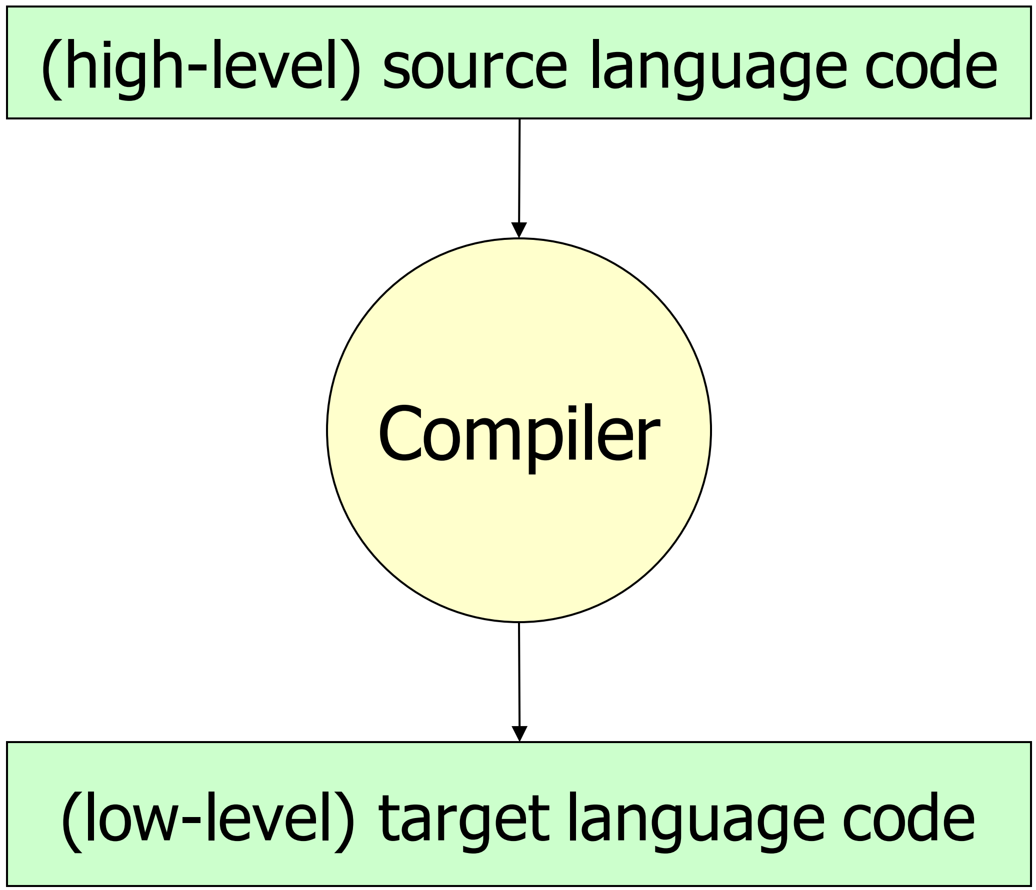
CS 301:
Compiler and Runtime System Design

What is a compiler?
What is a runtime system?

A compiler is a program translator.



A compiler is a program translator.

```
int expr(int n) {
  int d = 4 * n * n * (n + 1) * (n + 1);
  return d;
}
```

```
lda $30,-32($30)
stq $26,0($30)
stq $15,8($30)
bis $30,$30,$15
bis $16,$16,$1
stl $1,16($15)
lds $f1,16($15)
sts $f1,24($15)
ldl $5,24($15)
bis $5,$5,$2
s4addq $2,0,$3
ldl $4,16($15)
mull $4,$3,$2
ldl $3,16($15)
addq $3,1,$4
mull $2,$4,$2
ldl $3,16($15)
addq $3,1,$4
mull $2,$4,$2
stl $2,20($15)
ldl $0,20($15)
br $31,$33
$33:
bis $15,$15,$30
ldq $26,0($30)
ldq $15,8($30)
addq $30,32,$30
ret $31,($26),1
```

A compiler is a program improver.

Unoptimized Code

```
        lda $30,-32($30)
        stq $26,0($30)
        stq $15,8($30)
        bis $30,$30,$15
        bis $16,$16,$1
        stl $1,16($15)
        lds $f1,16($15)
        sts $f1,24($15)
        ldl $5,24($15)
        bis $5,$5,$2
        s4addq $2,0,$3
        ldl $4,16($15)
        mull $4,$3,$2
        ldl $3,16($15)
        addq $3,1,$4
        mull $2,$4,$2
        ldl $3,16($15)
        addq $3,1,$4
        mull $2,$4,$2
        stl $2,20($15)
        ldl $0,20($15)
        br $31,$33
$33:
        bis $15,$15,$30
        ldq $26,0($30)
        ldq $15,8($30)
        addq $30,32,$30
        ret $31,($26),1
```

Optimized Code

```
        s4addq $16,0,$0
        mull $16,$0,$0
        addq $16,1,$16
        mull $0,$16,$0
        mull $0,$16,$0
        ret $31,($26),1
```

A compiler is a program checker.
A compiler is a programmer's assistant.

