Collections, Part Three

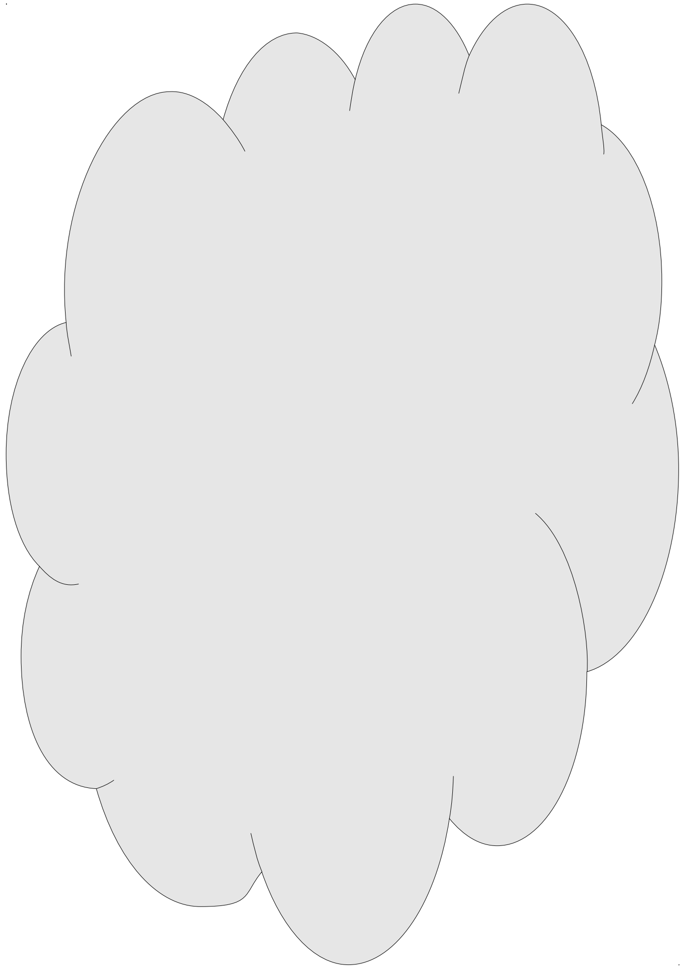
Announcements

- Assignment 1 due right now.
- Assignment 2 out, due Monday, April 23.
    - Play around with some **awesome** applications of collections classes.
    - Teach the computer to write!
    - YEAH hours next Wednesday, April 18 from 4:15 – 5:45.

Set

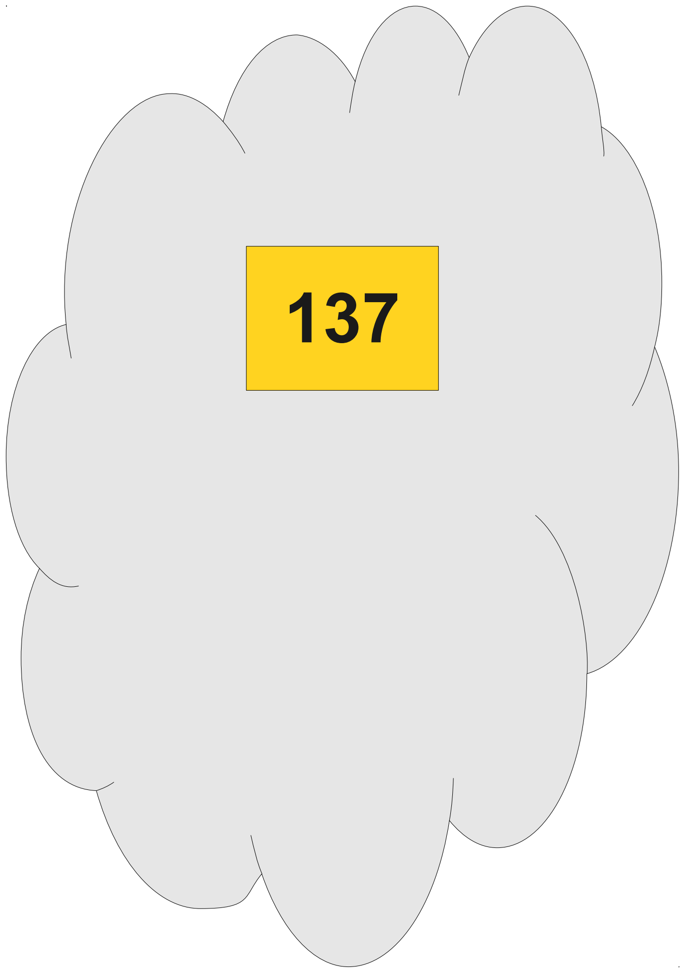
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



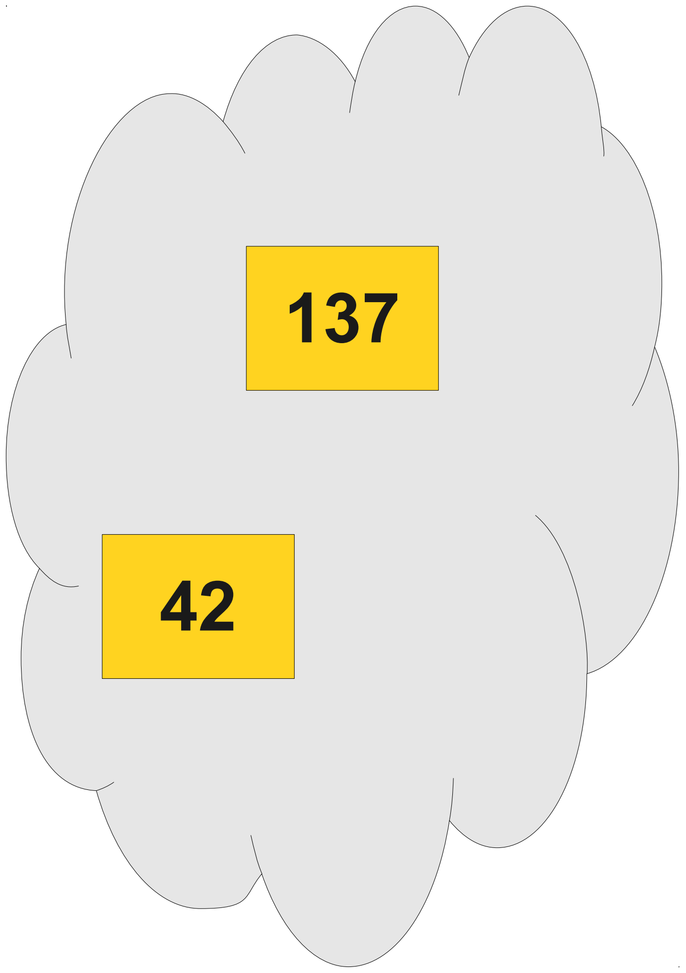
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



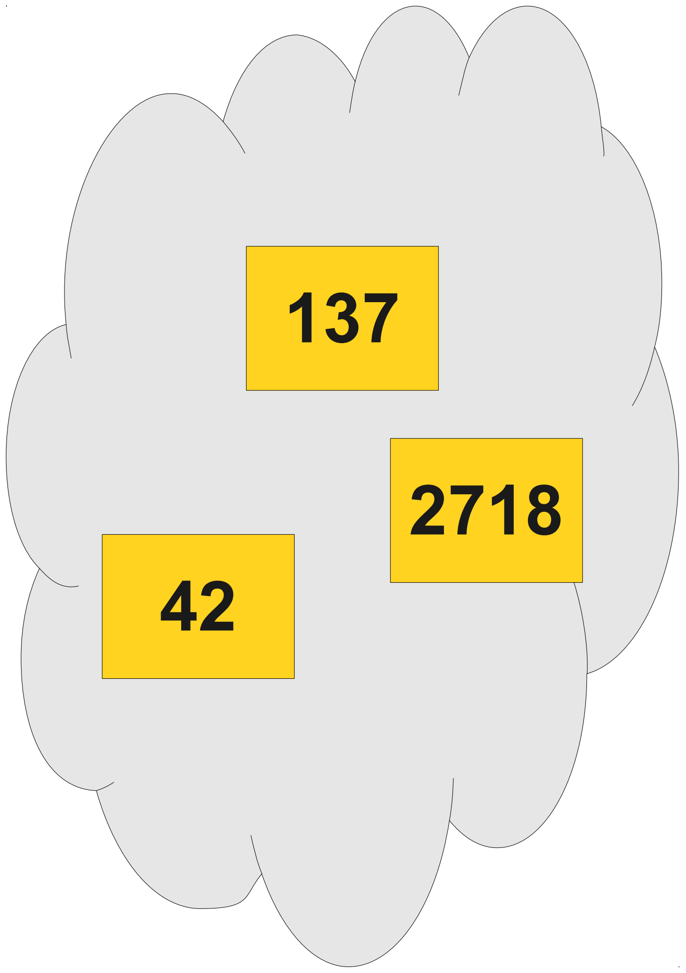
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



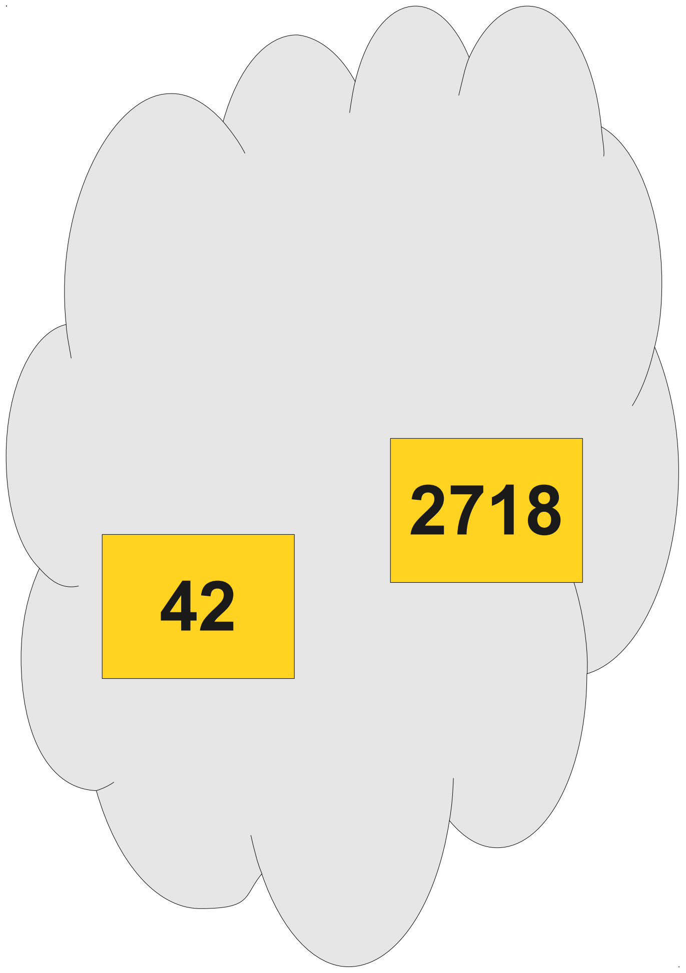
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



Operations on Sets

- You can add a value to a set by writing

  set += ***value***;

- You can remove a value from a set by writing

  set -= ***value***;

- You can check if a value exists by writing

  set`.contains(`***value***`)`

- Many more operations available (union, intersection, difference, subset, etc.), so be sure to check the documentation.

Lexicon

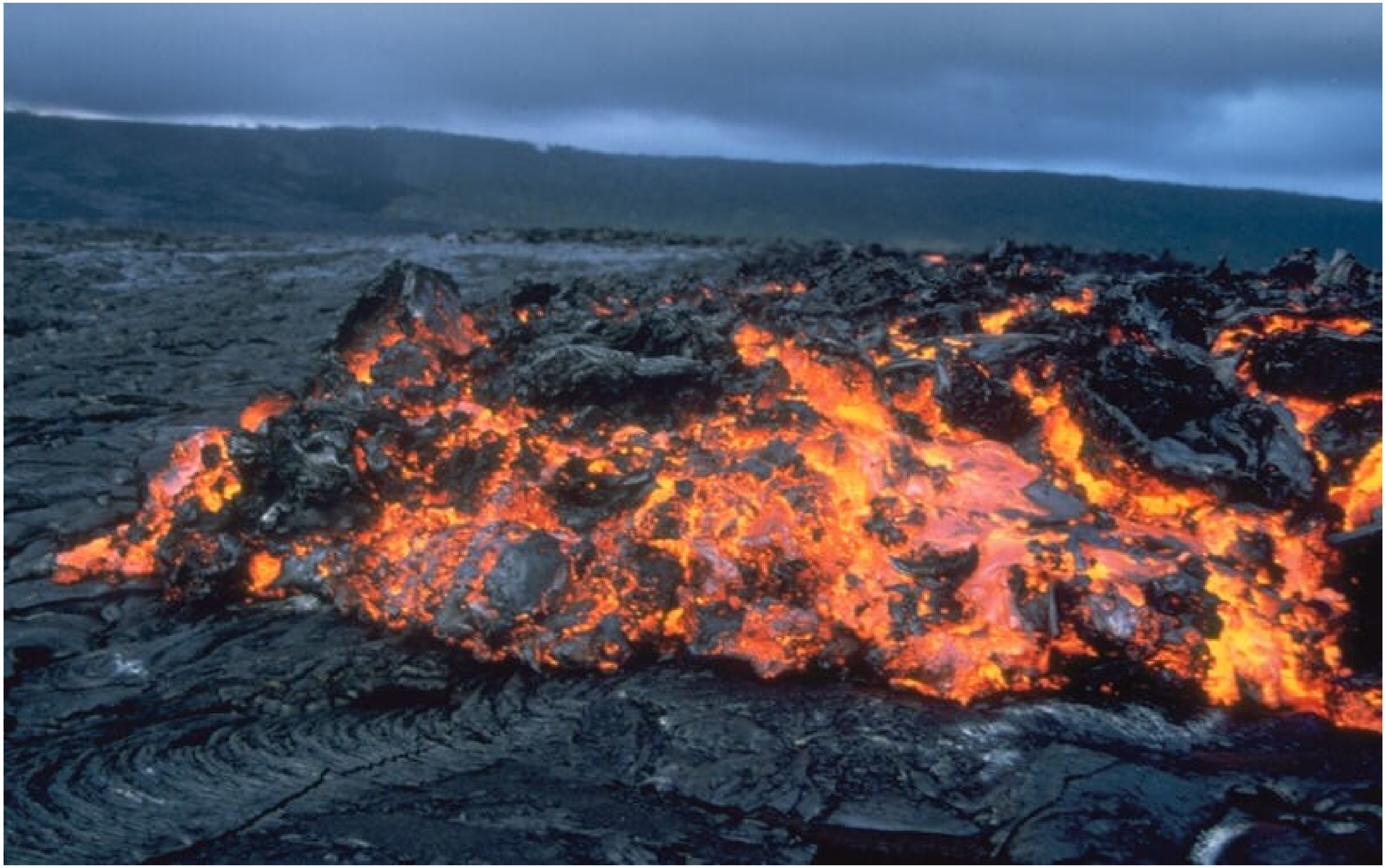
`Lexicon`

- The `Lexicon` is a collection of words in some language.
- Similar to a `Set`, but with additional operations appropriate to word lists.
    - e.g. Checking whether a string is a prefix of some word.

Tautonyms

- A **tautonym** is a word formed by repeating the same string twice.
  - For example: murmur, couscous, papa, etc.
- What tautonyms exist in English?

