What about DI on iOS?

What about DI on iOS?

Who am I?

Maciej Oczko

@maciejoczko

Software Engineer at Polidea

Polidea

What about DI on iOS?

Aims

- To talk about what DI is
- To encourage you to proceed with DI pattern
- To make your coding easier with Objection or Typhoon

What is it all about?

You always deal with
dependency management somehow…

What is it all about?

You always deal with
dependency management somehow…

…but not necessarily in a good way.

Example 1

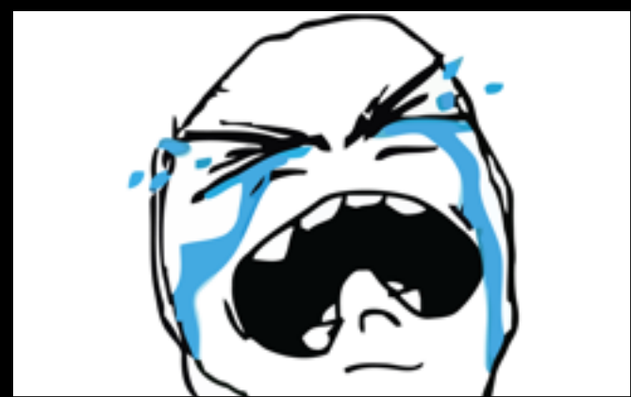
```
- (id)init {
    self = [super init];
    if (self) {
        _weatherClient = [[MYWeatherClient alloc] initWithY:y andX:x];
    }
    return self;
}
```

Example 1

```
- (id)init {
    self = [super init];
    if (self) {
        _weatherClient = [[MYWeatherClient alloc] initWithY:y andX:x];
    }
    return self;
}
```

Example 2

```
- (id)init {
    self = [super init];
    if (self) {
        _weatherClient = [MYWeatherClient sharedInstance];
    }
    return self;
}
```



Example 3

```
- (id)initWithWeatherClient:(id <MYWeatherClientProtocol>)client {
    self = [super init];
    if (self) {
        _weatherClient = client;
    }
    return self;
}
```

Example 3 DI

```
- (id)initWithWeatherClient:(id <MYWeatherClientProtocol>)client {
    self = [super init];
    if (self) {
        _weatherClient = client;
    }
    return self;
}
```

Why DI? Pros

- Classes are easier to test (or even possible)
- It promotes separation of concerns (single responsibility principle)
- Open-closed principle
- Makes app maintenance easier
- Makes introducing new features less painful

Example 1 << 2

```
describe(@"Super spec", ^{
   it(@"should always pass", ^{




   });
});
```

Example 1 << 2

```
describe(@"Super spec", ^{
   it(@"should always pass", ^{

      id <MYAPIAccessorProtocol> accessor =
         [KWMock mockForProtocol:@protocol(MYAPIAccessorProtocol)];

      id <MYCacheManagerProtocol> cache = [MYCacheManager new];

   });
});
```

Example 1 << 2

```
describe(@"Super spec", ^{
   it(@"should always pass", ^{

      id <MYAPIAccessorProtocol> accessor =
         [KWMock mockForProtocol:@protocol(MYAPIAccessorProtocol)];

      id <MYCacheManagerProtocol> cache = [MYCacheManager new];

      . . .

      MYWeatherClient *client = [[MYWeatherClient alloc]
                                    initWithX:accessor
                                         andY:cache];



   });
});
```

