REASON

DON’T PANIC

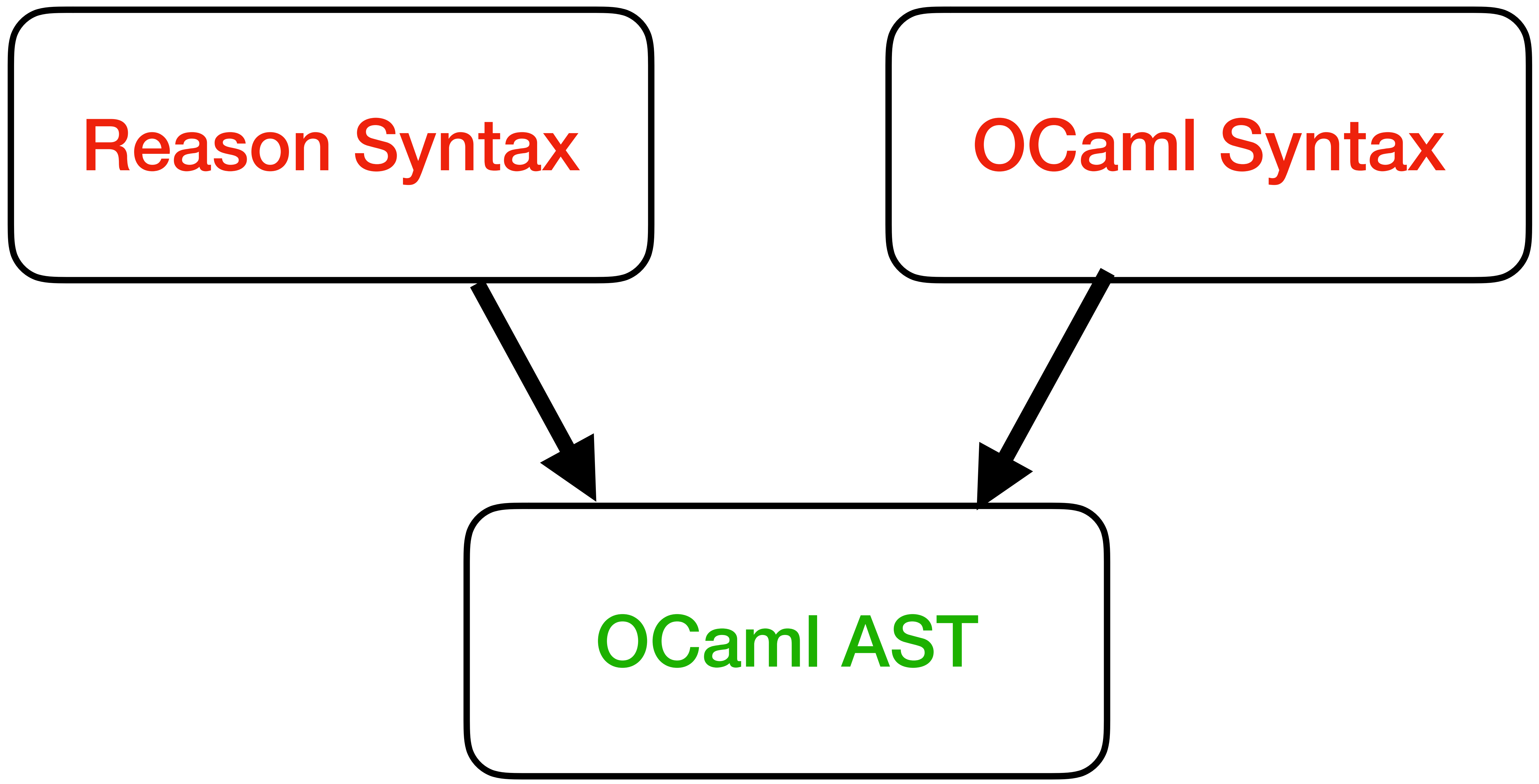
JavaScript developer friendly syntax

```
let meaningOfLife = 41 + 1;
let add = (x, y) => x + y;

add(2, 2);

add(41, 1);
```

```
let fruits = ["Apple", "Orange"];
```

```
if (true) {
  print_string("Hello World!");
};
```

OCaml semantics


Reason Syntax
OCaml Syntax
OCaml AST

Records

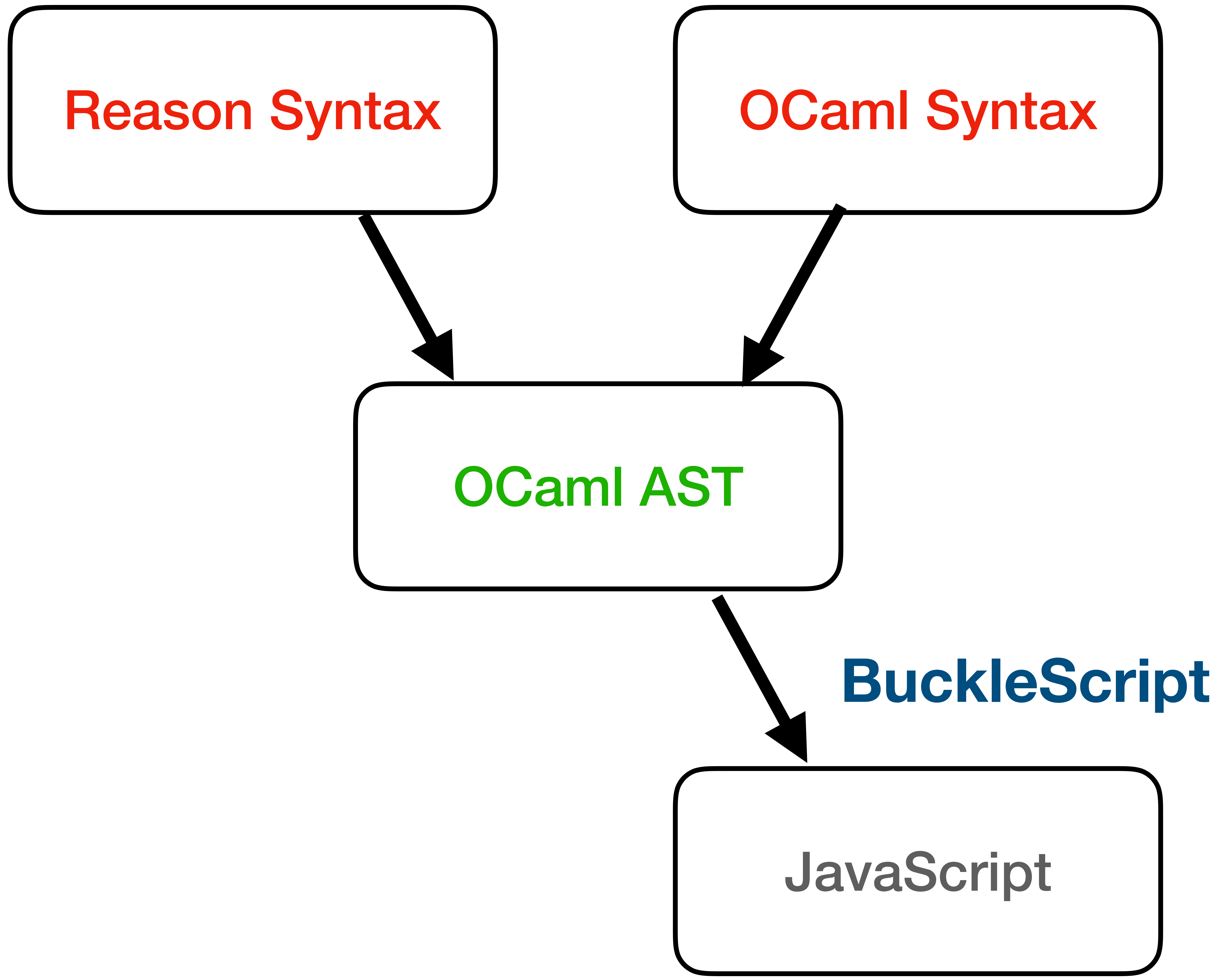
```
let jane = {name: "Jane", age: 40};
let jane = {name: "Jane", age: 40};
```

