ANIMATIONS ON FIRE

Brian Birtles, Mozilla Japan

Graphical Web 2014, Winchester

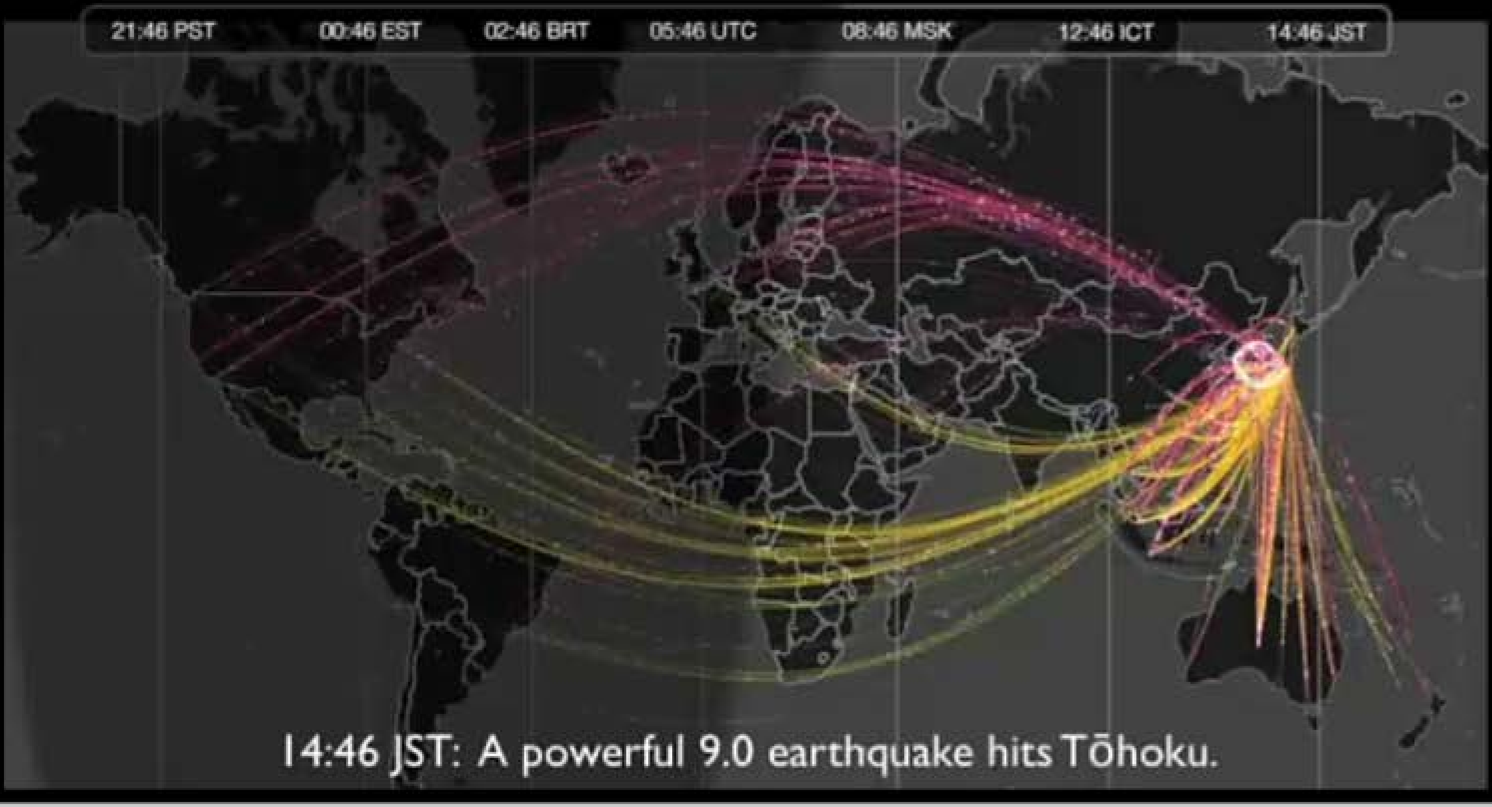
HTML version of slides: http://people.mozilla.org/~bbirtles/pres/graphical-web-2014/

ANIMATION IS AWESOME…

Source: Christopher Price 2013, http://topherchris.com/post/55109717733

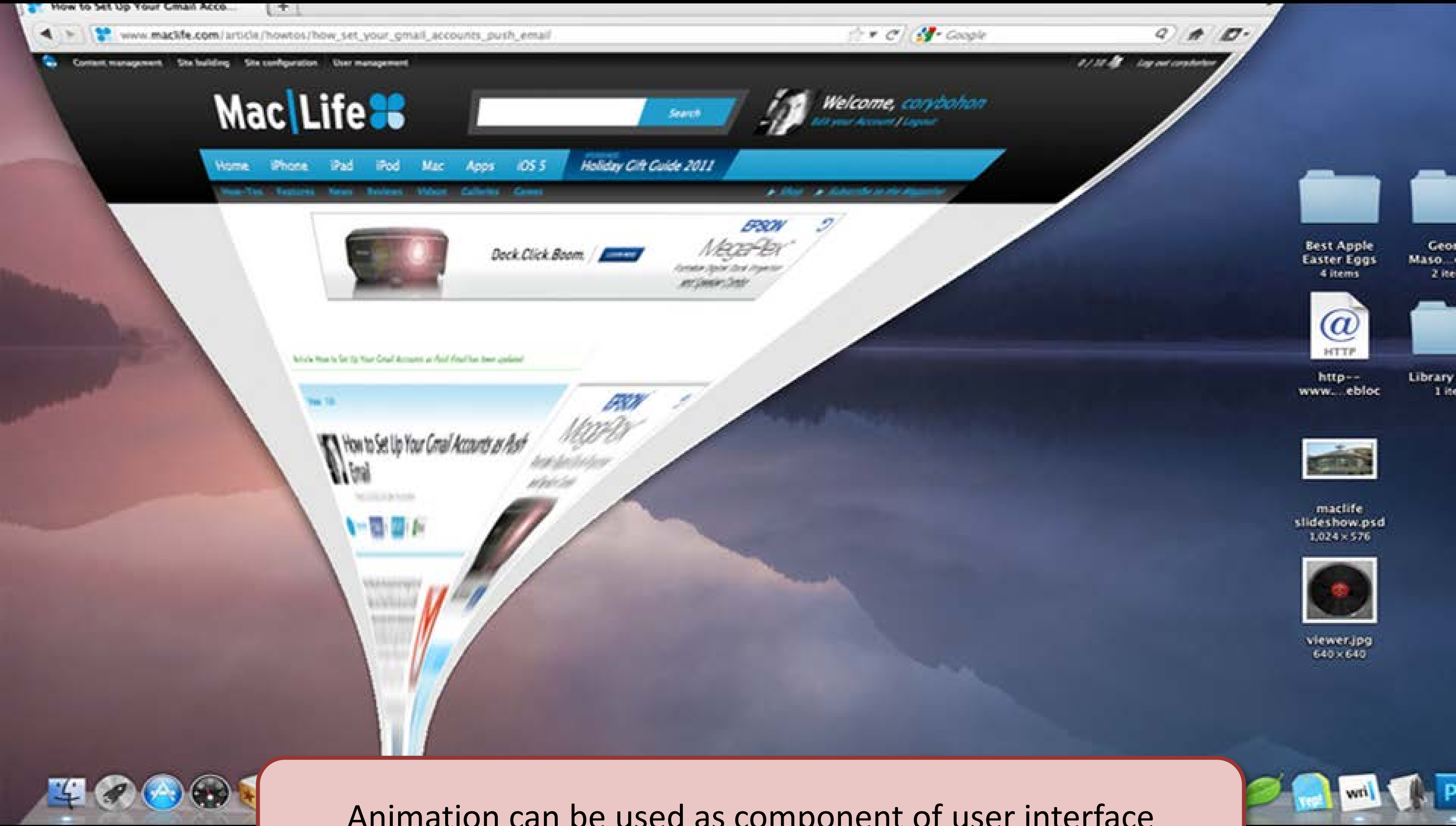
UVNdotTV 2011, http://www.youtube.com/watch?v=Yg5BZARVDAs

Animations can be used for more than just cat gifs.
They can be used to tell stories too.

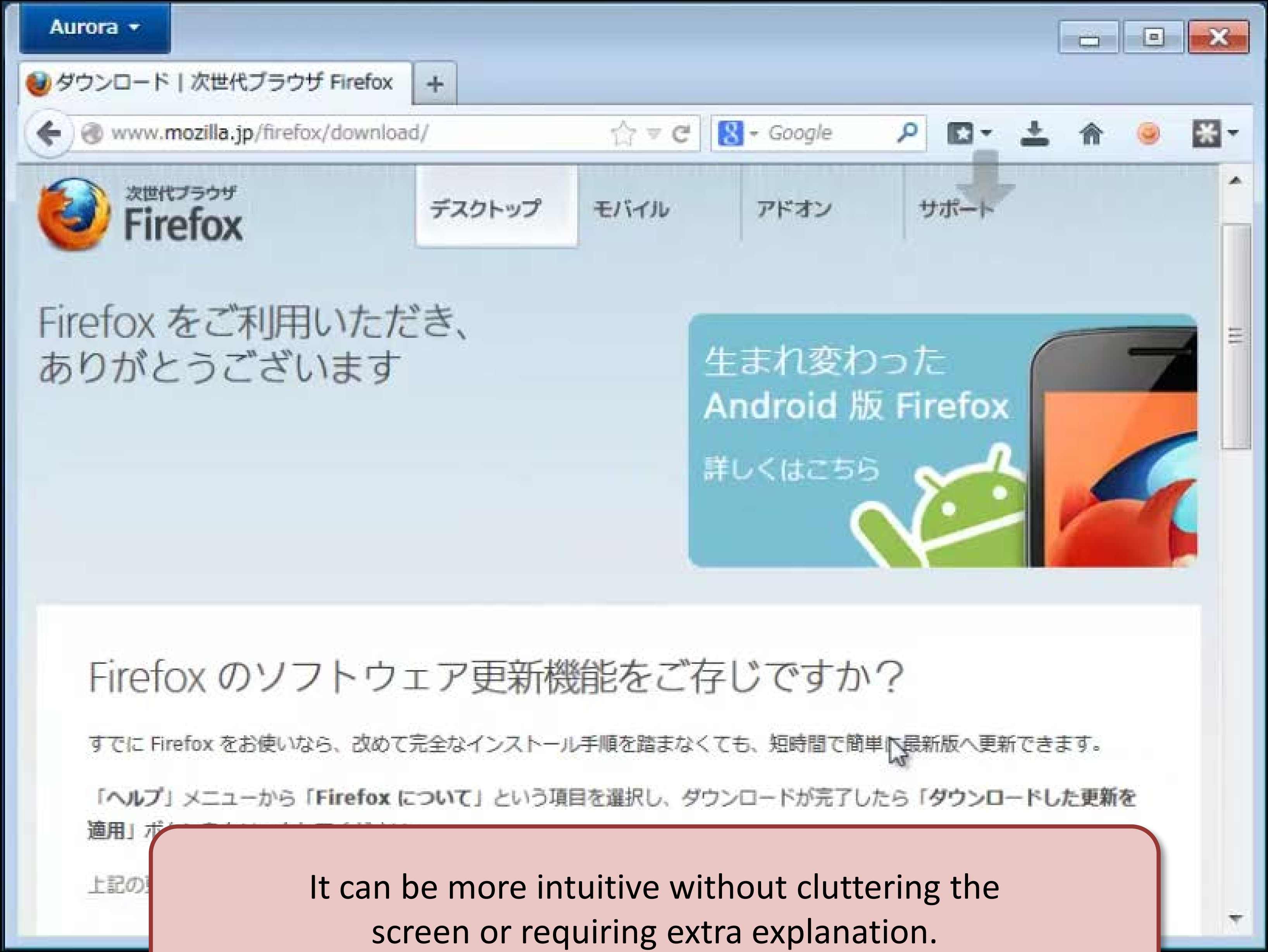


Twitter 2011, http://www.flickr.com/photos/twitteroffice/5885172082/in/photostream/

Animation is essentially about using time
to convey information.



Animation can be used as component of user interface design to describe the results of an action.



Firefox をご利用いただき、
ありがとうございます

Firefox のソフトウェア更新機能をご存じですか？

すでに Firefox をお使いなら、改めて完全なインストール手順を踏まなくても、短時間で簡単に最新版へ更新できます。

「ヘルプ」メニューから「**Firefox について**」という項目を選択し、ダウンロードが完了したら「**ダウンロードした更新を適用**」ボ

上記の

It can be more intuitive without cluttering the
screen or requiring extra explanation.

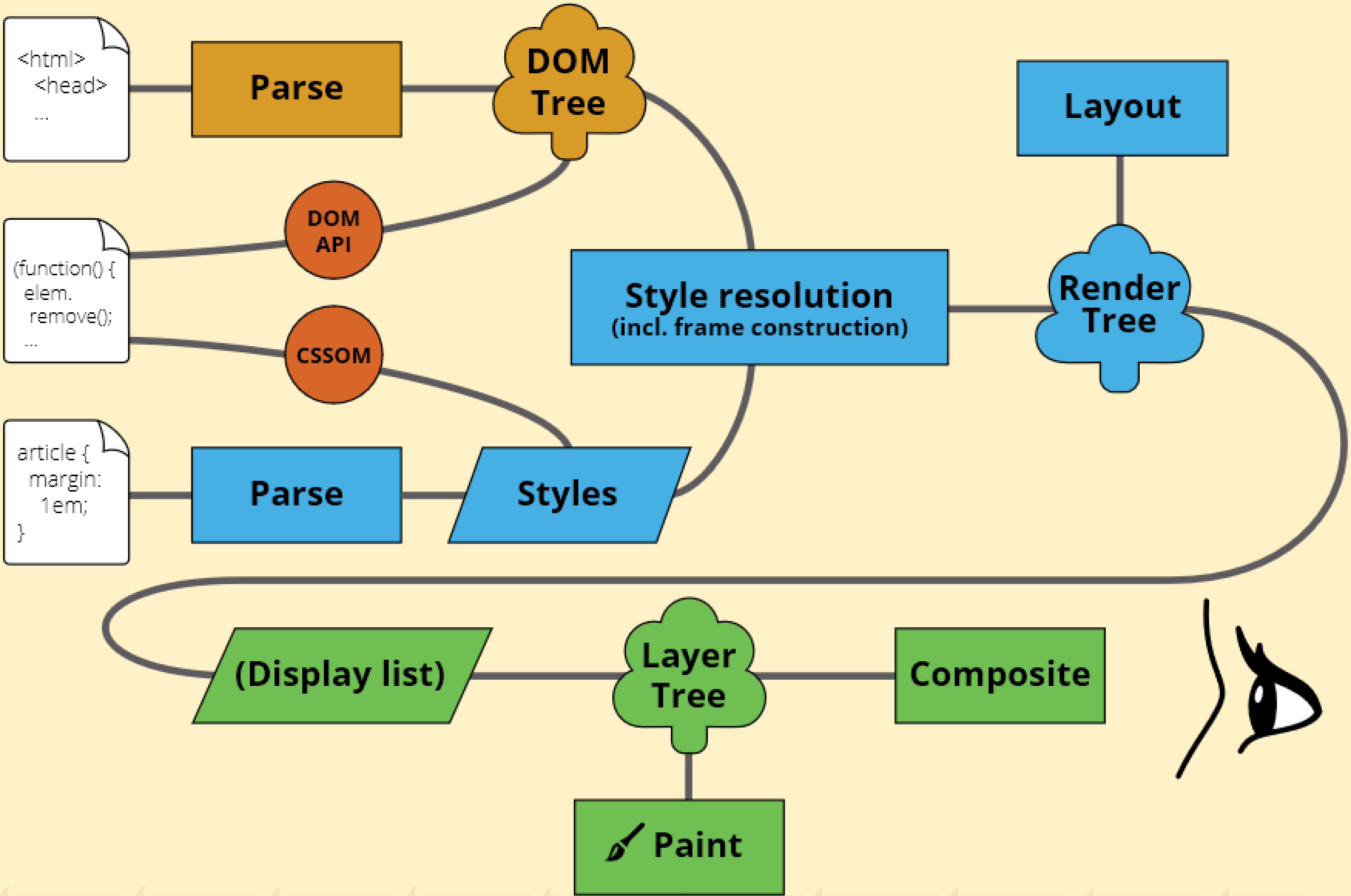
ANIMATION IS AWESOME… SOMETIMES

But when animation runs slowly or hesitates, that information is lost.

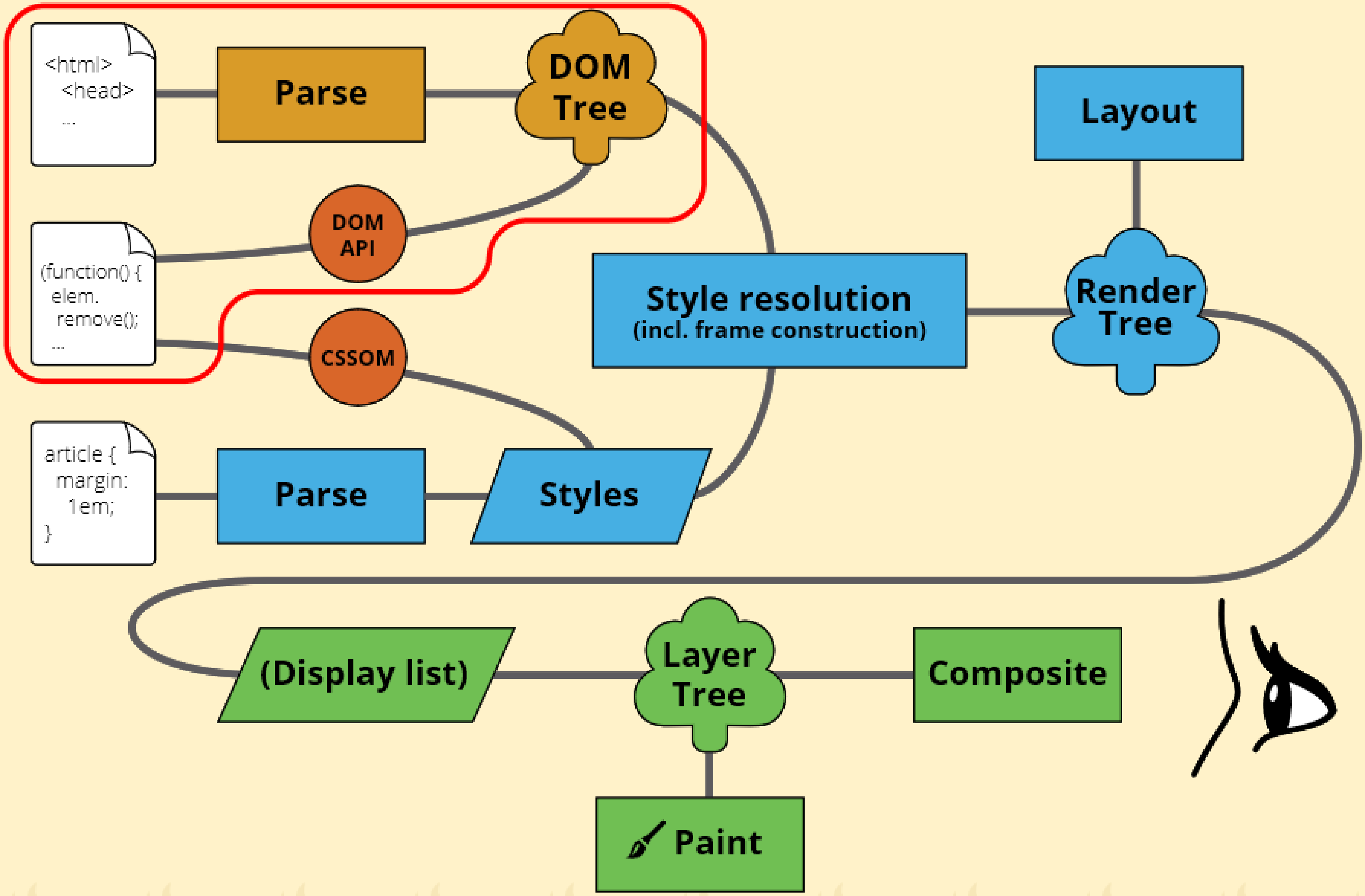
Hence for animation, performance is critical.

