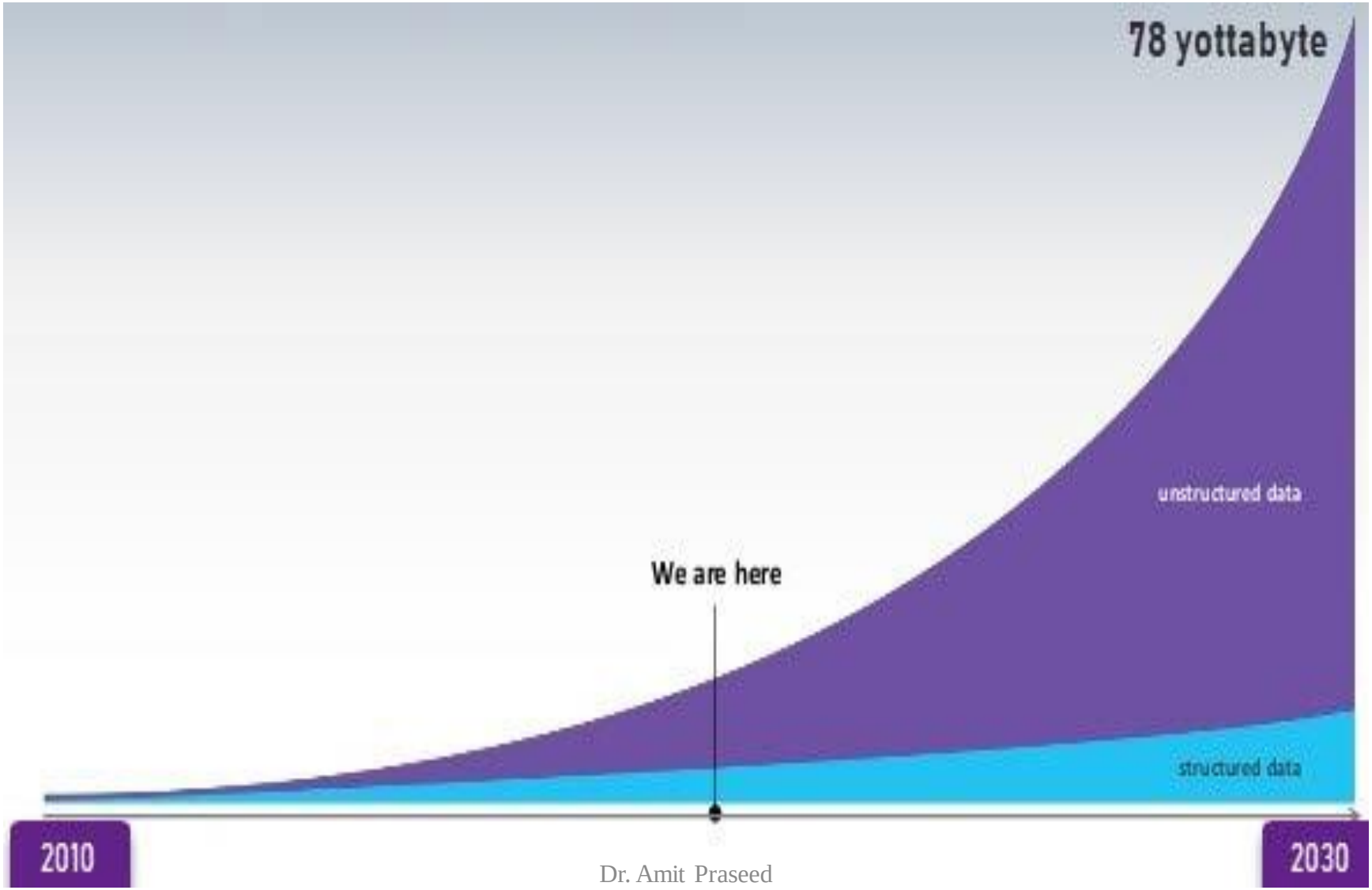
Big Data and its Implications on the Cloud

Dr. Amit Praseed

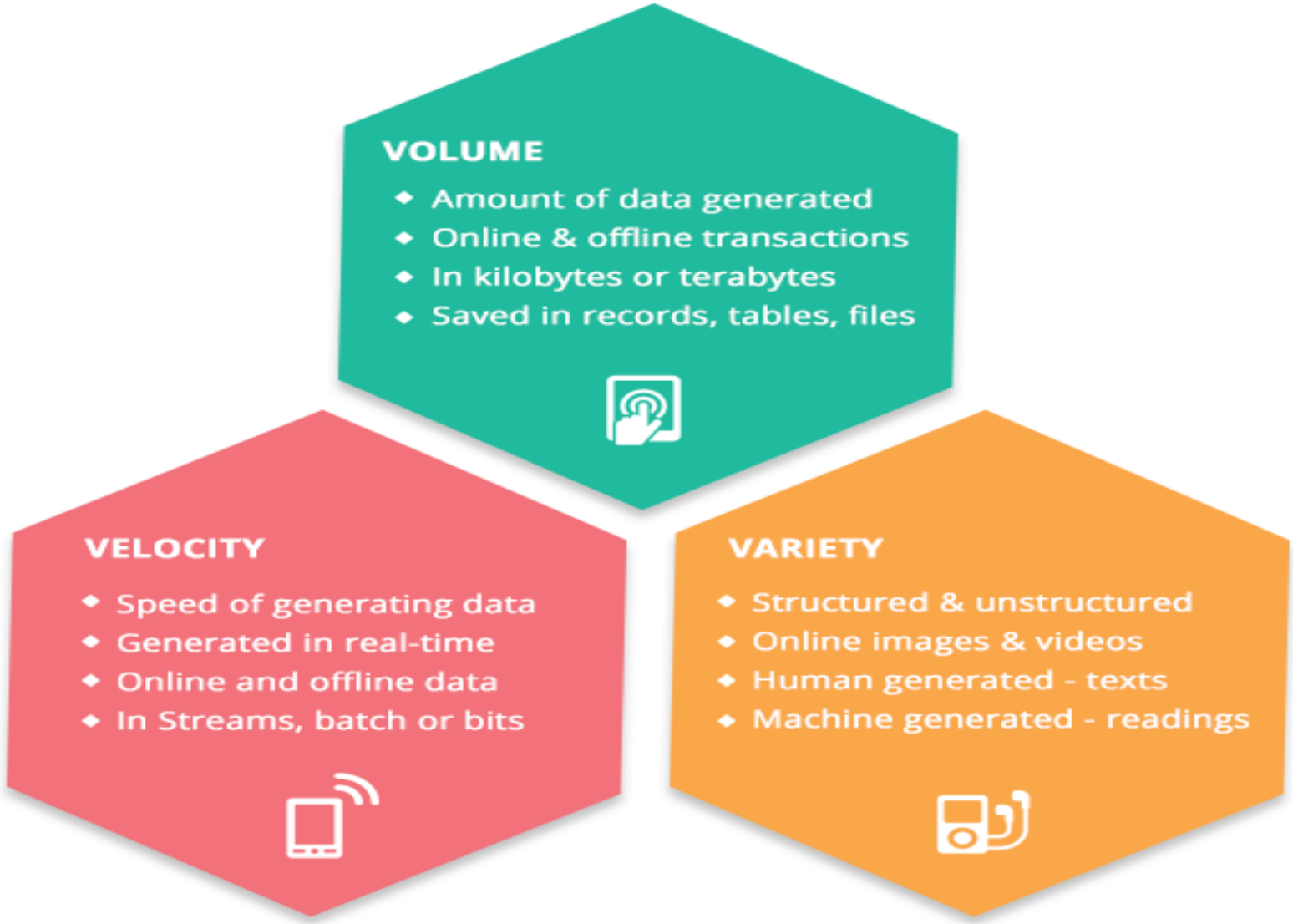
How much data do we generate?

- **Structured Data**
  - Relational Databases
  - Well defined schema
- **Unstructured Data**
  - Videos, audio, images etc.
- **Semi-structured Data**
  - structured in form but not well defined (no schema)
  - XML files

The Rise of Unstructured Data



THE 3Vs OF BIG DATA



The Rise of NoSQL

- NoSQL = Not only SQL
- A fundamental shift or alternative to storing data which does not conform to the relational format
- Features
  - Schema Agnostic
  - Non relational
  - Commodity hardware
  - Highly distributable

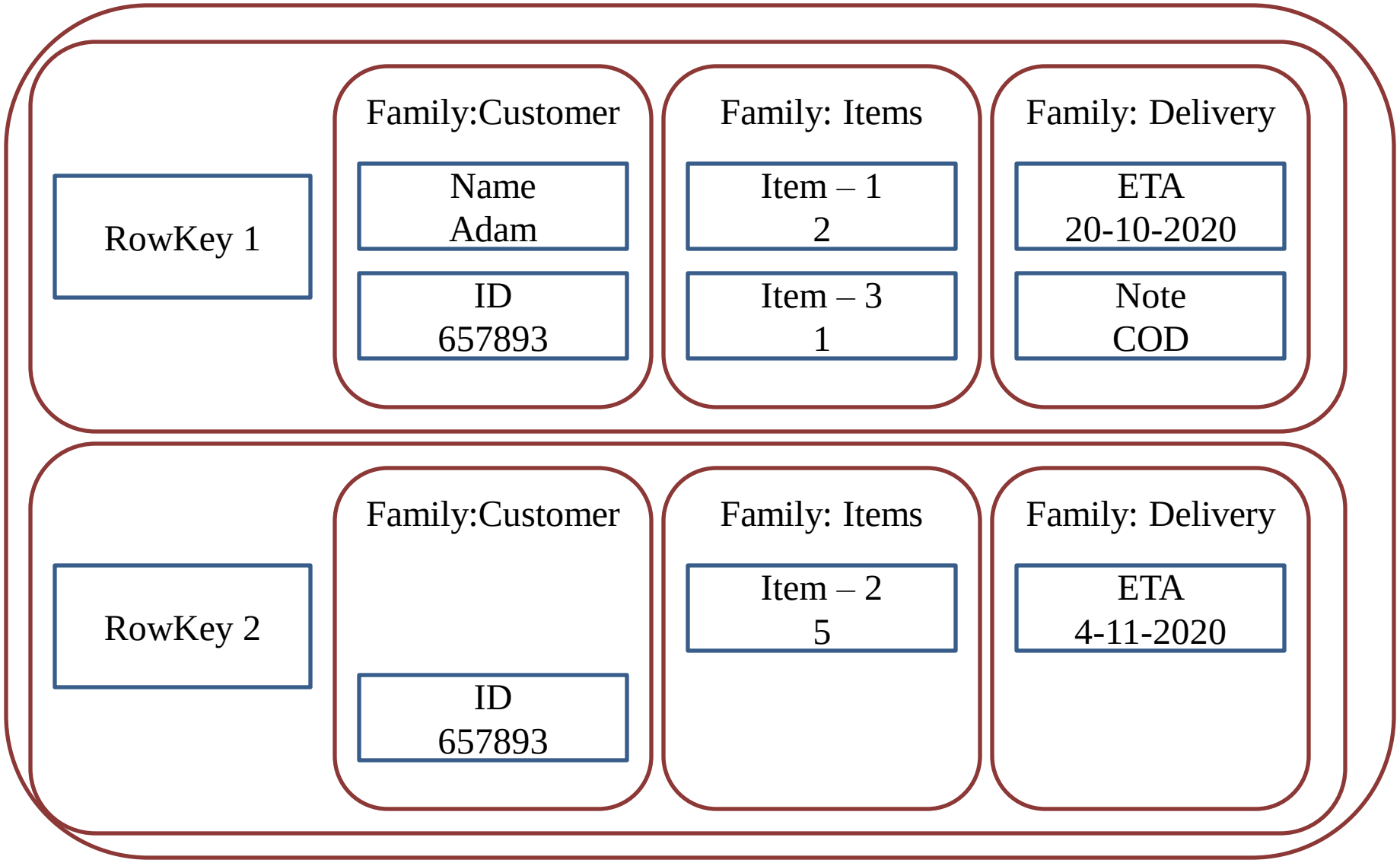
Four Core NoSQL Varieties

- Columnar
- Key-Value
- Triple
- Document

Columnar Databases

- Similar to relational model – concept of rows and columns still exist
- Optimized and designed for faster column access
- Ideal for running aggregate functions or for looking up records that match multiple columns
- A single record may consist of an ID field and multiple column families
- Each one of these column families consists of several fields. One of these column families may have multiple “rows”

Columnar Databases



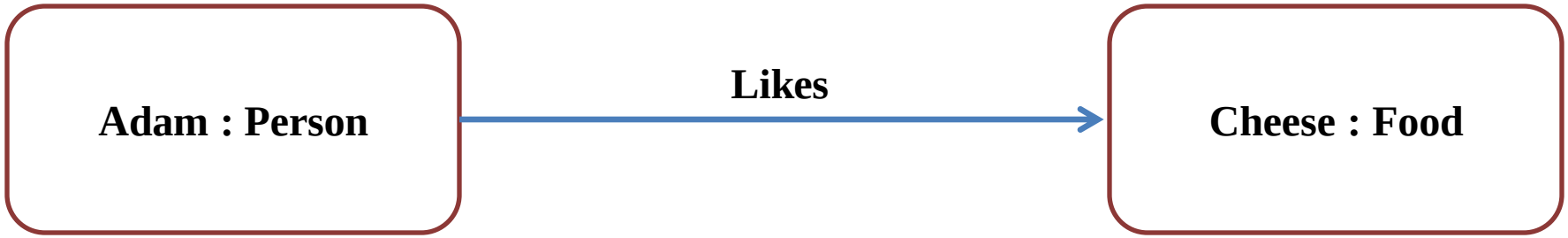
Key – Value Stores

- An ID field — the key in key-value stores — and a set of data
- Some key-value stores support typing (such as integers, strings, and Booleans) and more complex structures for values (such as maps and lists)
- Key-value stores are optimized for speed of ingestion and retrieval

Triple and Graph Stores

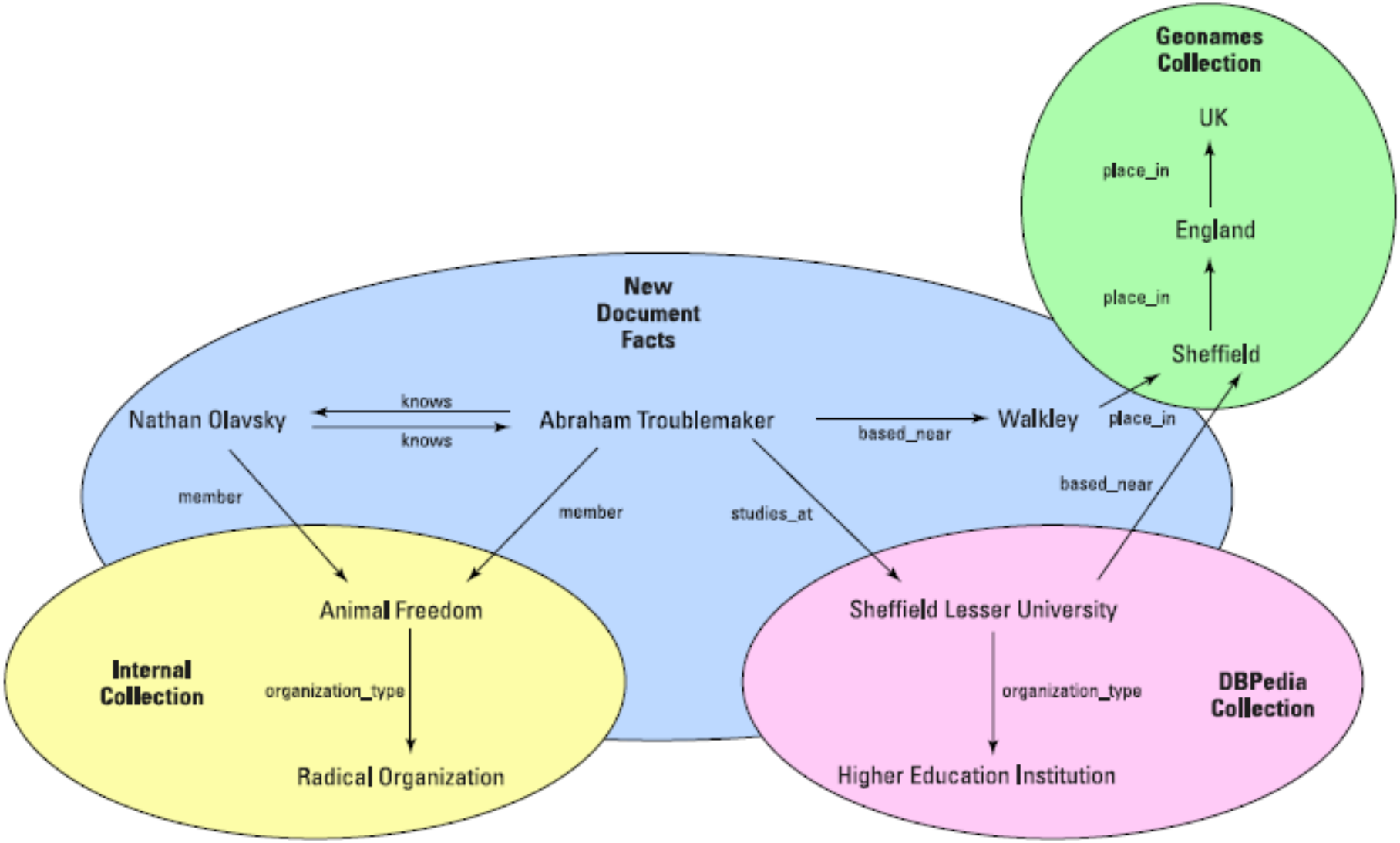
- Every *fact* or assertion is described as a triple of subject, predicate, and object:
  - A *subject* is the thing you're describing. It has a unique ID called an IRI. It may also have a type, which could be a physical object (like a person) or a concept (like a meeting).
  - A *predicate* is the property or relationship belonging to the subject. This again is a unique IRI that is used for all subjects with this property.
  - An *object* is the intrinsic value of a property (such as integer or Boolean, text) or another subject IRI for the target of a relationship.

