



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

This repository Search Pull requests Issues Gist

Unwatch 313 ★ Unstar 8,986 Fork 657

facebook / flow

Code Issues 762 Pull requests 28 Projects 0 Wiki Pulse Graphs

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

3,348 commits 13 branches 47 releases 241 contributors BSD-3-Clause

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

avikchaudhuri committed with Facebook Github Bot fix for incremental bug on non-@flow file move ... Latest commit 6ed92bd 2 days ago

Folder	Commit Message	Time Ago
examples	[flow docs] Main page, "Getting Started", & "Five Simple Examples" re...	8 months ago
flow-typed	./tool test --watch	3 months ago
hack	[hack] Main process exits with same code as child. Also classify Not_...	3 days ago
js	Remove hh_realpath error from /try	2 months ago
lib	Click and Double click are mouse events	2 days ago
newtests	Seal the JSX attributes object if there is no spread	4 days ago
npm-flow-lib	`flow-lib`: JS interfaces for Flow APIs	3 months ago
resources	deploy flow-parser to npm via Travis	10 days ago
scripts	Gracefully handle no git or hg on Windows	3 months ago
src	fix for incremental bug on non-@flow file move	2 days ago
tests	fix for incremental bug on non-@flow file move	2 days ago
tsrc	./tool script to profile flow check	12 days ago
website	fix broken any links	3 days ago
.flowconfig	Add npm-flow-lib to .flowconfig's ignore	2 months ago



Flow
@flowtype

Flow v0.33 is out! Lots of bugfixes & perf improvements. Stay tuned for news on some new tools+features soon as well



facebook/flow

flow - Adds static typing to JavaScript to improve developer productivity and code quality.

github.com

RETWEETS LIKES

60 111



7:35 PM - 29 Sep 2016

Reply to 60 111 ...



Naman Goel @naman34 · Sep 29

@flowtype so honestly do you literally have hundreds of people working on it or what?



JeffMo @lbljeffmo · Sep 30

@naman34 7 people on the team and some badass users/contributors :)

4

... 12



Shane Cavaliere
@shaneccy

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

LIKES
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.

RETWEETS
6

LIKES
12



5:38 PM - 4 Oct 2016

Reply to @irvinebroque @flowtype



philIKON @philikon · 5h

@irvinebroque @Vjeux @flowtype totally my experience as well. If I have to refactor old code, I first flowify it and its tests.

...



Mark Dagleish
@markdagleish

Follow

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

Reply to 12 75 ...



Andrey Popp
@andreyppopp

Follow

Recent @flowtype release adds \$ObjMap type constructor—makes it possible to express typesafe deserialisation w/ validation:

```
import {s} from 'flow-typed';
let schema = s.object({
  name: s.string,
  age: s.number
});

let value = validate(schema, '{name: "Andrey", age: 29}');

value.age
```

age	v number
hasOwnProperty	(prop: any) => boolean
name	v string
propertyIsEnumerable	(prop: any) => boolean
toLocaleString	() => string
toString	() => string
valueOf	() => Object

RETWEETS
44

LIKES
101



5:16 AM - 1 Oct 2016

Reply to 44 101 ...



Flow
@flowtype

Follow

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

- ➊ **Important not to introduce any latency** in workflow
 - Preserve illusion of quick edit-refresh cycle
 - Work in background, utilize idle time
- ➋ **Parallelize** as much as possible!
- ➌ **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 syntax in purple and red, and the line `length("")` is selected.

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

The sidebar on the left contains various icons for navigation and settings. The bottom status bar shows the file name `example1.js`, the current time `7:12`, and the zoom level `100%`. It also displays the count of errors (`4`) and warnings (`0`), and indicates that the code is in `Babel ES6 JavaScript` mode with `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 title bar "example3.js — /Users/avik/Documents/Flow - ETH Zurich/examples". The left sidebar contains icons for Home, example1.js, example2.js, example3.js (selected), example4.js, example5.js, example6.js, and example7.js. The main editor area displays the following JavaScript code:

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

The code editor includes a vertical toolbar on the left with icons for file operations, search, and other tools. At the bottom, there are navigation buttons, status indicators (4 errors, 0 warnings), and file information (example3.js, 9:14, 100%). The bottom right corner shows "LF", "UTF-8", "Babel ES6 JavaScript", and "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

The screenshot shows a developer's environment for static type checking of JavaScript code using Flow. The main window displays a file named 'example3.js' located at '/Users/avik/Documents/Flow - ETH Zurich/examples'. The code itself is as follows:

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

Annotations are present for the variable 'x' and the function's return type. The variable 'x' is annotated as X , which is then mapped via GetProp("length") to the return type Call(null) . The function's overall return type is annotated as $(x: X) \Rightarrow \dots$, and the specific call to `length(null)` is annotated as $\text{Call(null)} \Rightarrow \dots$.

The IDE interface includes a navigation bar with tabs for Home, example1.js, example2.js, example3.js (selected), example4.js, example5.js, example6.js, and example7.js. On the left, there is a sidebar with various icons for file operations like copy, paste, and search. The bottom of the screen features a status bar with information such as file name, line count (4), character count (0), encoding (LF), and update count (5).

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

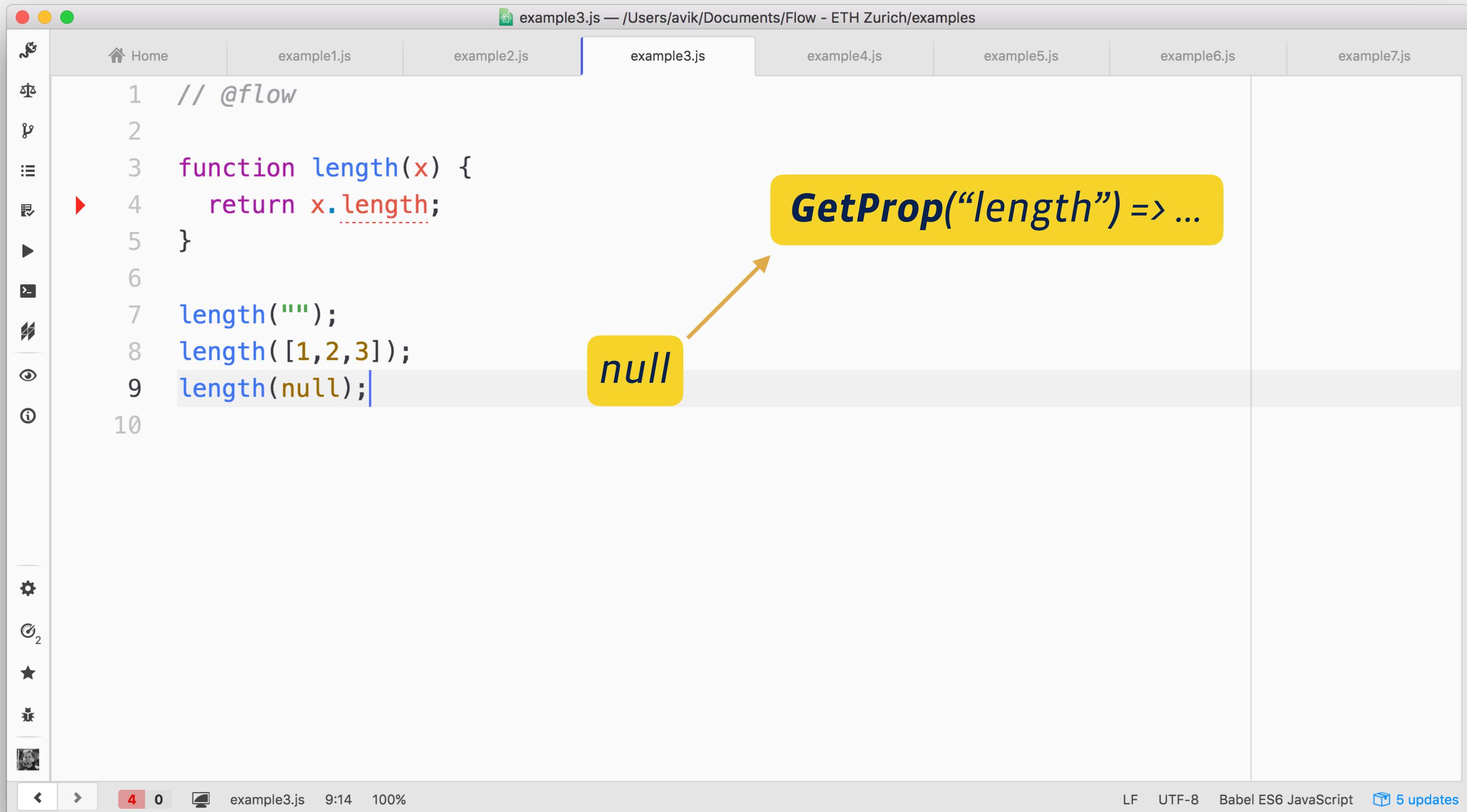
Home example1.js example2.js example3.js example4.js example5.js example6.js example7.js

```
// @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 example3.js 9:14 100% LF UTF-8 Babel ES6 JavaScript 5 updates

The screenshot shows a code editor window with a file named 'example3.js'. The code contains a function 'length' that returns the '.length' property of its argument. When the function is called with 'null', it results in an error because 'null' does not have a 'length' property. A yellow callout box labeled 'GetProp("length") => ...' points to the line 'length(null);'. Another yellow box labeled 'null' points to the word 'null' in the code. A large red 'Error!' is displayed prominently.

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);
```

Diagram illustrating flow analysis results:

- Variable **X** branches into:
 - Refine("== null") => X1**
 - Refine("!= null") => X2**
- Value **null** flows into variable **X**.

Tool interface:

- File menu icons: Home, Example 1-7.
- Search/Replace: ⌘F, ⌘R.
- Find Next: ⌘G.
- Find Previous: ⌘Shift-G.
- Line numbers: 1-13.
- Code editor status: 4 errors, 0 warnings, 5 updates.
- Bottom bar: LF, UTF-8, Babel ES6 JavaScript.

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 a static analysis or type inference process. A variable **X2** is shown pointing to a **GetProp("length")** operation, which then points to a type annotation **{length: string}**. This indicates that the variable **X2** is being analyzed to determine its type, specifically that it has a **length** property of type **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' });

```

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

R2 → R

(result): R2

GetProp("length") => R2

{length: string}

LF UTF-8 Babel ES6 JavaScript 5 updates

The diagram illustrates the type inference process for the `length` function. It starts with a call to `GetProp("length")` on an object of type R_2 , which results in the type R . This type R is then used as the return type for the `length` function. The flow of data from the object's `length` property to the function's return value is indicated by an orange arrow.

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}

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

The screenshot shows a code editor window for a file named 'example5.js'. The code uses Flow type annotations. An annotation 'R' is placed above the line 'const len = length(y);', with an arrow pointing to the word 'number'. Another annotation 'R' is placed above the line 'printItems({ length: 'not a number' });', with an arrow pointing to the word 'string'. The code itself includes a 'length' function that returns -1 for null and the length of the object for non-null objects, and a 'printItems' function that logs each element's index and value.

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...

example6.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     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 }
```

LF UTF-8 Babel ES6 JavaScript 5 updates

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 small "x" icon is located at the end of the parameter line.

property variance object spreads

optional properties functions as predicates extensible objects

exact objects open methods

Wide support for JavaScript idioms

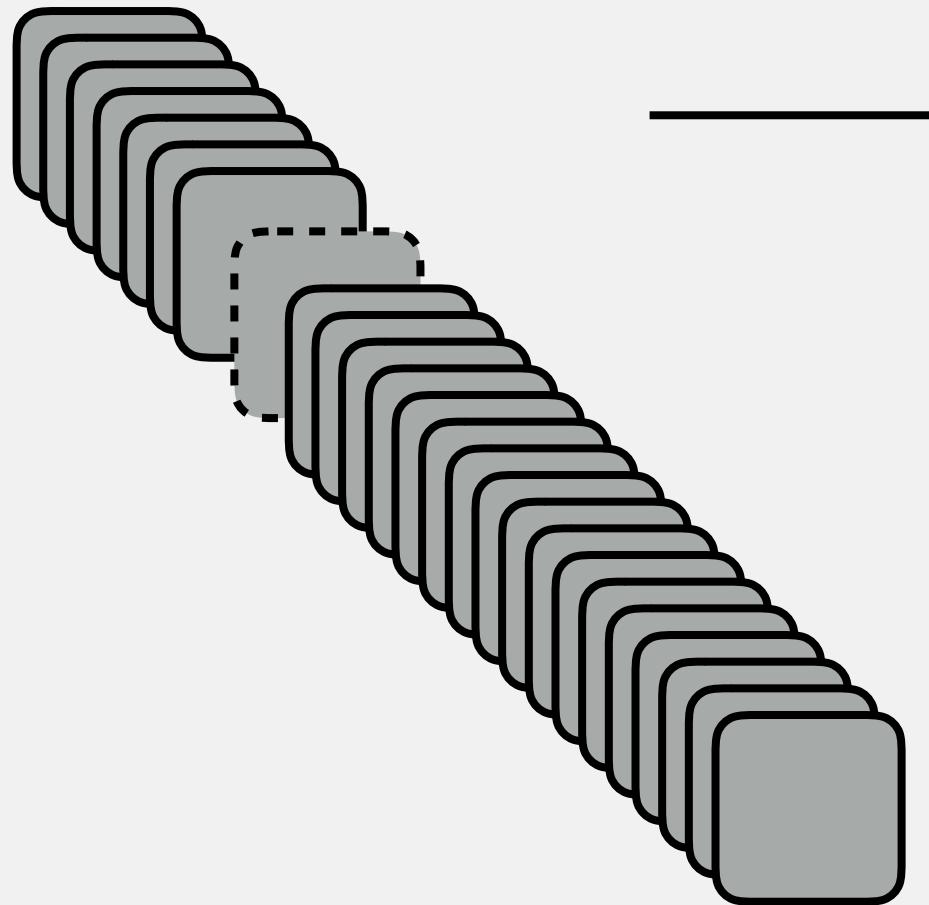
tuples disjoint unions type parameter variance

