CSE 484 : Computer Security and Privacy

(Software) Side Channel Attacks

Winter 2021

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 ...

Admin

- **Lab3 due** Next Friday
- **Homework 3 due** Monday
- **Last extra credit** reading due next Thursday
  - No late days
- **Final project due** next 03/16
  - No late days
  - Make sure you:
    - Include references
    - Include at least one legal/ethics discussion slide
    - Create original content
    - Go beyond class materials (if it's a topic we also covered)

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

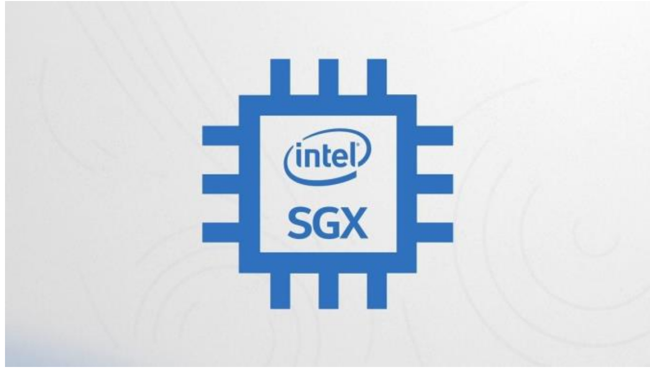
- Most commonly discussed in the context of cryptosystems
- But also prevalent in many contexts
  - E.g., we discussed the TENEX password implementation
  - E.g., we discussed browser fingerprinting

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



Timing Side-Channels

- Duration of a program (or operation) reveals information
- TENEX case
  - We… lied, sorry
  - Its… more complicated

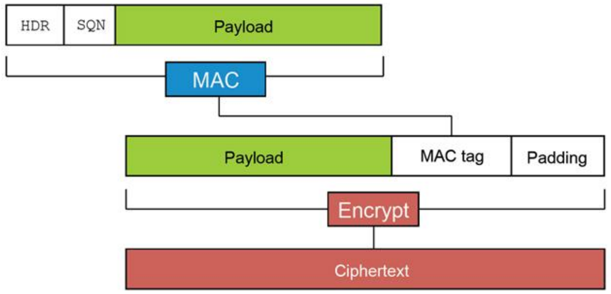
TENEX attack (for real)

- 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
- Why?
  - Pollev.com/dkohlbre



Attacking cryptographic 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

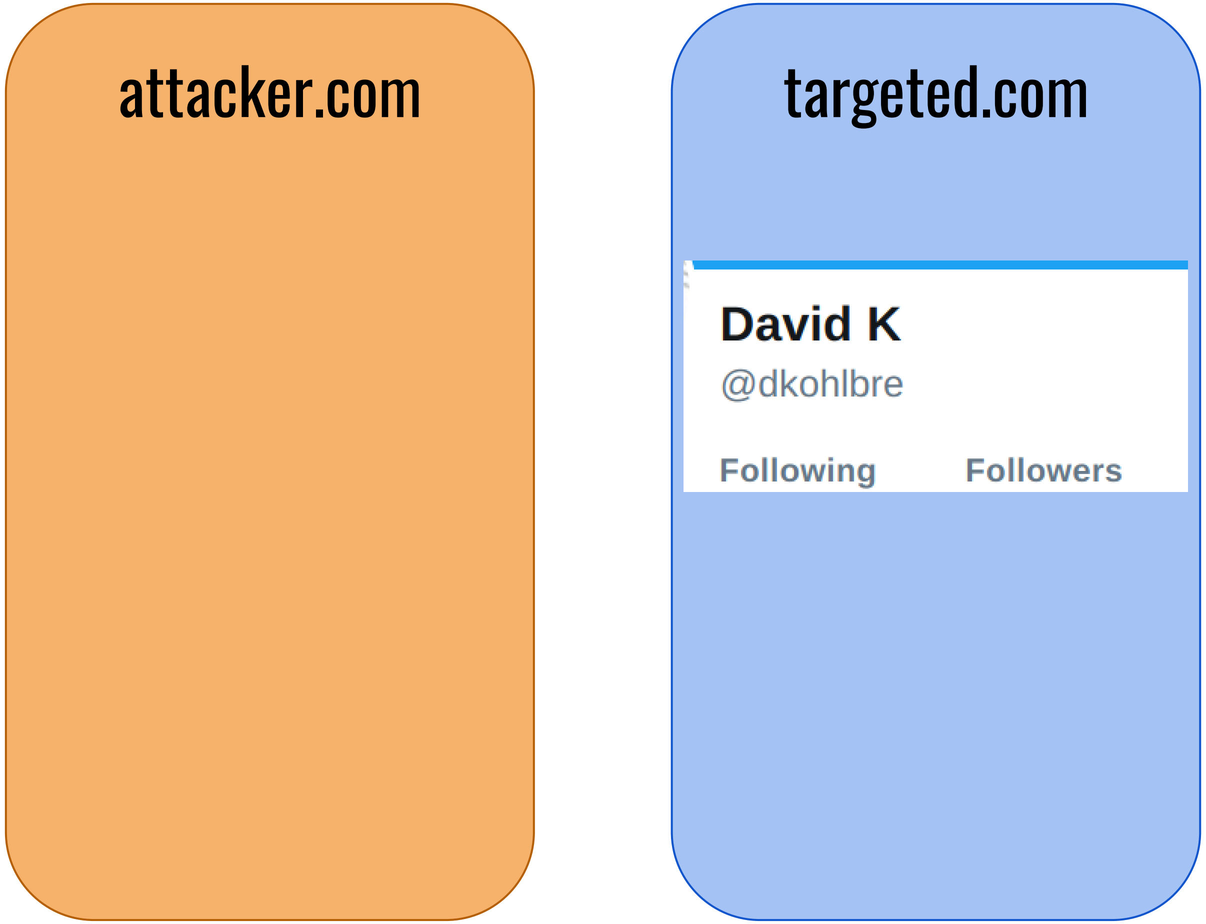
- **RSA:** Leverage key-dependent timings of modular exponentiations
  - https://www.rambus.com/timing-attacks-on-implementations-of-diffie-hellman-rsa-dss-and-other-systems/
  - http://crypto.stanford.edu/~dabo/papers/ssl-timing.pdf

- **Block Ciphers:** Leverage key-dependent cache hits/misses

How odd can this get?

