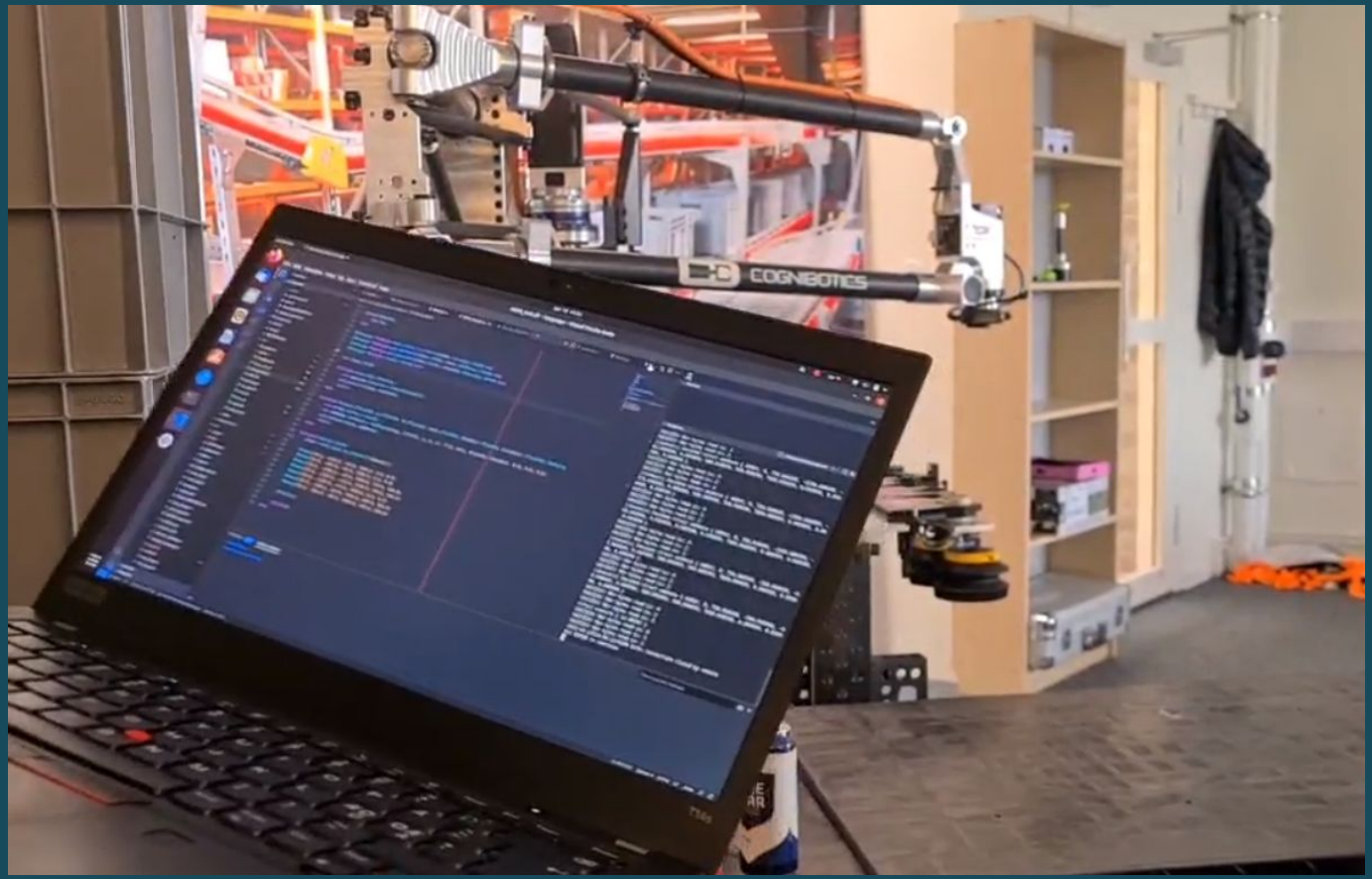
COGNIBOTICS

Romeo & Juliet:
Next-gen Robot Programming

Jesper Öqvist

COGNIBOTICS

COGNIBOTICS



COGNIBOTICS

COGNIBOTICS

Julia

Dynamic language for scientific computing:

- ★ Managed memory (GC)
- ★ Minimal syntax
- ★ Powerful libraries
- ★ JIT-compiled (LLVM)
- ★ Optional type declarations
- ★ First-class functions
- ★ Multimethods

COGNIBOTICS

Julia Syntax

```
f(x) = 2x
```

```
function mul2(x::Int)::Int
  x + x
end
```

```
for i in [1,2,3]
  println(mul2(i))
end
```

COGNIBOTICS

Data Types

```
struct Foo{T}
  x::T
  y::String
  z::Vector{T}
end
```

No member functions - instead polymorphic multimethods

COGNIBOTICS

Multimethods

```
struct Bird end
struct Fly end


can(::Fly,  ::Bird) = "fly cannot bird"
can(::Bird, ::Fly ) = "bird can fly"
can(::Any,  ::Any ) = "unknown"


can(Fly(), Bird())
can(Bird(), Fly())
```

Multimethods / Multiple Dispatch - runtime selection of most specific method (on all aruments).

COGNIBOTICS

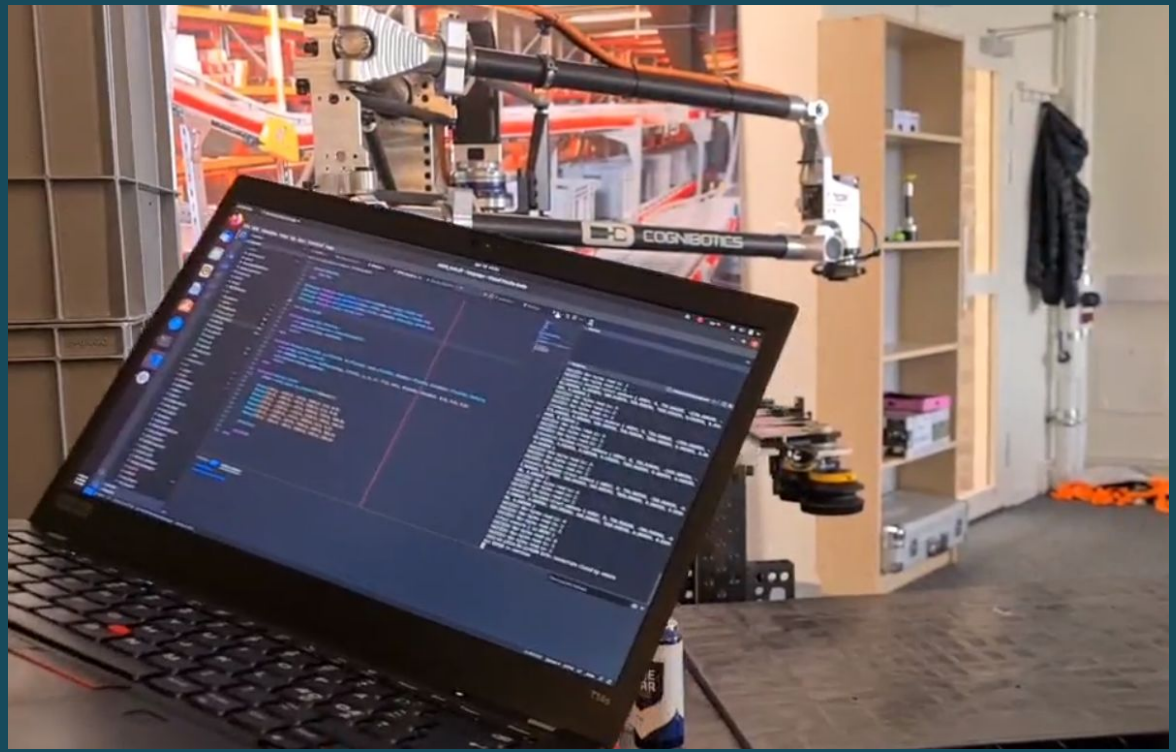
Polymorphism

```
struct Foo{T}
  x::T
end


unwrap(foo::Foo{T}) where T = foo.x



setindex!(arr::Array{T,D}, x::T, i::Int) where {T, D} = ...
```