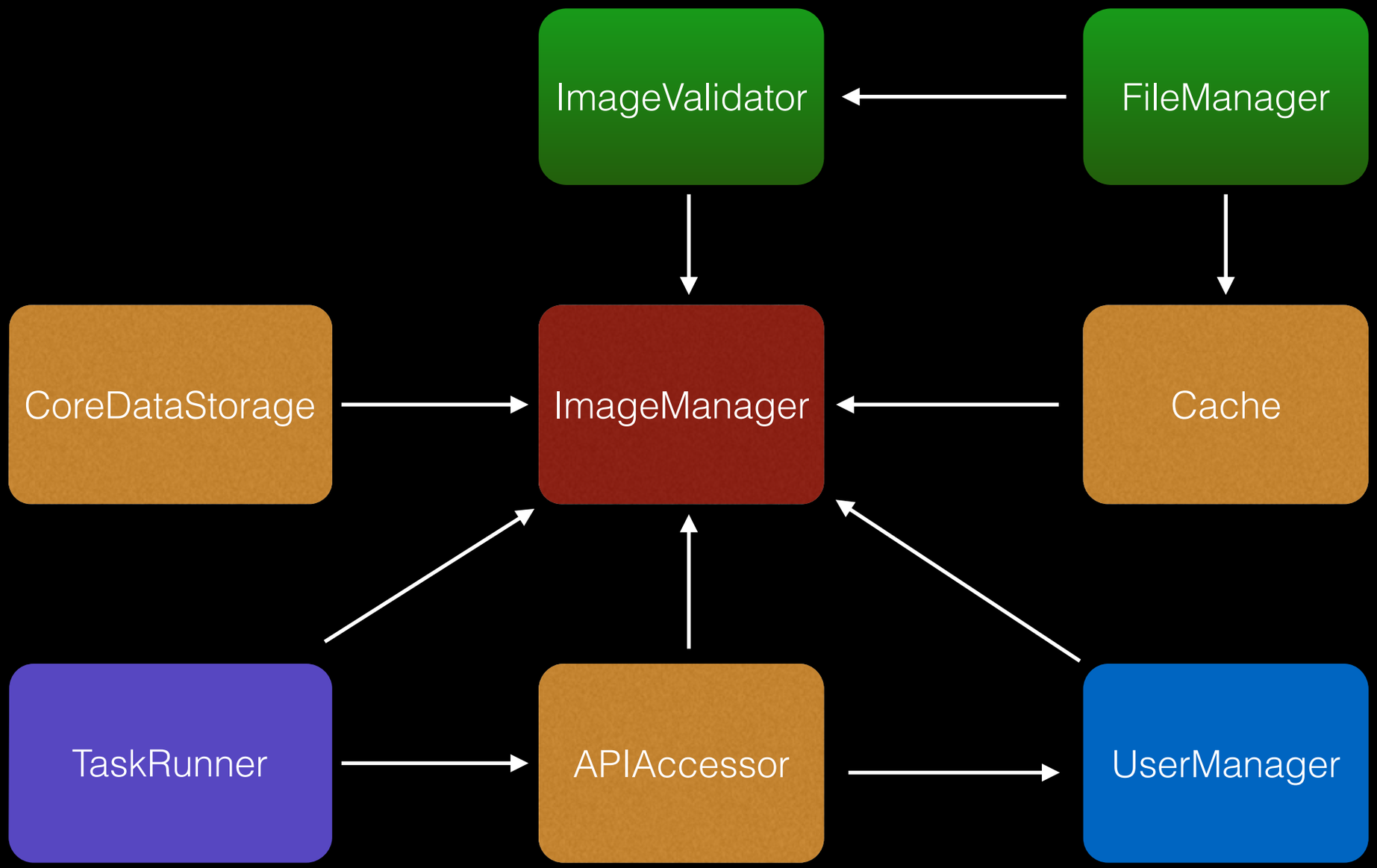
Example 1 << 2

```
describe(@"Super spec", ^{
   it(@"should always pass", ^{

      id <MYAPIAccessorProtocol> accessor =
         [KWMock mockForProtocol:@protocol(MYAPIAccessorProtocol)];

      id <MYCacheManagerProtocol> cache = [MYCacheManager new];

      . . .

      MYWeatherClient *client = [[MYWeatherClient alloc]
                                    initWithX:accessor
                                         andY:cache];


      [client getWeather];
      [[[client.weatherItems should] have:4] elements];

   });
});
```

Cons

- Life cycle management
- Objects’ creation get more complex
- Sometimes more code

Problems?


ImageValidator
FileManager
CoreDataStorage
ImageManager
Cache
TaskRunner
APIAccessor
UserManager

Problems?

```
- (id)initWithImageValidator:(id <MYImageValidatorProtocol>)imageValidator
                cacheManager:(id <MYCacheProtocol>)cacheManager
                 userManager:(id <MYUserManagerProtocol>)userManager
                 apiAccessor:(id <MYAPIProtocol>)apiAccessor
             coreDataManager:(id <MYCoreDataProtocol>)coreDataManager {
    self = [super init];
    if (self) {
        ...
        }
        return self;
    }
```

```
id <MYFileManagerProtocol> fileManager = [[MYFileManager alloc] init];
id <MYImageValidatorProtocol> imageValidator = [[MYImageValidator alloc] initWithFileManager:fileManager];

id <MYCacheProtocol> cacheManager = [[MYCacheManager alloc] initWithFileManager:fileManager];
id <MYCoreDataProtocol> coreDataManager = [[MYCoreDataManager alloc] init];

id <MYAPIProtocol> apiAccessor = [[MYAPIAccessor alloc] initWith:...];
...
MYImageManager *imageManager = [[MYImageManager alloc] initWithImageValidator:imageValidator
                                                                  cacheManager:cacheManager
                                                                   userManager:userManager
                                                                   apiAccessor:apiAccessor
                                                               coreDataManager:coreDataManager];
```