Some Aa



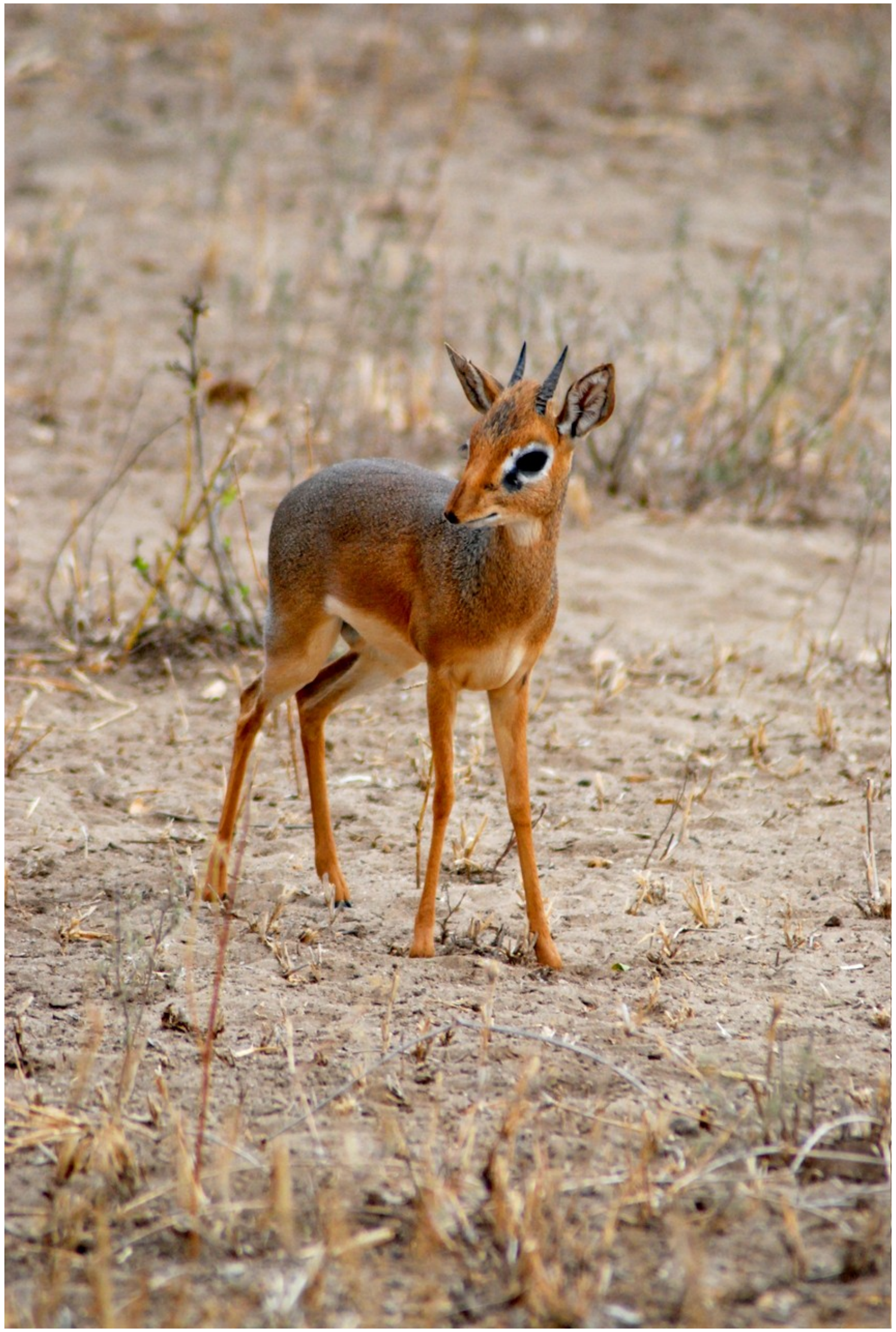
http://upload.wikimedia.org/wikipedia/commons/f/f1/Aa_large.jpg

One Bulbul

http://travel.paintedstork.com/blog/image/yellow_browed_bulbul.jpg

More than One Caracara

http://www.greglasley.net/images/CO/Crested-Caracara-F3.jpg



A dikdik

http://upload.wikimedia.org/wikipedia/commons/b/b6/Dik-dik_%28male%29_-Tarangire_National_Park_-Tanzania.jpg

Anagrams

- Two phrases are **anagrams** of one another if they have the same letters, but in a different order.
- Examples:
    - Stanford University → A Trusty Finned Visor
    - Keith Schwarz → Zither Whacks
    - Zachary Galant → Lazy Hangar Cat

Anagram Clusters

- An **anagram cluster** is a set of words that are all anagrams of one another.

  stop ↔ tops ↔ pots ↔ spot ↔ opts

- What is the largest anagram cluster in the English language?

TokenScanner

- The **`TokenScanner`** class can be used to break apart a string into smaller pieces.
- Construct a `TokenScanner` to piece apart a string as follows:

```
TokenScanner scanner(str);
```

- Configure options (ignore comments, ignore spaces, add operators, etc.)
- Use the following loop to read tokens one at a time:

```
while (scanner.hasMoreTokens()) {
    string token = scanner.nextToken();
    /* … process token … */
}
```

- Check the documentation for more details; there are some really cool tricks you can do with the `TokenScanner`!

Application: Evaluating Expressions

Evaluating Expressions

- Evaluating expressions is much trickier than it might seem due to issues of precedence.
  - 1 + 3 * 5 – 7 = 9
  - 4 / 2 + 2 = 2
  - 17 % 6 % 3 = 2
- How do we evaluate an expression?

Evaluating Expressions

- Two separate concerns in evaluating expressions:
  - **Scanning** the string and breaking it apart into its constituent components (*tokens*).
  - **Parsing** the tokens to determine what expression is being encoded.
- The scanning job is taken care of for us by the `TokenScanner` class.
- How do we handle parsing?

The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

Operands

The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

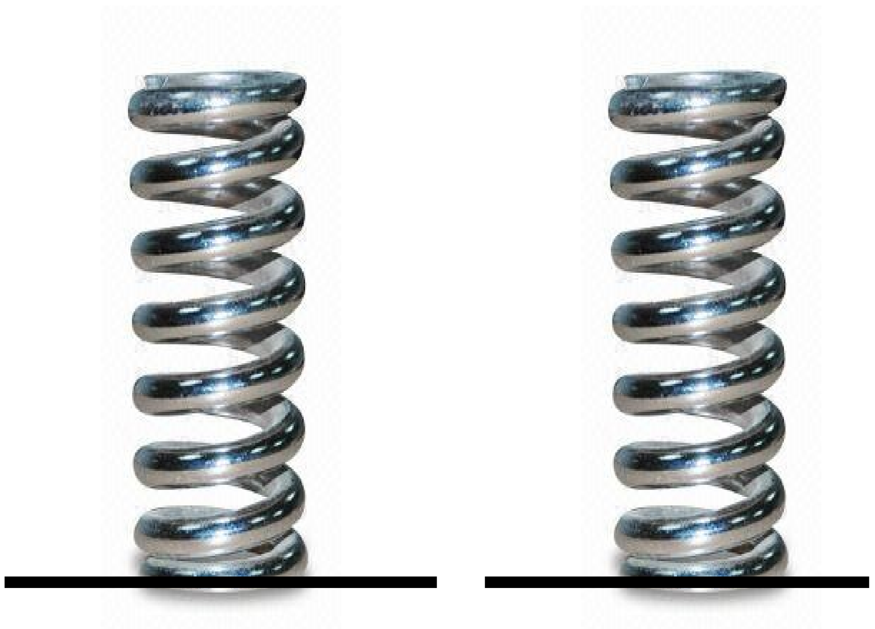
Operands

Operators

The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

2	+	3	*	5	-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

2	+	3	*	5	-	6	/	2

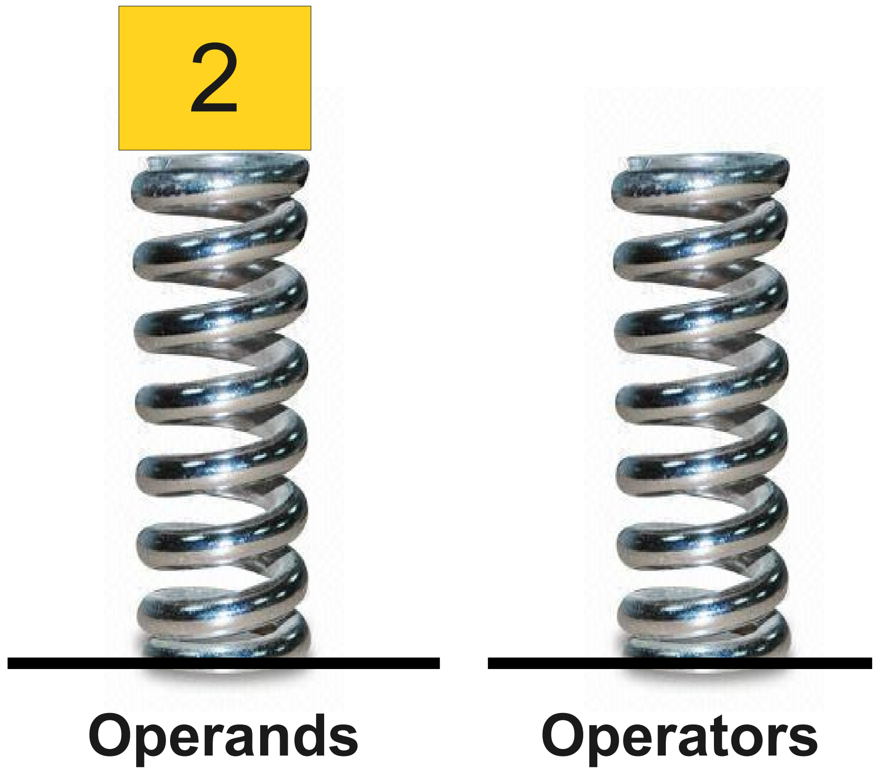
Operands

Operators

The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

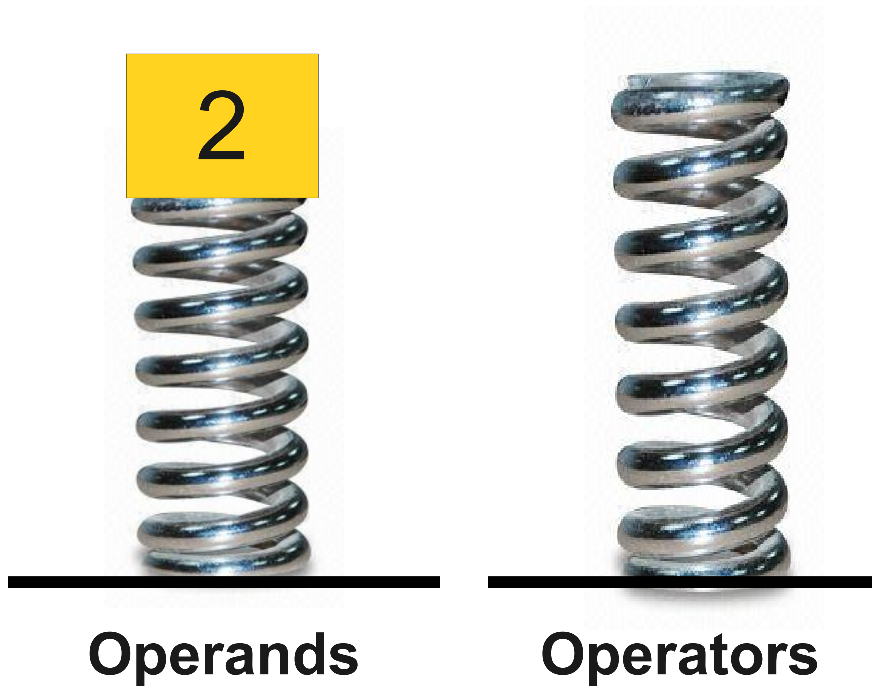
+	3	*	5	-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

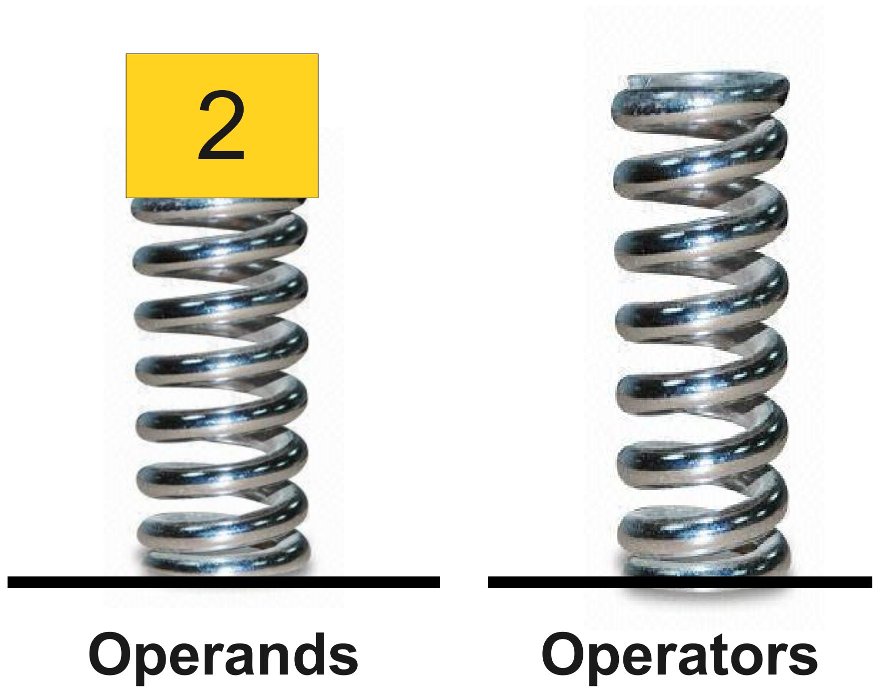
+	3	*	5	-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

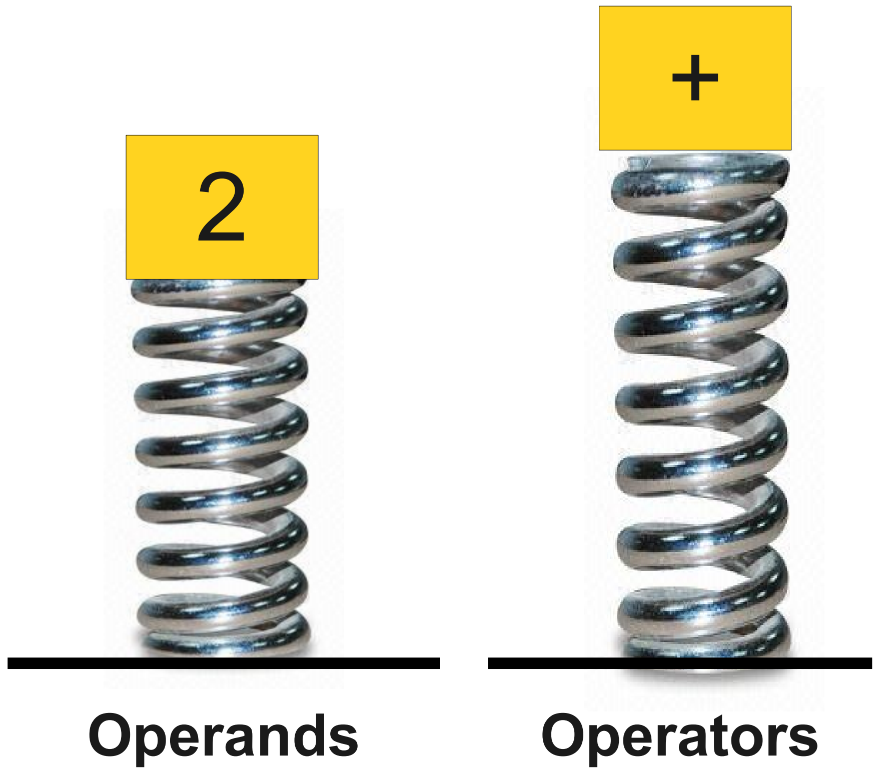
+	3	*	5	-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

