Skeletal semantics

actors ⟷ channels

“Skeletal Semantics”

“Skeletal Semantics”

Skeletal **Semantics**

$$\frac{t_1 \rightarrow \mathit{true} \qquad t_2 \rightarrow v_2}{\mathit{if}\ (t_1)\ \mathit{then}\ (t_2)\ \mathit{else}\ (t_3) \rightarrow v_2}$$

$$\frac{t_1 \rightarrow \mathit{false} \qquad t_3 \rightarrow v_3}{\mathit{if}\ (t_1)\ \mathit{then}\ (t_2)\ \mathit{else}\ (t_3) \rightarrow v_3}$$

Skeletal **Semantics**

Premises

$$\frac{t_1 \to \mathit{true} \qquad\qquad t_2 \to v_2}{\mathit{if}\ (t_1)\ \mathit{then}\ (t_2)\ \mathit{else}\ (t_3) \to v_2}$$

$$\frac{t_1 \to \mathit{false} \qquad\qquad t_3 \to v_3}{\mathit{if}\ (t_1)\ \mathit{then}\ (t_2)\ \mathit{else}\ (t_3) \to v_3}$$

Conclusions

Evaluation relation

“Skeletal Semantics”

Skeletal *Semantics*

$$\mathit{if}\,(t_1, t_2, t_3) := \left[\, H(x_i, t_1, x_1)\;;\left(\begin{array}{c} \mathrm{isTrue}(x_1)\;;H(x_i, t_2, x_o) \\ \mathrm{isFalse}(x_1)\;;H(x_i, t_3, x_o) \end{array}\right) \right]$$

Skeletal Semantics

$$\overset{\substack{\text{Constructor}\\ \downarrow}}{\mathit{if}}\,(t_1, t_2, t_3) := \left[\underset{\substack{\uparrow\\ \text{Hook judgement}}}{H}(x_i, t_1, x_1) ; \begin{pmatrix} \overset{\substack{\text{Filters}\\ \downarrow}}{\mathrm{isTrue}}(x_1) ; H(x_i, \overset{\substack{\text{Terms}\\ \downarrow}}{t_2}, x_o) \\ \mathrm{isFalse}(x_1) ; H(\underset{\substack{\uparrow\\ \text{Flow variables}}}{x_i}, t_3, x_o) \end{pmatrix} \right]$$

Skel and Necro

```
val eval_if (xi, t) =
  let If (t1, t2, t3) = t in
  let x1 = eval (xi, t1) in
  branch
    let x1 = isTrue (x1) in eval (x1, t2)
  or
    let x1 = isFalse (x1) in eval (x1, t3)
  end
```

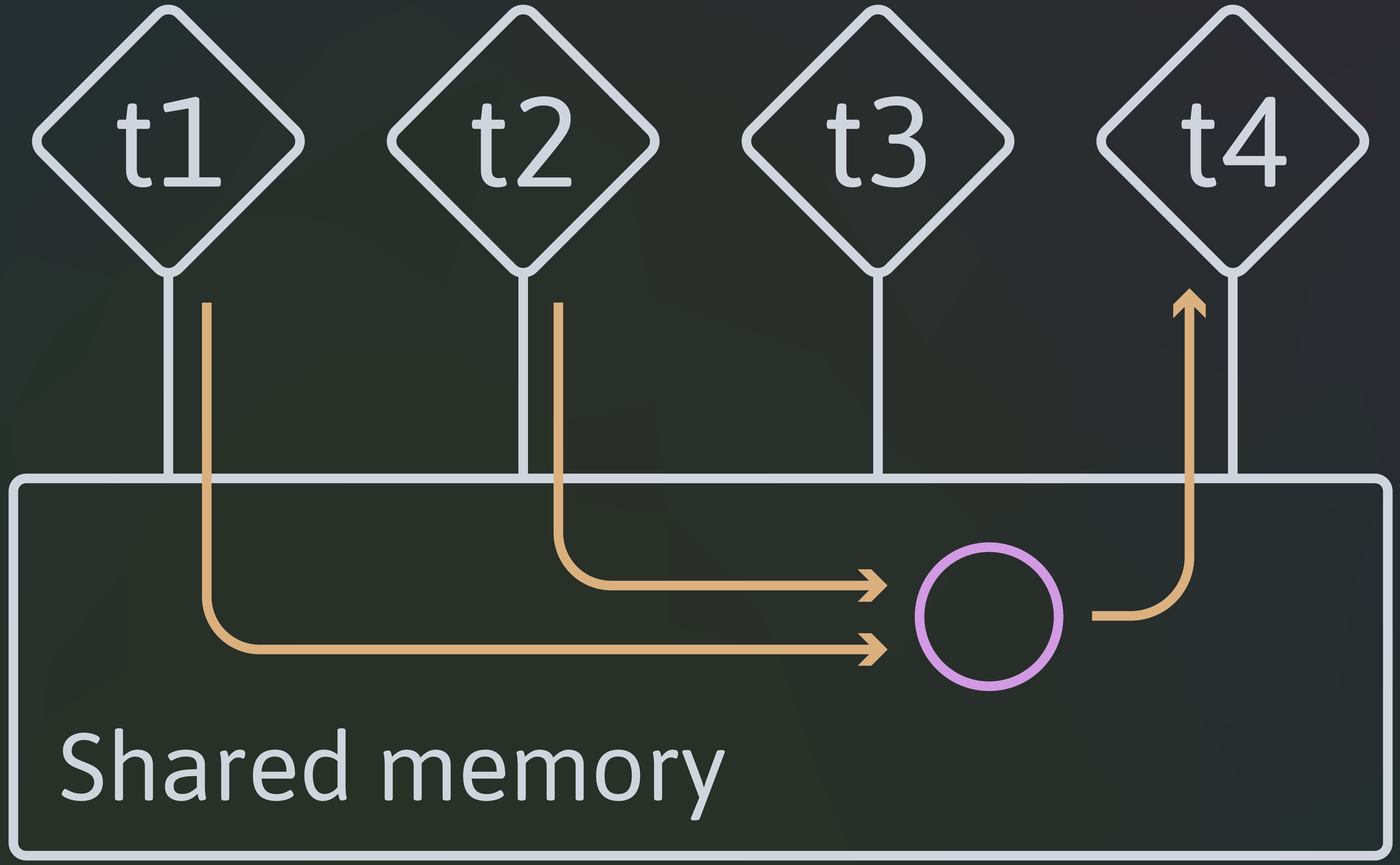
Skel and Necro

```
(* ... *)
let eval_if =
  function (xi, t) →
  begin match expr with
  | If (t1, t2, t3) →
    let* x1 = apply1 eval (xi, t1) in
    M.branch [
      (function () →
        let* x1 = apply1 isTrue x1 in
        apply1 eval (x1, t2)
      end) ;
      (function () →
        let* x1 = apply1 isFalse x1 in
        apply1 eval (x1, t3)
      end)
    ]
  | _ → M.fail ""
  end
(* ... *)
```

