Modal Types for Mobile Code

thesis defense

Tom Murphy VII

Robert Harper (co-chair)
Karl Crary (co-chair)
Frank Pfenning
Peter Sewell (Cambridge)

My thesis project is to design and implement a programming language for distributed computing based on logic.

Strategy

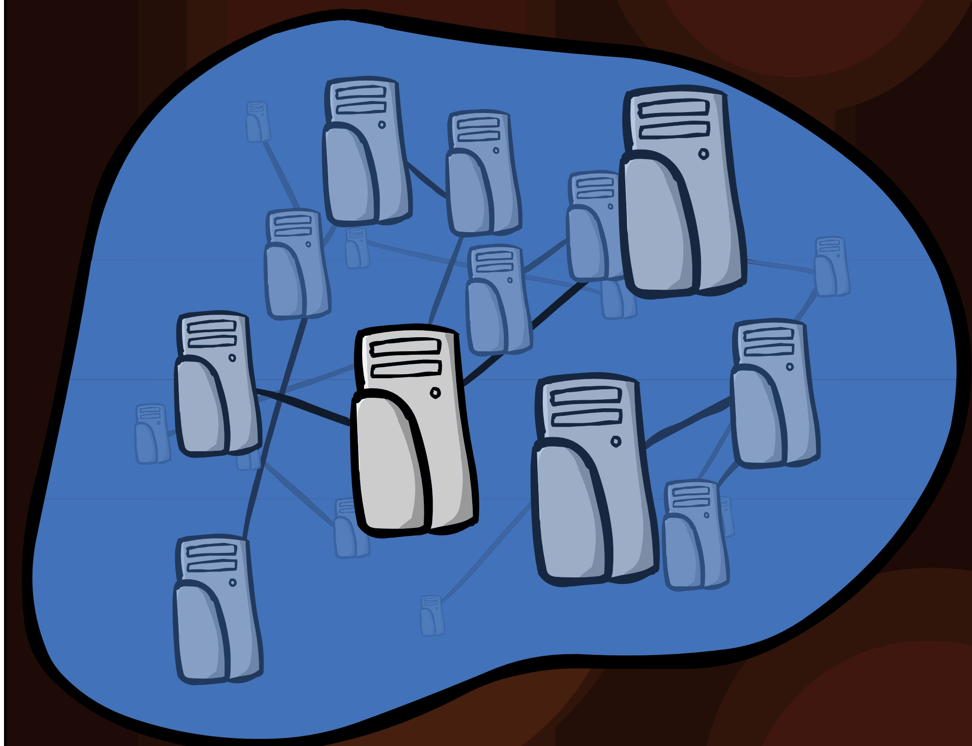
- Tell you what I did
- Argue for the thesis statement
- Present some of the best ideas from the work

Thesis statement

> "Modal type systems provide an elegant and practical means for controlling local resources in spatially distributed computer programs."

Modal type systems provide an elegant and practical means for controlling local resources in **spatially distributed computer programs**.

what?



A spatially distributed program is one that spans multiple computers in different places.

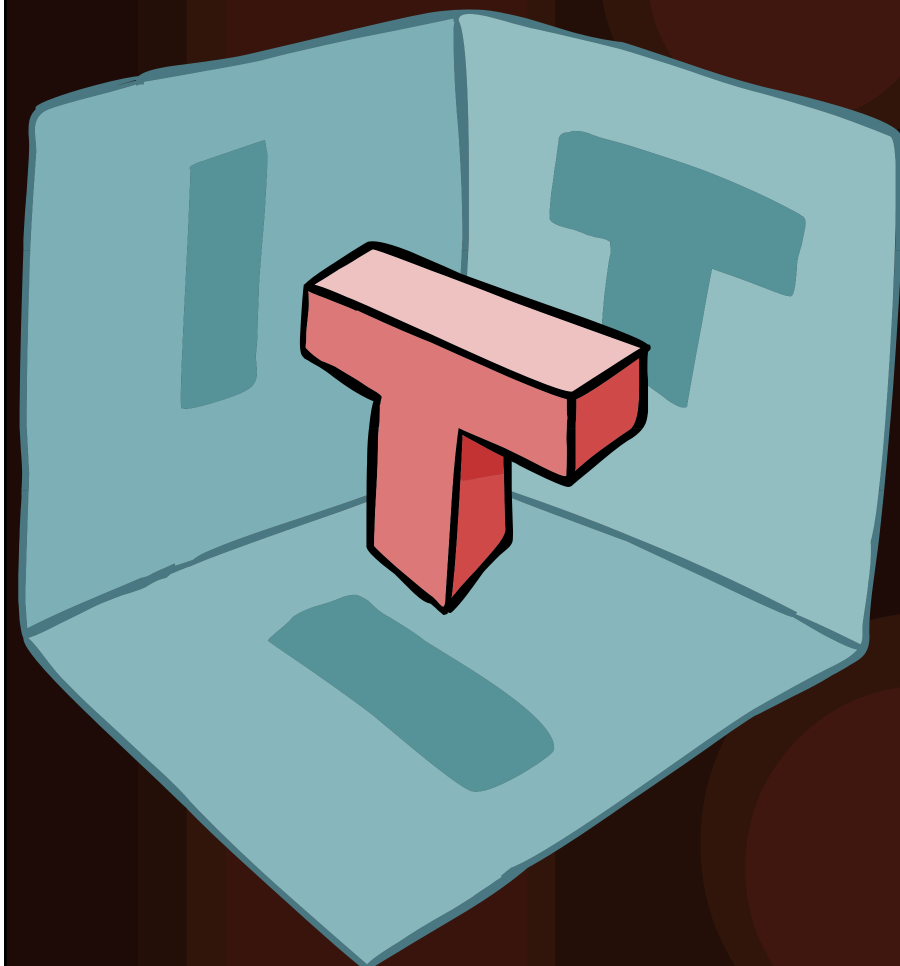
Modal type systems provide an elegant and practical means for controlling **local resources** in spatially distributed computer programs.

what?

They usually do so because of specific local resources that are only available in those places.

what?

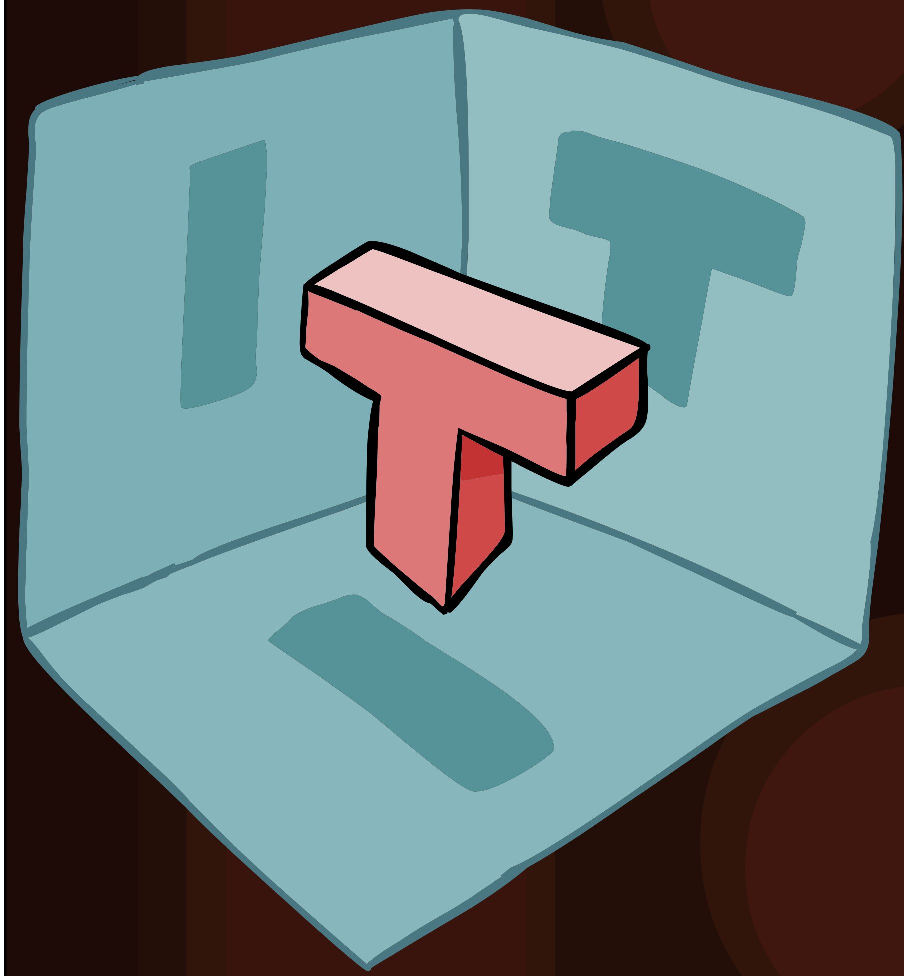
Modal type systems provide an elegant and practical means for controlling local resources in spatially distributed computer programs.



The technology I use is a modal type system, derived from modal logic. A modal logic is one that can reason about truth from multiple simultaneous perspectives, called worlds.

what?