COGNIBOTICS

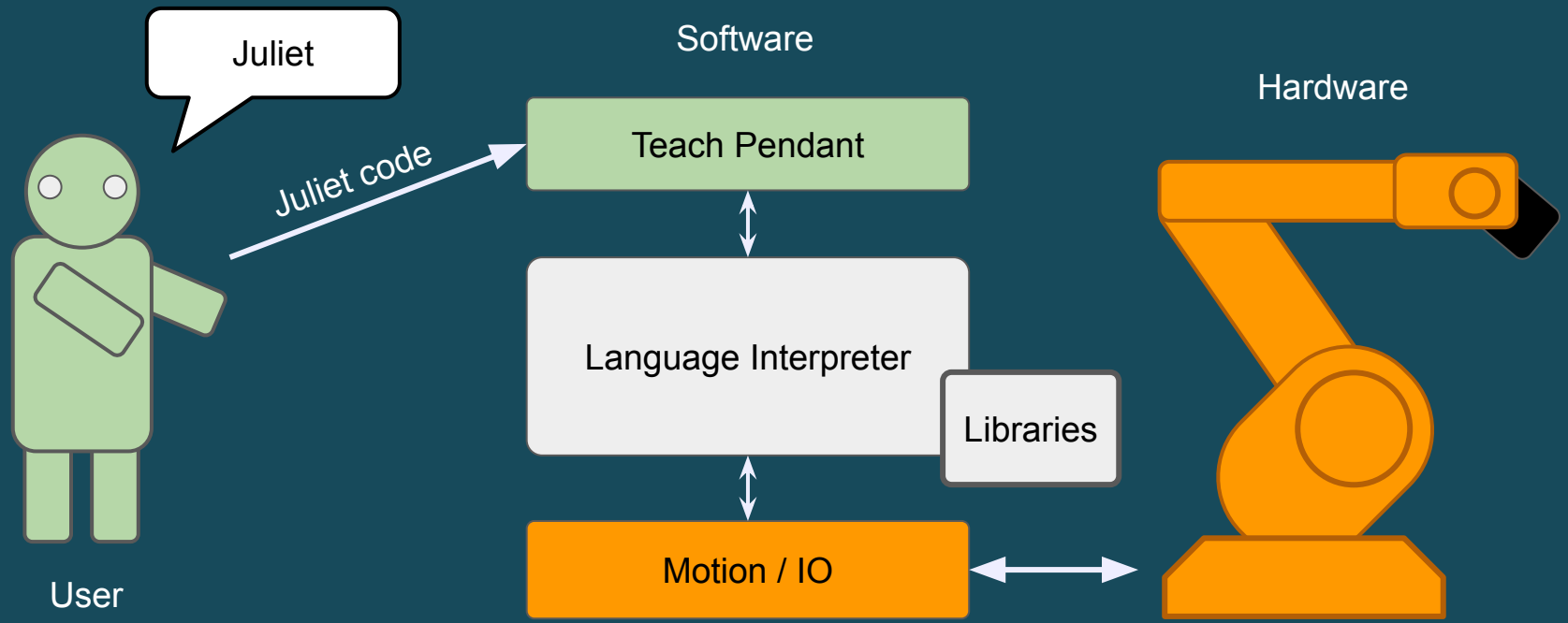
Juliet = Julia for Robots

- Statically typed Julia
- Compiled to bytecode
- Custom GC
- Robot VM (Romeo/RVM)
- IDE for robot programming



