Bridging the Gap - Functional Programming in Objective C

Chris Woodard
Tampa Bay Cocoaheads

Why Functional Programming?

Complex / Tangled Code

- Written mainly with programming statements
- Very high-dimension program state
- Any part of the code can modify state, creating dependencies
- Complex mental model needed

Complex / Tangled Code

- Hard to read
- Hard to reason about
- Hard to extend
- Hard to fix
- Hard to test

Complex / Tangled Code

- Code is read much more often than written
- Human beings have a mental bottleneck
  - Limited-capacity working memory; only so many similar items can fit at one time
- Information has to pass through this bottleneck to be integrated with our knowledge base

The Effects of Tangled Code On Human Coders

"The Magical Number Seven, Plus or Minus Two"
George Miller, http://www.musanim.com/miller1956/

- Human beings can only keep 5-9 items in their working memory before their performance on recall degrades significantly.

- Structuring the digits (phone#, social security#, etc) improved recall ability.

- "By organizing the stimulus input simultaneously into several dimensions and successively into a sequence or chunks, we manage to break (or at least stretch) this informational bottleneck"

Preventing Complex / Tangled Code

- Add structure to the code in the right places.
- Avoid the extra dependencies by minimizing or eliminating shared state.
- Adopt *functional style*.

“Functional Style”

Functional Programming “Rules”

1. Program with Short Functions / Methods more than Statements

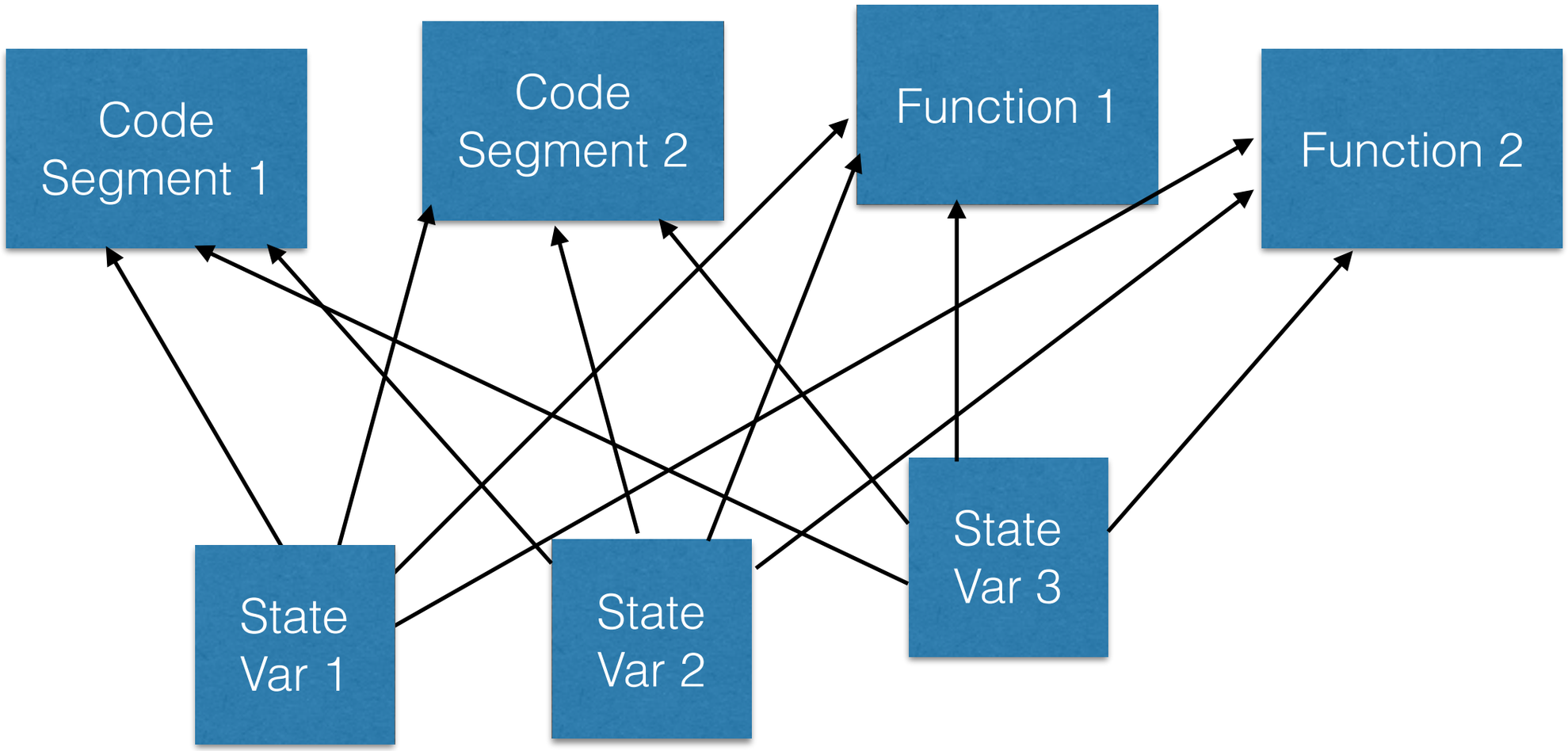
```
-(void)viewDidLoad {
  …
  NSString *stringList =
  [strings
  componentsJoinedByString
  :@“* “];
  _stringsLabel.text =
  stringList;

  …
}
```

```
-(void)viewDidLoad {
 …
  _stringsLabel.text =
  joinStrings(strings);
 …
}
```

2. Functions Should Not Use Shared Mutable State

- If a variable's value affects the execution of code, it's called "state".
- If two or more segments of code depend on the same value it's "shared state".
- If that state can be modified by any of those segments of code it's "shared mutable state".

