CS 575: Software Architecture & Design

Course Overview

About The Instructor

Academic Background

- **B.S., M.S., and Ph.D. in Computer Science**
- **M.E. in Computer and Telecommunication Engineering**
- **Involved with CS Department Teaching, Research and Industry collaboration since 1997**
- **Ph.D. work was is Software Architecture Recovery**

Industry Background

- **Distinguished Engineer, Chief Architect at Cigna (Business, Web, Mobile, SOA)**
- **Founder of Integrated SystemWare**

Contact Information

- **Email: bsm23@drexel.edu**
- **Office Hours: Generally before class, but by appointment**

Course Logistics

- We will be using Blackboard for assignments, deadlines, and content distribution
- I'm in the process of linking to where all up-to-date materials will be hosted on my GitHub course page: https://github.com/ArchitectingSoftware/CS575-SoftwareDesign
- Our course TA is Tri Le, tnl34@drexel.edu – his office hours are listed on the course blackboard page
- We will be starting promptly at 6:00
- Email responses will be best effort, we are going to use Slack for this class. Instructions on how to join our space are here: https://github.com/ArchitectingSoftware/CS575-SoftwareDesign/blob/master/SE-575-StartHere.md

About the Course Title – It probably needs to be changed…

Software Design?
Software Architecture?
Both?

This course has evolved over the years

- Started off as an advanced design course – projects aligned towards designing something non-trivial, GoF and code writing focus
- Next Evolution – "Now and Then" – looking at the collective works of the father of software design, David Parnas, and investigating how his seminal research can be applied today
- Pragmatic, Architecture Focused – Architecture focus on a variety of modern problems that blend application-, platform- and integration-concerns associated with building modern, highly reliable, highly scalable solutions.

Why study Software Design and Architecture?

The nature of systems that we commonly build stretch the design skills we first learned

- Most modern systems are distributed – this introduces a ton of design complexity
- The availability of choices
  - Users of your systems are not a locked in as they have been in the past - things like the cloud are changing things
  - If your design is not good, how can you keep up with competition?
- The pace of software development
  - Software development now requires rapid iterations and the delivery of incremental functionality – what happens if you have a poor design?
- New practices are emerging to support the design of modern systems
  - CI/CD, Feedback/Measurement built in, design for change as a first class concern, APIs and Microservices, …

Do I need to know how to Code? What do I need to know how to Code?

YES – BUT - This is not a deep coding course, but you will need to follow examples that involve the presentation of code. Your project also will require the implementation of a working solution.

You should know

- An modern or OO programing language – Java OK, but there are better choices
- Working knowledge of how a VM operates

You will be exposed to

- Dynamic languages – especially around how dynamic languages support polyglot software development
- Functional languages – eliminating boilerplate, and how functional languages work related to software design
- Designing and architecting for the "browser as a platform" – we will be looking at HTML5/JavaScript solutions and frameworks using my webpage as a working example.

How will this course operate?

This course has been upgraded since the last time that I taught it

- The good is that it will cover lots of new material covering the latest topics in software design and architecture
- The bad is that I will probably only be a week or two ahead of the class preping materials. Please be patient – if you want to work ahead feel free to look into materials related to the topics to be discussed.

Materials

- There will not be a course textbook. I could not cover the materials that I want without having multiple textbooks
- There will be links to required reading materials that will complement the lectures and lecture notesð

Speakers

- I will probably invite one or two outside speakers to address the class

How will my grades be derived?

This class will not have any exams. There will be a midterm deliverable and a final project. There will be other components to this class factoring into your grade:

- Participation in class and in the discussion forums
- Reading Papers, Watching Videos
- (Midterm) An architecture design project
- (Final) A team project where you will be building something

Course Deliverables - 1

Architecture Design Assignment

- You will conduct some independent research on the Internet of Things (IoT)
- You will be exposed to the concept of a reference architecture, and develop a reference architecture for IoT devices.
- You will use your creativity to propose some features for to solve the problem objectives in creating a "smart device".
- You will develop and document an architecture for your IoT device

Assignment Deliverables

- You will be provided several resources (videos, papers, etc) to learn about reference architectures and IoT devices
- You will develop your final deliverable, a 3-5 page document for your smart IoT device architecture.

We might allocate one class to examine the best, and most interesting case designs submitted…

Course Deliverables - 2

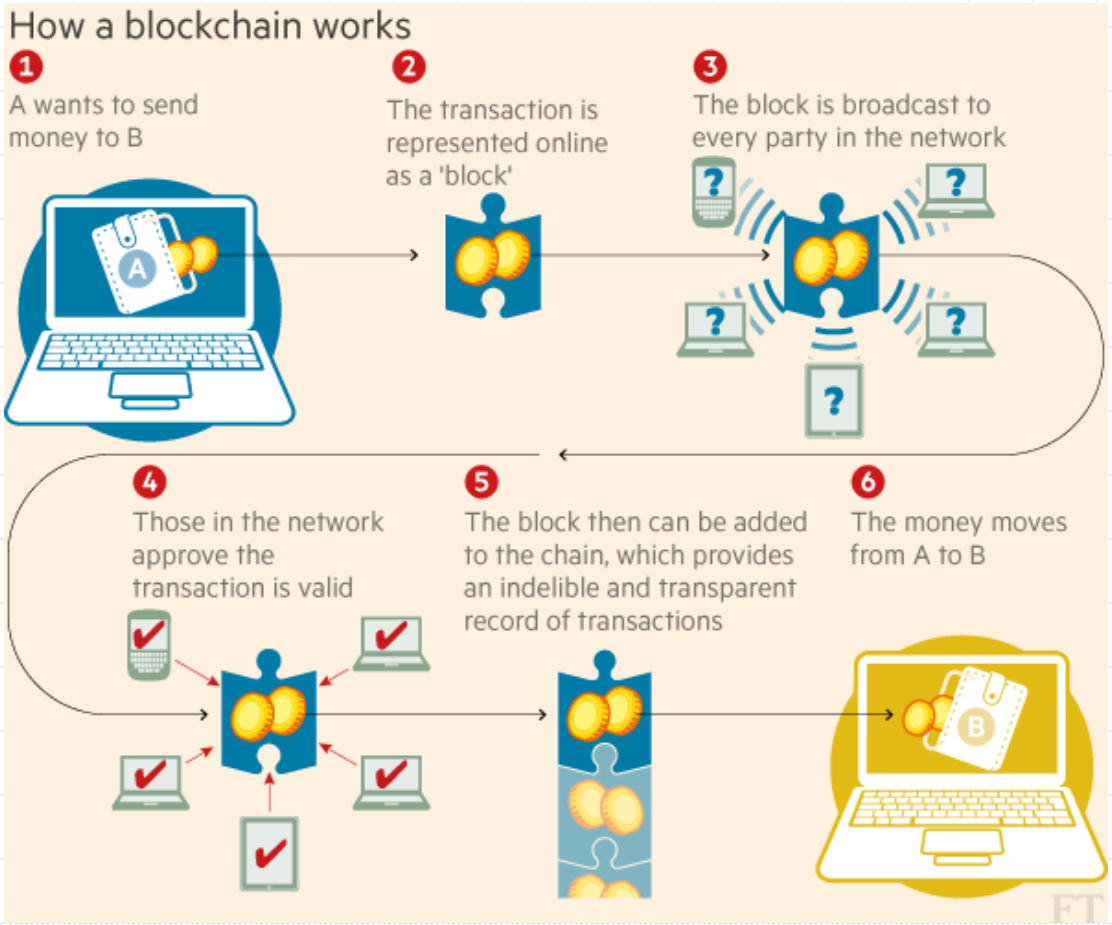
Implantation of a non-trivial project showcasing a modern software architecture

- You will implement and deploy a small system demonstrating some of the software architecture and design concepts covered in the class
- This is not expected to be a large software development effort, but what you build should embrace some of the modern software design and/or architecture concepts that we covered in class:
  - Poloyglot development
  - The use of functional programming languages – especially how they support cleaner design
  - The use of domain specific languages – how they change the approach to design and testing
  - Design and architecting for mobile
  - Designing and architecting for the cloud
  - Reactive systems, Resilience and/or Cloud Native Architectures
  - Modern use of web services in designing software
  - Use of cloud architectures

This will be a group project with 3 people per group unless otherwise approved. I will discuss group formation

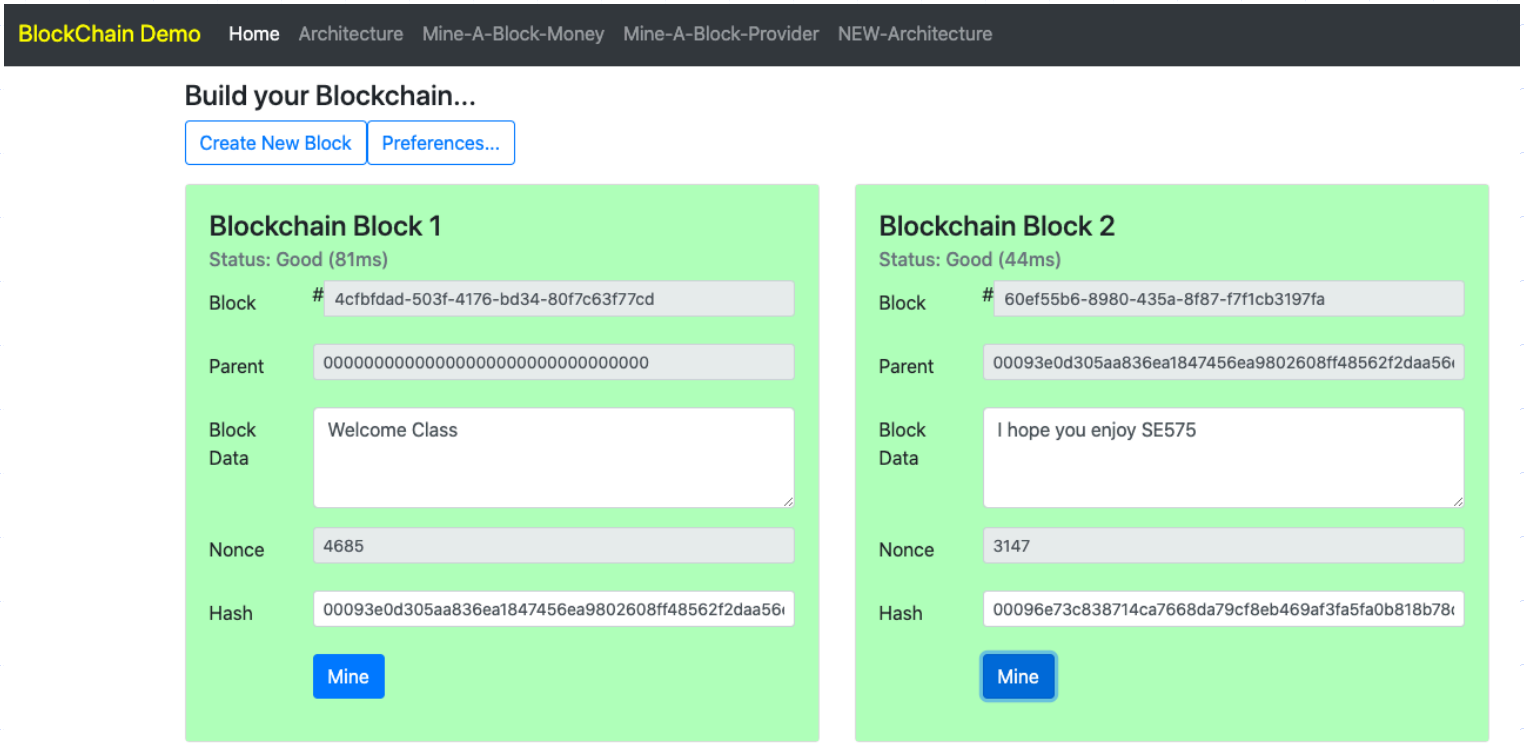
Course Deliverables - 2

The default project we will be considering is building a blockchain simulator, you can do something else if you want with approval.



