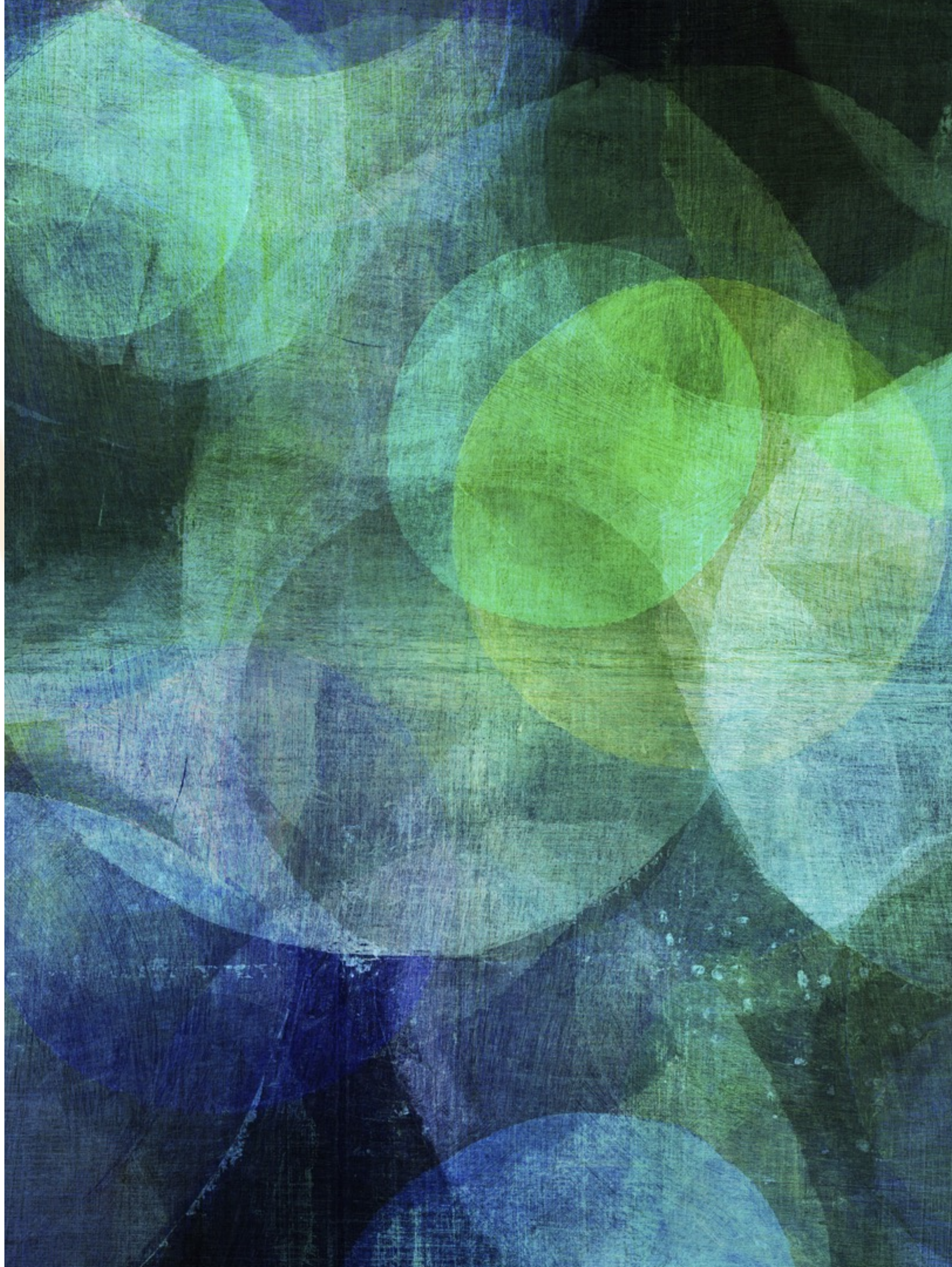
AWS DATA PROCESSING INFRASTRUCTURE 4A

Nan Dun
nan.dun@acm.org



COPYRIGHT POLICY 版权声明

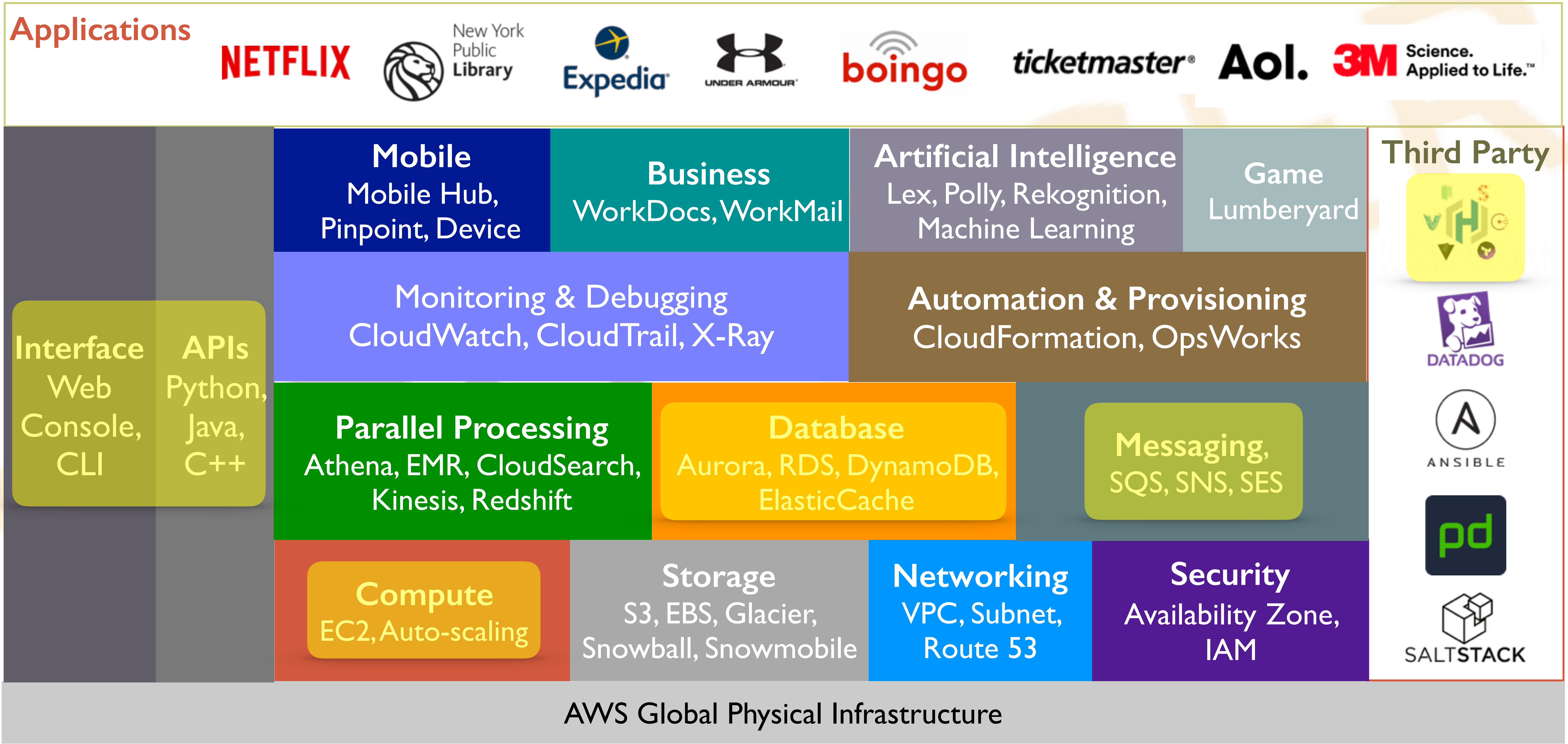
DISCLAIMER

I. *"All data, information, and opinions expressed in this presentation is for informational purposes only. I do not guarantee the accuracy or reliability of the information provided herein.This is a personal presentation.The opinions expressed here represent my own and not those of my employer."*

II. *"The copyright of photos, icons, charts, trademarks presented here belong to their authors."*

III. *"I could be wrong."*

TODAY'S TOPIC



Applications

NETFLIX

New York Public Library

Expedia

UNDER ARMOUR

boingo

ticketmaster

Aol.

3M Science. Applied to Life.™

Interface
Web Console, CLI

APIs
Python, Java, C++

Mobile
Mobile Hub, Pinpoint, Device

Business
WorkDocs, WorkMail

Artificial Intelligence
Lex, Polly, Rekognition, Machine Learning

Game
Lumberyard

Monitoring & Debugging
CloudWatch, CloudTrail, X-Ray

Automation & Provisioning
CloudFormation, OpsWorks

Parallel Processing
Athena, EMR, CloudSearch, Kinesis, Redshift

Database
Aurora, RDS, DynamoDB, ElasticCache

Messaging,
SQS, SNS, SES

Compute
EC2, Auto-scaling

Storage
S3, EBS, Glacier, Snowball, Snowmobile

Networking
VPC, Subnet, Route 53

Security
Availability Zone, IAM

Third Party

DATADOG

ANSIBLE

pd

SALTSTACK

AWS Global Physical Infrastructure

OUTLINE

- Simple Queue Service (SQS)
- DynamoDB
- Docker and Container
- Elastic Container Service (ECS)
- NYC Taxi Trip Explorer Project

SQS

SQS

- A reliable, and highly-scalable queue (message send/recv) service between applications
- For what purpose
  - Decoupling (one most import design principle for large-scale distributed systems)
  - Concurrency
  - Batch processing
  - Buffering
  - Coordinating
  - …
- Core concepts: Queue, message, send, receive, delete

