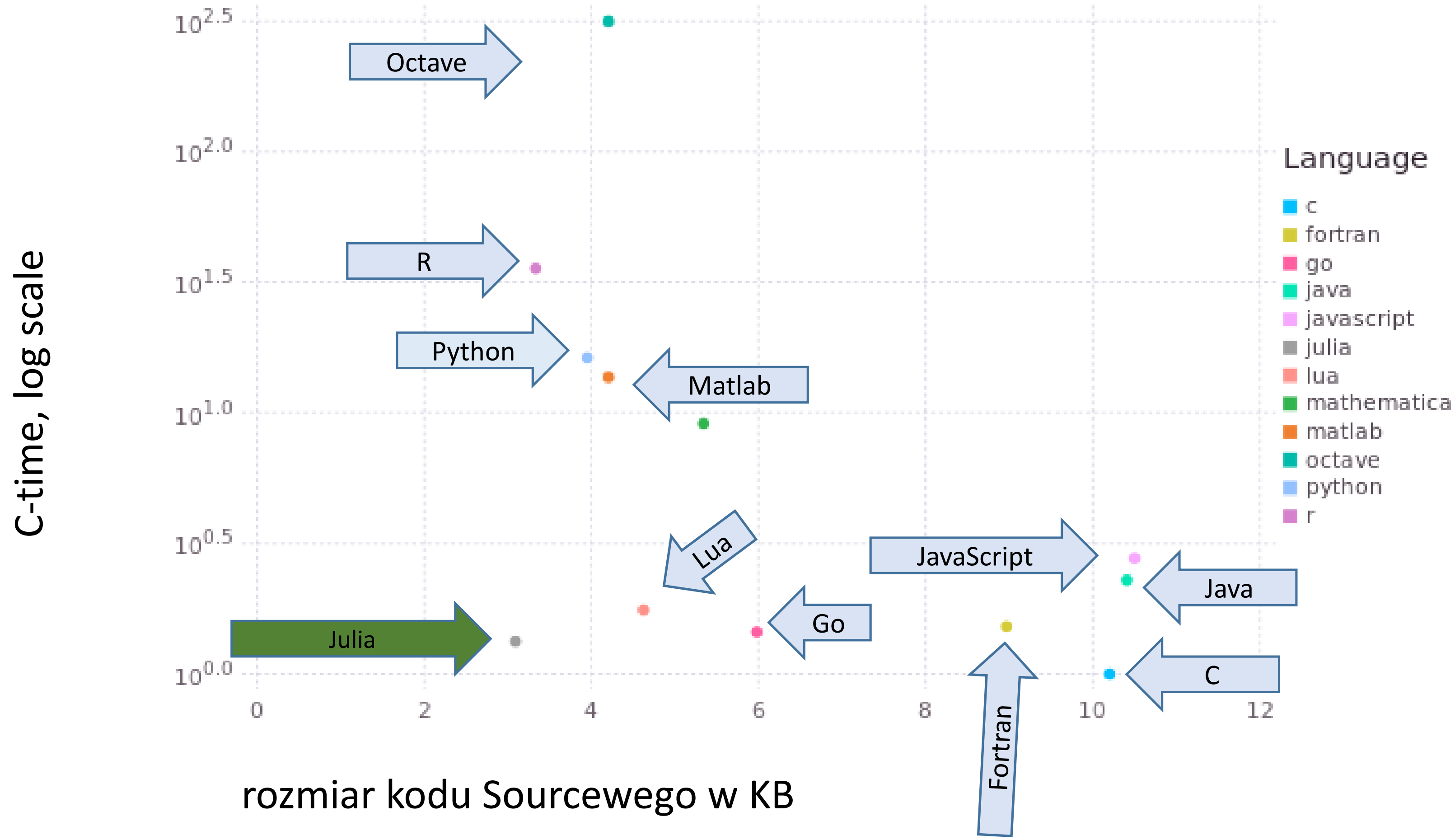
Metaprogramming with Julia

https://szufel.pl

Programmers effort vs execution speed



Source: http://www.oceanographerschoice.com/2016/03/the-julia-language-is-the-way-of-the-future/

