Consuming web services asynchronously with Futures and Rx Observables

Chris Richardson

Author of POJOs in Action

Founder of the original CloudFoundry.com

@crichardson

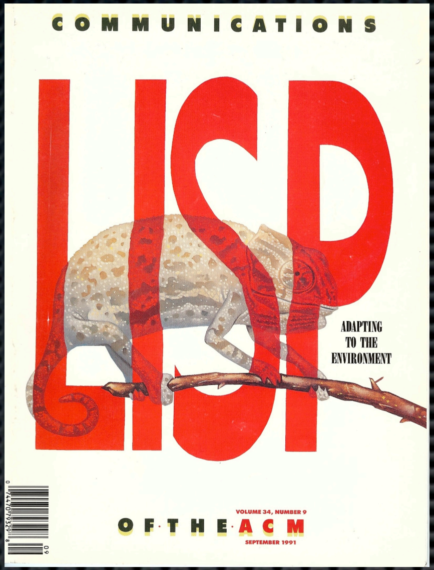
chris@chrisrichardson.net

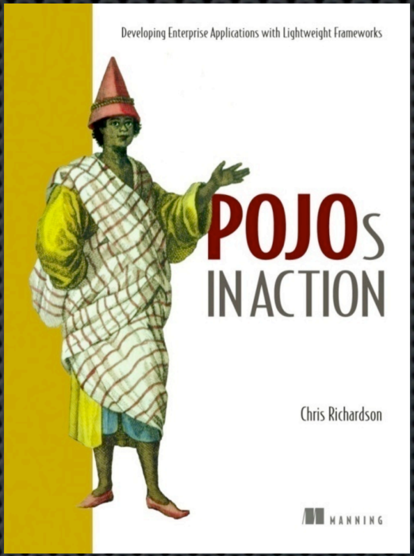
http://plainoldobjects.com

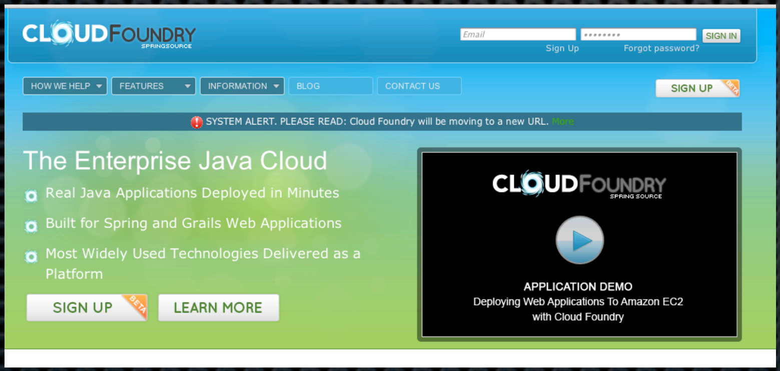
Presentation goal

Learn how to use (Scala) Futures and Rx Observables to write simple yet robust and scalable concurrent code

About Chris




Developing Enterprise Applications with Lightweight Frameworks
POJOs IN ACTION
Chris Richardson
MANNING


CLOUDFOUNDRY SPRINGSOURCE

Email | •••••••• | SIGN IN

Sign Up | Forgot password?

HOW WE HELP | FEATURES | INFORMATION | BLOG | CONTACT US | SIGN UP

SYSTEM ALERT. PLEASE READ: Cloud Foundry will be moving to a new URL. More

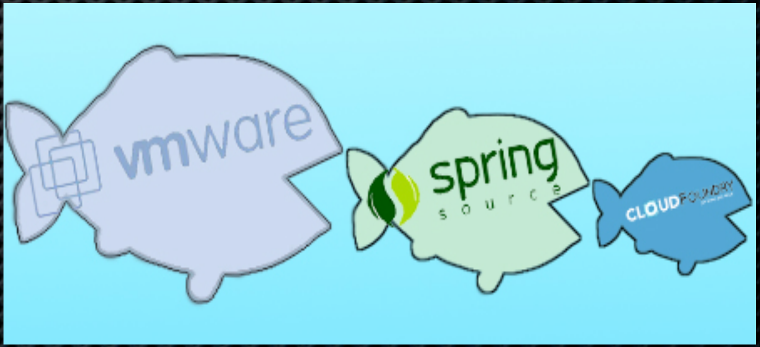
The Enterprise Java Cloud

- Real Java Applications Deployed in Minutes
- Built for Spring and Grails Web Applications
- Most Widely Used Technologies Delivered as a Platform

SIGN UP | LEARN MORE

CLOUDFOUNDRY SPRING SOURCE

APPLICATION DEMO
Deploying Web Applications To Amazon EC2 with Cloud Foundry



@crichardson

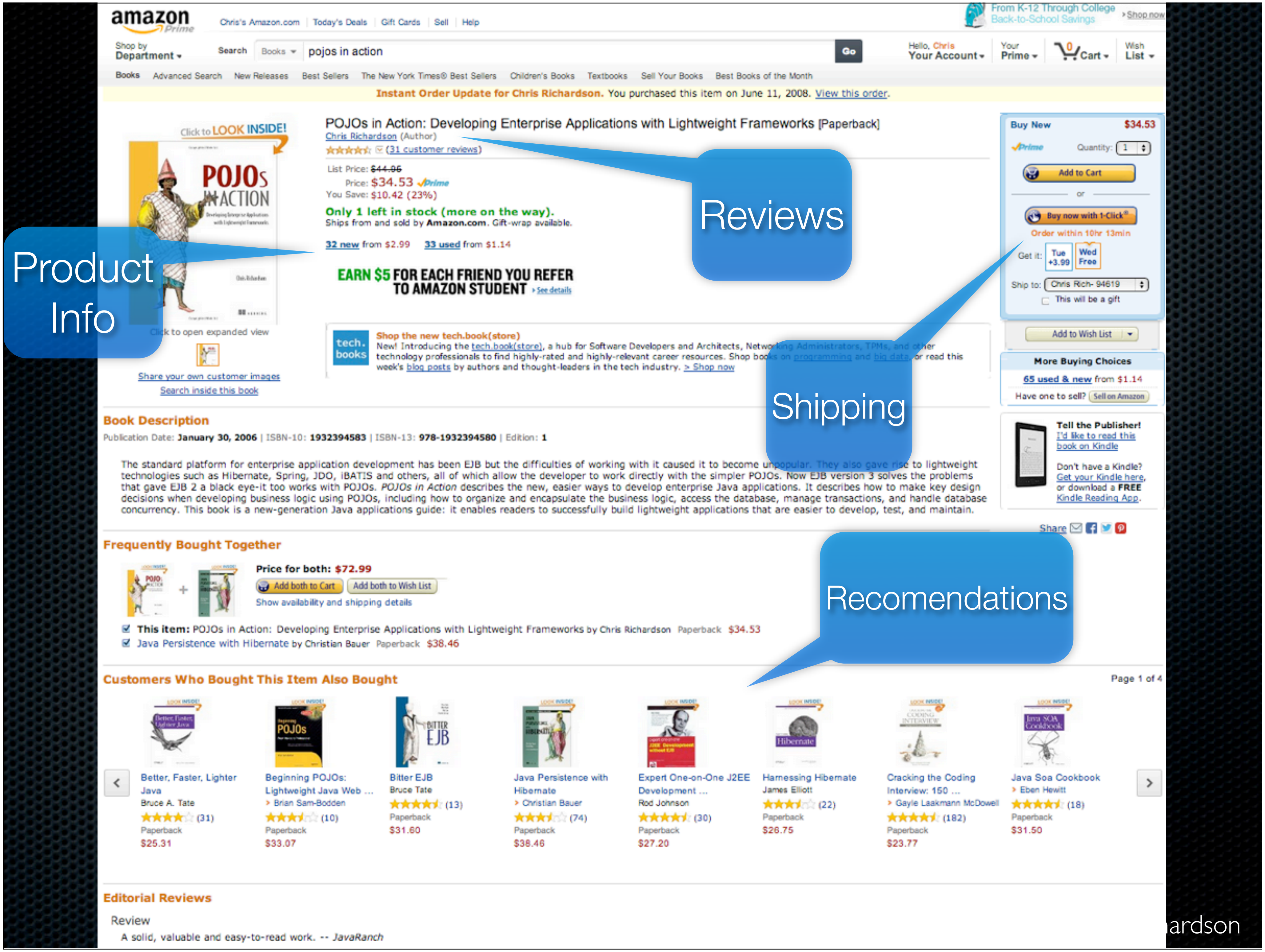
About Chris

- Founder of a buzzword compliant (stealthy, social, mobile, big data, machine learning, ...) startup
- Consultant helping organizations improve how they architect and deploy applications using cloud, micro services, polyglot applications, NoSQL, ...

Agenda

- The need for concurrency
- Simplifying concurrent code with Futures
- Consuming asynchronous streams with Reactive Extensions

Let’s imagine you are building an online store



amazon Prime | Chris's Amazon.com | Today's Deals | Gift Cards | Sell | Help | From K-12 Through College Back-to-School Savings › Shop now

Shop by Department | Search Books | pojos in action | Go | Hello, Chris Your Account | Your Prime | 0 Cart | Wish List

Books | Advanced Search | New Releases | Best Sellers | The New York Times® Best Sellers | Children's Books | Textbooks | Sell Your Books | Best Books of the Month

Instant Order Update for Chris Richardson. You purchased this item on June 11, 2008. View this order.

Click to LOOK INSIDE!

Click to open expanded view

Share your own customer images

Search inside this book

POJOs in Action: Developing Enterprise Applications with Lightweight Frameworks [Paperback]

Chris Richardson (Author)

(31 customer reviews)

List Price: ~~$44.95~~

Price: $34.53 Prime

You Save: $10.42 (23%)

Only 1 left in stock (more on the way).

Ships from and sold by **Amazon.com**. Gift-wrap available.

32 new from $2.99 | 33 used from $1.14

EARN $5 FOR EACH FRIEND YOU REFER TO AMAZON STUDENT › See details

Shop the new tech.book(store)

New! Introducing the tech.book(store), a hub for Software Developers and Architects, Networking Administrators, TPMs, and other technology professionals to find highly-rated and highly-relevant career resources. Shop books on programming and big data, or read this week's blog posts by authors and thought-leaders in the tech industry. > Shop now

Buy New $34.53

Prime | Quantity: 1

Add to Cart

or

Buy now with 1-Click®

Order within 10hr 13min

Get it: Tue +3.99 | Wed Free

Ship to: Chris Rich- 94619

This will be a gift

Add to Wish List

More Buying Choices

65 used & new from $1.14

Have one to sell? Sell on Amazon

Tell the Publisher!

I'd like to read this book on Kindle

Don't have a Kindle? Get your Kindle here, or download a **FREE** Kindle Reading App.

Share

Book Description

Publication Date: **January 30, 2006** | ISBN-10: **1932394583** | ISBN-13: **978-1932394580** | Edition: **1**

The standard platform for enterprise application development has been EJB but the difficulties of working with it caused it to become unpopular. They also gave rise to lightweight technologies such as Hibernate, Spring, JDO, iBATIS and others, all of which allow the developer to work directly with the simpler POJOs. Now EJB version 3 solves the problems that gave EJB 2 a black eye-it too works with POJOs. *POJOs in Action* describes the new, easier ways to develop enterprise Java applications. It describes how to make key design decisions when developing business logic using POJOs, including how to organize and encapsulate the business logic, access the database, manage transactions, and handle database concurrency. This book is a new-generation Java applications guide: it enables readers to successfully build lightweight applications that are easier to develop, test, and maintain.

