UIWorker and stuff

extensibility + speed

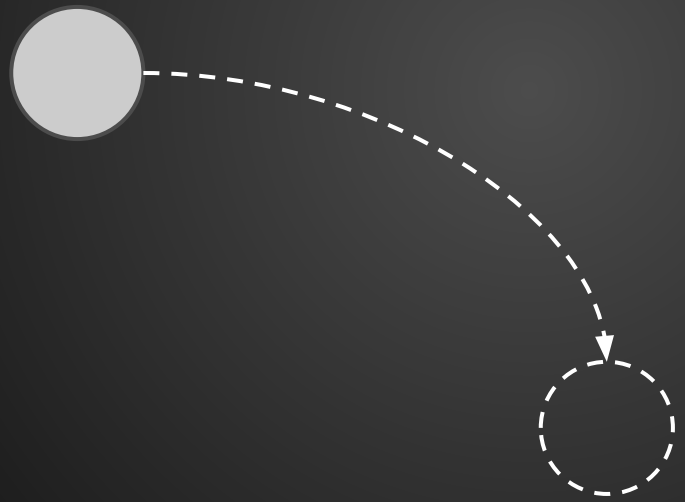
Problems

It’s tough to write performant, cross-UA code.
Can’t synchronize with threaded effects.

Can we make a pit of success?

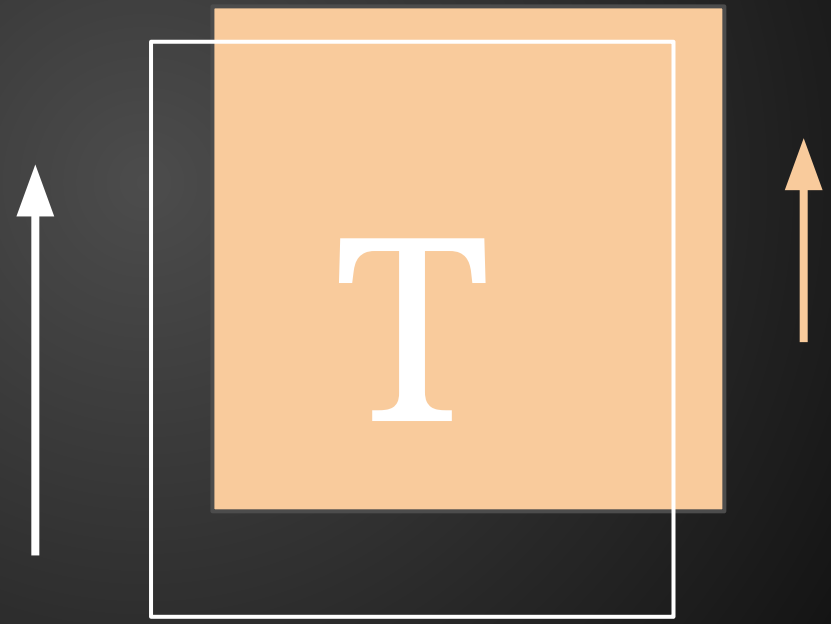
Motivating Examples

Dragging stuff



Motivating Examples

Parallaxing stuff



Motivating Example

(Lots of others)

- hidey bars
- snap points
- pull-to-refresh
- position:sticky
- ...

Problem

Current choices

- Extensibility
- Cross-UA Performance

Problem

Current choices

- Extensibility
- Cross-UA Performance

Can we have both?

