

REASON

Schedule

- Intro
- Data structures
- ReasonReact

**DON'T
PANIC**

JavaScript developer
friendly syntax

```
let meaningOfLife = 41 + 1;
```

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

```
add(2, 2);
```

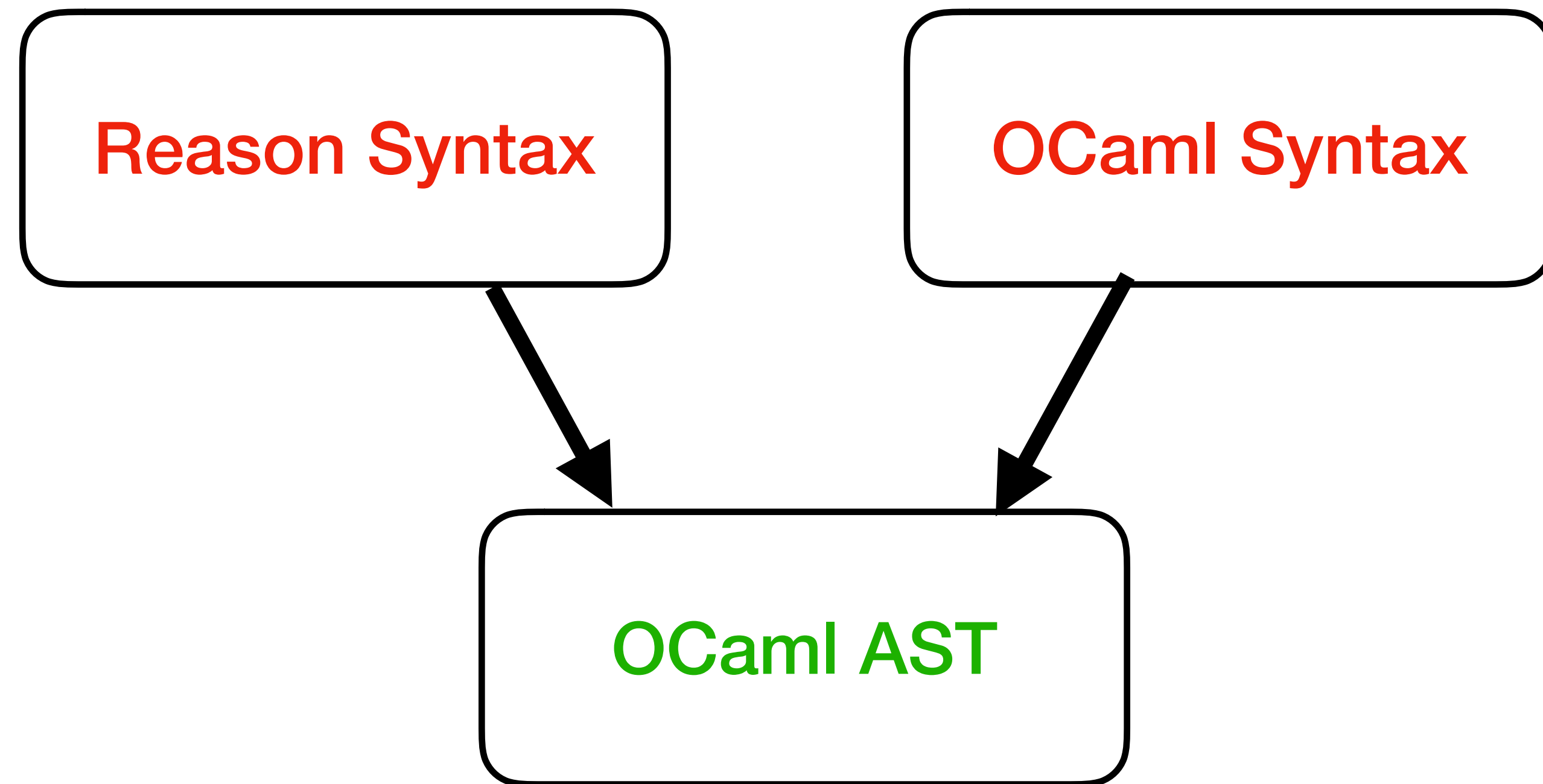
```
add(41, 1);
```

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