Static program analysis

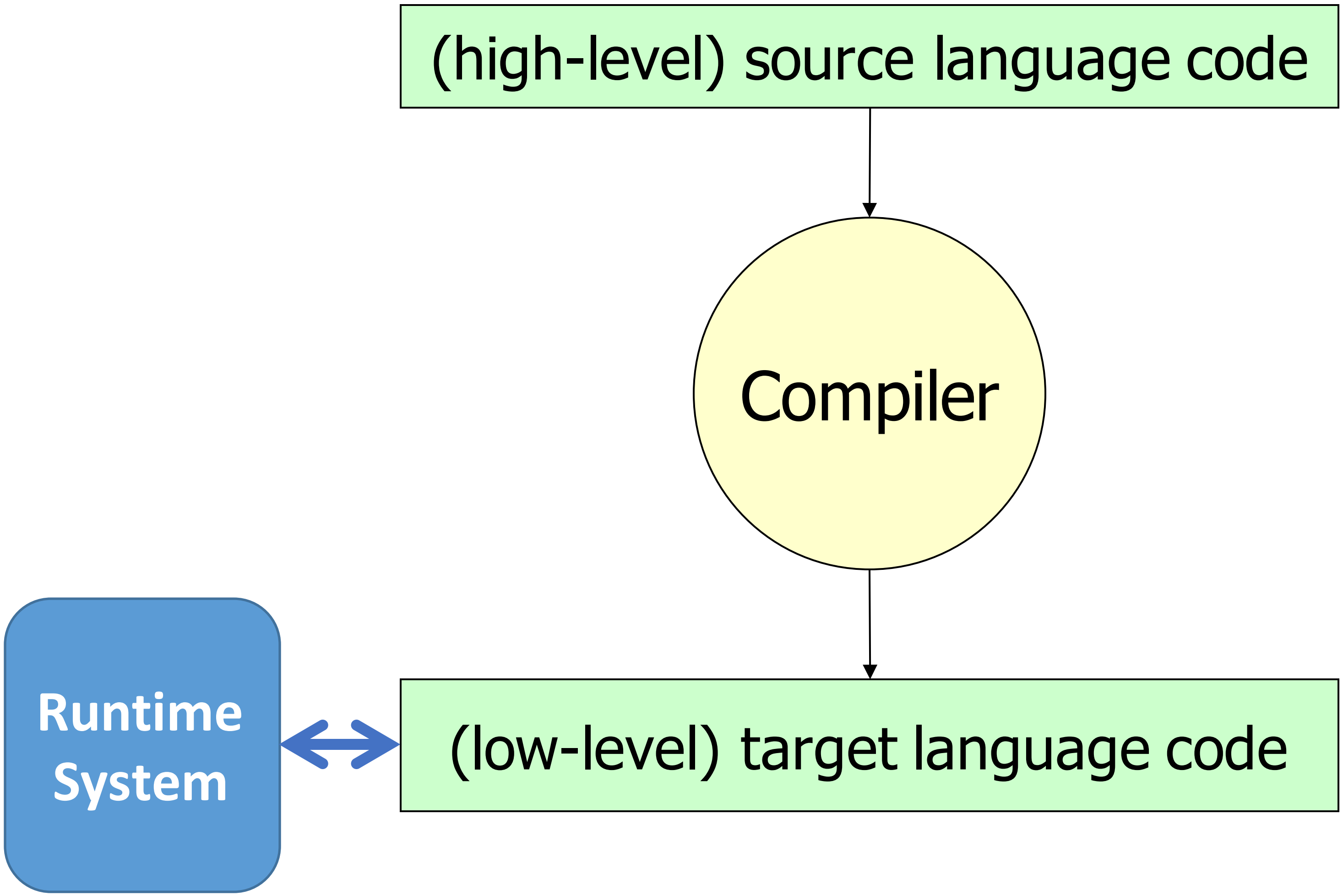
Type checking

Scope checking

Control flow checking

Instrumentation for dynamic program analysis

A runtime system is
a compiler's co-conspirator.



A runtime system is
{ a program translator, a program improver, a program checker, a programmer's assistant, a compiler's co-conspirator }

Memory management + Garbage Collection (GC)

