# Design Patterns

in

# Ruby

@russolsen

FORTRAN

Pascal

BASIC

C

C++

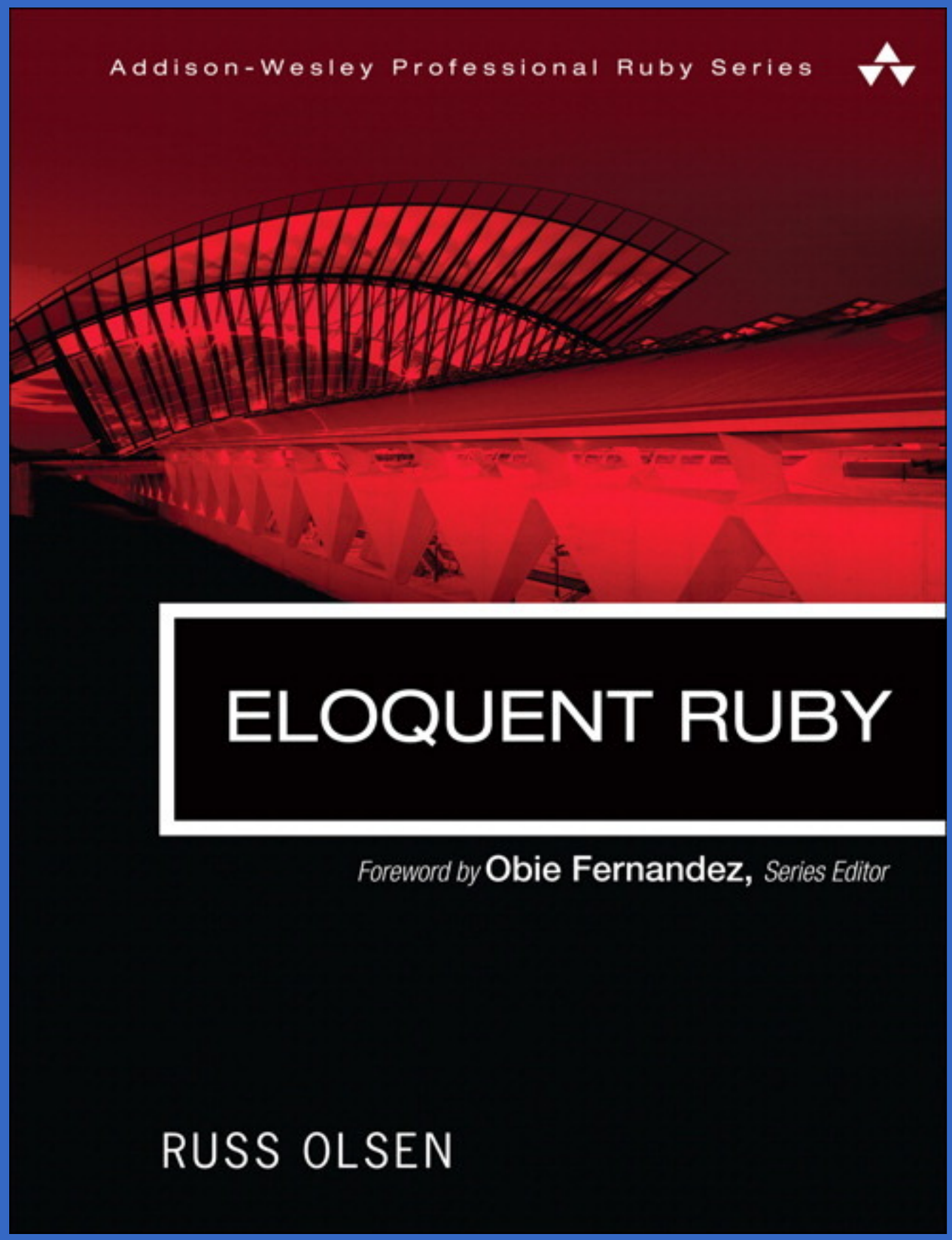
Python

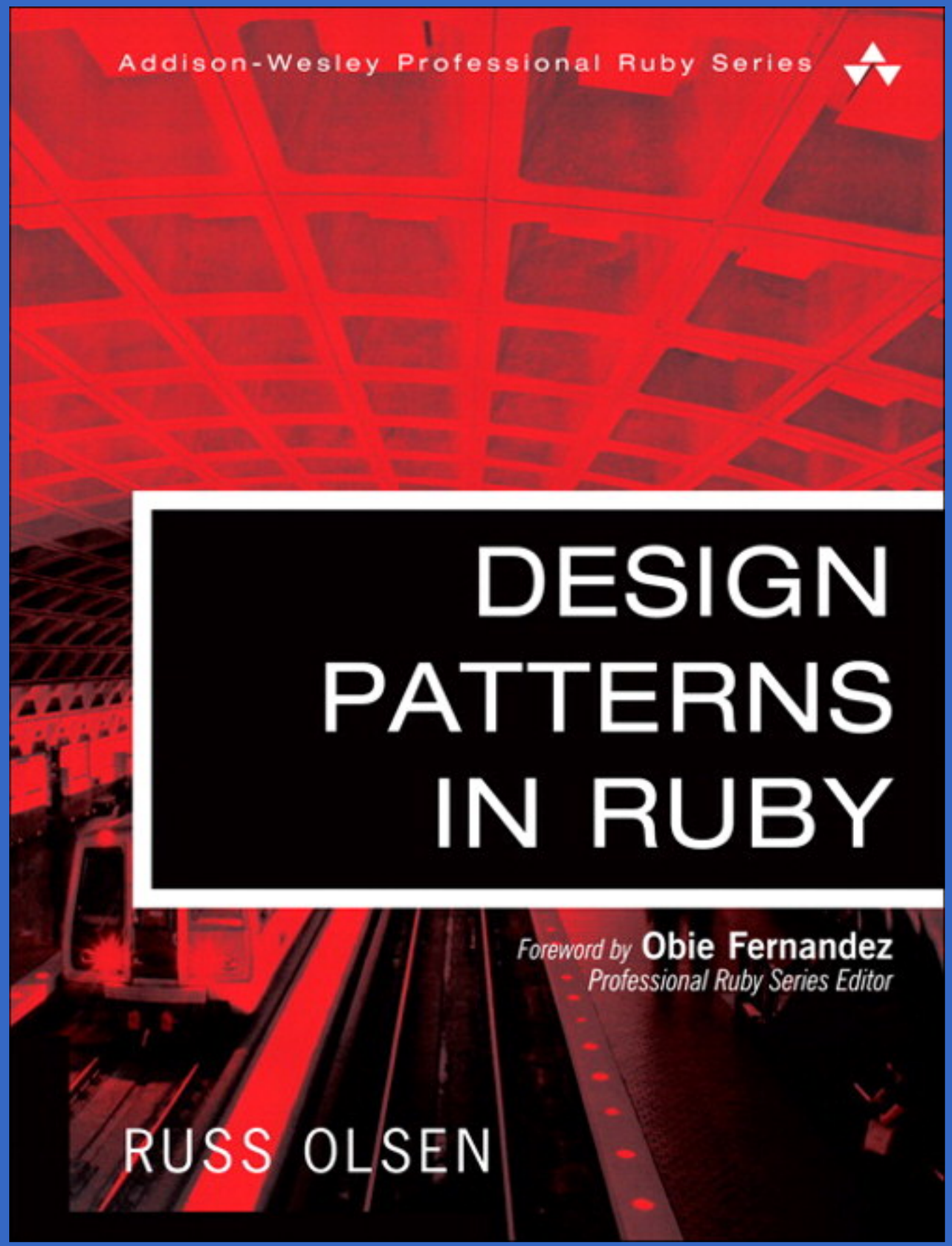
Java

Ruby

Clojure

Go


Addison-Wesley Professional Ruby Series
ELOQUENT RUBY
Foreword by Obie Fernandez, Series Editor
RUSS OLSEN


Addison-Wesley Professional Ruby Series
DESIGN PATTERNS IN RUBY
Foreword by Obie Fernandez
Professional Ruby Series Editor
RUSS OLSEN

# Design Patterns

What are they?