COGNIBOTICS

Highlevel System View



COGNIBOTICS

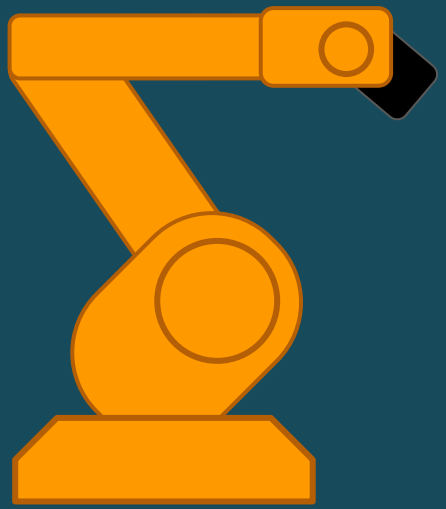
Robot Considerations

Slow predictable execution > fast unpredictable JIT

Static typing > runtime errors

Static allocations > dynamic arrays/lists

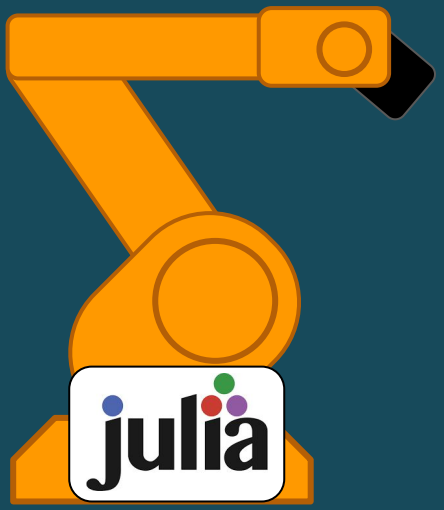
Robot tasks:
Welding
Painting
Machining
Pick/place