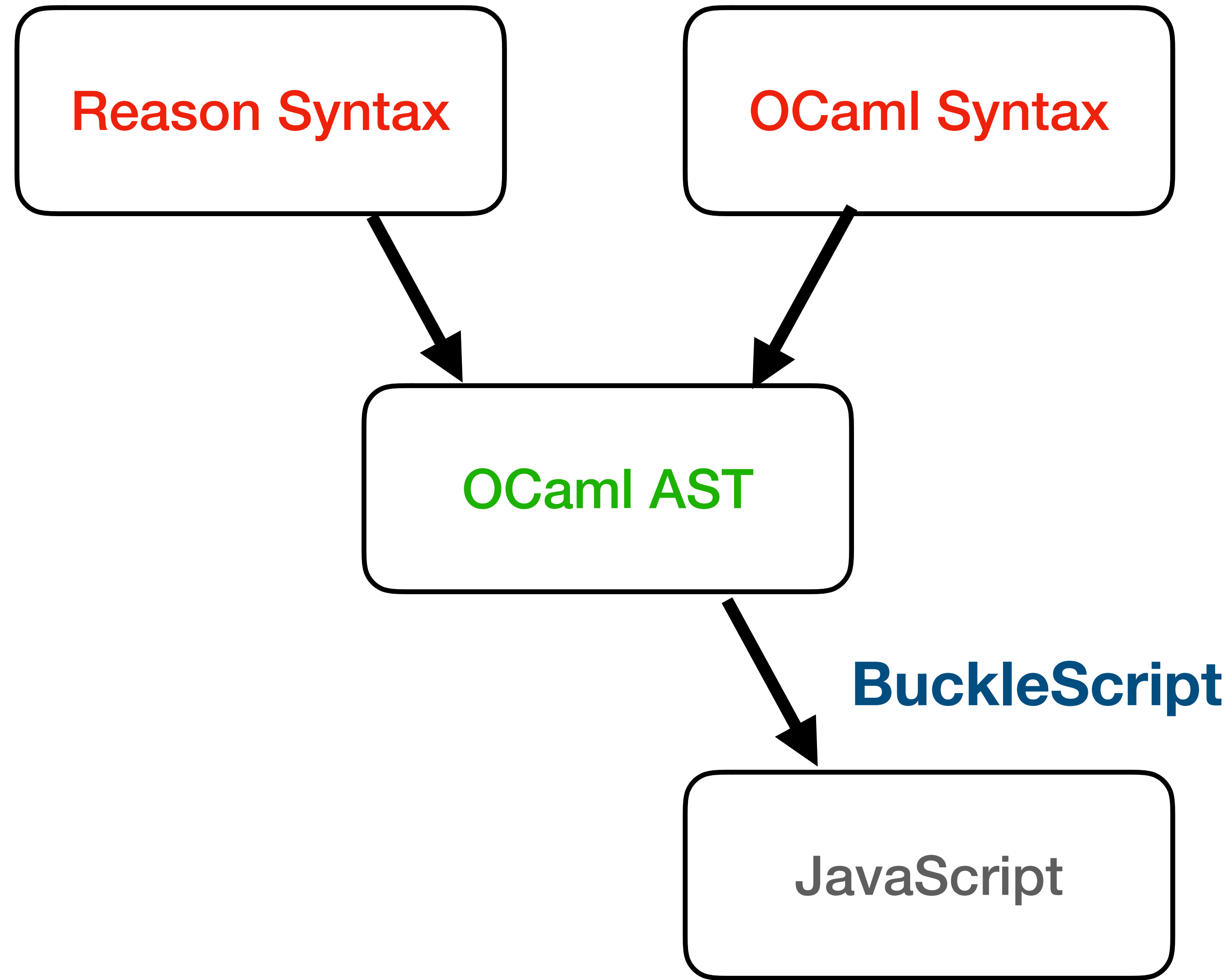


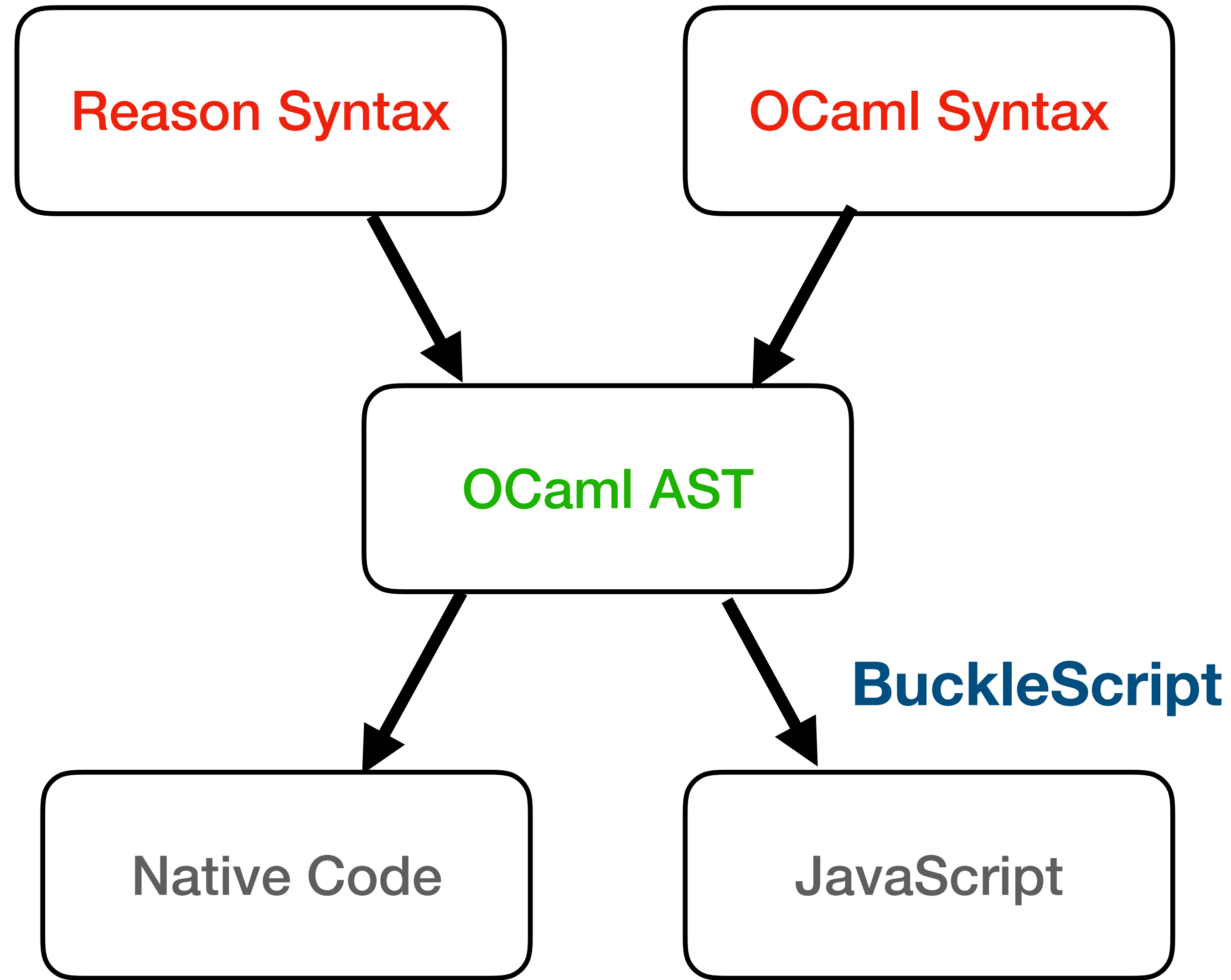
```
if (true) {  
    print_string("Hello World!");  
};
```