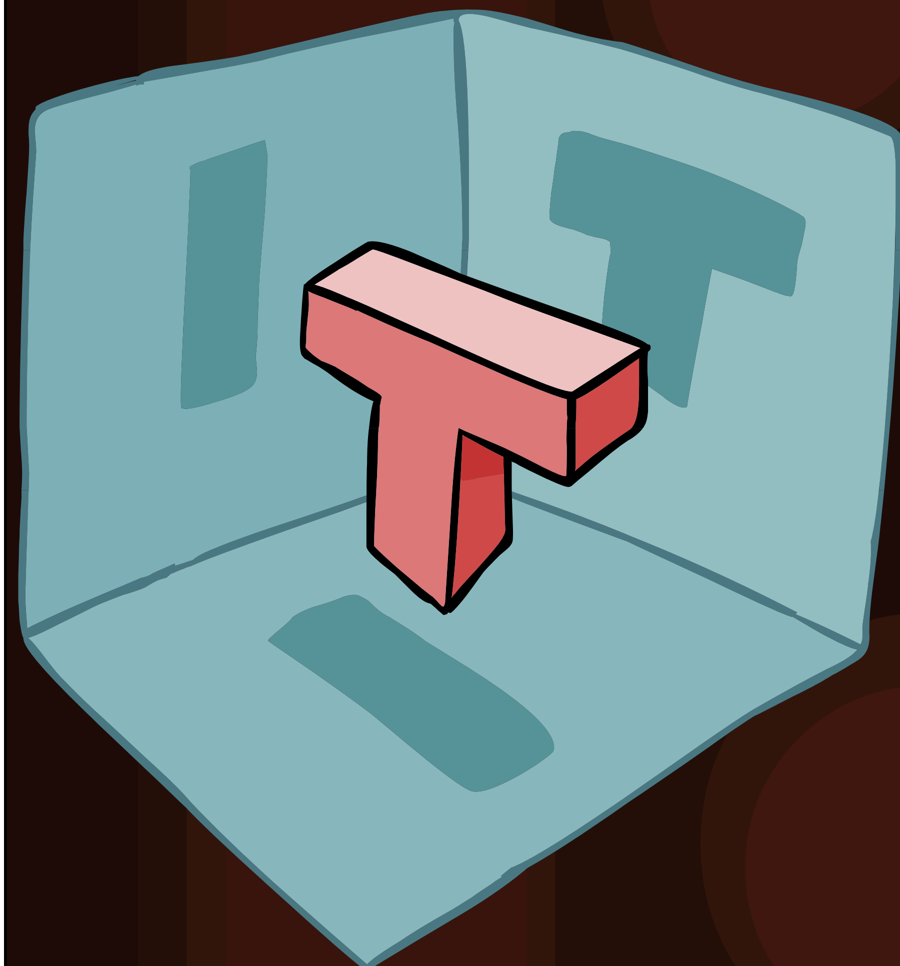
Modal type systems provide an elegant and practical means for controlling local resources in spatially distributed computer programs.



I interpret these **worlds** as the **places** in a distributed program, which leads to a methodology I call **located programming**.

how?

Modal type systems provide an elegant and practical **means for controlling** local resources in spatially distributed computer programs.



Each part of the program is associated with the **place** in which it makes sense. The language is **simultaneously aware** of each place's **differing perspective** on the **code** and **data**.

why?

Modal type systems provide an elegant and practical means for controlling local resources in spatially distributed computer programs.

To show it is elegant, I present a modal logic formulated for this purpose, show how a language can be derived from it, and prove properties of these in Twelf.

why?

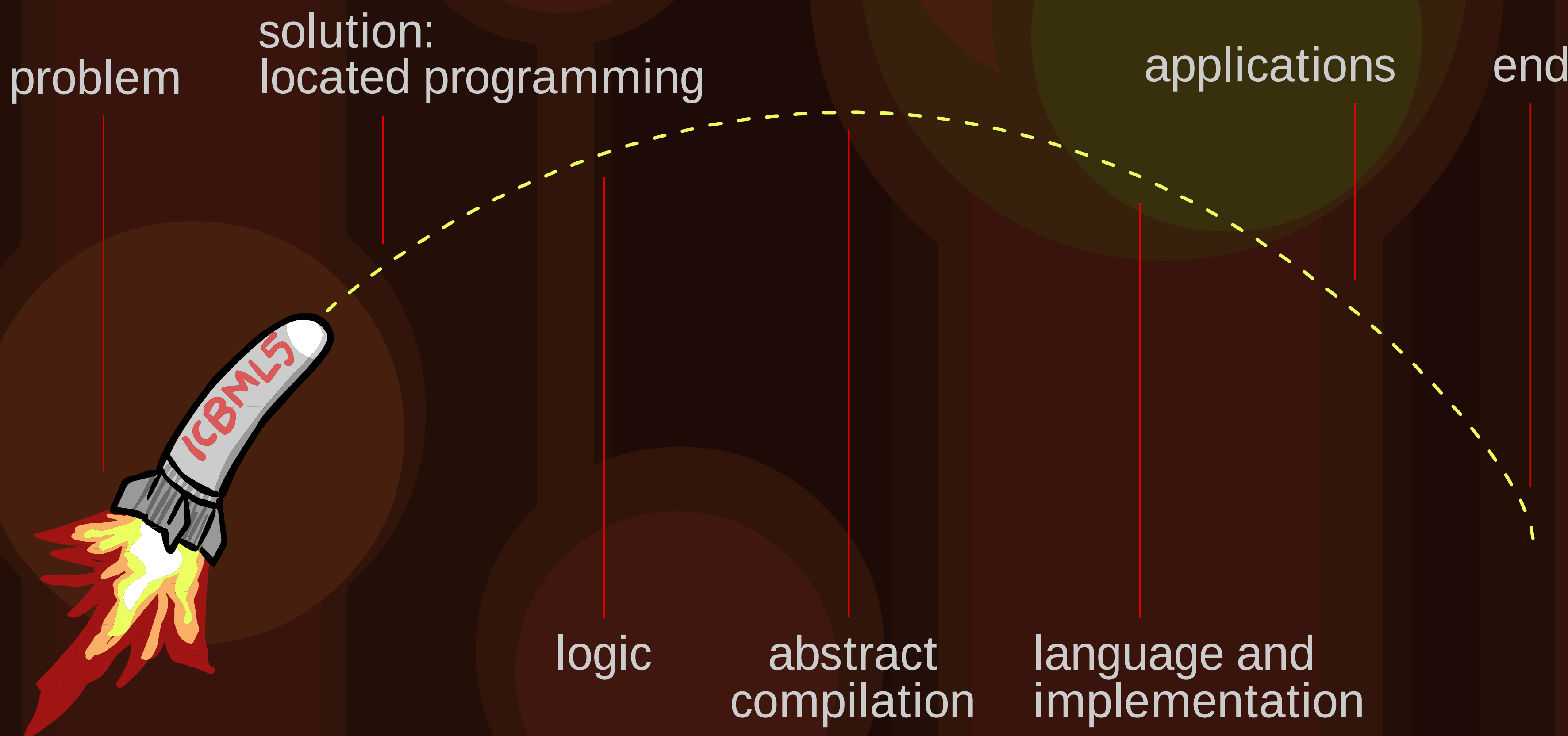
Modal type systems provide an elegant and **practical** means for controlling local resources in spatially distributed computer programs.

To show it is practical, I extend the language to a full-fledged programming language based on ML, specialized to web programming. I then build realistic applications in the language.

Outline

This work has a nice end-to-end character.

The talk is arranged according to the same trajectory as the research, dissertation.



The single-vision problem

The single-vision problem

Most languages: values and code **classified** from a single universal viewpoint.

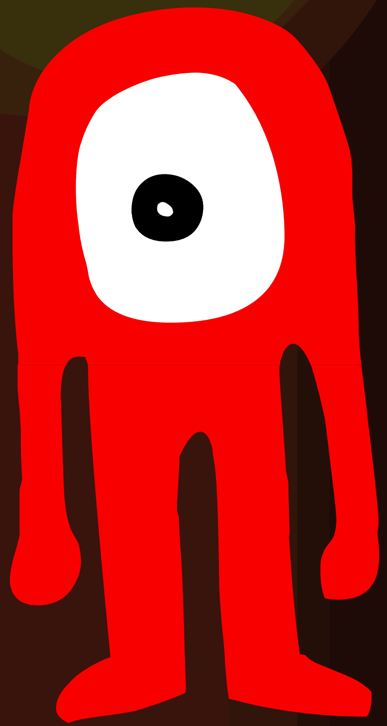
↳ "integer," "file handle," etc.

The single-vision problem

Most languages: values and code classified from a single universal viewpoint.

↳ "integer," "file handle," etc.

This monocularism leads to failures that are too early or too late.



The single-vision problem

Consider the remote procedure call.

Kurt

```
let
  val e = 5
  val y = h(e)
in
  print y
end
```

```
fun h(e : int) =
   e + 1
```

Bert

The single-vision problem

Consider the remote procedure call.