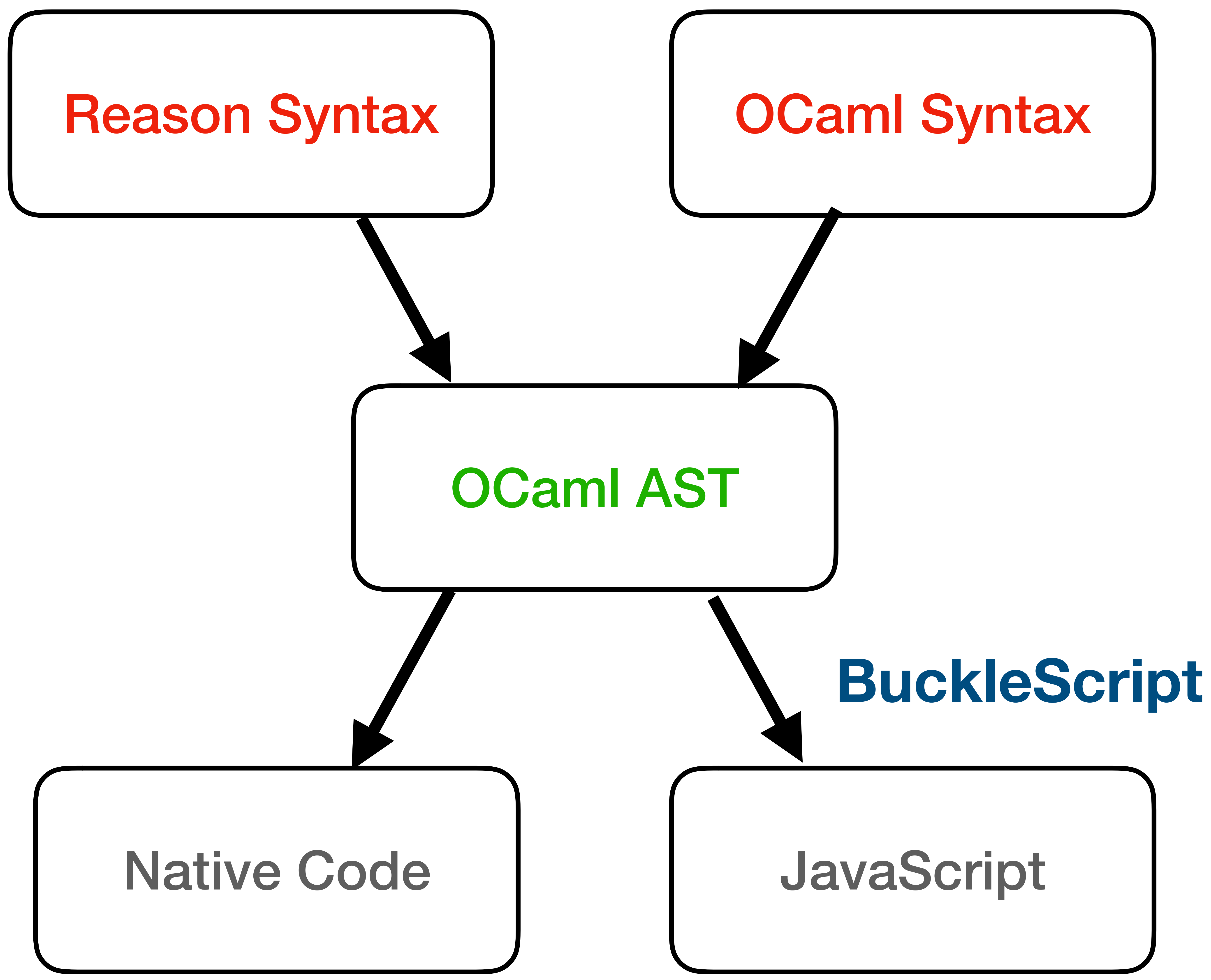
```
1 | let jane = {name: "Jane", age: 40};

The record field name can't be found.
```

```
type person = {
  name: string,
  age: int,
};

let jane = {name: "Jane", age: 40};
```

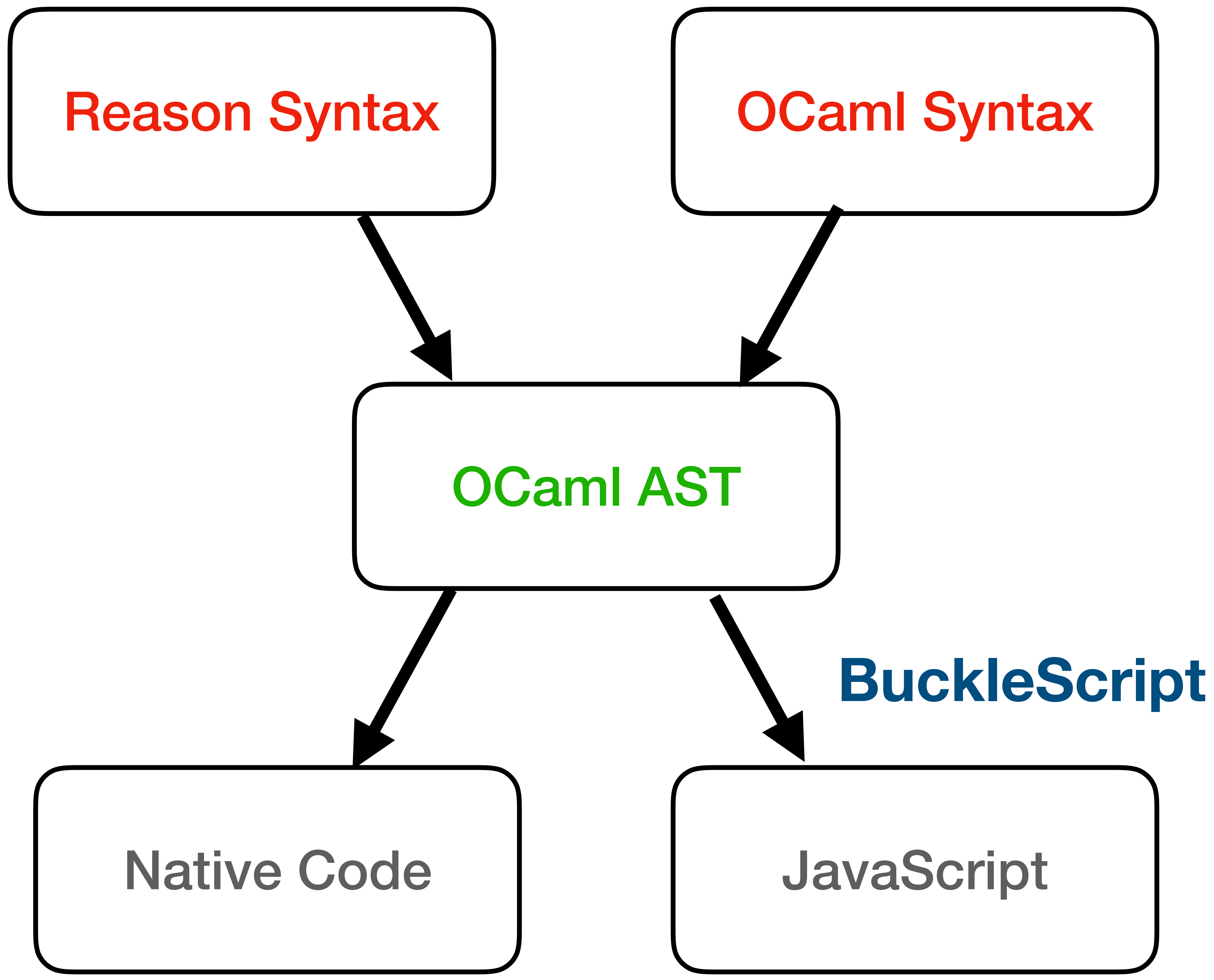
```
type person = {
  name: string,
  age: int,
};

let jane = {name: "Jane", age: 40};

let tim = {...jane, name: "Tim"};
```