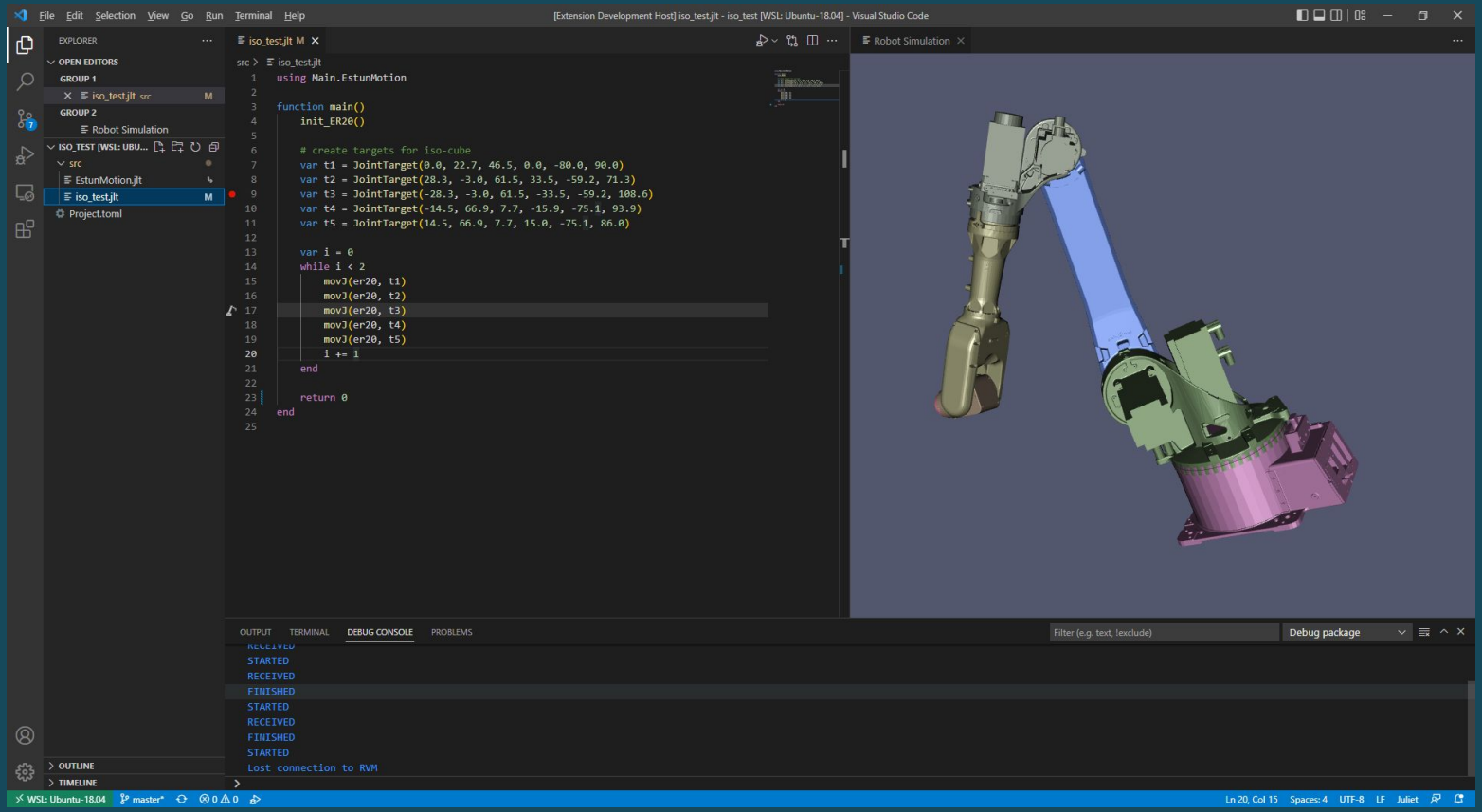
COGNIBOTICS

Why Juliet for Robots?

- **intuitive / minimal syntax**
- **powerful type system**
- **robot motions as library code**
- functional programming
- formalize backwards-stepping behavior
- access to high-quality libraries (Julia packages)



COGNIBOTICS



COGNIBOTICS

The Juliet Team

About 15 people contributing to Juliet runtime, compiler, GC, libraries, etc.

2 people working on the compiler:
Jesper Öqvist
Erik Jansson

COGNIBOTICS

Development Process

Incremental development
Divide and conquer, don't worry about making it perfect from the start.

Regression test!
Each new feature should be tested.

CI/CD
Run tests on a server for each commit (GitLab)

COGNIBOTICS

Juliet Compiler

- Written in C
- Scanner: Flex-generated
- Parser: Bison-generated (GLR)
- Semantic analysis: visitor-based
- Code generation: bytecode

COGNIBOTICS

Juliet Parsing

Julia is not context-free LR(1) because it allows almost any kind of whitespace anywhere

```
if x println(x) end

if x
  println(x)
end
```

COGNIBOTICS

Juliet Parsing

Julia is not context-free LR(1) because it allows almost any kind of whitespace anywhere

```
if x(y, z)
end


if x
  (y, z)
end

function f(x)
  return x
  + 3
end
```

COGNIBOTICS

Julia Parsing

Julia is not context-free LR(1) because it allows almost any kind of whitespace anywhere

```
function f(x)
  return x
  + 3
end
```

COGNIBOTICS

Julia is not LR(1)

```
program = es

// Expression list:
es = e | es w e

// Expression:
e  = n | n w EQ e

// Name:
n = ID

// Whitespace:
w = WHITESPACE
```

COGNIBOTICS

Juliet Modules

```
module Foo
  function foo() end
end

Main.Foo.foo()
```

Name analysis passes:

1. Find all modules and structs. Map structs to respective modules.
2. Declare all function names (in modules), lookup parameter types.
3. Do local name lookup and call resolution.
4. Do a type check pass.

COGNIBOTICS

Juliet Type Analysis

The subtype algorithm is described in Jeff Bezanson's Thesis.

