Immediate mode with immutable data

Tijs van der Storm
storm@cwi.nl / @tvdstorm

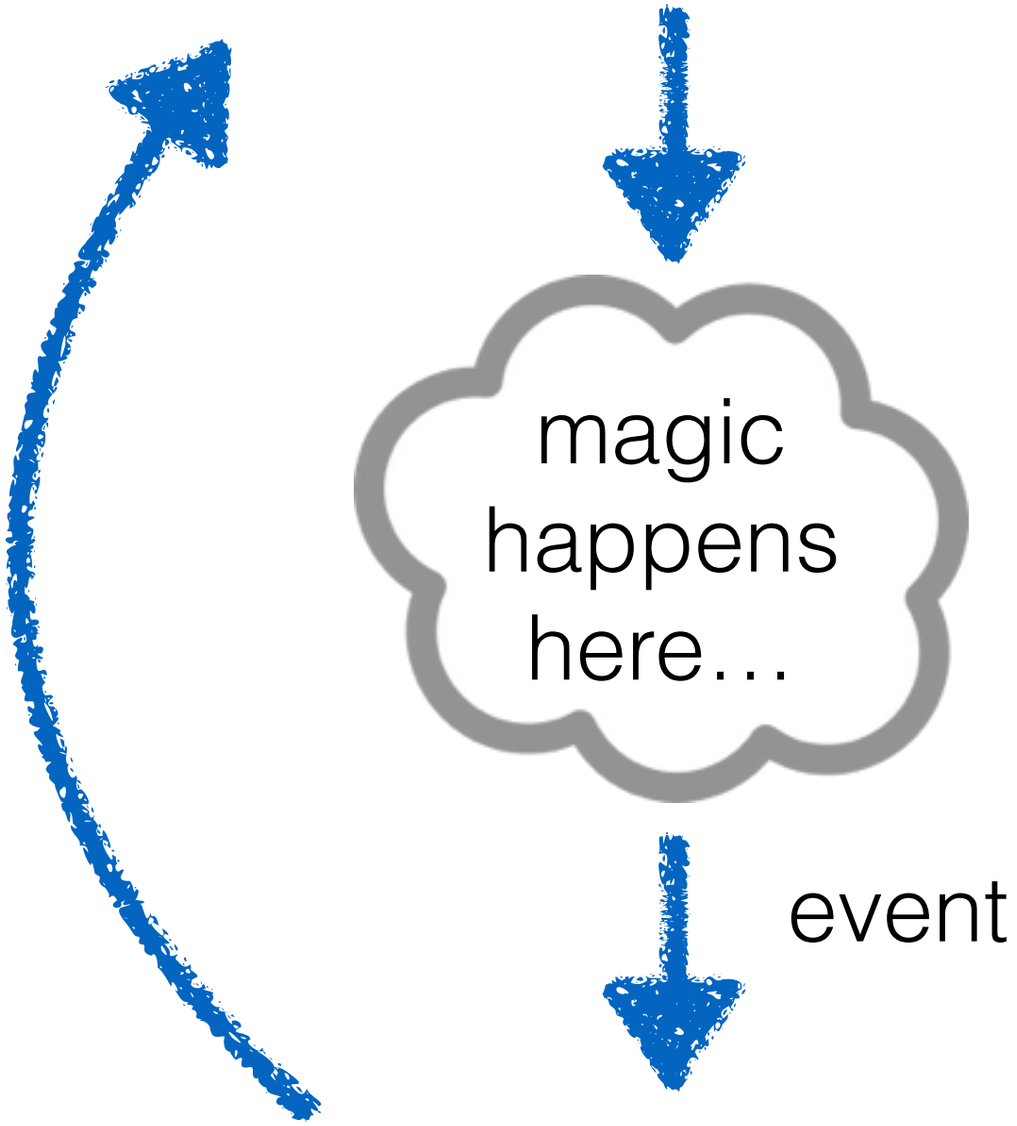
Immediate mode?

- Immediate mode vs retained mode UI
- Stateless drawing vs creating a stateful UI object tree
- No inversion of control (callbacks) or dependency networks; “straight line code”
- Origin in game development, now very popular: React, Elm etc. (virtual dom)

Immutable data?

- Functional programming!!! ;)
- “Copy on write data”
- Avoids the “goto of memory”: aliasing
- Snapshotting, time travel, undo etc. become easier.
- -> Elm

