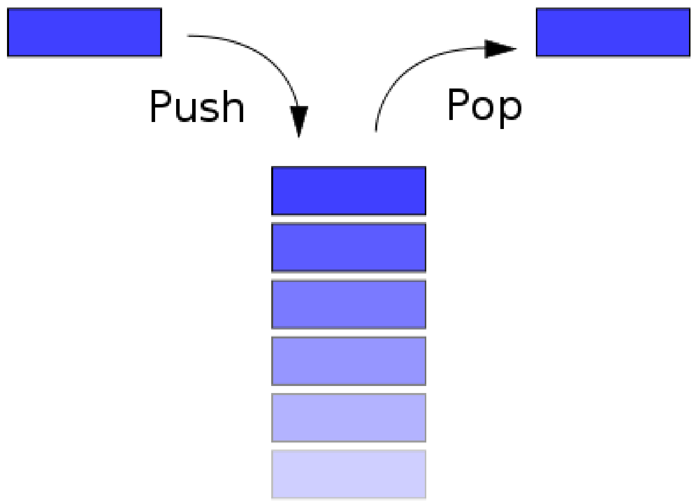
Stacks and queues

http://www.flickr.com/photos/mac-ash/4534203626/

http://www.flickr.com/photos/thowi/182298390/

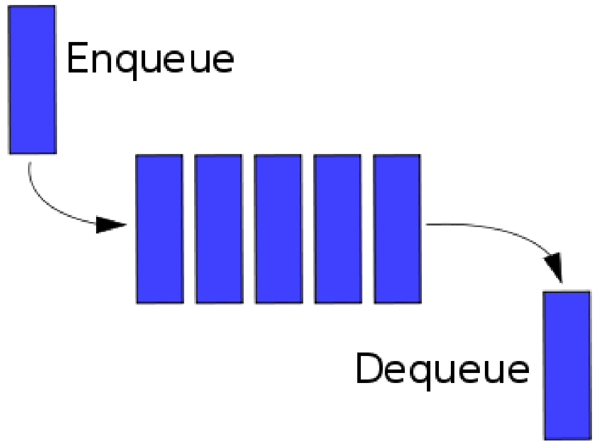
Overview

- Terminology
  - Abstract Data Types (ADT)
  - Data structures



- Stack ADT
  - Last-in first-out (LIFO)

- Queue ADT
  - First-in first-out (FIFO)



ADT vs. data structure

- Abstract Data Type (ADT)
  - A collection of data and a set of operations on that data
  - Why is it "abstract"?
    - Doesn't specify implementation details
    - Just describes what the type can do
    - You can use without knowing internal workings
  - e.g. Stack, Queue, SymbolTable, List, SortedList
- Data structure
  - How the data type is implemented in software
  - e.g. array, linked list, linked graph

