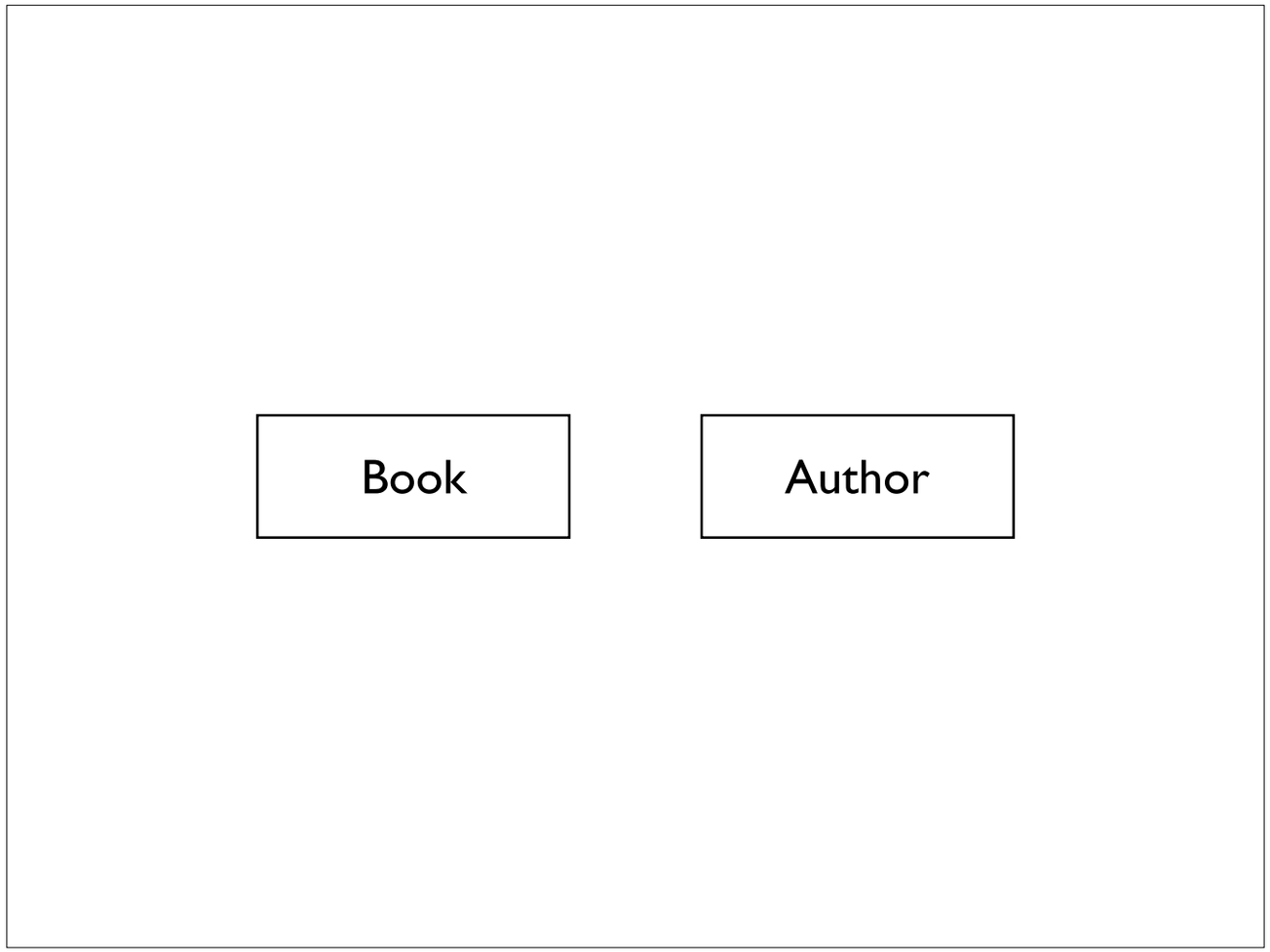
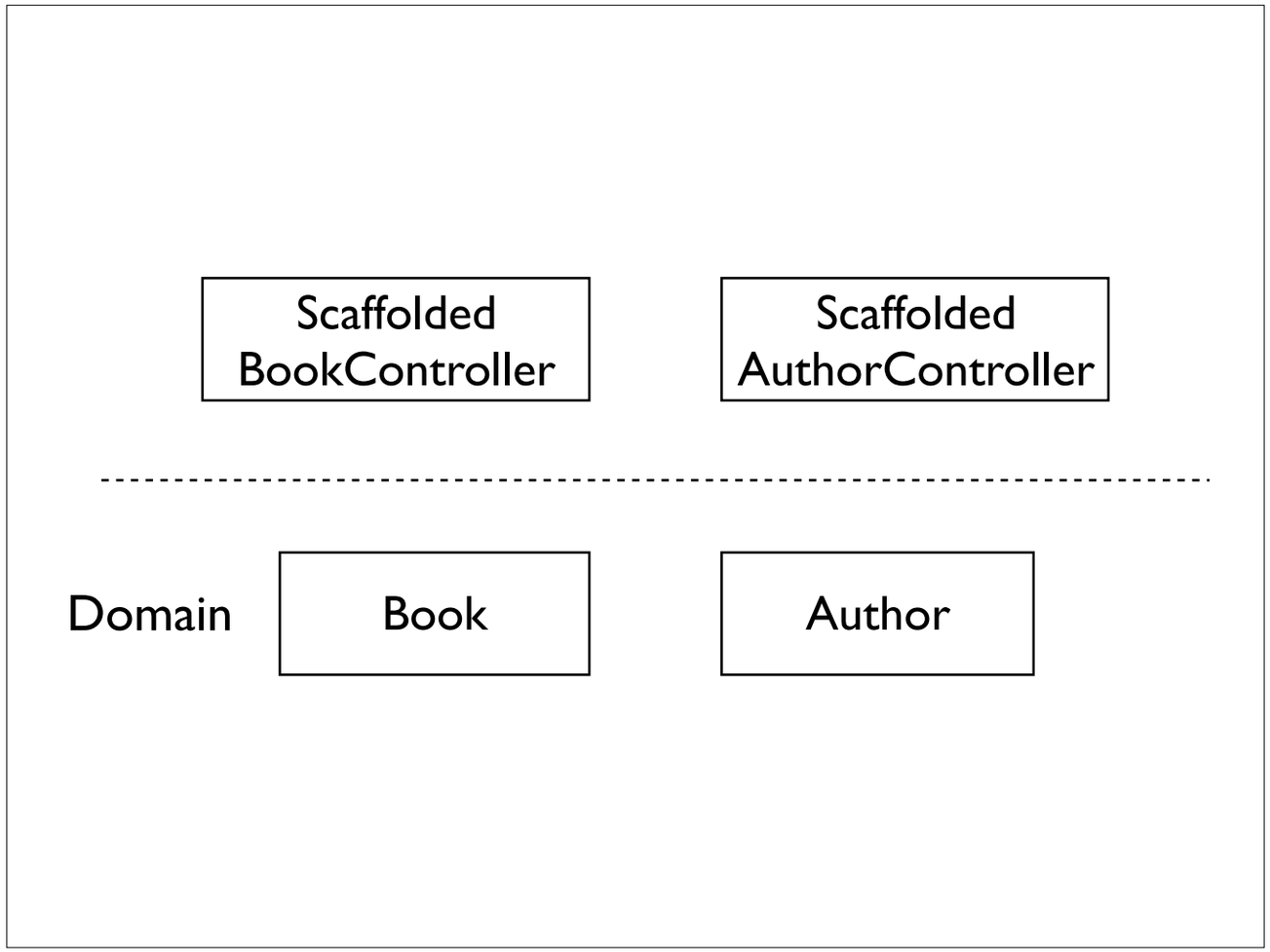
Application architectures in Grails