Course Deliverables - 2

BLOCKCHAIN DEMO TIME



Other Course Deliverables

Research Paper Summaries

- Read a paper and address specific questions in the assignment in an online discussion forum.

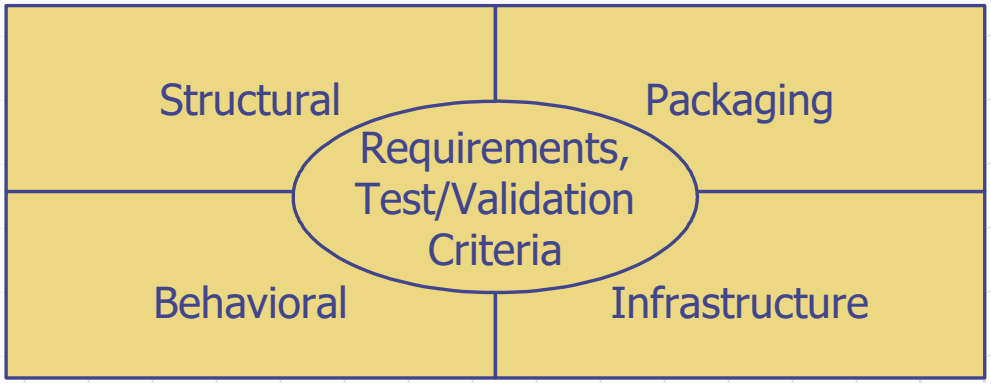
Feedback to the Speaker

- Watch a video from an industry leader and provide him/her with virtual feedback incorporating some of the course concepts.

Course Topics

Reviewing the basics…

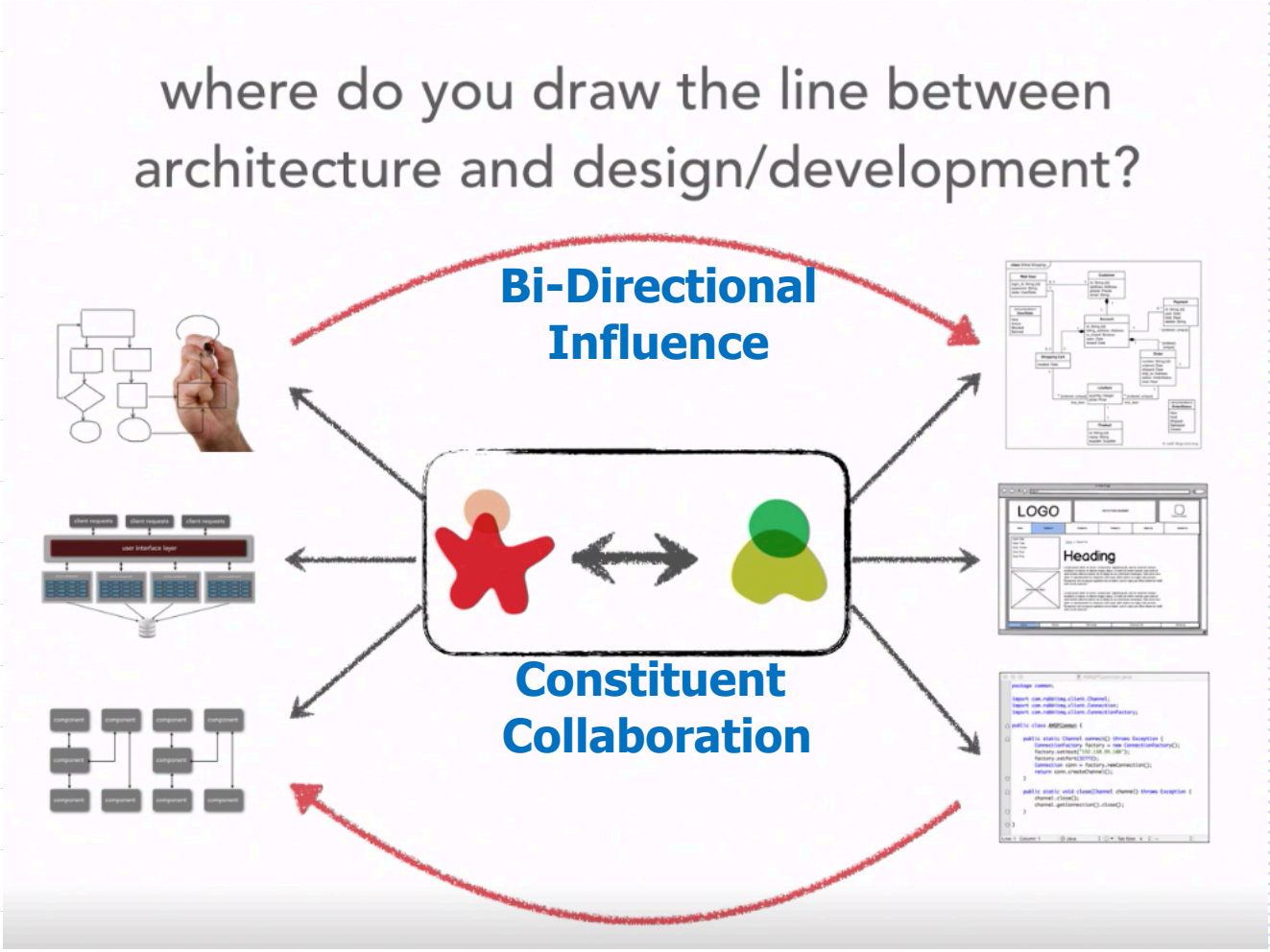
- What is software architecture
- How does it relate to design
- What is the difference between an architecture style, an architecture pattern and a design pattern
- What are views
- What are important views when documenting or describing the architecture of a software system
- Why do we need to do architecture
- Architecture and agile – can this be done, and if so, how?



Course Topics

The intersection between software design and architecture…

Ref: Neil Ford & Mark Richards
Software Architecture Fundamentals



Architecture Stuff

Design Stuff

Course Topics

Breaking bad habits & being exposed to new things...

- Compare different programing and design models
  - OO, FP
- Why do we have different models?
- What's wrong with OO (specifically Java), and why the current reemergence of FP?

Throughout this course many examples will be provided using Scala, Kotlin, GoLang and modern JavaScript/Typescript to highlight polyglot software development

Course Topics

Good design principals – e.g., SOLID Model

- Single responsibility principle: a class should have one, and only one, reason to change;
- Open-closed principle: it should be possible to extend the behavior of a class without modifying it;
- Liskov Substitution principle: subclasses should be substitutable for their superclasses;
- Interface segregation principle: many small, client-specific interfaces are better than one general purpose interface;
- Dependency inversion principle: high level modules should not depend on low level modules – both should depend on abstractions; abstractions should not depend upon the details. Details should depend upon the abstractions.

Course Topics

DSL – Domain Specific Languages and the modern software design stack…

- What are DSLs and why are they helpful
- Why are they used
- How to they aid in testability
- How do they layer into an overall architecture
- Internal versus External DSL
  - External: XML, Lex, Regular Expressions, YACC, SQL, …
  - Internal: Scala, Groovy, Ruby…
- Internal DSLs are like libraries but they favor "natural" language of the domain versus the host syntax
  - Metaprogramming in Groovy
  - Implicits and other "sugar" in Scala

Ill be touching on DSLs from time to time but we will be looking at the reactive DSLs a good bit

Course Topics

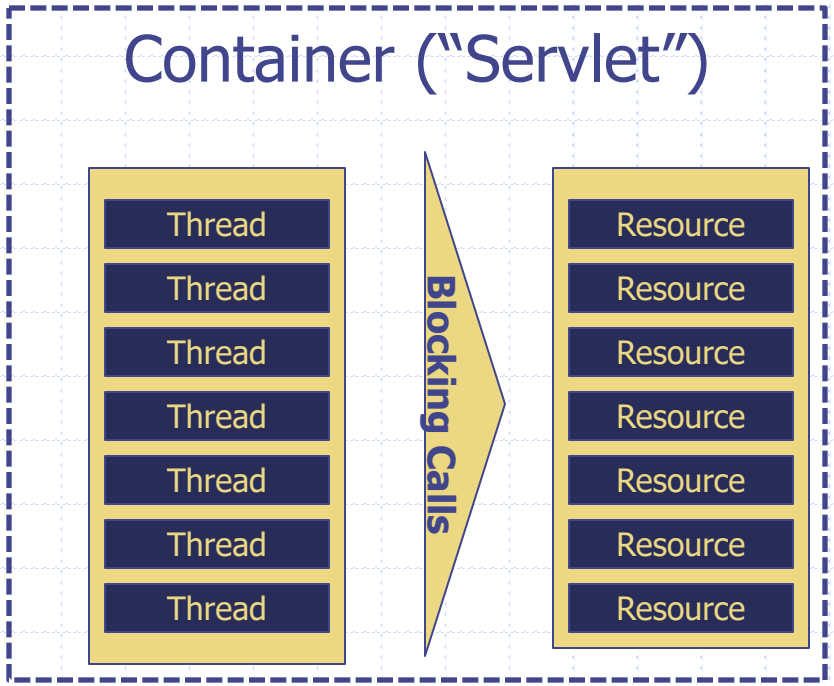
Designing and architecting for scale (using the Web as a case study)