Dynamic program analysis

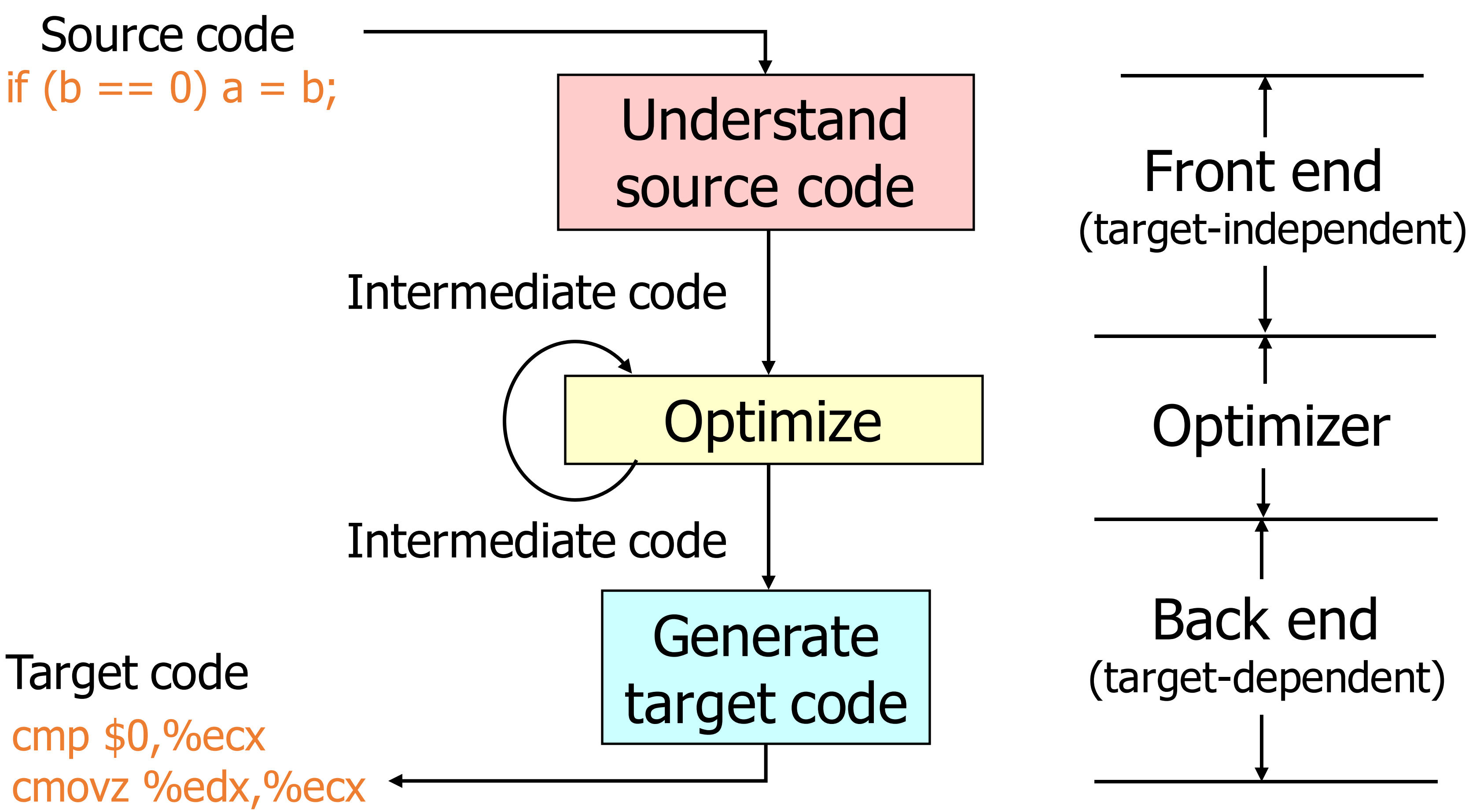
Just in Time compilation (JIT)

Feedback-directed optimization

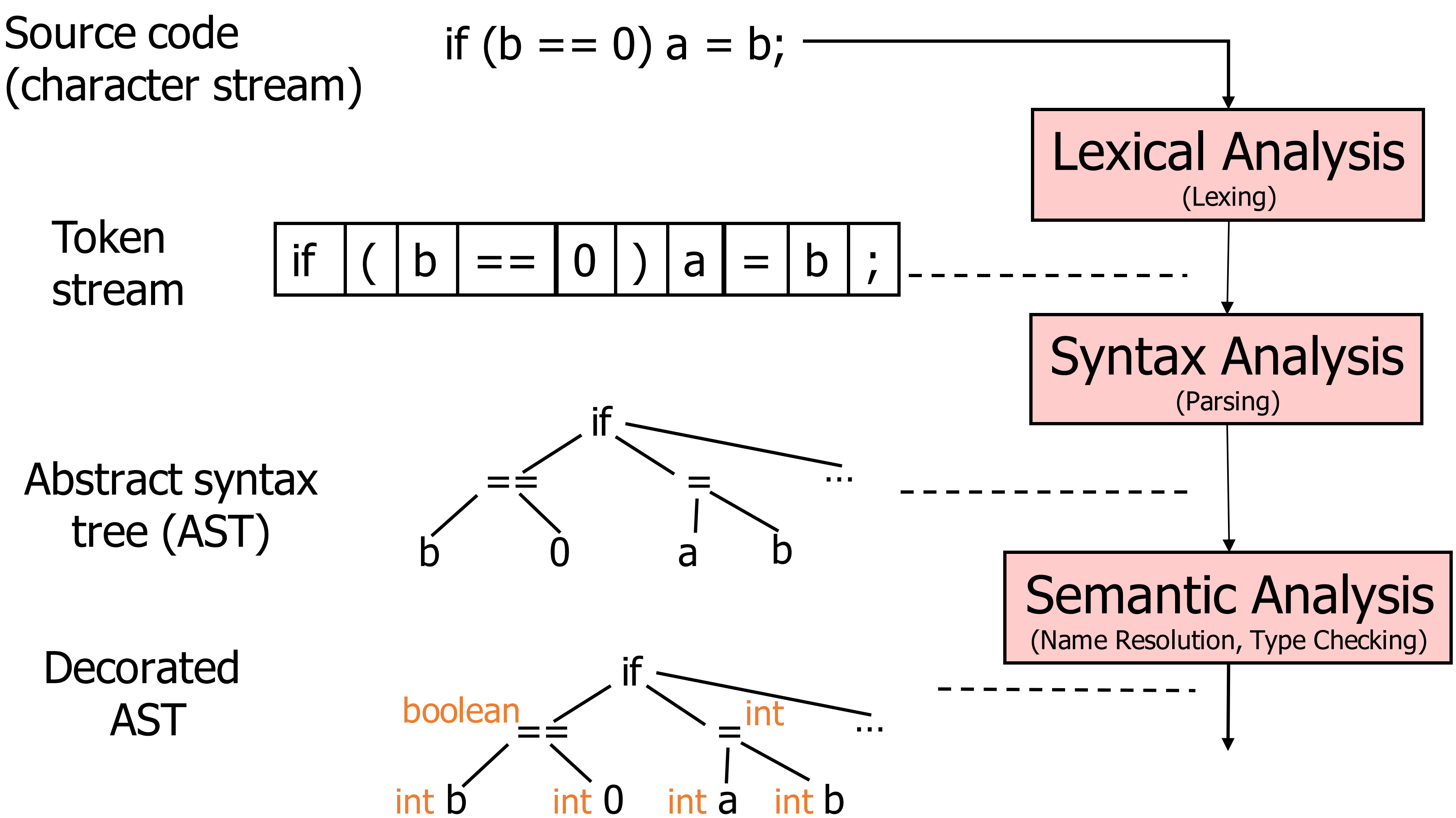
Profiling + profile guided optimization (PGO)