optional parameters literal types bounded generics

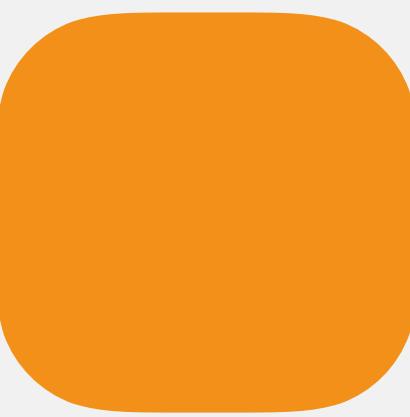
function overloading type transforms

Design for responsiveness

File system



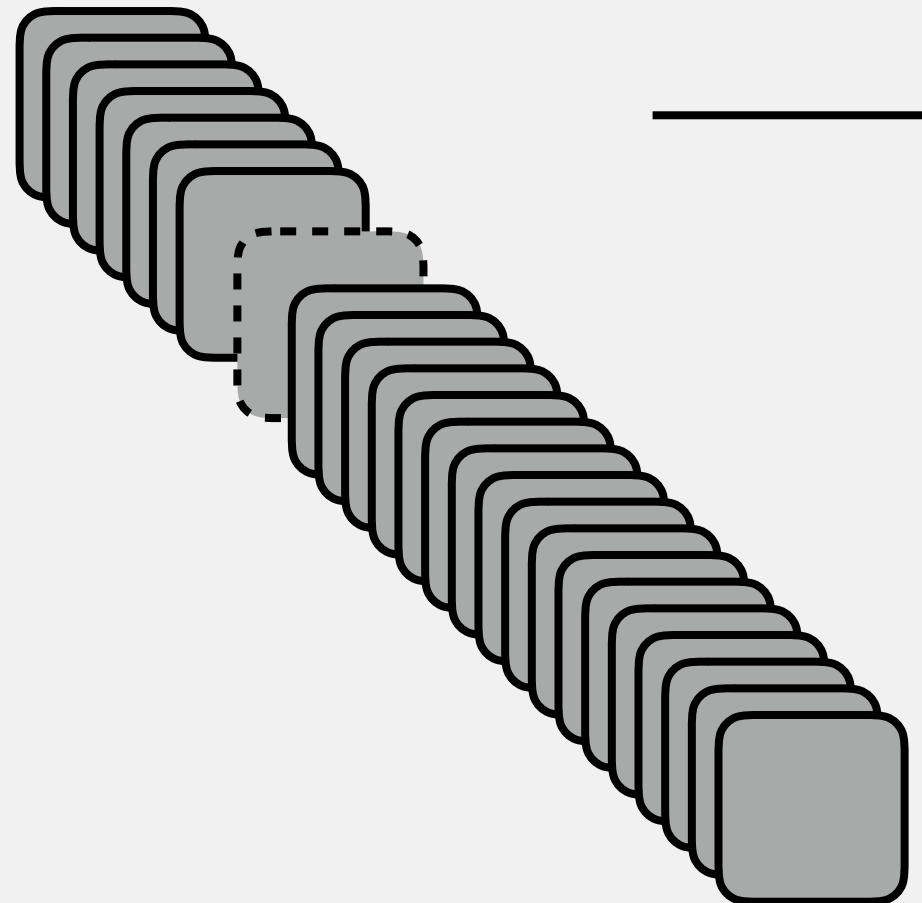
Server



init



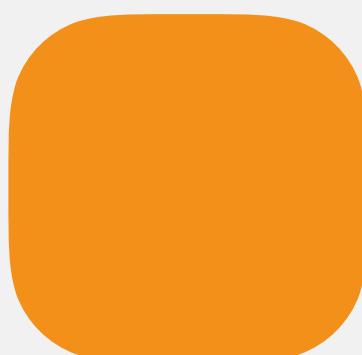
File system



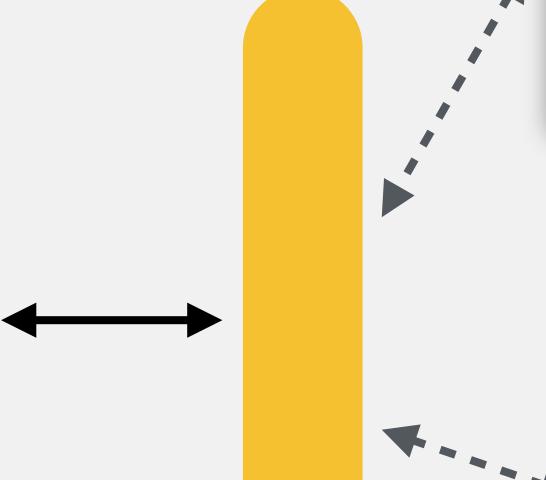
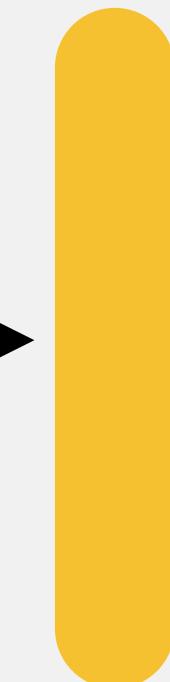
Server



init



Client



cmd

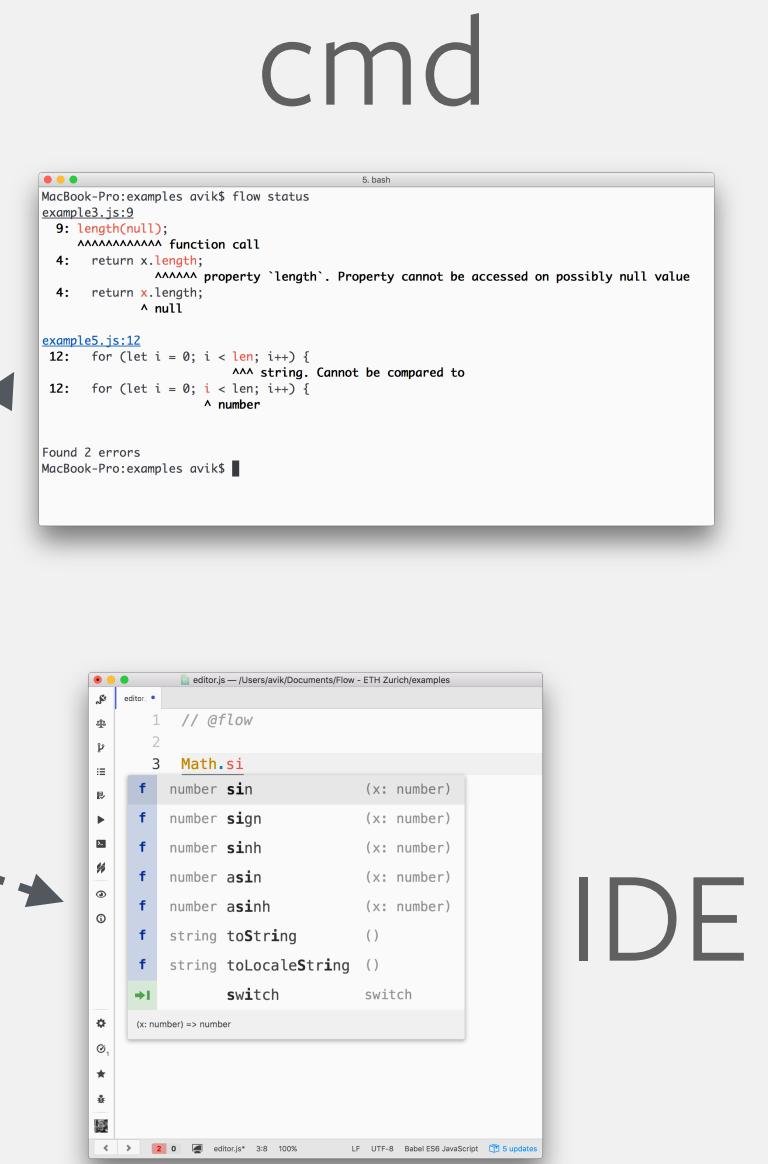
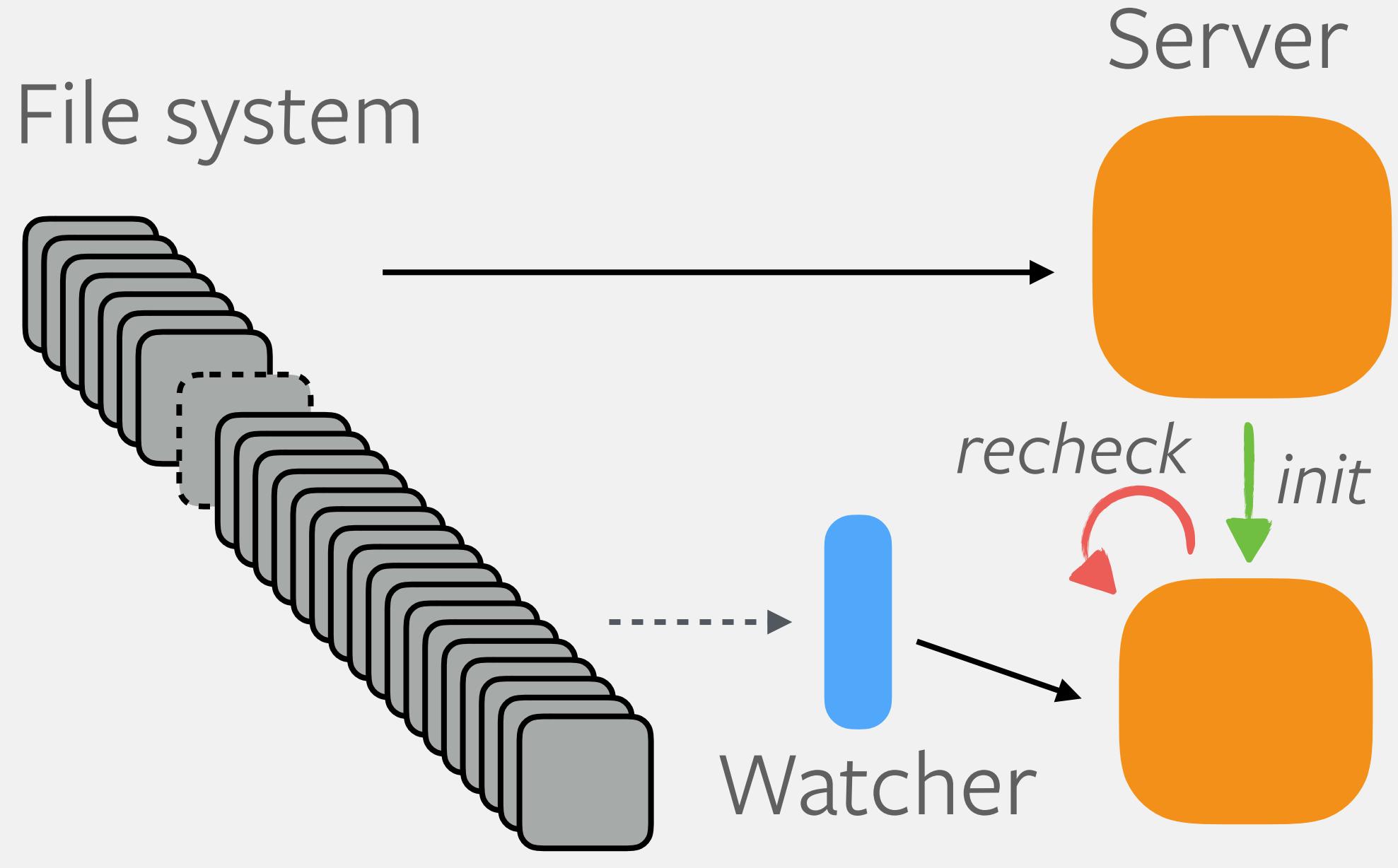
```
MacBook-Pro:examples avik$ flow status
example3.js:9
9: length:null;
    ^^^^^^^^^^^ function call
4:   return x.length;
        ^^^^^^ property `length`. Property cannot be accessed on possibly null value
4:   return x.length;
        ^ null

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$
```

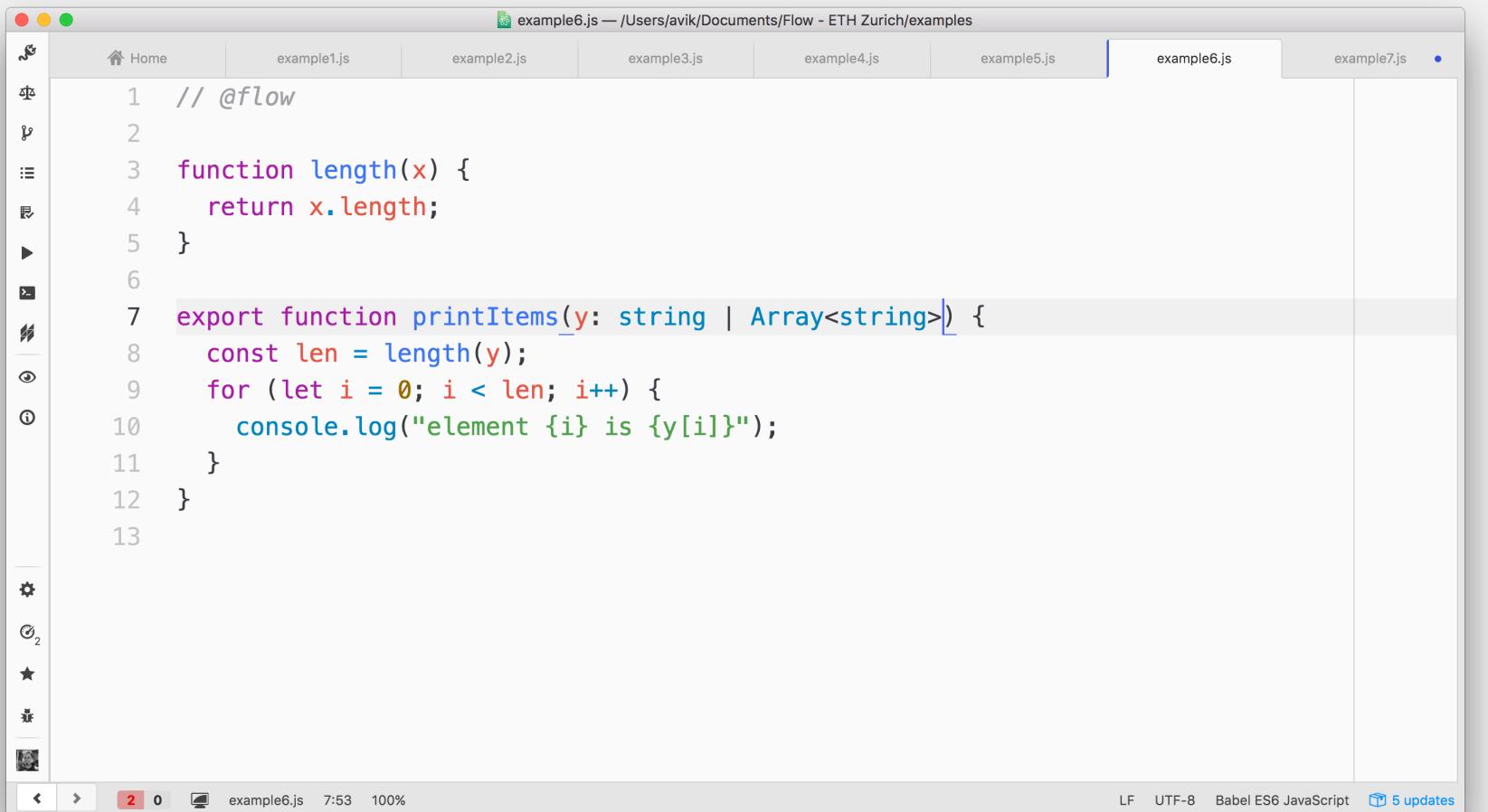
```
// @flow
Math.sin
number sin (x: number)
number sign (x: number)
number sinh (x: number)
number asin (x: number)
number asinh (x: number)
string toString ()
string toLocaleString ()
switch switch
(x: number) => number
```

IDE



Initializing...

example6.js



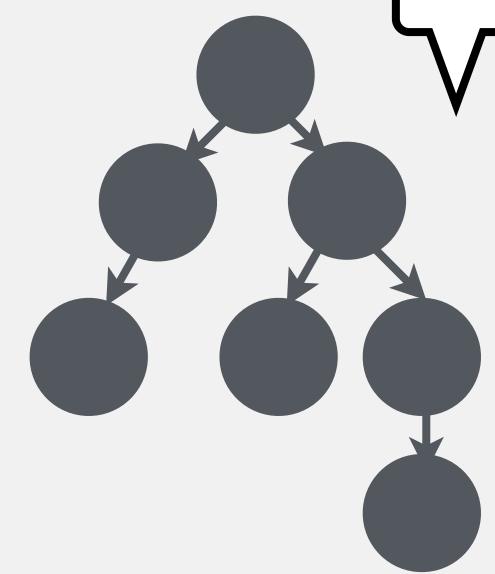
The screenshot shows the Flow IDE interface with the tab 'example6.js' selected. The code editor contains the following JavaScript code:

```
// @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]}");
  }
}
```