Triple and Graph Stores



- Three points of information
  - Adam is a person
  - Cheese is a food item
  - Adam likes cheese

Triple and Graph Stores



Document Stores

- Hold documents that combine information in a single logical unit
- Retrieving all information from a single document is easier with a database and is more logical for applications
- A document is any unstructured or tree-structured piece of information.
  - It could be a recipe, financial services trade, PowerPoint file, PDF, plain text, or JSON or XML document.

Examples of NoSQL Products

- **Columnar:** DataStax, Apache Cassandra, HBase, Apache Accumulo, Hypertable
- **Key-value:** Basho Riak, Redis, Voldemort, Aerospike, Oracle NoSQL
- **Triple/graph:** Neo4j, Ontotext's GraphDB (formerly OWLIM), MarkLogic, OrientDB, AllegroGraph, YarcData
- **Document:** MongoDB, MarkLogic, CouchDB, FoundationDB, IBM Cloudant, Couchbase

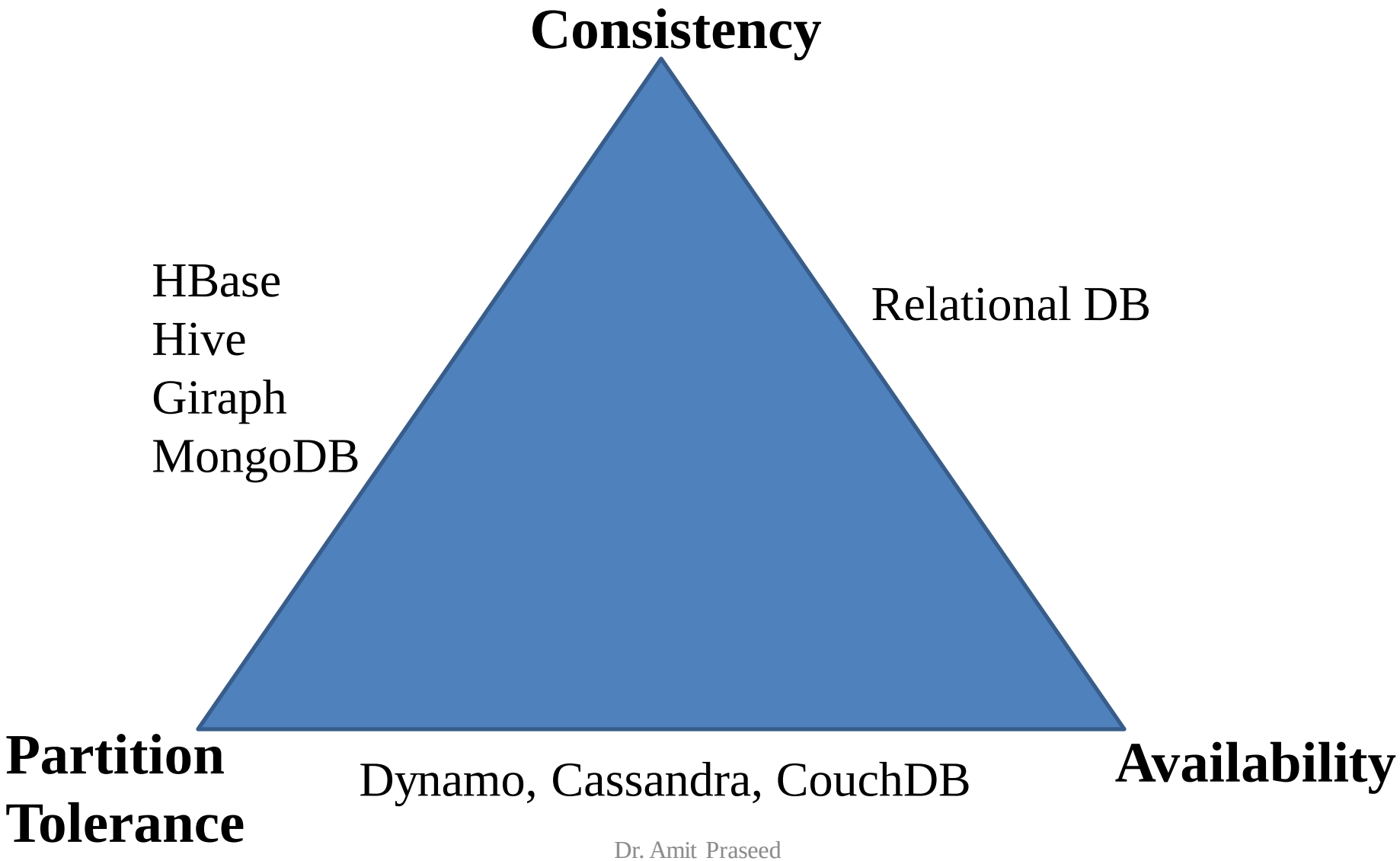
The CAP Theorem

- It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:
  - **Consistency**: Every read receives the most recent write or an error
  - **Availability**: Every request receives a response, without the guarantee that it contains the most recent write
  - **Partition tolerance**: The system continues to operate despite an arbitrary number of messages being dropped by the network between nodes

Consistency in NoSQL

- NoSQL relaxes the ACID consistency model used in relational databases
- NoSQL uses an eventual consistency model called BASE
  - Basically Available
  - Soft state
  - Eventual consistency

ACID, BASE and the CAP Theorem



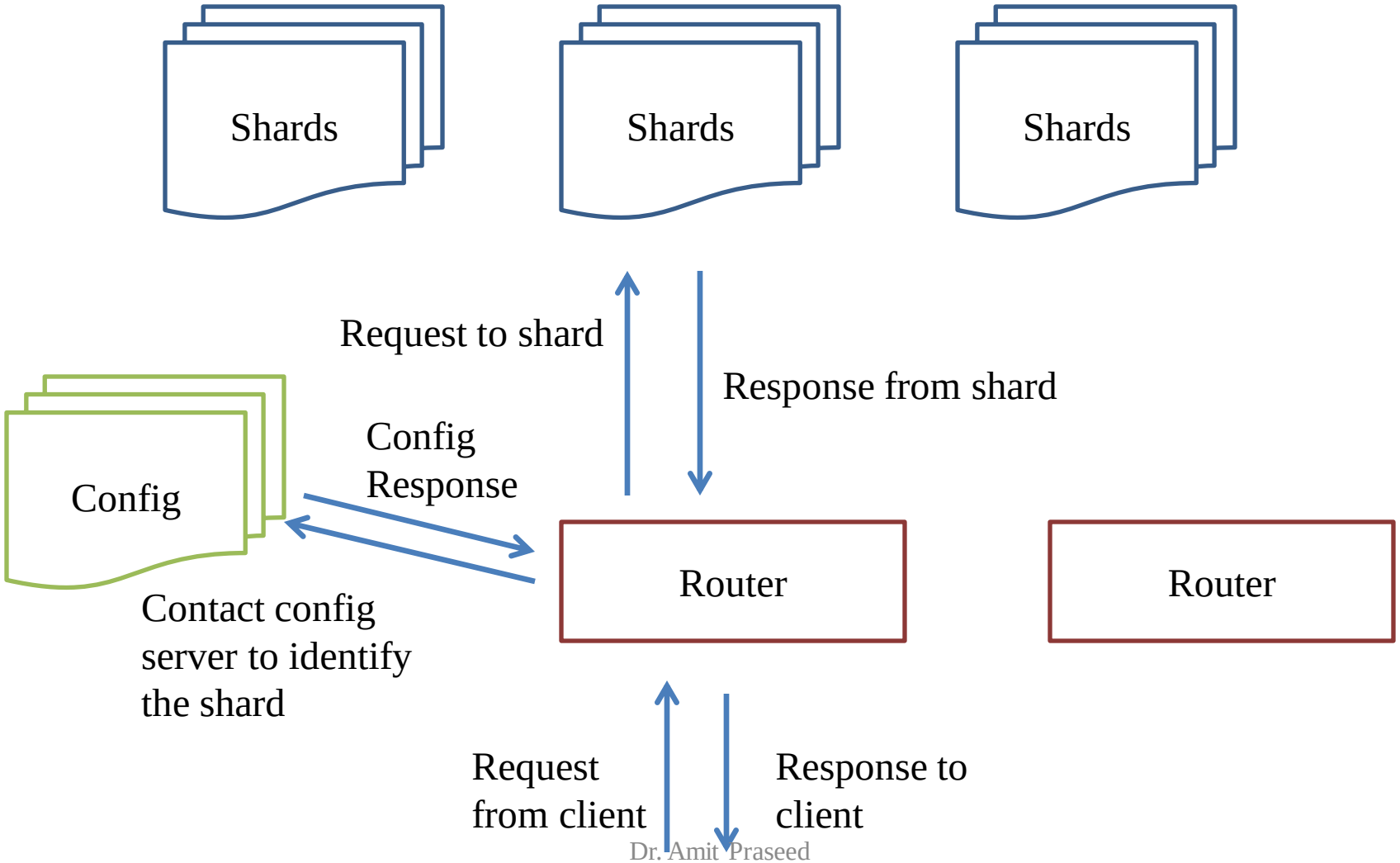
MongoDB

- A document-oriented, NoSQL database
  - Hash-based, schema-less database
  - Atomic writes and fully-consistent reads
  - Master-slave replication with automated failover (replica sets)
  - Built-in horizontal scaling via automated range-based partitioning of data (sharding)
  - No joins nor transactions
  - Consistency + Partition Tolerance

MongoDB Terminology

- A MongoDB instance may have zero or more ‘databases’
- A database may have zero or more ‘collections’
- A collection may have zero or more ‘documents’
- A document may have one or more ‘fields’
- MongoDB ‘Indexes’ function much like their RDBMS counterparts.

MongoDB Architecture

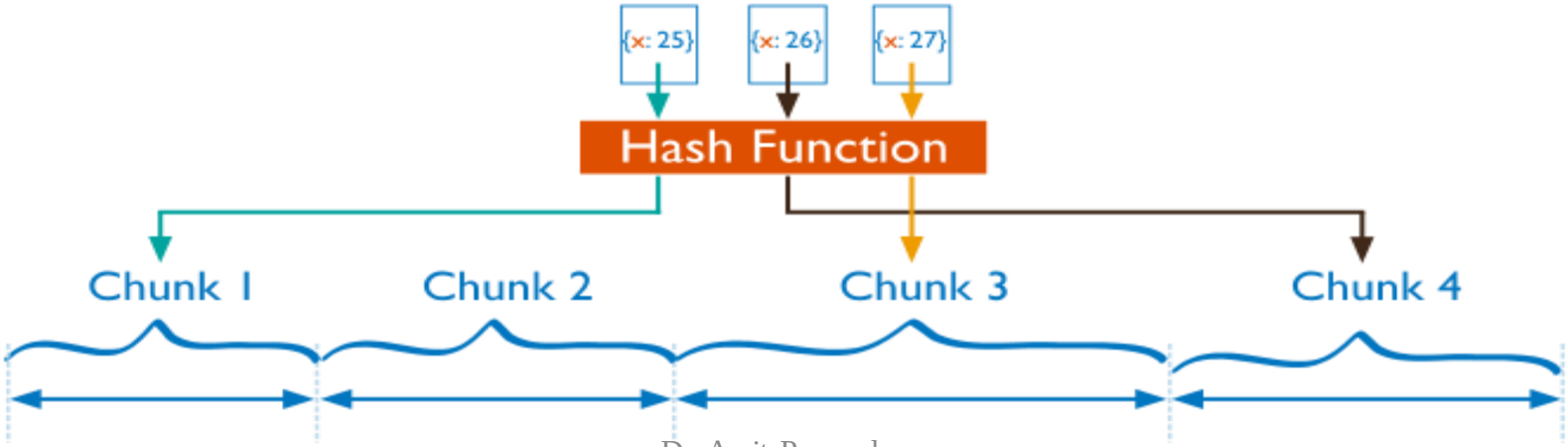


MongoDB Replication

- A replica set is a group of mongod instances that maintain the same data set.
  - A replica set contains several data bearing nodes and optionally one arbiter node.
  - Of the data bearing nodes, one member is deemed the primary node, while the other nodes are deemed secondary nodes.
- The primary node receives all write operations.
  - The primary records all changes to its data sets in its operation log, i.e. oplog
  - The secondaries replicate the primary's oplog and apply the operations to their data sets

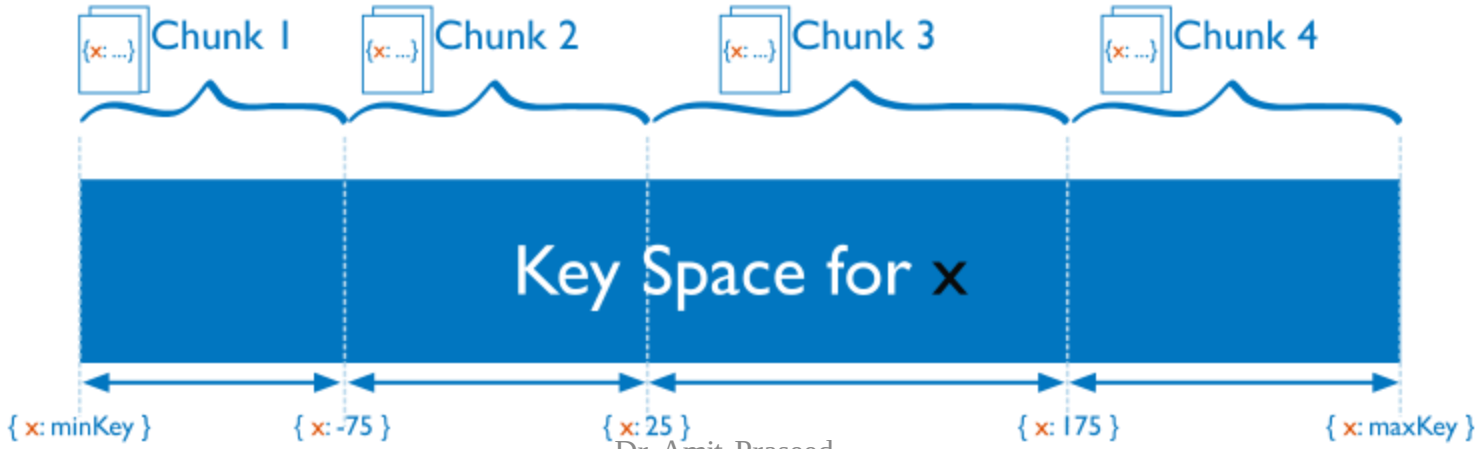
Sharding in MongoDB

- **Hashed Sharding**
  - Involves computing a hash of the shard key field's value.
  - Each chunk is then assigned a range based on the hashed shard key values.
  - Even data distribution, but may result in more cluster wide broadcast operations



Sharding in MongoDB

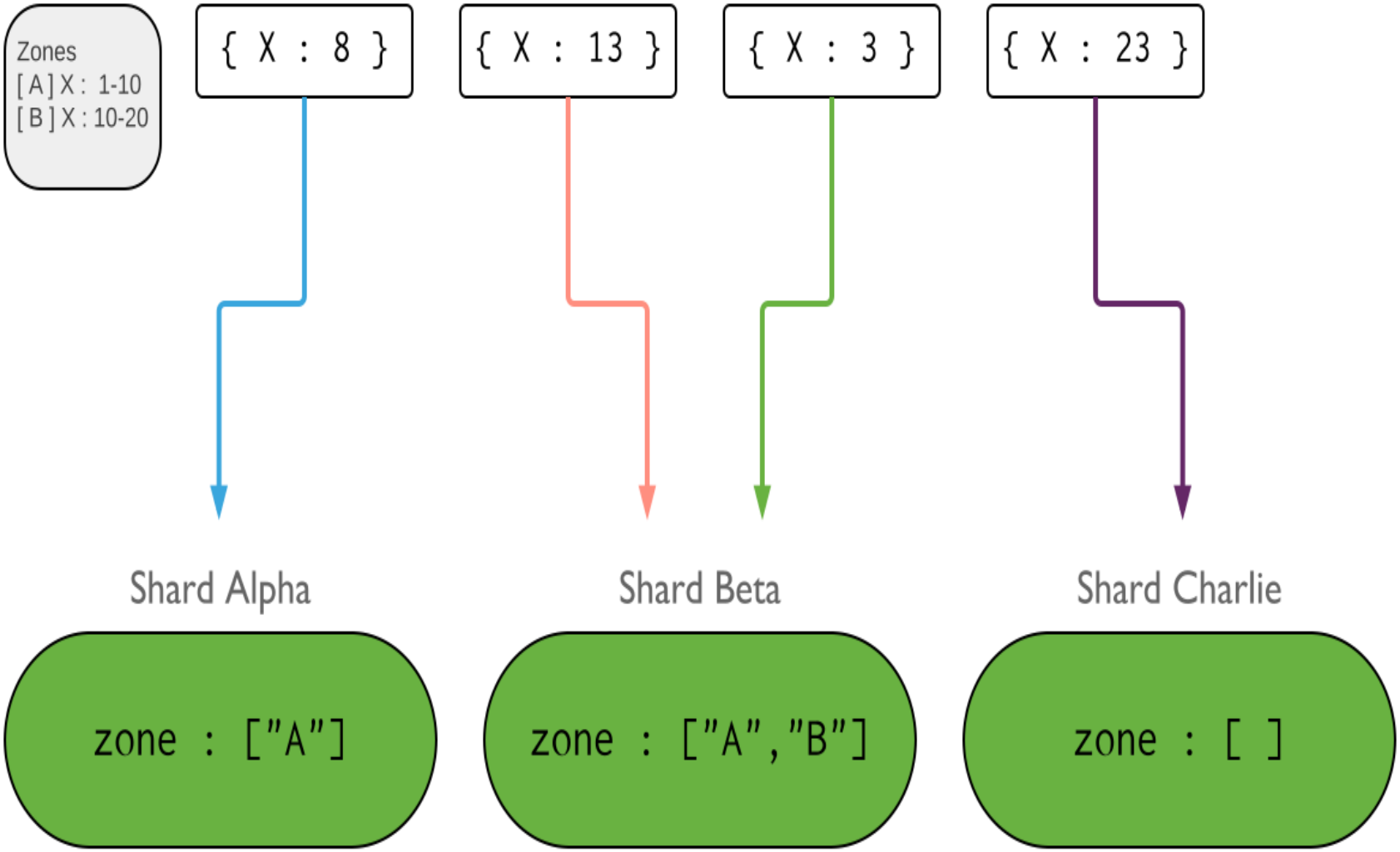
- **Ranged Sharding**
  - Involves dividing data into ranges based on the shard key values.
  - Each chunk is then assigned a range based on the shard key values
  - Allow targeted operations, but poorly chosen shard key may result in uneven data distribution



Sharding in MongoDB

- **Zoned Sharding**
  - Zones can help improve the locality of data for sharded clusters that span multiple data centers.
  - In sharded clusters, you can create zones of sharded data based on the shard key.
  - You can associate each zone with one or more shards in the cluster. A shard can associate with any number of zones.
  - In a balanced cluster, MongoDB migrates chunks covered by a zone only to those shards associated with the zone.
  - Each zone covers one or more ranges of shard key values

Zoned Sharding



Choosing a Shard Key

- The shard key is either an indexed field or indexed compound fields that determines the distribution of the collection's documents among the cluster's shards.
- All sharded collections must have an index that supports the shard key; i.e. the index can be an index on the shard key or a compound index where the shard key is a prefix of the index
- The choice of shard key affects the creation and distribution of the chunks across the available shards. This affects the overall efficiency and performance of operations within the sharded cluster.

