1 Why Julia is fast

2 Why Julia ~~is fast~~ can be fast

David P. Sanders

Department of Physics, Faculty of Sciences
National University of Mexico (UNAM)

2.1 What kind of language is Julia

- Julia **feels like** an interpreted scripting language
- But it is very different: everything is compiled, via LLVM
- This is **by design**: best of both worlds
- Compilation is automatic, in the moment, (almost) without you realising

3 What happens when call a function?

Let's define a simple function:

In [8]:
```
f(x,y) = x + y
```

```
WARNING: Method definition f(Any, Any) in module Main at In[7]:1 overwritten at In[8]:1.
```

Out[8]:
```
f (generic function with 1 method)
```

3 What happens when call a function?

Let's define a simple function:

In [8]:
```
f(x,y) = x + y
```

```
WARNING: Method definition f(Any, Any) in module Main at In[7]:1 overwri
tten at In[8]:1.
```

Out[8]:
```
f (generic function with 1 method)
```

And call it:

In [ ]:
```
f(3, 4)
```

3 What happens when call a function?

Let's define a simple function:

```
In [8]: f(x,y) = x + y
```

```
WARNING: Method definition f(Any, Any) in module Main at In[7]:1 overwritten at In[8]:1.
```

```
Out[8]: f (generic function with 1 method)
```

And call it:

```
In [9]: f(3, 4)

Out[9]: 7
```

What did Julia do? We can ask it (i.e. **introspect**) by looking inside the main steps in the process.

Step 1: "lowering"

- First, the code is converted into a lower-level representation;
- For a simple function like this, there is nothing to do:

```
In [*]: @code_lowered f(3, 4)
```

What did Julia do? We can ask it (i.e. **introspect**) by looking inside the main steps in the process.

Step 1: "lowering"

- First, the code is converted into a lower-level representation;
- For a simple function like this, there is nothing to do:

```
In [10]: @code_lowered f(3, 4)

Out[10]: LambdaInfo template for f(x, y) at In[8]:1
         :(begin
                 nothing
                 return x + y
             end)
```

?

Step 2: Type inference

Next, Julia runs **type inference** to work out which type everything has:

```
In [11]: @code_typed f(3, 4)

Out[11]: LambdaInfo for f(::Int64, ::Int64)
         :(begin
                 return (Base.box)(Int64,(Base.add_int)(x,y))
             end::Int64)
```

Julia has **inferred** (automatically worked out) the types.

Step 3: LLVM intermediate representation (IR)

Next, Julia generates an "intermediate representation" (IR) used by LLVM:

In [12]:
```
@code_llvm f(3, 4)
```

```
define i64 @julia_f_71393(i64, i64) #0 {
top:
  %2 = add i64 %1, %0
  ret i64 %2
}
```

Step 3: LLVM intermediate representation (IR)

Next, Julia generates an "intermediate representation" (IR) used by LLVM:

```
In [12]: @code_llvm f(3, 4)
```

```
define i64 @julia_f_71393(i64, i64) #0 {
top:
  %2 = add i64 %1, %0
  ret i64 %2
}
```

Step 3: LLVM intermediate representation (IR)

Next, Julia generates an "intermediate representation" (IR) used by LLVM:

In [12]:
```
@code_llvm f(3, 4)
```

```
define i64 @julia_f_71393(i64, i64) #0 {
top:
  %2 = add i64 %1, %0
  ret i64 %2
}
```

Step 3: LLVM intermediate representation (IR)

Next, Julia generates an "intermediate representation" (IR) used by LLVM:

In [12]:
```
@code_llvm f(3, 4)
```

```
define i64 @julia_f_71393(i64, i64) #0 {
top:
  %2 = add i64 %1, %0
  ret i64 %2
}
```

Step 4: Native machine code

Finally, LLVM compiles this to **native machine code** for your machine:

```
In [ ]: @code_native f(3, 4)
```

The @ means a **macro** -- a "super-function" that processes code expressions.

Step 4: Native machine code

Finally, LLVM compiles this to **native machine code** for your machine:

In [13]:
```
@code_native f(3, 4)
```

```
	.section	__TEXT,__text,regular,pure_instructions
Filename: In[8]
	pushq	%rbp
	movq	%rsp, %rbp
Source line: 1
	leaq	(%rdi,%rsi), %rax
	popq	%rbp
	retq
	nopw	(%rax,%rax)
```

4 Julia generates specialized code

What happens if we use arguments of a different type?

Let's try passing arguments of different types to the function f:

```
In [ ]: f(3.5, 4)
```

?

4 Julia generates specialized code

What happens if we use arguments of a different type?

Let's try passing arguments of different types to the function `f`:

In [14]:
```
f(3.5, 4)
```

Out[14]:
```
7.5
```

In [15]:
```
@which 3.5 + 4
```

Out[15]: +(x::Number, y::Number) at promotion.jl:190

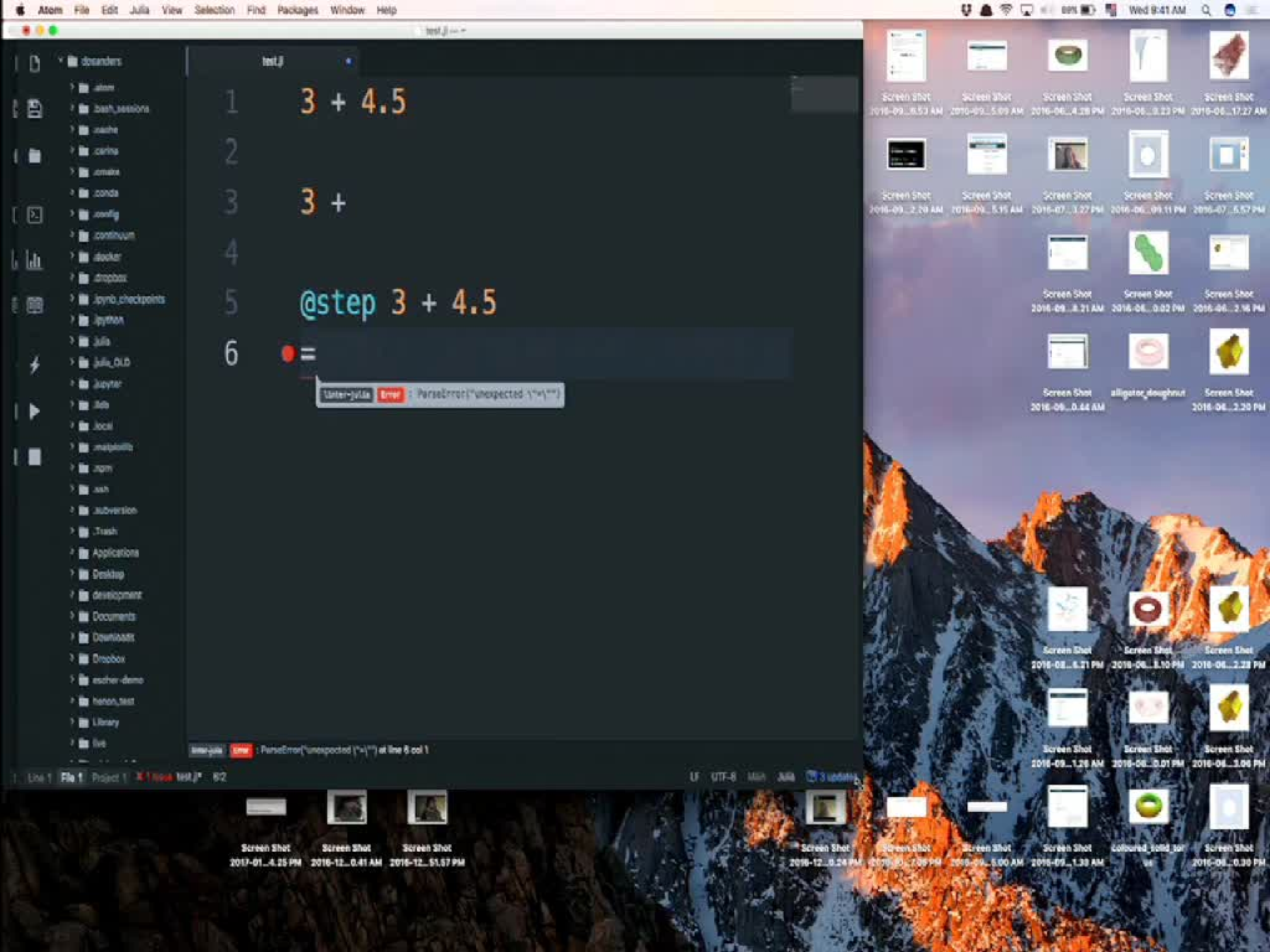
```
function promote_result{T<:Number,S<:Number}(::Type{T},::Type{S},::Type{Bottom},::Type{Bottom})
    @_pure_meta
    promote_to_supertype(T, S, typejoin(T,S))
end

# promote numeric types T and S to typejoin(T,S) if T<:S or S<:T
# for example this makes promote_type(Integer,Real) == Real without
# promoting arbitrary pairs of numeric types to Number.
promote_to_supertype{T<:Number           }(::Type{T}, ::Type{T}, ::Type{T}) = (@_pure_meta; T)
promote_to_supertype{T<:Number,S<:Number}(::Type{T}, ::Type{S}, ::Type{T}) = (@_pure_meta; T)
promote_to_supertype{T<:Number,S<:Number}(::Type{T}, ::Type{S}, ::Type{S}) = (@_pure_meta; S)
promote_to_supertype{T<:Number,S<:Number}(::Type{T}, ::Type{S}, ::Type) =
    error("no promotion exists for ", T, " and ", S)

+(x::Number, y::Number) = +(promote(x,y)...)
*(x::Number, y::Number) = *(promote(x,y)...)
-(x::Number, y::Number) = -(promote(x,y)...)
/(x::Number, y::Number) = /(promote(x,y)...)
^(x::Number, y::Number) = ^(promote(x,y)...)

fma(x::Number, y::Number, z::Number) = fma(promote(x,y,z)...)
muladd(x::Number, y::Number, z::Number) = muladd(promote(x,y,z)...)

(&)(x::Integer, y::Integer) = (&)(promote(x,y)...)
(|)(x::Integer, y::Integer) = (|)(promote(x,y)...)
($)(x::Integer, y::Integer) = ($)(promote(x,y)...)
```

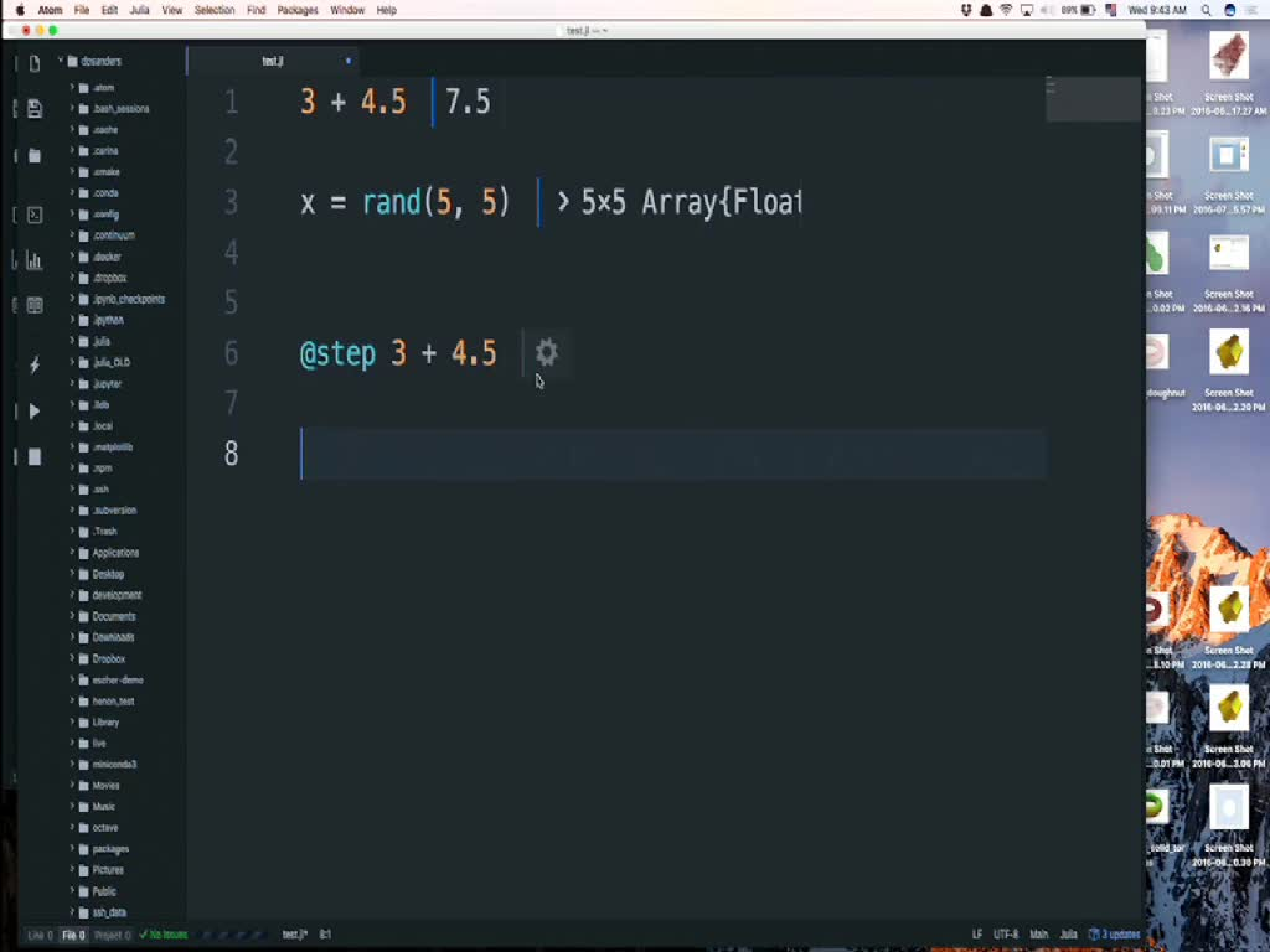
4 Julia generates specialized code

What happens if we use arguments of a different type?

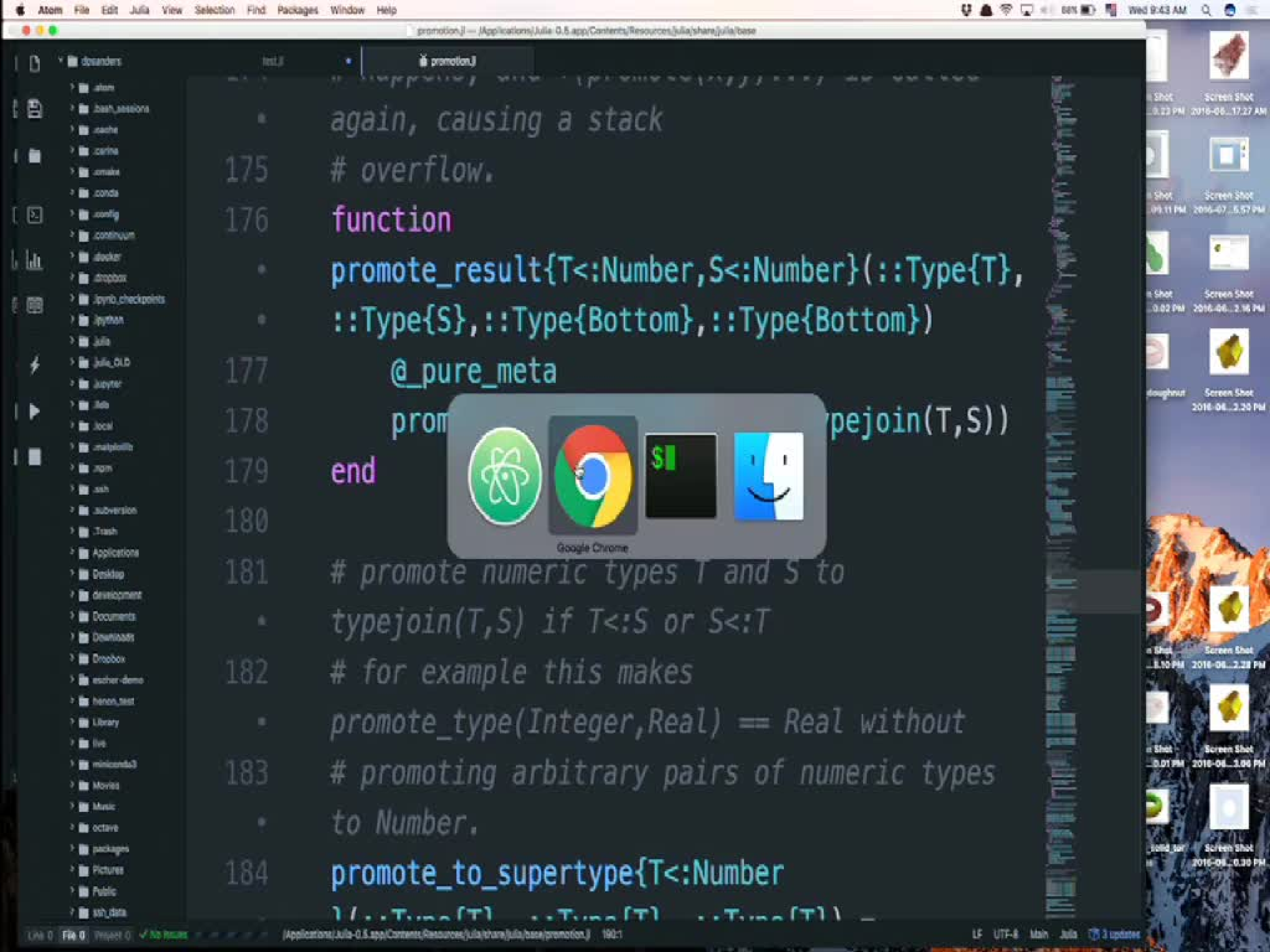
Let's try passing argu pes to the function f:

```
In [14]: f(3.5, 4)

Out[14]: 7.5

In [16]: promote(3.5, 4)

Out[16]: (3.5,4.0)
```


Atom File Edit Julia View Selection Find Packages Window Help
test.jl
3 + 4.5
3 +
@step 3 + 4.5
=
linter-julia Error : ParseError("unexpected \"=\"")


test.jl
3 + 4.5 7.5
x = rand(5, 5) › 5×5 Array{Float
@step 3 + 4.5

```
again, causing a stack
# overflow.
function
promote_result{T<:Number,S<:Number}(::Type{T},
::Type{S},::Type{Bottom},::Type{Bottom})
    @_pure_meta
    promote_to_supertype(T, S, typejoin(T,S))
end

# promote numeric types T and S to
typejoin(T,S) if T<:S or S<:T
# for example this makes
promote_type(Integer,Real) == Real without
# promoting arbitrary pairs of numeric types
to Number.
promote_to_supertype{T<:Number
```


promotion.jl — /Applications/Julia-0.5.app/Contents/Resources/julia/share/julia/base
promotion.jl
again, causing a stack
overflow.
function
promote_result{T<:Number,S<:Number}(::Type{T},
::Type{S},::Type{Bottom},::Type{Bottom})
@_pure_meta
pejoin(T,S))
end
promote numeric types T and S to
typejoin(T,S) if T<:S or S<:T
for example this makes
promote_type(Integer,Real) == Real without
promoting arbitrary pairs of numeric types
to Number.
promote_to_supertype{T<:Number
Google Chrome
Applications
Desktop
Documents
Downloads
Library
Movies
Music
Pictures
Public

4 Julia generates specialized code

What happens if we use arguments of a different type?

Let's try passing arguments of different types to the function `f`:

```
In [14]: f(3.5, 4)
```

```
Out[14]: 7.5
```

```
In [16]: promote(3.5, 4)
```

```
Out[16]: (3.5,4.0)
```

Out[17]:
```
LambdaInfo for f(::Float64, ::Int64)
:(begin
        return (Base.box)(Base.Float64,(Base.add_float)(x,(Base.box)
(Float64,(Base.sitofp)(Float64,y))))
    end::Float64)
```

In [18]:
```
@code_llvm f(3.5, 4)
```

```
define double @julia_f_71571(double, i64) #0 {
top:
  %2 = sitofp i64 %1 to double
  %3 = fadd double %2, %0
  ret double %3
}
```

In order to do the sum, the integer 4 is converted to Float64 (by sitofp)

In order to do the sum, the integer 4 is converted to `Float64` (by `sitofp`)

```
In [ ]: @code_native f(3.5, 4)
```

In order to do the sum, the integer 4 is converted to `Float64` (by `sitofp`)

```
In [19]: @code_native f(3.5, 4)

        .section        __TEXT,__text,regular,pure_instructions
Filename: In[8]
        pushq   %rbp
        movq    %rsp, %rbp
Source line: 1
        cvtsi2sdq       %rdi, %xmm1
        addsd   %xmm1, %xmm0
        popq    %rbp
        retq
```

So Julia generates **specialised code** according to the argument types.

If one of these specialised versions is later used, the corresponding specialised version will be called.

So Julia generates **specialised code** according to the argument types.

If one of these specialised versions is later used, the corresponding specialised version will be called.

?

5 Profiling

In [21]:
```
f(x) = x^2
```

```
WARNING: Method definition f(Any) in module Main at In[20]:1 overw
ritten at In[21]:1.
```

Out[21]:
```
f (generic function with 2 methods)
```

In [22]:
```
@time f(1)
```

```
  0.001289 seconds (355 allocations: 20.188 KB)
```

Out[22]:
```
1
```

In [23]:
```
@time f(1)
```

```
  0.000001 seconds (4 allocations: 160 bytes)
```

Out[23]:
```
1
```

5 Profiling

In [21]:
```
f(x) = x^2
```

```
WARNING: Method definition f(Any) in module Main at In[20]:1 overw
ritten at In[21]:1.
```

Out[21]:
```
f (generic function with 2 methods)
```

In [22]:
```
@time f(1)
```

```
  0.001289 seconds (355 allocations: 20.188 KB)
```

Out[22]:
```
1
```

In [23]:
```
@time f(1)
```

```
  0.000001 seconds (4 allocations: 160 bytes)
```

Out[23]:
```
1
```

In [ ]:
```
@time f(1)
```

5.1 Type instability

Let's look at a more complicated function, for a step of a random walk:

```
In [ ]: step1() = rand(-1:1) > 0 ? 1 : -1.0  # ternary operator; if then
```

5.1 Type instability

Let's look at a more complicated function, for a step of a random walk:

```
In [24]: step1() = rand(-1:1) > 0 ? 1 : -1.0  # ternary operator; if then

Out[24]: step1 (generic function with 1 method)

In [25]: @code_lowered step1()

Out[25]: LambdaInfo template for step1() at In[24]:1
         :(begin
                 nothing
                 unless (Main.rand)((Main.colon)(-1,1)) > 0 goto 4
                 return 1
```

5.1 Type instability

Let's look at a more complicated function, for a step of a random walk:

```
In [24]: step1() = rand(-1:1) > 0 ? 1 : -1.0  # ternary operator; if then
```

```
Out[24]: step1 (generic function with 1 method)
```

```
In [25]: @code_lowered step1()
```

```
Out[25]: LambdaInfo template for step1() at In[24]:1
:(begin
        nothing
        unless (Main.rand)((Main.colon)(-1,1)) > 0 goto 4
        return 1
        4:
        return -1.0
    end)
```

```
In [ ]: @code_typed g(3)
```

5.1 Type instability

Let's look at a more complicated function, for a step of a random walk:

In [24]:
```
step1() = rand(-1:1) > 0 ? 1 : -1.0  # ternary operator; if then
```

Out[24]:
```
step1 (generic function with 1 method)
```

In [25]:
```
@code_lowered step1()
```

Out[25]:
```
LambdaInfo template for step1() at In[24]:1
:(begin
        nothing
        unless (Main.rand)((Main.colon)(-1,1)) > 0 goto 4
        return 1
        4:
        return -1.0
    end)
```

?

