Checkin 7

Show the AST for the expression

(1 + x) * 2

Administrivia

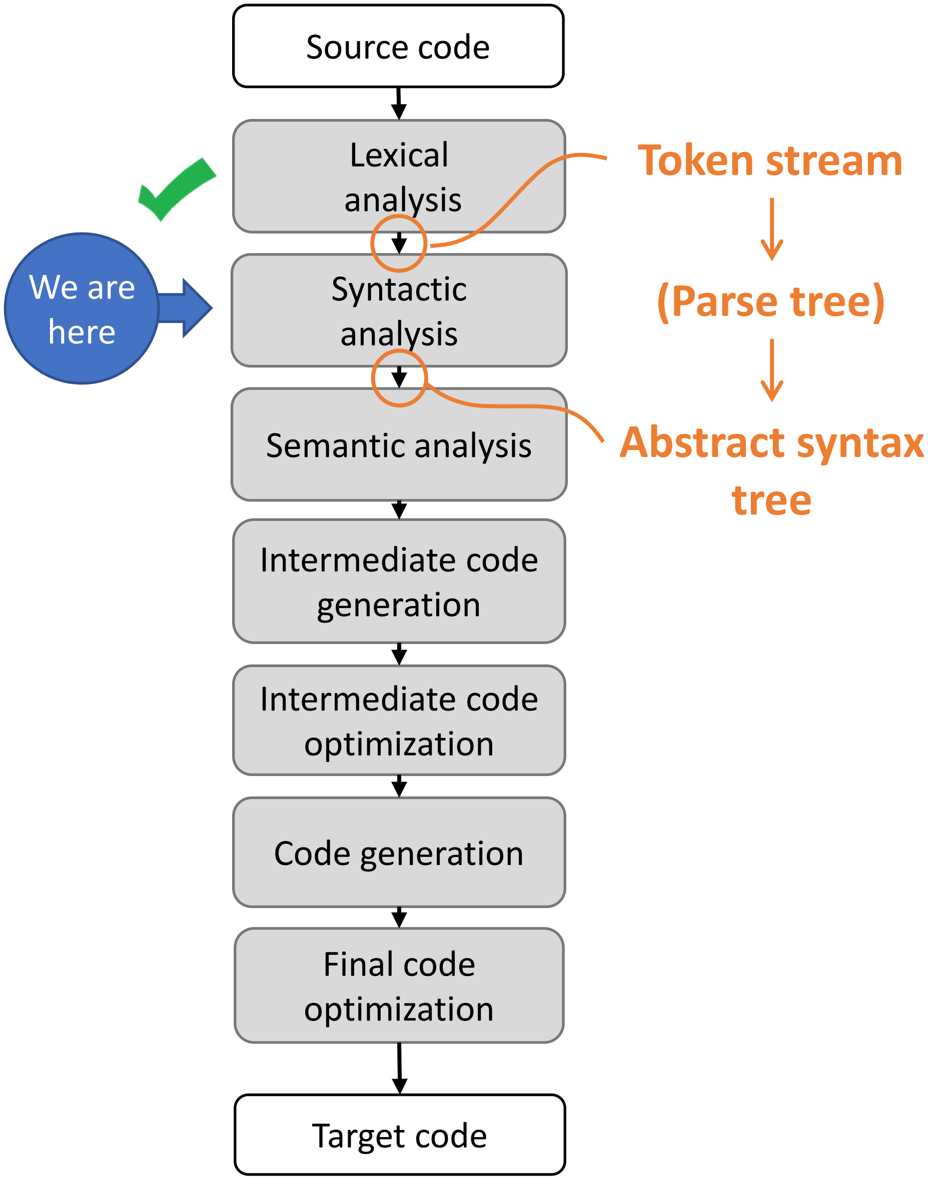
University of Kansas | Drew Davidson

EECS 665 COMPILER CONSTRUCTION

(Predictive) Parsing

Compiler Construction

Progress Pics



Syntactic Analysis

- We know ***what*** we want (induce an AST from a given token stream)

Last Time

Review - ASTs

Syntax-Directed Definition

- Showed how to use SDD to induce program's AST
- We can *specify* a correct syntactic analysis

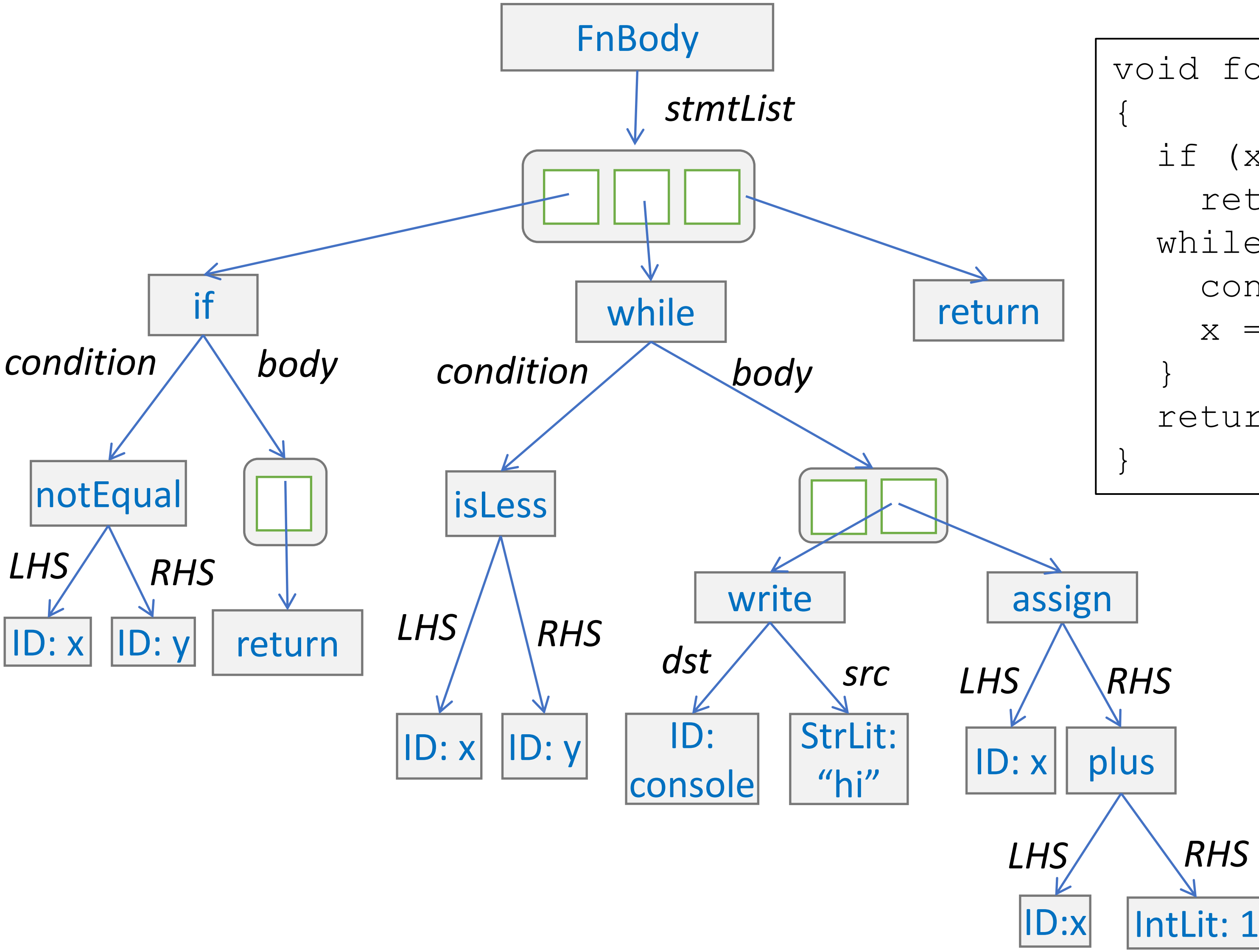
You Should Know

- The concepts of Parse Trees / ASTs
- How to specify syntax-directed definitions (to induce scalar values, data structures, etc. from parse trees)

