2110625 - Data Science Architecture

Coordination

Asst.Prof. Natawut Nupairoj, Ph.D.
Department of Computer Engineering
Chulalongkorn University
natawut.n@chula.ac.th

CHULA ENGINEERING
Foundation toward Innovation

cluster บางตัวสามารถกำหนด member ของ cluster hard code ใน config file ได้เลย แต่เวลา scale ต้องแก้ config file ทุกเครื่อง เลยมี service ที่ทำให้ทุกคน share config แล้ว member ใหม่แต่ละที่ๆ มาขึ้นที่นี่ว่าไปดูที่นี่ได้ข้อมูลอยู่

Needs of Cluster Coordination

- Distributed systems, especially large-scale clusters, need centralized coordination services (share ข้อมูล config)
  - Sharing data among nodes in the cluster e.g. membership, node status, configuration, metadata
  - Coordinating between nodes e.g. leader election, agreement resolution (ที่ทุกคนผลสุดท้ายของข้อตกลง)
  - Resource management e.g. compute nodes, data nodes

leader ทำให้มั่นใจ agreement และ solution
บางทีทำคำสั่งใดก่อน-หลัง (sequencer) เหมือน conductor

agreement เช่น 2 clients ยิงข้อความหา node ขอ update data ตัวเดียวกัน

Total ordering !!!

ยกตัวอย่าง cassandra ที่เป็น peer-to-peer ไม่มี leader ทำให้เน้นที่ consistency ยากขึ้น (AP)
ถ้า 2 client เข้ามา update ตัวเดียวกัน แต่คนละ node cassandra มีปัญหา

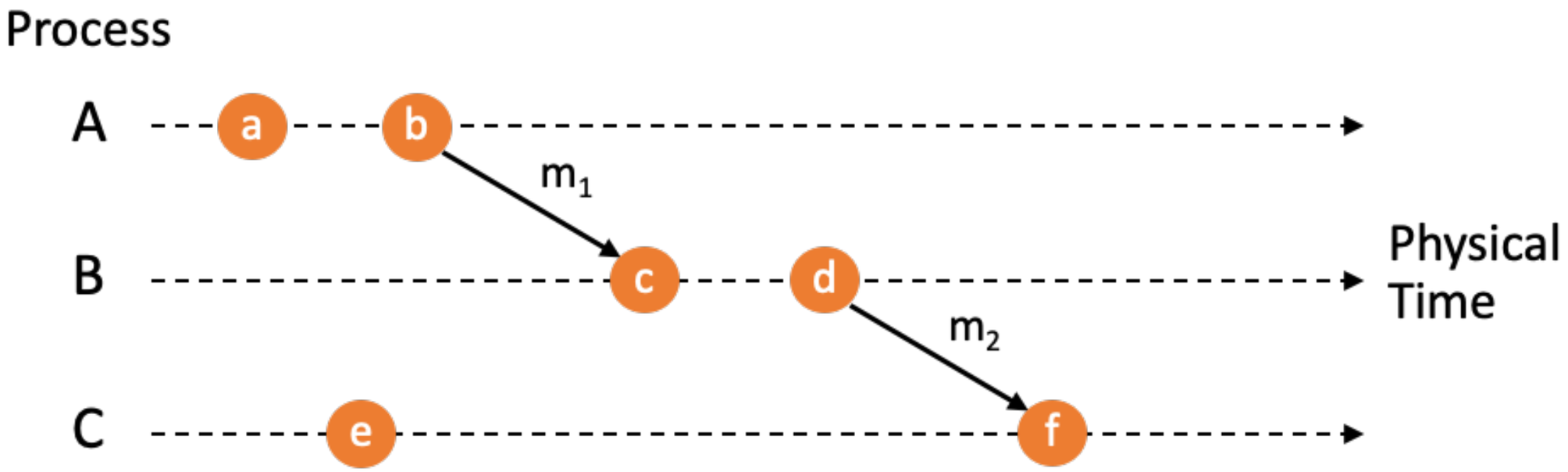
Understand Ordering

- In distributed system, there is no global clock and events can happen concurrently at different nodes
- Ordering is the result of execution based on certain order of events (e_z is executed/happened before e_y) that nodes agree
- Scope of agreements can be varied
  - FIFO ordering - orders are defined at a node that events are issued
  - Causal ordering - orders are defined based on happened-before relations
  - Total ordering - all nodes in the system agrees in one ordering - can be arbitrary-total (any order is OK, as long as everyone agrees), FIFO-total, and causal-total
- Ordering is very important to ensure certain system behaviors

Happened-Before Relation Revisited

- happened-before relation is
  - A partial order of events that reflects a flow of information
  - HB1: for any events in process A, if e is executed before e' then $e \rightarrow e'$ (e is happened before e')
- HB2: for any message m, $send(m) \rightarrow recv(m)$
- HB3: if $e \rightarrow e'$ and $e' \rightarrow e''$ then $e \rightarrow e''$