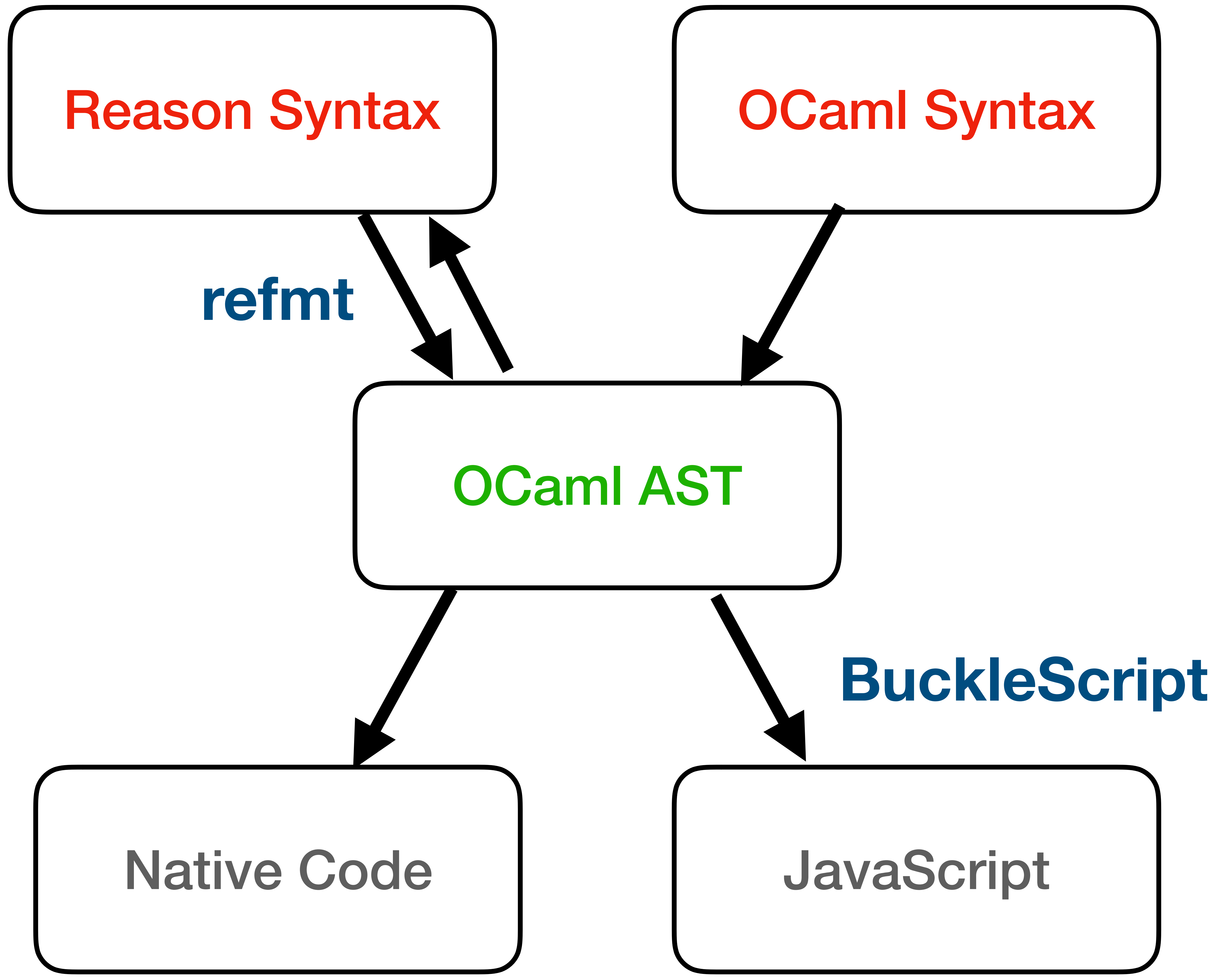
Compiles to JavaScript


Reason Syntax
OCaml Syntax
OCaml AST
BuckleScript
JavaScript


Reason Syntax
OCaml Syntax
OCaml AST
BuckleScript
Native Code
JavaScript

Fromatter


Reason Syntax
OCaml Syntax
OCaml AST
BuckleScript
Native Code
JavaScript


Reason Syntax
OCaml Syntax
refmt
OCaml AST
BuckleScript
Native Code
JavaScript

Statically Typed Language

Why yet another one?

JavaScript

```javascript
const pieces = [
  { kind: "king", color: "black", position: [3, 4] },
  { kind: "pawn", color: "black", position: [4, 2] },
  { kind: "knight", color: "white", position: [3, 3] }
];
```

TypeScript

```typescript
interface Piece {
  kind: string;
  color: string;
  position: number[];
}

const pieces = [
  { kind: "king", color: "black", position: [3, 4] },
  { kind: "pawn", color: "black", position: [4, 2] },
  { kind: "knight", color: "white", position: [3, 3] }
];

const getKinds = (pieces: Piece[]) => pieces.map(piece => piece.kind);
```

TypeScript

```typescript
interface Piece {
  kind: string;
  color: string;
  position: number[];
}

const pieces = [
  { kind: "king", color: "black", position: [3, 4] },
  { kind: "pawn", color: "black", position: [4, 2] },
  { kind: "knight", color: "white", position: [3, 3] }
];

const getKinds = (pieces: Piece[]) => pieces.map(piece => piece.kind);
```

Reason

```
type piece = {
  kind: string,
  color: string,
  position: (int, int),
};

let pieces = [
  {kind: "king", color: "black", position: (3, 4)},
  {kind: "pawn", color: "black", position: (4, 2)},
  {kind: "knight", color: "white", position: (3, 3)},
];

let getKinds = pieces => List.map(item => item.kind, pieces);
```