view: Model -> Html

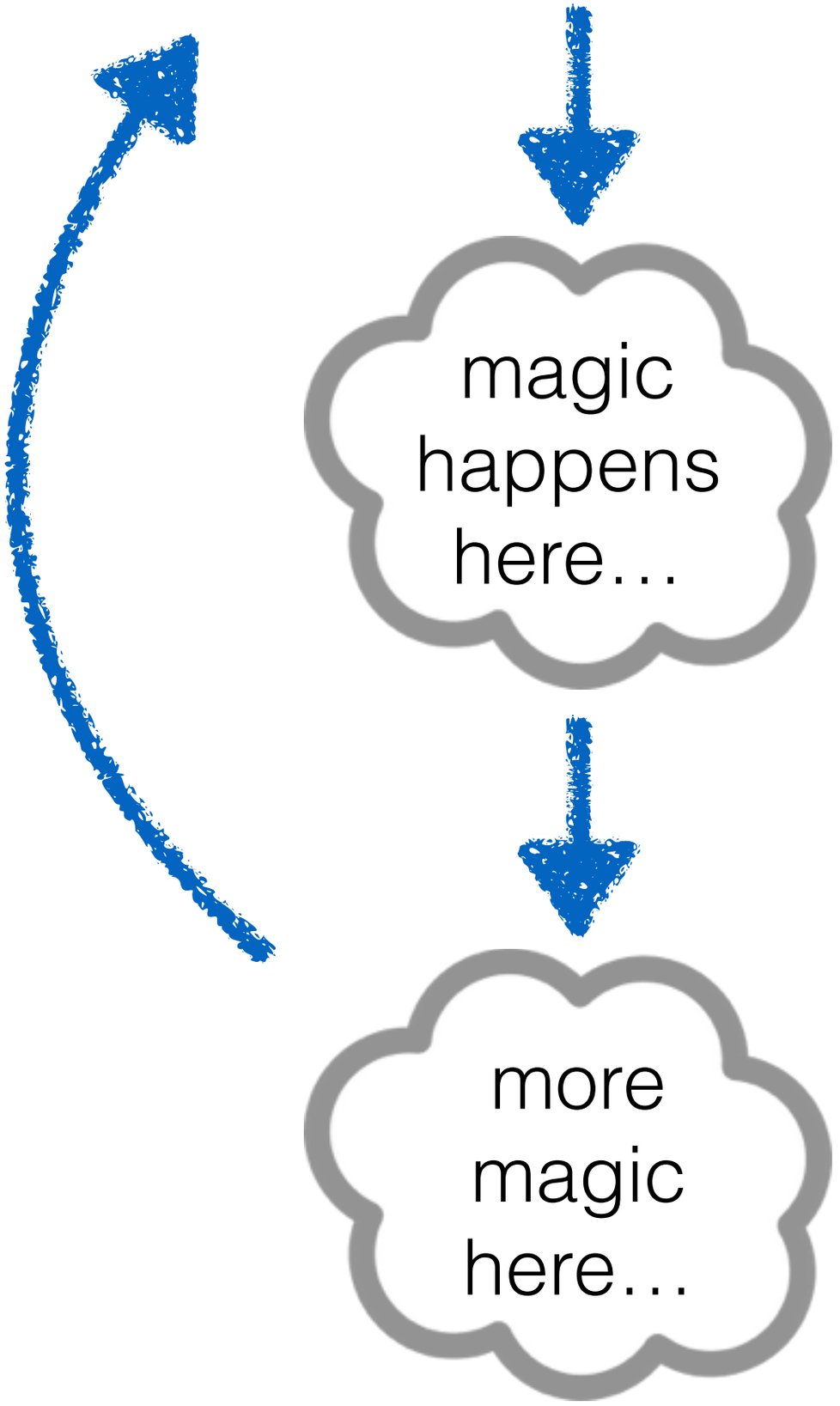


update: Msg -> Model -> Model

elm

"Twostones"

view: Model -> Html



Browser

Update function is **implied** by the view definition

A simple counter app

```
def app(m)
  div {
    button { out "+" on click m.count = m.count + 1 }
    out m.count
    button { out "-" on click m.count = m.count - 1 }
  }

model
 {count: 0}
```

The TwoStones DSL

- Syntactically partitioned:
  - View code = functions computing HTML trees
    - “on” blocks may “update” the model
- Instead of 2 functions (view/update), have 2 semantics of the same program
  - “render”: draw the HTML
  - “handle”: interpret an event into model update

$view_0$ = render(p, m_0)

... event e_0 happens ...

m_1 = handle(p, m_0, e_0)

$view_1$ = render(p, m_1)

... etc.

Two Semantics

- **Render**: takes a model and draws the UI (marks elements with event ids)

- **Handle**: takes a model and an event, “skips” all rendering, and constructs a new model based on the triggered event block.
(finds “on” blocks via event id)

Under the hood: cursors

- Variant of Huet's Zippers
- "FP equivalent of a pointer to a memory location"
- While traversing model, maintain way to "put back"
- In TwoStones, all expressions are evaluated to cursors.
- Data binding with immutable data

$$\text{eval}(e.y) = \langle v[y], \lambda x.put(v[y \mapsto x]) \rangle$$
$$\text{where eval}(e) = \langle v, put \rangle$$

$$\text{eval}(e_0[e_1]) = \langle v_0[v_1], \lambda x.put(v_0[v_1 \mapsto x]) \rangle$$
$$\text{where eval}(e_0) = \langle v_0, put \rangle$$
$$\text{and eval}(e_1) = \langle v_1, \_ \rangle$$

Benefits (?)

- No inversion of control
- No manual id management needed
- Compositional
  - 1-to-many, many-to-1, higher-order
- Time travel, undo, etc.

Limitations (?)

