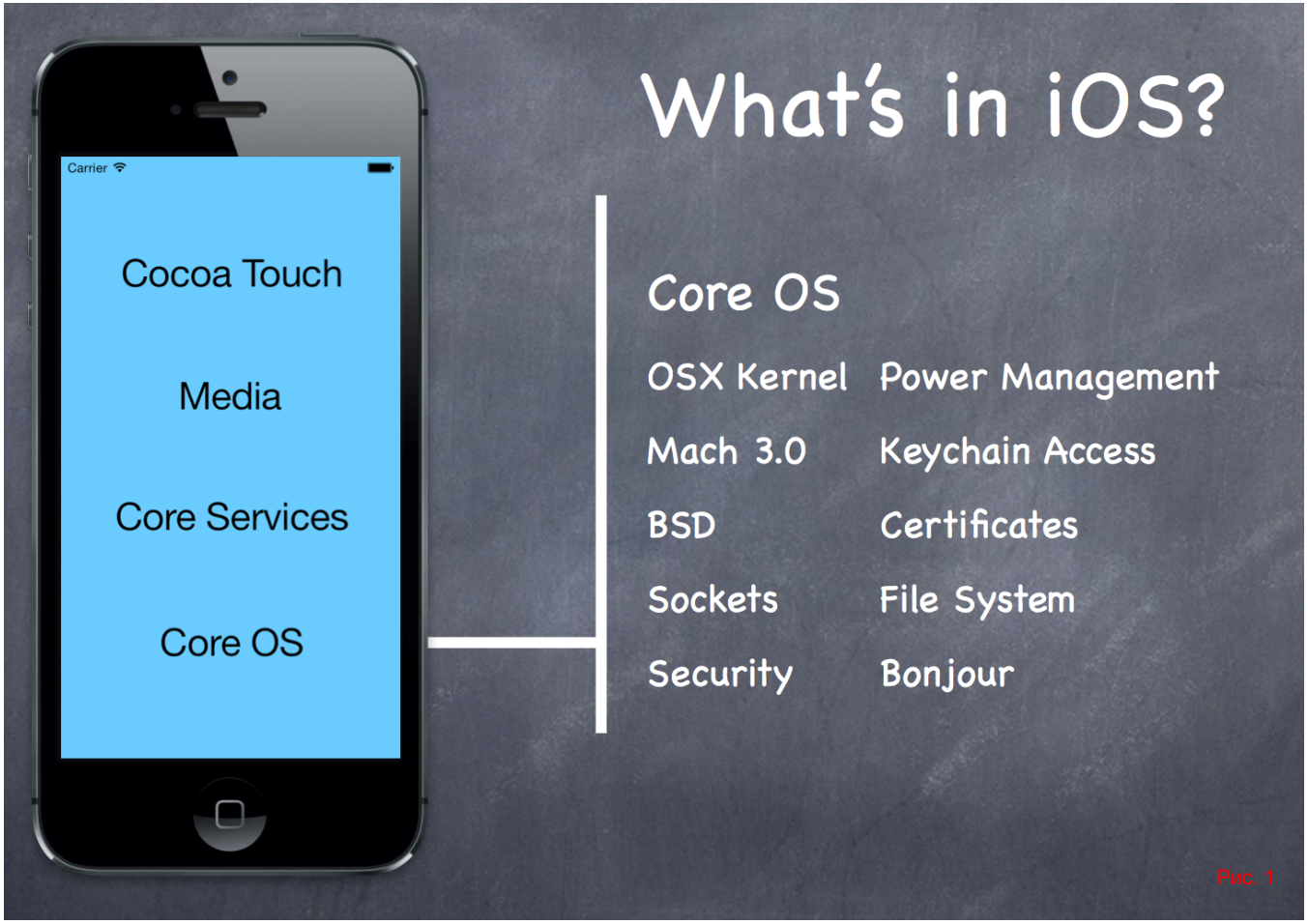

Carrier
Cocoa Touch
Media
Core Services
Core OS
What's in iOS?
Core OS
OSX Kernel
Mach 3.0
BSD
Sockets
Security
Power Management
Keychain Access
Certificates
File System
Bonjour

Рис. 1

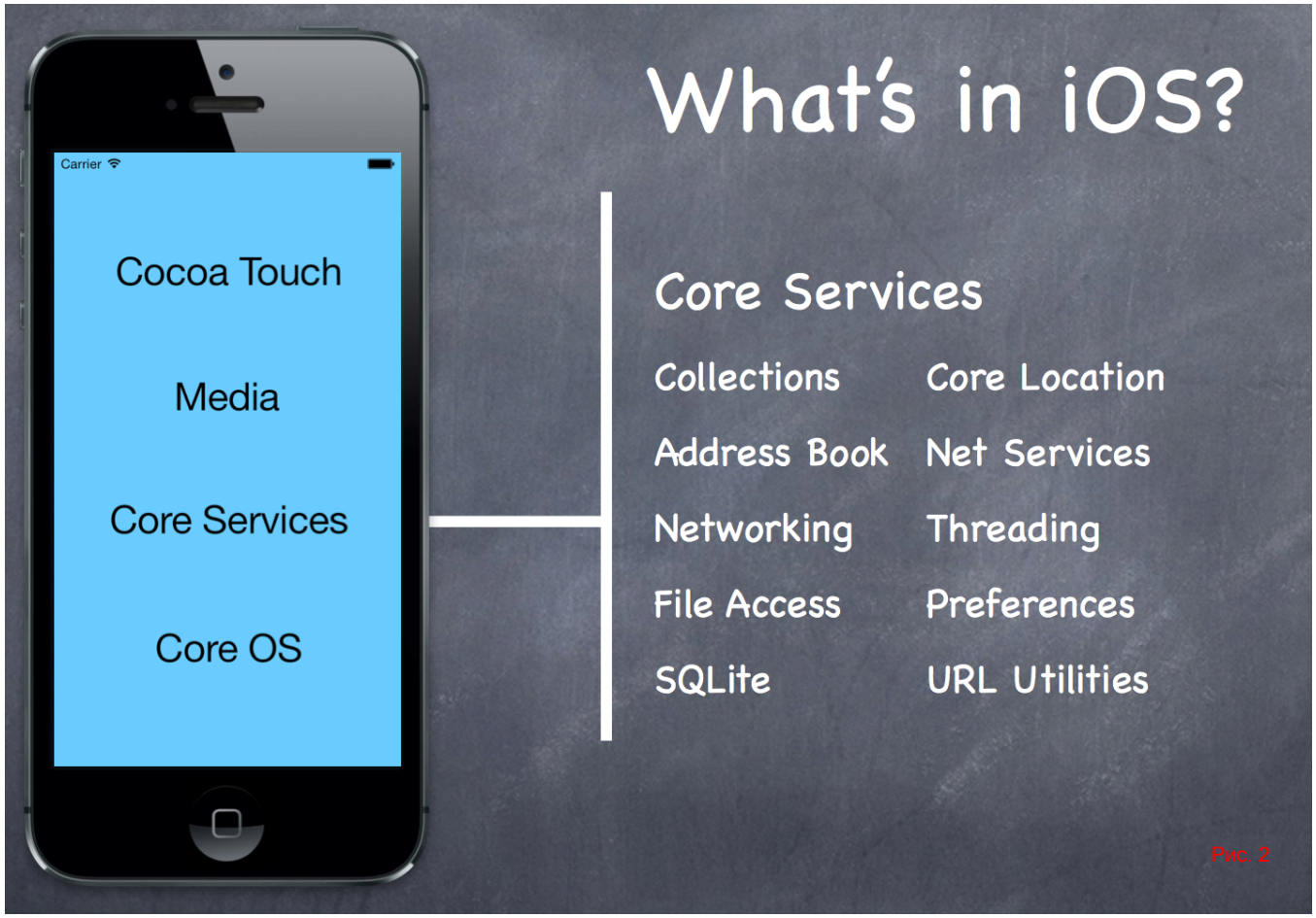
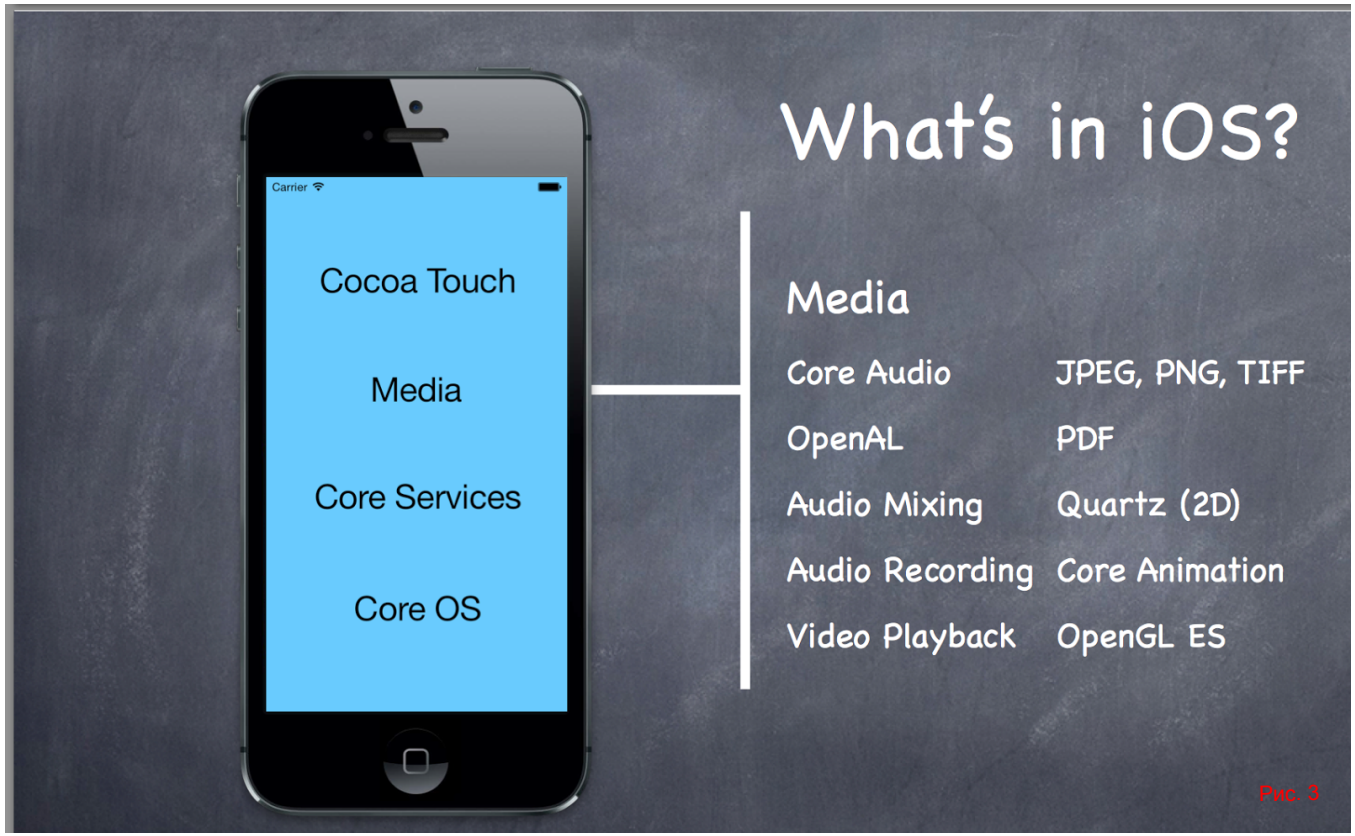

Carrier
Cocoa Touch
Media
Core Services
Core OS
What's in iOS?
Core Services
Collections
Core Location
Address Book
Net Services
Networking
Threading
File Access
Preferences
SQLite
URL Utilities

Рис. 2


What's in iOS?
Carrier
Cocoa Touch
Media
Core Services
Core OS
Media
Core Audio
OpenAL
Audio Mixing
Audio Recording
Video Playback
JPEG, PNG, TIFF
PDF
Quartz (2D)
Core Animation
OpenGL ES
Рис. 3


Carrier
Cocoa Touch
Media
Core Services
Core OS
What's in iOS?
Cocoa Touch
Multi-Touch
Core Motion
View Hierarchy
Localization
Controls
Alerts
Web View
Map Kit
Image Picker
Camera

Рис. 4

Platform Components

- Tools

Xcode 5

Instruments

- Language

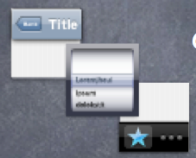
```
[display setTextColor:[UIColor blackColor]];
```

- Frameworks

Foundation

Core Data

Map Kit



UIKit

Core Motion

- Design Strategies

MVC

Рис. 5

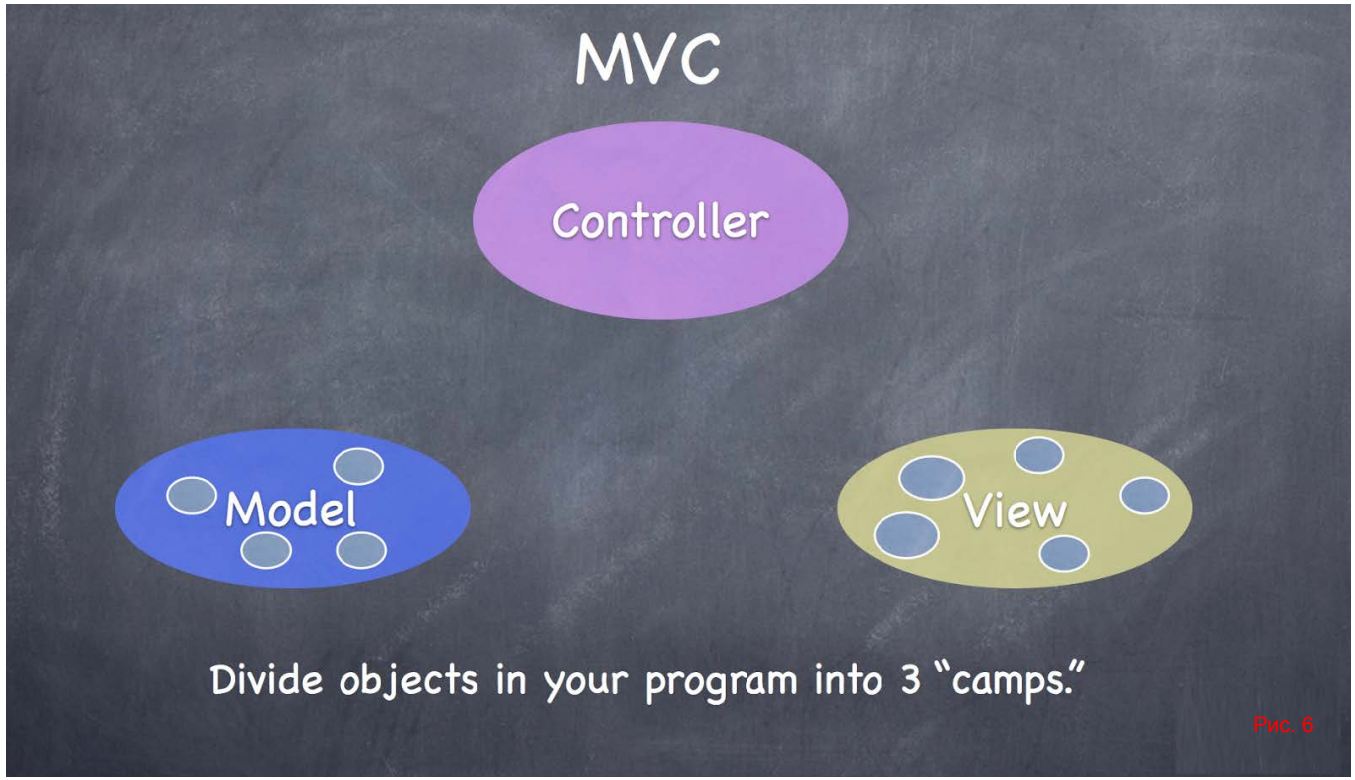
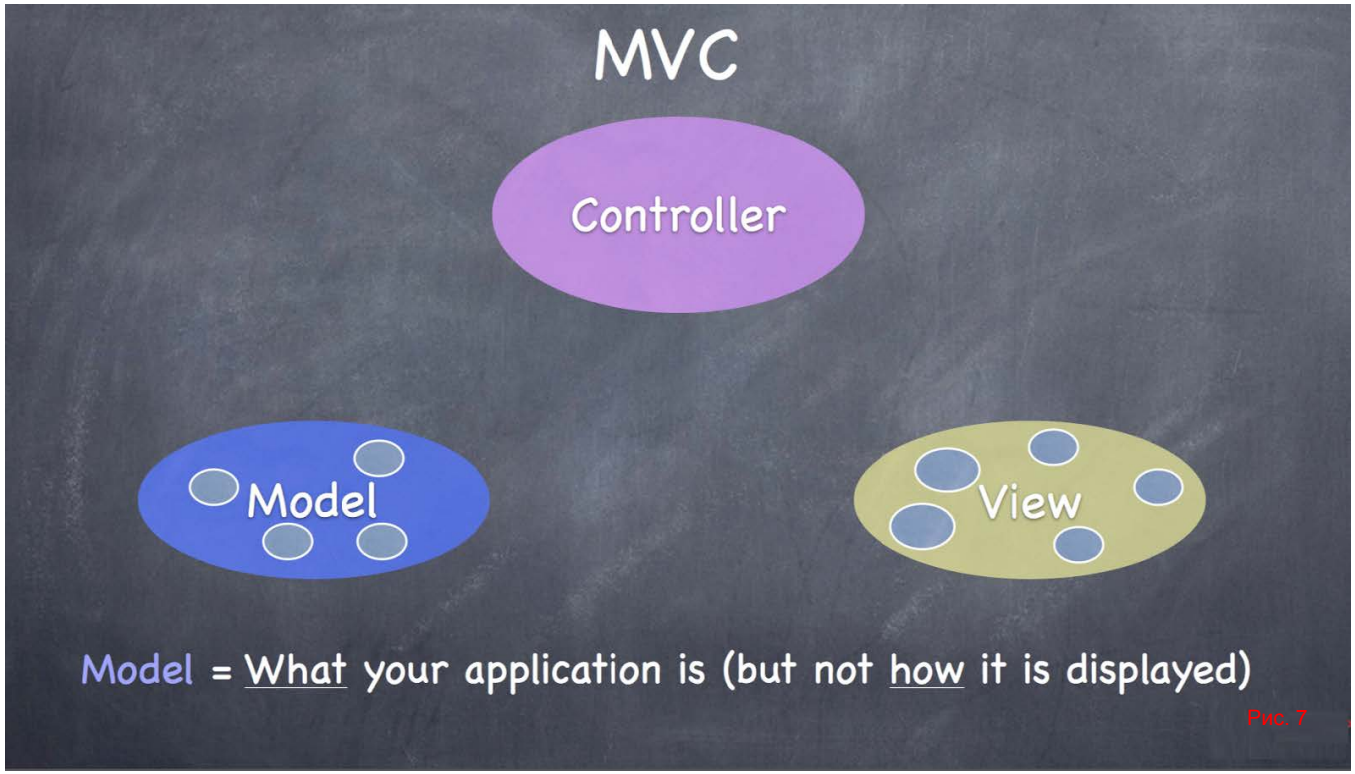

MVC
Controller
Model
View
Divide objects in your program into 3 "camps."

