**IMPORTANT NOTICE TO STUDENTS**

These slides are **NOT** to be used as a replacement for student notes.
These **slides** are sometimes **vague and incomplete on purpose** to spark class discussions

# Introduction to Cloud Computing


*CS 446/646 ECE452*
*Jul 4th, 2011*

**WATERLOO**
**CHERITON SCHOOL OF COMPUTER SCIENCE**

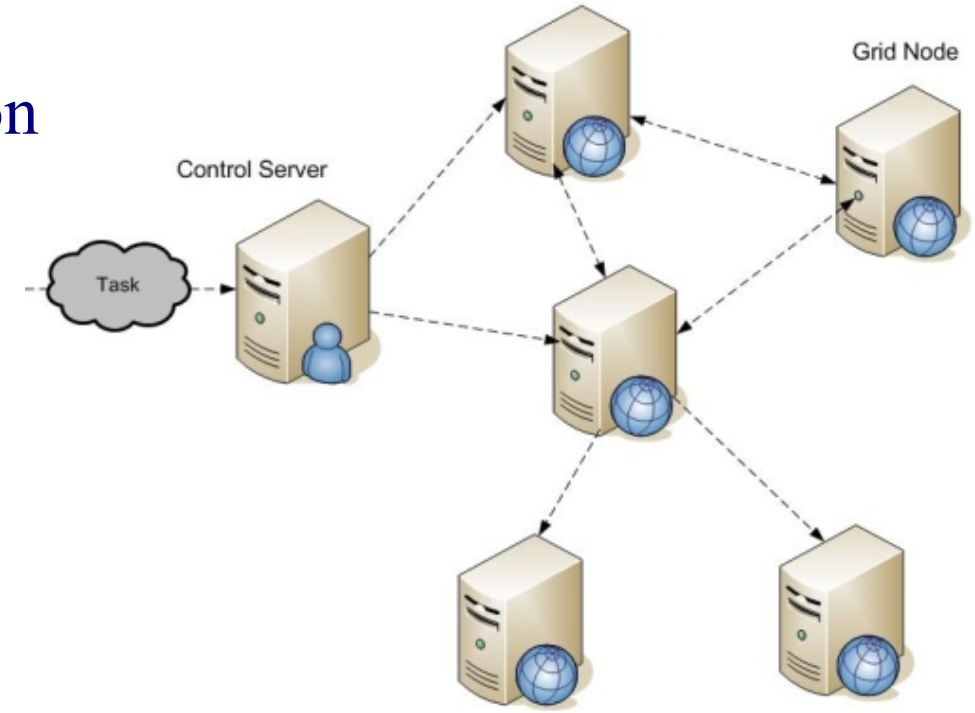
# Grid Computing

Def

- *"**combination of** computer **resources** from multiple administrative domains **applied** to a **common task**" [1]*

Characteristics

- distributed parallel computation
- many different applications
- constructed with ***middleware***
- variable size
- heterogeneous composition






[1] http://en.wikipedia.org/wiki/Grid_computing

# Cloud versus Grid Computing*

## Cloud Computing

- tightly coupled nodes (can be dissimilar)
- high inter-node bandwidth
- centralized management & job scheduling
- standards are being developed
- **virtualized resources**

## Grid Computing

- heterogeneous loosely coupled nodes
- unpredictable inter-node bandwidth
- distributed management & job scheduling
- standardized protocols & interfaces

# Cloud & Grid Computing*

Similarities

- distributed computing
- reduced cost
- increased reliability
- increased flexibility

# Utility Computing

Def

- *"The **packaging** of **computing** resources (computation, storage etc.) as a **metered service**"**

Observation

- not a new concept
  - *"If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry."*
    *John McCarthy, MIT Centennial in 1961*



* http://en.wikipedia.org/wiki/Utility_computing

# What is Cloud Computing

New Label?

- cloud computing → grid computing + utility computing
  - **yes**: *"the vision is the same"**
    - reduce cost, increase reliability & flexibility
  - **no**: "*on demand computing, larger amounts of data*"*
  - **yes**: *"fundamentally the problems are the same"**
    - management of nodes
    - consumer driven
    - parallel computation
- **difficult to define****
- NIST (National Institute of Standards & Technology)
  - "*universally*" accepted definition

*Cloud computing and grid computing 360-degree compared I. Foster et al. 2008

** "Twenty Experts Define Cloud Computing", SYS-CON Media Inc 2008.

# Cloud Computing – NIST

Definition

- *"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models."**

* http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

# Cloud Computing – NIST

## Definition

- *"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models."**

* http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

# Cloud Computing – NIST

## Definition

- *"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models."**

* http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

# Cloud Computing – NIST

## Definition

- *"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models."**

* http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

# Cloud Computing – NIST

## Definition

- *"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models."**

* http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

# Cloud Computing – NIST

Definition

- *"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models."**

* http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

# NIST Essential Characteristics

On-demand self-service*

- a consumer can *unilaterally* provision computing capabilities *without human interaction* with the service provider
- computing capabilities
  - server time
  - network storage
  - number of servers etc.

seems a bit strange to exclude human interaction from this process?

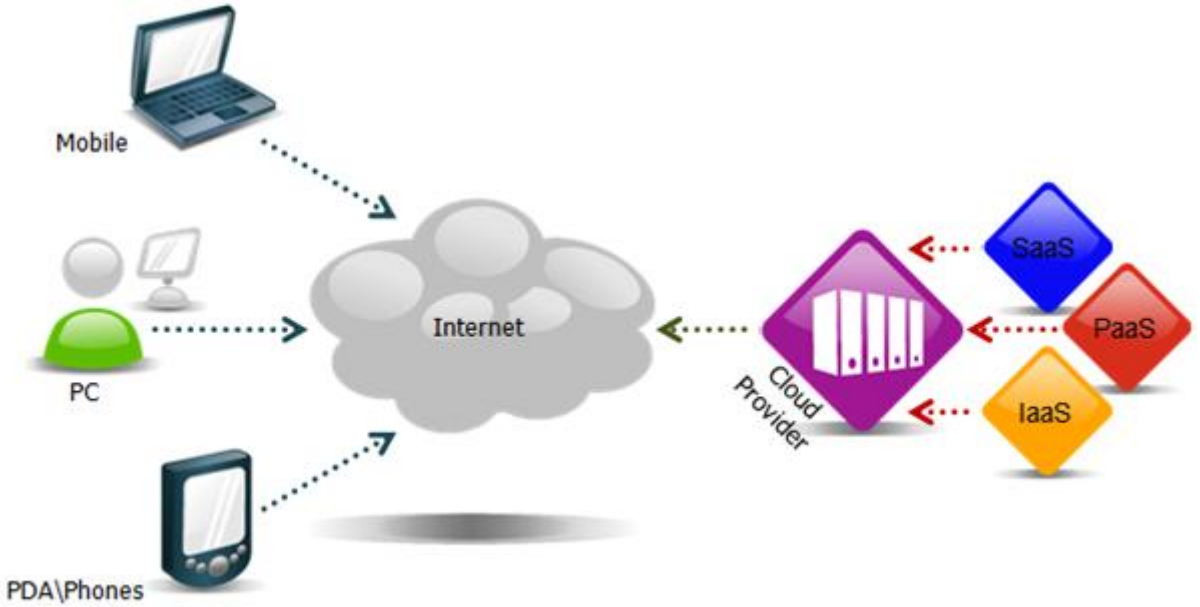


*Draft NIST Working Definition of Cloud Computing – 2009, Peter Mell and Tim Grance

# NIST Essential Characteristics

Ubiquitous network access

- capabilities are
  - **available** over the network
  - **accessed** through standard mechanisms
- to support & promote
  - heterogeneous (thin or thick) client platforms



# NIST Essential Characteristics

Multi-tenancy / Resource pooling

- provider's **computing resources** are **pooled** to **serve** multiple **consumers**
- computing resources
  - storage
  - processing
  - memory
  - network bandwidth & virtual machines
- location independence
  - no control over the exact location of the resources

# NIST Essential Characteristics

Multi-tenancy / Resource pooling

- has major **implications**
  - performance, scalability, security
- example (data & multi-tenancy)
  - to discuss in class
  - considerations (evolution, scalability, management etc.)
  - design
- non-functional concerns for multi-tenant applications
  - security & privacy, release management
  - configurable options (UI, workflows etc.)

