Grammar and Automata Correspondence

Grammar	Automaton
Regular Grammar	Finite-State Automaton
Context-Free Grammar	Push-Down Automaton
Context-Sensitive Grammar	Turing Machine

CFG Versus PDA

- CFGs and PDAs are of equivalent power
- Grammar Implementation Mechanism:
  - Translate CFG to PDA, then use PDA to parse input string
  - Foundation for bottom-up parser generators

Pushdown Automata

- Consists of
  - Pushdown stack (can have terminals and nonterminals)
  - Finite state automaton control
- Can do one of three actions (based on state and input):
  - Shift:
    - Shift current input symbol from input onto stack
  - Reduce:
    - If symbols on top of stack match right hand side of some grammar production NT $\rightarrow \beta$
    - Pop symbols (β) off of the stack
    - Push left hand side nonterminal (NT) onto stack
  - Accept the input string

Shift-Reduce Parser Example

Stack

Expr → *Expr Op Expr*

Expr → (*Expr*)

Expr → - *Expr*

Expr → num

Op → +

Op → -

Op → *

Input String

num	*	(	num	+	num	)

Shift-Reduce Parser Example

Expr → *Expr Op Expr*

Expr → (*Expr*)

Expr → - *Expr*

Expr → num

Op → +

Op → -

Op → *

num	*	(	num	+	num	)

Shift-Reduce Parser Example

Expr → *Expr Op Expr*

Expr → (*Expr*)

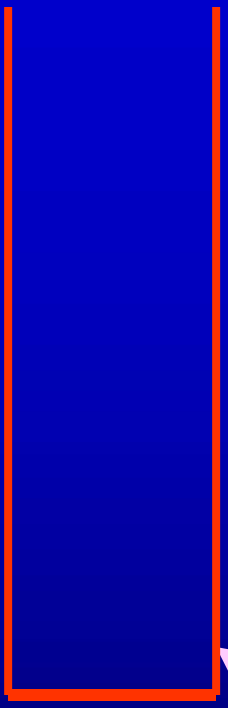
Expr → - *Expr*

Expr → num

Op → +

Op → -

Op → *



SHIFT

num	*	(	num	+	num	)

Shift-Reduce Parser Example

Expr → *Expr Op Expr*

Expr → (*Expr*)

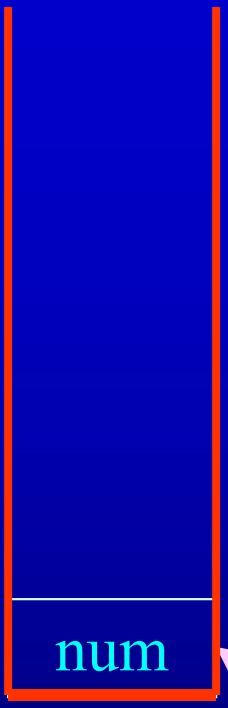
Expr → - *Expr*

Expr → num

Op → +

Op → -

Op → *



SHIFT

*	(	num	+	num	)

Shift-Reduce Parser Example

REDUCE

Expr → *Expr Op Expr*

Expr → (*Expr*)

Expr → - *Expr*

Expr → num

Op → +

Op → -

Op → *

*	(	num	+	num	)

Shift-Reduce Parser Example

Expr → *Expr Op Expr*

Expr → (*Expr*)

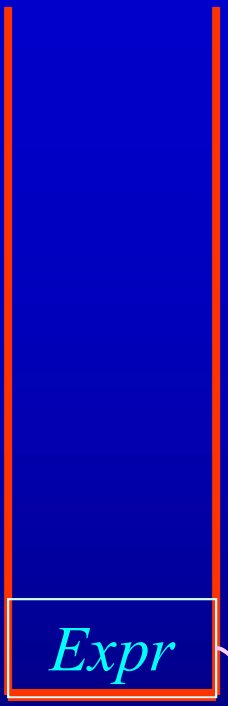
Expr → - *Expr*

Expr → num

Op → +

Op → -

Op → *



REDUCE

num

*	(	num	+	num	)

Shift-Reduce Parser Example

Expr → *Expr Op Expr*

Expr → (*Expr*)

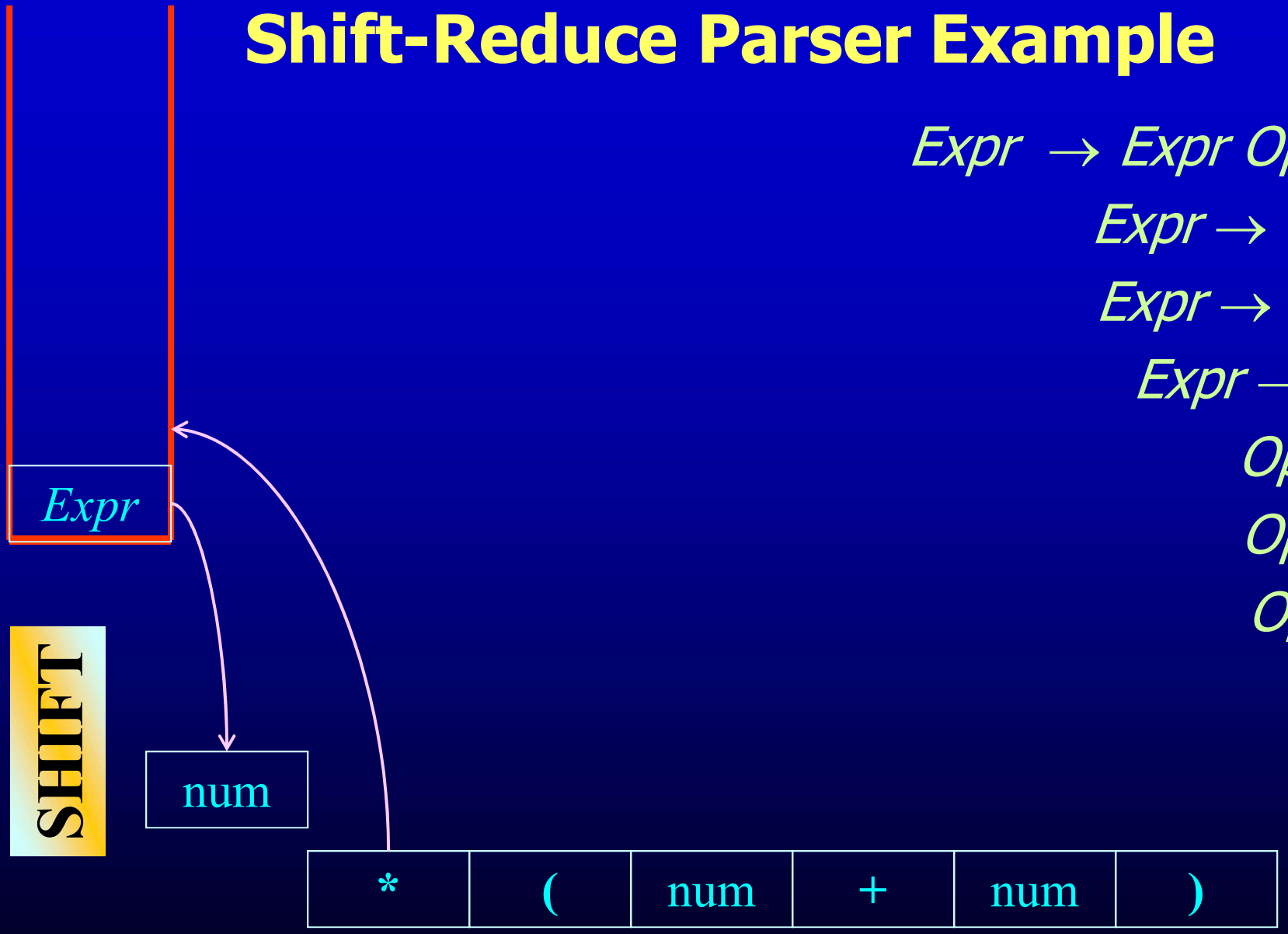
Expr → - *Expr*

Expr → num

Op → +

Op → -

Op → *



Shift-Reduce Parser Example

Expr → *Expr Op Expr*

Expr → (*Expr*)

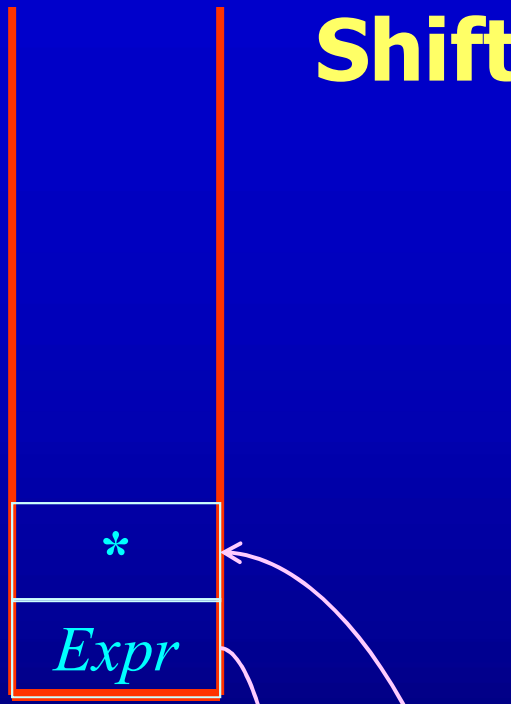
Expr → - *Expr*

Expr → num

Op → +

Op → -

Op → *



SHIFT

num

(	num	+	num	)

Shift-Reduce Parser Example

Expr → *Expr Op Expr*

Expr → (*Expr*)

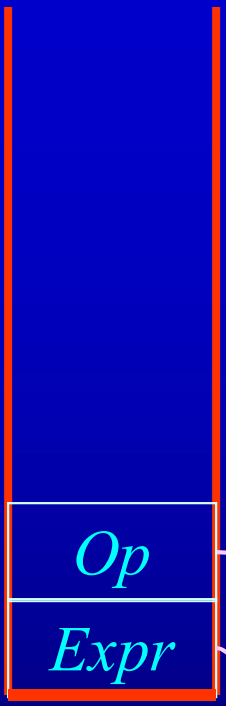
Expr → - *Expr*

Expr → num

Op → +

Op → -

Op → *



REDUCE

num	*

(	num	+	num	)

Shift-Reduce Parser Example

Expr → *Expr Op Expr*

Expr → (*Expr*)

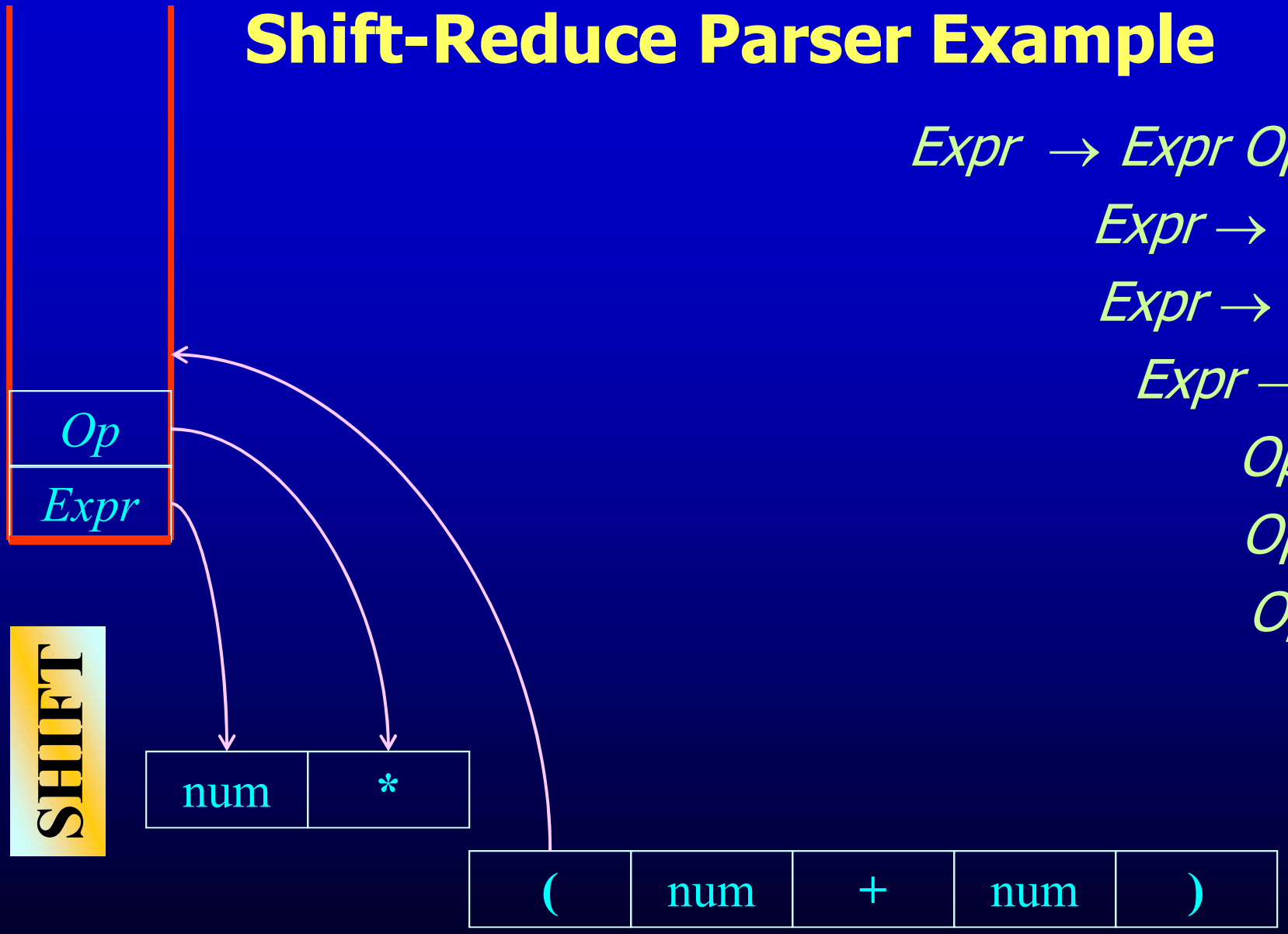
Expr → - *Expr*

Expr → num

Op → +

Op → -

Op → *



Shift-Reduce Parser Example

Expr → *Expr Op Expr*

Expr → (*Expr*)

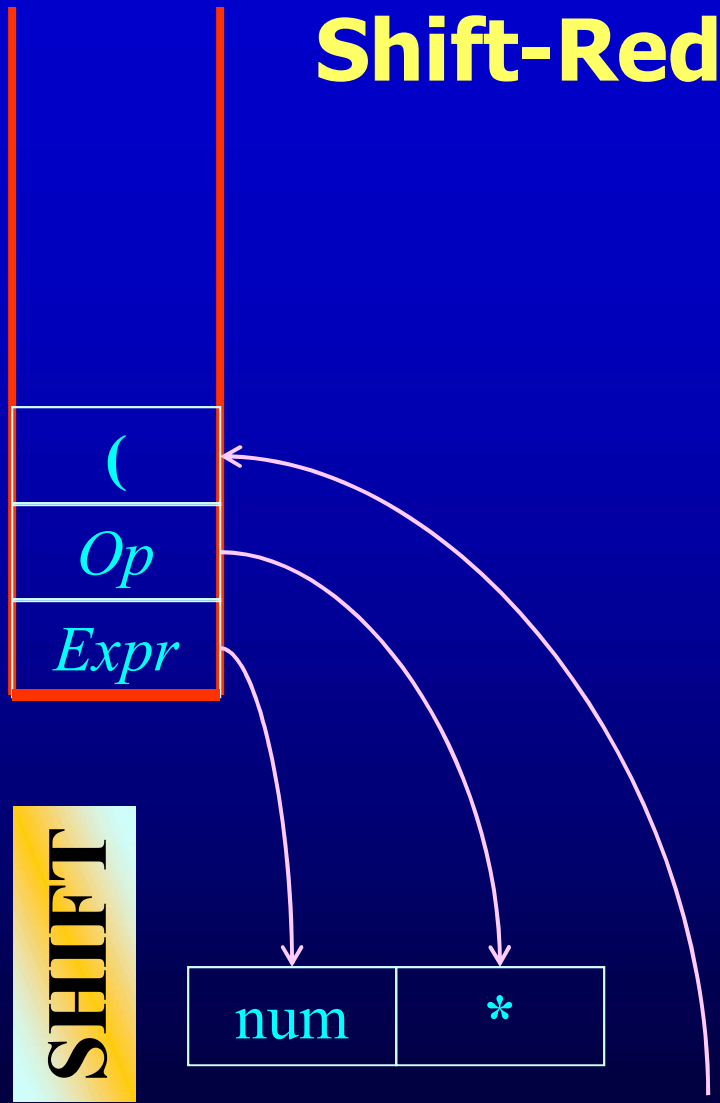
Expr → - *Expr*

Expr → num

Op → +

Op → -

Op → *



num	+	num	)

Shift-Reduce Parser Example

Expr → *Expr Op Expr*

Expr → (*Expr*)

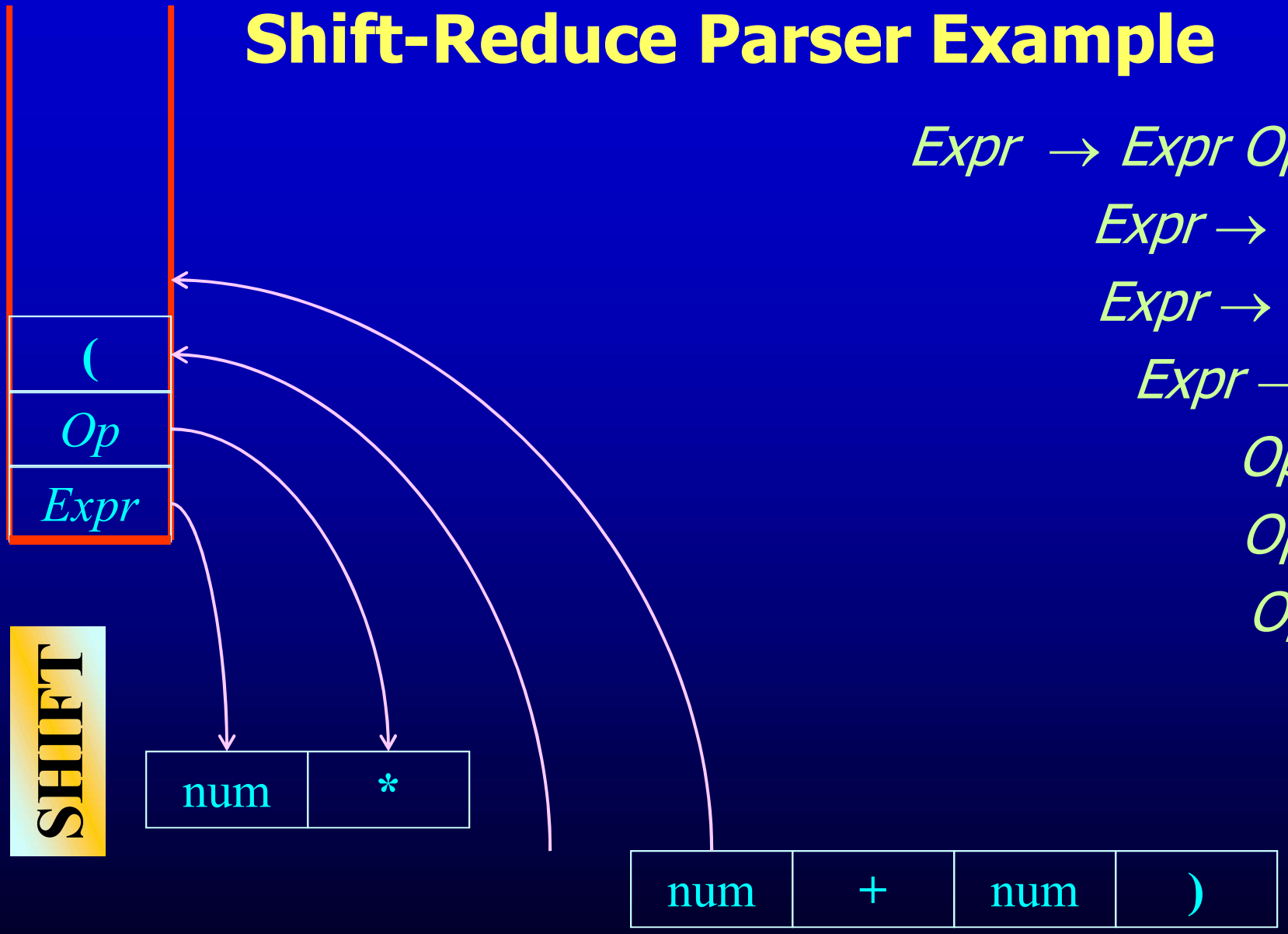
Expr → - *Expr*

Expr → num

Op → +

Op → -

Op → *



Shift-Reduce Parser Example

Expr → *Expr Op Expr*

Expr → (*Expr*)