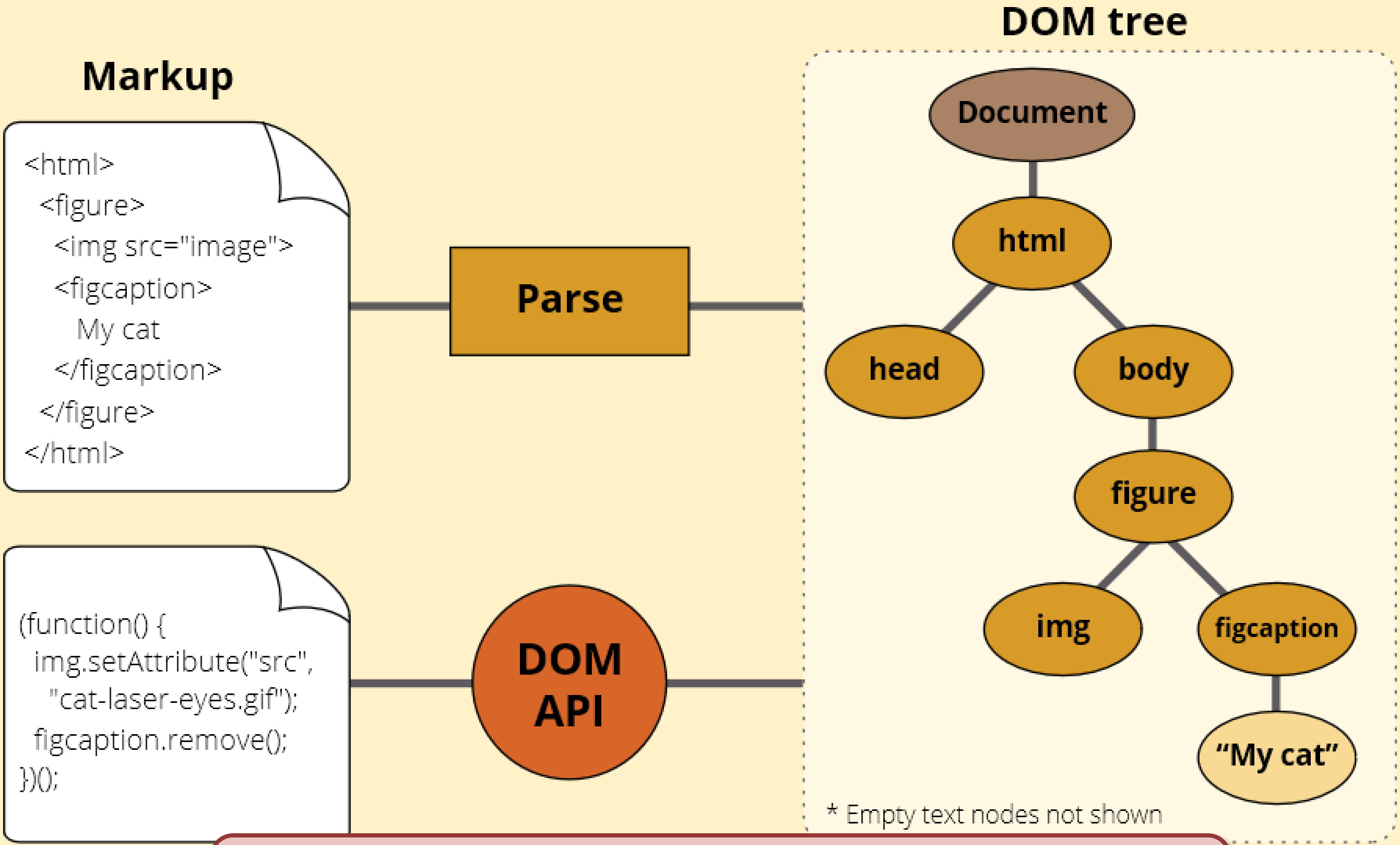
Scripted animation as displayed on Firefox and Chrome on Android HTC J

In order to fix animation performance in Web pages, we really need to understand how browsers work.



As we follow the journey from markup to our eyeballs, we will consider how we can make each step smoother or skip it all together.


<html>
<head>
...
Parse
DOM Tree
(function() {
elem.
remove();
...
DOM API
CSSOM
article {
margin:
1em;
}
Parse
Styles
Style resolution
(incl. frame construction)
Layout
Render Tree
(Display list)
Layer Tree
Composite
Paint



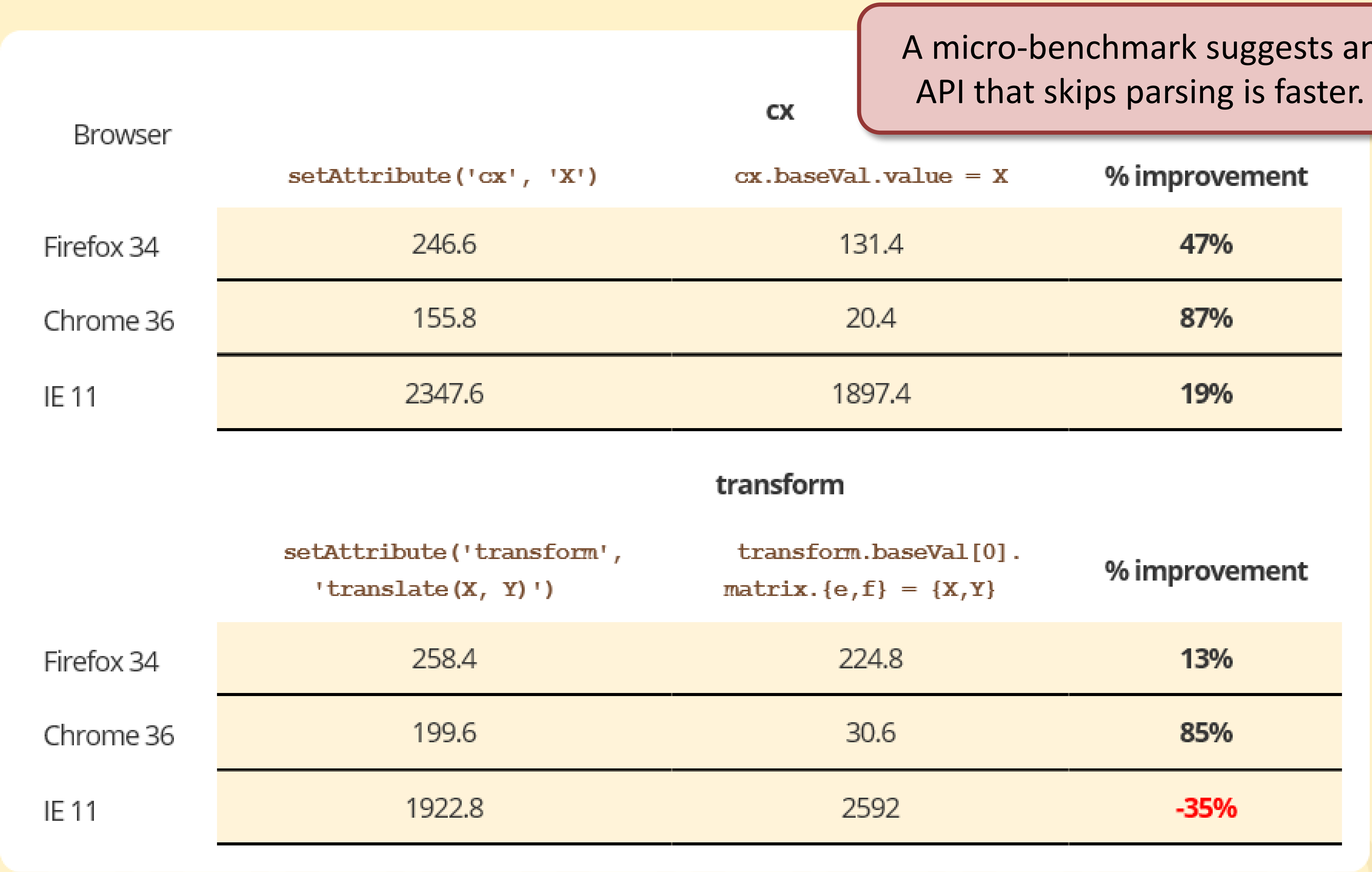
```
<html>
  <figure>
    <img src="image">
    <figcaption>
      My cat
    </figcaption>
  </figure>
</html>
```

```
(function() {
  img.setAttribute("src",
    "cat-laser-eyes.gif");
  figcaption.remove();
})();
```

Parsing can be slow. Most browsers do it in a separate thread.
If we skip parsing while animating surely it goes faster?

setAttribute VS SVG DOM

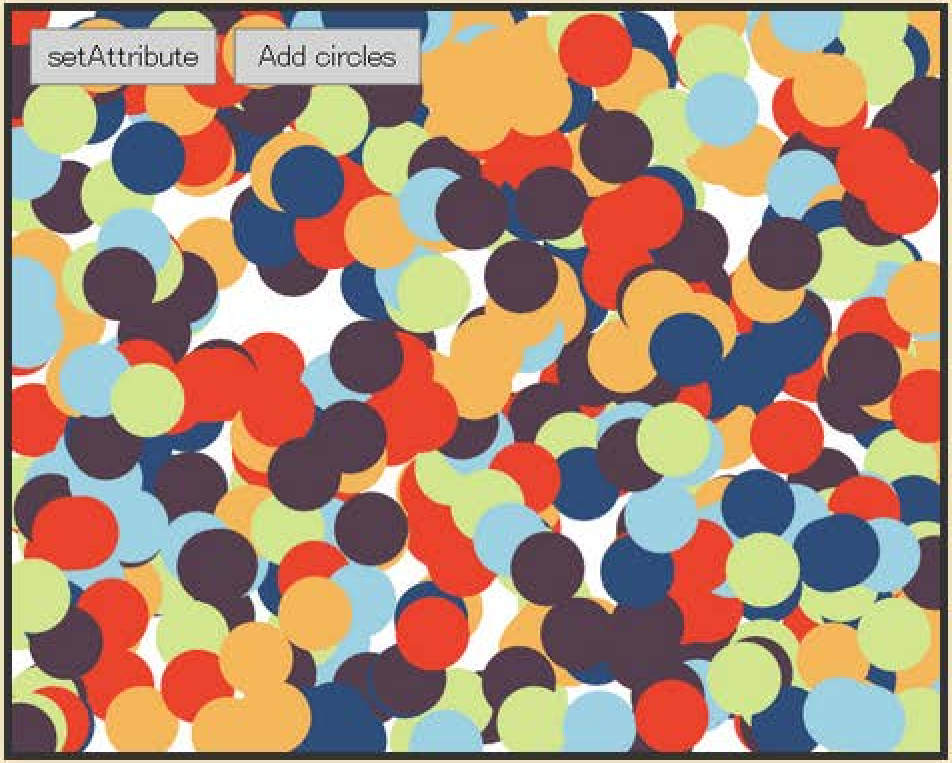
A micro-benchmark suggests an API that skips parsing is faster.



Browser		cx	
	`setAttribute('cx', 'X')`	`cx.baseVal.value = X`	% improvement
Firefox 34	246.6	131.4	**47%**
Chrome 36	155.8	20.4	**87%**
IE 11	2347.6	1897.4	**19%**

		transform	
	`setAttribute('transform', 'translate(X, Y)')`	`transform.baseVal[0].matrix.{e,f} = {X,Y}`	% improvement
Firefox 34	258.4	224.8	**13%**
Chrome 36	199.6	30.6	**85%**
IE 11	1922.8	2592	**-35%**

* Times are ms taken for 100,000 iterations averaged over 5 runs (lower numbers are faster)



More realistic test

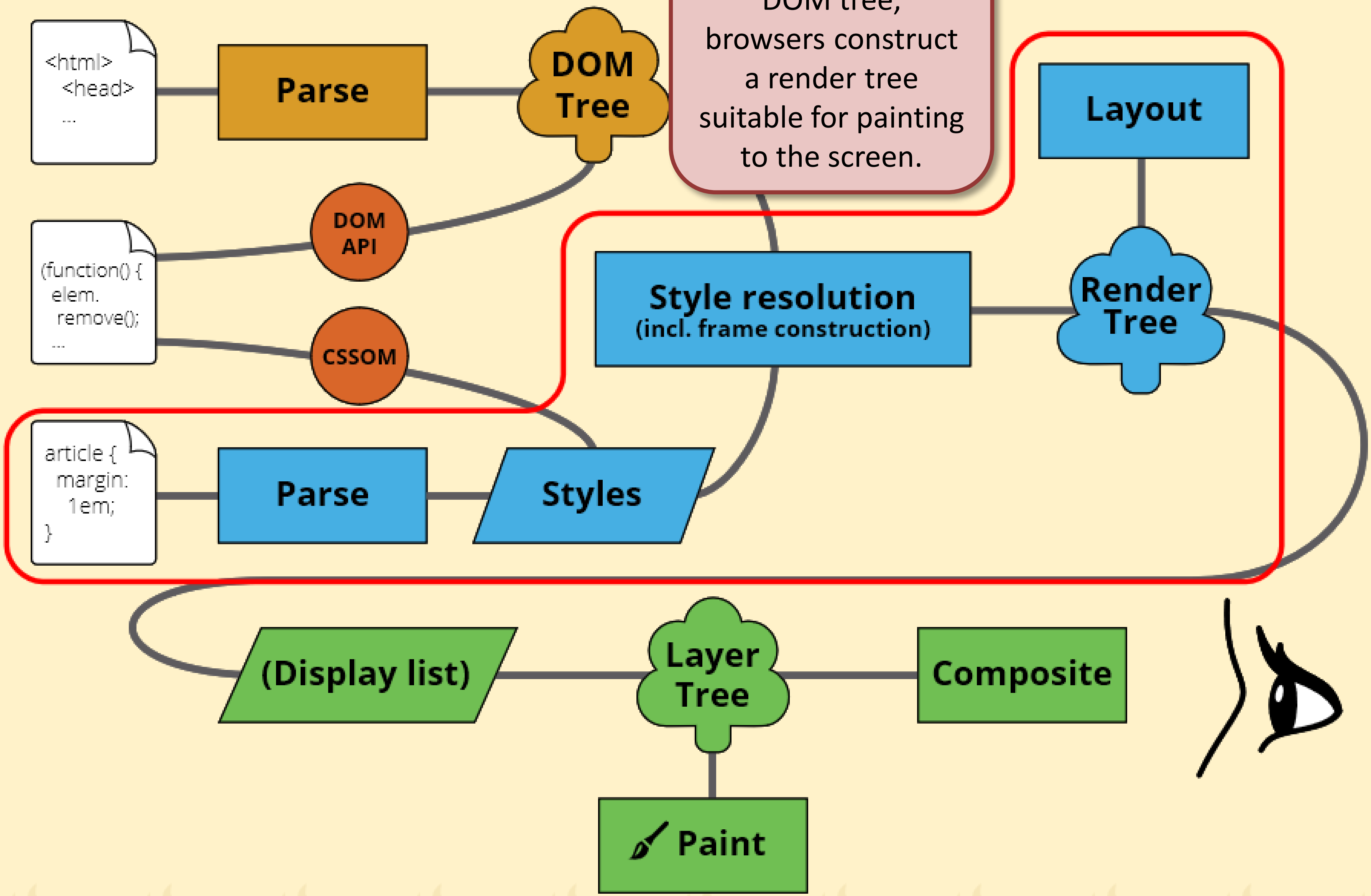
How about in a real-world animation?

It doesn’t make a lot of difference. Perhaps 3~4 fps at best.

More realistic test

Try it at home!

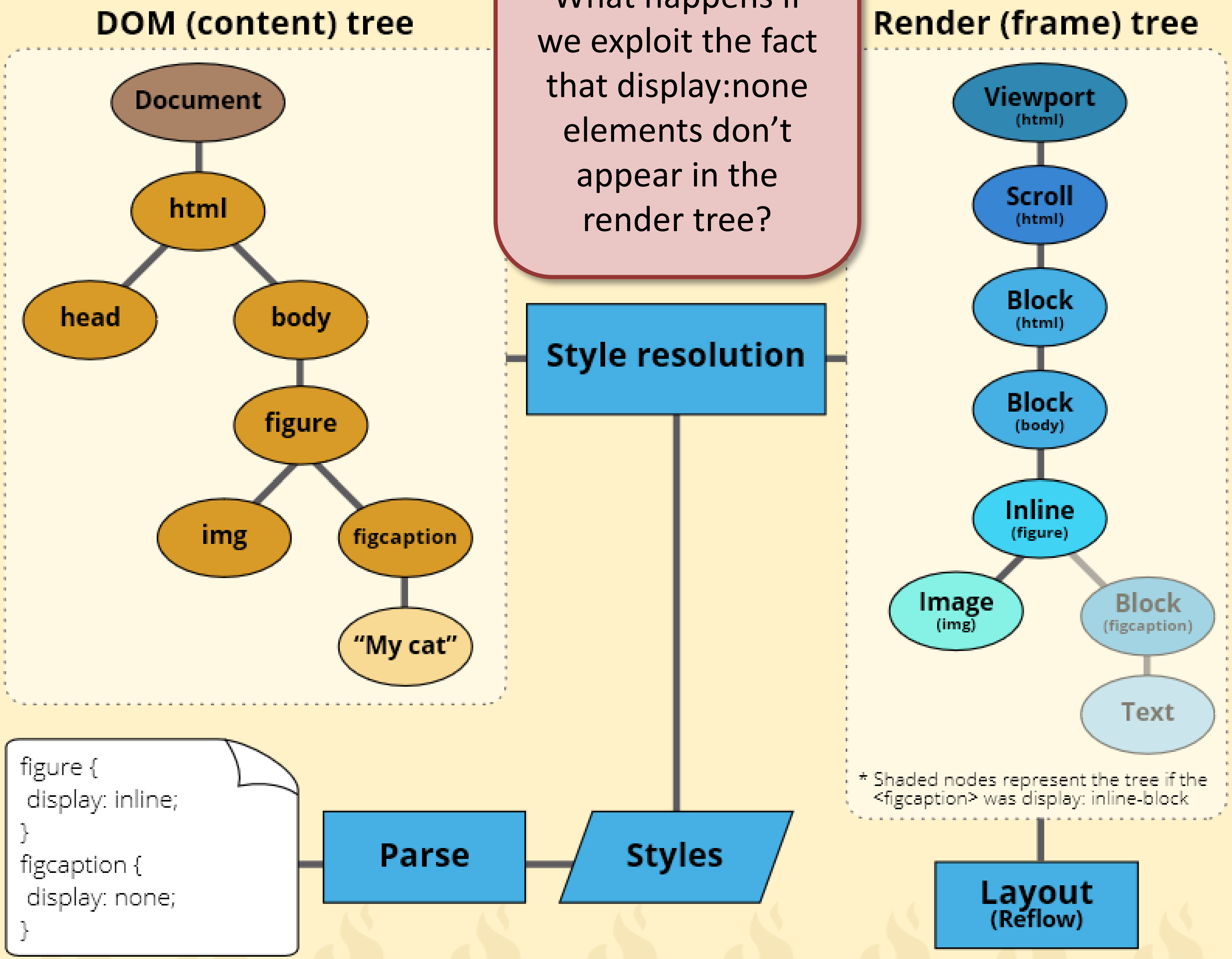
Try using a specialized API to avoid parsing



After creating a DOM tree, browsers construct a render tree suitable for painting to the screen.

There are bigger performance gains to be had from the style system.

What happens if we exploit the fact that display:none elements don't appear in the render tree?