Metaprogramming

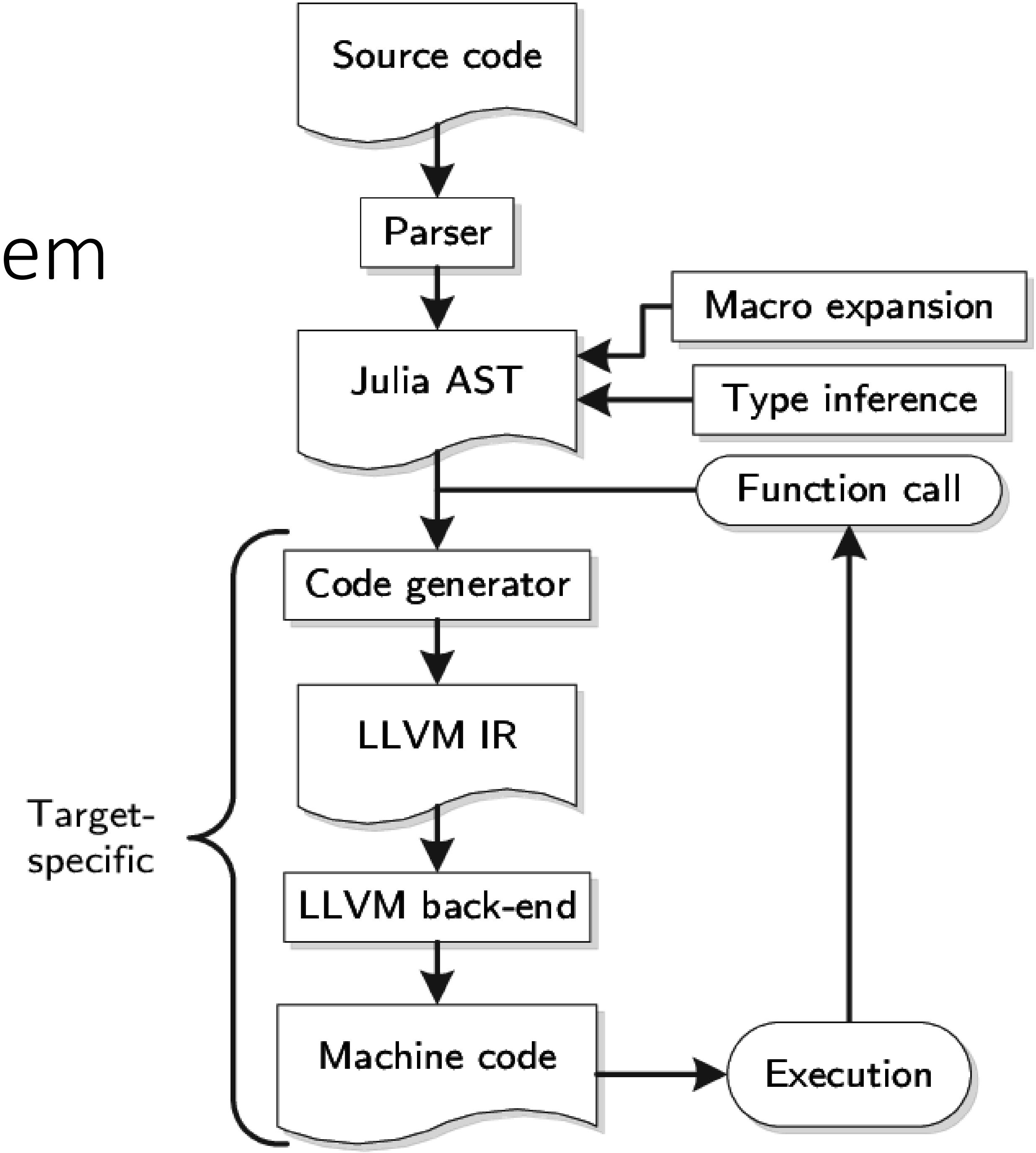
„Metaprogramming is a programming technique in which computer programs have the ability to treat other programs as their data. It means that a program can be designed to read, generate, analyze or transform other programs, and even modify itself while running.” (source: Wikipedia)

```
julia> code = Meta.parse("x=5")
:(x = 5)

julia> dump(code)
Expr
  head: Symbol =
  args: Array{Any}((2,))
    1: Symbol x
    2: Int64 5
```

Metaprogramming (cont.)

```
julia> code = Meta.parse("x=5")
:(x = 5)

julia> dump(code)
Expr
  head: Symbol =
  args: Array{Any}((2,))
    1: Symbol x
    2: Int64 5

julia> eval(code)
5

julia> x
5
```

Julia Compiler system not quite accurate picture...