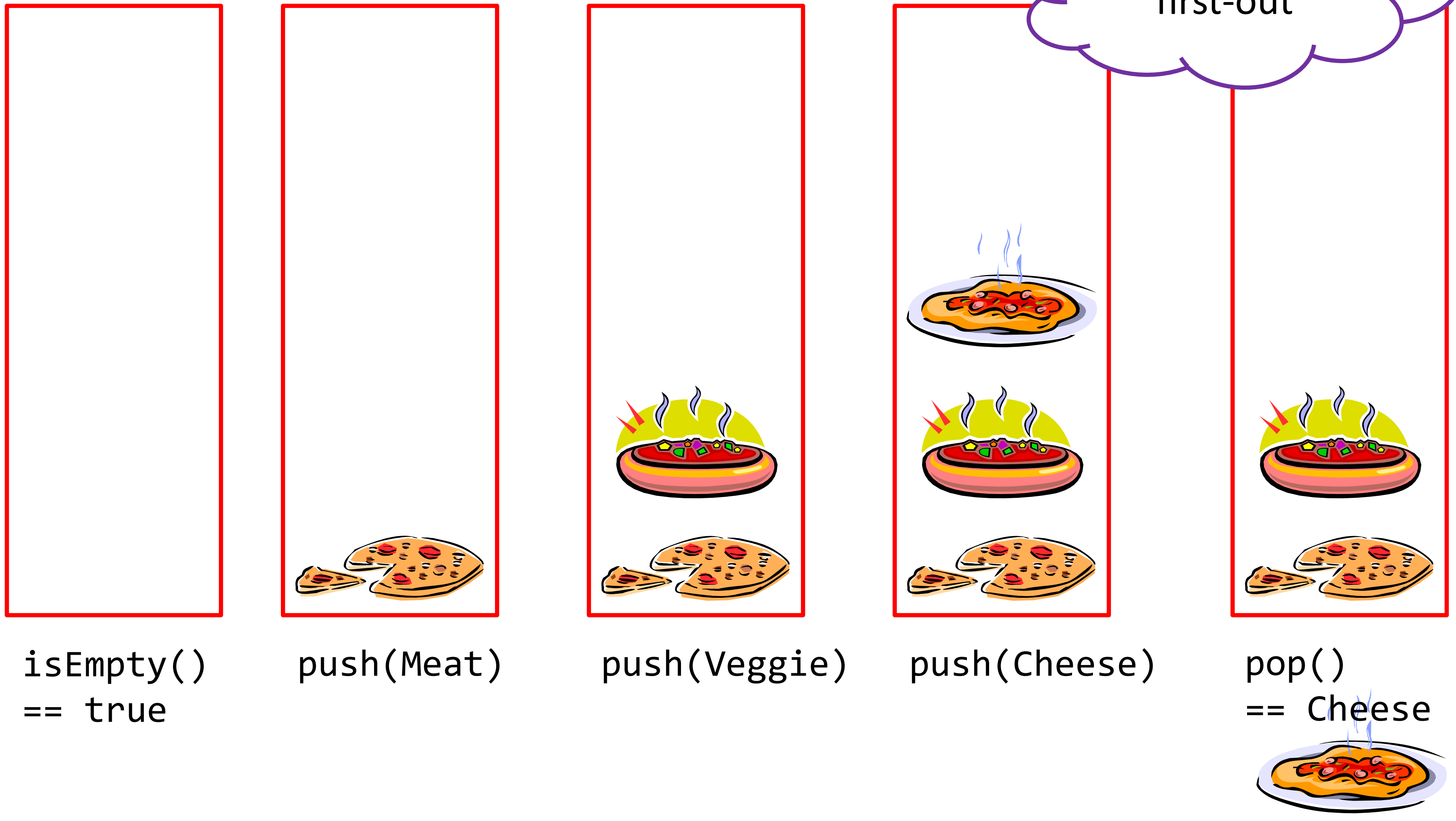
Collections

- Collection: A common data type for storing data
  - Allow users to insert item
  - Allow users to remove item, but *which one*?
  - Allow users to see if the collection is empty
- List
  - Remove at specified position
  - e.g. pile of resumes in order of GPA, Java's `ArrayList` class
- Stack
  - Remove the most recently added = LIFO (Last-In First-Out)
  - e.g. trays in the cafeteria
- Queue
  - Remove the least recently added = FIFO (First-In First-Out)
  - e.g. line at the grocery store
- Symbol Table
  - Remove item with a given key
  - e.g. phone book: maps a name to a phone number

