IOWA

CS5630

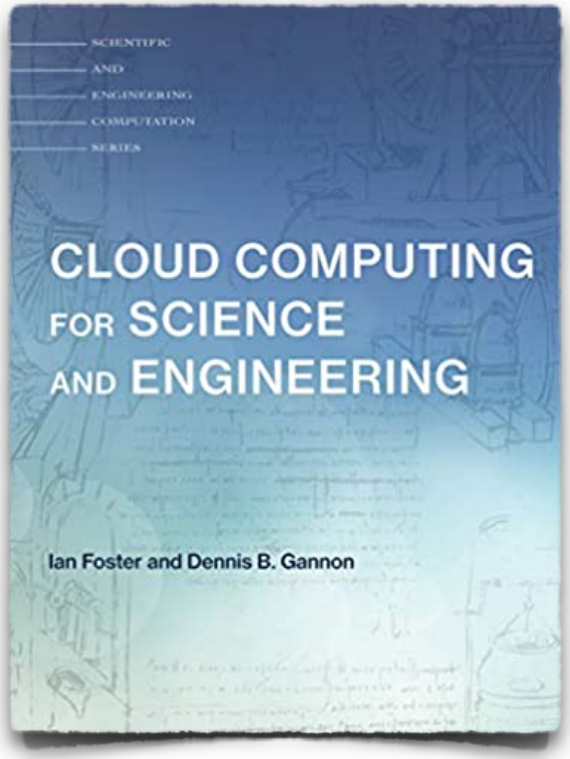
Foundation (2): Data in the Cloud

Prof. Supreeth Shastri
Computer Science
The University of Iowa

Lecture goals

Technical introduction to managing data on the cloud

- *Abstractions/models for data storage*
- *Data storage on cloud platforms*
- *Illustrative examples from AWS*



Chapters 2, 3

aws

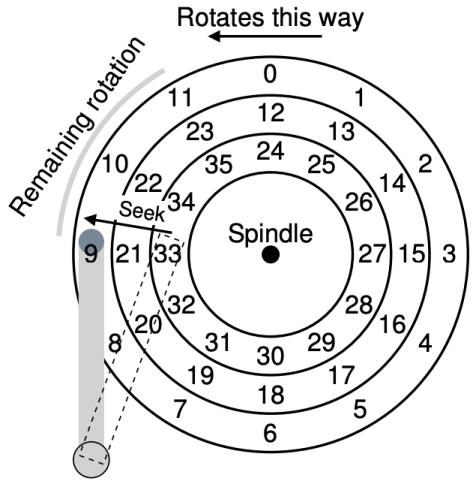
Google Cloud

IOWA

1. Block Storage

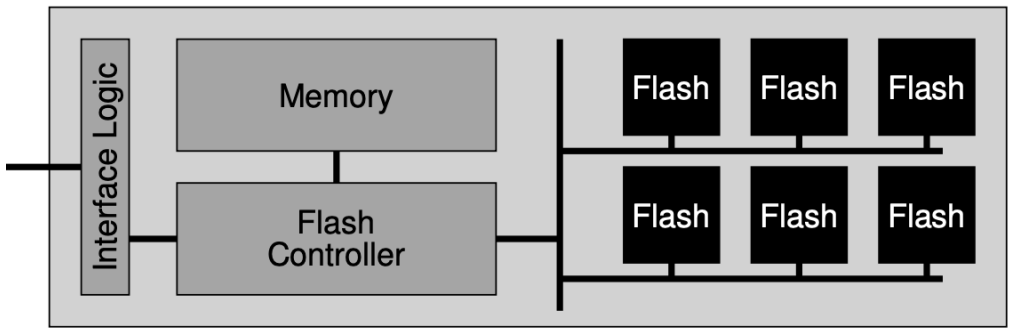
- Devices from which blocks of data can be read or written
- Operating systems rely on block storage for data persistence
- Hard disk drives (left) offer *inexpensive* storage (10x cheaper than SSD)
- Solid state devices (right) offer *highly performant* storage (2x - 100x more throughput than HDD)