Some Examples

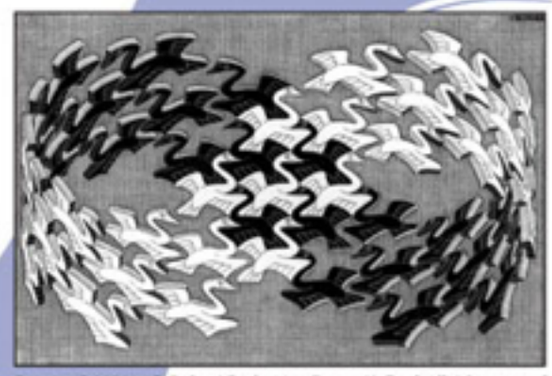
The Good & Bad

# What is a Design Pattern?

# Design Patterns

## Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



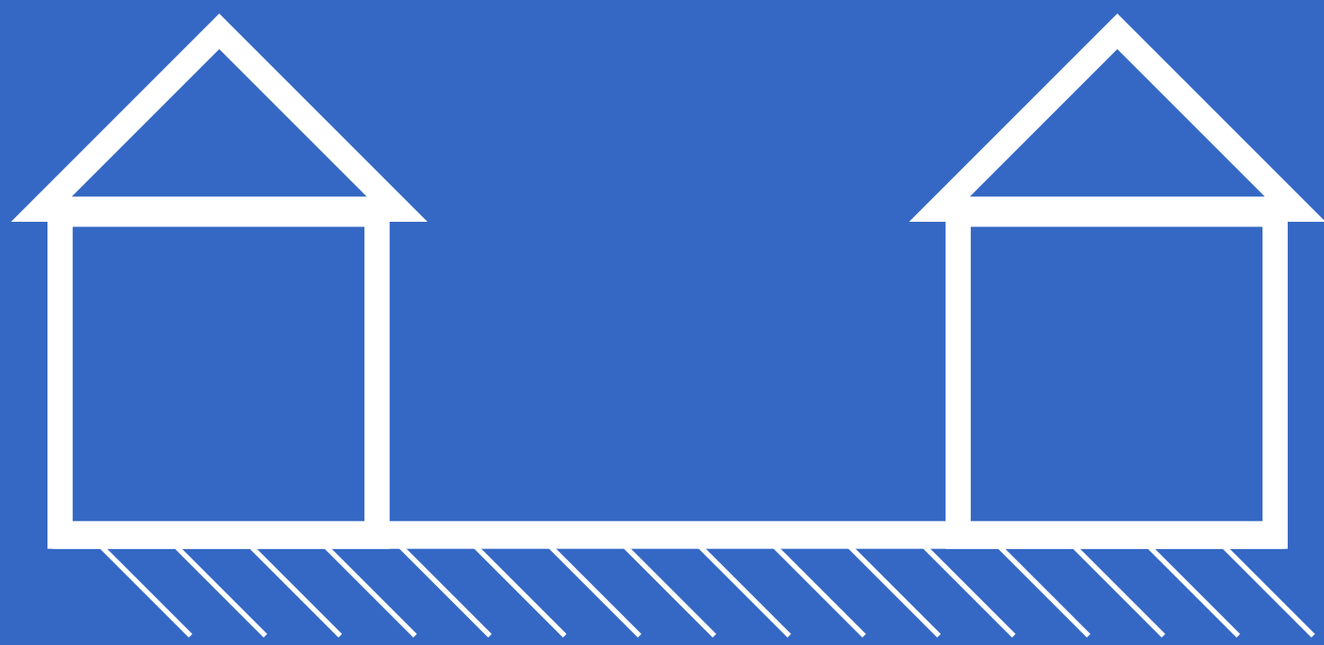


Foreword by Grady Booth

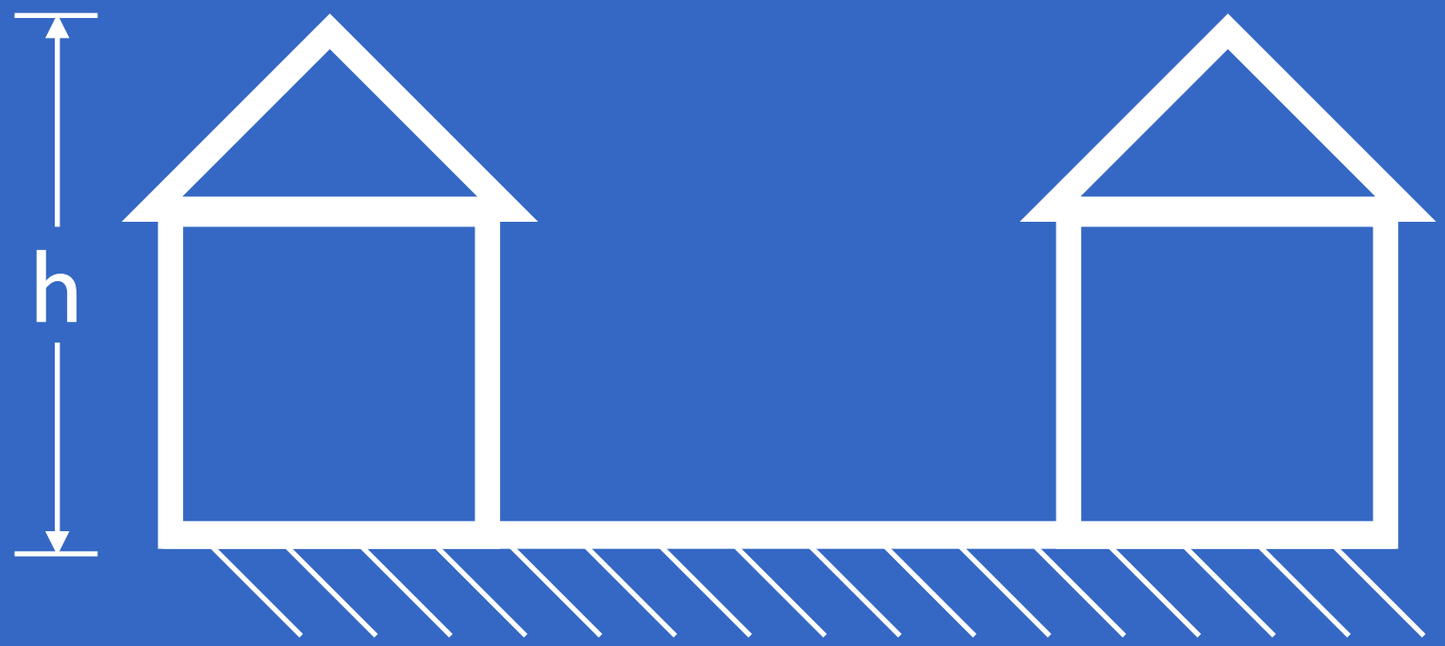
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

# Prepackaged Solution

# Common Problem



# Wide Boulevard




# Wide Boulevard


# Wide Boulevard

Name: Wide Boulevard

Problem: Open Feeling

Solution: Width = 2 * Ht

# Why Bother?

# Why Bother?

1) Prepackaged Solutions

# Why Bother?

1) Prepackaged Solutions

2) Examples of Good Design

# Why Bother?

1) Prepackaged Solutions

2) Examples of Good Design

3) Big Ideas

# Some Examples

Name: Template Method

Problem:

Solution:

Name: Template Method

Problem:Varying details

Solution:

# Documents

# &

# Rendering

template*.rb

## Document

```
def render
  render_title(title)
  text.each_line do|l|
    render_line(l)
  end
  ...
```

## PlainDocument

```
def render_title(t)
end

def render_line(l)
end
```

## HtmlDocument

```
def render_title(t)
end

def render_line(l)
end
```

# Template Method

Name: Template Method

Problem:Varying details

Solution: Push the details into a subclass

# Big Ideas

1) Separate out change

# The Problem

# is

# The Problem

# is

# Commitment

# I start off fine!

doc1 = HtmlDocument.new(...)

# I start off fine!

doc1 = HtmlDocument.new(...)

# But then I need to render doc1 in plain text...

```
# I start off fine!

doc1 = HtmlDocument.new(...)

# But then I need to render doc1 in plain text...

doc2 = PlainDocument.new(doc1.title, doc1.text)

doc2.render
```

# Try Again

# Strategy Pattern

## Document

```
def render
  @renderer.render(self)
end
```

## PlainRenderer

```
def render(doc)
  puts doc.title
  ...
end
```

## HtmlRenderer

```
def render(doc)
  puts "<title>"
  puts @doc.title
  puts "</title>"
  ...
end
```

# Strategy

## Document

```
def render
  @renderer.render(self)
end
```

## PlainRender

```
def render(doc)
  puts doc.title
  ...
end
```

## HtmlRenderer

```
def render(doc)
  puts "<title>"
  puts @doc.title
  puts "</title>"
  ...
end
```

# Strategy

Name: Strategy

Problem:Varying details

Solution: Push the whole job into a separate object

# Big Ideas

1) Separate out change
2) Composition over inheritance

# A Different Problem

Obs*.rb

## Document

```
def text=(t)
  @observers.each {...}
end
```

## Database

```
def on_doc_change(doc)
  # Save the doc
  ...
end
```

## View

```
def on_doc_change(doc)
  puts "<title>"
  puts @doc.title
  puts "</title>"
  ...
end
```

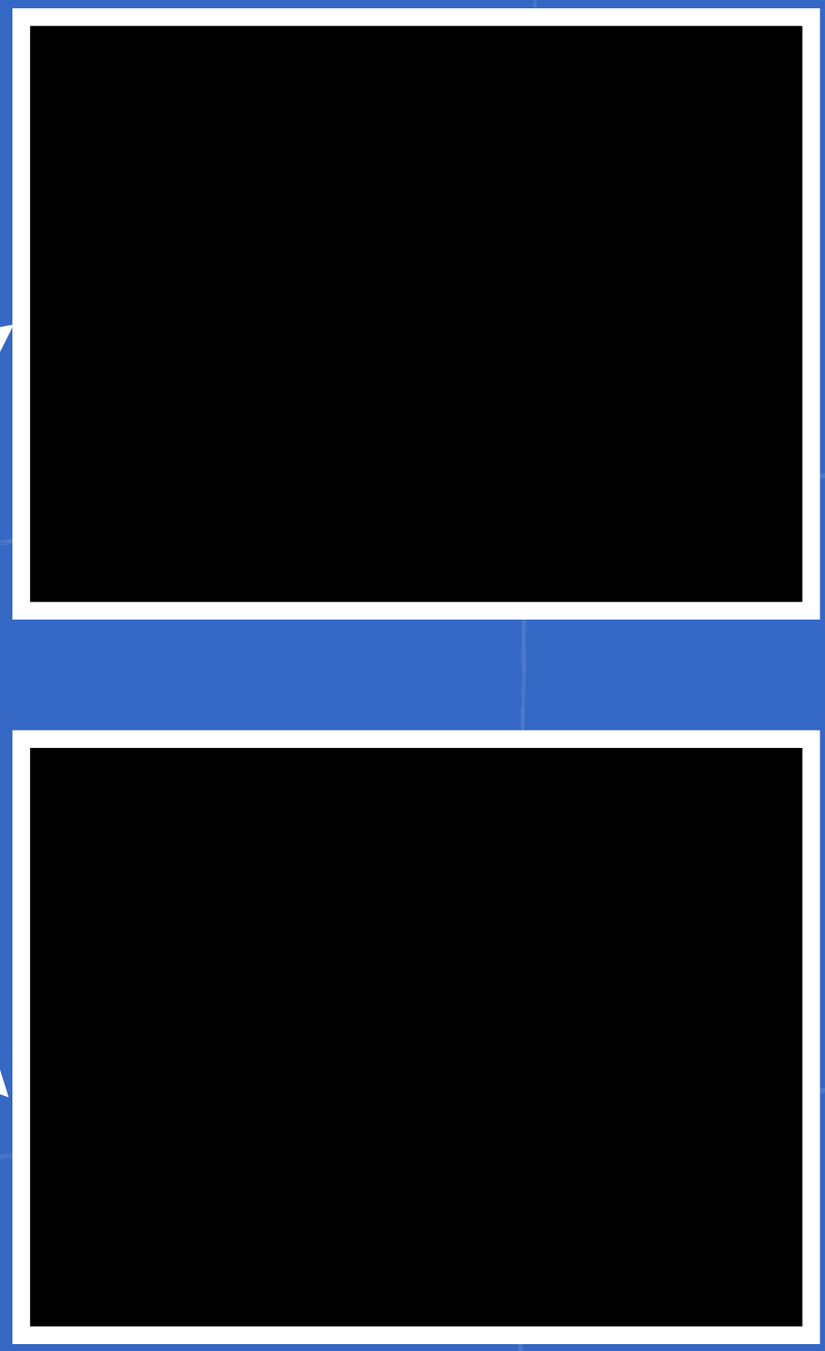
# Observer Pattern

# Observer

# Pattern

## Document

```
def text=(t)
  @observers.each {...}
end
```



on_doc_change(doc)

on_doc_change(doc)

# Black Boxes

# Big Ideas

1) Separate out change
2) Composition not inheritance
3) Interface not implementation