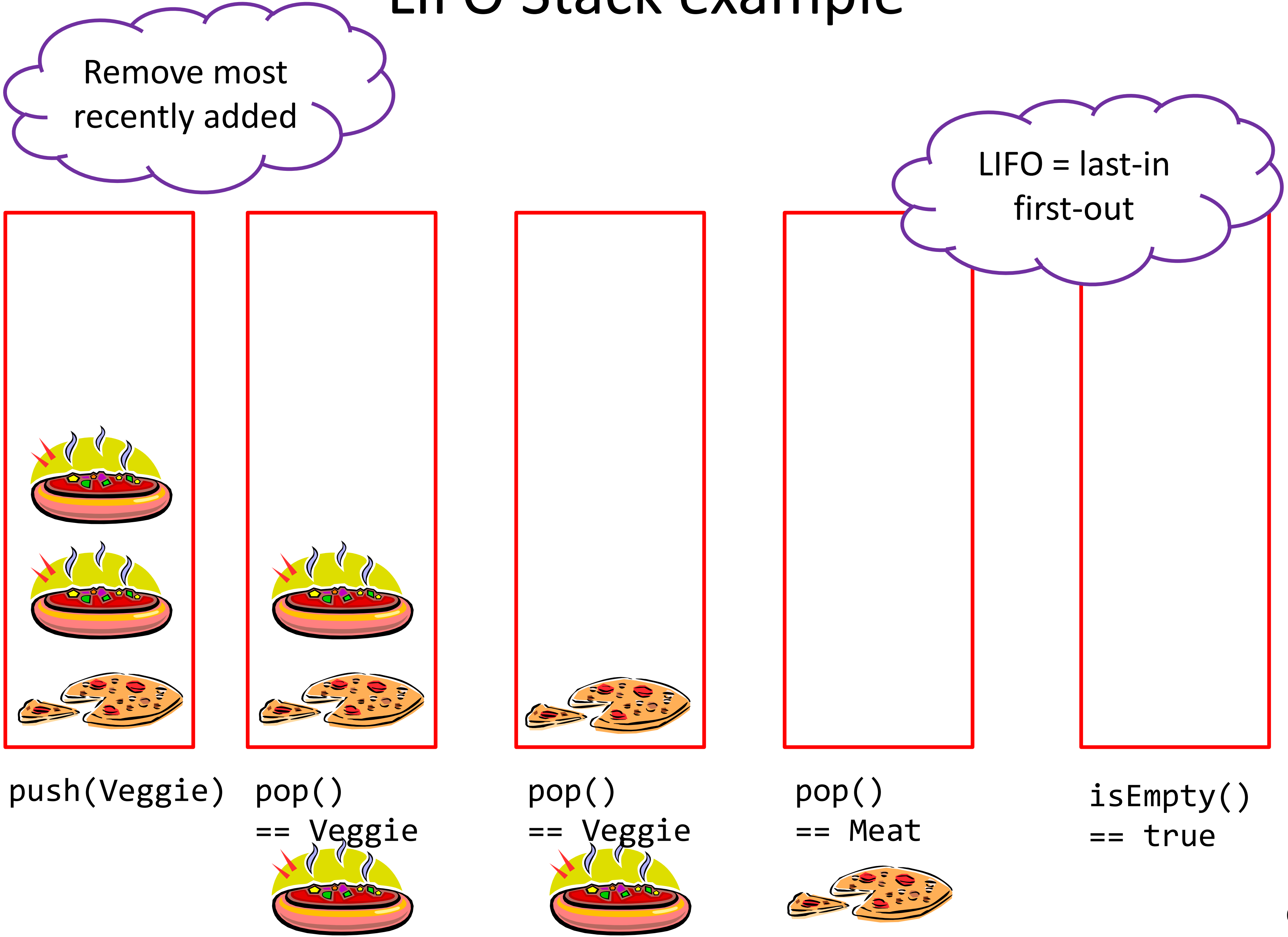
LIFO Stack example

Remove most recently added

LIFO = last-in first-out



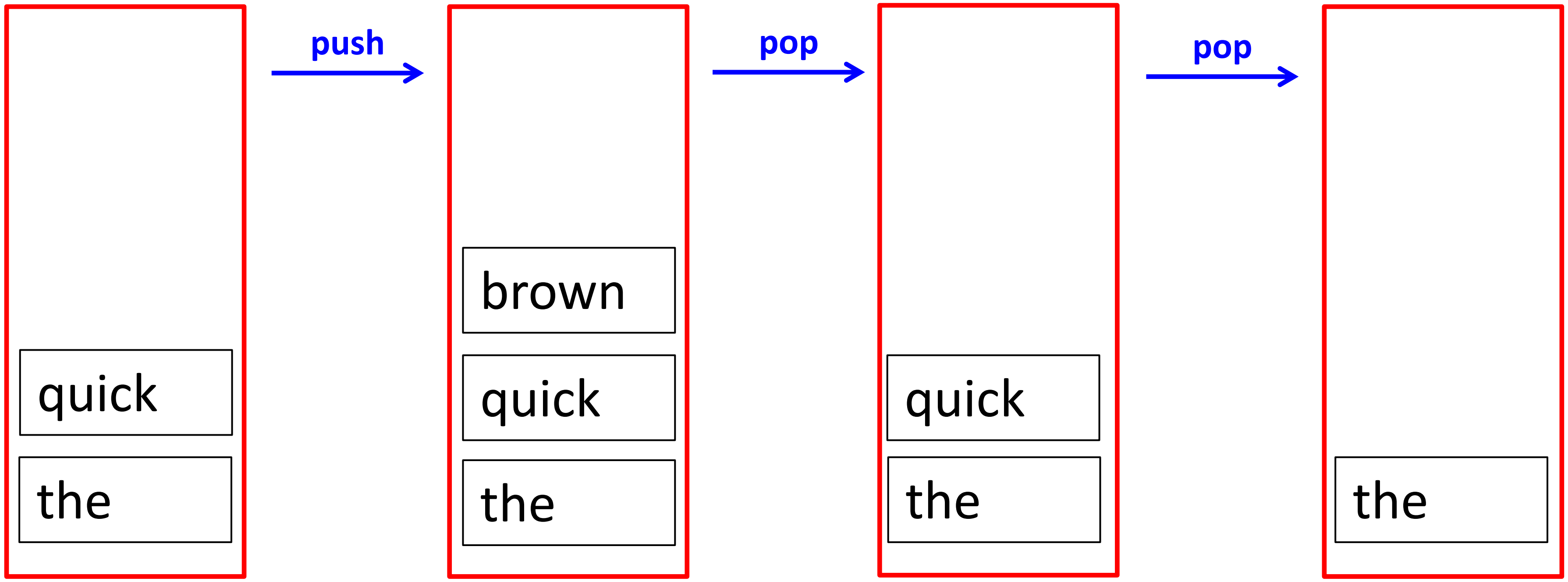
LIFO Stack example



```
push(Veggie)

pop()
== Veggie

pop()
== Veggie

pop()
== Meat

isEmpty()
== true
```

LIFO Stack API

```
public class StackOfStrings
----------------------------------------------------------------------
            StackOfStrings() // Construct a new stack
       void push(String s)   // Add a new string to the queue
     String pop()            // Remove the most recently added string
    boolean isEmpty()        // Check if the queue is empty
```



LIFO Stack example 1

- Goal: Reverse all the words in a file
  - "glory is fleeting but obscurity is forever" →
  - "forever is obscurity but fleeting is glory"
- Approach:
  - Use a Stack ADT as implemented by `StackOfStrings`
  - While more text available from standard input:
    - Read a word, push on stack
  - While stack is not empty:
    - Pop from stack, output word

Reverse words on standard input

```