Variants

```
type direction =
   | Up
   | Down
   | Left
   | Right;
```

```
type direction =
  | Up
  | Down
  | Left
  | Right;

let move = Left;
```

```
type direction =
  | Up(int)
  | Down(int)
  | Left(int)
  | Right(int);

let move = Left(2);
```

```
type data = {names: list(string)};

type request =
  | Loading
  | Error(int)
  | Success(data);
```

```
type color = Black | White;

type kind = Queen | King | Rook | Bishop | Knight | Pawn;

type piece = {
  color,
  kind,
  position: (int, int),
};

let pieces = [
  {kind: King, color: Black, position: (3, 4)},
  {kind: Pawn, color: Black, position: (4, 2)},
  {kind: Knight, color: White, position: (3, 3)},
];
```

```
type color = Black | White;

type kind = Queen | King | Rook | Bishop | Knight | Pawn;

type piece = {
  color,
  kind,
  position: (int, int),
};

let pieces = [
  {kind: King, color: Black, position: (3, 4)},
  {kind: Pawn, color: Black, position: (4, 2)},
  {kind: Knight, color: White, position: (3, 3)},
];
```

```
type color = Black | White;

type kind = Queen | King | Rook | Bishop | Knight | Pawn;

type piece = {
  color,
  kind,
  position: (int, int),
};

let pieces = [
  {kind: King, color: Black, position: (3, 4)},
  {kind: Pawn, color: Black, position: (4, 2)},
  {kind: Knight, color: White, position: (3, 3)},
];
```

Pattern Matching

```
switch (<value>) {
| <pattern1> => <case1>
| <pattern2> => <case2>
| ...
};
```

```
switch (1) {
| 0 => "off"
| 1 => "on"
| _ => "off"
};
```

```
let displayText =
  switch (1) {
  | 0 => "off"
  | 1 => "on"
  | _ => "off"
  };
```

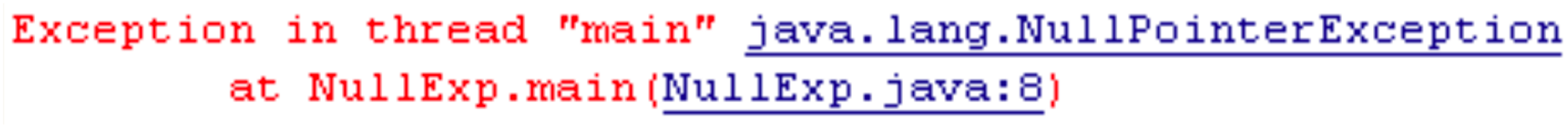
```
type data = {names: list(string)};

type request =
  | Loading
  | Error(int)
  | Success(data);

let ui =
  switch (Loading) {
  | Loading => "Loading ..."
  | Error(code) => "Something went wrong. Error: " ++ string_of_int(code)
  | Success(data) => List.fold_left((a, b) => a ++ b, "Names:", data.names)
  };
```

```
type data = {names: list(string)};

type request =
  | Loading
  | Error(int)
  | Success(data);

let ui =
  switch (Loadi[illegible] {
  | Loading => [illegible]ing ..."
  | Error(401) => "You aren’t authenticated."
  | Error(code) => "Something went wrong. Error: " ++ string_of_int(code)
  | Success(data) => List.fold_left((a, b) => a ++ b, "Names:", data.names)
  };
```

```
type data = {names: list(string)};

type request =
  | Loading
  | Error(int)
  | Success(data);

let ui =
  switch (Loadi     {
  | Loading =>      ling ..."
  | Error(401 | 402) => "You aren’t authenticated."
  | Error(code) => "Something went wrong. Error: " ++ string_of_int(code)
  | Success(data) => List.fold_left((a, b) => a ++ b, "Names:", data.names)
  };
```

“I call it my billion-dollar mistake …”

– Tony Hoare



```
Exception in thread "main" java.lang.NullPointerException
        at NullExp.main(NullExp.java:8)
```

```
Uncaught TypeError: Cannot read property 'undefined' of undefined
    at <anonymous>:3:11
```

Lesson I

Don’t implement anything just because it’s easy!

Lesson II

Null is BAD!

```
null; // doesn't exist!
```

Option

```rust
let foo = None;

let foo = Some(42);

let foo = Some([1, 2, 3]);

let foo = Some("Hello World!");
let foo = None;

let foo = Some(42);

let foo = Some([1, 2, 3]);

let foo = Some("Hello World!");

switch (foo) {
| None => "Sadly I don't know."
| Some(value) => "It's " ++ value
};
```

Functions

```
let add = (x, y) => x + y;

add(2, 2);
add(41, 1);
```

```
let name = (~firstName, ~lastName) => firstName ++ " " ++ lastName;

/* Jane Doe */
name(~firstName="Jane", ~lastName="Doe");

/* Jane Doe */
name(~lastName="Doe", ~firstName="Jane");
```

ReasonReact

Stateless Component

Greeting.re

```
let component = ReasonReact.statelessComponent("Greeting");

let make = (_children) => {
  ...component,
  render: _self => <h1>(ReasonReact.string("Hello"))</h1>,
};
```

App.re

```
ReactDOMRe.renderToElementWithId(<Greeting />, "root");
```

Props

Greeting.re

```
let component = ReasonReact.statelessComponent("Greeting");

let make = (~name, _children) => {
  ...component,
  render: _self => <h1>(ReasonReact.string(“Hello " ++ name))</h1>,
};
```

App.re

```
ReactDOMRe.renderToElementWithId(<Greeting name="Helsinki" />, "root");
```

Props

Greeting.re

```
let component [illegible]asonReact.statelessComponent("Greeting");

let make = (~name, _children) => {
  ...component,
  render: _self => <h1>(ReasonReact.string("Hello " ++ name))</h1>,
};
```

App.re

```
ReactDOMRe.renderToElementWithId(<Greeting name="Helsinki" />, "root");
```