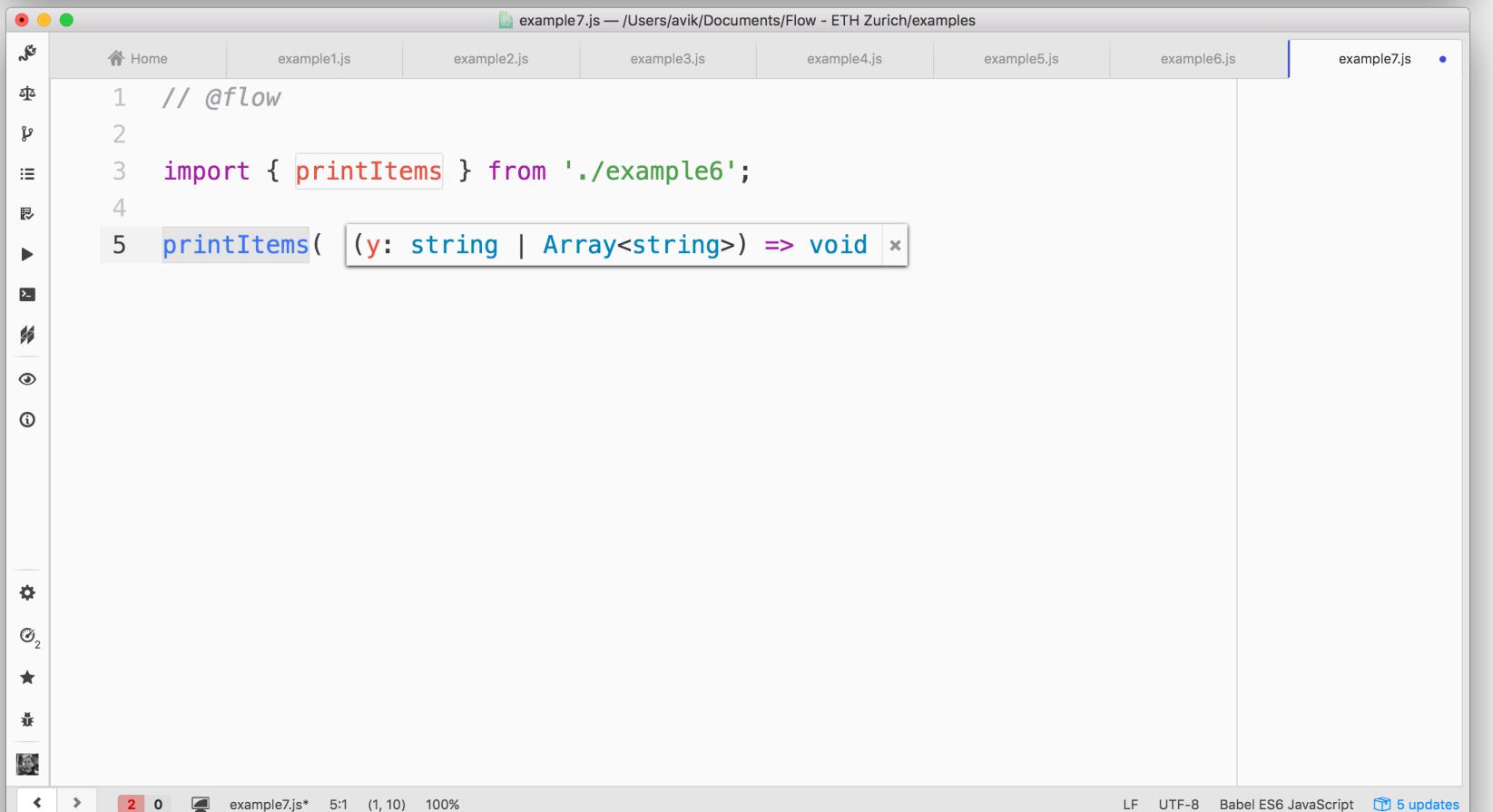
The status bar at the bottom indicates 'example6.js 7:53 100%'.

Parse



AST

example7.js

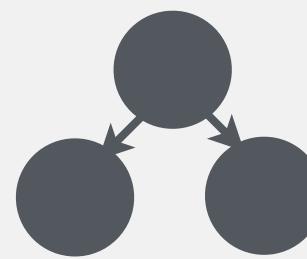


The screenshot shows the Flow IDE interface with the tab 'example7.js' selected. The code editor contains the following JavaScript code:

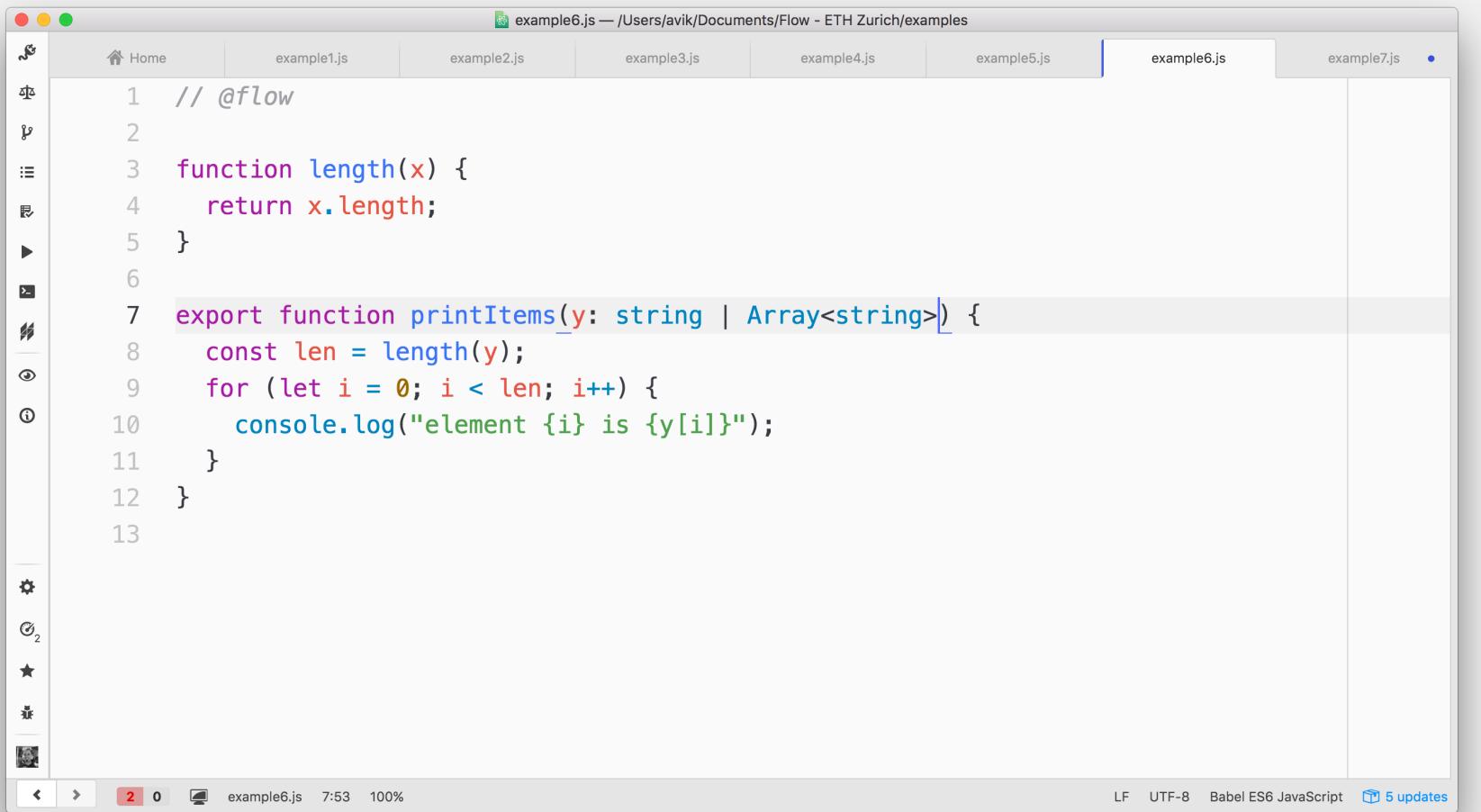
```
// @flow
import { printItems } from './example6';
printItems( (y: string | Array<string>) => void )
```

The status bar at the bottom indicates 'example7.js* 5:1 (1,10) 100%'.

Parse



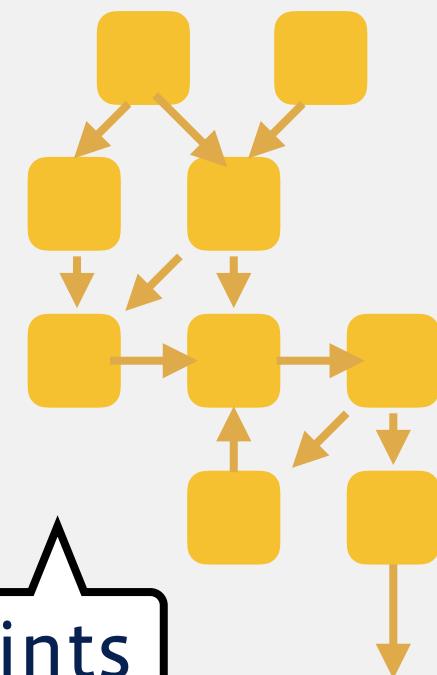
example6.js



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

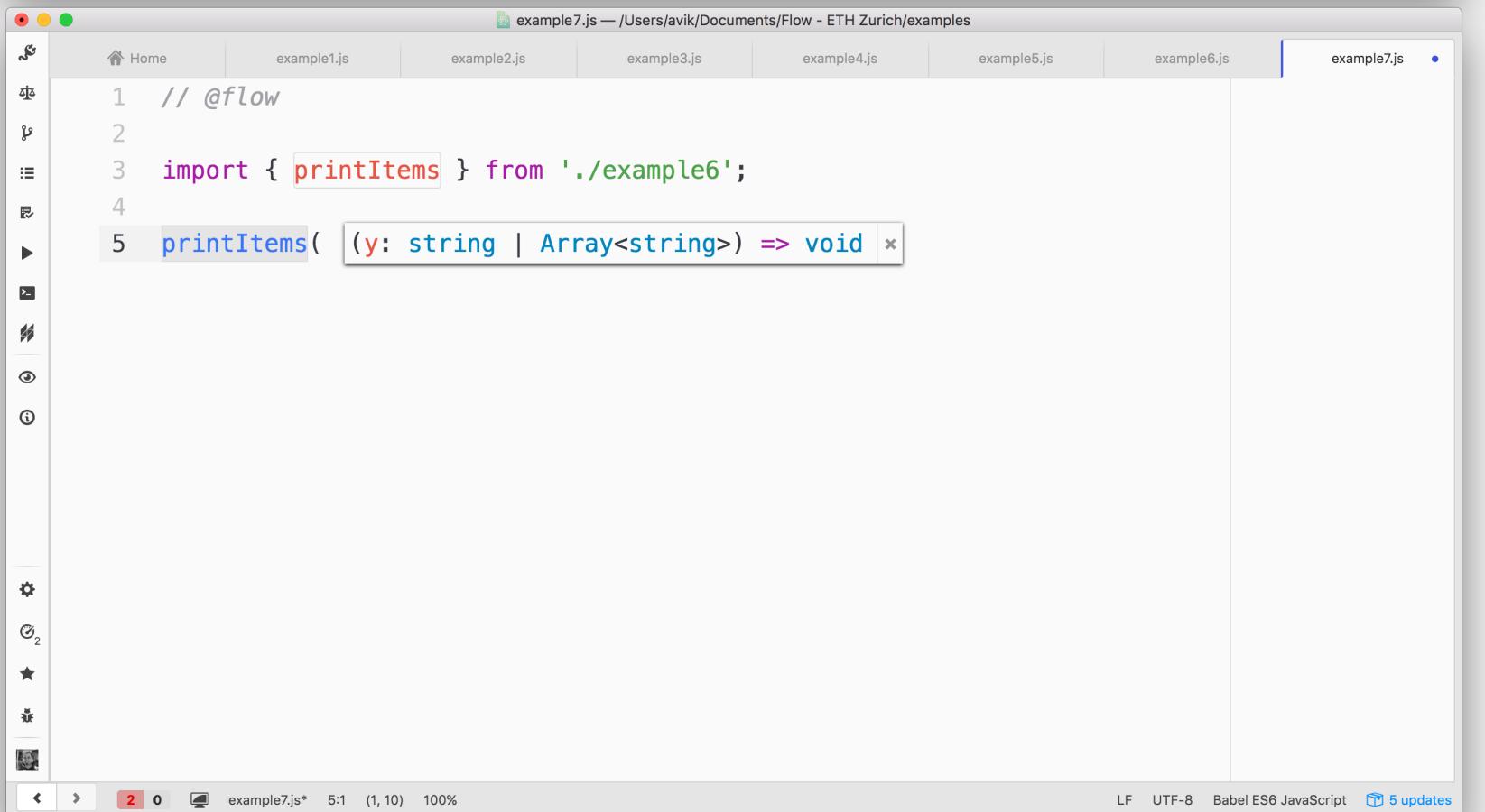
export function printItems(y: string | Array<string>) {
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log("element {i} is {y[i]}");
  }
}
```

Compile



Constraints

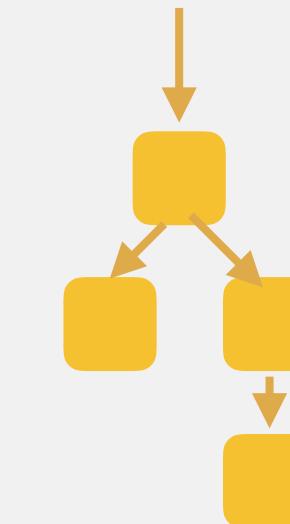
example7.js



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

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

Compile



Constraints

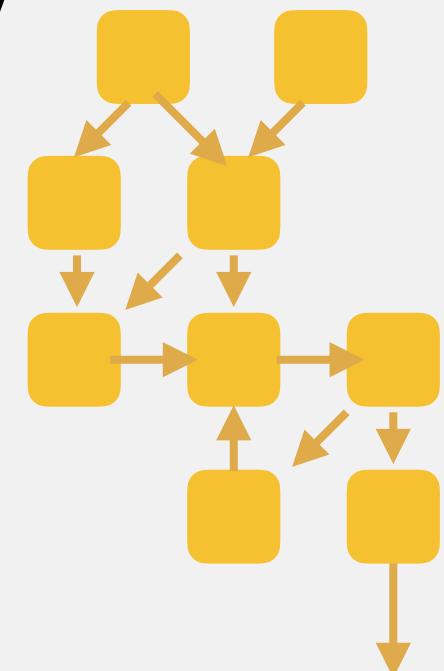
example6.js

```
// @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]}");  
    }  
}
```

Link

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

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 )

```

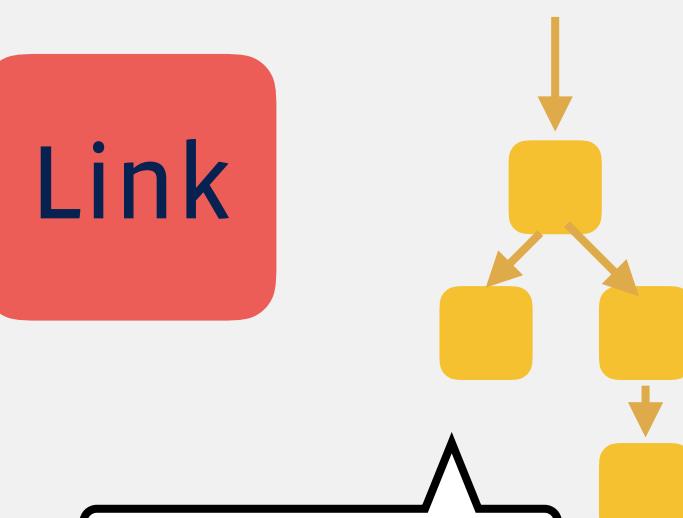
example6.js

Signature

example7.js

Link

Constraints



The image shows two separate code editors, each with a title bar indicating the file name and path: "example6.js — /Users/avik/Documents/Flow - ETH Zurich/examples" and "example7.js — /Users/avik/Documents/Flow - ETH Zurich/examples".

The top editor (example6.js) 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]}");
  }
}
```

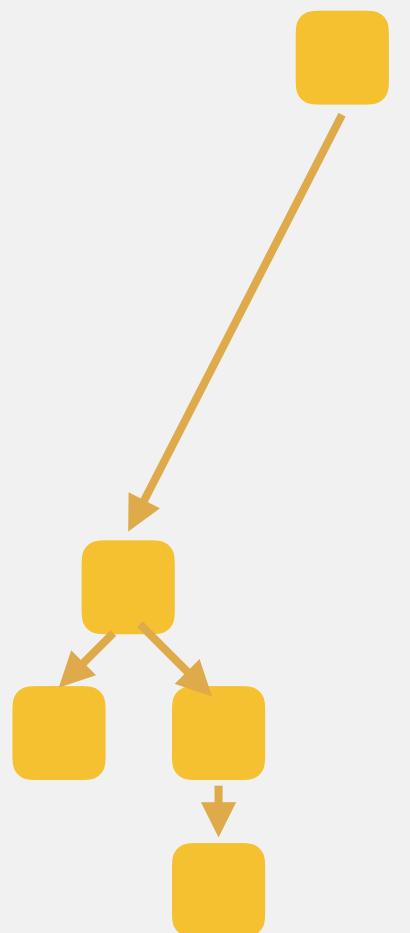
The bottom editor (example7.js) contains the following JavaScript code:

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

example6.js

example7.js

Link



Rechecking...



