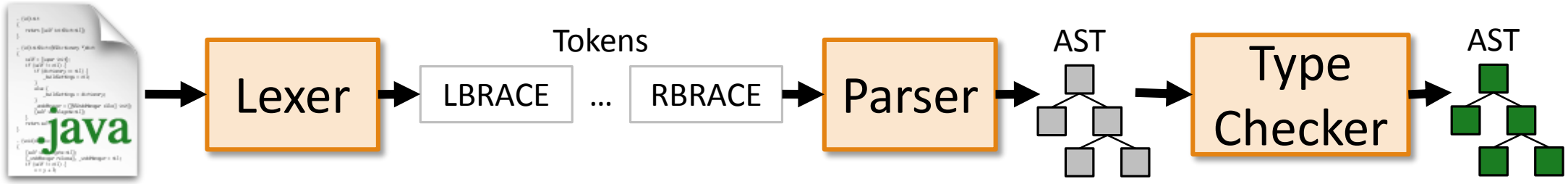
Code Generation

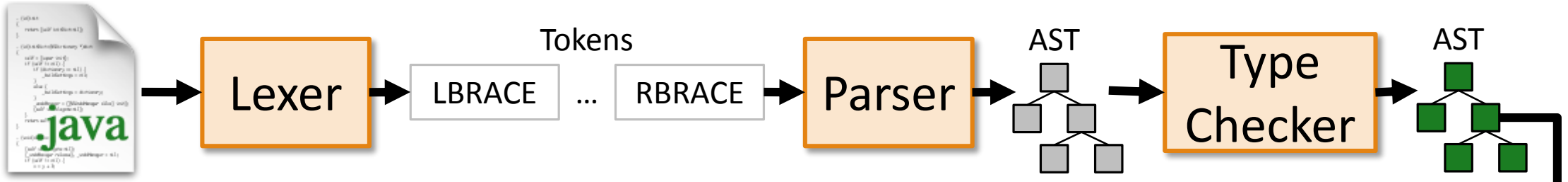
Compiler Overview

Front-end

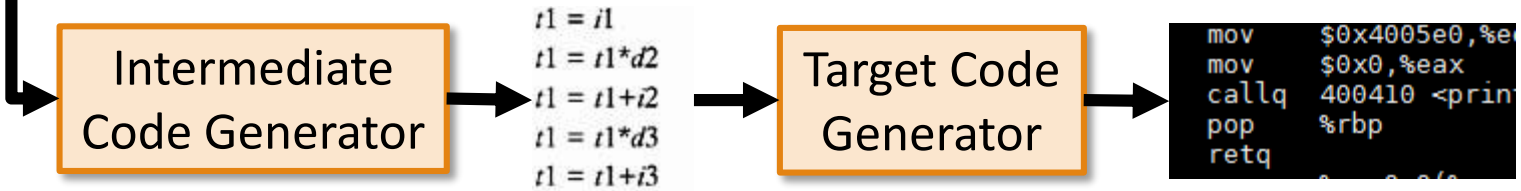


Compiler Overview

Front-end

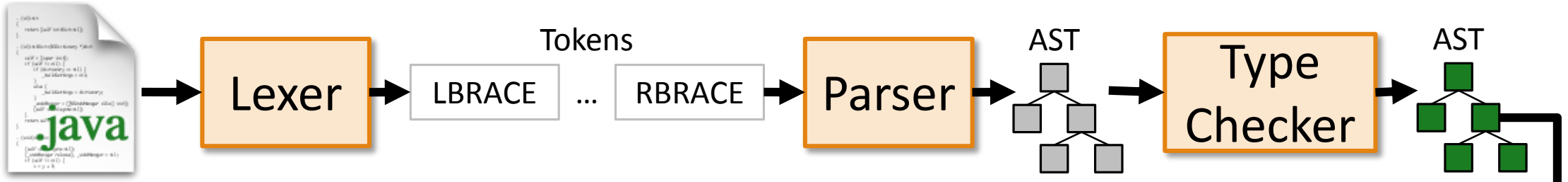


Back-end

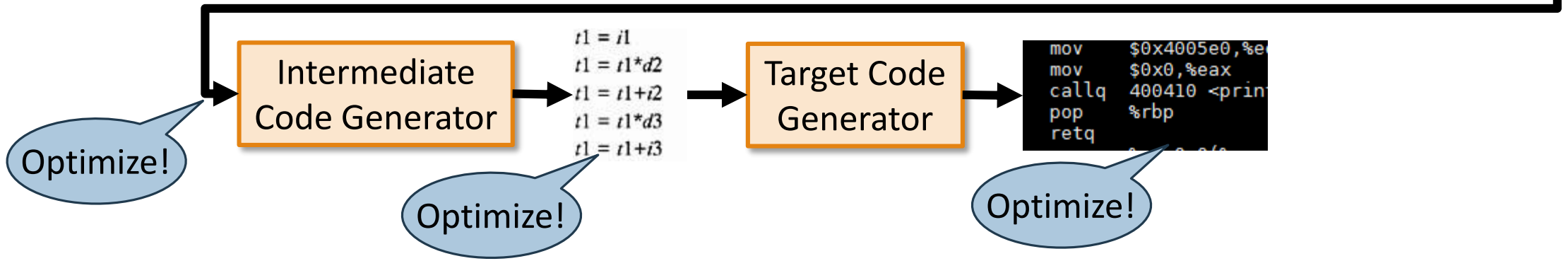


Compiler Overview

Front-end



Back-end



Generating Intermediate Code

Input: Annotated AST

- Encodes the Cool syntax
- File format described in PA4 (and PA3)

Output: 3-address code

- Secretly, this is Cool assembly code

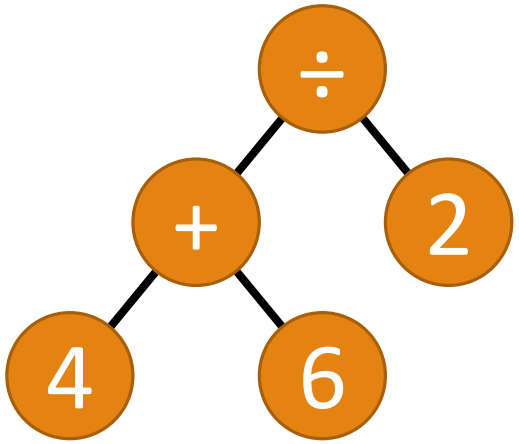
Build control flow, expressions, variables etc.