Source: https://www.researchgate.net/publication/301876510_High-level_GPU_programming_in_Julia

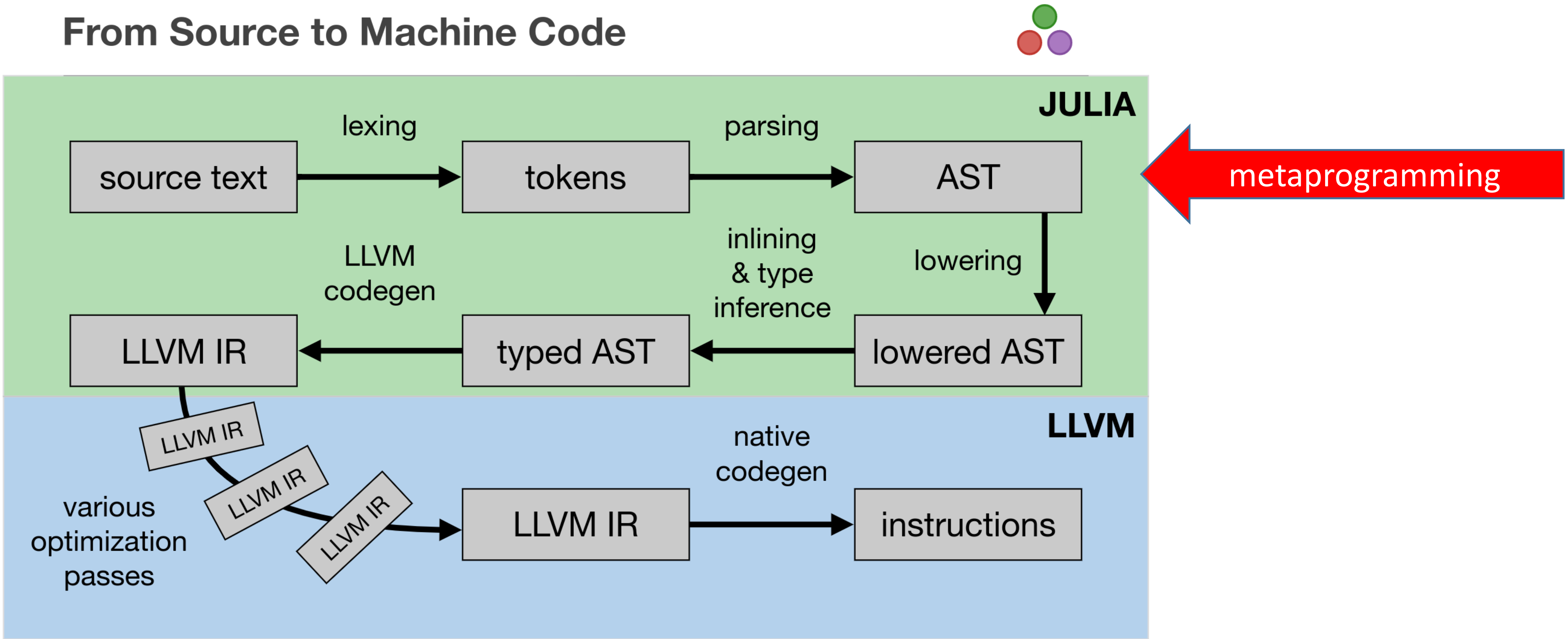
Example 1. Select a field from an object

```
function getValueOfA(x)
  return x.a
end

function getValueOf(x, name::String)
    return getproperty(x, Symbol(name))
end

function getValueOf2(name::String)
     field = Symbol(name)
     code = quote
         (obj) -> obj.$field
     end
     return eval(code)
end

function getValueOf3(name::String)
   return eval(Meta.parse("obj -> obj.$name"))
end
```

