



BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Module 7

Patterns – Part 3 Suppl Adaptable Systems.

Harvinder S Jabbal
SSZG653 Software Architectures

Categories

From Mud to Structure.

- Patterns in this category help you to avoid a 'sea' of components or objects.
- In particular, they support a controlled decomposition of an overall system task into cooperating subtasks.
- The category includes
 - the Layers pattern
 - the Pipes and Filters pattern
 - the Blackboard pattern

Distributed Systems.

- This category includes one pattern.
 - Broker
- and refers to two patterns in other categories,
 - Microkernel
 - Pipes and Filters
- The Broker pattern provides a complete infrastructure for distributed applications.
- The Microkernel and Pipes and Filters patterns only consider distribution as a secondary concern and are therefore listed under their respective primary categories.

Interactive Systems.

- This category comprises two patterns,
 - the Model-View-Controller pattern (well-known from Smalltalk,)
 - the Presentation-Abstraction-Control pattern.
- Both patterns support the structuring of software systems that feature human-computer interaction.

Adaptable Systems.

- This category includes
 - The Reflection pattern
 - the Microkernel pattern
- strongly support extension of applications and their adaptation to evolving technology and changing functional requirements.

Adaptable Systems.

- This category includes
 - The Reflection pattern
 - the Microkernel pattern
- strongly support extension of applications and their adaptation to evolving technology and changing functional requirements.

Adaptable

- **Systems evolve over time**
 - -new functionality is added and existing services are changed.
- **They must support new versions of**
 - operating systems,
 - user-interface platforms or
 - third-party components and libraries.
- **Adaptation to**
 - new standards or
 - hardware platforms may also be necessary.
- **During system design and implementation,**
 - customers may request new features, often urgently and at a late stage. Y
 - You may also need to provide services that differ from customer to customer.

Design for Change

- Design for change is therefore a major concern when specifying the architecture of a software system.
- An application should support its own modification and extension a priori.
- Changes should not affect the core functionality or key design abstractions, otherwise the system will be hard to maintain and expensive to adapt to changing requirements.



Adaptable Systems: Microkernel

Adaptable Systems: Microkernel



- The Microkernel architectural pattern applies to software systems that must be able to adapt to changing system requirements.
- It separates a minimal functional core from extended functionality and customer-specific parts.
- The microkernel also serves as a socket for plugging in these extensions and coordinating their collaboration.

Context

The development of several applications that use similar programming interfaces that build on the same core functionality.

Problem

- Developing software for an application domain that needs to cope with a broad spectrum of similar standards and technologies is a non-trivial task.
- Well-known examples are application platforms such as operating systems and graphical user interfaces.

FORCES:

- The application platform must cope with continuous hardware and software evolution.
- The application platform should be portable, extensible and adaptable to allow easy integration of emerging technologies
- The applications in your domain need to support different, but similar, application platforms.
- The applications may be categorized into groups that use the same functional core in different ways, requiring the underlying application platform to emulate existing standards.
- The functional core of the application platform should be separated into a component with minimal memory size, and services that consume as little processing power as possible.

Solution

- Encapsulate the fundamental services of your application platform in a microkernel component.
- The microkernel includes functionality that enables other components running in separate processes to communicate with each other.
- It is also responsible for maintaining system- wide resources such as files or processes.
- In addition, it provides interfaces that enable other components to access its functionality.

Solution - details

- Core functionality that cannot be implemented within the microkernel without unnecessarily increasing its size or complexity should be separated in internal servers.
- External servers implement their own view of the underlying microkernel.
- To construct this view, they use the mechanisms available through the interfaces of the microkernel. Every external server is a separate process that itself represents an application platform.
- Hence, a Microkernel system may be viewed as an application platform that integrates other application platforms.
- Clients communicate with external servers by using the communication facilities provided by the microkernel.

Structure

The Microkernel pattern defines five kinds of participating components:

- Internal servers
- External servers
- Adapters
- Clients
- Microkernel

Microkernel

- represents the main component of the pattern.
- It implements central services such as communication facilities or resource handling.
- Other components build on all or some of these basic services.
- They do this indirectly by using one or more interfaces that comprise the functionality exposed by the microkernel.

Microkernel Component

- A microkernel implements **atomic** services, which we refer to as **mechanisms**.
- These **mechanisms** serve as a fundamental base on which more complex functionality, called **policies**, are constructed.
- The microkernel is also responsible for maintaining **system resources** such as **processes** or **files**.
- It controls and coordinates the access to these resources.

Class	Collaborators
<p>Responsibility</p> <ul style="list-style-type: none"> • Provides core mechanisms. • Offers communication facilities. • Encapsulates system dependencies. • Manages and controls resources. 	<ul style="list-style-type: none"> • Internal Server

Internal Server Component

- Also known as a subsystem.
- They are extensions of the microkernel.
- Only accessible by the microkernel component.
- Extends the functionality provided by the microkernel.
- It is a separate component that offers additional functionality.
- The microkernel invokes the functionality of internal servers via service requests.
- Internal servers can therefore encapsulate some dependencies on the underlying hardware or software system.
- For example, device drivers that support specific graphics cards are good candidates for internal servers.

Class	Collaborators
<p>Internal Server</p> <p>Responsibility</p> <ul style="list-style-type: none"> • Implements additional services. • Encapsulates some system specifics. 	<ul style="list-style-type: none"> • Microkernel

External Server Component

- also known as a personality
- uses the microkernel for implementing its own view of the underlying application domain.
- a view denotes a layer of abstraction built on top of the atomic services provided by the microkernel.
- Different external servers implement different policies for specific application domains.
- External servers expose their functionality by exporting interfaces in the same way as the microkernel itself does.
- Each of these external servers runs in a separate process.
- It receives service requests from client applications, uses the communication facilities provided by the microkernel, interprets these requests, executes the appropriate services and returns results to its clients.
- The implementation of services relies on microkernel mechanisms, so external servers need to access the microkernel's programming interfaces.

<p>Class External Server</p> <p>Responsibility</p> <ul style="list-style-type: none"> • Provides programming interfaces for its clients. 	<p>Collaborators</p> <ul style="list-style-type: none"> • Microkernel
---	---

Client & Adapter

If the external server implements an existing application platform, the corresponding adapter mimics the programming interfaces of that platform.

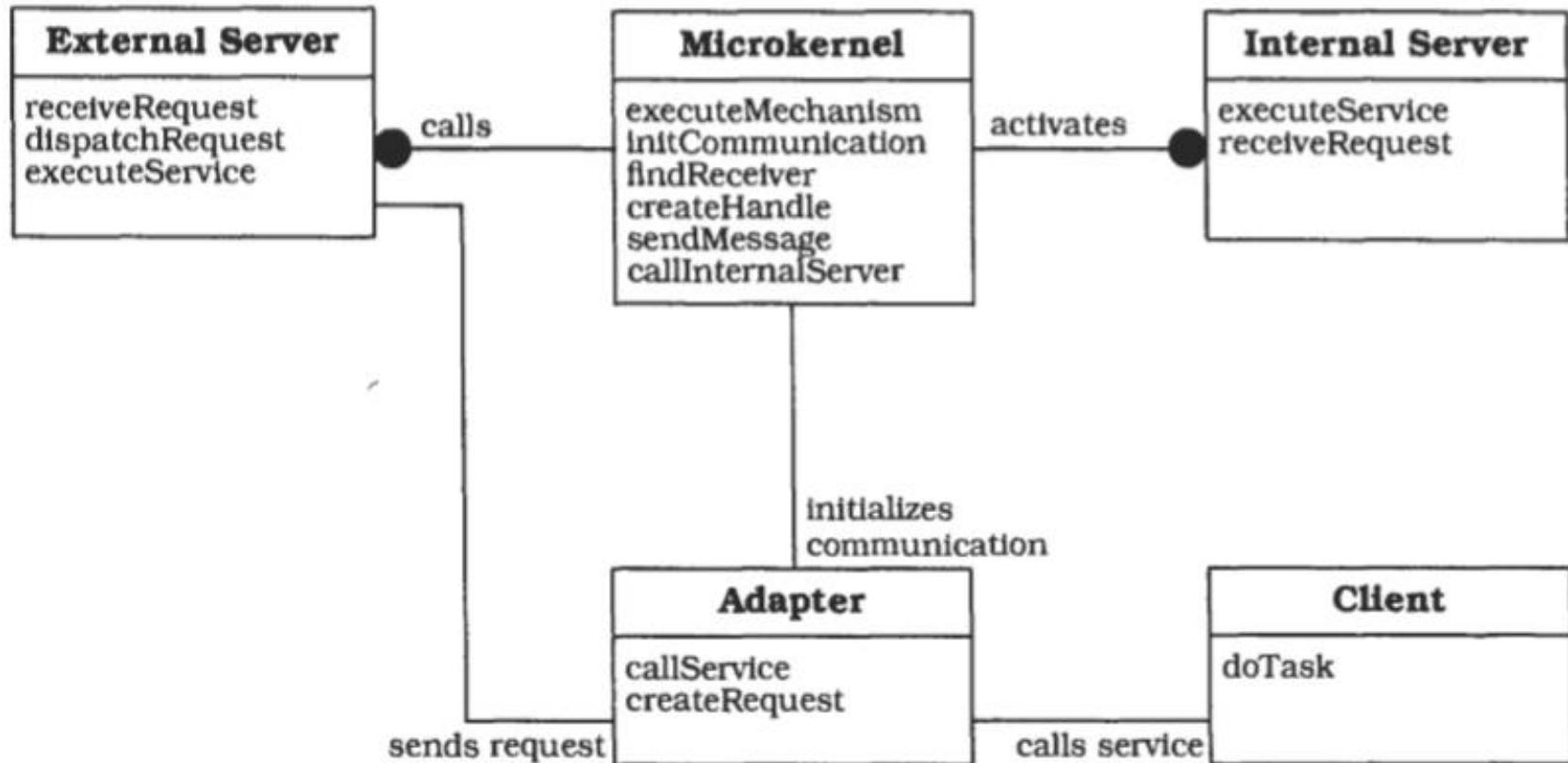
Class	Collaborators
Client	<ul style="list-style-type: none"> • Adapter
Responsibility	
<ul style="list-style-type: none"> • Represents an application. 	

Adapters

- also known as emulators
- represent these interfaces between clients and their external servers
- allow clients to access the services of their external server in a portable way

Class	Collaborators
Adapter	<ul style="list-style-type: none"> • External Server • Microkernel
Responsibility	
<ul style="list-style-type: none"> • Hides system dependencies such as communication facilities from the client. • Invokes methods of external servers on behalf of clients. 	

OMT Diagram



Implementation

1. Analyze the application domain.
2. Analyze external servers
3. Categorize the services.
4. Partition the categories.
5. Find a consistent and complete set of operations and abstractions for every category.
6. Determine strategies for request transmission and retrieval.
7. Structure the microkernel component.
8. specify the programming interface
9. The microkernel is responsible for managing all system resources such as memory blocks, devices or device contexts-a handle to an output area in a graphical user interface implementation.
10. Design and implement the internal servers as separate processes or shared libraries.
11. Implement the external servers
12. Implement the adapters
13. Develop client applications or use existing ones for the ready-to-run Microkernel system

Microkernel System with indirect Client-Server connections.

- A client that wants to send a request or message to an external server asks the microkernel for a communication channel.
- After the requested communication path has been established, client and server communicate with each other indirectly using the microkernel as a message backbone.
- Using this variant leads to an architecture in which all requests pass through the microkernel.
- You can apply it, for example, when security requirements force the system to control all communication between participants.

Variant: Distributed Microkernel System

- A microkernel can also act as a message backbone responsible for sending messages to remote machines or receiving messages from them.
- Every machine in a distributed system uses its own microkernel implementation.
- From the user's viewpoint the whole system appears as a single Microkernel system-the distribution remains transparent to the user.
- A distributed Microkernel system allows you to distribute servers and clients across a network of machines or microprocessors.
- To achieve this the micro kernels in a distributed implementation must include additional services for communicating with each other.

Benefits

-
1. Portability
 2. Flexibility and Extensibility
 3. Separation of policy and mechanism
 4. Scalability
 5. Reliability
 6. Transparency

liability

1. Performance.
2. Complexity of design and implementation.



Adaptable Systems: Reflection

Adaptable Systems: Reflection



- The Reflection architectural pattern provides a mechanism for changing structure and behaviour of software systems dynamically.
- It supports the modification of fundamental aspects, such as type structures and function call mechanisms.
- In this pattern, an application is split into two parts.
 - A meta level provides information about selected system properties and makes the software self-aware.
 - A base level includes the application logic.
- Its implementation builds on the meta level.
- Changes to information kept in the meta level affect subsequent base-level behaviour.

Context

- Building systems that support their own modification a priori.

Problem

- Software systems evolve over time.
- They must be open to modifications in response to changing technology and requirements.
- Designing a system that meets a wide range of different requirements *a priori* can be an overwhelming task.
- A better solution is to specify an architecture that is open to modification and extension.
- The resulting system can then be adapted to changing requirements on demand.
- In other words, we want to design for change and evolution.

Forces

- Changing software is tedious, error prone, and often expensive.
- Adaptable software systems usually have a complex inner structure.
- The more techniques that are necessary for keeping a system changeable, such as parameterization, subclassing, mix-ins, or even copy and paste, the more awkward and complex its modification becomes.
- Changes can be of any scale, from providing shortcuts for commonly-used commands to adapting an application framework for a specific customer.
- Even fundamental aspects of software systems can change, for example the communication mechanisms between components.

Solution

- Make the software self-aware, and make selected aspects of its structure and behaviour accessible for adaptation and change.
- This leads to an architecture that is split into two major parts:
 - a meta level and
 - a base level.

Meta Level

- The meta level provides a self-representation of the software to give it knowledge of its own structure and behavior, and consists of so-called metaobjects.
- Metaobjects encapsulate and represent information about the software.
- Examples include type structures, algorithms, or even function call mechanisms.

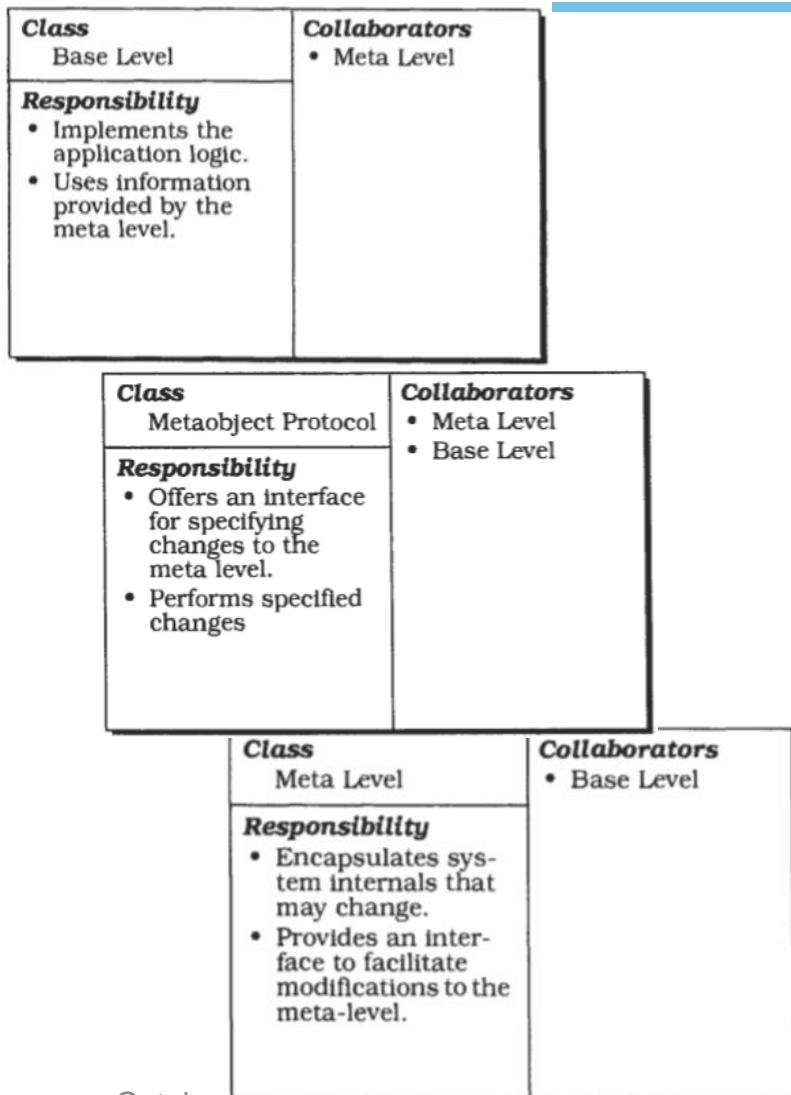
Base Level

- The base level defines the application logic.
- Its implementation uses the metaobjects to remain independent of those aspects that are likely to change.
- For example, base-level components may only communicate with each other via a metaobject that implements a specific user-defined function call mechanism.
- Changing this metaobject changes the way in which base-level components communicate, but without modifying the base-level code.

metaobject protocol (MOP)

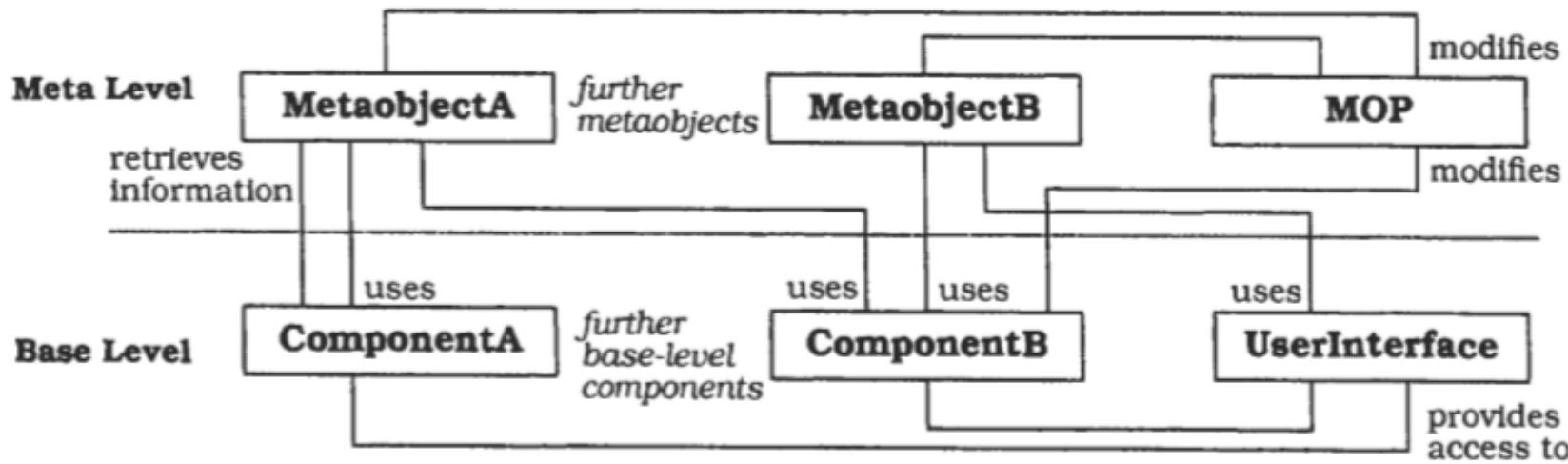
- An interface is specified for manipulating the metaobjects.
- It is called the metaobject protocol (MOP), and allows clients to specify particular changes, such as modification of the function call mechanism metaobject mentioned above.
- The metaobject protocol itself is responsible for checking the correctness of the change specification, and for performing the change.
- Every manipulation of metaobjects through the metaobject protocol affects subsequent base-level behavior, as in the function call mechanism example.

Structure



- The meta level consists of a set of metaobjects. Each metaobject encapsulates selected information about a single aspect of the structure, behaviour, or state of the base level.
- The base level models and implements the application logic of the software.
- The metaobject protocol (MOP) serves as an external interface to the meta level, and makes the implementation of a reflective system accessible in a defined way.

OMT Diagram



- Since the base-level implementation explicitly builds upon information and services provided by metaobjects, changing them has an immediate effect on the subsequent behaviour of the base level.
- The general structure of a reflective architecture is very much like a Layered system

Implementation

-
1. Define a model of the application.
 2. Identify varying behavior
 3. Identify structural aspects of the system, which, when changed, should not affect the implementation of the base level
 4. Identify system services that support both the variation of application services and the independence of structural details
 5. Define the metaobjects.
 6. Define the metaobject protocol.
 7. Define the base level.

Benefits

1. No explicit modification of source code.
2. Changing a software system is easy
3. Support for many kinds of change

Liabilities

1. Modifications at the meta level may cause damage
2. Increased number of components
3. Lower efficiency.
4. Not all potential changes to the software are supported.
5. Not all languages support refection.

Thank you

Credits: Material sourced from Text Book...

PATTERN-ORIENTED SOFTWARE ARCHITECTURE

A System of Patterns

- Frank Buschmann
- Regine Meunier
- Hans Rohnert
- Peter Sommerlad
- Michael Stal

of Siemens AG, Germany



BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Module 7

Patterns – Part 3

Harvinder S Jabbal
SSZG653 Software Architectures



Client-Server Pattern

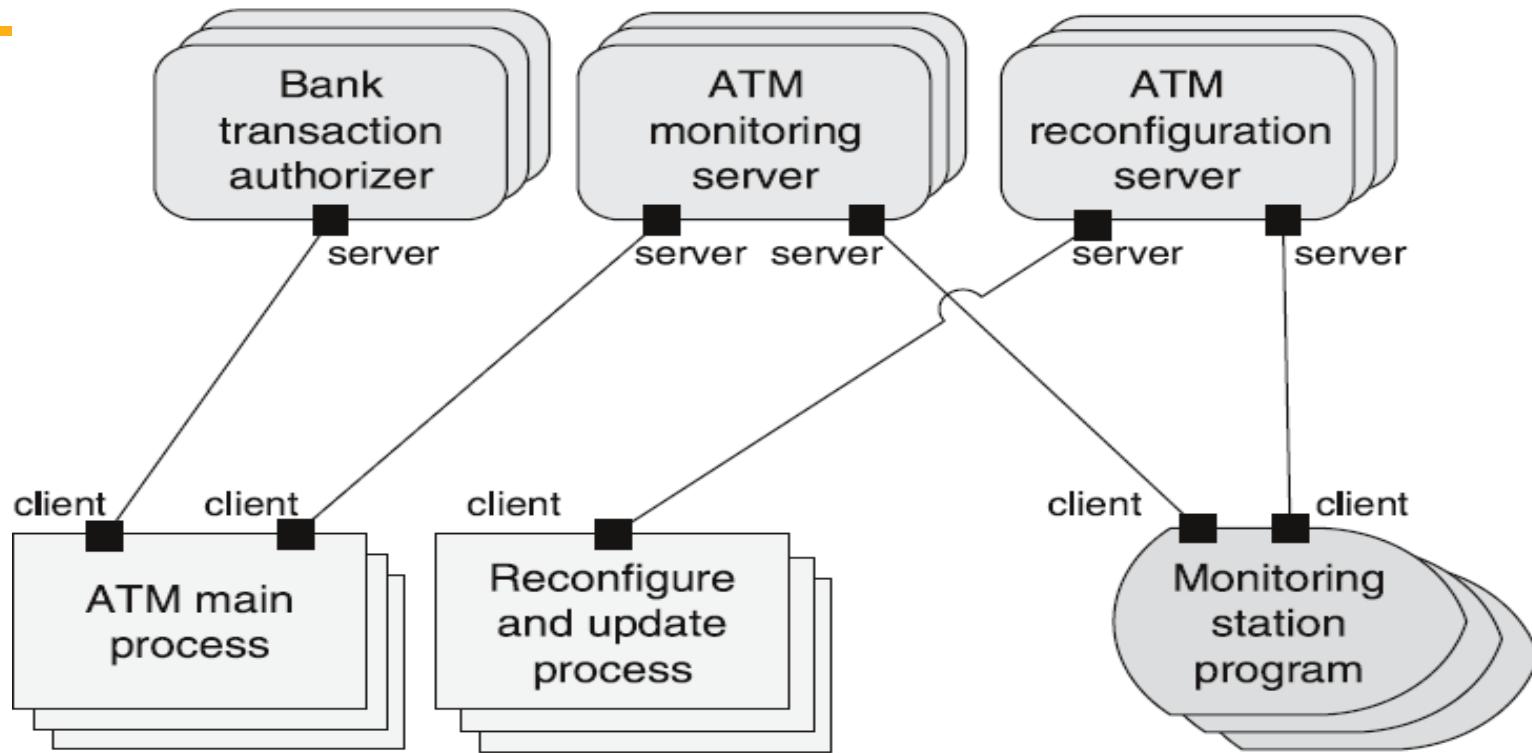
Client-Server Pattern

Context: There are shared resources and services that large numbers of distributed clients wish to access, and for which we wish to control access or quality of service.

Problem: By managing a set of shared resources and services, we can promote modifiability and reuse, by factoring out common services and having to modify these in a single location, or a small number of locations. We want to improve scalability and availability by centralizing the control of these resources and services, while distributing the resources themselves across multiple physical servers.

Solution: Clients interact by requesting services of servers, which provide a set of services. Some components may act as both clients and servers. There may be one central server or multiple distributed ones.

Client-Server Example



Key:

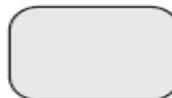


Client



Server

TCP socket connector with client and server ports



FTX server daemon



ATM OS/2 client process



Windows application

Client-Server Solution - 1

Overview: Clients initiate interactions with servers, invoking services as needed from those servers and waiting for the results of those requests.

Elements:

- *Client*, a component that invokes services of a server component. Clients have ports that describe the services they require.
- *Server*: a component that provides services to clients. Servers have ports that describe the services they provide.

Request/reply connector: a data connector employing a request/reply protocol, used by a client to invoke services on a server. Important characteristics include whether the calls are local or remote, and whether data is encrypted.

Client-Server Solution- 2

Relations: The *attachment* relation associates clients with servers.

Constraints:

- Clients are connected to servers through request/reply connectors.
- Server components can be clients to other servers.

Weaknesses:

- Server can be a performance bottleneck.
- Server can be a single point of failure.
- Decisions about where to locate functionality (in the client or in the server) are often complex and costly to change after a system has been built.



Peer-to-Peer Pattern

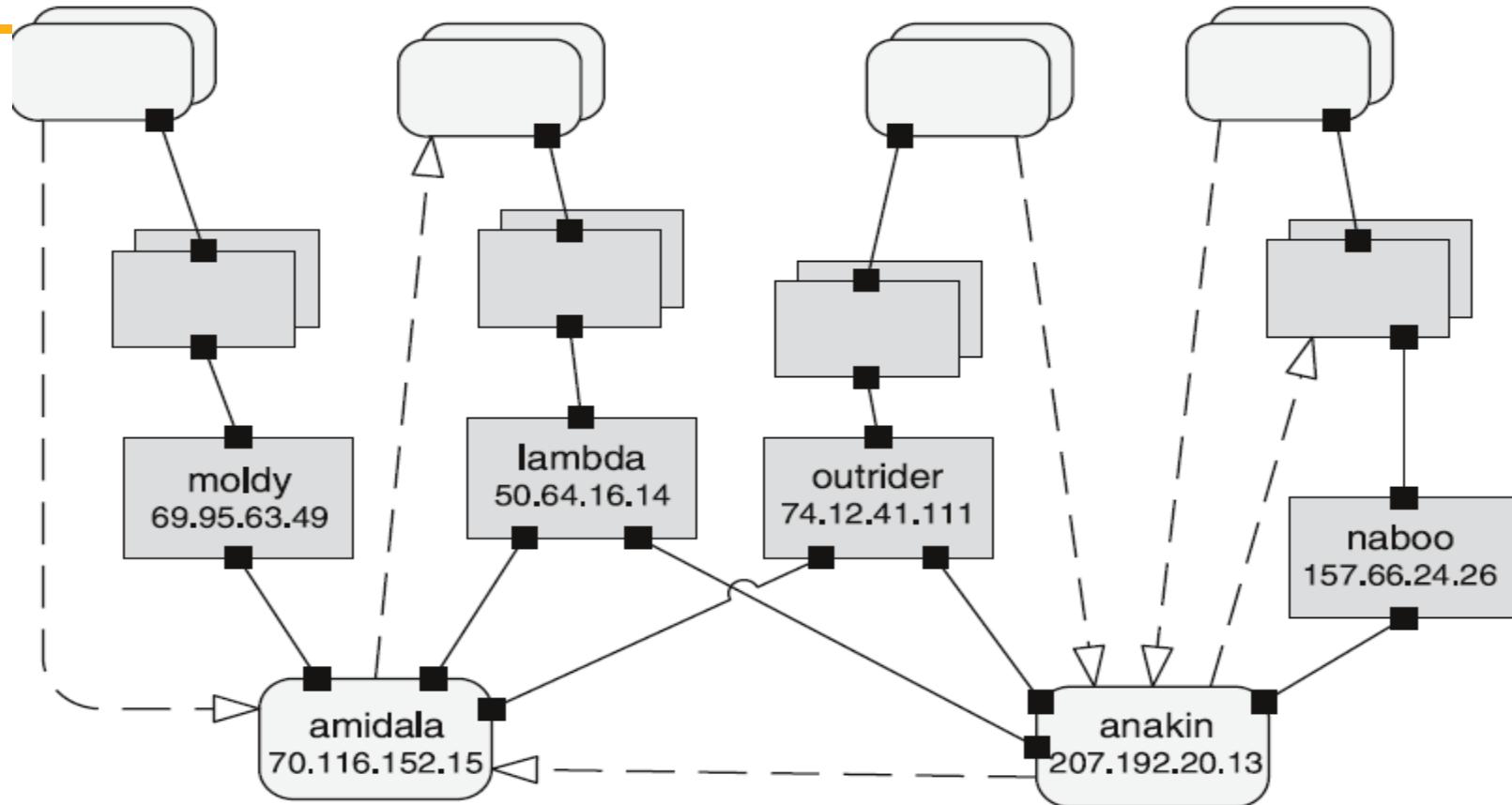
Peer-to-Peer Pattern

Context: Distributed computational entities—each of which is considered equally important in terms of initiating an interaction and each of which provides its own resources—need to cooperate and collaborate to provide a service to a distributed community of users.

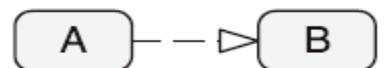
Problem: How can a set of “equal” distributed computational entities be connected to each other via a common protocol so that they can organize and share their services with high availability and scalability?

Solution: In the peer-to-peer (P2P) pattern, components directly interact as peers. All peers are “equal” and no peer or group of peers can be critical for the health of the system. Peer-to-peer communication is typically a request/reply interaction without the asymmetry found in the client-server pattern.

Peer-to-Peer Example



Key:

-  Leaf peer  Gnutella port  Request/reply using Gnutella protocol over TCP or UDP
-  Ultrapeer
-  A  B HTTP file transfer from A to B

Peer-to-Peer Solution - 1

Overview: Computation is achieved by cooperating peers that request service from and provide services to one another across a network.

Elements:

- *Peer*, which is an independent component running on a network node. Special peer components can provide routing, indexing, and peer search capability.
- *Request/reply connector*, which is used to connect to the peer network, search for other peers, and invoke services from other peers. In some cases, the need for a reply is done away with.

Relations: The relation associates peers with their connectors. Attachments may change at runtime.

Peer-to-Peer Solution - 2

Constraints: Restrictions may be placed on the following:

- The number of allowable attachments to any given peer
- The number of hops used for searching for a peer
- Which peers know about which other peers
- Some P2P networks are organized with star topologies, in which peers only connect to supernodes.

Weaknesses:

- Managing security, data consistency, data/service availability, backup, and recovery are all more complex.
- Small peer-to-peer systems may not be able to consistently achieve quality goals such as performance and availability.



Publish-Subscribe Pattern

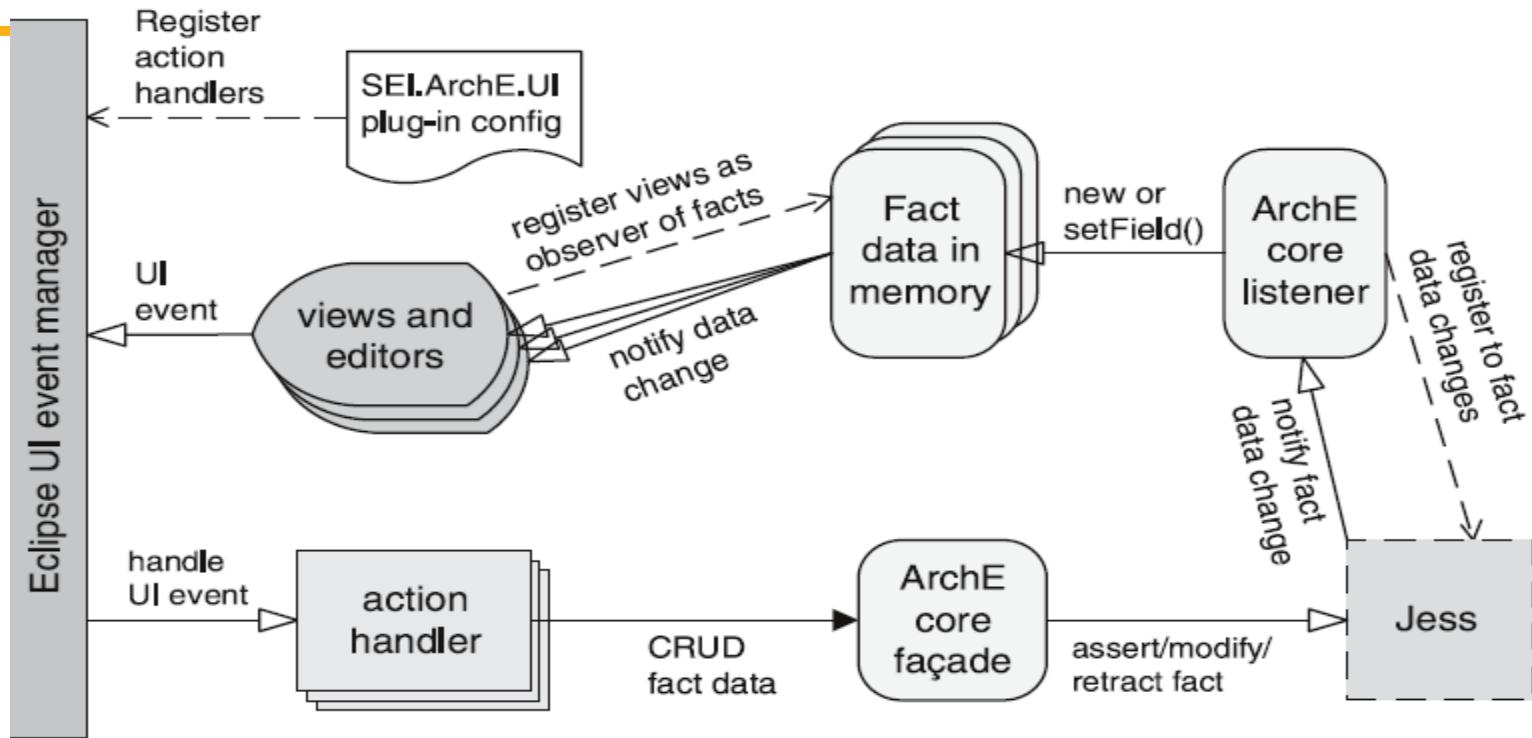
Publish-Subscribe Pattern

Context: There are a number of independent producers and consumers of data that must interact. The precise number and nature of the data producers and consumers are not predetermined or fixed, nor is the data that they share.

Problem: How can we create integration mechanisms that support the ability to transmit messages among the producers and consumers so they are unaware of each other's identity, or potentially even their existence?

Solution: In the publish-subscribe pattern, components interact via announced messages, or events. Components may subscribe to a set of events. Publisher components place events on the bus by announcing them; the connector then delivers those

Publish-Subscribe Example



Key:



Publish-Subscribe Solution – 1

Overview: Components publish and subscribe to events.

When an event is announced by a component, the connector infrastructure dispatches the event to all registered subscribers.

Elements:

- Any *C&C component* with at least one publish or subscribe port.
- *The publish-subscribe connector*, which will have *announce* and *listen* roles for components that wish to publish and subscribe to events.

Relations: The *attachment* relation associates components with the publish-subscribe connector by prescribing which components announce events and which components are registered to receive events.

Publish-Subscribe Solution - 2

Constraints: All components are connected to an event distributor that may be viewed as either a bus—connector—or a component. Publish ports are attached to announce roles and subscribe ports are attached to listen roles.

Weaknesses:

- Typically increases latency and has a negative effect on scalability and predictability of message delivery time.
- Less control over ordering of messages, and delivery of messages is not guaranteed.



Shared-Data Pattern

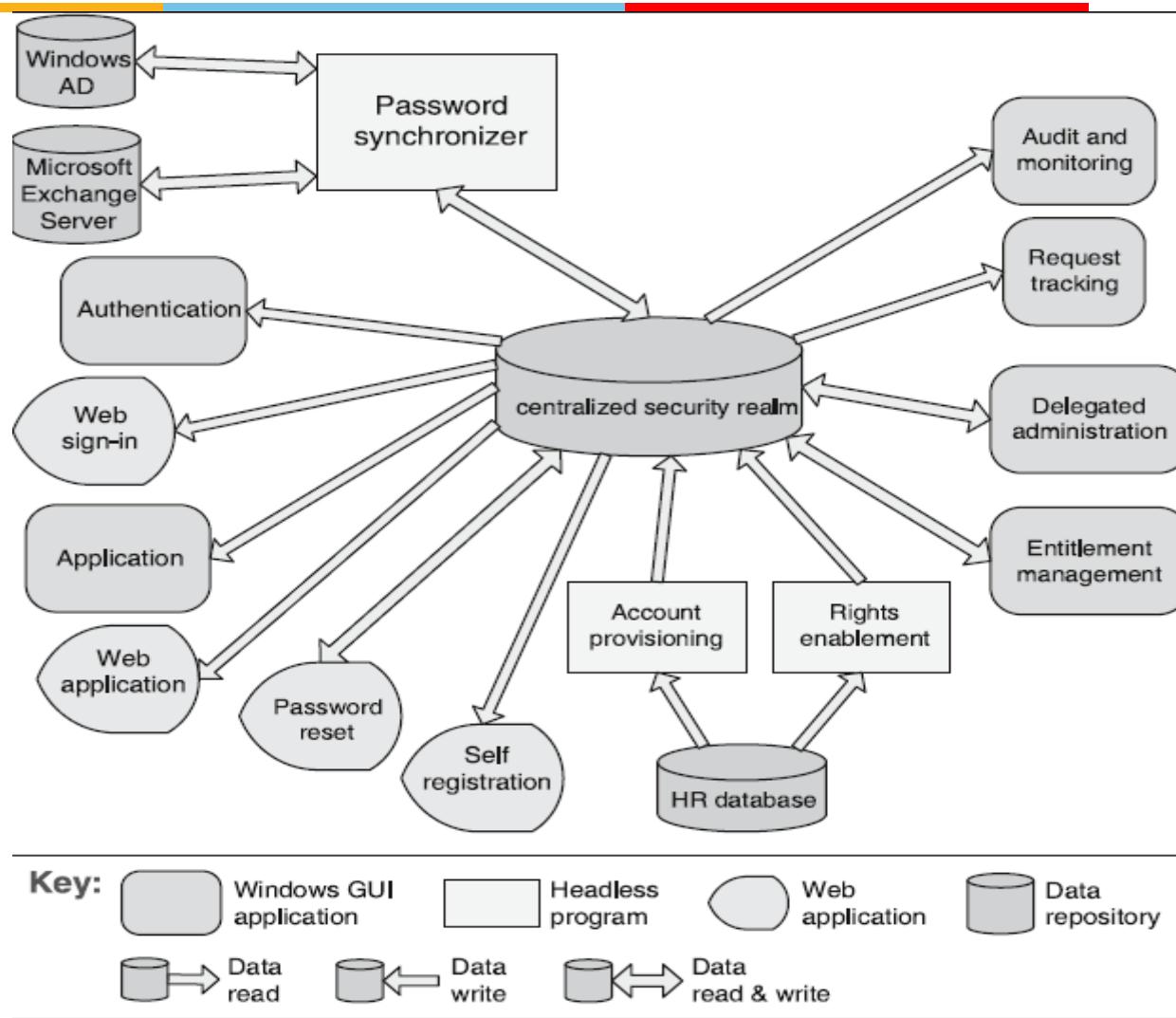
Shared-Data Pattern

Context: Various computational components need to share and manipulate large amounts of data. This data does not belong solely to any one of those components.

Problem: How can systems store and manipulate persistent data that is accessed by multiple independent components?

Solution: In the shared-data pattern, interaction is dominated by the exchange of persistent data between multiple *data accessors* and at least one *shared-data store*. Exchange may be initiated by the accessors or the data store. The connector type is *data reading and writing*.

Shared Data Example



Shared Data Solution - 1

Overview: Communication between data accessors is mediated by a shared data store. Control may be initiated by the data accessors or the data store. Data is made persistent by the data store.

Elements:

- *Shared-data store*. Concerns include types of data stored, data performance-oriented properties, data distribution, and number of accessors permitted.
- *Data accessor component*.
- *Data reading and writing connector*.

Shared Data Solution - 2

Relations: *Attachment* relation determines which data accessors are connected to which data stores.

Constraints: Data accessors interact only with the data store(s).

Weaknesses:

- The shared-data store may be a performance bottleneck.
- The shared-data store may be a single point of failure.
- Producers and consumers of data may be tightly coupled.

Thank you



BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Module 7

Patterns – Part 4

Harvinder S Jabbal
SSZG653 Software Architectures



Map-Reduce Pattern

Map-Reduce Pattern

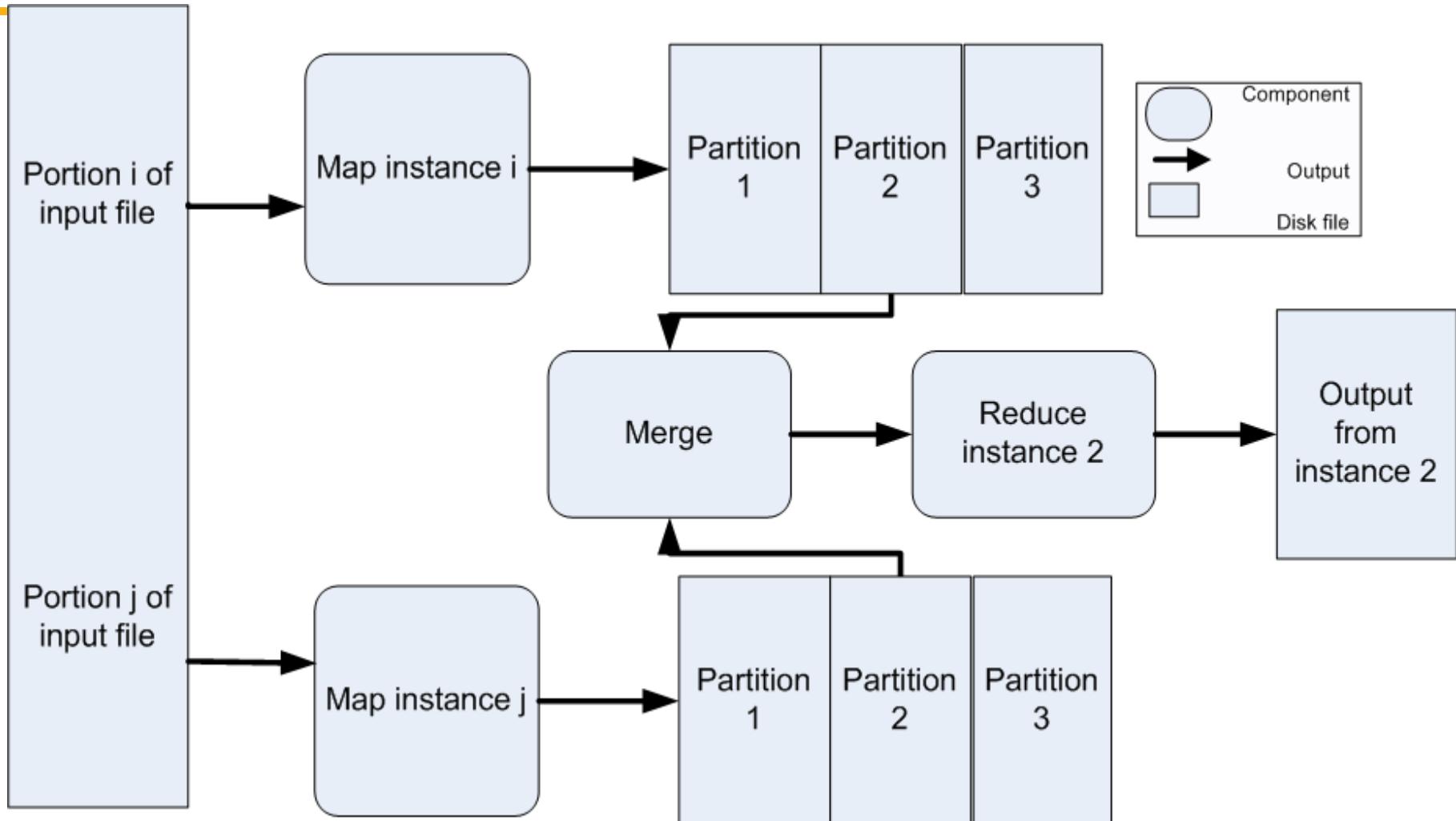
Context: Businesses have a pressing need to quickly analyze enormous volumes of data they generate or access, at petabyte scale.

Problem: For many applications with ultra-large data sets, sorting the data and then analyzing the grouped data is sufficient. The problem the map-reduce pattern solves is to efficiently perform a distributed and parallel sort of a large data set and provide a simple means for the programmer to specify the analysis to be done.

Solution: The map-reduce pattern requires three parts:

- A specialized infrastructure takes care of allocating software to the hardware nodes in a massively parallel computing environment and handles sorting the data as needed.
- A programmer specified component called the map which filters the data to retrieve those items to be combined.
- A programmer specified component called reduce which combines the results of the map

Map-Reduce Example



Map-Reduce Solution - 1

Overview: The map-reduce pattern provides a framework for analyzing a large distributed set of data that will execute in parallel, on a set of processors. This parallelization allows for low latency and high availability. The map performs the extract and transform portions of the analysis and the reduce performs the loading of the results.

Elements:

- Map is a function with multiple instances deployed across multiple processors that performs the extract and transformation portions of the analysis.
- Reduce is a function that may be deployed as a single instance or as multiple instances across processors to perform the load portion of extract-transform-load.
- The infrastructure is the framework responsible for deploying map and reduce instances, shepherding the data between them, and detecting and recovering from failure.

Map-Reduce Solution - 2

Relations:

- Deploy on is the relation between an instance of a map or reduce function and the processor onto which it is installed.
- Instantiate, monitor, and control is the relation between the infrastructure and the instances of map and reduce.

Constraints:

- The data to be analyzed must exist as a set of files.
- Map functions are stateless and do not communicate with each other.
- The only communication between map reduce instances is the data emitted from the map instances as `<key, value>` pairs.

Weaknesses:

- If you do not have large data sets, the overhead of map-reduce is not justified.
- If you cannot divide your data set into similar sized subsets, the advantages of parallelism are lost.
- Operations that require multiple reduces are complex to orchestrate.



Multi-Tier Pattern

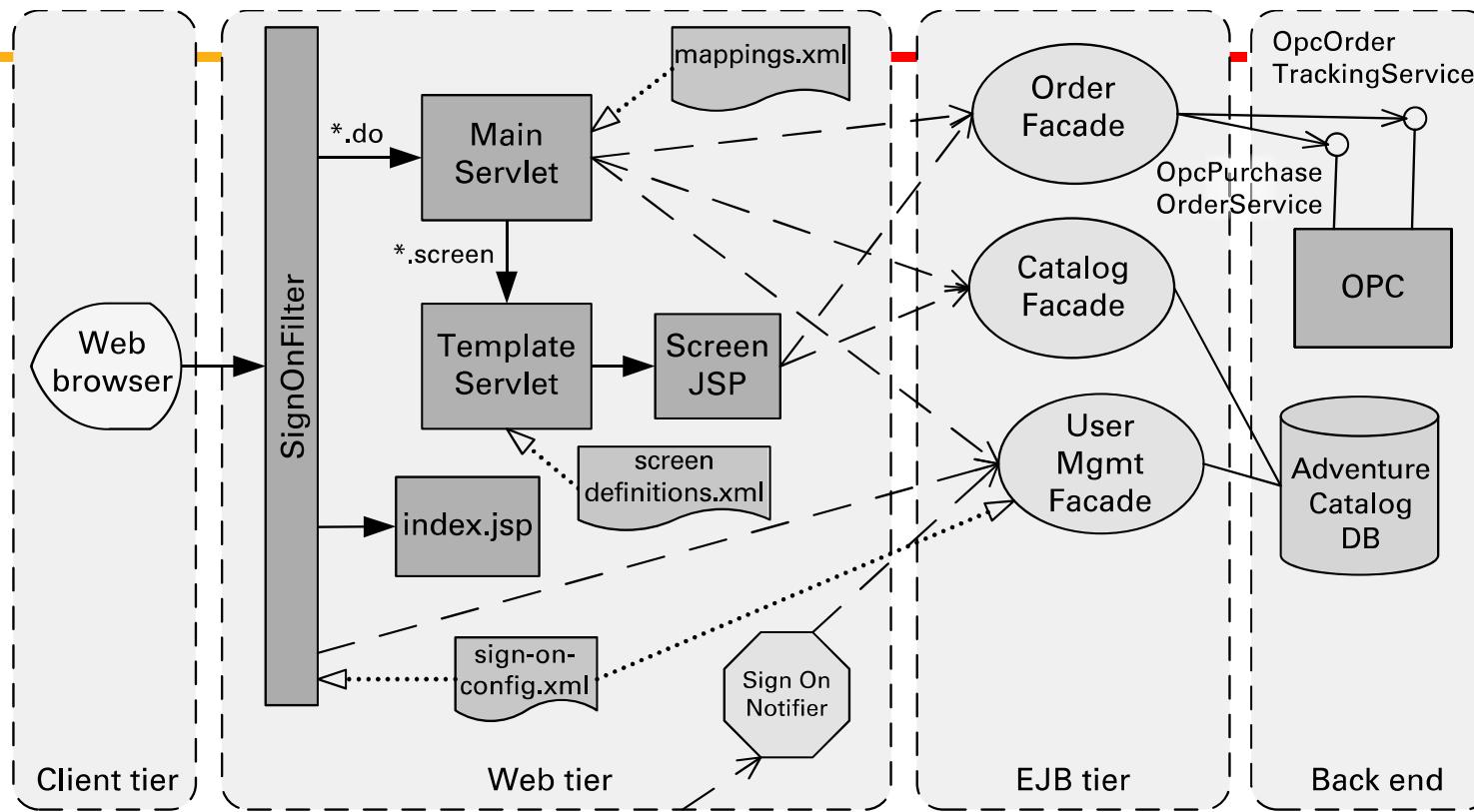
Multi-Tier Pattern

Context: In a distributed deployment, there is often a need to distribute a system's infrastructure into distinct subsets.