QUEUE OPERATIONS

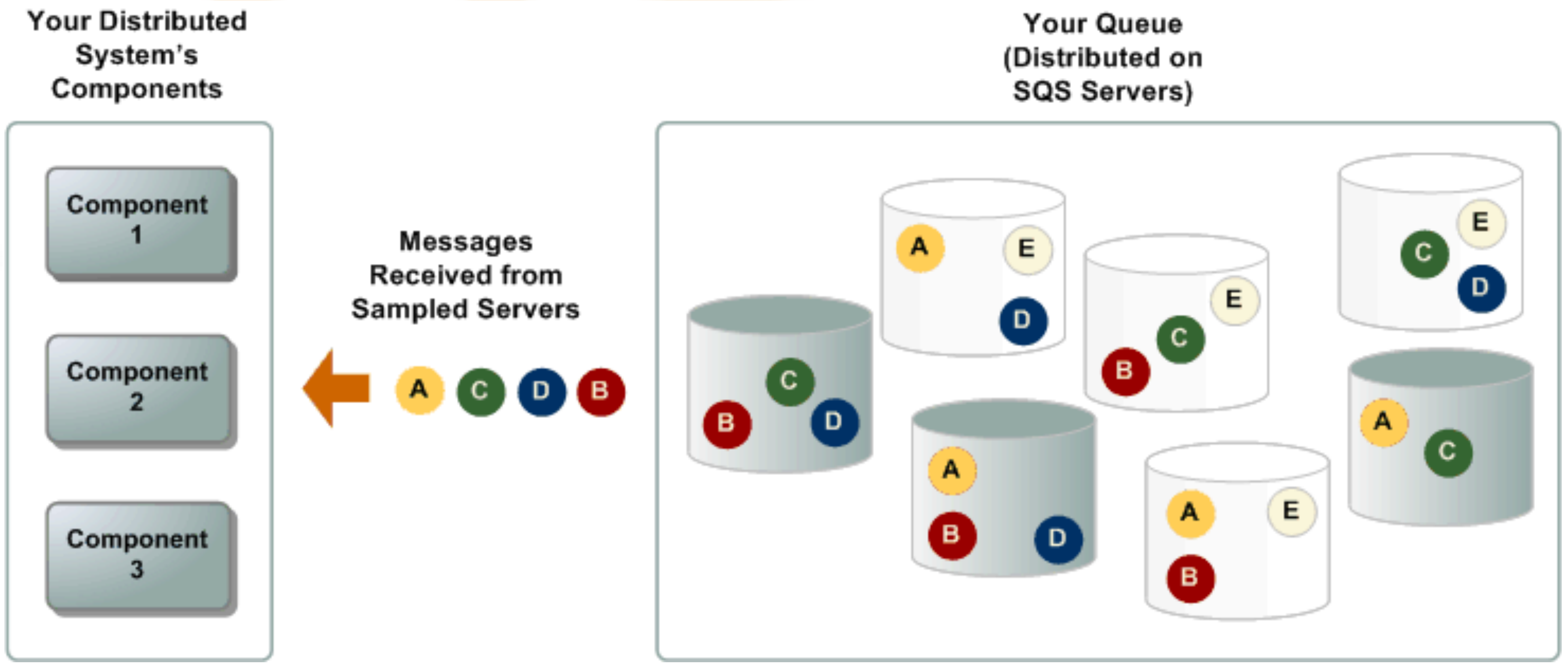
- Create a queue
- List all queues
- Add permission to a queue
  - IAM policies
- Purging a queue
- Delete a queue
- Subscribing a queue to a SNS topic
  - Store SNS notification to you queue

MESSAGE OPERATIONS

- Send
  - One message up to 256 KB, encoded as a string.
  - Messages can be sent in bulks of up to 10 (but the total size is capped at 256 KB).
- Receive
  - Up to 10 messages can be received in bulk, if available in the queue.
- Long polling
  - The request will wait up to 20 seconds for messages, if none are available initially
- Delete

STANDARD QUEUES

- Message order
    - best effort
- At-least-Once delivery
    - Receive or delete may fail, receive same message again
- Receive message by short polling
    - Sampling and deliver
    - Request may not be fulfilled
- Unlimited transactions


Your Distributed System's Components
Component 1
Component 2
Component 3
Messages Received from Sampled Servers
A C D B
Your Queue (Distributed on SQS Servers)
A E D
B C E
C E D
B C D
A B D
A E B
A C

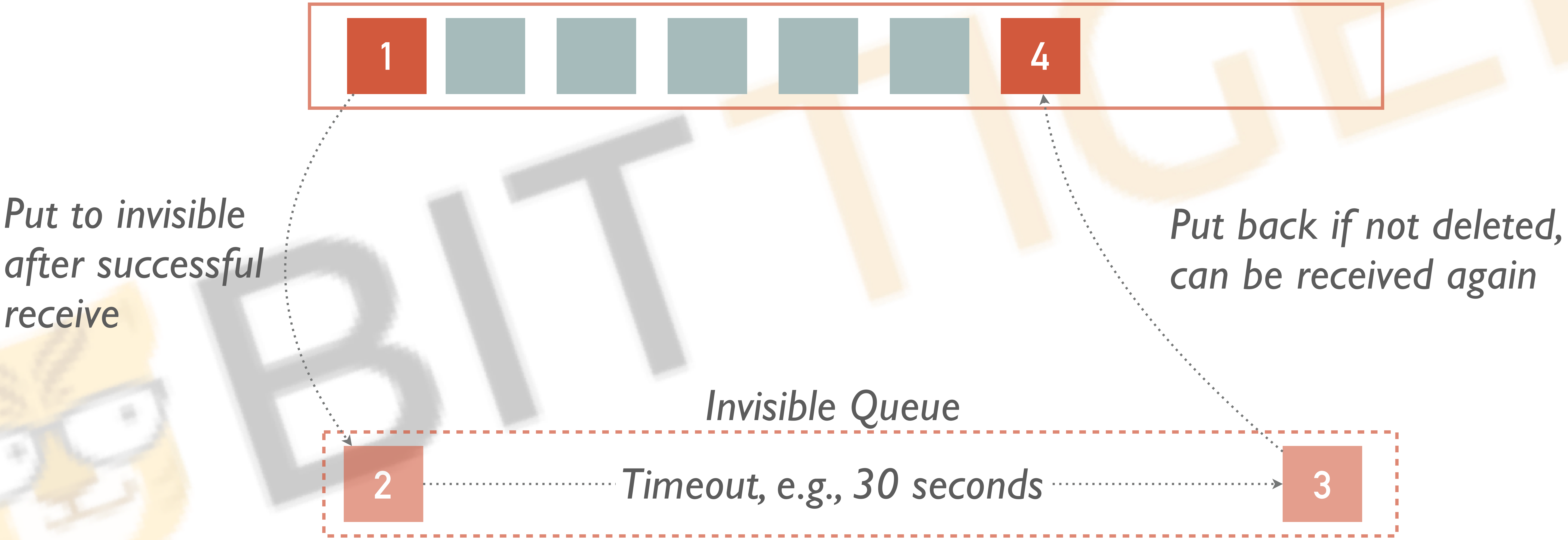
FIFO QUEUES

- Strict order of messages
  - Strict receive order as sent order
- Exactly-once processing
  - By providing a duplication ID
  - Or content-based
- Limited to 300 transactions per second
- Available only in us-west-2(oregan) and us-east-1(ohio)

QUEUE ATTRIBUTES

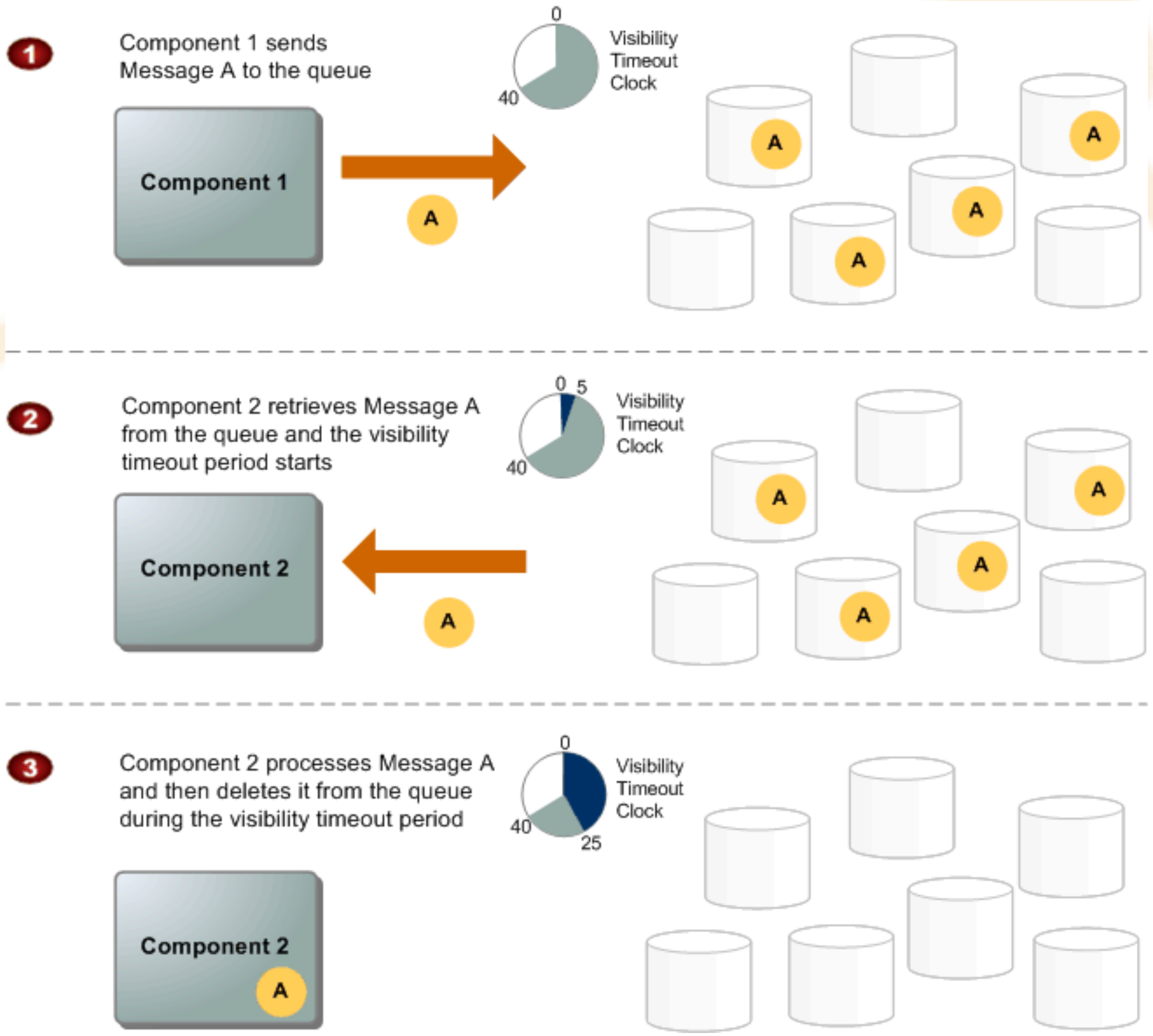
- Approximate number of messages
- Approximate number of message delayed
- Approximate number of message not visible