Frequently Bought Together

Price for both: $72.99

Add both to Cart | Add both to Wish List

Show availability and shipping details

☑ **This item:** POJOs in Action: Developing Enterprise Applications with Lightweight Frameworks by Chris Richardson Paperback $34.53

☑ Java Persistence with Hibernate by Christian Bauer Paperback $38.46

Customers Who Bought This Item Also Bought

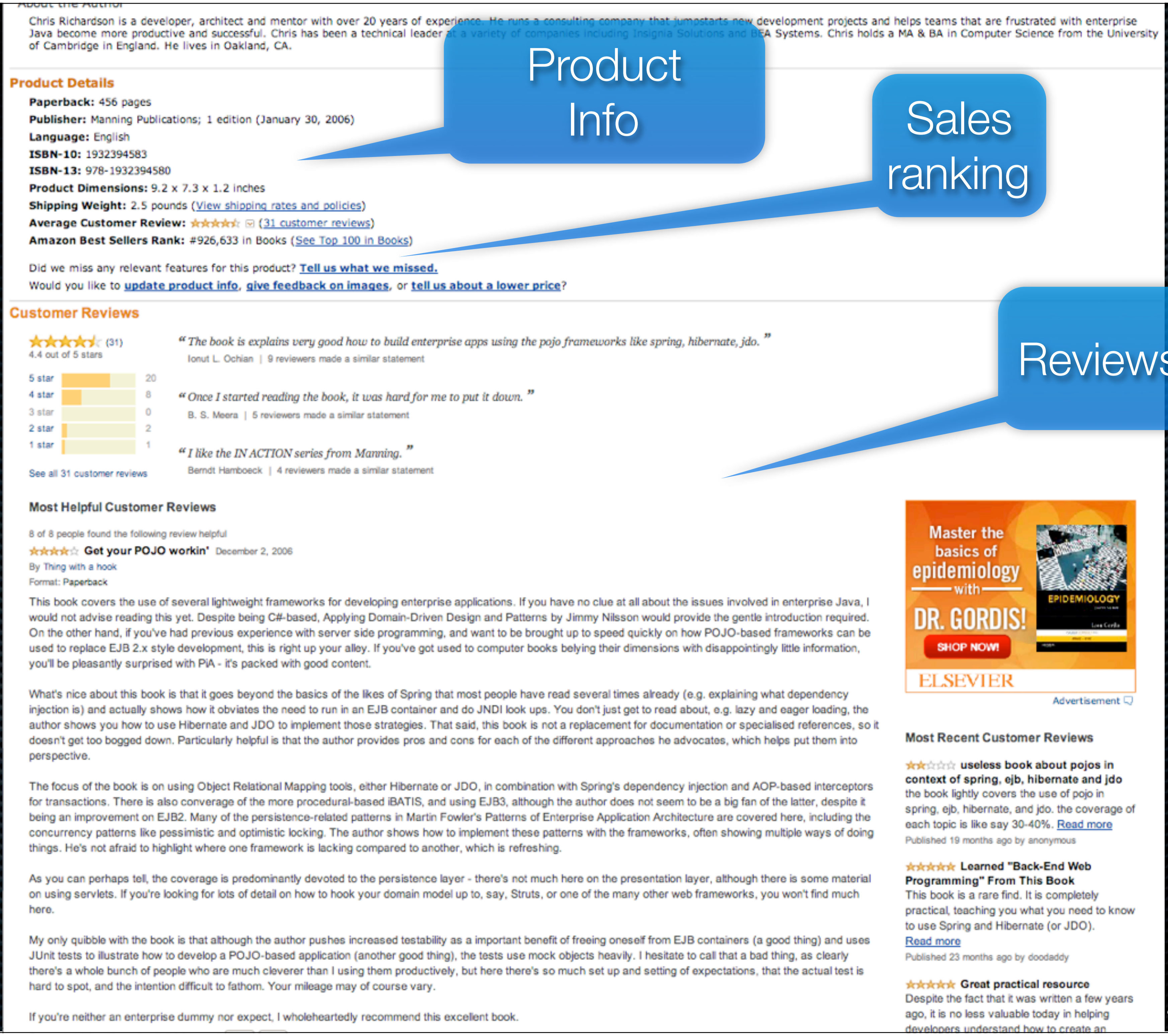
Page 1 of 4

Title	Author	Reviews	Format	Price
Better, Faster, Lighter Java	Bruce A. Tate	(31)	Paperback	$25.31
Beginning POJOs: Lightweight Java Web ...	› Brian Sam-Bodden	(10)	Paperback	$33.07
Bitter EJB	Bruce Tate	(13)	Paperback	$31.60
Java Persistence with Hibernate	› Christian Bauer	(74)	Paperback	$38.46
Expert One-on-One J2EE Development ...	Rod Johnson	(30)	Paperback	$27.20
Harnessing Hibernate	James Elliott	(22)	Paperback	$26.75
Cracking the Coding Interview: 150 ...	› Gayle Laakmann McDowell	(182)	Paperback	$23.77
Java Soa Cookbook	› Eben Hewitt	(18)	Paperback	$31.50

Editorial Reviews

Review

A solid, valuable and easy-to-read work. -- *JavaRanch*



Chris Richardson is a developer, architect and mentor with over 20 years of experience. He runs a consulting company that jumpstarts new development projects and helps teams that are frustrated with enterprise Java become more productive and successful. Chris has been a technical leader at a variety of companies including Insignia Solutions and BEA Systems. Chris holds a MA & BA in Computer Science from the University of Cambridge in England. He lives in Oakland, CA.

Product Details

Paperback: 456 pages
Publisher: Manning Publications; 1 edition (January 30, 2006)
Language: English
ISBN-10: 1932394583
ISBN-13: 978-1932394580
Product Dimensions: 9.2 x 7.3 x 1.2 inches
Shipping Weight: 2.5 pounds (View shipping rates and policies)
Average Customer Review: ★★★★☆ (31 customer reviews)
Amazon Best Sellers Rank: #926,633 in Books (See Top 100 in Books)

Did we miss any relevant features for this product? **Tell us what we missed.**
Would you like to **update product info**, **give feedback on images**, or **tell us about a lower price**?

Customer Reviews

★★★★☆ (31)
4.4 out of 5 stars

5 star	20
4 star	8
3 star	0
2 star	2
1 star	1

See all 31 customer reviews

"The book is explains very good how to build enterprise apps using the pojo frameworks like spring, hibernate, jdo."
Ionut L. Ochian | 9 reviewers made a similar statement

"Once I started reading the book, it was hard for me to put it down."
B. S. Meera | 5 reviewers made a similar statement

"I like the IN ACTION series from Manning."
Berndt Hamboeck | 4 reviewers made a similar statement

Most Helpful Customer Reviews

8 of 8 people found the following review helpful
★★★★☆ **Get your POJO workin'** December 2, 2006
By Thing with a hook
Format: Paperback

This book covers the use of several lightweight frameworks for developing enterprise applications. If you have no clue at all about the issues involved in enterprise Java, I would not advise reading this yet. Despite being C#-based, Applying Domain-Driven Design and Patterns by Jimmy Nilsson would provide the gentle introduction required. On the other hand, if you've had previous experience with server side programming, and want to be brought up to speed quickly on how POJO-based frameworks can be used to replace EJB 2.x style development, this is right up your alley. If you've got used to computer books belying their dimensions with disappointingly little information, you'll be pleasantly surprised with PiA - it's packed with good content.

What's nice about this book is that it goes beyond the basics of the likes of Spring that most people have read several times already (e.g. explaining what dependency injection is) and actually shows how it obviates the need to run in an EJB container and do JNDI look ups. You don't just get to read about, e.g. lazy and eager loading, the author shows you how to use Hibernate and JDO to implement those strategies. That said, this book is not a replacement for documentation or specialised references, so it doesn't get too bogged down. Particularly helpful is that the author provides pros and cons for each of the different approaches he advocates, which helps put them into perspective.

The focus of the book is on using Object Relational Mapping tools, either Hibernate or JDO, in combination with Spring's dependency injection and AOP-based interceptors for transactions. There is also converage of the more procedural-based iBATIS, and using EJB3, although the author does not seem to be a big fan of the latter, despite it being an improvement on EJB2. Many of the persistence-related patterns in Martin Fowler's Patterns of Enterprise Application Architecture are covered here, including the concurrency patterns like pessimistic and optimistic locking. The author shows how to implement these patterns with the frameworks, often showing multiple ways of doing things. He's not afraid to highlight where one framework is lacking compared to another, which is refreshing.

As you can perhaps tell, the coverage is predominantly devoted to the persistence layer - there's not much here on the presentation layer, although there is some material on using servlets. If you're looking for lots of detail on how to hook your domain model up to, say, Struts, or one of the many other web frameworks, you won't find much here.

My only quibble with the book is that although the author pushes increased testability as a important benefit of freeing oneself from EJB containers (a good thing) and uses JUnit tests to illustrate how to develop a POJO-based application (another good thing), the tests use mock objects heavily. I hesitate to call that a bad thing, as clearly there's a whole bunch of people who are much cleverer than I using them productively, but here there's so much set up and setting of expectations, that the actual test is hard to spot, and the intention difficult to fathom. Your mileage may of course vary.

If you're neither an enterprise dummy nor expect, I wholeheartedly recommend this excellent book.

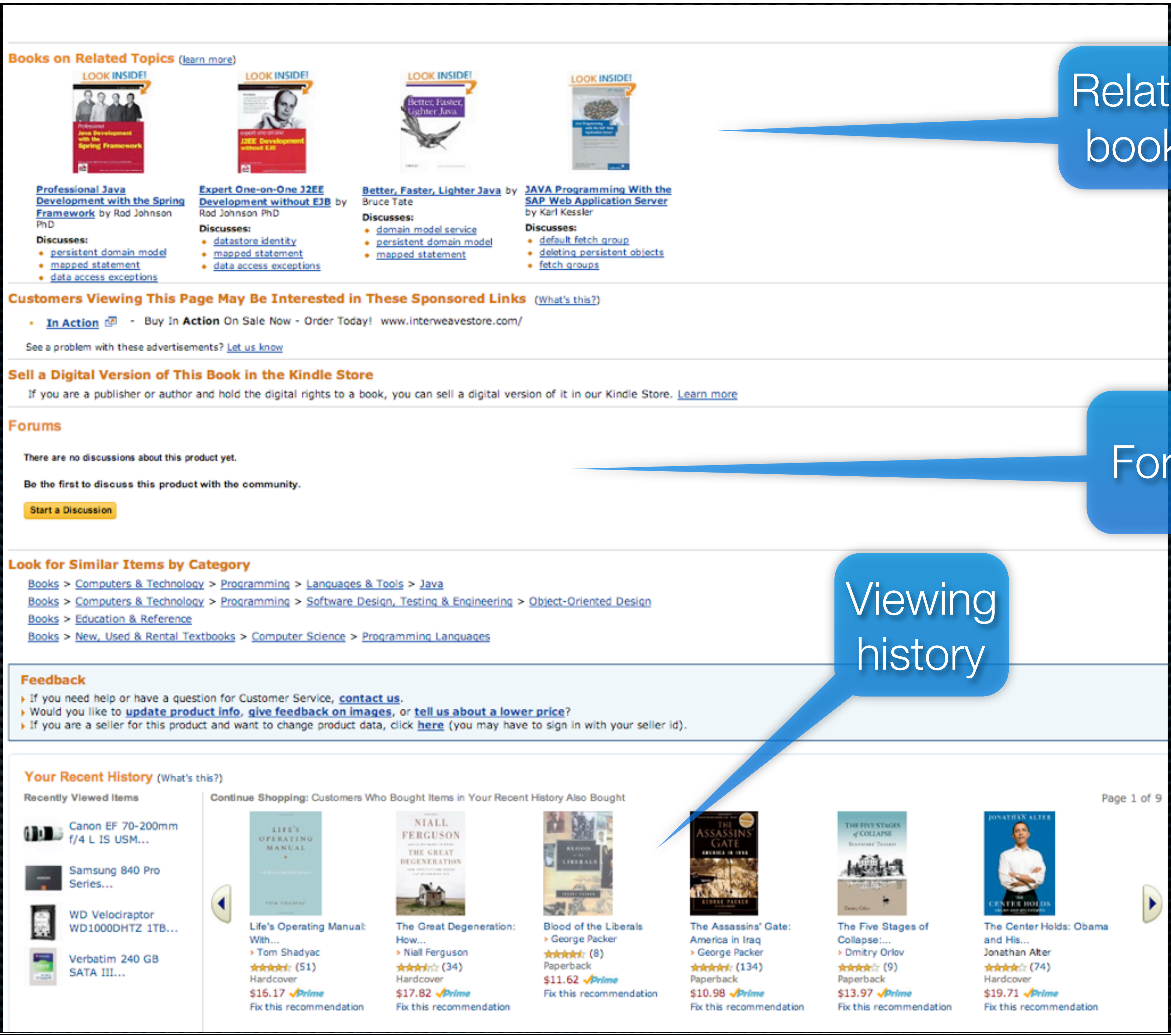
Master the basics of epidemiology with DR. GORDIS! SHOP NOW! ELSEVIER
Advertisement