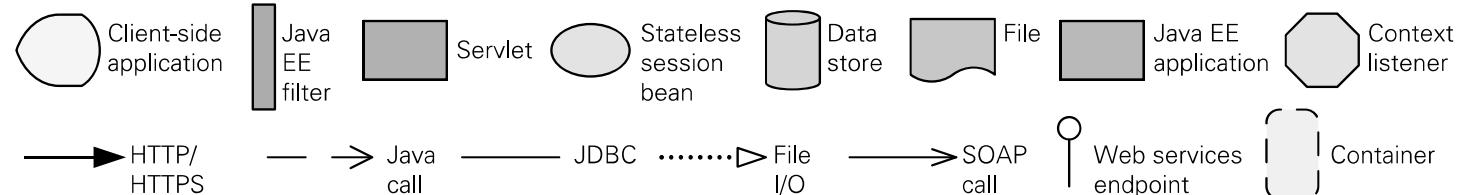
Problem: How can we split the system into a number of computationally independent execution structures—groups of software and hardware—connected by some communications media?

Solution: The execution structures of many systems are organized as a set of logical groupings of components. Each grouping is termed a tier.

Multi-Tier Example



Key



Multi-Tier Solution

Overview: The execution structures of many systems are organized as a set of logical groupings of components. Each grouping is termed a *tier*.

Elements:

- *Tier*, which is a logical grouping of software components.

Relations:

- *Is part of*, to group components into tiers.
- *Communicates with*, to show how tiers and the components they contain interact with each other.
- *Allocated to*, in the case that tiers map to computing platforms.

Constraints: A software component belongs to exactly one tier.

Weaknesses: Substantial up-front cost and complexity.



Tactics and Patterns

Relationships Between Tactics and Patterns



Patterns are built from tactics; if a pattern is a molecule, a tactic is an atom.

MVC, for example utilizes the tactics:

- Increase semantic coherence
- Encapsulation
- Use an intermediary
- Use run time binding

Tactics Augment Patterns

Patterns solve a specific problem but are neutral or have weaknesses with respect to other qualities.

Consider the broker pattern

- May have performance bottlenecks
- May have a single point of failure

Using tactics such as

- Increase resources will help performance
- Maintain multiple copies will help availability

Tactics and Interactions

Each tactic/pattern has pluses (its reason for being) and minuses – side effects.

Use of tactics can help alleviate the minuses.

But nothing is free...

Tactics and Interactions - 2

A common tactic for detecting faults is Ping/Echo.

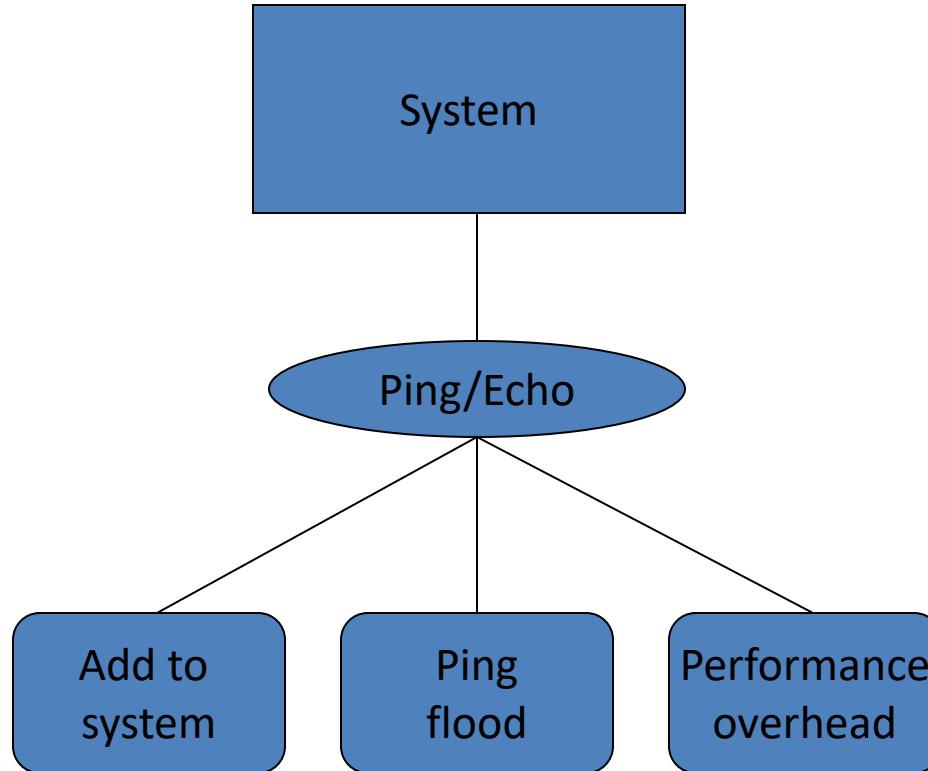
Common side-effects of Ping/Echo are:

security: how to prevent a ping flood attack?

performance: how to ensure that the performance overhead of ping/echo is small?

modifiability: how to add ping/echo to the existing architecture?

Tactics and Interactions - 3



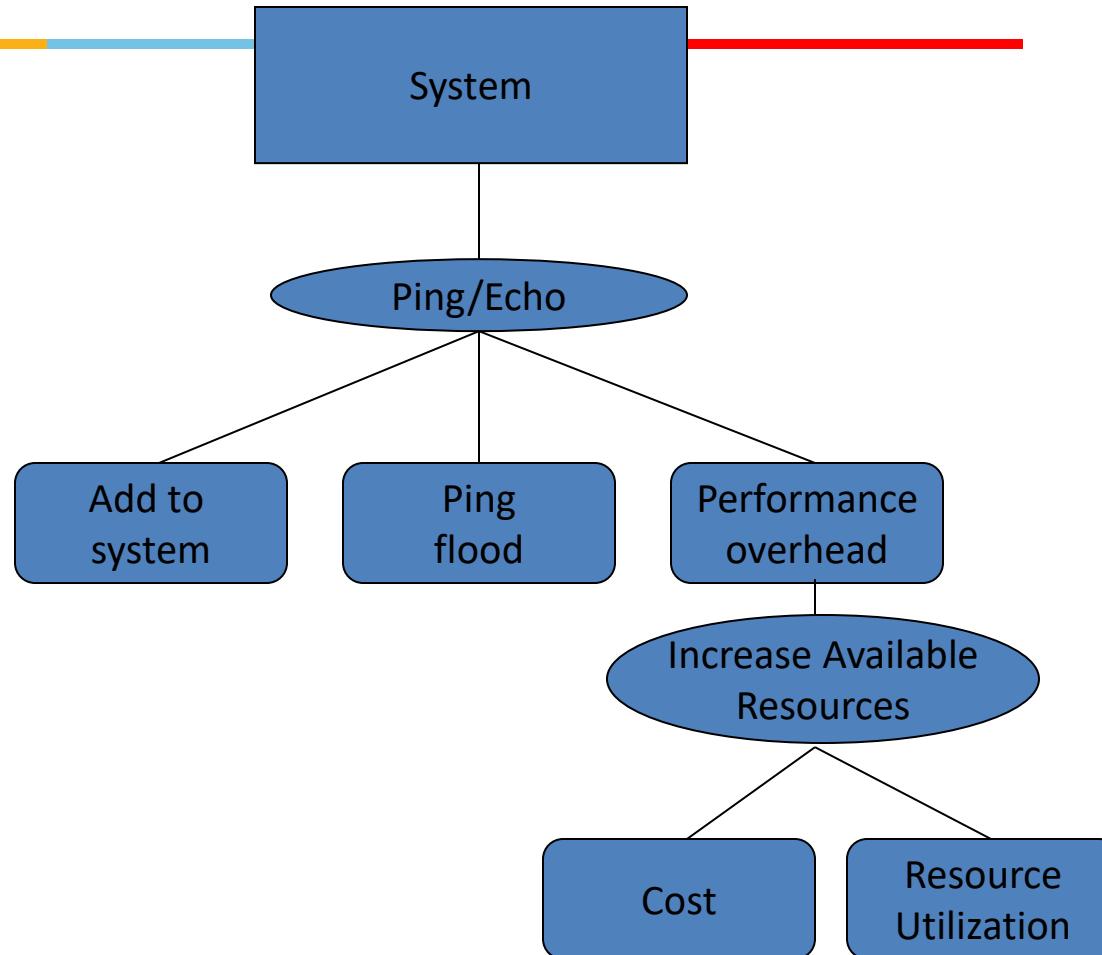
Tactics and Interactions - 4

A tactic to address the performance side-effect is “Increase Available Resources”.

Common side effects of Increase Available Resources are:
cost: increased resources cost more

performance: how to utilize the increase resources
efficiently?

Tactics and Interactions - 5



Tactics and Interactions - 6

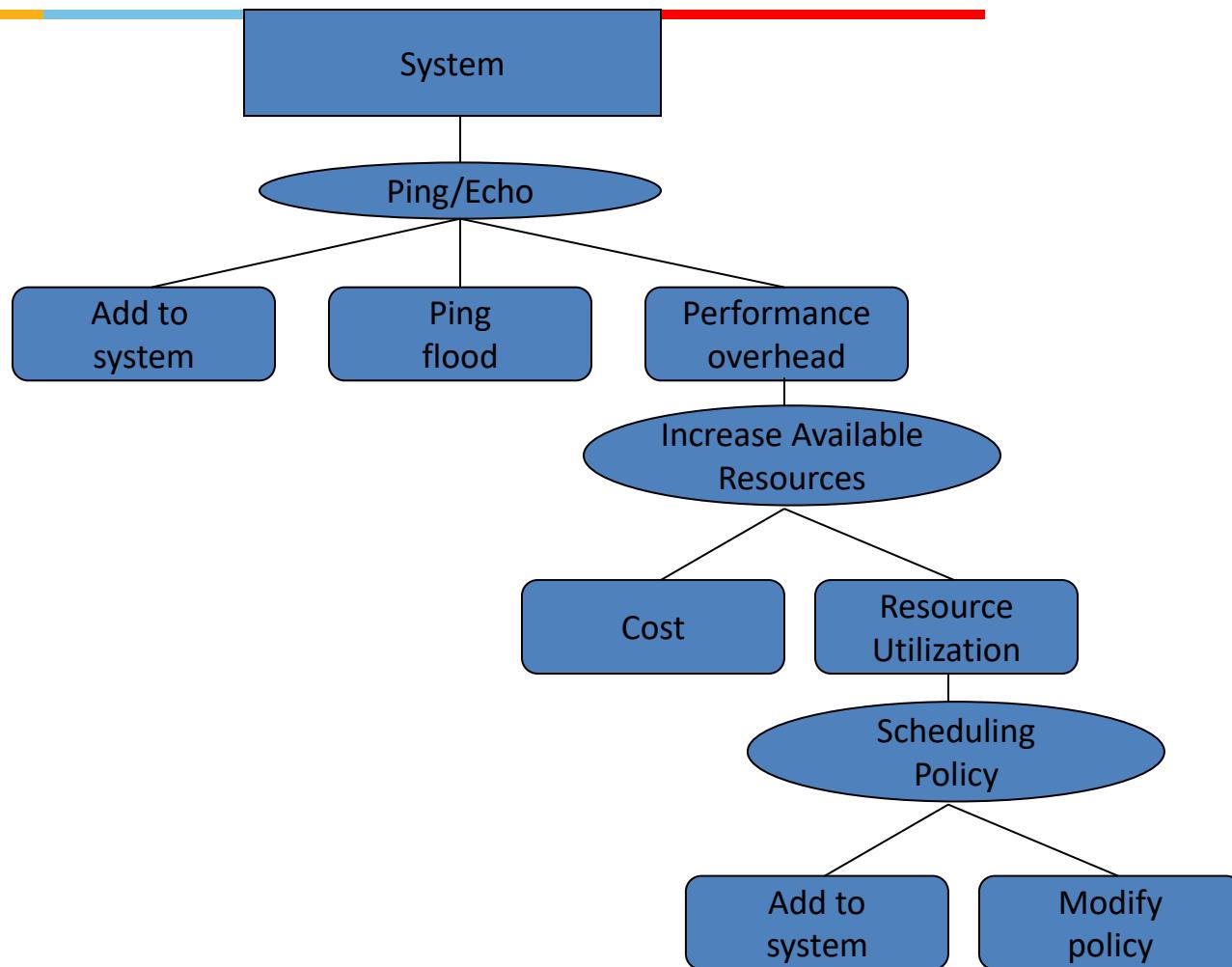
A tactic to address the efficient use of resources side-effect is “Scheduling Policy”.

Common side effects of Scheduling Policy are:

modifiability: how to add the scheduling policy to the existing architecture

modifiability: how to change the scheduling policy in the future?

Tactics and Interactions - 7



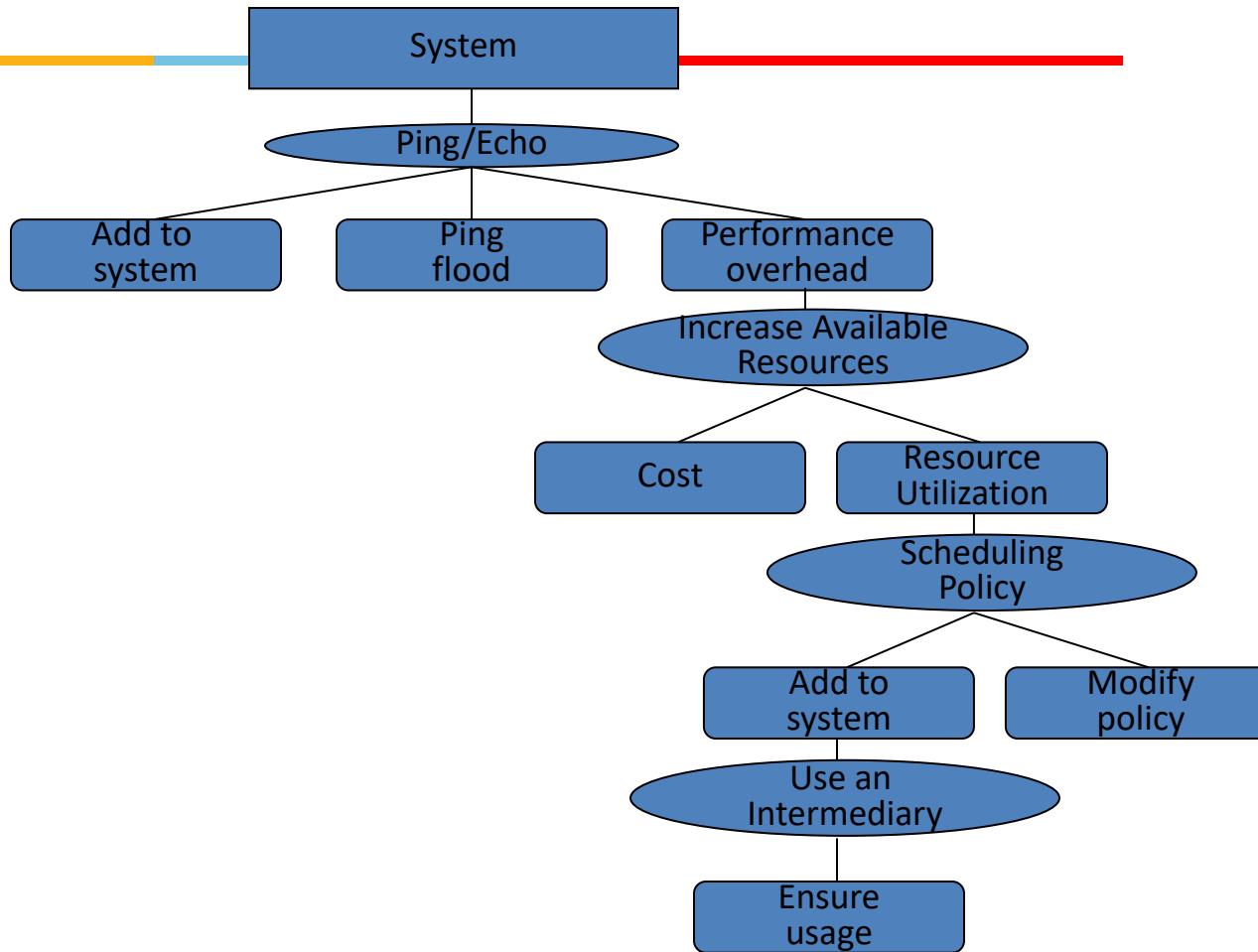
Tactics and Interactions - 8

A tactic to address the addition of the scheduler to the system is “Use an Intermediary”.

Common side effects of Use an Intermediary are:

modifiability: how to ensure that all communication passes through the intermediary?

Tactics and Interactions - 9



Tactics and Interactions – 10.

A tactic to address the concern that all communication passes through the intermediary is “Restrict Communication Paths”.

Common side effects of Restrict Communication Paths are: performance: how to ensure that the performance overhead of the intermediary are not excessive?

Note: this design problem has now become recursive!

How Does This Process End?

Each use of tactic introduces new concerns.

Each new concern causes new tactics to be added.

Are we in an infinite progression?

No. Eventually the side-effects of each tactic become small enough to ignore.

Summary

An architectural pattern

- is a package of design decisions that is found repeatedly in practice,
- has known properties that permit reuse, and
- describes a *class* of architectures.

Tactics are simpler than patterns

Patterns are underspecified with respect to real systems so they have to be augmented with tactics.

- Augmentation ends when requirements for a specific system are satisfied.



BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Module 8

Part 1

Architectures for the Cloud-1

Harvinder S Jabbal
SE ZG651/ SS ZG653 Software Architectures



Architectures for the Cloud

Chapter Outline

- Basic Cloud Definitions
- Service Models and Deployment Options
- Economic Justification

Basic Cloud Definitions (from NIST)



- *On-demand self-service*. A resource consumer can unilaterally provision computing services, such as server time and network storage, as needed automatically without requiring human interaction with each service's provider.
- *Ubiquitous network access*. Cloud services and resources are available over the network and accessed through standard networking mechanisms that promote use by a heterogeneous collection of clients.
- *Resource pooling*. The cloud provider's computing resources are pooled.
- *Location independence*. The location of the resources need not be of concern to the consumer of the resources.
- *Rapid elasticity*. Capabilities can be rapidly and elastically provisioned.
- *Measured service*. Resource usage can be monitored, controlled, and reported so that consumers of the services are billed only for what they use.
- *Multi-tenancy*. Applications and resources can be shared among multiple consumers who are unaware of each other.

Basic Service Models

- **Software as a Service (SaaS)**. The consumer in this case is an end user. The consumer uses applications that happen to be running on a cloud. E.g. mail services or data storage services.
- **Platform as a Service (PaaS)**. The consumer in this case is a developer or system administrator. The consumer deploys applications onto the cloud infrastructure using programming languages and tools supported by the provider.
- **Infrastructure as a Service (IaaS)**. The consumer in this case is a developer or system administrator. The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications.

Deployment Models

- *Private cloud*. The cloud infrastructure is owned solely by a single organization and operated solely for applications owned by that organization.
- *Public cloud*. The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.
- *Community cloud*. The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns.
- *Hybrid cloud*. The cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities.

Economic Justification

- Economies of scale
- Utilization of equipment
- Multi-tenancy

Economies of Scale

- Large data centers are cheaper to operate (per unit measure) than small data centers.
- *Large* in this context means 100,000+ servers
- *Small* in this context means <10,000 servers.

Reasons for Economies of Scale



- *Cost of power.* The cost of electricity to operate a data center currently is 15 to 20 percent of the total cost of operation.
- Per-server power costs are lower in large data centers
 - Sharing of items such as racks and switches.
 - Negotiated prices. Large power users can negotiate significant discounts.
 - Geographic choice. Large data centers can be located where power costs are lowest.
 - Acquisition of cheaper power sources such as wind farms and rooftop solar energy.
- *Infrastructure labor costs. More efficient utilization of system administrators*
 - Small data center administrators service ~150 servers.
 - Large data center administrators service >1000 servers.

More Reasons for Economies of Scale



- *Security and reliability.* Maintaining a given level of security, redundancy, and disaster recovery essentially requires a fixed level of investment. Larger data centers can amortize that investment over their larger number of servers.
- *Hardware costs.* Operators of large data centers can get discounts on hardware purchases of up to 30 percent over smaller buyers.

Utilization of Equipment

- Use of virtualization technology allows for easy co-location of distinct applications and their associated operating systems on the same server hardware. The effect of this co-location is to increase the utilization of servers.
- Variations in workload can be managed to increase utilization.
 - *Random access*. End users may access applications randomly. The more likely that the randomness of their accesses will end up imposing a uniform load on the server.
- *Time of day*.
 - Co-locate those services that are workplace related with those that are consumer related.
 - Consider time differences among geographically distinct locations.
- *Time of year*. Consider yearly fluctuations in demand.
 - Holidays, tax preparation season
- *Resource usage patterns*. Co-locate heavier CPU services with heavier I/O services.
- *Uncertainty*. Consider spikes in usage.

Multi-tenancy

- Some applications such as salesforce.com use a single application for multiple different consumers.
- This reduces costs by reducing costs of
 - Help desk support
 - Upgrade once, simultaneously, for all consumers
 - Single version of the software from a development and maintenance perspective.

Summary

- The cloud provides a new platform for applications with some different characteristics.



BITS Pilani
Pilani|Dubai|Goa|Hyderabad

October 28, 2022

Module 8

Part 2

Architectures for the Cloud -2

Harvinder S Jabbal
SE ZG651/ SS ZG653 Software Architectures



Architectures for the Cloud - 2

Session Outline

- Base Mechanisms
- Sample Technologies
- Architecting in a Cloud Environment
- Summary

Basic Mechanisms

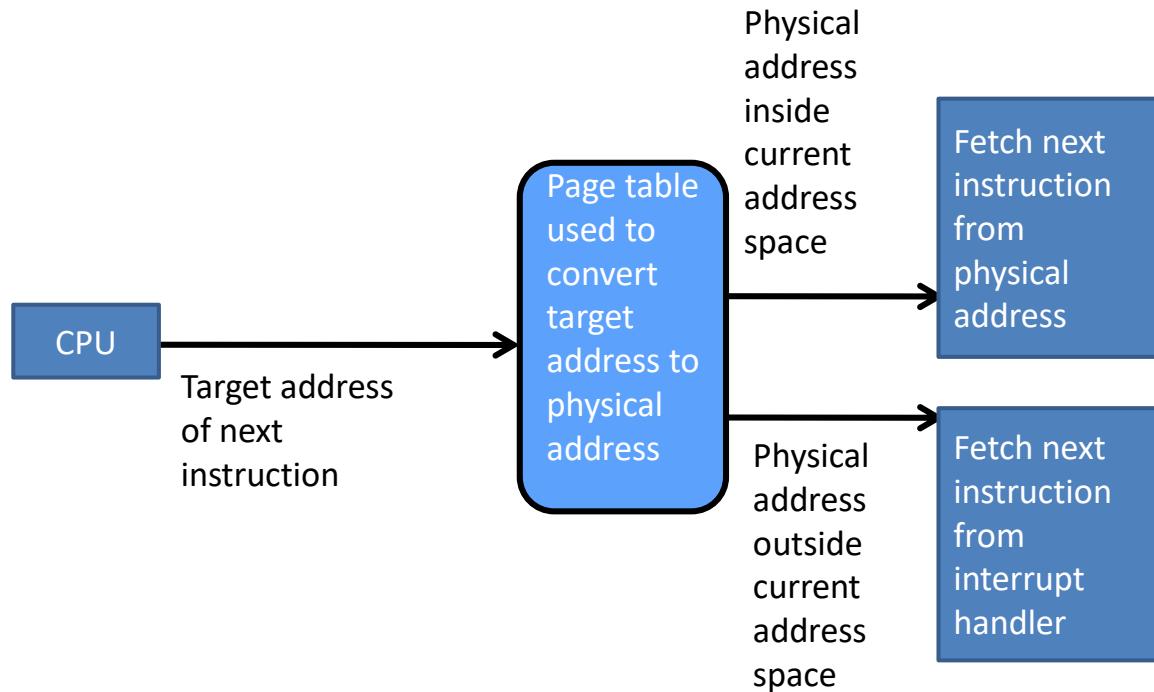
- Hypervisor
- Virtual Machine
- File system
- Network

Virtual Memory Page Table



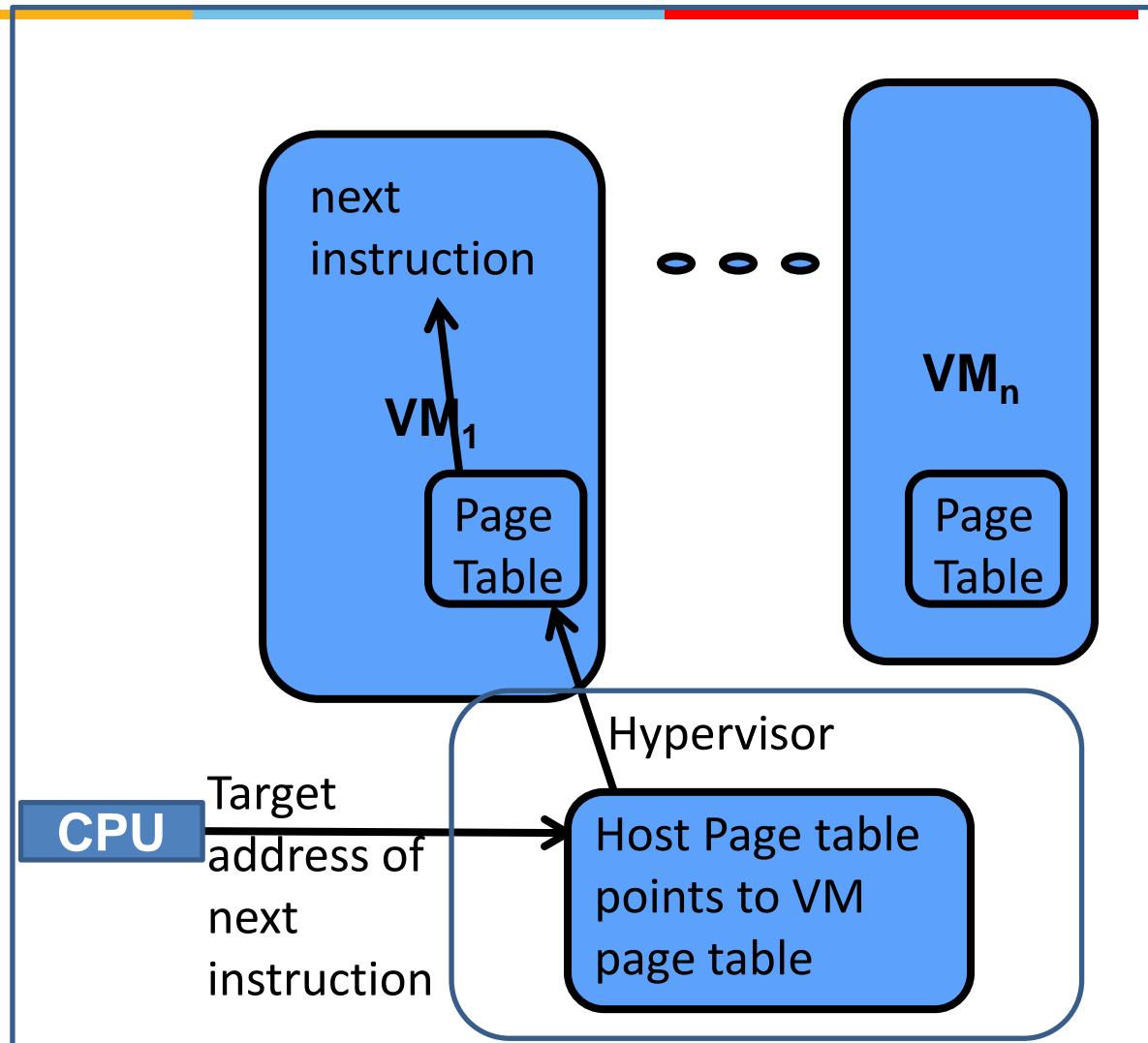
Virtual Memory Page Table

Virtual memory for non-virtualized application



Hypervisor Manages Virtualization

Host Server



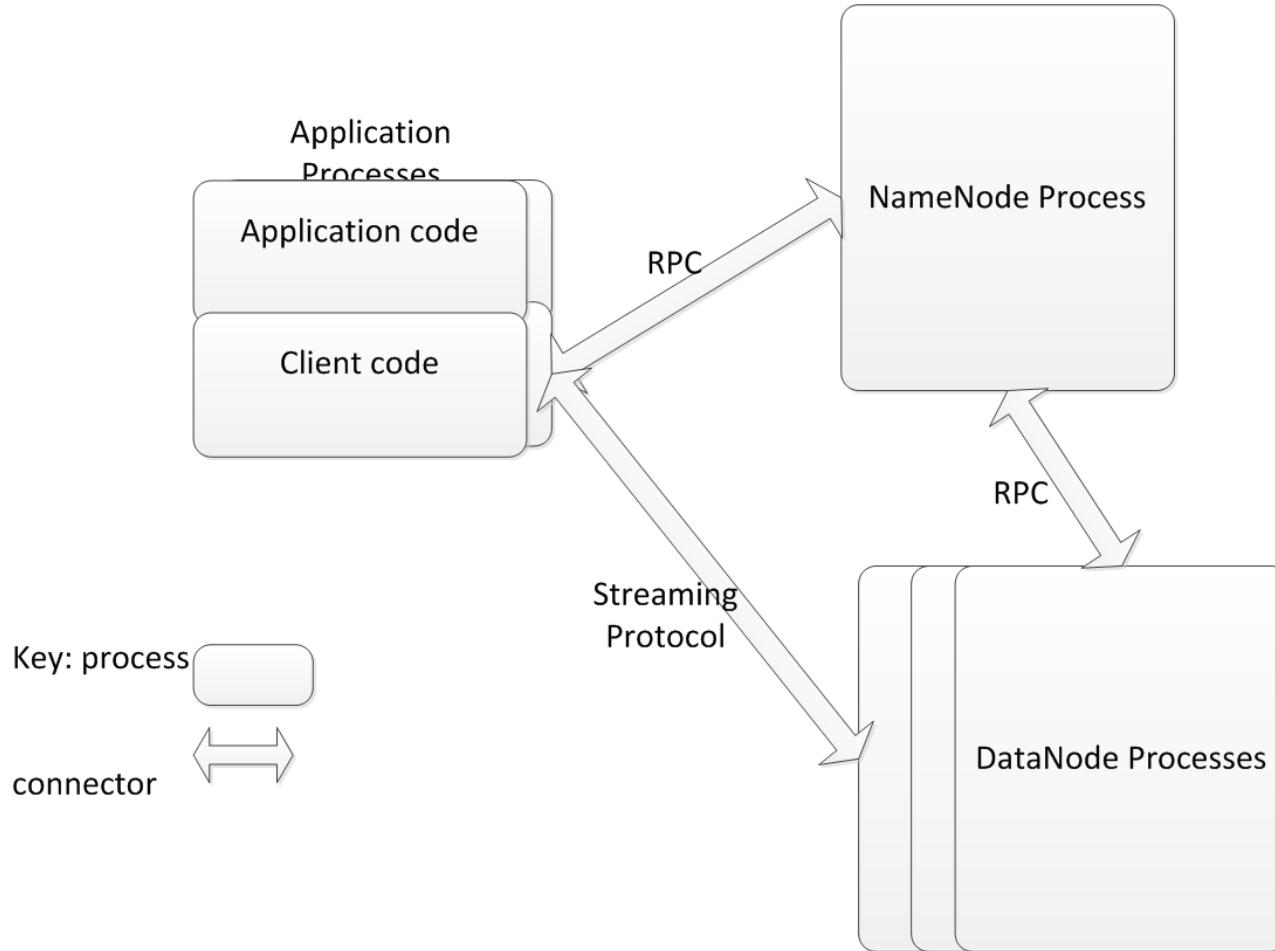
Virtual Machine

- A virtual machine has an address space isolated from any other virtual machine.
- Looks like a bare metal machine from the application perspective.
- Assigned an IP address and has network capability.
- Can be loaded with any operating system or applications that can execute on the processor of the host machine.

File System

-
- Each virtual machine has access to a file system.
 - We will present HDFS (Hadoop Distributed File System)
– a widely used open source cloud file system.
 - We describe how HDFS uses redundancy to ensure availability.

HDFS Components



HDFS Write – Sunny Day Scenario



- Application writes as to any file system
- Client buffers until it gets 64K block
- Client informs NameNode it wishes to write a new block
- NameNode returns list of three DataNodes to hold block
- Client sends block to first DataNode and informs DataNode of other two replicas.
- First DataNode writes block and sends it to second DataNode. Second DataNode writes block and sends it to last DataNode.
- Each DataNode reports to client when it has completed its write
- Client commits write to NameNode when it has heard from all three DataNodes.

HDFS Write – Failure Cases

- Client fails
 - Application detects and retries
 - Write is not complete until committed by Client
- NameNode fails
 - Backup NameNode takes over
 - Log file maintained to avoid losing information
 - DataNodes maintain true list of which blocks they each have
 - Client detects and retries
- DataNode fails
 - Client (or earlier DataNode in pipeline) detects and asks NameNode for different DataNode.
- Since each block is replicated three times, a failure in a DataNode does not lose any data.

Network

- Every Virtual Machine is assigned an IP address.
- Every message using TCP/IP includes IP address in header.
- Gateway for cloud can adjust IP address for various purposes.

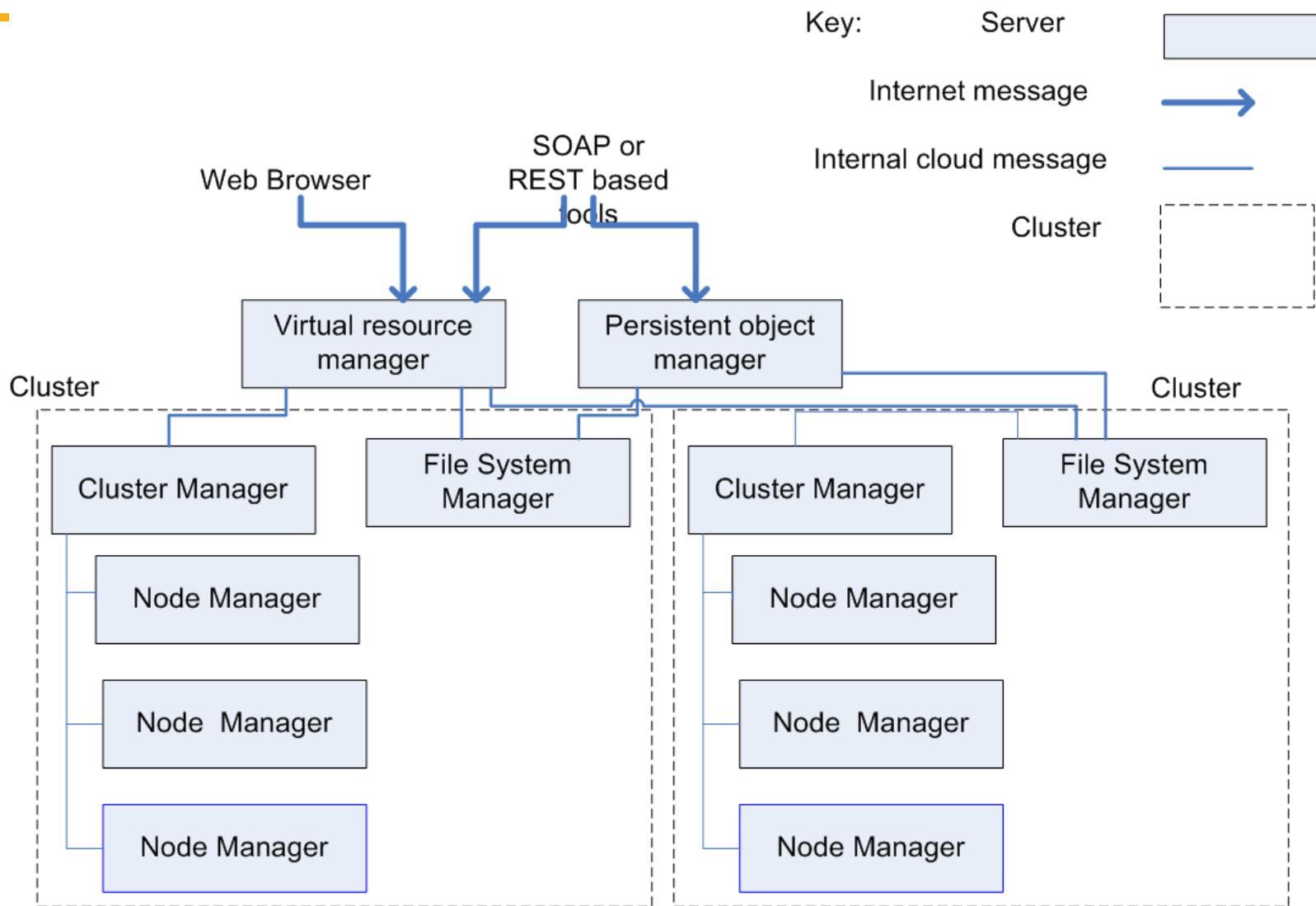
Sample Technologies

- IaaS
- PaaS
- DataBases

IaaS

- An arrangement of servers that manages the base technologies.
 - Servers are arranged in clusters
 - May be thousands of servers in a cluster
 - Some servers are used as the infrastructure of the IaaS
 - Every server has a hypervisor as its base.

IaaS Architecture



IaaS Architecture Components

- Cluster Manager responsible for managing each cluster
- Persistent Object Manager manages persistence
- Virtual Resource Manager manages other resources. It acts as a gateway for messages.
- The File System Manager is similar to HDFS. It manages the network wide file system.

Services Provided by IaaS

- Automatic reallocation of IP addresses in the case of a failure of the underlying virtual machine instance.
- Automatic Scaling. Create or delete new virtual machines depending on load.

PaaS

- Provides an integrated stack for developer.
- E.g. LAMP stack
 - Linux, Apache, MySQL, Python
- The developer writes code in Python and the PaaS manages assignment to underlying layers of the stack.

Databases

- Why relational databases came into question
 - Massive amounts of data are collected from web systems. Much of this data is processed sequentially and so RDBMSs introduce overhead, especially during creation and maintenance.
 - The CAP Theorem shows that it is not possible to simultaneously achieve consistency, availability, and partitioning.
 - The relational model is not the best model for some applications.
- Caused the introduction of new data models
 - Key-value
 - Document centric

Key Value – HBase

- One column designated as a key. The others are all values
- No schema so data can have key + any other values. The values are identified by their variable name.
- Data values are also time stamped
 - Hbase does not support transactions. Time stamps are used to detect collisions after the fact.

Document Centric – MongoDB

Stores objects rather than data

Access data through containing object

Objects can also contain links to other objects

No concept of primary or secondary index. A field is indexed or it is not.

What is Omitted From These DBs



Transactions. No locking is performed. The application must detect interference with other users.

Schemas. No predefined schemas. The application must use correct name.

Consistency. The CAP theorem says something must give. Usually consistency is replaced by “eventual consistency”

Normalization and Joins. Performing a join requires that the join field is indexed. Because there is not a guaranteed index field, joins cannot be performed. This means normalization of tables is not supported.

Architecting in a Cloud Environment



- Quality attributes that are different in a cloud
 - Security
 - Performance
 - Availability

Security

- Multi-tenancy introduces additional concerns over non-cloud environments.
 - Inadvertent information sharing. Possible that information may be shared because of shared use of resources. E.g. information on a disk may remain if the disk is reallocated.
 - A virtual machine “escape”. One user can break the hypervisor. So far, purely academic.
 - Side channel attacks. One user can detect information through monitoring cache, for example. Again, so far, purely academic.
 - Denial of Service attacks. One user can consume resources and deny them to other users.
- Organizations need to consider risks when deciding what applications to host in the cloud.

Performance

- Auto-scaling provides additional performance when load grows.
 - Response time for new resources may not be adequate
 - Architects need to be aware of resource requirements for applications
 - Build that knowledge into the applications
 - May applications self aware so that they can be proactive with respect to resource needs.

Availability

- Failure is a common occurrence in the cloud
 - With 1000s of servers, failure is to be expected
- Cloud providers ensure that the cloud itself will remain available with some notable exceptions.
- Application developers must assume instances will fail and build in detection and correction mechanisms in case of failure.

Summary

- The cloud provides a new platform for applications with some different characteristics.
- Architect needs to know how a cloud cluster works and pay special attention to
 - Security
 - Performance
 - Availability



BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Module 8

Part 03

Best Practices

Harvinder S Jabbal
SE ZG651/ SS ZG653 Software Architectures



Best Practices

AWS Cloud

The Cloud Computing Difference:

- IT Assets become programmable resources
- Global, Availability, and Unlimited Capacity
- Higher Level Managed Services
- Security Built in.

AWS: Design Principles

- Scalability
- Disposable Resources Instead of Fixed Servers
- Automation
- Loose Coupling
- Services, not Server
- Database
- Removing Single Point of Failure
- Optimise for Cost
- Caching
- Security

Multi-tenant Applications for the Cloud: Micro Soft case Study

Windows Azure

- Goals and Requirements
 - The Tenant's Perspective
 - The Provider's Perspective
- Single Tenant vs. Multiple Tenant
- Multi-Tenancy Architecture in Windows Azure
- Selecting Single-Tenant or Multi-Tenant Architecture
 - Architectural Consideration
 - Application Life Cycle
 - Customizing the Application
 - Financial Consideration

Other Topics: Microsoft Azure

- Choosing a Multi-Tenant Data Architecture
- Partitioning Multi-Tenant Application
- Maximising Availability, Scalability and Elasticity
- Securing Multi-Tenant Applications
- Managing and Monitoring Multi-Tenant Applications

Microsoft Application Architecture Guide



Software Architecture and Design

- What is Software Architecture?
- Key Principles of Software Architecture
- Architectural Patterns and Styles
- A Technique for Architecture and Design

Microsoft Application Architecture Guide: Design Fundamentals



- Layered Application Guidelines
- Presentation Layer Guidelines
- Business Layer Guidelines
- Data Layer Guidelines
- Service Layer Guidelines
- Component Guidelines
- Designing Presentation Components
- Designing Business Components
- Designing Business Entities
- Designing Workflow Components
- Designing Data Components
- Quality Attributes
- Crosscutting Concerns
- Communication and Messaging
- Physical Tiers and Deployment

Microsoft Application Architecture Guide: Application Archetypes



- Choosing an Application Type
- Designing Web Applications
- Designing Rich Client Applications
- Designing Rich Internet Applications
- Designing Mobile Applications
- Designing Service Applications
- Designing Hosted and Cloud Services
- Designing Office Business Applications
- Designing SharePoint LOB Applications

Thank you



BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Module 8 Part 4 Review/The Road Ahead

Harvinder S Jabbal
SE ZG651/ SS ZG653 Software Architectures



The Road Ahead

Items for Preview

-
- SOA
 - Cloud
 - CAP Theorem



SOA

Service Oriented Architecture Pattern

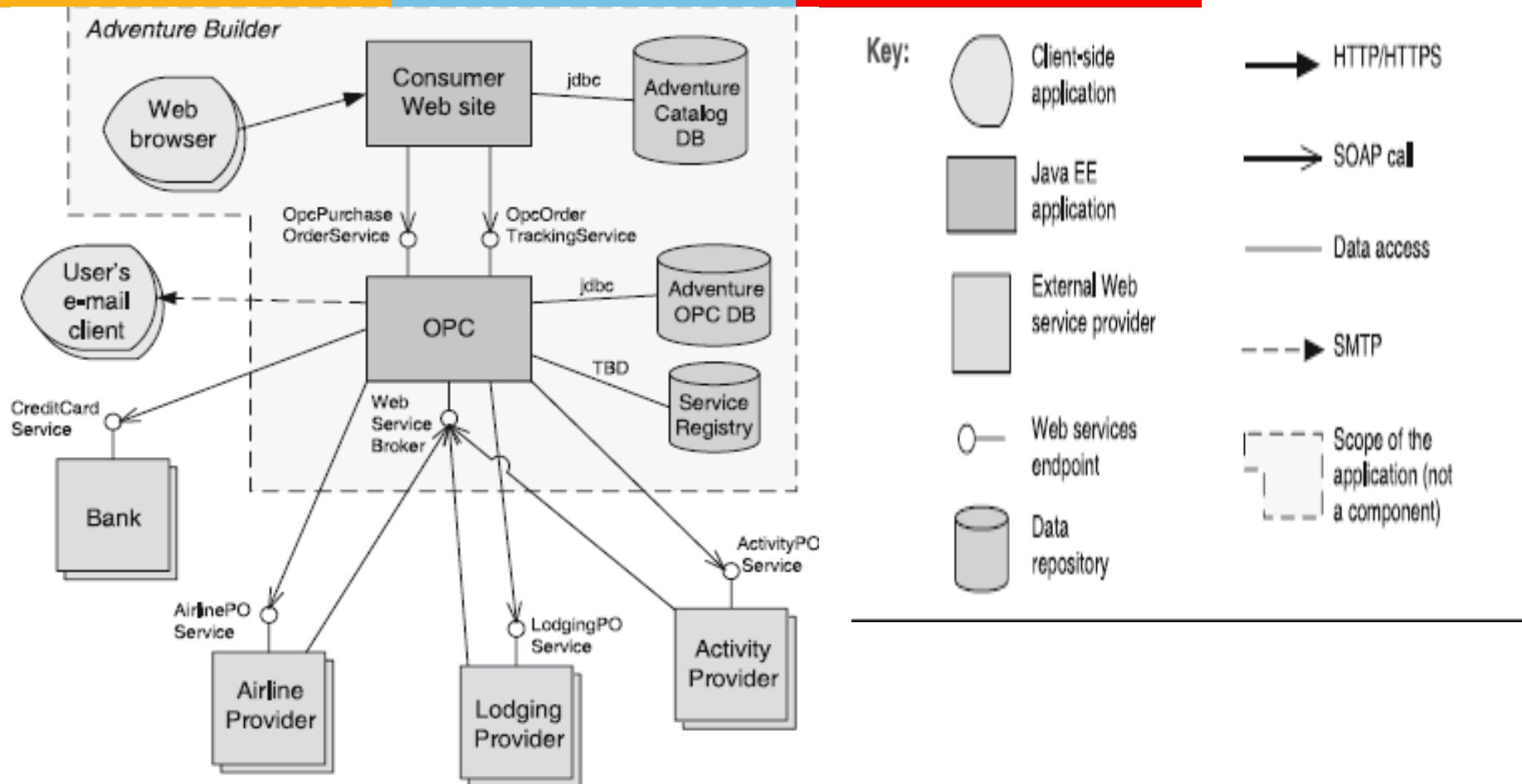


- **Context:** A number of services are offered (and described) by service providers and consumed by service consumers. Service consumers need to be able to understand and use these services without any detailed knowledge of their implementation.

Problem: How can we support interoperability of distributed components running on different platforms and written in different implementation languages, provided by different organizations, and distributed across the Internet?

Solution: The service-oriented architecture (SOA) pattern describes a collection of distributed components that provide and/or consume services.

Service Oriented Architecture Example



Service Oriented Architecture

Solution - 1



Overview: Computation is achieved by a set of cooperating components that provide and/or consume services over a network.

Elements:

- Components:
 - *Service providers*, which provide one or more services through published interfaces.
 - *Service consumers*, which invoke services directly or through an intermediary.
 - *Service providers* may also be service consumers.
- *ESB*, which is an intermediary element that can route and transform messages between service providers and consumers.
- *Registry of services*, which may be used by providers to register their services and by consumers to discover services at runtime.
- *Orchestration server*, which coordinates the interactions between service consumers and providers based on languages for business processes and workflows.

Service Oriented Architecture

Solution - 2



– Connectors:

- *SOAP connector*, which uses the SOAP protocol for synchronous communication between web services, typically over HTTP.
- *REST connector*, which relies on the basic request/reply operations of the HTTP protocol.
- *Asynchronous messaging connector*, which uses a messaging system to offer point-to-point or publish-subscribe asynchronous message exchanges.

Service Oriented Architecture

Solution - 3



Relations: Attachment of the different kinds of components available to the respective connectors

Constraints: Service consumers are connected to service providers, but intermediary components (e.g., ESB, registry, orchestration server) may be used.

Weaknesses:

- SOA-based systems are typically complex to build.
- You don't control the evolution of independent services.
- There is a performance overhead associated with the middleware, and services may be performance bottlenecks, and typically do not provide performance guarantees.



Cloud Computing Deployments Should Begin With Service Definition

- Dennis Smith

Cloud Computing Strategies

- Infrastructure and operations leaders often struggle to understand the role of cloud computing and develop strategies that exploit its potential.
- Infrastructure & Operations leaders should complete the prerequisites before making the technology decisions required for successful, service-centered cloud computing strategies.