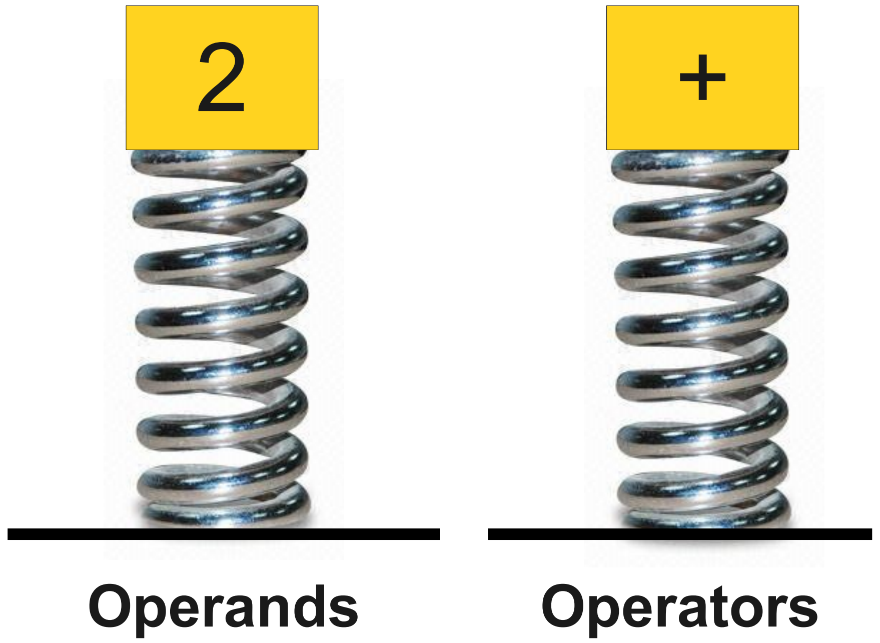
3	*	5	-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

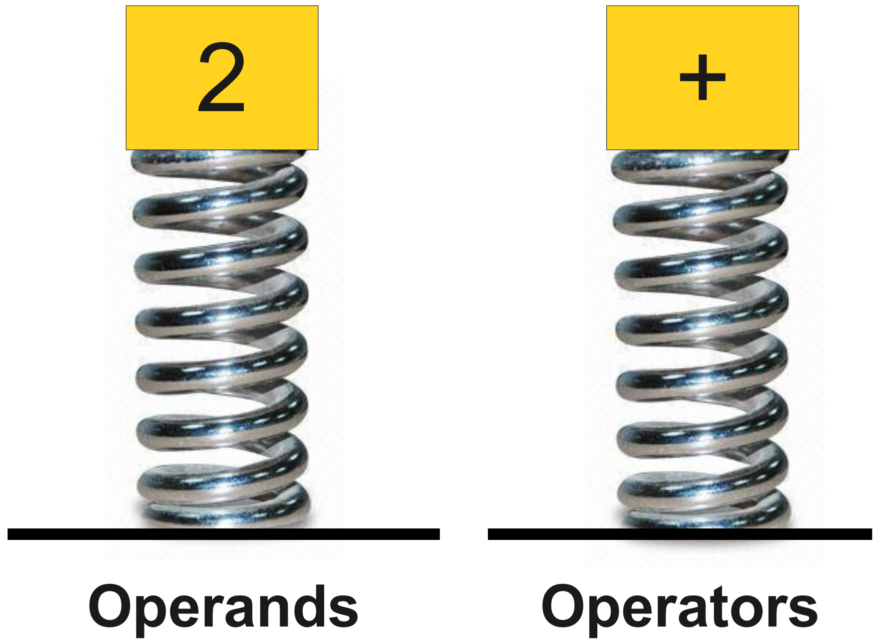
3	*	5	-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

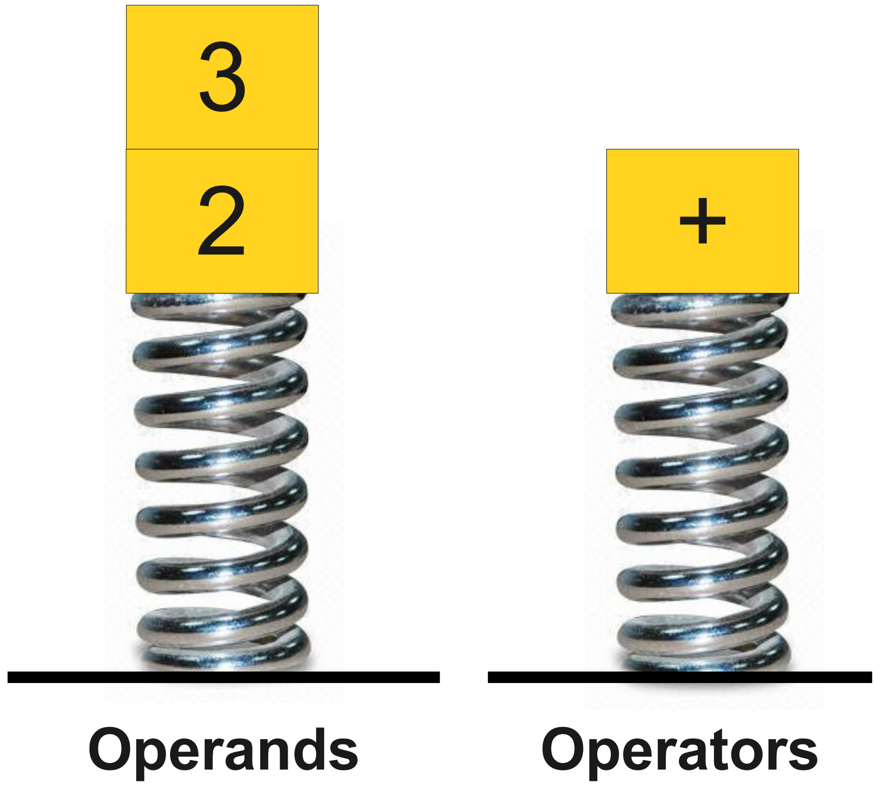
3	*	5	-	6	/	2



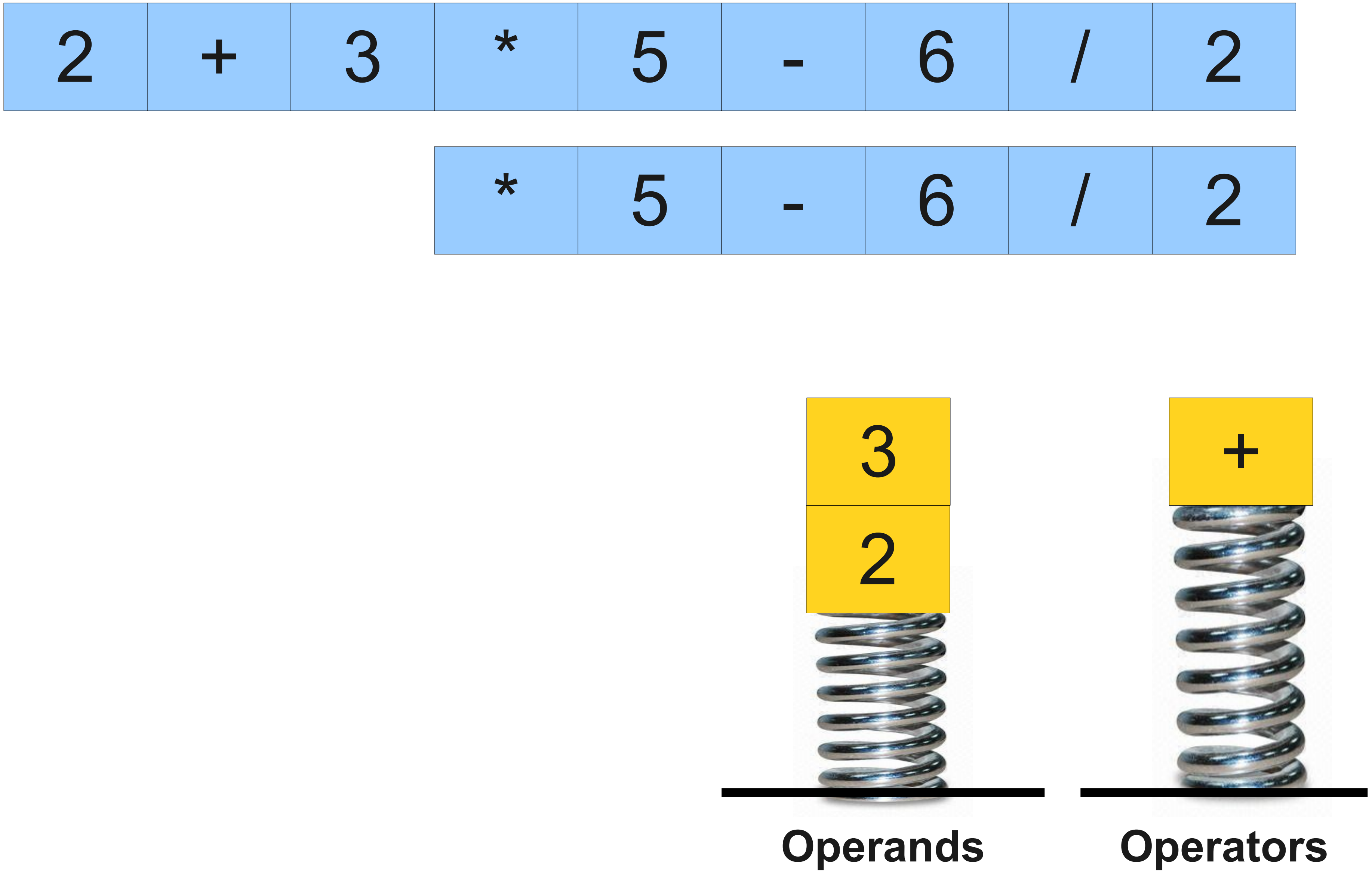
The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

*	5	-	6	/	2



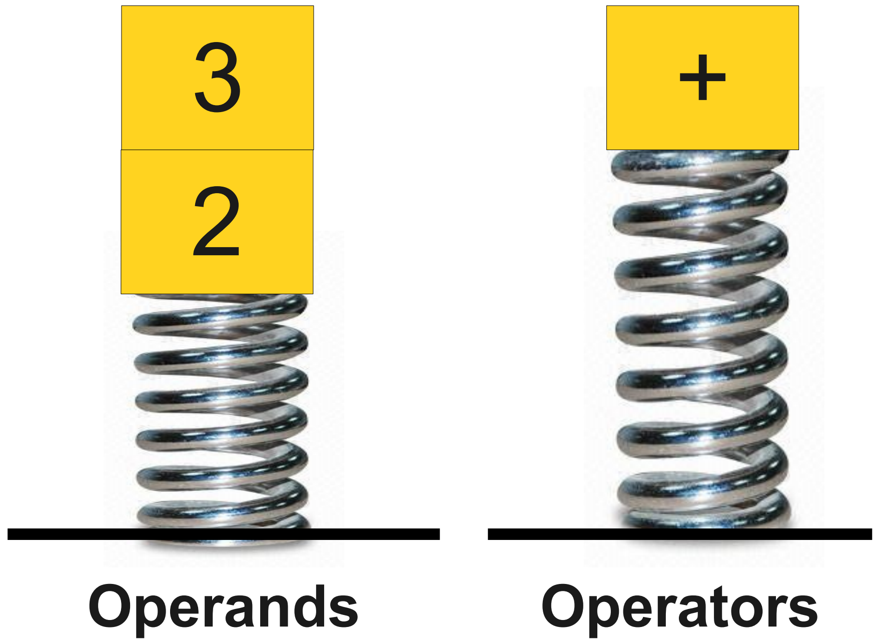
The Shunting-Yard Algorithm



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

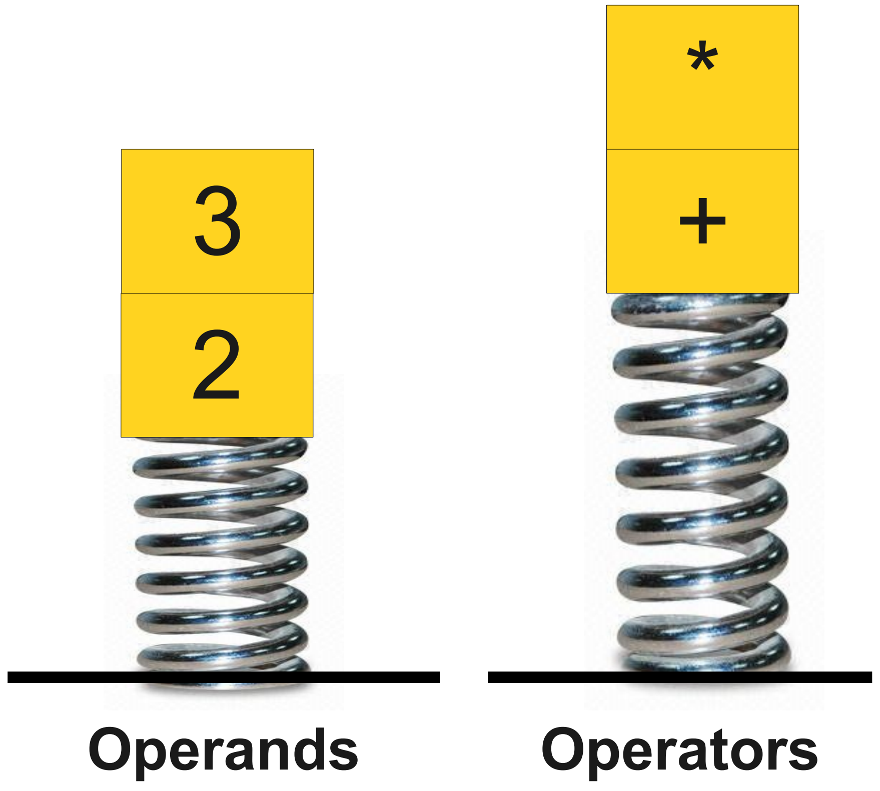
🟨 *	5	-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

