Type Theory for Mobility and Locality

Thesis Proposal

Jonathan Moody

Committee: Frank Pfennning, Karl Crary, Jeannette Wing, Andrew Gordon (MSR)

School of Computer Science
Carnegie Mellon University
Introduction

- Distributed computation — programming at more than one location.

- Are the locations distinguishable?
Introduction

Distributed computation — programming at more than one location.

Are the locations distinguishable?

No — We can safely ignore locations, we’re done...
Introduction

- Distributed computation — programming at more than one location.

- Are the locations distinguishable?
  
  **No** — We can safely ignore locations, we’re done...
  
  **Yes** — Must be careful when programs or values move.
Introduction

Benefits of being **location-aware**:
- account for localized code or values.
- reflect trust or administration boundaries.
- permit/deny some interactions between locations.
- reflect costs of remote access (bandwidth/latency).

A **location-aware** type theory:
- Specify and statically check properties defined in terms of location....
Introduction

- **Mobility** and **locality** as aspects of location:
  - **Mobility** — “is it location-independent?”
Introduction

- **Mobility** and **locality** as aspects of location:
  - **Mobility** — “is it location-independent?”
  - **Locality** — “is it here? or there?”. 
Thesis statement

“Modal logic can be understood as a type theory defining **mobility** and **locality**; This has practical applications to distributed programming.”
Thesis statement

“Modal logic can be understood as a type theory defining mobility and locality; This has practical applications to distributed programming.”

Proposed contributions:

- Relate modal logic to distributed computation.
Thesis statement

“Modal logic can be understood as a type theory defining **mobility** and **locality**; This has practical applications to distributed programming.”

- Proposed contributions:
  - Relate modal logic to distributed computation.
  - Core calculus with **mobility** and **locality** types.
Thesis statement

“Modal logic can be understood as a type theory defining mobility and locality; This has practical applications to distributed programming.”

Proposed contributions:

- Relate modal logic to distributed computation.
- Core calculus with mobility and locality types.
- Extensions that interact with mobility/locality.
Thesis statement

“Modal logic can be understood as a type theory defining **mobility** and **locality**; This has practical applications to distributed programming.”

- Proposed contributions:
  - Relate modal logic to distributed computation.
  - Core calculus with **mobility** and **locality** types.
  - Extensions that interact with mobility/locality.
  - Apply to distributed grid programming.
Propositions as types

- Functional language **typing rules** are often **logical**:

  “proof $\mathcal{P}$ that $A$ is true” $\iff$ “term $M$ has type $A$”

  \[
  \frac{P}{\Gamma \vdash A} \quad \iff \quad \Gamma^* \vdash_t M : A
  \]

  Proposition $\iff$ Type
  Proof $\iff$ Program
  Pf. Normalization $\iff$ Evaluation
Propositions as types

<table>
<thead>
<tr>
<th>Natural Deduction</th>
<th>Term Typing</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma, A \vdash B ) \Rightarrow ( \Gamma \vdash A \rightarrow B \rightarrow I )</td>
<td>( \Gamma^* \vdash_t A \vdash_t B \rightarrow \Gamma^* \vdash_t A \rightarrow B \rightarrow I )</td>
</tr>
<tr>
<td>( \Gamma \vdash A \rightarrow B ) \Rightarrow ( \Gamma \vdash A \rightarrow B \rightarrow B \rightarrow E )</td>
<td>( \Gamma^* \vdash_t A \rightarrow B \rightarrow B \rightarrow A \rightarrow E )</td>
</tr>
</tbody>
</table>
## Propositions as types

<table>
<thead>
<tr>
<th>Natural Deduction</th>
<th>Term Typing</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma, A \vdash B )</td>
<td>( \Gamma^*, x : A \vdash_t M : B )</td>
</tr>
<tr>
<td>( \Gamma \vdash A \rightarrow B )</td>
<td>( \Gamma^* \vdash_t \lambda x : A . M : A \rightarrow B )</td>
</tr>
<tr>
<td>( \rightarrow I )</td>
<td>( \rightarrow I )</td>
</tr>
<tr>
<td>( \Gamma \vdash A \rightarrow B )</td>
<td>( \Gamma^* \vdash_t M : A \rightarrow B )</td>
</tr>
<tr>
<td>( \rightarrow E )</td>
<td>( \Gamma^* \vdash_t N : A )</td>
</tr>
<tr>
<td>( \Gamma \vdash B )</td>
<td>( \Gamma^* \vdash_t MN : B )</td>
</tr>
<tr>
<td>( \rightarrow E )</td>
<td>( \rightarrow E )</td>
</tr>
</tbody>
</table>
Consequences

- Simplicity: the minimal (logically) complete calculus.
Consequences

- Simplicity: the minimal (logically) complete calculus.

- Some properties.behaviors are not captured:
  - Concurrency (permitted, but not described by types).
  - Complex, 2-way communication patterns.
Consequences

- Simplicity: the minimal (logically) complete calculus.

