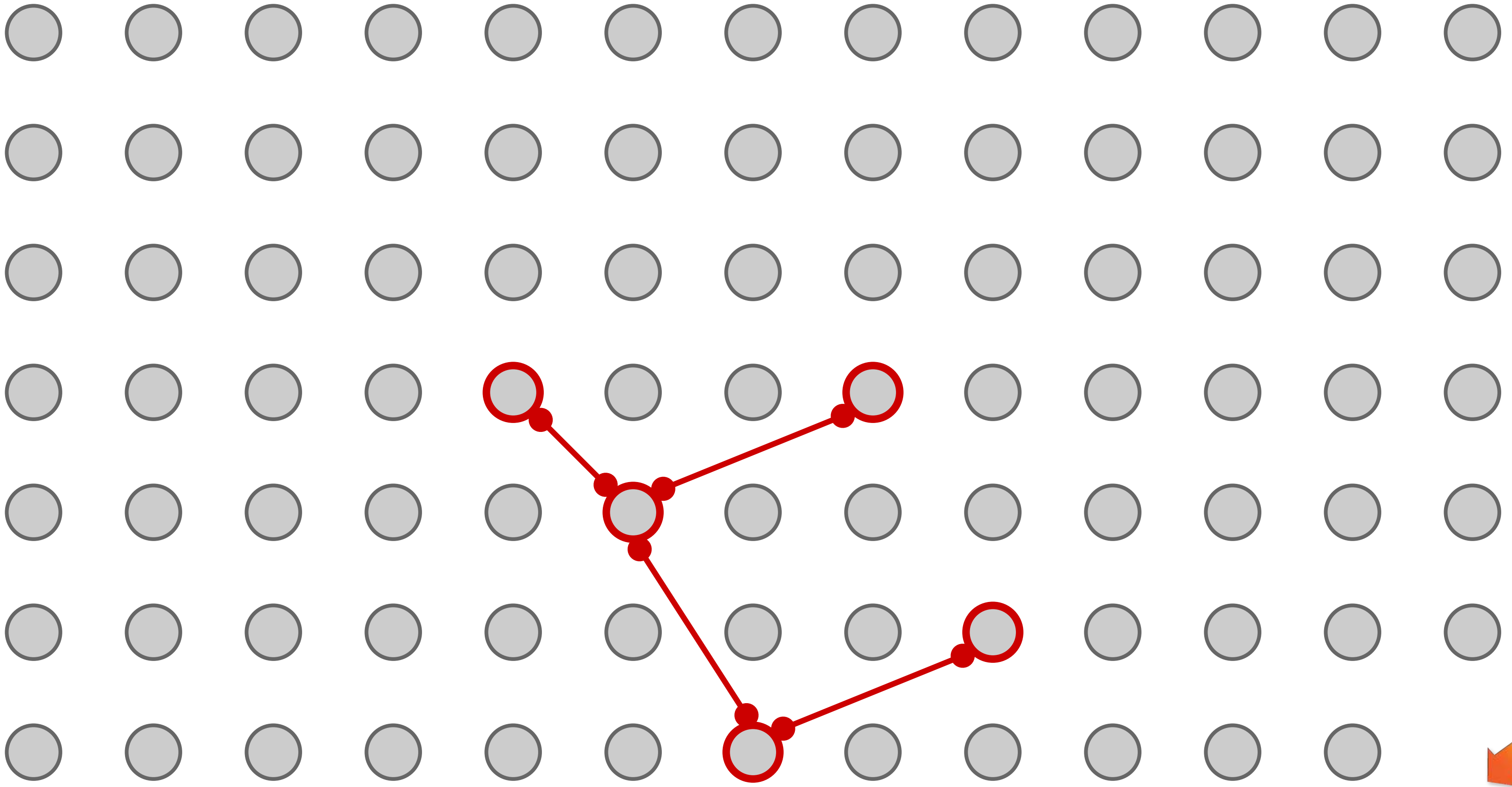
-XStaticPointers

Lightweight remotable computations

Mathieu Boespflug, Facundo Domínguez,
Simon Peyton Jones (MS Research)