VISIBILITY TIMEOUT


1
4
Put to invisible after successful receive
Put back if not deleted, can be received again
Invisible Queue
2
Timeout, e.g., 30 seconds
3

MESSAGE LIFECYCLE

- Send and distribute
- Receive and is hidden for visibility timeout
- Delete a message or receive again after visibility timeout



1
Component 1 sends Message A to the queue
Component 1
A
0
40
Visibility Timeout Clock
A
2
Component 2 retrieves Message A from the queue and the visibility timeout period starts
Component 2
A
0
5
40
Visibility Timeout Clock
A
3
Component 2 processes Message A and then deletes it from the queue during the visibility timeout period
Component 2
A
0
40
25
Visibility Timeout Clock

DEAD LETTER QUEUES

- Receive messages for
  - Message that is sent to a queue that does not exist
  - Target queue is full
  - Message length limit exceeded
  - Any other failures

MESSAGE ATTRIBUTES

- User-defined
  - Name
  - Type
  - Value
- Up to 10 attributes
- Used to help process the message

LONG POLLING

- Polling for longer than 20 seconds
- To
  - Reduce the number of empty responses
  - Eliminate false empty message by querying all servers
  - Returns as soon as message become available

MESSAGE TIMER

- Message’s internal visibility timeout, not queue default
- Use to
  - Delay specific messages
  - Implement task priority

DELAY QUEUES AND MESSAGE TIMERS

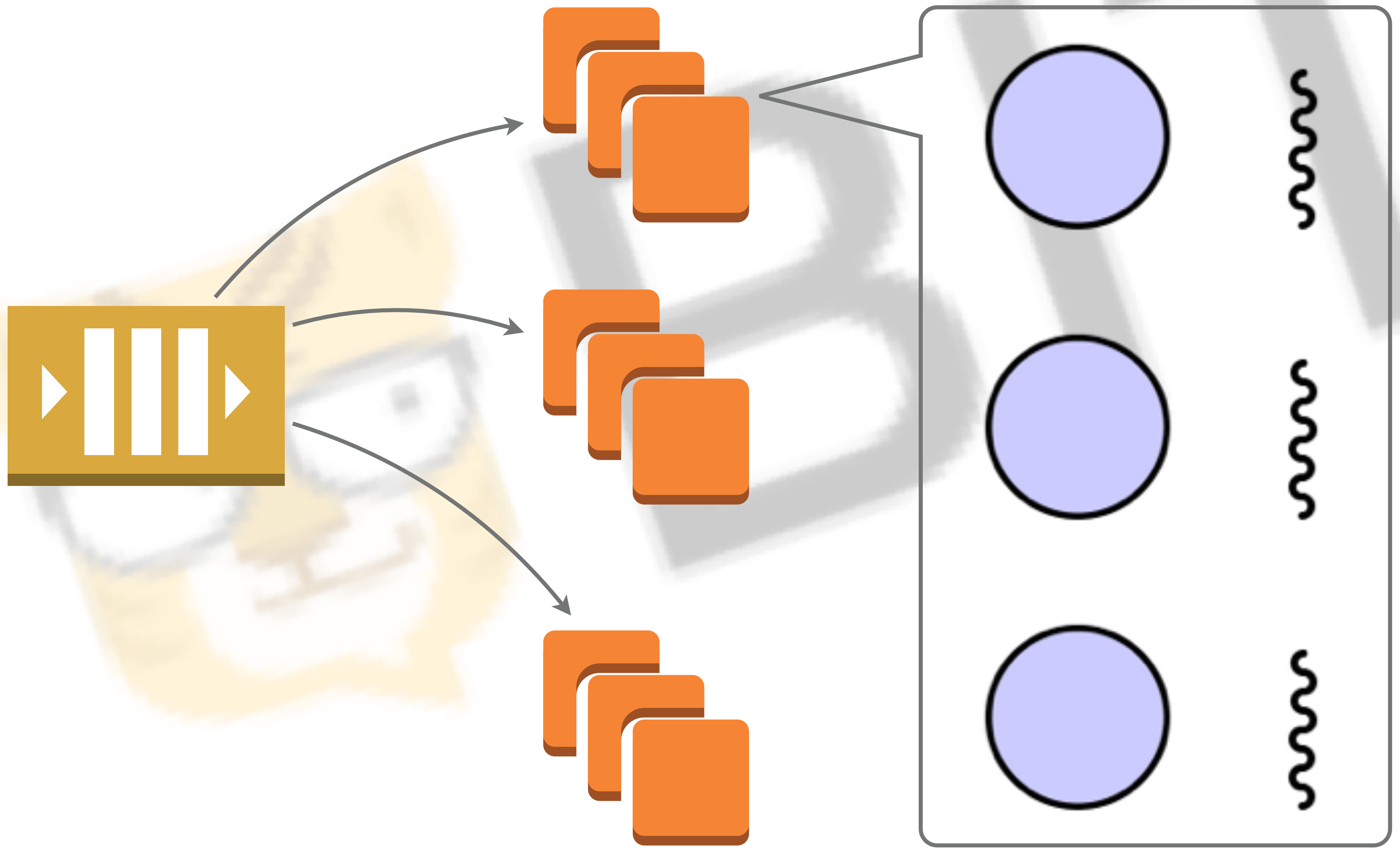
- Set a initial invisibility timeout
  - For default queue
  - or, message
- Message should wait for timeout to be visible
- 120,000 inflight message to reach OverLimit error
- For
  - Buffering

LARGE MESSAGES?

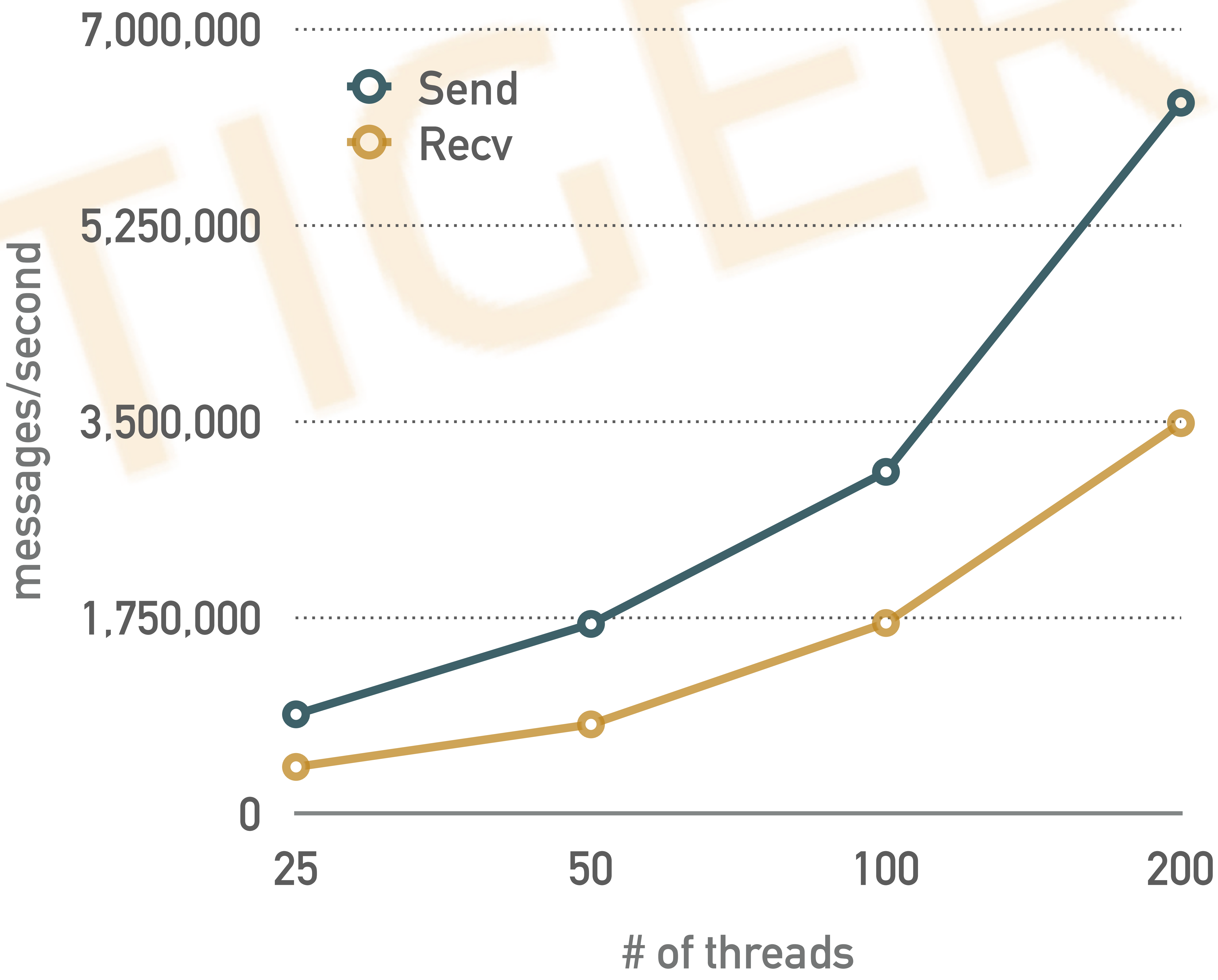
- 256 KB < message size < 2GB
- Data stored in S3, use message as a pointer/reference
- Operations
  - Select to store all on S3 or only when size > 256KB
  - Send a reference of message on S3
  - Receive a message on S3
  - Delete
- Java only with extended client library
  - https://github.com/awslabs/amazon-sqs-java-extended-client-lib

