CSEP 564 : Computer Security and Privacy

Side-Channel Attacks

Fall 2022

David Kohlbrenner

dkohlbre@cs.washington.edu

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Logistics

- **Lab3a due next week**

- One more reading, different expectations for writeup
  - See Canvas

- Next week may be run slightly differently
  - Part of lecture may not be recorded, I'll post an announcement

Lab 3 discussion

- Slightly “role-playing” exercise
- For things like severity in the writeup, we are looking for a coherent argument, not a specific value.
- Try and treat this like a software project you work on!

Side-Channel Attacks

Side-channels: conceptually

- A program's implementation (that is, the final compiled version) is different from the conceptual description

- Side-effects of the difference between the implementation and conception can reveal unexpected information
  - Thus: Side-channels

Detour: Covert-channels

- We'll see many unusual ways to have information flow from thing A to thing B
- If this is an *intentional* usage of side effects, it is a covert channel
- *Unintentional* means it is a side-channel
- The same *mechanism* can be used as a covert-channel, or abused as a side-channel

Side Channel Attacks

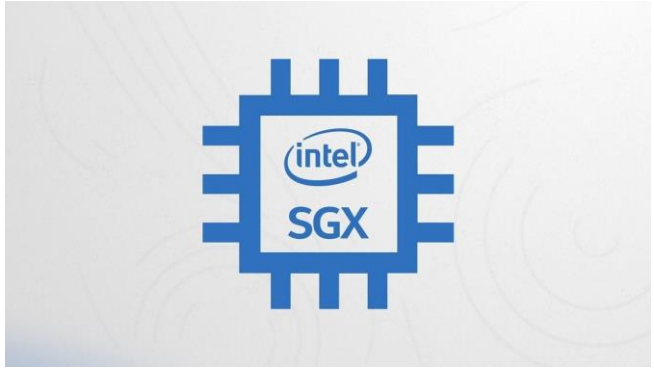
- Most commonly discussed in the context of cryptosystems
- But also prevalent in many contexts
    - E.g., we discussed the TENEX password implementation
    - E.g., we discussed browser fingerprinting

Why should we care about side-channels?

- Compromises happen via ‘simple’ methods
  - Phishing
  - Straight-forward attacks

- Embedded systems *do* see side-channel attacks

- “High Security” systems *do* see side-channel attacks



And they are getting more impactful…

- "The Secret Network has been vulnerable to the xAPIC and MMIO vulnerabilities that were publicly disclosed on August 9, 2022. These vulnerabilities could be used to extract the *consensus seed*, a master decryption key for the private transactions on the Secret Network. Exposure of the consensus seed would enable the complete retroactive disclosure of all Secret-4 private transactions since the chain began. We have helped Secret Network to deploy mitigations, especially the Registration Freeze on October 5, 2022."

Timing Side-Channels

- Duration of a program (or operation) reveals information
- TENEX case

password comparison

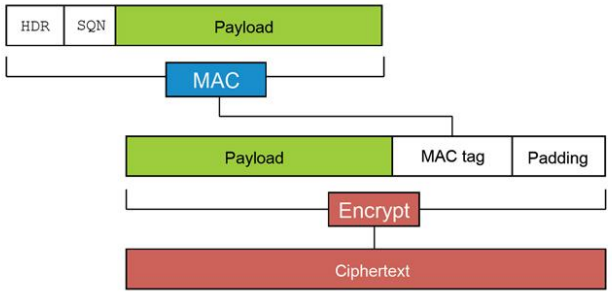
TENEX attack (for real)

- TENEX had an early *memory paging system*

- The original attack used page faults, not timing
  - Timing would’ve also worked ☺

Timing side-channels: round 2

- Cryptographic implementations fall down
  - #1 target for timing attacks
  - Extremely common to find vulnerabilities



- "Timing Attacks on Implementations of Diffe-Hellman, RSA, DSS, and Other Systems"
  - Was very far from the last paper on the topic

Attacking cryptography with side-channels

- ANY leakage is bad
  - E.g. 1 bit of key leaking is ‘catastrophic’
- Cryptographic implementations are complex
  - Many layers of protocols

Example Timing Attacks

- **RSA:** Leverage key-dependent timings of modular exponentiations
  - https://www.rambus.com/timing-attacks-on-implementations-of-diffie-hellman-rsa-dss-and-other-systems/
  - http://crypto.stanford.edu/~dabo/papers/ssl-timing.pdf
- **Block Ciphers:** Leverage key-dependent cache hits/misses

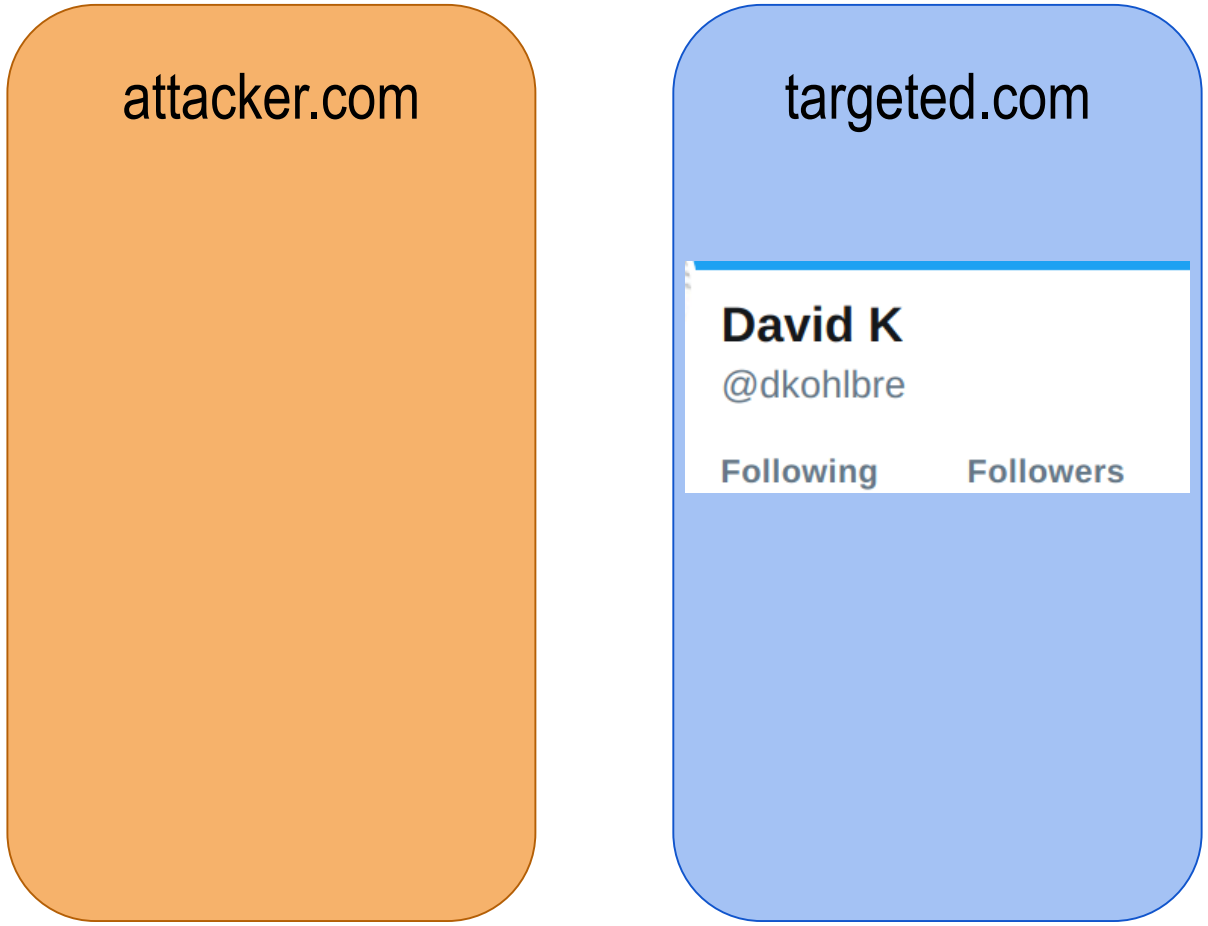
AES T-table

How odd can this get?

- Lets look at the sequel to Paul Stone's attacks

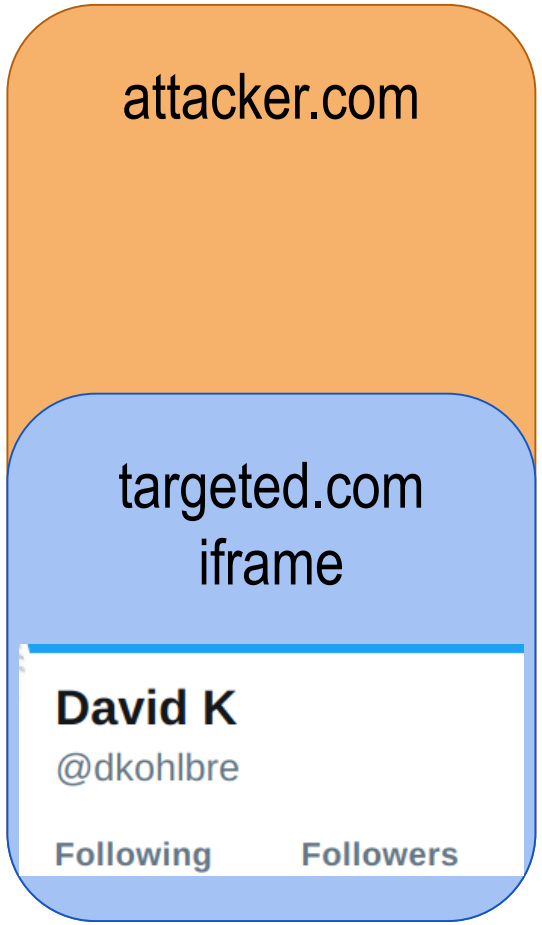
The SVG-filter pixel-stealing timing attack

- **Attacker**:
  - Hosts webpage
- **Victim**:
  - Visits attacker
  - Logged into target
- **Target**:
  - Website hosting private visual information



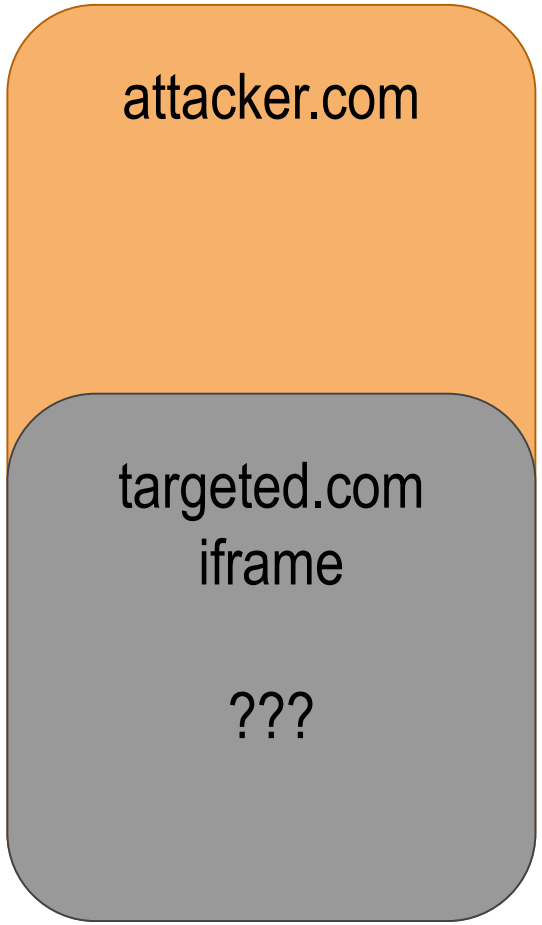
The SVG-filter pixel-stealing timing attack

- **Attacker**:
  - Hosts webpage
- **Victim**:
  - Visits attacker
  - Logged into target
- **Target**:
  - Website hosting private visual information



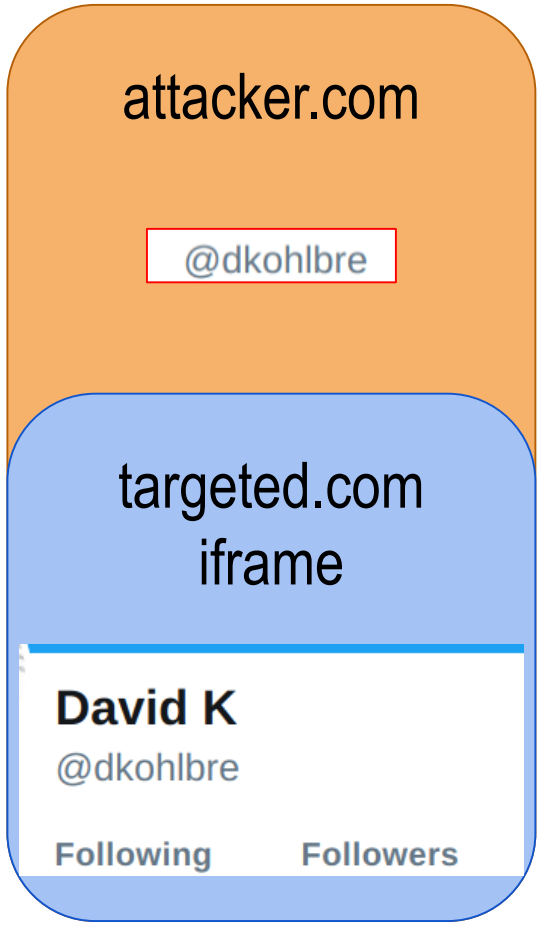
The SVG-filter pixel-stealing timing attack

- **Attacker**:
  - Hosts webpage
- **Victim**:
  - Visits attacker
  - Logged into target
- **Target**:
  - Website hosting private visual information