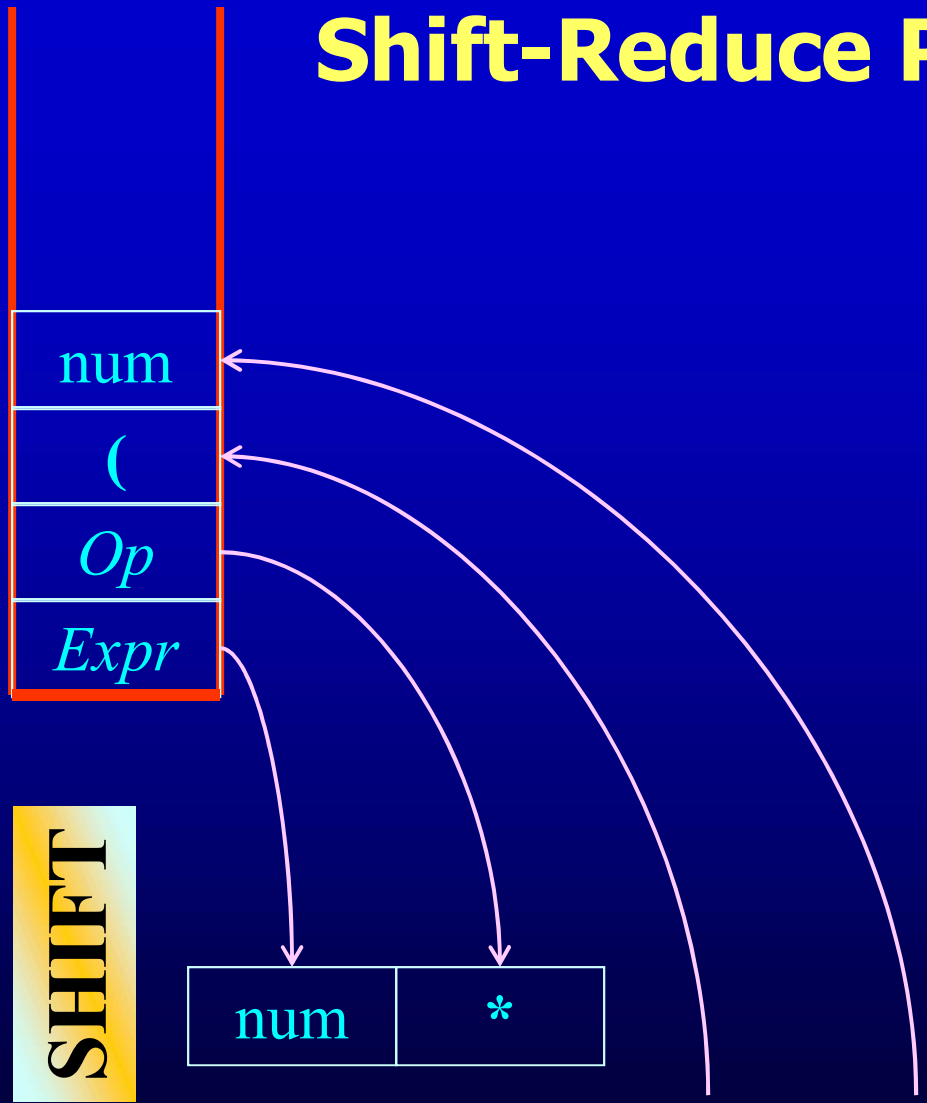
Expr → - *Expr*

Expr → num

Op → +

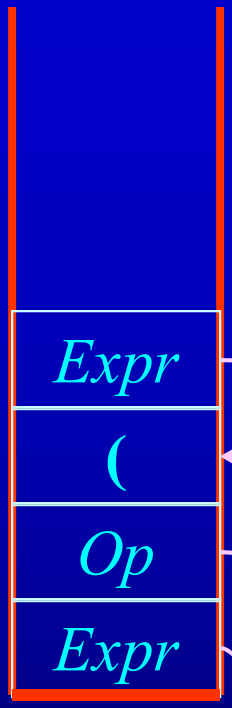
Op → -

Op → *



+	num	)

Shift-Reduce Parser Example



REDUCE

num | * | num

+ | num |)

Expr → *Expr Op Expr*

Expr → (*Expr*)

Expr → - *Expr*

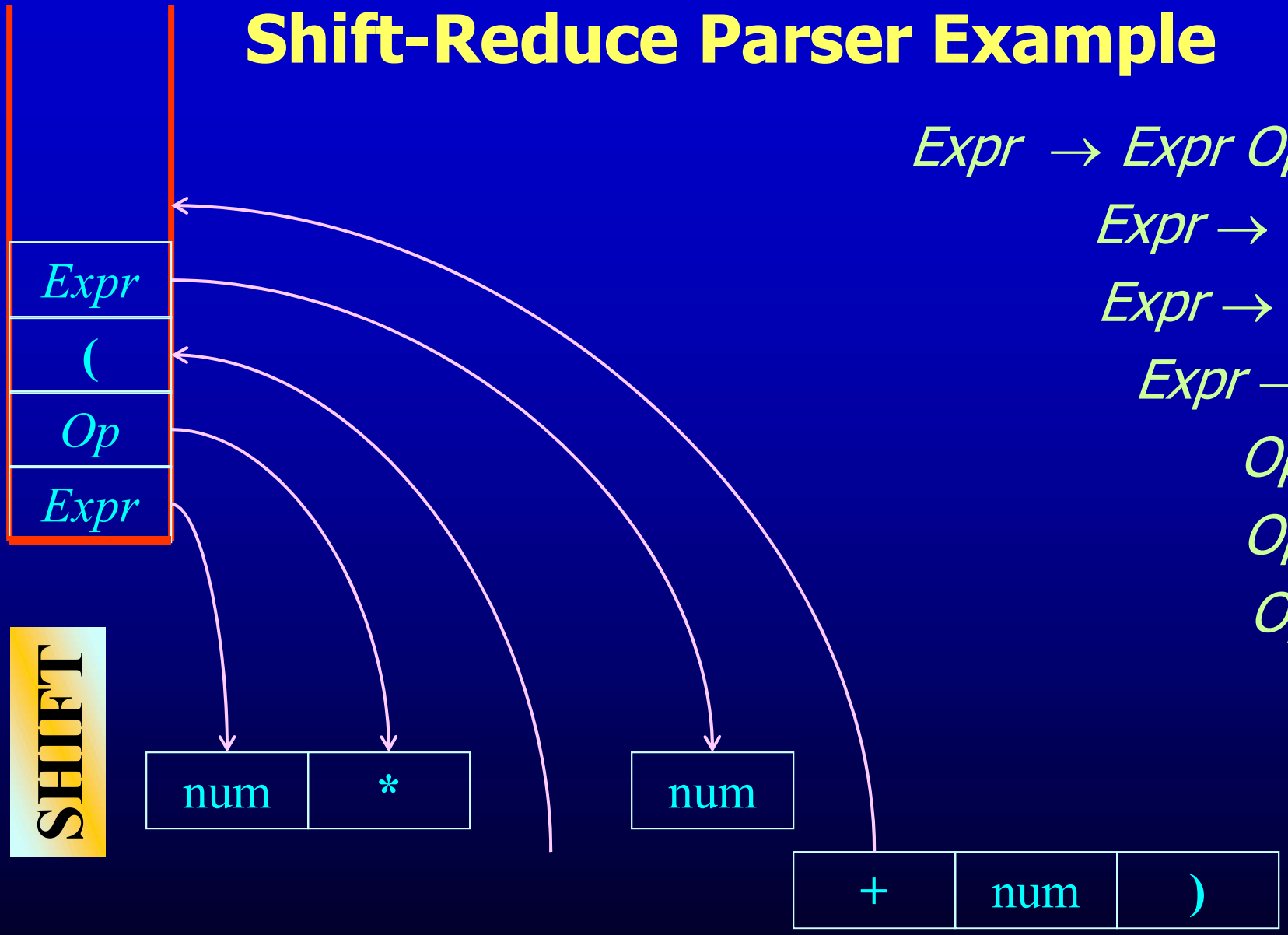
Expr → num

Op → +

Op → -

Op → *

Shift-Reduce Parser Example



Expr → *Expr Op Expr*

Expr → (*Expr*)

Expr → - *Expr*

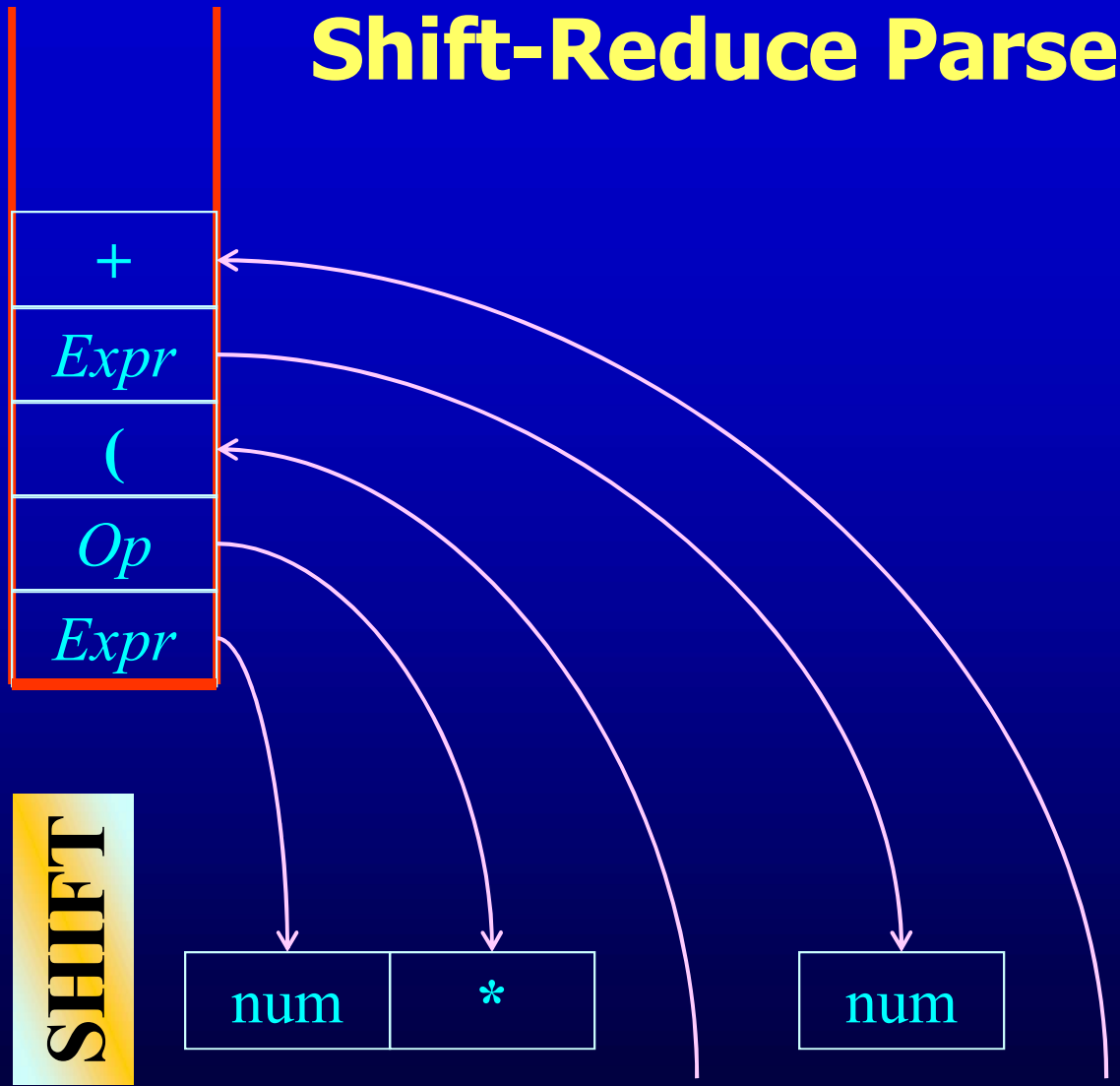
Expr → num

Op → +

Op → -

Op → *

Shift-Reduce Parser Example



Expr → *Expr Op Expr*

Expr → (*Expr*)

Expr → - *Expr*

Expr → num

Op → +

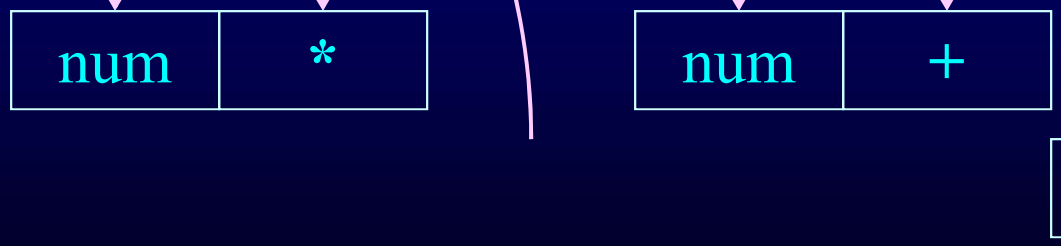
Op → -

Op → *

num	)

Shift-Reduce Parser Example

REDUCE



$Expr \rightarrow Expr\ Op\ \ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow -\ Expr$

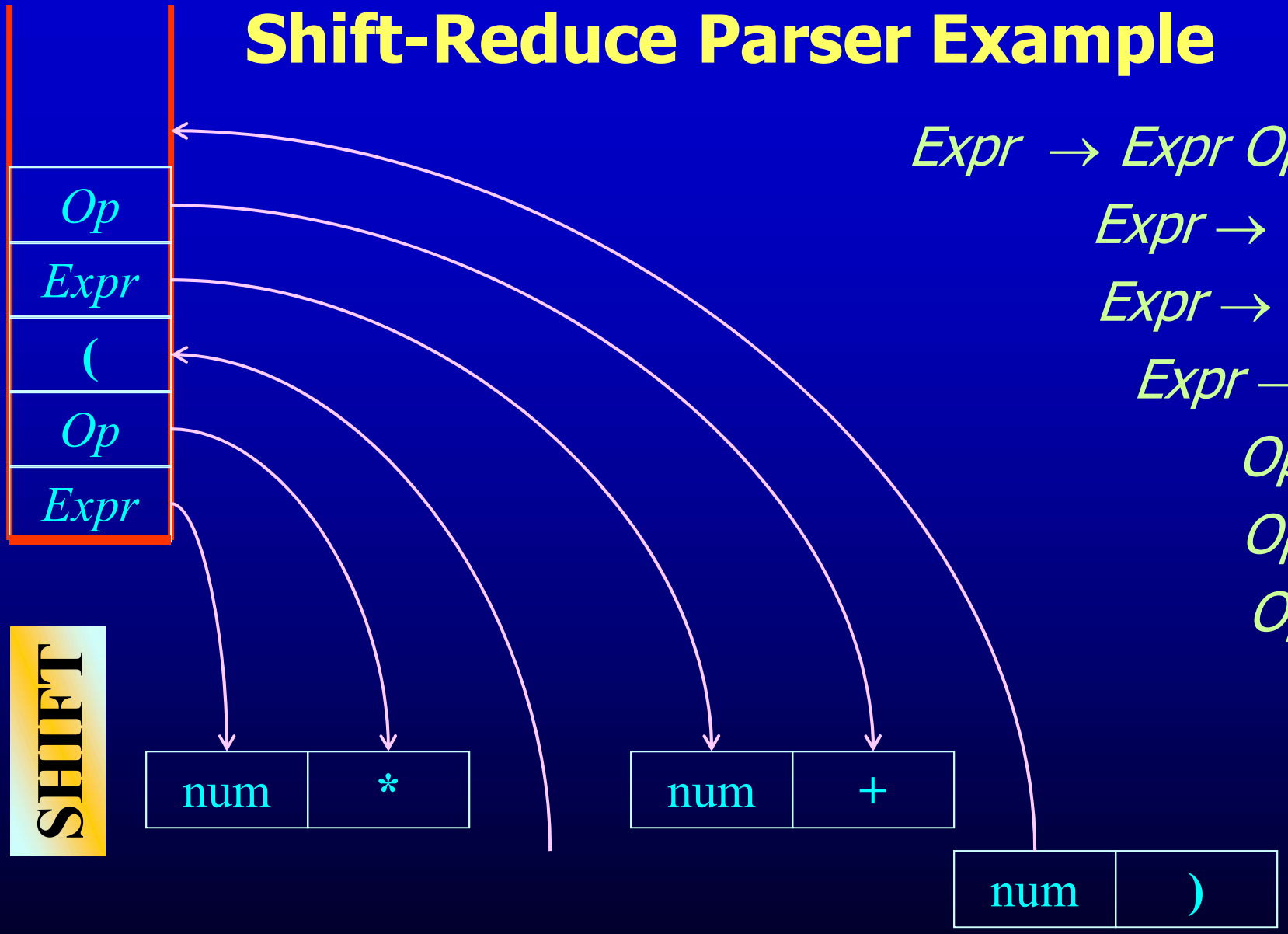
$Expr \rightarrow$ num

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Shift-Reduce Parser Example



Expr → *Expr Op Expr*

Expr → (*Expr*)