Let's look at a more complicated function, for a step of a random walk:

In [24]:
```
step1() = rand(-1:1) > 0 ? 1 : -1.0  # ternary operator; if then
```

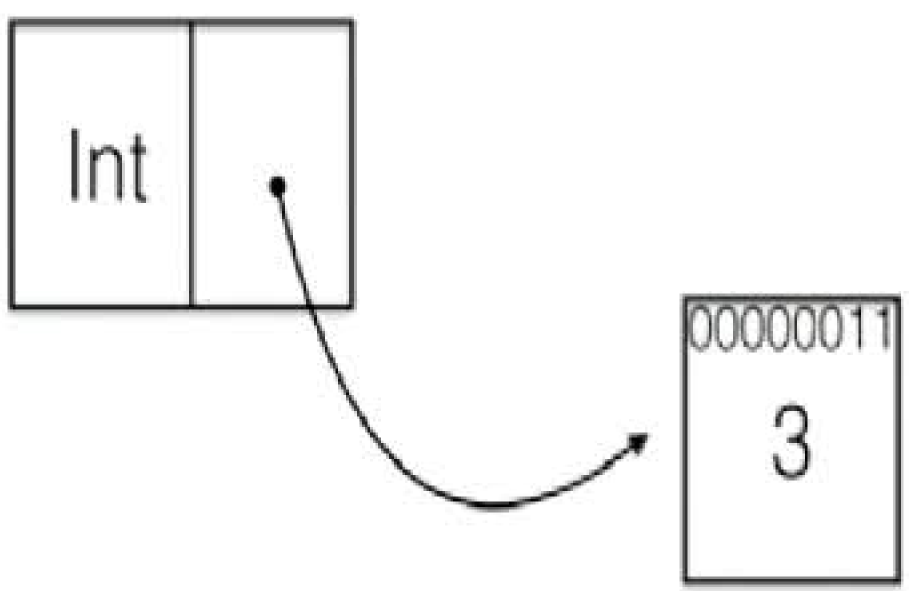
Out[24]:
```
step1 (generic function with 1 method)
```

In [25]:
```
@code_lowered step1()
```

Out[25]:
```
LambdaInfo template for step1() at In[24]:1
:(begin
        nothing
        unless (Main.rand)((Main.colon)(-1,1)) > 0 goto 4
        return 1
        4:
        return -1.0
    end)
```

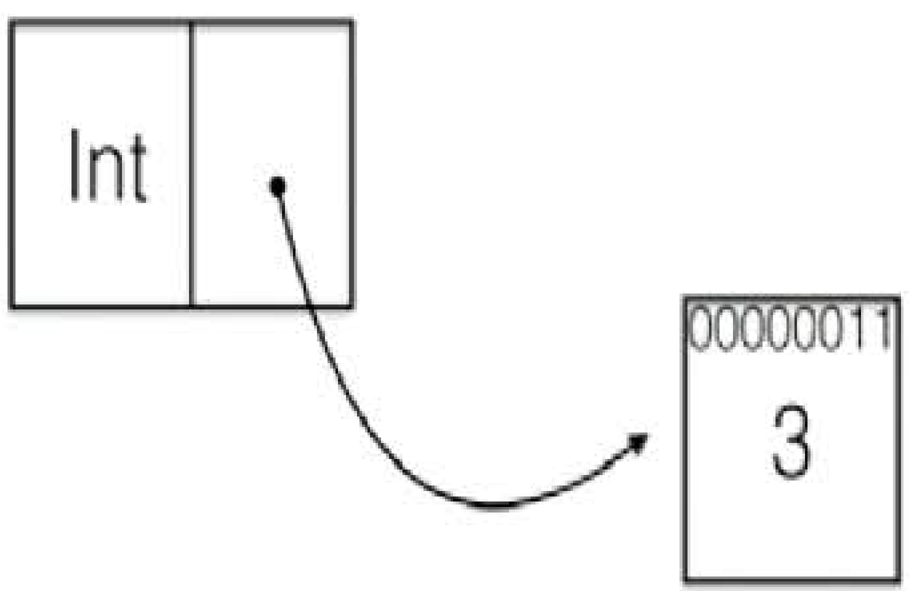
Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

This leads to inefficient code (at least currently), since the result will be **boxed**:



Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

This leads to inefficient code (at least currently), since the result will be **boxed**:

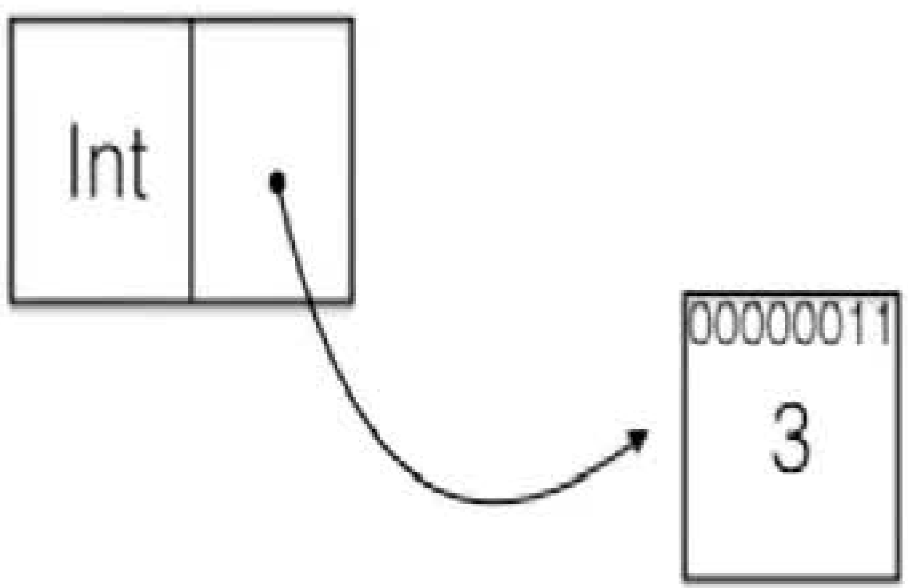


So Julia generates **specialised code** according to the argument types.

If one of these specialised versions is later used, the corresponding specialised version will be called.

Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

This leads to inefficient code (at least currently), since the result will be **boxed**:



This leads to a significant performance loss:

```
In [ ]: step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

        step2() = randn() > 0 ? 1 : -1    # only Int64
```

First, compile the functions by running each once:

```
In [ ]: step1()
        step2()
```

Slide Type Slide

1 Why Julia is fast

[...]

Slide Type Slide

2 Why Julia ~~is fast~~ can be fast

Slide Type Fragment

David P. Sanders

Slide Type -

Department of Physics, Faculty of Sciences

National University of Mexico (UNAM)

Slide Type Slide

jupyter Why Julia is fast Last Checkpoint: 15 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Navigate Widgets Help

Julia 0.5.0

Code CellToolbar

```
    end::Float64)
```

In [18]:

Slide Type Fragment

```
@code_llvm f(3.5, 4)
```

```
define double @julia_f_71571(double, i64) #0 {
top:
  %2 = sitofp i64 %1 to double
  %3 = fadd double %2, %0
  ret double %3
}
```

Slide Type Sub-Slide

In order to do the sum, the integer 4 is converted to `Float64` (by `sitofp`)

In [19]:

Slide Type -

```
@code_native f(3.5, 4)
```

```
        .section        __TEXT,__text,regular,pure_instructions
Filename: In[8]
        pushq   %rbp
        movq    %rsp, %rbp
Source line: 1
        cvtsi2sdq       %rdi, %xmm1
        addsd   %xmm1, %xmm0
        popq    %rbp
        retq
        nop
```

Slide Type Sub-Slide

So Julia generates **specialised code** according to the argument types.

jupyter Why Julia is fast Last Checkpoint: 15 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Navigate Widgets Help

Julia 0.5.0

Code CellToolbar

Slide Type Sub-Slide

In order to do the sum, the integer 4 is converted to `Float64` (by `sitofp`)

In [19]: Slide Type -

```
@code_native f(3.5, 4)
```

```
        .section        __TEXT,__text,regular,pure_instructions
Filename: In[8]
        pushq   %rbp
        movq    %rsp, %rbp
Source line: 1
        cvtsi2sdq       %rdi, %xmm1
        addsd   %xmm1, %xmm0
        popq    %rbp
        retq
        nop
```

Slide Type Sub-Slide

So Julia generates **specialised code** according to the argument types.

Slide Type -

If one of these specialised versions is later used, the corresponding specialised version will be called.

Slide Type Slide

5 Profiling

In [21]: Slide Type -

jupyter Why Julia is fast Last Checkpoint: 15 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Navigate Widgets Help

Julia 0.5.0

Code CellToolbar

Slide Type Sub-Slide

In order to do the sum, the integer 4 is converted to `Float64` (by `sitofp`)

In [19]:

Slide Type -

```
@code_native f(3.5, 4)
```

```
	.section	__TEXT,__text,regular,pure_instructions
Filename: In[8]
	pushq	%rbp
	movq	%rsp, %rbp
Source line: 1
	cvtsi2sdq	%rdi, %xmm1
	addsd	%xmm1, %xmm0
	popq	%rbp
	retq
	nop
```

Slide Type Sub-Slide

So Julia generates **specialised code** according to the argument types.

Slide Type -

If one of these specialised versions is later used, the corresponding specialised version will be called.

Slide Type Slide

jupyter Why Julia is fast (unsaved changes)

Logout

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

CellToolbar

```
  %3 = fadd double %2, %0
  ret double %3
}
```

Slide Type Sub-Slide

In order to do the sum, the integer 4 is converted to `Float64` (by `sitofp`)

In [19]: Slide Type -

```
@code_native f(3.5, 4)
```

```
        .section        __TEXT,__text,regular,pure_instructions
Filename: In[8]
        pushq   %rbp
        movq    %rsp, %rbp
Source line: 1
        cvtsi2sdq       %rdi, %xmm1
```

jupyter **Why Julia is fast** (unsaved changes)

Logout

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

Code CellToolbar

```julia
f(x) = x^2
```

```
WARNING: Method definition f(Any) in module Main at In[20]:1 overwritten at In[21]:1.
```

Out[21]: f (generic function with 2 methods)

In [22]: Slide Type: Fragment

```julia
@time f(1)
```

```
  0.001289 seconds (355 allocations: 20.188 KB)
```

Out[22]: 1

In [23]: Slide Type: Fragment

```julia
@time f(1)
```

```
  0.000001 seconds (4 allocations: 160 bytes)
```

jupyter Why Julia is fast (unsaved changes)

Logout

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

Code CellToolbar

```
tten at In[21]:1.
```

Out[21]: f (generic function with 2 methods)

In [22]: Slide Type Fragment

```julia
@time f(1)
```

```
0.001289 seconds (355 allocations: 20.188 KB)
```

Out[22]: 1

In [23]: Slide Type Fragment

```julia
@time f(1)
```

```
0.000001 seconds (4 allocations: 160 bytes)
```

Out[23]: 1

In [ ]: Slide Type Fragment

jupyter Why Julia is fast (unsaved changes)

Logout

File Edit View Insert Cell Kernel Navigate Widgets Help

Julia 0.5.0

Code

CellToolbar

Slide Type Sub-Slide

5.1 Type instability

Slide Type Fragment

Let's look at a more complicated function, for a step of a random walk:

In [24]: Slide Type -

```
step1() = rand(-1:1) > 0 ? 1 : -1.0  # ternary operator; if then
```

Out[24]: step1 (generic function with 1 method)

In [25]: Slide Type Fragment

Jupyter Why Julia is fast (unsaved changes)

Logout

File Edit View Insert Cell Kernel Navigate Widgets Help

Julia 0.5.0

Code

CellToolbar

5.1 Type instability

Slide Type Fragment

Let's look at a more complicated function, for a step of a random walk:

In [24]:

Slide Type -

```
step1() = rand(-1:1) > 0 ? 1 : -1.0  # ternary operator; if then
```

Out[24]:

```
step1 (generic function with 1 method)
```

In [25]:

Slide Type Fragment

```
@code_lowered step1()
```

Out[25]:

```
LambdaInfo template for step1() at In[24]:1
:(begin
        nothing
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

Toggle Header
Toggle Toolbar
Cell Toolbar

Code | CellToolbar

...stability

Slide Type: Fragment

Let's look at a more complicated function, for a step of a random walk:

In [24]: (Slide Type: -)

```
step1() = rand(-1:1) > 0 ? 1 : -1.0  # ternary operator; if then
```

Out[24]:

```
step1 (generic function with 1 method)
```

In [25]: (Slide Type: Fragment)

```
@code_lowered step1()
```

Out[25]:

```
LambdaInfo template for step1() at In[24]:1
:(begin
        nothing
        unless (Main.rand)((Main.colon)(-1,1)) > 0 goto 4
        return 1
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

5.1 Type instability

Slide Type Fragment

Let's look at a more complicated function, for a step of a random walk:

In [24]: Slide Type -

```
step1() = rand(-1:1) > 0 ? 1 : -1.0  # ternary operator; if then
```

Out[24]: step1 (generic function with 1 method)

In [25]: Slide Type Fragment

```
@code_lowered step1()
```

Out[25]:

```
LambdaInfo template for step1() at In[24]:1
:(begin
        nothing
        unless (Main.rand)((Main.colon)(-1,1)) > 0 goto 4
        return 1
        4:
        return -1.0
    end)
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
step1() = rand(-1:1) > 0 ? 1 : -1.0  # ternary operator; if then
```

Out[24]: step1 (generic function with 1 method)

In [25]: Slide Type Fragment

```
@code_lowered step1()
```

Out[25]:
```
LambdaInfo template for step1() at In[24]:1
:(begin
        nothing
        unless (Main.rand)((Main.colon)(-1,1)) > 0 goto 4
        return 1
        4:
        return -1.0
    end)
```

Slide Type Sub-Slide

Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

This leads to inefficient code (at least currently), since the result will be **boxed**:

Out[25]:
```
LambdaInfo template for step1() at In[24]:1
:(begin
        nothing
        unless (Main.rand)((Main.colon)(-1,1)) > 0 goto 4
        return 1
        4:
        return -1.0
    end)
```

Slide Type Sub-Slide

Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

This leads to inefficient code (at least currently), since the result will be **boxed**:

Slide Type -

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

In [24]: Slide Type -

```julia
step1() = rand(-1:1) > 0 ? 1 : -1.0  # ternary operator; if then
```

Out[24]: step1 (generic function with 1 method)

In [25]: Slide Type Fragment

```julia
@code_lowered step1()
```

Out[25]:

```
LambdaInfo template for step1() at In[24]:1
:(begin
        nothing
        unless (Main.rand)((Main.colon)(-1,1)) > 0 goto 4
        return 1
        4:
        return -1.0
    end)
```

Slide Type Sub-Slide

Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

In [24]: Slide Type -

```
step1() = rand(-1:1) > 0 ? 1 : -1.0  # ternary operator; if then
```

Out[24]: step1 (generic function with 1 method)

In [25]: Slide Type Fragment

```
@code_lowered step1()
```

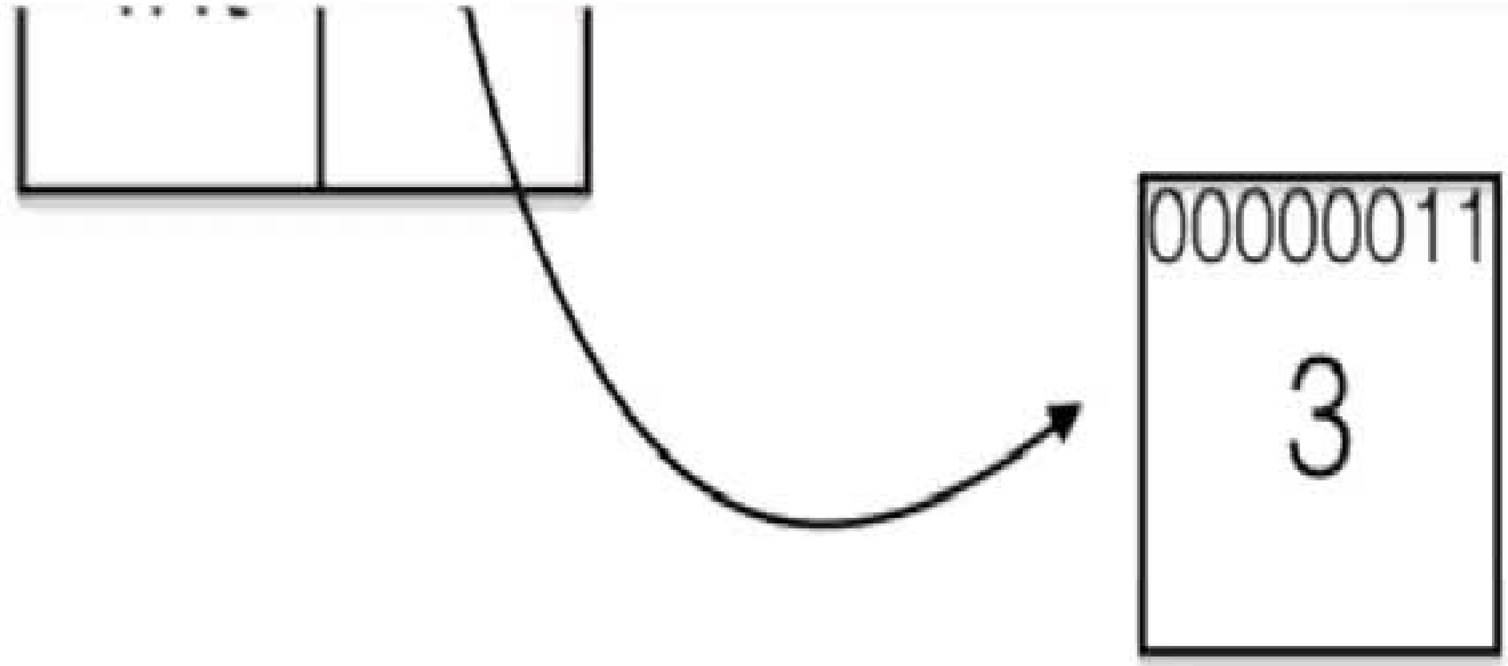
Out[25]:

```
LambdaInfo template for step1() at In[24]:1
:(begin
        nothing
        unless (Main.rand)((Main.colon)(-1,1)) > 0 goto 4
        return 1
        4:
        return -1.0
    end)
```

Slide Type Sub-Slide

Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0



Slide Type Sub-Slide

This leads to a significant performance loss:

In [29]: Slide Type -

```
step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

step2() = randn() > 0 ? 1 : -1    # only Int64
```

```
WARNING: Method definition step1() in module Main at In[28]:1 overwr
itten at In[29]:1.
WARNING: Method definition step2() in module Main at In[28]:3 overwr
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Slide Type Sub-Slide

This leads to a significant performance loss:

In [29]: Slide Type -

```
step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

step2() = randn() > 0 ? 1 : -1    # only Int64
```

```
WARNING: Method definition step1() in module Main at In[28]:1 overwr
itten at In[29]:1.
WARNING: Method definition step2() in module Main at In[28]:3 overwr
itten at In[29]:3.
```

Out[29]: step2 (generic function with 1 method)

Slide Type -

First, compile the functions by running each once:

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Slide Type Sub-Slide

This leads to a significant performance loss:

In [29]: Slide Type -

```
step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

step2() = randn() > 0 ? 1 : -1    # only Int64
```

```
WARNING: Method definition step1() in module Main at In[28]:1 overwr
itten at In[29]:1.
WARNING: Method definition step2() in module Main at In[28]:3 overwr
itten at In[29]:3.
```

Out[29]: `step2 (generic function with 1 method)`

Slide Type -

First, compile the functions by running each once:

In [30]: Slide Type -

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

In [31]: Slide Type -

```
step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

step2() = randn() > 0 ? 1 : -1    # only Int64
```

```
WARNING: Method definition step1() in module Main at In[29]:1 overwr
itten at In[31]:1.
WARNING: Method definition step2() in module Main at In[29]:3 overwr
itten at In[31]:3.
```

Out[31]: step2 (generic function with 1 method)

Slide Type -

First, compile the functions by running each once:

In [30]: Slide Type -

```
step1()
step2()
```

Out[30]: -1

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

Toggle Header
Toggle Toolbar
Cell Toolbar ▸

None
Edit Metadata
Raw Cell Format
Slideshow

```
d definition step2() in module Main at In[29]:3 overwr
]:3.
```

Out[3 ... thod)

Slide Type -

First, compile the fu ... once:

In [30]: Slide Type -

```
step1()
step2()
```

Out[30]: -1

In [ ]: Slide Type Sub-Slide

```
@time sum(step1() for i in 1:10^7)    # generator
```

In [ ]: Slide Type -

```
@time sum(step2() for i in 1:10^7)
```

```
itten at In[31]:1.
WARNING: Method definition step2() in module Main at In[29]:3 overwr
itten at In[31]:3.
```

Out[31]: step2 (generic function with 1 method)

First, compile the functions by running each once:

In [32]:
```
step1()
step2()
```

Out[32]: 1

In [ ]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

In [ ]:
```
@time sum(step2() for i in 1:10^7)
```

Excess allocations are usually a sign of type instability.

5.2 Inlining