HDD organization



$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

SSD architecture



2. File Systems

- System software responsible for storing and retrieving related groups of data (a.k.a. *files, directories*)
- Exposes APIs to create, populate, modify, and delete for directories, files, and their contents
- Examples of disk file systems: *FAT (Windows), EXT (Linux), APFS (Apple)*
- There are a variety of specialized file systems. For e.g., *Network file system, Lustre for HPC*

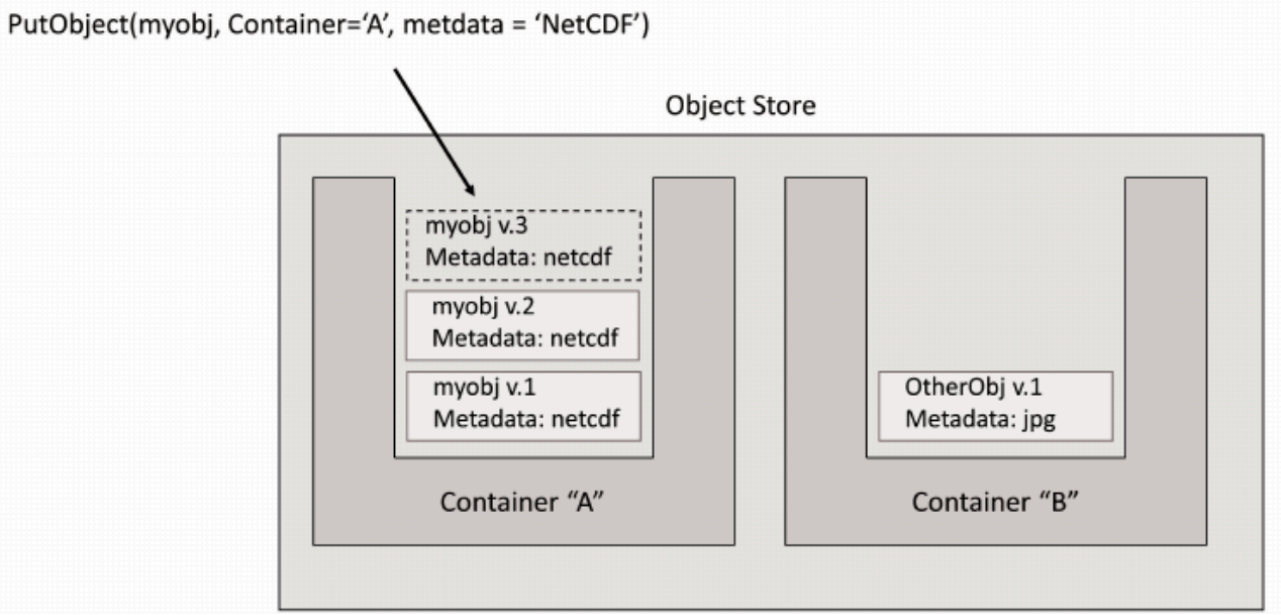
File and directory operations in Linux VFS

```
int (*lseek) (struct inode *, struct file *, off_t, int);
int (*read) (struct inode *, struct file *, char *, int);
int (*write) (struct inode *, struct file *, const char *, int);
int (*readdir) (struct inode *, struct file *, void *, filldir_t);
int (*select) (struct inode *, struct file *, int, select_table *);
int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
int (*mmap) (struct inode *, struct file *, struct vm_area_struct *);
int (*open) (struct inode *, struct file *);
void (*release) (struct inode *, struct file *);
int (*fsync) (struct inode *, struct file *);
int (*fasync) (struct inode *, struct file *, int);
int (*check_media_change) (kdev_t dev);
int (*revalidate) (kdev_t dev);
```

```
int (*create) (struct inode *,const char *,int,int,struct inode **);
int (*lookup) (struct inode *,const char *,int,struct inode **);
int (*link) (struct inode *,struct inode *,const char *,int);
int (*unlink) (struct inode *,const char *,int);
int (*symlink) (struct inode *,const char *,int,const char *);
int (*mkdir) (struct inode *,const char *,int,int);
int (*rmdir) (struct inode *,const char *,int);
int (*mknod) (struct inode *,const char *,int,int,int);
int (*rename) (struct inode *,const char *,int,struct inode *,const char *,int);
int (*readlink) (struct inode *,char *,int);
int (*follow_link) (struct inode *,struct inode *,int,int,struct inode **);
int (*readpage) (struct inode *, struct page *);
int (*writepage) (struct inode *, struct page *);
int (*bmap) (struct inode *,int);
void (*truncate) (struct inode *);
int (*permission) (struct inode *, int);
int (*smap) (struct inode *,int);
```

