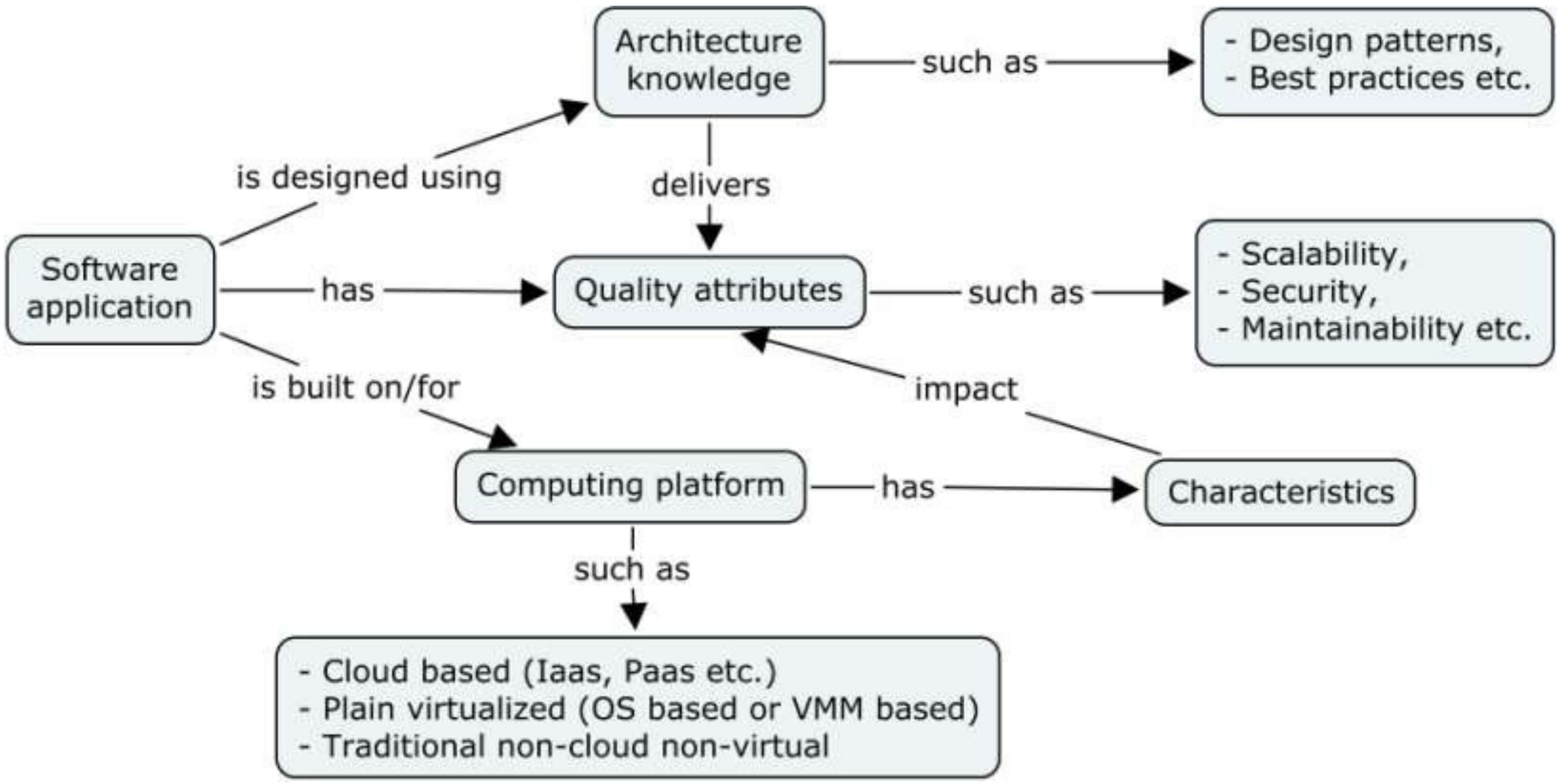
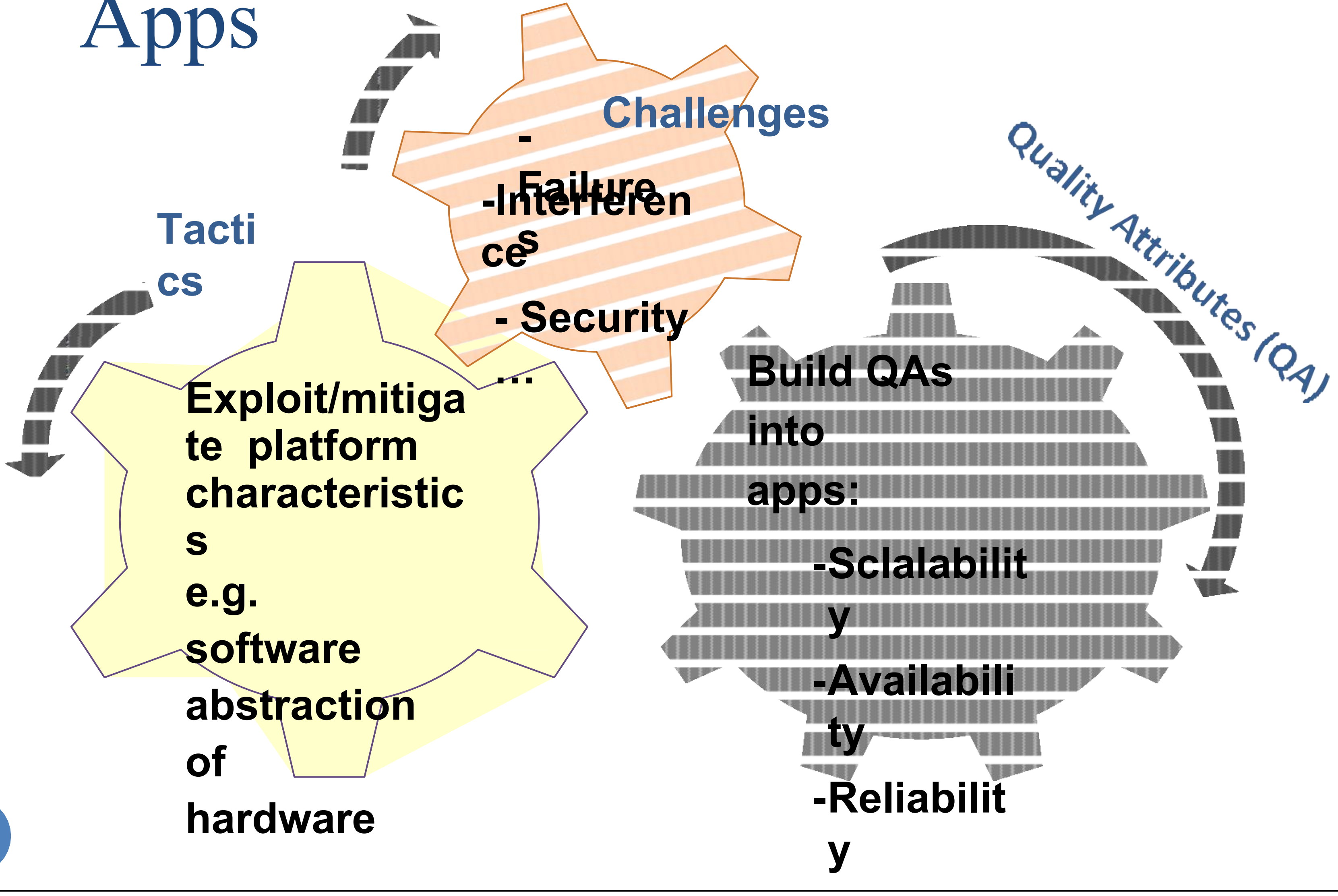
Cloud-specific Architecture Issues

Platforms & Applications Architecture



Building *Quality* Into Cloud Apps



Cloud vs. Non-cloud

- Cloud vendors address some issues faced by non-cloud data centers
  - Provisioning of server resources
  - Physical security
  - Hiring/training data center personnel
- Several other issues still remain the same
  - Network intrusion threats from outside
  - Isolating and managing production/test environments
  - Installation of updates/patches
- Some new or now-changed issues have arisen
  - These are the issues that we bring out

Key Issues Specific To Cloud

- Security/privacy
  - Multi-tenancy
  - Access keys/credentials
  - Dependency on geographic/legal jurisdiction
- Failure
  - Failure of VM instances
  - Data consistency failures
  - Software upgrade error

Key Issues Specific To Cloud

- Performance
  - Network latency
  - How fast can you provision
  - Elasticity Over/under provisioning
  - Performance interference due to VM co-location

Details about Key Issues

Failure Defined

- *A system failure occurs when the delivered service no longer complies with the specifications, the latter being an agreed description of the system's expected function and/or service*
- Related terms:
  - Faults (defects or bugs)
  - Errors (expected and actual behavior differs)
- Faults and errors *may* lead to failure

Failure Rates Get Amplified In Cloud

- Failure rates
  - Servers experience 2-4% annual failure rates (AFR)
  - Disk drives have about 4-6% AFR
- For server AFR of 3% MTBF is about 292000hrs
  - More than 30 years
- In a datacenter having 64000 servers having 2 disks each
  - Daily, more than 5 servers and 15 disks can fail

Handling Failure In Cloud

- Before you handle a failure you must detect it
- *Heartbeat* remains the key tactic for identifying failures
- Must have a *monitor* that watches aliveness of VMs
  - A monitor can be in: infrastructure, client or part of the application
- A VM must periodically show its aliveness to the monitor
  - By responding to some query
  - Sending some message

Stateful vs. Stateless Instances

- Applications can be stateful or stateless
- Stateful applications
  - Remember information across requests
  - State information kept in memory or external
- Stateless applications
  - Require that requests carry necessary information needed for understanding the context
- Failure recovery is different in stateful and stateless cases

Recovery For Stateful Instance

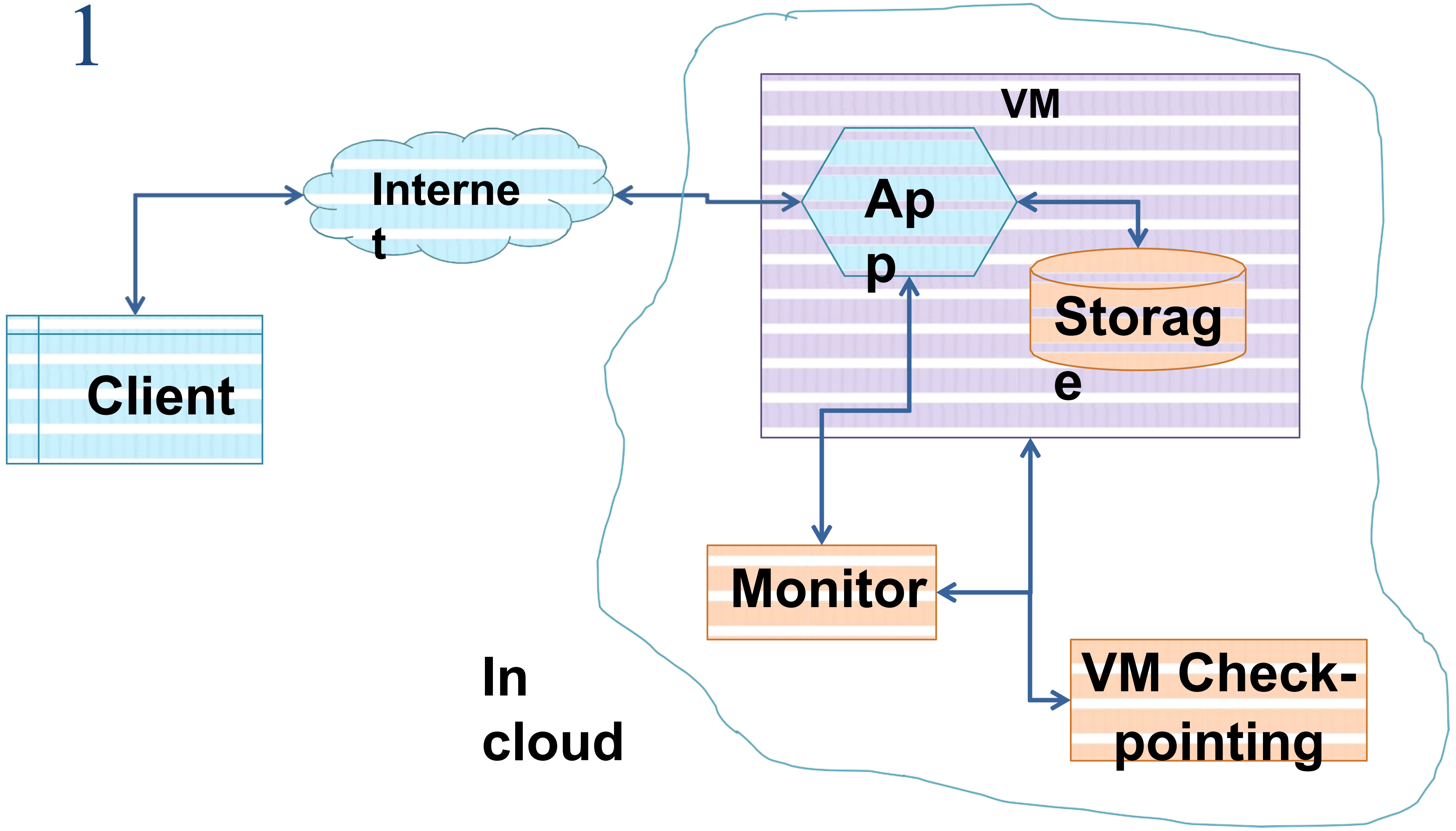
- Multiple tactics are possible depending upon
  - Whether application stored state in the VM itself or on external device
  - Application’s tolerance for loss of computation
  - Availability of VM check-pointing
- Basic tactic is to:
  - Keep saving most recent state data somewhere safely
  - Restore back to last saved state on detecting failure