Resource management

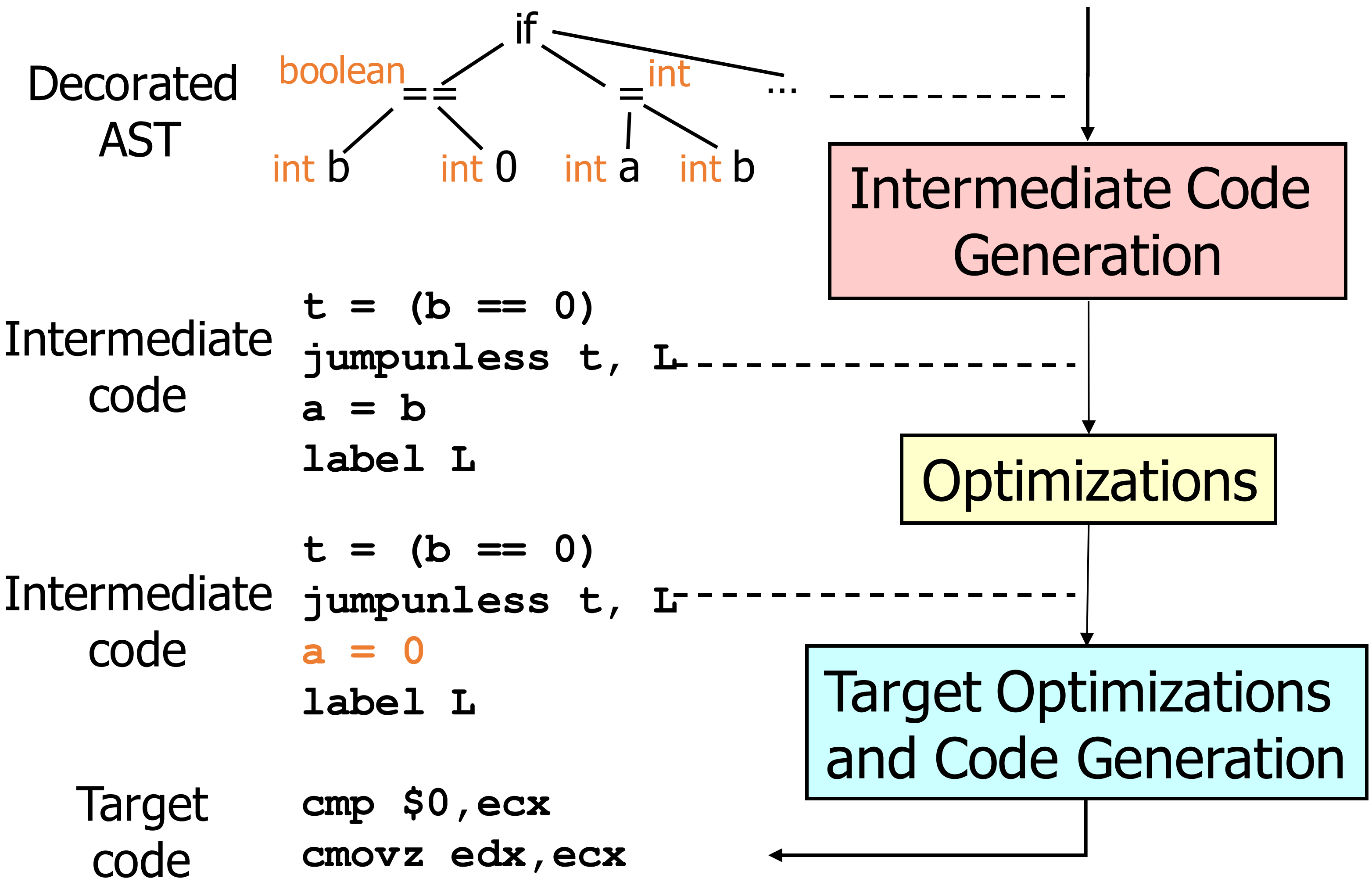
Simplified Compiler Structure



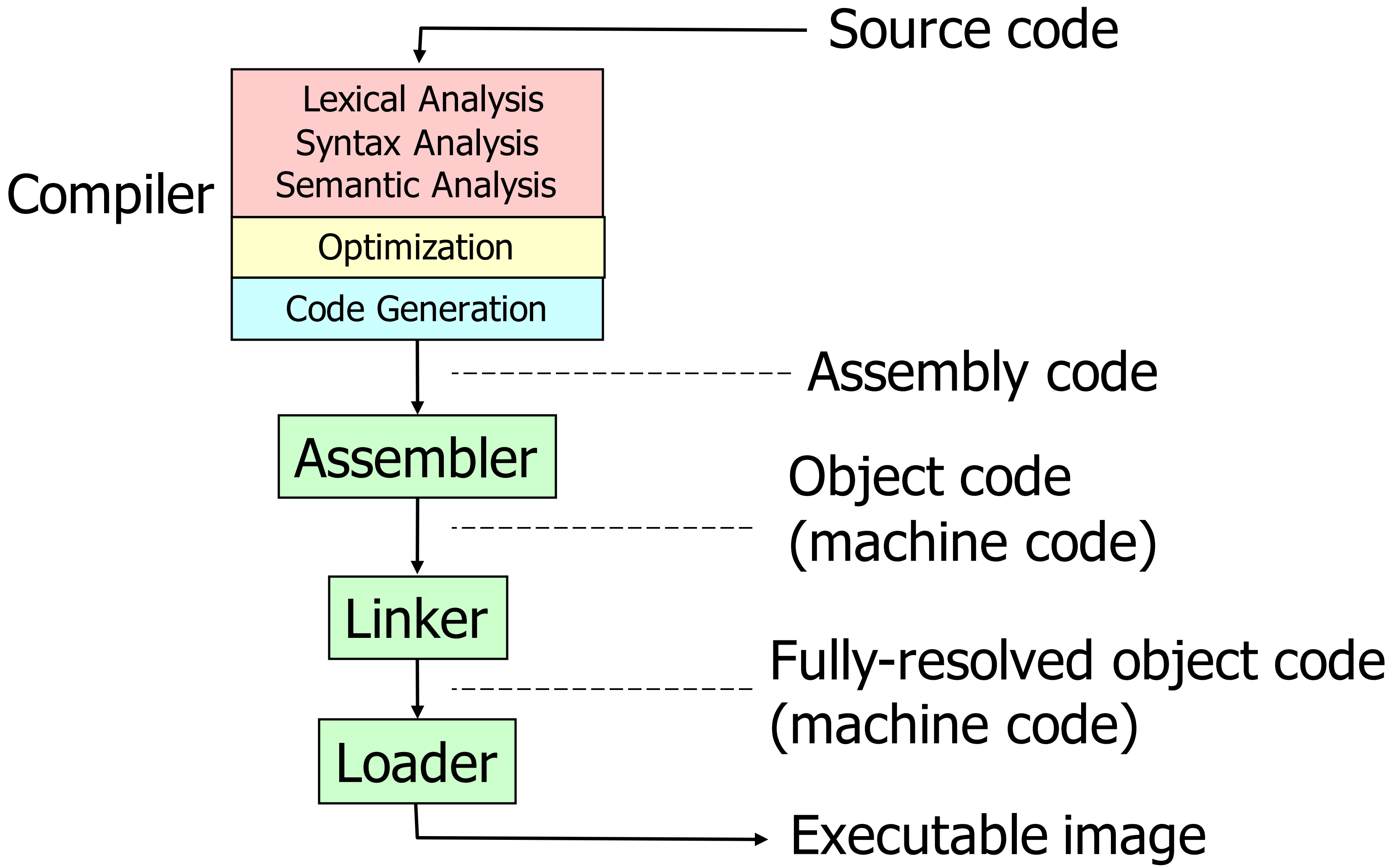
Compiler Front End



Compiler Middle / Back End



Zooming out: compiling to machine code…



Course Material

- Theoretical Foundations
- Implementation
- Synthesis
  - Programming Languages (251), Theory (235), Algorithms (231), Computer Systems (240), Software Construction

- Semester project: full compiler
  - for small statically typed language
  - mostly from scratch
