Type Theory for Mobility and Locality

Thesis Proposal

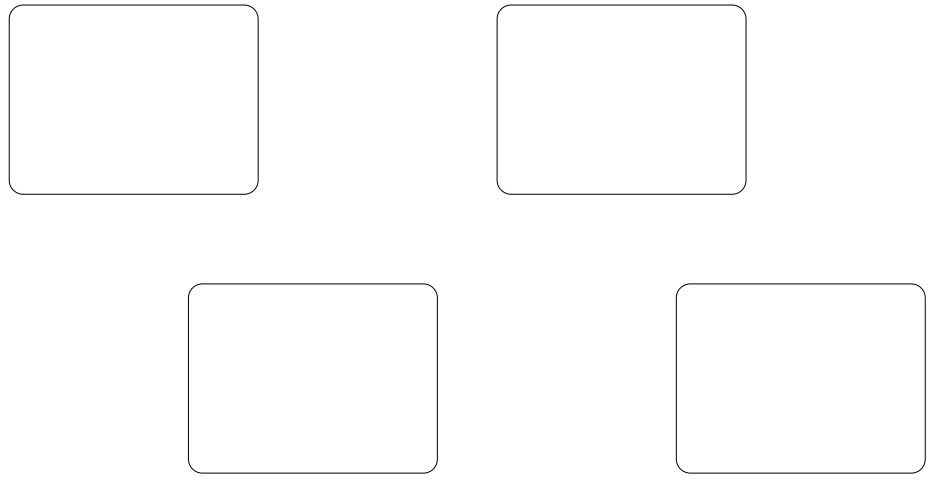
Jonathan Moody

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

School of Computer Science
Carnegie Mellon University

Introduction

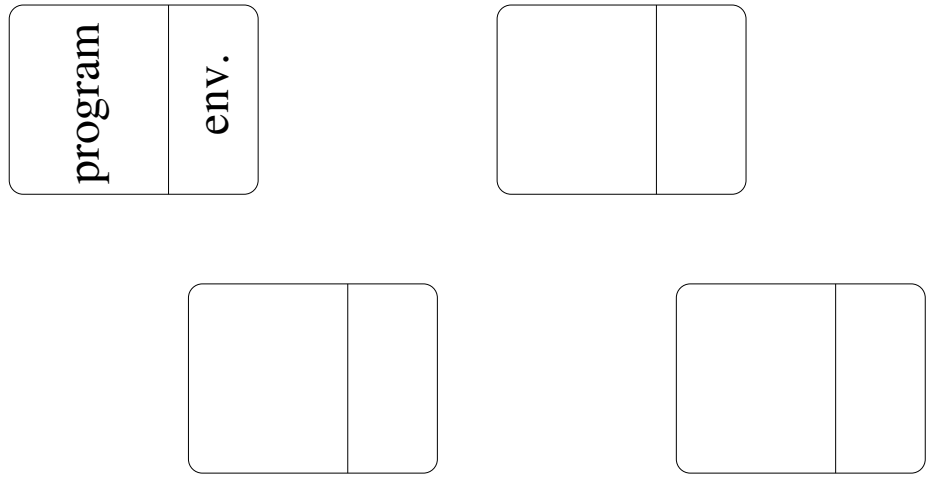
- Distributed computation — programming at more than one **location**.



- Are the locations distinguishable?

Introduction

- Distributed computation — programming at more than one **location**.

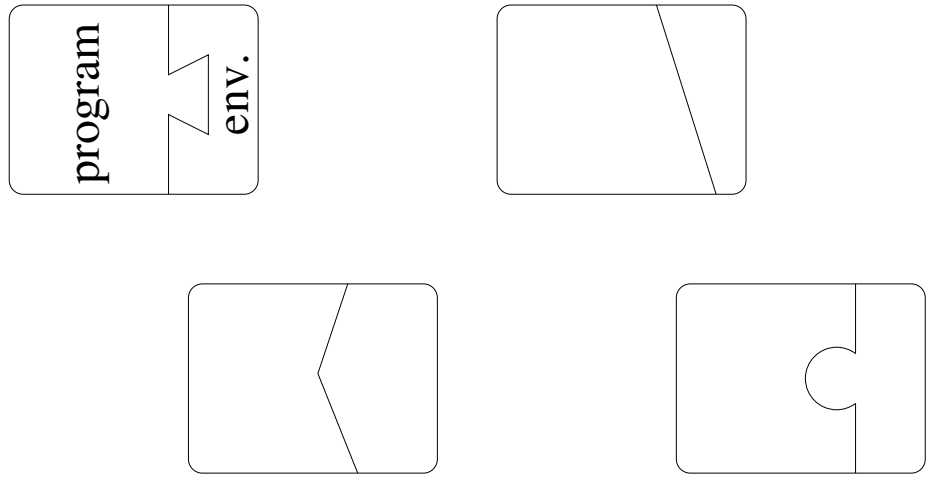


- Are the locations distinguishable?

  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.

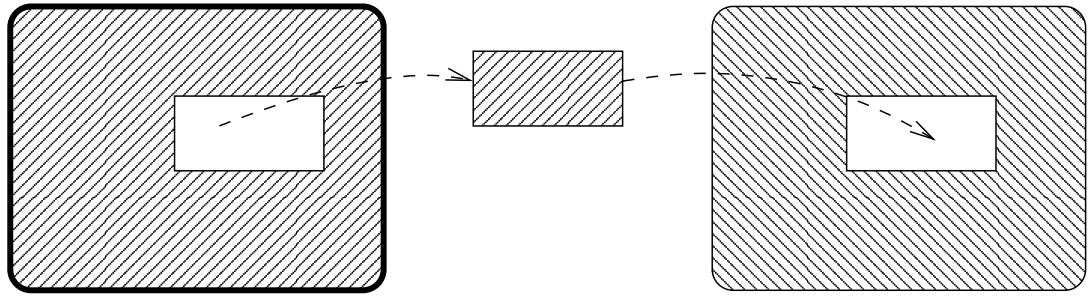
Introduction

- 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....