Stateful Recovery | Tactic 1

- Assumes that application keeps state in VM itself
  - No state dependency on external device
- Works for cloud only
- *Check-point VM state on regular intervals
- On detection of failure, restore and start from last check-pointed VM image
  - Requests arriving between time of failure until restoring to last checkpoint get lost

*** Sodhi & Prabhakar, "Cloud-oriented platforms: Bearing on Application Architecture and Design patterns" in IEEE Services 2012**

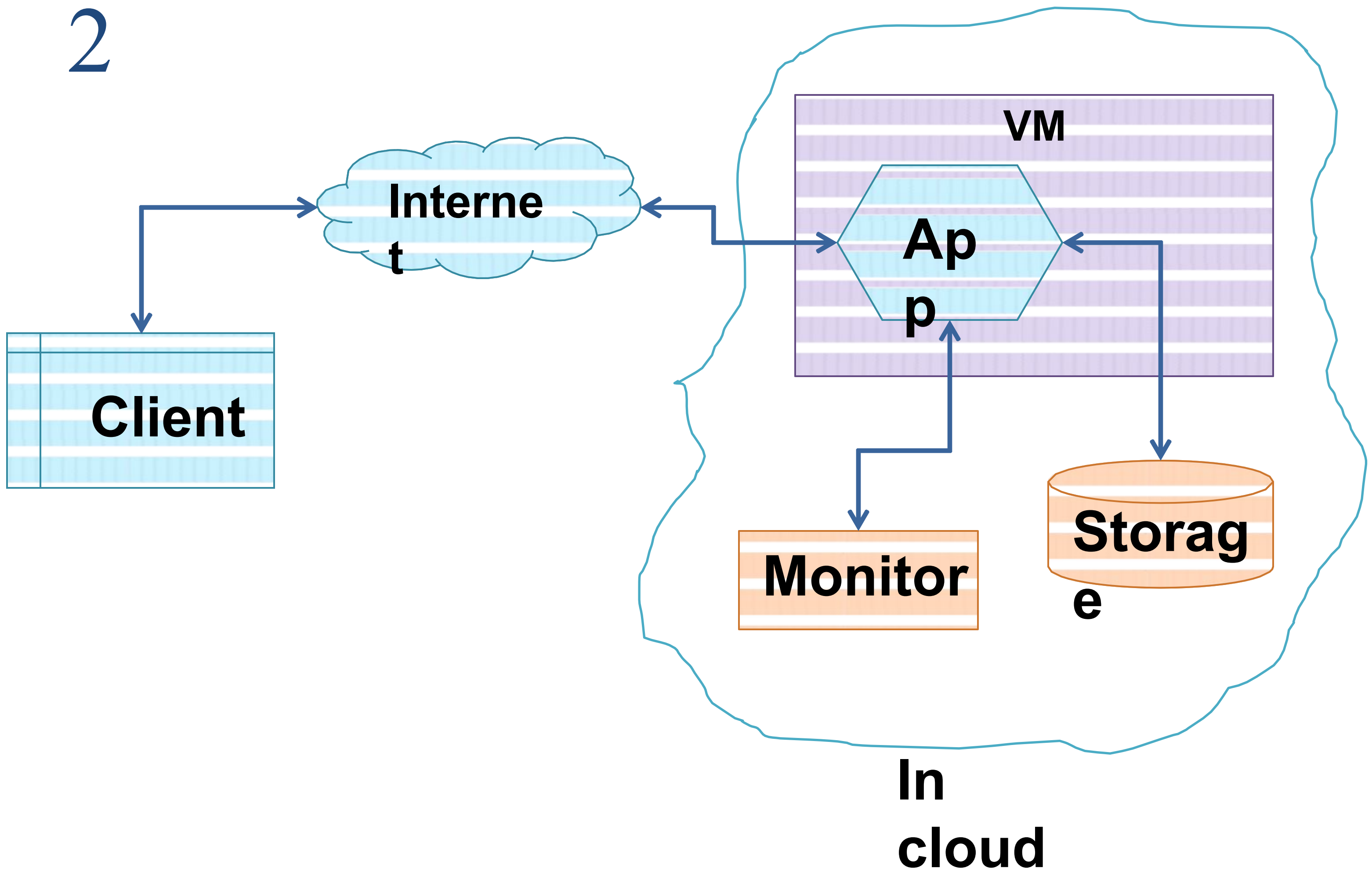
Stateful Recovery | Tactic 1



Stateful Recovery | Tactic 2

- Application saves recent state on external device
  - Done at regular intervals
- On detection of failure:
  - Check for recent state saved on external device
  - Resume from state restored from external device
- Requests arriving between time of failure until restoring to last saved state get lost
- Recovery mechanism needs to be coded into application itself

Stateful Recovery | Tactic 2



Avoiding Lost Requests In Tactic 2

- Application logs incoming request as well to external device
- On detection of failure:
  - Resume from state restored from external device
  - Read and play logged requests that arrived during down time
- No requests are lost if logging to external device and acknowledgement to client is atomic

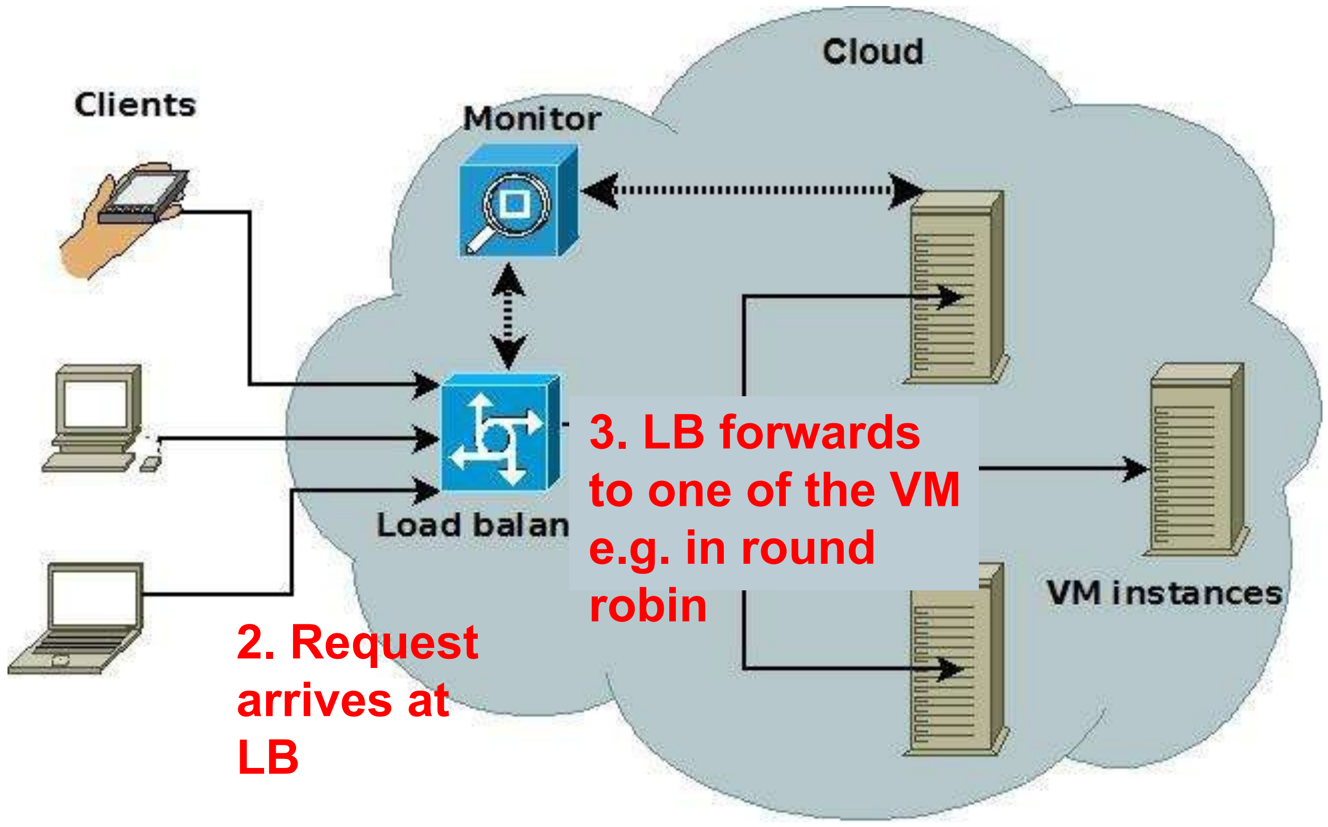
Stateless Applications | Requests Flow

- A request can be received by any instance
- New VMs can be created to meet QoS demands
  - Performance, reliability etc.
- Requests are routed to instances by a load balancer component

Routing Requests To Instances

- Decided by load balancer (LB)
  - LB policies take failures into account
  - LB is augmented by a monitor component
- Tactic has two flavors:
  - Push: LB decides which instance gets to serve a request
  - Pull: Instances pull the requests from a queue maintained by the LB