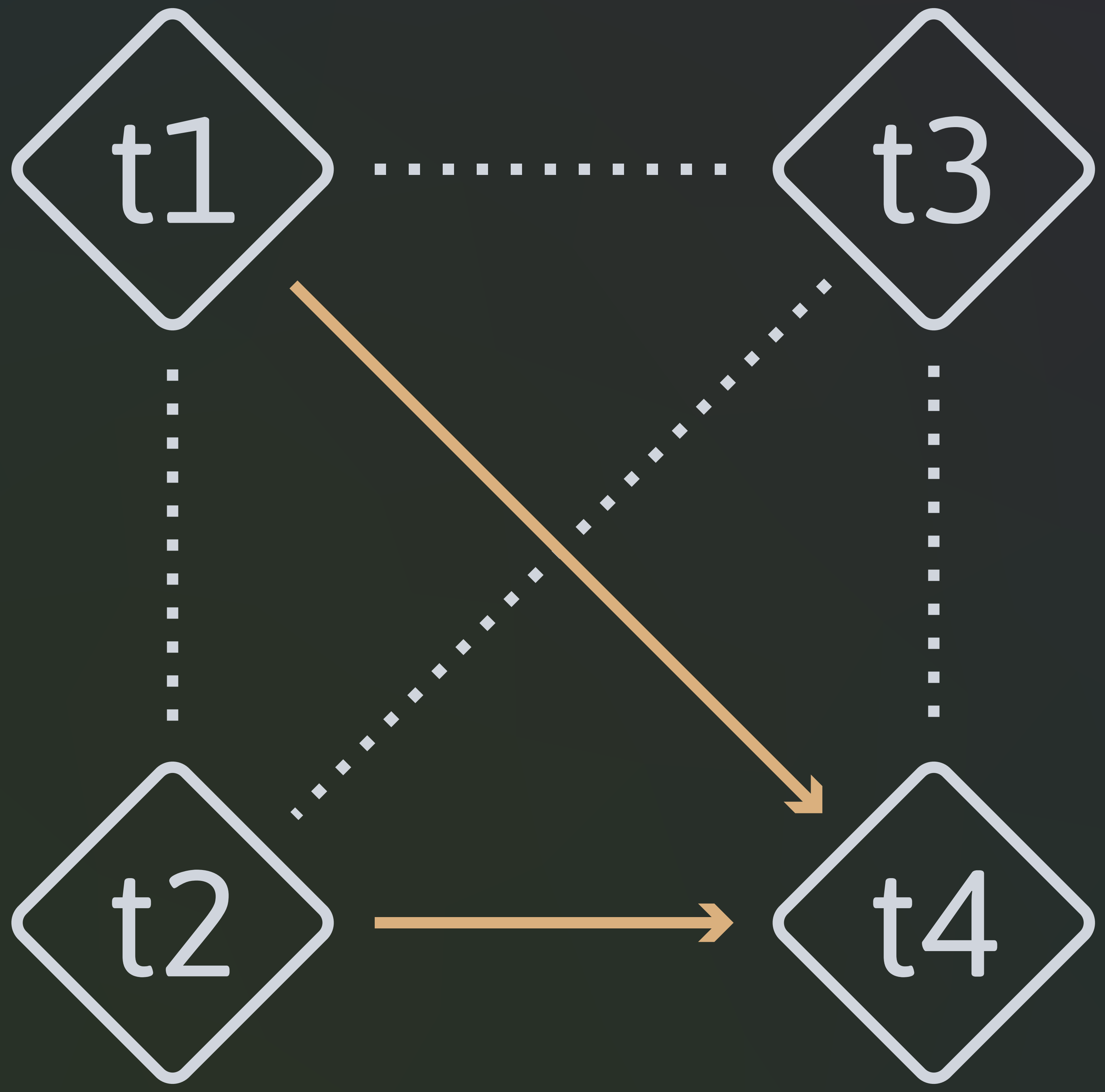
Concurrency

Shared memory
vs.
Message passing

Shared memory

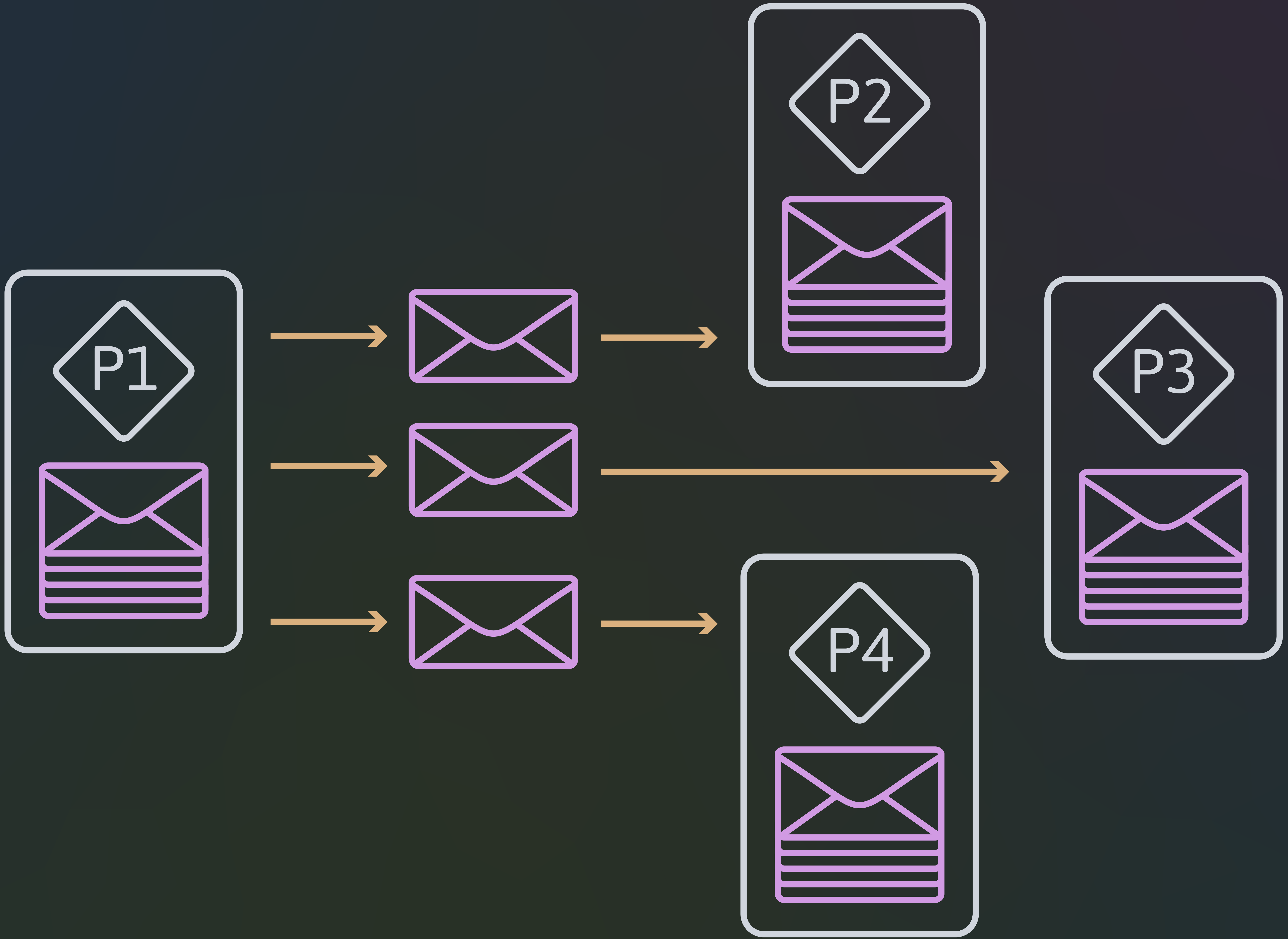