  - implemented in Scala
  - other tools: git, IntelliJ, gcc, gdb, ...
- Tutorial

Project Groups

- groups of 3 (or 2)
- you choose, I can help
- work with new people on the Tiny compiler this week
- Collaboration and Honor Code

Tutorial Meetings

I form trios with your schedule/people input, end of this week.

Weekly assignment for 1-hour small-group meeting with me.

Preparation: Read, work on all problems.

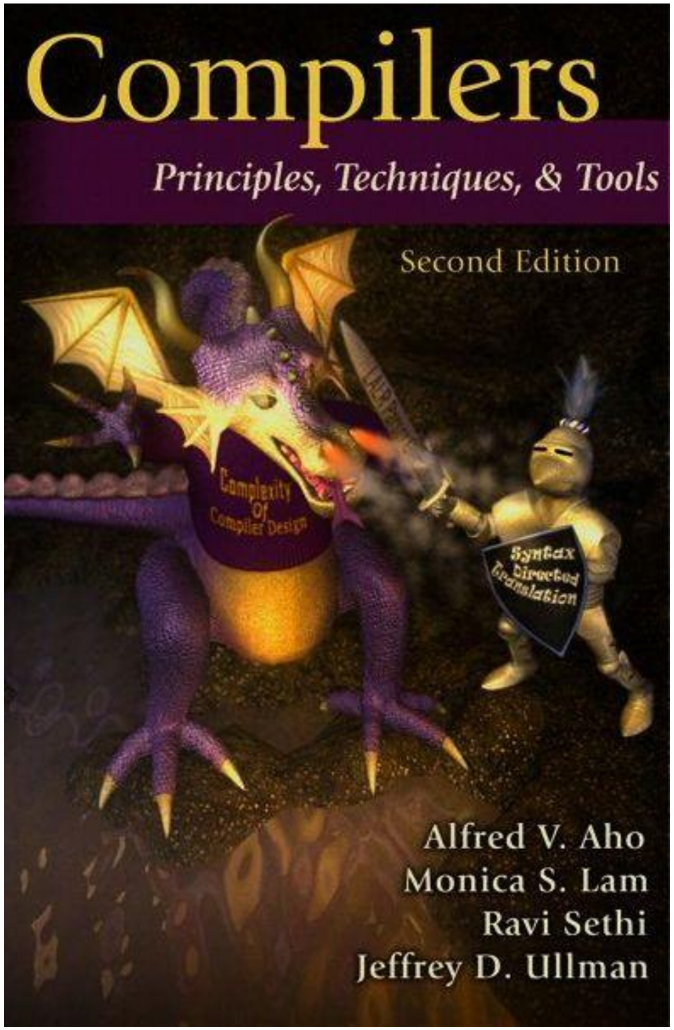
Participation: Discuss any of the problems in detail.

