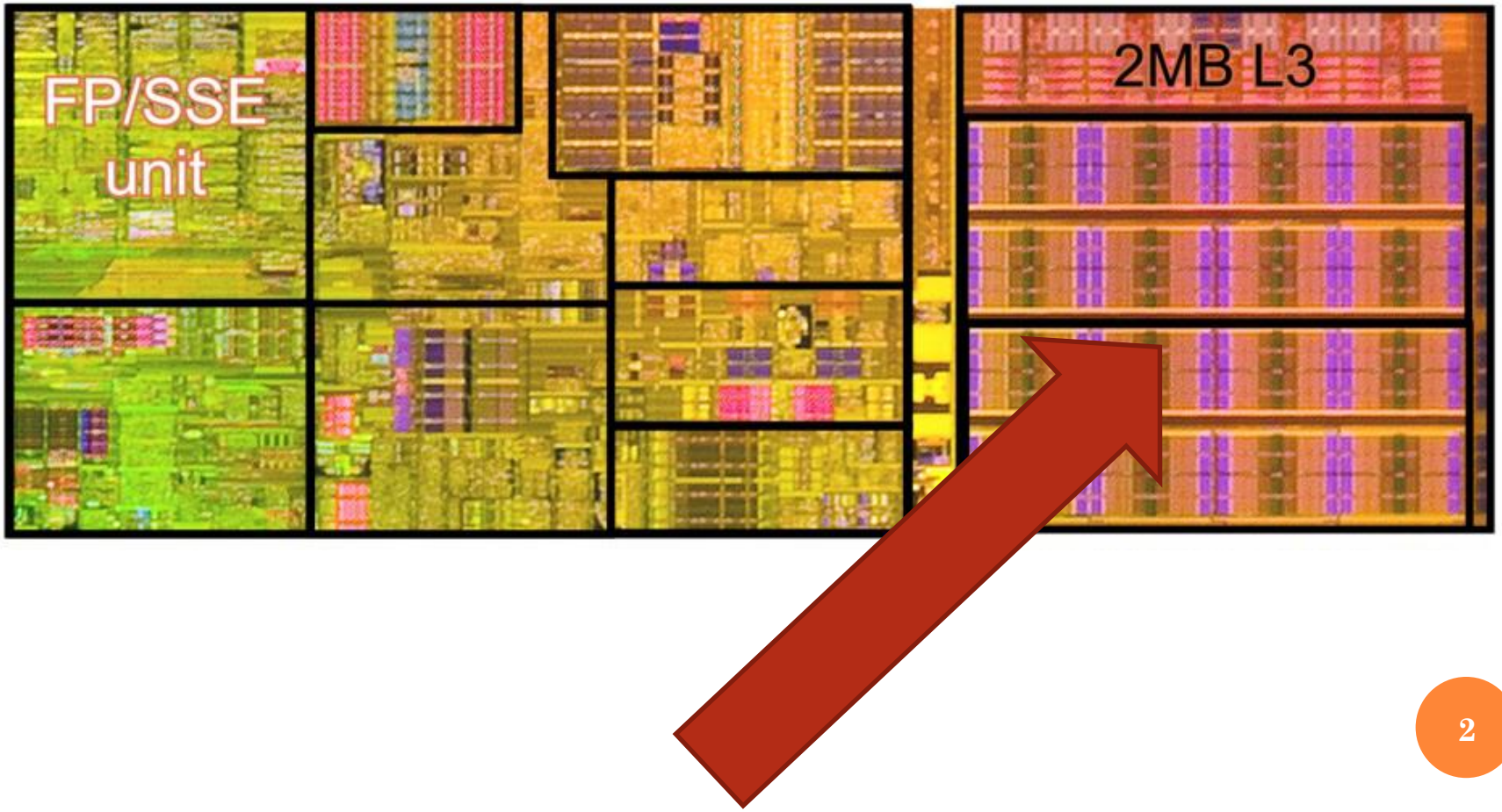
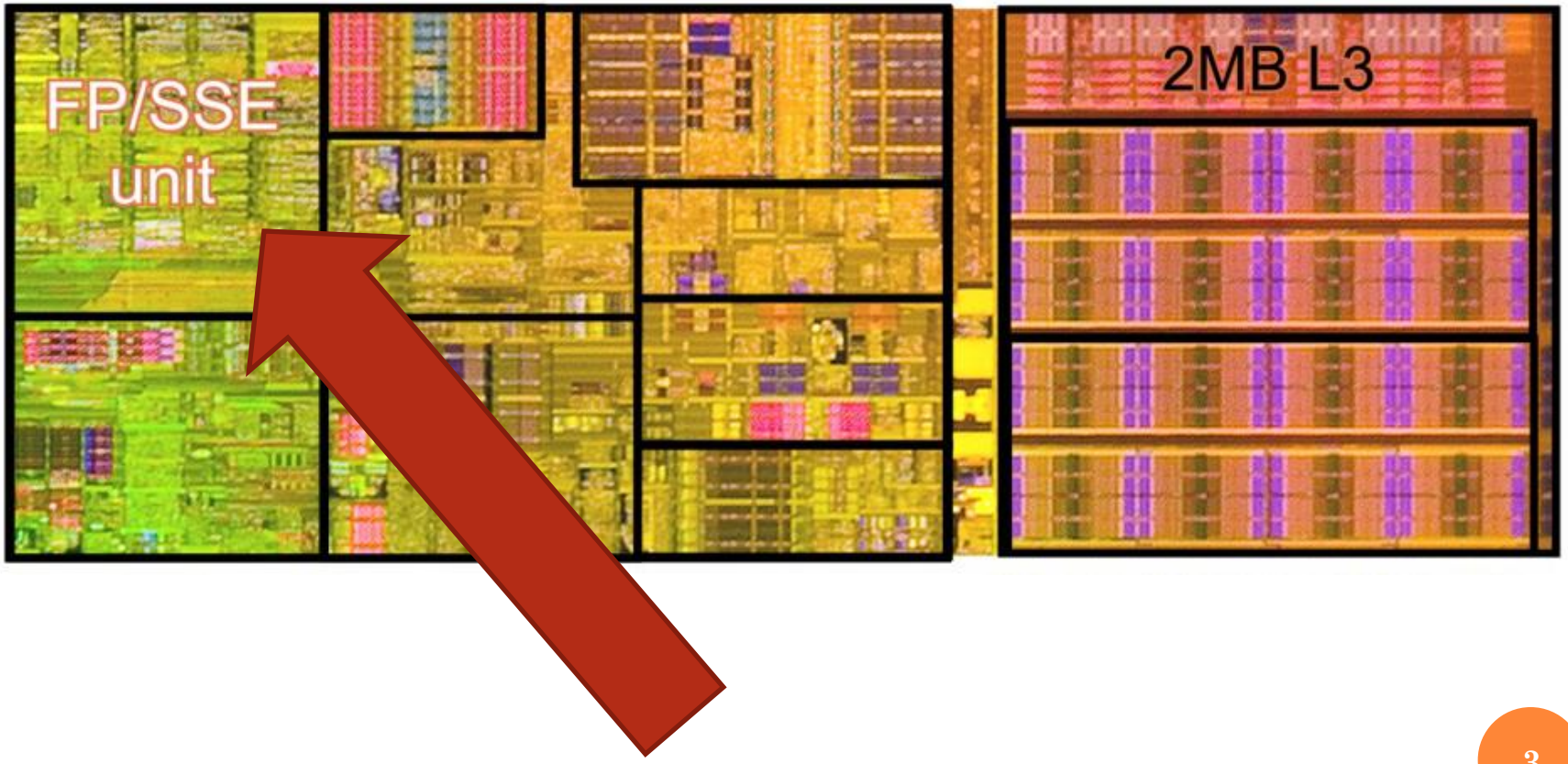
On Subnormal Floating Point and Abnormal Timing