# Good

# Bad

# Good Orange Beatles

# Bad Rampaging Patterns

## Document

```
def render
  @renderer.render(self)
end
```

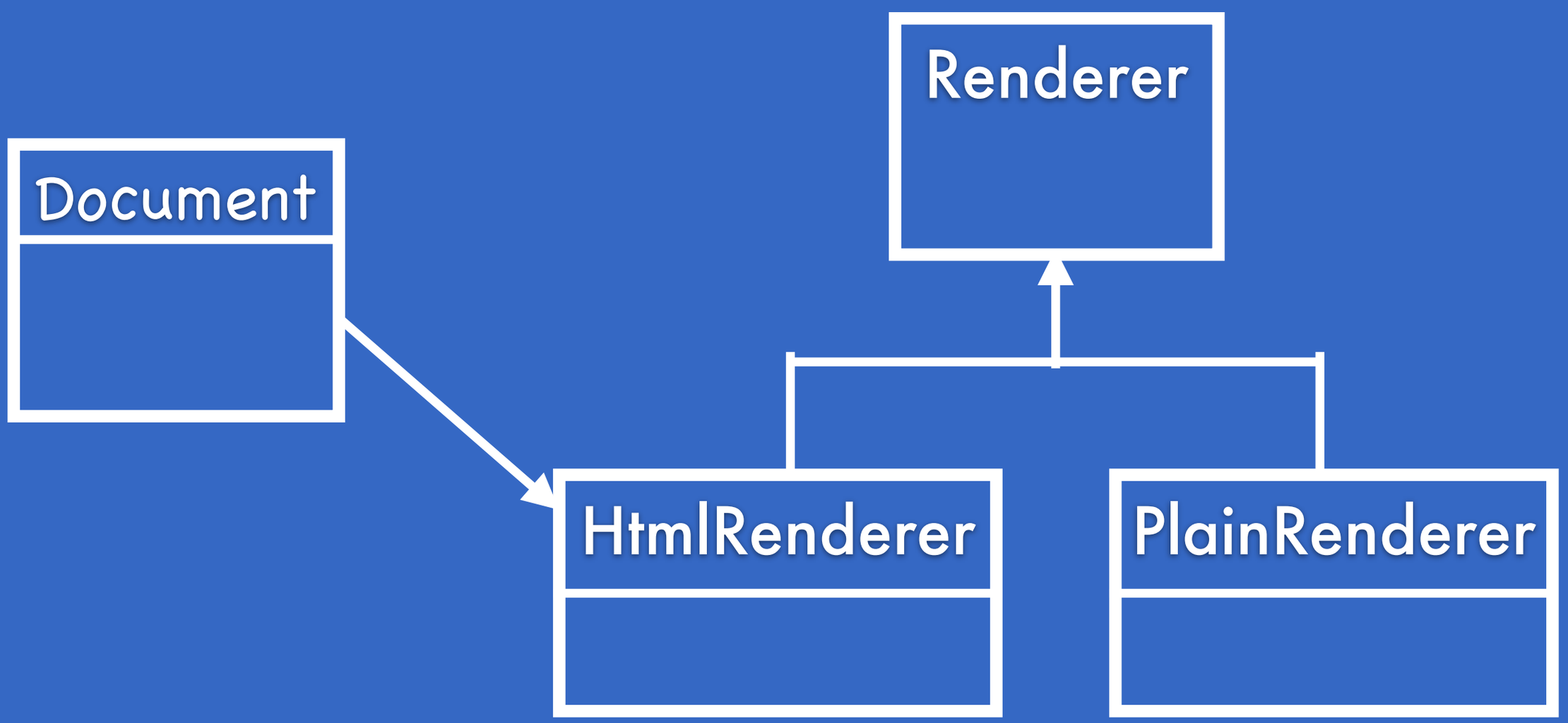
## PlainRender

```
def render(doc)
  puts doc.title
  ...
end
```