Message passing

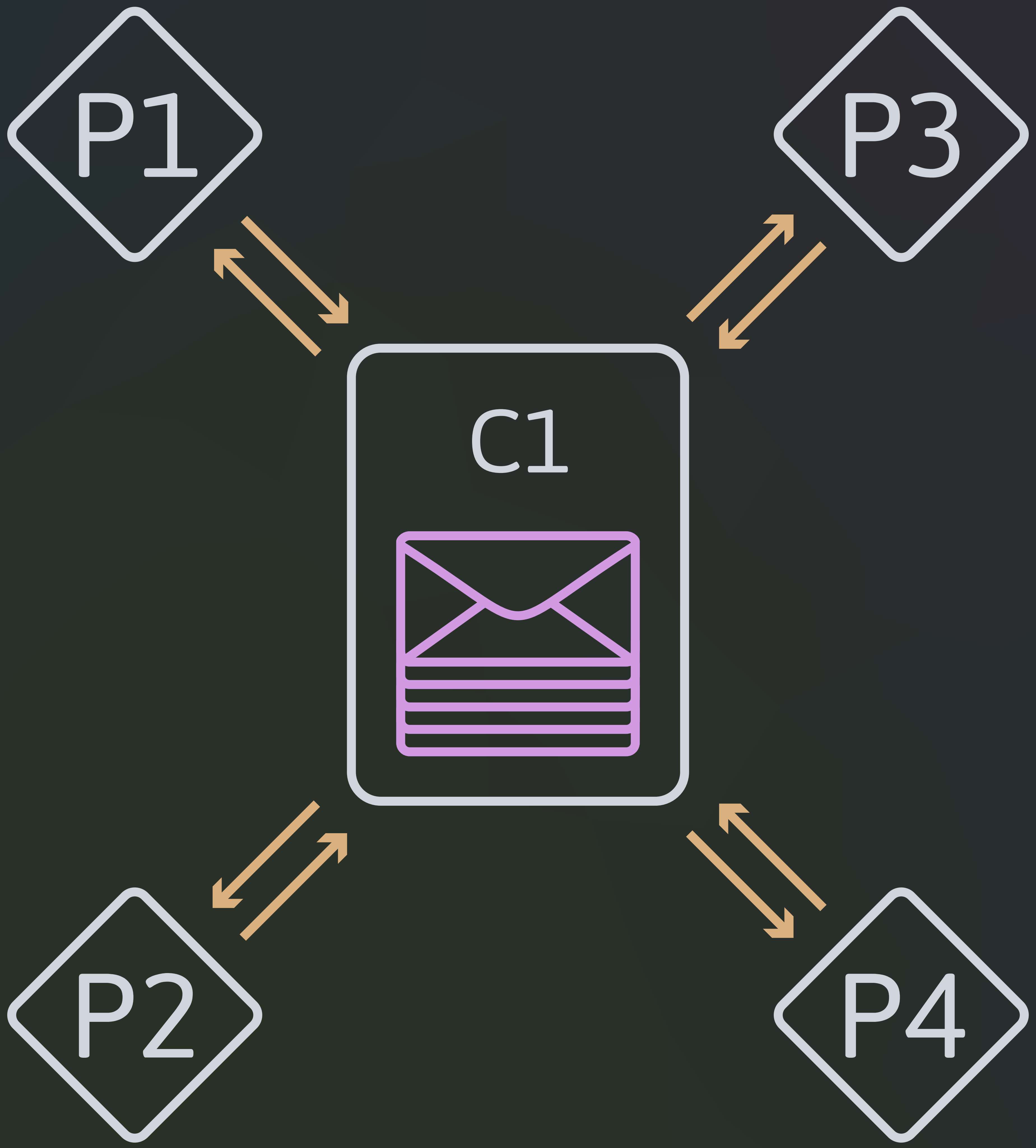


Actors
vs.
Channels

Actors



Channels



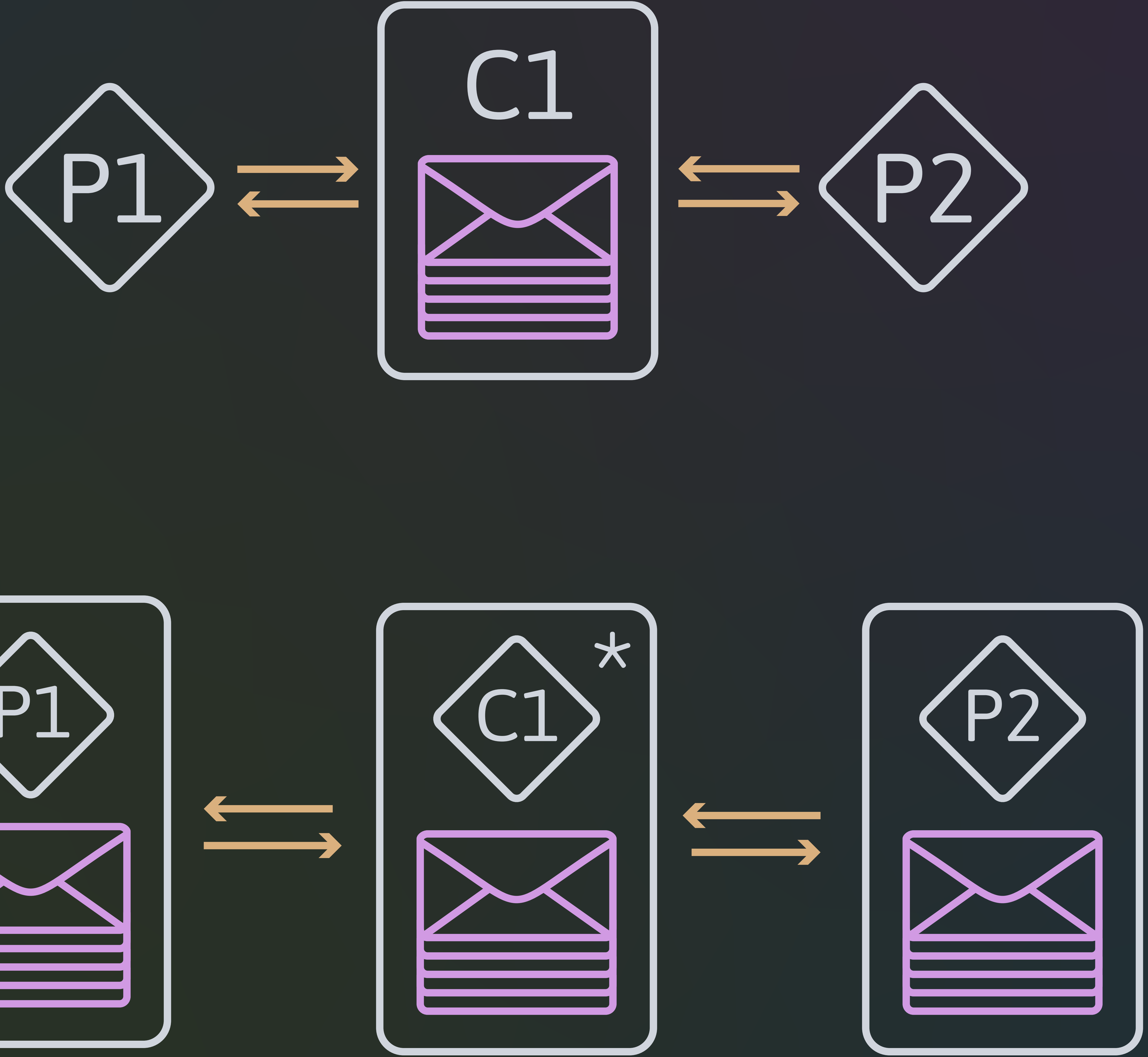