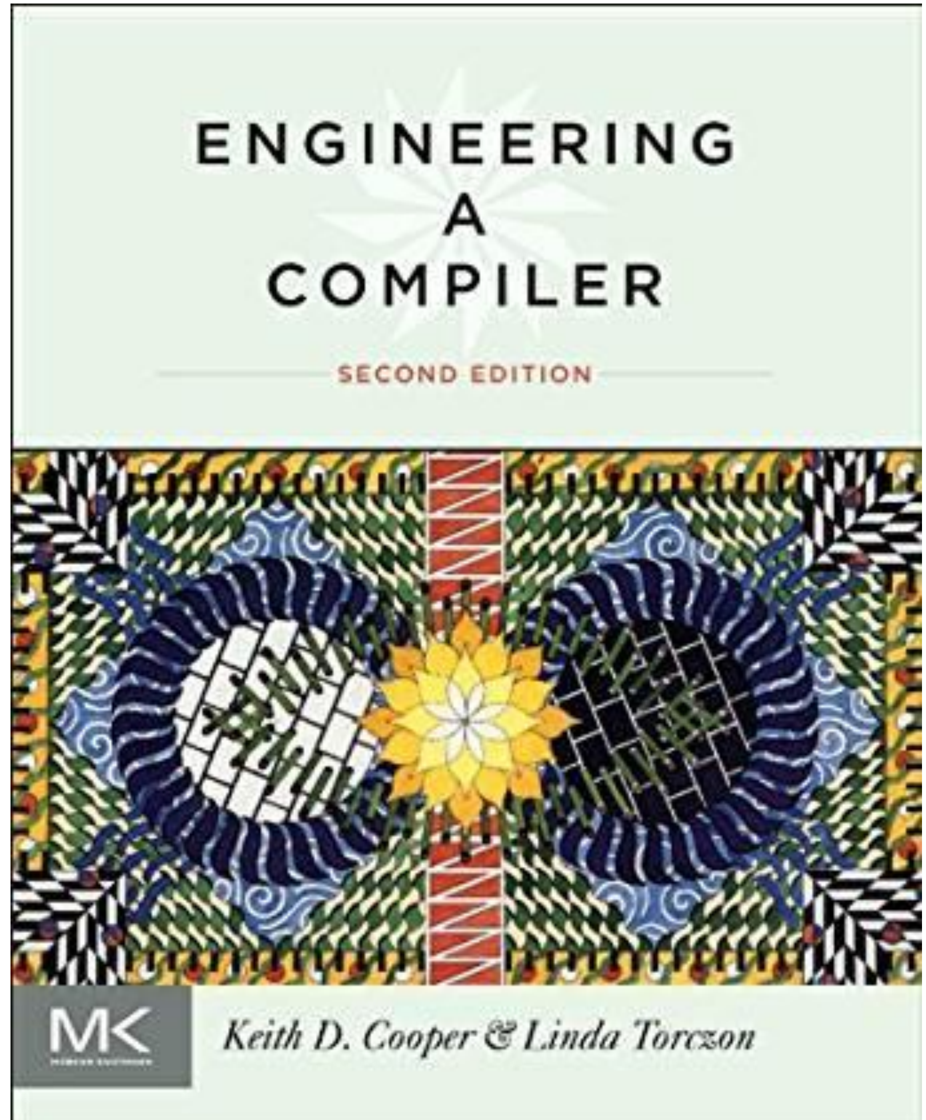
- work through solution on board
- extend problem in new directions
- explain where you got stuck and why

Reviews: Write clean solutions to a few questions I choose.

- in LaTeX
- due 24 hours after your meeting
- better prep before meeting => fewer written solutions after

Books


Compilers
Principles, Techniques, & Tools
Second Edition
Complexity Of Compiler Design
Syntax Directed Translation
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman


ENGINEERING
A
COMPILER
SECOND EDITION
MK
Keith D. Cooper & Linda Torczon

Weekly Schedule

Tutorial Day: Monday/Tuesday/Wednesday

- tutorial meetings
- next tutorial assignment posted
- Tuesday scheduled class time not required after today
  - May be used for tutorial
  - Otherwise: guaranteed project group availability, code review, etc.

Lab/Work Day: Friday scheduled class time (+ drop-in hours after)

- Introduce and start new project stages/checkpoints
- Project work time
- Code reviews
- Occasional mini-lectures framing upcoming tutorial material
- Project checkpoints / due dates

Other Days:

- Code reviews?
- Drop-in hours, appointments

Tiny compiler!