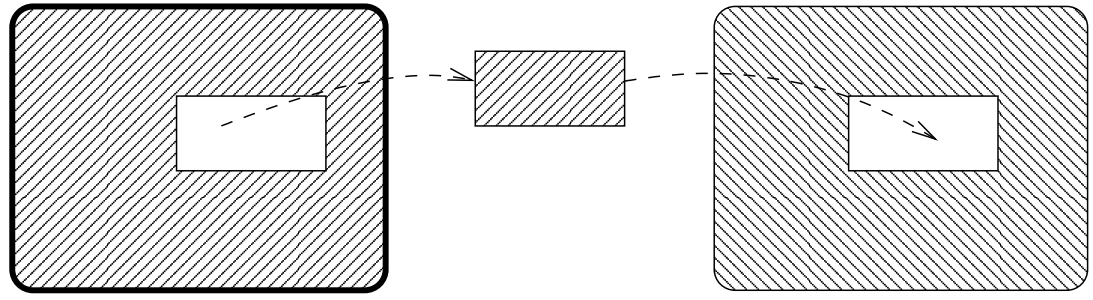
Introduction

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

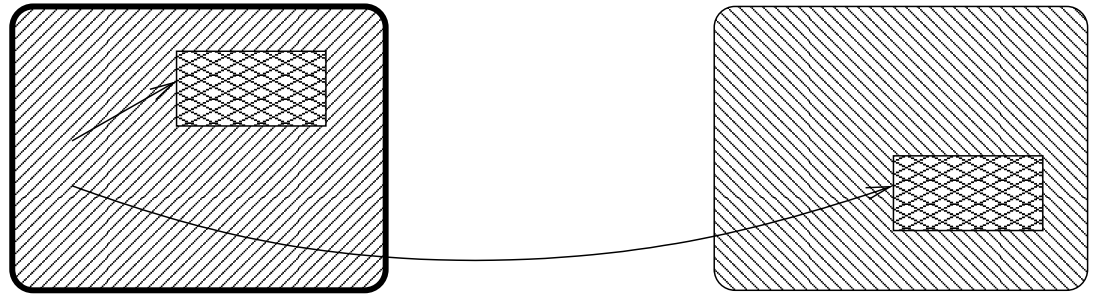


Introduction

- **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" $\Longleftrightarrow$ "term M has type A"

$$\frac{\mathcal{P}}{\Gamma \vdash A} \qquad \Gamma^* \vdash_t M : A$$

Proposition	$\Longleftrightarrow$	Type
Proof	$\Longleftrightarrow$	Program
Pf. Normalization	$\Longleftrightarrow$	Evaluation

Propositions as types

Natural Deduction	Term Typing
$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to I$	$\dfrac{\Gamma^*, \quad A \vdash_t \quad B}{\Gamma^* \vdash_t \qquad\qquad A \to B} \to I$
$\dfrac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B} \to E$	$\dfrac{\Gamma^* \vdash_t \quad A \to B \quad \Gamma^* \vdash_t \quad A}{\Gamma^* \vdash_t \quad B} \to E$

Propositions as types

Natural Deduction	Term Typing
$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to I$	$\dfrac{\Gamma^*, \mathtt{x} : A \vdash_t M : B}{\Gamma^* \vdash_t \lambda \mathtt{x} : A \,.\, M : A \to B} \to I$
$\dfrac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B} \to E$	$\dfrac{\Gamma^* \vdash_t M : A \to B \quad \Gamma^* \vdash_t N : A}{\Gamma^* \vdash_t MN : B} \to E$

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 ($*$), sums ($+$), *etc*...
  - polymorphism ($\forall\alpha$), abstract types ($\exists\alpha$)
  - refinement types, intersection ($\wedge$), union ($\vee$), dependent types Πi, Σi
  - 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 $\Longrightarrow$ 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.)

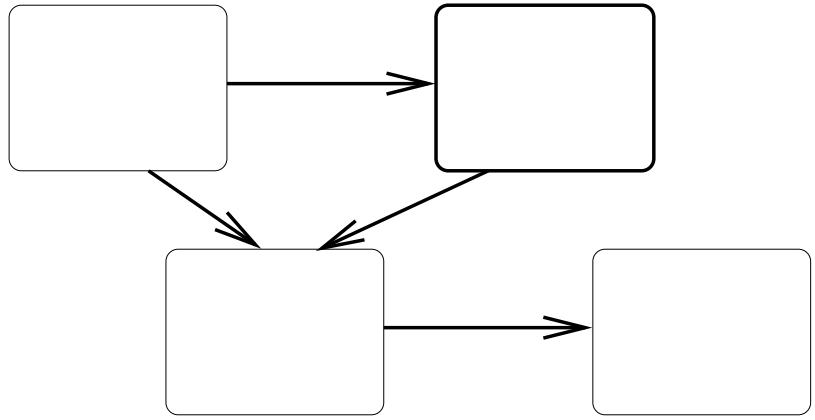
- Process Calculi (some which **model locations**):
  - Mobile Ambients & Ambient Logic: (various) (Cardelli, Caires, Ghelli, Gordon ’98-’02)
  - DPI: “Resource Access Control...” (Hennessy, *et al.* ’02)
  - Klaim: “Types for Access Control” (De Nicola, Ferrari, *et al.* ’00)

Outline

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

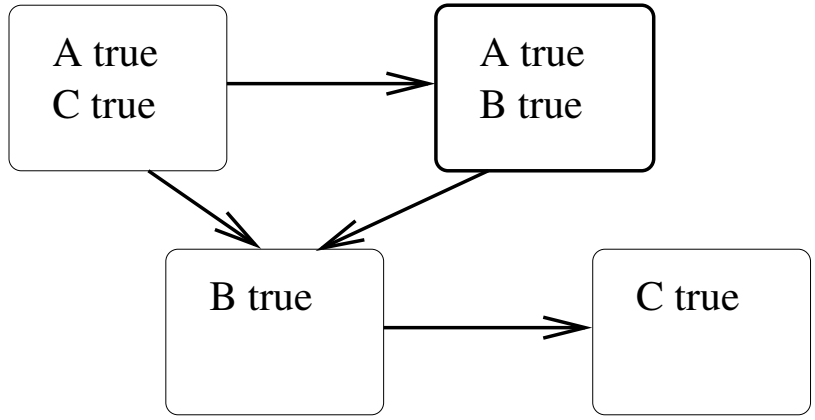
Concepts of modal logic

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



Concepts of modal logic

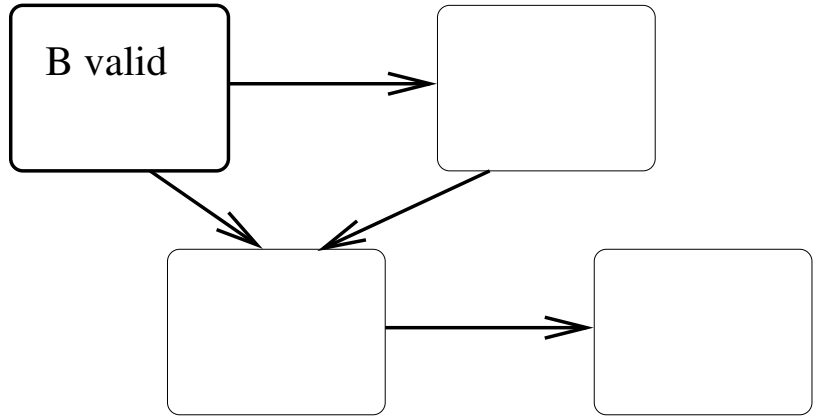
- Modal logics distinguish **modes** or degrees of truth.
- We have **worlds** related by **accessibility**.