**Futher Reading**: http://msdn.microsoft.com/en-us/library/aa479086.aspx

# NIST Essential Characteristics

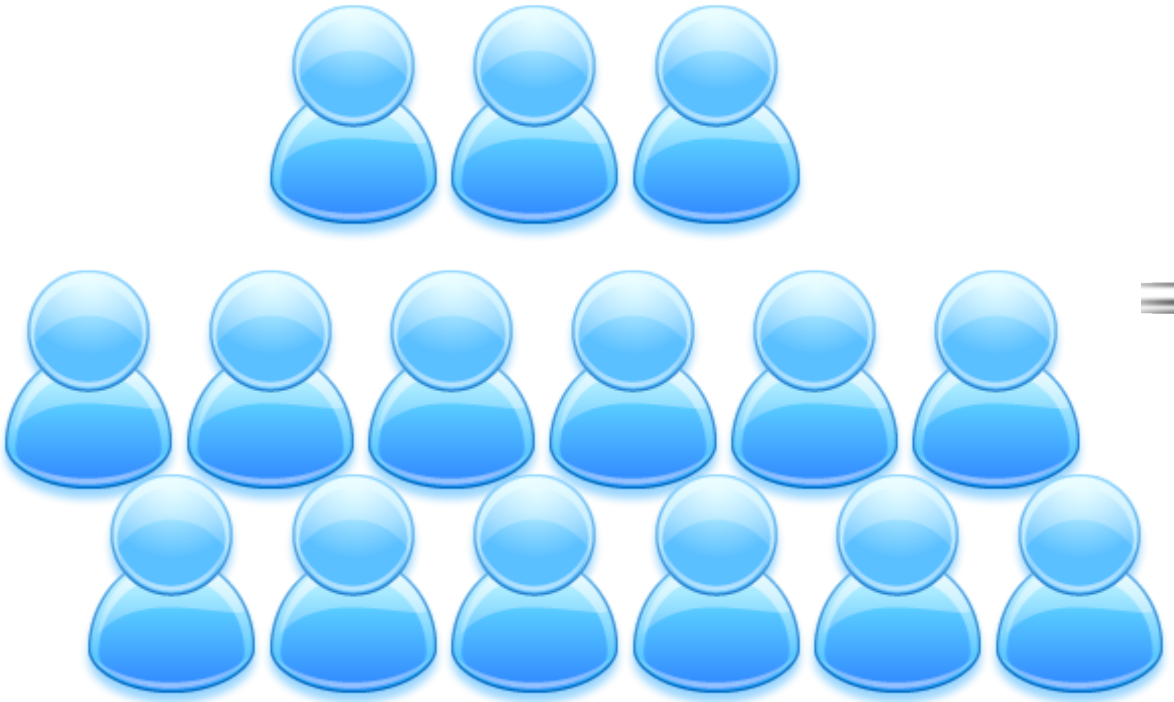
Rapid elasticity

- capabilities can be rapidly and elastically provisioned
- unlimited (virtual) resources
- predicting a ceiling is difficult

# NIST Essential Characteristics

Rapid elasticity

- capabilities can be rapidly and elastically provisioned
- unlimited (virtual) resources
- predicting a ceiling is difficult



# NIST Essential Characteristics

Measured service

- metering capability of service/resource abstractions
  - storage
  - processing
  - bandwidth
  - active user accounts
- **OK so what happened to utility computing – pay as you go model?**
  - NIST does not talk about $$
  - more on this later when we discuss deployment models

# Relevant Technologies

Access

- well defined protocols for communication
- broadband / high speed access

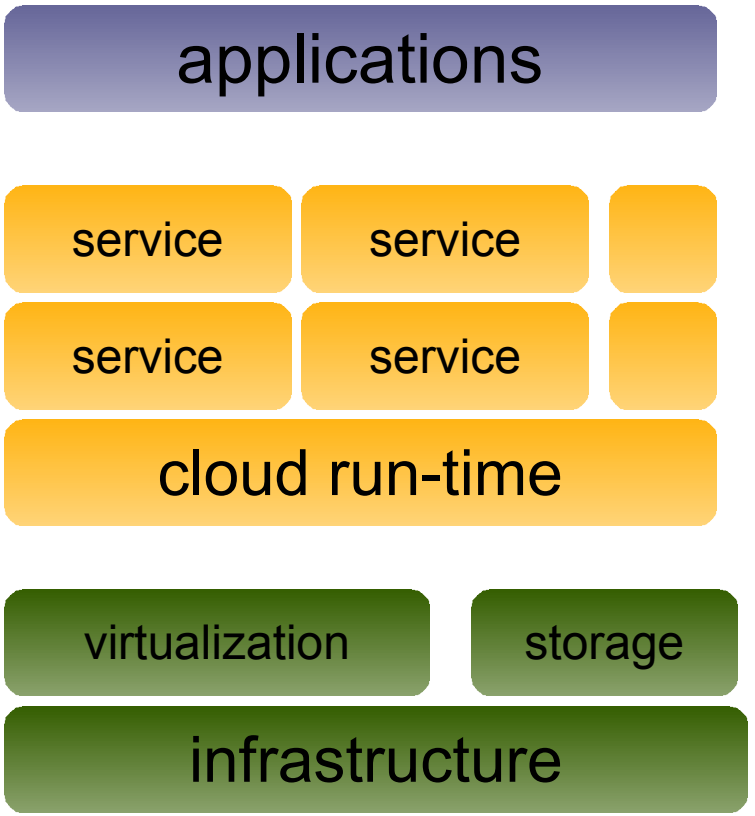
Distributed Computing

- for data storage and computation

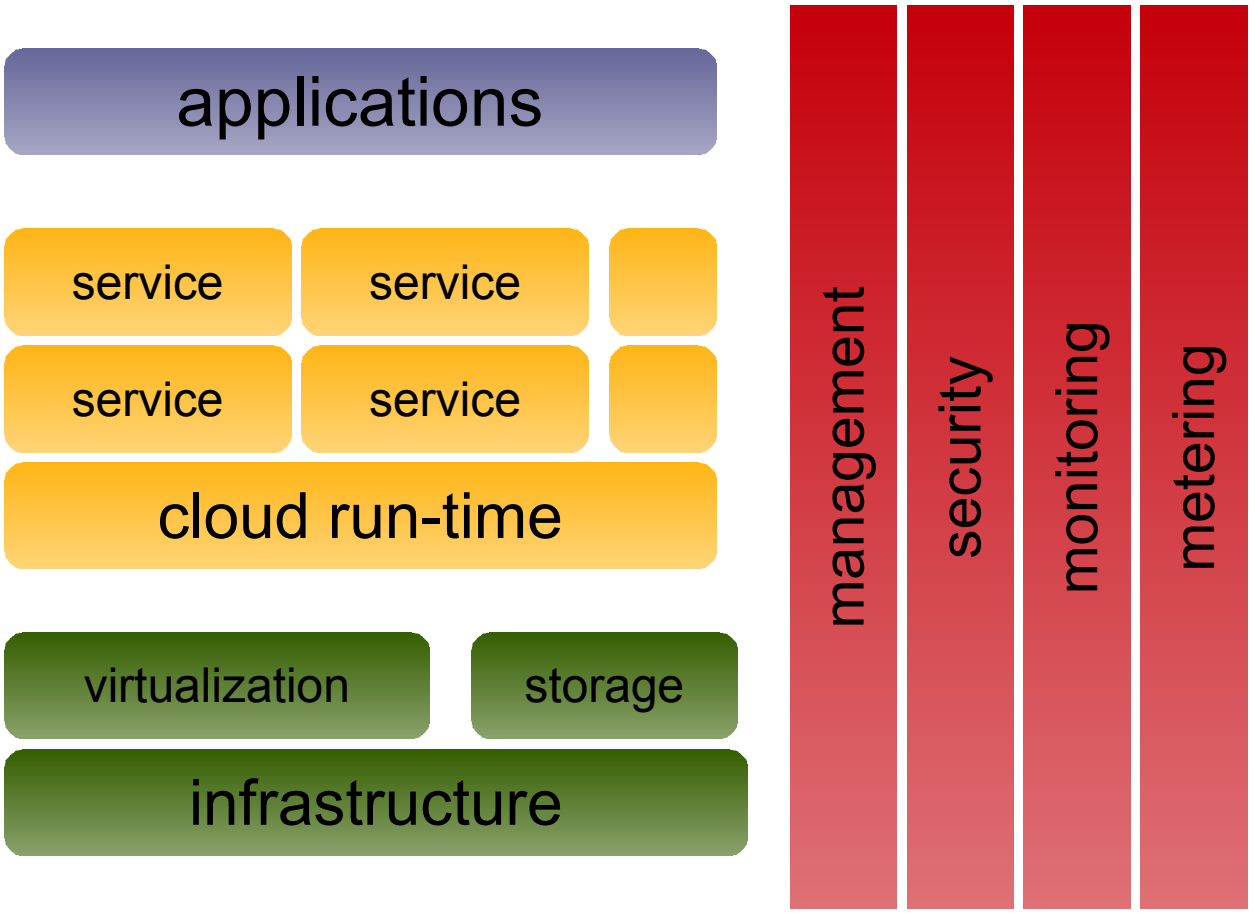
Virtualization

- decoupling from the physical computing resources
- types:
  - hardware, memory, data storage, data schema, network