Causal Ordering Revisited

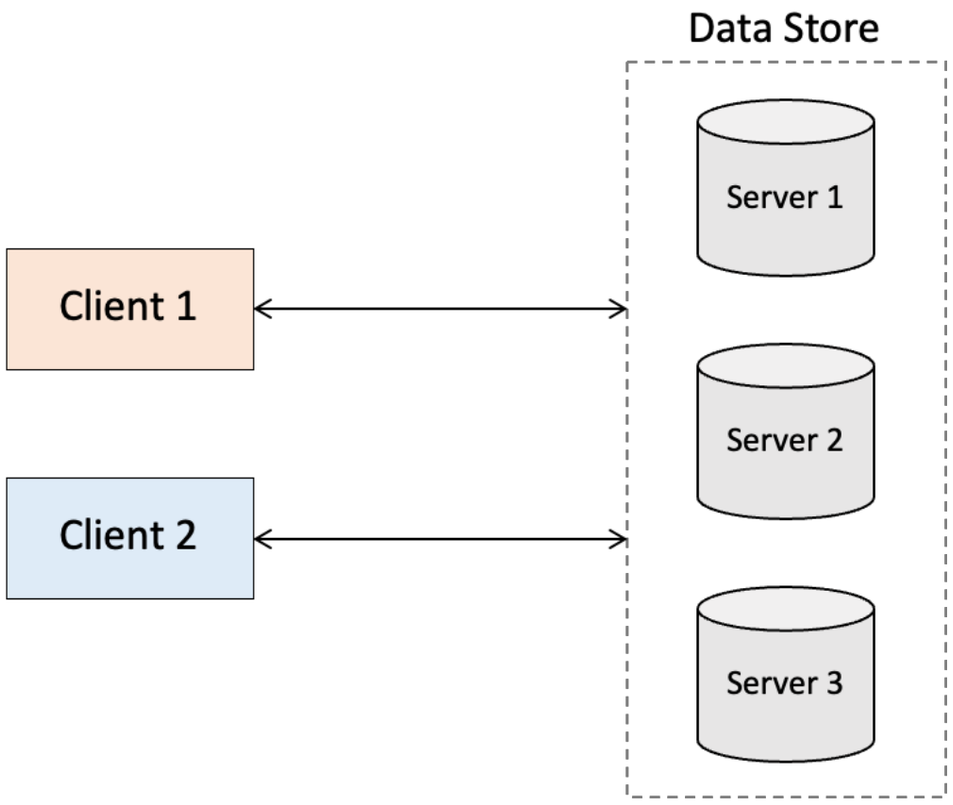


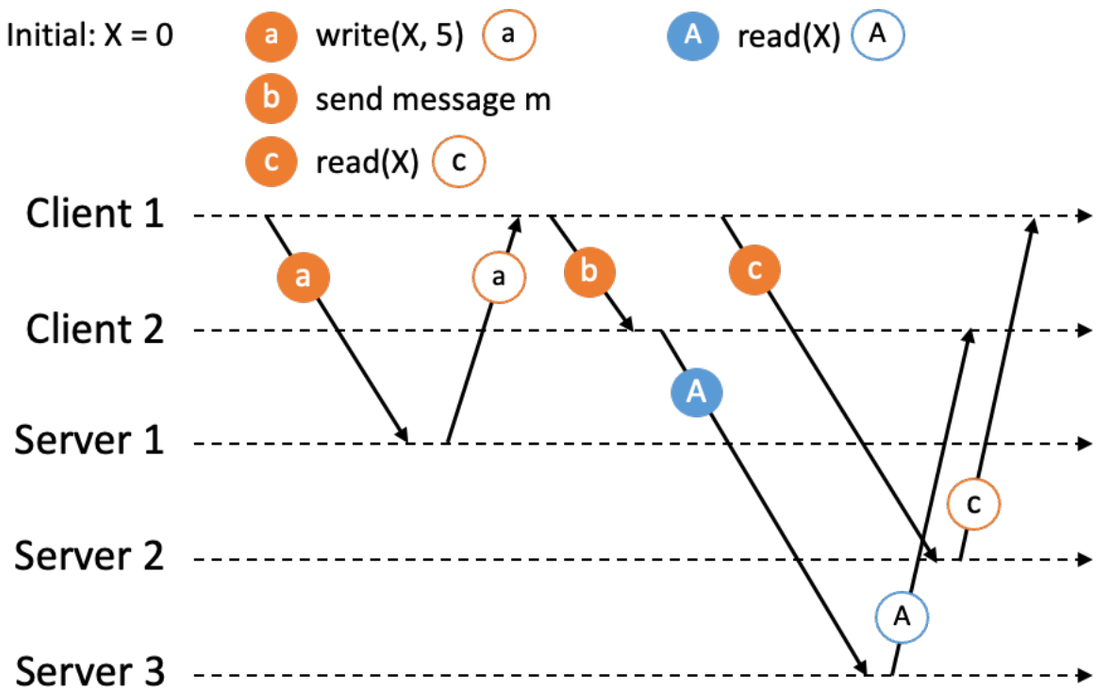
From HB1: $a \rightarrow b$ (at A) $c \rightarrow d$ (at B) $e \rightarrow f$ (at C)

From HB2: $b \rightarrow c$ (at m_1) $d \rightarrow f$ (at m_2)

From HB3: $a \rightarrow c$ $b \rightarrow d$ … $a \rightarrow f$

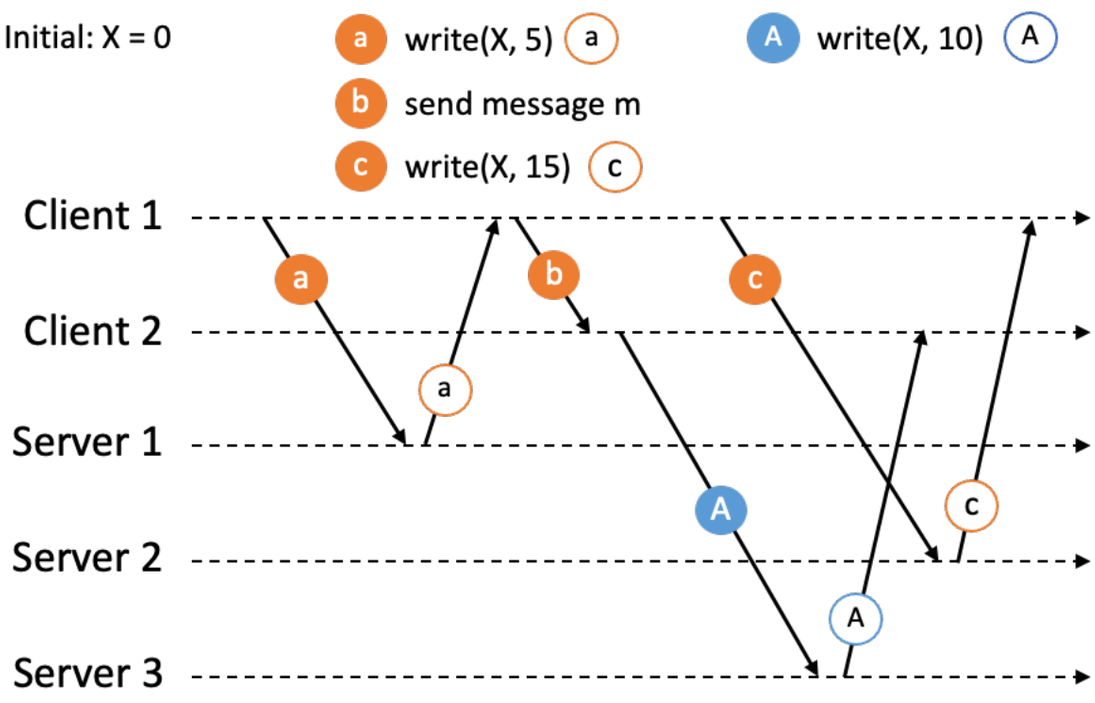
Ordering and Data Store





- Suppose a client can send a request to any server and the system guarantees ordering
- For FIFO ordering: as long as execute (c) after (a)
- Causal ordering - as long as execute (A) after (a)

Total Ordering ก็หมดว่าอะไร update ก่อนหลัง



- All servers will execute requests in the same order
- Arbitrary-total - in any order
- FIFO-total - as long as execute (c) after (a)
- Causal-total - as long as execute (A) after (a)

