On the effectiveness of mitigations against floating-point timing channels

David Kohlbrenner
Hovav Shacham
UC San Diego

How effective are ~~On the effectiveness of~~ mitigations against floating-point timing channels?

David Kohlbrenner
Hovav Shacham
UC San Diego

Safari	CVE-2017-7006
Firefox	CVE-2017-5407
Chrome	CVE-2017-5107

Safari	CVE-2017-7006
Firefox	CVE-2017-5407
Chrome	CVE-2017-5107

Pixel-stealing attacks on browsers
Using floating-point side-channels

Outline

- Pixel-stealing attacks
- Floating-point benchmarking
- Attacking with floats
- Beating defenses
- Conclusions

Background on pixel-stealing

- Pixel-stealing attacks
- Floating-point benchmarking
- Attacking with floats
- Beating defenses
- Conclusions

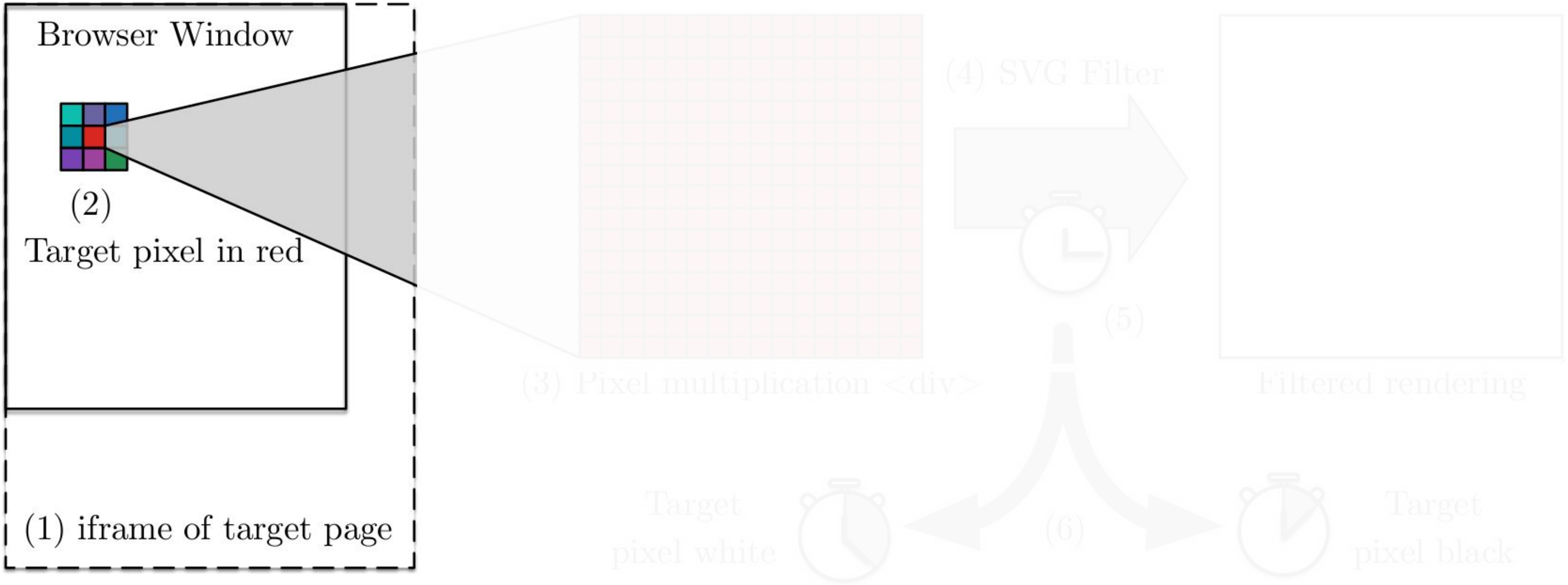
Pixel-stealing attacks - Terminology

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

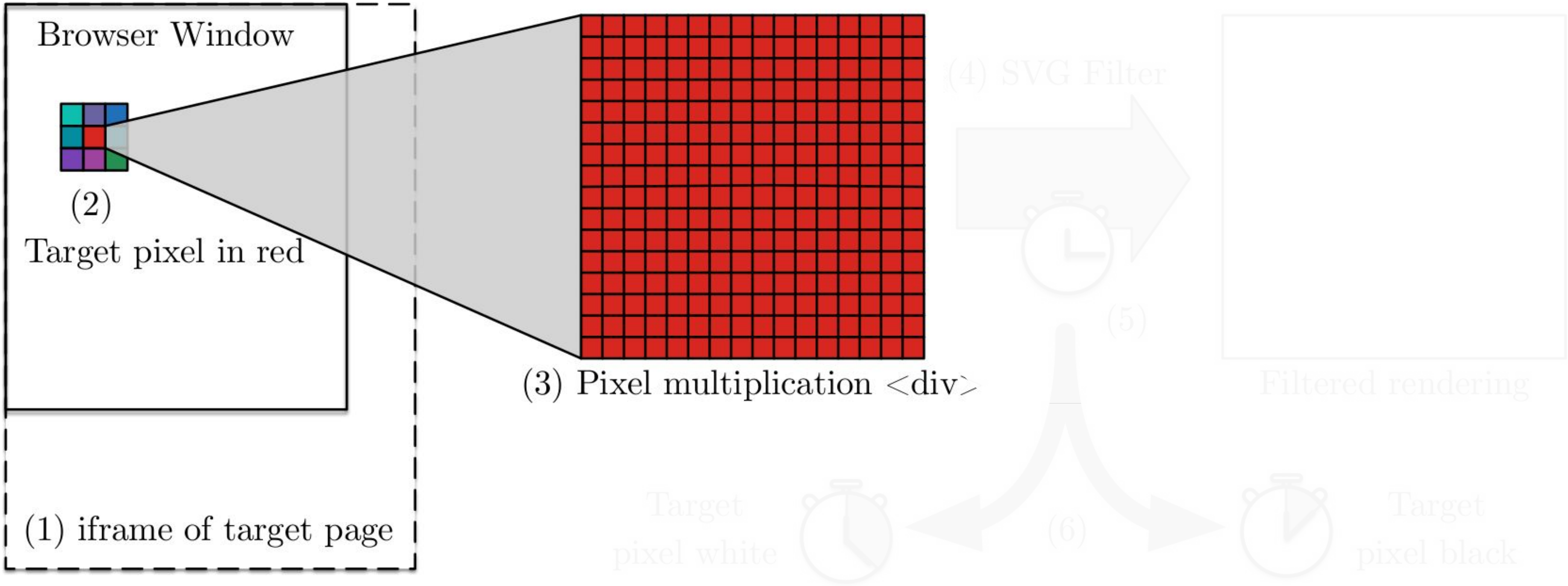
- Impact*:
  - Attacking page learns pixel information from target
  - Ex:
    - Bank information
    - Login status
    - Usernames

See Paul Stone's "Pixel Perfect Timing Attacks with HTML5"

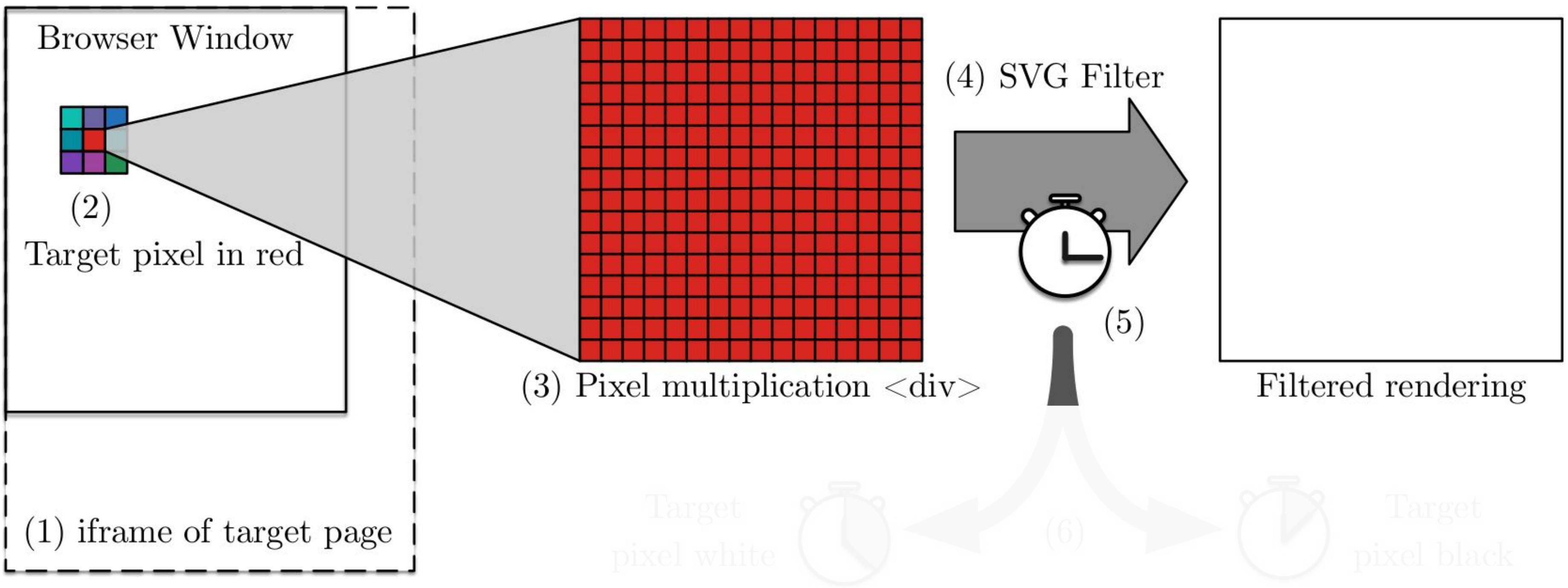
* Ask me about history sniffing!