CAPACITY

- EC2 m2.xlarge
  - 100K messages/minute
  - 1M messages/minute by 10 instances
- Depends on your applications



Source: http://www.warski.org/blog/2014/06/benchmarking-sqs/



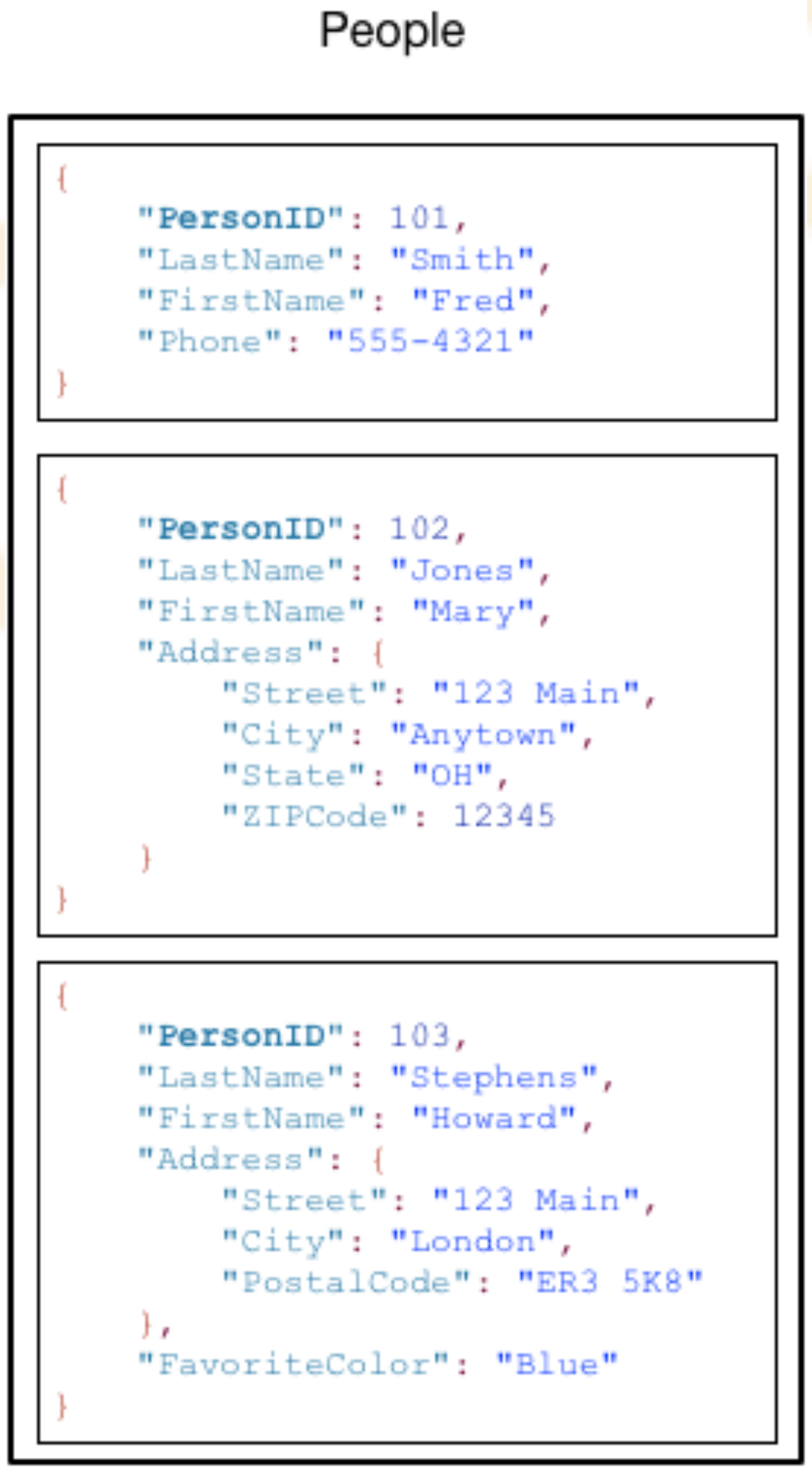
DYNAMODB

NOSQL VS. SQL

- NoSQL examples
  - Column
  - Key-value store
  - Document
  - Graph
  - …

CONCEPTS

- Tables
- Items
- Attributes
- Primary key
  - Partition key
  - Partition key + sorted key
- Secondary indexes
  - Global secondary index
    - Partition key + sorted key different from primary
  - Local secondary index
    - Same partition key as default



People

```
{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}
```

```
{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
        "Street": "123 Main",
        "City": "Anytown",
        "State": "OH",
        "ZIPCode": 12345
    }
}
```

```
{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
        "Street": "123 Main",
        "City": "London",
        "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```

DB OPERATIONS

- Table
  - Create
  - Describe
  - List
  - Update
  - Delete

- Data
  - Create: String|number|binary|bool|null
  - Read
    - Get, BatchGet
    - Query
    - Scan
  - Update
  - Delete
    - Delete
    - BatchWrite: del 25 once

READ CONSISTENCY

- Tables in different regions are independent
- Eventually consistency reads (default)
  - When you read data from a DynamoDB table, the response might not reflect the results of a recently completed write operation. The response might include some stale data. If you repeat your read request after a short time, the response should return the latest data.
- Strongly consistent reads
  - When you request a strongly consistent read, DynamoDB returns a response with the most up-to-date data, reflecting the updates from all prior write operations that were successful. A strongly consistent read might not be available in the case of a network delay or outage.

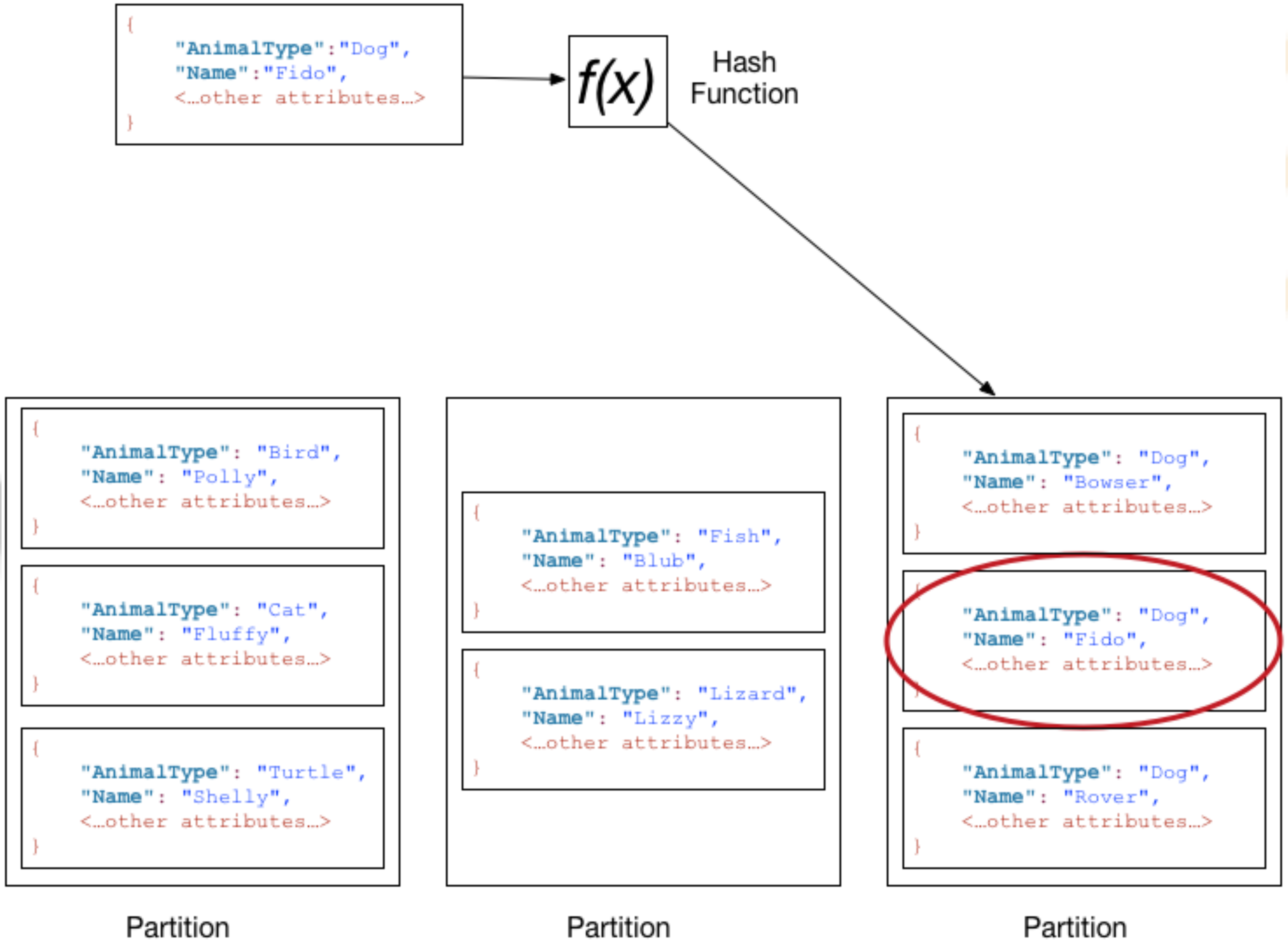
PROVISIONED THROUGHPUT

- Read Unit
  - 1 read unit = one strong consistency read per second or two eventually consistency reads per second, up to 4KB

  -
    ```
    3 KB / 4 KB = 0.75 --> 1
    6 KB / 4 KB = 1.5 --> 2
    1 read capacity unit per item × 80 reads per second = 80
    2 read capacity units per item × 100 reads per second = 200
    ```
- Write unit
  - 1 write unit = one write per second up to 1KB

  -
    ```
    512 bytes / 1 KB = 0.5 --> 1
    1.5 KB / 1 KB = 1.5 --> 2
    1 write capacity unit per item × 100 writes per second = 100
    2 write capacity units per item × 10 writes per second = 20
    ```

PARTITIONS