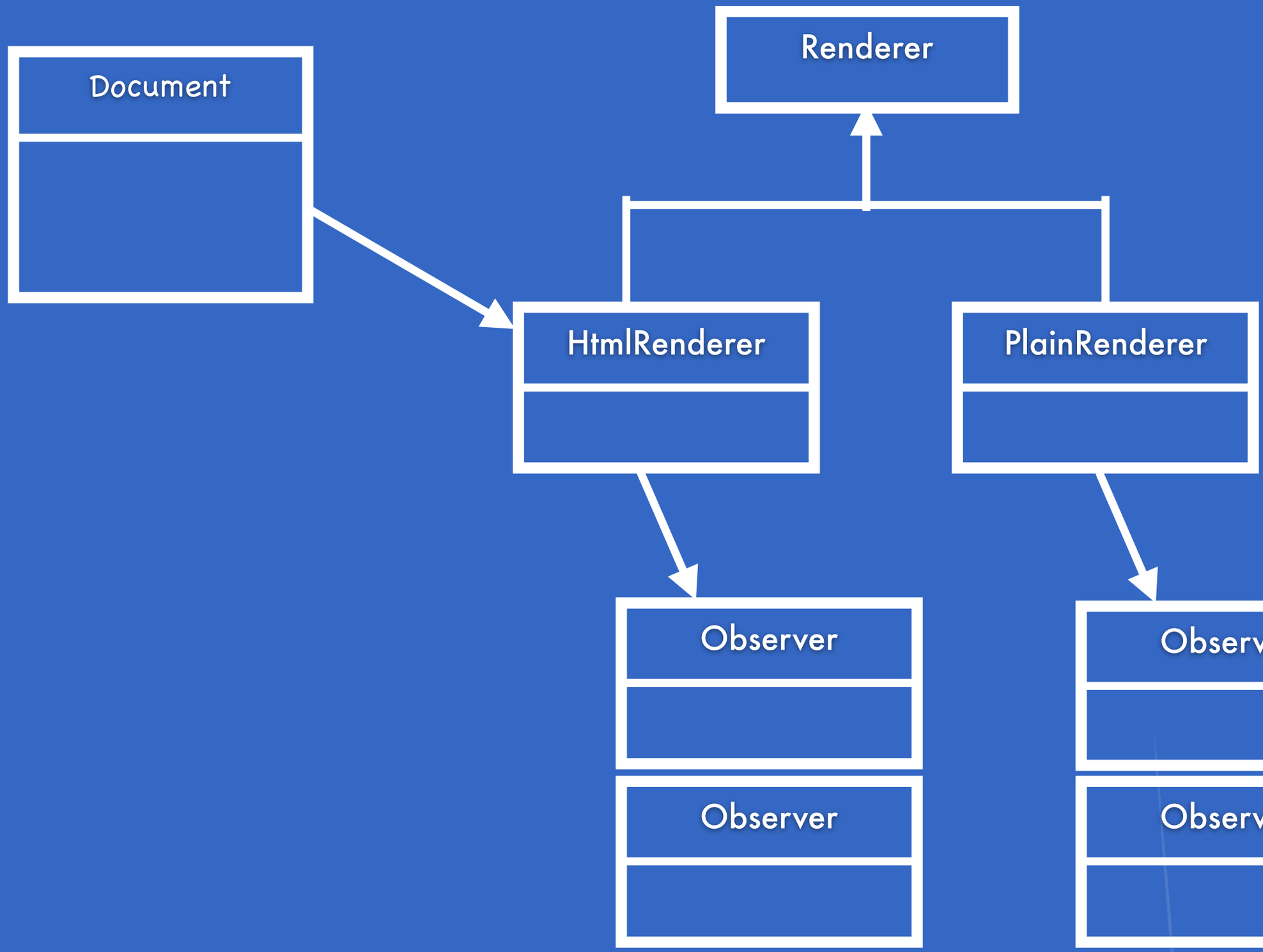
## HtmlRenderer

```
def render(doc)
  puts "<title>"
  puts @doc.title
  puts "</title>"
  ...
end
```

# Strategy




Strategy + Template




Strategy + Insanity

# Big Ideas

1) Separate out change
2) Composition not inheritance
3) Interface not implementation
4) Pain first, then the pattern

# Summary

# Prepackaged Solution

# Common Problem

## Document

```
def render
  render_title(title)
  text.each_line do|l|
    render_line(l)
  end
  ...
```

## PlainDocument

```
def render_title(t)
end

def render_line(l)
end
```

## HtmlDocument

```
def render_title(t)
end

def render_line(l)
end
```

Template Method

## Document

```
def render
  @renderer.render(self)
end
```

## PlainRender

```
def render(doc)
  puts doc.title
  ...
end
```

## HtmlRenderer

```
def render(doc)
  puts "<title>"
  puts @doc.title
  puts "</title>"
  ...
end
```

# Strategy

## Document

```
def text=(t)
  @observers.each {...}
end
```

## Database

```
def on_doc_change(doc)
  # Save the doc
  ...
end
```

## View

```
def on_doc_change(doc)
  puts "<title>"
  puts @doc.title
  puts "</title>"
  ...
end
```

# Observer Pattern

# Big Ideas

1) Separate out change
2) Composition not inheritance
3) Interface not implementation
4) Pain first, then the pattern

# Questions?

russ@russolsen.com

@russolsen

cognitect.com