คล้าย Google Chubby

Apache Zookeeper

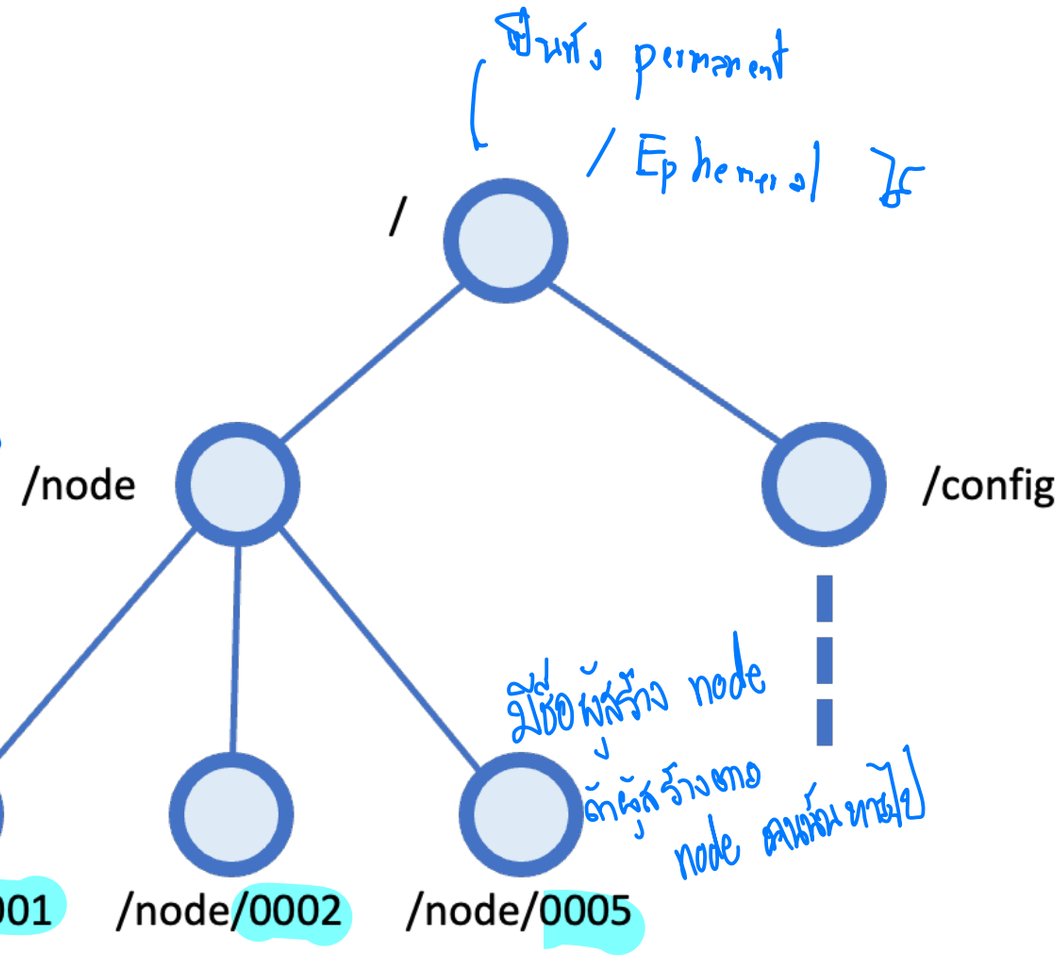
Coordination

Apache ZooKeeper

- A distributed hierarchical key-value store with services for configuration management and group services
- Provide services similar to Google's Chubby for coordinating processes of distributed applications
  - group messaging
  - wait-free shared registers (key-value datastore)
  - distributed lock services (distributed mutex)
- Strong consistency and ordering guaranteed
- Perform replication among ZooKeeper nodes to provide reliability and performance improvement

Zookeeper Primitive: Znodes

- ZooKeeper provides in-memory data objects (znodes) with 1MB storage and meta-data
- Znodes are organized in hierarchical name space
- Node lifetime
  - Permanent (manually delete)
  - Ephemeral (creator's session lifetime)
- Node's name (sequential flag)
  - Flag not set: normal name
  - Flag set: a new node will have a monotonically increasing counter appended to its name
- Client can ask zookeeper to notify if there is any change to a specific znode (watch)



ZooKeeper Session

- Client is assigned a session when connecting to ZooKeeper
- Client becomes faulty if fails to communicate with ZooKeeper within a period of time
- Session ends when client closes the session or ZooKeeper detects that a client is faulty
- Session persists across ZooKeeper servers allowing client to connect to any server using the same session

ZooKeeper API

```
create(path, data, flags)

getData(path, watch)

setData(path, data, version)

delete(path, version)

exists(path, watch)

getChildren(path, watch)

sync(path)
```

Running Zookeeper

- The simplest way to run a zookeeper instance is to use docker

```
docker pull wurstmeister/zookeeper
docker run -d --rm --name zookeeper -p 2181:2181 wurstmeister/zookeeper
```

- This will start zookeeper in your docker at port 2181 and map the port to your localhost

- You can also use docker-compose.yml and other example files in redis folder in datasci_architecture repo

- Kazoo - standard python package for Zookeeper client

```
pip install kazoo
```

ZooKeeper API - Python Kazoo

```
create(path, data, flags)

get(path, watch)

set(path, data, version)

delete(path, version)

exists(path, watch)

get_children(path, watch)

sync(path)
```

Example: Simple Kazoo Client

```
from kazoo.client import KazooClient

def zk_tree(zk, path, ident=''):
    print(ident, path)
    nextident = ident + '  '
    for c in zk.get_children(path):
        nextpath = path + ('' if path.endswith('/') else '/') + c
        zk_tree(zk, nextpath, nextident)

zk = KazooClient(hosts='localhost')
zk.start()
zk.ensure_path('/mycluster/nodes')
zn_node = zk.create('/mycluster/nodes/', ephemeral=True, sequence=True, makepath=True)
print('zn_node =', zn_node)
zk_tree(zk, '/mycluster')

zk2 = KazooClient(hosts='localhost')
zk2.start()
zn_node2 = zk2.create('/mycluster/nodes/', ephemeral=True, sequence=True, makepath=True)
print('zn_node2 =', zn_node2)
zk_tree(zk, '/mycluster')
zk2.stop()
print('after zk2 stops')
zk_tree(zk, '/mycluster')
zk.stop()

zk.start()
print('after both zk and zk2 stop')
zk_tree(zk, '/mycluster')
zk.delete('/mycluster', recursive=True)
zk.stop()
```

Results

```
...
zk.start()
zk.ensure_path('/mycluster/nodes')
zn_node = zk.create(...)
print('zn_node =', zn_node)
zk_tree(zk, '/mycluster')

zk2 = KazooClient(...)
zk2.start()
zn_node2 = zk2.create(...)
print('zn_node2 =', zn_node2)
zk_tree(zk, '/mycluster')
zk2.stop()
print('after zk2 stops')
zk_tree(zk, '/mycluster')
zk.stop()

zk.start()
print('after both zk and zk2 stop')
zk_tree(zk, '/mycluster')
...
```

```
zn_node = /mycluster/nodes/0000000000
 /mycluster
   /mycluster/nodes
     /mycluster/nodes/0000000000
Session 2
zn_node2 = /mycluster/nodes/0000000001
 /mycluster
   /mycluster/nodes
     /mycluster/nodes/0000000000
     /mycluster/nodes/0000000001
after zk2 stops
 /mycluster
   /mycluster/nodes
     /mycluster/nodes/0000000000
after both zk and zk2 stop
 /mycluster
   /mycluster/nodes
```