...

with just a stack, a heap, goto, and 4th-grade math.

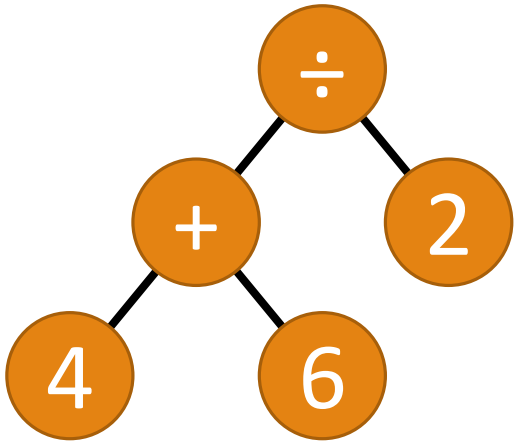
Stack Machines

- Stack + instruction stream.
- No RAM or registers (for now).
- Instructions pop operands, push results.



Stack Machines

- Stack + instruction stream.
- No RAM or registers (for now).
- Instructions pop operands, push results.

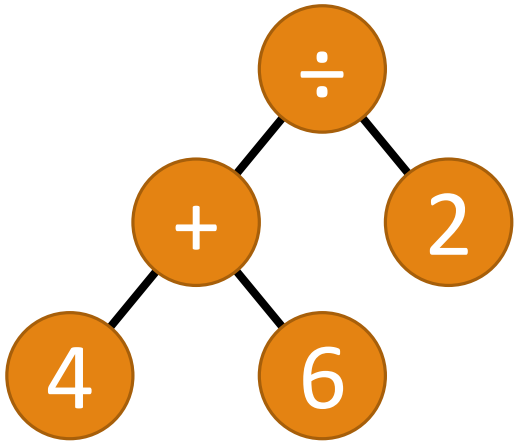



```
push 4
```

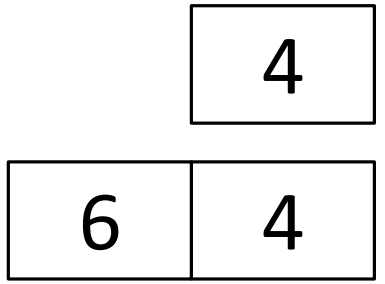
4

Stack Machines

- Stack + instruction stream.
- No RAM or registers (for now).
- Instructions pop operands, push results.