Architecture Of Push Based LB



3. LB forwards to one of the VM e.g. in round robin

2. Request arrives at LB

1. Client sends request

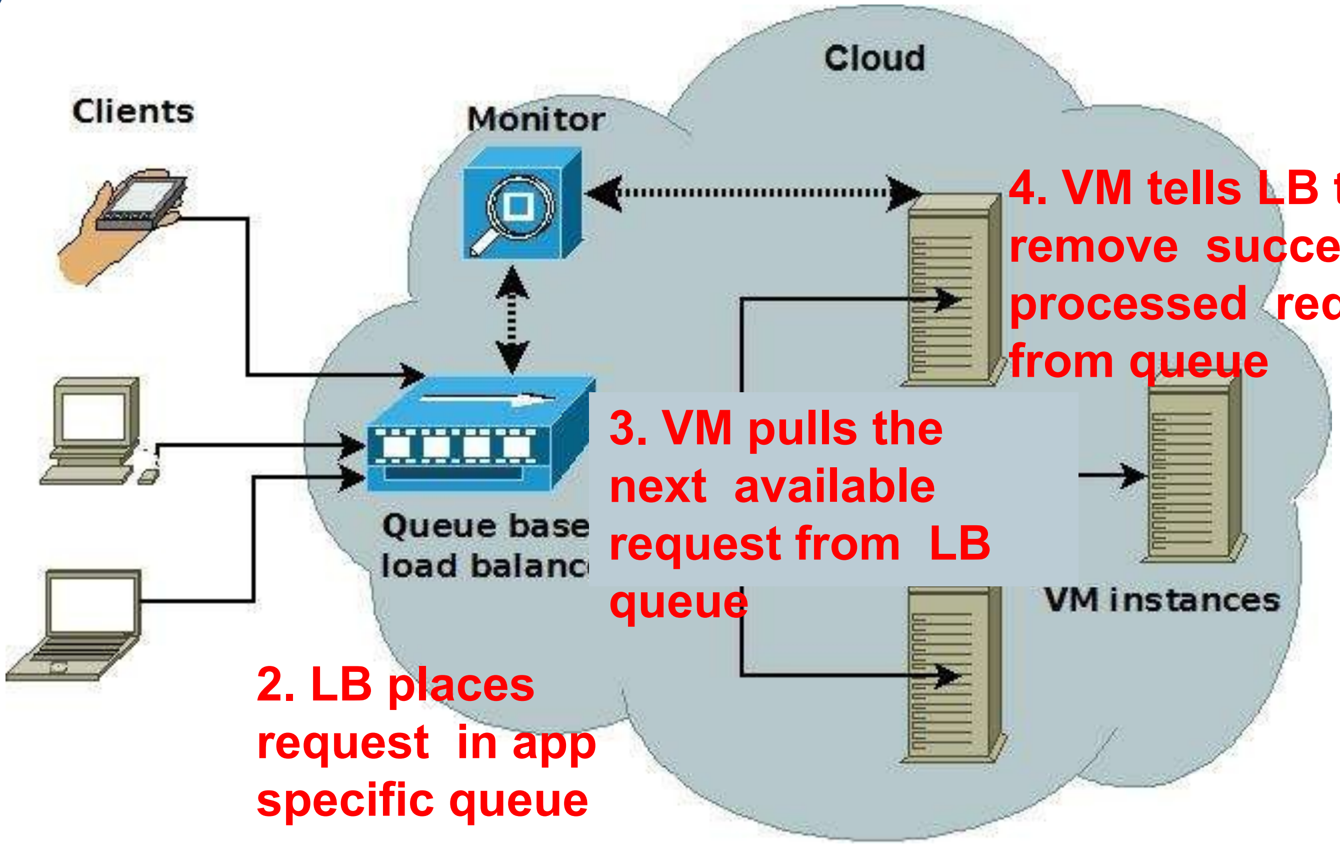
Role Of Monitor

- Observes the VMs
  - Resource (CPU, memory etc.) utilization
  - Requests load
  - Quality of service violations etc.
- Sends VM stats to load balancer
  - Include failures info
- Decides, based on some rules, when more resources are required

Working Of Push LB Pattern

- Monitor detects VM failure
  - E.g. when VM becomes non-responding
- LB gets to know this and stops sending requests to the failed VM
- Current in-progress requests handled by VM are lost
  - Client needs to detect this possibly via timeout
  - Client should resend the lost requests

Architecture Of Pull Based LB



4. VM tells LB to remove successfully processed request from queue

3. VM pulls the next available request from LB queue

2. LB places request in app specific queue

1. Client sends request

Role Of Monitor

- Watch the application specific queues on LB
  - Waiting time for requests in a queue
  - Length of the queue
- Infer load on a VM from queue stats
- Decides, based on some rules, when more resources are required

Working Of Pull LB Pattern

- LB knows when a request has been processed by a VM
- Requests that remain unhandled until a time limit get reassigned by LB
- A failed VM won't pick requests from it queue
  - This automatically takes a failed VM out of service
- Requests trapped in failed VMs
  - Can get processed when VM recovers
  - Application must handle duplicate processing scenario

VM Cleanup On IaaS Cloud

- When a VM fails:
  - It is not automatically de-allocated
  - It needs to be de-allocated by consumer
  - Cloud provider continue to charge until VM is de- allocated
- On de-allocation of a VM:
  - Its public and private IP addresses become available for reassignment
  - Infrastructure can be told to assign released public IP address to replacement VM

Summary

- LB and monitor are key components
  - LB policies take failures into account
  - LB is augmented by a monitor component
- Tactic has two flavors:
  - Push: LB decides which instance gets to serve a request
  - Pull: Instances pull the requests from a queue maintained by the LB

Data storage issues on cloud

Data Storage

- Provides ability to persists digitized information
  - e.g. plain text files, binary files containing photos
- Retains data for an interval of time
  - Length of time depends on storage type
    - e.g. RAM contents don’t survive machine restart, whereas hard disk contents can

Data Storage Components

- Can be have different categories, for example:
  - Raw files on a file systems (FS) on the operating system
  - Data stores such as RDBMS engines, key-value stores etc.
- Characteristics/behavior of each category can vary significantly
- An application can use multiple types of data storage components
  - E.g. can write to both the raw FS and a database table

Data Storage And Cloud

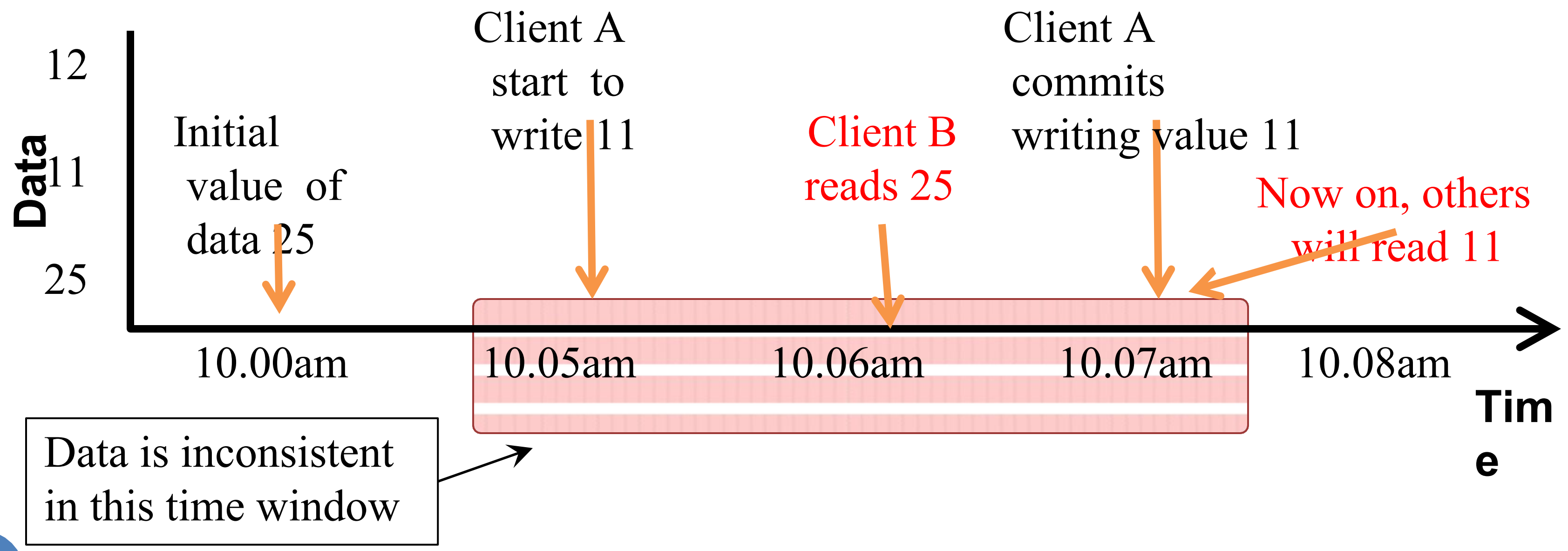
- Mainly we are concerned about data storage on:
  - IaaS cloud
  - PaaS cloud
- IaaS cloud because:
  - Cloud user has to manage the resources including storage
- PaaS cloud because:
  - As a developer you need to handle data storage from within the applications you write
- SaaS cloud case is not interesting
  - Because as a user you don't write any software here

Data Storage On Cloud

- IaaS cloud vendors offer two data storage types:
  - Ephemeral
  - Persistent
- Ephemeral storage doesn't survive instances failure
  - Typically available as a block device attached to the VM
    instance
- Persistent storage is long lived
  - Cloud vendor automatically replicates
  - Geographically distributed replicas
- Storage failures can lead to data consistency issues
  - Application needs to be prepared for this

Data Consistency In Applications

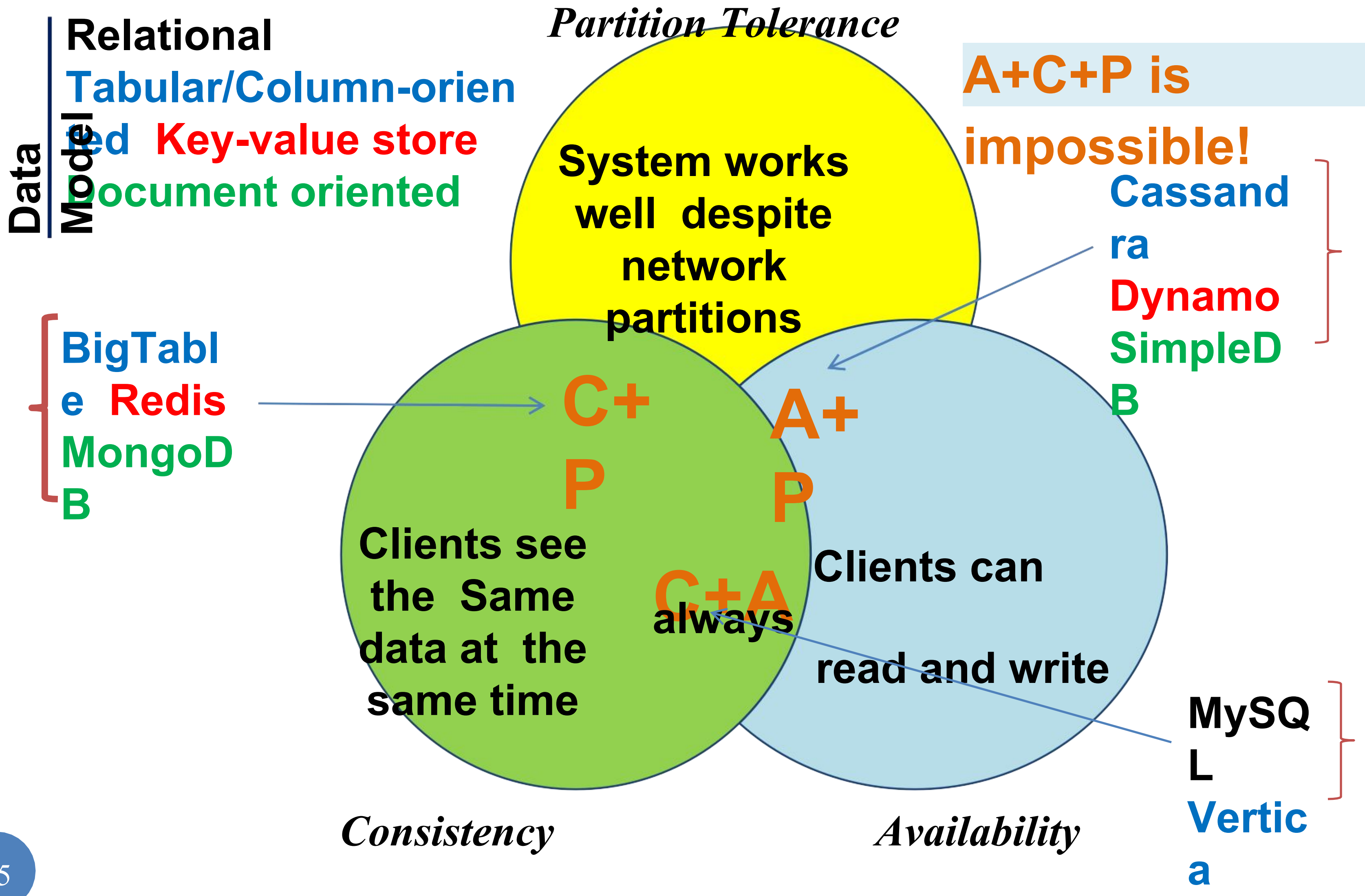
- Consistency
  - Disallow multiple values of same piece of data when seen by different clients at the same point in time


Data
12
11
25
Initial value of data 25
Client A start to write 11
Client B reads 25
Client A commits writing value 11
Now on, others will read 11
10.00am
10.05am
10.06am
10.07am
10.08am
Time
Data is inconsistent in this time window

Data Partition Tolerance

- Partitioning of data means:
  - Placing parts or copies of data on multiple nodes
    - A “node” here may mean a physical machine or just a different database server instance
  - Often needed to achieve massive scalability
- Tolerance for data partitioning means:
  - Application should not respond incorrectly even in the presence of data partitioning
  - i.e. data *consistency* is maintained in presence of partitions

Data Consistency | CAP Theorem



Data Consistency In Cloud

- Replication is a key mechanism
- Replicating data is time consuming
  - Replicas may not be in sync immediately
  - Clients may see inconsistent data for a tiny duration
- Typically, cloud vendors offer “consistent reads”
  - But there may be latency issues
- E.g. AWS S3 today offers read-after-write consistency for PUTs of new objects
  - But gives only eventual consistency for overwrite PUTS and DELETES

Why Is It Important?