เห็น node ของ _0

ไม่เห็นอันไหนหลัง stop

สังเกตว่า terminate ไม่มีผล เวลาสร้างใหม่ก็ไม่เริ่ม id = 0 หรือเริ่ม id = 2

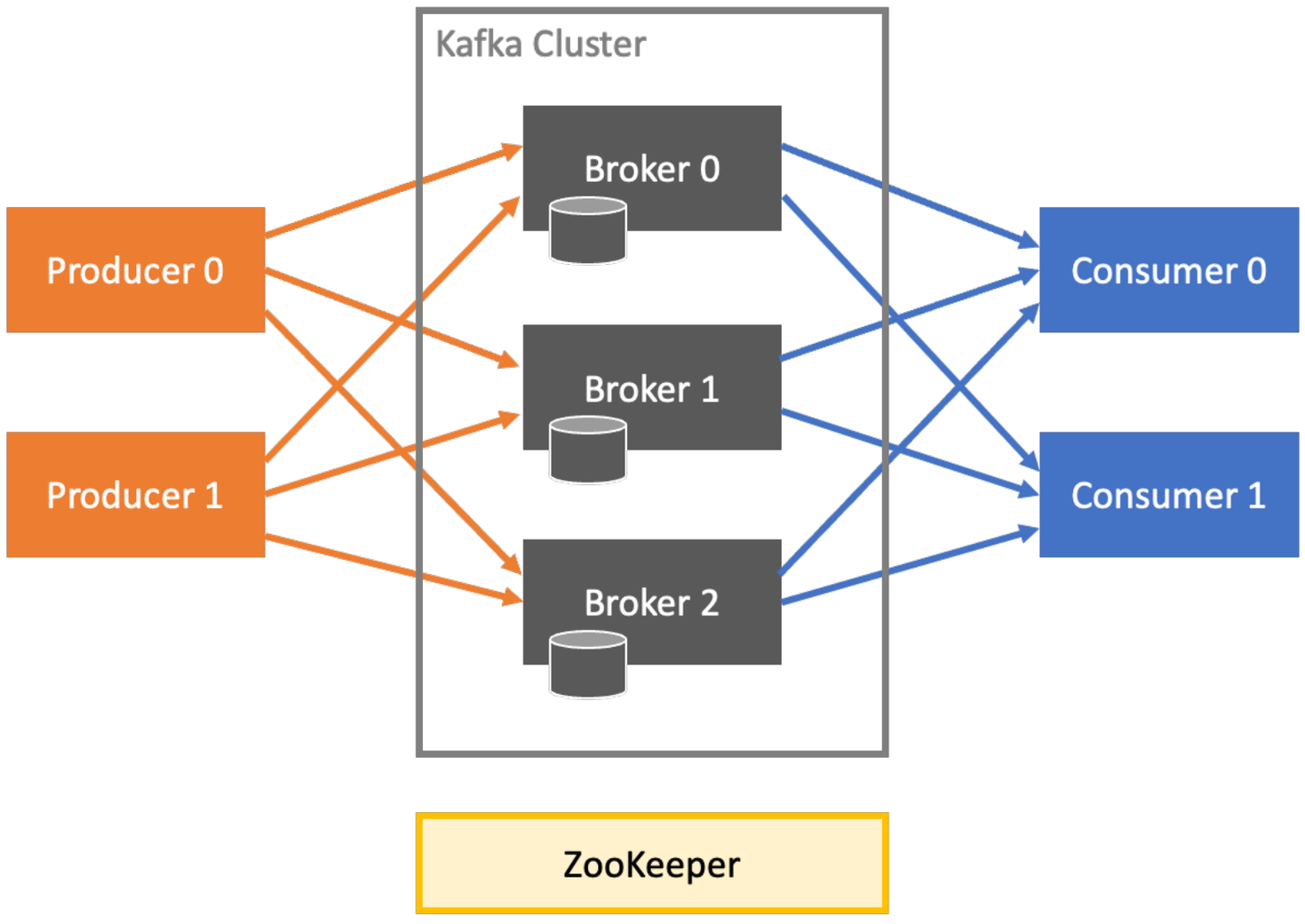
Example: Leader Election with Zookeeper

- Zookeeper can simplify several group services e.g. electing a leader of the group
- Elect a leader:
  - All members join a group and get unique ids by creating their own ephemeral znode in /members/ with sequential flag set and using the znode name as id
  - All members enter election by creating an ephemeral znode /leader; only one member will be successful and that member becomes the leader
  - Other non-leader nodes watch /leader znode, if the leader dies, /leader znode will be deleted and everyone re-enter election process again
- See more details in leader_election.py

คนที่สร้าง node ได้จะเป็น leader ที่เหลือเป็น watch leader (ephimeral)
ที่คอยรุมแทง leader แล้วถ้าตายก็สร้างใหม่ (รุมแห่ดันแทง)

message broker

Example: Kafka and ZooKeeper



Example: Kafka and ZooKeeper

- Data in ZooKeeper for kafka cluster
  - 2 brokers
    - nodes contain metadata
    - ephemeral nodes
  - 3 topics
    - contain responsible info
    - regular nodes (ephemeralOwner=0)

```
/
/cluster
/cluster/id
/controller_epoch
/controller
/brokers
/brokers/ids
/brokers/ids/1002
/brokers/ids/1001
/brokers/topics
/brokers/topics/twitter
/brokers/topics/twitter/partitions
/brokers/topics/twitter/partitions/0
/brokers/topics/twitter/partitions/0/state
/brokers/topics/sample
/brokers/topics/sample/partitions
/brokers/topics/sample/partitions/0
/brokers/topics/sample/partitions/0/state
/brokers/topics/sample/partitions/1
/brokers/topics/sample/partitions/1/state
/brokers/topics/avro
/brokers/topics/avro/partitions
/brokers/topics/avro/partitions/0
/brokers/topics/avro/partitions/0/state
/brokers/topics/avro/partitions/1
/brokers/topics/avro/partitions/1/state
/brokers/seqid
```

