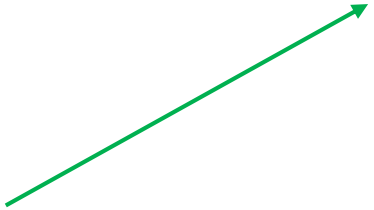
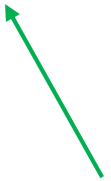
Compilation

The goal of a compiler

Translate from a **source language** to a **target language**



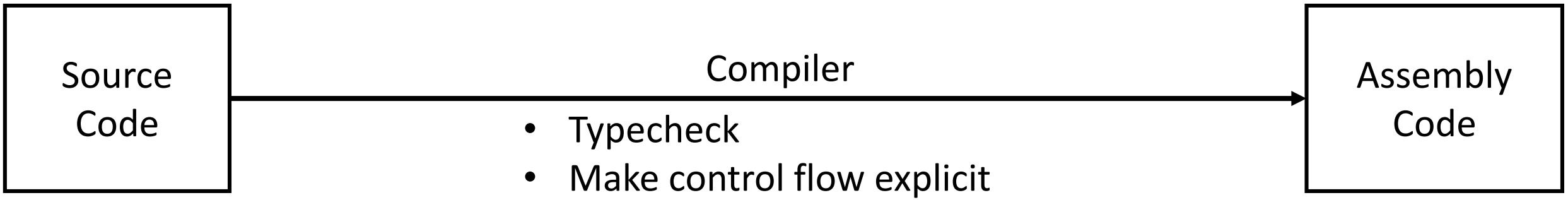


Usually a high-level language

- C
- SML
- Java
- Go
- Rust

Usually a low-level language

- X86 Assembly
- Java Bytecode
- C

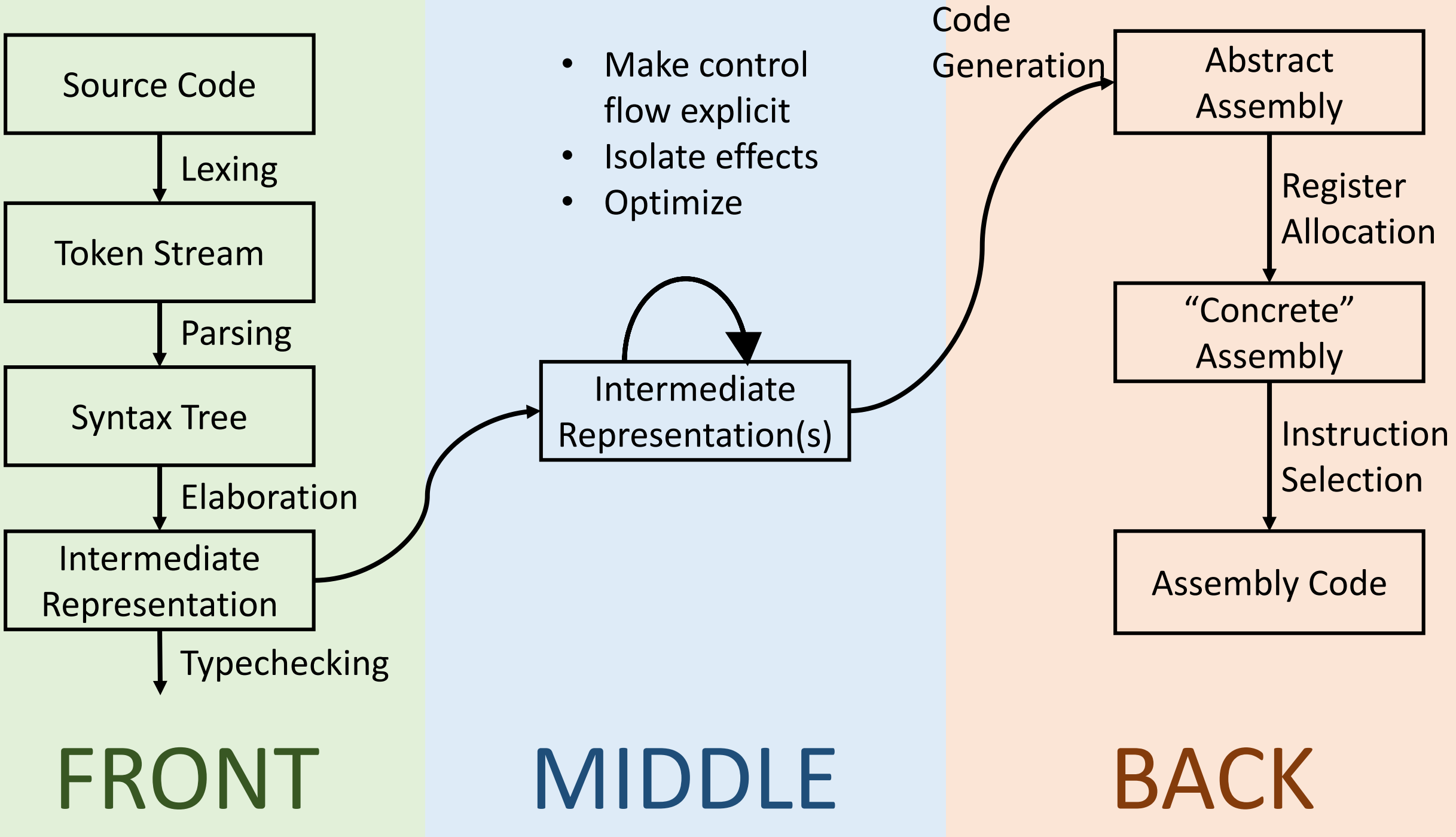

Source Code
Compiler
Typecheck
Make control flow explicit
Flatten expressions
Optimize
Allocate variables to registers
Assembly Code

Example Source Language: VSIMPL

Arithmetic Expression	$aexp$	::=	x	variable
			c	numerical constant
			$aexp$ + $aexp$	addition
			$aexp$ - $aexp$	subtraction
			$aexp$ * $aexp$	multiplication
			$aexp$ / $aexp$	division
Boolean Expression	$bexp$	::=	`true`	true constant
			`false`	false constant
			`!`$bexp$	logical negation
			$bexp$ `||` $bexp$	logical or
			$bexp$ `&&` $bexp$	logical and
			$aexp$ `==` $aexp$	equality
			$aexp$ `!=` $aexp$	inequality
			$aexp$ < $aexp$	less than
			$aexp$ > $aexp$	greater than
Command	cmd	::=	x `:=` $aexp$`;`	assignment
			`if` $bexp$ `{` $cmds$ `}` `else` `{` $cmds$ `}`	conditional
			`while` $bexp$ `{` $cmds$ `}`	loop
			`return` $aexp$`;`	return
Program	$program$	::=	`main` $(params)$ `{` $cmds$ `}`	

Example (Abstract) Assembly Language

Operand	*oper*	::=	x	variable
			c	constant
Instruction	*instruction*	::=	ℓ :	label
			$x \leftarrow oper$	mov
			$x \leftarrow oper + oper$	add