```
itten at In[31]:3.
```

Out[31]: step2 (generic function with 1 method)

First, compile the functions by running each once:

In [32]:
```
step1()
step2()
```

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
  0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
  0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

First, compile the functions by running each once:

In [32]:
```
step1()
step2()
```

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [32]:
```
step1()
step2()
```

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

5.2 Inlining

First, compile the functions by running each once:

In [32]:
```
step1()
step2()
```

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
step2() = randn() > 0 ? 1 : -1     # only Int64
```

```
WARNING: Method definition step1() in module Main at In[29]:1 overwr
itten at In[31]:1.
WARNING: Method definition step2() in module Main at In[29]:3 overwr
itten at In[31]:3.
```

Out[31]: step2 (generic function with 1 method)

First, compile the functions by running each once:

In [32]:
```
step1()
step2()
```

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)    # generator
```

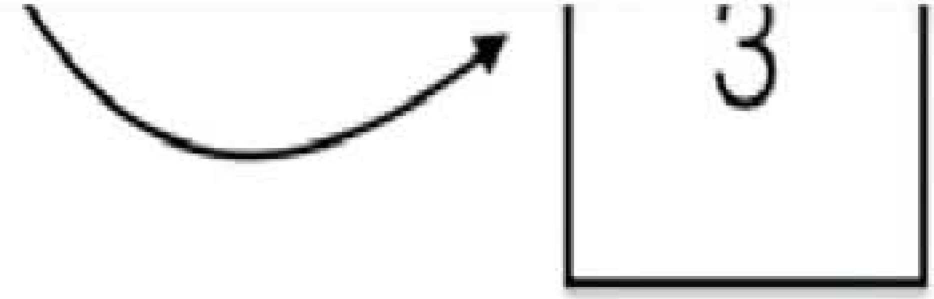
```
0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

Notebook saved | Julia 0.5.0

File Edit View Insert Cell Kernel Navigate Widgets Help



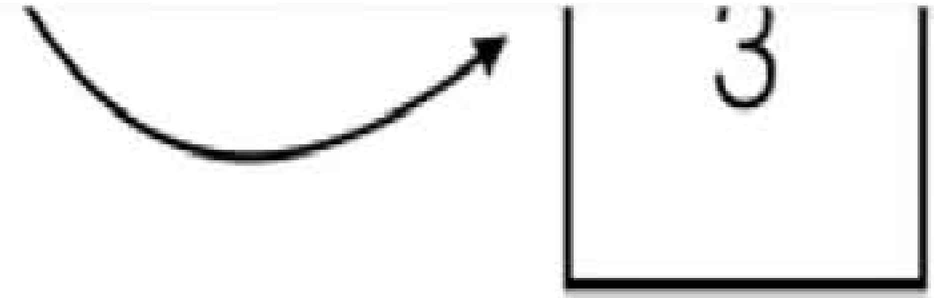
This leads to a significant performance loss:

In [31]:
```
step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

step2() = randn() > 0 ? 1 : -1    # only Int64
```

```
WARNING: Method definition step1() in module Main at In[29]:1 overwr
itten at In[31]:1.
WARNING: Method definition step2() in module Main at In[29]:3 overwr
itten at In[31]:3.
```

Out[31]:
```
step2 (generic function with 1 method)
```

First, compile the functions by running each once:



This leads to a significant performance loss:

In [31]:
```
step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

step2() = randn() > 0 ? 1 : -1    # only Int64
```

```
WARNING: Method definition step1() in module Main at In[29]:1 overwritten at In[31]:1.
WARNING: Method definition step2() in module Main at In[29]:3 overwritten at In[31]:3.
```

Out[31]: `step2 (generic function with 1 method)`

First, compile the functions by running each once:

In [32]:
```
step1()
step2()
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

This leads to a significant performance loss:

In [31]:
```
step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

step2() = randn() > 0 ? 1 : -1    # only Int64
```

```
WARNING: Method definition step1() in module Main at In[29]:1 overwritten at In[31]:1.
WARNING: Method definition step2() in module Main at In[29]:3 overwritten at In[31]:3.
```

Out[31]: step2 (generic function with 1 method)

First, compile the functions by running each once:

In [32]:
```
step1()
step2()
```

Out[32]: 1

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

This leads to a significant performance loss:

In [31]:
```
step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

step2() = randn() > 0 ? 1 : -1    # only Int64
```

```
WARNING: Method definition step1() in module Main at In[29]:1 overwr
itten at In[31]:1.
WARNING: Method definition step2() in module Main at In[29]:3 overwr
itten at In[31]:3.
```

Out[31]: step2 (generic function with 1 method)

First, compile the functions by running each once:

In [32]:
```
step1()
step2()
```

Out[32]: 1

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
  0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
  0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [*]:
```
@code_warntype step1()
```

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]:
```
5764
```

Excess allocations are usually a sign of type instability.

In [35]:
```
@code_warntype step1()
```

```
Variables:
  #self#::#step1
  r::UInt64
  rabs::Int64
  idx::Int64
  x::Float64
  #temp#@_6::Union{Bool,Int64}
  #temp#@_7::Float64
  #temp#@_8::Float64
  fx::Float64

Body:
  begin
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]:
```
5764
```

Excess allocations are usually a sign of type instability.

In [35]:
```
@code_warntype step1()
```

```
Variables:
  #self#::#step1
  r::UInt64
  rabs::Int64
  idx::Int64
  x::Float64
  #temp#@_6::Union{Bool,Int64}
  #temp#@_7::Float64
  #temp#@_8::Float64
  fx::Float64

Body:
  begin
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [35]: `@code_typed step1()`

```
Variables:
  #self#::#step1
  r::UInt64
  rabs::Int64
  idx::Int64
  x::Float64
  #temp#@_6::Union{Bool,Int64}
  #temp#@_7::Float64
  #temp#@_8::Float64
  fx::Float64

Body:
  begin
      $(Expr(:inbounds, false))
      # meta: location random.jl randn 1129
      $(Expr(:inbounds, false))
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [*]:
```
@code_typed step1()
```

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

In [ ]:
```
@code_lowered h2(10)
```

In [ ]:
```
@code_typed h2(10)
```

In [ ]:
```
@code_llvm h2(10)
```

Julia has **inlined** the function h1 into h2

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0 O

Excess allocations are usually a sign of type instability.

In [36]:
```
@code_typed step1()
```

Out[36]:
```
LambdaInfo for step1()
:(begin
        $(Expr(:inbounds, false))
        # meta: location random.jl randn 1129
        $(Expr(:inbounds, false))
        # meta: location random.jl randn 1129
        $(Expr(:inbounds, true)) # line 1130:
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52 263
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw 125
        $(Expr(:inbounds, false))
        # meta: location random.jl reserve_1 111
        unless ((Core.getfield)(Base.Random.GLOBAL_RNG,:idx)::Int64
 === (Base.box)(Int64,(Base.sext_int)(Int64,Base.Random.MTCacheLengt
h)))::Bool goto 25
        # meta: location random.jl gen_rand 107
        SSAValue(1) = (Core.getfield)(Base.Random.GLOBAL_RNG,:stat
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0 O

Excess allocations are usually a sign of type instability.

In [36]:
```julia
@code_typed step1()
```

Out[36]:
```
LambdaInfo for step1()
:(begin
        $(Expr(:inbounds, false))
        # meta: location random.jl randn 1129
        $(Expr(:inbounds, false))
        # meta: location random.jl randn 1129
        $(Expr(:inbounds, true)) # line 1130:
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52 263
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw 125
        $(Expr(:inbounds, false))
        # meta: location random.jl reserve_1 111
        unless ((Core.getfield)(Base.Random.GLOBAL_RNG,:idx)::Int64
 === (Base.box)(Int64,(Base.sext_int)(Int64,Base.Random.MTCacheLengt
h)))::Bool goto 25
        # meta: location random.jl gen_rand 107
        SSAValue(1) = (Core.getfield)(Base.Random.GLOBAL_RNG,:stat
e)::Base.dSFMT.DSFMT_state
        SSAValue(0) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Out[36]:
```
LambdaInfo for step1()
:(begin
        $(Expr(:inbounds, false))
        # meta: location random.jl randn 1129
        $(Expr(:inbounds, false))
        # meta: location random.jl randn 1129
        $(Expr(:inbounds, true)) # line 1130:
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52 263
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw 125
        $(Expr(:inbounds, false))
        # meta: location random.jl reserve_1 111
        unless ((Core.getfield)(Base.Random.GLOBAL_RNG,:idx)::Int64
 === (Base.box)(Int64,(Base.sext_int)(Int64,Base.Random.MTCacheLengt
h)))::Bool goto 25
        # meta: location random.jl gen_rand 107
        SSAValue(1) = (Core.getfield)(Base.Random.GLOBAL_RNG,:stat
e)::Base.dSFMT.DSFMT_state
        SSAValue(0) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
```

5.2 Inlining

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

Out[36]:
```
LambdaInfo for step1()
:(begin
        $(Expr(:inbounds, false))
        # meta: location random.jl randn 1129
        $(Expr(:inbounds, false))
        # meta: location random.jl randn 1129
        $(Expr(:inbounds, true)) # line 1130:
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52 263
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw 125
        $(Expr(:inbounds, false))
        # meta: location random.jl reserve_1 111
        unless ((Core.getfield)(Base.Random.GLOBAL_RNG,:idx)::Int64
 === (Base.box)(Int64,(Base.sext_int)(Int64,Base.Random.MTCacheLengt
h)))::Bool goto 25
        # meta: location random.jl gen_rand 107
        SSAValue(1) = (Core.getfield)(Base.Random.GLOBAL_RNG,:stat
e)::Base.dSFMT.DSFMT_state
        SSAValue(0) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
```

5.2 Inlining

```
        # meta: location random.jl randn 1129
        $(Expr(:inbounds, true)) # line 1130:
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52 263
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw 125
        $(Expr(:inbounds, false))
        # meta: location random.jl reserve_1 111
        unless ((Core.getfield)(Base.Random.GLOBAL_RNG,:idx)::Int64
 === (Base.box)(Int64,(Base.sext_int)(Int64,Base.Random.MTCacheLengt
h)))::Bool goto 25
        # meta: location random.jl gen_rand 107
        SSAValue(1) = (Core.getfield)(Base.Random.GLOBAL_RNG,:stat
e)::Base.dSFMT.DSFMT_state
        SSAValue(0) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
        $(Expr(:invoke, LambdaInfo for dsfmt_fill_array_close1_open
2!(::Base.dSFMT.DSFMT_state, ::Ptr{Float64}, ::Int64), :(Base.Rando
m.dsfmt_fill_array_close1_open2!), SSAValue(1), :((Core.ccall)(:jl_a
rray_ptr,(Core.apply_type)(Base.Ptr,Float64)::Type{Ptr{Float64}},(Co
```

5.2 Inlining

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

```
        # meta: location random.jl rand_ui52 263
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw 125
        $(Expr(:inbounds, false))
        # meta: location random.jl reserve_1 111
        unless ((Core.getfield)(Base.Random.GLOBAL_RNG,:idx)::Int64
 === (Base.box)(Int64,(Base.sext_int)(Int64,Base.Random.MTCacheLengt
h)))::Bool goto 25
        # meta: location random.jl gen_rand 107
        SSAValue(1) = (Core.getfield)(Base.Random.GLOBAL_RNG,:stat
e)::Base.dSFMT.DSFMT_state
        SSAValue(0) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
        $(Expr(:invoke, LambdaInfo for dsfmt_fill_array_close1_open
2!(::Base.dSFMT.DSFMT_state, ::Ptr{Float64}, ::Int64), :(Base.Rando
m.dsfmt_fill_array_close1_open2!), SSAValue(1), :((Core.ccall)(:jl_a
rray_ptr,(Core.apply_type)(Base.Ptr,Float64)::Type{Ptr{Float64}},(Co
re.svec)(Base.Any)::SimpleVector,SSAValue(0),0)::Ptr{Float64}), :((B
ase.arraylen)((Core.getfield)(Base.Random.GLOBAL_RNG,:vals)::Array{F
loat64,1})::Int64))) # line 108:
```

5.2 Inlining

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        # meta: location random.jl rand_ui52 263
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw 125
        $(Expr(:inbounds, false))
        # meta: location random.jl reserve_1 111
        unless ((Core.getfield)(Base.Random.GLOBAL_RNG,:idx)::Int64
 === (Base.box)(Int64,(Base.sext_int)(Int64,Base.Random.MTCacheLengt
h)))::Bool goto 25
        # meta: location random.jl gen_rand 107
        SSAValue(1) = (Core.getfield)(Base.Random.GLOBAL_RNG,:stat
e)::Base.dSFMT.DSFMT_state
        SSAValue(0) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
        $(Expr(:invoke, LambdaInfo for dsfmt_fill_array_close1_open
2!(::Base.dSFMT.DSFMT_state, ::Ptr{Float64}, ::Int64), :(Base.Rando
m.dsfmt_fill_array_close1_open2!), SSAValue(1), :((Core.ccall)(:jl_a
rray_ptr,(Core.apply_type)(Base.Ptr,Float64)::Type{Ptr{Float64}},(Co
re.svec)(Base.Any)::SimpleVector,SSAValue(0),0)::Ptr{Float64}), :((B
ase.arraylen)((Core.getfield)(Base.Random.GLOBAL_RNG,:vals)::Array{F
loat64,1})::Int64))) # line 108:
```

5.2 Inlining

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        $(Expr(:inbounds, false))
        # meta: location random.jl reserve_1 111
        unless ((Core.getfield)(Base.Random.GLOBAL_RNG,:idx)::Int64
 === (Base.box)(Int64,(Base.sext_int)(Int64,Base.Random.MTCacheLengt
h)))::Bool goto 25
        # meta: location random.jl gen_rand 107
        SSAValue(1) = (Core.getfield)(Base.Random.GLOBAL_RNG,:stat
e)::Base.dSFMT.DSFMT_state
        SSAValue(0) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
        $(Expr(:invoke, LambdaInfo for dsfmt_fill_array_close1_open
2!(::Base.dSFMT.DSFMT_state, ::Ptr{Float64}, ::Int64), :(Base.Rando
m.dsfmt_fill_array_close1_open2!), SSAValue(1), :((Core.ccall)(:jl_a
rray_ptr,(Core.apply_type)(Base.Ptr,Float64)::Type{Ptr{Float64}},(Co
re.svec)(Base.Any)::SimpleVector,SSAValue(0),0)::Ptr{Float64}), :((B
ase.arraylen)((Core.getfield)(Base.Random.GLOBAL_RNG,:vals)::Array{F
loat64,1})::Int64))) # line 108:
        # meta: location random.jl mt_setfull! 102
        (Core.setfield!)(Base.Random.GLOBAL_RNG,:idx,0)::Int64
        # meta: pop location
```

5.2 Inlining

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

```
        $(Expr(:inbounds, false))
        # meta: location random.jl reserve_1 111
        unless ((Core.getfield)(Base.Random.GLOBAL_RNG,:idx)::Int64
 === (Base.box)(Int64,(Base.sext_int)(Int64,Base.Random.MTCacheLengt
h)))::Bool goto 25
        # meta: location random.jl gen_rand 107
        SSAValue(1) = (Core.getfield)(Base.Random.GLOBAL_RNG,:stat
e)::Base.dSFMT.DSFMT_state
        SSAValue(0) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
        $(Expr(:invoke, LambdaInfo for dsfmt_fill_array_close1_open
2!(::Base.dSFMT.DSFMT_state, ::Ptr{Float64}, ::Int64), :(Base.Rando
m.dsfmt_fill_array_close1_open2!), SSAValue(1), :((Core.ccall)(:jl_a
rray_ptr,(Core.apply_type)(Base.Ptr,Float64)::Type{Ptr{Float64}},(Co
re.svec)(Base.Any)::SimpleVector,SSAValue(0),0)::Ptr{Float64}), :((B
ase.arraylen)((Core.getfield)(Base.Random.GLOBAL_RNG,:vals)::Array{F
loat64,1})::Int64))) # line 108:
        # meta: location random.jl mt_setfull! 102
        (Core.setfield!)(Base.Random.GLOBAL_RNG,:idx,0)::Int64
        # meta: pop location
        # meta: pop location
```

5.2 Inlining

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        $(Expr(:invoke, LambdaInfo for dsfmt_fill_array_close1_open2!(::Base.dSFMT.DSFMT_state, ::Ptr{Float64}, ::Int64), :(Base.Random.dsfmt_fill_array_close1_open2!), SSAValue(1), :((Core.ccall)(:jl_array_ptr,(Core.apply_type)(Base.Ptr,Float64)::Type{Ptr{Float64}},(Core.svec)(Base.Any)::SimpleVector,SSAValue(0),0)::Ptr{Float64}), :((Base.arraylen)((Core.getfield)(Base.Random.GLOBAL_RNG,:vals)::Array{Float64,1})::Int64))) # line 108:
        # meta: location random.jl mt_setfull! 102
        (Core.setfield!)(Base.Random.GLOBAL_RNG,:idx,0)::Int64
        # meta: pop location
        # meta: pop location
        #temp#@_6 = 0
        goto 27
        25:
        #temp#@_6 = false
        27:
        # meta: pop location
        $(Expr(:inbounds, :pop))
        #temp#@_6
```

5.2 Inlining

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        $(Expr(:invoke, LambdaInfo for dsfmt_fill_array_close1_open
2!(::Base.dSFMT.DSFMT_state, ::Ptr{Float64}, ::Int64), :(Base.Rando
m.dsfmt_fill_array_close1_open2!), SSAValue(1), :((Core.ccall)(:jl_a
rray_ptr,(Core.apply_type)(Base.Ptr,Float64)::Type{Ptr{Float64}},(Co
re.svec)(Base.Any)::SimpleVector,SSAValue(0),0)::Ptr{Float64}), :((B
ase.arraylen)((Core.getfield)(Base.Random.GLOBAL_RNG,:vals)::Array{F
loat64,1})::Int64))) # line 108:
        # meta: location random.jl mt_setfull! 102
        (Core.setfield!)(Base.Random.GLOBAL_RNG,:idx,0)::Int64
        # meta: pop location
        # meta: pop location
        #temp#@_6 = 0
        goto 27
        25:
        #temp#@_6 = false
        27:
        # meta: pop location
        $(Expr(:inbounds, :pop))
        #temp#@_6
```

5.2 Inlining

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        $(Expr(:invoke, LambdaInfo for dsfmt_fill_array_close1_open
2!(::Base.dSFMT.DSFMT_state, ::Ptr{Float64}, ::Int64), :(Base.Rando
m.dsfmt_fill_array_close1_open2!), SSAValue(1), :((Core.ccall)(:jl_a
rray_ptr,(Core.apply_type)(Base.Ptr,Float64)::Type{Ptr{Float64}},(Co
re.svec)(Base.Any)::SimpleVector,SSAValue(0),0)::Ptr{Float64}), :((B
ase.arraylen)((Core.getfield)(Base.Random.GLOBAL_RNG,:vals)::Array{F
loat64,1})::Int64))) # line 108:
        # meta: location random.jl mt_setfull! 102
        (Core.setfield!)(Base.Random.GLOBAL_RNG,:idx,0)::Int64
        # meta: pop location
        # meta: pop location
        #temp#@_6 = 0
        goto 27
        25:
        #temp#@_6 = false
        27:
        # meta: pop location
        $(Expr(:inbounds, :pop))
        #temp#@_6
        $(Expr(:inbounds, false))
```

5.2 Inlining

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

```
  0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
  0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [36]:
```
@code_typed step1()
```

```
        #temp#@_6 = 0
        goto 27
        25:
        #temp#@_6 = false
        27:
        # meta: pop location
        $(Expr(:inbounds, :pop))
        #temp#@_6
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw_inbounds 124
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Excess allocations are usually a sign of type instability.

In [36]: `@code_typed step1()`

```
        #temp#@_6 = 0
        goto 27
        25:
        #temp#@_6 = false
        27:
        # meta: pop location
        $(Expr(:inbounds, :pop))
        #temp#@_6
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw_inbounds 124
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_inbounds 117
        $(Expr(:inbounds, false))
        # meta: location random.jl mt_pop! 104
        $(Expr(:inbounds, true))
        SSAValue(2) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
        SSAValue(3) = (Base.box)(Int64,(Base.add_int)((Core.getfiel
d)(Base.Random.GLOBAL_RNG,:idx)::Int64,1))
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        # meta: pop location
        $(Expr(:inbounds, :pop))
        #temp#@_6
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw_inbounds 124
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_inbounds 117
        $(Expr(:inbounds, false))
        # meta: location random.jl mt_pop! 104
        $(Expr(:inbounds, true))
        SSAValue(2) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
        SSAValue(3) = (Base.box)(Int64,(Base.add_int)((Core.getfiel
d)(Base.Random.GLOBAL_RNG,:idx)::Int64,1))
        (Core.setfield!)(Base.Random.GLOBAL_RNG,:idx,SSAValue(3))::I
```

In [38]:
```
@code_warntype step1()
```

```
Variables:
  #self#::#step1
  r::UInt64
  rabs::Int64
  idx::Int64
  x::Float64
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

```
        # meta: pop location
        $(Expr(:inbounds, :pop))
        #temp#@_6
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw_inbounds 124
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_inbounds 117
        $(Expr(:inbounds, false))
        # meta: location random.jl mt_pop! 104
        $(Expr(:inbounds, true))
        SSAValue(2) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
        SSAValue(3) = (Base.box)(Int64,(Base.add_int)((Core.getfiel
d)(Base.Random.GLOBAL_RNG,:idx)::Int64,1))
        (Core.setfield!)(Base.Random.GLOBAL_RNG,:idx,SSAValue(3))::I
nt64
```

In [38]:

```
@code_warntype step1()
```

```
Variables:
  #self#::#step1
  r::UInt64
  rabs::Int64
  idx::Int64
  x::Float64
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        # meta: location random.jl rand_ui52_raw_inbounds 124
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_inbounds 117
        $(Expr(:inbounds, false))
        # meta: location random.jl mt_pop! 104
        $(Expr(:inbounds, true))
        SSAValue(2) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
        SSAValue(3) = (Base.box)(Int64,(Base.add_int)((Core.getfiel
d)(Base.Random.GLOBAL_RNG,:idx)::Int64,1))
        (Core.setfield!)(Base.Random.GLOBAL_RNG,:idx,SSAValue(3))::I
```

In [38]:

```
@code_warntype step1()
```

```
Variables:
  #self#::#step1
  r::UInt64
  rabs::Int64
  idx::Int64
  x::Float64
  #temp#@_6::Union{Bool,Int64}
  #temp#@_7::Float64
  #temp#@_8::Float64
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        $(Expr(:inbounds, true))
        SSAValue(2) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
        SSAValue(3) = (Base.box)(Int64,(Base.add_int)((Core.getfiel
d)(Base.Random.GLOBAL_RNG,:idx)::Int64,1))
        (Core.setfield!)(Base.Random.GLOBAL_RNG,:idx,SSAValue(3))::I
```

In [38]:

```
@code_warntype step1()
```

```
Variables:
  #self#::#step1
  r::UInt64
  rabs::Int64
  idx::Int64
  x::Float64
  #temp#@_6::Union{Bool,Int64}
  #temp#@_7::Float64
  #temp#@_8::Float64
  fx::Float64

Body:
  begin
      $(Expr(:inbounds, false))
```

```
$(Expr(:inbounds, true)) # line 1130:
$(Expr(:inbounds, false))
# meta: location random il rand ui52 263
```

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

In [ ]:
```
@code_lowered h2(10)
```

In [ ]:
```
@code_typed h2(10)
```

In [ ]:
```
@code_llvm h2(10)
```

Julia has **inlined** the function h1 into h2.