Syntax-Directed Definition

AST For Input Code Snippet

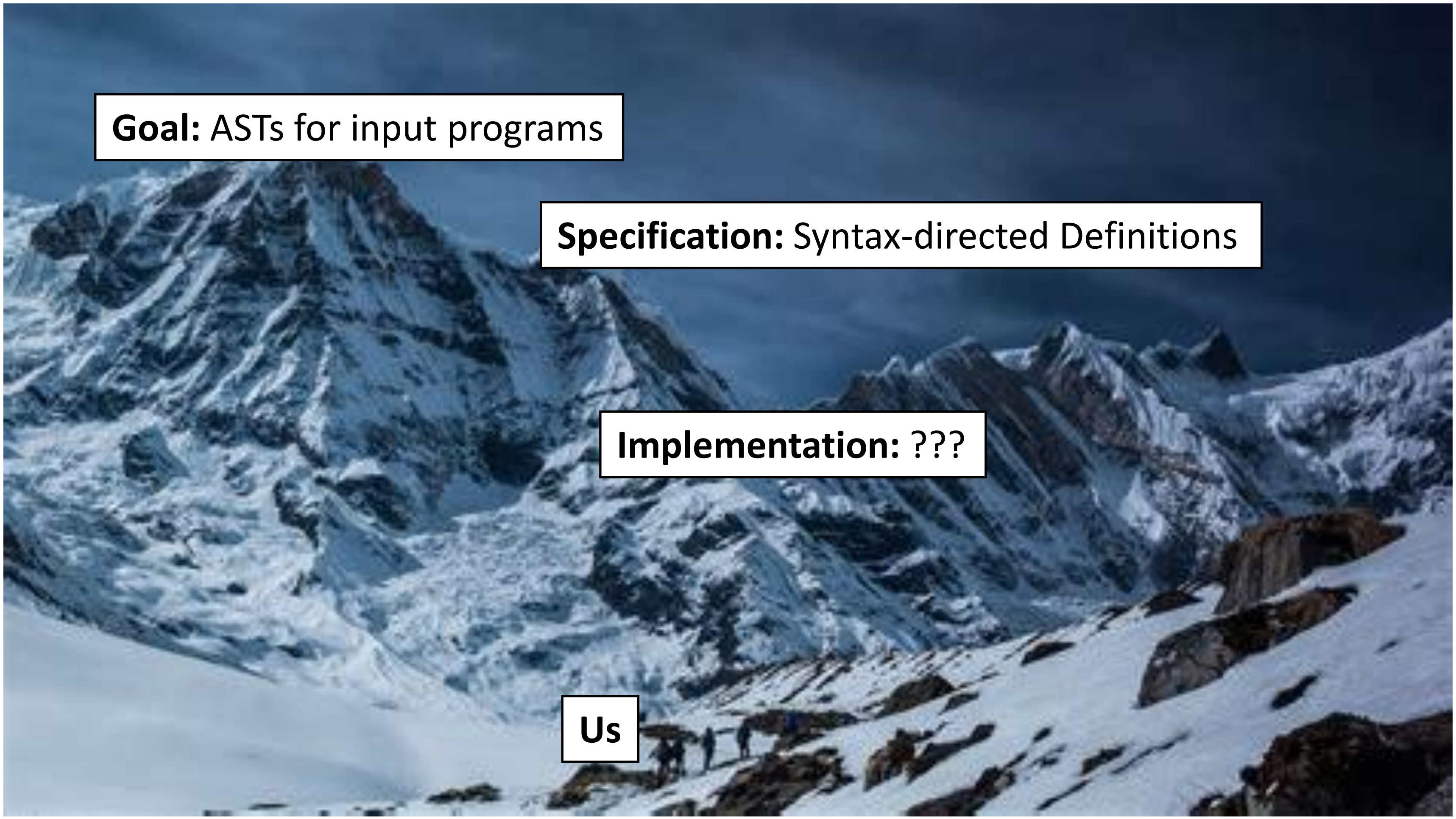
Abstract Syntax Trees – ASTs in Code




```
void foo(int x, int y)
{
  if (x != y)
    return;
  while ( x < y){
    console << "hi";
    x = (x + 1);
  }
  return;
}
```

The Goal and The Path

Abstract Syntax Trees – ASTs in Code



COMPILER


Code Generation
Execution
Runtime Environment
Intermediate Representation
Optimization
SDD
Parsing
Semantics
Lexical Analysis
Syntactic Definiton
GLOPPY
KING KANDY
CANDY CASTLE
QUEEN FROSTINE
GRAMMA NUTT
PRINCESS LOLLY
LORD LICORICE
LICORICE CASTLE
JOLLY
PLUMPY

Today's Outline

(Predictive) Parsing

Parsing

- Complexity

The LL(k) Languages

- Intro
- LL(1) parsing
- LL(1) transformations

Parsing

The Price of Expressive Power

(Predictive) Parsing – Parsing Complexity

Parsing is costly

- Jumping from Regular Languages to CF-Languages incurs penalties
  - Lose lots of language properties
  - Complexity for some operations goes up

Power always comes at a price

Parsing: Bad News/Good News

(Predictive) Parsing – Parsing Complexity

Bad News

- Complexity - CFL recognition:
  - CYK: $O(n^3)$
  - Best known bound: $O(n^{2.3728639})$
- Parsing is at least as hard

Good News

- We don't <u>need</u> to recognize all CFLs
  - We can assume ambiguity-free syntax
  - Exchange expressiveness for faster parsers

Scaling Back Expressive Power

(Predictive) Parsing – Parsing Complexity

Compilers aren't concerned with arbitrary languages

- Programming languages *should* be unambiguous
- Complex operations can be built from simple syntax

Paying for features we don't need!

"Don't buy the truck if you ain't haulin' freight"
-Drew Davidson

Restricting the Language Class

(Predictive) Parsing – Parsing Complexity

- Allows us to…
  - Build linear-time parsers
  - Detect ambiguity
- Let's consider one such class of restrictions

Disallowing Heavyweight Languages

Today's Outline

Preview Lecture 8 – Predictive Parsing

Parsing

- Complexity

The LL(k) languages

- Intro
- LL(1) parsing
- LL(1) transformations

Toyota Yaris: A sensible compact LL(1) languages are the Yaris of the Parsing world

Parser Type

LL(k) Parsers - Definitions

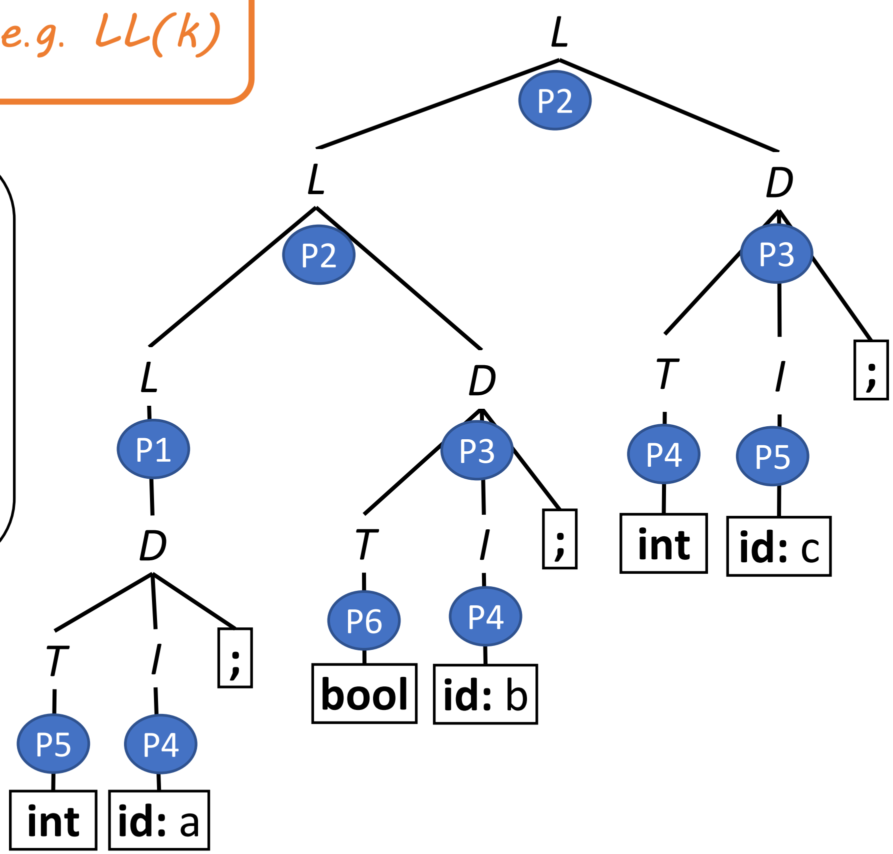
Most parsers classified by how they build the parse tree