Рис. 6


MVC
Controller
Model
View
Model = What your application is (but not how it is displayed)
Рис. 7

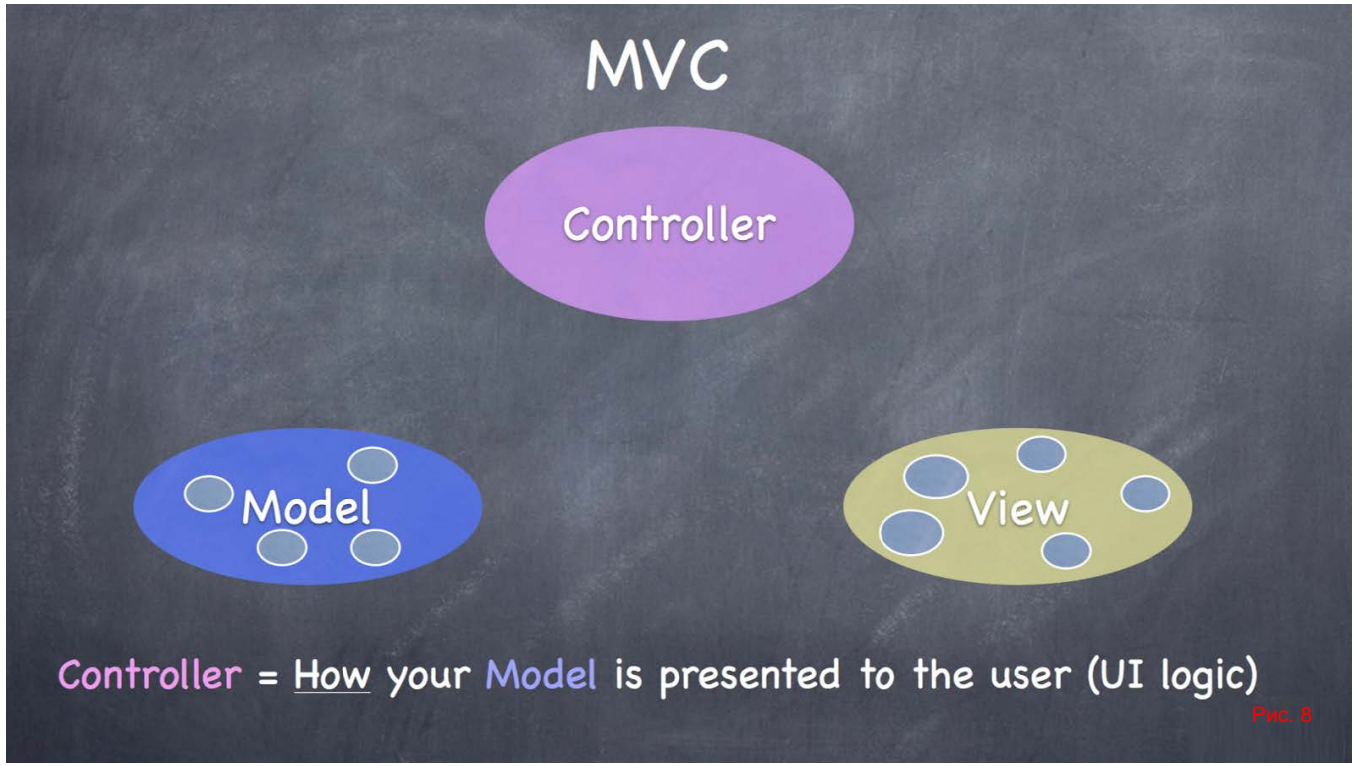

MVC
Controller
Model
View
Controller = How your Model is presented to the user (UI logic)

Рис. 8

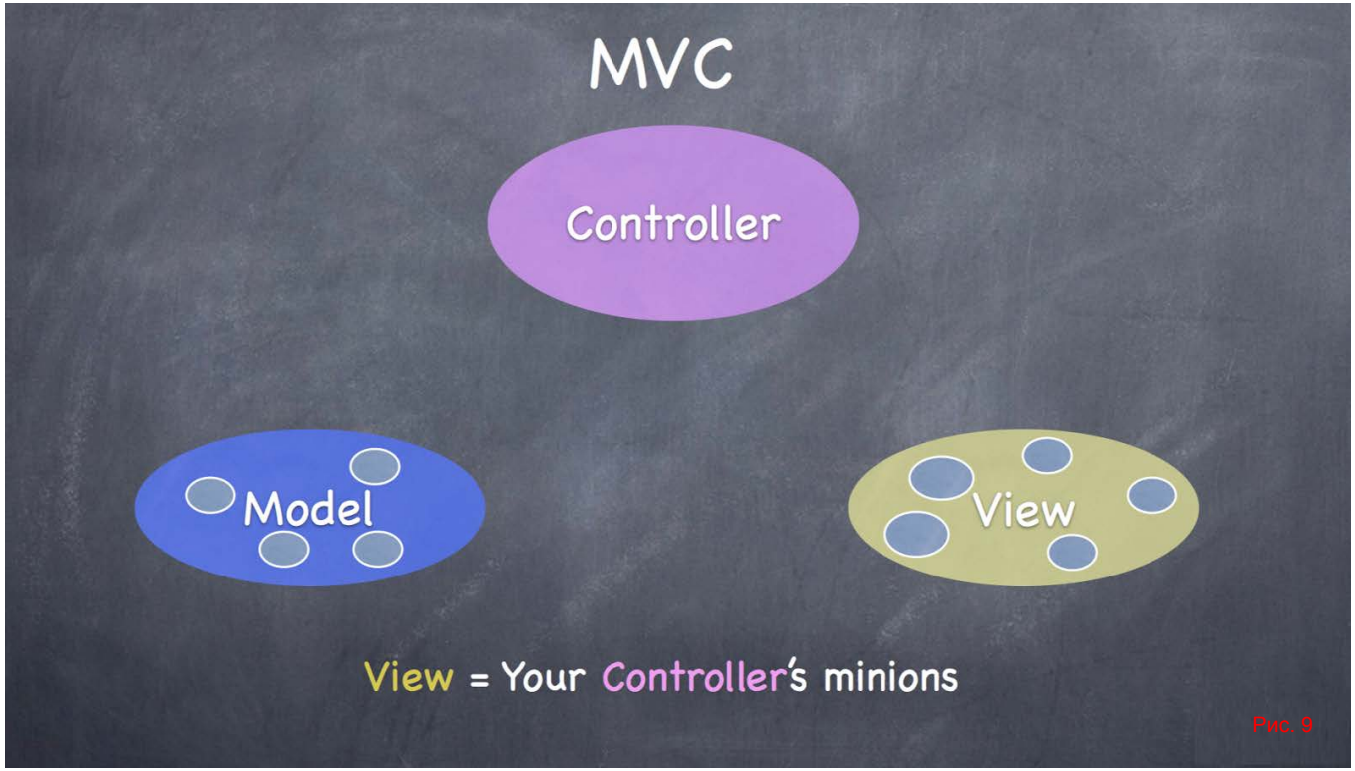


Рис. 9

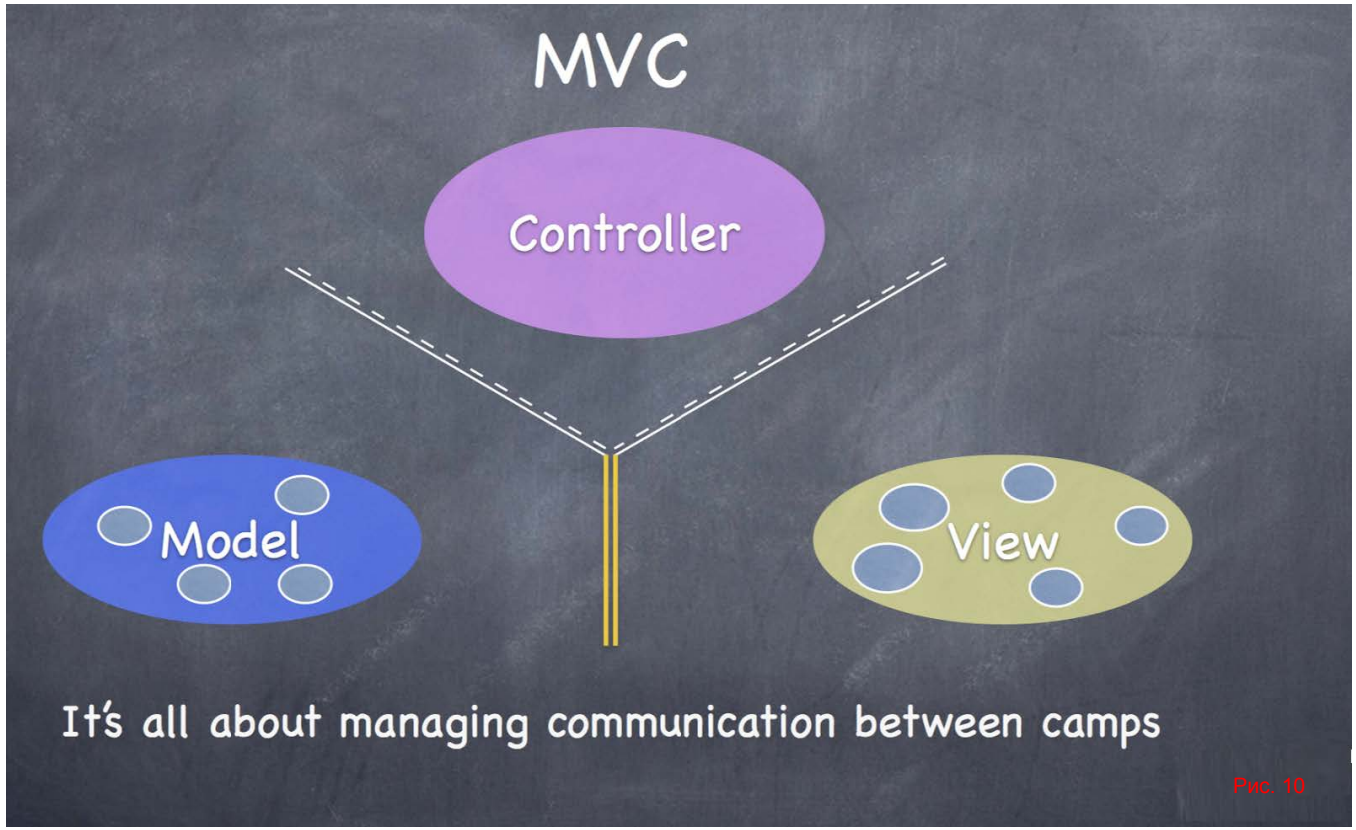

MVC
Controller
Model
View
It's all about managing communication between camps

Рис. 10

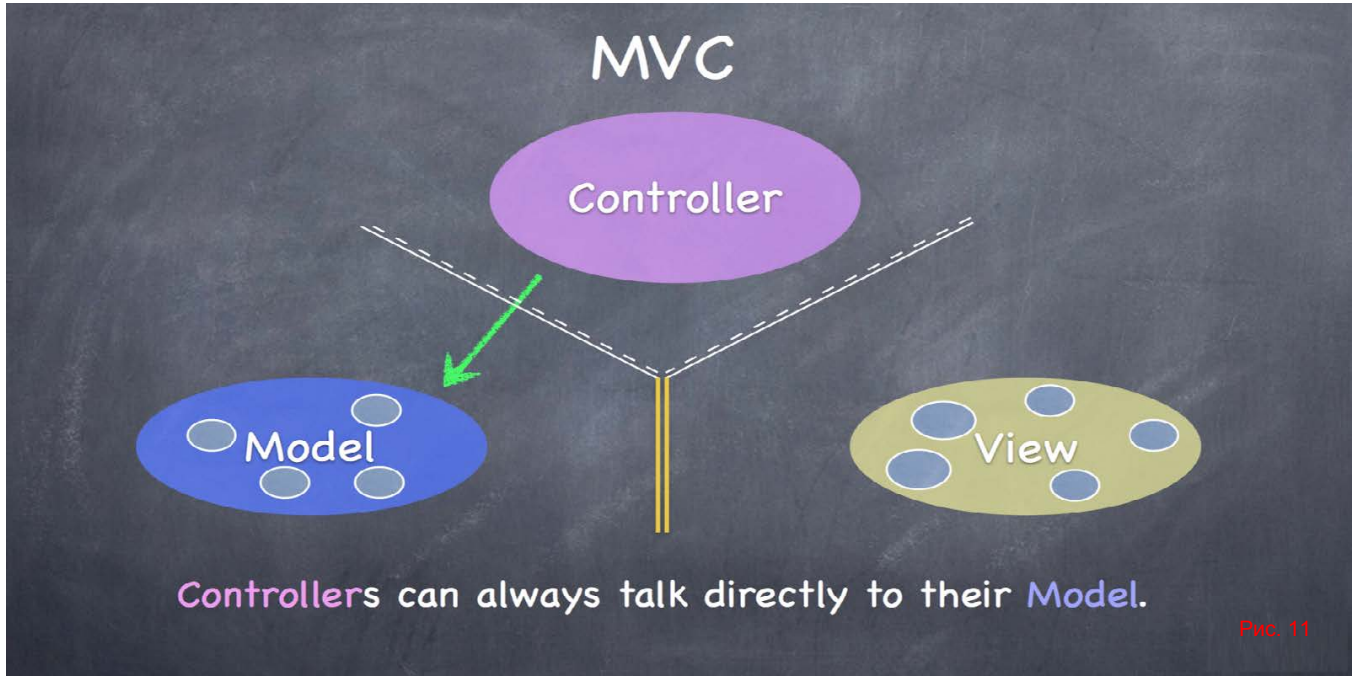

MVC
Controller
Model
View
Controllers can always talk directly to their Model.

Рис. 11

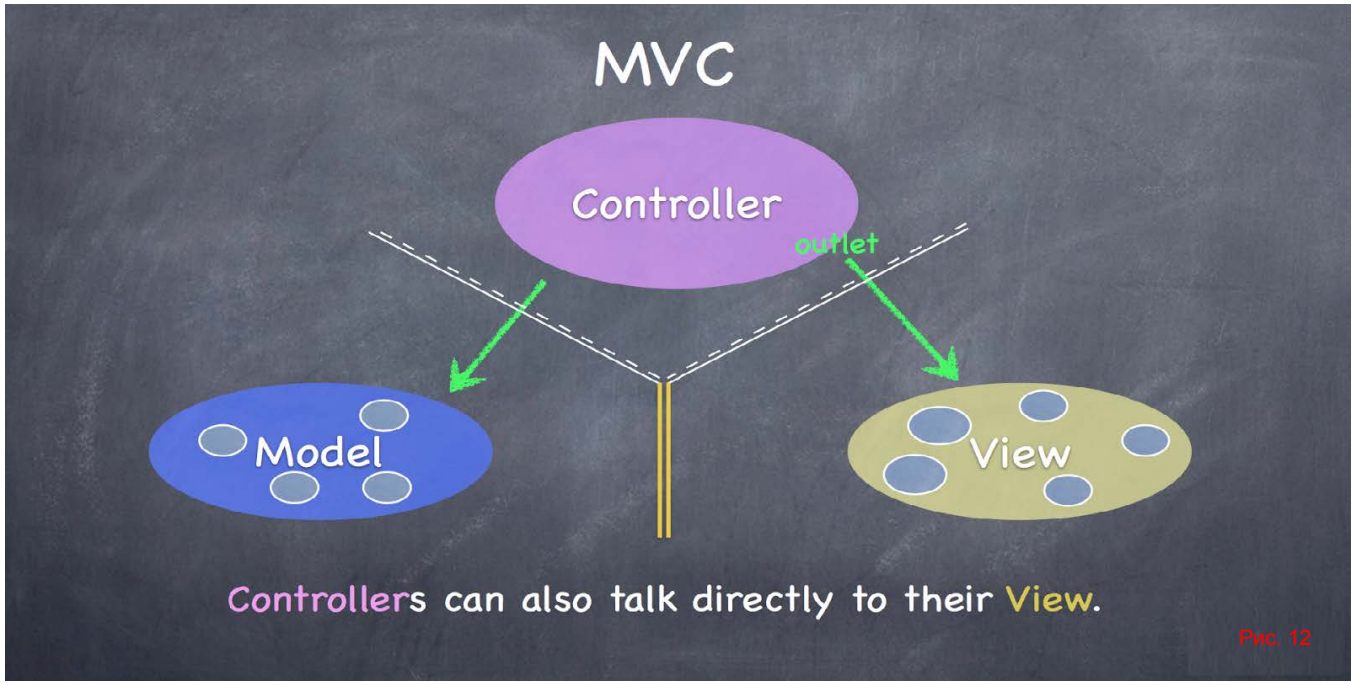

MVC
Controller
outlet
Model
View
Controllers can also talk directly to their View.