3. Object Stores

- Unlike file systems, these store *unstructured and immutable* binary objects
- Each object is identified by a unique ID, and can have metadata associated with it
- Objects can be deleted but all further modifications will be stored as different versions
- Why object stores? its simplified model offers *better scalability* (compared to file systems)
- However, object stores *do not help in data organization/search*; nor can OSes use them in lieu of file systems



4. Databases

- Database is simply a structured collection of data
- A database management system (DBMS) coordinates all operations on the data
- Database systems offer scalability and varying levels of operational guarantees (e.g., ACID)
- Classified based on the data model (Relational and NoSQL), on underlying hardware (distributed, embedded, in-memory, etc.), on purpose (graph, time-series, geospatial), among others

An example SQL query

```
select experiment-id from Experiments, People
where Experiments.person-id = People.person-id
      and People.name = "Smith";
```

5. Data Warehouses

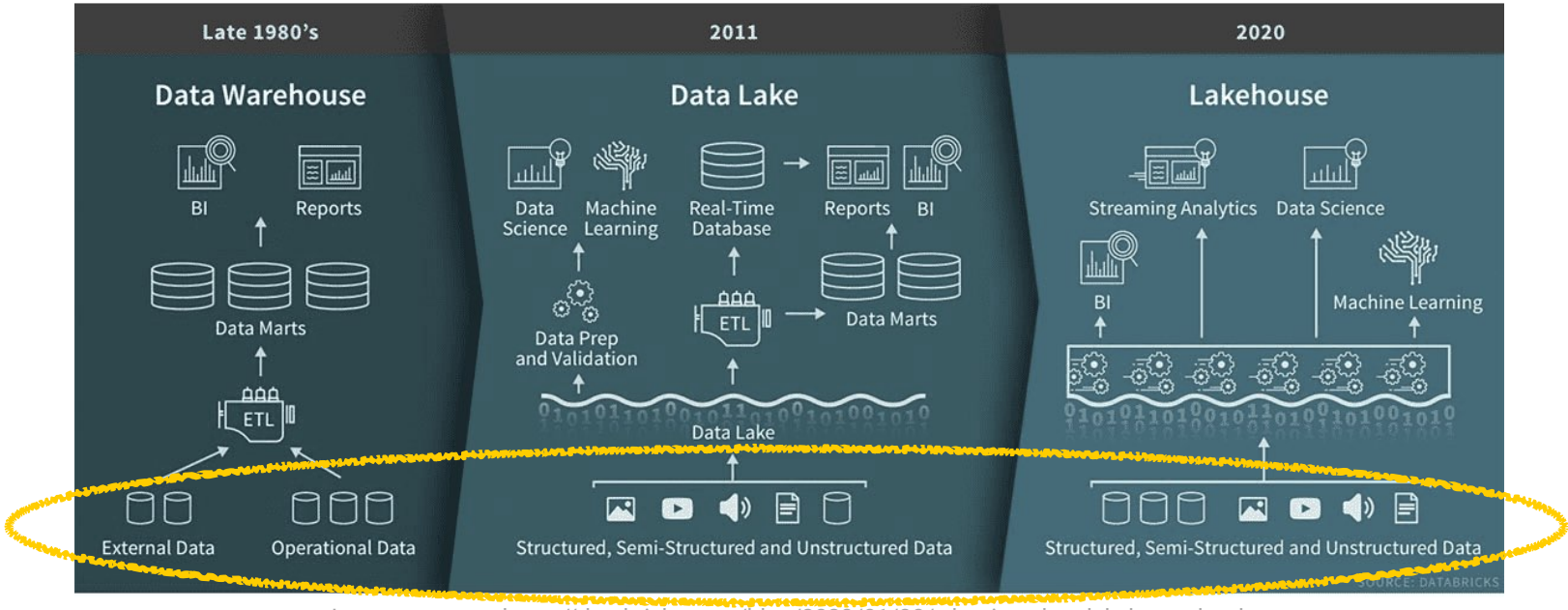


Image courtesy: https://databricks.com/blog/2020/01/30/what-is-a-data-lakehouse.html

