# Algorithms and Data Structures

## STORING AND ACCESSING DATA

**Robert Horvick**
SOFTWARE ENGINEER

@bubbafat   www.roberthorvick.com

# Overview

**Storing and accessing data using arrays**

- Creating arrays
- Adding and updating array data
- Enumerating array data

**Measuring algorithmic complexity**

- Asymptotic analysis
- Big-O notation

**Demo: Contact Manager**

- Overview of design and code

| Time | Reading |
|---|---|
| 06:12:00 | 1 |

| Time | Reading |
|---|---|
| 06:12:00 | 1 |
| 06:12:05 | 4 |

| Time | Reading |
|---|---|
| 06:12:00 | 1 |
| 06:12:05 | 4 |
| 06:12:10 | 3 |

| Time | Reading |
|---|---|
| 06:12:00 | 1 |
| 06:12:05 | 4 |
| 06:12:10 | 3 |
| 06:12:15 | 6 |

```
struct Reading {
  DateTime time;
  int value;
};

Reading r1 = Gauge.Read();
Thread.Sleep(5000);

Reading r2 = Gauge.Read();
Thread.Sleep(5000);

Reading r3 = Gauge.Read();
Thread.Sleep(5000);

Reading r4 = Gauge.Read();
Thread.Sleep(5000);

Reading r5 = Gauge.Read();
Thread.Sleep(5000);
```

◀ The gauge reading structure

◀ Read the gauge data into a variable

◀ Wait 5 seconds

◀ Repeat the read and wait process

# 5 seconds

1

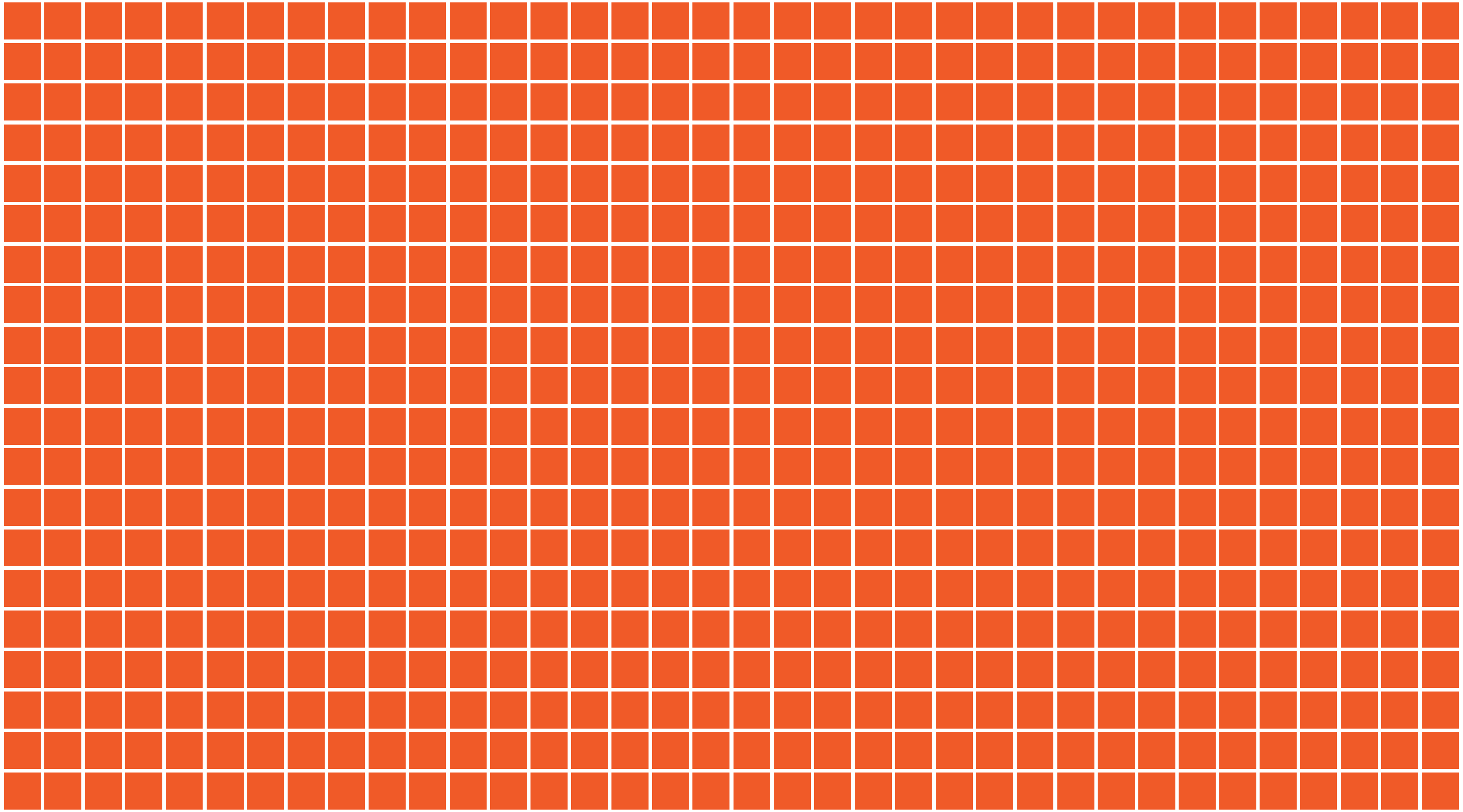
# 30 Seconds

| | | |
|---|---|---|
| 1 | 4 | 3 |
| 6 | 7 | 5 |

# 1 Minute

| | | | | | |
|---|---|---|---|---|---|
| 1 | 4 | 3 | 6 | 7 | 5 |
| 2 | 8 | 0 | 9 | 8 | 2 |

# 1 Hour



Can contain multiple instances of a type

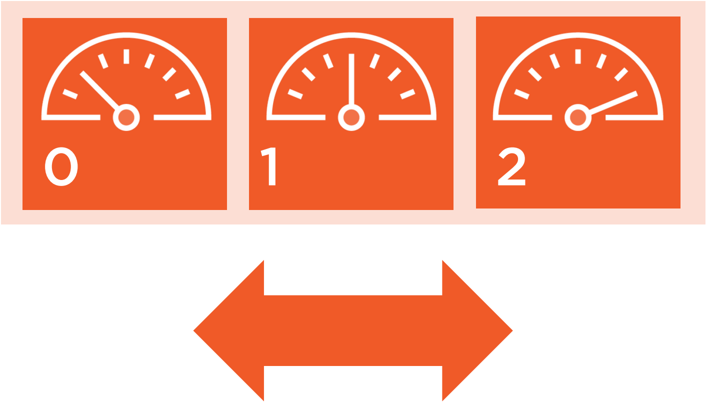
Can contain multiple instances of a type