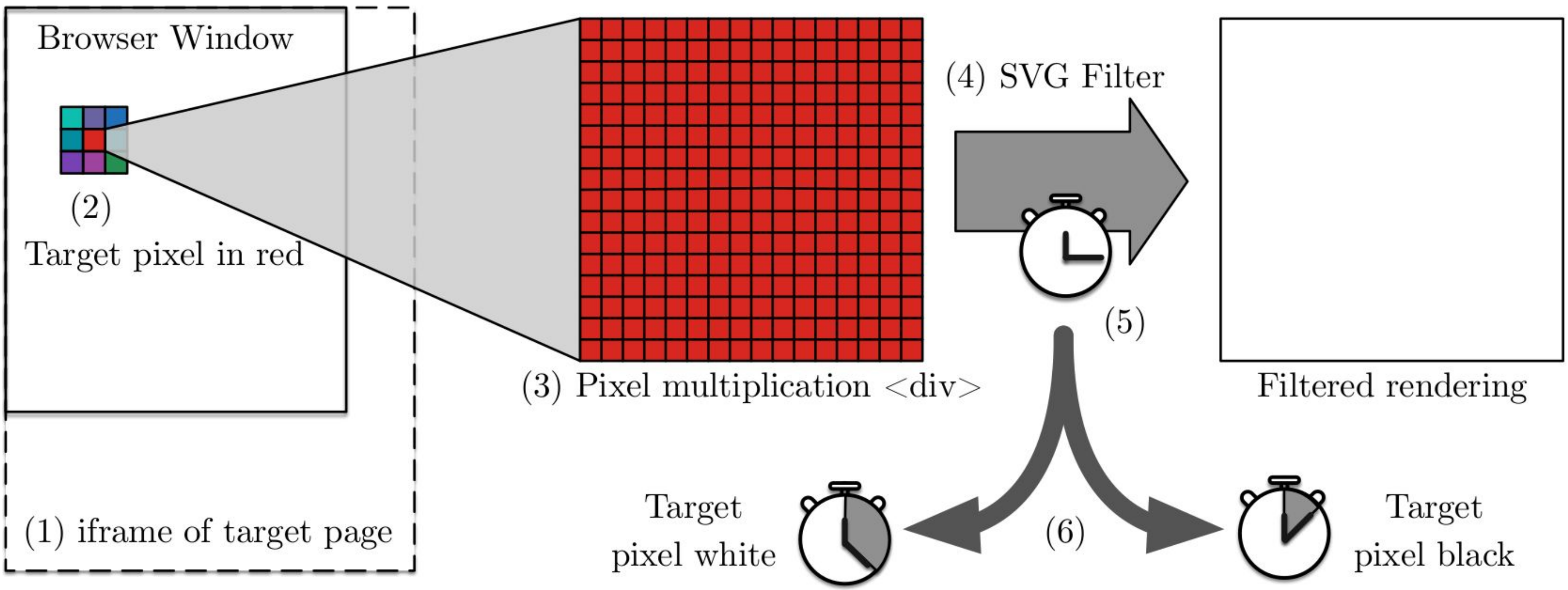
Pixel-stealing attack overview



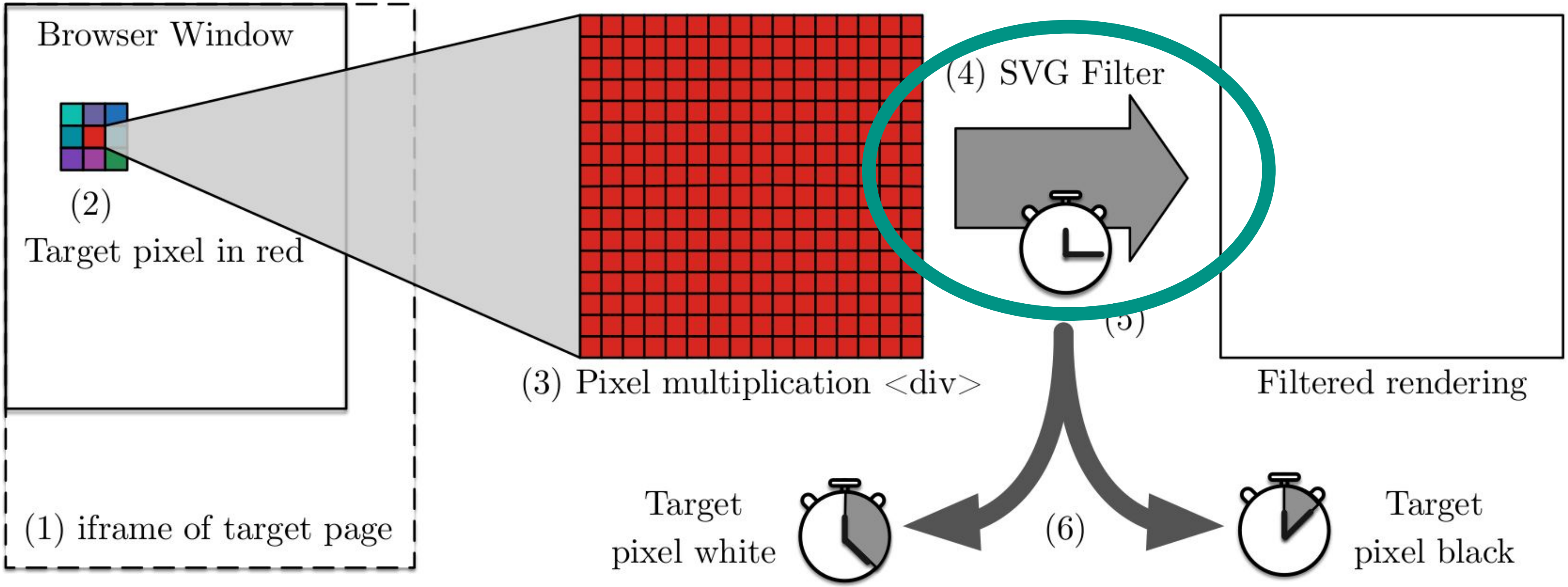
Pixel-stealing attack overview



Pixel-stealing attack overview



Pixel-stealing attack overview



See Paul Stone's "Pixel Perfect Timing Attacks with HTML5" and
Andrysco et al's "On subnormal floating point and abnormal timing"

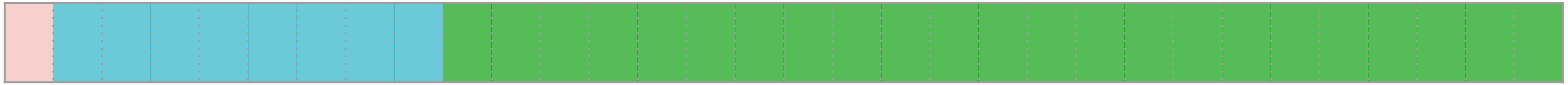
Pixel-stealing attack overview

Floating point format and performance

- Pixel-stealing attacks
- Floating-point benchmarking
- Attacking with floats
- Beating defenses
- Conclusions

IEEE-754 Floating point format (single/float)

exponent (8 bits) *significand* (23 bits +1)

sign 

$$\text{Value} = (-1)^{\textit{sign}} \times \textit{significand} \times 2^{(\textit{exponent} - \textit{bias})}$$

IEEE-754 Floating point format (single/float)

exponent (8 bits) *significand* (23 bits +1)

sign	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	1	0	0	1	0	1	0	1	1	1	0	1	0	1

Value = $(-1)^{sign} \times significand \times 2^{(exponent\ -\ bias)}$

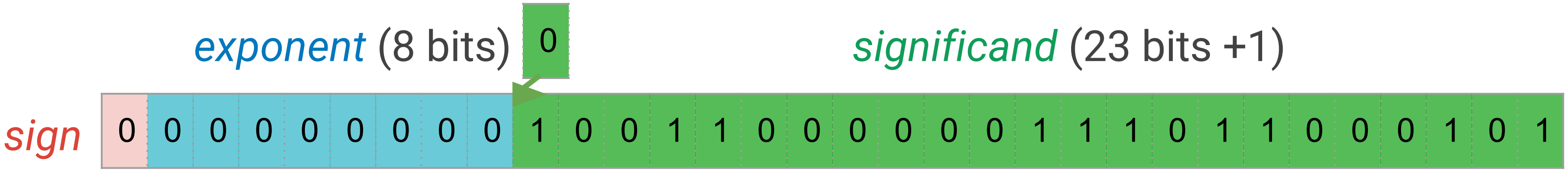
Normal values have nonzero *exponent*, implicit leading 1. before *significand*

IEEE-754 Floating point format (single/float)

Value = $(-1)^{\textit{sign}} \times \textit{significand} \times 2^{(\textit{exponent} - \textit{bias})}$

Normal values have nonzero *exponent*, implicit leading 1. before *significand*

IEEE-754 Floating point format (single/float)



Value = $(-1)^{sign} \times significand \times 2^{(exponent\ -\ bias)}$

Normal values have nonzero *exponent*, implicit leading 1. before *significand*

Subnormal values have all-zero *exponent*, implicit leading 0. before *significand*

IEEE-754 Floating point format (single/float)

Value = $(-1)^{sign} \times significand \times 2^{(exponent\ -\ bias)}$

Normal values have nonzero *exponent*, implicit leading 1. before *significand*

Subnormal values have all-zero *exponent*, implicit leading 0. before *significand*

Single precision normal minimum: 1.18e−38 !

	Divisor								
Dividend	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

double-precision SSE scalar division on Intel i5-4460

Floating point performance variation

	Divisor								
Dividend	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	$\boxed{12.25}$	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

double-precision SSE scalar division on Intel i5-4460

Floating point performance variation

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

double-precision SSE scalar division on Intel i5-4460

Floating point performance variation

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	$\boxed{6.52}$	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

double-precision SSE scalar division on Intel i5-4460

Floating point performance variation

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

double-precision SSE scalar division on Intel i5-4460

* Ask me why it is 175 cycles

Floating point performance variation

Operation	Default
Single Precision	
Add/Sub	–
Mul	Slow subnormal
Div	Slow subnormal
Sqrt	Many effects
Double Precision	
Add/Sub	–
Mul	Slow subnormal
Div	Many effects
Sqrt	Many effects