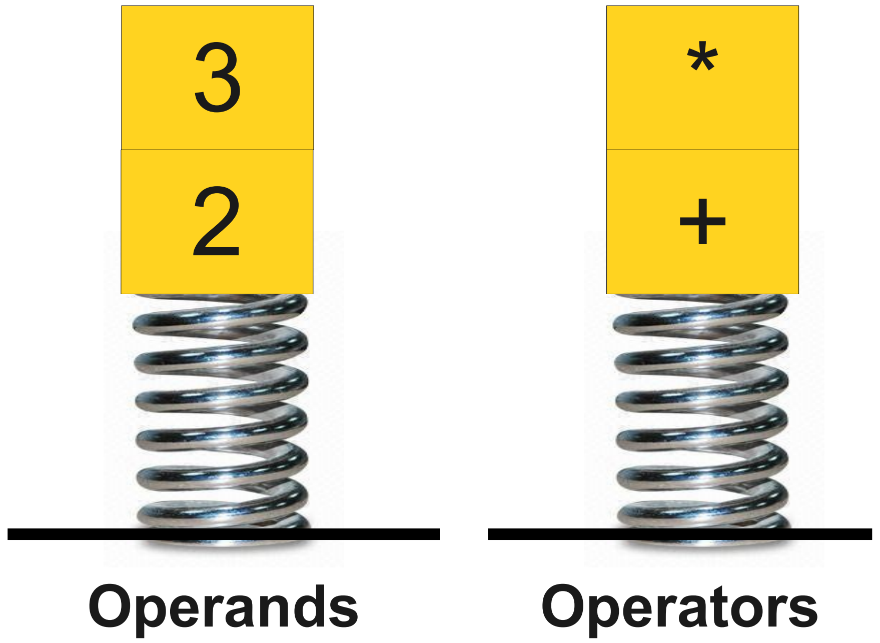
5	-	6	/	2



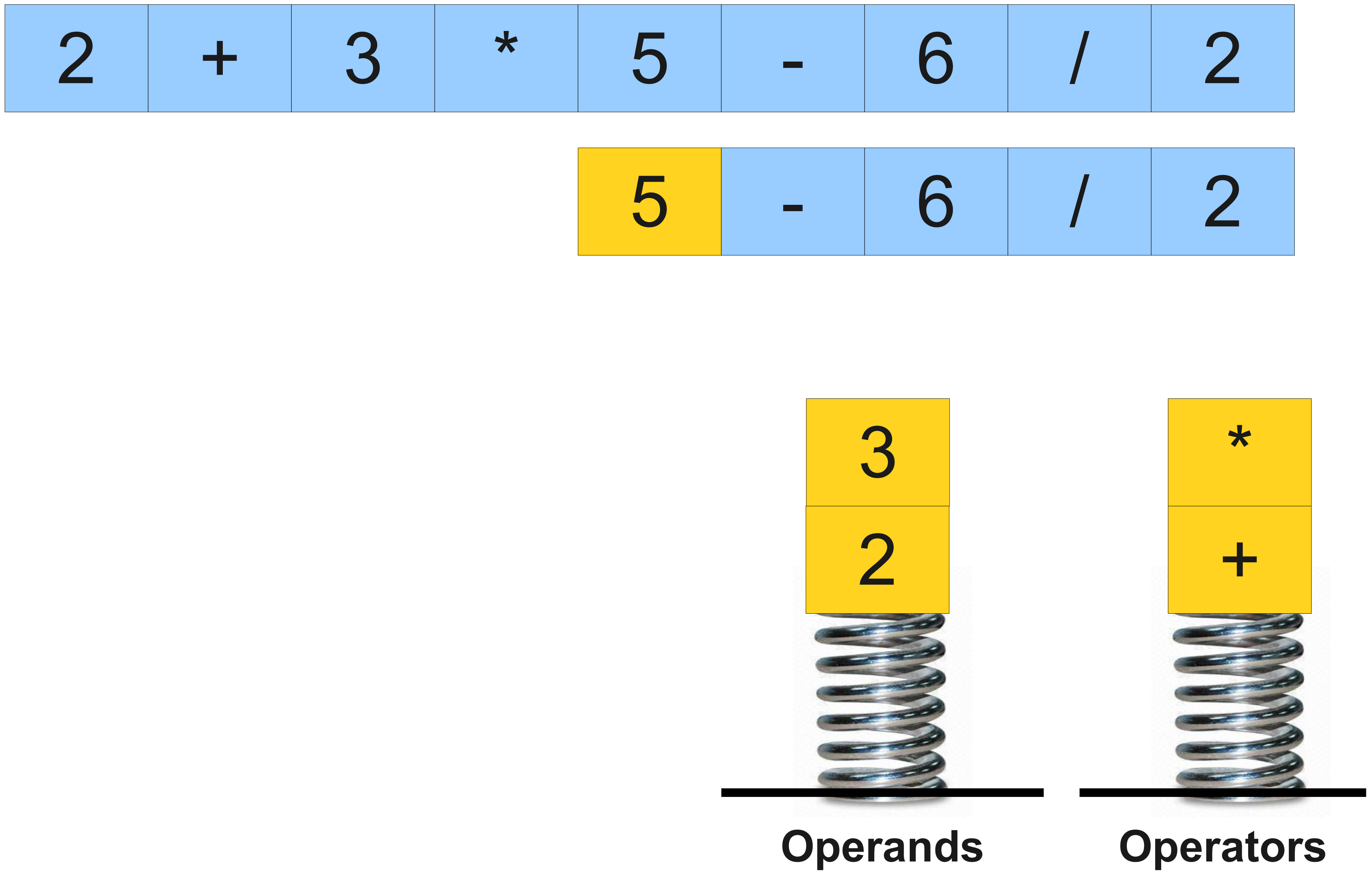
The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

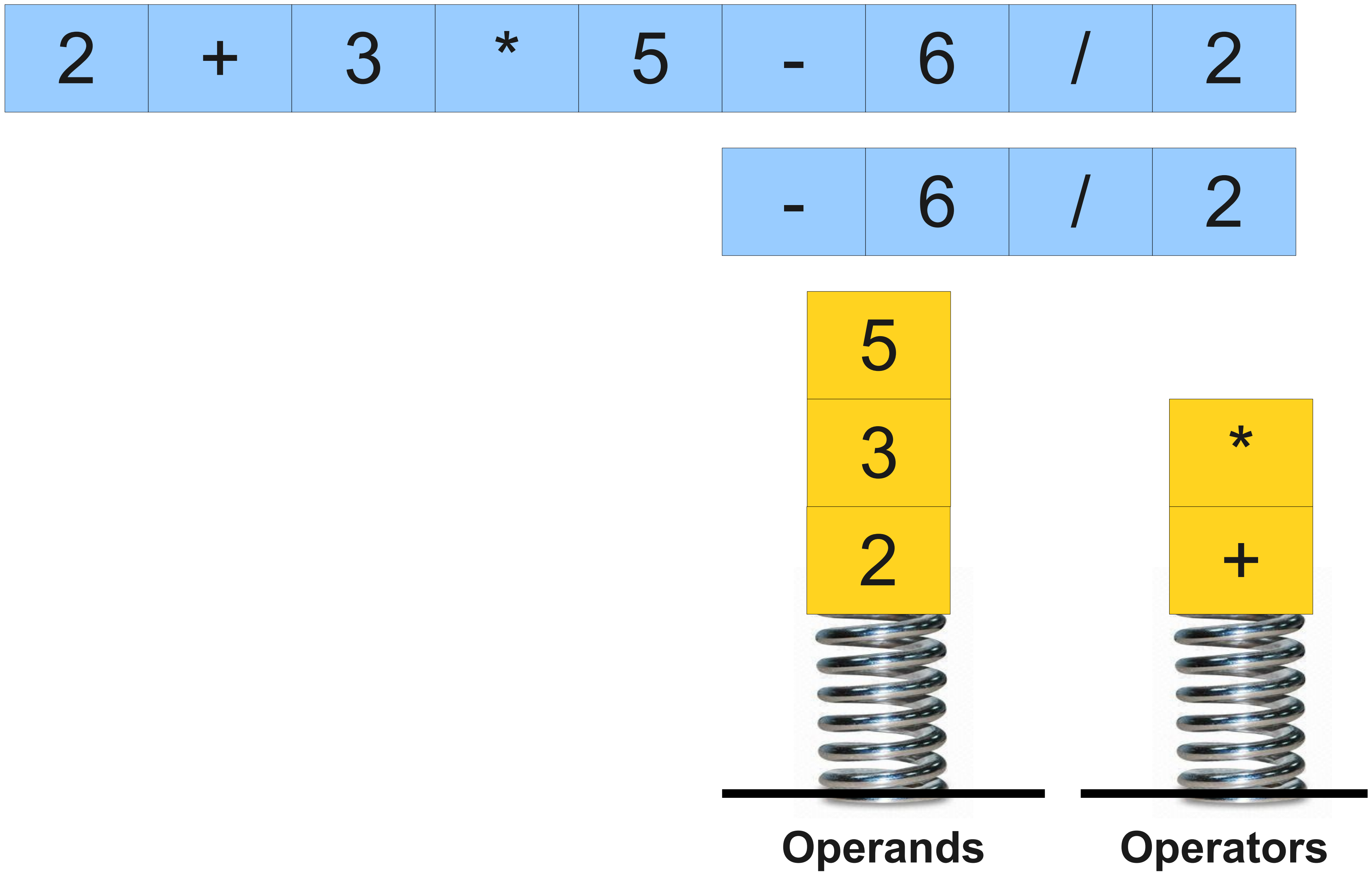
5	-	6	/	2



The Shunting-Yard Algorithm



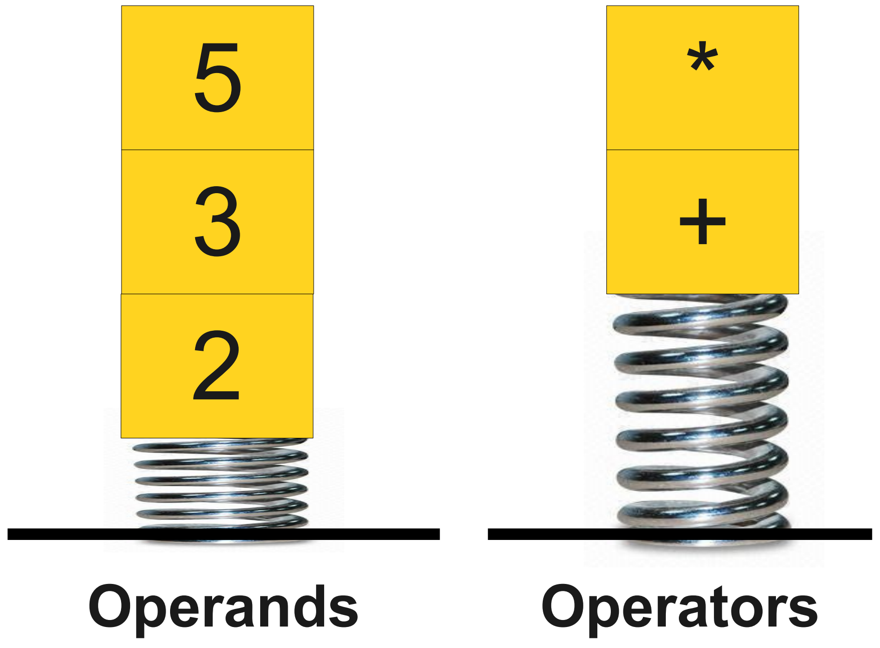
The Shunting-Yard Algorithm



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

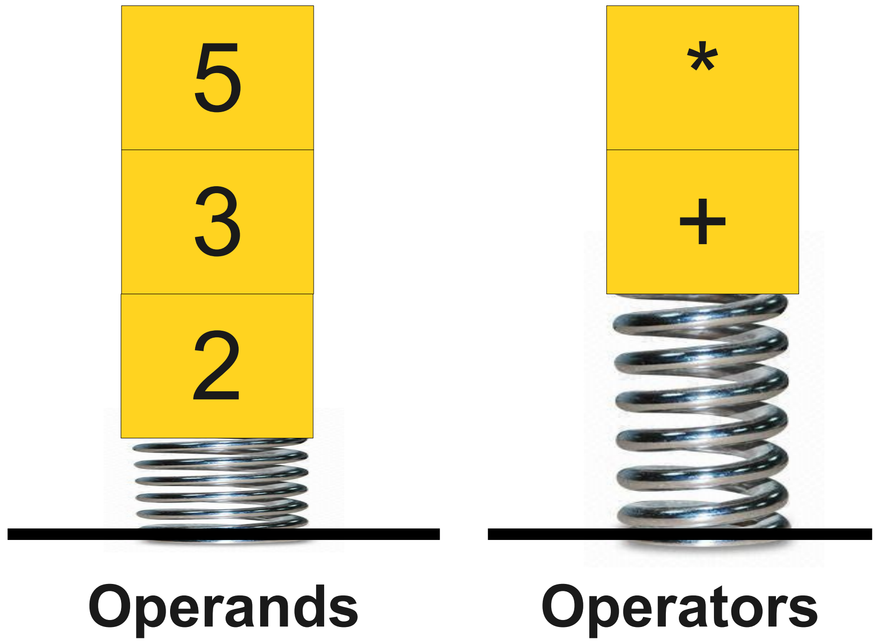
-	6	/	2



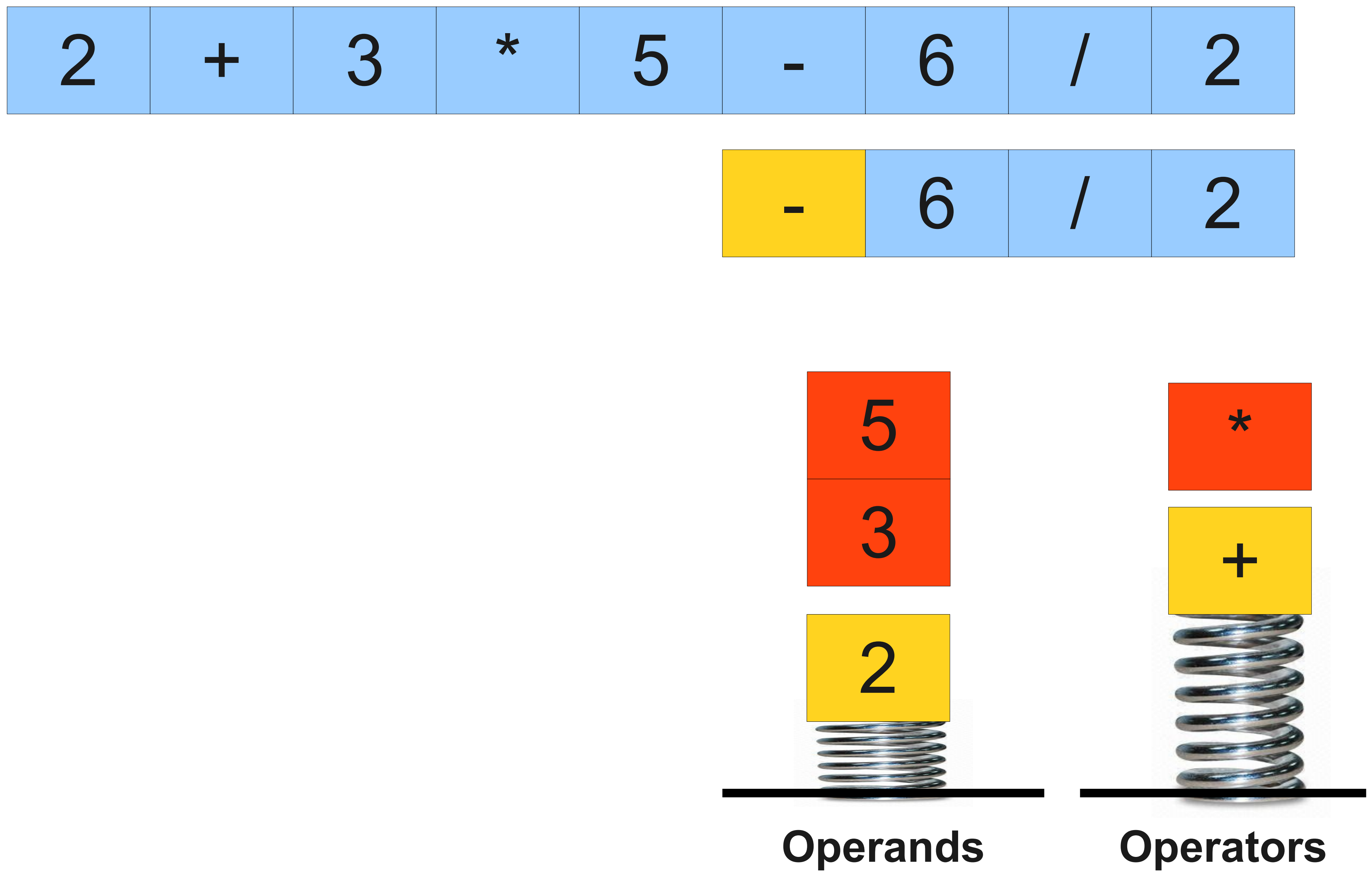
The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

