CS193P - Lecture 6

iPhone Application Development

Designing iPhone Applications
Model-View-Controller (Why and How?)
View Controllers

iPhone アプリの設計
MVC（なぜ・どのように）
View Controller

Announcements おしらせ（省略）

- Questions about Views?
- Friday's optional section...
  - Extended Office Hours
  - Gates 360, 3:30 - 5pm

Today's Topics 今日のトピックス

- Designing iPhone Applications iPhone アプリの設計論
- Model-View-Controller (Why and How?) MVC（なぜ・どのように）
- View Controllers View Controller

Designing iPhone Applications

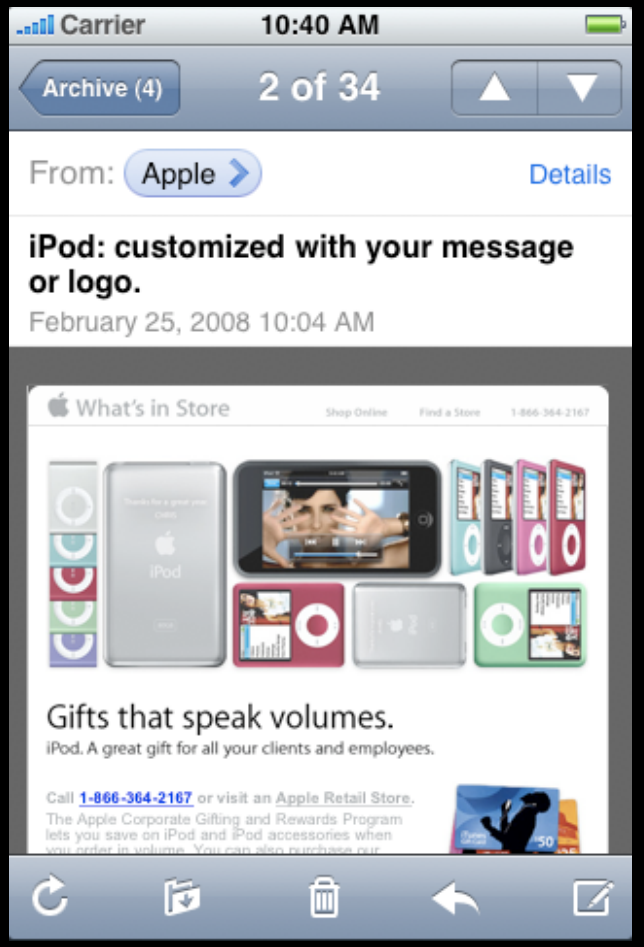
iPhone アプリの設計論

Two Flavors of Mail

メール（リーダ）の２つのフレーバー

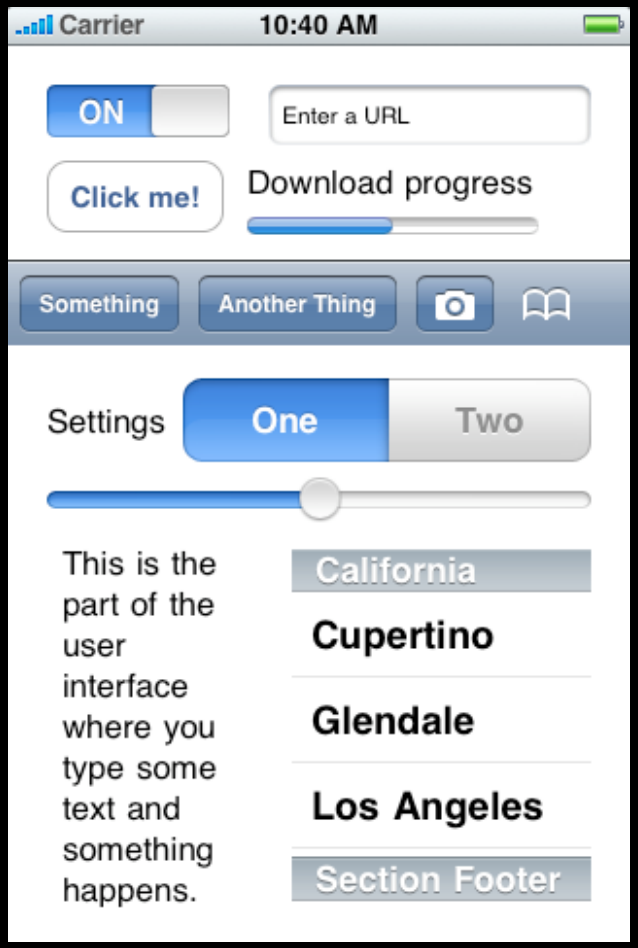
Mac の場合

iPhone の場合



どちらもメールアプリだが作りは「別モノ」

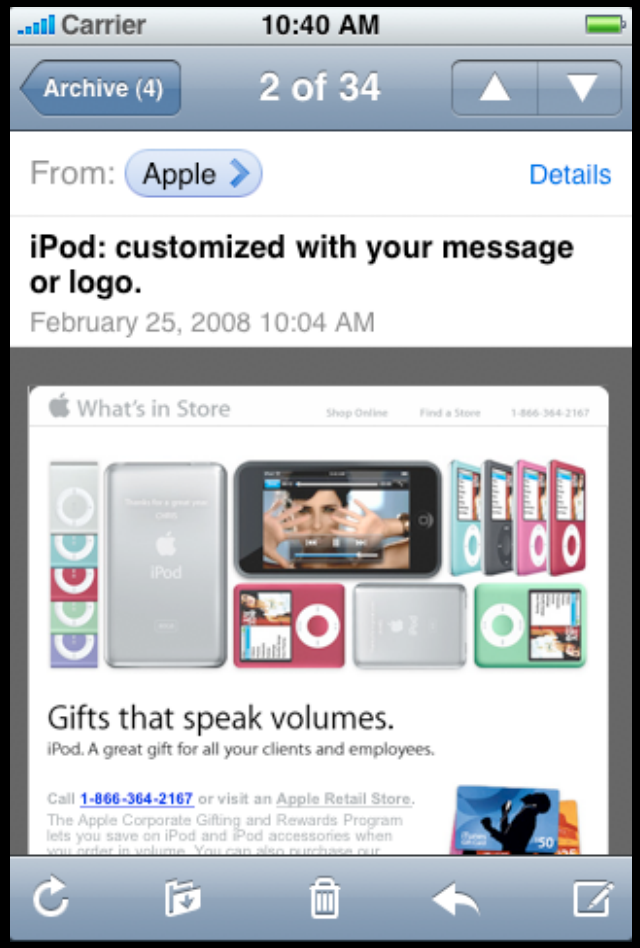
Organizing Content コンテンツを整理すること



ごちゃごちゃ

一緒に使うことが稀な操作アイテムが雑然と並んでいる・・・

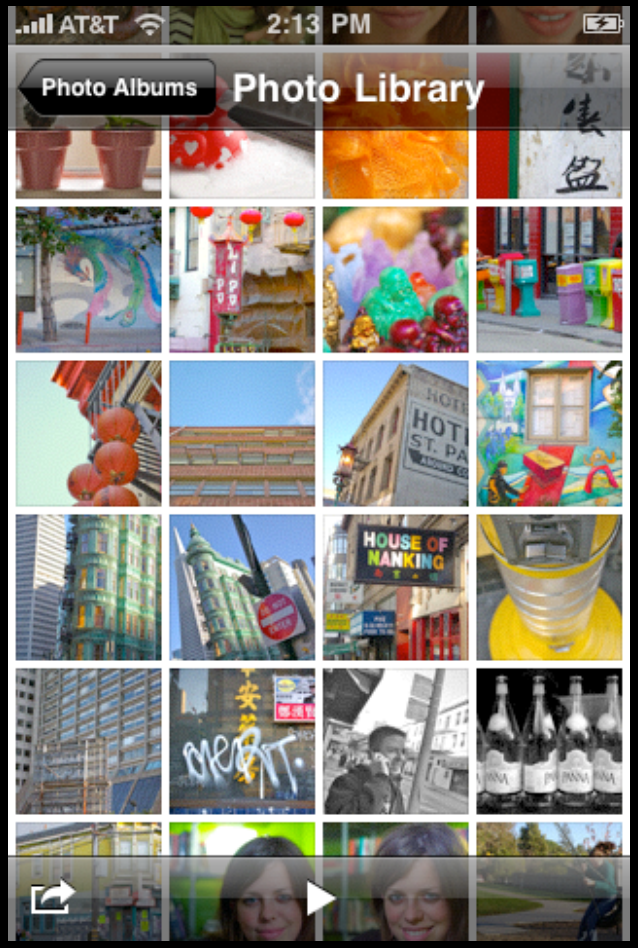