```
// @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]}");
  }
}

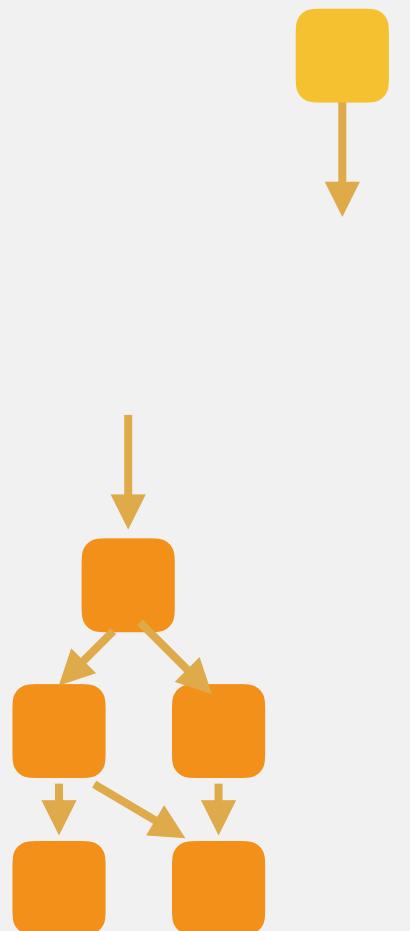
// @flow
import { printItems } from './example6';
printItems( (y: string | Array<string>) => void )
```

example6.js

example7.js

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]}");
  }
}
```

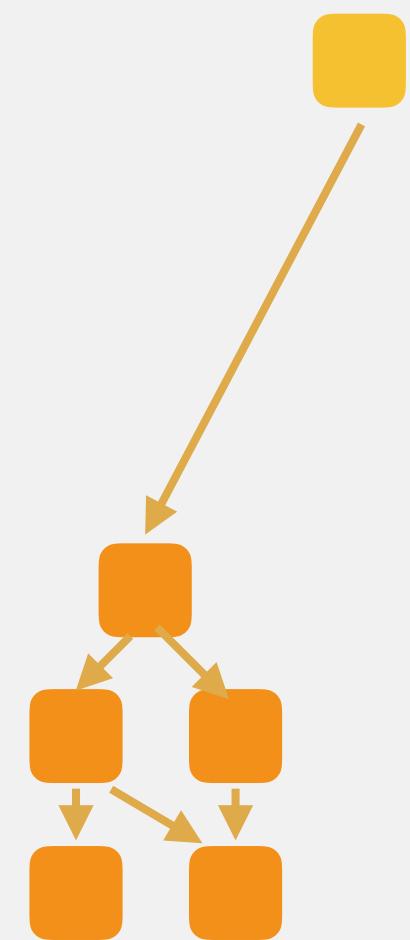


```
// @flow
import { printItems } from './example6';
printItems( (y: string | Array<string>) => void )
```

example6.js

example7.js

Link



EDIT

```
// @flow

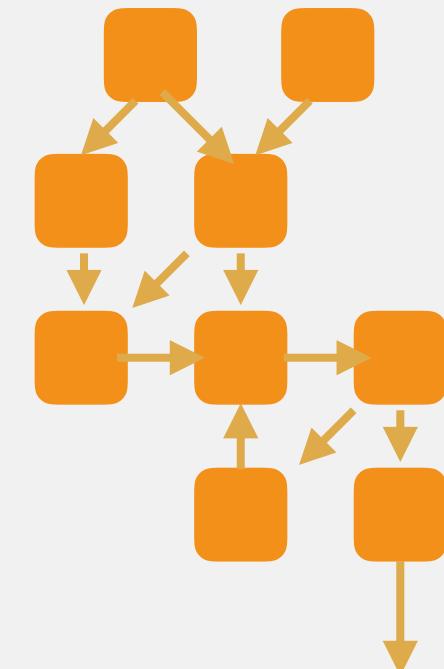
function length(x) {
  return x.length;
}

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

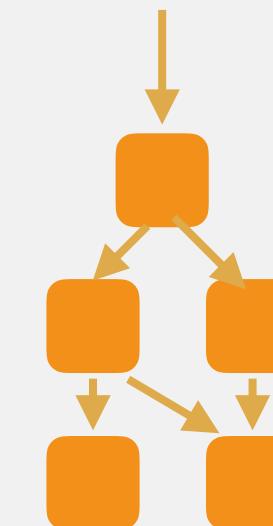


example7.js

```
// @flow

import { printItems } from './example6';

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



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

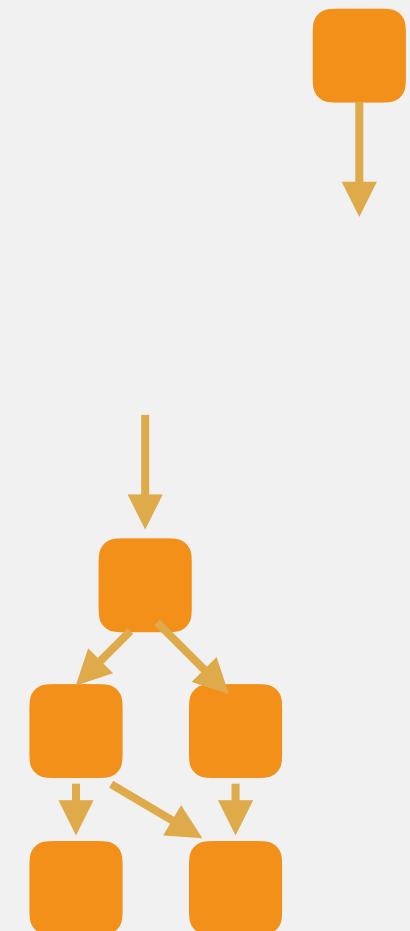
export function printItems(y: string | Array<string>){
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log("element {i} is {y[i]}");
  }
}

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

example6.js



example7.js



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

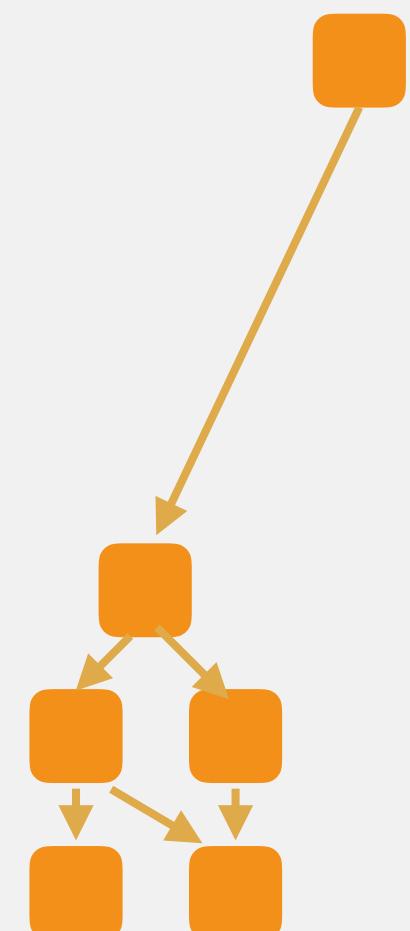
export function printItems(y: string | Array<string>){
  const len = length(y);
  for (let i = 0; i < len; i++) {
    console.log("element {i} is {y[i]}");
  }
}

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

example6.js

example7.js

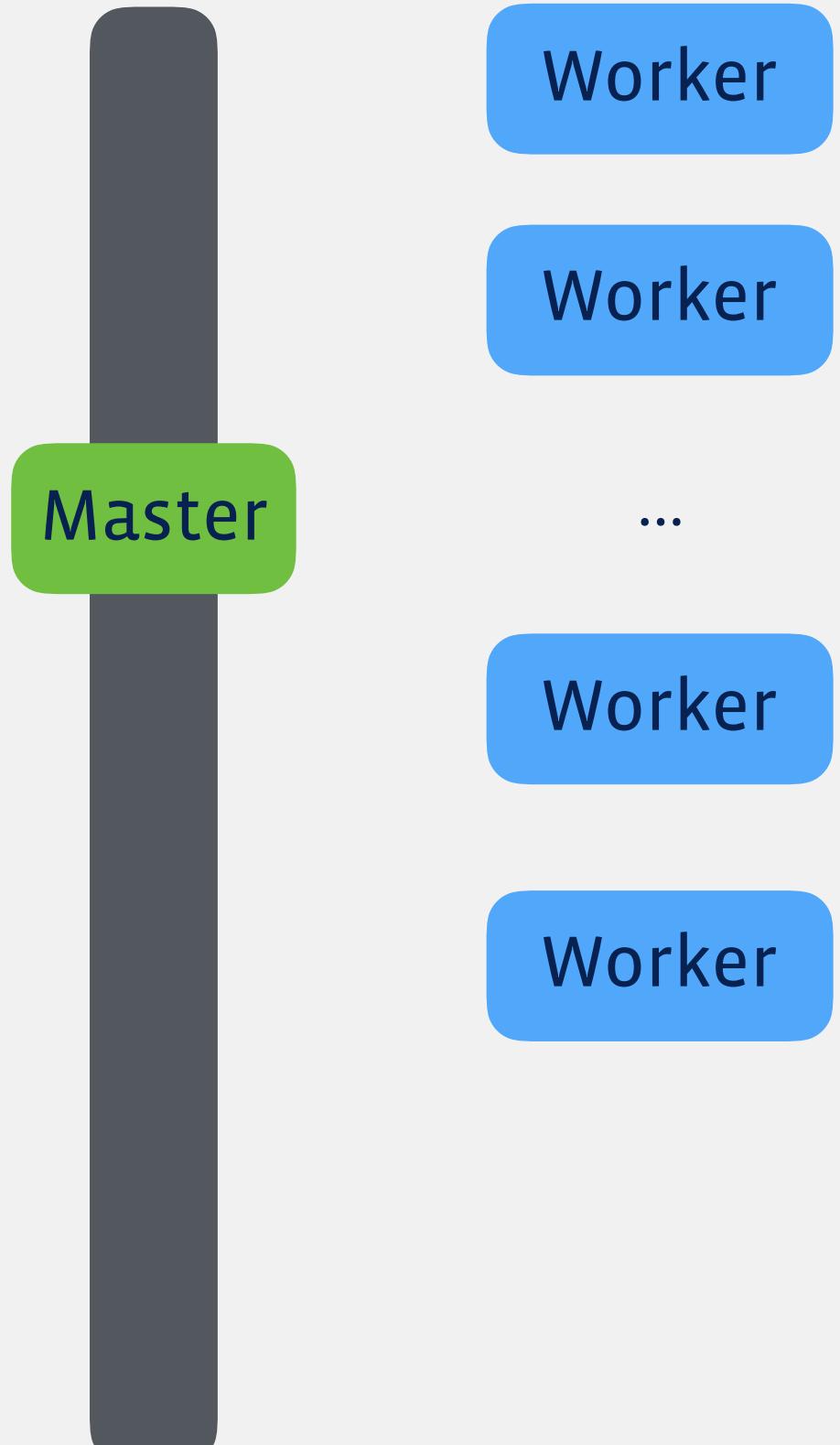
Link



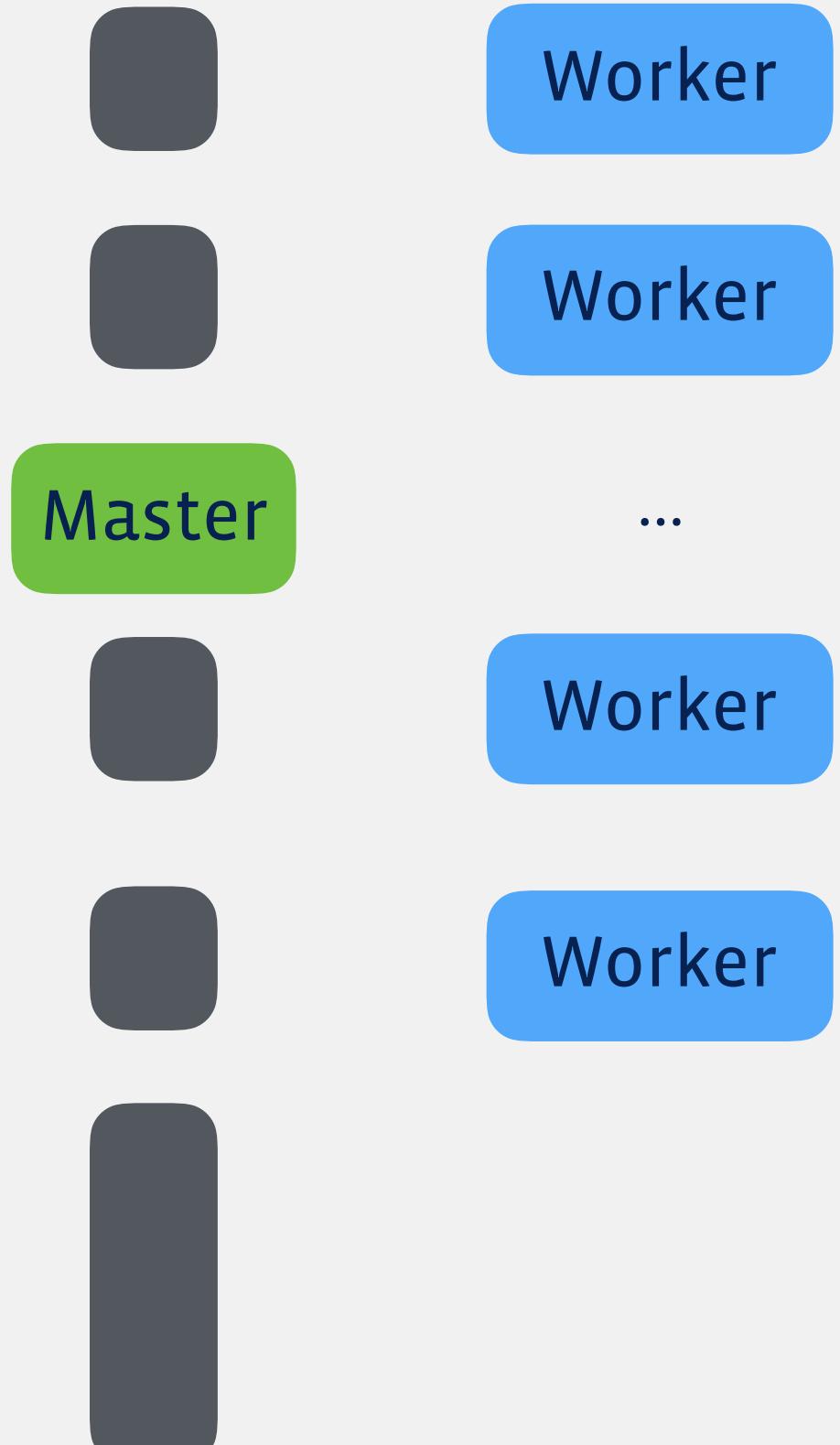
Parallelize everything!

