2110625 - Data Science Architecture

Wide Column Stores

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

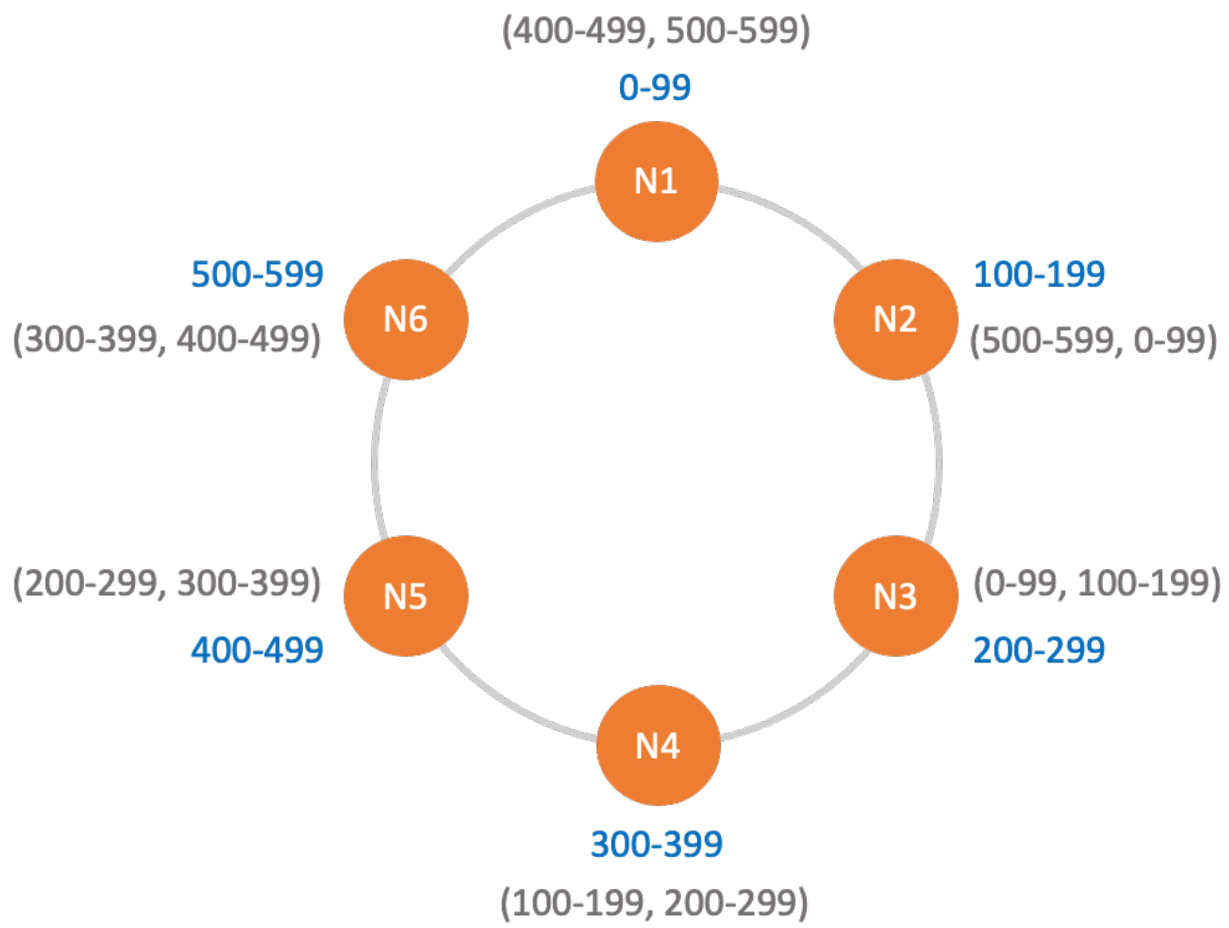
CHULA ENGINEERING
Foundation toward Innovation

Wide Column Store

- A column-oriented DBMS with tables, rows, and columns
- Names and format of the columns can vary from row to row in the same table
- A wide-column store can be interpreted as a two-dimensional key–value store with group of columns are stored in the same server
- Often support the notion of column families
  - Each column family typically contains multiple columns that are used together
  - Within a given column family, all data is stored in a row-by-row fashion, such that the columns for a given row are stored together

