INTRODUÇÃO A REACTIVE FUNCTIONAL PROGRAMMING.

by Guilherme Martinez Sampaio

Guilherme Martinez Sampaio

iOS Developer

@gsampaio


movile
CONNECT>EVERYONE


Let's
Play
Lupi

Agenda

- Functional Reactive Programming
- Reactive Cocoa
- Model View View Model
- Demo
- Referências

FUNCTIONAL REACTIVE PROGRAMMING?


State! State! State! State!

FUNCTIONAL REACTIVE PROGRAMMING
IS A PROGRAMMING PARADIGM FOR
reactive programming
USING THE BUILDING BLOCKS OF
functional programming.

FRP HAS BEEN USED FOR PROGRAMMING

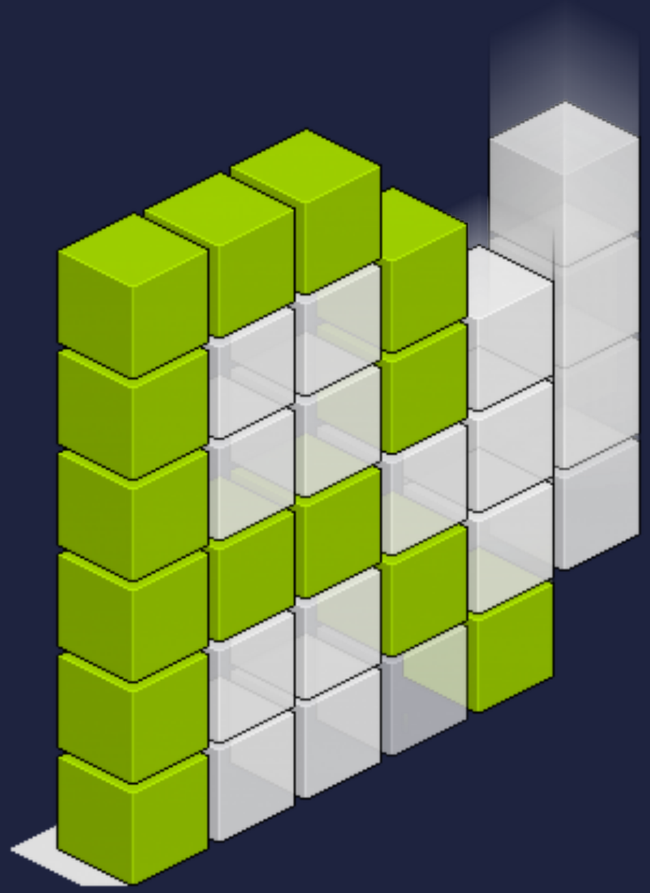
GUI, ROBOTICS, AND MUSIC,

AIMING TO SIMPLIFY THESE PROBLEMS BY EXPLICITLY

modeling time.

EM OUTRAS PALAVRAS
DESCREVEMOS NOSSOS PROGRAMAS
COMO EVENTOS QUE REAGEM COM
o tempo

Reactive Cocoa



RAC É FORTEMENTE BASEADA NO RXEXTENSIONS DA MICROSOFT PARA .NET

RAC SE BASEIA EM

Signals E Sequences

AO INVÉS DE USAR variáveis
QUE SÃO ALTERADAS
USAMOS sinais
QUE CAPTURA O
valor atual E
valores futuros.