- Some properties/behaviors are **not captured**:  
  - Concurrency (permitted, but not described by types).
  - Complex, 2-way communication patterns.

- Features that may introduce deadlock, interference, non-determinism are **absent** in the minimal core.
Consequences

Generality: new modal types defined orthogonally.
Consequences

Generality: new modal types defined orthogonally.

Consider **mobility** and **locality** in a familiar framework:

- products (\(\times\)), sums (\(\oplus\)), etc...
- polymorphism (\(\forall \alpha\)), abstract types (\(\exists \alpha\))
- refinement types, intersection (\(\wedge\)), union (\(\vee\)), dependent types \(\Pi i, \Sigma i\)
- information flow, resource bounds, correctness specifications (if type system sufficiently powerful)
Related work

Constructive modal logic:

“Judgmental Reconstruction of Modal Logic”
(Pfenning, Davies ’01)

“Proof Theory and Semantics of I.M.L.”
(Simpson ’94)

Parallel efforts: modal logic \(\implies\) distributed calculus.

“Modal Proofs As Distributed Programs”
(Jia, Walker ’03)

ongoing at CMU... (Crary, Murphy, et al.)

“Constructive Logic for Services and Info. Flow...”
(Borghuis, Feijs ’00)
Related work (contd.)

- Process Calculi (some which **model locations**):
  - Mobile Ambients & Ambient Logic: (various) (Cardelli, Caires, Ghelli, Gordon ’98-’02)
  - DPI: “Resource Access Control...” (Hennessy, *et al.* ’02)
  - Klaim: “Types for Access Control” (De Nicola, Ferrari, *et al.* ’00)
Outline

- Introduction and methodology.
- **Concepts of modal magic.**
- Core modal calculus.
- Properties of extensions.
- Proposed work.
Concepts of modal logic

- Modal logics distinguish **modes** or degrees of truth.
- We have **worlds** related by **accessibility**.

![Diagram of worlds related by accessibility]
Concepts of modal logic

- Modal logics distinguish **modes** or degrees of truth.
- We have **worlds** related by **accessibility**.

![Diagram]

Judgments for the **modes** of truth:
- \( A \text{ true} \) — true at **this** world.
- \( B \text{ valid} \) — true at **all** accessible world(s).
- \( C \text{ possible} \) — true at **some** accessible world.
Concepts of modal logic

Modal logics distinguish **modes** or degrees of truth.

We have **worlds** related by **accessibility**.

Judgments for the **modes** of truth:

- **A true** — true at **this** world.
- **B valid** — true at **all** accessible world(s).
- **C possible** — true at **some** accessible world.
Concepts of modal logic

Modal logics distinguish **modes** or degrees of truth.

We have **worlds** related by **accessibility**.

Judgments for the **modes** of truth:

- **$A$ true** — true at this world.
- **$B$ valid** — true at all accessible world(s).
- **$C$ possible** — true at some accessible world.
Concepts of modal logic

- Modal logics distinguish **modes** or degrees of truth.
- We have **worlds** related by **accessibility**.

Judgments for the **modes** of truth:

- **A true** — true at this world.
- **B valid** — true at all accessible world(s).
- **C possible** — true at some accessible world.
Modal logics distinguish **modes** or degrees of truth.

We have **worlds** related by **accessibility**.

![Diagram showing accessibility between worlds](image)

**Judgments for the modes of truth:**

- \(A \text{ true}\) — true at this world.
- \(B \text{ valid}\) — true at all accessible world(s).
- \(C \text{ possible}\) — true at some accessible world.
Concepts of modal logic

- Modal logics distinguish **modes** or degrees of truth.
- We have **worlds** related by **accessibility**.

![Diagram of modal logic concepts]

Judgments for the **modes** of truth:

- \( A \text{ true} \) — true at **this** world.
- \( B \text{ valid} \) — true at **all** accessible world(s).
- \( C \text{ possible} \) — true at **some** accessible world.
Concepts of modal logic

- Modal logics distinguish **modes** or degrees of truth.
- We have **worlds** related by **accessibility**.

![Diagram]

Judgments for the **modes** of truth:

- **A true** — true at this world.
- **B valid** — true at all accessible world(s).
- **C possible** — true at some accessible world.
Hypothetical judgments

- \( \Delta \rightarrow \text{“global” assumptions } A \text{ valid} \)
- \( \Gamma \rightarrow \text{“local” assumptions } A \text{ true} \)

\[ \Delta \vdash A \text{ valid} \quad \Delta; \Gamma \vdash A \text{ true} \quad \Delta; \Gamma \vdash A \text{ possible} \]
Hypothetical judgments

- $\Delta$ — “global” assumptions $A$ valid
- $\Gamma$ — “local” assumptions $A$ true