Most Recent Customer Reviews

★★☆☆☆ **useless book about pojos in context of spring, ejb, hibernate and jdo**
the book lightly covers the use of pojo in spring, ejb, hibernate, and jdo. the coverage of each topic is like say 30-40%. Read more
Published 19 months ago by anonymous

★★★★★ **Learned "Back-End Web Programming" From This Book**
This book is a rare find. It is completely practical, teaching you what you need to know to use Spring and Hibernate (or JDO).
Read more
Published 23 months ago by doodaddy

★★★★★ **Great practical resource**
Despite the fact that it was written a few years ago, it is no less valuable today in helping developers understand how to create an



Books on Related Topics (learn more)

Professional Java Development with the Spring Framework by Rod Johnson PhD
Discusses:
- persistent domain model
- mapped statement
- data access exceptions

Expert One-on-One J2EE Development without EJB by Rod Johnson PhD
Discusses:
- datastore identity
- mapped statement
- data access exceptions

Better, Faster, Lighter Java by Bruce Tate
Discusses:
- domain model service
- persistent domain model
- mapped statement

JAVA Programming With the SAP Web Application Server by Karl Kessler
Discusses:
- default fetch group
- deleting persistent objects
- fetch groups

Customers Viewing This Page May Be Interested in These Sponsored Links (What's this?)

- In Action - Buy In **Action** On Sale Now - Order Today! www.interweavestore.com/

See a problem with these advertisements? Let us know

Sell a Digital Version of This Book in the Kindle Store

If you are a publisher or author and hold the digital rights to a book, you can sell a digital version of it in our Kindle Store. Learn more

Forums

There are no discussions about this product yet.

Be the first to discuss this product with the community.

Start a Discussion

Look for Similar Items by Category

Books > Computers & Technology > Programming > Languages & Tools > Java
Books > Computers & Technology > Programming > Software Design, Testing & Engineering > Object-Oriented Design
Books > Education & Reference
Books > New, Used & Rental Textbooks > Computer Science > Programming Languages

Feedback

- If you need help or have a question for Customer Service, **contact us**.
- Would you like to **update product info**, **give feedback on images**, or **tell us about a lower price**?
- If you are a seller for this product and want to change product data, click **here** (you may have to sign in with your seller id).

Your Recent History (What's this?)

Recently Viewed Items

- Canon EF 70-200mm f/4 L IS USM...
- Samsung 840 Pro Series...
- WD Velociraptor WD1000DHTZ 1TB...
- Verbatim 240 GB SATA III...

Continue Shopping: Customers Who Bought Items in Your Recent History Also Bought

Page 1 of 9

Life's Operating Manual: With... › Tom Shadyac (51) Hardcover $16.17 Prime Fix this recommendation

The Great Degeneration: How... › Niall Ferguson (34) Hardcover $17.82 Prime Fix this recommendation

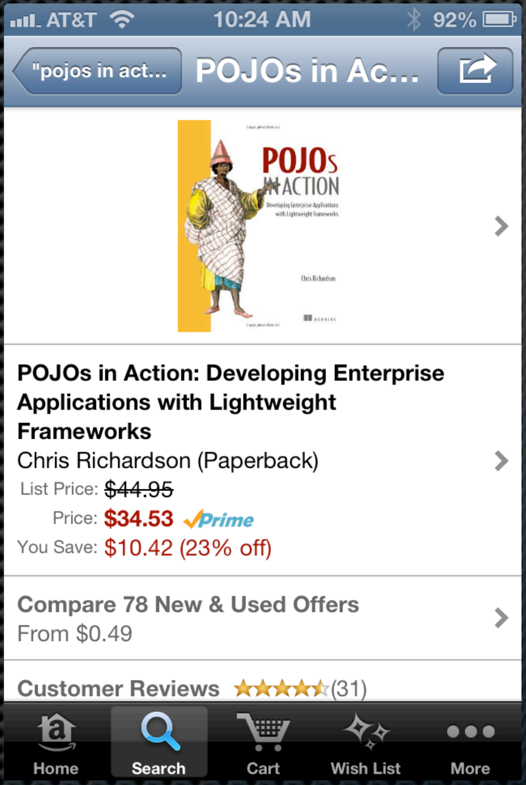
Blood of the Liberals › George Packer (8) Paperback $11.62 Prime Fix this recommendation

The Assassins' Gate: America in Iraq › George Packer (134) Paperback $10.98 Prime Fix this recommendation

The Five Stages of Collapse:... › Dmitry Orlov (9) Paperback $13.97 Prime Fix this recommendation

The Center Holds: Obama and His... Jonathan Alter (74) Hardcover $19.71 Prime Fix this recommendation

+ mobile apps



AT&T 10:24 AM 92%

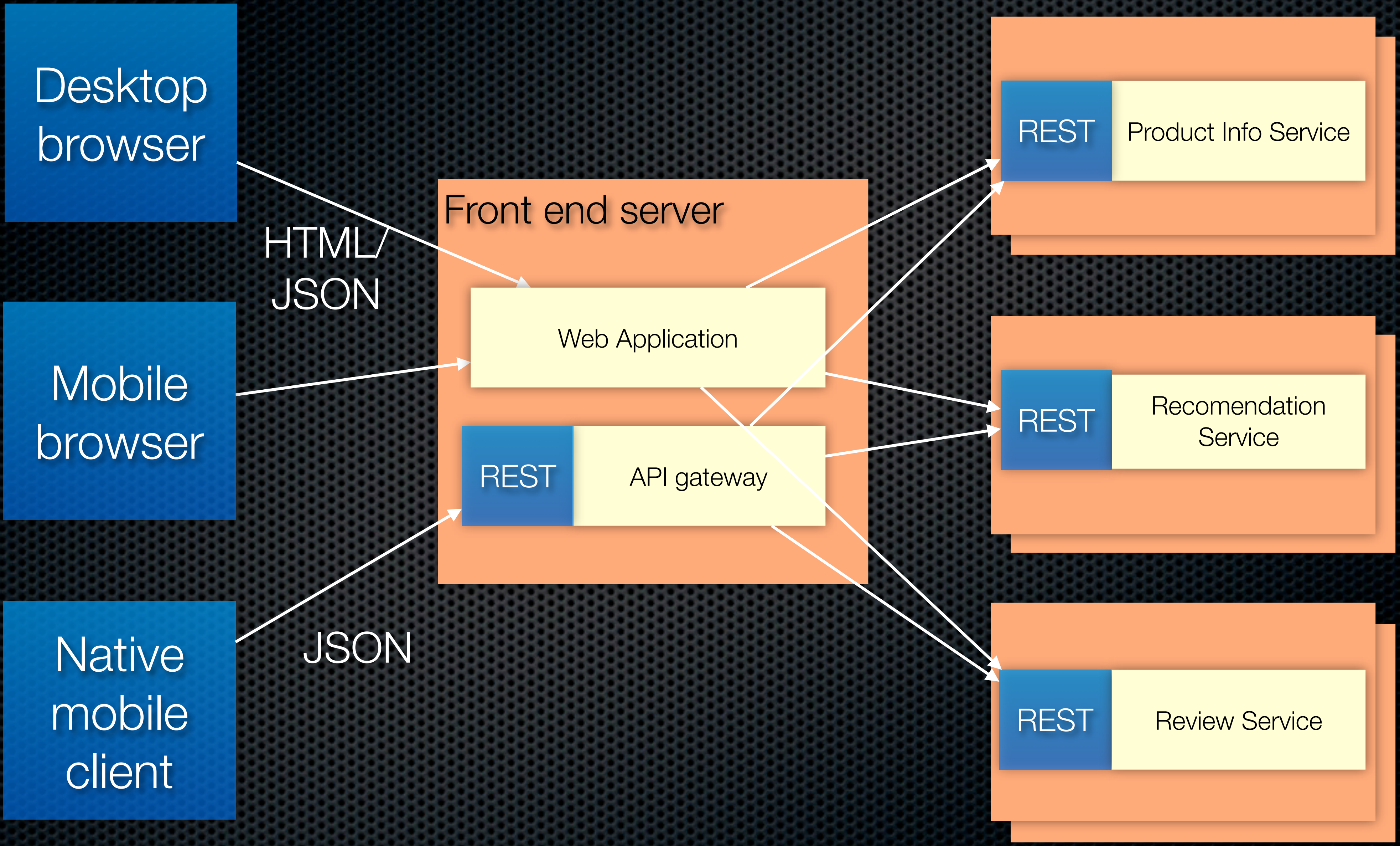
"pojos in act... POJOs in Ac...

POJOs in Action: Developing Enterprise Applications with Lightweight Frameworks
Chris Richardson (Paperback)
List Price: ~~$44.95~~
Price: $34.53 Prime
You Save: $10.42 (23% off)