Let’s test

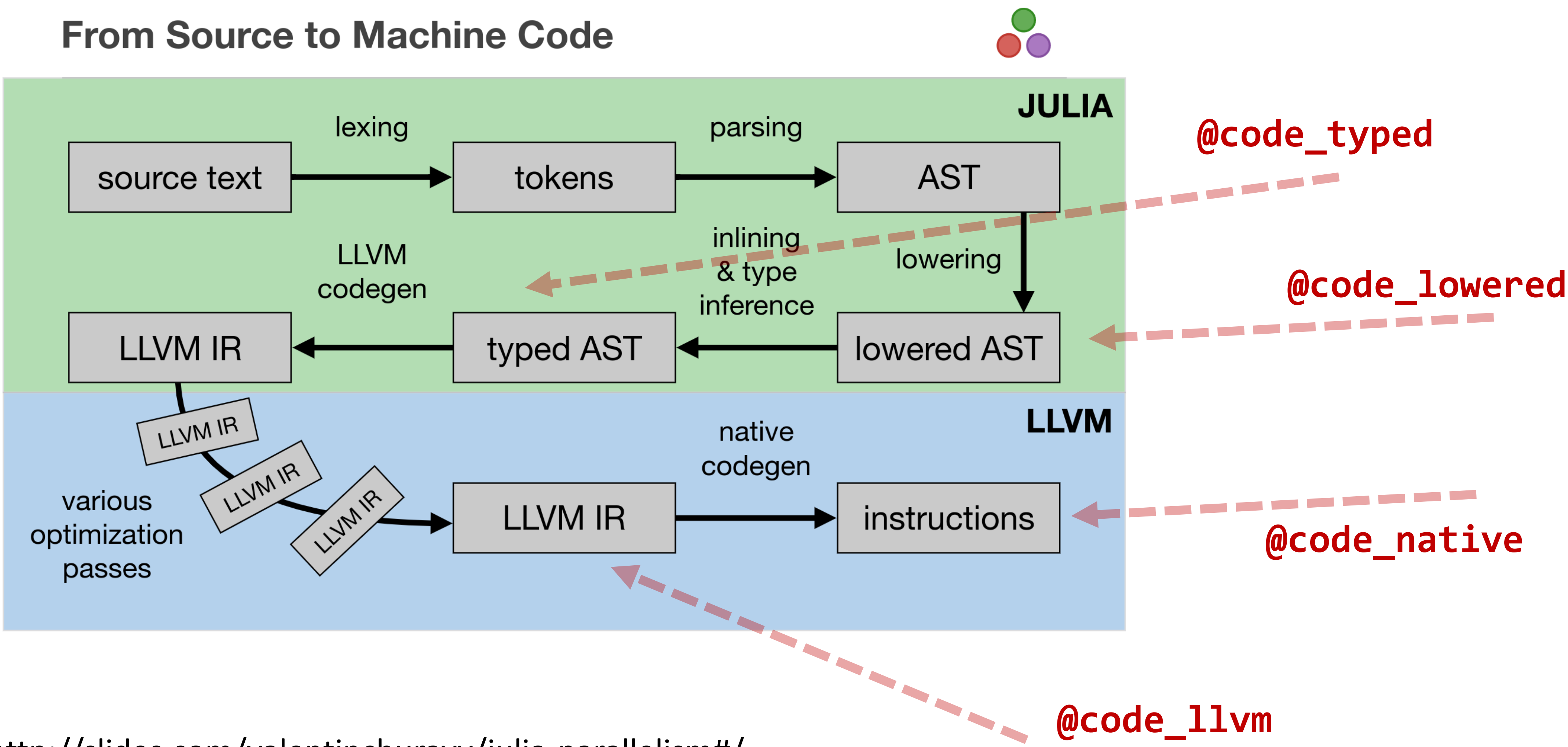
```
using BenchmarkTools
struct MyStruct
          a
          b
end

x = MyStruct(5,6)

@btime getValueOfA($x)
@btime getValueOf($x,"a")
const getVal2 = getValueOf2("a")
@btime getVal2($x)
const getVal3 = getValueOf3("a")
@btime getVal3_($x)
getValueOf3("a+1")(x)
```

Accurate picture



http://slides.com/valentinchuravy/julia-parallelism#/

Bardziej szczegółowy proces



http://slides.com/valentinchuravy/julia-parallelism#/

Example 2: loop unrolling

```
function avg2(vals::Vector{T})
    sum = vals[1]
    for i in 2:length(vals)
        sum += vals[i]
    end
    sum/length(vals)
end
```

Example 2: loop unrolling (Cont.)

```
struct Vector2{N,T}
    vals::Vector{T}
end

@generated function avgg(els::Vector2{N,T}) where {N,T}
    code = :(els.vals[1])
    for i=2:N
        code = :($code + els.vals[$i])
    end
    :(($code)/$N)
end
```

```
using BenchmarkTools
s = Vector2{4,Int64}([1,2,3,4])
@btime avgg($s)
@btime avg2($s.vals)
```

Macros

„Macros provide a method to include generated code in the final body of a program. A macro maps a tuple of arguments to a returned expression, and the resulting expression is compiled directly rather than requiring a runtime eval call. Macro arguments may include expressions, literal values, and symbols.”

