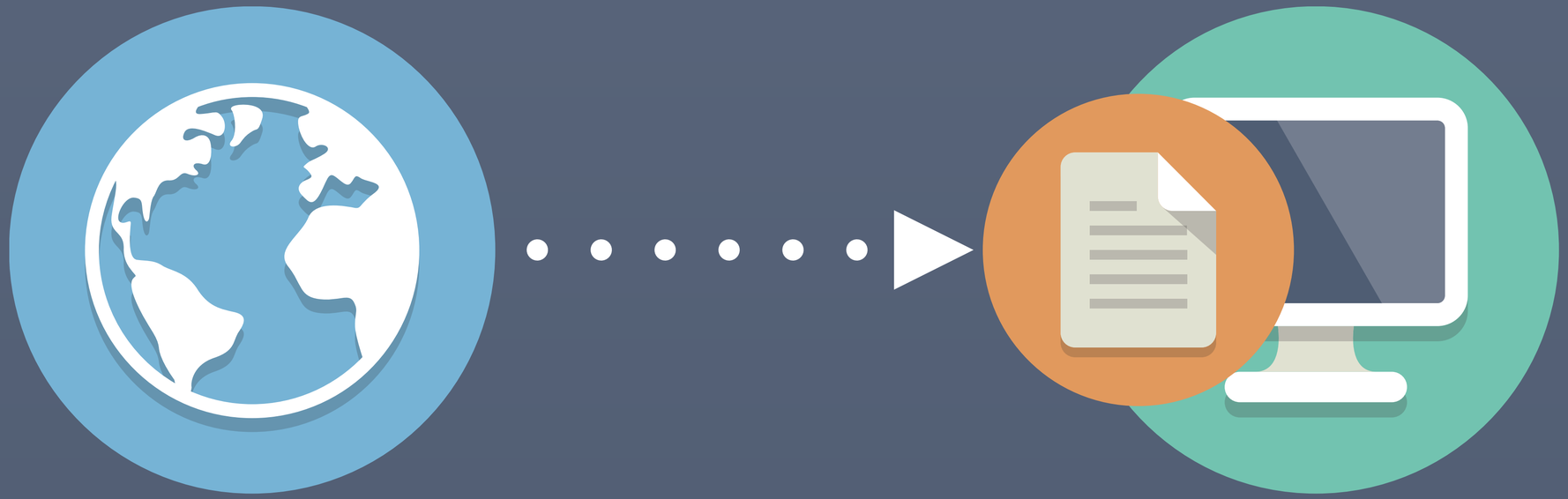
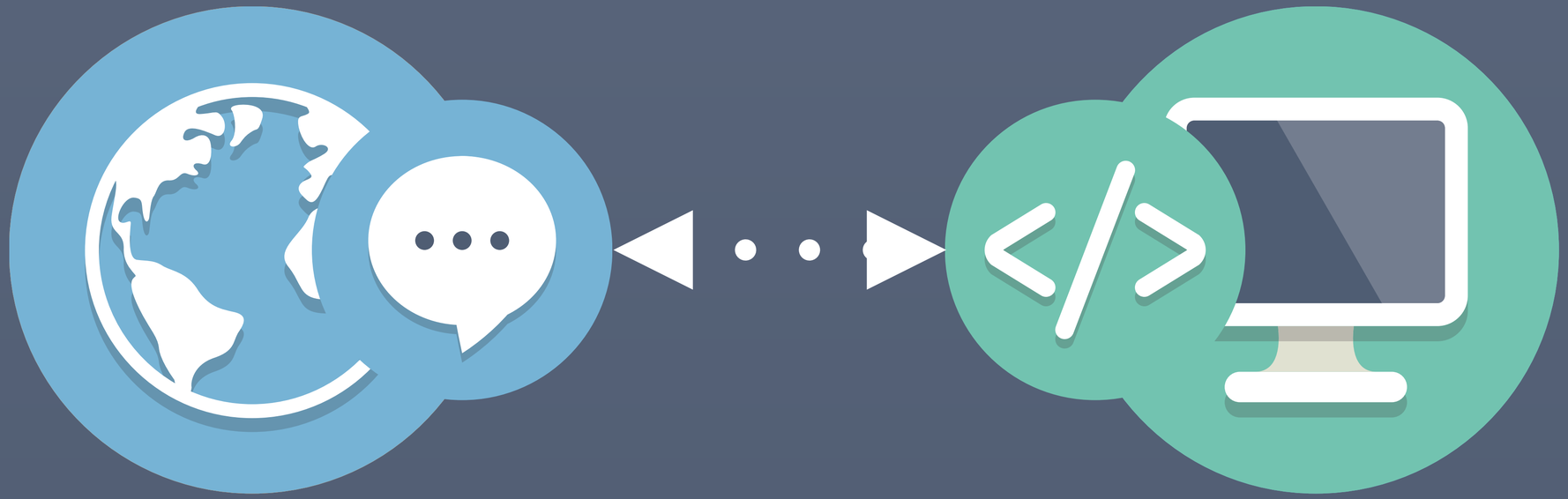
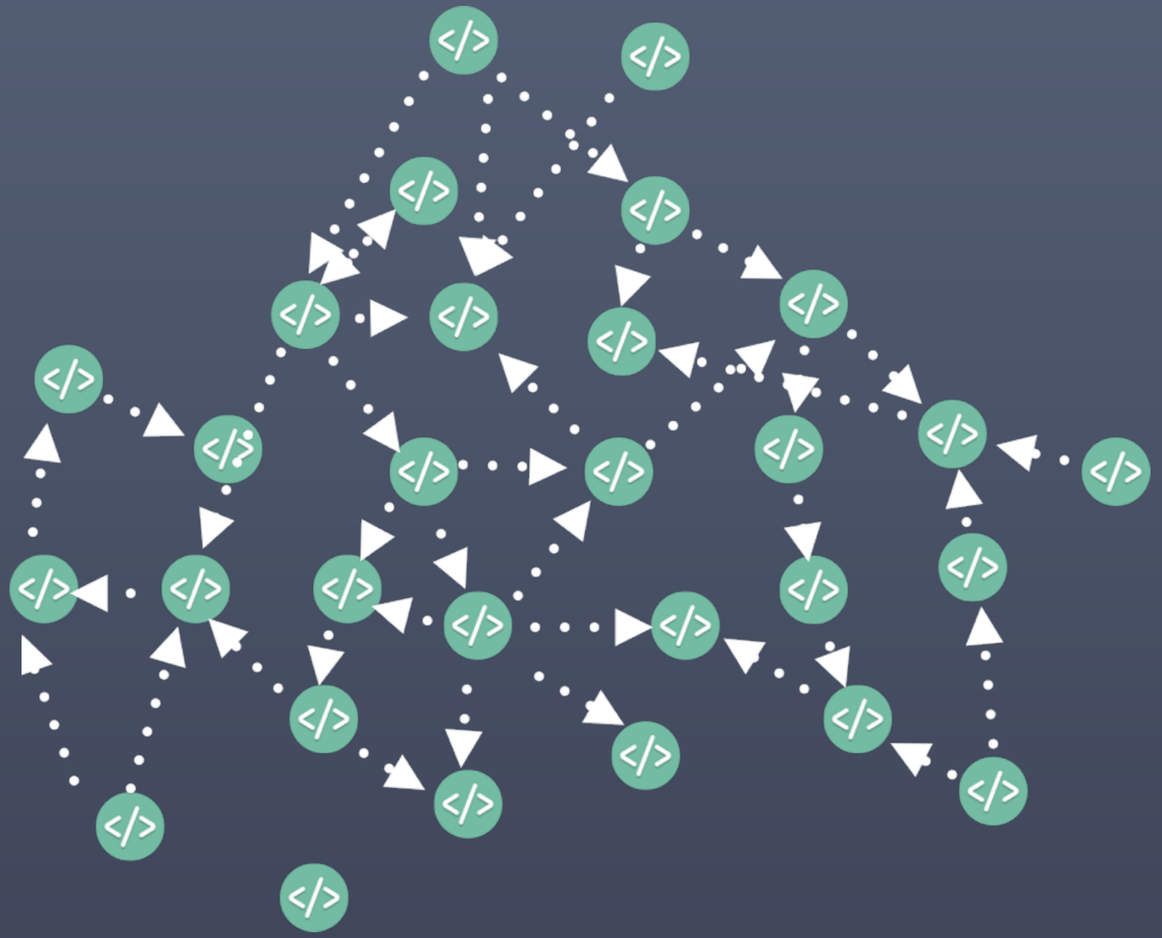
FUNCTIONAL PROGRAMMING IN VIEWS