Рис. 12

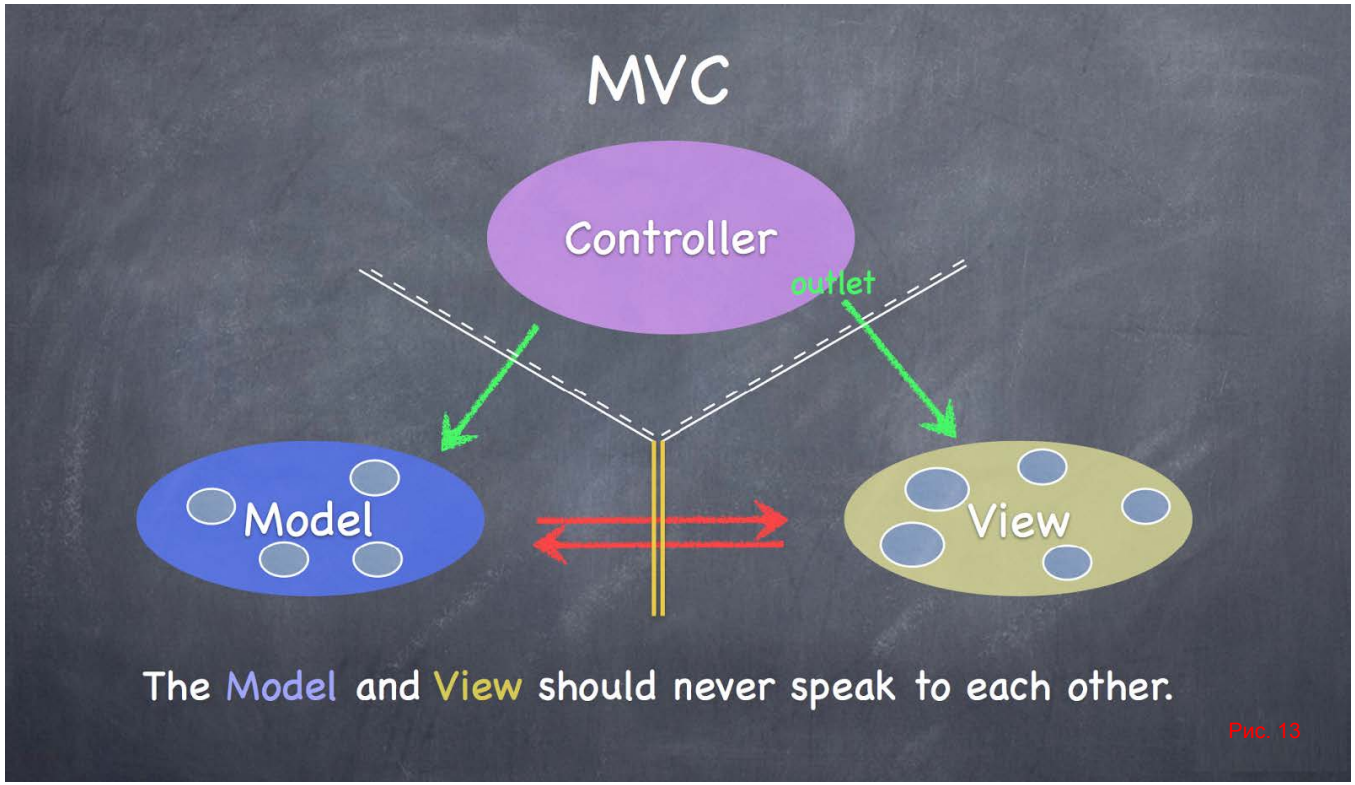

MVC
Controller
outlet
Model
View
The Model and View should never speak to each other.

Рис. 13

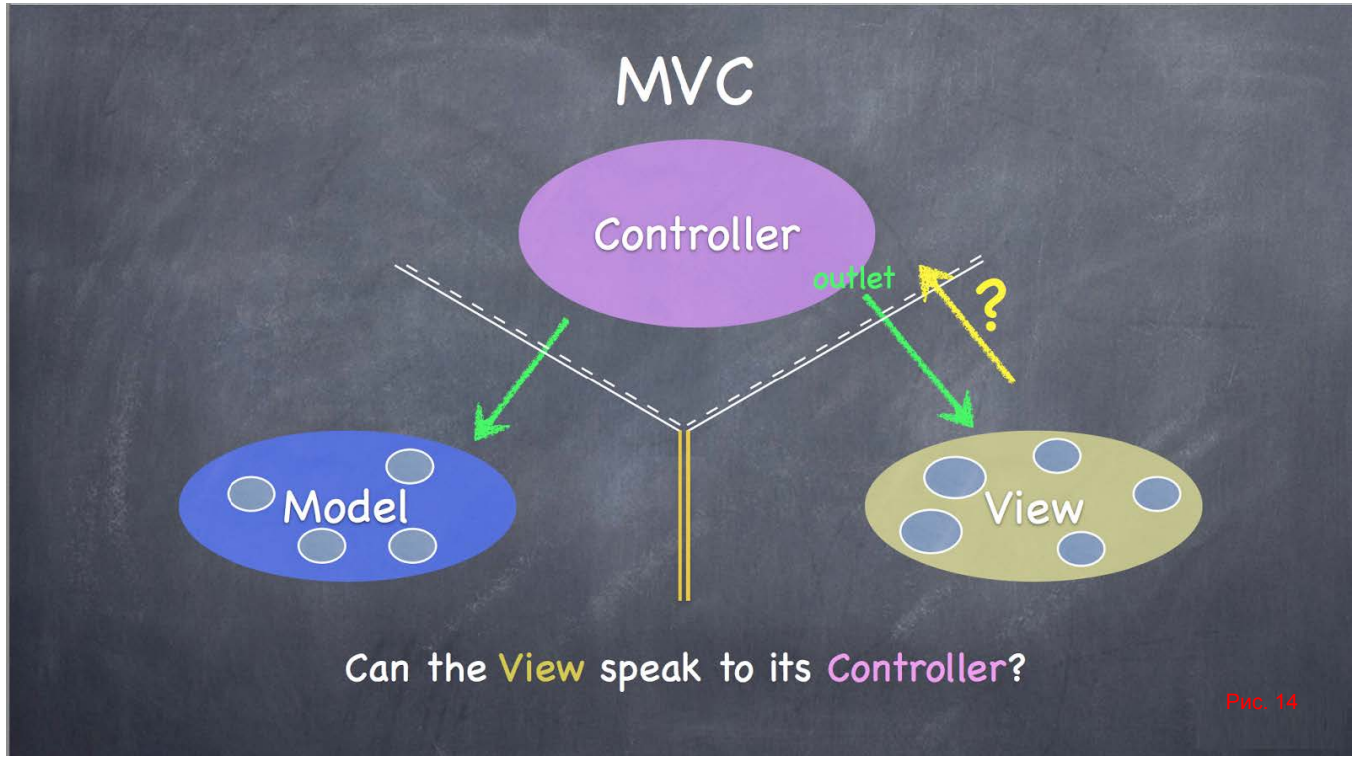

MVC
Controller
outlet
?
Model
View
Can the View speak to its Controller?

Рис. 14

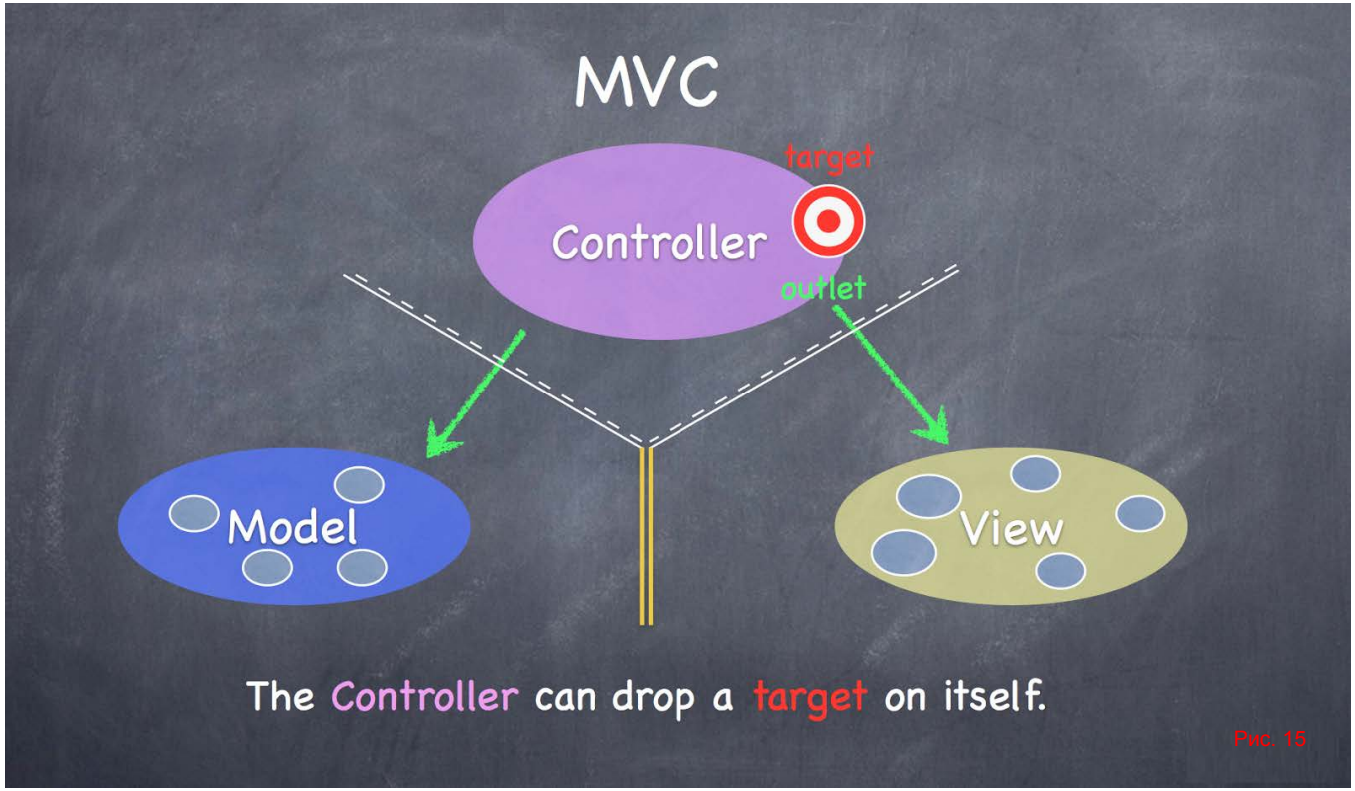

MVC
target
Controller
outlet
Model
View
The Controller can drop a target on itself.

Рис. 15

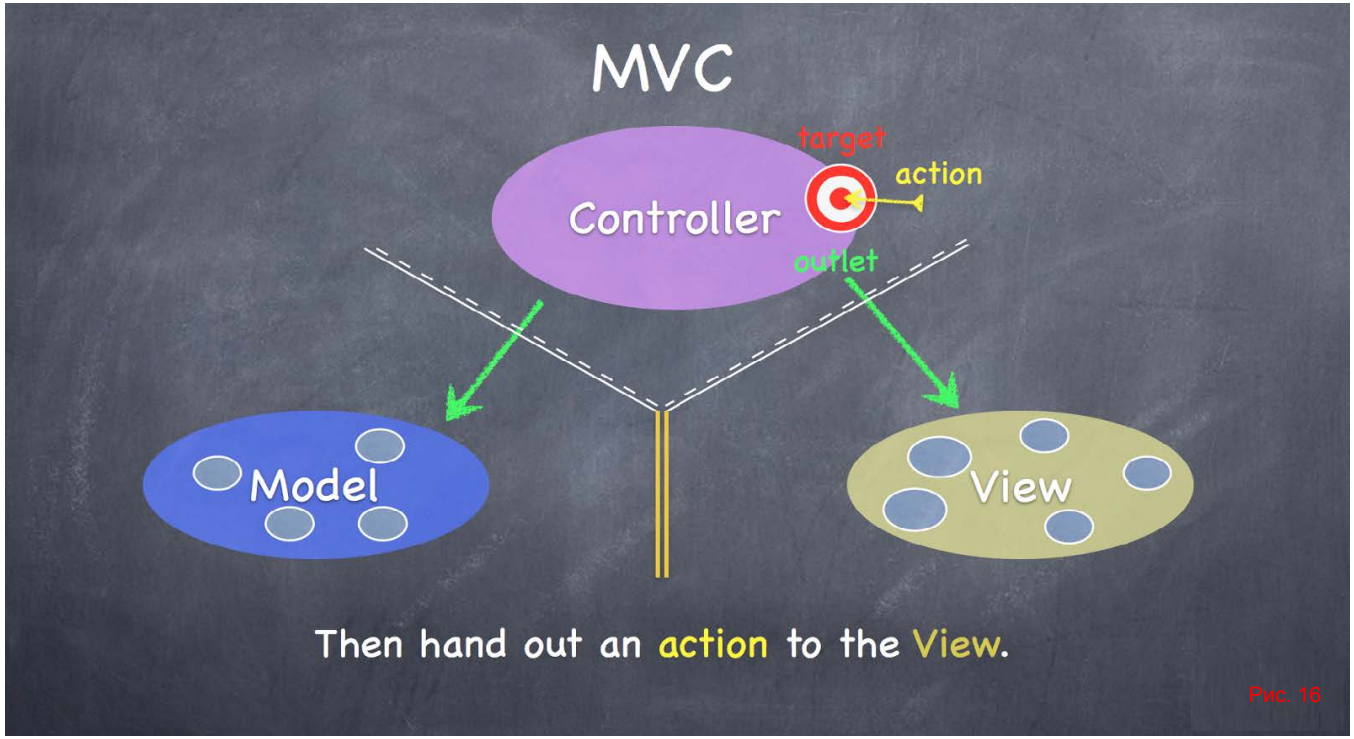

MVC
target
action
Controller
outlet
Model
View
Then hand out an action to the View.

Рис. 16

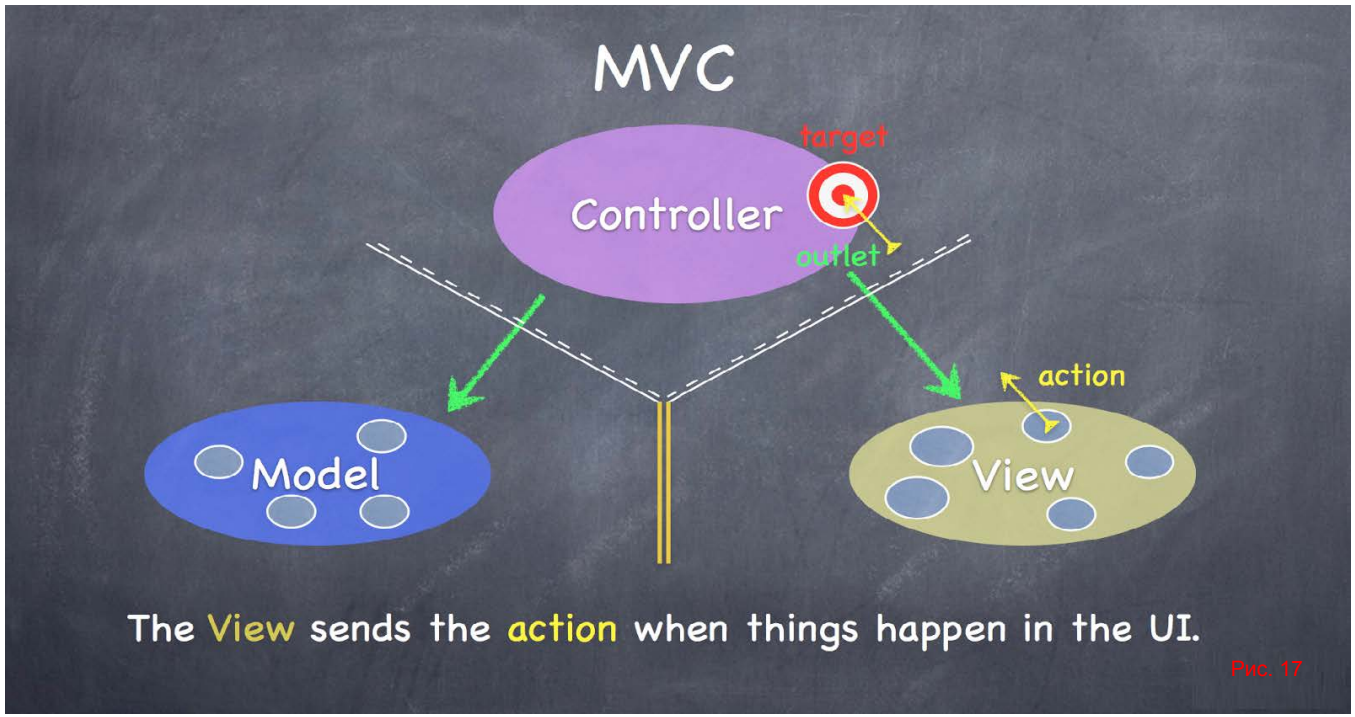

MVC
target
Controller
outlet
action
Model
View
The View sends the action when things happen in the UI.

Рис. 17

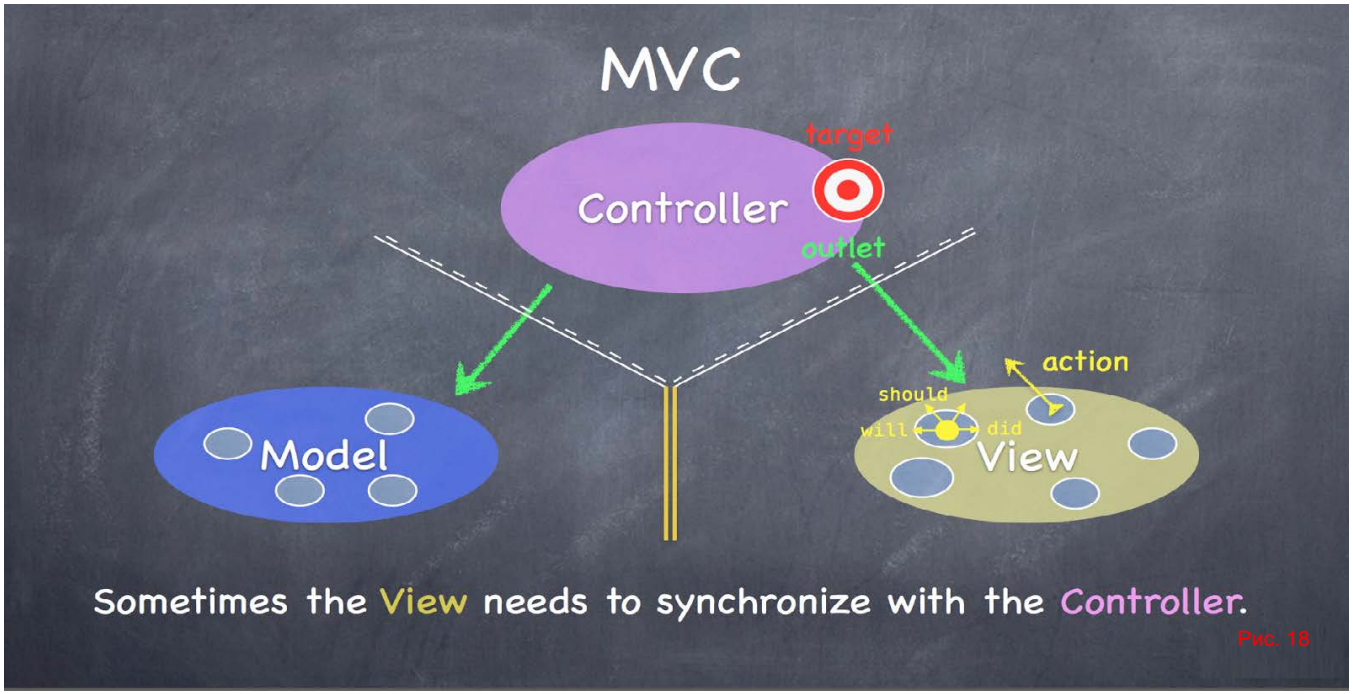

MVC
target
Controller
outlet
action
should
will
did
Model
View
Sometimes the View needs to synchronize with the Controller.