6 User-defined types

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
$(Expr(:inbounds, true)) # line 1130:
$(Expr(:inbounds, false))
# meta: location random il rand ui52 263
```

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

In [ ]:
```
@code_lowered h2(10)
```

In [ ]:
```
@code_typed h2(10)
```

In [ ]:
```
@code_llvm h2(10)
```

Julia has **inlined** the function h1 into h2.

6 User-defined types

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
    rabs::Int64
    idx::Int64
    x::Float64
    #temp#@_6::Union{Bool,Int64}
    #temp#@_7::Float64
    #temp#@_8::Float64
    fx::Float64

  Body:
    begin
        $(Expr(:inbounds, false))
        # meta: location random.jl randn 1129
        $(Expr(:inbounds, false))
        # meta: location random.jl randn 1129
        $(Expr(:inbounds, true)) # line 1130:
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52 263
```

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        25:
        #temp#@_6::Union{Bool,Int64} = false
        27:
        # meta: pop location
        $(Expr(:inbounds, :pop))
        #temp#@_6::Union{Bool,Int64}
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw_inbounds 124
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_inbounds 117
        $(Expr(:inbounds, false))
        # meta: location random.jl mt_pop! 104
        $(Expr(:inbounds, true))
        SSAValue(2) = (Core.getfield)(Base.Random.GLOBAL_RNG,:vals)::A
rray{Float64,1}
        SSAValue(3) = (Base.box)(Int64,(Base.add_int)((Core.getfield)
(Base.Random.GLOBAL_RNG,:idx)::Int64,1))
        (Core.setfield!)(Base.Random.GLOBAL_RNG,:idx,SSAValue(3))::Int
```

5.2 Inlining

In [ ]:

```
h1(x) = 3x
h2(x) = h1(5x)
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        # meta: location random.jl rand_ui52_raw_inbounds 124
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_inbounds 117
        $(Expr(:inbounds, false))
        # meta: location random.jl mt_pop! 104
        $(Expr(:inbounds, true))
        SSAValue(2) = (Core.getfield)(Base.Random.GLOBAL_RNG,:vals)::A
rray{Float64,1}
        SSAValue(3) = (Base.box)(Int64,(Base.add_int)((Core.getfield)
(Base.Random.GLOBAL_RNG,:idx)::Int64,1))
        (Core.setfield!)(Base.Random.GLOBAL_RNG,:idx,SSAValue(3))::Int
64
        #temp#@_7::Float64 = (Base.arrayref)(SSAValue(2),SSAValue
(3))::Float64
        goto 44
        $(Expr(:inbounds, :pop))
        44:
```

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

```
    85:
    return -1.0
  end::Union{Float64,Int64}
```

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

In [ ]:
```
@code_lowered h2(10)
```

In [ ]:
```
@code_typed h2(10)
```

In [ ]:
```
@code_llvm h2(10)
```

Julia has **inlined** the function h1 into h2.

6 User-defined types

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
(Base.Bool,(Base.or_int)((Base.eq_float)(fx::Float64,9.223372036854776e18)::Bool,(Base.slt_int)(0,(Base.box)(Int64,(Base.fptosi)(Int64,fx::Float64)))::Bool)))))) goto 85
        return 1
        85:
        return -1.0
  end::Union{Float64,Int64}
```

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

In [ ]:
```
@code_lowered h2(10)
```

In [ ]:
```
@code_typed h2(10)
```

In [ ]:
```
@code_llvm h2(10)
```

```
        fx::Float64 = (Base.box)(Float64,(Base.sitofp)(Float64,0))
        # meta: pop location
        # meta: pop location
        $(Expr(:inbounds, :pop))
        unless (Base.box)(Base.Bool,(Base.or_int)((Base.lt_float)(fx::
Float64,#temp#@_8::Float64)::Bool,(Base.box)(Base.Bool,(Base.and_in
t)((Base.eq_float)(fx::Float64,#temp#@_8::Float64)::Bool,(Base.box)
(Base.Bool,(Base.or_int)((Base.eq_float)(fx::Float64,9.223372036854
76e18)::Bool,(Base.slt_int)(0,(Base.box)(Int64,(Base.fptosi)(Int64,f
x::Float64)))::Bool)))))) goto 85
        return 1
        85:
        return -1.0
    end::Union{Float64,Int64}
```

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

In [ ]:
```
@code_lowered h2(10)
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        fx::Float64 = (Base.box)(Float64,(Base.sitofp)(Float64,0))
        # meta: pop location
        # meta: pop location
        $(Expr(:inbounds, :pop))
        unless (Base.box)(Base.Bool,(Base.or_int)((Base.lt_float)(fx::Float64,#temp#@_8::Float64)::Bool,(Base.box)(Base.Bool,(Base.and_int)((Base.eq_float)(fx::Float64,#temp#@_8::Float64)::Bool,(Base.box)(Base.Bool,(Base.or_int)((Base.eq_float)(fx::Float64,9.22337203685477 6e18)::Bool,(Base.slt_int)(0,(Base.box)(Int64,(Base.fptosi)(Int64,fx::Float64)))::Bool)))))) goto 85
        return 1
        85:
        return -1.0
    end::Union{Float64,Int64}
```

5.2 Inlining

In [ ]:

```
h1(x) = 3x
h2(x) = h1(5x)
```

In [ ]:

```
@code_lowered h2(10)
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

In [32]:
```
step1()
step2()
```

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
  0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
  0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [36]:
```
@code_typed step1()
```

```
        #temp#@_6 = 0
        goto 27
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

In [32]:
```
step1()
step2()
```

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
  0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

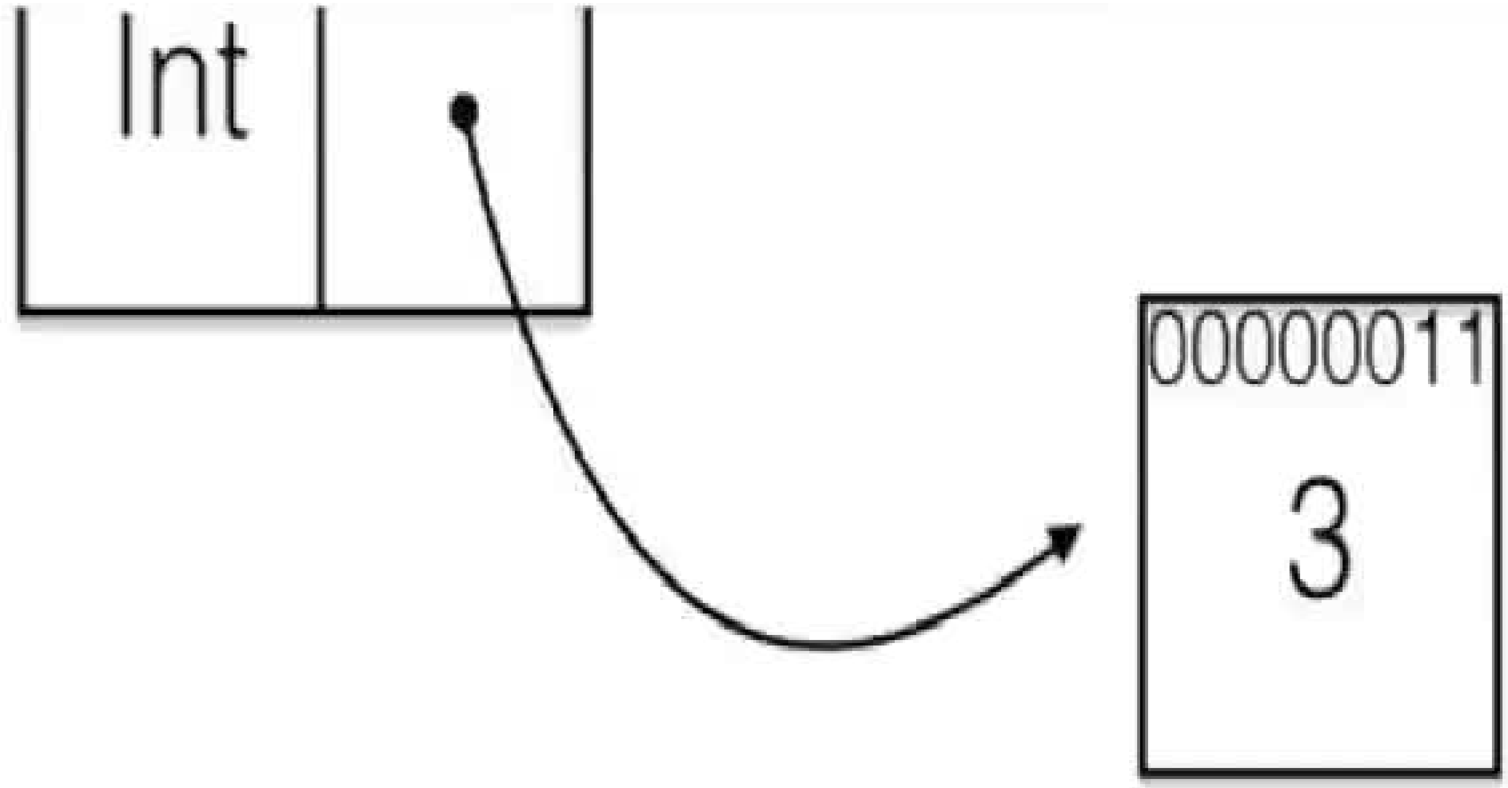
```
  0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [36]:
```
@code_typed step1()
```

```
#temp#@_6 = 0
goto 27
```



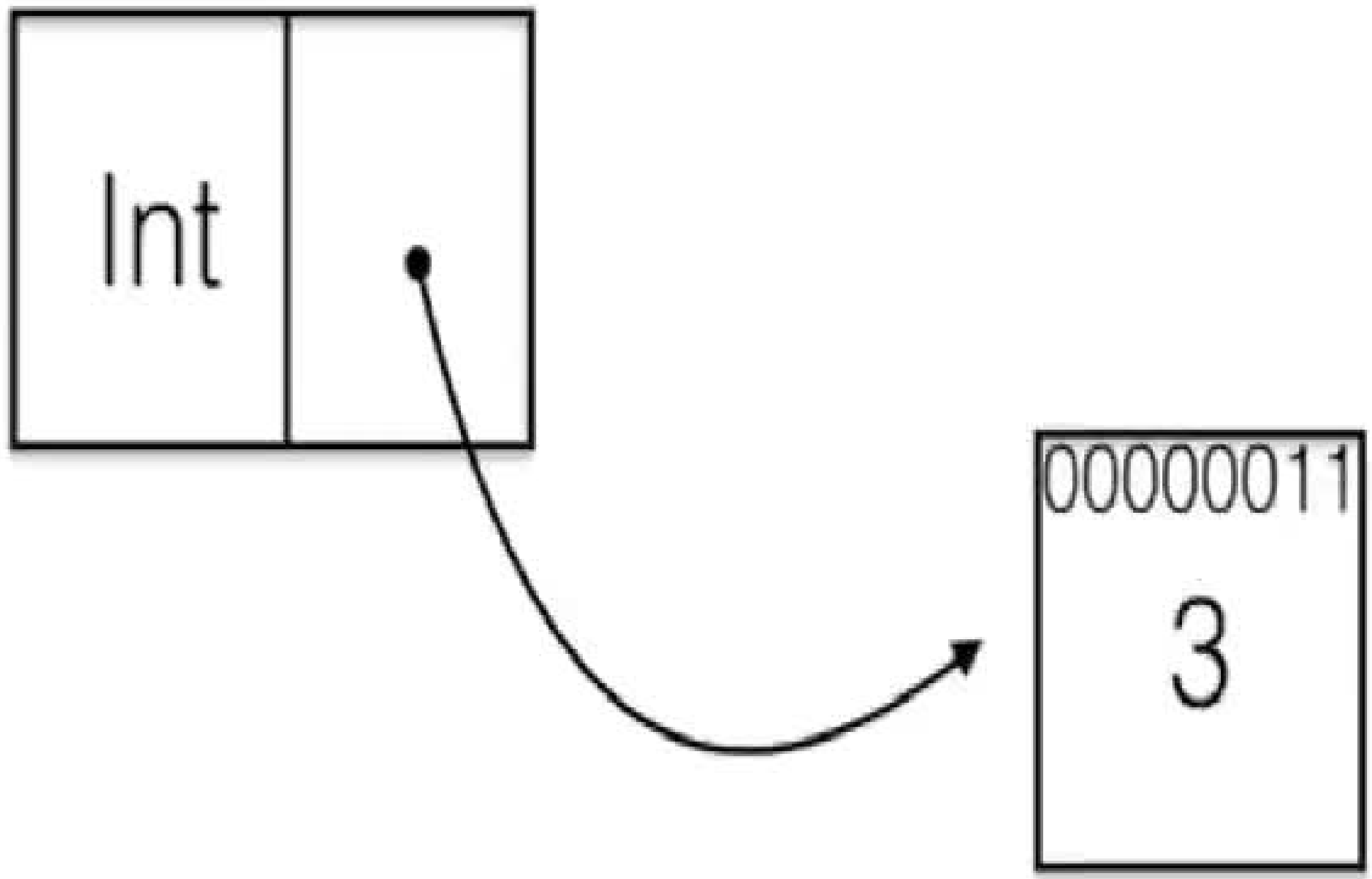
This leads to a significant performance loss:

In [31]:
```
step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

step2() = randn() > 0 ? 1 : -1    # only Int64
```

```
WARNING: Method definition step1() in module Main at In[29]:1 overwr
itten at In[31]:1.
WARNING: Method definition step2() in module Main at In[29]:3 overwr
itten at In[31]:3.
```

Out[31]: step2 (generic function with 1 method)

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

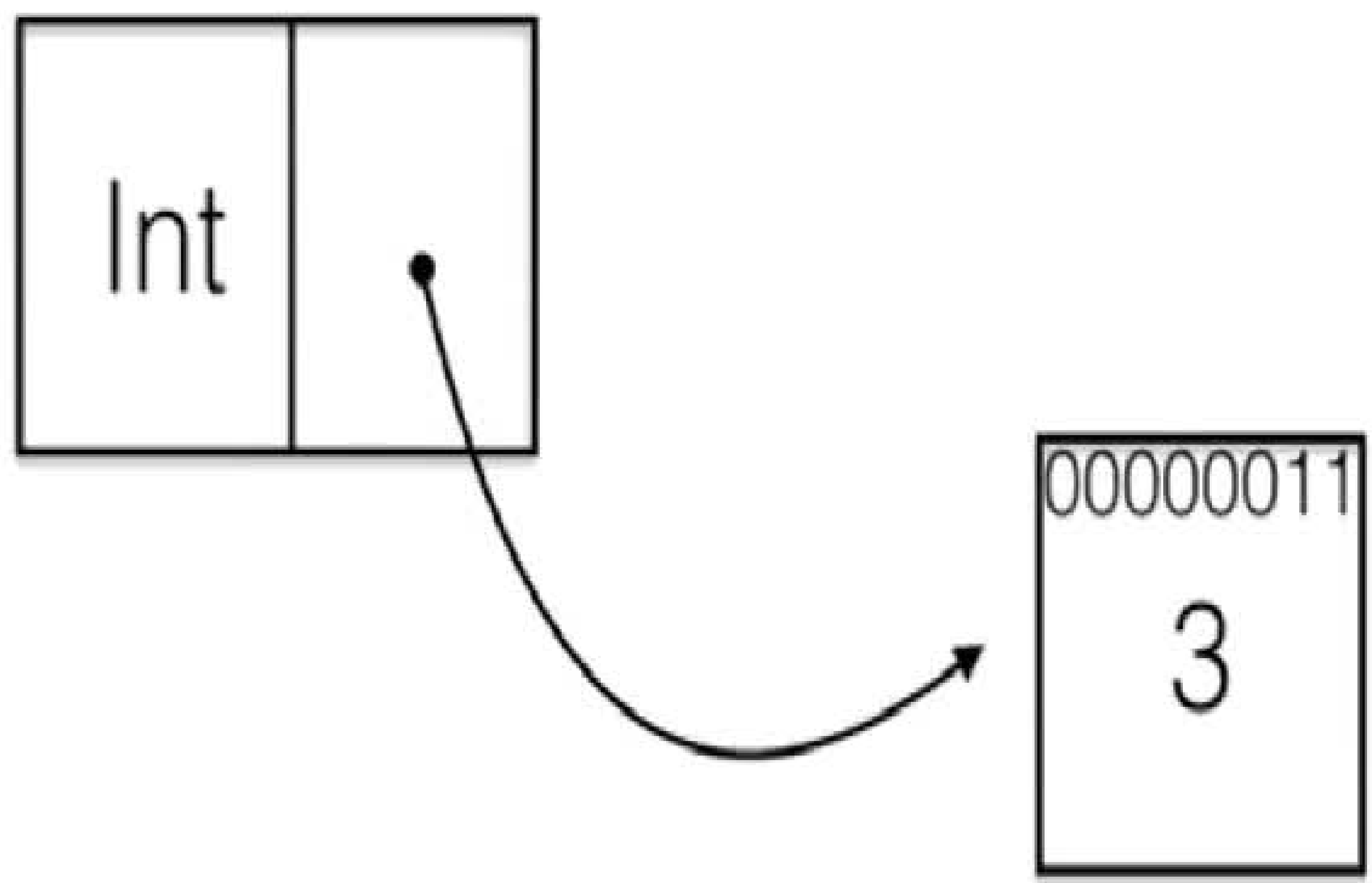


This leads to a significant performance loss:

In [31]:
```julia
step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