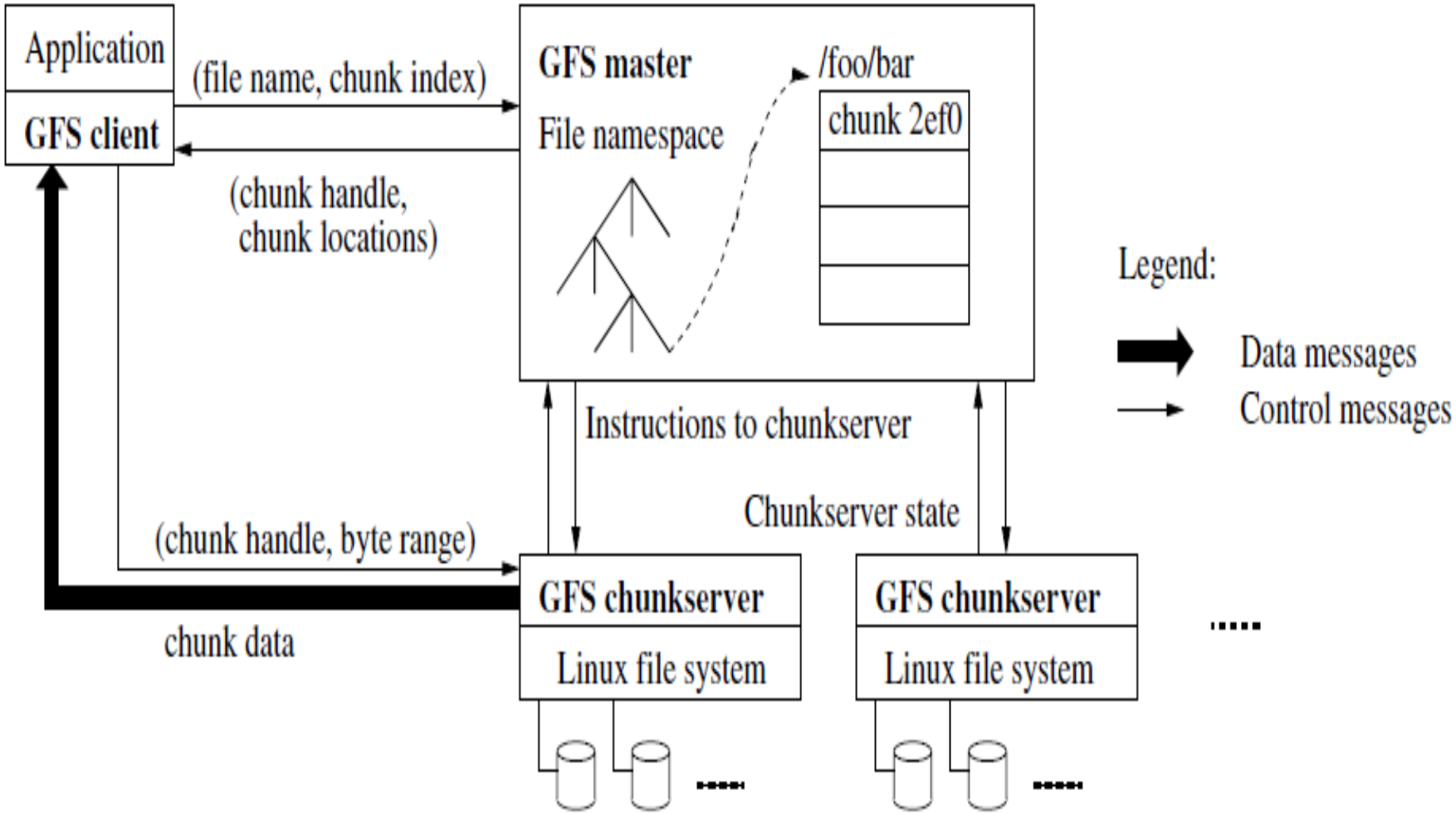
Google File System

Dr. Amit Praseed

Assumptions behind GFS

- Component failures are the norm rather than the exception
- Files are huge by traditional standards
- Common workloads encountered
  - large streaming reads and small random reads
  - large, sequential writes that append data to files
- System must efficiently implement well-defined semantics for multiple clients that concurrently append to the same file
- High sustained bandwidth is more important than low latency

GFS Architecture



GFS Cluster Organization

- GFS cluster has
  - Single master
  - Multiple chunk servers
- Files are divided into fixed size chunks
  - Default 64 MB size (Reasonably huge – why?)
  - Identified by a global 64 bit chunk handle
  - Each chunk replicated on multiple chunk servers (by default 3)

Operations of the Master Server

- Maintains file system metadata
  - Namespace
  - Access control information
  - Files ←→ chunks mapping
  - Chunk locations etc.
- Other bookkeeping tasks
  - Chunk lease management
  - Garbage collection
  - Chunk migration
- Checks chunk server availability and health
  - Heartbeat messages

Mutations and Leases

- A mutation is an operation that changes the contents or metadata of a chunk
  - write or an append operation.
- Each mutation is performed at all the chunk's replicas.
- Leases are used to maintain a consistent mutation order across replicas.
  - The master grants a chunk lease to one of the replicas [Primary]
  - The primary picks a serial order for all mutations to the chunk.
  - All replicas follow this order when applying mutations.
  - Thus, the global mutation order is defined by
    - the lease grant order chosen by the master, and
    - within a lease by the serial numbers assigned by the primary.

Handling Write Operations

- The client asks the master which chunk server holds the current lease
  - The master replies with the identity of the primary and the locations of the other (*secondary*) replicas
  - The client caches this data for future mutations
- The client pushes the data to all the replicas
  - Data flows from Client → Replica 1 → Replica 2 …
- Once all the replicas have acknowledged receiving the data, the client sends a write request to the primary

Handling Write Operations

- The primary assigns consecutive serial numbers to all the mutations it receives, possibly from multiple clients
  - It applies the mutation to its own local state in serial number order
- The primary forwards the write request to all secondary replicas.
  - Each secondary replica applies mutations in the same serial number order assigned by the primary
  - The secondaries all reply to the primary indicating that they have completed the operation.
- The primary replies to the client

Atomic Record Appends

- GFS provides an atomic append operation called *record append*.
  - Client pushes the data to all replicas and sends its request to the primary.
  - Primary checks to see if appending the record to the current chunk would cause the chunk to exceed the maximum size (64 MB).
    - If so, it pads the chunk to the maximum size, tells secondaries to do the same, and replies to the client indicating that the operation should be retried on the next chunk.
    - If the record fits within the maximum size, the primary appends the data to its replica, tells the secondaries to write the data at the exact offset where it has, and finally replies success to the client.
  - If a record append fails at any replica, the client retries the operation.
  - As a result, replicas of the same chunk may contain different data possibly including duplicates of the same record in whole or in part.
- GFS does not guarantee that all replicas are bytewise identical
  - It only guarantees that the data is written at least once as an atomic unit.

Chunk Placement and Replication

- When the master *creates* a chunk, it chooses where to place the initially empty replicas
  - Place new replicas on chunk servers with below-average disk space utilization
  - Limit the number of "recent" creations on each chunk server
  - Spread replicas of a chunk across racks.
- Master may need to re-replicate chunks
  - Chunk server becomes unavailable
  - Replica may be corrupted
  - Disk errors
  - Increased replication goal
- The master *rebalances* replicas periodically: it examines the current replica distribution and moves replicas for better disk space and load balancing

Garbage Collection

- Lazy Garbage Collection
  - When a file is deleted by the application, the master logs the deletion immediately just like other changes
  - The file is just renamed to a hidden name that includes the deletion timestamp.
  - During the master's regular scan of the file system namespace, it removes any such hidden items
  - Until then, the file can still be read under the new, special name and can be undeleted by renaming it back to normal.
  - In a similar regular scan of the chunk namespace, the master identifies orphaned chunks and erases the metadata for those chunks.
  - In a *HeartBeat* message regularly exchanged with the master, each chunkserver reports a subset of the chunks it has, and the master replies with the identity of all chunks that are no longer present in the master's metadata.
  - The chunkserver is free to delete its replicas of such chunks.

Hadoop Distributed File System

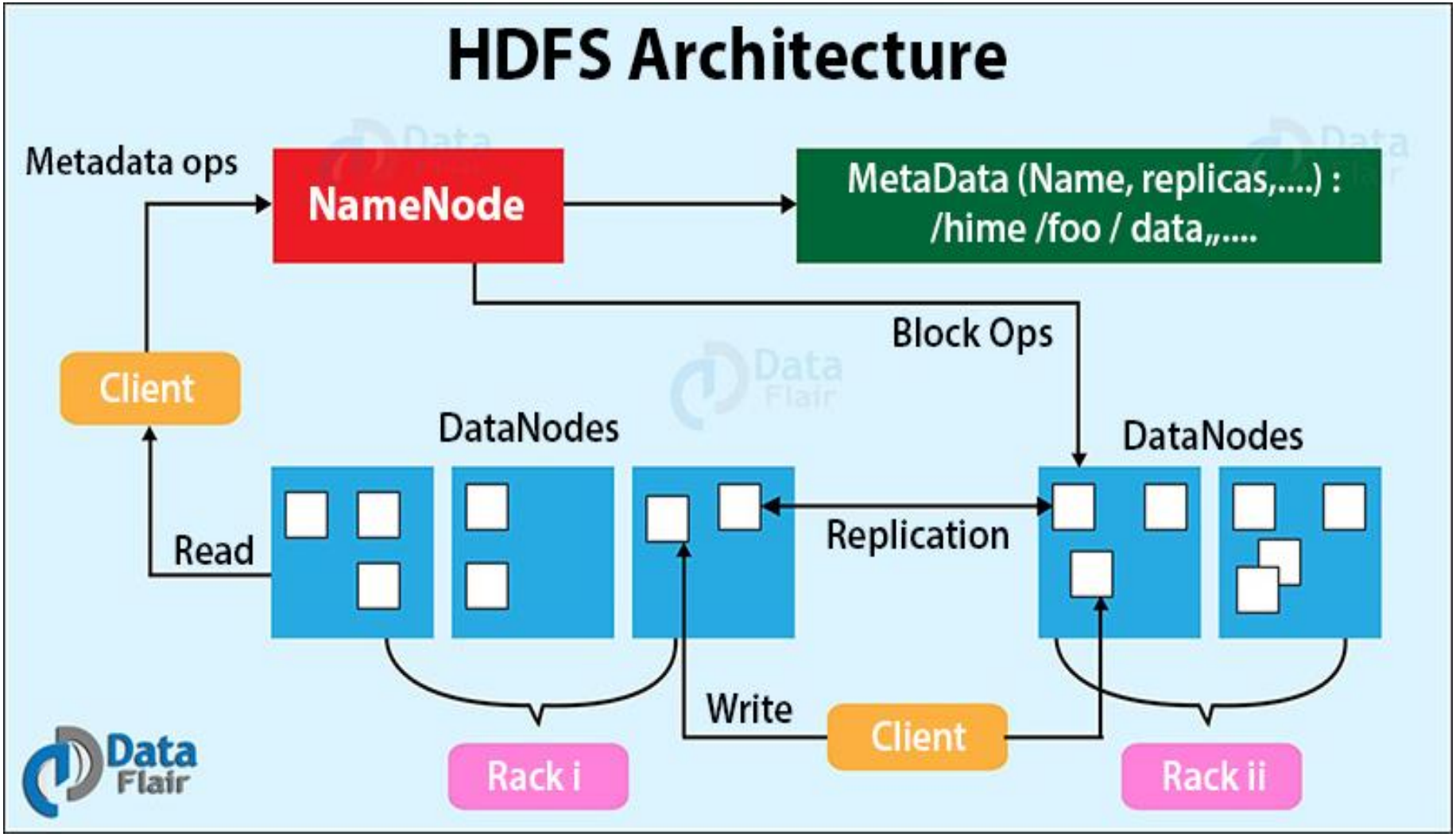
Dr. Amit Praseed

Assumptions behind HDFS

- Runs on commodity hardware – failure is common
- Works well with a number of large files
- Optimized for “Write once, read many times”
- Optimized for large streaming reads
- High throughput is more important than low latency

Notice the similarity with GFS!
This is because HDFS was designed and built based on GFS specifications

HDFS Architecture



HDFS Architecture

- Operates on top of an existing file system
- Files are stored as blocks
  - Default size is 64 MB
- Reliability through replication
- NameNode stores metadata and manages access
- No caching due to large file sizes

HDFS File Storage

- NameNode
  - Stores metadata – filename, location of blocks etc
  - Maintains metadata in memory
- DataNode
  - Stores file contents as blocks
  - Different blocks of same file are on different data nodes
  - Same block is replicated across data nodes

Failure and Recovery

- NameNodes keep track of DataNodes through periodic HeartBeat messages
- If no heartbeat is received for a certain duration, DataNode is assumed to be lost
  - NameNode determines which blocks were lost
  - Replicates the same on other DataNodes
- NameNode failure = File system failure
- Two options
  - Persistent backup and checkpointing
  - Secondary/backup NameNode

Balancing Hadoop Clusters

- Hadoop works best when data is evenly spread out
- Goal is to have all DataNodes filled up to a similar level
- Hadoop runs a balancer daemon
  - Redistributes blocks from over utilized DataNodes to underutilized ones
  - Runs in the background and can be throttled as necessary

