



flow

Fast and Precise Type Checking for JavaScript

Avik Chaudhuri



flow

A STATIC TYPE CHECKER FOR JAVASCRIPT

Type Inference

Flow uses type inference to find bugs even without type annotations. It precisely tracks the types of variables as they flow through your program.

Idiomatic JS

Flow is designed for JavaScript programmers. It understands common JavaScript idioms and very dynamic code.

Realtime Feedback

Flow incrementally rechecks your changes as you work, preserving the fast feedback cycle of developing plain JavaScript.

Flow can catch common bugs in JavaScript programs before they run, including:

- silent type conversions,
- `null` dereferences,
- and the dreaded `undefined` is not a function.

```
1 // @flow
2 function foo(x) {
3   return x * 10;
4 }
5 foo('Hello, world!');
```

show Flow output

facebook / flow



Code Issues 994 Pull requests 26 Projects 0 Wiki Pulse Graphs

Unwatch 321 ★ Unstar 9,891 Fork 757

Adds static typing to JavaScript to improve developer productivity and code quality. <http://flowtype.org/>



3,767 commits 17 branches 55 releases 276 contributors BSD-3-Clause

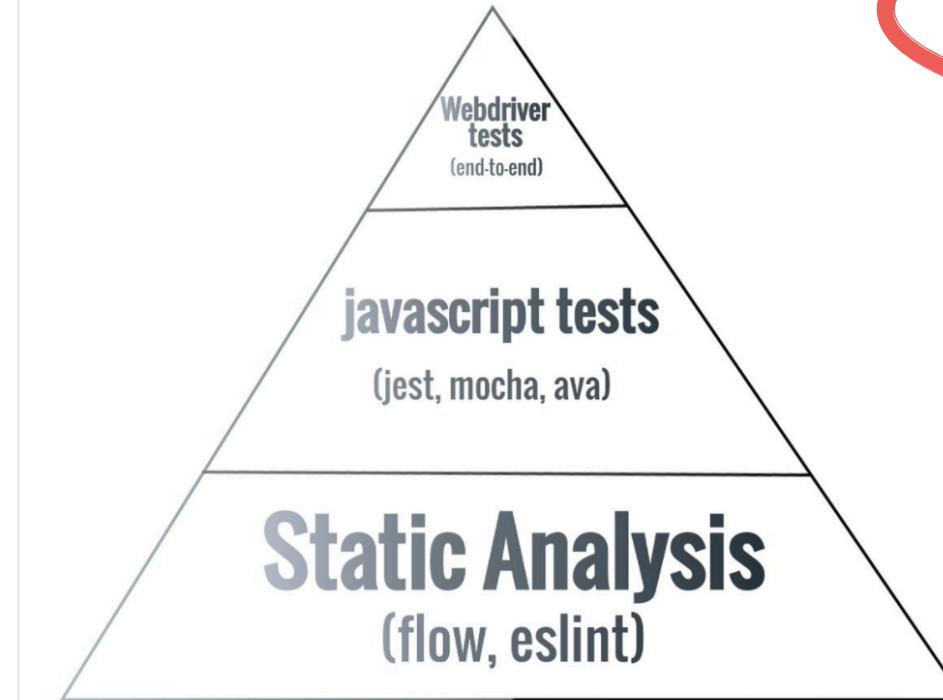
Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

 alexchow committed with facebook-github-bot [hack] Process read EOF might never arrive when child forks a daemon ...	Latest commit 77ce66f 20 hours ago
 examples [flow docs] Main page, "Getting Started", & "Five Simple Examples" re...	11 months ago
 flow-typed ./tool test --watch	6 months ago
 hack [hack] Process read EOF might never arrive when child forks a daemon	20 hours ago
 js Add emoji to server status messages	4 days ago
 lib \$ReadOnlyArray<+T> - a supertype for all tuples and arrays	3 days ago
 newtests Switch call arguments to be a `call_arg` list	2 days ago
 npm-flow-lib `flow-lib`: JS interfaces for Flow APIs	5 months ago
 resources [travis] use node6 when deploying	2 months ago
 scripts Gracefully handle no git or hg on Windows	6 months ago
 src fix returning complex types from constructors	a day ago
 tests fix returning complex types from constructors	a day ago
 tsrc Avoid 80char lint errors in tool tests & fix flow check	2 days ago
 website \$ReadOnlyArray<+T> - a supertype for all tuples and arrays	3 days ago
 .flowconfig Add npm-flow-lib to .flowconfig's ignore	4 months ago

Flow Retweeted

Dmitrii Abramov @abramov_dmitrii · 5 Dec 2016

testing pyramid in 2016



11 198 313 ...

Flow Retweeted

Jose 🙌😎🇺🇸🇲🇽 @jacortinas · 11 Nov 2016

I'm becoming a bigger and bigger fan of @flowtype the more I use it.
Seriously useful tool.

```
type Props = {  
  subreddit?: ?string;  
  filter: 'hot' | 'new' | 'controversial';  
}  
  
export default class SubredditListingScreen extends Component {  
  props: Props;  
  
  static defaultProps: Props = {  
    filter: 'hot'  
  };  
  render() {  
    return (  
      <View style={styles.container}>  
        <Text>{this.props.filter}</Text>  
      </View>  
    );  
  }  
}
```

1 3 18 ...



Shane Cavaliere
@shanejav

Follow

The better I get with @flowtype, the less of it I end up having to write. I'm loving how good it is at type inference.

RETWEETS

8

19



Brendan
@irvinebroque

Follow

Exact object types in @flowtype are such an amazing refactoring tool for big apps. Worth any pain of upgrading in order to start using them.

Flow Retweeted

Dan Abramov @dan_abramov · 11 Nov 2016

I like how @flowtype can save me from null reference errors even before I add any annotations.

```
let index = -1;  
const stack = [];  
const emptyObject = require('emptyObject');  
  
function getMaskedContext(contextTypes : any) {  
  if (index === -1) {  
    return emptyObject;  
  }  
  const currentContext = stack[index];  
  const maskedContext = {};  
  for (let key in contextTypes) {  
    maskedContext[key] = currentContext[key];  
  }  
  return maskedContext;  
};  
  
exports.popContext = function() {  
  stack[index] = null;  
  index--;  
};
```

GIF

9

50

166



Flow Retweeted

codemix @codemixers · Jan 4

Just released the first version of flow-runtime, @flowtype compatible run time type system for JS codemix.github.io/flow-runtime/

6

57

100

...

Flow Retweeted

Alden Daniels @alden_daniels · 11 Nov 2016

Been using @flowtype heavily in a recent project. It's a game-changer. After a while you forget that JS isn't typed natively. So good.

1

6

37

...



Mark Dalgleish
@markdalgleish

New team member opened a PR adding @flowtype to our established React + Webpack + Babel app. Really surprised how straightforward it was.

RETWEETS

12

LIKES

75



9:06 PM - 29 Sep 2016

...

12

75

...



Flow
@flowtype

New blog post! Introducing flow-typed: Tested, versioned, community-managed libdefs & libdef tooling for Flow. flowtype.org/blog/2016/10/1...

RETWEETS

73

LIKES

112



Mihaly Csikszentmihalyi:

Flow, the secret to happiness

TED2004 · 18:55 · Filmed Feb 2004

 32 subtitle languages ?

 View interactive transcript



Share this idea



Facebook



LinkedIn



Twitter



Link



Email



Embed

3,467,268 Total views

Mihaly Csikszentmihalyi asks, "What makes a life worth living?" Noting that money cannot make us happy, he looks to those who find pleasure and lasting satisfaction in activities that bring about a state of "flow."

Are you in flow?
(If not, how can Flow help?)

**MOVE
FAST AND
~~BREAK
THINGS~~**

A photograph of Mark Zuckerberg, founder of Facebook, standing on a stage and speaking into a microphone. He is wearing a dark grey t-shirt and dark trousers. The background features a large screen with a white and light blue geometric pattern on the left, and the text "MOVE FAST WITH STABLE INFRA" in large, bold, orange capital letters on the right.

**MOVE
FAST WITH
STABLE
INFRA**

At Facebook (and most companies)...

- **Outages are costly** (money, trust)
- **Challenge:** *Enable fast evolution without breaking stuff*
- As # engineers (and LOC) grow quickly...

How do they **understand** how things fit together?

How do they **fix** bugs / evolve things?

How do they **build** new things without being afraid?

Hello, types!

Design goal 1:

Precision

Precise type checking

- Follow along the code as much as possible
- **Reduce noise**

Give useful info about types, reaching definitions, ...

Don't hide real errors in a sea of spurious errors

- **Soundness is important**

Types should be trustworthy!

(... except in a few justifiably difficult cases)

Design goal 2:
Speed

Fast type checking

- **Important not to introduce** any **latency** in workflow

Preserve illusion of quick edit-refresh cycle

Work in background, utilize idle time

- **Parallelize** as much as possible!

- **Do as little** work as possible!

Type check incrementally, reusing most of the work

Let's see some examples!

The screenshot shows the Flow IDE interface with the file `example1.js` open. The code defines a function `length` that returns the length of its argument `x`. The IDE highlights the word `length` in blue, indicating it is a type annotation or identifier.

```
// @flow
function length(x) {
  return x.length;
}
length("");
```

The IDE's status bar at the bottom shows the file name `example1.js`, the current time `7:12`, and the zoom level `100%`. It also displays the number of errors (`4`) and warnings (`0`), the language mode `Babel ES6 JavaScript`, and a notification for `5 updates`.

The screenshot shows the Flow IDE interface with the file `example2.js` open. The code defines a function `length` that returns the length of its argument. It is annotated with `// @flow`. The code editor has syntax highlighting and line numbers. The status bar at the bottom shows file statistics: 4 errors, 0 warnings, and 5 updates.

```
// @flow
function length(x) {
  return x.length;
}

length("");
length([1,2,3]);
```

File statistics: 4 errors, 0 warnings, 5 updates

The screenshot shows the Flow IDE interface with the following details:

- Title Bar:** example3.js — /Users/avik/Documents/Flow - ETH Zurich/examples
- Toolbar:** Home, example1.js, example2.js, example3.js (selected), example4.js, example5.js, example6.js, example7.js
- Code Editor:** Displays the following JavaScript code:

```
1 // @flow
2
3 function length(x) {
4     return x.length;
5 }
6
7 length("");
8 length([1,2,3]);
9 length(null);
```
- Left Sidebar:** Contains icons for Home, Recent Files, Settings, Preferences, Help, and Profile.
- Bottom Status Bar:** Includes navigation icons, file count (4), error count (0), monitor icon, file name (example3.js), timestamp (9:14), zoom level (100%), and status indicators (LF, UTF-8, Babel ES6 JavaScript, 5 updates).

What happens at run time?

example3.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js example5.js example6.js example7.js

```
// @flow
function length(x) {
    return x.length;
}
length("");
length([1,2,3]);
length(null);
```

10 null

4 0 1 example3.js 9:14 100% LF UTF-8 Babel ES6 JavaScript 5 updates

example3.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js example5.js example6.js example7.js

```
// @flow
function length(x) {
  return x.length;
}
length("");
length([1,2,3]);
length(null);
```

LF UTF-8 Babel ES6 JavaScript 5 updates

example3.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js example5.js example6.js example7.js

```
// @flow
function length(x) {
  return x.length;
}
length("");
length([1,2,3]);
length(null);
```

LF UTF-8 Babel ES6 JavaScript 5 updates

Flow “runs” the same code
statically...

example3.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js example5.js example6.js example7.js

```
// @flow
function length(x) {
  return x.length; // string | null | [number, number, number]
}
length("");
length([1,2,3]);
length(null);
```

4 0 4 1 100% LF UTF-8 Babel ES6 JavaScript 5 updates

example3.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js example5.js example6.js example7.js

```
length: (x: X) => ...
2
3 function length(x) {
4     return x.length;
5 }
6
7 length("");
8 length([1,2,3]);
9 length(null);
10 null: null
```

⚙️ ⏳ ★ 🐛 🚙

4 0 🖥 example3.js 9:14 100%

LF UTF-8 Babel ES6 JavaScript 5 updates

example3.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js example5.js example6.js example7.js

```
// @flow
function length(x) {
  return x.length;
}
length("");
length([1,2,3]);
length(null);
```

X → GetProp("length") => ...

x: X

4 0 1 example3.js 9:14 100% LF UTF-8 Babel ES6 JavaScript 5 updates

example3.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js example5.js example6.js example7.js

```
// @flow
function length(x) {
  return x.length;
}
length("");
length([1,2,3]);
length(null);
```

$X \rightarrow \text{GetProp("length")} \Rightarrow \dots$

$(x: X) \Rightarrow \dots \rightarrow \text{Call(null)} \Rightarrow \dots$

length: $(x: X) \Rightarrow \dots$ null: null

4 0 100% LF UTF-8 Babel ES6 JavaScript 5 updates

example3.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js example5.js example6.js example7.js

```
// @flow
function length(x) {
  return x.length;
}
length("");
length([1,2,3]);
length(null);
```

X → GetProp("length") => ...

null → X

4 0 1 example3.js 9:14 100% LF UTF-8 Babel ES6 JavaScript 5 updates

example3.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js example5.js example6.js example7.js

```
// @flow
function length(x) {
  return x.length;
}
length("");
length([1,2,3]);
length(null);
```

GetProp("length") => ...

null

LF UTF-8 Babel ES6 JavaScript 5 updates

example3.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js example5.js example6.js example7.js

```
// @flow
function length(x) {
  return x.length;
}
length("");
length([1,2,3]);
length(null);
```

GetProp("length") => ...

null

Error!

4 0 1 example3.js 9:14 100% LF UTF-8 Babel ES6 JavaScript 5 updates

Let's fix this code!

The screenshot shows the Flow IDE interface with the file `example4.js` open. The code defines a function `length` that returns the length of a given argument. It handles `null` by returning `-1`. The code also includes three calls to `length` with different arguments: an empty string, an array, and `null`. The IDE's status bar at the bottom indicates 4 errors and 0 warnings.

```
// @flow
function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}
length("");
length([1,2,3]);
length(null);
```

File menu: Home, example1.js, example2.js, example3.js, example4.js (selected), example5.js, example6.js, example7.js

Toolbar icons: Home, Example 1, Example 2, Example 3, Example 4 (selected), Example 5, Example 6, Example 7

Status bar: 4 errors, 0 warnings, example4.js, 6:4, 100%, LF, UTF-8, Babel ES6 JavaScript, 5 updates

example4.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js **example4.js** example5.js example6.js example7.js

```
1 // @flow
2
3 function length(x) {
4     if (x == null) {
5         return -1;
6     }
7     return x.length;
8 }
9
10 length("");
11 length([1,2,3]);
12 length(null);
```

LF UTF-8 Babel ES6 JavaScript 5 updates

example4.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js **example4.js** example5.js example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

length("");
length([1,2,3]);
length(null);
```

```
graph LR
    X((X)) --> R1[Refine("== null") => X1]
    X --> R2[Refine("!= null") => X2]
    X2((X2)) --> GP[GetProp("length") => ...]
    null((null)) --> X
```

example4.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js **example4.js** example5.js example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

length("");
length([1,2,3]);
length(null);
```

Diagram illustrating Flow type annotations:

- The variable `x` is annotated with `@flow`.
- In the function `length`, the parameter `x` is annotated with `null`. This leads to two refinement paths:
 - `Refine("== null") => X1`
 - `Refine("!= null") => X2`
- The value `null` is annotated with `X2`.
- The expression `x.length` is annotated with `GetProp("length") => ...`.

Bottom status bar: 4 0 example4.js 6:4 100% LF UTF-8 Babel ES6 JavaScript 5 updates

example4.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js **example4.js** example5.js example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

length("");
length([1,2,3]);
length(null);
```

Diagram illustrating the flow analysis for the variable `x`:

- `null` flows to `X1`.
- `X1` flows to `X2`.
- `X2` flows to `GetProp("length") => ...`.

File: example4.js | Line: 6 | Column: 1 | 4 errors | 0 warnings | 6:4 | 100% | LF | UTF-8 | Babel ES6 JavaScript | 5 updates

Ready for some more code?

The screenshot shows a code editor window titled "example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples". The editor interface includes a toolbar with icons for file operations, a navigation bar with tabs for "Home", "example1.js", "example2.js", "example3.js", "example4.js", "example5.js" (which is active), "example6.js", and "example7.js", and a sidebar on the left with various icons.

```
// @flow
function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

The code uses Flow.js annotations, such as the `@flow` comment at the top and the `length` annotation on the object passed to `printItems`. The editor highlights these annotations in red and blue respectively. A red arrow icon is placed before the line containing the annotated object, likely indicating a warning or error. The status bar at the bottom shows the file name "example5.js", the current time "17:40", and the file encoding "96%".