DOM (content) tree

Document

html

head

body

figure

img

figcaption

"My cat"

Style resolution

Render (frame) tree

Viewport (html)

Scroll (html)

Block (html)

Block (body)

Inline (figure)

Image (img)

Block (figcaption)

Text

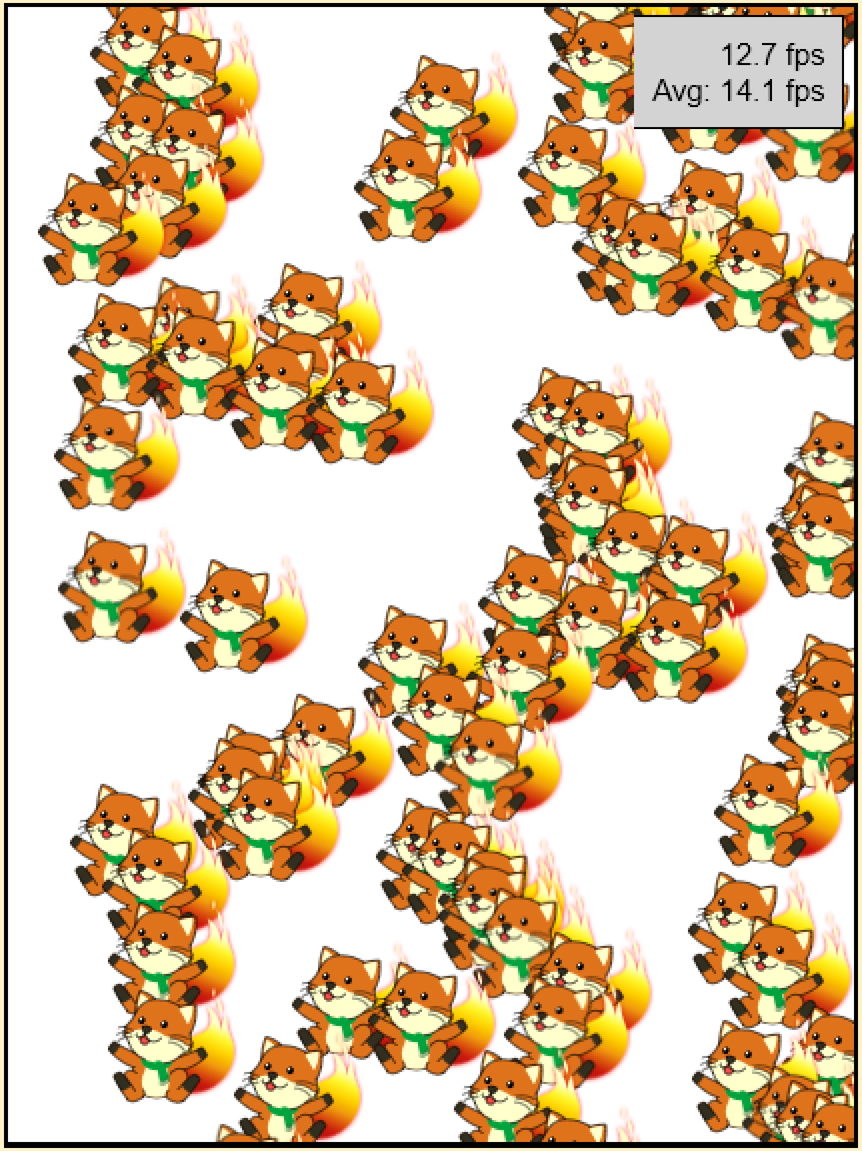
* Shaded nodes represent the tree if the <figcaption> was display: inline-block

```
figure {
 display: inline;
}
figcaption {
 display: none;
}
```

Parse

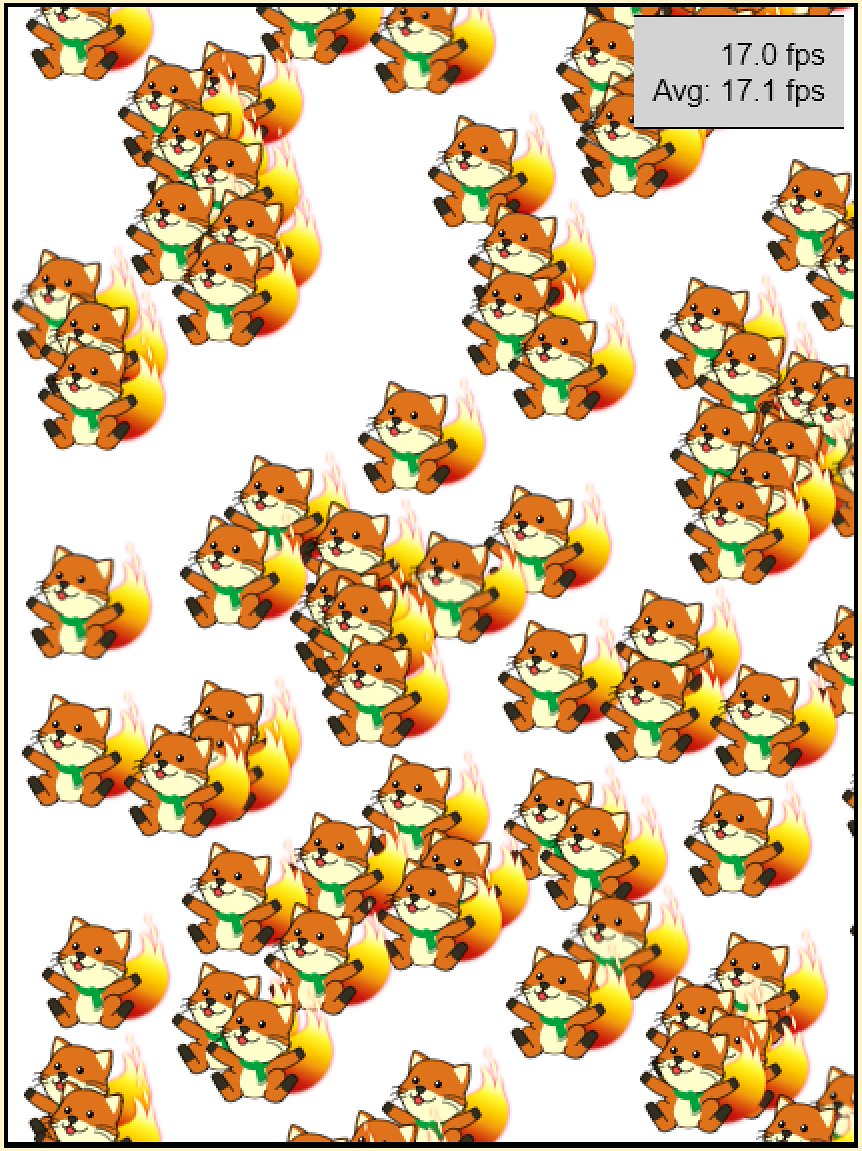
Styles

Layout (Reflow)



12.7 fps
Avg: 14.1 fps

Unoptimized



17.0 fps
Avg: 17.1 fps

Using display:none

USING display:none

Browser	Unoptimized	With display:none	Avg. improvement
Firefox 34	25.6fps	29fps	**3.4fps / 13%**
Chrome 36	5.1fps	12.5fps	**7.4fps / 145%**
IE 11	4.8fps	7.8fps	**3.0fps / 63%**

* Average result after 3~5 runs. Higher numbers are better.

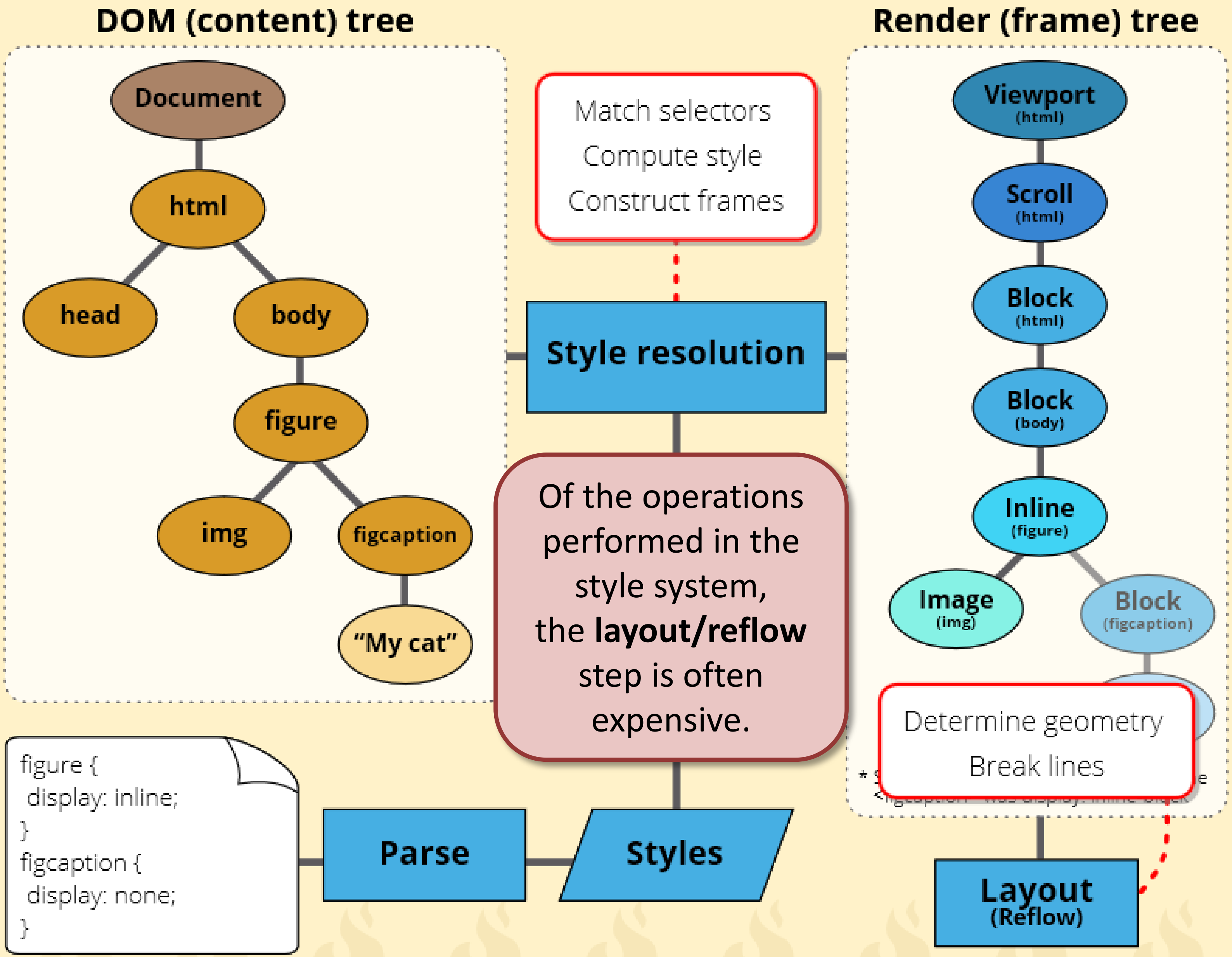
Try it at home!

Remove elements from the render tree with display:none

(Firefox doesn’t show such a big difference in this case since the test case animates ‘top’ which, as we’ll see, does not trigger reflow in Firefox so setting display:none doesn’t have as big an impact.)

parapara.mozlabs.jp

This technique improved performance for the Parapara animation project where characters are set to display:none when they are off-stage.



DOM (content) tree

Document

html

head

body

figure

img

figcaption

“My cat”

Render (frame) tree

Viewport
(html)

Scroll
(html)

Block
(html)

Block
(body)

Inline
(figure)

Image
(img)

Block
(figcaption)

Match selectors
Compute style
Construct frames

Style resolution

Of the operations performed in the style system, the **layout/reflow** step is often expensive.

```
figure {
 display: inline;
}
figcaption {
 display: none;
}
```

Parse

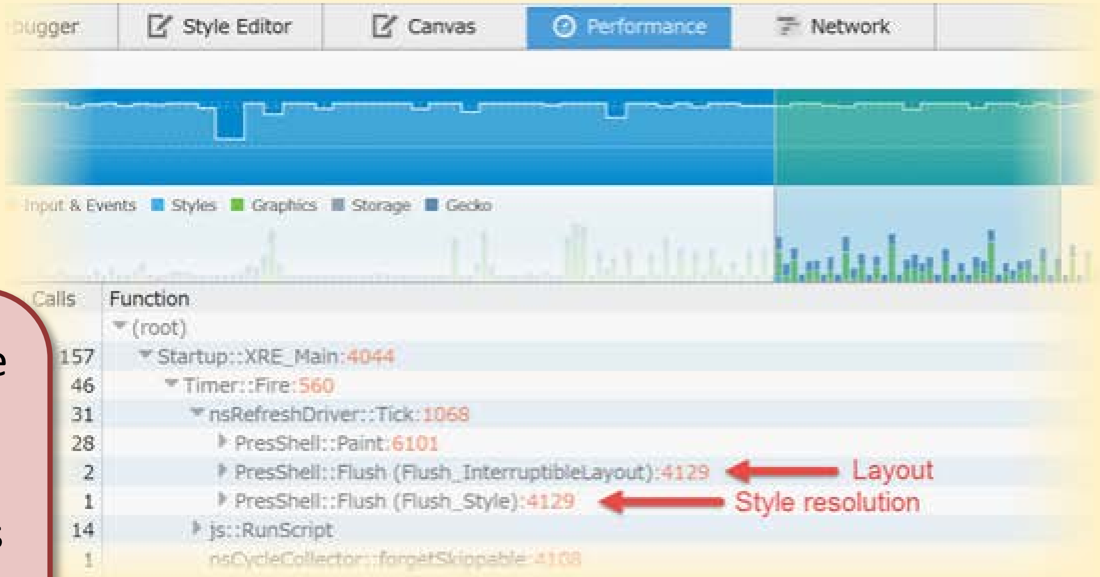
Styles

Determine geometry
Break lines

* S e
<figcaption> was display: inline-block

Layout
(Reflow)

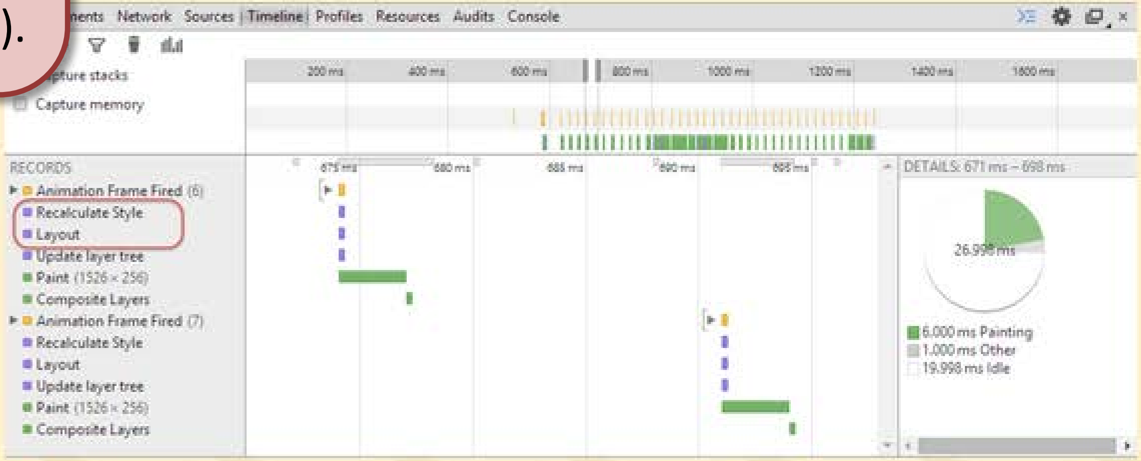
We can measure style resolution and layout time in profiling tools in Firefox (above) and Chrome (below).



```
Calls  Function
       (root)
157      Startup::XRE_Main:4044
 46        Timer::Fire:560
 31          nsRefreshDriver::Tick:1068
 28            PresShell::Paint:6101
  2            PresShell::Flush (Flush_InterruptibleLayout):4129
  1            PresShell::Flush (Flush_Style):4129
 14          js::RunScript
  1          nsCycleCollector::forgetSkippable:4108
```

Layout

Style resolution



pture stacks

□ Capture memory

RECORDS

- Animation Frame Fired (6)
- Recalculate Style
- Layout
- Update layer tree
- Paint (1526 × 256)
- Composite Layers
- Animation Frame Fired (7)
- Recalculate Style
- Layout
- Update layer tree
- Paint (1526 × 256)
- Composite Layers

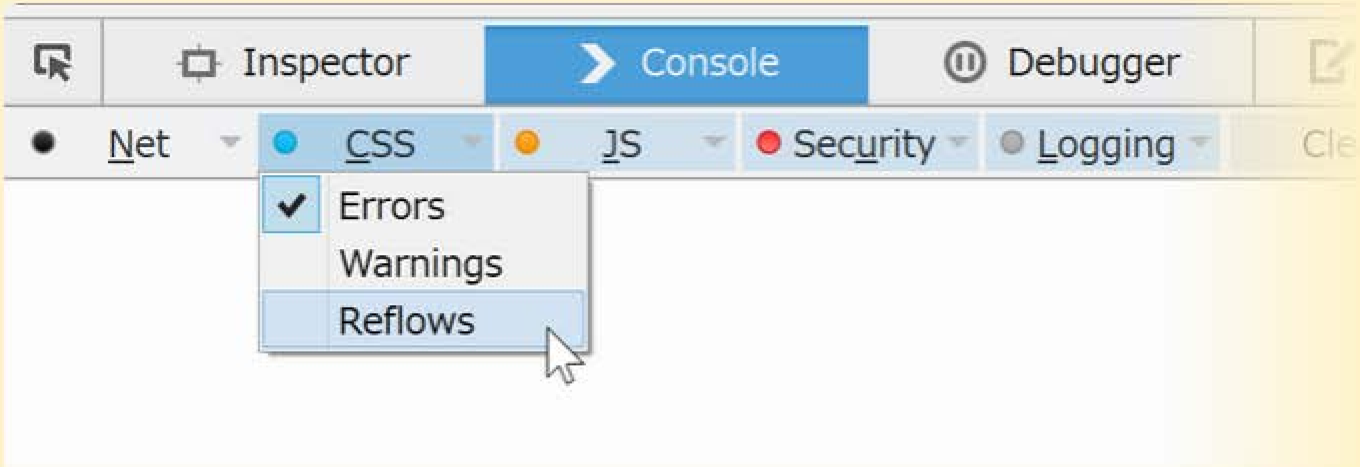
DETAILS: 671 ms – 698 ms

🟩 6.000 ms Painting

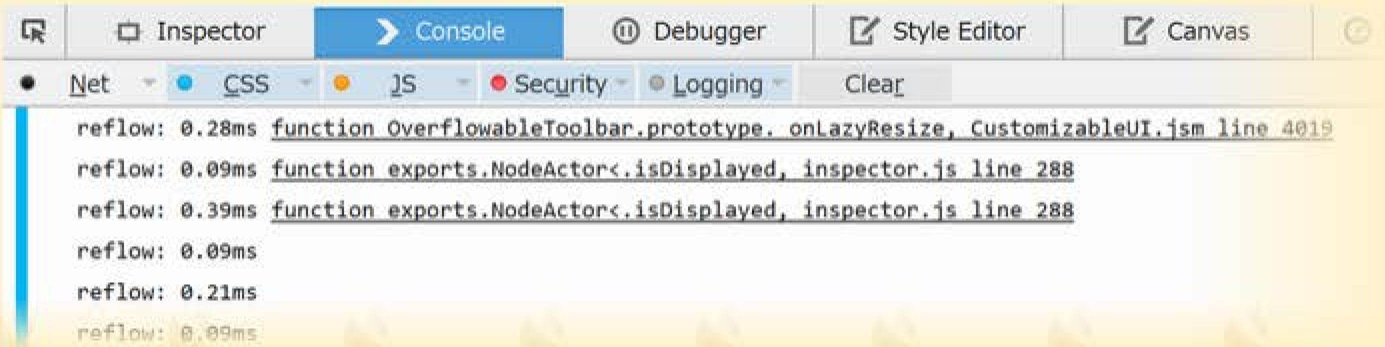
▨ 1.000 ms Other

□ 19.998 ms Idle

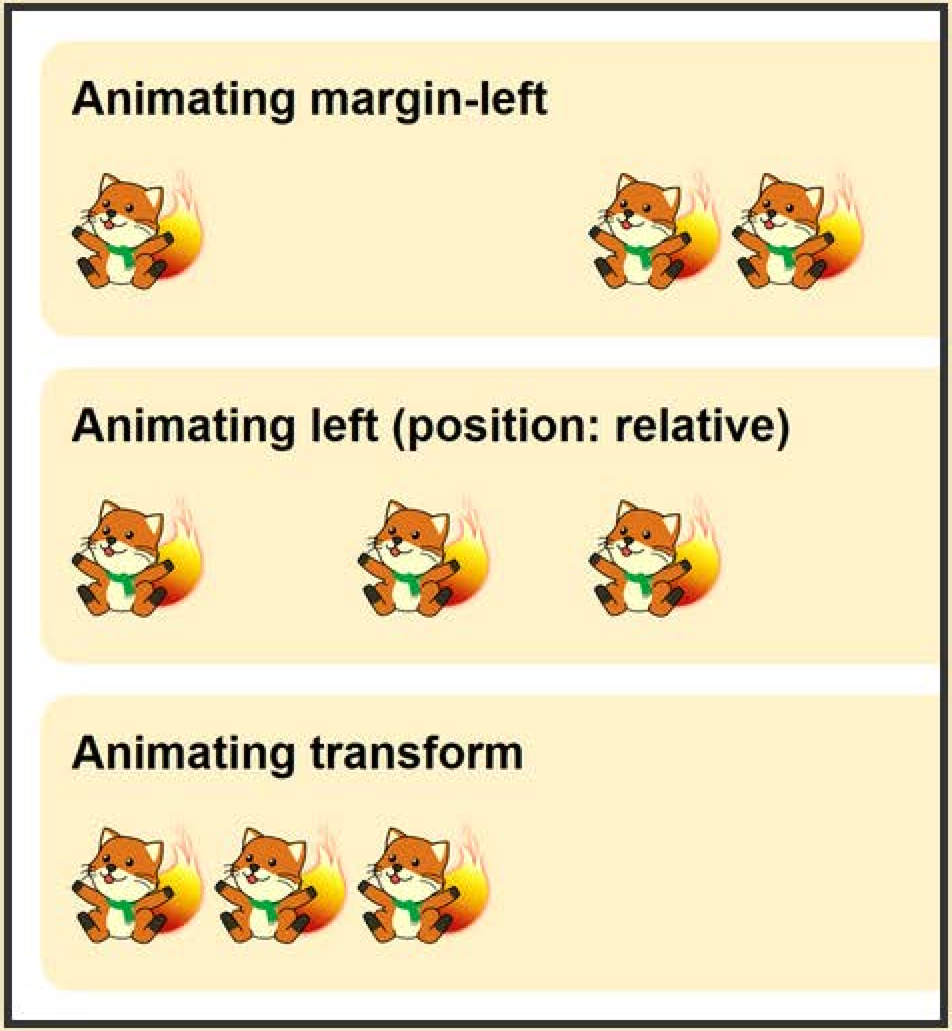
REFLOW LOGGING IN FIREFOX



Firefox lets you inspect reflow (layout) in the console.



```
reflow: 0.28ms function OverflowableToolbar.prototype._onLazyResize, CustomizableUI.jsm line 4019
reflow: 0.09ms function exports.NodeActor<.isDisplayed, inspector.js line 288
reflow: 0.39ms function exports.NodeActor<.isDisplayed, inspector.js line 288
reflow: 0.09ms
reflow: 0.21ms
reflow: 0.09ms
```