```
{
    "AnimalType":"Dog",
    "Name":"Fido",
    <…other attributes…>
}
```

f(x) Hash Function

```
{
    "AnimalType": "Bird",
    "Name": "Polly",
    <…other attributes…>
}
```

```
{
    "AnimalType": "Cat",
    "Name": "Fluffy",
    <…other attributes…>
}
```

```
{
    "AnimalType": "Turtle",
    "Name": "Shelly",
    <…other attributes…>
}
```

Partition

```
{
    "AnimalType": "Fish",
    "Name": "Blub",
    <…other attributes…>
}
```

```
{
    "AnimalType": "Lizard",
    "Name": "Lizzy",
    <…other attributes…>
}
```

Partition

```
{
    "AnimalType": "Dog",
    "Name": "Bowser",
    <…other attributes…>
}
```

```
{
    "AnimalType": "Dog",
    "Name": "Fido",
    <…other attributes…>
}
```

```
{
    "AnimalType": "Dog",
    "Name": "Rover",
    <…other attributes…>
}
```

Partition

EXAMPLES: CREATE A TABLE

```
CREATE TABLE Music (
    Artist VARCHAR(20) NOT NULL,
    SongTitle VARCHAR(30) NOT NULL,
    AlbumTitle VARCHAR(25),
    Year INT,
    Price FLOAT,
    Genre VARCHAR(10),
    Tags TEXT,
    PRIMARY KEY(Artist, SongTitle)
);
```

```
{
    TableName : "Music",
    KeySchema: [
        {
            AttributeName: "Artist",
            KeyType: "HASH", //Partition key
        },
        {
            AttributeName: "SongTitle",
            KeyType: "RANGE" //Sort key
        }
    ],
    AttributeDefinitions: [
        {
            AttributeName: "Artist",
            AttributeType: "S"
        },
        {
            AttributeName: "SongTitle",
            AttributeType: "S"
        }
    ],
    ProvisionedThroughput: {
        ReadCapacityUnits: 1,
        WriteCapacityUnits: 1
    }
}
```

EXAMPLE: READ AN ITEM

```
SELECT *  FROM Music  WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    }
}
```

EXAMPLE: QUERY A TABLE

```
SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today';
  {
      TableName: "Music",
      KeyConditionExpression: "Artist = :a and SongTitle = :t",
      ExpressionAttributeValues: {
          ":a": "No One You Know",
          ":t": "Call Me Today"
      }
  }
```

EXAMPLE: SCAN A TABLE

```
SELECT Artist, Title FROM Music;
```

```
{
    TableName:   "Music",
    ProjectionExpression: "Artist, Title"
}
```

EXAMPLE: SCAN A TABLE

```
UPDATE Music

SET RecordLabel = 'Global Records'

WHERE Artist = 'No One You Know' AND SongTitle = 'Call Me Today';
```

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ExpressionAttributeValues: {
        ":label": "Global Records"
    }
}
```

EXAMPLE: SCAN A TABLE

```
DELETE FROM Music

WHERE Artist = 'The Acme Band' AND SongTitle = 'Look Out, World';
```

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    }
}
```

PROGRAMMING INTERFACES: LOW-LEVEL

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization:AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>, Signature=<Signature>
X-Amz-Date: <Date>
X-Amz-Target: DynamoDB_20120810.GetItem

{
    "TableName": "Pets",
    "Key": {
        "AnimalType": {"S": "Dog"},
        "Name": {"S": "Fido"}
    }
}
```

PROGRAMMING INTERFACES: DOCUMENT AND OBJECT

- Java and .NET Only

```
public class MusicDocumentDemo {

    public static void main(String[] args) {

        AmazonDynamoDBClient client = \
           new AmazonDynamoDBClient();
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");
        GetItemOutcome outcome = table.getItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Call Me Today");

        int year = outcome.getItem().getInt("Year");

    }
}
```

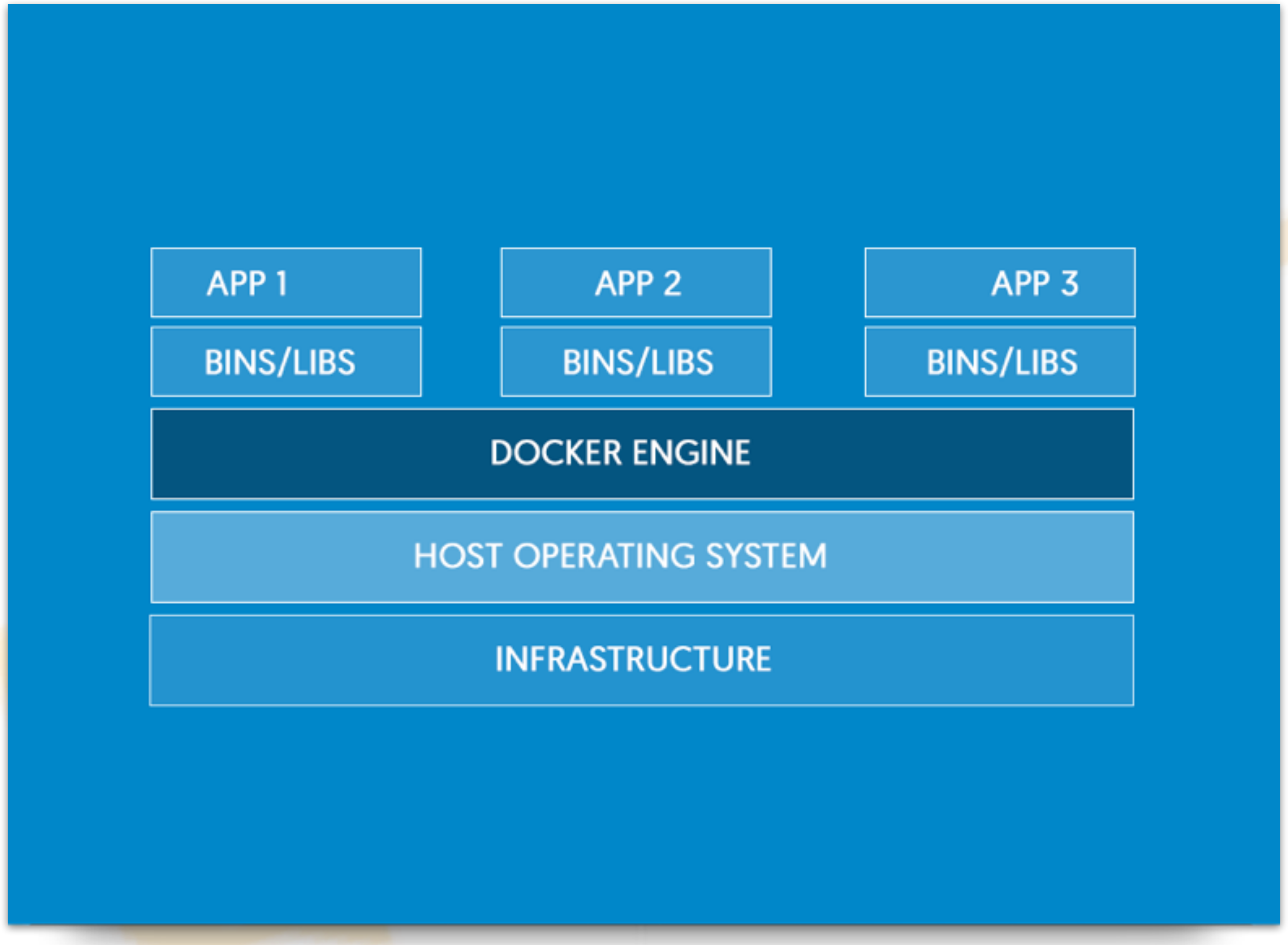
```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
    private String ISBN;
    private Set<String> bookAuthors;
    private String someProp;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id;}
    public void setId(Integer id) {this.id = id;}

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() {return title; }
    public void setTitle(String title) { this.title = title; }
}
```

> Whenever you find yourself on the side of majority, it is time to pause and reflect.