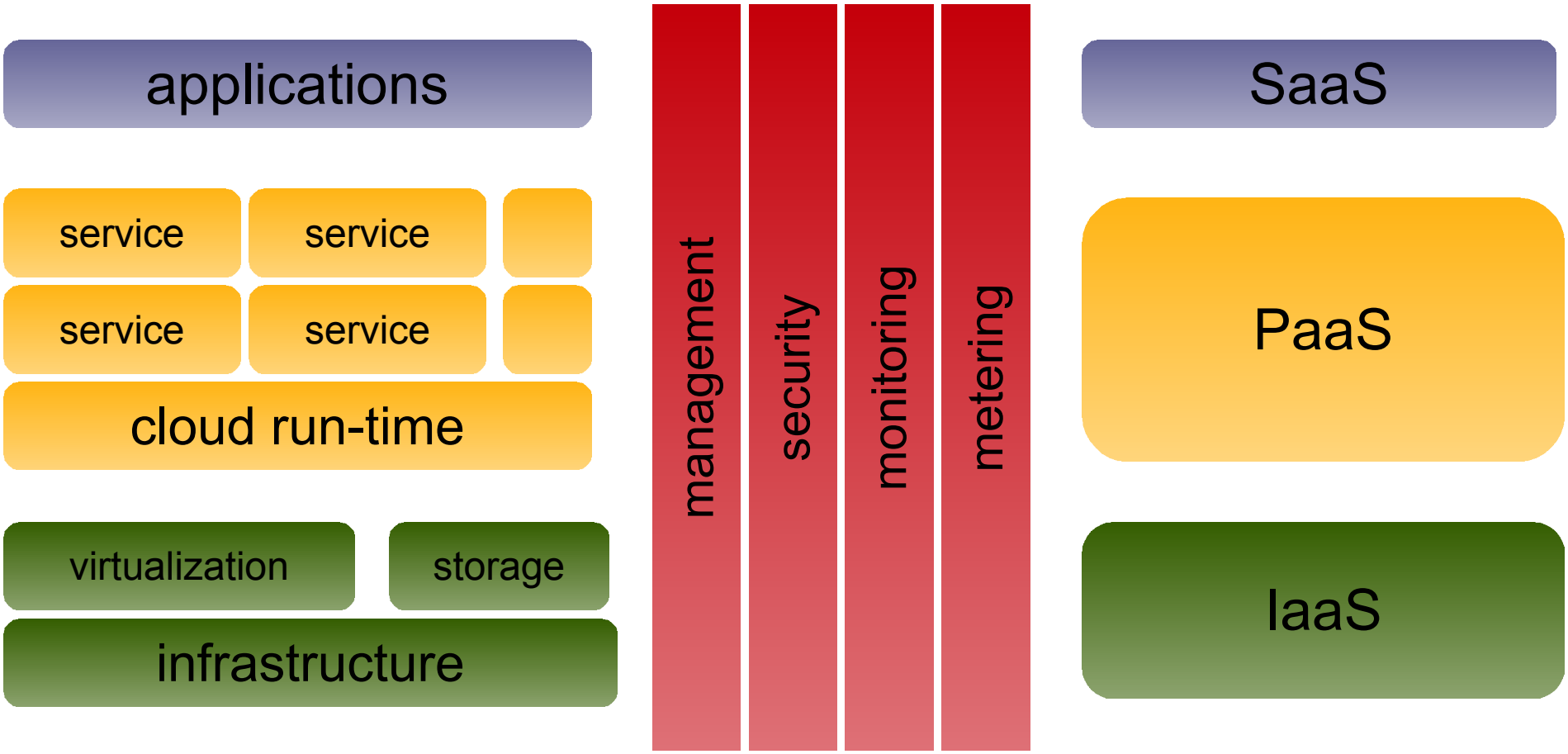
# Reference Architecture



# Reference Architecture



# Reference Architecture



# SPI Services

SaaS (Software-as-a-Service)

- vendor/provider controlled applications accessed over the network
- characteristics
  - network based access
  - multi-tenancy
  - single software release for all

SaaS Examples

  - Salesforce.com, Google Docs

# SPI Services

## SaaS & Multi-tenancy

- SaaS applications are multi-tenant applications
- application data
  - Google docs

## SaaS Application Design

- SaaS applications are '***net native***'
- configurability, efficiency, and scalability
- SOA & SaaS

# SPI Services

Net Native Application Characteristics

- cloud specific design, development & deployment
  - multi-tenant data model
  - builtin metering & management
  - browser based client & client tools
  - customization via configuration

# SPI Services

SaaS Disadvantages

- dependency on
  - network, cloud service provider
- performance
  - limited client bandwidth
- security
  - good: better security than personal computers
  - bad: CSP is in charge of the data
  - ugly: user privacy

# SPI Services

PaaS (Platform-as-a-Service)

- **vendor provided** development environment
  - tools & technology selected by vendor
  - control over data life-cycle

Advantages

- rapid development & deployment
- scalability & fault-tolerance offered by provider
- small start-up cost
  - considerations: required skills set, money

# SPI Services

PaaS – Architectural Characteristics

- multi-tenancy
  - data
- native scalability
  - load balancing & fail-over
- native integrated management & monitoring
  - performance
  - resource consumption/utilization
  - load

# SPI Services

PaaS Disadvantages

- inherits all from SaaS
- choice of development technology is limited to vendor provided/supported tools and services

PaaS Examples

- Google app engine
  - Google Site + Google Docs

# SPI Services

## IaaS (Infrastructure-as-a-Service)

- vendor provided and consumer provisioned computing resources
  - processing, storage, network, etc.
  - consumer is provided customized virtual machines
  - consumer has control over
    - OS, memory
    - storage
    - servers & deployment configurations
    - limited control over network resources

# SPI Services

Advantages

- infrastructure scalability
- native integrated management
  - performance, resource consumption/utilization, load
- economical cost
  - hardware, IT support