SINAIS FUNCIONAM COMO

Promises

SINAIS TEM *subscribers* QUE ESCUTAM SUAS ALTERAÇÕES

SUBSCRIBERS RECEBEM

3 eventos

NEXT

COMPLETED


ERROR

RACSIGNAL

```objc
__block NSUInteger subscribers = 0;
RACSignal *signal = [RACSignal createSignal:^ RACDisposable * (id<RACSubscriber> subscriber) {
    subscribers++;
    [subscriber sendNext:@(subscribers)];
    [subscriber sendCompleted];
    return nil;
}];

[signal subscribeNext:^(id x) {
        NSLog(@"%@", x)
} error:^(NSError *error) {
        NSLog(@"ERRO 🔫");
} completed:^{
        NSLog(@"Signal completo");
}]

/**
Output:
1
Signal completo
**/
```

SINAIS SÃO EXECUTADOS A
A PARTIR DO MOMENTO QUE
ELE TEM UM *subscriber*
ATÉ ELE ENVIAR UM SINAL DE
completed

SINAIS PODEM TER VÁRIOS

subscribers

PODEMOS COMPOR SINAIS USANDO

programação funcional

RACSIGNAL

```
__block NSUInteger subscribers = 2;
RACSignal *signal = [RACSignal createSignal:^ RACDisposable * (id<RACSubscriber> subscriber) {
    subscribers++;
    [subscriber sendNext:@(subscribers)];
    [subscriber sendCompleted];
    return nil;
}];

RACSignal *powSingal = [signal map:^NSNumber(NSNumber *value){
    NSUInteger intValue = [value unsignedIntegerValue];
    return @(intValue * intValue);
}];

[signal subscribeNext:^(id x) {
    NSLog(@"VALUE: %@", x);
}];

/**
Outputs:
9
**/
```

COLLECTIONS PODEM GERAR *RACSequence*

QUE POR SUA VEZ PODEMOS USAR APLICAR

Programação Funcional

RACSEQUENCE

```
NSArray *lettersArray = [@"A B C D E F G H I" componentsSeparatedByString:@" "];
RACSignal *letters = lettersArray.rac_sequence.signal;

[letters subscribeNext:^(NSString *x) {
    NSLog(@"%@", x);
}];

/**
Outputs:
A B C D E F G H I
**/
```

RACSEQUENCE

```
RACSignal *numbers = [@[@(1), @(2), @(3)].rac_sequence.signal;
RACSignal *pow = [numbers map:^NSNumber*(NSNumber *value){
    NSUInteger intValue = [value unsignedIntegerValue];
    return @(intValue * intValue);
}];

[pow subscribeNext:^(NSNumber *value) {
    NSLog(@"%@", value);
}];

/**
Outputs:
1 4 9
**/
```