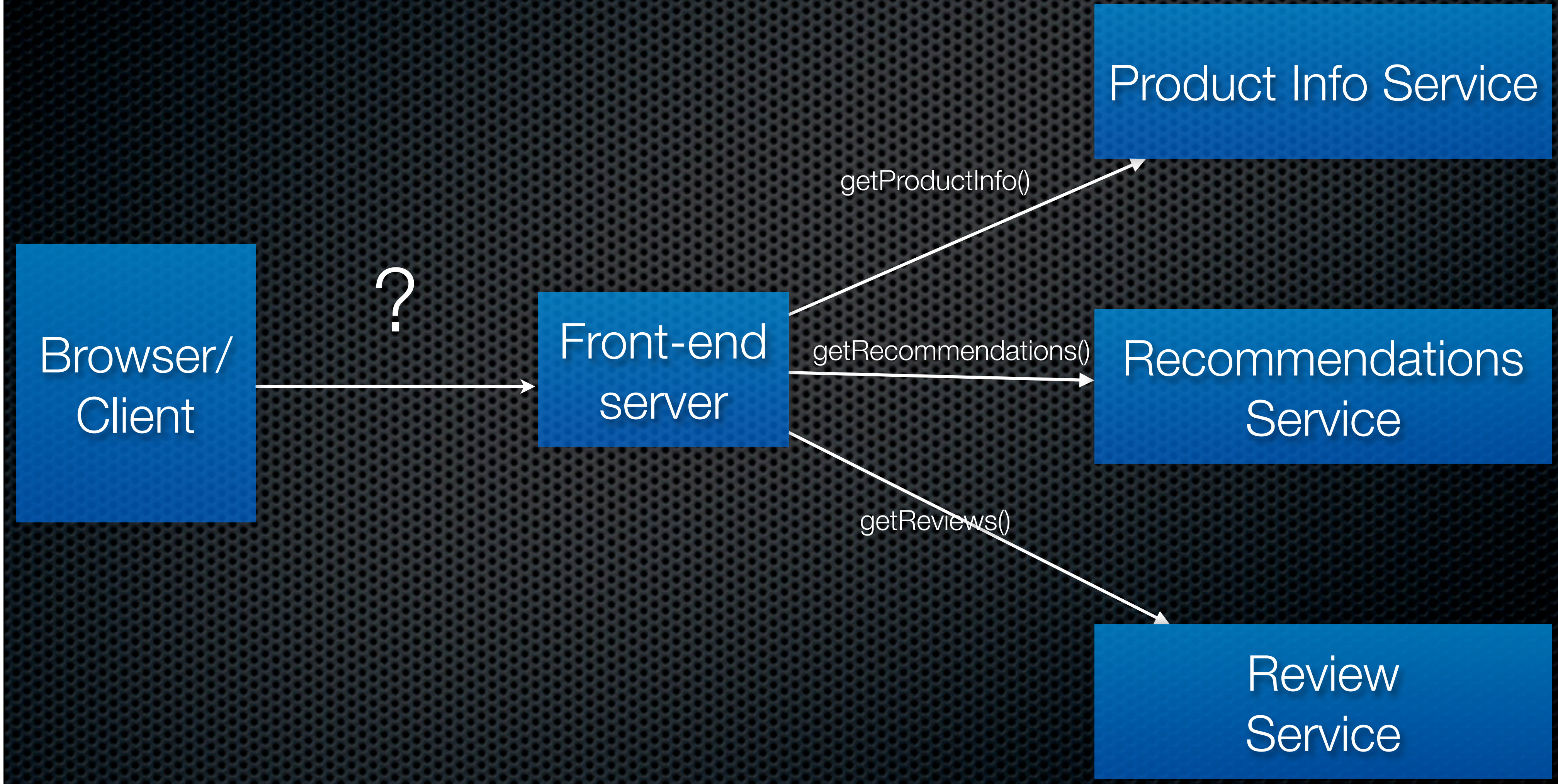
Compare 78 New & Used Offers
From $0.49

Customer Reviews (31)

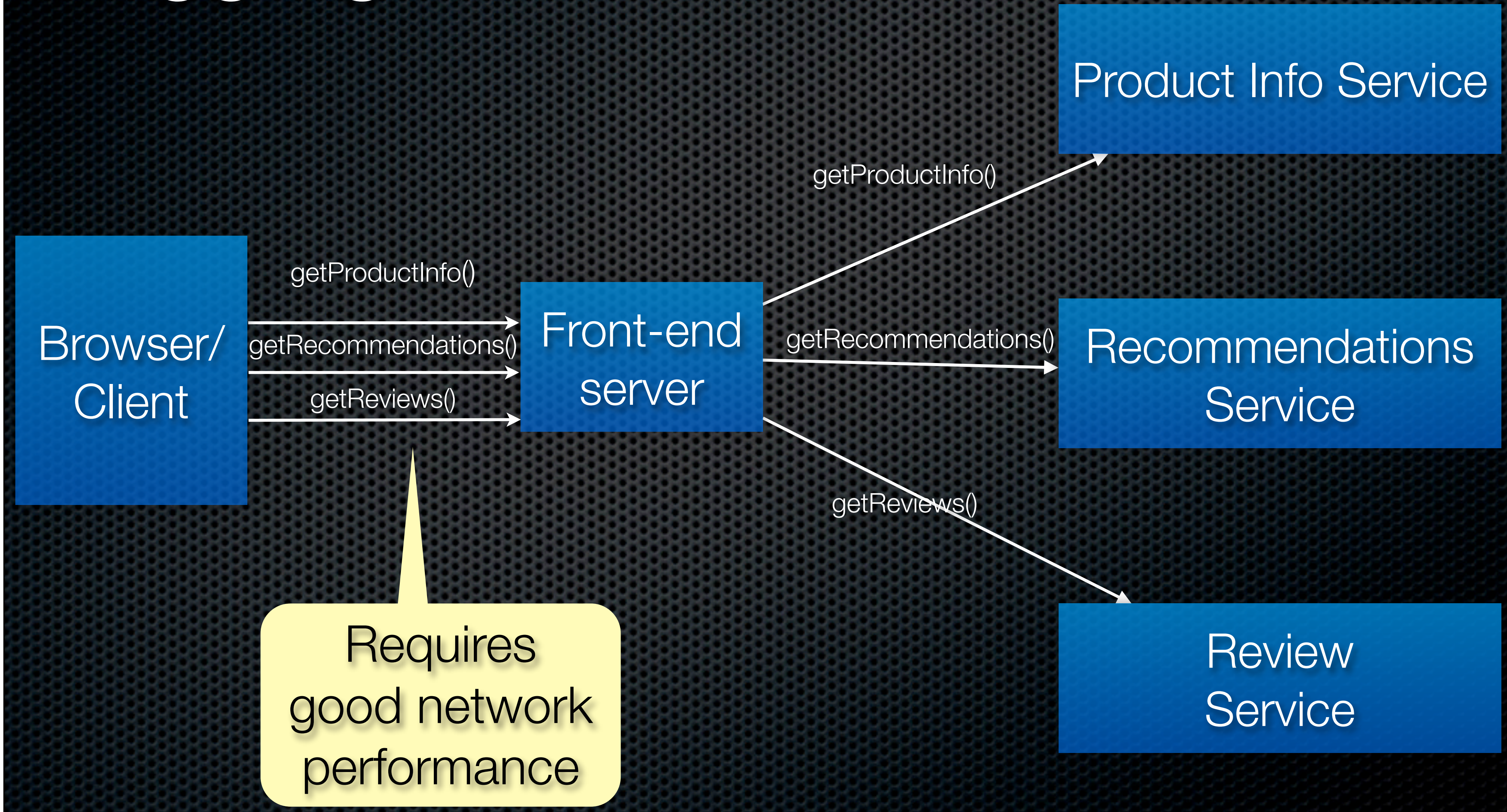
Application architecture



How does the client get product details?