Expr → - *Expr*

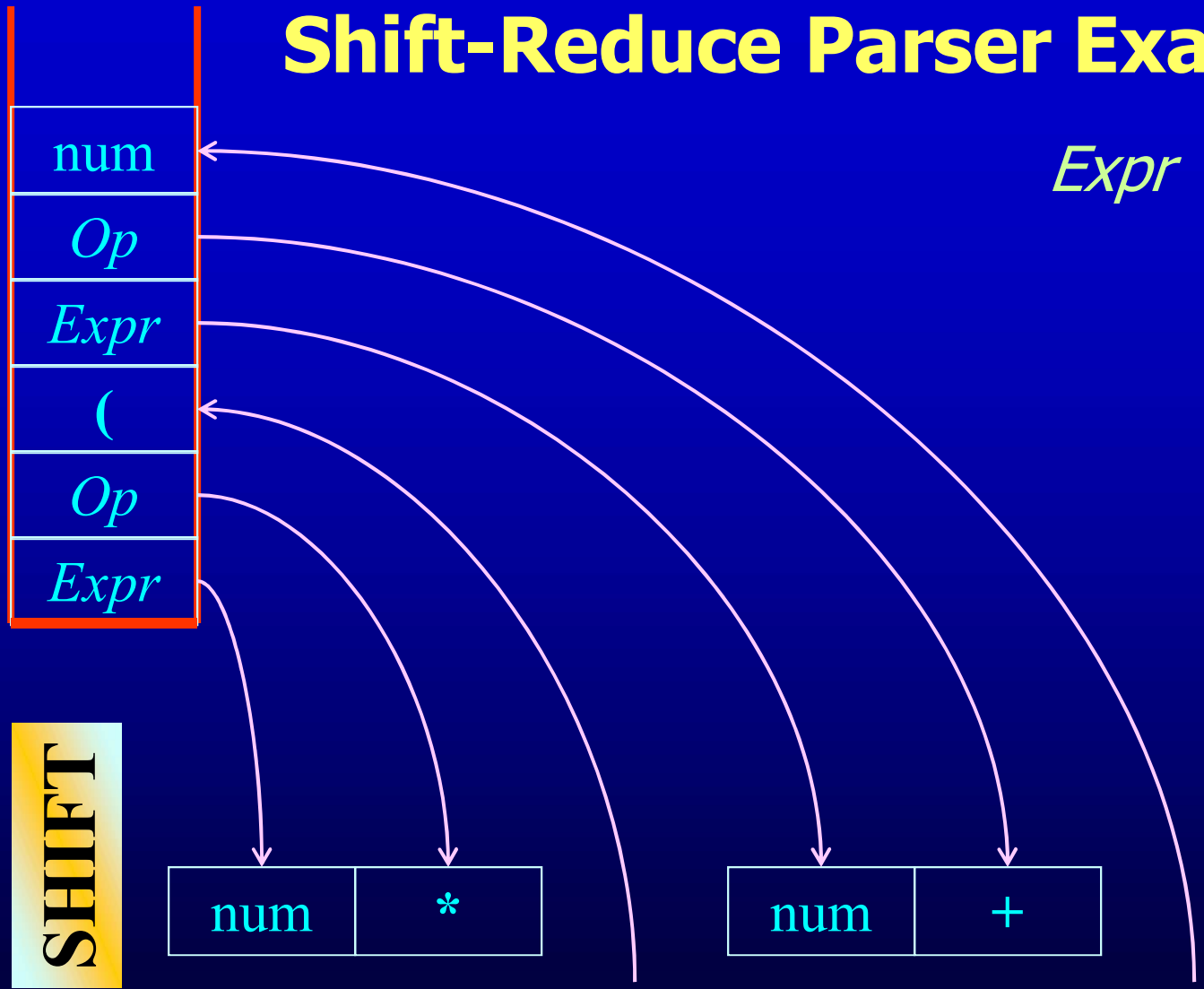
Expr → num

Op → +

Op → -

Op → *

Shift-Reduce Parser Example



Expr → *Expr Op Expr*

Expr → (*Expr*)

Expr → - *Expr*

Expr → num

Op → +

Op → -

Op → *

Shift-Reduce Parser Example

REDUCE

$Expr \rightarrow Expr\ Op\ \ Expr$

$Expr \rightarrow\ (Expr)$

$Expr \rightarrow\ -\ Expr$

$Expr \rightarrow$ num

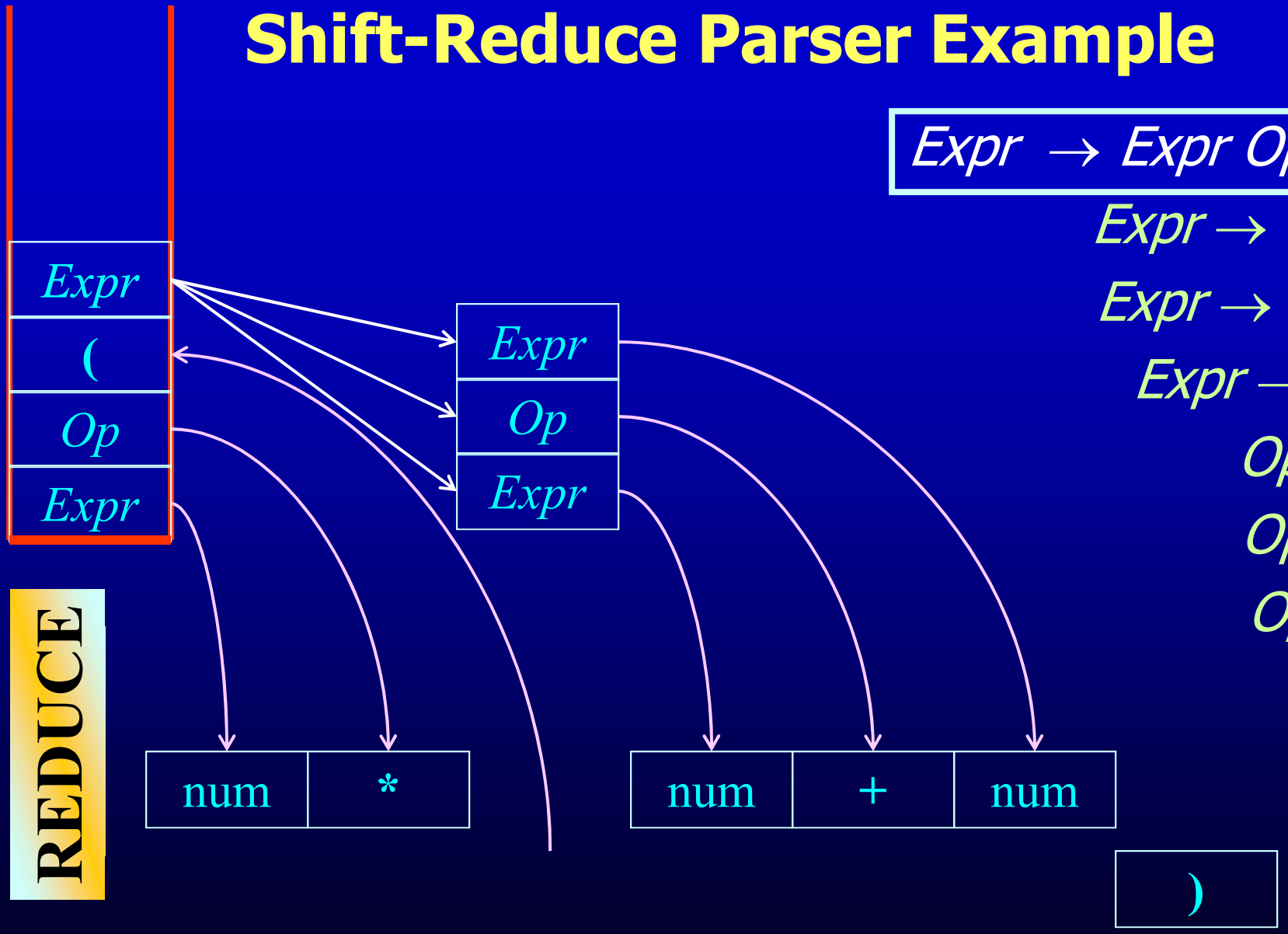
$Op \rightarrow\ +$

$Op \rightarrow\ -$

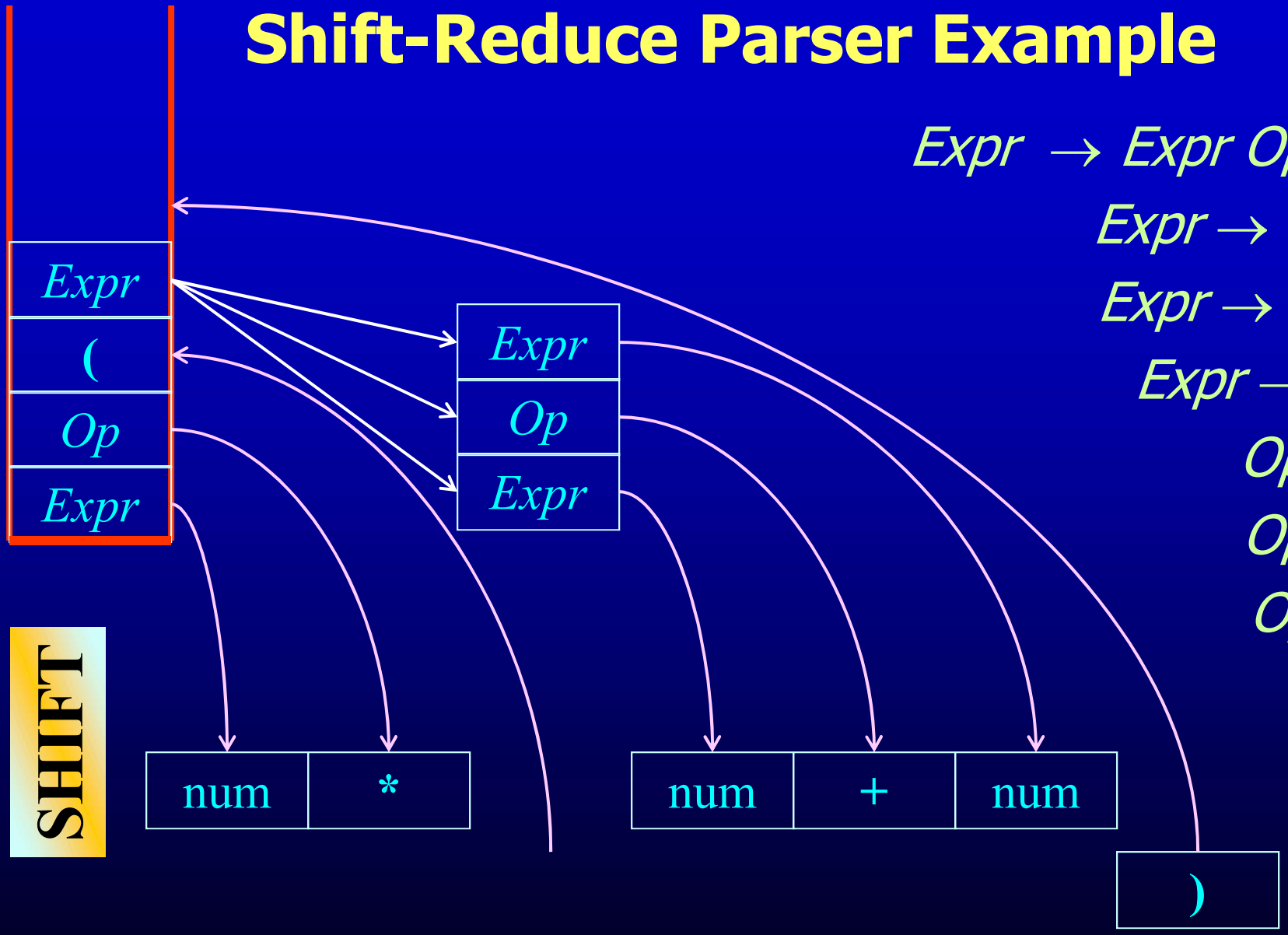
$Op \rightarrow\ *$

num | * | num | + | num |)

Shift-Reduce Parser Example



Shift-Reduce Parser Example



Expr → *Expr Op Expr*

Expr → (*Expr*)

Expr → - *Expr*

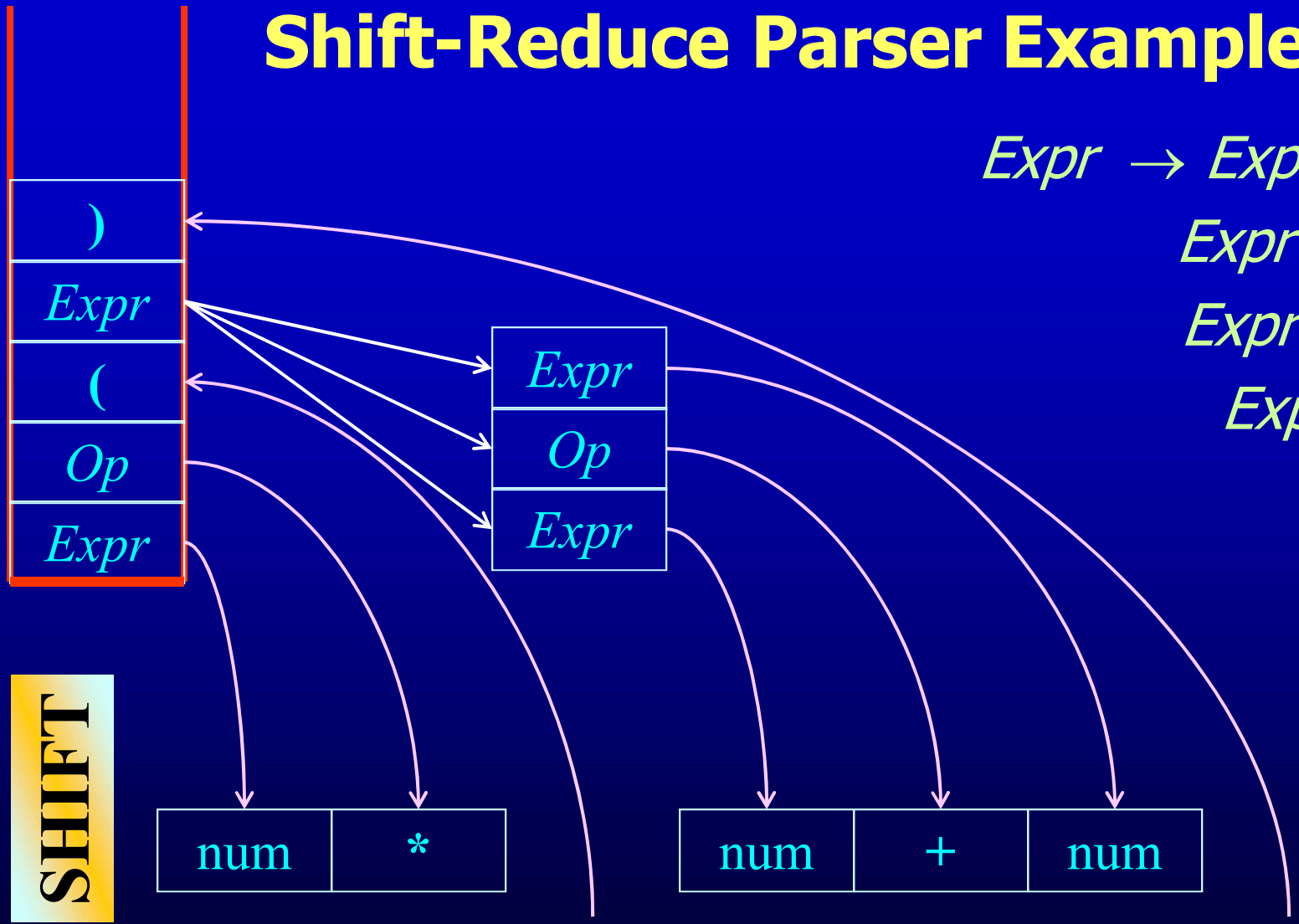
Expr → num

Op → +

Op → -

Op → *

Shift-Reduce Parser Example



Expr → *Expr Op Expr*

Expr → (*Expr*)