Peter Ledbrook

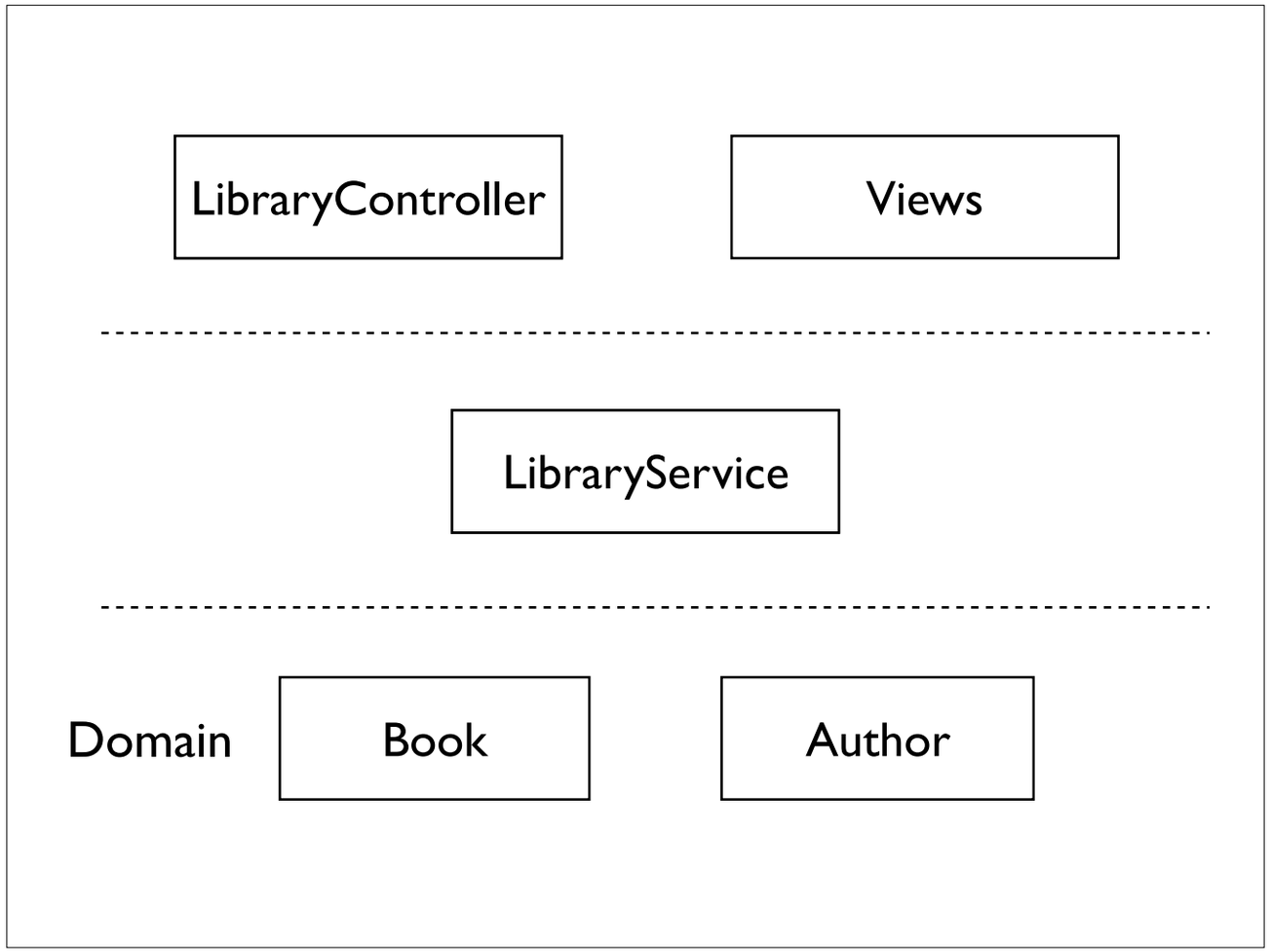
e: peter@cacoethes.co.uk
w: http://www.cacoethes.co.uk
t: @pledbrook



We start with our domain classes using the usual hasMany, belongsTo etc.



Create instant web UI with scaffolded controllers
Can be retained for administrative UI if secured by Spring Security, Shiro, etc.



Build out proper UI using controller actions and views, utilising business logic in the services
Controllers stick to HTTP management

Thank you

Some time left for questions...

Only joking

Is it always the best architecture?

How many people are using it?

Why do we start here?

Why are persistence classes so often the starting point?

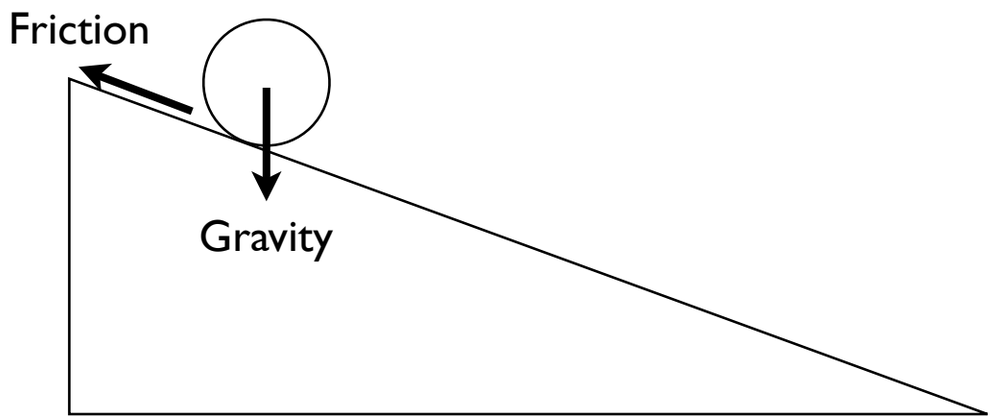
"The database is just a
detail that you don't need
to figure out right away"

NO DB - May 2012
Robert "Uncle Bob" Martin

Domain-driven Design

Not the same as domain classes first
Model your domain first without integration concerns
It’s in operation at all stages of development, not just up-front
Reminded of a problem domain related to managing meetings and attendees - focused so hard on the DB tables that the program logic was a dog’s breakfast.