Effects of Using Shared Mutable State



Lots of dependency paths

3. Functions Should Not Cause Side Effects

Side Effects include:

- modifying an external global variable or object property in the body of the function or method

- modifying an object referenced by an address parameter

FunctionalProgramming Techniques

Functions Only Use Parameter Values and Return Values

Antipatterns

In/Out Parameter

```
-(BOOL)removeItemAtURL:(NSURL *)URL error:(NSError **)error;
```

Unnecessary Completion Block

```
-(BOOL)loadImageAtURL:(NSURL *)URL completion:^(NSError
*err, NSString *imgName;
```

Functions Only Use Parameter Values and Return Values

Antipattern Antidote

```
NSDictionary *results = [self removeItemAtPath:cachedImagePath];
```

@{ @“OK”:YES, @“Error”:nil}

```
NSDictionary results = [self loadImageAtURL:logoURL];
```

@{ @“OK”:YES, @“Error”:nil, @“ImgName”:@“grumpyCat.jpg”}

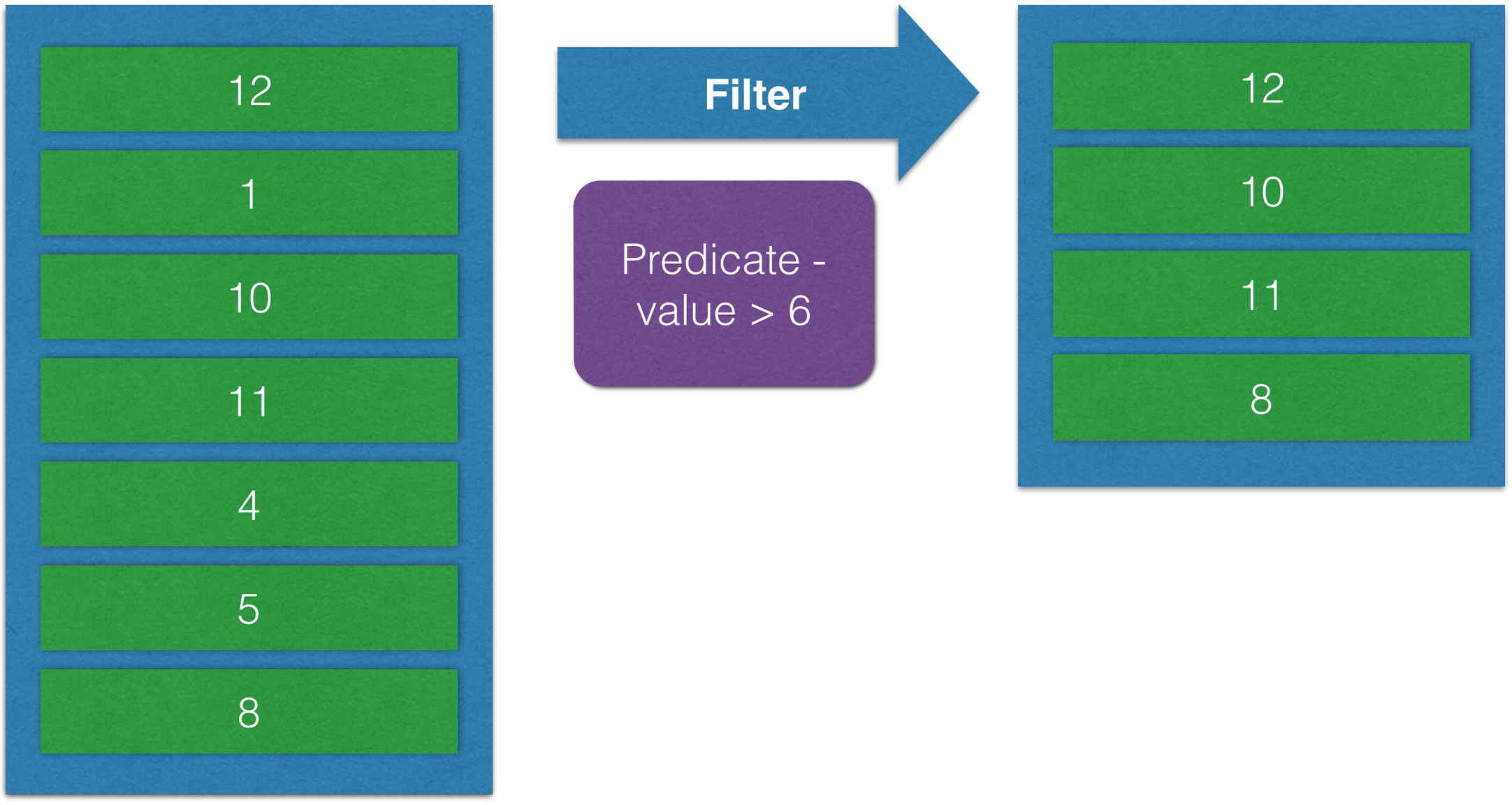
Think In Terms Of Lists, Not Arrays

- *List comprehensions* are operations performed on lists.