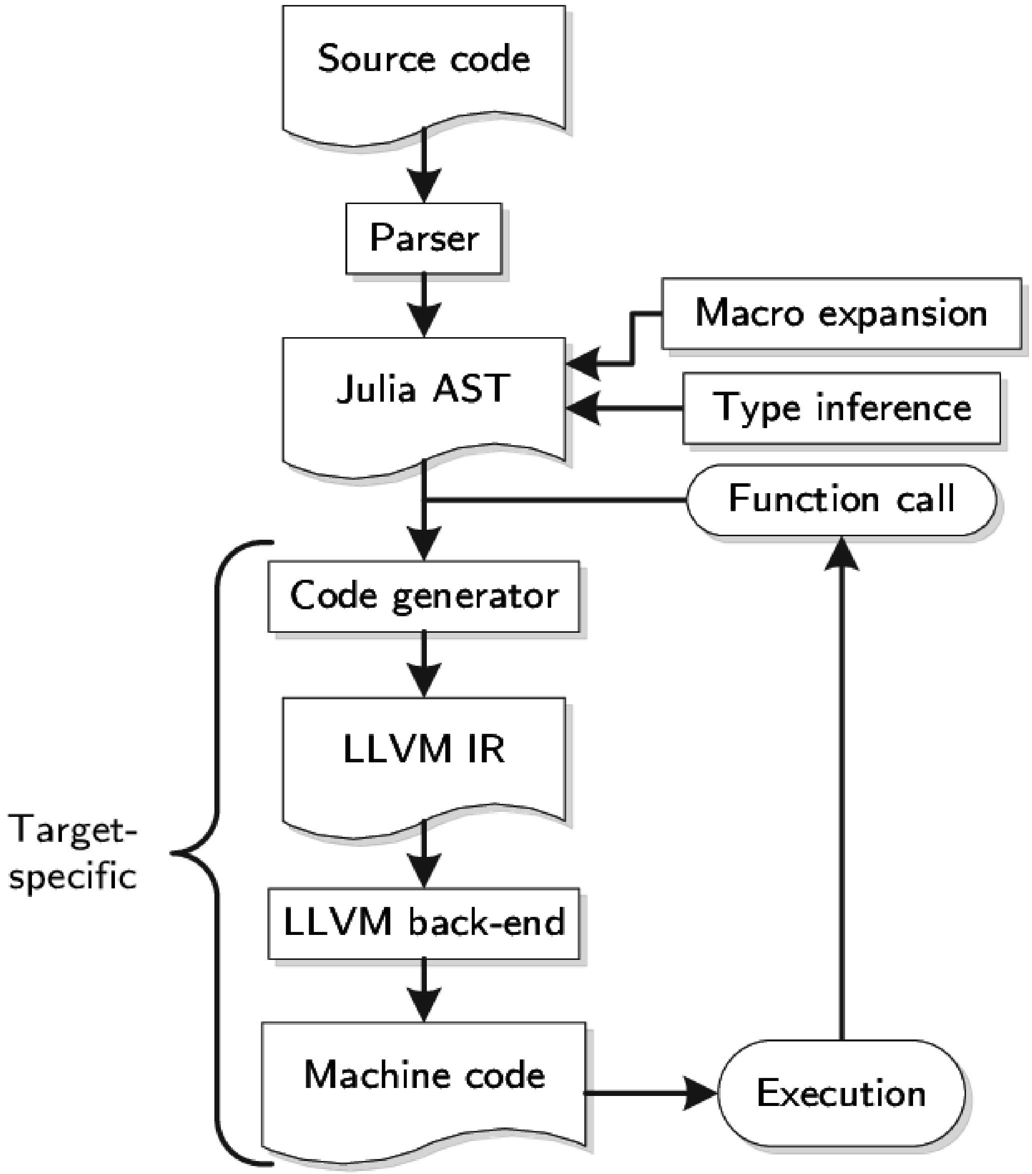
```
macro sayhello(name)
    return :( println("Hello, ", $name) )
end
```