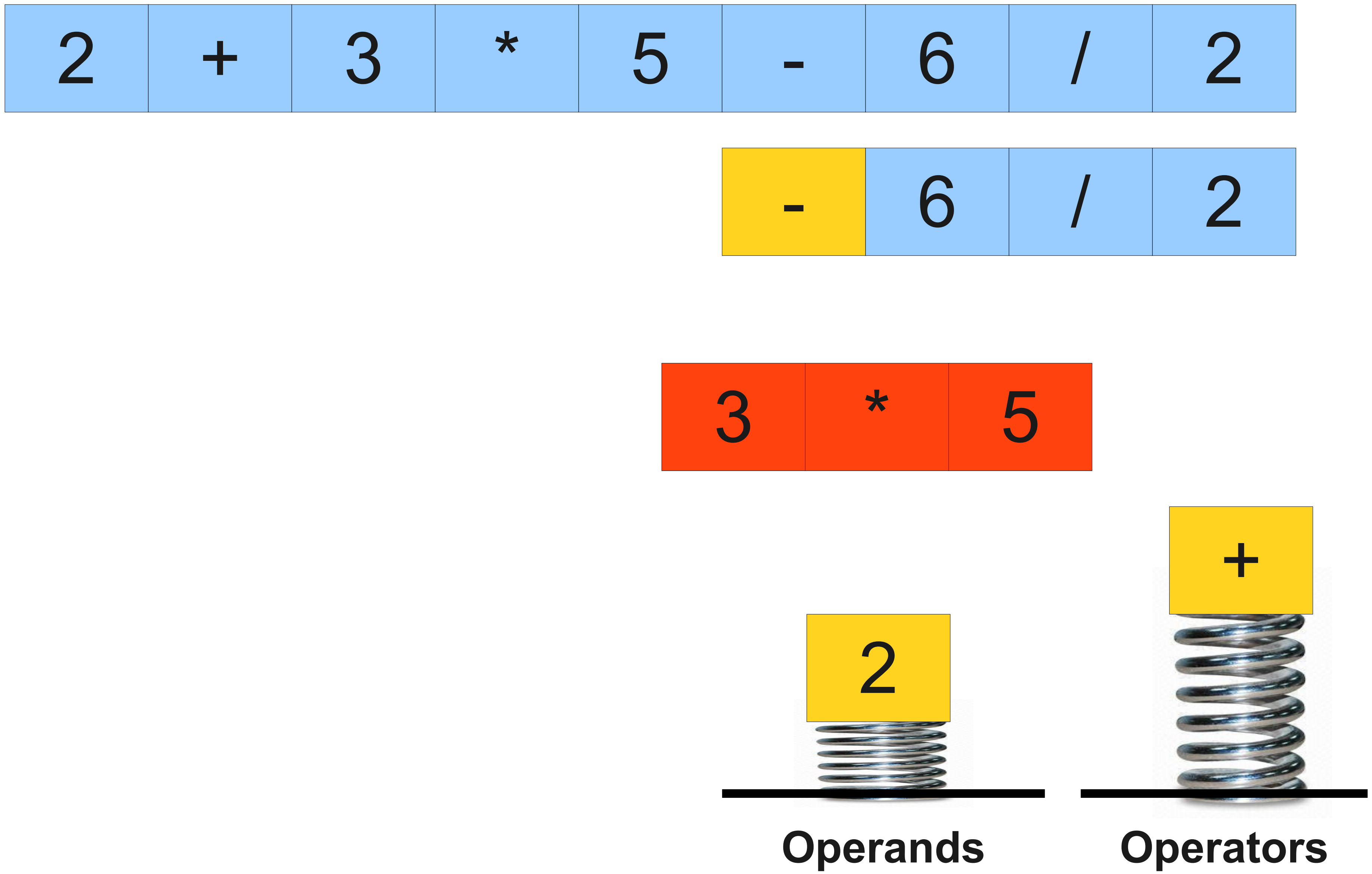
-	6	/	2



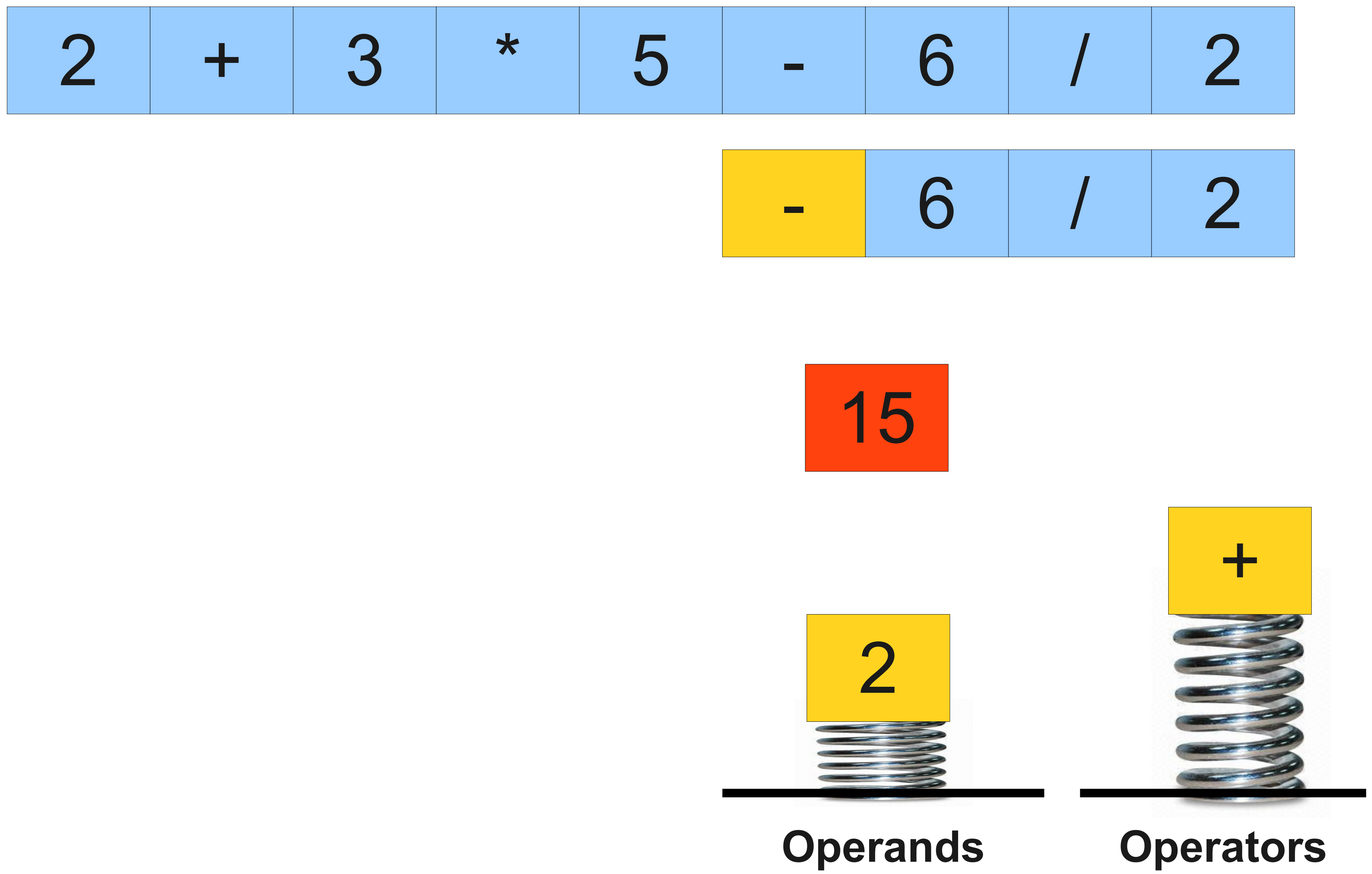
The Shunting-Yard Algorithm



The Shunting-Yard Algorithm



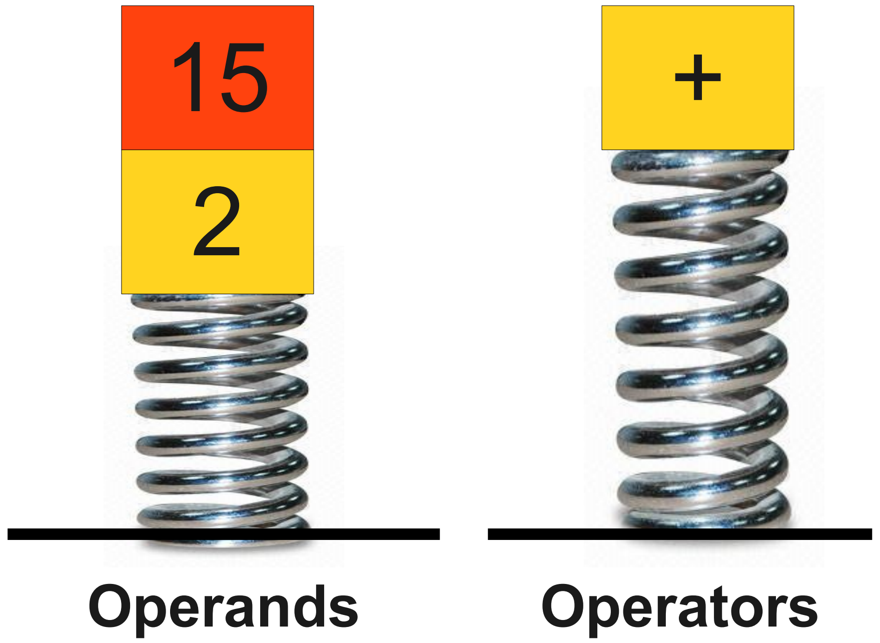
The Shunting-Yard Algorithm



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

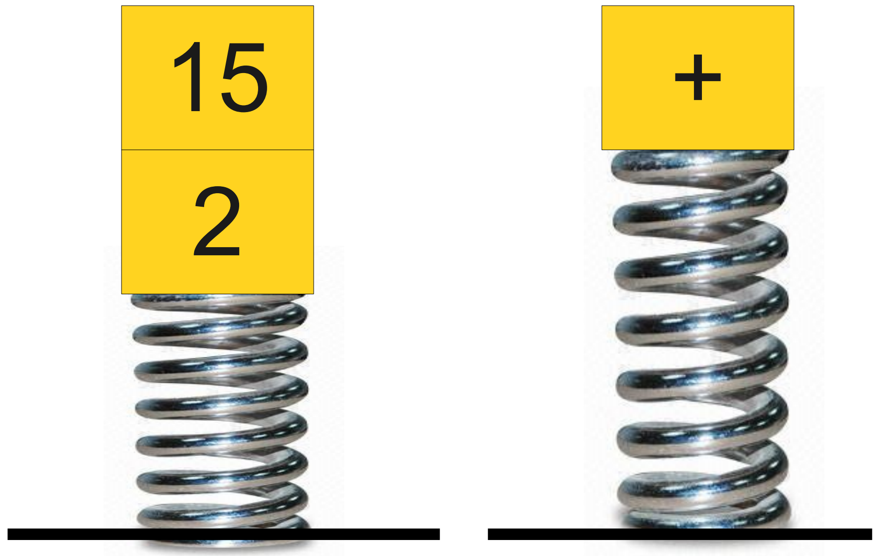
-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

-	6	/	2

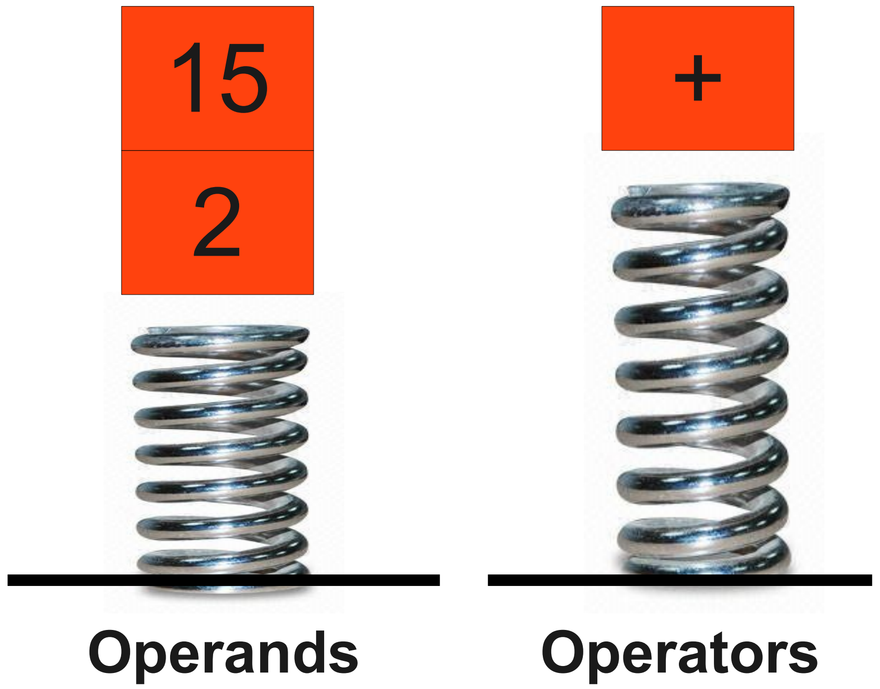


Operands Operators

The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

-	6	/	2

2	+	15

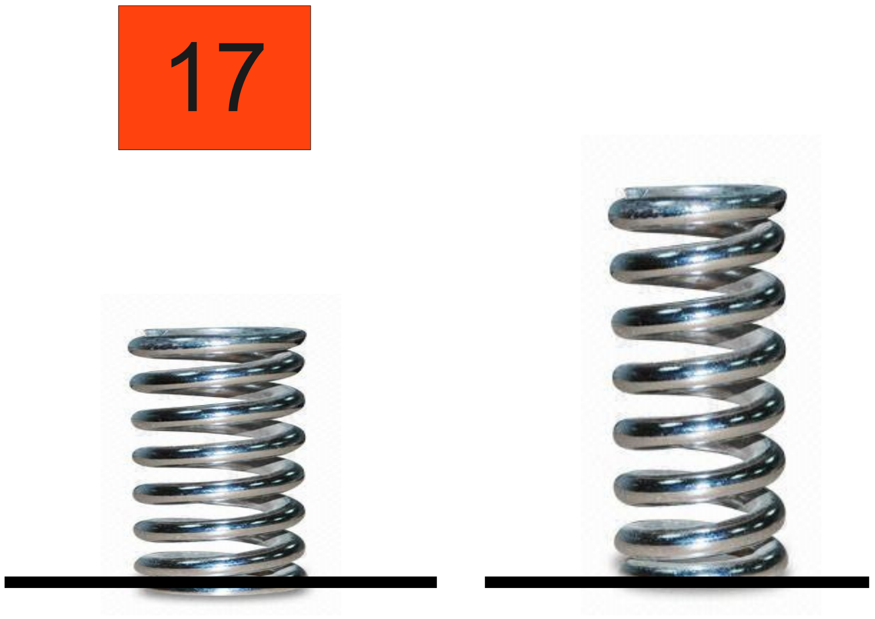
Operands

Operators

The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

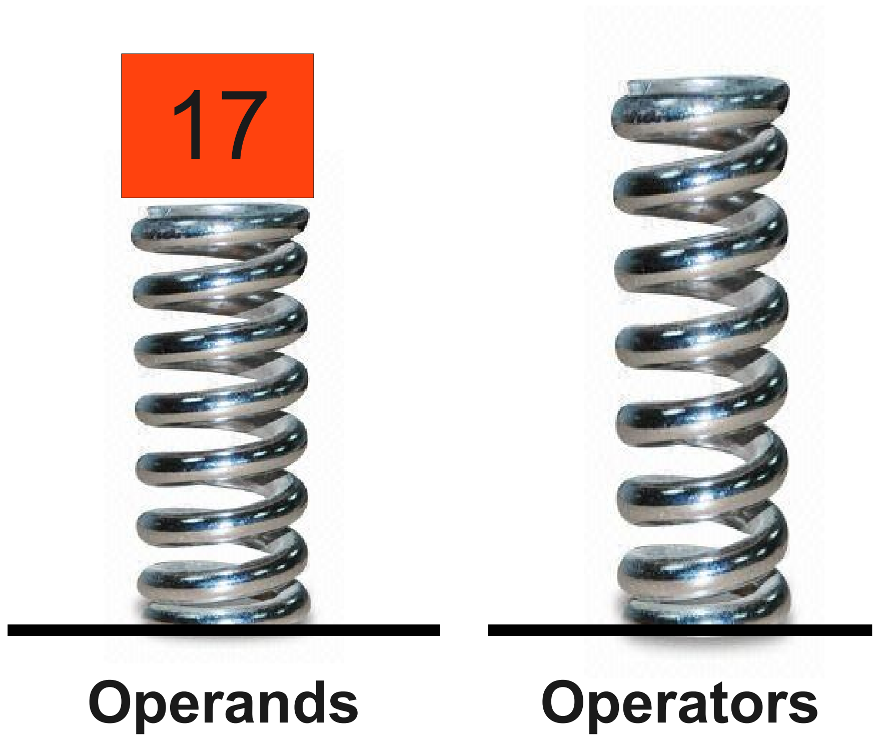
-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