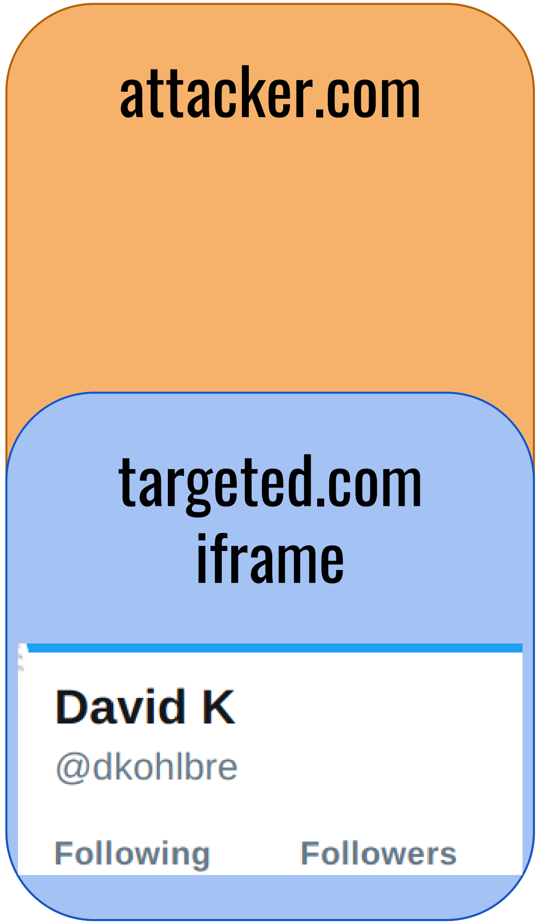
The SVG-filter pixel-stealing timing attack

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



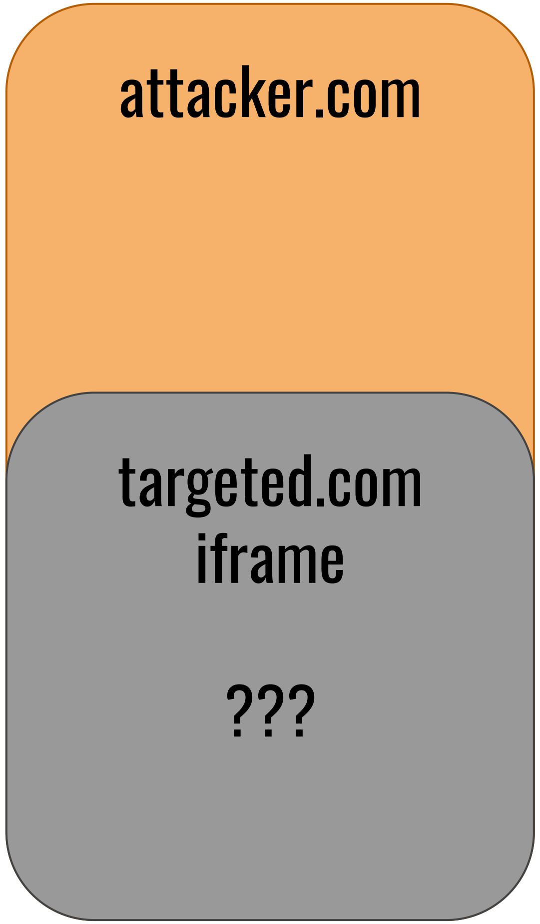
The SVG-filter pixel-stealing timing attack

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



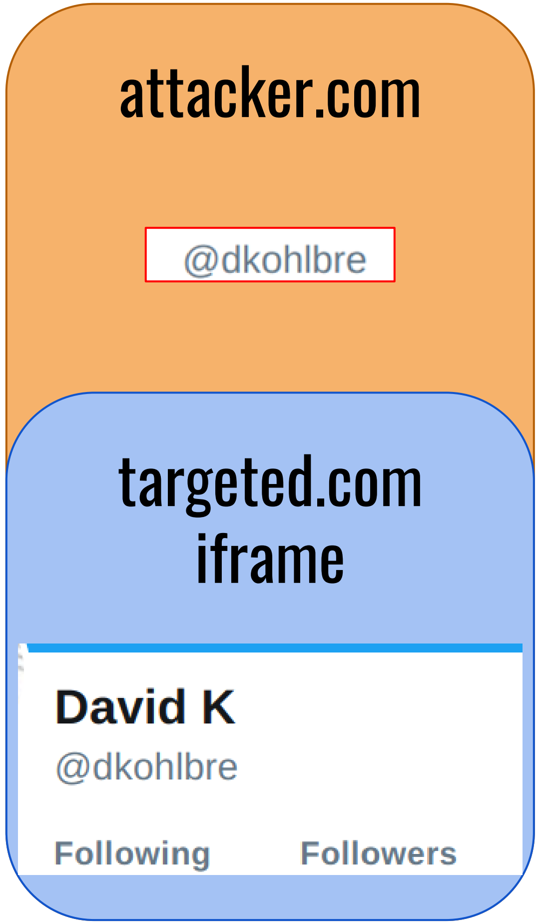
The SVG-filter pixel-stealing timing attack