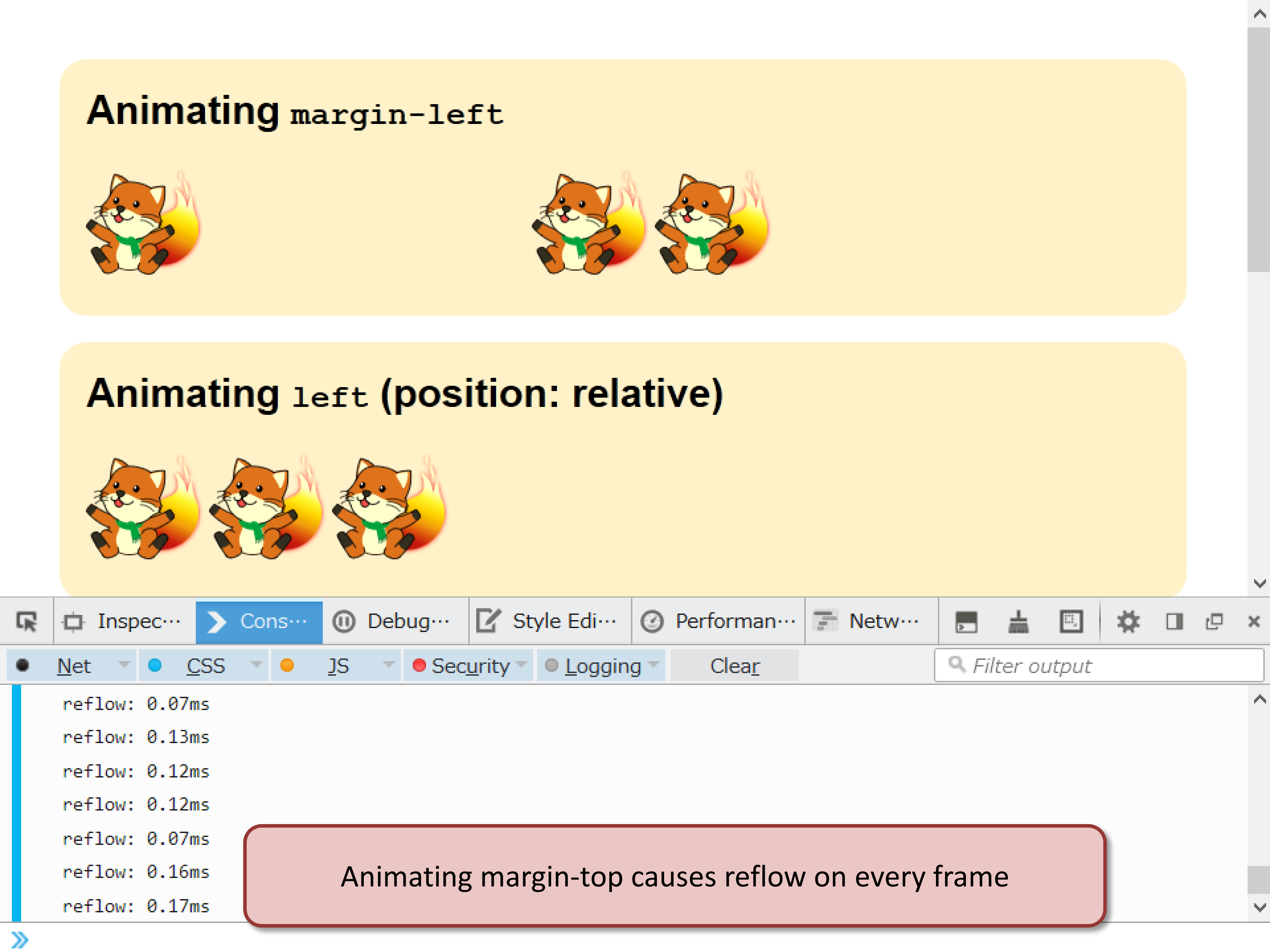


Animating margin-left

Animating left (position: relative)

Animating transform

Let’s see how different animations affect layout

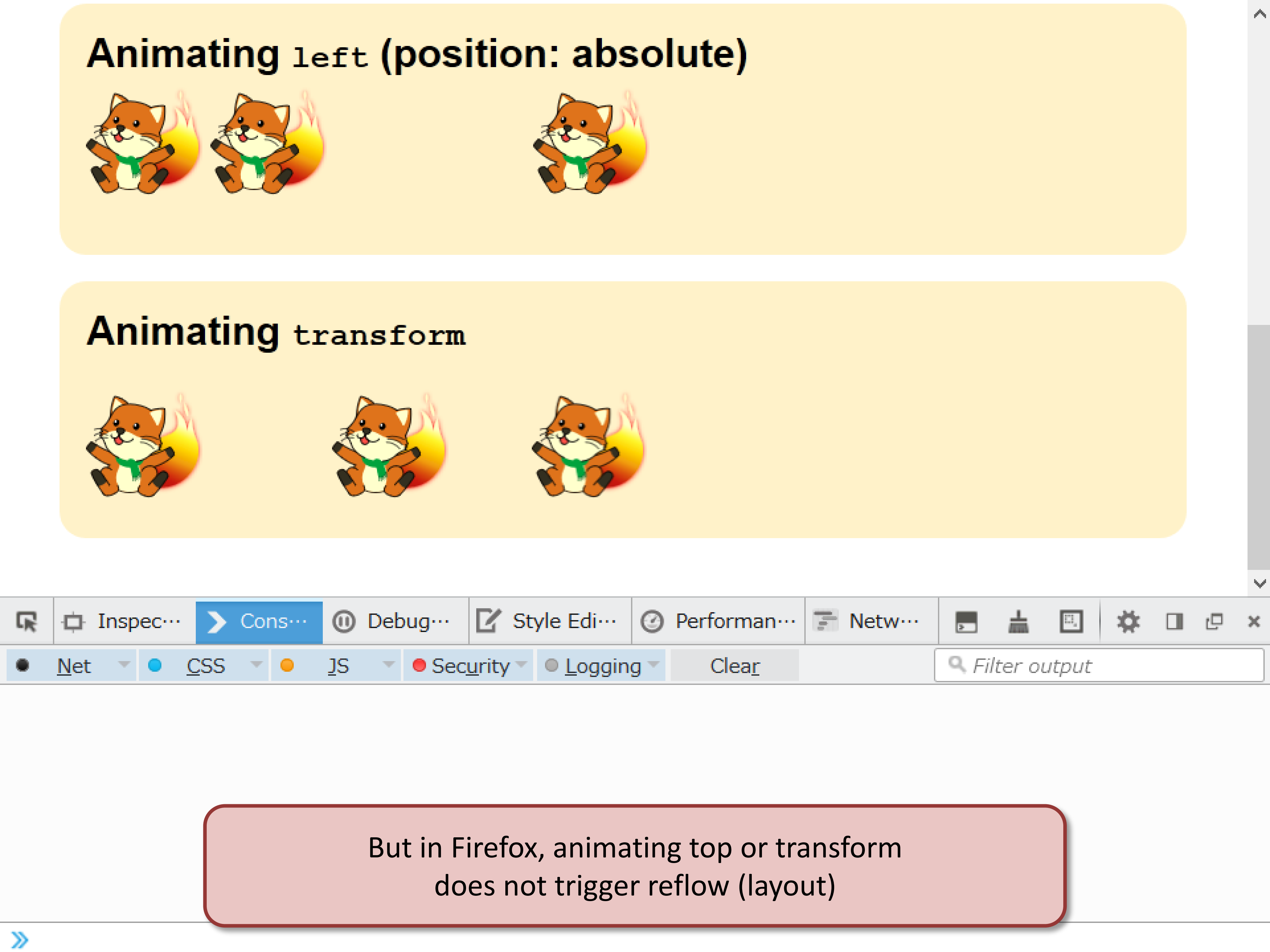


Animating `margin-left`

Animating `left` (position: relative)

```
reflow: 0.07ms
reflow: 0.13ms
reflow: 0.12ms
reflow: 0.12ms
reflow: 0.07ms
reflow: 0.16ms
reflow: 0.17ms
```

Animating margin-top causes reflow on every frame



Animating `left` (position: absolute)

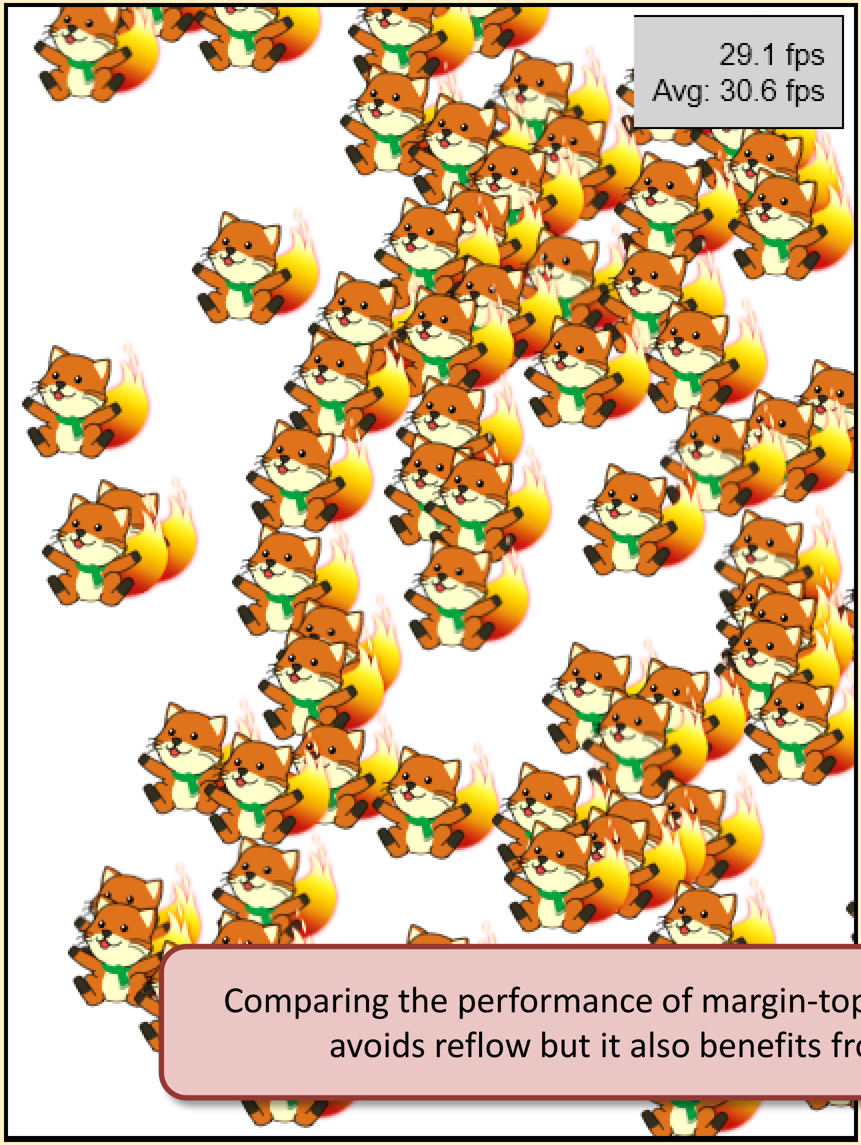
Animating `transform`

But in Firefox, animating top or transform
does not trigger reflow (layout)

TRIGGERING LAYOUT

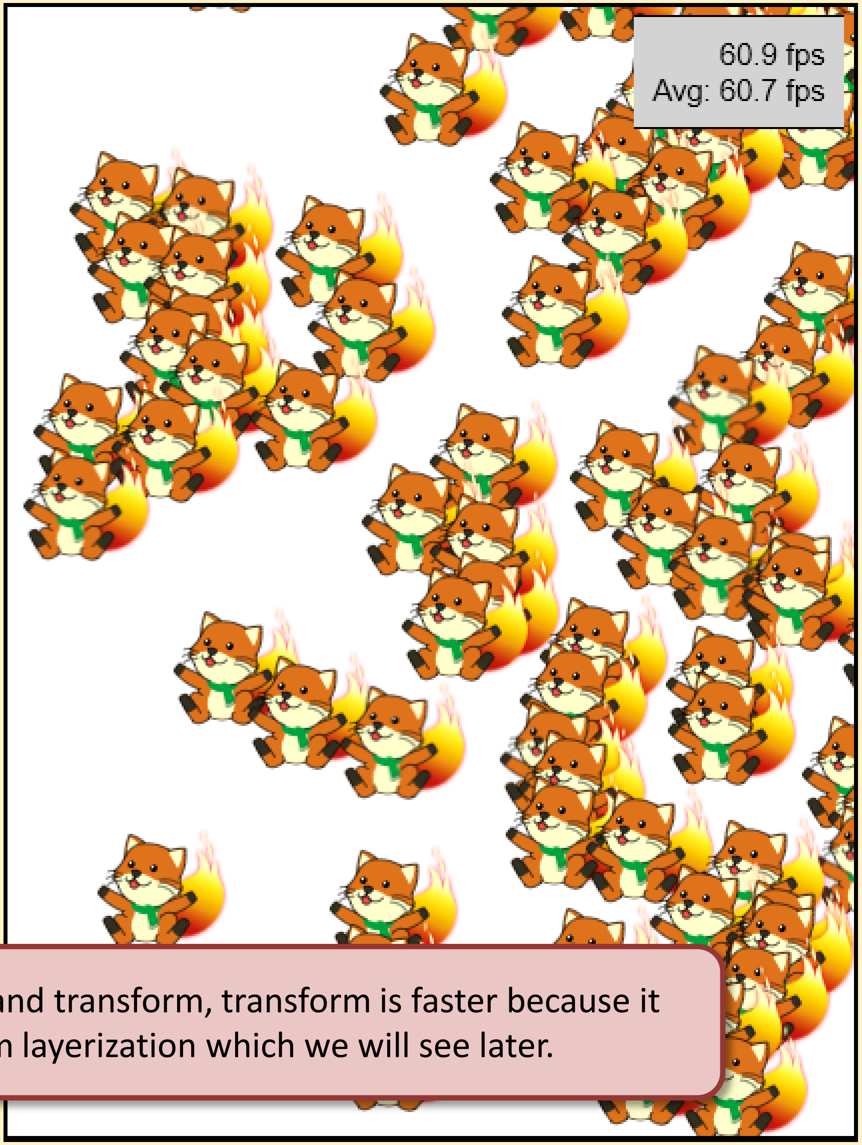
	Triggers layout?*		
property	Firefox 34	Chrome 36	IE 11
`margin-left`	●	●	●
`left` (position: relative)		●	●
`left` (position: absolute)		●	●
`transform`			

* Based on my inspection of profiles from this test case.



29.1 fps
Avg: 30.6 fps

`margin-top`



60.9 fps
Avg: 60.7 fps

`transform`

Comparing the performance of margin-top and transform, transform is faster because it avoids reflow but it also benefits from layerization which we will see later.

AVOIDING REFLOW

Try it at home!

Try `transform` instead of `top` / `left` / `marginTop` etc.

Try it at home!

Try animating elements that are `position:absolute`.

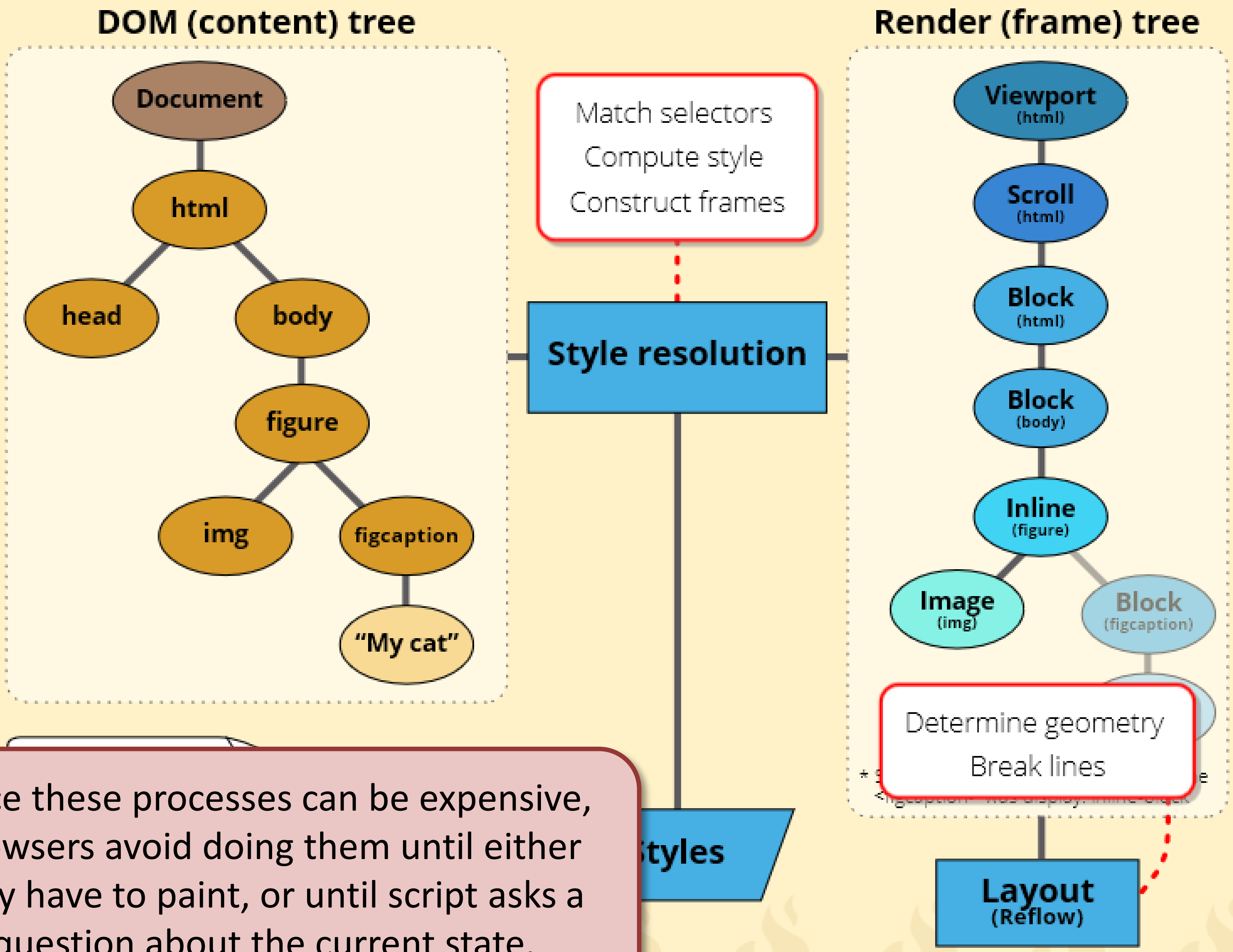
Try it at home!

Try non-geometric properties like `color`, `opacity` etc.

Try it at home!

Try `transform: scale` instead of `font-size`.

LET SLEEPING DOGS LIE



Since these processes can be expensive, browsers avoid doing them until either they have to paint, or until script asks a question about the current state.

WHAT TRIGGERS RECALC / REFLOW?

- `window.getComputedStyle(elem).color`

  → style recalc (typically)

- `window.getComputedStyle(elem).width`, `elem.offsetTop`, `elem.getClientRects()` etc.

  → reflow

DON'T DO THIS

```
for (var i = 1; i < containerElem.children.length; i++) {
  containerElem.children[i].style.top =
    containerElem.children[i-1].offsetTop + 10 + "px";
}
```

AVOIDING FORCING REFLOW

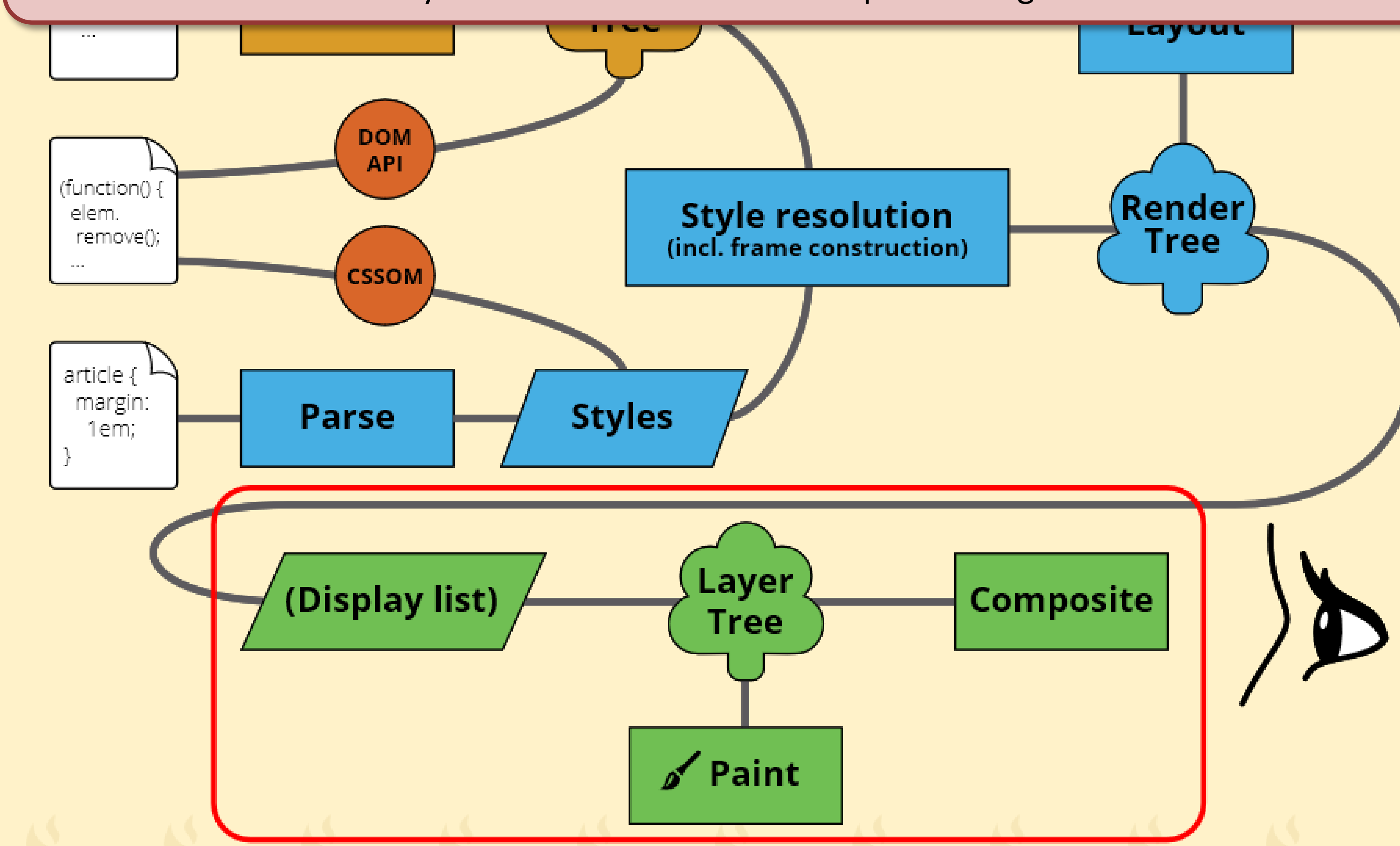
Browser	Triggering reflow	Not doing that	Avg. improvement
Firefox 34	42.1fps*	45.8fps*	**3.7fps / 9%**
Chrome 36	10.5fps	23.2fps	**12.7fps / 120%**
IE 11	8.2fps	19.1fps	**10.9fps / 132%**

* Average result after 3~5 runs of test A and test B. Results for Firefox were particularly variable but were generally only slightly faster since the test animates the `top` property which does not trigger reflow in Firefox.