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y); number | string ×
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

LF UTF-8 Babel ES6 JavaScript 5 updates

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

X → **Refine("!= null") => X2**

X2 → **GetProp("length") => ...**

2 0 17:40 96% LF UTF-8 Babel ES6 JavaScript 5 updates

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

X → **Refine("!= null") => X2**

X2 → **GetProp("length") => ...**

y: Y

Y → **X**

2 0 17:40 96% LF UTF-8 Babel ES6 JavaScript 5 updates

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

y: Y
```

$X \rightarrow \text{Refine}("!= \text{null}") \Rightarrow X_2$

$X_2 \rightarrow \text{GetProp}("length") \Rightarrow \dots$

$Y \rightarrow X$

$\{ \text{length}: \text{string} \} \rightarrow Y$

2 0 17:40 96% LF UTF-8 Babel ES6 JavaScript 5 updates

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

X → **Refine("!= null") => X2**

X2 → **GetProp("length") => ...**

X

{length: string}

2 0 17:40 96% LF UTF-8 Babel ES6 JavaScript 5 updates

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

X2 → **Refine("!= null") => X2**

GetProp("length") => ...

{length: string}

LF UTF-8 Babel ES6 JavaScript 5 updates

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

X2 → GetProp("length") => ...

{length: string}

LF UTF-8 Babel ES6 JavaScript 5 updates

The diagram illustrates the flow of the 'length' property. A yellow rounded rectangle labeled 'X2' has an arrow pointing to another yellow rounded rectangle containing the text 'GetProp("length") => ...'. A curved orange arrow originates from the bottom right of the 'GetProp' box and points back to the 'length' property in the code at line 17.

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

GetProp("length") => ...

{length: string}

LF UTF-8 Babel ES6 JavaScript 5 updates

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

(result): R2

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

GetProp("length") => R2

{length: string}

LF UTF-8 Babel ES6 JavaScript 5 updates

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow
function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}
function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}
printItems({ length: 'not a number' });

```

(return): R

$R_2 \rightarrow R$

GetProp("length") => R_2

$\{length: string\}$

LF UTF-8 Babel ES6 JavaScript 5 updates

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow
function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

(return): R

R2 → R

GetProp("length") => R2

R → number

{length: string}

2 0 17:40 96% LF UTF-8 Babel ES6 JavaScript 5 updates

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow
function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

(return): R

R2 → R

R2

R → number

string

LF UTF-8 Babel ES6 JavaScript 5 updates

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

R → **number**

R → **string**

2 0 17:40 96% LF UTF-8 Babel ES6 JavaScript 5 updates

```
graph LR; R1[R] --> number["number"]; R2[R] --> string["string"]
```

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js example5.js example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

number

string

LF UTF-8 Babel ES6 JavaScript 5 updates

example5.js — /Users/avik/Documents/Flow - ETH Zurich/examples

Home example1.js example2.js example3.js example4.js **example5.js** example6.js example7.js

```
// @flow

function length(x) {
  if (x == null) {
    return -1;
  }
  return x.length;
}

function printItems(y) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}

printItems({ length: 'not a number' });

```

Error!

number

string

LF UTF-8 Babel ES6 JavaScript 5 updates

Gah, let's pin down the API...

The screenshot shows a code editor window titled "example6.js — /Users/avik/Documents/Flow - ETH Zurich/examples". The left sidebar contains icons for Home, example1.js, example2.js, example3.js, example4.js, example5.js, example6.js (selected), and example7.js. The main editor area displays the following JavaScript code:

```
// @flow
function length(x) {
    return x.length;
}

export function printItems(y: string | Array<string>) {
    const len = length(y);
    for (let i = 0; i < len; i++) {
        console.log(`element ${i} is ${y[i]}`);
    }
}
```

The code uses Flow type annotations. The `length` function is annotated with `@flow`. The `printItems` function is annotated with `y: string | Array<string>`, indicating that it can accept either a single string or an array of strings. The `console.log` statement uses template literals to print each element of the array.

The screenshot shows the Flow IDE interface with the title bar "example7.js — /Users/avik/Documents/Flow - ETH Zurich/examples". The left sidebar contains icons for Home, example1.js, example2.js, example3.js, example4.js, example5.js, example6.js, and example7.js (which is currently selected). The main editor area displays the following code:

```
// @flow
import { printItems } from './example6';
printItems( (y: string | Array<string>) => void )
```

The word "printItems" is highlighted in red, indicating it is a reference to another file. The parameter "(y: string | Array<string>)" is highlighted in blue, indicating its type. A tooltip or completion dropdown is visible over the parameter, showing the type "y: string | Array<string>". The status bar at the bottom shows "example7.js*" with 2 errors and 0 warnings, and the file is saved in Babel ES6 JavaScript.

property variance object spreads

optional properties functions as predicates extensible objects

exact objects open methods

Wide support for JavaScript idioms

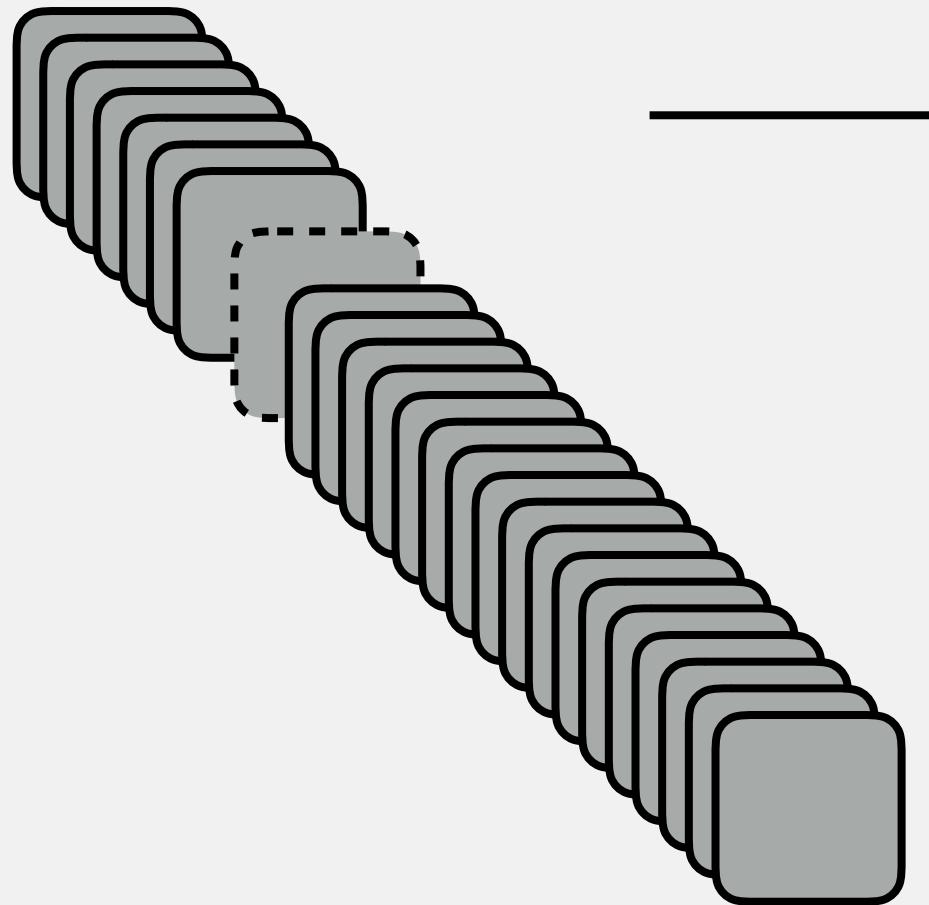
tuples disjoint unions type parameter variance

optional parameters literal types bounded generics

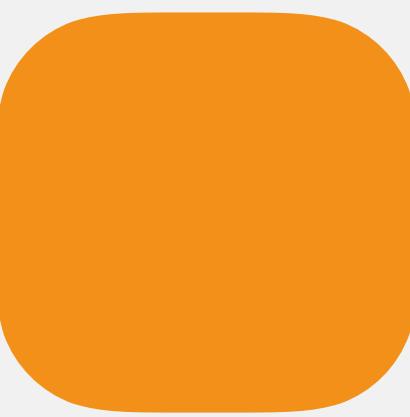
function overloading type transforms

Design for responsiveness

File system



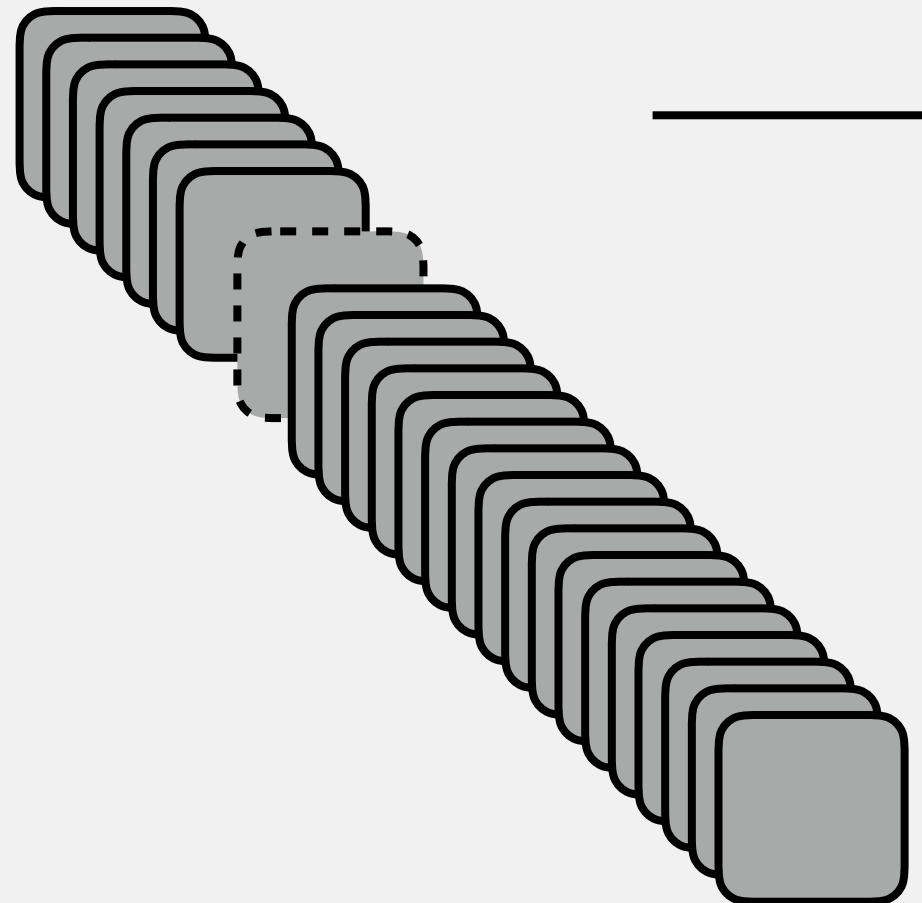
Server



init



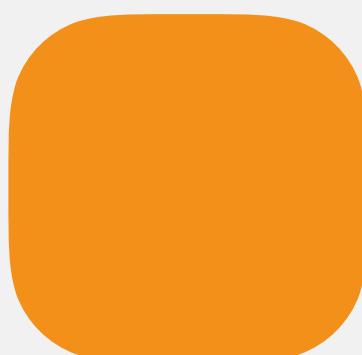
File system



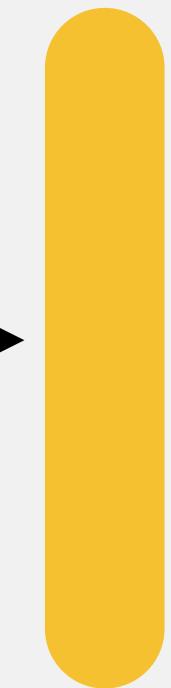
Server



init



Client



cmd

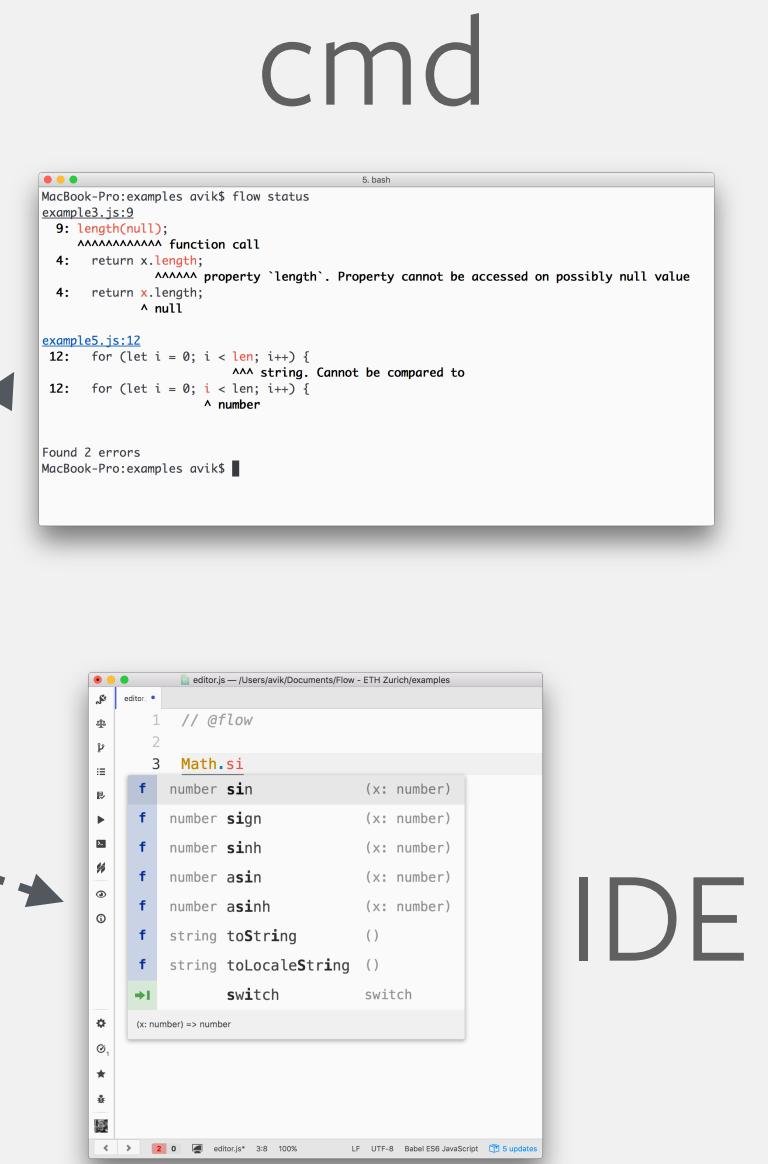
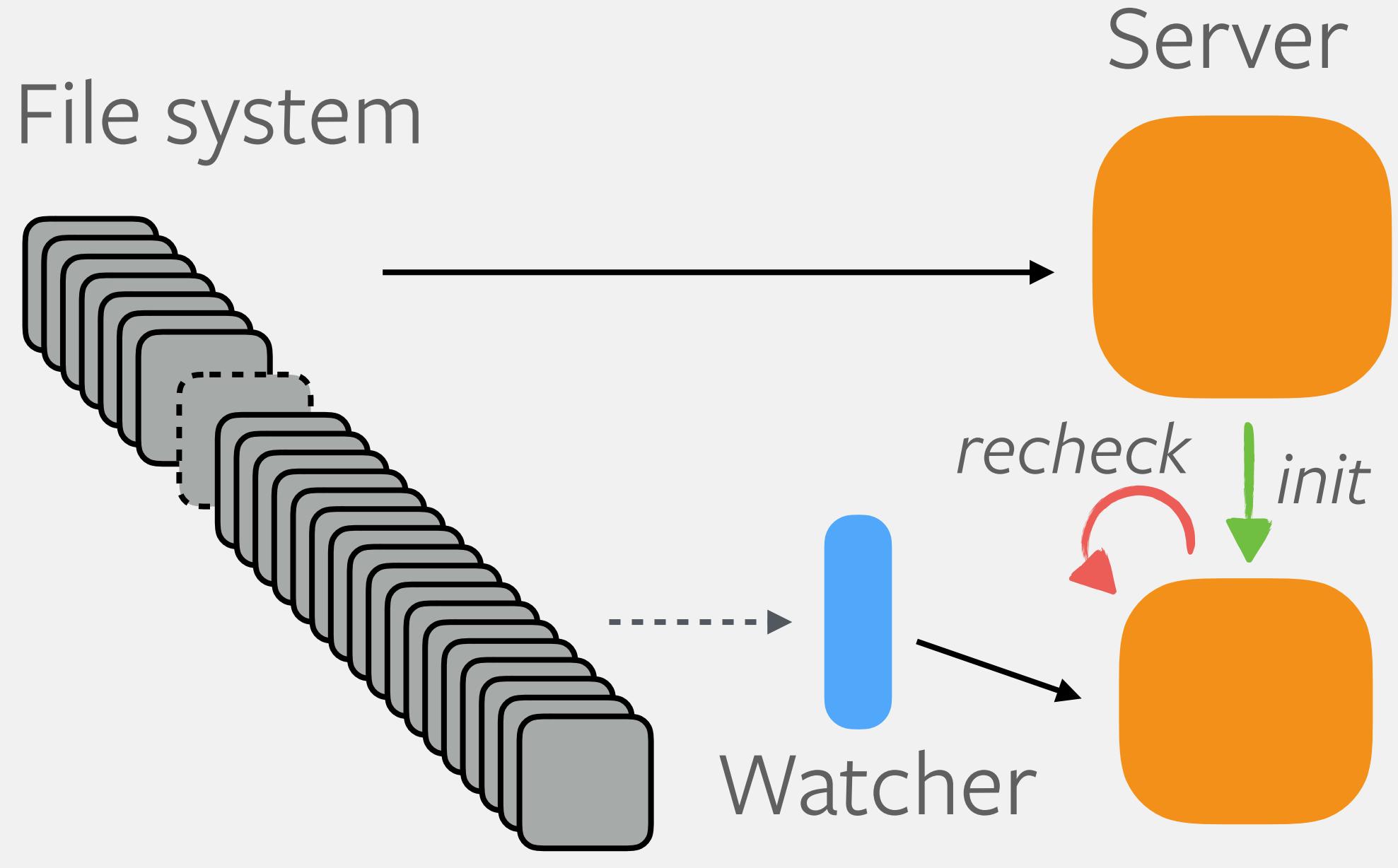
```
MacBook-Pro:examples avik$ flow status
example3.js:9
9: length:null;
         ^^^^^^^^^^^ function call
4:   return x.length;
        ^^^^^^ property `length`. Property cannot be accessed on possibly null value
4:   return x.length;
         ^ null

example5.js:12
12:  for (let i = 0; i < len; i++) {
           ^^^ string. Cannot be compared to
12:  for (let i = 0; i < len; i++) {
           ^ number

Found 2 errors
MacBook-Pro:examples avik$
```