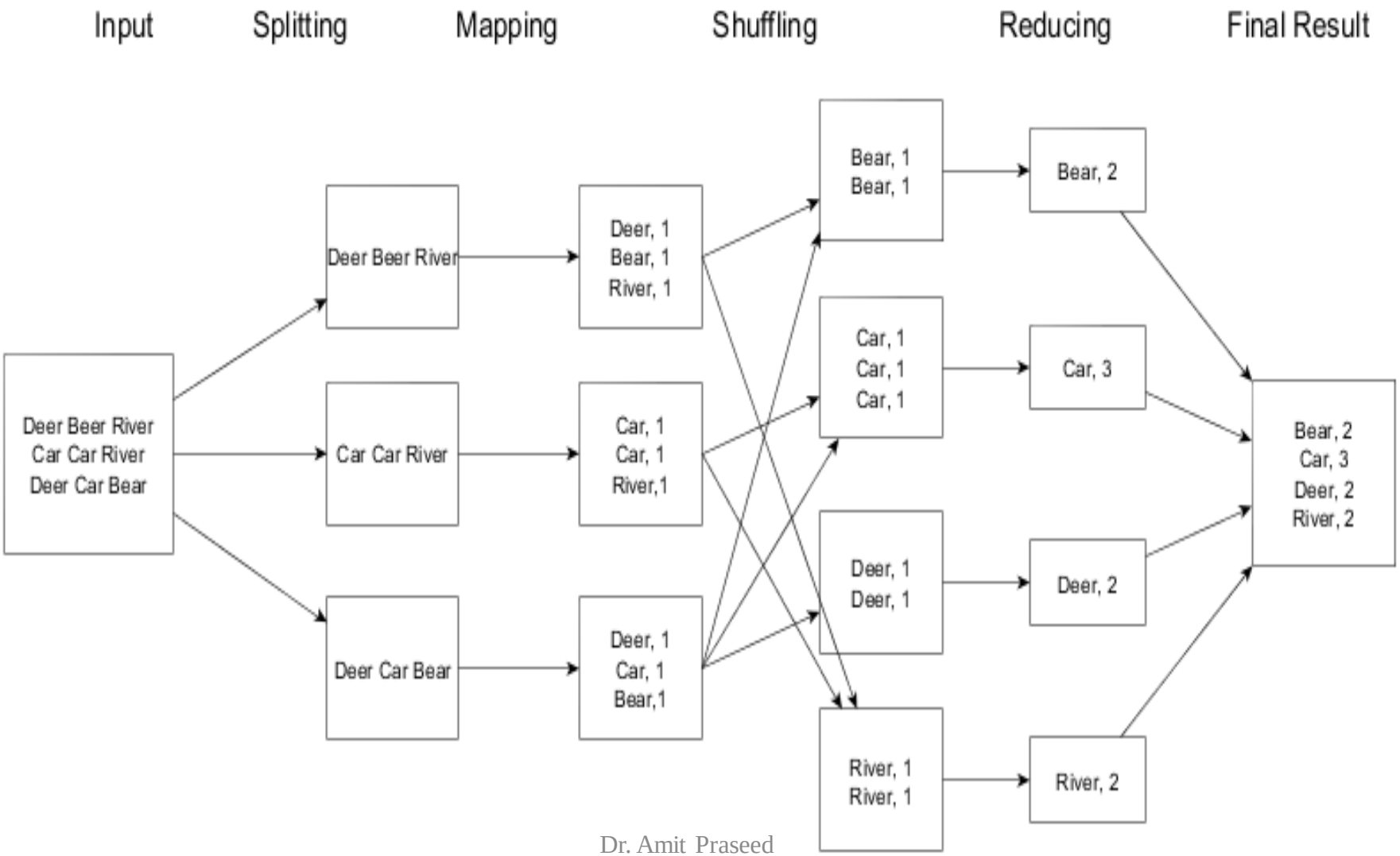
Introduction to MapReduce

Dr. Amit Praseed

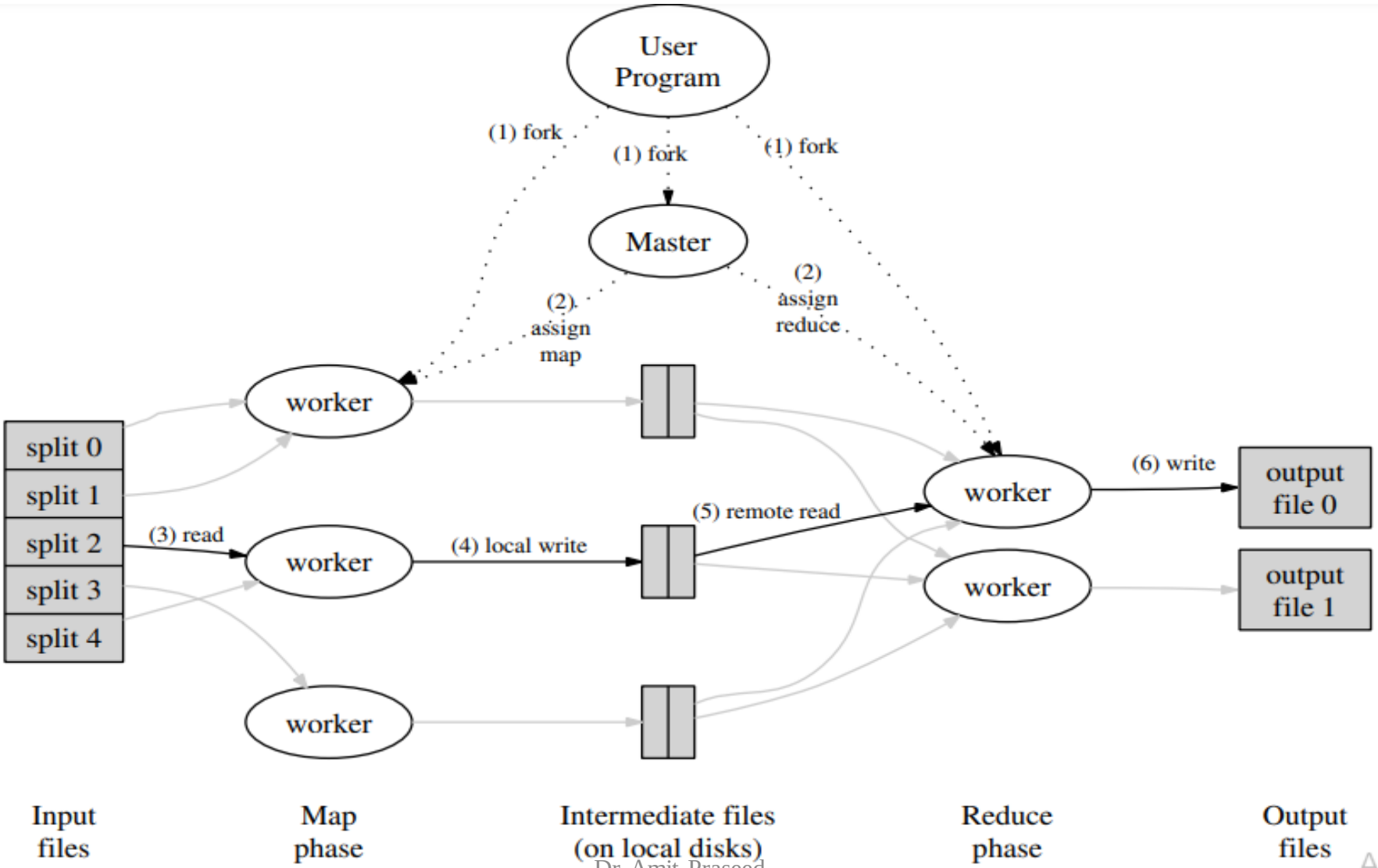
What is MapReduce?

- MapReduce is a software framework for processing large datasets in a distributed fashion
  - Map data into multiple <key, value> pairs
  - Reduce over all keys having the same data
- Consider a simple example of finding the count of every word present in a document

Word Count using MapReduce



MapReduce Workflow



Handling Node Failures

- Worker Node Failure
  - Any map task or reduce task in progress on a failed worker is set to idle and rescheduled
  - Any completed map task is also restarted, as output is stored on local disk
  - Completed reduce tasks don’t need to be restarted as output is written to a global file
- Master Node Failure
  - Periodic check pointing
  - Restart on a different master

Handling Stragglers

- Straggler : a machine that takes an unusually long time to complete one of the last few map or reduce tasks in the computation
  - a machine with a bad disk may experience slow read performance
  - Other scheduled other tasks on the machine
- When a MapReduce operation is close to completion
  - The master schedules backup executions of the remaining in-progress tasks
  - The task is marked as completed whenever either the primary or the backup execution completes

PageRank using MapReduce

- PageRank is the magic algorithm behind Google search results
- Every web site is awarded a rank based on certain features, most notable of which is the number of links from other sites to itself
- This algorithm needs to be executed on a large amount of data – hence MapReduce is a good option

Random Surfer Model

- Intuition for PageRank
- Imagine a surfer who starts on a randomly chosen page and then follows outgoing links at random
  - Markov Process
- PageRank is probability that user will arrive at a given page during this random walk

A little more complex!

- Model assumes that surfer doesn't always follow a link, but sometimes e.g. bookmarks instead.
- Before each move, surfer flips a coin
  - With probability α, follows an out-link
  - With probability 1- α, moves to a (uniformly chosen) random page

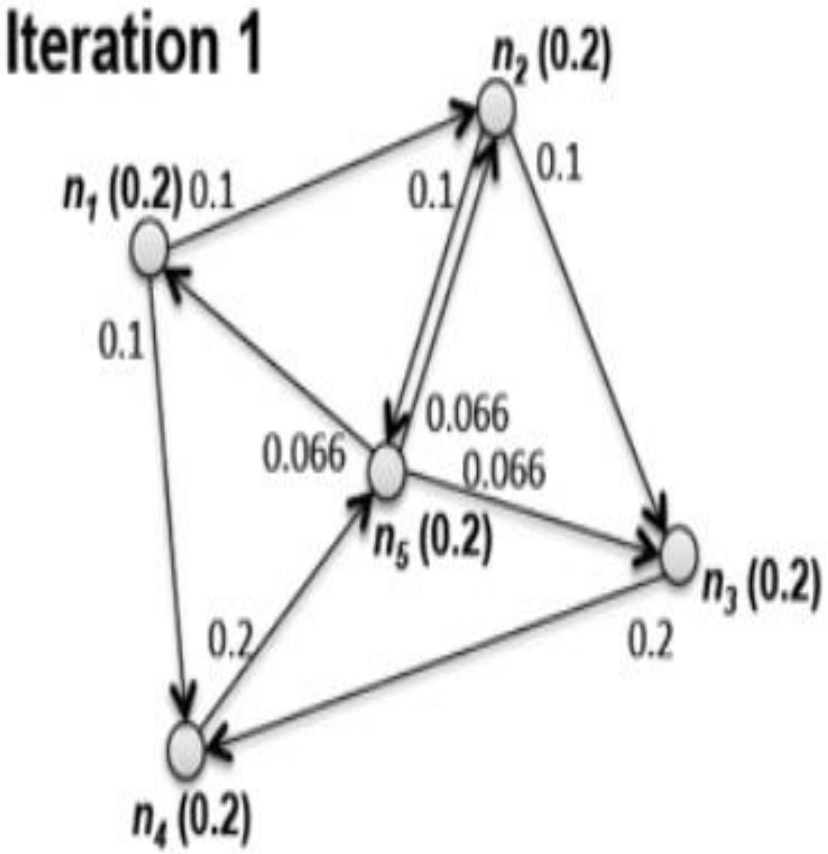
Simplified Page Rank

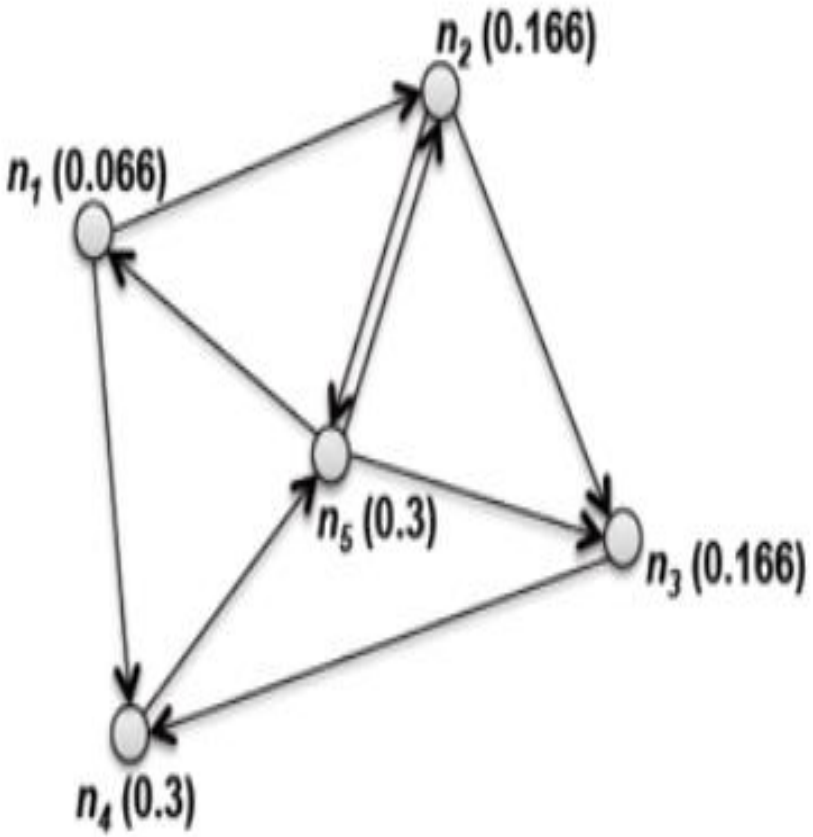
- PageRank of a page is based on the number of pages which link to it
  - But what if a page owner creates a swarm of dummy pages pointing to his website?
- PageRank of a page is based on the PageRank of the pages which link to it
  - A page divides its PageRank equally among all its outgoing links

$$P(n) = \sum_{m \in L(n)} \frac{P(m)}{C(m)}$$

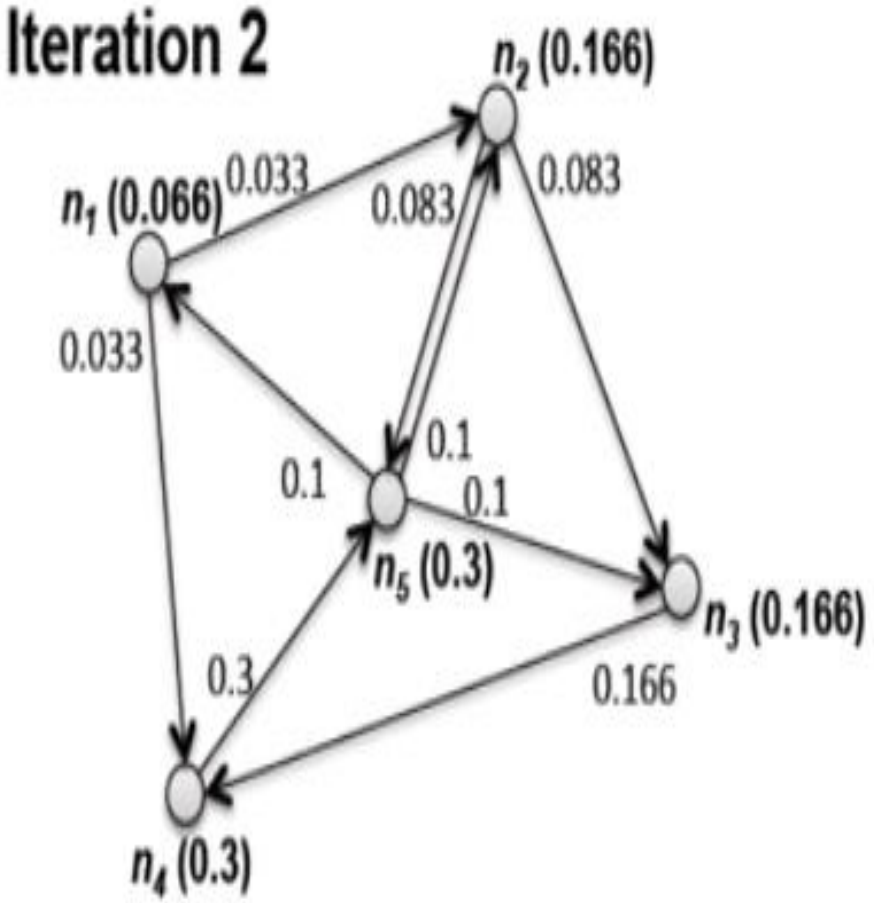
L(n) is the pages connected to n and C(m) is the number of pages that m is connected to

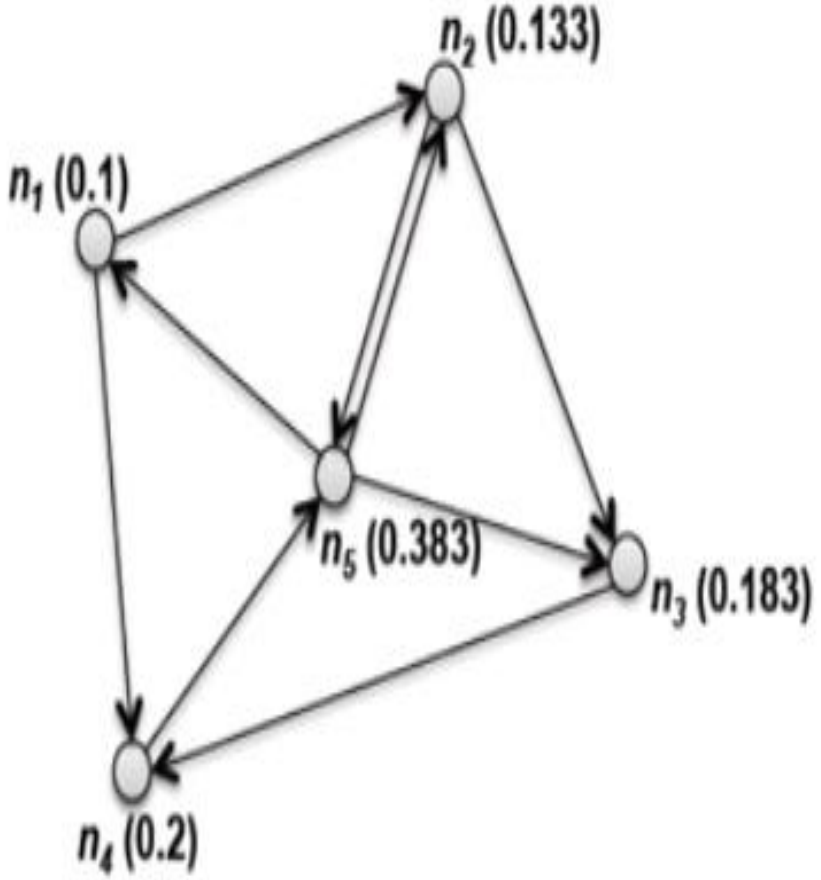
Example


Iteration 1
n_2 (0.2)
n_1 (0.2)
0.1
0.1
0.1
0.1
0.066
0.066
0.066
n_5 (0.2)
n_3 (0.2)
0.2
0.2
n_4 (0.2)


n_2 (0.166)
n_1 (0.066)
n_5 (0.3)
n_3 (0.166)
n_4 (0.3)

Example


Iteration 2
n_2 (0.166)
n_1 (0.066)
0.033
0.083
0.083
0.033
0.1
0.1
0.1
n_5 (0.3)
n_3 (0.166)
0.3
0.166
n_4 (0.3)


n_2 (0.133)
n_1 (0.1)
n_5 (0.383)
n_3 (0.183)
n_4 (0.2)