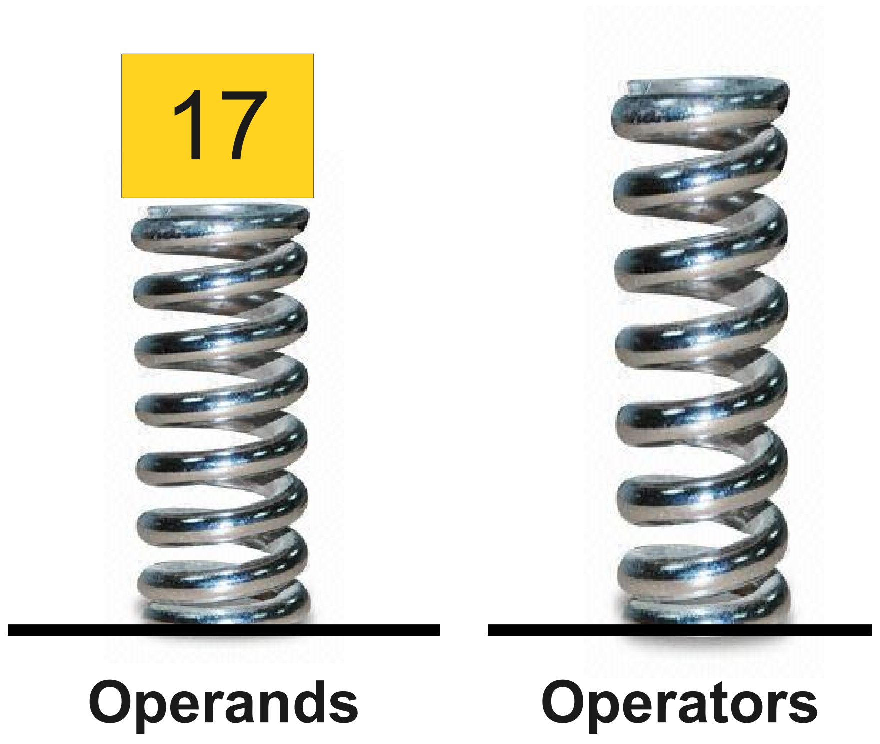
-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

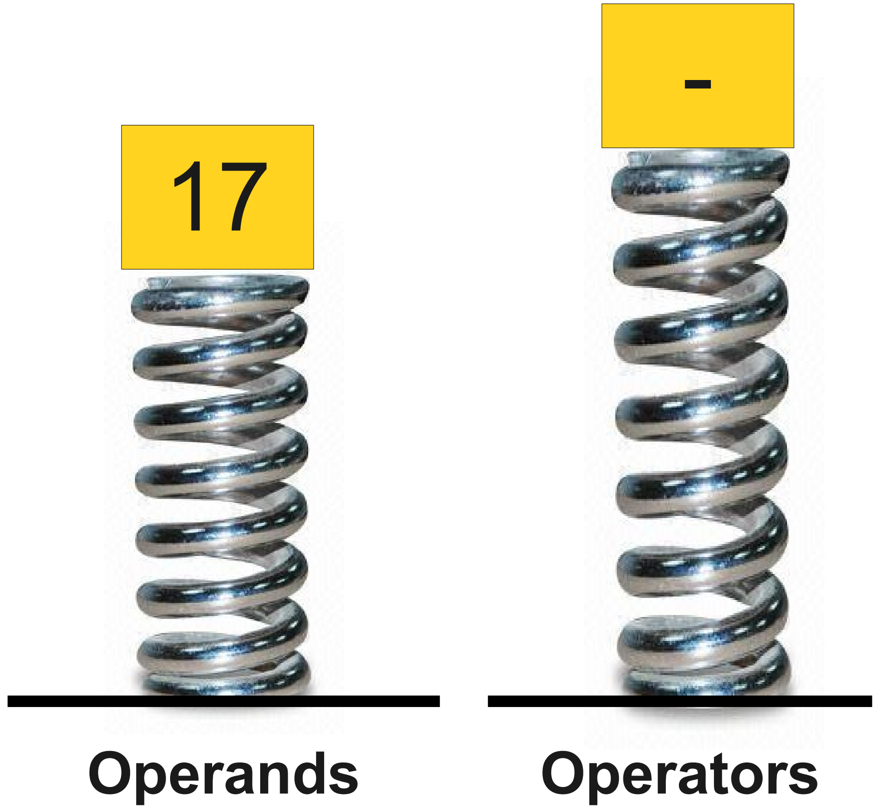
-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

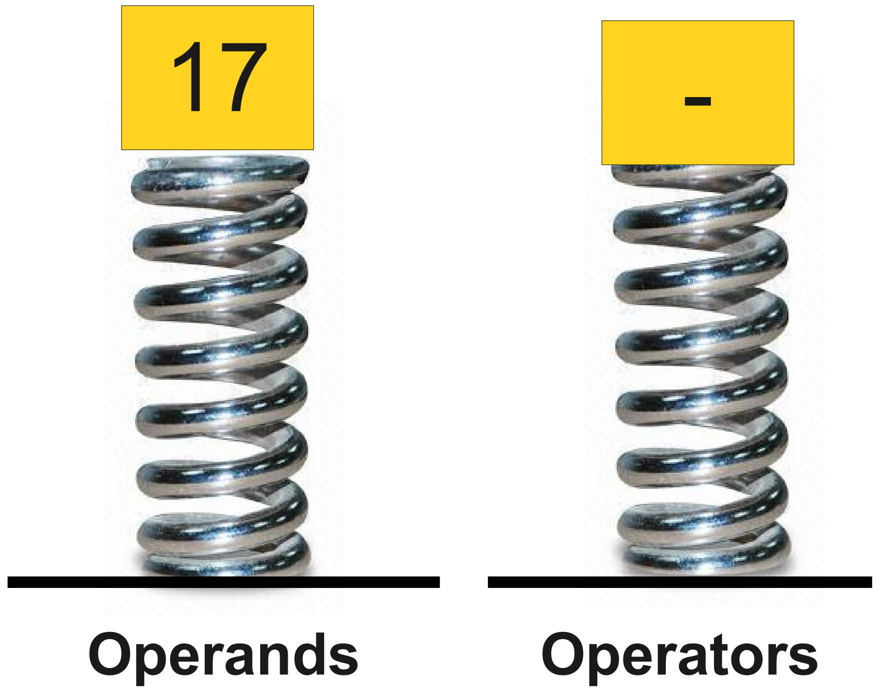
6	/	2



The Shunting-Yard Algorithm

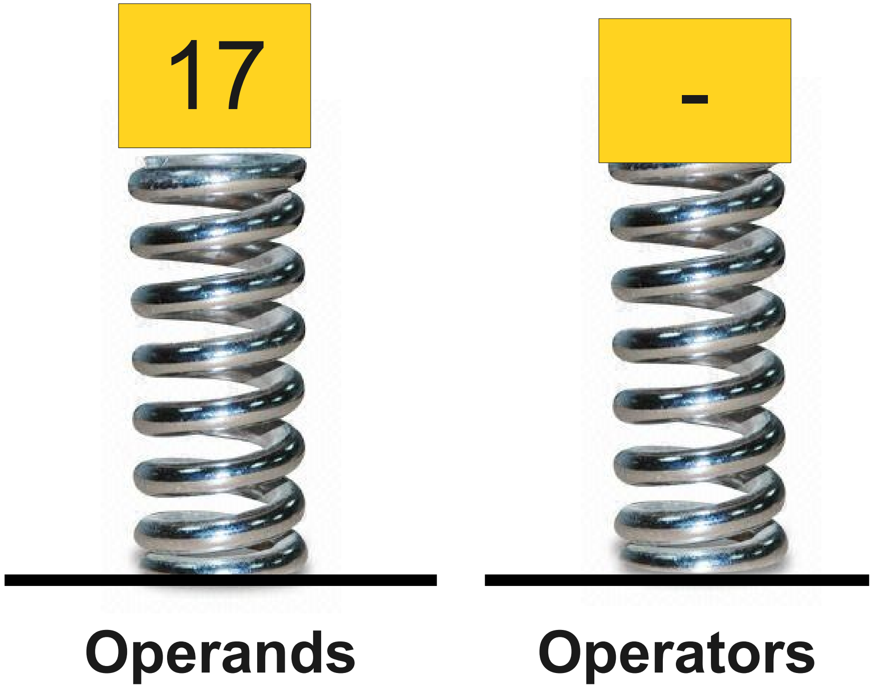
2	+	3	*	5	-	6	/	2

6	/	2



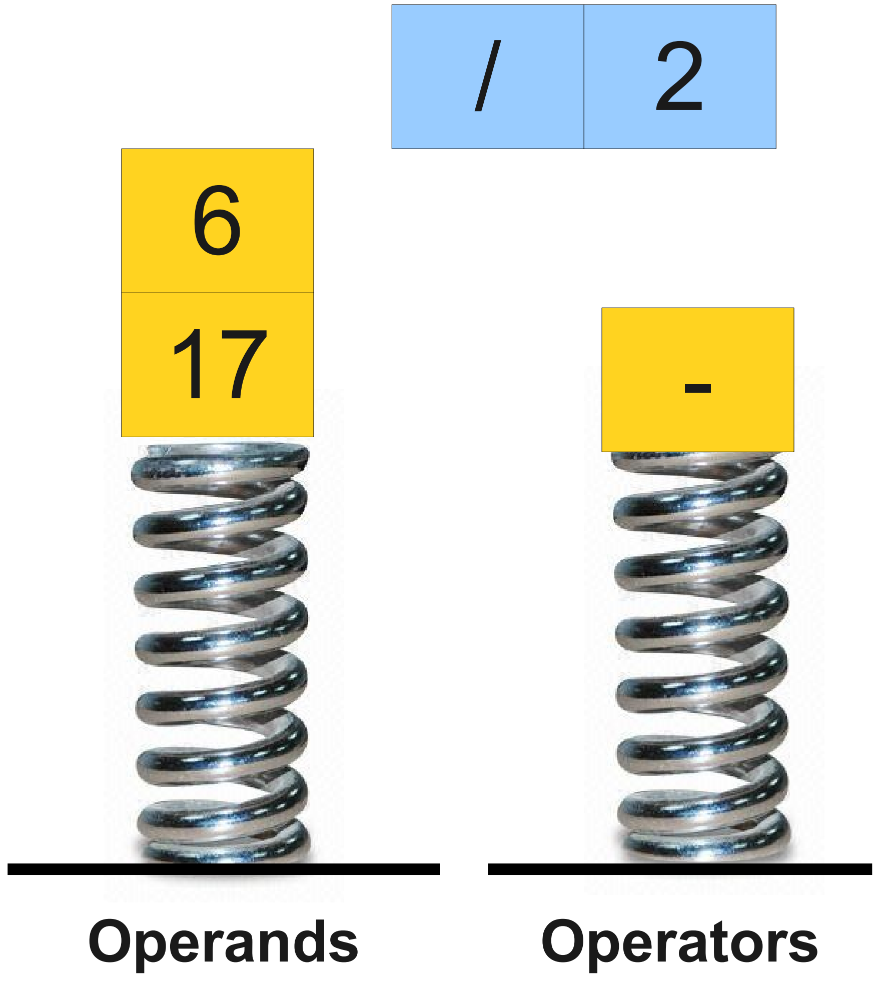
The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
						6	/	2