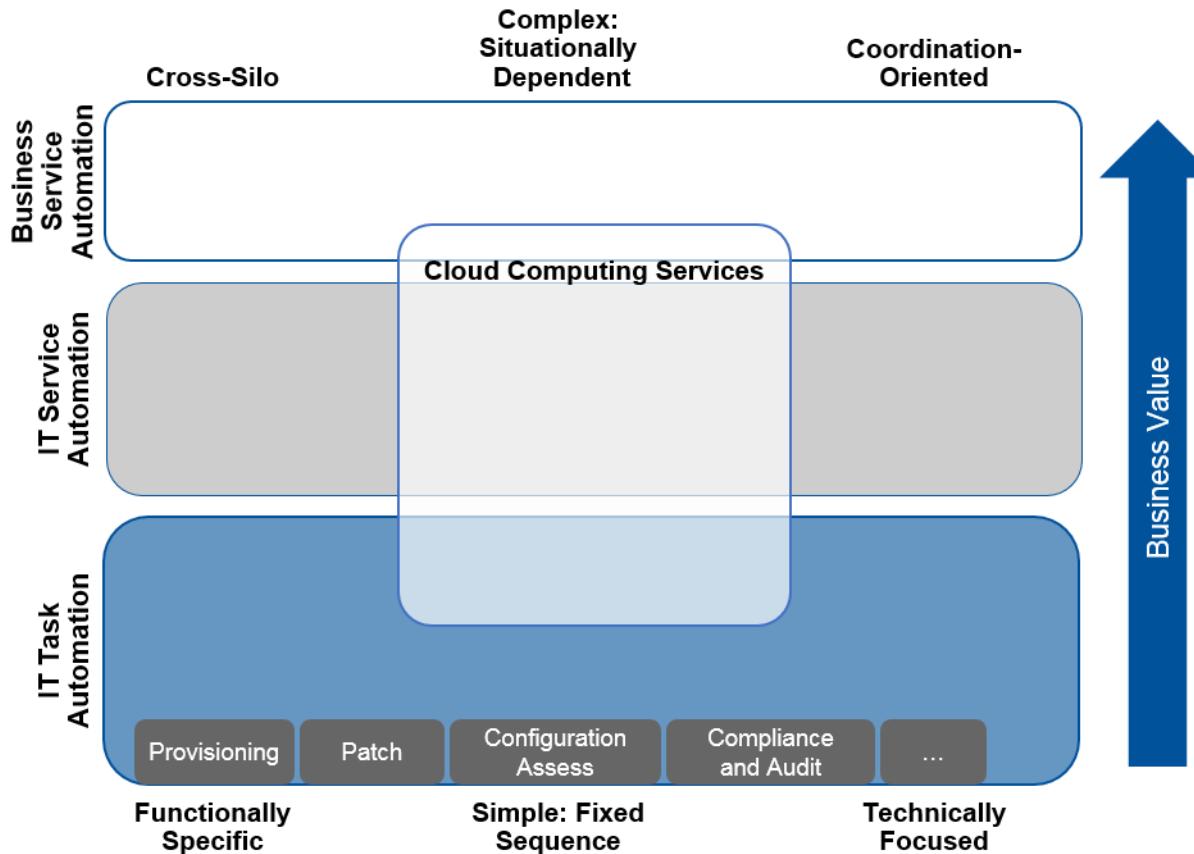
Key Challenges

- Many enterprises have failed to achieve success with cloud computing, because they failed to develop a cloud strategy rooted in the definition and delivery of IT services linked to business outcomes.
- Many companies are unsure how to initiate their cloud projects, which could cause them to miss chances to capitalize on business opportunities.

Recommendations

- Identify the cloud-computing-related IT services you will offer or procure.
- Document the internal processes that will be affected by the identified cloud services.
- Map applications and workloads to the associated cloud services.

Source: Gartner Report (July 2016)





CAP Theorem

CAP Theorem

- Described the *trade-offs involved in distributed system*
- It is impossible for a web service to provide following *three guarantees at the same time*:
 - **Consistency**
 - **Availability**
 - **Partition-tolerance**

CAP Theorem

Consistency:

- All nodes should see the same data at the same time

Availability:

- Node failures do not prevent survivors from continuing to operate

Partition-tolerance:

- The system continues to operate despite network partitions

- A distributed system can satisfy any two of these guarantees at the same time **but not all three**

Not Consistent

- Data A is being read a node in one partition.
- Data B is being written into a node another partition.
- This assures:

Availability:

- Node failures do not prevent survivors from continuing to operate

Partition-tolerance:

- The system continues to operate despite network partitions

Not Available

- Data A in one partition is locked while....
- Data B is being written into a node another partition.
- This assures:

Consistency:

- All nodes should see the same data at the same time

Partition-tolerance:

- The system continues to operate despite network partitions

Not Partition Tolerant

- Data A in one partition is being read based on update in a node another partition
- Data B is being written into the node in the other partition.
- This assures:

Availability:

- Node failures do not prevent survivors from continuing to operate

Consistency:

- All nodes should see the same data at the same time

Thank you



BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Module 9 Part 1

Management and Governance

Harvinder S Jabbal
SSZG653 Software Architectures



Management and Governance

Outline

Planning

Organizing

Implementing

Measuring

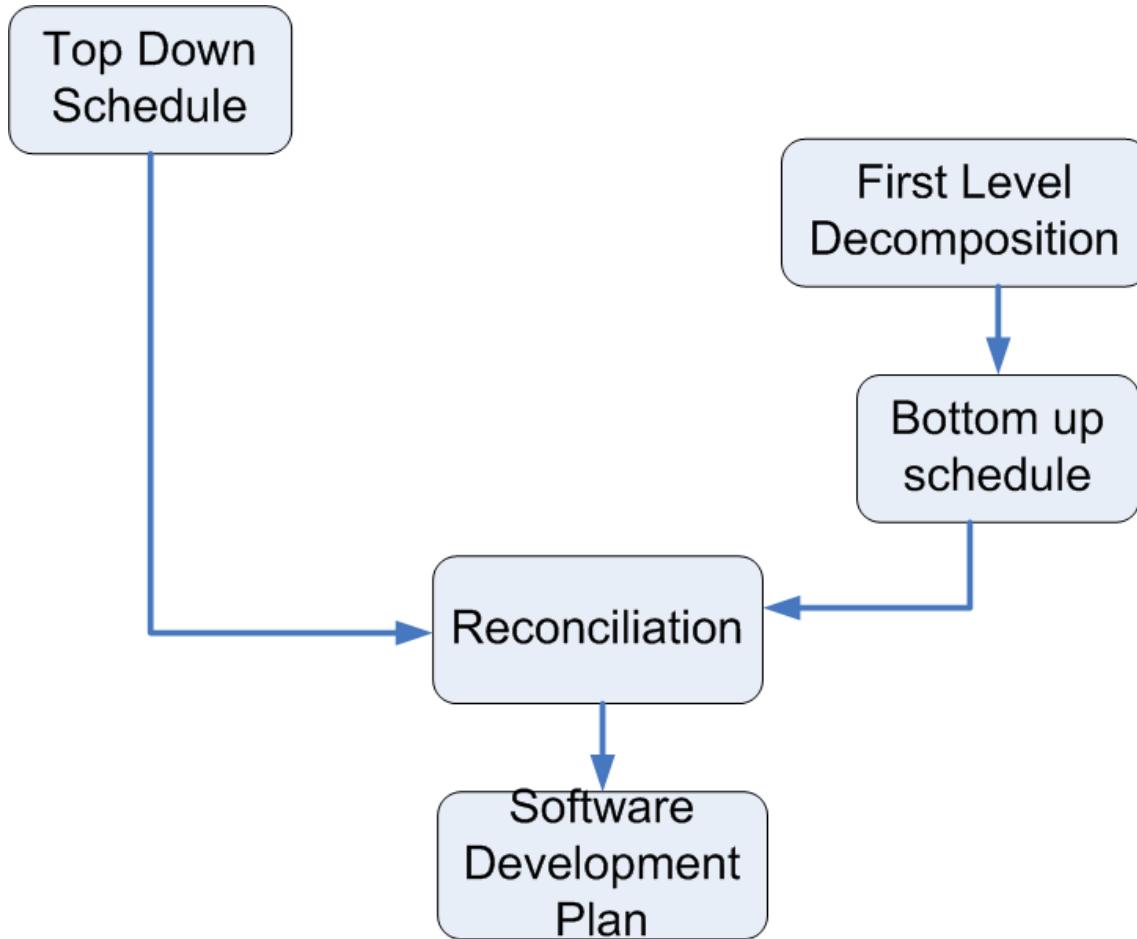
Governance

Summary

Planning

- The planning for a project proceeds over time.
- There is an initial plan that is necessarily top-down to convince upper management to build this system and give them some idea of the cost and schedule.
- This top-down schedule is inherently going to be incorrect, possibly by large amounts.
- Once the system has been given a go-ahead and a budget, the architecture team is formed and produces an initial architecture design.

The Planning Process



Top Down Schedule

- A top down schedule is needed to enable management to decide whether to do the project and to allocate resources.

Example: For a medium size project (~150K SLOC)

- Number of components to be estimated: ~150
- Paper design time per component: ~4 hours
- Time between engineering releases: ~8 weeks
- Overall project development allocation:
 - 40 percent design: 5 percent architectural, 35 percent detailed
 - 20 percent coding
 - 40 percent testing

Remaining Planning Steps

- An architecture team is created and they develop the first level decomposition of the architecture.
- Each member of the architecture team will be the lead architect for each major subsystem.
- A bottom up schedule is created by the architecture team
 - Typically more accurate than the top down schedule
 - The top down and the bottom up schedules must be reconciled to produce final (initial) schedule.
- Software development plan is written that specifies releases dates and features per release. This plan guides the initial activities of the project.

Organizing

-
- Division of responsibilities between project manager and software architect
 - Global Software Development

Project Manager and Software Architect



- This is the most important working relationship on the team.
- The people in each role—PM and SA—must
 - Respect each other
 - Coordinate
 - Stick to their respective spheres.

Project Management Body of Knowledge (PMBOK)



Published by the Project Management Institute

ANSI and IEEE standard

Nine project management areas

1. Integration Management
2. Scope Management
3. Time Management
4. Cost Management
5. Quality Management
6. Human Resource Management
7. Communications Management
8. Risk Management
9. Procurement Management

Integration Management

- Ensuring that the various elements of the project are properly coordinated.
- Developing, overseeing, and updating the project plan. Managing change control process.
 - PM: Organizes project, manages resources, budgets and schedules. Defines metrics and metric collection strategy. Oversees change control process.
 - SA: Creates design and organizes team around design. Manages dependencies. Implements the capture of the metrics. Orchestrates requests for changes. Ensures that appropriate IT infrastructure exists.

Scope Management

- Ensuring that the project includes all of the work required and only the work required.
- Requirements
 - PM: Negotiates project scope with marketing and software architect.
 - SA: Elicits, negotiates, and reviews run time requirements and generate development requirements. Estimates cost, schedule, and risk of meeting requirements.

Time Management

- Ensuring that the project completes in a timely fashion.
- Work breakdown structure and completion tracking. Project network diagram with dates.
 - PM: Oversees progress against schedule. Helps define work breakdown structure. Schedule coarse activities to meet deadlines.
 - SA: Helps define work breakdown structure. Defines tracking measures. Recommends assignment of resources to software development teams.

Cost Management

- Ensuring that the project is completed within the required budget.
- Resource planning, cost estimation, cost budgeting.
 - PM: Calculates cost to completion at various stages, makes decisions regarding build/buy and allocation of resources.
 - SA: Gathers costs from individual teams, makes recommendations regarding build/buy and resource allocations.

Quality Management

- Ensuring that the project will satisfy the needs for which it was undertaken.
- Quality & Metrics
 - PM: Defines productivity, size, and project-level quality measures.
 - SA: Designs for quality and tracks system against design. Defines code-level quality metrics.

Human Resource Management



- Ensuring that the project makes the most effective use of the people involved with the project.
- Managing people and their careers
 - PM: Maps skill sets of people against required skill sets. Ensures that appropriate training is provided. Monitors and mentors career paths of individuals. Authorizes recruitment.
 - SA: Defines required technical skill sets. Mentors developers about career paths. Recommends training. Interviews candidates.

Communications Management



- Ensuring timely and appropriate generation, collection, dissemination, storage, and disposition of project information.
- Communicating
 - PM: Manages communication between team and external entities. Reports to upper management.
 - SA: Ensures communication and coordination among developers. Solicits feedback as to progress, problems, and risks.

Risk Management

- Identify, analyze and respond to project risk.
- Risk Management
 - PM: Prioritizes risks, reports risks to management, takes steps to mitigate risks.
 - SA: Identifies and quantifies risks, adjusts architecture and processes to mitigate risk.

Procurement Management

- Acquire goods and services from outside organization.
- Technology
 - PM: Procures necessary resources. Introduces new technology.
 - SA: Determines technology requirements. Recommends technology, training, and tools.

Global Software Development



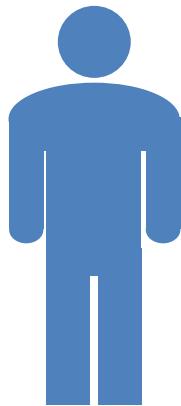
- A very common development context.
- Driven by
 - (Labor) costs
 - Skill sets and labor availability.
 - Local knowledge of markets.
- Global development means that coordination among teams is critical.

Team Coordination Induced by Module Interaction



If there is a dependency between two modules, the teams assigned to those modules must coordinate over the shared interfaces.

Team A



Coordination

Team B



Module A

Module B

Dependency

Coordination

- Local coordination can be informal and spontaneous.
- Remote coordination must be more structured.
- Coordination mechanisms:
 - Documentation
 - Meetings
 - Electronic media

Implementation Issues



- Trade-offs
- Incremental development
- Tracking progress

Trade-offs

- Software architect makes trade-offs among various quality attributes.
- Project manager makes trade-offs among
 - Features
 - Schedule
 - Quality
- Project manager should resist creeping functionality (scope creep)
 - Affects schedule
 - Can use a Change Control Board to manage (typically slow down) the pace of changes

Incremental Development



- A release may be in one of three states
 - Planning
 - Development
 - Test and repair
- All three states can be simultaneously active for different releases.

Tracking Progress

Progress can be tracked through

- Personal contact (doesn't scale)
- Meetings
- Metrics
- Risk management

Meetings

- Expensive use of time
- Either status or working – do not intermix
- One output of status meetings should be risks

Risks have

- Cost if they occur
- Likelihood of their occurrence

Project manager prioritizes risks

Measuring

- Metrics are an important tool for project managers. They enable the manager to have an objective basis both for their own decision making and for reporting to upper management on the progress of the project.
- Metrics can be global—pertaining to the whole project—or they may depend on a particular phase of the project.

Global Metrics

- Global metrics aid the project manager in obtaining an overall sense of the project and tracking its progress over time.
- Some example metrics, that any project should capture:
 - Size
 - Schedule deviation
 - Developer productivity
 - Defects
- Metrics should be tracked both historically for the organization and for the specific project.

Phase Metrics and Cost to Complete



Phase metrics

- Open issues
- Unmitigated risks

Cost to complete

- Bottom up metric
 - Responsibility of lead architect for each subsystem team

Governance

Four responsibilities of a governing board

1. Implementing a system of controls over the creation and monitoring of all architectural components and activities, to ensure the effective introduction, implementation, and evolution of architectures within the organization.
2. Implementing a system to ensure compliance with internal and external standards and regulatory obligations.
3. Establishing processes that support effective management of the above processes within agreed parameters.
4. Developing practices that ensure accountability to a clearly identified stakeholder community, both inside and outside the organization.

Summary

A project must be planned, organized, implemented, tracked, and governed.

- Top down schedule based on size
- Bottom up schedule based on top level decomposition.
- Reconciliation of two schedules is the basis for the software development plan.

Teams are created based on the software development plan.

The software architect and the project manager must coordinate to oversee the implementation.

Global development creates a need for an explicit coordination strategy.

Management trade offs are between schedule, function, and cost.

Progress must be tracked.

Larger systems require formal governance mechanisms.



Module 9 Part 2

Mobile application architecture

BITS Pilani

Harvinder S Jabbal
SSZG653 Software Architectures



Architecting Mobile applications

Can you give some examples of mobile apps?

Mobile applications

Examples

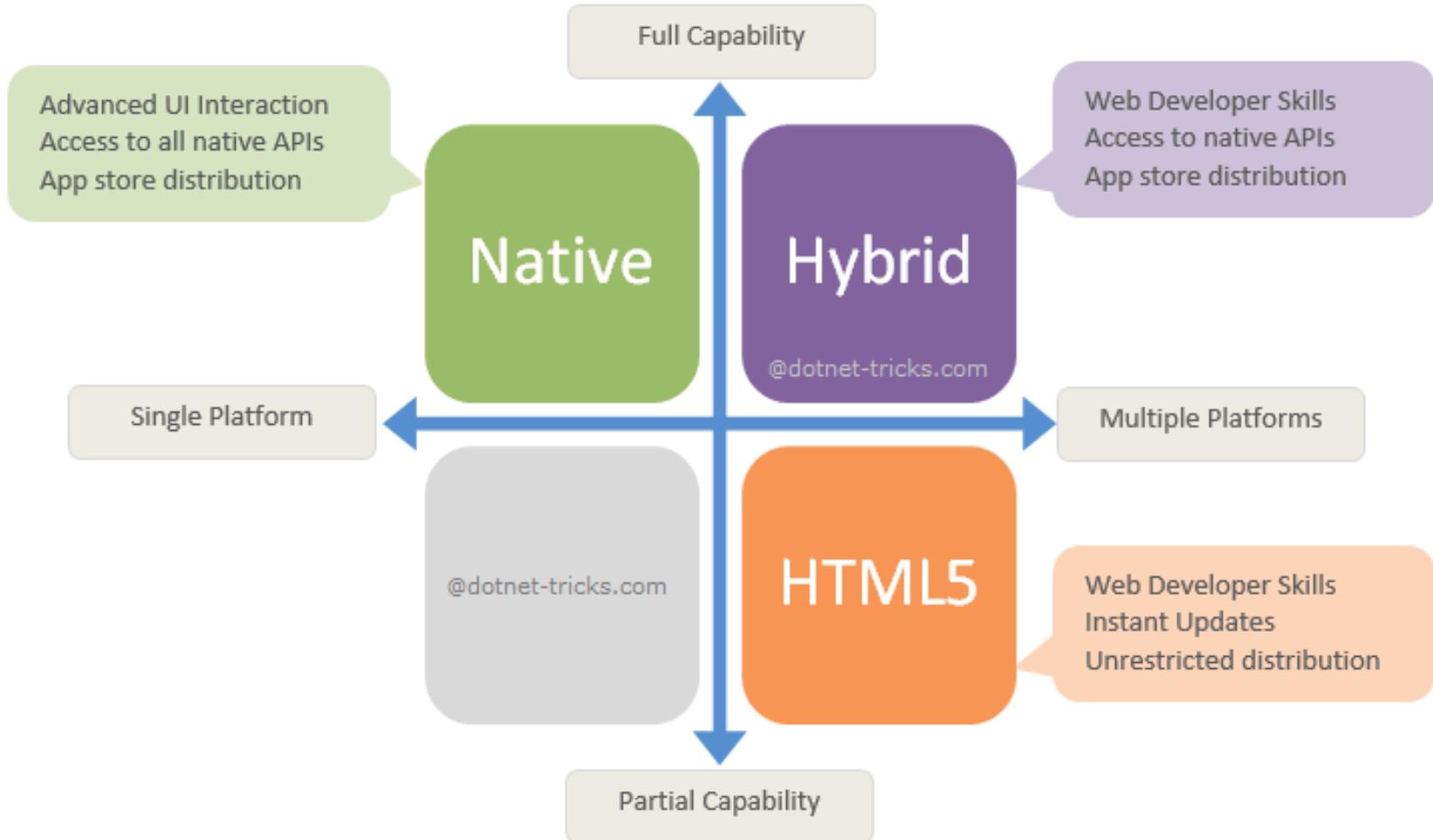
- Uber
- Swiggy
- Courier delivery
- eCom
- Banking
- Spotify
- Where Is My Train
- ...

Mobile Application: Design considerations



- Simple User interface: Easy to type, Large buttons, Minimal features, menu options, actions
- Responsive design: Adapt to different screen sizes & orientations
- Compact code: Less usage of CPU, memory, storage
- Few layers to ensure performance
- Connectivity: Store data locally and synchronize later if connection is poor

Types of Mobile Apps



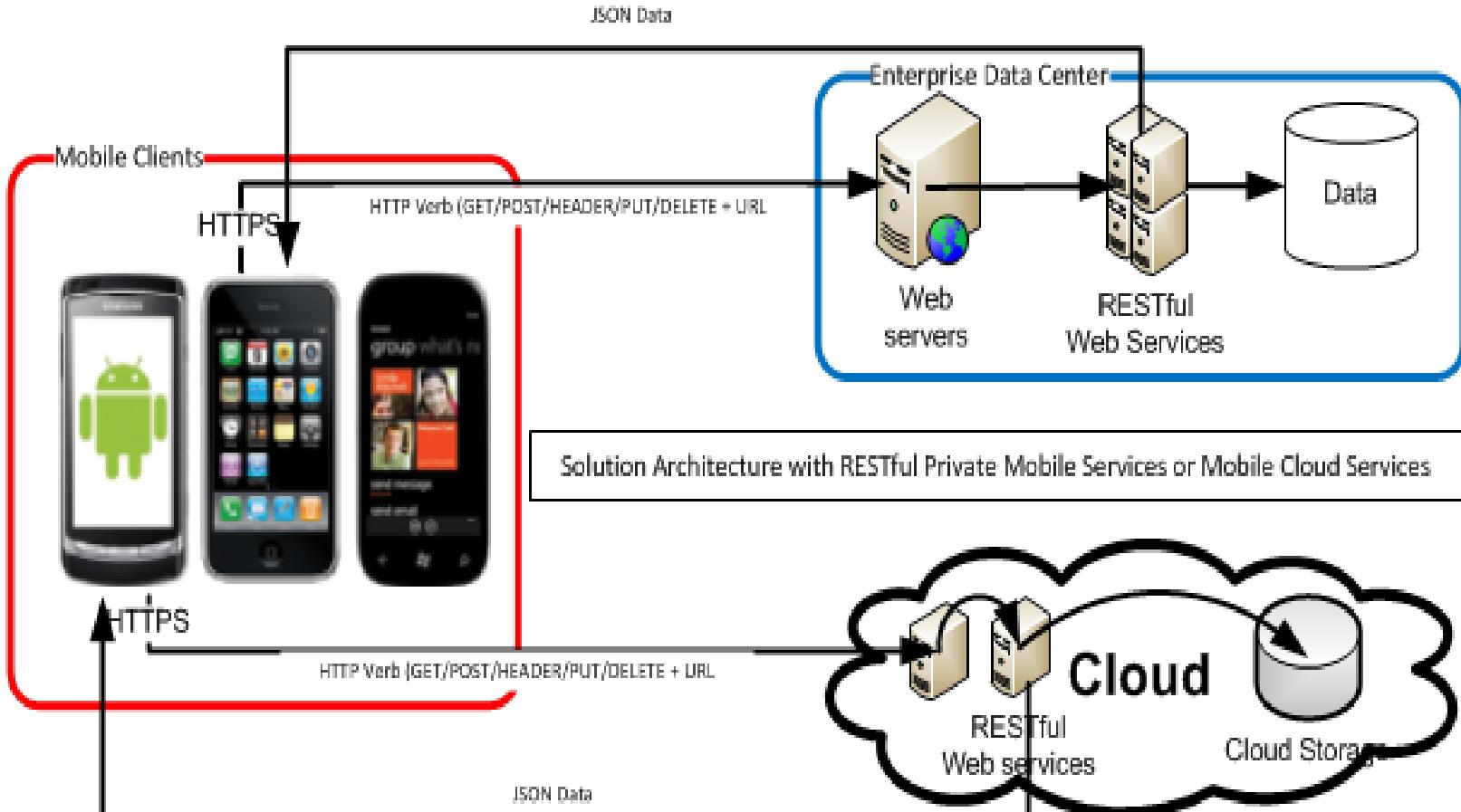
Mobile Native app

- They are built for specific platforms
 - Examples: Google Maps, Facebook, LinkedIn – one version for iOS and one for Android
- Languages used:
 - Native iOS : Swift or Objective-C
 - Native Android: Java or Kotlin
- Integrated Development Environment (IDE) such as Android studio are used for this

Mobile Web apps

- Progressive web apps use modern web technology to deliver app-like experiences to users, right in their browsers.
- Examples:
 - Flipkart
 - BookMyShow
 - MakeMyTrip
- Uses HTML5, CSS3, JavaScript and runs on a browser

Mobile web application



Hybrid app

- A hybrid app combines elements of both native apps and web applications.
 - Examples: Twitter, Uber, Instagram
 - Hybrid apps are essentially web apps (HTML, CSS, Javascript) that have been put in a native app shell.
 - The shell is able to connect to native capabilities of the mobile platform such as camera, accelerometer, GPS, etc.
 - Tools such as Xamarin and React Native allows app to run across platforms
-

Pros & Cons

App type	Pro	Con
Native	<ul style="list-style-type: none"> • High performance • Superior user experience • Access to all features of OS 	<ul style="list-style-type: none"> • Runs only on one platform • Need to know special language • Need to update versions
Web App	<ul style="list-style-type: none"> • Easy to deploy new versions • Common code base 	<ul style="list-style-type: none"> • Little scope to use device hardware • Lower user experience • Need to search for app
Hybrid	<ul style="list-style-type: none"> • Does not need browser • Single code base • Access to device hardware 	<ul style="list-style-type: none"> • Slower

UI design patterns

- Action bars for quick access to frequently used actions
- Login using Facebook, Google, etc. instead of separate user id / password
- Large buttons for ease of use
- Notifications of recent activity
- Discoverable controls: Controls show up only when an item is selected (ex. In WhatsApp, the Forward button shows up when a message is selected)

Mobile optimized web site



All features



DH E paper | Prajavani | PV E Paper | Sudha | Mayura | The Printers Mysore | DH Classifieds

Home | News | Karnataka | Bengaluru | Business | Budget 2018 | Supplements | Sports | Ente

Three soldiers killed as avalanche hits army post in Kashmir | Bad luck follows the 10 rupee coin | Swaraj, Ne



U-19 World Cup final: India limit Australia to 216 after brilliant bowling show



National health scheme to cost govt Rs 12K cr a year

For 2018-19, the government has made an initial provision of Rs 2,000 crore



Reduced features



GET APP



TOP STORIES NATION CITY BUDGET 2018



National health scheme to cost govt Rs 12K cr a year



The Union government is likely to pay an annual premium of less than Rs 1,200 per family for the ambitious national health protection scheme, for which approximately Rs 12,000 crore...



Bandh called off after High Court terms it illegal

- Muslims opposing Ram temple must go to Pak: UP Shia Waqf board chief
- Mehbooba says she can accept going to hell to save Kashmir

Responsive design

innovate

achieve

lead

Example: Boston Globe News

BOSTON.COM | CARS | JOBS | REAL ESTATE

50° 50° WEATHER | TRAFFIC

Red Sox Live 6 4 1st

News | Celtics

Boston's deadliest hour



THE BOSTON GLOBE

Extending club licenses until 3 or 4 a.m. won't eliminate violence, writes Lawrence Harten. But it would thin out the crowds and give police a better opportunity to arrest more troublemakers. 9:00 pm

Boston police seek public's help to solve soldier's murder

Rondo drives Celtics to win

Rajon Rondo had a triple-double Friday night to lead the Celtics to an overtime victory over the Atlanta Hawks in the third game of their first-round playoff series.

Orioles outlast Red Sox

Chris Davis drove in the go-ahead run with a single in the top of the

House targets health spending

Massachusetts House leaders called for new limits on the fees charged by hospitals and doctors and for creation of an agency to monitor medical spending.

Vt. on verge of historic 'fracking' ban

Vermont's governor is expected to sign a proposal approved by the state House on Friday that would ban the controversial natural gas drilling technique.

Jobs report shows US hiring slowed in April

Employers added 126,000 jobs, fewer than forecast and the slowest gain in six months, adding to concern the economic expansion is cooling.

Campaign2012

Tagg Romney announces birth of twin sons

It was reportedly the second time that Mitt Romney's eldest son and his wife had used a surrogate, which could spark serious social and religious debates.

Adam Yauch of the Beastie Boys dead at 47

Yauch's representatives said the rapper died Friday morning in New York after a

5/11/2012 | SATURDAY, MAY 5, 2012

NEXT ISSUE | GET ACCESS

MY SAVINGS

100 YEARS

ORIOLES vs. RED SOX

5/4-5/6

KETS STILL AVAILABLE

Boston Globe ePaper →

What is an ePaper?

The complete print edition, in its exact layout. Browse the print edition page by page, including stories and ads.

- Sign up now for full, free access to ePaper
- Download the app now

Globe Insiders →

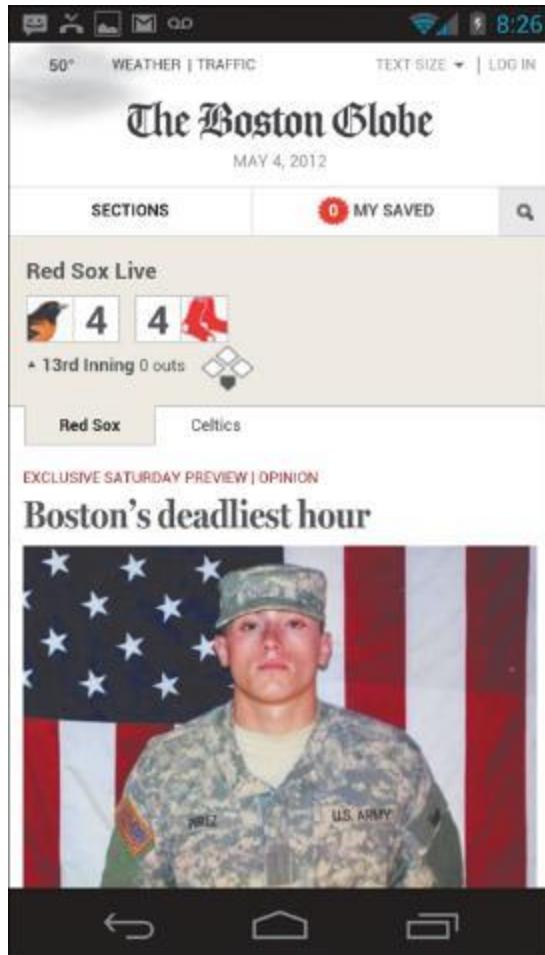
A conversation with Ben Mezrich

The New York Times best-selling author discusses his novels "Elating Down the House" and "The Social Network."

This is how the display looks on desktop

Responsive design

Example: Boston Globe News



This is how
the display
looks on
mobile

Adjusted
vertically

Responsive design

Single website for laptop & mobile & tablet

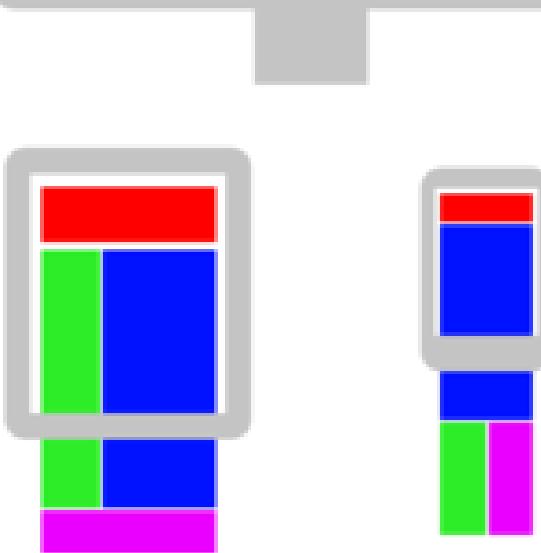
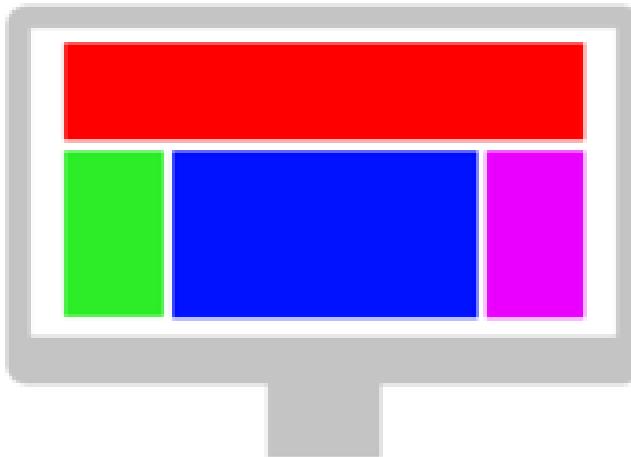
Principle: Adapt rendering depending on screen sizes & orientation



Ref: Wikipedia

Responsive design

Ref: Wikipedia



Flexi grids

- Divide a screen into multiple columns
- Assign HTML elements to one or more columns
- Choose a different layout depending on screen size

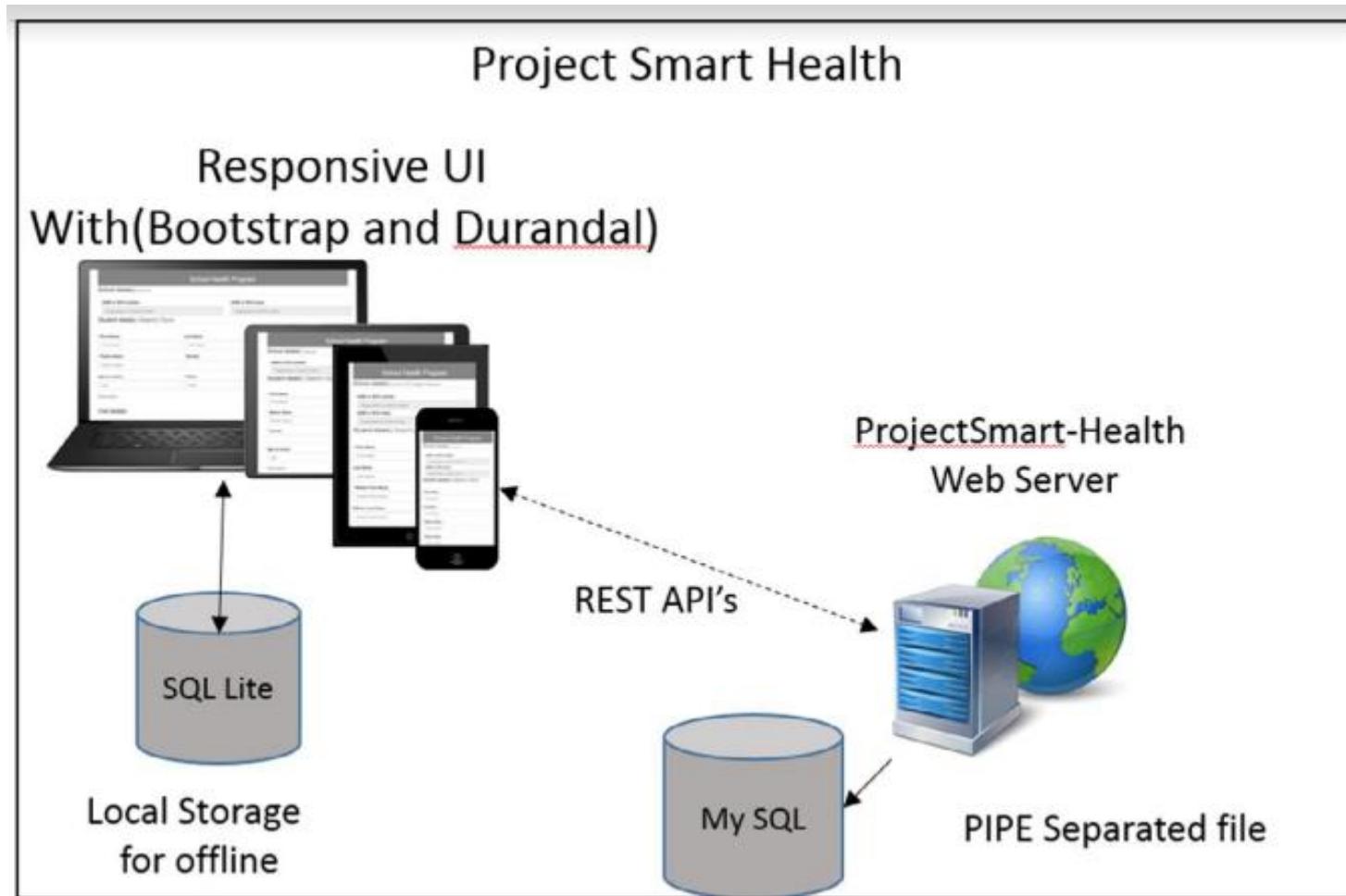
Responsive design

Technique

- Use CSS and HTML to shrink, hide or move content
- Flexible grids (CSS 3)
 - Use media queries to determine screen size
 - Specify grid width as % of screen size rather than fixed pixels
- Flexible images – Specify image size as % of grid size

Store locally, Sync later

In case of intermittent connectivity



Doctors enter patient data in mobile, which gets synced with server later

Mobile Application Architecture

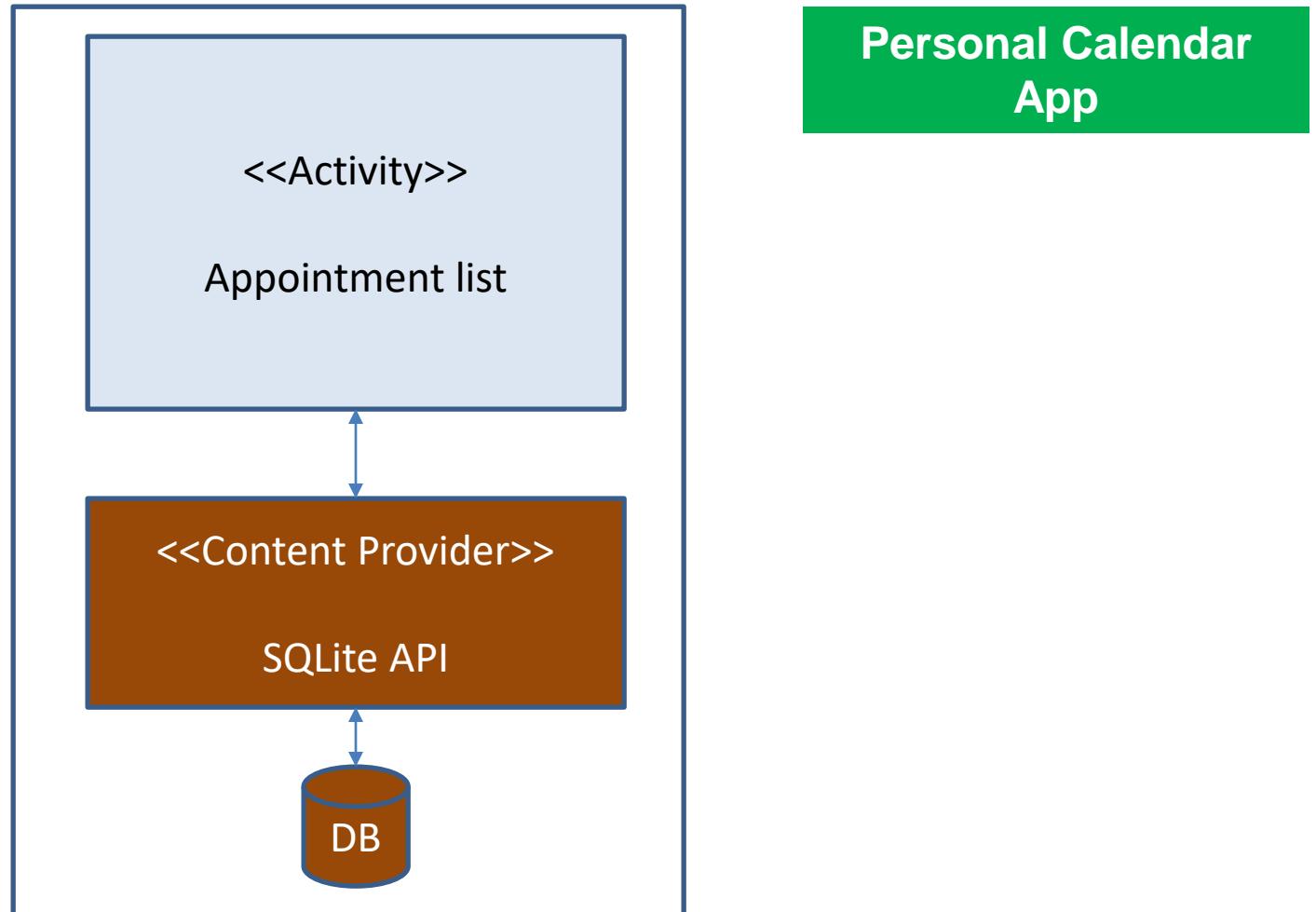
Android application architecture

4 types of components

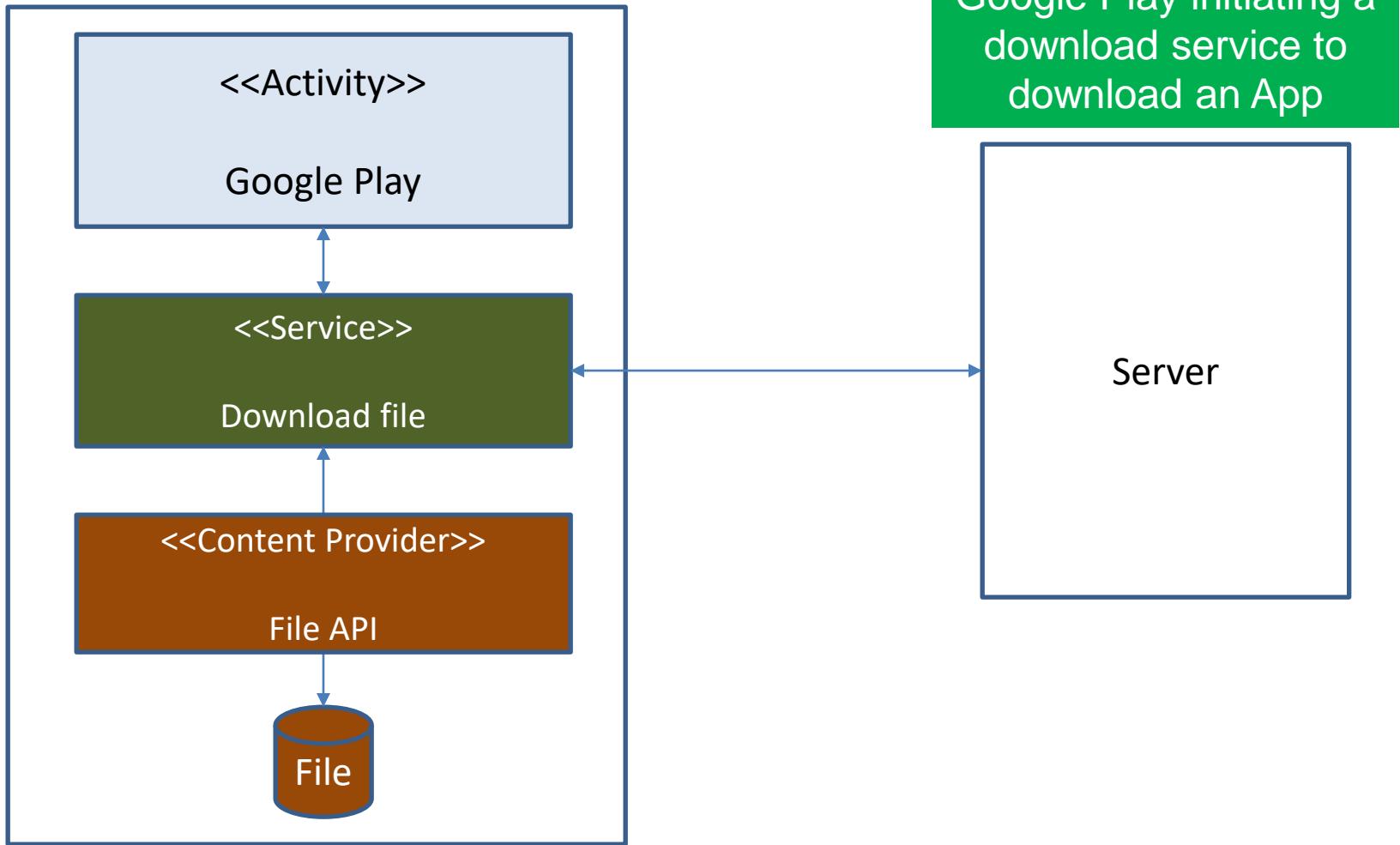
- Activity (UI)
- Service (background process) - ex. playing music, download
- Content provider (Storage) - ex. SQLite, files,
- Broadcast receiver (Acts on events received from OS and other apps) - Ex arrival of SMS

Examples of components

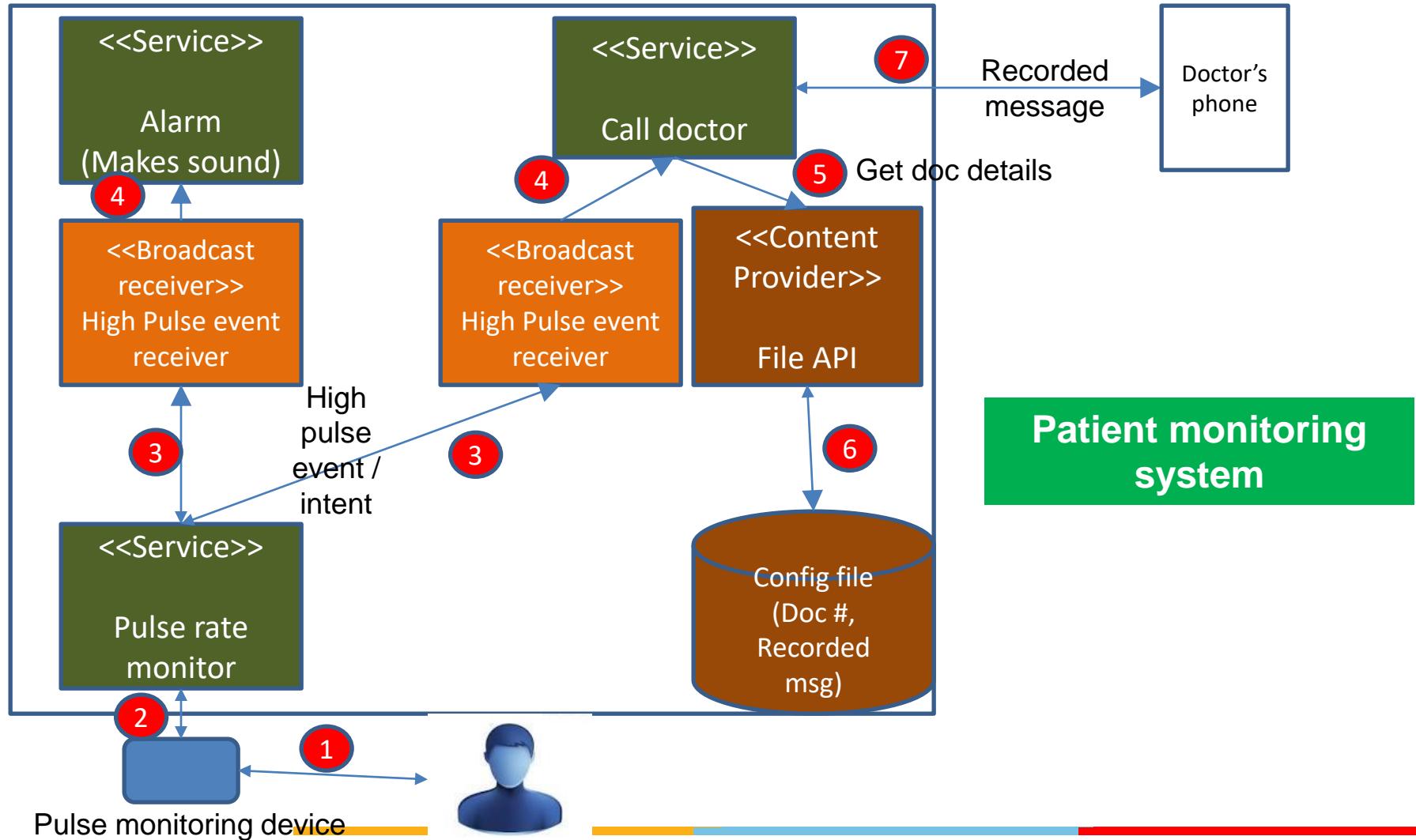
Activity & Content providers



(Background) Service

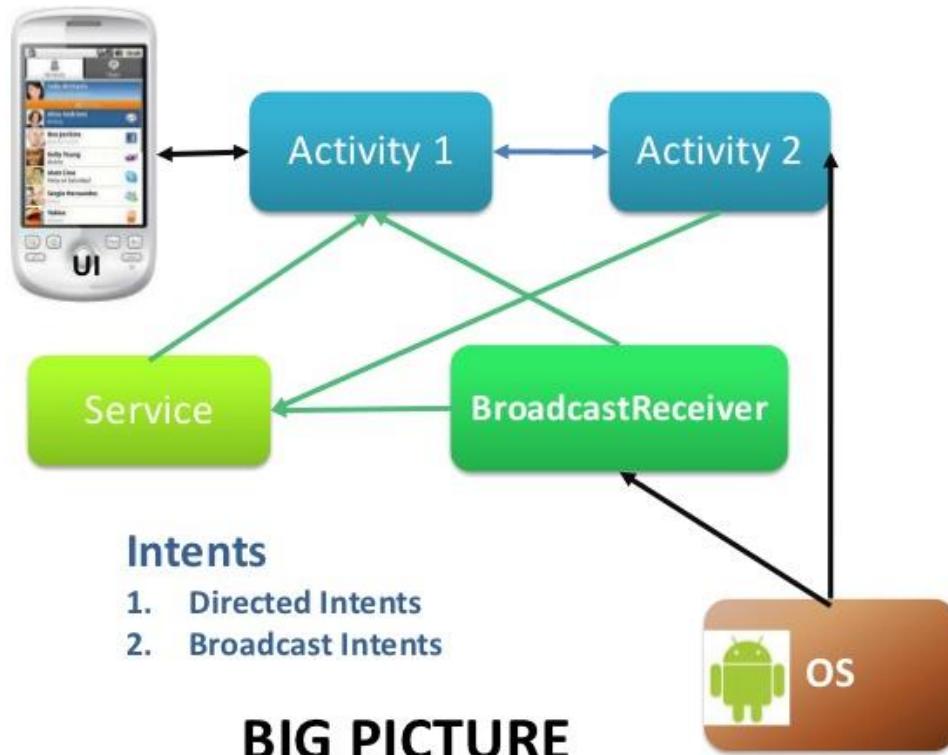


Broadcast receiver (Event handlers)



Android application structure

Android Application Anatomy



Exercise: Mobile app components



Give examples of mobile app components in **Uber app**.