e.g. LL(k)

Top-Down

- Start at the root
- Work downwards
- Use productions

L ::= D



Bottom-Up

- Start at the leaves
- Work upwards
- Use reductions

D ::= T I ;

Input String:

```
int a; bool b; int c;
```

Predictive Parsers

LL(k) Parsers - Definitions

- “Predict” correct production for the given nonterminal before seeing all RHS symbols
- Based on input prefix “Guess and check” the suffix

LL Grammars: Anatomy of Notation

LL Grammars - Intro

2nd L: Performs
Leftmost derivation

LL(k)

Lookahead:
Number of tokens
needed to choose
next production

1st L: Single
Left-to-right Scan
of token stream

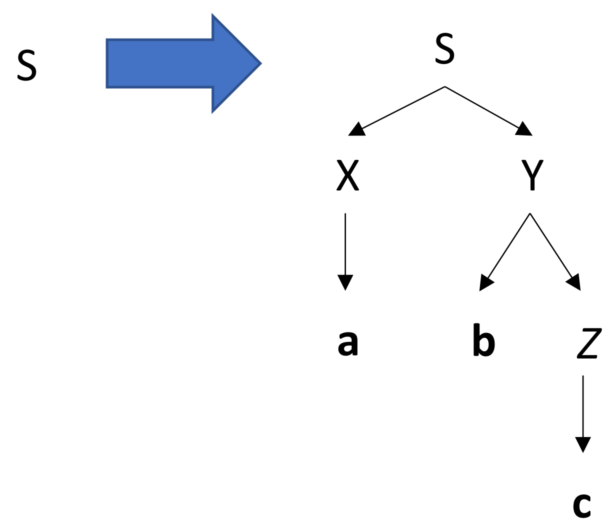
Bigger K means more
expressive languages

LL Grammar Examples

LL(0) Grammars

Takes leftmost derivation step knowing **zero** underived input tokens

S ::= *X Y*
X ::= **a**
Y ::= **b** Z
Z ::= **c**



LL Grammar Examples

LL Grammars - Intro

LL(0) Grammars

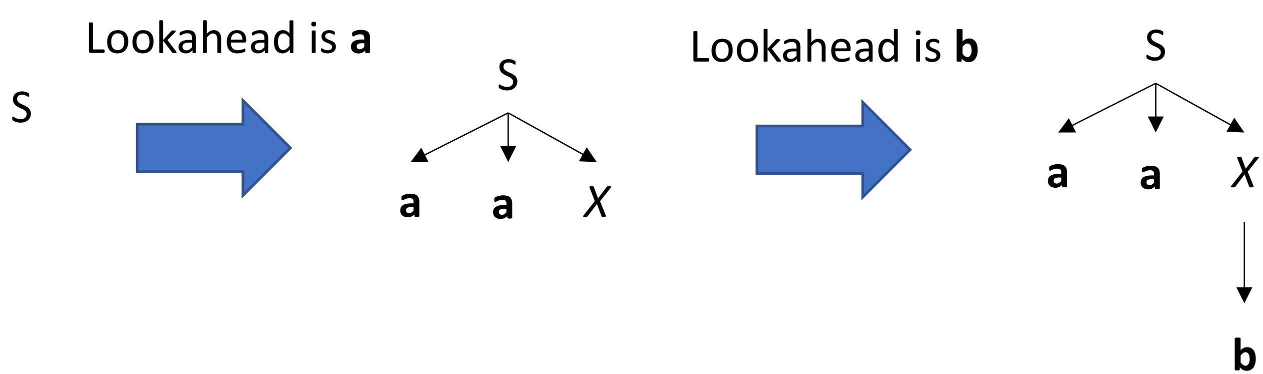
Takes leftmost derivation step knowing **zero** underived input tokens

S ::= *X Y*
X ::= **a**
Y ::= **b** Z
Z ::= **c**

LL(1) Grammars

Takes leftmost derivation step knowing **one** underived input token

S ::= **a a** *X*
| **b** *X*
X ::= **a**
| **b**



LL Grammar Examples

LL Grammars - Intro

LL(0) Grammars

Takes leftmost derivation step knowing **zero** underived input tokens

S ::= *X* *Y*
X ::= **a**
Y ::= **b** Z
Z ::= **c**

LL(1) Grammars

Takes leftmost derivation step knowing **one** underived input token

S ::= **a a** *X*
 | **b** *X*
X ::= **a**
 | b

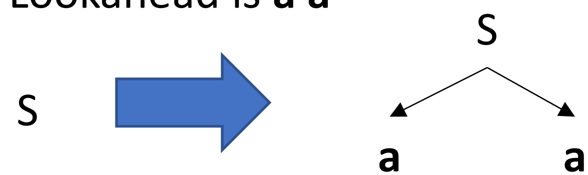
LL(2) Grammars

Takes leftmost derivation step knowing **two** underived input tokens

S ::= **a a**
 | **a b**

(Lookahead **a** is insufficient)