Equivalence between actors and channels

Channels to actors

Channels

↓

Actors

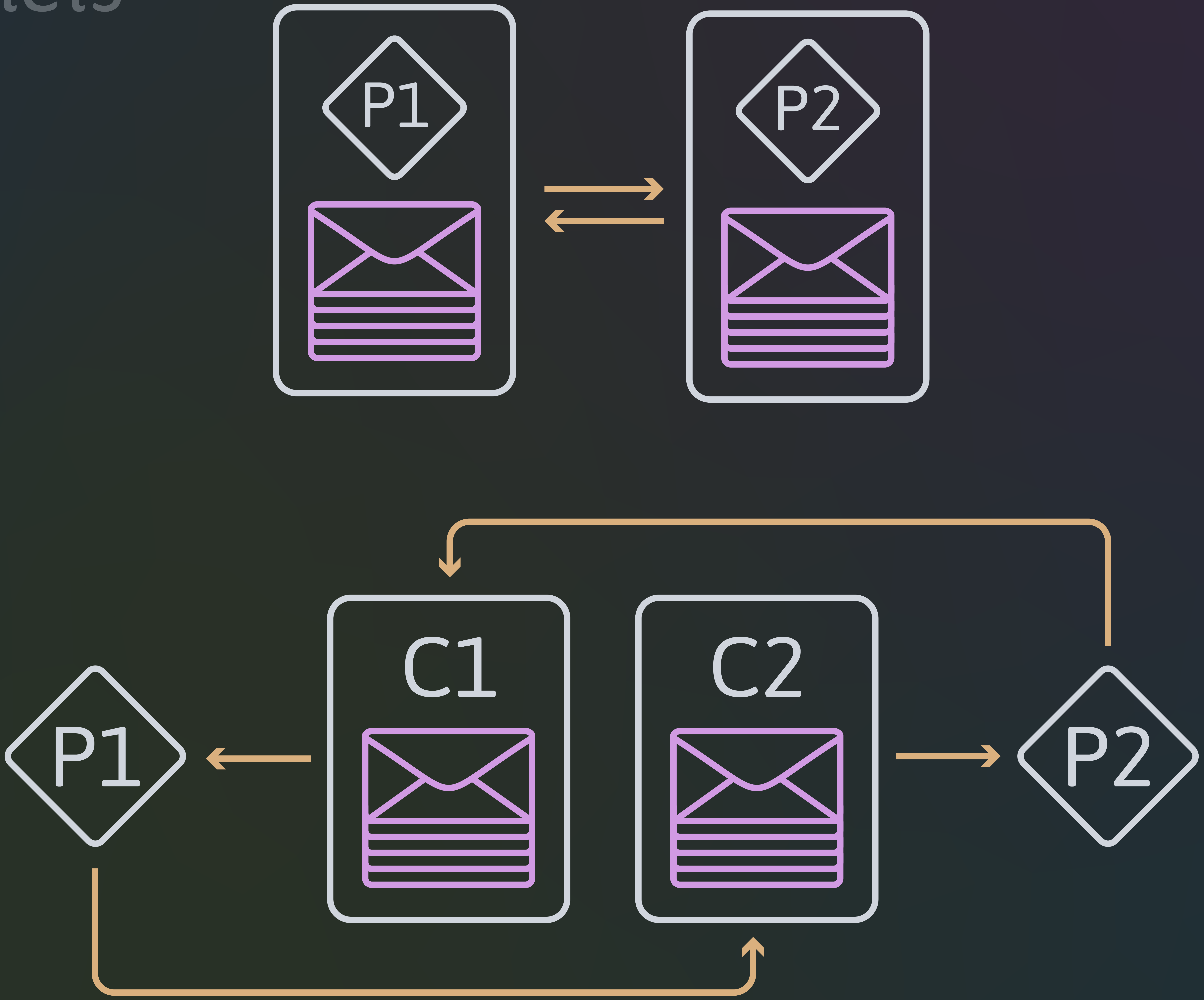


Actors to channels

