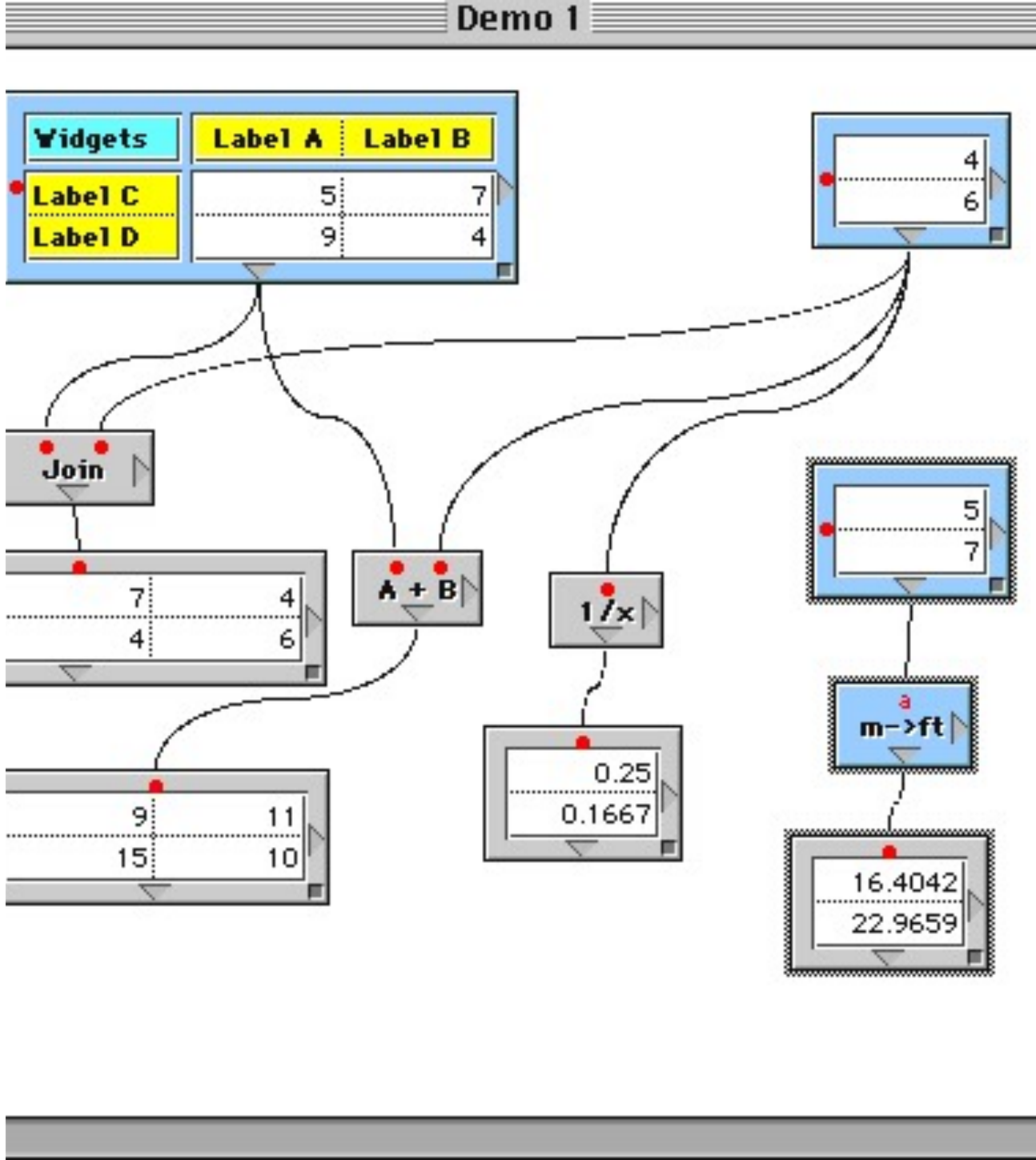
Reactivity

Signals, Effects & Derivations

What is reactivity?

$$a = b * c$$

— Where the value of **a** updates whenever the value of **b** or **c** changes



Disclosure

https://medium.com/@gethylgeorge/how-virtual-dom-and-diffing-works-in-react-6fc805f9f84e

State of the art

Easy to wrap, modularize and execute relationships associated.

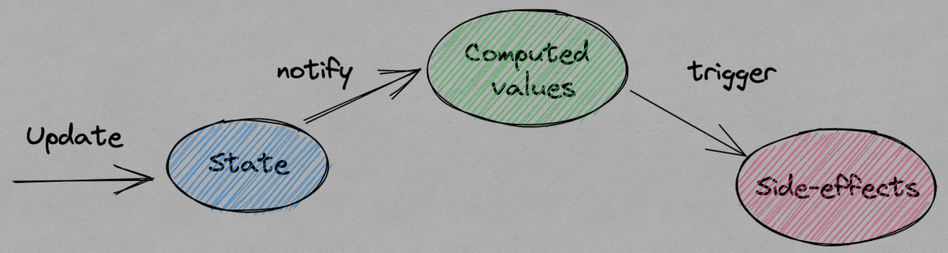
```
// Svelte 3
<script>
    export let title;

    // this will update `document.title` whenever
    // the `title` prop changes
    $: document.title = title;

    $: {
        console.log(`multiple statements can be combined`);
        console.log(`the current title is ${title}`);
    }
</script>


// Vue 3
this.count++
<script>
export default {
  // state
  data() {
    return {
      count: 0
    }
  },
  // actions
  methods: {
    increment() {
      this.count++
    }
  }
}
</script>

<!-- view -->
<template>{{ count }}</template>
```