Apache
CASSANDRA

peer-to-peer ไม่มี leader
ทุกคนศักดิ์ศรีเท่ากัน

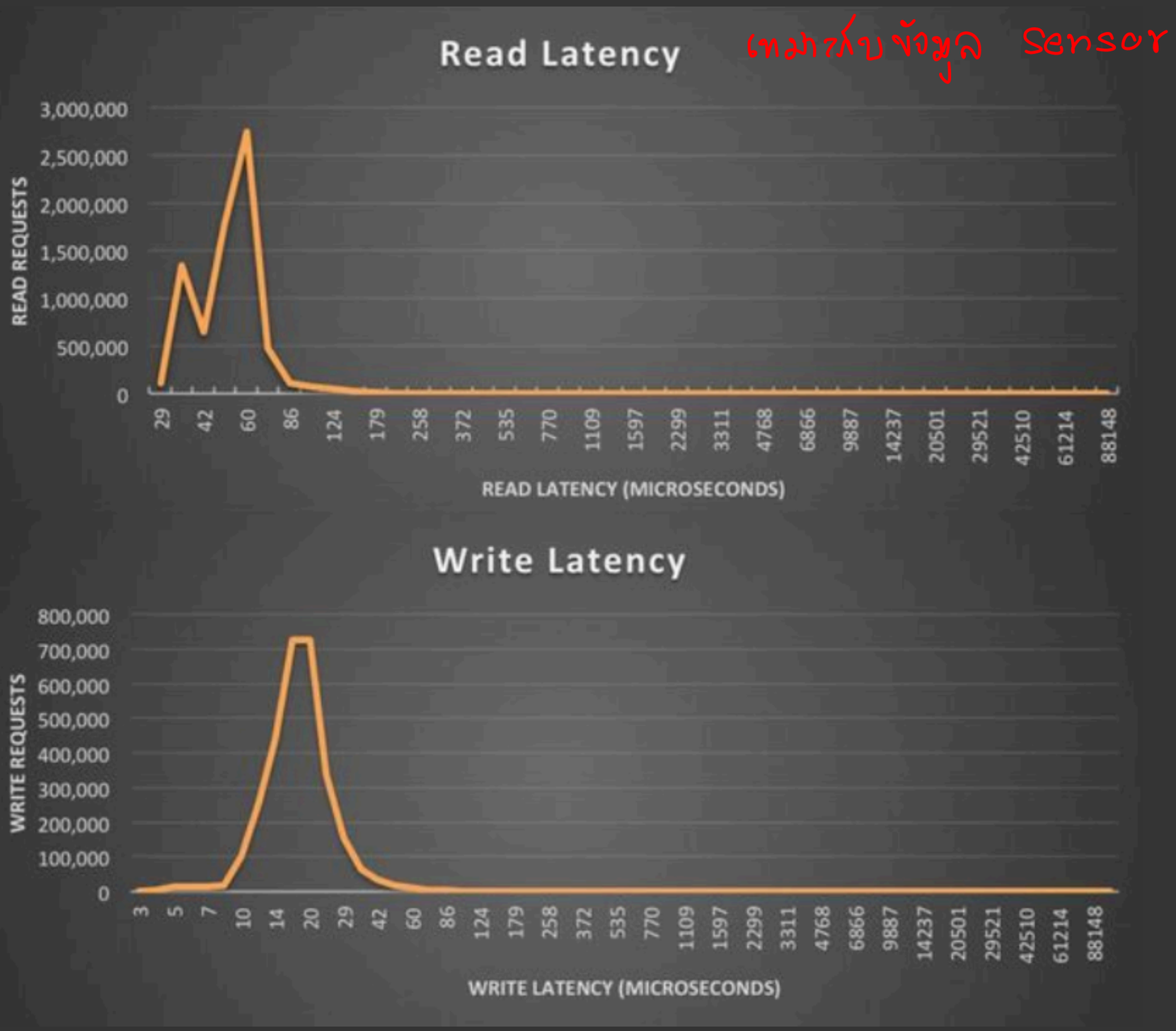


Apache Cassandra

Cassandra

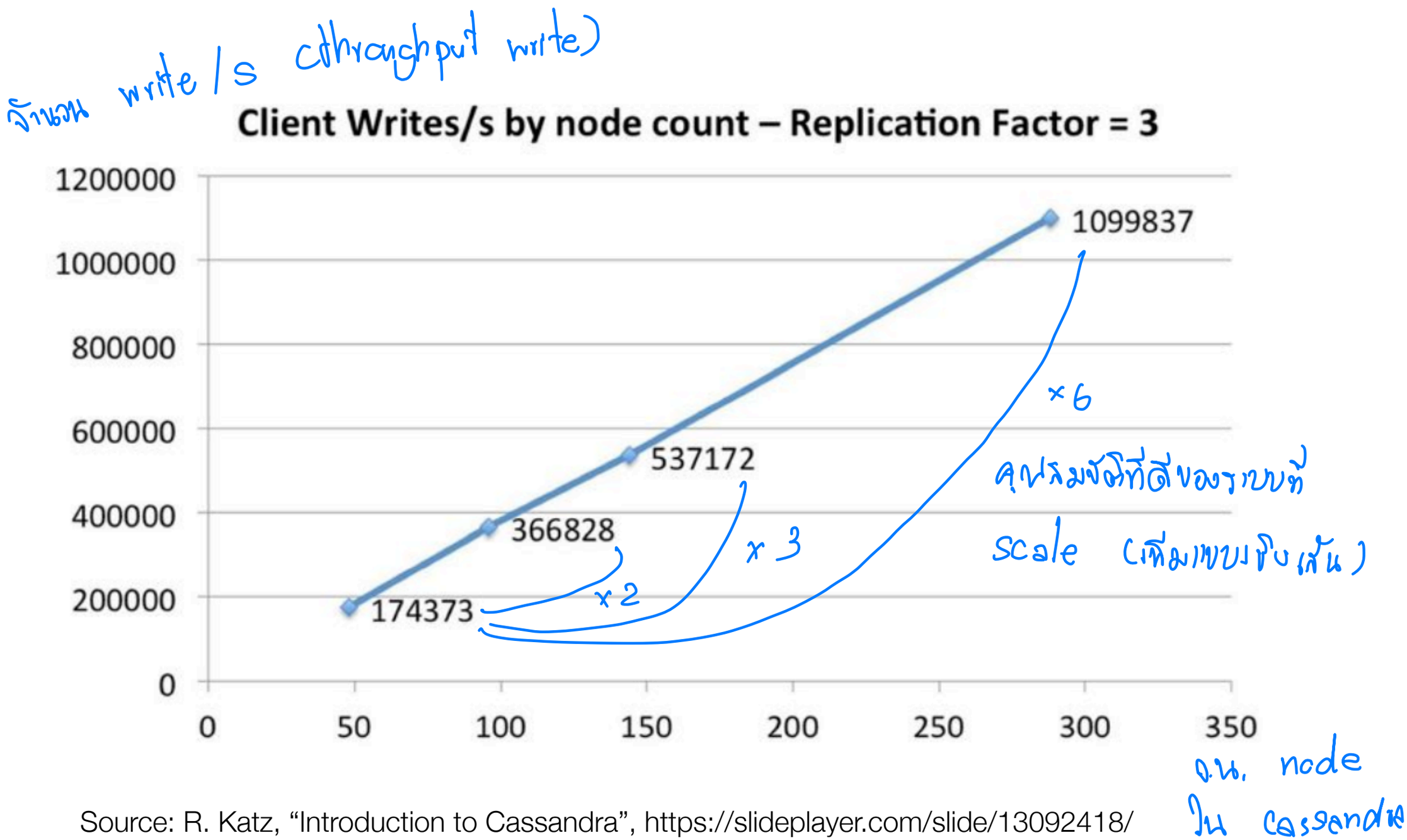
- Distributed storage system for large volume structured data
- Adapted features from Amazon Dynamo and Google Bigtable with AP concept
- Designed for a cluster of hundreds of commodity servers across multiple data centers
- Provide high availability with no single point of failure
- Focus on low latency with high write throughput (adjustable)
- Query using CQL (similar to SQL)

ยิ่งเครื่องเยอะ ยิ่งเพิ่ม throughput



Source: R. Katz, “Introduction to Cassandra”, https://slideplayer.com/slide/13092418/

Cassandra is highly scalable



Source: R. Katz, “Introduction to Cassandra”, https://slideplayer.com/slide/13092418/

RDBMS vs. Cassandra

RDBMS	Cassandra
Database	Keyspace
Table	Table
Row / Tuple	Row
Column	Column
Table Join	Denormalization
Primary Key	Partition Key + Clustering Column

คล้าย sort key ใน dynamoDB

Cassandra Data Model

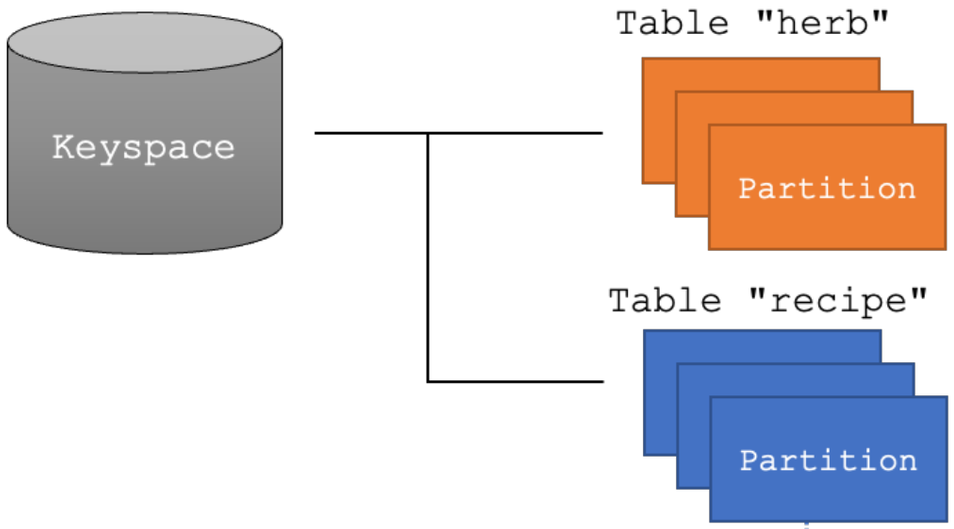
มีหลาย column ใน 1 row แต่ละ row อาจมี column ไม่เท่ากัน