@mikaelbrevik

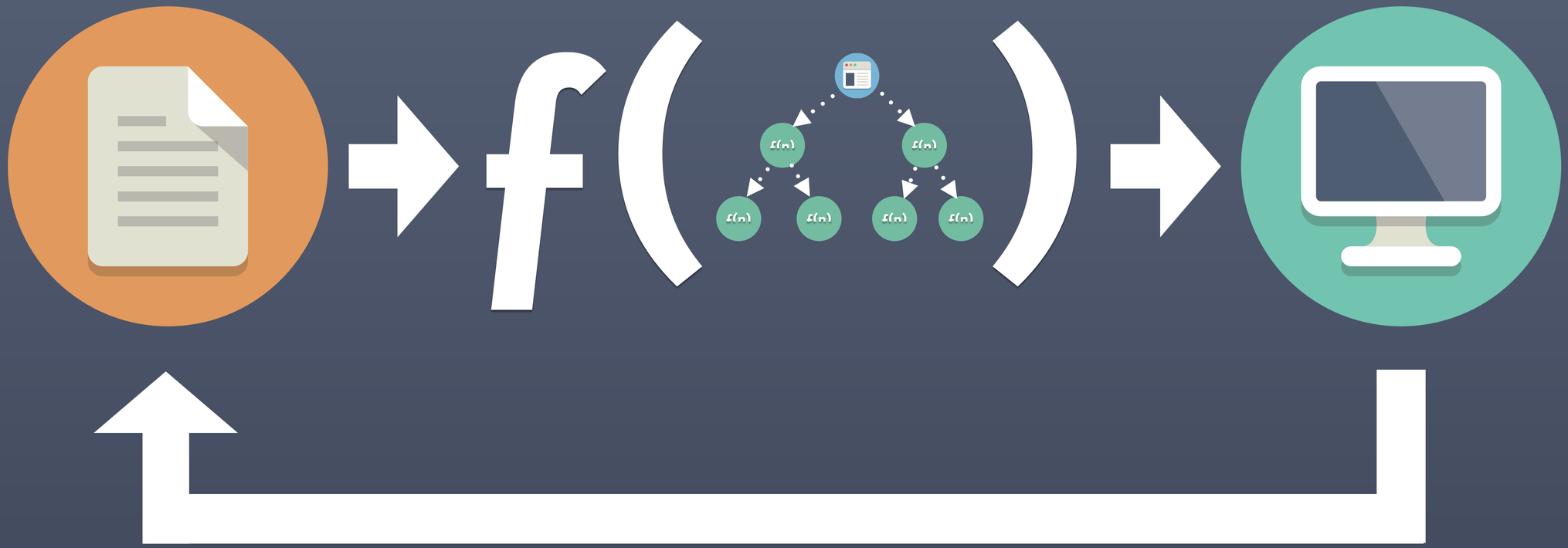




VIEWS OUT OF CONTROL

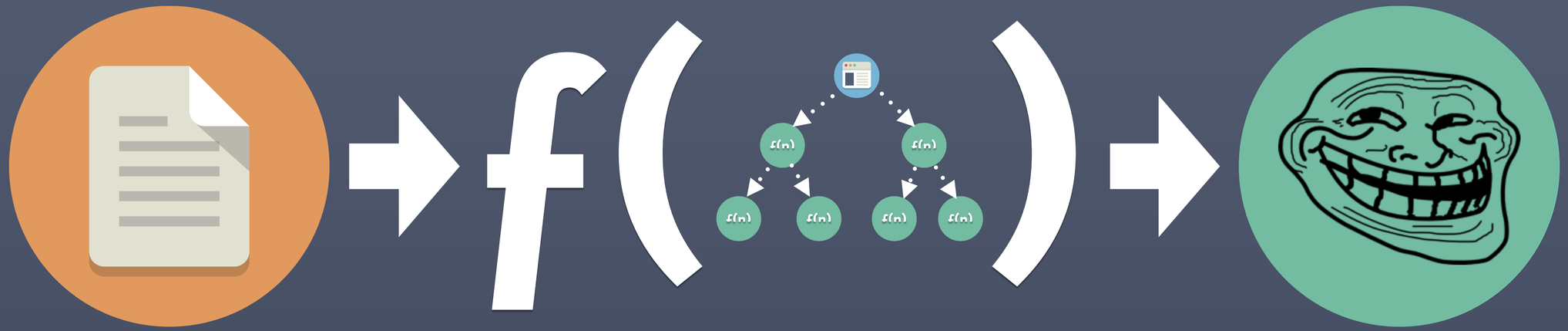


WHAT WE WANT

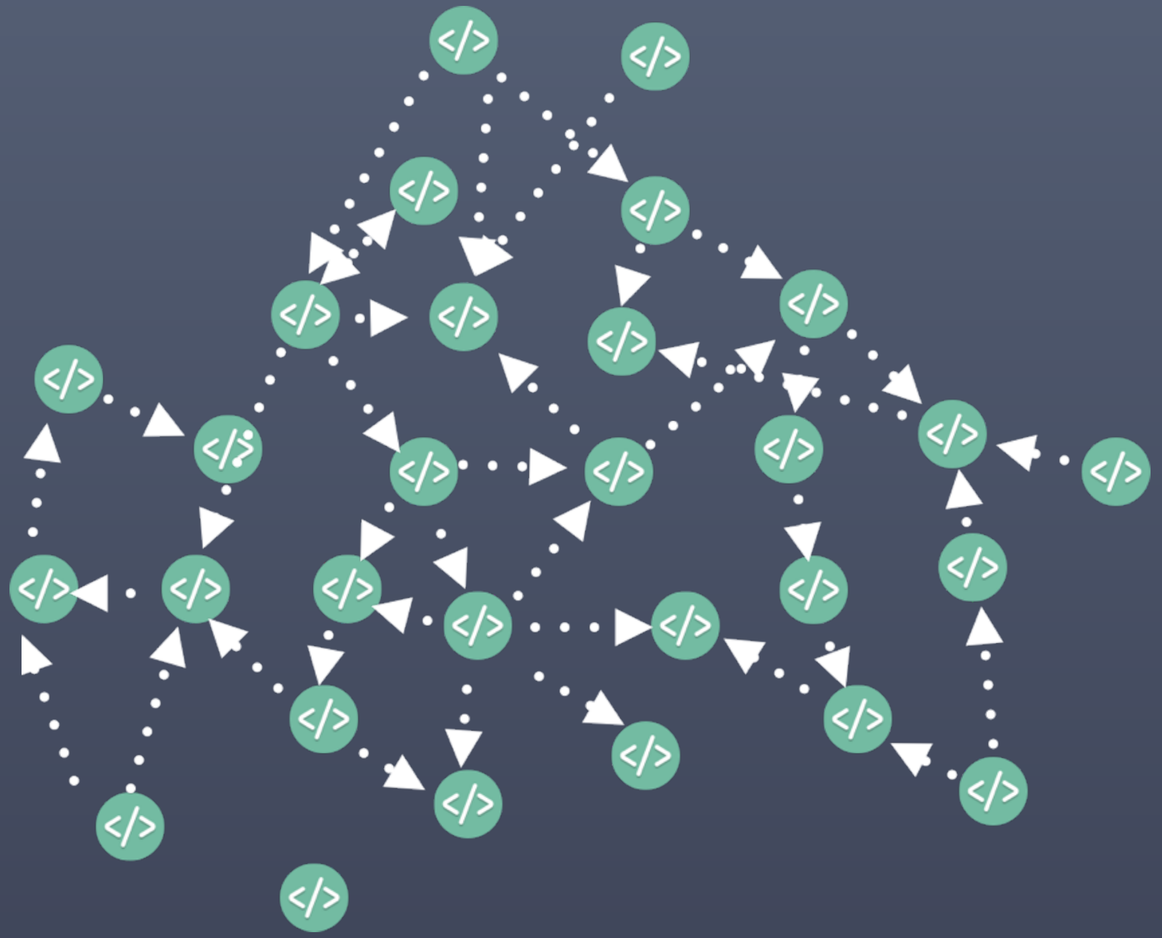


```
f(x) -> 2x
f(2) = 4
f(2) = 4
```

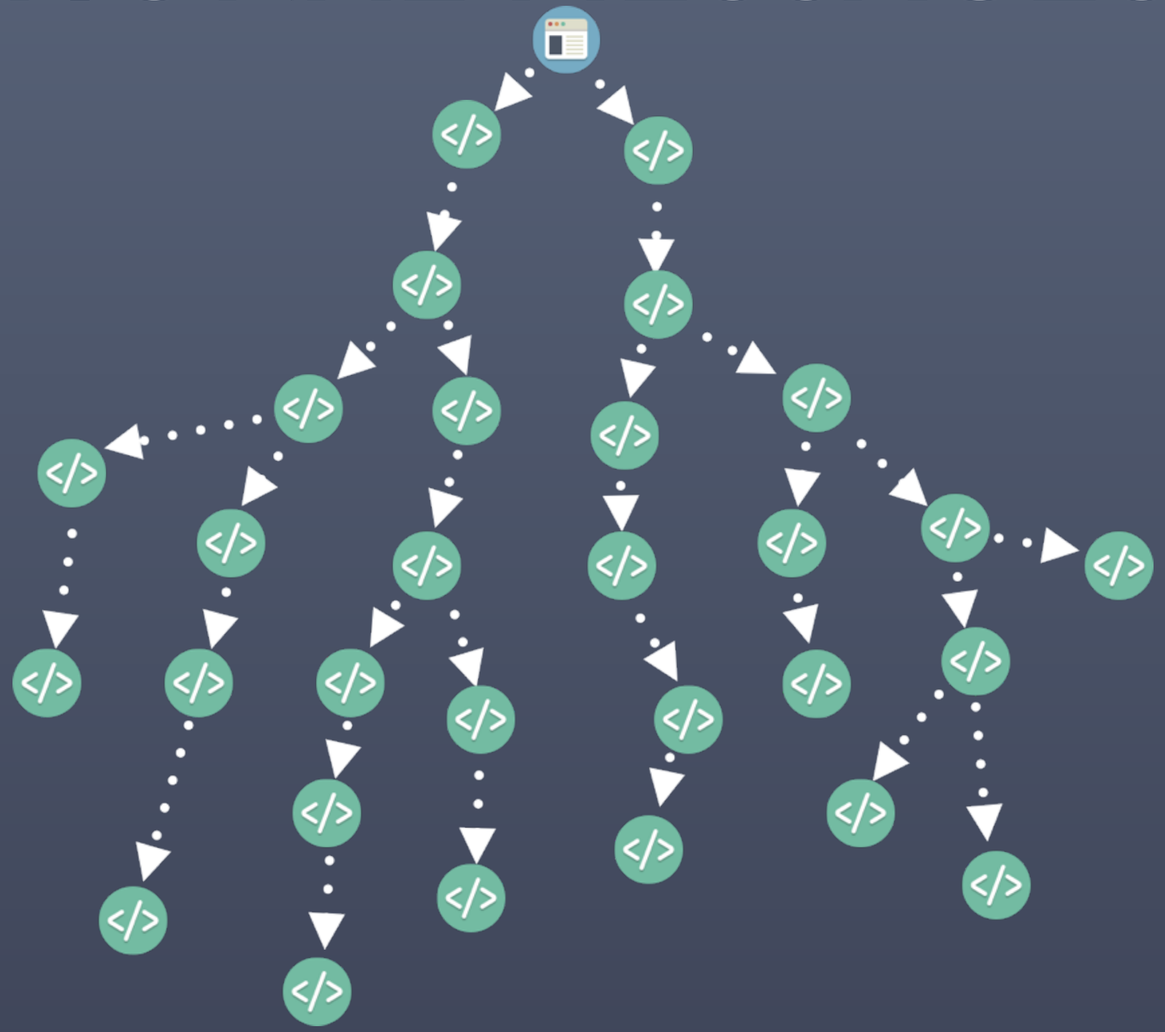
quote @torgeir


f(
f(n)
f(n)
f(n)
f(n)
f(n)