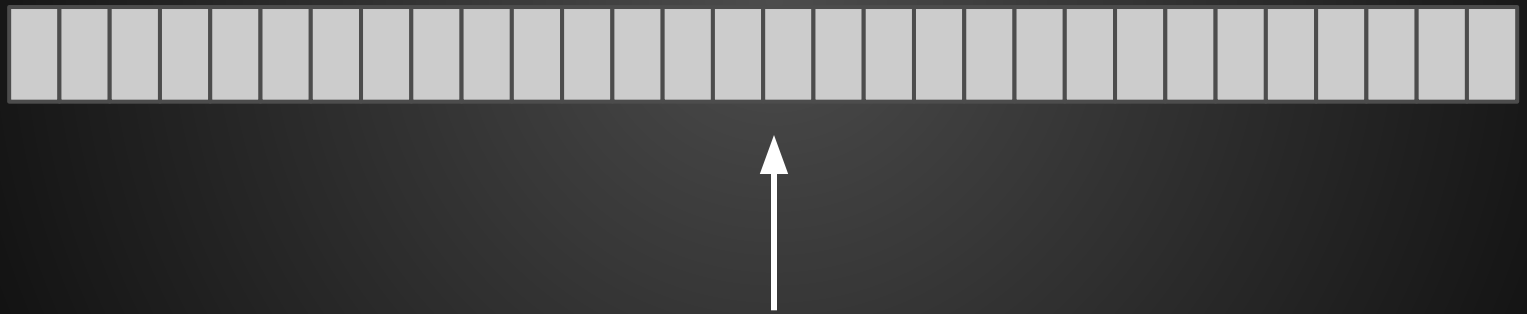
Sorta!

Stratospheric View

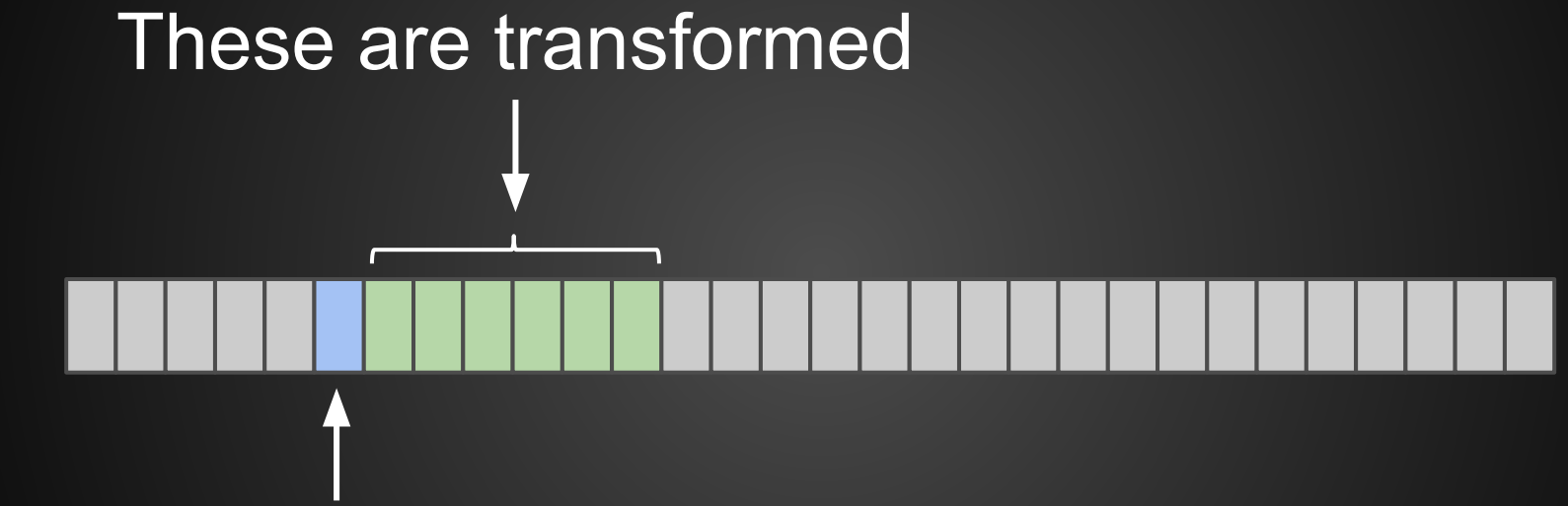
Here’s your page

Stratospheric View

Here’s your page



Stratospheric View



Stratospheric View

Can I mutate this in sync with scroll? touch?