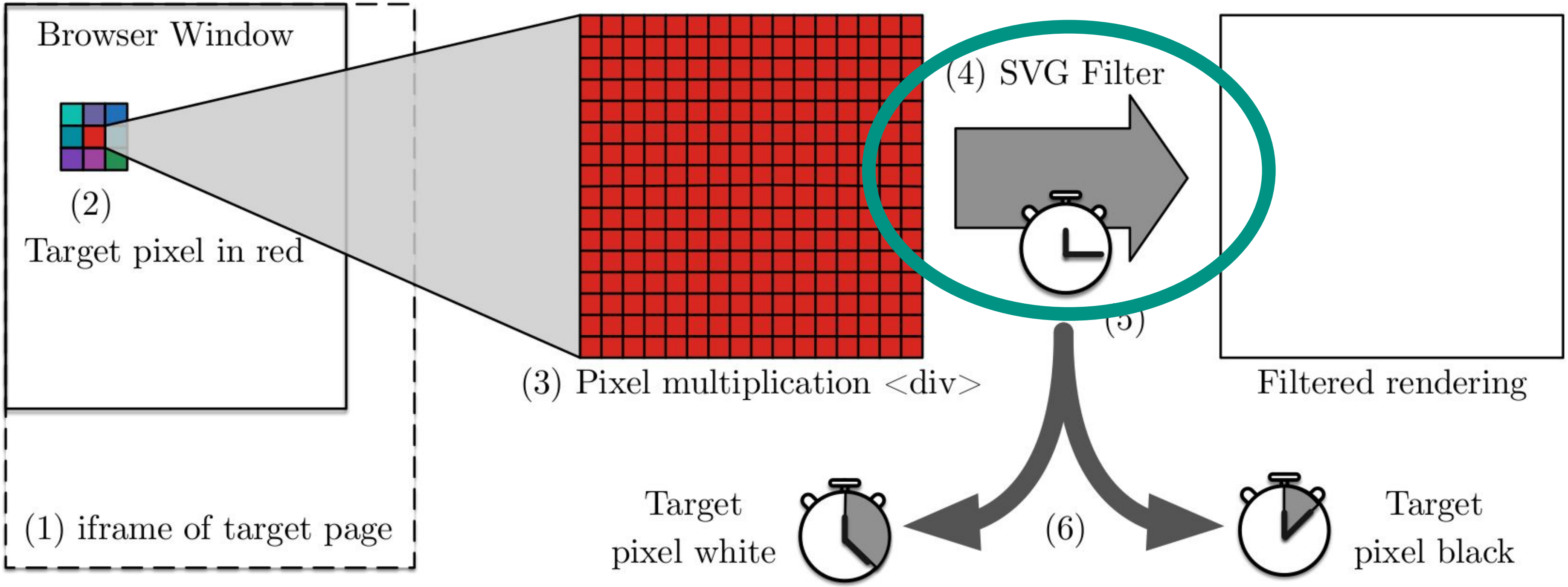
Intel i5-4460

- **Slow subnormals**: Subnormal operands induce slowdowns
- **Fast zero:** All zero significands cause speedups
- **Many effects:** Analog combinations of previous effects

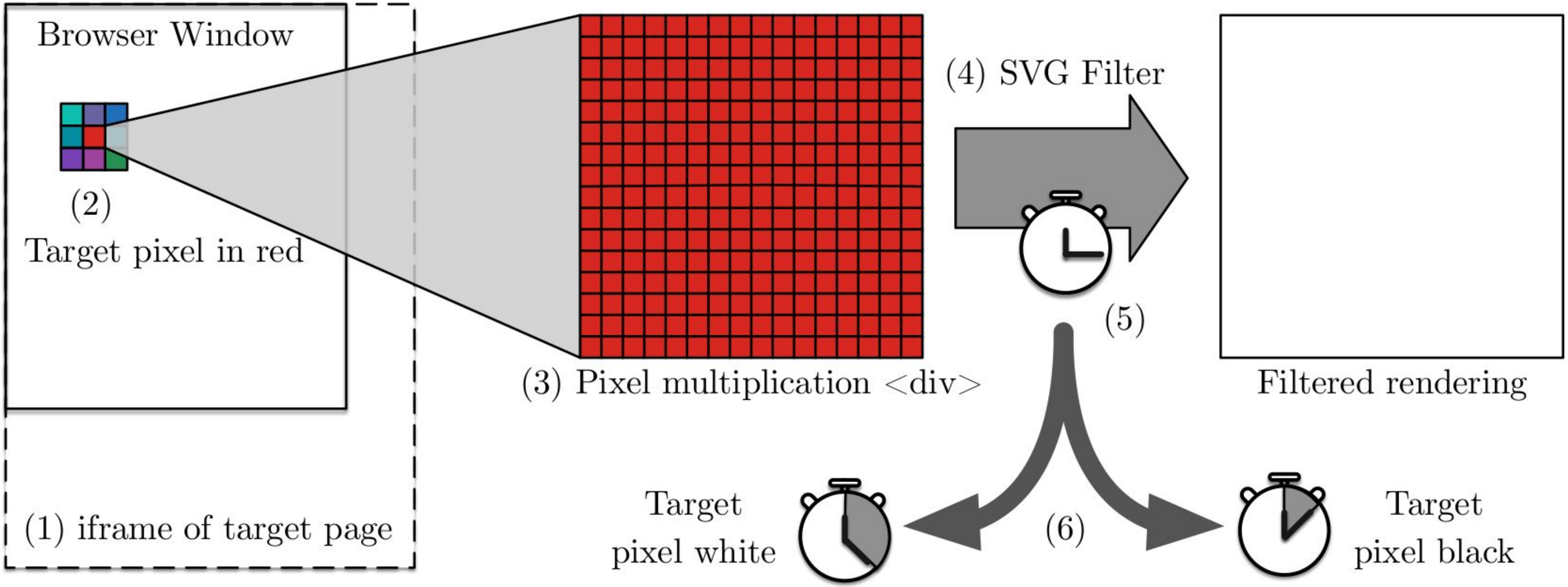
Summary of floating-point performance variations

Using floats for pixel-stealing

- Pixel-stealing attacks
- Floating-point benchmarking
- Attacking with floats
- Beating defenses
- Conclusions

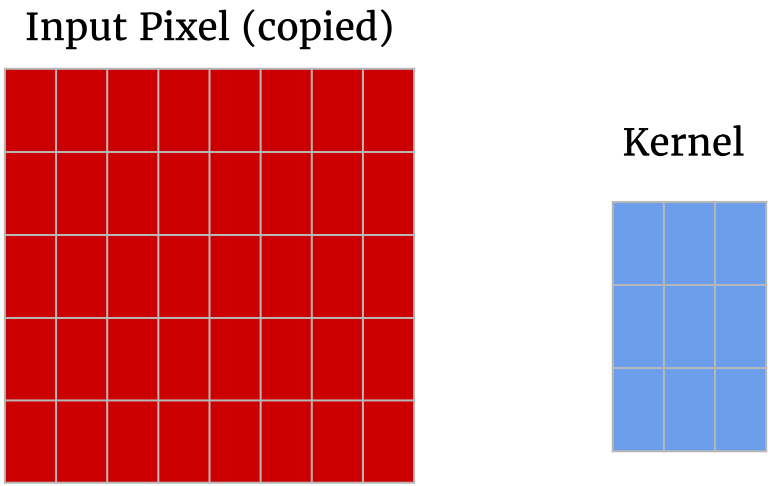


Pixel-stealing attack overview

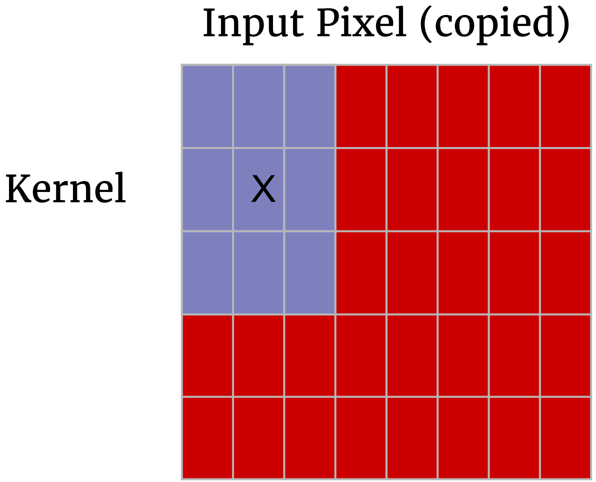


Pixel-stealing attack overview

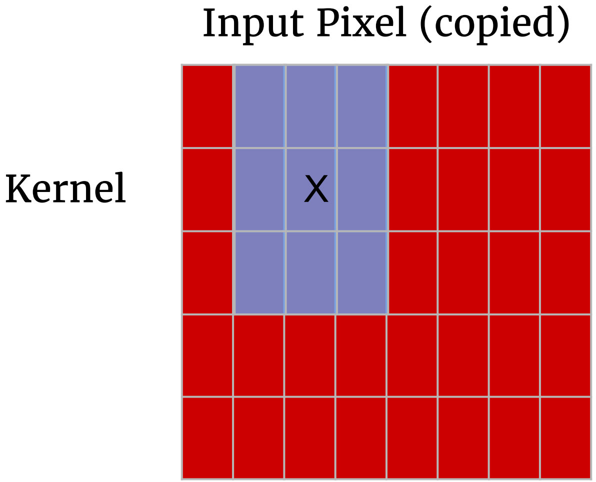
`feConvolveMatrix` as a target



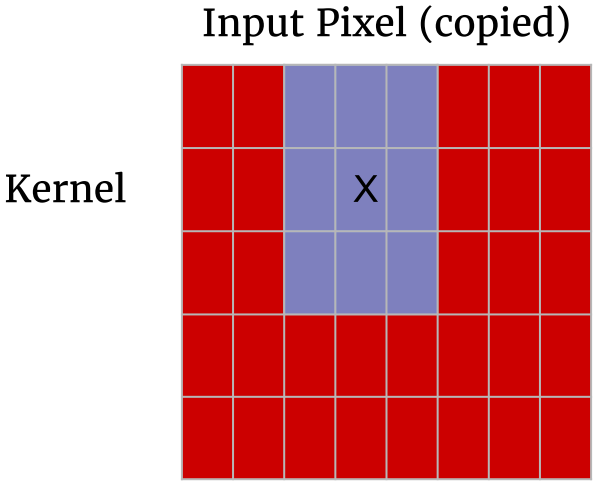
`feConvolveMatrix` as a target



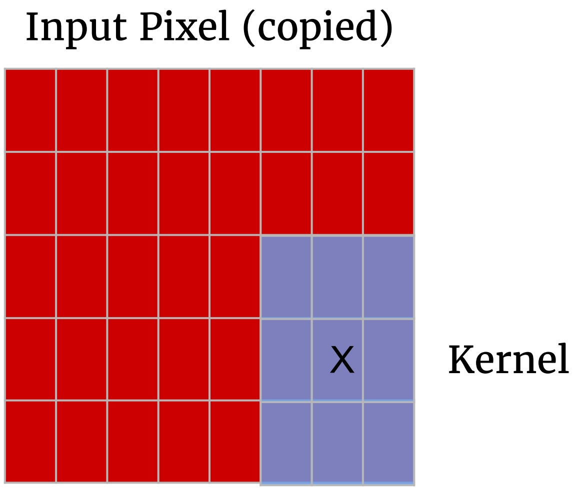
`feConvolveMatrix` as a target



`feConvolveMatrix` as a target



`feConvolveMatrix` as a target



Pixel-stealing on 3 major browsers

- Pixel-stealing attacks
- Floating-point benchmarking
- Attacking with floats
- Beating defenses
    - Browsers
    - Escort
- Conclusions

Safari

- No response to previous attacks
- `feConvolveMatrix` still a target

Safari

- No response to previous attacks
- `feConvolveMatrix` still a target

- Attack modifications
  - Frame counting
  - Pixel-expansion

Safari

- No response to previous attacks
- `feConvolveMatrix` still a target

- Attack modifications
  - Frame counting
  - Pixel-expansion

- Most stable of all attacks!