Expr → - *Expr*

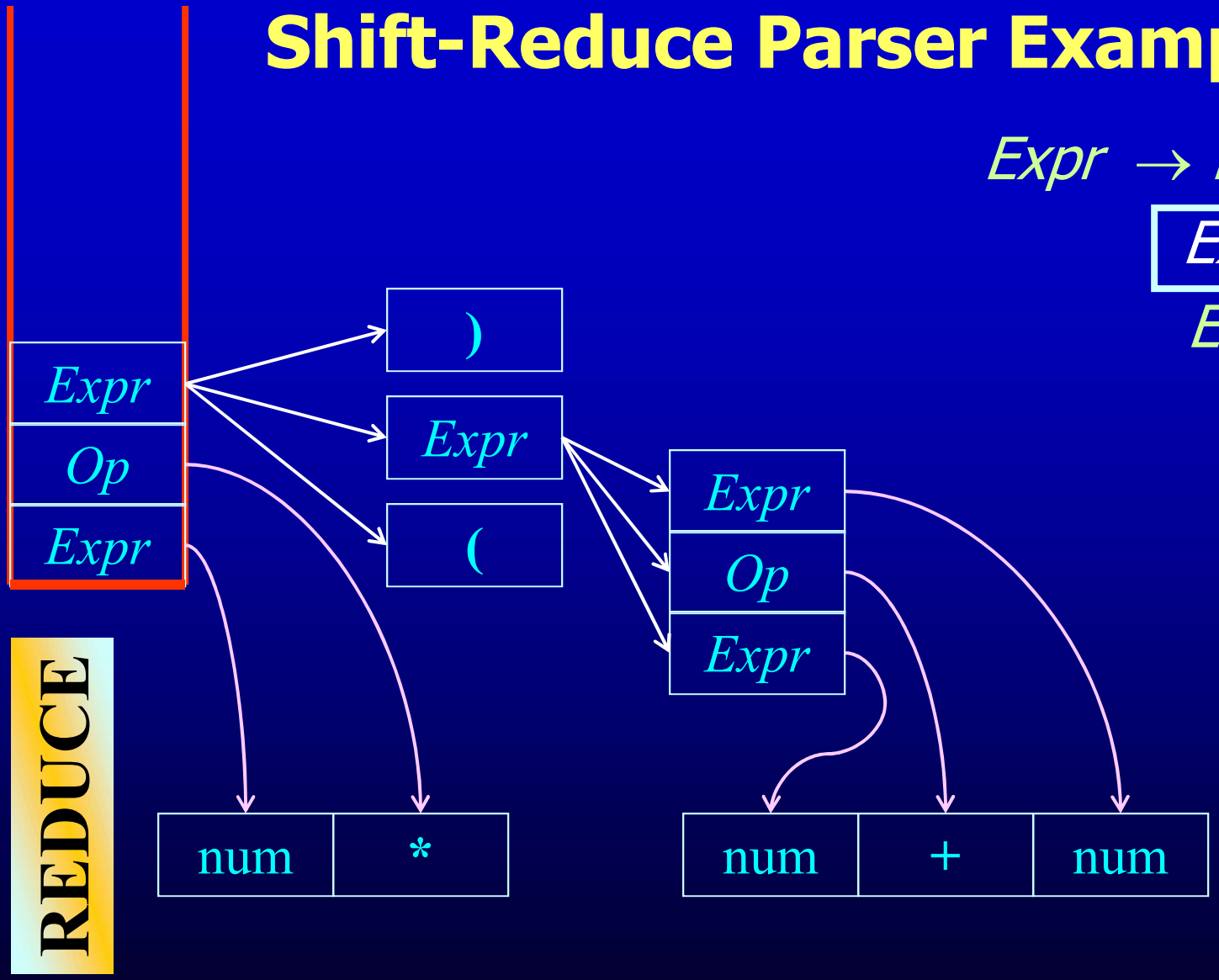
Expr → num

Op → +

Op → -

Op → *

Shift-Reduce Parser Example



$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow -\ Expr$

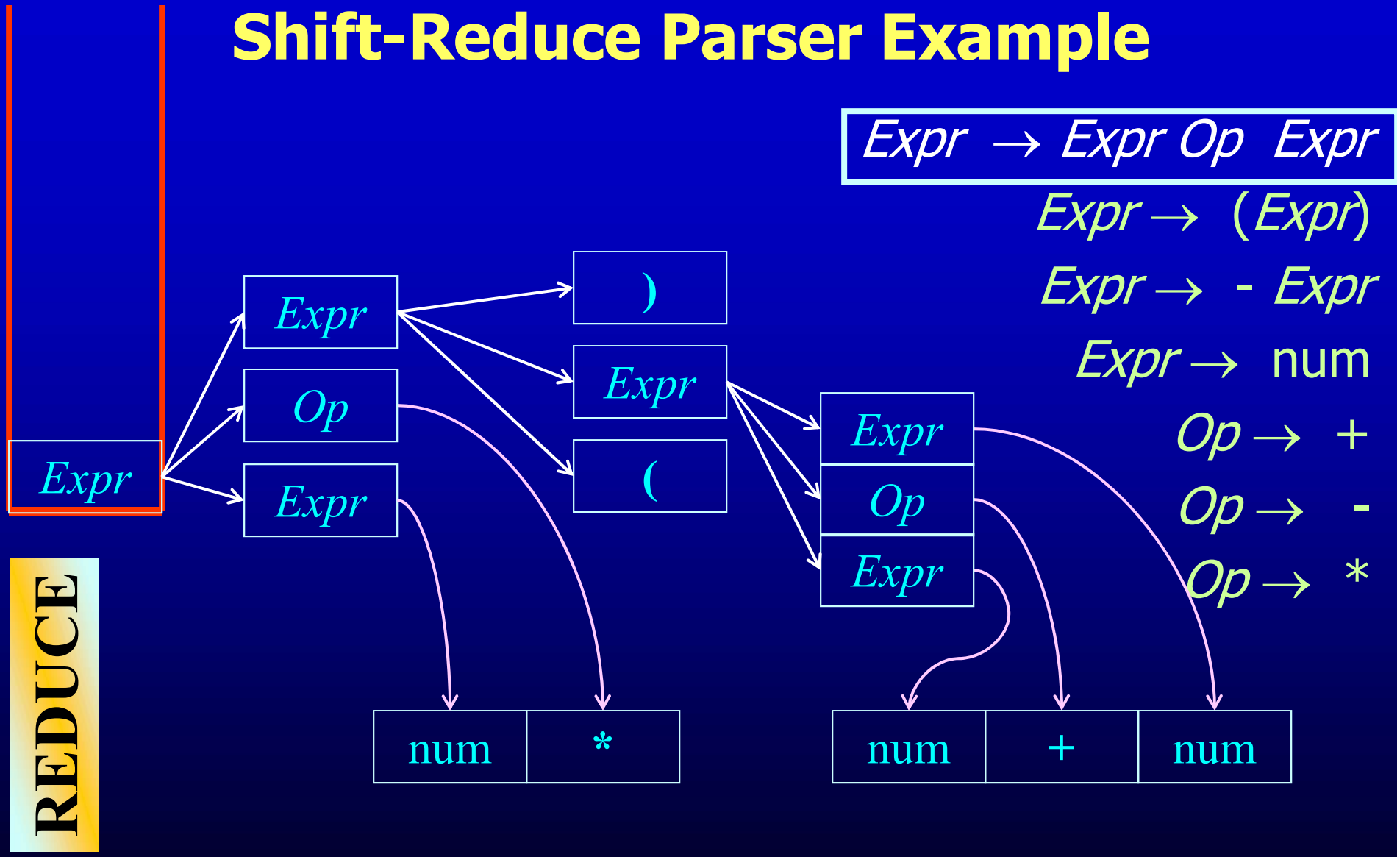
$Expr \rightarrow num$

$Op \rightarrow +$

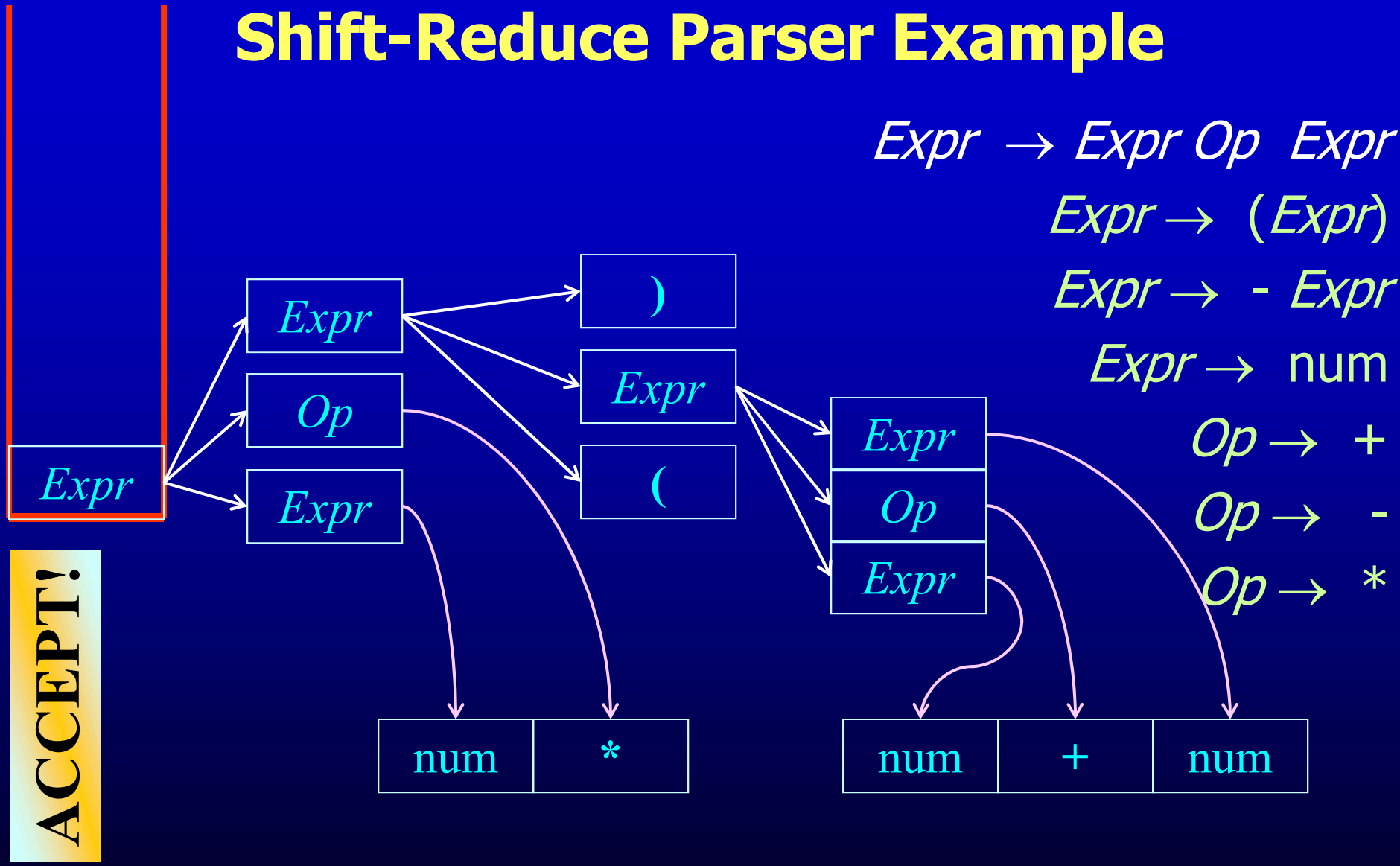
$Op \rightarrow -$

$Op \rightarrow *$

Shift-Reduce Parser Example



Shift-Reduce Parser Example



Basic Idea

- Goal: reconstruct parse tree for input string
- Read input from left to right
- Build tree in a bottom-up fashion
  - Apply production rules backwards (RHS → LHS)
- Build tree from right to left (LR parser)
- Use stack to hold pending sequences of terminals and nonterminals

Outline

- Bottom-Up parsing
- **Conflicts**
- Constructing a Shift-Reduce Parser
- Parser and syntax trees

Potential Conflicts

- Reduce/Reduce Conflict
  - Top of the stack may match RHS of multiple productions
  - Which production to use in the reduction?
- Shift/Reduce Conflict
  - Stack may match RHS of production
  - But that may not be the right match
  - May need to shift an input and later find a different reduction

Conflicts

- Original Grammar

  Expr → *Expr Op Expr*

  Expr → (*Expr*)

  Expr → - *Expr*

  Expr → num

  Op → +

  Op → -

  Op → *

- New Grammar

  Expr → *Expr Op Expr*

  Expr → *Expr* - *Expr*

  Expr → (*Expr*)