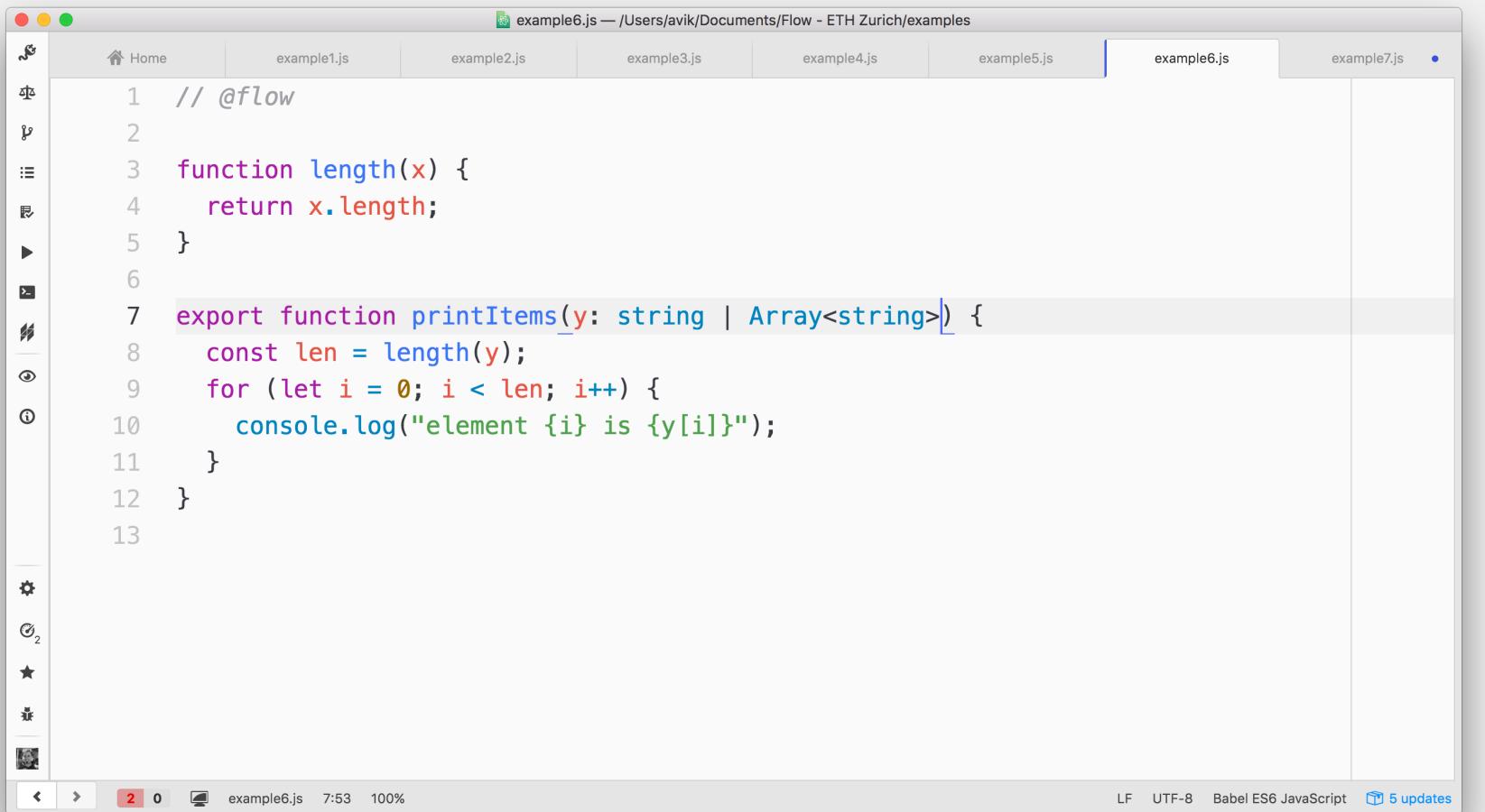
```
// @flow
Math.sin
number sin (x: number)
number sign (x: number)
number sinh (x: number)
number asin (x: number)
number asinh (x: number)
string toString ()
string toLocaleString ()
switch switch
(x: number) => number
```

IDE



Initializing...

example6.js



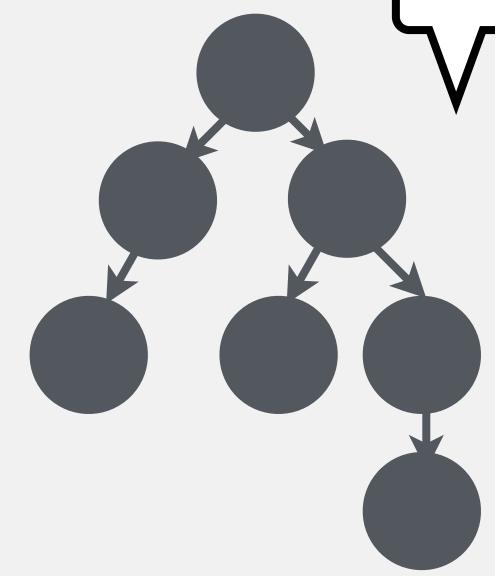
The screenshot shows the Flow IDE interface with the tab 'example6.js' selected. The code editor contains the following JavaScript code:

```
// @flow
function length(x) {
  return x.length;
}

export function printItems(y: string | Array<string>) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log("element {i} is {y[i]}");
  }
}
```

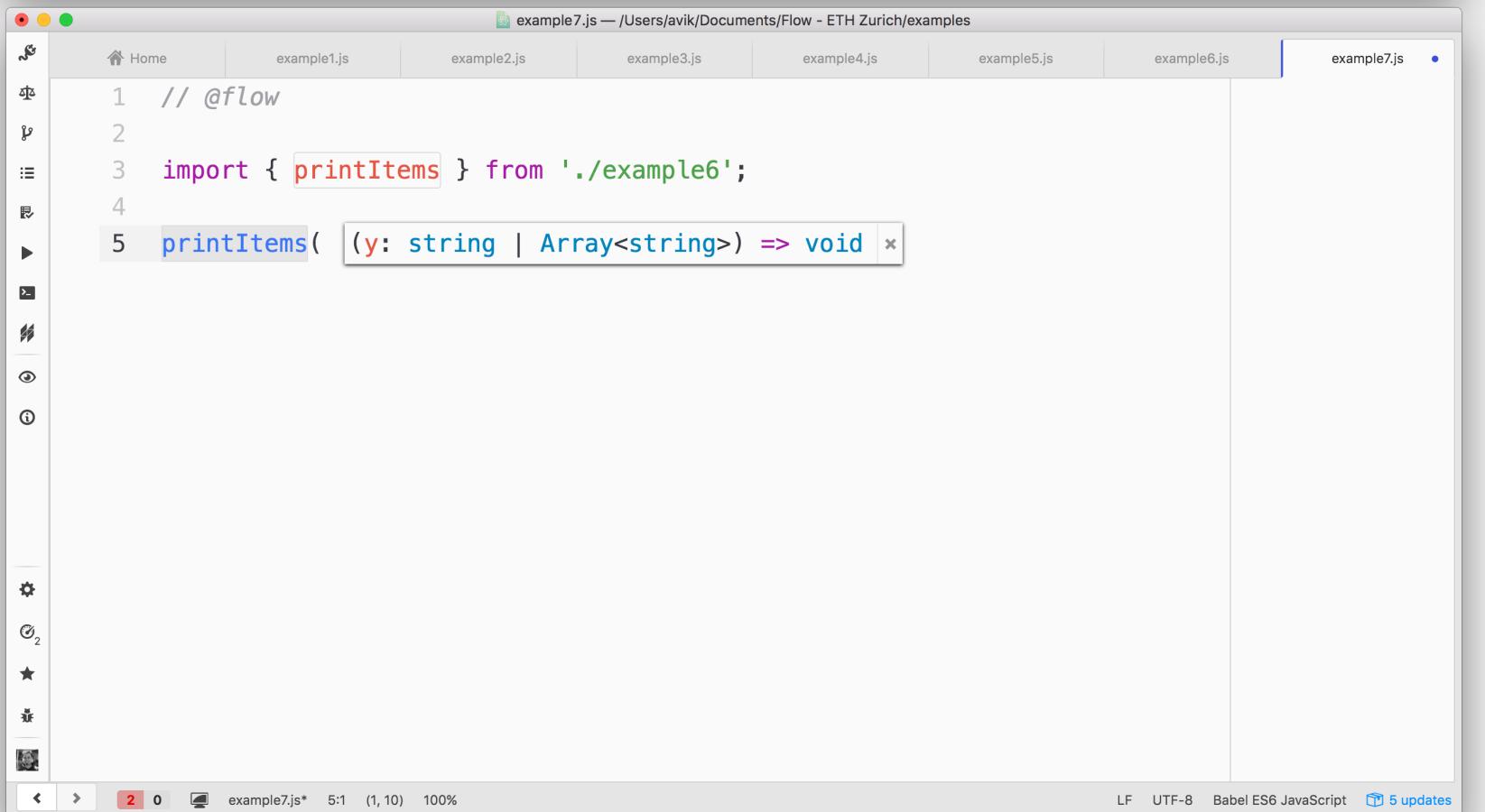
The status bar at the bottom indicates 'example6.js 7:53 100%' and 'LF UTF-8 Babel ES6 JavaScript 5 updates'.

Parse



AST

example7.js

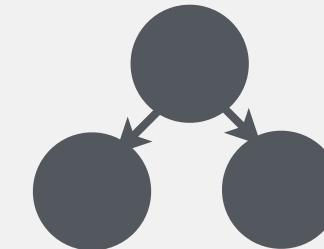


The screenshot shows the Flow IDE interface with the tab 'example7.js' selected. The code editor contains the following JavaScript code:

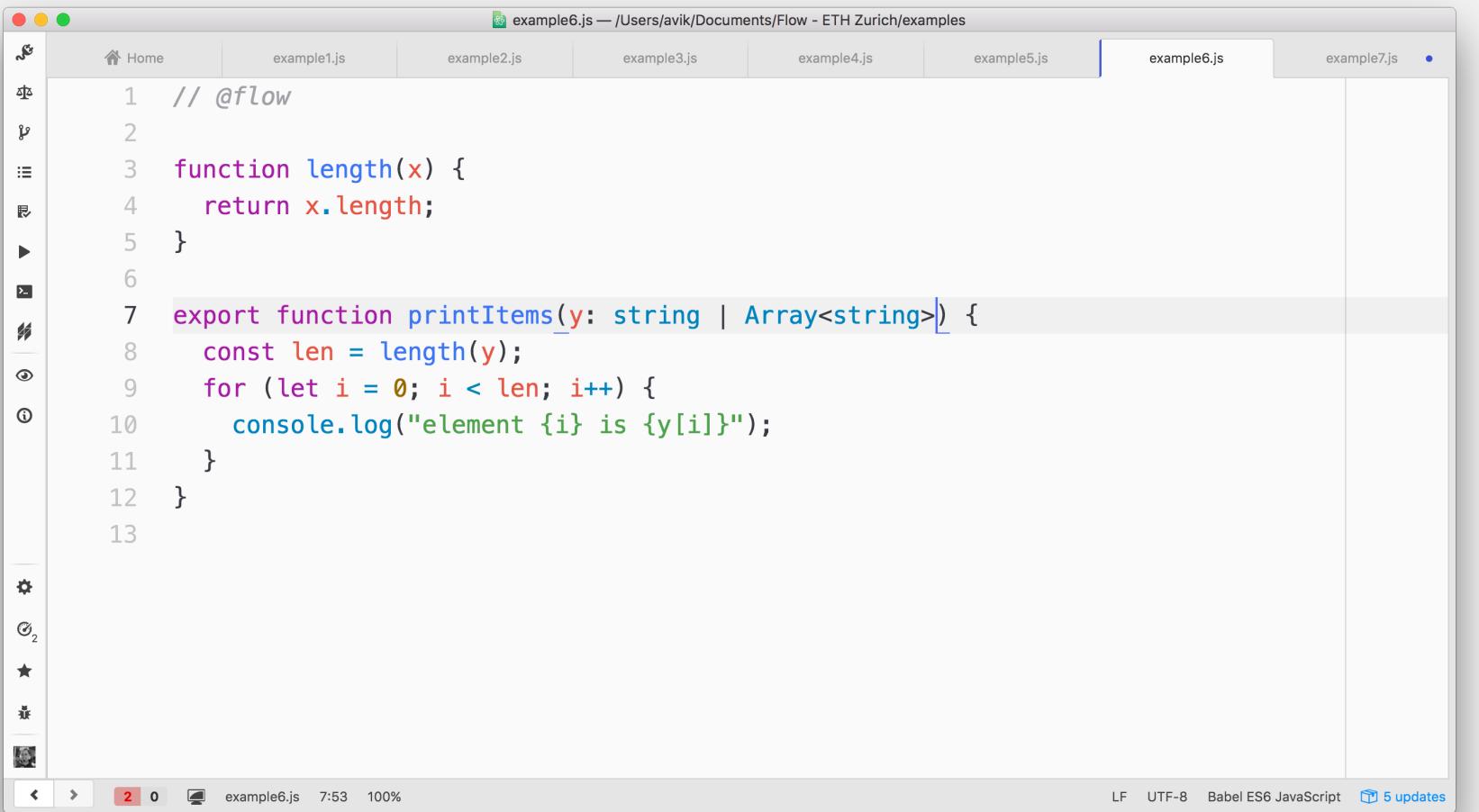
```
// @flow
import { printItems } from './example6';
printItems( (y: string | Array<string>) => void )
```

The status bar at the bottom indicates 'example7.js* 5:1 (1, 10) 100%' and 'LF UTF-8 Babel ES6 JavaScript 5 updates'.

Parse



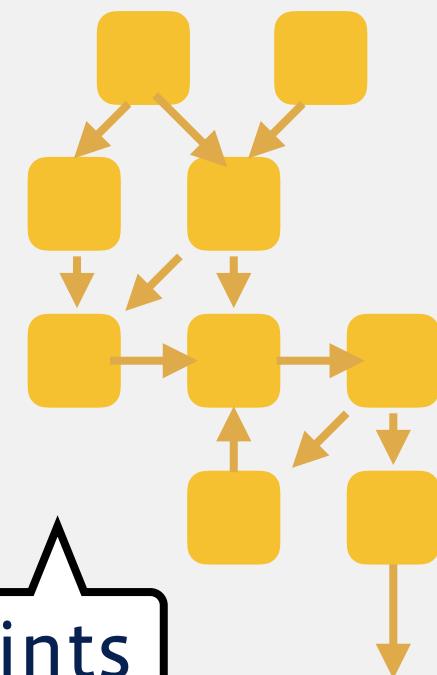
example6.js



```
// @flow
function length(x) {
  return x.length;
}

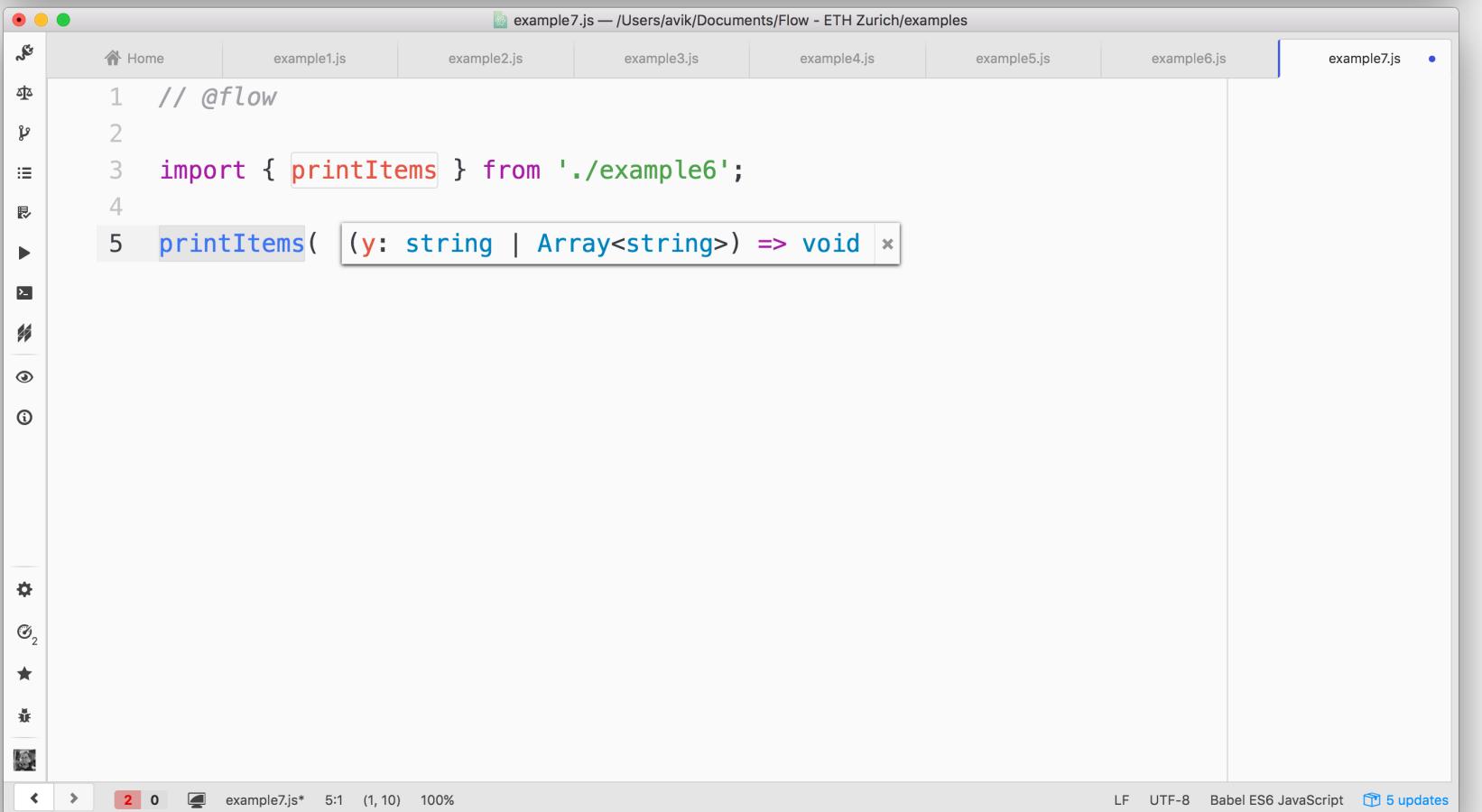
export function printItems(y: string | Array<string>) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log(`element ${i} is ${y[i]}`);
  }
}
```