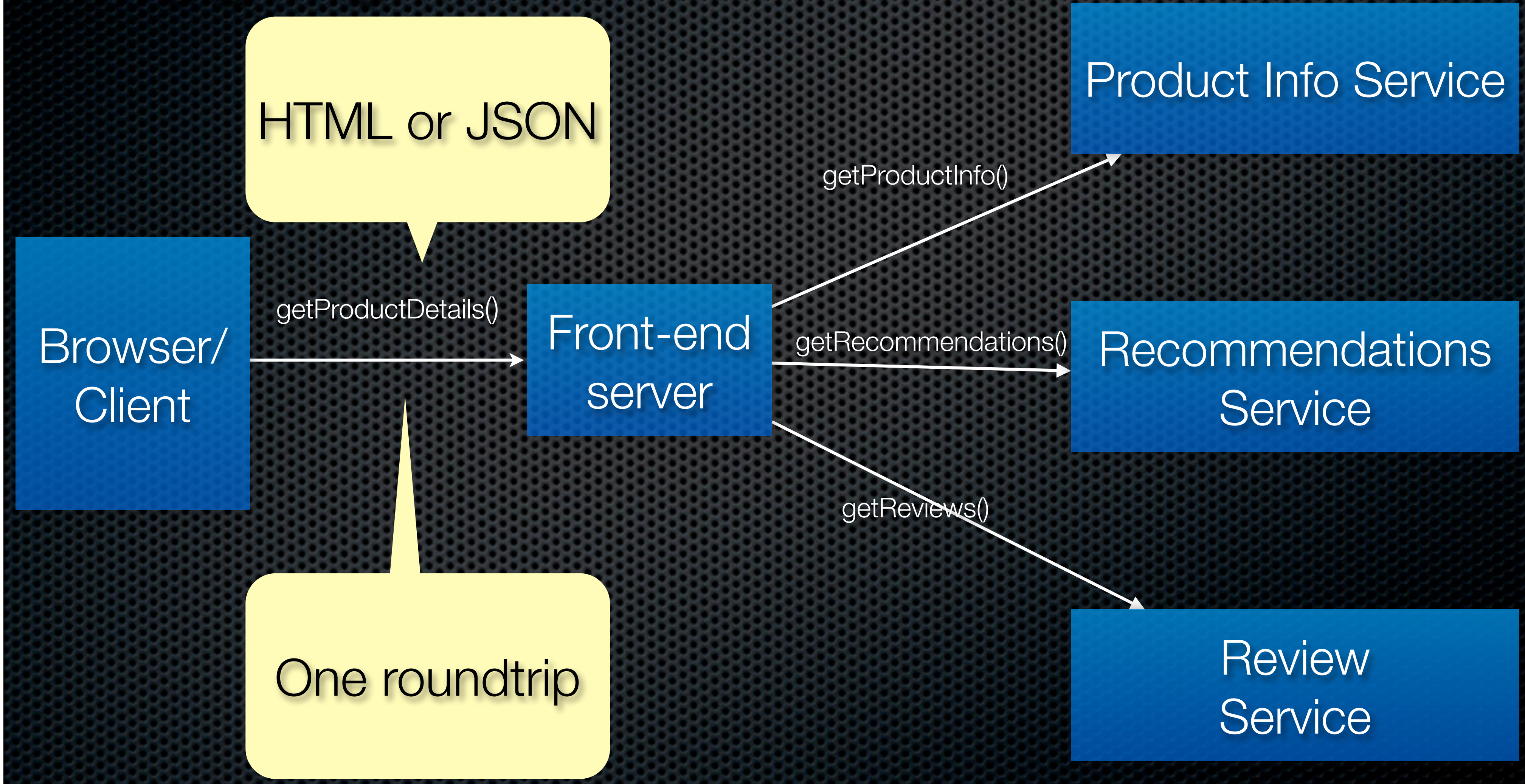


Product details - client-side aggregation



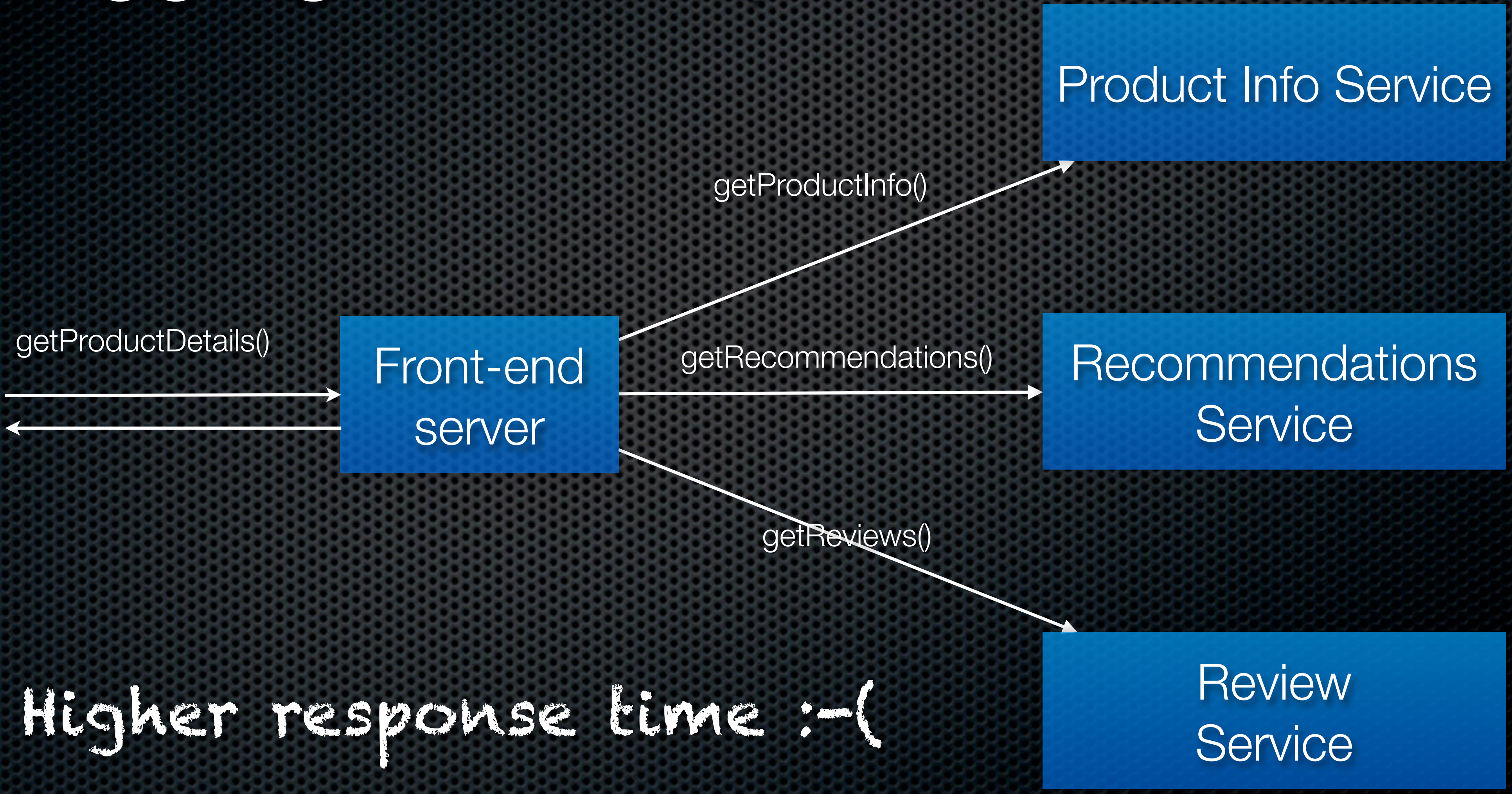
@crichardson

Product details - server-side aggregation



@crichardson

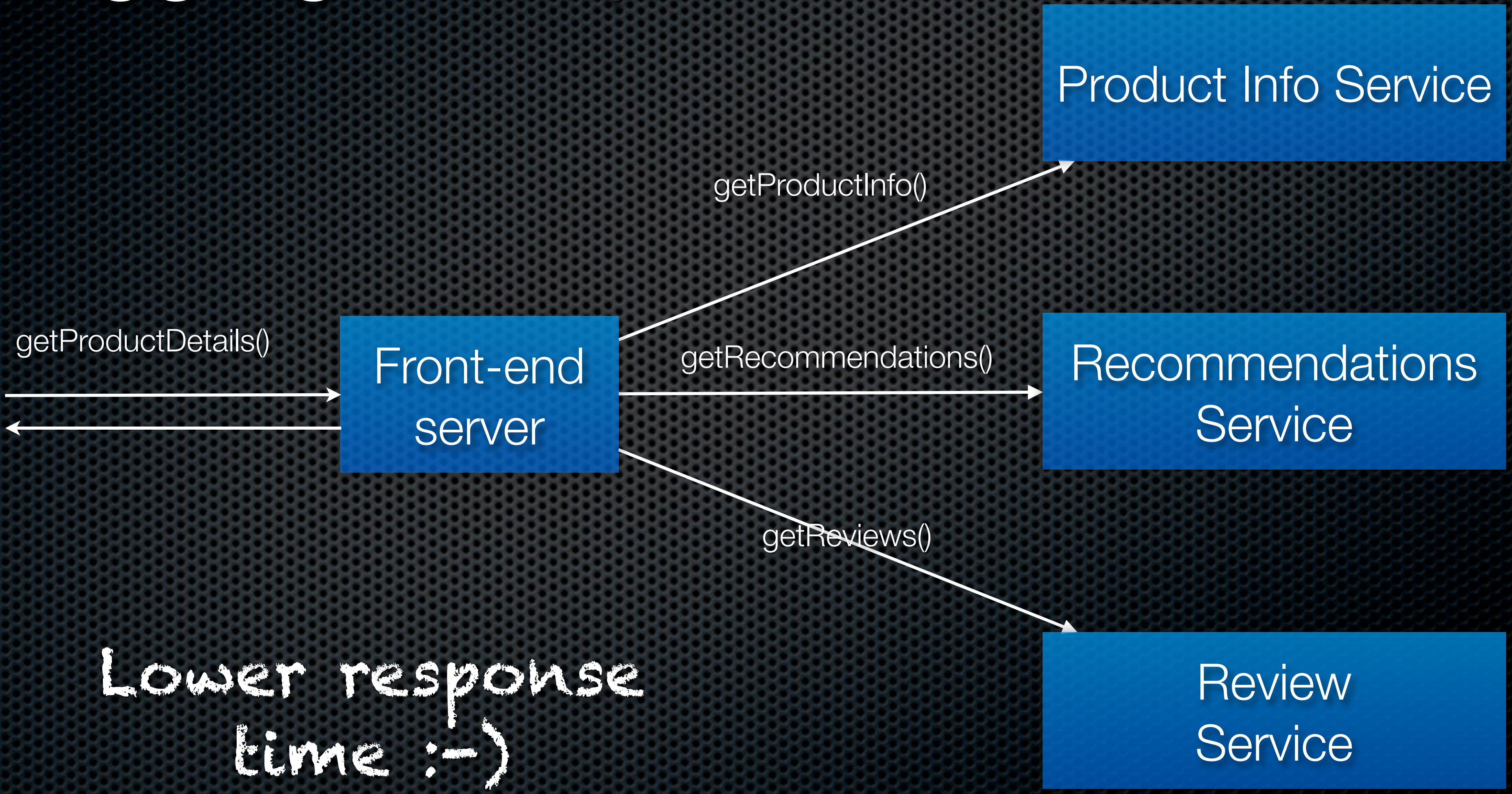
Product details - server-side aggregation: sequentially



Higher response time :-(

@crichardson

Product details - server-side aggregation: parallel



Lower response time :-)

@crichardson

Implementing a concurrent REST client

- Thread-pool based approach
  - executorService.submit(new Callable(...))
  - Simpler but less scalable - lots of idle threads consuming memory
- Event-driven approach
  - NIO with completion callbacks
  - More complex but more scalable

And it must handle partial failures

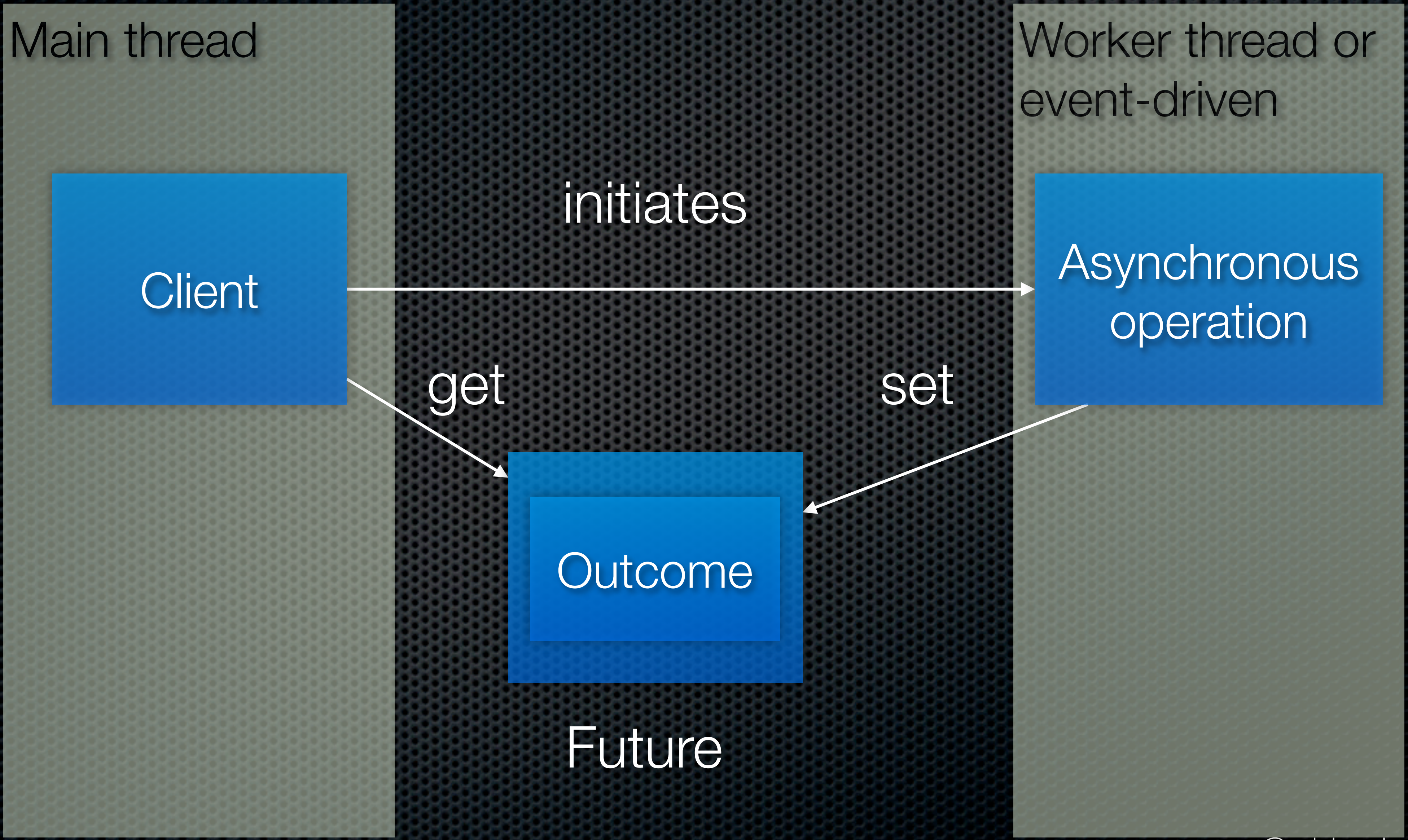
Agenda

- The need for concurrency
- Simplifying concurrent code with Futures
- Consuming asynchronous streams with Reactive Extensions