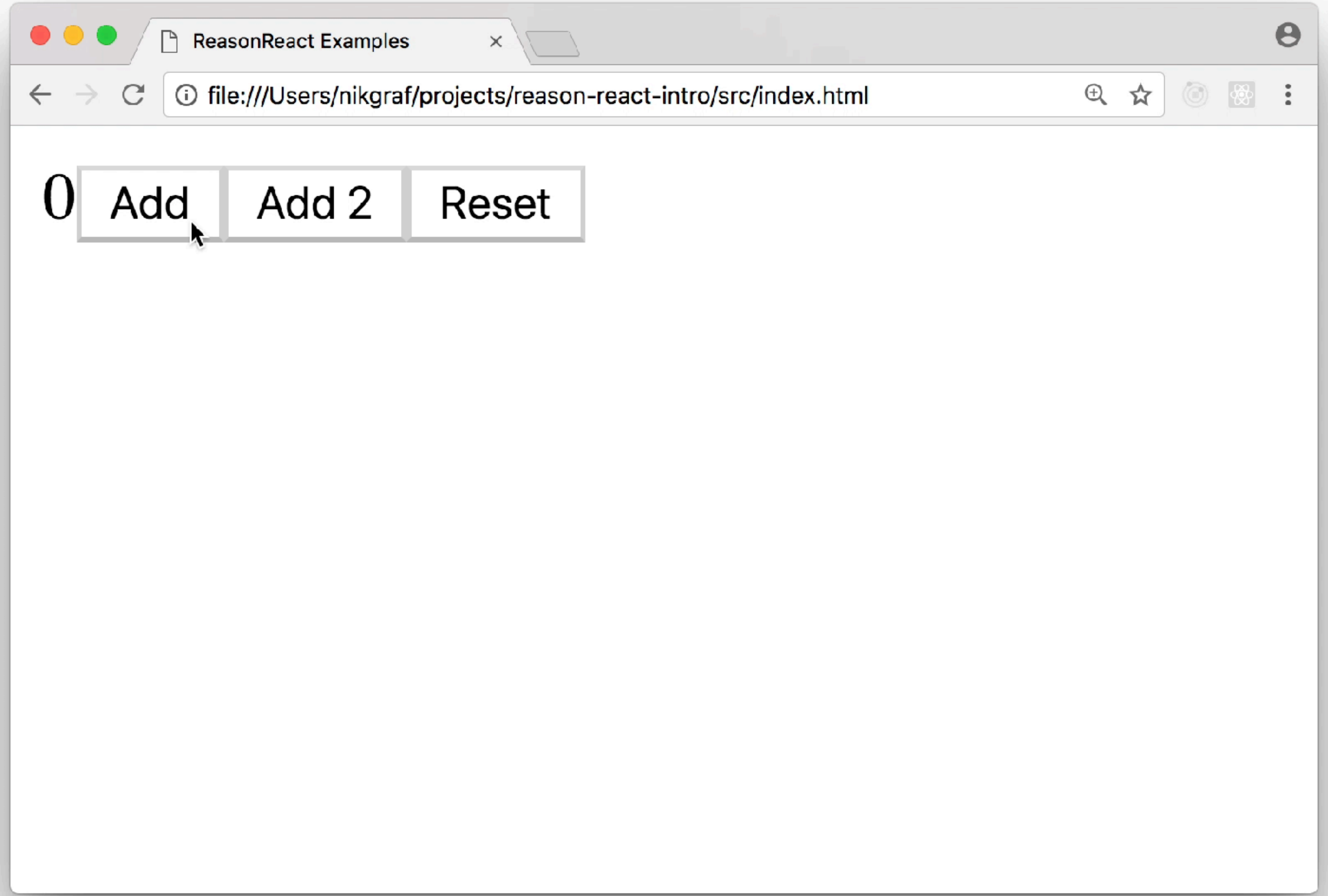
Props

Greeting.re

```
let component = ReasonReact.statelessComponent("Greeting");

let make = (~name, _children) => {
  ...component,
  render: _self => <h1>(ReasonReact.string("Hello " ++ name))</h1>,
};
```

App.re

```
ReactDOMRe.renderToElementWithId(<Greeting name="Helsinki" />, "root");
```


ReasonReact Examples
file:///Users/nikgraf/projects/reason-react-intro/src/index.html
0
Add
Add 2
Reset

```
type state = {count: int};
type state = {count: int};

type action =
  | Add(int)
  | Reset;
```

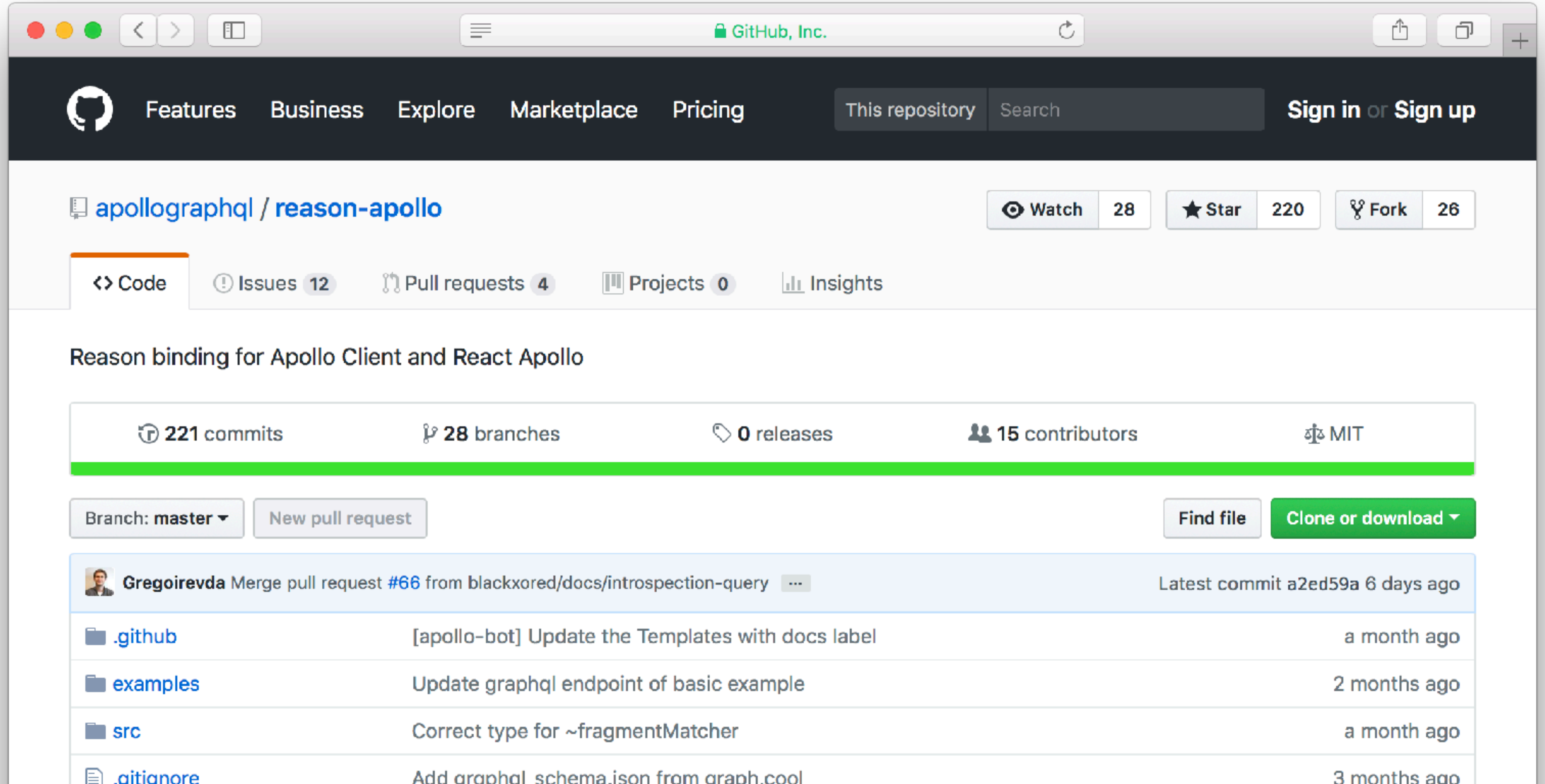
```
type state = {count: int};

type action =
  | Add(int)
  | Reset;

let s = ReasonReact.string;
```

```
type state = {count: int};

type action =
  | Add(int)
  | Reset;

let s = ReasonReact.string;

let component = ReasonReact.reducerComponent("Counter");
```

```
type state = {count: int};

type action =
  | Add(int)
  | Reset;

let s = ReasonReact.string;

let component = ReasonReact.reducerComponent("Counter");

let make = _children => {
  ...component,
  initialState: () => {count: 0},
type state = {count: int};

type action =
  | Add(int)
  | Reset;

let s = ReasonReact.string;

let component = ReasonReact.reducerComponent("Counter");

let make = _children => {
  ...component,
  initialState: () => {count: 0},
  reducer: (action, state) =>
    switch (action) {
    | Add(value) => ReasonReact.Update({count: state.count + value})
    | Reset => ReasonReact.Update({count: 0})
    },
type state = {count: int};

type action =
  | Add(int)
  | Reset;

let s = ReasonReact.string;

let component = ReasonReact.reducerComponent("Counter");

let make = _children => {
  ...component,
  initialState: () => {count: 0},
  reducer: (action, state) =>
    switch (action) {
    | Add(value) => ReasonReact.Update({count: state.count + value})
    | Reset => ReasonReact.Update({count: 0})
    },
  render: self =>
    <div>
      (s(string_of_int(self.state.count)))
      <button onClick={_event => self.send(Add(1))}> (s(“Add")) </button>
```

```
let s = ReasonReact.string;

let component = ReasonReact.reducerComponent("Counter");

let make = _children => {
  ...component,
  initialState: () => {count: 0},
  reducer: (action, state) =>
    switch (action) {
    | Add(value) => ReasonReact.Update({count: state.count + value})
    | Reset => ReasonReact.Update({count: 0})
    },
  render: self =>
    <div>
      (s(string_of_int(self.state.count)))
      <button onClick=(_event => self.send(Add(1)))> (s(“Add")) </button>
      <button onClick=(_event => self.send(Add(2)))> (s("Add 2")) </button>
      <button onClick=(_event => self.send(Reset))> (s("Reset")) </button>
    </div>,
};
```