Try it at home!

Try reading computed style (especially geometry) less often or not at all.

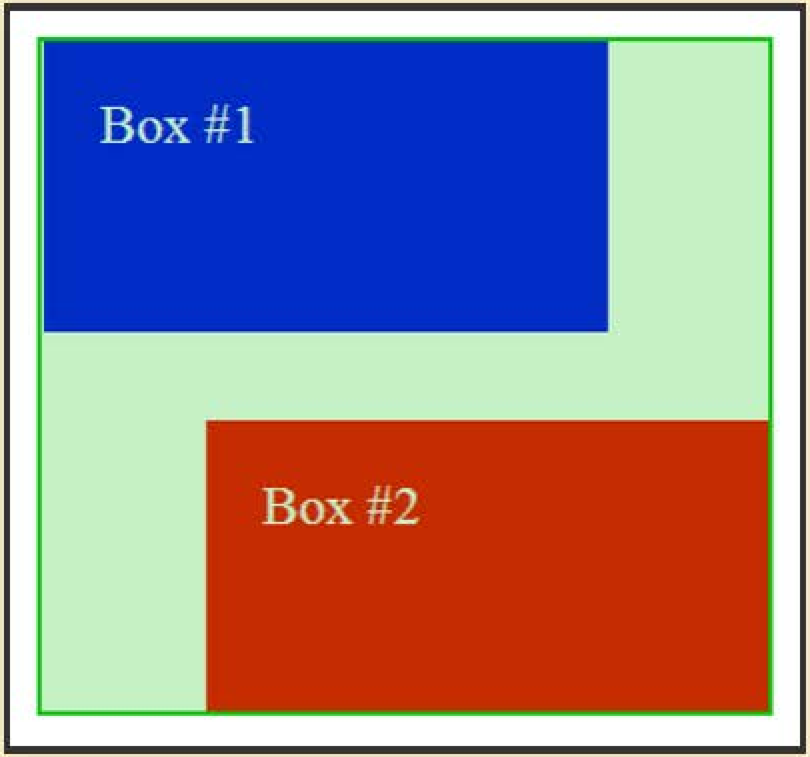
Painting is often the most expensive part. Firefox creates a display list of items to paint, then creates a layer tree into which it paints. The layers in the tree are then composited together.



(function() {
 elem.
 remove();
 ...

article {
 margin:
 1em;
}

PAINT COST

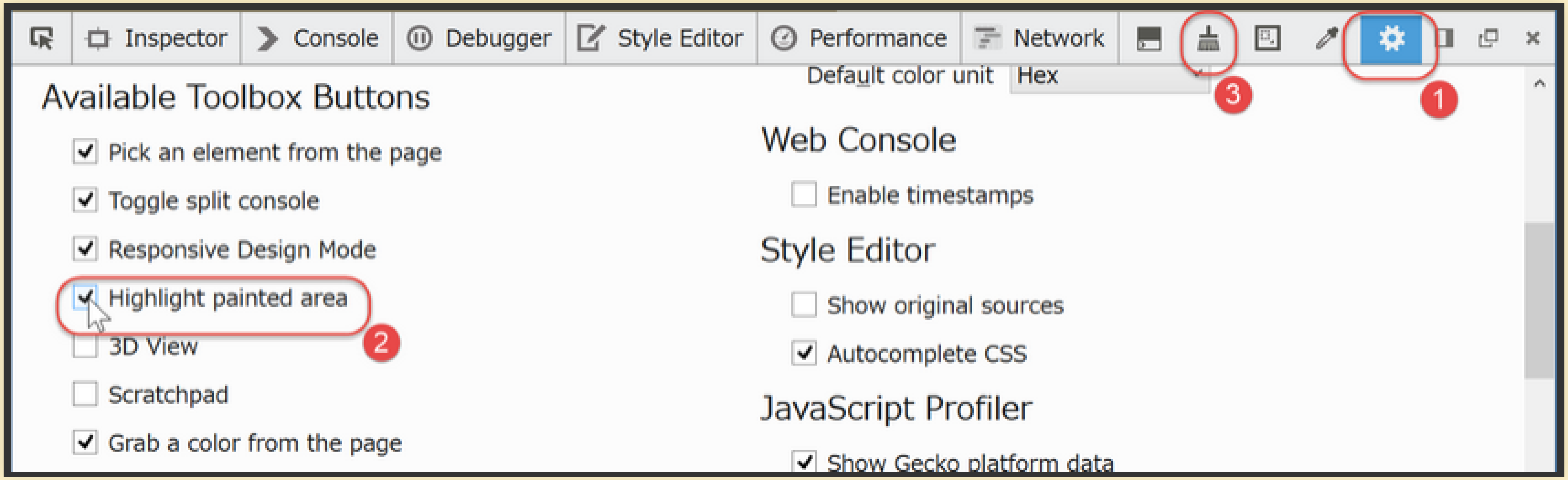


Paint area

×

Paint complexity

PAINT FLASHING (FIREFOX)



Available Toolbox Buttons

- ☑ Pick an element from the page
- ☑ Toggle split console
- ☑ Responsive Design Mode
- ☑ Highlight painted area
- □ 3D View
- □ Scratchpad
- ☑ Grab a color from the page

Default color unit Hex

Web Console

- □ Enable timestamps

Style Editor

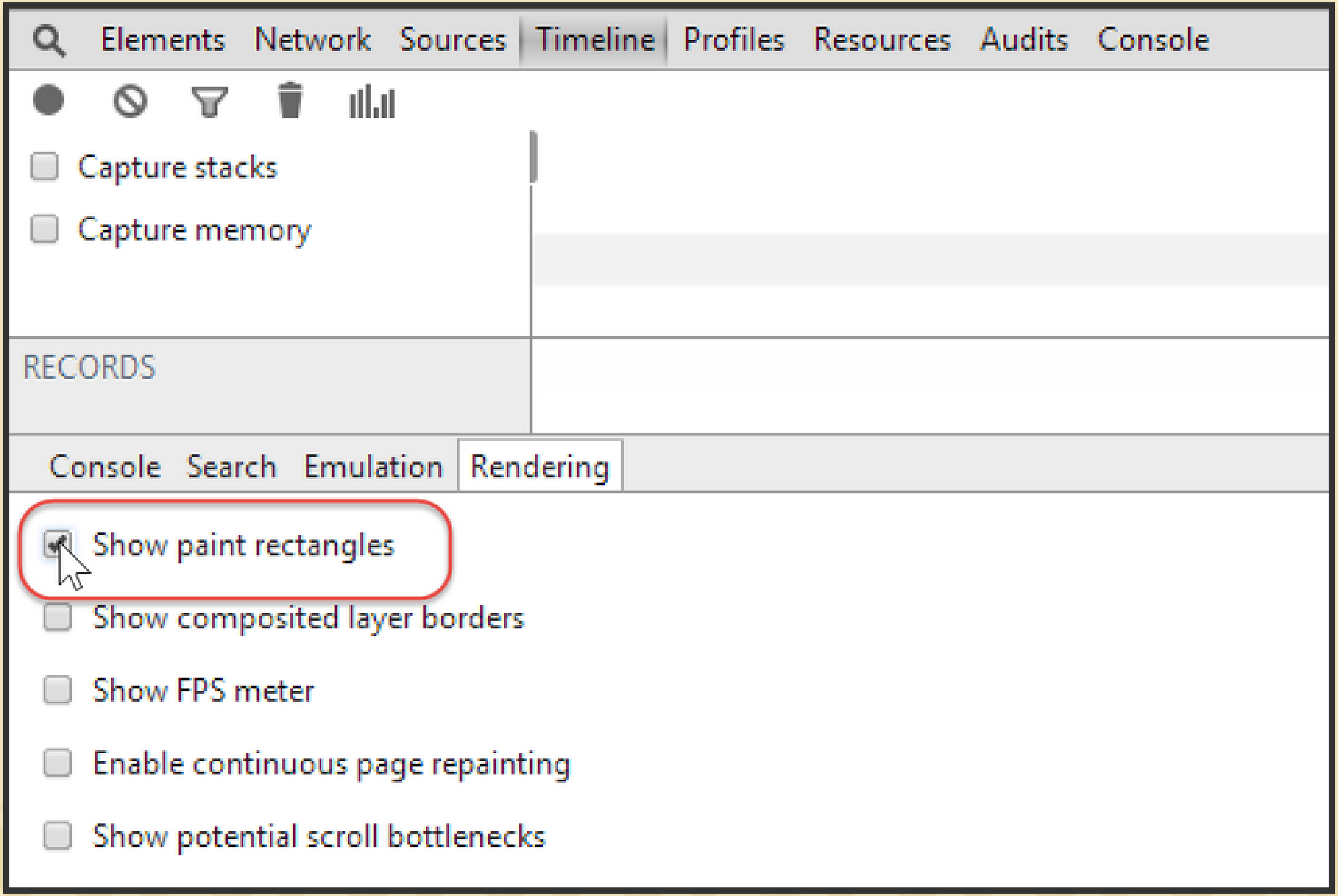
- □ Show original sources
- ☑ Autocomplete CSS

JavaScript Profiler

- ☑ Show Gecko platform data

We can see exactly what area is being painted

PAINT RECTANGLES (CHROME)



□ Capture stacks

□ Capture memory

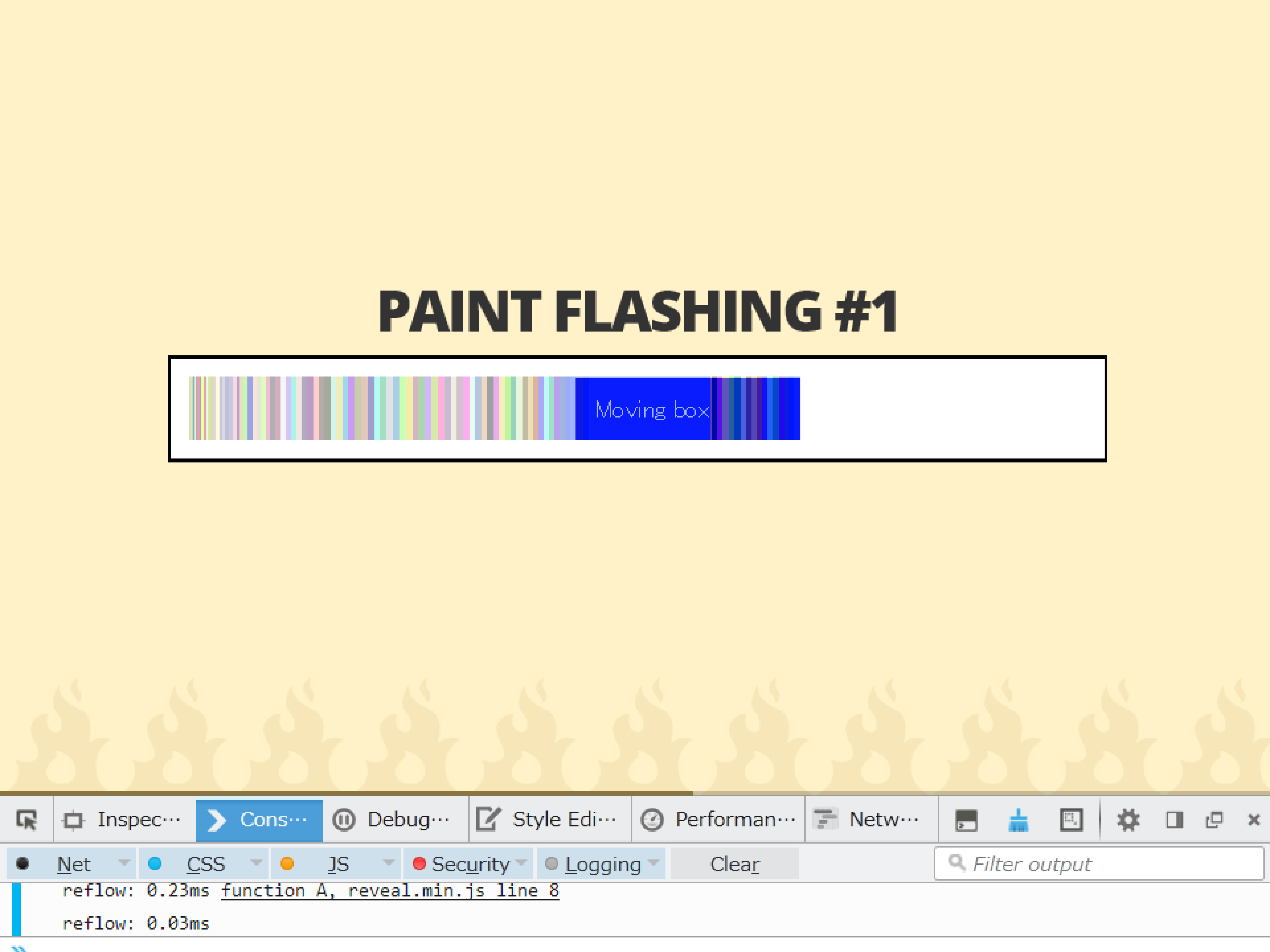
☑ Show paint rectangles

□ Show composited layer borders

□ Show FPS meter

□ Enable continuous page repainting

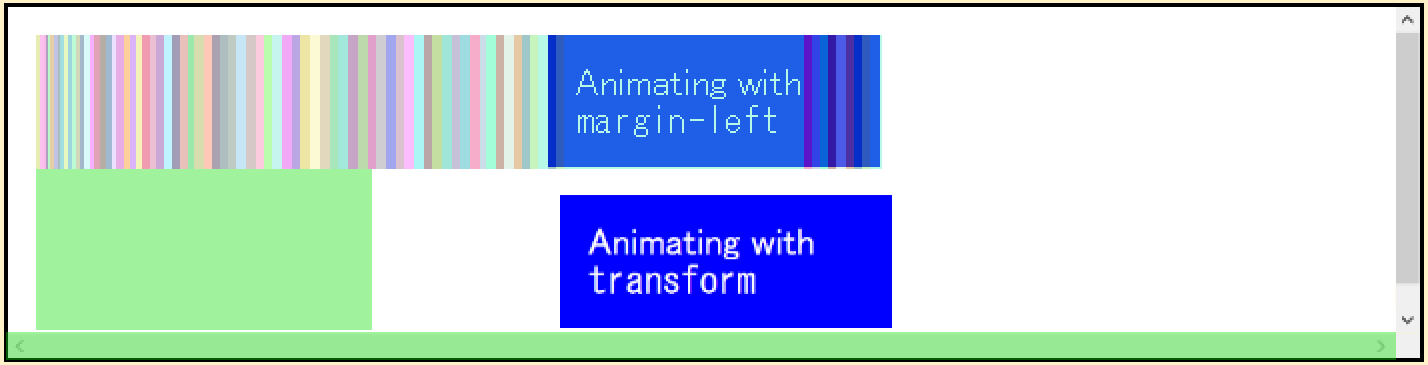
□ Show potential scroll bottlenecks



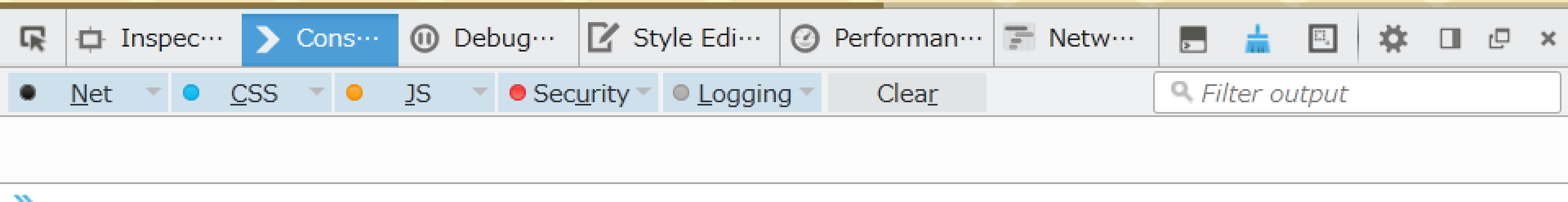
PAINT FLASHING #1

```
reflow: 0.23ms function A, reveal.min.js line 8
reflow: 0.03ms
```

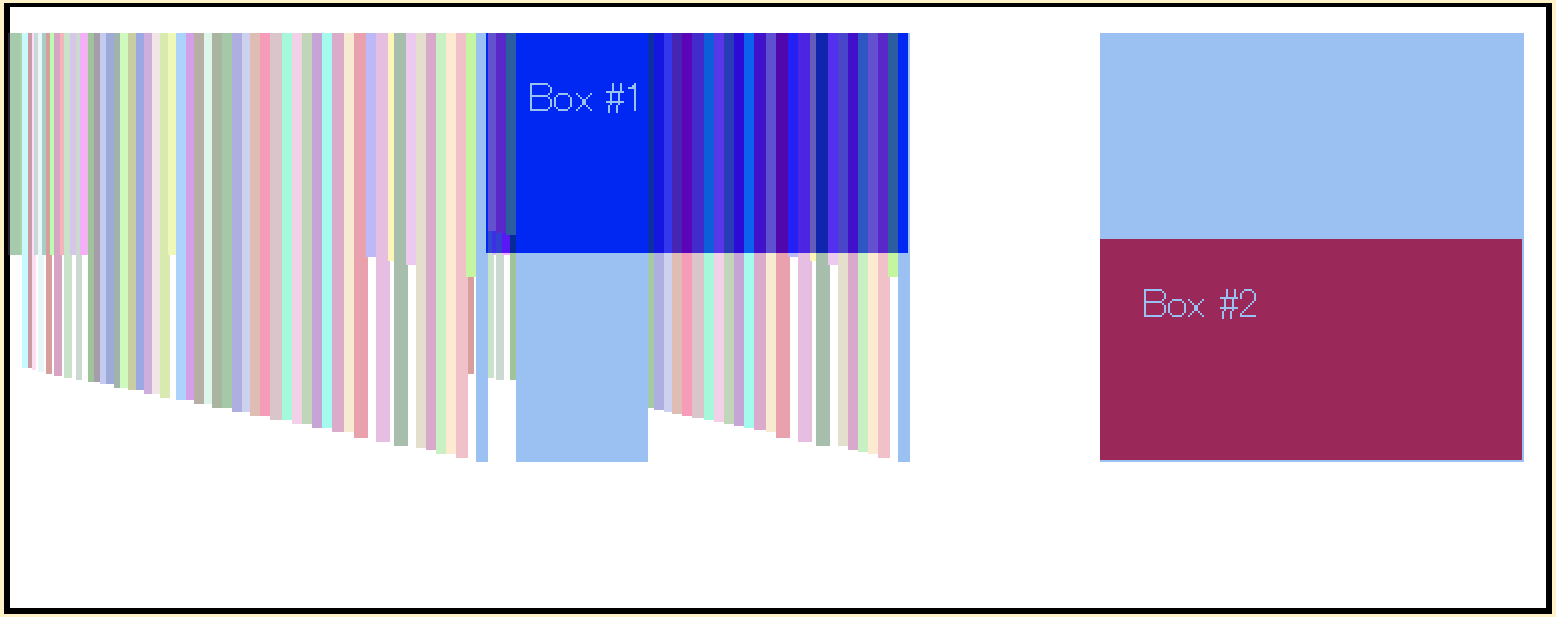
PAINT FLASHING #2



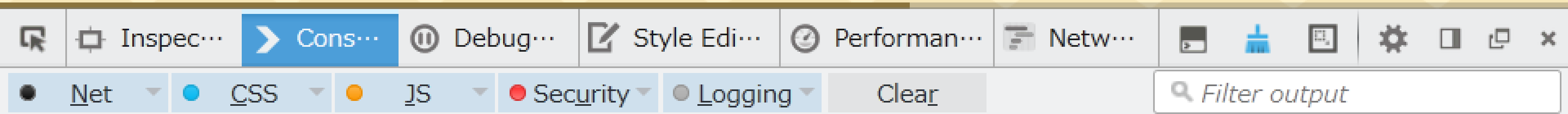
When animating transform we only paint once at the start.
This is because it gets put in its own layer.