Lookahead is **a a**



LL Grammar Examples

LL Grammars - Intro

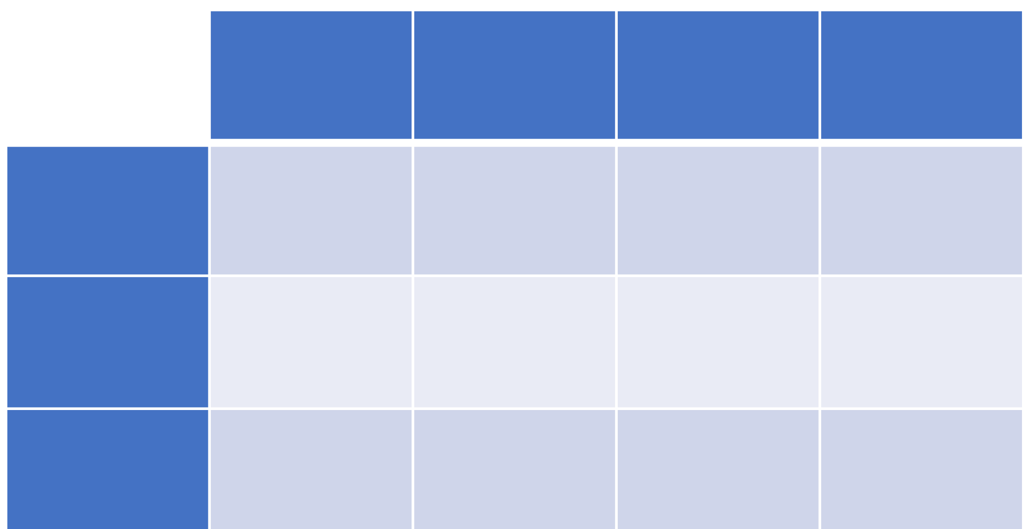
Efficient table-driven parsers

LL(1) Grammars

Takes leftmost derivation step knowing **one** underived input token

<u>Column</u>
Lookahead token

<u>Row:</u>
leftmost leaf nonterminal



Cell:
Production

LL Grammar Examples

LL Grammars - Intro

With some grammars you can
just eyeball the selector table

$$S ::= (\ S\ )$$
$$\quad |\ \{\ S\ \}$$
$$\quad |\ \varepsilon$$

Column
Lookahead token

Row:
leftmost leaf nonterminal

	(	{	)	}
S	(S)	{ S }	ε	ε

Cell:
Production

Today's Outline

(Predictive) Parsing

Parsing

- Complexity

A New Type of Language - LL

- Intro
- LL(1) parsing
- LL(1) transformations

Parsing

LL(1) Algorithm Psuedocode

Predictive Parsing - LL(1) Parsing

```
stack.push(eof)
stack.push(Start non-term)
lookahead = scanner.first_token()
Repeat
  if stack.top is a terminal
    match stack.top with lookahead
    pop y from the stack
    lookahead = scanner.next_token()
  if stack.top is a nonterminal
    X = stack.pop()
    get P = table[X,lookahead]
    push P's RHS symbols Right-to-Left
Until one of the following:
  stack is empty (accept)
  stack.top is a terminal that doesn't match t (reject)
  stack.top is a non-term and table entry is empty (reject)
```

Initialization

Case 1
Terminal on stack:
check prediction

Case 2
Non-terminal on stack:
Derive new prediction

Exit

LL(1) Example

(Predictive) Parsing - LL(1) Parsing

Example Grammar

$$S ::= (\ S\ )\ \mid\ \{\ S\ \}\ \mid\ \varepsilon$$

Initialization

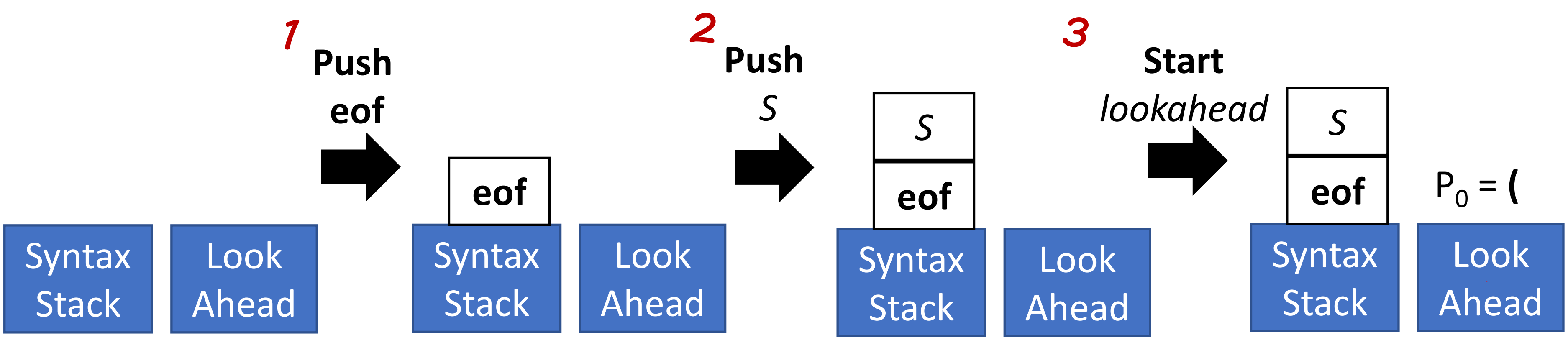
1 `stack.push(`**`eof`**`)`
2 `stack.push(`*`Start`*` non-term)`
3 `lookahead = scanner.first_token()`

Selector Table

	(	)	{	}
S	(*S*)	ε	{ *S* }	ε

Input Stream

(	{	}	)	eof
P_0	P_1	P_2	P_3	P_4



LL(1) Example

(Predictive) Parsing - LL(1) Parsing

Example Grammar

S ::= **(** S **)**
| **{** S **}**
| $\boldsymbol{\varepsilon}$

Initialization

1 `stack.push(`**`eof`**`)`
2 `stack.push(`*`Start`*` non-term)`
3 `lookahead = scanner.first_token()`

Selector Table

	(	)	{	}
S	(S)	ε	{ S }	ε

Input Stream

(	{	}	)	eof
P_0	P_1	P_2	P_3	P_4

"Predicted" Derivation Sequence

S

Syntax Stack | Look Ahead

Becomes

S
eof
Syntax Stack

P_0 = **(**
Look Ahead

LL(1) Example

(Predictive) Parsing - LL(1) Parsing

<u>Example Grammar</u>

$$S ::= (\ S\ )\ |\ \{\ S\ \}\ |\ \varepsilon$$

Nonterminal on stack:

1 `X = pop(stack)`
2 `get P = table[X,lookahead]`
3 `push P's symbols` *`Right-to-Left`*

<u>Selector Table</u>

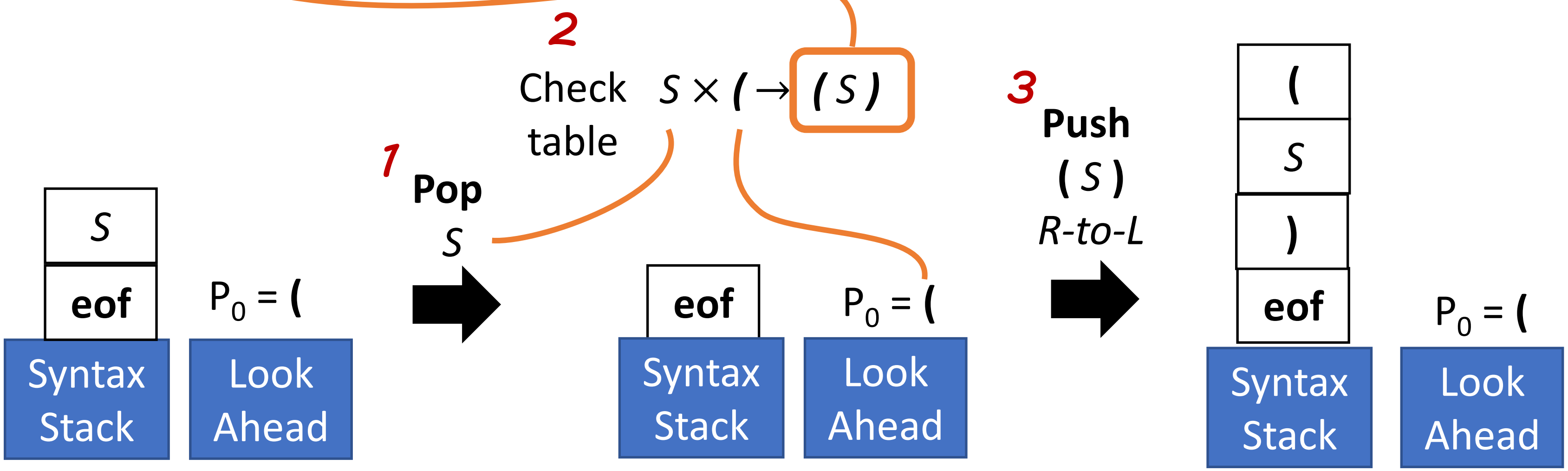
	(	)	{	}
S	(*S*)	ε	{ *S* }	ε

<u>Input Stream</u>

(	{	}	)	eof
P_0	P_1	P_2	P_3	P_4

<u>“Predicted” Derivation Sequence</u>

S



LL(1) Example

(Predictive) Parsing - LL(1) Parsing

Example Grammar

$S ::= (\ S\ )$
$\quad | \ \{\ S\ \}$
$\quad | \ \varepsilon$

Nonterminal on stack:

1 `X = pop(stack)`
2 `get P = table[X,lookahead]`
3 `push P's symbols` *`Right-to-Left`*

Selector Table

	(	)	{	}
S	(*S*)	ε	{ *S* }	ε

Input Stream

(	{	}	)	eof
P_0	P_1	P_2	P_3	P_4

"Predicted" Derivation Sequence

$S \Rightarrow (\ S\ )$

Syntax Stack	Look Ahead
S	
eof	P_0 = (

Becomes

Syntax Stack	Look Ahead
(	
S	
)	
eof	P_0 = (

LL(1) Example

(Predictive) Parsing - LL(1) Parsing

Example Grammar

S ::= **(** S **)**
| **{** S **}**
| $\boldsymbol{\varepsilon}$

Terminal on stack:

```
1 match stack.top with lookahead
2 pop stack
3 lookahead = scanner.next_token()
```

Selector Table

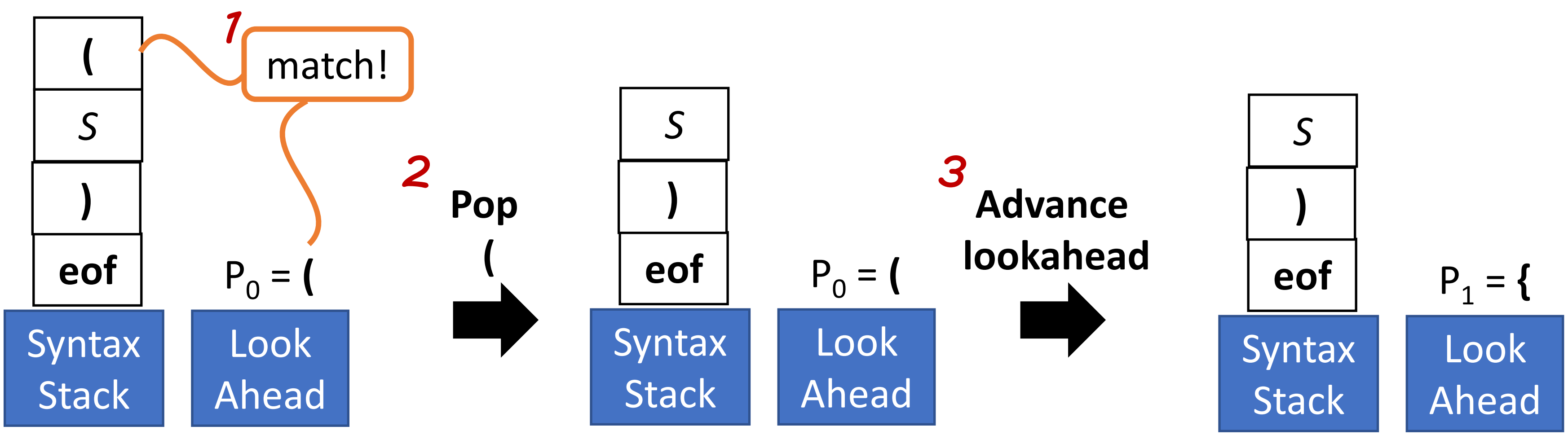
	(	)	{	}
S	**(** S **)**	**ε**	**{** S **}**	**ε**

Input Stream

(	{	}	)	eof
P_0	P_1	P_2	P_3	P_4

"Predicted" Derivation Sequence

S ⇒ **(** S **)**



LL(1) Example

(Predictive) Parsing - LL(1) Parsing

Example Grammar

$$
\begin{aligned}
S &::= \mathbf{(}\ S\ \mathbf{)} \\
  &\mid \mathbf{\{}\ S\ \mathbf{\}} \\
  &\mid \varepsilon
\end{aligned}
$$

Terminal on stack:

1. `match stack.top with lookahead`
2. `pop stack`
3. `lookahead = scanner.next_token()`

Selector Table

	(	)	{	}
S	**(** *S* **)**	**ε**	**{** *S* **}**	**ε**

Input Stream

(	{	}	)	eof
P_0 ✔	P_1	P_2	P_3	P_4

"Predicted" Derivation Sequence

$S \Rightarrow (\ S\ )$

Syntax Stack:

(
S
)
eof

Look Ahead: P_0 = **(**

Becomes

Syntax Stack:

S
)
eof

Look Ahead: P_1 = **{**

LL(1) Example

(Predictive) Parsing - LL(1) Parsing

Example Grammar

$$S ::= (\ S\ ) \mid \{\ S\ \} \mid \varepsilon$$

Nonterminal on stack:

1 `X = pop(stack)`
2 `get P = table[X,lookahead]`
3 `push P's RHS symbols` *`Right-to-Left`*

Selector Table

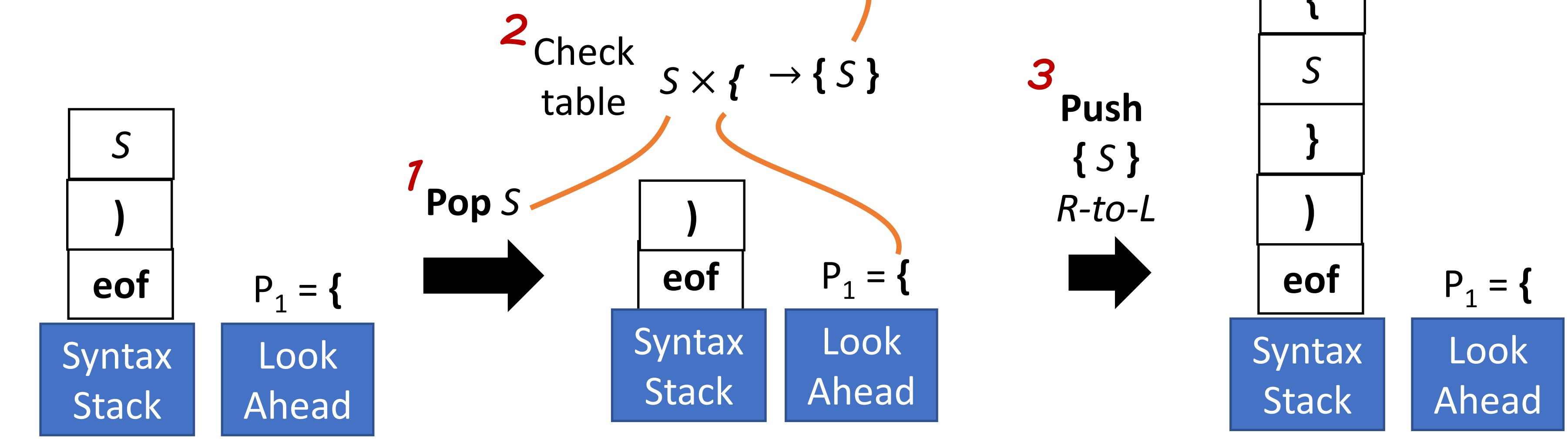
	(	)	{	}
S	(*S*)	ε	$\boxed{\{\ S\ \}}$	ε

Input Stream

(	{	}	)	eof
P_0 ✔	P_1	P_2	P_3	P_4

"Predicted" Derivation Sequence

$S \Rightarrow (\ S\ )$



LL(1) Example

(Predictive) Parsing - LL(1) Parsing

Example Grammar

$$S ::= (\ S\ )\ \mid\ \{\ S\ \}\ \mid\ \varepsilon$$

Nonterminal on stack:

```
1 X = pop(stack)
2 get P = table[X,lookahead]
3 push P's RHS symbols Right-to-Left
```

Selector Table

	(	)	{	}
S	(*S*)	ε	{ *S* }	ε

Input Stream

(	{	}	)	eof
P_0 ✔	P_1	P_2	P_3	P_4

"Predicted" Derivation Sequence

S ⇒ (S)
⇒ ({ S })