RAC CONTEM MACROS PARA FAZER *data-binding*

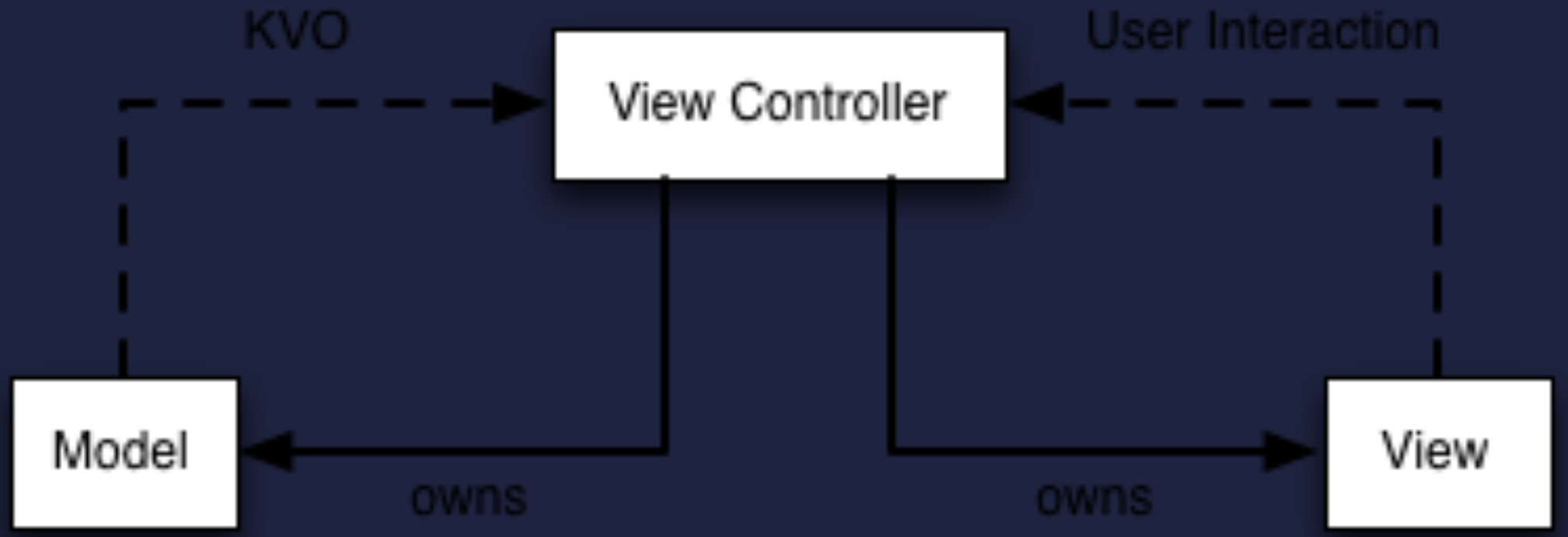
RAC()

```
@interface ViewController ()
@property(nonatomic) NSUInteger *foo;
@property(nonatomic) NSUInteger *bar;
@end

@implementation ViewController
- (void)viewDidLoad {
    self.foo = 0;
    self.bar = 0;
    NSLog(@"FOO: %@", @(self.foo));

    RAC(self, foo) = RACObserve(self, bar);
    self.bar = 1;
    NSLog(@"FOO: %@", @(self.foo));
}
@end
/**
Outputs:
FOO: 0
FOO: 1
**/
```