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

Concepts of modal logic

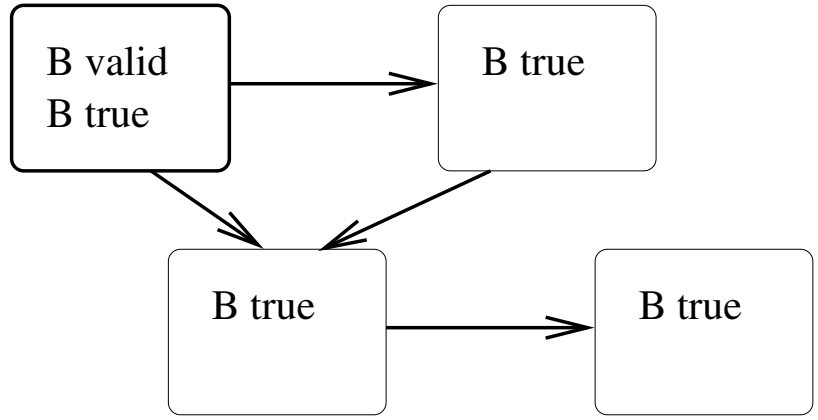
- Modal logics distinguish **modes** or degrees of truth.
- We have **worlds** related by **accessibility**.



- 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

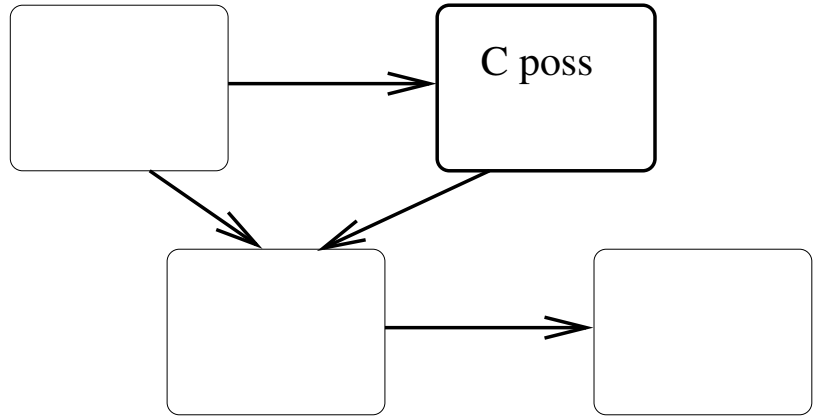
- Modal logics distinguish **modes** or degrees of truth.
- We have **worlds** related by **accessibility**.



- 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

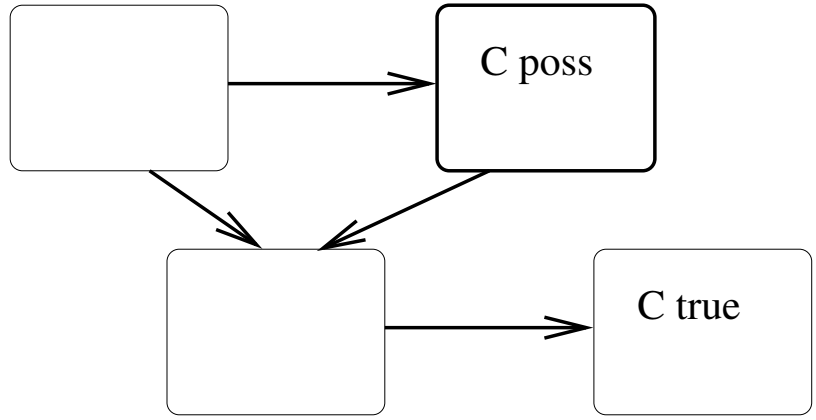
- Modal logics distinguish **modes** or degrees of truth.
- We have **worlds** related by **accessibility**.



- 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.

Concepts of modal logic

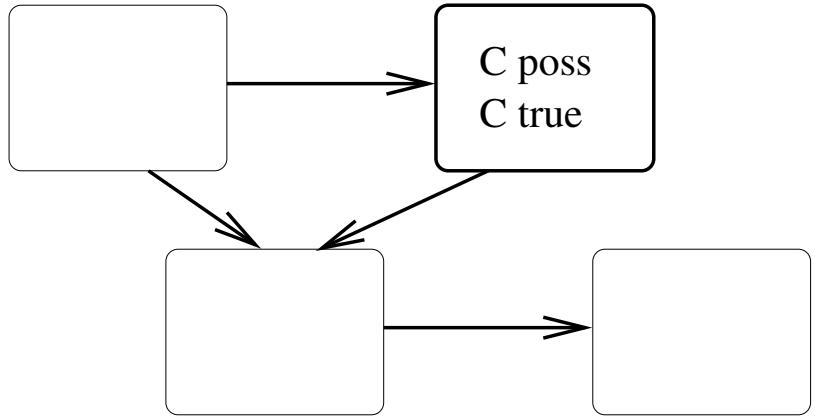
- Modal logics distinguish **modes** or degrees of truth.
- We have **worlds** related by **accessibility**.



- 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

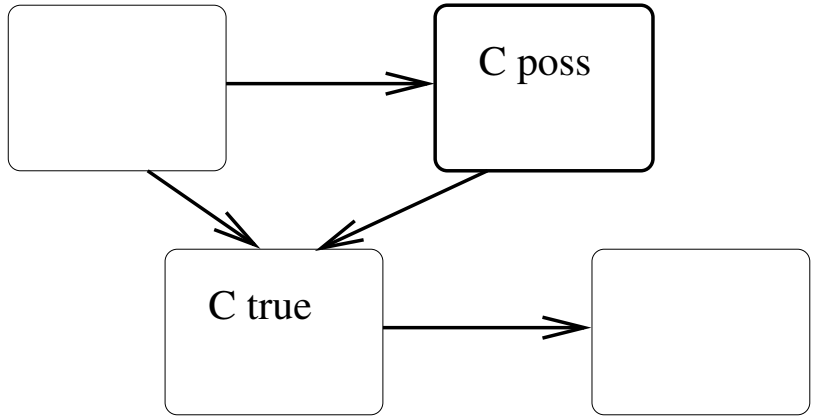
- Modal logics distinguish **modes** or degrees of truth.
- We have **worlds** related by **accessibility**.



- 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.

