Jane Street

A Non-allocating Option

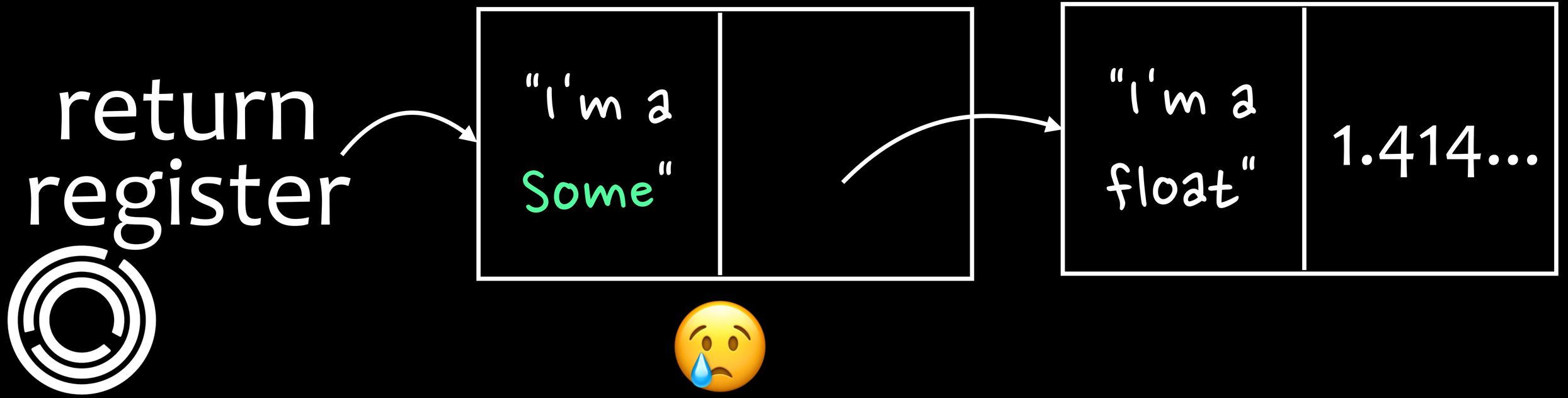
Richard A. Eisenberg
Jane Street
reisenberg@janestreet.com

Diana Kalinichenko
Jane Street
dkalinichenko@janestreet.com

Saturday, 7 September 2024
OCaml Workshop
Milan, Italy

```
type 'a option =
  | None
  | Some of 'a
```

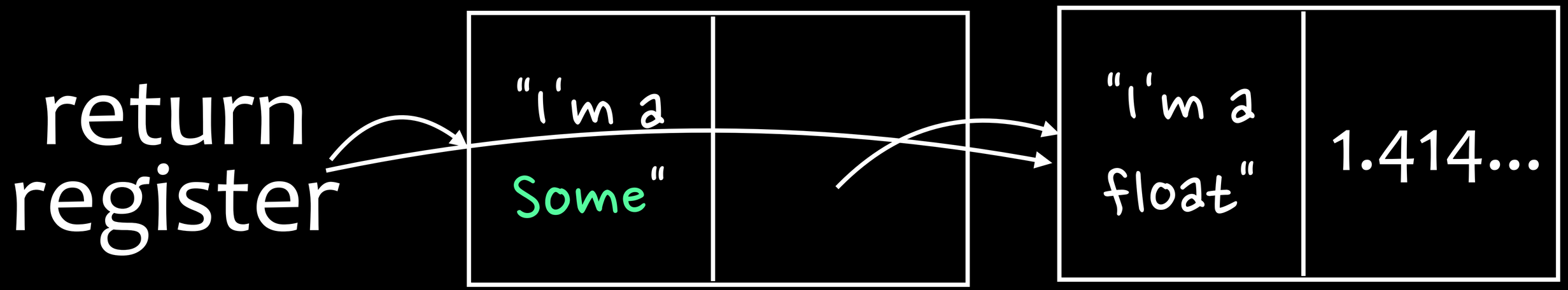
```
let safe_sqrt f =
    if f < 0.
    then None
    else Some (Float.sqrt f)
```



Can we avoid that allocation?

Idea:

Store Some x just like x.



Challenge:
What to do with None?

None has to be distinct from every other value.

None has to be distinct from every other value.

Use NULL.

Use NULL.

Other values:

Pointer	Tagged int
Not NULL	Not NULL

Challenge:

None vs Some None

NULL 💡 NULL

Disallow

`'a option option`

Challenge:
None vs Some None

Disallow

`'a or_null or_null`

Challenge:
`Null` vs `This Null`

Disallow

`'a or_null or_null`

Key Idea:
Separate *with-null types* from *no-null types*

like the way
`[@@immediate]` works

Demo Time

Challenge:
Type inference & defaults

ask me afterwards

Challenge: Flat float arrays

- Array representation is controlled by the first element.
- Thus, every type must be *separable*.

either every value is a `float`,
or every value is a non-`float`

- Thus, every type must be *separable*.

either every value is a `float`,
or every value is a non-`float`

`float or_null` is *not* separable.

😢

`float or_null` is *not* separable.

We cannot allow
`float or_null array`.

We cannot allow
`float or_null array`.

~~Ban `float or_null`?~~
~~This would also ban `float t`!~~

Ban `'a or_null array`?

An alternative approach that doesn't affect the type system at all is simply incompatible with the flat float array optimization. 😢

ask me for details

Implementation mostly complete on our branch!

Jane Street

A Non-allocating Option

Richard A. Eisenberg
Jane Street
reisenberg@janestreet.com

Diana Kalinichenko
Jane Street
dkalinichenko@janestreet.com

Saturday, 7 September 2024
OCaml Workshop
Milan, Italy

Question:

Can I call

```
val id : 'a -> 'a
```

with something of type

```
string or_null
```

?

it's complicated

```
module type S = sig
  type t
end
```

we can't have both M1 and F2.

```
module M1 : S = struct
  type t = string or_null
end
```

we choose F2.

```
module F2 (X : S) = struct
  type t = X.t or_null
end
```

we choose F2.

```
module F2 (X : S) = struct
  type t = X.t or_null
end
```

```
module M : S = ...
... M.t or_null ...
```

choosing F2 increases availability

```
module M : sig
  type 'a t
end = struct
  type 'a t = 'a
end
```

no-null

must also be no-null

`string or_null t` needs an annotation to work 😢

```
module M : sig
  val f : 'a -> 'a
end = struct
  let f x =
    (* build an ['a collection] *)
end
```

must be no-null

must be no-null

Library functions will need annotations to accept e.g. `string or_null`. 😢