Compile



Constraints

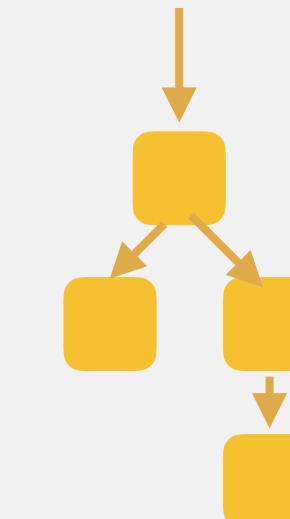
example7.js



```
// @flow
import { printItems } from './example6';

printItems( (y: string | Array<string>) => void )
```

Compile



Constraints

```
// @flow
function length(x) {
  return x.length;
}

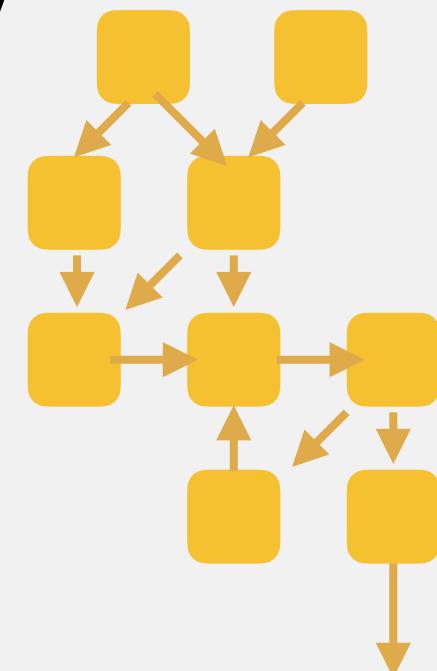
export function printItems(y: string | Array<string>) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log("element " + i + " is " + y[i]);
  }
}

// @flow
import { printItems } from './example6';
printItems( (y: string | Array<string>) => void )
```

example6.js

Link

example7.js



```
// @flow
function length(x) {
  return x.length;
}

export function printItems(y: string | Array<string>) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log("element {i} is {y[i]}");
  }
}

// @flow
import { printItems } from './example6';
printItems( (y: string | Array<string>) => void )
```

example6.js

Link

Signature

example7.js

```

example6.js — /Users/avik/Documents/Flow - ETH Zurich/examples
1 // @flow
2
3 function length(x) {
4   return x.length;
5 }
6
7 export function printItems(y: string | Array<string>) {
8   const len = length(y);
9   for (let i = 0; i < len; i++) {
10     console.log("element {i} is {y[i]}");
11   }
12 }
13

example7.js — /Users/avik/Documents/Flow - ETH Zurich/examples
1 // @flow
2
3 import { printItems } from './example6';
4
5 printItems( (y: string | Array<string>) => void )

```

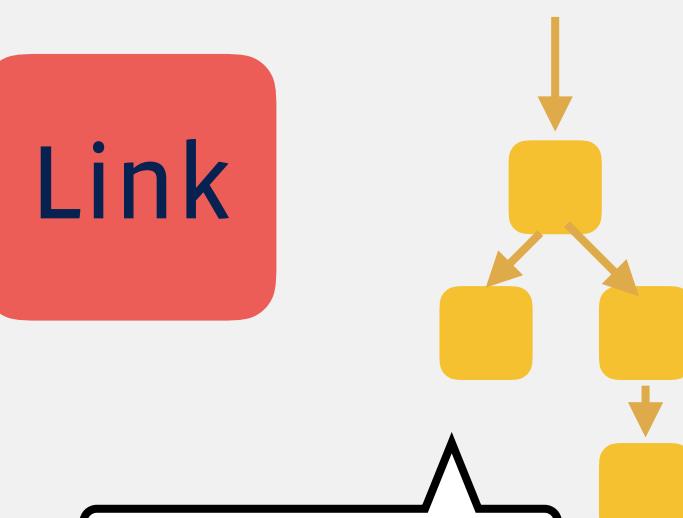
example6.js

Signature

example7.js

Link

Constraints



The image shows two separate code editors side-by-side, both titled "example6.js" and "example7.js".

example6.js:

```
// @flow
function length(x) {
  return x.length;
}

export function printItems(y: string | Array<string>) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log("element {i} is {y[i]}");
  }
}
```

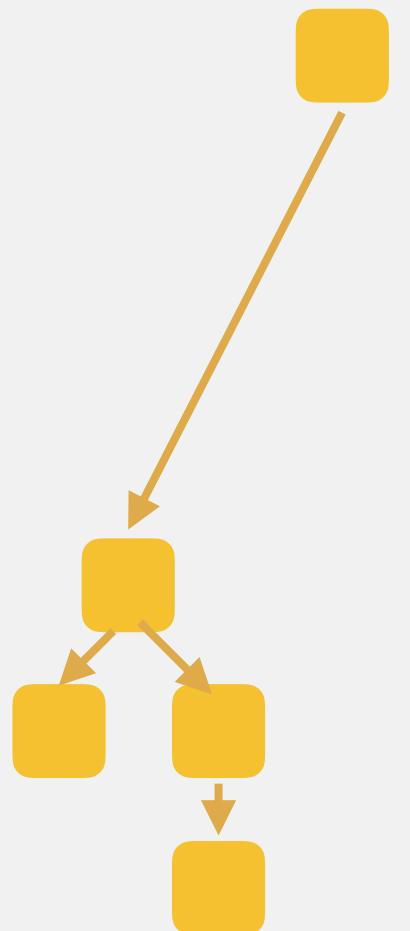
example7.js:

```
// @flow
import { printItems } from './example6';
printItems( (y: string | Array<string>) => void )
```

example6.js

example7.js

Link



Rechecking...



example6.js — /Users/avik/Documents/Flow - ETH Zurich/examples

```
1 // @flow
2
3 function length(x) {
4   return x.length;
5 }
6
7 export function printItems(y: string | Array<string>) {
8   const len = length(y);
9   for (let i = 0; i < len; i++) {
10     console.log("element {i} is {y[i]}");
11   }
12 }
13
```

example7.js — /Users/avik/Documents/Flow - ETH Zurich/examples

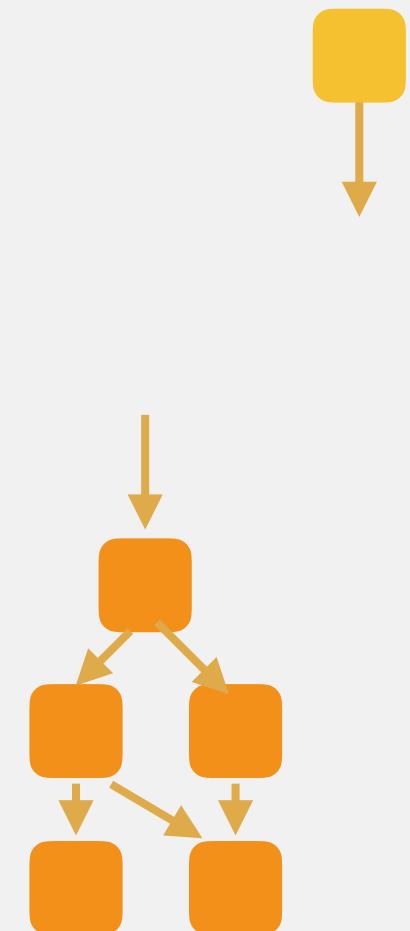
```
1 // @flow
2
3 import { printItems } from './example6';
4
5 printItems( (y: string | Array<string>) => void )
```

example6.js

example7.js

Parse

Compile



EDIT

```
// @flow
function length(x) {
  return x.length;
}

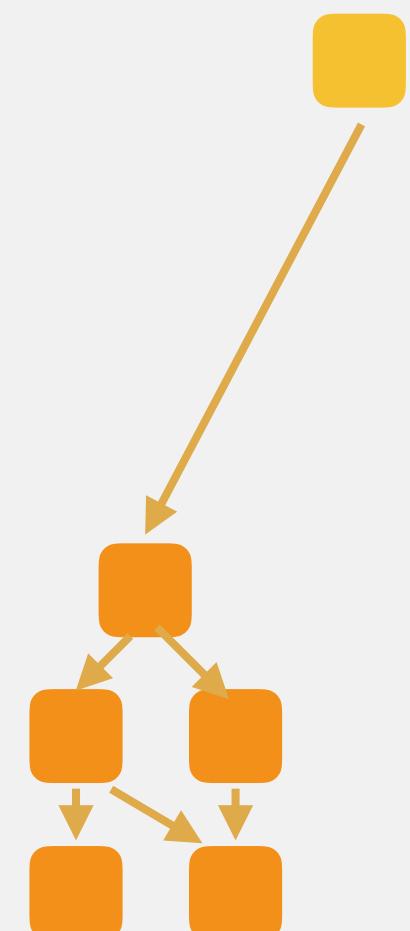
export function printItems(y: string | Array<string>) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log("element {i} is {y[i]}");
  }
}
```

```
// @flow
import { printItems } from './example6';
printItems( (y: string | Array<string>) => void )
```

example6.js

example7.js

Link



EDIT

```
// @flow

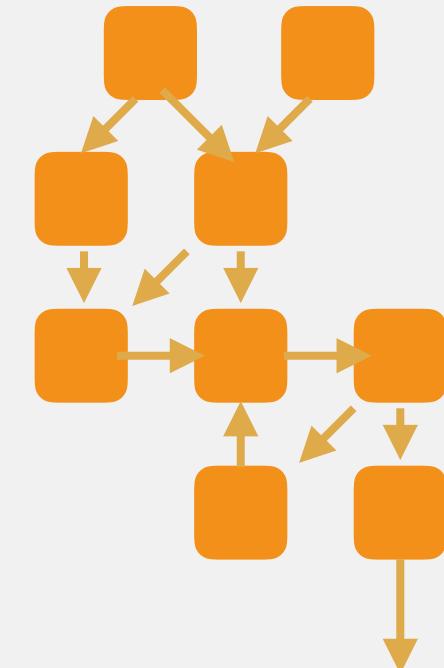
function length(x) {
  return x.length;
}

export function printItems(y: string | Array<string>){
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log("element {i} is {y[i]}");
  }
}
```

example6.js

Parse

Compile

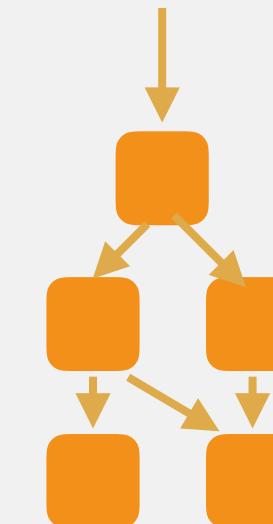


example7.js

```
// @flow

import { printItems } from './example6';

printItems( (y: string | Array<string>) => void )
```



```
// @flow
function length(x) {
  return x.length;
}

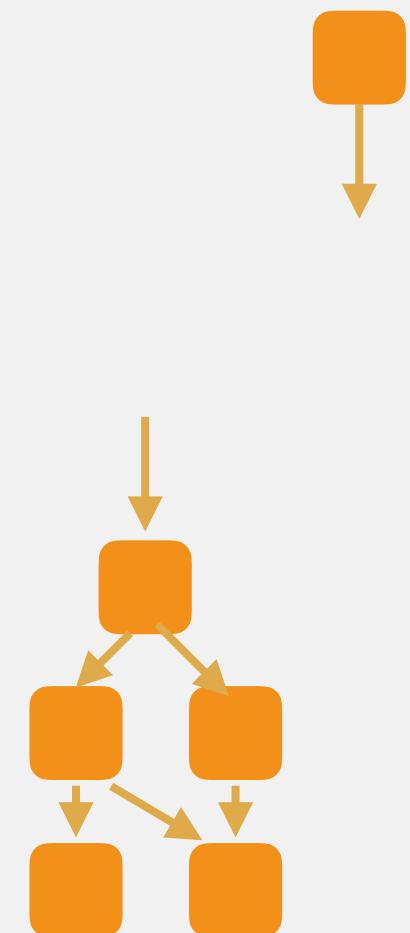
export function printItems(y: string | Array<string>){
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log("element {i} is {y[i]}");
  }
}

// @flow
import { printItems } from './example6';
printItems( (y: string | Array<string>) => void )
```

example6.js



example7.js



```
// @flow
function length(x) {
  return x.length;
}

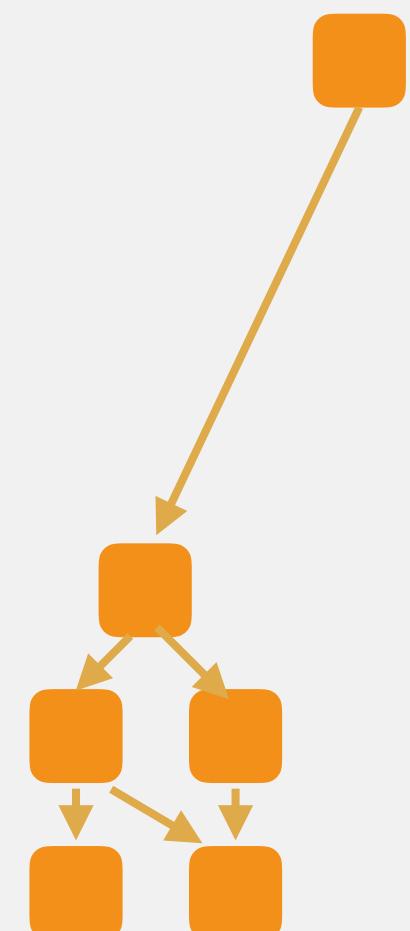
export function printItems(y: string | Array<string>){
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log("element {i} is {y[i]}");
  }
}

// @flow
import { printItems } from './example6';
printItems( (y: string | Array<string>) => void )
```