Organizing Content コンテンツを整理すること



わりとスッキリ

メッセージ（送信元・件名・本文）に集中，
他の操作要素は上下に圧縮配置．

Organizing Content コンテンツを整理すること



- Focus on your user's data
- One thing at a time
- Screenfuls of content

- ユーザデータに焦点をあて
- 一度にひとつ
- スクリーンいっぱい
（ナビゲーションバーやツールバー
の裏側まで画像コンテンツを表示）

内容を整理するためのデザインパターン

Patterns for Organizing Content

ナビゲーション バー
Navigation Bar

親　タイトル

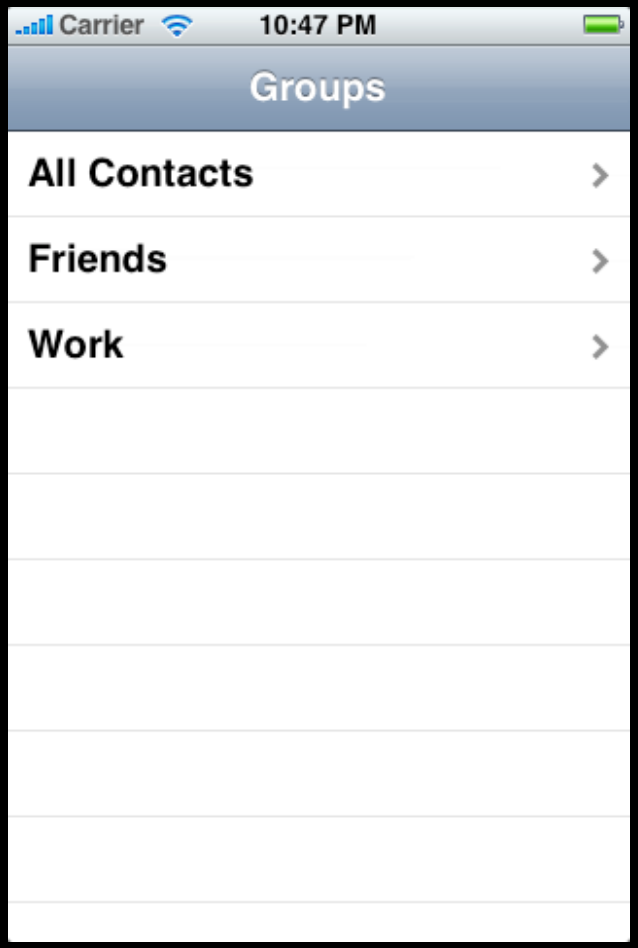
子が
ならぶ

タブ バー
Tab Bar

アプリ機能の切り替え

ナビゲーション バー

Navigation Bar



- Hierarchy of content
- Drill down into greater detail

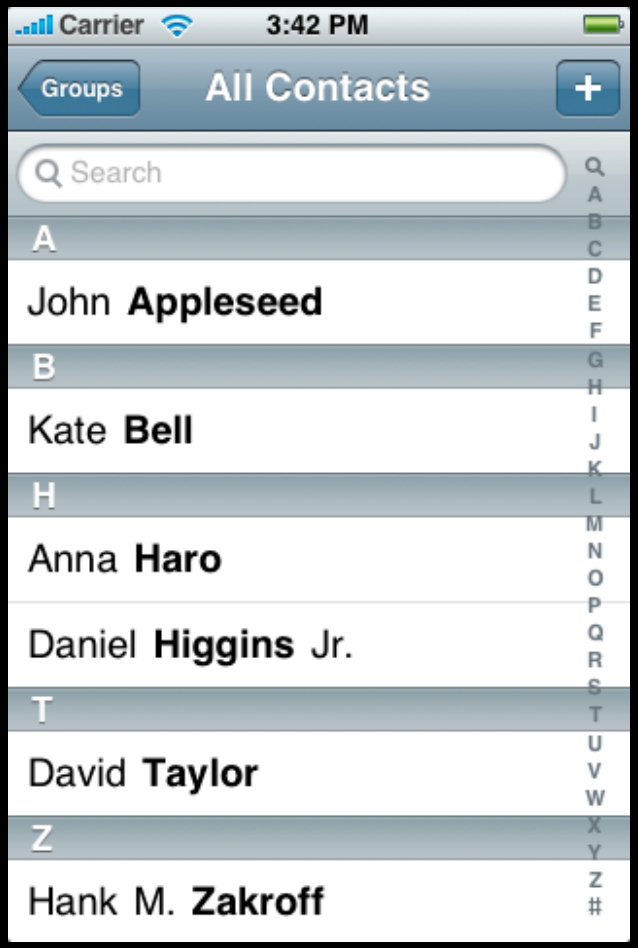
- コンテンツの階層
- より細かな部分に掘り下げていく

Tab Bar タブ バー

- Self-contained modes

- 独立した（複数の）モード
  （「ミニ アプリ」の切り替え）

A Screenful of Content

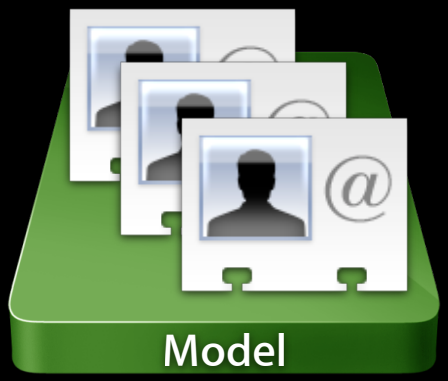


- Slice of your application
- Views, data, logic

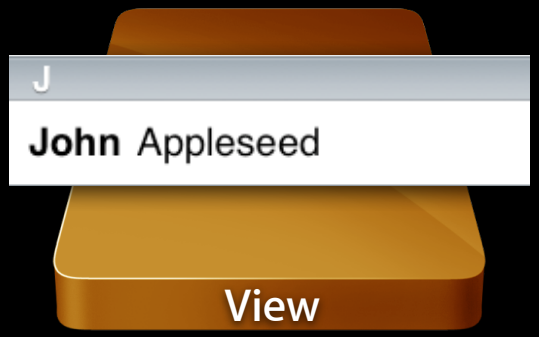
スクリーン１枚分のコンテンツ
- アプリのスライス（薄切り１枚）
- ビュー，データ，ロジック