Syntax Stack	Look Ahead
S	
)	
eof	P_1 = {

Becomes

Syntax Stack	Look Ahead
{	
S	
}	
)	
eof	P_1 = {

LL(1) Example

(Predictive) Parsing - LL(1) Parsing

<u>Example Grammar</u>

S ::= **(** S **)**
  | **{** S **}**
  | ε

Terminal on stack:

```
1 match stack.top with lookahead
2 pop stack
3 lookahead = scanner.next_token()
```

<u>Selector Table</u>

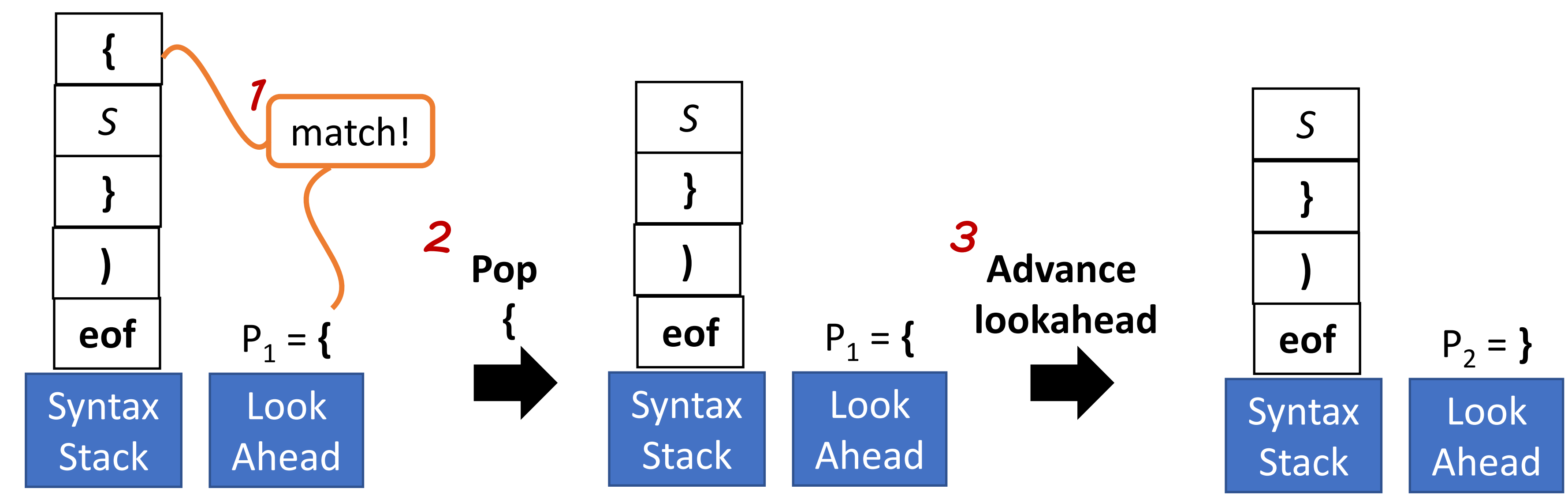
	(	)	{	}
S	(S)	ε	{ S }	ε

<u>Input Stream</u>

(	{	}	)	eof
P_0 ✔	P_1	P_2	P_3	P_4

<u>"Predicted"</u>
<u>Derivation Sequence</u>

$S \Rightarrow (S)$
$\Rightarrow (\{ S \})$



Example Grammar

$$S ::= (S) \mid \{ S \} \mid \varepsilon$$

LL(1) Example

(Predictive) Parsing - LL(1) Parsin)

Terminal on stack:

1. `match stack.top with lookahead`
2. `pop stack`
3. `lookahead = scanner.next_token()`

Selector Table

	(	)	{	}
S	(S)	ε	{ S }	ε

Input Stream

(	{	}	)	eof
P_0 ✓	P_1 ✓	P_2	P_3	P_4

"Predicted" Derivation Sequence

$$S \Rightarrow (S) \Rightarrow (\{ S \})$$

Syntax Stack
{
S
}
)
eof

Look Ahead: P_1 = {

Becomes

Syntax Stack
S
}
)
eof

Look Ahead: P_2 = }

LL(1) Example

(Predictive) Parsing - LL(1) Parsin)

<u>Example Grammar</u>

S ::= **(** S **)**
  | **{** S **}**
  | $\boldsymbol{\varepsilon}$

Nonterminal on stack:

1 `X = pop(stack)`
2 `get P = table[X,lookahead]`
3 `push P's RHS symbols` *`Right-to-Left`*

<u>Selector Table</u>

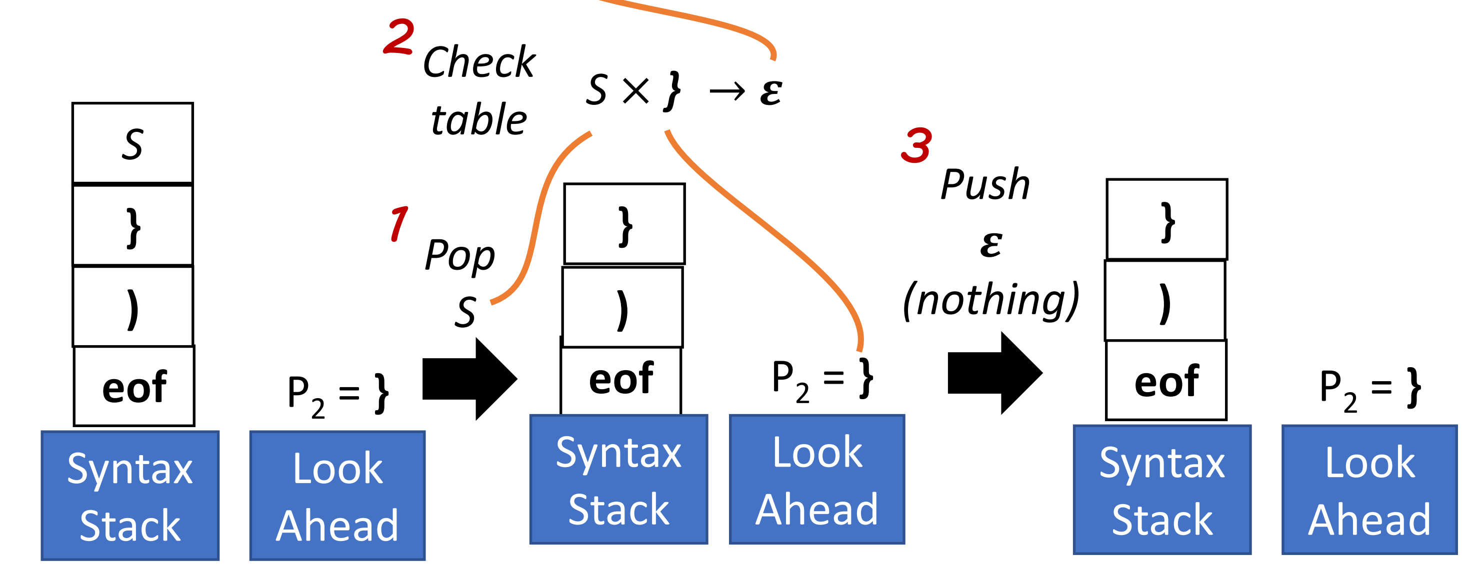
	(	)	{	}
S	(S)	ε	{ S }	ε

<u>Input Stream</u>

(	{	}	)	eof
P_0	P_1	P_2	P_3	P_4
✔	✔			

<u>“Predicted”</u>
<u>Derivation Sequence</u>

S $\Rightarrow$ (S)
$\Rightarrow$ ({ S })



LL(1) Example

(Predictive) Parsing - LL(1) Parsing

Example Grammar

S ::= **(** S **)**
   | **{** S **}**
   | **ε**

Nonterminal on stack:

```
1 X = pop(stack)
2 get P = table[X,lookahead]
3 push P's RHS symbols Right-to-Left
```