step2() = randn() > 0 ? 1 : -1    # only Int64
```

Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

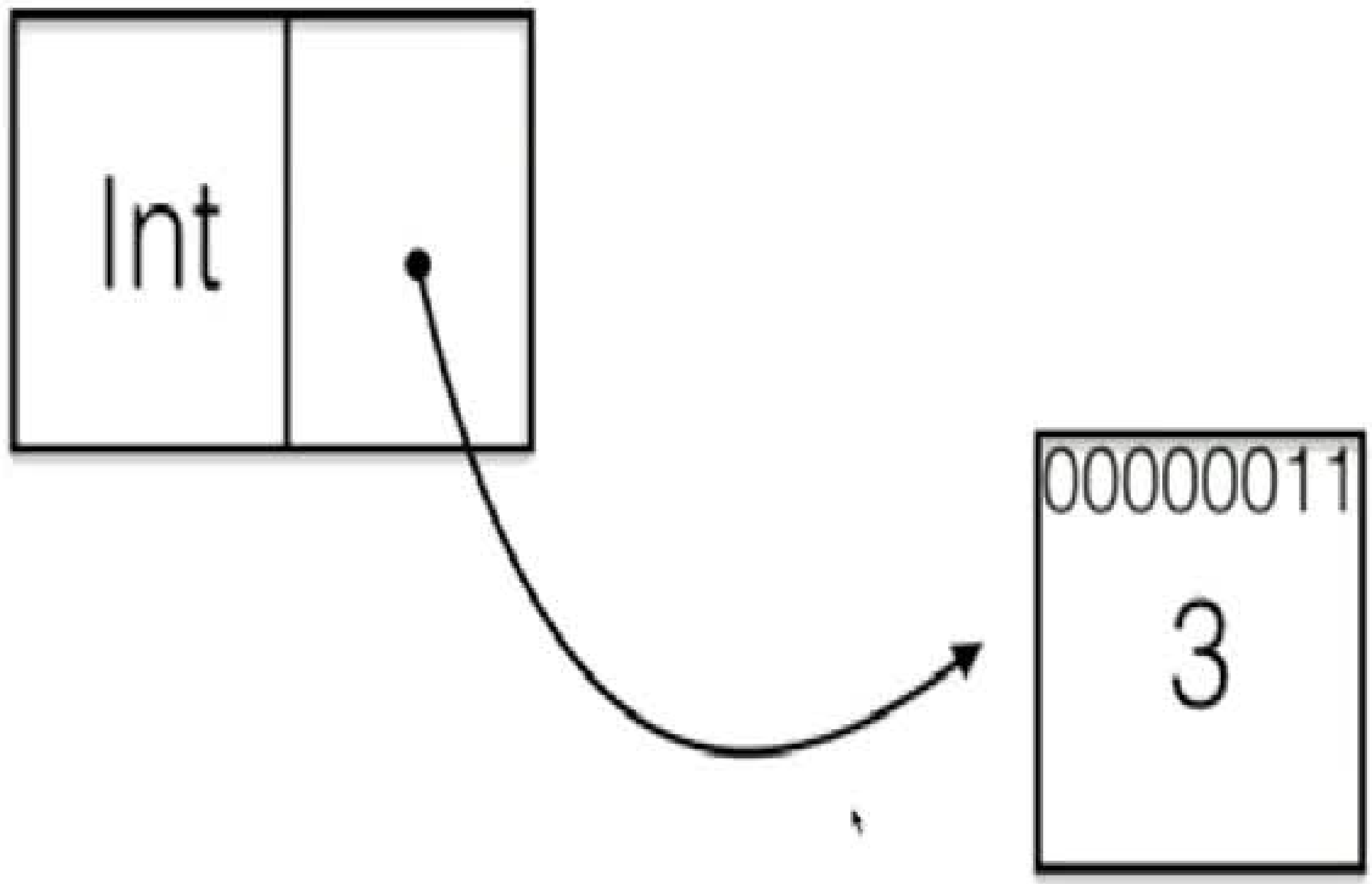
This leads to inefficient code (at least currently), since the result will be **boxed**:



This leads to a significant performance loss:

Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

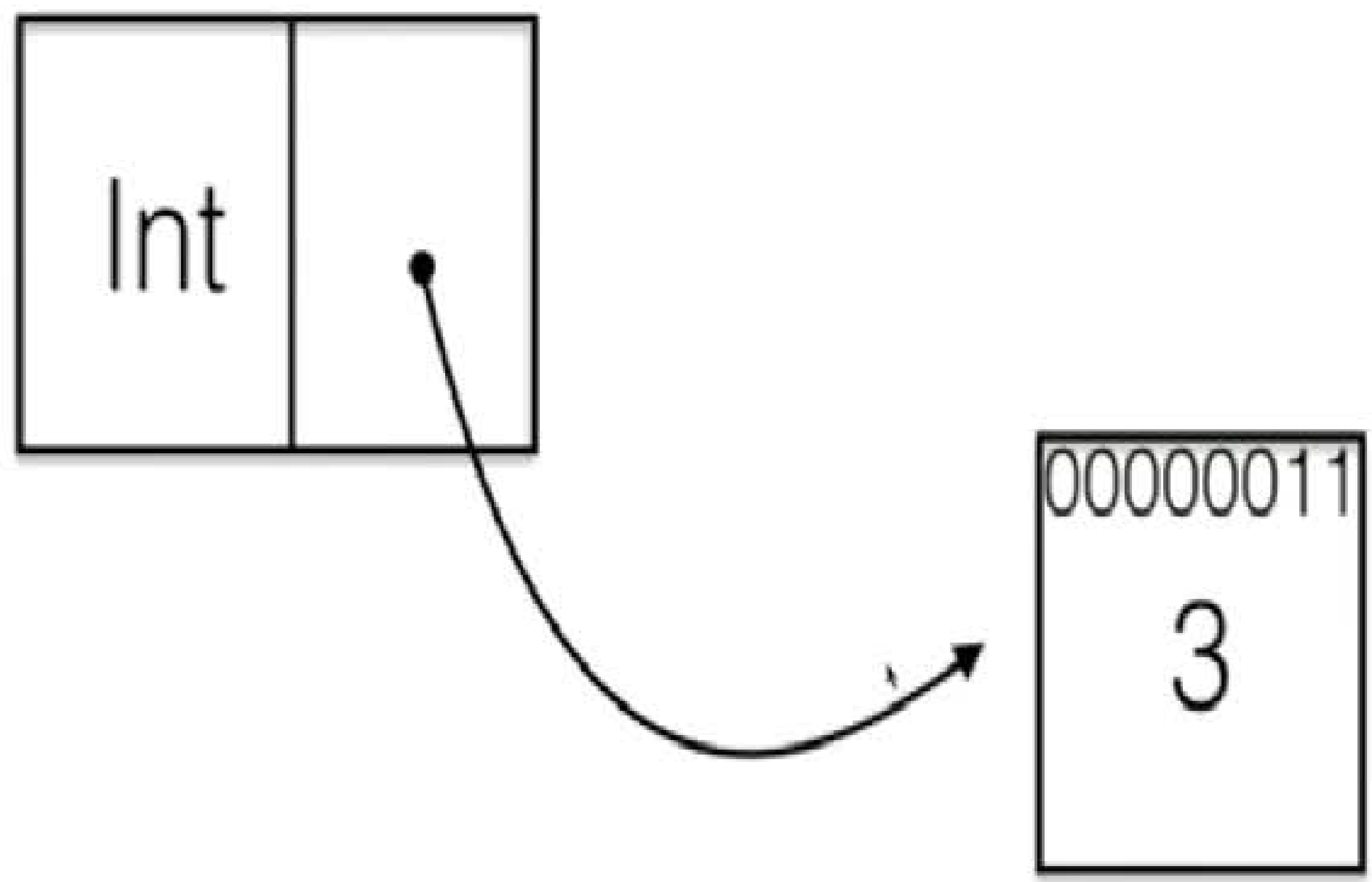
This leads to inefficient code (at least currently), since the result will be **boxed**:



This leads to a significant performance loss:

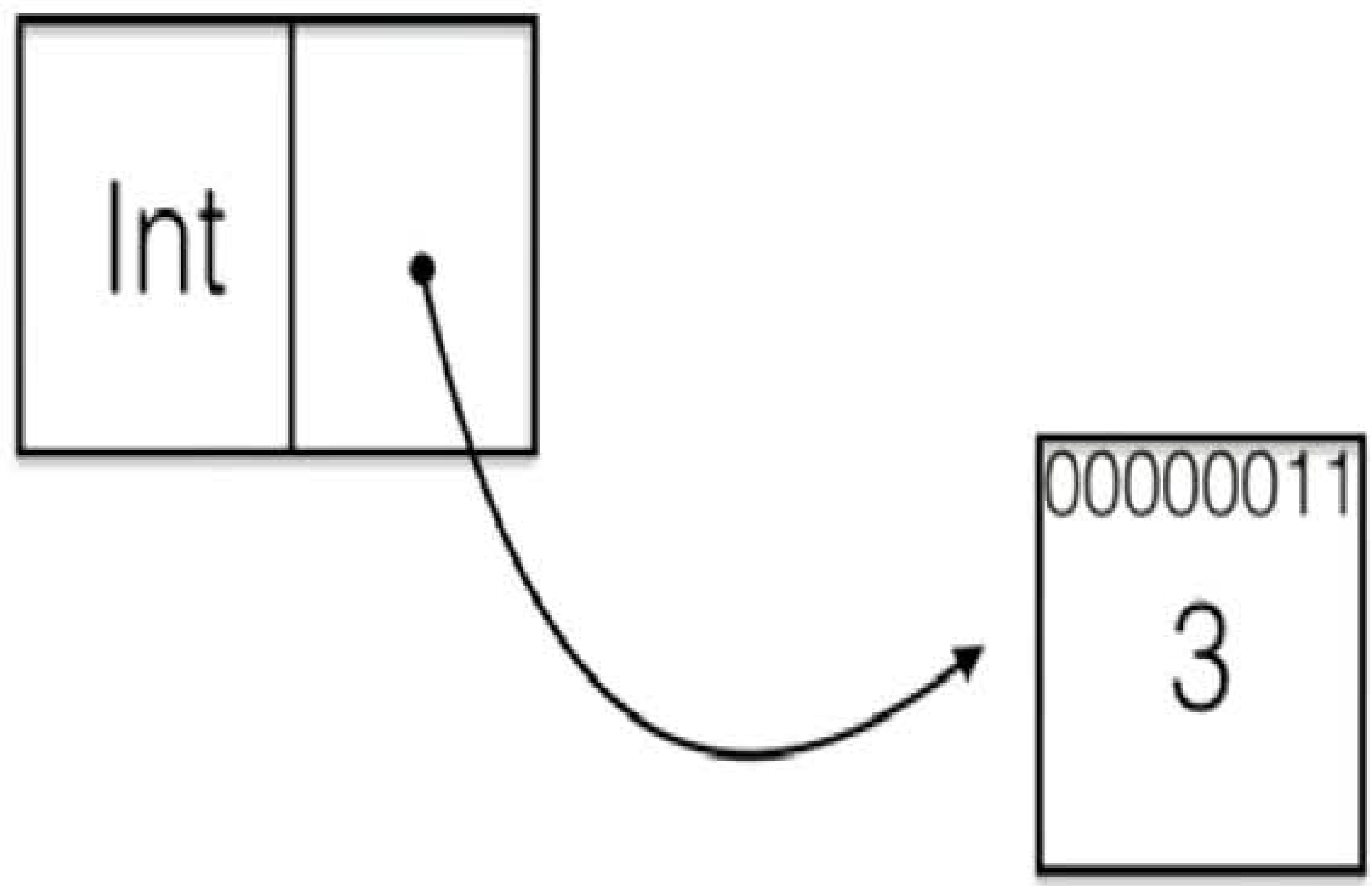
Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

This leads to inefficient code (at least currently), since the result will be **boxed**:



Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

This leads to inefficient code (at least currently), since the result will be **boxed**:



This leads to a significant performance loss:

In [31]:
```
step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

step2() = randn() > 0 ? 1 : -1    # only Int64
```

```
WARNING: Method definition step1() in module Main at In[29]:1 overwr
itten at In[31]:1.
WARNING: Method definition step2() in module Main at In[29]:3 overwr
itten at In[31]:3.
```

Out[31]: `step2 (generic function with 1 method)`

First, compile the functions by running each once:

In [32]:
```
step1()
step2()
```

Out[32]: `1`

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: `-188.0`

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
  0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
  0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [36]:
```
@code_typed step1()
```

```
#temp#@_6 = 0
goto 27
25:
#temp#@_6 = false
27:
# meta: pop location
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
Out[32]: 1

In [33]: @time sum(step1() for i in 1:10^7)   # generator

  0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)

Out[33]: -188.0

In [34]: @time sum(step2() for i in 1:10^7)

  0.103496 seconds (16.22 k allocations: 697.330 KB)

Out[34]: 5764
```

Excess allocations are usually a sign of type instability.

```
In [36]: @code_typed step1()

        #temp#@_6 = 0
        goto 27
        25:
        #temp#@_6 = false
        27:
        # meta: pop location
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
step2()
```

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)    # generator
```

```
  0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
  0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [36]:
```
@code_typed step1()
```

```
        #temp#@_6 = 0
        goto 27
        25:
        #temp#@_6 = false
        27:
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [36]:
```julia
@code_typed step1()
```

```
          25:
          #temp#@_6 = false
          27:
          # meta: pop location
          $(Expr(:inbounds, :pop))
          #temp#@_6
          $(Expr(:inbounds, false))
          # meta: location random.jl rand_ui52_raw_inbounds 124
          $(Expr(:inbounds, false))
          # meta: location random.jl rand_inbounds 117
          $(Expr(:inbounds, false))
          # meta: location random.jl mt_pop! 104
          $(Expr(:inbounds, true))
          SSAValue(2) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
          SSAValue(3) = (Base.box)(Int64,(Base.add_int)((Core.getfiel
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        # meta: pop location
        $(Expr(:inbounds, :pop))
        #temp#@_6
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw_inbounds 124
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_inbounds 117
        $(Expr(:inbounds, false))
        # meta: location random.jl mt_pop! 104
        $(Expr(:inbounds, true))
        SSAValue(2) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
        SSAValue(3) = (Base.box)(Int64,(Base.add_int)((Core.getfiel
d)(Base.Random.GLOBAL_RNG,:idx)::Int64,1))
        (Core.setfield!)(Base.Random.GLOBAL_RNG,:idx,SSAValue(3))::I
nt64
        #temp#@_7 = (Base.arrayref)(SSAValue(2),SSAValue(3))::Float6
```

In [38]:
```
@code_warntype step1()
```

```
        $(Expr(:inbounds, :pop))
        # meta: pop location
        $(Expr(:inbounds, :pop))
        $(Expr(:inbounds, false))
        # meta: location operators.jl > 64
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        $(Expr(:inbounds, :pop))
        #temp#@_6
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_ui52_raw_inbounds 124
        $(Expr(:inbounds, false))
        # meta: location random.jl rand_inbounds 117
        $(Expr(:inbounds, false))
        # meta: location random.jl mt_pop! 104
        $(Expr(:inbounds, true))
        SSAValue(2) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
        SSAValue(3) = (Base.box)(Int64,(Base.add_int)((Core.getfiel
d)(Base.Random.GLOBAL_RNG,:idx)::Int64,1))
        (Core.setfield!)(Base.Random.GLOBAL_RNG,:idx,SSAValue(3))::I
nt64
        #temp#@_7 = (Base.arrayref)(SSAValue(2),SSAValue(3))::Float6
```

In [38]: `@code_warntype step1()`

```
        $(Expr(:inbounds, :pop))
        # meta: pop location
        $(Expr(:inbounds, :pop))
        $(Expr(:inbounds, false))
        # meta: location operators.jl > 64
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        # meta: location operators.jl > 64
        # meta: location float.jl < 323
        fx::Float64 = (Base.box)(Float64,(Base.sitofp)(Float64,0))
        # meta: pop location
        # meta: pop location
        $(Expr(:inbounds, :pop))
        unless (Base.box)(Base.Bool,(Base.or_int)((Base.lt_float)(fx::
Float64,#temp#@_8::Float64)::Bool,(Base.box)(Base.Bool,(Base.and_in
t)((Base.eq_float)(fx::Float64,#temp#@_8::Float64)::Bool,(Base.box)
(Base.Bool,(Base.or_int)((Base.eq_float)(fx::Float64,9.223372036854
776e18)::Bool,(Base.slt_int)(0,(Base.box)(Int64,(Base.fptosi)(Int64,f
x::Float64)))::Bool)))))) goto 85
        return 1
        85:
        return -1.0
    end::Union{Float64,Int64}
```

5.2 Inlining

In [ ]:

```
h1(x) = 3x
h2(x) = h1(5x)
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
Float64,#temp#@_8::Float64)::Bool,(Base.box)(Base.Bool,(Base.and_in
t)((Base.eq_float)(fx::Float64,#temp#@_8::Float64)::Bool,(Base.box)
(Base.Bool,(Base.or_int)((Base.eq_float)(fx::Float64,9.223372036854776e18)::Bool,(Base.slt_int)(0,(Base.box)(Int64,(Base.fptosi)(Int64,f
x::Float64)))::Bool)))))) goto 85
      return 1
      85:
      return -1.0
  end::Union{Float64,Int64}
```

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

In [ ]:
```
@code_lowered h2(10)
```

In [ ]:
```
@code_typed h2(10)
```

In [ ]:
```
@code_llvm h2(10)
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        # meta: location float.jl < 325
        fx::Float64 = (Base.box)(Float64,(Base.sitofp)(Float64,0))
        # meta: pop location
        # meta: pop location
        $(Expr(:inbounds, :pop))
        unless (Base.box)(Base.Bool,(Base.or_int)((Base.lt_float)(fx::Float64,#temp#@_8::Float64)::Bool,(Base.box)(Base.Bool,(Base.and_int)((Base.eq_float)(fx::Float64,#temp#@_8::Float64)::Bool,(Base.box)(Base.Bool,(Base.or_int)((Base.eq_float)(fx::Float64,9.22337203685477 6e18)::Bool,(Base.slt_int)(0,(Base.box)(Int64,(Base.fptosi)(Int64,fx::Float64)))::Bool)))))) goto 85
        return 1
        85:
        return -1.0
    end::Union{Float64,Int64}
```

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

In [ ]:
```
@code_lowered h2(10)
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
itten at In[31]:3.
```

Out[31]:
```
step2 (generic function with 1 method)
```

First, compile the functions by running each once:

In [32]:
```julia
step1()
step2()
```

Out[32]:
```
1
```

In [33]:
```julia
@time sum(step1() for i in 1:10^7)    # generator
```

```
  0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]:
```
-188.0
```

In [34]:
```julia
@time sum(step2() for i in 1:10^7)
```

```
  0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]:
```
5764
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
step2() = rand() > 0 ? 1 : -1    # only Int64
```

```
WARNING: Method definition step1() in module Main at In[29]:1 overwr
itten at In[31]:1.
WARNING: Method definition step2() in module Main at In[29]:3 overwr
itten at In[31]:3.
```

Out[31]: step2 (generic function with 1 method)

First, compile the functions by running each once:

In [32]:
```
step1()
step2()
```

Out[32]: 1

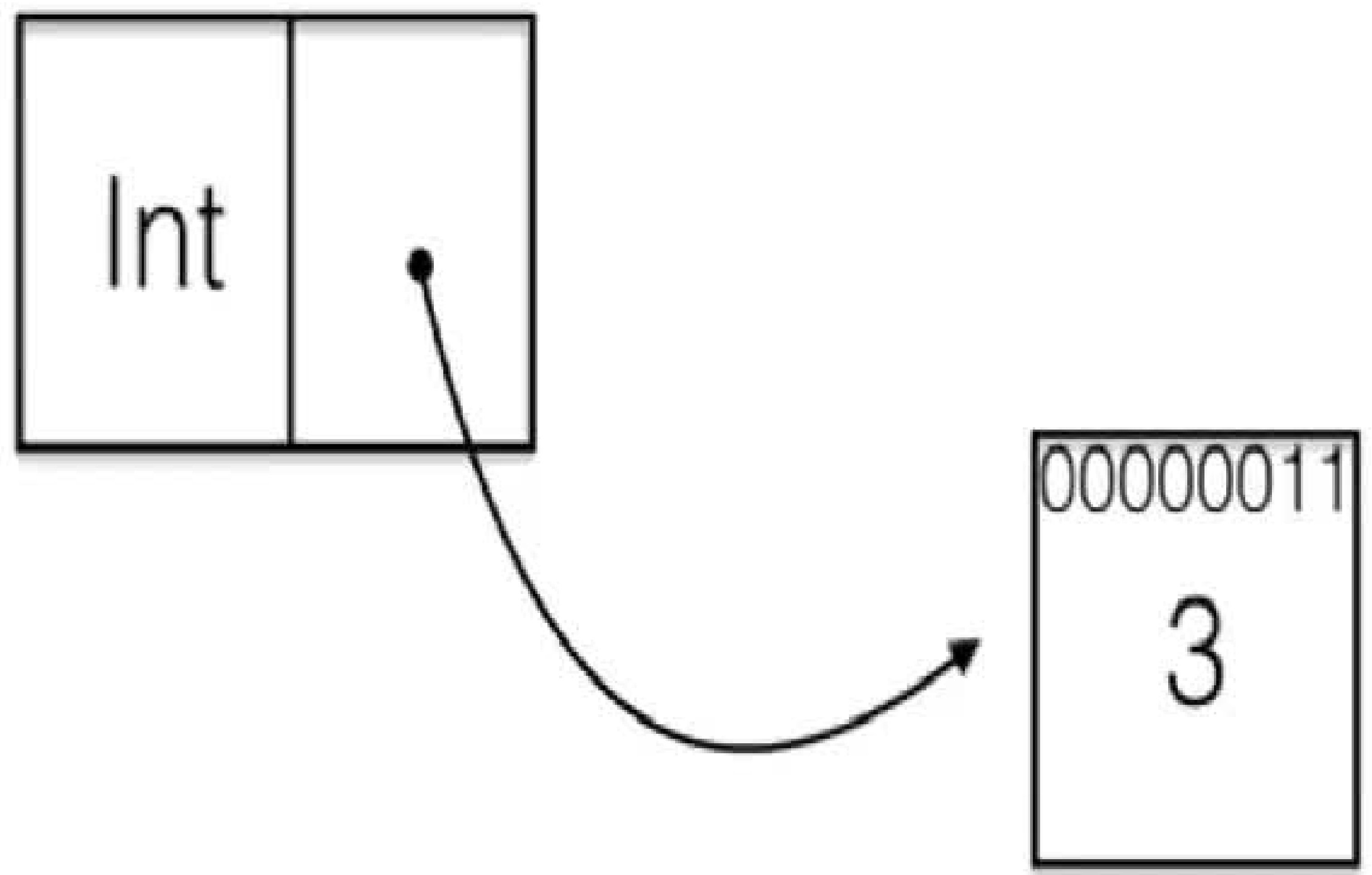
In [33]:
```
@time sum(step1() for i in 1:10^7)    # generator
```

```
0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0



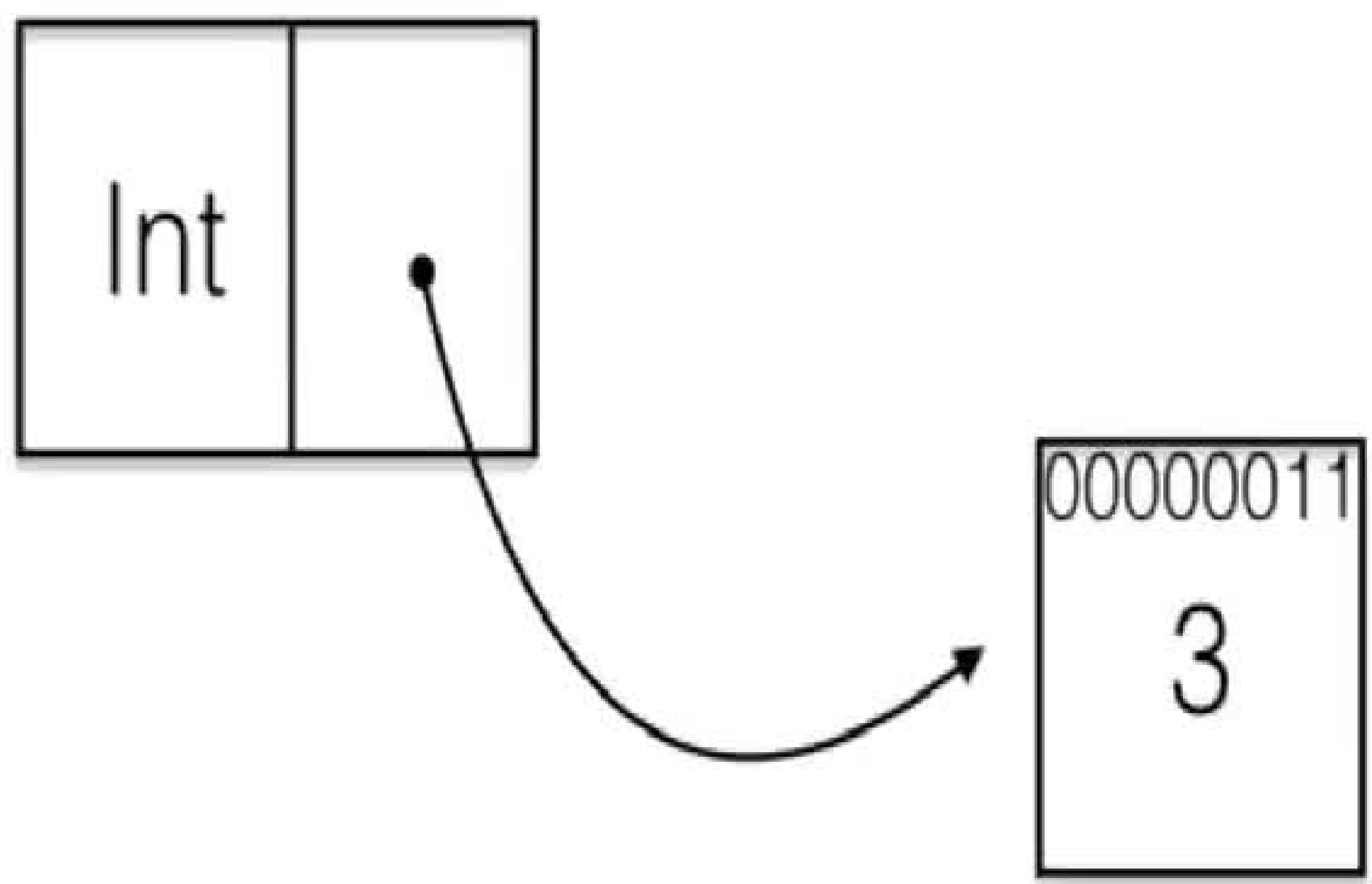
This leads to a significant performance loss:

In [31]:
```
step1() = randn() > 0 ? 1 : -1.0   # Int64 and Float64
step2() = randn() > 0 ? 1 : -1     # only Int64
```

