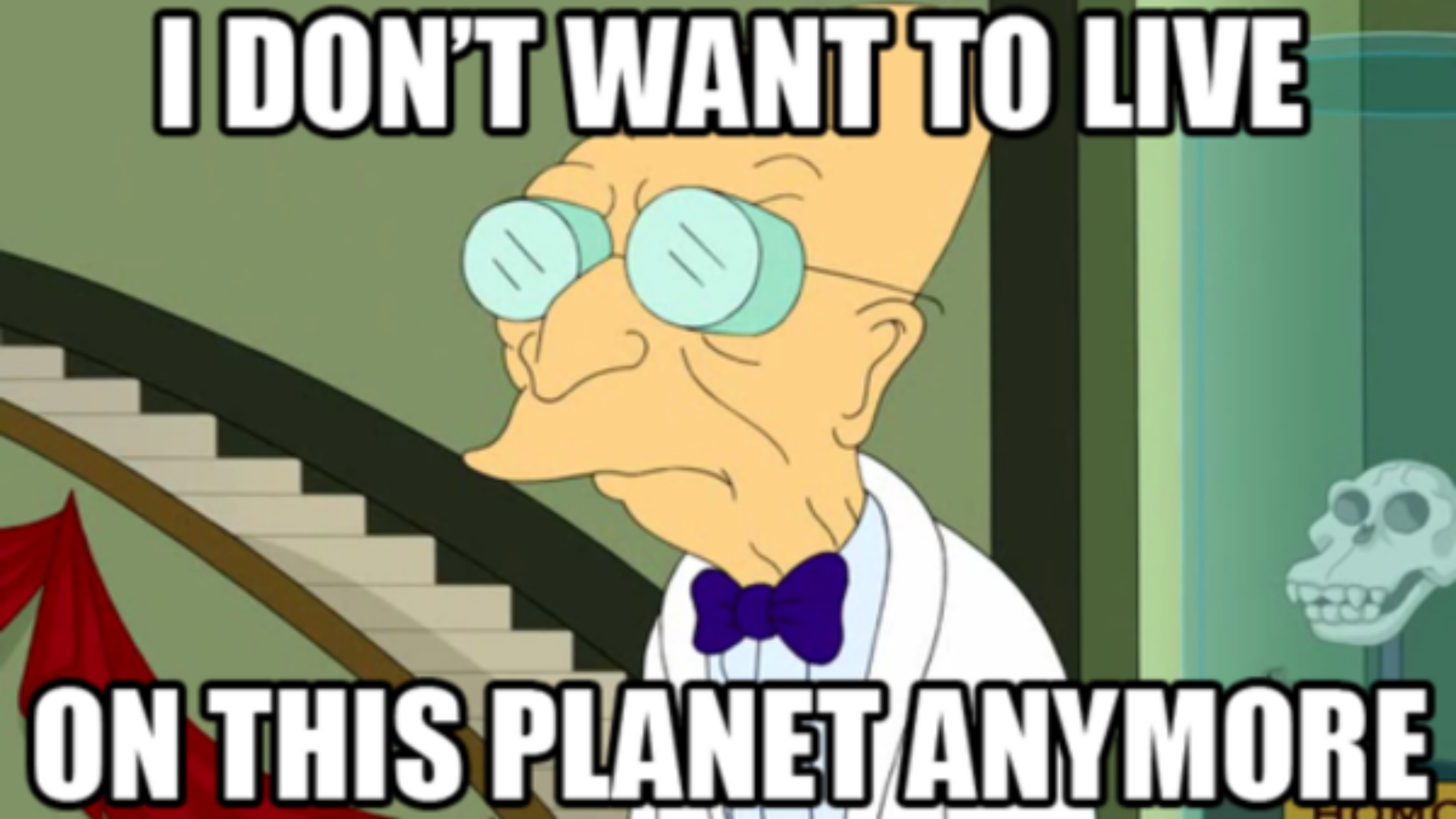
A Better MVC

“You’re holding it wrong.”

Dave DeLong – @davedelong
A guy who thinks too deeply about stuff

Massive View Controller


I DON'T WANT TO LIVE
ON THIS PLANET ANYMORE

(This is our own fault)

“I really think we should follow bad programming principles, violate encapsulation, and make things tightly coupled.”

–Literally no one, ever

Why do we make this mistake?

- Apple “tells us” to
- Revert to the default
- External constraints

“…when experience is not retained, … infancy is perpetual. Those who cannot remember the past are condemned to repeat it.”