VIEWS OUT OF CONTROL



LOOSELY COUPLED & UNIDIRECTIONAL MESSAGES



REFERENTIAL TRANSPARENCY

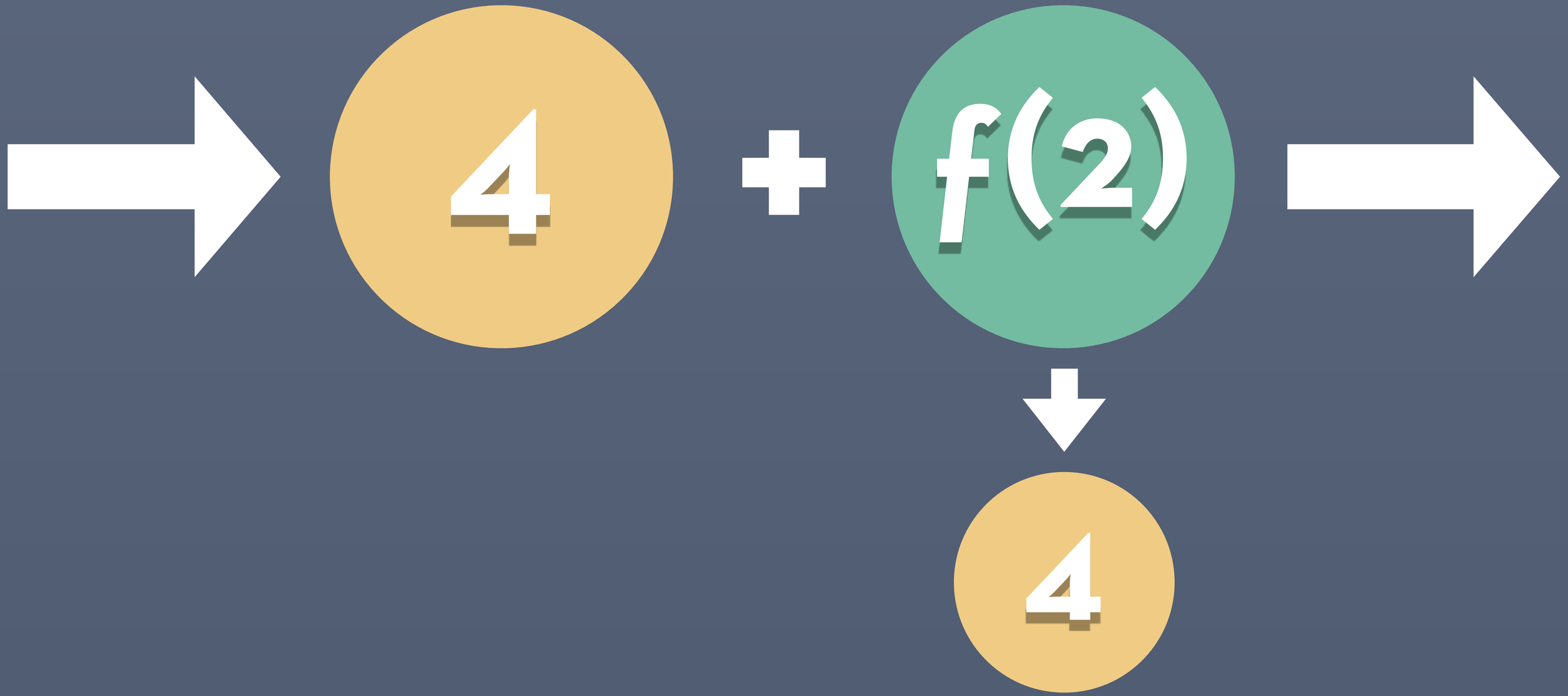
```
> 4 + 4
> 2 * 4
> 8
```

REFERENTIAL TRANSPARENCY

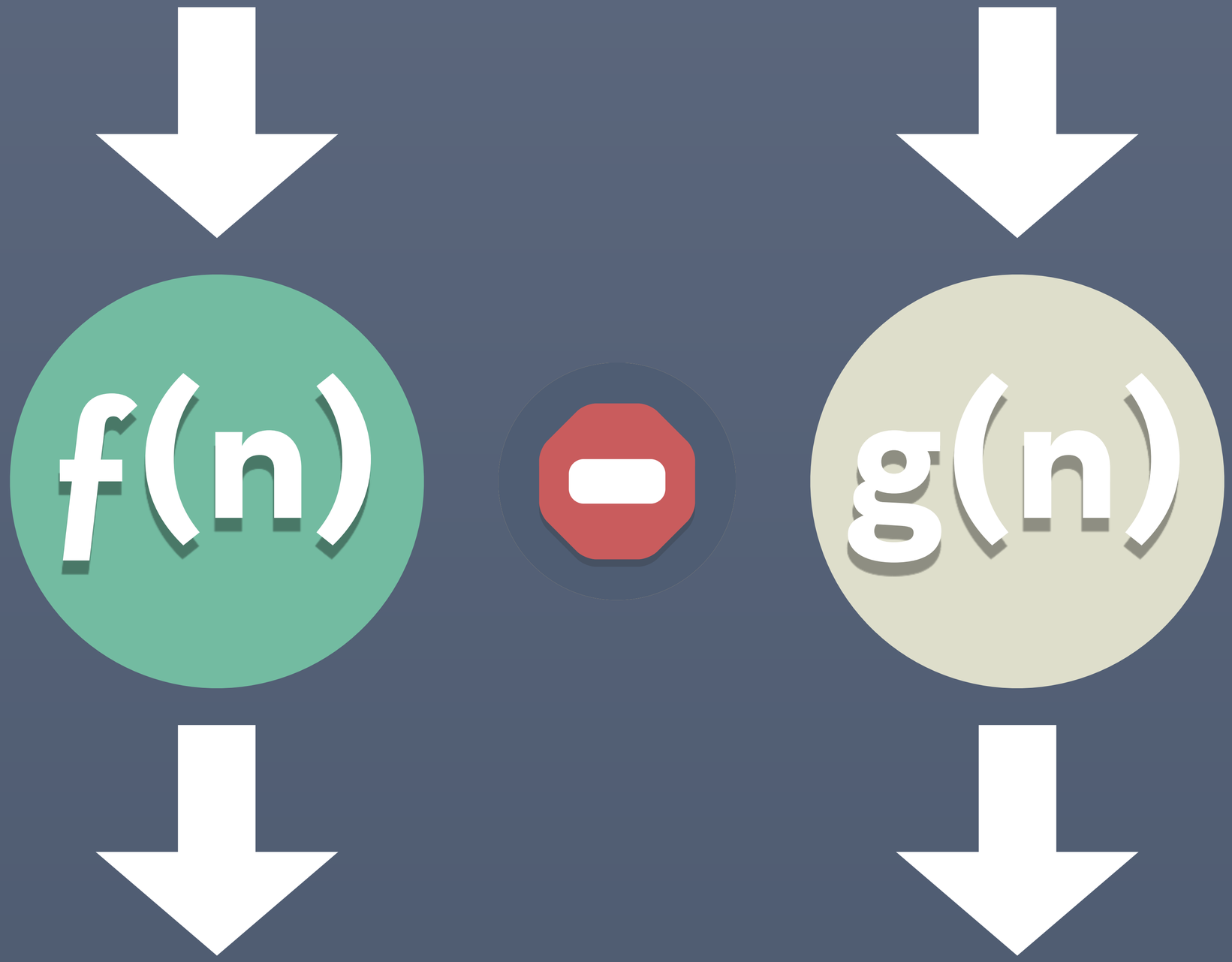
```
const square = (n) => n * n;
> square(2) + square(2);
> 2 * square(2);
> 2 * 4
> 8
```

PREDICTABILITY

```
const square = (n) => n * n;
> square(3); // 9
> square(3); // 9
> square(3); // 9
> square(3); // 9
```


4
f(2)
4


8


f(n)
g(n)

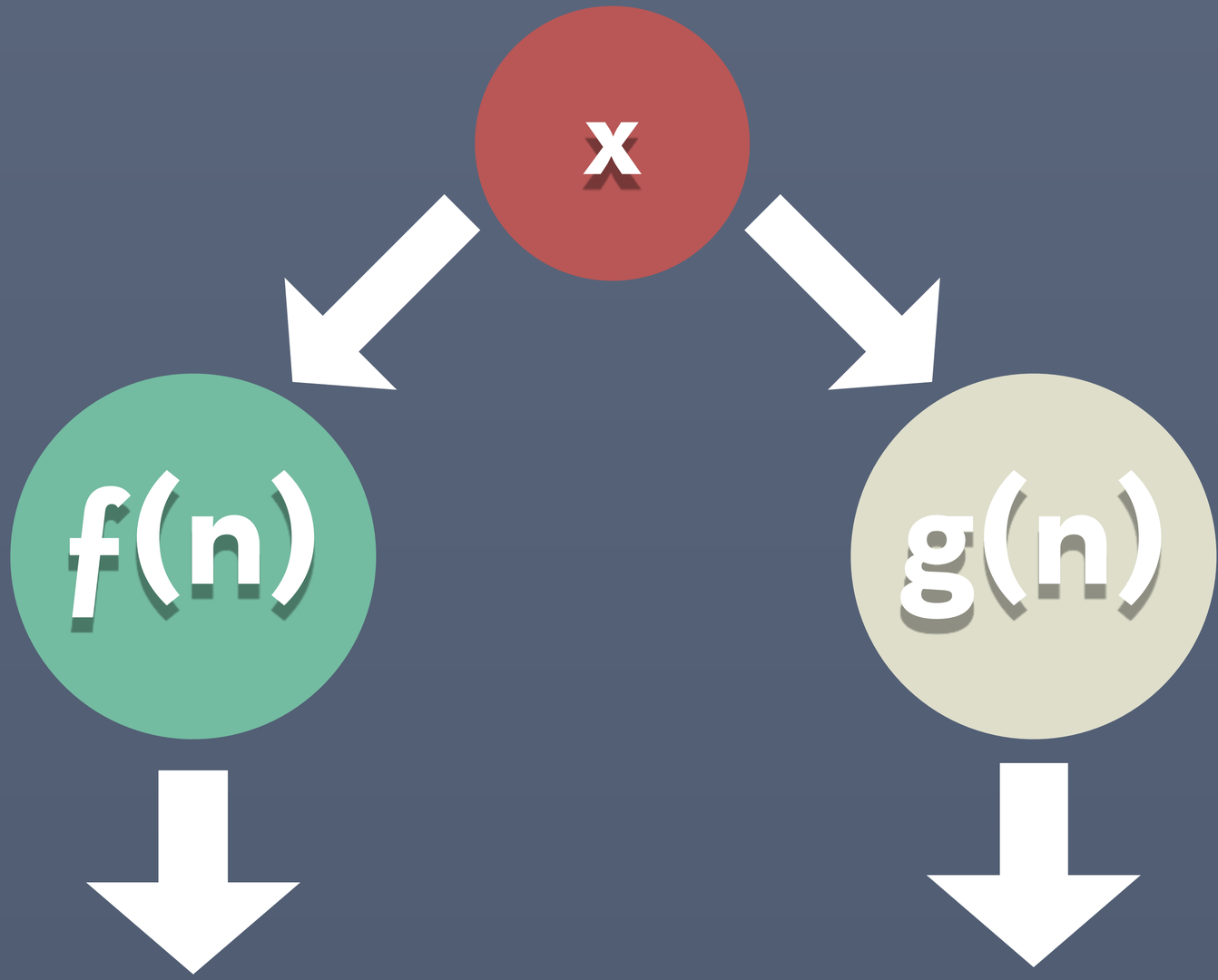
RESULTS OF SHARED MUTABLE STATE

ACCIDENTAL SHARED STATE

```
const setName = (obj, val) => obj.name = val;
const logInTime = (obj) =>
  setTimeout(() => log(obj.name));

const myObj = { name: 'Janet van Dyne' };
logInTime(myObj); // Logs 'Oops'
setName(myObj, 'Oops');
```

CAN'T PASS MUTABLE OBJECT



IMMUTABILITY

REFERENCE CHECKS

```
const obj = { name: "T'Challa" };

> obj === obj;
> obj !== { name: "T'Challa" };
> deepEqual(obj, { name: "T'Challa" });
```

REFERENCE CHECKS

```
const obj = { name: "T'Challa" };

> obj === obj;
> obj !== { name: "T'Challa" };
> deepEqual(obj, { name: "T'Challa" });
```