MVVM

MODEL VIEW VIEW-MODEL É UM ARCHITECTURAL PATTERN SEMELHANTE AO MVC

MODEL VIEW CONTROLLER

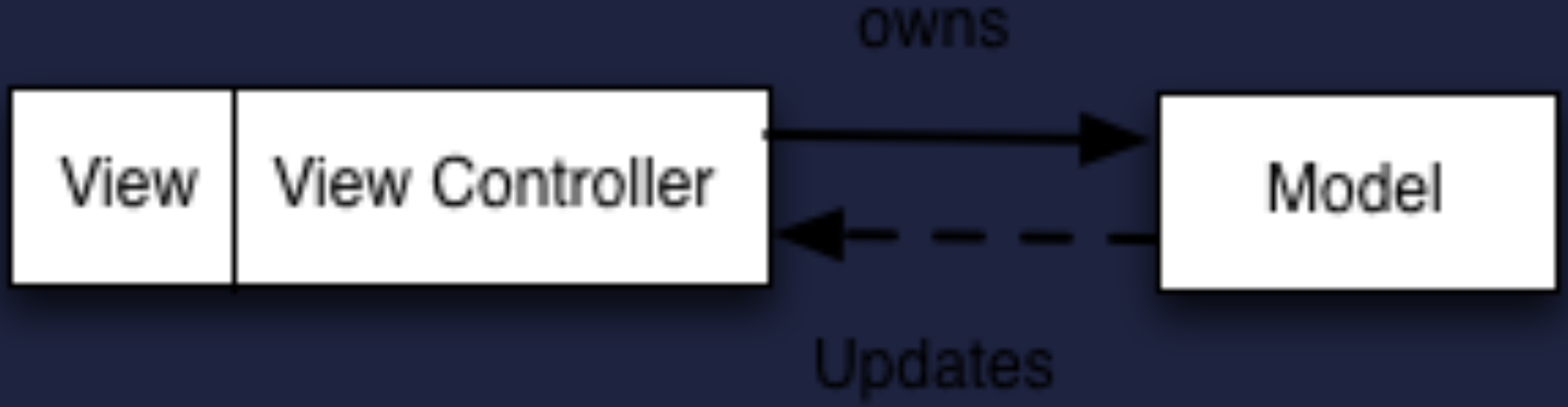


MAS NA REALIDADE

O QUE ACONTECE É QUE FICAMOS COM

view controllers gigantes

MODEL VIEW CONTROLLER



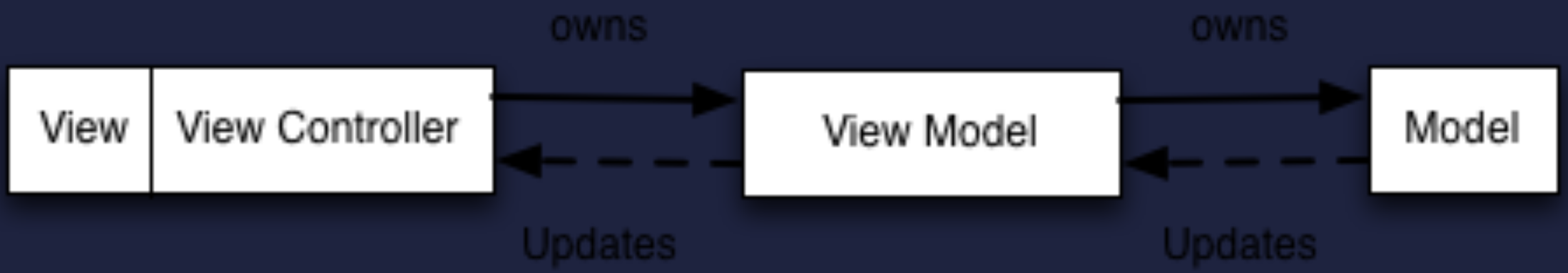
COM MVVM MOVEMOS A LÓGICA DE NEGOCIO TOTALMENTE FORA DO VC

E TRATAMOS O VC COMO PARTE DA VIEW

UMA VEZ QUE ELE TRATA

APRESENTAÇÃO, ROTAÇÃO E NAVEGAÇÃO.

MODEL VIEW VIEW-MODEL



PARA CADA VIEW CRIAMOS UM VIEW MODEL QUE TRATA ELA.

USAMOS DATA BINDING PARA FAZER COMUNICAÇÃO.

MODEL VIEW VIEW MODEL

```
@interface ViewController ()
@property(nonatomic, strong) ViewModel *viewModel;
@property(nonatomic, weak) UITextField *textField;
@property(nonatomic, weak) UILabel *label;
@end

@implementation ViewController
- (void)viewDidLoad {
    RAC(self.viewModel, text) = [self.textField rac_textSignal];
    RAC(self.label, text) = [self.viewModel expensiveComputation];
}
@end
```

VIEW MODELS *facilitam escrever testes de uma view.*

UMA VEZ QUE TESTAMOS O COMPORTAMENTO DAS VIEWS AO INVÉS DA VIEW EM SI

VIEW MODELS PREGAM

Composição sobre Herança.