-Mark Twain

DOCKER

DOCKER VS. VM



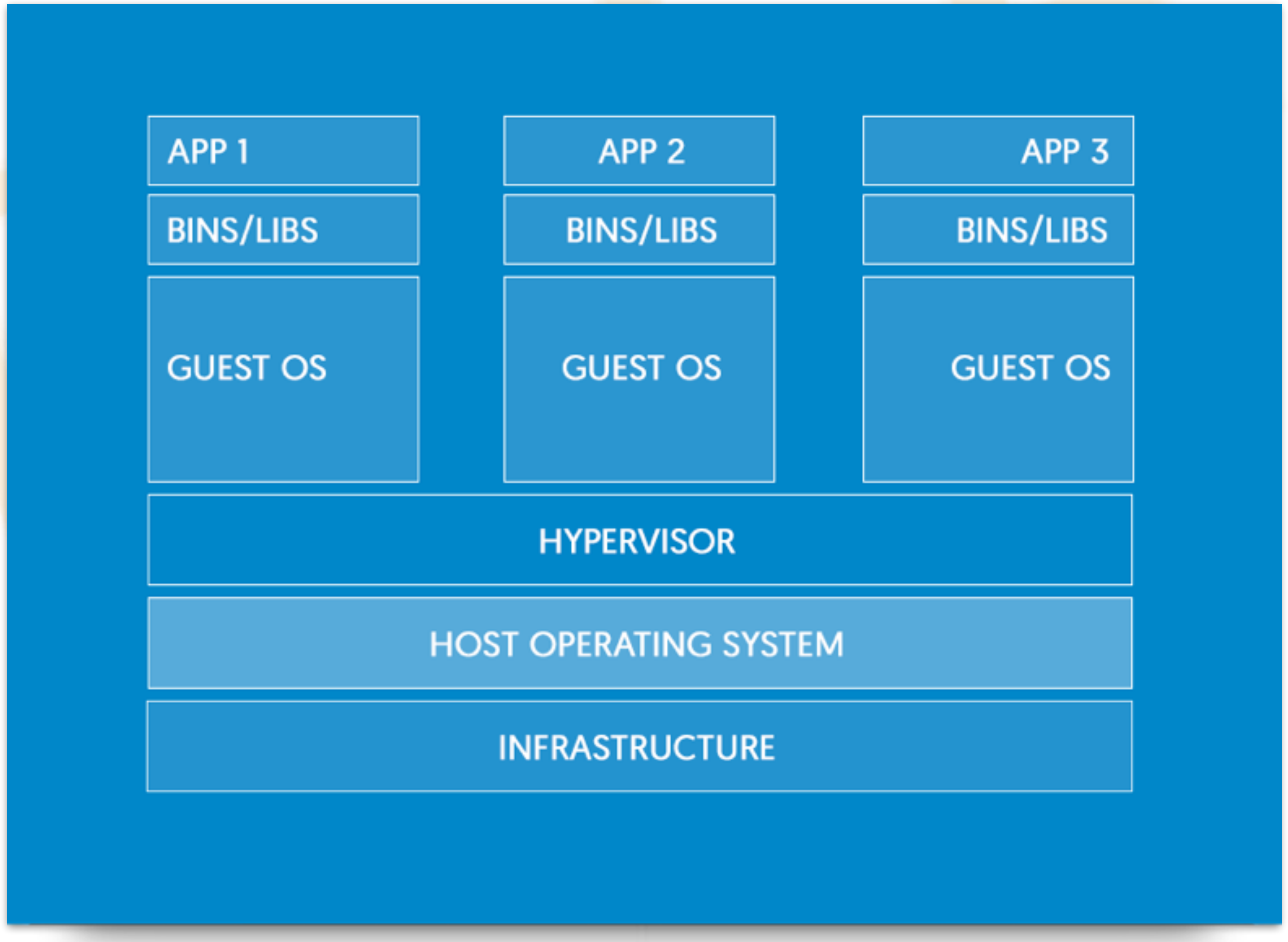
APP 1 APP 2 APP 3

BINS/LIBS BINS/LIBS BINS/LIBS

DOCKER ENGINE

HOST OPERATING SYSTEM

INFRASTRUCTURE



APP 1 APP 2 APP 3

BINS/LIBS BINS/LIBS BINS/LIBS

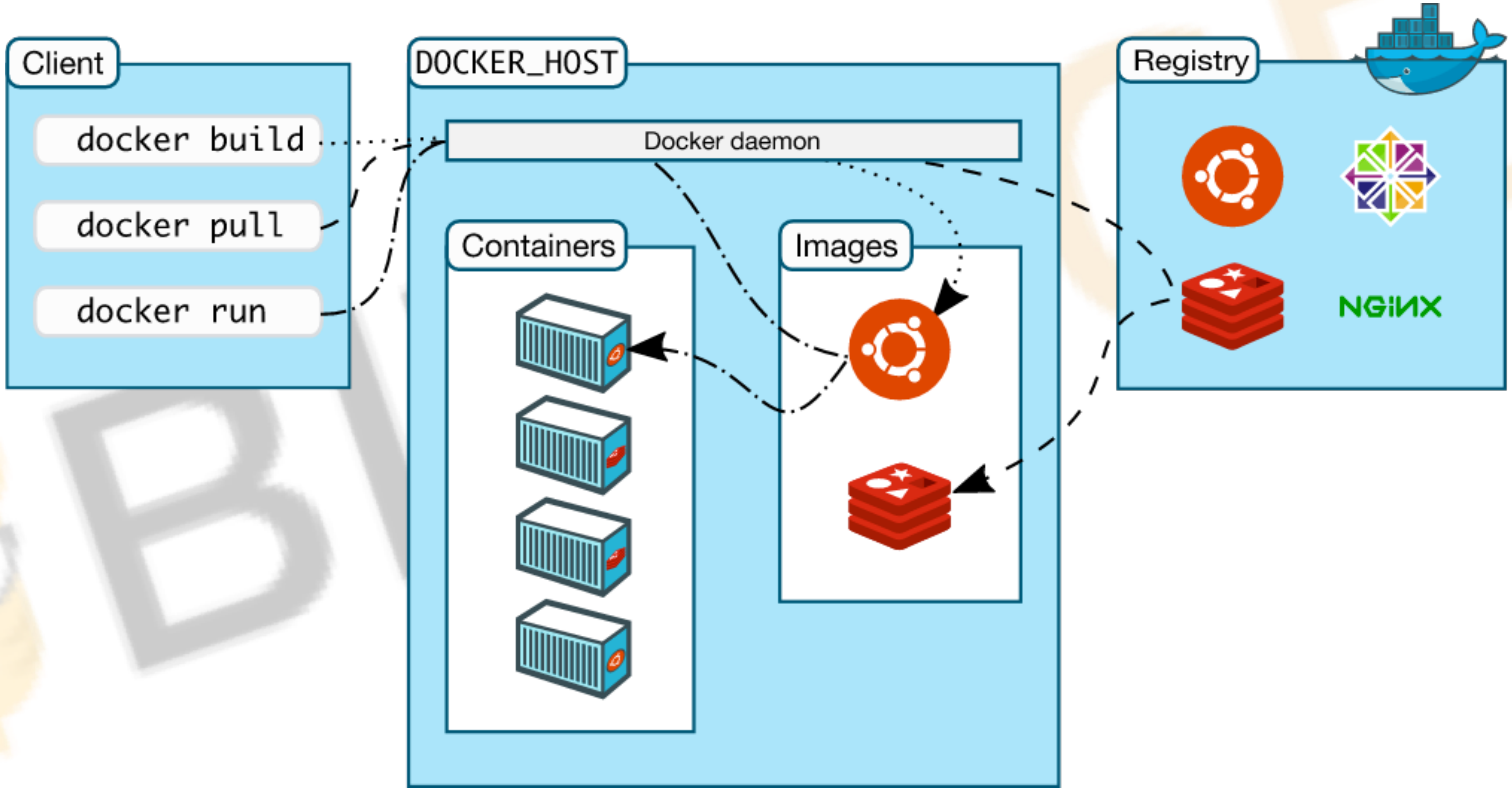
GUEST OS GUEST OS GUEST OS

HYPERVISOR

HOST OPERATING SYSTEM

INFRASTRUCTURE

ARCHITECTURE



WHY MICROSERVICE? I MEAN, FUNDAMENTALLY

- Think about functions and process pool in programming language
  - Easy to write, debug, reuse a function
  - Functions talks each other by parameters and return values
- Advantages
  - Isolation
  - Scalability, of course for deployment, not performance
- Disadvantages
  - Operational overhead
  - Performance overhead
  - I/O, error handling
  - Compatibility

ENABLE TECHNOLOGY

- Process separation, not hardware separation
- Namespace feature provided by Linux kernel
    - pid: Process isolation
    - net: network interface
    - ipc: interprocess communication
    - mnt: filesystem mount points
    - uts: kernel and version identifiers
- Control groups
- Union File Systems
- Container Format

KEY CONCEPTS

- Docker image
  - Like AMI + Packer
- Docker engine
  - Like Ansible
- Docker hub and registry
  - Like github

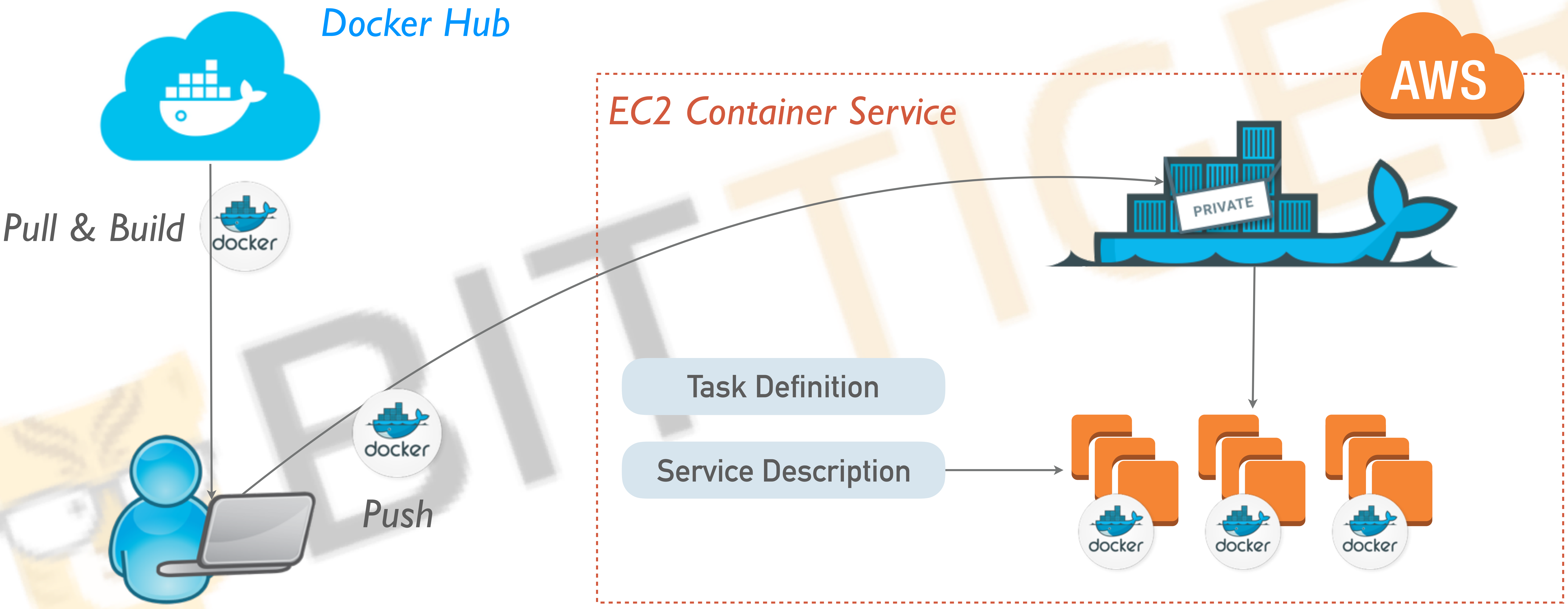
DOCKERFILE

- FROM
- RUN
- CMD
- ENTRYPOINT
- ADD/COPY
- EXPOSE
- .dockerignore

```
FROM ubuntu

# Install vnc, xvfb in order to create a 'fake' display and firefox
RUN apt-get update && apt-get install -y x11vnc xvfb firefox
RUN mkdir ~/.vnc
# Setup a password
RUN x11vnc -storepasswd 1234 ~/.vnc/passwd
# Autostart firefox (might not be the best way, but it does the trick)
RUN bash -c 'echo "firefox" >> /.bashrc'

EXPOSE 5900
CMD    ["x11vnc", "-forever", "-usepw", "-create"]
```

