Alexander Späth, Mtnr: 0368267

Jan van Bruegge, Mtnr: 03671594

20th of June 2018

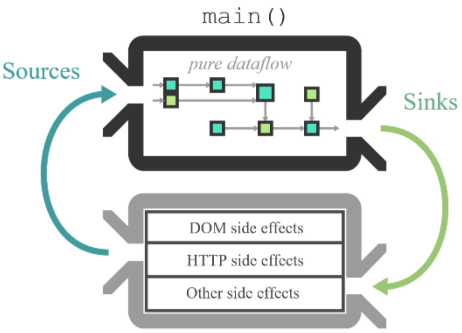
Cycle.js – a functional and reactive JavaScript framework

"Cycle is like the physicist's dream of a unified theory of everything, but for JavaScript" (Nick Johnstone)

History of Cycle:

- Focus should be on Streams: solves the cyclic dependency of streams which emerge during dialogues between the Human and the Computer

What is cycle?



- JavaScript Framework
- A function taking sources as input and returning sinks as result
- Common interface are streams
- Cycle app = function from sources to sinks
- Expandable with own side effects

Why cycle is great!!

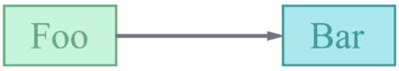
- Combines Functional Programming concepts, Reactive Programming, Observables/RxJS
  - → *Dataflow:* See your data flowing through your app
  - → *Predictable:* Functional and Reactive
  - → *Simple and Concise: many Java Script Functions, small & readable*
  - → *Composable:* functions can be reused in a larger cycle app
  - → Extensible and Testable: drivers take messages from sinks and calls imperative functions, application is a pure function

- Supports: Virtual DOM rendering, JSX, TypeScript, ReactNative (Beta), Time Traveling…

What cycle has to offer:

Reactive Programming:

Normally: whenever Foo does a network request, increment a counter in bar



Cycle: Bar listens to an event happening in Foo

➔ Bar is **reactive**: - listens to an event happening in Foo

- fully responsible for managing its own state

- Foo is unaware of the existence of the arrow originating from its network request event

Advantage:

Inversion of control (Bar is responsible for itself + functions can be hidden as privat)

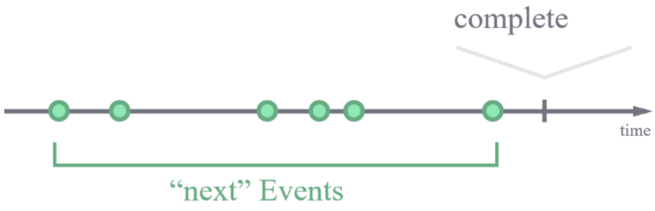
self-responsible modules which focus on their own functionality, rather than changing external state

changing external state => Seperation of Concerns !!

Cycle.js and Streams:

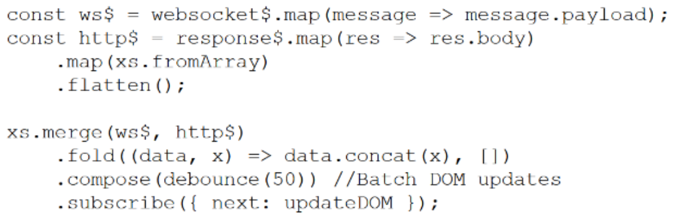
- *Xstream: an event stream, which can emit zero or more events, may or not finish, Streams can be listened to by handlers*

  - *Typical Streams contract:*



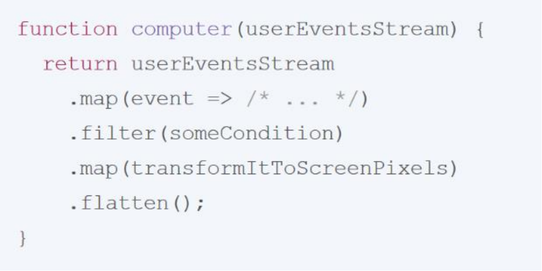
Streams in cycle

- ease to make an initial request to the server and write the data to the DOM



```
const ws$ = websocket$.map(message => message.payload);
const http$ = response$.map(res => res.body)
    .map(xs.fromArray)
    .flatten();

xs.merge(ws$, http$)
    .fold((data, x) => data.concat(x), [])
    .compose(debounce(50)) //Batch DOM updates
    .subscribe({ next: updateDOM });
```