Numeric indexing

Can contain multiple instances of a type

Numeric indexing

Access individual items

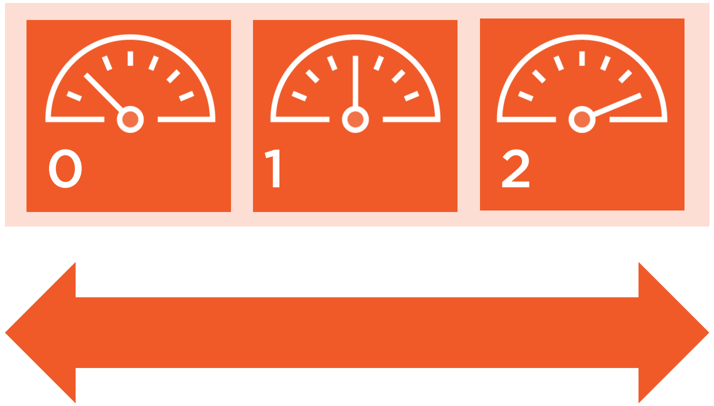


Can contain multiple instances of a type

Numeric indexing

Access individual items

Static or dynamic sizing

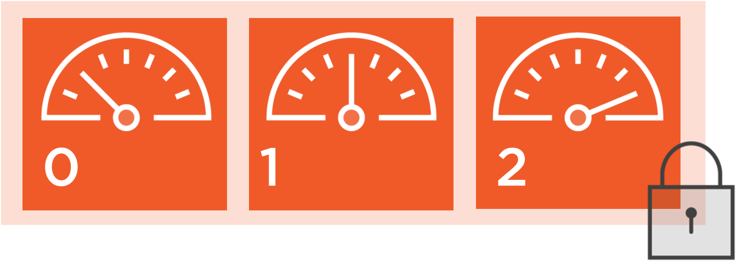


Can contain multiple instances of a type

Numeric indexing

Access individual items

Static or dynamic sizing



Can contain multiple instances of a type

Numeric indexing

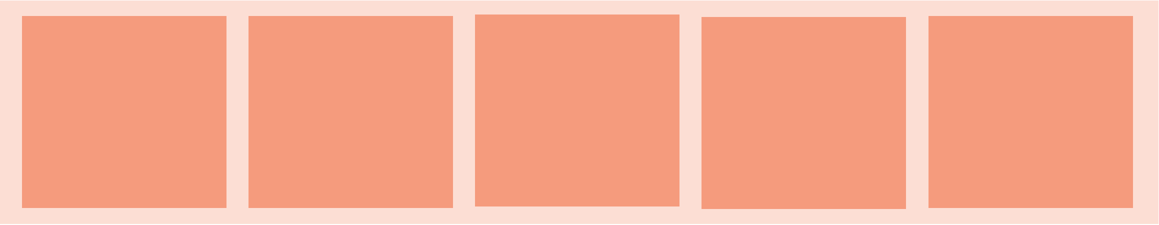
Access individual items

Static or dynamic sizing

Fixed size once created

```
Reading[] readings = new Reading[5];
```

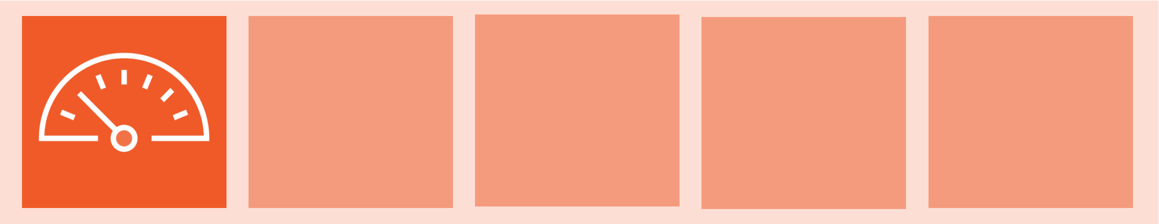
# Creating an Array



```
Reading[] readings = new Reading[5];

readings[0] = Gauge.Read();
```

# Adding Data to an Array

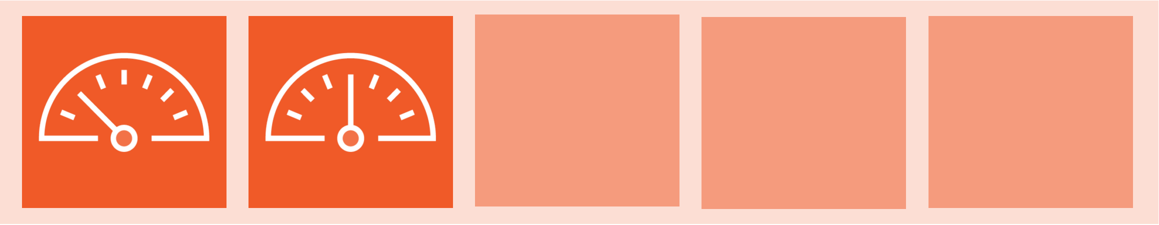


```
Reading[] readings = new Reading[5];

readings[0] = Gauge.Read();

readings[1] = Gauge.Read();
```

# Adding Data to an Array



```
Reading[] readings = new Reading[5];

readings[0] = Gauge.Read();

readings[1] = Gauge.Read();

readings[2] = Gauge.Read();

readings[3] = Gauge.Read();

readings[4] = Gauge.Read();
```

# Adding Data to an Array

```
Reading r = readings[2];
```

# Accessing Array Data

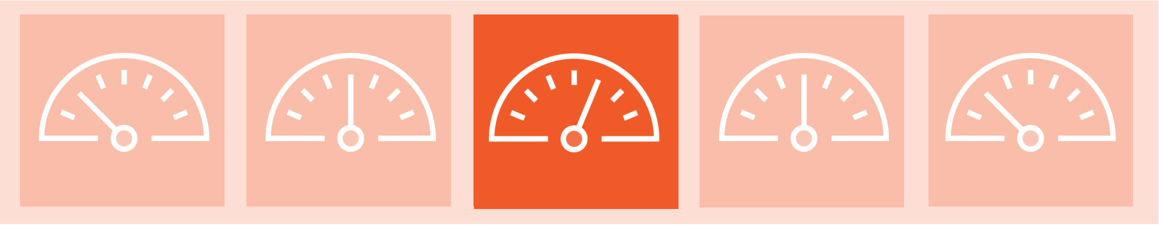
```
Reading r = readings[2];


for(int i = 0; i < 5; i++) {
    Reading r = readings[i];
}
```

# Accessing Array Data

```
readings[2] = Gauge.Read();
```

# Updating Array Values



# Asymptotic Analysis of Algorithms

# Resources

### Operations

The number of times we need to perform some operations

### Memory

How much memory is consumed by the algorithms

### Others

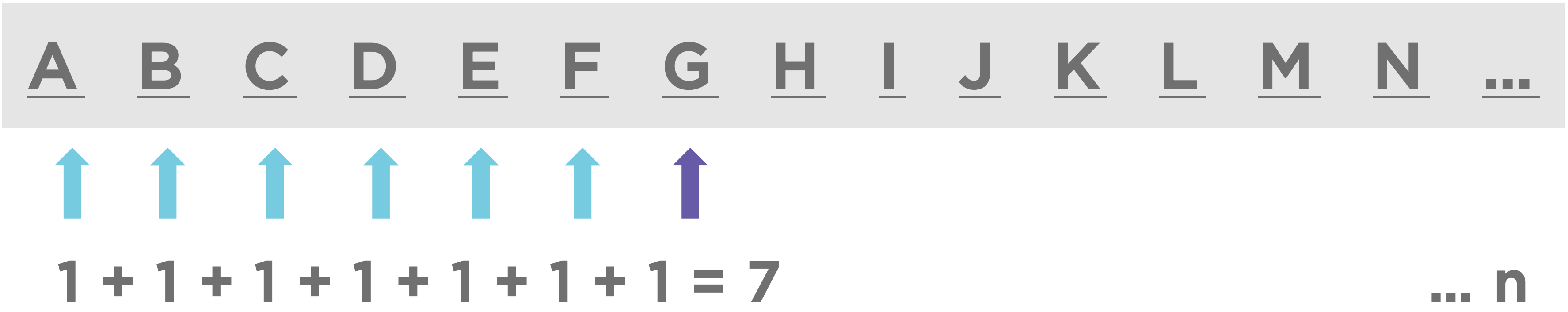
Network transfer, compression ratios, disk usage

```
char[] letters = ”ABCDEFGHIJKLMNOPQRSTUVWXYZ";


int index = 0;

while(letters[index] != 'G')

   index++;
```