Manage your State with GraphQL



apollographql / reason-apollo

Reason binding for Apollo Client and React Apollo

221 commits | 28 branches | 0 releases | 15 contributors | MIT

Gregoirevda Merge pull request #66 from blackxored/docs/introspection-query ... Latest commit a2ed59a 6 days ago

.github	[apollo-bot] Update the Templates with docs label	a month ago
examples	Update graphql endpoint of basic example	2 months ago
src	Correct type for ~fragmentMatcher	a month ago
.gitignore	Add graphql_schema.json from graph.cool	3 months ago

Interop with JavaScript

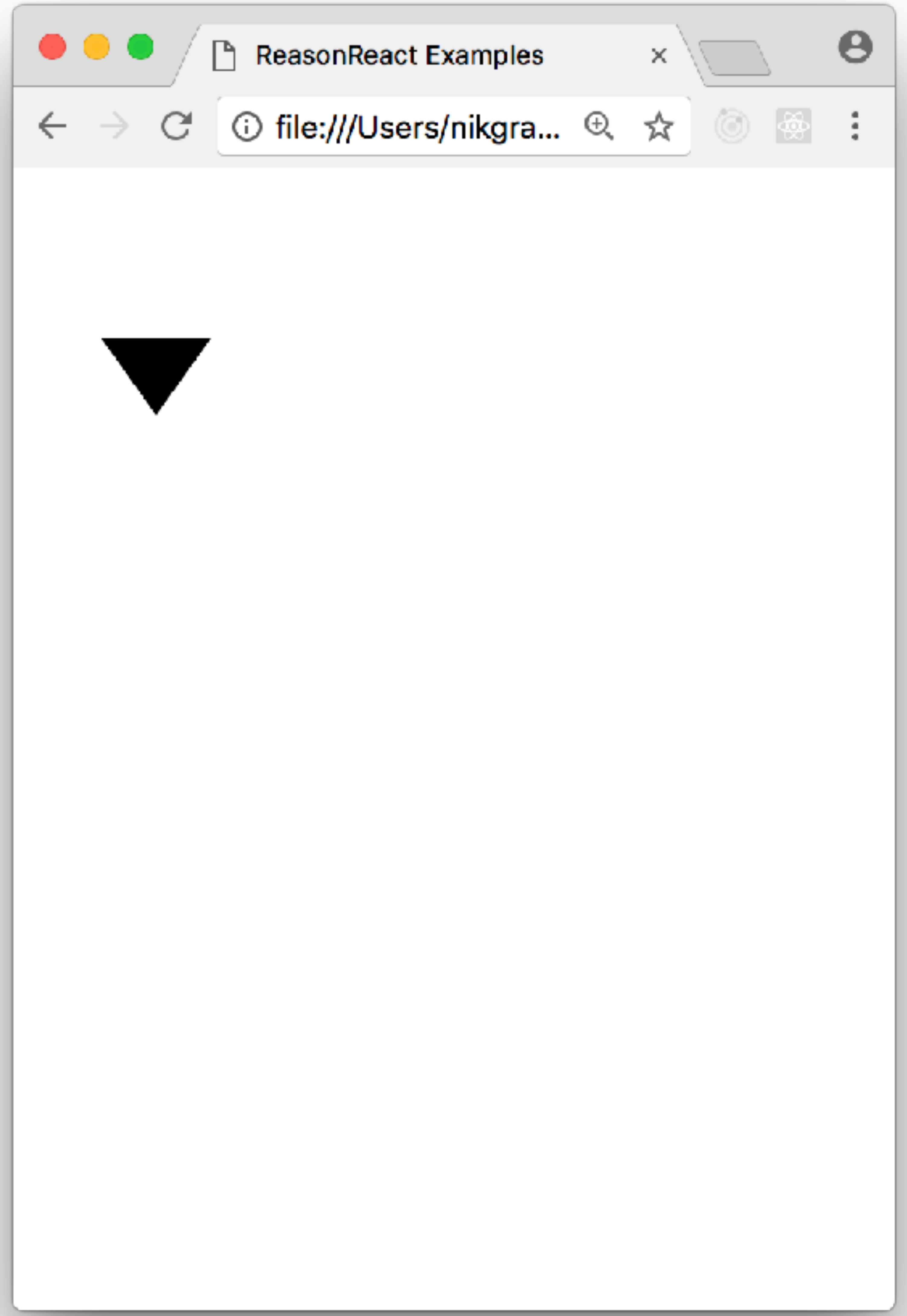
BuckleScript allows us to write bindings

ReasonReact

- wrapJsForReason
- wrapReasonForJs

```reason
[@bs.module "rebass"] external jsArrow : ReasonReact.reactClass = "Arrow";

let make = (~direction: string, children) =>
  ReasonReact.wrapJsForReason(
    ~reactClass=jsArrow,
    ~props={"direction": direction},
    children,
  );
```

```
<Arrow direction="down" />
```