PageRank using MapReduce

- Full algorithm is iterative
- Initialize the nodes to uniform distribution
- Run the two MR jobs described iteratively
  - Map job emits page rank distributions continuously
  - Reduce job constantly collects these emitted page ranks and combines them for every node
- Until convergence (no change)
- At the end there is a clean up/normalization phase to correct the distributions

$$p' = \alpha \frac{1}{|G|} + (1 - \alpha)\left(\frac{m}{|G|} + p\right)$$

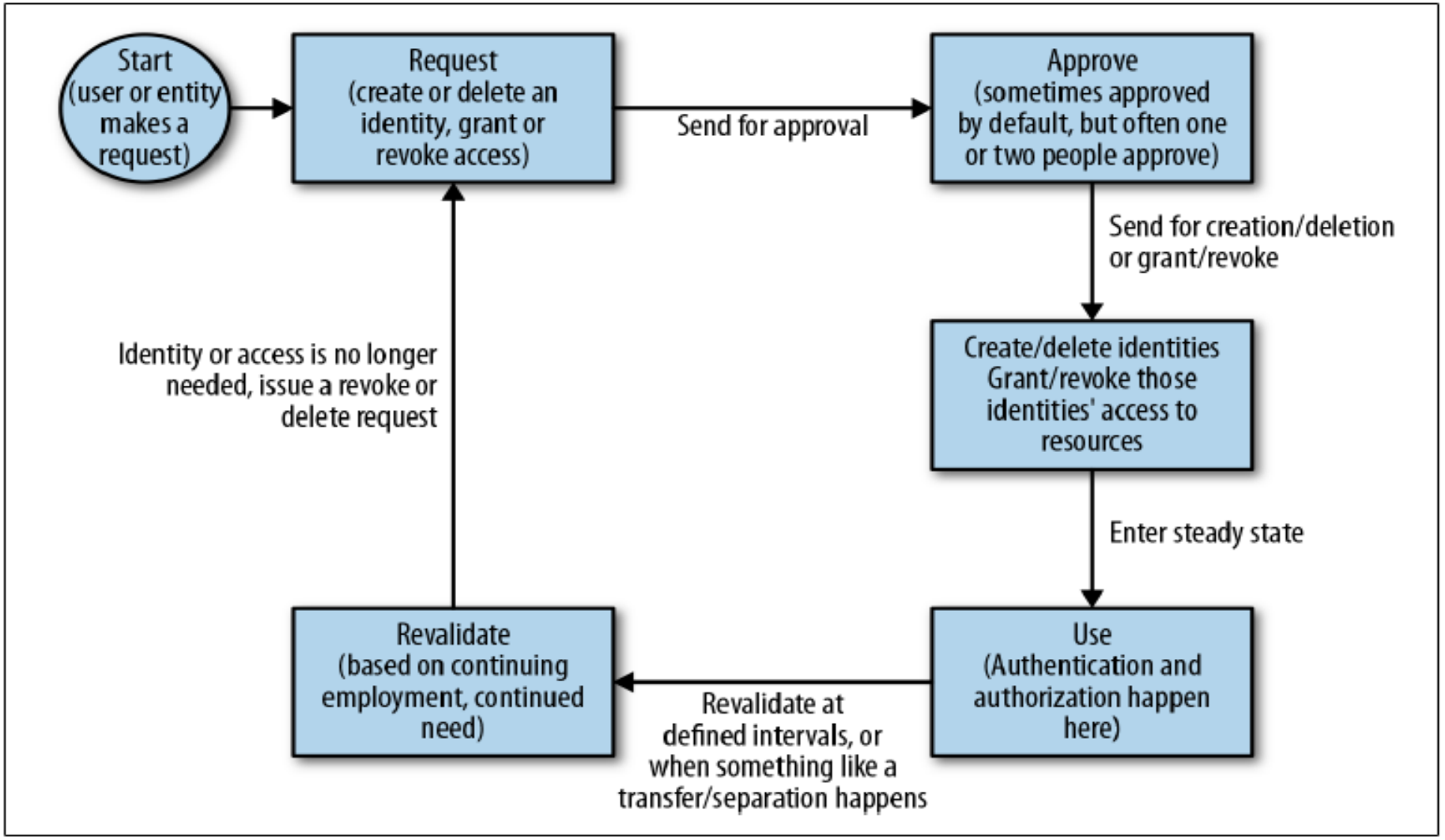
IAM in Cloud Computing

Dr. Amit Praseed

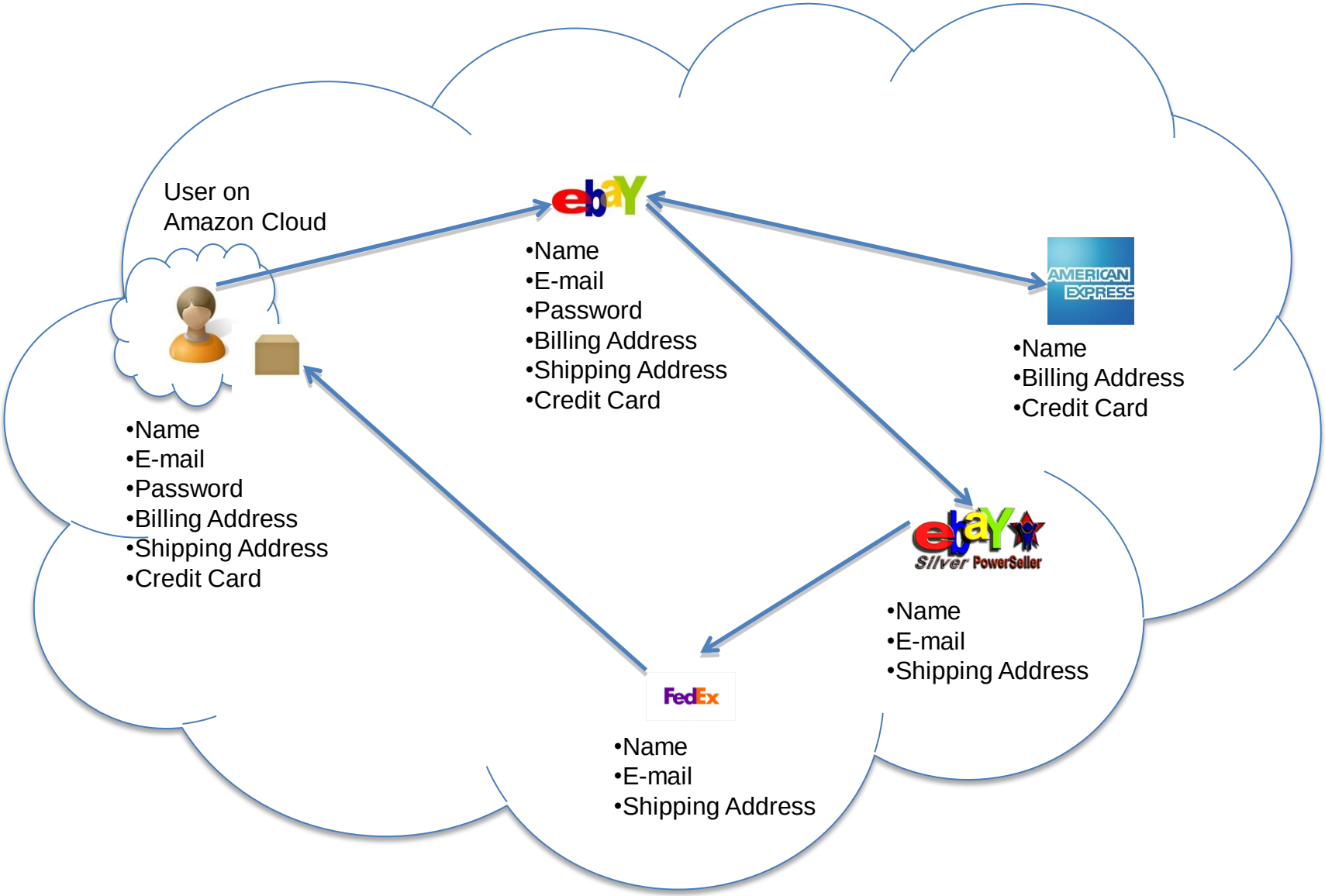
Identity and Access Management

- Each entity (such as a user, administrator, or system) needs an identity
  - The process of verifying that identity is called *authentication*
- Access management is about ensuring that entities can perform only the tasks they need to perform.
  - The process of checking what access an entity should have is called *authorization*

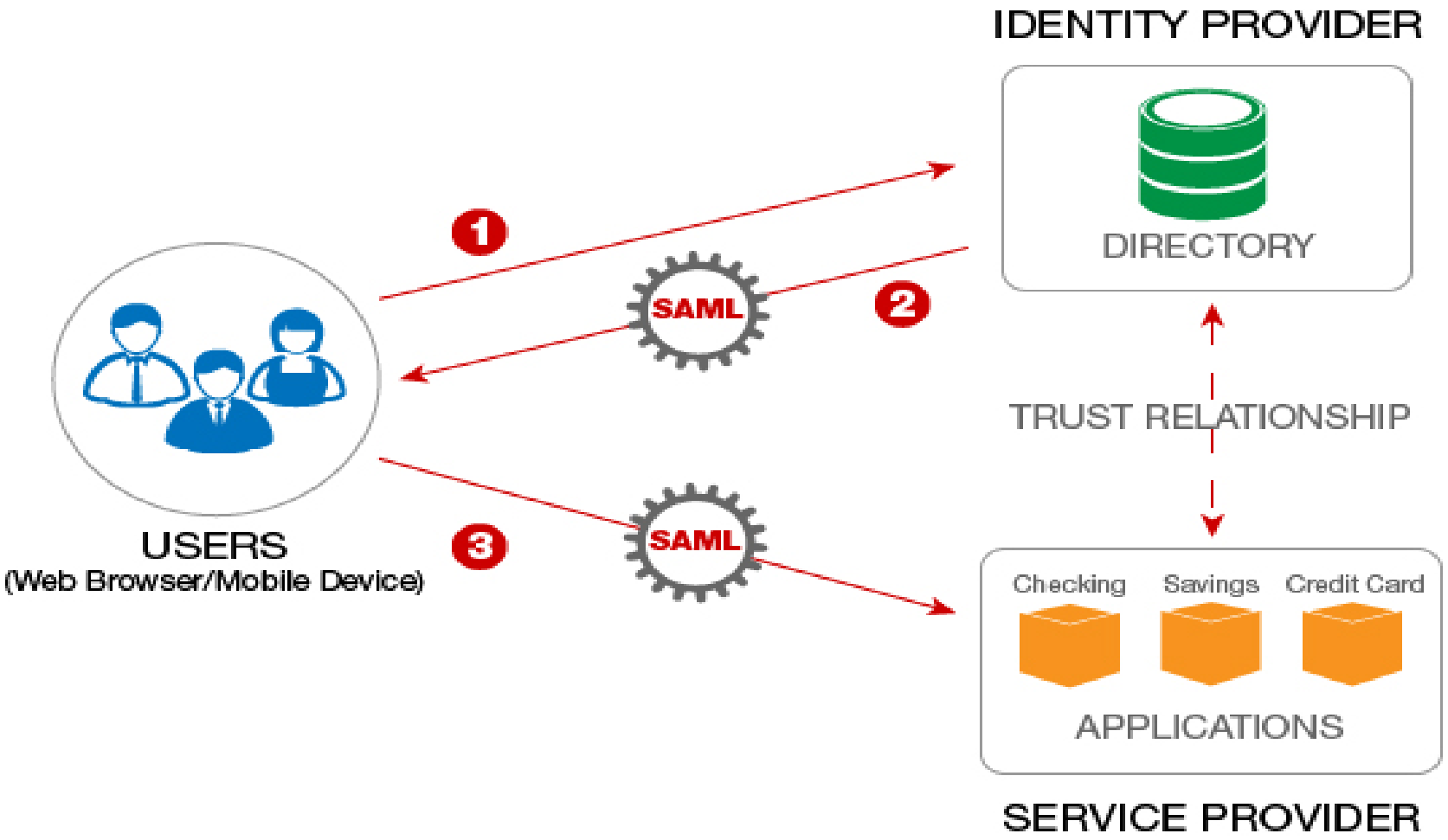
IAM Life Cycle



So many Identities!!!



SSO to the Rescue!



Protocols for SSO

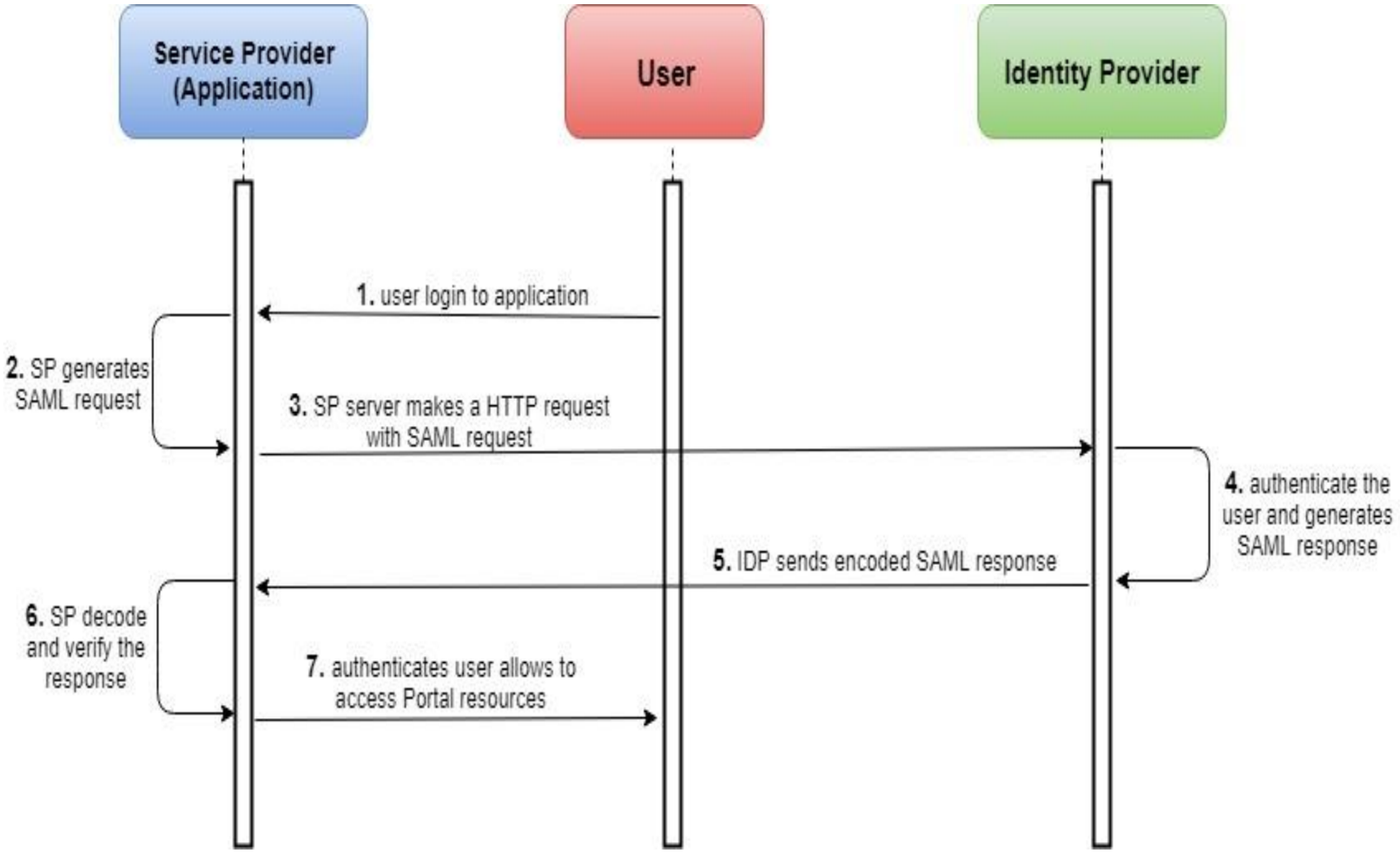
- There are three popular mechanisms that are used to provide SSO
  - Security Assertion Markup Language (SAML)
  - Open Authorization (OAuth)
  - OpenID

SAML

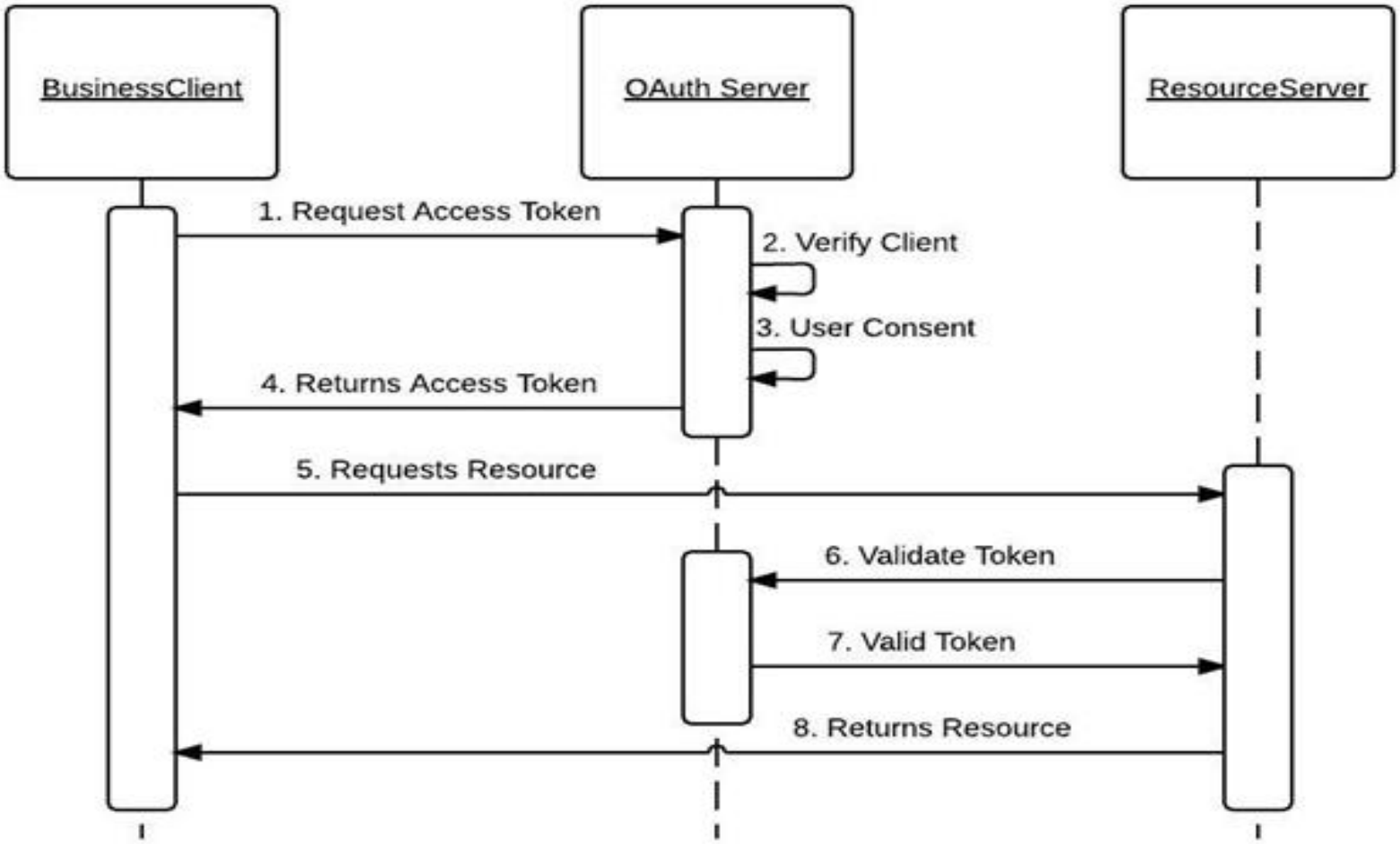
- SAML was developed in the early 2000s
  - define an XML framework for exchanging authentication and authorization information
  - allows a user's identity to be passed from one place to another with digitally signed XML (eXtensible Markup Language) documents.
  - SAML Info: Version, ID, ProviderName, IssueInstant, Destination, ProtocolBinding, AssertionConsumerServiceURL, and Issuer

```
<samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" ID="ONELOGIN_809707f0030a5d00620c9d9df97f627afe9dcc24"
Version="2.0" ProviderName="SP test" IssueInstant="2014-07-16T23:52:45Z"
Destination="http://idp.example.com/SSOService.php" ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
AssertionConsumerServiceURL="http://sp.example.com/demo1/index.php?acs">
    <saml:Issuer>http://sp.example.com/demo1/metadata.php</saml:Issuer>
    <samlp:NameIDPolicy Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress"
    AllowCreate="true" />
    <samlp:RequestedAuthnContext Comparison="exact">
        <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</saml:AuthnContextClassRef>
    </samlp:RequestedAuthnContext>
</samlp:AuthnRequest>
```

