rjit

Olivier Flückiger

Petr Maj
Jan Vitek

SCG Seminar, Oct 27 2015

Outline

R

a language for statistical computing and graphics

thriving ecosystem rooted in the statistics community

R

S family

lazy evaluation

R

weak abstraction boundaries

really dynamic

R implementation

slow ast interpreter / libraries in C and Forth

single threaded / not thread safe

slow garbage collector / no defragmentation

R interpreter

Naive ast interpreter

+ Simple bytecode interpreter

R interpreter

Naive ast interpreter

(semantically mismatching)

+ Simple bytecode interpreter

rjit

LLVM based jit compiler for R

reuse as many components as possible from legacy implementation

full compatibility with existing packages

rjit

R ast → llvm bitcode

R bytecode

```
switch (opcode) {
case POP_OP:
            POP_OP
case GETVAR_OP:
            GETVAR_OP
case BRIFNOT_OP:
            BRIFNOT_OP

…
}
```

R bytecode

```
switch (opcode) {
case POP_OP:
          POP_OP
case GETVAR_OP:
          GETVAR_OP
case BRIFNOT_OP:
          BRIFNOT_OP

…
}
```

```
void POP_OP() {

}

SEXP GETVAR_OP(SEXP s, SEXP rho){


}

void BRIFNOT_OP(SEXP cond) {


}
```

R bytecode

```
switch (opcode) {
case POP_OP:


case GETVAR_OP:


case BRIFNOT_OP:


…
}
```

```
void POP_OP() {
    POP_OP
}

SEXP GETVAR_OP(SEXP s, SEXP rho){
    GETVAR_OP
}

void BRIFNOT_OP(SEXP cond) {
    BRIFNOT_OP
}
```

R bytecode

```
Native add Function:
   call POP_OP
   call POP_OP
   call ADD_OP
   call RETURN_OP
```

```
void POP_OP() {
    POP_OP
}

SEXP GETVAR_OP(SEXP s, SEXP rho){
    GETVAR_OP
}

void BRIFNOT_OP(SEXP cond) {
    BRIFNOT_OP
}
```

R bytecode

```
SEXP GETVAR_OP(SEXP s, SEXP rho){
    GETVAR_OP
}

void BRIFNOT_OP(SEXP cond) {
    BRIFNOT_OP
}
```

R bytecode

```
function() {
  if(a)
    b
  else
    c
}
```

```
SEXP GETVAR_OP(SEXP s, SEXP rho){
    GETVAR_OP
}

void BRIFNOT_OP(SEXP cond) {
    BRIFNOT_OP
}
```

R bytecode

```
define @rfunction(%body, %rho, %useCache) {
start:
    %0 = call @getVar((SEXP*) 13327080, %rho)
    %condition = call @brIfNot(%0)
    br %condition, label %ifTrue, label %ifFalse
ifTrue:
    %1 = call @getVar((SEXP*) 28742656), %rho)
    br label %next
ifFalse:
    %2 = call @getVar((SEXP*) 16660224, %rho)
    br label %next
next:
    %3 = phi [%1, %ifTrue], [%2, %ifFalse]
    ret %3
}
```

```
SEXP GETVAR_OP(SEXP
    GETVAR_OP
}

void BRIFNOT_OP(SEX
    BRIFNOT_OP
}
```

R bytecode

```
define @rfunction(%body, %rho, %useCache) {
start:
    %0 = call @getVar((SEXP*) 13327080, %rho)
    %condition = call @brIfNot(%0)
    br %condition, label %ifTrue, label %ifFalse
ifTrue:
    %1 = call @getVar((SEXP*) 28742656), %rho)
    br label %next
ifFalse:
    %2 = call @getVar((SEXP*) 16660224, %rho)
    br label %next
next:
    %3 = phi [%1, %ifTrue], [%2, %ifFalse]
    ret %3
}
```

```
SEXP GETVAR_OP(SEXP
    GETVAR_OP
}

void BRIFNOT_OP(SEX
    BRIFNOT_OP
}
```