example6.js

example7.js

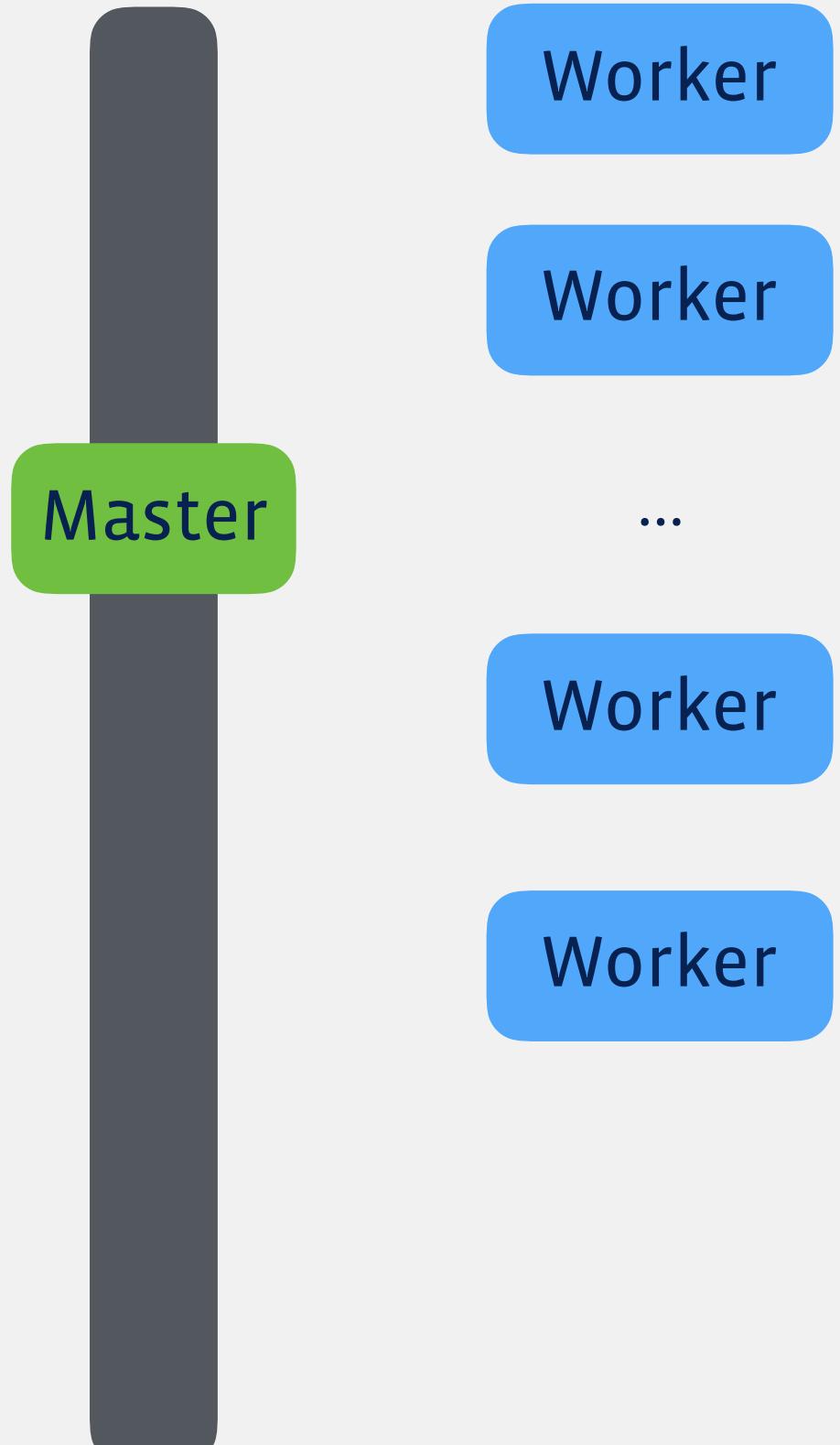
Link



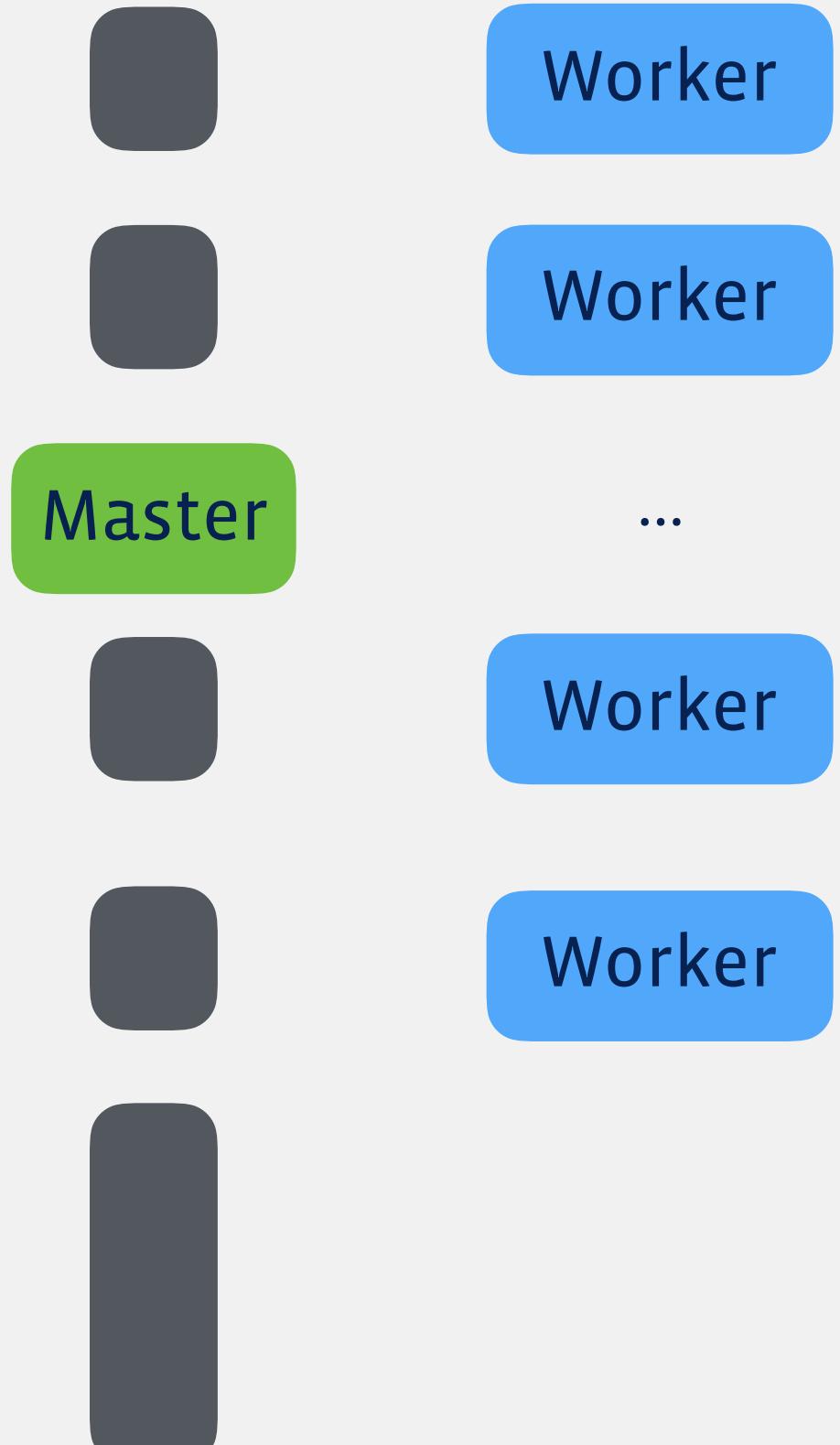
Parallelize everything!