$\Delta \vdash A$ valid $\quad \Delta; \Gamma \vdash A$ true $\quad \Delta; \Gamma \vdash A$ possible

$\Delta \vdash A$ valid $\equiv \Delta; \cdot \vdash A$ true
Modal propositions

\( \square A \rightarrow \text{"necessarily } A\)"

\[
\begin{align*}
\text{hypothesis}^* & \quad \frac{\Delta; A \text{ valid}; \Gamma \vdash A \text{ true}}{\Delta, \Gamma \vdash \square A \text{ true}} \quad \frac{\Delta; \vdash A \text{ true}}{\Delta; \Gamma \vdash \square A \text{ true}} \quad \square I \\
\text{modus ponens} & \quad \frac{\Delta; \Gamma \vdash \square A \text{ true}}{\Delta, A \text{ valid}; \Gamma \vdash C \text{ true}} \quad \frac{\Delta, A \text{ valid}; \Gamma \vdash C \text{ true}}{\Delta; \Gamma \vdash C \text{ true}} \quad \square E
\end{align*}
\]
Modal propositions

\( \square A \) — “necessarily \( A \)”

\[
\begin{align*}
\text{\( \Delta, A \text{ valid; } \Gamma \vdash A \text{ true} \)} & \quad \text{\( \text{hyp}^* \)} & \quad \Delta; \Gamma \vdash A \text{ true} & \quad \square I \\
\end{align*}
\]

\[
\begin{align*}
\Delta; \Gamma \vdash \square A \text{ true} & \quad \Delta, A \text{ valid; } \Gamma \vdash C \text{ true} \\
\quad & \quad \Delta; \Gamma \vdash C \text{ true} & \quad \square E
\end{align*}
\]

\( \Diamond A \) — “possibly \( A \)”

\[
\begin{align*}
\Delta; \Gamma \vdash A \text{ true} & \quad \text{poss} & \quad \Delta; \Gamma \vdash A \text{ possible} & \quad \Diamond I \\
\Delta; \Gamma \vdash \Diamond A \text{ true} & \quad \Delta; \Gamma \vdash A \text{ true} & \quad \Diamond E
\end{align*}
\]

\[
\begin{align*}
\Delta; \Gamma \vdash \Diamond A \text{ true} & \quad \Delta; A \text{ true} \vdash C \text{ possible} \\
\quad & \quad \Delta; \Gamma \vdash C \text{ possible} & \quad \Diamond E
\end{align*}
\]
Outline

- Introduction and methodology.
- Concepts of modal logic.
- **Core modal calculus.**
- Properties of extensions.
- Proposed work.
Towards a distributed calculus

Judgements

<table>
<thead>
<tr>
<th>Logical</th>
<th>Typing</th>
<th>Operational</th>
</tr>
</thead>
<tbody>
<tr>
<td>[\Delta; \Gamma \vdash A \text{ true}]</td>
<td>[\Delta^<em>; \Gamma^</em> \vdash M : A]</td>
<td>“evaluate to (A) locally”</td>
</tr>
<tr>
<td>[\Delta; \Gamma \vdash A \text{ possible}]</td>
<td>[\Delta^<em>; \Gamma^</em> \vdash E : A]</td>
<td>“produce (A) somewhere”</td>
</tr>
</tbody>
</table>

Propositions

<table>
<thead>
<tr>
<th>Prop/Type</th>
<th>Logical Reading</th>
<th>Type Reading</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\square A)</td>
<td>“necessarily (A)”</td>
<td>“mobile (A)”</td>
</tr>
<tr>
<td>(\Diamond A)</td>
<td>“possibly (A)”</td>
<td>“remote (A)”</td>
</tr>
</tbody>
</table>
Typing: □A

\[ \Delta, u :: A; \Gamma \vdash u : A \quad \text{hypothesis} \quad \frac{\Delta; \Gamma \vdash M : A}{\Delta; \Gamma \vdash \text{box} M : \square A} \quad \text{□I} \]

\[ \frac{\Delta; \Gamma \vdash M : \square A \quad \Delta, u :: A; \Gamma \vdash N : B}{\Delta; \Gamma \vdash \text{let box u = M in N : B}} \quad \text{□E} \]
Intuition

```ocaml
let box u = 
    box M 
  in N
```

Formally

```ocaml
let box = box in
    let box = box in
```

Opportunistic concurrent evaluation — not essential to logical necessity.
Operational: □A

Intuition

\[
\text{let box } u = \\
\quad \text{box } M \\
\quad \text{in } N
\]

Formally

\[
\text{let box } = \\
\quad \text{box } = \\
\quad \text{Opportunistic concurrent evaluation — not essential to logical necessity.}
\]
Operational: □A

Intuition

N
... u ...