–George Santayana

Making some observations

Huh, that’s interesting…

#1: MVC is not a Pattern

- MVC is a philosophy
- Separate storage from networking from parsing from routing from business logic from rendering from view hierarchy from …
- “Who should care about this logic?”

#2: Patterns are Tools

- Similar problems → similar solutions → similar patterns
- Different problems → different solutions → different patterns
- Good developers learn patterns
- Great developers learn *problems*

#3: Naming is Hard

- Just because it has "Controller" in the name, doesn't mean it's a Controller
- `UIViewController` is not a Controller
- It performs view-related things

#4: Views don’t fill the screen

- Your screen of app UI is not a single `UIView`
- Why is your screen of app UI a single `UIViewController`?
- `UIViewControllers` don’t have to fill the screen

Thinking this through

That’s nice; so what?

#1: Decompose `UIViewControllers`

- Decomposition is the fundamental skill of programming
- Break apart your `UIViewControllers`
- A `UIViewController` that …
  - Only shows an image?
  - Only shows a single horizontal line?
  - Is a cell in a `UITableView` or `UICollectionView`?

#2: Compose UI

- Build your UI by composing `UIViewControllers`
- `UIViewController.addChild(_:) // added in iOS 5`

#3: Reuse `UIViewControllers`

- You don’t rewrite `UILabel` every time you need to show text
- Expect to use build and re-use `UIViewControllers`
  - ContainerViewController
  - StackViewController
  - ScrollingContentViewController
- Accidental consistency

#4: Forget `UIView`

- You'll rarely subclass `UIView`
- `UIView` is for rendering or interaction
  - Render UI with `UILabel` and `UIImageView`
  - Handle interaction with `UIGestureRecognizer`
- Consider composing complex views in a `UIViewController`