ECS

DOCKER ON EC2



CONTAINER INSTANCE

- Using AMI with agent enabled
  - Amazon ECS-optimized AMI (http://docs.aws.amazon.com/AmazonECS/latest/developerguide/container_agent_versions.html#ecs-optimized-ami-agent-versions)
  - No more HVM and PV
- IAM Role: AmazonEC2ContainerServiceforEC2Role

TASK DEFINITION

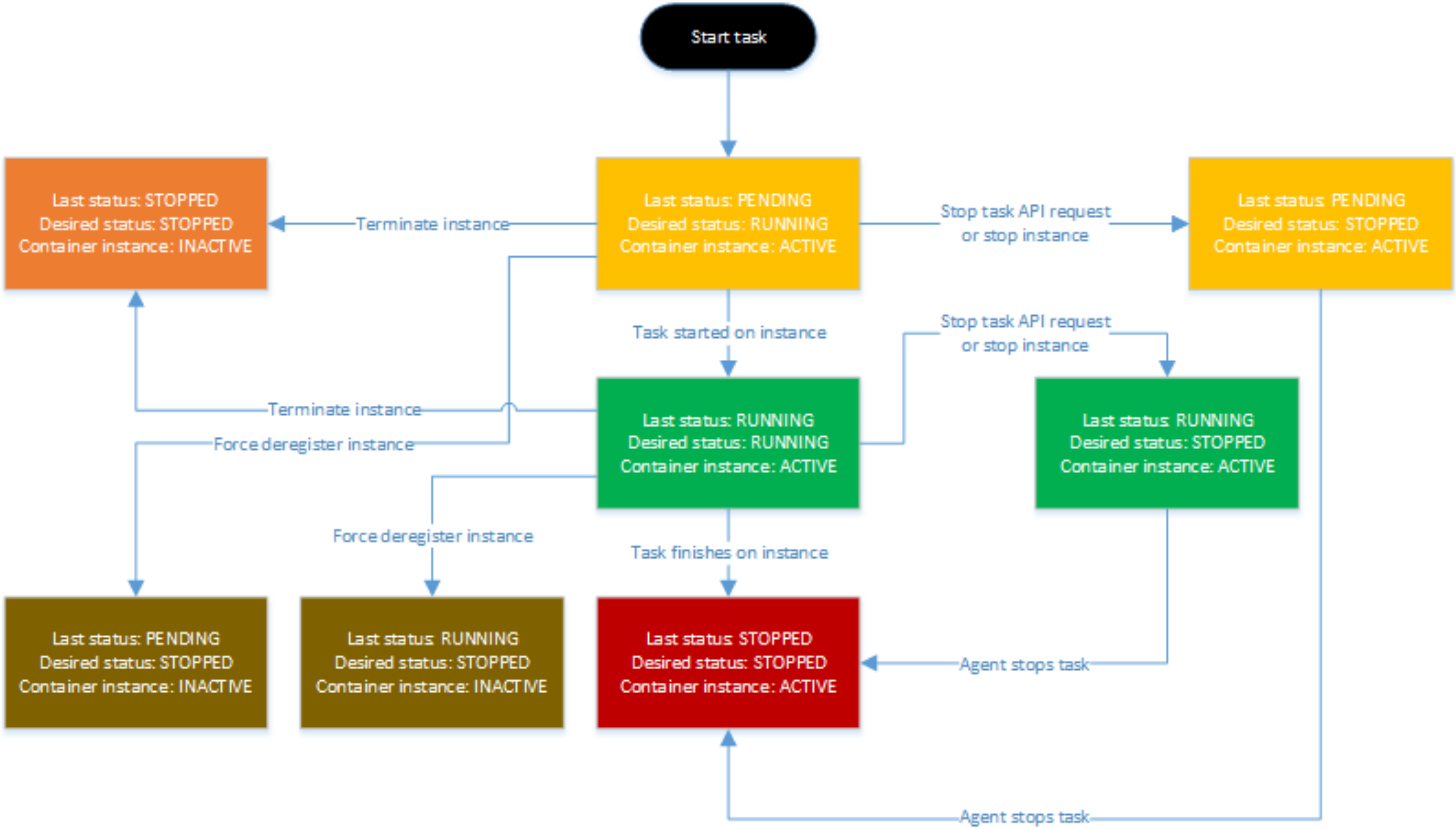
- Specification of run Docker containers

```
{
  "containerDefinitions": [
    {
      "name": "sample-app",
      "image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/aws-nodejs-sample:v1",
      "memory": "200",
      "cpu": "10",
      "essential": true
    }
  ],
  "family": "example_task_3",
  "taskRoleArn": "arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole"
}
```

TASK SCHEDULING

- Run task with scheduler
  - Specify how many tasks (workers) you need
- Task placement
  - binpack
    - Least available amount of CPU or memory of instance
  - random
  - spread
    - Based on specific attributed, such as availability zones

TASK LIFECYCLE



SERVICE

- A set of instance running given tasks
  - The number of tasks are maintained to desired level
  - Service can be behind a load balancer
  - Auto-balancing across availability zones
  - Stop service in a optimal way across availability zones
    - Stop instance where its zones as more instance running

NYC TAXI

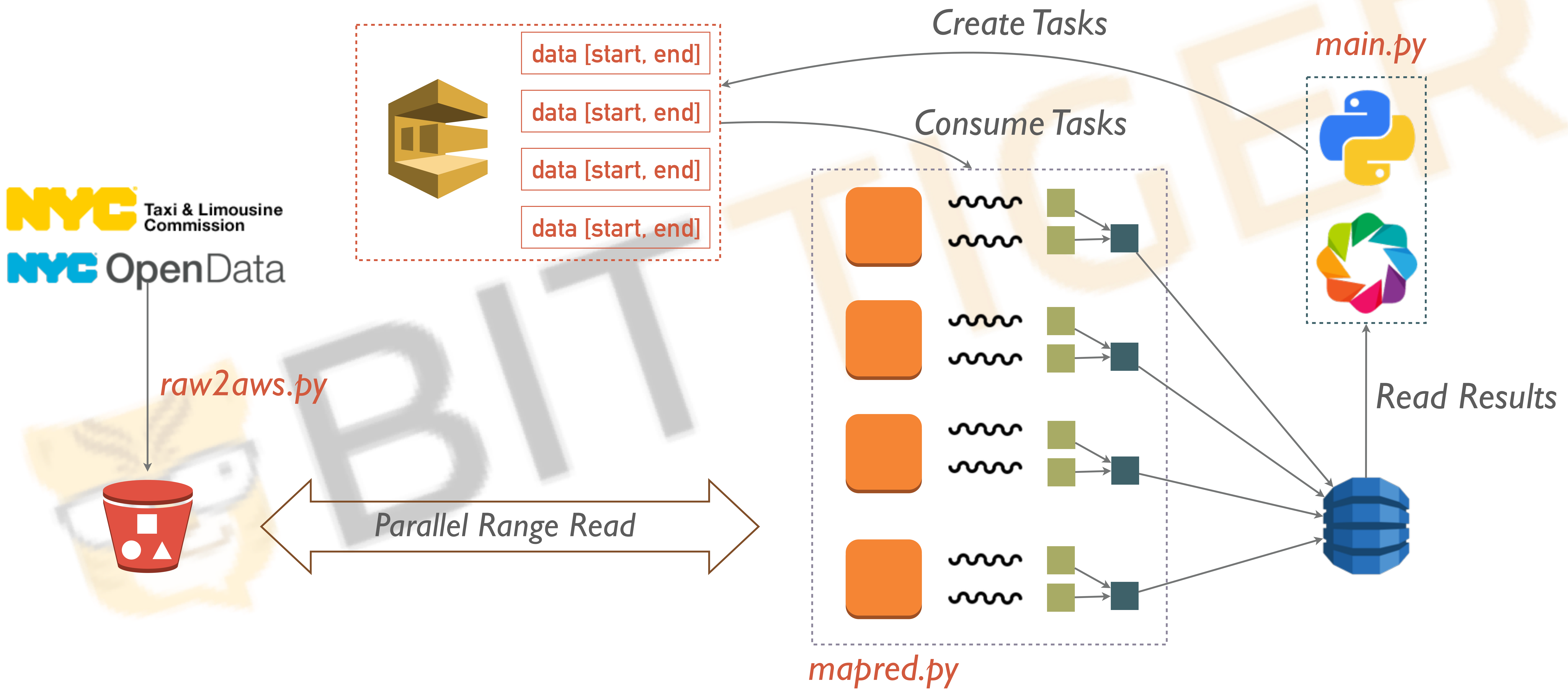
WHAT WE HAVE NOW

- Well-formate input data
  - 80 bytes per record
- A to map-reduce program to count statistics
  - Calculate each trip in which districts
- How to connect them together to process data in parallel?
  - A queue to provide concurrency and task tracking
  - A result table to aggregate statistics

MAJOR COMPUTATION

- Test is a point is in polygon
  - A point (longitude, latitude)
  - A polygon described by a series of boundary points
    - http://erich.realtimerendering.com/ptinpoly/
    - Python shapely package: Polygon.contains(Point())
- Geojson Data
  - https://github.com/dwillis/nyc-maps
- 99% computation