- Only data binding
- Only one update per event
- The model must be a “view model”
- Tricky to embed in existing languages (requires proxies, macros, …)

Demo time

twostones.js

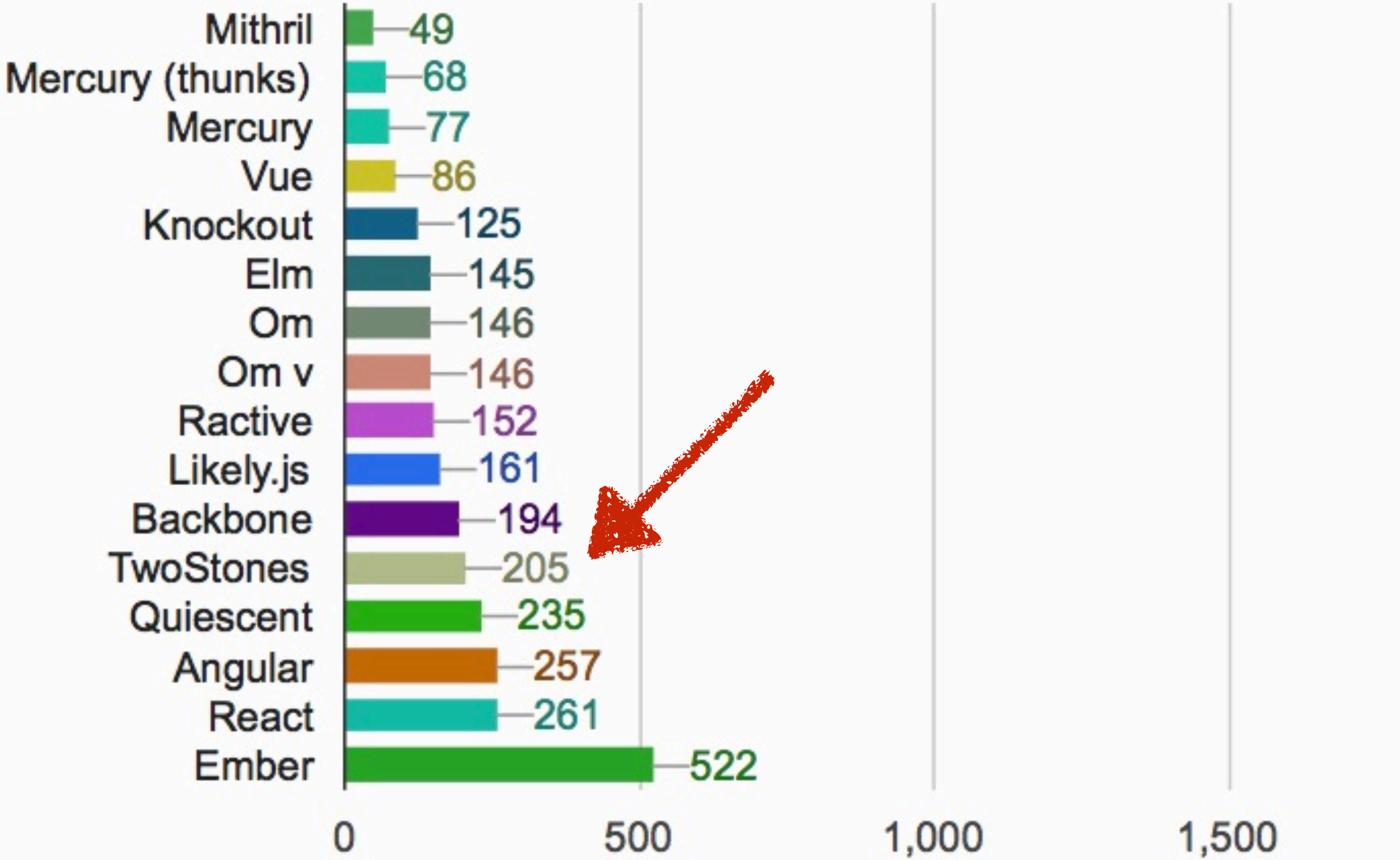
- Embedding into Javascript
- ± 300 SLOC framework code
- Closures for nesting
- Uses immutable.js as model representation
- Uses proxies to simulate cursors

```
function spinboxView(num) {
    input({type: 'number', min: 1, max: 10000, step: 100, value: num.get}, () => {
        on('input', ev => num.put(ev.value));
    });
}

function textView(num) {
    input({type: 'text', value: num.get}, () => {
        on('input', ev => num.put(ev.value));
    });
}


function addressView(title, addr) {
    h5(title.get);
    ul(() => {
        li(() => {
            out('Street: ');
            input({type: 'text', value: addr.street.get}, () => {
                on('input', ev => addr.street.put(ev.value));
            });
        });
        li(() => {
            out('Zipcode: ');
            input({type: 'text', value: addr.zipcode.get}, () => {
                on('input', ev => addr.zipcode.put(ev.value));
            });
        });
    });
}
```

TodoMVC Benchmark



Best time in milliseconds over 2 runs (lower is better)

Conclusion

- Immediate mode: ordinary control-flow
- Immutable data: time travel, undo etc.
- TwoStones: 1 language, 2 semantics
- Cursors under the hood to update app model
- Compositional UI programming with just functions