Parts of a Screenful

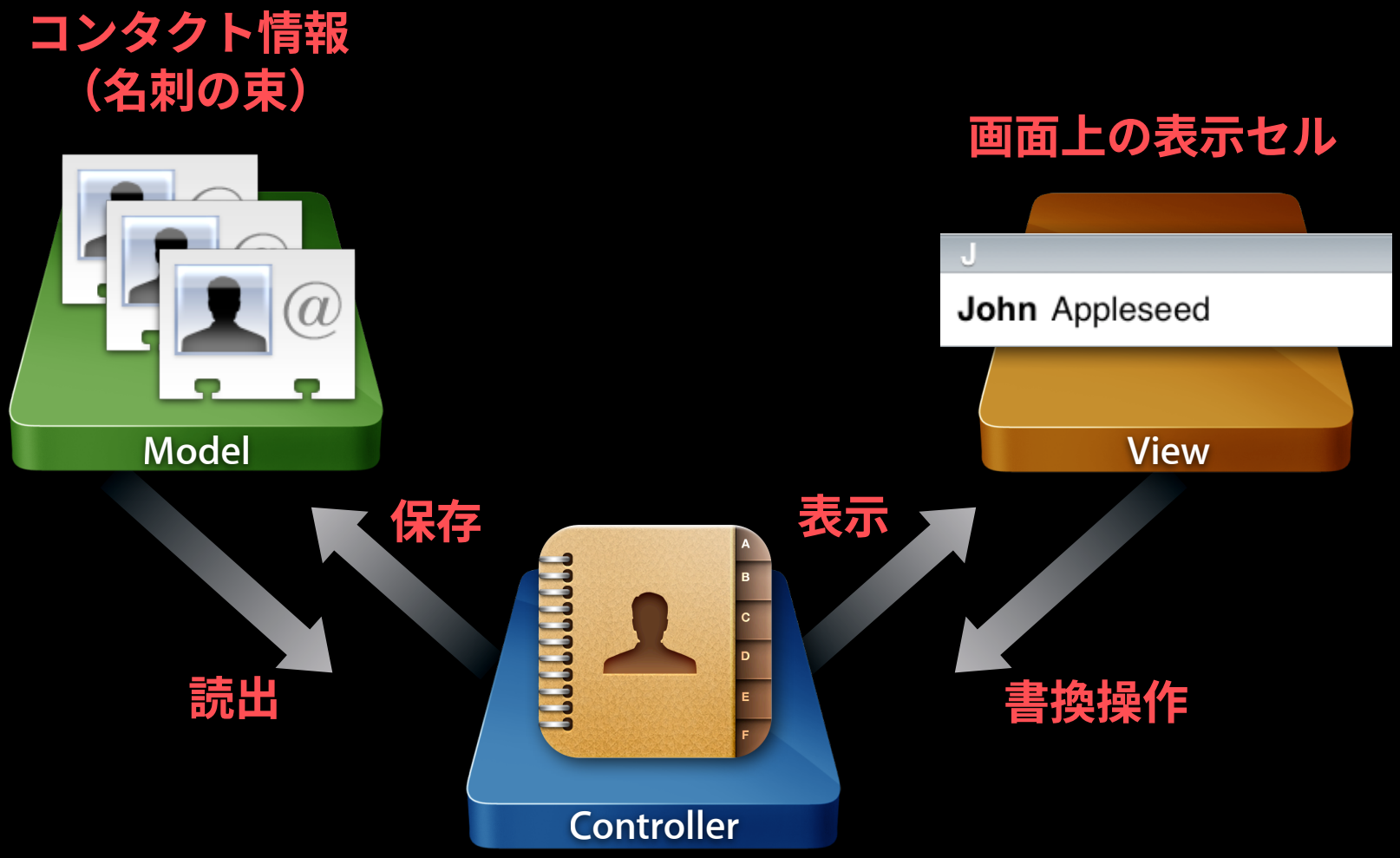
コンタクト情報
（名刺の束）



画面上の表示セル

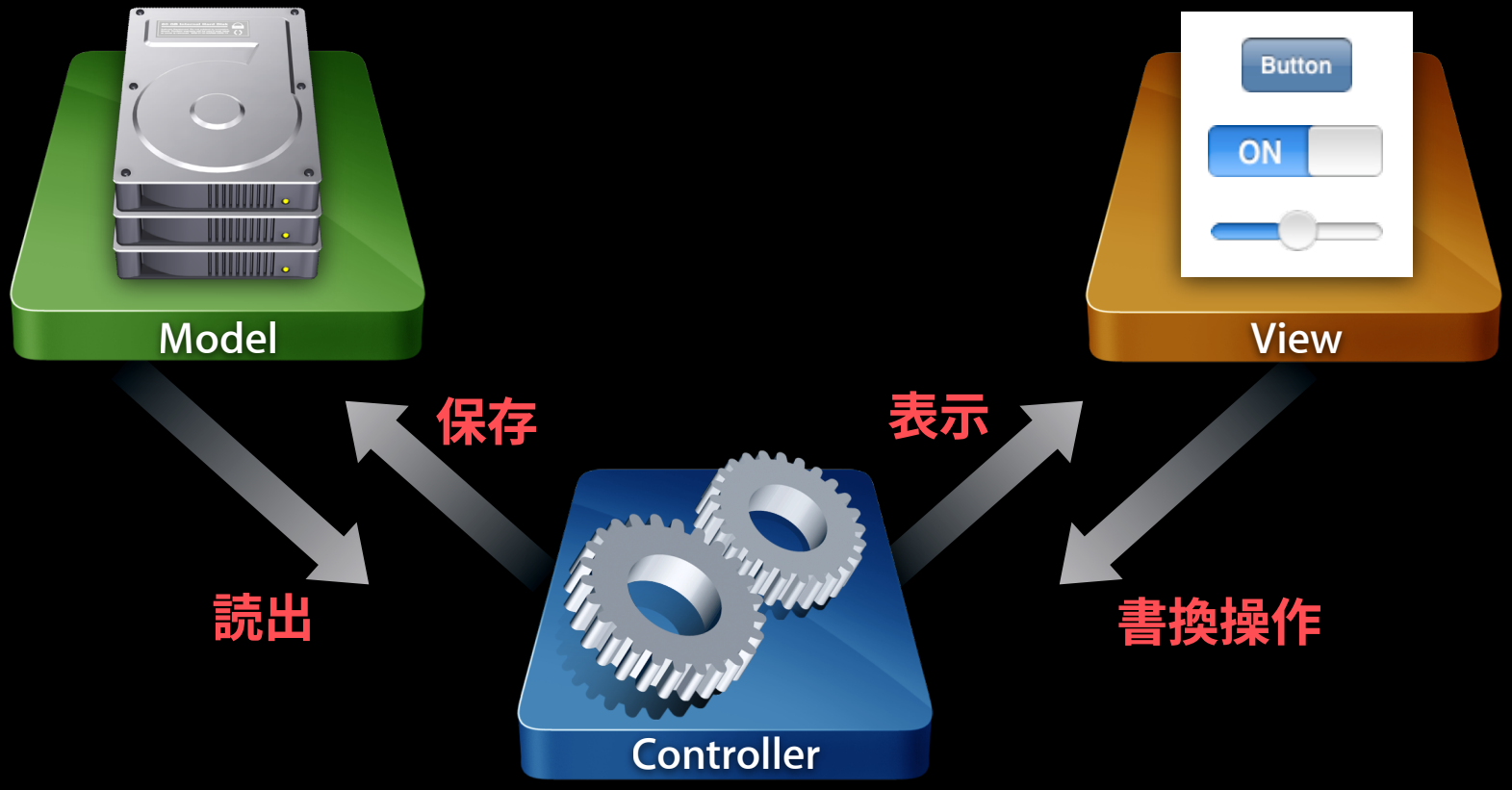


Parts of a Screenful



Parts of a Screenful

一般化すると…



Model-View-Controller MVC (Why and How?)

なぜ MVC を使うのか
どのように MVC でアプリを設計すればよいか

Why Model-View-Controller?

「こんがらがった」

- Ever used the word “spaghetti” to describe code?
- Clear responsibilities make things **easier to maintain**
- Avoid having one monster class that does everything

メンテしやすい
ように責任を
明確にする

１つの
巨大クラスに
すべてを実行させる
のは避けよう

Why Model-View-Controller?

「こんがらがった」

- Ever used the word “spaghetti” to describe code?
- Clear responsibilities make things **easier to maintain**
- Avoid having one monster class that does everything

メンテしやすいように責任を明確にする

１つの巨大クラスにすべてを実行させるのは避けよう

Why Model-View-Controller?

- Separating responsibilites also leads to **reusability**
- By minimizing dependencies, you can take a model or view class you've already written and use it elsewhere
- Think of ways to **write less code**

- 「責任」を切り分けることは，再利用可能性につながる．
- 依存性を最小にすることで，作成した M や V のクラスは ほかの場所でも利用可能になる．