WORKFLOW

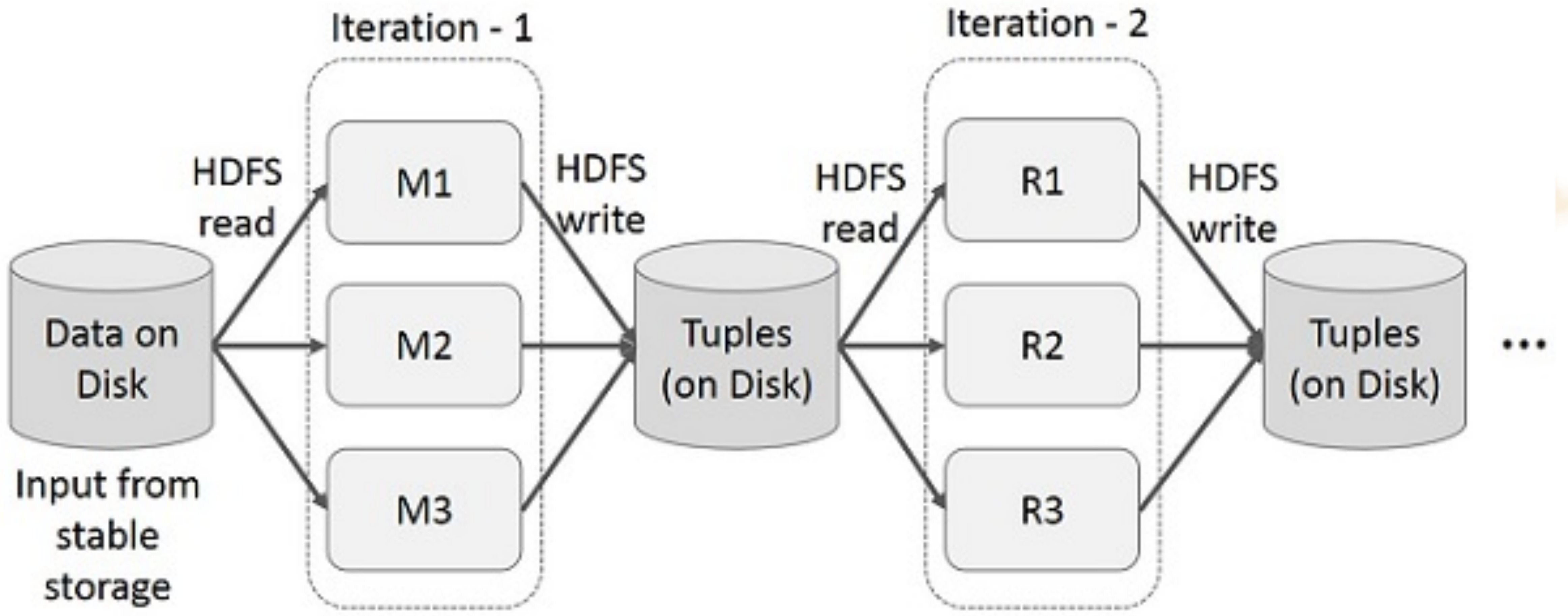

data [start, end]
data [start, end]
data [start, end]
data [start, end]
Create Tasks
main.py
Consume Tasks
Taxi & Limousine Commission
NYC OpenData
raw2aws.py
Parallel Range Read
Read Results
mapred.py

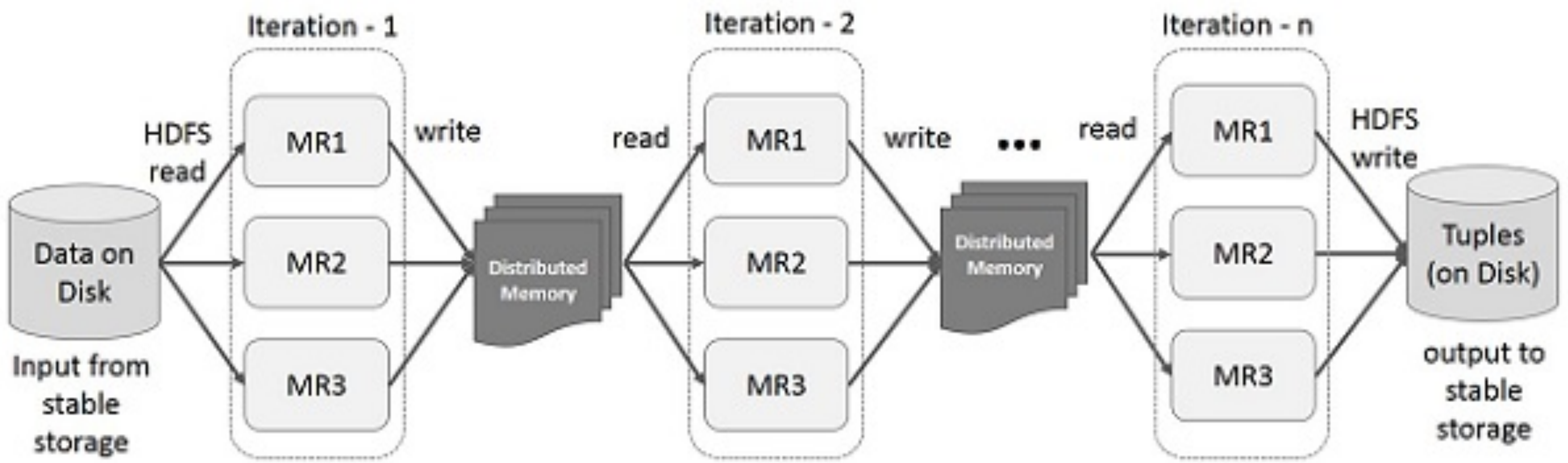
DEPLOYMENT

- Classic
  1. Build AMI
  2. Terraform for infrastructure
  3. Configure instances via user-data or Ansible
  4. Run tasks via user-data
- Container (in a infrastructure)
  1. Build Docker Image and push to registry
  2. Define tasks and service
  3. Run

WHY NOT SPARK?

- We have no sharing state/data at all!
  - Why bother using RDD?!
- We have only one step!
- Overhead to run on top of Spark
- Too complex for out tasks
- For this task, we are much faster than using Spark



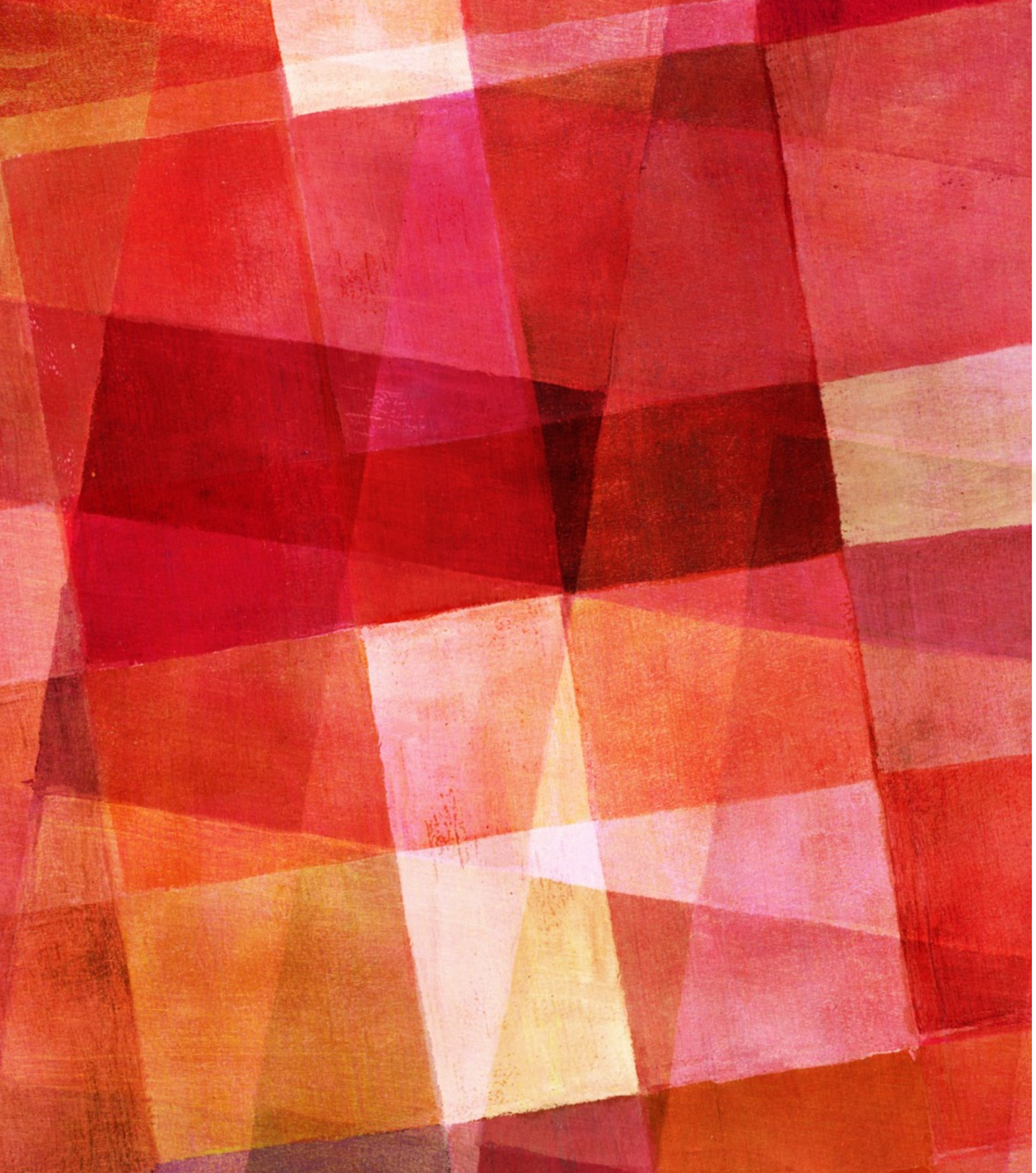


USE CONTAINER?

- All tasks are homogenous
  - Load from one single executable
- All tasks runs at full speed of CPU
- We can run as many as processes if memory allows…
- Low-level process/thread management is more efficient
- Container not idea for this case? but VM overhead
  - Overhead of packing and deploy
  - Overhead of isolation
  - Overhead of Docker daemon

BOKEH ボケ

- A Python Interactive visualization library mainly for web
- Translate to Javascript
- Examples used in project
  - https://demo.bokehplots.com/apps/movies
  - http://bokeh.pydata.org/en/latest/docs/gallery/texas.html
  - https://www.quantinsti.com/blog/python-data-visualization-using-bokeh/



QUESTIONS

- bittiger-aws@googlegroups.com

BITTIGER