Infrastructure

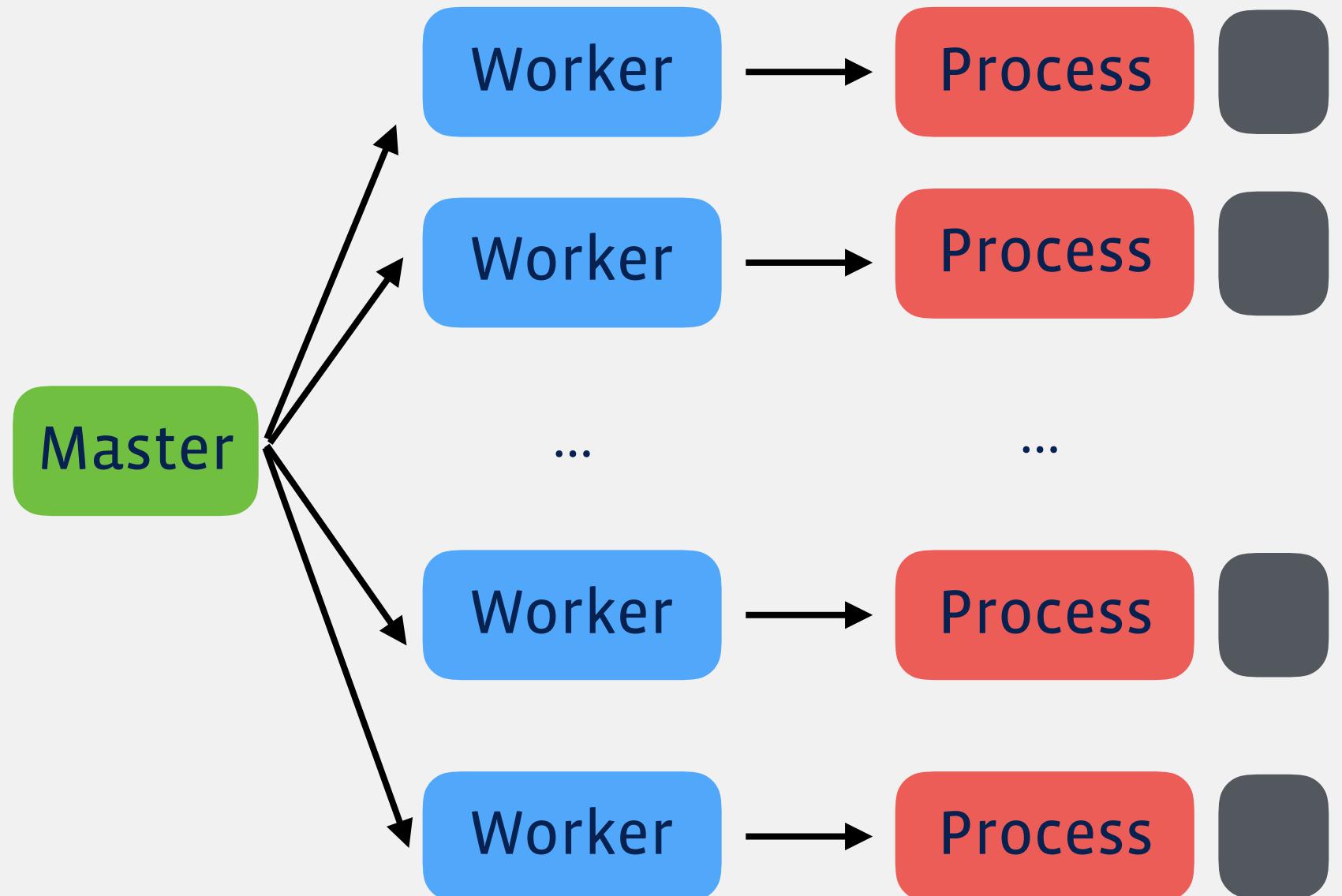
large number of jobs

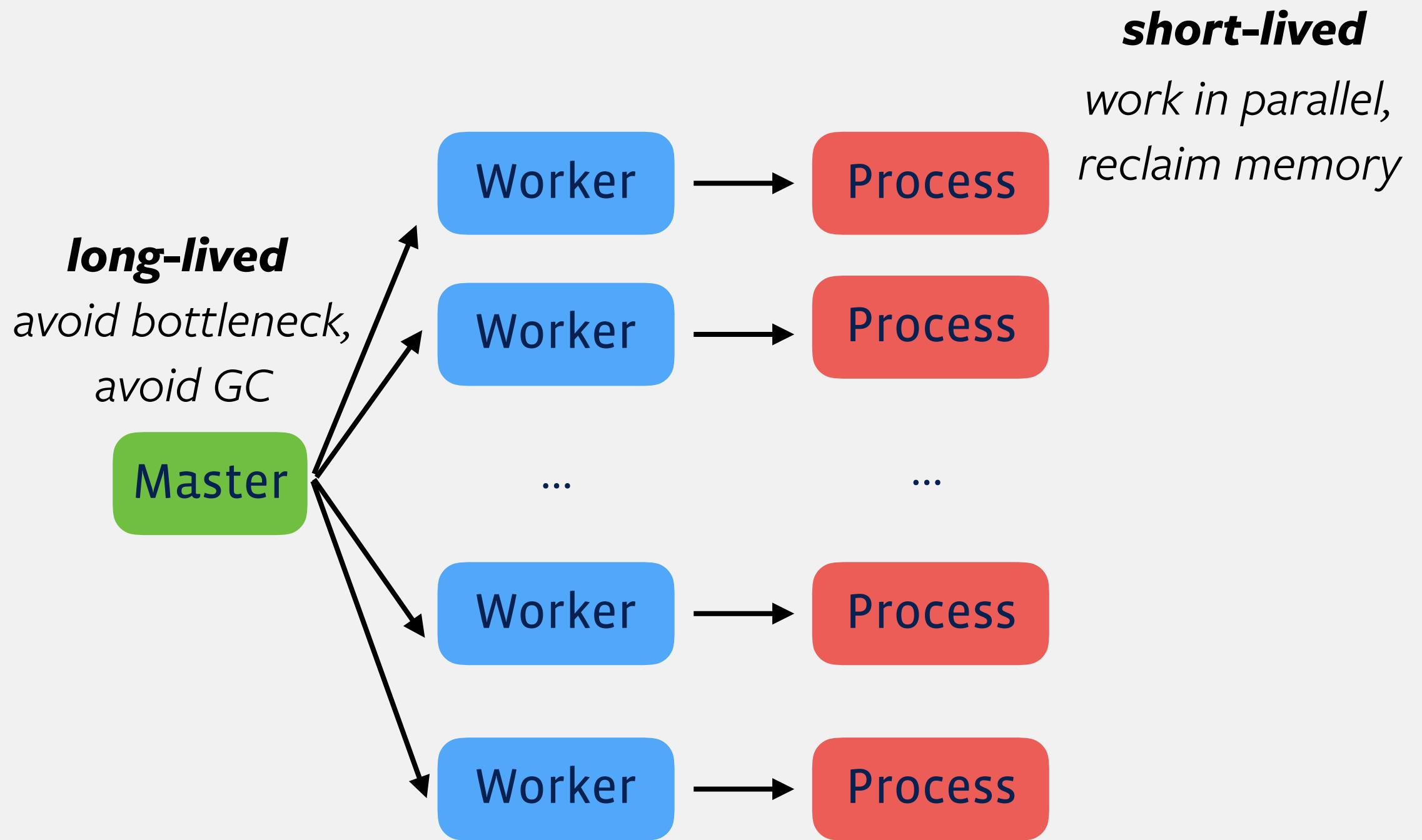


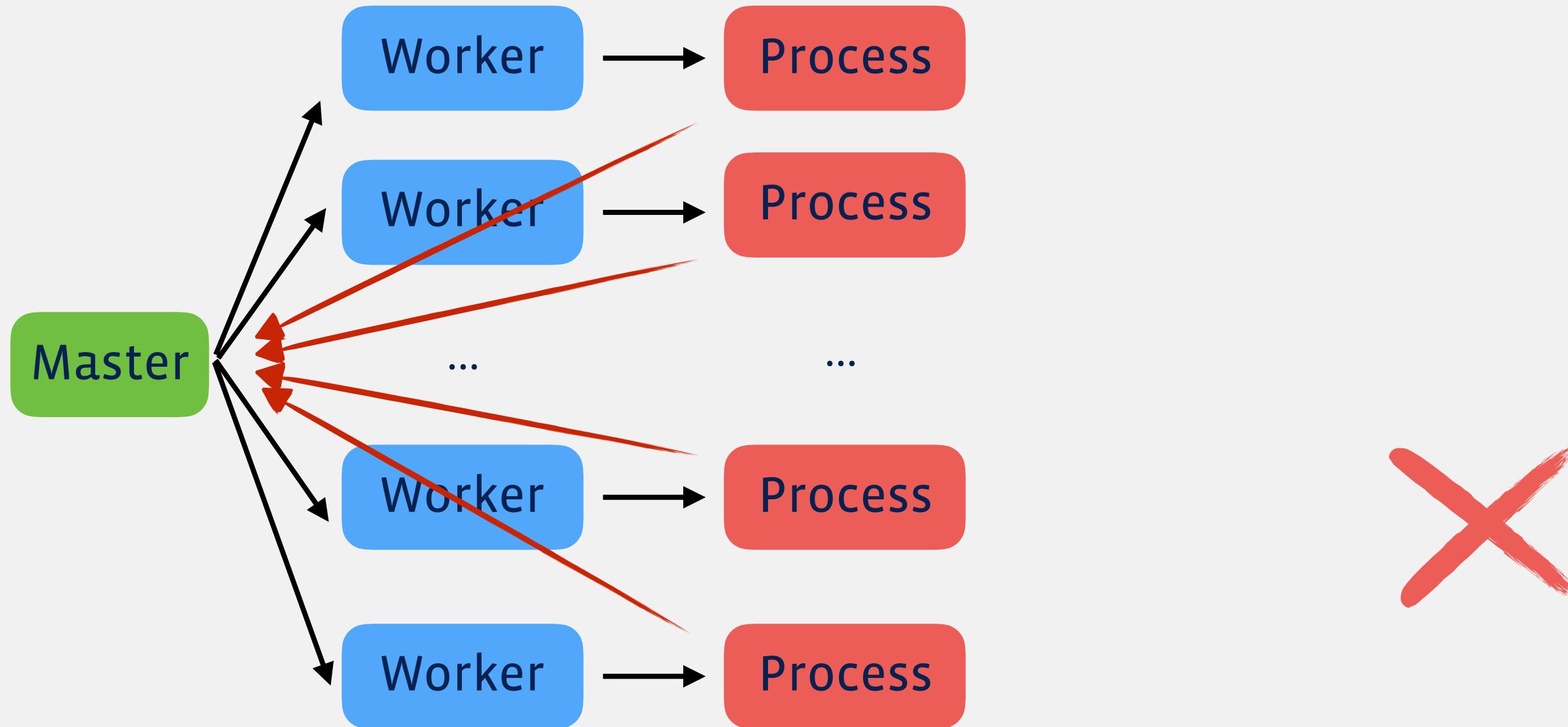
divide into buckets

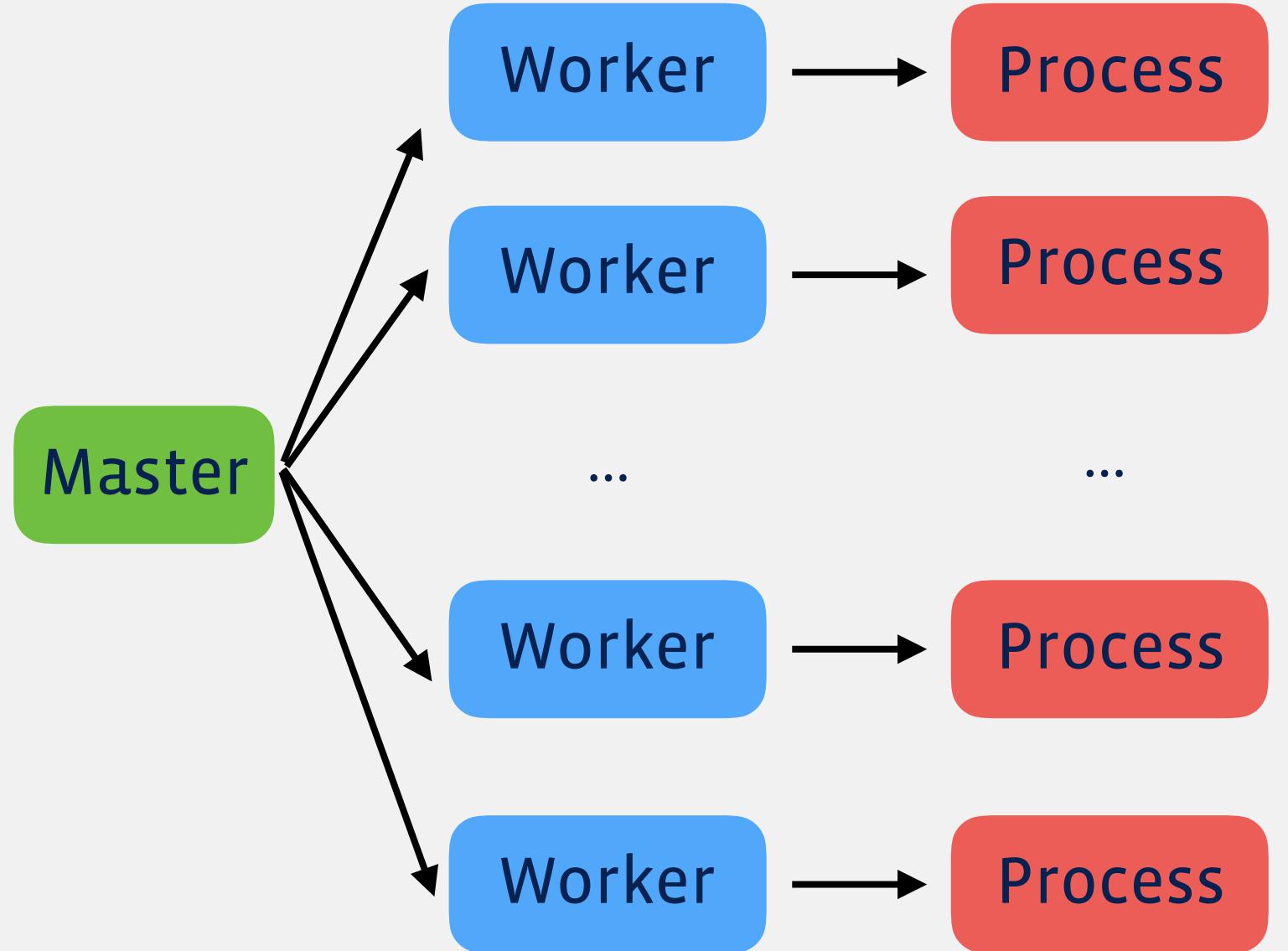


distribute!

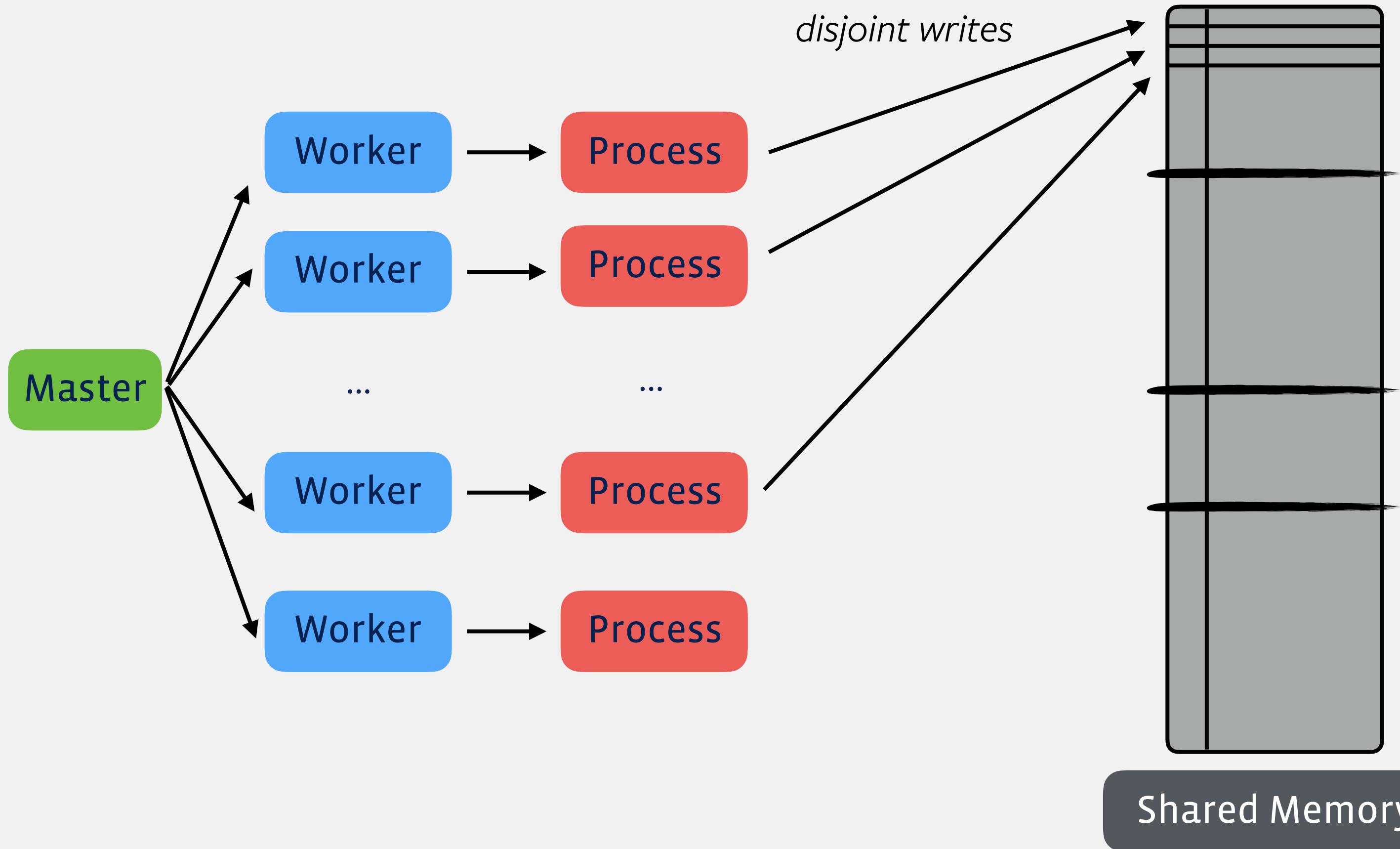


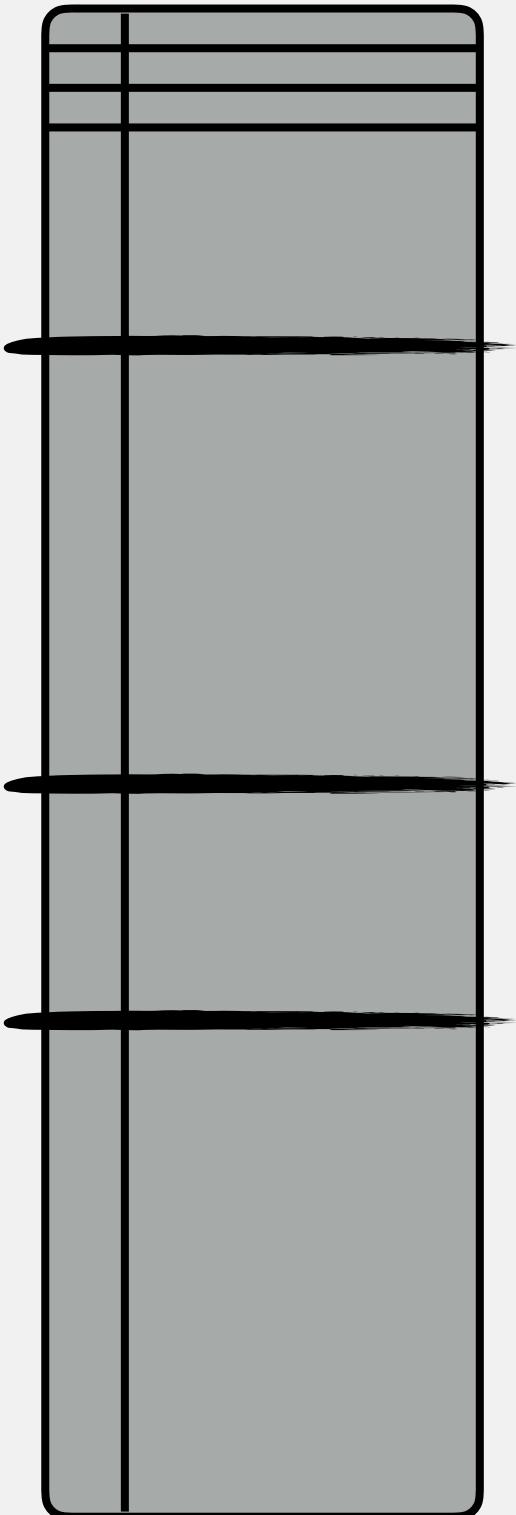
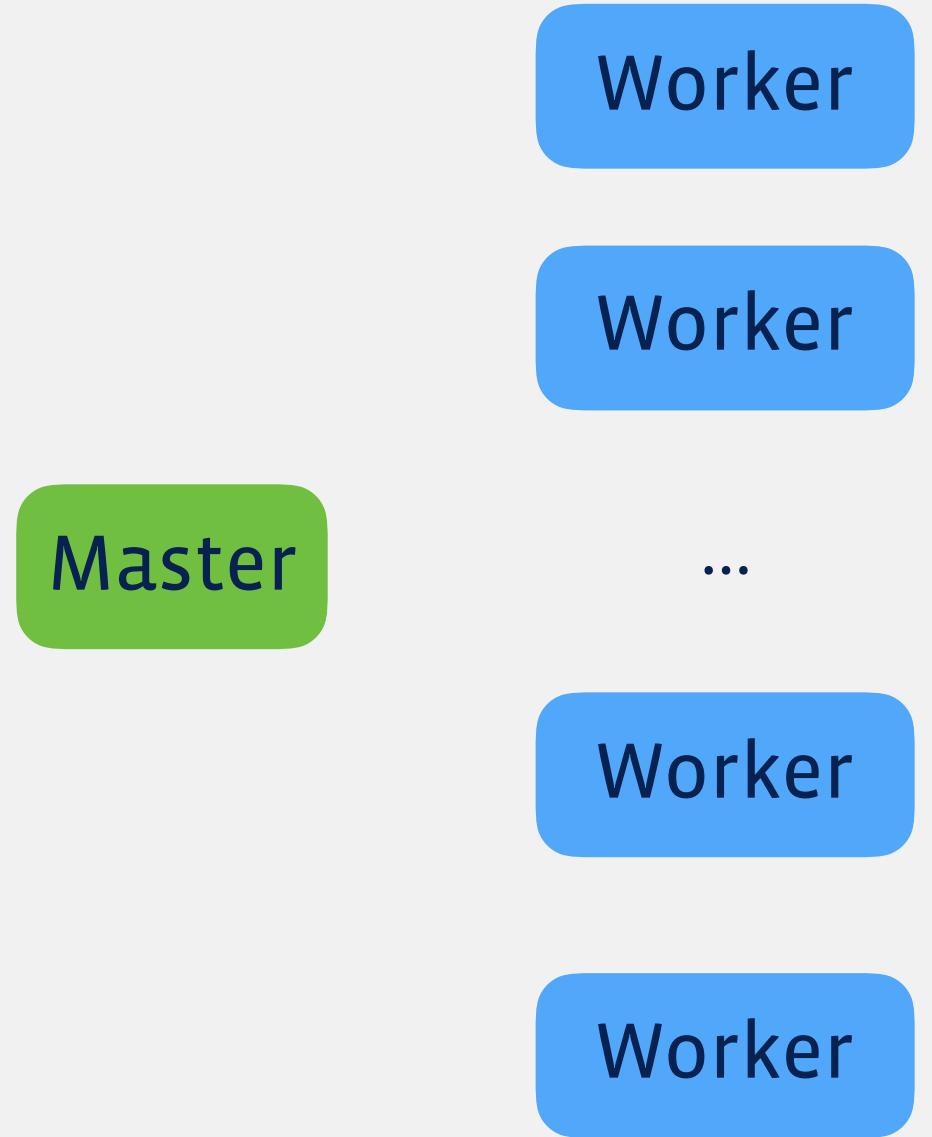