broker id ใน cluster

topic 1

topic 2

topic 3

client คุย zk เหมือน kafka ได้

save ลง zk ให้หาต่อได้

```
zk.get('/brokers/ids/1001')
```

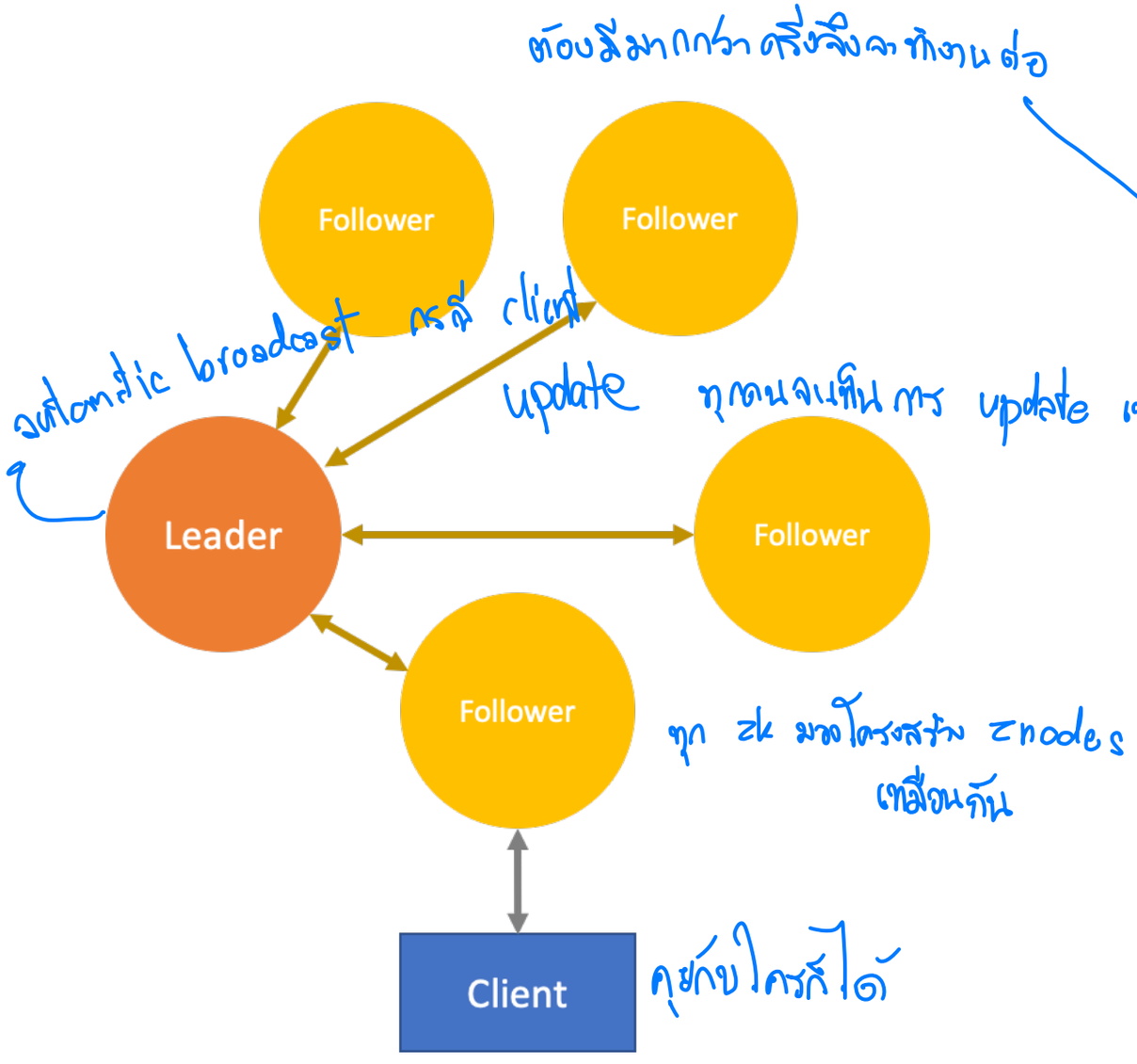
```
(b'{"features":{},"listener_security_protocol_map":{"PLAINTEXT":"PLAINTEXT"},"endpoints":["PLAINTEXT://192.168.68.137:9093"],"jmx_port":-1,"port":9093,"host":"192.168.68.137","version":5,"timestamp":"1630329937692"}',
 ZnodeStat(czxid=41, mzxid=41, ctime=1630329937720, mtime=1630329937720, version=1, cversion=0, aversion=0, ephemeralOwner=72065036800950272, dataLength=212, numChildren=0, pzxid=41))
```

topic : twitter

```
zk.get("/brokers/topics/twitter")
```

```
(b'{"version":2,"partitions":{"0":[1002]},"adding_replicas":{},"removing_replicas":{}}',
 ZnodeStat(czxid=72, mzxid=72, ctime=1630329946816, mtime=1630329946816, version=0, cversion=1, aversion=0, ephemeralOwner=0, dataLength=83, numChildren=1, pzxid=74))
```

ZooKeeper Architecture



- ZooKeeper cluster is called **ensemble**
- Ensemble is still working as long as half of the members are still functioned
- There is one leader who handles all information updating
- Once updated, leader sends an atomic broadcast to update all other nodes (followers)
- Both leader and followers provide services to clients
- If the leader fails, one of the followers will become a leader
- ZooKeeper guarantees
  - Linearizable writes
  - FIFO client order

Apache Yarn เพิ่มใน Hadoop v.2

Coordination

Apache Yarn

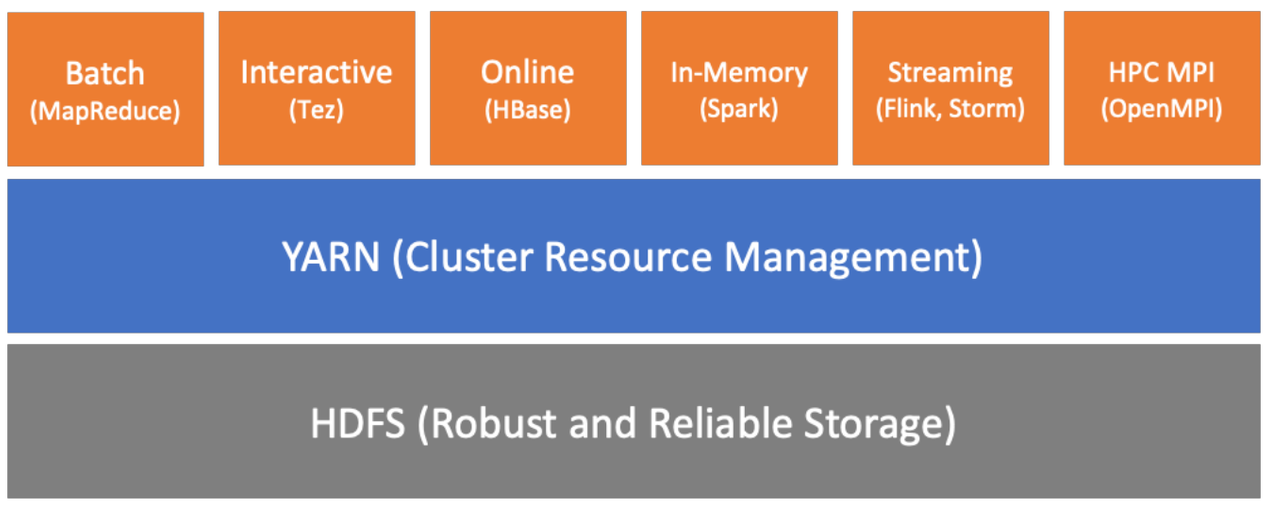
HDFS เอา data ไว้ใน dataNode

เมื่อรัน mapReduce Hadoop จะโยน code ที่ data (เลือก node ที่มี data ที่ต้องการจะ run แล้วเอาโปรแกรมไป)

