Closures

aka… _blocks!

“STRAIGHT & FORWARD”: 1) take a chunk of code, 2) *enclose* it in a ***closure***!

```
// this is some code...

me.events = result;
me.orderedMonthKeys = order;
[me.tableView reloadData];
```

```
// Well... this is a closure !!!

^{

 // some code:

    me.events = result;
    me.orderedMonthKeys = order;
    [me.tableView reloadData];


}
```

POWER TO THE CLOSURES !

```
^{

 // some code:

     me.events = result;
     me.orderedMonthKeys = order;
     [me.tableView reloadData];


 }
```

- *They **capture the** state !!! (that is: local variables)*
- *Not only you can store a chunk of code*
- *But also you can preserve access to the all **SCOPE** !!!*

```
- (void)viewDidLoad
{
    [super viewDidLoad];


    __weak typeof(self) weakSelf = self;

    self.loadTableBlock = ^(BOOL success, NSDictionary *result, NSArray *order) {

                    [[NSOperationQueue mainQueue] addOperationWithBlock:
                        ^{
                         // some code:
                             weakSelf.events = result;
                             weakSelf.orderedMonthKeys = order;
                             [weakSelf.tableView reloadData];
                         }];

                };


    [self getTheEventsWithComponents:components
                    withCompletion:self.loadTableBlock];

}
```

This is another method!!!

Another method => Another scope

```
-(void)yearPickerViewControllerDidFinishPickingYear:(NSString *)year {

    NSDate* today = [NSDate new];
    NSCalendar* calendar = [NSCalendar currentCalendar];

    NSDateComponents* components = [calendar components:NSCalendarUnitYear
                                                 fromDate:today];
    components.year = [year integerValue];

    [self getTheEventsWithComponents:components
                        withCompletion:self.loadTableBlock];
}
```

So…

F.A.Q.

Why are they so useful ?

- *They reduce the amount of code you need to write, which reduces the amount of code you need to maintain and debug.*
- *Communication with blocks helps maintain a high level of encapsulation while keeping your code readable and concise.*

What are common use cases ?

- *Many animation related methods take block arguments.*
- *Often used in networking apis, sorting / mapping functions, and multithreading.*
- *As complition blocks they can let you know when a long operation has come to an end, or act as callbacks.*

What can I do with blocks?

- Pass a block as a *parameter / argument to a method.*
- *Store a block in a local variable, or a property as well.*
- You can also pass *additional arguments to the block* itself.
- They can also have a return value!!!

As a **method parameter**:

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

- -

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

```
[someObject doSmtWithBlock: ^returnType (parameters) {...}];
```

What can I do with blocks?

- Pass a block as an parameter / *argument to a method.*
- *Store a block in a local variable, or a property as well.*
- You can also pass *additional arguments to the block* itself.
- They can also have a return value!!!
- And of course you can nest them...

As a **local variable**:

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

- -

As a **property**:

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

What can I do with blocks?

- Pass a block as an parameter / *argument to a method.*
- *Store a block in a local variable,*
  or a property as well.
- You can also pass *additional arguments to the block* itself.
- They can also have a return value!!!
- And of course you can nest them...

What can I do with blocks?

- Pass a block as an parameter / *argument to a method.*
- *Store a block in a local variable, or a property as well.*
- You can also pass *additional arguments to the block* itself.
- They can also have a return value!!!
- And of course you can nest them...

```
(returnType (^)(parameterTypes))blockName;
```

(parameter)

```
returnType (^blockName)(parameterTypes);
```

(var declaration)

```
[Obj methodWithargument:^returnType (parameters) {...} ];
```

(argument)

What can I do with blocks?

- Pass a block as an parameter / *argument to a method.*
- *Store a block in a local variable, or a property as well.*
- You can also pass *additional arguments to the block* itself.
- They can also have a return value!!!
- And of course you can nest them...

What other languages support closures ?

Well… A LOT!!!

- Objective-C ..... block
- Swift ..... closure
- Ruby ..... lambda
- Java ..... lambda
- Javascript ..... lambda
- C (some libraries) ..... calback
- C# ..... delegate
- C++ ..... function obj

And many other languages ..... closure!!!