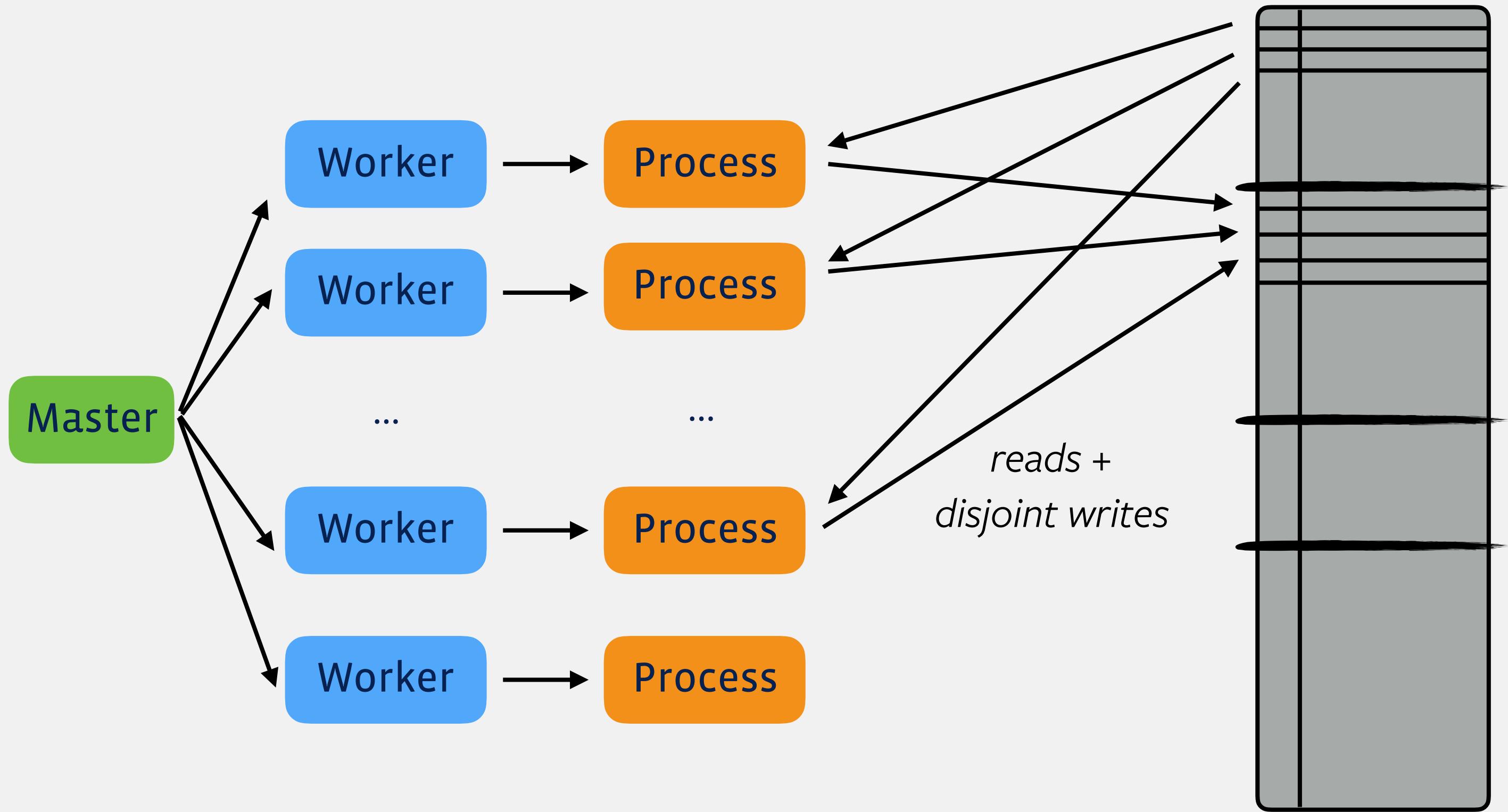


Shared Memory

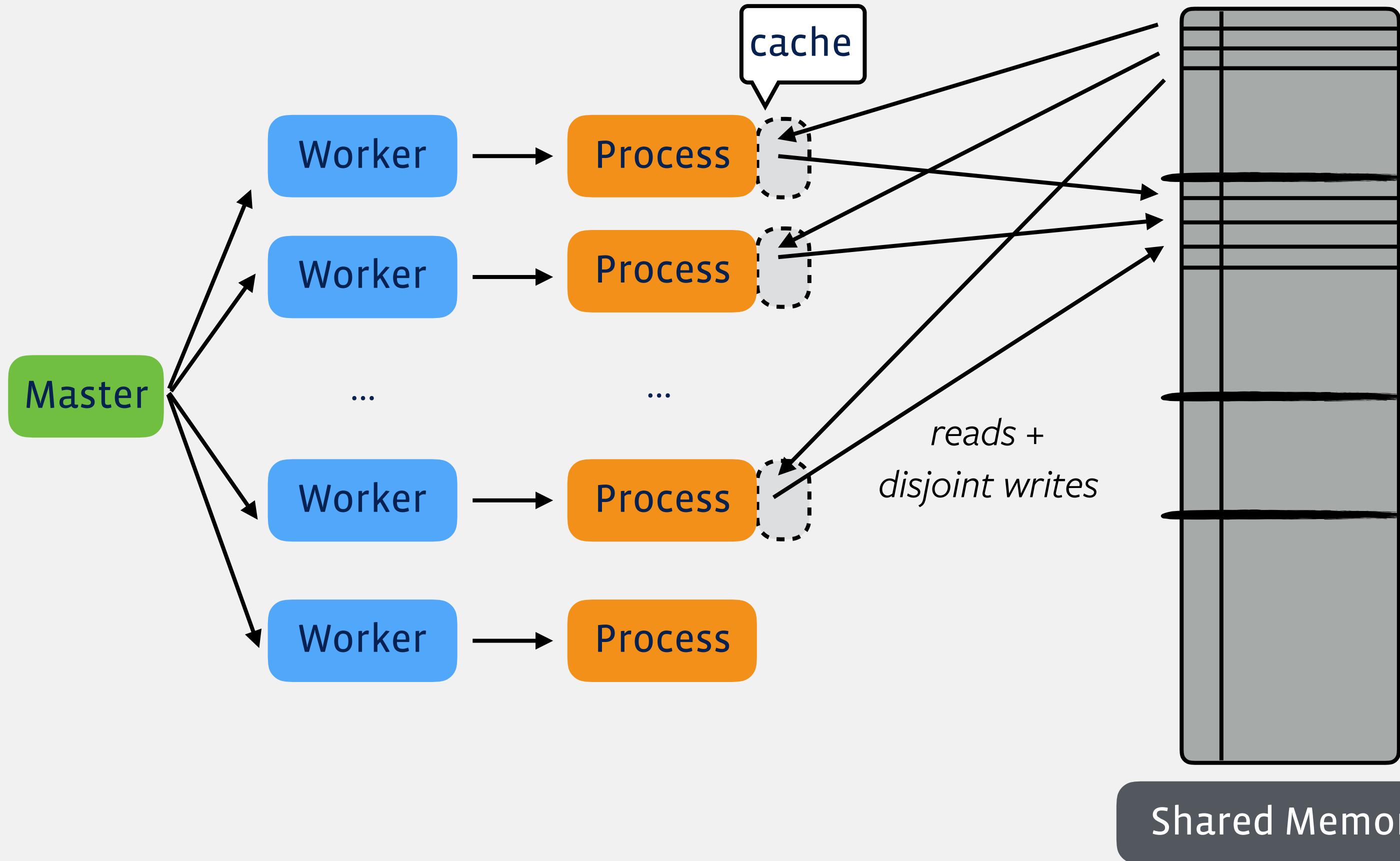


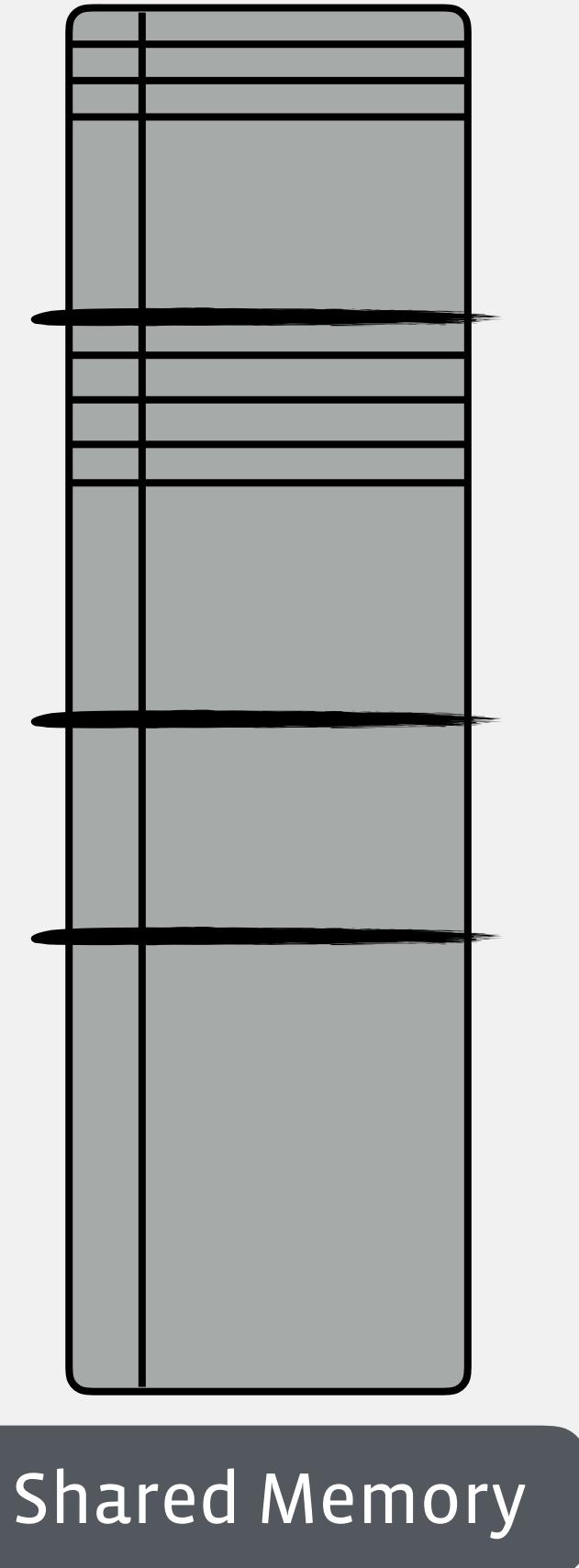
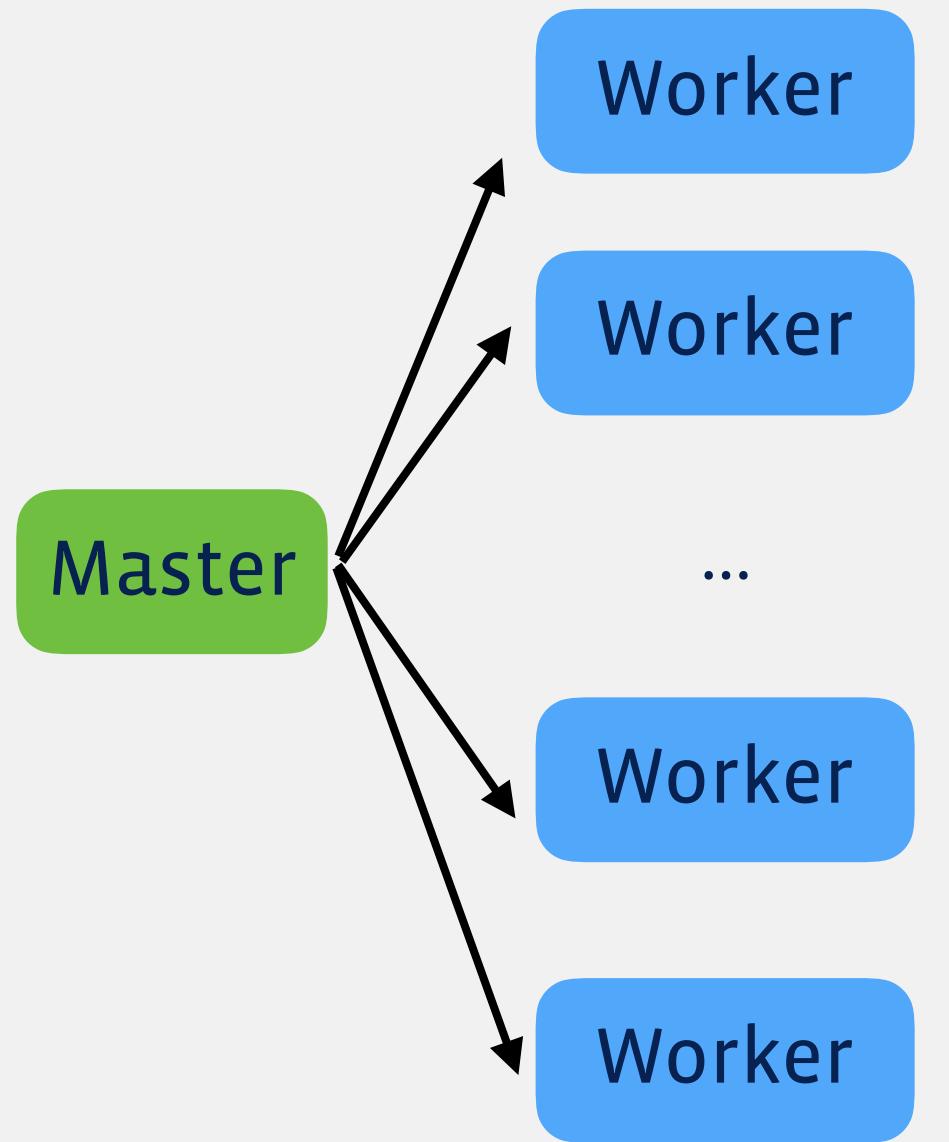