- コード記述量を減らす方法を考えよう．

Communication and MVC

- How should objects communicate?
- Which objects know about one another?

責任分割されたオブジェクトは
どのようにコミュニケーションを
とればよいか

どのオブジェクトたちを
互いに知り合うように
させればよいか

Communication and MVC

- How should objects communicate?
- Which objects know about one another?

Model

裏道場２Ａ

- Example: **Polygon class**
- Not aware of views or controllers
- Typically the **most reusable**
- Communicate generically using...
  - Key-value observing = KVO
  - Notifications

VC のことは考えない

最も再利用しやすい

- M ← C （M の ivar を「監視」する）
- M → C （M から C に「通知」する）

Communication and MVC

- How should objects communicate?
- Which objects know about one another?

View

裏道場３

- Example: **PolygonView class**
- Not aware of controllers, may be aware of relevant model objects

Cのことは考えない

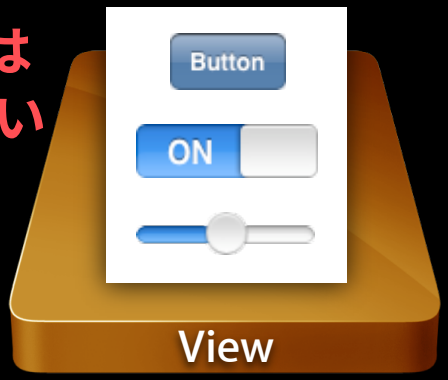
- Also **tends to be reusable**

多少は再利用可

- Communicate with controller using...
  - Target-action
  - Delegation

ターゲット=アクション

デリゲーション（委譲）



Communication and MVC

- How should objects communicate?
- Which objects know about one another?