FAZENDO COM QUE TENHAMOS UM CÓDIGO

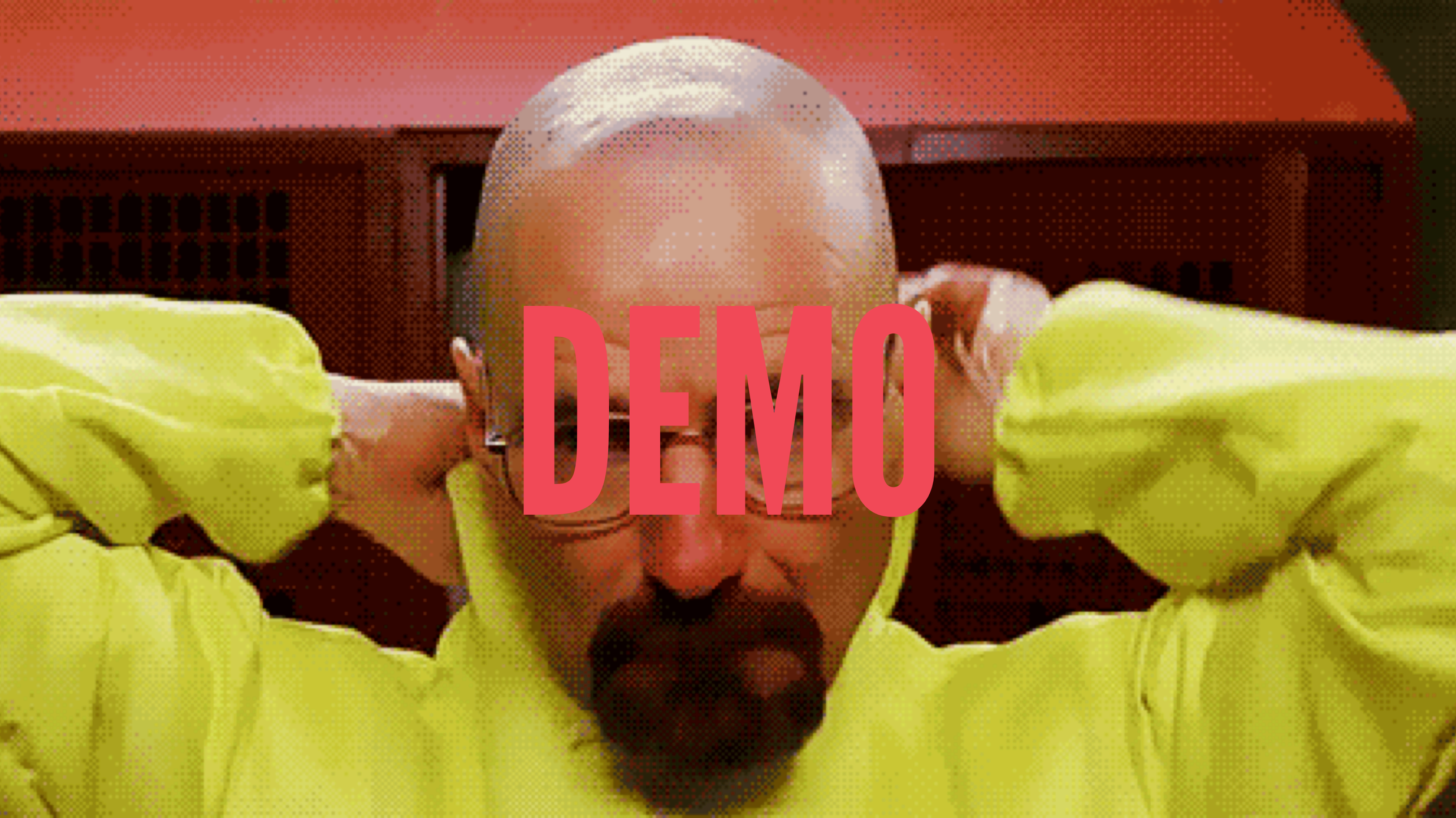
mais reaproveitável.

UMA VEZ QUE PODEMOS PLUGAR OS VIEW MODELS QUE QUEREMOS NOS NOSSOS VC

COM ALGUMA EXCESSÕES

VIEW MODELS SÃO

independentes de plataforma.


DEMO

REFERÊNCIAS

- **FRP** - http://en.wikipedia.org/wiki/Functional_reactive_programming
- **Reactive Cocoa** - https://github.com/ReactiveCocoa/ReactiveCocoa
- **Model View View Model** - http://www.objc.io/issue-13/mvvm.html

TWITTER

- @jspahrsummers
- @joshaber
- @rob_rix
- @indragie
- @ashfurrow

SLIDES E DEMO

https://github.com/gsampaio/RACMWMPresentation


THANKS