Tweag I/O

Tweag I/O



Example: remote calculator (I)

```
data Arith = Plus Double Double
           | Mult Double Double
           | Neg Double

instance Serializable ArithOp
```

```
server :: Process ()
server = forever $ do
  expect >>= say . show . \case of
    Plus x y -> x + y
    Mult x y -> x * y
    Neg x -> negate x
```

```
let'sDoSomeMath :: ProcessId -> Process ()
let'sDoSomeMath there = do
    send there $ Plus 10 2
    send there $ Mult (2^10) (3^10)
    send there $ Neg 1
```

Tweag I/O

Life and death of calculators

```
data Arith = Plus Double Double
           | Mult Double Double
           | Neg Double

instance Serializable ArithOp
```

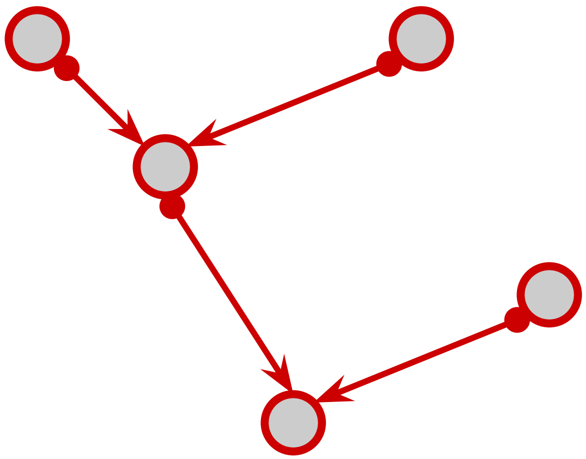
```
server :: Process ()
server = forever $ do
  expect >>= say . show . \case of
    Plus x y -> x + y
    Mult x y -> x * y
    Neg x -> negate x
```

```
I'mASupervisor :: ProcessId -> Process ()
I'mASupervisor there = forever $ do
    monitor there
    MonitorRef _ _ <- expect
    pid <- restartServer
    I'mASupervisor pid
```

Tweag I/O

Supervision hierarchies

“Reports to”



Calculation type is ...

★ … indirect;
★ … error prone;
★ … difficult to extend;
★ … antimodular.

Example: Remote calculator (II)

```
plus, mult :: Int -> Int -> Process ()
plus x y = say $ show $ x + y
mult x y = say $ show $ x * y

neg :: Int -> Process ()
neg x = say $ show $ negate x
```

```
let'sDoSomeMath :: NodeId -> Process ()
let'sDoSomeMath there = do
    spawn there $ plus 10 2
    spawn there $ mult (2^10) (3^10)
    spawn there $ neg 1
```

Tweag I/O

Spawning functions is …

- … direct;
- … error proof;
- … easy to extend;
- … modular.

Tweag I/O

```haskell
send :: Serializable a => NodeId -> a -> Process ()
expect :: Serializable a => Process a

spawn :: NodeId -> Closure (Process ()) -> Process ()
```

Serialization

```
data Static a
data Closure a =
  Closure (Static a)      -- code pointer
          Env
type Env = ByteString

instance Serializable (Static a)
instance Serializable (Closure a)
```

Tweag I/O

Serialization

```
data Static a
data Closure a =
  Closure (Static (Env -> a))
          Env
type Env = ByteString

instance Serializable (Static a)
instance Serializable (Closure a)
```

Winging it with Template Haskell

```haskell
plus, mult :: (Int, Int) -> Process ()
plus (x, y) = say $ show $ x + y
mult (x, y) = say $ show $ x * y

neg :: Int -> Process ()
neg x = say $ show $ negate x

remotable ['plus, 'mult, 'neg]
```

```haskell
let'sDoSomeMath :: NodeId -> Process ()
let'sDoSomeMath there = do
    spawn there $ $(mkClosure 'plus) (10, 2)
    spawn there $ $(mkClosure 'mult) (2^10, 3^10)
    spawn there $ $(mkClosure 'neg) 1
```

Tweag I/O

What's missing …

- ★ TH splices clunky, bad error messages.
- ★ Unnatural uncurrying of remotable functions.
- ★ Need to declare upfront what is remotable.
- ★ Can't remote arbitrary expressions.
- ★ Remote tables management anti-modular.