Hypothetical judgments

- Δ — "global" assumptions A `valid`
- Γ — "local" assumptions A `true`

$\Delta \vdash A$ `valid` $\qquad$ $\Delta; \Gamma \vdash A$ `true` $\qquad$ $\Delta; \Gamma \vdash A$ `possible`

Hypothetical judgments

- Δ — "global" assumptions $A\ \texttt{valid}$
- Γ — "local" assumptions $A\ \texttt{true}$

$$\Delta \vdash A\ \texttt{valid} \qquad \Delta;\Gamma \vdash A\ \texttt{true} \qquad \Delta;\Gamma \vdash A\ \texttt{possible}$$

$$\Delta \vdash A\ \texttt{valid} \quad \equiv \quad \Delta;\cdot \vdash A\ \texttt{true}$$

Modal propositions

$\Box A$ — "necessarily A"

$$\frac{}{\Delta, A\ \mathtt{valid}; \Gamma \vdash A\ \mathtt{true}}\ hyp^* \qquad \frac{\Delta; \cdot \vdash A\ \mathtt{true}}{\Delta; \Gamma \vdash \Box A\ \mathtt{true}}\ \Box I$$

$$\frac{\Delta; \Gamma \vdash \Box A\ \mathtt{true} \quad \Delta, A\ \mathtt{valid}; \Gamma \vdash C\ \mathtt{true}}{\Delta; \Gamma \vdash C\ \mathtt{true}}\ \Box E$$

Modal propositions

$\Box A$ — "necessarily A"

$$\frac{}{\Delta, A\ \mathtt{valid}; \Gamma \vdash A\ \mathtt{true}}\ hyp^* \qquad \frac{\Delta; \cdot \vdash A\ \mathtt{true}}{\Delta; \Gamma \vdash \Box A\ \mathtt{true}}\ \Box I$$

$$\frac{\Delta; \Gamma \vdash \Box A\ \mathtt{true} \quad \Delta, A\ \mathtt{valid}; \Gamma \vdash C\ \mathtt{true}}{\Delta; \Gamma \vdash C\ \mathtt{true}}\ \Box E$$

$\Diamond A$ — "possibly A"

$$\frac{\Delta; \Gamma \vdash A\ \mathtt{true}}{\Delta; \Gamma \vdash A\ \mathtt{possible}}\ poss \qquad \frac{\Delta; \Gamma \vdash A\ \mathtt{possible}}{\Delta; \Gamma \vdash \Diamond A\ \mathtt{true}}\ \Diamond I$$

$$\frac{\Delta; \Gamma \vdash \Diamond A\ \mathtt{true} \quad \Delta; A\ \mathtt{true} \vdash C\ \mathtt{possible}}{\Delta; \Gamma \vdash C\ \mathtt{possible}}\ \Diamond E$$

Outline

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

Towards a distributed calculus

- Judgements

Logical	Typing	Operational
$\Delta;\Gamma \vdash A\ \mathtt{true}$	$\Delta^*;\Gamma^* \vdash M : A$	"evaluate to A locally"
$\Delta;\Gamma \vdash A\ \mathtt{possible}$	$\Delta^*;\Gamma^* \vdash E \div A$	"produce A somewhere"

- Propositions

Prop/Type	Logical Reading	Type Reading
$\Box A$	"necessarily A"	"mobile A"
$\Diamond A$	"possibly A"	"remote A"

Typing: $\Box A$

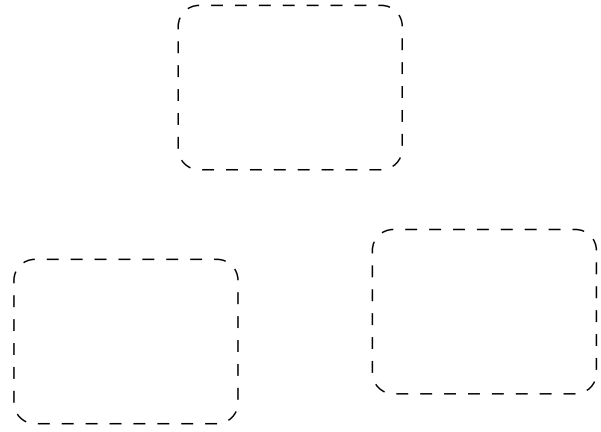
$$\frac{}{\Delta, \mathtt{u} :: A; \Gamma \vdash \mathtt{u} : A}\ hyp^* \qquad \frac{\Delta; \cdot \vdash M : A}{\Delta; \Gamma \vdash \mathtt{box}\, M : \Box A}\ \Box I$$

$$\frac{\Delta; \Gamma \vdash M : \Box A \quad \Delta, \mathtt{u} :: A; \Gamma \vdash N : B}{\Delta; \Gamma \vdash \mathtt{let\ box}\, \mathtt{u} = M\, \mathtt{in}\, N : B}\ \Box E$$