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

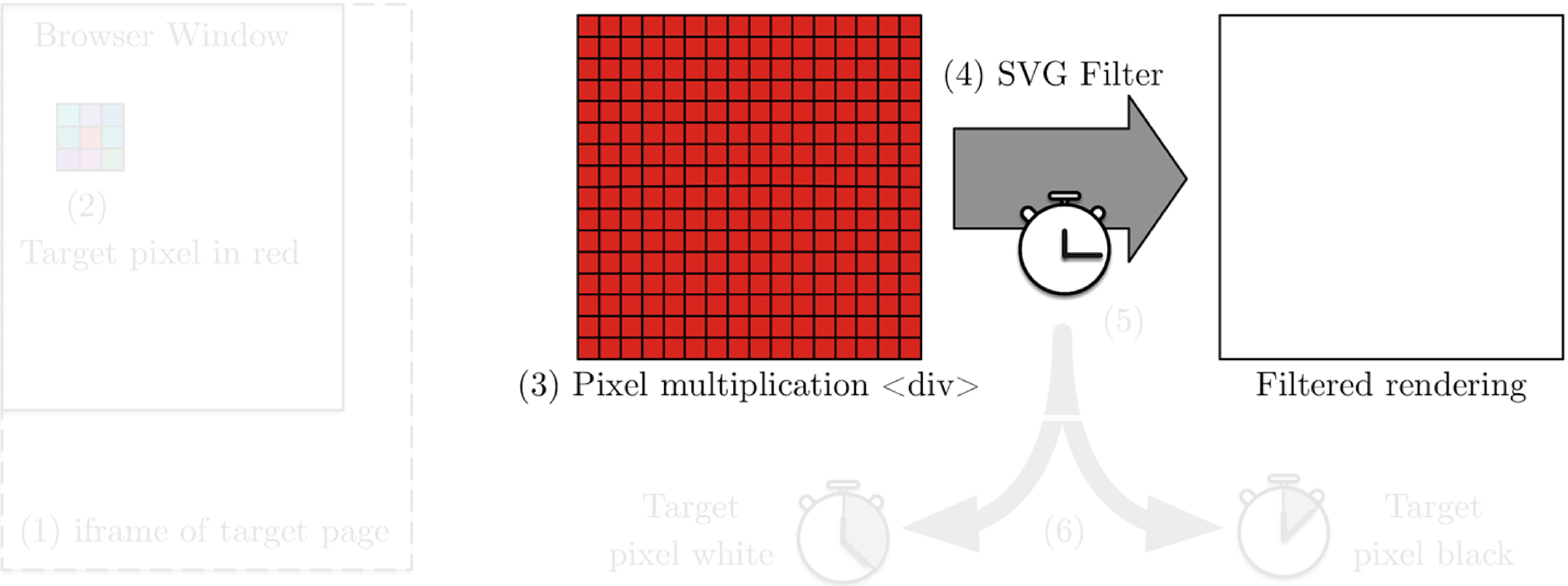


The SVG-filter pixel-stealing timing attack

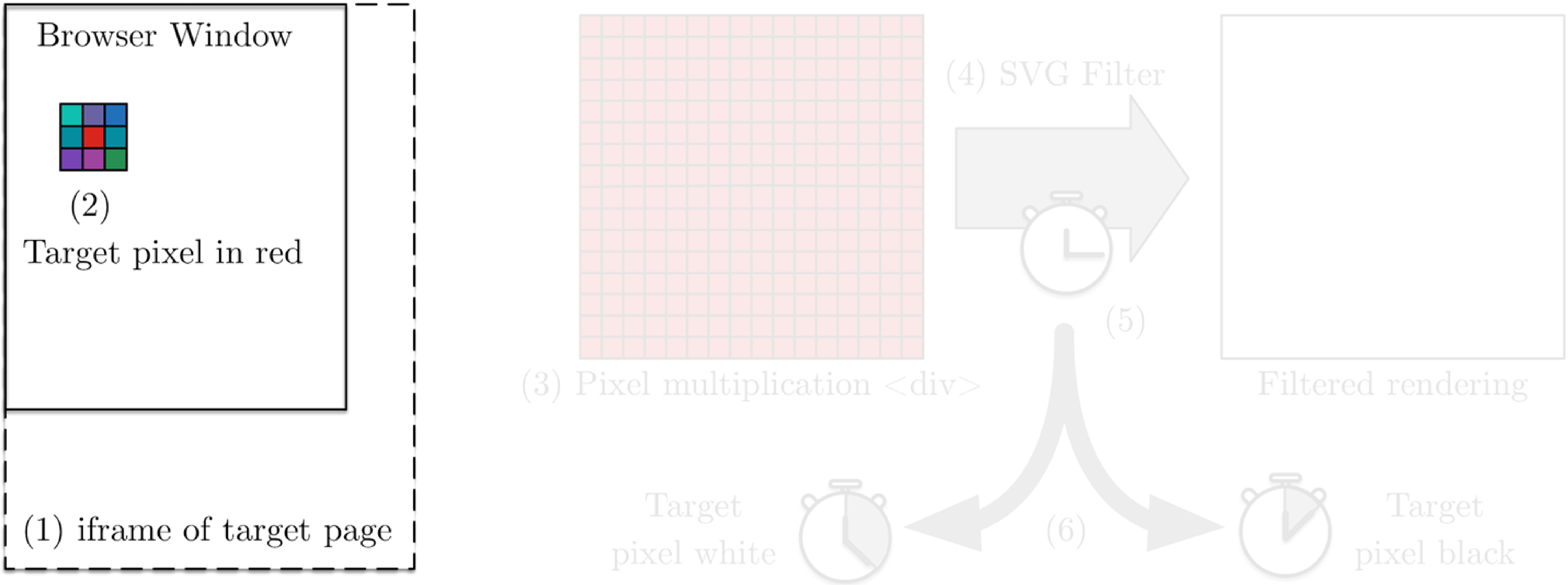
- **Attacker**:
  - Hosts webpage
- **Victim**:
  - Visits attacker
  - Logged into target
- **Target**:
  - Website hosting private visual information



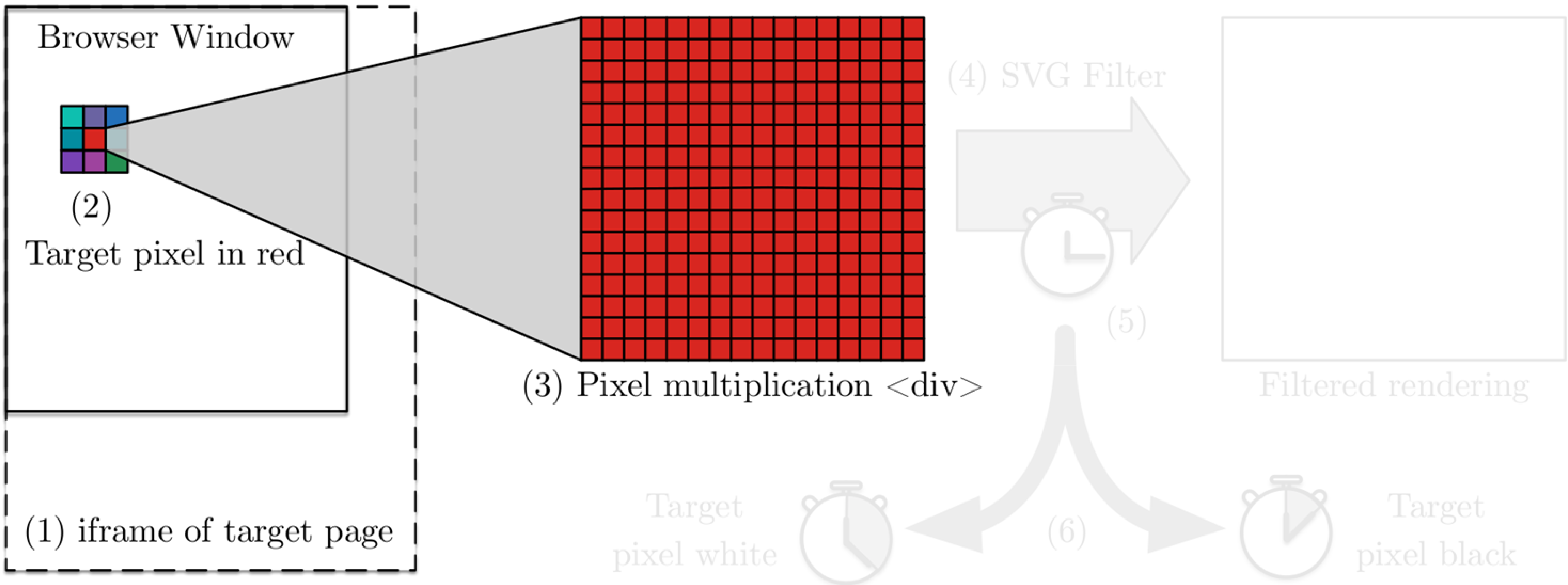
SVG-filter Pixel-stealing attack overview



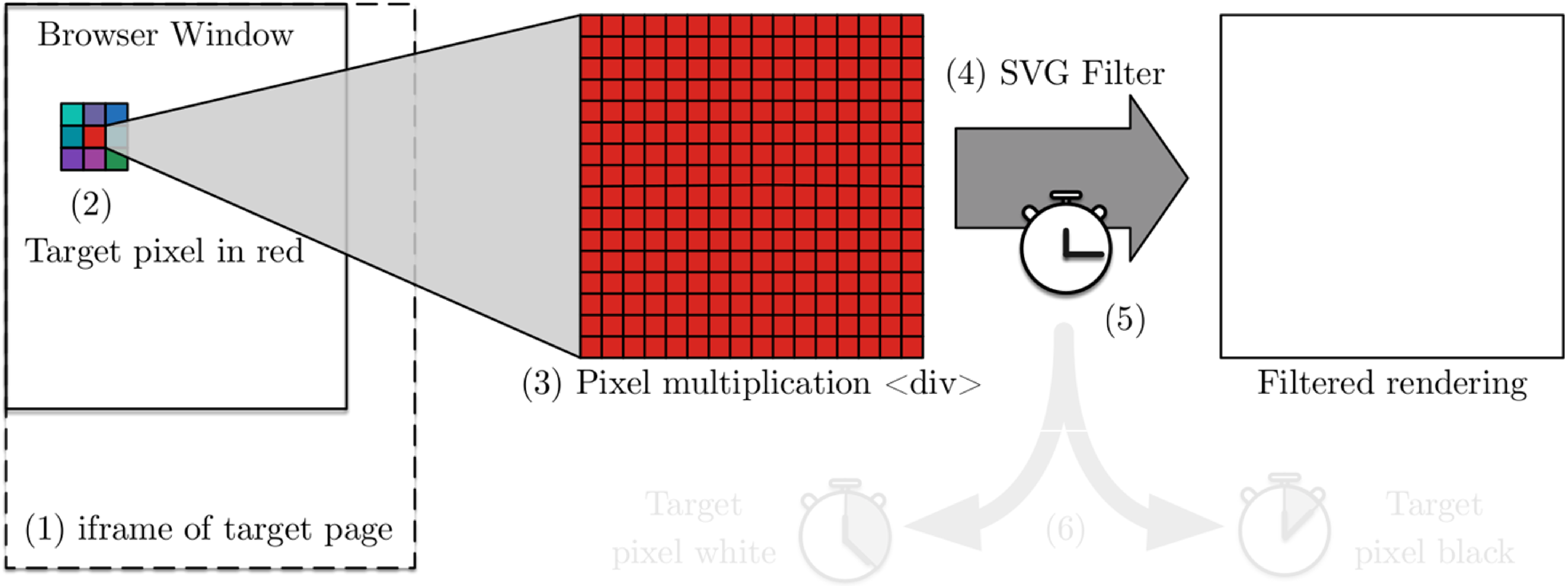
SVG-filter Pixel-stealing attack overview