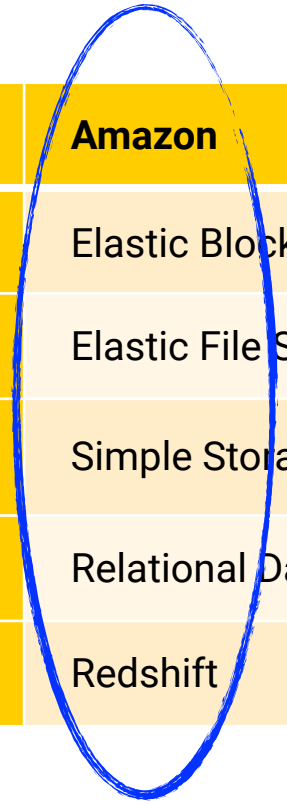
- A repository of data (*originating from disparate sources*) stored in their natural format
- Data variety in an organization: structured (employee records), semi-structured (logs, emails), unstructured (design documents), binary (images, audio, video), and others
- Motivated by the need to do analytics spanning data from multiple sources
- Key characteristics: integrated, time-variant, non-volatile, and high scale
- Warehouses that are ill-managed are referred to as *data swamps* or *data graveyards*

Cloud Storage

Mapping our storage models on to cloud offerings

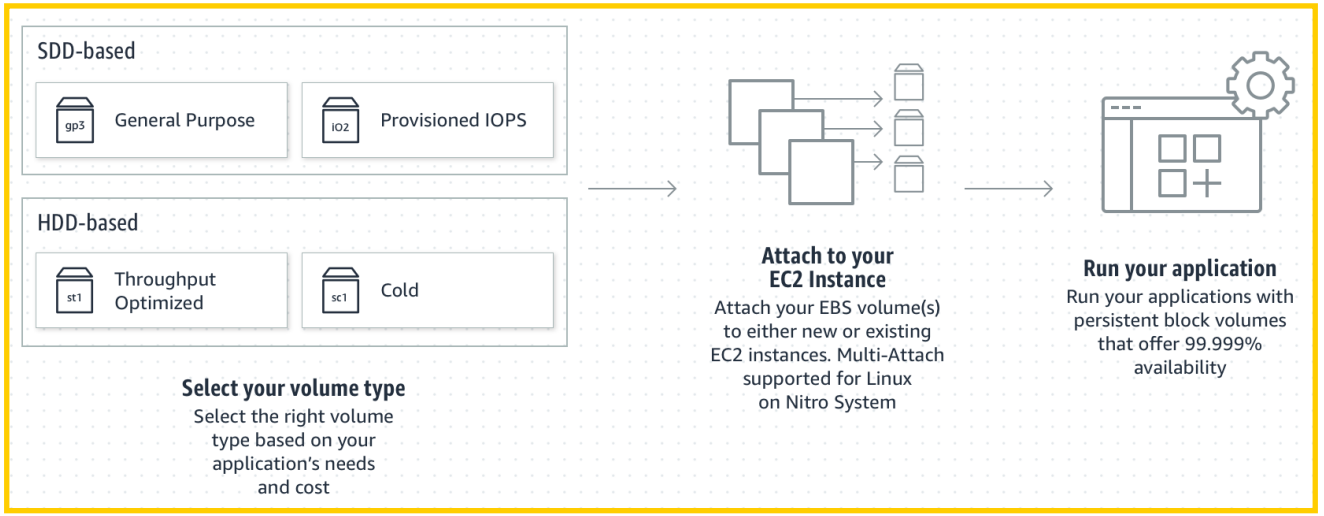
Cloud Storage Landscape

Storage model	Amazon	Google	Microsoft
1. Block storage	Elastic Block Store	Persistent disk	Azure Disk Storage
2. File system	Elastic File System	Filestore	Azure File Storage
3. Object store	Simple Storage Service	Cloud Storage	Azure Blob Storage
4. Database system	Relational Data Service	Cloud SQL	Azure SQL
5. Data warehouse	Redshift	BigQuery	Azure Data Lake



focus of our discussion

1. Elastic Block Store (EBS)



Provisioned IOPS
latency-sensitive apps

*Durability: **99.999%***
*Max IOPS: **64K***
*Max throughput: **1000 MB/s***
Price: 12.5 ¢/GB-month