V
Operational: □ A

- **Intuition**

![Diagram](image)

- **Formally**

\[
\langle r_1 : \mathcal{R}[\text{let box } u = \text{box } M \text{ in } N] \rangle \implies \langle r_2 : M \rangle, \ \langle r_1 : \mathcal{R}[r_2/u]N \rangle
\]

\[
\langle r_2 : V \rangle, \ \langle r_1 : \mathcal{R}[r_2] \rangle \implies \langle r_2 : V \rangle, \ \langle r_1 : \mathcal{R}[V] \rangle
\]

\[
\langle r_1 : \mathcal{R}[\text{let box } u = \text{box } V \text{ in } N] \rangle \implies \langle r_1 : \mathcal{R}[\mathcal{R}[V/u]N] \rangle
\]

- **Opportunistic** concurrent evaluation — **not essential** to logical necessity.
Example: Higher-order mobility

(* times\_k : □nat -> □(nat->nat) *)
fun times\_k k =
  let
    box u = k
  in
    box (\x:nat . x * u)

 Lexically-scoped mobile closures
 capture mobile bindings (u ∈ Δ).
Example: Higher-order mobility

(* pmap : □(nat->nat) -> list □nat
    -> list □nat *)

fun pmap f [] = []
pmap f (x::tl) =
    let box f' = f
    box x' = x
    box v = box (f' x') (* spawn work *)
in
    ((box v)::(pmap f tl))

(* double_lst : list □nat -> list □nat *)
val double_lst = pmap (times_k (box 2))

Clean account of mobility at function types □(A → B).
Example: Divide & conquer

(* fib :: □nat -> nat *)

`mfun fib x =`

`let box n = x in`

`if (n < 2) then n else`

`let`

`(* spawn a,b concurrently *)`

`box a = box (fib box(n-1))`

`box b = box (fib box(n-2))`

`in (a + b)`

Note: `mfun` defines a mobile recursive function.
Typing: ▷ $A$

$$
\frac{\Delta; \Gamma \vdash M : A \quad \text{poss}}{\Delta; \Gamma \vdash \{M\} : A} \quad \frac{\Delta; \Gamma \vdash E : A}{\Delta; \Gamma \vdash \text{dia} E : \Diamond A} \quad \Diamond I
$$

$$
\frac{\Delta; \Gamma \vdash M : \Diamond A \quad \Delta; x : A \vdash F : B}{\Delta; \Gamma \vdash \text{let} \, \text{dia} x = M \in F : B} \quad \Diamond E
$$
Operational: ♠A

Intuition

```
let dia x =
  dia V
in F
```
Operational: $\downarrow A$

Intuition

```
let dia(x =
  dia_1
in F) \{x, F\}
```

Formally

```
let dia = dia_1 \{\}
```

Type Theory for Mobility and Locality – p.24/49
Operational: $\Diamond \lambda$

Intuition

$[V/x] \ F$
Operational: \(\Diamond A\)

- **Intuition**

\[
[V/x] F
\]

- **Formally**

\[
\begin{align*}
\langle l_1 : S[ \text{let} \ dia x = \text{dia} l_2 \text{ in } F] \rangle, & \quad \langle l_2 : \{ V \} \rangle \\
\implies & \quad \langle l_1 : S[l'_2] \rangle, \quad \langle l'_2 : [V/x] F \rangle, \quad \langle l_2 : \{ V \} \rangle \\
\langle l_1 : S[ \text{let} \ dia x = \text{dia} \{ V \} \text{ in } F] \rangle & \quad \implies \quad \langle l_1 : S[[V/x] F] \rangle
\end{align*}
\]
Example: A remote queue

(* rqueue : \{\{insert:nat->unit, ...\}\} *)
val rqueue = bind_queue ...

(* insert (x : □nat) into rqueue *)
let
  box v = x
  dia q = rqueue (* jump to queue *)
in
  let val _ = q.insert v (* v mobile *)
in ...

.Requires a mobile value (□nat) because queue is remote.
Core calculus — summary

Type $A, B ::= A \rightarrow B \mid \square A \mid \Diamond A$

Term $M, N ::= x \mid u$

$\mid \lambda x : A . M \mid M \ N$

$\mid \text{box } M \mid \text{dia } E$

$\mid \text{let } \text{box } u = M \text{ in } N$

Expr. $E, F ::= \{ M \} \mid \text{let } \text{box } u = M \text{ in } F$

$\mid \text{let } \text{dia } x = M \text{ in } F$
Core calculus — summary

Type $A, B ::= A \rightarrow B \mid \square A \mid \Diamond A$

Term $M, N ::= r \mid x \mid u$
$\mid \lambda x : A . M \mid M N$
$\mid \text{box } M \mid \text{dia } E$
$\mid \text{let } \text{box } u = M \text{ in } N$

Expr. $E, F ::= l \mid \{ M \} \mid \text{let } \text{box } u = M \text{ in } F$
$\mid \text{let } \text{dia } x = M \text{ in } F$

Label $w ::= r \mid l$

Process $\pi ::= \langle r : M \rangle \mid \langle l : E \rangle$

Config. $C ::= \cdot \mid C, \pi$
Process configurations

\[ C \rightarrow C' \quad \text{— “}C\text{” steps to } C'\text{”}.