- Because users needs to be happy
  - “500 Internal Server Error” is the last thing I want to see after punching in my credit card details
    - Next time I’ll shop elsewhere
- It impacts* the businesses too
  - Extra 0.1s in response time costs Amazon 1% in sales
  - Google found that 0.5s jump in latency leads to 20% drop in traffic

* http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it

How To Address It

- You need to choose any two from among:
  - Availability, consistency and partition tolerance
- Drop availability
  - Services are unavailable until data is consistent on all nodes
- Drop partition tolerance
  - Avoid partitions from happening
- Drop consistency
  - Expect that data becomes consistent eventually

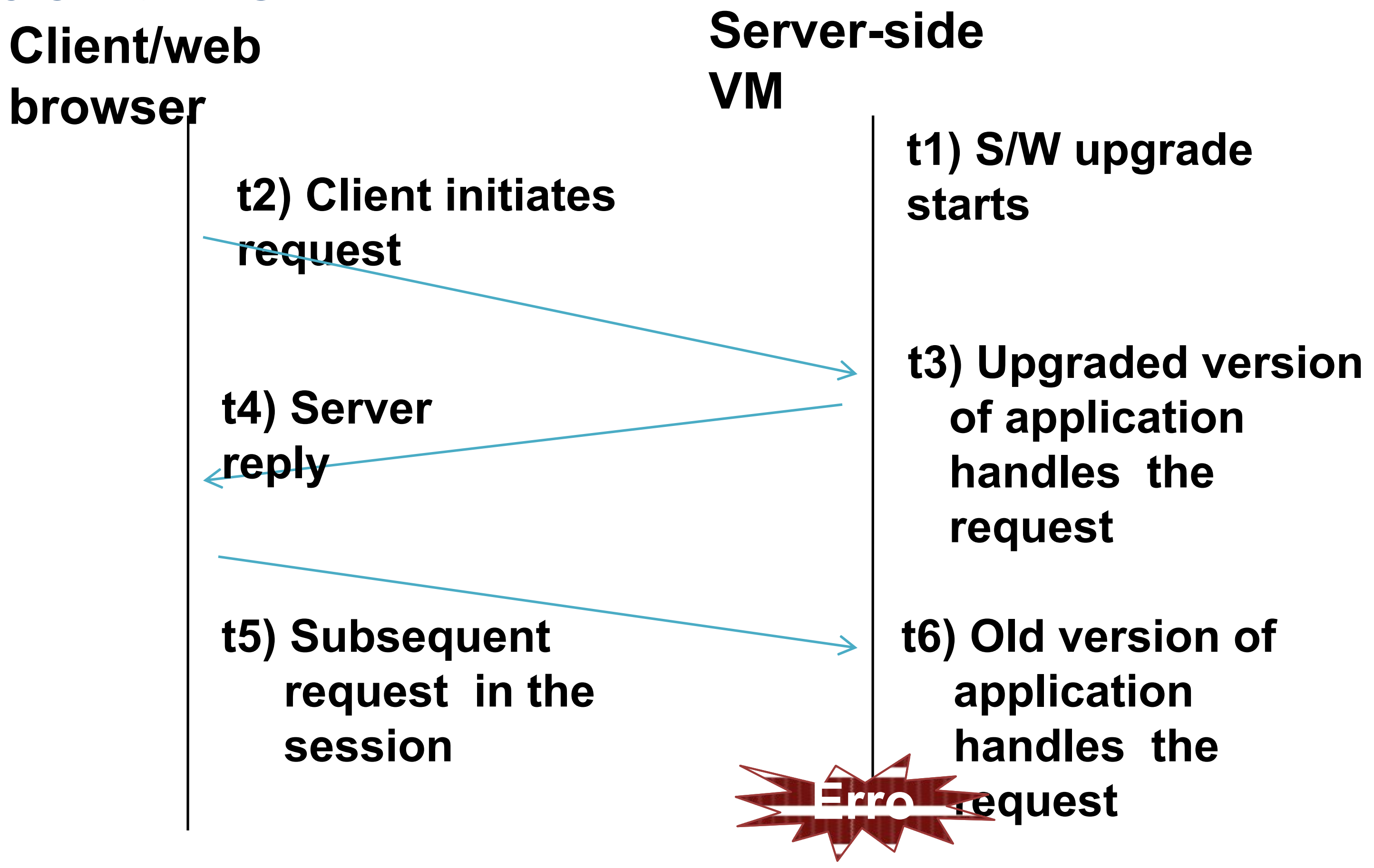
Summary

- It is important to understand characteristics of data storage mechanisms on cloud
  - Different storage services may behave differently
- Data consistency in applications
  - Cannot achieve all three at the same time:
    - Consistency of data
    - Availability of data/services
    - Partition tolerance

Software Upgrade Induced Failures

- Upgrading and patching software is common
- Main reasons for upgrades and patching
  - Releasing new features
  - Fixing defects
- Users may experience failures during such upgrades and patching of software applications
  - How to avoid such failures?

An Example Scenario


Client/web browser
Server-side VM
t1) S/W upgrade starts
t2) Client initiates request
t3) Upgraded version of application handles the request
t4) Server reply
t5) Subsequent request in the session
t6) Old version of application handles the request
Erro

Handling S/W Upgrade Failures

- What do we need from the solution
  - Clients should always interact with latest application version on server side
  - Requests should get load balanced regardless of active application versions on server side
- We need to make some assumptions:
  - Backward compatibility of application versions
  - Clients can be dynamically told about latest application version number available

A Solution Idea

- Treat each application version as a separate destination for requests
- Client knows the latest version number that it has interacted with
- Load balancer routes requests based on version number present in request header
  - A version xxx in the header is routed to instance whose version is ≥ xxx
  - Route normally if no version number found in request header

Security and privacy

Information Security

- "Information security means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, perusal, inspection, recording or destruction"

  (From 44 US Code § 3542)
  http://www.law.cornell.edu/uscode/text/44/3542

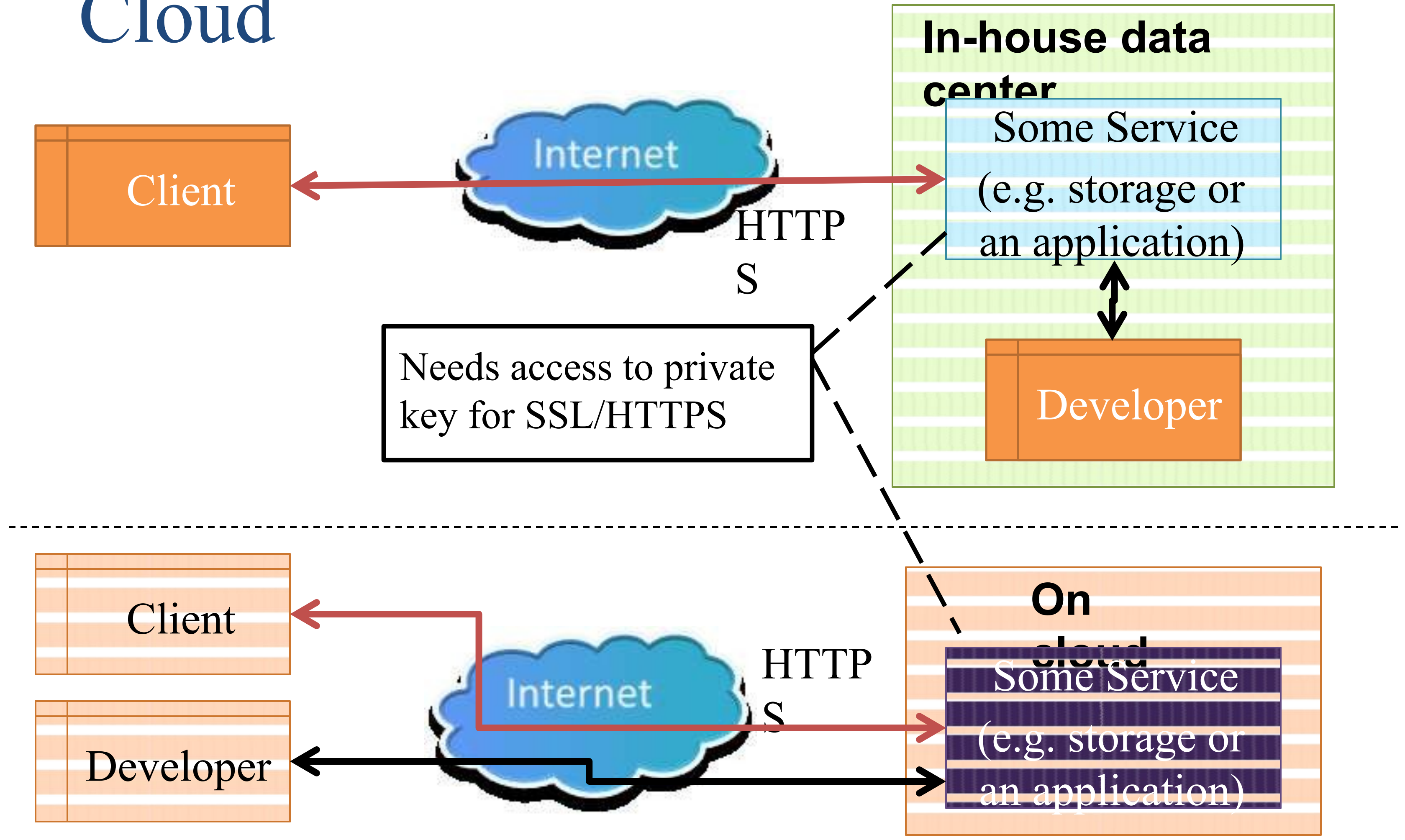
Core Artifacts: Credentials and Keys

- Credentials in IT are typically, a combination of login ID and password
  - Identify the user holding them
- Mainly credentials are used for controlling access to IT resources
  - Authentication: you are who you say you are
  - Authorization: you have the rights to perform certain actions
  - Non-repudiation: you cannot deny you did something
- A key is a number used in cryptography for
  - Encrypting/decrypting data
  - Digital credentials

Some Common Security Measures

- Encrypting sensitive data
- Checking for buffer overruns
- Input validation
- Role based access to data and functionality
- Transaction monitoring and auditing
- Limiting access to servers (close unnecessary ports)
- etc.

Security Scenario | In-house vs. Cloud


In-house data center
Client
Internet
HTTPS
Some Service (e.g. storage or an application)
Developer
Needs access to private key for SSL/HTTPS
On cloud
Client
Developer
Internet
HTTPS
Some Service (e.g. storage or an application)

Important Security Aspects On Cloud

- Central issue:
  - Data and applications being in 3rd party (cloud vendor) custody
  - Lack of trust on the cloud service provider
- Access credentials and encryption keys
- Management of keys and credentials
- Privacy and security in multi-tenant environment
- Dependency on legal/geographical jurisdiction
  - Local laws governing a cloud service provider