REFERENCE CHECKS

```
const obj = { name: "T'Challa" };

> obj === obj;
> obj !== { name: "T'Challa" };
> deepEqual(obj, { name: "T'Challa" });
```

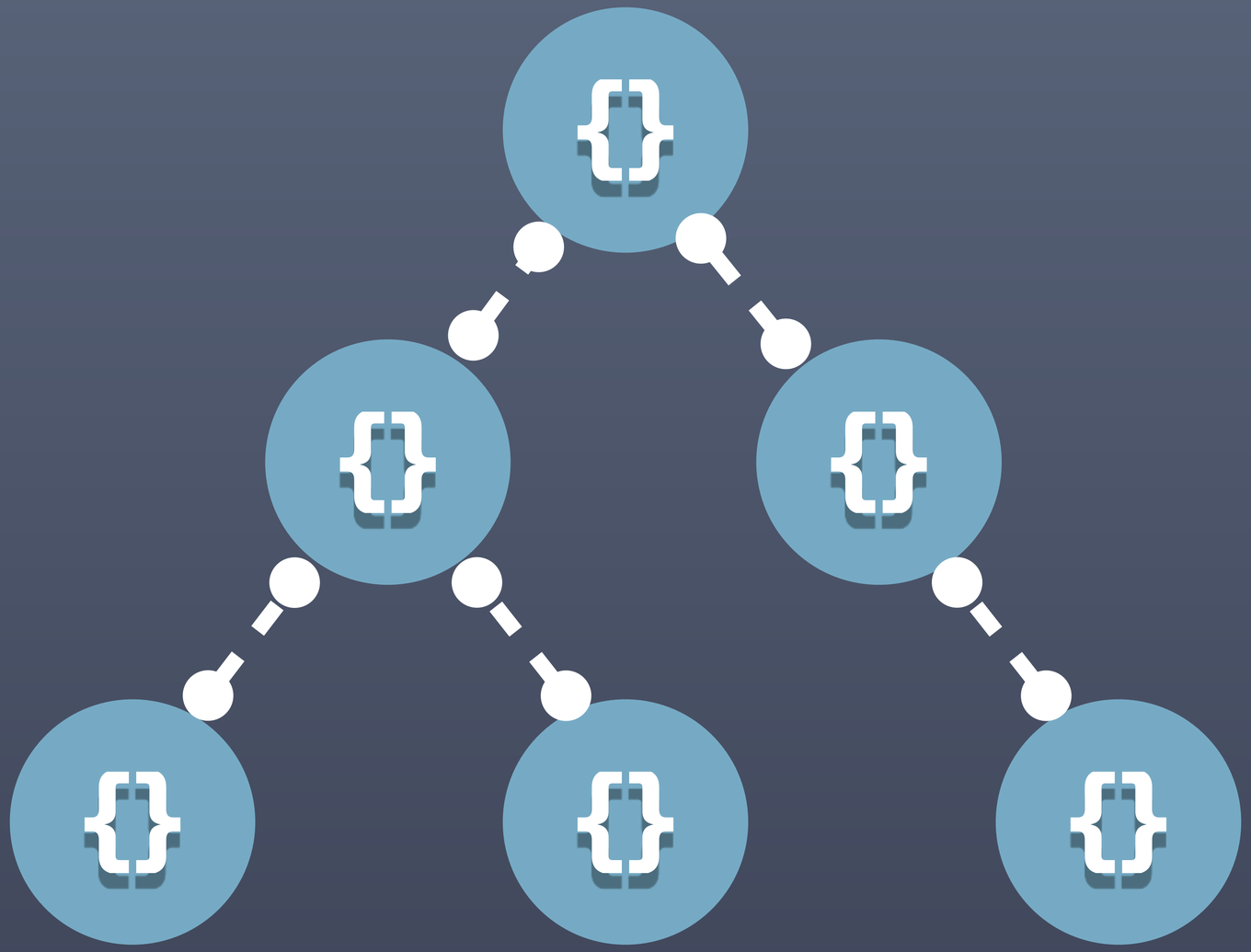
IMMUTABLE MAPS

```
const obj = Immutable.Map({ name: "T'Challa" });
const obj2 = obj.set('name', 'Black Panther');
obj.get('name'); //=> T'Challa
obj2.get('name'); //=> Black Panther
```

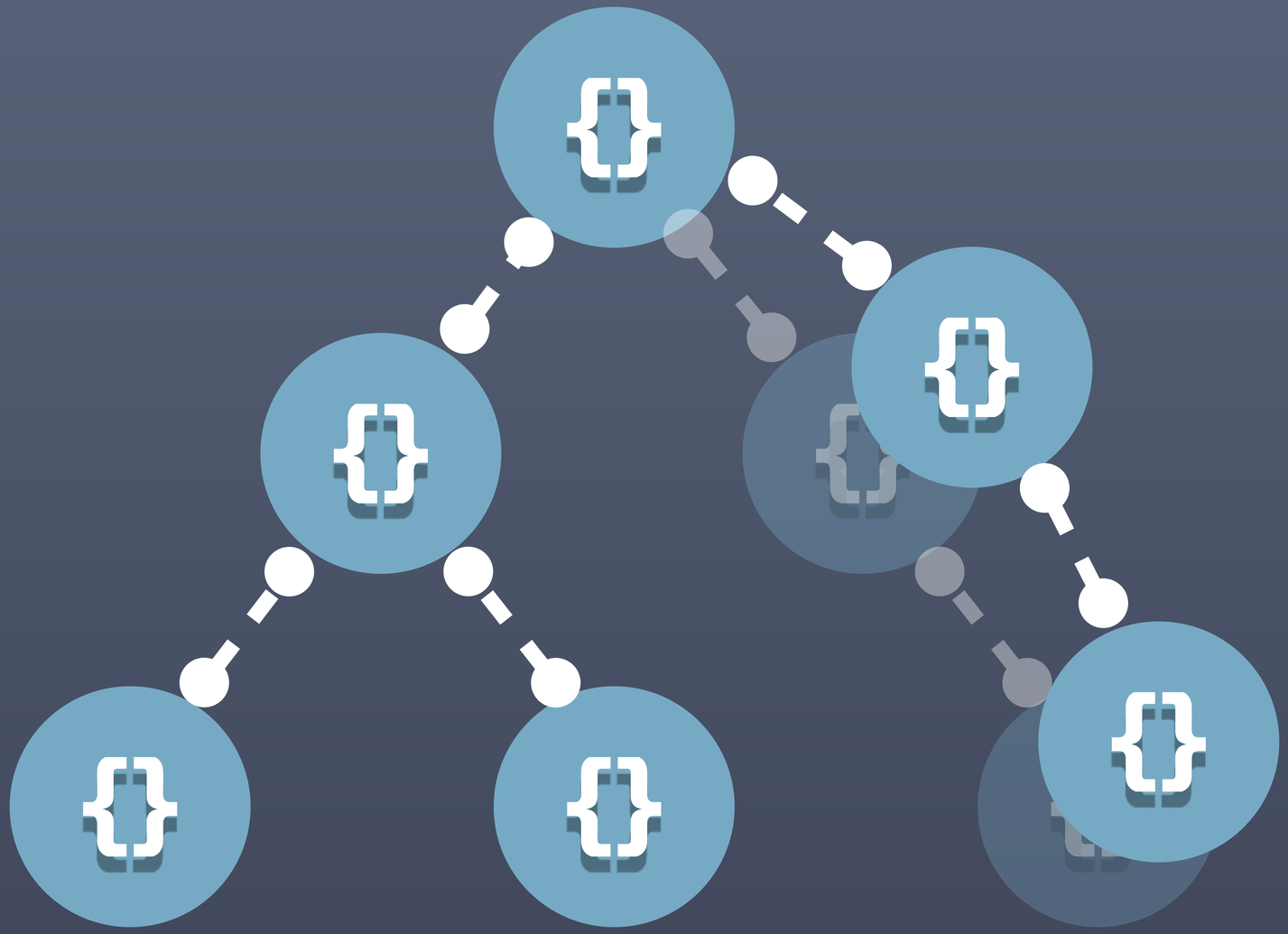
REFERENCES IN IMMUTABLE

```
const obj = Immutable.Map({ name: "T'Challa" });
const obj2 = obj.set('name', 'Black Panther');
obj !== obj2;
obj === obj;
```

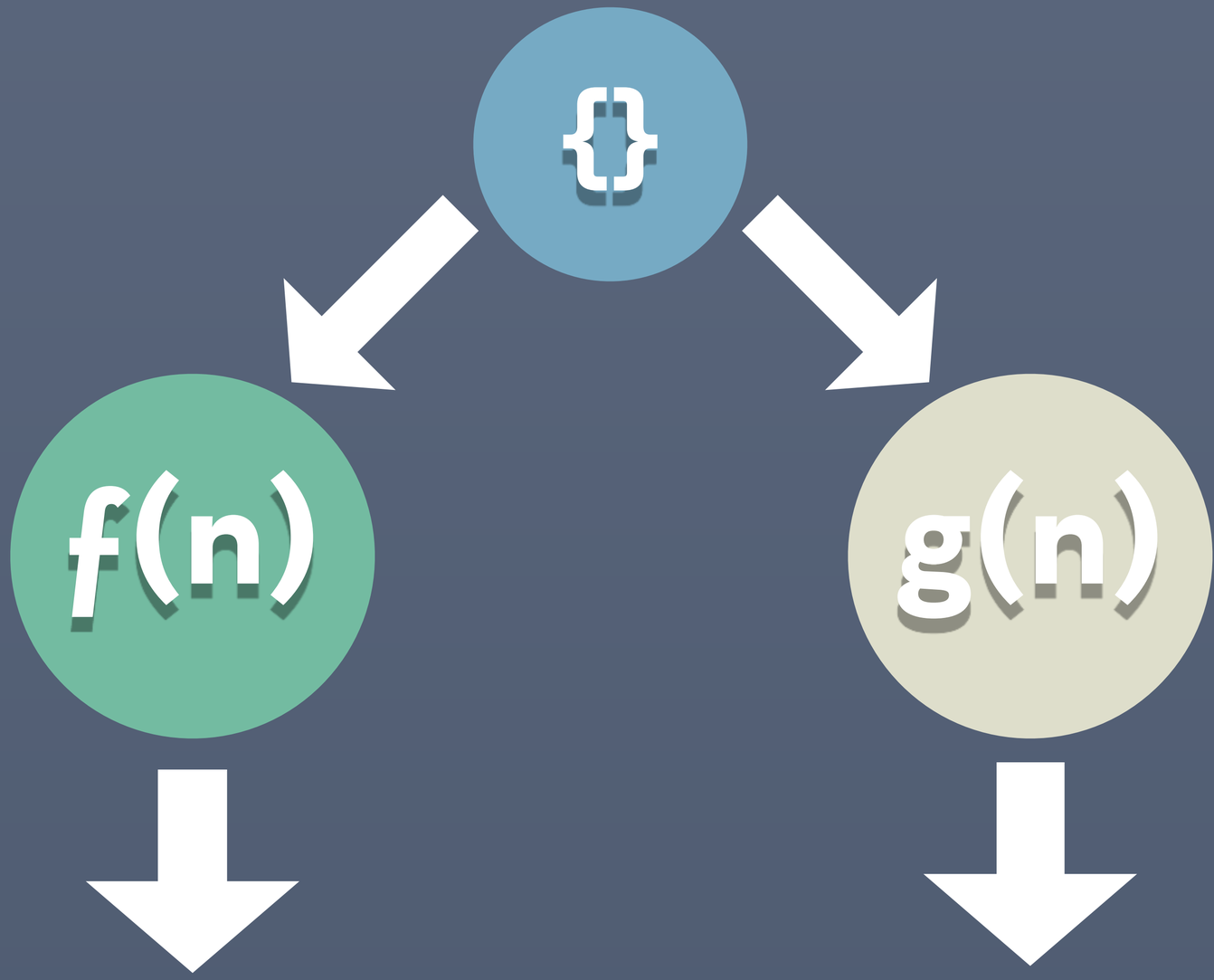
STRUCTURAL SHARING



STRUCTURAL SHARING



CAN PASS IMMUTABLE OBJECT



PURITY IN A STATEFUL ENVIRONMENT (DOM)