- Non-deterministic choice of process (concurrency).
- Synchronization rule either lazy or strict (don’t care).
Process configurations

\[ C \Rightarrow C' \] — “\( C \) steps to \( C' \)”.

- Non-deterministic choice of process (concurrency).
- Synchronization rule either lazy or strict (don’t care).

\[ \psi \vdash^c C : \Lambda \] — “conf. \( C \) has type \( \Lambda \) (under \( \psi \))”.

- \( \Lambda = r_1 :: A, \ldots, l_2 \vdash A, \ldots \)
- \( \psi \) — determines scope/accessibility of labels.
Properties

- Type preservation:

  \[ \psi \vdash^c C : \Lambda \quad \text{and} \quad C \rightarrow C' \]

  \[ \Rightarrow \quad \exists \Lambda' \supseteq \Lambda \cdot \exists \psi' \cdot \psi' \vdash^c C' : \Lambda' \]

- \( \Lambda' \) and \( \psi' \) grow as processes are created.
Properties

- **Type preservation:**
  \[ \psi \vdash^c C : \Lambda \quad \text{and} \quad C \rightarrow C' \]
  \[ \Rightarrow \quad \exists \Lambda' \supseteq \Lambda \cdot \exists \psi' . \psi' \vdash^c C' : \Lambda' \]

- \( \Lambda' \) and \( \psi' \) grow as processes are created.

- **Progress:**
  \[ \psi \vdash^c C : \Lambda \quad \text{and} \quad \psi \text{ noncyclic} \]
  \[ \Rightarrow \quad \exists C' . C \rightarrow C' \quad \text{or} \quad C \text{ terminal} \]

- \( \psi \text{ noncyclic} \) — permits inductive argument.
- **deadlocked:** \( \langle r_1 : r_2 (\lambda x : A . x) \rangle, \langle r_2 : r_1 (\lambda x : A . x) \rangle \)
Properties

- **Termination**: sequences $C_1 \rightarrow C_2 \rightarrow \ldots$ halt
  - core calculus (without fixpoints).
  - $\psi |-^c C_1 : \Lambda_1$ — well-formed configuration.
  - $\psi$ noncyclic — no recursion through “backdoor”.
Properties

- **Termination**: sequences $C_1 \rightarrow C_2 \rightarrow \ldots$ halt
  - core calculus (without fixpoints).
  - $\psi \vdash^c C_1 : \Lambda_1$ — well-formed configuration.
  - $\psi$ noncyclic — no recursion through “backdoor”.

- **Confluence** holds for well-formed config:
  - under same general conditions as above...
  - modulo $(C \equiv D)$ synchronization-equiv.
Outline

- Introduction and methodology.
- Concepts of modal logic.
- Core modal calculus.
- **Properties of extensions.**
- Proposed work.
Example: marshalling

(* marshall_nat :: nat -> □nat *)
fun marshall_nat n =
  case n of
    zero => box zero (* boxed val *)
  | succ(x) =>
    let
      box u = marshall_nat x
    in
      box (succ(u)) (* boxed val *)

Spawning a concurrent process is optional.
Example: marshalling

- PROHIBITED: marshalling arbitrary closures.

(*) closure over binding c *)
val c = 42
fun f y = if y > 0 then c else y

(*) marshall_n2n: (nat->nat) -> □(nat->nat) *)
fun marshall_n2n f =
  box f (* ill-typed occurrence *)

- An arbitrary (nat → nat) may capture local binding.
Locality of effects

- Locality and effects are naturally connected.
  - **Observable effects** should execute at **definite** locations.
  - **Machine state** underlying effects is localized.
Locality of effects

- Locality and effects are naturally connected.
- **Observable effects** should execute at **definite** locations.
- **Machine state** underlying effects is localized.

**Nutshell:** add effect monad \((\Box A)\) and **local computations**.

<table>
<thead>
<tr>
<th>Typing</th>
<th>Operational</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Delta; \Gamma \vdash M : A)</td>
<td>“evaluate to (A) locally”</td>
</tr>
<tr>
<td>(\Delta; \Gamma \vdash P \sim A)</td>
<td>“produce (A) locally with effects”</td>
</tr>
<tr>
<td>(\Delta; \Gamma \vdash E \div A)</td>
<td>“produce (A) somewhere with effects”</td>
</tr>
</tbody>
</table>
Example: mutable ref

- PROHIBITED: mobility for mutable references.

  (* counter : ref nat *)
  val counter = ref 0

  (* bump :: unit -> unit *)
  mfun bump () =
    counter := !counter + 1

  box _ = box (bump ()) (* bump *)
  box _ = box (bump ()) (* twice *)
  (* !counter = 0? *)

- Type system (⊙A) disallows effects in spawned terms.