General purpose
balances price-performance

Durability: 99.9%
Max IOPS: 16K
*Max throughput: **1000 MB/s***
Price: 8 ¢/GB-month

Throughput optimized
throughput-intensive workloads

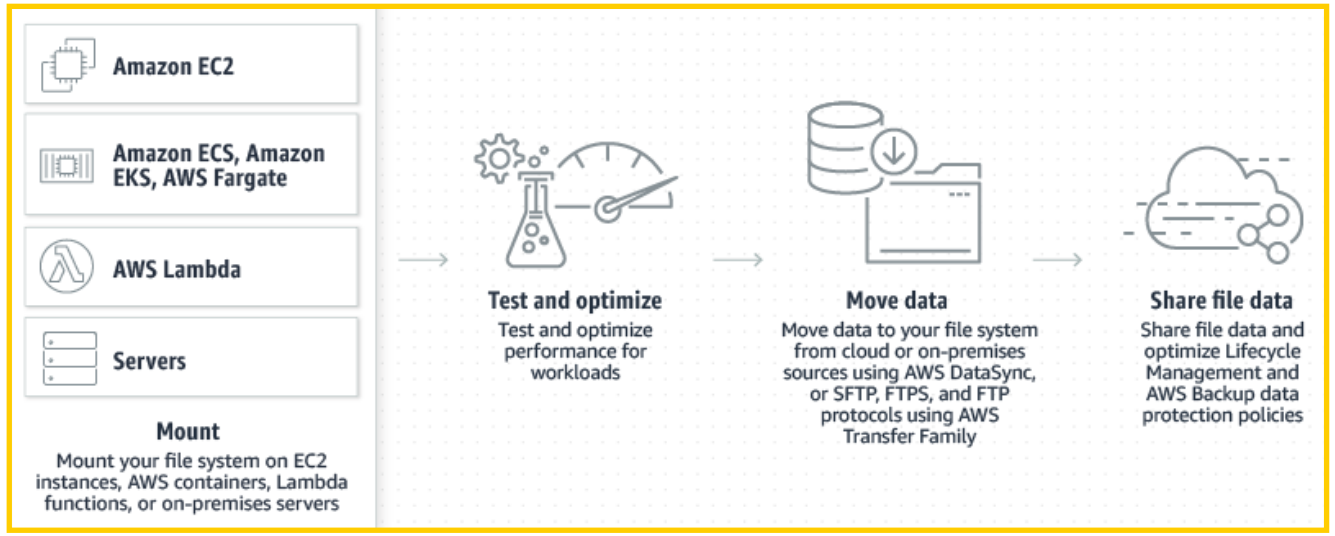
Durability: 99.9%
Max IOPS: 500
Max throughput: 500 MB/s
Price: 4.5 ¢/GB-month

Cold storage
less accessed workloads

Durability: 99.9%
Max IOPS: 250
Max throughput: 250 MB/s
*Price: **1.5 ¢/GB-month***

☞ AWS also provides ephemeral block storage: *attached locally to every EC2 instance but only lasts till its lifetime*