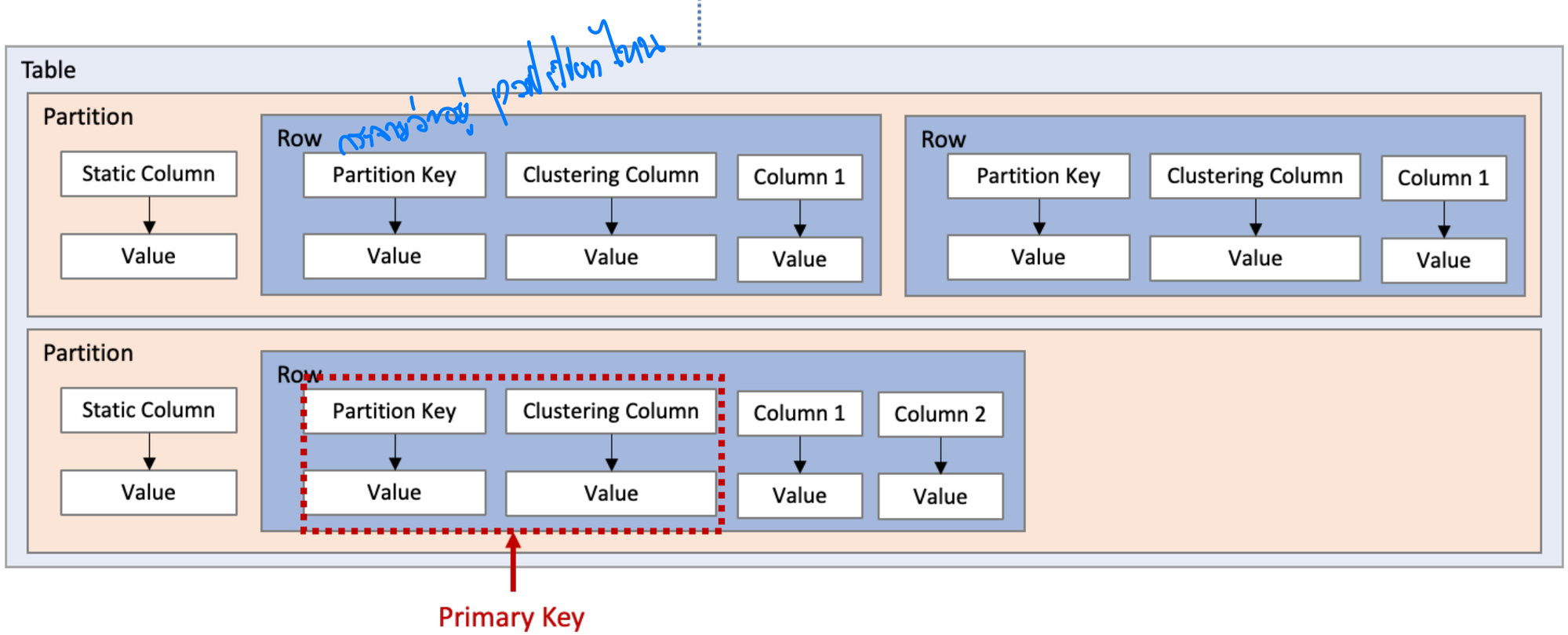
- Data model is query-driven, access patterns and application queries determine the structure of data

- Data in a table is divided into partitions using a column called ‘partition key’, which can be a composite key (multiple columns)

- Data in the same partition are sorted ascending or descending order based on the clustering columns

Cassandra Data Model





Cassandra Data Types

- Cassandra supports several data types including
  - Native types (see a table of examples in the next page)
  - Collection types (lists, sets, maps)
  - User defined types (a set of fields)
  - Tuple types (a list of anonymous fields of different types)
  - Custom types (Java class loaded by Cassandra)

Type	Constants supported	Description
bigint	integer	64-bit signed long
blob	blob	Arbitrary bytes (no validation)
boolean	boolean	Either true or false
counter	integer	Counter column (64-bit signed value)
date	integer, string	A date (with no corresponding time value)
decimal	integer, float	Variable-precision decimal
double	integer float	64-bit IEEE-754 floating point
duration	duration,	A duration with nanosecond precision
inet	string	An IP address: IPv4 (4 bytes long) or IPv6 (16 bytes long)
int	integer	32-bit signed int
text	string	UTF8 encoded string
timestamp	integer, string	A timestamp (date and time) with millisecond precision
uuid	uuid	A UUID (of any version)
varchar	string	UTF8 encoded string
varint	integer	Arbitrary-precision integer

There are other data types, refer to: https://cassandra.apache.org/doc/latest/cassandra/cql/types.html

Playing with Cassandra

- Server-side: use docker compose file from cassandra folder in datasci_architecture repo (this will use official cassandra image from docker hub)

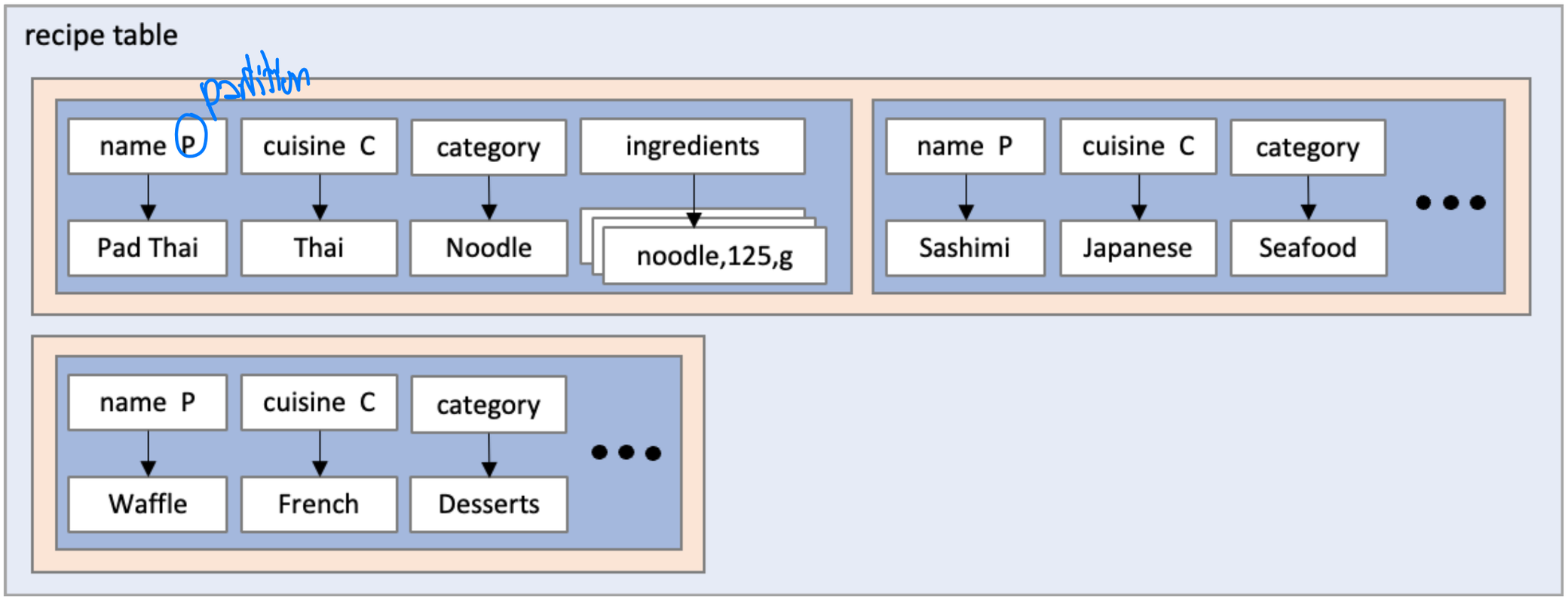
```
cd cassandra

sudo docker compose up
```

- From client-side, you use cqlsh in one of the cassandra nodes

```
sudo docker exec -it cs_0 cqlsh
```

- Note
  - You will need to set your docker desktop to have at least 4GBs
  - You can use nodetool to examine the cluster, see nodetool.sh for more details

Example


recipe table
name P
cuisine C
category
ingredients
Pad Thai
Thai
Noodle
noodle,125,g
name P
cuisine C
category
Sashimi
Japanese
Seafood
name P
cuisine C
category
Waffle
French
Desserts