Bank vs. At-home | A Metaphor

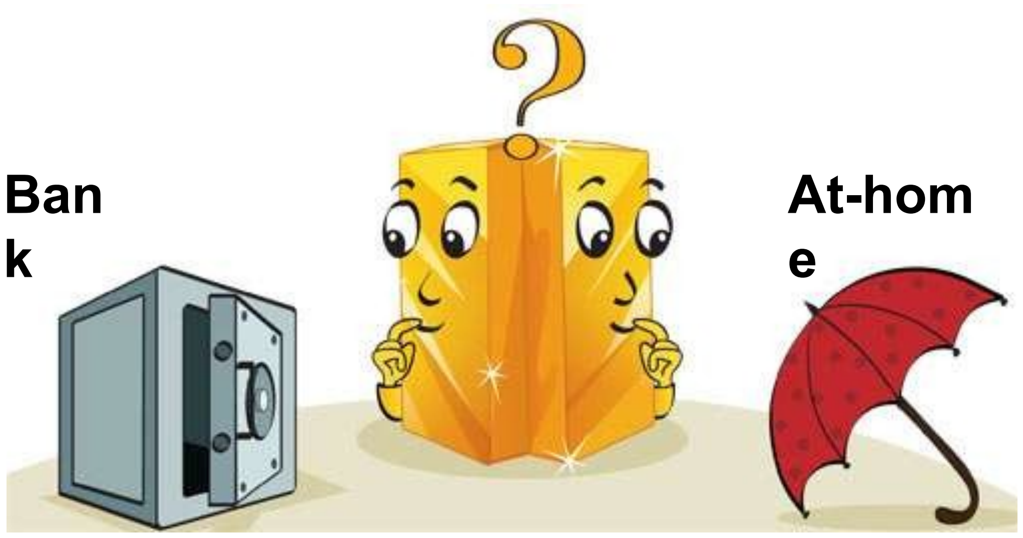
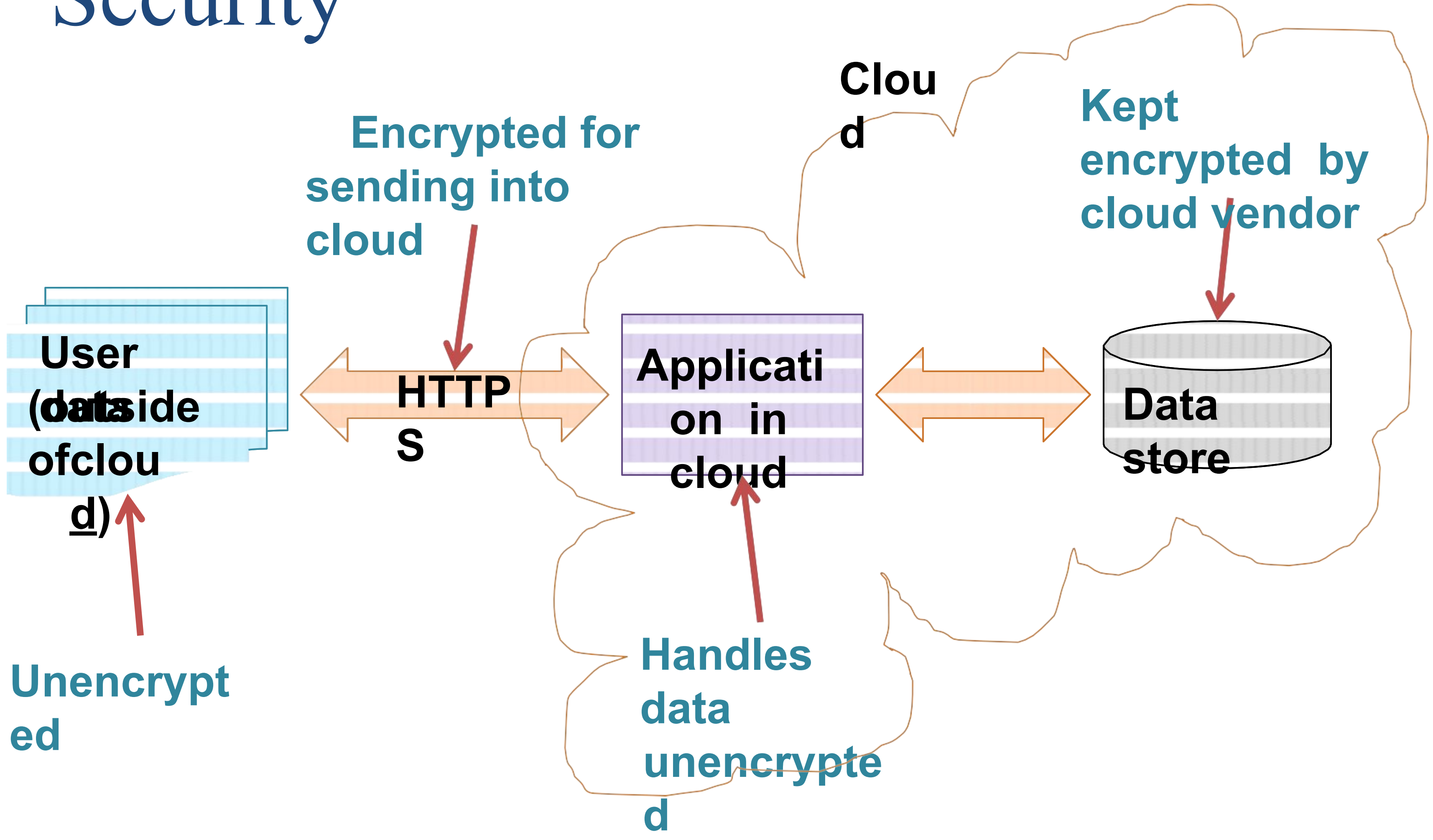


Image credit: Business Today

Elementary Data Security


Cloud
Encrypted for sending into cloud
Kept encrypted by cloud vendor
User data (outside of cloud)
HTTPS
Application in cloud
Data store
Unencrypted
Handles data unencrypted

Issues With Elementary Data Security

- A cloud provider responding to a legal order may:
  - Potentially share your data with govt. agencies
  - Have to provide decrypted data
- To address the above you can encrypt your data before sending to cloud
  - Thus cloud provider can share only encrypted data
- However, if legal orders are directed at you then you need to comply with decrypted data

Credentials Management For Cloud

- Important considerations:
  - Who has/needs access to credentials?
  - Will you need to change credentials? When?
  - Storage and automated provisioning of credentials
- Some options for providing credentials in cloud
  - Build into the VM image
  - Supply as parameter during instance launch
  - Keep in some persistent storage
  - Send from client every time a new instance starts

Summary

- Several security aspects remain same on cloud as they were in-house
  - e.g. from tightening of firewall rules to input validation in you application
- Few things have changed
  - Data/apps now lives in a 3rd party custody
  - It is cloud consumer's responsibility to protect its sensitive data
    - Can use encryption technologies to achieve this

Geo/Legal Jurisdiction Dependency

- Requirements related to storage of personal information (PI)
  - The EU mandates that PI can leave the EU only for jurisdictions that guarantee equivalent privacy
- Some jurisdictions claim access rights on all data stored inside their borders
  - US Patriot Act allows any data stored in US to be examined by US govt.

Impact On Cloud Consumers

- Awareness of cloud vendor data centers location
  - Some do not provide locations of its data centers
- Backup locations may be chosen by the vendor
  - For optimization of its resources etc.
- Abilities to allow users to control data locations
  - Based on type of the data
  - Every vendor may not offer

How To Deal With It?

- Use anonymization techniques for securing PI
  - E.g. replace PI with tokens and keep the PI ↔ token mapping locally
- Example:
  - Original data:
    - Shiva Kumar {Sensitive data}, e.g., bank account number
      - Anonymization tokens (kept locally)
    - Shiva Kumar {Token}, e.g., a number
  - Data stored in cloud:
    - Token
    - Sensitive data
  - Restore original data by taking join of token table and cloud data table

performance

Performance Of A System

- Characterized by quantum of useful work done by a system for a given amount of time and computing resources
- Often measured by
  - Response time (smaller the better)
  - Latency (smaller the better)
  - Throughput (higher the better)
  - Resource utilization (higher the better)

Achieving Better Performance

- System should do:
  - more work
  - at a faster rate
  - by consuming less computing resources
- Time-proven design principles apply on cloud as well
  - Exploit parallelism
  - Pooling of shared resources
  - Put processing near the data/resources it needs
  - Minimizing round trips

… and rest of the good stuff

Key Points For Cloud

- Consolidation of computing resources
  - Improves overall utilization in a data center
  - Use virtualization to achieve this
- Rapid elasticity
  - On-demand provisioning of resources
  - Fast scaling of applications
  - Latencies can still be an issue
- Multi-tenancy
  - May result in performance interference

Elasticity On Cloud Platforms

- Elasticity: provision computing resources on demand
- Consumers can define auto-scaling rules
  - E.g. on an existing VM when CPU usage remains above 80% for 10 minutes, launch a new VM
  - Eliminates manual intervention for provisioning
- Auto-scaling strategies
  - A matter of research
  - Provisioning a VM and starting the apps on it takes time

Provisioning Latency: An Important Issue

- Small Instance
  - 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit),
    1.7 GB of memory, 160 GB of instance storage,
    32-bit platform with a base install of CentOS 5.3
    AMI
  - Can take 5 to 6 minutes us-east-1c from launch to availability
- Large Instance
  - 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each), 7.5 GB of memory, 850 GB of instance storage, 64-bit platform with a base install of CentOS 5.3 AMI
  - Can take 11 to 18 minutes us-east-1c

[http://www.philchen.com/2009/04/21/how-long-does-it-take-to-launch-an-amazon-ec2-instance]

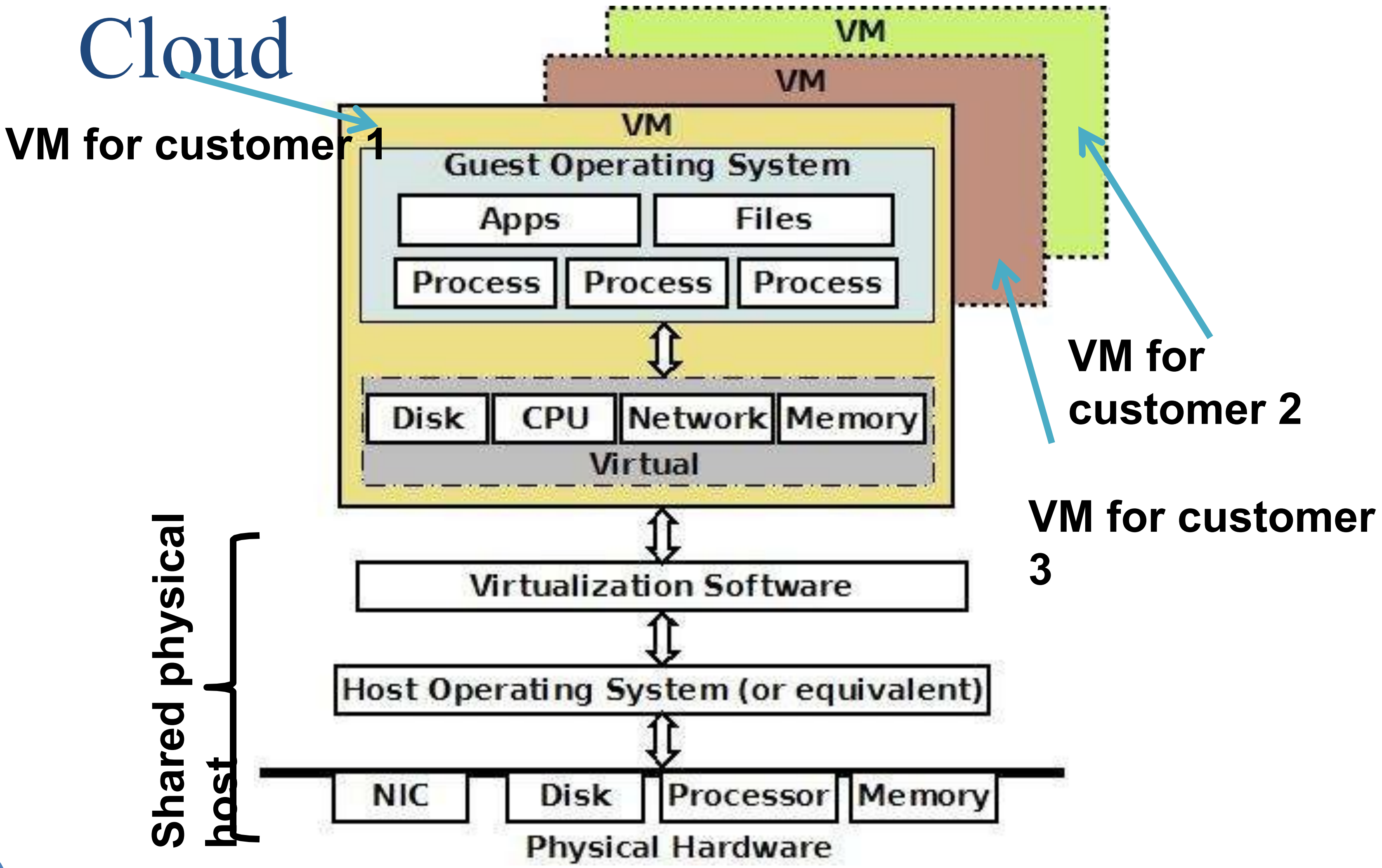
Addressing Provisioning Latency Issue

- You need to forecast the number of resources needed for optimal service
- Basic idea is to:
  - Watch for events that may indicate potential surge in load
  - Calculate the amount of additional resources needed to handle load
- Success depends on:
  - How well you pick right events that indicate load surge
  - Accuracy of calculating needed resources