- Investigation into web architectures
- What's the problem with traditional web architectures that impact scalability?
- Promoting stateless and distributing computing to the client
- Functional programming
- The "Reactor" pattern and how it is implemented – Node.js, Netty
- Making distributed computing easier – Actors, Future/Promise models, coroutines, etc.
- Reactive Programming Model and Frameworks

Designing/ Architecting for Scale

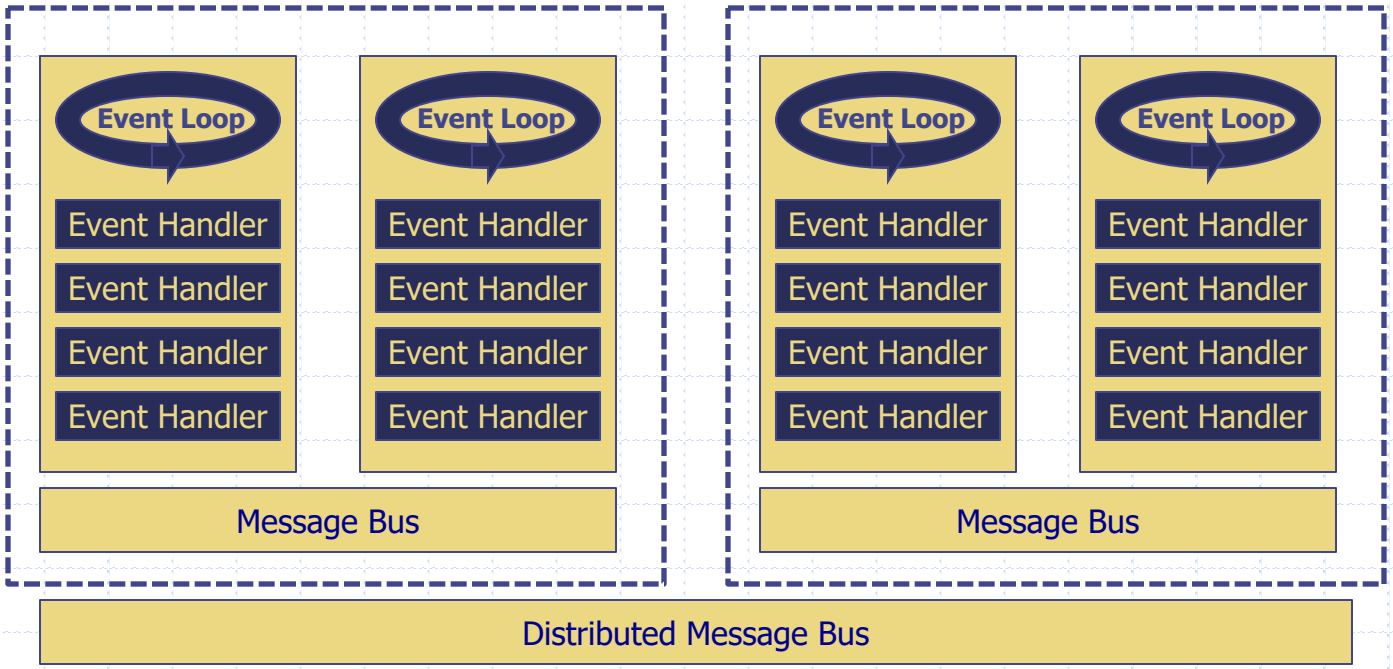
Traditional Web Architecture

Requests are routed by a container to a thread. The request lives on the thread (uses it) until all of its I/O requests are satisfied and a result is sent back to the caller. Throughput is governed by the number of threads that are allocated to the container



Designing/ Architecting for Scale

(Multi-)Reactor Pattern



In this model each server runs a small number of threads, each thread executes an event loop. As requests come in they are assigned to an event handler. The event handler provides a "callback" and gives up control while it is waiting for I/O from a resource

Designing/ Architecting for Scale

Reactive applications and frameworks that support them are becoming popular for architecting high performance applications.

- Resilient: The ability to recover and repair itself automatically in order to provide seamless business continuity.
- Interactive: Rich, engaging, single page user interfaces that provide instant feedback based on user interactions and other stimuli.
- Scalable: Can scale within and across nodes elastically to provide compute power on-demand when it's needed.
- Event-Driven: Enables parallel, asynchronous processing of messages or events with ease.

Used in the Typesafe Stack, Neflix Architecture, Microsoft Extensions, Modern Web Frameworks (Angular, React, Vue), Mobile, etc

Designing/ Architecting for Scale

Modern languages such as scala make implementing the asynchronous model easier by abstracting threads using an actor model and higher-order concepts such as futures.

```
val f: Future[List[String]] = future {
  session.getRecentPosts
}
f onComplete {
  case Success(posts) => for (post <- posts) println(post)
  case Failure(t) => println("An error has occured: " + t.getMessage)
}
```

```
val f: Future[List[String]] = future {
  session.getRecentPosts
}
f onFailure {
  case t => println("An error has occured: " + t.getMessage)
}
f onSuccess {
  case posts => for (post <- posts) println(post)
}
```

Lets take a look at: http://docs.scala-lang.org/overviews/core/futures.html

We will also examine how these concepts fit into the Javascript promise model, Kotlin Coroutines, and Java 8

Designing/ Architecting for Scale - Kotlin

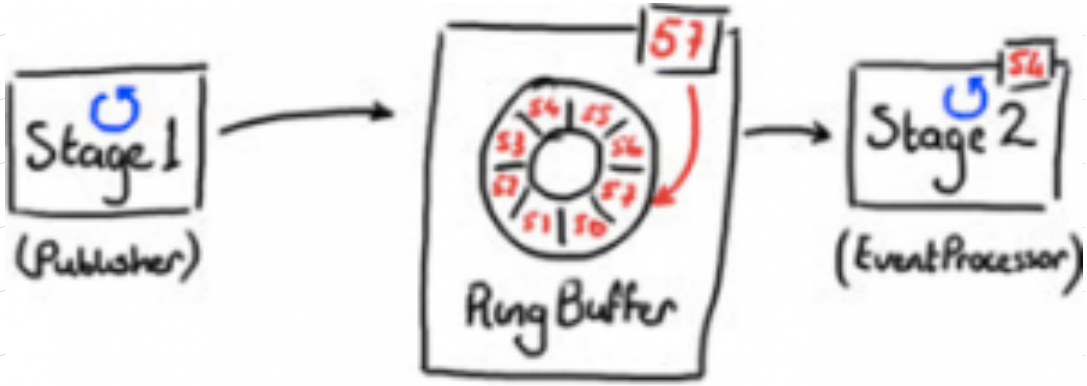
```
fun main(args: Array<String>) = runBlocking<Unit> {
    val time = measureTimeMillis {
        val one = async { doSomethingUsefulOne() }
        val two = async { doSomethingUsefulTwo() }
        println("The answer is ${one.await() + two.await()}")
    }
    println("Completed in $time ms")
}
```

```
fun main(args: Array<String>) = runBlocking<Unit> {
    val channel = Channel<Int>()
    launch {
        // this might be heavy CPU-consuming computation or async logic, we'll just send five squares
        for (x in 1..5) channel.send(x * x)
    }
    // here we print five received integers:
    repeat(5) { println(channel.receive()) }
    println("Done!")
}
```

```
// This function launches a new counter actor
fun CoroutineScope.counterActor() = actor<CounterMsg> {
    var counter = 0 // actor state
    for (msg in channel) { // iterate over incoming messages
        when (msg) {
            is IncCounter -> counter++
            is GetCounter -> msg.response.complete(counter)
        }
    }
}
```

Novel Patterns for Scale

We will also examine novel patterns that were created to support solutions that run at a very high throughput – one example is the LMAX Disruptor – designed to process 6 million orders per second on a single JVM

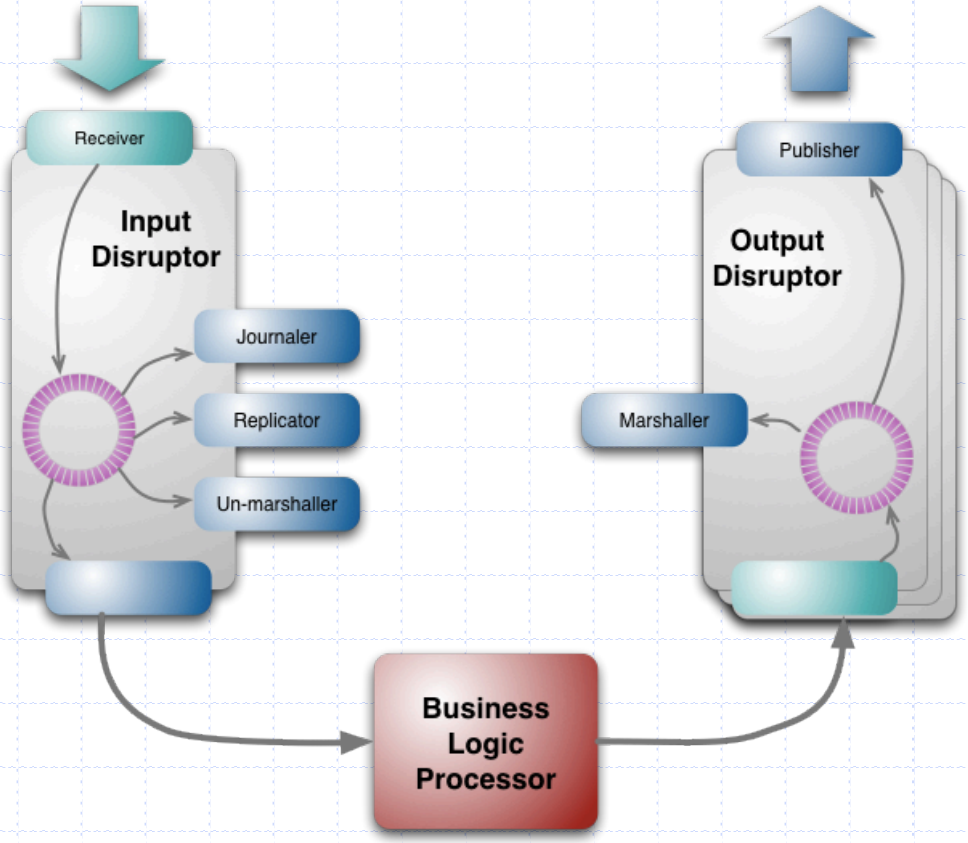


Use ring buffer data structure to enable contention-free thread sharing, the only thing that is shared is a single volatile variable to ensure that the CPU does not reorder instructions.

