A language of while loops

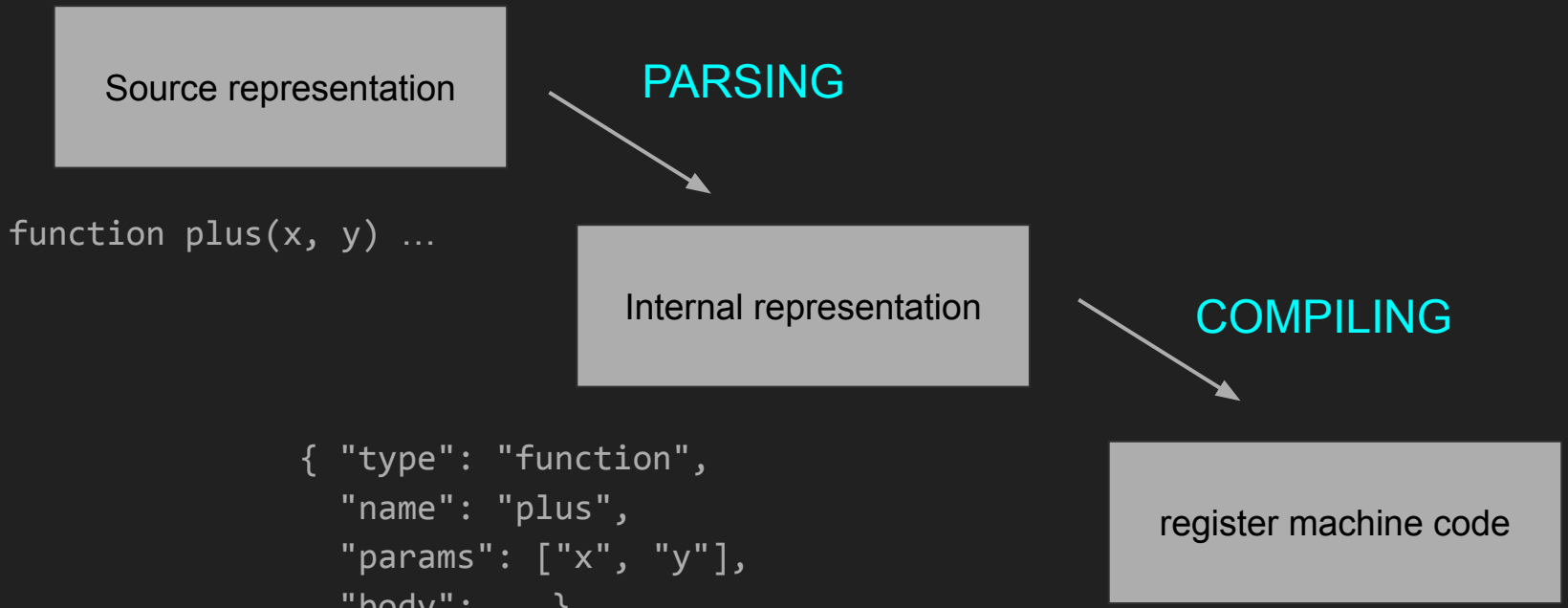
Program ::= F …

F ::= **function** *name*(*var, …*) { SS }

S ::= *var* = E
var ++
var --
if E { S } **else** { S }
while E { S }
return E

E ::= *number*
var
name(E, …)

The compilation process



Compiler interface

Functions:

parse(string)	source program (string) → tree representation
compile(tree)	tree representation → symbolic program with subprograms
expand(prog)	expand subprograms into symbolic program
run(prog)	run a symbolic program

Parsing

Produces a tree representation of the program

Representation of functions:

```
{
  "type": "function",
  "name": name of function,
  "params": parameters of the function,
  "body": tree representation of body statement
}
```

Parsing

Representation of statements:

```
{
  "type": "assign",
  "var": variable assigned,
  "exp": tree representation of expression assigned
}

{
  "type": "if",
  "cond": tree representation of condition expression,
  "then": tree representation of then expression,
  "else": tree representation of else expression
}
...
```

Parsing

Representation of expressions:

```
{
  "type": "number",
  "value": number representation
}
```

```
{
  "type": "var",
  "var": variable name
}
```

```
{
  "type": "call",
  "name": function name to call,
  "args": array of tree
          representations for
          arguments
}
```

Compiling

Transform tree representation into sequence of register machine instructions

We saw the compilation process last time — missing function calls

E⟦ *name*(*E*$_1$, …) ⟧ *target* =

```
TEMPORARY T1
…
E⟦ E1 ⟧ T1
…
EXECUTE name T1,…,target
⟨ empty T1 ⟩
…
```

Expanding and running the program

Expanding: resolve **EXECUTE** pseudo-instructions (Homework 6) into a standalone symbolic program

Running: resolve register names and labels (Homework 5) and run the resulting register machine program