Marc Andrysco, **David Kohlbrenner,** Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham

UC San Diego


FP/SSE
unit
2MB L3


FP/SSE
unit
2MB L3

Lets run some code

Normal Floating Point

```c
#include <stdio.h>
#include <stdint.h>

void main(int argc, char* argv[]){

  double x = 1.0;
  double z,y = 1.0;
  uint32_t i;
  for(i=0; i<100000000; i++){
    z = y*x;
  }
}
```

Subnormal Floating Point

```c
#include <stdio.h>
#include <stdint.h>

void main(int argc, char* argv[]){

  double x = 1.0e-323;
  double z,y = 1.0;
  uint32_t i;
  for(i=0; i<100000000; i++){
    z = y*x;
  }
}
```

Lets run some code

Normal Floating Point

```c
#include <stdio.h>
#include <stdint.h>

void main(int argc, char* argv[]){

  double x = 1.0;
  double z,y = 1.0;
  uint32_t i;
  for(i=0; i<100000000; i++){
    z = y*x;
  }
}
```

0.204s

Subnormal Floating Point

```c
#include <stdio.h>
#include <stdint.h>

void main(int argc, char* argv[]){

  double x = 1.0e-323;
  double z,y = 1.0;
  uint32_t i;
  for(i=0; i<100000000; i++){
    z = y*x;
  }
}
```

4.332s

20 times slower?

- Who knew?
  - Numerical analysts
  - CPU designers
  - Game engine authors

20 TIMES SLOWER?

- Who knew?
  - Numerical analysts
  - CPU designers
  - Game engine authors

- Who should know?
  - "What Every Computer Scientist Should Know About Floating-Point Arithmetic" – Goldberg '91

20 TIMES SLOWER?

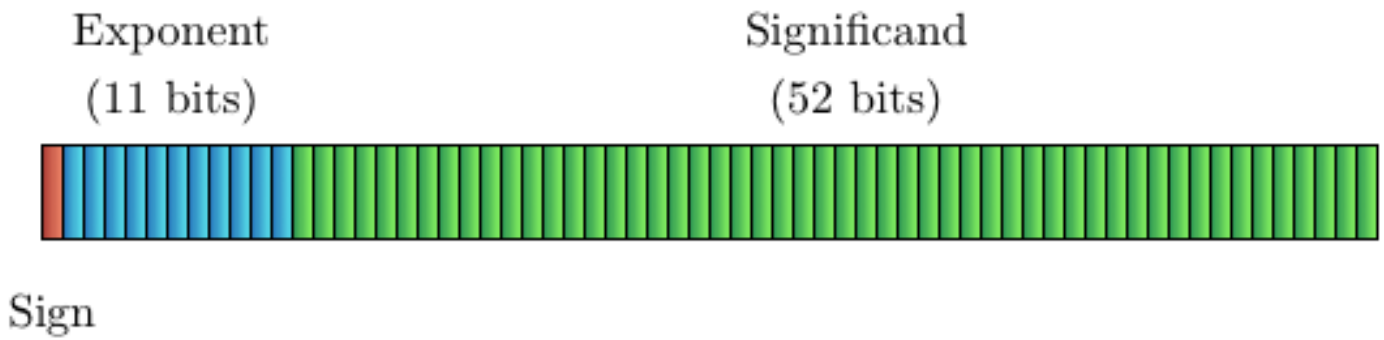
- Who knew?
  - Numerical analysts
  - CPU designers
  - Game engine authors
- Who should know?
  - "What Every Computer Scientist Should Know About Floating-Point Arithmetic" – Goldberg '91
- Academic researchers claim to "effectively close[s] all known remotely exploitable channels"
  - Specifically referring to timing side channels!

Floating Point and Timing

What happened?