Example CQL — See init.cql

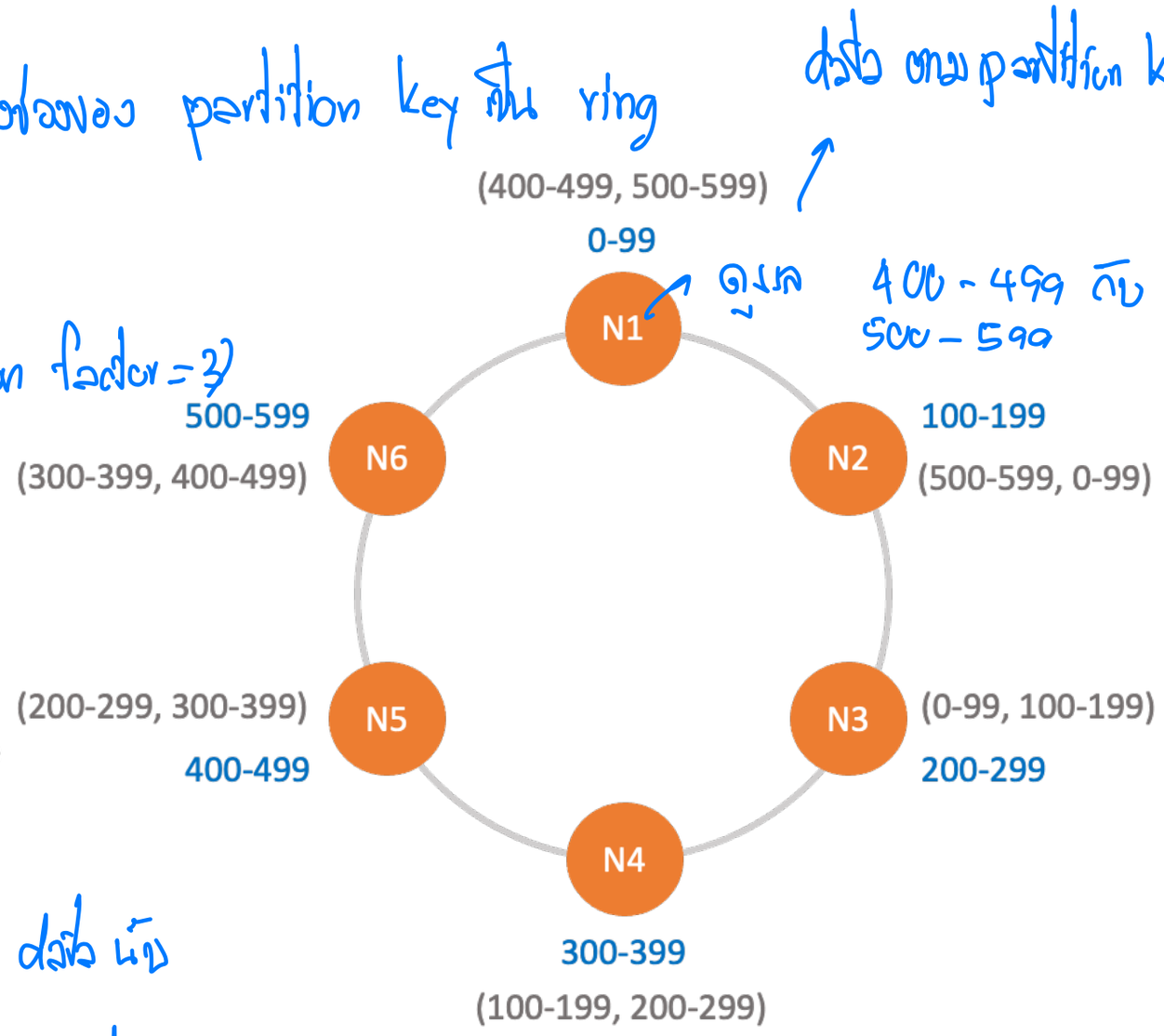
```
CREATE KEYSPACE restaurant WITH replication = {'class': 'SimpleStrategy',
'replication_factor' : 2};

USE restaurant;

CREATE TYPE ingredient (item text, quantity int, unit text);

CREATE TABLE recipe (name text, cuisine text, category text, ingredients
list<frozen<ingredient>>, primary key ((name), cuisine));

CREATE INDEX ON recipe (category);

INSERT INTO recipe (name, cuisine, category, ingredients) VALUES ('Pad_Thai',
'Thai', 'Noodle', [{item: 'noodle', quantity: 125, unit: 'g'}, {item: `shrimp',
quantity: 150, unit: 'g'}, {item: 'brown sugar', quantity: 3, unit: 'tbsp'}]);

INSERT INTO recipe (name, cuisine, category, ingredients) VALUES ('Sashimi',
'Japanese', 'Seafood', [{item: 'rice', quantity: 100, unit: 'g'}, {item:
'salmon', quantity: 50, unit: 'g'}]);

INSERT INTO recipe (name, cuisine, category, ingredients) VALUES ('Chicken_Rice',
'Thai', 'Rice', [{item: 'rice', quantity: 300, unit: 'g'}, {item: 'chicken',
quantity: 100, unit: 'g'}]);

INSERT INTO recipe (name, cuisine, category, ingredients) VALUES ('Tom_Yum',
'Thai', 'Soup', [{item: 'shrimp', quantity: 200, unit: 'g'}, {item: 'water',
quantity: 500, unit: 'g'}, {item: 'lemon grass', quantity: 10, unit: 'leaves'}]);
```

Example CQL — See init.cql

```
SELECT * FROM recipe;

SELECT * FROM recipe WHERE name='Pad_Thai';

SELECT * FROM recipe WHERE category='Noodle';

SELECT token(name), name, cuisine FROM recipe;
```

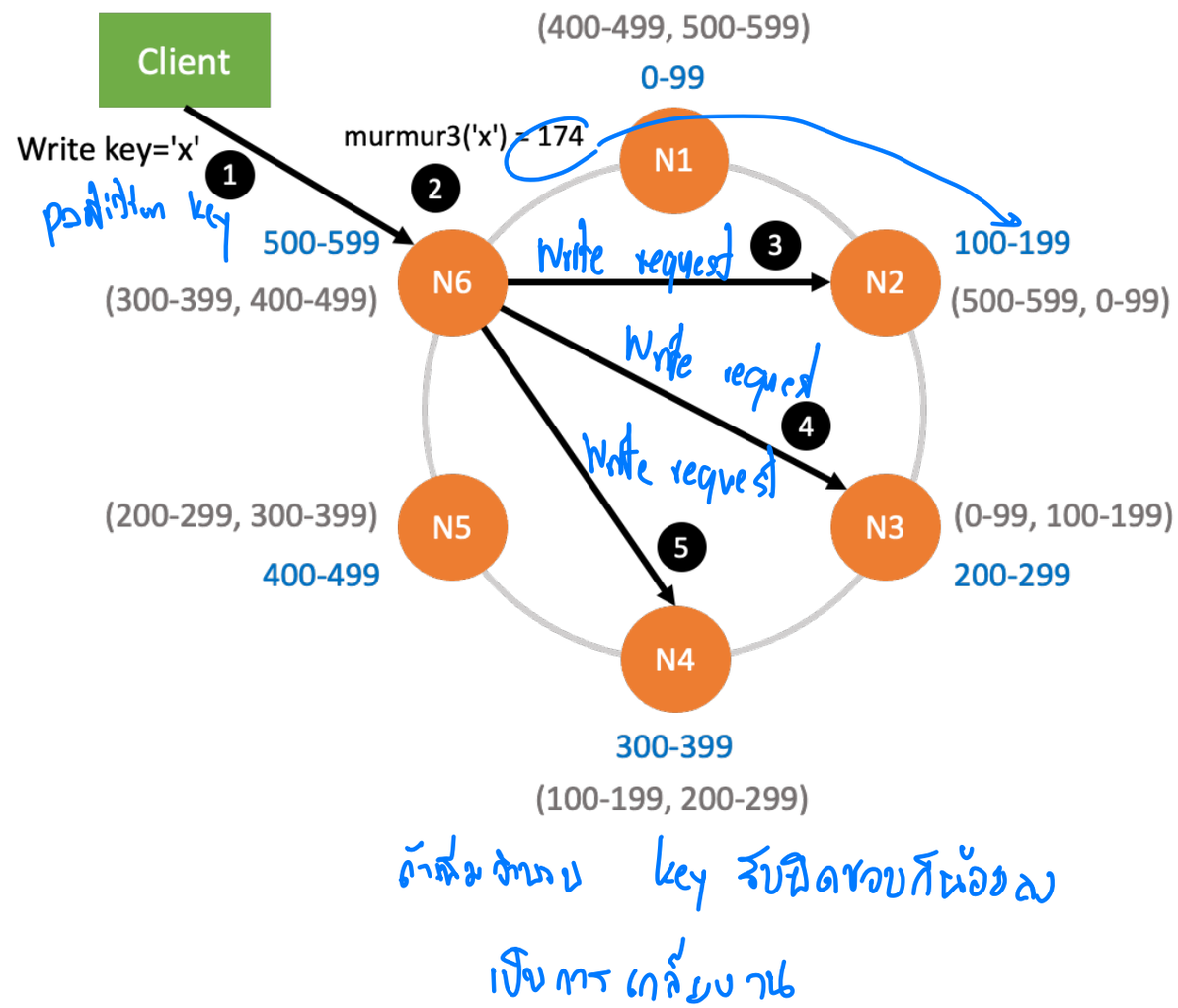
Cassandra Cluster

- Nodes in cluster form a ring topology
- A token is generated by hashing partition key with Murmur3 partitioning function
- Token value indicates the node to store the data
- Key space may not be evenly distributed
- Client can request to any node (that node becomes an “coordinator”) and the request will be routed to the node responsible for storing data
- Ring can be rebalanced
  - expand to accommodate more nodes
  - collapse to remove broken nodes