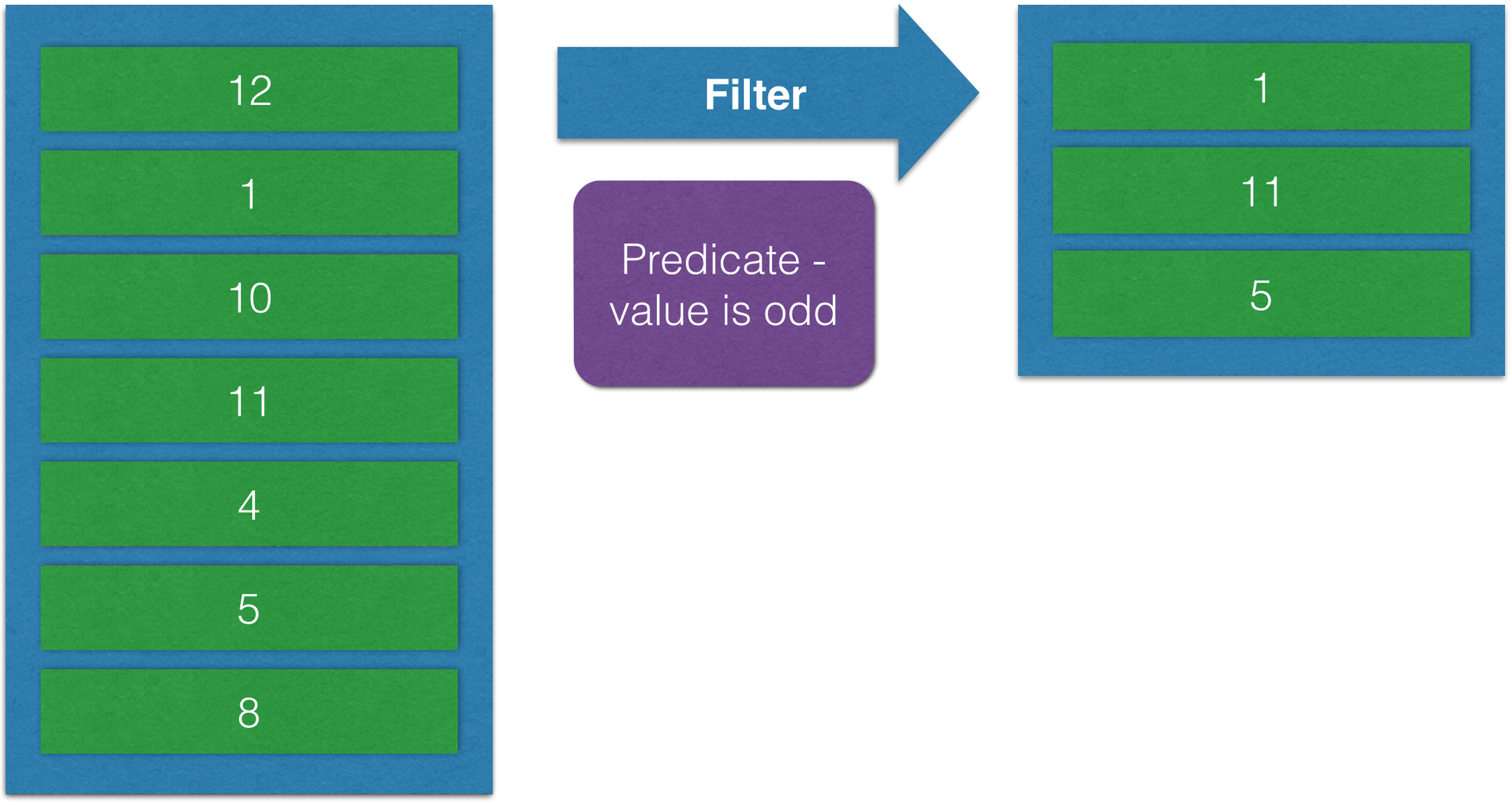
- A list comprehension operating on a list *produces a new list*.