Just implement that ->

$$\frac{{}^{B}_{A}X^{L}, \Gamma \vdash T \le S}{\Gamma \vdash \exists\, {}^{B}_{A}X\ T \le S} \qquad \frac{{}^{B}_{A}X^{R}, \Gamma \vdash T \le S}{\Gamma \vdash T \le \exists\, {}^{B}_{A}X\ S} \qquad \frac{}{\Gamma \vdash X \le X}$$

$$\frac{{}^{B}X^{L}, {}_{A}Y^{L}, \Gamma \vdash B \le Y \ \vee\ X \le A}{{}^{B}X^{L}, {}_{A}Y^{L}, \Gamma \vdash X \le Y} \qquad \frac{{}^{B}_{A}X^{R}, Y^{R}, \Gamma \vdash B \le A}{{}^{B}_{A}X^{R}, Y^{R}, \Gamma \vdash X \le Y}$$

$$\frac{{}^{B}_{A}X^{R}, \Gamma \vdash T \le B}{{}^{B}_{A\cup T}X^{R}, \Gamma \vdash T \le X} \qquad \frac{{}^{B}_{A}X^{R}, \Gamma \vdash A \le T}{{}^{T}_{A}X^{R}, \Gamma \vdash X \le T}$$

$$\frac{{}_{A}X^{L}, \Gamma \vdash T \le A}{{}_{A}X^{L}, \Gamma \vdash T \le X} \qquad \frac{{}^{B}X^{L}, \Gamma \vdash B \le T}{{}^{B}X^{L}, \Gamma \vdash X \le T}$$

Bezanson, J. W. *Abstraction in Technical Computing*. PhD thesis, Department of Electrical Engineering and Computer
Science, Massachusetts Institute of Technology. June, 2015.

COGNIBOTICS

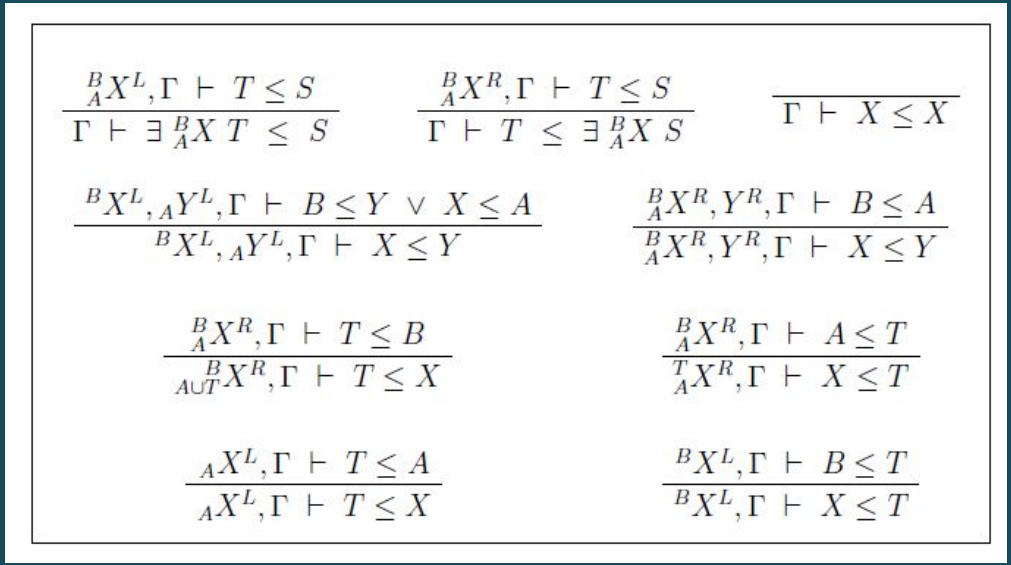
Juliet Type Analysis

The subtype algorithm is described in Jeff Bezanson's Thesis.