Replication

- Cassandra replicates data to provide availability
- Client can write to any node in the cluster (that node becomes a “coordinator”)
- The coordinator node creates a token by hashing a partition key
- Using the token, the coordinator can find all replicas

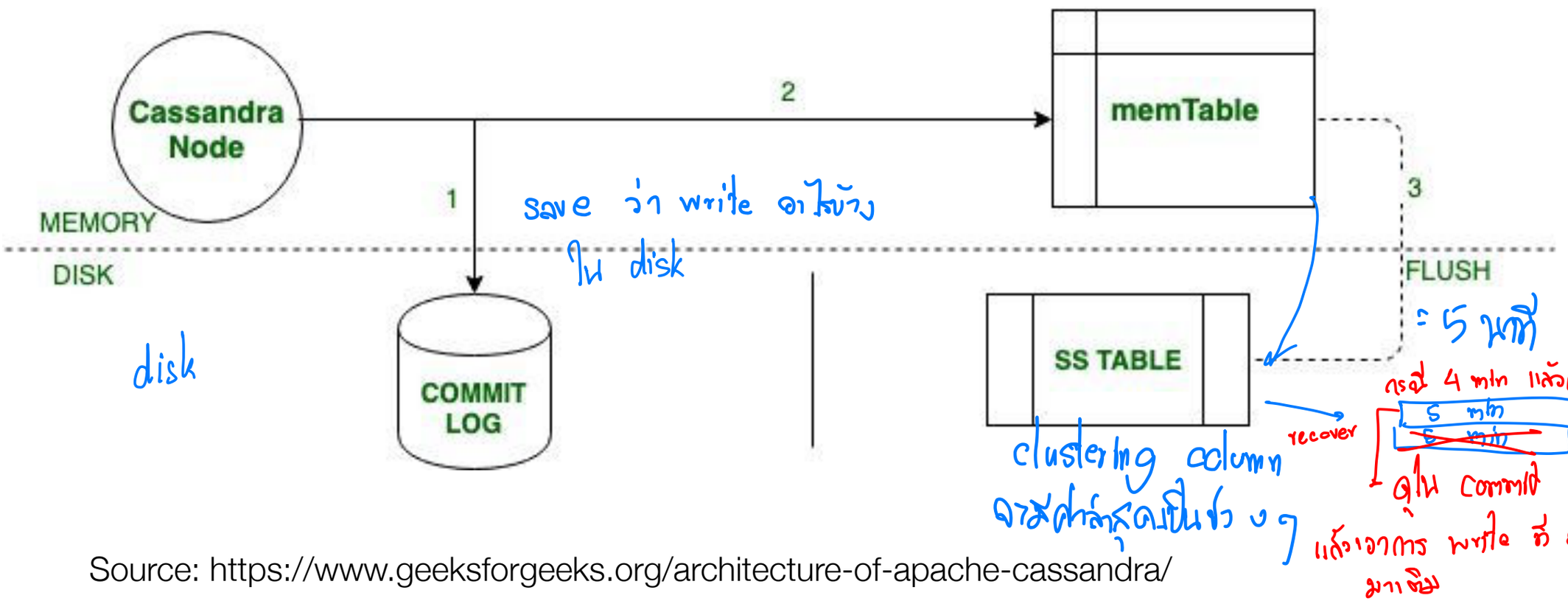


Read Operation

- In Read Operation there are three types of read requests that a coordinator can send to a replica. The node that accepts the write requests called coordinator for that particular operation
  - Direct Request: In this operation coordinator node sends the read request to one of the replicas
  - Digest Request: In this operation coordinator will contact to replicas specified by the consistency level. For Example: CONSISTENCY TWO; It simply means that Any two nodes in data center will acknowledge
  - Read Repair Request: If there is any case in which data is not consistent across the node then background Read Repair Request initiated that makes sure that the most recent data is available across the nodes

Write Operation

- Key concept: Data is kept in memory and lazily written to the disk
- Step-1: In Write Operation as soon as we receives request then it is first dumped into commit log to make sure that data is saved
- Step-2: Insertion of data into table that is also written in MemTable that holds the data till it's get full
- Step-3: If MemTable reaches its threshold then data is flushed to SS Table