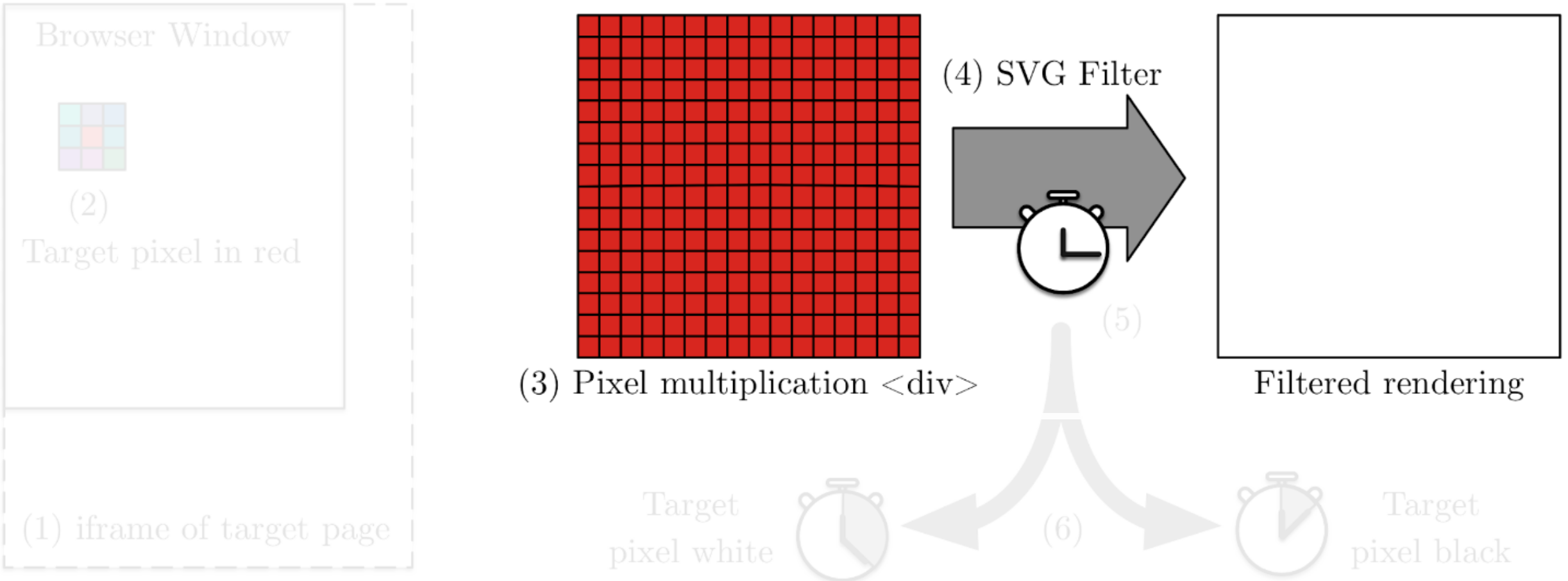


The SVG-filter pixel-stealing timing attack

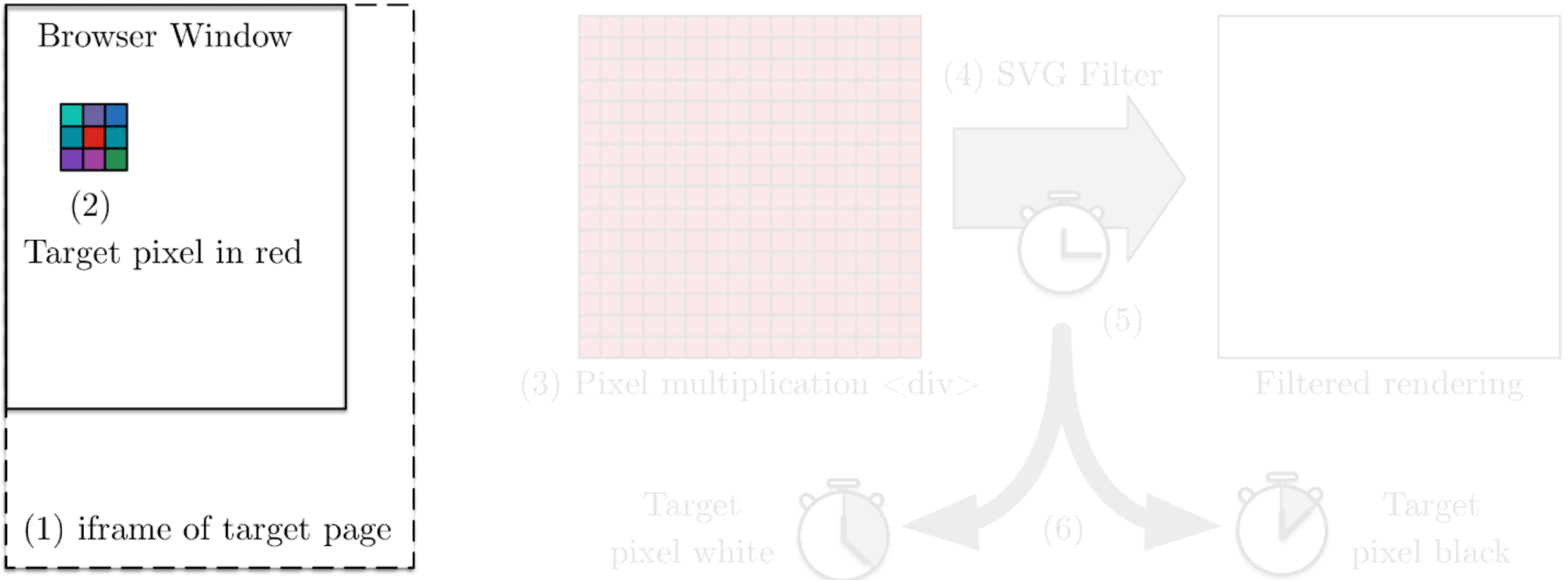
- **Attacker**:
  - Hosts webpage
- **Victim**:
  - Visits attacker
  - Logged into target
- **Target**:
  - Website hosting private visual information



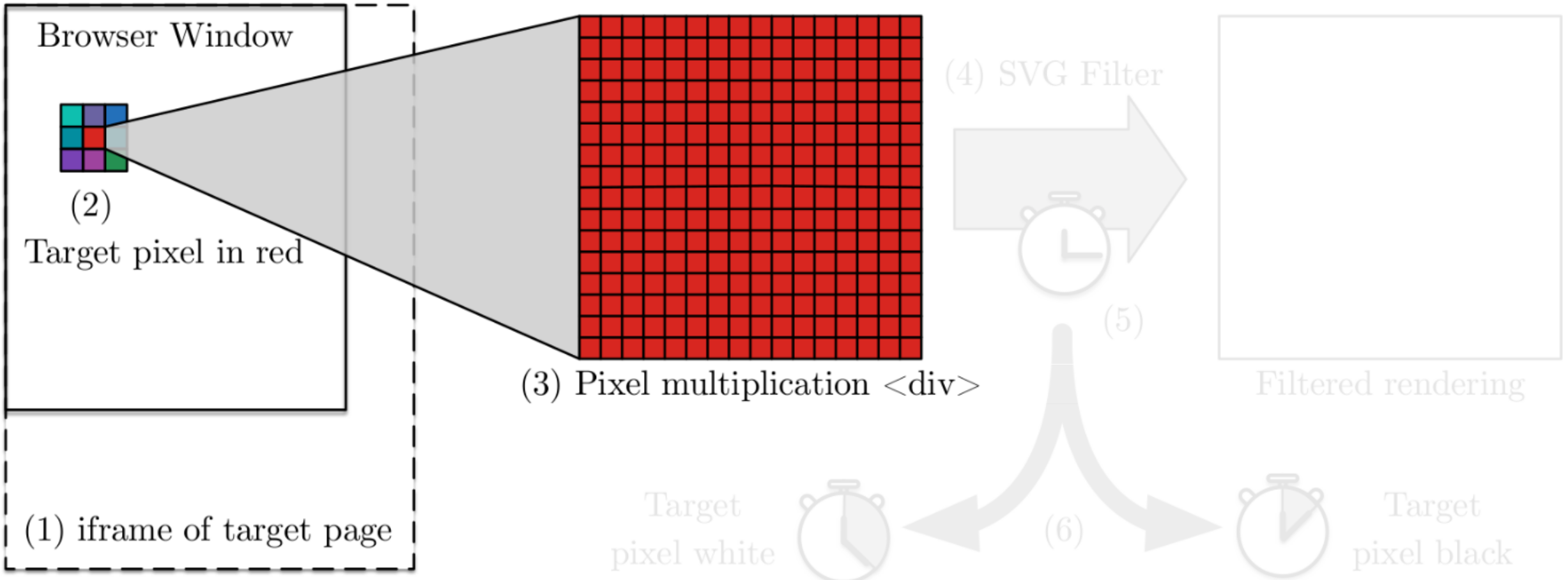
SVG-filter Pixel-stealing attack overview



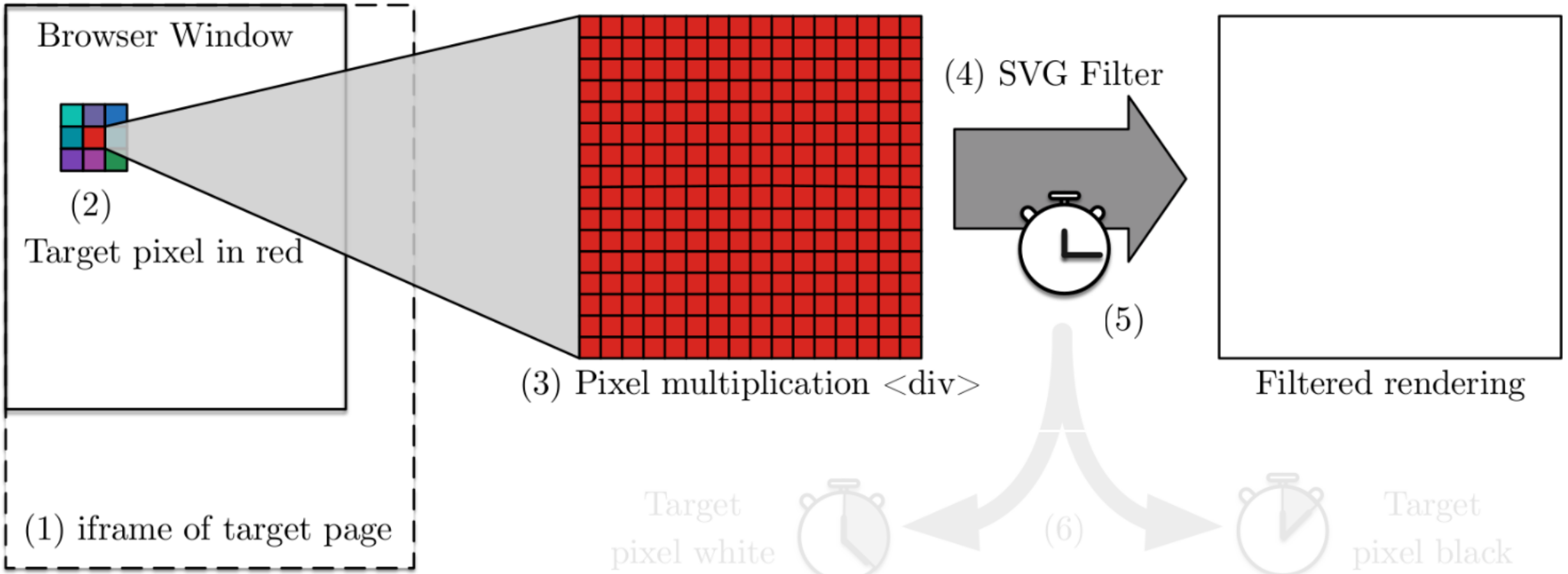
SVG-filter Pixel-stealing attack overview



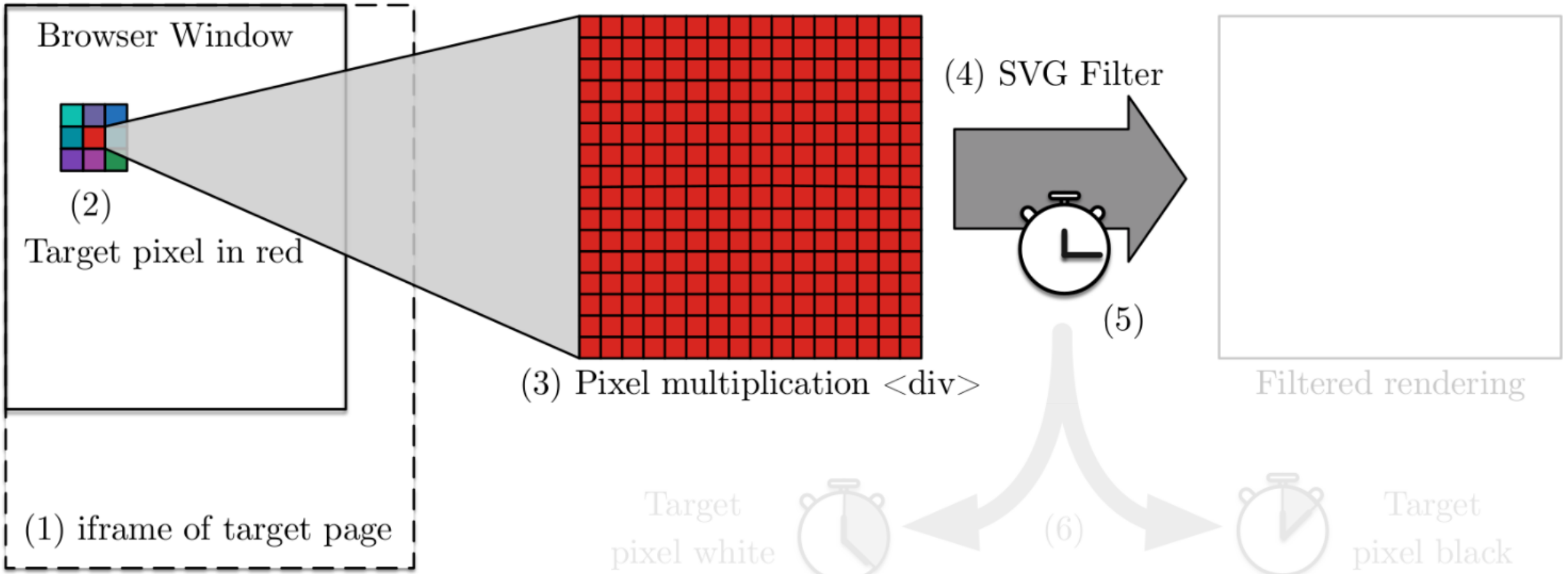
SVG-filter Pixel-stealing attack overview



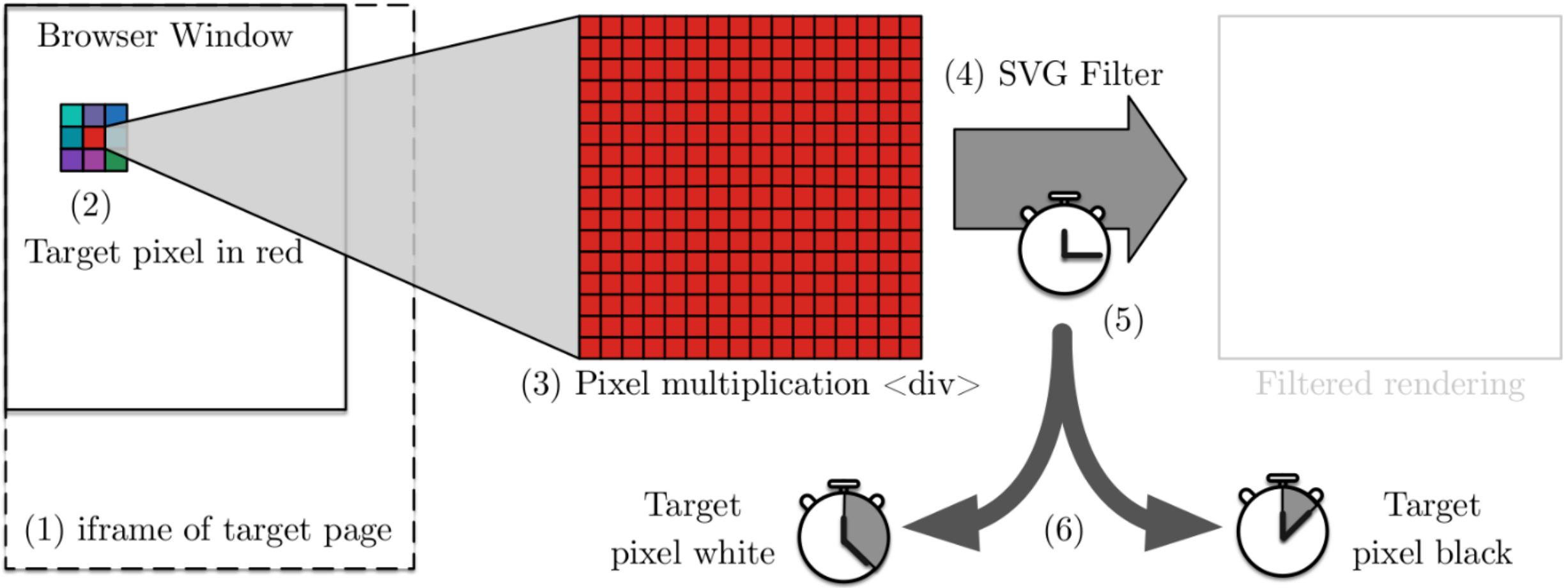
SVG-filter Pixel-stealing attack overview