Рис. 18

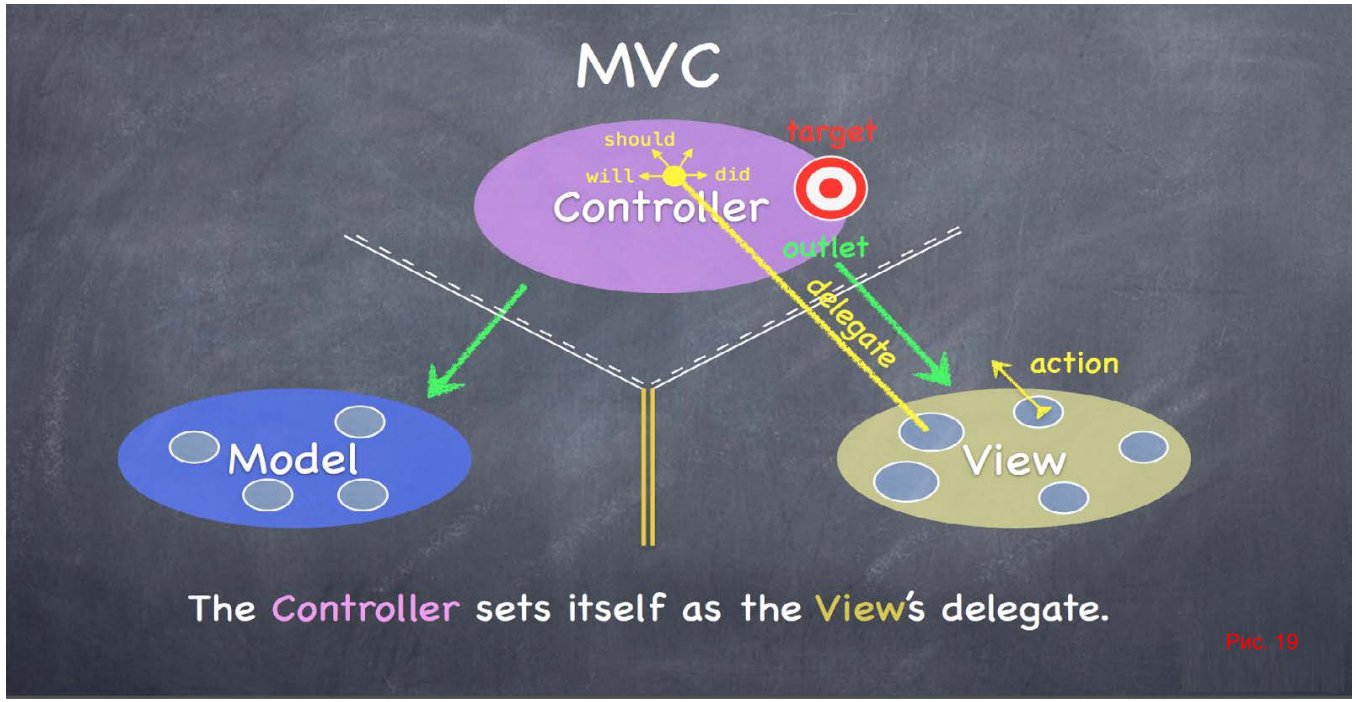

MVC
should
will
did
Controller
target
outlet
delegate
action
Model
View
The Controller sets itself as the View's delegate.

Рис. 19

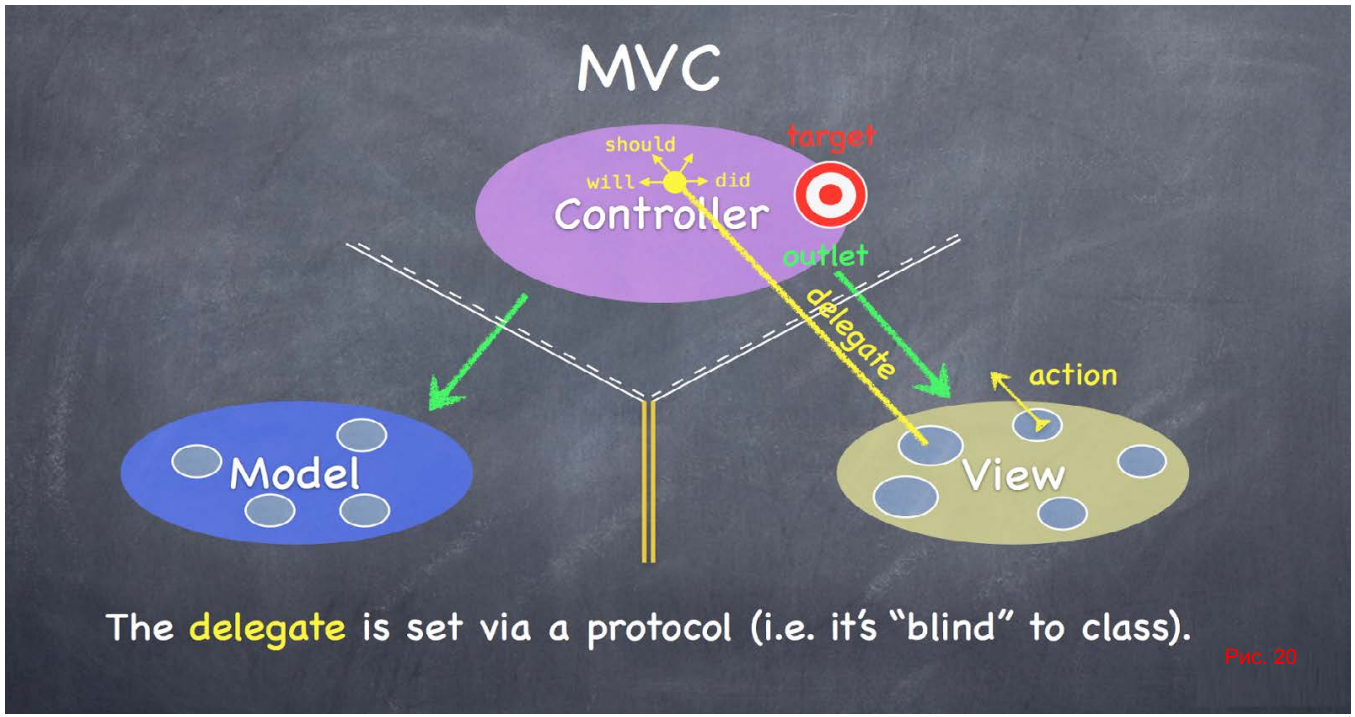

MVC
should
will
did
Controller
target
outlet
delegate
action
Model
View
The delegate is set via a protocol (i.e. it's "blind" to class).

Рис. 20

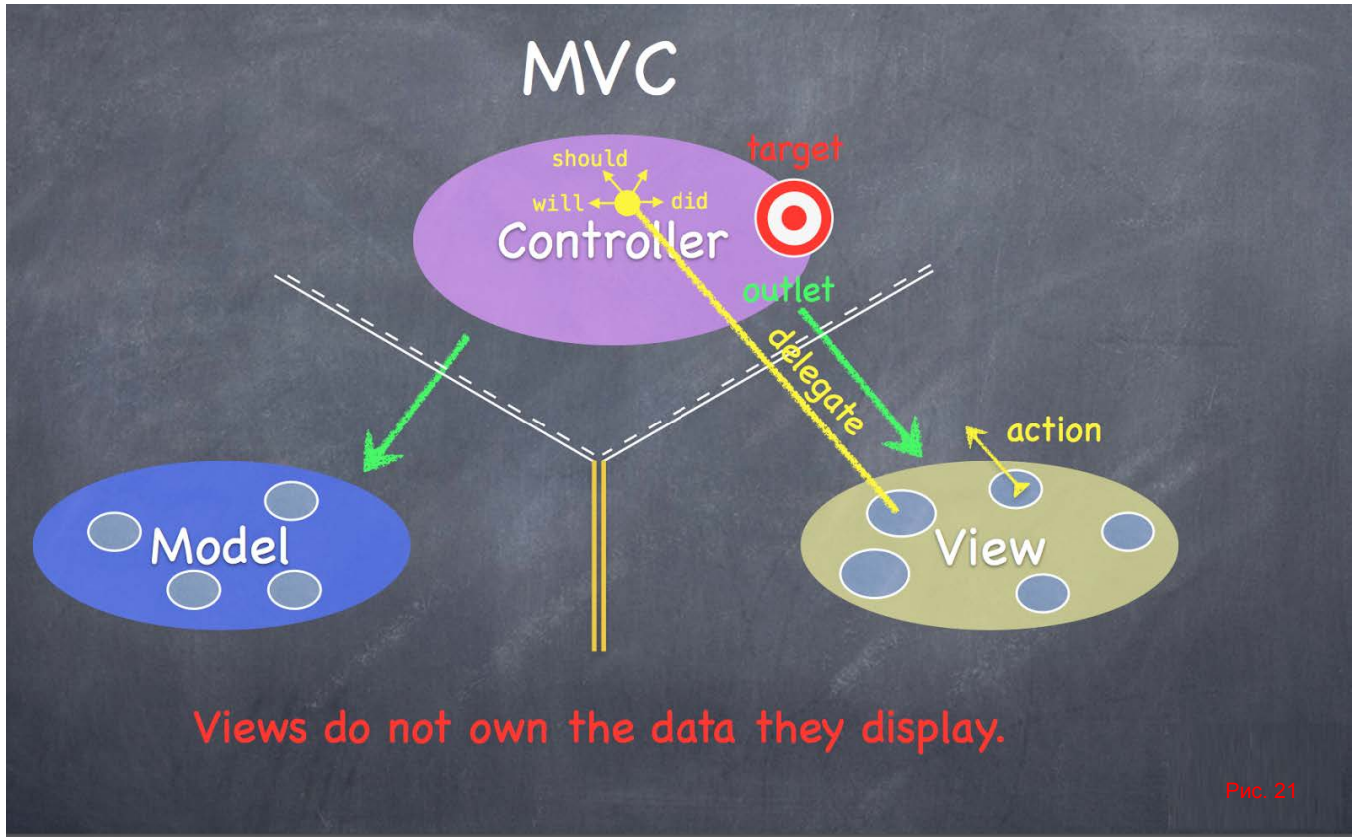

MVC
should
will
did
Controller
target
outlet
delegate
action
Model
View
Views do not own the data they display.

Рис. 21

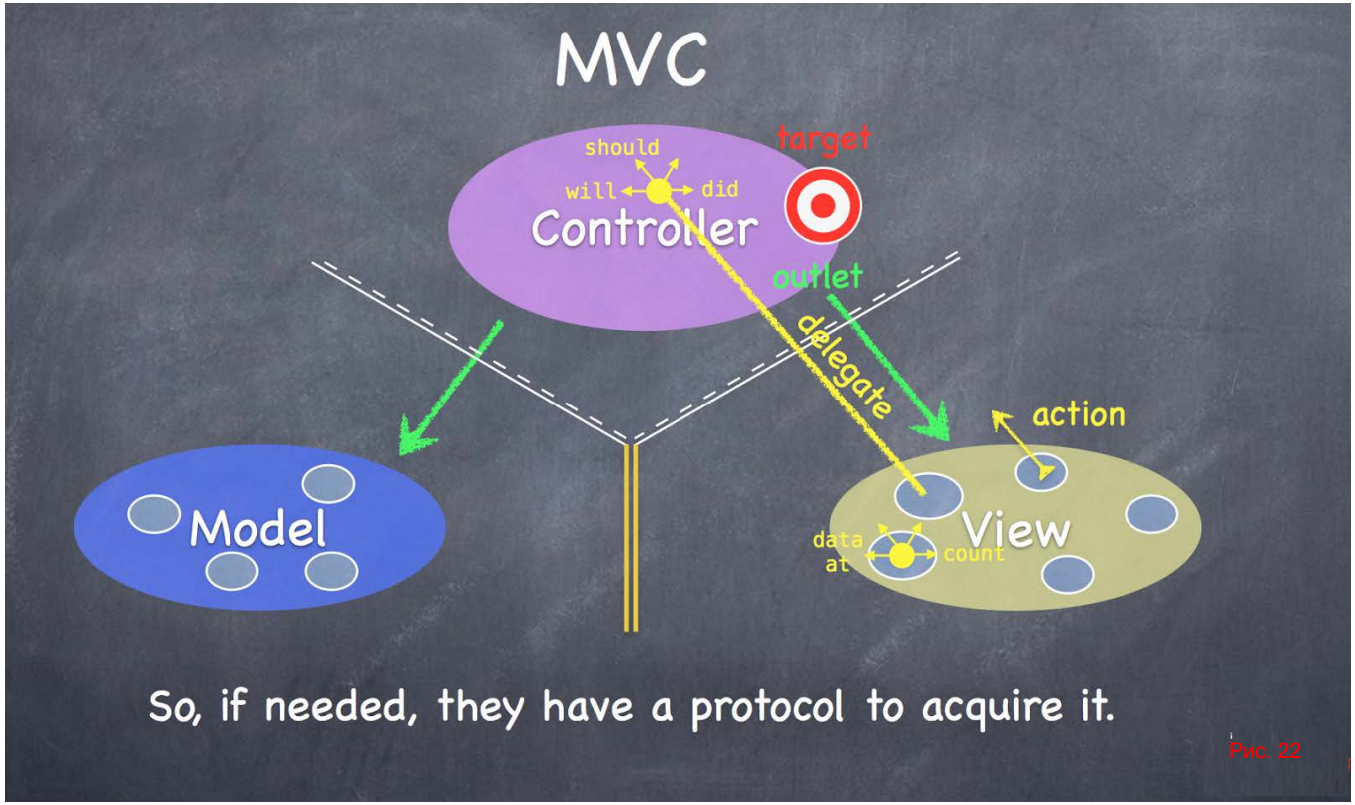

MVC
should
will
did
Controller
target
outlet
delegate
action
Model
data
at
count
View
So, if needed, they have a protocol to acquire it.

Рис. 22

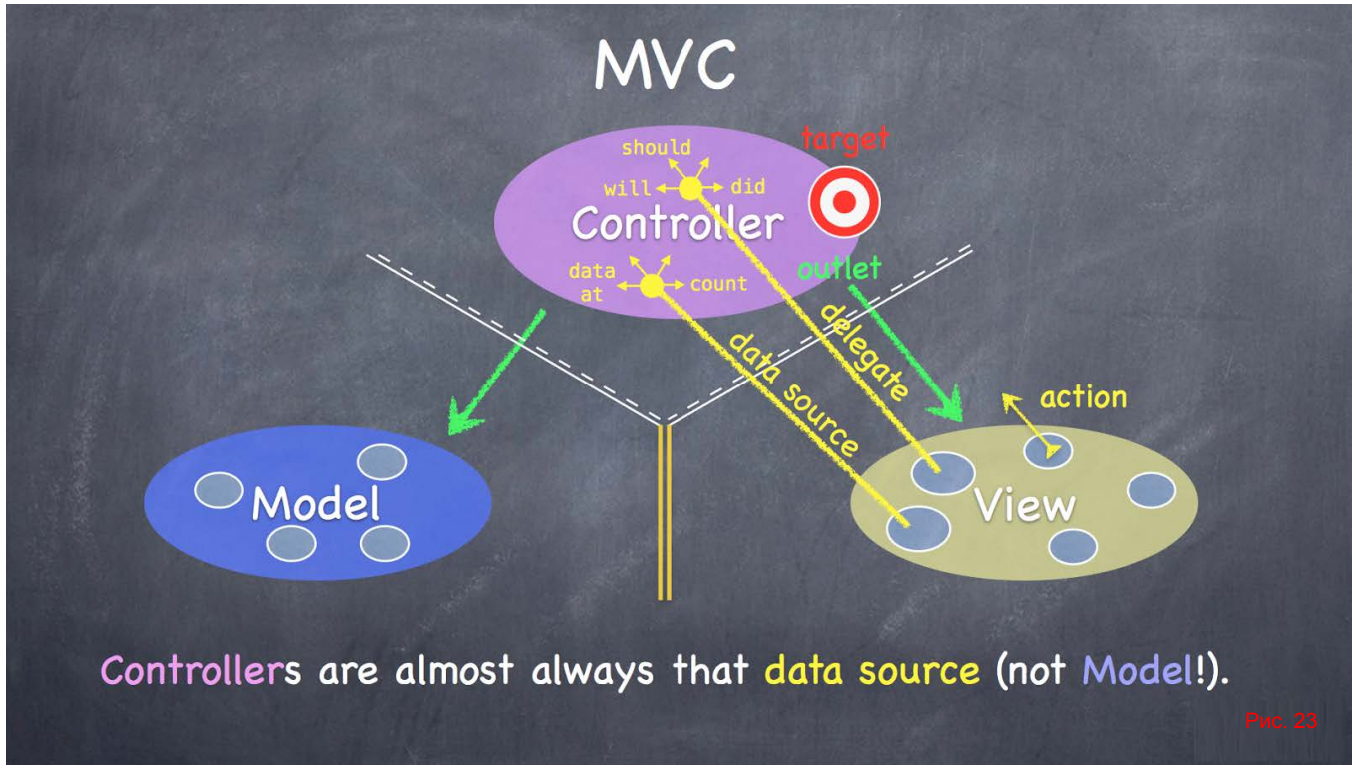

MVC
should
will
did
Controller
data
at
count
target
outlet
delegate
data source
action
Model
View
Controllers are almost always that data source (not Model!).