Think Physics



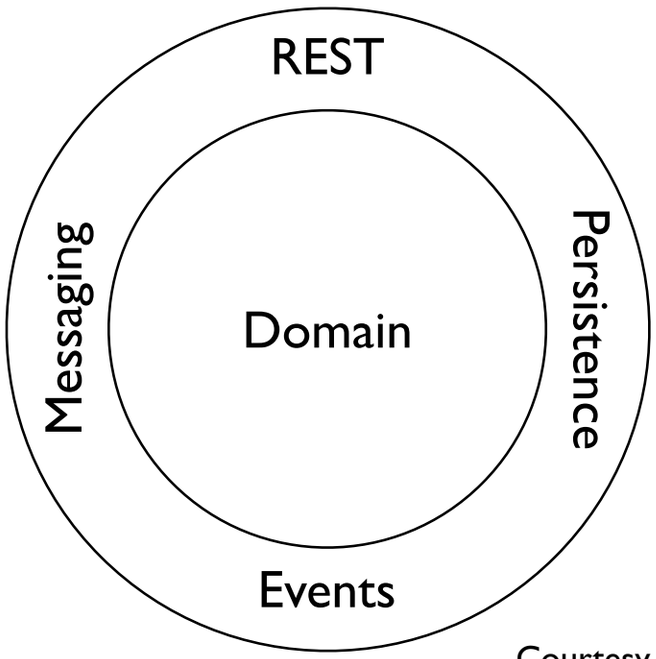
from Wikmedia Commons

We can use the model to calculate useful information, such as how long it takes for a ball to roll down a hill
The model only includes significant complexity – ignores the rest
Formula 1 makes use of CFD because they need it at the bleeding edge

Remember: you're trying to solve a business problem

You need to understand the problem domain
The model needs to reflect that understanding
Gradle is a great example of a rich, evolving, and useful domain model

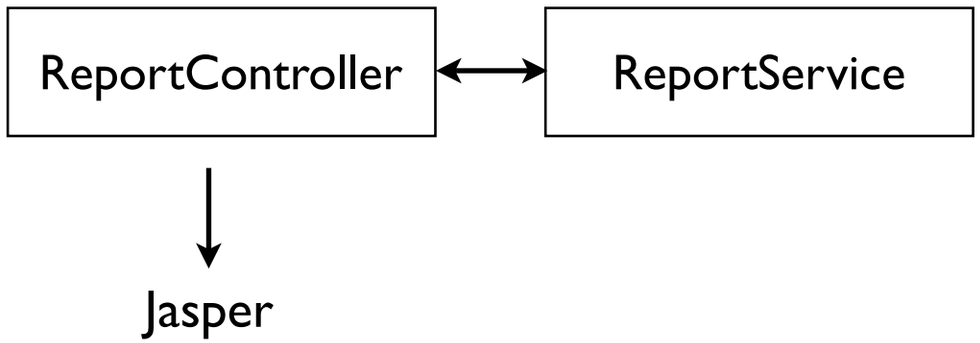
The Life Preserver



Courtesy of Simplicity Itself

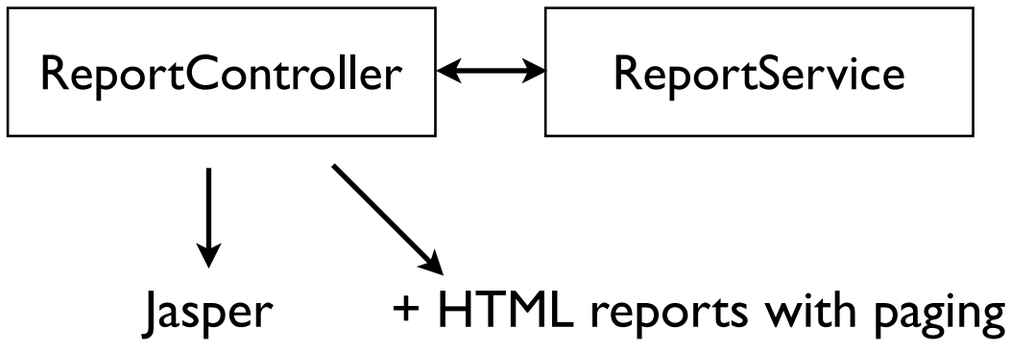
Note how persistence is treated as an integration point
Opens up novel approaches
Could use mybatis + Flyway instead of GORM for example

An example - reporting



High volume transactional web site, optimised for write
Everything was OK at this point

An example - reporting



Breakage!

The logic for building reports was complex
Who is responsible for the paging? The HTML generation?
Where is the state kept? The service? A domain class?

An example - reporting

It's a command object!

Let's try again
The logic for building the report and pagination is in the PublisherReport class

An example - reporting

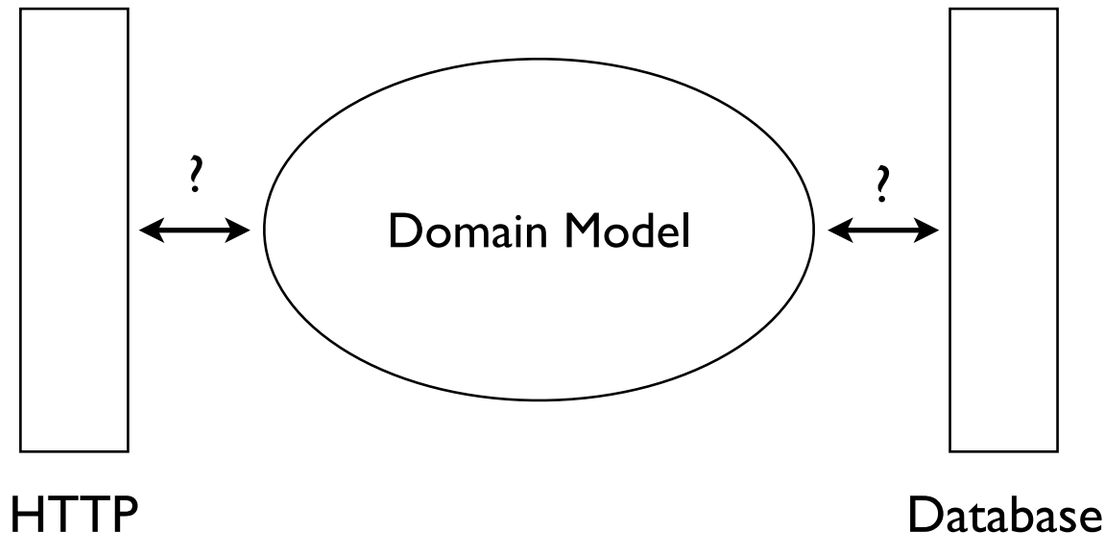
```
class ReportController {
    def print(PublisherReport report) {
        JasperRenderer.render report
    }

    def json(PublisherReport report) {
        render report as JSON
    }

    ...
}
```

The controller is now very thin
The report can support parameters for sub-reports etc.
The domain model is embodied in the command object

What is my domain?