ACCIDENTAL SHARED STATE

```
const setName = (obj, val) => obj.name = val;
const logInTime = (obj) =>
  setTimeout(() => log(obj.name));

const myObj = { name: 'Janet van Dyne' };
logInTime(myObj); // Logs 'Oops'
setName(myObj, 'Oops');
```

ACCIDENTAL SHARED STATE IN THE DOM

```
function updateH1 (h1, val) {
  h1.textContent = val;
  return h1;
}

// <h1>Janet van Dyne</h1>
var el = document.querySelector('h1');
logInTime(el); // Logs 'Oops'
updateH1(el, 'Oops');
```

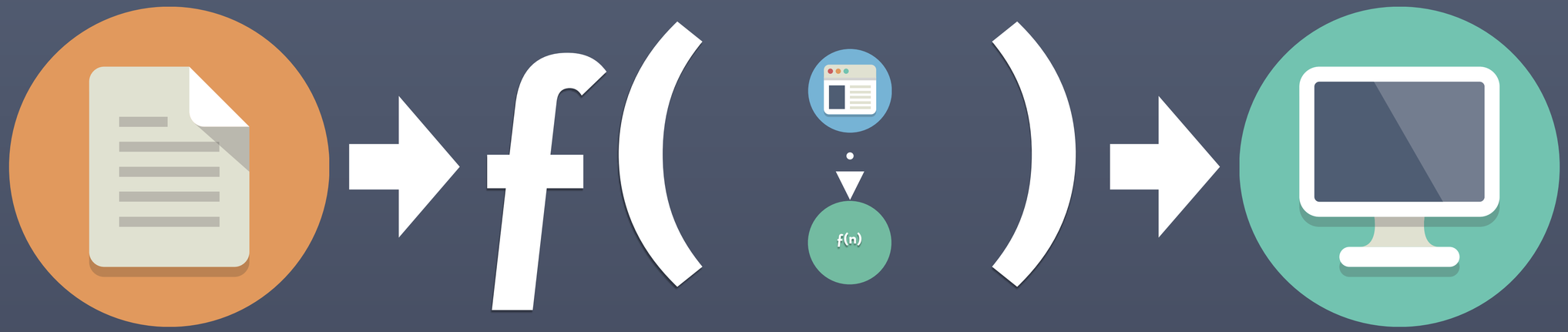
THE DOM AS A SIDE-EFFECT

```
f(x) -> 2x
f(2) = 4
```

SYSTEM AS A FUNCTION OF STATE

System Input

System Output



All of our UI code

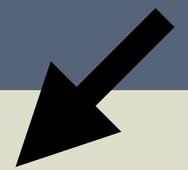
SYSTEM AS A FUNCTION OF STATE

```
var systemOutput = allOfUICode(systemInput);

React.render(systemOutput, outputDomElement);
```

SYSTEM AS A FUNCTION OF STATE

A lot of View Components



```
var systemOutput = allOfUICode(systemInput);

React.render(systemOutput, outputDomElement);
```

COMPONENTS

```
const MyHeader = (name) => <h1>Hello, {name}!</h1>;
```

```
const MyApp =({children}) => (
  <article>
    <Header>Hello, {children}!</Header>
  </article>
);
```

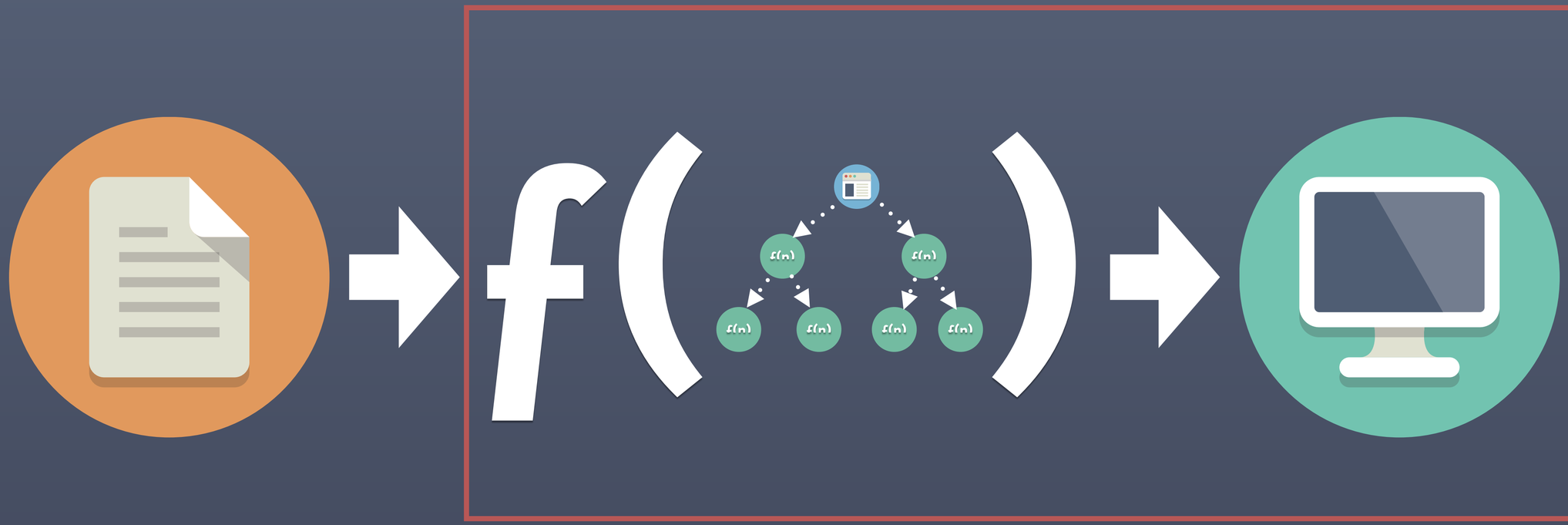
PREDICTABLE

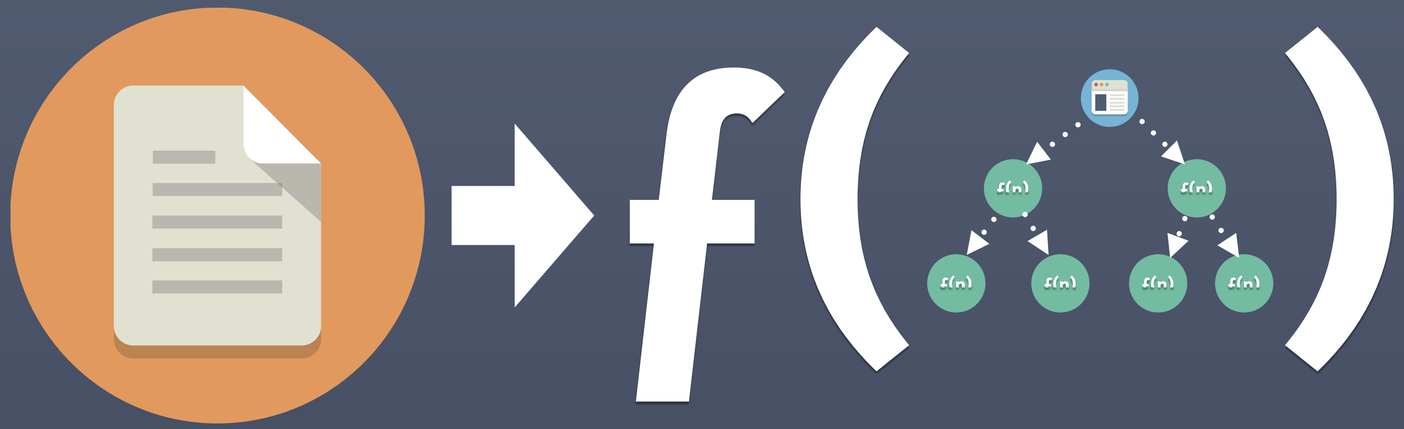
```
render(<MyApp>Robert Bruce Banner</MyApp>, el);
```

```
render(<MyApp>Robert Bruce Banner</MyApp>, el);
```

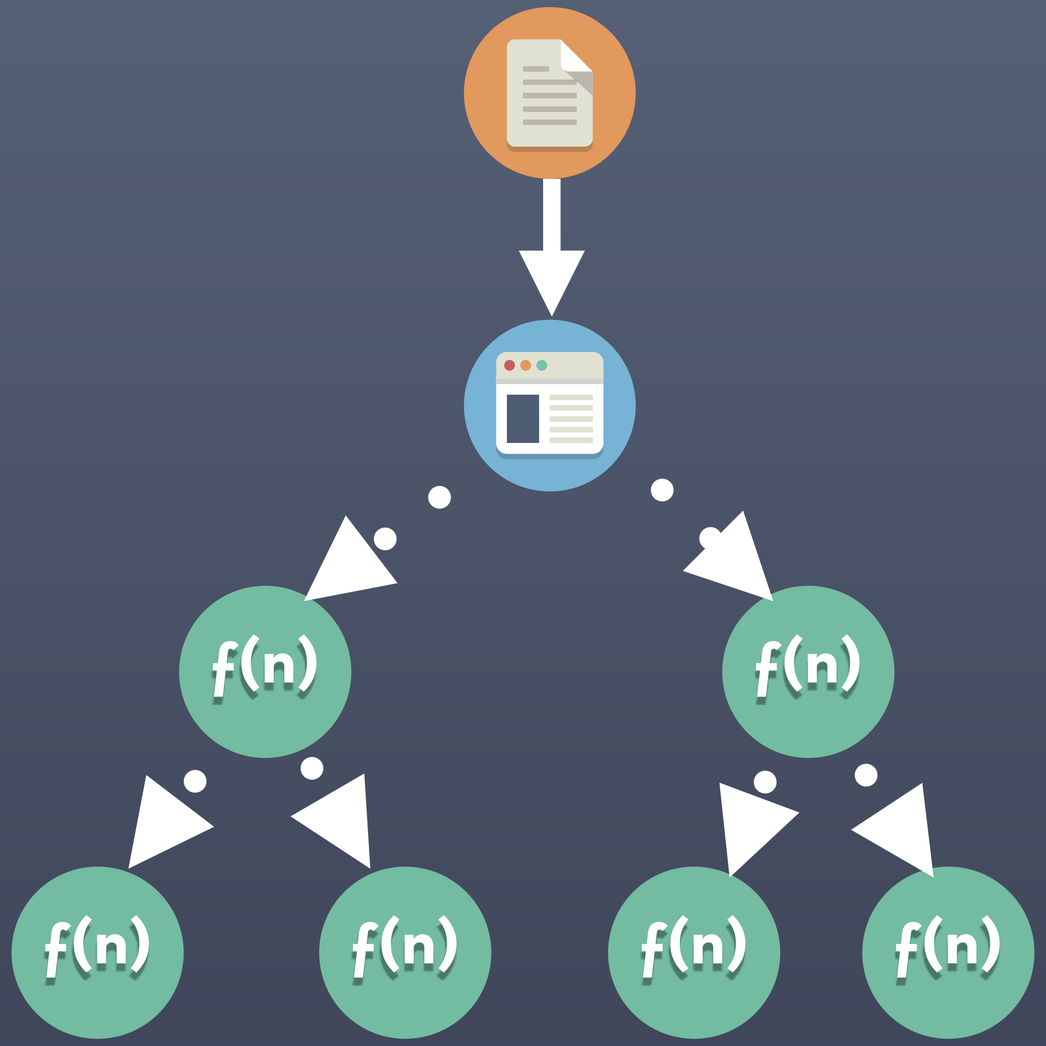
```
render(<MyApp>Robert Bruce Banner</MyApp>, el);
```