Infrastructure

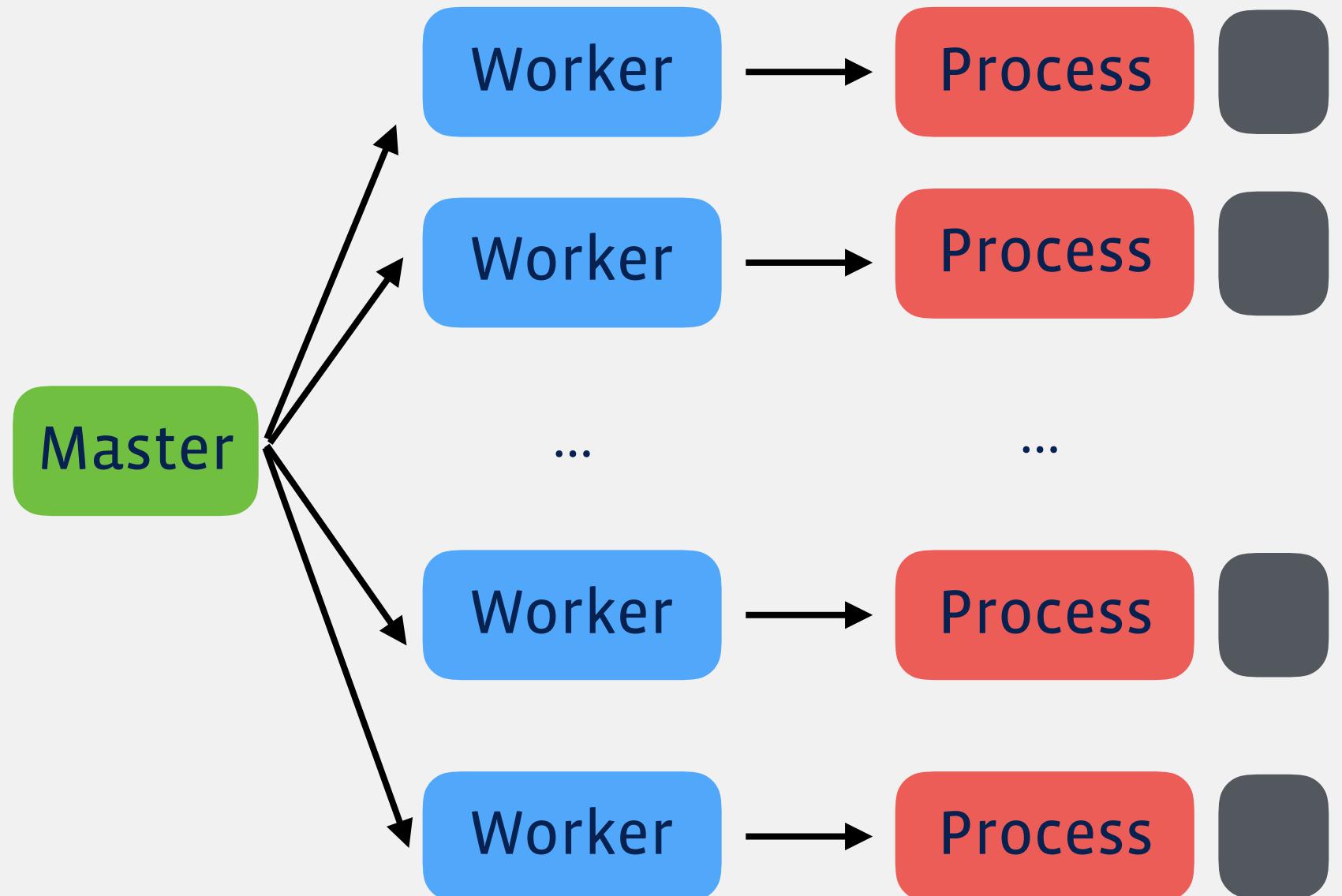
large number of jobs

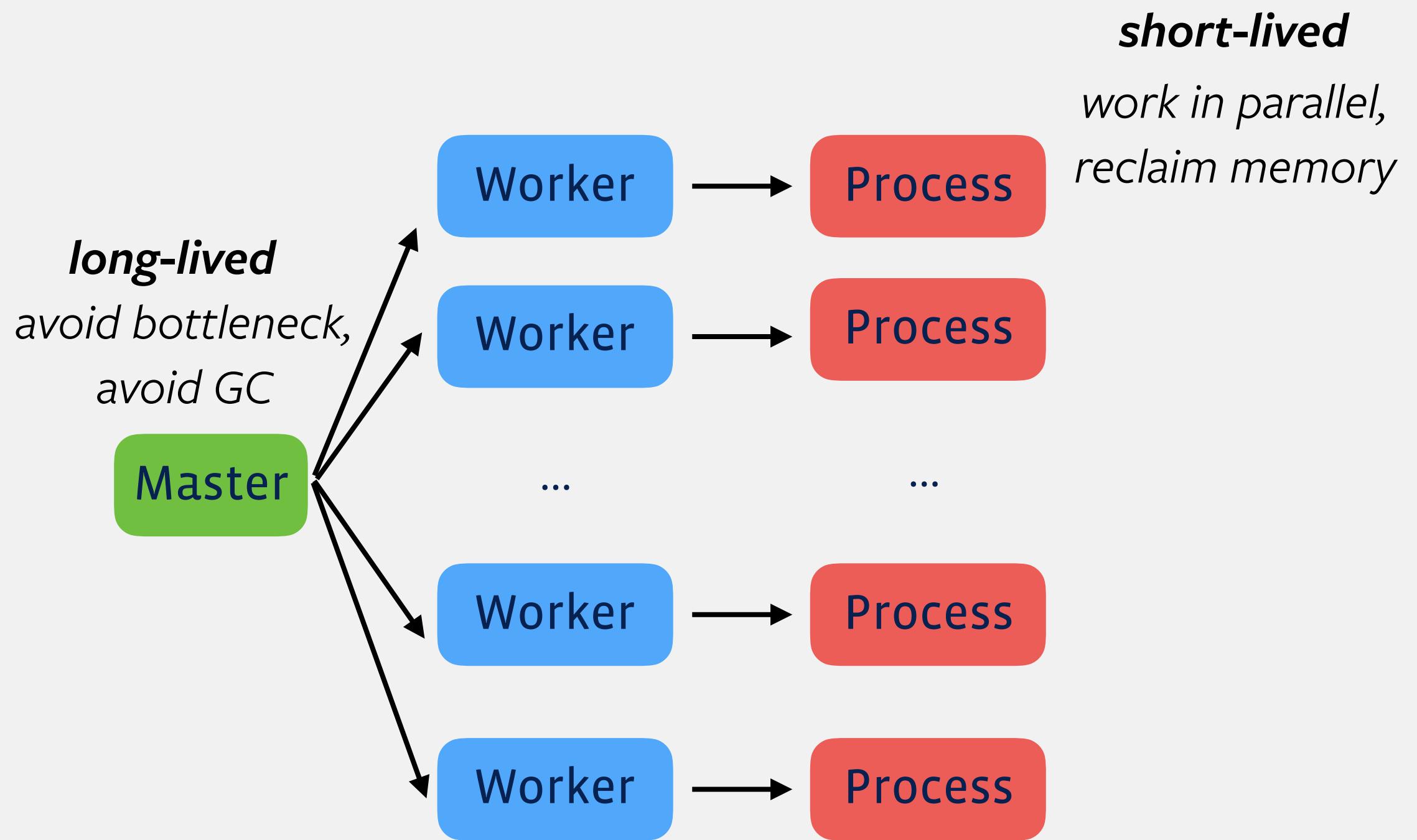


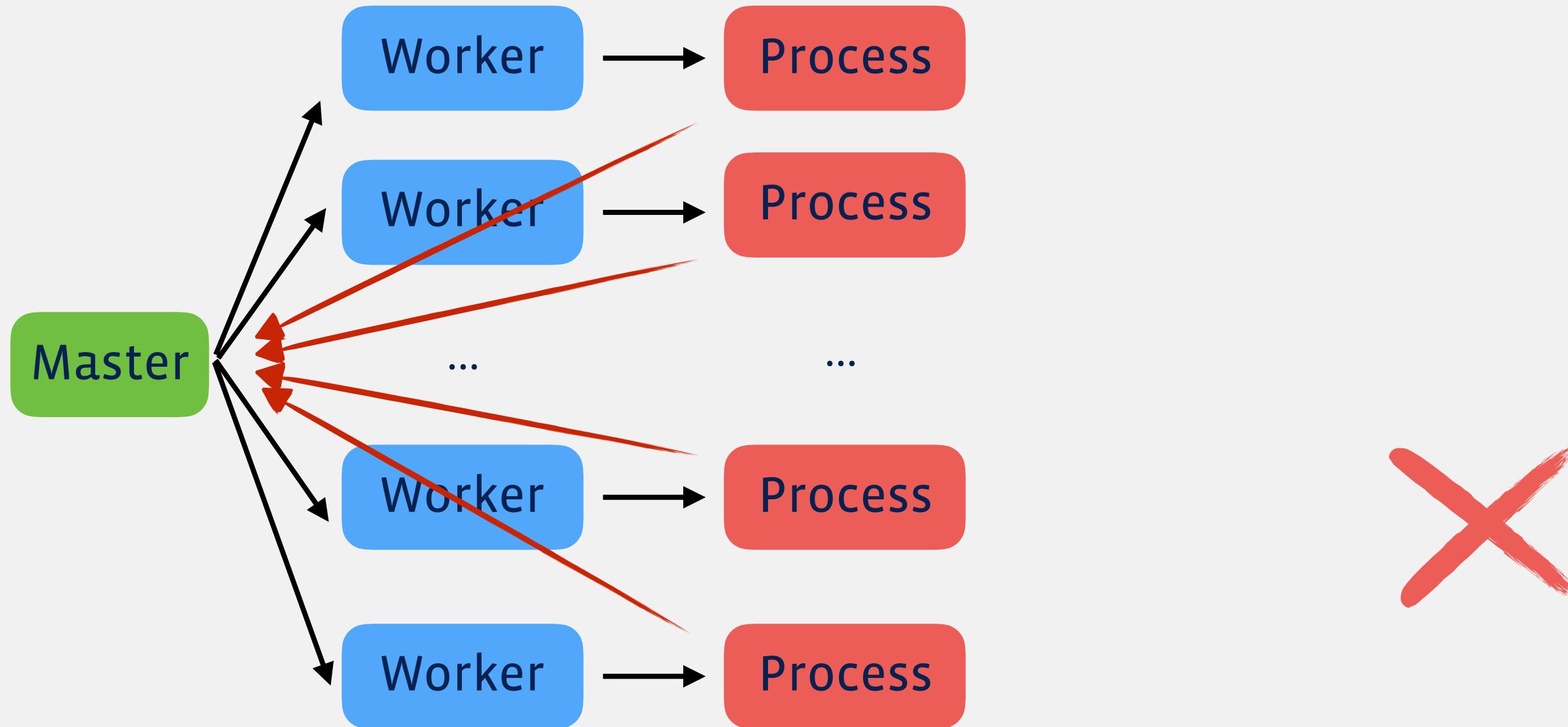
divide into buckets

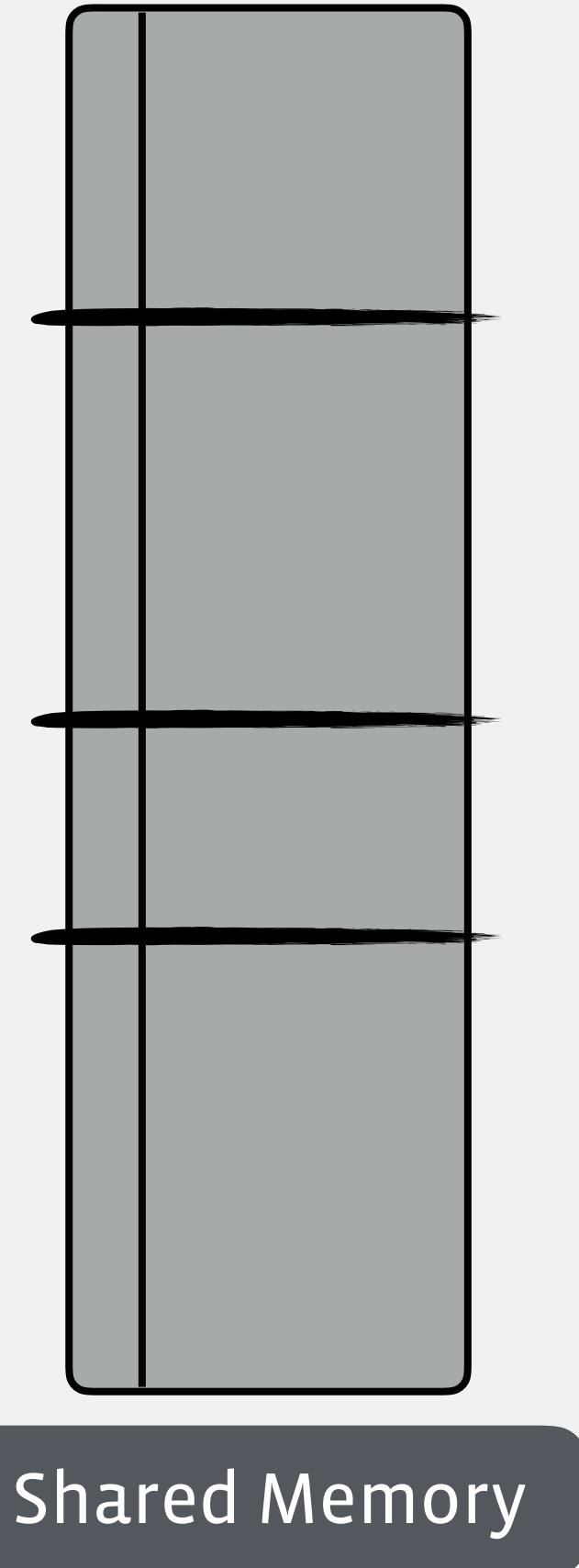
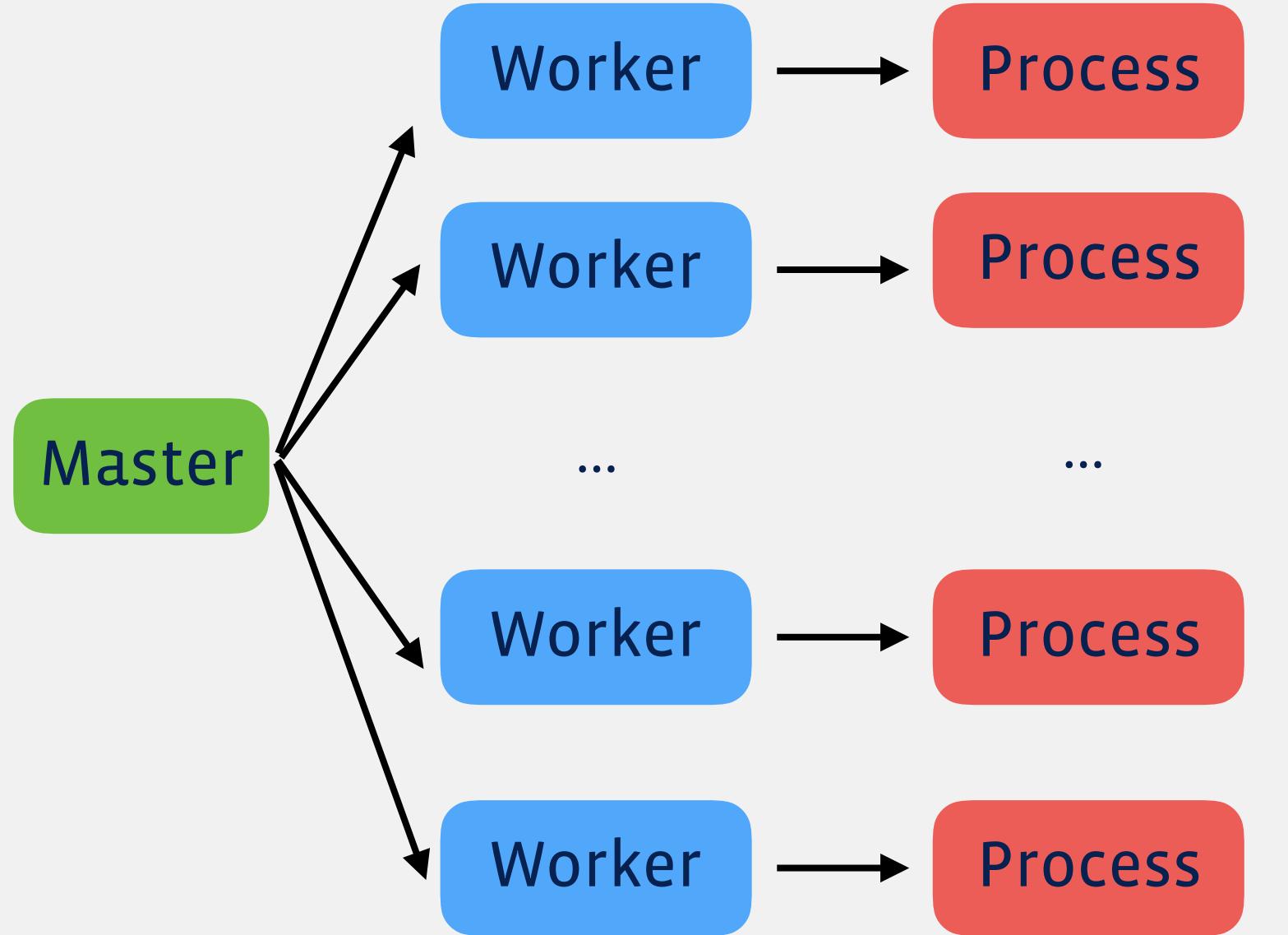