Operational: $\Box A$

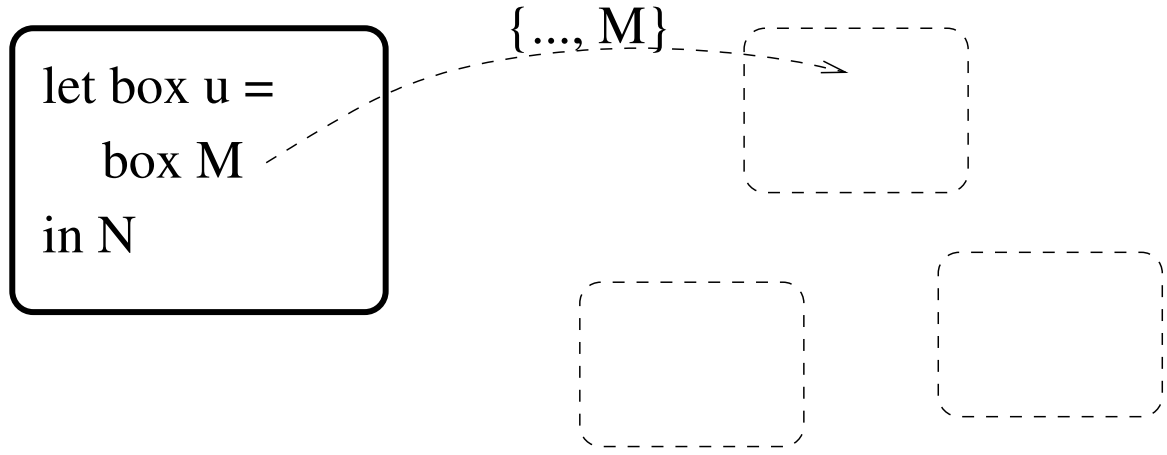
- Intuition

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



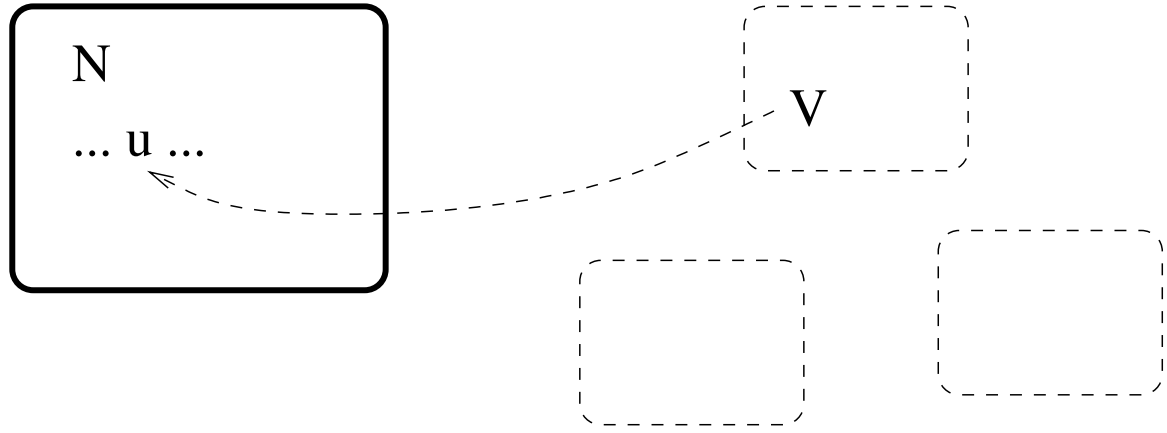
Operational: $\Box A$

- Intuition



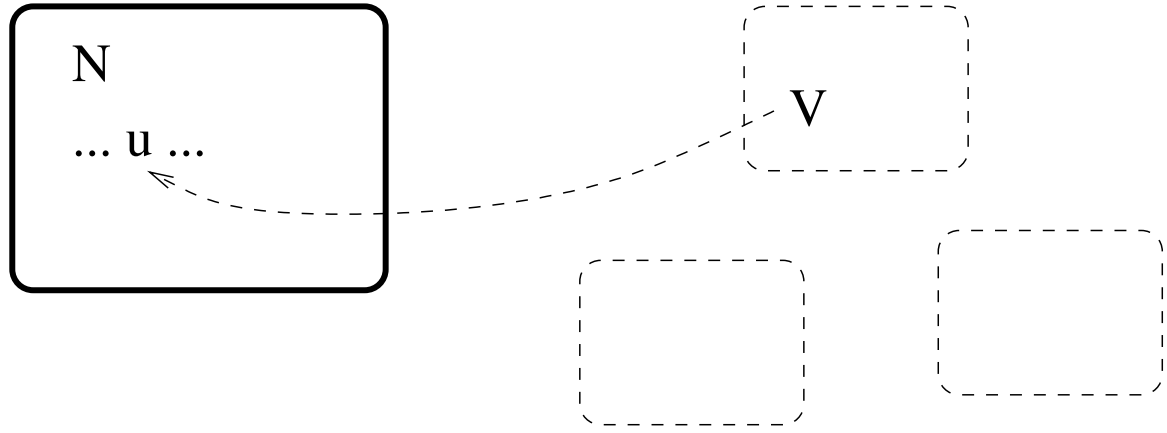
Operational: $\Box A$

- Intuition



Operational: $\Box A$

- Intuition



- Formally

$$\langle r_1 : \mathcal{R}[\,\texttt{let box}\,\mathbf{u} = \texttt{box}\,M\,\texttt{in}\,N\,]\rangle \quad \Longrightarrow \quad \langle r_2 : M\rangle, \quad \langle r_1 : \mathcal{R}[\,[\![r_2/\mathbf{u}]\!]N\,]\rangle$$

$$\langle r_2 : V\rangle, \quad \langle r_1 : \mathcal{R}[\,r_2\,]\rangle \quad \Longrightarrow \quad \langle r_2 : V\rangle, \quad \langle r_1 : \mathcal{R}[\,V\,]\rangle$$

$$\langle r_1 : \mathcal{R}[\,\texttt{let box}\,\mathbf{u} = \texttt{box}\,V\,\texttt{in}\,N\,]\rangle \quad \Longrightarrow \quad \langle r_1 : \mathcal{R}[\,[\![V/\mathbf{u}]\!]N\,]\rangle$$