Futures are a great concurrency abstraction

http://en.wikipedia.org/wiki/Futures_and_promises

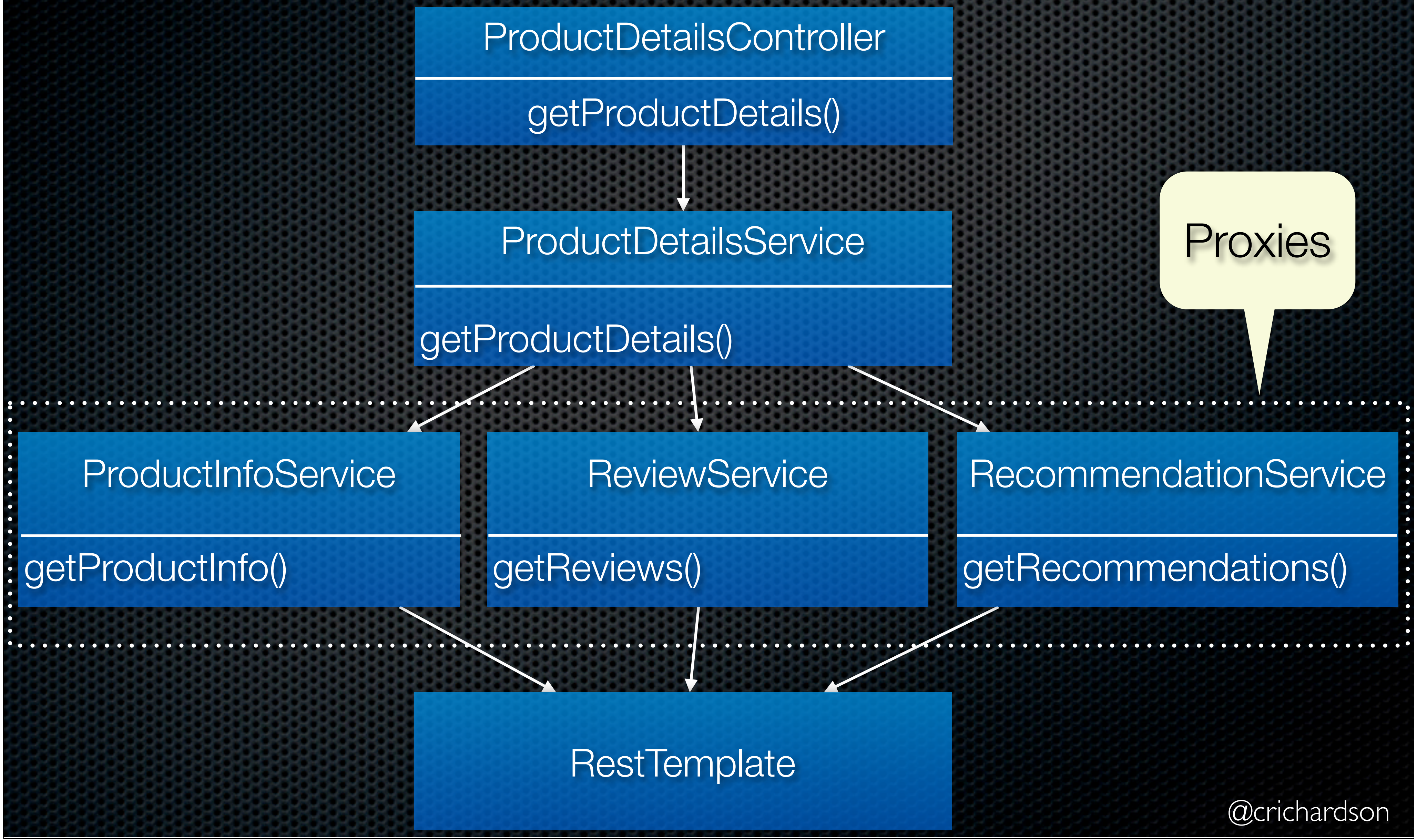
How futures work



Benefits

- Simple way for two concurrent activities to communicate safely
- Abstraction:
  - Client does not know how the asynchronous operation is implemented
- Easy to implement scatter/gather:
  - Scatter: Client can invoke multiple asynchronous operations and gets a Future for each one.
  - Gather: Get values from the futures

Front-end server design: handling GetProductDetails request



REST client using Spring @Async

```
trait ProductInfoService {
  def getProductInfo(productId: Long):
               java.util.concurrent.Future[ProductInfo]
}
```

```
@Component
class ProductInfoServiceImpl extends ProducInfoService {

  val restTemplate : RestTemplate = ...

  @Async
  def getProductInfo(productId: Long) = {
    new AsyncResult(restTemplate.getForObject(....)...)
  }

}
```

Execute asynchronously in thread pool

A fulfilled Future

@crichardson

ProductDetailsService

```
@Component
class ProductDetailsService
       @Autowired()(productInfoService: ProductInfoService,
                            reviewService: ReviewService,
                            recommendationService: RecommendationService) {

  def getProductDetails(productId: Long): ProductDetails = {
    val productInfoFuture = productInfoService.getProductInfo(productId)
    val recommendationsFuture =
              recommendationService.getRecommendations(productId)
    val reviewsFuture = reviewService.getReviews(productId)
    val productInfo = productInfoFuture.get(300, TimeUnit.MILLISECONDS)
    val recommendations =
         recommendationsFuture.get(10, TimeUnit.MILLISECONDS)
    val reviews = reviewsFuture.get(10, TimeUnit.MILLISECONDS)

    ProductDetails(productInfo, recommendations, reviews)
  }

}
```

ProductController

```
@Controller
class ProductController
    @Autowired()(productDetailsService : ProductDetailsService)
{

  @RequestMapping(Array("/productdetails/{productId}"))
  @ResponseBody
  def productDetails(@PathVariable productId: Long) =
     productDetailsService.getProductDetails(productId)
```

Not bad but...

```
class ProductDetailsService
  def getProductDetails(productId: Long): ProductDetails = {

    val productInfo =
       productInfoFuture.get(300, TimeUnit.MILLISECONDS)
```

Gathering blocks Tomcat thread until all Futures complete

Not so scalable :-(

... and also...

- Java Futures work well for a single-level of asynchronous execution

BUT

- #fail for more complex, scalable scenarios
- Difficult to compose and coordinate multiple concurrent operations
- See this blog post for more details:

http://techblog.netflix.com/2013/02/rxjava-netflix-api.html

Better: Futures with callbacks ⇒ no blocking!

```
def asyncSquare(x : Int)
       : Future[Int] = ... x * x...

val f = asyncSquare(25)

f onSuccess {
  case x : Int => println(x)
}
f onFailure {
  case e : Exception => println("exception thrown")
}
```

Partial function applied to successful outcome

Applied to failed outcome

Guava ListenableFutures, Spring 4 ListenableFuture
Java 8 CompletableFuture, **Scala Futures**

@crichardson

But
callback-based scatter/gather
⇒
Messy, tangled code
(aka. callback hell)

Composable futures hide the mess

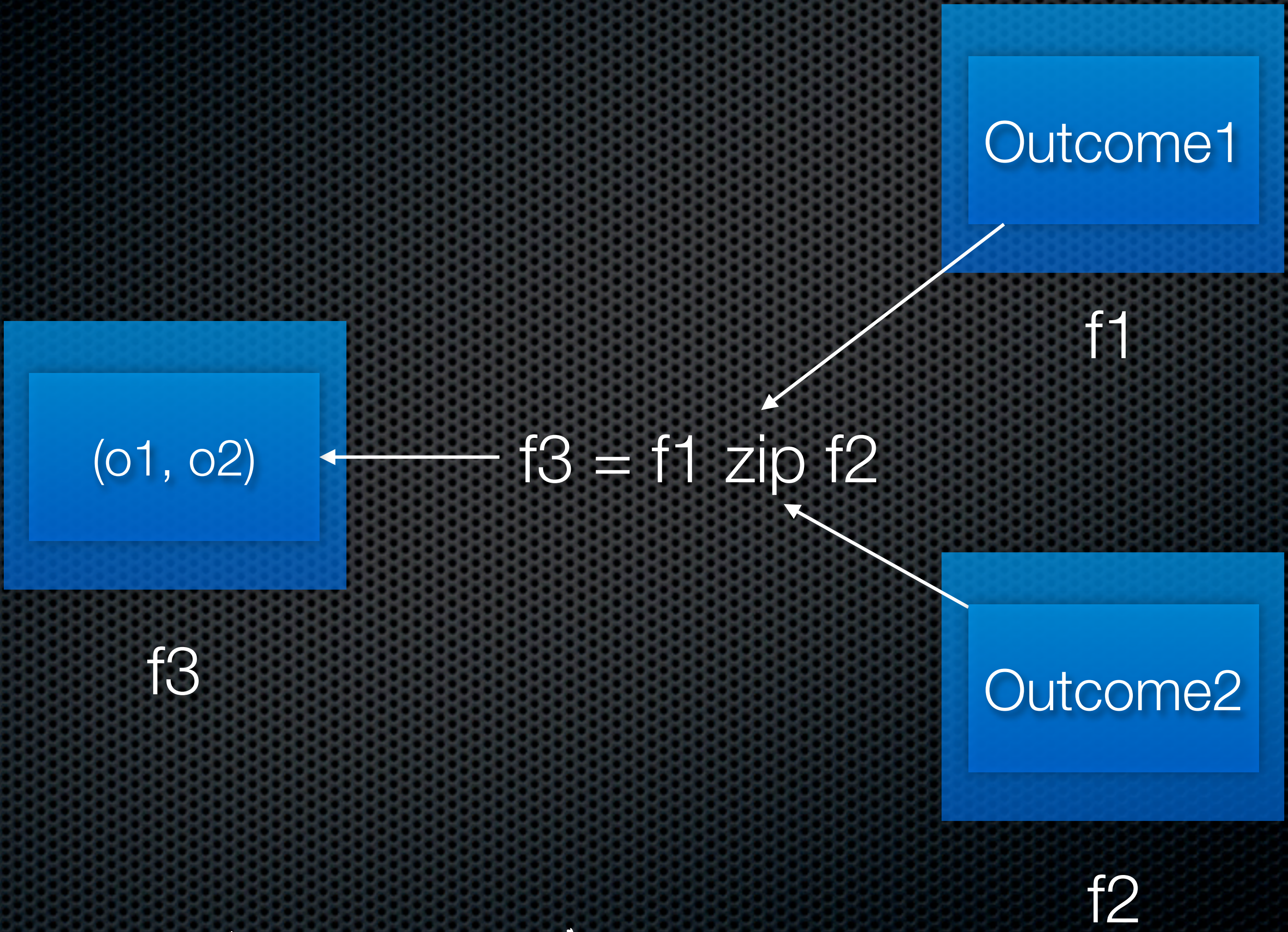
Combines two futures

```
val fzip = asyncSquare(5) zip asyncSquare(7)
assertEquals((25, 49), Await.result(fzip, 1 second))


val fseq = Future.sequence((1 to 5).map { x =>
              asyncSquare(x)
})

assertEquals(List(1, 4, 9, 16, 25),
             Await.result(fseq, 1 second))
```

Transforms list of futures to a future

Scala, Java 8 CompletableFuture (partially)

zip() is asynchronous



Implemented using callbacks

Transforming futures

```
def asyncPlus(x : Int, y : Int) = ... x + y ...

val future2 = asyncPlus(4, 5).map{ _ * 3 }

assertEquals(27, Await.result(future2, 1 second))
```

Asynchronously transforms future

Scala, Java 8 CompletableFuture (partially)

Chaining asynchronous operations

Calls asyncSquare() with the eventual outcome of asyncPlus()

```
val f2 = asyncPlus(5, 8).flatMap { x => asyncSquare(x) }

assertEquals(169, Await.result(f2, 1 second))
```

Scala, Java 8 CompletableFuture (partially)

Scala futures are Monads

Two calls execute in parallel

```
(asyncPlus(3, 5) zip asyncSquare(5))
  .flatMap {
      case (a, b) =>
       asyncPlus(a, b) map { _ * 2 }
  }

result onSuccess { .... }
```

And then invokes asyncPlus()

x 2

Rewrite using 'for'

Scala futures are Monads

```
val result = for {
  (a, b) <- asyncPlus(3, 5) zip asyncSquare(5);
  c <- asyncPlus(a, b)
} yield c * 2

result onSuccess { .... }
```

Two calls execute in parallel

And then invokes asyncPlus()

x 2

'for' is shorthand for map() and flatMap()

ProductInfoService: using Scala Futures

```
import scala.concurrent.Future

@Component
class ProductInfoService {

  def getProductInfo(productId: Long): Future[ProductInfo]
    = {
    Future { restTemplate.getForObject(....) }
  }

}
```

Scala Future

Executed in a threaded pool

@crichardson

ProductDetailsService: using Scala Futures

Return a Scala Future

```
class ProductDetailsService ...


  def getProductDetails(productId: Long) : Future[ProductDetails] = {
    val productInfoFuture = productInfoService.getProductInfo(productId)
    val recommendationsFuture =
                  recommendationService.getRecommendations(productId)
    val reviewsFuture = reviewService.getReviews(productId)

    for (((productInfo, recommendations), reviews) <-
             productInfoFuture zip recommendationsFuture zip reviewsFuture)
          yield ProductDetails(productInfo, recommendations, reviews)


  }
}
```

Gathers data without blocking

@crichardson

Async ProductController: using Spring MVC DeferredResult

```
@Controller
class ProductController ... {

  @RequestMapping(Array("/productdetails/{productId}"))
  @ResponseBody
  def productDetails(@PathVariable productId: Long)
                  : DeferredResult[ProductDetails] = {
    val productDetails =
          productDetailsService.getProductDetails(productId)
    val result = new DeferredResult[ProductDetails]
     productDetails onSuccess {
       case r => result.setResult(r)
     }
     productDetails onFailure {
       case t => result.setErrorResult(t)
     }
    result
  }
```

Spring MVC
DeferredResult
≅
Future

Convert Scala Future
to
DeferredResult

Servlet layer is asynchronous

BUT

the backend uses thread pools

⇒

Need event-driven REST client

Spring AsyncRestTemplate

- New in Spring 4
- Mirrors RestTemplate
- Can use HttpComponents NIO-based AsyncHttpClient
- Methods return a ListenableFuture
  - JDK 7 Future + callback methods

Yet another "Future"!