Firefox

- Switch to *fixed-point* in Firefox 28
    - `feConvolveMatrix` no longer vulnerable

Firefox

- Switch to *fixed-point* in Firefox 28
  - `feConvolveMatrix` no longer vulnerable

- We can use other filters!
  - `feSpecularLighting`
  - Not ported to fixed point yet

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);
```

Chrome + FPU Flags

- Disable subnormals with *FTZ/DAZ*
    - FPU state flags
    - Difficult to manage

Chrome + FPU Flags

- Disable subnormals with *FTZ/DAZ*
  - FPU state flags
  - Difficult to manage
  - Not always effective!

Dividend	Divisor 0.0	1.0	1e10	1e+200	1e-300	1e-42	256	257	1e-320
	Cycle count								
0.0	6.58	6.59	6.58	6.55	6.59	6.54	6.54	6.56	6.56
1.0	6.55	6.55	12.23	12.19	12.22	12.22	6.56	12.25	6.56
1e10	6.58	6.59	12.22	12.22	12.21	12.21	6.59	12.23	6.59
1e+200	6.57	6.59	12.22	12.20	12.17	12.21	6.58	12.17	6.57
1e-300	6.59	6.57	12.18	12.23	12.24	12.22	6.59	12.24	6.57
1e-42	6.58	6.56	12.21	12.25	12.23	12.18	6.56	12.21	6.58
256	6.57	6.60	12.20	12.22	12.24	12.24	6.57	12.23	6.54
257	6.57	6.58	12.22	12.23	12.25	12.20	6.57	12.23	6.58
1e-320	6.57	6.58	6.60	6.51	6.59	6.57	6.58	6.55	6.58

Division timing for double precision floats on Intel i5-4460+FTZ/DAZ

FTZ/DAZ benchmarking – Bad news

Dividend	Divisor								
	0.0	1.0	1e10	1e+200	1e-300	1e-42	256	257	1e-320
	Cycle count								
0.0	6.58	6.59	6.58	6.55	6.59	6.54	6.54	6.56	6.56
1.0	6.55	6.55	12.23	12.19	12.22	12.22	6.56	12.25	6.56
1e10	6.58	6.59	12.22	12.22	12.21	12.21	6.59	12.23	6.59
1e+200	6.57	6.59	12.22	12.20	12.17	12.21	6.58	12.17	6.57
1e-300	6.59	6.57	12.18	12.23	12.24	12.22	6.59	12.24	6.57
1e-42	6.58	6.56	12.21	12.25	12.23	12.18	6.56	12.21	6.58
256	6.57	6.60	12.20	12.22	12.24	12.24	6.57	12.23	6.54
257	6.57	6.58	12.22	12.23	12.25	12.20	6.57	12.23	6.58
1e-320	6.57	6.58	6.60	6.51	6.59	6.57	6.58	6.55	6.58

Division timing for double precision floats on Intel i5-4460+FTZ/DAZ

FTZ/DAZ benchmarking – Bad news

Dividend	Divisor 0.0	1.0	1e10	1e+30	1e-30	1e-41	1e-42	256	257