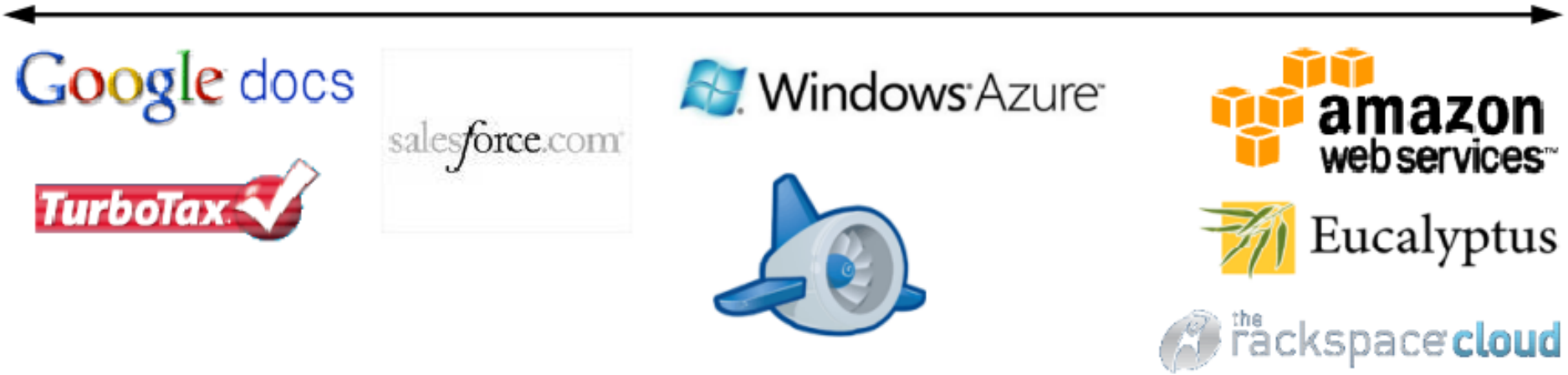
IaaS Examples

- Amazon Elastic Compute Cloud – EC2

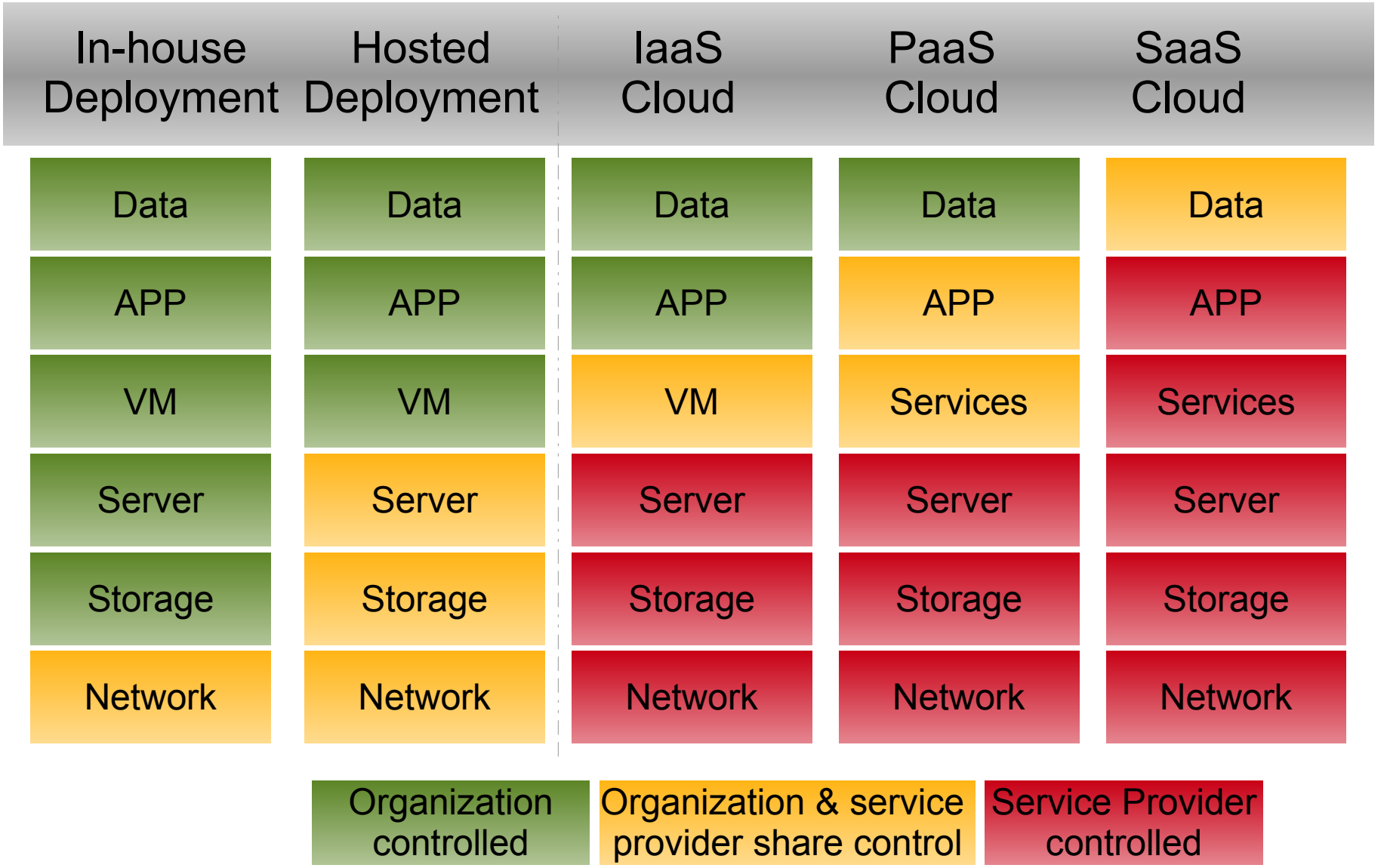
# SPI Services

**less flexible**
**more constrained**
**less effort**

**more flexible**
**less constrained**
**more effort**



# SPI Services & Control







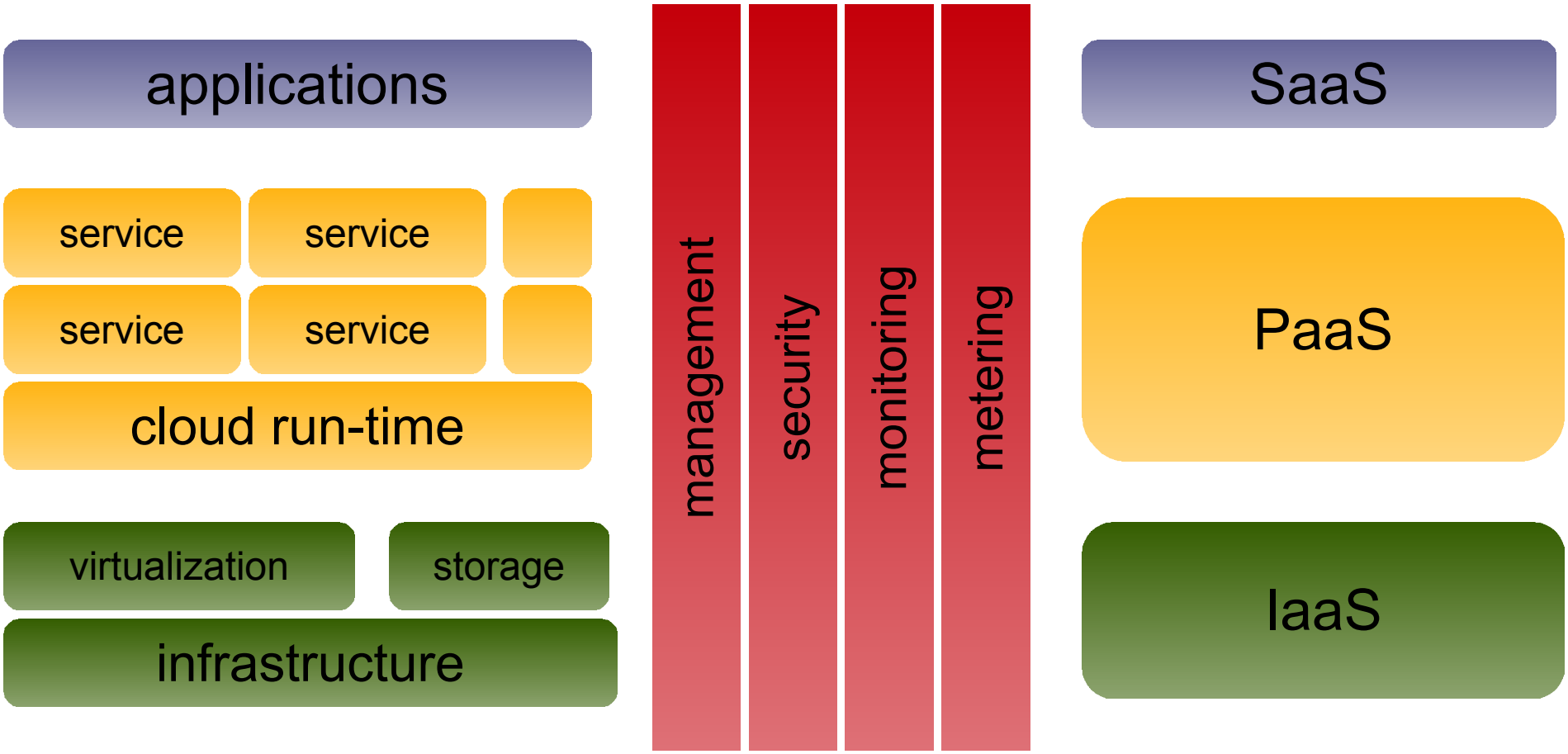
[1] **Visualizing the Boundaries of Control in the Cloud. Dec 2009.**
http://kscottmorrison.com/2009/12/01/visualizing-the-boundaries-of-control-in-the-cloud/