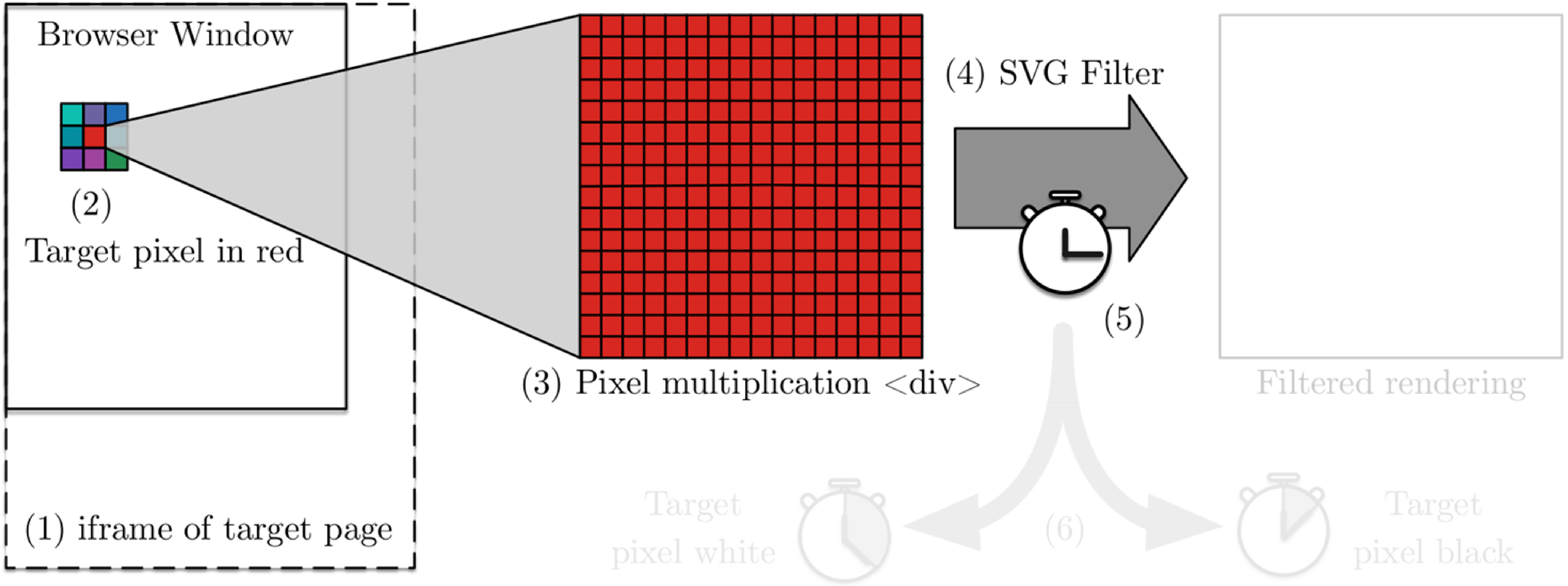
SVG-filter Pixel-stealing attack overview



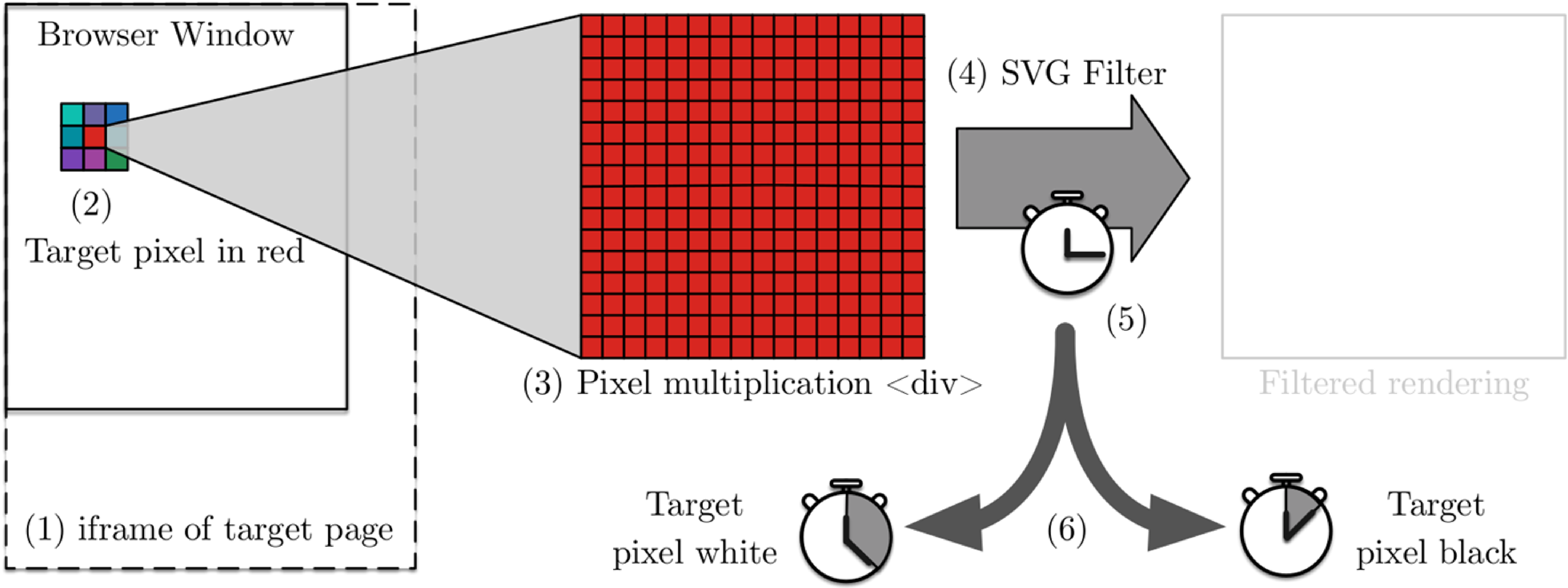
SVG-filter Pixel-stealing attack overview