Kurt

```
let
  val e = 5
  val y = h(e)
in
  print y
end
```

5

6

```
fun h(e : int) =
  e + 1
```

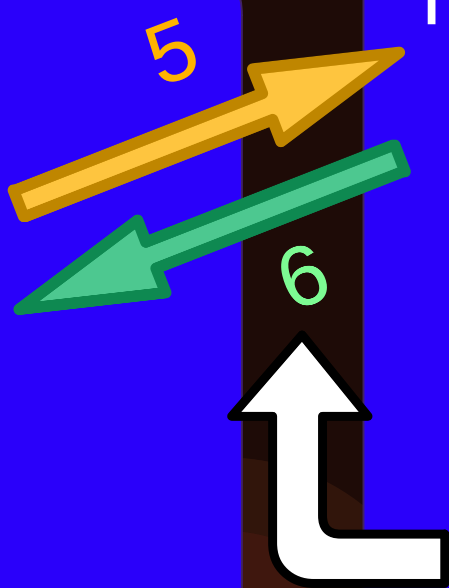
Bert

The single-vision problem

Consider the remote procedure call.

Kurt

```
let
  val e = 5
  val y = h(e)
in
  print y
end
```




```
fun h(e : int) =
  e + 1
```

also, marshaling

Bert

The single-vision problem

What about local resources?

Kurt

```
let
  val e : file =
     open "thesis.tex"
  val y = g(e)
in
  (* ... *)
end
```

```
fun g(e : file) =
  (* ... *)
```

Bert

The single-vision problem

What about local resources?

Kurt

```
let
  val e : file =
     open "thesis.tex"
  val y = g(e)
in
  (* ... *)
end
```

?

```
fun g(e : file) =
  (* ... *)
```

Bert

The single-vision problem

What happens depends on the language.

The single-vision problem

What happens depends on the language.

POD. Program is rejected statically.

"You may only send plain old data."

– [DCOM/CORBA/XMLRPC, etc.]

RPC. Program fails at RPC time.

"Can't serialize local resources."

– [Java/Acute/Alice, etc.]

The single-vision problem

DYN. Program continues, might fail in function g.

"Decide at the last second."

– [Dynamically typed languages/Grid/ML, etc.]

MOB. Transparent mobility.

[D'caml, etc.]

Diagnosis

(POD) is overconservative.

- ↳ fun g(f : file) = f
- ↳ occurs in practice!
  (Callbacks)

Diagnosis

(POD) is overconservative.

- fun g(f : file) = f
- occurs in practice!

(RPC) admits runtime failures.

- even on safe programs such as above

Diagnosis

(POD) is overconservative.

- fun g(f : file) = f
- occurs in practice!

(RPC) admits runtime failures.

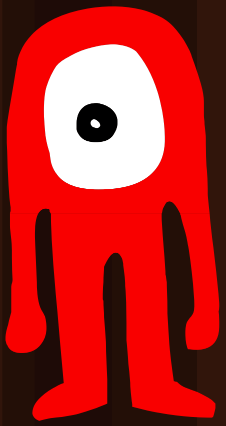
- even on safe programs such as above

(DYN) admits runtime failures.

- allows fun g(f : file) = f
- fails on fun g(f : file) = write(f, "hello")

What's going on?

Even though a file handle is a local resource, we have a single global notion (type) of file.



What's going on?

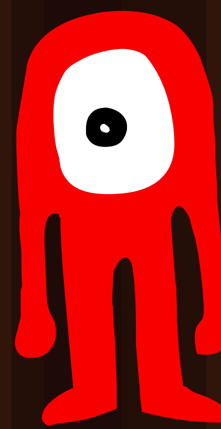
Even though a file handle is a local resource, we have a single global notion (type) of file.

If Bert has a file, he (reasonably) expects to be able to write to it.

(POD) and (RPC) prevent Bert from ever getting the file.

(DYN) checks that every file access is local.

(MOB) makes every file global.



(LOC) ...

Located programming

Instead: treat all code and data as relative to a world.

- ↳ e.g. Kurt, Burt
- ↳ allows language notion of "Kurt's file"

Located programming

Kurt's code

```
let
  val e : kurt's file =
     open "thesis.tex"
  val y = g(e)
in
  write(y, "hello")
end
```

```
fun g(e : kurt's file) =
  e
```

Bert's code

Located programming

This excludes unsafe uses **statically**.

Kurt's code

```
let
  val e : kurt's file =
    open "thesis.tex"
  val y = g(e)
in
  (* ... *)
end
```

```
fun g(e : kurt's file) =
  write(e, "oops")
```

X type error

Bert's code

Located programming

Kurt

```
let
  val e : kurt's int
                   = 5
  val y = h(e)
in
  print y
end
```

```
fun h(e : kurt's int) =
   e + 1
```

Bert

Located programming

Kurt

```
let
  val e : kurt's int
                   = 5
  val y = h(e)
in
  print y
end
```

```
fun h(e : kurt's int) =
   e + 1    ?
```

Bert

Located programming

Kurt

```
let
  val e : kurt's int
                    = 5

  val y = h(e)
in
  print y
end
```

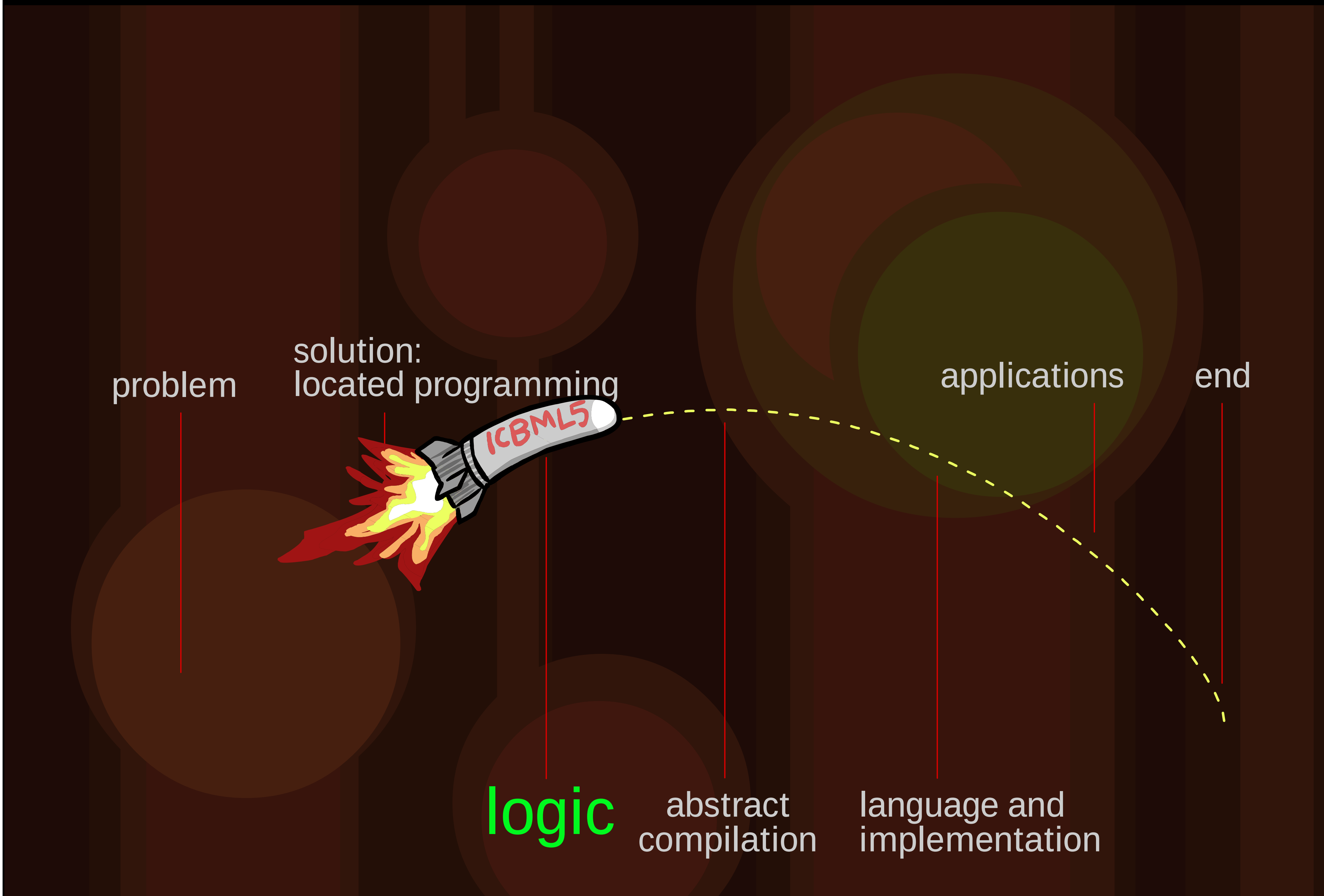

```
fun h(e : bert's int) =
  e + 1
```

Bert

Located programming

Semantic question: When can we convert Kurt's t to Bert's t?