- The three most common are *filter*, *map* and *reduce.*

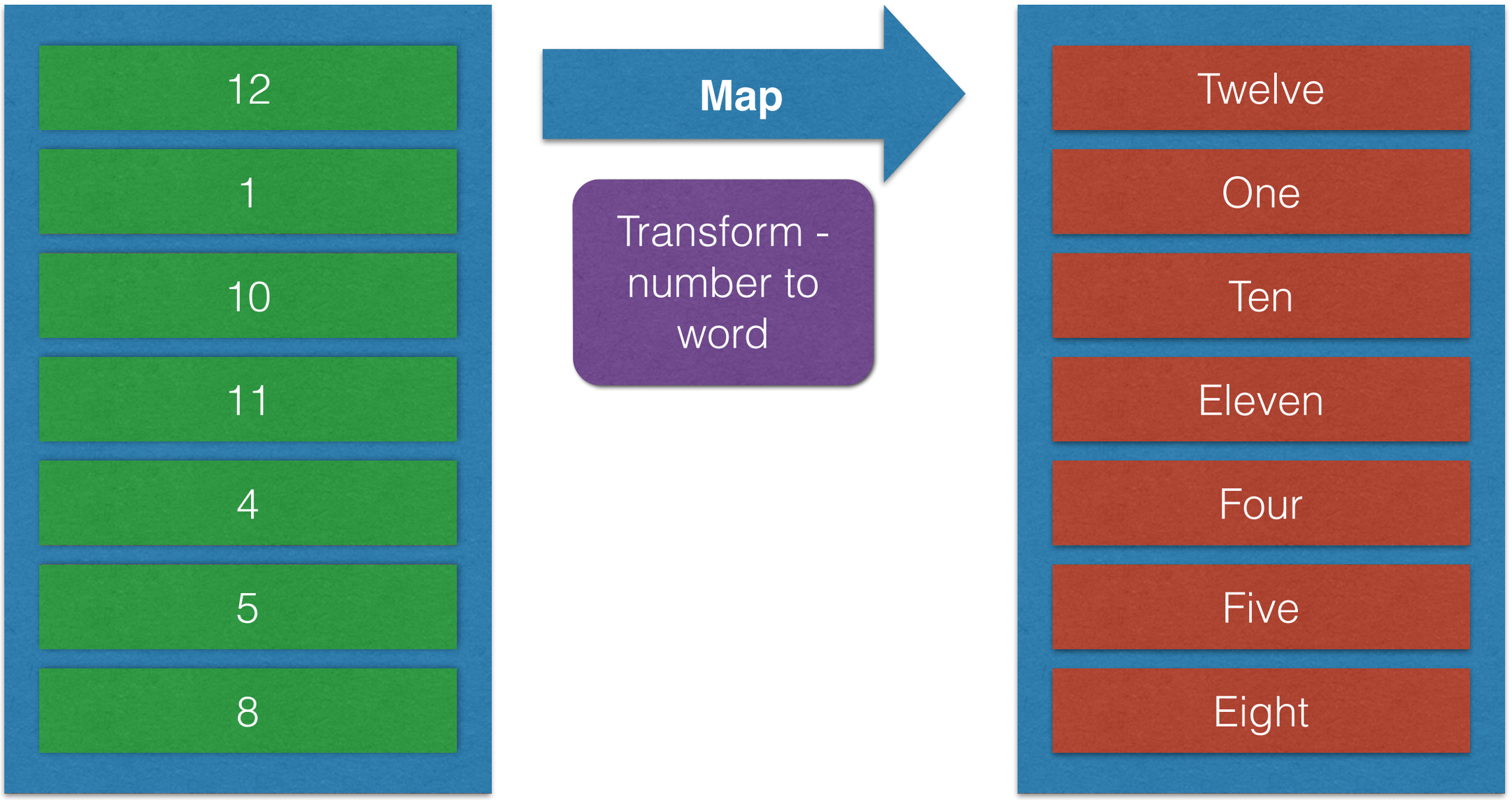
Filter



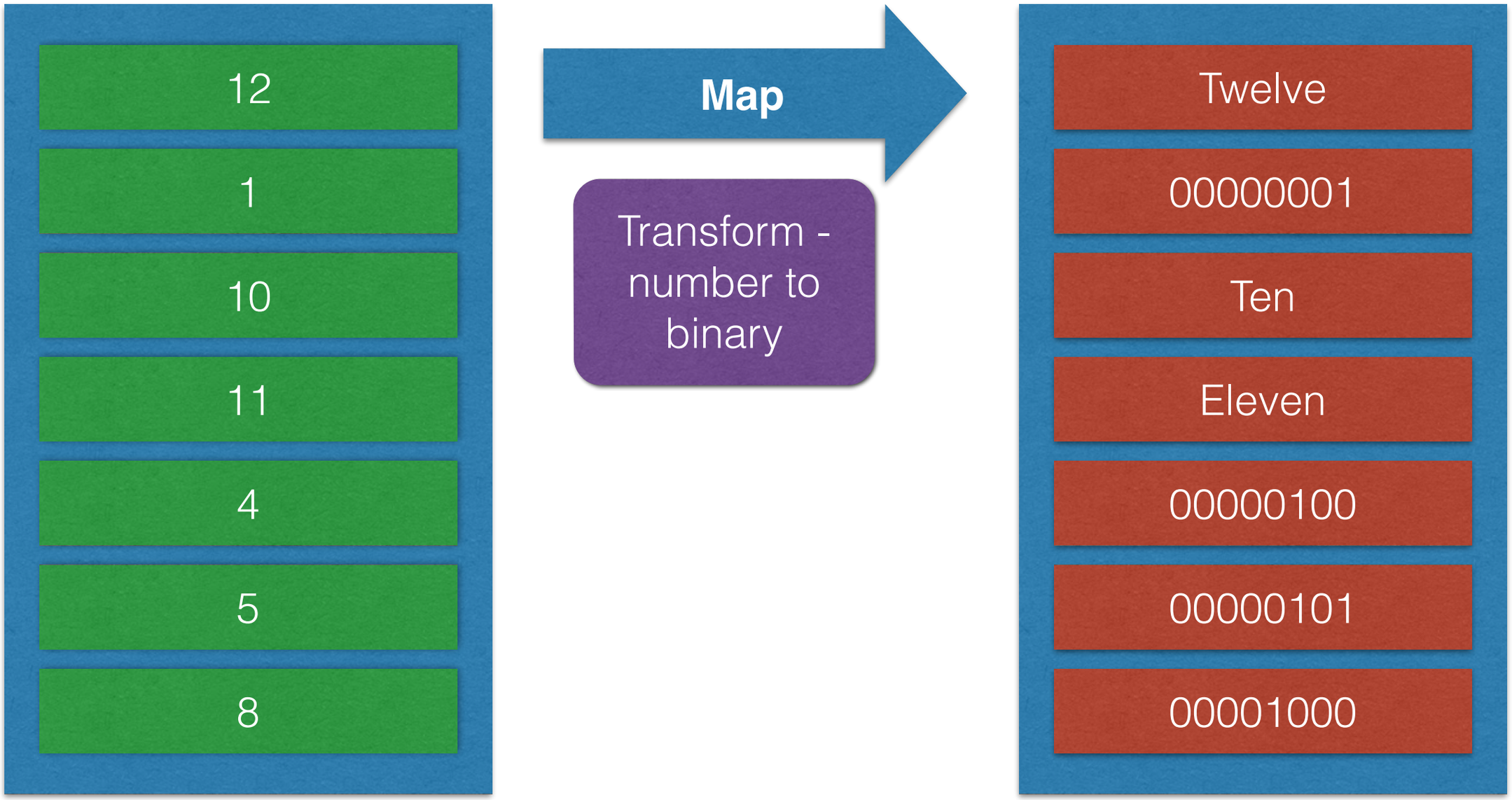