  See proposal document for details...
Outline

- Introduction and methodology.
- Concepts of modal logic.
- Core modal calculus.
- Properties of extensions.
- Proposed work.
Progress

Proposed contributions:

1. Relate modal logic to distributed computation.
2. Core calculus with mobility and locality types.
3. Extensions that interact with mobility/locality.
4. Apply to distributed grid programming.
Progress

Proposed contributions:

✓ Relate modal logic to distributed computation.
Progress

Proposed contributions:

✓ Relate modal logic to distributed computation.
✓ Core calculus with mobility and locality types.
Progress

Proposed contributions:

- Relate modal logic to distributed computation.
- Core calculus with mobility and locality types.
  Extensions that interact with mobility/locality.
Progress

Proposed contributions:

✓ Relate modal logic to distributed computation.
✓ Core calculus with mobility and locality types. Extensions that interact with mobility/locality. Apply to distributed grid programming.
Extensions

Complete

- Concrete data: products, sums, recursive types.
- Effects: effect monad ($\bigcirc A$), mutable refs.
Extensions

- Complete
  - Concrete data: products, sums, recursive types.
  - Effects: effect monad ($\odot A$), mutable refs.

- Proposed: Polymorphism ($\forall$), abstract types ($\exists$).
  - Well-known problem sharing abstract values $\alpha$ between locations.
  - Permutations of $\exists\alpha . B$ and $\square \Diamond$ seem interesting...
Application: ConCert grid

- ConCert runtime for **trustless grid computing**:
  - **Trustless** — execute certified fragments of code.
  - **Grid** — network of peers provide compute cycles.
- Certification is based on type/proof checking, not trust.
Application: ConCert grid

- ConCert runtime for **trustless grid computing**:
  - **Trustless** — execute certified fragments of code.
  - **Grid** — network of peers provide compute cycles.
- Certification is based on type/proof checking, not trust.

- **Mobility** and **locality** matter in ConCert:
  - All the general reasons, plus...
  - Confidentiality, abstraction.
  - Efficiency (space/bandwidth costs).
  - Safety policies (move **only certified code**).
Application: ConCert grid

Prototype compiler **Hemlock**:  
- Programming with `spawn` and `sync` model.  
- Type system: assume all values are mobile.  
- Marshalling  
  - mutable refs — by copying.  
  - code — problematic for local libraries.
Application: ConCert grid

Prototype compiler Hemlock:
- Programming with `spawn` and `sync` model.
- Type system: assume all values are mobile.
- Marshalling
  - mutable refs — by copying.
  - code — problematic for local libraries.

Mobility and locality types provide:
- Statically safe variant of `spawn/sync` ($\square A$).
- Link with local libraries (trusted/certified mix).
- Bind and use remote resources ($\Diamond A$).
Details

- **Hemlock** extensions/modifications:
  - Type system for □, ◊ and effects.
Details

- **Hemlock** extensions/modifications:
  - Type system for □, ◊ and effects.
  - Code generation for new box/dia features.
Details

- **Hemlock** extensions/modifications:
  - Type system for □, ◊ and effects.
  - Code generation for new box/dia features.
  - Marshalling: migrate to format compatible with (□A).
**Details**

**Hemlock** extensions/modifications:
- Type system for □, ◊ and effects.
- Code generation for new box/dia features.
- Marshalling: migrate to format compatible with (□A).

**ConCert** runtime extensions (support □,◊ model):
- Mapping, binding to (◊A) resources.
Details

- **Hemlock** extensions/modifications:
  - Type system for $\square$, $\Diamond$ and effects.
  - Code generation for new box/dia features.
  - Marshalling: migrate to format compatible with ($\square A$).

- **ConCert** runtime extensions (support $\square$, $\Diamond$ model):
  - Mapping, binding to ($\Diamond A$) resources.
  - Targeted “closures” $\{x \mapsto \bullet; E\}$
    (arising from $\text{let } \text{dia } x = M \text{ in } E$)
## Strategy: Tasks and priorities

<table>
<thead>
<tr>
<th>Priority</th>
<th>Effort</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>med</td>
<td>Polymorphism, abstract types</td>
</tr>
<tr>
<td>high</td>
<td>med</td>
<td>Parsing, typechecking (◻, ◊, &amp; effects)</td>
</tr>
<tr>
<td>high</td>
<td>high</td>
<td>Hemlock runtime marshalling code (TALT)</td>
</tr>
<tr>
<td>high</td>
<td>high</td>
<td>TALT code gen. box/dia features</td>
</tr>
</tbody>
</table>
## Strategy: Tasks and priorities