Рис. 23

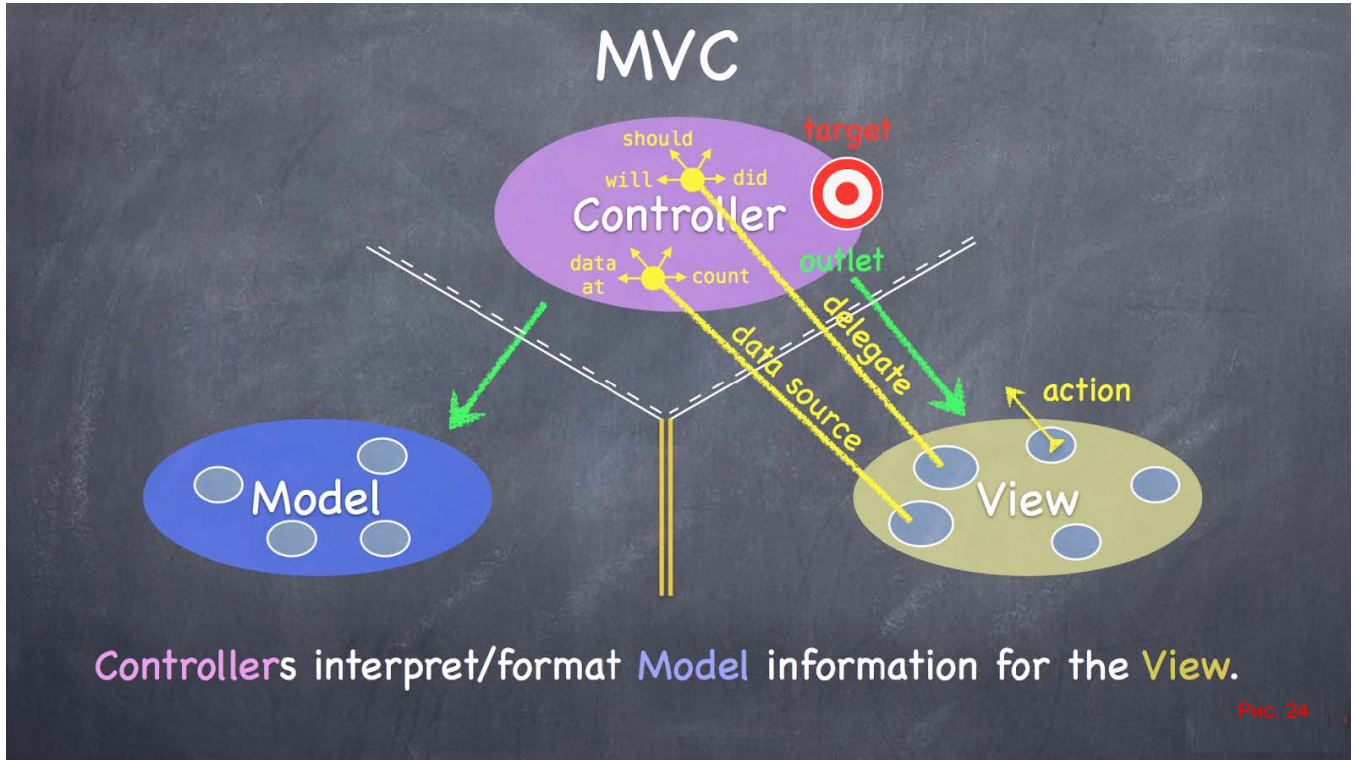

MVC
should
will
did
Controller
target
data
at
count
outlet
delegate
data source
action
Model
View
Controllers interpret/format Model information for the View.

Рис. 24

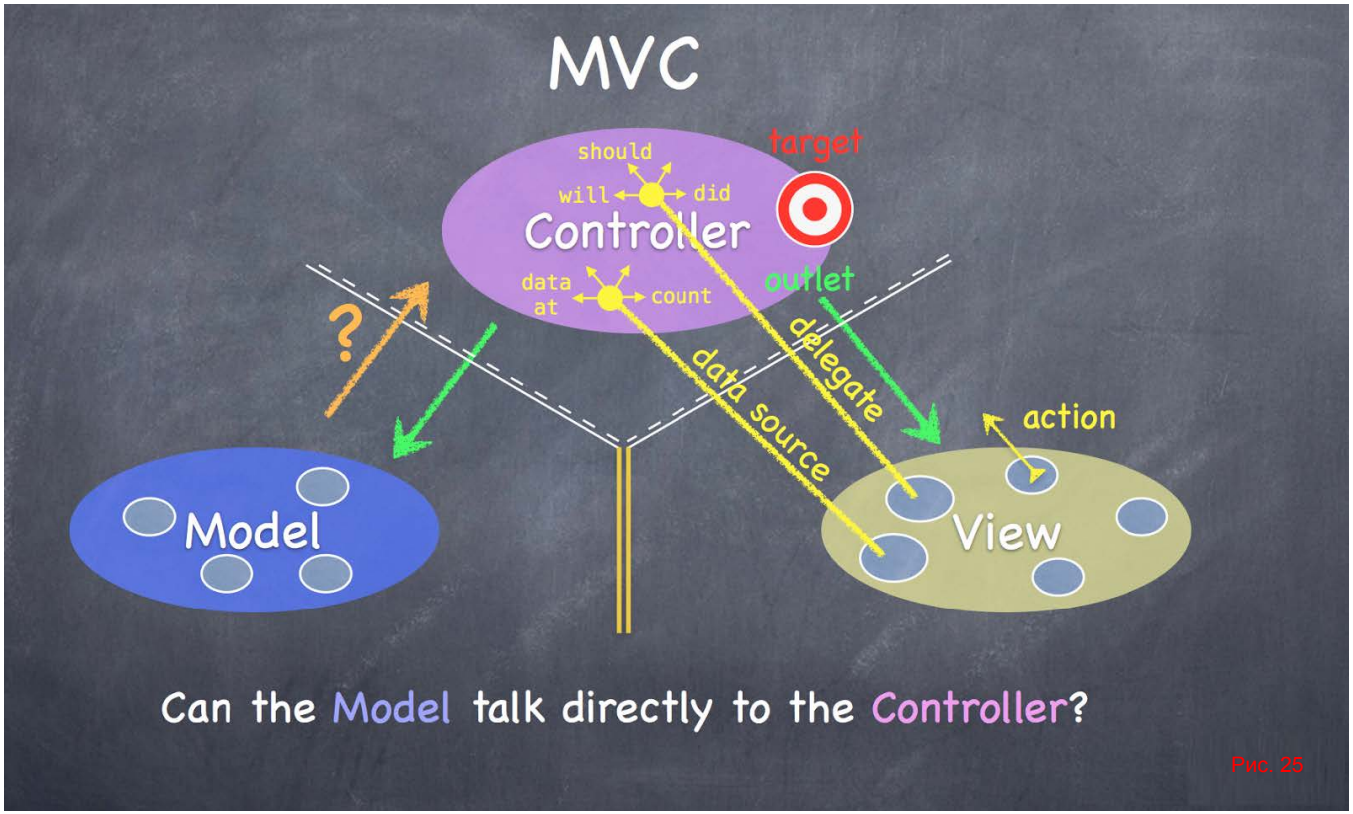

MVC
should
will
did
Controller
data
at
count
target
outlet
delegate
data source
action
?
Model
View
Can the Model talk directly to the Controller?

Рис. 25

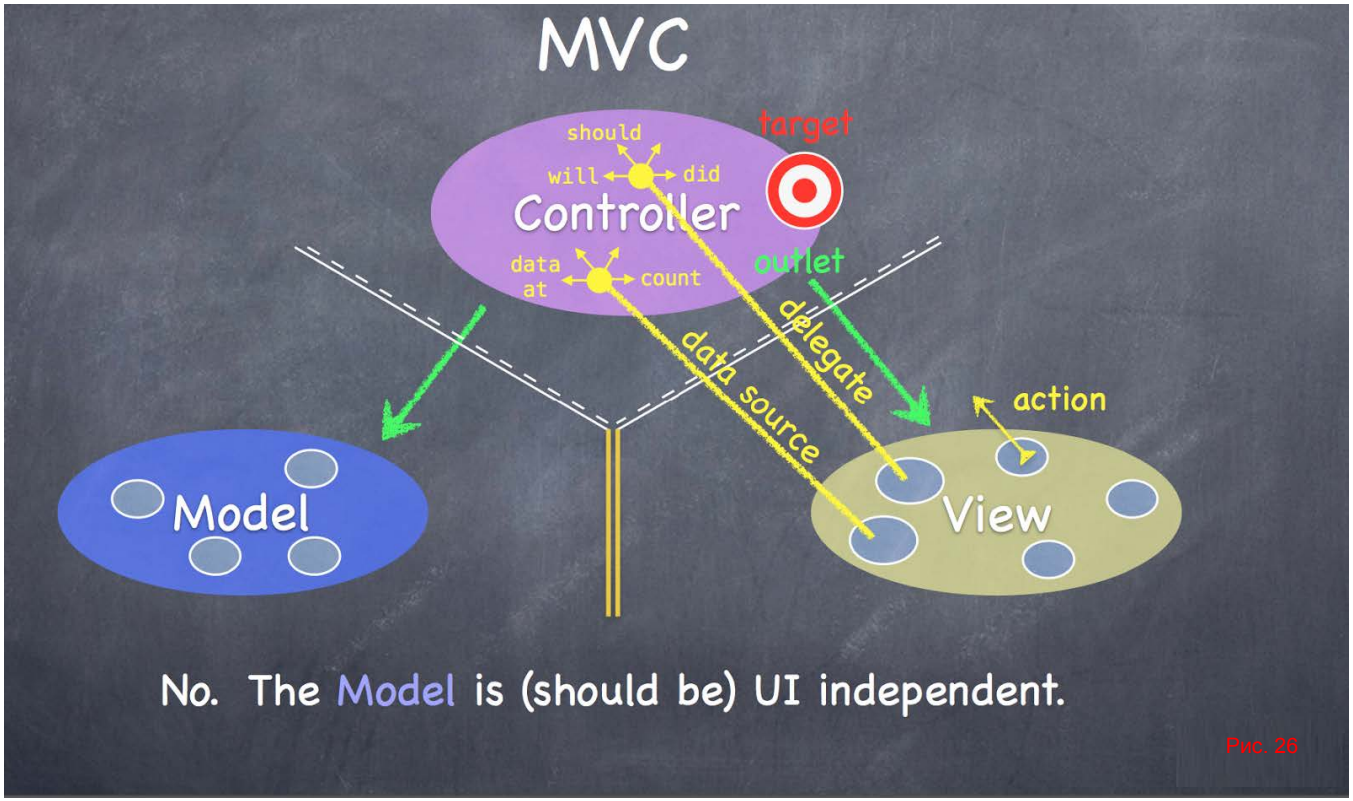
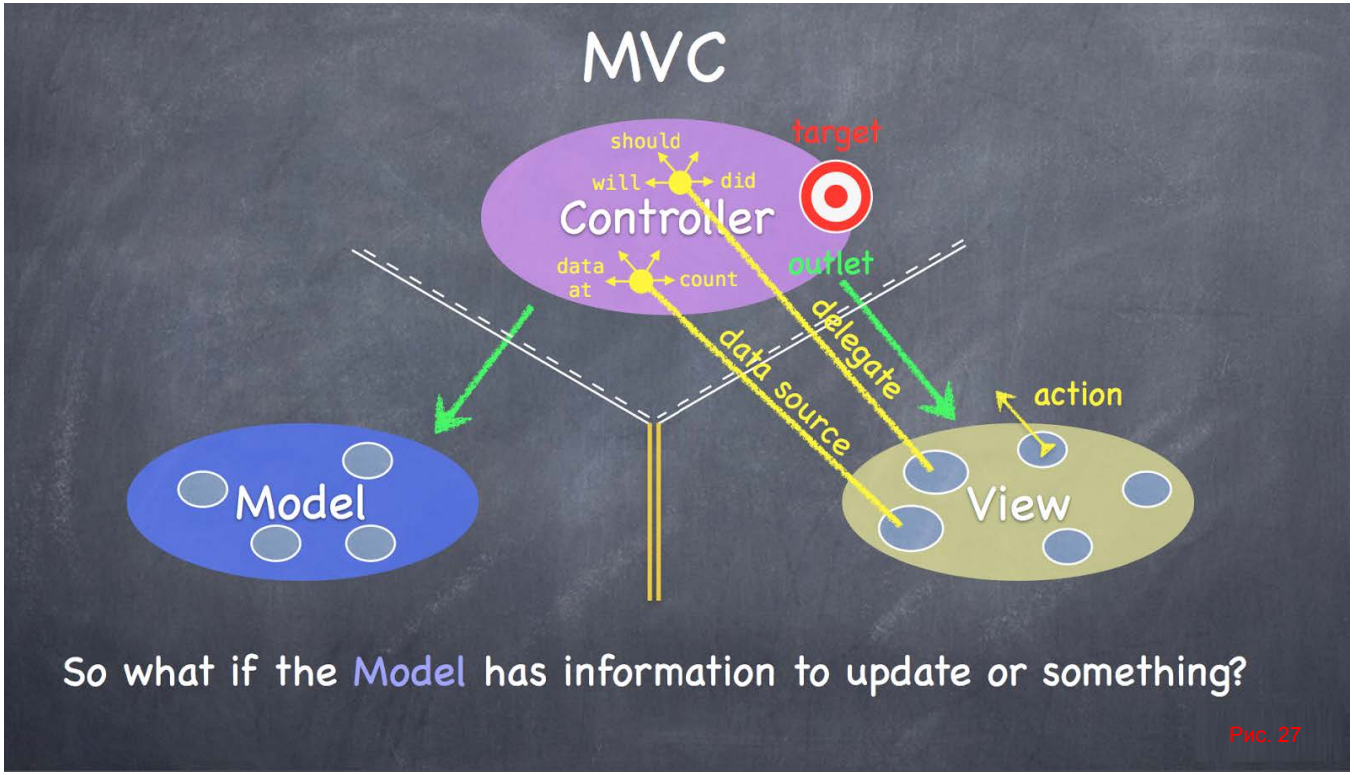

MVC
should
will
did
Controller
data
at
count
target
outlet
delegate
data source
action
Model
View
No. The Model is (should be) UI independent.

Рис. 26


MVC
should
will
did
Controller
target
data
at
count
outlet
delegate
data source
action
Model
View
So what if the Model has information to update or something?
Рис. 27

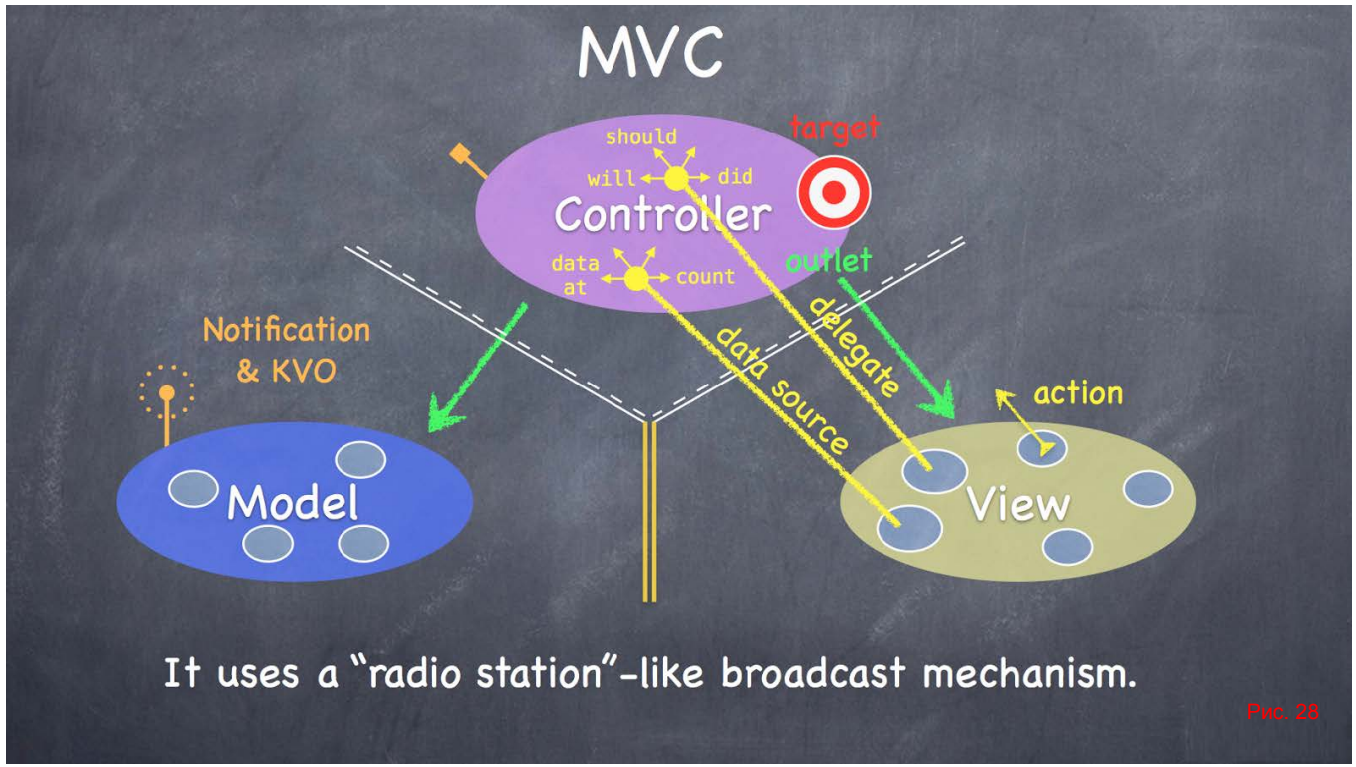

MVC
should
will
did
Controller
target
data
at
count
outlet
Notification
& KVO
delegate
data source
action
Model
View
It uses a "radio station"-like broadcast mechanism.

Рис. 28

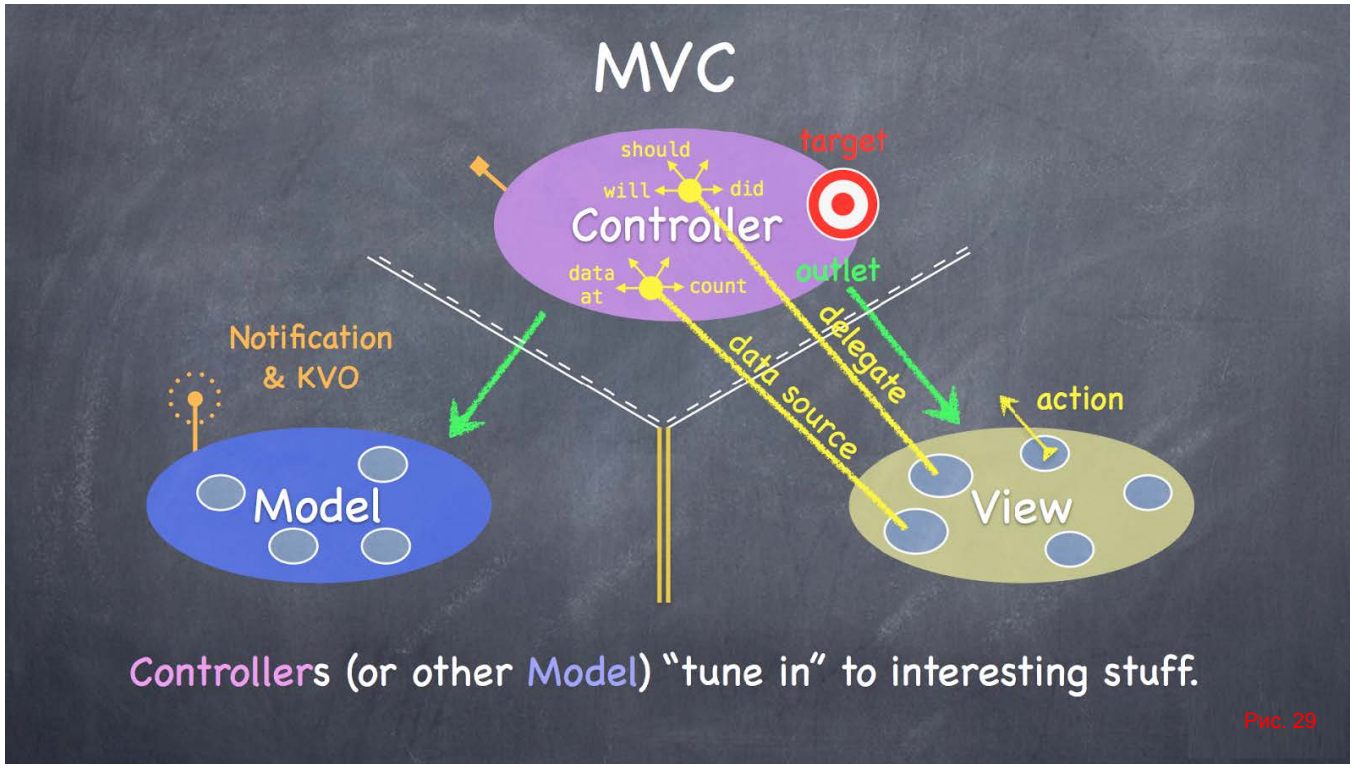

MVC
should
will
did
Controller
target
data
at
count
outlet
delegate
data source
Notification
& KVO
action
Model
View
Controllers (or other Model) "tune in" to interesting stuff.