# XaaS

XaaS (Everything-as-a-Service)

- composite second level services
- Security-as-a-Service
  - McAfee*
    - McAfee SaaS Email Archiving
    - McAfee SaaS Email Inbound Filtering
    - McAfee Vulnerability Assessment SaaS (PEN Tests)
- CaaS – Communication-as-a-Service
  - VoIP, private PBX

# Reference Architecture



# NIST Cloud Deployment Models

## Private cloud

- infrastructure is operated solely for an organization
- managed by the organization or by a third party

## Community cloud

- supports a specific community
- infrastructure is shared by several organizations

# NIST Cloud Deployment Models

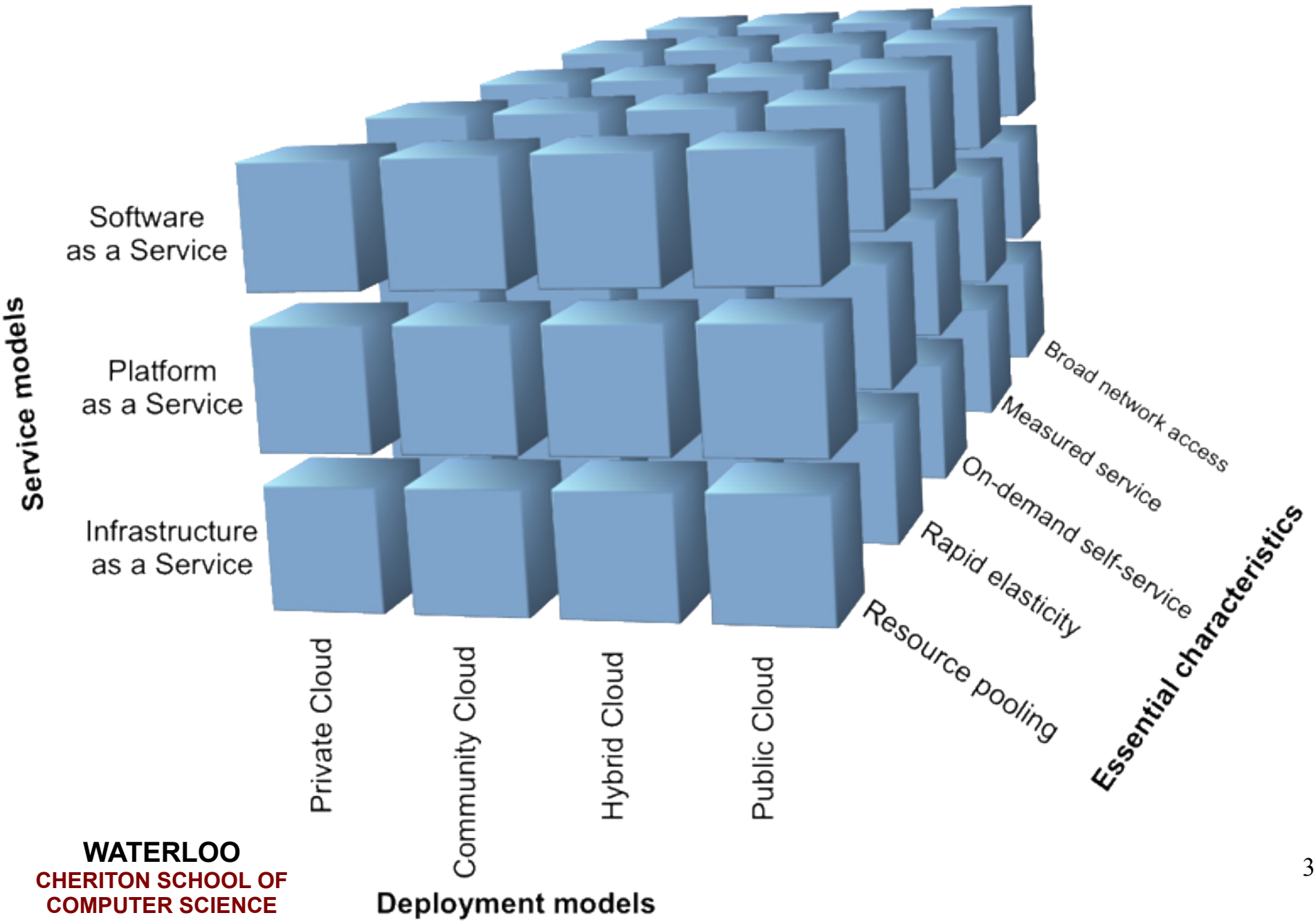
## Public cloud

- infrastructure is made available to the general public
- owned by an organization selling cloud services

## Hybrid cloud

- infrastructure is a composition of two or more clouds deployment models
- enables data and application portability

# NIST Cloud Computing Model




# Distributed Storage & Clouds

# Distributed Storage

## CAP Theorem*

- web services cannot ensure all three of the following properties at once
  - consistency
    - set of operations has occurred all at once
  - availability
    - an operation must terminate in an intended response
  - partition tolerance
    - operations will complete, even if individual components are unavailable



* Eric Brewer, University of California, Berkeley

# Distributed Storage

## Horizontal Storage Scaling

- *"any horizontal scaling strategy is based on data partitioning"**
  - forced to decide between consistency & availability

## Recall

- ACID provides strong data consistency guarantees at the **cost of availability**

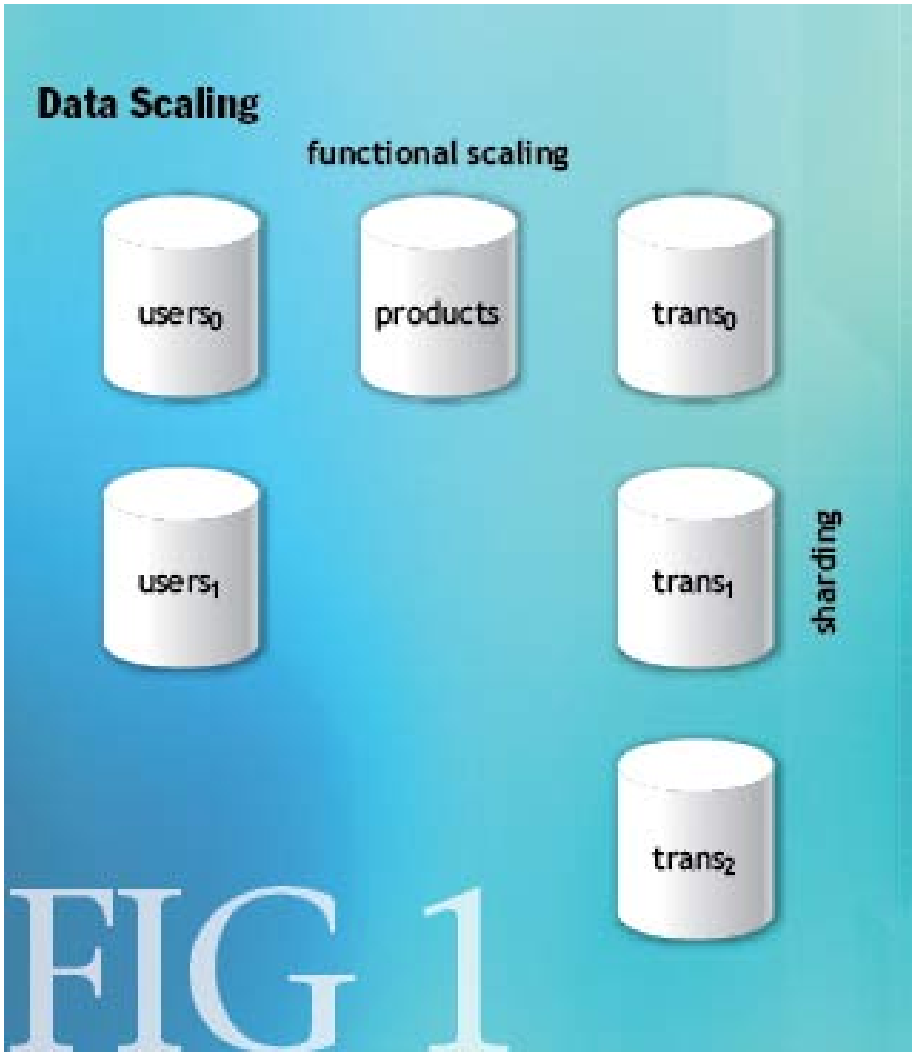
* BASE: An Acid Alternative, Dan Pritchett
http://queue.acm.org/detail.cfm?id=1394128