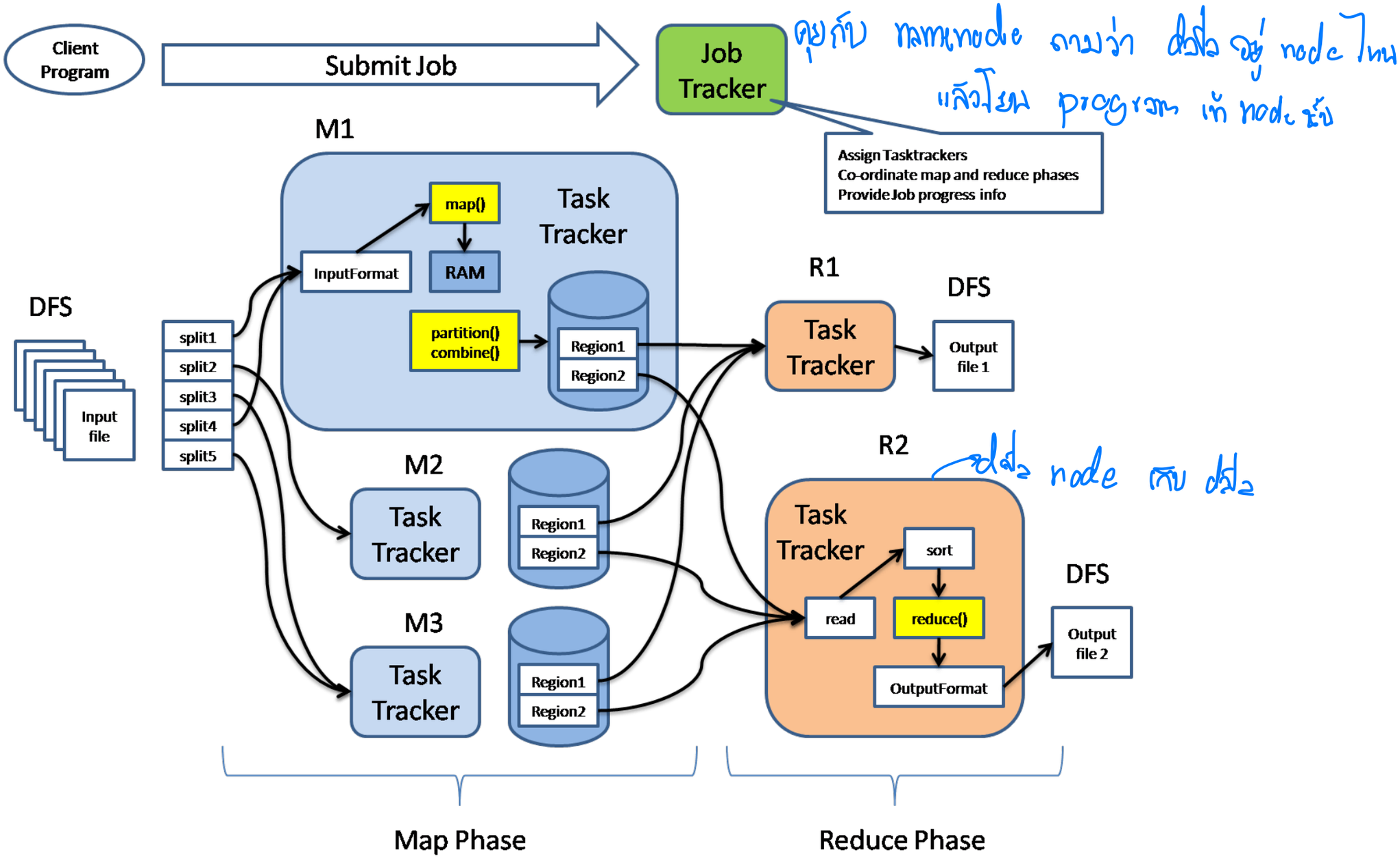
- Hadoop version 2 introduced a new resource manager called Yarn (Yet Another Resource Negotiator)
- Replace JobTracker, which was inflexible and limited scalability
- Provide lots of functionality: scalability, multi-tenancy, reliability, etc.
- Provide REST API
- Flexible for many programming models e.g. MapReduce, Spark, Flint, etc.

↑ เดิมเรียก JobTracker ที่ใช้ mapReduce โดยเฉพาะ จึงเกิด YARN เพื่อความหลากหลายของ model