public class ReverseWords
{

    public static void main(String [] args)
    {
        StackOfStrings stack = new StackOfStrings();


        while (!StdIn.isEmpty())
            stack.push(StdIn.readString());


        while (!stack.isEmpty())
            System.out.print(stack.pop() + " ");

        System.out.println();
    }

}
```

Create an instance of a stack ADT. Notice we don't specify a size, the class promises to handle any size.

Next word goes on top of the stack that contains all the previously read in words.

Start peeling off words starting with the last one pushed on top of the stack.

LIFO Stack example 2

- Goal: Check for balanced ()'s and []'s

[((a + b) * d) + (e * f)] → balanced

[([a + b] * d) + (e * f)] → balanced

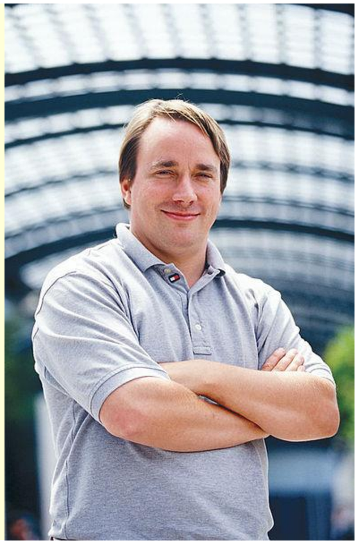
[((a + b) * d) + (e * f) → unbalanced

(a + b) * d) + (e * f) → unbalanced

[((a + b) * d) + (e * f)) → unbalanced

"I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. **Bad programmers worry about the code. Good programmers worry about data structures** and their relationships."