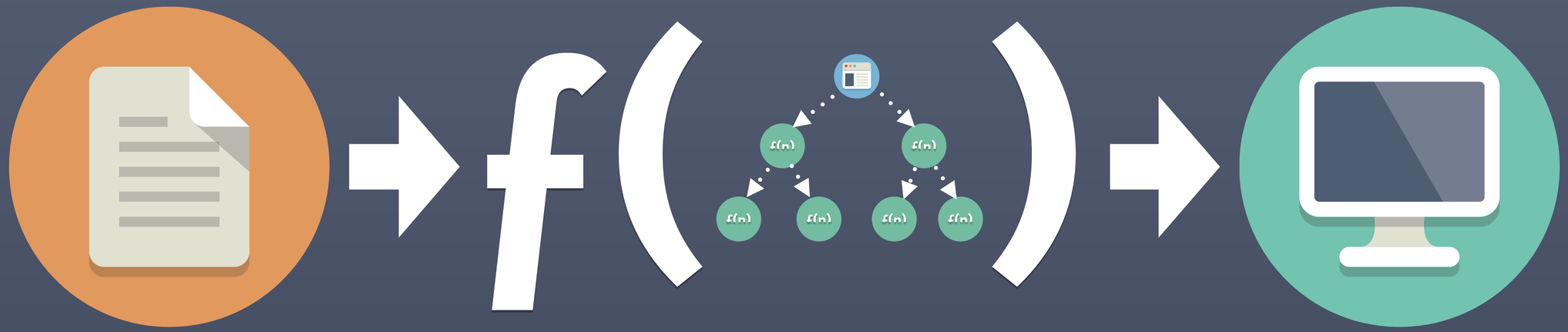
```
render(<MyApp>Robert Bruce Banner</MyApp>, el);
```


f(
f(n)
f(n)
f(n)
f(n)
f(n)
f(n)
)

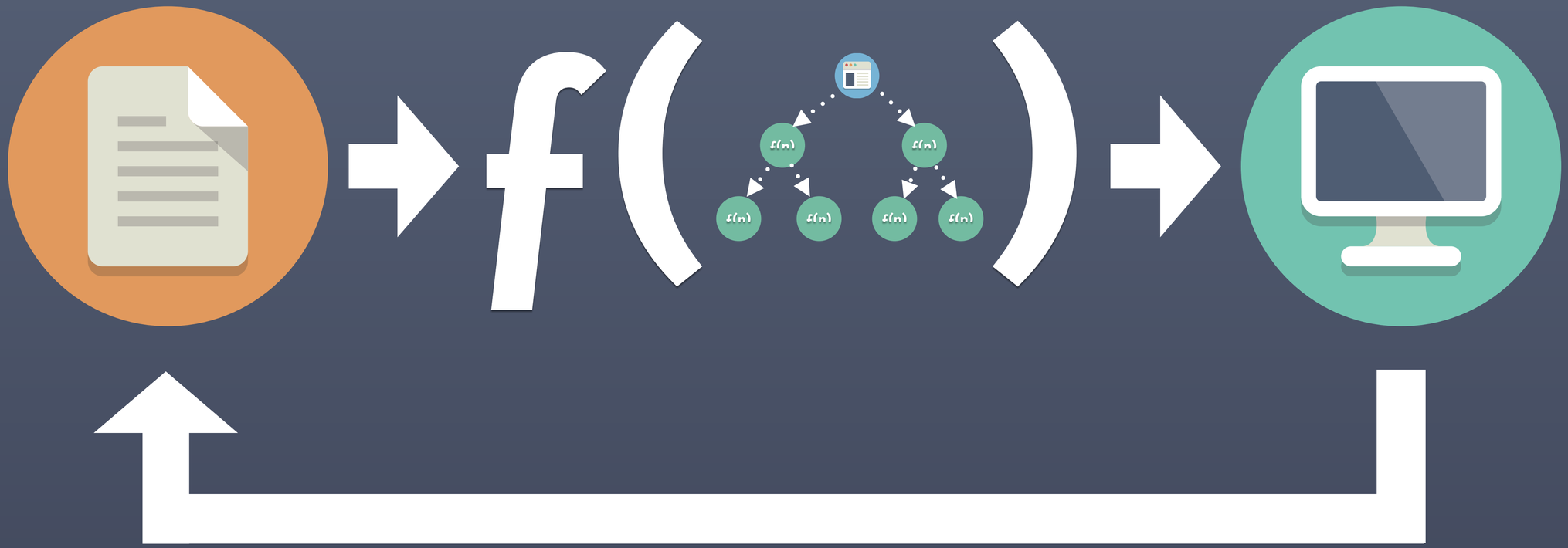

f(n)
f(n)
f(n)
f(n)
f(n)
f(n)

GLOBAL STATE

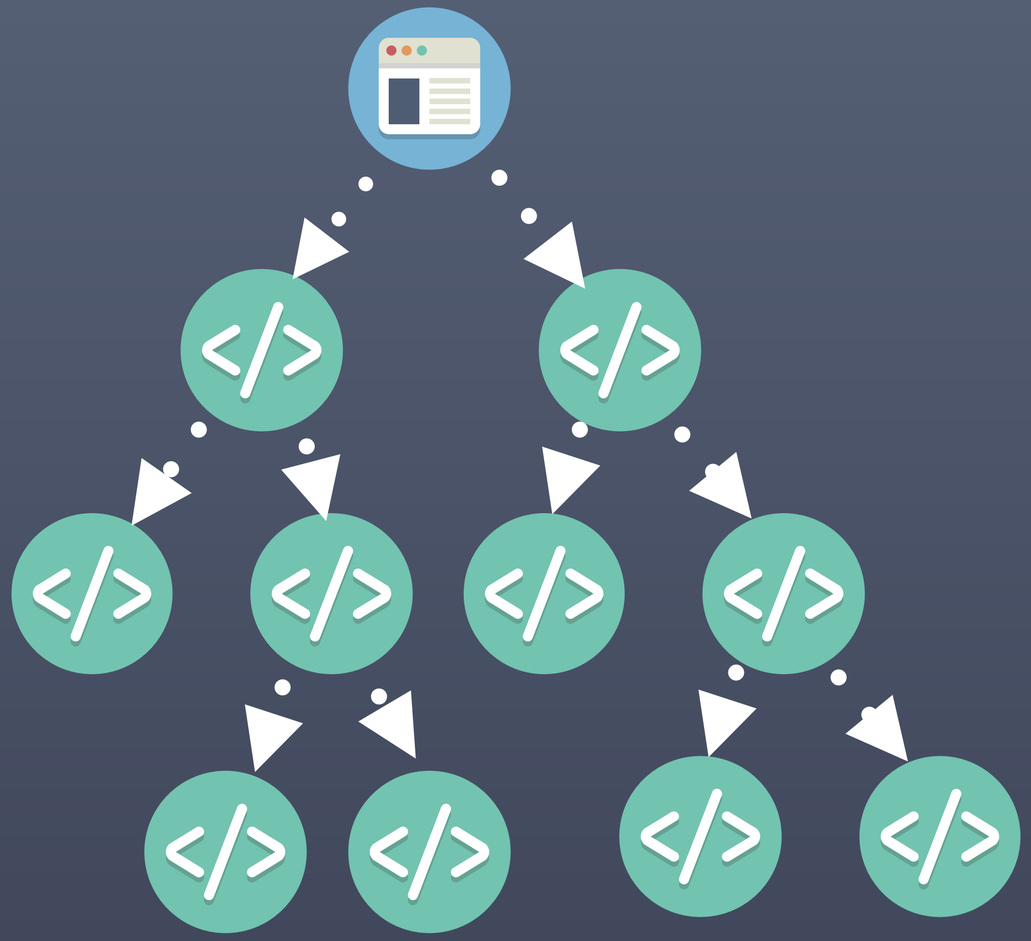



f(
f(n)
f(n)
f(n)
f(n)
f(n)

ALWAYS RERENDER ENTIRE APP



ISN’T THIS SLOW?