OCaml semantics

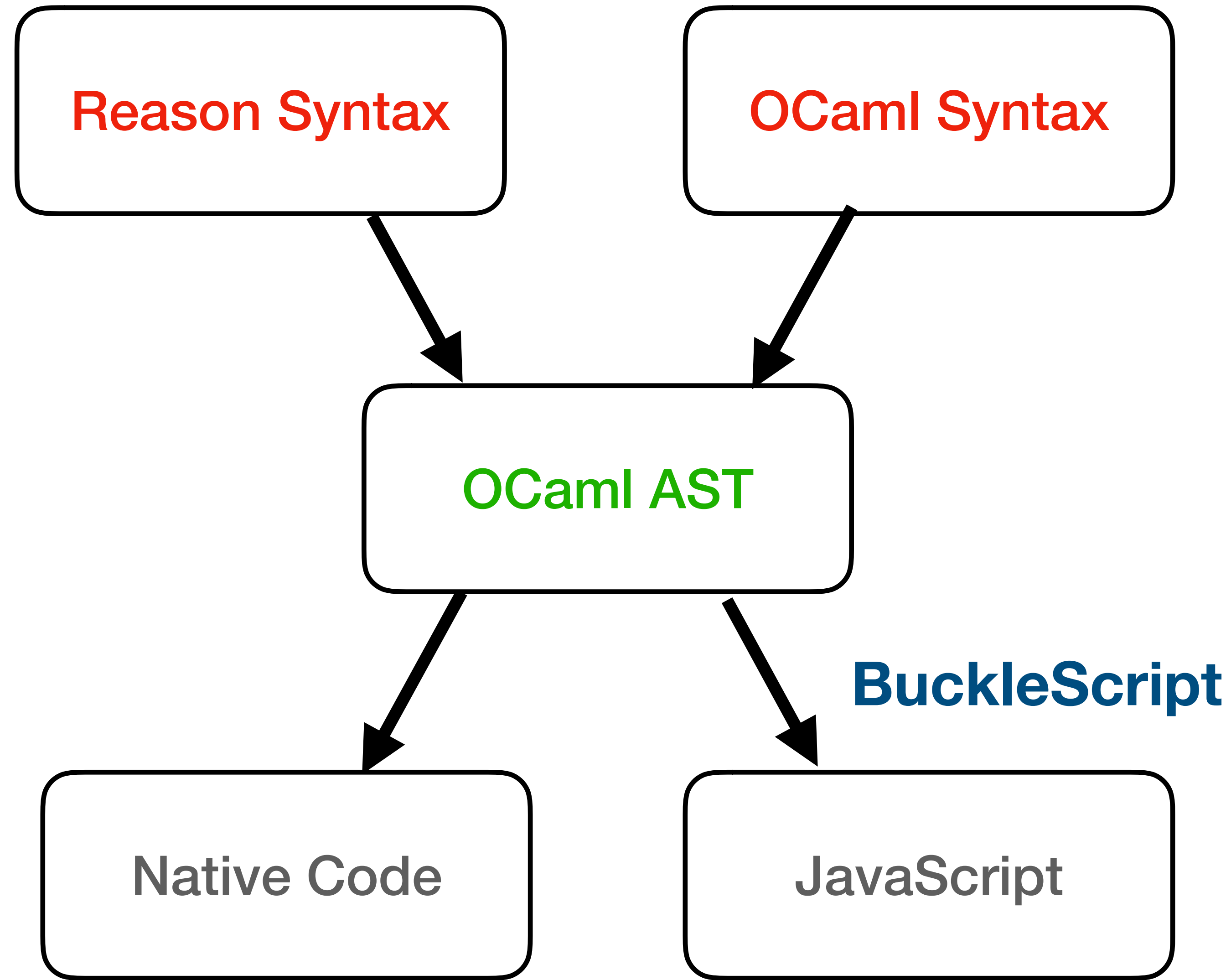


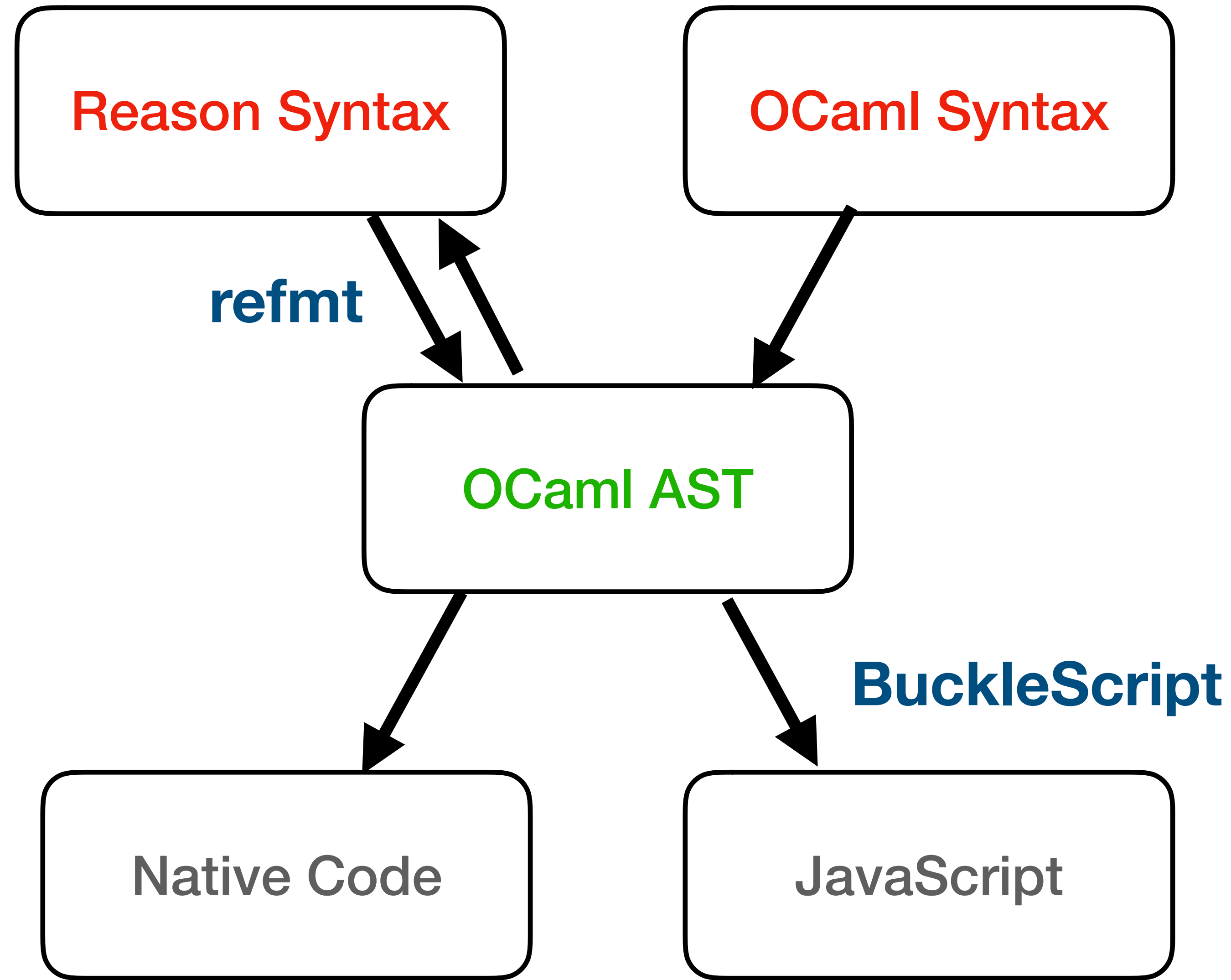
Compiles to JavaScript





Formatter





But why?

Statically Typed Language

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"};
```


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);
```

JavaScript

```
const state = {  
  loading: true,  
  error: false,  
  data: {},  
}
```

```
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)},  
];
```

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 (Loading) => {  
    | Loading => "Loading ..."  
    | 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 (Loading) => {  
    | Loading => "Loading ..."  
    | 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

```
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");
```

BuckleScript



```
$ npm install -g bs-platform
```

Major differences between the OCaml <--> **Reason** ecosystem:

- **Reason** is commonly used in tandem with BuckleScript
- **Reason** is focusing the **npm / yarn** workflow
- **Reason** tries to **unify tools** and optimizes them for the **JS** use-case
- **Reason (as a syntax)** can be used with every major OCaml build tool

Major differences between the **OCaml** <--> Reason ecosystem:

- **OCaml** is using the **opam** package manager & dune (prev. jbuilder) build-tool to target native binaries
- **OCaml** native binaries are **insanely fast** and **efficient**

Goals of the Reason Project:

- Offer an **alternative syntax**, which makes it easier for JavaScript developers to get into OCaml
- Leverage the **OCaml type system** to build **type-safe webapps**
- **Modernize docs** of the OCaml ecosystem during the process