- IEEE 754 specifies *subnormal* floating point values

Floating Point Normal and Subnormal

- 

- Value = $(-1)^{sign} * significand * 2^{(exponent-bias)}$
  - The exponent is non-zero
  - Normal values have an *implicit* leading 1-bit on the significand

- A *subnormal* value is a special encoding
  - The exponent is all zeroes
  - The significand has an *implicit* leading 0-bit

Subnormal Details

- Subnormal ranges (double)
  - Minimum: $\sim 4.9 \times 10^{-324}$
  - Maximum: $\sim 2.23 \times 10^{-308}$
  - Planck length: 1.6×10^{-35} m

- Why?
  - Extend the range of floating point
  - Graceful underflow

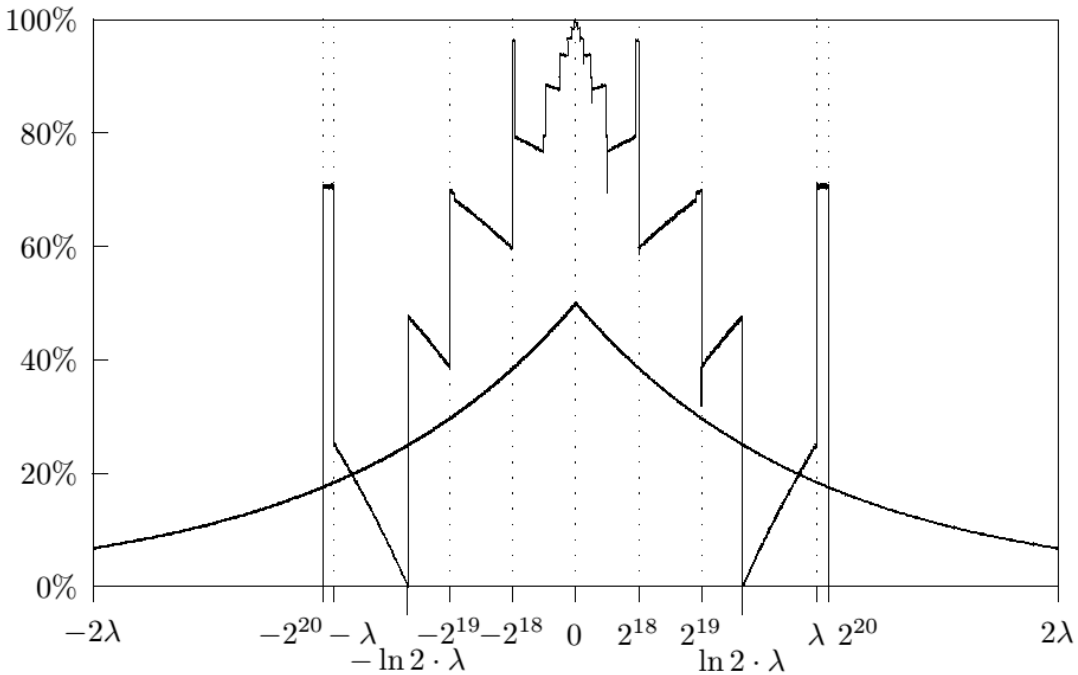
```
if(a != b)
    x = c / (a-b);
```

What happened?

- IEEE 754 specifies *subnormal* floating point values
- FPUs are optimized for pure speed
  - Subnormals are *not* the common case
  - So let's pretend they don't matter!
- Subnormals are a hardware slowpath
  - The Alpha trapped to kernel for subnormals!
  - Most GPUs don’t support them

Floating Point is a Security Issue

- Ilya Mironov on Laplacian noise generation



- Lack of dependable results
  - `gcc -O1` vs `gcc -O3`

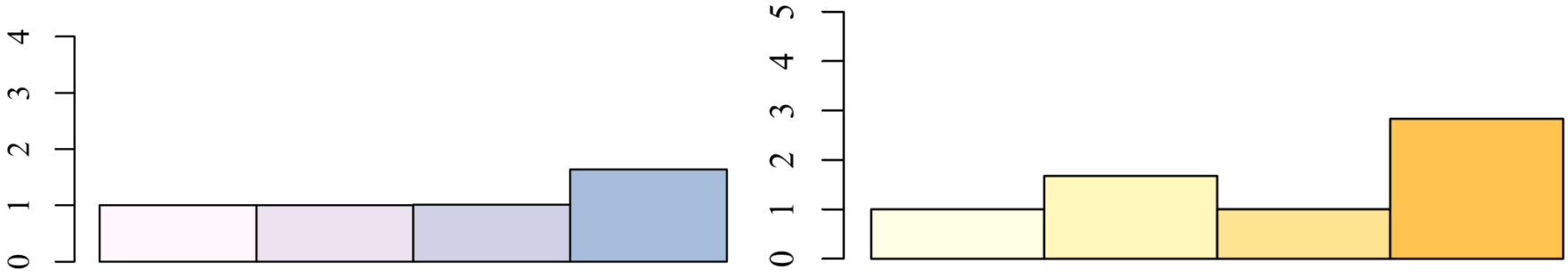
Leveraging subnormal floating point into attacks

Floating Point as a Side-Channel

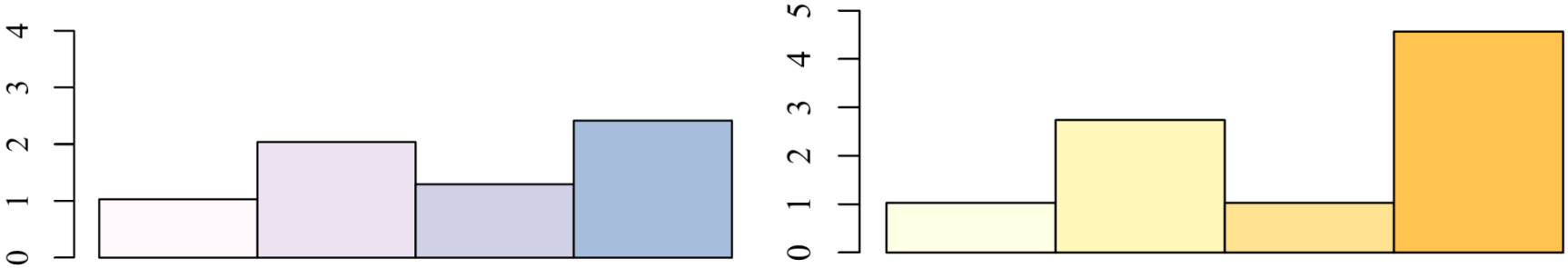
- Code that operates on *secret* and *attacker* values can result in timing side channels
  - From instruction traces
  - Or memory access patterns
  - Or IO usage
  - Etc.