- a short example: Computer() function takes human's output as its input and vice versa



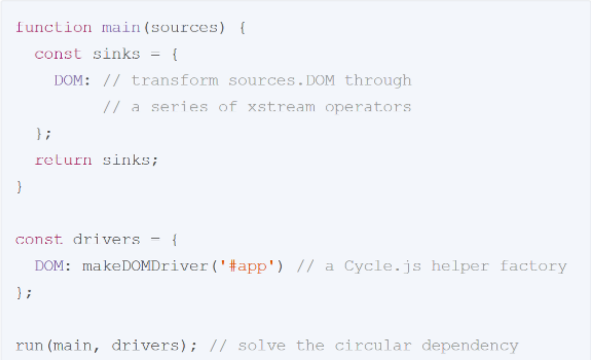
```
function computer(userEventsStream) {
  return userEventsStream
    .map(event => /* ... */)
    .filter(someCondition)
    .map(transformItToScreenPixels)
    .flatten();
}
```

JavaScript:

- Computer() can be implemented as a chain of xstreams

BUT: human () cannot be implemented as xstreams

=> driver functions needed to communicate



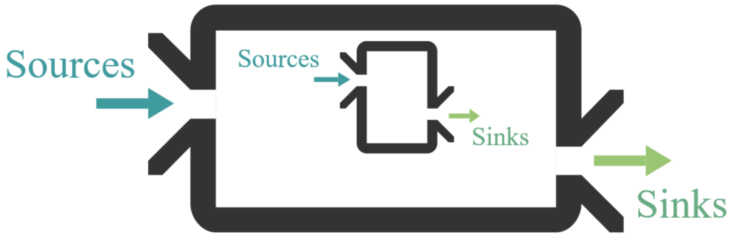
```
function main(sources) {
  const sinks = {
    DOM: // transform sources.DOM through
         // a series of xstream operators
  };
  return sinks;
}

const drivers = {
  DOM: makeDOMDriver('#app') // a Cycle.js helper factory
};

run(main, drivers); // solve the circular dependency
```

Cycle.js:

- specify main() and domDriver() and give it to Cycle.js run()

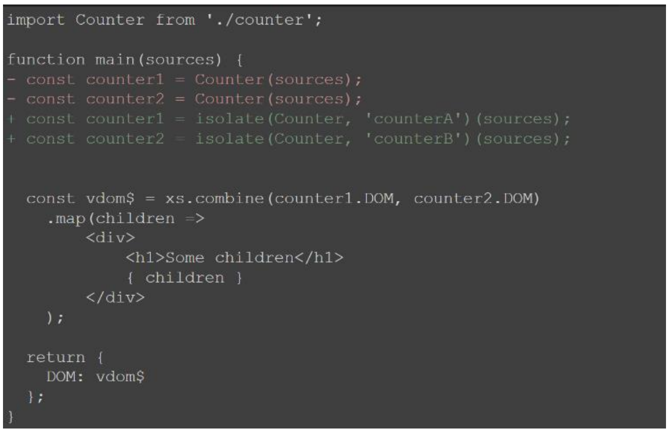
Implications of Cycle App:

- The app gets sources as input and returns sinks as output



Isolation:

- no isolation by default, must be explicit
- state or events are isolated between the different components
- e.g. clickable button



```
import Counter from './counter';

function main(sources) {
- const counter1 = Counter(sources);
- const counter2 = Counter(sources);
+ const counter1 = isolate(Counter, 'counterA')(sources);
+ const counter2 = isolate(Counter, 'counterB')(sources);


  const vdom$ = xs.combine(counter1.DOM, counter2.DOM)
    .map(children =>
        <div>
            <h1>Some children</h1>
            { children }
        </div>
    );

  return {
    DOM: vdom$
  };
}
```

Onionify

- A fractal state management tool for Cycle.js applications
- Collects all states at one certain space