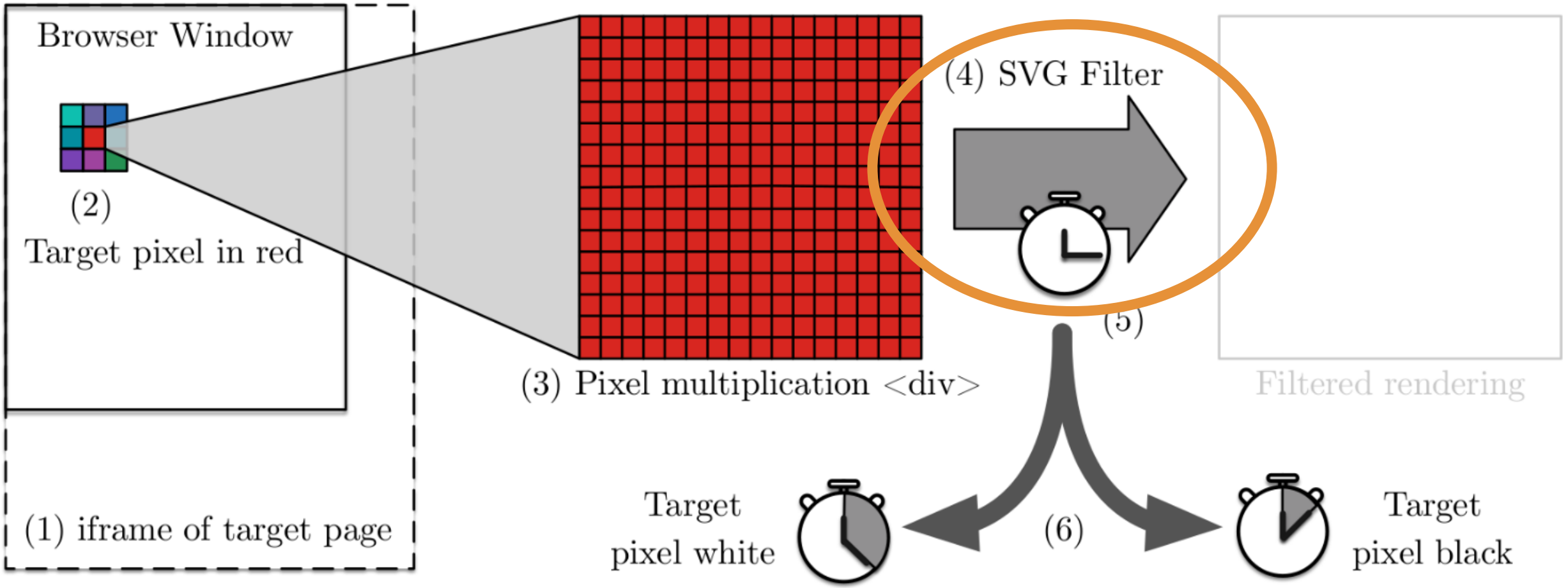
SVG-filter Pixel-stealing attack overview



SVG-filter Pixel-stealing attack overview



SVG-filter Pixel-stealing attack overview



Paul Stone’s Version

Stone, 2013

```
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++) {
    [...]
```

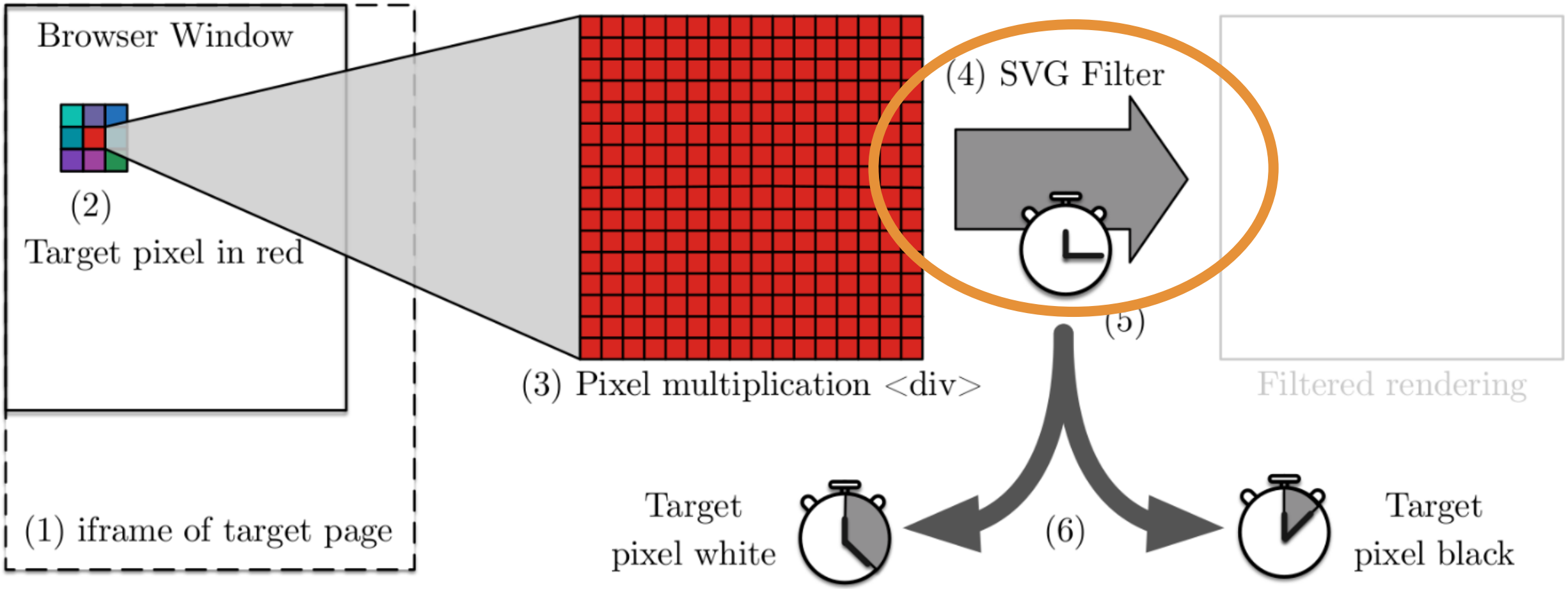
Stone, 2013

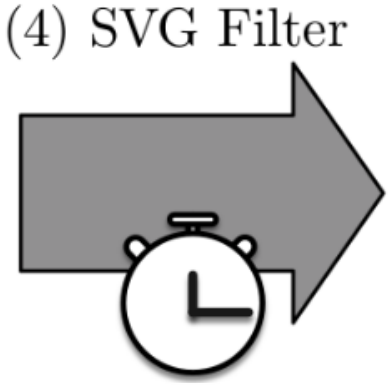
```
if (x == rect.x || xExt[0]  <= startX || xExt[1] <= startX ||
    xExt[2] <= startX || xExt[3] <= startX) {
  [...]
} else { // We only need to look at the newest column
  for (PRUint32 y1 = startY; y1 <= endY; y1++) {
    [...]
```

Stone, 2013

```
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++) {
    [...]
```

```c
// Constant-time max and min functions for unsigned arguments
static inline unsigned
umax(unsigned a, unsigned b)
{
  return a - ((a - b) & -(a < b));
}

static inline unsigned
umin(unsigned a, unsigned b)
{
  return a - ((a - b) & -(a > b));
}
```


Browser Window
(2)
Target pixel in red
(1) iframe of target page
(3) Pixel multiplication
(4) SVG Filter
(5)
Filtered rendering
Target
pixel white
(6)
Target
pixel black



Implemented with *Floating-point math*

Variable time instructions?

Intel i5-4460 double-precision floating-point multiply

Cycle count

	0.0	1.0	1e10	1e+200	1e-300	1e-42	256	257	1e-320
0.0	6.59	6.56	6.59	6.58	6.58	6.57	6.58	6.59	6.57
1.0	6.57	6.59	6.55	6.57	6.57	6.56	6.56	6.56	130.89
1e10	6.55	6.55	6.56	6.58	6.56	6.56	6.56	6.57	130.95
1e+200	6.55	6.57	6.56	6.58	6.59	6.53	6.55	6.58	130.92
1e-300	6.51	6.57	6.56	6.59	6.57	6.57	6.55	6.58	6.54
1e-42	6.55	6.57	6.55	6.57	6.55	6.58	6.58	6.58	6.55
256	6.58	6.53	6.56	6.54	6.56	6.56	6.58	6.57	130.94
257	6.59	6.57	6.60	6.56	6.58	6.56	6.57	6.59	130.90
1e-320	6.59	130.90	130.92	130.94	6.59	6.58	130.95	130.91	6.56

Intel i5-4460 double-precision floating-point multiply

	0.0	1.0	1e10	1e+200	1e-300	1e-42	256	257	1e-320
	Cycle count								
0.0	6.59	6.56	6.59	6.58	6.58	6.57	6.58	6.59	6.57
1.0	6.57	6.59	6.55	6.57	6.57	6.56	6.56	6.56	130.89
1e10	6.55	6.55	6.56	6.58	6.56	6.56	6.56	6.57	130.95
1e+200	6.55	6.57	6.56	6.58	6.59	6.53	6.55	6.58	130.92
1e-300	6.51	6.57	6.56	6.59	6.57	6.57	6.55	6.58	6.54
1e-42	6.55	6.57	6.55	6.57	6.55	6.58	6.58	6.58	6.55
256	6.58	6.53	6.56	6.54	6.56	6.56	6.58	6.57	130.94
257	6.59	6.57	6.60	6.56	6.58	6.56	6.57	6.59	130.90
1e-320	6.59	130.90	130.92	130.94	6.59	6.58	130.95	130.91	6.56

Intel i5-4460 double-precision floating-point multiply

"secret"	0.0	1.0	1e10	1e+200	1e-300	1e-42	256	257	1e-320
	Cycle count								
0.0	6.59	6.56	6.59	6.58	6.58	6.57	6.58	6.59	6.57
1.0	6.57	6.59	6.55			6.56	6.56	6.56	130.89
1e10	6.55	6.55	6.56			6.56	6.56	6.57	130.95
1e+200	6.55	6.57	6.56			6.53	6.55	6.58	130.92
1e-300	6.51	6.57	6.56			6.57	6.55	6.58	6.54
1e-42	6.55	6.57	6.55	6.57	6.55	6.58	6.58	6.58	6.55
256	6.58	6.53	6.56	6.54	6.56	6.56	6.58	6.57	130.94
257	6.59	6.57	6.60	6.56	6.58	6.56	6.57	6.59	130.90
1e-320	6.59	130.90	130.92	130.94	6.59	6.58	130.95	130.91	6.56

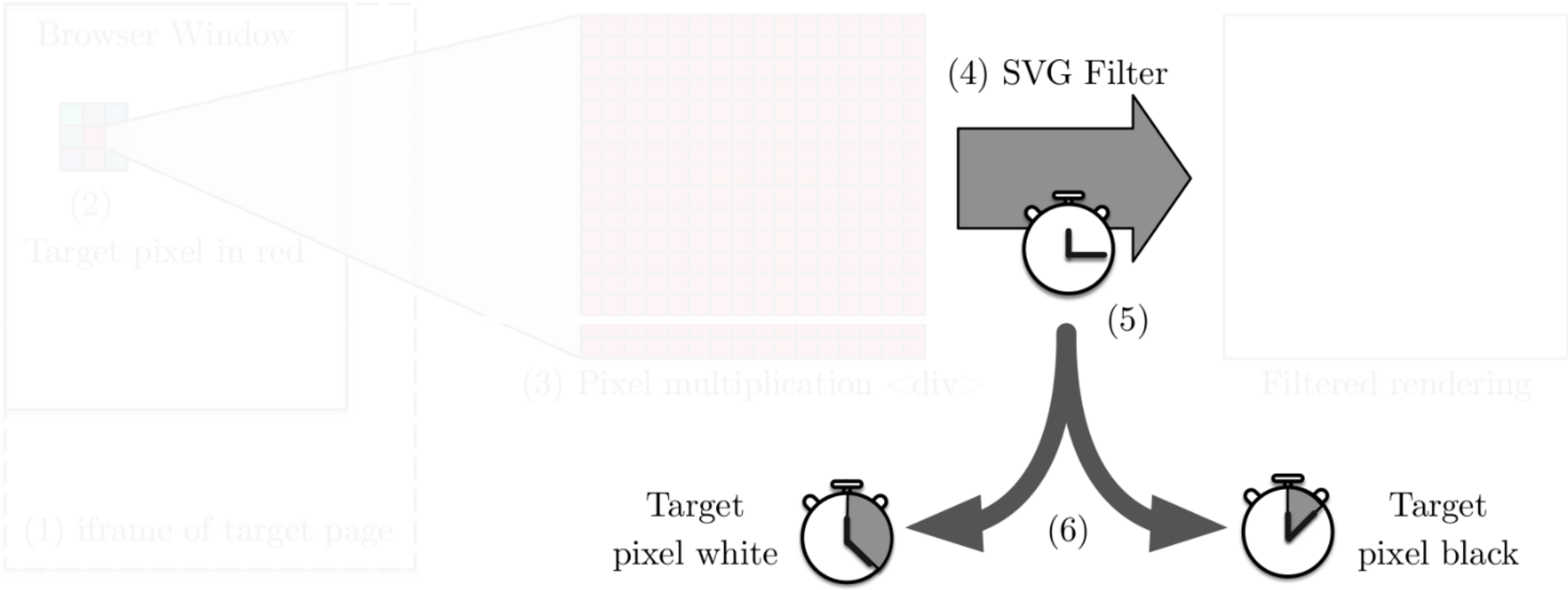
secret x 1e-320

Intel i5-4460 double-precision floating-point multiply

	0.0	1.0	1e10	1e+200	1e-300	1e-42	256	257	1e-320
	Cycle count								
0.0	6.59	6.56	6.59	6.58	6.58	6.57	6.58	6.59	6.57
1.0	6.57	6.59	6.55	6.57	6.57	6.56	6.56	6.56	130.89
1e10	6.55	6.55	6.56	6.58	6.56	6.56	6.56	6.57	130.95
1e+200	6.55	6.57	6.56	6.58	6.59	6.53	6.55	6.58	130.92
1e-300	6.51	6.57	6.56	6.59	6.57	6.57	6.55	6.58	6.54
1e-42	6.55	6.57	6.55	6.57	6.55	6.58	6.58	6.58	6.55
256	6.58	6.53	6.56	6.54	6.56	6.56	6.58	6.57	130.94
257	6.59	6.57	6.60	6.56	6.58	6.56	6.57	6.59	130.90
1e-320	6.59	130.90	130.92	130.94	6.59	6.58	130.95	130.91	6.56

Intel i5-4460 double-precision floating-point divide

Dividend	Divisor: 0.0	1.0	1e10	1e+200	1e-300	1e-42	256	257	1e-320
	Cycle count								
0.0	6.56	6.59	6.58	6.55	6.57	6.58	6.57	6.57	6.59
1.0	6.58	6.58	12.19	12.17	12.22	12.24	6.57	12.24	165.76
1e10	6.58	6.55	12.25	12.20	12.23	12.25	6.57	12.22	165.81
1e+200	6.60	6.60	12.25	12.20	12.22	12.22	6.58	12.24	165.79
1e-300	6.59	6.57	175.22	12.24	12.17	12.22	6.52	12.23	165.83
1e-42	6.60	6.53	12.23	12.22	12.21	12.24	6.58	12.21	165.79
256	6.57	6.55	12.24	12.20	12.20	12.20	6.53	12.22	165.79
257	6.55	6.58	12.24	12.22	12.24	12.23	6.56	12.21	165.80
1e-320	6.56	150.73	165.79	6.59	165.78	165.76	150.66	165.80	165.78


Browser Window
(2)
Target pixel in red
(1) iframe of target page
(3) Pixel multiplication
(4) SVG Filter
(5)
Filtered rendering
Target
pixel white
(6)
Target
pixel black

Can we find unsafe math operations?