```
WARNING: Method definition step1() in module Main at In[29]:1 overwr
```

Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

This leads to inefficient code (at least currently), since the result will be **boxed**:



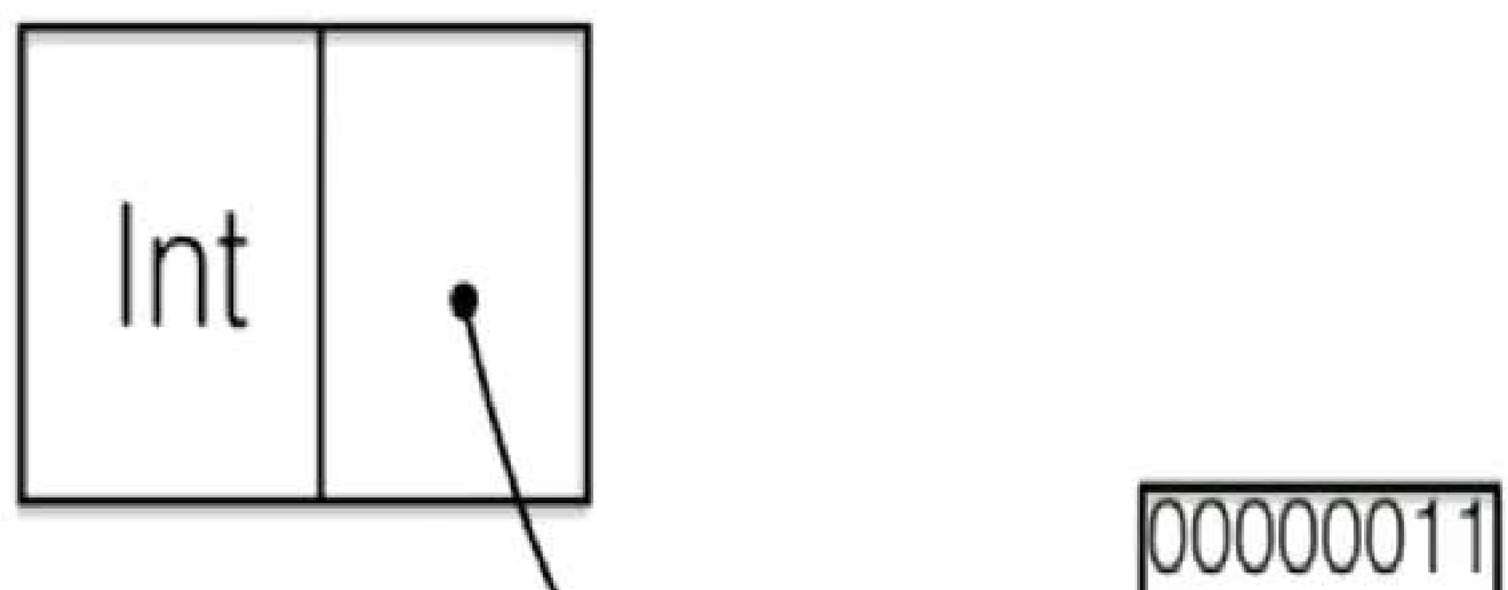
This leads to a significant performance loss:

Out[25]: LambdaInfo template for step1() at In[24]:1

```
:(begin
        nothing
        unless (Main.rand)((Main.colon)(-1,1)) > 0 goto 4
        return 1
        4:
        return -1.0
    end)
```

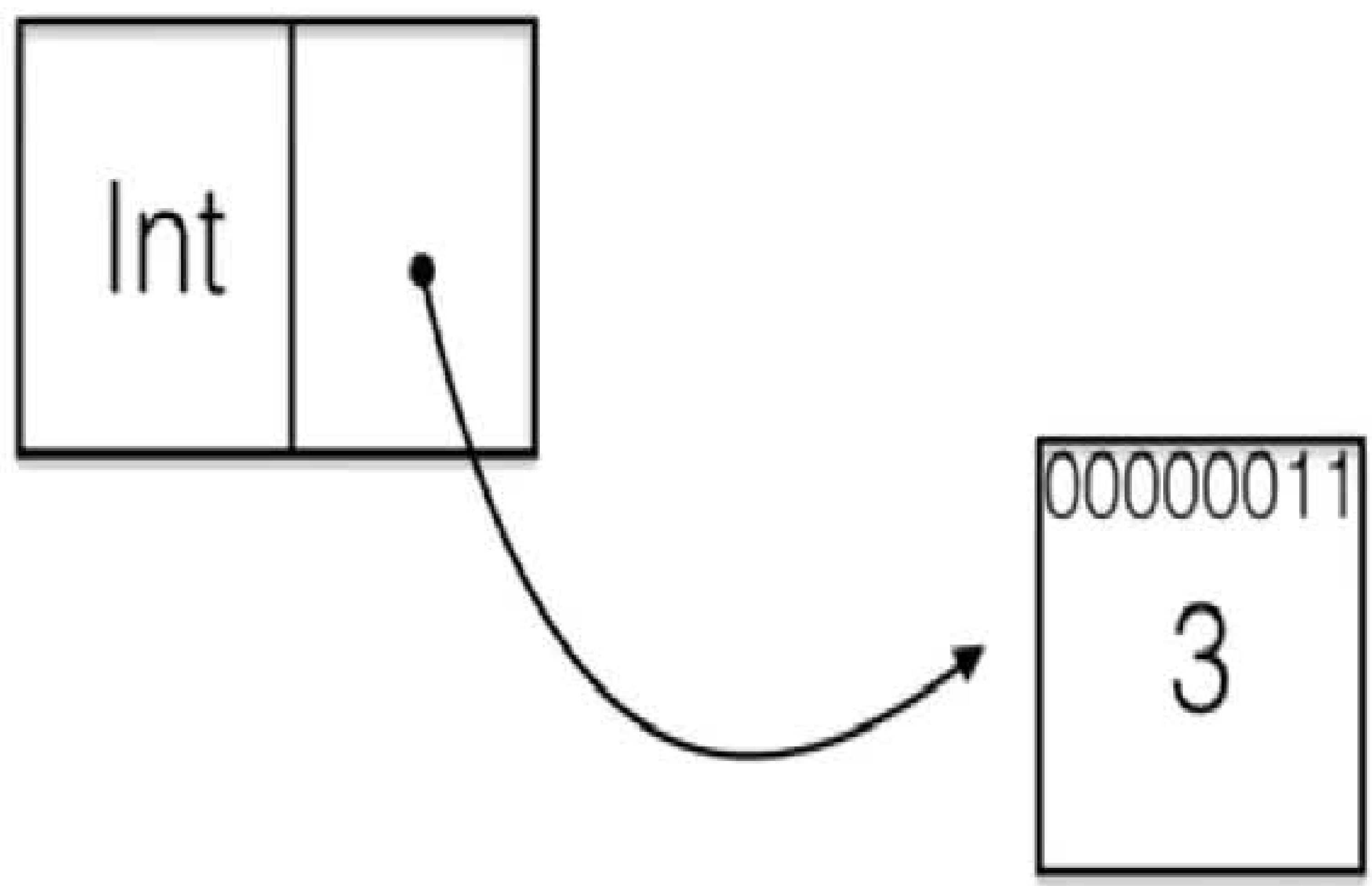
Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

This leads to inefficient code (at least currently), since the result will be **boxed**:

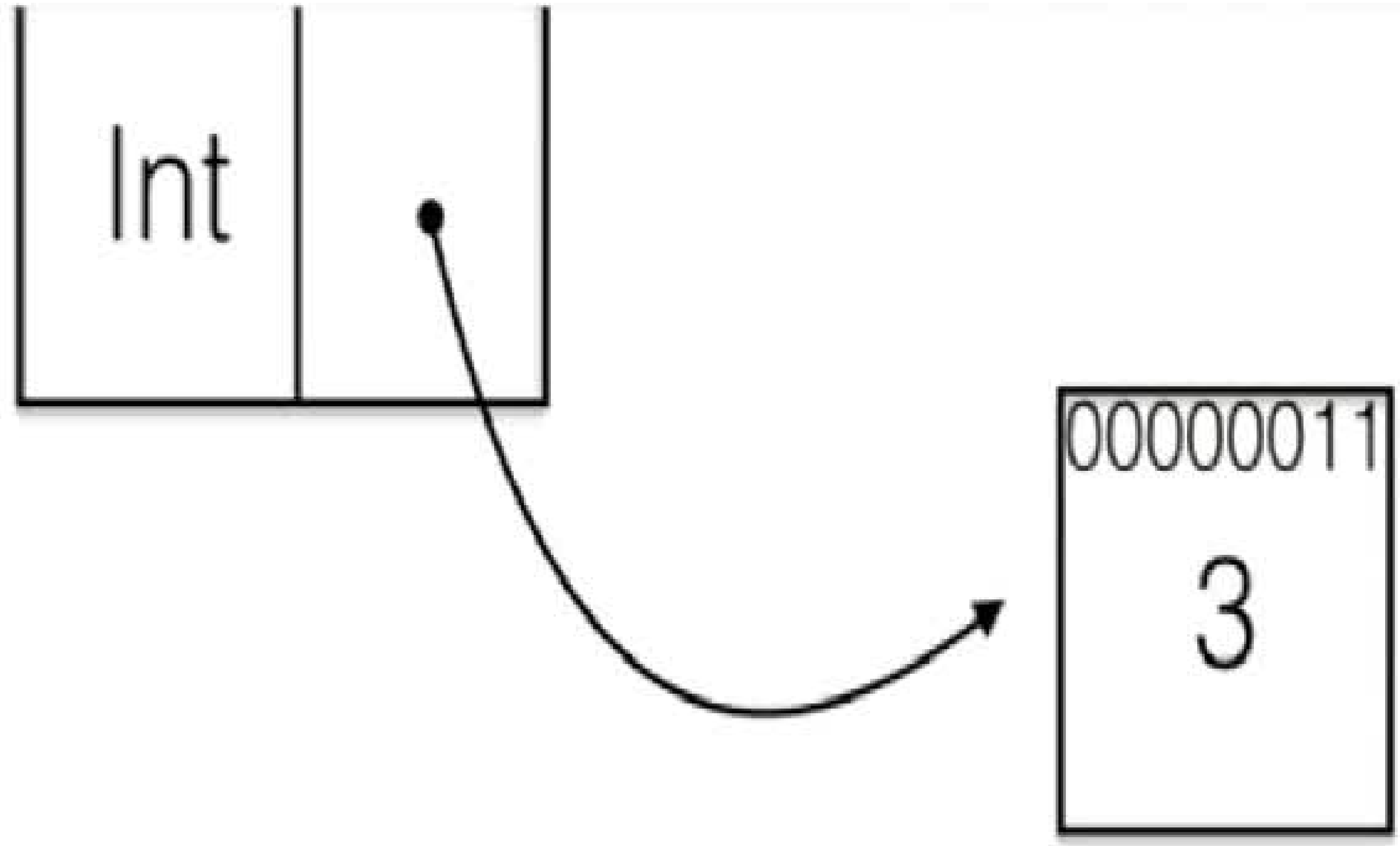


Julia realises that the output of the function can either be an `Int64` or a `Float64`, i.e. there is a **type instability**.

This leads to inefficient code (at least currently), since the result will be **boxed**:



This leads to a significant performance loss:

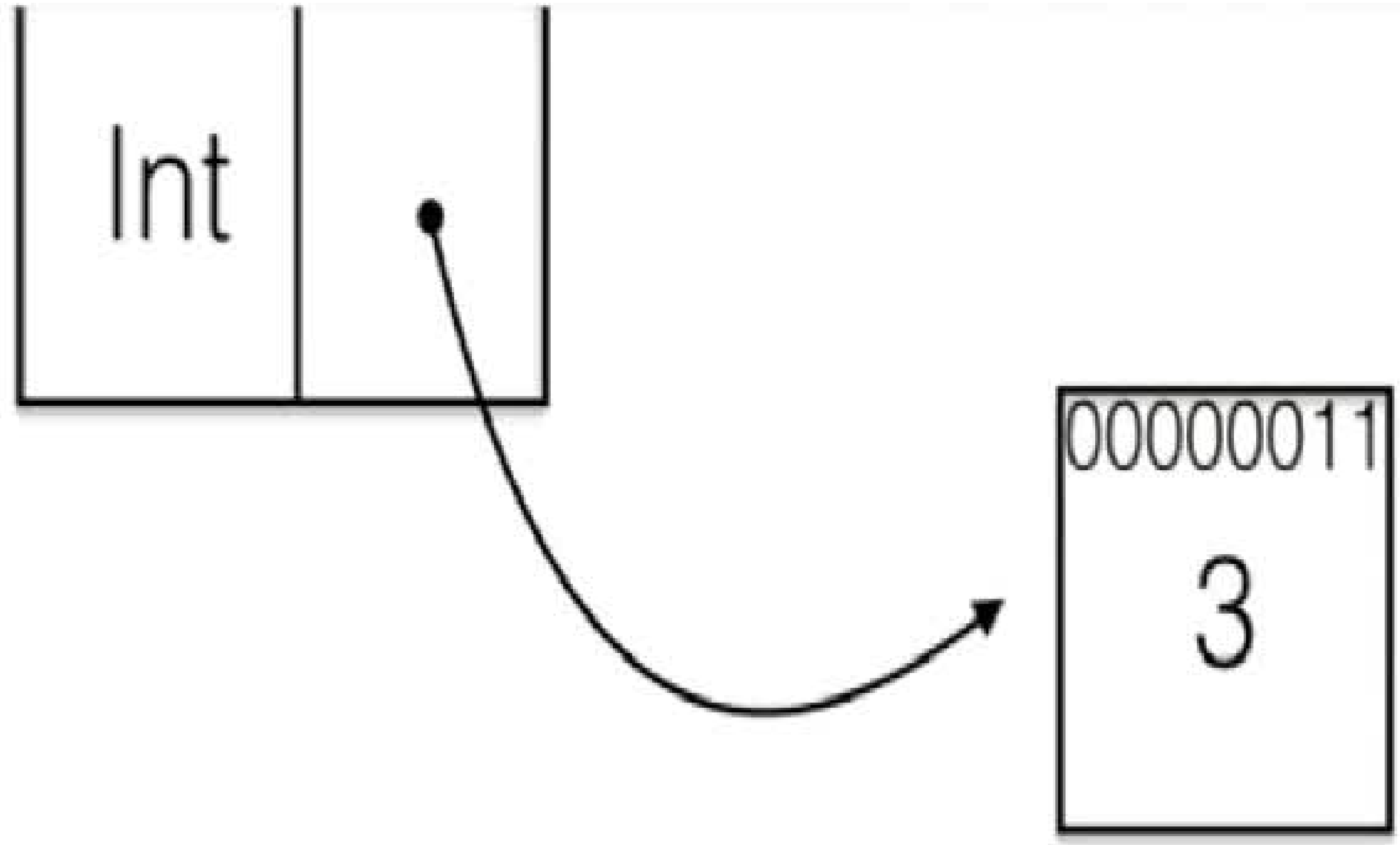


This leads to a significant performance loss:

In [31]:
```
step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

step2() = randn() > 0 ? 1 : -1    # only Int64
```

```
WARNING: Method definition step1() in module Main at In[29]:1 overwr
itten at In[31]:1.
WARNING: Method definition step2() in module Main at In[29]:3 overwr
itten at In[31]:3.
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0



This leads to a significant performance loss:

In [31]:
```
step1() = randn() > 0 ? 1 : -1.0  # Int64 and Float64

step2() = randn() > 0 ? 1 : -1    # only Int64
```

```
WARNING: Method definition step1() in module Main at In[29]:1 overwr
itten at In[31]:1.
WARNING: Method definition step2() in module Main at In[29]:3 overwr
itten at In[31]:3.
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

First, compile the functions by running each once:

In [32]:
```
step1()
step2()
```

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [36]:
```
@code_typed step1()
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

In [32]:
```
step1()
step2()
```

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [36]:
```
@code_typed step1()
```

```
25:
#temp#@_6 = false
27:
# meta: pop location
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
step2()
```

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
  0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
  0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [36]:
```
@code_typed step1()
```

```
25:
#temp#@_6 = false
27:
# meta: pop location
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
step2()
```

Out[32]: 1

In [33]:
```
@time sum(step1() for i in 1:10^7)   # generator
```

```
  0.497668 seconds (15.02 M allocations: 229.910 MB, 2.91% gc time)
```

Out[33]: -188.0

In [34]:
```
@time sum(step2() for i in 1:10^7)
```

```
  0.103496 seconds (16.22 k allocations: 697.330 KB)
```

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [36]:
```
@code_typed step1()
```

```
        25:
        #temp#@_6 = false
        27:
        # meta: pop location
        $(Expr(:inbounds, :pop))
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Out[34]: 5764

Excess allocations are usually a sign of type instability.

In [36]: `@code_typed step1()`

```
            25:
            #temp#@_6 = false
            27:
            # meta: pop location
            $(Expr(:inbounds, :pop))
            #temp#@_6
            $(Expr(:inbounds, false))
            # meta: location random.jl rand_ui52_raw_inbounds 124
            $(Expr(:inbounds, false))
            # meta: location random.jl rand_inbounds 117
            $(Expr(:inbounds, false))
            # meta: location random.jl mt_pop! 104
            $(Expr(:inbounds, true))
            SSAValue(2) = (Core.getfield)(Base.Random.GLOBAL_RNG,:val
s)::Array{Float64,1}
            SSAValue(3) = (Base.box)(Int64,(Base.add_int)((Core.getfiel
d)(Base.Random.GLOBAL_RNG,:idx)::Int64,1))
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
    return -1.0
  end::Union{Float64,Int64}
```

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

In [ ]:
```
@code_lowered h2(10)
```

In [ ]:
```
@code_typed h2(10)
```

In [ ]:
```
@code_llvm h2(10)
```

Julia has **inlined** the function h1 into h2.

6 User-defined types

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

5.2 Inlining

In [ ]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

In [ ]:
```
@code_lowered h2(10)
```

In [ ]:
```
@code_typed h2(10)
```

In [ ]:
```
@code_llvm h2(10)
```

Julia has **inlined** the function h1 into h2.

6 User-defined types

It is common to define new types. For example, we can define a type to automatically track

5.2 Inlining

In [39]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

Out[39]:
```
h2 (generic function with 1 method)
```

In [40]:
```
@code_lowered h2(10)
```

Out[40]:
```
LambdaInfo template for h2(x) at In[39]:2
:(begin
        nothing
        return (Main.h1)(5 * x)
    end)
```

In [ ]:
```
@code_typed h2(10)
```

In [ ]:
```
@code_llvm h2(10)
```

Julia has **inlined** the function h1 into h2

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

5.2 Inlining

In [39]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

Out[39]:
```
h2 (generic function with 1 method)
```

In [40]:
```
@code_lowered h2(10)
```

Out[40]:
```
LambdaInfo template for h2(x) at In[39]:2
:(begin
        nothing
        return (Main.h1)(5 * x)
    end)
```

In [ ]:
```
@code_typed h2(10)
```

In [ ]:
```
@code_llvm h2(10)
```

Julia has **inlined** the function h1 into h2.

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

In [39]:
```
h1(x) = 3x
h2(x) = h1(5x)
```

Out[39]:
```
h2 (generic function with 1 method)
```

In [40]:
```
@code_lowered h2(10)
```

Out[40]:
```
LambdaInfo template for h2(x) at In[39]:2
:(begin
        nothing
        return (Main.h1)(5 * x)
    end)
```

In [41]:
```
@code_typed h2(10)
```

Out[41]:
```
LambdaInfo for h2(::Int64)
:(begin
        return (Base.box)(Int64,(Base.mul_int)(3,(Base.box)(Int64,(B
ase.mul_int)(5,x))))
    end::Int64)
```

In [ ]:
```
@code_llvm h2(10)
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

In [40]:
```
@code_lowered h2(10)
```

Out[40]:
```
LambdaInfo template for h2(x) at In[39]:2
:(begin
        nothing
        return (Main.h1)(5 * x)
    end)
```

In [41]:
```
@code_typed h2(10)
```

Out[41]:
```
LambdaInfo for h2(::Int64)
:(begin
        return (Base.box)(Int64,(Base.mul_int)(3,(Base.box)(Int64,(B
ase.mul_int)(5,x))))
    end::Int64)
```

In [42]:
```
@code_llvm h2(10)
```

```
define i64 @julia_h2_71906(i64) #0 {
top:
  %1 = mul i64 %0, 15
  ret i64 %1
}
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        return (Main.h1)(5 * x)
    end)
```

In [41]:
```
@code_typed h2(10)
```

Out[41]:
```
LambdaInfo for h2(::Int64)
:(begin
        return (Base.box)(Int64,(Base.mul_int)(3,(Base.box)(Int64,(B
ase.mul_int)(5,x))))
    end::Int64)
```

In [42]:
```
@code_llvm h2(10)
```

```
define i64 @julia_h2_71906(i64) #0 {
top:
  %1 = mul i64 %0, 15
  ret i64 %1
}
```

Julia has **inlined** the function h1 into h2.

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
    end)
```

In [41]:
```
@code_typed h2(10)
```

Out[41]:
```
LambdaInfo for h2(::Int64)
:(begin
        return (Base.box)(Int64,(Base.mul_int)(3,(Base.box)(Int64,(B
ase.mul_int)(5,x))))
    end::Int64)
```

In [42]:
```
@code_llvm h2(10)
```

```
define i64 @julia_h2_71906(i64) #0 {
top:
  %1 = mul i64 %0, 15
  ret i64 %1
}
```

Julia has **inlined** the function h1 into h2.

6 User-defined types

In [41]:
```
@code_typed h2(10)
```

Out[41]:
```
LambdaInfo for h2(::Int64)
:(begin
        return (Base.box)(Int64,(Base.mul_int)(3,(Base.box)(Int64,(B
ase.mul_int)(5,x))))
    end::Int64)
```

In [42]:
```
@code_llvm h2(10)
```

```
define i64 @julia_h2_71906(i64) #0 {
top:
  %1 = mul i64 %0, 15
  ret i64 %1
}
```

Julia has **inlined** the function h1 into h2.

6 User-defined types

In [41]:
```
@code_typed h2(10)
```

Out[41]:
```
LambdaInfo for h2(::Int64)
:(begin
        return (Base.box)(Int64,(Base.mul_int)(3,(Base.box)(Int64,(Base.mul_int)(5,x))))
    end::Int64)
```

In [42]:
```
@code_llvm h2(10)
```

```
define i64 @julia_h2_71906(i64) #0 {
top:
  %1 = mul i64 %0, 15
  ret i64 %1
}
```

Julia has **inlined** the function h1 into h2.

6 User-defined types

```
define i64 @julia_h2_71906(i64) #0 {
top:
  %1 = mul i64 %0, 15
  ret i64 %1
}
```

Julia has **inlined** the function h1 into h2.

6 User-defined types

It is common to define new types. For example, we can define a type to automatically track the derivative through a calculation:

In [ ]:
```
immutable Dual    # use `struct` in Julia v0.6 instead of `immutable`
    value::Float64
    derivative::Float64
end
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Julia has **inlined** the function h1 into h2.