The Shunting-Yard Algorithm

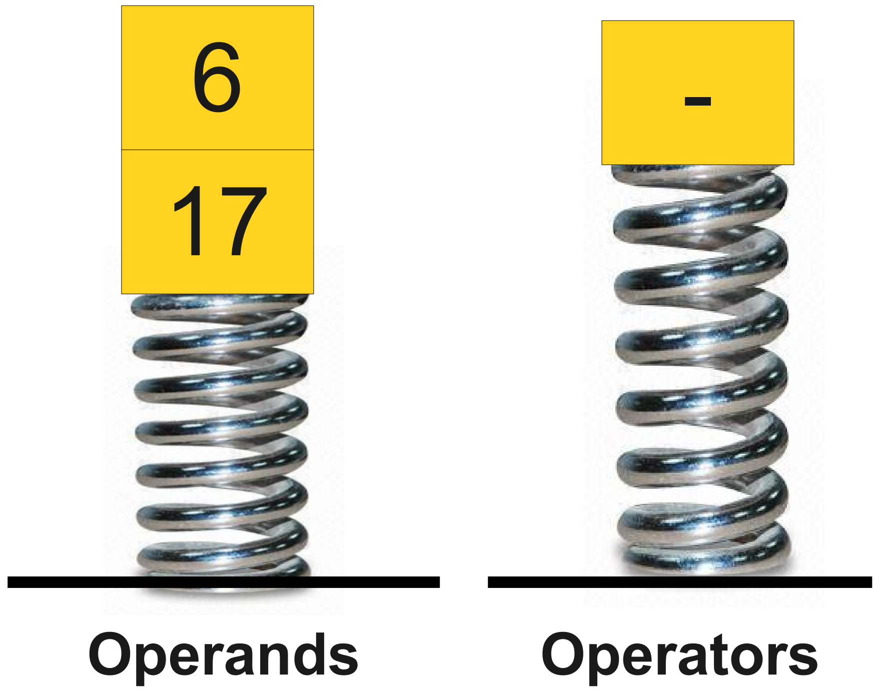
2	+	3	*	5	-	6	/	2



The Shunting-Yard Algorithm

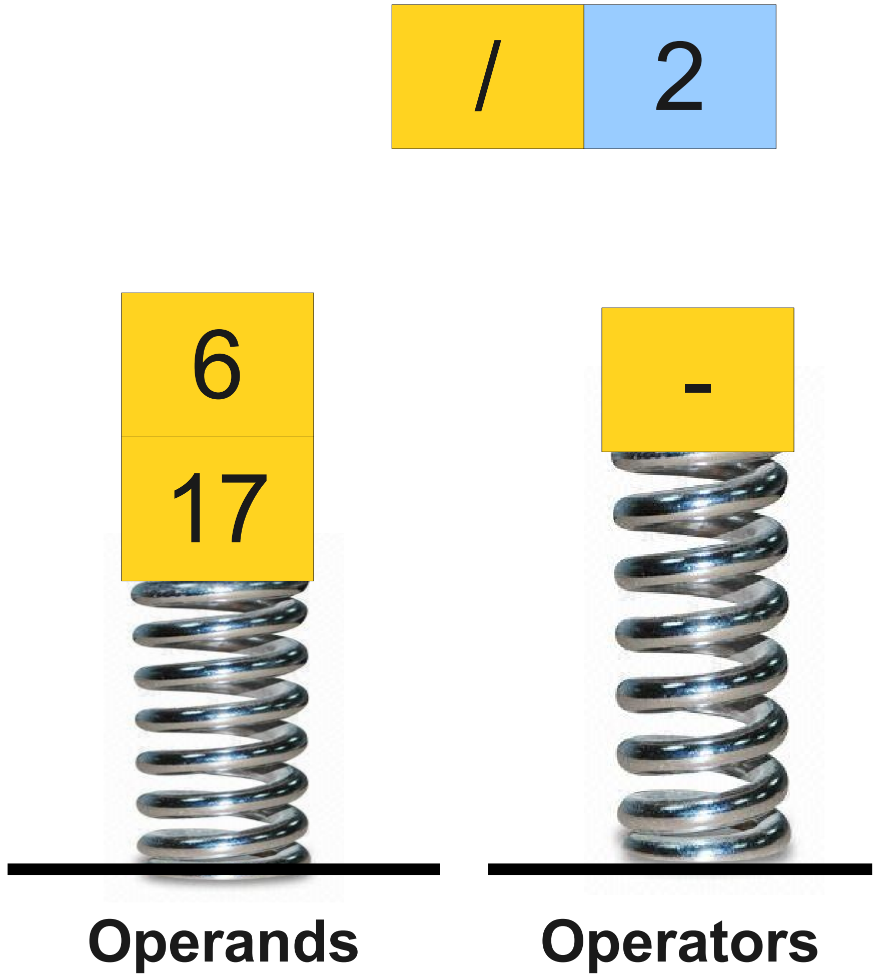
2	+	3	*	5	-	6	/	2

/	2

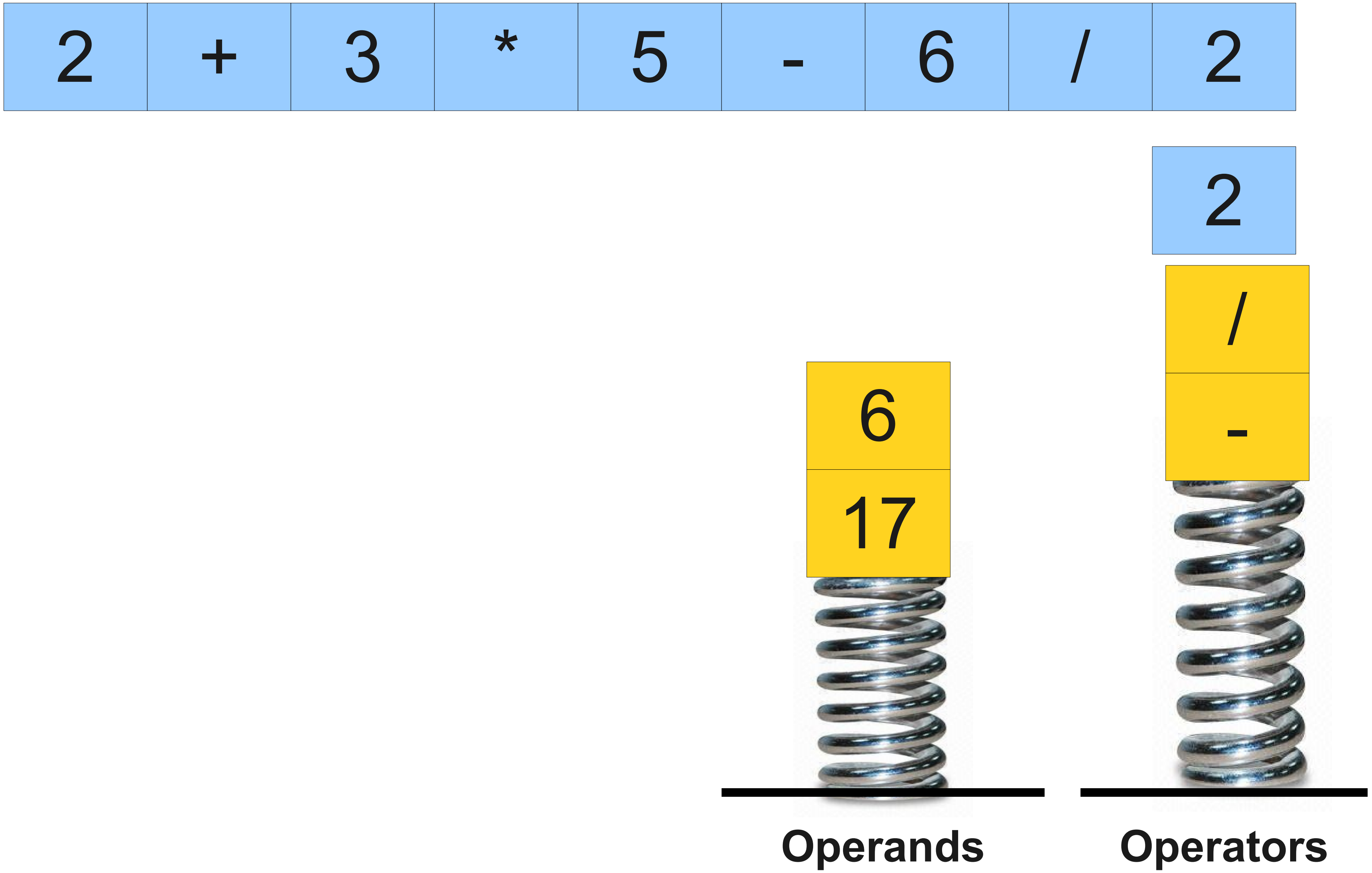


The Shunting-Yard Algorithm

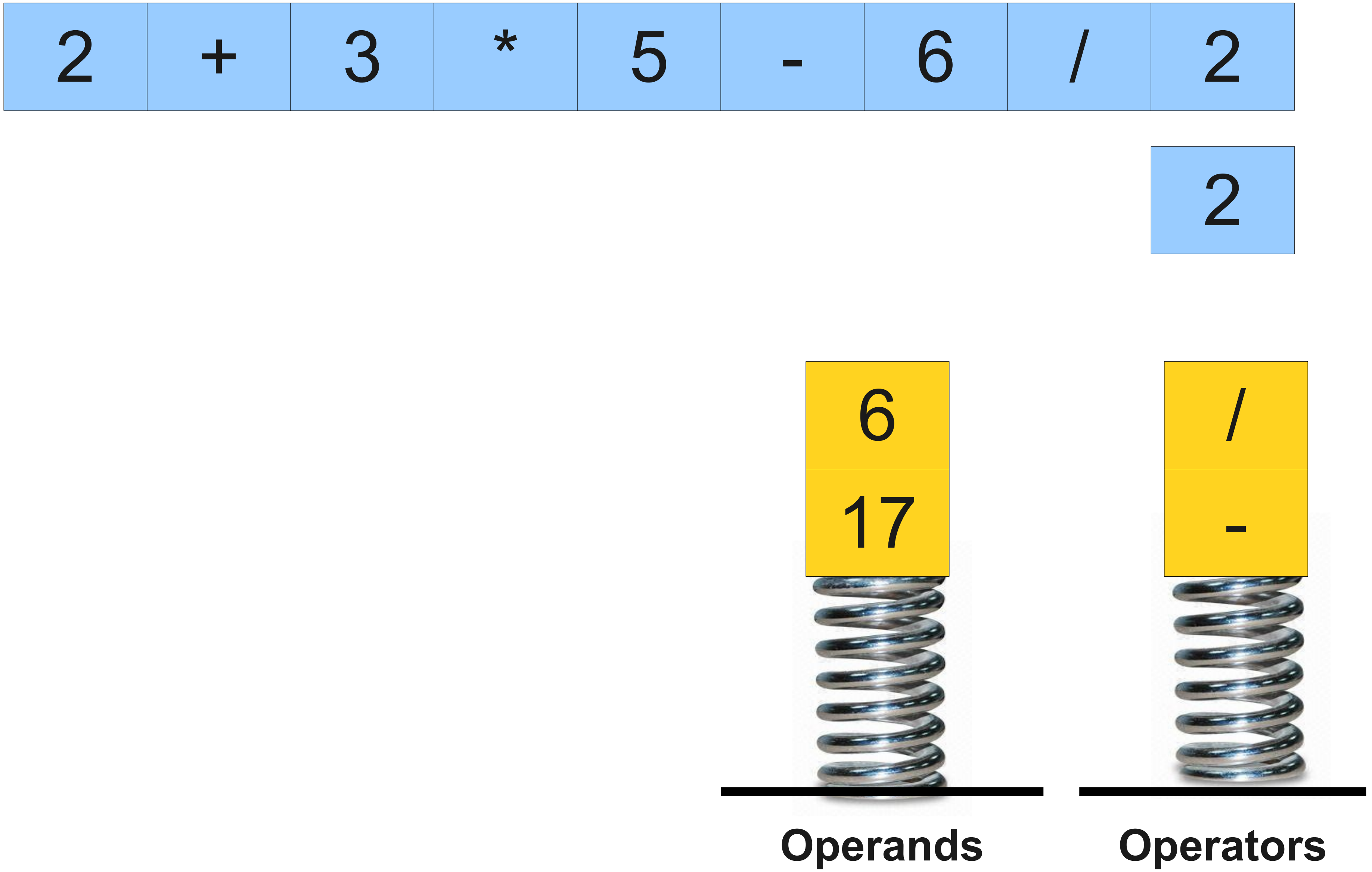
2	+	3	*	5	-	6	/	2



The Shunting-Yard Algorithm

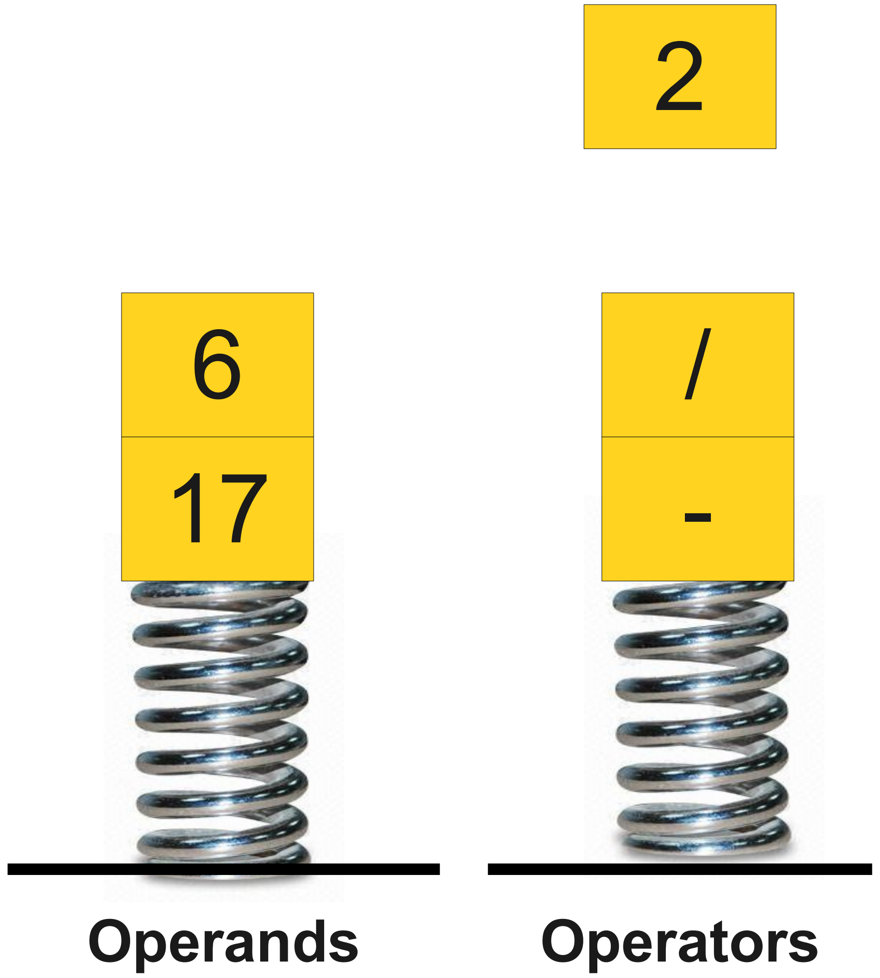


The Shunting-Yard Algorithm



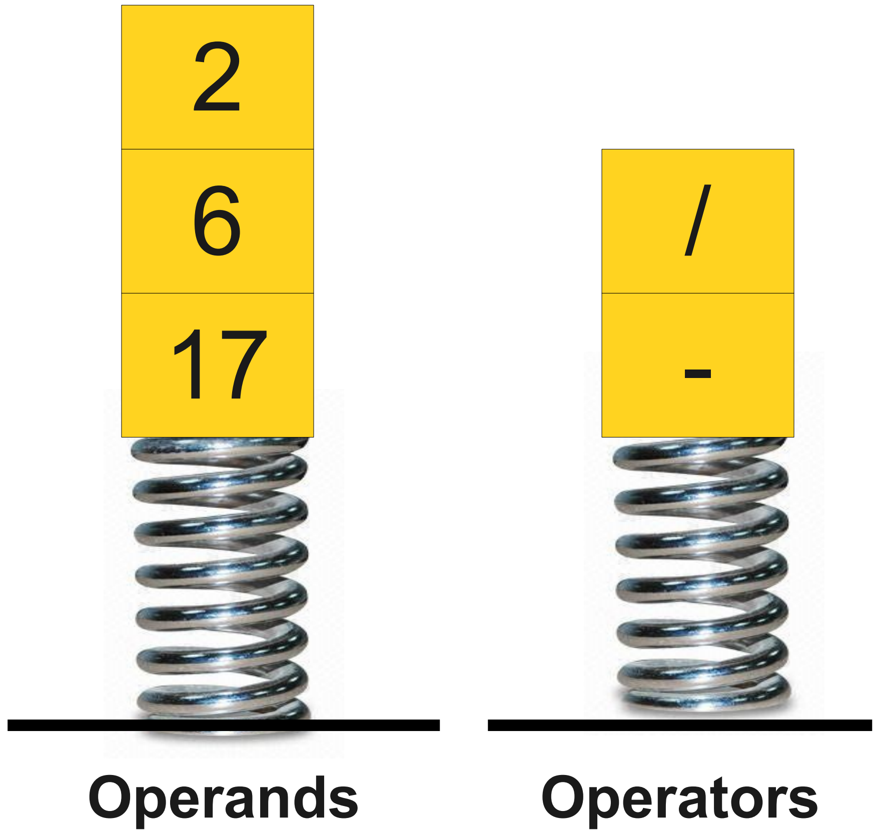
The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2



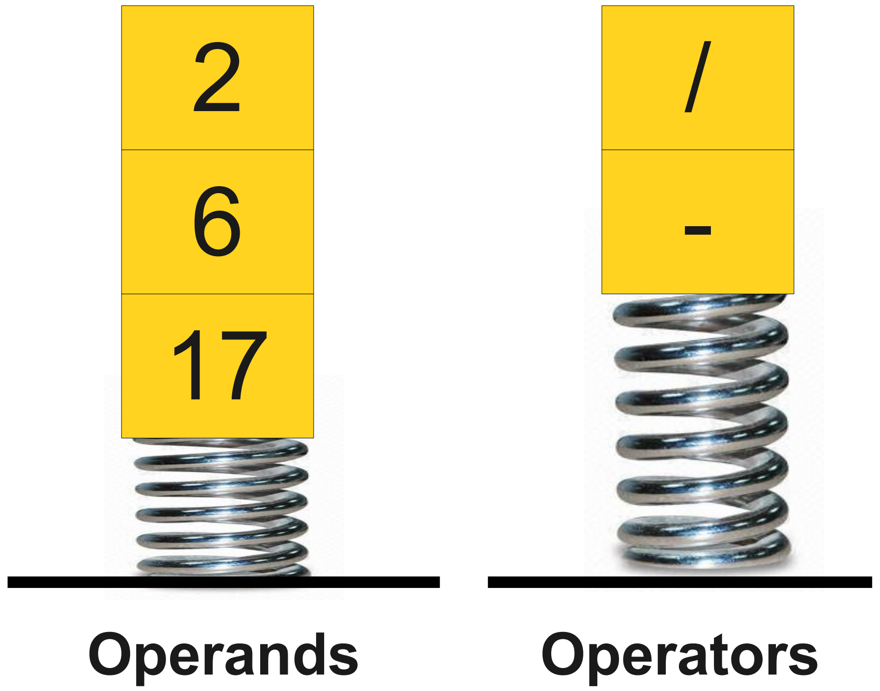
The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2



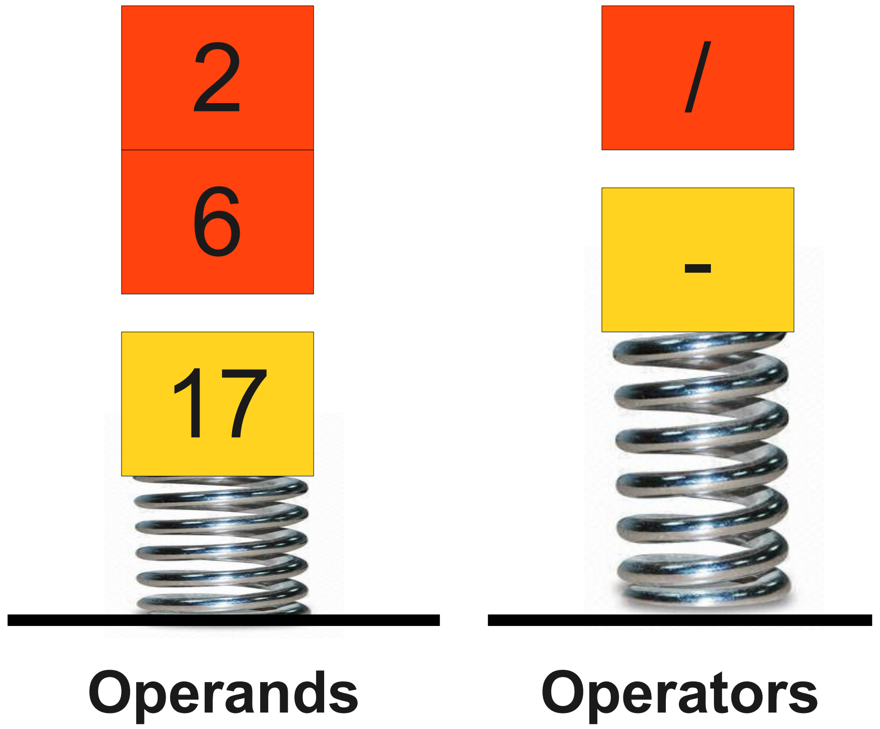
The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2