PAINT FLASHING #3



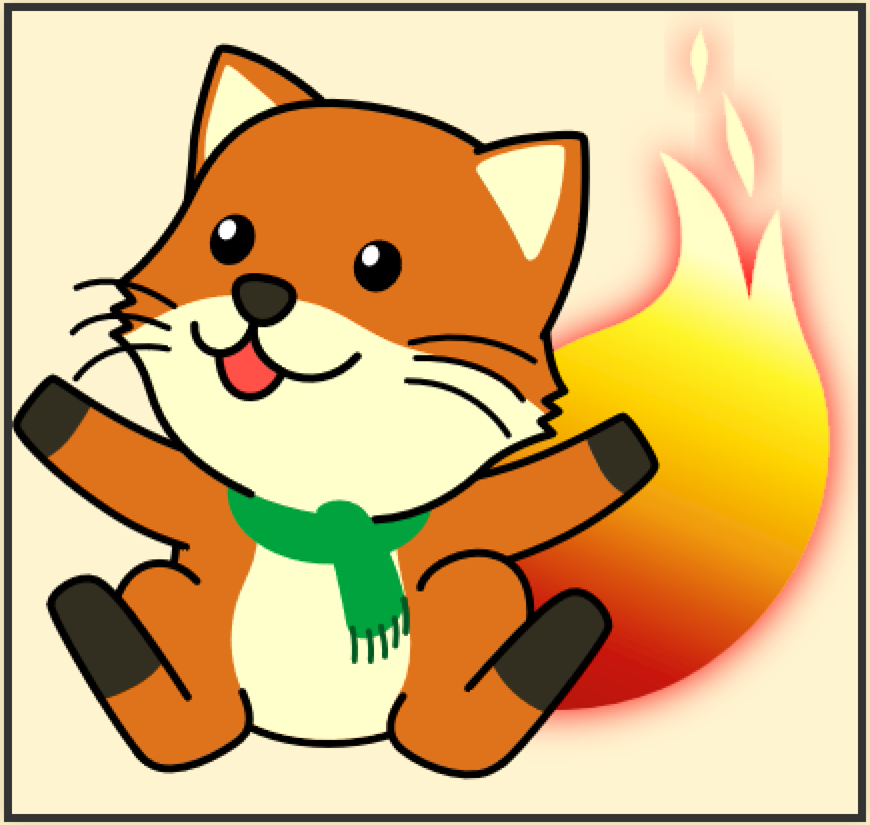
When animating independent areas Chrome seems to paint the union of dirty areas so layerization can be more important there.



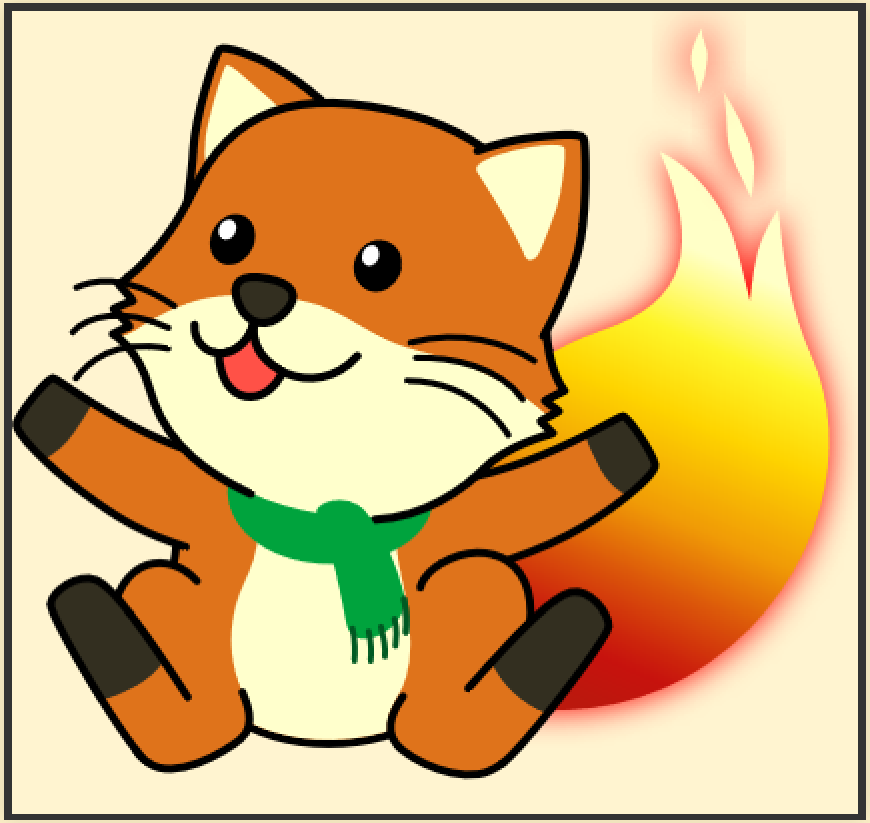
PAINT COMPLEXITY

- `box-shadow`
- `border-radius`
- SVG filters…

However, SVG filters are often hardware accelerated.
Sometimes the combination of features is what is slow.



Animating `stdDeviation` on `<feGaussianBlur>` → **33fps**

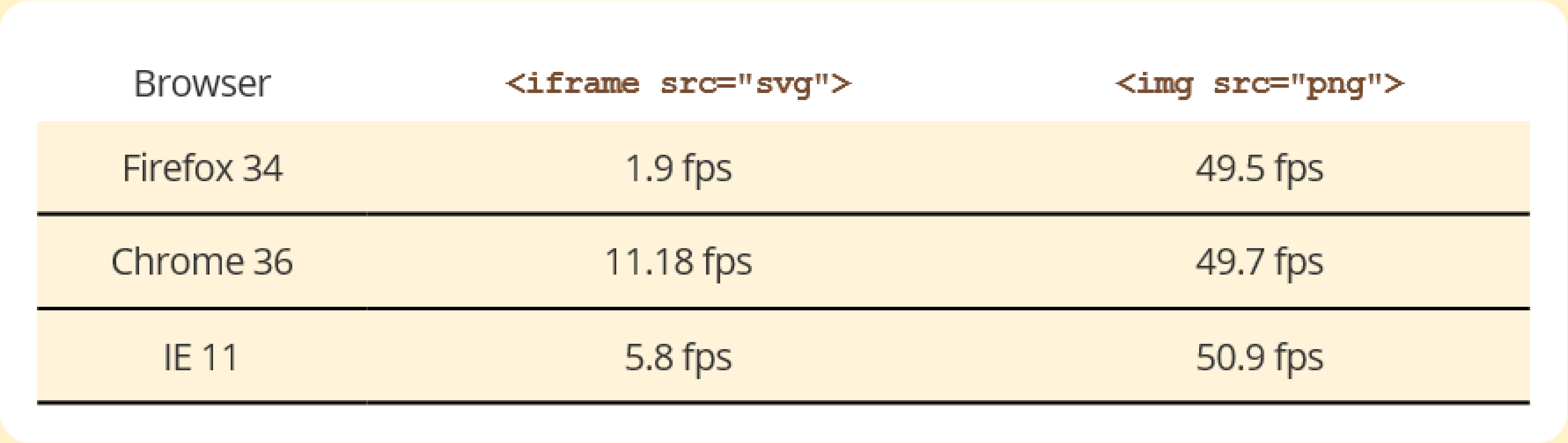


Animating `transform` (scale) of blurred copy → **49fps**

Try it at home!

Try replacing expensive effects with simpler ones

PRE-RENDERING



Browser	`<iframe src="svg">`	`<img src="png">`
Firefox 34	1.9 fps	49.5 fps
Chrome 36	11.18 fps	49.7 fps
IE 11	5.8 fps	50.9 fps

Try it at home!

Pre-render expensive assets

We can sometimes make things faster by pre-rendering.
Desktop apps, native apps, Flash apps, everyone does it.

PRE-RENDERING

Browser	`<iframe src="svg">`	`<img src="svg">`	`<img src="png">`
Firefox 34	1.9 fps	**49.1 fps**	49.5 fps
Chrome 36	11.18 fps	**13.0 fps***	49.7 fps
IE 11	5.8 fps	**15.5 fps**	50.9 fps

* Some rendering defects

Try it at home!

Try using `<img>` to embed SVG images instead of `<iframe>` (or `<object>`, `<embed>`).

Alternatively, for SVG, simply referring to the SVG using <img> instead of <iframe> lets the browser make more optimizations. Especially Firefox.

HARDWARE ACCELERATION

- Paths, text **Not much**
- Filters **Coming**
- Compositing **OK**

Most browsers hardware accelerate layer compositing.
That means they can often paint an animated element once then
just change its transform, opacity etc. and let the GPU re-composite.
That saves a lot of painting.

WHAT GETS A LAYER?

Animated `transform`

Animated `opacity`

3D `transform`

It's up to the browser what gets a layer.
Typically it's things like the above.

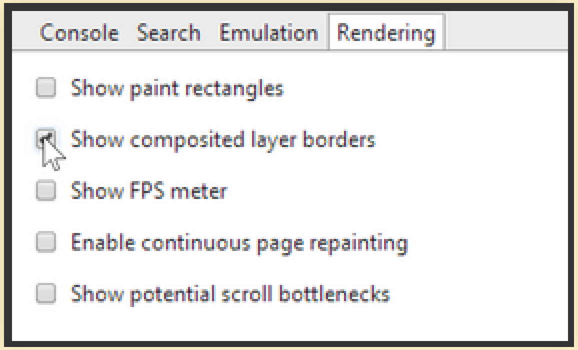
INSPECTING LAYERS

- Firefox: `about:config`

  → `layers.draw-borders` to true

  (requires `layers.offmainthreadcomposition.enabled` to be true)