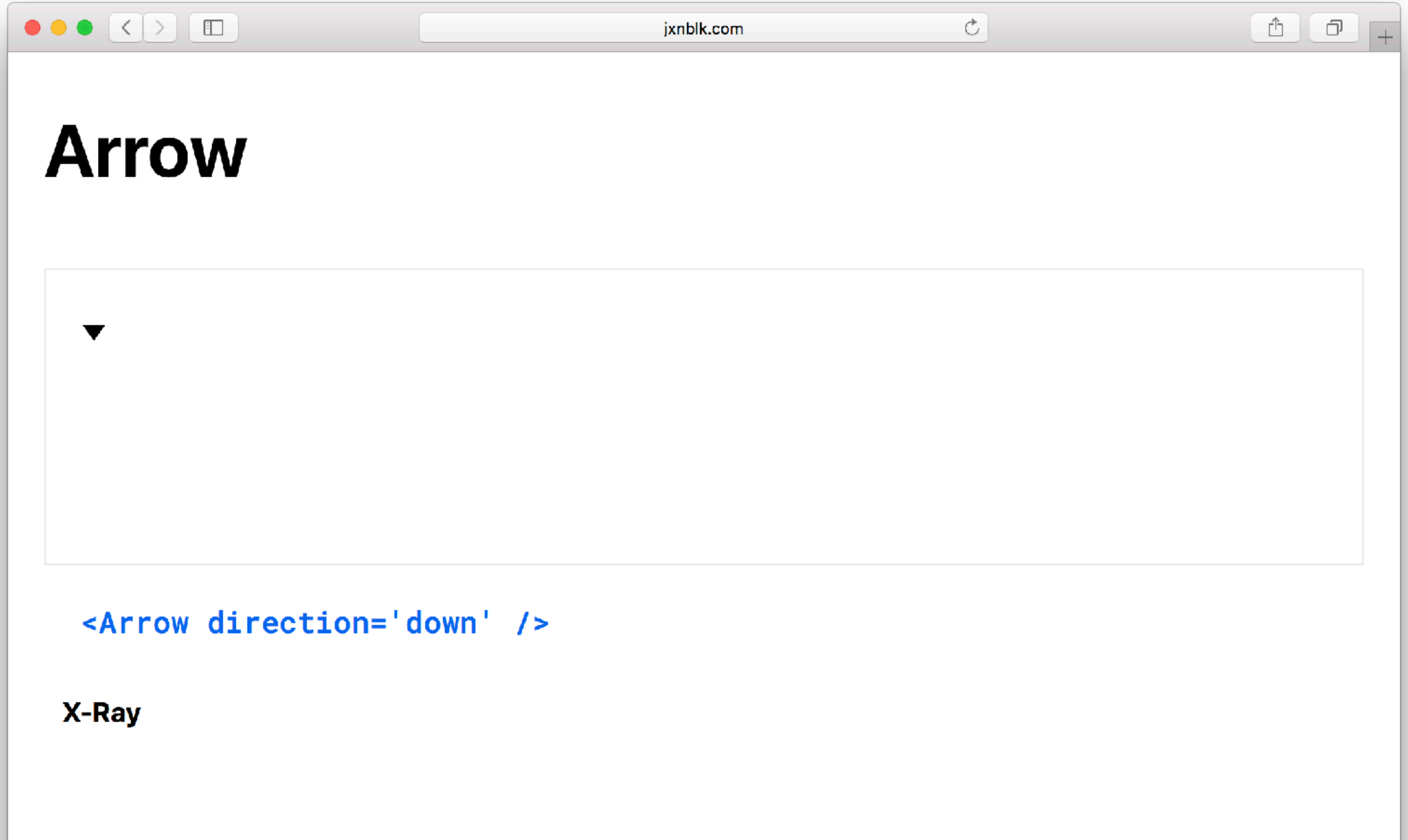

ReasonReact Examples
file:///Users/nikgra...

```
<Arrow direction="left" />
```

ReasonReact Examples
file:///Users/nikgra...

```
<Arrow direction="notRight"/>
```

ReasonReact Examples
file:///Users/nikgra...



Arrow

▼

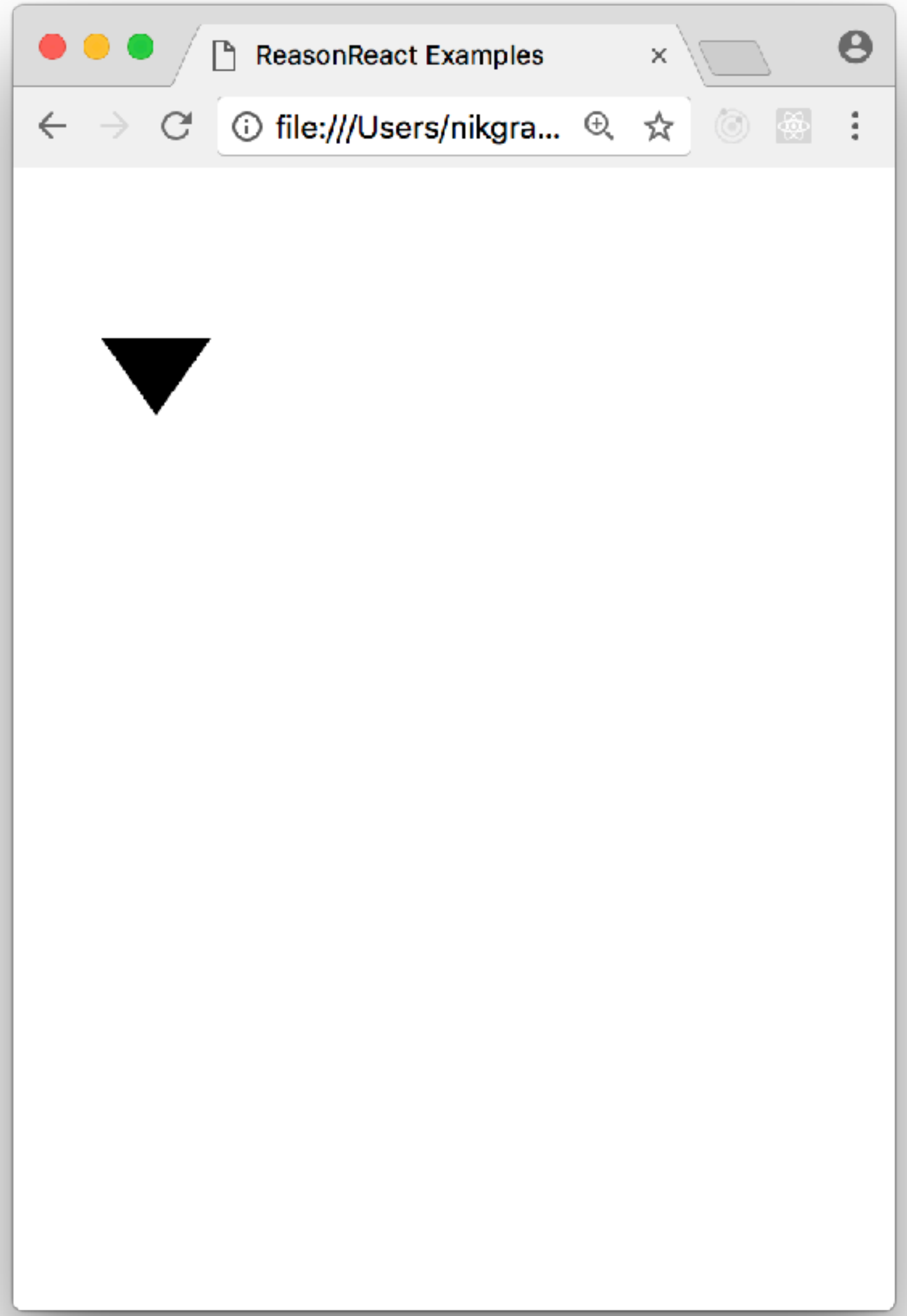
```
<Arrow direction='down' />
```

X-Ray

Variants to the rescue!

```
[@bs.module "rebass"] external jsArrow : ReasonReact.reactClass = "Arrow";

type direction =
  | Up
  | Down;

let make = (~direction, children) => {
  let directionString =
    switch (direction) {
    | Up => "up"
    | Down => "down"
    };
  ReasonReact.wrapJsForReason(
    ~reactClass=jsArrow,
    ~props={"direction": directionString},
    children,
  );
};
```

```
<Arrow direction=Arrow.Down />;
```


ReasonReact Examples
file:///Users/nikgra...