<table>
<thead>
<tr>
<th>Priority</th>
<th>Effort</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>med</td>
<td>Polymorphism, abstract types</td>
</tr>
<tr>
<td>high</td>
<td>med</td>
<td>Parsing, typechecking (☐, ◊, &amp; effects)</td>
</tr>
<tr>
<td>high</td>
<td>high</td>
<td>Hemlock runtime marshalling code (TALT)</td>
</tr>
<tr>
<td>high</td>
<td>high</td>
<td>TALT code gen. box/dia features</td>
</tr>
<tr>
<td>med</td>
<td>med</td>
<td>ConCert runtime support (◊) (ML)</td>
</tr>
<tr>
<td>med</td>
<td>high</td>
<td>Abstract resources (◊∃), Modules?</td>
</tr>
</tbody>
</table>
## Strategy: Tasks and priorities

<table>
<thead>
<tr>
<th>Priority</th>
<th>Effort</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>med</td>
<td>Polymorphism, abstract types</td>
</tr>
<tr>
<td>low</td>
<td>med</td>
<td>Dependent types, policy-related types</td>
</tr>
<tr>
<td>high</td>
<td>med</td>
<td>Parsing, typechecking (□, ◇, &amp; effects)</td>
</tr>
<tr>
<td>high</td>
<td>high</td>
<td>Hemlock runtime marshalling code (TALT)</td>
</tr>
<tr>
<td>high</td>
<td>high</td>
<td>TALT code gen. box/dia features</td>
</tr>
<tr>
<td>med</td>
<td>med</td>
<td>ConCert runtime support (◇) (ML)</td>
</tr>
<tr>
<td>med</td>
<td>high</td>
<td>Abstract resources (◇∃), Modules?</td>
</tr>
</tbody>
</table>
Questions?
## Strategy: Tasks and Priorities

<table>
<thead>
<tr>
<th>Time</th>
<th>Date</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Polymorphism, abstract types</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>Dependent types, policy-related types</td>
</tr>
<tr>
<td>2-4</td>
<td>3-5</td>
<td>Parsing, typechecking ($\square$, $\diamond$, &amp; effects)</td>
</tr>
<tr>
<td>3-4</td>
<td>6-9</td>
<td>Hemlock runtime marshalling code (TALT)</td>
</tr>
<tr>
<td>3-4</td>
<td>9-13</td>
<td>TALT code gen. box/dia features</td>
</tr>
<tr>
<td>2-3</td>
<td>11-16</td>
<td>ConCert runtime support ($\diamond$) (ML)</td>
</tr>
<tr>
<td>2-3</td>
<td>13-19</td>
<td>Abstract resources ($\diamond \exists$), Modules?</td>
</tr>
</tbody>
</table>
Programming Models

- →,nat,*,+,... (local pure functional programs).
- □ (spawn mobile terms).
- □◊ (spawn mobile terms, jump among locations).
- ⊙ (local effects)
- □⊙ (spawn mobile, local effects)
- □□◊ (spawn, local effects, jumping, remote effects)
Recall that ConCert assumes P2P grid with unreliable nodes...

Some characteristics of the S4 formalism:
- No explicit world annotations necessary in calculus.
- Non-trivial $\Box \Diamond A$ values are extra-logical. Programmer can’t create them ($A \not\leftrightarrow \Box \Diamond A$).

This simplifies the runtime support layer:
- Flexible scheduling. Program fragments run anywhere, or nearly anywhere.
- Most locations are “stateless”. Node can leave network after producing result.
Typing Rules

\[
\frac{\Delta; \Gamma, x : A, \Gamma' \vdash_J x : A}{\Delta; \Gamma, x : A, \Gamma' \vdash_J x : A} \quad \text{hyp}
\]

\[
\frac{\Delta; \Gamma, x : A \vdash_J M : B}{\Delta; \Gamma \vdash_J \lambda x : A . M : A \rightarrow B} \rightarrow I
\]

\[
\frac{\Delta, u :: A, \Delta'; \Gamma \vdash_J u : A}{\Delta, u :: A, \Delta'; \Gamma \vdash_J u : A} \quad \text{hyp}^*
\]

\[
\frac{\Delta; \Gamma \vdash_J M : A \rightarrow B \quad \Delta; \Gamma \vdash_J N : A}{\Delta; \Gamma \vdash_J M N : B} \rightarrow E
\]

\[
\frac{\Delta; \cdot \vdash_J M : A}{\Delta; \Gamma \vdash_J \text{box } M : \square A} \quad \square I
\]

\[
\frac{\Delta; \Gamma \vdash_J M : A}{\Delta; \Gamma \vdash_J \{M\} \div A} \quad \text{poss}
\]

\[
\frac{\Delta; \Gamma \vdash_J M : \diamond A \quad \Delta; x : A \vdash_J F \div B}{\Delta; \Gamma \vdash_J \text{let } \text{dia } x = M \text{ in } F \div B} \quad \diamond E
\]

\[
\frac{\Delta; \Gamma \vdash_J E \div A}{\Delta; \Gamma \vdash_J \text{dia } E : \diamond A} \quad \diamond I
\]

\[
\frac{\Delta; \Gamma \vdash_J M : \square A \quad \Delta, u :: A; \Gamma \vdash_J F \div B}{\Delta; \Gamma \vdash_J \text{let } \text{box } u = M \text{ in } F \div B} \quad \square E_p
\]
Extensions