Controller

MV のことを知っている

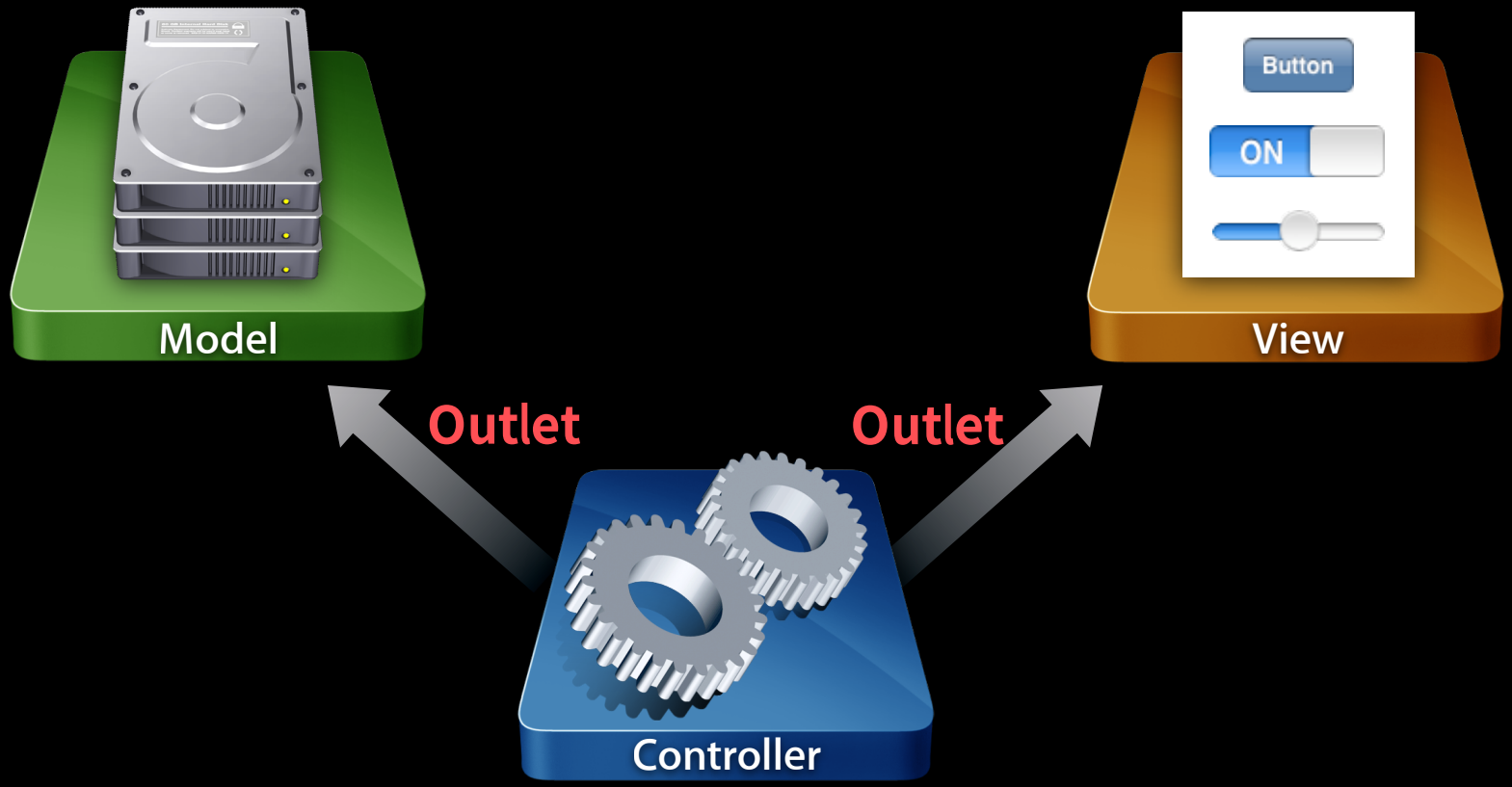
- Knows about model and view objects
- The brains of the operation 頭脳
- Manages relationships and data flow
- Typically app-specific,
  so **rarely reusable**