↳ มีหลายงานที่พร้อมทำ เช่น เงินอยู่คนละ programming model

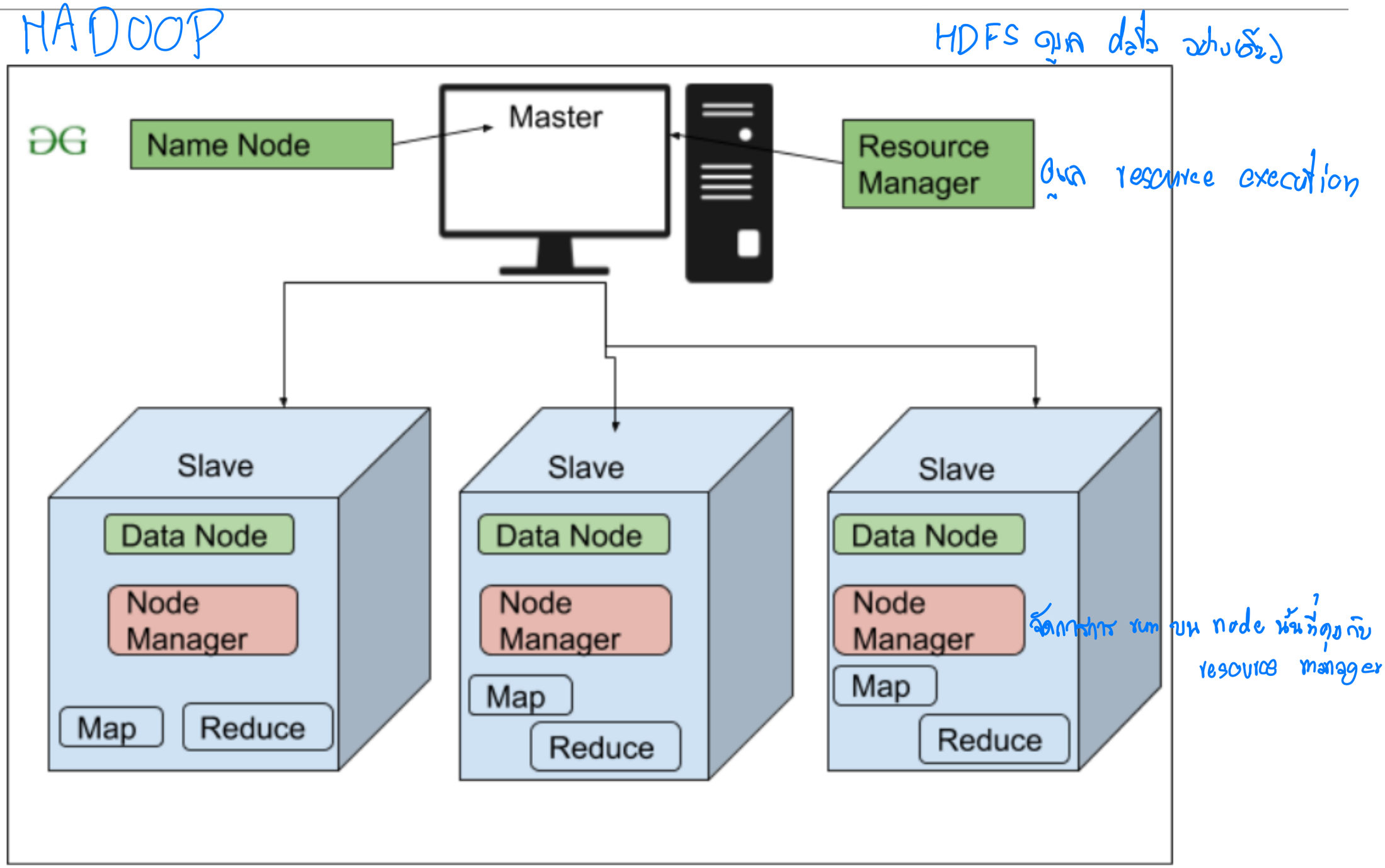


Hadoop Version 1 - JobTracker



https://www.section.io/engineering-education/understanding-map-reduce-in-hadoop/

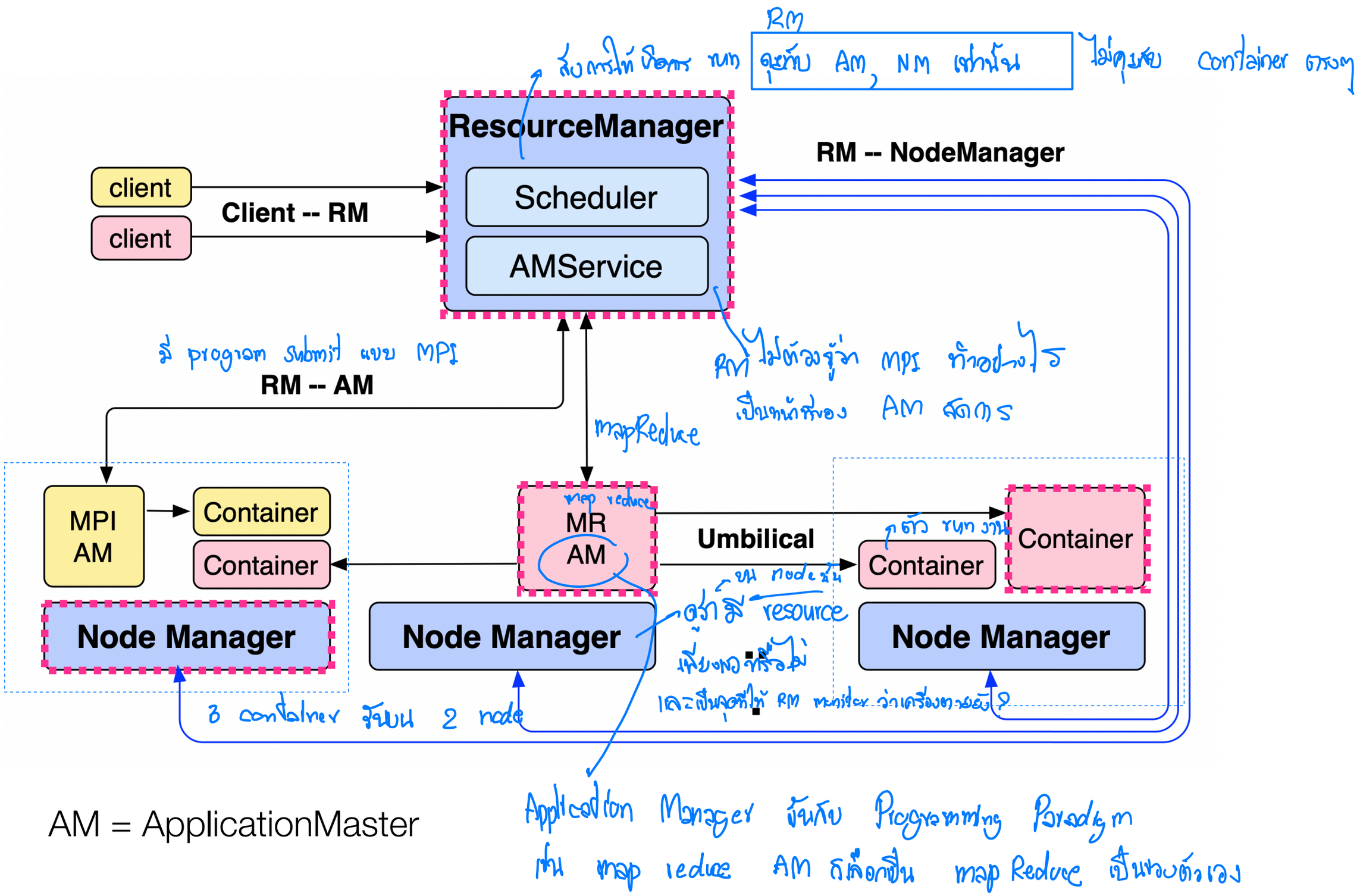
Hadoop Version 2 and Later



Source: https://www.geeksforgeeks.org/hadoop-architecture/

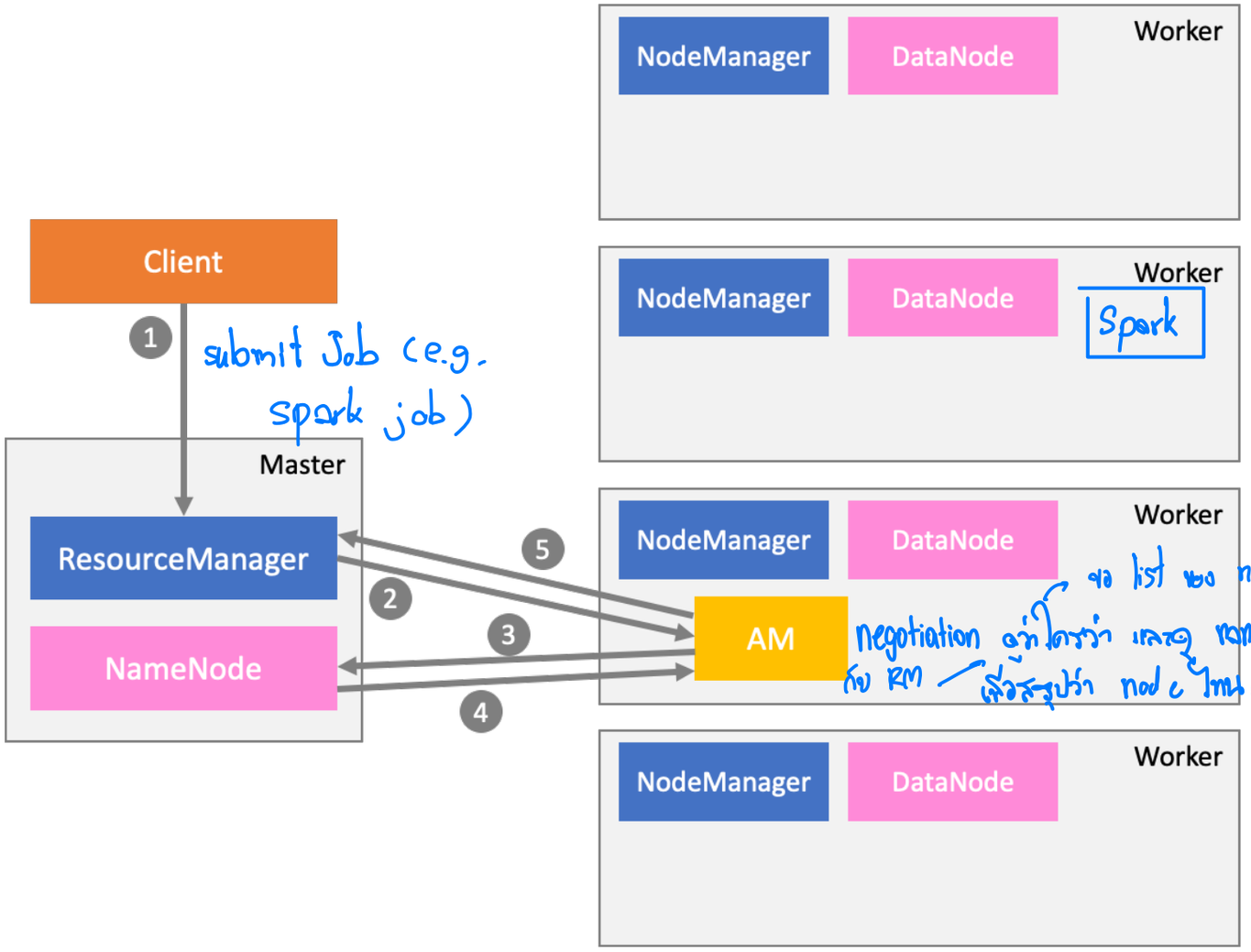
RM สั่งการให้เกิดการ run แต่ detail การ run จะมี AM เกิดขึ้น