Macro – hello world...

```
macro sayhello(name)
    return :( println("Hello, ", $name) )
end

julia> macroexpand(Main,:(@sayhello("aa")))
:((Main.println)("Hello, ", "aa"))

julia> @sayhello "world!"
Hello, world!
```

When the macro is compiled (inaccurate picture)

```
macro sayhello2(name)
    println("Macro started!")
    return quote
        println("Hello, ", $name)
    end
end

@sayhello2 "World„

for i in 1:3
    @sayhello2 "FROM THE LOOP"
end
```



Example 3 memoization.

```
function fib(n)
    n <= 2 ? 1 : fib(n-1)+fib(n-2)
end

julia> fib(4)
3

julia> @time fib(40)
  0.498755 seconds (5 allocations: 176 bytes)
102334145
```

Example 4. memoization - function

```
function memoit(f::Function,p)
    if !isdefined(Main,:my_memoit_cache)
        global my_memoit_cache =
                Dict{Function,Dict{Any,Any}}()
    end
    cache = haskey(my_memoit_cache,f) ?
              my_memoit_cache[f]
            : my_memoit_cache[f]=Dict()
    haskey(cache,p) ? cache[p] : cache[p] = f(p)
end
```

Example 4. memoization - packing a function to a macro

```
macro memo(e)
   (!(typeof(e) <: Expr) || !(e.head == :call)) &&
      error("Wrong @memo params - required a function call")
   return quote
      memoit($(e.args[1]),$(esc(e.args[2])))
   end
end
```

Example 4. memoization

```
function fib2(n)
    n <= 2 ? 1 : memoit(fib2,n-1)+memoit(fib2,n-2)
end

julia> function fib3(n)
    n <= 2 ? 1 : (@memo fib3(n-1)) + (@memo fib3(n-2))
end
```

Example 4. memoization – performance tests

```
julia> fib2(4);

julia> @time fib2(40)
  0.000178 seconds (58 allocations: 2.328 KiB)
102334155


julia> fib3(4);

julia> @time fib3(40)
  0.000183 seconds (58 allocations: 2.328 KiB)
102334155
```

Why Metaprogramming Success examples

1. StaticArrays.jl – fast computing with small arrays

```
# Create an SVector using various forms, using
constructors, functions or macros
v1 = SVector(1, 2, 3)
v2 = SVector{3,Float64}(1, 2, 3) # length 3, eltype
Float64
v3 = @SVector [1, 2, 3]
```

```
============================================
    Benchmarks for 3×3 Float64 matrices
============================================

Matrix multiplication               -> 8.2x speedup
Matrix multiplication (mutating)    -> 3.1x speedup
Matrix addition                     -> 45x speedup
Matrix addition (mutating)          -> 5.1x speedup
Matrix determinant                  -> 170x speedup
Matrix inverse                      -> 125x speedup
Matrix symmetric eigendecomposition -> 82x speedup
Matrix Cholesky decomposition       -> 23.6x speedup
```

Source:
https://github.com/JuliaArrays/StaticArrays.jl

2. JuMP.jl – linear and nonlinear programming

```
using JuMP, Clp
m =Model(solver = ClpSolver());
@variable(m, x₁ >= 0)
@variable(m, x₂ >= 0)
@objective(m, Min, 50x₁ + 70x₂)
@constraint(m, 100x₁ + 1000x₂ >= 900)
@constraint(m, 30x₁ +   20x₂ >=  500)
@constraint(m, 7x₁   +   11x₂ >=   60)
solve(m)
```

3. Distributed computing built-in to the language

This single instruction causes the **for** loop to iterate over all workers on all nodes within the computing cluster

```
res = @distributed (append!) for s in sweep
    rng = deepcopy(rngs[myid()])
    profit = 0.0
    for sim in 1:5000
        profit += sim_inventory(s[1],s[2],days=s[3],rng=rng)
    end
    DataFrame(worker=myid(), reorder_q=s[1], reorder_point=s[2],
        days=s[3], profit=profit/5000)
end
```