データの
流れを制御

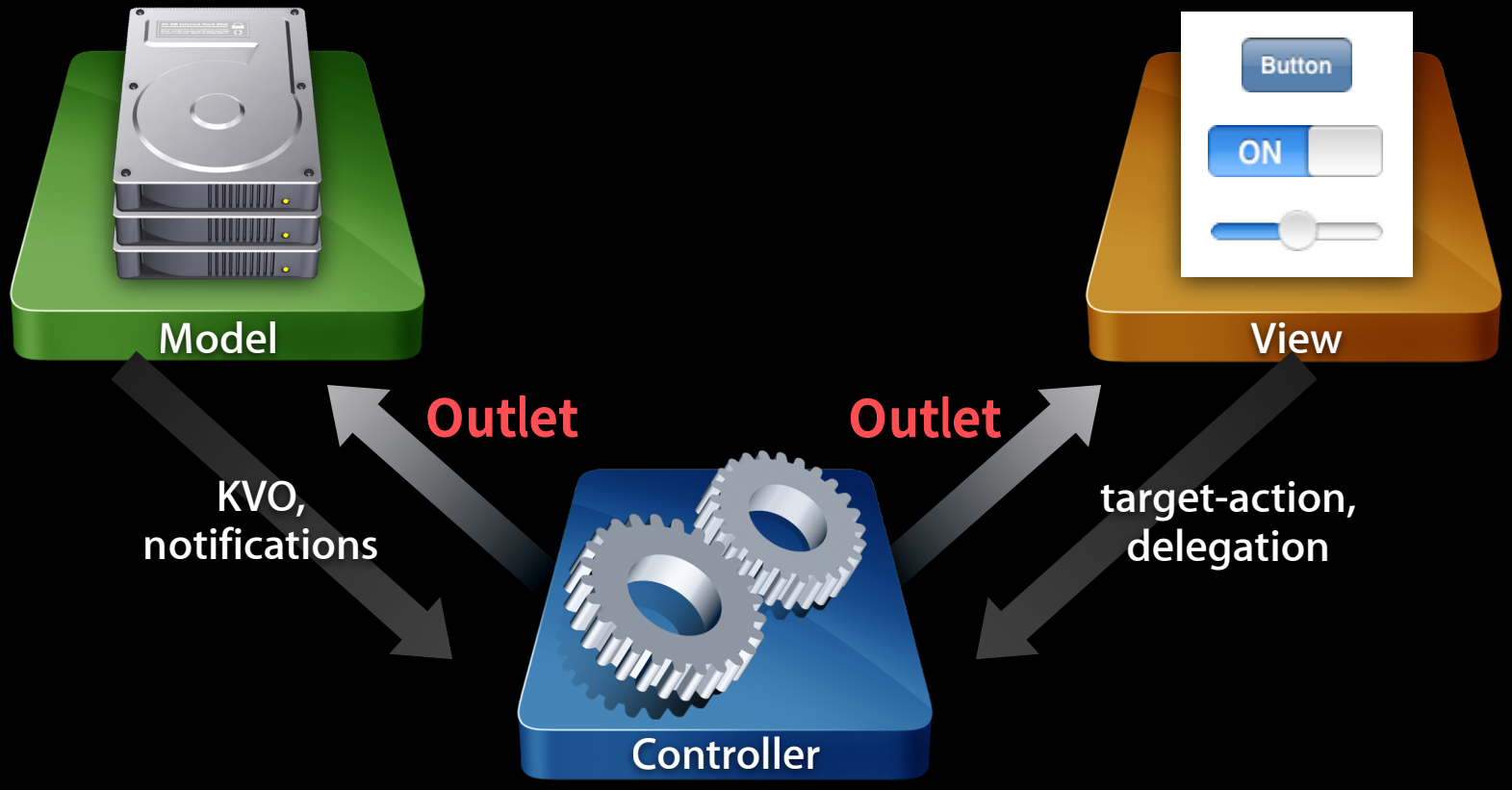
アプリごとに特殊化され，再利用は稀．

Communication and MVC

Communication and MVC



Communication and MVC



View Controllers UIViewController

スクリーン１枚分のマネジメント

Problem: Managing a Screenful

コントローラは１枚分の画面・データ・処理ロジックを管理

- Controller manages views, data and application logic
- Apps are made up of many of these　アプリはコントローラの集合体
- Would be nice to have a well-defined starting point
  - A la UIView for views
  - Common language for talking about controllers

View として UIView が標準的な出発点となるように
Controller には UIViewController が使える

Problem: Building Typical Apps

- Some application flows are very common　よくあるアプリのフロー
  - Navigation-based
  - Tab bar-based
  - Combine the two
- Don’t reinvent the wheel　「車輪を再び発明する」のは愚か
- Plug individual screens together to build an app

個々のスクリーンをつなげて
ひとつのアプリをつくる

UIViewController

- Basic building block　View Controller の基本的な構成要素
- Manages a screenful of content　スクリーン１枚分のコントロール
- Subclass to add your application logic

UIViewController

- Basic building block　View Controller の基本的な構成要素
- Manages a screenful of content　スクリーン１枚分のコントロール
- Subclass to add your application logic

アプリごとの処理ロジックを与えるには
UIViewController クラスからサブクラスをつくる

UINavigationController と
UITabBarController については
それらからサブクラスをつくらずに
直接使うことが多い

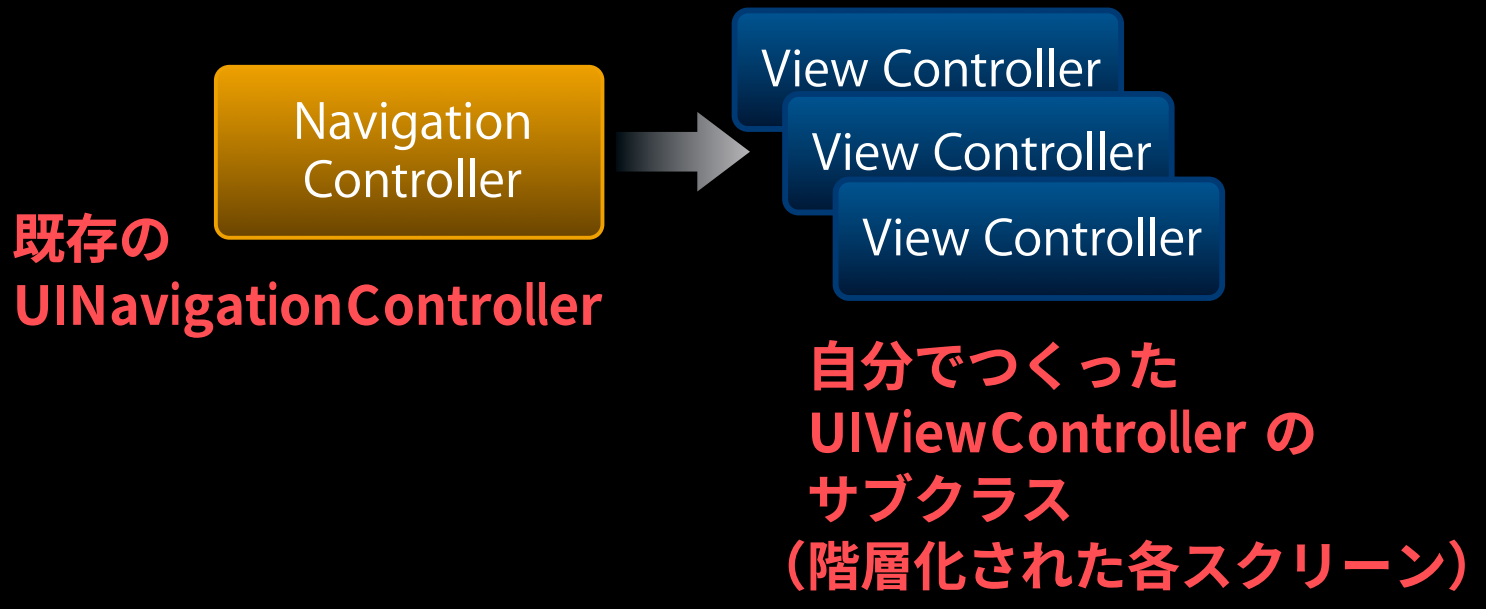
“Your” and “Our” View Controllers

- **Create your own** UIViewController subclass for each screenful
- Plug them together using existing **composite** view controllers

- スクリーン１枚分ごとに UIViewConroller のサブクラスをつくる
- それらを既存の複合 View Controller（Navigation Controller など）に接続する.