-Linus Torvalds, creator of Linux



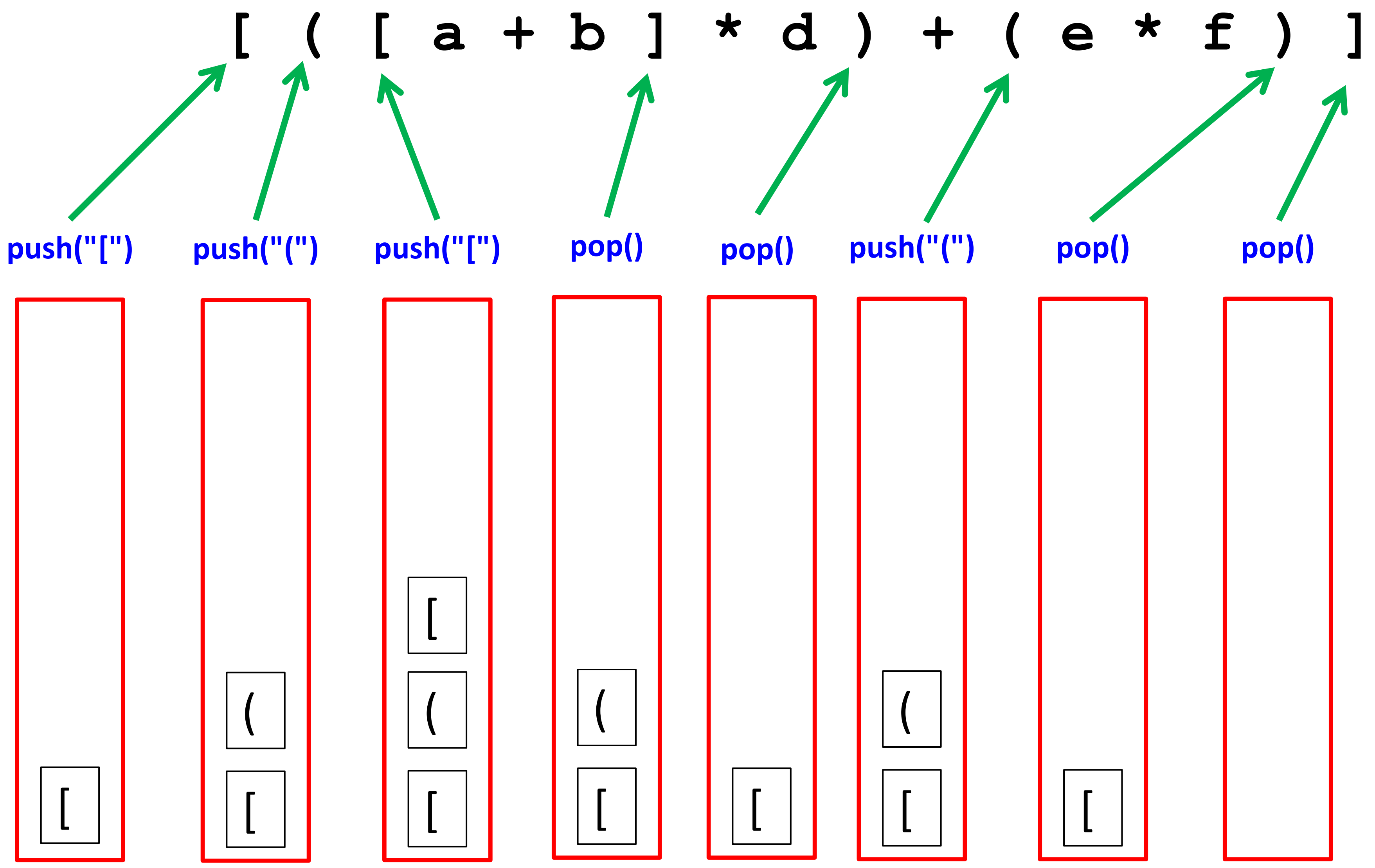
LIFO Stack example 2

- Goal: Check for balanced ()'s and []'s

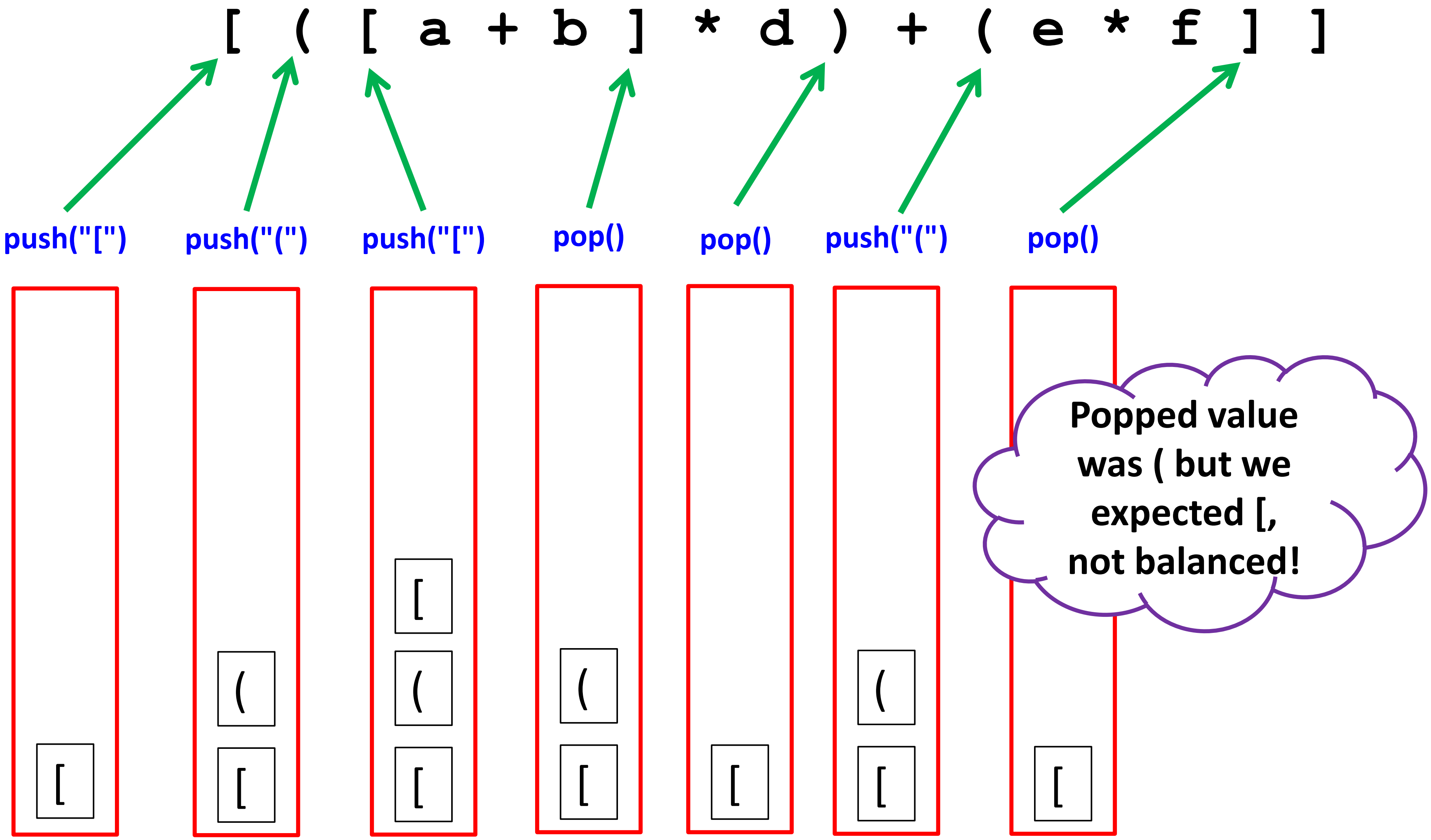
[((a + b) * d) + (e * f)]	→ balanced
[([a + b] * d) + (e * f)]	→ balanced
[((a + b) * d) + (e * f)	→ unbalanced
(a + b) * d) + (e * f)	→ unbalanced
[((a + b) * d) + (e * f))	→ unbalanced

- Approach:
  - Use a Stack ADT as implemented by `StackOfStrings`
  - If token is (or [then push onto stack
  - If token is) then pop stack and make sure popped value is (
  - If token is] then pop stack and make sure popped value is [
  - Any other token, ignore