ProductInfoService: using the AsyncRestTemplate

```
class ProductInfoService {
 val asyncRestTemplate = new AsyncRestTemplate(
        new HttpComponentsAsyncClientHttpRequestFactory())


 override def getProductInfo(productId: Long) = {

  val listenableFuture =
   asyncRestTemplate.getForEntity("{baseUrl}/productinfo/{productId}",
                    classOf[ProductInfo],
                    baseUrl, productId)

   toScalaFuture(listenableFuture).map { _.getBody  }
 }
```

Convert to Scala Future and get entity

http://hc.apache.org/httpcomponents-asyncclient-dev/

@crichardson

Converting ListenableFuture to Scala Future

```
def toScalaFuture[T](lf : ListenableFuture[T]) :
                                        Future[T] = {
    val p = promise[T]()

    lf.addCallback(new ListenableFutureCallback[T] {
      def onSuccess(result: T) { p.success(result)}
      def onFailure(t: Throwable) { p.failure(t)  }
    })
    p.future
  }
```

Creates a promise = producer API

Propagate outcome to promise

Return future

@crichardson

Now everything is non-blocking :-)

We have achieved scaling Nirvana

WT*#*# is my code doing?

- Operations initiated in one thread but fail in another
  - Lack of a full stack trace can make debugging difficult
  - Inherent problem of async/event driven programming
- Futures make it very easy to forget to handle errors
  - `someFuture.foreach { handleTheHappyPath }`
  - Error is quietly ignored: similar to an empty catch {} block

Agenda

- The need for concurrency
- Simplifying concurrent code with Futures
- Consuming asynchronous streams with Reactive Extensions

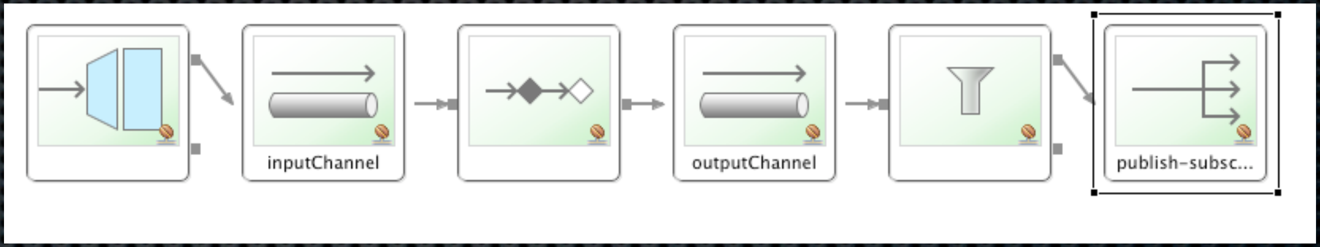
Let’s imagine you have a stream of trades
and
you need to calculate the 15 minute rolling average price of each stock

Where is the high-level abstraction that simplifies solving this problem?

Future[List[T]]

Not applicable to infinite streams

Pipes and Filters
e.g. Spring Integration
+
Complex event processing (CEP)



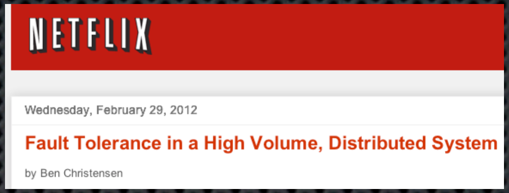
Not bad but tends to be an external DSL, heavy weight, statically defined, ...

Introducing Reactive Extensions (Rx)

The Reactive Extensions (Rx) is a library for composing asynchronous and event-based programs using observable sequences and LINQ-style query operators. Using Rx, developers **represent asynchronous data streams with Observables** , **query asynchronous data streams using LINQ operators** , and .....

https://rx.codeplex.com/

About RxJava


NETFLIX

Wednesday, February 29, 2012

Fault Tolerance in a High Volume, Distributed System

by Ben Christensen

- Reactive Extensions (Rx) for the JVM
- Original motivation for Netflix was to provide rich Futures
- Implemented in Java
- Adaptors for Scala, Groovy and Clojure

https://github.com/Netflix/RxJava

RxJava core concepts

An asynchronous stream of items

```
trait Observable[T] {
  def subscribe(observer : Observer[T]) : Subscription
  ...
}
```

Notifies

```
trait Observer[T] {
  def onNext(value : T)
  def onCompleted()
  def onError(e : Throwable)
}
```

Used to unsubscribe

Comparing Observable to...

- Observer pattern - similar but adds
  - Observer.onComplete()
  - Observer.onError()
- Iterator pattern - mirror image
  - Push rather than pull
- Future - similar but
  - Represents a stream of **multiple** values

@crichardson

So what?

Fun with observables

```
val oneItem = Observable.items(-1L)

val every10Seconds = Observable.interval(10 seconds)

val ticker = oneItem ++ every10Seconds

val subscription = ticker.subscribe ( new Observer[Long] {
  override def onNext(value: Long) = {  println("value=" + value)    }
})
...
subscription.unsubscribe()
```

-1	0	1	...
t=0	t=10	t=20	...

Creating observables

```
Observable.create({ observer: Observer[T] =>
    ...
    observer.onNext(...)
    ...
    observer.onCompleted()
    ...
    observer.onError(...)
    ...
   Subscription{ .... }
   }
 })
```

Function called when Observer subscribes

Called when observer unsubscribes

Creating observables from Scala Futures

Query AWS

```
def getTableStatus(tableName: String) = {
  val future = dynamoDbClient.describeTable(new DescribeTableRequest(tableName))

  Observable.create({ observer.onNext: Observer[DynamoDbStatus] =>
    future.onComplete {
      case Success(response) =>
        observer.onNext(DynamoDbStatus(response.getTable.getTableStatus))
        observer.onCompleted()

      case Failure(t: ResourceNotFoundException) =>
        observer.onNext(DynamoDbStatus("NOT_FOUND"))
        observer.onCompleted()

      case Failure(someError) =>
        observer.onError(someError)

    }
    Subscription({})
  })
}
```

Propagate outcome

Transforming observables

```
val tableStatus = ticker.flatMap { i =>
    logger.info("{}th describe table", i + 1)
    getTableStatus(name)
}
```

Status1	Status2	Status3	...
t=0	t=10	t=20	...

+ Usual collection methods: map(), filter(), take(), drop(), ...

Back to the stream of Trades averaging example...

Calculating averages

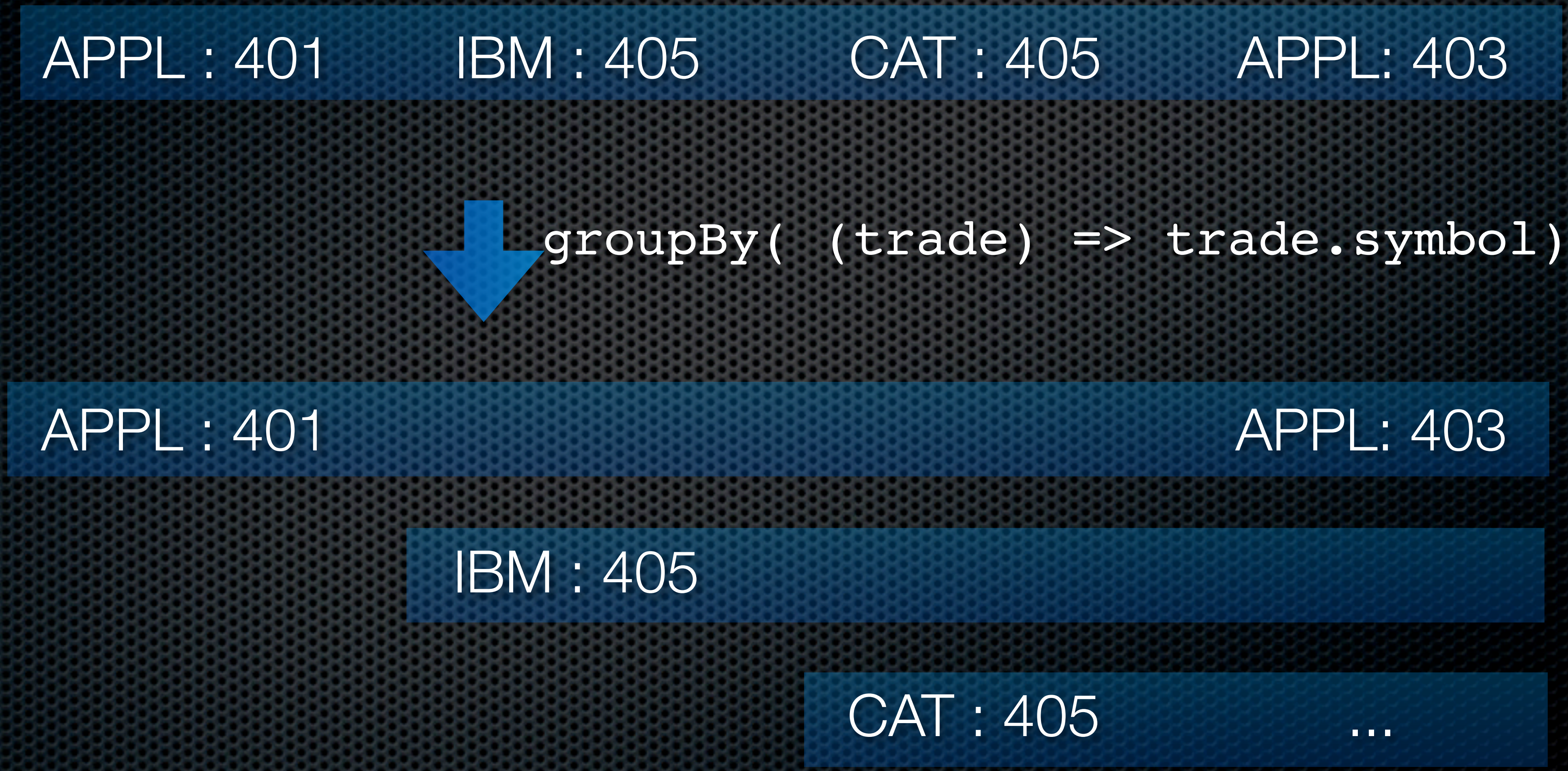
```
class AverageTradePriceCalculator {

  def calculateAverages(trades: Observable[Trade]):
      Observable[AveragePrice] = {
    ...
  }
```

```
case class Trade(
 symbol : String,
 price : Double,
 ...
)
```

```
case class AveragePrice(
  symbol : String,
  price : Double,
  ...)
```