R bytecode

```
define @rfunction(%body, %rho, %useCache) {
start:
    %0 = call @getVar((SEXP*) 13327080, %rho)
    %condition = call @brIfNot(%0)
    br %condition, label %ifTrue, label %ifFalse
ifTrue:
    %1 = call @getVar((SEXP*) 28742656), %rho)
    br label %next
ifFalse:
    %2 = call @getVar((SEXP*) 16660224, %rho)
    br label %next
next:
    %3 = phi [%1, %ifTrue], [%2, %ifFalse]
    ret %3
}
```

```
SEXP GETVAR_OP(SEXP
    GETVAR_OP
}

void BRIFNOT_OP(SEX
    BRIFNOT_OP
}
```

R bytecode

```
define @rfunction(%body, %rho, %useCache) {
start:
   %0 = call @getVar((SEXP*) 13327080, %rho)
   %condition = call @brIfNot(%0)
   br %condition, label %ifTrue, label %ifFalse
ifTrue:
   %1 = call @getVar((SEXP*) 28742656), %rho)
   br label %next
ifFalse:
   %2 = call @getVar((SEXP*) 16660224, %rho)
   br label %next
next:
   %3 = phi [%1, %ifTrue], [%2, %ifFalse]
   ret %3
}
```

```
SEXP GETVAR_OP(SEXP
     GETVAR_OP
}

void BRIFNOT_OP(SEX
     BRIFNOT_OP
}
```

R CC

R CC

$p_1 \leftarrow$ function(a, b, c) a

$p_1 \leftarrow$ function(a, b, c) a

p_1(1, 2, 3)

$p_1 \leftarrow$ function(a, b, c) a

p_1(1, 2, 3)

p_1(b = 2, 1, 3)

$p_1 \leftarrow$ function(a, b, c) a

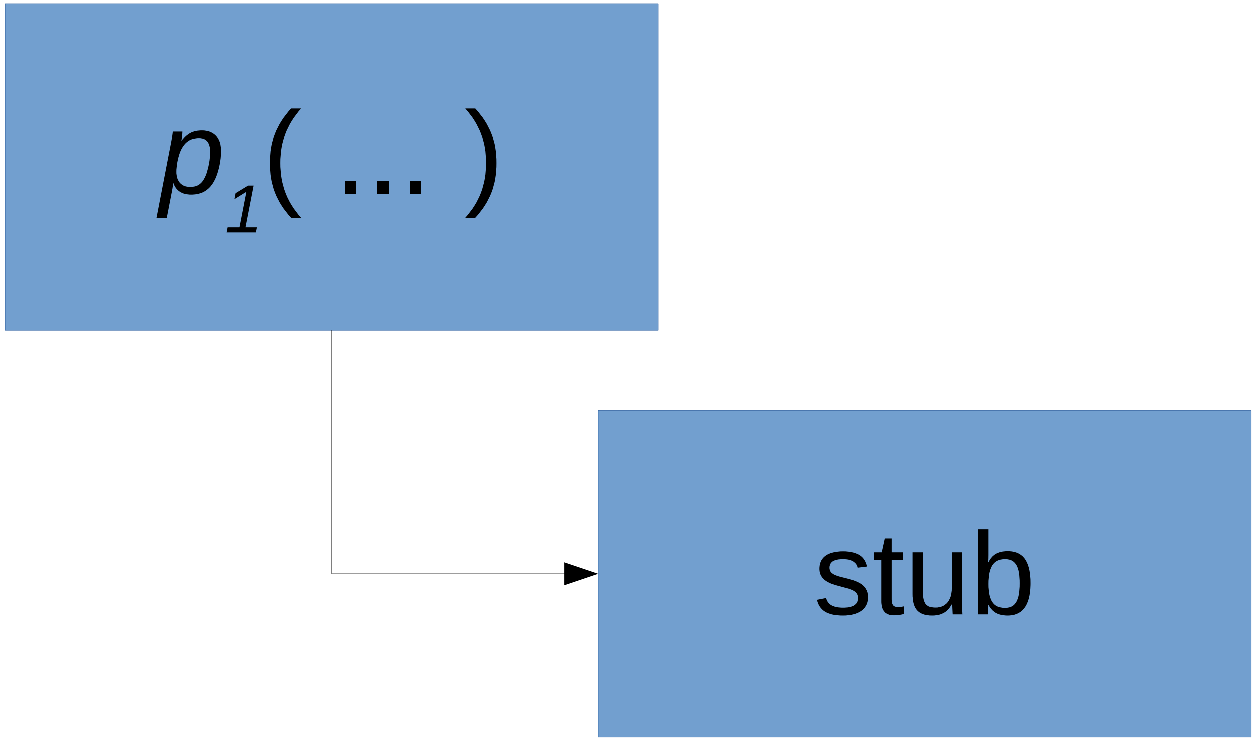
p_1(1, 2, 3)
p_1(b = 2, 1, 3)
p_1(1)