A B C D E F G H I J K L M N …

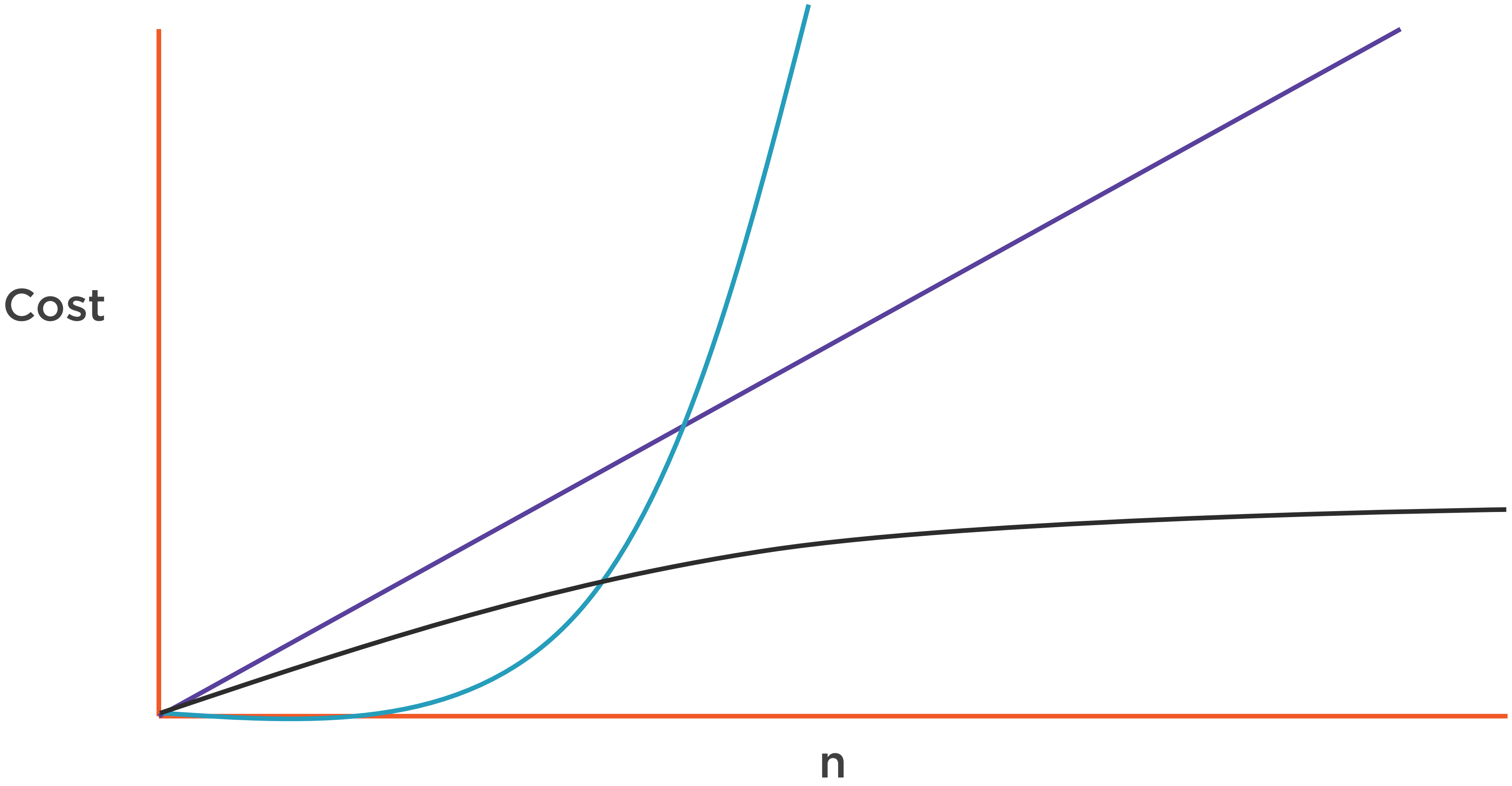
1 + 1 + 1 + 1 + 1 + 1 + 1 = 7

… n

# Big-O Notation

# $O(n)$

# How Does the Algorithm Scale?

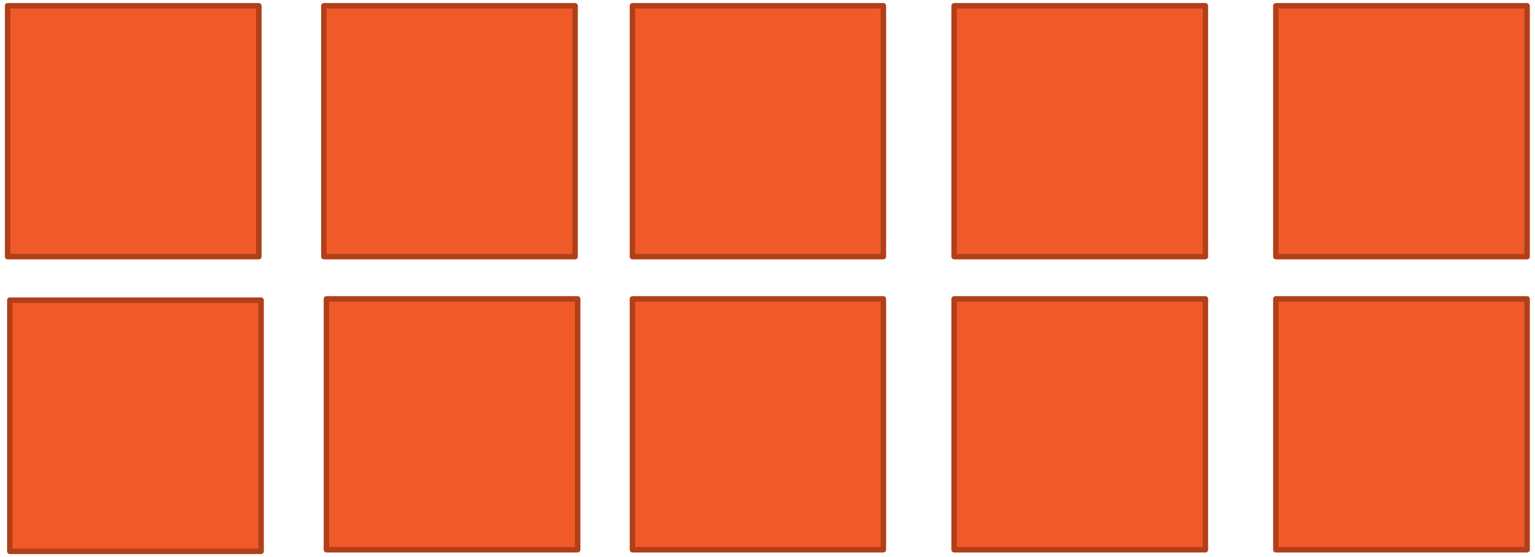


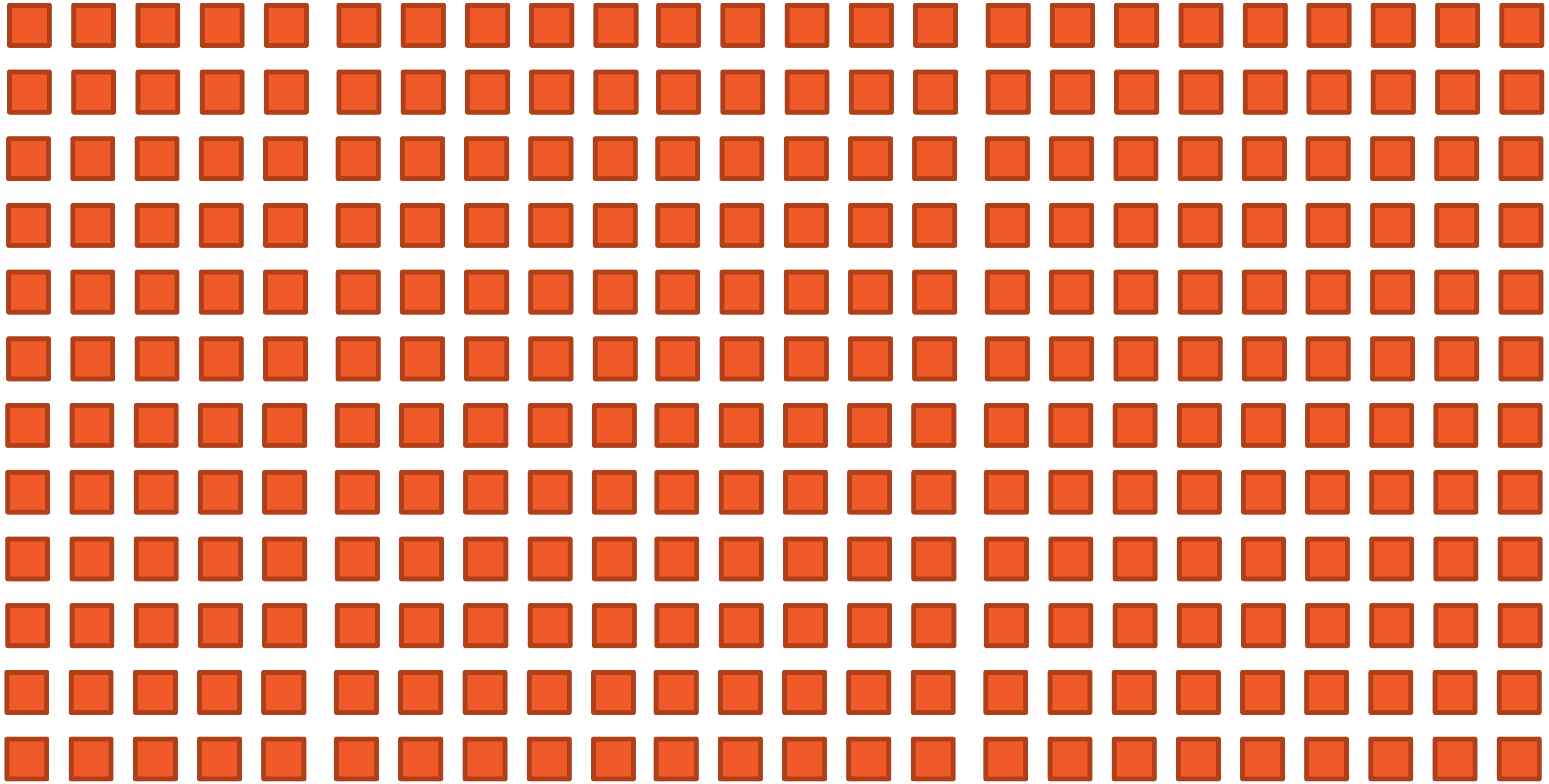