Multi-tenancy On Cloud

- Different consumer’s applications and data hosted on shared infrastructure
  - e.g. Single physical disk holding data from different consumers
- Needed for optimizing resource utilization
  - Allows cloud provider to leverage economies of scale
- Consumers can assume they are sandboxed
  - i.e. their apps and data is isolated from other’s

Multi-tenancy On Cloud

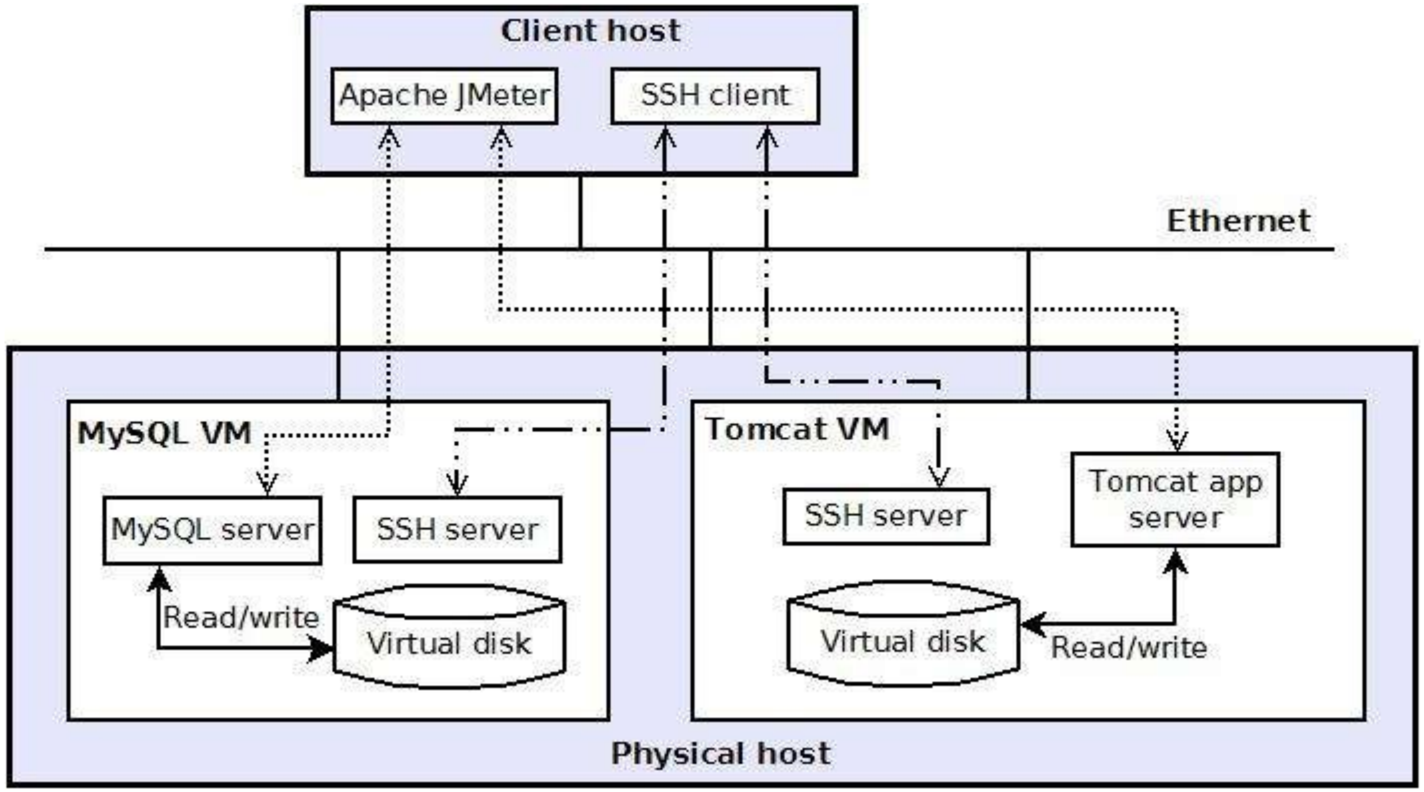


Issues Arising Due To Multi-tenancy

- Performance interference
  - VMs co-located on same physical host may affect each other's performance
  - E.g. simply forcing expensive cache invalidation on the host CPU
- Potential for one VM breaking into another
  - Bugs in virtualization software
  - Encrypting data by user can help

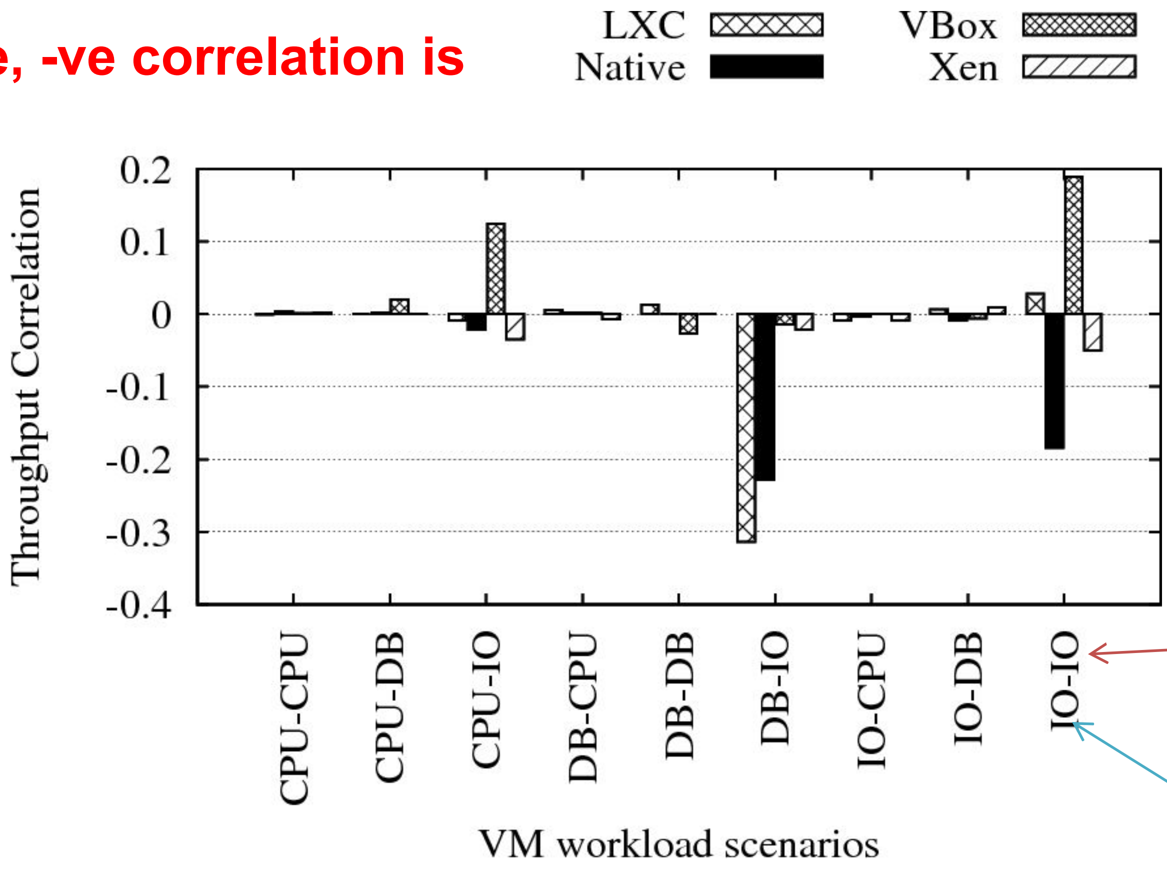
Sodhi & Prabhakar, "Cloud Platforms: Impact on Guest Application Quality Attributes" in IEEE APSCC 2012

Measuring Interference



Performance Interference (Throughput)

Here, -ve correlation is bad!



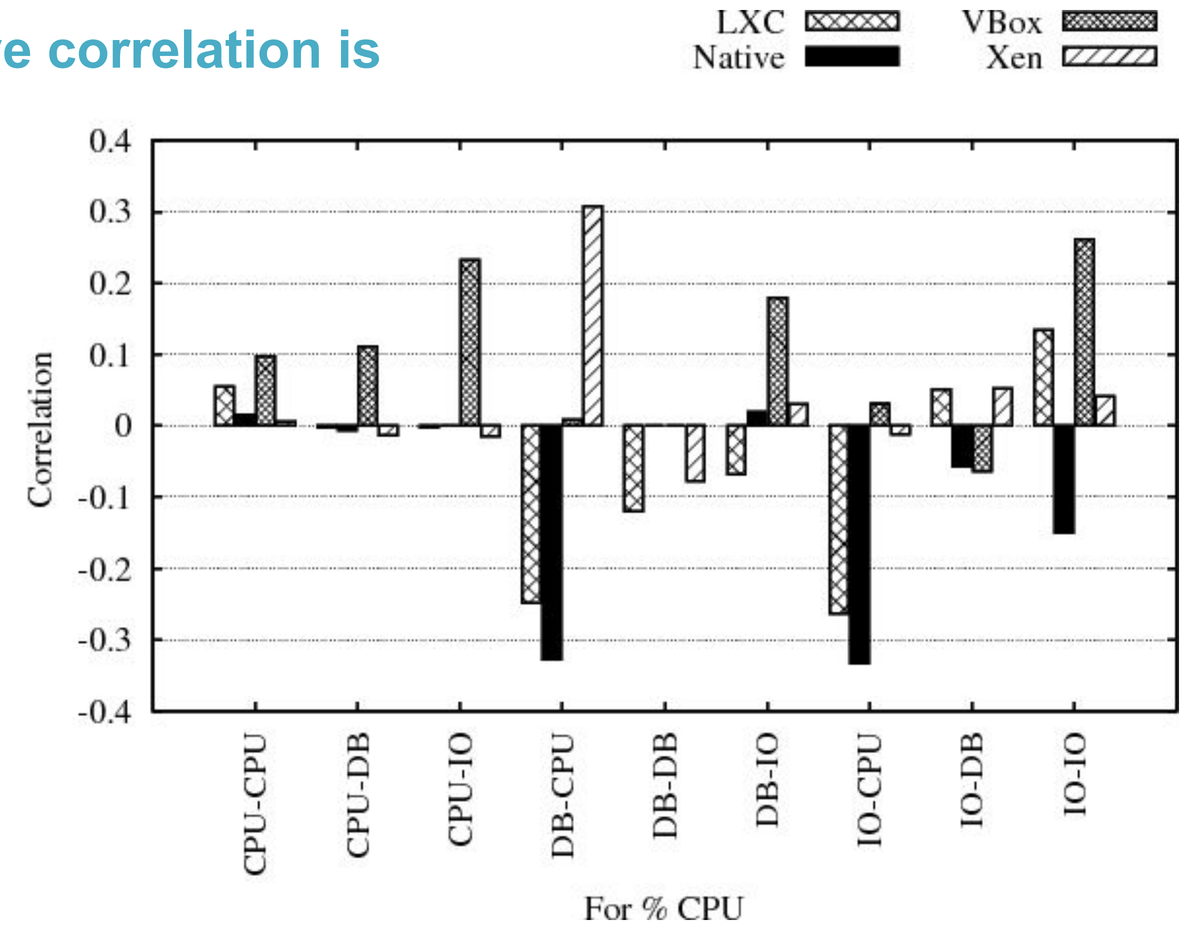
Increasing Load type

Steady Load type

Sodhi & Prabhakar, "Performance Characteristics of Virtualized Platforms from Applications Perspective" in GLOBE 2012 (Springer LNCS)