2. Elastic File System (EFS)

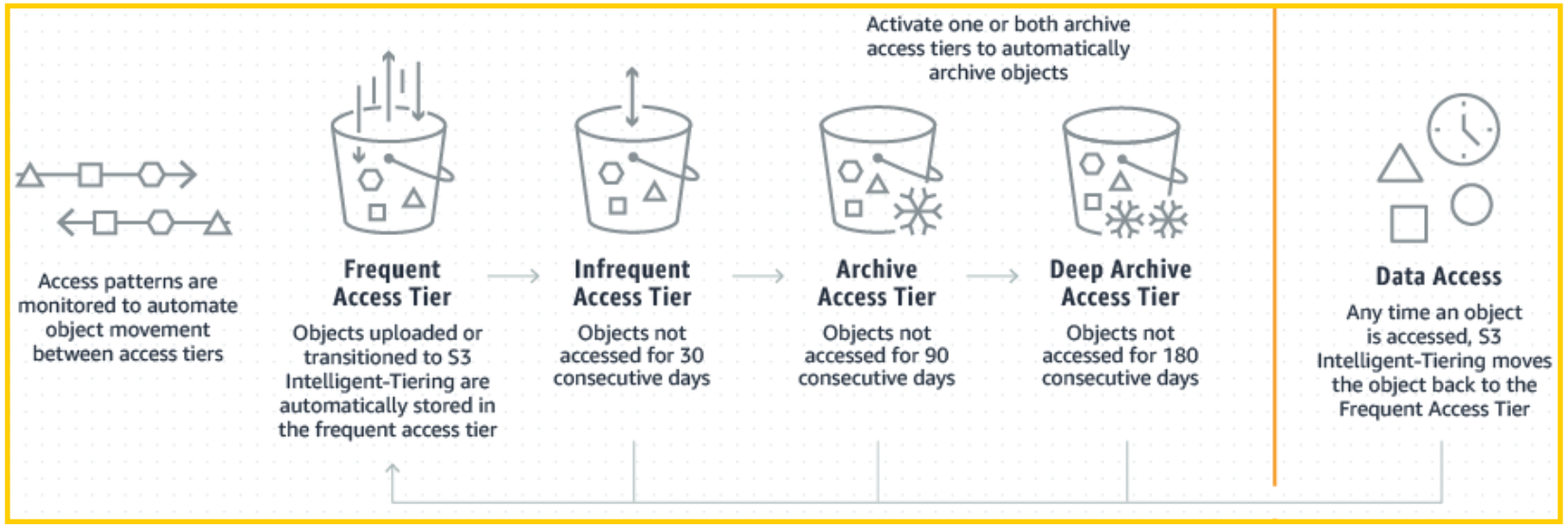


EFS internally uses a distributed data storage architecture

- **Scale**: allows data to scale to petabytes (EBS limits storage to 64TB)
- **Access**: up to thousand EC2 instances can simultaneously connect to a given EFS (EBS limits this to just one)
- **Durability**: data is stored redundantly in across multiple availability zones (EBS is in only one zone)
- **Throughput**: 1000 - 3000 Mb/s for TB-scale file systems (EBS offers a max throughput of 1000 Mb/s)
- **Cost**: 30 ¢/GB-month for *EFS Standard* and 2.5 ¢/GB-month for *EFS Infrequent Access*

☞ EFS Infrequent Access: any access to data is charged at *10 ¢/GB*

3. Simple Storage Service (S3)



- **Scale**: no limits on the number of objects or total size of the buckets (advertised support up to exabytes)
- **Access**: integrates as a storage solution with most of the AWS services
- **Durability**: provides 99.999999999% (eleven 9s) *=> with 1B objects, you may likely go 100 years without losing one*
- **Throughput**: up to 100 Gb/s with EC2 instances (EBS and EFS have a max throughputs of 1-3 Gb/s)
- **Storage cost** (¢/GB-month): 2.1 for *Standard*, 1.25 for *Infrequent Access*, *0.4 for Glacier*, *0.1 for Glacier Deep*
- **Access cost** (¢/1000 req): 0.5 for *Standard*, 1 for *Infrequent Access*, *5 for Glacier*, *10 for Glacier Deep*

☞ Access costs are lower for reads/GET; costs are based on US-East region

4. Relational DB Service (RDS)

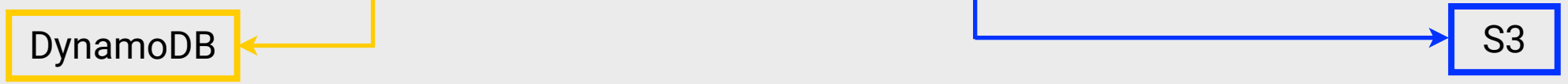
- Conventional relational databases setup and administered by AWS
- **Configurability**: users choose the initial storage/compute capacity, but AWS allows push-button scaling
- **Compute cost**: 17.8 ¢/hour (on-demand), 11.4 ¢/hour (1y-reserved)—both for PostgreSQL
- **Storage cost**: 12.5 ¢/GB-month (on SSD)
- **Data transfer cost**: free (into RDS) and 5–9 ¢/GB (out of RDS)
- **Does it really make sense?** MySQL in the cloud at Airbnb —"there is an API for that"