- file: no, int: yes
- This is not the same as marshaling


problem
solution:
located programming
ICBML5
applications
end
logic
abstract
compilation
language and
implementation

Modal logic

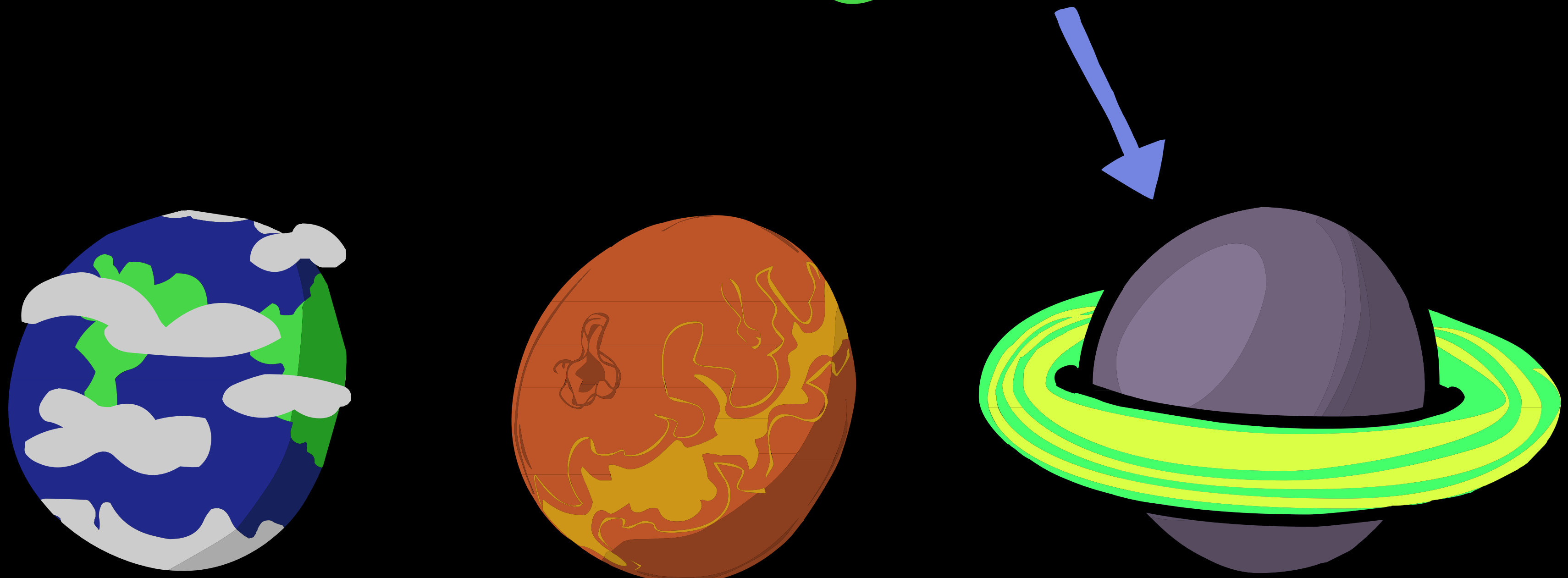
A **logic** is concerned with the truth of propositions.

"A true"

Modal logic

Modal logic is concerned with the truth of propositions, relative to a set of worlds.

"A true @ w_1"



Modal logic

Modal logic is concerned with the truth of propositions, relative to a set of worlds.

"A true @ w_1"

(A proposition might only be true in some worlds because of different contingent facts at those worlds.)

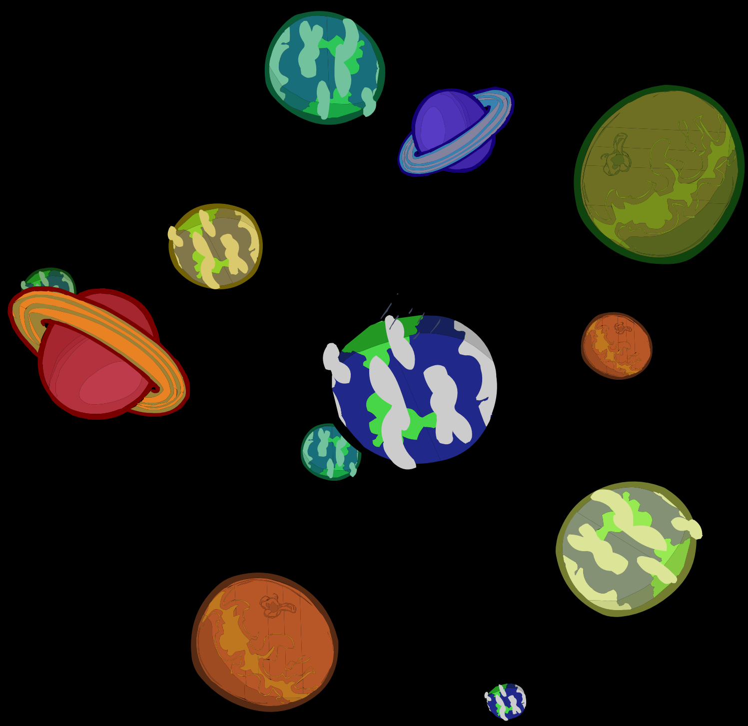
Modal logic

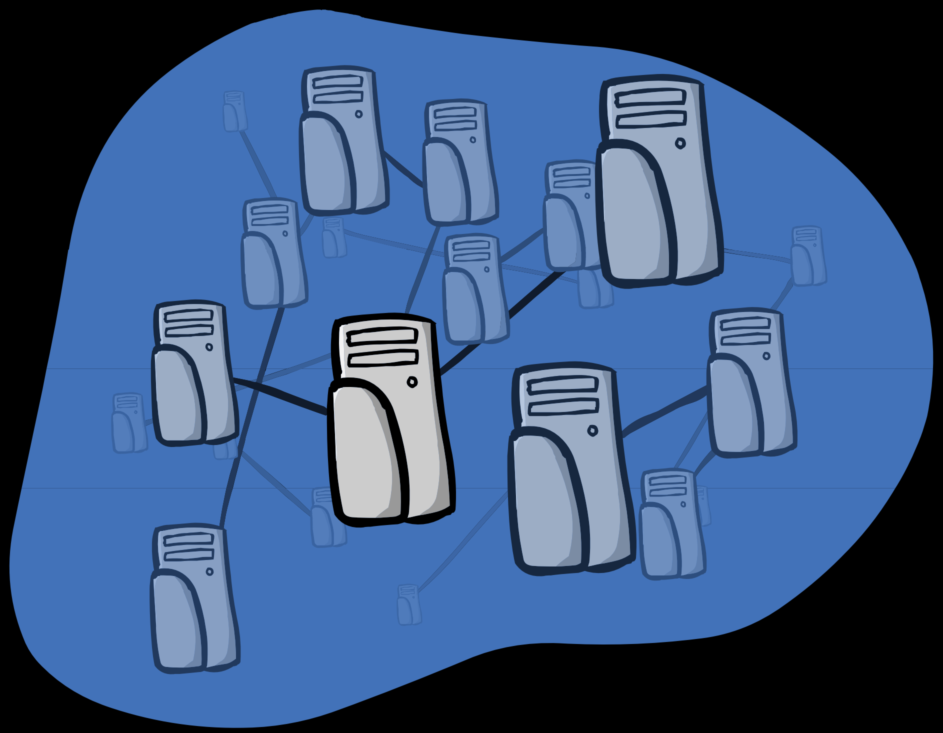
Contingent facts are represented by hypotheses, themselves relative to a set of worlds.