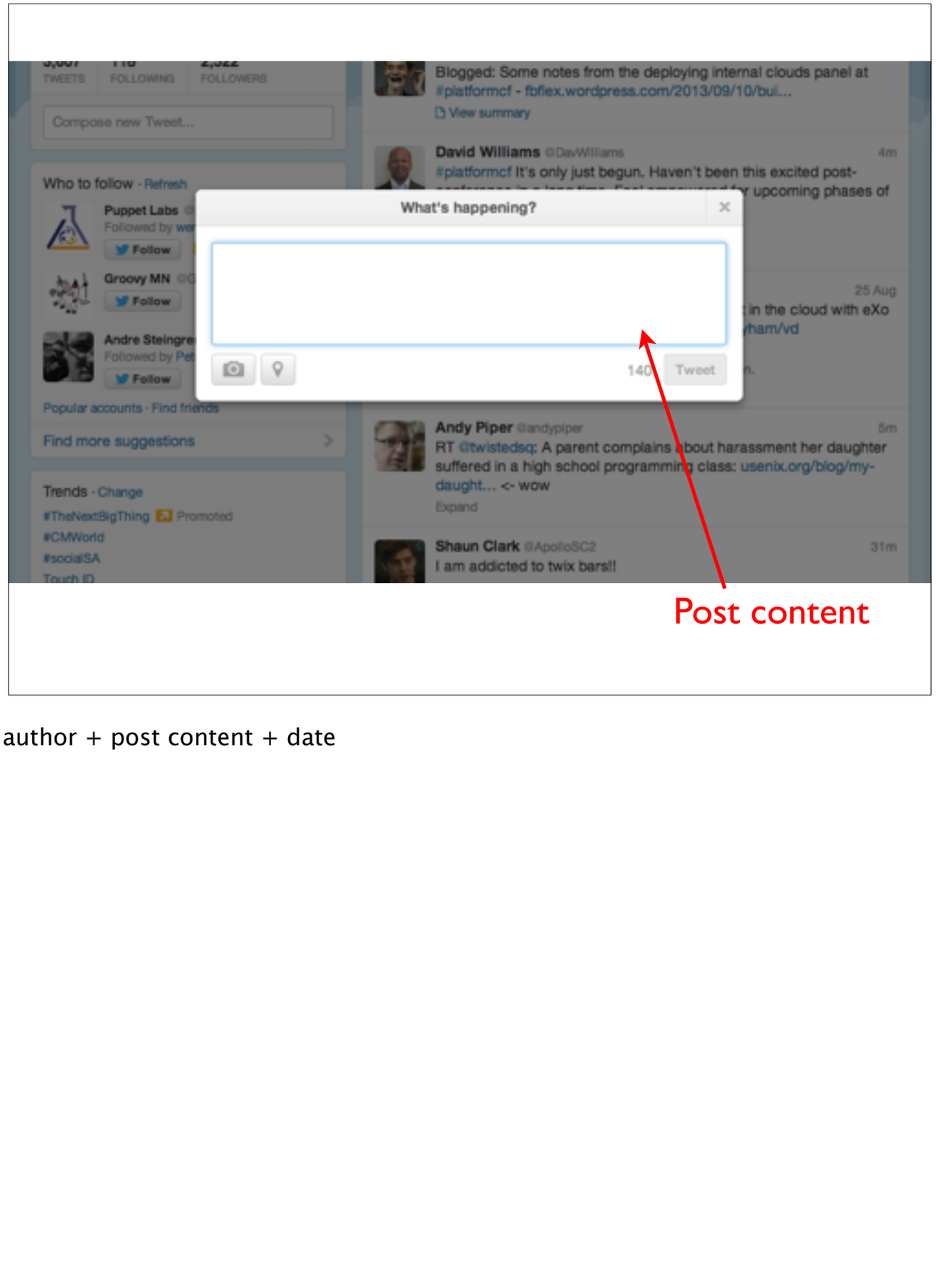


Always ask yourself this question throughout life of project
And is it closer to the user's perspective or the persistence model? Or neither?
Former argues for a model based on command objects, the latter based on domain classes.

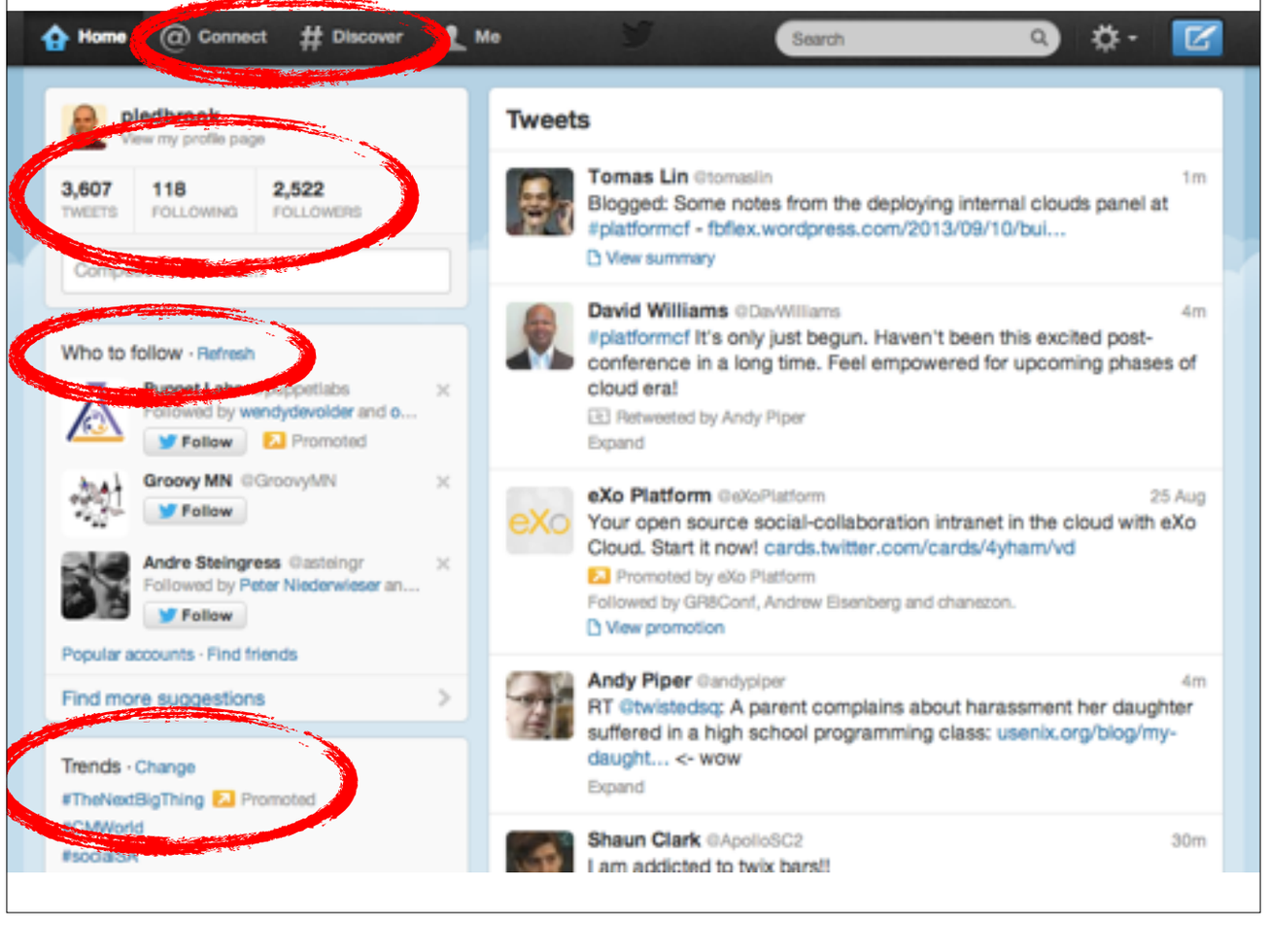
CQRS

Command
Query
Responsibility
Segregation

The writes use a different model from the reads
Will be coming back to this later



The command model is very simple: author + post content + date



Query model much more complex
Multiple timelines
Conversation threads
Retweets

So working from your domain first is a good thing
And remember that different contexts have potentially different views of the model, i.e. the user/client, persistence, other system components
DDD doesn't preclude the CRUD/service-based architecture
So what are the driving forces behind architecture beyond the model?