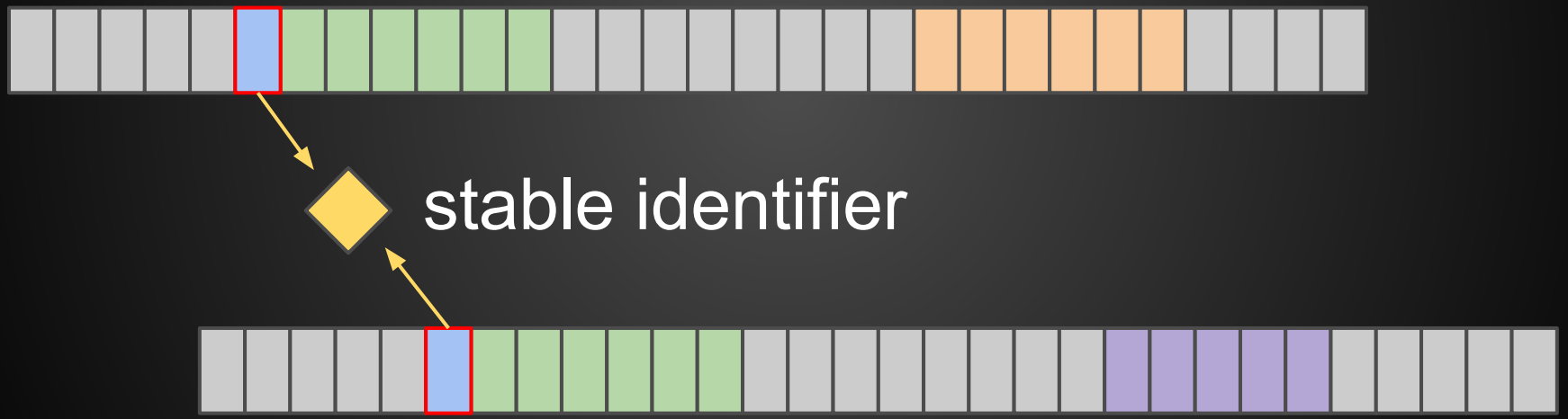
Yup.

Stratospheric View

main thread stuff

compositor thread stuff

Stratospheric View



Stratospheric View

```
// some javascript to muck with
```

Strawperson: UIWorker

```
// On main thread.
worker = new UIWorker('ui.js');
worker.postMessage(
    spinner.bindAnimatedProperty('transform'));
```

Strawperson: UIWorker

```
// On compositor thread.
var token = null;
onmessage = function(e) {
    token = e.data;
    var context = new RAFContext([token]);
    requestAnimationFrame(context, tick);
};
```

Strawperson: UIWorker

```
function tick(context) {
    var seconds = context.timestamp / 1000.0;
    var matrix = context.getMatrix(token);
    matrix.m14 =
        200.0 + 100.0 * Math.sin(seconds);
    context.setMatrix(token, matrix);
    requestAnimationFrame(context, tick);
}
```

Demo!

But that API is pretty gross

- Tokens and context aren’t “webby”.
- Doesn’t handle input.

What might a nice API look like?

Idea 1: Squeeze input into UIWorker

```
// main
var t1 = elem.bind(“touchmove”);
var t2 = elem.bind(“beforescroll”);
```