Рис. 29

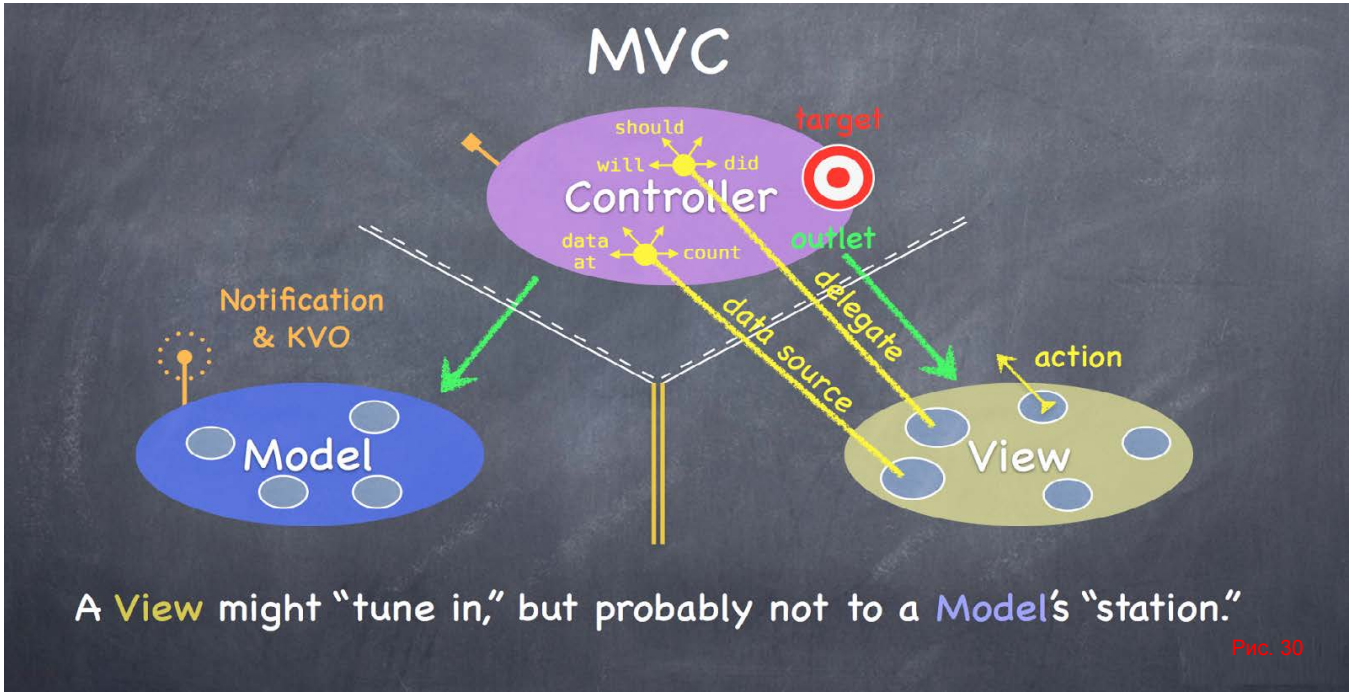

MVC
should
will
did
Controller
target
data
at
count
outlet
delegate
data source
Notification
& KVO
action
Model
View
A View might "tune in," but probably not to a Model's "station."

Рис. 30

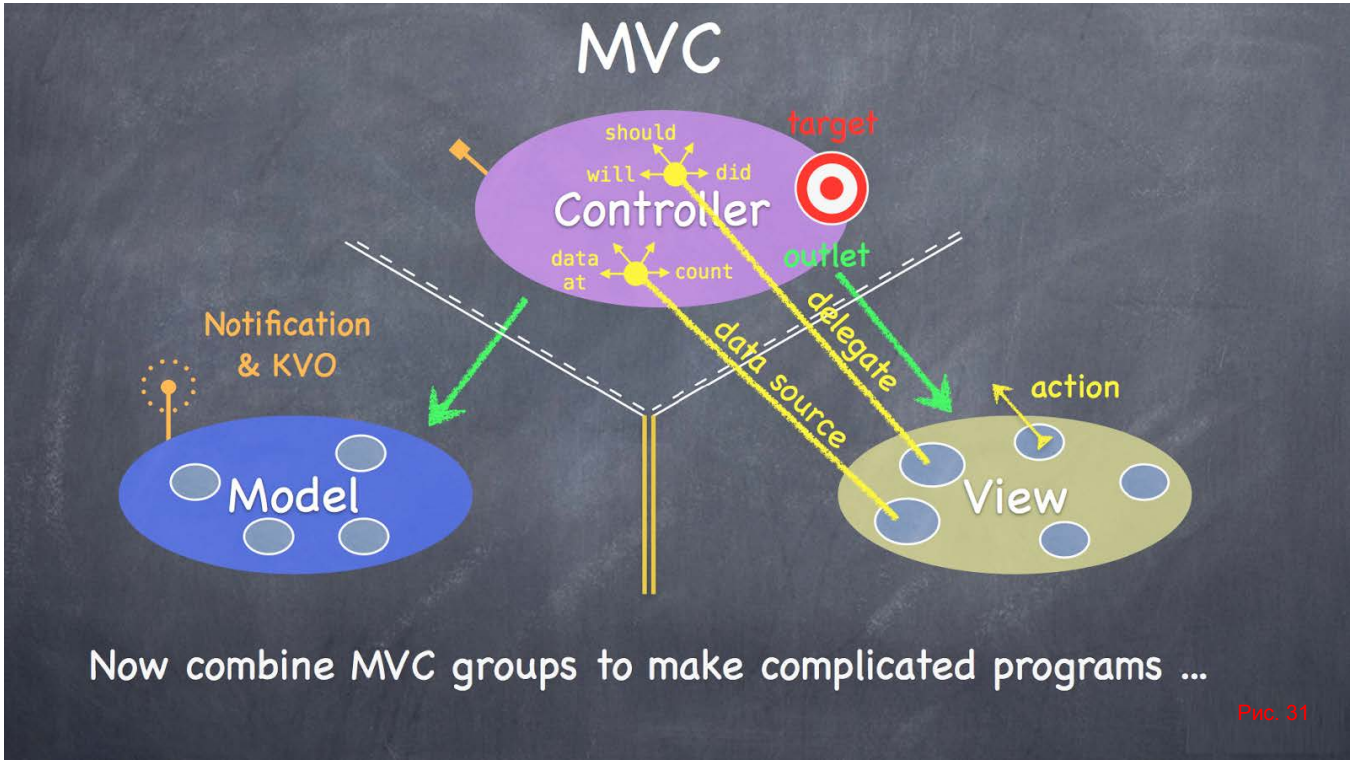

MVC
should
will
did
Controller
target
data
at
count
outlet
Notification
& KVO
delegate
data source
action
Model
View
Now combine MVC groups to make complicated programs ...

Рис. 31

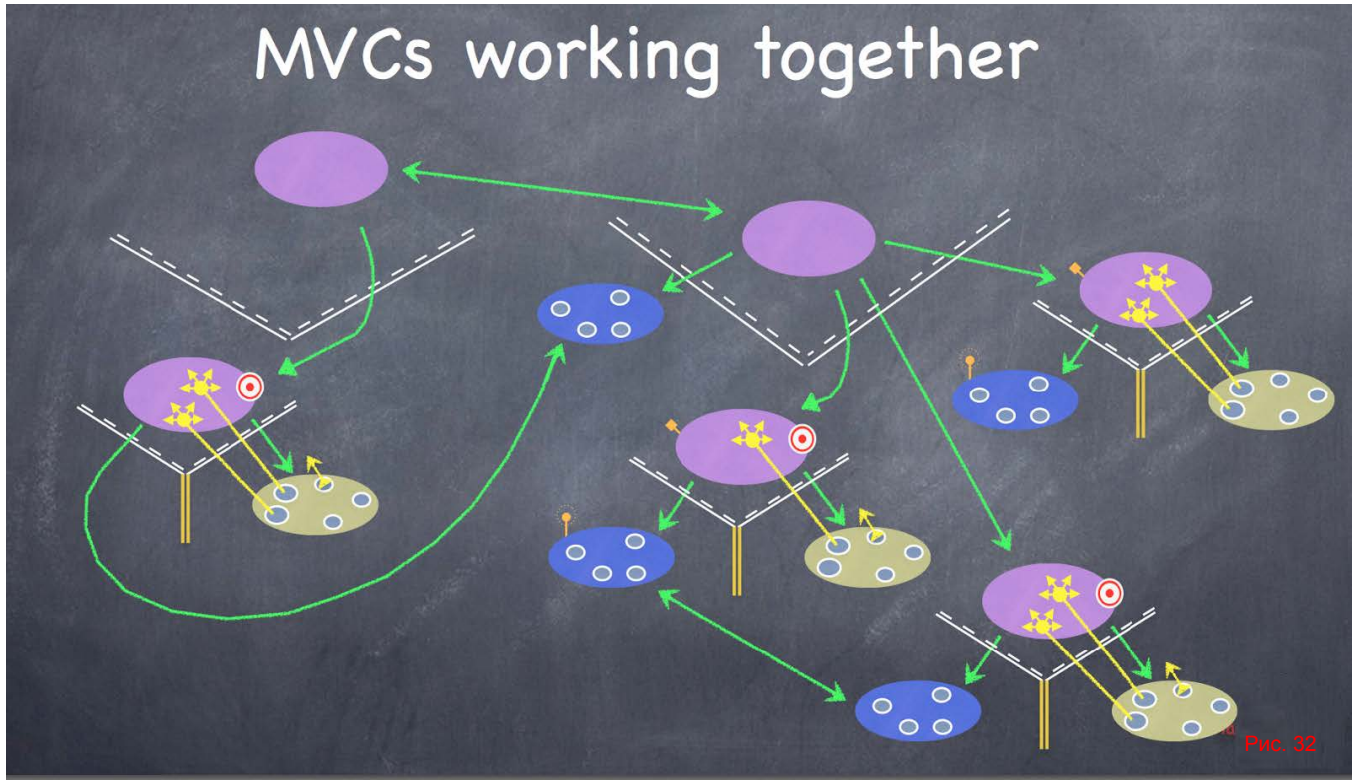


Рис. 32

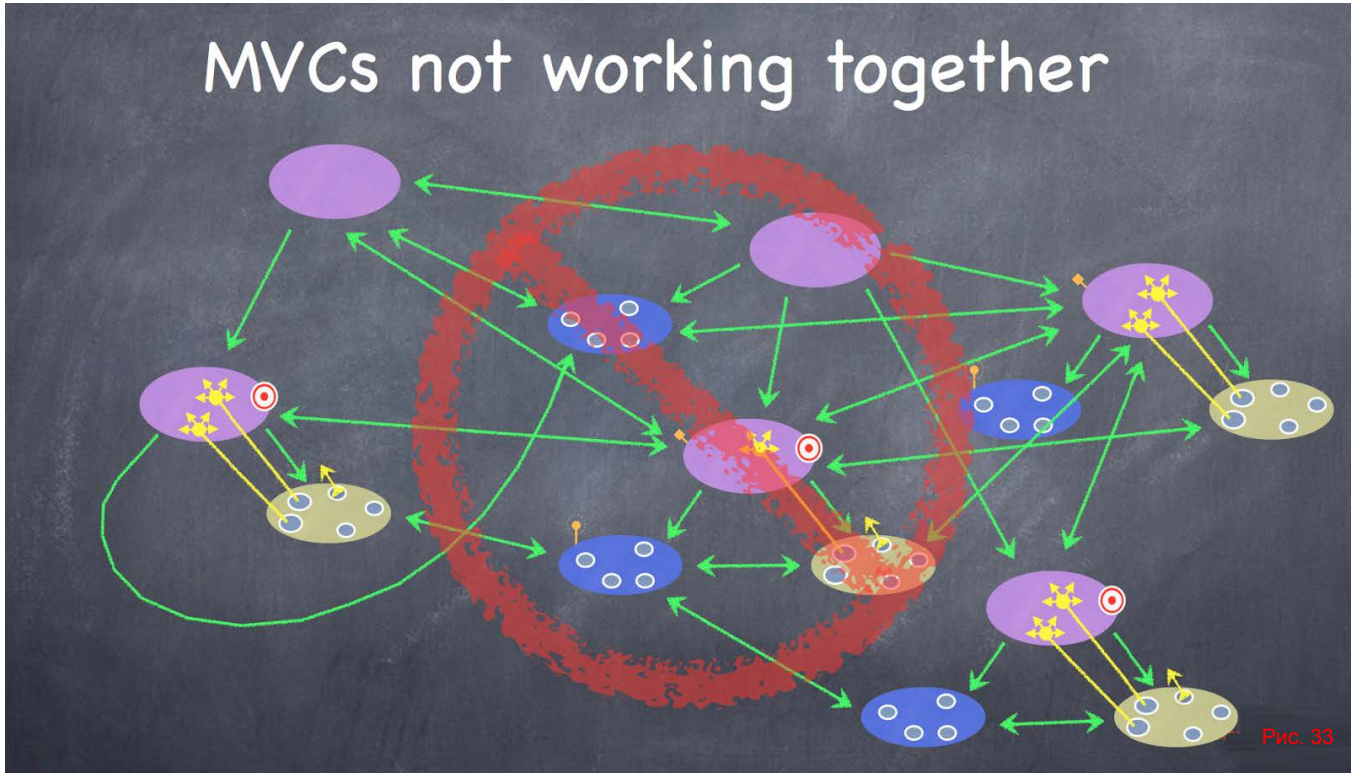


Рис. 33

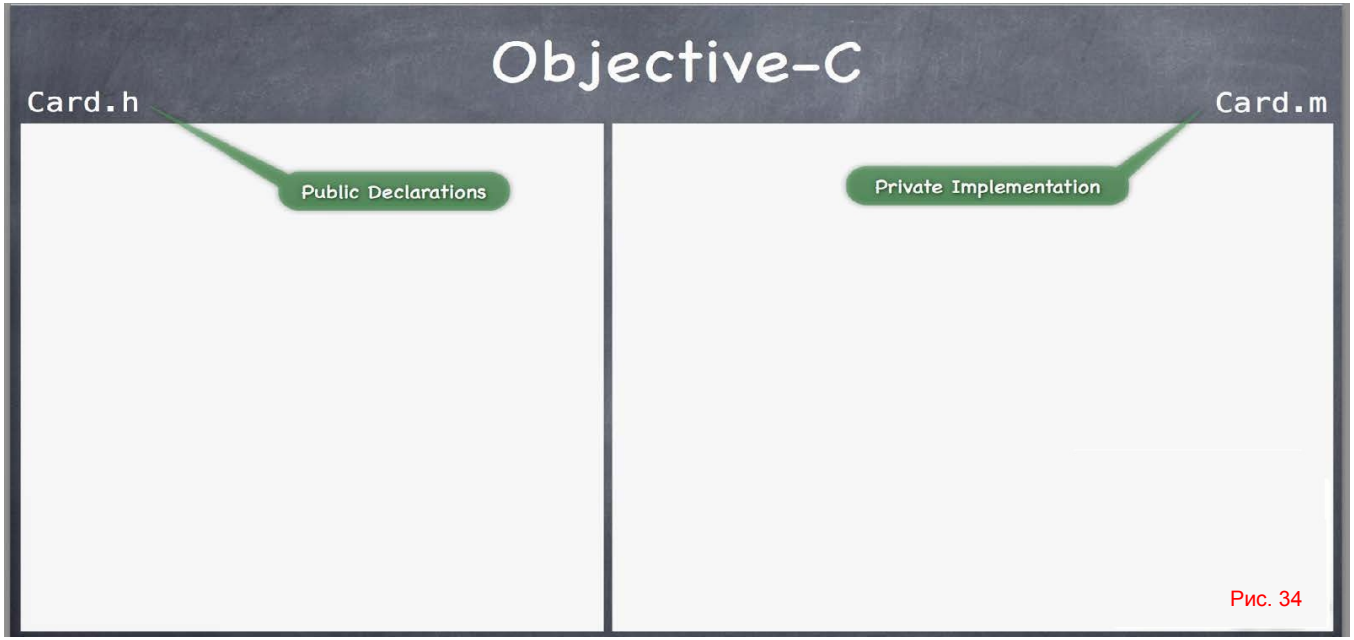

Objective-C
Card.h
Card.m
Public Declarations
Private Implementation

Рис. 34

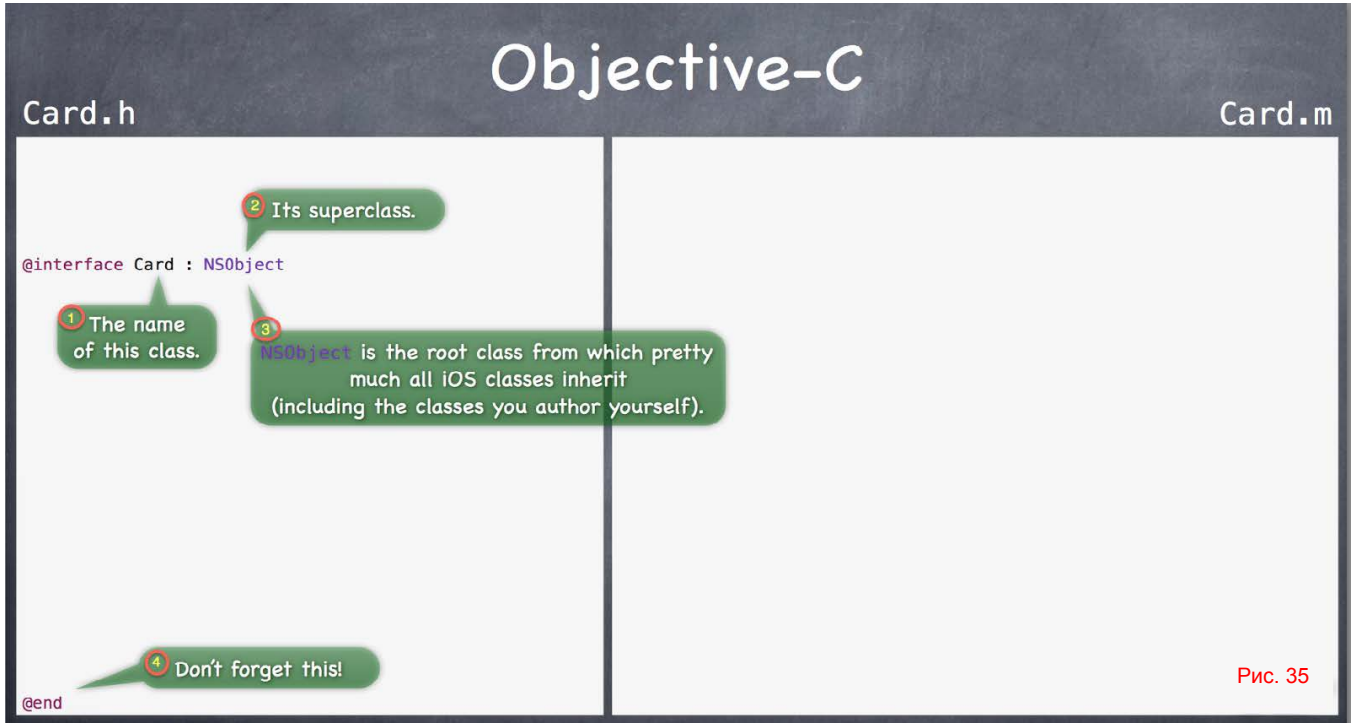

Objective-C
Card.h
Card.m
2 Its superclass.
@interface Card : NSObject
1 The name of this class.
3 NSObject is the root class from which pretty much all iOS classes inherit (including the classes you author yourself).
4 Don't forget this!
@end