UI

Screen to book a cab

Broadcast receiver

Receive location of cab from backend server and provide to UI for display

Service

Provide cab location to backend server after journey starts

Database

Configuration file containing user data

Exercise

Consider a mobile app carried by the courier delivery boy.

The app should support the following functions:

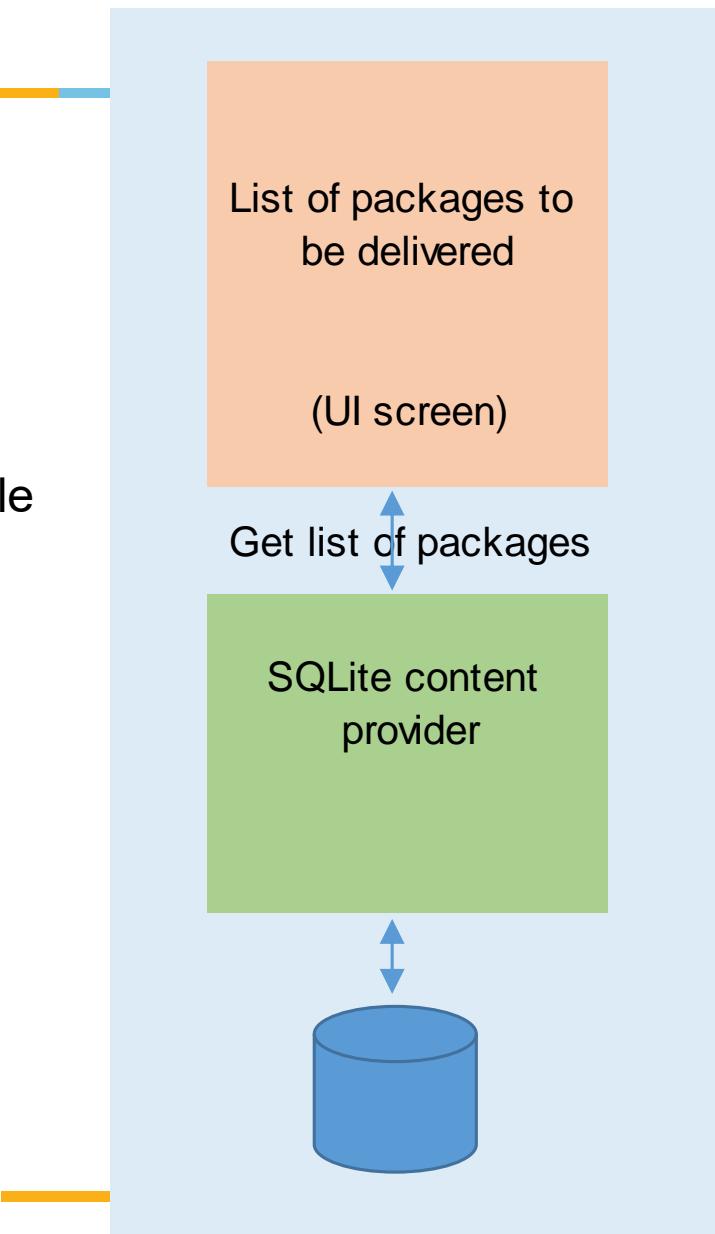
- a) View courier packages to be delivered
- b) Mark a package as delivered.
- c) Upon this event, the app should send information to central server

Identify the components of a mobile app & its inter-connections and draw an appropriate software architecture diagram

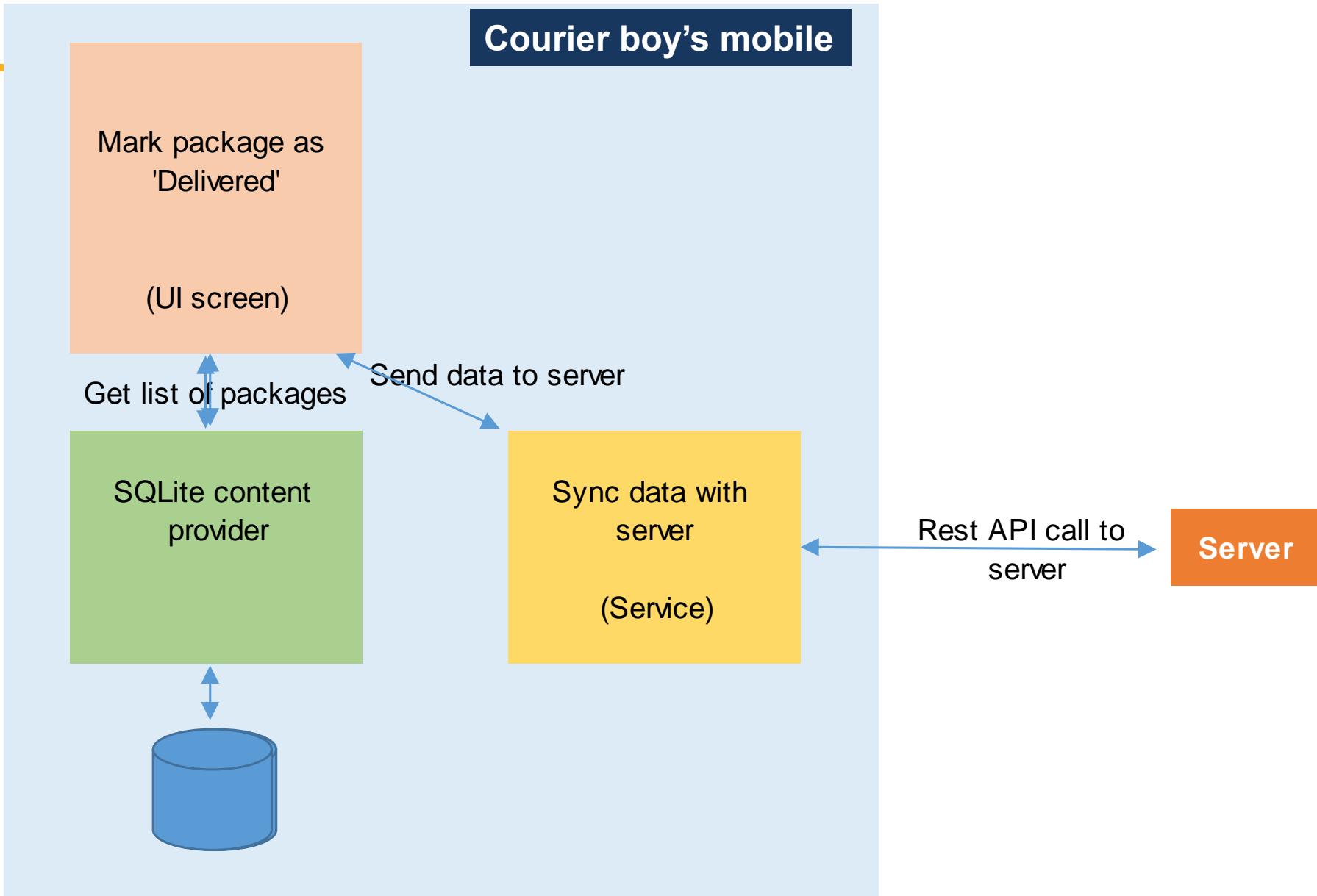
a) View courier packages to be delivered



Courier boy's mobile



b) Mark a package as delivered. Upon this event, the app sends information to central server



Review questions

Mobile apps

1. What is a cross platform mobile app?
2. If connectivity is poor, how do we ensure consistency between data in mobile phone and backend server?
3. What is 'Broadcast receiver' in an Android app?



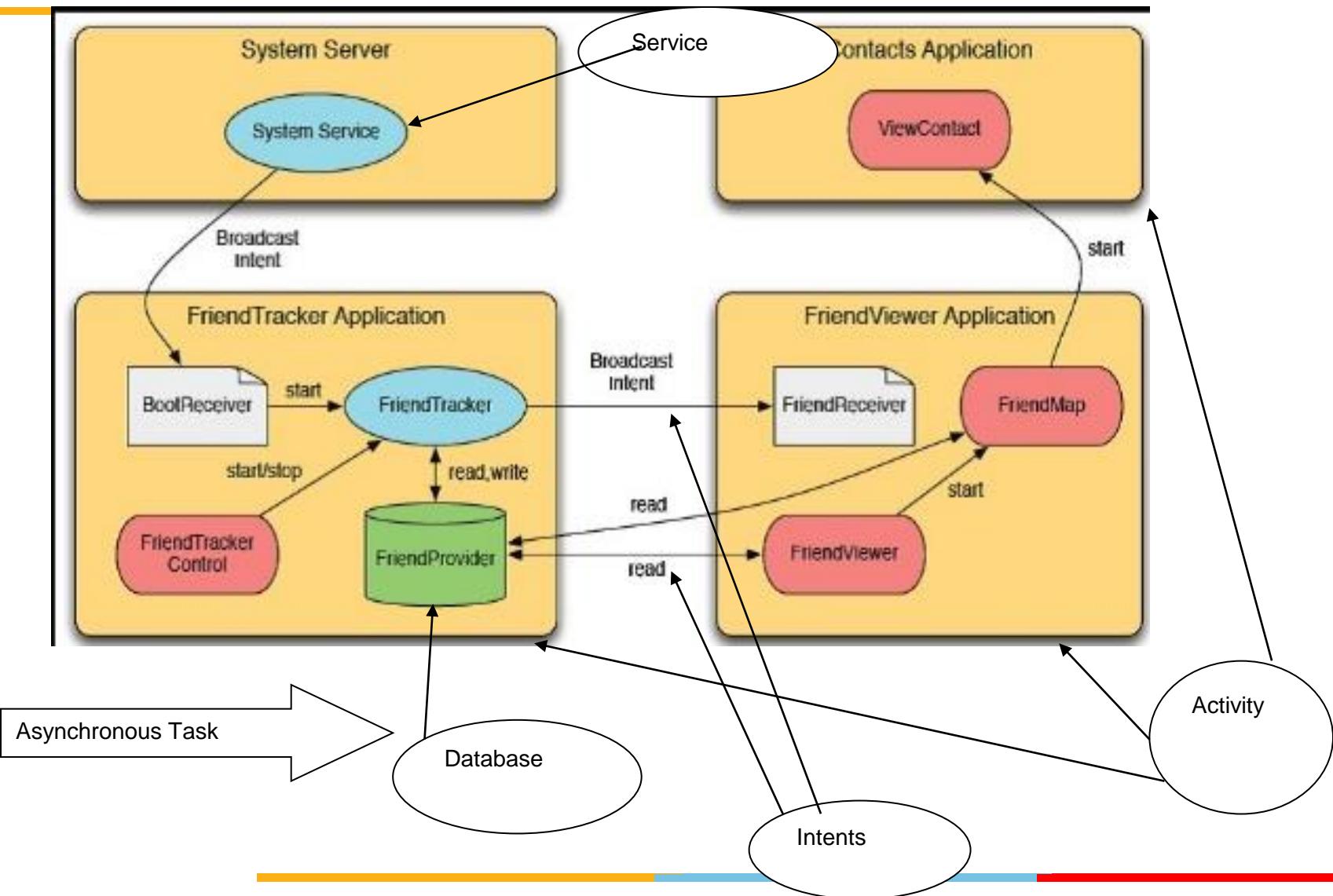
Appendix

Popular services of cloud vendors



Google App Engine	Microsoft Azure	Amazon Web Services
<ul style="list-style-type: none">• Python & Java development environment• Auto scaling• Database replication	<ul style="list-style-type: none">• .Net environment• Auto scaling• Load balancing• Failure detection & auto replication of services• Database replication	<ul style="list-style-type: none">• Java development• Auto scaling• Load balancing• Failure detection & auto replication of services• Database replication• Data caching (Memcached)• Service discovery (SoA)• Notification of events (SNS)• Message queue (SQS)• CDN (Content Delivery Network) – YouTube

Mobile app: Example



Amazon's approach to handling issues in distributed systems



- To scale you have to partition, so you are left with choosing either high consistency or high availability for a particular system. You must find the right overlap of availability and consistency.
- Choose a specific approach based on the needs of the service.
- For the checkout process you always want to honor requests to add items to a shopping cart because it's revenue producing. In this case you choose high availability. Errors are hidden from the customer and sorted out later.
- When a customer submits an order you favor consistency because several services--credit card processing, shipping and handling, reporting--are simultaneously accessing the data.

Issues in Cloud based systems



- Availability
 - Cloud vendors promise high availability ex. 99.95%
 - But still not 100%
 - Need to design for the 0.05%
 - One approach: Store same data in different geographical zones as done by Netflix



Exercise

Scenario

A start-up company is developing a GST tax returns filing system. This software will be deployed on the Cloud and offered to small and medium businesses as a SaaS.

The clients will have to input their data or upload data using an Excel file. They also need to provide other details such as GST #, etc. The software will process the data and file the returns into the Government's GST system on behalf of the client.

After developing the software, what options exist for deploying it in the cloud?

Different ways to deploy applications on the cloud



Scenario

A start-up company – Tailspin - is developing a **customer survey & analysis** application. This software will be deployed on the Cloud and offered to clients as a SaaS.

- The clients (subscriber of the application) can create and launch a survey.
- After this the survey participants will access the application and answer the survey questions.
- After the data has been gathered the application will perform analysis and present the results to the client organization

What options exist for Tailspin to deploy the application on the cloud?

Context diagram

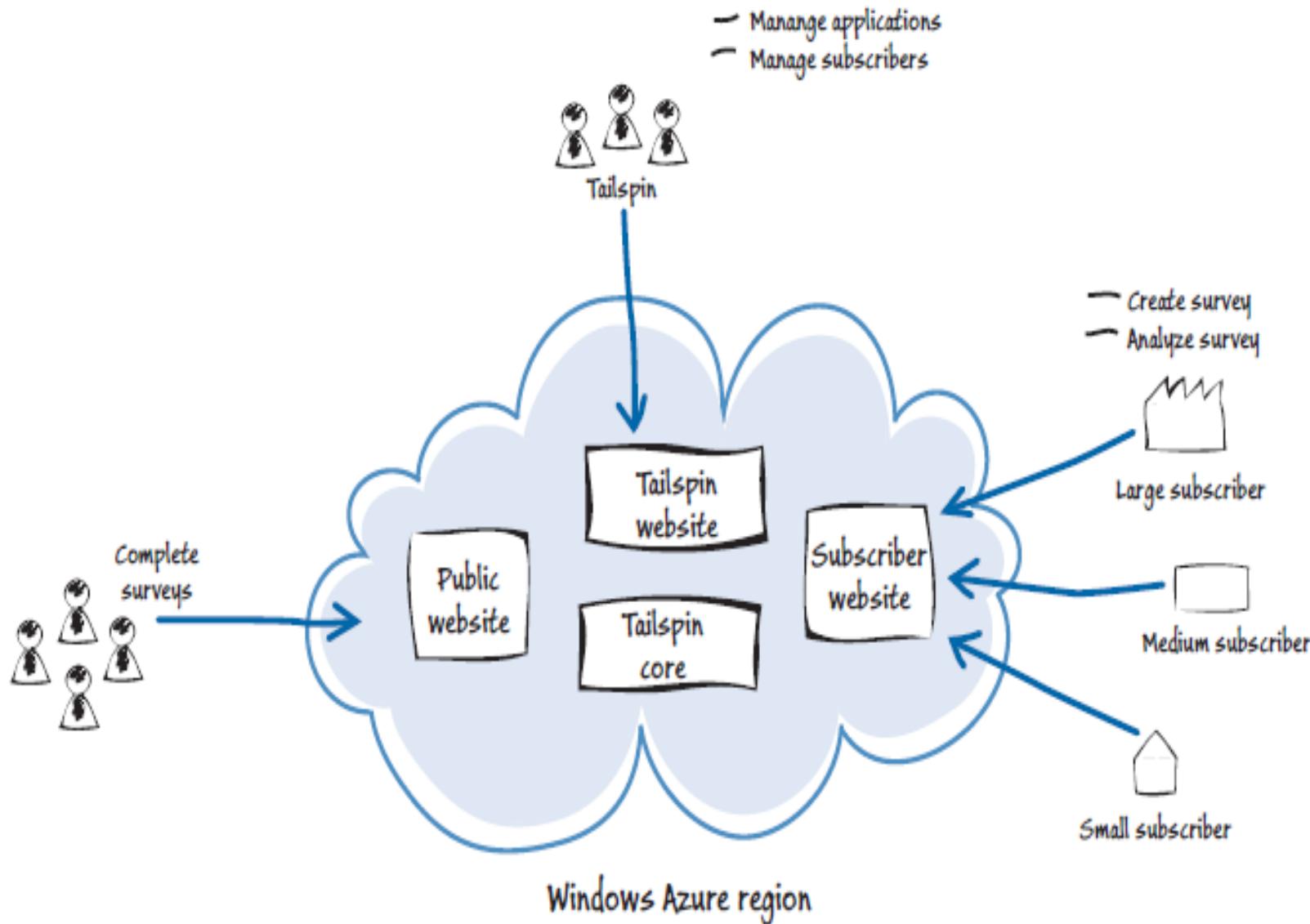


FIGURE 1
The Surveys application

Multi-tenant application Architecture – High level

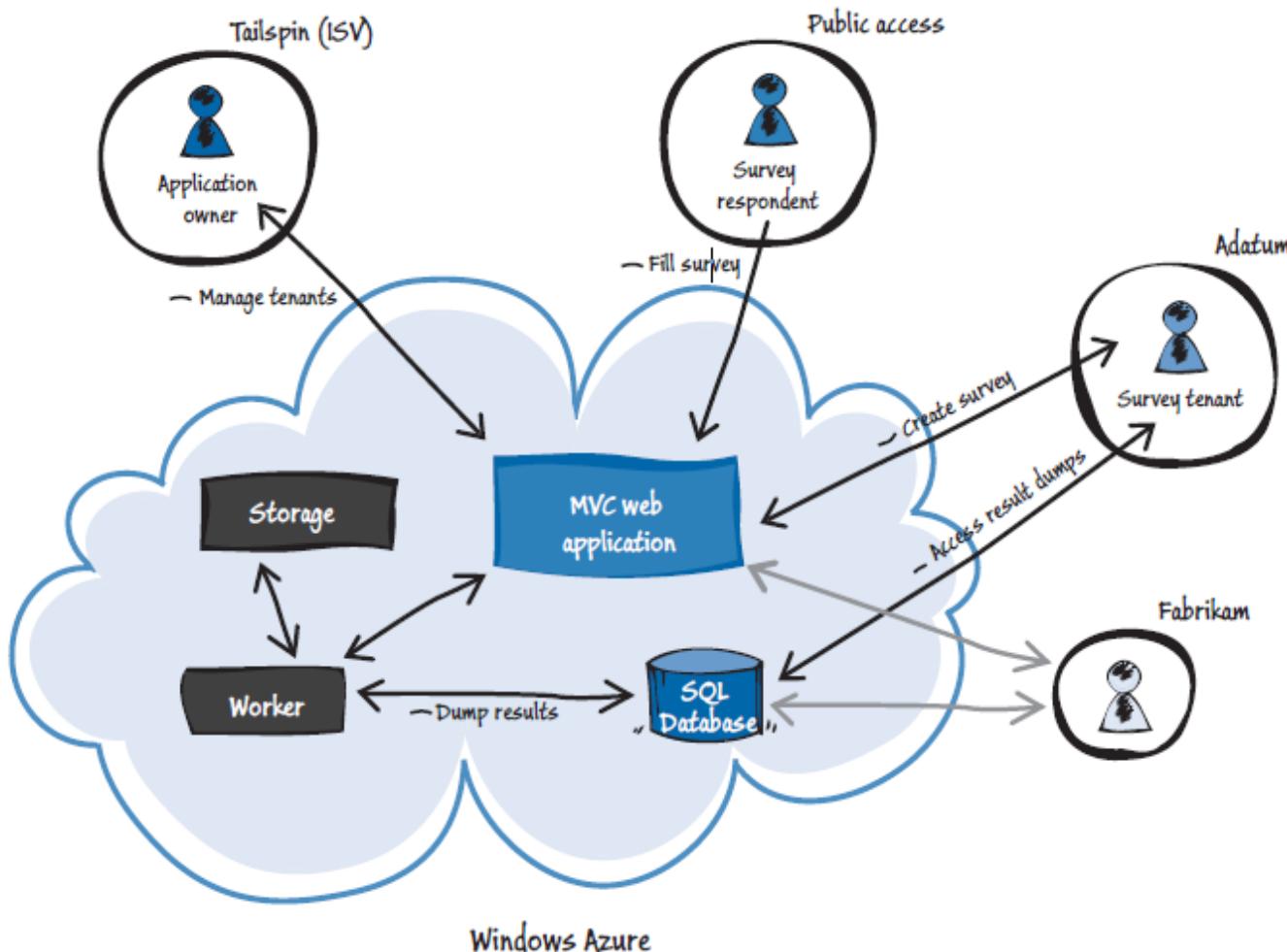
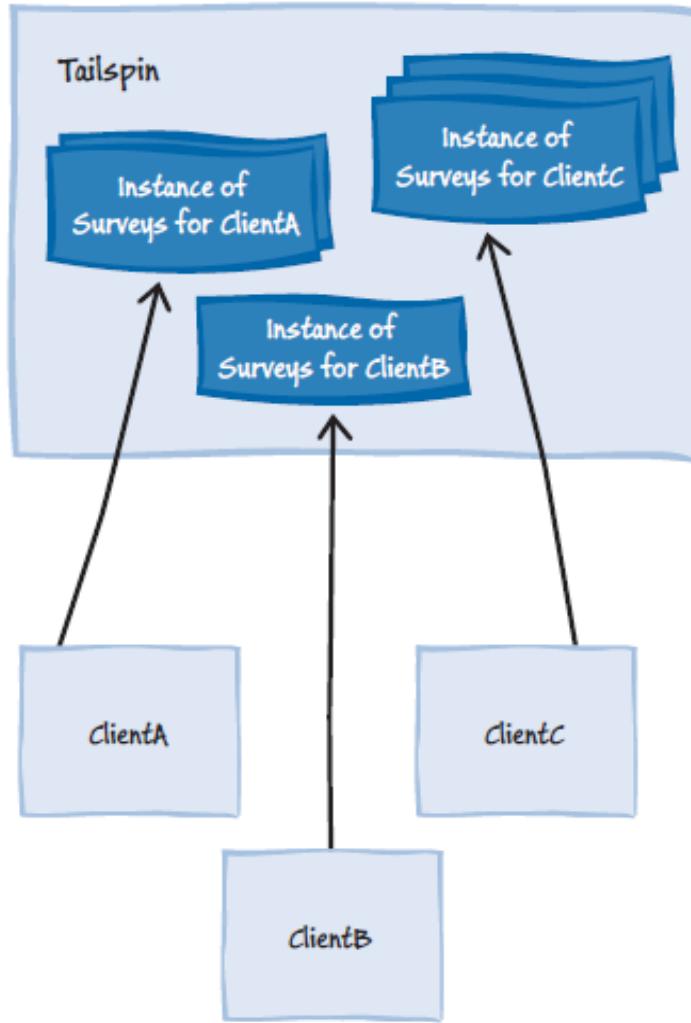


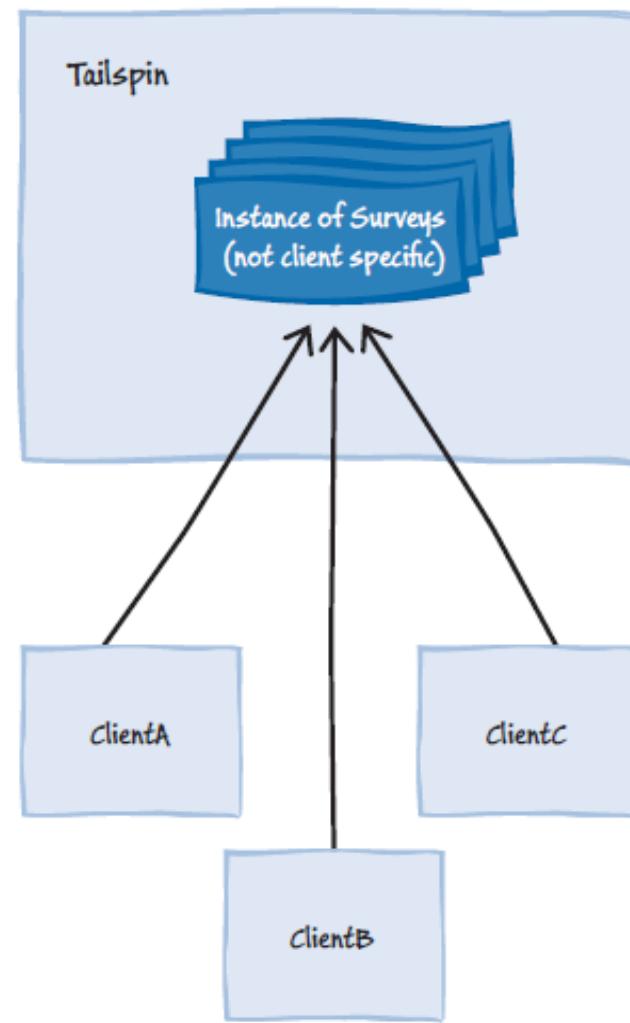
FIGURE 2
The Surveys application architecture

Architecture Options – High level

Multi-instance, single tenant



Single instance, multi-tenant

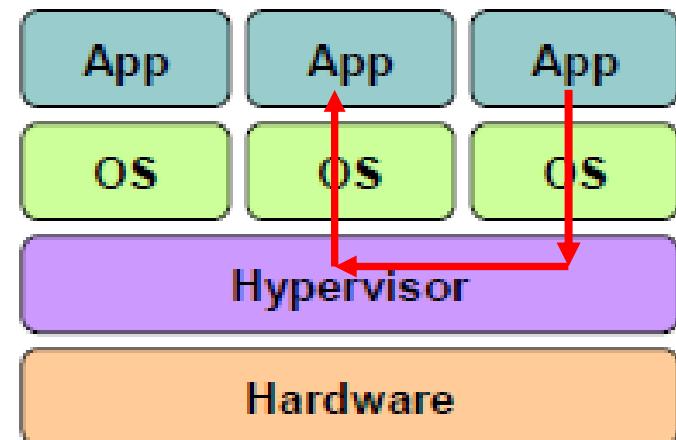


Issues in Cloud based systems

- Security issue due to multi-tenancy
 - Poor design leading to inadvertent sharing of information
 - Virtual machine 'Escape'

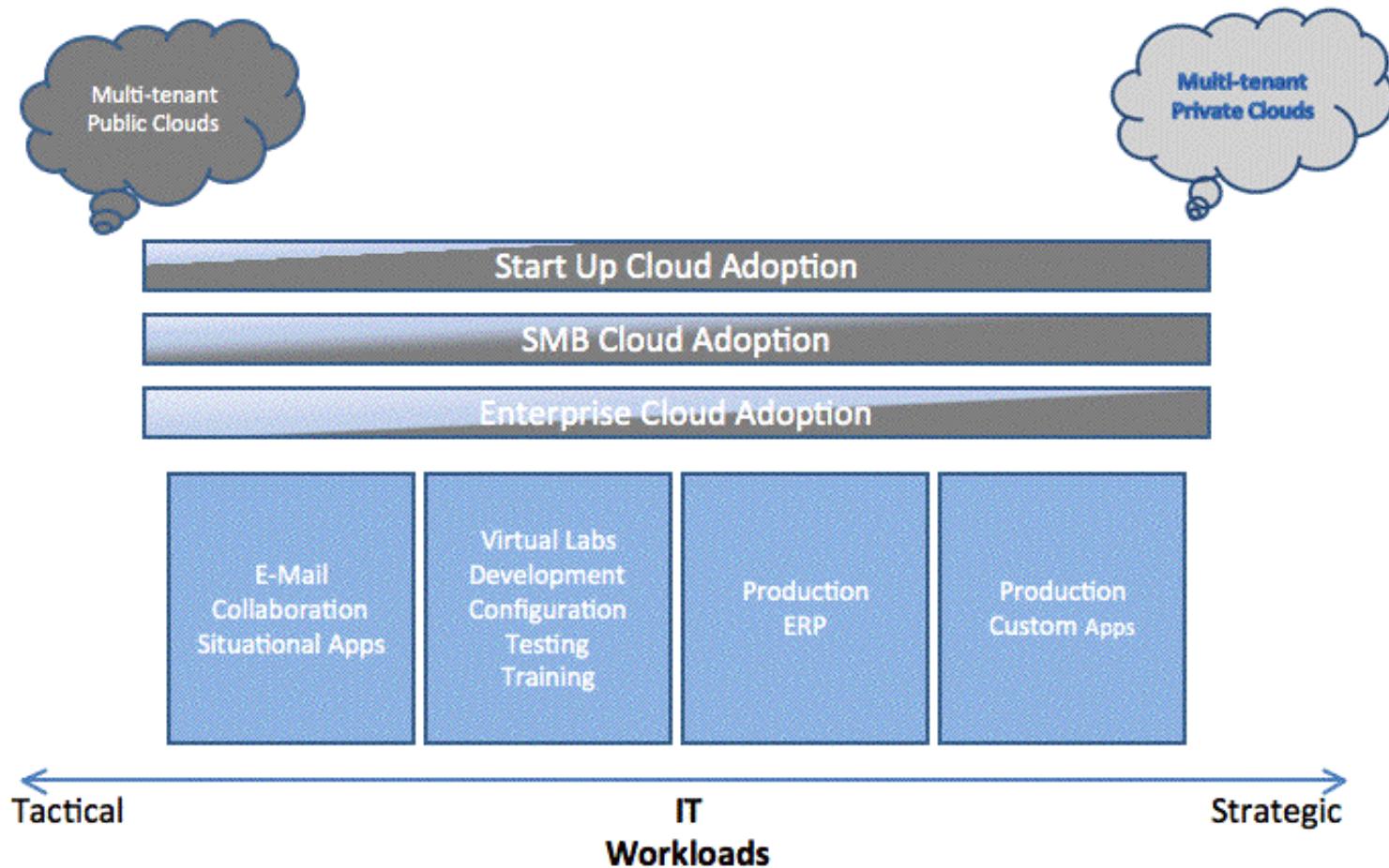
Other fields of the table					OU_ID
					Org1
					Org1
					Org1
					Org2
					Org2

Same table contains data of different organizations



Virtual machine 'Escape'

How to choose your multi-tenancy degree?



IT workloads will reside in a hybrid environment of public and private clouds. Both will be multi-tenant.

How to choose your multi-tenancy degree?



The **characteristics of the workload** in question have to be carefully studied first, including the workload's **utilitarian versus strategic value, volatility, security, etc.**

Higher degrees of multi-tenancy are best suited for cross-industry utilitarian workloads such as **e-mail, expense reporting, travel authorization and sales force management.**

These applications can very easily share the **same schema**.

Degree of multi-tenancy

- Highest degree: IaaS and PaaS are multi-tenant. SaaS is fully multi-tenant also.
 - Middle degree: IaaS and PaaS are multi-tenant. Small SaaS clusters are multi-tenant.
 - Lowest degree: IaaS and PaaS are multi-tenant. SaaS is single tenant.
-

‘Cloud Native’ applications

- Cloud-native computing takes advantage of many modern techniques, including
 - PaaS,
 - multicloud,
 - microservices,
 - agile methodology,
 - containers,
 - CI/CD, and
 - devops



What is cloud
native?

Exercise

When would you use the following options:

a) App 1

Web tier – Multi-tenant

App tier – Single tenant

Data tier – Multi-tenant

b) App 2

Web tier – Single tenant

App tier – Multi-tenant

Data tier – Single tenant

Exercise

When would you use the following options:

a) App 1

Web tier – Multi-tenant

App tier – Single tenant

Data tier – Multi-tenant

Answer: Web tier processing is light, App processing is heavy, data is not confidential

b) App 2

Web tier – Single tenant

App tier – Multi-tenant

Data tier – Single tenant

Answer: Web tier processing is heavy, App processing is light, data is confidential

Mobile Application Architecture



Types of mobile apps	Characteristics
Native app	<p>Makes use of OS and native devices.</p> <p>Ex. Games</p>
Cross platform app	<p>Same code runs on multiple mobile platforms such as Android and iOS</p>
Mobile web application	<p>Has a mobile component which interacts with a server component.</p> <p>Ex. Uber, PayTM, Banking</p>

Tools and technology for mobile app development



Development Approach	Native	Cross-Mobile Platforms	Mobile Web
Definition and Tools	<p>Build the app using native frameworks:</p> <ul style="list-style-type: none">- iPhone SDK- Android SDK- Windows Phone SDK	<p>Build once, deploy on multiple platforms as native apps:</p> <ul style="list-style-type: none">- RhoMobile- Titanium Appcelerator- PhoneGap- Worklight- Etc.	<p>Build using web technologies:</p> <ul style="list-style-type: none">- HTML5- Sencha- JQuery Mobile- Etc.
Underlying Technology	<ul style="list-style-type: none">- iPhone: Objective C- Android: Java- Windows Phone: .NET	<ul style="list-style-type: none">- RhoMobile: Ruby on Rails- Appcelerator: Javascript, HTML- PhoneGap: Javascript, HTML- Worklight: Javascript, HTML	<ul style="list-style-type: none">- Javascript, HTML

Xamarin – cross platform tool



Module 9 Part 3

Big Data technologies

BITS Pilani

Harvinder S Jabbal
SSZG653 Software Architectures

Contents

1. Big data
 2. Hadoop, HDFS, Map-Reduce
 3. Analytics & Real time analytics
 4. In-Memory database
 5. NoSQL databases
-

Big data & Analytics

[Wikipedia](#) defines "***Big Data***" as a collection of data sets so **large and complex** that it becomes difficult to process using on-hand database management tools or traditional data processing applications.

History of Big Data

Lots of data got created due to

- Proliferation of Internet
- Social media
- eCommerce

Before we begin, let us understand the scale of data we deal with

Unit Power of 10

Mega byte	6
Giga	9
Tera	12
Peta	15

Big Data Statistics

- Volume of data
 - 500+ new websites are created every minute of the day
 - 100 Terabytes of data is uploaded to Facebook every day
 - Twitter generates 12 Terabytes of data every day
 - YouTube users upload 48 hours of new video content **every minute** of the day
- Processing
 - Decoding of the human genome used to take 10 years. Now it can be done in 7 days
 - Facebook Stores, Processes, and Analyzes more than 30 Petabytes of user generated data
 - LinkedIn processes and mines Petabytes of user data to power the "People You May Know" feature

Source: [Wikibon - A Comprehensive List of Big Data Statistics](#)

Characteristics of Big data



Surveillance videos, satellites images, cell phone location, health of power station turbines, furnaces & other industrial machinery, weather and meteorological data, click stream

Archived data:
Patient records, Scanned copies of agreements, records of ex-employees/completed projects, banking transactions older than the compliance regulations

Ex. Trading/stock exchange data, tweets on Twitter, status updates/likes/shares on Facebook

Use of big data

- Recommend cancer medication based on what worked well in similar situations for other patients
 - Weather prediction for fishermen, farmers
 - Predict equipment malfunctioning in large nuclear power plant, chemical plants, etc.
 - Credit card fraud detection
-

Example of handling big data

Google search

Do you know how Google Search works?

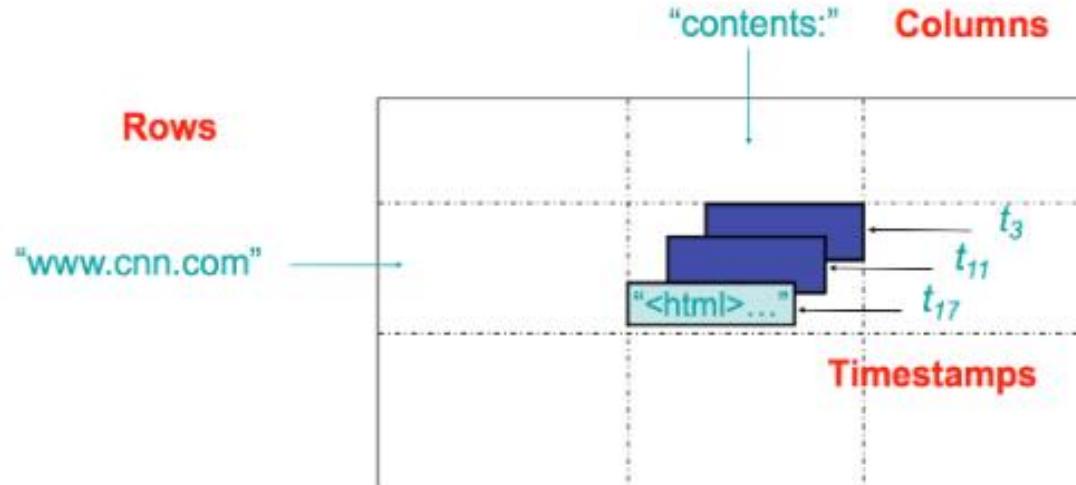
3 steps:

- **Catalog & index:** Even before you search, Google goes through various web sites and linked web sites (crawling) and catalogs all the web sites. This runs into several Tera bytes.
- **Understand & enrich your query:** Correct spelling mistakes, consider synonyms, etc.
- **Search:** When the search is requested, it uses this catalog to determine which web sites closely match the requirement. For this it uses a 'Page Rank' (named after Larry Page) algorithm which considers factors such as which page has max number of occurrences of the search string, how many other web sites refer to this web site, what is the reputation of the websites that refer to this website, etc.

Google – Big table

So Google invented a data storage structure called Big Table

Distributed multi-dimensional sparse map
 $(row, column, timestamp) \rightarrow cell\ contents$



Rows are ordered lexicographically

Google BigTable

Google BigTable is a wide table containing several attributes of a website;

Here are some attributes:

- The contents of Web page
- Anchor text*
- Websites referencing the page.
- Time stamp when the data was stored

Google BigTable is built on technologies like Google File System (GFS)

Google BigTable is used by applications such as Google Maps, Google Analytics, etc.

*Anchor text is the text that appears in Blue colour in search result. It contains a link to the website

Google Big Table

Features:

- Versioning of data,
- Compression,
- Distribution across servers,
- Fault tolerant,
- Fast access,
- Dynamic addition of servers,
- Load balancing

Google disclosed the design of Big table. Then came Hadoop Distributed File System (Yahoo) and several NoSQL databases

Use of Big Data

Banking and Financial Services

- Fraud Detection to detect the possible fraud or **suspicious transactions in Accounts, Credit Cards, Debit Cards, and Insurance etc.**

Retail

- Targeting customers with different discounts, coupons, and promotions etc. based on demographic data like gender, age group, location, occupation, dietary habits, **buying patterns**, and other information which can be useful to differentiate/categorize the customers.

Sentiment Analysis

- Organizations use the data from social media sites like Facebook, Twitter etc. to understand **what customers are saying about the company, its products, and services.**
- Words like “I like this phone”, “This food is too salty”, etc.. indicate sentiments
- This type of analysis is also performed to understand which companies, brands, services, or technologies people are talking about.

Use of Big Data ...

Customer Service

- IT Services and BPO companies analyze the call records/logs **to gain insights into customer complaints and feedback**, call center executive response/ability to resolve the ticket, and to improve the overall quality of service.
- Call center data from telecommunications industries can be used to **analyze the call records/logs and optimize the price, and calling plan, messaging plan, and data plans**

Industrial equipment monitoring & alerting

- A large power plant or chemical factory has thousands of critical equipment that needs to be monitored
- The equipment data needs to be analysed to detect any malfunctioning or danger of accidents

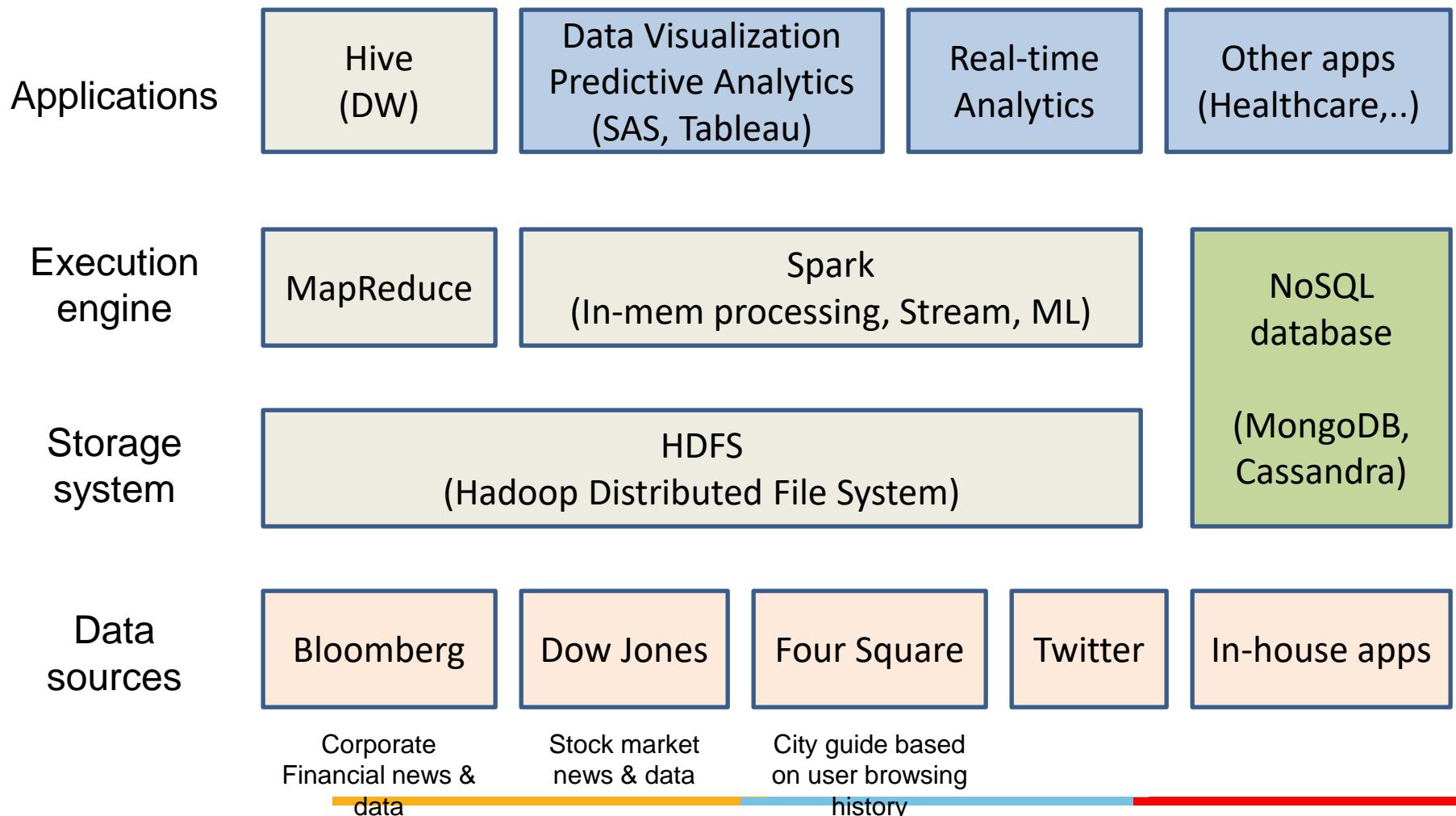
Weather forecasting

- Satellite data from remote sensing satellites need to be analysed at high speed to warn fishermen, farmers and public about potential cyclones, delayed monsoon, etc.

BigTable can be used to store...

- **Time-series data**, such as CPU and memory usage over time for multiple servers.
 - **Marketing data**, such as purchase histories and customer preferences.
 - **Financial data**, such as transaction histories, stock prices, and currency exchange rates.
 - **Internet of Things data**, such as usage reports from energy meters and home appliances.
 - **Graph data**, such as information about how users are connected to one another.
-

Big data architecture / Eco-system



Hadoop

Hadoop is an open source framework, from the Apache foundation, capable of **processing large amounts of heterogeneous data sets** in a **distributed fashion** across **clusters** of commodity computers and hardware using a simplified programming model.

Hadoop provides a **reliable** shared storage and **analysis** system.

Components of Hadoop

HDFS (Hadoop Distributed File System)

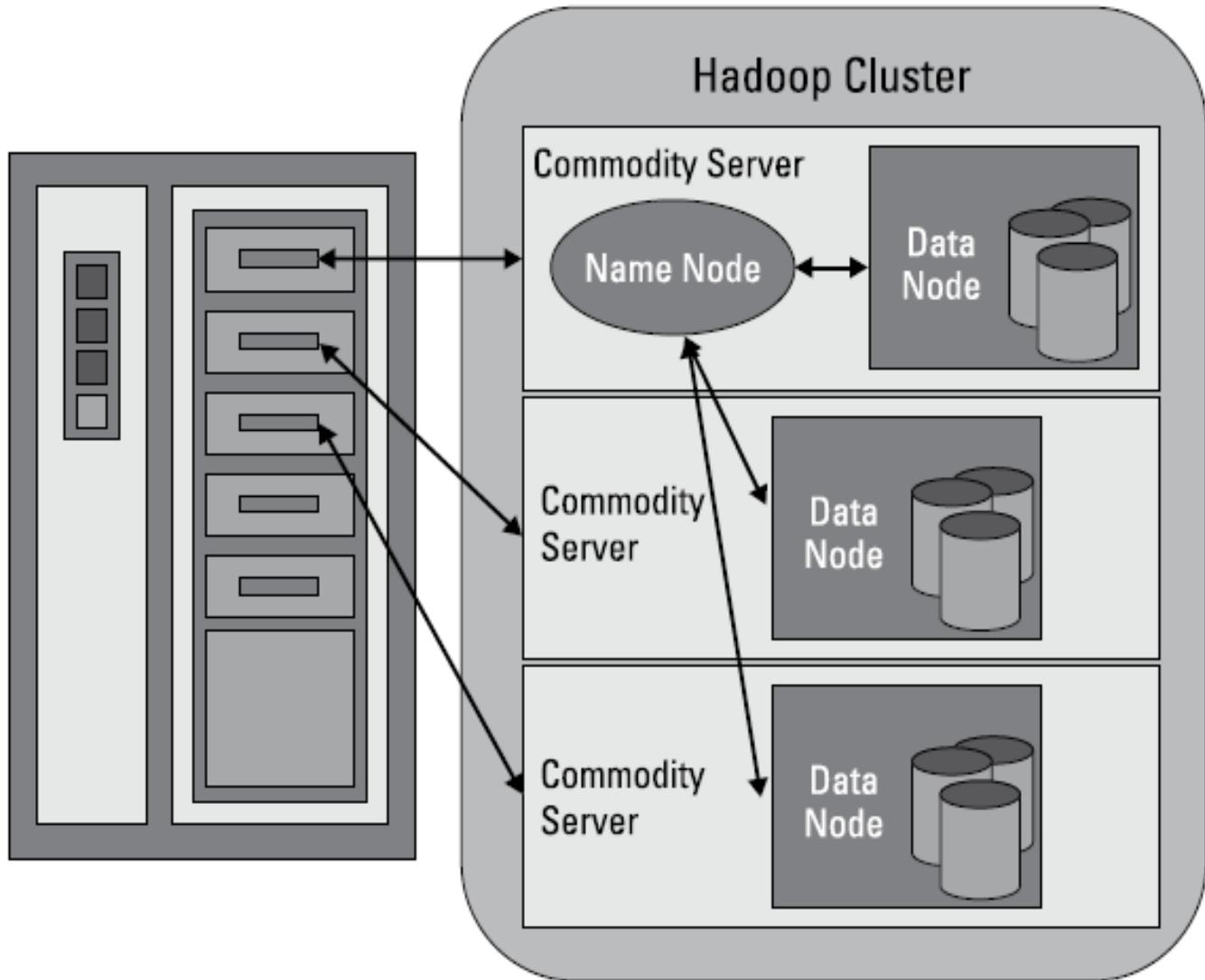
- HDFS offers a highly reliable and **distributed storage**, and ensures reliability, even on a commodity hardware, by **replicating** the data across multiple nodes.
- Unlike a regular file system, when data is pushed to HDFS, it will automatically split into multiple blocks (configurable parameter) and stores/replicates the data across various data nodes. This ensures high availability and fault tolerance.

MapReduce

- MapReduce offers an **analysis system** which can perform complex computations on large datasets.
- This component is responsible for performing all the computations and works by **breaking down** a large complex computation **into multiple tasks** and assigns those to **individual worker/slave nodes** and takes care of coordination and consolidation of results

Hadoop - HDFS

Figure 9-1:
How a
Hadoop
cluster is
mapped to
hardware.



Ref: Big data for Dummies

Hadoop - HDFS

Data

- Large files are broken down into blocks (128 MB usually) and spread across Data nodes.
- Data blocks are replicated and Degree of replication can be adjusted

Meta data

- Name node stores meta data – data about files, distribution of data (which block is in which nodes), etc.
- For good performance, all the metadata is loaded into the physical memory of the NameNode server.