Filter



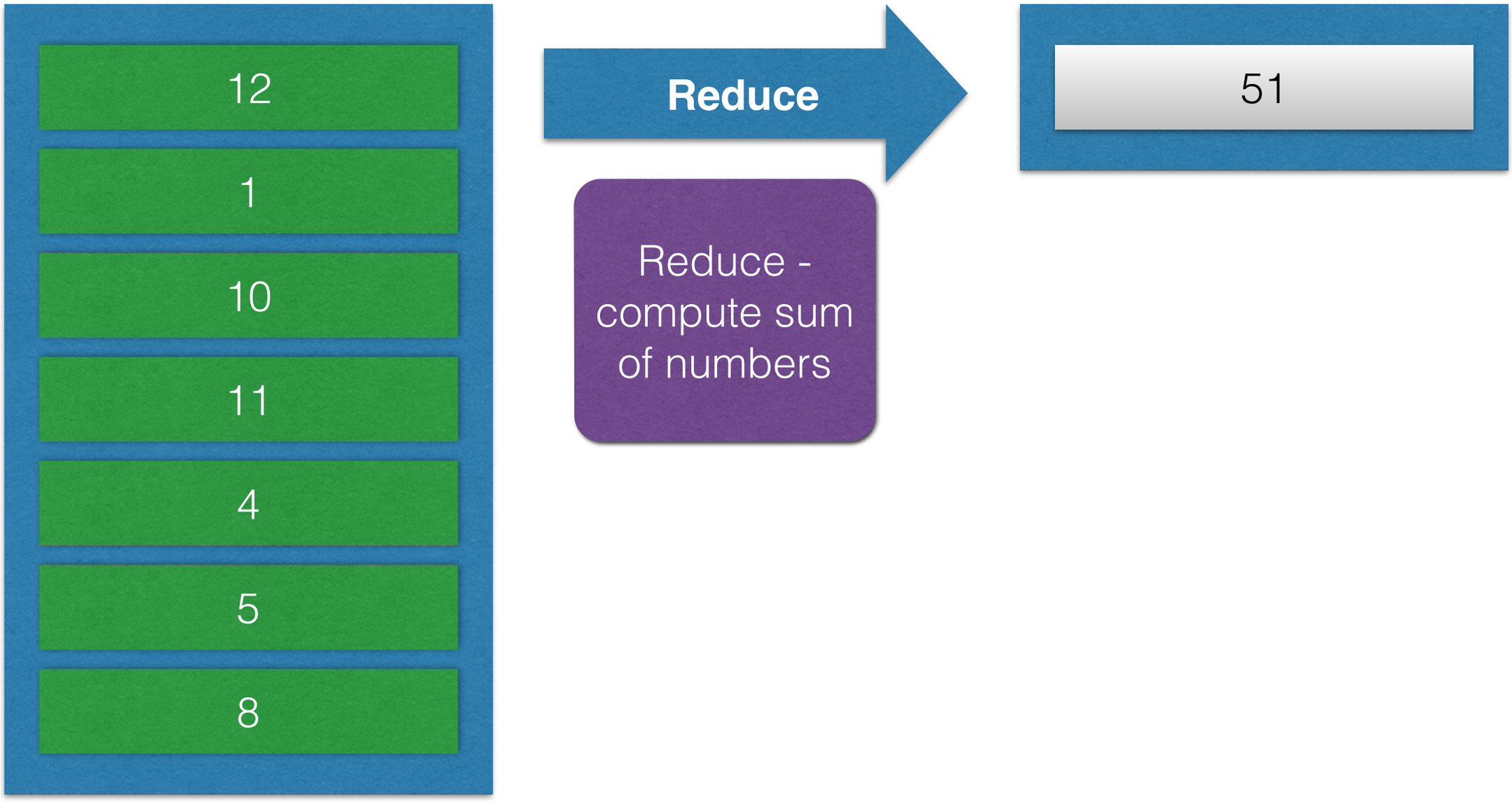
Map


12
1
10
11
4
5
8
Map
Transform - number to word
Twelve
One
Ten
Eleven
Four
Five
Eight