# Scaling Storage

Two approaches to Scaling

- vertical – bigger hardware

- horizontal – more hardware
  - functional partitioning
  - sharding




http://queue.acm.org/detail.cfm?id=1394128



*http://en.wikipedia.org/wiki/Shard_%28database_architecture%29

# Distributed Storage

BASE – an ACID alternative*

- basically **available**, soft state, **eventual consistency**
- characteristic
  - *"optimistic and accepts that the database consistency will be in a state of flux"**
  - supports partial failures
- scalability promise
  - *"leads to levels of scalability that cannot be obtained with ACID"**
  - at the cost of consistency

* BASE: An Acid Alternative, Dan Pritchett
http://queue.acm.org/detail.cfm?id=1394128

# Distributed Storage

## Eventual Consistency

- consistency across functional groups is easy to relax
- we encounter this on daily basis
- some scenarios
  - update of online user profile
  - online master card payment
  - ATM cheque deposit
- **idempotent operations**
  - permit partial failures

# Distributed Storage

General Characteristics

- simplified data model
- built on distributed storage systems
  - GFS - Google File System
  - HDFS – Hadoop Distributed File System
- highly available
  - relaxed consistency
- fault-tolerant via replication and distributed components

# Distributed Storage

## General Characteristics

- eventual consistency
  - all replicas will be updated at different times and in different order
- examples
  - Google BigTable
  - Yahoo PNUTS
  - Amazon S3

# Google *Bigtable**

## Introduction

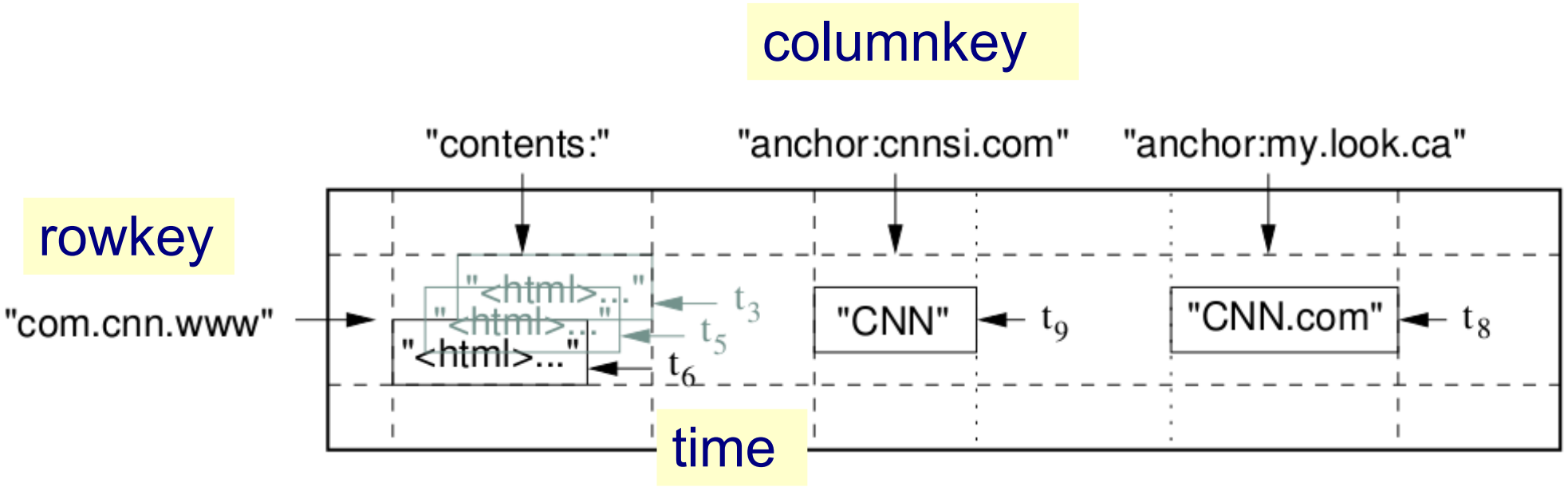
- distributed storage for managing structured data
- designed to
  - scale to very large size
  - store different types of data (URLs, images)
  - achieve high availability, low latency & fault-tolerence

*Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In Proc. USENIX Symposium on Operating System Design and Implementation (OSDI'06), 2006

# Google *Bigtable*

## Data Model

- multi-dimensional sorted map (**not** a relational database)
- (*rowkey*, *column key*, *timestamp*) → byte[]




- row name is reversed URL
- contents column has three versions
- anchor column has one version each

# Google *Bigtable*

Observations

- rows are maintained in lexicographic order
  - data locality → access latency
  - rowkey must be unique across all
- column family has to be defined first
- **unbounded number of columns** can be added to each column family
- time indexing allows for
  - historical versions of the data

# Google *Bigtable*

Scalability, Availability & Fault-tolerance

- master-slave type architecture
- master sever
  - manages data distribution to different slave nodes
  - monitors the life-cycle of slave nodes
  - slave nodes can be added or removed dynamically
- client data utilization
  - connects to the master to get slave node addresses
  - connect directly to the slave nodes to manage data


# Distributed Computation in Clouds (Map Reduce)

# Distributed Computation

Motivation

- facilitate computation over a large data-set
- fault-tolerance
  - single computations can fail
- redundancy
  - same computation can be performed by different nodes
- easy management & configuration setup
  - shield the developers from the details

# Distributed Computation

Basic Idea

- functional programming
- functional decomposition
  - large problem broken into a set of small problems
- design
  - functional transformation of input data → pipes & filters
  - isolated execution → parallel computing
- server (task) farm to solve the big problem
  - cloud elasticity

# Distributed Computation

*wc* the cloud (class activity)

- design considerations
  - large data-sets
  - parallel execution
  - fault-tolerance
  - scalability
  - reusability