SAML Workflow



OAuth Workflow



SSO and Privacy

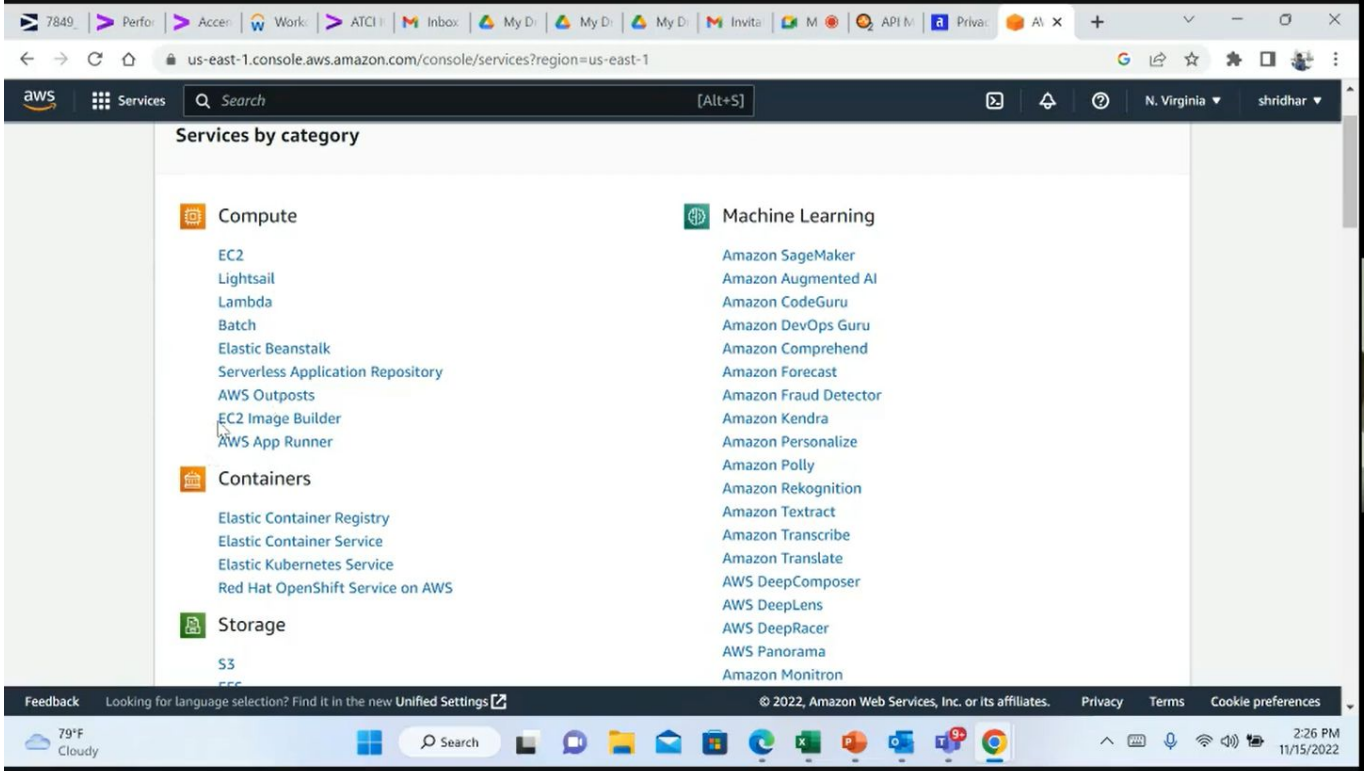
- Cloud introduces several issues to IDM
  - Collusion between Cloud Services
    - Users have multiple accounts associated with multiple service providers.
    - Sharing sensitive identity information between services can lead to undesirable mapping of the identities to the user.
  - Lack of trust
    - Cloud hosts are untrusted
    - Use of Trusted Third Party is not an option
  - Loss of control
    - Service-centric IDM Model

SSO and Privacy

- Anonymous Identification
  - Based on cryptographic zero knowledge proofs
- Multi party Authentication
  - Based on secure multi party computation
  - Information is distributed and managed by multiple identity providers, all of whom hold non-overlapping information
  - Unless *k* IdPs collude, user information cannot be effectively leaked

AWS Fundamentals and Why?

- AWS provides on-demand delivery of IT resources via the Internet on a secure cloud services platform, offering compute power, storage, databases, content delivery, and other functionality to help businesses scale and grow


us-east-1.console.aws.amazon.com/console/services?region=us-east-1
aws
Services
Search
[Alt+S]
N. Virginia
shridhar
Services by category
Compute
EC2
Lightsail
Lambda
Batch
Elastic Beanstalk
Serverless Application Repository
AWS Outposts
EC2 Image Builder
AWS App Runner
Containers
Elastic Container Registry
Elastic Container Service
Elastic Kubernetes Service
Red Hat OpenShift Service on AWS
Storage
S3
Machine Learning
Amazon SageMaker
Amazon Augmented AI
Amazon CodeGuru
Amazon DevOps Guru
Amazon Comprehend
Amazon Forecast
Amazon Fraud Detector
Amazon Kendra
Amazon Personalize
Amazon Polly
Amazon Rekognition
Amazon Textract
Amazon Transcribe
Amazon Translate
AWS DeepComposer
AWS DeepLens
AWS DeepRacer
AWS Panorama
Amazon Monitron
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
79°F
Cloudy
Search
2:26 PM
11/15/2022


Dr. Shridhar Domanal IBM


Global in minutes
Variable vs. capital expense
Stop guessing capacity
Six advantages
Economies of scale
Focus on business differentiators
Increase speed and agility

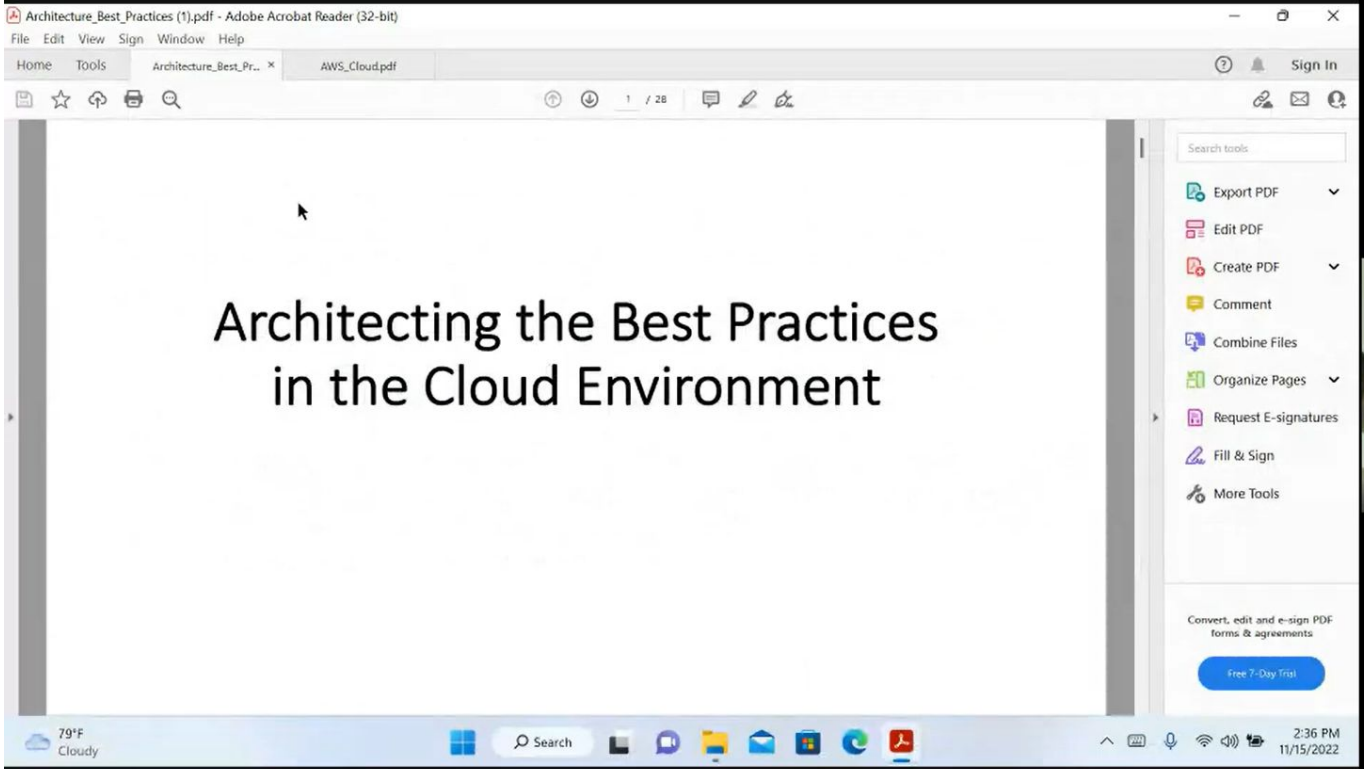

Dr. Shridhar Domanal IBM

Deployment Models

- All-in cloud-based
  - Is fully deployed in the cloud, with all components of the application running in the cloud
- Hybrid Model
  - Is a common approach taken by many enterprises that connects infrastructure and applications between cloud-based resources and existing resources, typically on an existing data center.

Global Infrastructure, Security and Compliance

- AWS serves over one million active customers in more than 190 countries, and it continues to expand its global infrastructure steadily to help organizations achieve lower latency and higher throughput for their business needs.
- AWS provides a highly available technology infrastructure platform with multiple locations worldwide
  - Regions and Availability Zones
- Service Organization Controls (SOC) 1/International Standard on Assurance Engagements (ISAE) 3402, SOC 2, and SOC 3
- Federal Information Security Management Act (FISMA), Department of Defense Information Assurance Certification and Accreditation Process (DIACAP), and Federal Risk and Authorization Management Program (FedRAMP)
- Payment Card Industry Data Security Standard (PCI DSS) Level 1
- International Organization for Standardization (ISO) 9001, ISO 27001, and ISO 27018


Architecture_Best_Practices (1).pdf - Adobe Acrobat Reader (32-bit)
File Edit View Sign Window Help
Home Tools
Architecture_Best_Pr...
AWS_Cloud.pdf
Sign In
1 / 28
Architecting the Best Practices
in the Cloud Environment
Search tools
Export PDF
Edit PDF
Create PDF
Comment
Combine Files
Organize Pages
Request E-signatures
Fill & Sign
More Tools
Convert, edit and e-sign PDF forms & agreements
Free 7-Day Trial
79°F
Cloudy
Search
2:36 PM
11/15/2022


Dr. Shridhar Domanal IBM

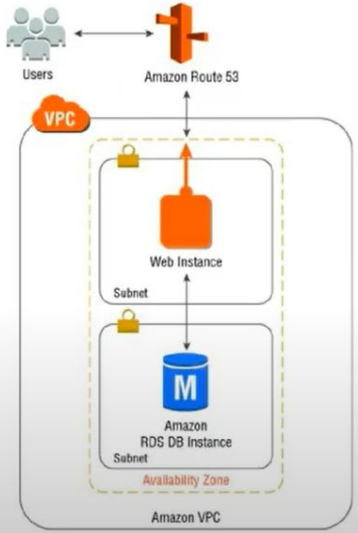
Best Practices

- Design for failure and nothing will fail.
- Implement elasticity.
- Leverage different storage options.
- Build security in every layer.
- Think parallel.
- Loose coupling sets you free.

CC Guest Lecture (2022-11-15 14:19 GMT+5:30)

1. Design for Failure and Nothing Fails

- Consider a typical Web Application with its components

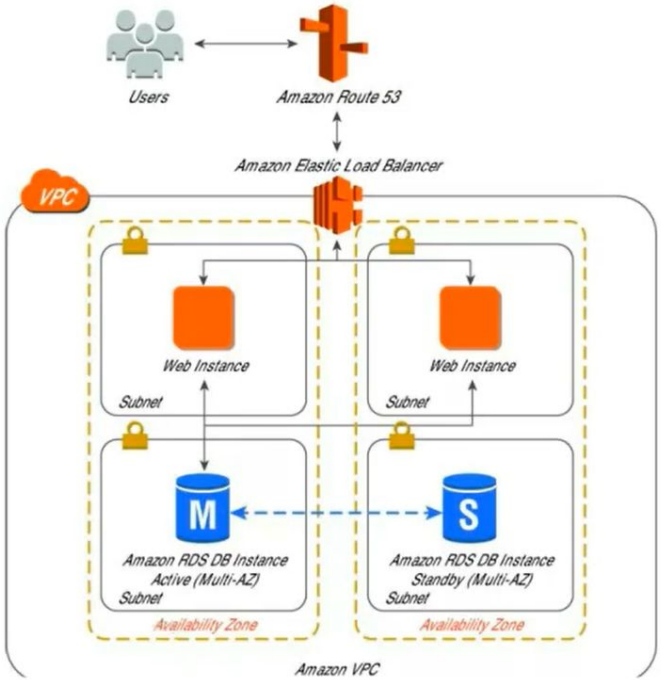


18:05 / 36:21

Demerits of the model and Remedy

- No redundancy or Failover which results in single point of failure.
  - If the single web server fails, the system fails.
  - If the single database fails, the system fails.
  - If the Availability Zone (AZ) fails, the system fails.
- Bottom line, there are too many eggs in one basket.
- Addressed by redundancy
  - Standby mode
  - Active mode

- Improved design with redundancy



2. Implement Elasticity

- Elasticity is the ability of a system to grow to handle increased load, whether gradually over time or in response to a sudden change in business needs.
- These architectures should provide scale in a linear manner, where adding extra resources results in at least a proportional increase in ability to serve additional system load.
- The growth in resources should introduce economies of scale, and cost should follow the same dimension that generates business value out of that system. How?

CC Guest Lecture (2022-11-15 14:19 GMT+5:30)

Scaling Vertically

- Vertical scaling takes place through an increase in the specifications of an individual resource (for example, upgrading a server with a larger hard drive, more memory, or a faster CPU).
- Amazon Elastic Compute Cloud (Amazon EC2), this can easily be achieved by stopping an instance and resizing it to an instance type that has more RAM, CPU, I/O, or networking capabilities.
- Easy to implement and mainly suited for short term goals.

30:53 / 36:21

Press Esc to exit full screen

Scaling Horizontally

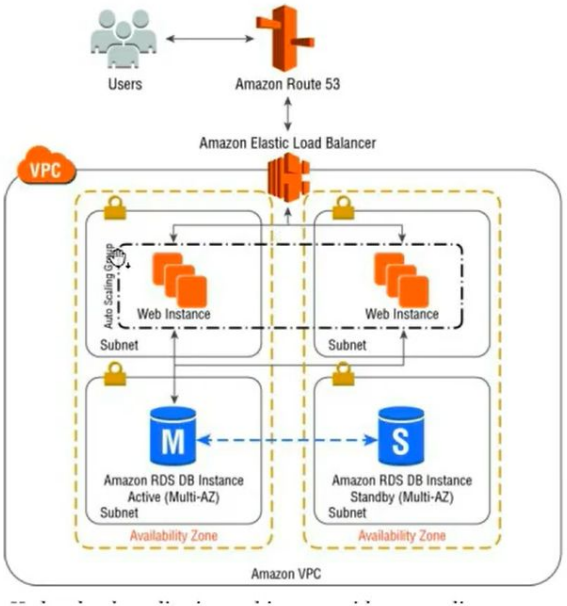
- Horizontal scaling takes place through an increase in the number of resources (for example, adding more hard drives to a storage array or adding more servers to support an application).
- Not all architectures are designed to distribute their workload to multiple resources, and it is important to understand system characteristics that can affect a system's ability to scale horizontally.
  - System characteristics are mainly depend on the type of the application
    - Stateless and statefull

CC Guest Lecture (2022-11-15 14:19 GMT+5:30)

Stateless Features

- A stateless application needs no knowledge of the previous interactions and stores no session information.
- A stateless application can scale horizontally, because any request can be serviced by any of the available system compute resources.
- When excess capacity is no longer required, any individual resource can be safely terminated.
  - For the previous example: A great way to introduce elasticity and horizontal scaling is by leveraging Auto Scaling for web instances. An Auto Scaling group can automatically add Amazon EC2 instances to an application in response to heavy traffic and remove them when traffic slows.