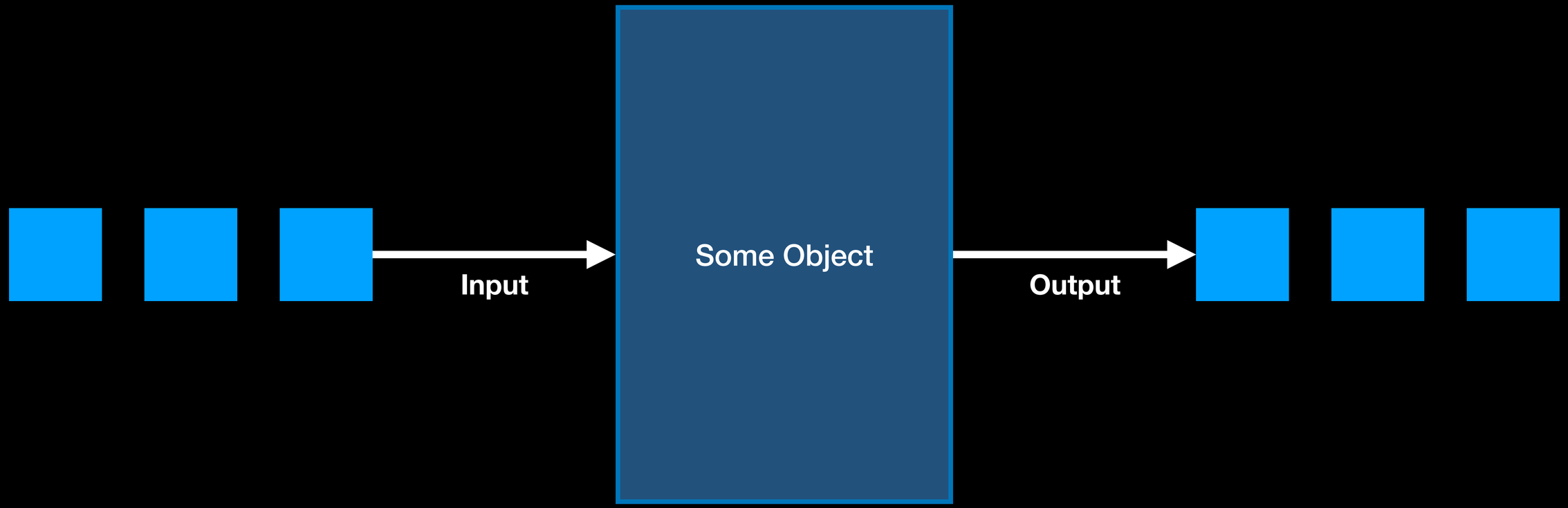
Putting this in to practice

Your ideas intrigue me and I wish to subscribe to your newsletter

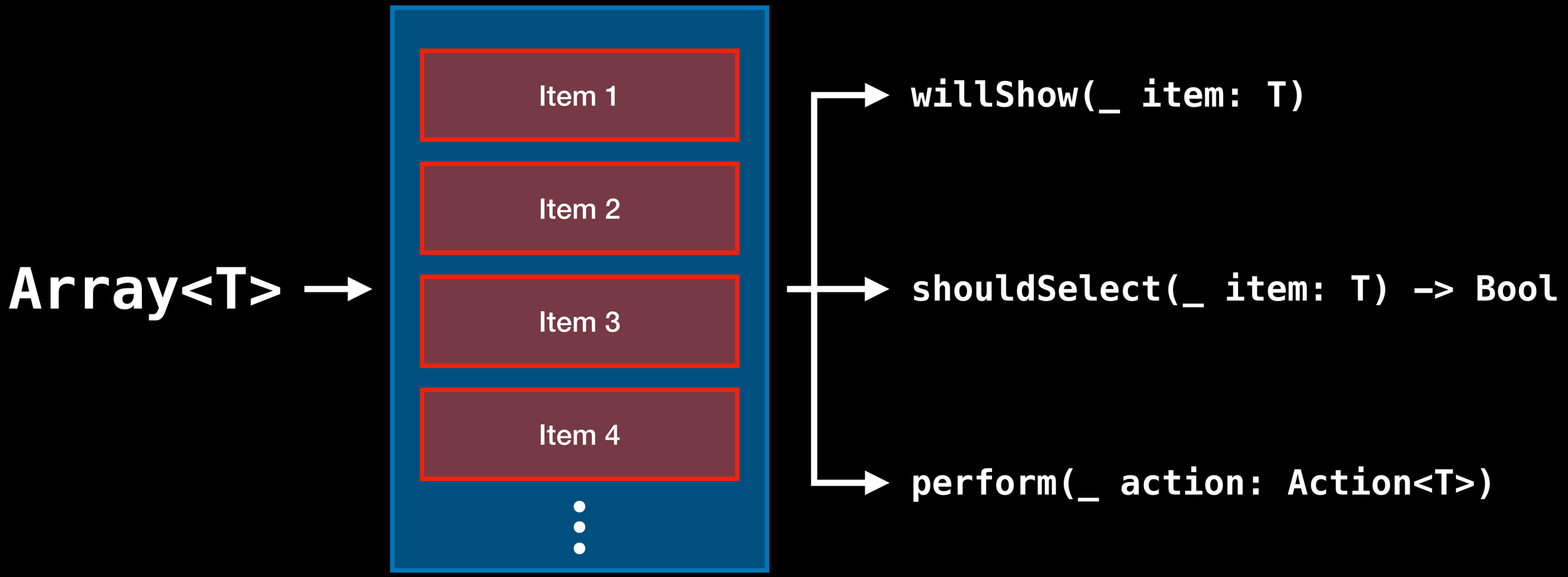
#1: Show Data or Children

- Generally, `UIViews` either compose or render
  - Rendering happens with `UILabel` and `UIImageView`
  - Everything else composes those
- Aim for the same with `UIViewControllers`