# Distributed Computation

*wc* the cloud with MapReduce

But a word about MapReduce first

# *MapReduce**

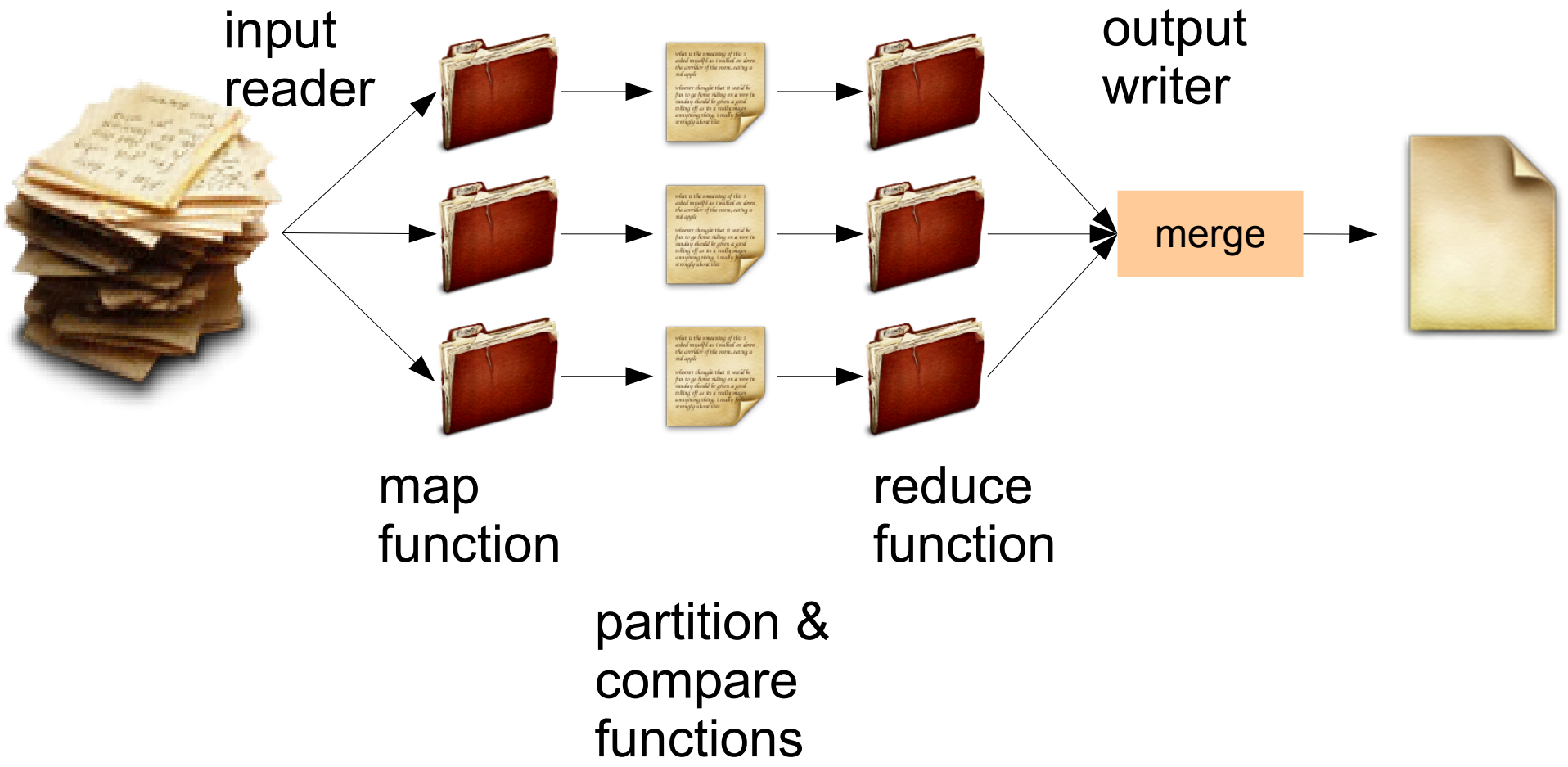
Programming Model

- input: a set of key value paris
- output: a set of key value pairs
- computation: transform input set into output set
  - map function (user defined)
  - reduce function (user defined)

*MapReduce: Simplified Data Processing on Large Clusters. Jeffrey Dean and Sanjay Ghemawat. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004. http://labs.google.com/papers/mapreduce.html

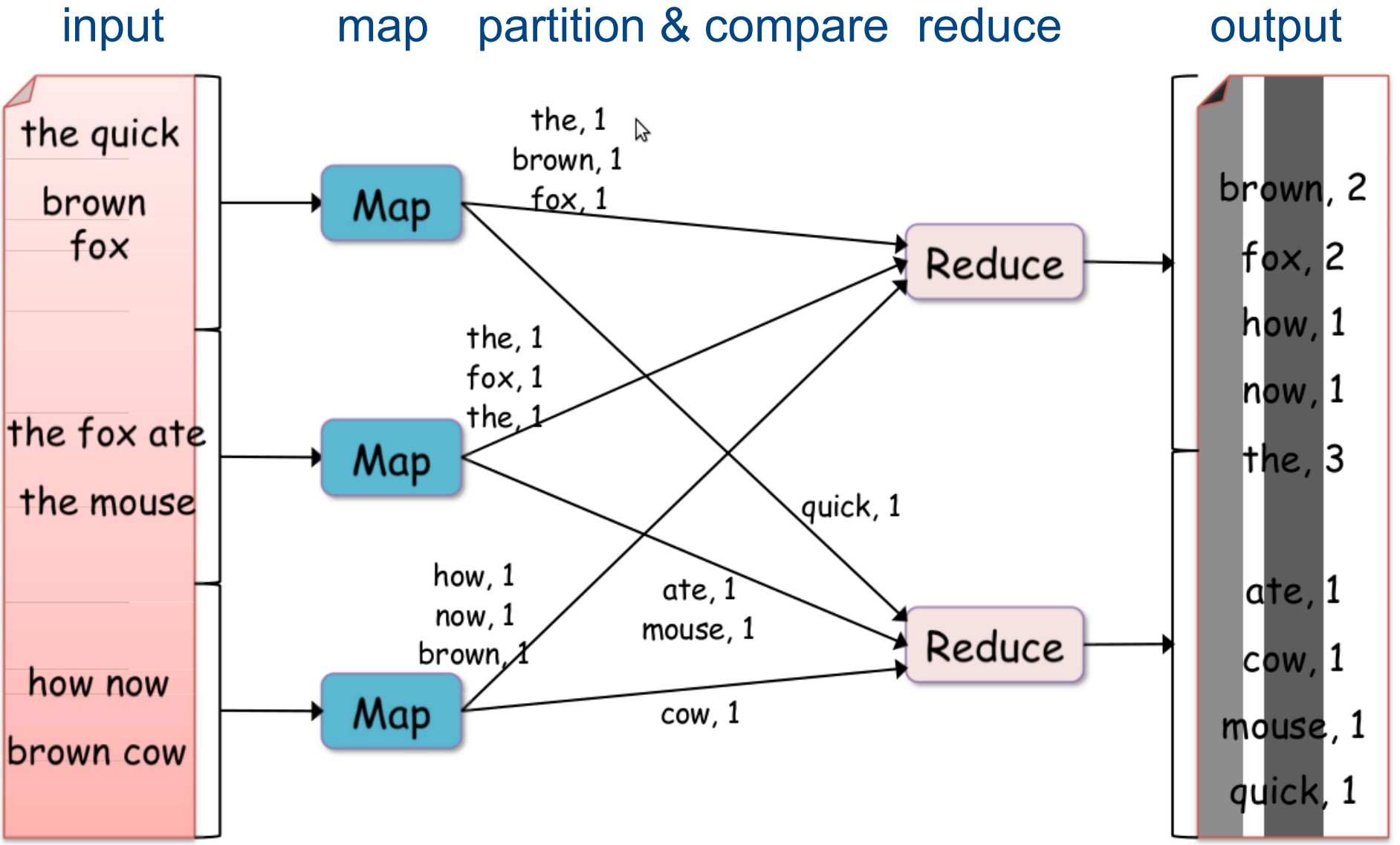
# MapReduce

## Control Flow



# Distributed Computation

*wc* the cloud with MapReduce



# Distributed Computation

*wc* the cloud with MapReduce

```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
EmitIntermediate(w, "1");
```

*MapReduce: Simplified Data Processing on Large Clusters. Jeffrey Dean and Sanjay Ghemawat. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004. http://labs.google.com/papers/mapreduce.html

# Distributed Computation

*wc* the cloud with MapReduce

```
reduce(String key, Iterator values):
// key: a word
// values: a list of counts
int result = 0;
for each v in values:
result += ParseInt(v);
Emit(AsString(result));
```

*MapReduce: Simplified Data Processing on Large Clusters. Jeffrey Dean and Sanjay Ghemawat. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004. http://labs.google.com/papers/mapreduce.html

# Distributed Computation

Observations

- MapReduce framework
  - **manage**s the job
  - job decomposition into independent functional units
- user provides the data transformation operations
  - map & reduce functions only
- work is distributed over multiple nodes
  - ***what is the performance bottleneck here?***
  - ***how can we improve upon this?***

# More Examples

Distributed Grep

- **map** → *emits a line if it matches a supplied pattern*
- **reduce** → *copies the supplied intermediate data to the output*

URL Access Frequency

- **map** → *processes logs of web page requests and outputs <URL, 1>*
- **reduce** → *adds together all values for the same URL and emits a <URL, total count>*

*MapReduce: Simplified Data Processing on Large Clusters. Jeffrey Dean and Sanjay Ghemawat. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004. http://labs.google.com/papers/mapreduce.html

# More Examples

Inverted Index

- **map** → *parses each document, and emits a sequence of <word, document ID>*
- **reduce** → *for a given word,emits a <word, list(document ID)> pair.*

*MapReduce: Simplified Data Processing on Large Clusters. Jeffrey Dean and Sanjay Ghemawat. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004. http://labs.google.com/papers/mapreduce.html

# Cloud Architecture – Best Practices*

## Scalability

- *"a scalable architecture is critical to take advantage of a scalable infrastructure" **