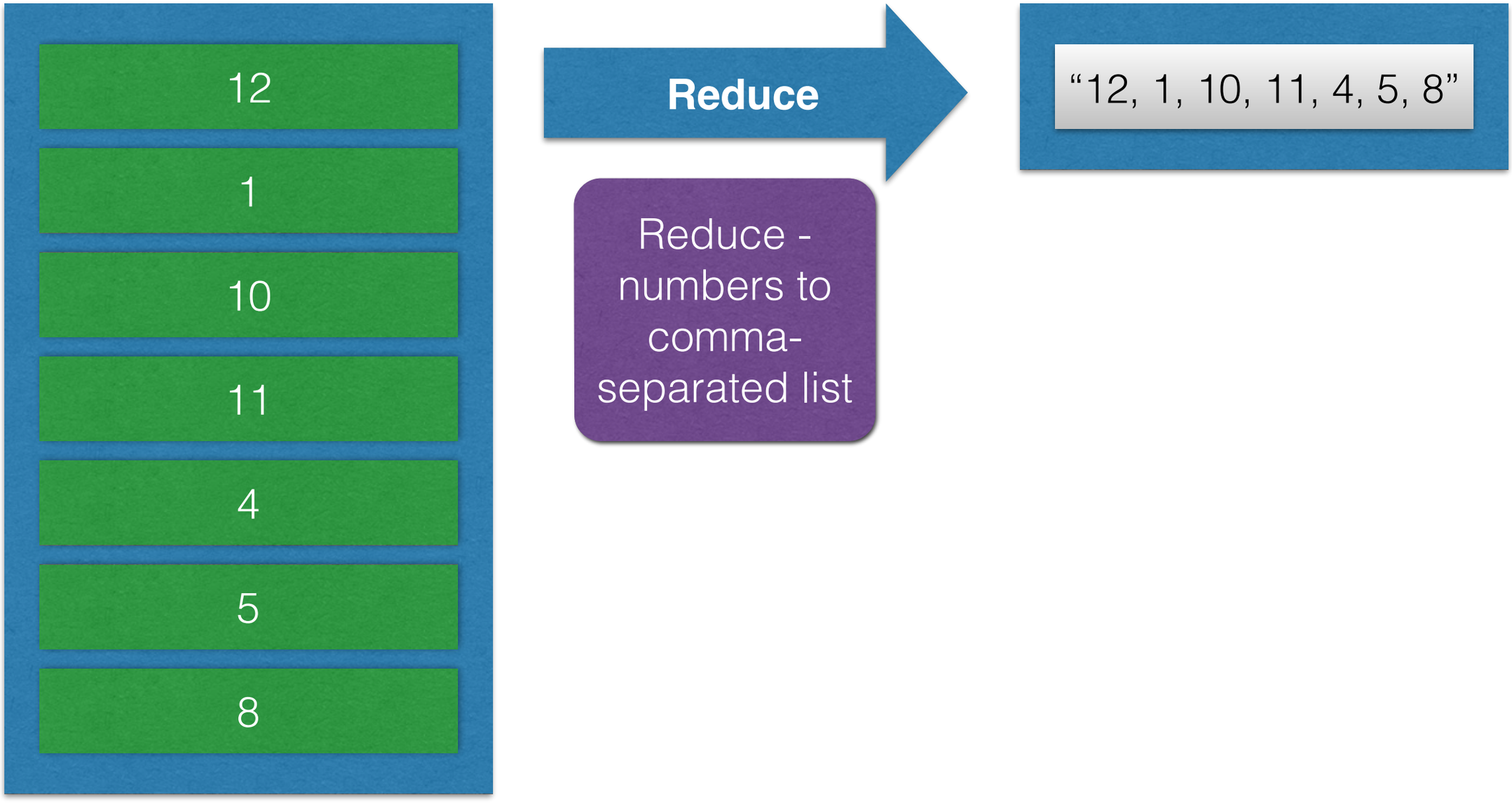
Map


12
1
10
11
4
5
8
Map
Transform - number to binary
Twelve
00000001
Ten
Eleven
00000100
00000101
00001000

Reduce



Reduce


12
1
10
11
4
5
8
Reduce
Reduce - numbers to comma-separated list
“12, 1, 10, 11, 4, 5, 8”

Great. How do we do *that*?