6 User-defined types

It is common to define new types. For example, we can define a type to automatically track the derivative through a calculation:

In [ ]:
```
immutable Dual     # use `struct` in Julia v0.6 instead of `immutable`
    value::Float64
    derivative::Float64
end
```

Now we can define arithmetic:

In [ ]:
```
import Base: +, *

+(a::Dual, b::Dual) = Dual(a.value + b.value, a.derivative + b.deriva
*(a::Dual, b::Dual) = Dual(a.value*b.value, a.derivative*b.value + b.
*(a::Number, b::Dual) = Dual(a*b.value, a*b.derivative)
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

6 User-defined types

It is common to define new types. For example, we can define a type to automatically track the derivative through a calculation:

In [43]:
```
immutable Dual     # use `struct` in Julia v0.6 instead of `immutable`
    value::Float64
    derivative::Float64
end
```

Now we can define arithmetic:

In [ ]:
```
import Base: +, *

+(a::Dual, b::Dual) = Dual(a.value + b.value, a.derivative + b.deriva
*(a::Dual, b::Dual) = Dual(a.value*b.value, a.derivative*b.value + b.
*(a::Number, b::Dual) = Dual(a*b.value, a*b.derivative)
```

In [ ]:
```
a = Dual(1, 2)
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

It is common to define new types. For example, we can define a type to automatically track the derivative through a calculation:

In [43]:
```
immutable Dual     # use `struct` in Julia v0.6 instead of `immutable`
    value::Float64
    derivative::Float64
end
```

Now we can define arithmetic:

In [44]:
```
import Base: +, *

+(a::Dual, b::Dual) = Dual(a.value + b.value, a.derivative + b.deriva
*(a::Dual, b::Dual) = Dual(a.value*b.value, a.derivative*b.value + b.
*(a::Number, b::Dual) = Dual(a*b.value, a*b.derivative)
```

Out[44]:
```
* (generic function with 151 methods)
```

In [ ]:
```
a = Dual(1, 2)
a + a
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

In [43]:
```
immutable Dual     # use `struct` in Julia v0.6 instead of `immutable`
    value::Float64
    derivative::Float64
end
```

Now we can define arithmetic:

In [44]:
```
import Base: +, *

+(a::Dual, b::Dual) = Dual(a.value + b.value, a.derivative + b.deriva
*(a::Dual, b::Dual) = Dual(a.value*b.value, a.derivative*b.value + b.
*(a::Number, b::Dual) = Dual(a*b.value, a*b.derivative)
```

Out[44]:
```
* (generic function with 151 methods)
```

In [ ]:
```
a = Dual(1, 2)
a + a
```

Julia is written with **generic code** when possible:

In [ ]:
```
M = [1 2; 3 4]
```

Julia 0.5.0 O

File Edit View Insert Cell Kernel Navigate Widgets Help

In [43]:
```
immutable Dual    # use `struct` in Julia v0.6 instead of `immutable`
    value::Float64
    derivative::Float64
end
```

Now we can define arithmetic:

In [44]:
```
import Base: +, *

+(a::Dual, b::Dual) = Dual(a.value + b.value, a.derivative + b.deriva
*(a::Dual, b::Dual) = Dual(a.value*b.value, a.derivative*b.value + b.
*(a::Number, b::Dual) = Dual(a*b.value, a*b.derivative)
```

Out[44]:
```
* (generic function with 151 methods)
```

In [ ]:
```
a = Dual(1, 2)
a + a
```

Julia is written with **generic code** when possible:

Notebook saved | Julia 0.5.0

File Edit View Insert Cell Kernel Navigate Widgets Help

```
    value::Float64
    derivative::Float64
end
```

Now we can define arithmetic:

In [44]:
```
import Base: +, *

+(a::Dual, b::Dual) = Dual(a.value + b.value, a.derivative + b.deriva
*(a::Dual, b::Dual) = Dual(a.value*b.value, a.derivative*b.value + b.
*(a::Number, b::Dual) = Dual(a*b.value, a*b.derivative)
```

Out[44]: `* (generic function with 151 methods)`

In [ ]:
```
a = Dual(1, 2)
a + a
```

Julia is written with **generic code** when possible:

```
    value::Float64
    derivative::Float64
end
```

Now we can define arithmetic:

In [44]:
```
import Base: +, *

+(a::Dual, b::Dual) = Dual(a.value + b.value, a.derivative + b.deriva
*(a::Dual, b::Dual) = Dual(a.value*b.value, a.derivative*b.value + b.
*(a::Number, b::Dual) = Dual(a*b.value, a*b.derivative)
```

Out[44]:
```
* (generic function with 151 methods)
```

In [ ]:
```
a = Dual(1, 2)
a + a
```

Julia is written with **generic code** when possible:

In [ ]:
```
M = [1 2; 3 4]
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
In [43]: immutable Dual    # use "struct" in Julia v0.6 instead of "immutable"
             value::Float64
             derivative::Float64
         end
```

Now we can define arithmetic:

```
In [ ]: +
```

```
In [44]: import Base: +, *

         +(a::Dual, b::Dual) = Dual(a.value + b.value, a.derivative + b.deriva
         *(a::Dual, b::Dual) = Dual(a.value*b.value, a.derivative*b.value + b.
         *(a::Number, b::Dual) = Dual(a*b.value, a*b.derivative)

Out[44]: * (generic function with 151 methods)
```

```
In [ ]: a = Dual(1, 2)
        a + a
```

Julia is written with **generic code** when possible:

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

In [43]:
```
immutable Dual     # use "struct" in Julia v0.6 instead of "immutable"
    value::Float64
    derivative::Float64
end
```

Now we can define arithmetic:

In [45]:
```
+
```

Out[45]: + (generic function with 164 methods)

In [44]:
```
import Base: +, *

+(a::Dual, b::Dual) = Dual(a.value + b.value, a.derivative + b.deriva
*(a::Dual, b::Dual) = Dual(a.value*b.value, a.derivative*b.value + b.
*(a::Number, b::Dual) = Dual(a*b.value, a*b.derivative)
```

Out[44]: * (generic function with 151 methods)

In [ ]:
```
a = Dual(1, 2)
a + a
```

Julia is written with generic code when possible:

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

In [43]:
```
immutable Dual    # use `struct` in Julia v0.6 instead of `immutable`
    value::Float64
    derivative::Float64
end
```

Now we can define arithmetic:

In [45]:
```
+
```

Out[45]: + (generic function with 164 methods)

In [*]:
```
methods(+)
```

In [44]:
```
import Base: +, *

+(a::Dual, b::Dual) = Dual(a.value + b.value, a.derivative + b.deriva
*(a::Dual, b::Dual) = Dual(a.value*b.value, a.derivative*b.value + b.
*(a::Number, b::Dual) = Dual(a*b.value, a*b.derivative)
```

Out[44]: * (generic function with 151 methods)

In [ ]:
```
a = Dual(1, 2)
```

In [43]:
```julia
immutable Dual    # use `struct` in Julia v0.6 instead of `immutable`
    value::Float64
    derivative::Float64
end
```

Now we can define arithmetic:

In [45]:
```julia
+
```

Out[45]: + (generic function with 164 methods)

In [46]:
```julia
methods(+)
```

Out[46]: 164 methods for generic function +:

- +(x::**Bool**, z::**Complex{Bool}**) at complex.jl:136
- +(x::**Bool**, y::**Bool**) at bool.jl:48
- +(x::**Bool**) at bool.jl:45
- +*{T<:AbstractFloat}*(x::**Bool**, y::**T**) at bool.jl:55
- +(x::**Bool**, z::**Complex**) at complex.jl:143
- +(x::**Bool**, A::**AbstractArray{Bool,N<:Any}**) at arraymath.jl:91
- +(x::**Float32**, y::**Float32**) at float.jl:239

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

```julia
immutable Dual    # use `struct` in Julia v0.6 instead of `immutable`
    value::Float64
    derivative::Float64
end
```

Now we can define arithmetic:

In [45]:
```julia
+
```

Out[45]:
```
+ (generic function with 164 methods)
```

In [46]:
```julia
methods(+)
```

Out[46]:

164 methods for generic function +:

- +(x::**Bool**, z::**Complex{Bool}**) at complex.jl:136
- +(x::**Bool**, y::**Bool**) at bool.jl:48
- +(x::**Bool**) at bool.jl:45
- +*{T<:AbstractFloat}*(x::**Bool**, y::**T**) at bool.jl:55
- +(x::**Bool**, z::**Complex**) at complex.jl:143
- +(x::**Bool**, A::**AbstractArray{Bool,N<:Any}**) at arraymath.jl:91
- +(x::**Float32**, y::**Float32**) at float.jl:239

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

Now we can define arithmetic:

In [45]:
```
+
```

Out[45]: + (generic function with 164 methods)

In [46]:
```
methods(+)
```

Out[46]: 164 methods for generic function +:

- +(x::**Bool**, z::**Complex{Bool}**) at complex.jl:136
- +(x::**Bool**, y::**Bool**) at bool.jl:48
- +(x::**Bool**) at bool.jl:45
- +*{T<:AbstractFloat}*(x::**Bool**, y::**T**) at bool.jl:55
- +(x::**Bool**, z::**Complex**) at complex.jl:143
- +(x::**Bool**, A::**AbstractArray{Bool,N<:Any}**) at arraymath.jl:91
- +(x::**Float32**, y::**Float32**) at float.jl:239
- +(x::**Float64**, y::**Float64**) at float.jl:240
- +(z::**Complex{Bool}**, x::**Bool**) at complex.jl:137
- +(z::**Complex{Bool}**, x::**Real**) at complex.jl:151
- +(a::**Float16**, b::**Float16**) at float16.jl:136

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

In [45]:
```
+
```

Out[45]:
```
+ (generic function with 164 methods)
```

In [46]:
```
methods(+)
```

Out[46]: 164 methods for generic function +:

- +(x::**Bool**, z::**Complex{Bool}**) at complex.jl:136
- +(x::**Bool**, y::**Bool**) at bool.jl:48
- +(x::**Bool**) at bool.jl:45
- +*{T<:AbstractFloat}*(x::**Bool**, y::**T**) at bool.jl:55
- +(x::**Bool**, z::**Complex**) at complex.jl:143
- +(x::**Bool**, A::**AbstractArray{Bool,N<:Any}**) at arraymath.jl:91
- +(x::**Float32**, y::**Float32**) at float.jl:239
- +(x::**Float64**, y::**Float64**) at float.jl:240
- +(z::**Complex{Bool}**, x::**Bool**) at complex.jl:137
- +(z::**Complex{Bool}**, x::**Real**) at complex.jl:151
- +(a::**Float16**, b::**Float16**) at float16.jl:136
- +(x::**Char**, y::**Integer**) at char.jl:40

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Out[45]: + (generic function with 164 methods)

In [46]: `methods(+)`

Out[46]: 164 methods for generic function **+**:

- +(x::**Bool**, z::**Complex{Bool}**) at complex.jl:136
- +(x::**Bool**, y::**Bool**) at bool.jl:48
- +(x::**Bool**) at bool.jl:45
- +*{T<:AbstractFloat}*(x::**Bool**, y::**T**) at bool.jl:55
- +(x::**Bool**, z::**Complex**) at complex.jl:143
- +(x::**Bool**, A::**AbstractArray{Bool,N<:Any}**) at arraymath.jl:91
- +(x::**Float32**, y::**Float32**) at float.jl:239
- +(x::**Float64**, y::**Float64**) at float.jl:240
- +(z::**Complex{Bool}**, x::**Bool**) at complex.jl:137
- +(z::**Complex{Bool}**, x::**Real**) at complex.jl:151
- +(a::**Float16**, b::**Float16**) at float16.jl:136
- +(x::**Char**, y::**Integer**) at char.jl:40
- +(c::**BigInt**, x::**BigFloat**) at mpfr.jl:240
- +(a::**BigInt**, b::**BigInt**, c::**BigInt**, d::**BigInt**, e::**BigInt**) at gmp.jl:298

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

- +(c::**BigInt**, x::**BigFloat**) at mpfr.jl:240
- +(a::**BigInt**, b::**BigInt**, c::**BigInt**, d::**BigInt**, e::**BigInt**) at gmp.jl:298
- +(a::**BigInt**, b::**BigInt**, c::**BigInt**, d::**BigInt**) at gmp.jl:291

In [44]:
```
import Base: +, *

+(a::Dual, b::Dual) = Dual(a.value + b.value, a.derivative + b.deriva
*(a::Dual, b::Dual) = Dual(a.value*b.value, a.derivative*b.value + b.
*(a::Number, b::Dual) = Dual(a*b.value, a*b.derivative)
```

Out[44]:
```
* (generic function with 151 methods)
```

In [ ]:
```
a = Dual(1, 2)
a + a
```

Julia is written with **generic code** when possible:

In [ ]:
```
M = [1 2; 3 4]
```

Broadcasting using dot (pointwise / elementwise) notation:

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

- +(c::**BigInt**, x::**BigFloat**) at mpfr.jl:240
- +(a::**BigInt**, b::**BigInt**, c::**BigInt**, d::**BigInt**, e::**BigInt**) at gmp.jl:298
- +(a::**BigInt**, b::**BigInt**, c::**BigInt**, d::**BigInt**) at gmp.jl:291

In [47]:

```
import Base: +, *

+(a::Dual, b::Dual) = Dual(a.value + b.value, a.derivative + b.deriva
*(a::Dual, b::Dual) = Dual(a.value*b.value, a.derivative*b.value + b.
*(a::Number, b::Dual) = Dual(a*b.value, a*b.derivative)
```

```
WARNING: Method definition +(Main.Dual, Main.Dual) in module Main at
In[44]:3 overwritten at In[47]:3.
WARNING: Method definition *(Main.Dual, Main.Dual) in module Main at
In[44]:4 overwritten at In[47]:4.
WARNING: Method definition *(Number, Main.Dual) in module Main at In
[44]:5 overwritten at In[47]:5.
```

Out[47]:

```
* (generic function with 151 methods)
```

In [ ]:

```
a = Dual(1, 2)
a + a
```

Julia is written with **generic code** when possible:

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

```
+(a::Dual, b::Dual) = Dual(a.value + b.value, a.derivative + b.deriva
*(a::Dual, b::Dual) = Dual(a.value*b.value, a.derivative*b.value + b.
*(a::Number, b::Dual) = Dual(a*b.value, a*b.derivative)
```

```
WARNING: Method definition +(Main.Dual, Main.Dual) in module Main at
In[44]:3 overwritten at In[47]:3.
WARNING: Method definition *(Main.Dual, Main.Dual) in module Main at
In[44]:4 overwritten at In[47]:4.
WARNING: Method definition *(Number, Main.Dual) in module Main at In
[44]:5 overwritten at In[47]:5.
```

Out[47]: * (generic function with 151 methods)

In [48]:
```
a = Dual(1, 2)
a + a
```

Out[48]: Dual(2.0,4.0)

Julia is written with **generic code** when possible:

In [ ]:
```
M = [1 2; 3 4]
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
WARNING: Method definition *(Main.Dual, Main.Dual) in module Main at In[44]:4 overwritten at In[47]:4.
WARNING: Method definition *(Number, Main.Dual) in module Main at In[44]:5 overwritten at In[47]:5.
```

Out[47]:
```
* (generic function with 151 methods)
```

In [48]:
```
a = Dual(1, 2)
a + a
```

Out[48]:
```
Dual(2.0,4.0)
```

Julia is written with **generic code** when possible:

In [ ]:
```
M = [1 2; 3 4]
```

Broadcasting using dot (pointwise / elementwise) notation:

In [ ]:
```
M2 = Dual.(M, [1])   # in 0.6:  Dual.(M, 1)
```

In [ ]:
```
M2 * M2  # matrix multiply
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
[44]:5 overwritten at In[47]:5.
```

Out[47]:
```
* (generic function with 151 methods)
```

In [48]:
```julia
a = Dual(1, 2)
a + a
```

Out[48]:
```
Dual(2.0,4.0)
```

Julia is written with **generic code** when possible:

In [*]:
```julia
M = [1 2; 3 4]
```

Broadcasting using dot (pointwise / elementwise) notation:

In [ ]:
```julia
M2 = Dual.(M, [1])   # in 0.6:  Dual.(M, 1)
```

In [ ]:
```julia
M2 * M2  # matrix multiply
```

Julia has automatically used the correct + and * methods to do the matrix multiply with

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

Out[48]:
```
Dual(2.0,4.0)
```

Julia is written with **generic code** when possible:

In [49]:
```
M = [1 2; 3 4]
```

Out[49]:
```
2x2 Array{Int64,2}:
 1  2
 3  4
```

Broadcasting using dot (pointwise / elementwise) notation:

In [ ]:
```
M2 = Dual.(M, [1])   # in 0.6:  Dual.(M, 1)
```

In [ ]:
```
M2 * M2  # matrix multiply
```

Julia has automatically used the correct + and * methods to do the matrix multiply with `Dual` objects.

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Out[48]: `Dual(2.0,4.0)`

Julia is written with **generic code** when possible:

In [49]:
```
M = [1 2; 3 4]
```

Out[49]:
```
2x2 Array{Int64,2}:
 1  2
 3  4
```

Broadcasting using dot (pointwise / elementwise) notation:

In [ ]:
```
M2 = Dual.(M, [1])   # in 0.6:  Dual.(M, 1)
```

In [ ]:
```
M2 * M2  # matrix multiply
```

Julia has automatically used the correct + and * methods to do the matrix multiply with `Dual` objects.

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Julia is written with **generic code** when possible:

In [49]:
```
M = [1 2; 3 4]
```

Out[49]:
```
2x2 Array{Int64,2}:
 1  2
 3  4
```

Broadcasting using dot (pointwise / elementwise) notation:

In [*]:
```
M2 = Dual.(M, [1])   # in 0.6:  Dual.(M, 1)
```

In [ ]:
```
M2 * M2  # matrix multiply
```

Julia has automatically used the correct + and * methods to do the matrix multiply with `Dual` objects.

6.1 User-defined types are efficient!

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

Julia is written with **generic code** when possible:

In [49]:
```
M = [1 2; 3 4]
```

Out[49]:
```
2x2 Array{Int64,2}:
 1  2
 3  4
```

Broadcasting using dot (pointwise / elementwise) notation:

In [50]:
```
M2 = Dual.(M, [1])   # in 0.6:  Dual.(M, 1)
```

Out[50]:
```
2x2 Array{Dual,2}:
 Dual(1.0,1.0)  Dual(2.0,1.0)
 Dual(3.0,1.0)  Dual(4.0,1.0)
```

In [ ]:
```
M2 * M2  # matrix multiply
```

Julia has automatically used the correct + and * methods to do the matrix multiply with `Dual` objects.

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

In [49]:
```
M = [1 2; 3 4]
```

Out[49]:
```
2x2 Array{Int64,2}:
 1  2
 3  4
```

Broadcasting using dot (pointwise / elementwise) notation:

In [50]:
```
M2 = Dual.(M, [1])   # in 0.6:  Dual.(M, 1)
```

Out[50]:
```
2x2 Array{Dual,2}:
 Dual(1.0,1.0)  Dual(2.0,1.0)
 Dual(3.0,1.0)  Dual(4.0,1.0)
```

In [ ]:
```
M2 * M2  # matrix multiply
```

Julia has automatically used the correct + and * methods to do the matrix multiply with `Dual` objects.

6.1 User-defined types are efficient!

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

In [49]:
```
M = [1 2; 3 4]
```

Out[49]:
```
2x2 Array{Int64,2}:
 1  2
 3  4
```

Broadcasting using dot (pointwise / elementwise) notation:

In [50]:
```
M2 = Dual.(M, [1])   # in 0.6:  Dual.(M, 1)
```

Out[50]:
```
2x2 Array{Dual,2}:
 Dual(1.0,1.0)  Dual(2.0,1.0)
 Dual(3.0,1.0)  Dual(4.0,1.0)
```

In [51]:
```
M2 * M2  # matrix multiply
```

Out[51]:
```
2x2 Array{Dual,2}:
 Dual(7.0,7.0)    Dual(10.0,9.0)
 Dual(15.0,11.0)  Dual(22.0,13.0)
```

Julia has automatically used the correct + and * methods to do the matrix multiply with `Dual` objects.

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Broadcasting using dot (pointwise / elementwise) notation:

In [50]:
```
M2 = Dual.(M, [1])   # in 0.6:  Dual.(M, 1)
```

Out[50]:
```
2x2 Array{Dual,2}:
 Dual(1.0,1.0)  Dual(2.0,1.0)
 Dual(3.0,1.0)  Dual(4.0,1.0)
```

In [51]:
```
M2 * M2  # matrix multiply
```

Out[51]:
```
2x2 Array{Dual,2}:
 Dual(7.0,7.0)    Dual(10.0,9.0)
 Dual(15.0,11.0)  Dual(22.0,13.0)
```

click to scroll output; double click to hide

Julia has automatically used the correct + and * methods to do the matrix multiply with `Dual` objects.

6.1 User-defined types are efficient!

Broadcasting using dot (pointwise / elementwise) notation:

In [50]:
```
M2 = Dual.(M, [1])   # in 0.6:  Dual.(M, 1)
```

Out[50]:
```
2x2 Array{Dual,2}:
 Dual(1.0,1.0)  Dual(2.0,1.0)
 Dual(3.0,1.0)  Dual(4.0,1.0)
```

In [51]:
```
M2 * M2  # matrix multiply
```

Out[51]:
```
2x2 Array{Dual,2}:
 Dual(7.0,7.0)    Dual(10.0,9.0)
 Dual(15.0,11.0)  Dual(22.0,13.0)
```

Julia has automatically used the correct + and * methods to do the matrix multiply with `Dual` objects.

6.1 User-defined types are efficient!

In [ ]:
```
h(a, b) = (a[1]*b[1], a[1]*b[2] + b[1]*a[2])
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

Broadcasting using dot (pointwise / elementwise) notation:

In [50]:
```
M2 = Dual.(M, [1])   # in 0.6:  Dual.(M, 1)
```

Out[50]:
```
2x2 Array{Dual,2}:
 Dual(1.0,1.0)  Dual(2.0,1.0)
 Dual(3.0,1.0)  Dual(4.0,1.0)
```

In [51]:
```
M2 * M2  # matrix multiply
```

Out[51]:
```
2x2 Array{Dual,2}:
 Dual(7.0,7.0)    Dual(10.0,9.0)
 Dual(15.0,11.0)  Dual(22.0,13.0)
```

Julia has automatically used the correct + and * methods to do the matrix multiply with `Dual` objects.