Firefox lighting code - Core loop

```
int32_t sourceIndex = y * sourceStride + x;
int32_t targetIndex = y * targetStride + 4 * x;

Point3D normal = GenerateNormal(sourceData, sourceStride,
                                x, y, mSurfaceScale,
                                aKernelUnitLengthX, aKernelUnitLengthY);

IntPoint pointInFilterSpace(aRect.x + x, aRect.y + y);
Float Z = mSurfaceScale * sourceData[sourceIndex] / 255.0f;
Point3D pt(pointInFilterSpace.x, pointInFilterSpace.y, Z);
Point3D rayDir = mLight.GetVectorToLight(pt);
uint32_t color = mLight.GetColor(lightColor, rayDir);
```

Firefox lighting code - Core loop

```
int32_t sourceIndex = y * sourceStride + x;
int32_t targetIndex = y * targetStride + 4 * x;

Point3D normal = GenerateNormal(sourceData, sourceStride,
                                x, y, mSurfaceScale,
                                aKernelUnitLengthX, aKernelUnitLengthY);

IntPoint pointInFilterSpace(aRect.x + x, aRect.y + y);
Float Z = mSurfaceScale * sourceData[sourceIndex] / 255.0f;
Point3D pt(pointInFilterSpace.x, pointInFilterSpace.y, Z);
Point3D rayDir = mLight.GetVectorToLight(pt);
uint32_t color = mLight.GetColor(lightColor, rayDir);
```

Firefox lighting code - Core loop

```
int32_t sourceIndex = y * sourceStride + x;
int32_t targetIndex = y * targetStride + 4 * x;

Point3D normal = GenerateNormal(sourceData, sourceStride,
                                x, y, mSurfaceScale,
                                aKernelUnitLengthX, aKernelUnitLengthY);

IntPoint pointInFilterSpace(aRect.x + x, aRect.y + y);
Float Z = mSurfaceScale * sourceData[sourceIndex] / 255.0f;
Point3D pt(pointInFilterSpace.x, pointInFilterSpace.y, Z);
Point3D rayDir = mLight.GetVectorToLight(pt);
uint32_t color = mLight.GetColor(lightColor, rayDir);
```

Firefox lighting code - Core loop

```
int32_t sourceIndex = y * sourceStride + x;
int32_t targetIndex = y * targetStride + 4 * x;

Point3D normal = GenerateNormal(sourceData, sourceStride,
                                x, y, mSurfaceScale,
                                aKernelUnitLengthX, aKernelUnitLengthY);

IntPoint pointInFilterSpace(aRect.x + x, aRect.y + y);
Float Z = mSurfaceScale * sourceData[sourceIndex] / 255.0f;
Point3D pt(pointInFilterSpace.x, pointInFilterSpace.y, Z);
Point3D rayDir = mLight.GetVectorToLight(pt);
uint32_t color = mLight.GetColor(lightColor, rayDir);
```

Firefox lighting code - Core loop

```
int32_t sourceIndex = y * sourceStride + x;
int32_t targetIndex = y * targetStride + 4 * x;

Point3D normal = GenerateNormal(sourceData, sourceStride,
                                x, y, mSurfaceScale,
                                aKernelUnitLengthX, aKernelUnitLengthY);

IntPoint pointInFilterSpace(aRect.x + x, aRect.y + y);
Float Z = mSurfaceScale * sourceData[sourceIndex] / 255.0f;
Point3D pt(pointInFilterSpace.x, pointInFilterSpace.y, Z);
Point3D rayDir = mLight.GetVectorToLight(pt);
uint32_t color = mLight.GetColor(lightColor, rayDir);
```

Attacker JS parameter

Firefox lighting code - Core loop

```
int32_t sourceIndex = y * sourceStride + x;
int32_t targetIndex = y * targetStride + 4 * x;

Point3D normal = GenerateNormal(sourceData, sourceStride,
                                x, y, mSurfaceScale,
                                aKernelUnitLengthX, aKernelUnitLengthY);

IntPoint pointInFilterSpace(aRect.x + x, aRect.y + y);
Float Z = mSurfaceScale * sourceData[sourceIndex] / 255.0f;
Point3D pt(pointInFilterSpace.x, pointInFilterSpace.y, Z);
Point3D rayDir = mLight.GetVectorToLight(pt);
uint32_t color = mLight.GetColor(lightColor, rayDir);
```

Attacker JS parameter

Secret Pixel data

Firefox lighting code - Core loop

```
int32_t sourceIndex = y * sourceStride + x;
int32_t targetIndex = y * targetStride + 4 * x;

Point3D normal = GenerateNormal(sourceData, sourceStride,
                                x, y, mSurfaceScale,
                                aKernelUnitLengthX, aKernelUnitLengthY);

IntPoint pointInFilterSpace(aRect.x + x, aRect.y + y);
Float Z = mSurfaceScale * sourceData[sourceIndex] / 255.0f;
Point3D pt(pointInFilterSpace.x, pointInFilterSpace.y, Z);
Point3D rayDir = mLight.GetVectorToLight(pt);
uint32_t color = mLight.GetColor(lightColor, rayDir);
```

Attacker JS parameter

subnormal

Secret Pixel data

{1,0}

Firefox lighting code - Core loop

```
int32_t sourceIndex = y * sourceStride + x;
int32_t targetIndex = y * targetStride + 4 * x;

Point3D normal = GenerateNormal(sourceData, sourceStride,
                                x, y, mSurfaceScale,
                                aKernelUnitLengthX, aKernelUnitLengthY);

IntPoint pointInFilterSpace(aRect.x + x, aRect.y + y);
Float Z = mSurfaceScale * sourceData[sourceIndex] / 255.0f;
Point3D pt(pointInFilterSpace.x, pointInFilterSpace.y, Z);
Point3D rayDir = mLight.GetVectorToLight(pt);
uint32_t color = mLight.GetColor(lightColor, rayDir);
```

1e-42

0.0	6.59
1.0	130.85

Attacker JS parameter
subnormal

Secret Pixel data
{1,0}

Firefox lighting code - Core loop

```
int32_t sourceIndex = y * sourceStride + x;
int32_t targetIndex = y * targetStride + 4 * x;

Point3D normal = GenerateNormal(sourceData, sourceStride,
                                x, y, mSurfaceScale,
                                aKernelUnitLengthX, aKernelUnitLengthY);

IntPoint pointInFilterSpace(aRect.x + x, aRect.y + y);
Float Z = mSurfaceScale * sourceData[sourceIndex] / 255.0f;
Point3D pt(pointInFilterSpace.x, pointInFilterSpace.y, Z);
Point3D rayDir = mLight.GetVectorToLight(pt);
uint32_t color = mLight.GetColor(lightColor, rayDir);
```

1e-42

255

0.0	6.59
1.0	130.85

0.0	6.58
1e-42	152.55

Attacker JS parameter
subnormal

Secret Pixel data
{1,0}

Firefox lighting code - Core loop

```
int32_t sourceIndex = y * sourceStride + x;
int32_t targetIndex = y * targetStride + 4 * x;

Point3D normal = GenerateNormal(sourceData, sourceStride,
                                x, y, mSurfaceScale,
                                aKernelUnitLengthX, aKernelUnitLengthY);

IntPoint pointInFilterSpace(aRect.x + x, aRect.y + y);
Float Z = mSurfaceScale * sourceData[sourceIndex] / 255.0f;
Point3D pt(pointInFilterSpace.x, pointInFilterSpace.y, Z);
Point3D rayDir = mLight.GetVectorToLight(pt);
uint32_t color = mLight.GetColor(lightColor, rayDir);
```

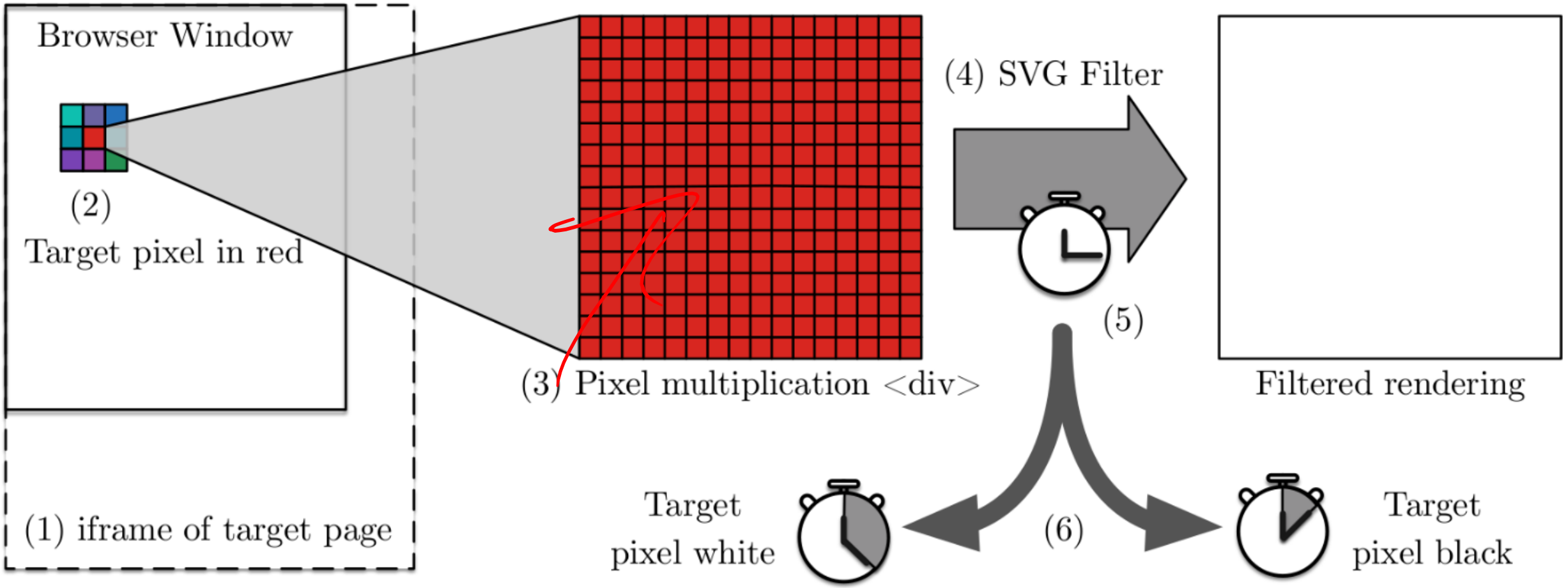
	1e-42		255
13 cycles	0.0	6.59	0.0	6.58
283 cycles	1.0	130.85	1e-42	152.55

Attacker JS parameter

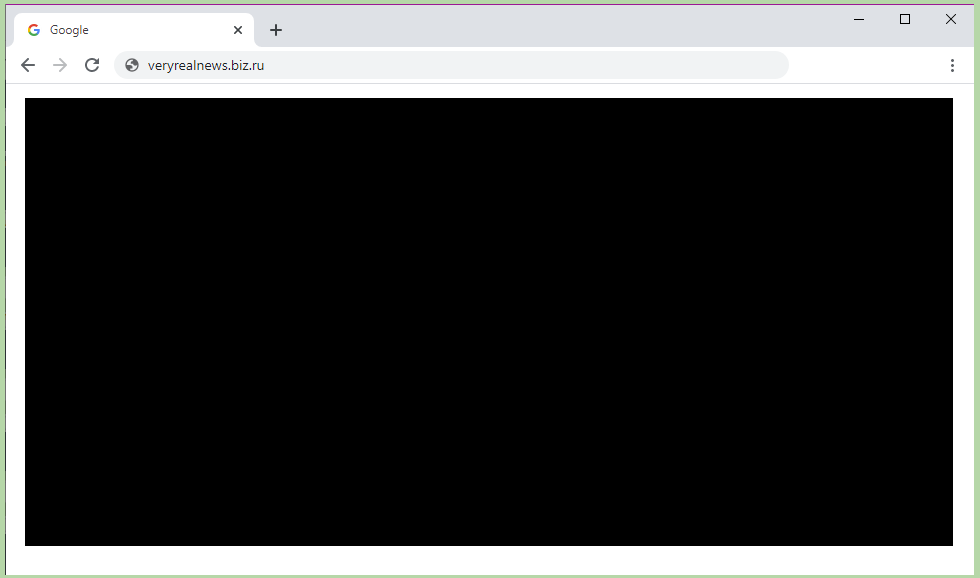
subnormal

Secret Pixel data

{1,0}


Browser Window
(2)
Target pixel in red
(1) iframe of target page
(3) Pixel multiplication
(4) SVG Filter
(5)
Filtered rendering
Target
pixel white
(6)
Target
pixel black

Attack in Action



Attacker's Web Server

Pixel stealing takeaways

- Combines web security, hardware knowledge, and software design
- Side-channels are real, and viable ☺
- And they just keep coming back

X-frame-options

deny

sameorigin

Aside: Power side-channels

Power-side channels

- The amount of *power* used by a computer is related to what it is doing
- How can you use this?
  - Think broadly.
  - What if power is only coarsely related to work? (E.g. doing a GPU operation vs a CPU one)
  - What if power is very finely related to work? (E.g. adding 0+0 takes less power than adding 0xffffffff + 0xffffffff)
- How might you, the attacker, measure power usage?

Platypus RAPL

Cache side-channels

Cache side-channels

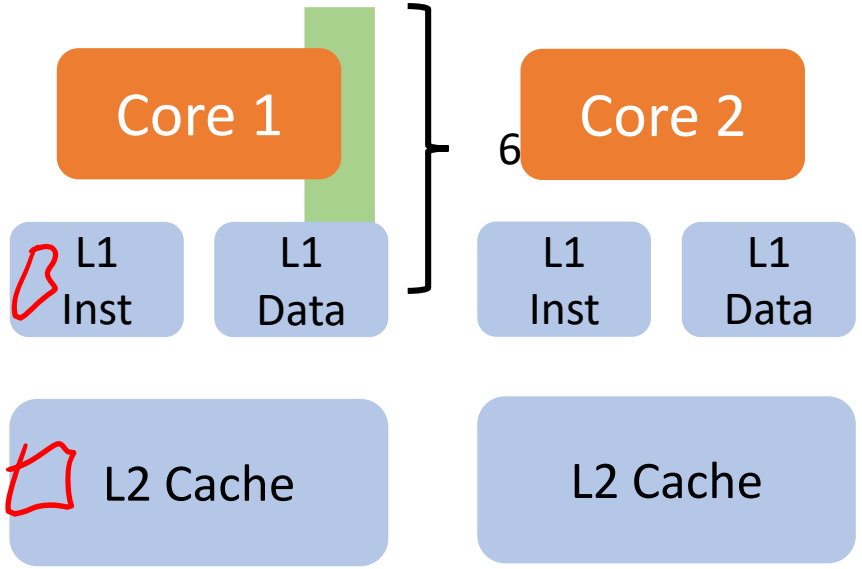
- **Idea**: The cache's current state implies something about prior memory accesses

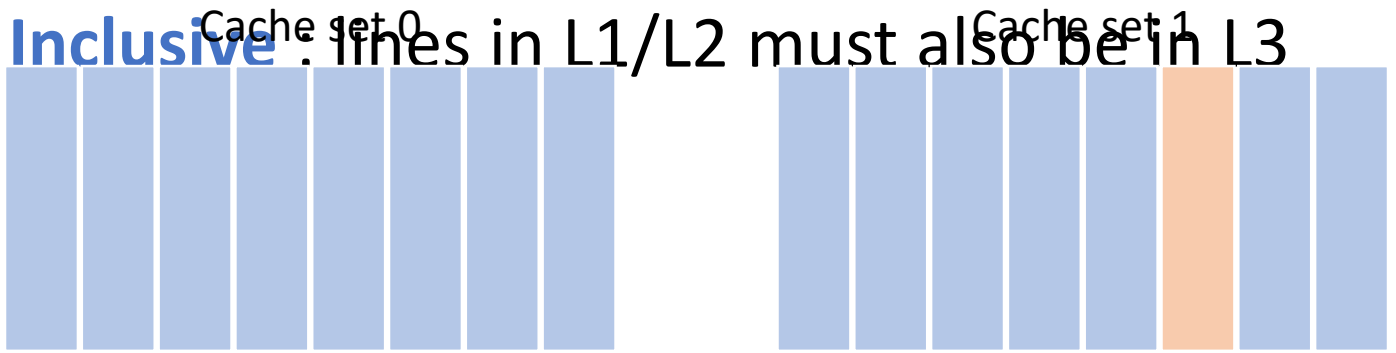
- **Insight**: Prior memory accesses can tell you a lot about a program!

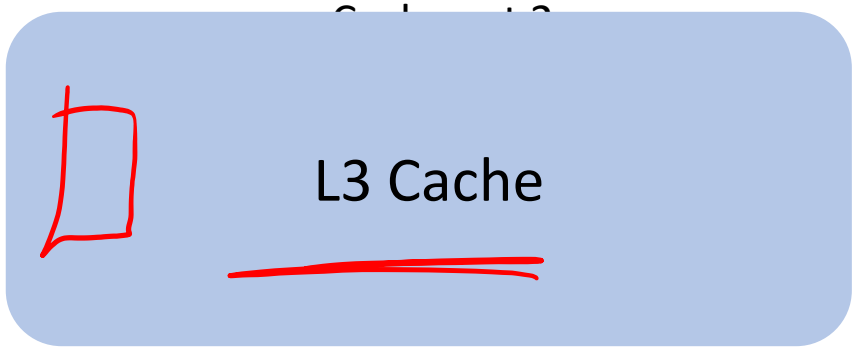
Many thanks to Craig Disselkoen for the animations.