The type theory of $\rightarrow\square\lozenge$ is easily extensible:

- products $(A \times B)$, sums $(A + B)$, recursive types $(\mu\alpha \cdot B)$
  ...
  (straightforward)

\[
\begin{align*}
\Delta; \Gamma \vdash M : A & \quad \Delta; \Gamma \vdash N : B \\
\quad \frac{}{\Delta; \Gamma \vdash (M, N) : A \times B} & \\
\Delta; \Gamma \vdash M : A \times B & \\
\quad \frac{}{\Delta; \Gamma \vdash \text{fst} \, M : B} & \\
\Delta; \Gamma \vdash M : A \times B & \\
\quad \frac{}{\Delta; \Gamma \vdash \text{snd} \, M : A}
\end{align*}
\]
The type theory of \((\to \Box \diamond)\) is easily extensible:

- Products \((A \times B)\), sums \((A + B)\), recursive types \((\mu \alpha . B)\)
  ... (straightforward)

\[
\begin{align*}
\Delta; \Gamma \vdash M : A & \quad \Delta; \Gamma \vdash N : B \\
\hline \\
\Delta; \Gamma \vdash (M, N) : A \times B
\end{align*}
\]

- **Fixpoints:** \(\text{fixv}(u :: A) . M\) and \(\text{fix}(x : A) . M\)

\[
\begin{align*}
\Delta, u :: A; \cdot \vdash M : A & \\
\hline \\
\Delta; \Gamma \vdash \text{fixv}(u :: A) . M : A
\end{align*}
\]

\[
\begin{align*}
\Delta; \Gamma \vdash M : A & \\
\hline \\
\Delta; \Gamma \vdash \text{fix}(u : A) . M : A
\end{align*}
\]
Extensions

Each extension interacts with **mobility** and **locality**.
Extensions

- Each extension interacts with **mobility** and **locality**.

- A heuristic...
  - Location-**neutral** $\implies$ term $M : A$
  - Location-**dependent** $\implies$ expression $E \div A$
  (actually or potentially)
Extensions

- Each extension interacts with mobility and locality.

- A heuristic...
  - Location-neutral \( \Rightarrow \) term \( M : A \)
  - Location-dependent \( \Rightarrow \) expression \( E \div A \)
    (actually or potentially)

- \( (A \to \square A) \rightarrow A \) can be made mobile (marshalling)
  - Some values \( A \) are location-independent.
  - Others not!
Effect Typing

\[
\begin{align*}
\Delta; \Gamma \vdash_J P \leadsto A & \quad \Rightarrow \Delta; \Gamma \vdash_J \text{comp } P : \square A & \quad \text{I} \\
\Delta; \Gamma \vdash_J M : A & \quad \Rightarrow \Delta; \Gamma \vdash_J [M] \leadsto A & \quad \text{comp} \\
\Delta; \Gamma \vdash_J P \leadsto A & \quad \Rightarrow \Delta; \Gamma \vdash_J \{P\} \div A & \quad \text{poss'} \\
\end{align*}
\]

\[
\begin{align*}
\Delta; \Gamma \vdash_J M : \square A & \quad \Rightarrow \Delta; \Gamma \vdash_J \text{let } \text{comp } x = M \text{ in } Q \leadsto B & \quad \text{E} \\
\Delta; \Gamma \vdash_J M : \square A & \quad \Rightarrow \Delta; \Gamma \vdash_J \text{let } \text{box } u = M \text{ in } Q \leadsto B & \quad \text{El} \\
\Delta; \Gamma \vdash_J M : \square A & \quad \Rightarrow \Delta; \Gamma \vdash_J \text{let } \text{comp } x = M \text{ in } F \div B & \quad \text{Ep} \\
\end{align*}
\]

Primitive effects:

\[
\begin{align*}
\Theta = \Theta_1, a^w : A, \Theta_2 & \quad \Rightarrow \Theta; \Delta; \Gamma \vdash_w a^w : \text{ref } A & \quad \text{addr} \\
\Delta; \Gamma \vdash_J M : \text{ref } A & \quad \Rightarrow \Delta; \Gamma \vdash_J \text{ref } M \leadsto \text{ref } A & \quad \text{talloc} \\
\Delta; \Gamma \vdash_J M : \text{ref } A & \quad \Rightarrow \Delta; \Gamma \vdash_J \text{tget } M \leadsto A & \quad \text{tget} \\
\Delta; \Gamma \vdash_J M : \text{ref } A & \quad \Rightarrow \Delta; \Gamma \vdash_J \text{tset } M := N \leadsto 1 & \quad \text{tset} \\
\end{align*}
\]