	Cycle count								
0.0	5.66	5.59	5.62	5.64	5.64	5.62	5.66	5.65	5.60
1.0	5.66	5.66	5.65	5.66	5.65	5.63	5.65	5.66	5.65
1e10	5.65	5.65	5.62	5.63	5.62	5.64	5.64	5.62	5.63
1e+30	5.65	5.62	5.62	5.59	5.65	5.65	5.64	5.65	5.65
1e-30	5.62	5.63	5.61	5.58	5.63	5.65	5.60	5.64	5.64
1e-41	5.64	5.66	5.64	5.63	5.65	5.65	5.65	5.65	5.66
1e-42	5.65	5.61	5.65	5.62	5.63	5.62	5.64	5.64	5.64
256	5.66	5.63	5.64	5.64	5.64	5.65	5.63	5.60	5.61
257	5.65	5.64	5.65	5.57	5.63	5.65	5.65	5.63	5.63

Division timing for single precision floats on Intel i5-4460+FTZ/DAZ

FTZ/DAZ benchmarking – Good news

Operand	Cycle count
0.0	8.85
1.0	8.85
1e10	11.60
1e+30	11.68
1e-30	11.63
1e-41	8.84
1e-42	8.84
256	8.84
257	11.62

Square root timing for single precision floats on Intel i5-4460+FTZ/DAZ

FTZ/DAZ benchmarking - Bad news

Chrome + FPU Flags

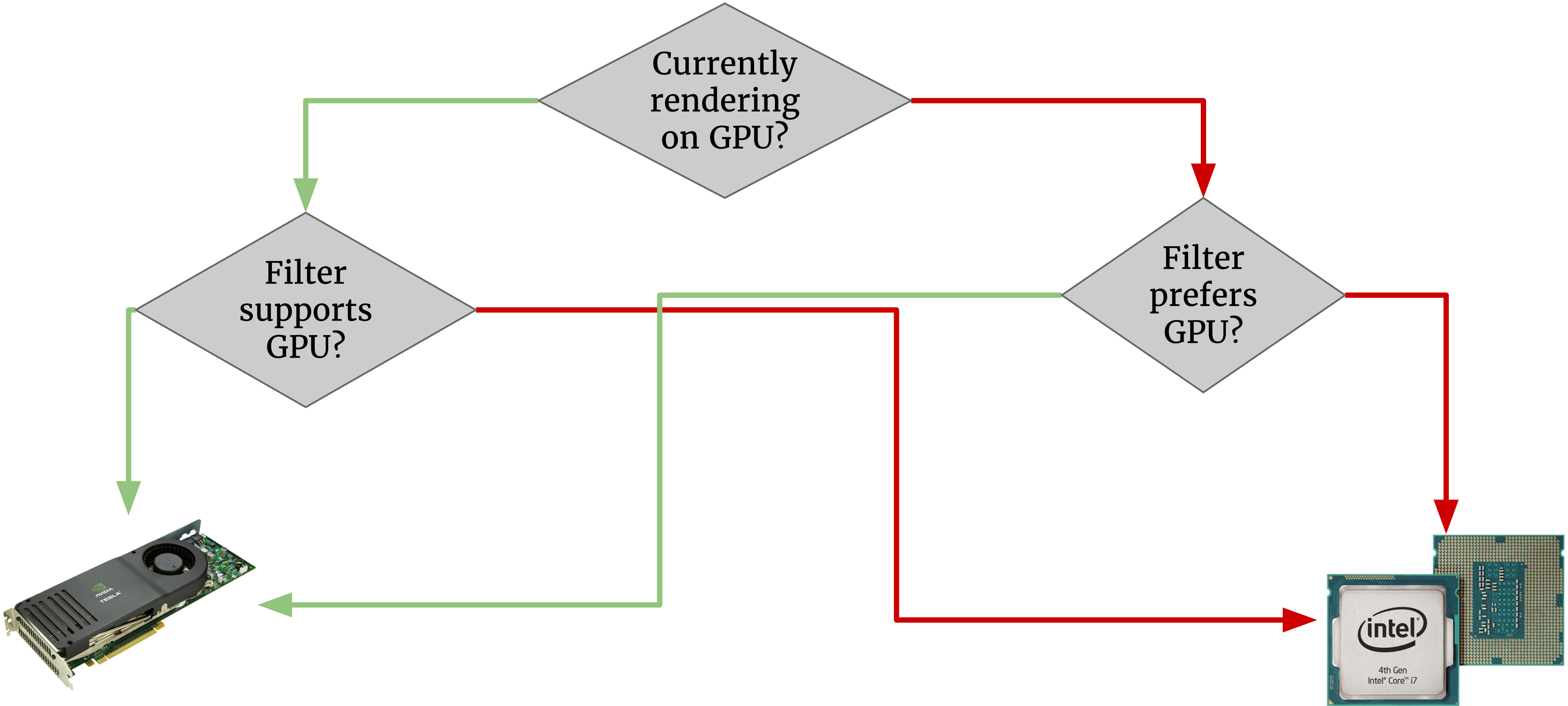
- Disable subnormals with *FTZ/DAZ*
  - FPU state flags
  - Difficult to manage
  - Not always effective!

Chrome + FPU Flags

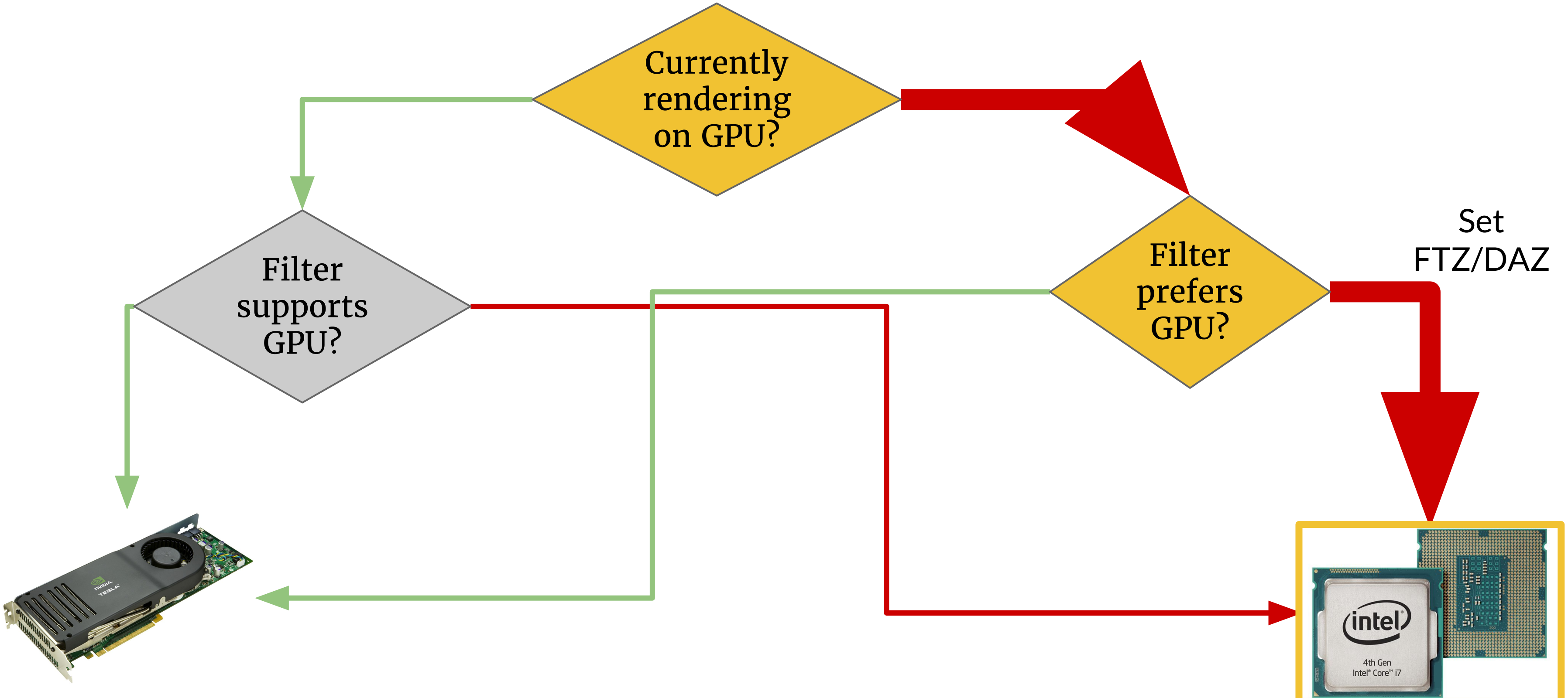
- Disable subnormals with *FTZ/DAZ*
  - FPU state flags
  - Difficult to manage
  - Not always effective!

- Filter on GPU then bail to CPU
  - Doesn’t set FPU flags correctly

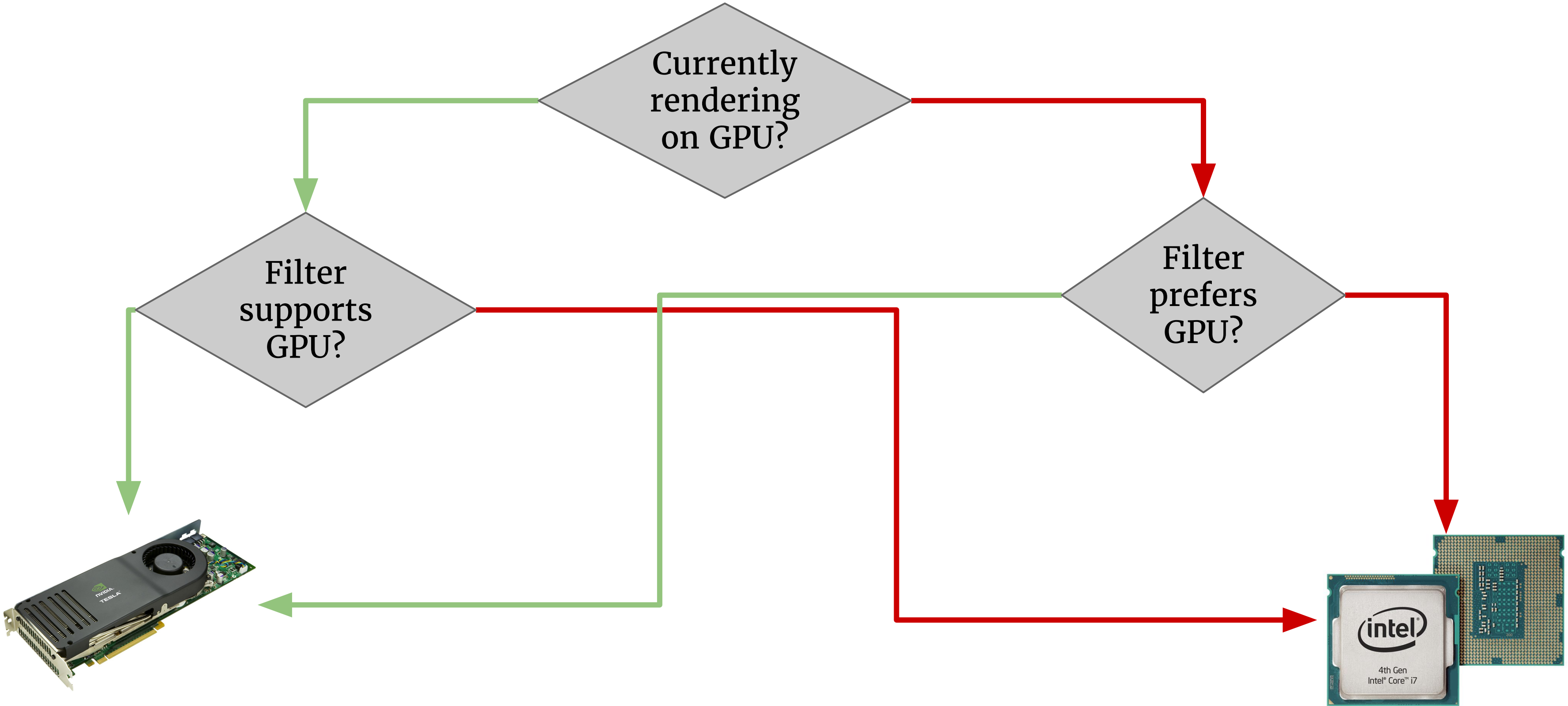
Chrome filter rendering flow



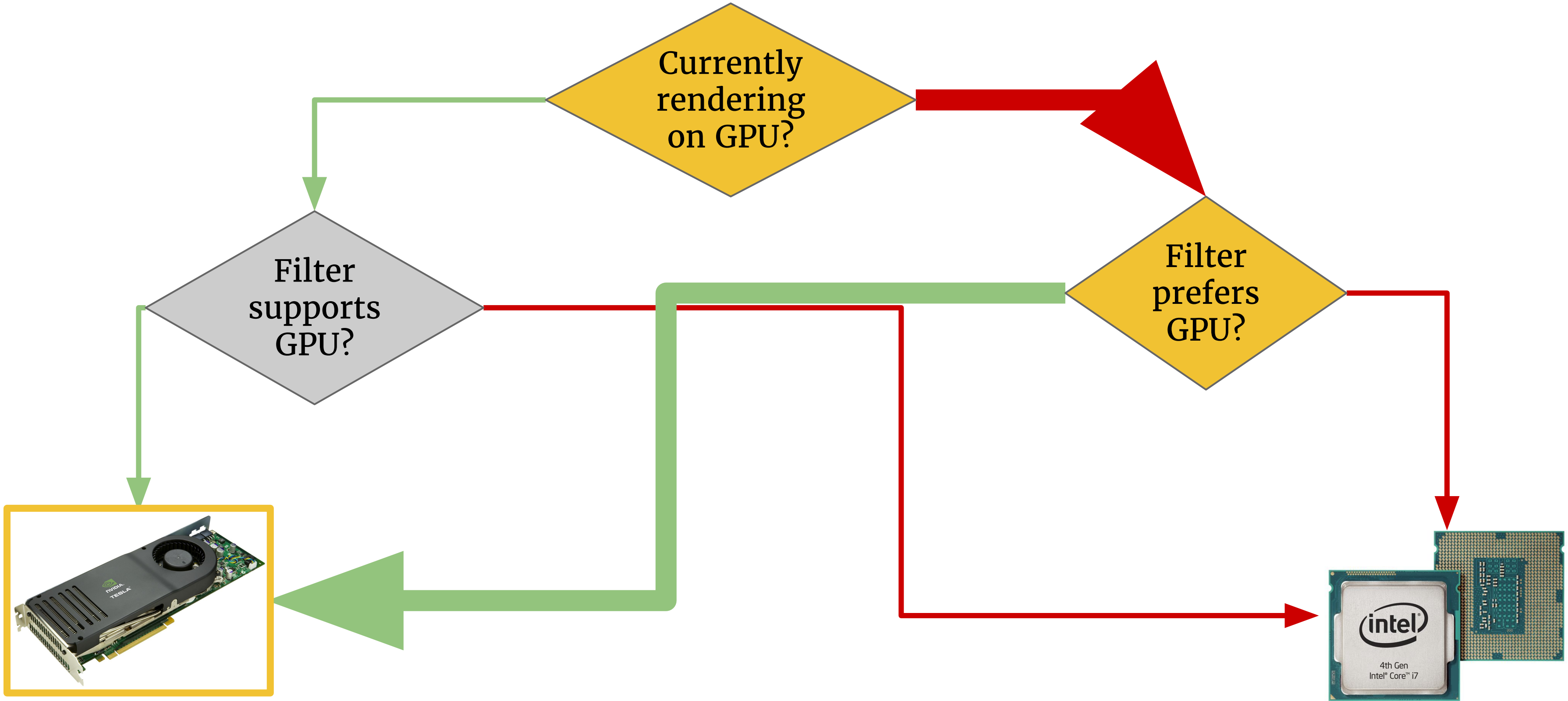
Chrome filter rendering flow – default



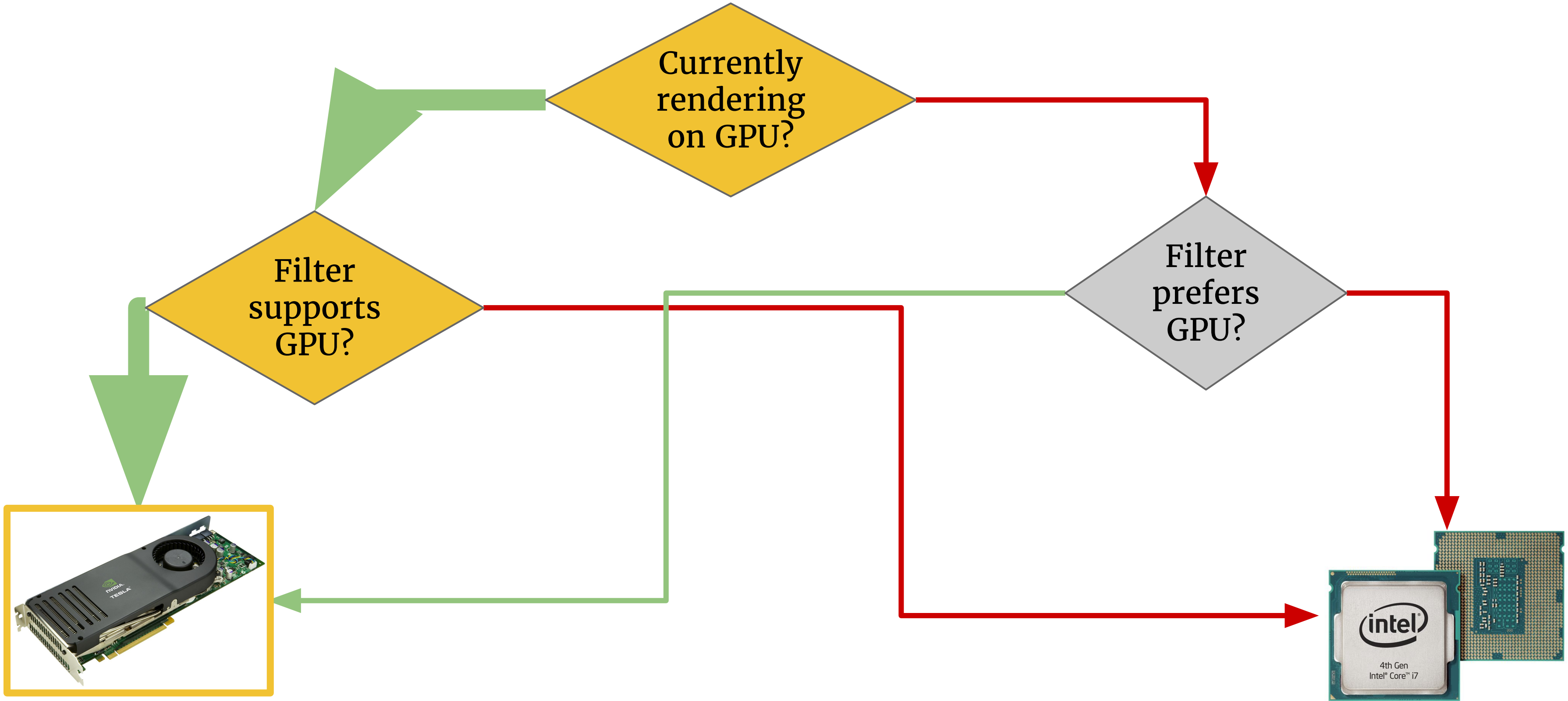
Chrome attack flow



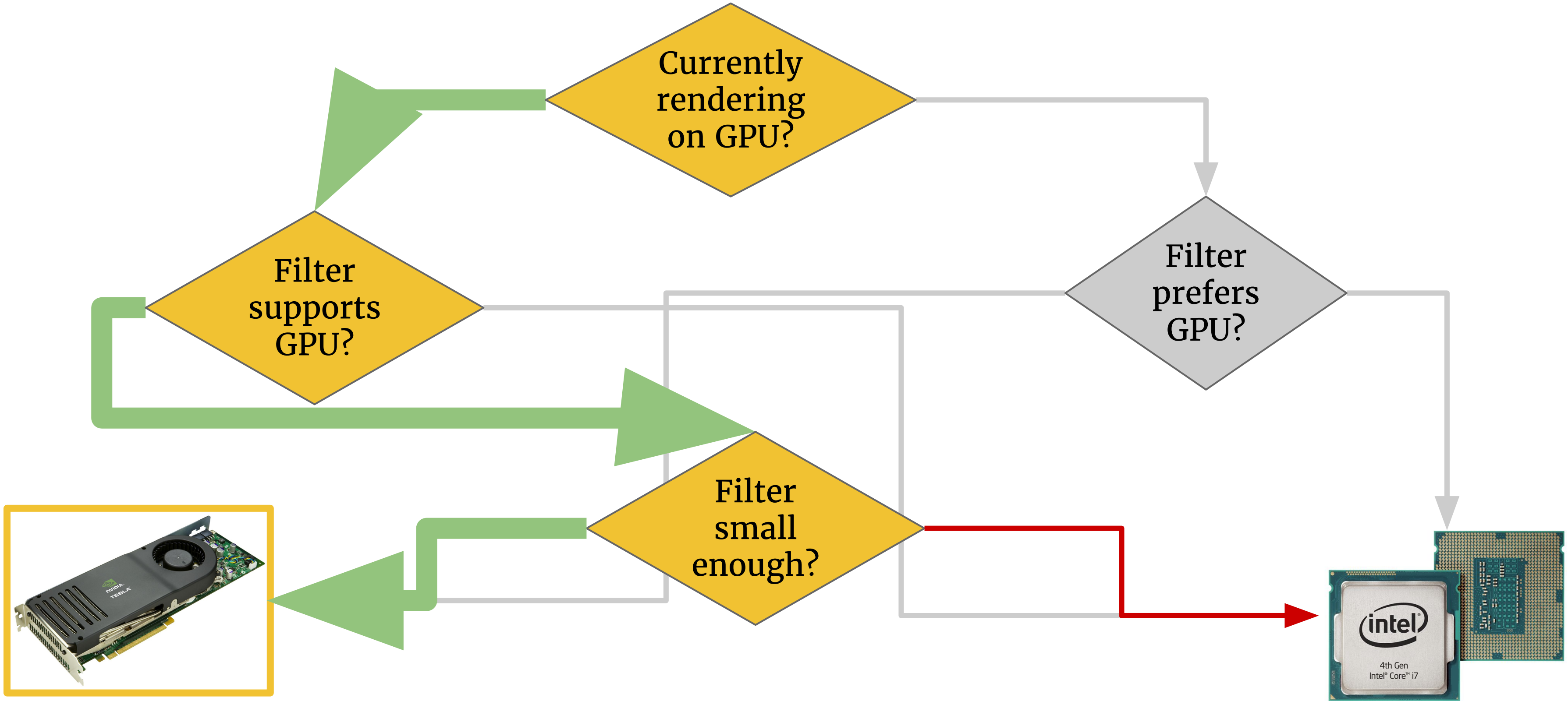
Chrome attack flow – Force to GPU



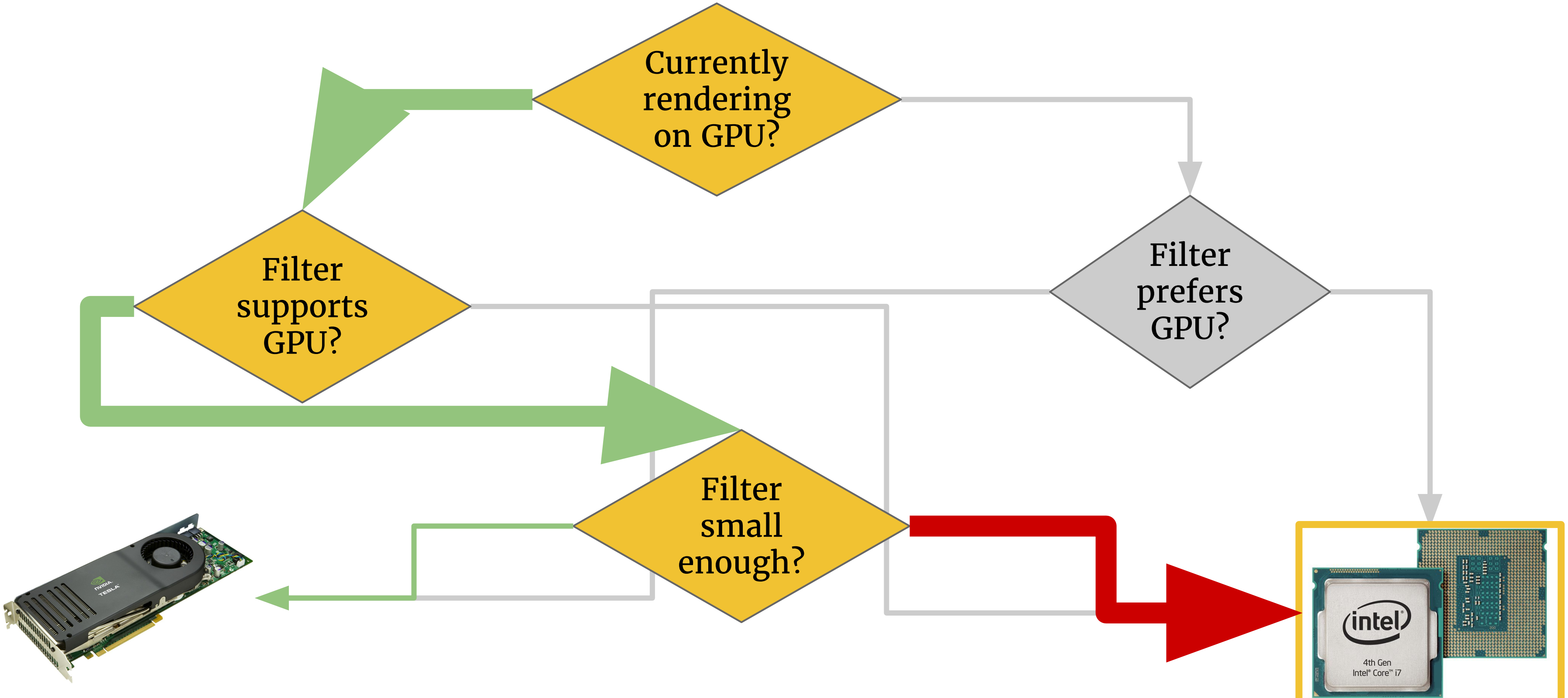
Chrome attack flow – `feConvolveMatrix`



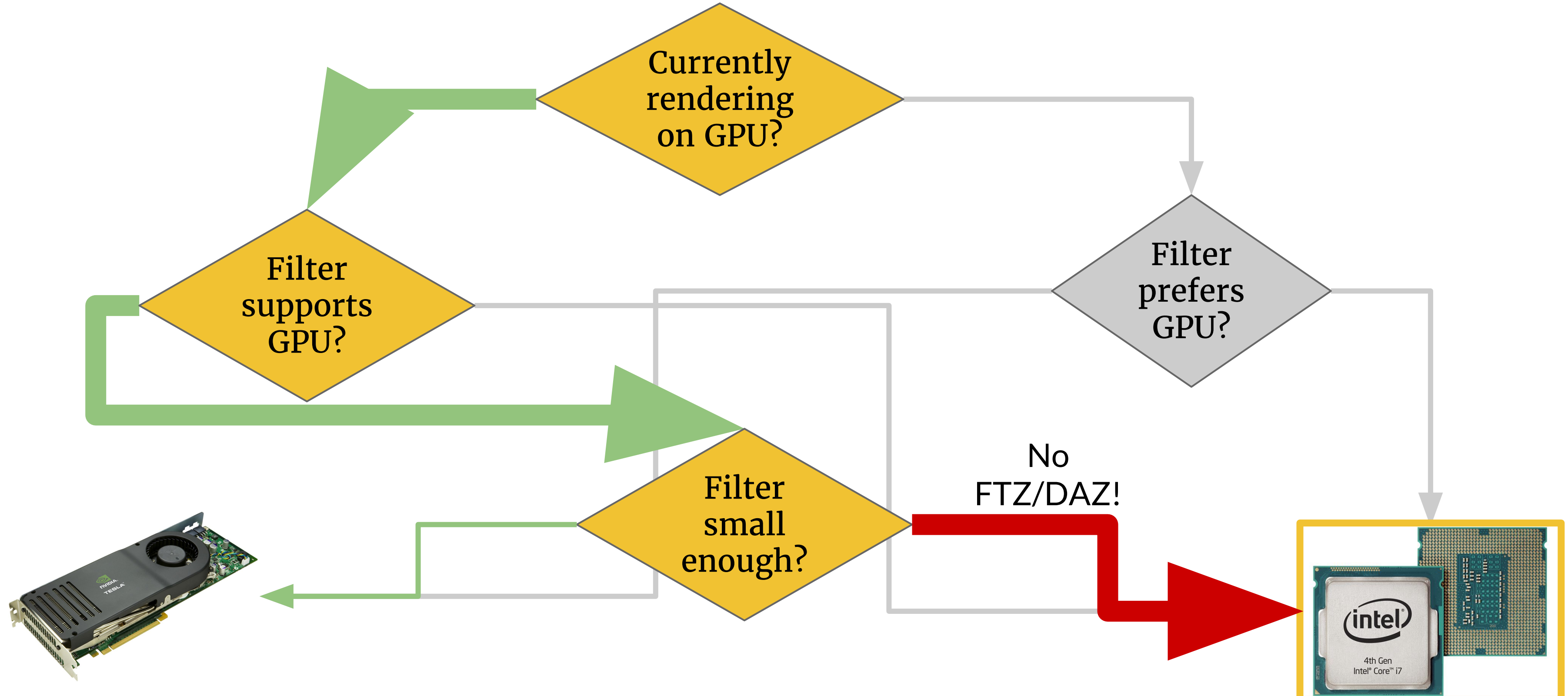
Chrome attack flow – `feConvolveMatrix`



Chrome attack flow – `feConvolveMatrix`



Chrome attack flow – `feConvolveMatrix`

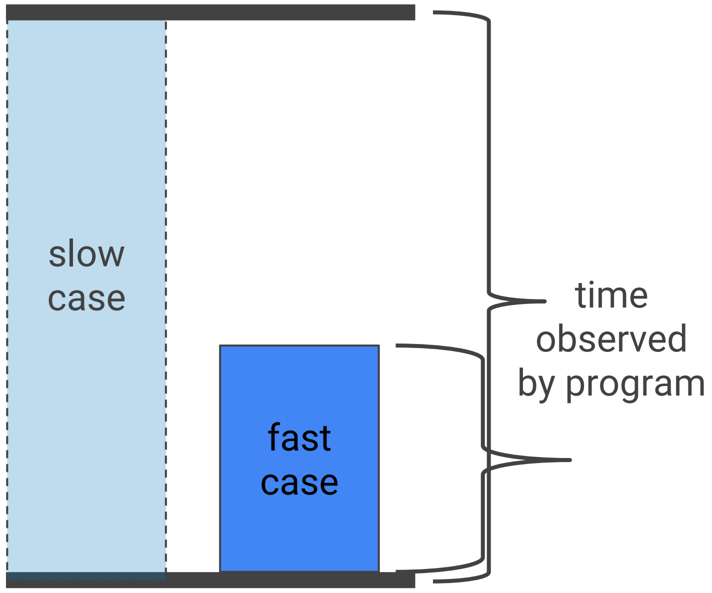


Examining Escort: a proposed hardware-based defense

- Pixel-stealing attacks
- Floating-point benchmarking
- Attacking with floats
- Beating defenses
  - Browsers
  - Escort