  Expr → *Expr* -

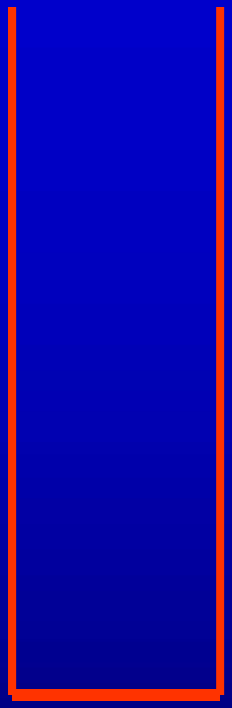
  Expr → num

  Op → +

  Op → -

  Op → *

Conflicts



Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

Expr → num

Op → +

Op → -

Op → *

num	-	num

Conflicts

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

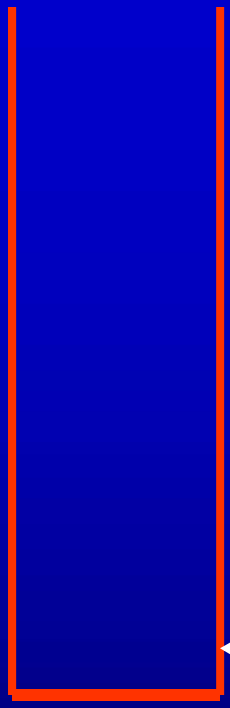
Expr → *Expr* -

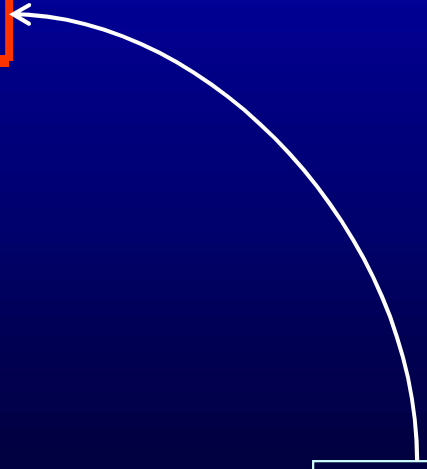
Expr → num

Op → +

Op → -

Op → *





SHIFT

num	-	num

Conflicts

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

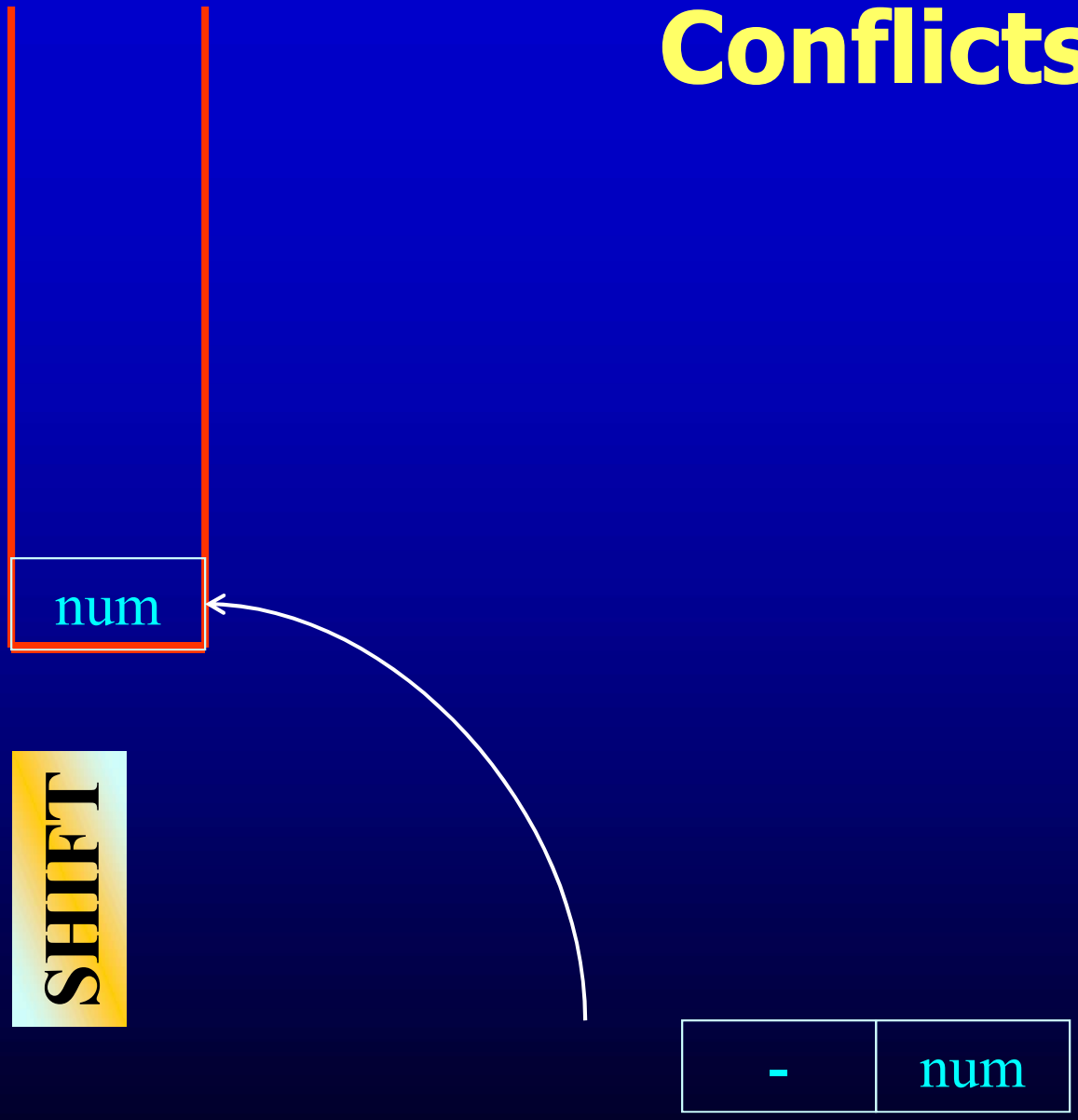
Expr → *Expr* -

Expr → num

Op → +

Op → -

Op → *



Conflicts

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

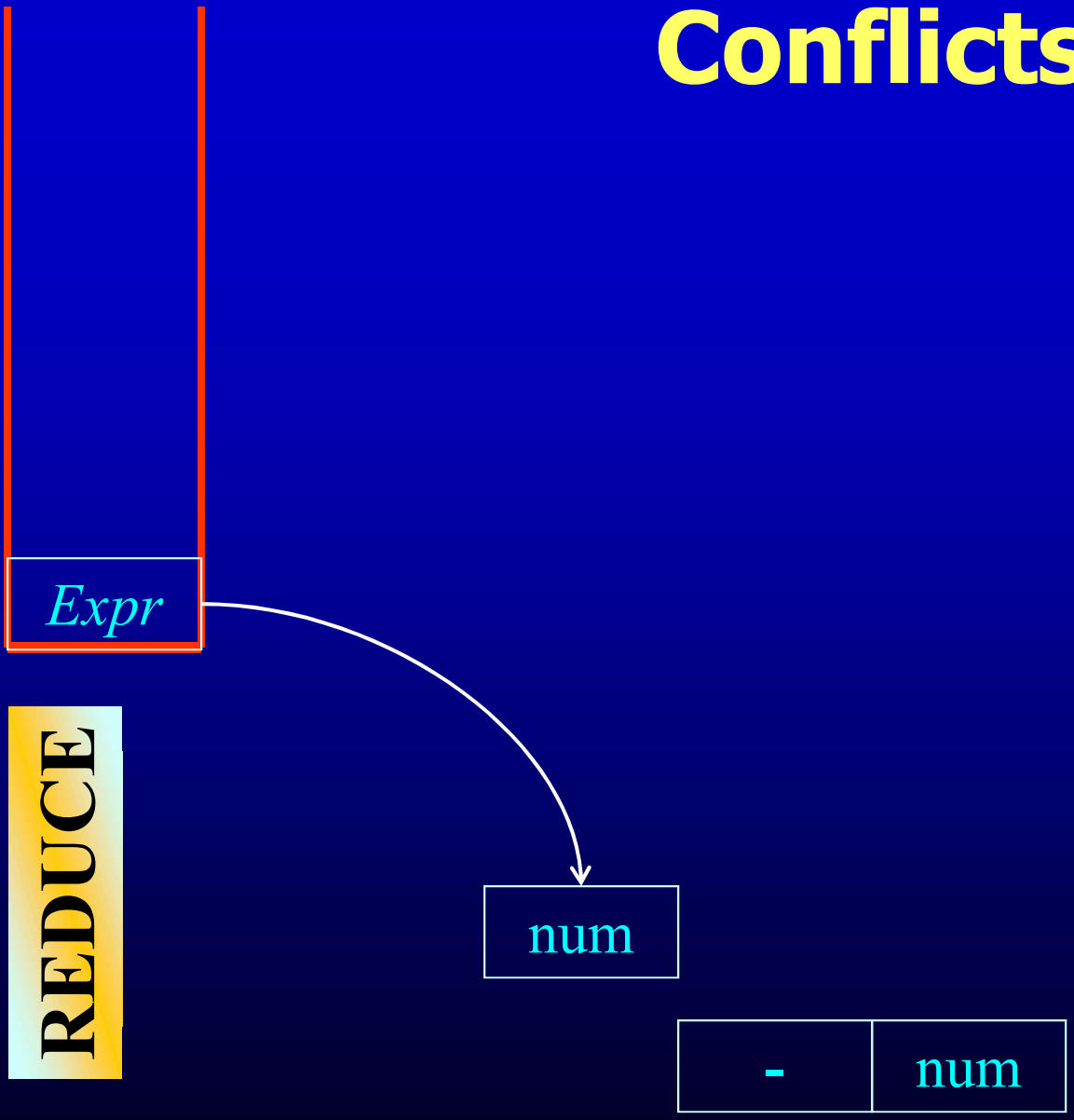
Expr → *Expr* -

Expr → num

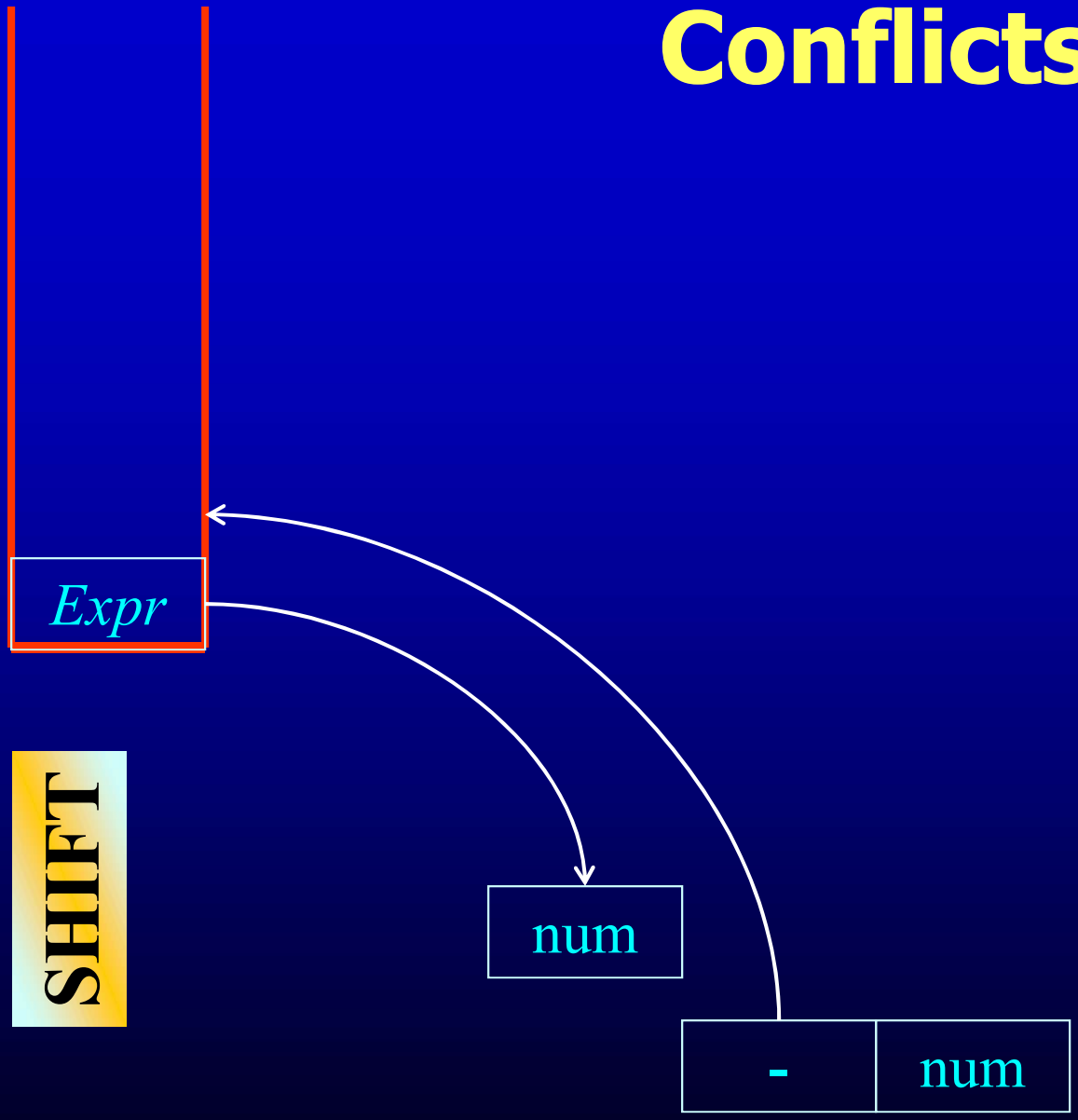
Op → +

Op → -

Op → *



Conflicts



Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

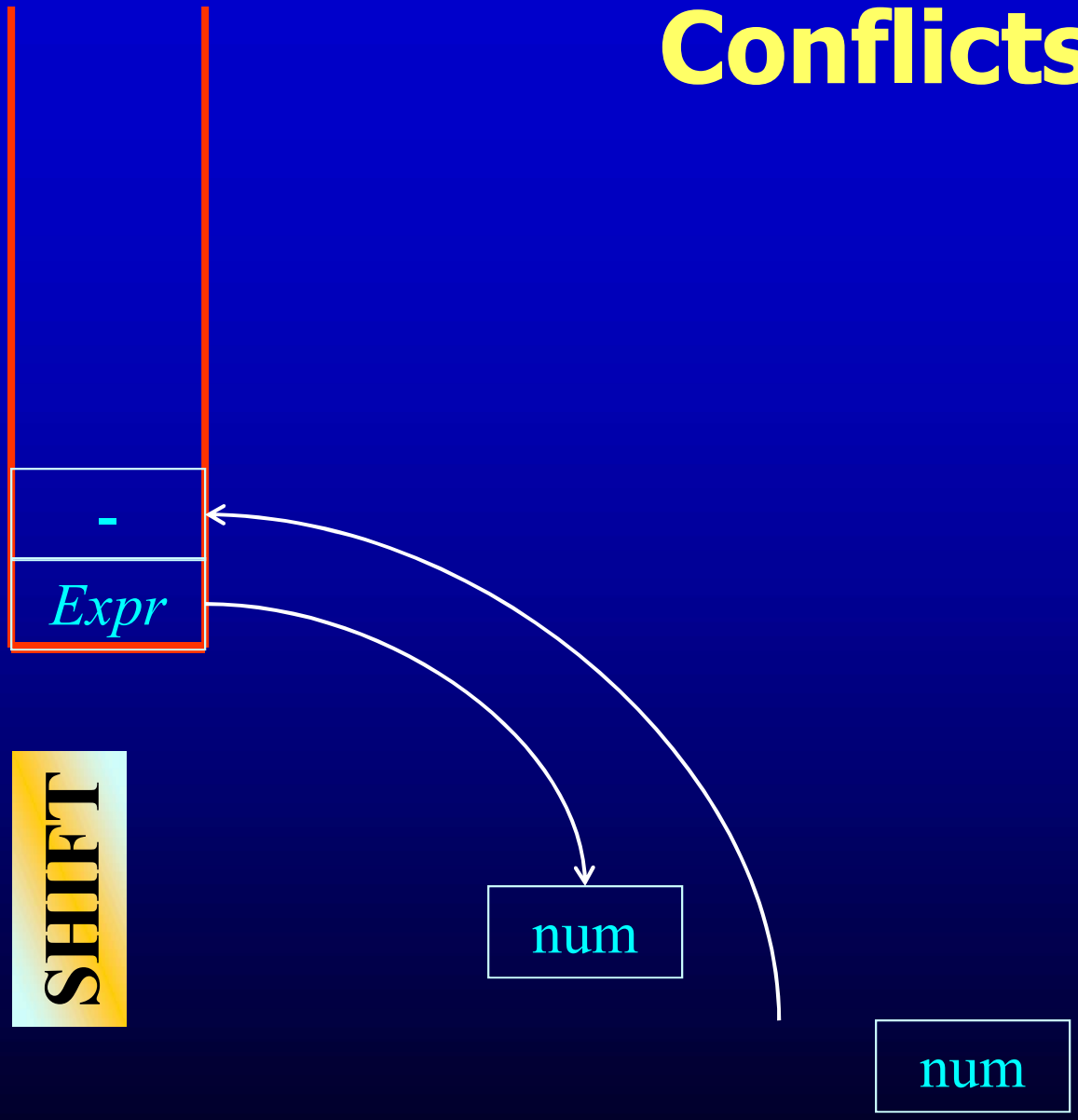
Expr → num

Op → +

Op → -

Op → *

Conflicts



Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

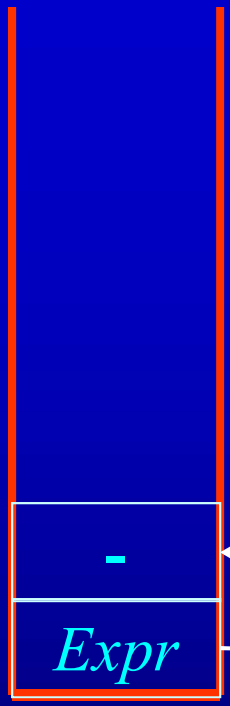
Expr → num

Op → +

Op → -

Op → *

Shift/Reduce/Reduce Conflict



Options:

- Reduce
- Reduce
- Shift

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

Expr → num

Op → +

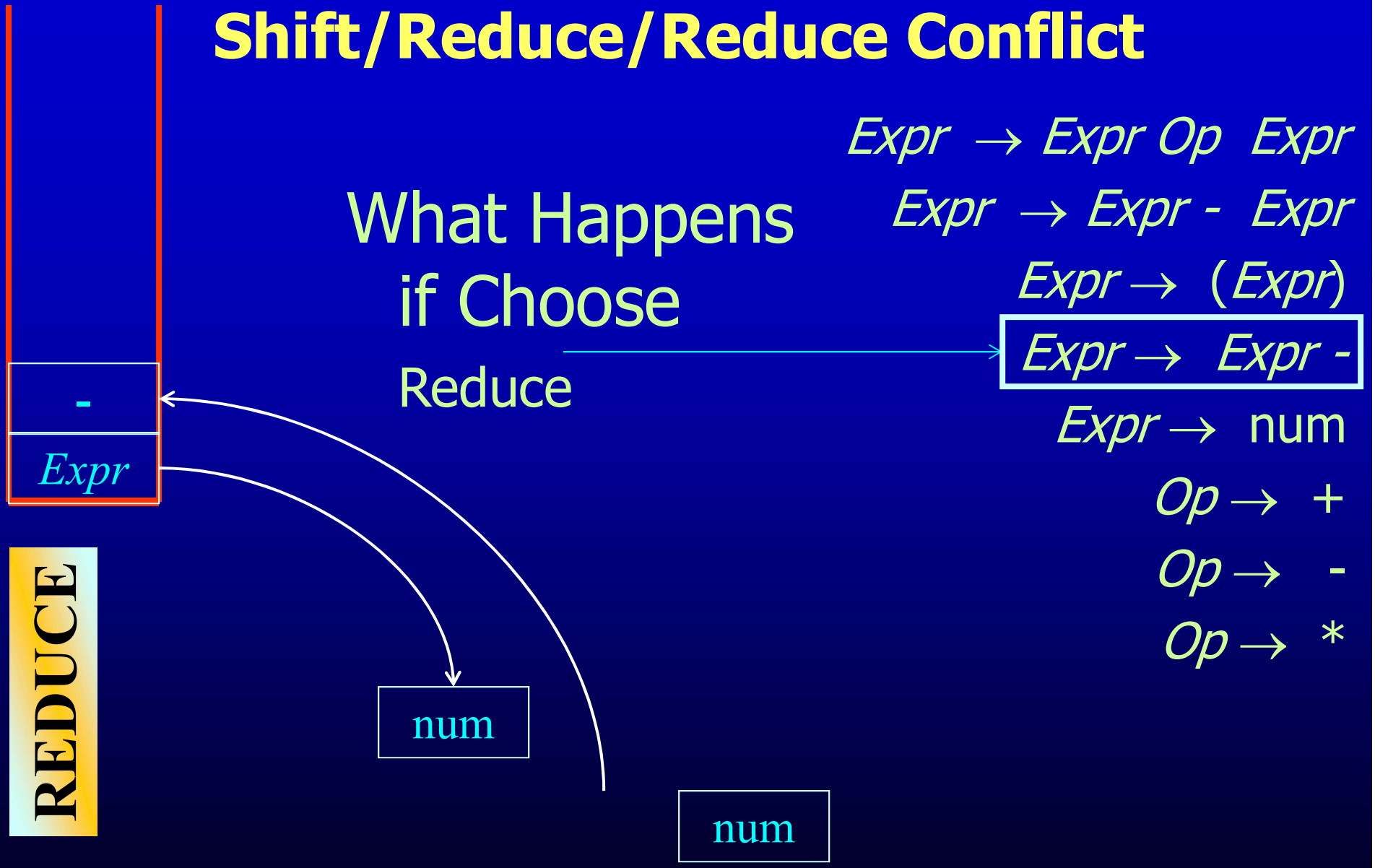
Op → -

Op → *

num

num

Shift/Reduce/Reduce Conflict



Shift/Reduce/Reduce Conflict

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

Expr → num

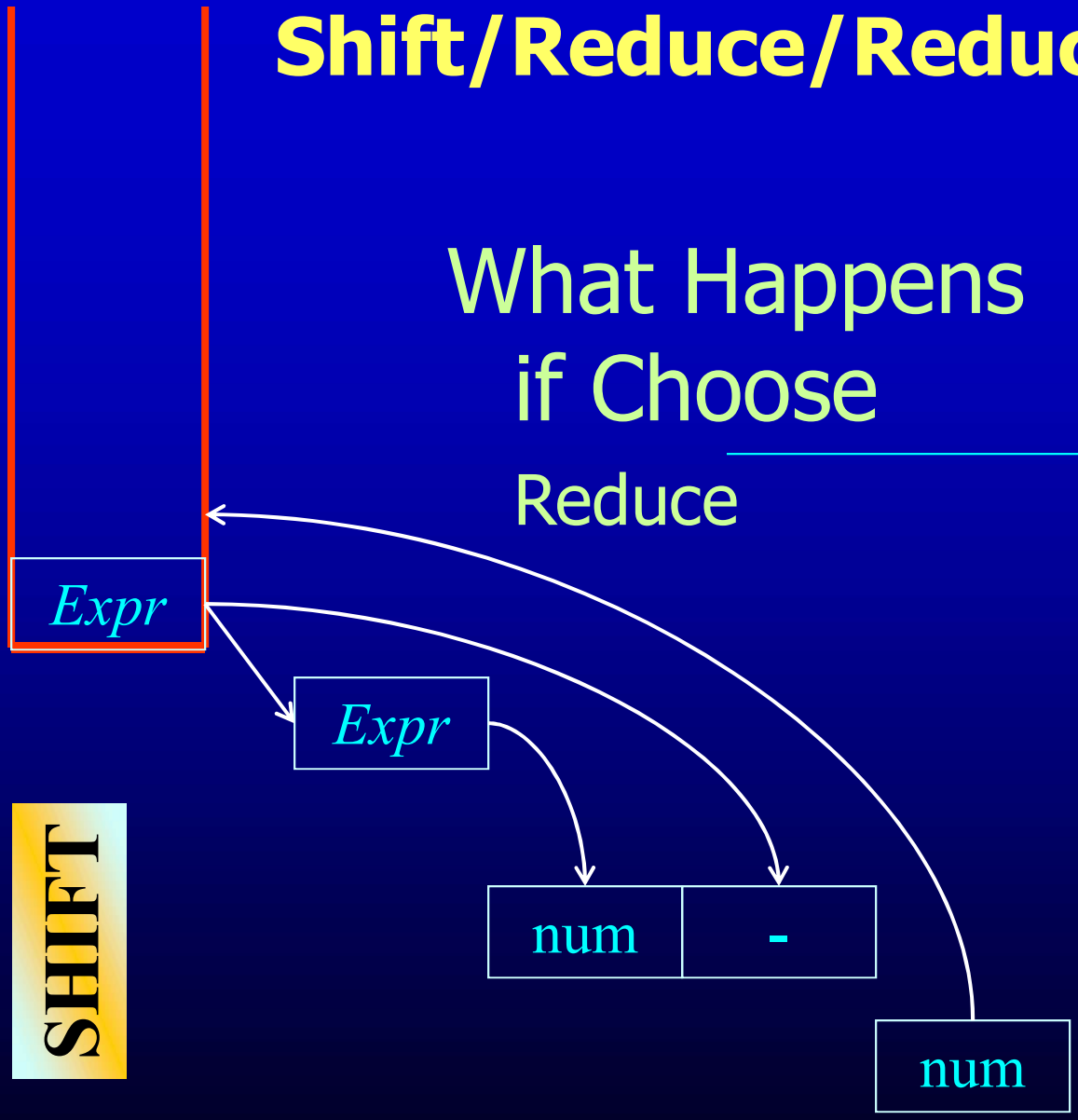
Op → +

Op → -

Op → *

What Happens if Choose

→ Reduce



Shift/Reduce/Reduce Conflict

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

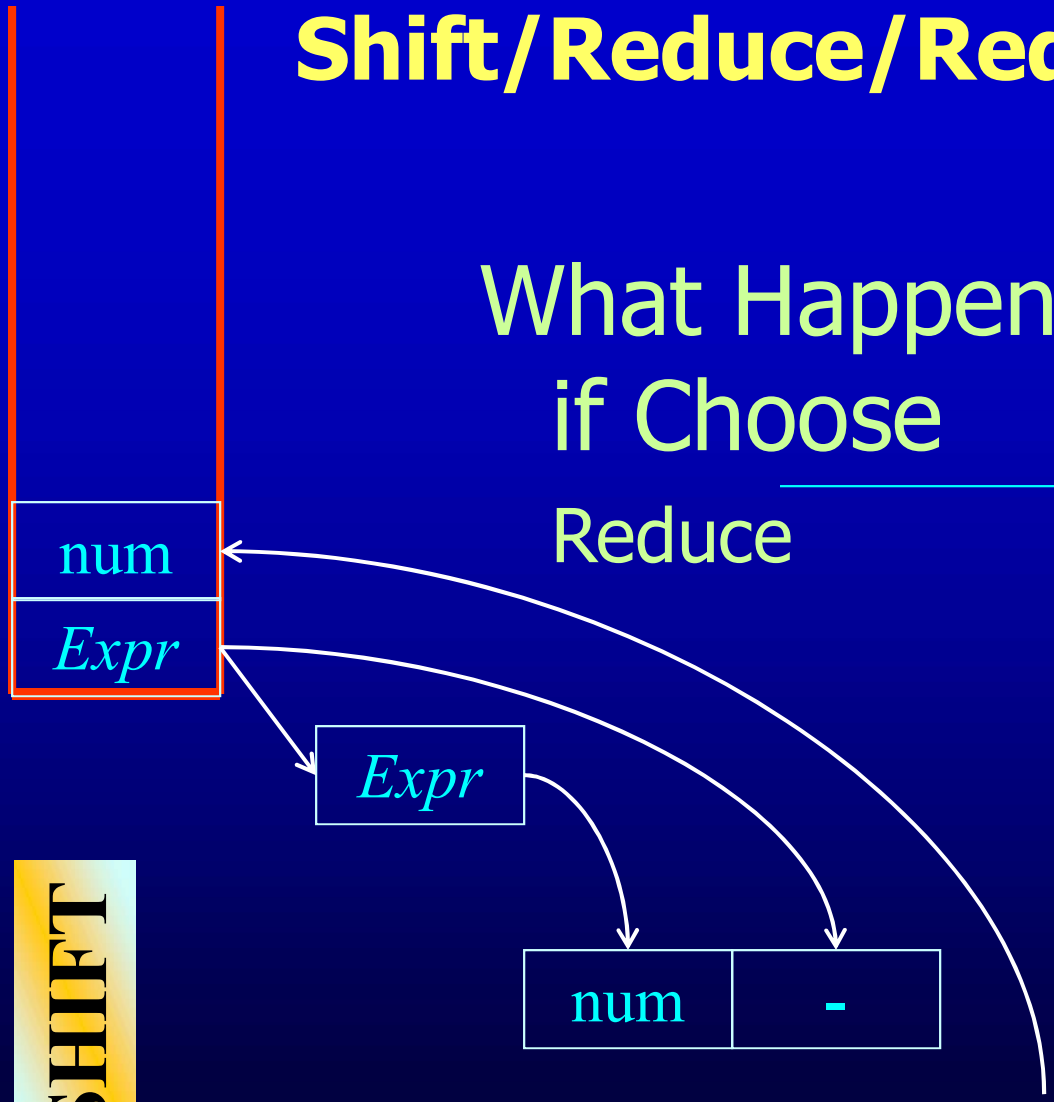
Expr → num

Op → +

Op → -

Op → *

What Happens if Choose Reduce



Shift/Reduce/Reduce Conflict

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

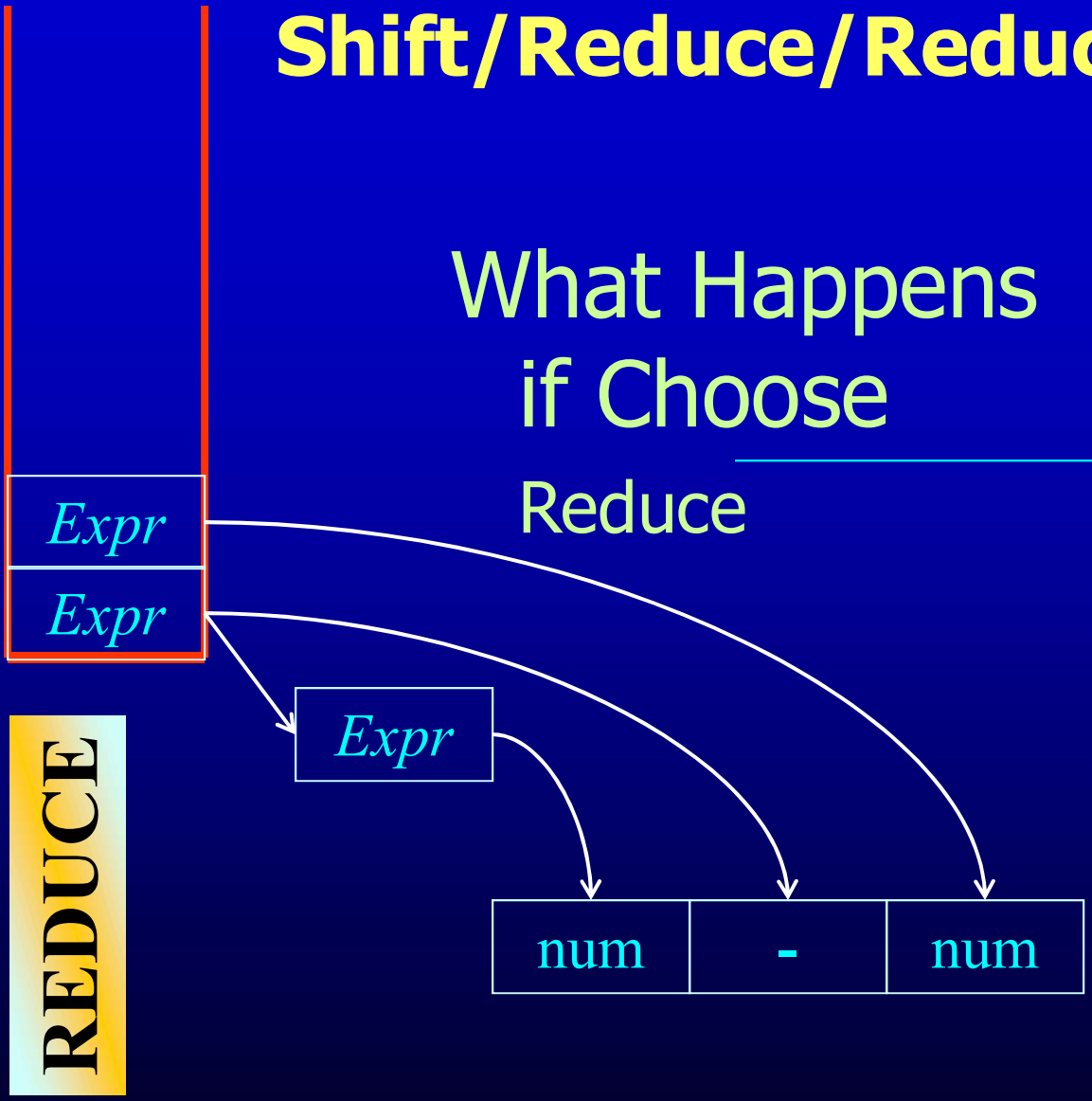
Expr → num

Op → +

Op → -

Op → *

What Happens if Choose Reduce



Shift/Reduce/Reduce Conflict

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

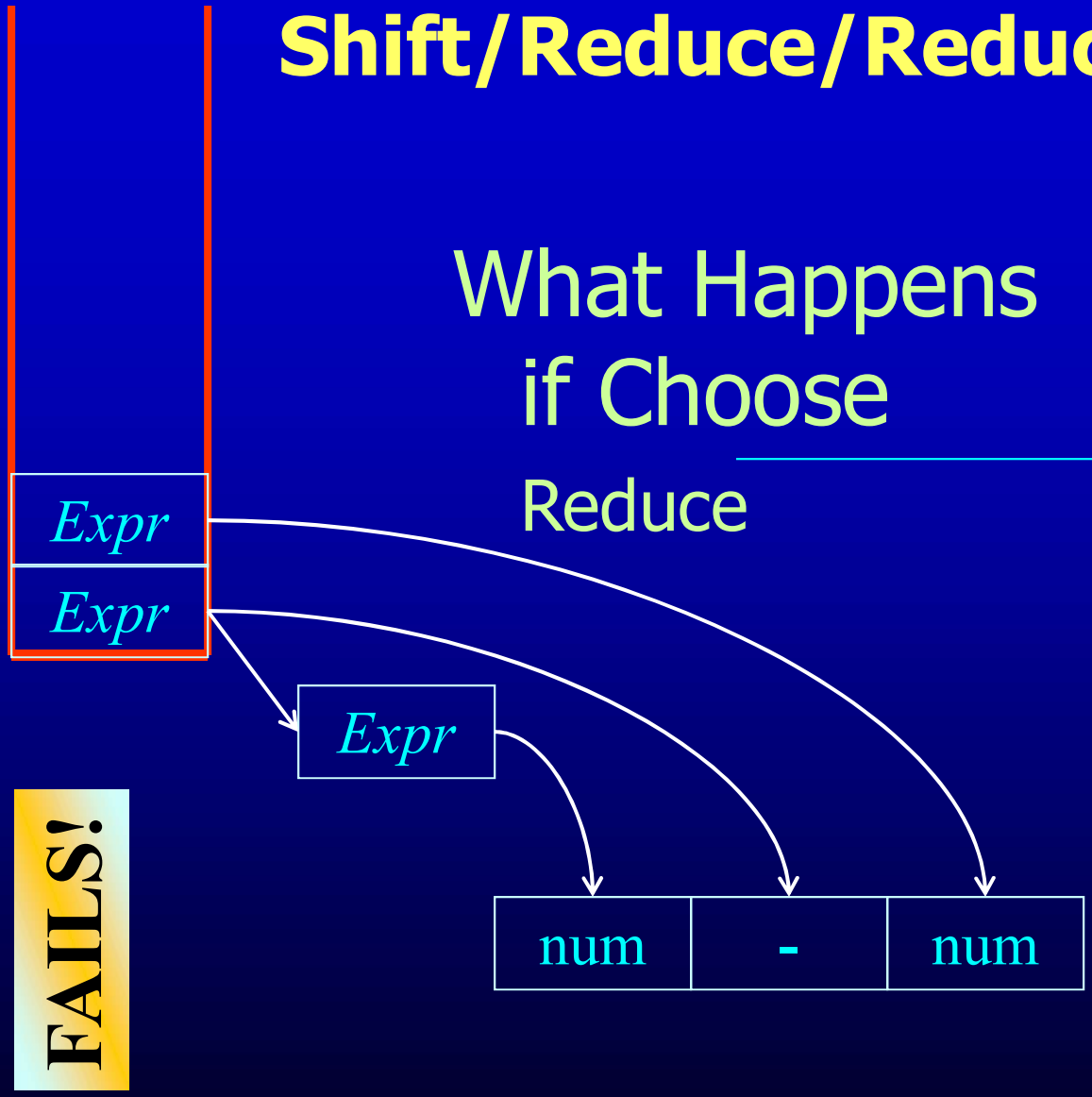
Expr → num

Op → +

Op → -

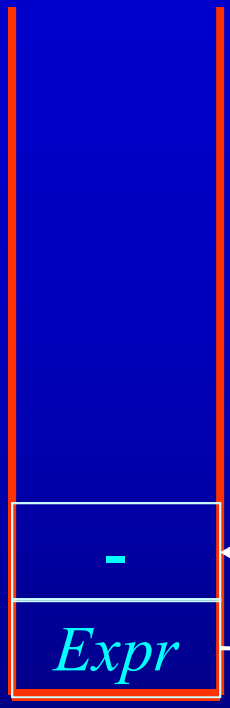
Op → *

What Happens if Choose Reduce



FAILS!