Cache Basics

- **Cache lines** : fixed-size units of data
- **Cache set** : holds multiple cache lines
- **Set index** : assigns cache line to cache set
- **Eviction** : removing cache lines to make room
- **L1, L2, L3** : different levels of cache
- **Inclusive** : lines in L1/L2 must also be in L3

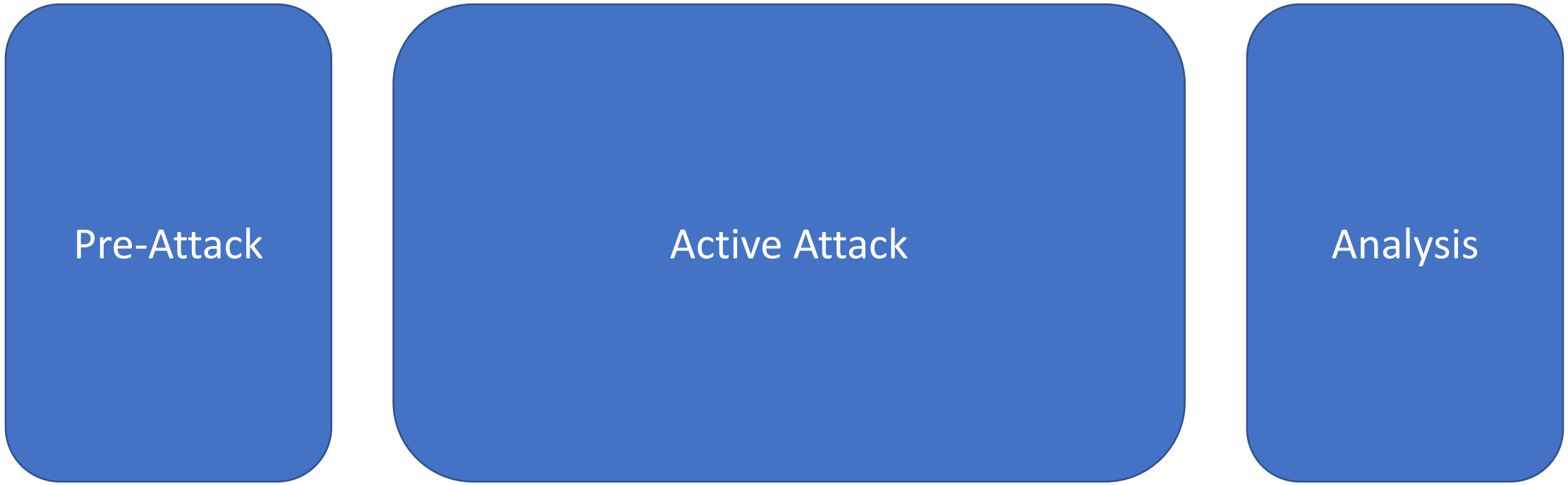




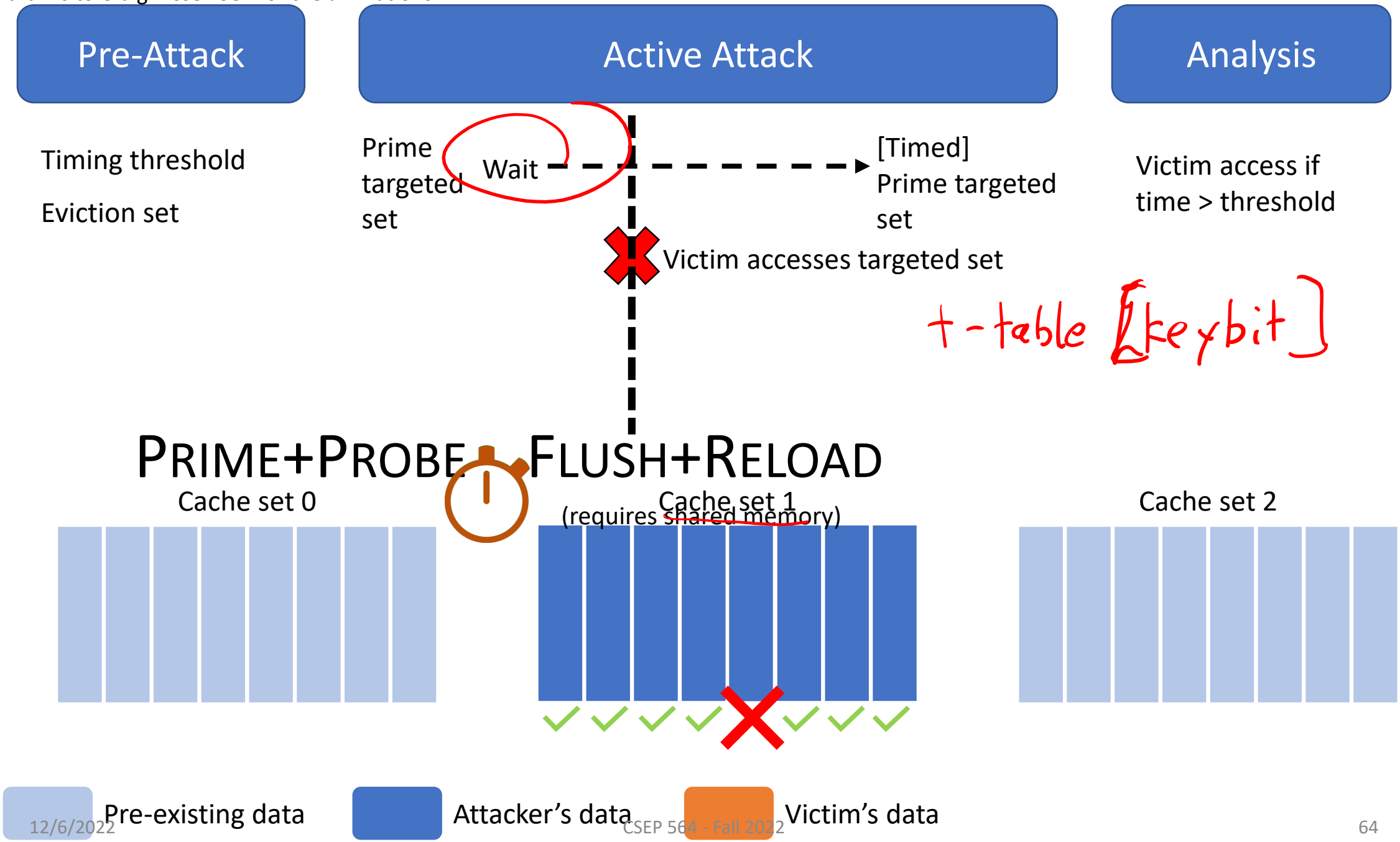


Many thanks to Craig Disselkoen for the animations.

Cache Attacks: Structure

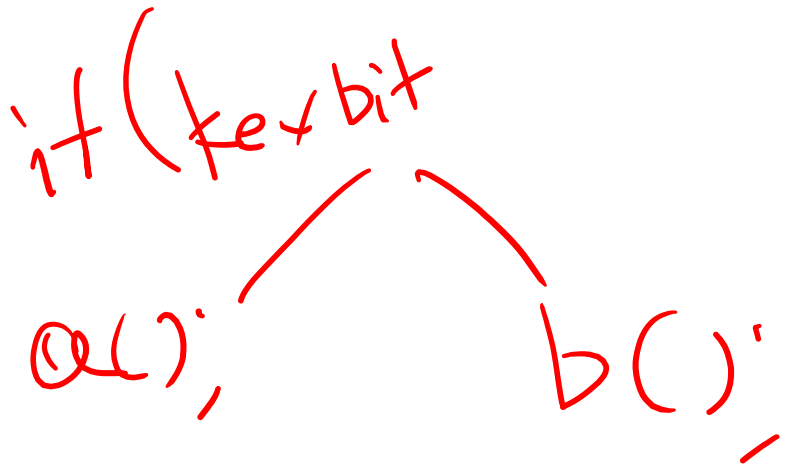


Many thanks to Craig Disselkoen for the animations.



FLUSH + RELOAD

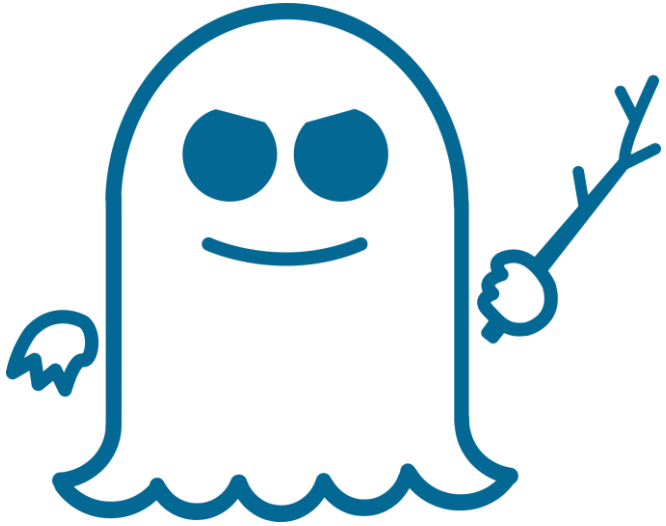
- Even simpler!
- Kick line L out of cache
- Let victim run
- Access L
  - Fast? Victim touched it
  - Slow? Victim didn't touch it



Cache attacks wrapup

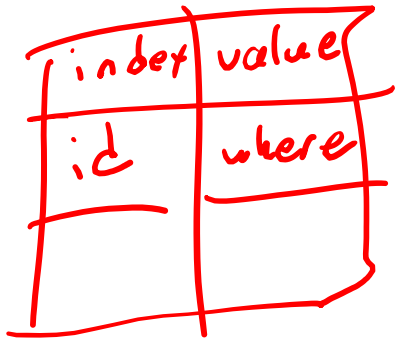
7:40

- Cache attacks are a core element of many side-channels
- Generally “assumed to work” these days
- New variations/tricks/mitigations published constantly
- Randomized caches are the current hotness



Speculative Execution Attacks – Spectre & Co.

Paper Discussion Time!



“Spectre Attacks: Exploiting Speculative Execution”

Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom

Choose one/more and discuss with neighbors:

- What does Variant 1 (Bounds-check-bypass) let an attacker do?
- What does Variant 2 (Indirect Branches) let an attacker do?
- Why is a cache side-channel critical to these attacks?
- What code might contain a Spectre ‘gadget’?
- Speculative execution was a known optimization for 20 years, why now?

Spectre + Friends

- First reported in 2017
- Disclosed in 2018
  - https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html
- Novel class of attack: speculative execution attacks
  - Aka: Spectre-class attacks
- (Academic paper published 2019… long story)

Speculative Execution (the fast version)

- All modern processors are capable of speculative execution
- How much, in what ways, and when differs
- Speculative execution allows a processor to ‘guess’ about the result of an instruction
  - And either confirm or correct itself later
- A branch predictor bases a guess on the program’s previous behavior

Example: Speculate on branch

```
int foo(int* address){
     int y = globalarray[0];
     int x = *address;
     if( x < 100 ){
          y = globalarray[10];
     }
     return y;
}
```

Example: Speculate on branch

```
int foo(int* address){
     int y = globalarray[0];
     int x = *address;
     if( x < 100 ){
          y = globalarray[10];
     }
     return y;
}
```

Example: Speculate on branch

```
int foo(int* address){
     int y = globalarray[0];
     int x = *address;
     if( x < 100 ){
          y = globalarray[10];
     }
     return y;
}
```

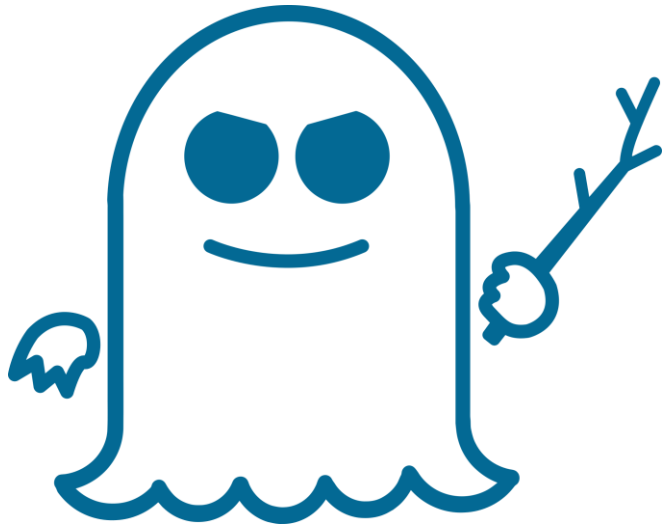
Example: Speculate on *indirect* branch

```
int caller(int(*fptr)()){
    int y = fptr();

    return y;
}
```

```
int foo(){
     return 10;
}

int bar(){
     return 0;
}
```

What happens when we speculate wrong?

- Eventually, a *squash* occurs
  - All work done under the incorrect guess is undone
- Bad guess on branch?
  - Undo everything in the branch!
  - Undo everything related!
- World reverts back to before guess …almost



Example: Speculate on branch

```
int foo(int* address){
     int y = globalarray[0];  // Brought into cache
     int x = *address; // Brought into cache
     if( x < 100 ){
          y = globalarray[10]; // Brought into cache maybe
     }
     return y;
}
```

Speculative attacks

Three stages:

1. Mistrain predictor

2. Run mistrained code with adversarial input

3. Recover leftover state information

Spectre variant 1

- “Bounds-check bypass”

```
if( x < len(array))
      array[x];
```

Spectre variant 1

- “Bounds-check bypass”

```
if( x < len(array))
      array2[array[x] * 4096];
```

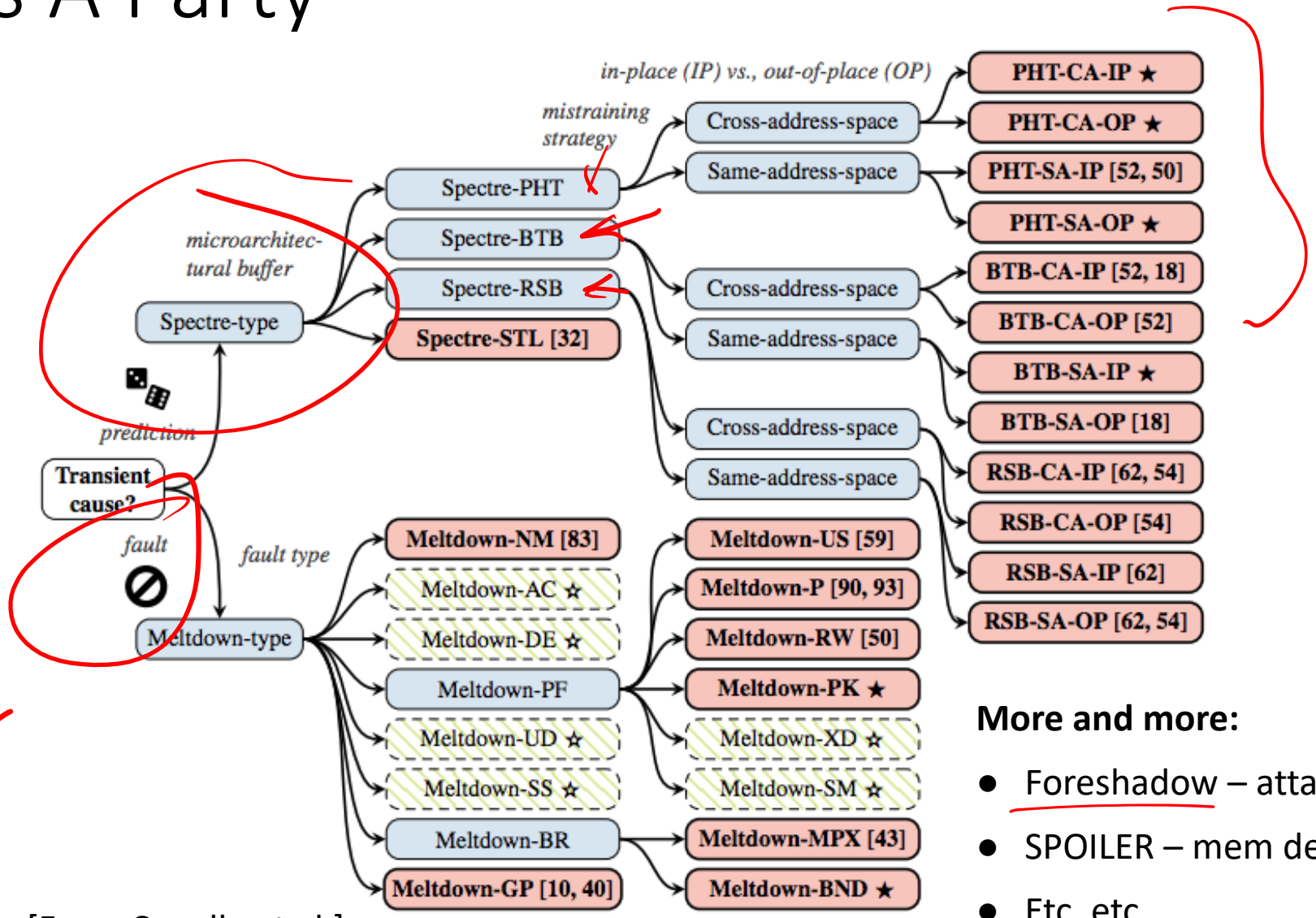
Spectre variant 2

- “Branch target injection”

```
int caller(int(*fptr)()){

    int y = fptr(x);

    return y;
}
```

```
int foo(x){
     array2[array1[x] * 4096];
}

int bar(x){
     return x;
}
```

It's A Party



LITF

More and more:

- Foreshadow – attacks SGX
- SPOILER – mem dependence
- Etc. etc.