- We present the first *instruction data* based timing side channel attack on a commodity desktop processor
  - Proposed by Kocher 20 years ago!

Floating Point Hardware Data



Core i7-3667U SSE and x87



Atom D2550 SSE and x87

- addsd normal,normal
- addsd normal,subnormal
- divsd normal,normal
- divsd normal,subnormal

- fadd normal,normal
- fadd normal,subnormal
- fmul normal,normal
- fmul normal,subnormal

Amplifying Timing Differences

- Even a 100 cycle difference is hard to spot
  - Especially with a loaded system

```
#include <stdio.h>
#include <stdint.h>

void main(int argc, char* argv[]){

  double x = 1.0e-323;
  double z,y = 1.0;
  uint32_t i;
  for(i=0; i<100000000; i++){
    z = y*x;
  }
}
```

Amplifying Timing Differences

- Even a 100 cycle difference is hard to spot
  - Especially with a loaded system
- We need an *amplifier*
- Remember our sample code?
  - We need tight math loops

```
#include <stdio.h>
#include <stdint.h>

void main(int argc, char* argv[]){

  double x = 1.0e-323;
  double z,y = 1.0;
  uint32_t i;
  for(i=0; i<100000000; i++){
    z = y*x;
  }
}
```

Detour Time!

Firefox SVG Filters and Previous Attacks

Firefox SVG Filters

- Turn this

Firefox SVG Filters

- Into this!

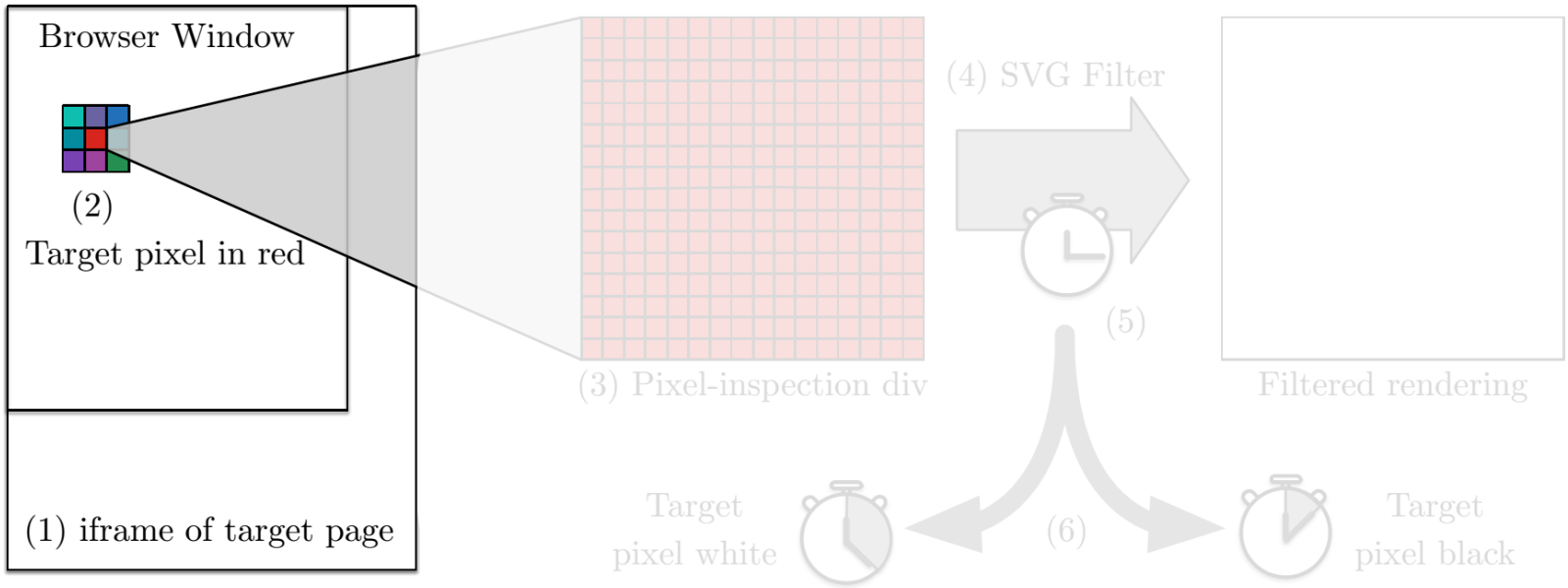
```
<svg><filter>
<feGaussianBlur
stdDeviation="3"/>
</filter></svg>
```

Firefox SVG Filters

- CSS defined filters
  - `<div>`
  - `<iframe>`
  - Really any element
- Run various functions
  - `convolve`
  - `blur`
  - `skew`
  - `gradient`
  - `clipping`
- Stackable!