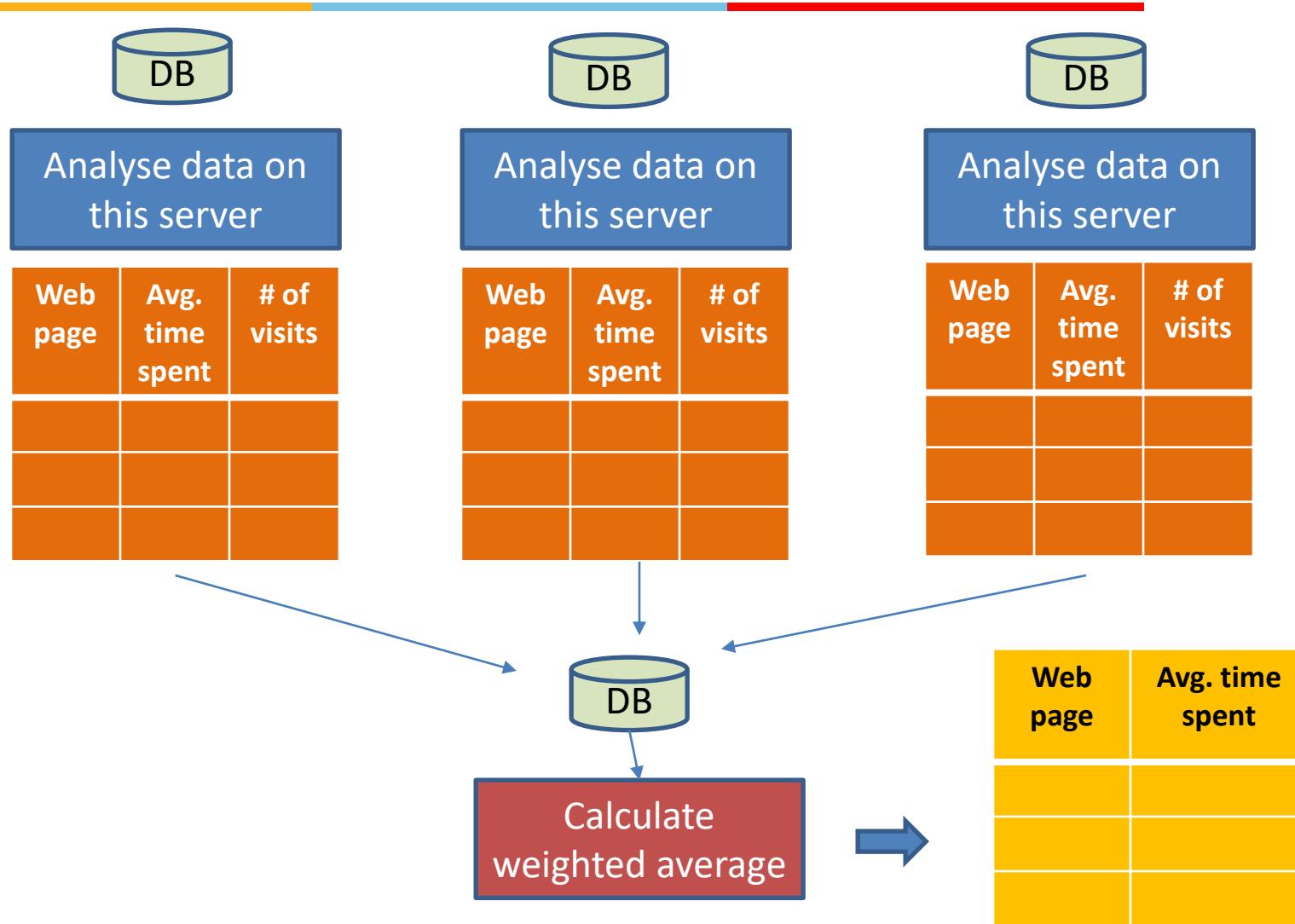
Features

- Data nodes provide Heart beat messages to Name node
- Supports data pipelines. A connection between data nodes to move data from one node to another
- Rebalancer: Balances distribution of data

Map-Reduce pattern

- Used to analyse vast amount of data
- Suppose we keep track of every click of the user on a web site and store these details in a database
- Let us say we want to find out the average time spent by users on each web page of the web site, across thousands of users who visited the web site in the last 30 days
- How can we speed up the analysis?

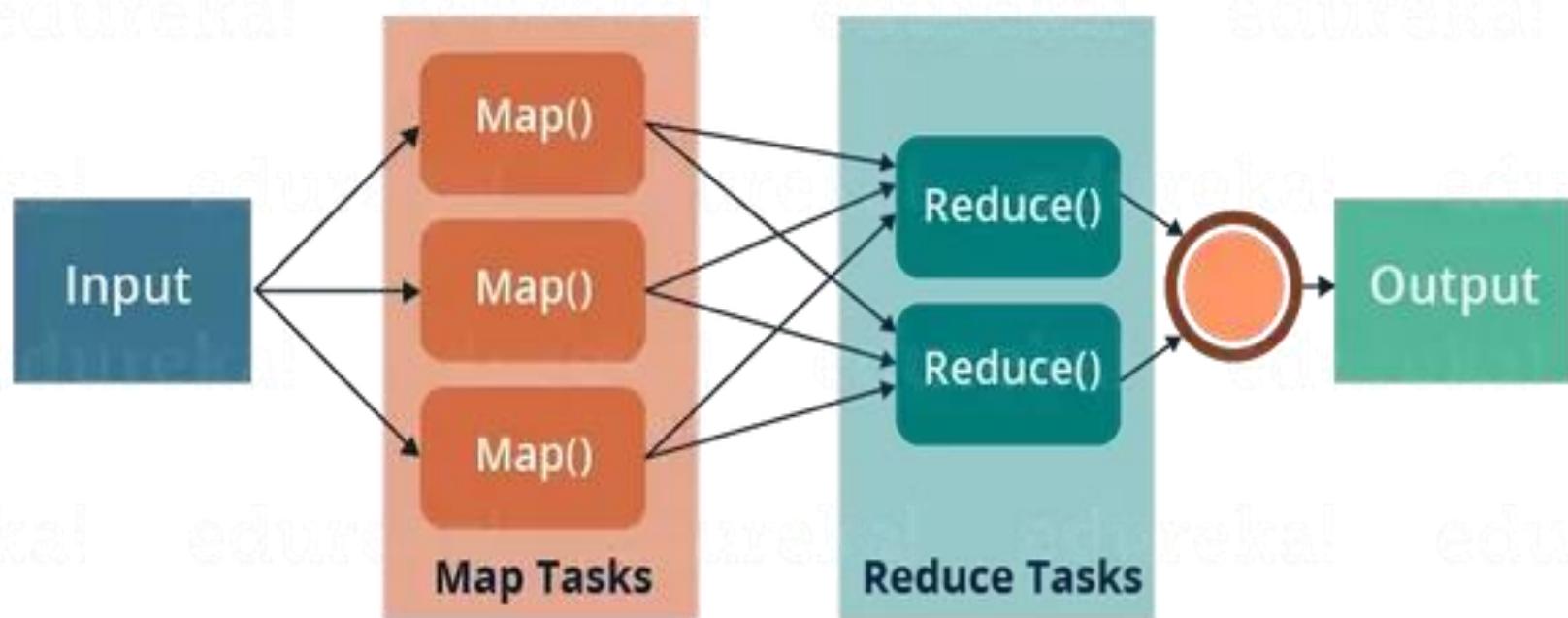
Map-Reduce pattern: Example



Map-Reduce pattern

- Executes in parallel
- Leads to low latency & high availability
- Map performs extract & transform and produces <Key, Value> instances
- Reduce summarizes transformed data

Map Reduce pattern



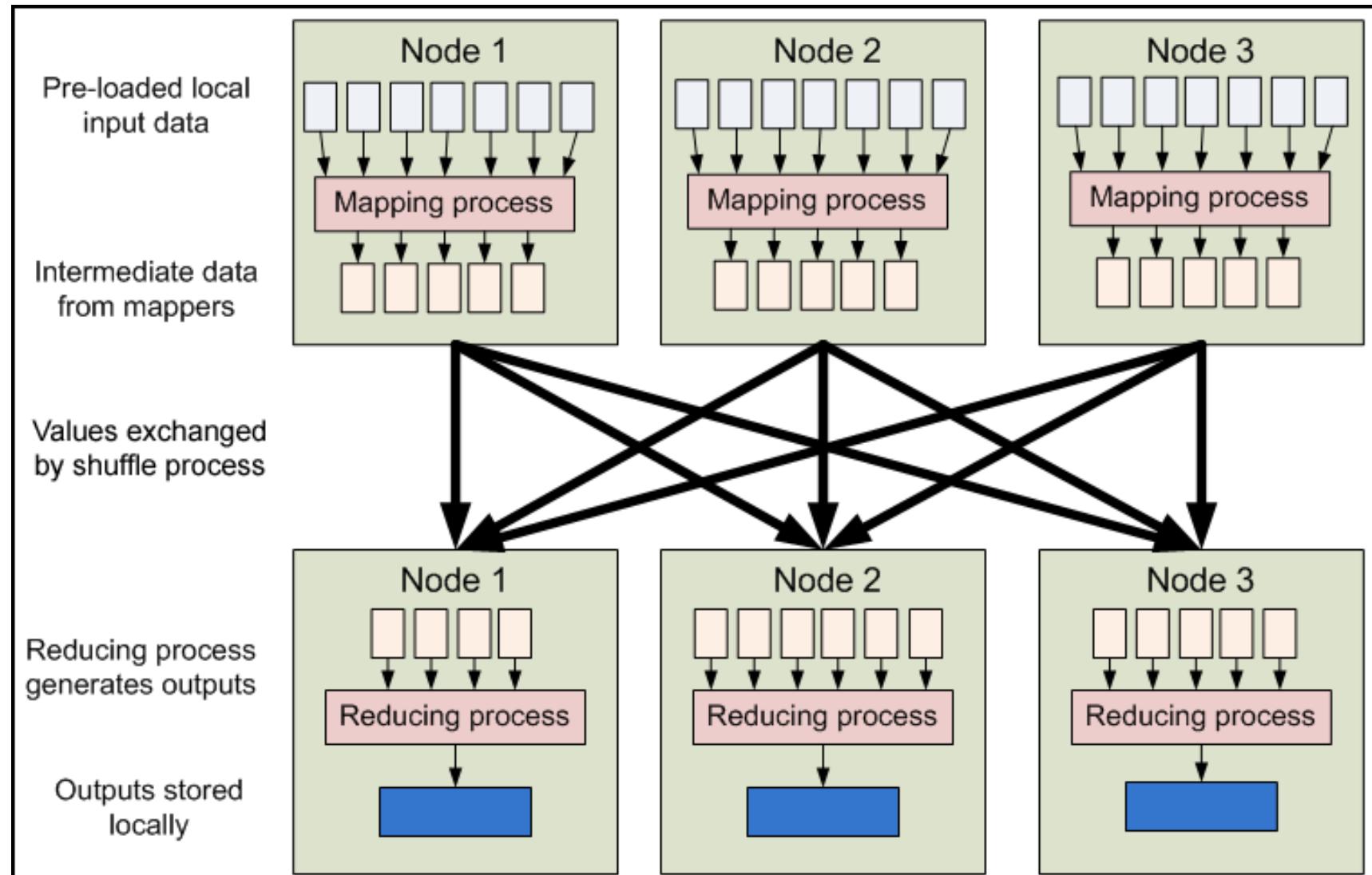
Source: <https://www.quora.com/What-is-the-relationship-between-MapReduce-and-Hadoop>

Map-Reduce pattern

- **Example:** Determine the average time (duration) spent by users on different web pages
- Step 1: **Map** processes data on each node and outputs <Web page, (Avg time, # of users)>
- Step 2: **Reduce** produces <Web page, weighted Avg time>

Map – Reduce pattern

Example: Determine the average duration spent by users on different web pages



Experience Sharing

Have you come across systems that use this pattern?

Hadoop - HDFS

Features

- Can store peta bytes of data
- Distributed
- Replicated
- Fault tolerant (self healing)

Using Hadoop

When to Use Hadoop (Hadoop Use Cases)

Hadoop can be used in various scenarios including some of the following:

- Analytics
- Search
- Data Retention
- Log file processing
- Analysis of Text, Image, Audio, & Video content
- Recommendation systems like in E-Commerce Websites

When Not to Use Hadoop

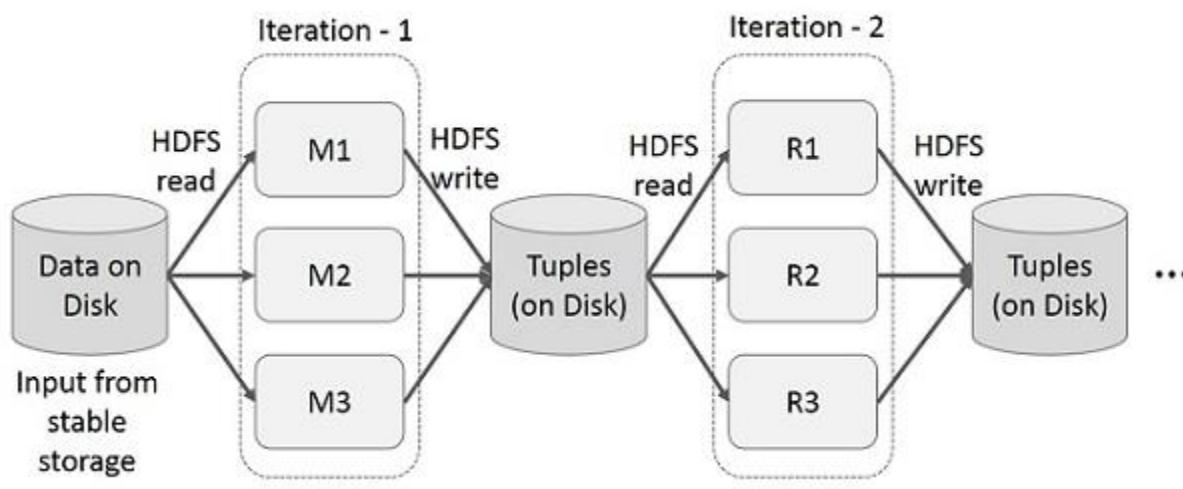
There are few scenarios in which Hadoop is not the right fit. Following are some of them:

- Low-latency or near real-time data access.
- If you have a large number of small files to be processed. This is due to the way Hadoop works. Namenode holds the file system metadata in memory and as the number of files increases, the amount of memory required to hold the metadata increases.
- Multiple writes scenario or scenarios requiring arbitrary writes or writes between the files.

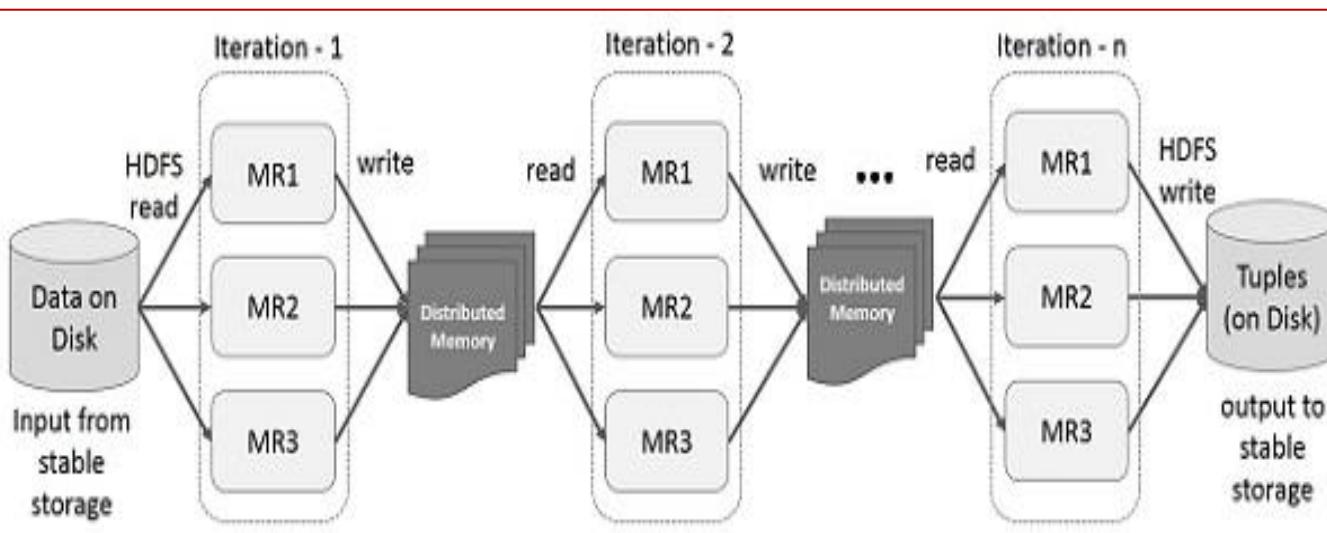
Hadoop may still be slower for some use cases

- Sometimes we require even faster processing than Map-Reduce, for example in real time fraud detection
- Map Reduce is disk based
- If we can retrieve disk data into memory and then use it for further processing, we can get even better response time

Difference between Hadoop & Spark



Iterative operation using Hadoop using disk



Iterative operation using Spark using memory

Source:
https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm

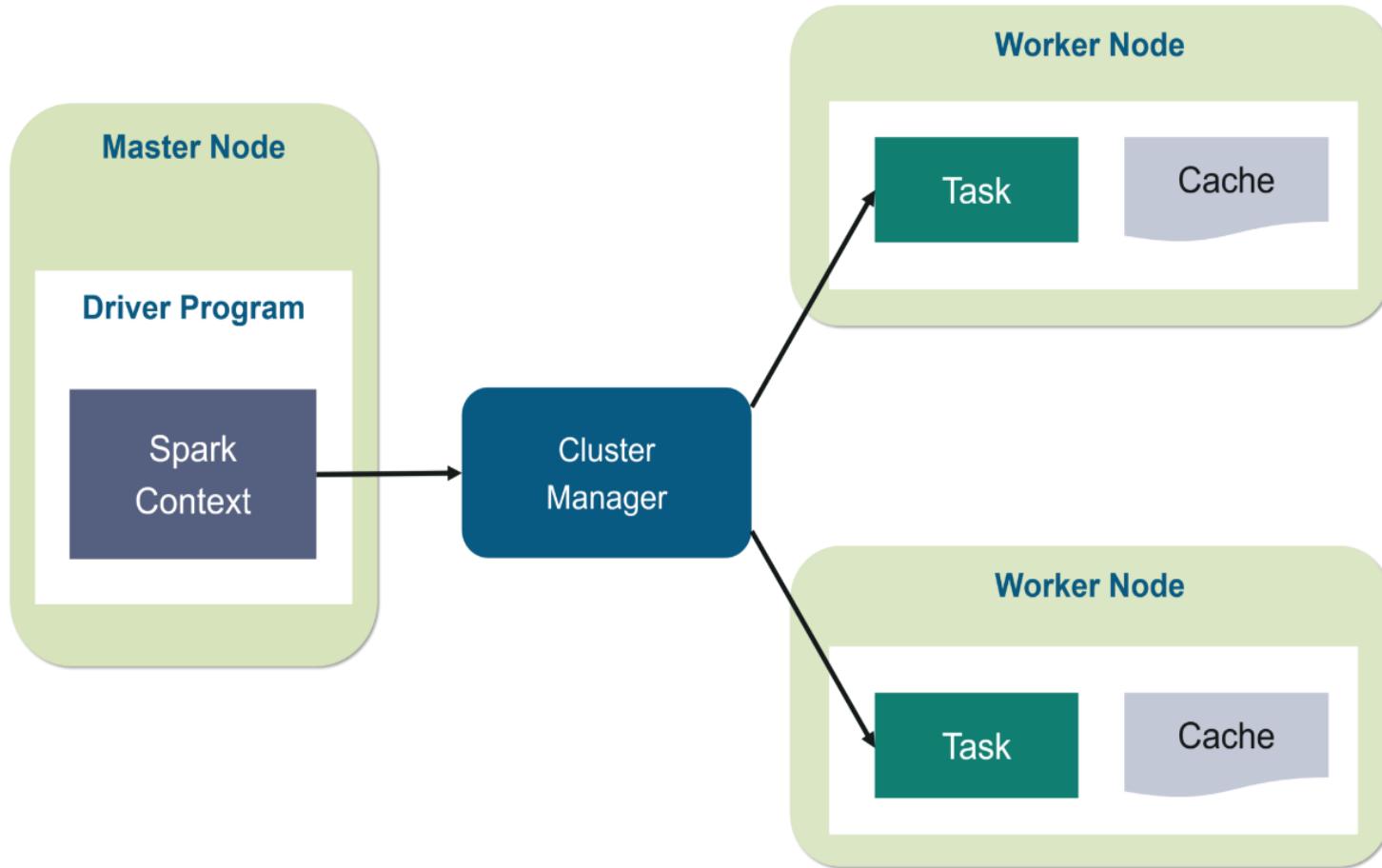
Apache Spark

- Apache Spark is open source, general-purpose distributed computing engine used for processing and analyzing a large amount of data
- Main feature: In-memory cluster computing
- Useful for real time computations

Apache Spark Components



Spark architecture

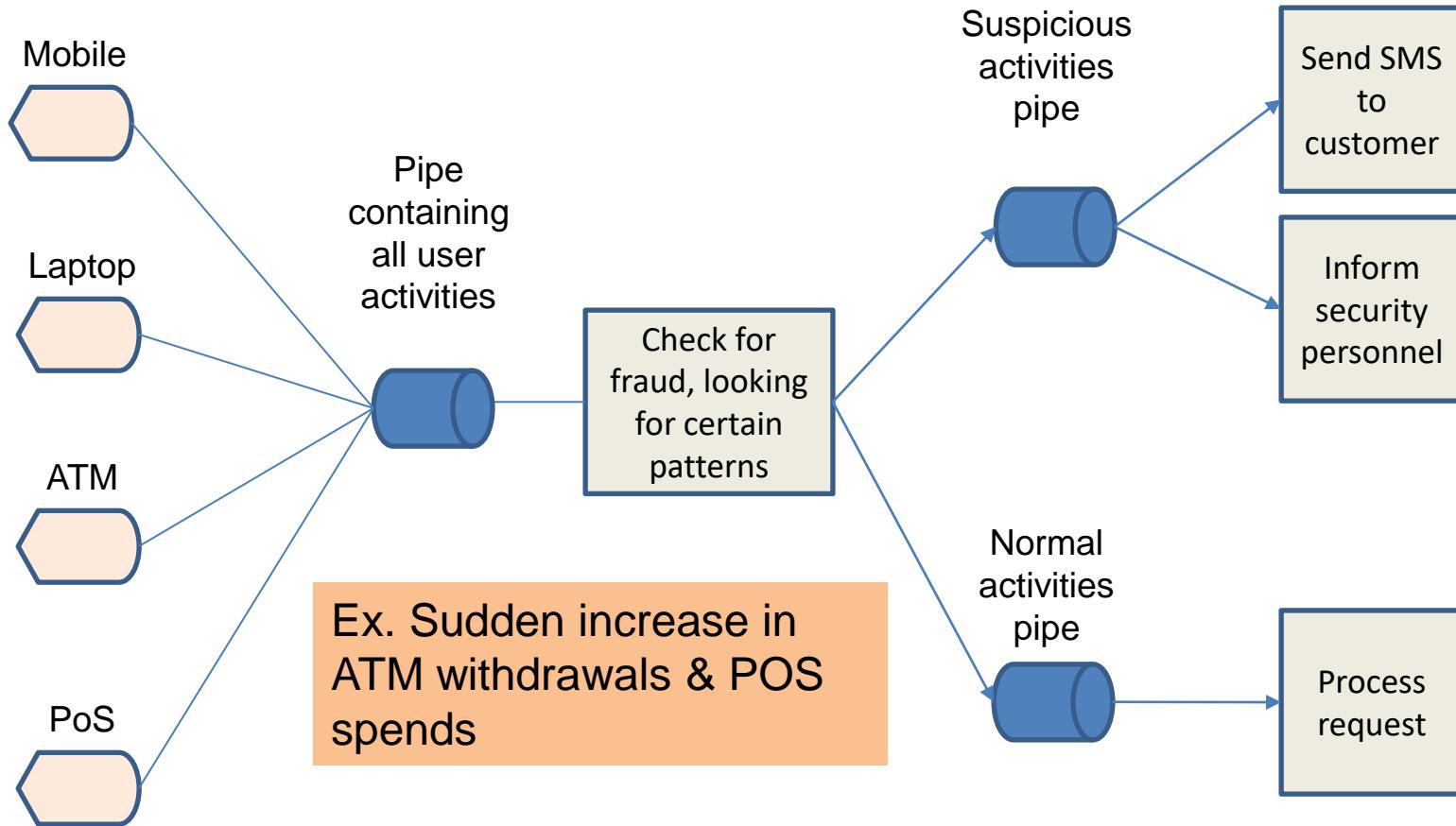


Real time analytics

- Real time analytics lets users see, analyze and understand data as it arrives in a system.
- It can give users insights for making real-time decisions.
- Examples:
 - Real time advertising
 - Identify security breaches
 - Sensor data processing to predict issues in machines

Real time analytics - Fraud detection in bank

Continuous monitoring of client's activity to see if there are any potential issues



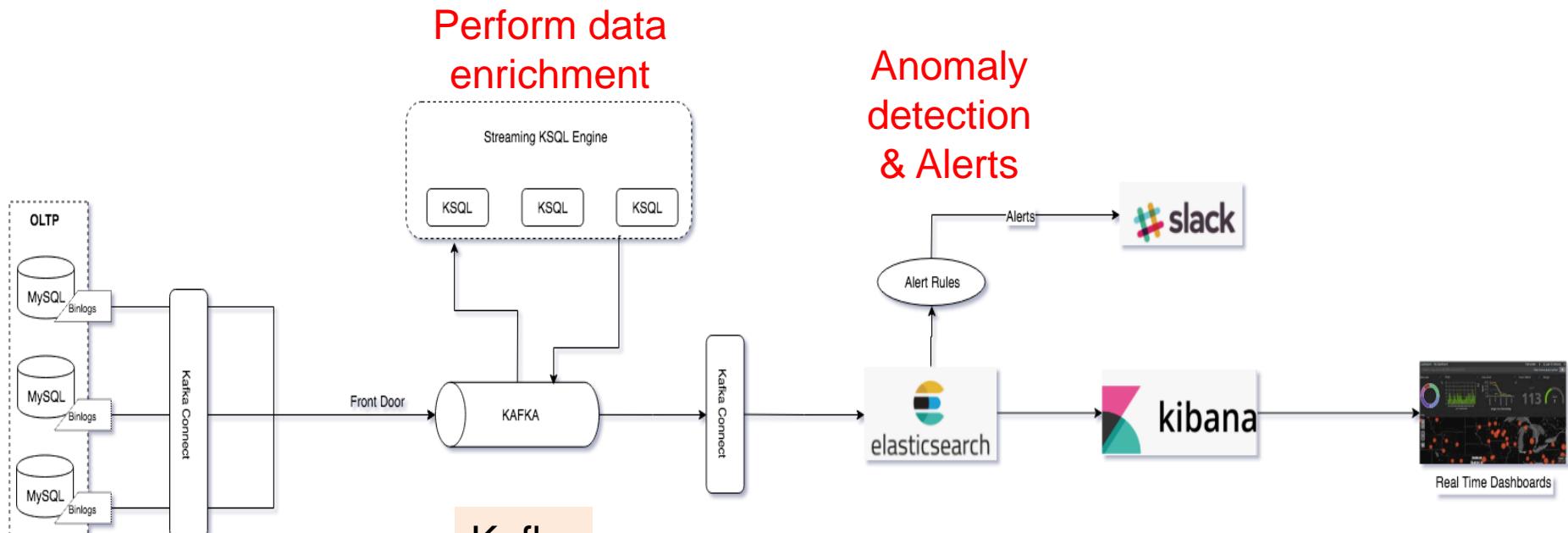
Real time analytics at Dream11 – a fantasy sports platform



Objectives:

1. Know the real-time rate of contest joins
2. Know the real-time aggregated status of payment gateways
3. Identify real-time anomalies eg: unusual traffic on the system
4. Realtime aggregated view of outcome of marketing campaigns
5. How customers are using discount coupons once promotion goes live
6. Realtime alerting once Mega contest is above 90%

Real time analytics at Dream11 – a fantasy sports platform



Transaction
systems

Ref: medium.com

Real-time
data store

Monitoring



Dream11 case
study

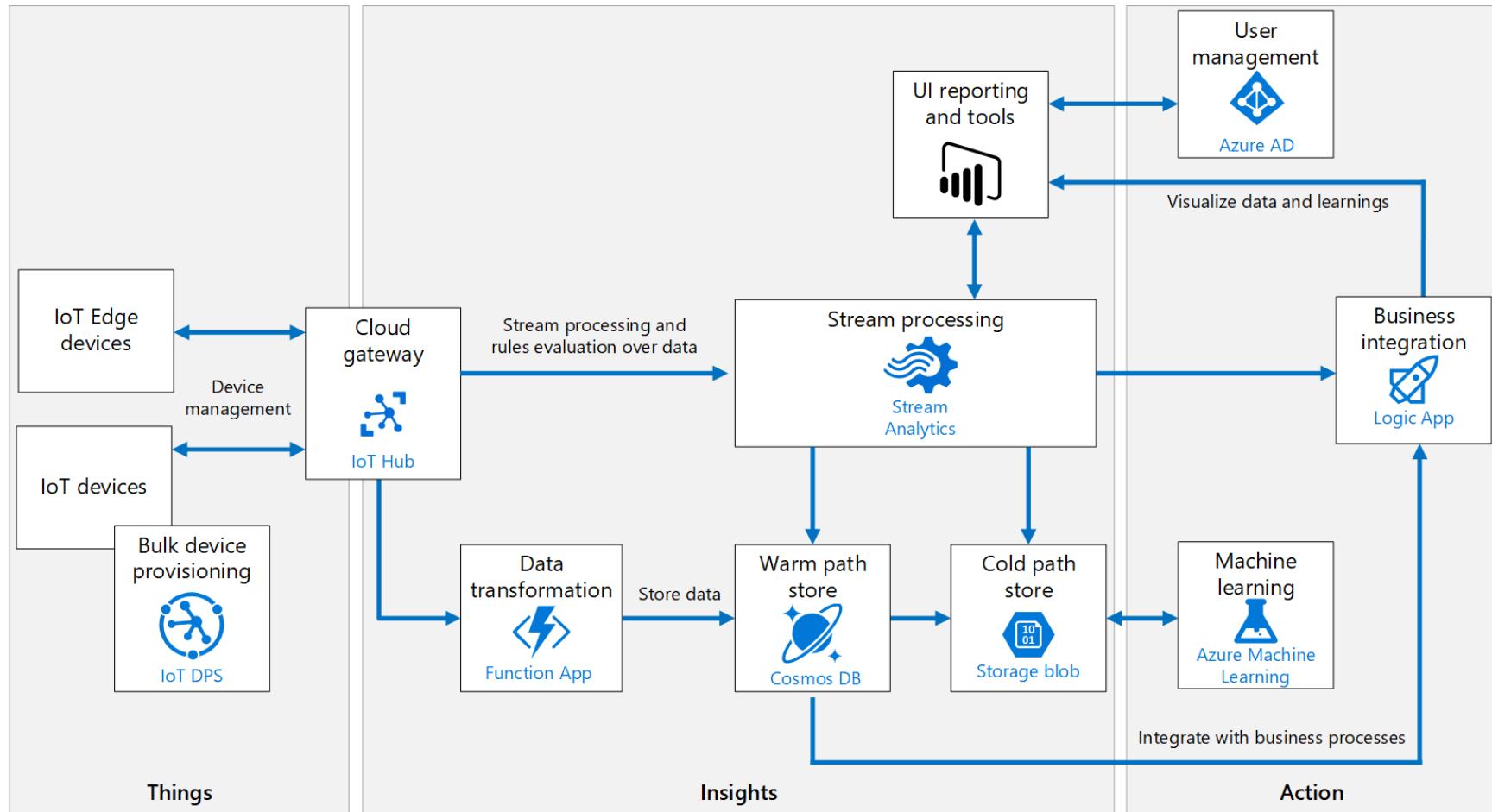
IoT: Internet of Things

Inter-connected devices which work together and perform operations with little human intervention

Examples:

- Tracking machine parameters using sensors and controlling for optimum performance
 - Tracking goods, real time information exchange about inventory among suppliers and retailers
 - Sensing for soil moisture and nutrients, controlling water usage for plant growth
-

Azure IoT reference architecture



Appendix



Module 9 Part 4

NoSQL Databases

BITS Pilani

Harvinder S Jabbal
SSZG653 Software Architectures

NoSQL databases

- While **Relational databases** (SQL databases) are good for transaction management, not all applications need this feature.
- Many **web applications** such as marketing applications, IoT applications, **need fast processing** of data but do not need transaction management
- NoSQL databases came as a response to this need

<https://www.dataversity.net/a-brief-history-of-non-relational-databases/#>

NoSQL databases

Traditional databases also have limitations on storage

NoSQL databases are

- Scalable (Sharding)
- Fast (In Memory)
- Available (Replication)
- Handle semi-structured and unstructured data
- Rapidly adapt to changing data needs

However most lack

- Transaction support (ACID)
- Join feature

Examples of usage of NoSQL DB



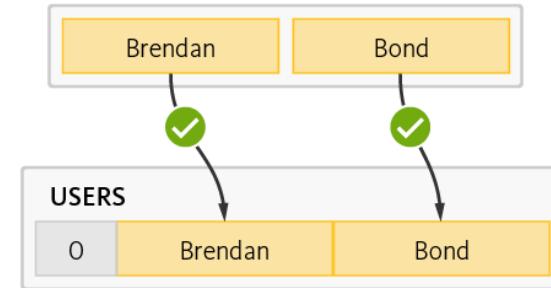
- **Tesco**, Europe's #1 retailer, uses NoSQL to manage **millions of products**, promotions supply chain, etc.
- **Ryanair**, the world's busiest airline, uses NoSQL to **power its mobile app** serving over 3 million users
- **Marriott** is deploying NoSQL for its **reservation system** that books \$38 billion annually
- **Gannett** (USA Today) the #1 U.S. newspaper publisher, uses NoSQL for its proprietary **content management** system, Presto
- **GE** is deploying NoSQL for its Predix platform to help manage the **Industrial Internet**

NoSQL - Flexibility

In RDBMS, if we want to add a new column, we need to change the schema

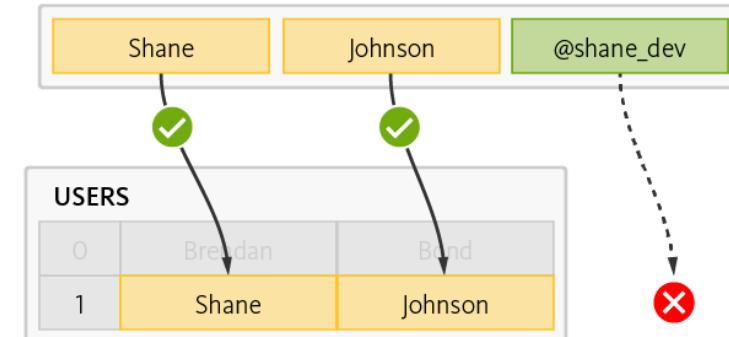
Iteration 1 — First, Last

Schema Utilized		
USERS		
ID	First	Last



Iteration 2 — First, Last, Twitter

Schema Utilized		
USERS		
ID	First	Last

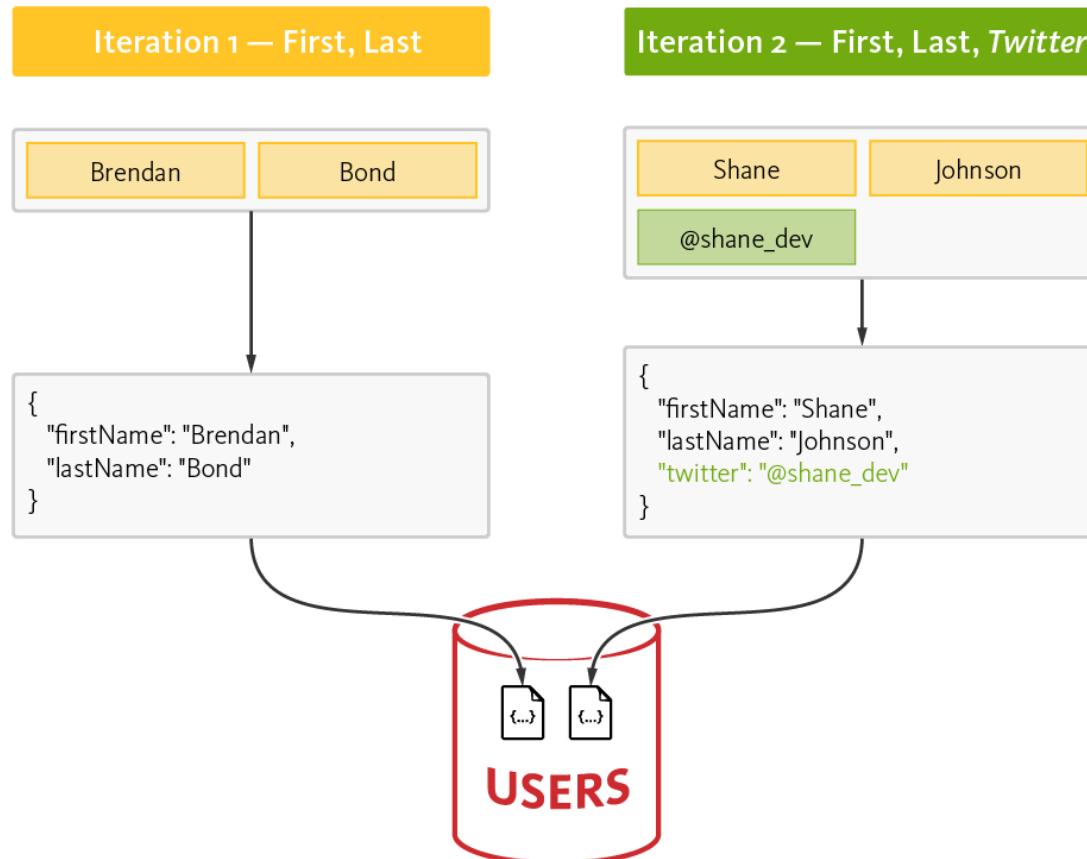


This can not be stored

Ref: <https://www.couchbase.com/resources/why-nosql>

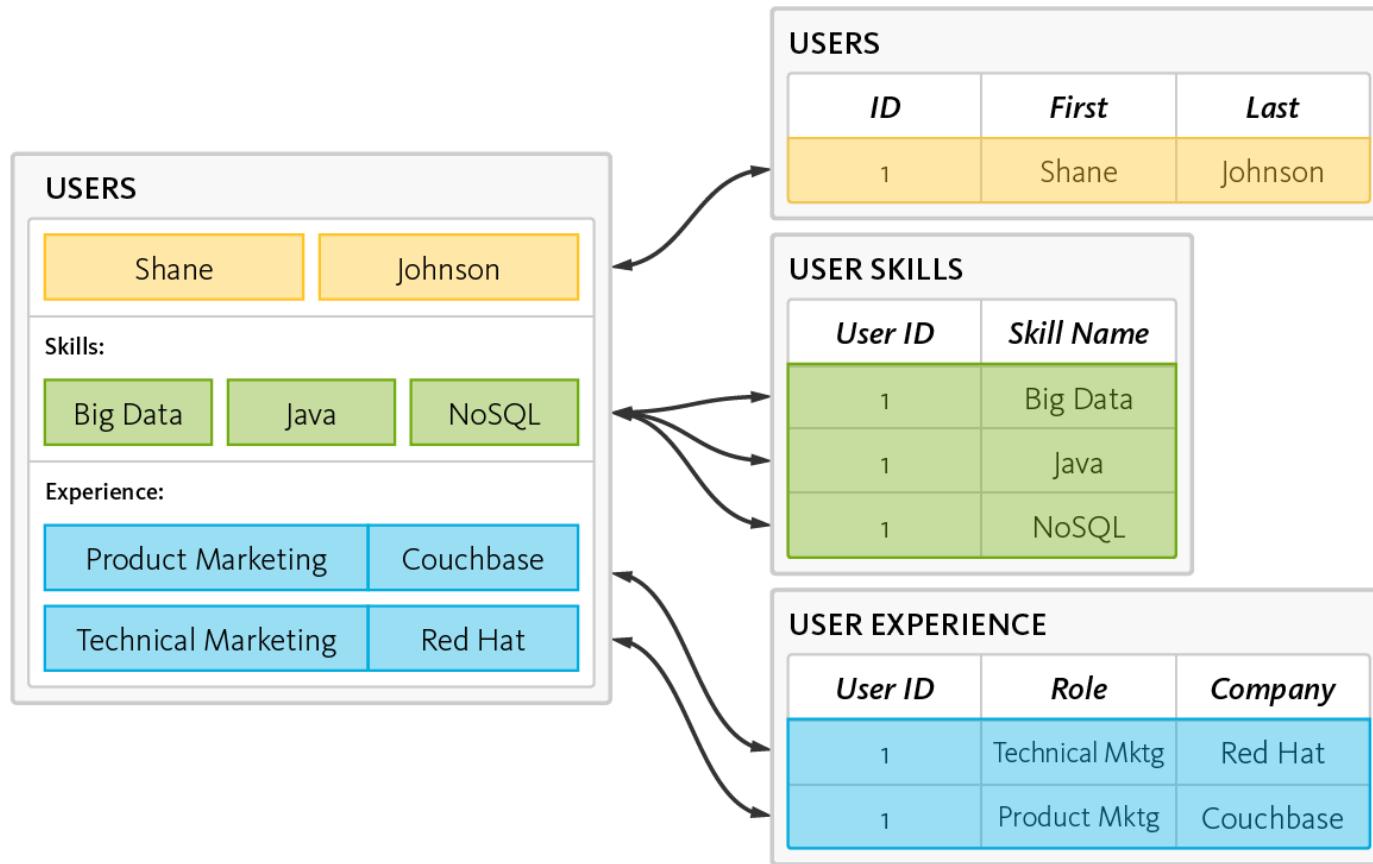
NoSQL - Flexibility

In NoSQL DB, we can easily add new columns.



Ref: <https://www.couchbase.com/resources/why-nosql>

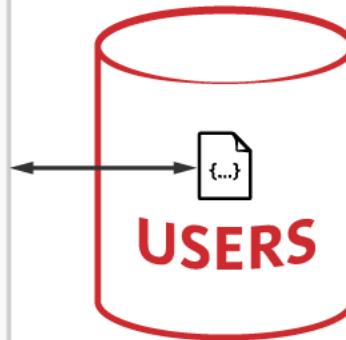
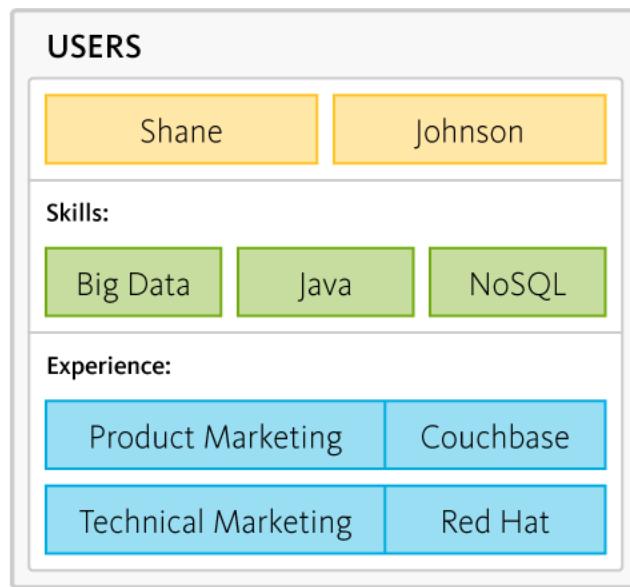
NoSQL - Simplicity



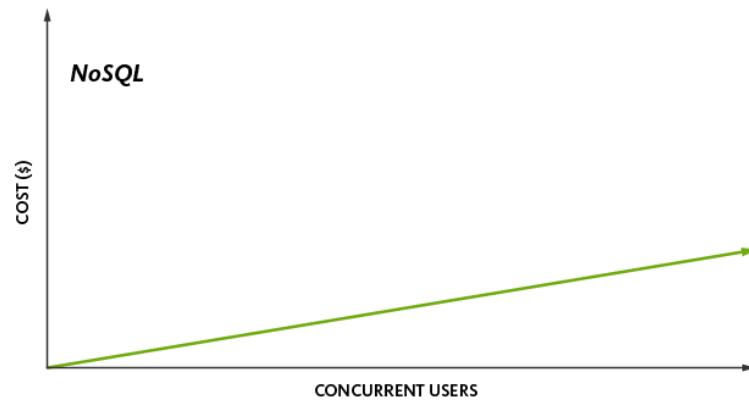
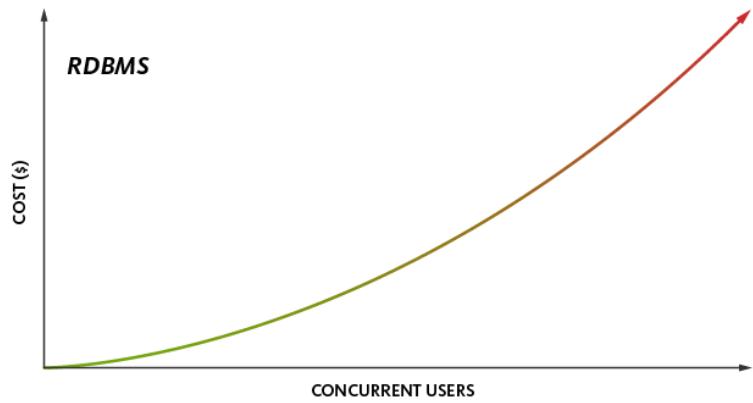
NoSQL

SQL

NoSQL - Simplicity



NoSQL – Cost effective



Cost: Memory, CPU, storage

NoSQL Database

Types of NoSQL databases:

- Document
- Key Value
- Column stores
- Graph

Document NoSQL database

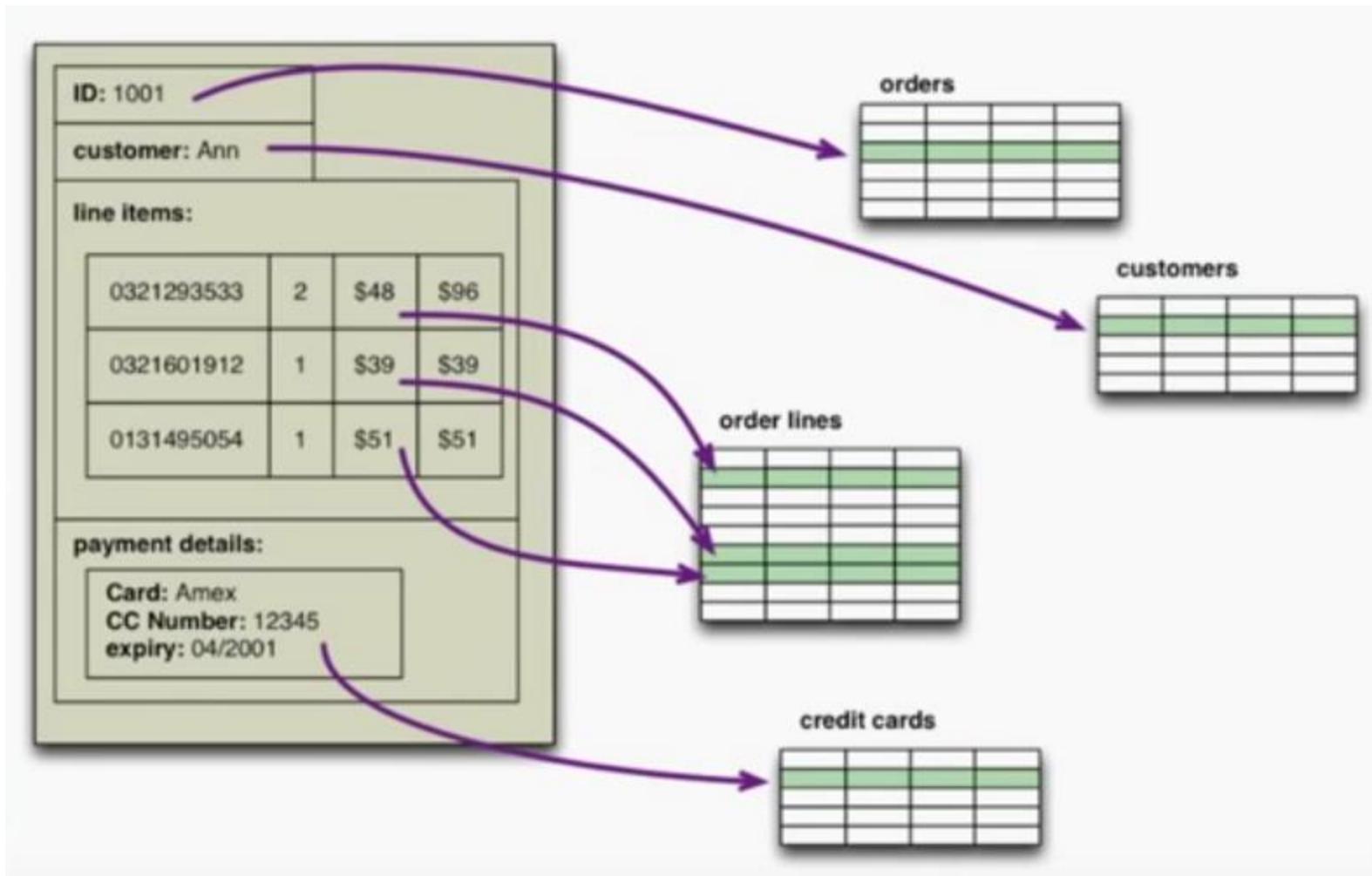
- The database stores and retrieves documents, which can be XML, JSON, BSON, and so on.
- Some of the popular document databases we have seen are [MongoDB](#), [CouchDB](#), [Terrastore](#), [OrientDB](#), [RavenDB](#), and of course the well-known and often reviled Lotus Notes that uses document storage.

```
<Key=CustomerID>

{
  "customerid": "fc986e48ca6" ← Key
  "customer":
  {
    "firstname": "Pramod",
    "lastname": "Sadalage",
    "company": "ThoughtWorks",
    "likes": [ "Biking", "Photography" ]
  }
  "billingaddress":
  {
    "state": "AK",
    "city": "DILLINGHAM",
    "type": "R"
  }
}
```

Example of one record

Document DB vs Relational DB



Good for

- Ecommerce platform
- Content management systems

Features of Mongo DB

- Indexing
- Ad hoc search
- Replication
- Partitioning / Sharding
 - Ex. Partition data by Product or Geography

Key-Value database

- Data model is Key value pair
- **Uses hashing for fast access**
- The DB does not care what is contained in the value
- Example scenarios: Phone directory, Stock trading
- Some of the popular key-value databases are [Riak](#), [Redis](#) (often referred to as Data Structure server), [Memcached](#) and its flavors, [Berkeley](#) [DB](#), [upscaledb](#) (especially suited for embedded use), Amazon DynamoDB (not open-source), Project Voldemort and [Couchbase](#).

Phone Directory

Key	Value
Bob	(123) 456-7890
Jane	(234) 567-8901
Tara	(345) 678-9012
Tiara	(456) 789-0123

Example of key value database

Stock Trading

This example uses a list as the value.

The list contains the stock ticker, whether its a “buy” or “sell” order, the number of shares, and the price.