# Asymptotic Analysis

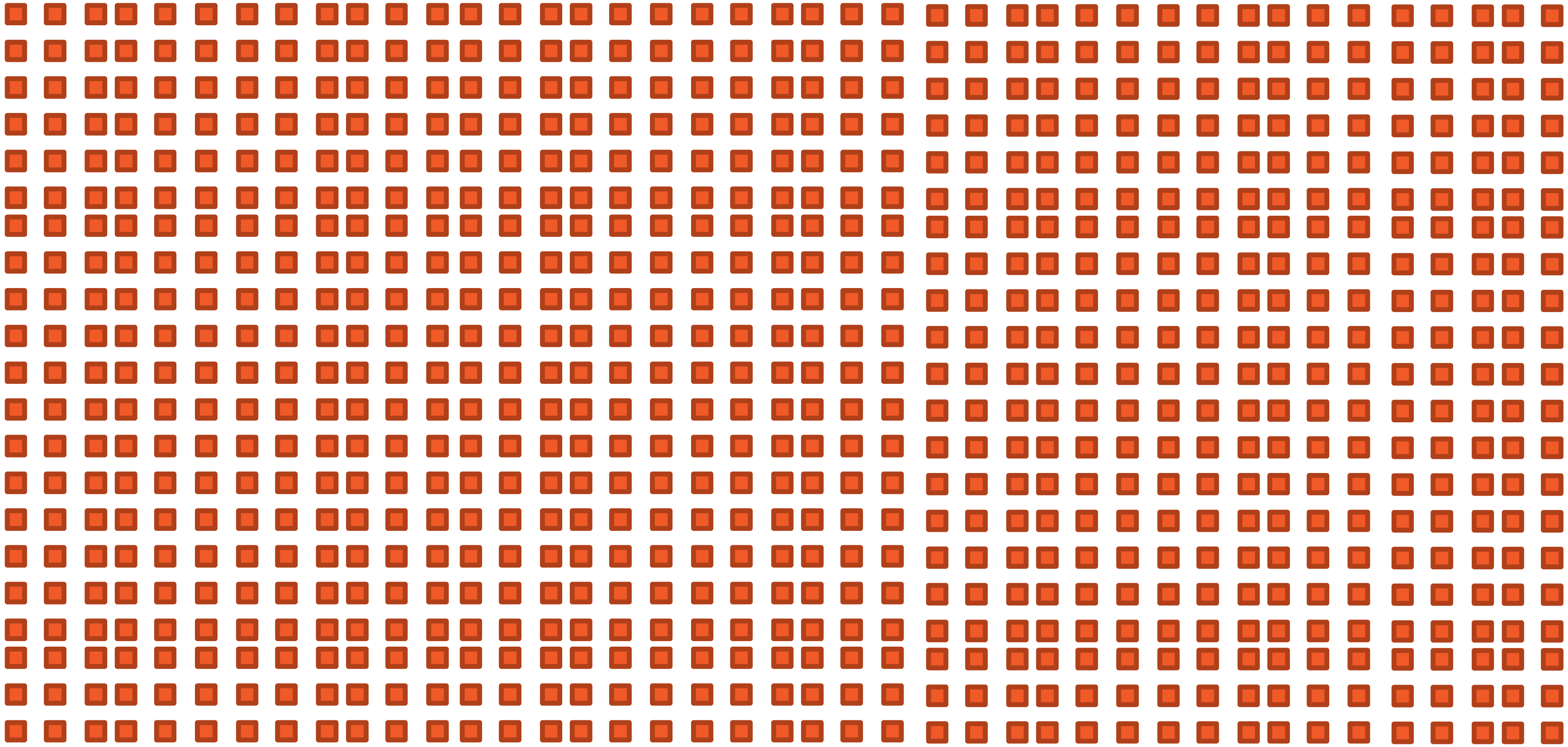
# Asymptote

The asymptote of a curve is a line where the distance between the curve and the line approach zero as they tend towards infinity.

Big numbers are more interesting







What is the upper limit?