Note: It **doesn't want to replace OCaml nor JavaScript** (all languages even can be mixed inside a project)

WARNING

- Reason is something you are probably not used to
- Some type errors will be confusing
- Your editor will eventually get freaky sometimes
- As soon as you get over the first big hurdles and understand the basic concepts, it will gradually be more enlightening
- Don't overthink it! Go slow and ask questions if something is unclear

```
/* unit */  
let nothing = ();  
  
let str = "Some string";  
  
/* Int is its own data type */  
let someInt = 1;  
  
/* The dot signals a floating point number */  
let someFloat = 1.;  
  
/* Yeah, Reason also supports single characters */  
let someChar = 'c';  
  
/* List is immutable, good for small number of entries */  
let someList = [1, 2, 3];  
  
/* Arrays are quicker and mutable... good for JS interop */  
let someArray = [|1, 2, 3|];  
  
/* Tuples always contain a strict fixed number of elements */  
let someTuple = (1, 2);  
  
/* You can annotate variables as well */  
let someAnnotated: string = "";  
  
/* Some record (needs type definition of given record) */  
let someRecord = {test: "test", good: true };
```

```
/* This is a variant type `color` with 3 tags */  
type color = Red | Green | Blue;
```

```
/* Tags don't have any concrete value.  
Note that we never have to annotate `myColor` */  
let myColor = Red;
```

```
/* You can define type constructors, which  
can attach data to provided Tags */  
type distance = int;  
type movement =  
    | Up(distance)  
    | Down(distance)  
    | Left(distance)  
    | Right(distance);
```

```
/* When we want to use `Up`, we need to provide a value */  
let myMove = Up(10);
```

The End