Actors

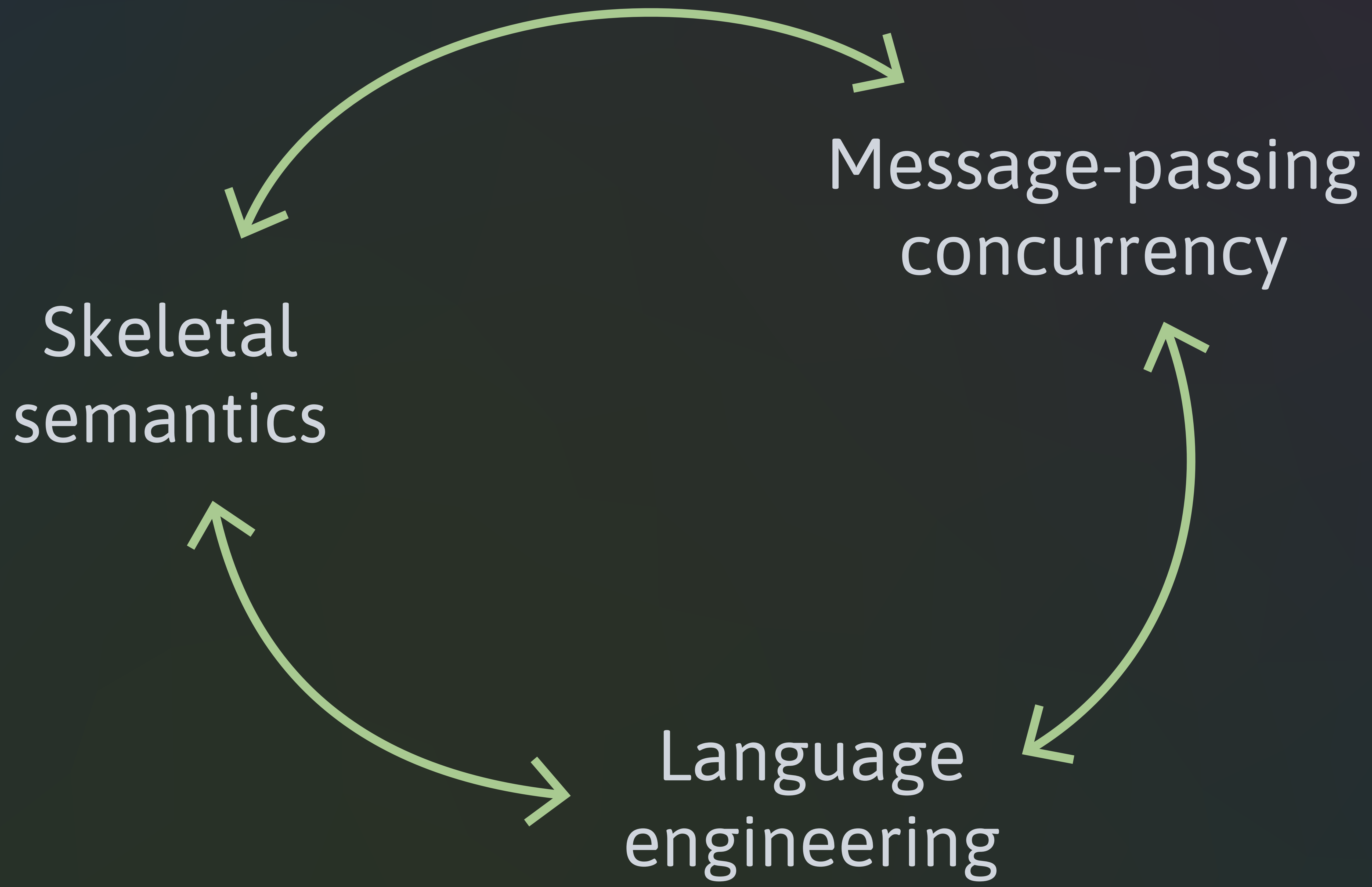
↓

Channels

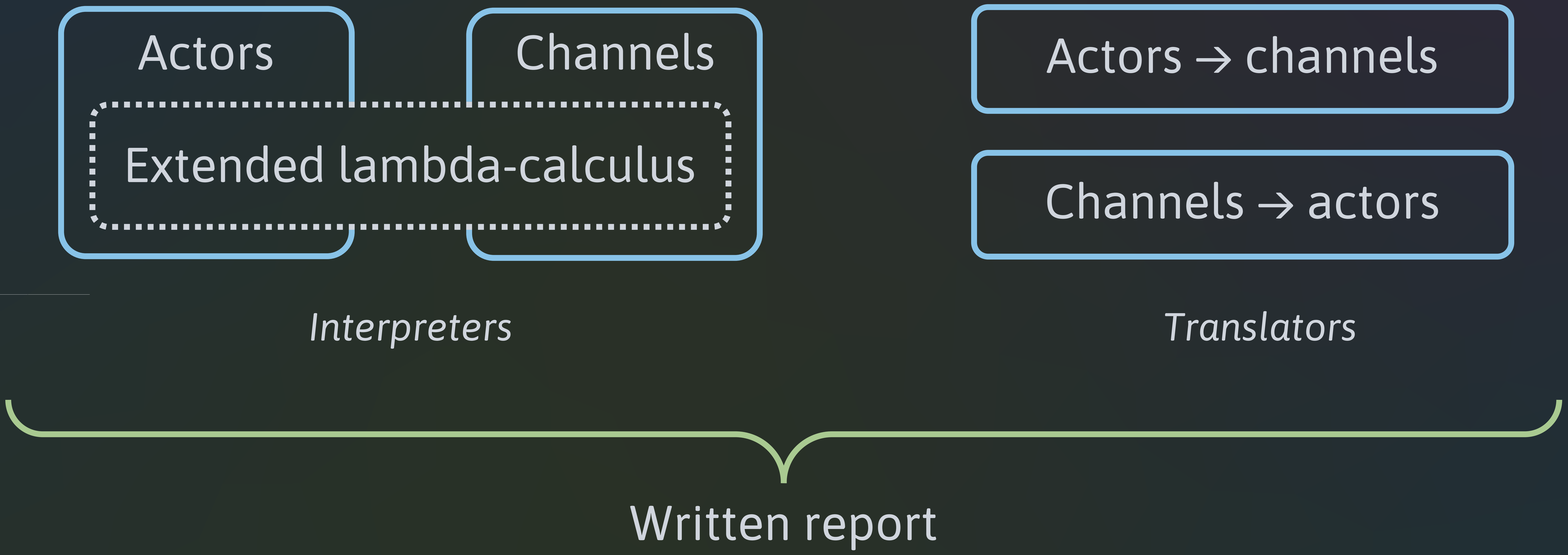


The project

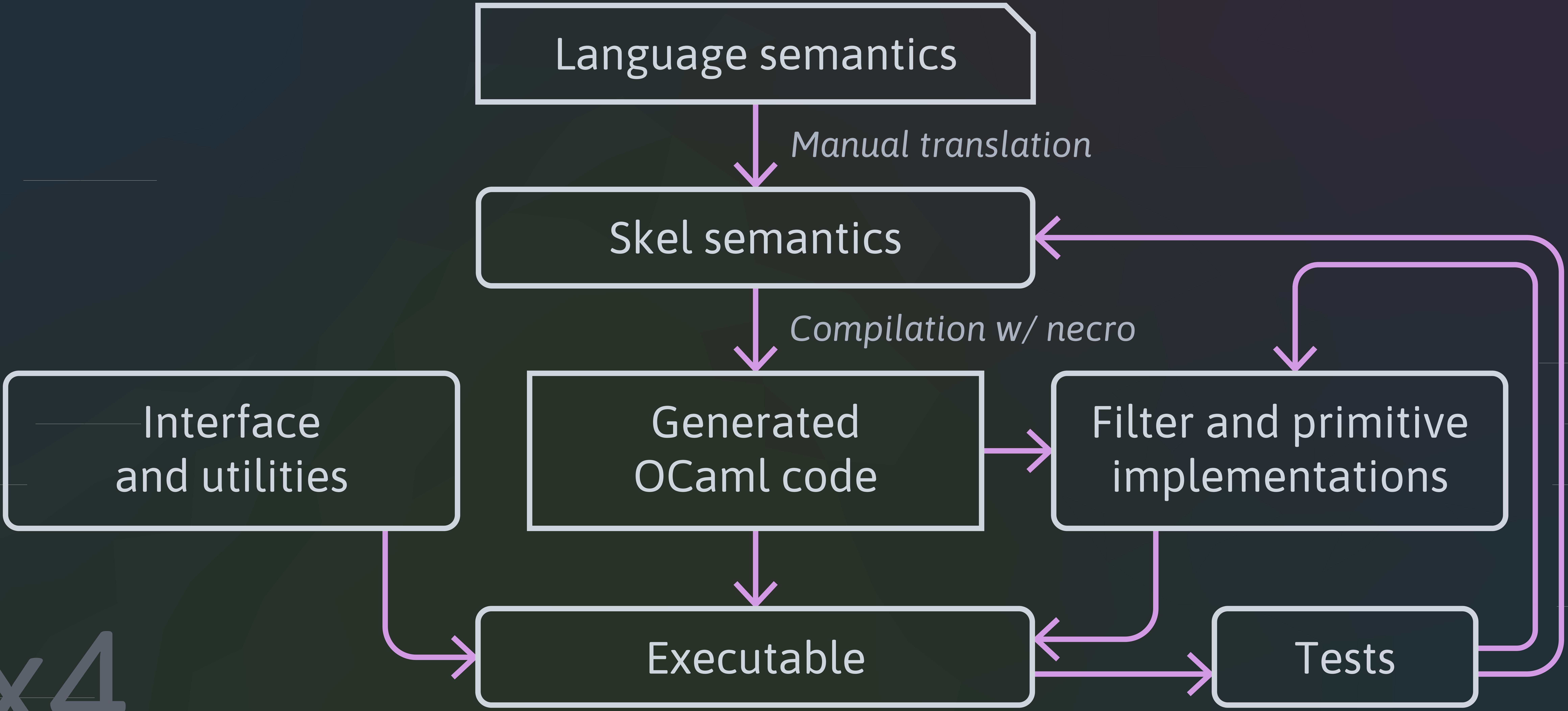
Goals


Message-passing concurrency
Skeletal semantics
Language engineering

Deliverables



Workflow


Language semantics
Manual translation
Skel semantics
Compilation w/ necro
Interface and utilities
Generated OCaml code
Filter and primitive implementations
Executable
Tests