It depends

(Hint: look at your domain)

$O(n+1)$ is $O(n)$

$O(2n)$ is $O(n)$

# Big-O Examples

# O(1)

The cost of the algorithm is unchanged by the input size.

# O(1) Growth

| Input Size | Cost |
|---|---|
| 1 | 1 |
| 100 | 1 |
| 1000 | 1 |
| 1000000 | 1 |

# O(n)

A function whose cost scales linearly with the size of the input.

```
char[] letters = "abcdefghijklmnopqrstuvwxyz";

for(int i = 0; i < letters.Length; i++) {
   Console.WriteLine(letters[i]);
}
```

# O(n)

Iterating over a collection of data once often indicates an O(n) algorithm.

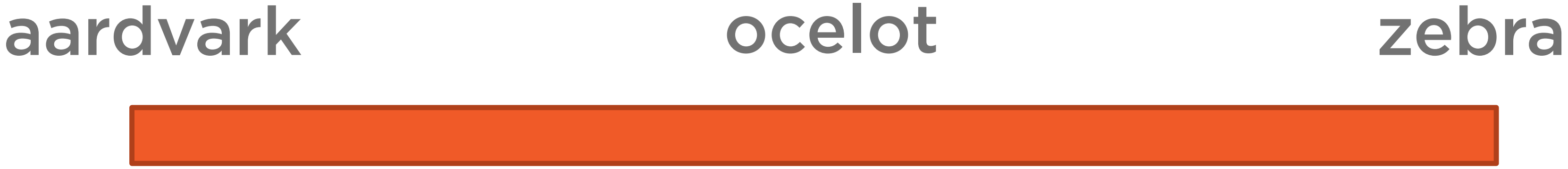
# O(n) Growth

| Input Size | Cost |
|---|---|
| 1 | 1 |
| 100 | 100 |
| 1000 | 1000 |
| 1000000 | 1000000 |

# O(log n)

A function whose cost scales logarithmically with the input size

# O(log n)

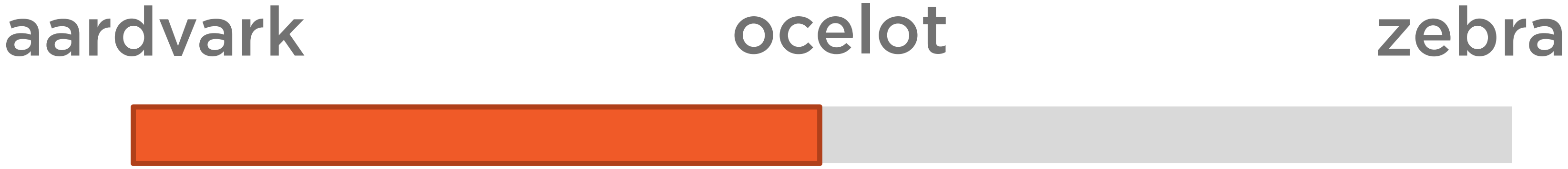


aardvark

ocelot

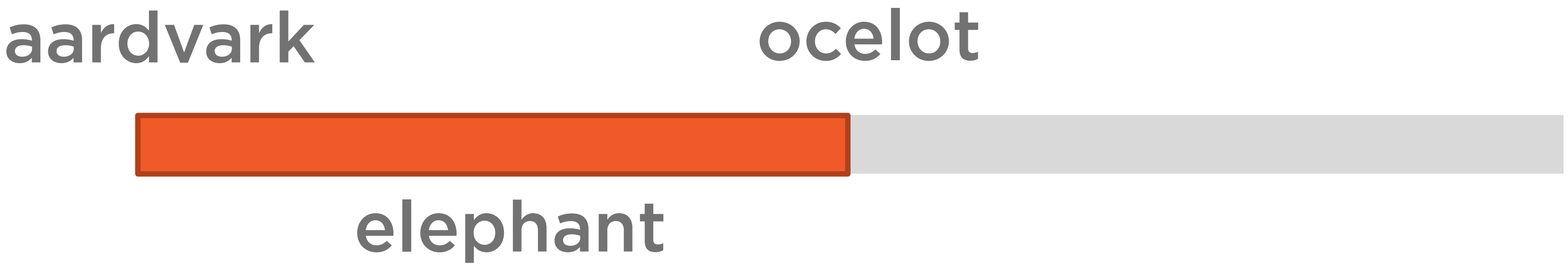
zebra

# O(log n)



aardvark ocelot zebra

# O(log n)

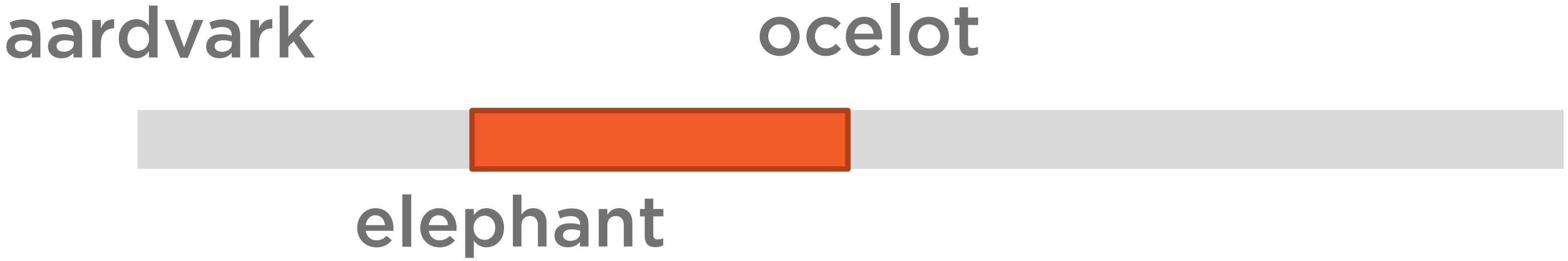


aardvark

ocelot

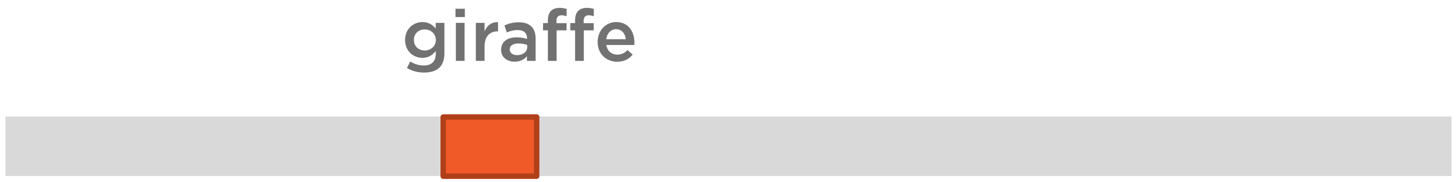
elephant

# O(log n)



# O(log n)

# O(log n)



giraffe

# O(log n) Growth

| Input Size | Cost |
|---|---|
| 1 | 1 |
| 10 | 1 |
| 1000 | 3 |
| 1000000 | 6 |

# $O(n^2)$

A function that exhibits quadratic growth relative to the input size.

```
void quad(char[] input, int count) {
    for (int i = 0; i < count; i++)
        for (int x = 0; x < count; x++)
            process(input, i, x);
}
```

# $O(n^2)$

A doubly-nested loop is an indication that you might have an $O(n^2)$ algorithm.