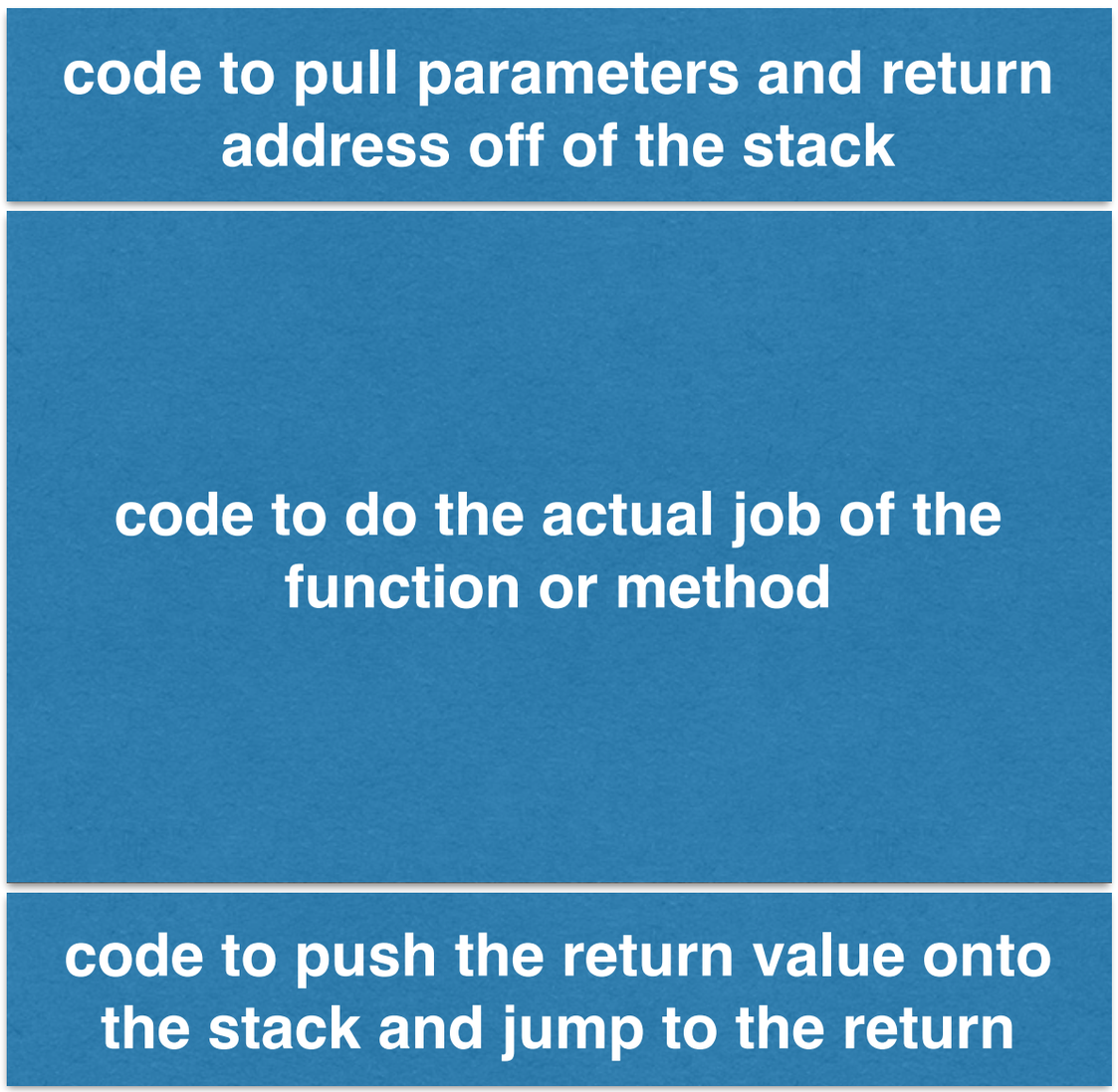
Blocks

Functions Are “First Class Objects”

- Functions can be assigned as values
- Functions can be passed in as values
- How does this work?

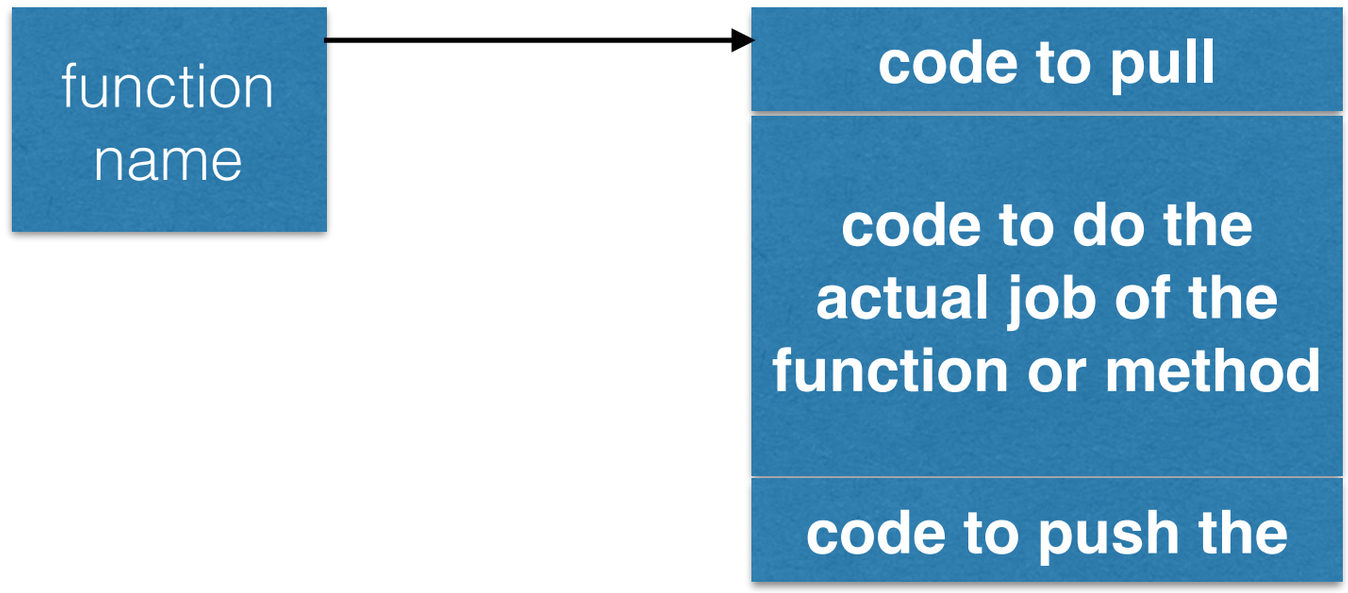
“All Functions and Methods in Objective C are Blocks”

- Blocks, functions and methods are sections of compiled code.
- They all have a common structure.