Balanced, success

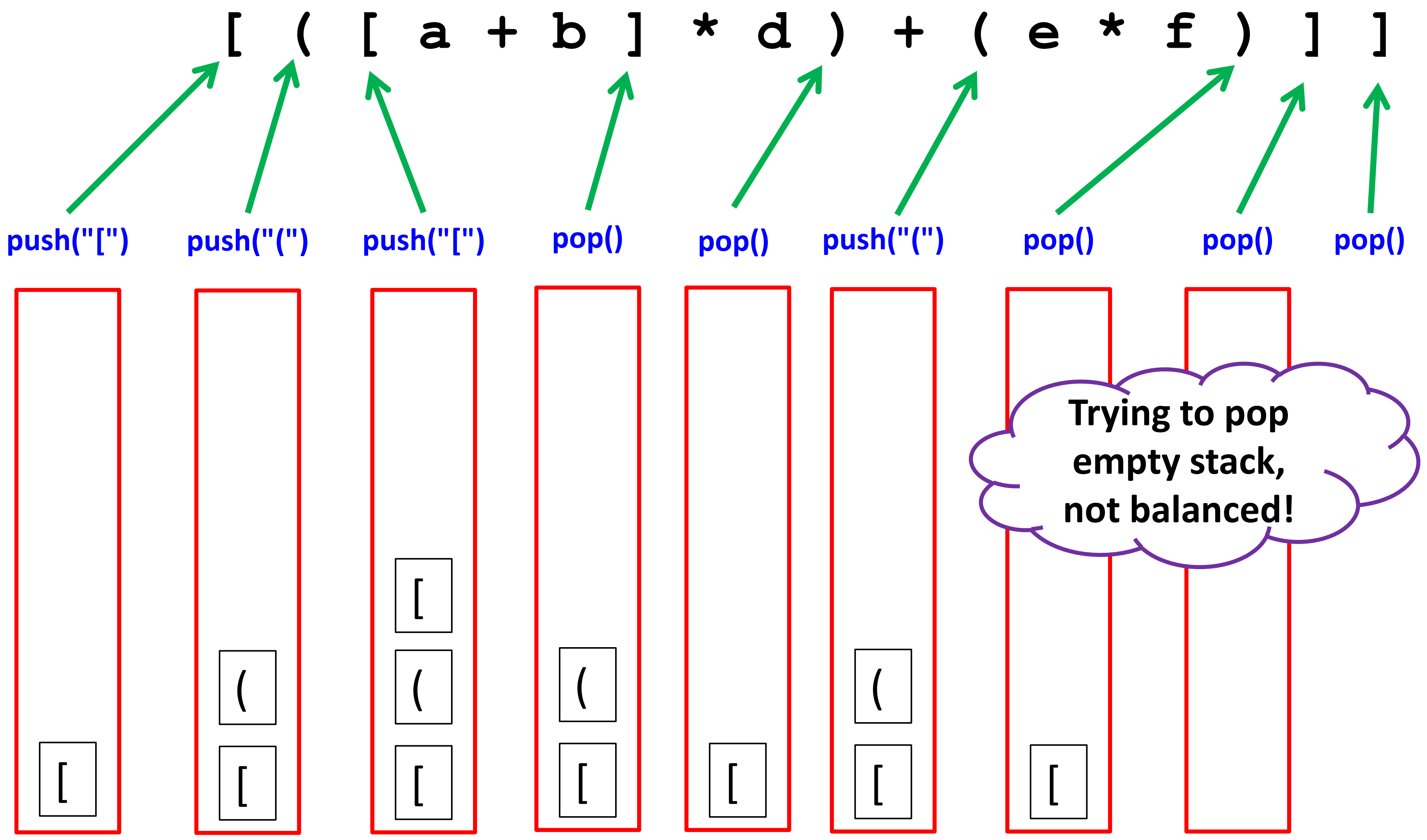


Balanced, failure 1



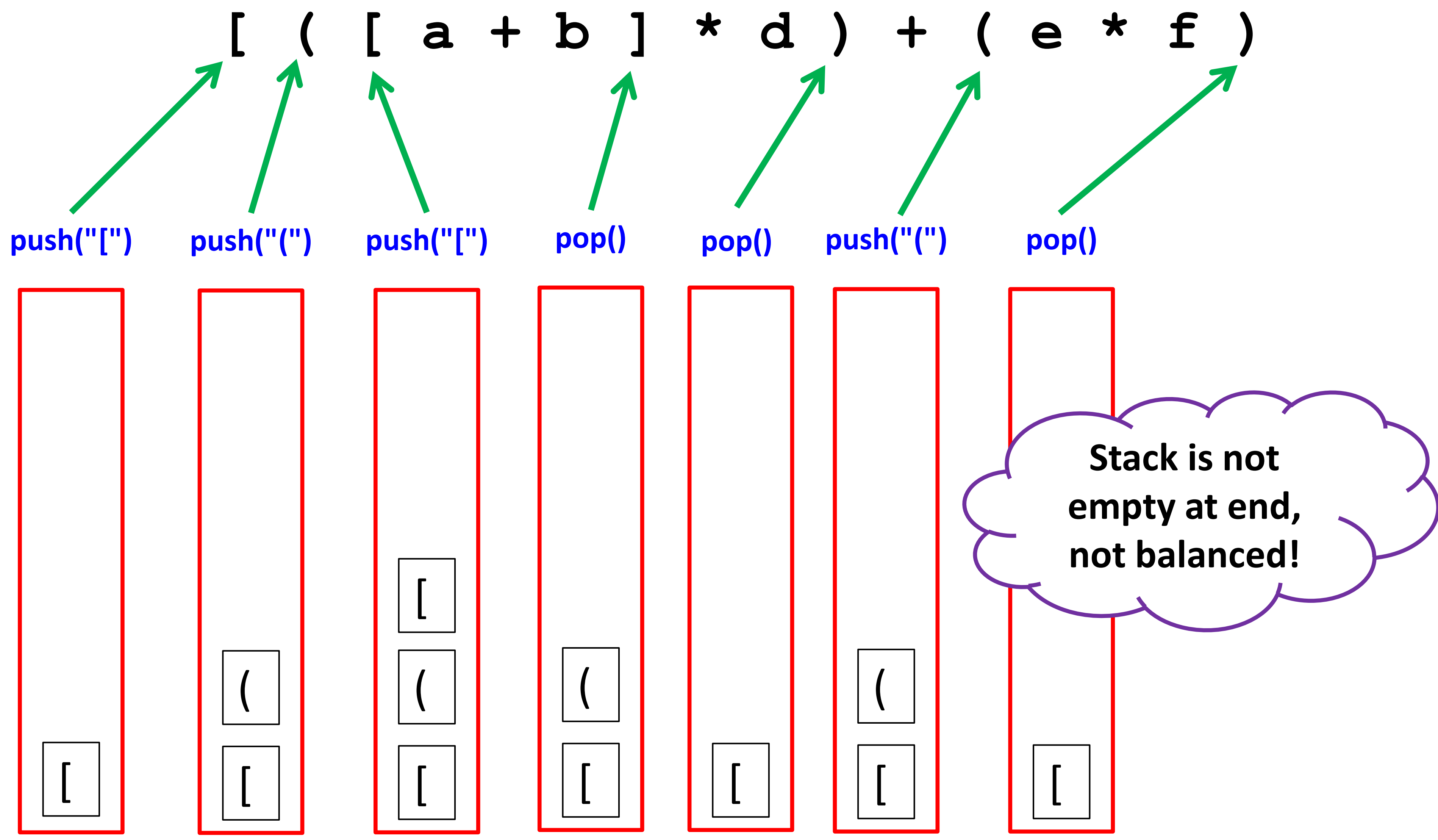
[([a + b] * d) + (e * f]]
push("[")
push("(")
push("[")
pop()
pop()
push("(")
pop()
[
(
[
(
[
[
(
[
[
(
[
[
Popped value was (but we expected [, not balanced!