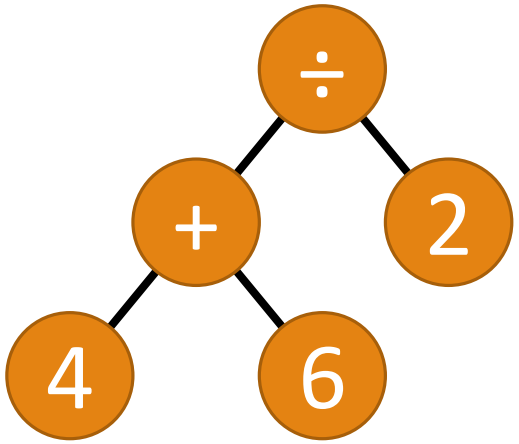


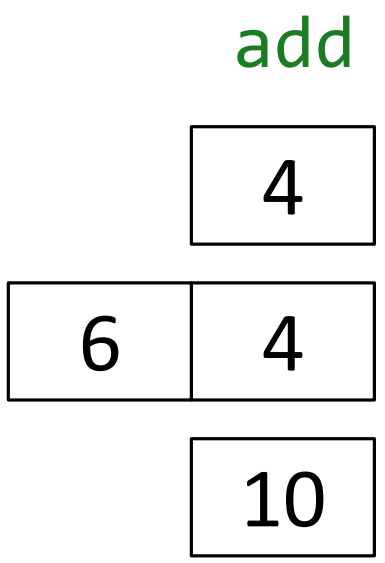

```
push 6
```



Stack Machines

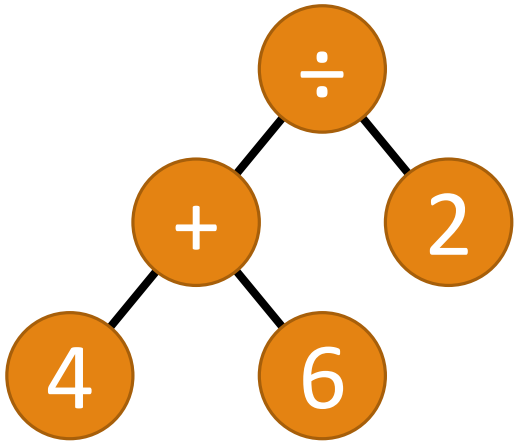
- Stack + instruction stream.
- No RAM or registers (for now).
- Instructions pop operands, push results.



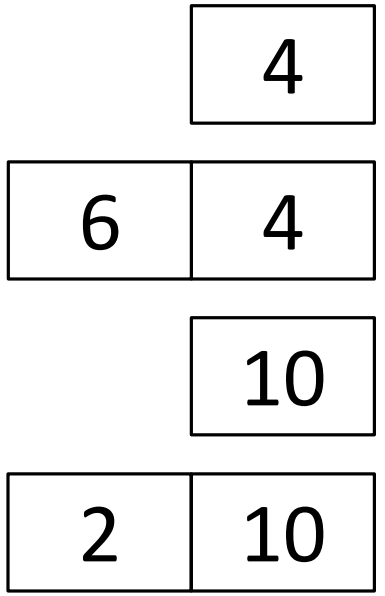


Stack Machines

- Stack + instruction stream.
- No RAM or registers (for now).
- Instructions pop operands, push results.