x4

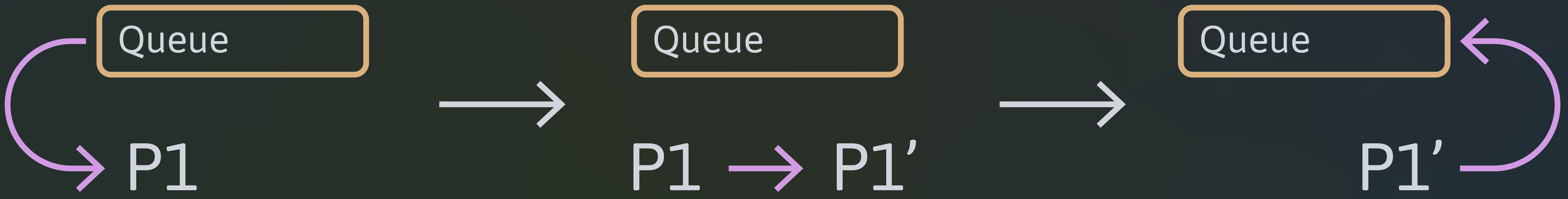
Results

Results

$$P1 \parallel P2 \equiv P2 \parallel P1$$

$$(P1 \parallel P2) \parallel P3 \equiv P1 \parallel (P2 \parallel P3)$$

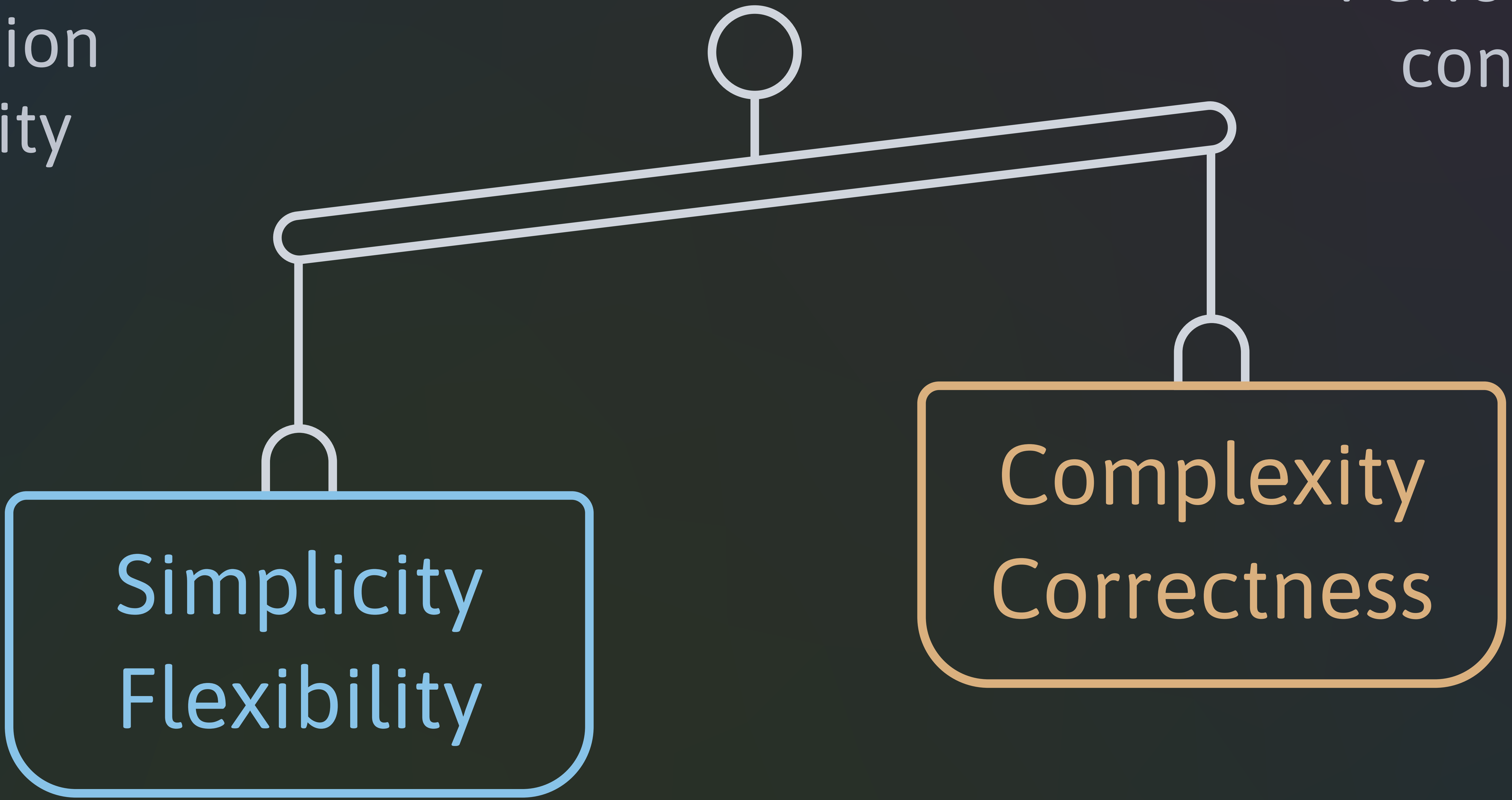
$$\frac{P1 \rightarrow P1'}{P1 \parallel P2 \rightarrow P1' \parallel P2}$$



Results

Specification complexity

Performance concerns



Target platform

Host langauge

Architecture requirements