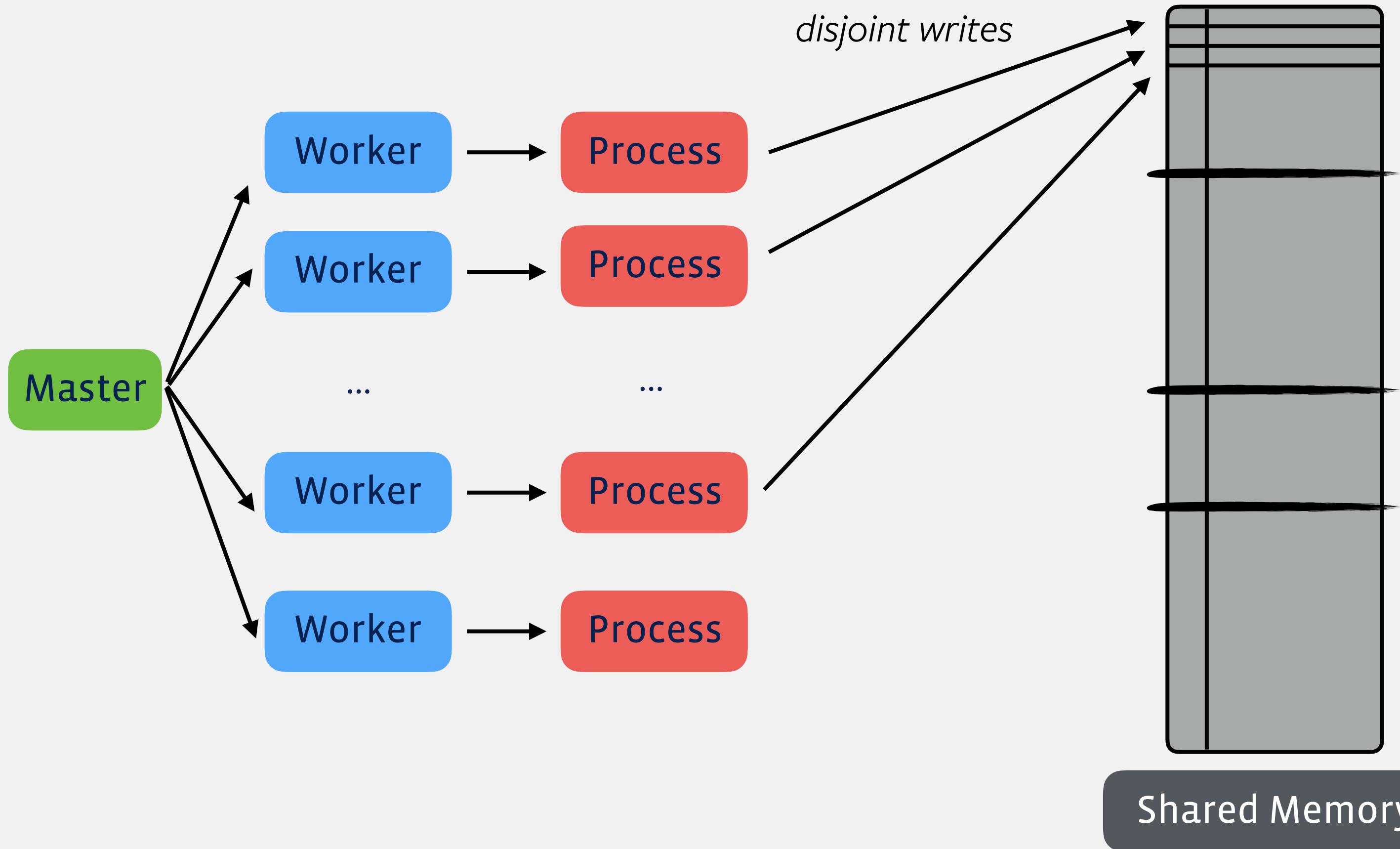
distribute!

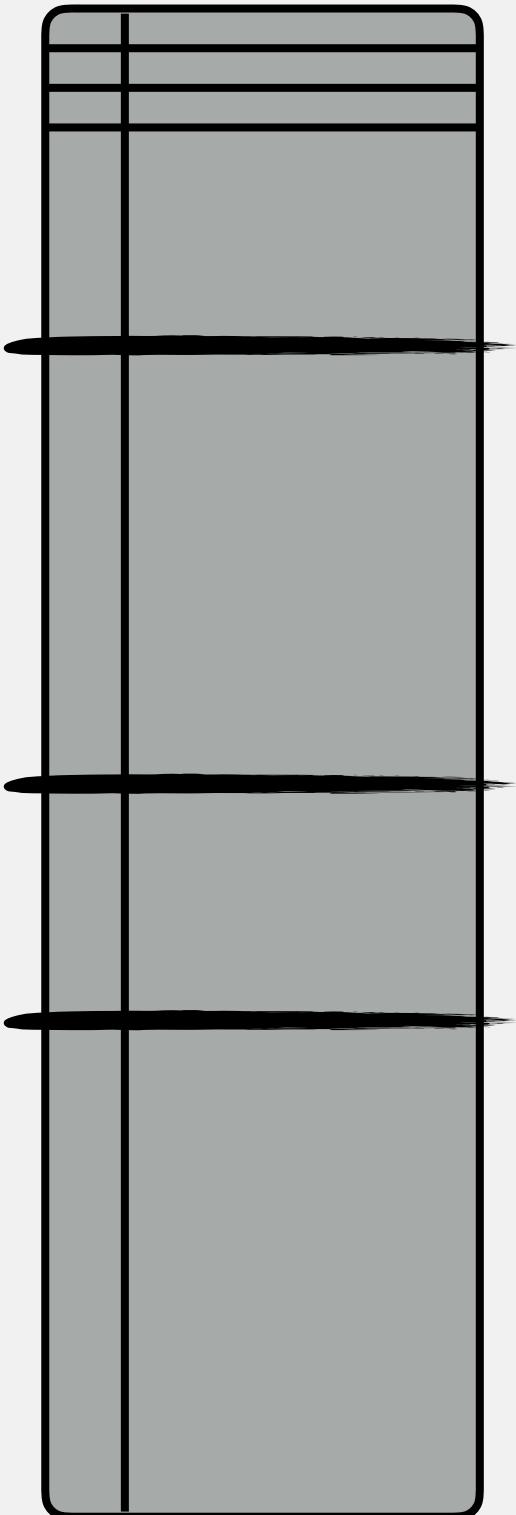
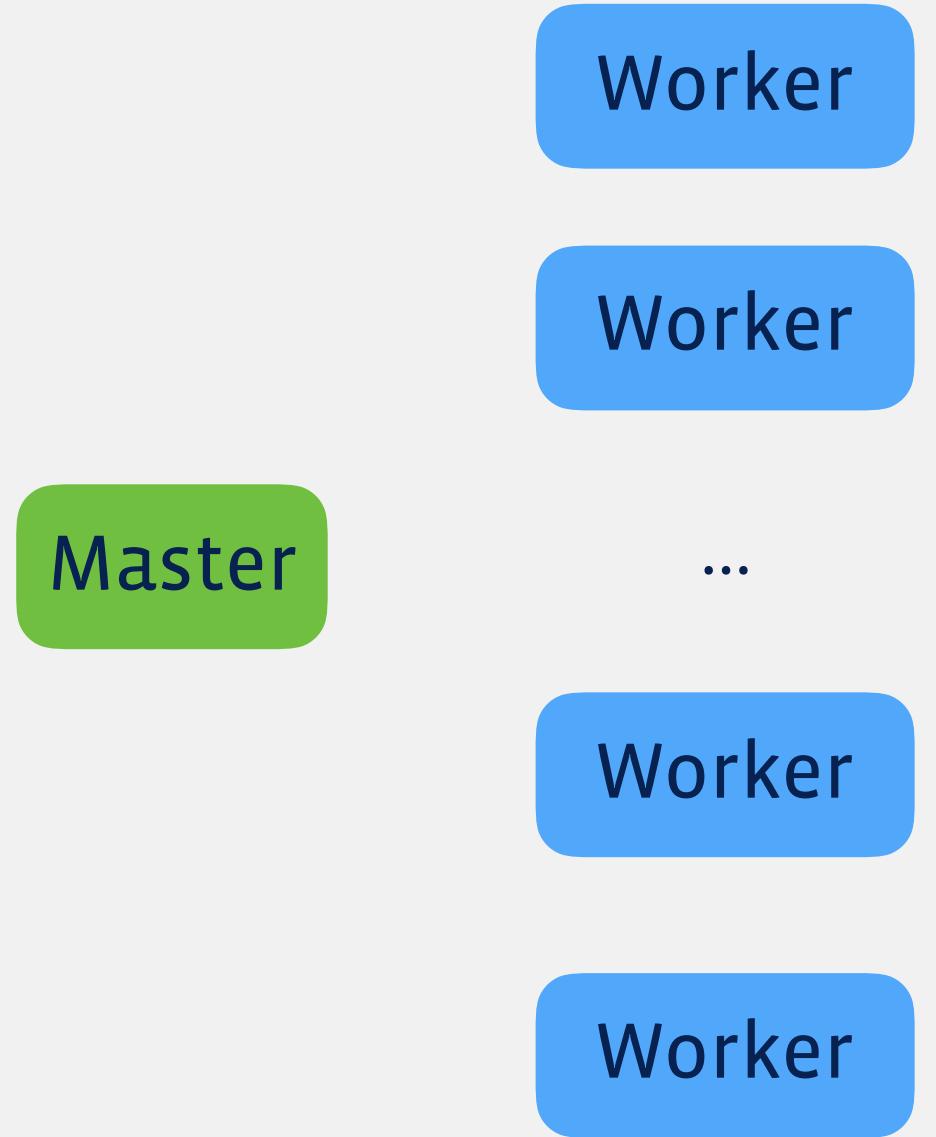