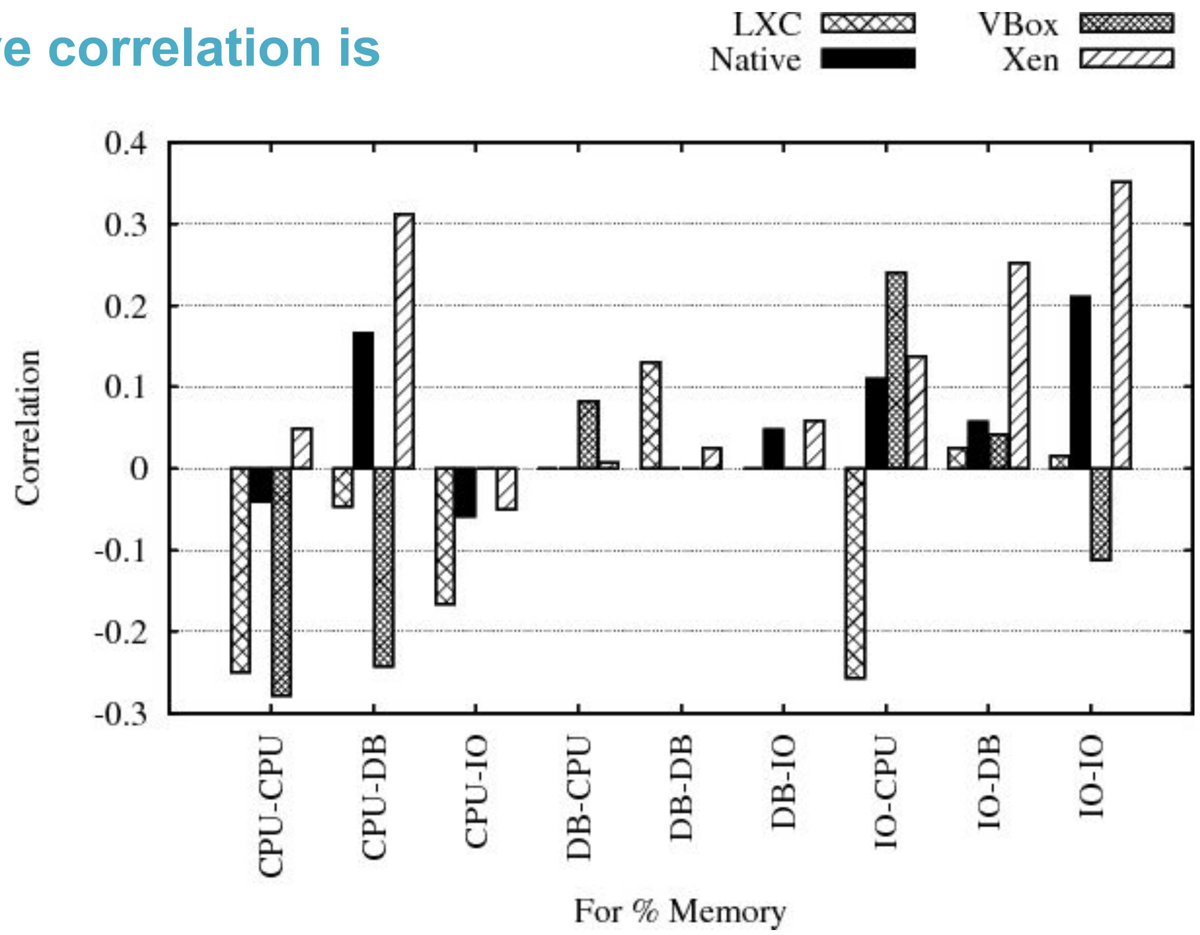
Performance Interference (CPU)

Here, +ve correlation is bad!



Performance Interference (Memory)

Here, +ve correlation is bad!



Summary

- Performance best practice from non-cloud environments continue to apply on cloud
- But you can leverage cloud specific characteristics
  - Elasticity of computing resources
  - Virtual nature of resources
  - On-demand provisioning
- Performance remains a design issue on cloud as well
- One has to design performance into the application
    - It does not come automatically!

Impact Of Platform Characteristics

Platform Characteristics

- A computing platform (?aaS) carries characteristics unique to it
  - Chars. = {Functional + non-functional attributes of platform}
- For example:
  - Software abstraction of hardware resources (V12N)
  - Coarse-grained multi-tenancy (IaaS/PaaS)
  - Limited control of underlying infrastructure (PaaS)
  - Location transparency (XaaS)
- They impact guest application architecture
  - E.g ability of a guest app to achieve certain QAs

Finding Impact on Application Architecture

- Examine platform characteristics in light of architecture knowledge and reverse-engineer the tactics and patterns

QA (Possible tactic for it)	Remarks
Reliability (State checkpoint)	On finding a failure, the system can be restored to a prior check-pointed consistent state.
Performance (Add concurrency)	Processing different sets of tasks in parallel by creating additional threads.
Security (Limit exposure)	Services are deployed on hosts in a manner that reduces overall damage when the host is compromised.

Sodhi & Prabhakar, "Cloud Platforms: Impact on Guest Application Quality Attributes" in IEEE APSCC 2012

Characteristics' Impact on QAs

- Some examples:

Characteristic	Impacted QA
Software abstraction of hardware	DR, Efficiency (+)
Programmatic self-serviced provisioning	Deployability, Scalability (+)
VM/resource check-pointing and snap shots	Availability, DR, Reliability (+)
Lack of computing assets custody	Privacy, Security (-)
Relative anonymity behind subscription and usage	Security (-)

Sodhi & Prabhakar, "Cloud Platforms: Impact on Guest Application Quality Attributes"
in IEEE APSCC 2012

How To Make Use Of This?

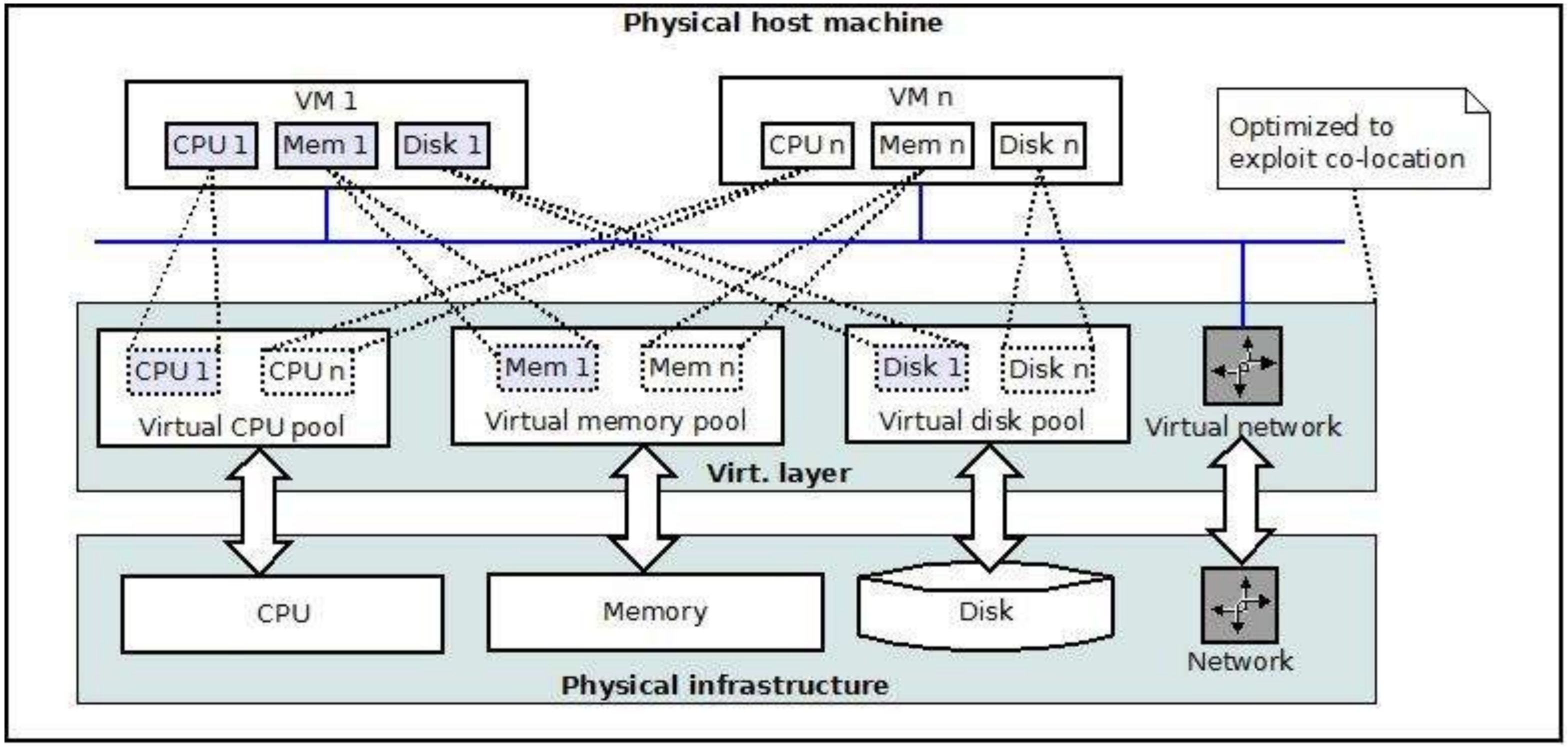
- Can be used address different types of application scenarios
  - Implement failure recovery at platform level
  - Co-locate different tiers on same host keeping them isolated
  - Selective request/computation transfer

Sodhi & Prabhakar, "Cloud-oriented platforms: Bearing on Application Architecture and Design patterns" in IEEE Services 2012

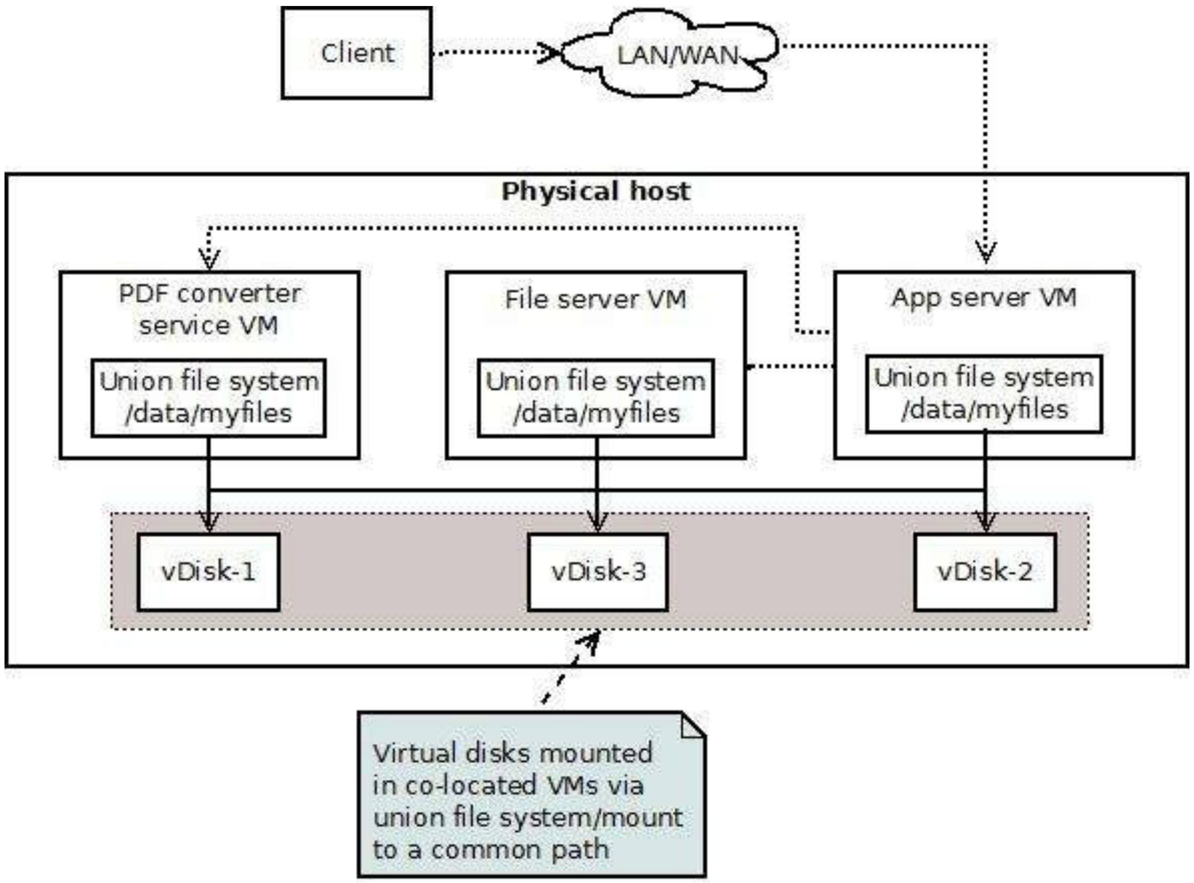
Tier Co-location: Problem Scenario

- Application consisting of multiple tiers
  - e.g. a N-tier web application
- There is some over-the-wire data exchanged among tiers
  - This is costly!
- You want to retain tier isolation, but still minimize inter-tier communication overheads