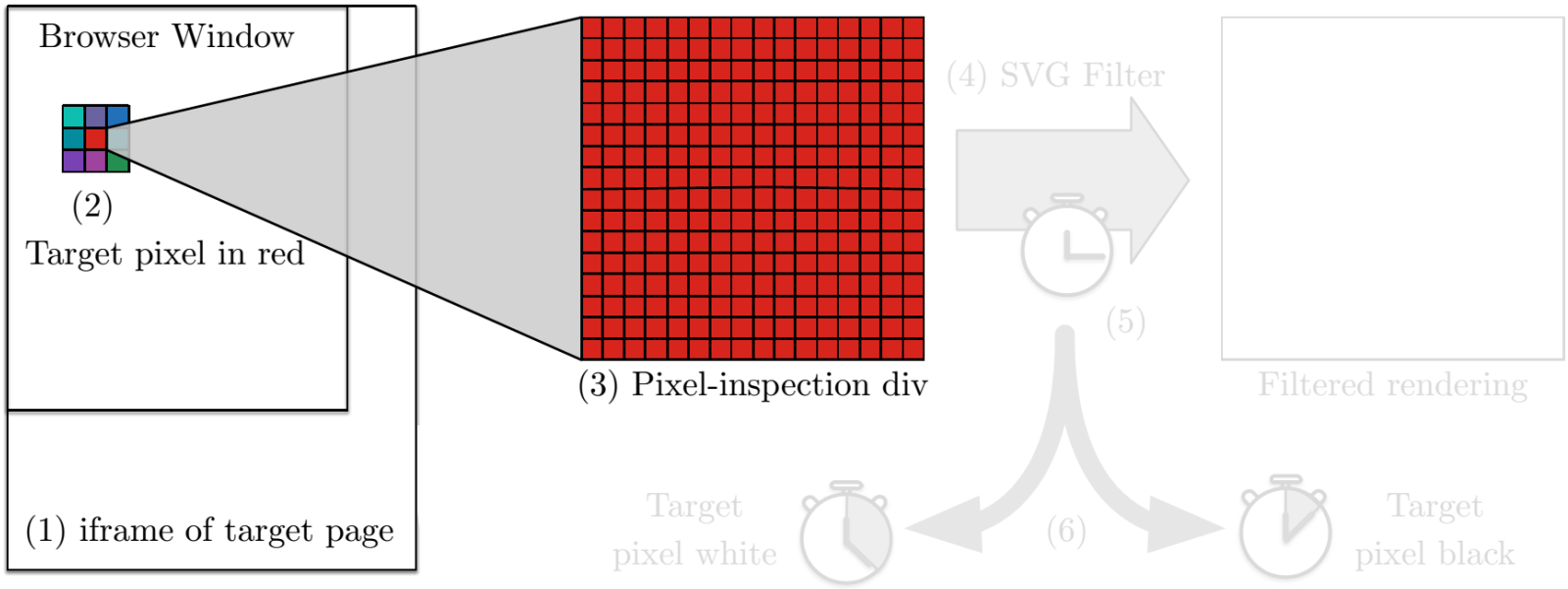
Firefox SVG Filter Timing Attack

- See Paul Stone's "Pixel Perfect Timing Attacks with HTML 5"



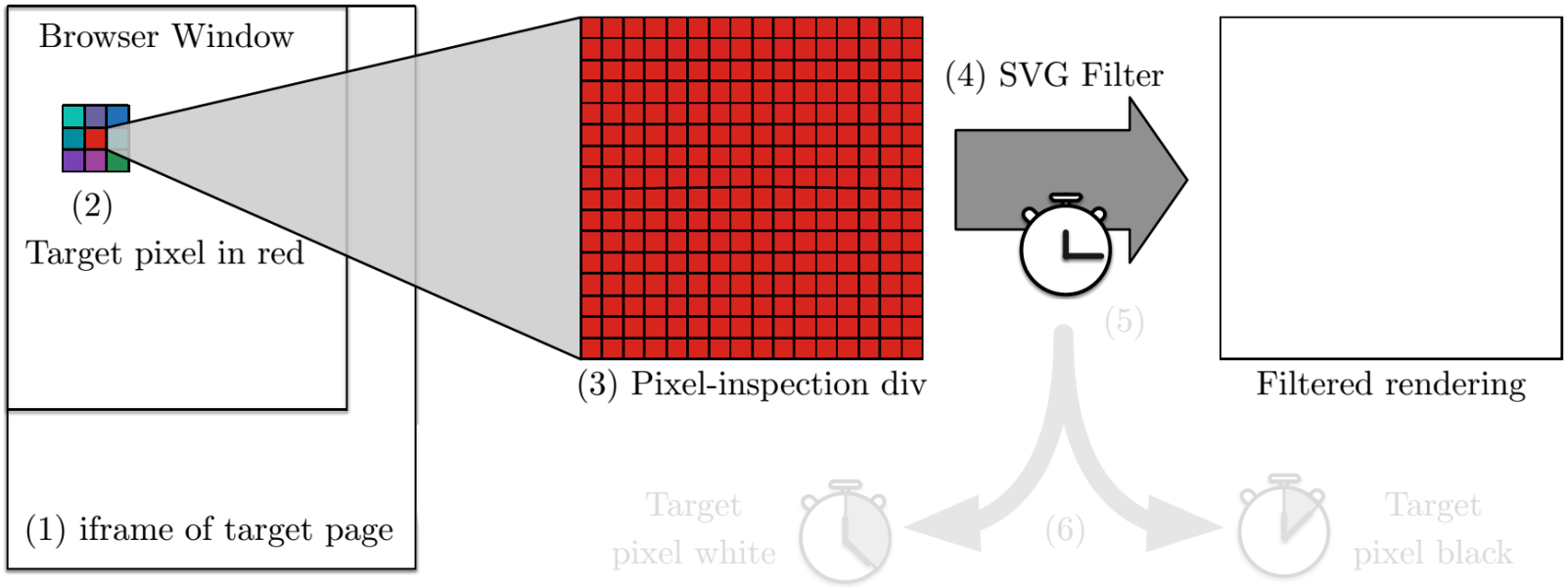
Firefox SVG Filter Timing Attack

- See Paul Stone's "Pixel Perfect Timing Attacks with HTML 5"