YARN Architecture



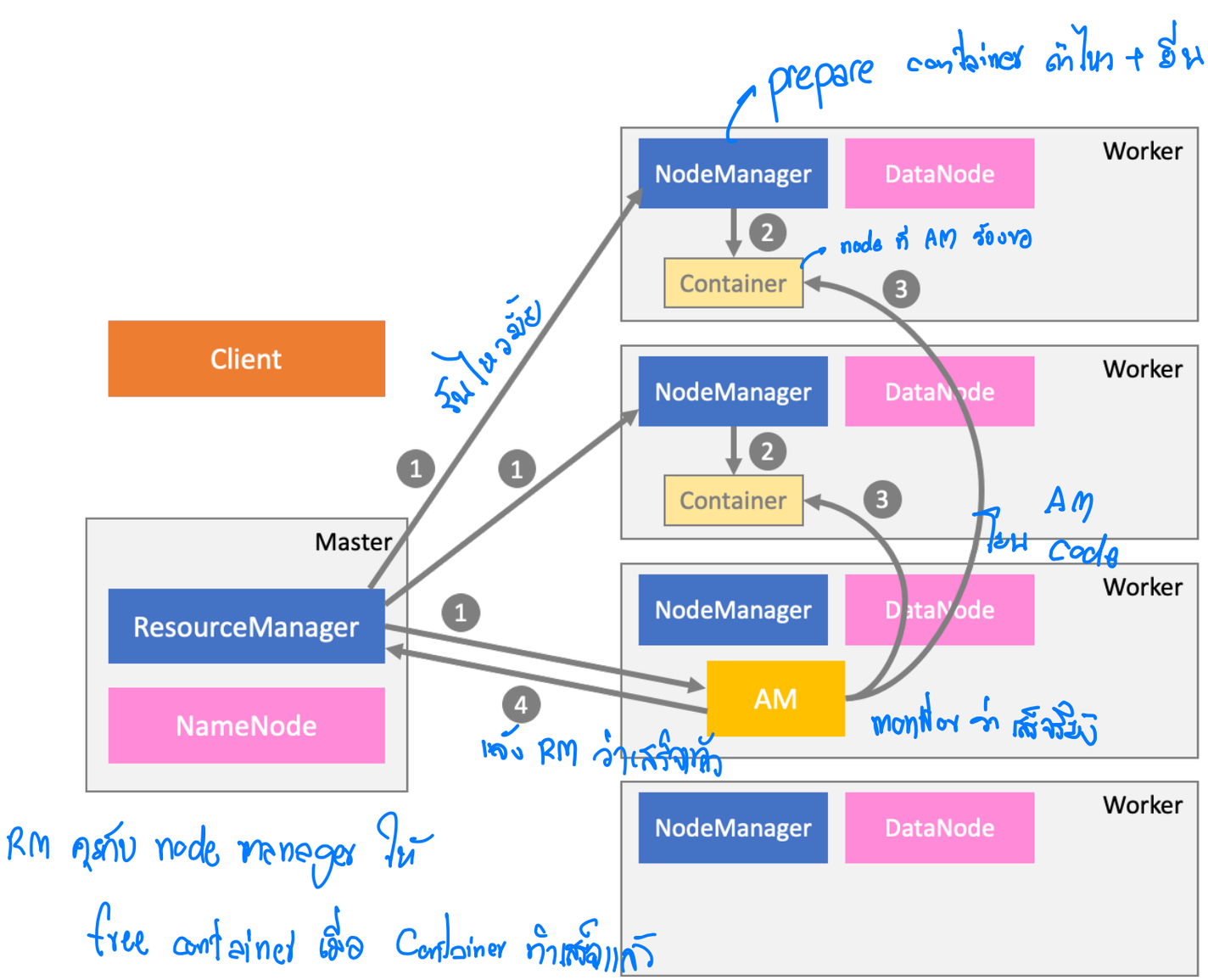
AM = ApplicationMaster

Resource Negotiation



1. Client requests resources from RM
2. RM reserves a container from a worker to run AM
3. AM asks NameNode a list of DataNode with data to-be-processed in HDFS
4. NameNode provides the list
5. AM chooses DataNodes and sends request to RM

Resource Usage



1. RM informs NodeManagers to prepare containers for AM
2. NodeManagers prepare containers
3. AM submits tasks to containers
4. AM informs RM to release containers when execution is complete

Yarn Scalability and Reliability

- There are 4 possible failure points ตัวหลัก
  - Container - AM asks for a new container from RM AM ขอ container ใหม่
  - AM - RM monitors AM status and restart AM in a new container (if necessary). It is AM responsibility to recover from the failures RM จะรู้ว่า AM ตาย ที่ node ใหม่ แล้วรัน AM และ AM ตัวใหม่ต้อง recovery เอง
  - NodeManager - RM monitors NodeManager and removes failed NodeManager and its containers. RM informs AM to recovery from the failures RM จะรู้ว่าตาย ก็ลบ NM ออก (สิ่งที่หายคือ container ด้วย) แล้ว AM ว่า NM ตาย ไปจัดการที่ AM ก็ negotiate กับ RM ที่ใหม่ให้ container
  - RM - used to be a single point of failure, however, has been developed with active-standby model in the later versions ตายสถานเดียว Hadoop v.3 RM มีการทำ RM backup (Journal node) RM เป็น Active, standby

Conclusion

- In large-scale cluster, coordinating between components become important, including:
  - group membership and leader election
  - resource management / placement
  - meta-data / configuration information storages
  - locking mechanism
- Apache ZooKeeper provides centralized services for coordination and synchronization
- Apache Yarn provides resource management services with flexibility
- All centralized services must have failure recovery processes as failure can happen

Reading Assignments

- P. Hunt et al, “ZooKeeper: Wait-free coordination for Internet-scale systems”, USENIX annual technical conference. Vol. 8. No. 9. 2010.

- V. Vavilapalli et al, “Apache Hadoop YARN: Yet Another Resource Negotiator”, Proceedings of the 4th annual Symposium on Cloud Computing. 2013.