Avoids issues with using queues and also addresses most concurrency solutions work with either mainly-full or mainly-empty queues – hard to get to a balanced producer/consumer steady state with events.

Novel Patterns for Scale

Lets see how this all comes together by chaining disruptors to form an async event pipeline (picture from MartinFowler.com)



Distributed Computing Patterns – Reactive Patterns

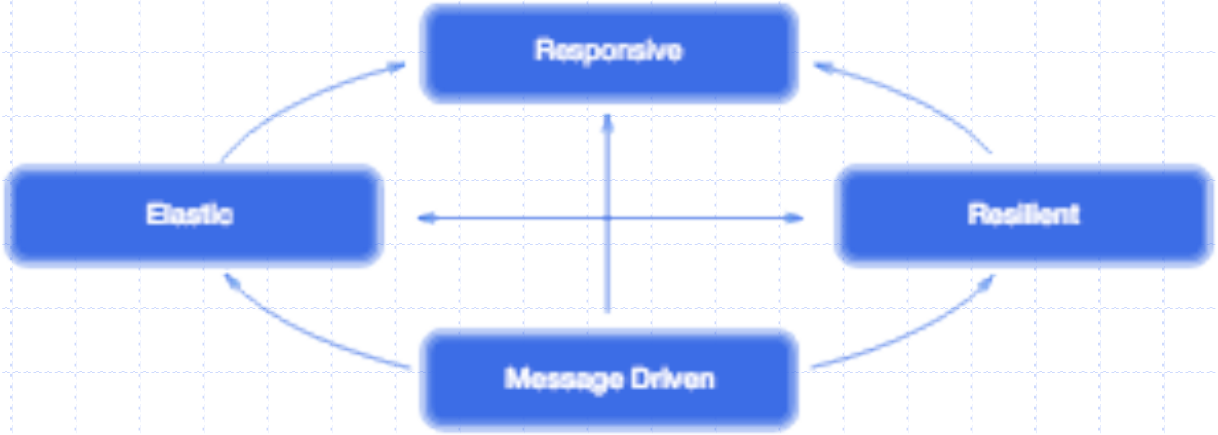
	One	Many
Synchronous	`Try[T]`	`Iterable[T]`
Asynchronous	`Future[T]`	`Observable[T]`

Considering the architecture of the hardware becomes important for designing systems that scale. Goal is to make code non-blocking by using other modern constructs introduced as part of reactive extensions libraries.

Idea is to use fewer threads, take advantage of all cores, don't block, and embrace eventing and streaming.

Nice demo here: https://www.learnrxjs.io

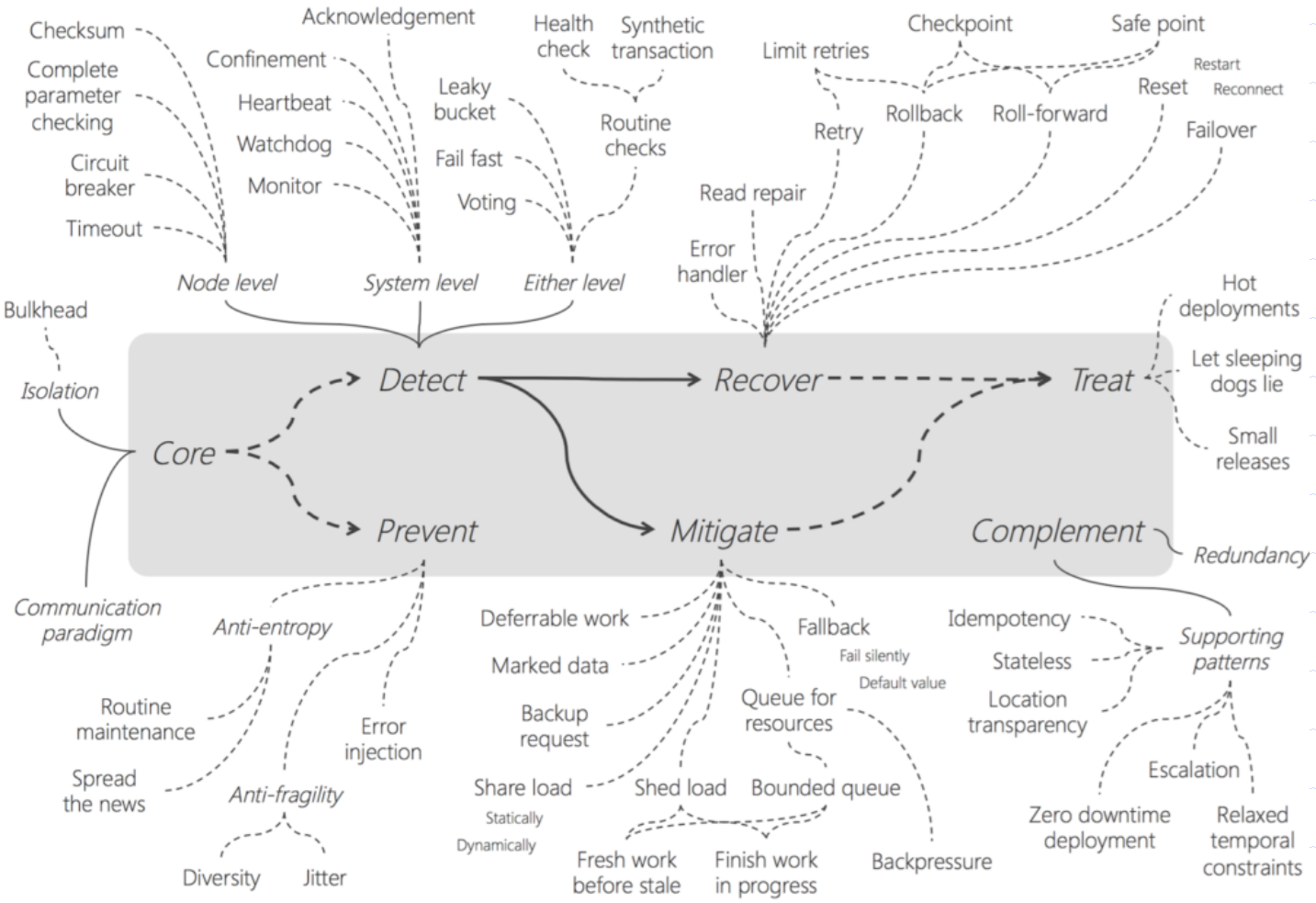
The Reactive Manifesto



See: http://www.reactivemanifesto.org/

- **Responsive – The system responds in a timely manner**
- **Resilient – The system stays responsive in the face of failure**
- **Elastic – The system stays responsive under varying workloads**
- **Message Driven – The system relies on asynchronous message passing to establish a boundary between components that ensure loose coupling**