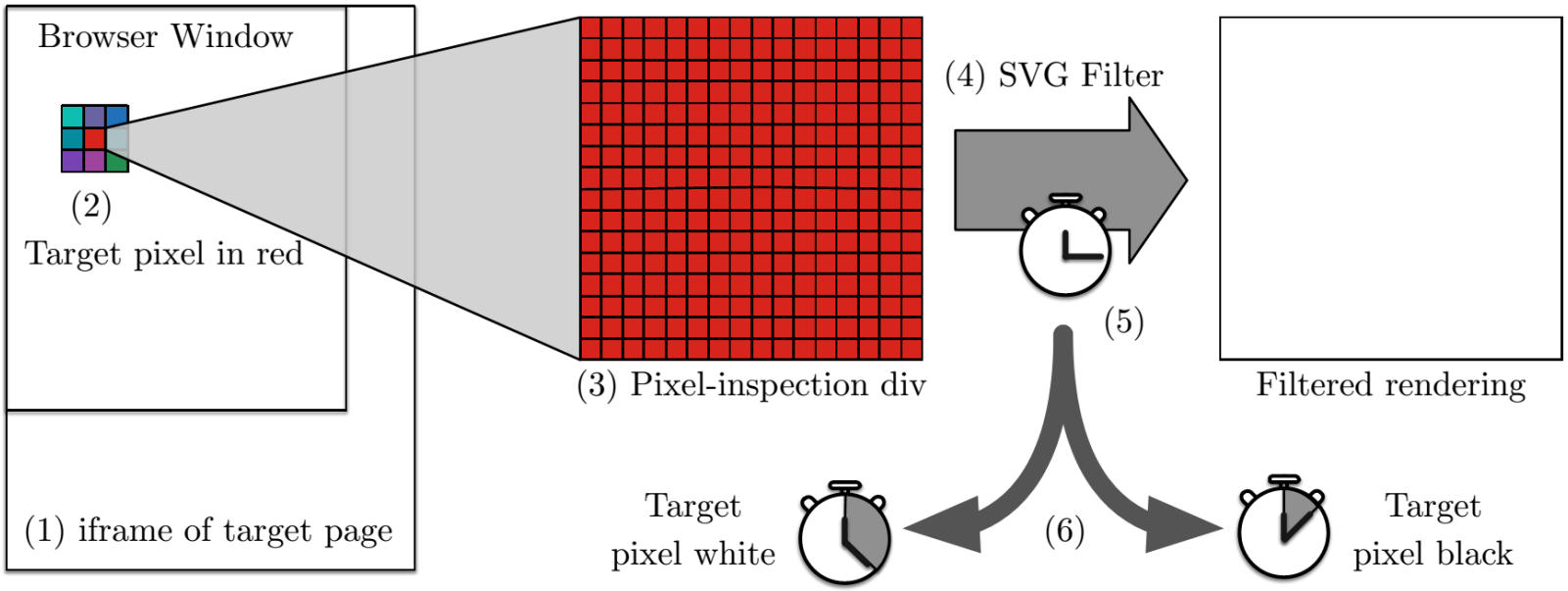
Firefox SVG Filter Timing Attack

- See Paul Stone's "Pixel Perfect Timing Attacks with HTML 5"



Firefox SVG Filter Timing Attack

- See Paul Stone's "Pixel Perfect Timing Attacks with HTML 5"



Paul Stone's SVG Timing Side Channel

- Relied on a *fast path optimization* in the `feMorphology` SVG filter
  - In cases of a solid color image, filter ran much faster

```
if (x == rect.x || xExt[0]  <= startX || xExt[1] <= startX ||
    xExt[2] <= startX || xExt[3] <= startX){
  [...]
 } else { // We only need to look at the newest column
  for (PRUint32 y1 = startY; y1 <= endY; y1++) {
    [...]
```

- Fix was to write constant time code!
  - Took ~2 years to land, and 150+ comment bug thread
  - "the problem boils down to: how to implement constant-time min(a, b) and max(a, b) in C++?" – Bugzilla thread

Back to The Present

New Firefox SVG Filter Attack

- Firefox SVG Filters are still ‘vulnerable’ pending a timing difference
- We have a new timing side-channel source

New Firefox SVG Filter Attack

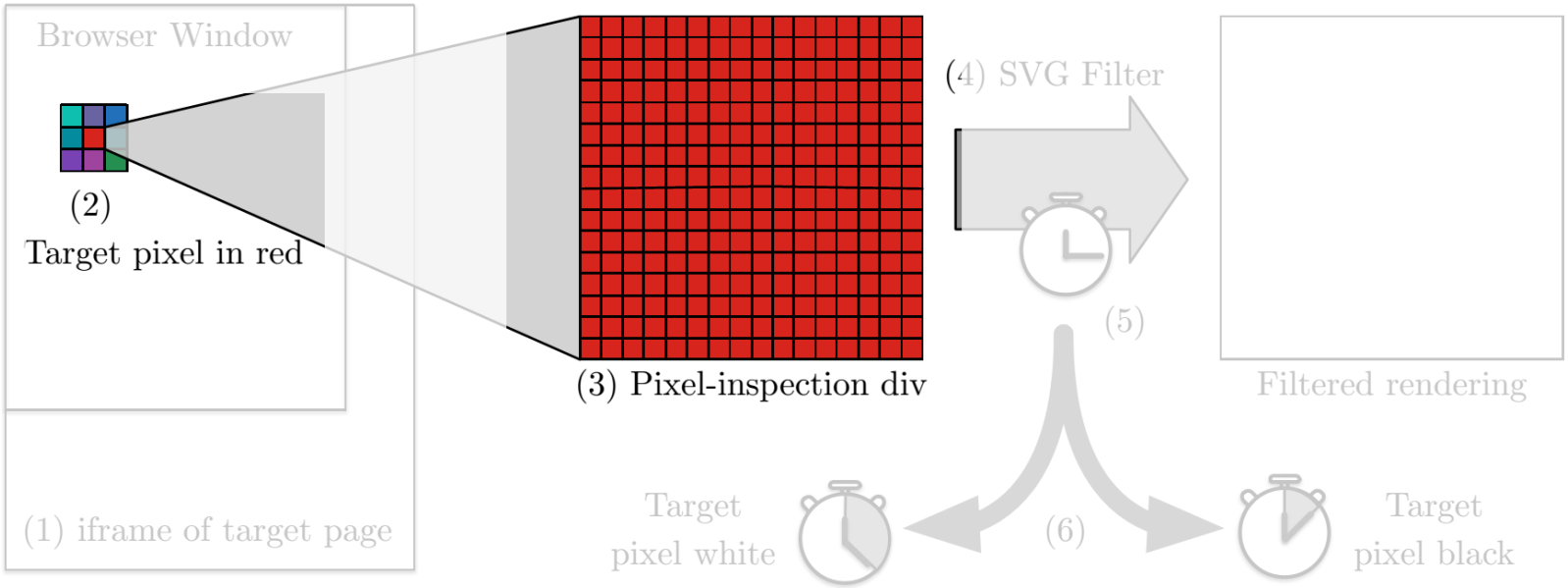
- Firefox SVG Filters are still ‘vulnerable’ pending a timing difference
- We have a new timing side-channel source
- SVG Filters run *floating point math!*