Shift/Reduce/Reduce Conflict



Both of These Actions Work

Reduce

Shift

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

Expr → num

Op → +

Op → -

Op → *

Shift/Reduce/Reduce Conflict

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

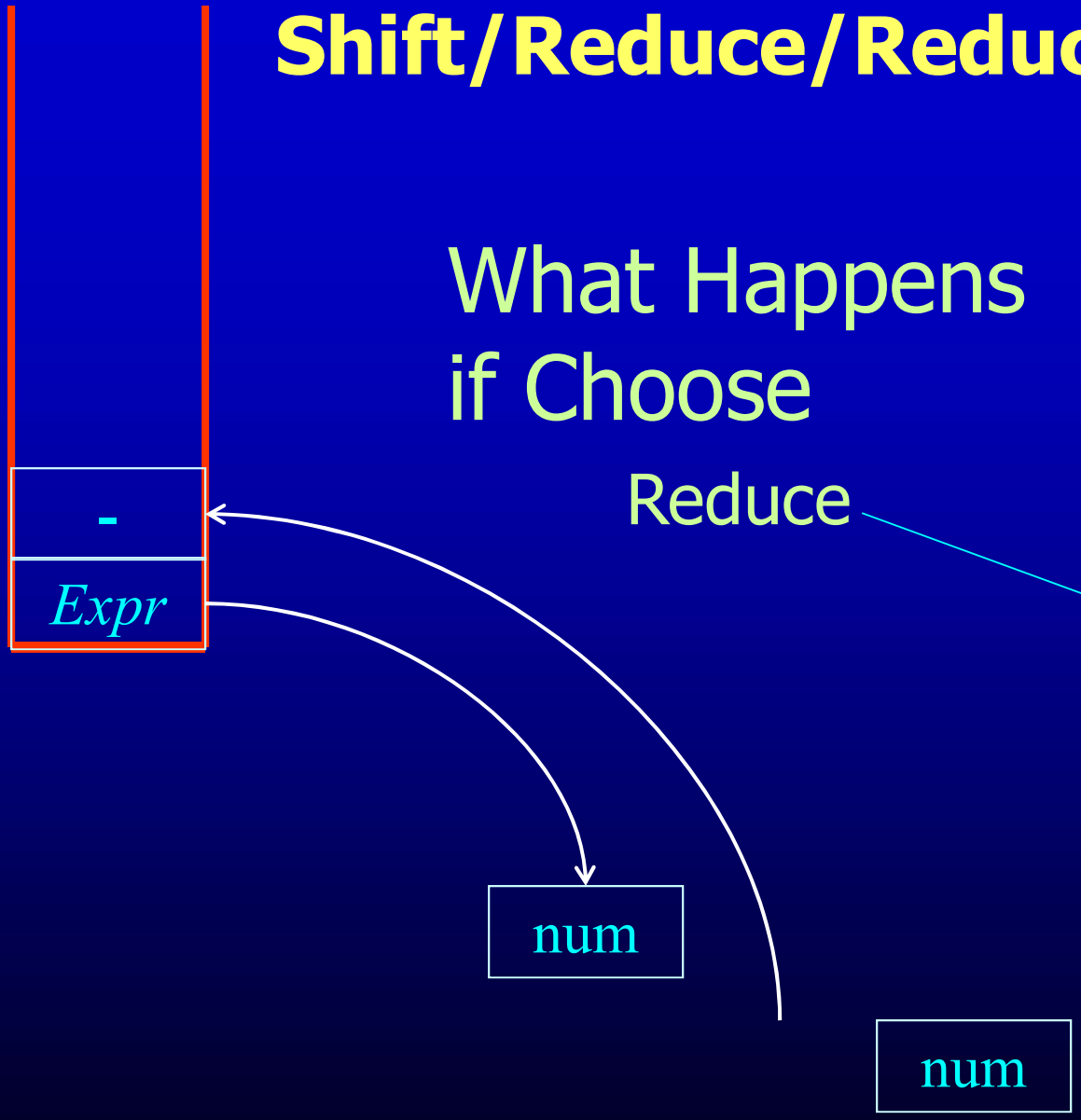
Expr → num

Op → +

Op → -

Op → *

What Happens if Choose Reduce



Shift/Reduce/Reduce Conflict

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

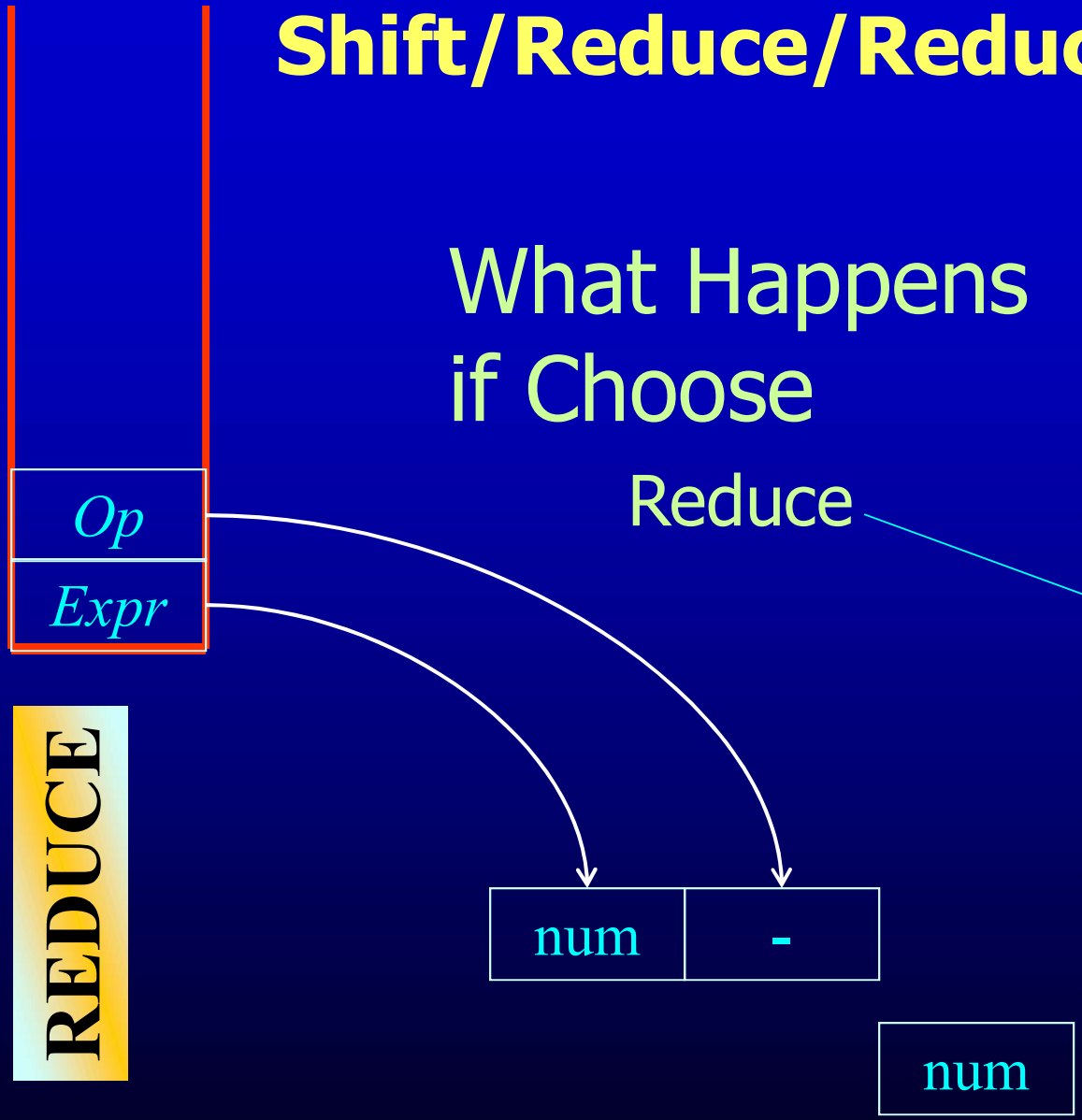
Expr → num

Op → +

Op → -

Op → *

What Happens if Choose Reduce



Shift/Reduce/Reduce Conflict

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

Expr → num

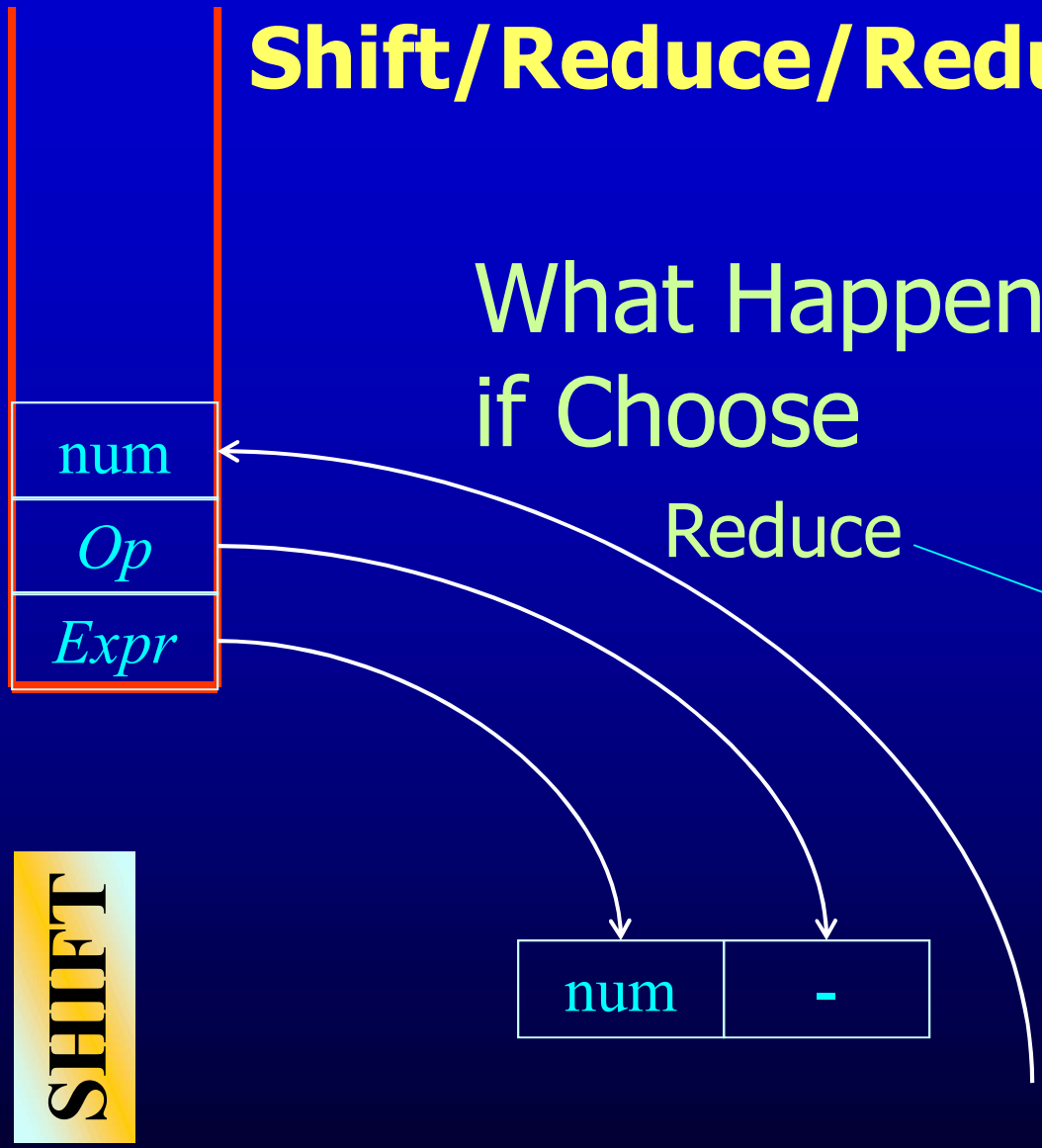
Op → +

Op → -

Op → *

What Happens if Choose

Reduce



Shift/Reduce/Reduce Conflict

REDUCE

What Happens if Choose Reduce

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

Expr → num

Op → +

Op → -

Op → *

Shift/Reduce/Reduce Conflict

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

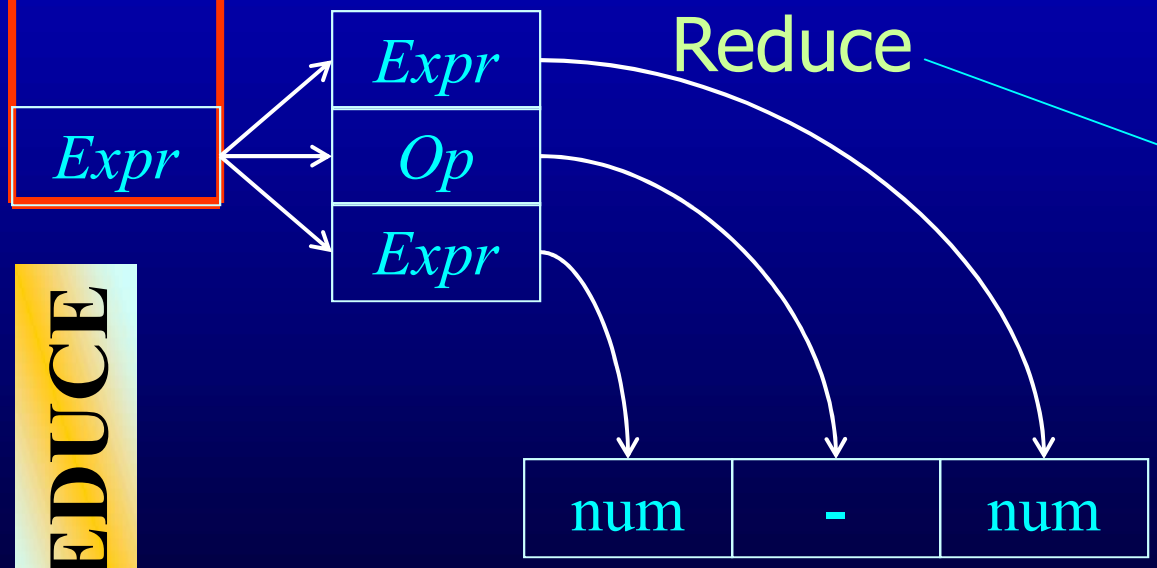
Expr → num

Op → +

Op → -

Op → *

What Happens if Choose Reduce



REDUCE

Shift/Reduce/Reduce Conflict

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

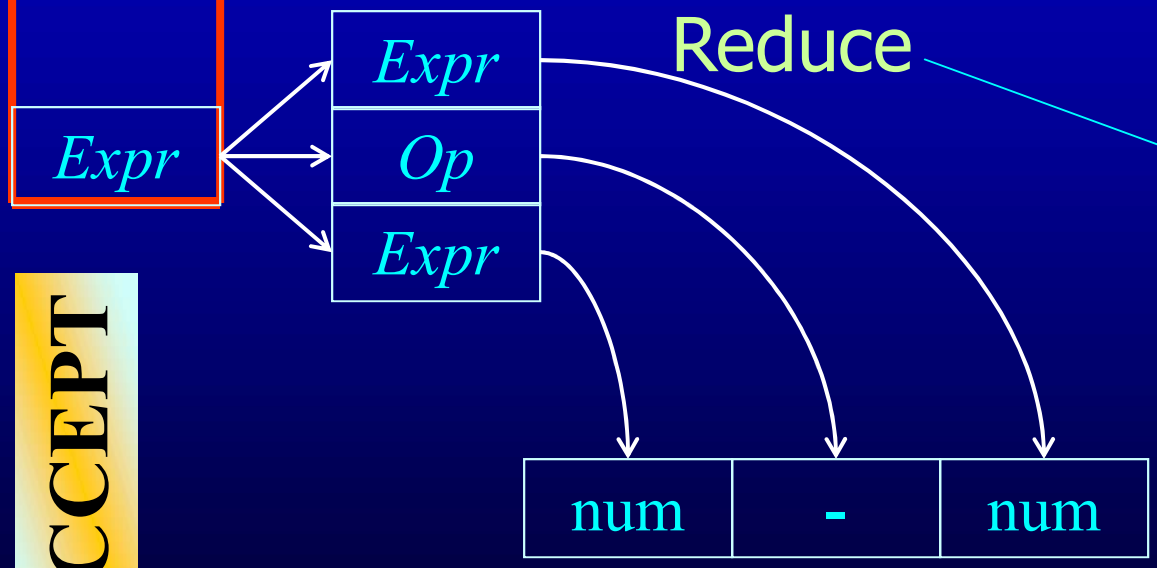
Expr → num

Op → +

Op → -

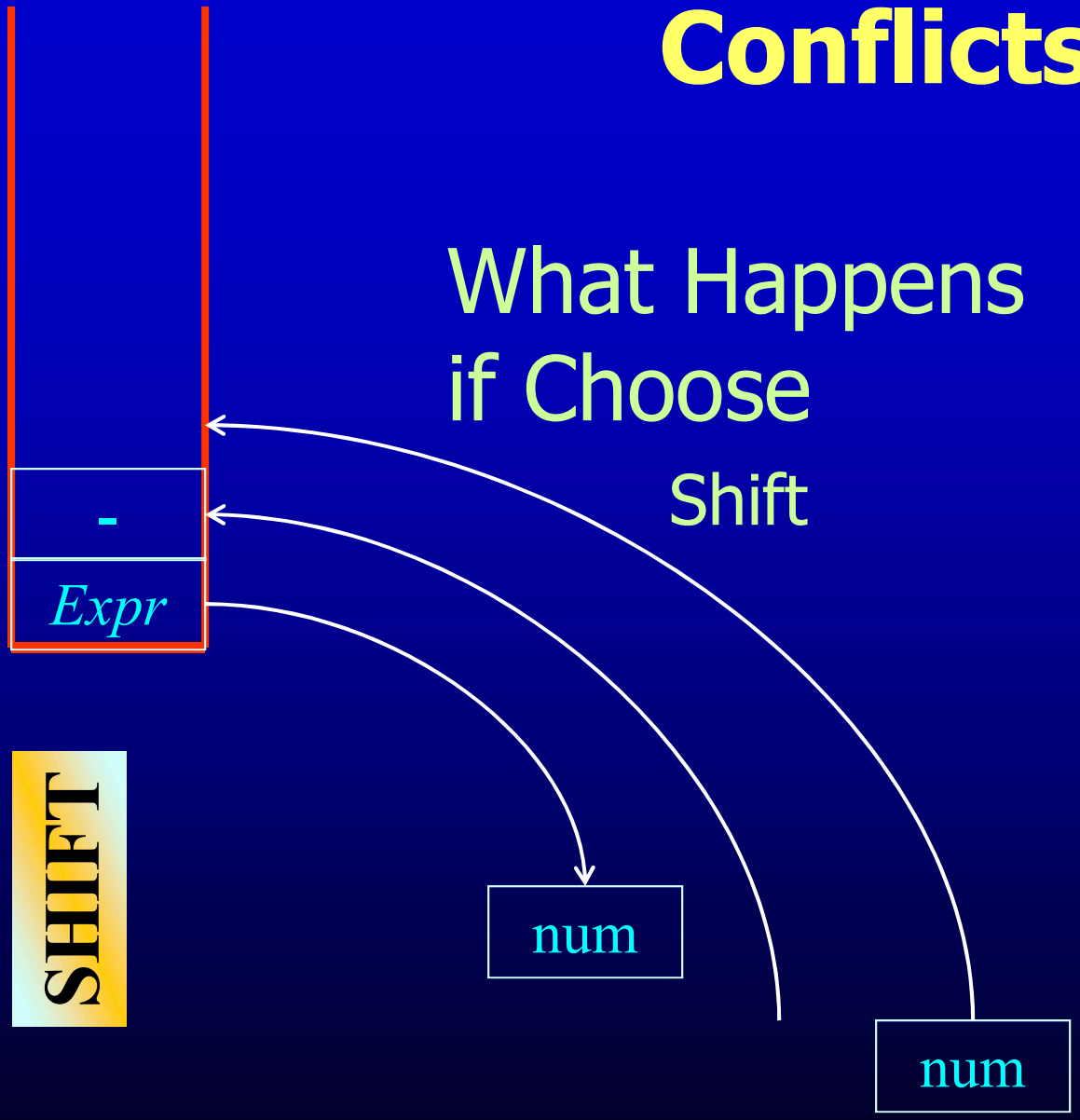
Op → *

What Happens if Choose Reduce



ACCEPT

Conflicts



What Happens if Choose Shift

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow Expr$ - $Expr$

$Expr \rightarrow$ ($Expr$)