Рис. 35

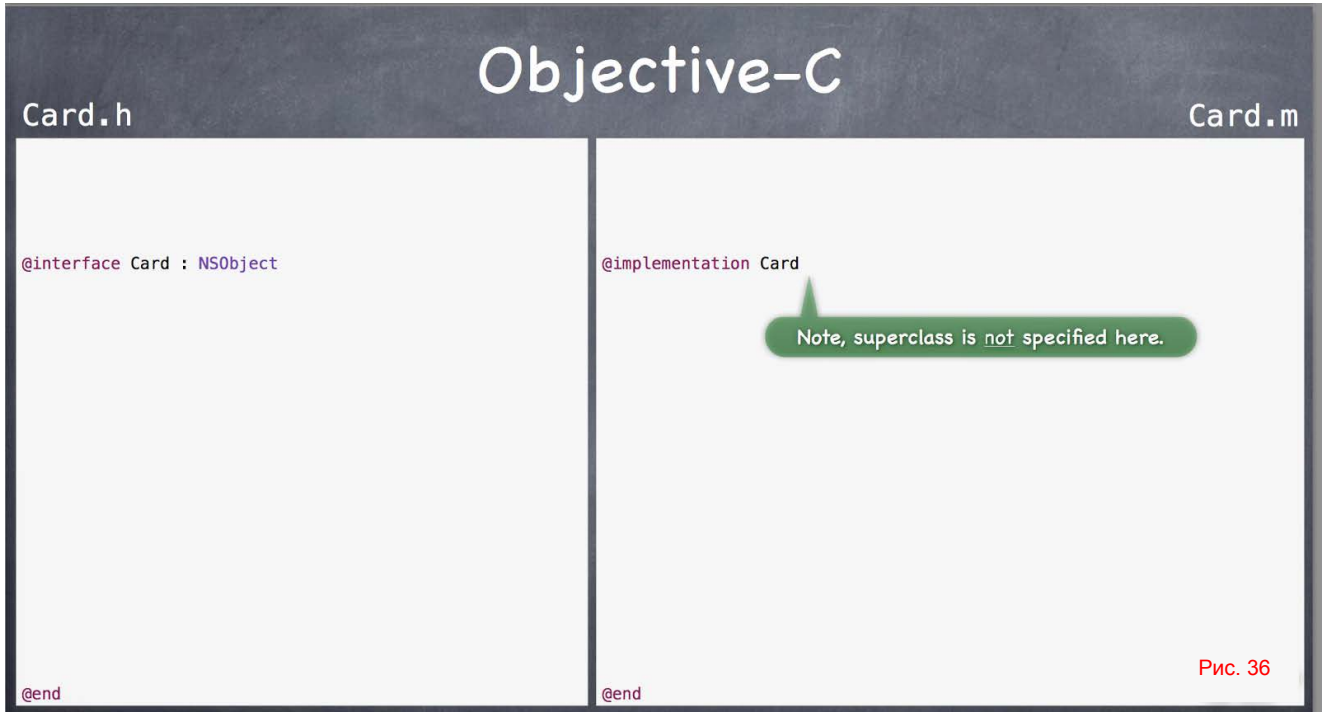

Objective-C
Card.h
Card.m
@interface Card : NSObject
@end
@implementation Card
Note, superclass is not specified here.
@end

Рис. 36

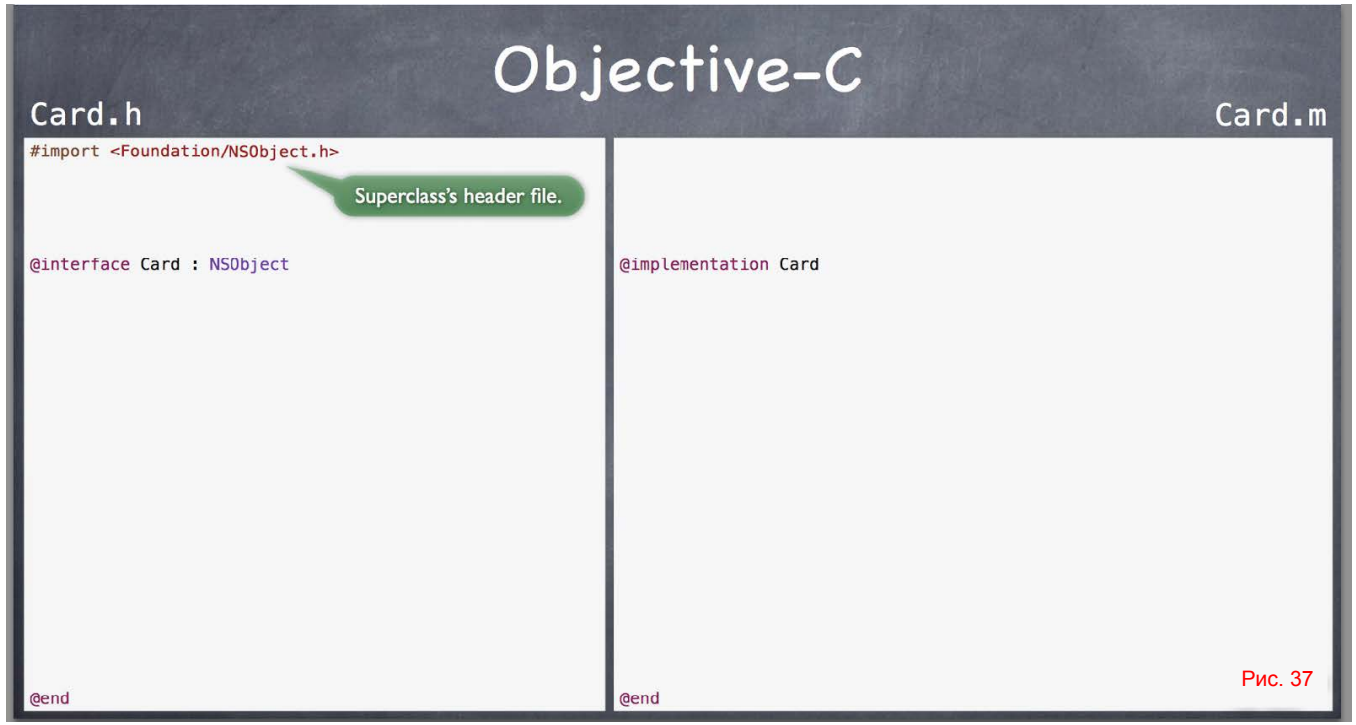

Objective-C
Card.h
#import <Foundation/NSObject.h>
Superclass's header file.
@interface Card : NSObject
@end
Card.m
@implementation Card
@end

Рис. 37

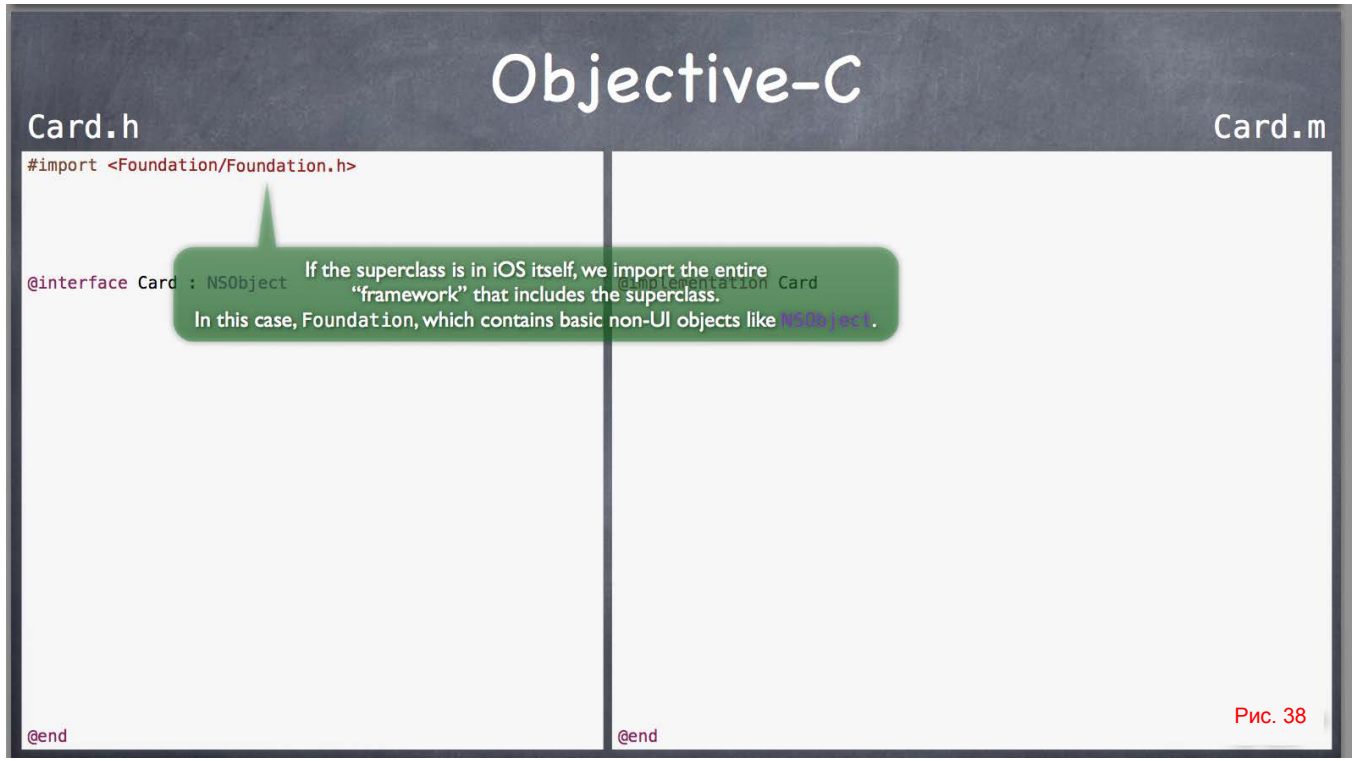

Objective-C
Card.h
Card.m
#import <Foundation/Foundation.h>
@interface Card : NSObject
@end
@implementation Card
@end
If the superclass is in iOS itself, we import the entire "framework" that includes the superclass.
In this case, Foundation, which contains basic non-UI objects like NSObject.

Рис. 38

Objective-C

Card.h

```
@import Foundation;

@interface Card : NSObject

@end
```

In fact, in iOS 7 (only), there is special syntax for importing an entire framework called @import.

Card.m

```
@implementation Card

@end
```

Рис. 39

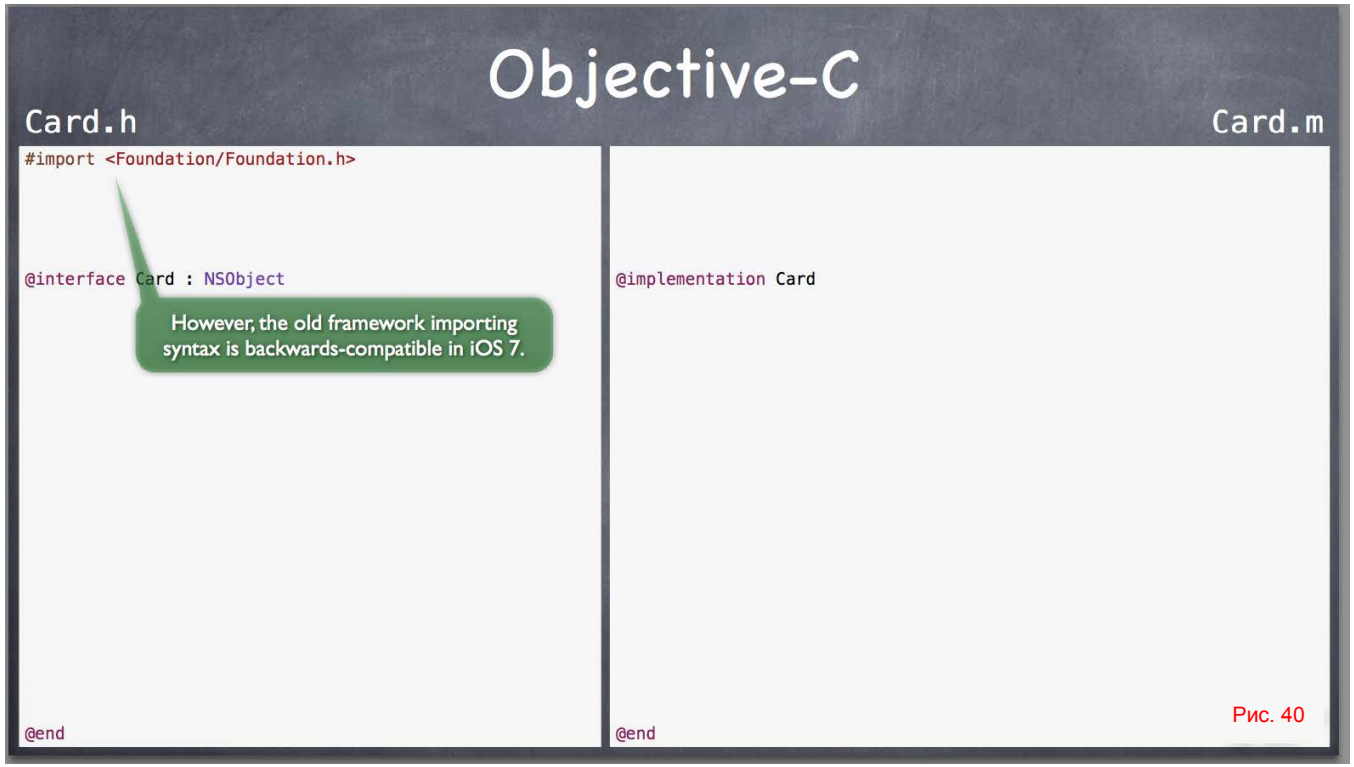

Objective-C
Card.h
Card.m
#import <Foundation/Foundation.h>
@interface Card : NSObject
However, the old framework importing syntax is backwards-compatible in iOS 7.
@end
@implementation Card
@end

Рис. 40

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject









@end
```

Card.m

```
#import "Card.h"



@implementation Card









@end
```

Our own header file must be imported into our implementation file.

Рис. 41

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject









@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card









@end
```

Private declarations can go here.

Рис. 42

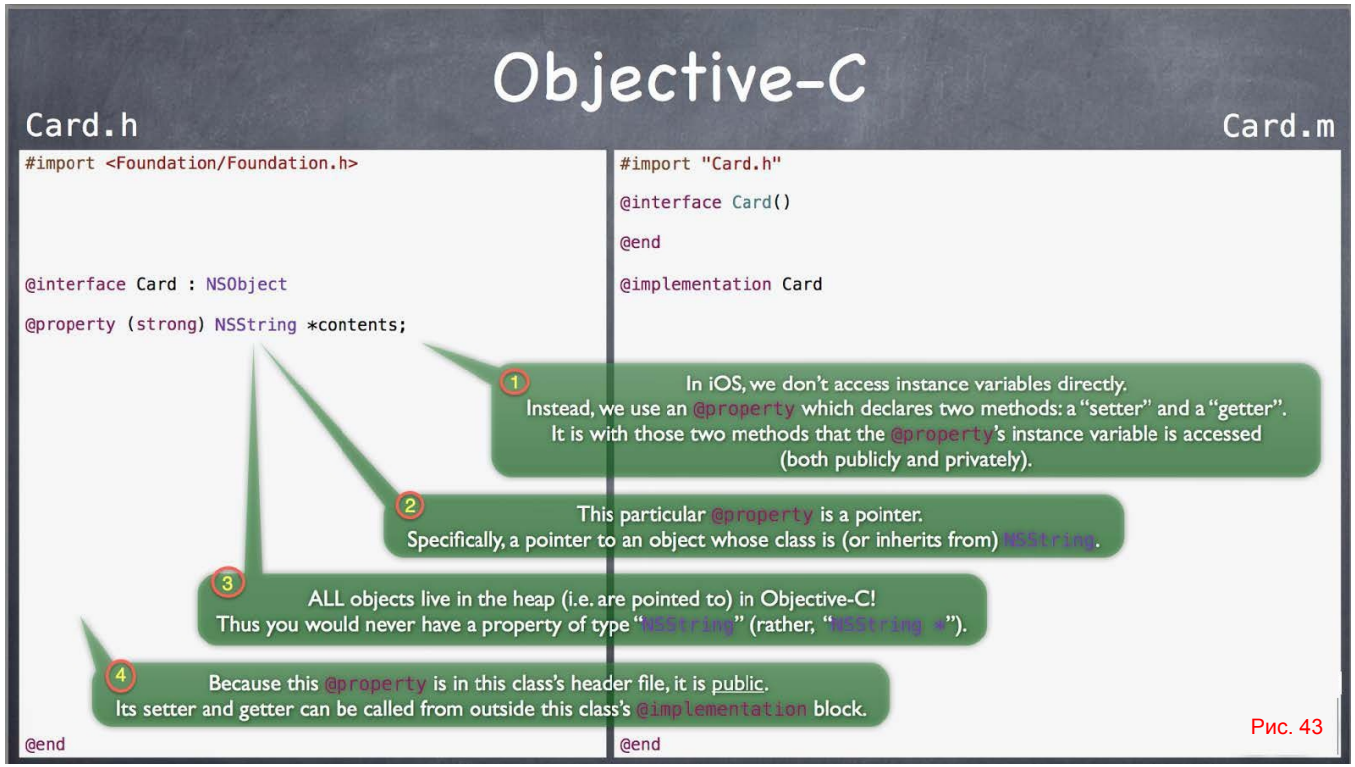

Objective-C
Card.h
Card.m
#import <Foundation/Foundation.h>
@interface Card : NSObject
@property (strong) NSString *contents;
@end
#import "Card.h"
@interface Card()
@end
@implementation Card
@end
1
In iOS, we don't access instance variables directly.
Instead, we use an @property which declares two methods: a "setter" and a "getter".
It is with those two methods that the @property's instance variable is accessed
(both publicly and privately).
2
This particular @property is a pointer.
Specifically, a pointer to an object whose class is (or inherits from) NSString.
3
ALL objects live in the heap (i.e. are pointed to) in Objective-C!
Thus you would never have a property of type "NSString" (rather, "NSString *").
4
Because this @property is in this class's header file, it is public.
Its setter and getter can be called from outside this class's @implementation block.