Frameworks

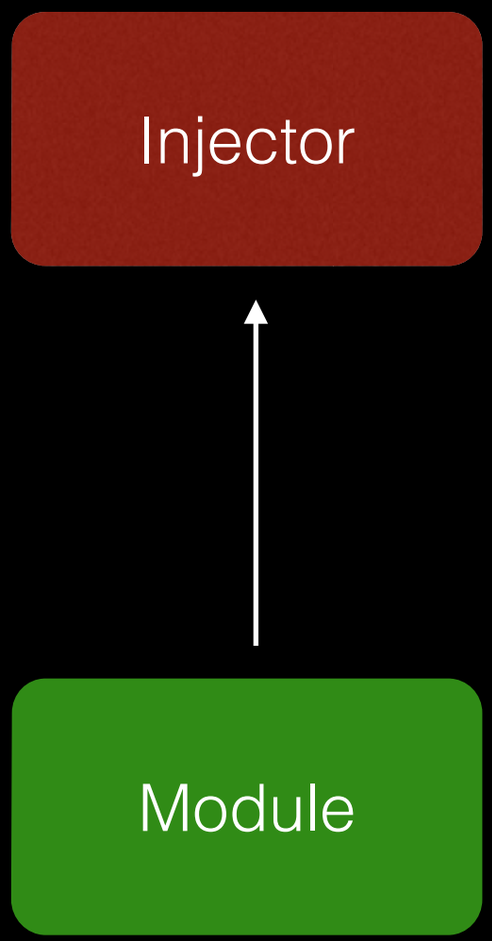
Objection

Objection

A lightweight dependency injection framework for Objective-C

- ‘Annotation’ based DI
- Initializer support
- Proporties auto-wiring
- Lazily instantiated dependencies
- Custom providers
- Class bindings
- Protocol bindings
- Instance bindings
- Life cycle management
- Cyclic dependencies

How it works, quickly




```
id obj = [injector getObject:@protocol(FancyProtocol)];
```

```
[self bindProtocol:@protocol(FancyProtocol)
           toClass:[MYFancy class]];
```

Initializer

```
@implementation MYObject

objection_initializer_sel(@selector(initWithClient:))

- (id)initWithClient:(id <MYClientProtocol>)client {
    self = [super init];
    if (self) {
        _client = client;
    }
    return self;
}

@end
```

Initializer

```
@implementation

objection_initializer_sel

- (
      _client

)

@end
```

```
id obj = [injector getObject:[MYObject class] argumentList:@[ client ]];
```

Properties

```
@interface MYObject
@property(nonatomic, readonly) id <MYClientProtocol> client;
@property(nonatomic, readonly) id <MYAPIProtocol> apiAccessor;
@end

@implementation MYObject

objection_requires_sel(@selector(client), @selector(apiAccessor))

- (id)init {
    self = [super init];
    if (self) {

    return self;
}

- (void)awakeFromObjection { }

@end
```

Properties

```
@interface
@property
@property
@end

@implementation

objection_requires_sel
- (id


}
- (void
@end
```

```
        id obj = [injector getObject:[MYObject class]];
```

Pros & Cons

- Easy to use
- Fairly simple
- Configuration in place

- Invasive
- **Properties driven**
- A few defects

Typhoon

- All Objection features
- No macros or XML required but both supported
- Any order of dependencies
- Supports configuration management
- Supports injection of view controllers and storyboard integration

How it works, quickly

Component Factory

```
MYObject *object = [componentFactory
                      componentForType:[MYObject class];
```


Assembly

```
- (id)myObject {
    return [TyphoonDefinition withClass:[MYObject Class]];
}
```

Assembly

```
@interface MYAssemby : TyphoonAssembly

@implementation MYAssembly

- (id)imageManager {
    return [TyphoonDefinition withClass:[MYImageManager class]
```

Assembly

```
@interface MYAssemby : TyphoonAssembly

@implementation MYAssembly

- (id)imageManager {
    return [TyphoonDefinition withClass:[MYImageManager class]
initialization:^(TyphoonInitializer* initializer) {

        initializer.selector = @selector(initWithClient:);
        [initializer injectWithDefinition:[self defaultClient]];

    }
```