- Conclusions

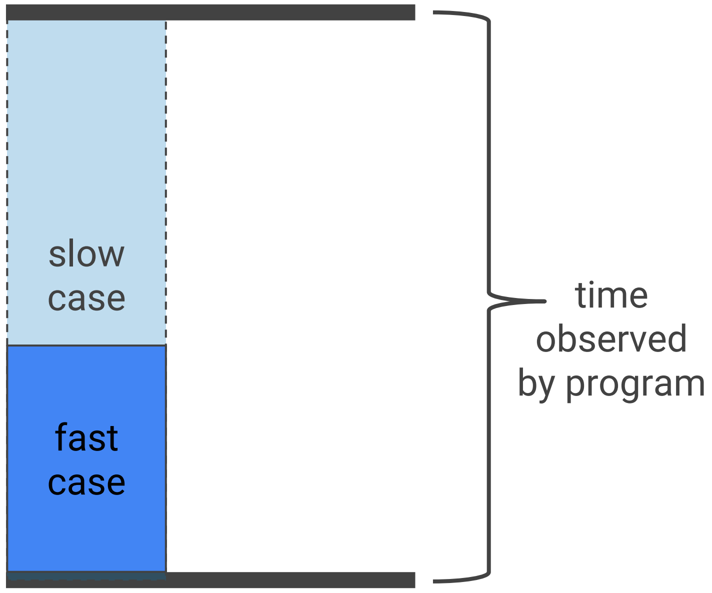
Escort – Rane, Lin and Tiwari

- New hardware-based approach
- Run (some) FP ops on SIMD unit
- dummy “escort” op, runs worst-case



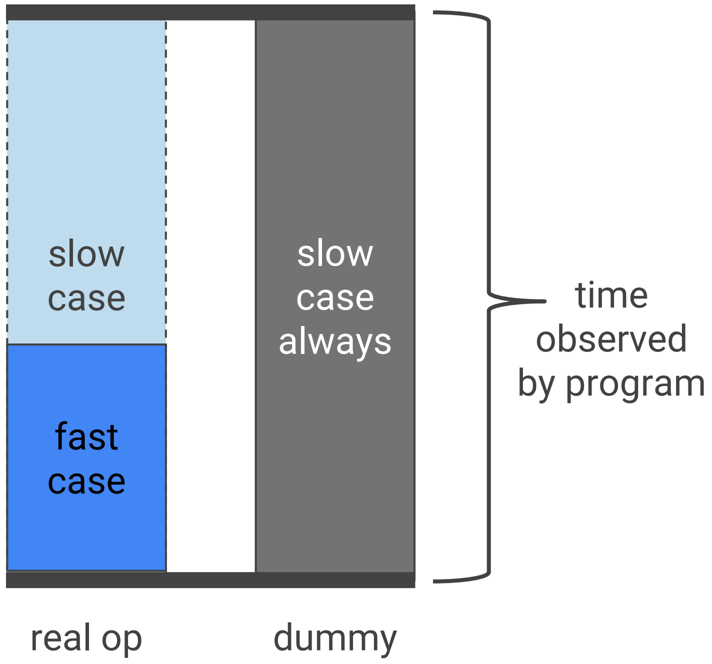
Escort – Rane, Lin and Tiwari

- New hardware-based approach
- Run (some) FP ops on SIMD unit
- dummy “escort” op, runs worst-case



Escort – Rane, Lin and Tiwari

- New hardware-based approach
- Run (some) FP ops on SIMD unit
- dummy “escort” op, runs worst-case
- Conjecture: real, dummy ops run in parallel



Dividend	Divisor 0.0	1.0	1e10	1e+200	1e-300	1e-42	256	257	1e-320
	Cycle count								
0.0	186.46	186.48	186.50	186.44	186.42	186.49	186.50	186.48	186.51
1.0	186.45	186.48	195.93	195.94	195.93	195.86	186.48	195.87	186.48
1e10	186.51	186.49	195.92	195.90	195.92	195.87	186.47	195.86	186.46
1e+200	186.50	186.50	195.90	195.94	195.89	195.91	186.46	195.90	186.50
1e-300	186.48	186.44	195.91	195.88	195.93	195.92	186.53	195.95	186.44
1e-42	186.44	186.51	195.92	195.94	195.87	195.89	186.51	195.93	186.47
256	186.49	186.49	195.91	195.91	195.87	195.89	186.45	195.91	186.44
257	186.46	186.47	195.96	195.92	195.92	195.96	186.49	195.98	186.45
1e-320	186.49	186.49	186.43	186.48	186.49	186.49	186.50	186.52	186.46

Division timing for double precision floats on Intel i5-4460+Escort

Escort `libdrag` benchmarking

	0.0	1.0	1e10	1e+200	1e-300	1e-42	256	257	1e-320
					Cycle count				
0.0	136.55	136.59	136.45	136.60	136.55	136.55	136.58	136.43	136.56
1.0	136.55	136.56	136.58	136.61	136.63	136.57	136.60	136.57	136.64
1e10	136.58	136.52	136.59	136.58	136.51	136.58	136.58	136.52	136.60
1e+200	136.59	136.51	136.59	136.57	136.57	136.57	136.58	136.59	136.52
1e-300	136.57	136.58	136.55	136.59	136.59	136.60	136.57	136.56	136.55
1e-42	136.52	136.57	136.54	136.57	136.57	136.58	136.53	136.56	136.55
256	136.57	136.56	136.57	136.48	136.57	136.59	136.60	136.53	136.51
257	136.58	136.54	136.61	136.59	136.59	136.57	136.57	136.57	136.64