- Chrome: DevConsole → Rendering → Show composited layer borders



□ Show paint rectangles

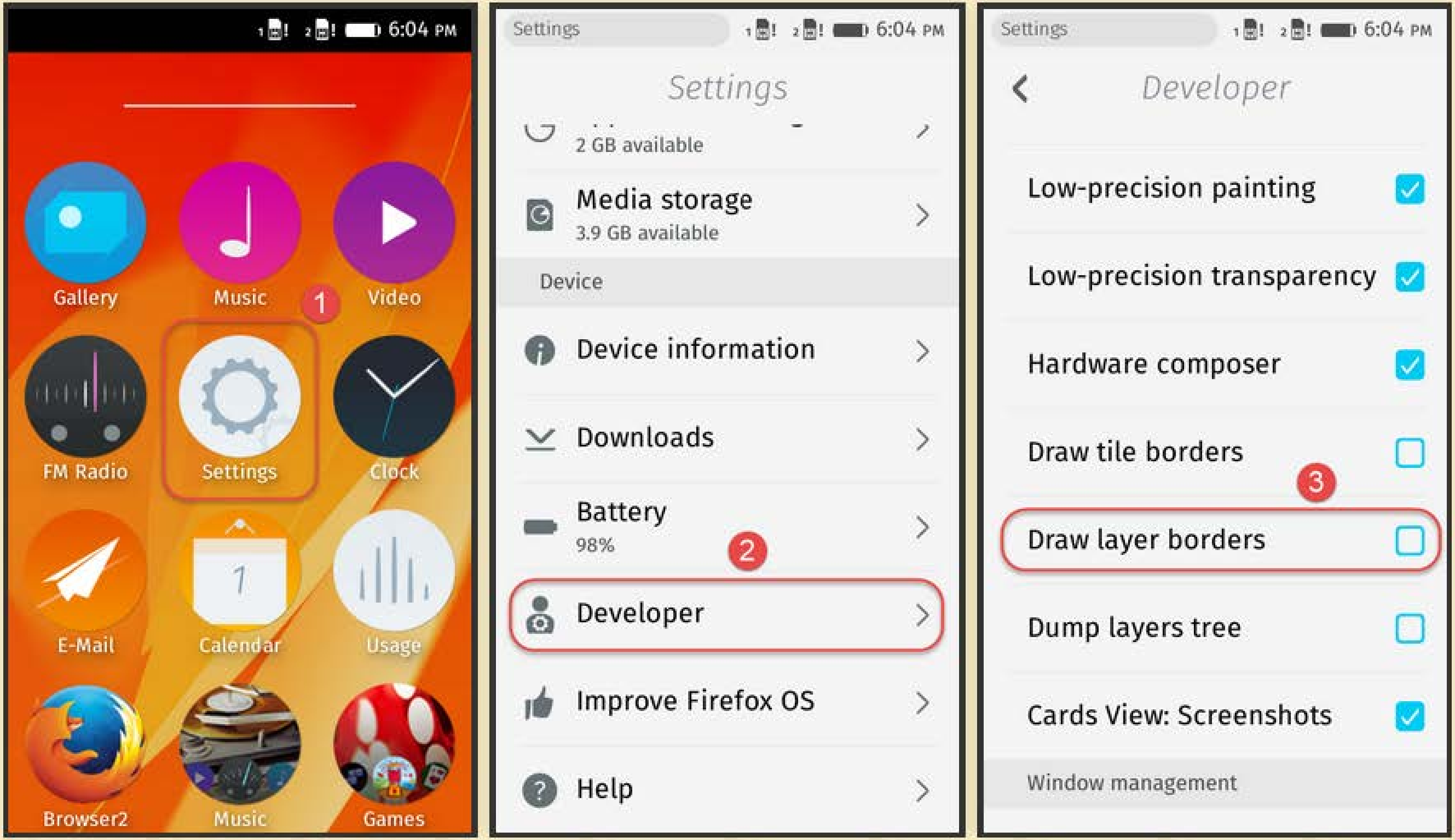
☑ Show composited layer borders

□ Show FPS meter

□ Enable continuous page repainting

□ Show potential scroll bottlenecks

INSPECTING LAYERS



Low-precision painting ☑

Low-precision transparency ☑

Hardware composer ☑

Draw tile borders □

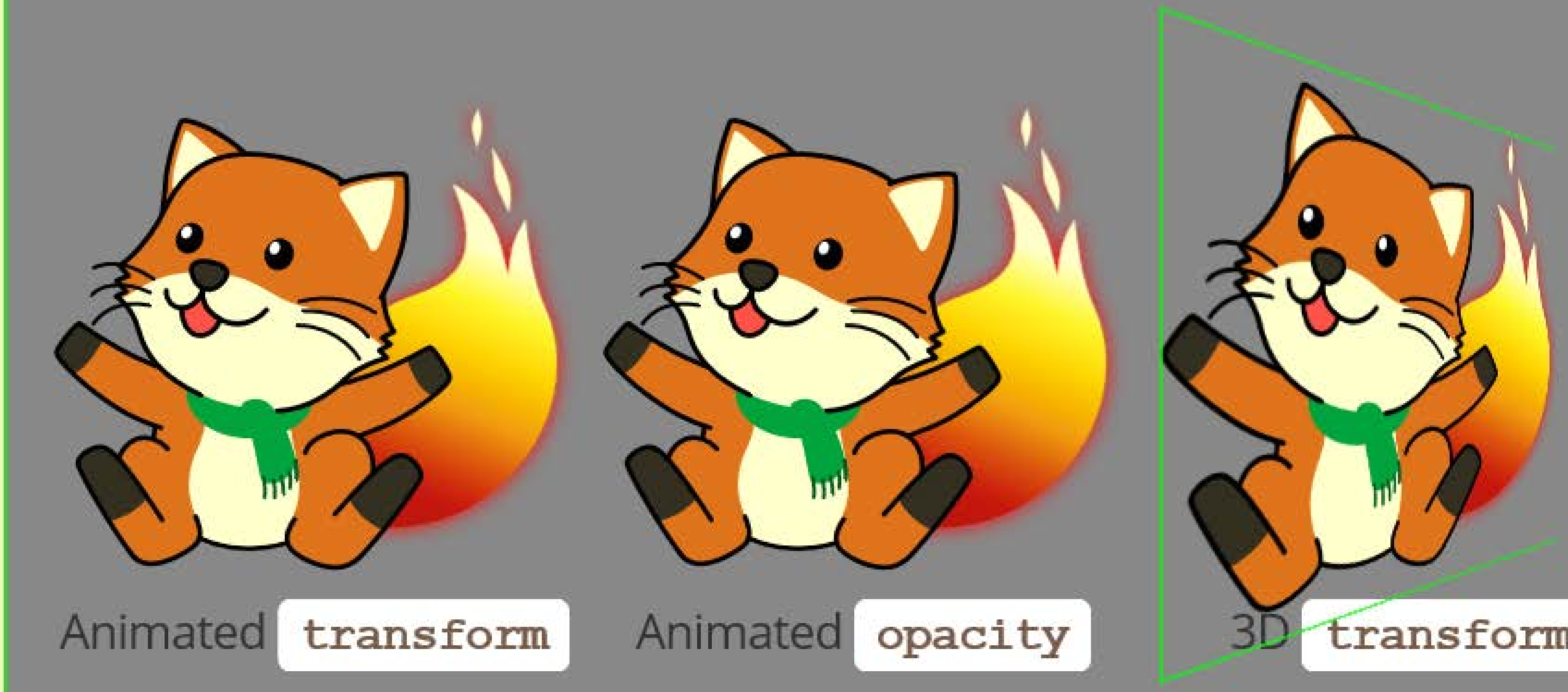
Draw layer borders □

Dump layers tree □

Cards View: Screenshots ☑

Firefox OS

INSPECTING LAYERS

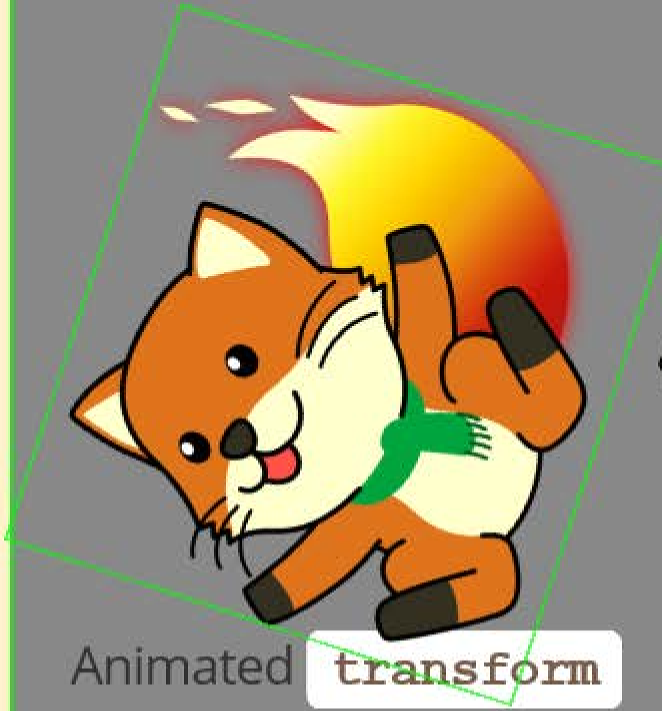


Animated `transform`

Animated `opacity`

3D `transform`

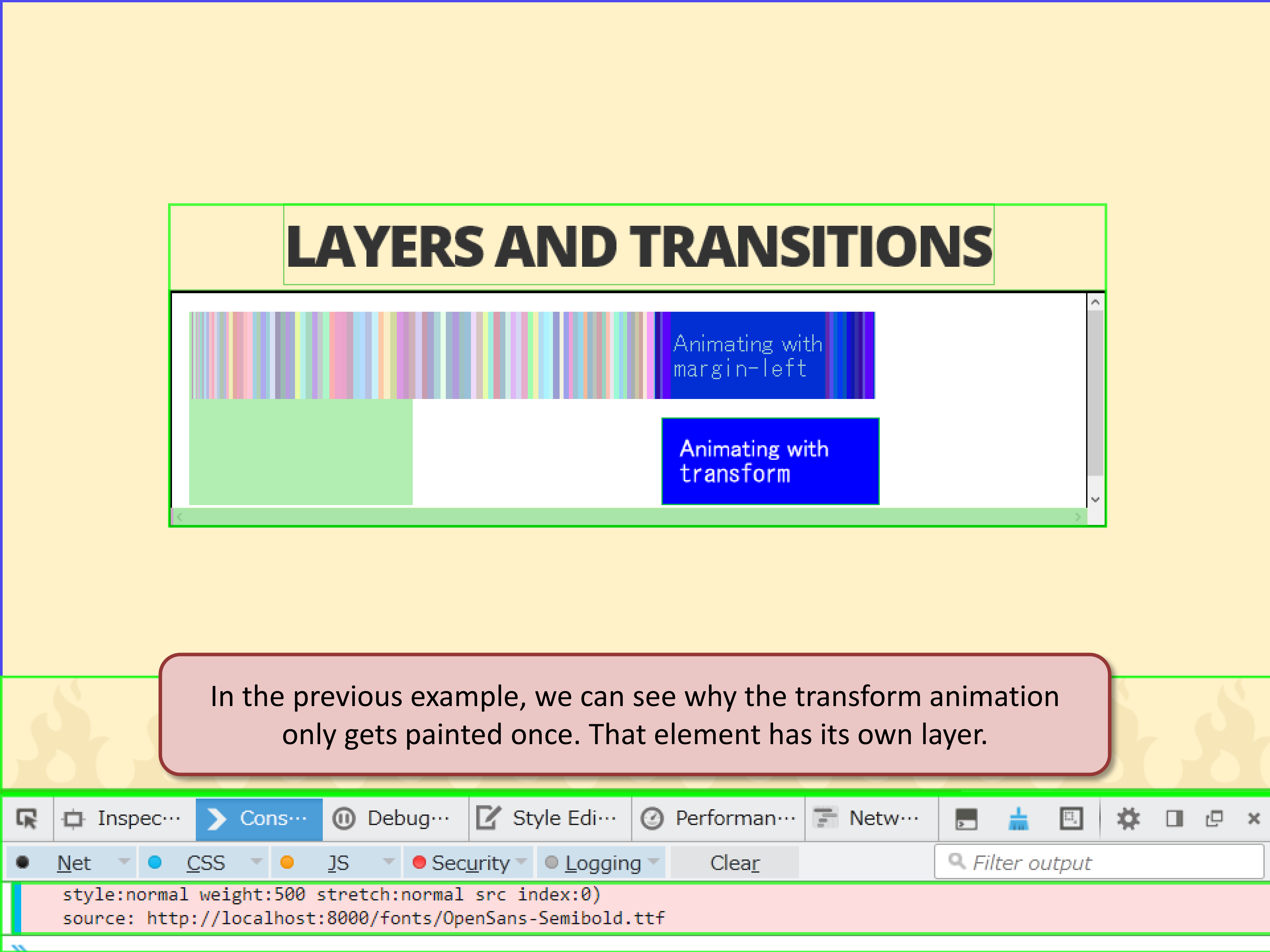
INSPECTING LAYERS



Animated `transform`

Animated `opacity`

3D `transform`



LAYERS AND TRANSITIONS

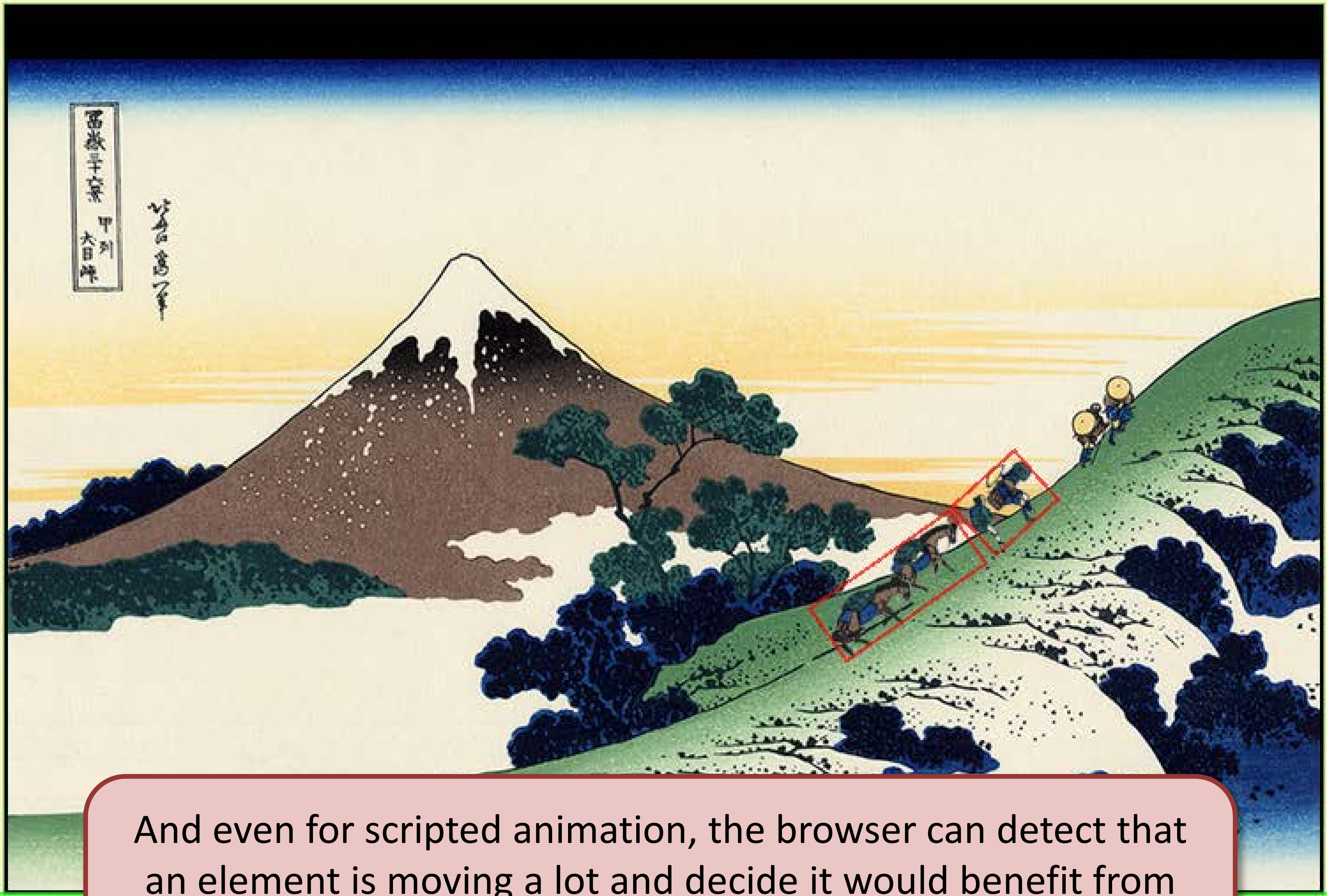
In the previous example, we can see why the transform animation only gets painted once. That element has its own layer.

```
style:normal weight:500 stretch:normal src index:0)
source: http://localhost:8000/fonts/OpenSans-Semibold.ttf
```

LAYERS AND SVG ANIMATION

Layerization is performed by the browser so it can automatically do it for SVG (SMIL) animation too.

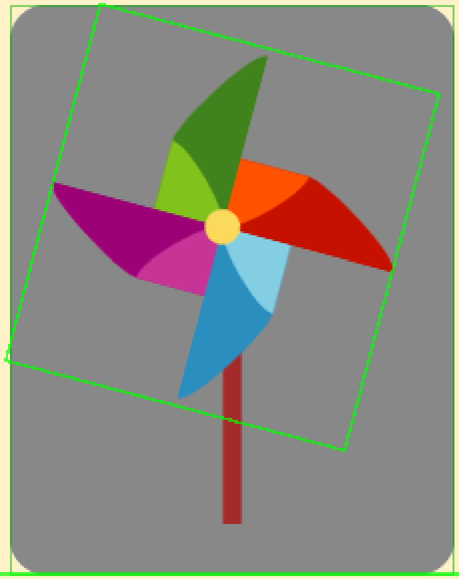
LAYERS AND SCRIPTED ANIMATION



And even for scripted animation, the browser can detect that an element is moving a lot and decide it would benefit from being on a separate layer.

(The red boxes in this example indicate image layers.)

STARTING AN ANIMATION



Often, however, the browser won’t create a layer until an element starts animating. Sometimes that can be too late and can cause the animation to stutter at the start as the browser sets up the layer.

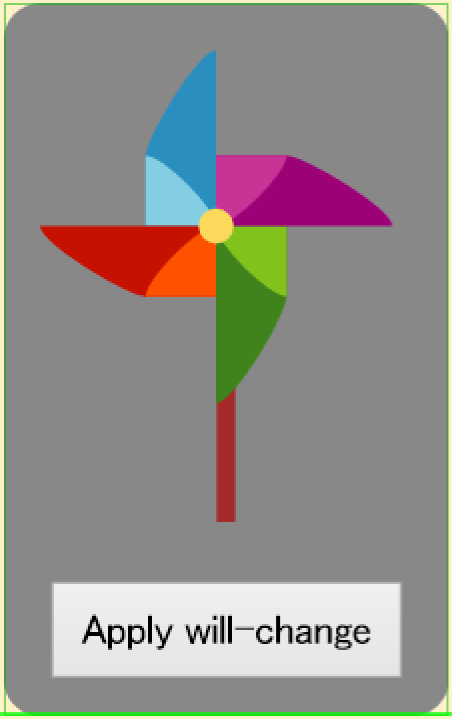
ENTER `will-change`

- `will-change:<property>`
  - `will-change:transform`
  - `will-change:opacity`
- `will-change:scroll-position`
- `will-change:contents`
- ~~`transform:translateZ(0)`~~
- Firefox: need `layout.css.will-change.enabled` in `about:config`.

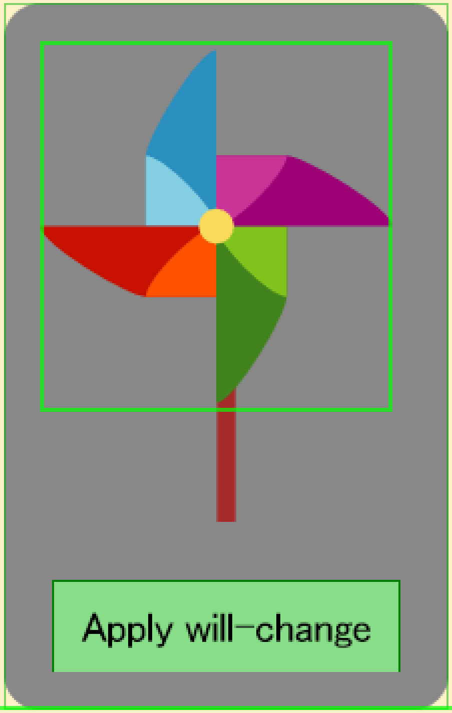
will-change:transform/ opacity/etc. lets the browser create the layer in advance if it thinks that would help improve performance.

transform:translateZ(0) doesn't work cross-browser

APPLYING `will-change`



APPLYING `will-change`

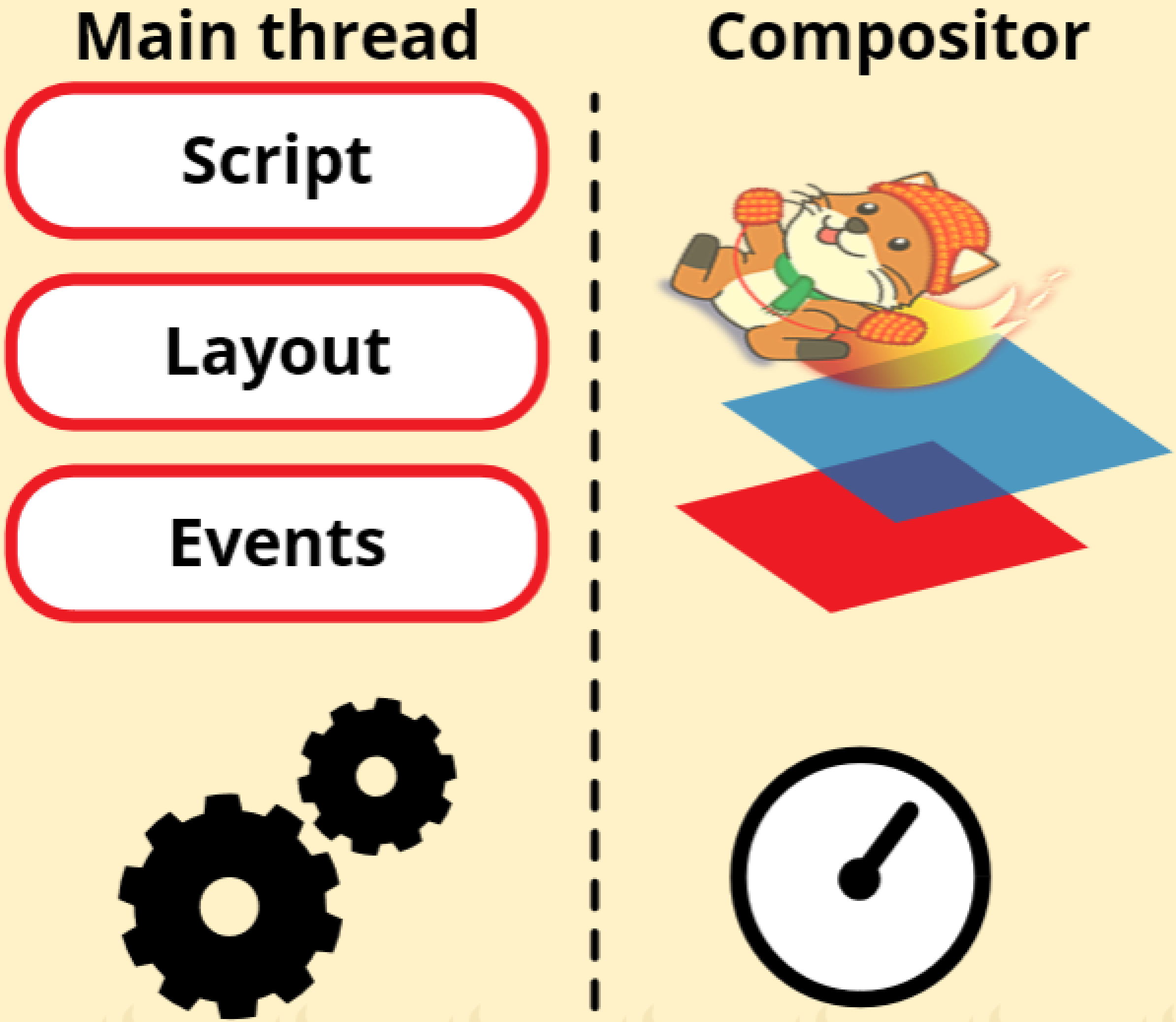


YOUR BROWSER IS A JANK

Interrupt

Apart from low frame-rates, animation performance is affected by other processes on the same thread like layout, garbage collection, or other scripts, that cause the animation to stop and start (jank).

COMPOSITOR ANIMATION



Main thread

Compositor

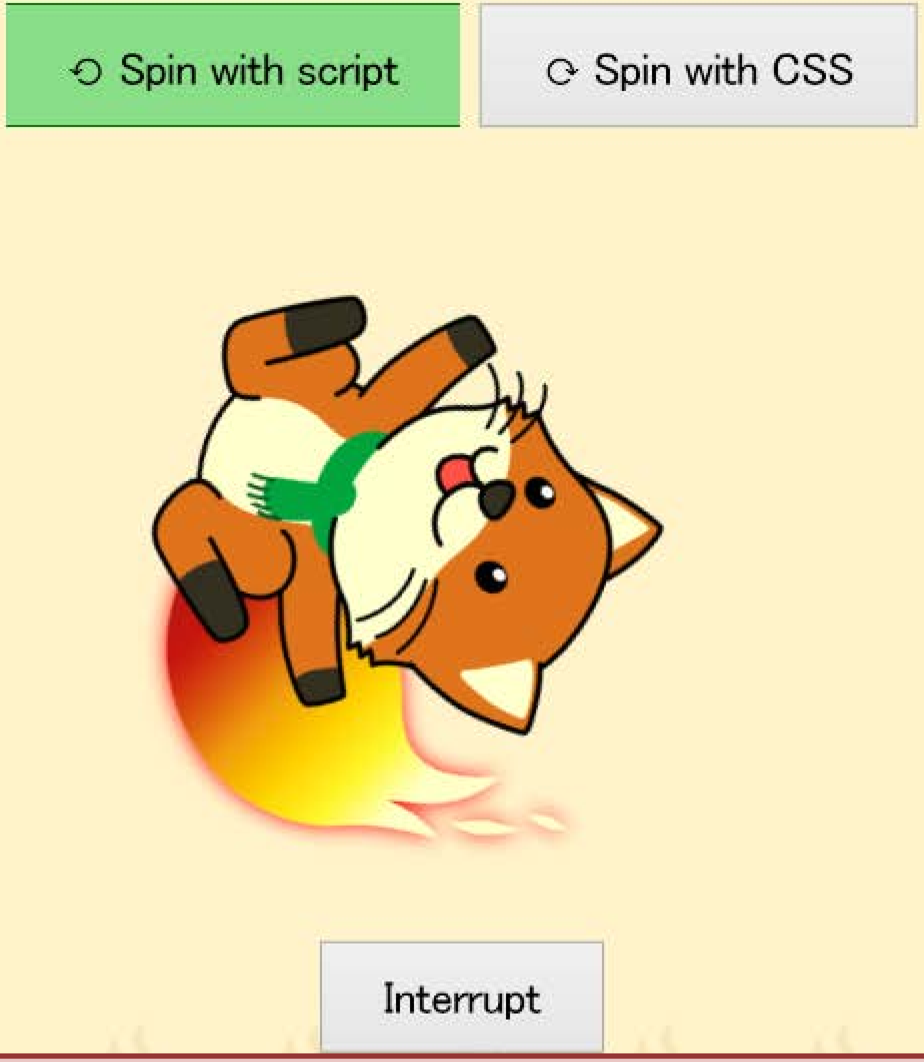
Script

Layout

Events

To avoid jank, some animations can be run on a separate thread/process.

ANIMATION ON THE COMPOSITOR



Unlike animations running on the main thread which stop and start…

ANIMATION ON THE COMPOSITOR

⟲ Spin with script

⟳ Spin with CSS

Interrupt

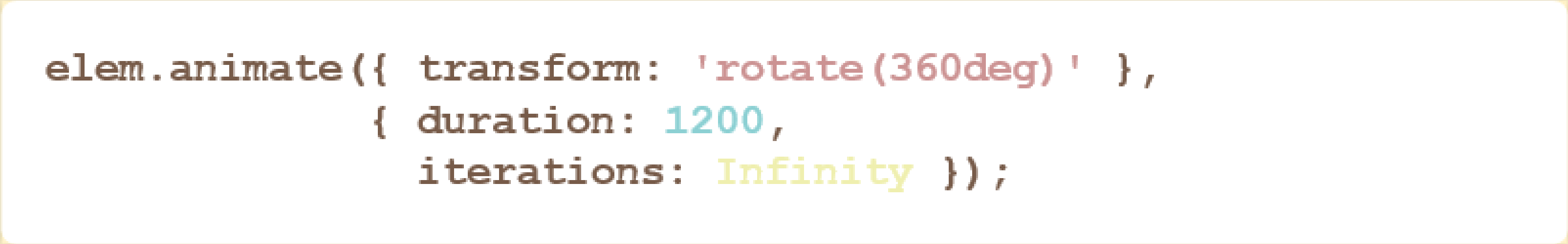
… these animations continue along uninterrupted.

NOT SO FAST...

- Representable by compositor? (e.g. transform, opacity)
- Supported platform? (e.g. Firefox OS)
  (`layers.offmainthreadcomposition.async-animations` → true)
- Other limitations: top/left also animated?
- Controlled by the browser? (e.g. CSS Animations)

But not everything can be animated in this way.
In particular, when the browser doesn’t know all the parameters of the animation—like most scripted animations—the browser can’t delegate the animation to another thread/process.