Source: https://www.geeksforgeeks.org/architecture-of-apache-cassandra/

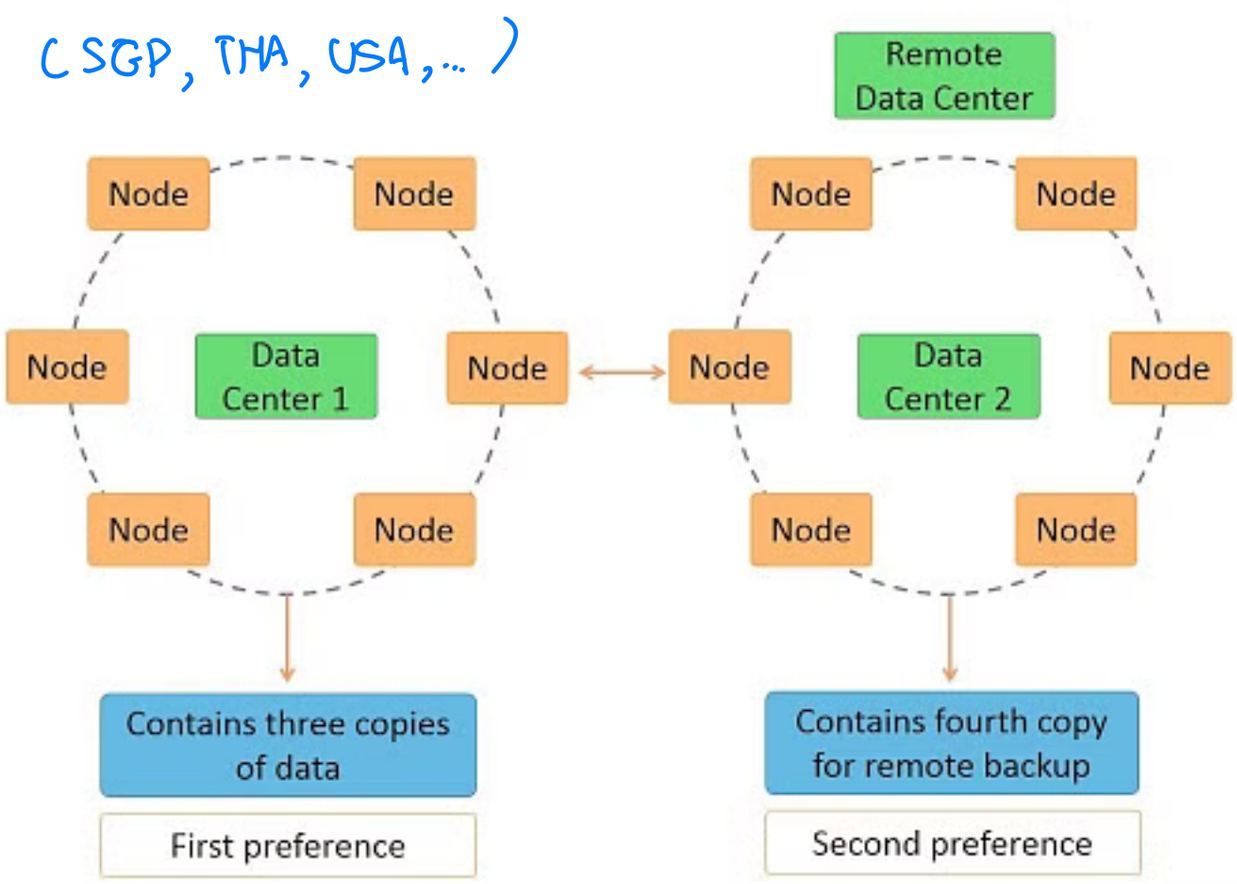
Cassandra มี ความ center-aware

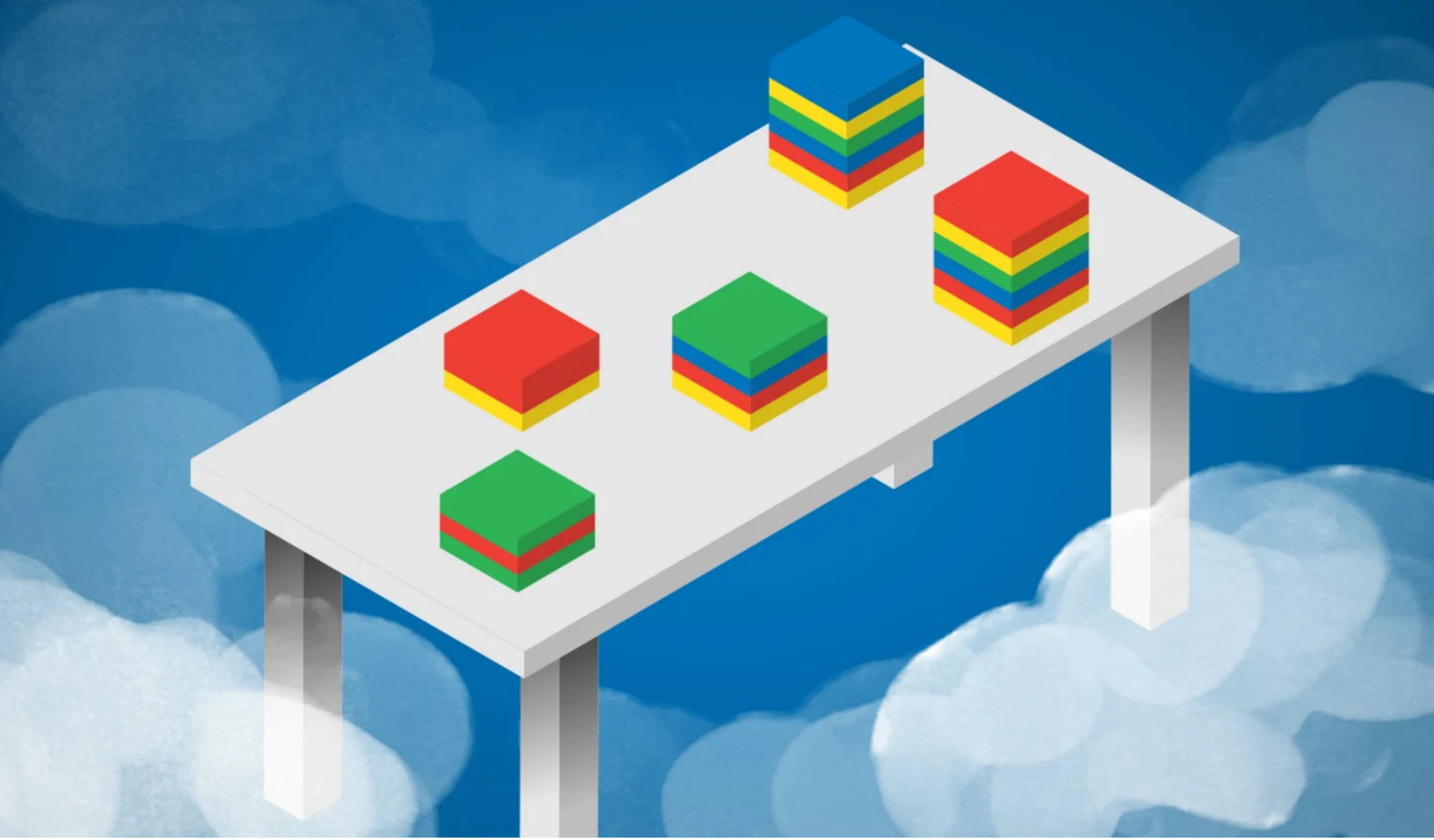
Additional Features

มี 1 copy อยู่คนละ rack

- Cassandra is rack-aware for availability (rack failure) and read performance (locality)
- Cassandra supports multiple data centers which data can be replicated across data centers
- See https://www.simplilearn.com/tutorials/big-data-tutorial/cassandra-architecture

ไม่ใช่ทุก copy ในแต่ละ zone (SGP, THA, USA, ...)
ข้อมูล update หมด





Google Bigtable

Google Bigtable

- A fully managed wide-column and key-value NoSQL database service by Google since 2015

เก็บ content website ที่ Google call มา

- Main focuses
  - Scalability: stores petabytes of data on thousands of nodes (machines)
  - Wide applicability: serves throughput oriented batch processing and latency sensitive services
  - High performance and high availability

Bigtable Data Model เก็บ content website

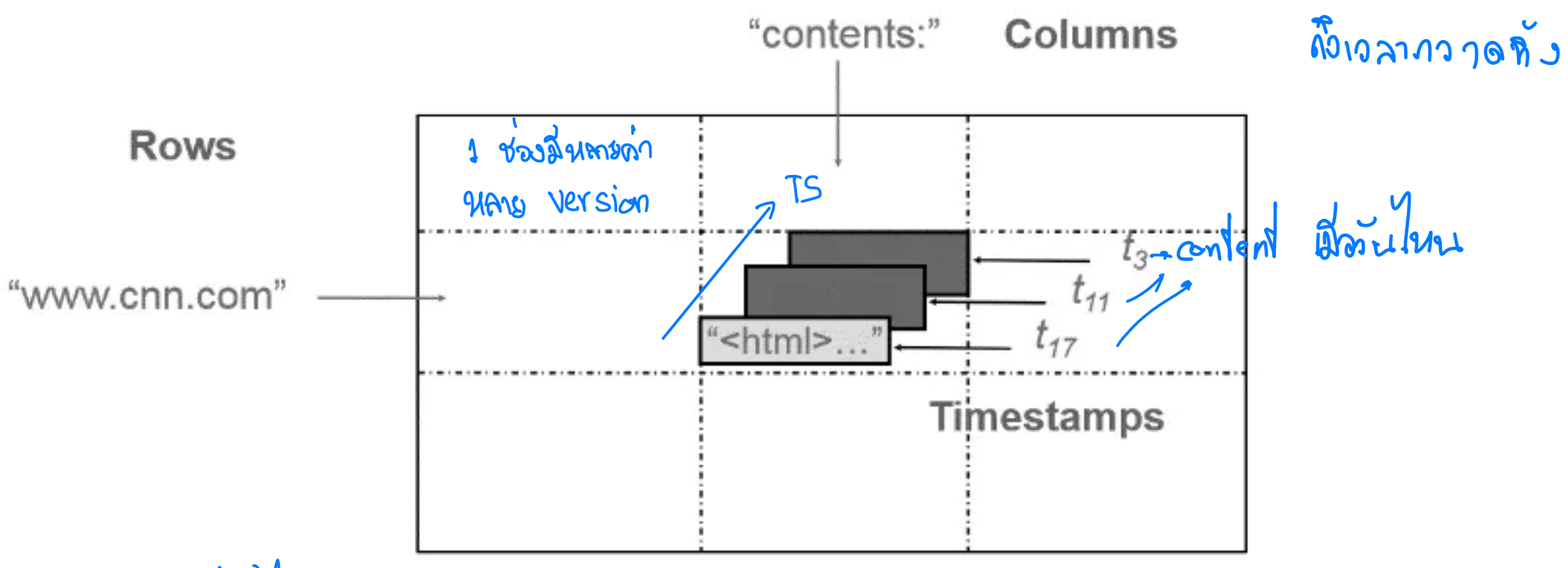
เก็บ 3 มิติ

- Data is an uninterpreted string stored in a cell indexed using row, column, and timestamp

$$(row : string, column : string, timestamp : int64) \rightarrow string$$

- Thus, a single cell can stored multiple values (limited by Garbage Collection (GC) policy)