6.1 User-defined types are efficient!

In [ ]:
```
h(a, b) = (a[1]*b[1], a[1]*b[2] + b[1]*a[2])
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

In [51]:
```
M2 * M2  # matrix multiply
```

Out[51]:
```
2x2 Array{Dual,2}:
 Dual(7.0,7.0)    Dual(10.0,9.0)
 Dual(15.0,11.0)  Dual(22.0,13.0)
```

Julia has automatically used the correct + and * methods to do the matrix multiply with `Dual` objects.

6.1 User-defined types are efficient!

In [ ]:
```
h(a, b) = (a[1]*b[1], a[1]*b[2] + b[1]*a[2])

@code_native h((1, 1), (1, 1))
```

In [ ]:
```
a = Dual(1, 1)
@code_native a + a
```

- The code is almost identical to the hand-written version

```
Dual(7.0,7.0)    Dual(10.0,9.0)
Dual(15.0,11.0)  Dual(22.0,13.0)
```

Julia has automatically used the correct + and * methods to do the matrix multiply with `Dual` objects.

6.1 User-defined types are efficient!

In [ ]:
```
h(a, b) = (a[1]*b[1], a[1]*b[2] + b[1]*a[2])

@code_native h((1, 1), (1, 1))
```

In [ ]:
```
a = Dual(1, 1)
@code_native a + a
```

- The code is almost identical to the hand-written version
- There is **no overhead** from using types.
- The design of Julia means that user-defined types have the **same status** as

Julia has automatically used the correct + and * methods to do the matrix multiply with `Dual` objects.

6.1 User-defined types are efficient!

In [ ]:
```
h(a, b) = (a[1]*b[1], a[1]*b[2] + b[1]*a[2])

@code_native h((1, 1), (1, 1))
```

In [ ]:
```
a = Dual(1, 1)
@code_native a + a
```

- The code is almost identical to the hand-written version
- There is **no overhead** from using types.
- The design of Julia means that user-defined types have the **same status** as standard "built-in" Julia types

Notebook saved | Julia 0.5.0

File Edit View Insert Cell Kernel Navigate Widgets Help

Julia has automatically used the correct + and * methods to do the matrix multiply with Dual objects.

6.1 User-defined types are efficient!

In [ ]:
```
h(a, b) = (a[1]*b[1], a[1]*b[2] + b[1]*a[2])

@code_native h((1, 1), (1, 1))
```

In [ ]:
```
a = Dual(1, 1)
@code_native a + a
```

- The code is almost identical to the hand-written version
- There is **no overhead** from using types.
- The design of Julia means that user-defined types have the **same status** as standard "built-in" Julia types

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Julia has automatically used the correct + and * methods to do the matrix multiply with Dual objects.

6.1 User-defined types are efficient!

In [52]:
```
h(a, b) = (a[1]*b[1], a[1]*b[2] + b[1]*a[2])

@code_native h((1, 1), (1, 1))
```

```
	.section	__TEXT,__text,regular,pure_instructions
Filename: In[52]
	pushq	%rbp
	movq	%rsp, %rbp
Source line: 1
	movq	(%rsi), %rax
	movq	(%rdx), %rcx
	movq	%rcx, %r8
	imulq	%rax, %r8
	imulq	8(%rdx), %rax
	imulq	8(%rsi), %rcx
	addq	%rax, %rcx
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
In [52]: h(a, b) = (a[1]*b[1], a[1]*b[2] + b[1]*a[2])

         @code_native h((1, 1), (1, 1))
```

```
        .section        __TEXT,__text,regular,pure_instructions
Filename: In[52]
        pushq   %rbp
        movq    %rsp, %rbp
Source line: 1
        movq    (%rsi), %rax
        movq    (%rdx), %rcx
        movq    %rcx, %r8
        imulq   %rax, %r8
        imulq   8(%rdx), %rax
        imulq   8(%rsi), %rcx
        addq    %rax, %rcx
        movq    %r8, (%rdi)
        movq    %rcx, 8(%rdi)
        movq    %rdi, %rax
        popq    %rbp
        retq
        nopw    (%rax,%rax)
```

```
In [ ]: a = Dual(1, 1)
```

```
	movq	(%rsi), %rax
	movq	(%rdx), %rcx
	movq	%rcx, %r8
	imulq	%rax, %r8
	imulq	8(%rdx), %rax
	imulq	8(%rsi), %rcx
	addq	%rax, %rcx
	movq	%r8, (%rdi)
	movq	%rcx, 8(%rdi)
	movq	%rdi, %rax
	popq	%rbp
	retq
	nopw	(%rax,%rax)
```

In [ ]:

```julia
a = Dual(1, 1)
@code_native a + a
```

- The code is almost identical to the hand-written version
- There is **no overhead** from using types.
- The design of Julia means that user-defined types have the **same status** as standard "built-in" Julia types

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        movq    (%rdx), %rcx
        movq    %rcx, %r8
        imulq   %rax, %r8
        imulq   8(%rdx), %rax
        imulq   8(%rsi), %rcx
        addq    %rax, %rcx
        movq    %r8, (%rdi)
        movq    %rcx, 8(%rdi)
        movq    %rdi, %rax
        popq    %rbp
        retq
        nopw    (%rax,%rax)
```

In [53]:

```
a = Dual(1, 1)
@code_native a + a
```

```
        .section        __TEXT,__text,regular,pure_instructions
Filename: In[47]
        pushq   %rbp
        movq    %rsp, %rbp
Source line: 3
        movsd   (%rsi), %xmm0           ## xmm0 = mem[0],zero
        movsd   8(%rsi), %xmm1          ## xmm1 = mem[0],zero
        addsd   (%rdx), %xmm0
```

```
        imulq   8(%rdx), %rax
        imulq   8(%rsi), %rcx
        addq    %rax, %rcx
        movq    %r8, (%rdi)
        movq    %rcx, 8(%rdi)
        movq    %rdi, %rax
        popq    %rbp
        retq
        nopw    (%rax,%rax)
```

In [53]:

```
a = Dual(1, 1)
@code_native a + a
```

```
        .section        __TEXT,__text,regular,pure_instructions
Filename: In[47]
        pushq   %rbp
        movq    %rsp, %rbp
Source line: 3
        movsd   (%rsi), %xmm0           ## xmm0 = mem[0],zero
        movsd   8(%rsi), %xmm1          ## xmm1 = mem[0],zero
        addsd   (%rdx), %xmm0
        addsd   8(%rdx), %xmm1
        movsd   %xmm0, (%rdi)
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

```
        retq
        nopw    (%rax,%rax)
```

In [53]:

```julia
a = Dual(1, 1)
@code_native a + a
```

```
        .section        __TEXT,__text,regular,pure_instructions
Filename: In[47]
        pushq   %rbp
        movq    %rsp, %rbp
Source line: 3
        movsd   (%rsi), %xmm0           ## xmm0 = mem[0],zero
        movsd   8(%rsi), %xmm1          ## xmm1 = mem[0],zero
        addsd   (%rdx), %xmm0
        addsd   8(%rdx), %xmm1
        movsd   %xmm0, (%rdi)
        movsd   %xmm1, 8(%rdi)
        movq    %rdi, %rax
        popq    %rbp
        retq
        nopw    %cs:(%rax,%rax)
```

- The code is almost identical to the hand-written version

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

```
        nopw    (%rax,%rax)
```

In [53]:

```julia
a = Dual(1, 1)
@code_native a + a
```

```
        .section        __TEXT,__text,regular,pure_instructions
Filename: In[47]
        pushq   %rbp
        movq    %rsp, %rbp
Source line: 3
        movsd   (%rsi), %xmm0           ## xmm0 = mem[0],zero
        movsd   8(%rsi), %xmm1          ## xmm1 = mem[0],zero
        addsd   (%rdx), %xmm0
        addsd   8(%rdx), %xmm1
        movsd   %xmm0, (%rdi)
        movsd   %xmm1, 8(%rdi)
        movq    %rdi, %rax
        popq    %rbp
        retq
        nopw    %cs:(%rax,%rax)
```

- The code is almost identical to the hand-written version

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
        retq
        nopw    (%rax,%rax)
```

In [54]:
```julia
a = Dual(1, 1)
@code_native a * a
```

```
        .section        __TEXT,__text,regular,pure_instructions
Filename: In[47]
        pushq   %rbp
        movq    %rsp, %rbp
Source line: 4
        movsd   (%rsi), %xmm0           ## xmm0 = mem[0],zero
        movsd   (%rdx), %xmm1           ## xmm1 = mem[0],zero
        movapd  %xmm0, %xmm2
        mulsd   %xmm1, %xmm2
        mulsd   8(%rsi), %xmm1
        mulsd   8(%rdx), %xmm0
        addsd   %xmm1, %xmm0
        movsd   %xmm2, (%rdi)
        movsd   %xmm0, 8(%rdi)
        movq    %rdi, %rax
        popq    %rbp
        retq
```

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

In [54]:
```julia
a = Dual(1, 1)
@code_native a * a
```

```
	.section	__TEXT,__text,regular,pure_instructions
Filename: In[47]
	pushq	%rbp
	movq	%rsp, %rbp
Source line: 4
	movsd	(%rsi), %xmm0           ## xmm0 = mem[0],zero
	movsd	(%rdx), %xmm1           ## xmm1 = mem[0],zero
	movapd	%xmm0, %xmm2
	mulsd	%xmm1, %xmm2
	mulsd	8(%rsi), %xmm1
	mulsd	8(%rdx), %xmm0
	addsd	%xmm1, %xmm0
	movsd	%xmm2, (%rdi)
	movsd	%xmm0, 8(%rdi)
	movq	%rdi, %rax
	popq	%rbp
	retq
```

- The code is almost identical to the hand-written version
- There is **no overhead** from using types

```
Source line: 4
        movsd   (%rsi), %xmm0              ## xmm0 = mem[0],zero
        movsd   (%rdx), %xmm1              ## xmm1 = mem[0],zero
        movapd  %xmm0, %xmm2
        mulsd   %xmm1, %xmm2
        mulsd   8(%rsi), %xmm1
        mulsd   8(%rdx), %xmm0
        addsd   %xmm1, %xmm0
        movsd   %xmm2, (%rdi)
        movsd   %xmm0, 8(%rdi)
        movq    %rdi, %rax
        popq    %rbp
        retq
```

- The code is almost identical to the hand-written version
- There is **no overhead** from using types.
- The design of Julia means that user-defined types have the **same status** as standard "built-in" Julia types

```
movsd   (%rax), %xmm1             ## xmm1 = mem[0],zero
movapd  %xmm0, %xmm2
mulsd   %xmm1, %xmm2
mulsd   8(%rsi), %xmm1
mulsd   8(%rdx), %xmm0
addsd   %xmm1, %xmm0
movsd   %xmm2, (%rdi)
movsd   %xmm0, 8(%rdi)
movq    %rdi, %rax
popq    %rbp
retq
```

- The code is almost identical to the hand-written version
- There is **no overhead** from using types.
- The design of Julia means that user-defined types have the **same status** as standard "built-in" Julia types

6.2 Parametric types

Types may be made more flexible using **parameters**:

```
        popq    %rbp
        retq
```

- The code is almost identical to the hand-written version
- There is **no overhead** from using types.
- The design of Julia means that user-defined types have the **same status** as standard "built-in" Julia types

6.2 Parametric types

Types may be made more flexible using **parameters**:

In [ ]:
```
immutable Dual2{T}
    value::T
    derivative::T
end
```

In [ ]:
```
Dual2(3, 4)
```

6.2 Parametric types

Types may be made more flexible using **parameters**:

In [ ]:
```
immutable Dual2{T}
    value::T
    derivative::T
end
```

In [ ]:
```
Dual2(3, 4)
```

In [ ]:
```
Dual2(3.5, 4.5)
```

In [ ]:
```
Dual2(3.5, 4)
```

It is natural to want to be able to do this, so just **add a method**:

In [ ]:
```
Dual2{S,T}(a::S, b::T) = ( (a2, b2) = promote(a, b); Dual2(a2, b2) )
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

6.2 Parametric types

Types may be made more flexible using **parameters**:

In [55]:
```
immutable Dual2{T}
    value::T
    derivative::T
end
```

In [ ]:
```
Dual2(3, 4)
```

In [ ]:
```
Dual2(3.5, 4.5)
```

In [ ]:
```
Dual2(3.5, 4)
```

It is natural to want to be able to do this, so just **add a method**:

In [ ]:
```
Dual2{S,T}(a::S, b::T) = ( (a2, b2) = promote(a, b); Dual2(a2, b2) )
```

In [ ]:
```
Dual2(3.4, 5)
```

Types may be made more flexible using **parameters**.

In [55]:
```
immutable Dual2{T}
    value::T
    derivative::T
end
```

In [56]:
```
Dual2(3, 4)
```

Out[56]:
```
Dual2{Int64}(3,4)
```

In [57]:
```
Dual2(3.5, 4.5)
```

Out[57]:
```
Dual2{Float64}(3.5,4.5)
```

In [ ]:
```
Dual2(3.5, 4)
```

It is natural to want to be able to do this, so just **add a method**:

In [ ]:
```
Dual2{S,T}(a::S, b::T) = ( (a2, b2) = promote(a, b); Dual2(a2, b2) )
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
    value::T
    derivative::T
end
```

In [56]:
```
Dual2(3, 4)
```

Out[56]:
```
Dual2{Int64}(3,4)
```

In [57]:
```
Dual2(3.5, 4.5)
```

Out[57]:
```
Dual2{Float64}(3.5,4.5)
```

In [ ]:
```
Dual2(3.5, 4)
```

It is natural to want to be able to do this, so just **add a method**:

In [ ]:
```
Dual2{S,T}(a::S, b::T) = ( (a2, b2) = promote(a, b); Dual2(a2, b2) )
```

In [ ]:
```
Dual2(3.4, 5)
```

There is pattern matching of type parameters.

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

```
    derivative::T
end
```

In [56]:
```
Dual2(3, 4)
```

Out[56]:
```
Dual2{Int64}(3,4)
```

In [57]:
```
Dual2(3.5, 4.5)
```

Out[57]:
```
Dual2{Float64}(3.5,4.5)
```

In [58]:
```
Dual2(3.5, 4)
```

```
MethodError: no method matching Dual2{T}(::Float64, ::Int64)
Closest candidates are:
  Dual2{T}{T}(::T, ::T) at In[55]:2
  Dual2{T}{T}(::Any) at sysimg.jl:53
```

It is natural to want to be able to do this, so just **add a method**:

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

6.2 Parametric types

Types may be made more flexible using **parameters**:

In [55]:
```
immutable Dual2{T}
    value::T
    derivative::T
end
```

In [56]:
```
Dual2(3, 4)
```

Out[56]: Dual2{Int64}(3,4)

In [57]:
```
Dual2(3.5, 4.5)
```

Out[57]: Dual2{Float64}(3.5,4.5)

In [58]:
```
Dual2(3.5, 4)
```

```
MethodError: no method matching Dual2{T}(::Float64, ::Int64)
Closest candidates are:
  Dual2{T}{T}(::T, ::T) at In[55]:2
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

In [55]:
```
immutable Dual2{T}
    value::T
    derivative::T
end
```

In [56]:
```
Dual2(3, 4)
```

Out[56]:
```
Dual2{Int64}(3,4)
```

In [57]:
```
Dual2(3.5, 4.5)
```

Out[57]:
```
Dual2{Float64}(3.5,4.5)
```

In [58]:
```
Dual2(3.5, 4)
```

```
MethodError: no method matching Dual2{T}(::Float64, ::Int64)
Closest candidates are:
  Dual2{T}{T}(::T, ::T) at In[55]:2
  Dual2{T}{T}(::Any) at sysimg.jl:53
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
    derivative::T
end
```

In [56]:
```
Dual2(3, 4)
```

Out[56]:
```
Dual2{Int64}(3,4)
```

In [57]:
```
Dual2(3.5, 4.5)
```

Out[57]:
```
Dual2{Float64}(3.5,4.5)
```

In [58]:
```
Dual2(3.5, 4)
```

```
MethodError: no method matching Dual2{T}(::Float64, ::Int64)
Closest candidates are:
  Dual2{T}{T}(::T, ::T) at In[55]:2
  Dual2{T}{T}(::Any) at sysimg.jl:53
```

It is natural to want to be able to do this, so just **add a method**:

In [ ]:
```
Dual2{S,T}(a::S, b::T) = ( (a2, b2) = promote(a, b); Dual2(a2, b2) )
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

In [56]:
```julia
Dual2(3, 4)
```

Out[56]:
```
Dual2{Int64}(3,4)
```

In [57]:
```julia
Dual2(3.5, 4.5)
```

Out[57]:
```
Dual2{Float64}(3.5,4.5)
```

In [58]:
```julia
Dual2(3.5, 4)
```

```
MethodError: no method matching Dual2{T}(::Float64, ::Int64)
Closest candidates are:
  Dual2{T}{T}(::T, ::T) at In[55]:2
  Dual2{T}{T}(::Any) at sysimg.jl:53
```

It is natural to want to be able to do this, so just **add a method**:

In [ ]:
```julia
Dual2{S,T}(a::S, b::T) = ( (a2, b2) = promote(a, b); Dual2(a2, b2) )
```

In [ ]:
```julia
Dual2(3.4, 5)
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Out[56]:
```
Dual2{Int64}(3,4)
```

In [57]:
```julia
Dual2(3.5, 4.5)
```

Out[57]:
```
Dual2{Float64}(3.5,4.5)
```

In [58]:
```julia
Dual2(3.5, 4)
```

```
MethodError: no method matching Dual2{T}(::Float64, ::Int64)
Closest candidates are:
  Dual2{T}{T}(::T, ::T) at In[55]:2
  Dual2{T}{T}(::Any) at sysimg.jl:53
```

It is natural to want to be able to do this, so just **add a method**:

In [ ]:
```julia
Dual2{S,T}(a::S, b::T) = ( (a2, b2) = promote(a, b); Dual2(a2, b2) )
```

In [ ]:
```julia
Dual2(3.4, 5)
```

File Edit View Insert Cell Kernel Navigate Widgets Help

Out[57]: Dual2{Float64}(3.5,4.5)

In [58]:
```
Dual2(3.5, 4)
```

```
MethodError: no method matching Dual2{T}(::Float64, ::Int64)
Closest candidates are:
  Dual2{T}{T}(::T, ::T) at In[55]:2
  Dual2{T}{T}(::Any) at sysimg.jl:53
```

It is natural to want to be able to do this, so just **add a method**:

In [59]:
```
Dual2{S,T}(a::S, b::T) = ( (a2, b2) = promote(a, b); Dual2(a2, b2) )
```

Out[59]: Dual2{T}

In [60]:
```
Dual2(3.4, 5)
```

Out[60]: Dual2{Float64}(3.4,5.0)

File Edit View Insert Cell Kernel Navigate Widgets Help | Julia 0.5.0

In [58]:
```
Dual2(3.5, 4)
```

```
MethodError: no method matching Dual2{T}(::Float64, ::Int64)
Closest candidates are:
  Dual2{T}{T}(::T, ::T) at In[55]:2
  Dual2{T}{T}(::Any) at sysimg.jl:53
```

It is natural to want to be able to do this, so just **add a method**:

In [59]:
```
Dual2{S,T}(a::S, b::T) = ( (a2, b2) = promote(a, b); Dual2(a2, b2) )
```

Out[59]:
```
Dual2{T}
```

In [60]:
```
Dual2(3.4, 5)
```

Out[60]:
```
Dual2{Float64}(3.4,5.0)
```

There is pattern matching of type parameters.

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
MethodError: no method matching Dual2{T}(::Float64, ::Int64)
Closest candidates are:
  Dual2{T}{T}(::T, ::T) at In[55]:2
  Dual2{T}{T}(::Any) at sysimg.jl:53
```

It is natural to want to be able to do this, so just **add a method**:

In [59]:
```
Dual2{S,T}(a::S, b::T) = ( (a2, b2) = promote(a, b); Dual2(a2, b2) )
```

Out[59]:
```
Dual2{T}
```

In [60]:
```
Dual2(3.4, 5)
```

Out[60]:
```
Dual2{Float64}(3.4,5.0)
```

There is pattern matching of type parameters.

In [ ]:
```
methods(Dual2)
```

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

```
Closest candidates are:
  Dual2{T}{T}(::T, ::T) at In[55]:2
  Dual2{T}{T}(::Any) at sysimg.jl:53
```

It is natural to want to be able to do this, so just **add a method**:

```
In [59]: Dual2{S,T}(a::S, b::T) = ( (a2, b2) = promote(a, b); Dual2(a2, b2) )

Out[59]: Dual2{T}
```

```
In [60]: Dual2(3.4, 5)

Out[60]: Dual2{Float64}(3.4,5.0)
```

There is pattern matching of type parameters.

```
In [ ]: methods(Dual2)
```

- Key foundation of Julia: **multiple dispatch**

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

It is natural to want to be able to do this, so just **add a method**:

In [59]:
```
Dual2{S,T}(a::S, b::T) = ( (a2, b2) = promote(a, b); Dual2(a2, b2) )
```

Out[59]: Dual2{T}

In [60]:
```
Dual2(3.4, 5)
```

Out[60]: Dual2{Float64}(3.4,5.0)

There is pattern matching of type parameters.

In [ ]:
```
methods(Dual2)
```

- Key foundation of Julia: **multiple dispatch**
- The "version" of a function to use is chosen from a patchwork of related functions according to the types of its arguments

File Edit View Insert Cell Kernel Navigate Widgets Help Julia 0.5.0

Out[59]:
```
Dual2{T}
```

In [60]:
```
Dual2(3.4, 5)
```

Out[60]:
```
Dual2{Float64}(3.4,5.0)
```

There is pattern matching of type parameters.

In [ ]:
```
methods(Dual2)
```

- Key foundation of Julia: **multiple dispatch**
- The "version" of a function to use is chosen from a patchwork of related functions according to the types of its arguments

7 Performance tips

- Don't use global variables: they can't be inferred

- Key foundation of Julia: **multiple dispatch**
- The "version" of a function to use is chosen from a patchwork of related functions according to the types of its arguments

7 Performance tips

- Don't use global variables: they can't be inferred
- Put everything in a function
- Use `@code_warntype` to detect type instability
- Profile to detect performance problems