- **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 (λ x:nat . x * u)
```

- Lexically-scoped **mobile** closures capture **mobile** bindings ($u \in \Delta$).

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 $\Box(A \to B)$.

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: $\Diamond A$

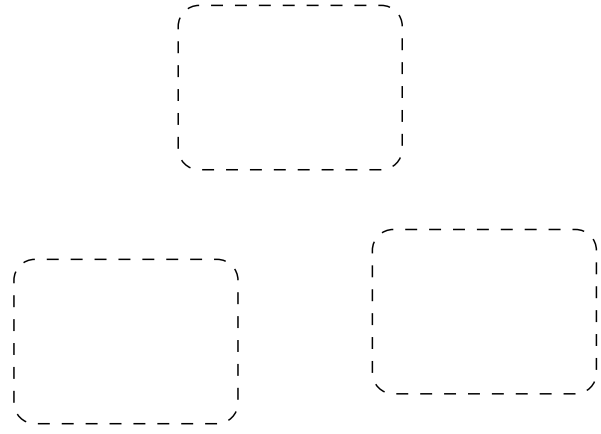
$$\frac{\Delta;\Gamma \vdash M : A}{\Delta;\Gamma \vdash \{M\} \div A}\ poss \qquad \frac{\Delta;\Gamma \vdash E \div A}{\Delta;\Gamma \vdash \mathtt{dia}\, E : \Diamond A}\ \Diamond I$$

$$\frac{\Delta;\Gamma \vdash M : \Diamond A \quad \Delta;\mathbf{x} : A \vdash F \div B}{\Delta;\Gamma \vdash \mathtt{let\ dia}\,\mathbf{x} = M\ \mathtt{in}\ F \div B}\ \Diamond E$$

Operational: $\Diamond A$

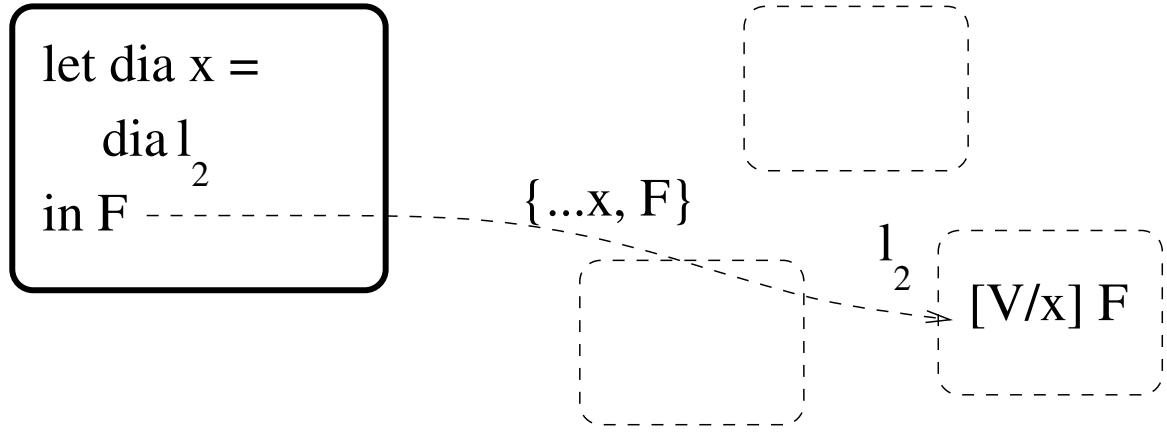
- Intuition

let dia x =
    dia V^*
in F



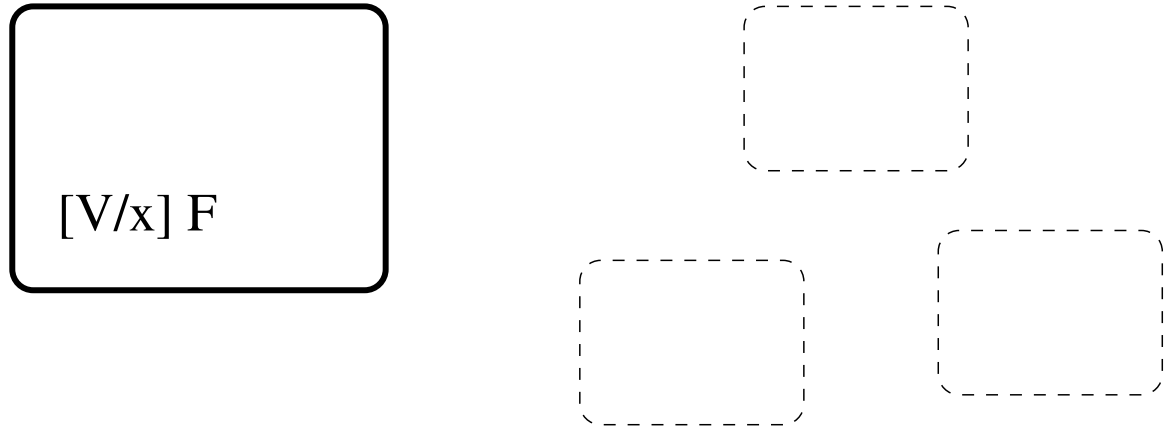
Operational: $\Diamond A$

- Intuition



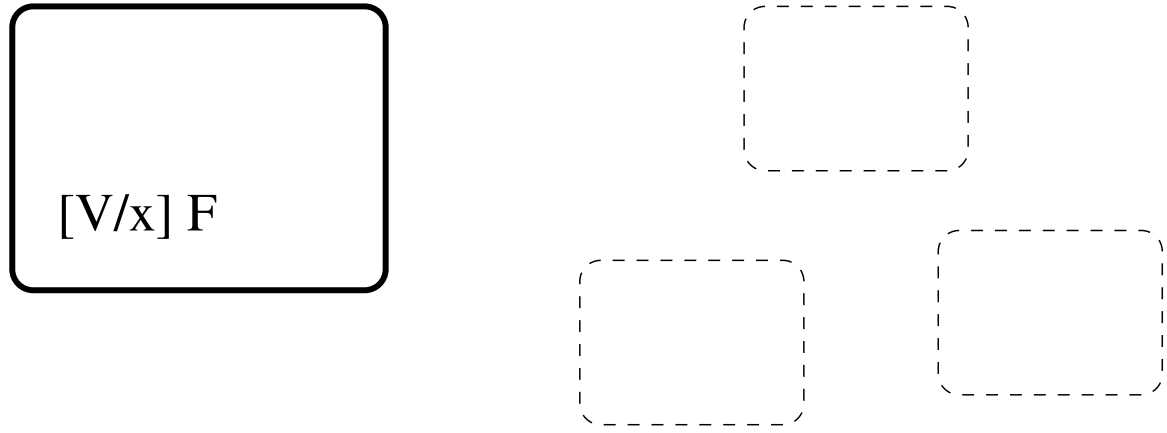
Operational: $\Diamond A$

- Intuition



Operational: $\Diamond A$

- Intuition



- Formally

$$\langle l_1 : \mathcal{S}[\,\texttt{let dia}\,\mathbf{x} = \texttt{dia}\,l_2\,\texttt{in}\,F\,]\rangle, \quad \langle l_2 : \{V\}\rangle$$
$$\Longrightarrow \quad \langle l_1 : \mathcal{S}[\,l_2'\,]\rangle, \quad \langle l_2' : [V/\mathbf{x}]F\rangle, \quad \langle l_2 : \{V\}\rangle$$

$$\langle l_1 : \mathcal{S}[\,\texttt{let dia}\,\mathbf{x} = \texttt{dia}\,\{V\}\,\texttt{in}\,F\,]\rangle$$
$$\Longrightarrow \quad \langle l_1 : \mathcal{S}[\,[V/\mathbf{x}]F\,]\rangle$$

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

$$
\begin{array}{rrl}
\text{Type } A, B & ::= & A \to B \;\mid\; \Box A \;\mid\; \Diamond A \\
\text{Term } M, N & ::= & \mathtt{x} \;\mid\; \mathtt{u} \\
& & \mid\; \lambda \mathtt{x} : A \,.\, M \;\mid\; M\ N \\
& & \mid\; \mathtt{box}\, M \;\mid\; \mathtt{dia}\, E \\
& & \mid\; \mathtt{let\ box}\, \mathtt{u} = M\ \mathtt{in}\ N \\
\text{Expr. } E, F & ::= & \{ M \} \;\mid\; \mathtt{let\ box}\, \mathtt{u} = M\ \mathtt{in}\ F \\
& & \mid\; \mathtt{let\ dia}\, \mathtt{x} = M\ \mathtt{in}\ F
\end{array}
$$

Core calculus — summary

$$
\begin{array}{rcl}
\text{Type } A, B & ::= & A \to B \;\mid\; \Box A \;\mid\; \Diamond A \\
\text{Term } M, N & ::= & \boxed{r} \;\mid\; \mathtt{x} \;\mid\; \mathtt{u} \\
& & \mid\; \lambda \mathtt{x} : A \,.\, M \;\mid\; M\ N \\
& & \mid\; \mathtt{box}\, M \;\mid\; \mathtt{dia}\, E \\
& & \mid\; \mathtt{let\ box}\, \mathtt{u} = M\, \mathtt{in}\, N \\
\text{Expr. } E, F & ::= & \boxed{l} \;\mid\; \{M\} \;\mid\; \mathtt{let\ box}\, \mathtt{u} = M\, \mathtt{in}\, F \\
& & \mid\; \mathtt{let\ dia}\, \mathtt{x} = M\, \mathtt{in}\, F \\
\text{Label } w & ::= & r \;\mid\; l \\
\text{Process } \pi & ::= & \langle r : M \rangle \;\mid\; \langle l : E \rangle \\
\text{Config. } C & ::= & \cdot \;\mid\; C, \pi
\end{array}
$$

Process configurations

$\boxed{C \Longrightarrow C'}$ — "C steps to C'".

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

Process configurations

$\boxed{C \Longrightarrow C'}$ — "C steps to C'".

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

$\boxed{\psi \vdash^c C : \Lambda}$ — "conf. C has type Λ (under ψ)".

- $\Lambda = r_1 :: A, \ldots, l_2 \div A, \ldots$
- ψ — determines scope/accessibility of labels.

Properties

- **Type preservation**:

$$\psi \vdash^{c} C : \Lambda \quad \text{and} \quad C \Longrightarrow C'$$
$$\Longrightarrow \quad \exists \Lambda' \supseteq \Lambda \,.\, \exists \psi' \,.\, \psi' \vdash^{c} C' : \Lambda'$$

  - Λ' and ψ' grow as processes are created.

Properties

- **Type preservation**:

$$\psi \vdash^c C : \Lambda \quad \text{and} \quad C \Longrightarrow C'$$
$$\Longrightarrow \quad \exists \Lambda' \supseteq \Lambda \,.\, \exists \psi' \,.\, \psi' \vdash^c C' : \Lambda'$$

  - Λ' and ψ' grow as processes are created.

- **Progress**:

$$\psi \vdash^c C : \Lambda \quad \text{and} \quad \psi \,\texttt{noncyclic}$$
$$\Longrightarrow \quad \exists C' \,.\, C \Longrightarrow C' \quad \text{or} \quad C \,\texttt{terminal}$$

  - ψ `noncyclic` — permits inductive argument.
  - deadlocked: $\langle r_1 : r_2\,(\lambda \mathtt{x} : A \,.\, \mathtt{x})\rangle, \langle r_2 : r_1\,(\lambda \mathtt{x} : A \,.\, \mathtt{x})\rangle$

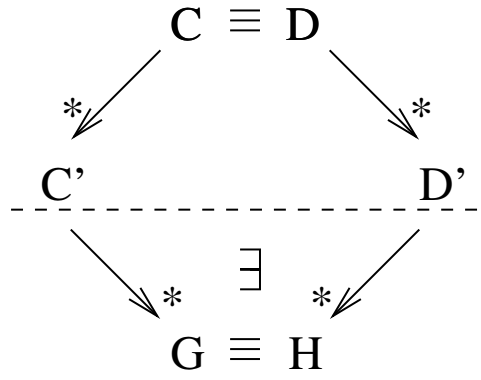
Properties

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

Properties

- **Termination**: sequences $C_1 \Longrightarrow C_2 \Longrightarrow \ldots$ halt
  - core calculus (without fixpoints).
  - $\psi \vdash^c C_1 : \Lambda_1$ — well-formed configuration.
  - ψ `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 ($\texttt{nat} \to \texttt{nat}$) 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 ($\bigcirc A$) and **local computations**.

Typing	Operational
$\Delta;\Gamma \vdash M : A$	"evaluate to A locally"
$\Delta;\Gamma \vdash P \approx A$	"produce A locally with effects"
$\Delta;\Gamma \vdash E \div A$	"produce A somewhere with effects"

Example: mutable ref

- PROHIBITED: mobility for mutable refereces.