Rich clients

We’re not talking Warren Buffet here
Things like GMail

Google I/O 2012

400 million

Android activations to date

Apple WWDC 2012

365 million

iOS devices sold to date

Let’s not forget Firefox OS
Lots of people potentially hitting a site at any one time!
Typical Grails architecture may struggle to handle the load (OpenSessionInViewInterceptor, transactions, GSPs, thread-per-request)

AJAX + JSON endpoints

enabler for async

Rich UIs don't talk HTML - use JSON endpoints (aka "REST")
Asset delivery via Resources or asset-pipeline plugins
More scope for asynchronicity, since no wait for full page update
Grails 2.3 introduces some nice features for REST

What's the need for SiteMesh & GSP then?

Difficult to impossible to remove these currently
Grails 3 will finally extricate them, allowing you to remove them from your project

An aside

If the whole Java client thing had worked out, would you use it for every web application you wrote?

Would you use it for Wikipedia?

Before jumping onto the whole “single-page app” bandwagon, work out whether it’s appropriate for your app

Async

for scalability

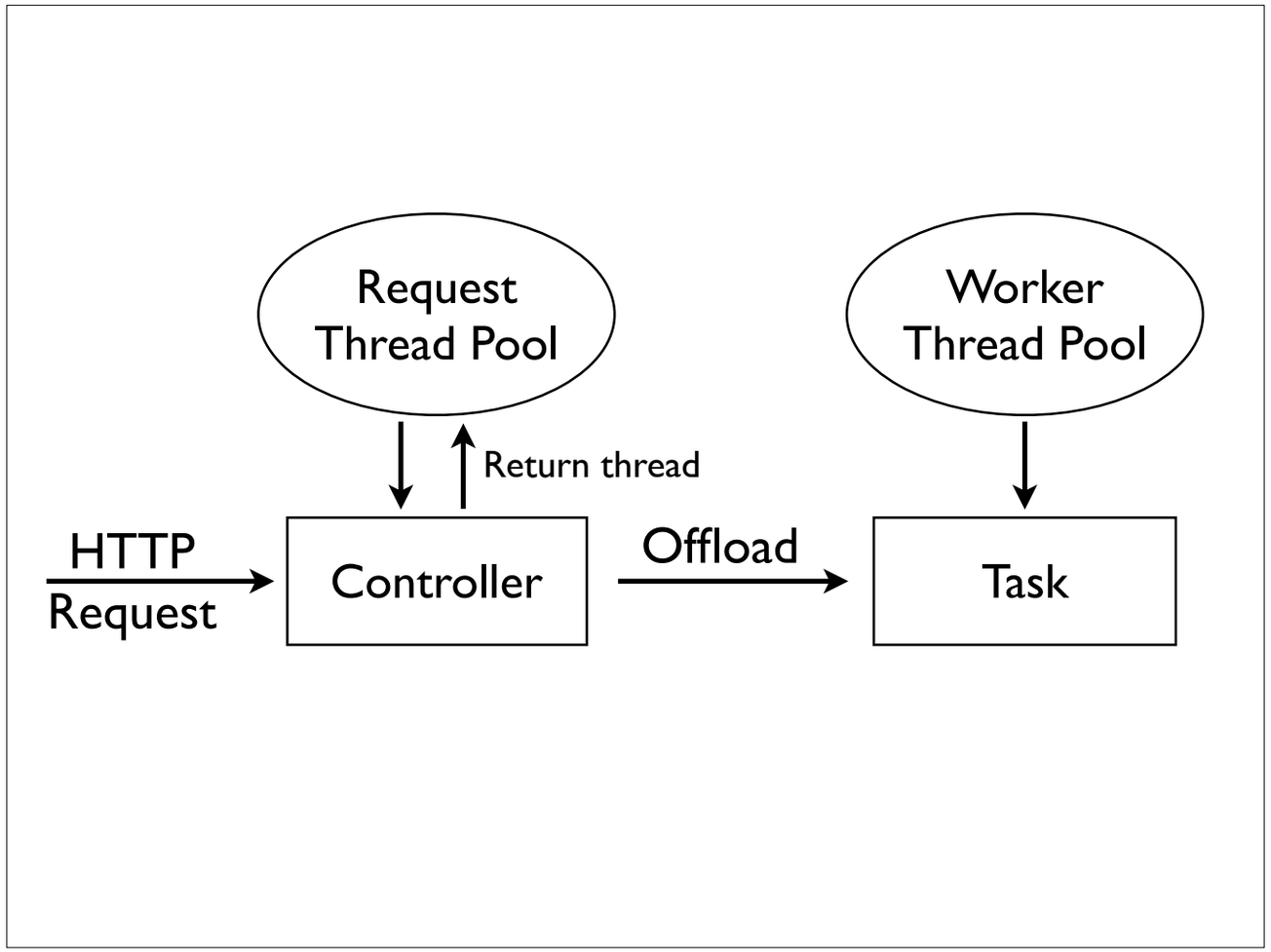
To solve the problem of dealing with large number of concurrent requests
Without adding lots more servers

Grails Promise API

```
import static grails.async.Promises.*

class ReportController {
    def print(PublisherReport report) {
        task {
            // Expensive report creation here
        }
    }
    ...
}
```

We can now return Promise instances from actions
The expensive task no longer blocks the request thread, but...



The request threads are now free, but burden is on worker thread pool
If all worker tasks are synchronous, have we gained scalability?
In cases where just a few URLs are blocking for long(ish) periods of time, yes (kind of)
But otherwise, now bottleneck is on worker thread pool