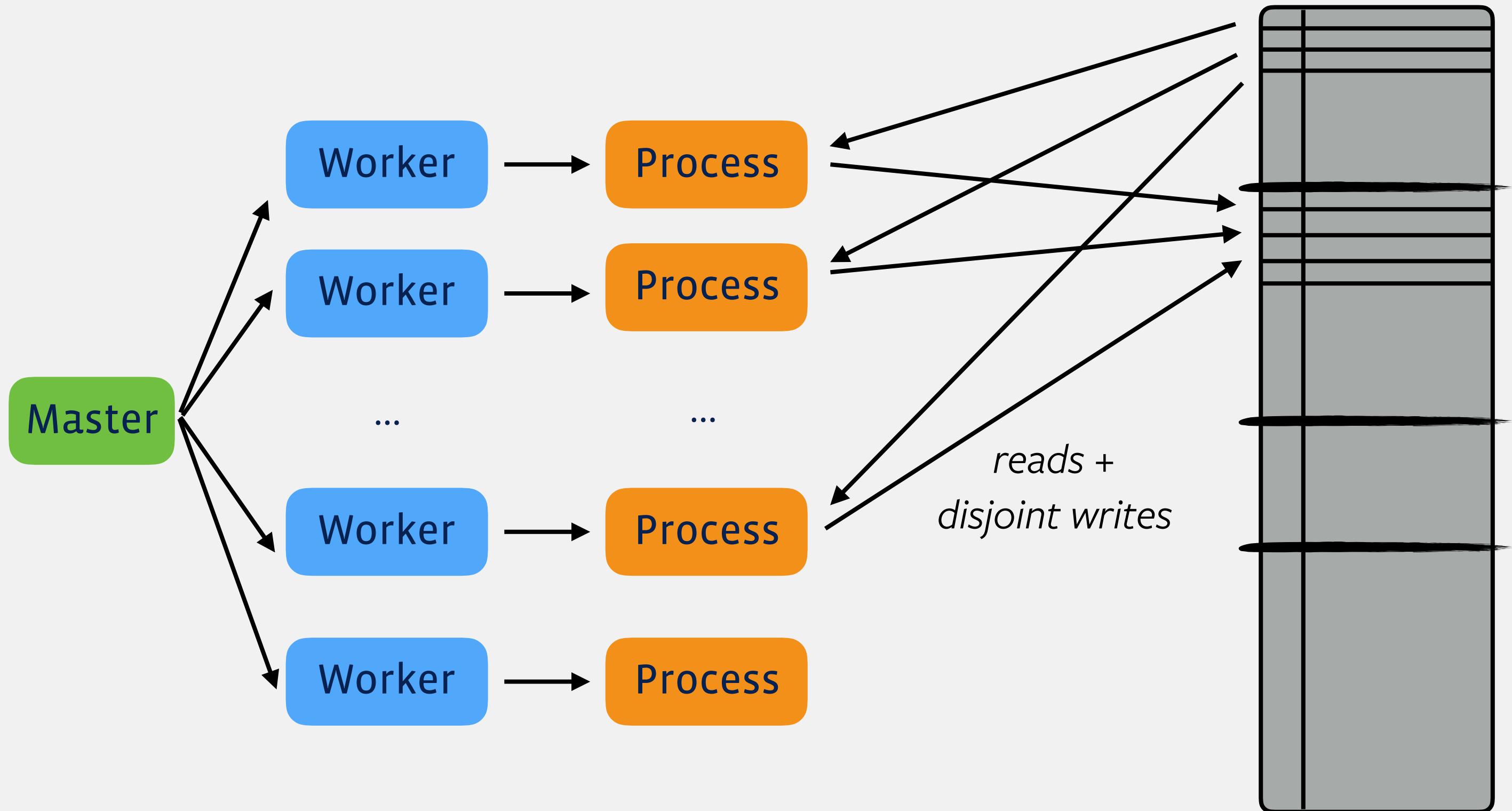




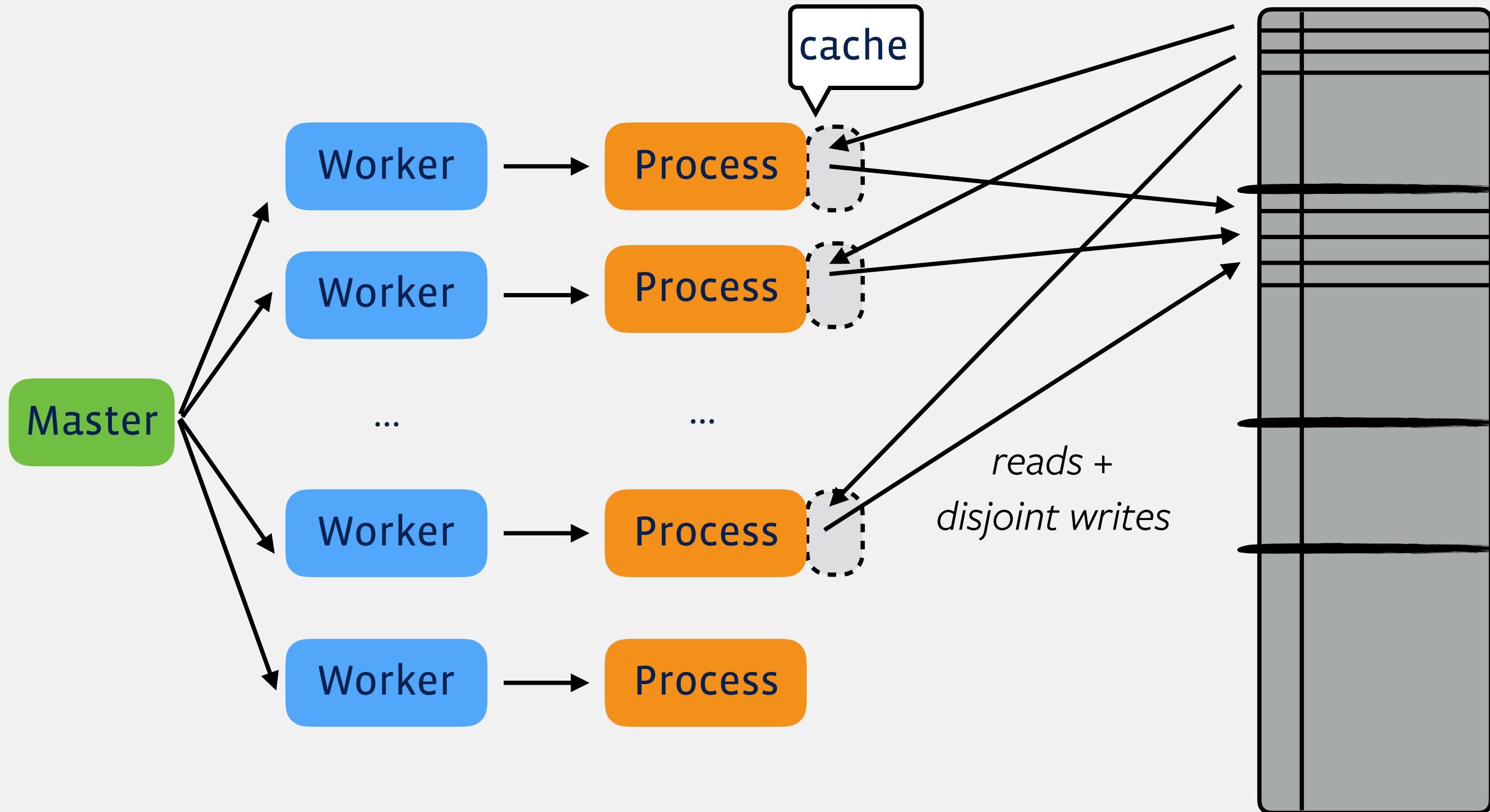




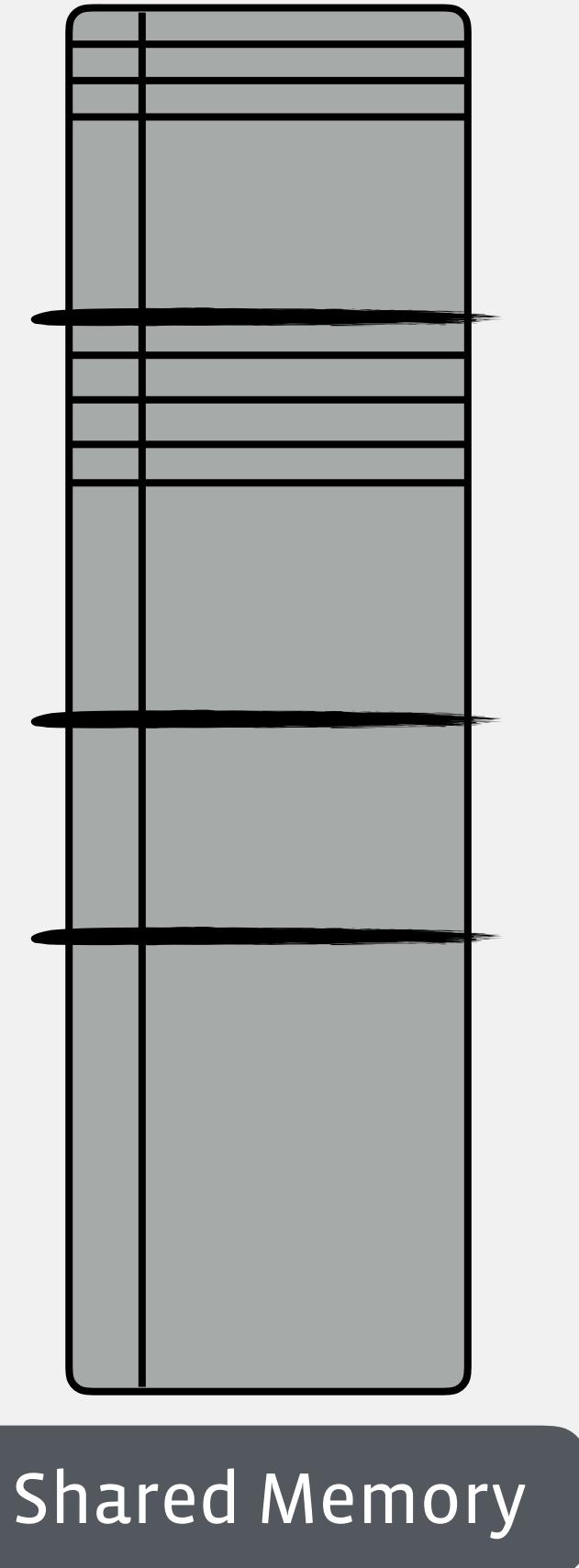
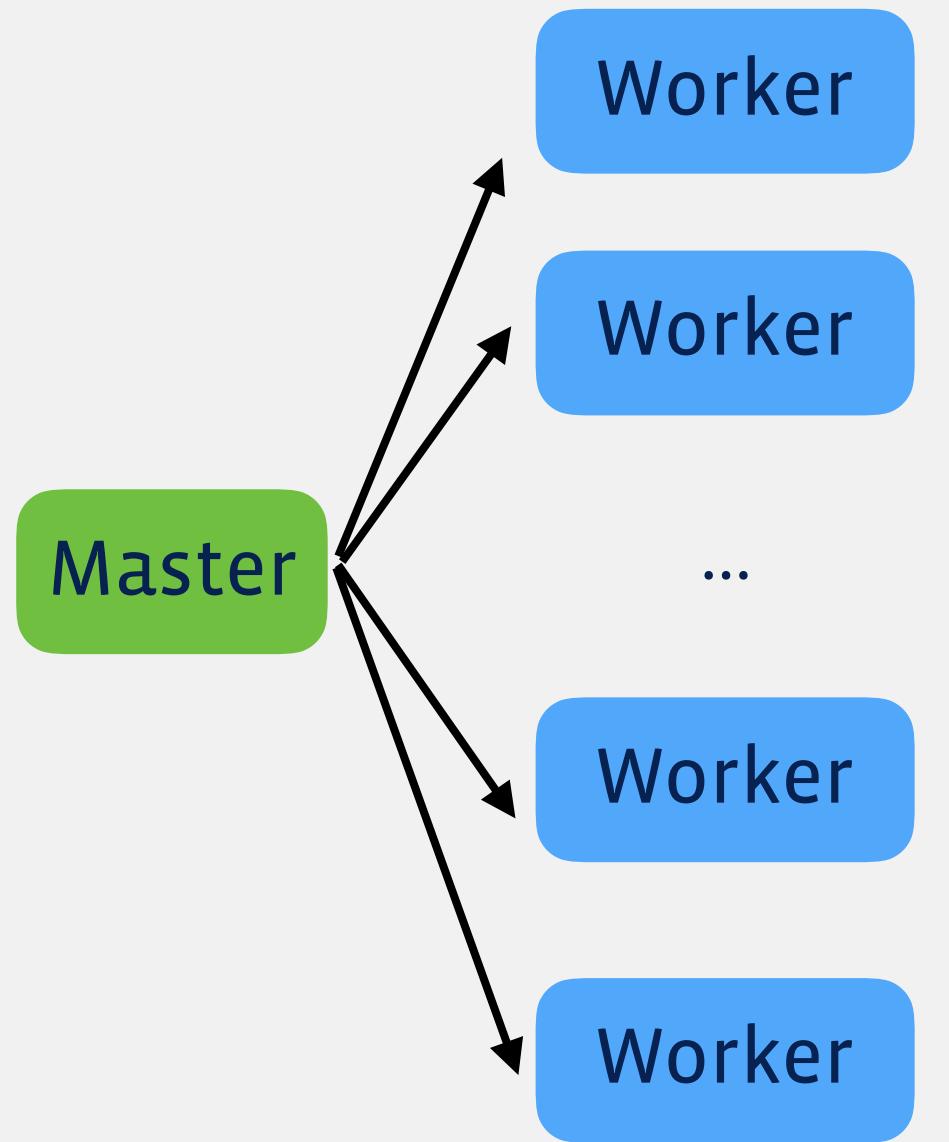
Shared Memory