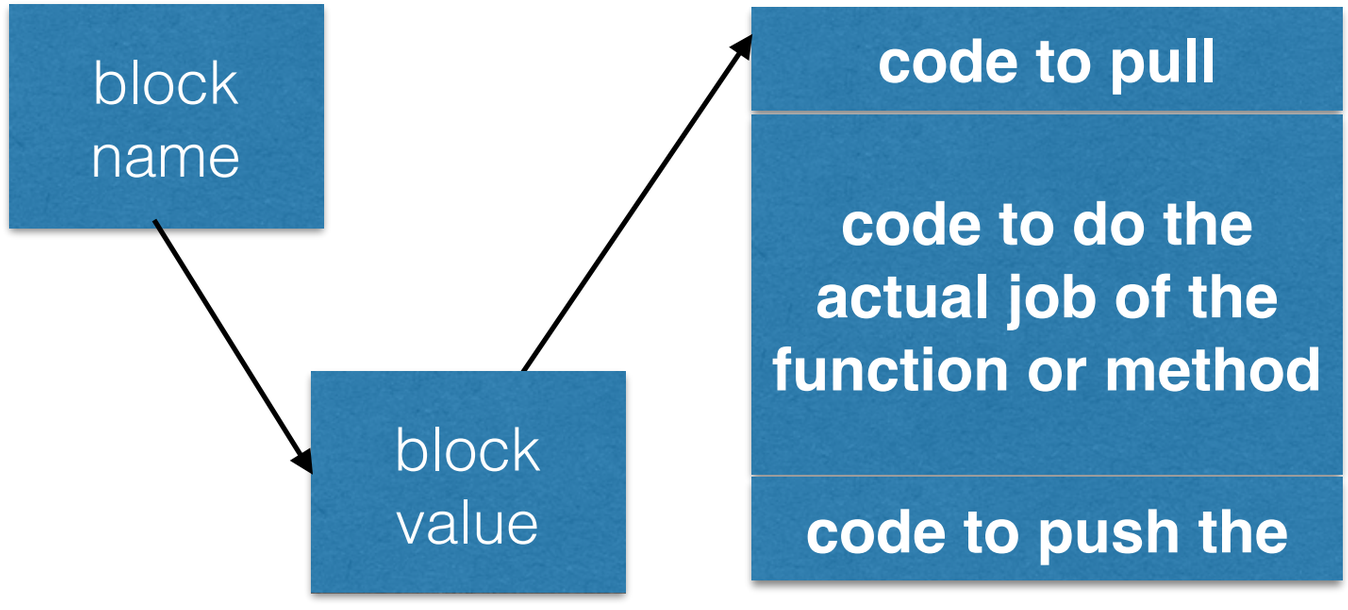


“All Functions and Methods in Objective C are Blocks”

- *Functions* have readonly names that directly reference the code.



- *Blocks* and *function pointers* indirectly reference the code.



“All Functions and Methods in Objective C are Blocks”

Calls the block

Push parameter values onto stack

Push return address onto stack

“Jump” to the code

“the block”

Pop parameter values and return address off of the stack

Do whatever

Push return value onto the stack

“Jump” to the return address

Pop the return value off the stack

Do whatever…

“All Functions and Methods in Objective C are Blocks”

- Compiler needs to know how to push the arguments on to the stack
- Compiler needs to know how to pop the return value off the stack
- For functions and methods this is already baked into the cake via *prototypes* and *class interfaces*.

How does that work for Blocks?

“That Gosh-Darn Block Syntax”

As a **local variable**:

```
returnType (^blockName)(parameterTypes) = ^returnType(parameters) {…};
```

As a **property**:

```
@property (nonatomic, copy) returnType (^blockName)(parameterTypes);
```

As a **method parameter**:

```
- (void)someMethodThatTakesABlock:(returnType (^)(parameterTypes))blockName;
```

As an **argument to a method call**:

```
[someObject someMethodThatTakesABlock:^returnType (parameters) {…}];
```

As a **typedef**:

```
typedef returnType (^TypeName)(parameterTypes);
TypeName blockName = ^returnType(parameters) {...};
```

http://goshdarnblocksyntax.com

Show Da Code

Takeaways

- Code should be readable and understandable.
- Keep your methods and functions short.
- Adopting a functional style helps a lot with that.
- Learn how to use block syntax.

Takeaways

- Don’t use shared state or you will end up with something you don’t want.

- You can write functional-style code in any language that supports anonymous functions.

Resources

- http://goshdarnblocksyntax.com
- https://leanpub.com/b/thinkingfunctionallyinobjective-c
- http://mentalfaculty.tumblr.com/post/64952009090/the-lessons-of-functional-programming-for-cocoa
- http://bou.io/FunctionalProgrammingInObjectiveC.html
- http://psychclassics.yorku.ca/Miller/
- https://github.com/sciprojguy/360idev2015