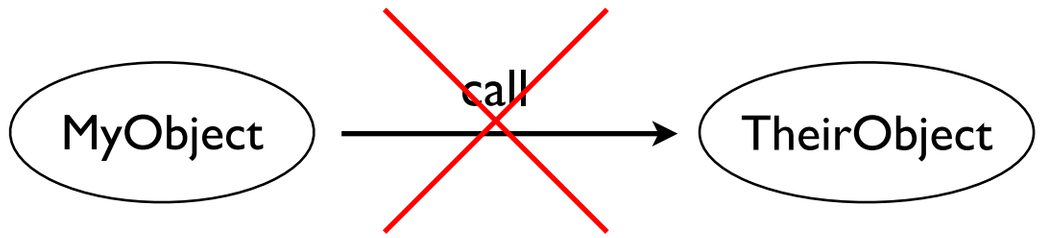
Make efficient use of server resources

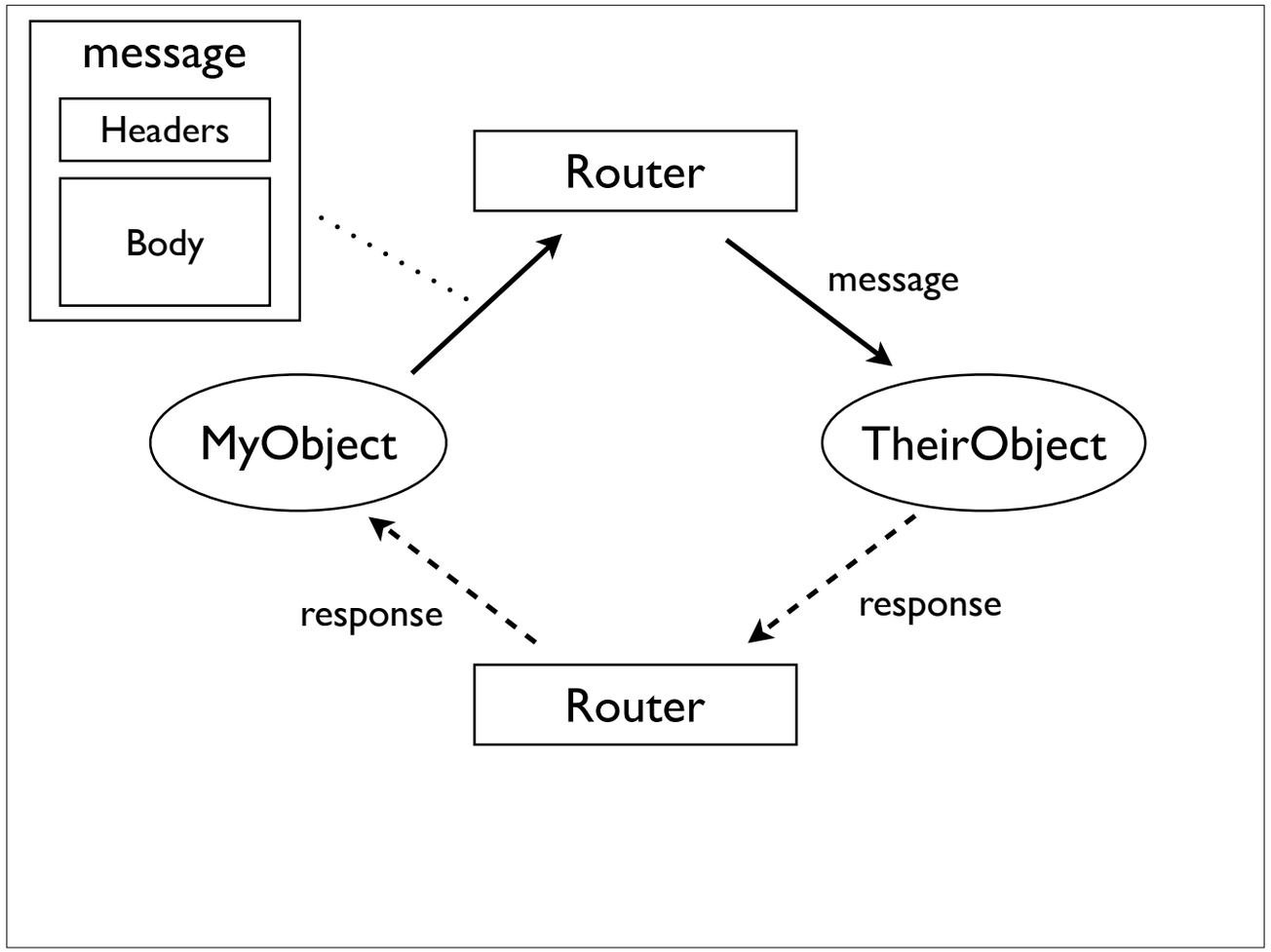
Build your model around async
Loosely coupled REST services, messaging, event bus
Remember that some things are inherently synchronous (think Fibonacci)

Concurrency is still not easy!