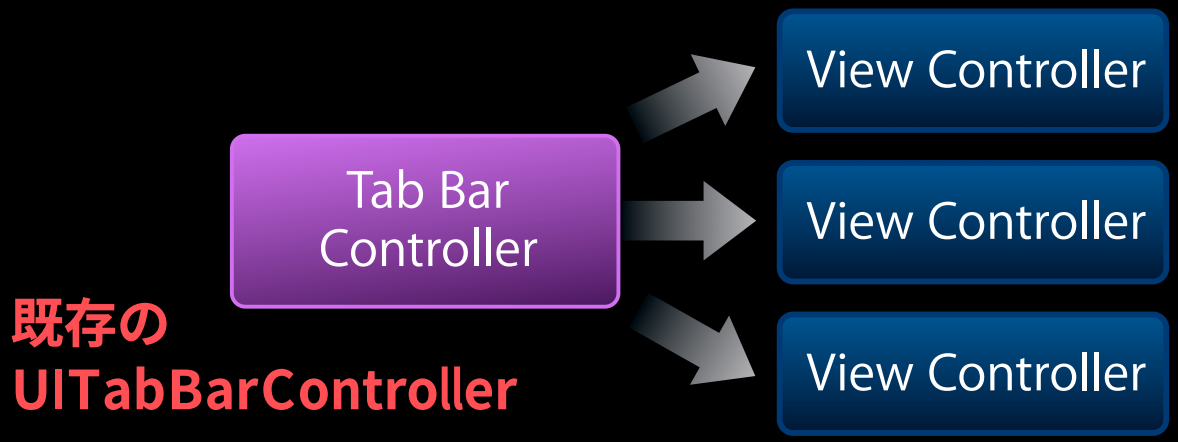
“Your” and “Our” View Controllers

- **Create your own** UIViewController subclass for each screenful
- Plug them together using existing **composite** view controllers



“Your” and “Our” View Controllers

- **Create your own** UIViewController subclass for each screenful
- Plug them together using existing **composite** view controllers



既存の
UITabBarController

自分でつくった
UIViewController の
サブクラス
（切替可能な各スクリーン）

UIViewController のサブクラスの作り方

Your View Controller Subclass

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

@interface MyViewController : UIViewController {
  // A view controller will usually
  // manage views and data
  NSMutableArray *myData;
  UILabel *myLabel;
}

// Expose some of its contents to clients
@property (readonly) NSArray *myData;

// And respond to actions
- (void)doSomeAction:(id)sender;
```

自分のサブクラス名　　親クラス

M へのポインタ　（IB を使う場合は
V へのポインタ　　IBOutlet を追加）

セッタ・ゲッタの宣言

メソッドの宣言
（IB を使う場合は
(IBAction) とする）

The “View” in “View Controller”

- UIViewController superclass has a view property
  - `@property (retain) UIView *view;`
- Loads lazily
  - On demand when requested
  - Can be purged on demand as well (low memory)
- Sizing and positioning the view?
  - Depends on where it’s being used
  - Don’t make assumptions, be flexible

UIViewController は
UIView へのポインタをもつ

View は必要に応じて自動ロード

必要に応じて
メモリから自動解放

重力加速度の方向
ステータスバーの有無 → これらに柔軟に対応できるように
タブバーの有無

When to call -loadView?

- **Don't do it!** 直接 -loadView を呼び出さないこと
- Cocoa tends to embrace a lazy philosophy 「怠け者の哲学」
  - Call -release instead of -dealloc
  - Call -setNeedsDisplay instead of -drawRect:
- Allows work to be deferred or coalesced 仕事を遅らせる 別の仕事と一緒にする
  - Performance!

Creating Your View in Code

- Override -loadView
  - Never call this directly
- Create your views
- Set the view property
- Create view controller with -init

1．View を生成

2．View を Controller に登録

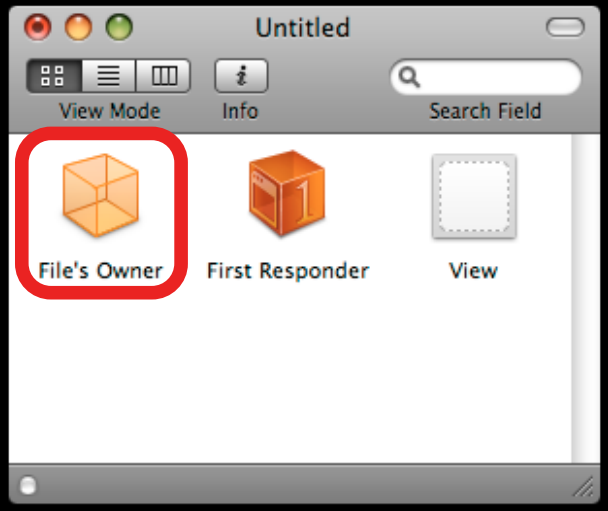
-initWithNibName

```
// Subclass of UIViewController
- (void)loadView
{
1 MyView *myView = [[MyView alloc] initWithFrame:frame];
2 self.view = myView; // The view controller now owns the view
  [myView release];
}
```

IB 上で View と View Controller をつくるには

Creating Your View with Interface Builder

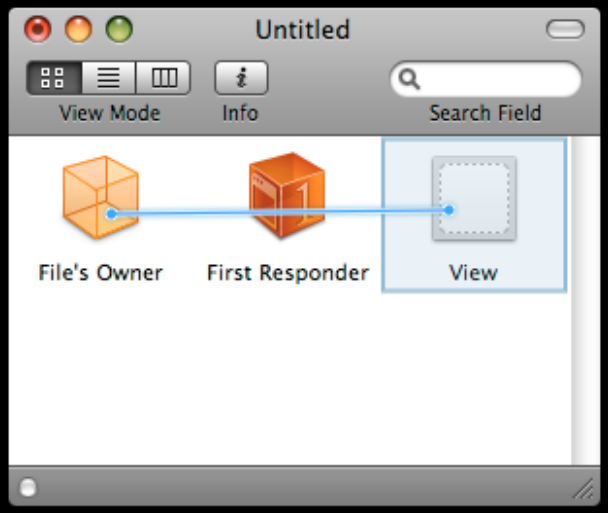
- Lay out a view in Interface Builder
  IB 上で View を配置
- File's owner is view controller class
  File's Owner とは View Controller クラス



IB 上で View と View Controller をつくるには

Creating Your View with Interface Builder

- Lay out a view in Interface Builder
  IB 上で View を配置
- File's owner is view controller class
  File's Owner とは View Controller のこと
- Hook up view outlet
  アウトレットを接続
- Create view controller
  with -initWithNibName:bundle:
  View Controller を -initWithNibName で生成



Demo:
View Controllers with IB

デモ： IB による View Controller のつくりかた

View Controller Lifecycle