## Application Scalability Characteristics

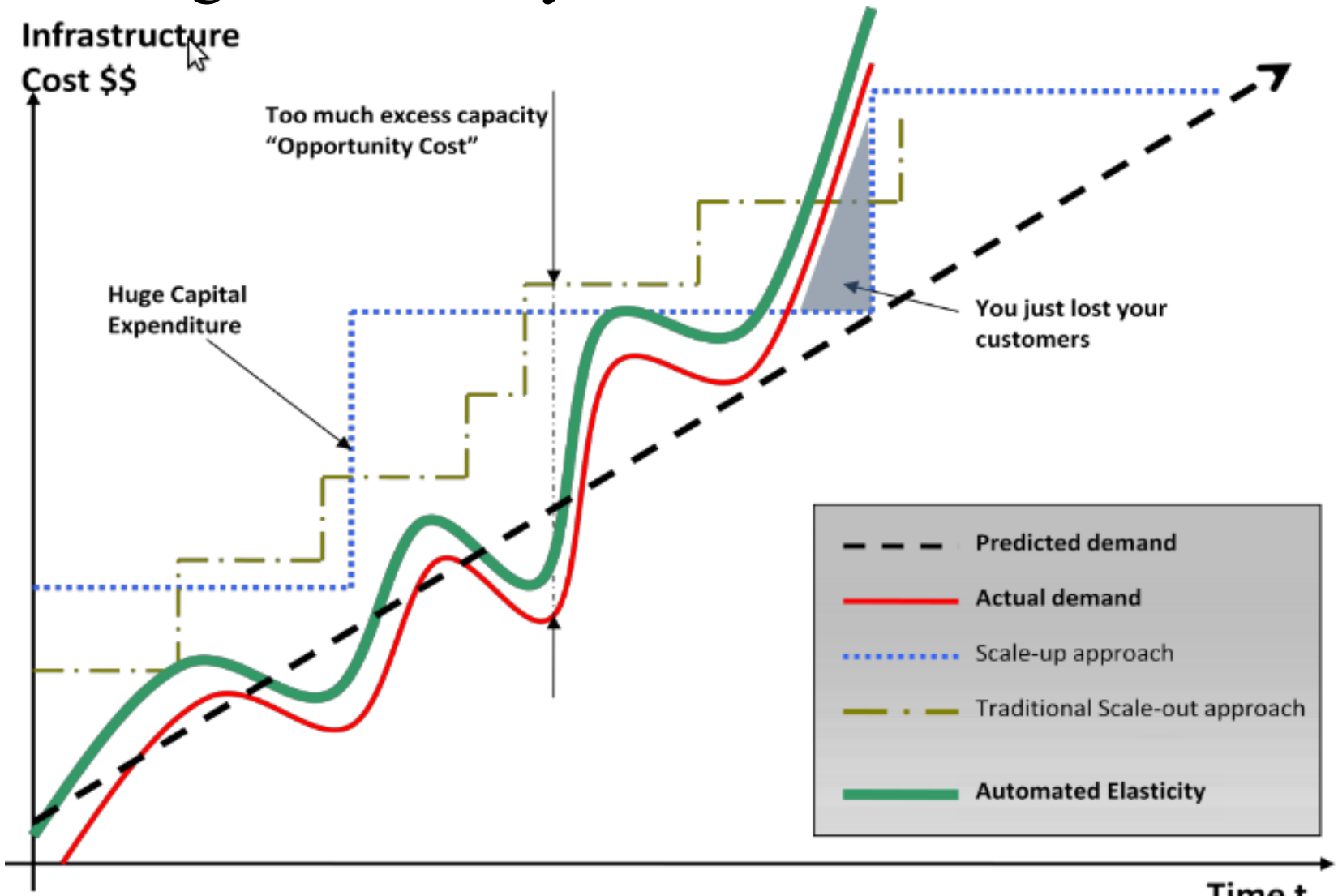
- increase resources results in proportional increase in performance
- scalable service is capable of handling heterogeneity
- scalable service is resilient



*Architecting for the Cloud: Best Practices - by J Varia - 2010

# Cloud Architecture – Best Practices*

## Scaling & Elasticity




*Architecting for the Cloud: Best Practices - by J Varia - 2010

# Cloud Architecture – Best Practices*

Design for failure and nothing will fail

- assume everything will fail and ***design backwards***
  - hardware failure, software failure
  - unexpected increase in system load
- avoid "*single point of failure*"
  - added redundancy and fault-tolerence
- plan for automated failure recognition, reporting, replacement
- *ideas for data failure recovery?*
  - *partitioning, replication, write-logs*



*Architecting for the Cloud: Best Practices - by J Varia - 2010

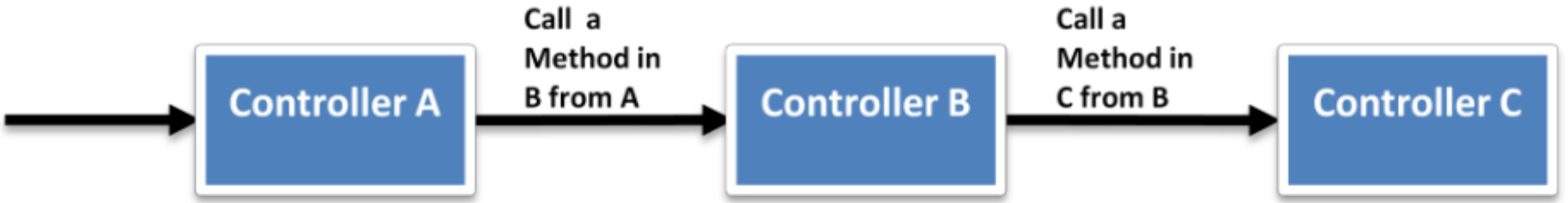
# Cloud Architecture – Best Practices*

## Decouple System Components

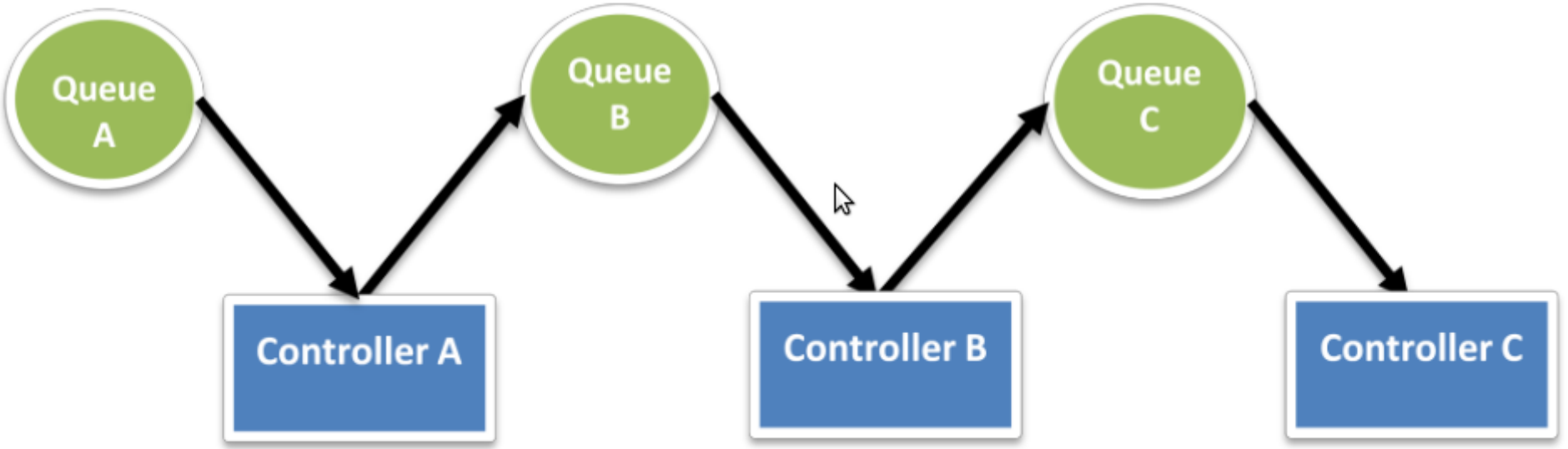
- web based application
  - decouple web server from app server from database server
  - system components to act as black boxes
  - system components to interact via interfaces

# Cloud Architecture – Best Practices*

## Decouple System Components




Tight coupling (procedural programming)




Loose coupling (independent phases using queues)

*Architecting for the Cloud: Best Practices - by J Varia - 2010

# Cloud Architecture – Best Practices*

## Data Partitioning

- dynamic data
  - keep the data in the cloud if possible to avoid network latencies
  - moving computation to the data
- static data:
  - utilize content delivery services to cache data at the edge locations (closer to the client/user)
  - replication & caching

# Cloud Architecture – Best Practices*

Parallel Processing

- "*cloud makes parallelization effortless*"
- implement parallelization wherever **posisble**
- automate parallelization
- ***request parallelization***
  - via ***thread safe*** & ***share nothing*** principles
- examples
  - parallel hardware, data access, & computation