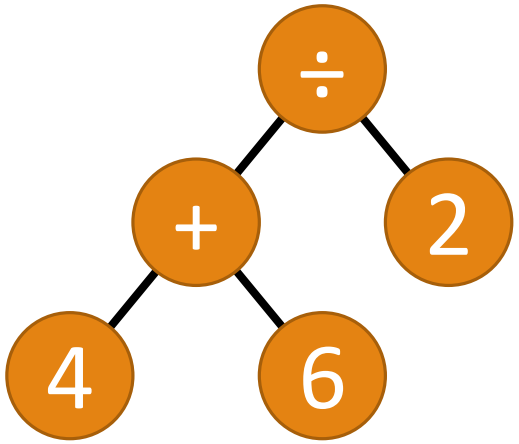



```
push 2
```

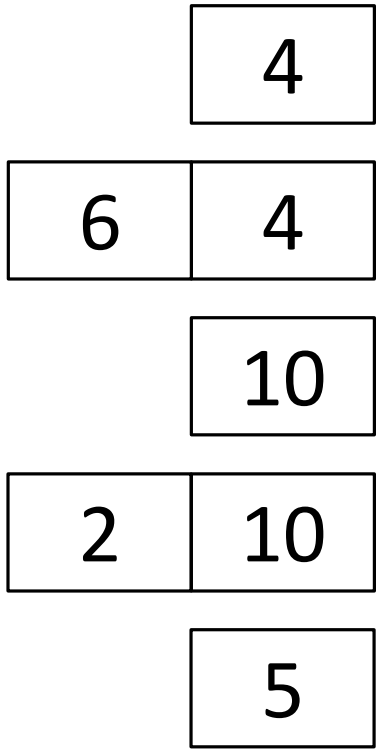


Stack Machines

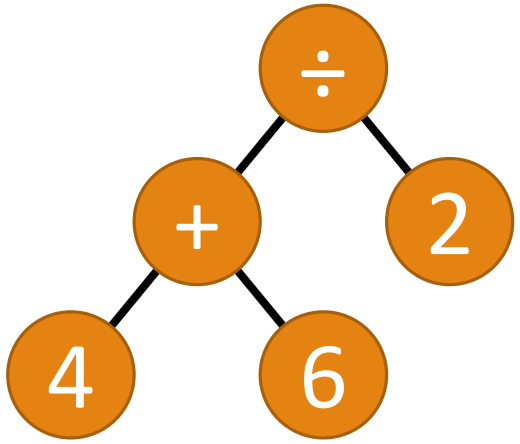
- Stack + instruction stream.
- No RAM or registers (for now).
- Instructions pop operands, push results.




```
divide
```



Stack Machines



?

```
push 4
push 6
add
push 2
divide
```

How Far Have We Come?

Using Cool's built-in integers and strings,

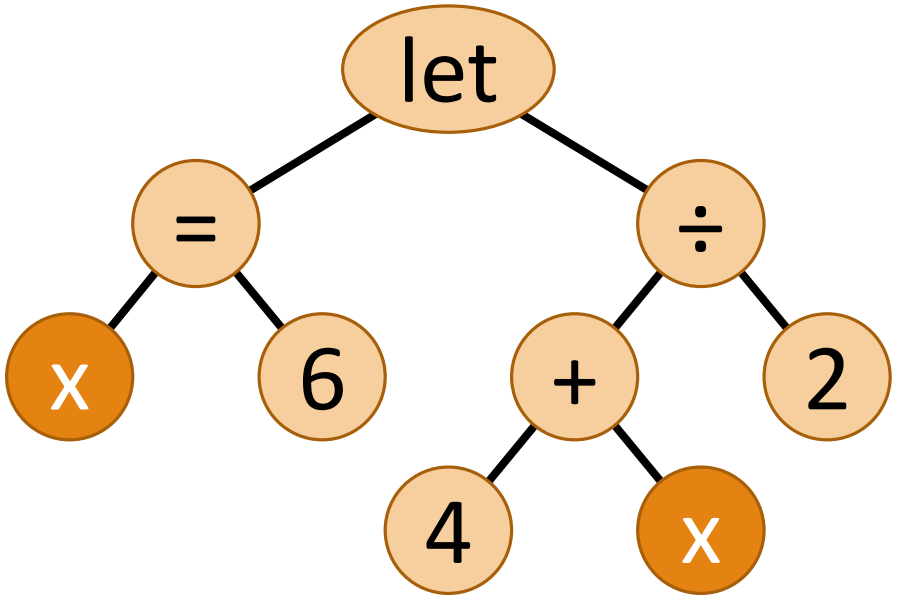
- Which expressions can we implement?