[From Canella et al.]

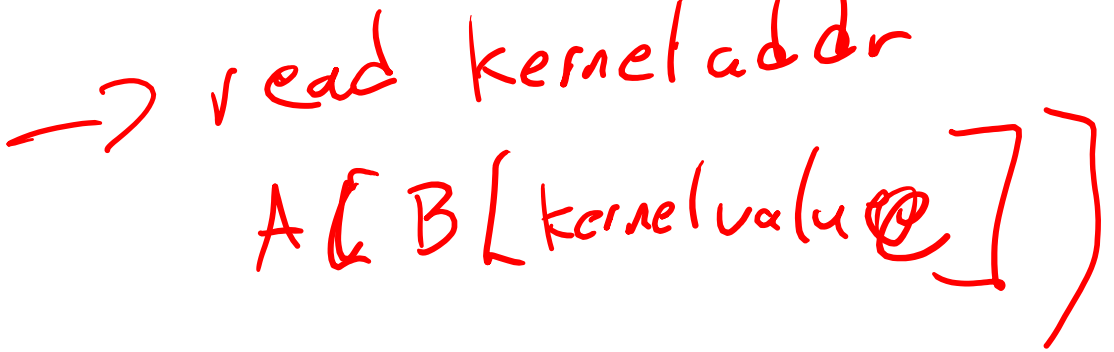
What about ‘Meltdown’?

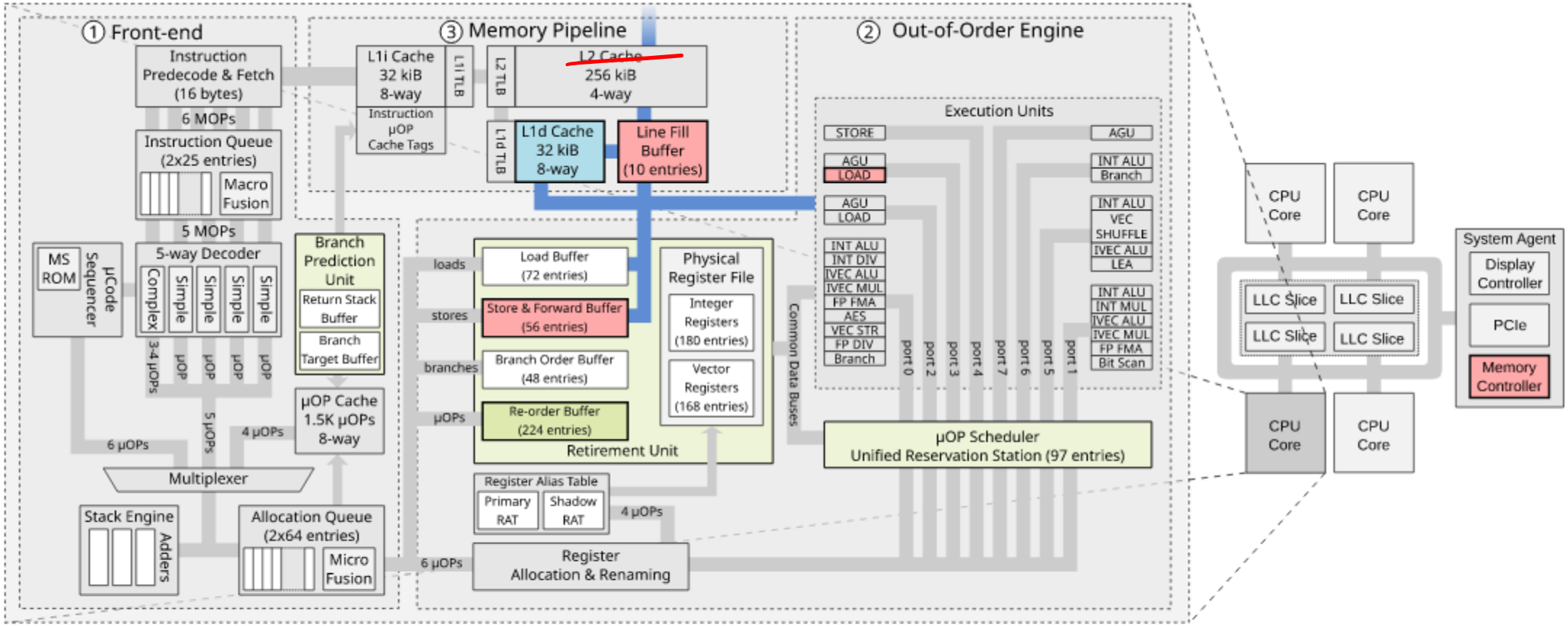
- Also called Spectre variant 3 (“rogue data cache load”)
- Spectre v1/v2 require the victim program to have the vulnerable code pattern
  - Just like the victim program has to have a buffer overflow!
  - Spectre is a global problem with speculation conceptually
- Meltdown allows the attacking program to do whatever it wants!

Meltdown: An Intel specific problem

- Memory permissions weren't checked during speculation
  - At least for some cases

"Imagine the following instruction executed in usermode
`mov rax,[somekernelmodeaddress]`
It will cause an interrupt when retired, [...]"





Click on the various components to interact with them. The full interactive version can be found here and the raw SVG can be found here. There is also a more vibrant colored version (the one used in our paper), which can be found here. These diagrams have been made by Stephan van Schaik (@themadstephan).

https://mdsattacks.com/

Speculative Attacks wrapup

- Spectre vulnerabilities are here to stay, for a long time
- MDS+Meltdown (hopefully) aren't

Pollev

8:45

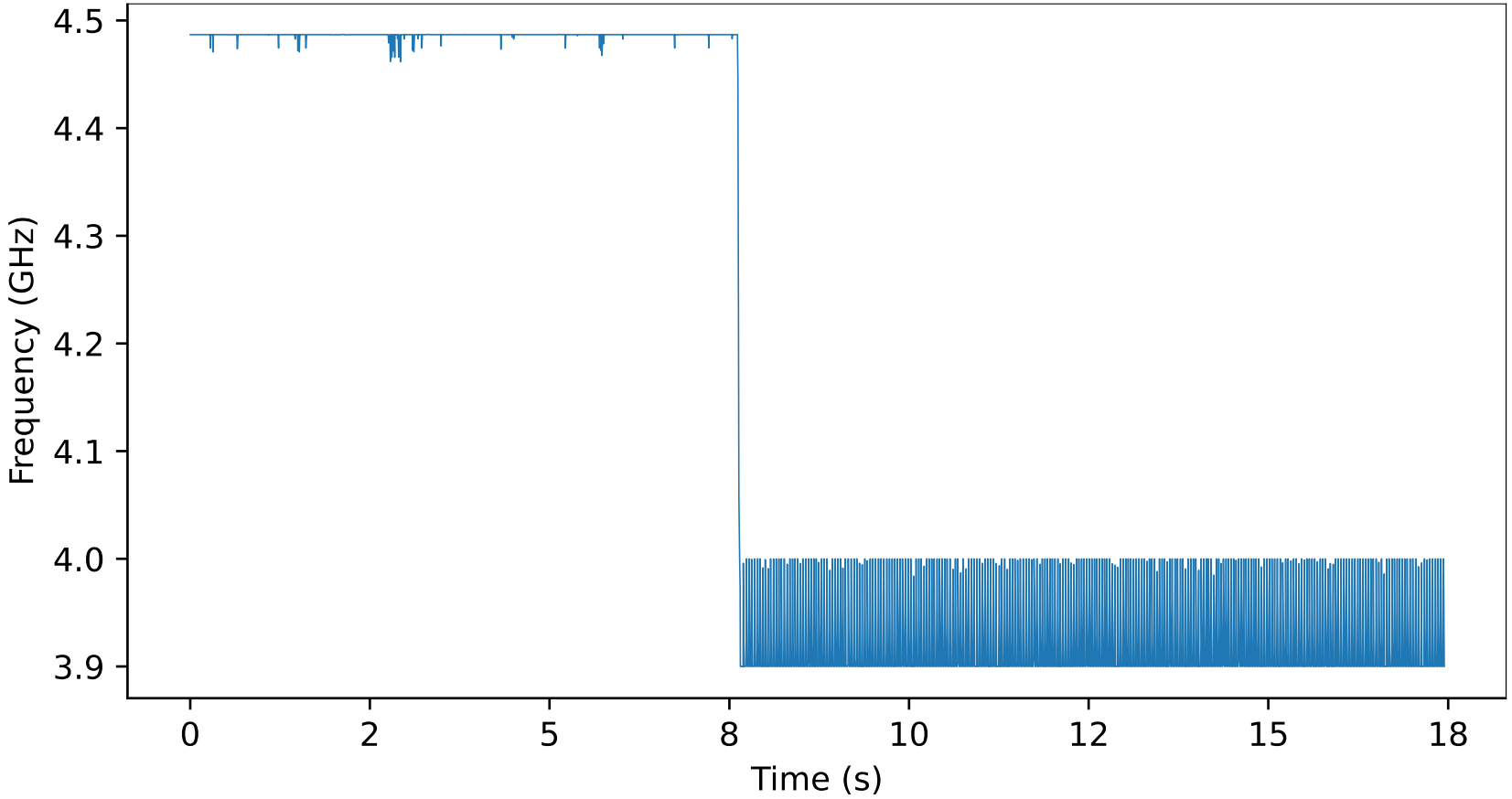
- Browsers had to scramble to deal with Spectre type vulnerabilities as they were exploitable from webpages and allowed for arbitrary memory reads.

- How would you have tried to handle receiving a disclosure like this **as the browser vendors**?
- You can either discuss technical ideas **or** policy objectives for a strategy to handle the vulnerabilities.

Frequency Attacks – “Hertzbleed”

DVFS on a modern Intel CPU

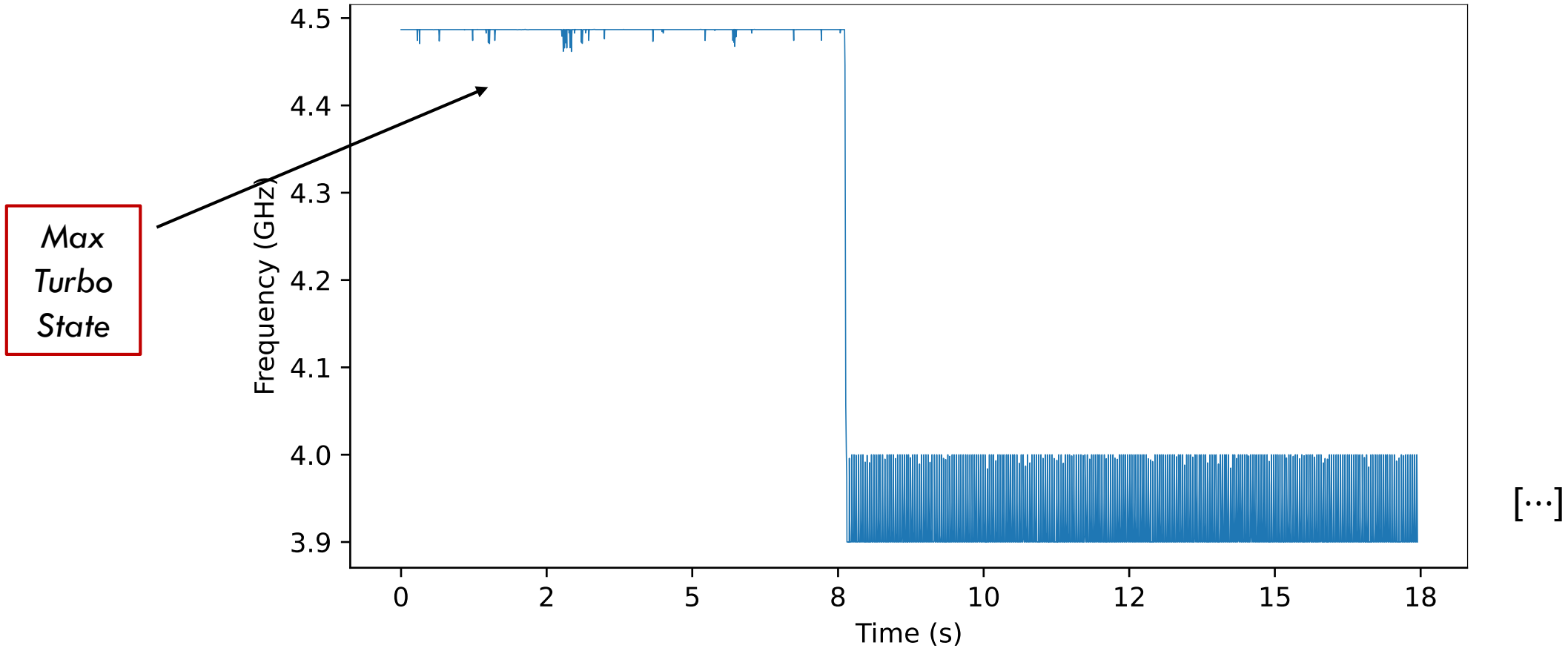
DVFS on a modern Intel CPU



[…]

Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86 – Yingchen Wang & Riccardo Paccagnella