$$A \text{ true} @ w_1, B \text{ true} @ w_2 \vdash A \text{ true} @ w_1$$

$$A \text{ true} @ w_1, B \text{ true} @ w_2 \nvdash A \text{ true} @ w_2$$

Modal logic

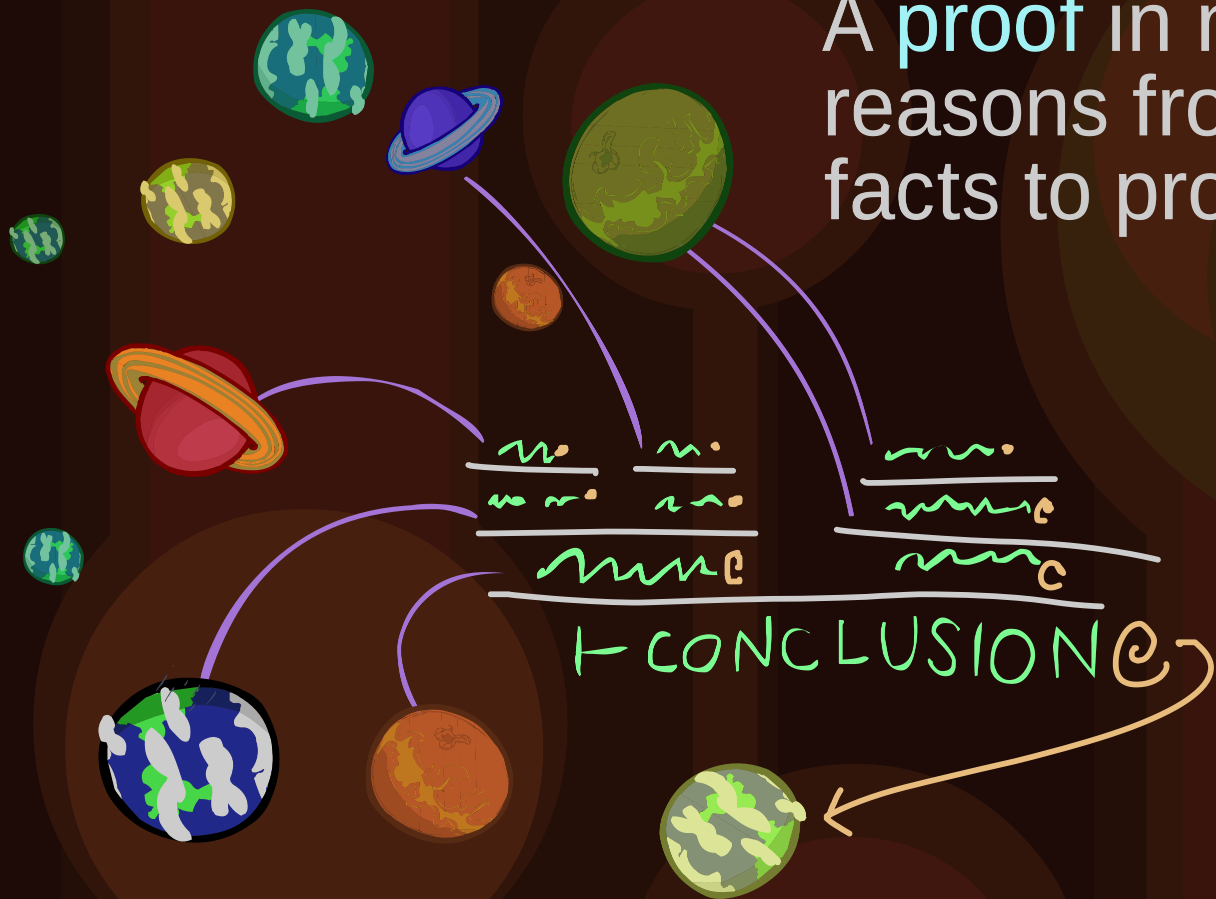




(Again, we'll think of worlds as hosts on the network.)

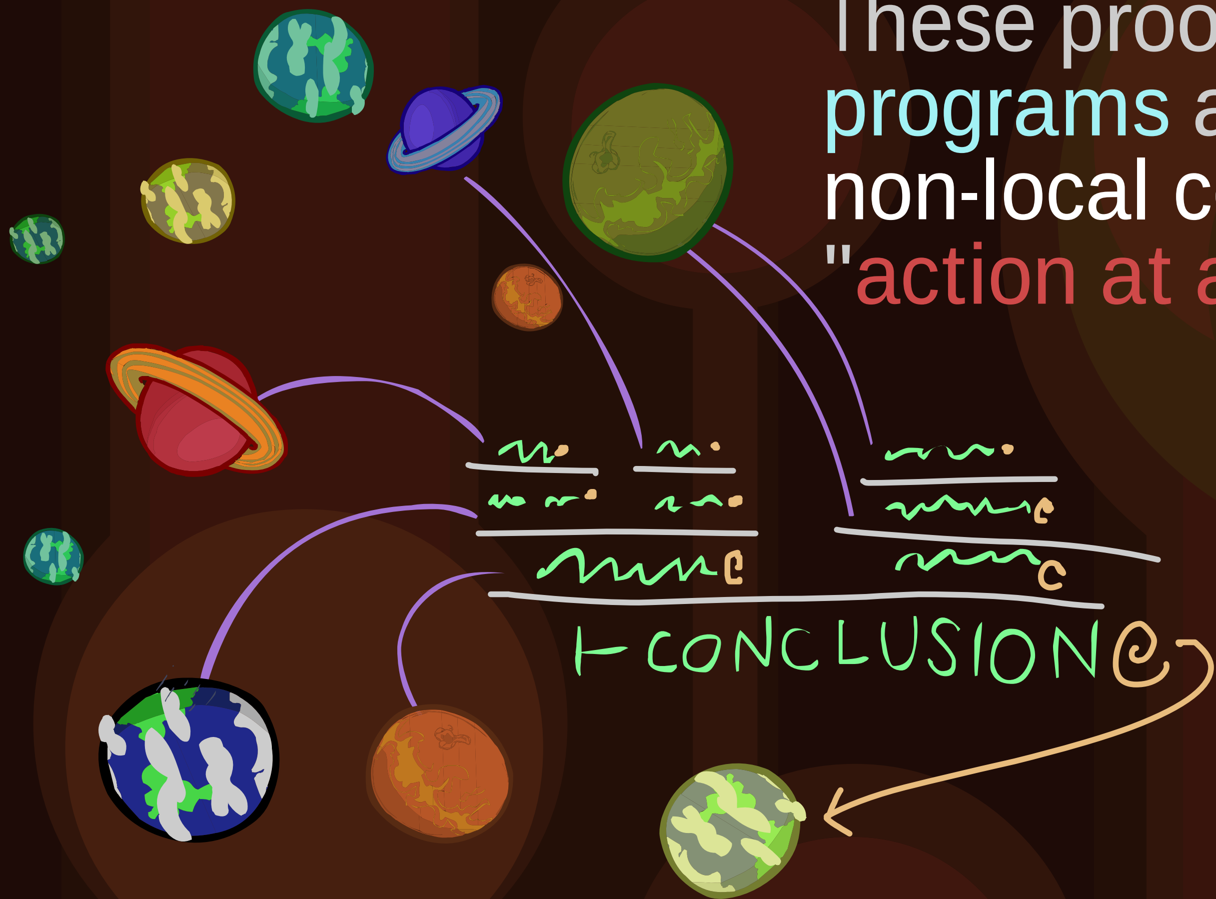
Modal logic

A proof in modal logic reasons from these distributed facts to produce a conclusion.

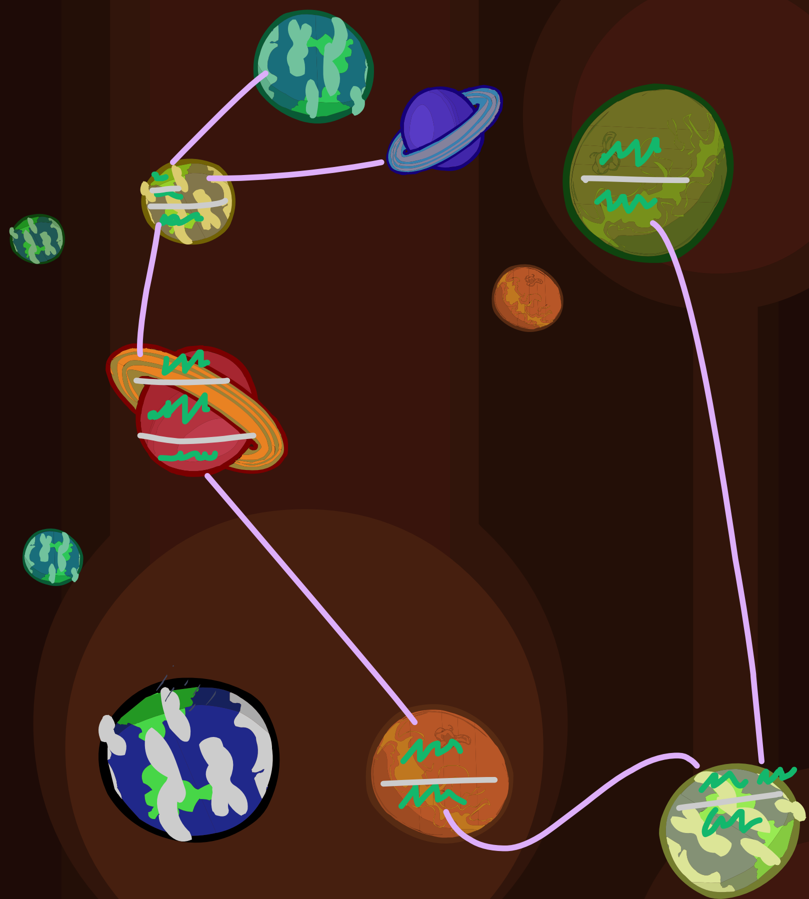


Modal logic

These proofs interpreted as programs appear to require non-local computation, or "action at a distance."



Lambda 5



A novel formulation of modal logic: Lambda 5

- reasoning (computation) is always local
- a single rule allows us to move facts (data) between worlds

"get"

This formulation of modal logic is:

- Logically faithful

  (Proved sound, complete, equivalent to known logics.)

Lambda 5

This formulation of modal logic is:

- **Logically faithful**
  (Proved sound, complete, equivalent to known logics.)

- **Computationally realistic**
  (Straightforward type-safe dynamic semantics.)

This formulation of modal logic is:

- **Logically faithful**
  (Proved sound, complete, equivalent to known logics.)

- **Computationally realistic**
  (Straightforward type-safe dynamic semantics.)