30:13 / 36:21


Users
Amazon Route 53
Amazon Elastic Load Balancer
VPC
Auto Scaling Group
Web Instance
Web Instance
Subnet
Subnet
M
S
Amazon RDS DB Instance
Active (Multi-AZ)
Subnet
Amazon RDS DB Instance
Standby (Multi-AZ)
Subnet
Availability Zone
Availability Zone
Amazon VPC


Dr. Shridhar Domanal IBM

Press Esc to exit full screen

Example

- For example, web applications can use HTTP cookies to store information about a session at the client's browser (such as items in the shopping cart). The browser passes that information back to the server at each subsequent request so that the application does not need to store it. However, there are two drawbacks.
  - The content of the HTTP cookies can be tampered with at the client side, so you should always treat them as untrusted data that needs to be validated.
  - HTTP cookies are transmitted with every request, which means that you should keep their size to a minimum to avoid unnecessary latency.

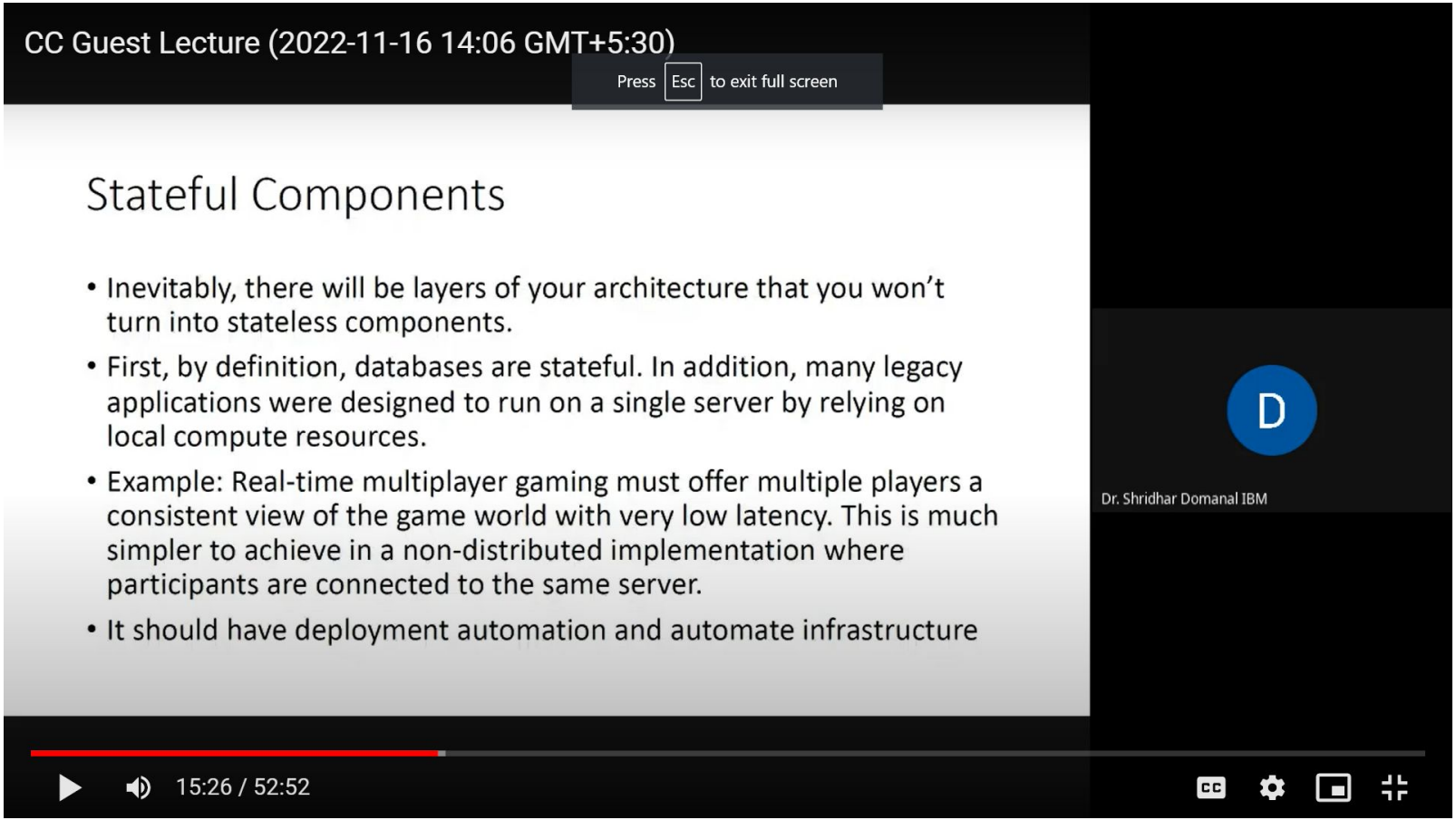
Dr. Shridhar Domanal IBM

Press Esc to exit full screen

Solution

- Consider only storing a unique session identifier in a HTTP cookie and storing more detailed user session information server-side. Most programming platforms provide a native session management mechanism that works this way; however, these management mechanisms often store the session information locally by default. This would result in a stateful architecture.
- A common solution to this problem is to store user session information in a database. Amazon DynamoDB is a great choice due to its scalability, high availability, and durability characteristics. For many platforms, there are open source, drop-in replacement libraries that allow you to store native sessions in Amazon DynamoDB.

Dr. Shridhar Domanal IBM


CC Guest Lecture (2022-11-16 14:06 GMT+5:30)
Press Esc to exit full screen
Stateful Components
• Inevitably, there will be layers of your architecture that you won't turn into stateless components.
• First, by definition, databases are stateful. In addition, many legacy applications were designed to run on a single server by relying on local compute resources.
• Example: Real-time multiplayer gaming must offer multiple players a consistent view of the game world with very low latency. This is much simpler to achieve in a non-distributed implementation where participants are connected to the same server.
• It should have deployment automation and automate infrastructure
D
Dr. Shridhar Domanal IBM
15:26 / 52:52

Bootstrap the Instances

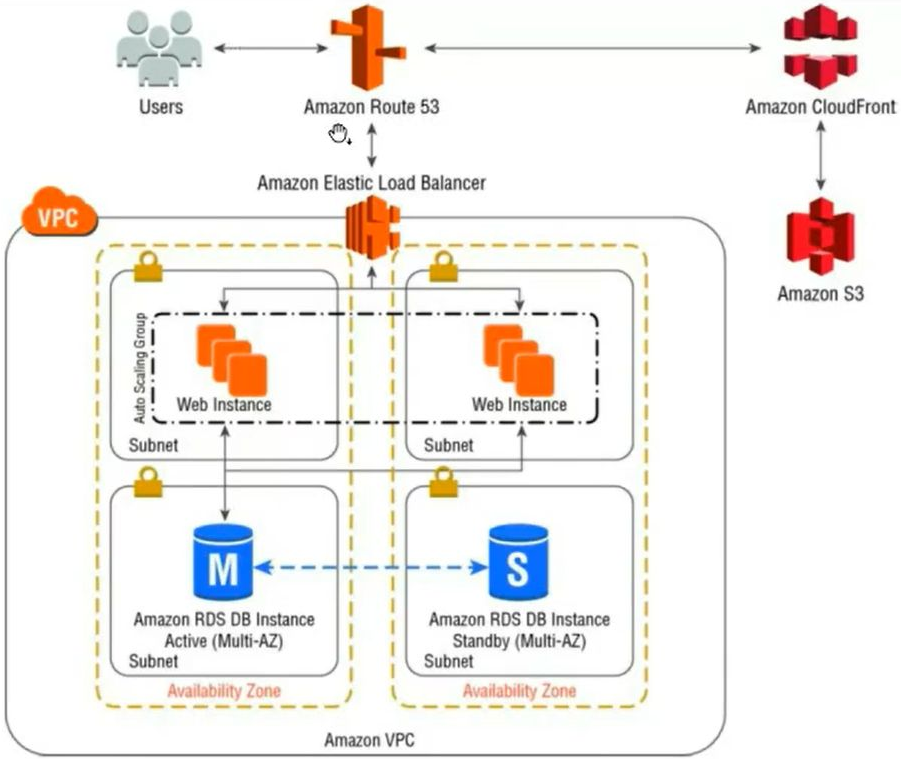
- Every instance should have a role to play in the environment (such as database server, application server, or slave server in the case of a web application).
- Advantages
  - Recreate environments (for example, development, staging, production) with few clicks and minimal effort.
  - Maintain more control over your abstract, cloud-based resources.
  - Reduce human-induced deployment errors.
  - Create a self-healing and self-discoverable environment that is more resilient to hardware failure.

3. Leverage Different Storage Options

- AWS offers a broad range of storage choices for backup, archiving, and disaster recovery, as well as block, file, and object storage to suit different use cases.
- For example, services like Amazon Elastic Block Storage (Amazon EBS), Amazon S3, Amazon RDS, and Amazon CloudFront provide a wide range of choices to meet different storage needs.
- It is important from a cost, performance, and functional aspect to leverage different storage options available in AWS for different types of datasets.
- One Size Does Not Fit All. How can we decide the suitable ones?

Dr. Shridhar Domanal IBM

Sample Scenario	Storage Option
Your web application needs large-scale storage capacity and performance.	
-or-	Amazon S3
You need cloud storage with high data durability to support backup and active archives for disaster recovery.	
You require cloud storage for data archiving and long-term backup.	Amazon Glacier
You require a content delivery network to deliver entire websites, including dynamic, static, streaming, and interactive content using a global network of edge locations.	Amazon CloudFront
You require a fast and flexible NoSQL database with a flexible data model and reliable performance.	Amazon DynamoDB
You need reliable block storage to run mission-critical applications such as Oracle, SAP, Microsoft Exchange, and Microsoft SharePoint.	Amazon EBS
You need a highly available, scalable, and secure MySQL database without the time-consuming administrative tasks.	Amazon RDS
You need a fast, powerful, fully-managed, petabyte-scale data warehouse to support business analytics of your e-commerce application.	Amazon Redshift
You need a Redis cluster to store session information for your web application.	Amazon ElastiCache
You need a common file system for your application that is shared between more than one Amazon EC2 instance.	Amazon Elastic File System (Amazon EFS)

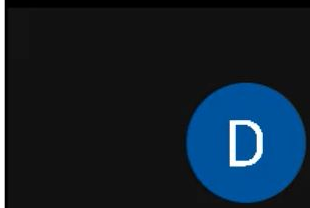


D

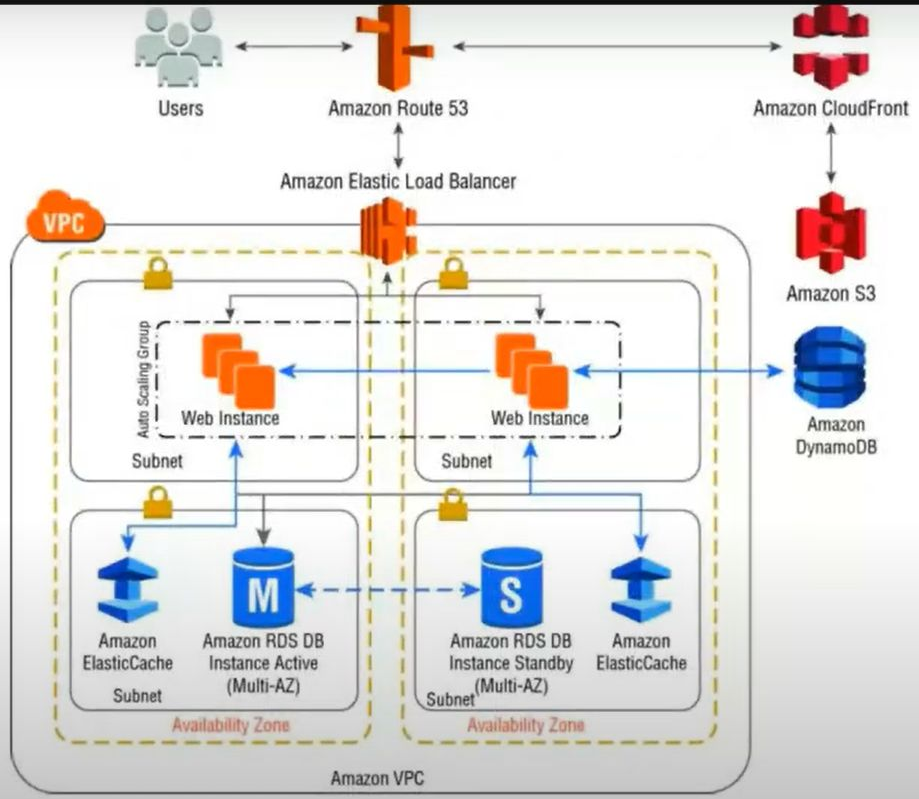
Dr. Shridhar Domanal IBM

Further Optimization is possible?

- For our scenario, we will use Amazon DynamoDB to store the session information because the AWS Software Development Kits (SDK) provide connectors for many popular web development frameworks that make storing session information in Amazon DynamoDB easy
- By removing session state from our web tier, the web instances do not lose session information when horizontal scaling from Auto Scaling happens. Additionally, we will leverage Amazon ElastiCache to store common database query results, thereby taking the load off of our database tier.



CC Guest Lecture (2022-11-16 14:06 GMT+5:30)



D

Dr. Shridhar Domanal IBM

22:04 / 52:52

CC Guest Lecture (2022-11-16 14:06 GMT+5:30)

4. Build Security in Every Layer

Dr. Shridhar Domanal IBM

22:47 / 52:52

Press Esc to exit full screen

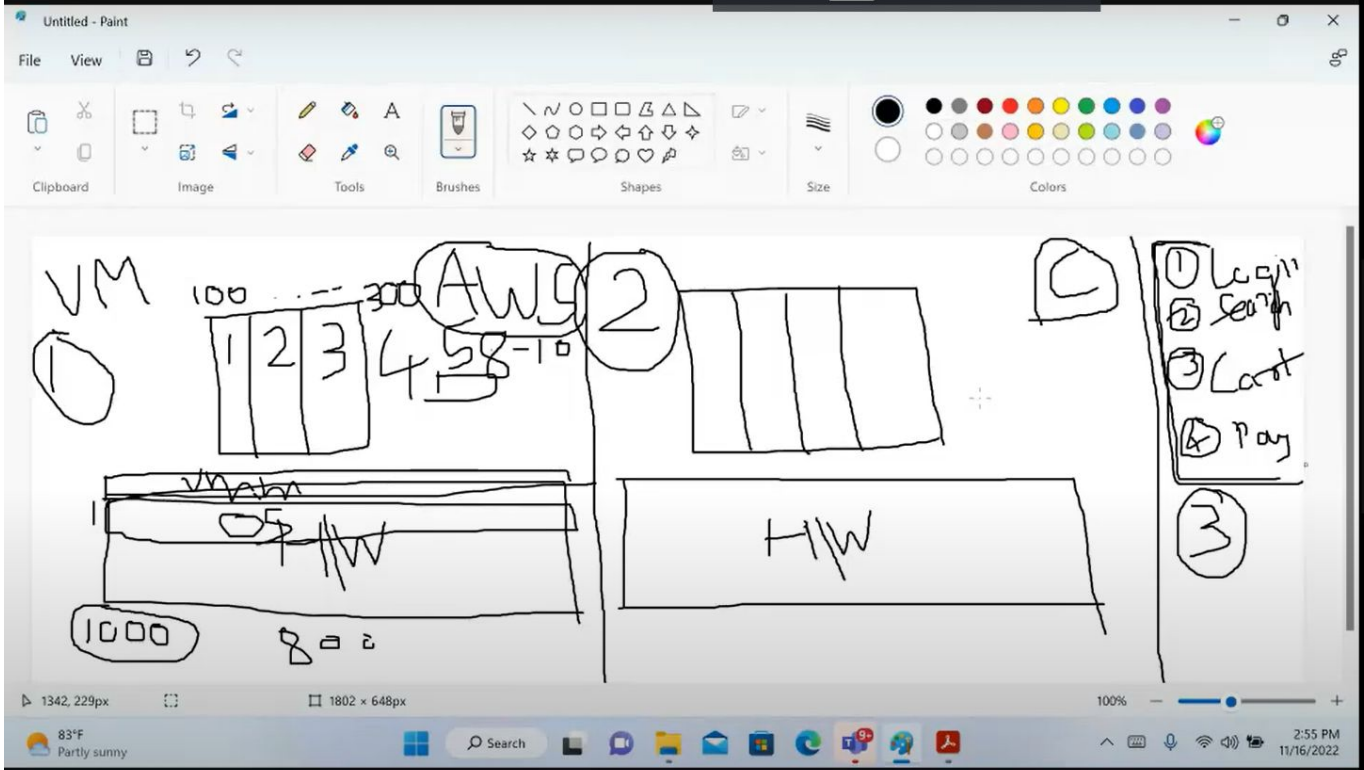
Use AWS Features for Defense in Depth

- Build an Amazon Virtual Private Cloud (Amazon VPC) topology that isolates parts of the infrastructure through the use of subnets, security groups, and routing controls.
- Services like AWS Web Application Firewall (AWS WAF) can help protect your web applications from SQL injection and other vulnerabilities in your application code.
- For access control, you can use AWS Identity and Access Management (IAM) to define a granular set of policies and assign them to users, groups, and AWS resources.
- Finally, the AWS platform offers a breadth of options for protecting data with encryption, whether the data is in transit or at rest.