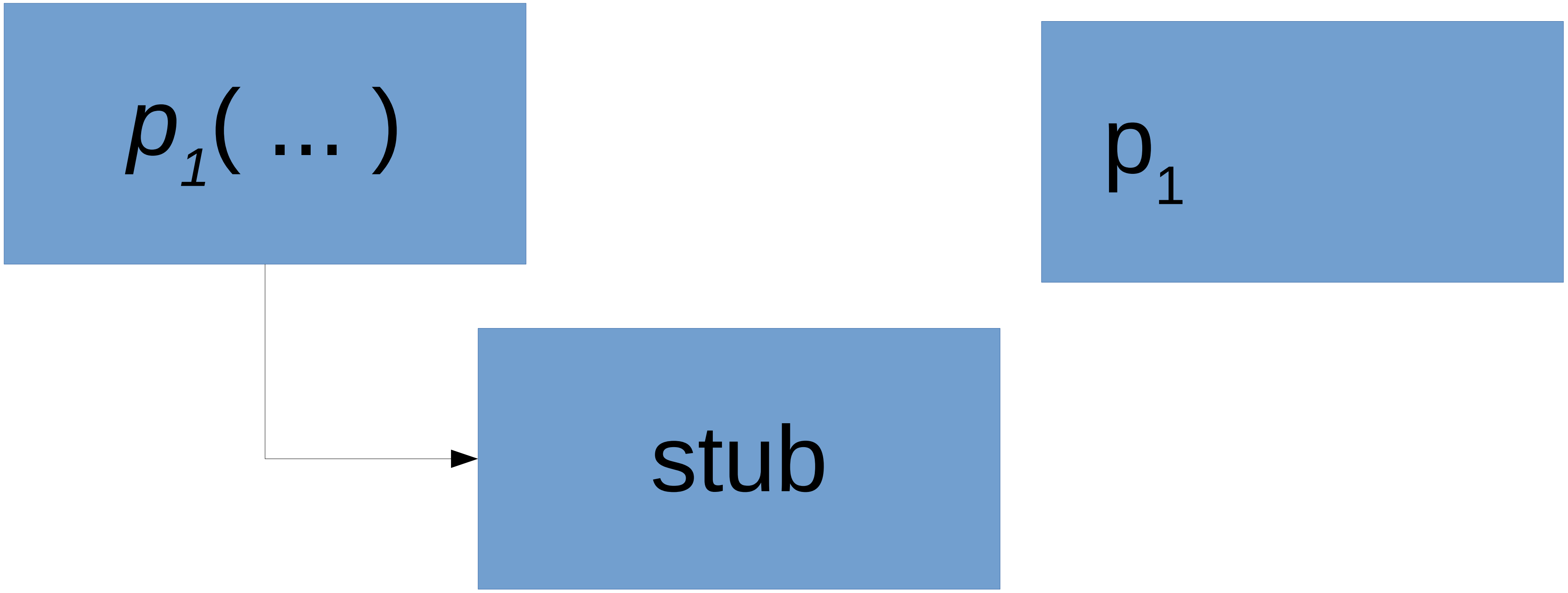
$p_1 \leftarrow$ function(a, b, c) a

p_1(1, 2, 3)