Selector Table

	(	)	{	}
S	(S)	ε	{ S }	ε

Input Stream

(	{	}	)	eof
P_0	P_1	P_2	P_3	P_4
✔	✔			

"Predicted" Derivation Sequence

S ⇒ (S)
⇒ ({ S })
⇒ ({ })

Syntax Stack
S
}
)
eof

Look Ahead: P_2 = }

Becomes

Syntax Stack
}
)
eof

Look Ahead: P_2 = }

Example Grammar

$$S ::= (\ S\ )\ |\ \{\ S\ \}\ |\ \varepsilon$$

LL(1) Example

(Predictive) Parsing - LL(1) Parsing

Terminal on stack:

1 `match stack.top with lookahead`
2 `pop stack`
3 `lookahead = scanner.next_token()`

Selector Table

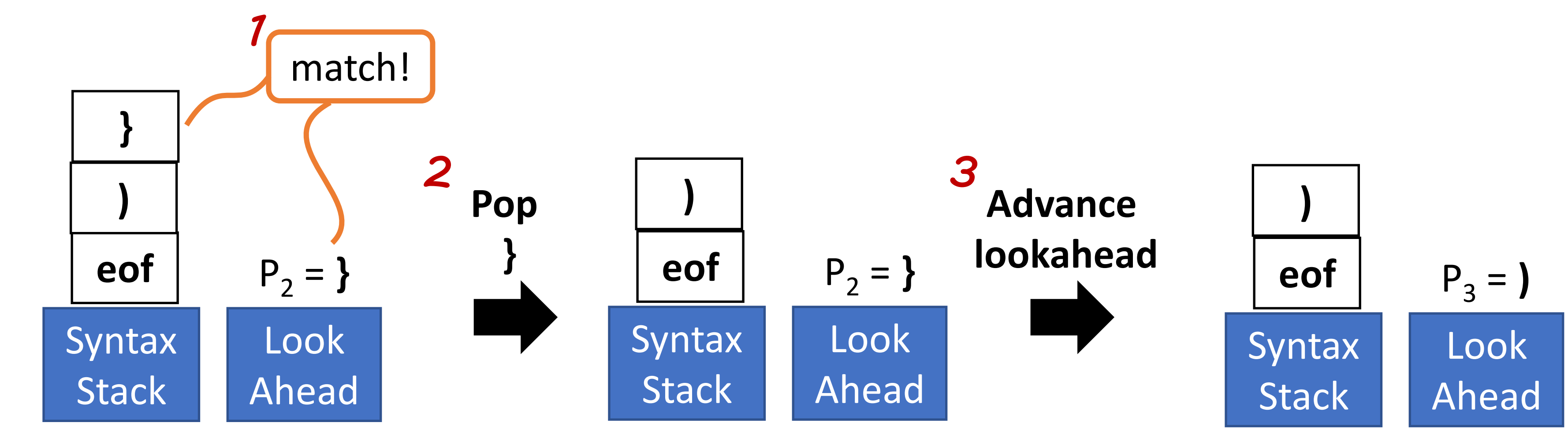
	(	)	{	}
S	(S)	ε	{ S }	ε

Input Stream

(	{	}	)	eof
P_0	P_1	P_2	P_3	P_4
✔	✔	✔		

"Predicted" Derivation Sequence

$$S \Rightarrow (\ S\ ) \Rightarrow (\ \{\ S\ \}\ ) \Rightarrow (\ \{\ \}\ )$$



LL(1) Example

(Predictive) Parsing - LL(1) Parsing

Example Grammar

$$S ::= (\ S\ ) \;|\; \{\ S\ \} \;|\; \varepsilon$$

Terminal on stack:

1. `match stack.top with lookahead`
2. `pop stack`
3. `lookahead = scanner.next_token()`

Selector Table

	(	)	{	}
S	(*S*)	ε	{ *S* }	ε

Input Stream

(	{	}	)	eof
P_0	P_1	P_2	P_3	P_4
✓	✓	✓		

"Predicted" Derivation Sequence

$$S \Rightarrow (\ S\ ) \Rightarrow (\ \{\ S\ \}\ ) \Rightarrow (\ \{\ \}\ )$$

Syntax Stack	Look Ahead
)	
eof	P_3 =)

LL(1) Example

(Predictive) Parsing - LL(1) Parsing

<u>Example Grammar</u>

$S ::= (\ S\ )$
$\quad |\ \{\ S\ \}$
$\quad |\ \varepsilon$

Terminal on stack:

```
1 match stack.top with lookahead
2 pop stack
3 lookahead = scanner.next_token()
```

<u>Selector Table</u>

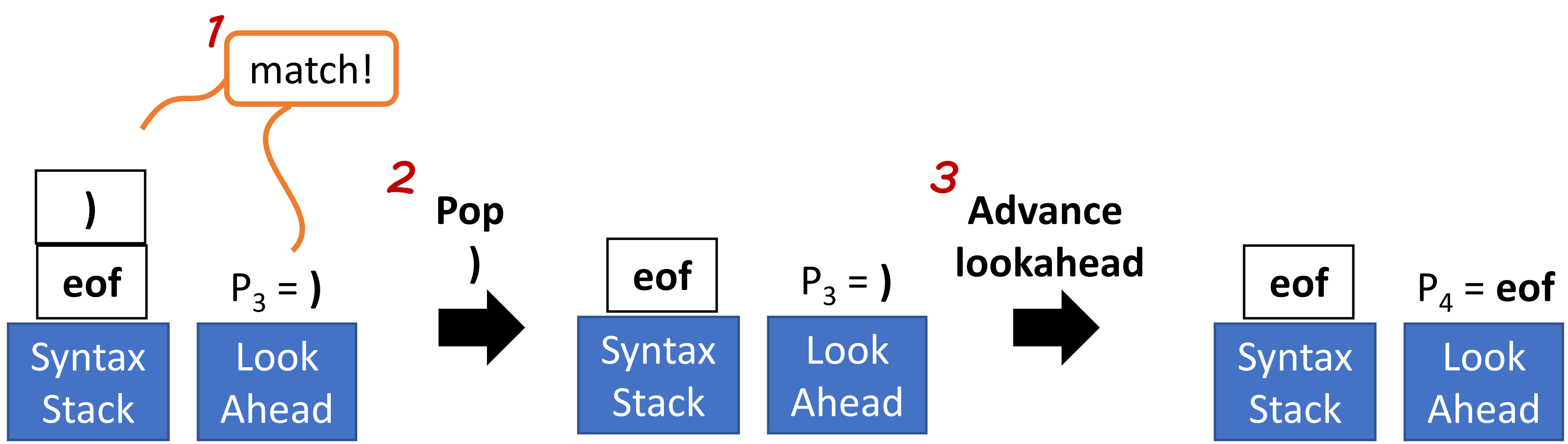
	(	)	{	}
S	(S)	ε	{ S }	ε

<u>Input Stream</u>

(	{	}	)	eof
P_0	P_1	P_2	P_3	P_4
✔	✔	✔		

<u>"Predicted" Derivation Sequence</u>

$S \Rightarrow (\ S\ )$
$\quad \Rightarrow (\ \{\ S\ \}\ )$
$\quad \Rightarrow (\ \{\ \}\ )$



Example Grammar

$S ::= (\ S\ )$
$\quad |\ \{\ S\ \}$
$\quad |\ \varepsilon$

LL(1) Example

(Predictive) Parsing - LL(1) Parsing

Terminal on stack:

1 `match stack.top with lookahead`
2 `pop stack`
3 `lookahead = scanner.next_token()`