1e-320	136.60	136.58	136.64	136.58	136.59	136.55	136.59	136.59	136.63

Multiplication timing for double precision floats on Intel i5-4460+Escort

Escort `libdrag` benchmarking

Operation	Default	Escort
Single Precision		
Add/Sub	–	–
Mul	Slow Subnormals	–
Div	Slow Subnormals	Fast Zero
Sqrt	Many effects	Fast Zero
Double Precision		
Add/Sub	–	–
Mul	Slow Subnormals	–
Div	Many effects	Fast Zero
Sqrt	Many effects	Fast Zero

Intel i5-4460

- **Slow subnormals**: Subnormal operands induce slowdowns
- **Fast zero:** All zero significands cause speedups
- **Many effects:** Analog combinations of previous effects

Escort benchmarking summary

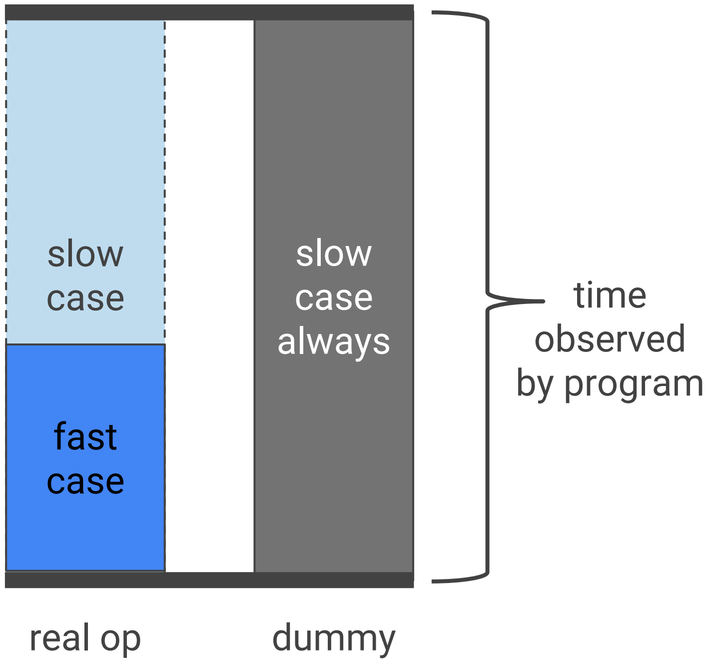
Operation	Escort Intel	Escort AMD
Single Precision		
Add/Sub	–	Slow Subnormals
Mul	–	–
Div	Fast zero	–
Sqrt	Fast zero	–
Double Precision		
Add/Sub	–	Slow Subnormals
Mul	–	–
Div	Fast zero	–
Sqrt	Fast zero	–

Escort benchmarking summary

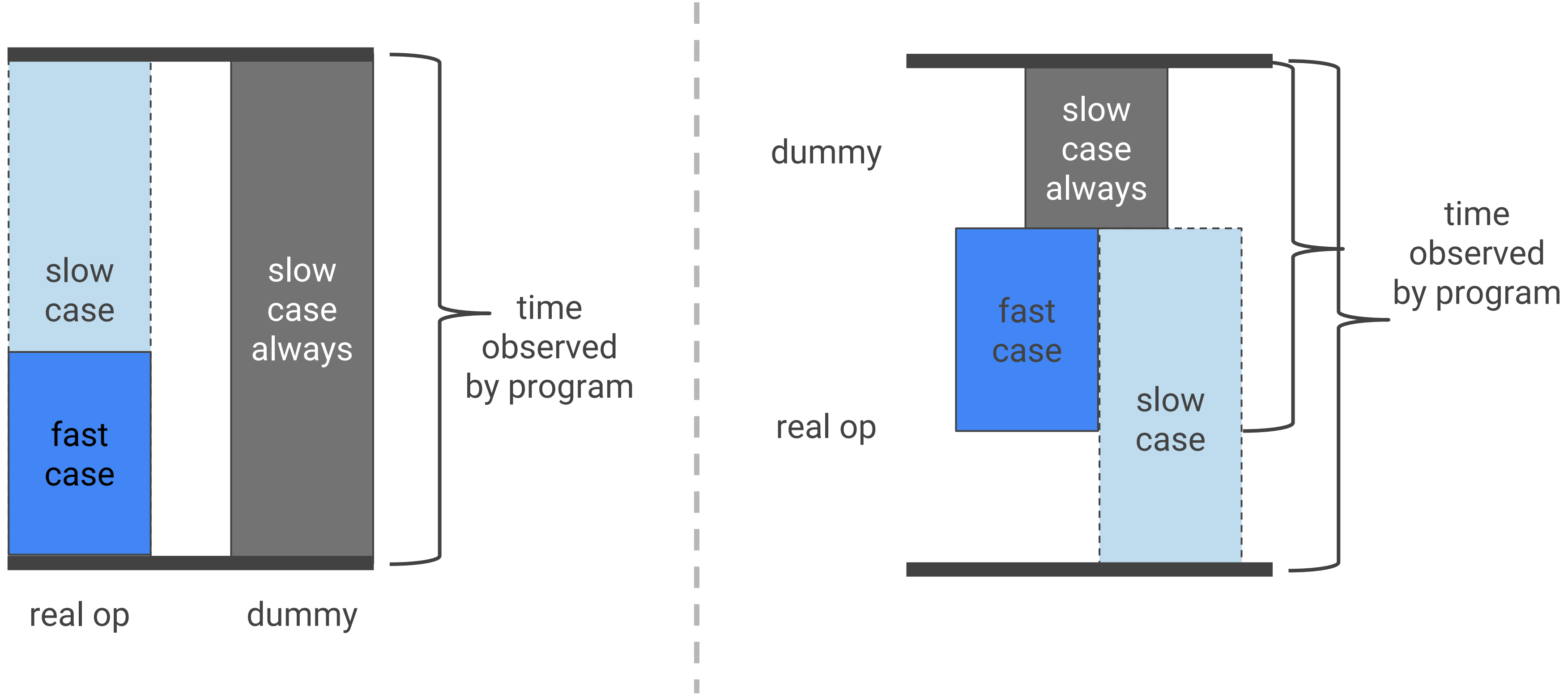
Operation	Escort Intel	Escort AMD
Single Precision		
Add/Sub	–	Slow Subnormals
Mul	–	–
Div	Fast zero	–
Sqrt	Fast zero	–
Double Precision		
Add/Sub	–	Slow Subnormals
Mul	–	–
Div	Fast zero	–
Sqrt	Fast zero	–

Escort benchmarking summary

Escort – Rane, Lin and Tiwari

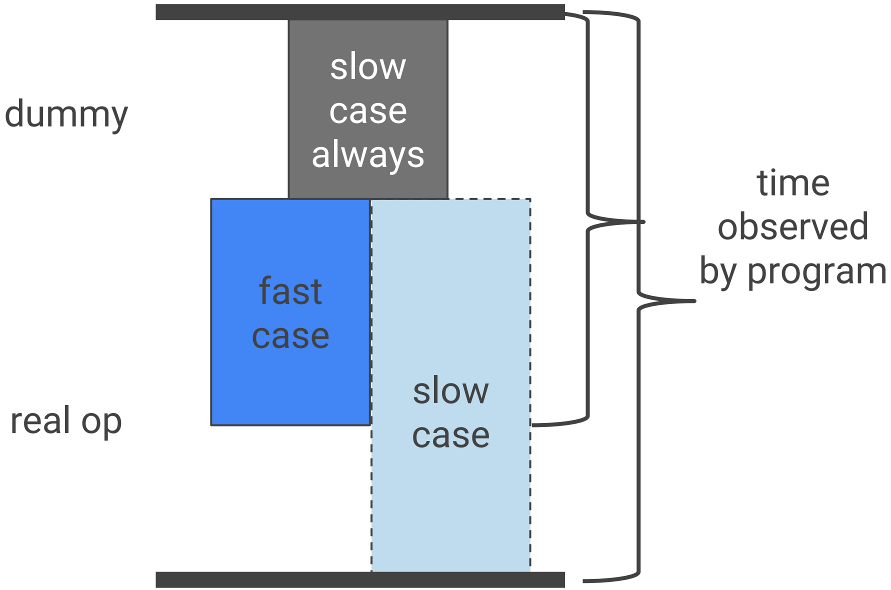


Escort – Rane, Lin and Tiwari



Escort – SIMD Implementation

- All evidence points to:
  - Sequential execution
  - Execution in microcode
  - Slowdown on non-subnormals
- Examined inputs
- Backed up by performance counters
- Potentially useful
  - Tricky to use safely



Fixes deployed and the future

- Pixel-stealing attacks
- Floating-point benchmarking
- Attacking with floats
- Beating defenses
  - Browsers
  - Escort
- Conclusions

Fixes deployed

- Firefox 52 - CVE-2017-5407
  - Restricted range of `surfaceScale` operand

Fixes deployed

- Firefox 52 - CVE-2017-5407
  - Restricted range of `surfaceScale` operand

- Chrome 60 - CVE-2017-5107
  - Ensured FTZ/DAZ scope includes GPU bail

Fixes deployed

- Firefox 52 - CVE-2017-5407