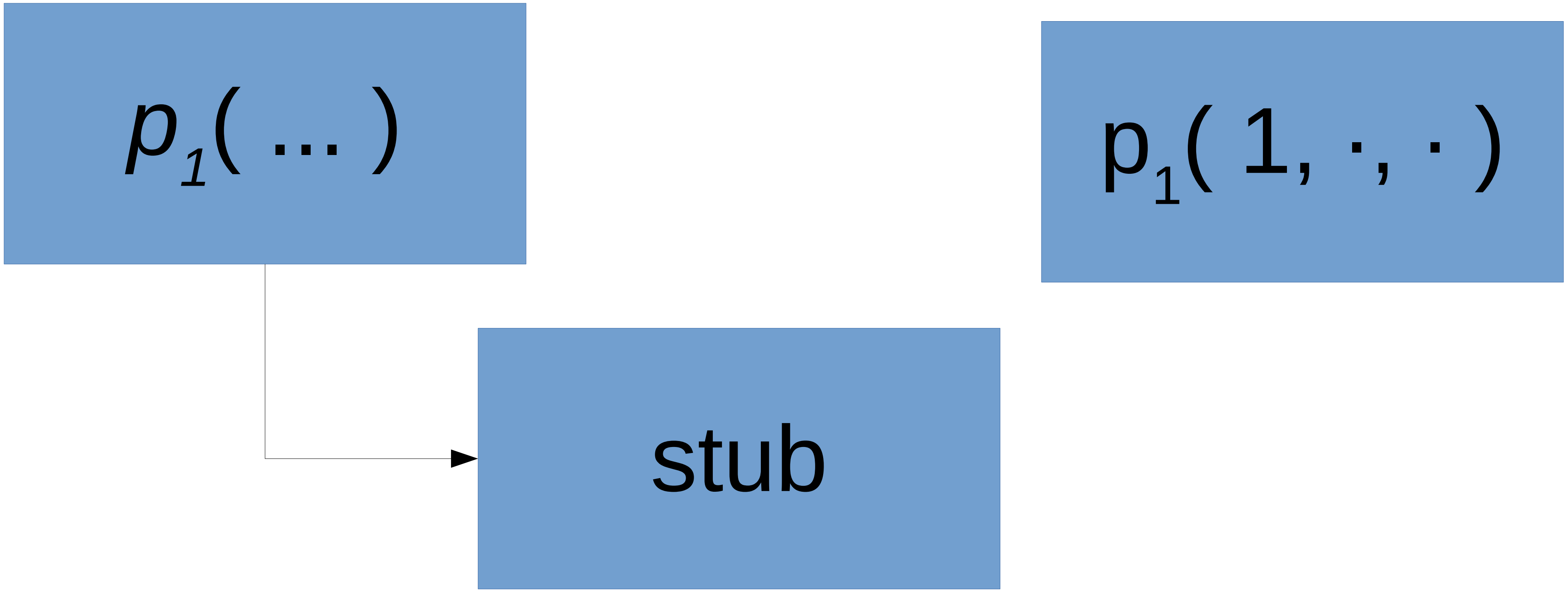
p_1(b = 2, 1, 3) $\rightarrow$ p_1(1, ·, ·)

p_1(1)