New Firefox SVG Filter Attack

- Firefox SVG Filters are still 'vulnerable' pending a timing difference
- We have a new timing side-channel source
- SVG Filters run *floating point math!*
- We need an *amplifier*

New Firefox SVG Filter Attack

- We need an *amplifier*



Firefox SVG Filters and Subnormals

```
def do_one_convolve(source_patch):
  source_patch = [[ ?, ?, ? ]
                  [ ?, ?, ? ]
                  [ ?, ?, ? ]]

  kernel = [[ 1e-42, 1e-42, 1e-42 ]
            [ 1e-42, 1e-42, 1e-42 ]
            [ 1e-42, 1e-42, 1e-42 ]]

  for x,y in source_patch:
      tmp[x][y] = source_patch[x][y] * kernel[x][y]

  result = 0

  for x,y in tmp:
      result += tmp[x][y]

  return result

for x,y in source_image:
  source_image[x][y] = do_one_convolve(swatch(x,y))
```

Firefox SVG Filters and Subnormals

```
def do_one_convolve(source_patch):
  source_patch = [[ ?, ?, ? ]
                  [ ?, ?, ? ]
                  [ ?, ?, ? ]]

  kernel = [[ 1e-42, 1e-42, 1e-42 ]
            [ 1e-42, 1e-42, 1e-42 ]
            [ 1e-42, 1e-42, 1e-42 ]]

  for x,y in source_patch:
      tmp[x][y] = source_patch[x][y] * kernel[x][y]

  result = 0

  for x,y in tmp:
      result += tmp[x][y]

  return result

for x,y in source_image:
  source_image[x][y] = do_one_convolve(swatch(x,y))
```

$1 \times s$

$0 \times s$

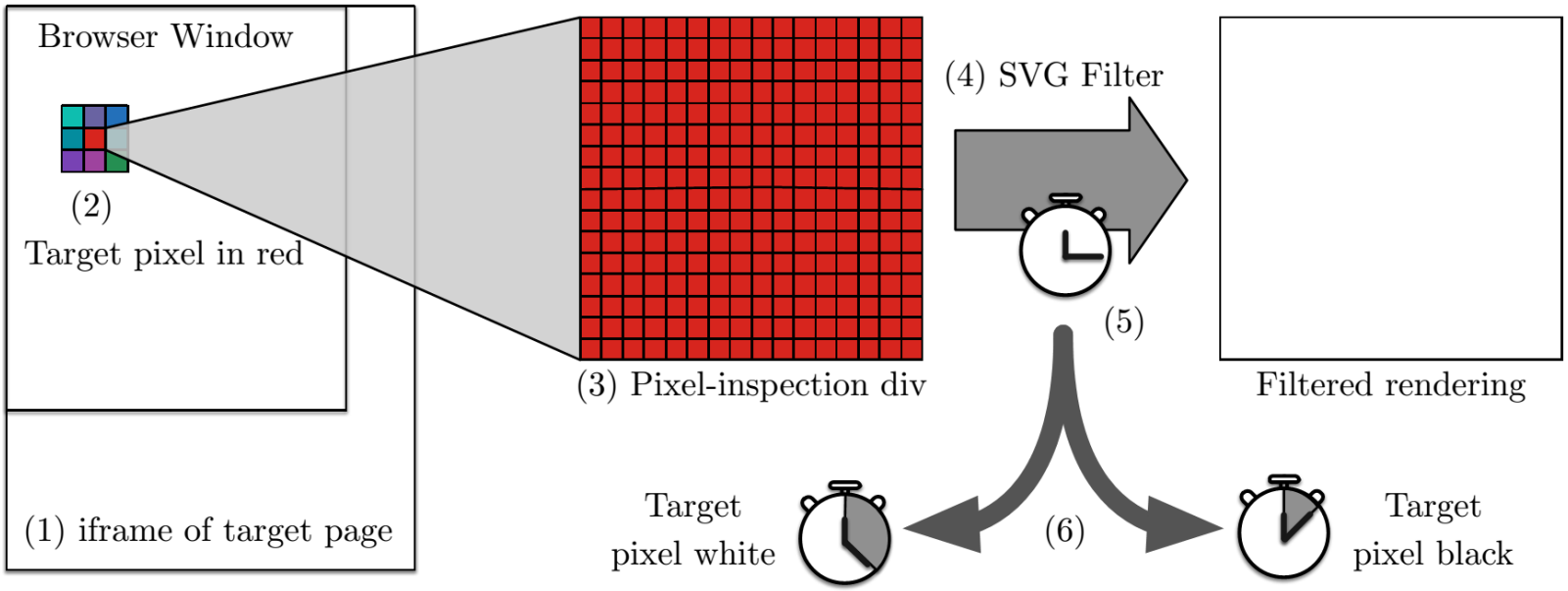
Firefox SVG Filters and Subnormals

```
def do_one_convolve(source_patch):
  source_patch = [[ ?, ?, ? ]
                  [ ?, ?, ? ]
                  [ ?, ?, ? ]]

  kernel = [[ 1e-42, 1e-42, 1e-42 ]
            [ 1e-42, 1e-42, 1e-42 ]
            [ 1e-42, 1e-42, 1e-42 ]]

  for x,y in source_patch:
      tmp[x][y] = source_patch[x][y] * kernel[x][y]

  result = 0

  for x,y in tmp:
      result += tmp[x][y]

  return result

for x,y in source_image:
  source_image[x][y] = do_one_convolve(swatch(x,y))
```