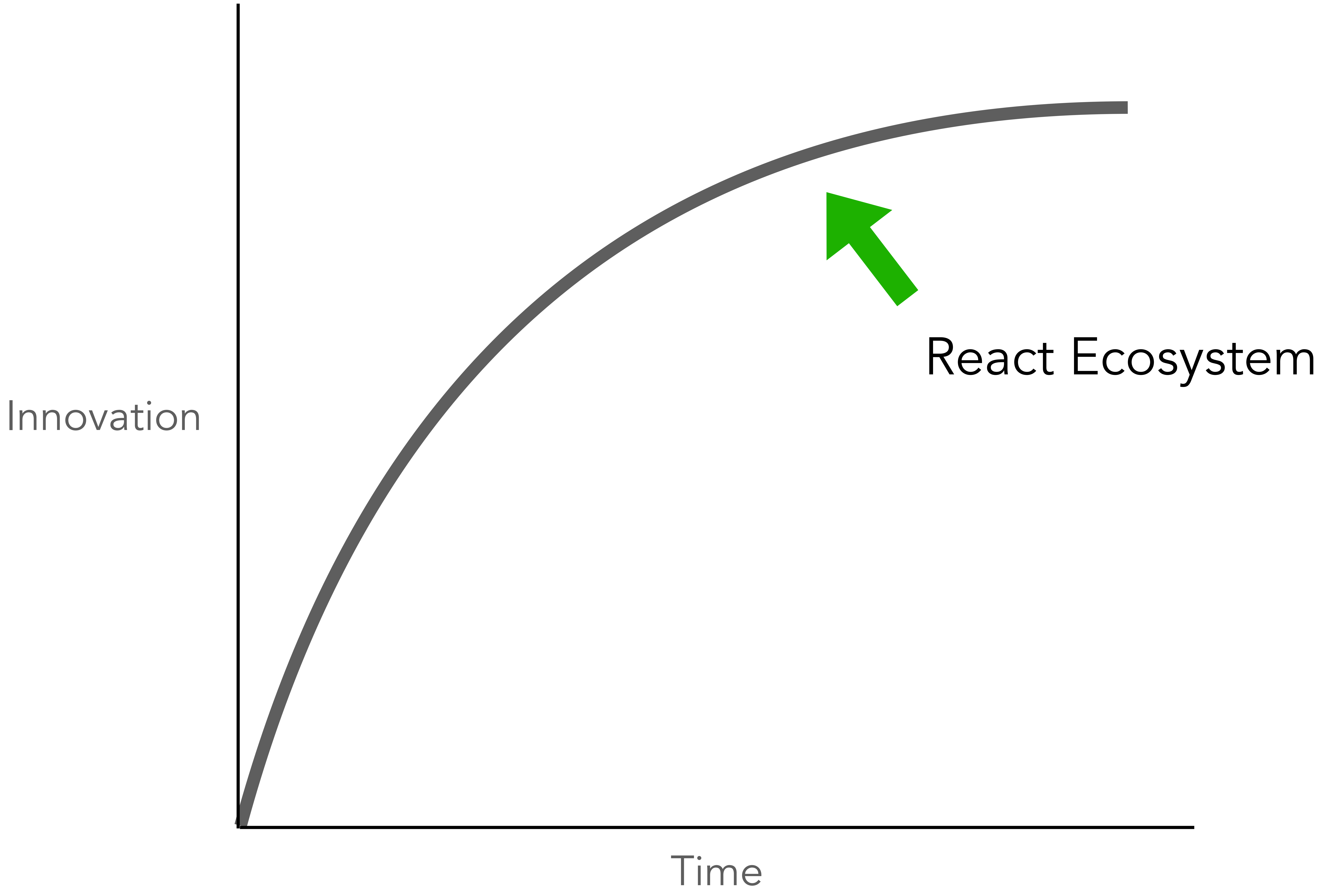
```
<Arrow direction=Arrow.Left />;
```

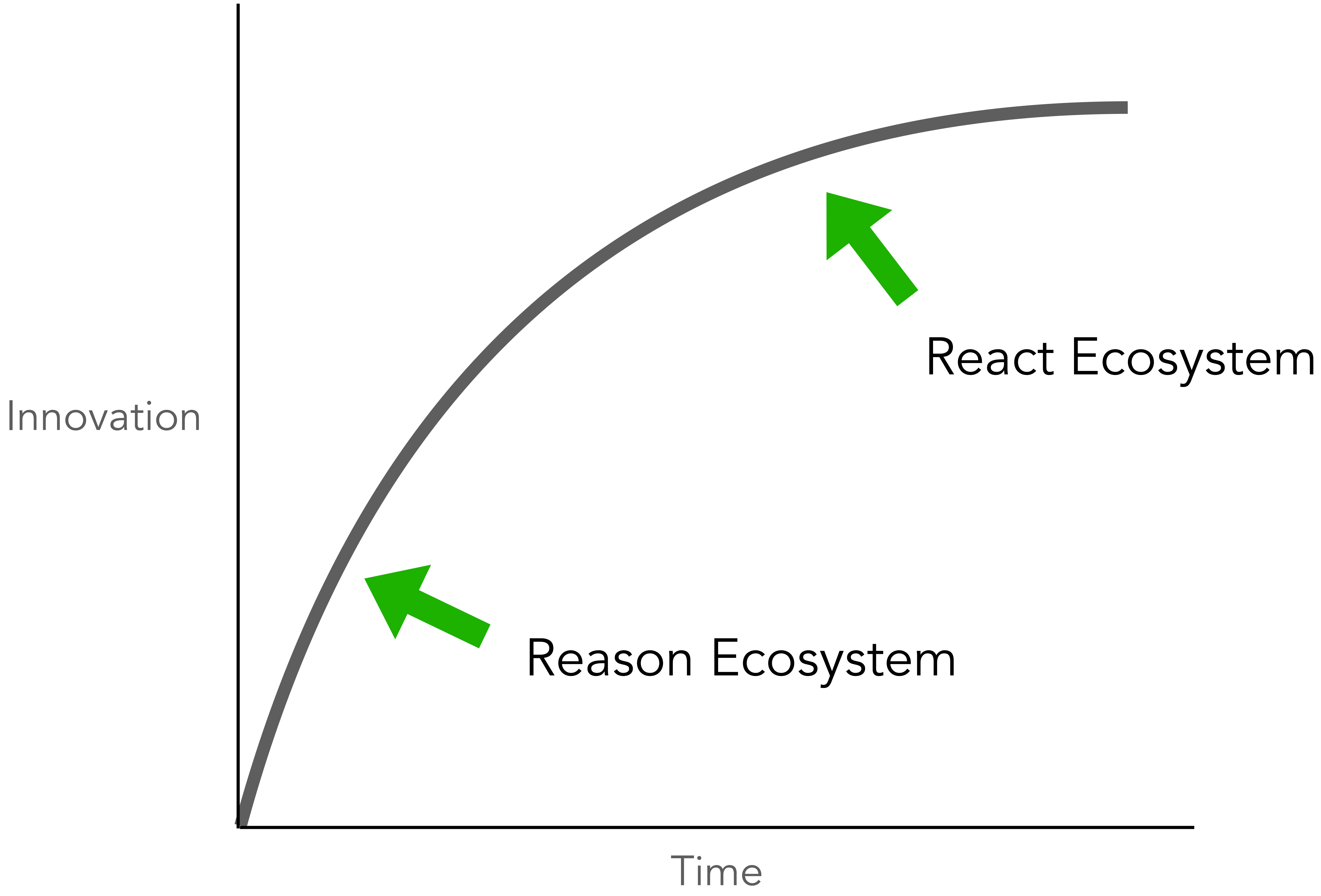
```
<Arrow direction=Arrow.Left />;
```

```
3 | let make = _children => {
4 |   ...component,
5 |   render: _self => <div> <Arrow direction=Arrow.Left /> </div>,
6 | };

The variant constructor Arrow.Left can't be found.
```

So what now?

Don’t be that person


React Ecosystem
Innovation
Time


Innovation
React Ecosystem
Reason Ecosystem
Time

REASONCONF

11–13 May 2018

World’s first Reason conference for web-developers & OCaml enthusiasts

Vienna, Austria

Day 1: Workshop

Day 2: Talks

Day 3: Hackathon

The End