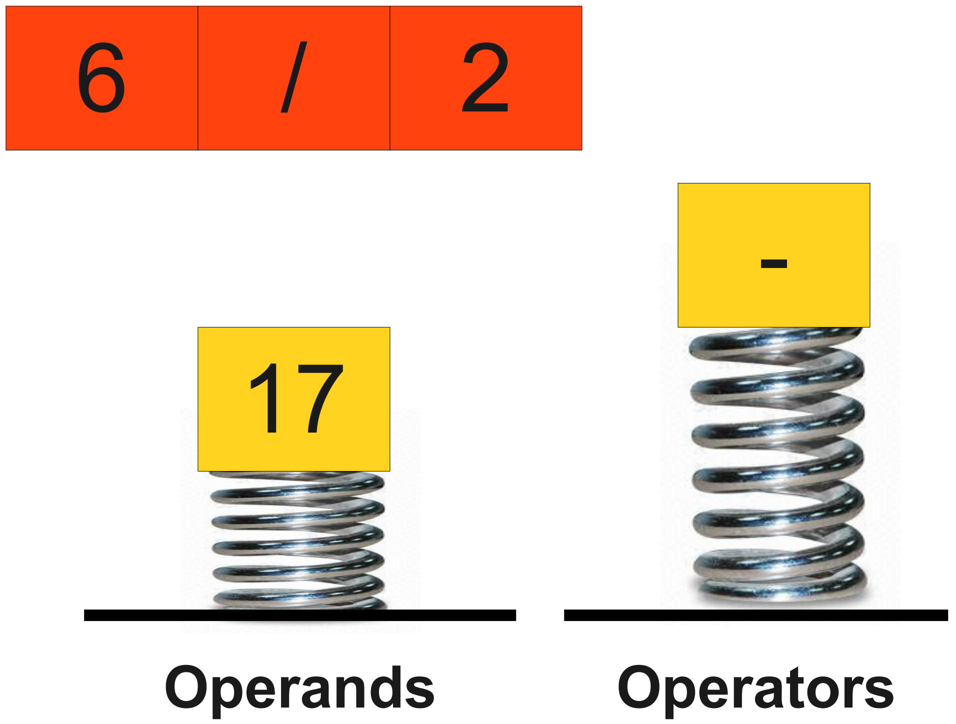
The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2



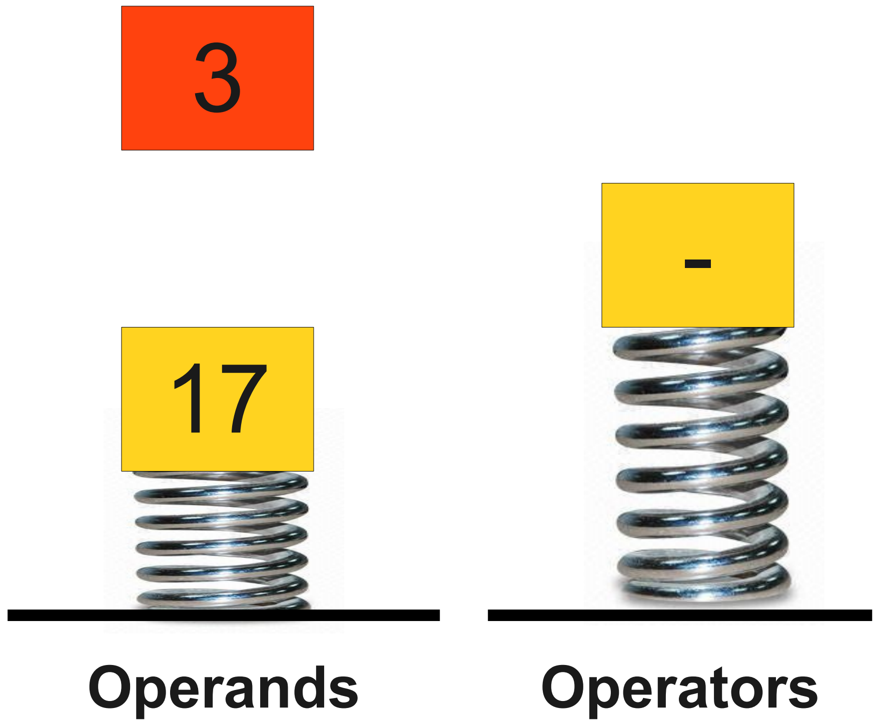
The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2



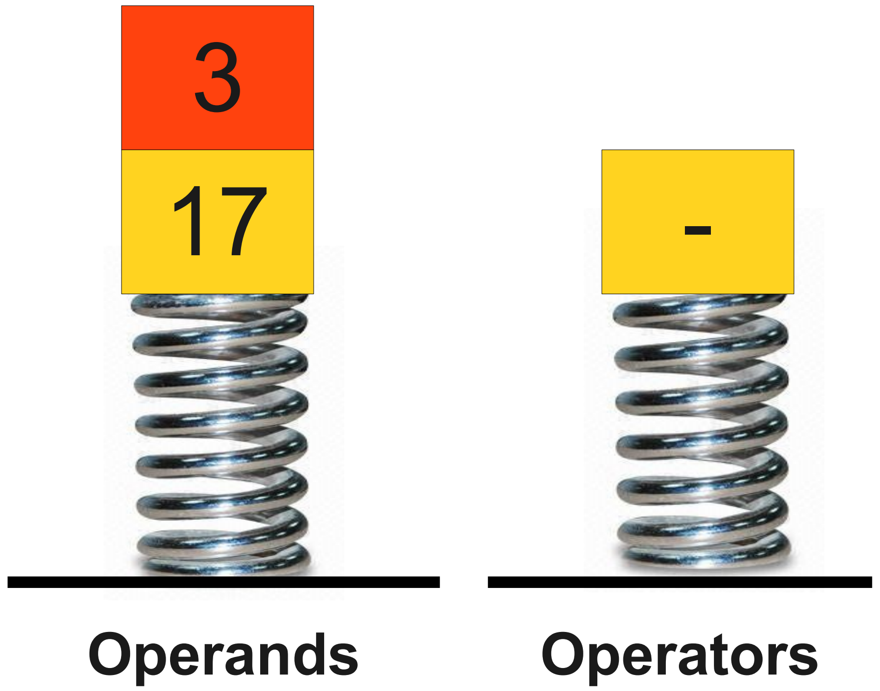
The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2



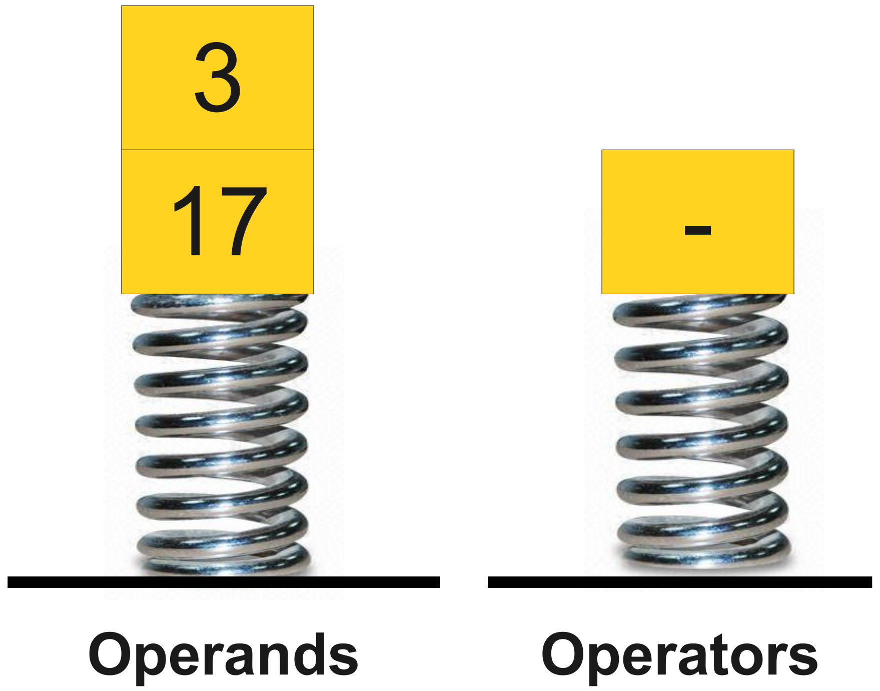
The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2



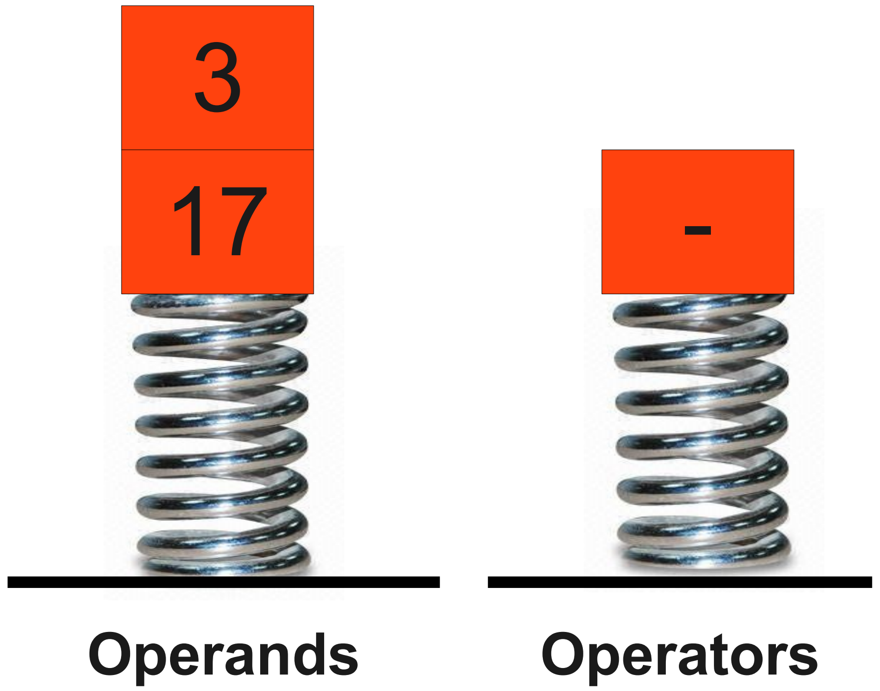
The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2

17	-	3

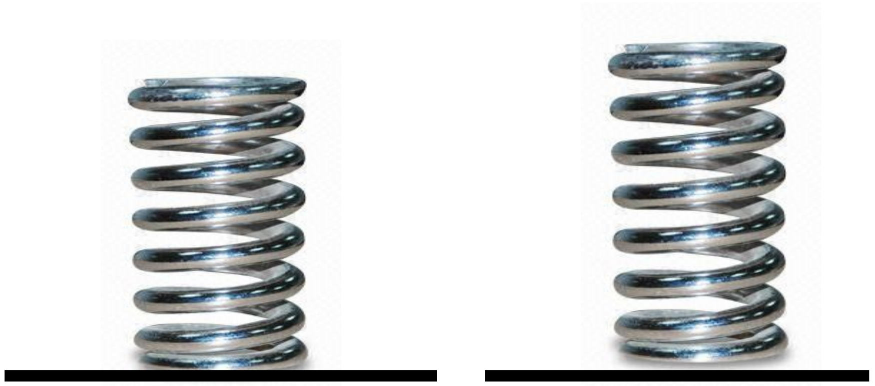
Operands

Operators

The Shunting-Yard Algorithm

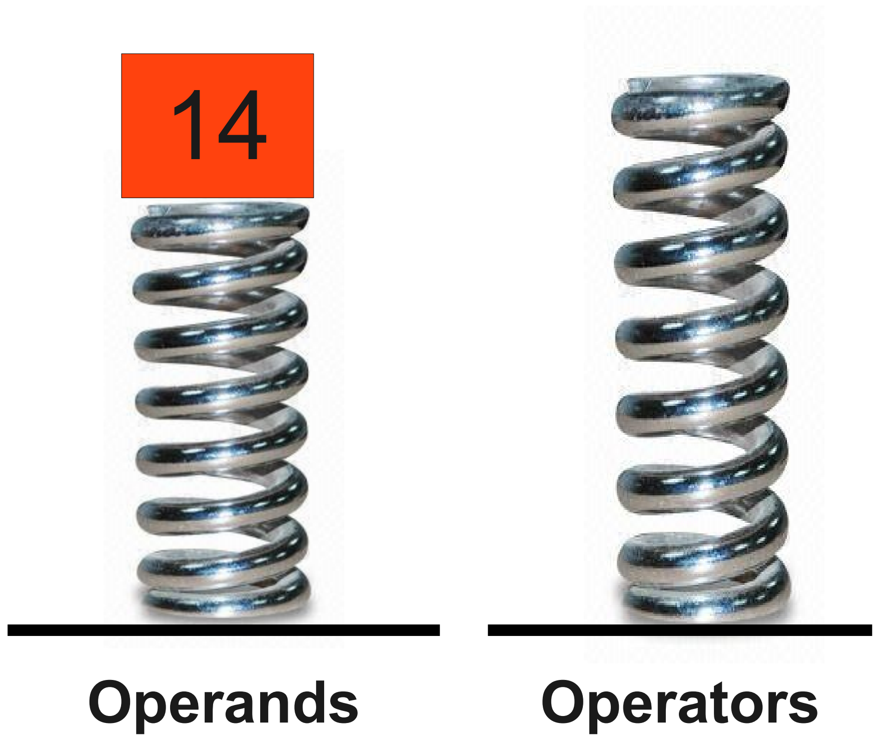
2	+	3	*	5	-	6	/	2

14



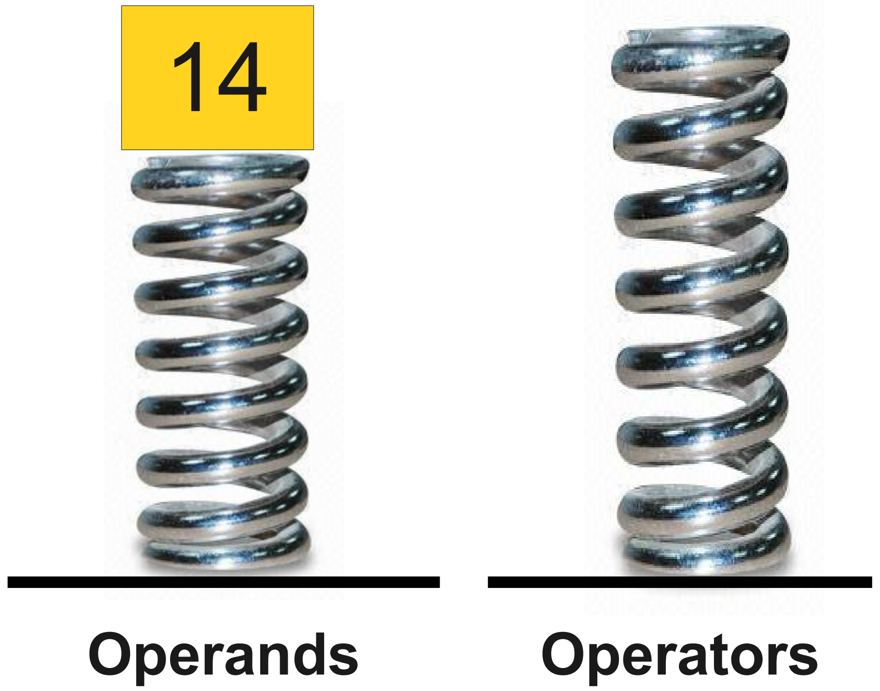
The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2



The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2



The Shunting-Yard Algorithm

- Maintain a stack of operators and operands.
- For each token:
  - If it's a number, push it onto the operand stack.
  - If it's an operator:
    - Keep evaluating operands until the current operator has higher precedence than the most recent operator.
    - Push the operator onto the operator stack.
- Once all input is done, keep evaluating operators until no operators remain.
- The value on the operand stack is the overall result.

Extensions to Shunting-Yard

- How might you handle/report syntax errors in the input?
- How might you handle parentheses?
- What about functions like sin, cos, and tan?
- Could you add support for variables?

Next Time

- **Thinking Recursively**
  - Just how much mileage can we get from recursion?
  - How do you think about problems recursively?