# $O(n^2)$ Growth

| Input Size | Cost |
|---|---|
| 1 | 1 |
| 10 | 100 |
| 1000 | 1000000 |
| 1000000 | 1e+12 |

# O(nm)

A function which has two inputs that contribute to the growth

```
void nm(char[] n, int nc, char[] m, int mc) {
    for (int i = 0; i < nc; i++)
        for (int x = 0; x < mc; x++)
            process(n[i], m[x]);
}
```

# O(nm)

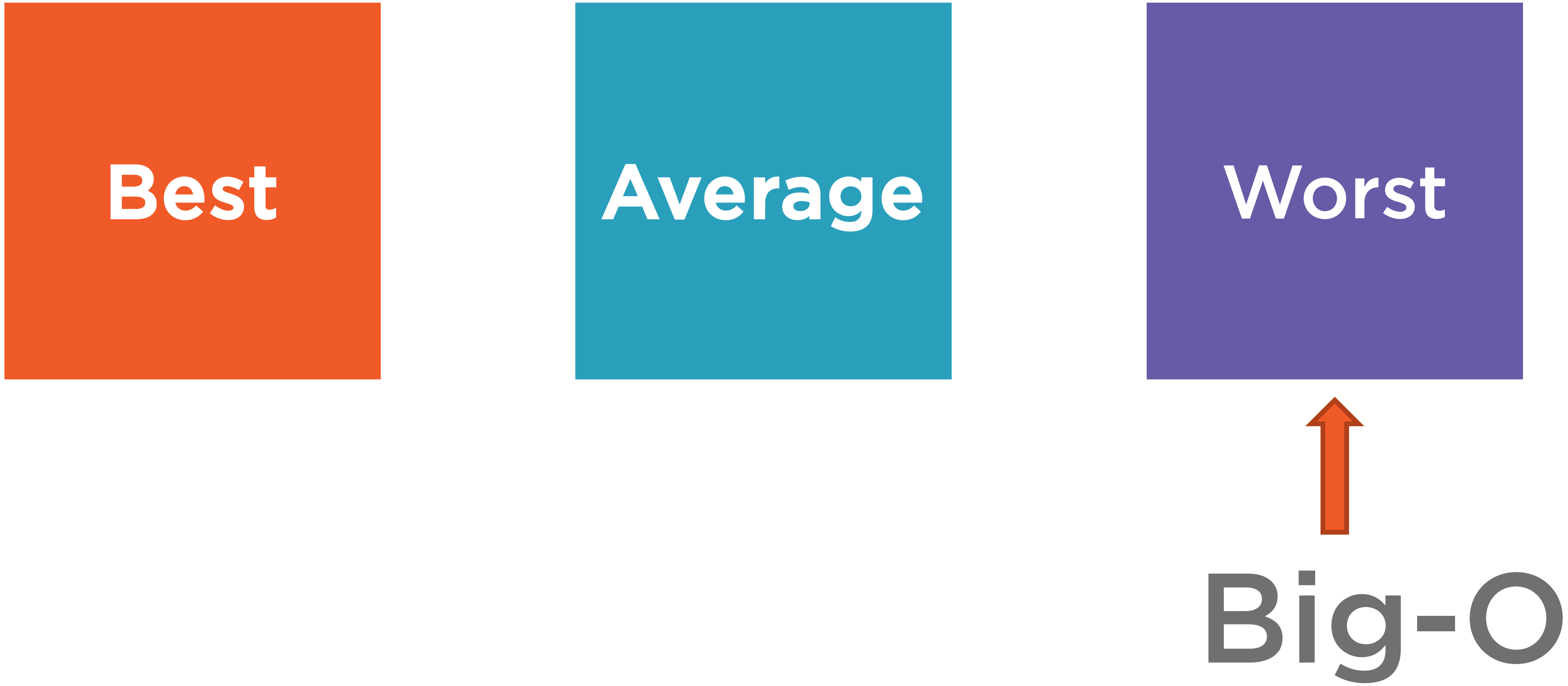
A nested loop that iterates over two distinct collections of data might indicate an O(nm) algorithm.

Predicting behavior means understanding your domain.

# Relative Timing

| Big-O | Elapsed Time |
|---|---|
| O(1) | 1 ms |
| O(log n) | 6 ms |
| O(n) | 16.67 minutes |
| O(nm) | (16.67 * m) minutes |
| O($n^2$) | 11.57 days |
| O($n^3$) | 3.16888e7 years |

This seems bad!