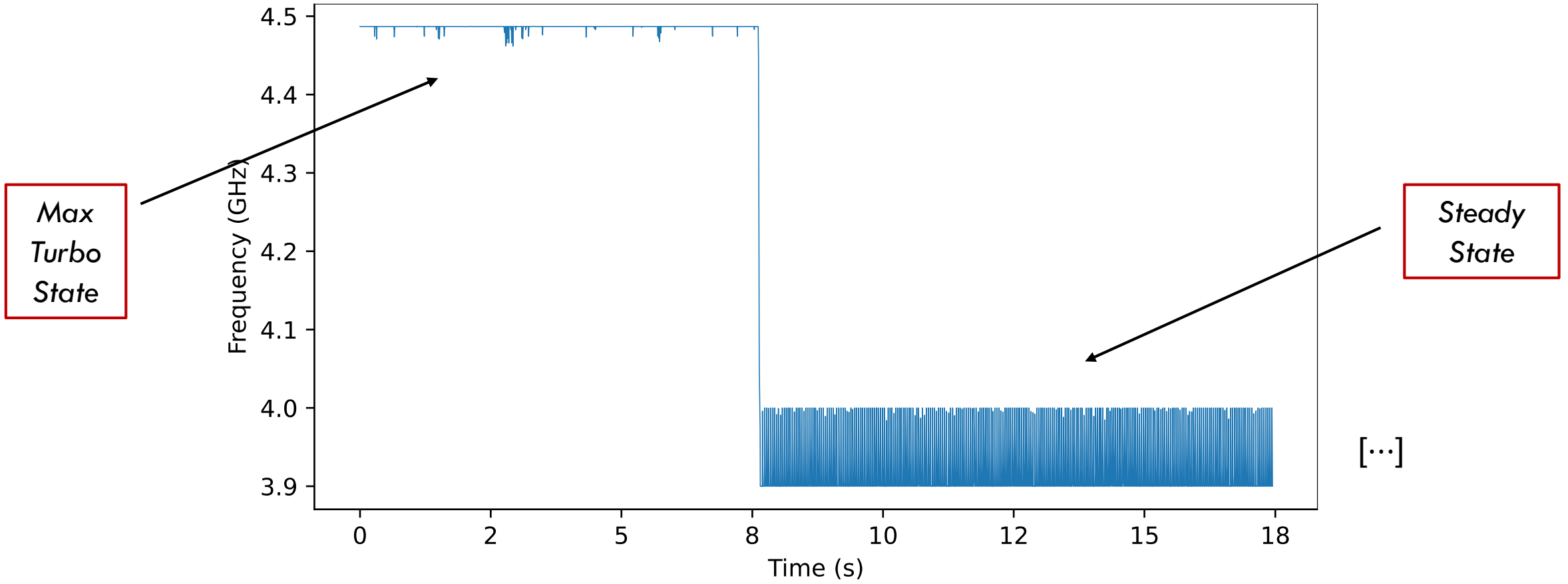
DVFS on a modern Intel CPU



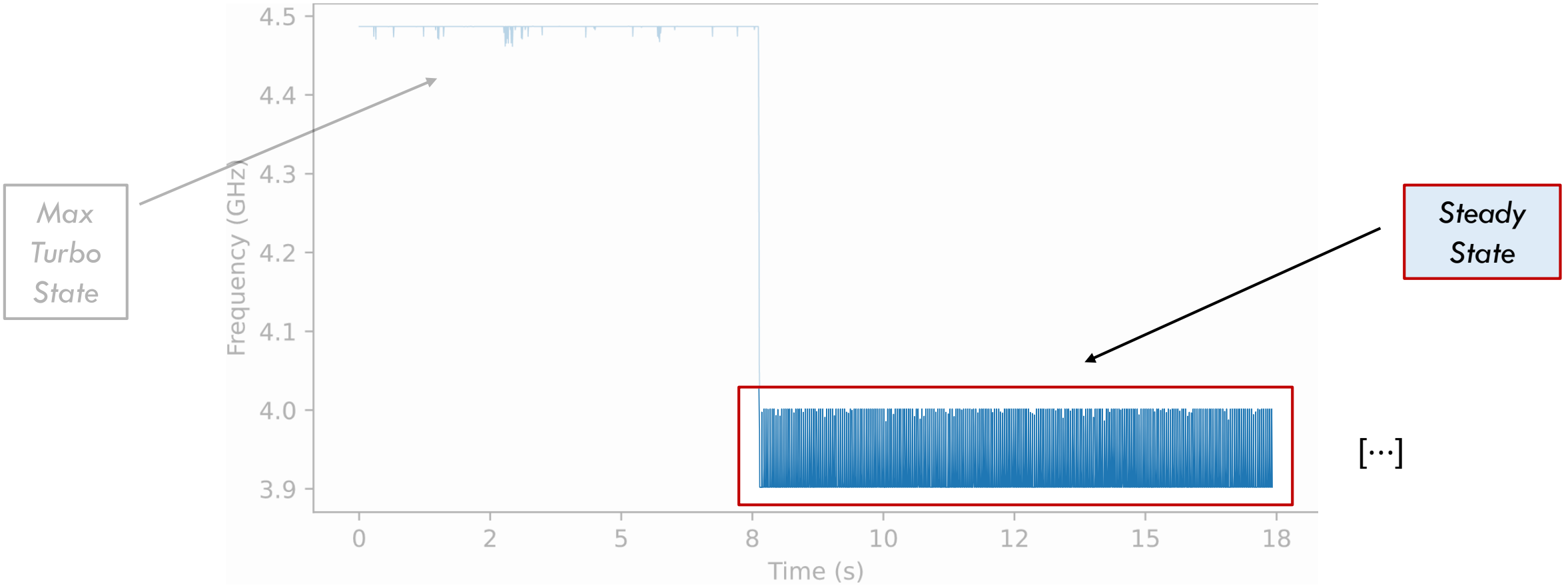
[…]

Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86 – Yingchen Wang & Riccardo Paccagnella

DVFS on a modern Intel CPU



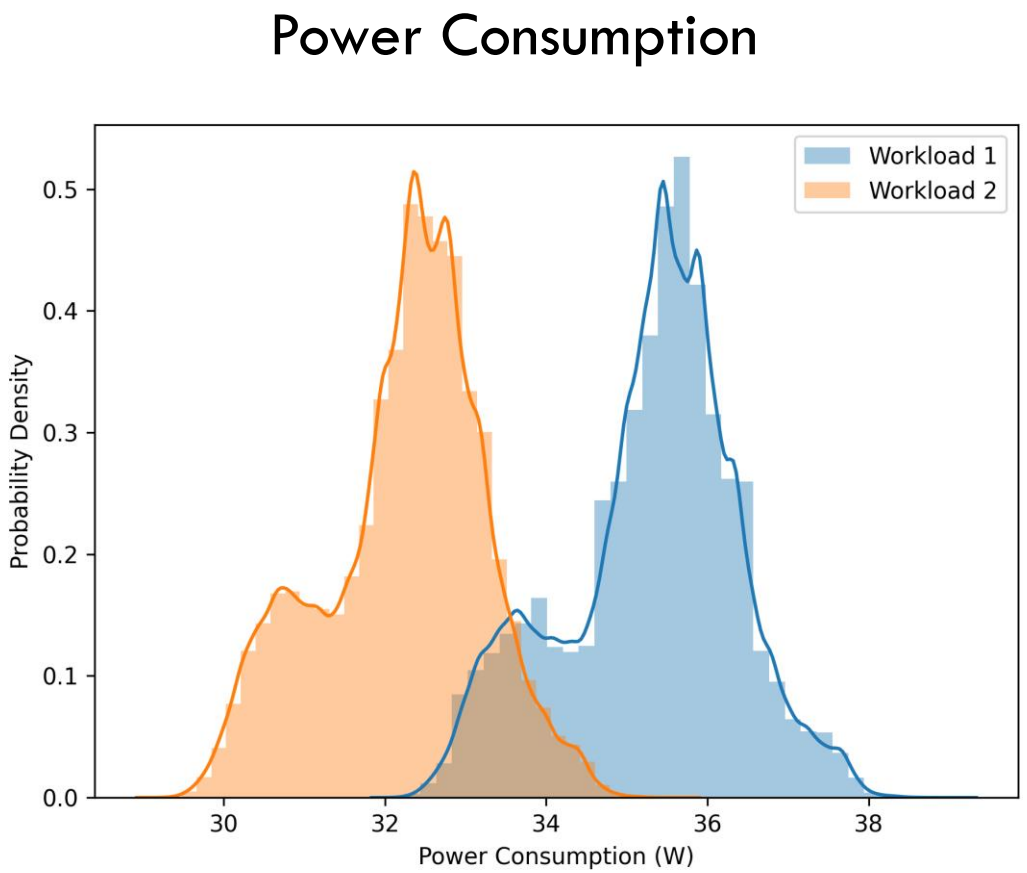
DVFS on a modern Intel CPU



Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86 – Yingchen Wang & Riccardo Paccagnella

Frequency Depends on Power

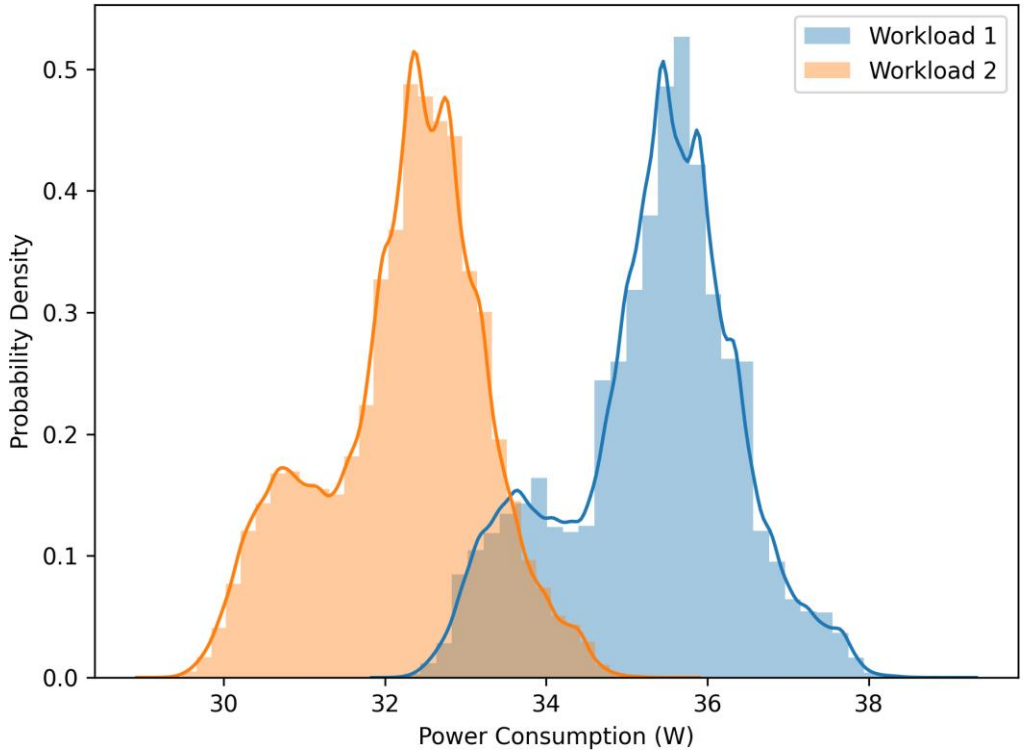
Frequency Depends on Power



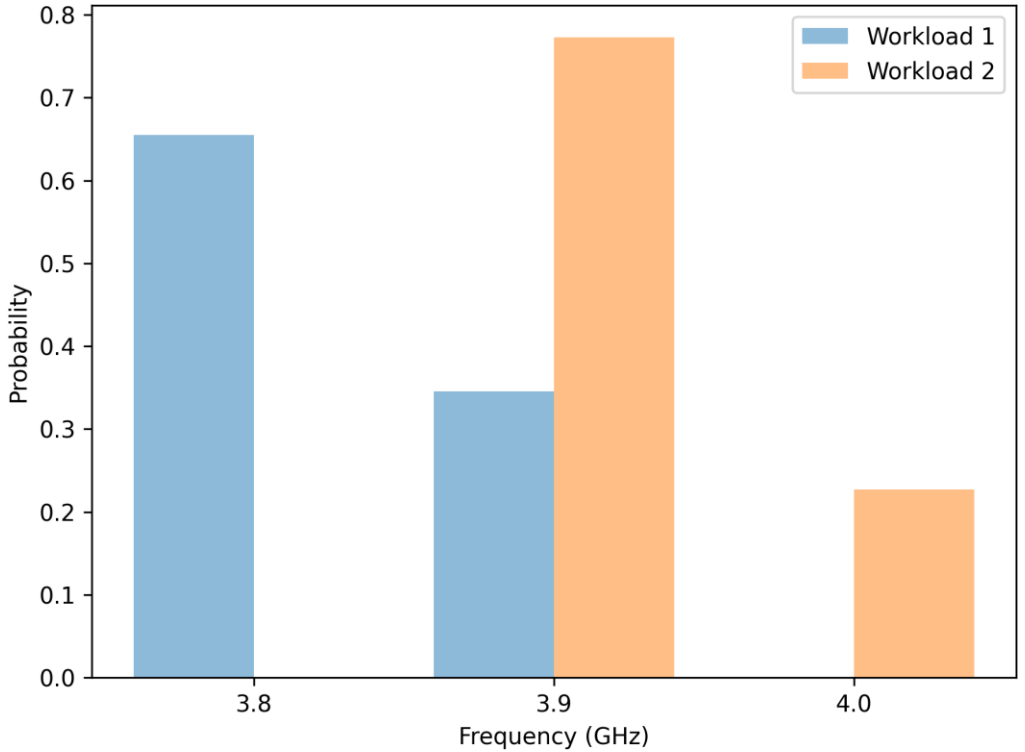
Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86 – Yingchen Wang & Riccardo Paccagnella

Frequency Depends on Power

Power Consumption



CPU Frequency



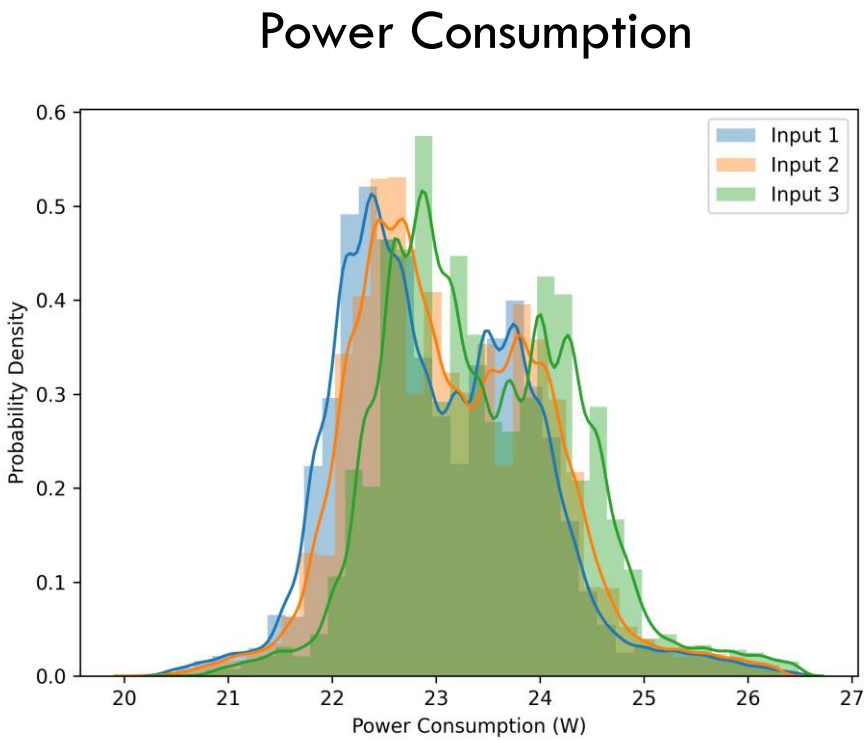
Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86 – Yingchen Wang & Riccardo Paccagnella

Frequency Depends on Data

- Only vary the data values being processed (“Input”).

Frequency Depends on Data

- Only vary the data values being processed (“Input”).

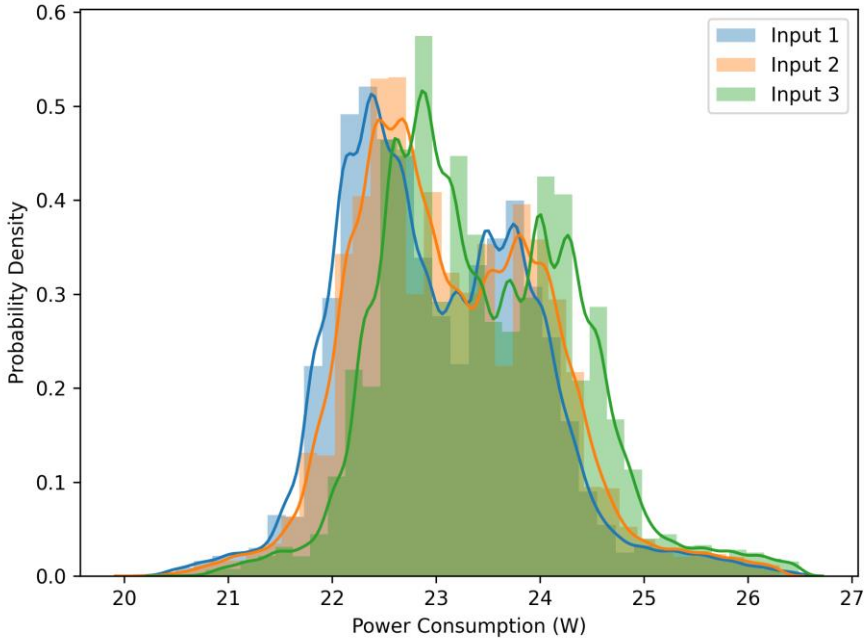


Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86 – Yingchen Wang & Riccardo Paccagnella

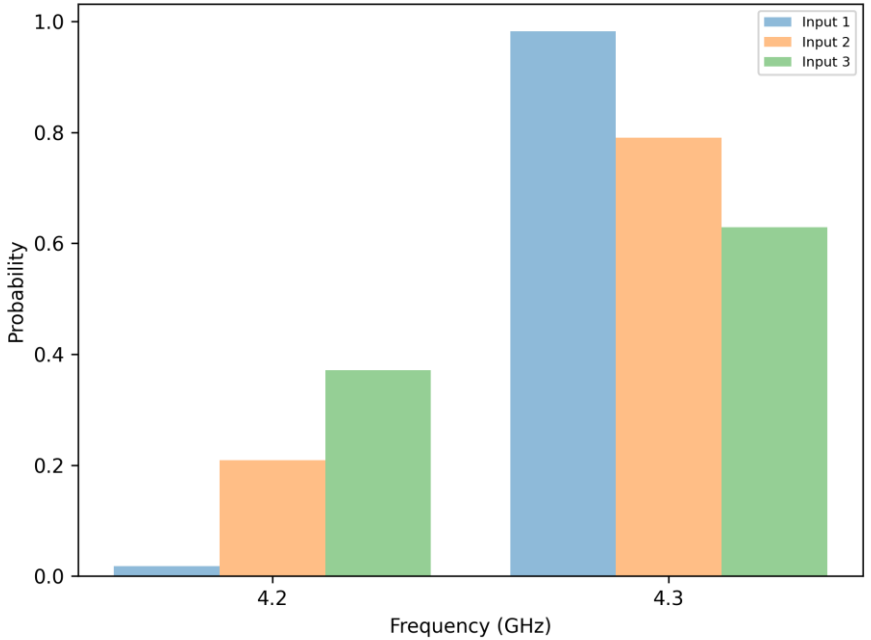
Frequency Depends on Data

- Only vary the data values being processed (“Input”).