Most common pitfall

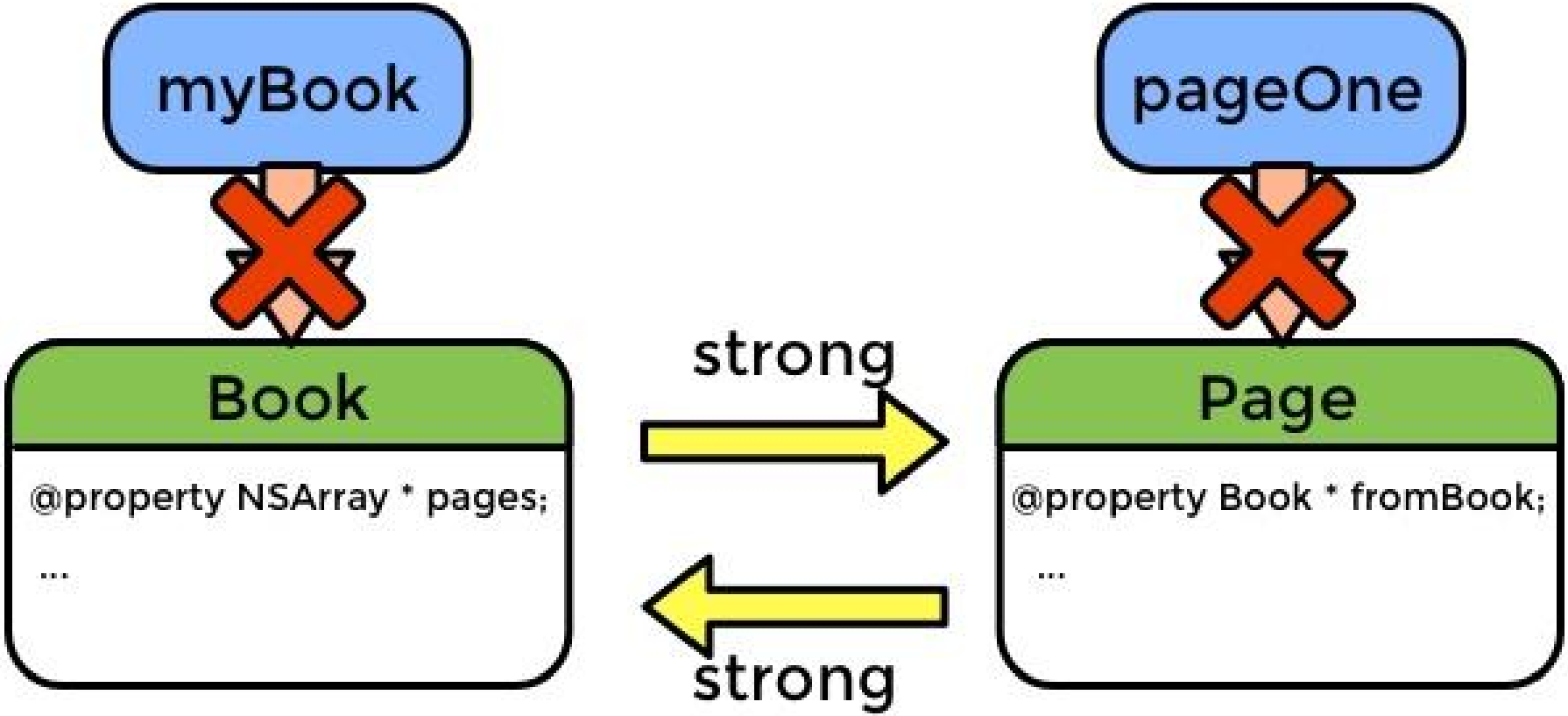
```
__block NSBlockOperation *operation = [[NSBlockOperation alloc] init];

MMVoidBlock thumbnailOperationBlock = ^{

    if (!operation.isCancelled) {
        workerBlock();
    }

    [self.thumbnailOperationList removeObjectForKey:key];
};

[operation addExecutionBlock:thumbnailOperationBlock];
```


myBook
pageOne
Book
@property NSArray * pages;
...
strong
strong
Page
@property Book * fromBook;
...

```
__block NSBlockOperation *operation = [[NSBlockOperation alloc] init];

__weak typeof(self)weakSelf = self;

MMVoidBlock thumbnailOperationBlock = ^{
    if (!operation.isCancelled) {
        workerBlock();
    }

    [weakSelf.thumbnailOperationList removeObjectForKey:key];
};


[operation addExecutionBlock:thumbnailOperationBlock];
```

```
__block NSBlockOperation *operation = [[NSBlockOperation alloc] init];

__weak typeof(self) weakSelf = self;

__weak typeof(operation) weakOp = operation;

MMVoidBlock thumbnailOperationBlock = ^{
    if (!weakOp.isCancelled) {
        workerBlock();
    }

    [weakSelf.thumbnailOperationList removeObjectForKey:key];
};


[operation addExecutionBlock:thumbnailOperationBlock];
```

RECAP

So, a closure is basically a

snapshot *of the stack,*

at the moment in which

it's created.