เก็บได้กี่ version / นานแค่ไหน



เวลาเปลี่ยนไป content เปลี่ยน แม้เว็บเดิม

Bigtable Rows and Tablets

tablet คือ unit ของการ handle data ของ node ของตัว Bigtable (รวบรวม row ที่ใกล้กันเป็น set ข้อมูล)

row key ใกล้กัน

- Each table has only one index, the row key; each row key must be unique

เรียงตาม row key แล้ว row key ที่ใกล้กันรวมเป็น 1 tablet

- Rows are sorted lexicographically by row keys and rows with consecutive keys are grouped into a tablet for better system performance

ซึ่งอาจมีขนาด 100 - 200 MB

- A tablet can hold up to 100-200 MB of data and can be handled by one node at any time

- If a tablet becomes too big or has significant more loads than other tablets, it can be further partitioned into multiple tablets

สามารถแยกขนาด tablet เพิ่มได้

Group ของ row = tablet

Bigtable Columns and Column Families

- A column key is a two level naming structure, consisted of column family and optional qualifiers

$$Column\ name \rightarrow Column\ family : optional\ qualifier$$

family เสียงสัมย์ access control เหมือนกัน เช่น manage บ่ายหัน

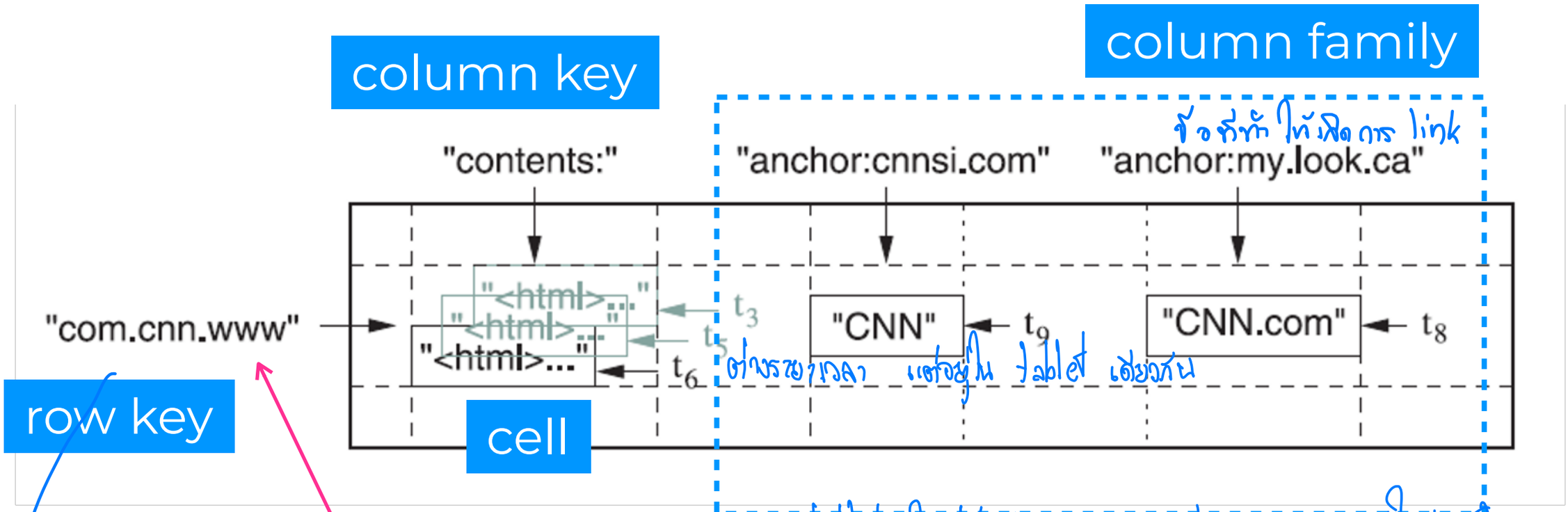
- Column family is the basic unit of access control and must be created before storing data under column key in that family

- Column families are not sorted, but columns are sorted in lexicographic order within the family

- Bigtable tables are sparse; a column does not take up space in a row that does not use the column

Example: Web Table

Bigtable ใช้ row key ค้นทีละอย่างเดียว



ช่องที่ทั้งให้เห็นการ link

ตำแหน่งเวลา เช่นอยู่ใน Tablet เดียวกัน

ถ้า Tablet ใหญ่ขึ้นเรื่อย ๆ จนจุดหนึ่งจะแยก row ที่ใหญ่ ๆ เป็นคนละ Tablet

notice a reversed hostname

reverse เพราะถ้าไม่ reverse โอกาสข้อมูลติดกัน มีมากเวลา group ข้างหน้าไปเหมือนกัน

เวลา search "com.cnn.www" ก็ใกล้กัน

Important Notes

- All operations are atomic at the row level; an operation affects the either an entire row or none of the row

- Design row key based on the queries you will use
  - Row key can consist of multiple delimited values e.g. thai#pad_thai or seafood:sashimi or thai/soup/tom_yum
  - Well-designed row key prefix can improve query performance as they can be used to retrieve data without a full table scan e.g. retrieve all Thai dishes in the table

- Put related columns with similar data retention needs in the same column family

Playing with Bigtable Emulator

- Server-side: use docker compose file from bigtable folder in datasci_architecture repo (this will use bigtable emulator image from docker hub)

```
cd bigtable

sudo docker compose up
```

- From client-side, install google bigtable python package

```
pip install google-cloud-bigtable
```

- Run a python example script, simple_client.py to create an example recipe table and demonstrate how to retrieve data from bigtable emulator

Google Bigtable Architecture

Cassandra ไม่มี service ภายนอก ของตัวเอง

API forward request ไปยัง big table node / tablet server
ที่ถูกต้อง ดูก row key ที่ consult กับ chubby service ว่า row key นี้อยู่ tablet server ตัวไหน

- Front-end servers (API)
- Tablet servers (Bigtable nodes)
  - Manage a set of tablets
  - Handle read and write requests to the tablets
  - Split tablets that have grown too large
- File Systems (Colossus)
  - Store logs and data files
  - Tablet is stored in SSTable (Sorted String Table)
  - Immutable map from keys to values
- Chubby