Рис. 43

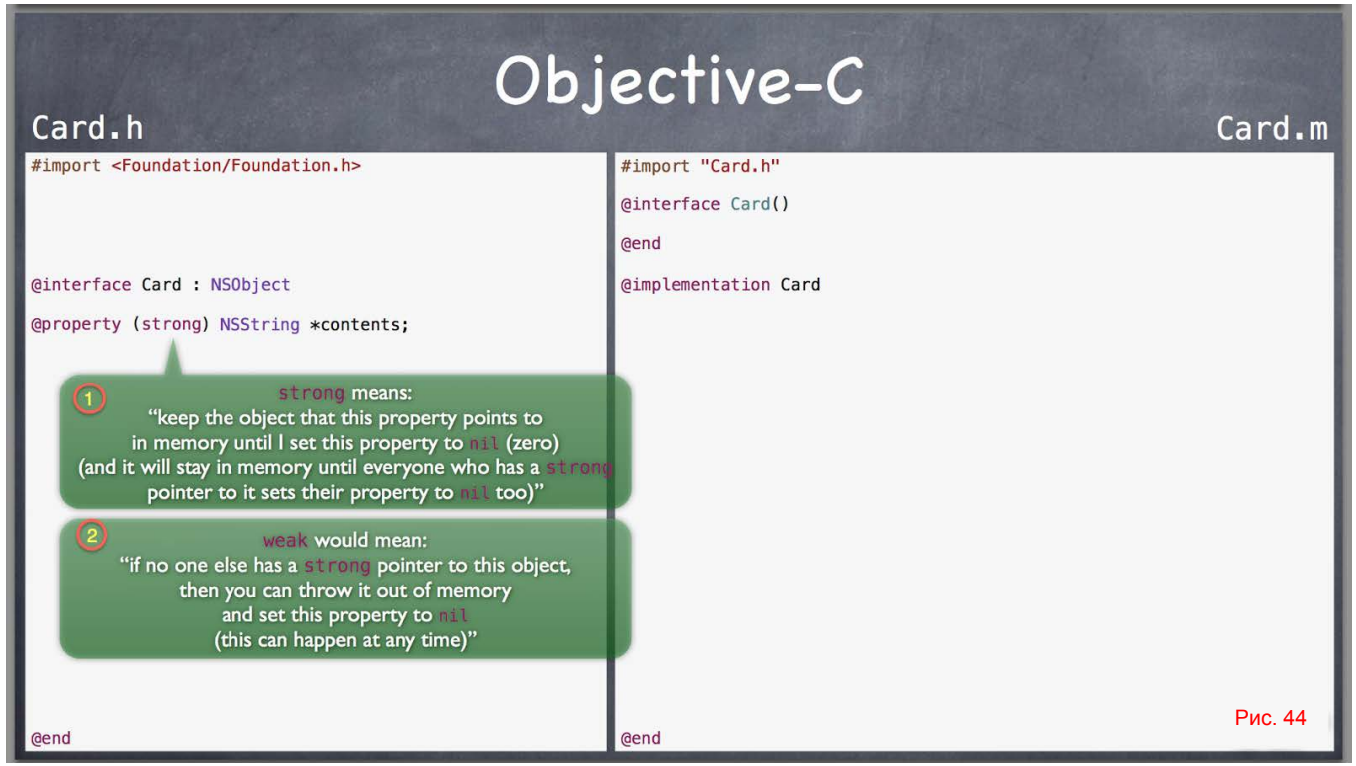

Objective-C
Card.h
Card.m
#import <Foundation/Foundation.h>
@interface Card : NSObject
@property (strong) NSString *contents;
@end
#import "Card.h"
@interface Card()
@end
@implementation Card
@end
1
strong means:
"keep the object that this property points to
in memory until I set this property to nil (zero)
(and it will stay in memory until everyone who has a strong
pointer to it sets their property to nil too)"
2
weak would mean:
"if no one else has a strong pointer to this object,
then you can throw it out of memory
and set this property to nil
(this can happen at any time)"

Рис. 44

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;



@end
```

nonatomic means:
"access to this property is not thread-safe".
We will always specify this for object pointers in this course.
If you do not, then the compiler will generate locking code that will complicate your code elsewhere.

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card



@end
```

Рис. 45

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@end
```

1 This is the @property implementation that the compiler generates automatically for you (behind the scenes).
You are welcome to write the setter or getter yourself, but this would only be necessary if you needed to do something in addition to simply setting or getting the value of the property.

Card.m

```
#import "Card.h"

@interface Card ()

@end

@implementation Card

@synthesize contents = _contents;

- (NSString *)contents
{
    return _contents;
}

- (void)setContents:(NSString *)contents
{
    _contents = contents;
}

@end
```

2 This @synthesize is the line of code that actually creates the backing instance variable that is set and gotten.
Notice that by default the backing variable's name is the same as the property's name but with an underbar in front.

Рис. 46

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;









@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card
```

Because the compiler takes care of everything you need to implement a property, it's usually only one line of code (the `@property` declaration) to add one to your class.

```
@end
```

Рис. 47

Objective-C

Card.h

```
#import <Foundation/Foundation.h>

@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic) BOOL chosen;
@property (nonatomic) BOOL matched;

@end
```

2 Notice no strong or weak here.
Primitive types are not stored in the heap, so there's no need to specify how the storage for them in the heap is treated.

Let's look at some more properties.
1 These are not pointers.
They are simple BOOLs.

Properties can be
3 any C type.
That includes int, float, etc., even C structs.

C does not define a "boolean" type.
This BOOL is an Objective-C typedef.
4 It's values are YES or NO.

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card

@end
```

Рис. 48

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic) BOOL chosen;
@property (nonatomic) BOOL matched;









@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card

@synthesize chosen = _chosen;
@synthesize matched = _matched;

- (BOOL)chosen
{
    return _chosen;
}
- (void)setChosen:(BOOL)chosen
{
    _chosen = chosen;
}

- (BOOL)matched
{
    return _matched;
}
- (void)setMatched:(BOOL)matched
{
    _matched = matched;
}

@end
```

Here's what the compiler is doing behind the scenes for these two properties.

Рис. 49

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;









@end
```

1. It is actually possible to change the name of the getter that is generated. The only time you'll ever see that done in this class (or anywhere probably) is boolean getters.

2. This is done simply to make the code "read" a little bit nicer. You'll see this in action later.

Card.m

```
#import "Card.h"

@interface Card()
@implementation Card

@synthesize chosen = _chosen;
@synthesize matched = _matched;

- (BOOL)isChosen
{
    return _chosen;
}
- (void)setChosen:(BOOL)chosen
{
    _chosen = chosen;
}

- (BOOL)isMatched
{
    return _matched;
}
- (void)setMatched:(BOOL)matched
{
    _matched = matched;
}

@end
```

3. Note change in getter method.

4. Note change in getter method.

Рис. 50

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card

@end
```

Remember, unless you need to do something besides setting or getting when a property is being set or gotten, the implementation side of this will all happen automatically for you.

Рис. 51

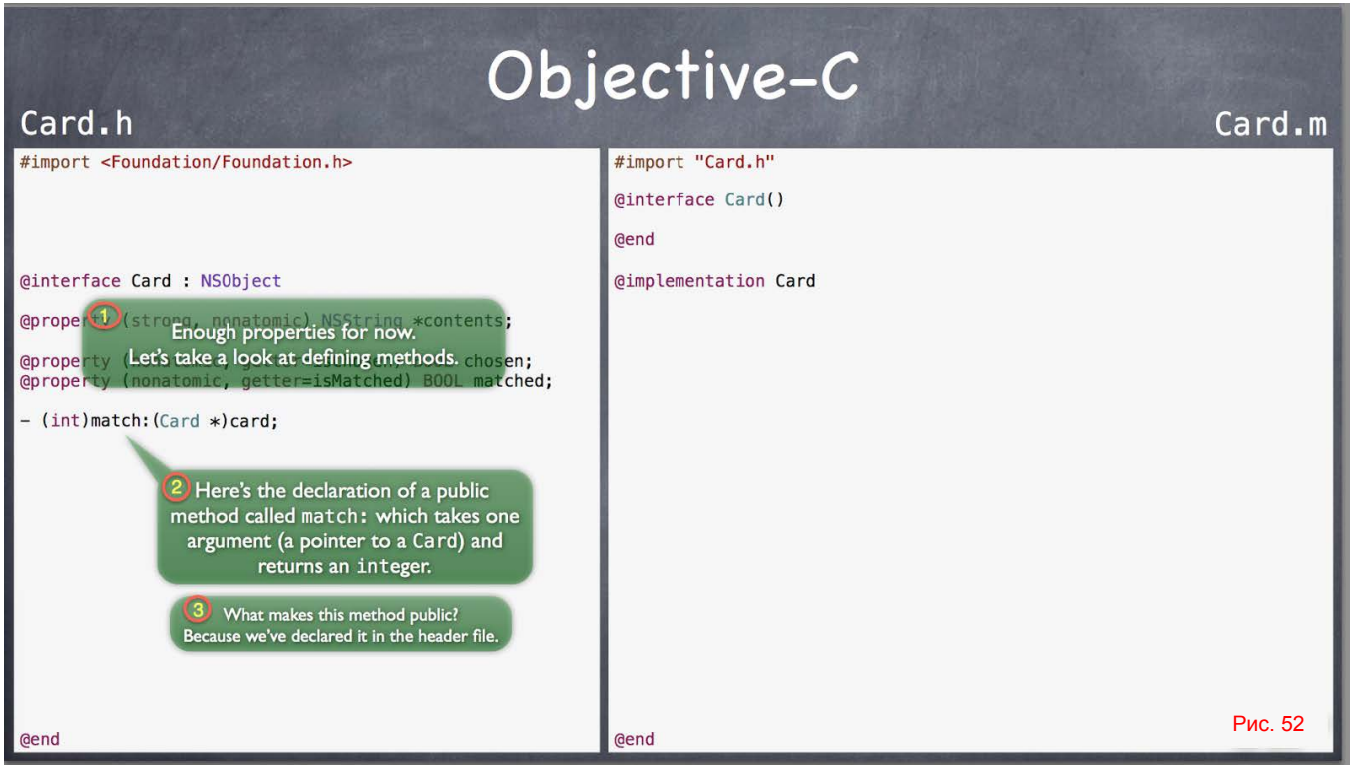

Objective-C
Card.h
Card.m
#import <Foundation/Foundation.h>
@interface Card : NSObject
@property (strong, nonatomic) NSString *contents;
@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;
- (int)match:(Card *)card;
@end
#import "Card.h"
@interface Card()
@end
@implementation Card
@end
1 Enough properties for now. Let's take a look at defining methods.
2 Here's the declaration of a public method called match: which takes one argument (a pointer to a Card) and returns an integer.
3 What makes this method public? Because we've declared it in the header file.

Рис. 52

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

- (int)match:(Card *)card;



@end
```

1 Here's the declaration of a public method called `match:` which takes one argument (a pointer to a `Card`) and returns an `integer`.

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card




- (int)match:(Card *)card
{
    int score = 0;

    return score;
}


@end
```

`match:` is going to return a "score" which says how good a match 2 the passed `card` is to the `Card` that is receiving this message. 0 means "no match", higher numbers mean a better match.

Рис. 53

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

- (int)match:(Card *)card;









@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card



- (int)match:(Card *)card
{
    int score = 0;


    if ([card.contents isEqualToString:self.contents]) {
        score = 1;
    }


    return score;
}



@end
```

1 There's a lot going on here! For the first time, we are seeing the "calling" side of properties (and methods).

2 For this example, we'll return 1 if the passed card has the same contents as we do or 0 otherwise (you could imagine more complex scoring).

Рис. 54

Objective-C

Card.h

```
#import <Foundation/Foundation.h>




@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

- (int)match:(Card *)card;









@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card









- (int)match:(Card *)card
{
    int score = 0;


    if ([card.contents isEqualToString:self.contents]) {
        score = 1;
    }


    return score;
}



@end
```

Notice that we are calling the "getter" for the contents @property (both on our self and on the passed card). This calling syntax is called "dot notation." It's only for setters and getters.

Рис. 55

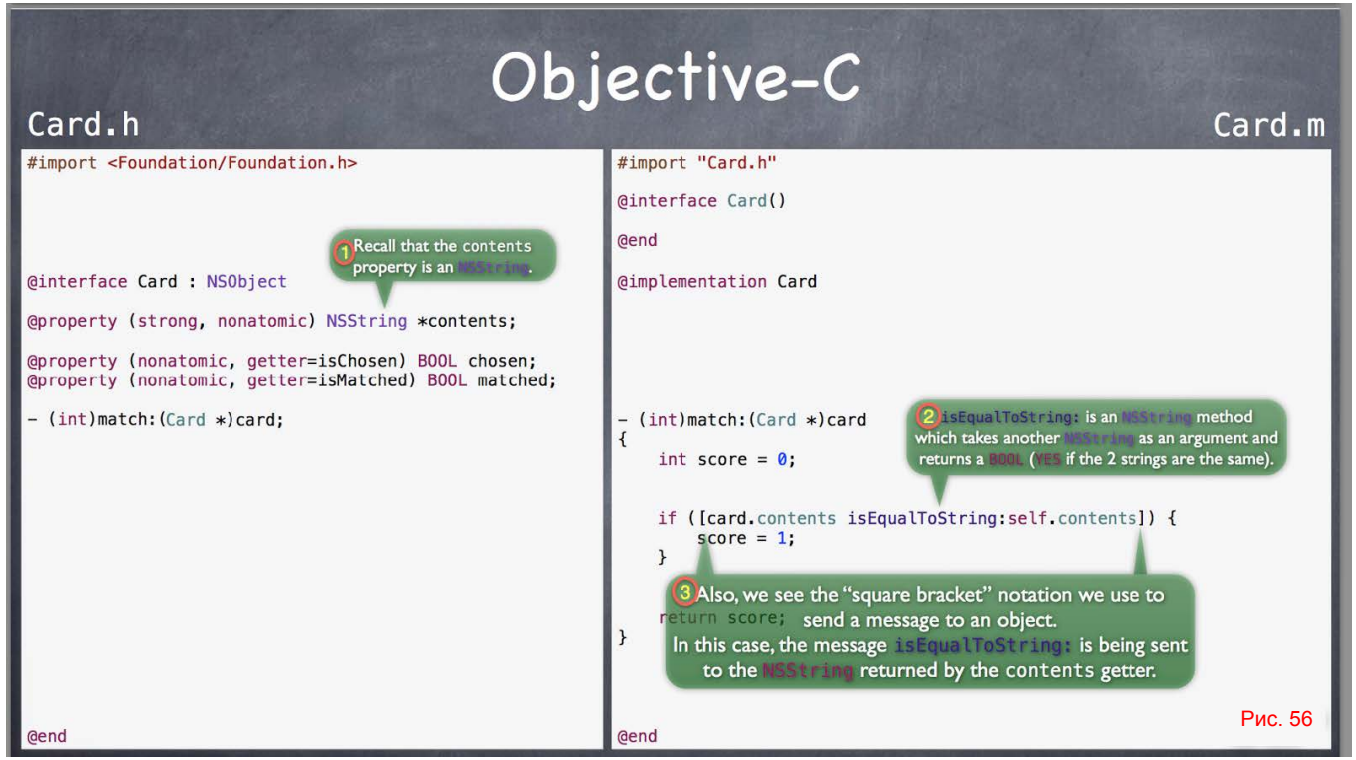

Objective-C
Card.h
Card.m
#import <Foundation/Foundation.h>
@interface Card : NSObject
@property (strong, nonatomic) NSString *contents;
@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;
- (int)match:(Card *)card;
@end
1 Recall that the contents property is an NSString.
#import "Card.h"
@interface Card()
@end
@implementation Card
- (int)match:(Card *)card
{
int score = 0;
if ([card.contents isEqualToString:self.contents]) {
score = 1;
}
return score;
}
@end
2 isEqualToString: is an NSString method which takes another NSString as an argument and returns a BOOL (YES if the 2 strings are the same).
3 Also, we see the "square bracket" notation we use to send a message to an object.
In this case, the message isEqualToString: is being sent to the NSString returned by the contents getter.

Рис. 56

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

- (int)match:(NSArray *)otherCards;



@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card



- (int)match:(NSArray *)otherCards
{
    int score = 0;

    if ([card.contents isEqualToString:self.contents]) {
        score = 1;
    }

    return score;
}

@end
```

We could make match: even more powerful by allowing it to match against multiple cards by passing an array of cards using the NSArray class in Foundation.

Рис. 57

Objective-C

Card.h

```objc
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

- (int)match:(NSArray *)otherCards;









@end
```

Card.m

```objc
#import "Card.h"

@interface Card()

@end

@implementation Card

- (int)match:(NSArray *)otherCards
{
    int score = 0;

    for (Card *card in otherCards) {
        if ([card.contents isEqualToString:self.contents]) {
            score = 1;
        }
    }

    return score;
}



@end
```

1. We'll implement a very simple match scoring system here which is to score 1 point if ANY of the passed otherCards' contents match the receiving Card's contents. (You could imagine giving more points if multiple cards match.)

2. Note the for-in looping syntax here. This is called "fast enumeration." It works on arrays, dictionaries, etc.

Рис. 58