REMEMBER: IMMUTABLE MAPS

```
const obj = Immutable.Map({ name: "T'Challa" });
const obj2 = obj.set('name', 'Black Panther');
obj !== obj2;
obj === obj;
```

```
function fibonacci(n) {
  if (n === 0 || n === 1)
    return n;
  else
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

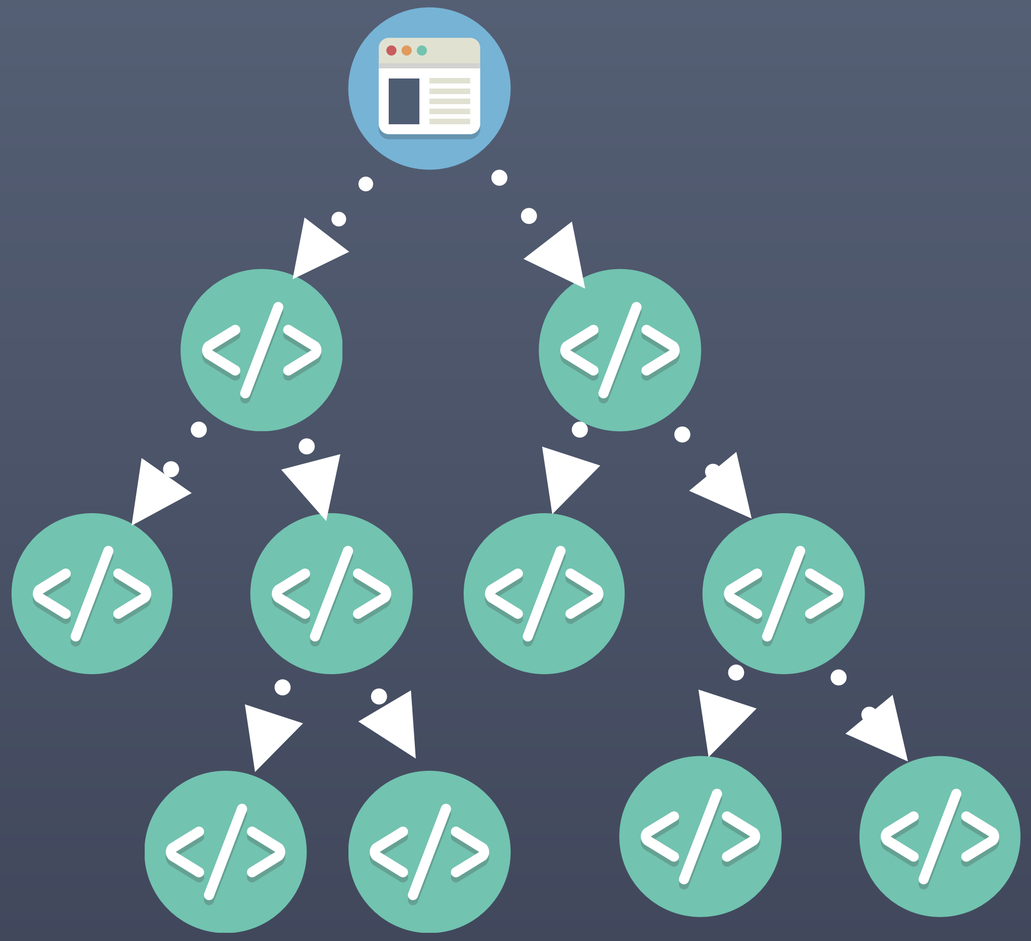
MEMOIZATION

```
function fibonacci = memoize(
  function(n) {
    if (n === 0 || n === 1)
      return n;
    else
      return fibonacci(n - 1) + fibonacci(n - 2);
  }
);
```

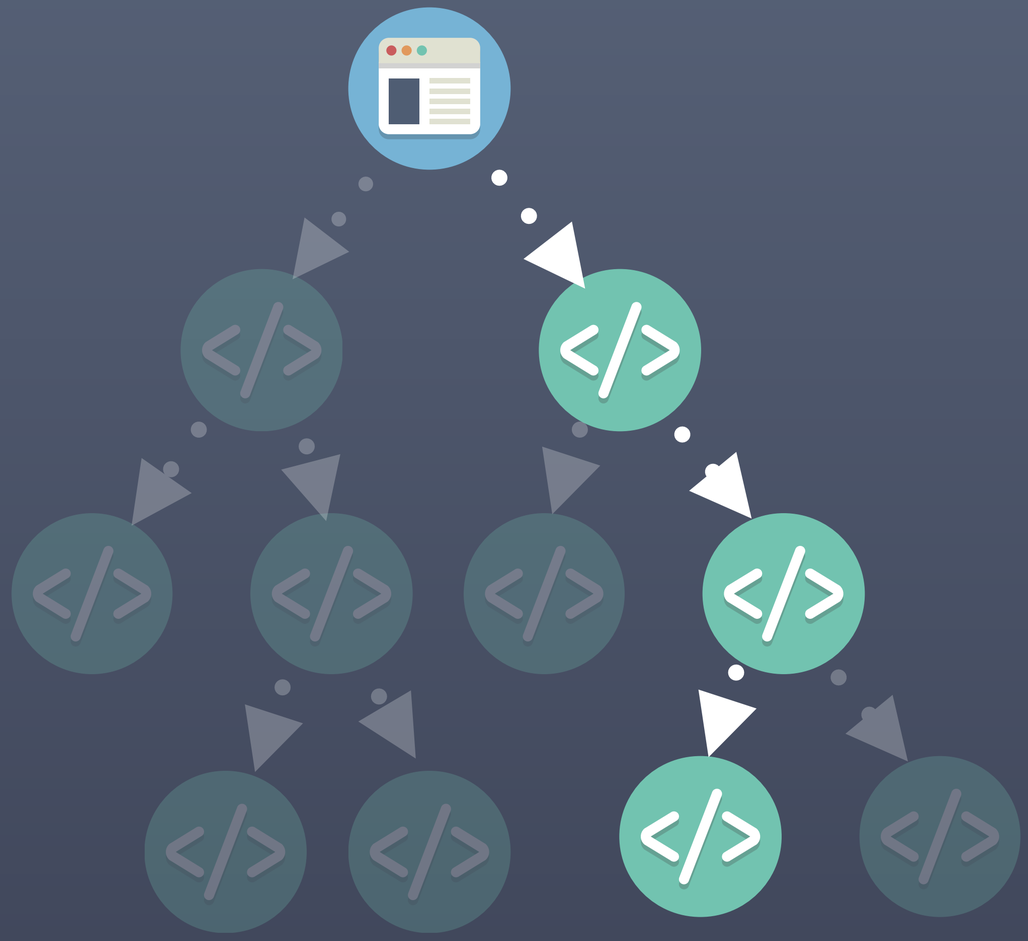
MEMOIZATION