Power Consumption



CPU Frequency



Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86 – Yingchen Wang & Riccardo Paccagnella

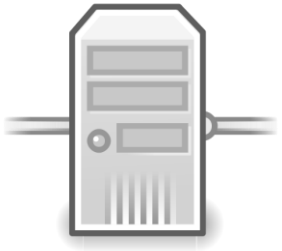
Theory:

- If power depends on secrets
- And heat depends on power
- And processor frequency depends on heat+power
- And execution time depends on frequency

- Thus, execution time depends on secrets
  - *Even if the code takes the exact same number of CPU cycles no matter what!*

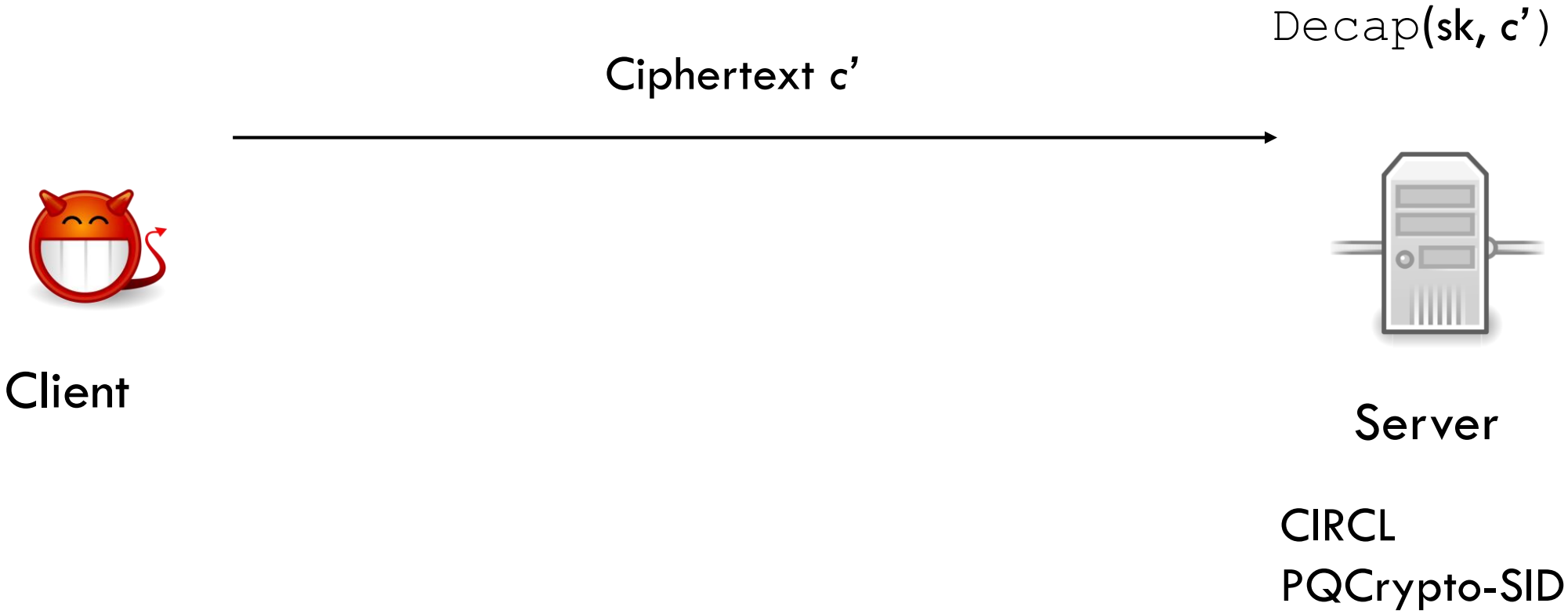
Remote Timing Attack Model

Client

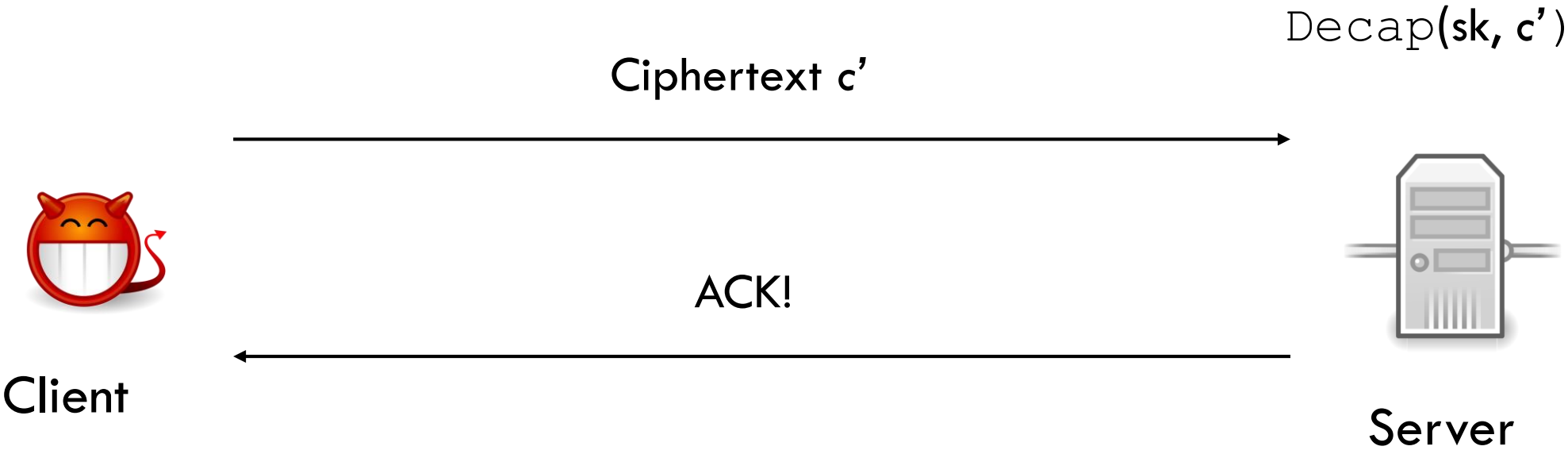


Server

Remote Timing Attack Model



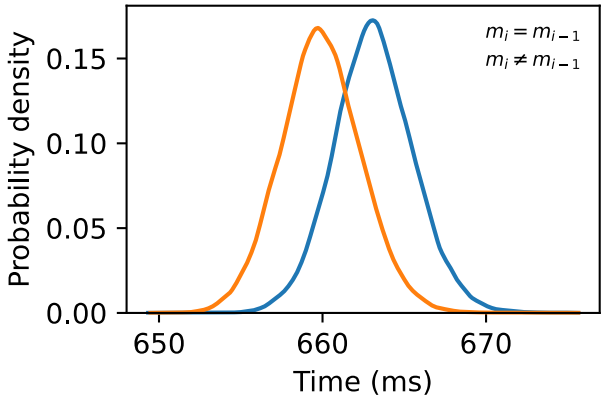
Remote Timing Attack Model

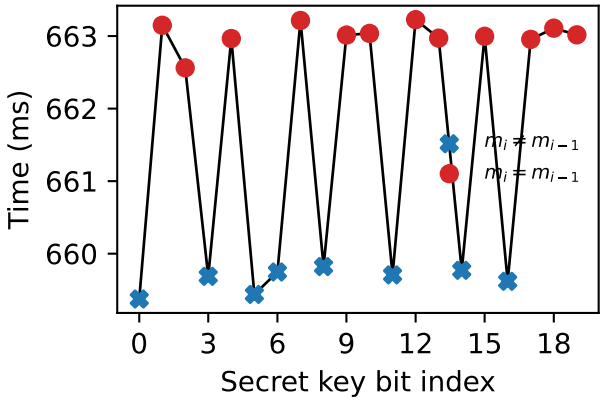


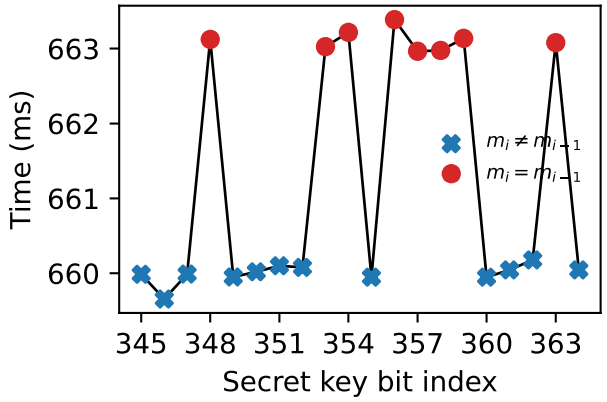
How long it takes to finish *n* concurrent requests

CIRCL
PQCrypto-SIDH

Remote Timing Attack Results



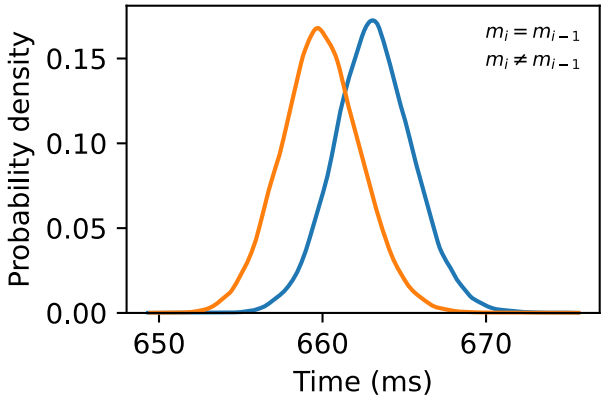


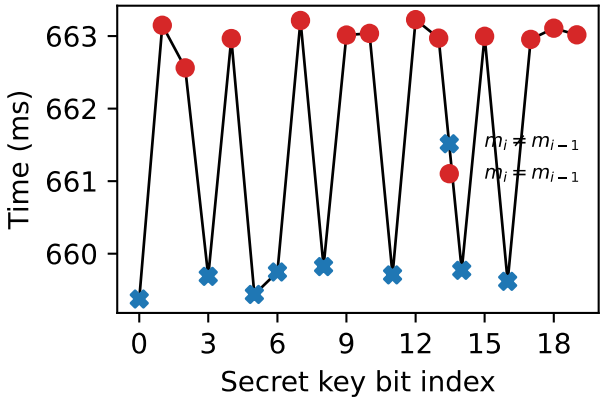


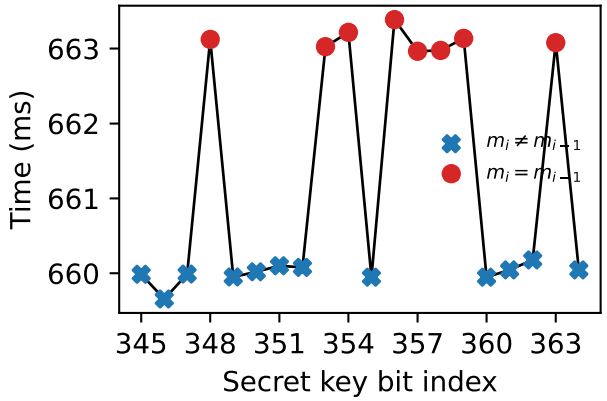
CIRCL:

Recovered full key in 36 hours

Remote Timing Attack Results

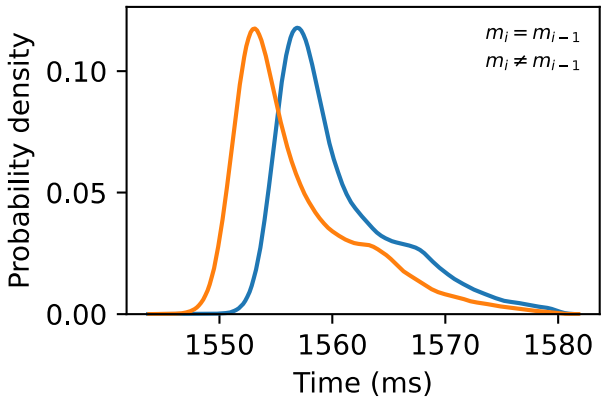


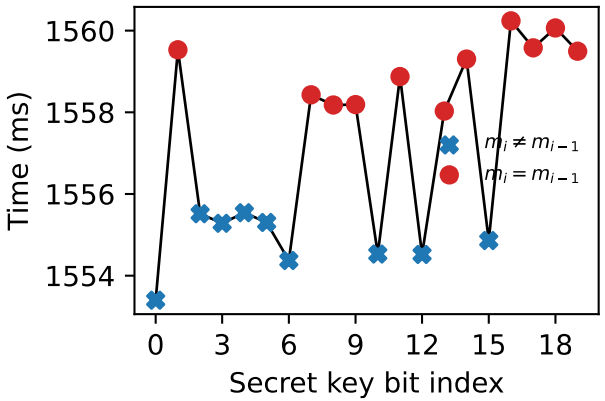


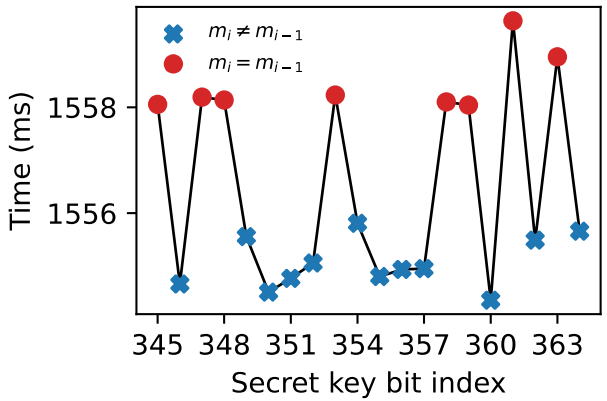


CIRCL:

Recovered full key in 36 hours







PQCrypto-SIDH:

Recovered full key in 89 hours

Mitigating Side-Channels

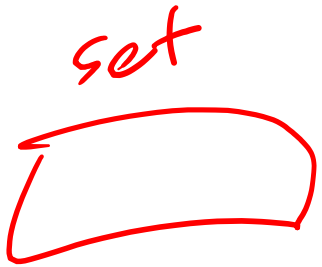
Several approaches

- Remove the source of leakage in code
- Fix the problem in hardware
- Mask the leakage
- Isolation

Spectre Defenses

- Disable User/Kernel memory space sharing
  - KAISER/KPTI defense
- “Fence” dangerous code patterns
  - Extra instruction that block speculation past some point
- Microcode updates for processors
  - MDS-class fixes

Cache Side-Channel Defenses



- Isolation:
  - Partition the cache into discrete parts, don't share them
- Randomization:
  - Cache placement is randomized
- Code changes:
  - Write code that never makes a secret-dependent memory access

a[0] a[64]

Generalized Timing Attacks

- Rewrite code to be timing-independent
  - "Always do" every operation, don't branch
- Delay results until the maximum response time
- Randomize timing information
- Granularize timing information

Constant ~~time~~ cycle

```
// Constant-time max and min functions for unsigned arguments
static inline unsigned
umax(unsigned a, unsigned b)
{
  return a - ((a - b) & -(a < b));
}

static inline unsigned
umin(unsigned a, unsigned b)
{
  return a - ((a - b) & -(a > b));
}
```

Wrapping up side-channels