Tier Co-location: Logical View



Tier Co-location: Example

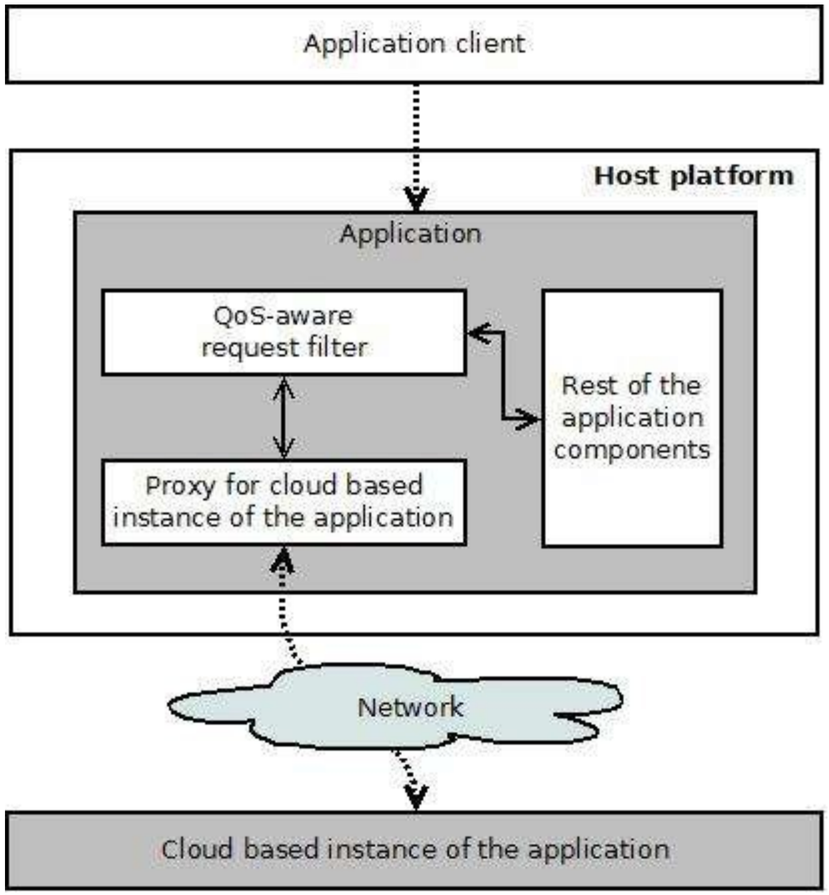

Client
LAN/WAN
Physical host
PDF converter service VM
Union file system /data/myfiles
File server VM
Union file system /data/myfiles
App server VM
Union file system /data/myfiles
vDisk-1
vDisk-3
vDisk-2
Virtual disks mounted in co-located VMs via union file system/mount to a common path

Selective Computation Transfer

- Problem context:
  - An application functionality is such that:
    - Different requests may take different amount of resources to process
    - Some requests may require execution of privileged components
    - Majority of requests can be served from normal deployment environment
  - You want to maintain certain QoS for *ALL* requests

Selective Computation Transfer

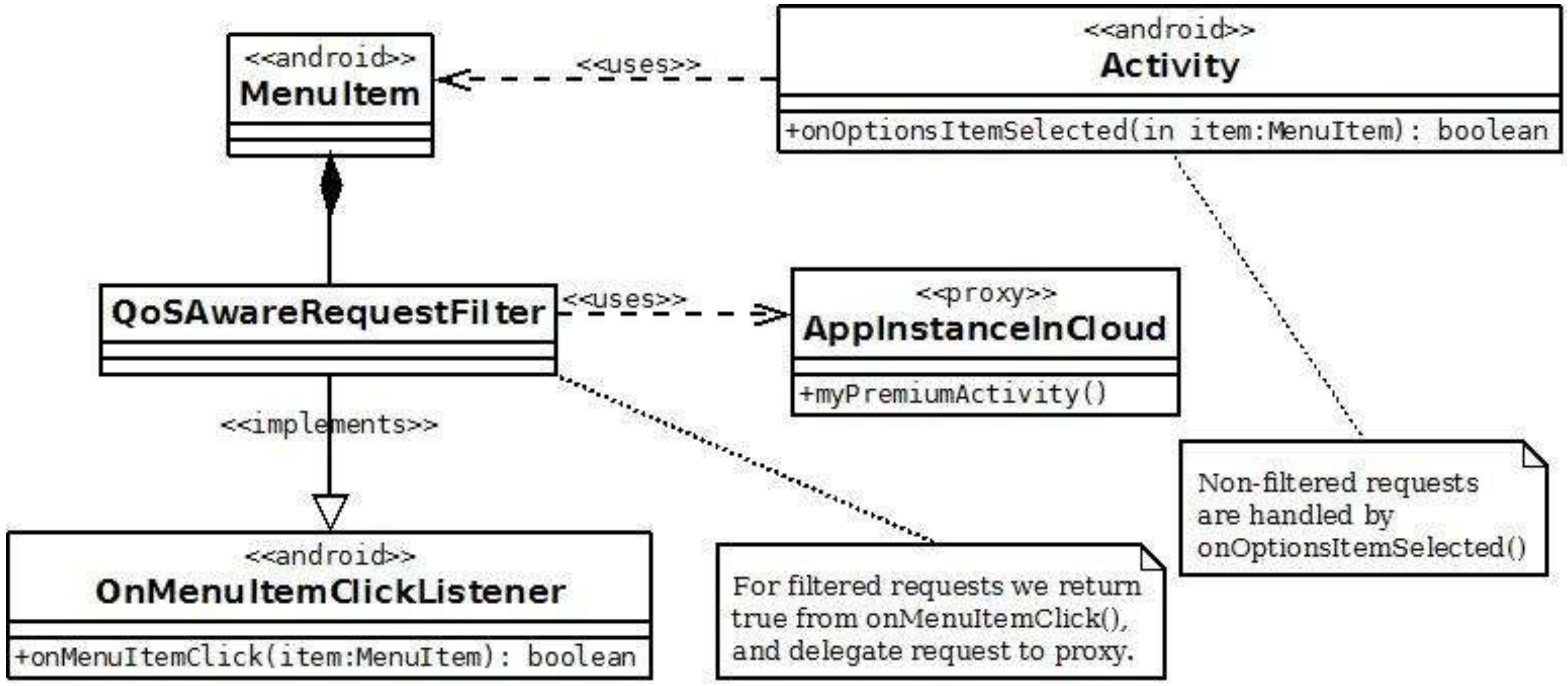
Architectu re



Application client
Host platform
Application
QoS-aware request filter
Rest of the application components
Proxy for cloud based instance of the application
Network
Cloud based instance of the application

Selective Computation Transfer

- Example implementation

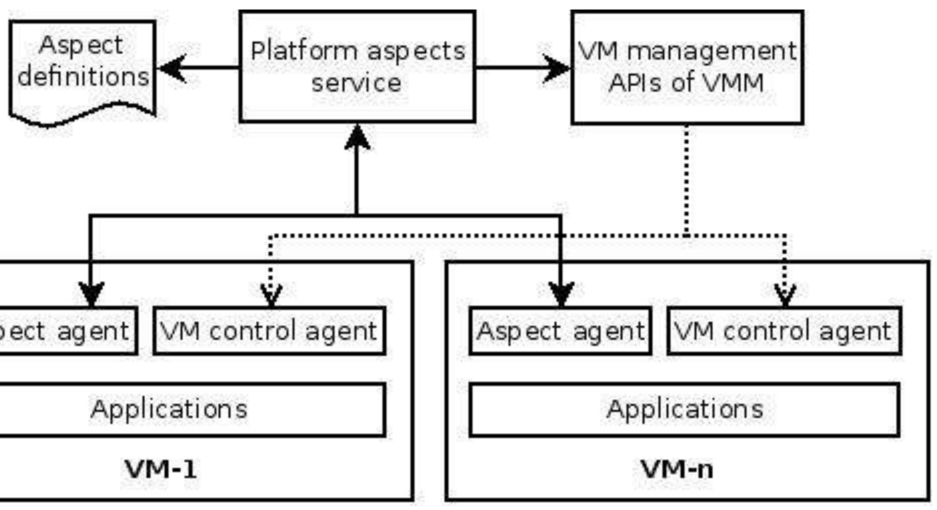


<<android>>
MenuItem
<<uses>>
<<android>>
Activity
+onOptionsItemSelected(in item:MenuItem): boolean
QoSAwareRequestFilter
<<uses>>
<<proxy>>
AppInstanceInCloud
+myPremiumActivity()
<<implements>>
Non-filtered requests are handled by onOptionsItemSelected()
<<android>>
OnMenuItemClickListener
+onMenuItemClick(item:MenuItem): boolean
For filtered requests we return true from onMenuItemClick(), and delegate request to proxy.

Aspect-orientation at Platform-level

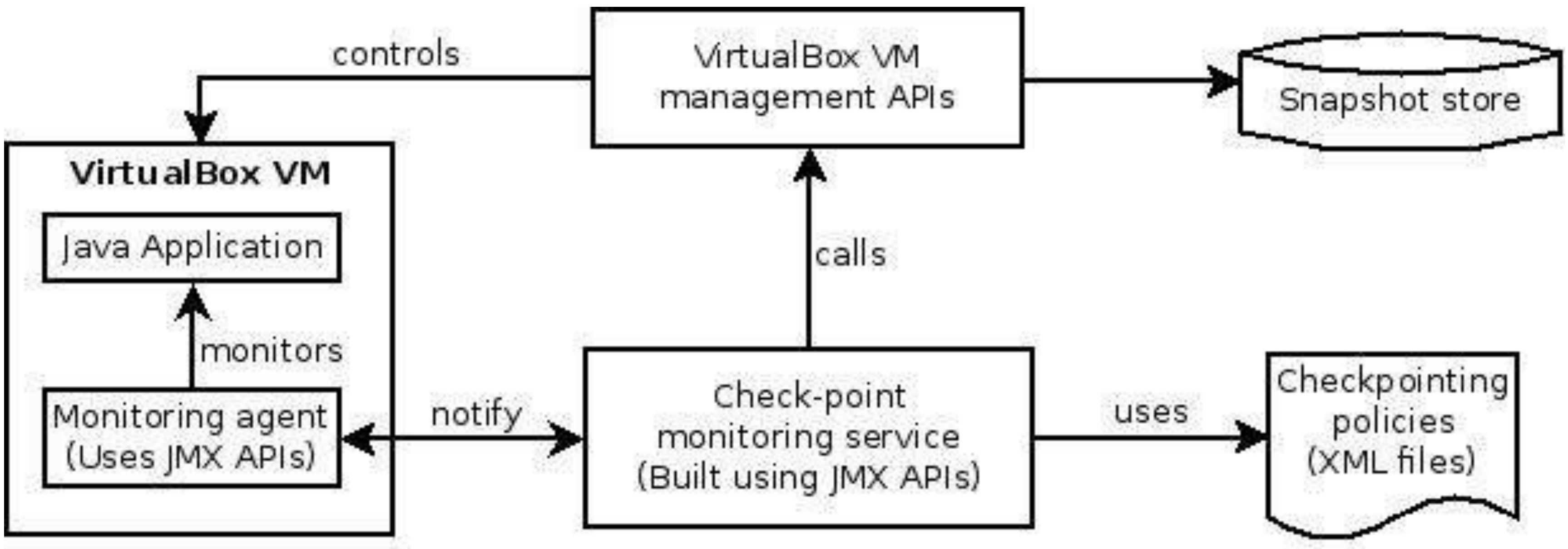
- Problem context:
  - There are software design and implementation concerns which apply at the coarse grained computing environment/platform level, and they cut across the applications
    - E.g. monitoring and reacting to platform level events

Aspect-orientation at Platform-level



Aspect-orientation at Platform-level

- Example implementation



controls
VirtualBox VM management APIs
Snapshot store
VirtualBox VM
Java Application
monitors
Monitoring agent (Uses JMX APIs)
calls
notify
Check-point monitoring service (Built using JMX APIs)
uses
Checkpointing policies (XML files)

Summary

- It is important to determine and understand platform characteristics
  - Cloud and virtualization based platforms have several unique characteristics
- Devise solutions to application scenarios
  - Leverage platform characteristics