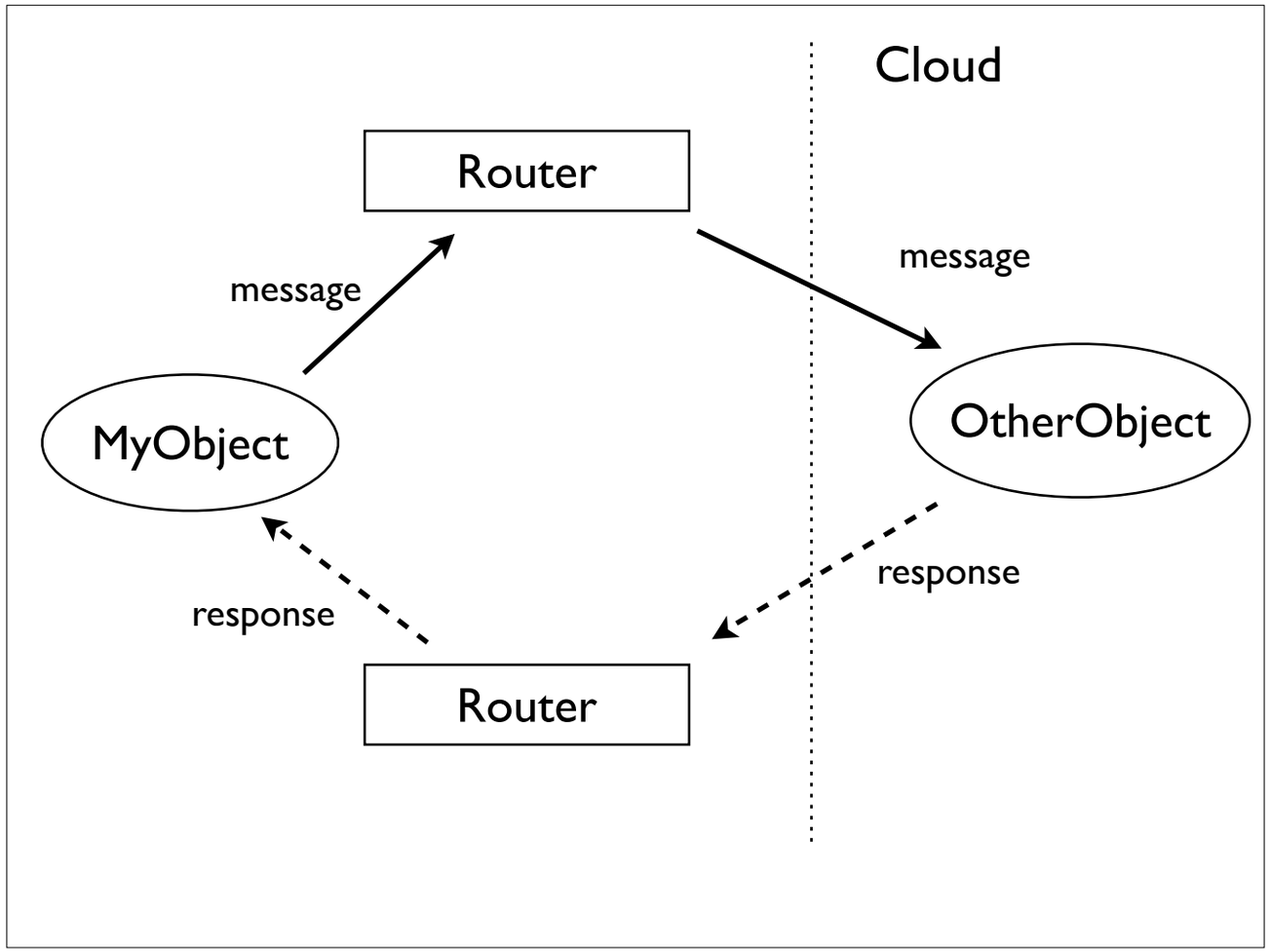
Messaging

A common solution to concurrency and scale


MyObject
call
TheirObject



Decoupling via messages
Encourages separation of concerns & responsibilities

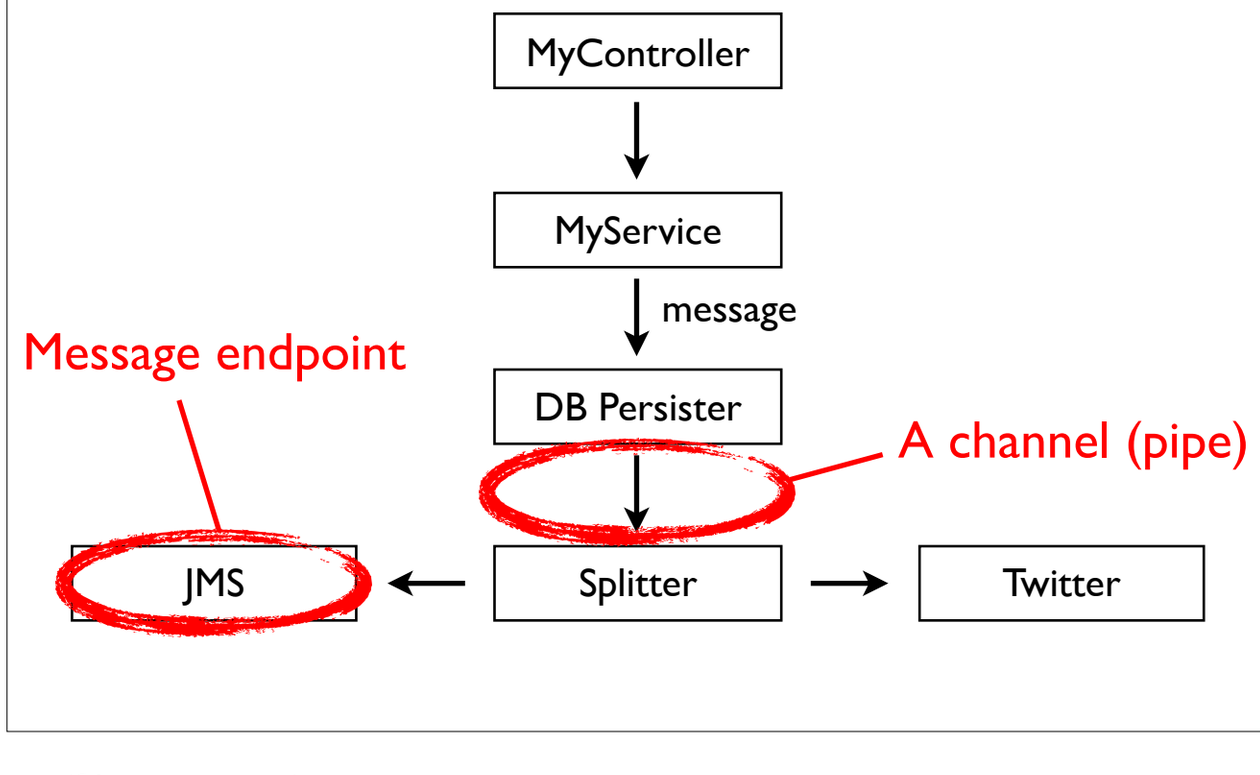


Easy to change and move objects
Scales well (think Actor model of concurrency)

Internal	External
Events	JMS
Spring Integration	RabbitMQ
Apache Camel	

Internal and external can be integrated
Events is a special case of messaging (which I look at next)

Spring Integration



Based on Enterprise Integration Patterns (filters & pipes)
Many options for routing and transforming messages
Logging adapters and wire tapping for debug

Spring Integration Groovy DSL

```
def builder = new IntegrationBuilder()

def ic = builder.doWithSpringIntegration {
    messageFlow("flow") {
      filter { it == "World" }
      transform(inputChannel: "transformerChannel") {
          "Hello " + it
      }
      handle { println "**** $it ****" }
    }
}

ic.send "flow.inputChannel", "World"
ic.send "transformerChannel", "Earth"
```

Debugging

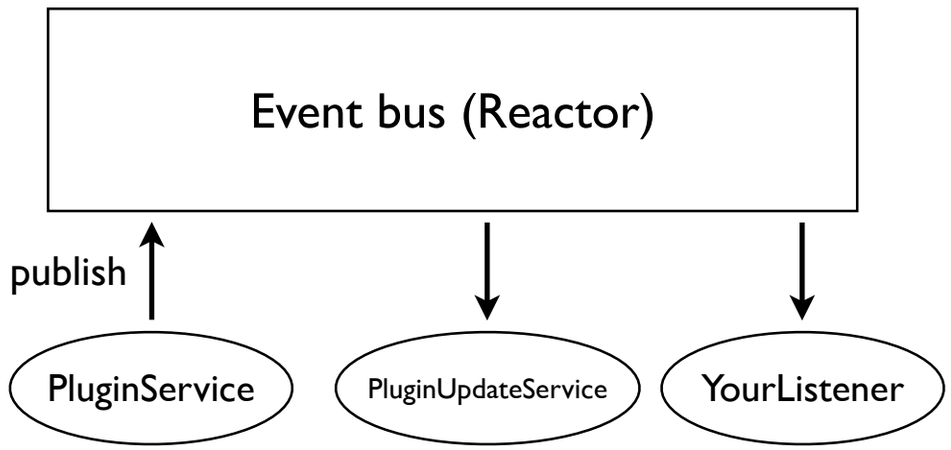
Code comprehension

Performance (kind of)