Using groupBy()

```
Observable[Trade]
```

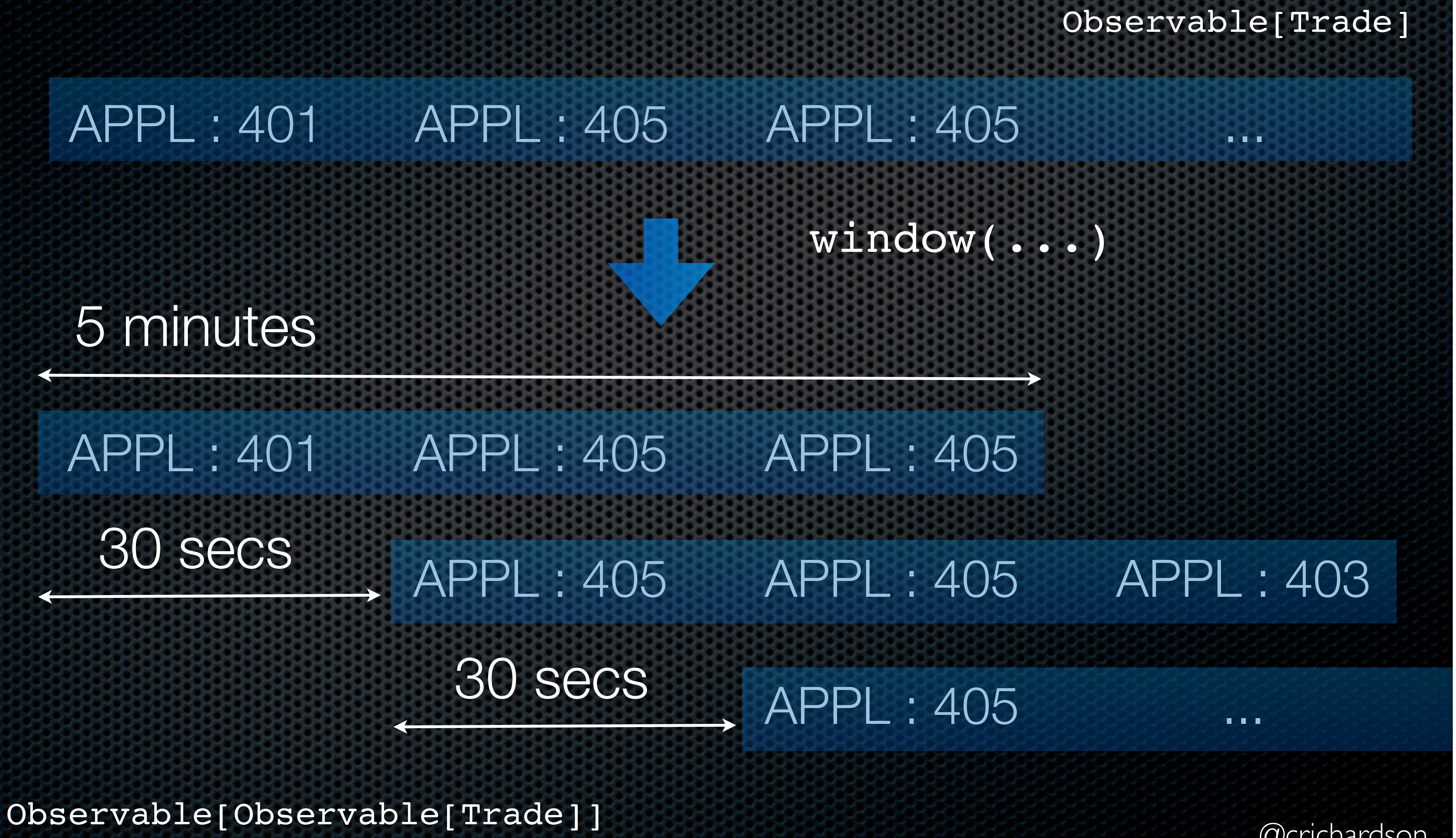


```
groupBy( (trade) => trade.symbol)
```

```
Observable[GroupedObservable[String, Trade]]
```

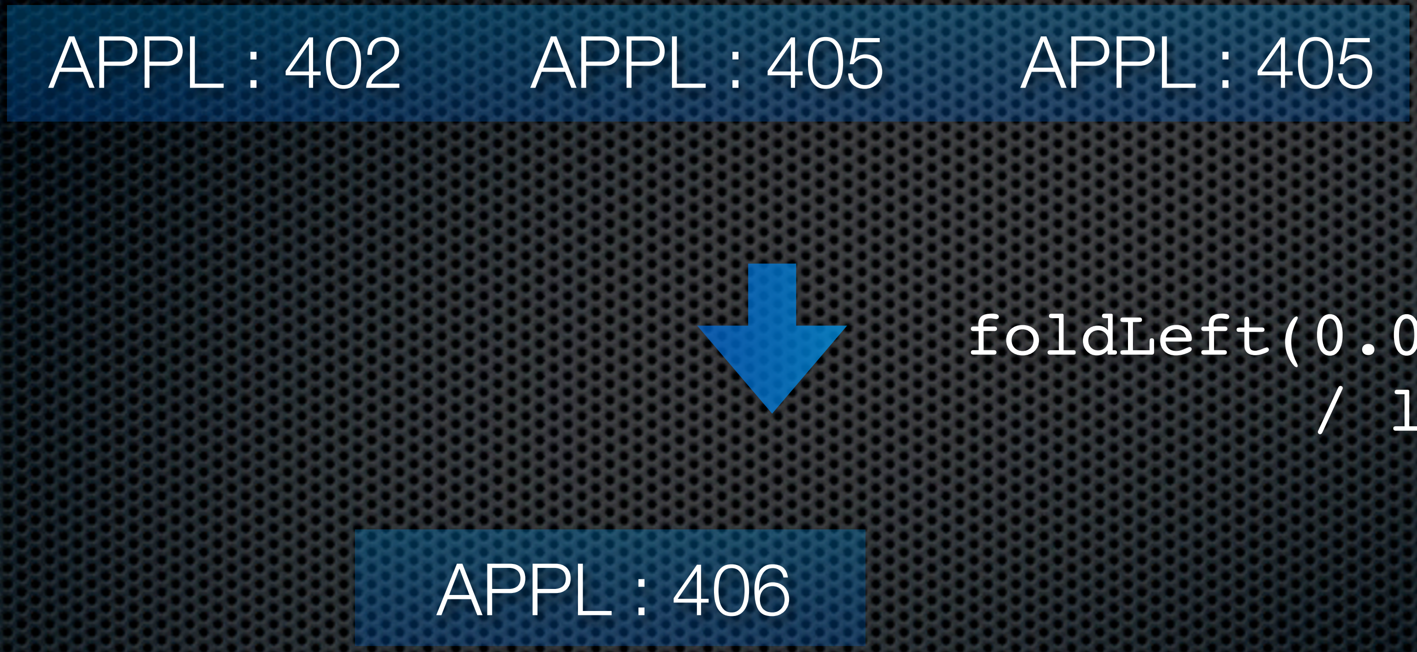
@crichardson

Using window()



Using foldLeft()

`Observable[Trade]`

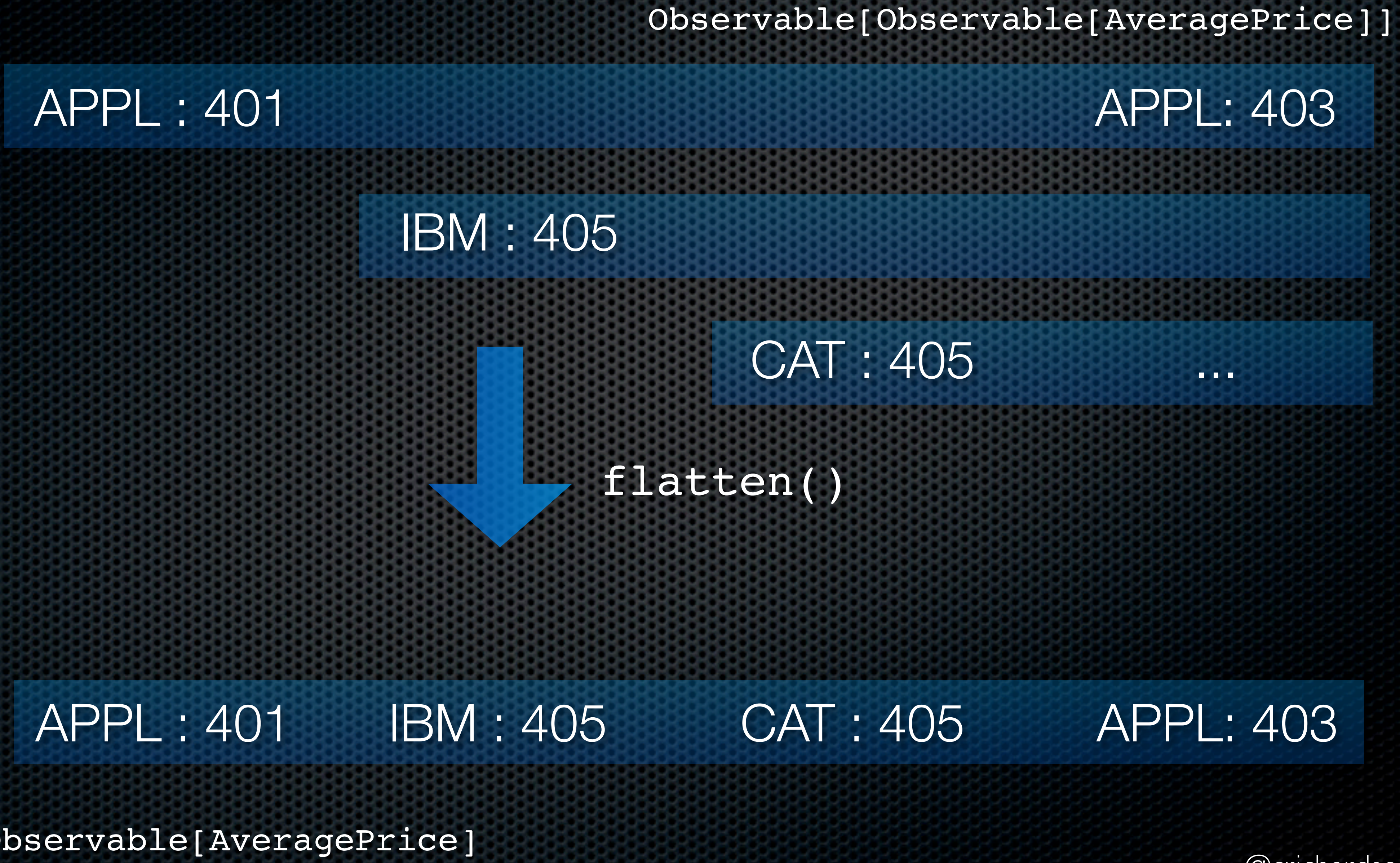


```
foldLeft(0.0)(_ + _.price)
        / length
```

`Observable[AveragePrice]`

Singleton

Using flatten()



Calculating average prices

```
def calculateAverages(trades: Observable[Trade]): Observable[AveragePrice] = {

  trades.groupBy(_.symbol).map { symbolAndTrades =>
    val (symbol, tradesForSymbol) = symbolAndTrades
      val openingEverySecond =
           Observable.items(-1L) ++ Observable.interval(1 seconds)
      def closingAfterSixSeconds(opening: Any) =
          Observable.interval(6 seconds).take(1)

    tradesForSymbol.window(...).map {
      windowOfTradesForSymbol =>
        windowOfTradesForSymbol.fold((0.0, 0, List[Double]())) { (soFar, trade) =>
           val (sum, count, prices) = soFar
          (sum + trade.price, count + 1, trade.price +: prices)
        } map { x =>
            val (sum, length, prices) = x
            AveragePrice(symbol, sum / length, prices)
        }
    }.flatten
  }.flatten
}
```

@crichardson

Summary

- Consuming web services asynchronously is essential
- Scala-style Futures are a powerful concurrency abstraction
- Rx Observables
    - even more powerful
    - unifying abstraction for a wide variety of use cases

@crichardson chris@chrisrichardson.net

Questions?

http://plainoldobjects.com