Idea 2: AnimationProxy

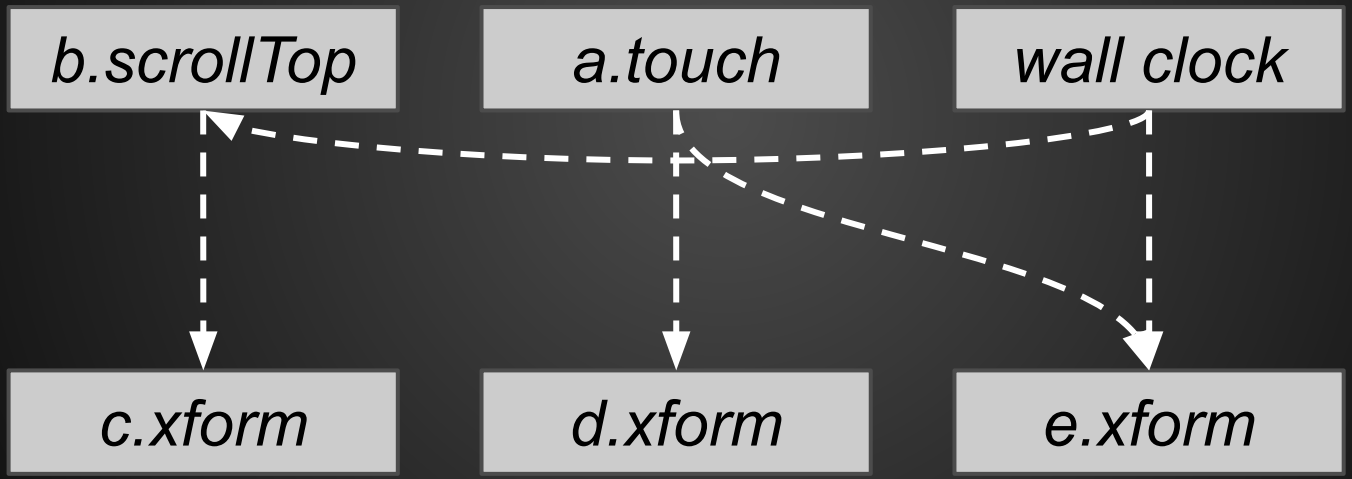
```
// main
var proxy = elem.getProxy(“transform”);

// compositor
proxy.transform = thingy;
```

Idea 3: Scene Graph

What if we had a DAG?

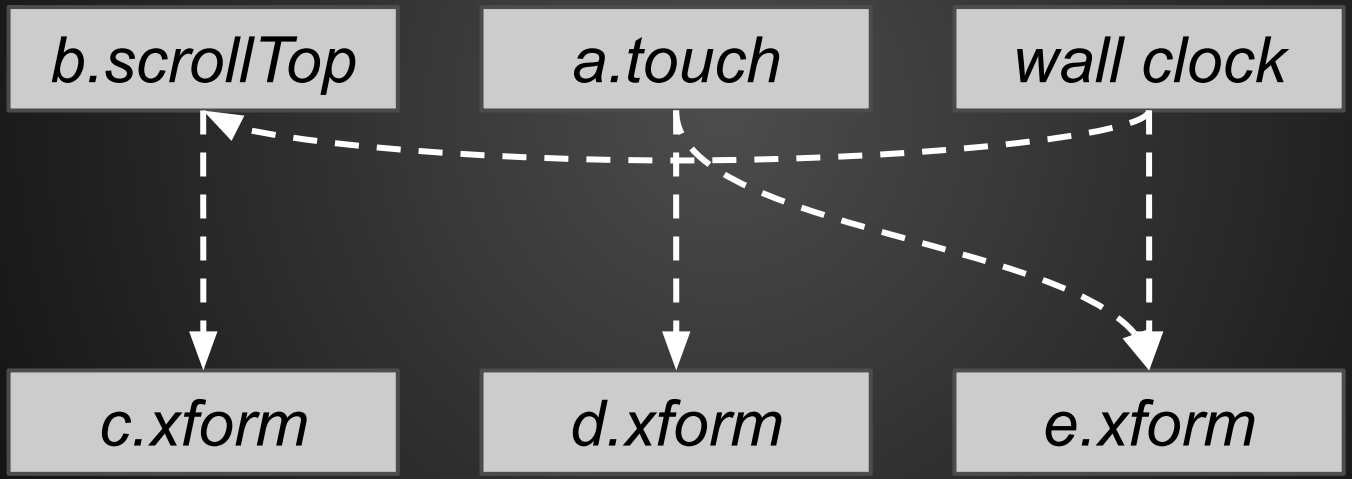
Idea 3: Scene Graph

What if we had a DAG?