#2: Think with generic functions



ListViewController<T>



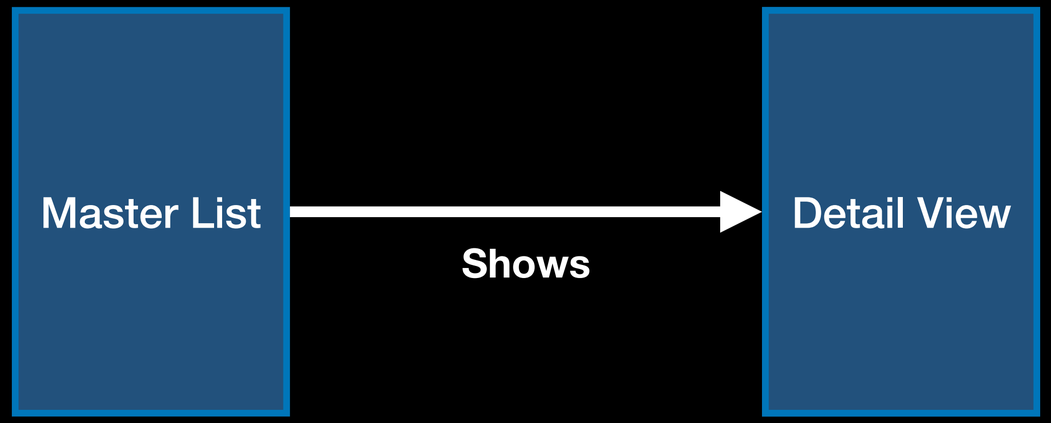
#2: Think with generic functions

- `(T) → T` and `(T) → Void`
- “What goes in, must come out”
  - Input: datasource, parameters, signal/observable…
  - Output: delegate, callbacks, signal/observable…
- Violating this violates encapsulation

#3: Prefer XIBs over Storyboards

- Segues are Problematic™
  - Navigation violates the “`(T) → T`” constraint
  - They break reusability
  - `-prepareForSegue:` is called on the wrong object
- XIBs have a 1-to-1 relation between `UIViewController` and UI
- Can still use outlets and custom initializers

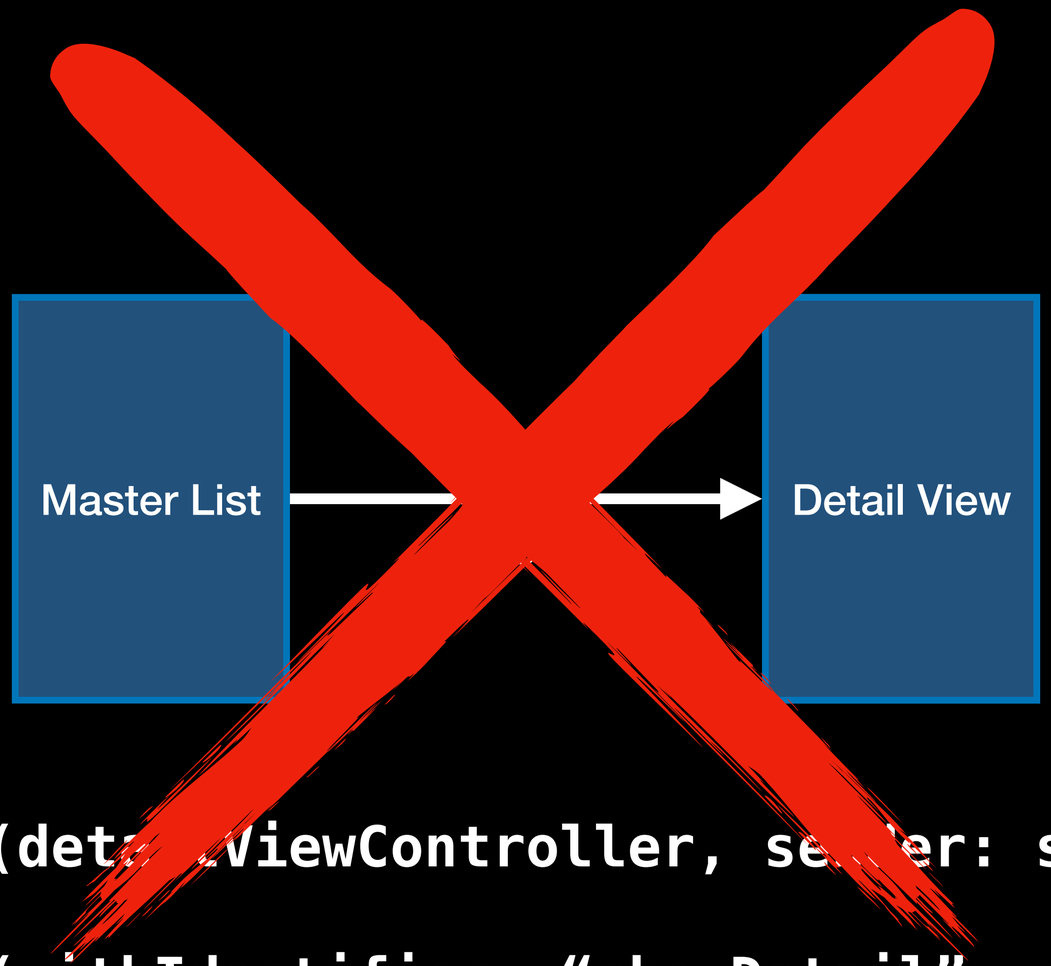
#3: Prefer XIBs over Storyboards




```
show(detailViewController, sender: self)

performSegue(withIdentifier: “showDetail”, sender: self)
```