  - Restricted range of `surfaceScale` operand
- Chrome 60 - CVE-2017-5107
  - Ensured FTZ/DAZ scope includes GPU bail
- Safari 10.1.2 - CVE-2017-7006
  - **Removed cross-origin SVG filters!***

* Ask me about history sniffing!

Future

- Other browsers should remove cross-origin SVG

- Fixed-point still very promising
  - `libftfp` proved constant time*

- GPUs, ARM, etc
  - Also probably vulnerable

* Almeida et al USENIX 2016

	Divisor								
Dividend	0.0	1.0	1e10	1e+30	1e-30	1e-41	1e-42	256	257
	Cycle count								
0.0	5.17	5.85	5.85	5.85	5.85	5.89	5.89	5.85	5.85
1.0	6.19	2.59	2.59	2.59	2.59	8.64	8.64	2.59	2.59
1e10	6.19	2.59	2.59	2.59	5.96	8.64	8.64	2.59	2.59
1e+30	6.19	2.59	2.59	2.59	5.96	8.64	8.64	2.59	2.59
1e-30	6.19	2.59	7.82	6.51	2.59	8.40	8.40	2.59	2.59
1e-41	6.19	10.21	8.92	8.92	8.13	8.41	8.41	10.23	10.23
1e-42	6.19	10.21	8.92	8.92	8.13	8.41	8.41	10.23	10.23
256	6.19	2.59	2.59	2.59	2.59	8.64	8.64	2.59	2.59
257	6.19	2.59	2.59	2.59	2.59	8.64	8.64	2.59	2.59

Division timing for single precision floats on Nvidia GeForce GT 430

Floating point performance variation - extra

	0.0	1.0	1e10	1e+30	1e-30	1e-41	1e-42	256	257
	Cycle count								
0.0	7.01	7.01	7.01	7.01	7.01	216.22	216.16	7.01	7.01
1.0	7.01	7.01	7.01	7.01	7.01	48.07	48.06	7.01	7.01
1e10	7.01	7.01	7.01	7.01	7.01	48.06	48.06	7.01	7.01
1e+30	7.01	7.01	7.01	7.01	7.01	48.06	48.06	7.01	7.01
1e-30	7.01	7.01	7.01	7.01	7.01	48.07	48.06	7.01	7.01
1e-41	216.17	48.05	48.05	48.06	48.06	216.20	216.17	48.05	48.05
1e-42	216.22	48.06	48.05	48.05	48.05	216.16	216.16	48.05	48.05
256	7.01	7.01	7.01	7.01	7.01	48.06	48.06	7.01	7.01
257	7.01	7.01	7.01	7.01	7.01	48.06	48.06	7.01	7.01

Addition timing for single precision floats on AMD Phenom II X2 550

Floating point performance variation - extra

Questions?

Tools/Results available:
https://cseweb.ucsd.edu/~dkohlbre/floats

- Pixel-stealing attacks
- Floating-point benchmarking
- Attacking with floats
- Beating defenses
  - Browsers
  - Escort
- Conclusions