```
(* 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 ($\bigcirc A$) 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:

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 ($\bigcirc A$), mutable refs.

- Proposed: Polymorphism ($\forall$), abstract types ($\exists$).
  - **Well-known** problem sharing abstract values α between locations.
  - Permutations of $\exists\alpha\,.\,B$ and $\Box\Diamond$ 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` ($\Box A$).
  - Link with **local** libraries (trusted/certified mix).
  - Bind and use remote resources ($\Diamond A$).

Details

- **Hemlock** extensions/modifications:
  - Type system for $\Box$, $\Diamond$ and effects.

Details

- **Hemlock** extensions/modifications:
  - Type system for $\Box$, $\Diamond$ and effects.
  - Code generation for new box/dia features.

Details

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

Details

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

- **ConCert** runtime extensions (support $\Box$,$\Diamond$ model):
  - Mapping, binding to $(\Diamond A)$ resources.

Details

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

- **ConCert** runtime extensions (support $\Box$,$\Diamond$ model):
  - Mapping, binding to $(\Diamond A)$ resources.
  - Targeted "closures" $\{\mathtt{x} \mapsto \bullet; E\}$
    (arising from $\mathtt{let\ dia}\,\mathtt{x} = M\ \mathtt{in}\ E$)

Strategy: Tasks and priorities

Priority	Effort	Task
high	med	Polymorphism, abstract types
high	med	Parsing, typechecking ($\Box$, $\Diamond$, & effects)
high	high	Hemlock runtime marshalling code (TALT)
high	high	TALT code gen. box/dia features

Strategy: Tasks and priorities

Priority	Effort	Task
high	med	Polymorphism, abstract types
high	med	Parsing, typechecking ($\Box$, $\Diamond$, & effects)
high	high	Hemlock runtime marshalling code (TALT)
high	high	TALT code gen. box/dia features
med	med	ConCert runtime support ($\Diamond$) (ML)
med	high	Abstract resources ($\Diamond\exists$), Modules?

Strategy: Tasks and priorities

Priority	Effort	Task
high	med	Polymorphism, abstract types
low	med	Dependent types, policy-related types
high	med	Parsing, typechecking ($\Box$, $\Diamond$, & effects)
high	high	Hemlock runtime marshalling code (TALT)
high	high	TALT code gen. box/dia features
med	med	ConCert runtime support ($\Diamond$) (ML)
med	high	Abstract resources ($\Diamond\exists$), Modules?

End

Questions?

Strategy: Tasks and Priorities

Time	Date	Task
1	1	Polymorphism, abstract types
?	?	Dependent types, policy-related types
2-4	3-5	Parsing, typechecking ($\Box$, $\Diamond$, & effects)
3-4	6-9	Hemlock runtime marshalling code (TALT)
3-4	9-13	TALT code gen. box/dia features
2-3	11-16	ConCert runtime support ($\Diamond$) (ML)
2-3	13-19	Abstract resources ($\Diamond\exists$), Modules?