Balanced, failure 2



[([a + b] * d) + (e * f)]]
push("[")
push("(")
push("[")
pop()
pop()
push("(")
pop()
pop()
pop()
[
(
[
[
(
[
(
[
[
(
[
[
Trying to pop empty stack, not balanced!

Balanced, failure 3


[([a + b] * d) + (e * f)
push("[")
push("(")
push("[")
pop()
pop()
push("(")
pop()
[
([
[([
([
[
([
[
Stack is not empty at end, not balanced!

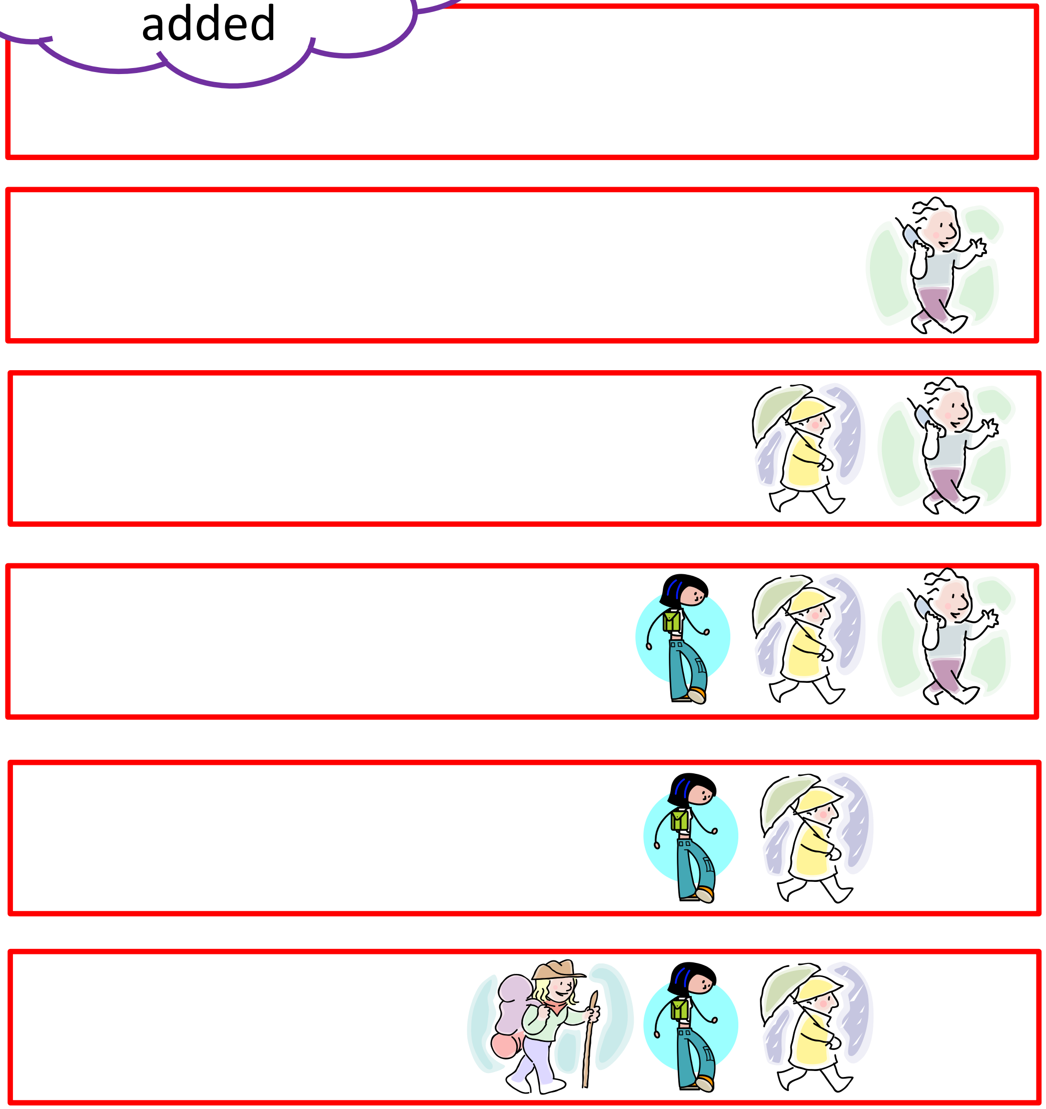
```java
public static void main(String [] args)
{
    StackOfStrings stack = new StackOfStrings();

    while (!StdIn.isEmpty())
    {
        String token = StdIn.readString();
        if ((token.equals("(")) || (token.equals("[")))
        {
            stack.push(token);
        }
        else if (token.equals(")"))
        {
            if ((stack.isEmpty()) || (!stack.pop().equals("(")))
            {
                System.out.println("Not balanced");
                return;
            }
        }
        else if (token.compareTo("]") == 0)
        {
            if ((stack.isEmpty()) || (!stack.pop().equals("[")))
            {
                System.out.println("Not balanced");
                return;
            }
       }
    }

    if (stack.isEmpty())
        System.out.println("Balanced");
    else
        System.out.println("Not balanced");
}
```