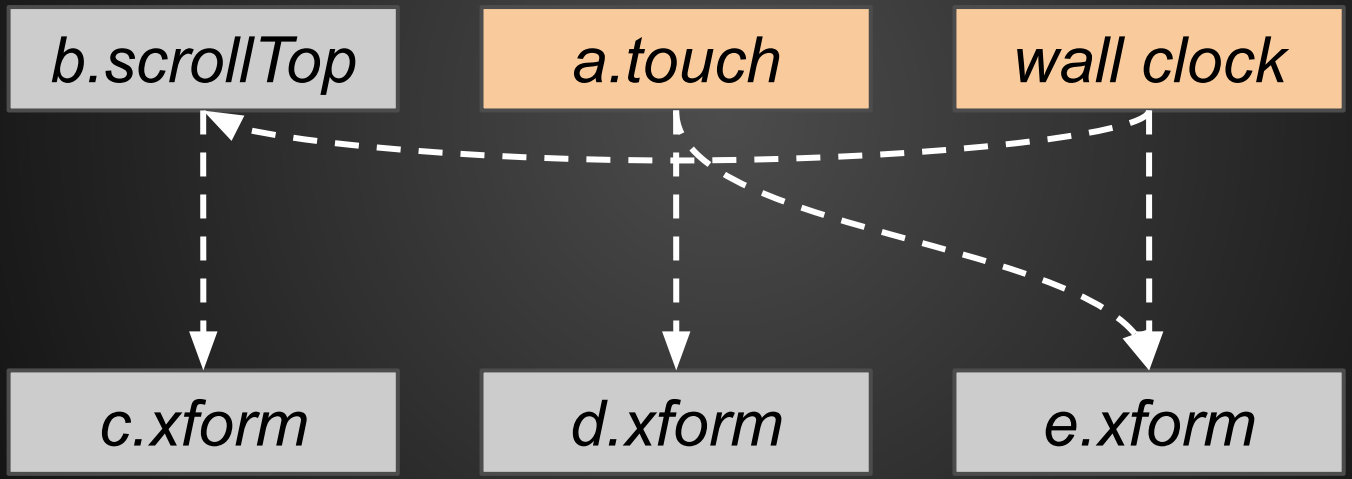
Idea 3: Scene Graph

Permits minimal updates.



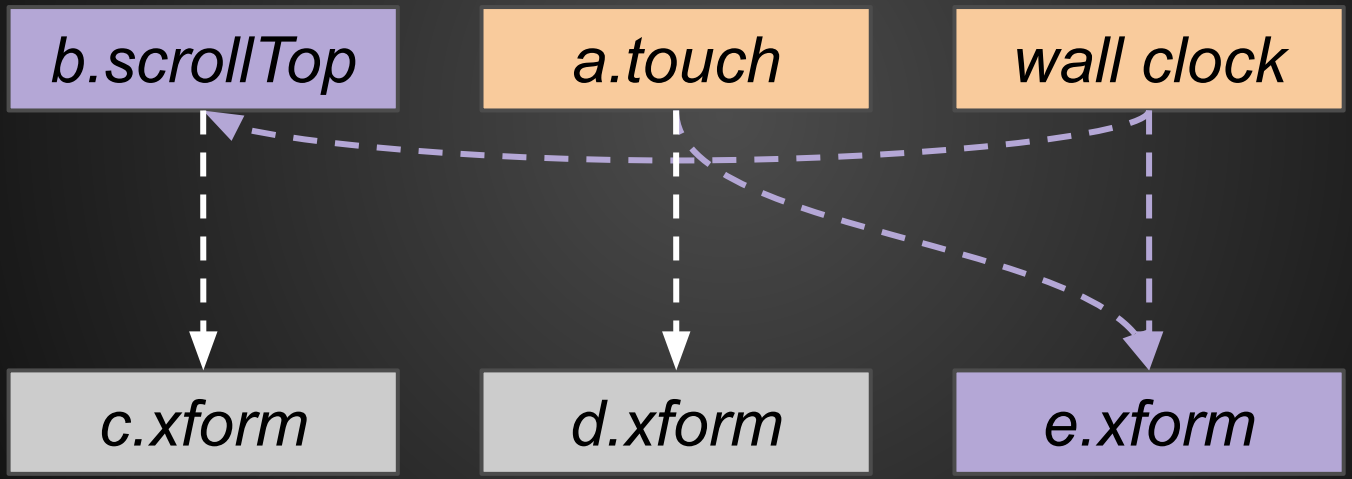
Idea 3: Scene Graph

Say touch and time have changed..