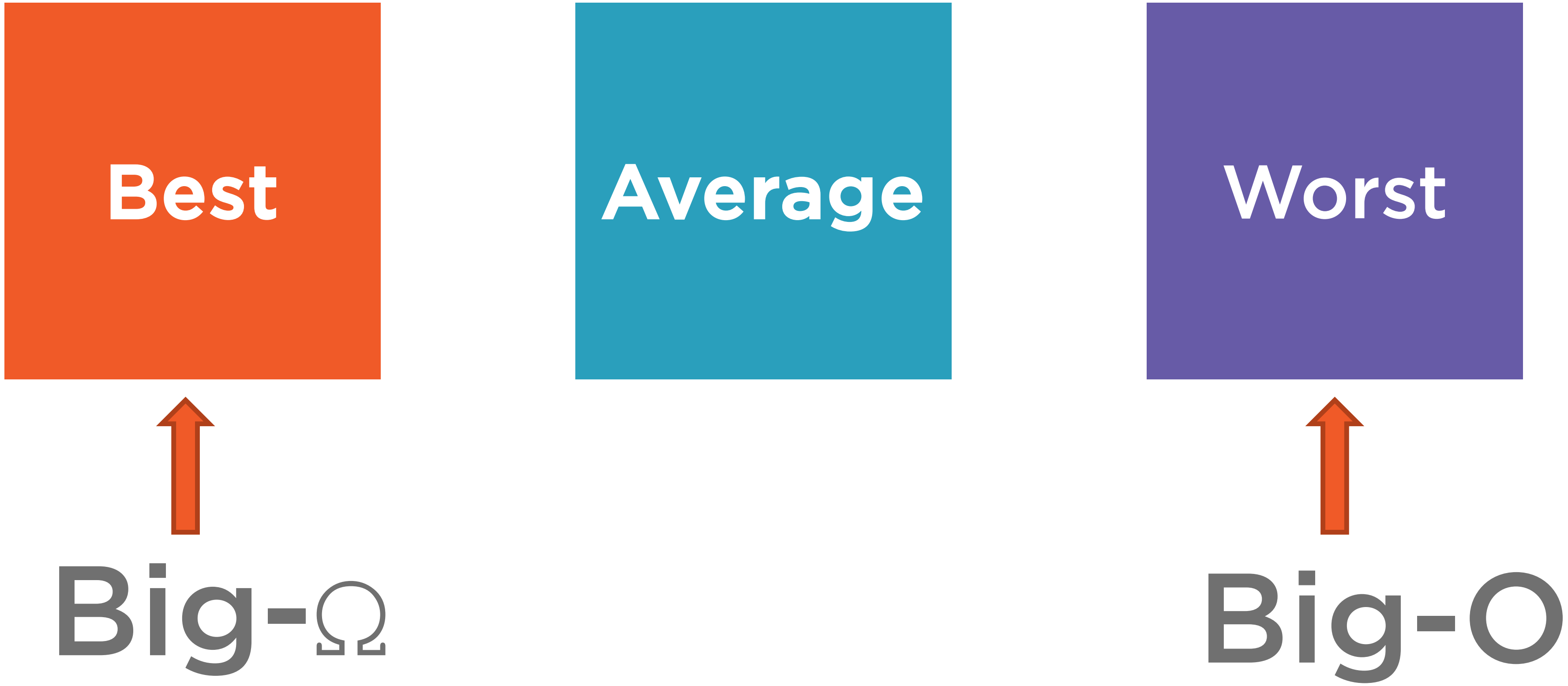


Best

Average

Worst

Big-O



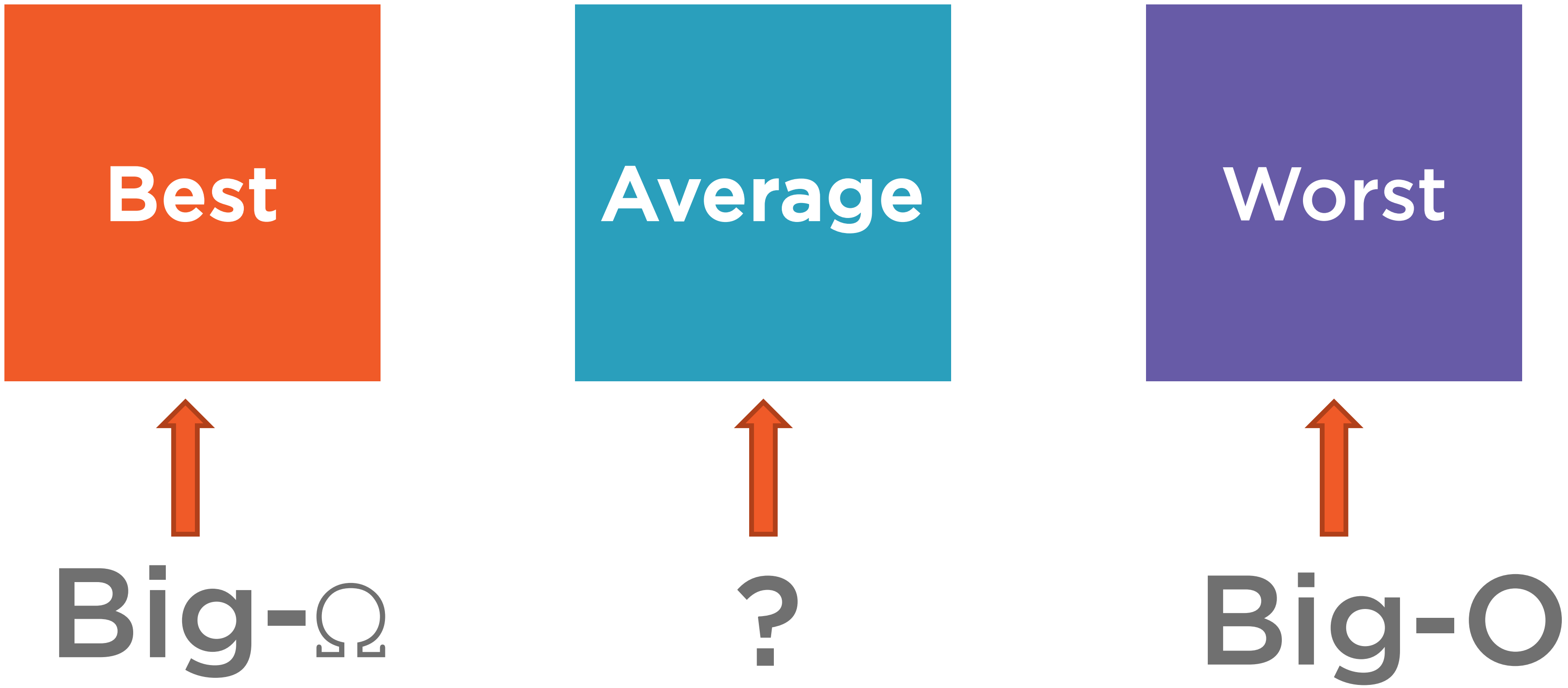
Best

Average

Worst

Big-Ω

Big-O



Best

Average

Worst

Big-Ω

?

Big-O

So we just use Big-O

# Demo

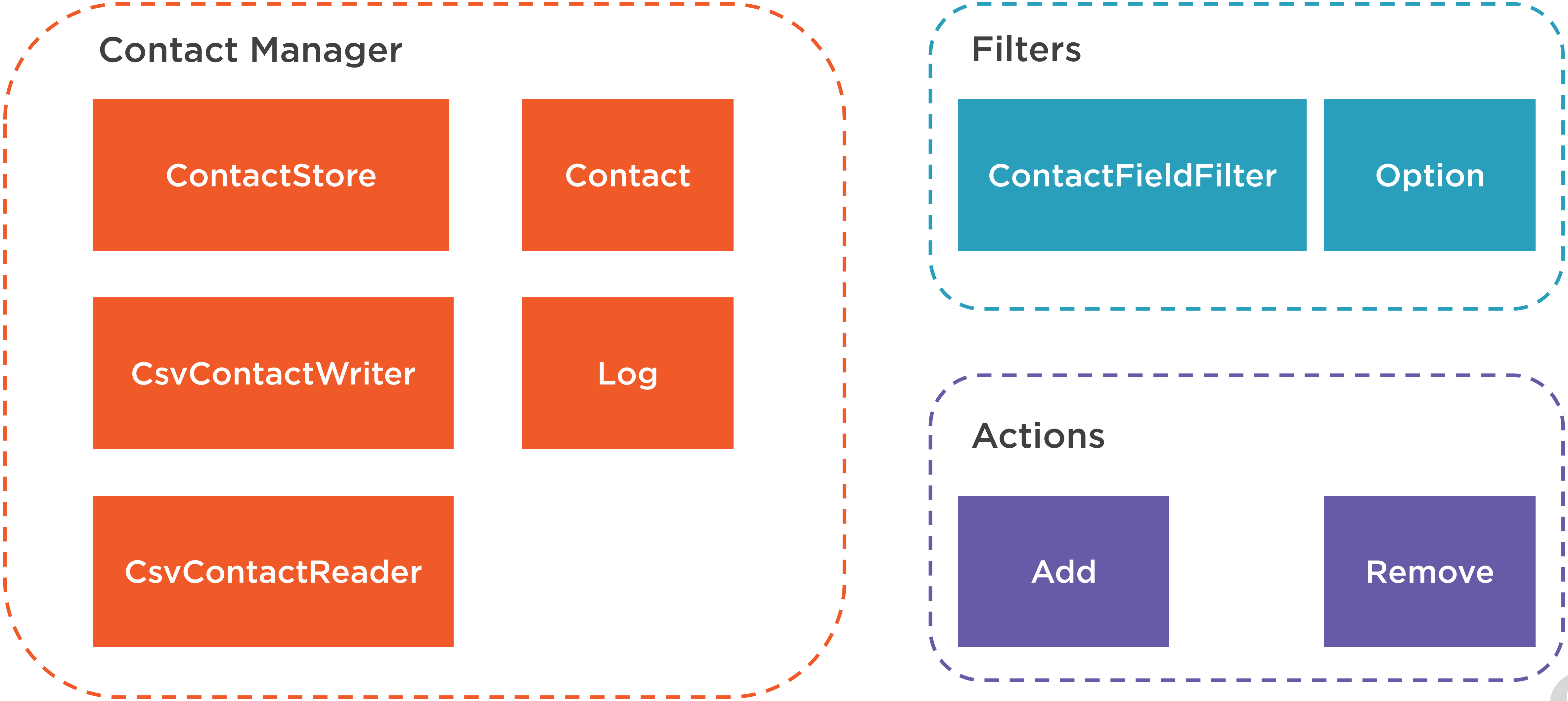
**Contact Manager**

- Command line application
- Written in C#

**Operations**

- Adding and removing contacts
- Searching by various criteria
- Loading and saving to disk