			$x \leftarrow oper - oper$	sub
			$x \leftarrow oper \times oper$	mul
			$x \leftarrow oper \div oper$	div
			$x \leftarrow oper = oper$	eq
			$x \leftarrow oper < oper$	lt
			`JUMP` ℓ	jump
			`IF` *oper* `THEN` ℓ `ELSE` ℓ	conditional jump
			`RET` *oper*	return
Program	*program*	::=	`main` $(params) = instructions$	


Source Code
Lexing
Token Stream
Parsing
Syntax Tree
Elaboration
Intermediate Representation
Typechecking
FRONT
Make control flow explicit
Isolate effects
Optimize
Intermediate Representation(s)
MIDDLE
Code Generation
Abstract Assembly
Register Allocation
"Concrete" Assembly
Instruction Selection
Assembly Code
BACK

Notes that may be useful for the homework

- Intermediate representations (especially pages 4-7)
  http://www.cs.cmu.edu/~janh/courses/411/17/lec/11-irtrees.pdf
- Code Generation (especially pages 1-6)
  http://www.cs.cmu.edu/~janh/courses/411/17/lec/02-instsel.pdf

15-411 (Compiler Design)

^ hey that a good class

Take 15-411 (Compiler Design)

Example (Abstract) Assembly Language

Operand	*oper*	::=	x	variable
			c	constant
Instruction	*instruction*	::=	ℓ :	label
			$x \leftarrow oper$	mov
			$x \leftarrow oper + oper$	add
			$x \leftarrow oper - oper$	sub
			$x \leftarrow oper \times oper$	mul
			$x \leftarrow oper \div oper$	div
			$x \leftarrow oper = oper$	eq
			$x \leftarrow oper < oper$	lt
			`JUMP` ℓ	jump
			`IF` *oper* `THEN` ℓ `ELSE` ℓ	conditional jump
			`RET` *oper*	return
Program	*program*	::=	`main` $(params) = instructions$	