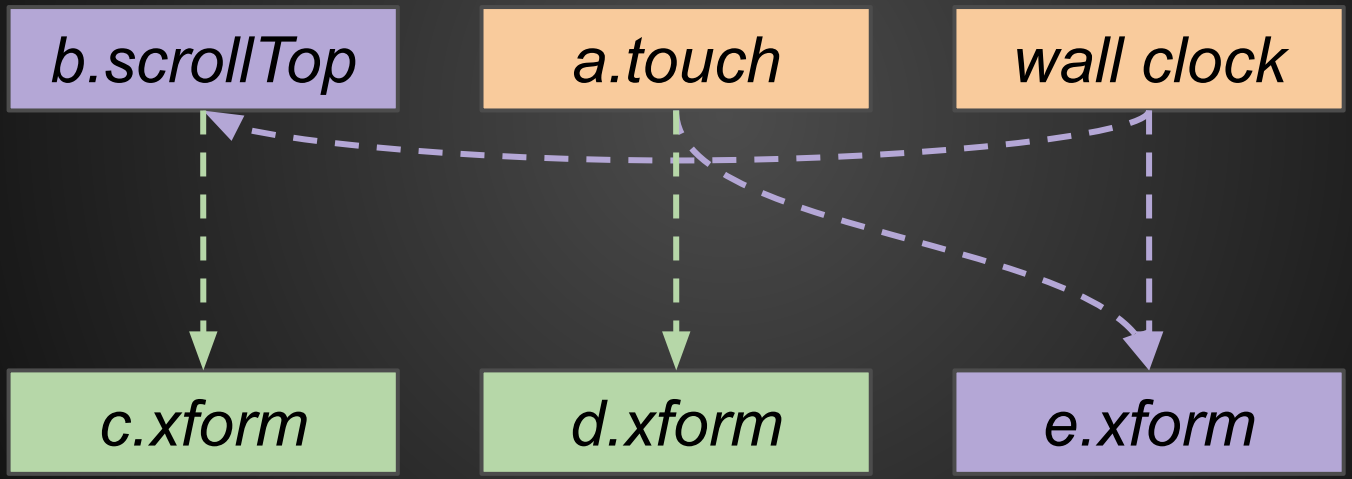
Idea 3: Scene Graph

Cleaning step 1



Idea 3: Scene Graph

Cleaning step 2



Idea 3: Scene Graph

```
// Parallax. (Warning: magical, illegal JS to follow.)
element.style.transform = pureFunction(top) {
    return CSSTransform(“translateY(“ + (top * 0.8) + “px)”);
}.fancyBind(other, “scrollTop”);
```

Idea 3: Scene Graph

```
// This lets us reuse our unbound functions.
var parallax = pureFunction(top, rate) {
    return CSSTransform(“translateY(“ + (top * rate) + “px)”);
}

back_a_bit.style.transform =
    parallax.fancyBind(other, “scrollTop”, 0.8);
way_back.style.transform =
    parallax.fancyBind(other, “scrollTop”, 0.2);
```

Idea 3: Scene Graph

```
// Before scroll.
element.beforescroll = pureFunction(scrollState) {
    scrollState.delta *= 0.5;
    return scrollState;
}

// Before scroll, custom bubbling.
element.beforescroll = pureFunction(scrollState) {
    ...
}.fancyBind(scroller, “beforescroll”);
```

Idea 3: Scene Graph

```
// Traditional animation, bound to wall clock.
element.style.opacity =
    pureFunction(timestamp){...}.fancyBind(time);

element.style.transform = pureFunction(e1, e2) {
    // Do some stuff while the mouse is over.
}.fancyBind(element, “touchmove”, window, “touchmove”);
```

Next Steps

- Implement behind a flag.
- Attempt to build tough test cases.

Further Reading

UIWorker explainer

Tracking bug