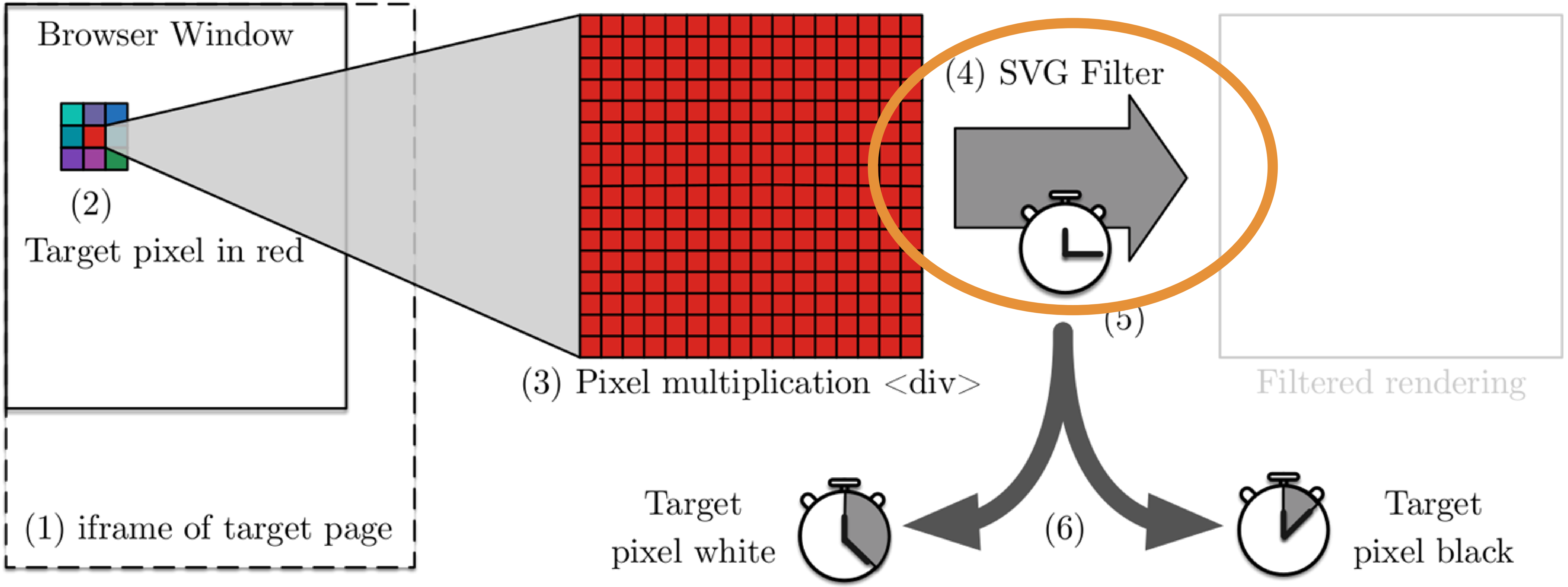
SVG-filter Pixel-stealing attack overview



SVG-filter Pixel-stealing attack overview



SVG-filter Pixel-stealing attack overview



How?

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

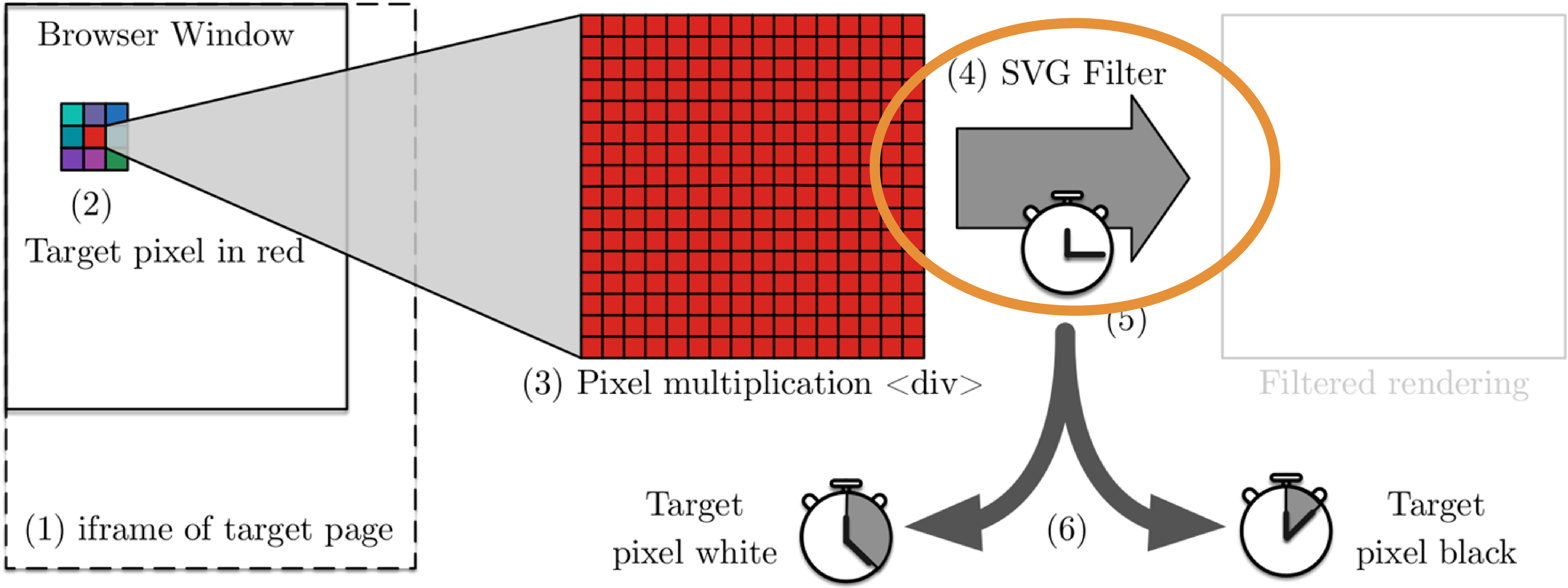
```
if (x == rect.x || xExt[0]  <= startX || xExt[1] <= startX ||
    xExt[2] <= startX || xExt[3] <= startX) {
  [...]
} else { // We only need to look at the newest column
  for (PRUint32 y1 = startY; y1 <= endY; y1++) {
    [...]
```