Primitives

Signal

Derivate

Effect



Signal

Read:

- Return the value

Write:

- Update value

```
const [getter, setter] = createSignal(INITIAL_VALUE)
```

Signal

Read:

- Return the value

Write:

- Update value

```
export function createSignal(value) {
  const read = () ⇒ value;

  const write = (nextValue) ⇒ value = nextValue;

  return [read, write];
}
```

Signal

Read:

- Return the value
- Add to subscribers
- Check for current observer

Write:

- Update value
- Notify subscribers about the change

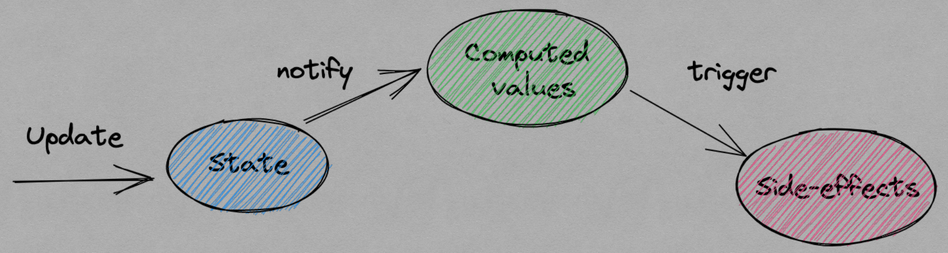
```
const context = [];

function subscribe(running, subscriptions) {
  subscriptions.add(running);
  running.dependencies.add(subscriptions);
}

export function createSignal(value) {
  const subscriptions = new Set();

  const read = () => {
    const running = context[context.length - 1];
    if (running) subscribe(running, subscriptions);
    return value;
  };

  const write = (nextValue) => {
    value = nextValue;

    for (const sub of [...subscriptions]) {
      sub.execute();
    }
  };
  return [read, write];
}
```

Primitives

Signal

Derivate

Effect



Effect

Execution:

- Execute wrapped functions when one of the values change

```
const [jsMeetups, setJsMeetups] = createSignal(55);

createEffect(() ⇒ console.log(`Meetup Js #${jsMeetups()}`))

setJsMeetups(56);
// Meetup Js #56
```

Effect

Execution:

- Execute wrapped functions when one of the values change

Dependency tracking:

- Cleanup dependencies
- Push self onto stack
- Execute provided function
- Finally pop self off the stack

```
function cleanup(running) {
  for (const dep of running.dependencies) {
    dep.delete(running);
  }
  running.dependencies.clear();
}

export function createEffect(fn) {
  const execute = () => {
    cleanup(running);
    context.push(running);
    try {
      fn();
    } finally {
      context.pop();
    }
  };

  const running = {
    execute,
    dependencies: new Set()
  };

  execute();
}
```

```
const context = [];

function subscribe(running, subscriptions) {
  subscriptions.add(running);
  running.dependencies.add(subscriptions);
}

export function createSignal(value) {
  const subscriptions = new Set();

  const read = () => {
    const running = context[context.length - 1];
    if (running) subscribe(running, subscriptions);
    return value;
  };

  const write = (nextValue) => {
    value = nextValue;

    for (const sub of [...subscriptions]) {
      sub.execute();
    }
  };
  return [read, write];
}
```

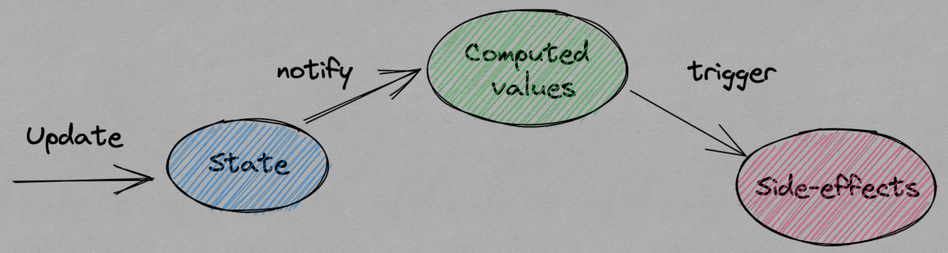
```
function cleanup(running) {
  for (const dep of running.dependencies) {
    dep.delete(running);
  }
  running.dependencies.clear();
}

export function createEffect(fn) {
  const execute = () => {
    cleanup(running);
    context.push(running);
    try {
      fn();
    } finally {
      context.pop();
    }
  };

  const running = {
    execute,
    dependencies: new Set()
  };

  execute();
}
```

Primitives

Signal

Derivate

Effect



Derivations

(Naive implementation ™)

Computations:

- Cache work from expensive computations
- One of multiple dependencies In computation

But...

- Do not prevent multiple creation effects

```
export function createMemo(fn) {
  const [s, set] = createSignal();
  createEffect(() ⇒ set(fn()));
  return s;
}
```

https://medium.com/hackernoon/becoming-fully-reactive-an-in-depth-explanation-of-mobservable-55995262a254

Thoughts

Questions?