# Contact Manager (Library)



Contact Manager

ContactStore

Contact

CsvContactWriter

Log

CsvContactReader

Filters

ContactFieldFilter

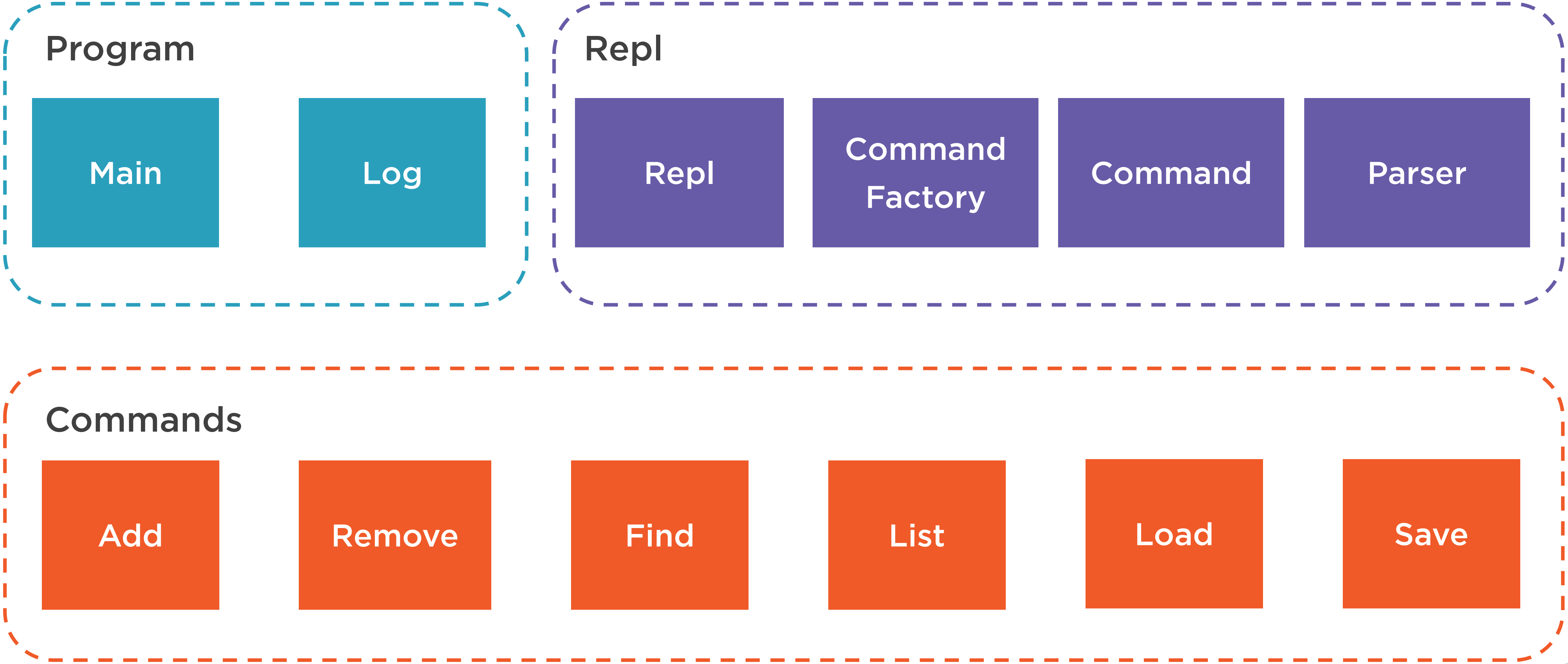
Option

Actions

Add

Remove

# Contact Manager (Application)



Program

Main

Log

Repl

Repl

Command Factory

Command

Parser

Commands

Add

Remove

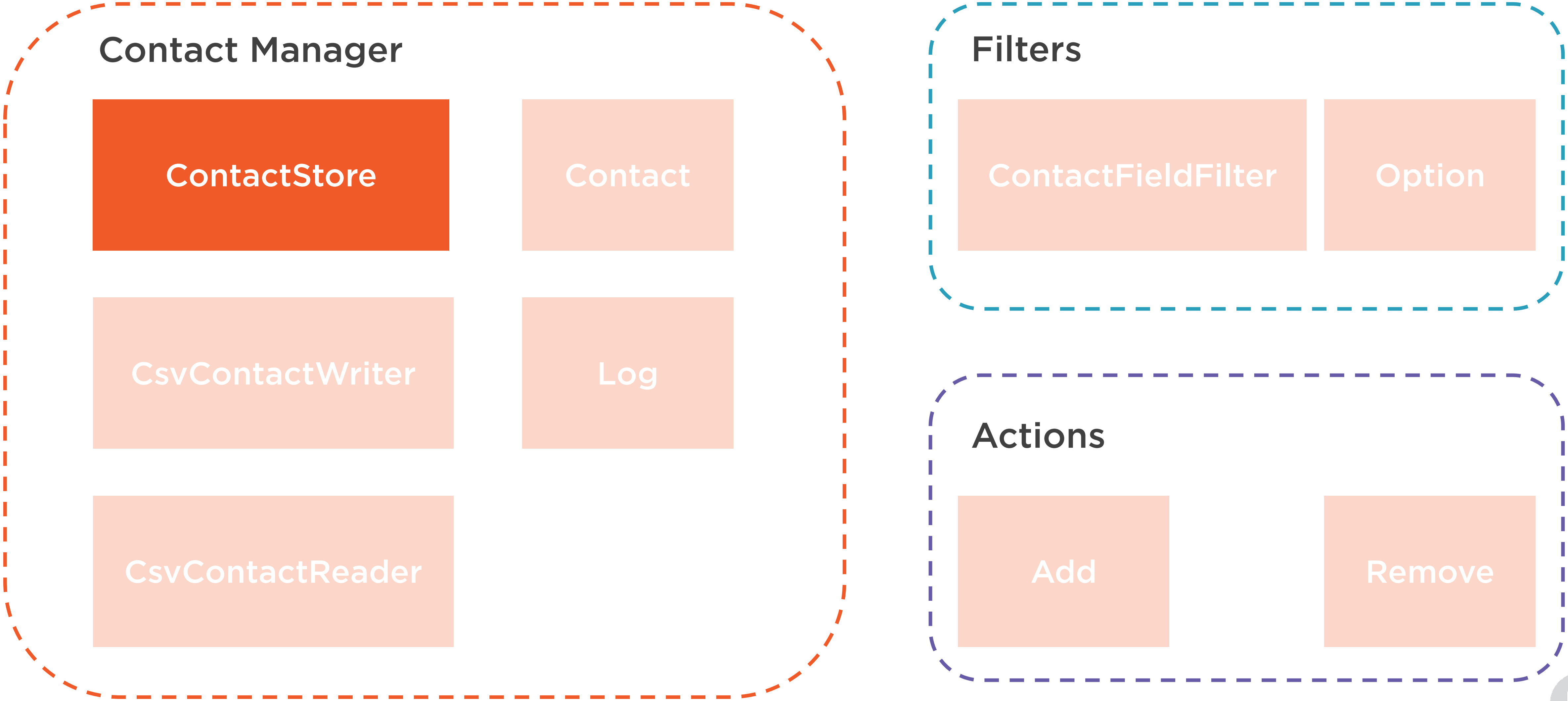
Find

List

Load

Save

# Contact Manager (Library)



Contact Manager

ContactStore

Contact

CsvContactWriter

Log

CsvContactReader

Filters

ContactFieldFilter

Option

Actions

Add

Remove