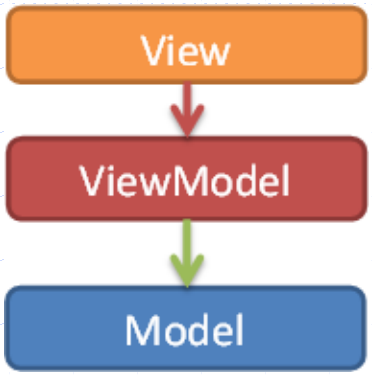
Resiliency Patterns

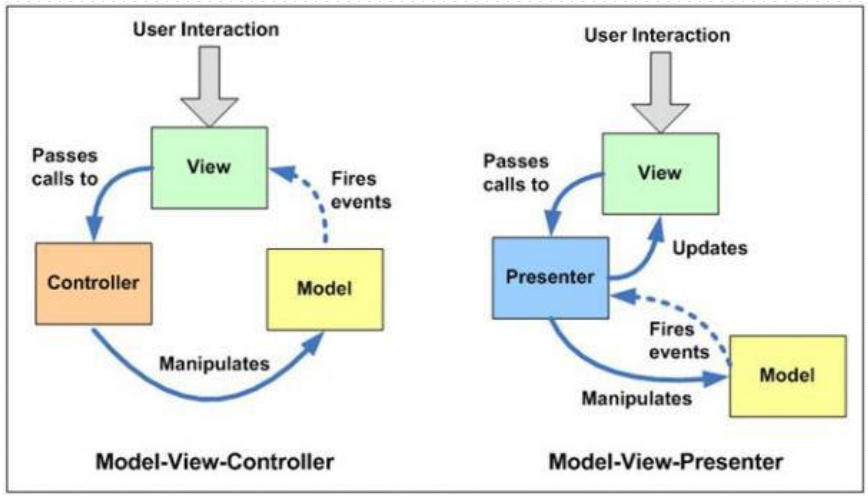


Course Topics

Modern Web Architectures

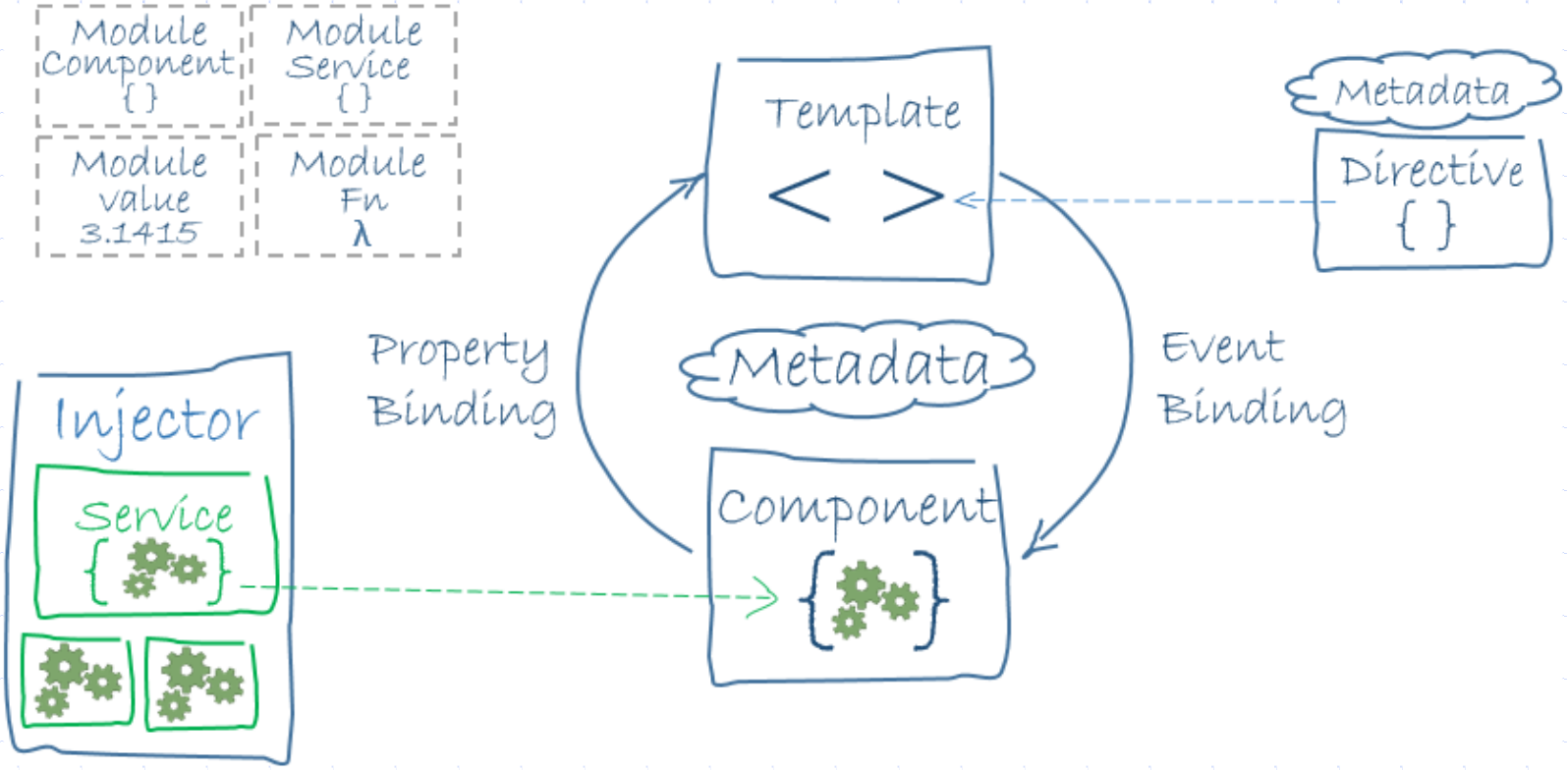
- The HTML 5 ecosystem
- Javascript as a platform
- The MVC, MVVM patterns move to browser
- Bringing server capabilities to the browser – Dependency Injection, Dependency Management, Promise/Future, Configuration Management, etc.
- Application distribution via CDN





Course Topics

Building Modern Web Applications – Modern Architecture – Web Components



Course Topics

Service Oriented Architecture

- Why is SOA so important and popular
- Is SOA an Architecture?
- Types of SOA – REST and SOAP

Event and Streaming Architectures

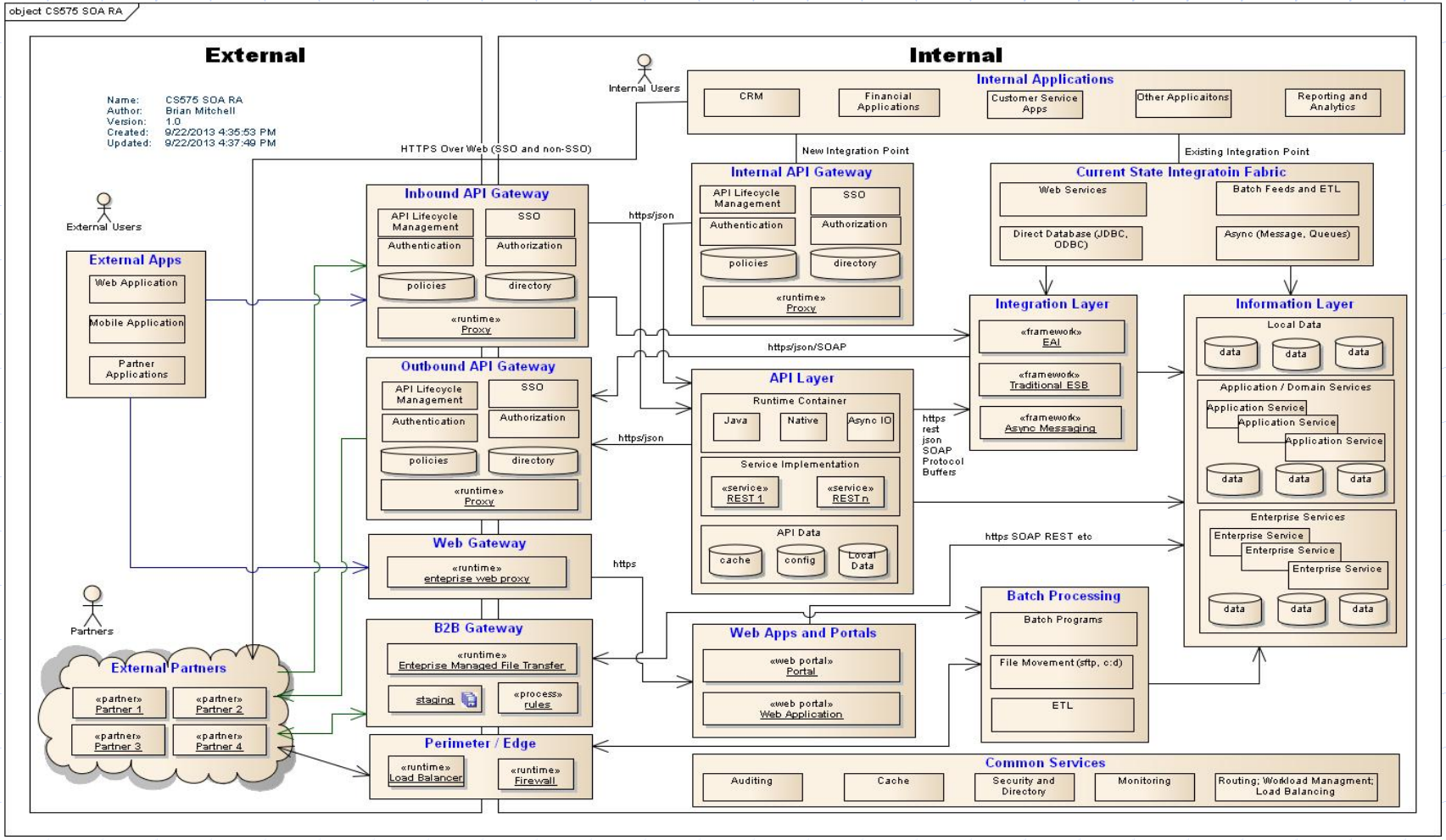
(https://martinfowler.com/articles/201701-event-driven.html)

- What are these?
- What are the interesting patterns?
  - CQRS
  - Event Carried State Transfer
  - Event Sourcing

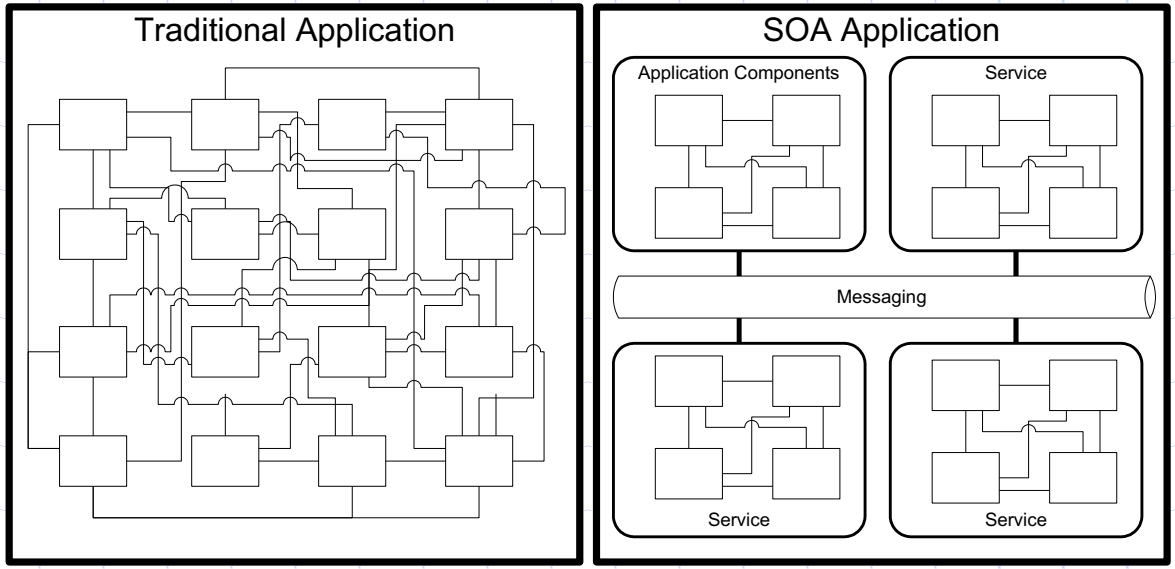
Microservice Architectures (vs Monolithic Architectures)

- Design considerations

Course Topics - SOA

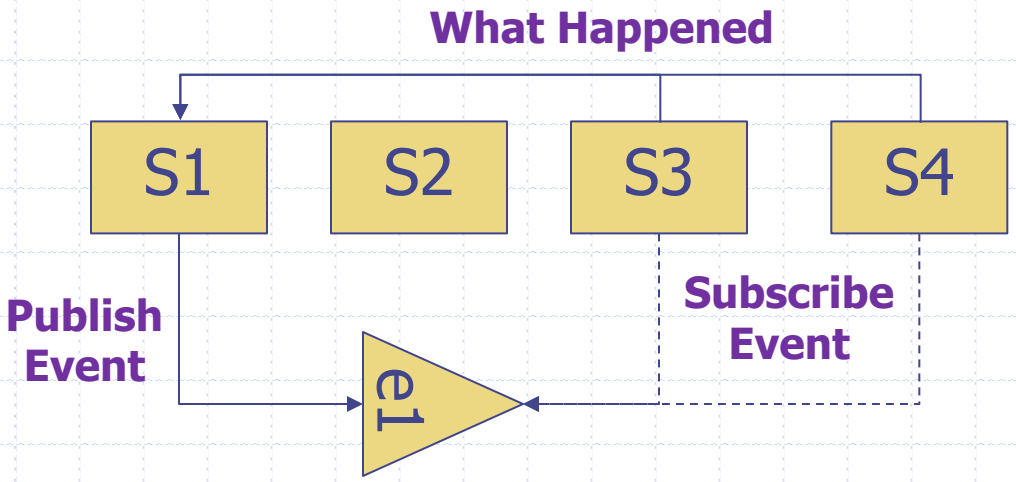


Traditional versus SOA applications – a line and box view…



- **Applications designed in conjunction with an SOA style are easier to design, implement, maintain and extend due to the loose coupling of components that reside in different services**
  - **The pieces (services) in an SOA are designed with a courser granularity than an design based on traditional components, making the system simpler to deal with**
  - **The messaging interfaces between the services in an SOA have structure and semantics allowing them to be intelligently routed and provisioned based on application-specified policies**