Source code
(character stream)

`if (b == 0) a = b;`

Lexical Analysis

Token stream

if	(	b	==	0	)	a	=	b	;

Identifiers:	`x   y11   elsen   _i00`
Integers:	`2   1000   5L`
Floating point:	`2.0   .02   1.   1e5   0.e-10`
Strings:	`"x"   "She said, \"Hey!\""`
Comments:	`/* don't change this */   // or this`
Keywords:	`if   else   while   break`
Symbols:	`+   *   {   }   =   <   << ==   [   ] >=`

Regular Expressions *(regex, RE)*

A **language** is a set of words: { SCI, KSC }, { a,b,c,d,... }

A **regular expression** describes a (regular) language

`abab    a|b   (a|b)*    [1-9][0-9]*     [a-z][a-z0-9]*`

Regular expression primitives:

a	ordinary character stands for itself
ε	the empty string (*epsilon*)
R|S	either R or S *(alternation)*, where R,S are REs
RS	R followed by S *(concatenation)*
R*	R repeated 0 or more times *(Kleene star)*

L(R) = the language defined by regular expression R

L(**`a(SCI|KSC)`**) = { aSCI, aKSC }

L(**`[1-9][0-9]*`**) = { 1,2,3,4,5,6,7,8,9,10,11,12,13,....}

Regular Expression Extensions

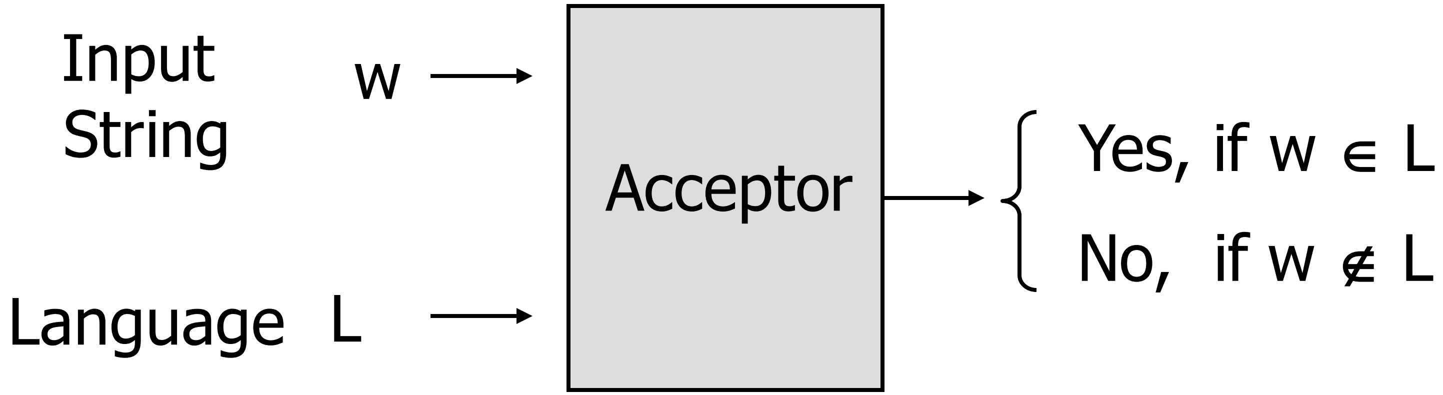
If R is a regular expressions, so are:

`R?`	= `ε|R`	*(zero or one R)*
`R+`	= `RR*`	*(one or more R's)*
`(R)`	= `R`	*(no effect: grouping)*
`[abc]`	= `a|b|c`	*(any of the listed characters)*
`[a-e]`	= `a|b|…|e`	*(character ranges)*
`[^ab]`	= `c|d|…`	*(any character except the listed characters)*
`.`	= `a|b|c|d|…`	*(any character except newline)*
`\.`	= *a literal . character*	
`\*`	= *a literal * character*	
`\\`	= *a literal \ character*	

. . .

Acceptors:
(a.k.a. recognizers)

Abstract machines that determine if an input string belongs to a language, answering Yes/No.