Balanced.java

FIFO Queue example

Remove least recently added

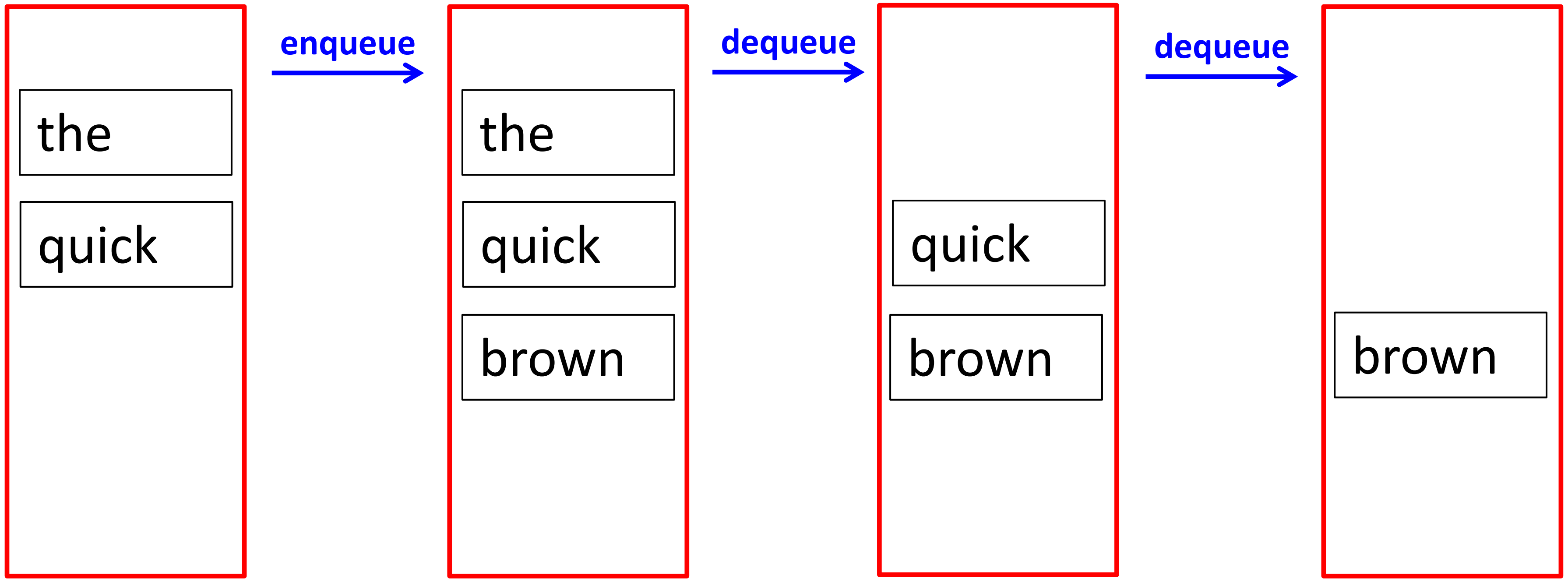
FIFO = first-in first-out



```
isEmpty() == true

enqueue(Abe)

enqueue(Bill)

enqueue(Carol)

dequeue() == Abe

enqueue(Diana)
```

FIFO Queue API

```
public class QueueOfStrings
----------------------------------------------------------
            QueueOfStrings()   // Construct a new queue
       void enqueue(String s)  // Add a new string to the queue
     String dequeue()          // Remove the least recently added string
    boolean isEmpty()          // Check if the queue is empty
```



FIFO Queue example

- Goal: Parental spelling obfuscation aid
  - "After the kids go to sleep let's have some…"
  - Parent types "cookies" into computer
  - Computer spells out each letter, "c--o--o--k--i--e--s"
    - Pausing one second between letters
- Approach:
  - Use a Queue ADT as implemented by `QueueOfStrings`
  - Queue a each new letter as it is typed
  - Delay 1s before dequeue'ing
    - Display letter
    - Play WAV audio file

```java
public static void main(String[] args)
{
    StdDraw.setFont(new Font("SansSerif", Font.BOLD, 120));
    int delay = 0;
    QueueOfStrings queue = new QueueOfStrings();

    while (true)
    {
        if (StdDraw.hasNextKeyTyped())
        {
            char key = StdDraw.nextKeyTyped();
            if ((key >= 'a') && (key <= 'z'))
                queue.enqueue("" + key);
        }
        StdDraw.show(100);
        delay += 100;

        if (delay >= 1000)
        {
            delay = 0;
            StdDraw.clear();
            if (!queue.isEmpty())
            {
                String letter = queue.dequeue();
                StdDraw.text(.5, .5, letter);
                StdAudio.play(letter + ".wav");
            }
        }
    }
}
```

Speller.java

Create an instance of a Queue data type. Notice we don't specify a size, class promises to handle any size.

Latest character goes at the back of the line. All other character have to play first.

Play the character that has been waiting the longest in the queue.

Summary

- Abstract Data Types (ADTs)
  - A collection of data and operations on that data
  - LIFO Stack
    - Push and pop items, always pops the last thing pushed
    - Examples: reversing words in a sentence, check for balanced parameters
  - FIFO Queue
    - Enqueue and dequeue items
    - Always dequeue the thing that has been waiting the longest
    - Examples: tracking and eventually servicing asynchronous events (keys typed by parent)
- Data structures
  - Implementation of an ADT (there may be many ways!)
  - e.g. using a normal array, using an `ArrayList`, ...