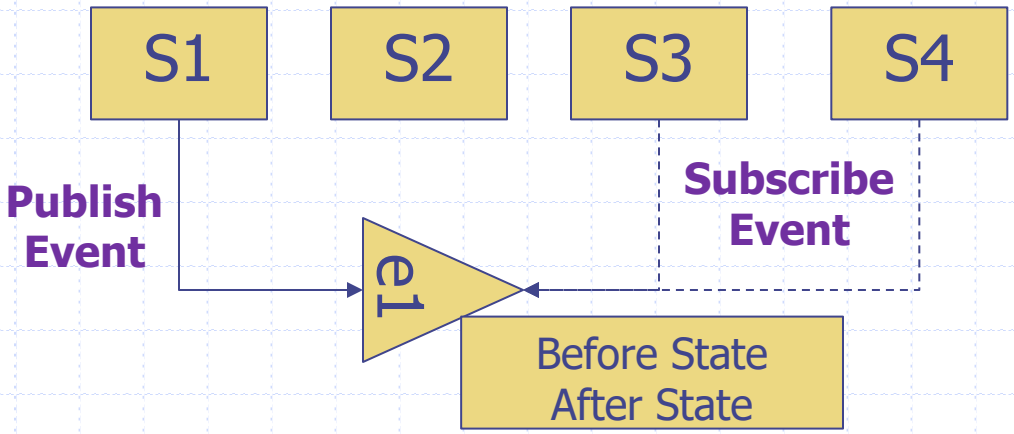
Event and Streaming Architectures



Event Notification
Loosen coupling with pub/sub patterns
Interest is only in if the event has occurred

Source: The Many Meanings of Event-Driven Architecture – Martin Fowler

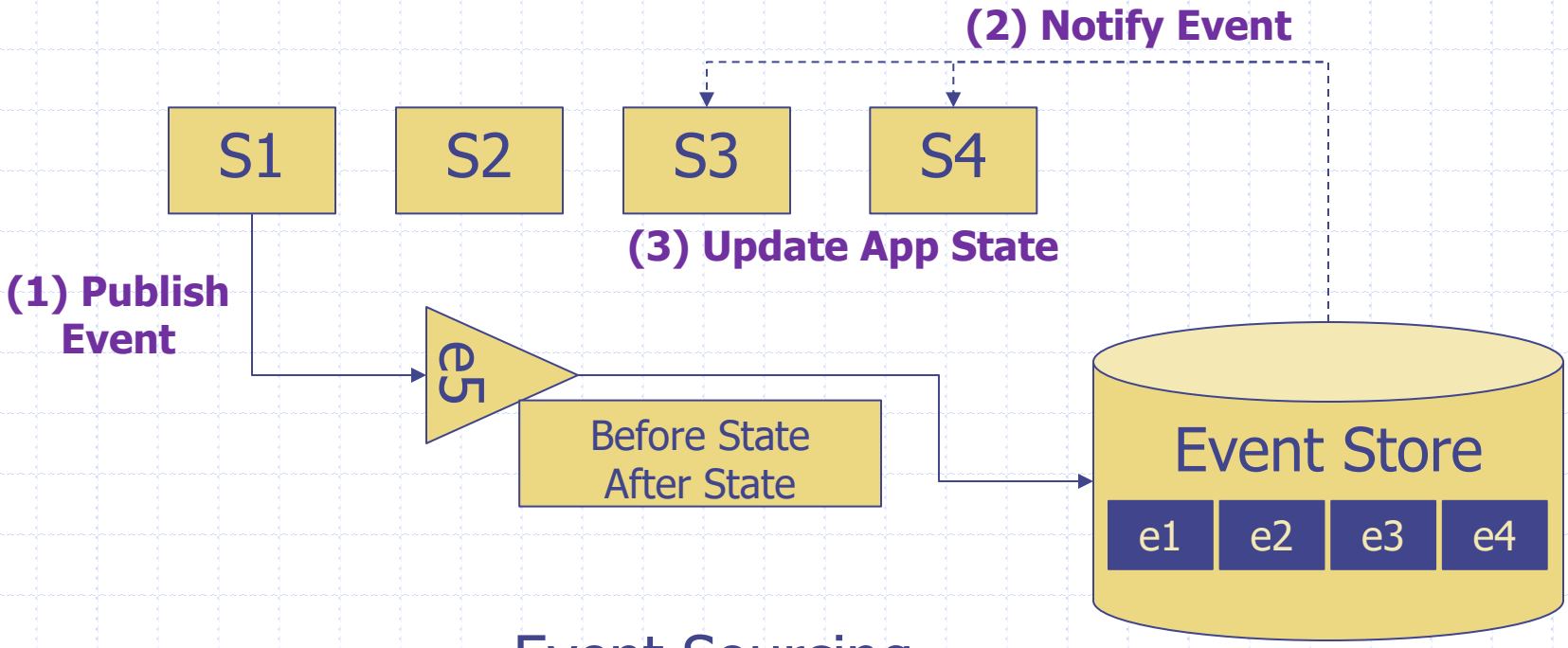
Event and Streaming Architectures



Event Carried State Transfer
Loosen coupling with pub/sub patterns
Eventual Consistency

Source: The Many Meanings of Event-Driven Architecture – Martin Fowler

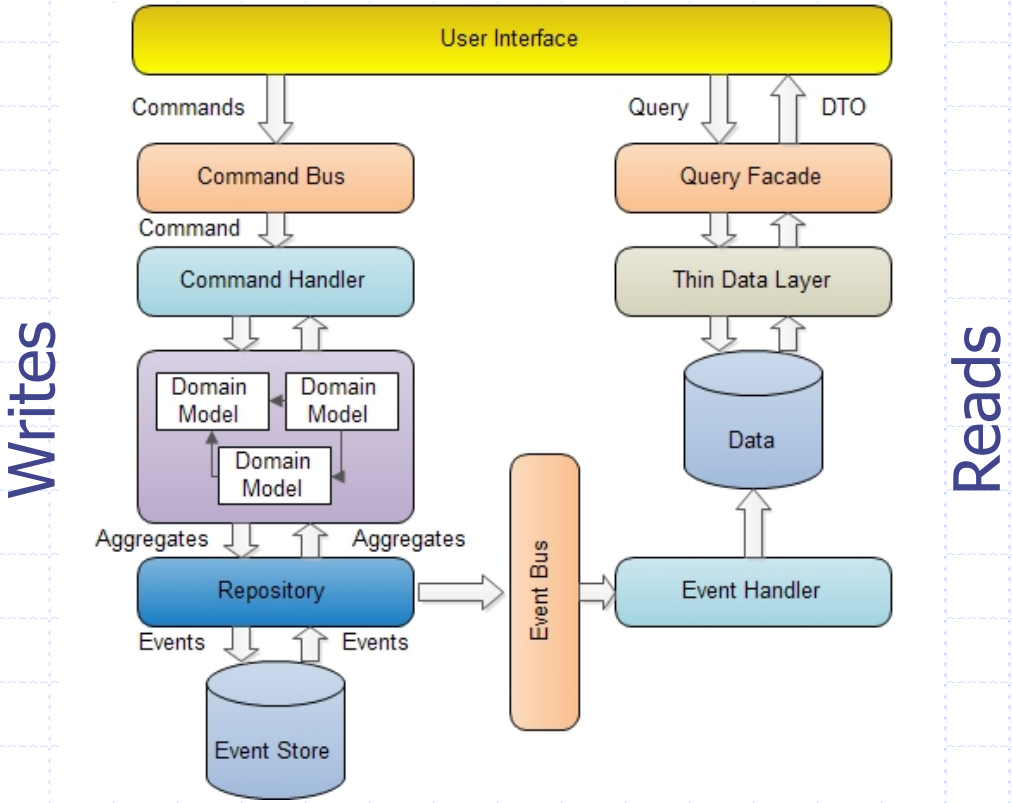
Event and Streaming Architectures



Event Sourcing
Event Store Maintains State Over Time
Loosens Coupling

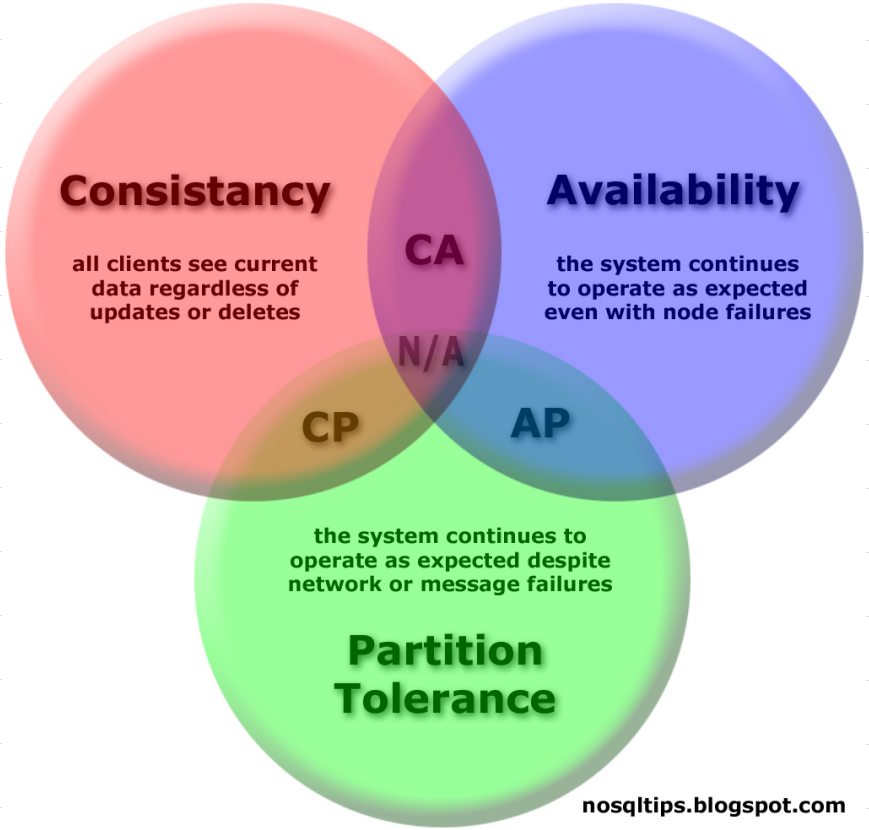
Source: The Many Meanings of Event-Driven Architecture – Martin Fowler