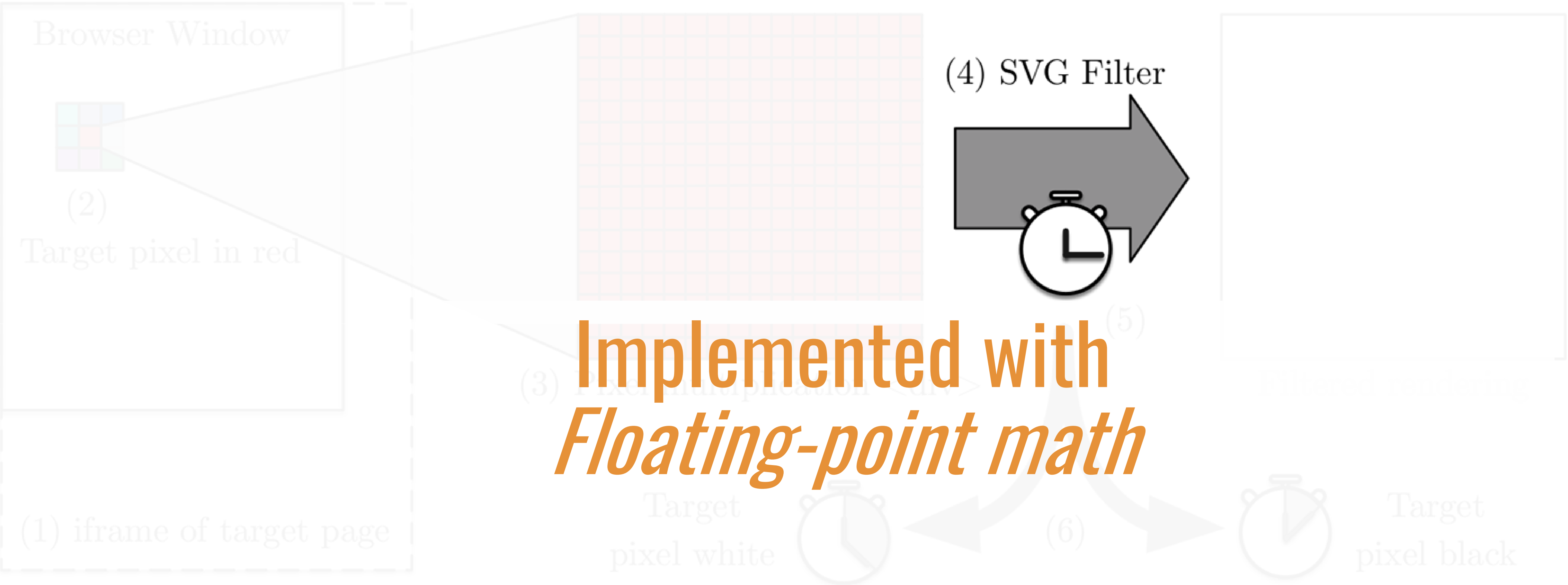
Stone, 2013

```
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 <div>
(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

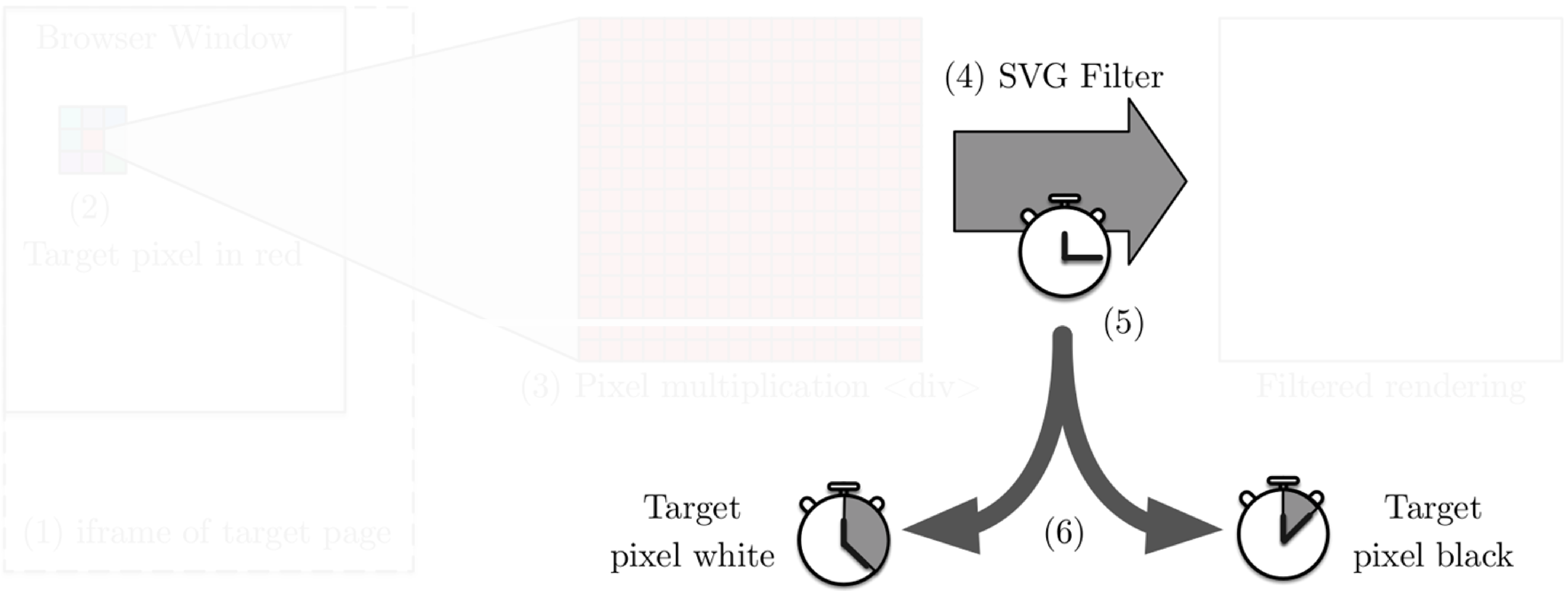
"secret"

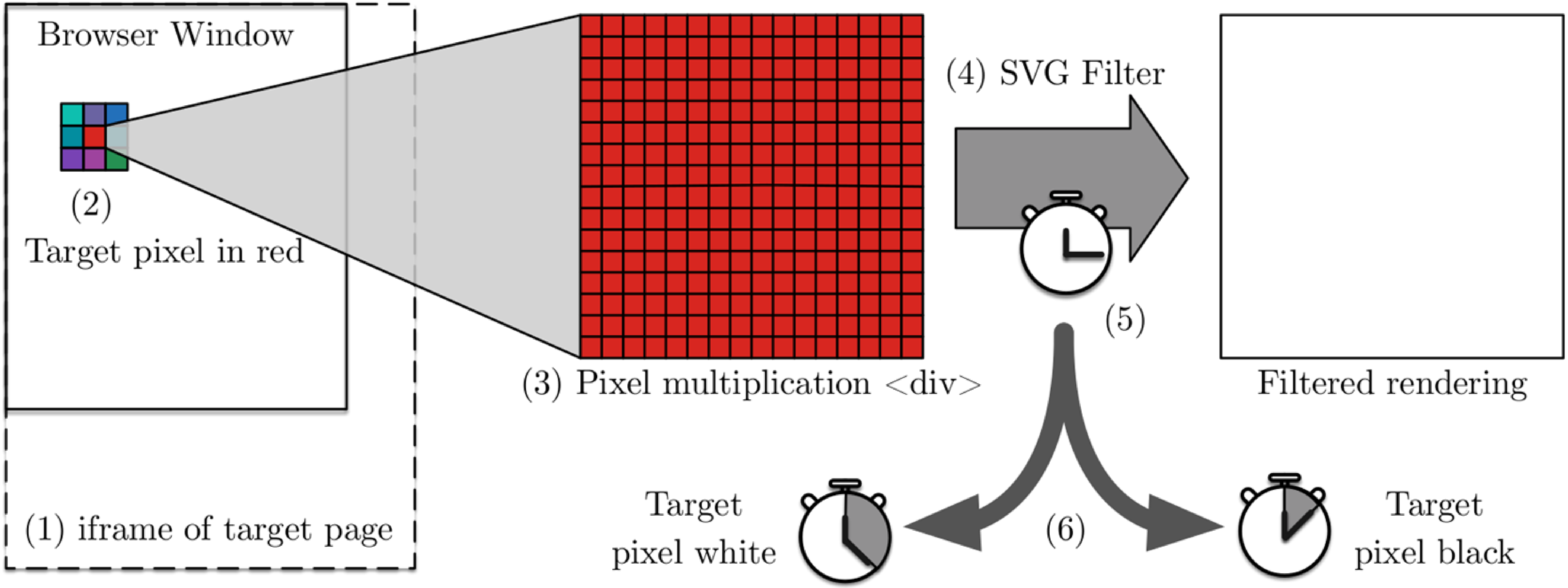
	0.0	1.0	1e10	1e+200	1e-300	1e-42	256	257	1e-320
	Cycle count								
0.0	6.59	6.56	6.59	6.58	6.58	6.57	6.58	6.59	6.57
1.0	6.57	6.59	6.55	[illegible]	[illegible]	6.56	6.56	6.56	130.89
1e10	6.55	6.55	6.56	[illegible]	[illegible]	6.56	6.56	6.57	130.95
1e+200	6.55	6.57	6.56	[illegible]	[illegible]	6.53	6.55	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

secret x 1e-320

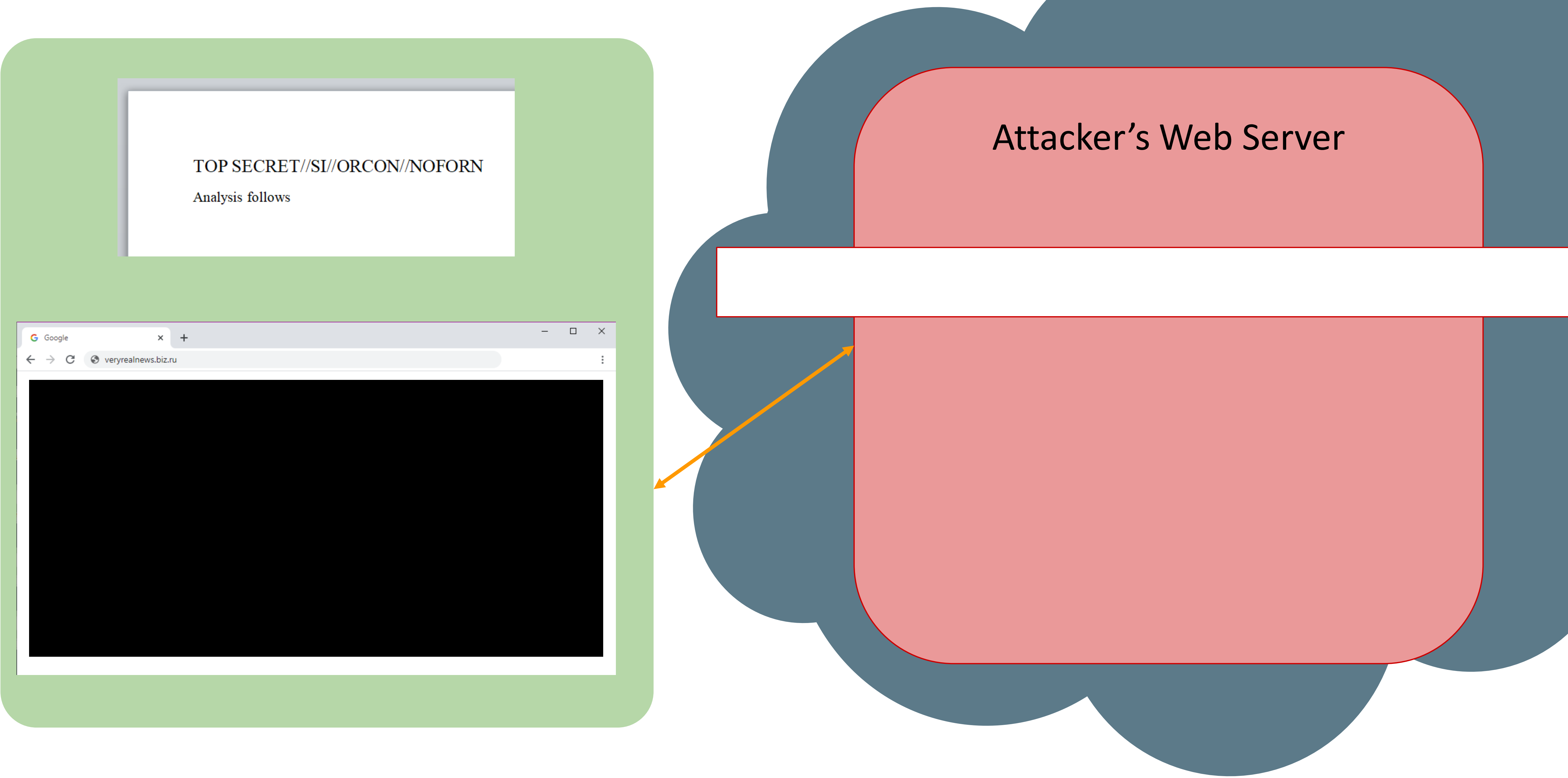
Intel i5-4460 double-precision floating-point multiply