Variables

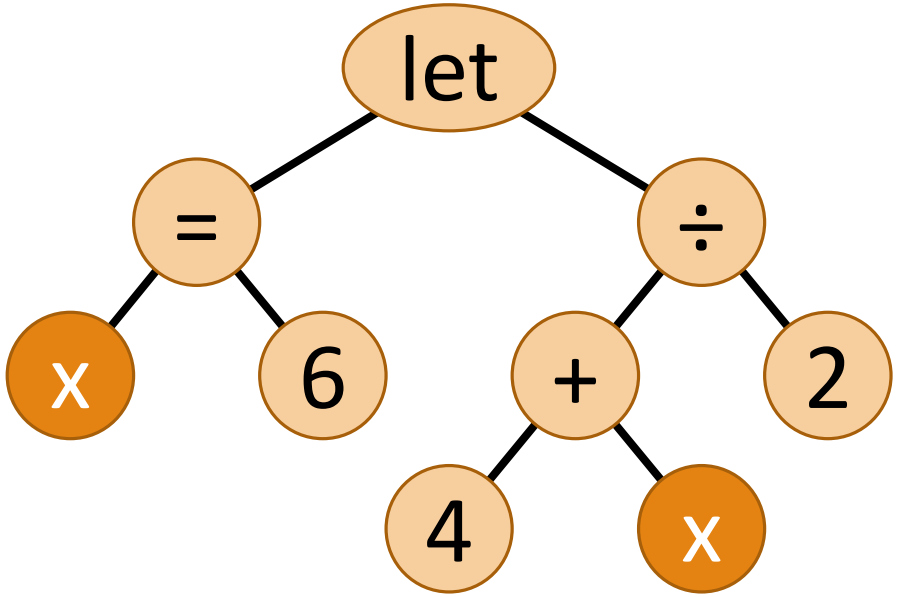
Method parameters

Let expressions

Instance variables (later)

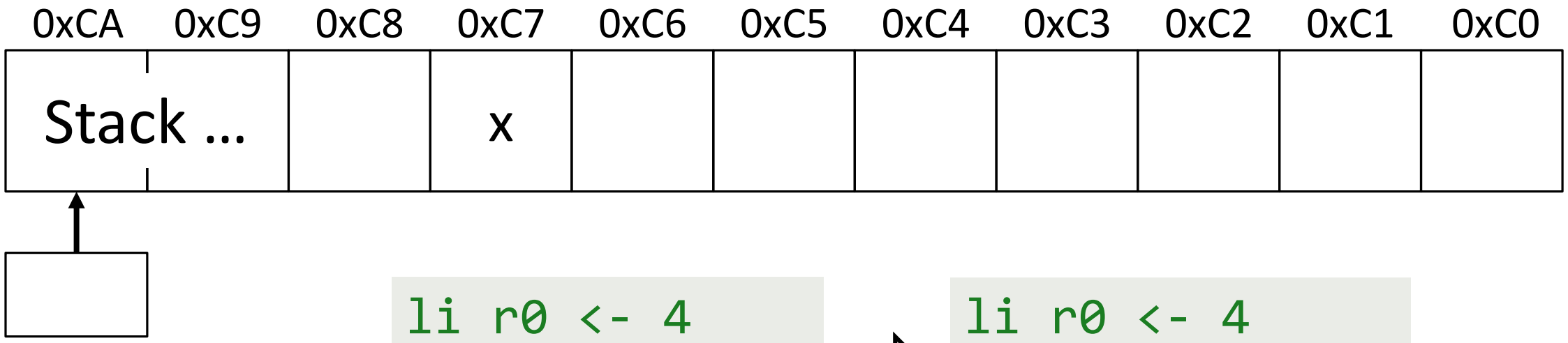


Variables



Need to access data that is ***not on top*** of stack!

Stack Machine with Random Access




```
li r0 <- 4
add r0 <- r0 x
```

```
li r0 <- 4
ld r1 <- fp[-3]
add r0 <- r1 r0
```

Symbol Tables

Track all in-scope variables

- Variable ***name***.
- ***Type*** (size, field locations).
- Memory ***location*** (offset from FP or from object).

Cool uses ***lexical*** scoping, not dynamic ***scoping***.