```
- (id)initWithNibName:(NSString *)nibName
bundle:(NSBundle *)bundle
{
  if (self ✗= [super init...]) {
    // Perform initial setup, nothing view-related
    myData = [[NSMutableArray alloc] init];
    self.title = @“Foo”;
  }
  return self;
}
```

title は NavigationController や TabBarController で効いてくる

nib/xib ファイルを指定して
View Controller を新しく生成する
（普段はコメントアウトされている）

View Controller Lifecycle

```
- (void)viewDidLoad
{
    // Your view has been loaded
    // Customize it here if needed
    view.someWeirdProperty = YES;
}
```

nib/xib ファイルから
View が生成された直後に
呼び出されるメソッド
（初期化に便利）

（これも普段はコメントアウトされている）

View Controller Lifecycle

```
- (void)viewWillAppear:(BOOL)animated
{
  [super viewWillAppear:animated];

  // Your view is about to show on the screen
  [self beginLoadingDataFromTheWeb];
  [self startShowingLoadingProgress];
}
```

たとえば NavigationController では
新しい View が左右いずれかからスライドインする．
このとき，View が現われ始める直前に，
このメソッド viewWillAppear が呼び出される．
（これも初期化に便利）

出現が完了したら viewDidAppear が呼び出される．

View Controller Lifecycle

```
- (void)viewWillDisappear:(BOOL)animated
{
  [super viewWillDisappear:animated];

  // Your view is about to leave the screen
  [self rememberScrollPosition];
  [self saveDataToDisk];
}
```

たとえばスライドアウトするなど，
View が消滅しようとする直前に
viewWilldisappear が呼び出される．

同様に，消滅が完了したら，
viewDidDisappear が呼び出される．

Loading & Saving Data

View の出入りに合わせて
データもロード/セーブする
ことが多い

- Lots of options out there, depends on what you need
  - NSUserDefaults 【DEMO】
  - Property lists
  - CoreData
  - SQLite
  - Web services
- Covering in greater depth in Lecture 9 on 4/29

Demo:
Loading & Saving Data

iPhone Simulator 4.0 でのデモ

More View Controller Hooks

ほかにもある
View Controller が
受け取るメソッド

- Automatically rotating your user interface
- Low memory warnings

インタフェースを自動回転させる

メモリ不足の警告

インタフェースを回転させるには

Supporting Interface Rotation

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
       (UIInterfaceOrientation)interfaceOrientation
{
  // This view controller only supports portrait
  return (interfaceOrientation ==
          UIInterfaceOrientationPortrait);
}
```

正立のみ Yes（サポート）

インタフェースを回転させるには

Supporting Interface Rotation

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
       (UIInterfaceOrientation)interfaceOrientation
{
  // This view controller supports all orientations
  // except for upside-down.
  return (interfaceOrientation !=
          UIInterfaceOrientationPortraitUpsideDown);
}
```

逆さま以外は Yes（サポート）

Demo:
Rotating Your Interface

View の自動リサイズ

Autoresizing Your Views

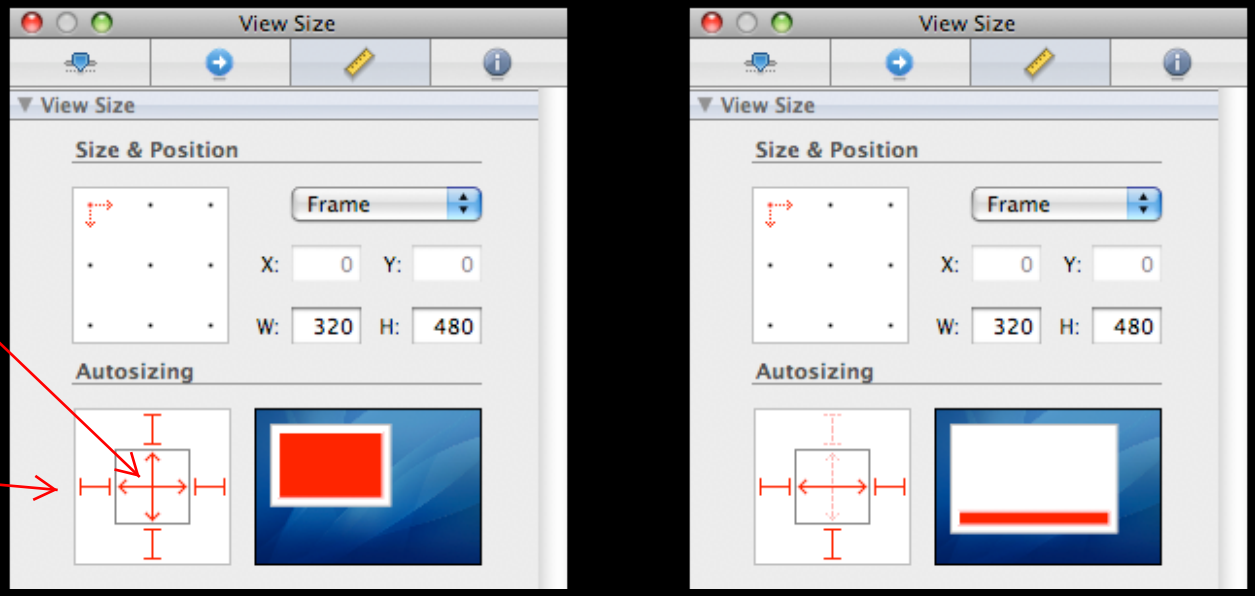
プログラム内で指定する場合

```
view.autoresizingMask = UIViewAutoresizingFlexibleWidth |
                        UIViewAutoresizingFlexibleHeight;

view.autoresizingMask = UIViewAutoresizingFlexibleWidth |
                        UIViewAutoresizingFlexibleTopMargin;
```

伸縮可能の指定

固定マージンの指定



IB の「モノサシ」項目で調整する場合

Questions?