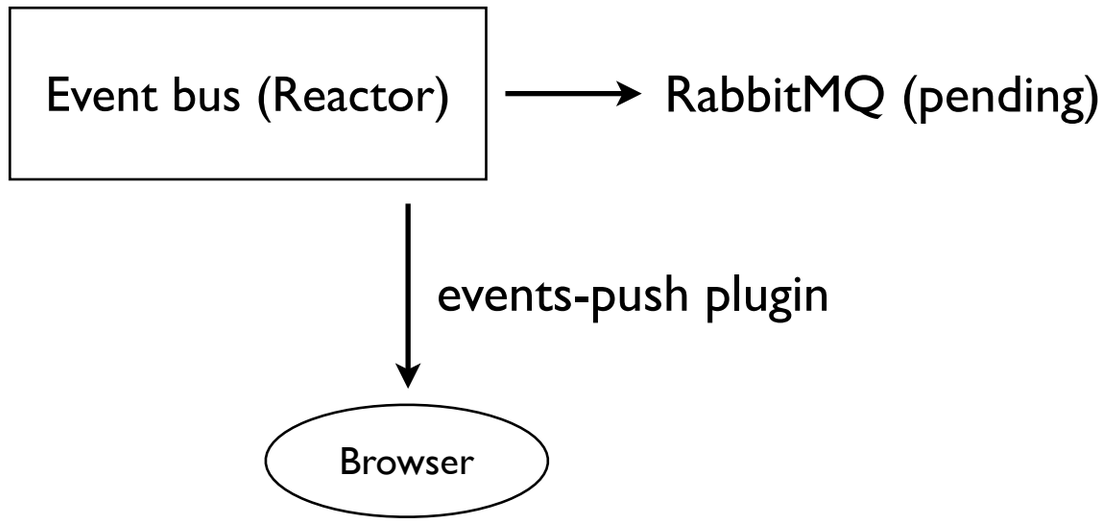
Events

Special case of messaging

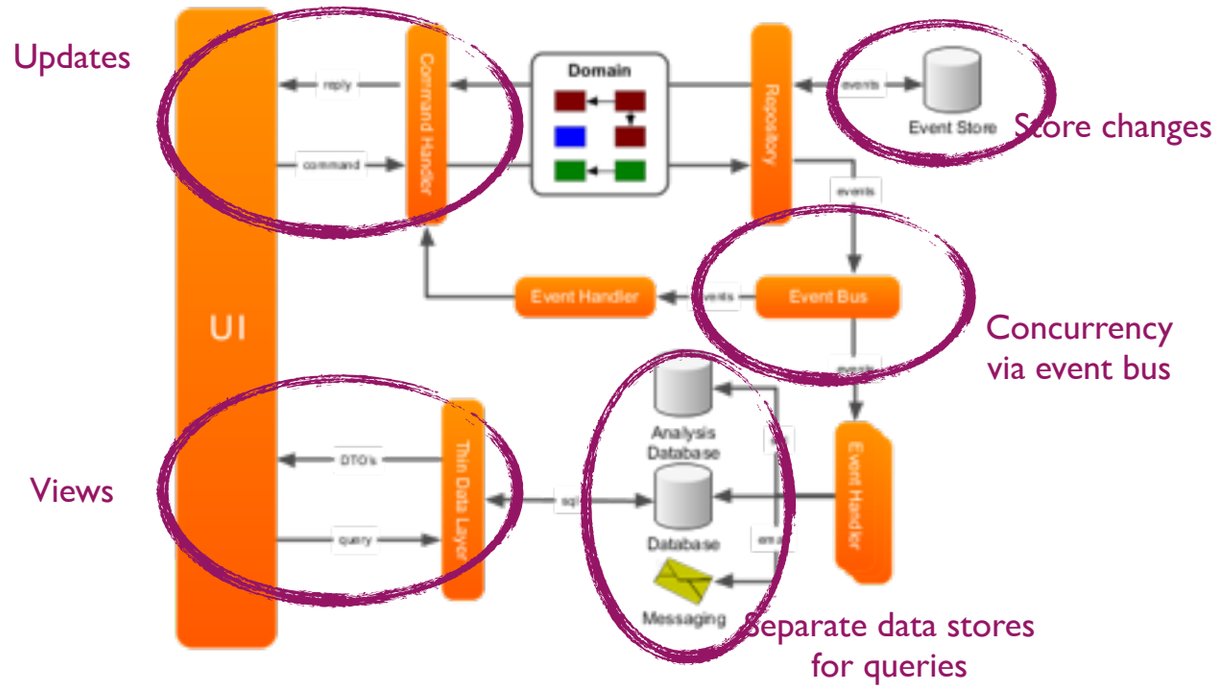
Event bus - (grails-)events



Event bus - (grails-)events

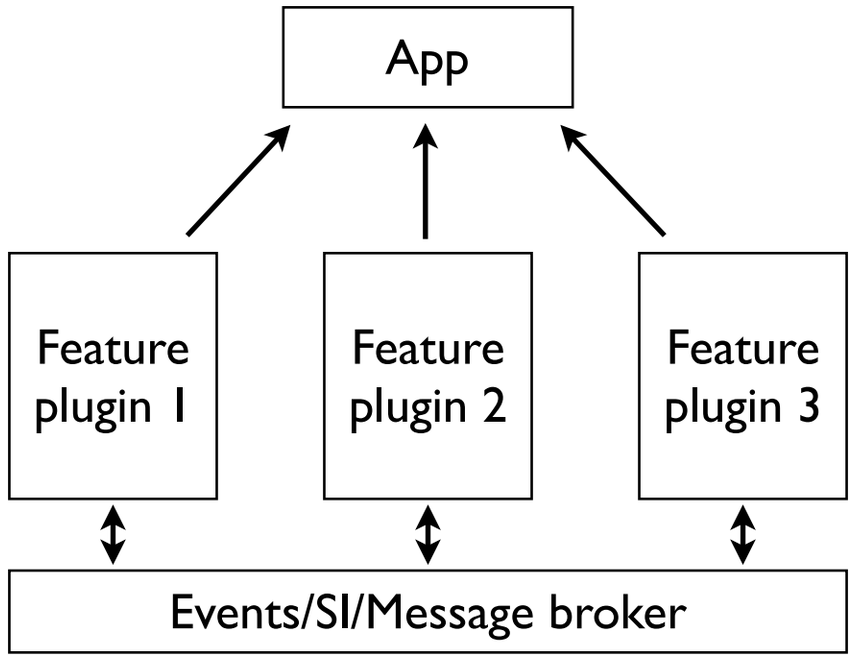


A CQRS architecture



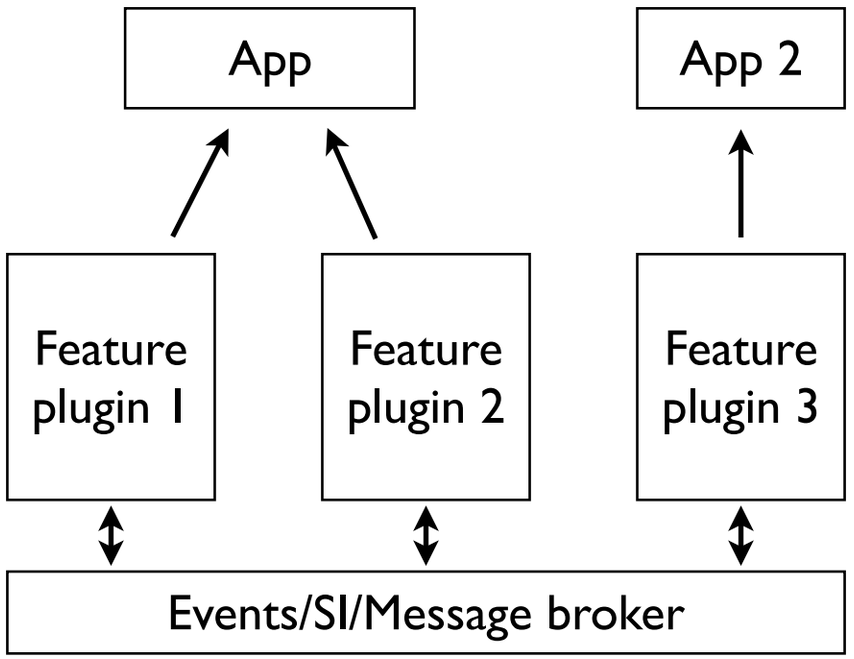
Why? Updates and querying often have different data requirements.
For example, Lanyrd use Redis structured data support
All read databases can be rebuilt from master events DB
CQRS designed for scale

Plugin Architectures



So why use messages to interact between the plugins?

Plugin Architectures



Easy to separate out into apps deployed independently

Ultimately, think about what you need...

...don't just go the "standard" route automatically

Thank you