Just implement that ->

Actually it's not that simple. Must avoid infinite expanding types. Must be precise with keeping track of separate instances of typenames.

```
struct Onion{A, B}
  onion::Onion{B, A}
end
```



$$\frac{{}^{B}_{A}X^{L}, \Gamma \vdash T \leq S}{\Gamma \vdash \exists\, {}^{B}_{A}X\ T \leq S} \qquad \frac{{}^{B}_{A}X^{R}, \Gamma \vdash T \leq S}{\Gamma \vdash T \leq \exists\, {}^{B}_{A}X\ S} \qquad \frac{}{\Gamma \vdash X \leq X}$$

$$\frac{{}^{B}X^{L}, {}_{A}Y^{L}, \Gamma \vdash B \leq Y \vee X \leq A}{{}^{B}X^{L}, {}_{A}Y^{L}, \Gamma \vdash X \leq Y} \qquad \frac{{}^{B}_{A}X^{R}, Y^{R}, \Gamma \vdash B \leq A}{{}^{B}_{A}X^{R}, Y^{R}, \Gamma \vdash X \leq Y}$$

$$\frac{{}^{B}_{A}X^{R}, \Gamma \vdash T \leq B}{{}^{B}_{A \cup T}X^{R}, \Gamma \vdash T \leq X} \qquad \frac{{}^{B}_{A}X^{R}, \Gamma \vdash A \leq T}{{}^{T}_{A}X^{R}, \Gamma \vdash X \leq T}$$

$$\frac{{}_{A}X^{L}, \Gamma \vdash T \leq A}{{}_{A}X^{L}, \Gamma \vdash T \leq X} \qquad \frac{{}^{B}X^{L}, \Gamma \vdash B \leq T}{{}^{B}X^{L}, \Gamma \vdash X \leq T}$$

Bezanson, J. W. *Abstraction in Technical Computing*. PhD thesis, Department of Electrical Engineering and Computer
Science, Massachusetts Institute of Technology. June, 2015.

COGNIBOTICS

Arrays

Array indexing is translated to function calls:

```
var a = zeros(Int8, 10)
a[4] = 10                   # setindex!(a, 10, 4)
println(a[4])               # getindex(a, 4)
```

Multimethods make it possible to overload the indexing operator for custom types!l

COGNIBOTICS

Exceptions

```
try
  error("oopsie")
catch e
  println("failed")
finally
  return 0
end
```

COGNIBOTICS

Union Dispatch

Formal parameters can have union type:

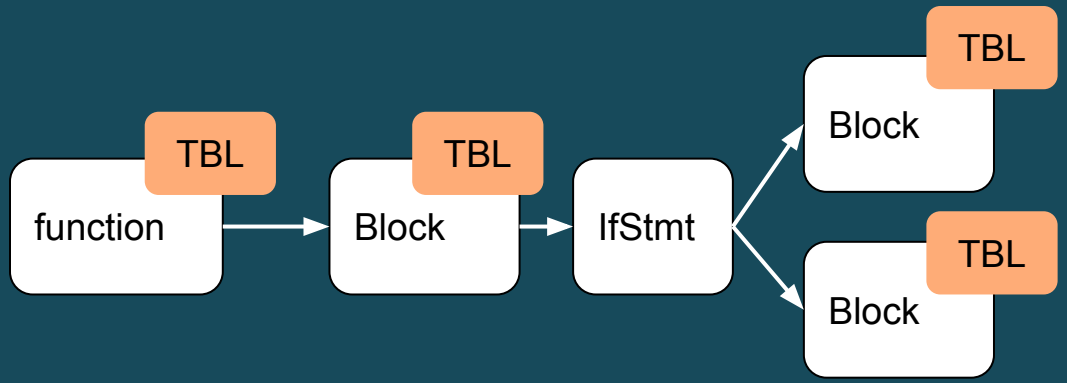
```
function foo(x :: Union{Bool, String}) end
foo(true)
foo("x")
```

Conversely, multiple functions can be dispatched with a union typed argument.

COGNIBOTICS

Declaration Chains

The typical design for name analysis with visitors uses name tables at each scope (function, block, etc).



COGNIBOTICS

Declaration Chains

Idea: instead store chain of previous declarations from each name use, mapping out the reverse pre-order of matching name declarations.