Event and Streaming Architectures



Scalability and Consistency via Separate Handling of Reads and Writes (CQRS)

Course Topics - CAP



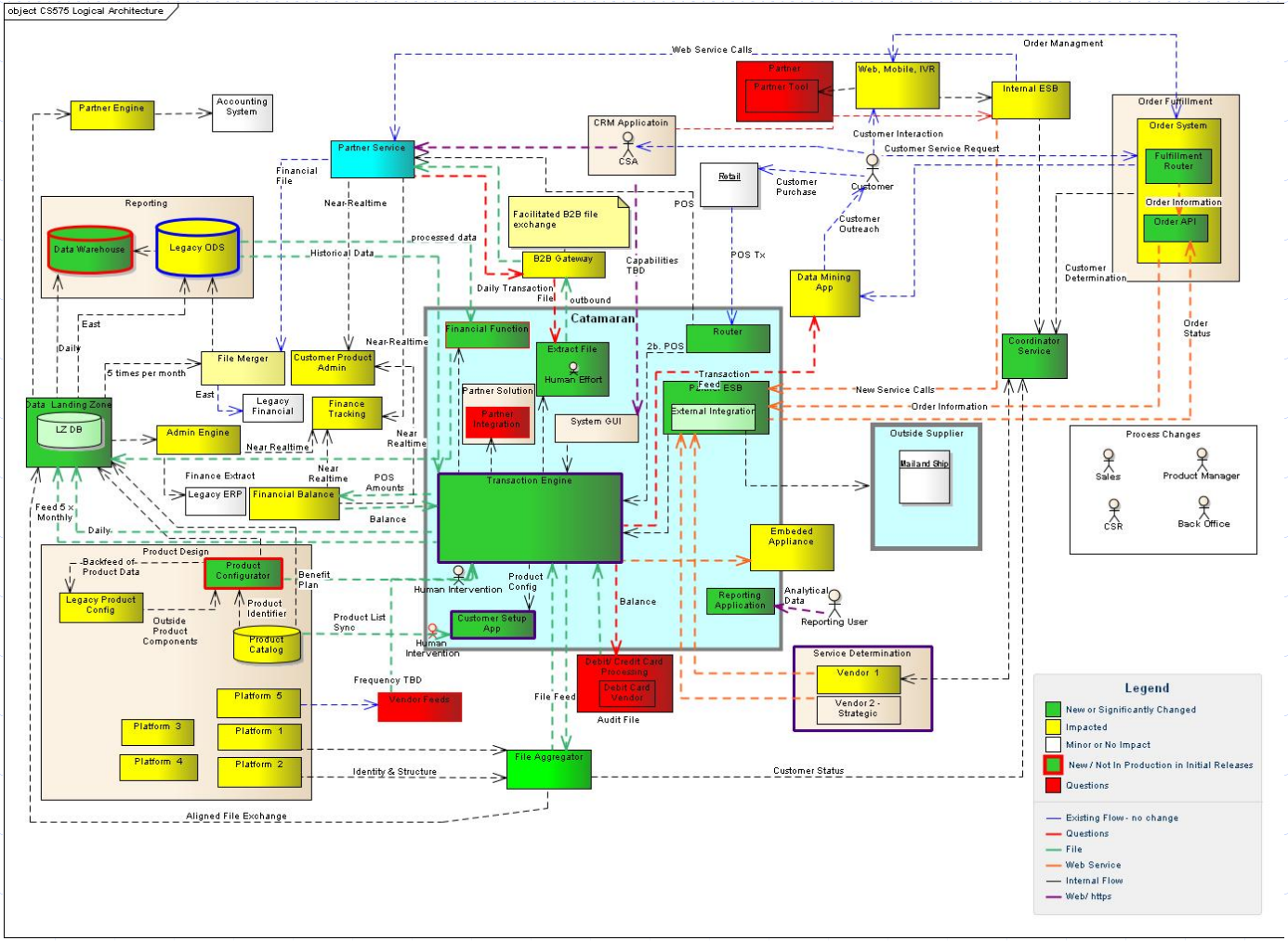
CAP Theorem and its impact on design and architecture

Course Topics

Modeling Software Architectures

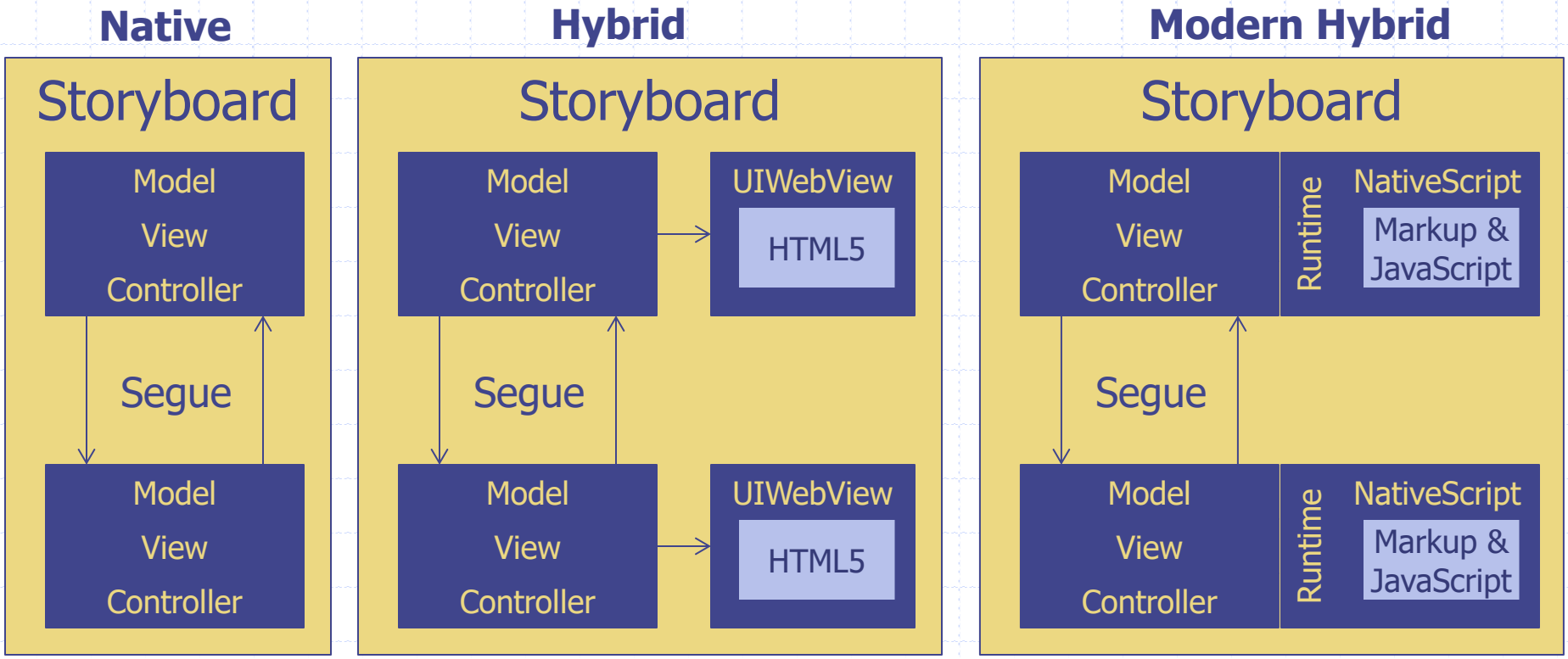
- Why do we model
- Important views
- Communicating to stakeholders
- Reference, Conceptual, Logical, and Application architectures
- Use of tooling to model software architectures