$Expr \rightarrow Expr$ -

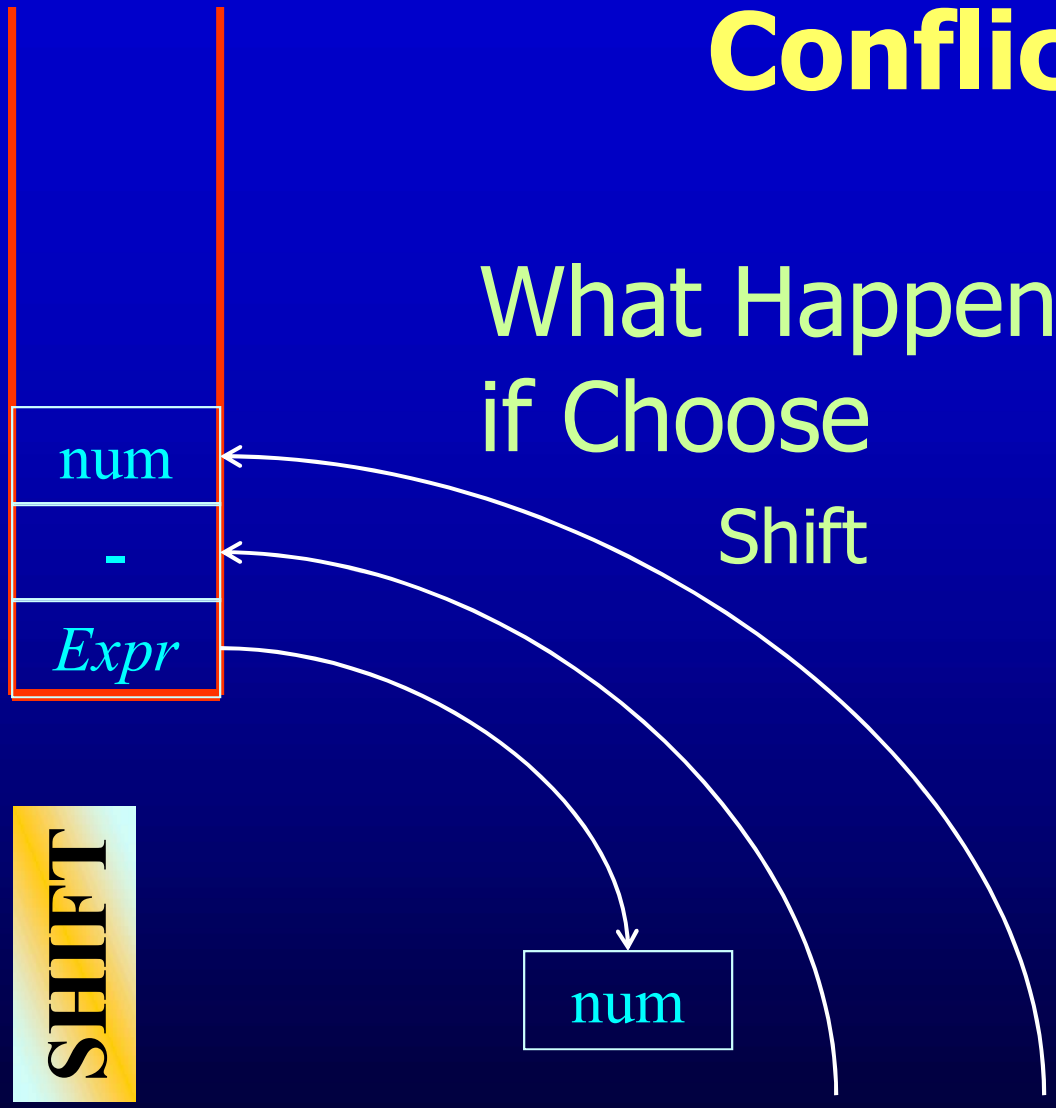
$Expr \rightarrow$ num

$Op \rightarrow$ +

$Op \rightarrow$ -

$Op \rightarrow$ *

Conflicts



What Happens if Choose Shift

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

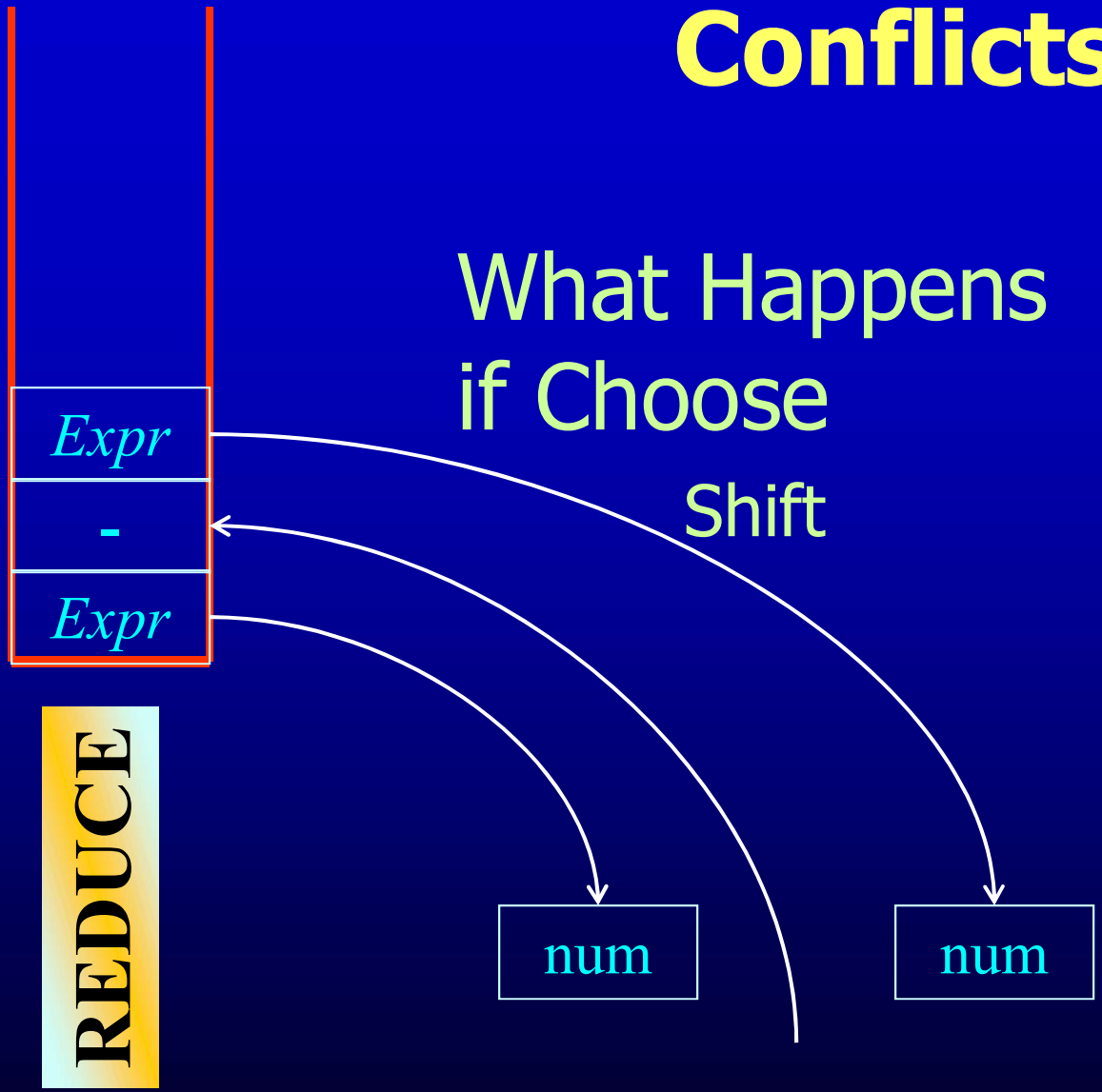
Expr → num

Op → +

Op → -

Op → *

Conflicts



What Happens if Choose Shift

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

Expr → num

Op → +

Op → -

Op → *

Conflicts

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

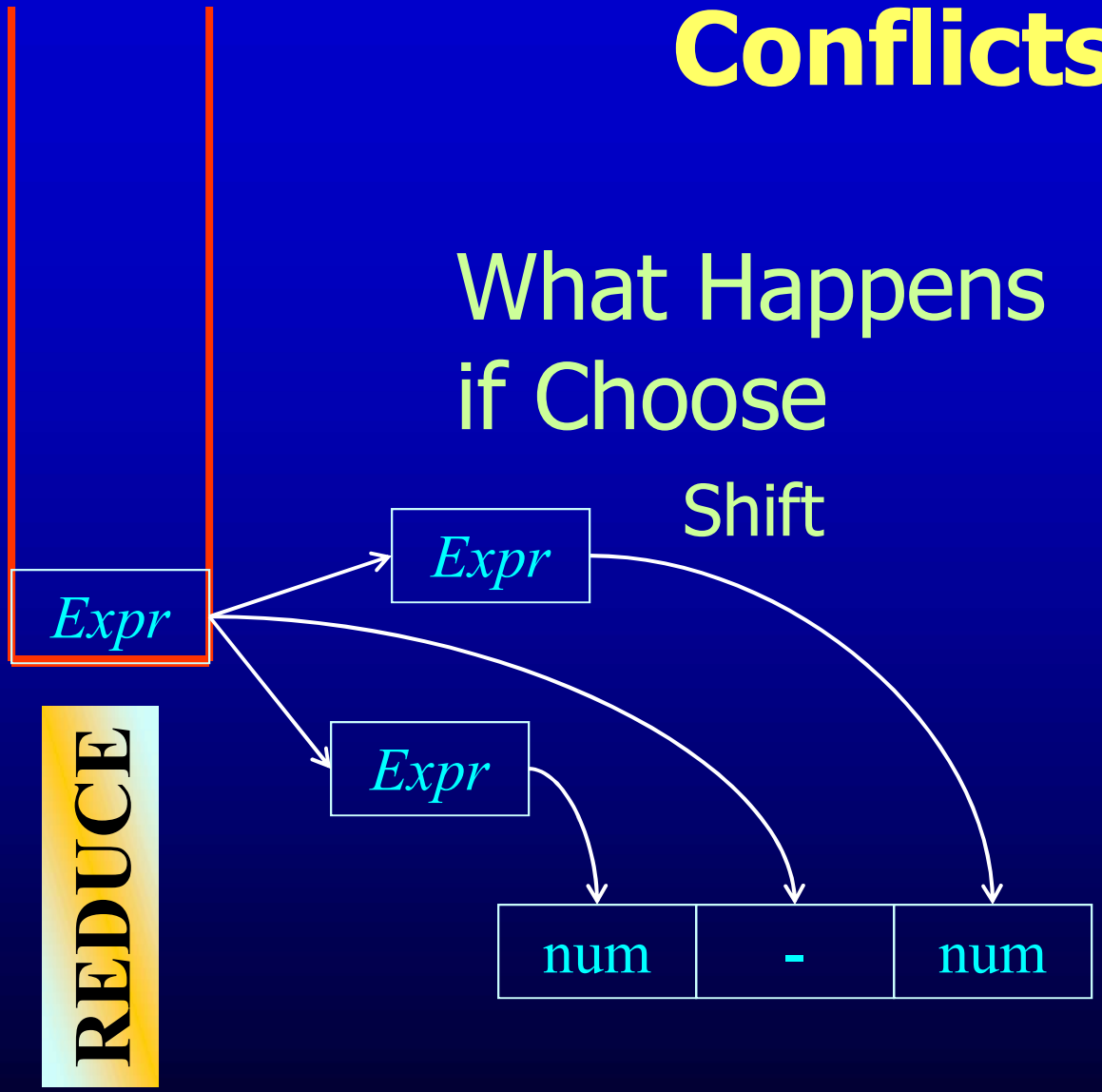
Expr → num

Op → +

Op → -

Op → *

What Happens if Choose



Conflicts

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

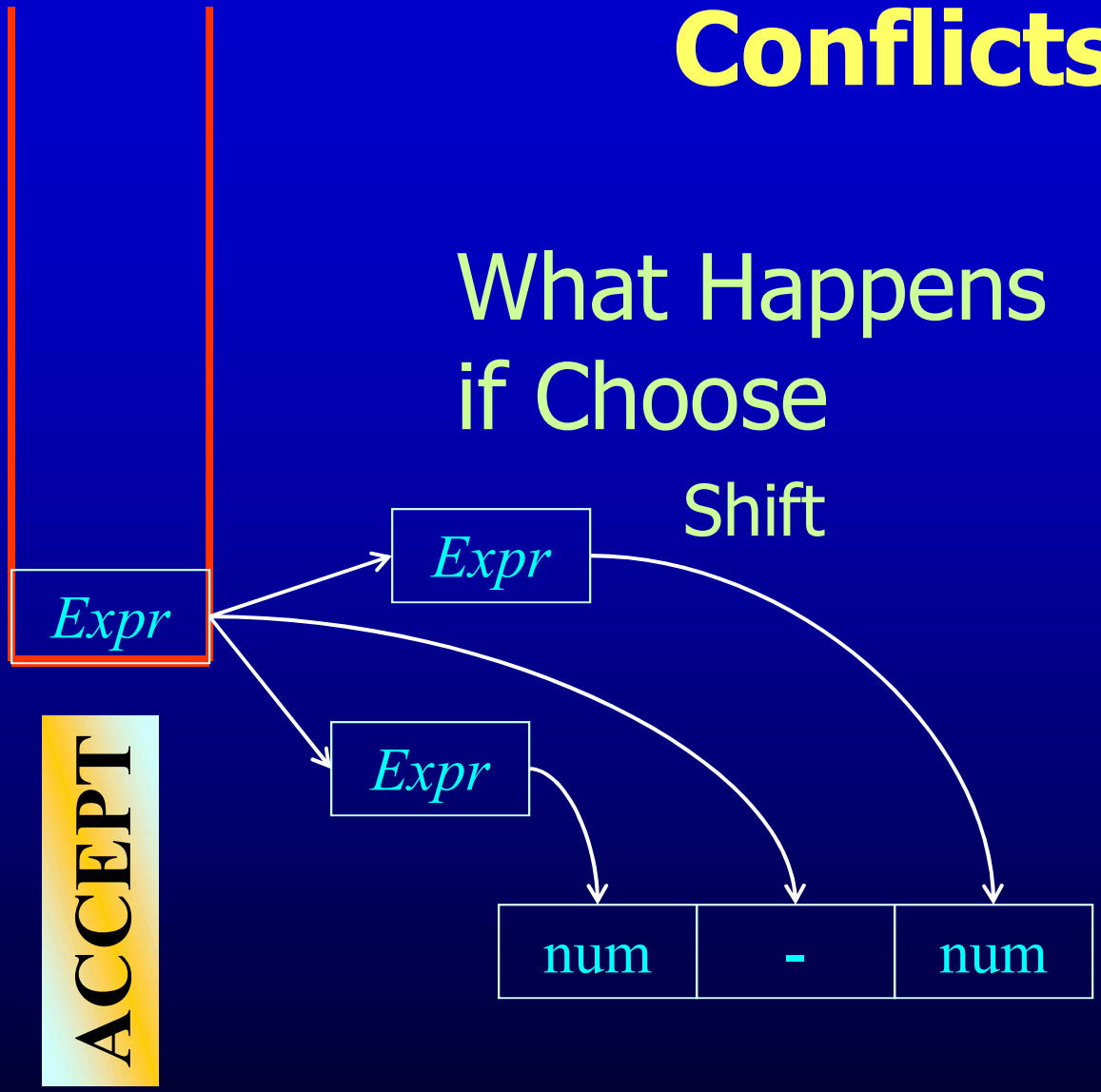
Expr → num

Op → +

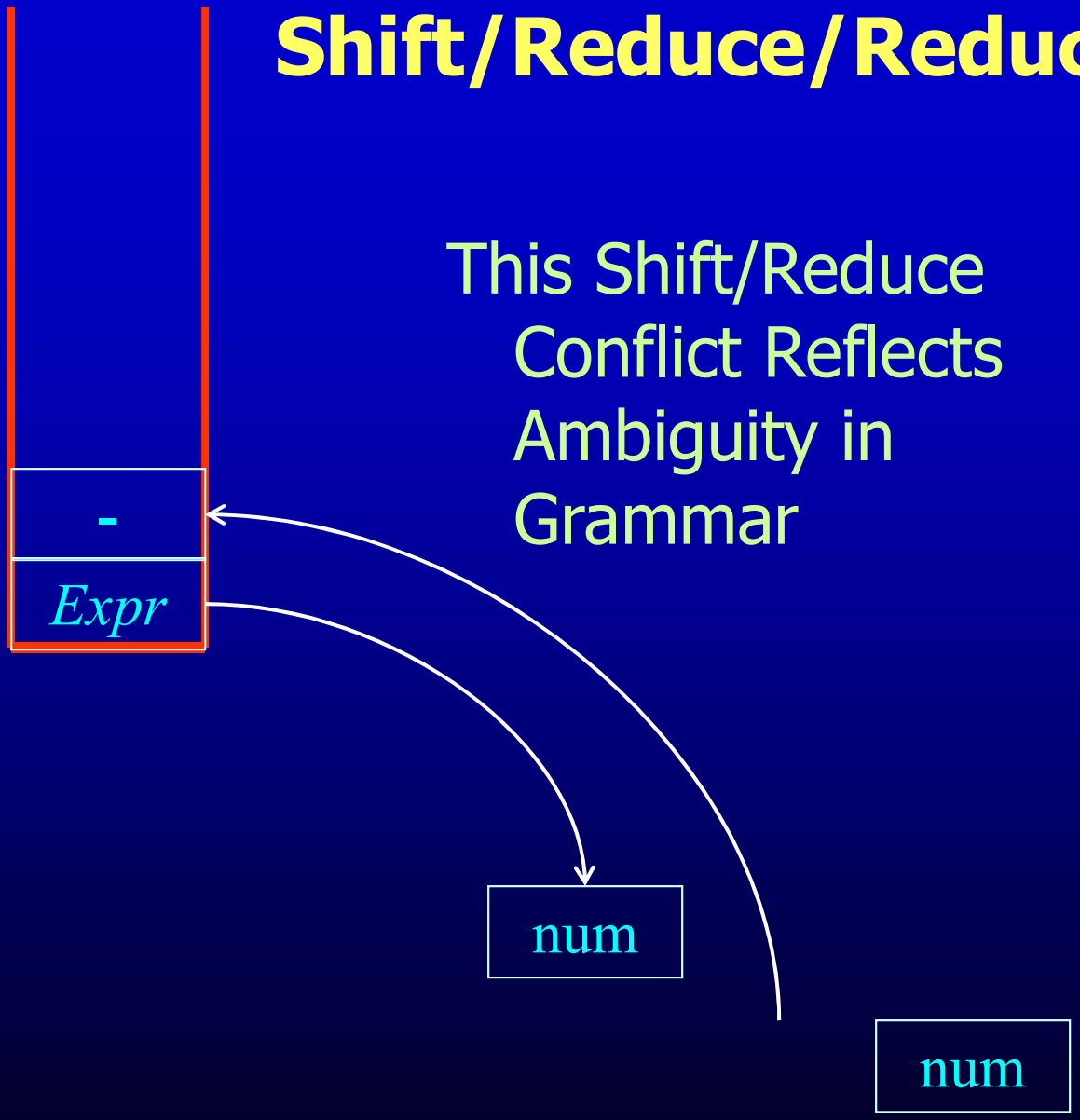
Op → -

Op → *

What Happens if Choose Shift



Shift/Reduce/Reduce Conflict



This Shift/Reduce Conflict Reflects Ambiguity in Grammar

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

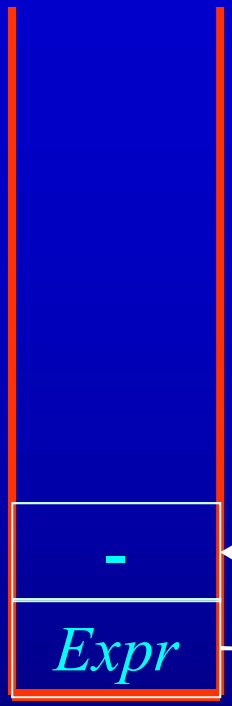
Expr → num

Op → +

Op → -

Op → *

Shift/Reduce/Reduce Conflict



This Shift/Reduce Conflict Reflects Ambiguity in Grammar

Eliminate by Hacking Grammar

Expr → *Expr Op Expr*

~~*Expr* → *Expr* - *Expr*~~

Expr → (*Expr*)

Expr → *Expr* -

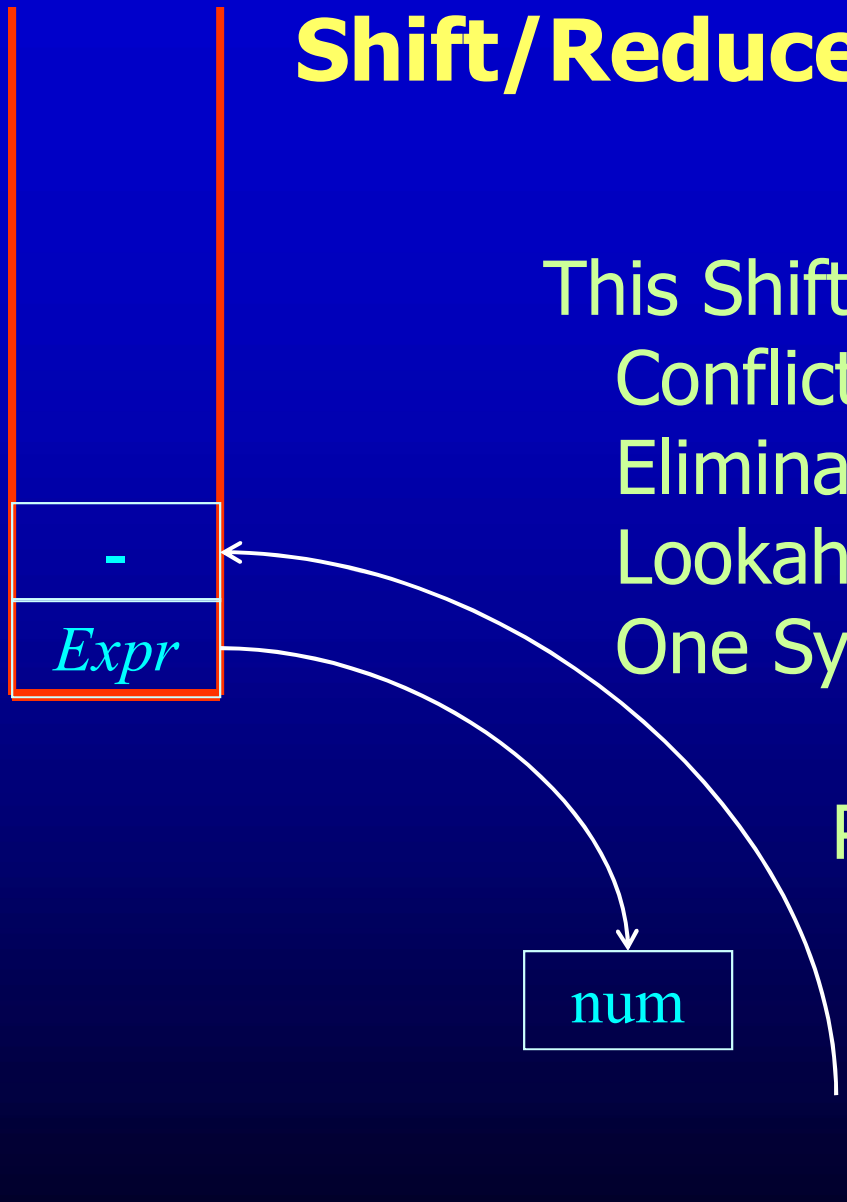
Expr → num

Op → +

Op → -

Op → *

Shift/Reduce/Reduce Conflict



This Shift/Reduce Conflict Can Be Eliminated By Lookahead of One Symbol

Parser Generator Should Handle It

Expr → *Expr Op Expr*

Expr → *Expr* - *Expr*

Expr → (*Expr*)

Expr → *Expr* -

Expr → num

Op → +

Op → -

Op → *

Outline

- Ambiguity and Grammar Design
- Bottom-Up parsing
- Conflicts
- **Constructing a Shift-Reduce Parser**
- Parser and syntax trees

Constructing a Parser

- We will construct version with no lookahead
- Key Decisions
  - Shift or Reduce
  - Which Production to Reduce
- Basic Idea
  - Build a DFA to control shift and reduce actions
  - In effect, convert grammar to pushdown automaton
  - Encode finite state control in parse table

Parser State

- Input Token Sequence ($ for end of input)
- Current State from Finite State Automaton
- Two Stacks
  - State Stack (implements finite state automaton)
  - Symbol Stack (terminals from input and nonterminals from reductions)

Integrating Finite State Control

- Actions
  - Push Symbols and States Onto Stacks
  - Reduce According to a Given Production
  - Accept
- Selected action is a function of
  - Current input symbol
  - Current state of finite state control
- Each action specifies next state
- Implement control using parse table

Parse Tables

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Implements finite state control
- At each step, look up
  - Table[top of state stack] [input symbol]
- Then carry out the action

Parse Table Example

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack Symbol Stack Input

(())

s0

X

Grammar

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

Parser Tables

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Shift to s*n*
  - Push input token into the symbol stack
  - Push s*n* into state stack
  - Advance to next input symbol

Parser Tables

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Reduce (*n*)
  - Pop both stacks as many times as the number of symbols on the RHS of rule *n*
  - Push LHS of rule *n* into symbol stack

Parser Tables

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Reduce (*n*) (continued)
  - Look up Table[top of the state stack][top of symbol stack]
  - Push that state (in goto part of table) onto state stack

Parser Tables

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Accept
  - Stop parsing and report success

Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack Symbol Stack Input Grammar

(())$

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

s0

Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack Symbol Stack Input Grammar

(()$

$S \rightarrow X\$$ (1)

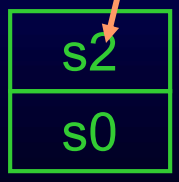
$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack | Symbol Stack | Input | Grammar



(()$

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

Parse Table In Action

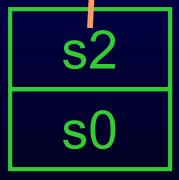
	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack

Symbol Stack

Input

Grammar



())$

$S \rightarrow X\$$ (1)

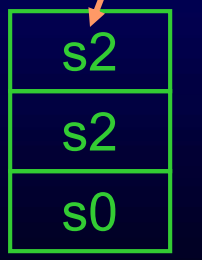
$X \rightarrow (X)$ (2)

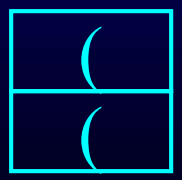
$X \rightarrow (\ )$ (3)

Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack Symbol Stack Input Grammar





())$

$S \rightarrow X\$$ (1)

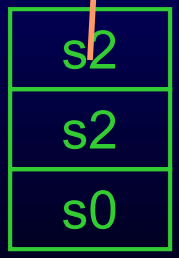
$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

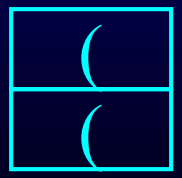
Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack



Symbol Stack



Input

))$

Grammar

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

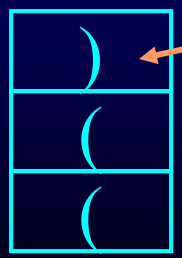
$X \rightarrow (\ )$ (3)

Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack

Symbol Stack



Input

))$

Grammar

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

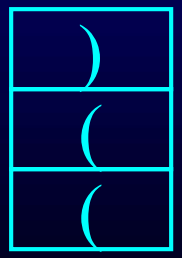
$X \rightarrow (\ )$ (3)

Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack

Symbol Stack



Input

)$

Grammar

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

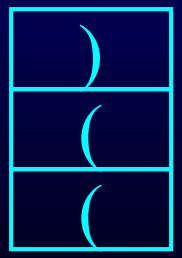
$X \rightarrow (\ \ )$ (3)

Step One: Pop Stacks

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack

Symbol Stack



Input

)$

Grammar

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ \ )$ (3)

Step One: Pop Stacks

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

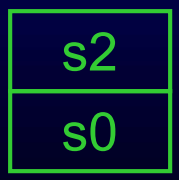
State Stack | Symbol Stack | Input | Grammar

)$

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)



Step Two: Push Nonterminal

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

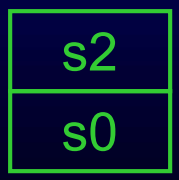
State Stack | Symbol Stack | Input | Grammar

)$

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)



Step Two: Push Nonterminal

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

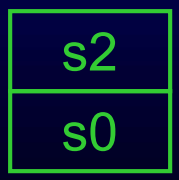
State Stack | Symbol Stack | Input | Grammar

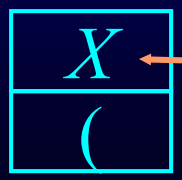
)$

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

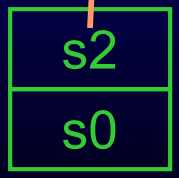




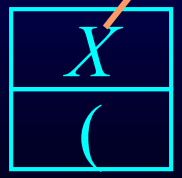
Step Three: Use Goto, Push New State

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack



Symbol Stack



Input

)$

Grammar

$S \rightarrow X\$$ (1)

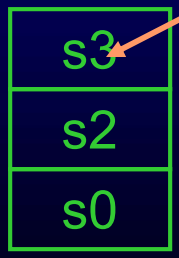
$X \rightarrow (X)$ (2)

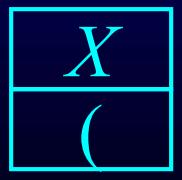
$X \rightarrow (\ )$ (3)

Step Three: Use Goto, Push New State

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack Symbol Stack Input Grammar





)$

$S \rightarrow X\$$ (1)

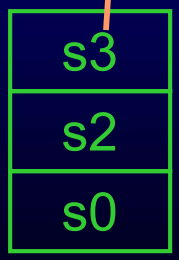
$X \rightarrow (X)$ (2)

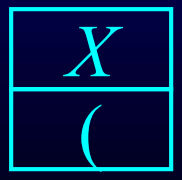
$X \rightarrow (\ )$ (3)

Parse Table In Action

State	ACTION (	ACTION)	ACTION $	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack | Symbol Stack | Input | Grammar





)$

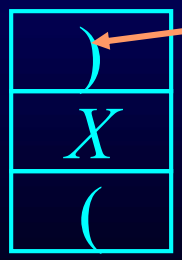
$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	



State Stack Symbol Stack Input Grammar

)$

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

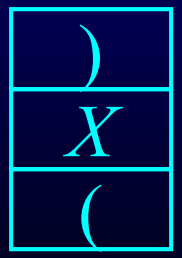
$X \rightarrow (\ )$ (3)

Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack

Symbol Stack



Input

$

Grammar

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

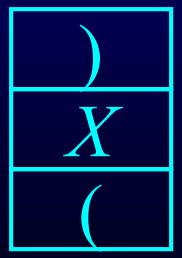
$X \rightarrow (\ )$ (3)

Step One: Pop Stacks

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack

Symbol Stack



Input

$

Grammar

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

Step One: Pop Stacks

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack | Symbol Stack | Input | Grammar

$

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

s0

Step Two: Push Nonterminal

		ACTION		Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack | Symbol Stack | Input | Grammar

$

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

s0

Step Two: Push Nonterminal

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack Symbol Stack Input Grammar

$

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

s0

X

Step Three: Use Goto, Push New State

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack

Symbol Stack

Input

$

Grammar

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

Step Three: Use Goto, Push New State

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

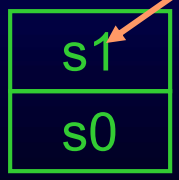
State Stack | Symbol Stack | Input | Grammar

$

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)



Accept the String!

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

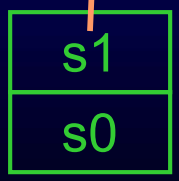
State Stack Symbol Stack Input Grammar

$

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)



Key Concepts

- Pushdown automaton for parsing
  - Stack, Finite state control
  - Parse actions: shift, reduce, accept
- Parse table for controlling parser actions
  - Indexed by parser state and input symbol
  - Entries specify action and next state
  - Use state stack to help control
- Parse tree construction
  - Reads input from left to right
  - Bottom-up construction of parse tree

Outline

- Ambiguity and Grammar Design
- Bottom-Up parsing
- Conflicts
- Constructing a Shift-Reduce Parser
- **Parser and syntax trees**

Parser

- Converts program into a parse tree
- Can be written by hand
- Or produced automatically by parser generator
  - Accepts a grammar as input
  - Produces a parser as output
- Practical problem
  - Parse tree for hacked grammar is complicated
  - Would like to start with more intuitive parse tree

Solution

- Abstract versus Concrete Syntax
  - Abstract syntax corresponds to "intuitive" way of thinking of structure of program
    - Omits details like superfluous keywords that are there to make the language unambiguous
    - Abstract syntax may be ambiguous
  - Concrete Syntax corresponds to full grammar used to parse the language
- Parsers are often written to produce abstract syntax trees.