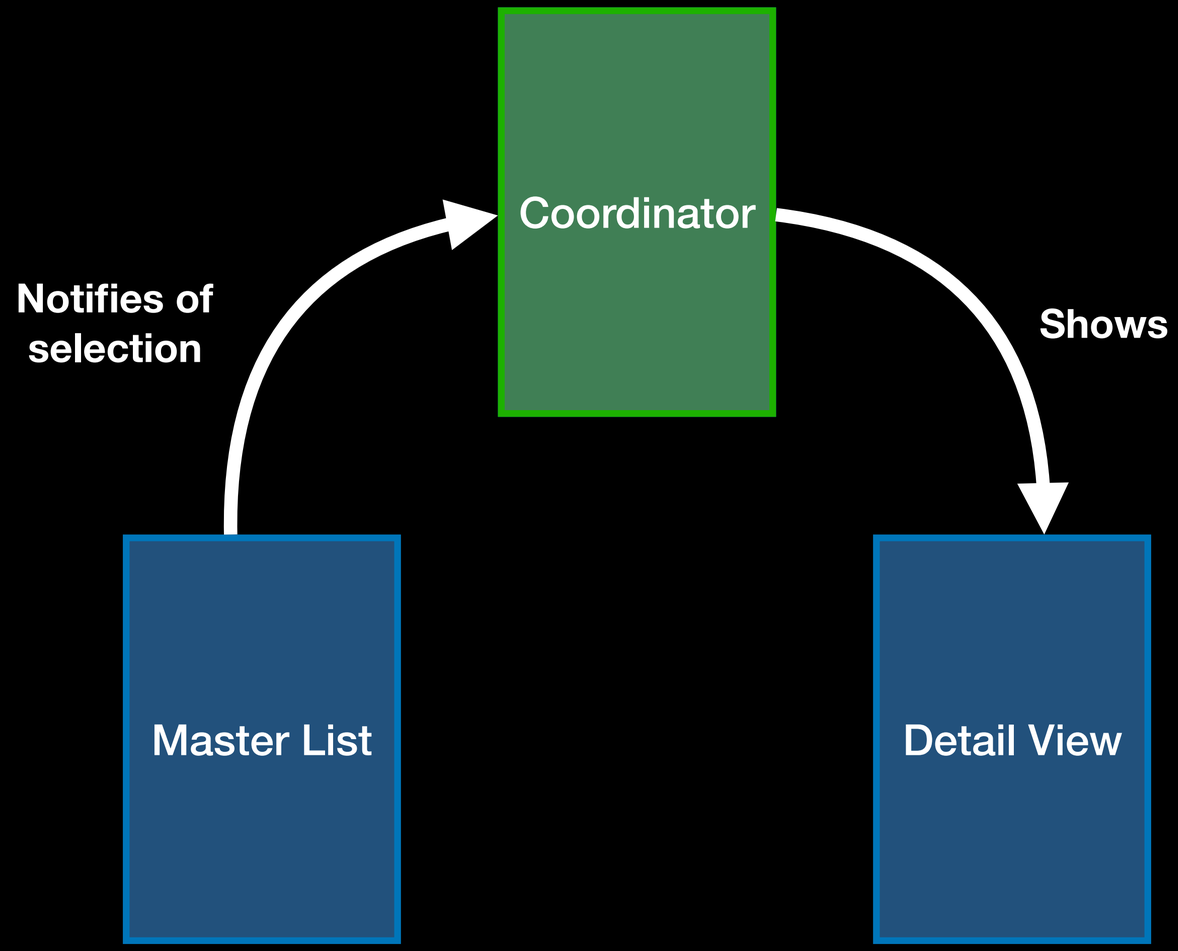
#3: Prefer XIBs over Storyboards




```
show(detailViewController, sender: self)

performSegue(withIdentifier: “showDetail”, sender: self)
```

#3: Prefer XIBs over Storyboards



#4: Adopt this piecemeal

- Don’t rewrite; refactor!
- Refactor to composition as able

Results

- Tiny `UIViewControllers` (usually under 200 lines)
- *Rarely* subclass `UIView`
- Extremely reusable `UIViewControllers`
- Clear separation of concern
- Happy programmers 😊

Thanks!

social media

me@davedelong.com

website

email