Modeling Architecture Example



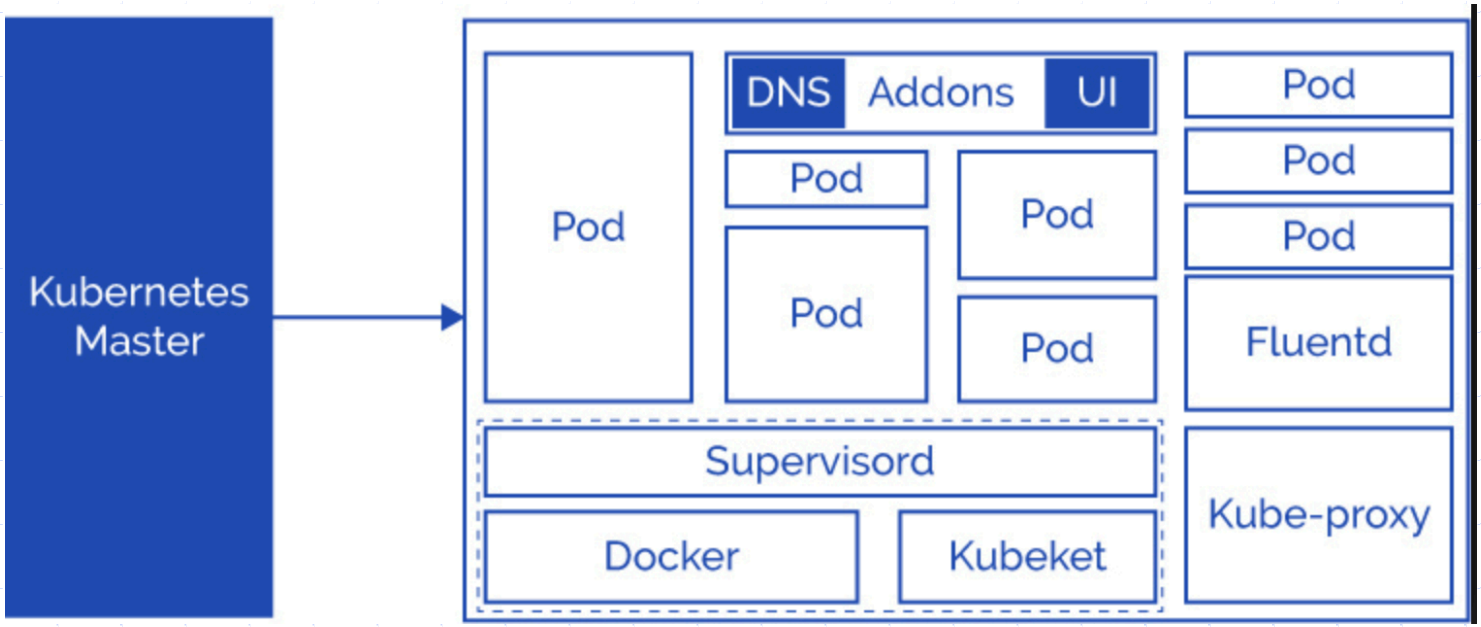
Course Topics

Types of Mobile Architectures

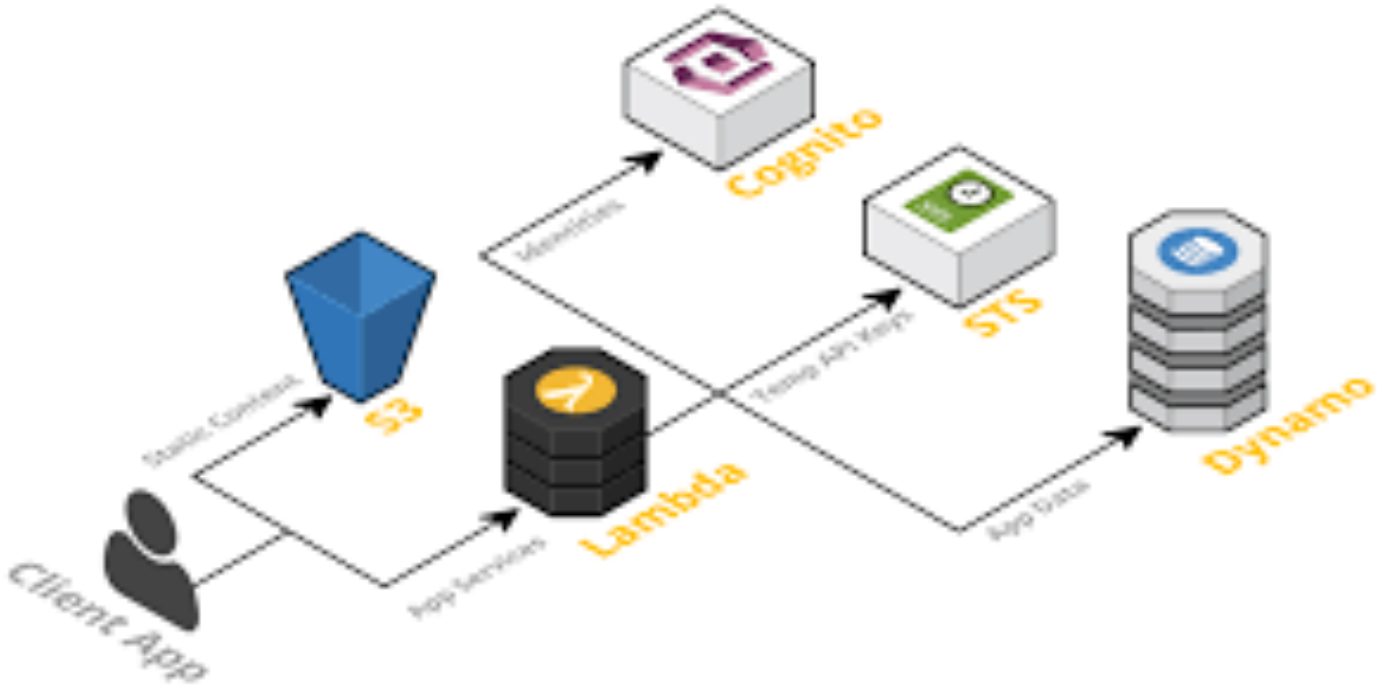


Cloud Native Applications

Cloud Native Applications – Container Orchestration

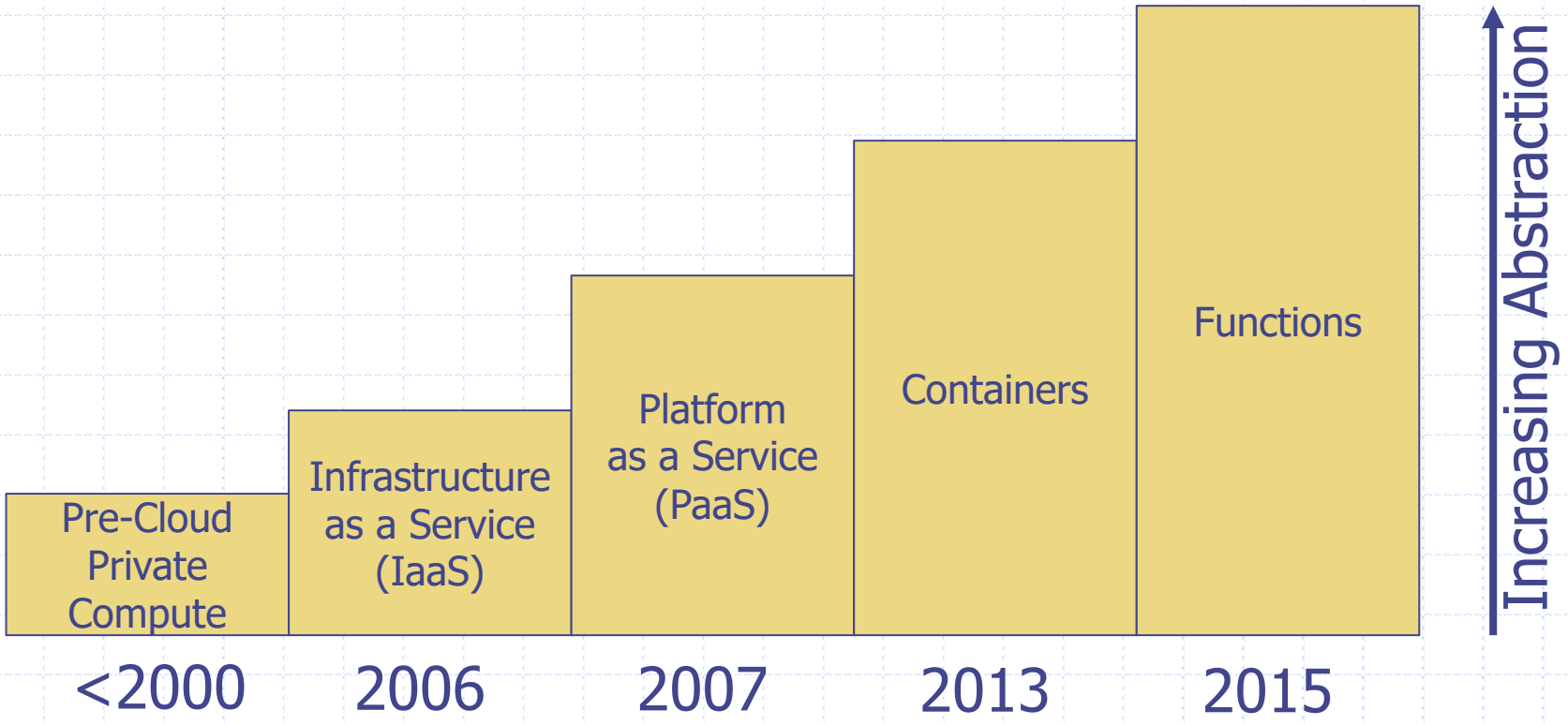


Cloud Native Applications – Serverless Architectures



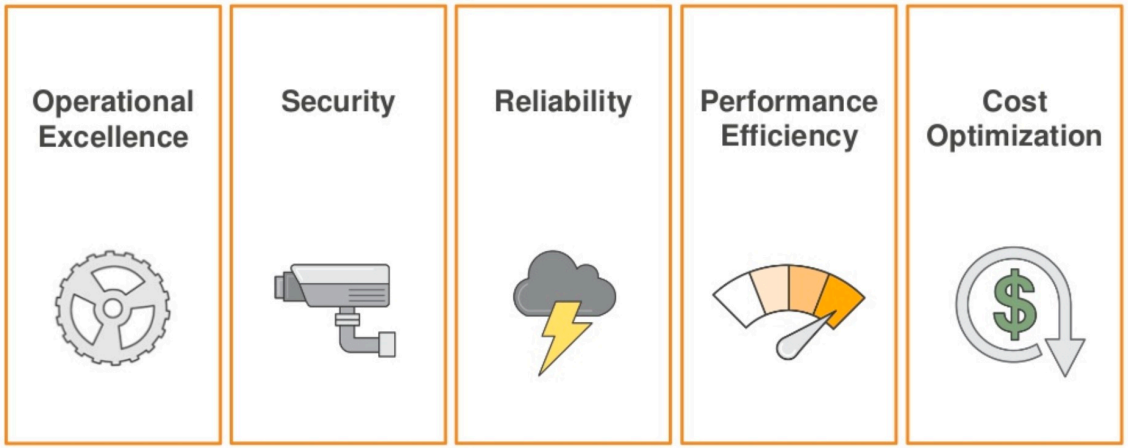
Course Topics

Cloud Models



Course Topics

Architecting for the cloud

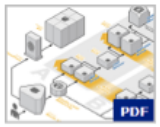


Materials adopted from: http://www.oracle.com/technetwork/articles/cloudcomp/jimerson-ha-arch-cloud-1669855.html

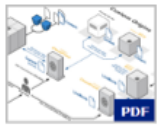
Course Topics

Cloud Specific Reference Architecures

http://aws.amazon.com/architecture/



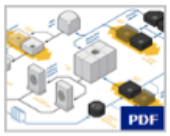
Web Application Hosting
Build highly-scalable and reliable web or mobile-web applications (PDF)



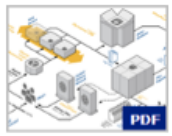
Content and Media Serving
Build highly reliable systems that serve massive amounts of content and media (PDF)

Batch Processing
Build auto-scalable batch processing systems like video processing pipelines (PDF)

Fault tolerance and High Availability
Build systems that quickly failover to new instances in an event of failure (PDF)



Media Sharing
Cloud-powered Media Sharing Framework (PDF)

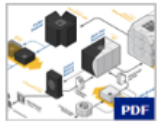


Online Games
Build powerful online games (PDF)

Log Analysis
Analyze massive volumes of log data in the cloud (PDF)

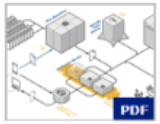
Financial Services Grid Computing
Build highly scalable and elastic grids for the Financial Services Sector (PDF)

Large Scale Processing and Huge Data sets
Build high-performance computing systems that involve Big Data (PDF)



Ad Serving
Build highly-scalable online ad serving solutions (PDF)

Disaster Recovery for Local Applications
Build cost-effective Disaster Recovery solutions for on-premises applications (PDF)



File Synchronization
Build simple file synchronization service (PDF)

E-Commerce Website Part 1: Web Frontend
Build elastic Web Front-ends for an e-Commerce website (PDF)

E-Commerce Website Part 2: Checkout Pipeline
Build highly scalable checkout pipeline for an e-Commerce website (PDF)

E-Commerce Website Part 3: Marketing and Recommendations
Build highly scalable recommendation engine for an e-Commerce website (PDF)