Key	Value
123456789	APPL, Buy, 100, 84.47
234567890	CERN, Sell, 50, 52.78
345678901	JAZZ, Buy, 235, 145.06
456789012	AVGO, Buy, 300, 124.50

More examples: User profiles, Blog comments

Uses of Redis

- Session cache, with persistence

Column oriented database

This is useful for data analysis scenarios

Example: Calculate the average usage of electricity in 2018 in East Bangalore region

Traditional way: Read all the billing records of 2018

Cust id, Name, Addrs, Region, Month, Year, Usage, Amt,

Record 1: 001, John Mancha, Addr 1, East, Jan, 2018, 100, 600

Record 2: 002, Vivek Kulkarni, Addr 2, East, Jan, 2018, 90, 540

Record 3: 003, Shanti Sharma, Addr 3, West, Jan, 2018, 110, 660

....

....

Column oriented database

Instead if we store the data as follows:

Record 1: 001, John Mancha, Addr 1, East // Customer details

Record 2: 002, Vivek Kulkarni, Addr 2, East

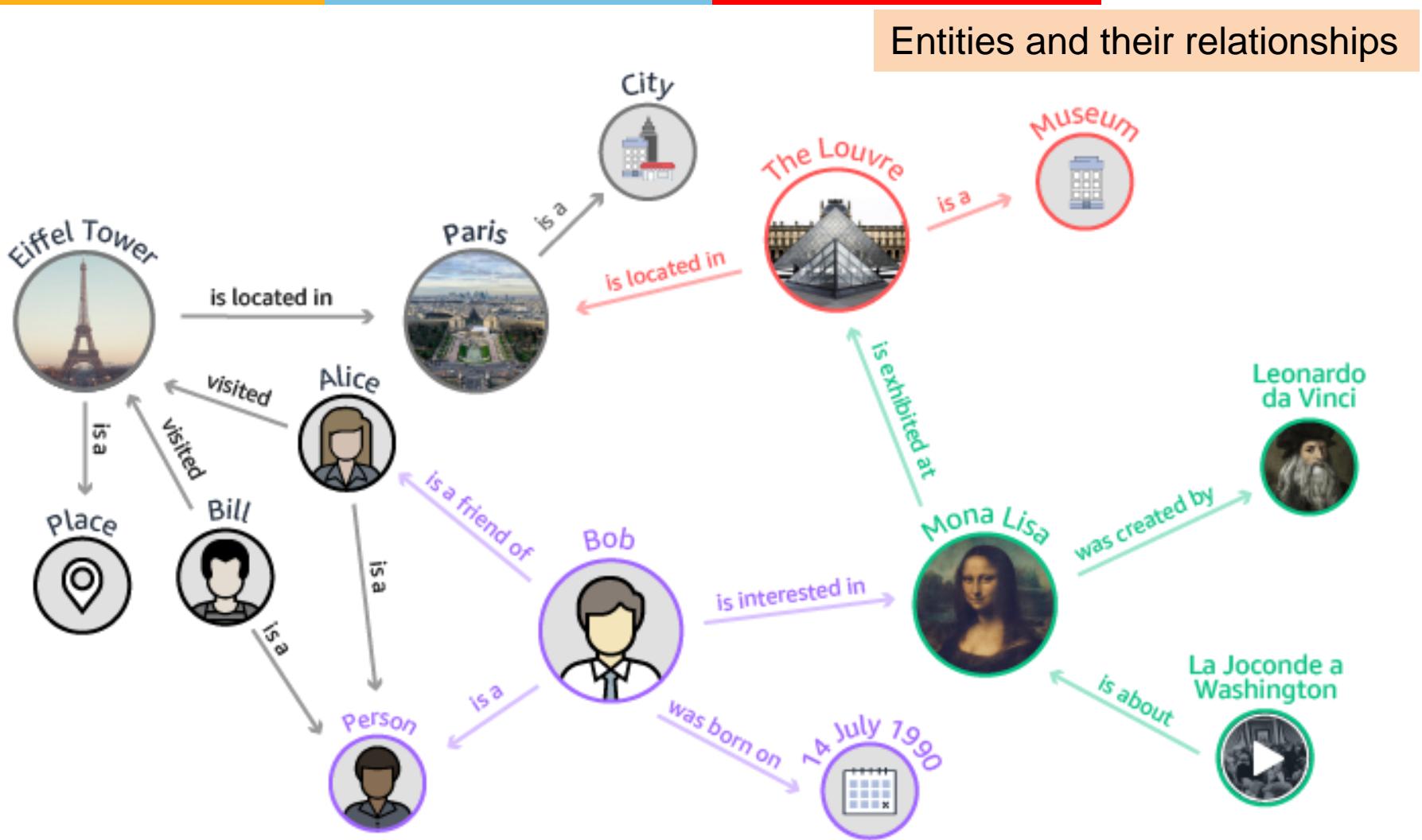
Record A: Jan, 2018, 100, 90, 110, ... // Usage – in customer order

Record B: Feb, 2018, 110, 92, 115, ...

Only one record 'Record A' is needed to calculate average usage in Jan 2018

Good for summarization

Graph database: Example: Knowledge graph



Graph database

Entities and relationships have properties (attributes)

Ex.

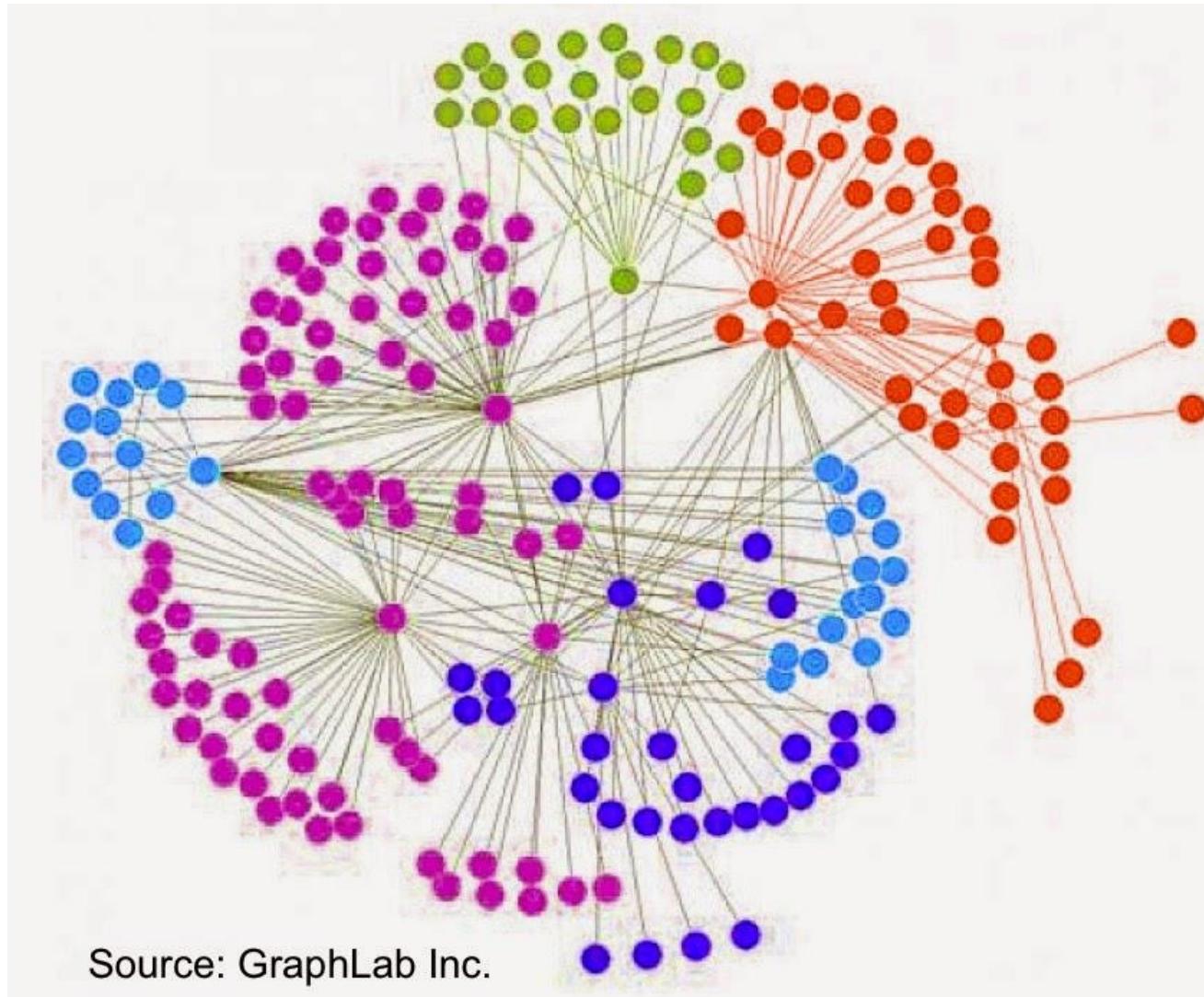
- Eifel tower properties can be height, date of construction
- Visited relationship can have properties such as date of visit

Uses

- Store a large amount of inter-related information and to search for an entity along with its relationships
- Fraud detection
- Social networks

Source: <https://opensourceforu.com/2017/05/different-types-nosql-databases/>

Can be use to detect hidden patterns



Easy to understand clusters

Graph database

- Graph databases allow you to store entities and relationships between these entities.
- Entities are also known as nodes, which have properties. Think of a node as an instance of an object in the application.
- Relations are known as edges that can have properties. Edges have directional significance; nodes are organized by relationships which allow you to find interesting patterns between the nodes.
- There are many graph databases available, such as [Neo4J](#), [Infinite Graph](#), [OrientDB](#), or [FlockDB](#)

In-Memory databases

- An **in-memory database (IMDB; also main memory database system or MMDB or memory resident database)** is a database management system that primarily relies on main memory
- Applications where response time is critical, such as those running telecommunications network equipment and mobile advertising networks, often use main-memory databases
- With the introduction of non-volatile random access memory technology (Flash memory), in-memory databases will be able to run at full speed and maintain data in the event of power failure
- Popular In-memory databases are **SAP's HANA**, IBM DB2 BLU, Oracle
- **These databases support OLTP and OLAP (Online Analytical Processing)**

Acknowledgement

Sources

Hadoop

<https://www.mssqltips.com/sqlservertutorial/77/dattatreysindol/>

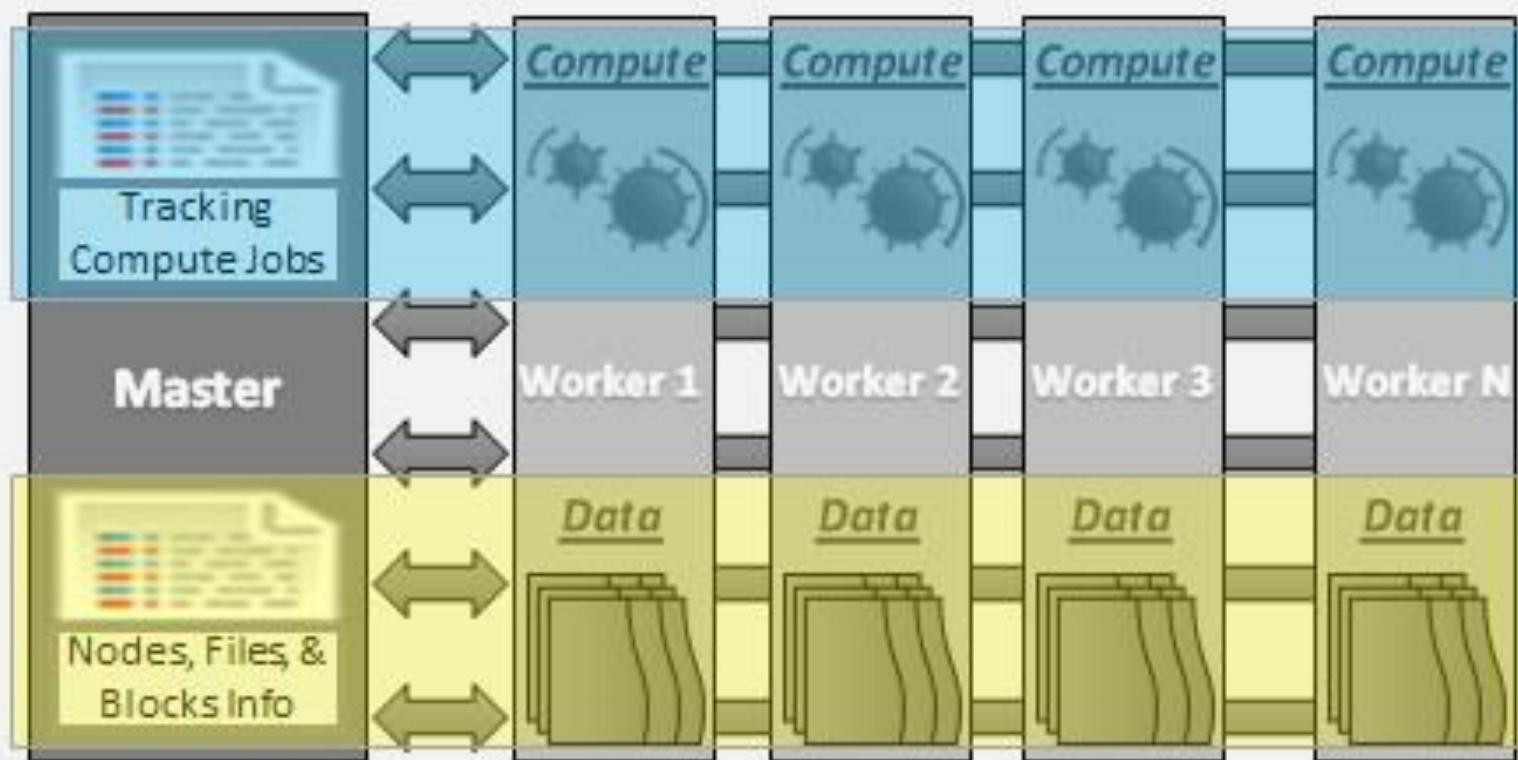
NoSQL Database

<https://www.thoughtworks.com/insights/blog/nosql-databases-overview>

Appendix

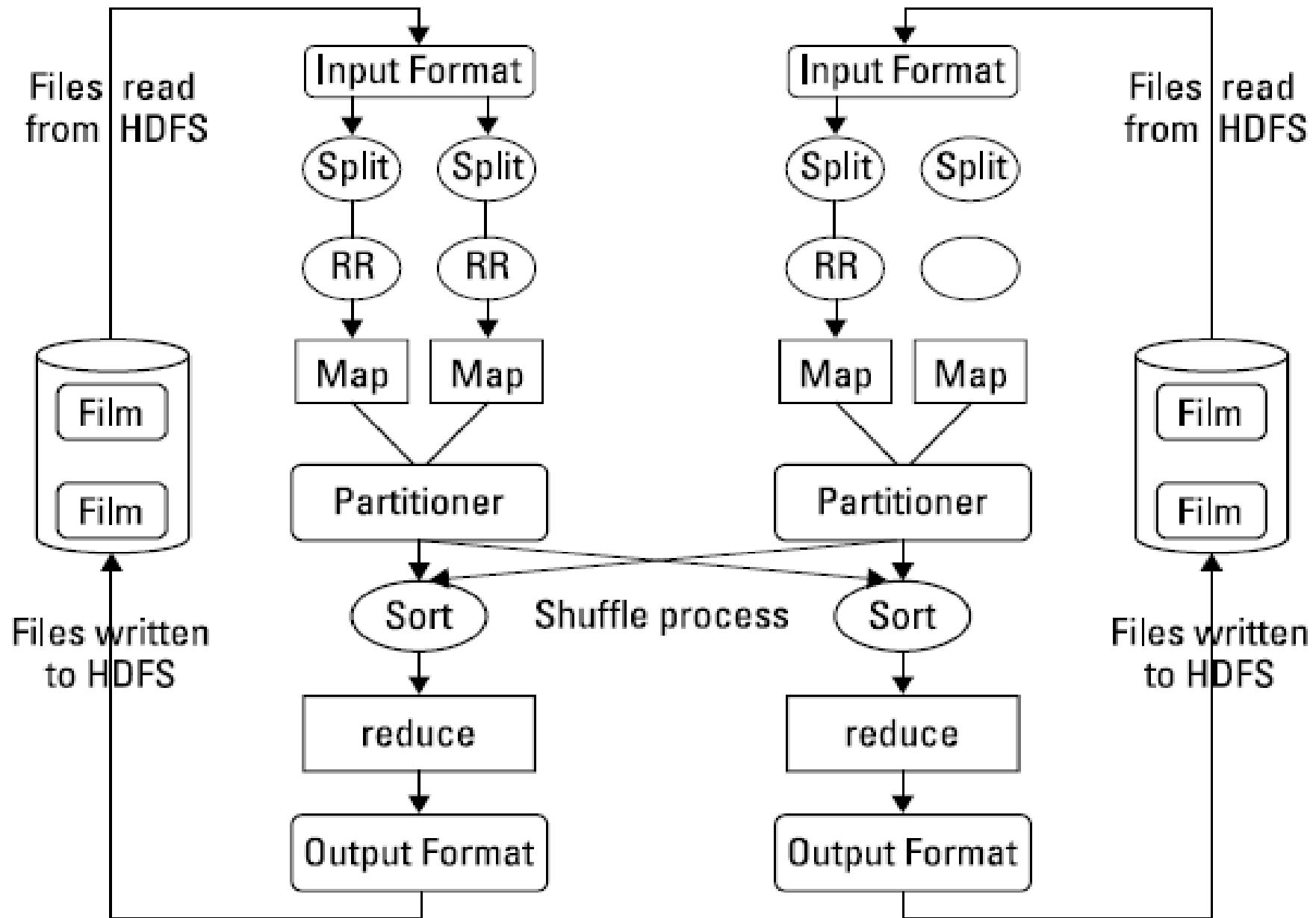
Typical Architecture of Hadoop

Across Same or Different Rack, Data Center, or Region

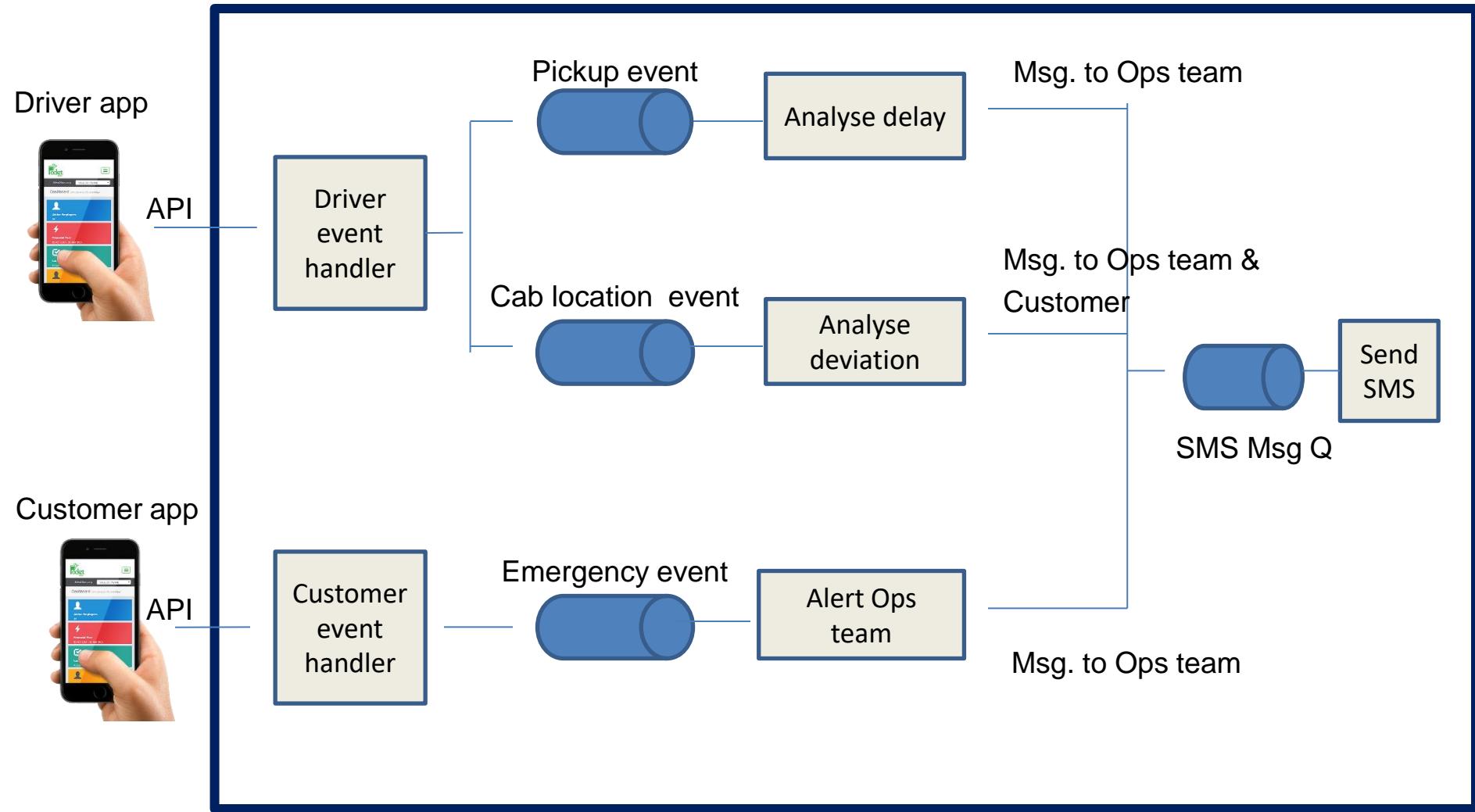


MapReduce

HDFS



Example of Stream processing in a cab hailing company like Ola / Uber



Analytics

Data Visualization

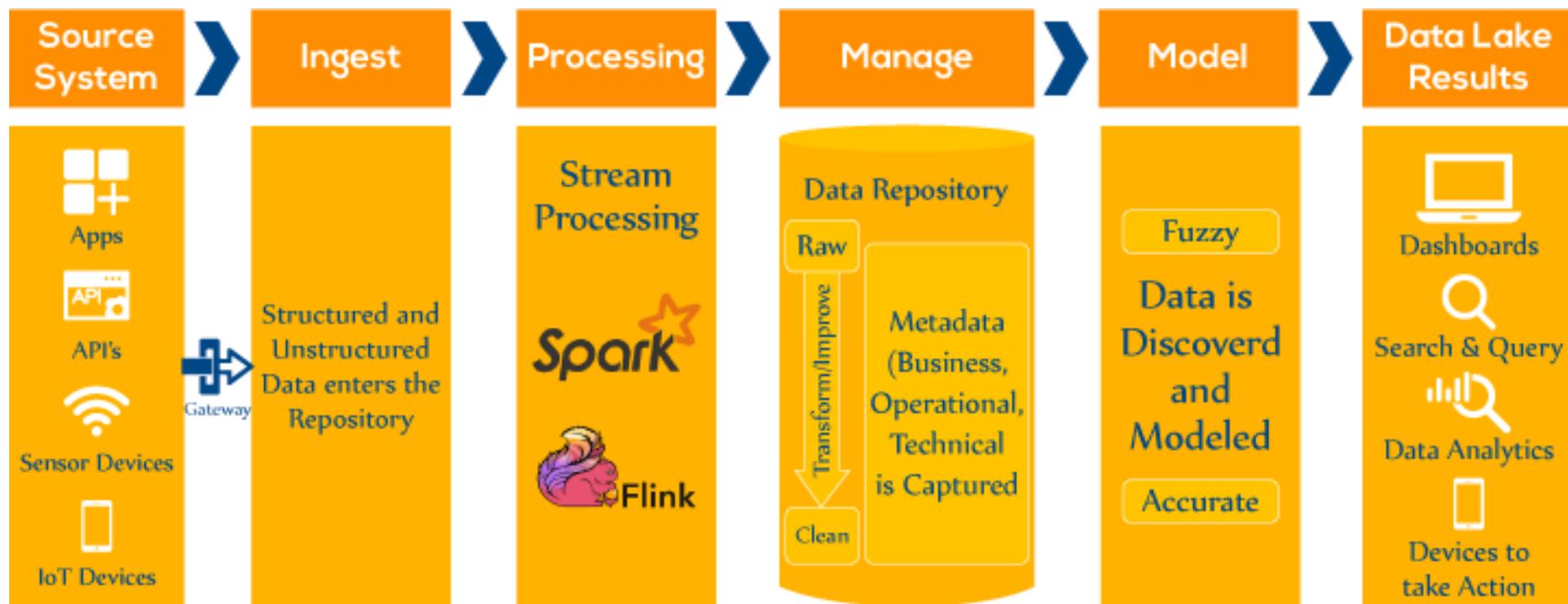
Multi-dimensional data

Data mining

Examples...

Tools

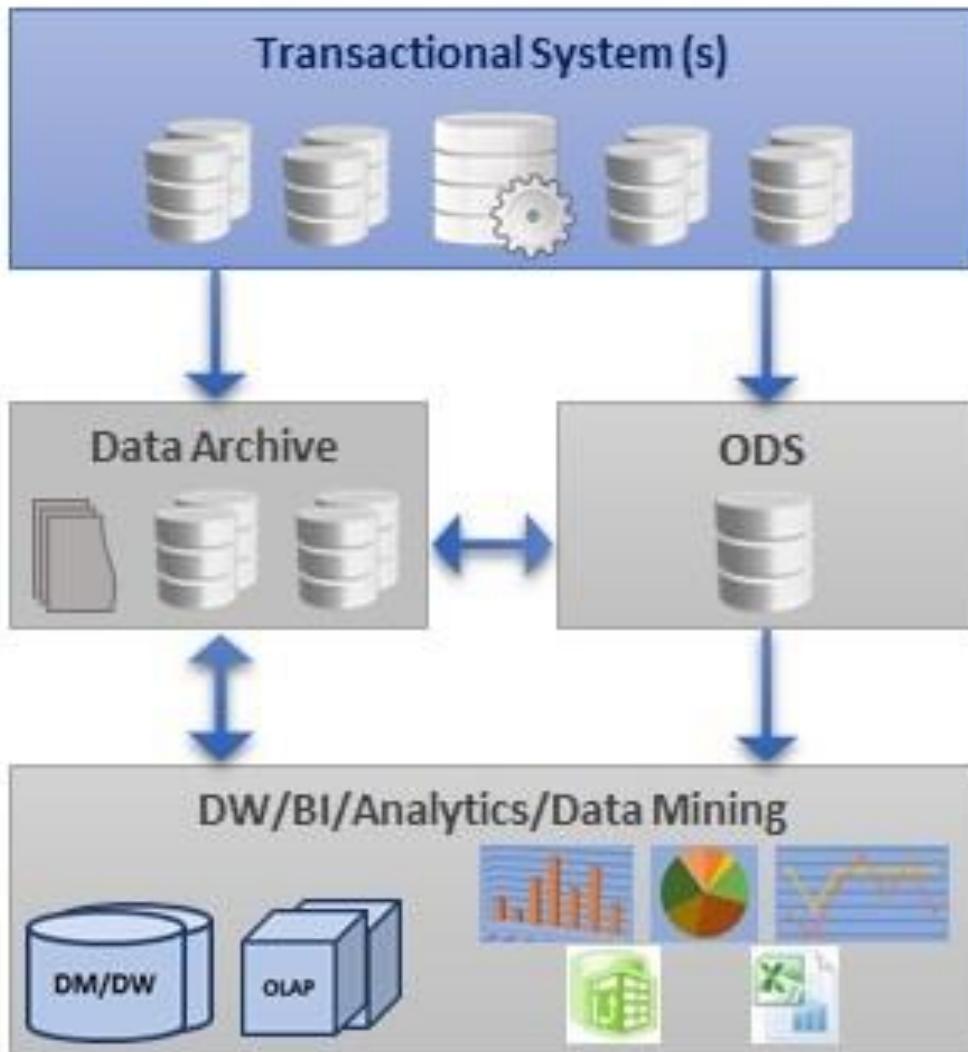
Real-Time Streaming and Data Analytics For IoT



Source: <https://www.xenonstack.com/blog/big-data-engineering/real-time-streaming-analytics-tools/>

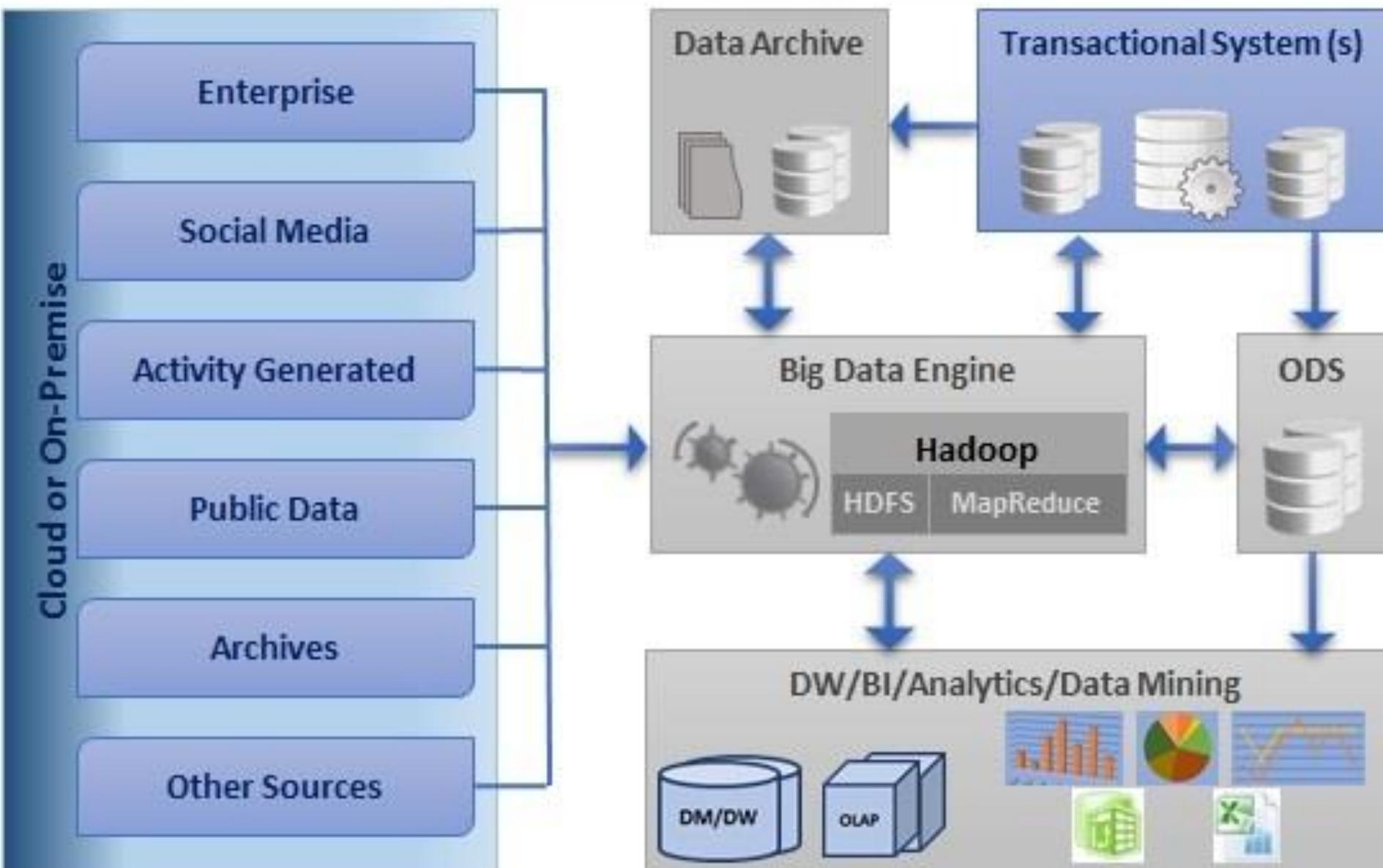
Traditional Data Processing

Traditional Data Processing & Management



ODS: Operational Data Store

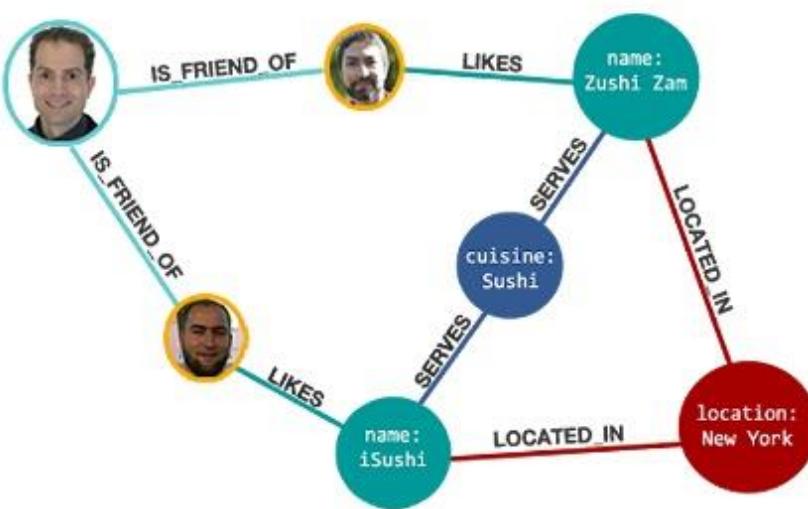
Modern (Next Generation) Data Processing & Management



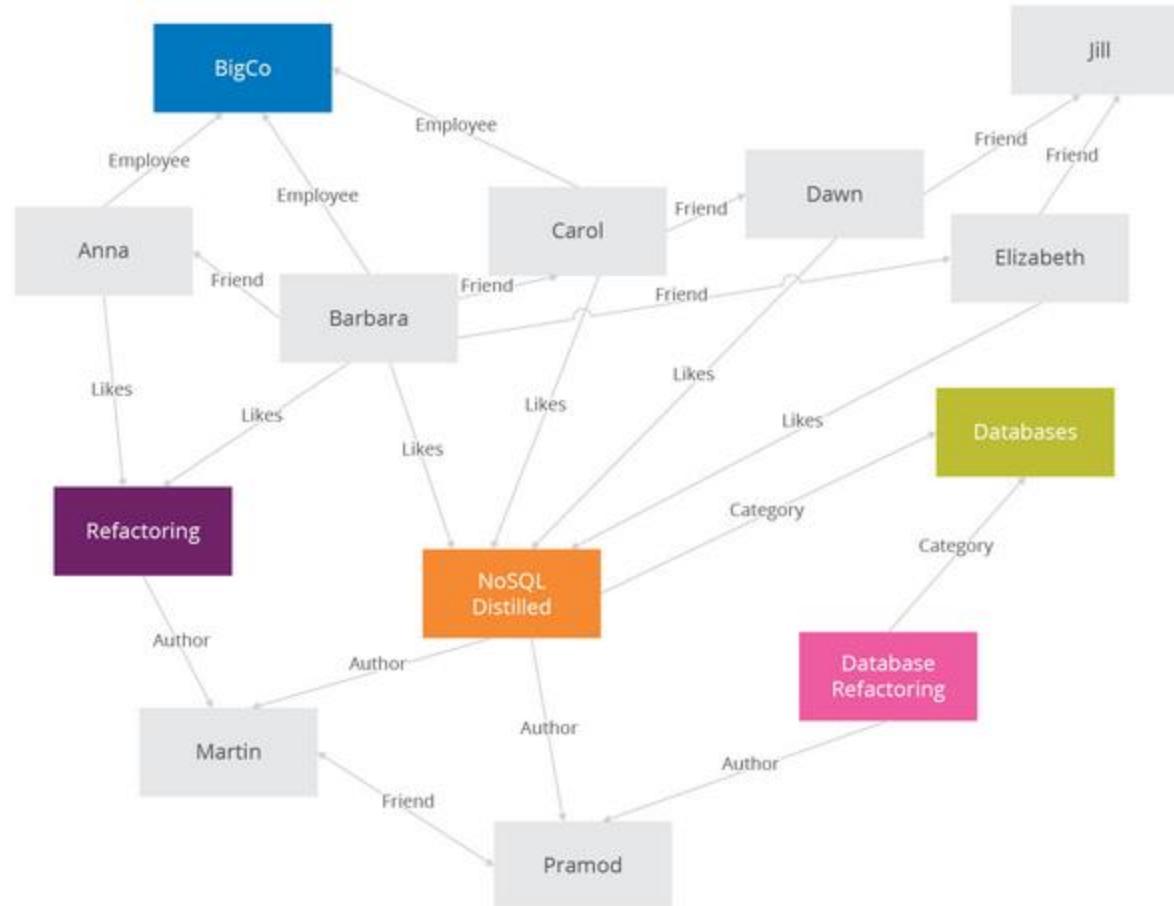
Use cases of Stream Processing

Following are some of the use cases.

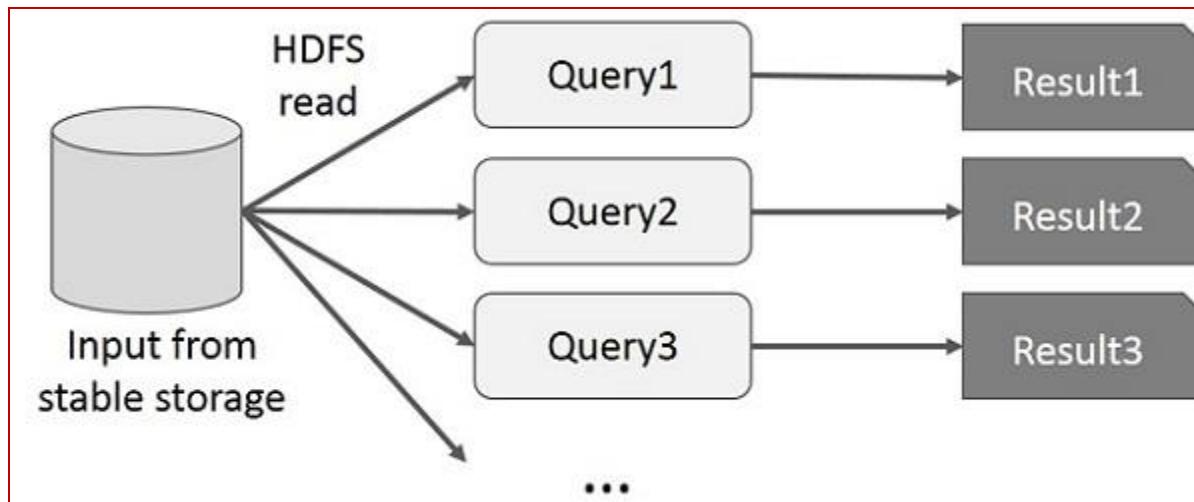
- Algorithmic Trading, [Stock Market Surveillance](#),
- Smart Patient Care
- [Monitoring a production line](#)
- Supply chain optimizations
- Intrusion, Surveillance and Fraud Detection (e.g. [Uber](#))
- Most Smart Device Applications : Smart Car, Smart Home ..
- [Smart Grid](#)—(e.g. load prediction and outlier plug detection see [Smart grids, 4 Billion events, throughout in range of 100Ks](#))
- Traffic Monitoring, Geo fencing, Vehicle and Wildlife tracking—e.g. [TFL London Transport Management System](#)
- Sport analytics—Augment Sports with realtime analytics (e.g. this is a work we did with a real football game (e.g. [Overlaying realtime analytics on Football Broadcasts](#))
- Context-aware promotions and advertising
- Computer system and network monitoring
- Predictive Maintenance, (e.g. [Machine Learning Techniques for Predictive Maintenance](#))
- Geospatial data processing



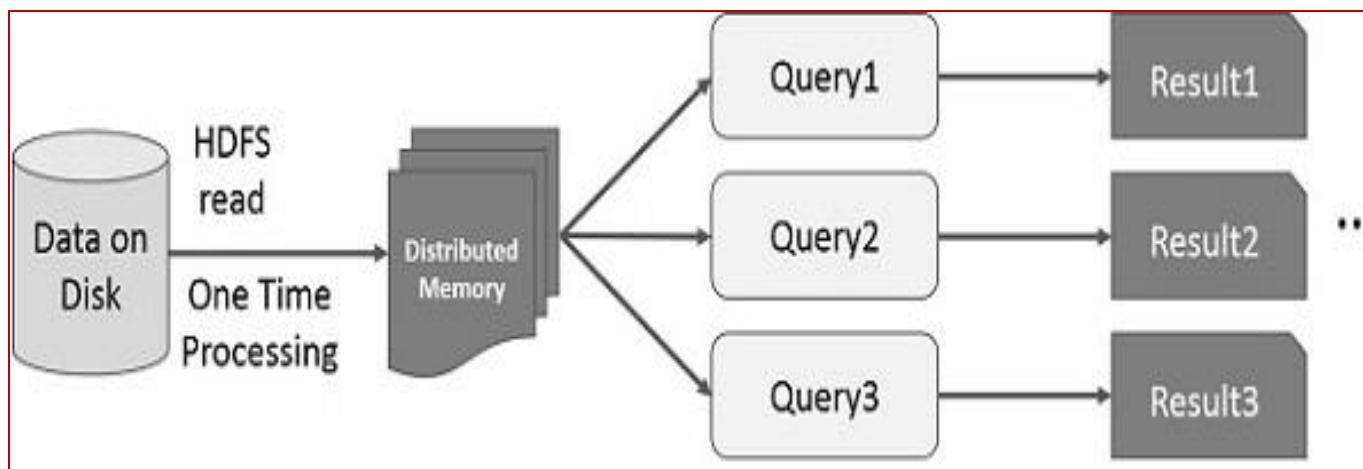
Graph database



Difference between Hadoop & Spark...



Interactive operation using Hadoop



Interactive operation using Spark

Source:
https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm

Real-time analytics

Detecting bank fraud requires real-time analytics as events happen

Such situations demand processing of each event as they happen rather processing a batch of data on disk

This led to tools such as Spark Streams and Storm which support in-memory processing, than disk based processing

Storm for real time computing

Apache Storm is a free and open source **distributed real time computation system**.

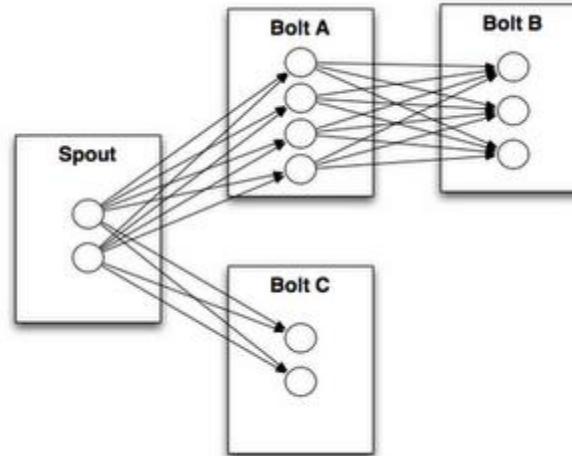
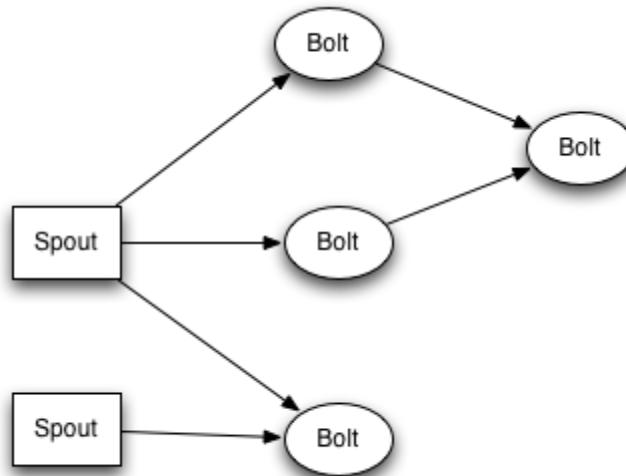
Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing. Storm is simple, can be used with any programming language, and is a lot of fun to use!

Storm has many **use cases: real time analytics, online machine learning, continuous computation**, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.

Storm **integrates with the queueing and database technologies** you already use. A Storm topology consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed. Read more in the tutorial.

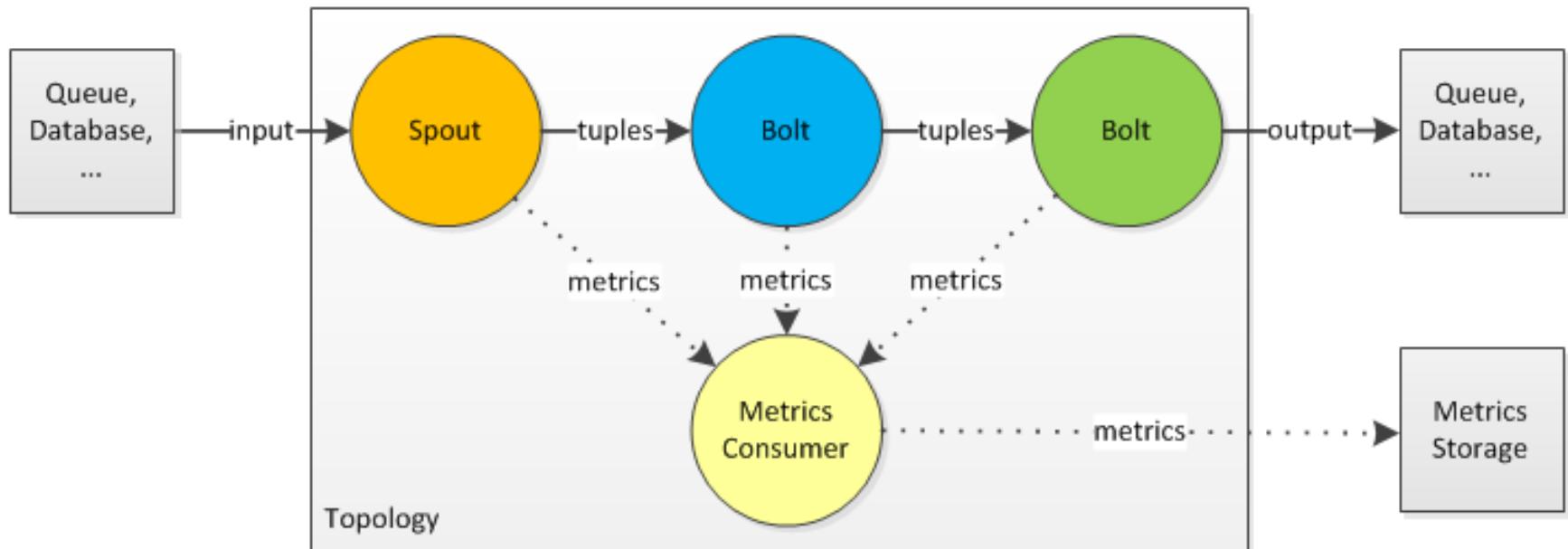
Source: <http://storm.apache.org/>

Storm topology



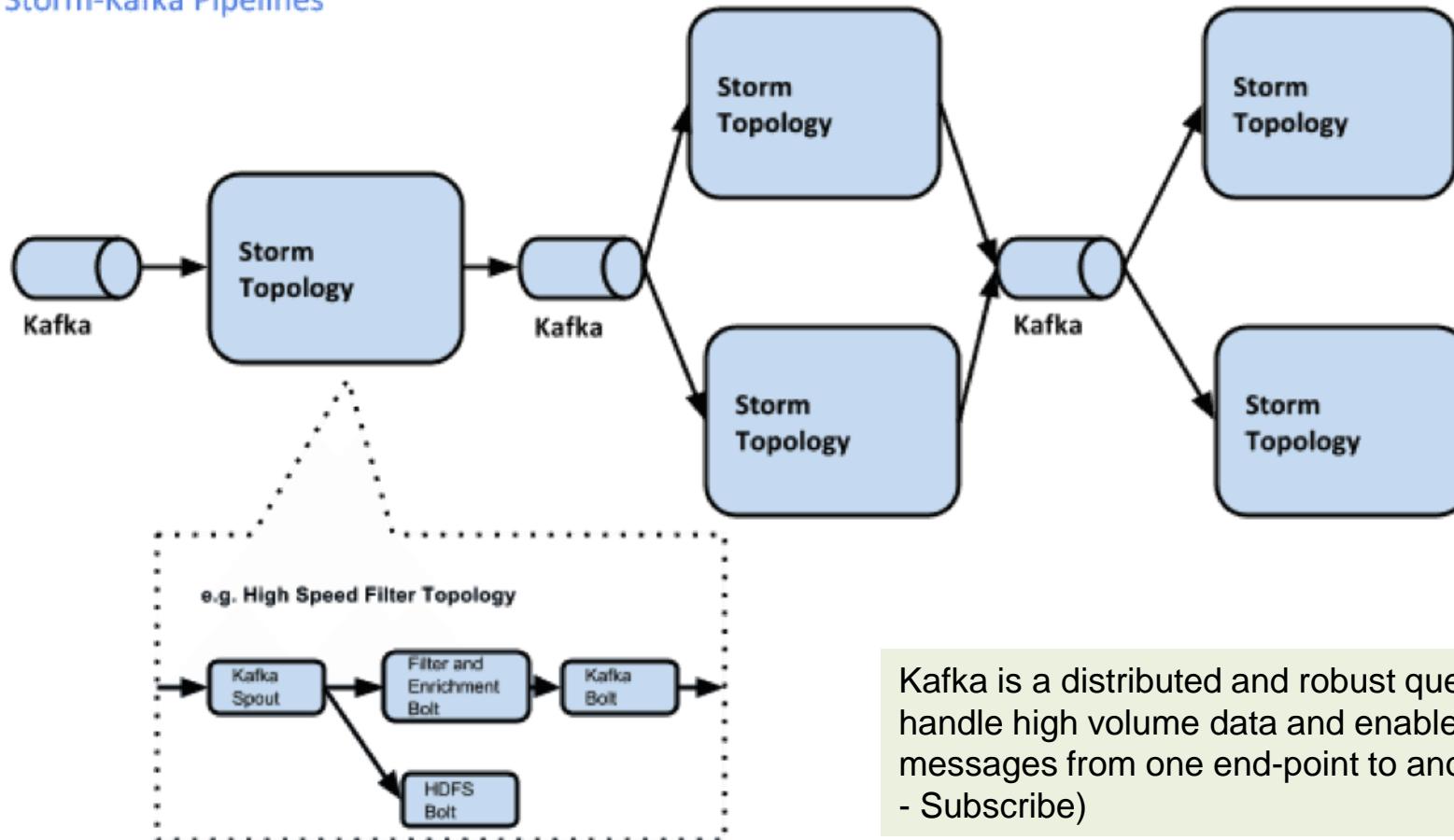
Twitter uses Storm for real-time analytics, personalization, search, revenue optimization
 Groupon uses Storm for Real-time data integration systems
 Yahoo! Uses Storm for processing user events, content feeds, and application logs.