Tweag I/O

Example: Remote calculator (IV)

```
plus, mult :: Int -> Int -> Process ()
plus x y = say $ show $ x + y
mult x y = say $ show $ x * y

neg :: Int -> Process ()
neg x = say $ show $ negate x
```

```
let'sDoSomeMath :: NodeId -> Process ()
let'sDoSomeMath there = do
  spawn there $ closure $ static (plus 10 2)
  spawn there $ closure $ static (mult (2^10) (3^10))
  spawn there $ closure $ static (neg 1)
```

Tweag I/O

-XStaticPointers

- ★ New syntactic form:

$$e \ ::= \ \ldots \ | \ \textbf{static} \ e_1$$

- ★ Meaning: “label of e_1”.
- ★ Restrictions: e_1 must be *closed*.

$$\frac{\Delta \vdash e :: a}{\Delta; \Gamma \vdash \mathtt{static}\ e :: \mathtt{Static}\ a}$$

Desugaring static pointers

C[static e] ↘ f :: a; f = e; C[s]

where

★ C is any context;
★ s is a *value* of type Static a;
★ f is a fresh (*unique*) name.

Tweag I/O

Static Pointers are ...

- ★ . . . guaranteed to be unique;
- ★ . . . free;
- ★ . . . modular;
- ★ . . . safe.

Tweag I/O

RPC calls

```
type Dict c = c => Dict

spawn :: NodeId -> Closure (Process ()) -> Process ()
send :: Serializable a => NodeId -> a -> Process ()
expect :: Serializable a => Process a

call :: Static (Dict (Serializable a))
     -> NodeId
     -> Closure (Process a)
     -> Process a
```

Let's implement (simplified) `call` ...

```
call :: Static (Dict (Serializable a))
     -> NodeId
     -> Closure (Process a)
     -> Process a
call sdict there cf = do
    here <- getSelfPid
    spawn there (closure cf `bindCP` sendBack dict here)
    x <- expect
    return x
```

```
bindCP :: Closure (Process a)
       -> Closure (a -> Process b)
       -> Closure (Process b)
```

Tweag I/O

Let's implement bindCP ...

```
bindCP :: Closure (Process a)
       -> Closure (a -> Process b)
       -> Closure (Process b)
bindCP cm ck =
    static (>>=) `closureApply` cm `closureApply` ck
```

Tweag I/O

Let's implement bindCP ...

```
bindCP :: forall a b. Closure (Process a)
       -> Closure (a -> Process b)
       -> Closure (Process b)
bindCP = Λa.Λb.λcm.λck.
    static (>>=Process a b) `closureApply` cm `closureApply` ck
```

Tweag I/O

Typing rule for polymorphic expr.

★ New syntactic form:

$$\texttt{e ::= ...} \mid \texttt{\textbf{static}}\ \texttt{e}_1$$

★ Meaning: “label of e_1”.

★ Restrictions: e_1 must be ***term*** *closed*.

$$\frac{\Delta;\, \textbf{types}(\Gamma) \vdash \texttt{e :: a}}{\Delta;\, \Gamma \vdash \texttt{static e} :: \texttt{Static a}}$$

Tweag I/O

From here to there

GHC Extension	implemented
distributed-static (combinator library)	implemented
distributed-process (Cloud Haskell)	implemented

- ★ Next up: **simplify**, **review**, **merge**.
- ★ Ideally: resistance to *malicious segfaults*.

Tweag I/O

Serialization

```
data Static a
data Closure a =
  Closure (Static (Env -> a))
          Env
type Env = ByteString

instance Serializable (Static a)
instance Serializable (Closure a)
```

Tweag I/O

Related work

★ Serialisation support in the RTS (Eden, Jost Berthold).
★ Serialisation as a library (Packman, Jost Berthold).
★ HdpH closures (Maier, Stewart, Trinder).

Tweag I/O

Conclusion

★ Static pointers: very general notion of
"value with a name".

★ Polymorphism is just as important in a distributed setting.

★ Much smaller language extension than native support for runtime reflection.

★ Beware of unimplemented sidenotes…

Tweag I/O

Thank you!