	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


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


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

Attack in Action



Pixel stealing takeaways

- Combines web security, hardware knowledge, and software design
- Side-channels are real, and viable ☺

Power-side channels

- The amount of *power* used by a computer is related to what it is doing
- How can you use this?
- Canvas

Cache side-channels

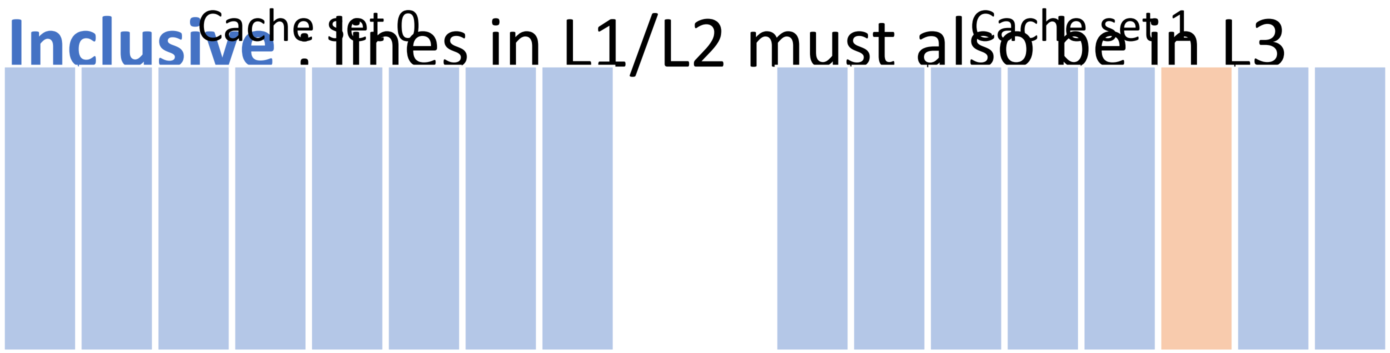
- **Idea**: The cache's current state implies something about prior memory accesses

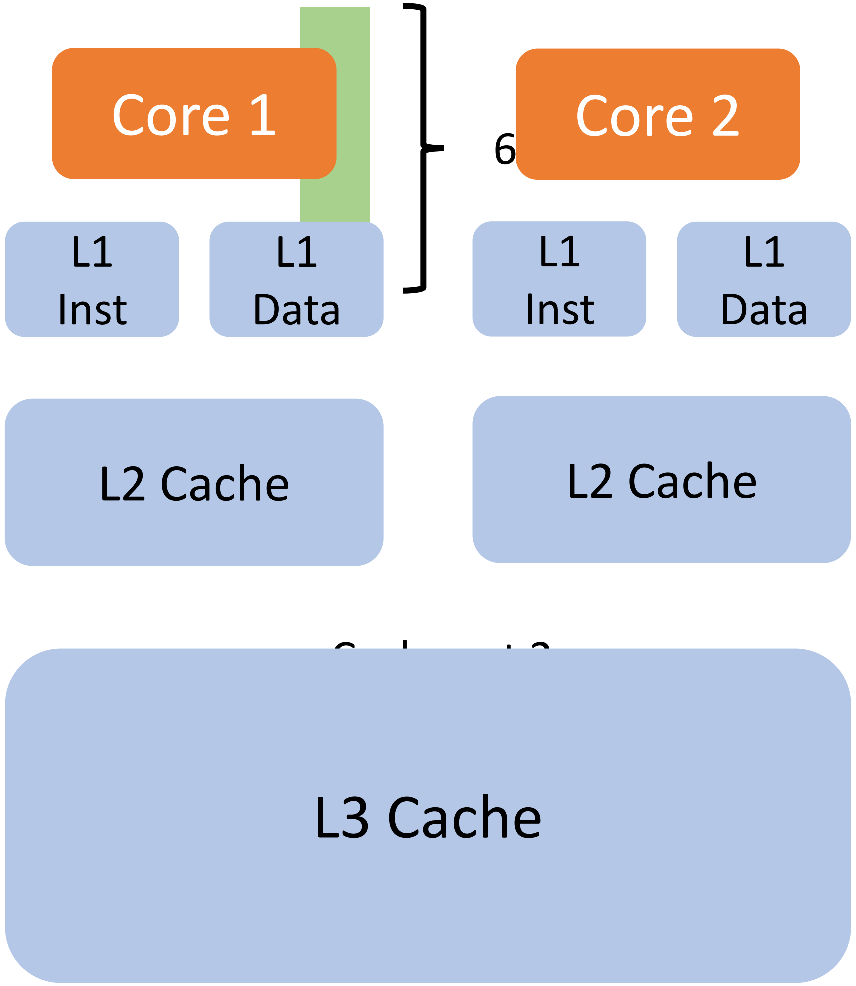
- **Insight**: Prior memory accesses can tell you a lot about a program!

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