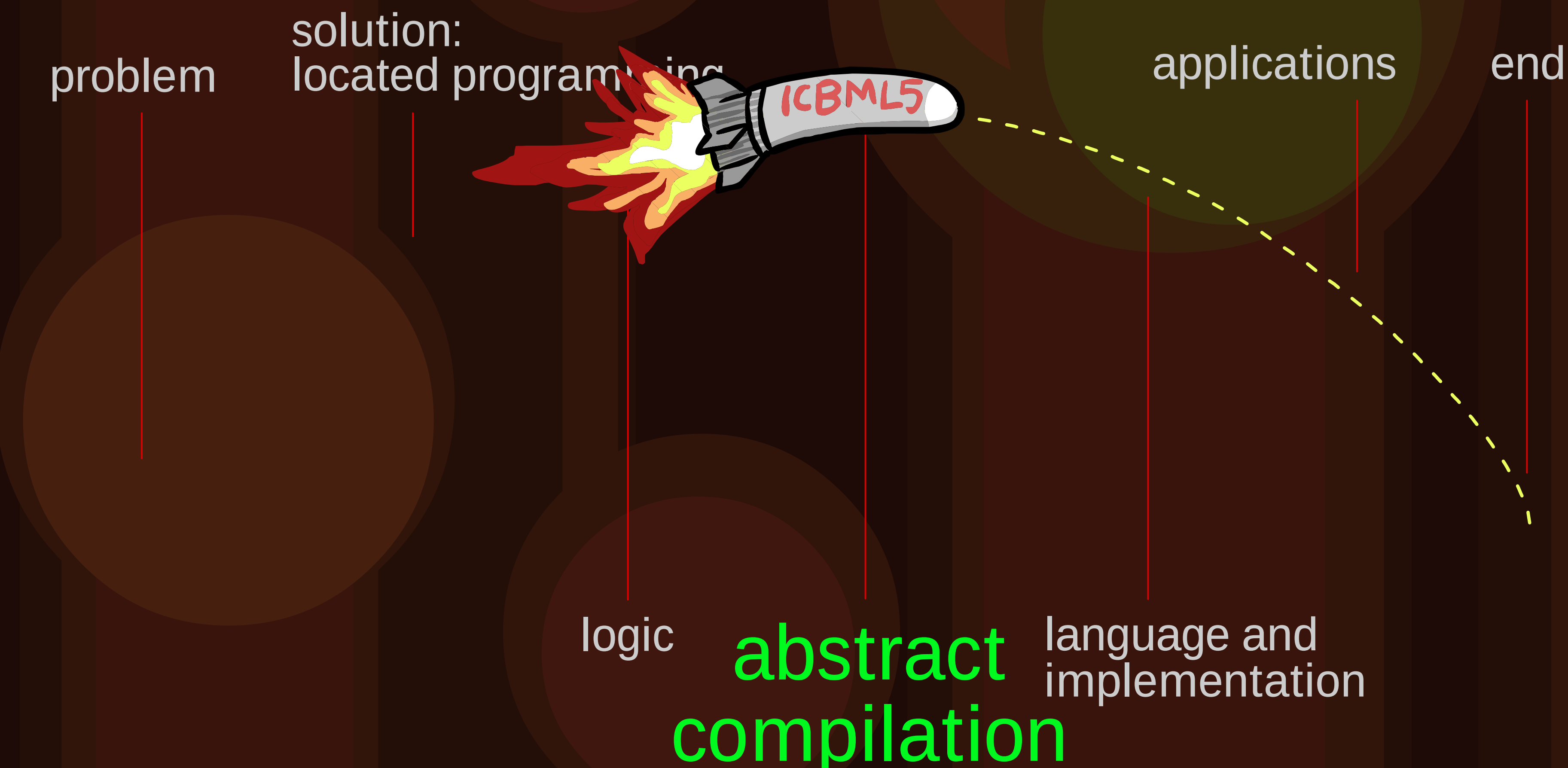
- **Not enough**
  (I study two extensions in detail: classical reasoning and global reasoning.)

Lambda 5

This formulation of modal logic is:

- **Logically faithful**
  (Proved sound, complete, equivalent to known logics.)
- **Computationally realistic**
  (Straightforward type-safe dynamic semantics.)
- **Not enough**
  (I study two extensions in detail: classical reasoning and global reasoning.)

[All proofs in Twelf]


problem
solution:
located program
ICBML5
applications
end
logic
abstract compilation
language and implementation

Abstract compilation

Next, I take the extended modal lambda calculus and carefully show how it can be compiled.

- Mini version of ML5

  (Leaves out the complications of a full-fledged language.)

Abstract compilation

Next, I take the extended modal lambda calculus and carefully show how it can be compiled.

- Mini version of ML5
  (Leaves out the complications of a full-fledged language.)
- Formalize several phases:
  - Elimination of syntactic sugar
  - Continuation passing style transformation
  - Closure conversion

Abstract compilation

Next, I take the extended modal lambda calculus and carefully show how it can be compiled.

- Mini version of ML5
- Formalize several phases
- Feedback of ideas into logic/language
  - ↳ Typed compilation is a good exercise of a language's expressiveness!

Abstract compilation

Next, I take the extended **modal lambda calculus** and carefully show how it can be compiled.

- Mini version of ML5
- Formalize several phases
- Feedback of ideas into logic/language
  - ↳ Typed compilation is a good exercise of a language's expressiveness!
- Prove static correctness for each phase

[All proofs in Twelf]

problem

solution:
located programming

logic

abstract
compilation

applications

end

language and implementation

ML5 is an ML-like programming language with a modal type system.

ML5 is an ML-like programming language with a modal type system.

Its implementation is specialized to web programming.

- Exactly two worlds: the browser ("home") and "server"
- AJAX-style applications (single page)

ML5

ML5 is an ML-like programming language with a **modal type system**.

Its implementation is specialized to web programming.

- Exactly two **worlds**: the browser ("home") and "server"
- AJAX-style applications (single page)
- A compiler (ML5/pgh)
- A runtime system including a web server

Modal type systems

A **type system** assigns a *type* to an expression, to *classify the values* it may produce.

Modal type systems

A type system assigns a type to an expression, to classify the values it may produce.

ML5's modal type system assigns a type and world to an expression, to classify the values it may produce and the location in which it may be evaluated.

Modal type systems

M : A

shape of value
that results

M : A @ w

where exp can
be evaluated

v : A

shape of value

v : A @ w

where value can
be used

Modal type systems

js.prompt "What is your name?" : string @ home

Returns a string and can only be evaluated on the web browser.

Modal type systems

```
js.prompt "What is your name?"   :  string @ home
```

Returns a string and can only be evaluated on the web browser.

```
db.lookup "name"   :  string @ server
```

Returns a string and can only be evaluated on the web server.

Local resources

Variables like js.prompt are the contingent (local) resources that form the context for type checking.

Local resources

Variables like js.prompt are the contingent (local) resources that form the context for type checking.

$$\frac{\text{js.prompt} : \textbf{string} \to \textbf{string} \mathbin{@} \text{client}, \ldots}{\vdash \text{js.prompt} : \textbf{string} \to \textbf{string} \mathbin{@} \text{client}}$$

Local resources

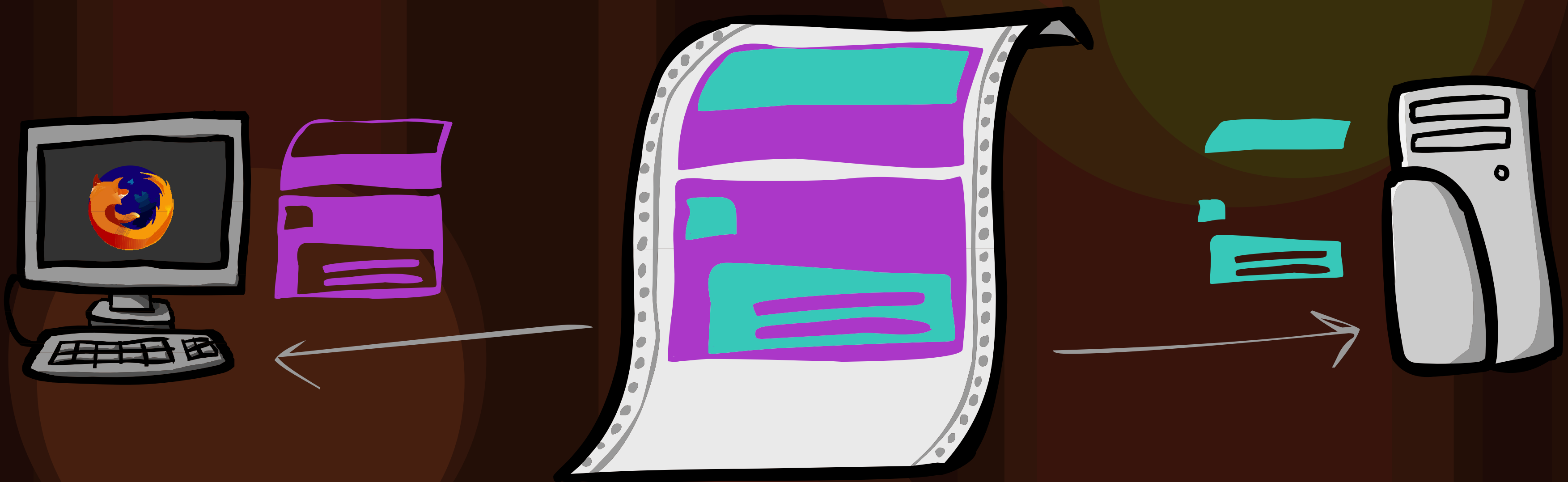
The programmer can declare a local resource by importing it at a name, type and world.

```
extern val js.prompt \@1: string -> string @ home
extern val js.alert \>1: string -> unit @ home

extern val db.lookup \>1: string -> string @ server
extern val version \>1: unit -> string @ server
```

ML5 model

ML5 source code includes parts for both the browser and server.