Architecture of Storm system



Combining Kafka and Storm for real time computing

Storm-Kafka Pipelines



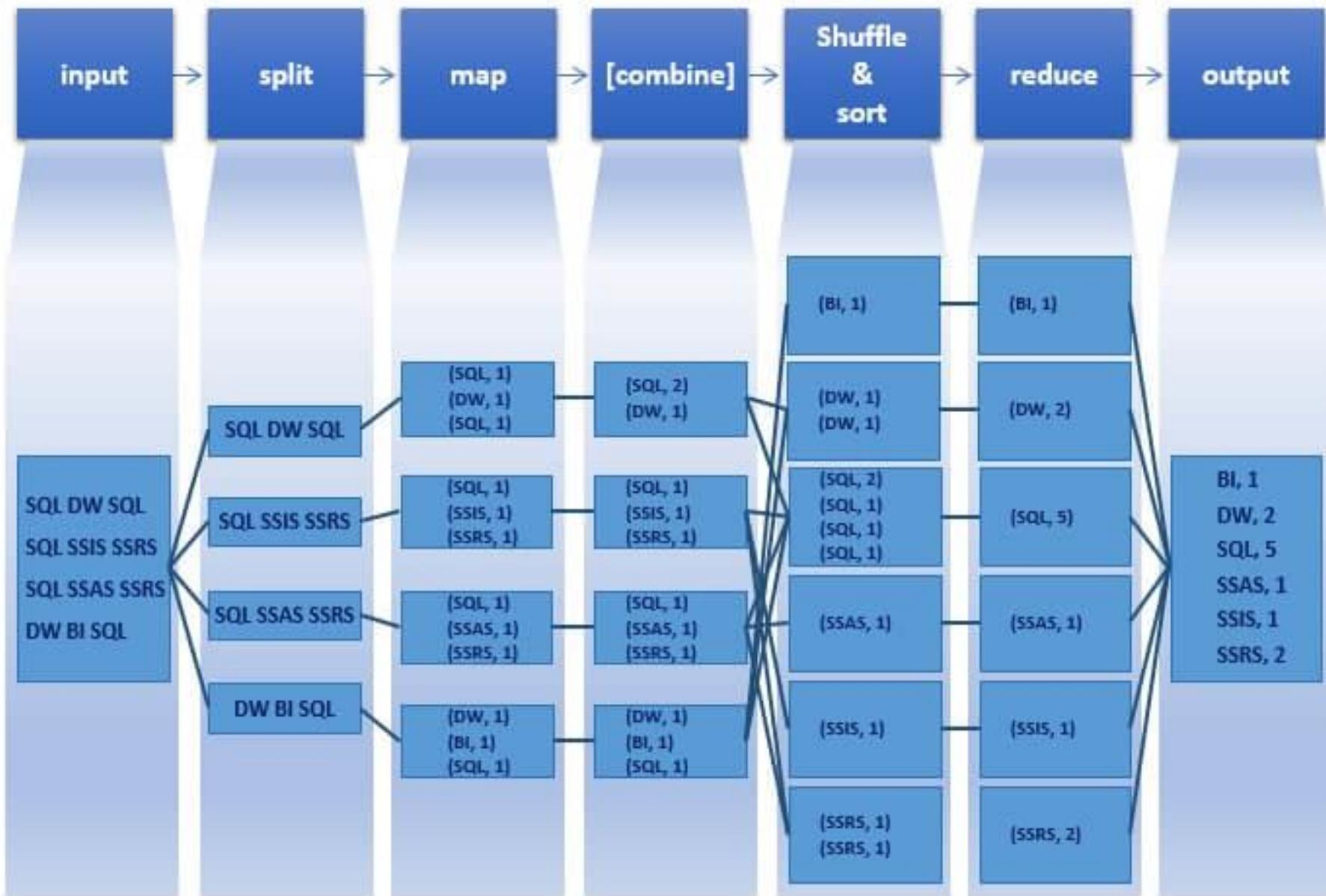
Source: <https://hortonworks.com/blog/storm-kafka-together-real-time-data-refinery/>

Storm & Kafka

The common flow of these tools (as I know it) goes as follows:

real-time-system --> Kafka --> Storm --> NoSql --> BI(optional)

MapReduce – Word Count Example Flow





Module 9 Part 5

Security Technology & Tools

BITS Pilani

Harvinder S Jabbal
SSZG653 Software Architectures

Contents

- Transport Layer Security (TLS)
 - OpenID & OpenAuth
 - LDAP
 - Identity & Access management
 - Firewalls
-

Introduction

The objective of this session is to provide an introduction to a few important technology topics.

Transport layer security (TLS)

(Older version is SSL)



- Used for secure communication between **client & server** (example between browser and a web site)
 - It provides
 - **Privacy:** No intruder can know what communication is going on
 - **Data integrity:** Data being communicated can not be modified by an intruder. If he does, it can be detected
-

TLS: How does it work?

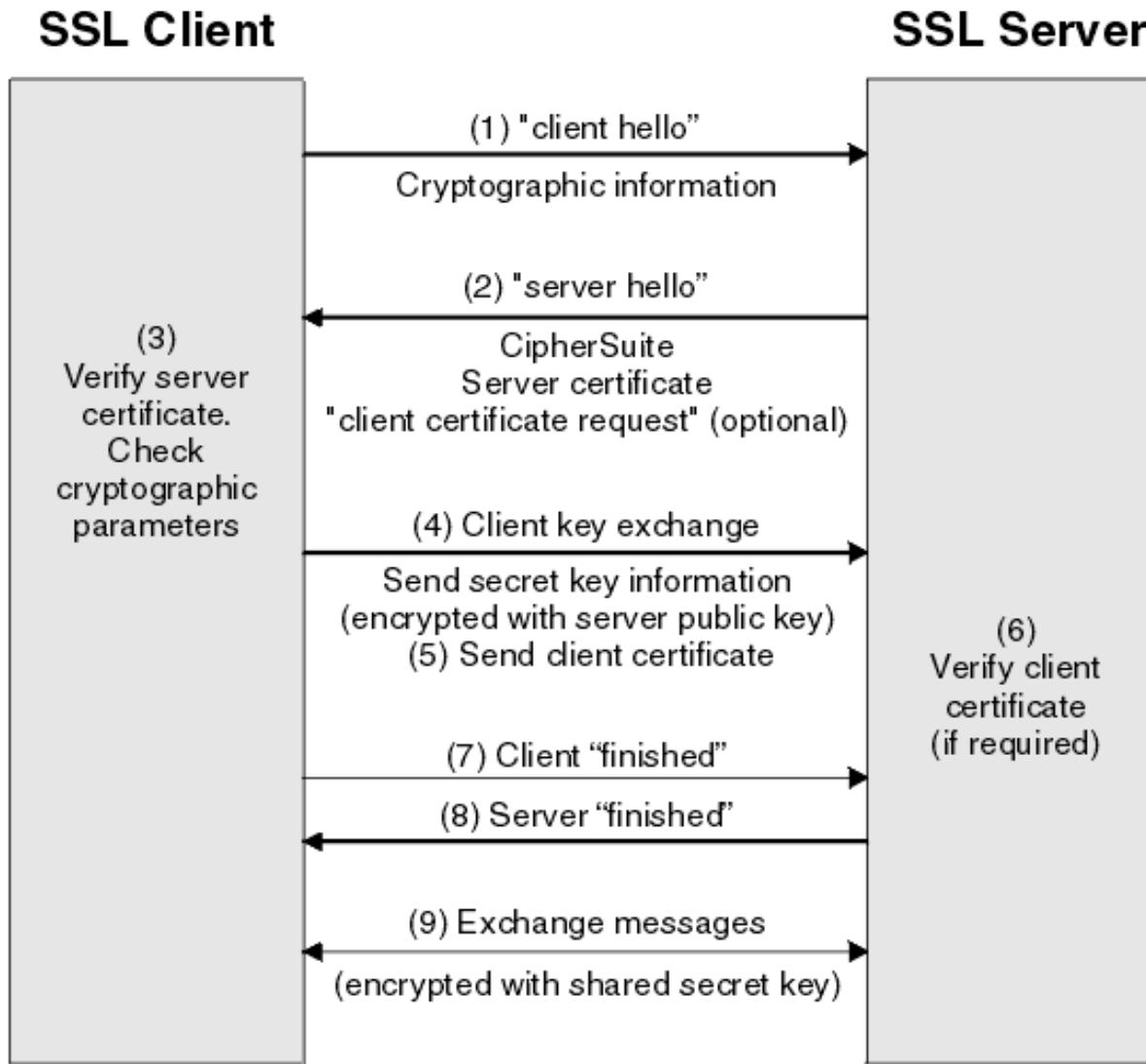
Client & server do the following:

- Agree on the version of the TLS protocol to use.
- Select cryptographic algorithms to use
- Authenticate each other by validating digital certificates.
- Generate a shared secret key, for the symmetric encryption of messages (This is faster than asymmetric encryption)

Encryption algorithms used

- Key Exchange Algorithms (RSA, DH, ECDH, DHE, ECDHE, PSK)
 - Authentication/Digital Signature Algorithm (RSA, ECDSA, DSA)
 - Bulk Encryption Algorithms (AES, CHACHA20, Camellia, ARIA)
 - Message Authentication Code Algorithms (SHA-256, POLY1305)
-

TLS / SSL steps



OpenID

- We use several websites.
- One issue we face is, remembering user ids & passwords of several websites
- With **OpenID technology**, we can use a single account, such as Facebook, Google or Yahoo, to sign-in to thousands of websites

Sample login page: American Cancer Society



American Cancer Society®

THE OFFICIAL SPONSOR OF BIRTHDAYS®

Welcome
Sign In | Register | My ACS

Español
Asian Language Materials
1-800-227-2345

DONATE »

HOME LEARN ABOUT CANCER STAY HEALTHY FIND SUPPORT & TREATMENT EXPLORE RESEARCH GET INVOLVED IN YOUR AREA

SIGN IN TO THE AMERICAN CANCER SOCIETY

Sign in using your account with:

- ACS Account
- Google**
- Yahoo
- Facebook
- Windows Live ID
- AOL
- OpenID

Sign In With Your Google Account

Registering and signing in allows you to interact with your American Cancer Society the way you want to. Automatically receive the cancer information you're interested in, connect with events and resources in your area, and customize your site to save relevant articles. You can even use an ID you may already have - including Facebook, Google, Yahoo, and more.

SIGN IN

HOW CAN WE HELP YOU?

Enter search terms or ask a question

You can call us any time

THE OFFICIAL SPONSOR OF BIRTHDAYS®

American Cancer Society®

Sample login page: Kodak Tips and project Exchange



Kodak

United States All Kodak Products & Services

KODAK Store KODAK Gallery Experience Kodak Tips & Projects Center Organize Print & Share Help Center Search Login / Register

tips & projects exchange

Exchange Home Photo Projects Learn Forums People FAQ

Inspire and Be Inspired

Share Your Back to School Projects with the community

Close X

Login

Login to the Kodak Tips & Projects Exchange using your Kodak account

Email Address:

OR

Password:

Forgot your password?

Sign In

Learn P

Spotlight

Photograph

Photography Techniques

Top 10 Lists

Do More with Your Photos

Scrapbooking Central

Create a Kodak account

★★★★★ 2 Ratings

★★★★★ 0 Ratings

Picture of the Day

Follow Us Online

Sign in using your account with:

Google

YAHOO!

Facebook

twitter

myspace

WindowsLive ID

OpenID: How does it work?

(One typical approach)



- The website redirects the user to an OpenID provider such as Facebook or Google
- The OpenID provider authenticates the user
- The user is redirected back to the website along with the end-user's credentials (such as user id, but not the password)

Reference: <https://en.wikipedia.org/wiki/OpenID>

OAuth

OpenID is to authenticate users.

OAuth authorizes a client to access your data stored in another website such as Yahoo (data such as your profile, contacts in Yahoo)

Reference: <https://tools.ietf.org/html/draft-ietf-oauth-use-cases-01#section-2.1>

OAuth: Use cases

Use case 1:

- An application can *use OAuth* to obtain permission from users to store files in their Google Drives.

Use case 2

- A photo printing application (www.printphotos.example.com) can access your photographs stored on a server www.storephotos.example.com and then print them

OAuth: How does it work?

- You try to log onto a website and it offers opportunities to log on using Google, Yahoo, etc. Let us say you choose Google.
- You are redirected to Google
- Google authenticates you, and returns a token to the website.
- The token enables the website to login to Google as you, and access resources such as your contacts.

<https://www.csoonline.com/article/3216404/what-is-oauth-how-the-open-authorization-framework-works.html>

LDAP: Lightweight Directory Access Protocol



Scenario suitable for LDAP

- Imagine you have a website that has a million registered users with thousands of page requests per second.
- By using LDAP, you can easily offload the user validation and gain significant performance improvement.
- Good use cases for LDAP:
 - You need to locate ONE piece of data many times and you want it fast
 - You don't update, add, or delete the data very often
 - The size of each data entry is small

LDAP

- LDAP is an Internet protocol to talk to a Directory service such as Active Directory of Microsoft
- Directory services store information in a tree structure
- LDAP is used in many open source solutions such as Docker, Kubernetes, Jenkins, etc.

Identity & Access management

- Identity and access management (IAM), is a framework for ensuring that **people in an enterprise have the appropriate access** to technology resources.
 - IAM solutions are typically used in large organizations to ensure regulatory compliance.
 - **Features of IAM**
 - Authentication of a user
 - Authorization to use applications
 - Definition of Roles. A User belonging to a group is authorized to perform certain operations defined for the role. Operations such as create sales order, approve credit card request, etc.
 - Delegation of permission to another user
 - Interchange identify information with trusted entities using OpenID, etc.
-

Leading Identity & Access management products



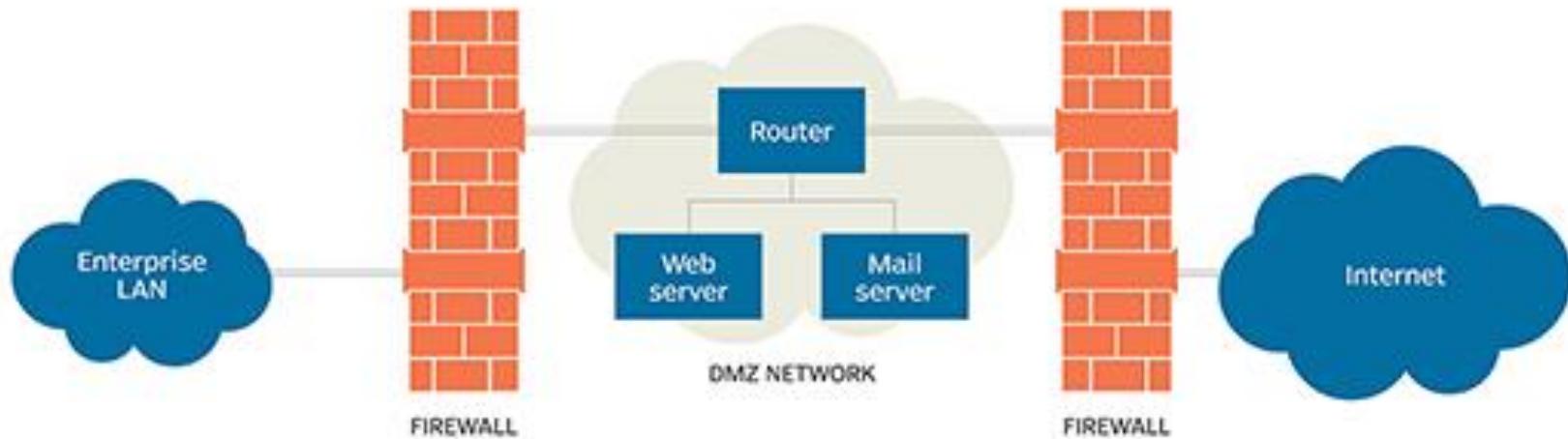
- Azure Active Directory
- IBM Security Identity and Access Assurance
- Oracle Identity Cloud Service
- Okta

Firewall

- A firewall is a network security device that **monitors incoming and outgoing network traffic** and decides whether to **allow or block** specific traffic based on a defined set of **security rules**.

De-Militarized Zones (DMZ)

DMZ network architecture



DMZ acts a buffer between Internet and organization network

How DMZs work?

- DMZs are intended to function as a sort of **buffer zone between the public internet and the organizational network**.
 - If a better-prepared threat actor is able to get through the first firewall, they must then gain unauthorized access to those services before they can do any damage
-

Some Firewall features

- Intrusion detection: Identify & block security threats such as malware, spyware, etc.
- Grant access to users based on business need
- Fingerprint applications and track their usage: # users, bandwidth usage
- Allocate bandwidth to applications (ex. Allocate more BW to SalesForce.com and less to YouTube)

https://www.cio.com.au/article/365101/top_seven_firewall_capabilities_effective_application_control/

<https://www.fortinet.com/products/next-generation-firewall.html#services>

<https://www.securedgenetworks.com/blog/11-Features-to-Look-for-in-Your-Next-Generation-Firewall>

Firewall techniques

Techniques used:

1. Packet filtering: Looks at IP address and Port # and drops packets coming from or destined to certain IP addresses
2. Circuit level gateways: Detect conversations by looking at end-point pairs
3. Application layer filtering: Detects applications trying to use disallowed protocols or ports
4. Hide addresses and perform network address translation.
 - Hacker can not know the IP address of the server that receives the message. Even if it knows, the firewall will block it.



Appendix

Business Process Management

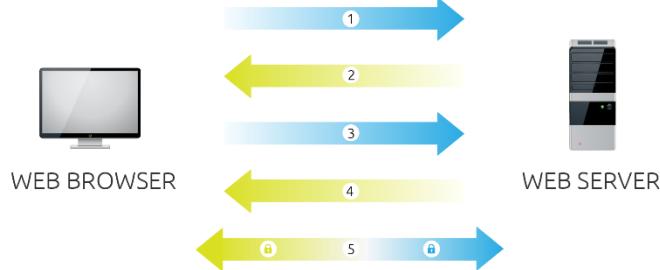


Business Process Management Tools: Business Process

Management (BPM) tools are used for automating, measuring and optimizing business processes.

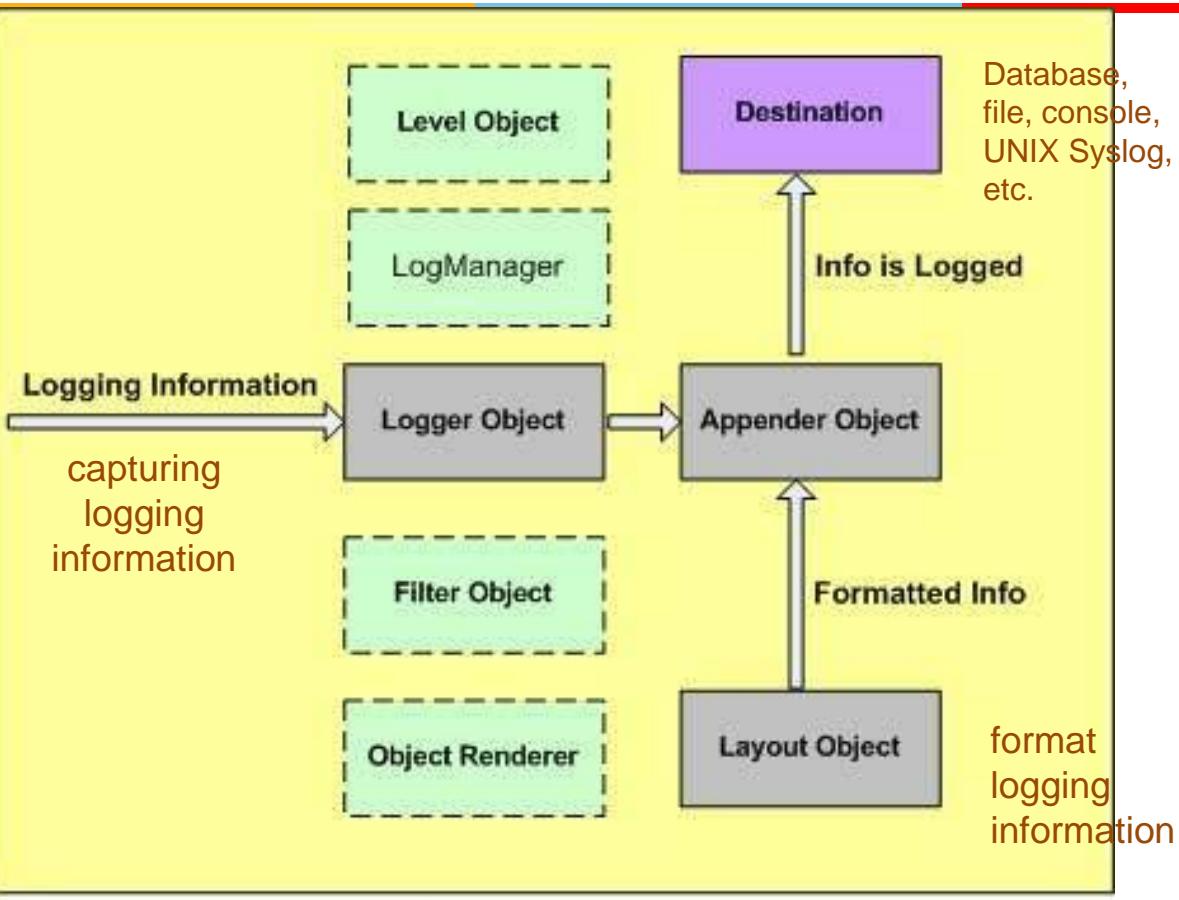
BPM tools use workflow and collaboration to provide meaningful metrics to business leaders

Example: Appian, Zoho



1. Browser connects to a web server (website) secured with SSL (https). **Browser requests that the server identify itself.**
2. Server sends a copy of its SSL Certificate, including the server's public key.
3. **Browser checks the certificate** root against a list of trusted CAs and that the certificate is unexpired, unrevoked, and that its common name is valid for the website that it is connecting to. If the browser trusts the certificate, it creates, encrypts, and **sends back a symmetric session key** using the server's public key.
4. Server decrypts the symmetric session key using its private key and **sends back an acknowledgement** encrypted with the session key to start the encrypted session.
5. **Server and Browser now encrypt all transmitted data with the session key.**

Logging: Apache Log4j

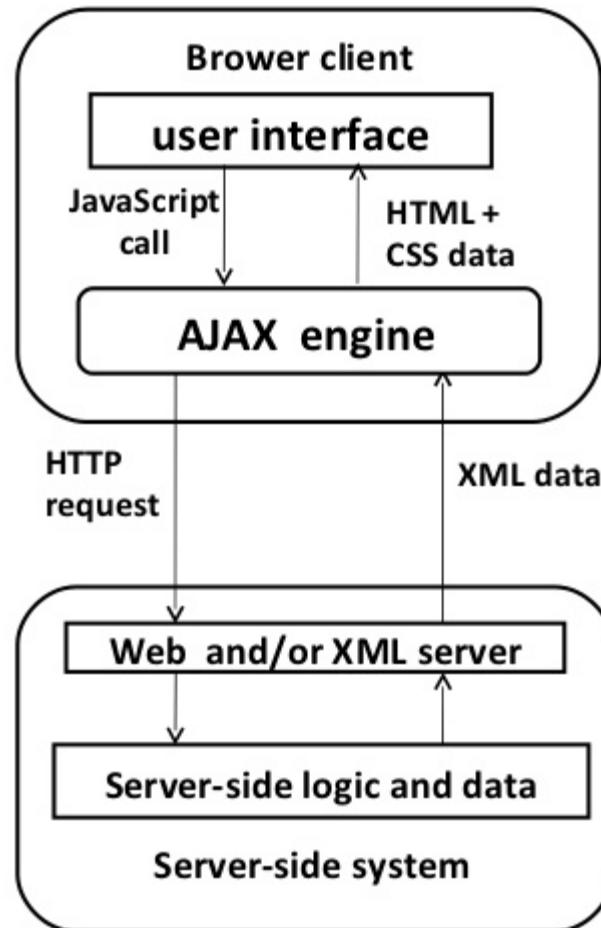


- Logging is an important component of the software development.
- A well-written logging code offers quick debugging, easy maintenance, and structured storage of an application's runtime information.
- Logging does have its drawbacks also. It can slow down an application.



Asynchronous operation

AJAX Architecture



AJAX stands for **A**synchronous **J**ava**S**cript and **X**ML.

AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

Some famous web applications that use AJAX:

Google Maps (Drag entire map)
Google Suggest (Google suggests as you type)

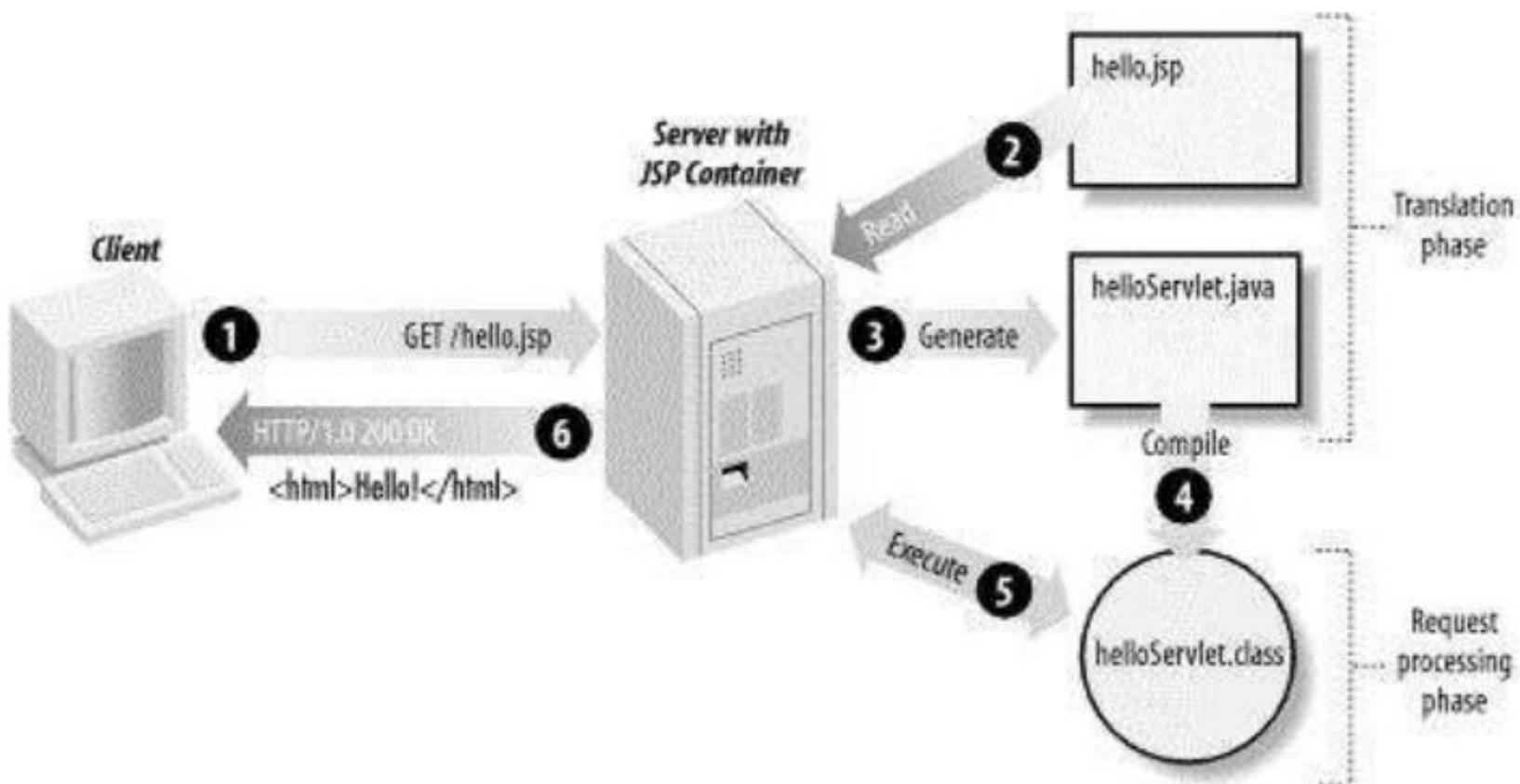
Simple Web application architecture

innovate

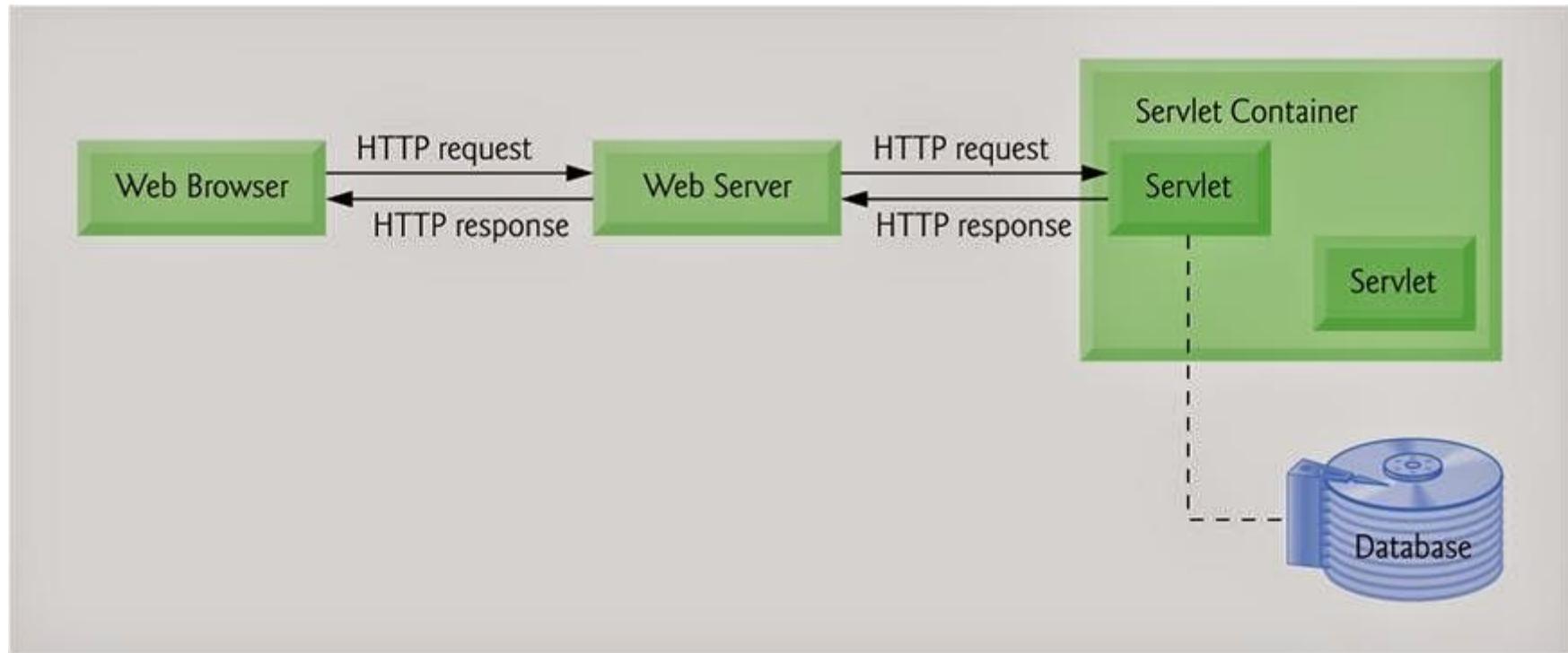
achieve

lead

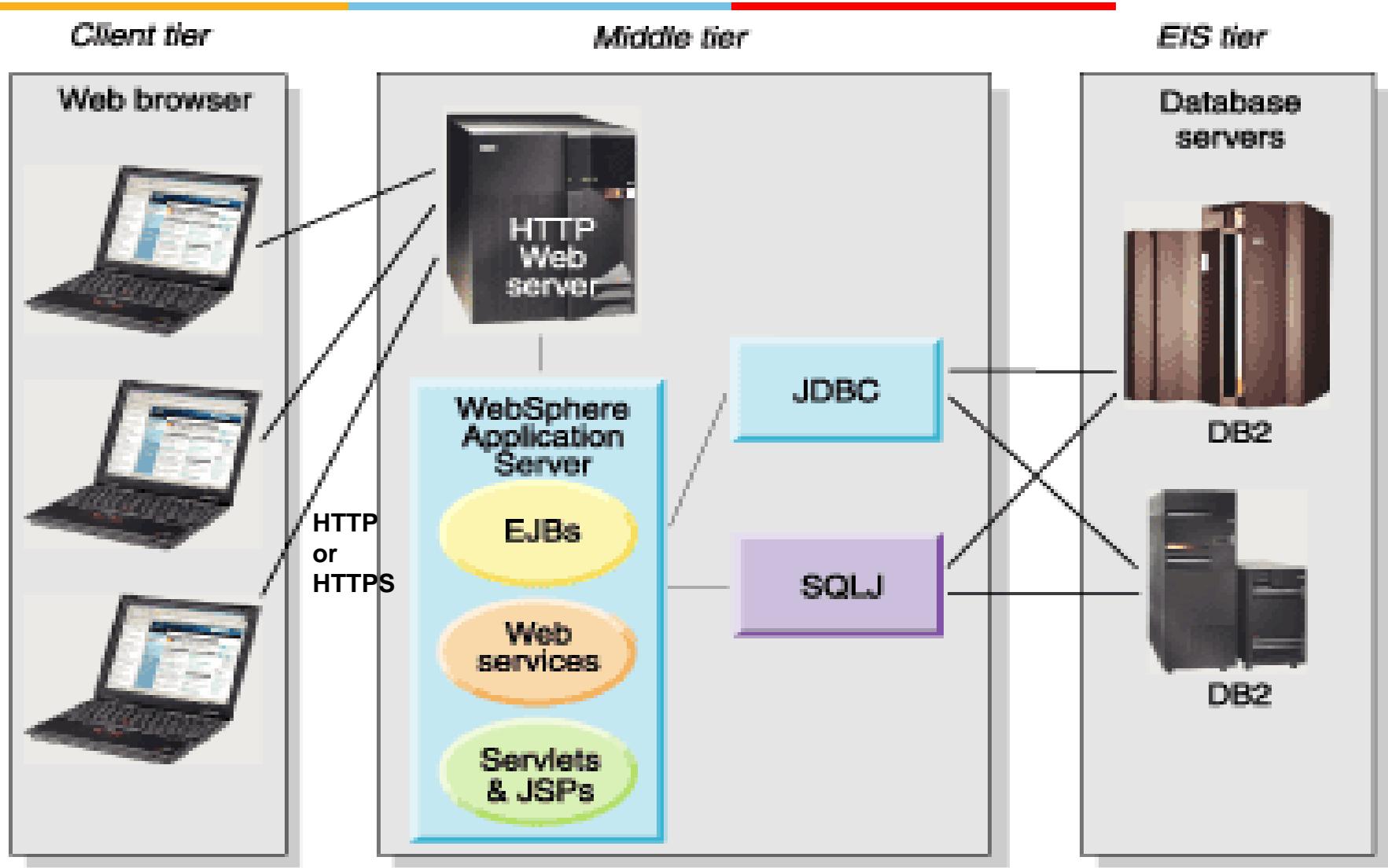
Dynamic web pages – using JSP and Servlet



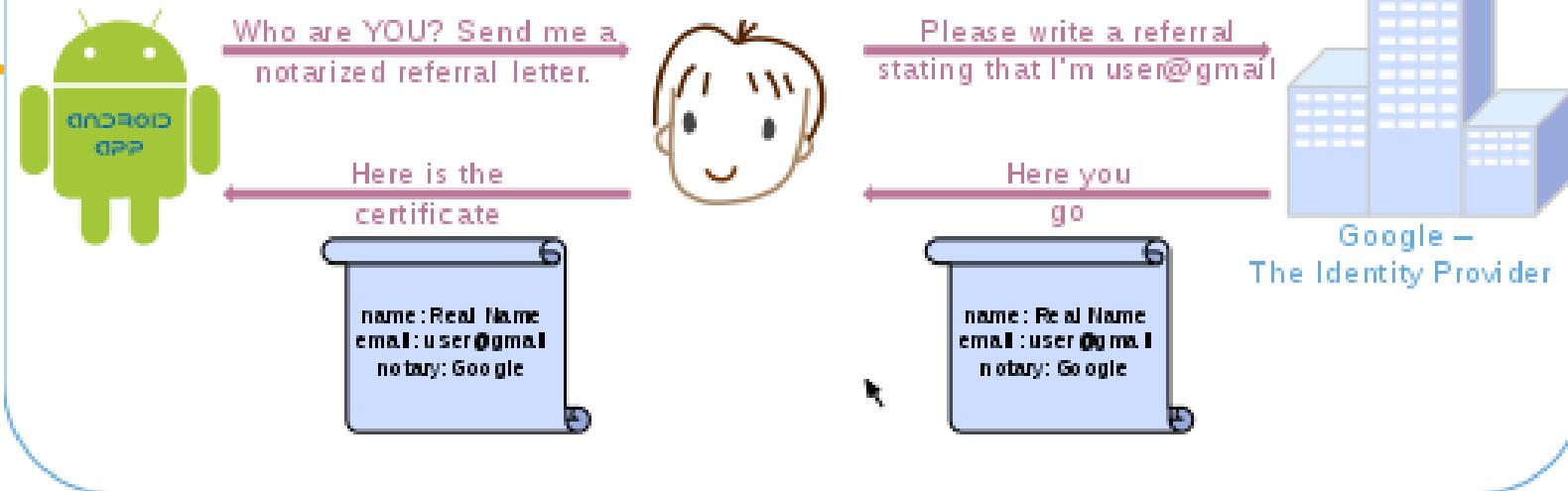
Dynamic web pages



Web application architecture



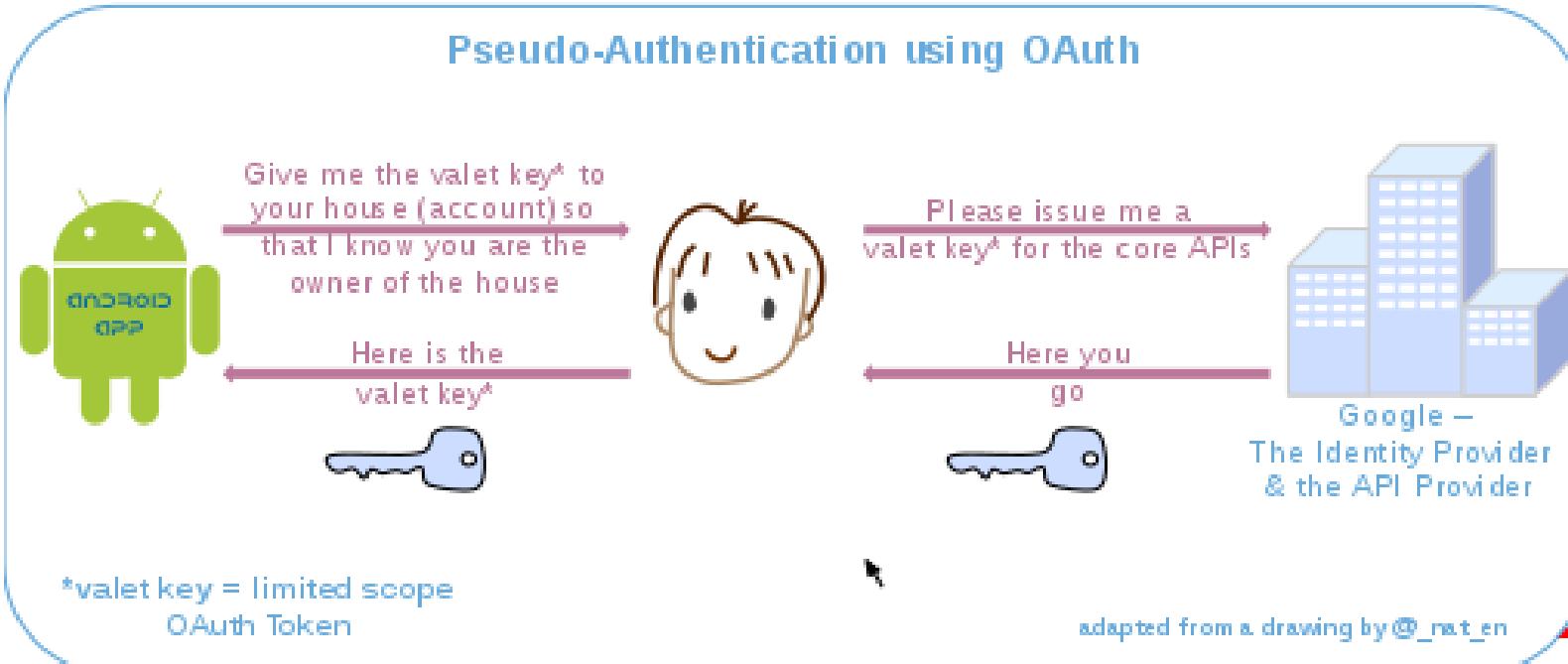
OpenID Authentication



Source:
Wikipedia

vs.

Pseudo-Authentication using OAuth





Module 9 Part 6

Machine Learning

BITS Pilani

Harvinder S Jabbal
SSZG653 Software Architectures

Contents

- Introduction
 - What is Machine Learning (ML)?
 - Applications of ML
- Different types of ML
 - Algorithms used in ML
- Steps to build a ML model
- Architecture of ML system
- Popular tools for ML

Introduction

What is ML?

ML is the ability of machines to **detect patterns** and perform activities that humans do:

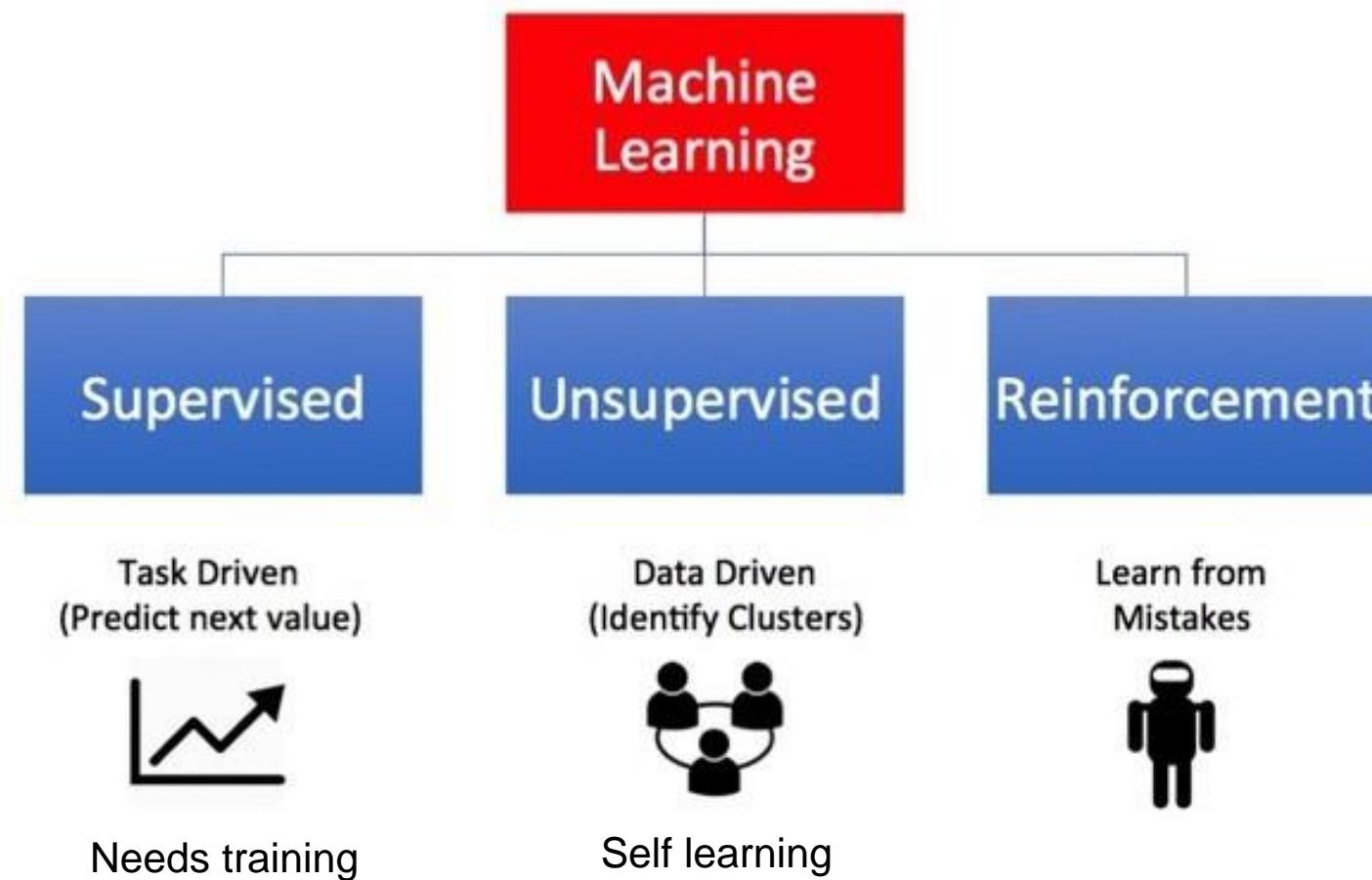
Applications

- Determine treatment for a patient based on their genetic makeup, demographic (age, gender, ethnicity) and psychographic characteristics (lifestyle, attitude, values)
- Pro-active maintenance of industrial equipment by looking at health parameters such as temperature, vibration, oil level
- Credit card Fraud detection by looking at spend frequency, amount, time of day, place of transaction, product purchased, customer profile

Major types of machine learning

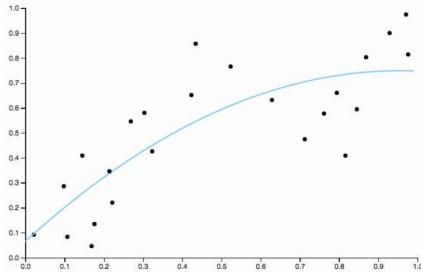


Types of Machine Learning



Some algorithms used in ML

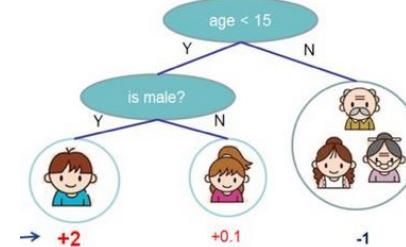
Regression



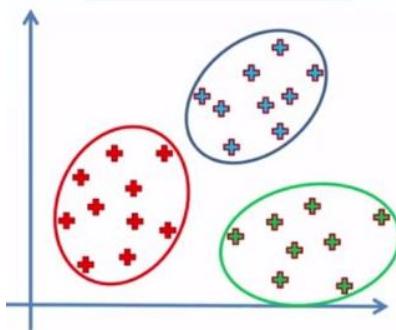
Eg. Predict price of house given size, location, etc.

Decision tree

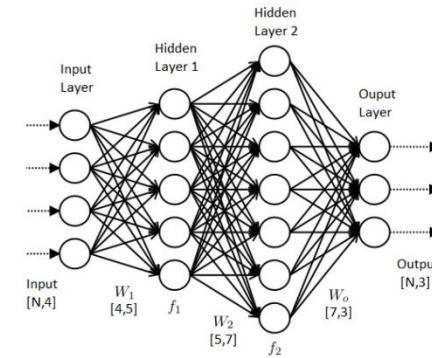
Does the person like computer games



Clustering



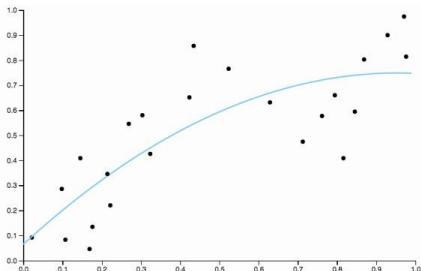
Neural networks



Supervised learning

- **Supervised:** In this approach, we provide a labelled dataset to the machine.
- Labelled dataset consists of features and result or class.
- Using the dataset, the machine builds a model (an equation or structure) to predict.

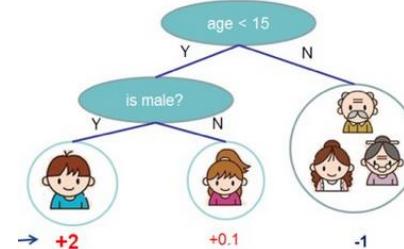
Regression



Eg. Predict price of house given size, location, etc.