```
const MemoizedComponent = memoize(SomeComponent);
```

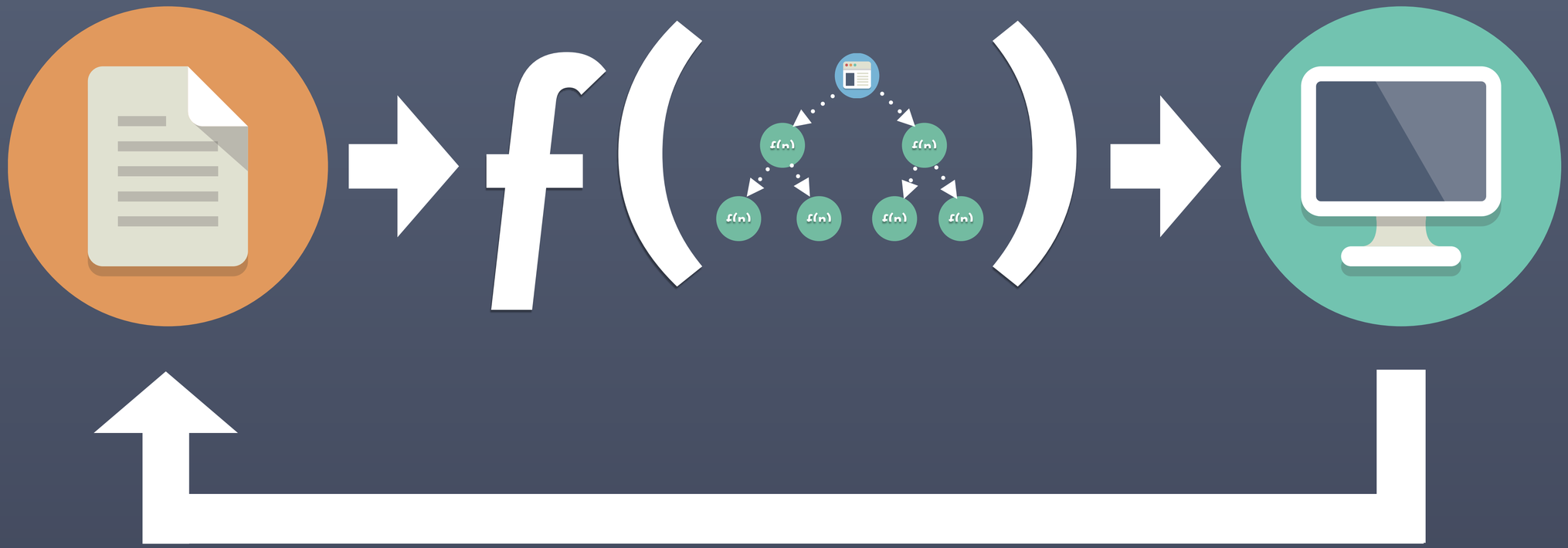
SMART TREE UPDATE



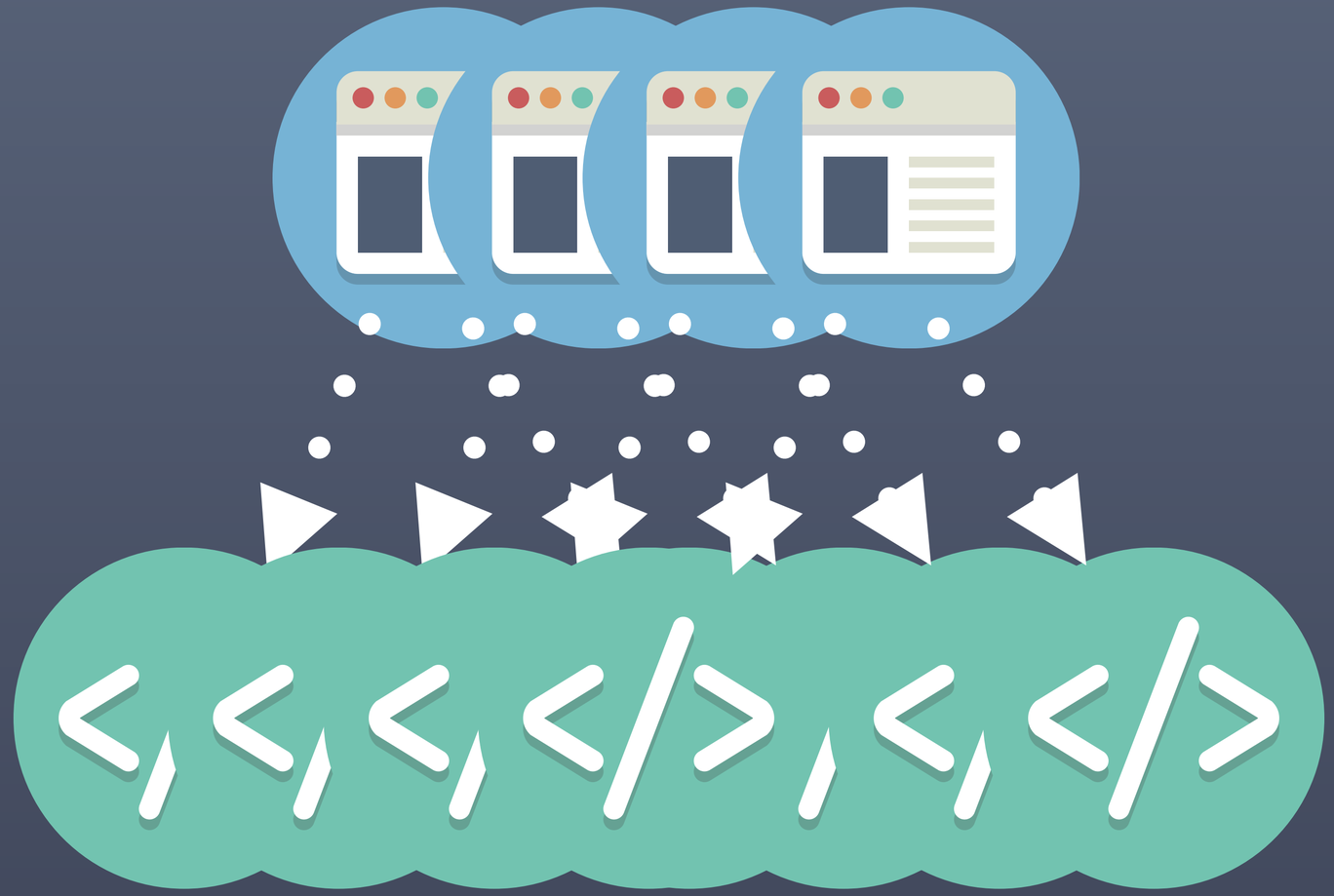
SMART TREE UPDATE



GAME LOOP OF THE APP

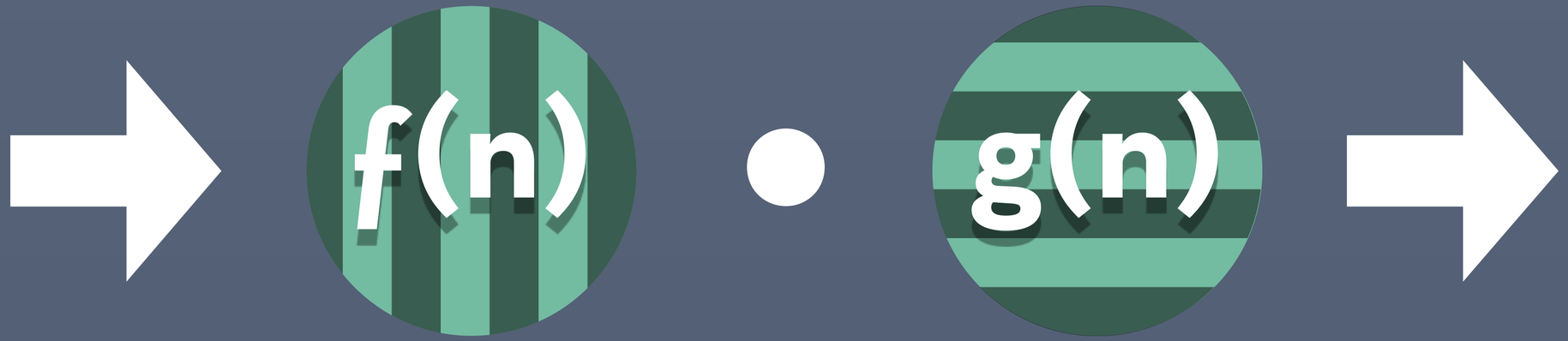


STATE OVER TIME

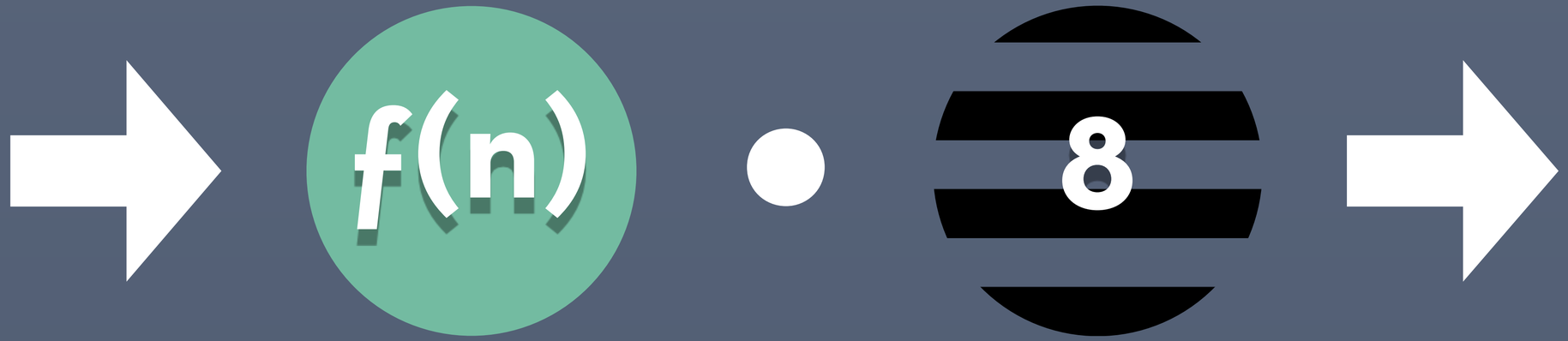


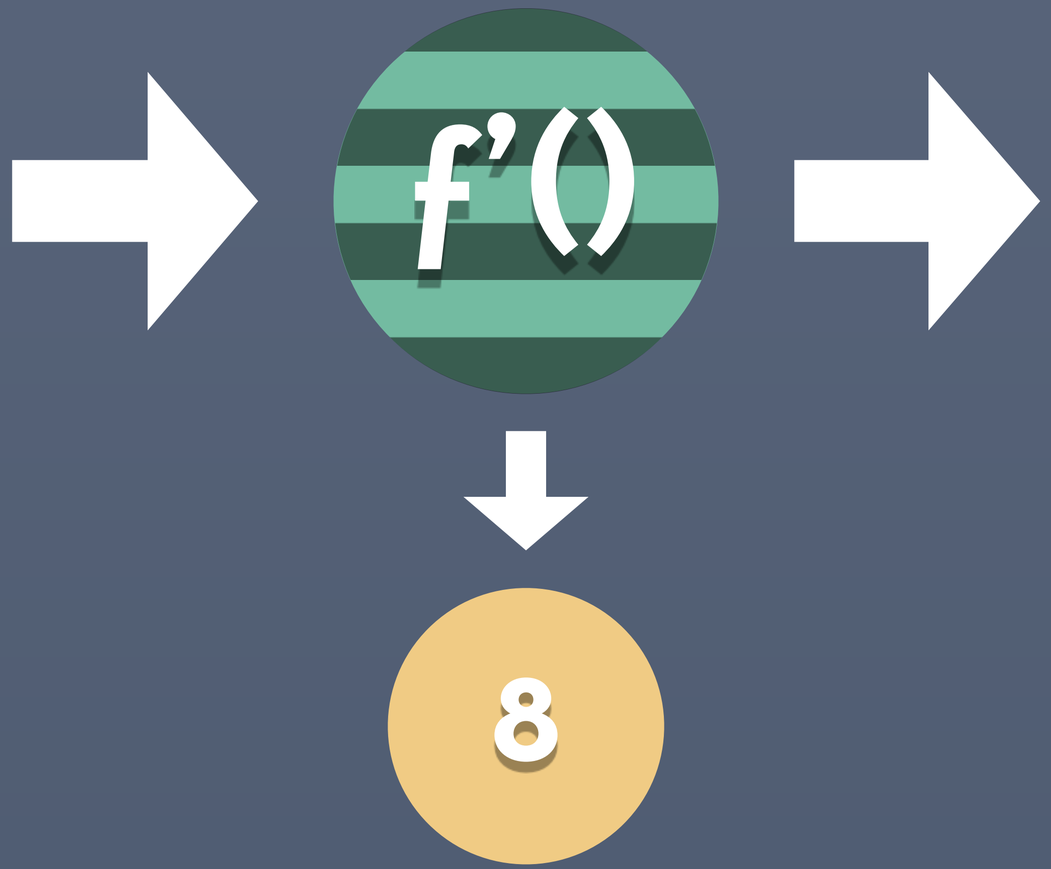
STATE OVER TIME

FUNCTIONAL PROGRAMMING: COMPOSABILITY


f(n)
g(n)


h(n)


f(n)
8


f’()
8

(HIGHER ORDER) COMPONENTS

COMPOSABLE COMPONENTS

```
const Header = ({children}) => (..);
const Italic = ({children}) => (..);
const ItalicHeader = compose(Header, Italic);
```

```
const MyComponent =({children}) => (
  <article>
    <ItalicHeader>Hello, {children}!</ItalicHeader>
  </article>
);
```

```
const Herolist =
  ['Wasp', 'Falcon', 'Harry Potter']
    .filter(isSuperHero)
    .map((title) =>
      <MyComponent>{title}</MyComponent>
    );
```

COMPOSABLE COMPONENTS

```
const Header ={{type, children}) => (
  <h1 className={type}>{children}</h1>
);
const MainHeader = partial(Header, ‘main-title’);
const PostHeader = partial(Header, 'post-title');
```

COMPOSABLE COMPONENTS

```
const MainHeader = partial(Header, 'main-title');
const ItalicHeader = compose(MainHeader, Italic);
const FailSafeItalicHeader = maybe(ItalicHeader);
```

```
const MainHeader = partial(Header, 'main-title');
const ItalicHeader = compose(MainHeader, Italic);
const FailSafeItalicHeader = maybe(ItalicHeader);
```

```
render(
 <FailSafeItalicHeader>
  Robert Bruce Banner
 </FailSafeItalicHeader>, el);
```

```
render(
 <FailSafeItalicHeader>
   Robert Bruce Banner
 </FailSafeItalicHeader>, el);
```

```
render(
 <FailSafeItalicHeader>
   Robert Bruce Banner
 </FailSafeItalicHeader>, el);
```

```
render(
 <FailSafeItalicHeader>
   Robert Bruce Banner
 </FailSafeItalicHeader>, el);
```

- FUNCTIONAL PARADIGMS IN UI
- SYSTEM AS A FUNCTION OF STATE
- GLOBAL ALL-KNOWING APP STATE
- RENDER LOOP AND STATIC MENTAL MODEL
- COMPOSABLE (HIGHER ORDER) COMPONENTS

Omniscient

OMNISCIENT.JS

THIS ARCHITECTURE AS A HELPING LIBRARY
HTTP://OMNISCIENTJS.GITHUB.IO/

IMMUTABLE DATA STRUCTURES

VIDEO ON HOW IMMUTABLE.JS WORKS
HTTPS://WWW.YOUTUBE.COM/WATCH?V=I7IDS-PBEGI

SIMPLER UI REASONING

ARTICLE FORMAT OF THIS PRESENTATION
HTTP://OMNISCIENTJS.GITHUB.IO/GUIDES/01-SIMPLER-UI-REASONING-WITH-UNIDIRECTIONAL/

FUNCTIONAL PATTERNS FOR VIEWS

ARTICLE ON FUNCTIONAL PATTERNS IN FOR VIEWS
HTTPS://BLOG.RISINGSTACK.COM/FUNCTIONAL-UI-AND-COMPONENTS-AS-HIGHER-ORDER-FUNCTIONS/

OM

IMPLEMENTATION OF THIS ARCHITECTURE IN CLOJRESCRIPT
HTTPS://GITHUB.COM/OMCLJS/OM

FUNCTIONAL PROGRAMMING IN VIEWS

twitter @mikaelbrevik
github @mikaelbr