Selector Table

	(	)	{	}
S	(S)	ε	{ S }	ε

Input Stream

(	{	}	)	**eof**
P_0	P_1	P_2	P_3	P_4
✔	✔	✔	✔	

"Predicted" Derivation Sequence

$S \Rightarrow (\ S\ )$
$\Rightarrow (\ \{\ S\ \}\ )$
$\Rightarrow (\ \{\ \}\ )$

eof

Syntax Stack

P_4 = **eof**

Look Ahead

Example Grammar

$$S ::= (\ S\ )\ \mid\ \{\ S\ \}\ \mid\ \varepsilon$$

LL(1) Example

(Predictive) Parsing - LL(1) Parsing

Terminal on stack:

1 `match stack.top with lookahead`
2 `pop stack`
3 `lookahead = scanner.next_token()`

Selector Table

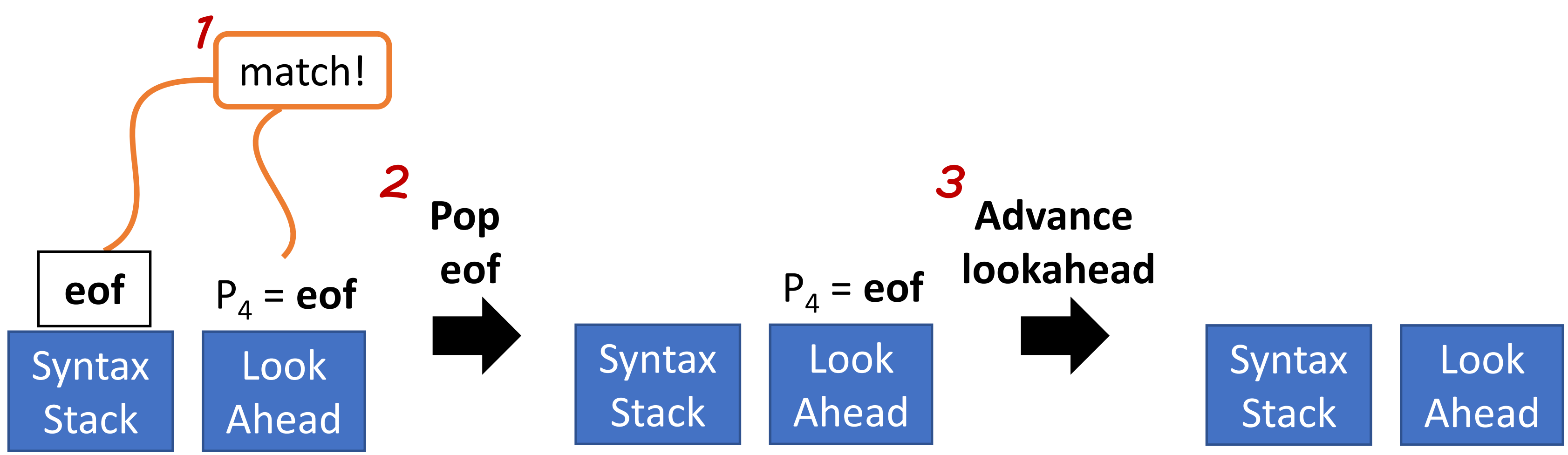
	(	)	{	}
S	(*S*)	ε	{ *S* }	ε

Input Stream

(	{	}	)	eof
P_0	P_1	P_2	P_3	P_4
✔	✔	✔	✔	✔

"Predicted" Derivation Sequence

$$S \Rightarrow (\ S\ ) \Rightarrow (\ \{\ S\ \}\ ) \Rightarrow (\ \{\ \}\ )$$



LL(1) Example

(Predictive) Parsing - LL(1) Parsing

Example Grammar

$$S ::= (\ S\ )\ |\ \{\ S\ \}\ |\ \varepsilon$$

Terminal on stack:

```
1 match stack.top with lookahead
2 pop stack
3 lookahead = scanner.next_token()
```

Selector Table

	(	)	{	}
S	(*S*)	ε	{ *S* }	ε

Input Stream

(	{	}	)	eof
P_0	P_1	P_2	P_3	P_4
✔	✔	✔	✔	✔

"Predicted" Derivation Sequence

$$S \Rightarrow (\ S\ ) \Rightarrow (\ \{\ S\ \}\ ) \Rightarrow (\ \{\ \}\ )$$

Syntax Stack

Look Ahead

LL(1) Example

(Predictive) Parsing - LL(1) Parsing

<u>Example Grammar</u>

S ::= **(** S **)**
  | **{** S **}**
  | $\boldsymbol{\varepsilon}$

Exit

```
Stack and Lookahead both empty:
   Accept!
```

<u>Selector Table</u>

	(	)	{	}
S	(S)	ε	{ S }	ε

<u>Input Stream</u>

(	{	}	)	eof
P_0	P_1	P_2	P_3	P_4
⇒({})	⇒({})	⇒({})	⇒({})	⇒({})

<u>"Predicted" Derivation Sequence</u>

$S \Rightarrow (S)$
$\Rightarrow (\{ S \})$
$\Rightarrow (\{ \ \})$

Syntax Stack

Look Ahead

Running LL(1)

(Predictive) Parsing - LL(1) Parsing

Linear parse!

- Once we verify a position we never look at it again

Pretty fast given general parsing complexity

- Especially once you start combining steps

Let’s do another quick example

- This time we’ll show rejecting bad input

LL(1) Reject Example

(Predictive) Parsing - LL(1) Parsing

Example Grammar

S ::= **(** S **)**
  | **{** S **}**
  | ε

Selector Table

	(	)	{	}
S	(S)	ε	{ S }	ε

Input Stream

(	}	eof
P_0	P_1	P_2

"Predicted" Derivation Sequence

S ⇒ (S)

Initialization
1,2,3

Syntax Stack
S
eof

Look Ahead: P_0 = **(**

Nonterminal Actions
1,2,3

Syntax Stack
(
S
)
eof

Look Ahead: P_0 = **(**

Terminal Actions
1,2,3

LL(1) Reject Example

(Predictive) Parsing - LL(1) Parsing

Example Grammar

S ::= **(** S **)**
  | **{** S **}**
  | ε

Selector Table

	(	)	{	}
S	(*S*)	ε	{ *S* }	ε

Input Stream

(	}	eof
P_0	P_1	P_2

"Predicted" Derivation Sequence

S $\Rightarrow$ **(** S **)**

Initialization
1,2,3

Syntax Stack: *S*, **eof**
Look Ahead: P_0 = **(**

Nonterminal Actions
1,2,3

Syntax Stack: **(**, *S*, **)**, **eof**
Look Ahead: P_0 = **(**

Terminal Actions
1,2,3

Syntax Stack: *S*, **)**, **eof**
Look Ahead: P_1 = **}**

LL(1) Reject Example

(Predictive) Parsing - LL(1) Parsing

Example Grammar

S ::= **(** S **)**
 | **{** S **}**
 | **ε**

Selector Table

	(	)	{	}
S	**(** *S* **)**	**ε**	**{** *S* **}**	**ε**

Input Stream

(	**}**	**eof**
P_0	P_1	P_2

"Predicted" Derivation Sequence

S ⇒ **(** S **)**

Syntax Stack	Look Ahead
S) **eof**	P_1 = **}**

Nonterminal Actions
1,2,3

Syntax Stack	Look Ahead
) **eof**	P_1 = **}**

Terminal Actions
1 – Fails!

1
mismatch!

Syntax Stack	Look Ahead
) **eof**	P_1 = **}**

This Grammar is Great!

(Predictive) Parsing - LL(1) Parsing

With 1 token we know exactly what production it starts

The Language may be LL(1) Even when the Grammar is not LL(1)

…as long as the grammar is LL(1)

Consider a Concerning Grammar…

(Predictive) Parsing - LL(1) Parsing

Our prior LL(2) Grammar		Equivalent LL(1) Grammar
S ::= **a a** | **a b**	⇨	S ::= **a** X X ::= **a** | **b**

The Language may be LL(1) Even when the Grammar is not LL(1)

S, a

S, a b