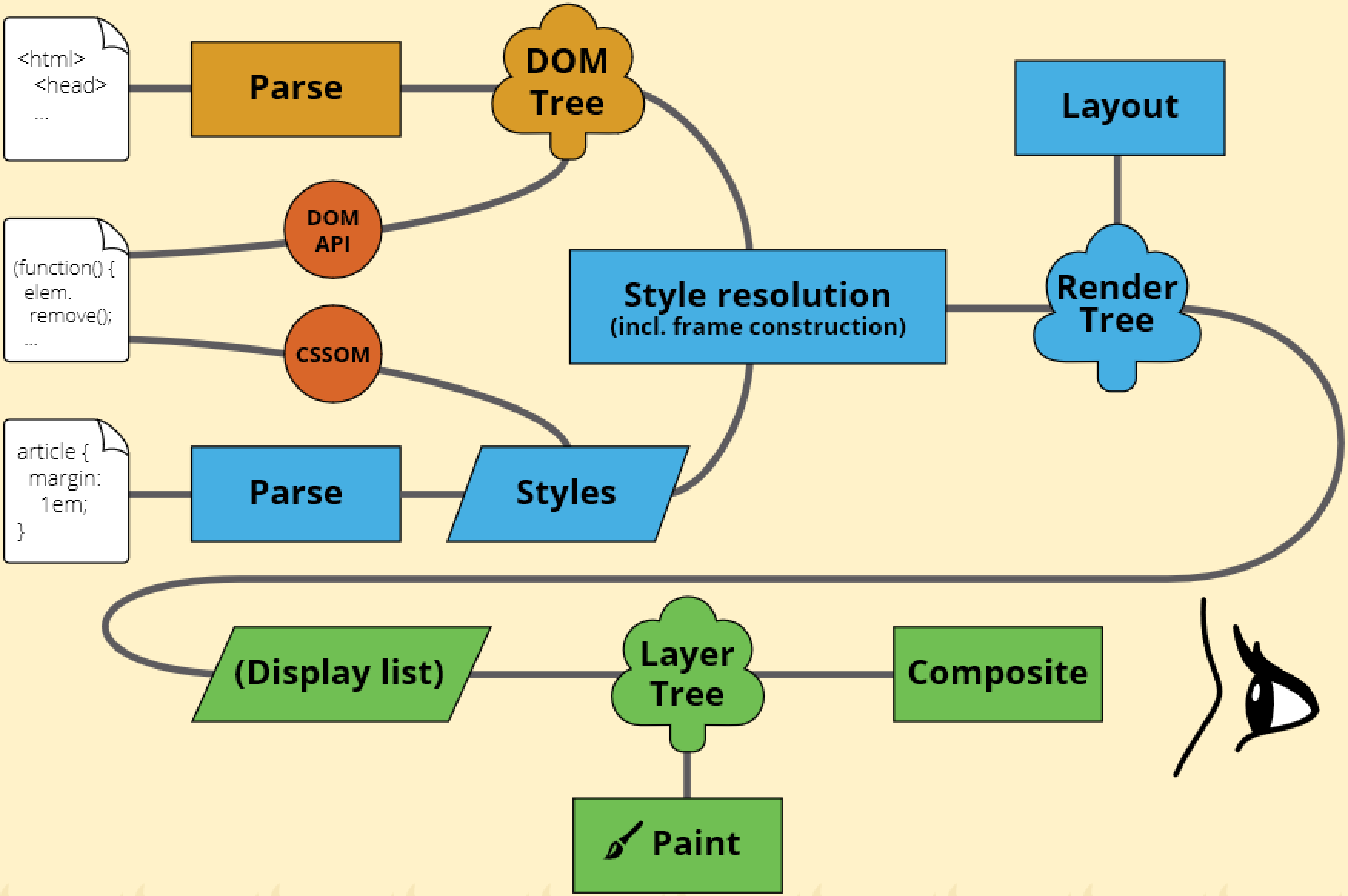
WHAT ABOUT SCRIPT?



```
elem.animate({ transform: 'rotate(360deg)' },
             { duration: 1200,
               iterations: Infinity });
```

- Chrome 36
- Polyfill web-animations-js and web-animations-next

One way around this is to use the Web Animations API to create animations.
This lets the browser optimize the animation in the same way as it
does for declarative animations like CSS Animations/Transitions.



```
<html>
  <head>
  ...
```

Parse

DOM Tree

```
(function() {
  elem.
   remove();
  ...
```

DOM API

CSSOM

```
article {
  margin:
   1em;
}
```

Parse

Styles

Style resolution
(incl. frame construction)

Layout

Render Tree

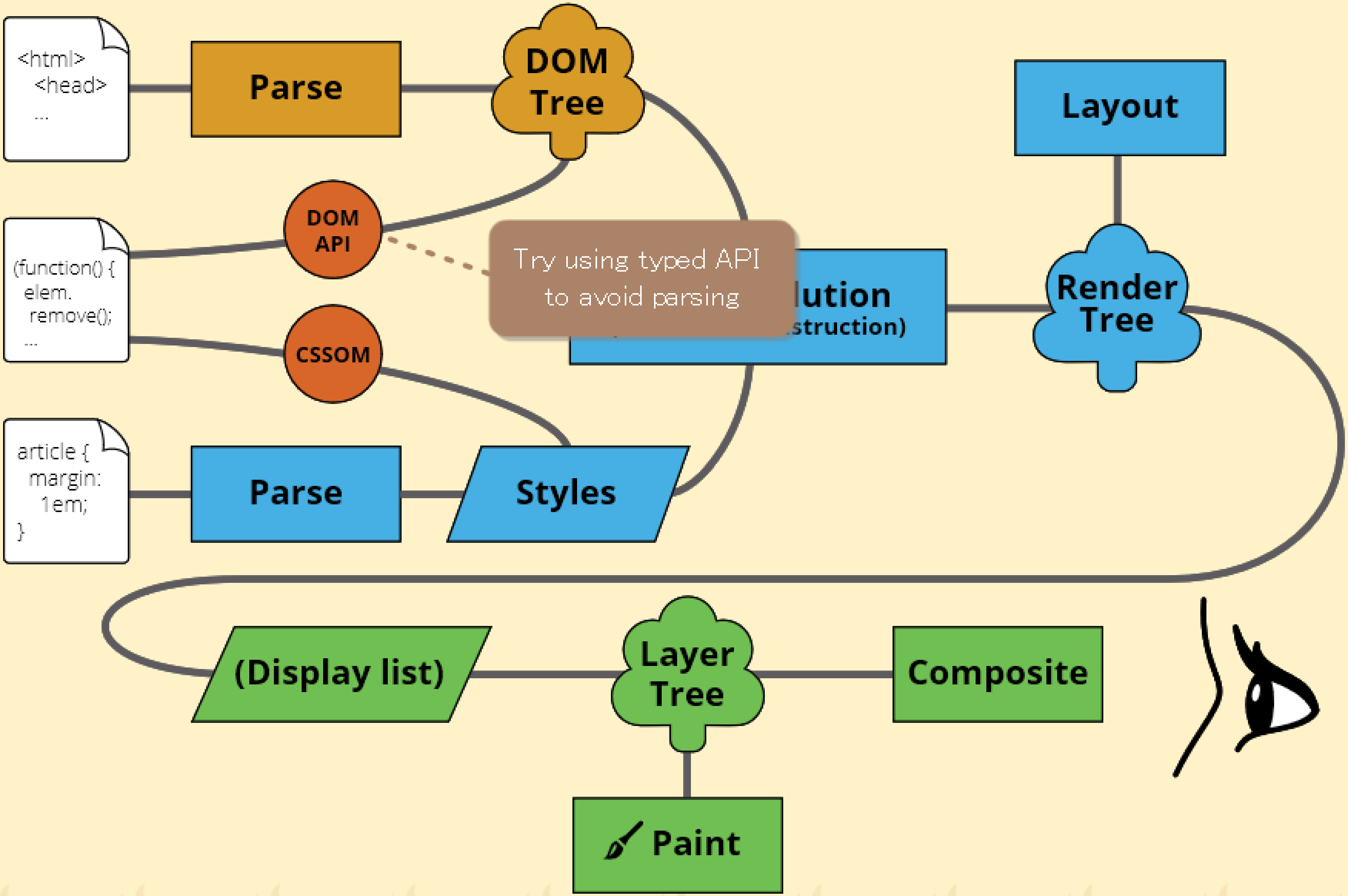
(Display list)

Layer Tree

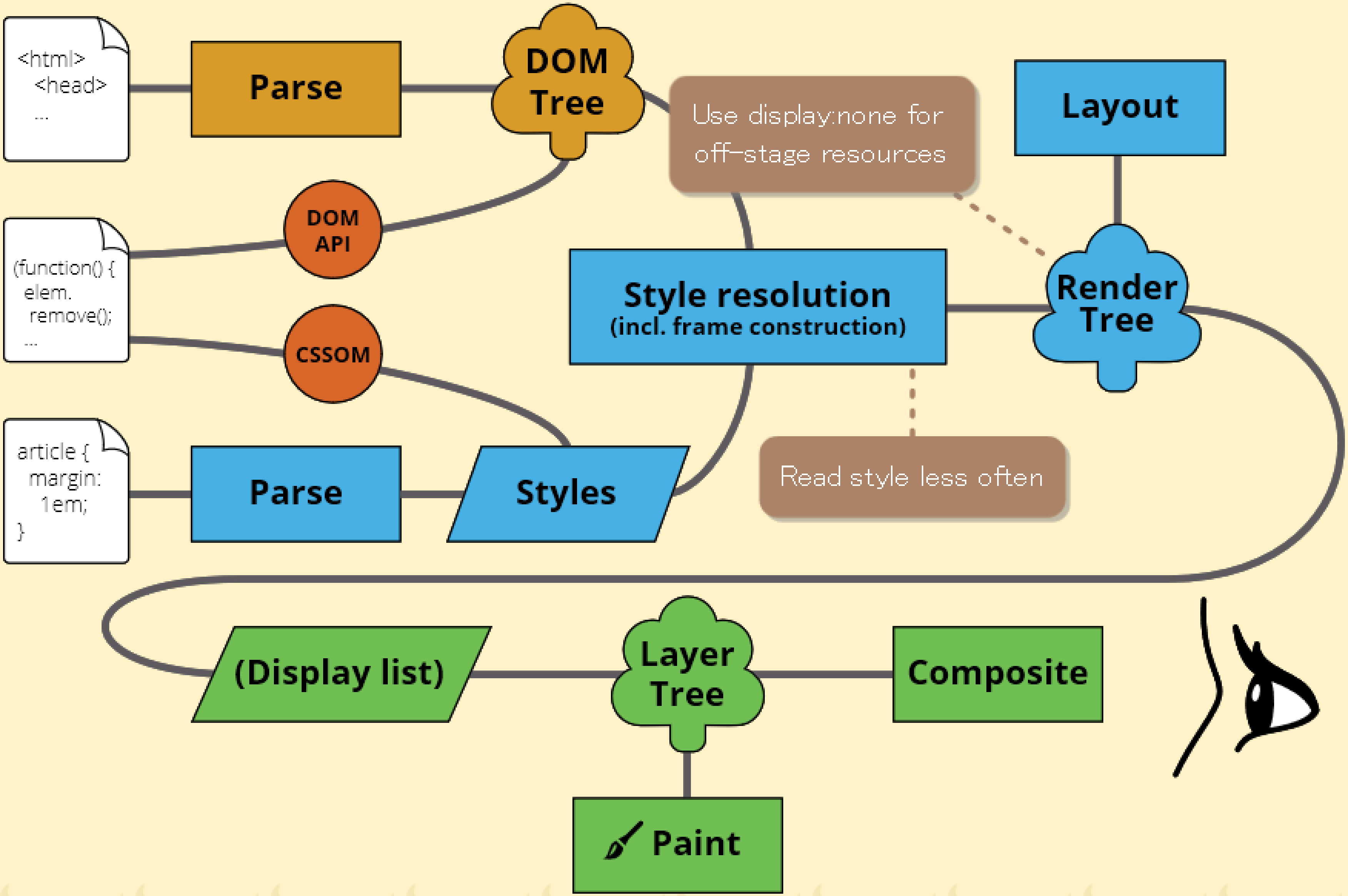
Composite

Paint

Summarizing our journey…



Try using typed API to avoid parsing



```
<html>
  <head>
  ...
```

Parse

DOM Tree

```
(function() {
  elem.
   remove();
  ...
```

DOM API

CSSOM

```
article {
  margin:
   1em;
}
```

Parse

Styles

Use display:none for off-stage resources

Style resolution (incl. frame construction)

Read style less often

Layout

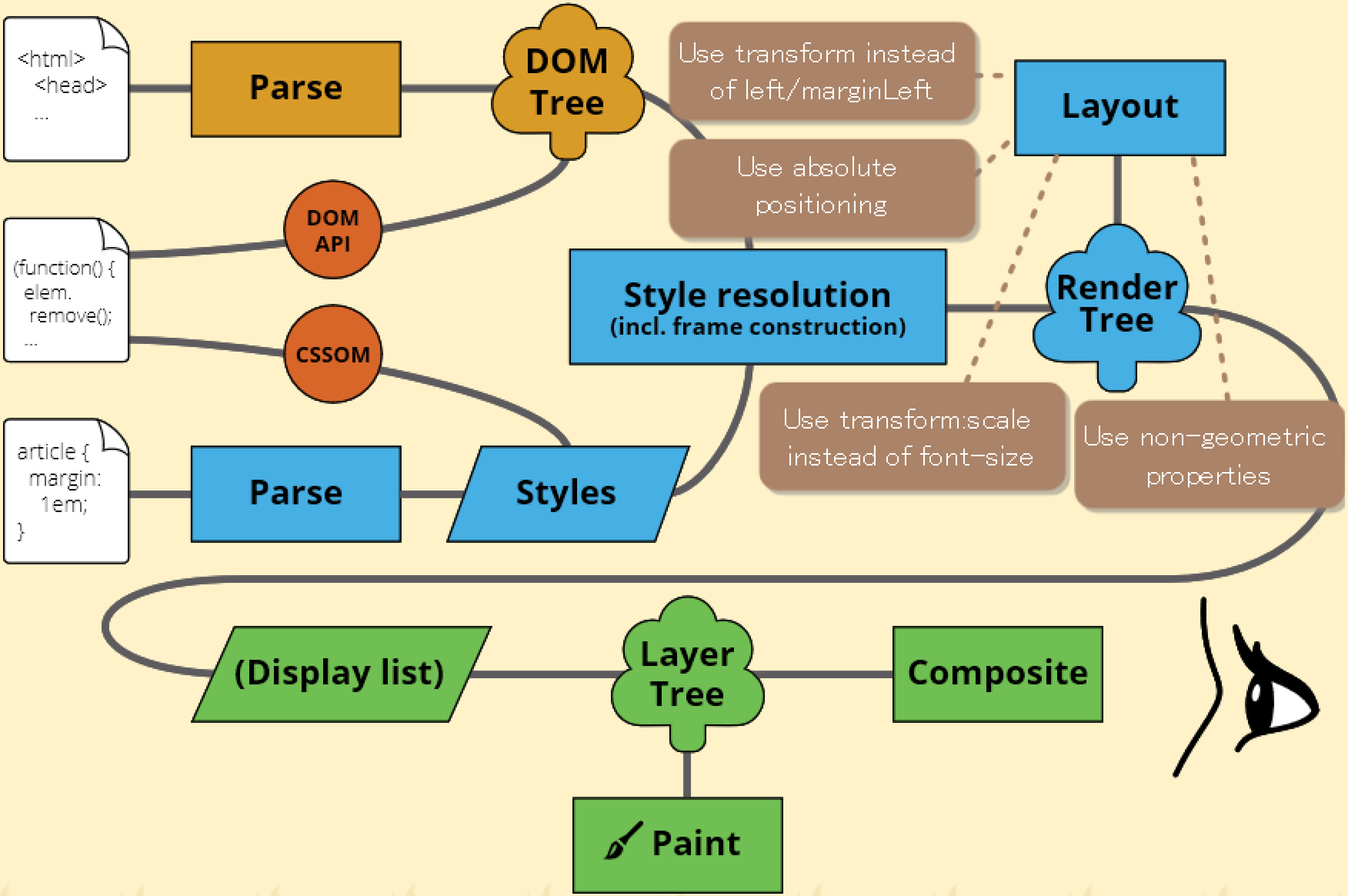
Render Tree

(Display list)

Layer Tree

Paint

Composite



```
<html>
  <head>
  ...
```

Parse

DOM Tree

```
(function() {
  elem.
   remove();
  ...
```

DOM API

CSSOM

```
article {
  margin:
   1em;
}
```

Parse

Styles

Style resolution
(incl. frame construction)

Use transform instead of left/marginLeft

Use absolute positioning

Layout

Render Tree

Use transform:scale instead of font-size

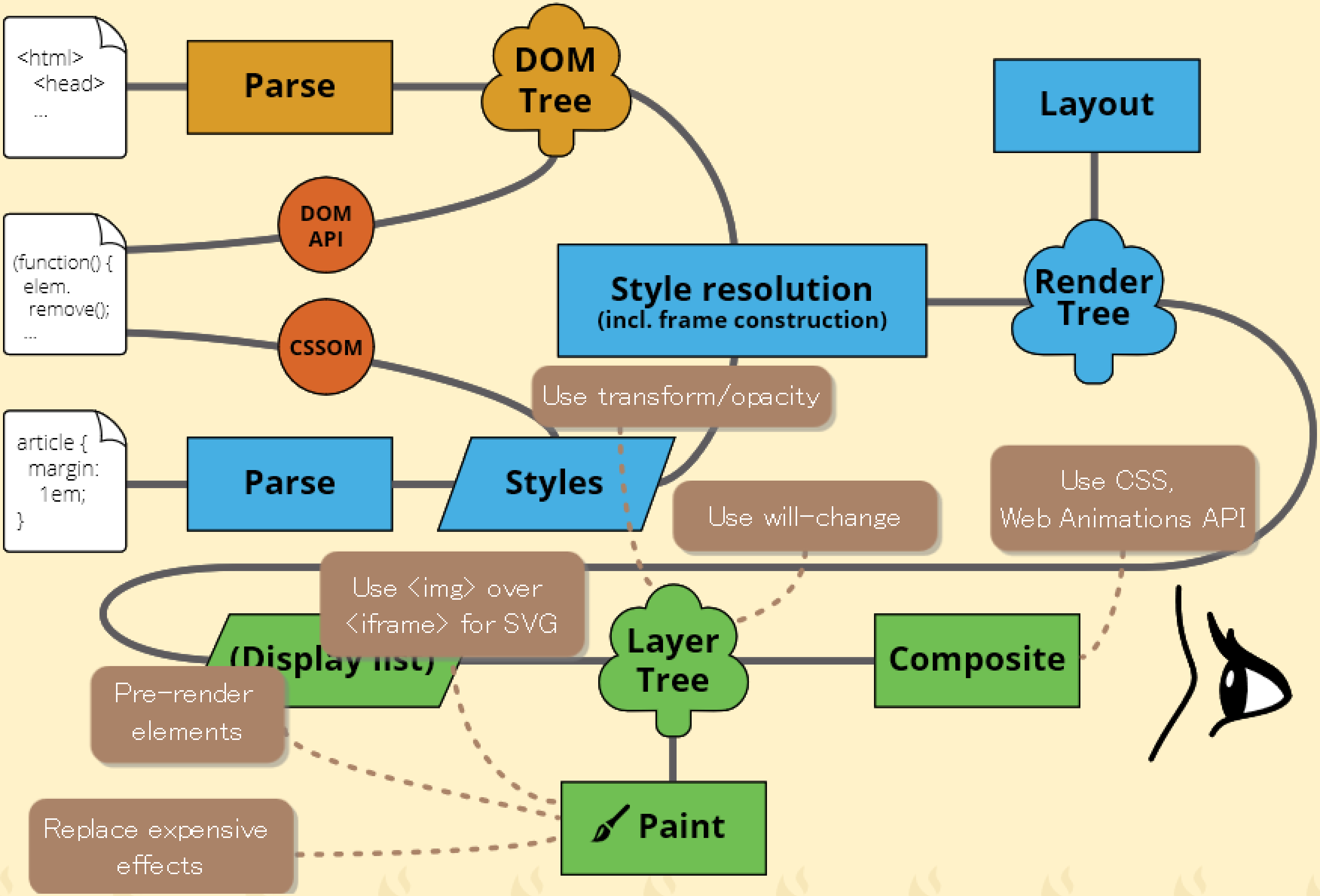
Use non-geometric properties

(Display list)

Layer Tree

Paint

Composite



```
<html>
  <head>
  ...
```

Parse

DOM Tree

DOM API

```
(function() {
  elem.
   remove();
 ...
```

CSSOM

```
article {
  margin:
   1em;
}
```

Parse

Styles

Style resolution
(incl. frame construction)

Layout

Render Tree

Use transform/opacity

Use will-change

Use CSS,
Web Animations API

Use <img> over
<iframe> for SVG

(Display list)

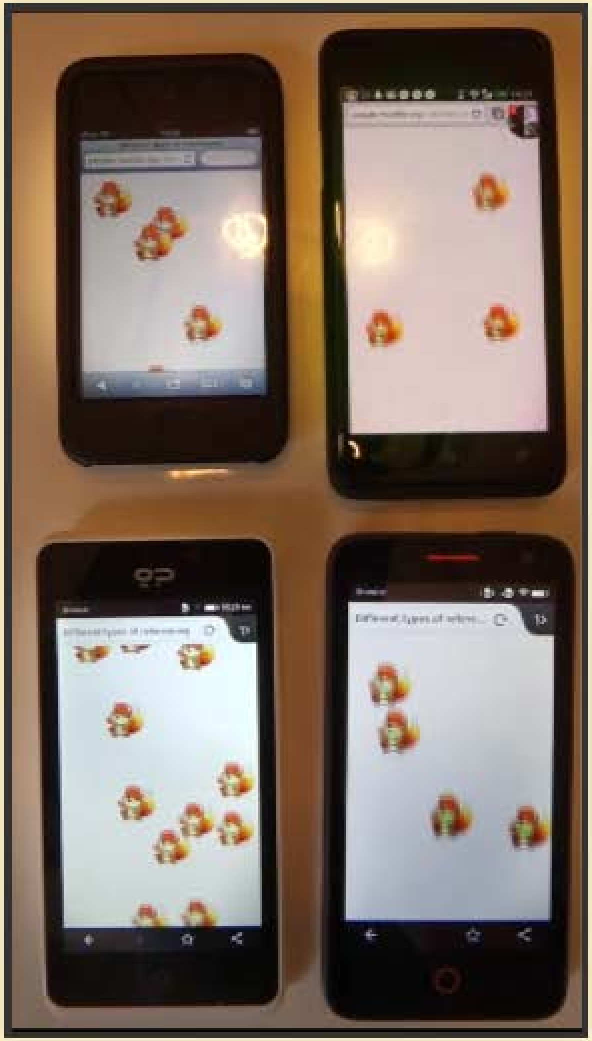
Layer Tree

Composite

Pre-render
elements

Replace expensive
effects

Paint



Using our knowledge of how browsers work we can make animations that run smoothly on any browser on any device and convey their intended effect.

Web Animations spec

dev.w3.org/fxtf/web-animations/

Brian Birtles

bbirtles@mozilla.com

@brianskold

HTML version of slides: **http://people.mozilla.org/~bbirtles/pres/graphical-web-2014/**