ML5 model

ML5 source code includes parts for both the browser and server.

ML5 model

Execution begins in the web browser.

ML5 model

Control may flow to the server and back during execution.

Get

This is done with the language construct from ... get ....

Get

This is done with the language construct from ... get ....

```
js.alert (from server get version());
```

Transfers control to server to evaluate expression.

Get

This is done with the language construct from ... get ....

```
js.alert (from server get version());
```

Transfers control to server to evaluate expression.

Get

This is done with the language construct from ... get ....

```
js.alert (from server get version());
```

Transfers control to server to evaluate expression.

Get

The get construct is (exclusively) how control and data flow between worlds.

Get

The get construct is (exclusively) how control and data flow between worlds.

$$\frac{\Gamma \vdash M : w'\ \mathrm{addr} @ w \qquad \Gamma \vdash N : A @ w' \qquad \text{+ 1 more premise...}}{\Gamma \vdash \mathrm{from}\ M\ \mathrm{get}\ N : A @ w}$$

Address of remote world (IP/port, etc.)

Expression to evaluate

Get

When we get, a value v : A @ w'
becomes a value v : A @ w

This only makes sense for certain types of values...

$$\frac{\Gamma \vdash M : w'\ \mathrm{addr}\ @\ w \qquad \Gamma \vdash N : A\ @\ w' \qquad \text{+ 1 more premise...}}{\Gamma \vdash \mathbf{from}\ M\ \mathbf{get}\ N : A\ @\ w}$$

Get

When we get, a value v : A @ w'
becomes a value v : A @ w

This only makes sense for certain types of values...

$$\frac{\Gamma \vdash M : w'\ \mathrm{addr}\ @\ w \qquad \Gamma \vdash N : A\ @\ w' \qquad A\ \mathrm{mobile}}{\Gamma \vdash \mathrm{from}\ M\ \mathrm{get}\ N : A\ @\ w}$$

Mobile types

A type is **mobile** if every value that inhabits it is portable.

$$\frac{}{\text{int mobile}}$$

$$\frac{}{\text{w addr mobile}}$$

$$\frac{\text{A mobile} \quad \text{B mobile}}{(\text{A} \times \text{B})\ \text{mobile}}$$

(ps: mobility has a logical justification)

Mobile types

A type is **mobile** if every value that inhabits it is portable.

$$\frac{}{\text{int mobile}}$$

$$\frac{}{\text{w addr mobile}}$$

$$\frac{\text{A mobile} \quad \text{B mobile}}{(A \times B)\ \text{mobile}}$$

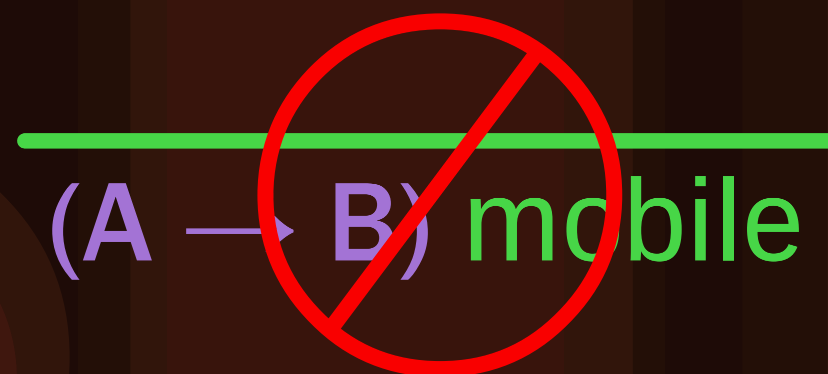
~~$$\frac{}{\text{file mobile}}$$~~

~~$$\frac{}{(A \to B)\ \text{mobile}}$$~~

Mobile types

```
(* string -> string @ client *)
from server get db.lookup
```

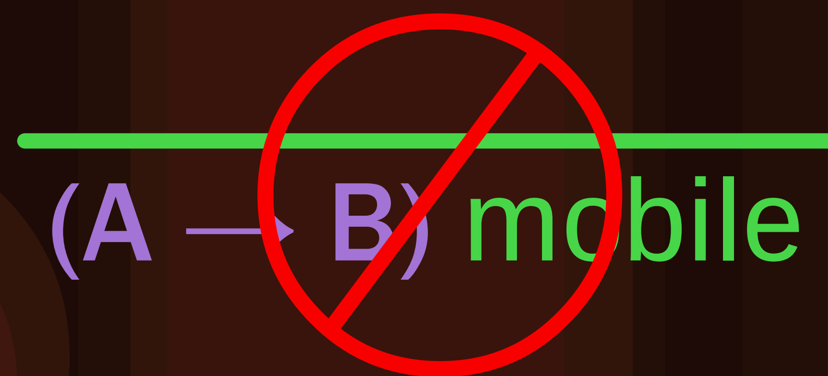
Would try to access a local database when called on the client!



Mobile types

```
(* string -> string @ client *)
from server get db.lookup
```

Would try to access a local database when called on the client!



(ML5 **statically excludes** such wrong-world accesses.)

Mobility vs. validity

Not *every* function value is portable, so function types are not mobile.

Mobility vs. validity

Not *every* function value is portable, so function types are not mobile.

(fn x ⇒ x)

However, some *particular* functions are portable. We have a way to demonstrate this in the type system: validity.

(ps: validity has a logical justification)

Validity

Valid hypotheses are bindings that can be used anywhere.

$$\frac{}{x \sim A \vdash x : A @ w}$$

Validity

Just as ML **type inference** automatically makes definitions maximally polymorphic, ML5 type inference makes definitions maximally valid:

```
(* map ~ ('a -> 'b) -> 'a list -> 'b list *)
fun \@1map f nil = nil
  |\>1map f (h :: t) = (f h) :: map f t
```

Libraries

Validity

$$\frac{\Gamma, \omega' \text{ world} \vdash v : A \mathbin{@} \omega' \qquad \Gamma, x \sim A \vdash N : C \mathbin{@} w}{\Gamma \vdash \texttt{let val } x = v \texttt{ in } N : C \mathbin{@} w}$$

To validate a binding, hypothesize the existence of a world ω'. If the value is well-typed there, then it would be well-typed anywhere, since we know nothing about ω'.

Validity

$$\frac{\dfrac{\Gamma, \omega' \text{ world}, x : \text{int} @ \omega' \vdash x : \text{int} @ \omega'}{\Gamma, \omega' \text{ world} \vdash \text{fn } x \Rightarrow x : \text{int} \rightarrow \text{int} @ \omega'} \quad \ldots}{\Gamma \vdash \text{let val } x = (\text{fn } x \Rightarrow x) \text{ in } \ldots : C @ w}$$

Validity

$$\dfrac{\dfrac{\Gamma, \omega' \text{ world}, x : \text{int} @ \omega' \vdash x : \text{int} @ \omega'}{\Gamma, \omega' \text{ world} \vdash \text{fn } x \Rightarrow x : \text{int} \to \text{int} @ \omega' \quad \ldots}}{\Gamma \vdash \text{let val } x = (\text{fn } x \Rightarrow x) \text{ in } \ldots : C @ w}$$

Note: values only! (*cf*. ML value restriction)

```
(* r : int ref @ client *)
val r = ref 0
```

Modalities

The **judgments** x ~ A and x : A @ w allow us to define new types that encapsulate the notions of validity and locality.

Modalities

The **judgments** x ~ A and x : A @ w allow us to define new types that encapsulate the notions of validity and locality.

 A — A valid value of type A.

A at w — An encapsulated value of type A that can be used only at w.

(Can also have as derived forms: □A ◇A)

Modalities

These are all mobile no matter what A is.

⌘ A — A valid value of type A.

A at w — An encapsulated value of type A that can be used only at w.

(Can also have as derived forms: $\Box$A $\Diamond$A)

ML-like features

ML5 has most of the features of core SML.

- algebraic datatypes, extensible types
- pattern matching
- mutable references
- exceptions
- mutual recursion

ML-like features

ML5 has most of the features of core SML.

- algebraic datatypes, extensible types
- pattern matching
- mutable references
- exceptions
- mutual recursion

... and some extensions:

- first-class continuations, threads
- quote/antiquote

ML-like features

Most features behave as they do in SML.

We usually just need to consider whether a given type should be mobile.

```
datatype (a, b) t =
    First of a * int
  | Second of (b at home) * t
```

The type (t_1, t_2) t is mobile if both arms (with t_1, t_2 filled in) carry mobile types.

ML-like features

Most features behave as they do in SML.

We usually just need to consider whether a given type should be mobile.

```
datatype (a, b) t' =
    First of a * int
  | Second of (b at home) * t'
  | Third of a → b
```

The type $(t_1, t_2)\ t$ is mobile if both arms (with t_1, t_2 filled in) carry mobile types.