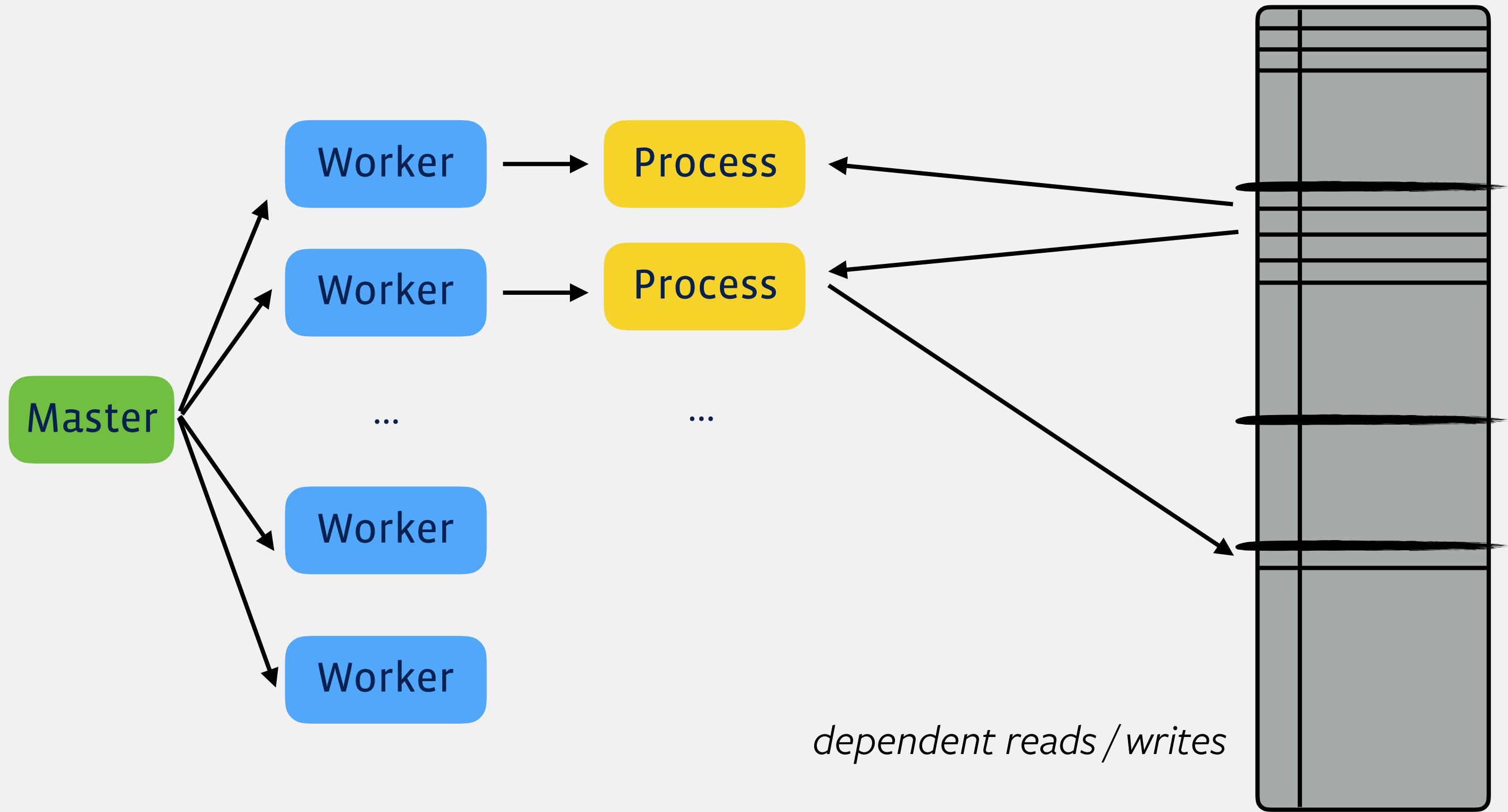
Shared Memory



Shared Memory

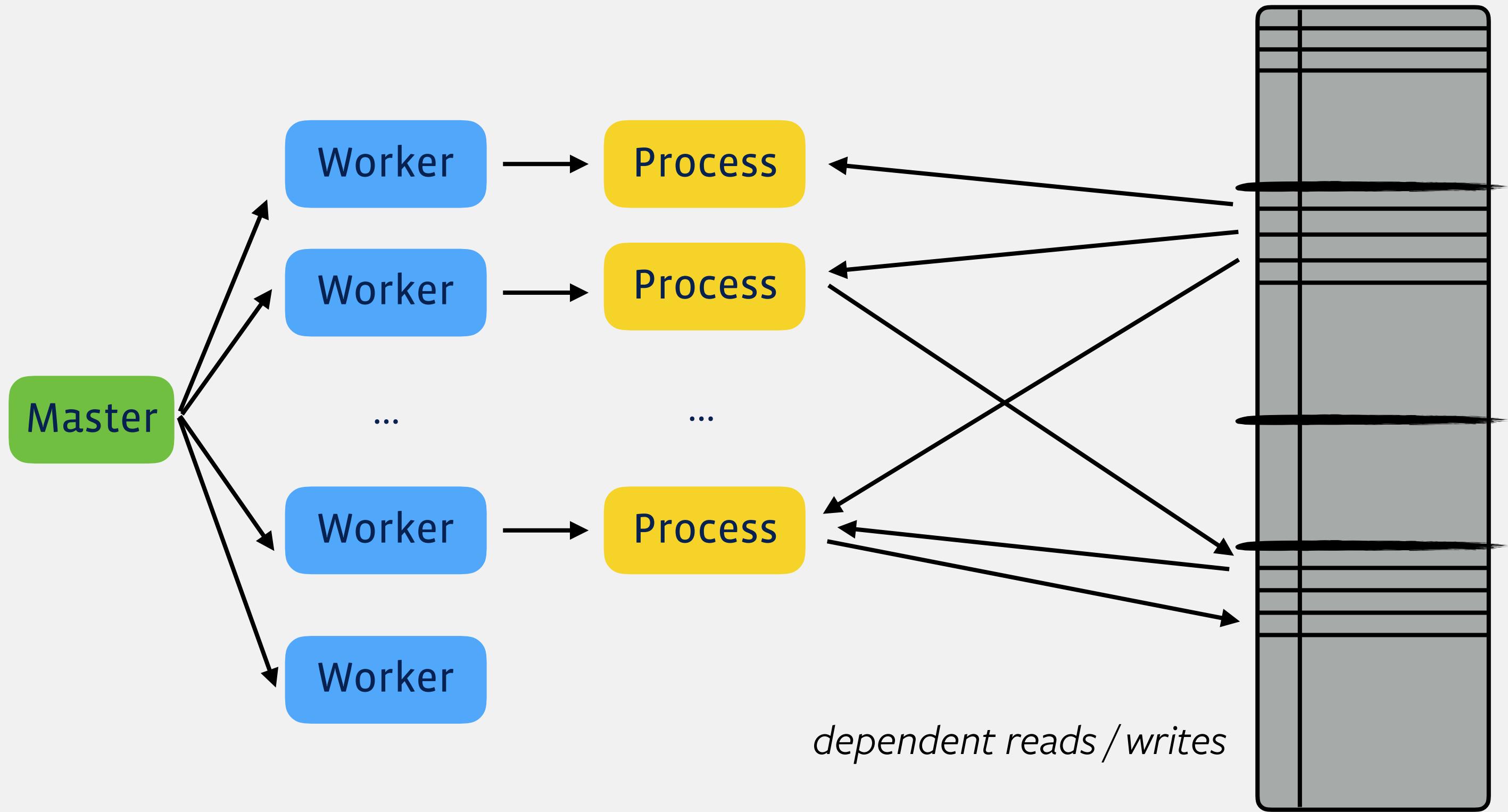




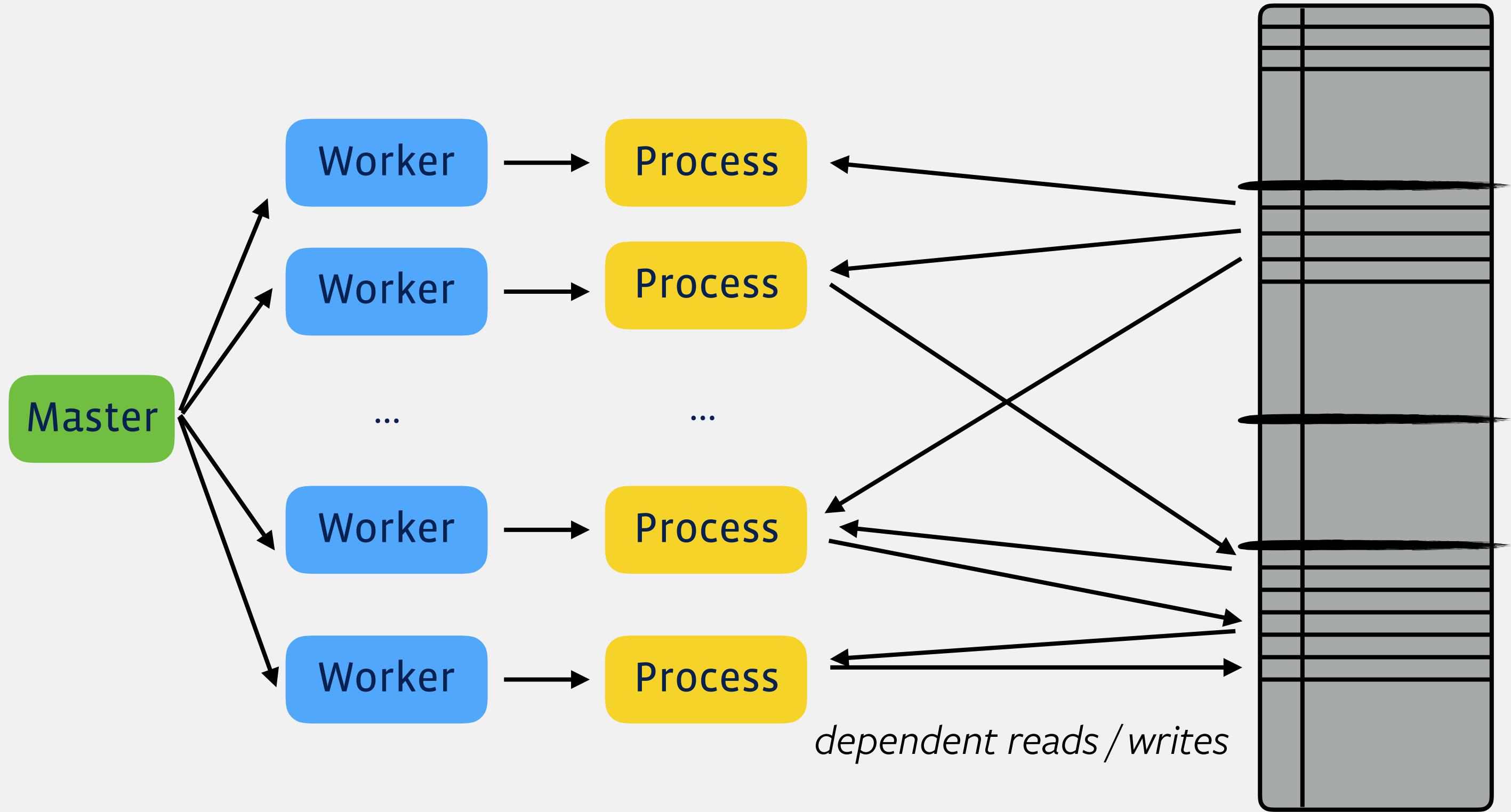


dependent reads / writes

Shared Memory



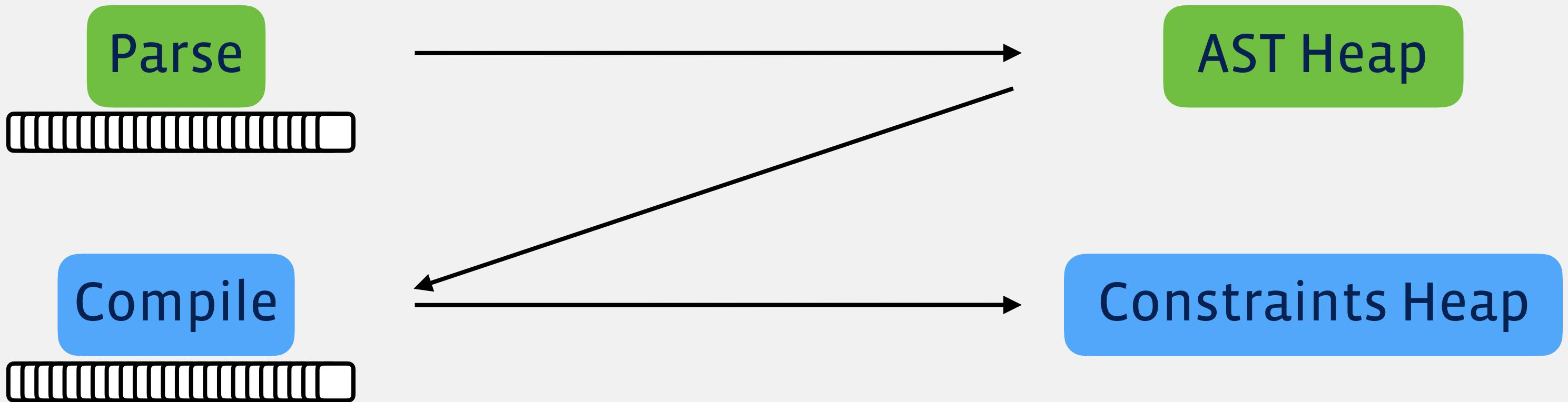
Shared Memory

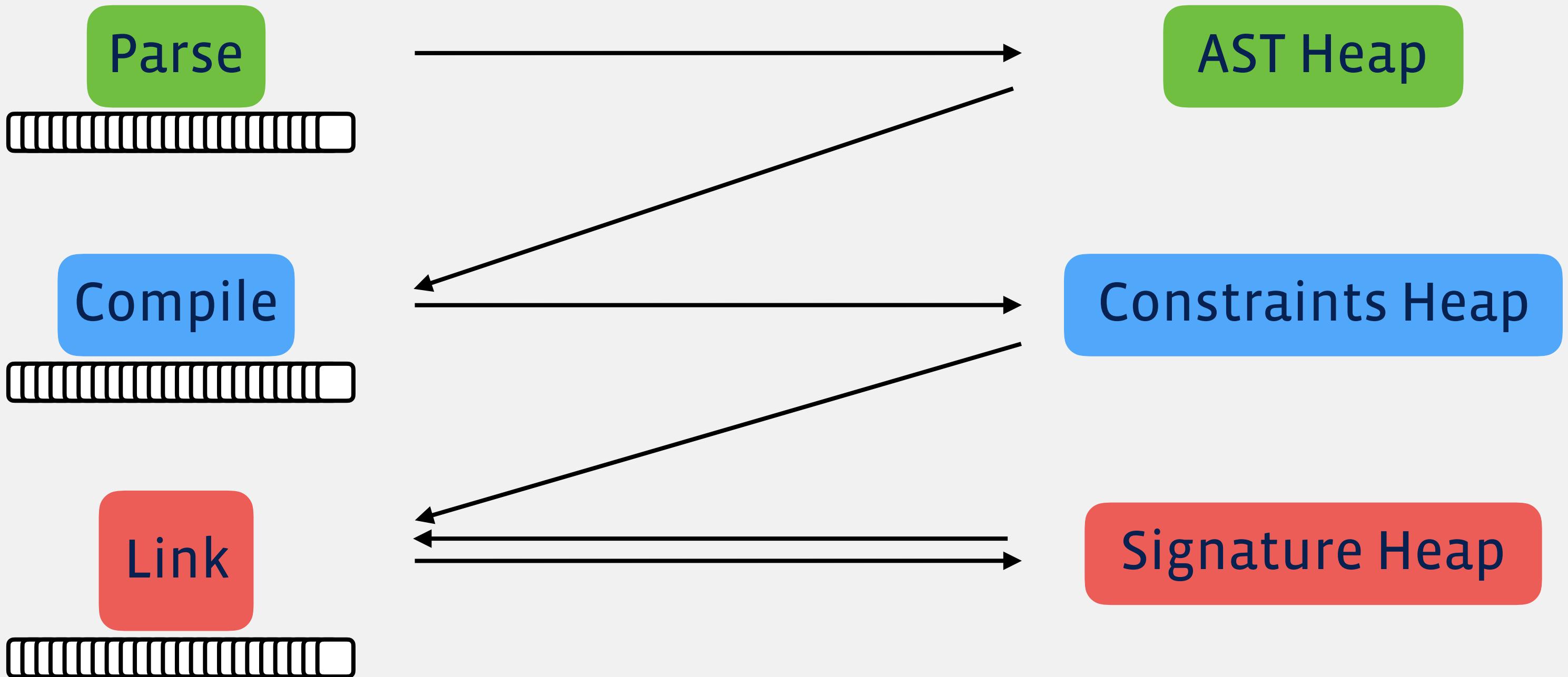


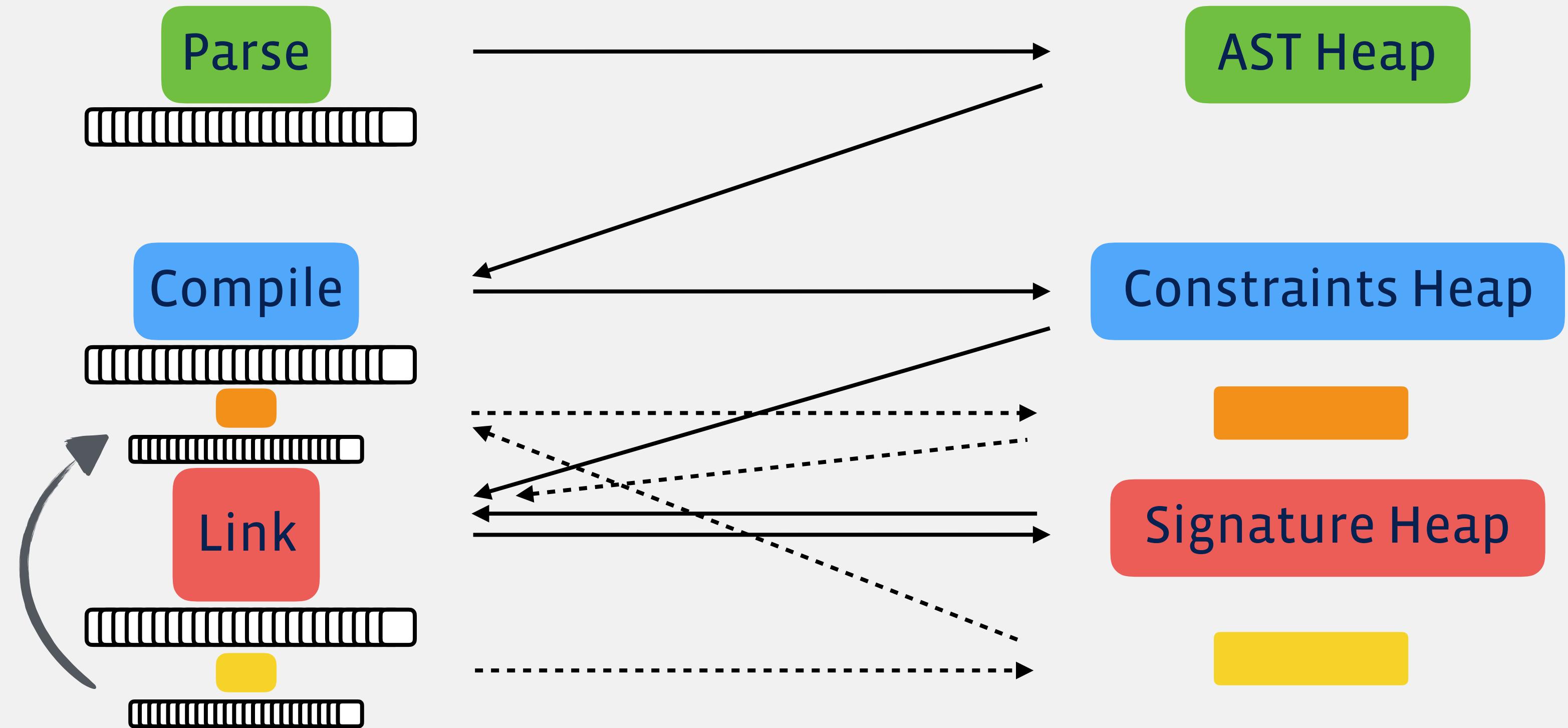
Shared Memory

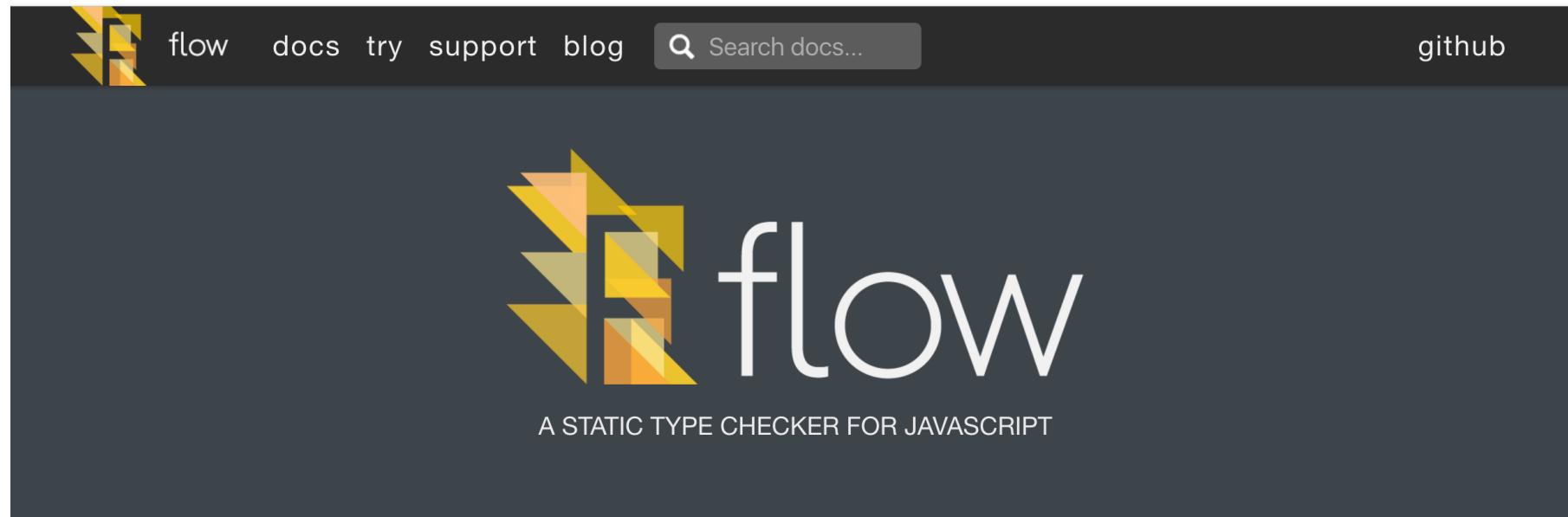
Putting everything together...











The screenshot shows the official Flow.js website. At the top, there's a dark header bar with the Flow logo, navigation links for 'flow', 'docs', 'try', 'support', and 'blog', a search bar containing 'Search docs...', and a 'github' link. Below the header is a large dark section featuring the Flow logo and the word 'flow' in white, followed by the subtitle 'A STATIC TYPE CHECKER FOR JAVASCRIPT'.

Type Inference

Flow uses type inference to find bugs even without type annotations. It precisely tracks the types of variables as they flow through your program.

Flow can catch common bugs in JavaScript programs before they run, including:

- silent type conversions,
- null dereferences,
- and the dreaded `undefined` is not a function.

```
1 // @flow
2 function foo(x) {
3   return x * 10;
4 }
5 foo('Hello, world!');
```

Idiomatic JS

Flow is designed for JavaScript programmers. It understands common JavaScript idioms and very dynamic code.

Realtime Feedback

Flow incrementally rechecks your changes as you work, preserving the fast feedback cycle of developing plain JavaScript.



Fast
 Precise

[show Flow output](#)

Are you in  flow?



flow