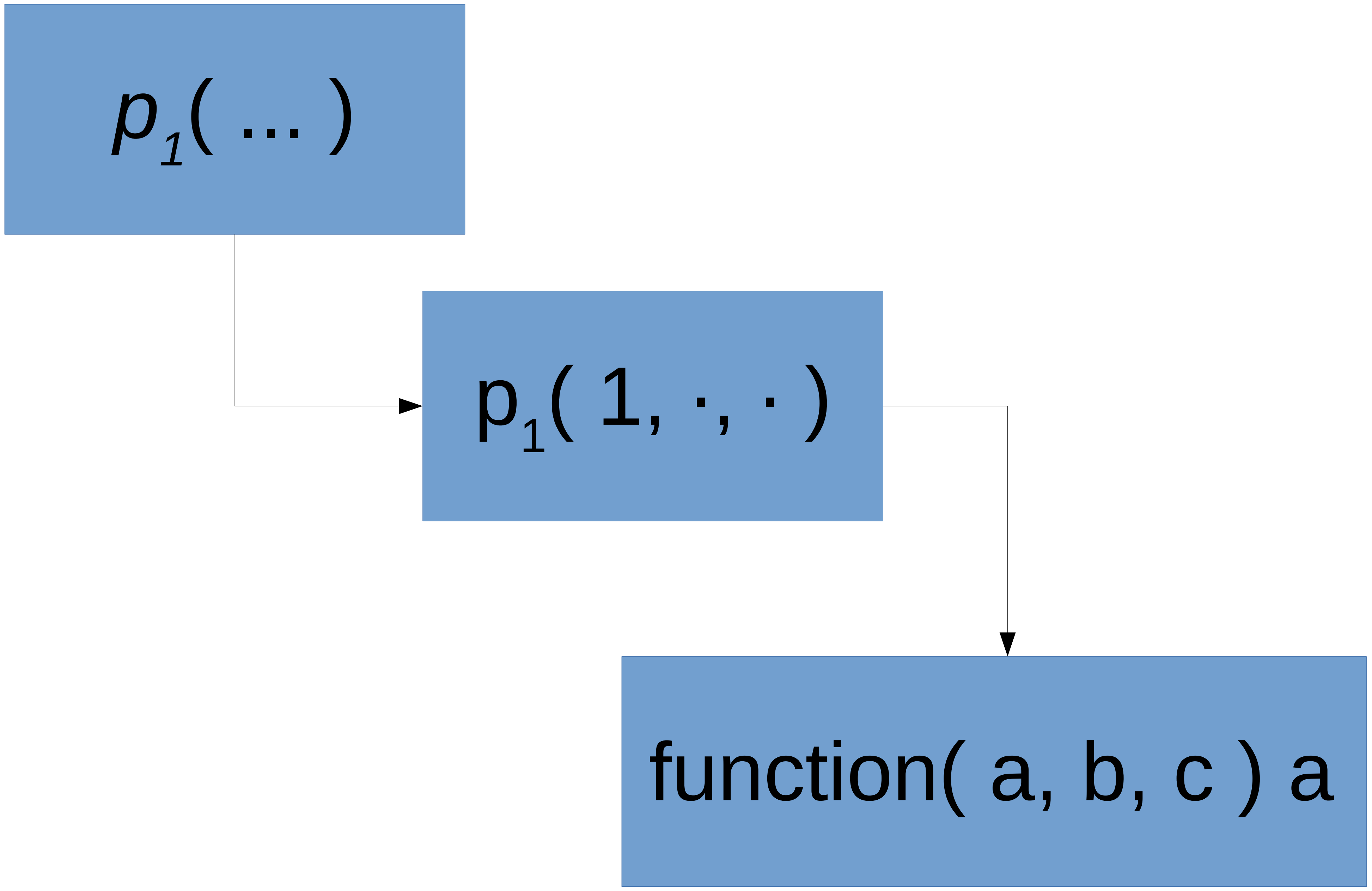

$p_1(\ldots)$
stub



$p_1(\ldots)$
stub
p_1


$p_1(\dots)$
stub
$\mathrm{p}_1(1, \cdot, \cdot)$

$$\begin{tikzcd}
p_1(\ \ldots\ ) \arrow[dr] & \\
& \mathrm{p}_1(\ 1,\ \cdot,\ \cdot\ )
\end{tikzcd}$$


$p_1(\dots)$
$p_1(1, \cdot, \cdot)$
function(a, b, c) a

rjit status

passes non-trivial part of R testsuite

only small part of the language is understood by the compiler

rjit outlook

generic bitcode optimization framework

rjit outlook

optimistic optimization

rjit outlook

sublanguage with stricter semantics