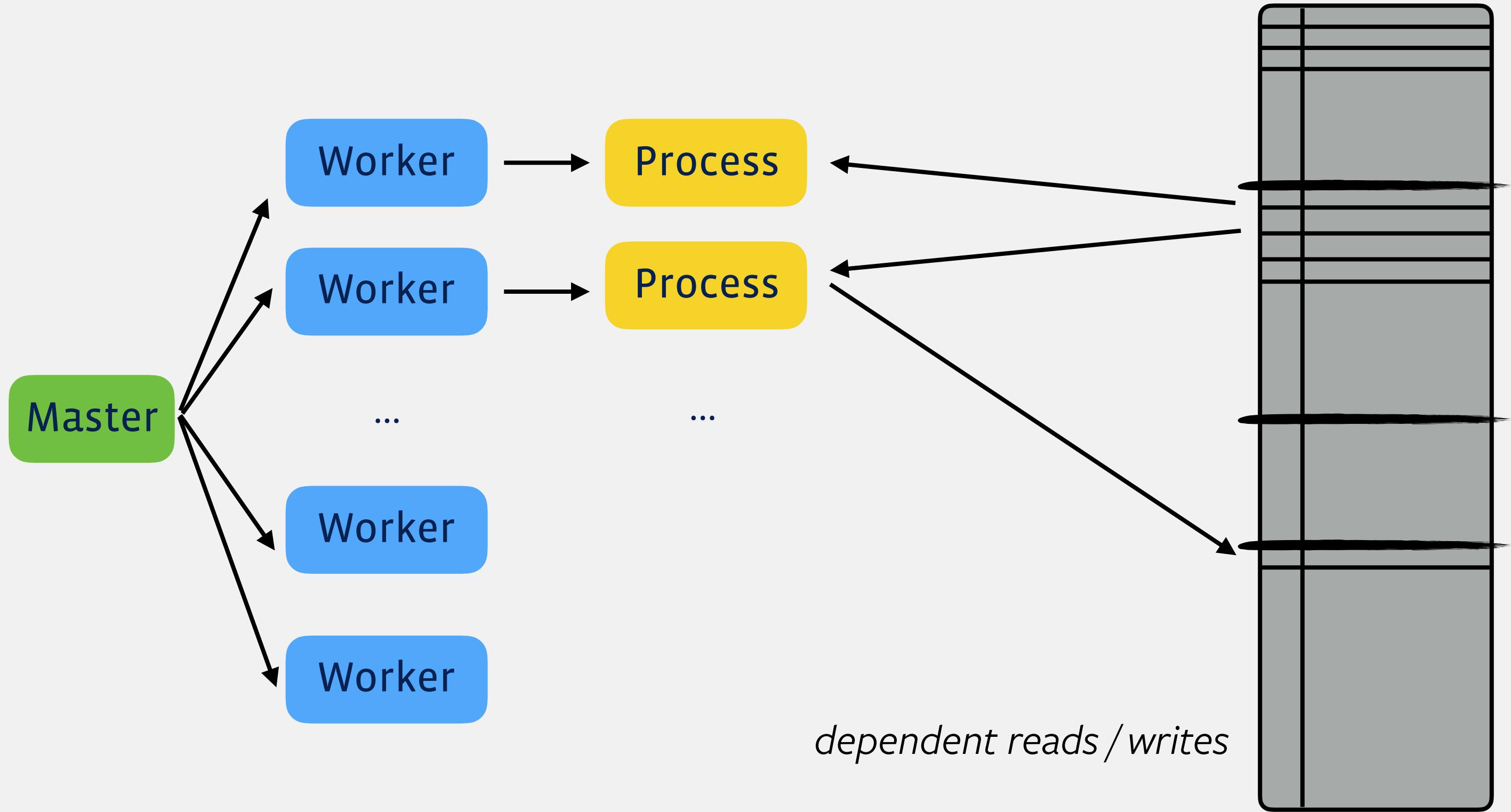


Shared Memory



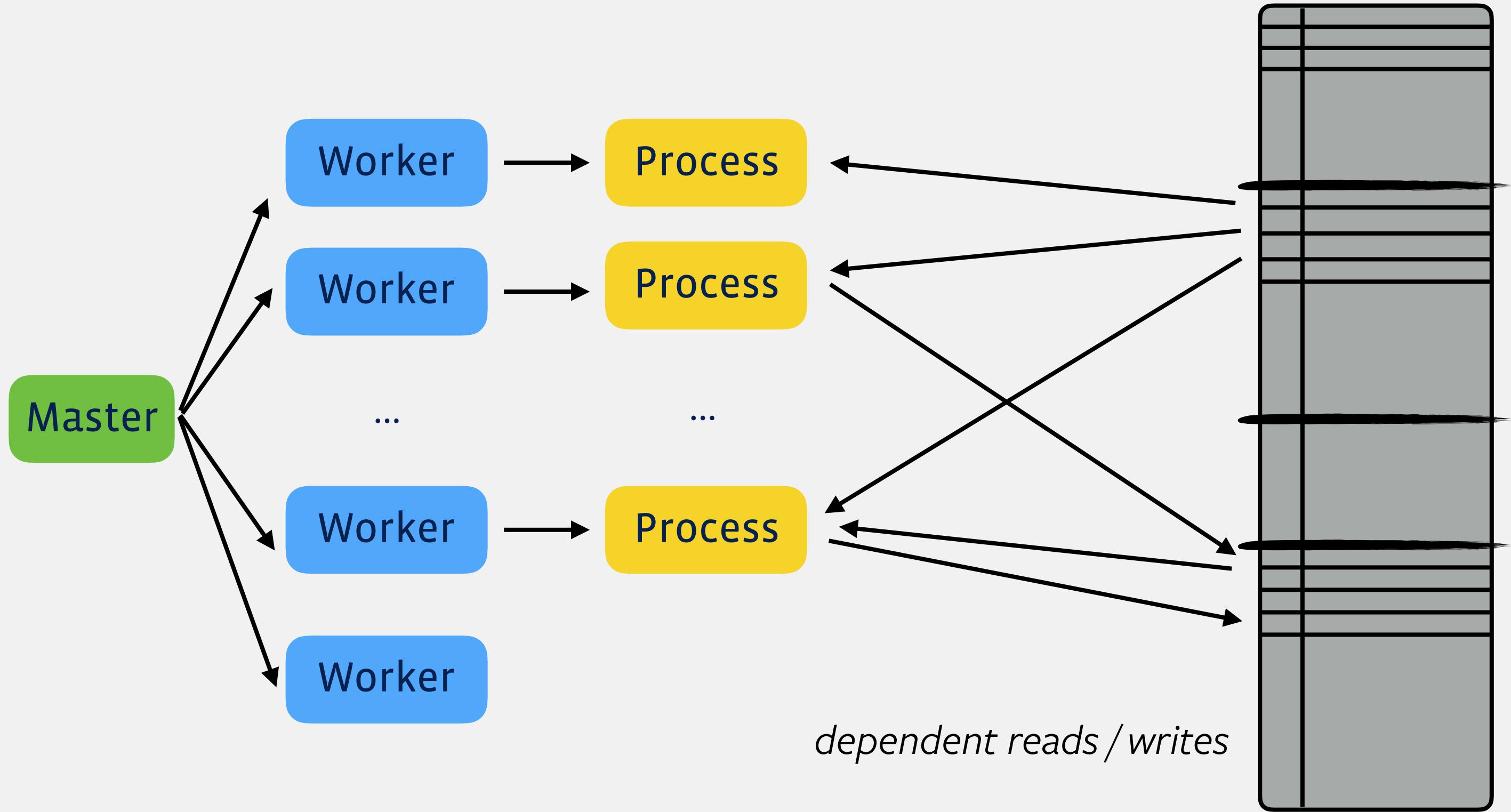
Shared Memory



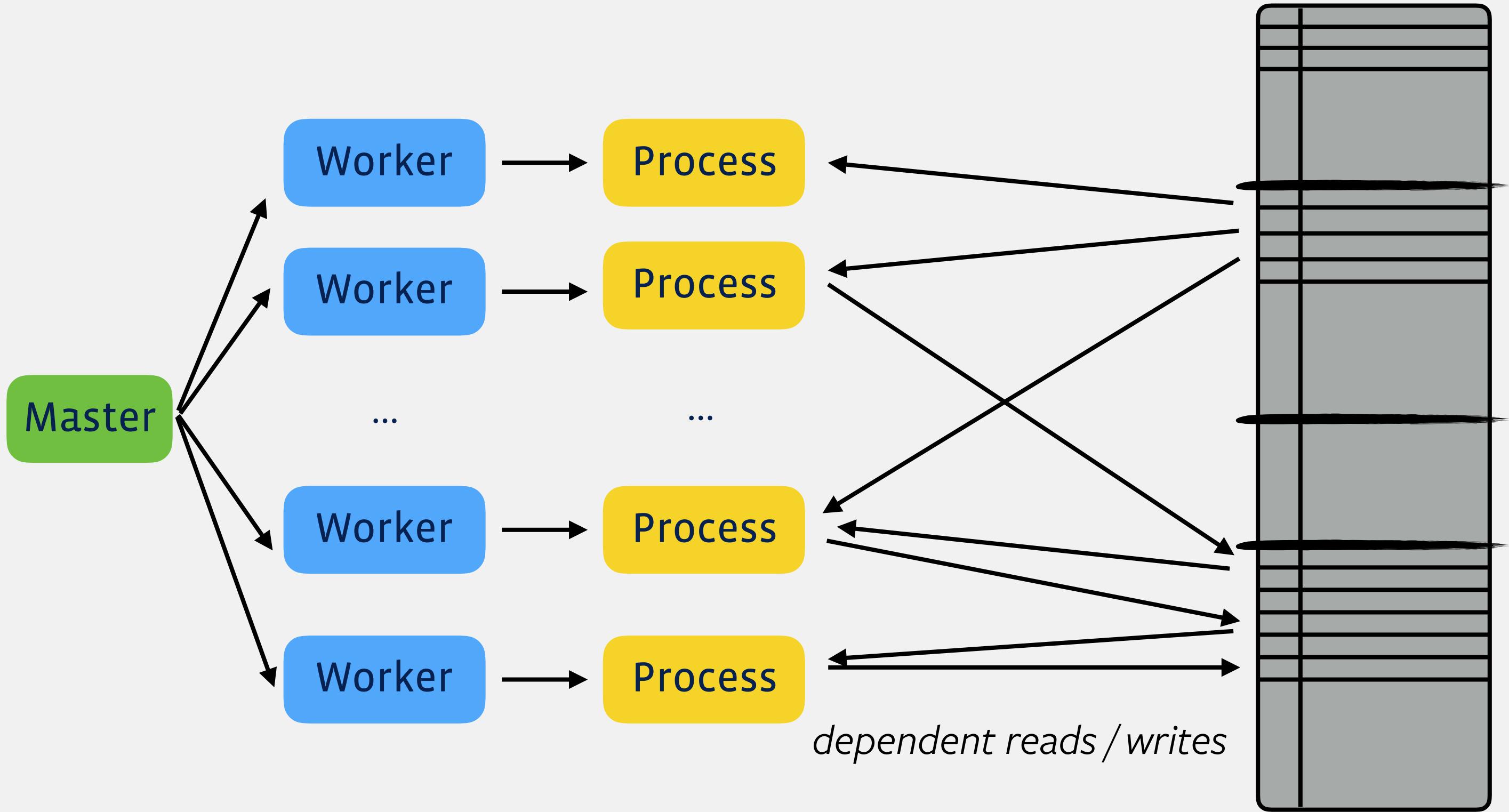


dependent reads / writes

Shared Memory



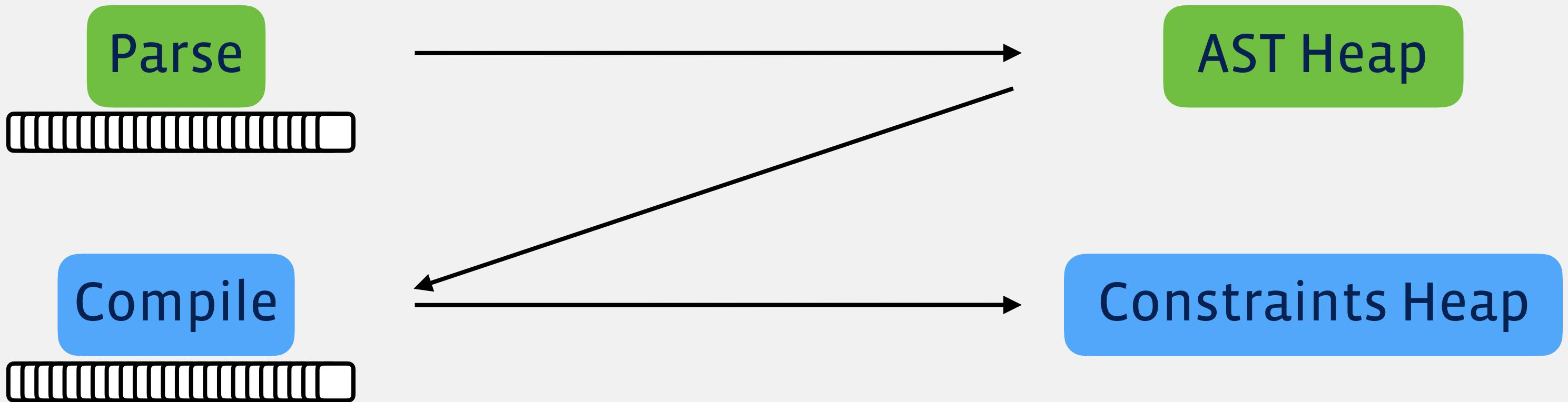
Shared Memory

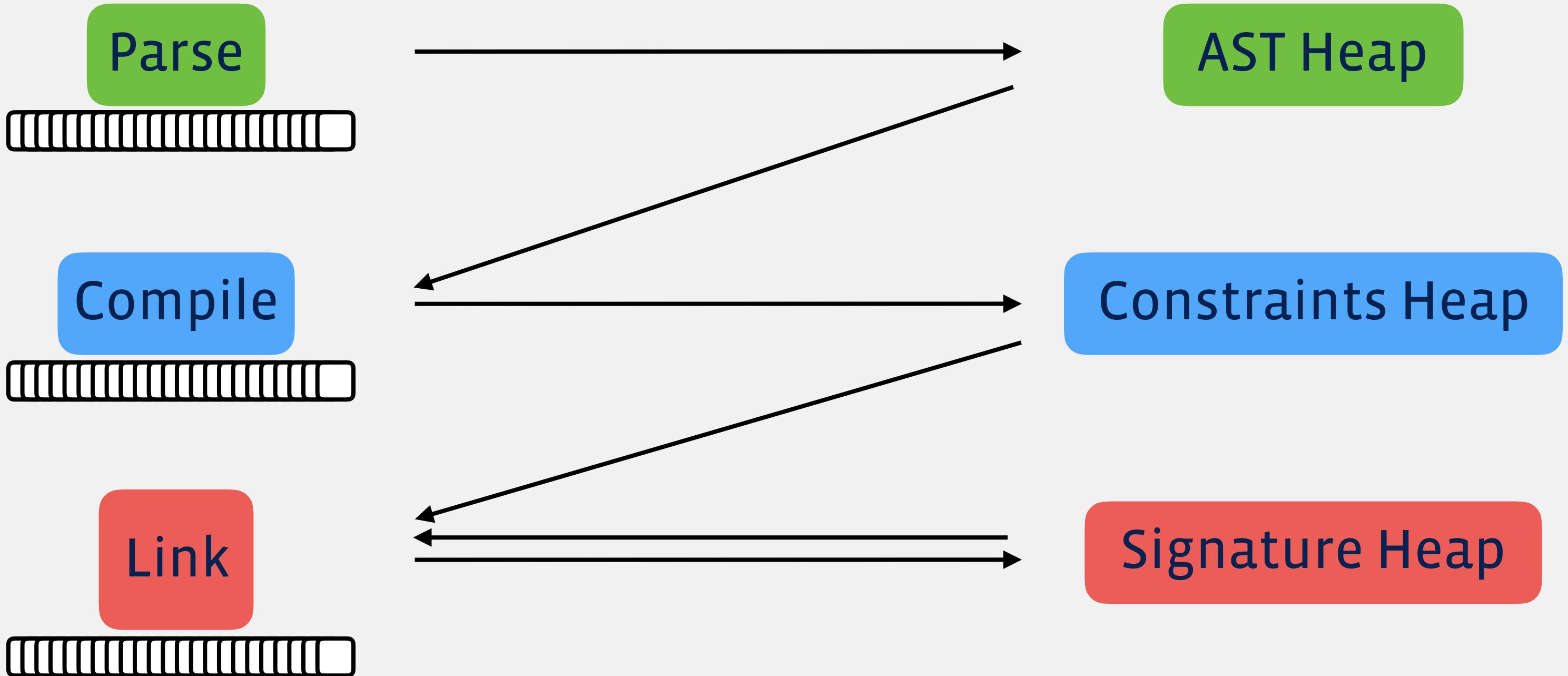


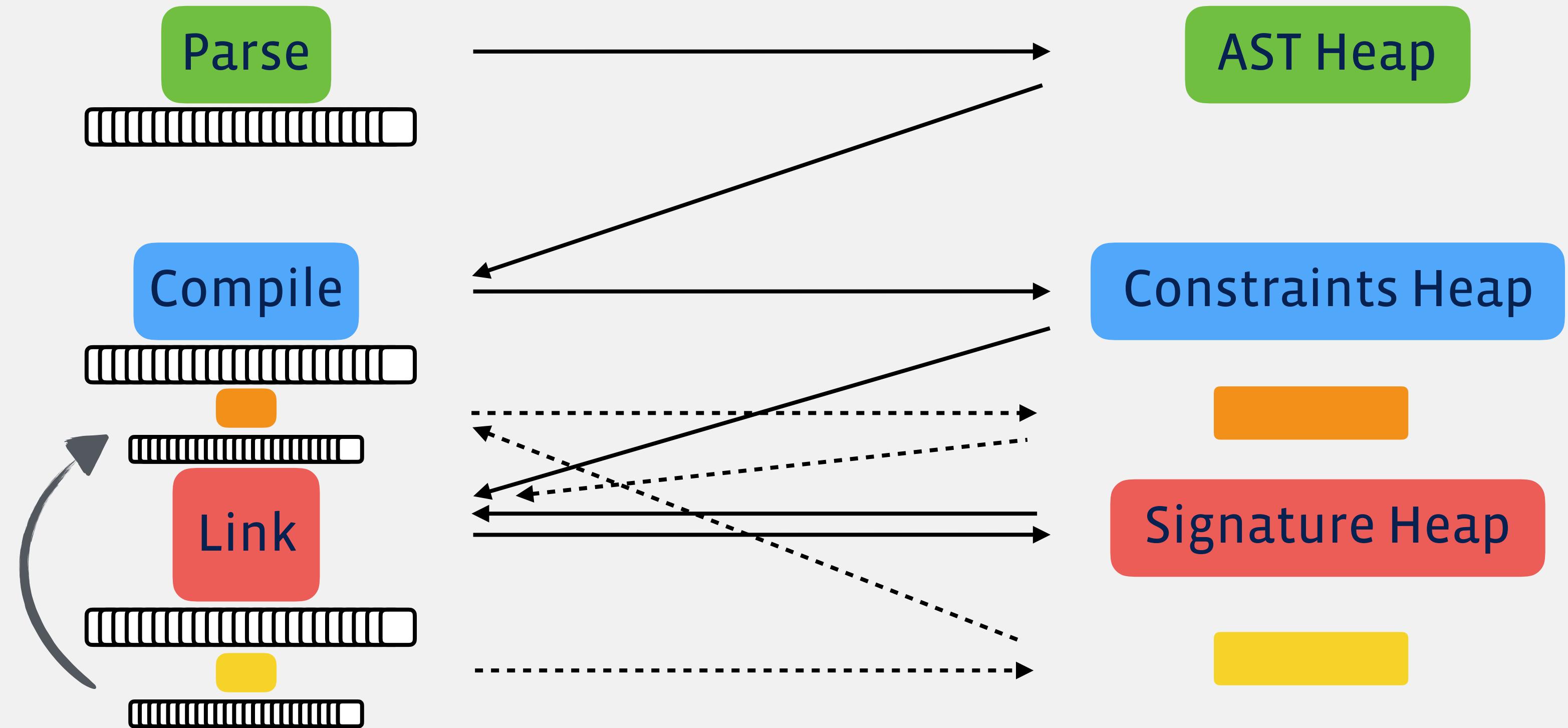
Shared Memory

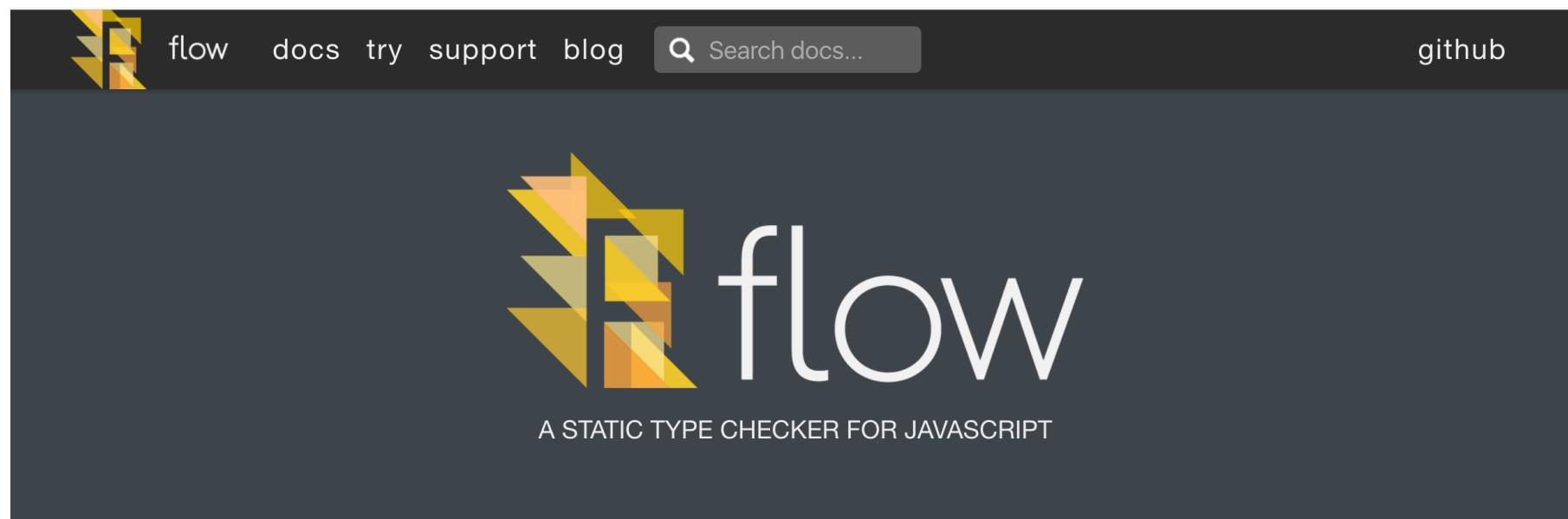
Putting everything together...











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