☞ Costs are for db.m5.large instance based in US-East region

☞ https://medium.com/airbnb-engineering/mysql-in-the-cloud-at-airbnb-336e5666bc94

Creating a data system on AWS

We have a collection of data samples stored in our personal computer. Each sample is associated with four metadata items: experiment id, item number, creation date, and a comment. We want to ***upload all data samples*** *to cloud storage and create a* ***searchable table*** *containing the metadata and URL for each data sample.*



Reference: Foster and Gannon, Cloud Computing for Science and Engineering, MIT Press, 2017

Set up an S3 bucket

```
import boto3
s3 = boto3.resource('s3')
s3.create_bucket(Bucket='datacont', CreateBucketConfiguration={
    'LocationConstraint': 'us-west-2'})
```

Create a DynamoDB table

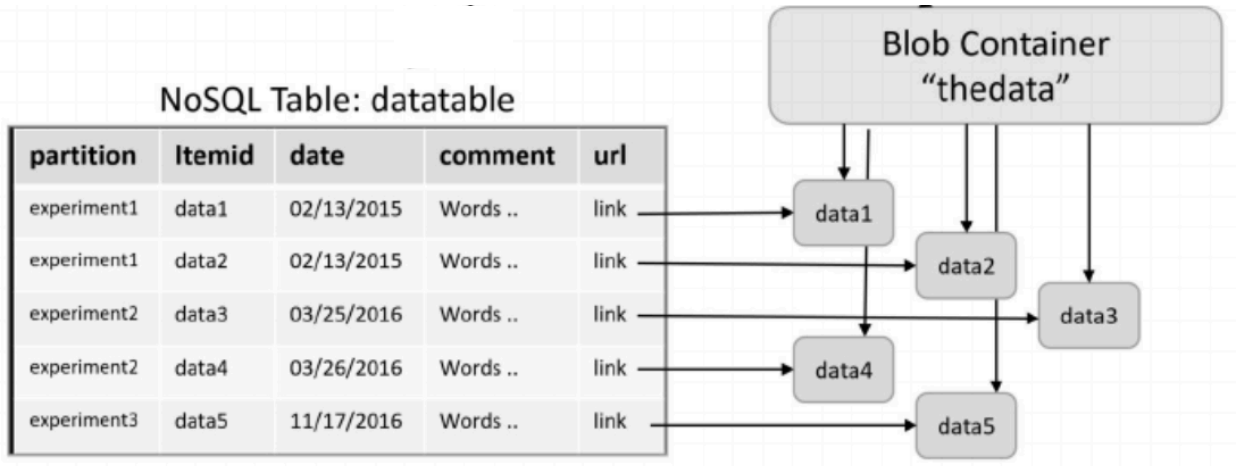
```
dyndb = boto3.resource('dynamodb', region_name='us-west-2' )

# The first time that we define a table, we use
table = dyndb.create_table(
    TableName='DataTable',
    KeySchema=[
        { 'AttributeName': 'PartitionKey', 'KeyType': 'HASH'},
        { 'AttributeName': 'RowKey', 'KeyType': 'RANGE' }
    ],
    AttributeDefinitions=[
        { 'AttributeName': 'PartitionKey', 'AttributeType': 'S' },
        { 'AttributeName': 'RowKey',       'AttributeType': 'S' }
    ]
)

# Wait for the table to be created
table.meta.client.get_waiter('table_exists')
                            .wait(TableName='DataTable')
```

Metadata CSV <experiment id, item id, date, filename, comment>

Populate data

```
import csv
urlbase = "https://s3-us-west-2.amazonaws.com/datacont/"
with open('\path-to-your-data\experiments.csv', 'rb') as csvfile:
    csvf = csv.reader(csvfile, delimiter=',', quotechar='|')
    for item in csvf:
        body = open('path-to-your-data\datafiles\\'+item[3], 'rb')
        s3.Object('datacont', item[3]).put(Body=body)
        md = s3.Object('datacont', item[3]).Acl()
            .put(ACL='public-read')
        url=urlbase +item[3]
        metadata_item={'PartitionKey': item[0], 'RowKey': item[1],
            'description' : item[4], 'date' : item[2], 'url':url}
        table.put_item(Item=metadata_item)
```

Data in the cloud



Spot Quiz (ICON)