Programming Models

- $\rightarrow$,$\mathtt{nat}$,$*$,$+$,... (local pure functional programs).
- $\Box$ (spawn mobile terms).
- $\Box\Diamond$ (spawn mobile terms, jump among locations).
- $\bigcirc$ (local effects)
- $\Box\bigcirc$ (spawn mobile, local effects)
- $\Box\bigcirc\Diamond$ (spawn, local effects, jumping, remote effects)

S4 and P2P Grid

- 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 ($A \not\to \Box\Diamond A$).
- 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

$$\frac{}{\Delta;\Gamma,\mathtt{x}:A,\Gamma' \vdash_J \mathtt{x}:A}\ hyp \qquad \frac{\Delta;\Gamma,\mathtt{x}:A \vdash_J M:B}{\Delta;\Gamma \vdash_J \lambda\mathtt{x}:A\,.\,M:A\to B}\ \to I$$

$$\frac{}{\Delta,\mathtt{u}::A,\Delta';\Gamma \vdash_J \mathtt{u}:A}\ hyp^* \qquad \frac{\Delta;\Gamma \vdash_J M:A\to B \quad \Delta;\Gamma \vdash_J N:A}{\Delta;\Gamma \vdash_J M\ N:B}\ \to E$$

$$\frac{\Delta;\cdot \vdash_{J\triangleleft} M:A}{\Delta;\Gamma \vdash_J \mathtt{box}\,M:\Box A}\ \Box I \qquad \frac{\Delta;\Gamma \vdash_J M:\Box A \quad \Delta,\mathtt{u}::A;\Gamma \vdash_J N:B}{\Delta;\Gamma \vdash_J \mathtt{let\ box}\,\mathtt{u}=M\,\mathtt{in}\,N:B}\ \Box E$$

$$\frac{\Delta;\Gamma \vdash_J M:A}{\Delta;\Gamma \vdash_J \{M\}\div A}\ poss \qquad \frac{\Delta;\Gamma \vdash_J M:\Diamond A \quad \Delta;\mathtt{x}:A \vdash_{J\triangleleft} F\div B}{\Delta;\Gamma \vdash_J \mathtt{let\ dia}\,\mathtt{x}=M\,\mathtt{in}\,F\div B}\ \Diamond E$$

$$\frac{\Delta;\Gamma \vdash_J E\div A}{\Delta;\Gamma \vdash_J \mathtt{dia}\,E:\Diamond A}\ \Diamond I \qquad \frac{\Delta;\Gamma \vdash_J M:\Box A \quad \Delta,\mathtt{u}::A;\Gamma \vdash_J F\div B}{\Delta;\Gamma \vdash_J \mathtt{let\ box}\,\mathtt{u}=M\,\mathtt{in}\,F\div B}\ \Box E_p$$

Extensions

- The type theory of $(\to\Box\Diamond)$ is easily extensible:
- products $(A * B)$, sums $(A + B)$, recursive types $(\mu\alpha\,.\,B)$ ... (straightforward)

$$\frac{\Delta;\Gamma \vdash M : A \quad \Delta;\Gamma \vdash N : B}{\Delta;\Gamma \vdash (M, N) : A * B} \qquad \frac{\Delta;\Gamma \vdash M : A * B}{\Delta;\Gamma \vdash \mathtt{fst}\, M : B} \qquad \frac{\Delta;\Gamma \vdash M : A * B}{\Delta;\Gamma \vdash \mathtt{snd}\, M : A}$$

Extensions

- The type theory of $(\to\Box\Diamond)$ is easily extensible:
- products $(A * B)$, sums $(A + B)$, recursive types $(\mu\alpha\,.\,B)$ ... (straightforward)

$$\frac{\Delta;\Gamma \vdash M : A \quad \Delta;\Gamma \vdash N : B}{\Delta;\Gamma \vdash (M, N) : A * B} \qquad \frac{\Delta;\Gamma \vdash M : A * B}{\Delta;\Gamma \vdash \mathtt{fst}\, M : B} \qquad \frac{\Delta;\Gamma \vdash M : A * B}{\Delta;\Gamma \vdash \mathtt{snd}\, M : A}$$

- Fixpoints: $\mathtt{fixv}\,(\mathtt{u} :: A)\,.\,M$ and $\mathtt{fix}\,(\mathtt{x} : A)\,.\,M$

$$\frac{\Delta, \mathtt{u} :: A; \cdot \vdash M : A}{\Delta;\Gamma \vdash \mathtt{fixv}\,(\mathtt{u} :: A)\,.\,M : A} \qquad \frac{\Delta;\Gamma, \mathtt{x} : A \vdash M : A}{\Delta;\Gamma \vdash \mathtt{fix}\,(\mathtt{u} : A)\,.\,M : A}$$

Extensions

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

Extensions

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

- A heuristic...

  Location-**neutral** $\Longrightarrow$ term $M : A$

  Location-**dependent** $\Longrightarrow$ expression $E \div A$

  (actually or potentially)

Extensions

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

- A heuristic...

  Location-**neutral** $\Longrightarrow$ term $M : A$

  Location-**dependent** $\Longrightarrow$ expression $E \div A$

  (actually or potentially)

- $(A \to \Box A)$ — A can be made **mobile** (marshalling)
  - Some values A are location-**independent**.
  - Others **not**!

Effect Typing

$$\frac{\Delta;\Gamma \vdash_J P \rightsquigarrow A}{\Delta;\Gamma \vdash_J \mathtt{comp}\, P : \bigcirc A}\ \bigcirc I \qquad \frac{\Delta;\Gamma \vdash_J M : \bigcirc A \quad \Delta;\Gamma, \mathbf{x} : A \vdash_J Q \rightsquigarrow B}{\Delta;\Gamma \vdash_J \mathtt{let\ comp}\, \mathbf{x} = M\, \mathtt{in}\, Q \rightsquigarrow B}\ \bigcirc E$$

$$\frac{\Delta;\Gamma \vdash_J M : A}{\Delta;\Gamma \vdash_J [M] \rightsquigarrow A}\ comp \qquad \frac{\Delta;\Gamma \vdash_J M : \Box A \quad \Delta, \mathbf{u} :: A;\Gamma \vdash_J Q \rightsquigarrow B}{\Delta;\Gamma \vdash_J \mathtt{let\ box}\, \mathbf{u} = M\, \mathtt{in}\, Q \rightsquigarrow B}\ \Box E_l$$

$$\frac{\Delta;\Gamma \vdash_J P \rightsquigarrow A}{\Delta;\Gamma \vdash_J \{P\} \div A}\ poss' \qquad \frac{\Delta;\Gamma \vdash_J M : \bigcirc A \quad \Delta;\Gamma, \mathbf{x} : A \vdash_J F \div B}{\Delta;\Gamma \vdash_J \mathtt{let\ comp}\, \mathbf{x} = M\, \mathtt{in}\, F \div B}\ \bigcirc E_p$$

Primitive effects:

$$\frac{\Theta = \Theta_1, a^w : A, \Theta_2}{\Theta;\Delta;\Gamma \vdash_w a^w : \mathtt{ref}\, A}\ addr \qquad \frac{\Delta;\Gamma \vdash_J M : A}{\Delta;\Gamma \vdash_J \mathtt{ref}\, M \rightsquigarrow \mathtt{ref}\, A}\ talloc$$

$$\frac{\Delta;\Gamma \vdash_J M : \mathtt{ref}\, A}{\Delta;\Gamma \vdash_J\ !M \rightsquigarrow A}\ tget \qquad \frac{\Delta;\Gamma \vdash_J M : \mathtt{ref}\, A \quad \Delta;\Gamma \vdash_J N : A}{\Delta;\Gamma \vdash_J M := N \rightsquigarrow 1}\ tset$$