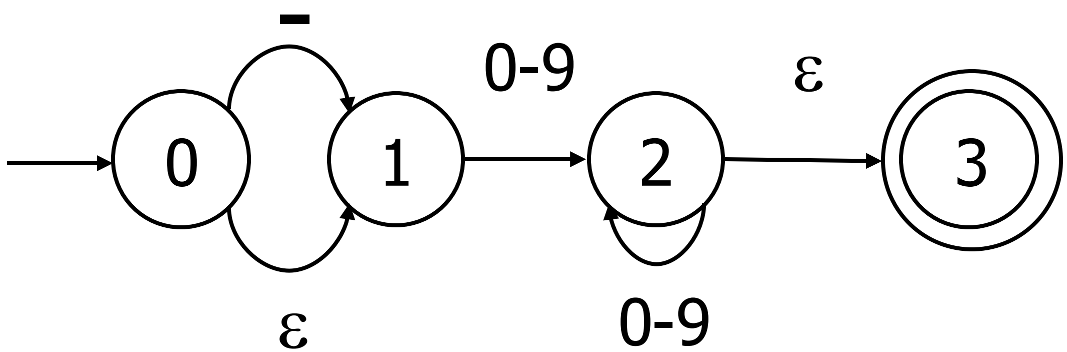


Finite Automata:
acceptors for languages described by regular expressions

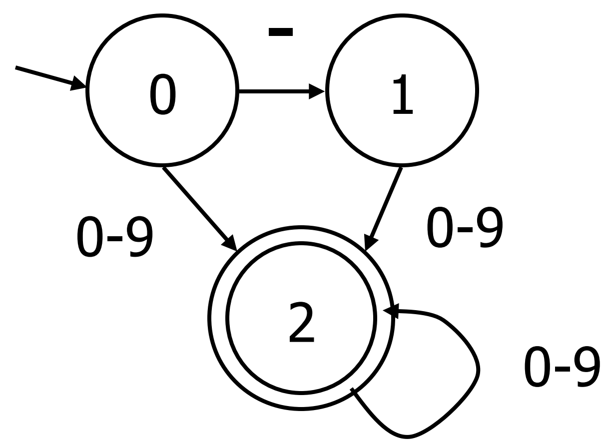
Finite Automata

Regular Expression: (-|ε)[0-9][0-9]*

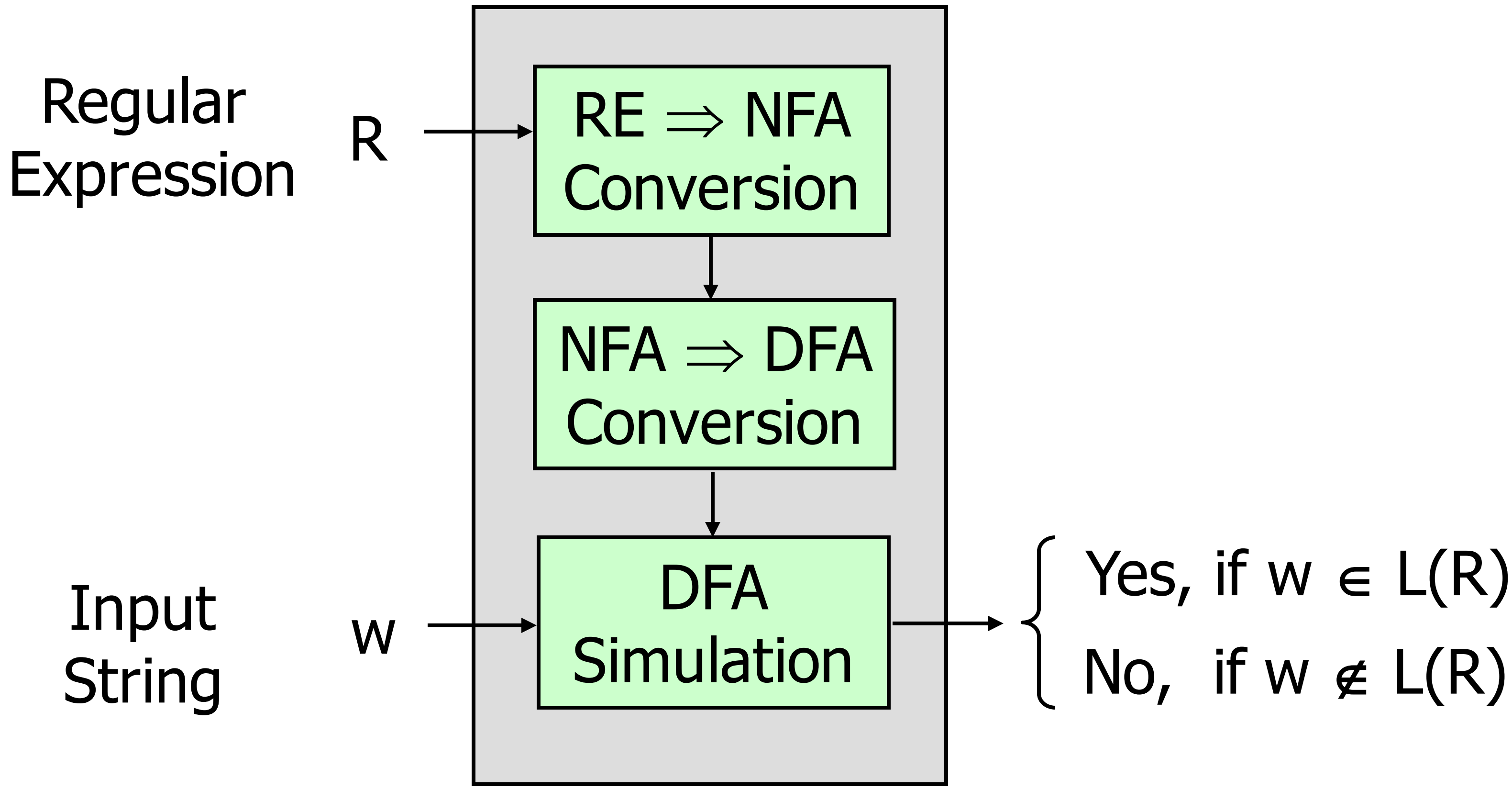
Nondeterministic Finite Automaton:



Deterministic Finite Automaton:



Building an acceptor for a regular expression:



Example Lexer Generator Specification

```
%%

digits     = 0|[1-9][0-9]*
letter     = [A-Za-z]
identifier = {letter}({letter}|[0-9_])*
whitespace = [\ \t\n\r]+

%%

{whitespace} {/* discard */}
{digits}     { return new Token(INT, parseInt(yytext()); }
"if"         { return new Token(IF, yytext()); }
"while"      { return new Token(WHILE, yytext()); }
…
{identifier}      { return new Token(ID, yytext()); }
```