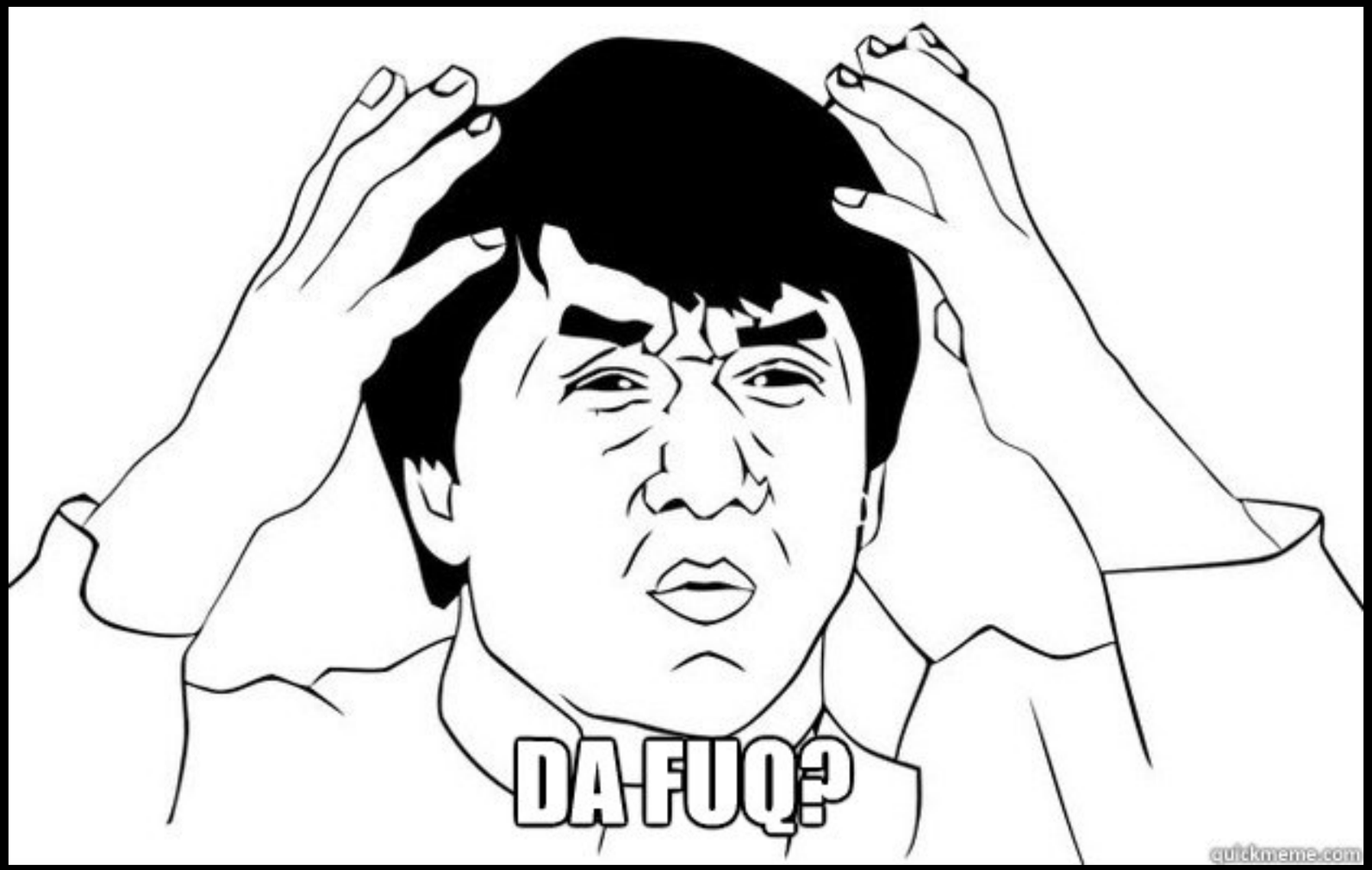
Assembly

```
@interface MYAssemby : TyphoonAssembly

@implementation MYAssembly

- (id)imageManager {
    return [TyphoonDefinition withClass:[MYImageManager class]
initialization:^(TyphoonInitializer* initializer) {
        initializer.selector = @selector(initWithClient:);
        [initializer injectWithDefinition:[self defaultClient]];

    }
properties:^(TyphoonDefinition* definition) {

        [definition injectProperty:@selector(api) withDefinition:[self api]];
        [definition injectProperty:@selector(task) withValueAsText:@"${tasks.first}"];

        [definition setAfterPropertyInjection:@selector(configureBeforeUse)];
        [definition setScope:TyphoonScopeSingleton];
    }];
}
```

Pros & Cons

- One configuration place
- Configuration in place
- Not invasive
- Lots of features

- More complicated code base
- Clarity


DA FUQ?
quickmeme.com


WeKnowMemes

Summary

- DI makes testing a lot easier
- DI improves your code effectiveness
- DI makes your classes cleaner and more reusable
- DI frameworks usage makes your life easier

Thanks!

maciek.oczko@polidea.com
@maciejoczko