Pre-Attack

Active Attack

Analysis

Many thanks to Craig Disselkoen for the animations.

Pre-Attack

Timing threshold

Eviction set

Active Attack

Prime targeted set

Wait

[Timed] Prime targeted set

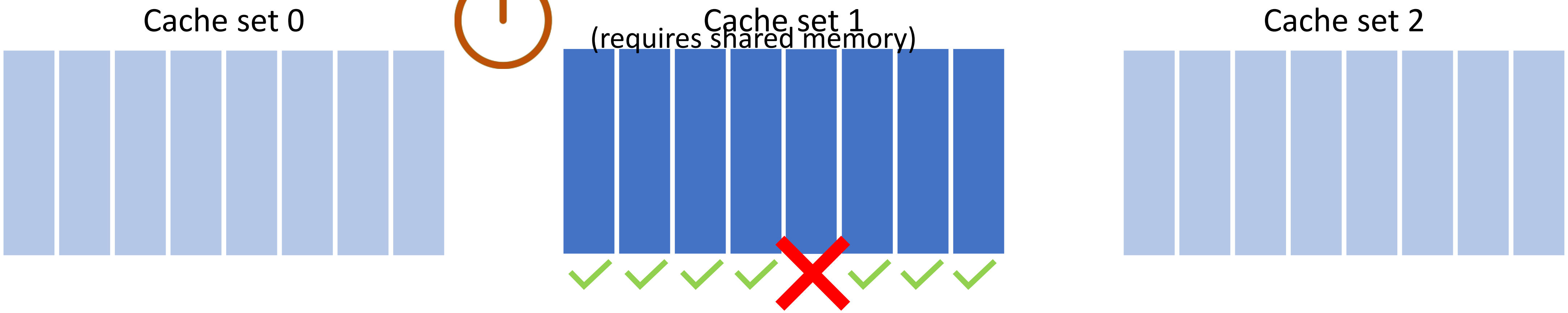
Victim accesses targeted set

Analysis

Victim access if time > threshold

PRIME+PROBE → FLUSH+RELOAD

(requires shared memory)



Pre-existing data

Attacker's data

Victim's data

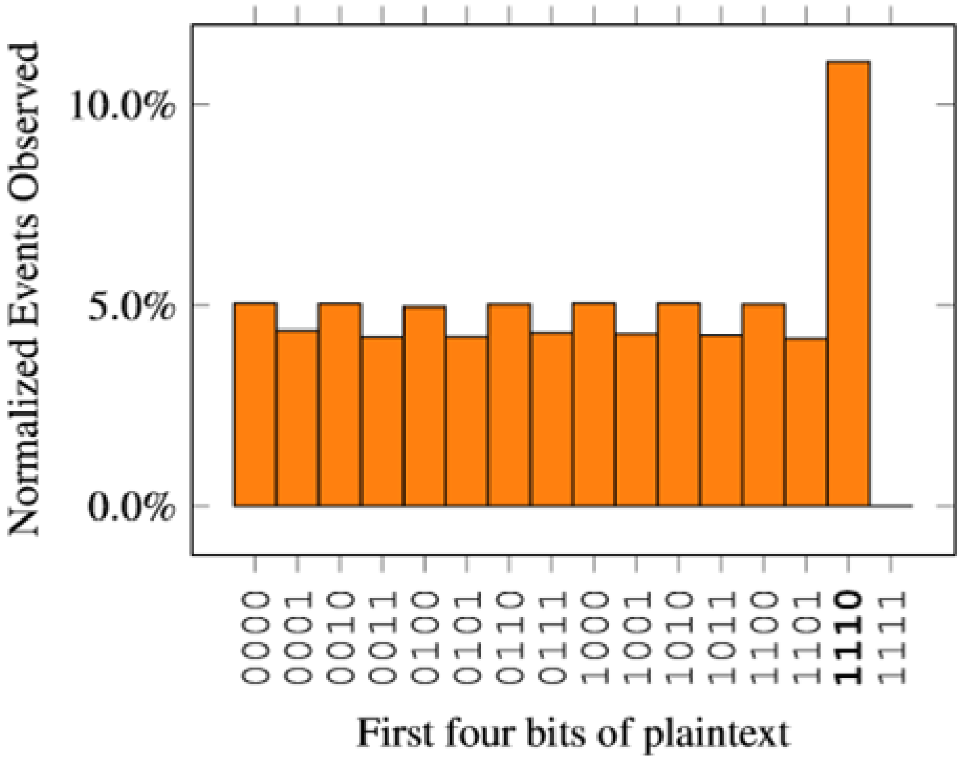
T-Table AES

- Tested against OpenSSL's T-Table implementation of AES
- Traditional target for cache attacks
- No longer used in practice, but useful for comparison
- Chosen plaintext, key recovery

Disselkoen et al. 2017

T-Table AES

PRIME+PROBE



PRIME+ABORT