$s + s$

0+0

Firefox SVG Filter Timing Attack

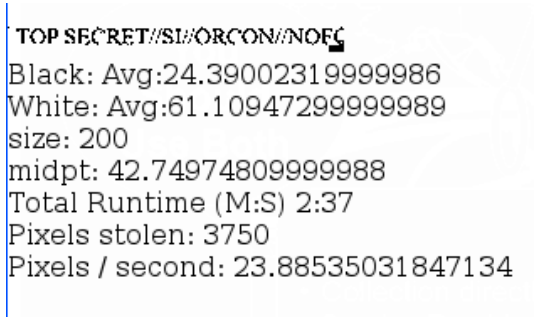


Firefox SVG Filters Attack Impact

- Firefox does not consider running SVG filters over foreign pixels a violation of SOP
  - We disagree
  - Cross Origin Resource Sharing (CORS) is the obvious solution

Reading Pixels

- From other origins
  - Reconstruct characters (OCR)
  - Extract usernames, login status, user information, etc
  - Blocked with frame options or CSP



- From *our* origin
  - History sniffing

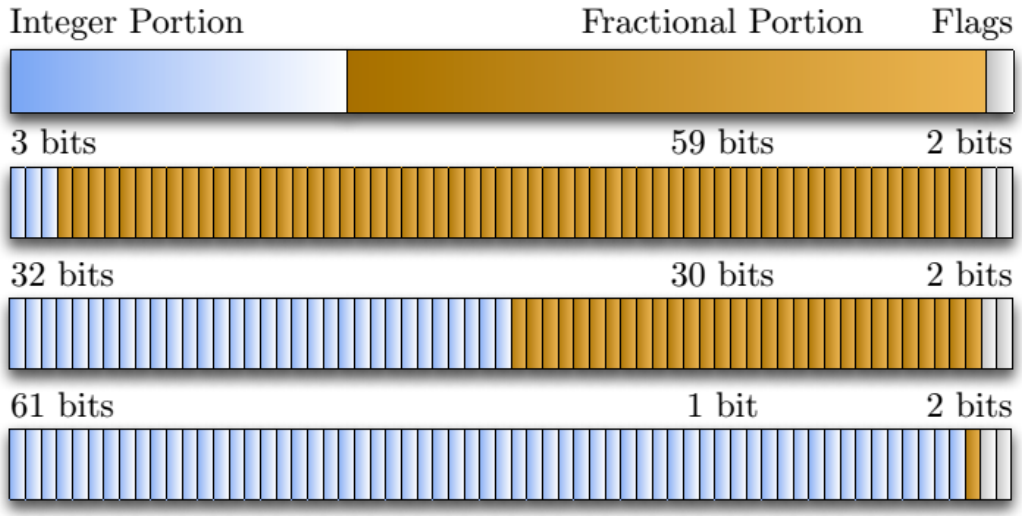
Avoiding Floating Point Problems

Recommendations

- Don't use floating point in security critical code
  - Unpredictable results
  - Large timing variations
  - Highly processor and build dependent
- Use Fixed Point if you need non-integer math

LibFTFP – Fixed Time Fixed Point

- C library implementing most math operations
  - Add, divide, etc
  - Transcendentals
  - Exponents, logs, etc
- Variable Width



- Constant time! (Probably!)

Building LibFTFP

- Techniques
  - No data dependent jumps (&&, `if`, etc.)
  - No known variable time instructions (`div`, `idiv`, etc.)
  - No look-up tables (due to caching)
- We cannot be 100% sure of the constant-ness of our code
  - Intel doesn't release any information about instruction data dependency
  - We cannot exhaustively test processors and instruction arguments
- Writing constant time code is a battle against all future processors and compilers
- LibFTFP uses approximations

LibFTFP Statistics

- Comparing to hardware slightly unfair
- Comparing to infinite precision software (MPFR) also slightly unfair

Function	FTFP	SSE	MPFR
neg	6	5	12-20
abs	9	4	10-17
cmp	21	5	10-15
add	15	4	15-58
sub	15	5	14-61
mul	43	5	16-76
div	381	7-15	15-170
floor	8	5	12-48
ceil	11	5	12-56
exp	1,460	7-16	37-13,330
ln	681	11-20	18-6,900
log2	679	9-20	19-24,000
log10	674	9-21	19-18,000
sqrt	7,870	7-16	9-154
pow	2,330	11-78	40-72,000
sin	1,998	–	11-33,000
cos	1,990	–	34-29,000
tan	2,380	–	13-37,000
print	443	350-600	210-230

github.com/kmowery/libfixedtimefixedpoint

Takeaways

- Security critical code should **omit floating point** or be extremely careful
- Writing **provably constant time code** is impossible
  - Intel? Some help here?
- Browsers should require **CORS/CSP for computing over all foreign data**
  - Like pixels

Future Work

- Firefox attack works on FF 23-27
  - Attack stopped working when filters changed to GPU
- GPU floating point implementations
  - "On NVIDIA GPUs starting with the Fermi architecture [...] multi-instruction sequences such as square root and [...] reciprocal square root, must do extra work and take a slower path for denormal values "
- Other math operation data side channels
  - `imul, div/idiv` cycle counts are data dependent
  - What can we break with that?

Questions?

dkohlbre@cs.ucsd.edu

LibFTFP: github.com/kmowery/libfixedtimefixedpoint