ML-like features

The exn type and other extensible types are always mobile.

```
exception TagA of int
exception TagB of unit -> unit

(* ! *)
do case (from server get e) : exn of
        \@1TagA _ => ()
      | \>1TagB f => f ()
```

ML-like features

The exn type and other extensible types are always mobile.

```
exception TagA of int
exception TagB of unit -> unit

(* ! *)
do case (from server get e) : exn of
        \@1TagA _ => ()
      | \>1TagB f => f ()
```

The extensible type tags give permission to retrieve the stored value.

ML-like features

The exn type and other extensible types are always mobile.

```
vexception TagA of int          \@3(* valid *)
exception TagB of unit -> unit     \>3(* can't be valid *)

(* ! *)
do case (from server get e) : exn of
       \@1TagA _ => ()
     | \>1TagB f => f ()
```

The extensible type tags give permission to retrieve the stored value.

Put

Another construct put can evaluate an expression and validate the resulting binding, but only if its type is mobile.

$$\frac{\Gamma \vdash M : A \mathbin{@} w \qquad A \text{ mobile} \qquad \Gamma, x \sim A \vdash N : C \mathbin{@} w}{\Gamma \vdash \text{let put } x = M \text{ in } N : C \mathbin{@} w}$$

(no communication)

Example: proxy

```
let
   \@1extern val db.lookup : string -> string @ server

   \>1(* plookup ~ string -> string *)
   \>1fun plookup s =
   \>1   \@2let \@3put s' = s
         \>2in \>3from server get (db.lookup s')
         \>2end
in
\>1(* ... *)
end
```

Ok.

Implementation

The ML5 implementation consists of a compiler, and a web server that hosts and runs the server part of programs.

Compilation

The ML5/pgh compiler transforms the source program into client-side JavaScript and server-side bytecode.

- Elaboration and type inference
- CPS conversion
- Type and world representation
- Closure conversion
- Code generation

} type directed

CPS conversion

CPS conversion allows us to support first-class continuations and threads.

from ... get ... replaced with to ... go ... :

```
k (from server
   get e)
```

CPS conversion

CPS conversion allows us to support first-class continuations and threads.

from ... get ... replaced with to ... go ... :

```
k (from server
   get e)
```

becomes

```
put back = localhost ()
(to server
 go put ret = e
     (to back
      go k(ret)))
```

Type and world representation

Marshaling uses type and world information at run-time, so we must represent these as data.

$$\alpha \text{ type}, \omega \text{ world}, \ldots \vdash A \mathbin{@} w$$

Type and world representation

Marshaling uses type and world information at run-time, so we must represent these as data.

$$\alpha \text{ type}, \omega \text{ world}, \ldots \vdash A @ w$$

⤵

$$\alpha \text{ type}, u_\alpha \sim \alpha \text{ rep}, \omega \text{ world}, u_\omega \sim \omega \text{ rep}, \ldots \vdash A @ w$$

Closure conversion

Closure conversion explicitly constructs closures so that we can label each piece of code.

This means abstracting over any free variables:

$$x : A \mathbin{@} w_1,\ u \sim B \vdash C \to D \mathbin{@} w_2$$

Closure conversion

Closure conversion explicitly constructs closures so that we can label each piece of code.

This means abstracting over any free variables:

$$x : A \mathbin{@} w_1,\ u \sim B \vdash C \to D \mathbin{@} w_2$$

$$\cdot \vdash (C \times A\ \underline{\text{at}}\ w_1 \times \text{⌘} B) \to D \mathbin{@} w_2$$

modalities internalize judgments

Code generation

For each piece of closed code, we use its world to decide what code we must generate for it.

- @ server - generate bytecode
- @ client - generate javascript
- @ ω - generate both (polymorphic)

Typing guarantees that code @ server will only use server resources.

Runtime

The runtime system:

- Web server delivers code, starts session
- Runs server code, database, etc.
- Marshaling and maintaining communication
- Thread scheduling, event handling

I'll mention these in the demo.

problem

solution:
located programming

logic

abstract
compilation

language and
implementation

applications

end

Applications

Built realistic applications with ML5.

- Evaluate its practicality, expressiveness
- Discover performance bottlenecks
- Missing features
- Feedback of ideas into language, compiler

Demo

Time!

problem

solution:
located programming

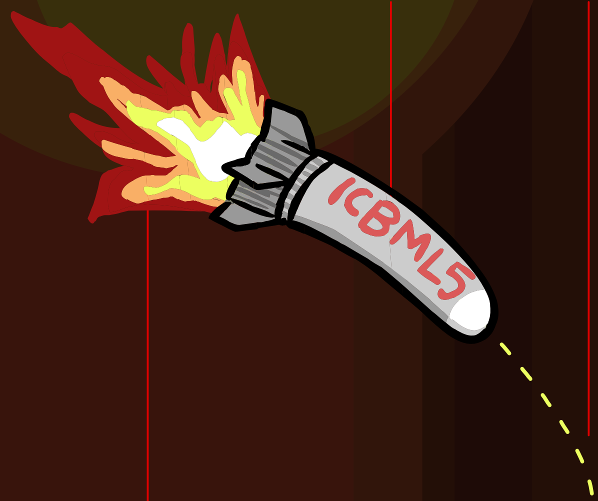
logic

abstract
compilation

language and
implementation

applications

end



Conclusion

In conclusion,

Conclusion

In conclusion,

> Modal type systems provide an elegant and practical means for controlling local resources in spatially distributed computer programs.

Conclusion

In conclusion,

- New programming language for spatially distributed computing.

Conclusion

In conclusion,

- New programming language for spatially distributed computing.
  - Express locality of resources
  - Statically-typed, higher order programming

Conclusion

In conclusion,

- New programming language for spatially distributed computing.
  - Express locality of resources
  - Statically-typed, higher order programming
- Based on novel formulation of modal logic.

Conclusion

In conclusion,

- New programming language for spatially distributed computing.
  - Express locality of resources
  - Statically-typed, higher order programming
- Based on novel formulation of modal logic.
- Mechanized theory and usable implementation.

Dankon

Thanks! Questions?

Bonus topics: security tierless

Dankon

Thanks! Questions?

Bonus topics: security tierless

Dankon

Thanks! Questions?

Bonus topics: security tierless

Dankon

Thanks! Questions?

Bonus topics: security tierless

Dankon

Thanks! Questions?

Bonus topics: security tierless

Dankon

Thanks! Questions?

Bonus topics: security tierless

Dankon

Thanks! Questions?

Bonus topics: security tierless

Dankon

Thanks! Questions?

Bonus topics: security tierless

Security

Security is a difficult problem in the presence of uncooperative participants: We have no real control over what the client does with his Javascript.

Security

Compilation obscures some security issues.

```
let
   extern format : unit -> unit @ server
   val password = "my_cool_password"
   put input = js.prompt ("password?")
in
   from server get
     if input = password
     then (\@1from client get js.alert ("Formatting...");
              \>1format ())
     else ()
end
```

Security

Compilation obscures some security issues.

```
let
   extern format : unit -> unit @ server
   val password = "my_cool_password"
   put input = js.prompt ("password?")
in
   from server get
     if input = password
     then (\@1from client get js.alert ("Formatting...");
              \>1format ())
     else ()
end
```

Does client source contain "my_cool_password"?

Security

Compilation obscures some security issues.

```
let
  extern format : unit -> unit @ server
  val password = "my_cool_password"
  put input = js.prompt ("password?")
in
  from server get
    if input = password
    then (\@1from client get js.alert ("Formatting...");
              \>1format ())
    else ()
end
```

server entry point 1

server entry point 2

Security

Types can help...

```
let
   extern format : unit -> unit @ server
   val password : string @ server = "my_cool_password"
   put input = js.prompt ("password?")
in
   from server get
     if input = password
     then (\@1from client get js.alert ("Formatting...");
              \>1format ())
     else ()
end
```

Tierless programming

Links programming language (Wadler et al.)
- built-in notion of "client" and "server" (only)
  - tied to function calls
  - marshaling can fail at runtime

Hop (Serrano et al.)
- based on scheme (just one type)
  - no static checks
- two gets, specialized to client/server

ML5 or bust

Twelf code, implementation, dissertation at

http://tom7.org/ml5/

ML5 or bust

Twelf code, implementation, dissertation at

http://tom7.org/ml5/

ML5 or bust

Twelf code, implementation, dissertation at

http://tom7.org/ml5/

Addresses

A host can compute its address with localhost.

Addresses

A host can compute its address with localhost.

$$\frac{}{\Gamma \vdash \text{localhost}() : w\ \text{addr} \ @\ w}$$

Addresses

A host can compute its address with localhost.

$$\frac{}{\Gamma \vdash \text{localhost}() : w\ \text{addr} @ w}$$

For now assume we have two worlds client and server and variables in context:

client : client addr @ server

server : server addr @ client

Addresses

client : client addr @ server
server : server addr @ client ⊢

```
from server get
  (\@1db.update ("greeting", "hello");
   \>1from client get
   \>1    js.alert "greeting updated!")
```