Dr. Shridhar Domanal IBM

CC Guest Lecture (2022-11-16 14:06 GMT+5:30)

Press Esc to exit full screen



D

Dr. Shridhar Domanal IBM

48:43 / 52:52

VM Security in Cloud Computing

Dr. Amit Praseed

Managing Compute Assets

- Compute assets typically take data, process it, and do something with the results.
  - For example, a very simple compute resource might take data from a database and send it to a web browser on request, or send it to a business partner, or combine it with data in another database.
- Compute resources may also store data, particularly temporary data.
  - With some types of regulated data, it may be necessary to ensure that you're tracking every place that data could be

Detecting Virtualization

- Can a malicious entity detect whether it is operating in a virtualized environment?
  - Malware was designed to lay dormant on virtualized environments
  - With the spread of virtualization, malware is now configured to detect virtualization and exploit hypervisor specific vulnerabilities
    - VMWare “get version” command
    - Resource discrepancies
    - Timing discrepancies

VM Security

- VMs share the same *physical* system with other cloud customers.
  - "noisy neighbor" problems : using up all of the processor time, network bandwidth, or storage bandwidth
- Two types of attacks against VMs
  - Hypervisor Breakout / VM Escape
  - Side Channel Attacks

Hypervisor Breakout

- An attacker runs code on a virtual machine that allows an operating system running inside the hypervisor to break out and interact directly with it.
- This type of attack could allow the attacker access to the host operating system as well as all virtual machines running on that host.
- Several techniques are available for launching these attacks

Side Channels in the Cloud

- Hypervisors are meant to isolate virtual machines
- Side channel attacks use a medium that is not explicitly meant for communication
  - Eg: CPU data caches
- Co-residence is often an essential condition for executing these attacks
- Verifying co-residence
  - Same Dom0 IP
  - Small packet RTT
  - Numerically close IP addresses

Side Channels in the Cloud

- A malicious instance can utilize side channels to learn information about co-resident instances
  - time-shared caches allow an attacker to measure when other instances are experiencing computational load
  - any physical machine resources multiplexed between the attacker and target forms a potentially useful channel: network access, CPU branch predictors and instruction cache, DRAM memory bus, CPU pipelines , scheduling of CPU cores and timeslices, disk access etc

Side Channels in the Cloud

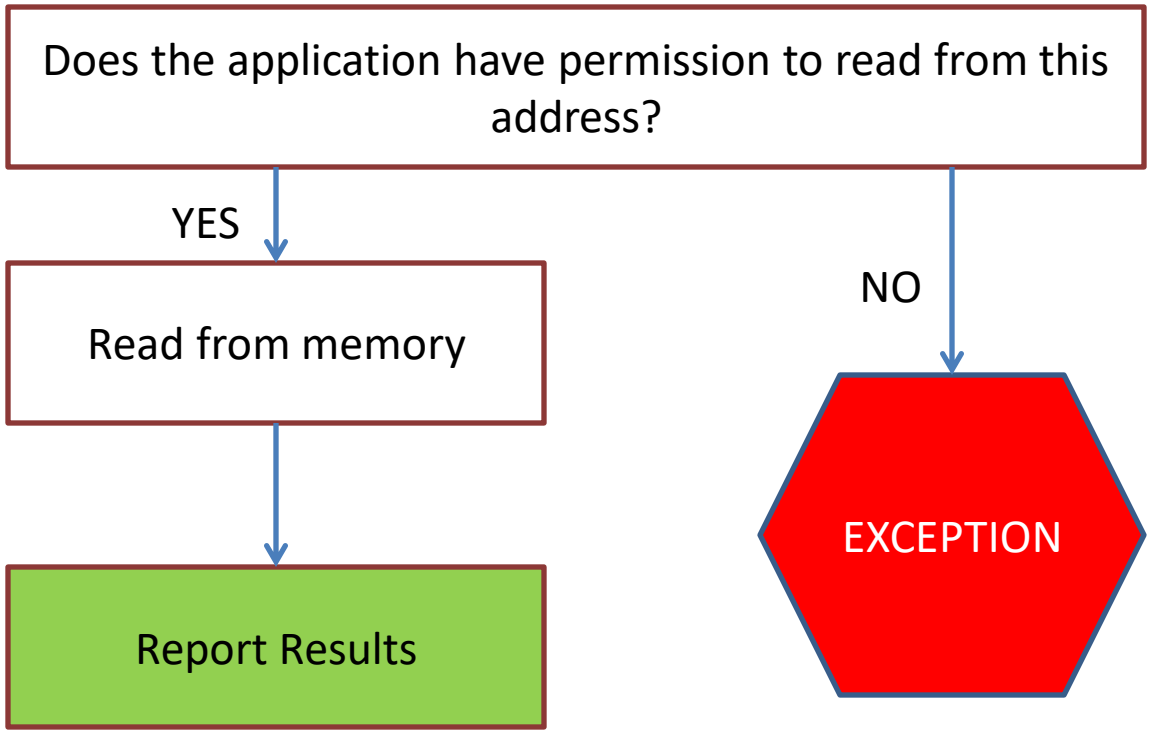
- An attacking instance can measure the utilization of CPU caches on its physical machine.
  - These measurements can be used to estimate the current load of the machine
  - Variants of Flush + Reload
    - Flush the cache lines
    - Wait for victim program to run
    - Access the cache and check the timing
      - Less time ➔ victim accessed the cache line
      - More time ➔ victim did not access the cache line

Side Channel Attacks

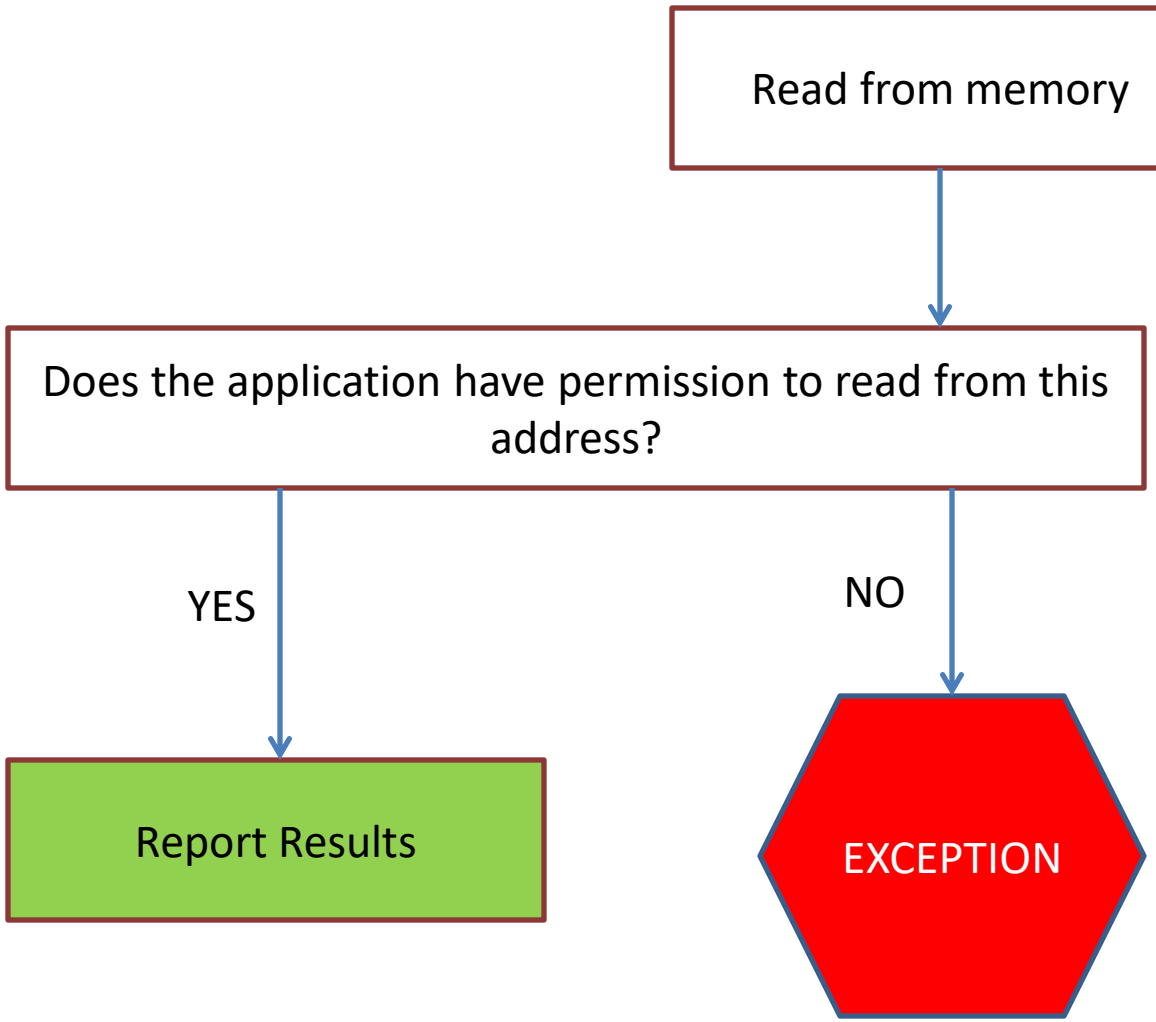
- The attacker tries to find any relation between an encryption process and accessed cache lines
  - To derive a profile of cache activities, the attacker manipulates cache by evicting memory lines of victim process.
- The attacker observes cache activities, i.e., memory lines which are accessed by an encryption process during its execution to obtain a sequence of cache hits and cache misses
  - Eg: observing which memory accesses to a lookup table lead to cache hits allows disclosing indices in the lookup table.
  - After capturing enough samples an offline phase permits to infer the secret data that is used by the victim process.

Spectre and Meltdown

- Side channel attacks exploiting speculative execution



How Meltdown Works



- **By the time the exception is raised, the secret data is already loaded into the cache!!**
- **An attacker can use the Flush + Reload technique to read this data!!**

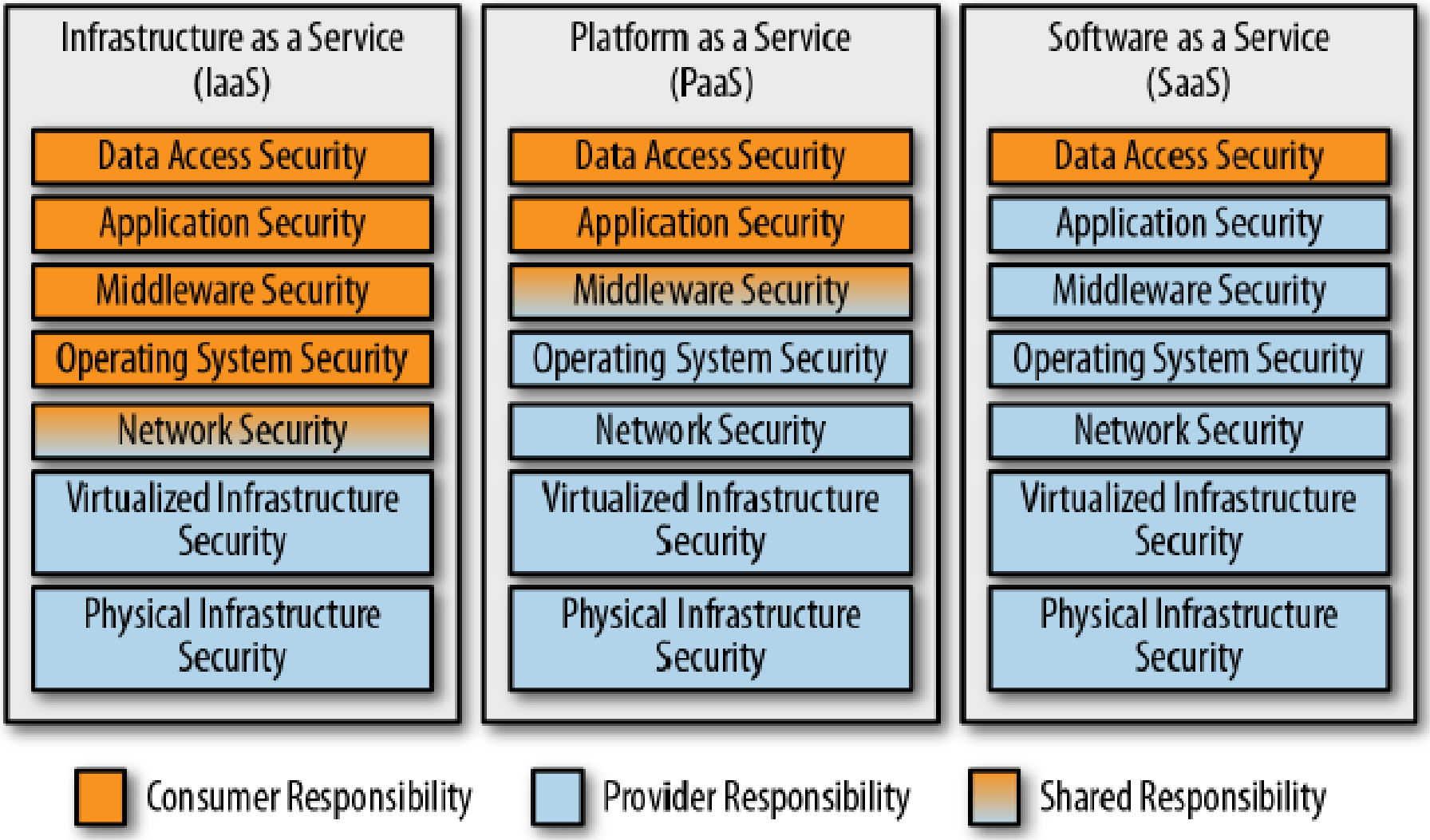
Preventing Side Channel Attacks

- **Time-padding** ensures that the execution time of a protected function is independent of the function input secret data.
  - Padding prevents an attacker from measuring the execution time of the function.
- **Cache cleansing** prevents obtaining the state of the cache after running the sensitive function.
- **Cache partitioning** allows protecting resources of a trusted process from being accessed by an untrusted process during its execution

Security in Cloud Computing

Dr. Amit Praseed

Whose Responsibility is it?



Whose Responsibility is it?

- If the provider offers virtualized environments, the virtualized infrastructure security controls keeping your virtual environment separate from other virtual environments are the provider's responsibility.
  - Spectre and Meltdown vulnerabilities (2018)
- Operating system security is usually straightforward:
  - Your responsibility if you're using IaaS
  - Provider's responsibility if you're purchasing platform or software
- **If you have the ability to break it, you usually have the responsibility for securing it!**

Whose Responsibility is it?

- Root cause of Cloud Security issues is an assumption that the cloud provider is handling something, when it turns out *nobody* was handling it.
- AWS S3 storage is secure and encrypted, but none of that helps if you don't set your access controls properly.
  - Data on 198 million US voters
  - Auto-tracking company records
  - Wireless customer records
  - Over 3 million demographic survey records
  - Over 50,000 Indian citizens' credit reports
- Misunderstandings and Misinformation
  - 77% of IT decision makers believe that public cloud providers were responsible for securing customer data in the cloud
  - 68% said they believed these providers were responsible for securing customer applications as well

Data Asset Management

- Classify your data – low, medium and high security
  - Use tagging to keep track of data
- Understand security regulations and compliance
  - EU GDPR
  - US FISMA
  - Global PCI DSS

Cloud Data Protection

- Tokenization
  - store something that functions similarly to the data but is useless to an attacker
  - Eg: credit card numbers - replace a piece of sensitive data with a token
  - Token generally has the same characteristics as the original data, so underlying systems that are built to take that data don't need to be modified
  - Only one place (a "token service") knows the actual sensitive data.

Cloud Data Protection

- Encryption
  - Data can be in three states: in motion, use or rest
  - Encrypting Data in Use
    - Relatively new concept
    - requires support in the hardware platform, and it must be exposed by the cloud provider
    - encrypt process memory so that even a privileged cannot read it, and the processor can read it only when that specific process is running
    - Eg: Intel SGX, AMD SME, and IBM Z Pervasive Encryption.

Cloud Data Protection

- Encrypting Data at Rest
  - once you've encrypted the data, you now have an encryption key that can be used to access it
  - Hardware security module (HSM) to hold your encryption keys, usually in the form of an expansion card or a module accessed over the network
  - key management service (KMS), a multitenant service that uses an HSM on the backend to keep keys safe

Issues with KMS

- Simple Approach:
  - Use key management is to generate a key, encrypt the data with that key, stuff the key into the KMS, and then write the encrypted data to disk along with a note indicating which key was used to encrypt it
  - Problems?
    - Load on the KMS – too many keys
    - Erasure of Data
      - Delete the key – have to trust the KMS
      - Overwrite your data – time consuming

Issues with KMS

- Maintaining two keys
  - Data Encryption Key and Key Encryption Key
  - the key encryption key is used to encrypt (or “wrap”) data encryption keys, and the wrapped keys are stored right next to the data.
  - The key encryption key usually stays in the KMS and never comes out, for safety.
  - The wrapped data encryption keys are sent to the HSM for unwrapping when needed, and then the unwrapped keys are used to encrypt or decrypt the data
  - Delete the data? Delete the data encryption key!

Server-side and Client-side encryption

- Server Side Encryption
  - The storage service will automatically create data encryption keys, wrap them using a key encryption key that you can manage in the KMS, and store the wrapped keys along with the data.
  - Multitenant storage service does have the ability to decrypt your data!!!
- Client Side Encryption
  - Encryption and Decryption handled by client
  - No server-side searches, calculation, indexing, malware scans, or other high-value tasks can be performed

Homomorphic Encryption

- Homomorphic encryption is a method of encryption that allows any data to remain encrypted while it's being processed and manipulated.
  - Enables you or a third party (such as a cloud provider) to apply functions on encrypted data without needing to reveal the values of the data.
- Uses a public key to encrypt data and allows only the individual with the matching private key to access its unencrypted data
  - It uses an algebraic system to allow you or others to perform a variety of computations (or operations) on the encrypted data.

Homomorphic Encryption can solve Real World Problems!!!

- **Securing Data Stored in the Cloud**
- **Enabling Data Analytics in Regulated Industries**
- **Improving Election Security and Transparency**

Types of Homomorphic Encryption

- **Partially homomorphic encryption (PHE)**
  - allows select mathematical functions to be performed on encrypted values
  - one operation can be performed an unlimited number of times on the ciphertext.
  - Some examples of PHE include ElGamal encryption (a multiplication scheme) and Paillier encryption (an addition scheme).
- **Somewhat homomorphic encryption (SHE)**
  - supports limited operations (for example, either addition *or* multiplication) up to a certain complexity
  - These operations can only be performed a set number of times.
- **Fully homomorphic encryption (FHE)**
  - still in the development stage
  - capable of using any efficiently computable functions (such as addition *and* multiplication, not just one or the other) any number of times
  - makes secure multi-party computation more efficient.
  - Practically extremely slow