Decision tree

Does the person like computer games



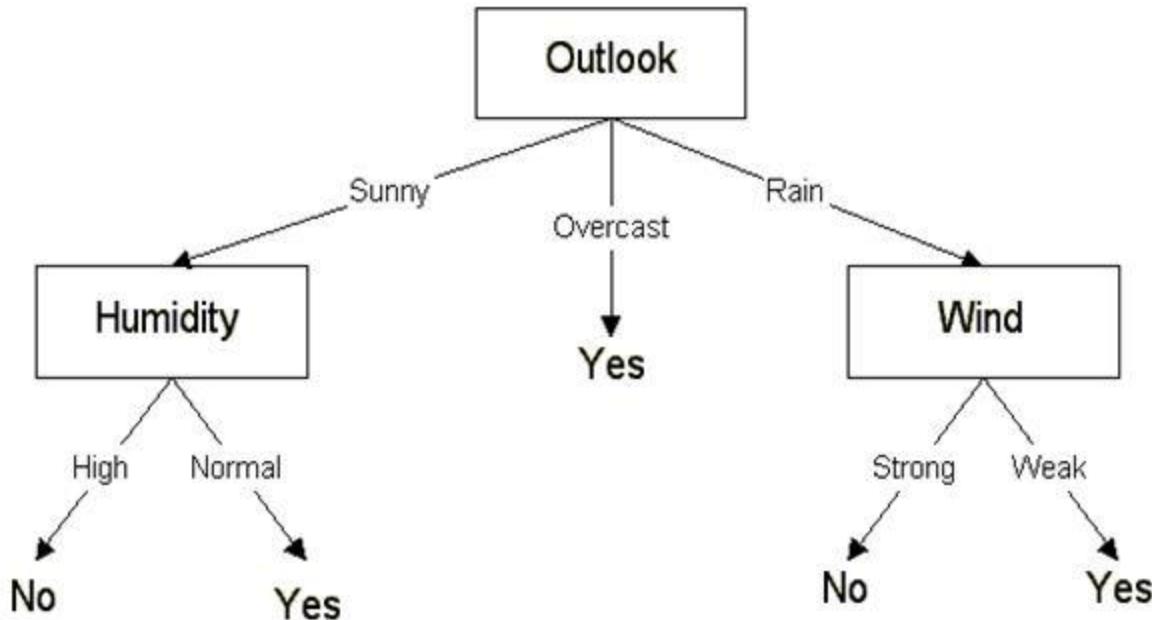
Eg. Does this person like computer game? (based on age and gender)

Decision tree: Data provided to algorithm



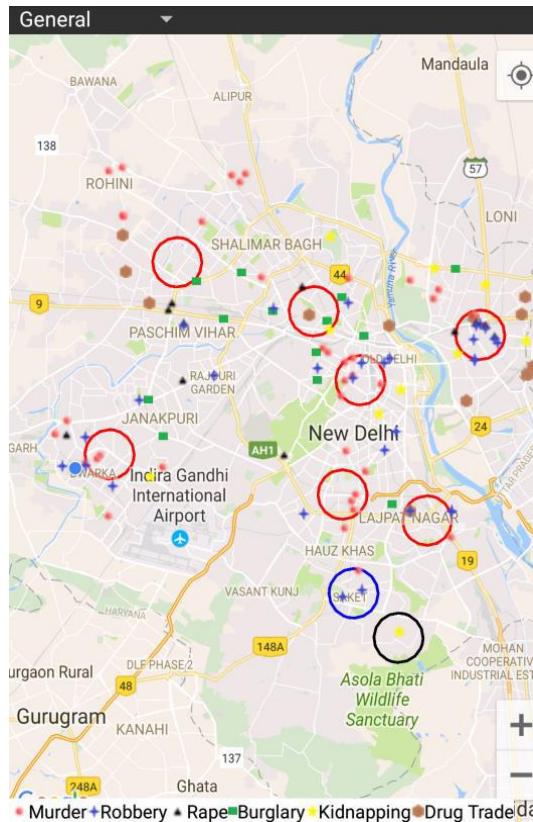
Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision tree built by algorithm to predict whether to play or not



Unsupervised Learning

- ***Unsupervised***: Here we have an unlabelled dataset. We do not know what all features will constitute a class. The machine learns by itself and builds a model.



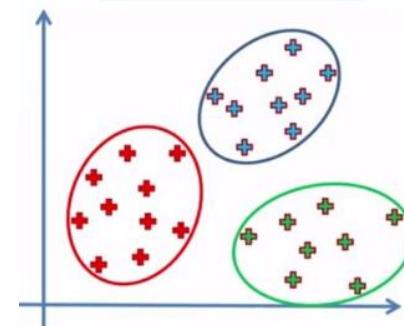
Circles indicates clusters of crime areas in Delhi

Ref: Crime Prediction using K-means Algorithm:
 Global Research and Development Journal for Engineering |
 Volume 2 | Issue 5 | April 2017

Clustering

- Customers can be segmented (clustered) based on Gender, age, annual income, products purchased, etc. (Luxury car buyers)
- We can source potential customer data and determine to which segment they belong to.
- Based on the segment, we can target them and send promotion details for the right product

Clustering



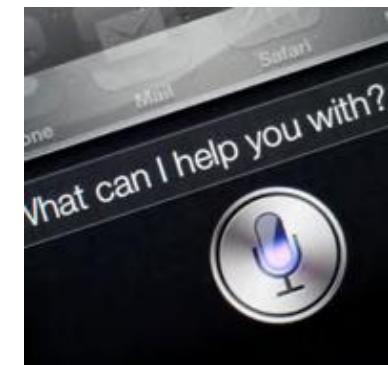
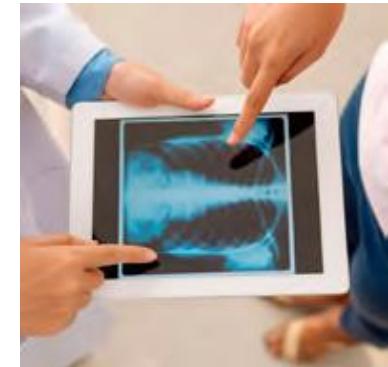
Reinforcement Learning



- ***Reinforcement Learning:*** In this, the machine learns from the environment by interacting with it.
- The machine is provided a set of allowed actions, rules and potential end states.
- By exploring different actions and observing resulting reactions the machine learns to exploit the rules to create a desired outcome.
- Eg. Game of chess, Robotics
- Algorithms used: Neural networks, Learning Automata, Q-Learning, Markov decision process

Deep Learning

- Deep learning: Used to understand and analyse image, sound and video. Uses Neural networks
- Used to understand human language (Natural Language Processing - NLP). Eg. Chatbots



Neural networks

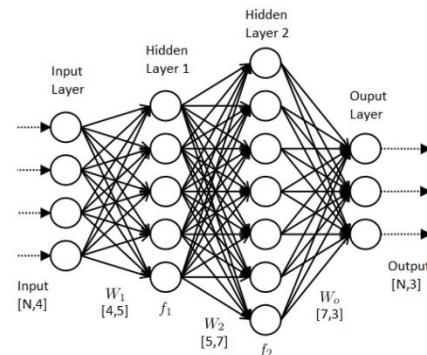


- These networks can learn and **model the relationships between inputs and outputs that are complex**.
- **Examples:** Detecting rare events such as frauds, help doctors with an opinion

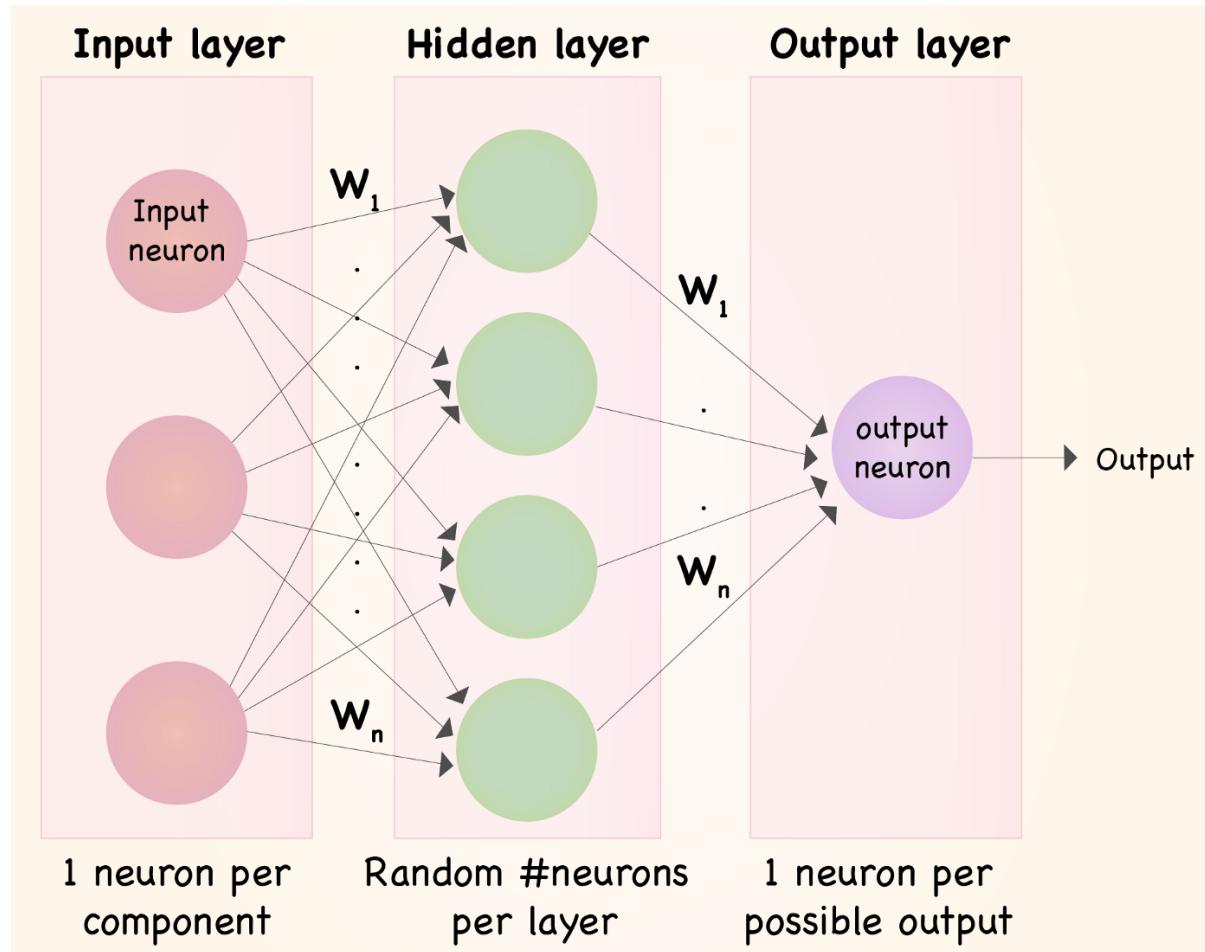


Neural networks -
SAS

Neural networks



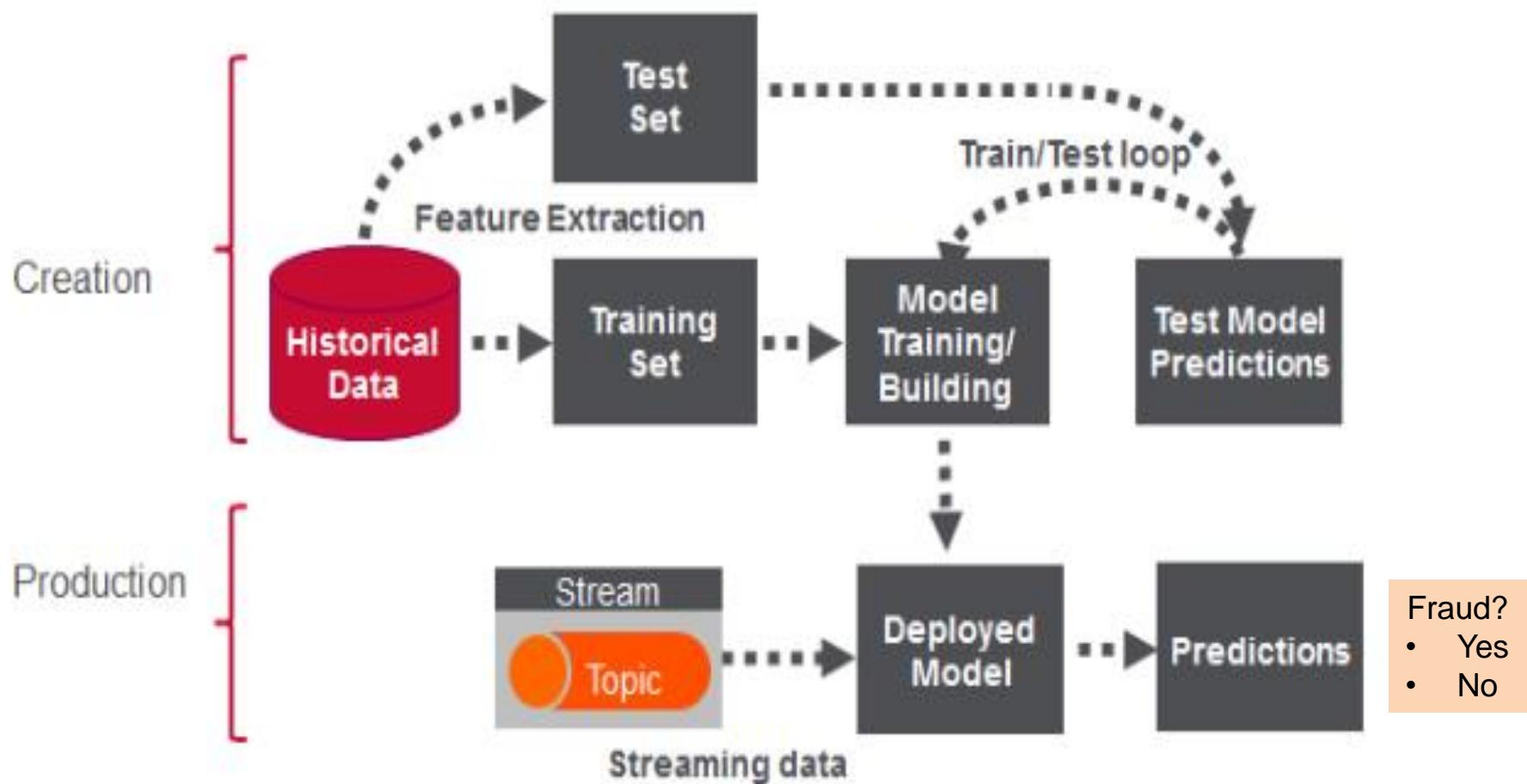
How does a neural network look like?



Steps to build ML model

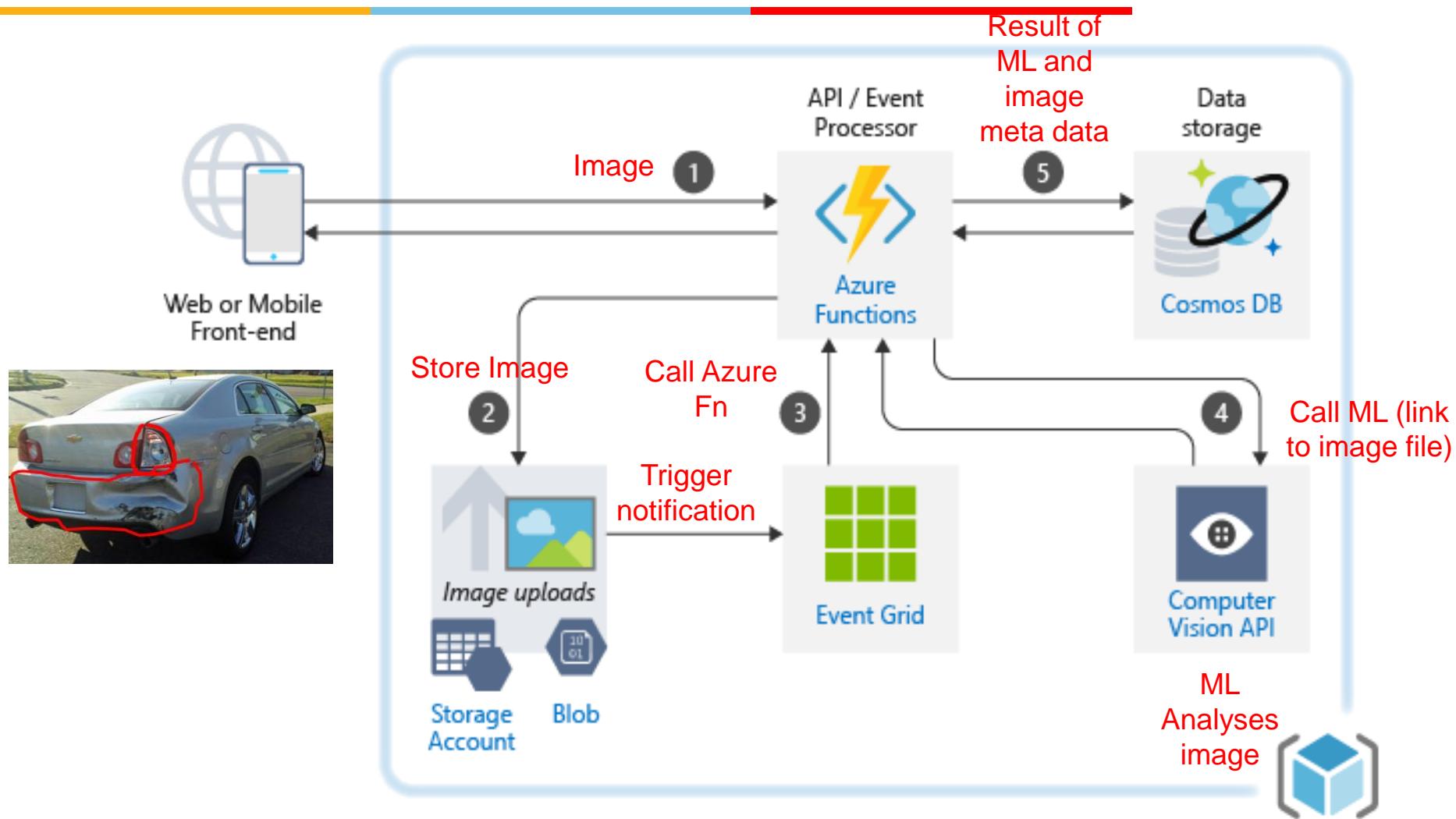
1. Identify the problem to be solved (how will it benefit the business?)
 2. Identify the features: This needs domain knowledge, creativity and lots of time. Ex: If we want to categorize customers, the features used to categorize can be age, salary, product purchased, where purchased, when, etc.
 3. Decide on the model: What algorithm to use – supervised, unsupervised, reinforcement?
 4. Train-test-validate the model
 5. Experiment: Keep improving the model
-

Building the model: Fraud detection



Architecture of ML system:

Image classification for insurance claims



Ref: <https://docs.microsoft.com/en-us/azure/architecture/example-scenario/ai/intelligent-apps-image-processing>

Resource Group

Popular tools

- Scikit Learn - It provides models and algorithms for Classification, Regression, Clustering, Dimensional reduction, Model selection
 - PyTorch – Neural Networks
 - Tensor Flow – Neural networks
 - Apache Mahout - Regression, Clustering, Recommenders, and Distributed Linear Algebra.
 - Spark MLlib –
-

Experience sharing

What problem did you solve using ML?

What steps did you follow to develop the system?

What were the key challenges you faced?



References



ML primer SAS



Case studies in
ML



Workflow of ML
project



5 steps to build a
ML system



Appendix



Reference Chapter 23
Software Architecture in Practice
Third Edition
Len Bass
Paul Clements
Rick Kazman

Module 10

Economic Evaluation through

Cost Benefit Analysis Method



BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Harvinder S Jabbal
SSZG653 Software Architectures

Chapter Outline

Decision-Making Context

The Basis for the Economic Analyses

Putting Theory into Practice: The CBAM

Case Study: The NASA ECS Project

Summary

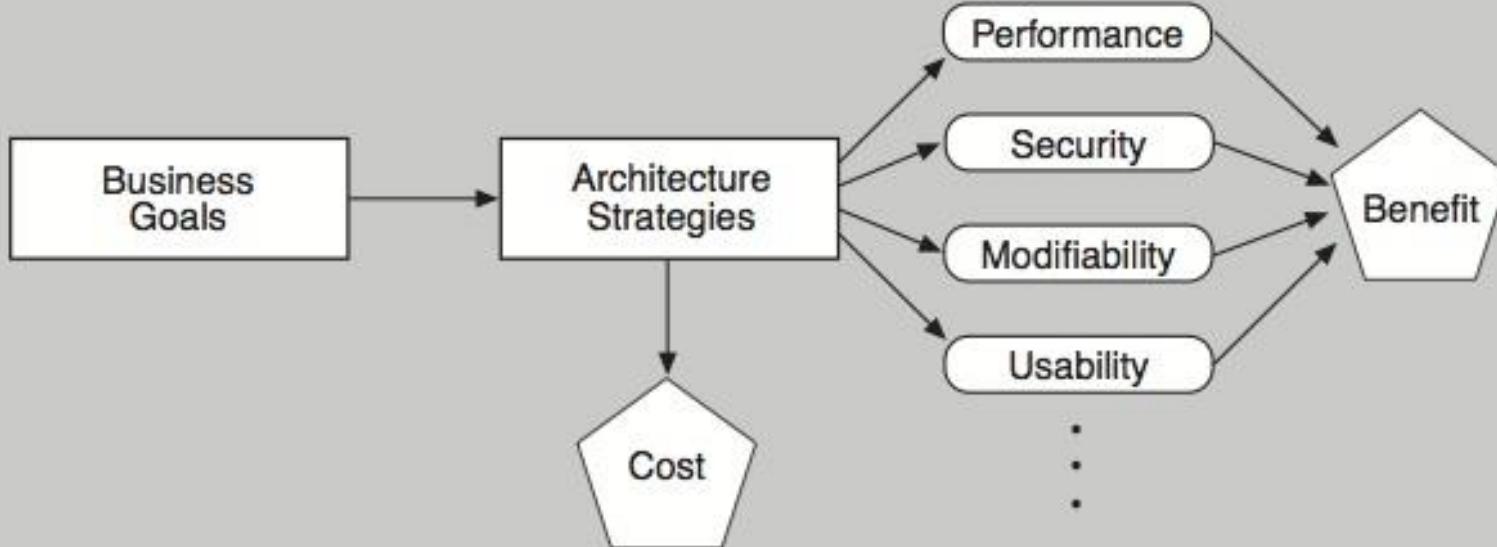


Decision-Making Context

Decision-making Context

The quality attributes achieved by architecture decisions have economic implications in terms of the benefits (*utility*) that can be derived from those decisions.

This interplay between the costs and the benefits of architectural decisions guides (and torments) the architect.



Decision-making Context

Example:

- Using redundant hardware to achieve a desired level of availability has a cost
- Checkpointing to a disk file has a different cost.
- Both of these architectural decisions will result in measurable levels of availability that will have some value to the organization developing the system.

Knowing the costs and benefits associated with particular decisions enables reasoned selection from among competing alternatives.

Economic analysis does not make decisions for the stakeholders. It simply aids in the elicitation and documentation of *value for cost* (VFC).

- VFC: a function of the costs, benefits, and uncertainty of a “portfolio” of architectural investments.
- It gives the stakeholders a framework within which they can apply a rational decision-making process that suits their needs and their risk aversion.

Economic analysis isn't something to apply to every architectural decision, but rather to the most basic ones that put an overarching architectural strategy in place.



Basis for Economic Analyses

Basis for Economic Analyses

Begin with a collection of scenarios generated from requirements elicitation, architectural evaluation, or specifically for economic analysis.

Examine how these scenarios differ in the values of their projected responses.

Assign utility to those values.

- The utility is based on the importance of each scenario with respect to its anticipated response value.

Consider the architectural strategies that lead to the various projected responses.

- Each strategy has a cost, and each impacts *multiple* quality attributes.
- That is, an architectural strategy could be implemented to achieve some projected response, but while achieving that response, it also affects some other quality attributes.
- The utility of these “side effects” must be taken into account when considering a strategy’s overall utility.

Combine this overall utility with the project cost of an architectural strategy to calculate a final VFC measure.

Utility-Response Curves

Our economic analysis uses quality attribute scenarios (from Chapter 4) as the way to concretely express and represent specific quality attributes.

When we vary the values of the responses, and ask what the utility is of each response we get a set of points that we can plot: a *utility-response curve*.

Each scenario's stimulus-response pair provides some utility (value) to the stakeholders, and the utility of different possible values for the response can be compared.

- To help us make major architectural decisions, we might wish to compare the value of high performance against the value of high modifiability against the value of high usability, and so forth. The concept of utility lets us do that.

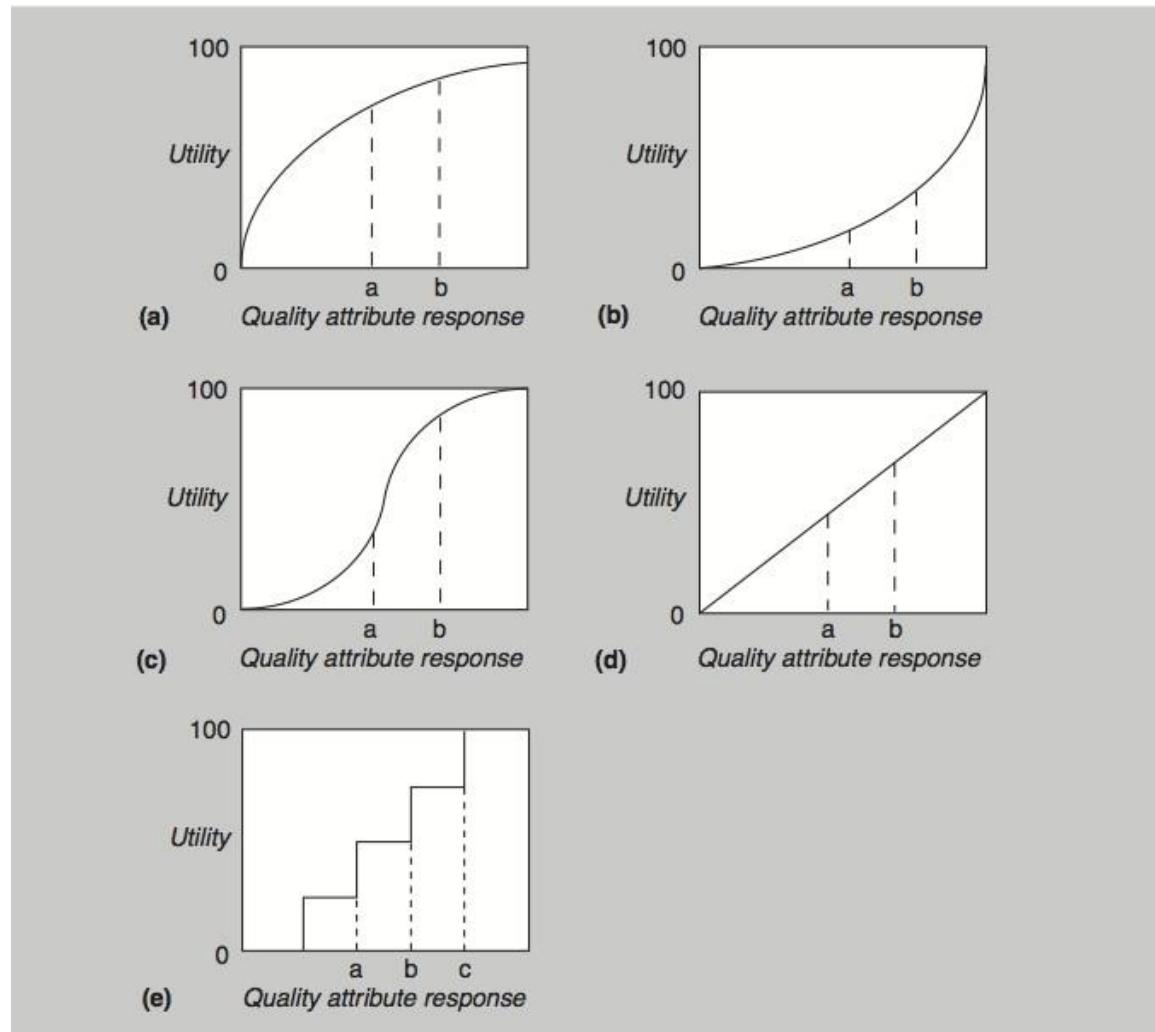
With prodding, stakeholders can express their needs using concrete response measures, such as “99.999 percent available.” But how much would they value slightly less demanding quality attributes, such as “99.99 percent available”? Would that be almost as good?

If so, then the lower cost of achieving that lower value might make that the preferred option,

Capturing the utility of alternative responses of a scenario better enables the architect to make tradeoffs involving that quality attribute.

We can portray each relationship between a set of utility measures and a corresponding set of response measures as a graph—a utility-response curve.

Example Utility-Response Curves



Weighting the Scenarios

Different scenarios will have different importance to the stakeholders.

To make a choice of architectural strategies that is best suited to the stakeholders' desires, we must weight the scenarios.

- It does no good to spend a great deal of effort optimizing a scenario in which the stakeholders actually have very little interest.

Determining Benefit and Normalization



The overall benefit of an architectural strategy across various quality attribute scenarios is the sum of the utility associated with each scenario, weighted by the importance of the scenario.

For each architectural strategy i , its benefit B_i over j scenarios (each with weight W_j) is

$$\bullet B_i = \sum_j (b_{i,j} \times W_j)$$

Each $b_{i,j}$ is calculated as the change in utility (over whatever architectural strategy is currently in place, or is in competition with the one being considered) brought about by the architectural strategy with respect to this scenario:

$$\bullet b_{i,j} = U_{expected} - U_{current}$$

This is the utility of the expected value of the architectural strategy minus the utility of the *current* system relative to this scenario.

Calculating Value for Cost

The VFC for each architectural strategy is the ratio of the total benefit, B_i , to the cost, C_i , of implementing it:

- $VFC = B_i / C_i$

The cost C_i is estimated using a model appropriate for the system and the environment being developed, such as a cost model that estimates implementation cost by measuring an architecture's interaction complexity.

You can use this VFC score to rank-order the architectural strategies under consideration.



Putting Theory into Practice: The CBAM

Practicalities of Utility Curve Determination



To build the utility-response curve, we first determine the quality attribute levels for the best-case and worst-case situations.

The best-case quality attribute level is that above which the stakeholders foresee no further utility.

- For example, a system response to the user of 0.1 second is perceived as instantaneous, so improving it further so that it responds in 0.03 second has no additional utility.

The worst-case quality attribute level is a minimum threshold above which a system must perform; otherwise it is of no use to the stakeholders.

These levels—best-case and worst-case—are assigned utility values of 100 and 0, respectively.

Practicalities of Weighting Determination



One method of weighting the scenarios is to prioritize them and use their priority ranking as the weight.

- For N scenarios, the highest priority one is given a weight of 1, the next highest is given a weight of $(N-1)/N$, and so on.
- This turns the problem of weighting the scenarios into one of assigning priorities.

The stakeholders can determine the priorities through a variety of voting schemes.

- One simple method is to have each stakeholder prioritize the scenarios (from 1 to N) and the total priority of the scenario is the sum of the priorities it receives from all of the stakeholders.
- Voting can be public or secret.

Other schemes are possible. Regardless of the scheme used, it must make sense to the stakeholders and it must suit their culture.

Practicalities of Cost Determination



There are very few cost models for various architectural strategies.

There are many software cost models, but they are based on overall system characteristics such as size or function points.	These are inadequate to answer the question of how much does it cost to, for example, use a publish-subscribe pattern in a particular portion of the architecture.	There are cost models that are based on complexity of modules (by function point analysis according to the requirements assigned to each module) and the complexity of module interaction, but these are not widely used in practice.	More widely used in practice are corporate cost models based on previous experience with the same or similar architectures, or the experience and intuition of senior architects.
--	--	---	---

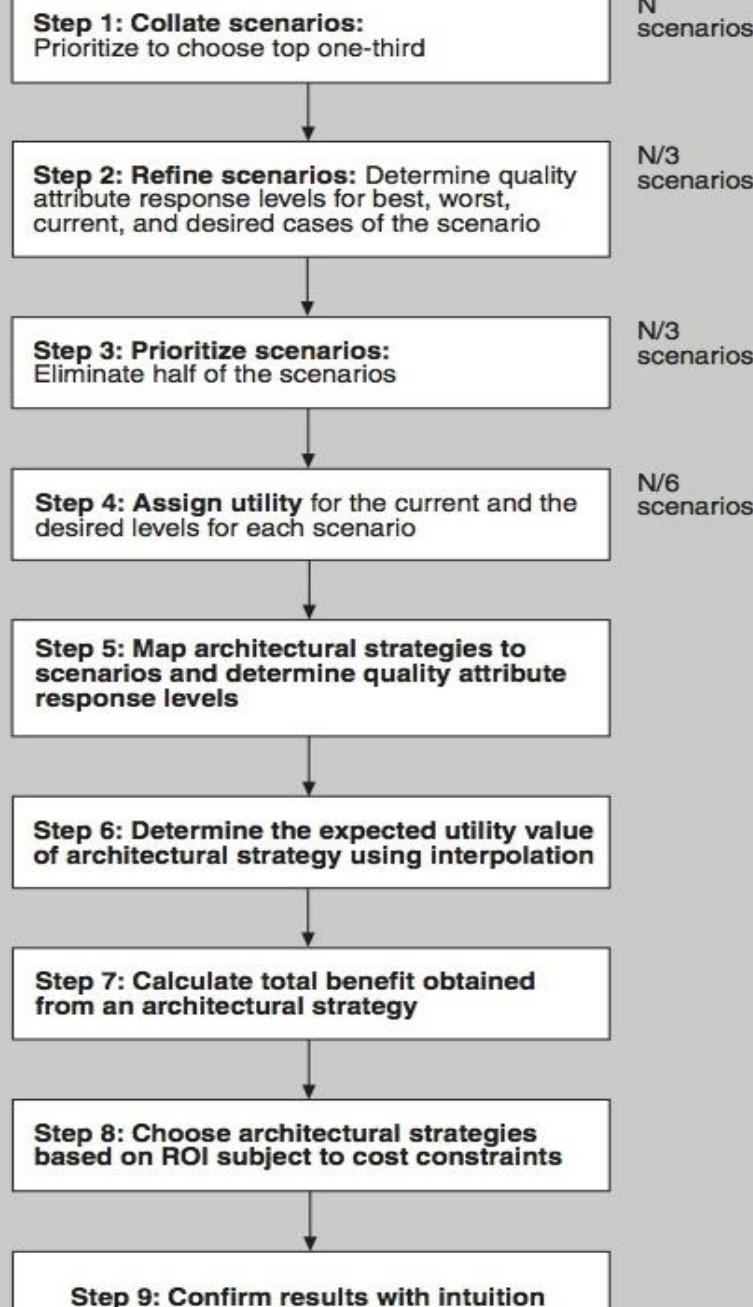
Practicalities of Cost Determination



Architects often turn to cost estimation techniques.

An absolute number for cost isn't necessary to rank candidate architecture strategies.

- You can often say something like "Suppose strategy A costs \$x. It looks like strategy B will cost \$2x, and strategy C will cost \$0.5x." That's enormously helpful.
- A second approach is to use very coarse estimates. Or if you lack confidence for that degree of certainty, you can say something like "Strategy A will cost a lot, strategy B shouldn't cost very much, and strategy C is probably somewhere in the middle."



Cost Benefit Analysis Method (CBAM)

The CBAM places the principles just discussed into a set of steps: a method.

CBAM Stakeholders

The stakeholders in a CBAM exercise include people who can authoritatively speak to the utility of various quality attribute responses.

They probably include the same people who were the source of the quality attribute scenarios being used as input.

Step 1

Collate scenarios.

- Give the stakeholders the chance to contribute new scenarios.
- Ask the stakeholders to prioritize the scenarios based on satisfying the business goals of the system.
- This can be an informal prioritization using a simple scheme such as “high, medium, low” to rank the scenarios.
- Choose the top one-third for further study.

Step 2

Refine scenarios.

- Refine the scenarios chosen in step 1, focusing on their stimulus-response measures.
- Elicit the worst-case, current, desired, and best-case quality attribute response level for each scenario.
- For example, a refined performance scenario might tell us that worst-case performance for our system's response to user input is 12 seconds, the best case is 0.1 seconds, and our desired response is 0.5 seconds. Our current architecture provides a response of 1.5 seconds.

Step 3

Prioritize scenarios.

- Prioritize the refined scenarios, based on stakeholder votes.
- Give 100 votes to each stakeholder and have them distribute the votes among the scenarios, where their voting is based on the desired response value for each scenario.
- Total the votes and choose the top 50 percent of the scenarios for further analysis.
- Assign a weight of 1.0 to the highest-rated scenario; assign the other scenarios a weight relative to the highest rated.
- This becomes the weighting used in the calculation of a strategy's overall benefit.
- Make a list of the quality attributes that concern the stakeholders.

Step 4

Assign utility.

- Determine the utility for each quality attribute response level (worst-case, current, desired, best-case) for the scenarios from step 3.
- You can conveniently capture these utility curves in a table (one row for each scenario, one column for each of the four quality attribute response levels).

Scenario	Worst Case	Current	Desired	Best Case
Scenario #17: Response to user input	12 seconds	1.5 seconds	0.5 seconds	0.1 seconds
	Utility 5	Utility 50	Utility 80	Utility 85

Step 5

Map architectural strategies to scenarios and determine their expected quality attribute response levels.

- For each architectural strategy under consideration, determine the expected quality attribute response levels that will result for each scenario.

Step 6

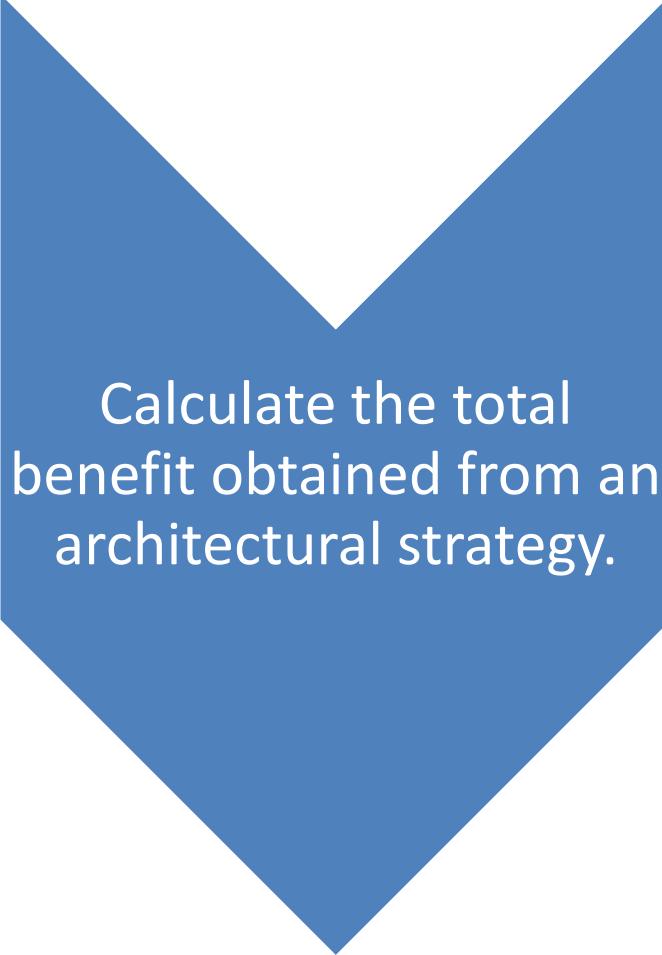
Determine the utility of the expected quality attribute response levels by interpolation.

- Using the elicited utility values (that form a utility curve), determine the utility of the expected quality attribute response level for the architectural strategy.
- Do this for each relevant quality attribute enumerated in step 3.
- For example, if we are considering a new architectural strategy that would result in a response time of 0.7 seconds, we would assign a utility proportionately between 50 (which it exceeds) and 80 (which it doesn't exceed).
- The formula for interpolation between two data points (x_a, y_a) and (x_b, y_b) is:

$$y = y_a + (y_b - y_a) \frac{(x - x_a)}{(x_b - x_a)}$$

- For us, the x values are the quality attribute response levels and the y values are the utility values. So, employing this formula, the utility value of a 0.7-second response time is 74 .

Step 7



Calculate the total benefit obtained from an architectural strategy.

- Subtract the utility value of the “current” level from the expected level and normalize it using the votes elicited in step 3.
- Sum the benefit due to a particular architectural strategy across all scenarios and across all relevant quality attributes.

Step 8

Choose architectural strategies based on VFC subject to cost and schedule constraints.

- Determine the cost and schedule implications of each architectural strategy.
- Calculate the VFC value for each as a ratio of benefit to cost. Rank-order the architectural strategies according to the VFC value and choose the top ones until the budget or schedule is exhausted.

Step 9



Confirm results with intuition.

- For the chosen architectural strategies, consider whether these seem to align with the organization's business goals.
- If not, consider issues that may have been overlooked while doing this analysis.
- If there are significant issues, perform another iteration of these steps.



CASE STUDY:

The NASA ECS Project

Case Study: The NASA ECS Project



The Earth Observing System is a constellation of NASA satellites that gathers data for the U.S. Global Change Research Program and other scientific communities worldwide.

The Earth Observing System Data Information System (EOSDIS) Core System (ECS) collects data from various satellite downlink stations for further processing.

ECS's mission is to process the data into higher-form information and make it available to scientists in searchable form. The goal is to provide both a common way to store (and hence process) data and a public mechanism to introduce new data formats and processing algorithms, thus making the information widely available.

Case Study: The NASA ECS Project



The ECS processes an input stream of hundreds of gigabytes of raw environment-related data per day. The computation of 250 standard “products” results in thousands of gigabytes of information that is archived at eight data centers in the United States. The system has important performance and availability and modifiability requirements.

The ECS project manager had a limited annual budget to maintain and enhance his current system. The manager used the CBAM to make a rational decision based on the economic criterion of return on investment.

Step 1: Collate Scenarios

IN PRIORITY ORDER

Note that they are not yet well formed and that some of them do not have defined responses. These issues are resolved in step 2, when the number of scenarios is reduced.

- 1 • Reduce data distribution failures that result in hung distribution requests requiring manual intervention.
- 2 • Reduce data distribution failures that result in lost distribution requests.
- 3 • Reduce the number of orders that fail on the order submission process.
- 4 • Reduce order failures that result in hung orders that require manual intervention.
- 5 • Reduce order failures that result in lost orders.
- 6 • There is no good method of tracking ECSGuest failed/canceled orders without much manual intervention (e.g., spreadsheets).
- 7 • Users need more information on why their orders for data failed.
- 8 • Because of limitations, there is a need to artificially limit the size and number of orders.
- 9 • Small orders result in too many notifications to users.
- 10 • The system should process a 50-GB user request in one day, and a 1-TB user request in one week.

Step 2: Refine Scenarios

The scenarios were refined, paying particular attention to precisely specifying their stimulus-response measures.

The worst-case, current-case, desired-case, and best-case response goals for each scenario were elicited and recorded in a table.

Step 2: Refine Scenarios

Scenario	Worst	Current	Desired	Best
1	10% hung	5% hung	1% hung	0% hung
2	> 5% lost	< 1% lost	0% lost	0% lost
3	10% fail	5% fail	1% fail	0% fail
4	10% hung	5% hung	1% hung	0% hung
5	10% lost	< 1% lost	0% lost	0% lost
6	50% need help	25% need help	0% need help	0% need help
7	10% get information	50% get information	100% get information	100% get information
8	50% limited	30% limited	0% limited	0% limited
9	1/granule	1/granule	1/100 granules	1/1,000 granules
10	< 50% meet goal	60% meet goal	80% meet goal	> 90% meet goal

Step 3: Prioritize Scenarios

In voting on the refined representation of the scenarios, the close-knit team deviated slightly from the method.

Rather than vote individually, they chose to discuss each scenario and arrived at a determination of its weight via consensus.

The votes allocated to the entire set of scenarios were constrained to 100.

Although the stakeholders were not required to make the votes multiples of 5, they felt that this was a reasonable resolution and that more precision was neither needed nor justified.

Step 3: Prioritize Scenarios

Scenario	Votes	Worst	Current	Desired	Best
1	10	10% hung	5% hung	1% hung	0% hung
2	15	> 5% lost	< 1% lost	0% lost	0% lost
3	15	10% fail	5% fail	1% fail	0% fail
4	10	10% hung	5% hung	1% hung	0% hung
5	15	10% lost	< 1% lost	0% lost	0% lost
6	10	50% need help	25% need help	0% need help	0% need help
7	5	10% get information	50% get information	100% get information	100% get information
8	5	50% limited	30% limited	0% limited	0% limited
9	10	1/granule	1/granule	1/100 granules	1/1,000 granules
10	5	< 50% meet goal	60% meet goal	80% meet goal	> 90% meet goal

Step 4: Assign Utility

In this step the utility for each scenario was determined by the stakeholders, again by consensus.

A utility score of 0 represented no utility; a score of 100 represented the most utility possible.

Step 4: Assign Utility

Scenario	Votes	Utility Scores			
		Worst	Current	Desired	Best
1	10	10	80	95	100
2	15	0	70	100	100
3	15	25	70	100	100
4	10	10	80	95	100
5	15	0	70	100	100
6	10	0	80	100	100
7	5	10	70	100	100
8	5	0	20	100	100
9	10	50	50	80	90
10	5	50	50	80	90

Their Expected Quality Attribute Response Levels

Based on the requirements implied by the preceding scenarios, a set of 10 architectural strategies was developed by the ECS architects.

For each architectural strategy/scenario pair, the response levels expected to be achieved with respect to that scenario are shown (along with the current response, for comparison purposes).

Step 5 Results (Scenarios 1-5)

Strategy	Name	Description	Scenarios Affected	Current Response	Expected Response
1	Order persistence on submission	Store an order as soon as it arrives in the system.	3	5% fail	2% Fail
			5	<1% lost	0% lost
			6	25% need help	0% need help
2	Order chunking	Allow operators to partition large orders into multiple small orders.	8	30% limited	15% limited
3	Order bundling	Combine multiple small orders into one large order.	9	1 per granule	1 per 100
			10	60% meet goal	55% meet goal
4	Order segmentation	Allow an operator to skip items that cannot be retrieved due to data quality or availability issues.	4	5% hung	2% hung
5	Order reassignment	Allow an operator to reassign the media type for items in an order.	1	5% hung	2% hung

Step 5 Results (Scenarios 5-10)

6	Order retry	Allow an operator to retry an order or items in an order that may have failed due to temporary system or data problems.	4	5% hung	3% hung	
7	Forced order completion	Allow an operator to override an item's unavailability due to data quality constraints.	1	5% hung	3% hung	
8	Failed order notification	Ensure that users are notified only when part of their order has truly failed and provide detailed status of each item; user notification occurs only if operator okays notification; the operator may edit notification.	6	25% need help	20% need help	
			7	50% get information	90% get information	
9	Granule-level order tracking	An operator and user can determine the status for each item in their order.	6	25% need help	10% need help	
			7	50% get information	95% get information	
10	Links to user information	An operator can quickly locate a user's contact information. Server will access SDSRV information to determine any data restrictions that might apply and will route orders/order segments to appropriate distribution capabilities, including DDIST, PDS, external <u>subsetters</u> and data processing tools, etc.	7	50% get information	60% get information	

Step 6: Determine the Utility of the “Expected” Quality



Attribute Response Levels by Interpolation

Once the expected response level of every architectural strategy has been characterized with respect to a set of scenarios, their utility can be calculated by consulting the utility scores for each scenario's current and desired responses for all of the affected attributes.

Using these scores, we may calculate, via interpolation, the utility of the expected quality attribute response levels for the architectural strategy/scenario pair.

Step 6 Results

Strategy	Name	Scenarios Affected	Current Utility	Expected Utility
1	Order persistence on submission	3	70	90
		5	70	100
		6	80	100
2	Order chunking	8	20	60
3	Order bundling	9	50	80
		10	70	65
4	Order segmentation	4	80	90
5	Order reassignment	1	80	92
6	Order retry	4	80	85
7	Forced order completion	1	80	87
8	Failed order notification	6	80	85
		7	70	90
9	Granule-level order tracking	6	80	90
		7	70	95
10	Links to user information	7	70	75

Step 7: Calculate the Total Benefit Obtained from an Architectural Strategy



Total benefit of each architectural strategy can now be calculated:

- $B_i = \sum_j (b_{i,j} \times W_j)$

This equation calculates total benefit as the sum of the benefit that accrues to each scenario, normalized by the scenario's relative weight.

Using this formula, the total benefit scores for each architectural strategy are now calculated.

Step 7

Strategy	Scenario Affected	Scenario Weight	Raw Architectural Strategy Benefit	Normalized Architectural Strategy Benefit	Total Architectural Strategy Benefit
1	3	15	20	300	
1	5	15	30	450	
1	6	10	20	200	950
2	8	5	40	200	200
3	9	10	30	300	
3	10	5	-5	-25	275
4	4	10	10	100	100
5	1	10	12	120	120
6	4	10	5	50	50
7	1	10	7	70	70
8	6	10	5	50	
8	7	5	20	100	150
9	6	10	10	100	
9	7	5	25	125	225
10	7	5	5	25	25

Step 8: Choose Architectural Strategies Based on VFC Subject to Cost Constraints

To complete the analysis, the team estimated cost for each architectural strategy.

The estimates were based on experience with the system, and a return on investment for each architectural strategy was calculated.

Using the VFC, we were able to rank each strategy.

The ranks roughly follow the ordering in which the strategies were proposed: strategy 1 has the highest rank; strategy 3 the second highest. Strategy 9 has the lowest rank; strategy 8, the second lowest.

This simply validates stakeholders' intuition about which architectural strategies were going to be of the greatest benefit. For the ECS these were the ones proposed first.

Step 8

Strategy	Cost	Total Strategy Benefit	Strategy VFC	Strategy Rank
1	1200	950	0.79	1
2	400	200	0.5	3
3	400	275	0.69	2
4	200	100	0.5	3
5	400	120	0.3	7
6	200	50	0.25	8
7	200	70	0.35	6
8	300	150	0.5	3
9	1000	225	0.22	10
10	100	25	0.25	8

Results of the CBAM Exercise



The most obvious results of the CBAM are the ordering of architectural strategies based on their predicted VFC.



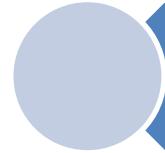
However, there are social and cultural benefits as well.



The CBAM process provides structure to what is always largely unstructured discussions, where requirements and architectural strategies are freely mixed and where stimuli and response goals are not clearly articulated.



The CBAM process forces the stakeholders to make their scenarios clear in advance, to assign utility levels of specific response goals, and to prioritize these scenarios based on the resulting determination of utility.



Finally, it produces clarification of both scenarios and requirements, which by itself is a significant benefit.

Summary

Architecture-based economic analysis is grounded on understanding the utility-response curve of various scenarios and casting them into a form that makes them comparable.

Once they are in this common form—based on the common coin of utility—the VFC for each architecture improvement, with respect to each relevant scenario, can be calculated and compared.

Applying the theory in practice has a number of practical difficulties, which CBAM solves.

The application of economic techniques is inherently better than the ad hoc decision-making approaches that projects employ today.

Giving people the appropriate tools to frame and structure their discussions and decision making is an enormous benefit to the disciplined development of a complex software system.

Thank you.....

Credits

- **Chapter Reference from Text T1: 23**
- Slides have been adapted from Authors Slides
Software Architecture in Practice – Third Ed.
 - Len Bass
 - Paul Clements
 - Rick Kazman