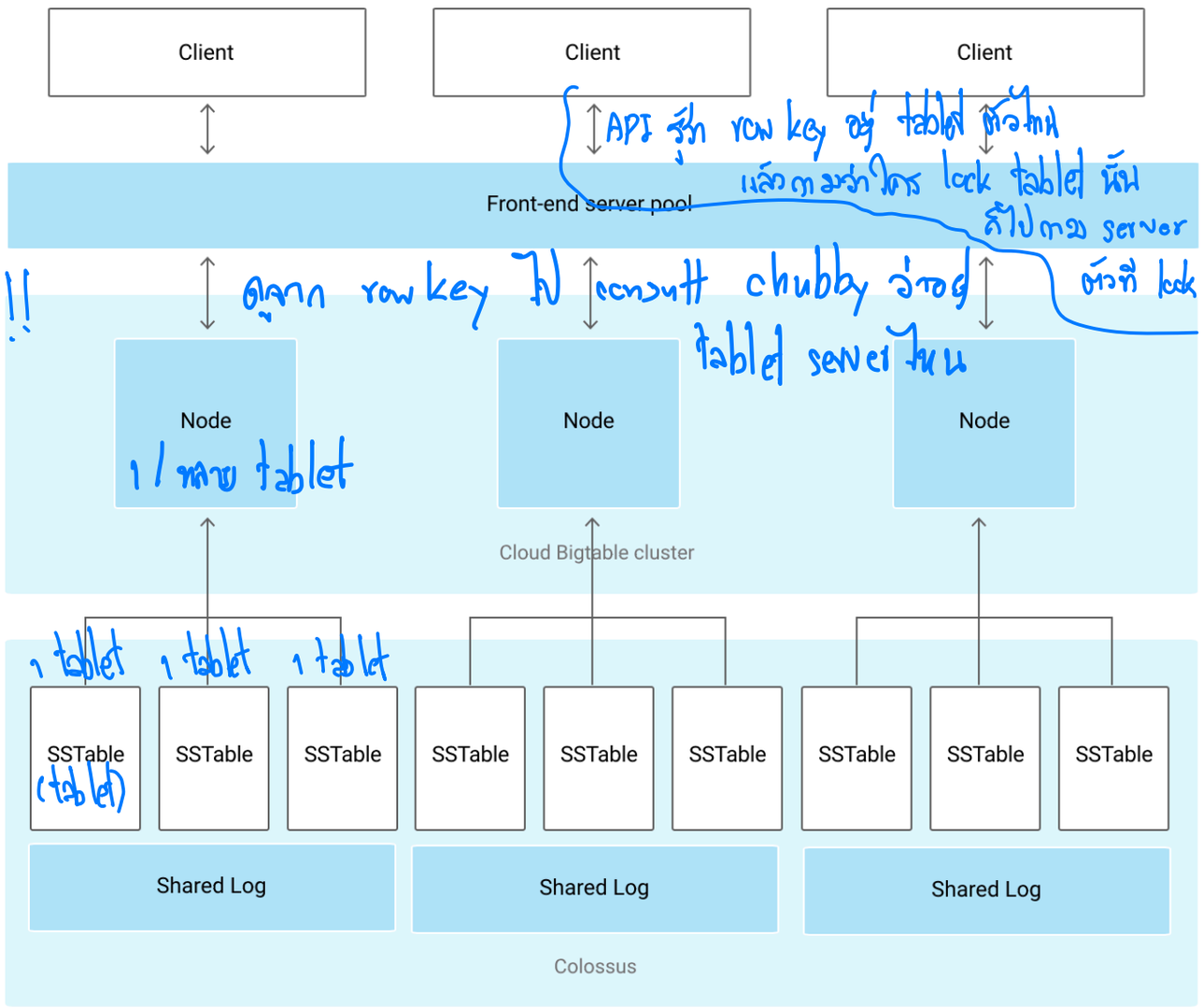
จัดการ R/W tablet ที่ Bigtable node เท่านั้น !!

HDFS / Google File System

SS Table

row ที่อยู่ติดกันอยู่ใน colossus

Colossus กระจายไปหลายเครื่อง



1 tablet ถูกรับผิดชอบโดย tablet server ตัวเดียวเท่านั้นในช่วงระยะเวลานึง
แต่ 1 tablet server รับผิดชอบได้ > 1 tablet ณ เวลาใดเวลานึง

ข้อมูลเก็บใน Colossus

1 Tablet = 1 SSTable

หน้าที่ ① เลือกตัวหัวหน้า tablet server ที่ลงทะเบียน

Chubby = Coordinator Service

Chubby

② maintain mapping ระหว่าง tablet server กับ tablet

tablet ทุกตัวต้อง register กับ Chubby และ heartbeat กับ Chubby ตลอดเวลา

- Highly-available and persistent distributed lock service with 5 replicas and one is elected as a master

คน lock ตาย ปลดล็อกอัตโนมัติ

ถ้า timeout หรือตาย จะลบ tablet server ออกจาก list ที่ตัวเองมีอยู่

- Provide a namespace of nodes, similar to Linux file systems (directories and small files)

คน lock คือ tablet server ที่ lock id

- Node can be permanent or ephemeral; Ephemeral nodes go away as no client using the node go away

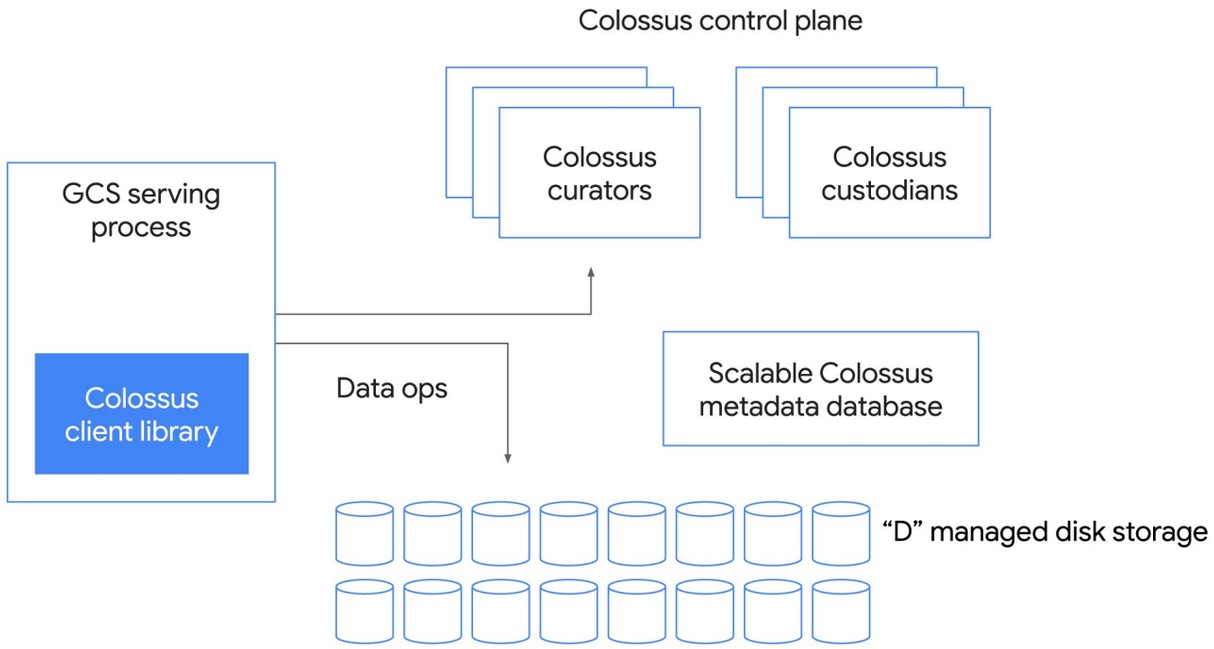
↘ lock ที่ทำคน lock ตาย lock นี้จะปลด

- See https://medium.com/coinmonks/chubby-a-centralized-lock-service-for-distributed-applications-390571273052 for more information

- Bigtable uses Chubby for checking if tablet servers are active, ensuring there is only one tablet server master, keeping schema and ACL, etc.

ทุกคนจะรู้ว่าใคร ① มีใคร lock tablet นั้นอยู่ไหน → API จะอยู่ตรงนี้ในการบอกว่าจะคุยกับใคร เพราะคนที่ lock ไว้คือ tablet server ของ tablet ตัวนั้น (tablet server lock id ของแต่ละ tablet ที่เก็บใน chubby)

Colossus



- Developed by Google to replace Google File System (GFS) as its cluster-level file system
- Clients perform software RAID by directly reading and writing data from/ to D file server to minimize latency
- Data typically written using Reed-Solomon (Error Correction Codes)
- Curators handle metadata layer similar to HDFS NameNode (but distributed)
- Custodians perform background tasks to maintain durability and availability of data as well as overall efficiency

Source: https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system

Big table scale ดีเพราะ ① สามารถยืดหยุ่นในการจัดกลุ่ม tablet ใหญ่-เล็ก ที่ row ก็ได้
② เลือกจำนวน tablet server ที่จะมาดูแล tablet ได้ (1-1, 1-m)
③ การมี Chubby และ master server ดูแลการ mapping ทำให้การจัดการมัน simple

Tablet Servers

ไปดูแล tablet ดูแลการ lock tablet

- One master server - one server is elected to be a master
  - Assigning tablets to tablet servers ตรวจให้ server ไหนรับผิดชอบ tablet
  - Detecting addition and expiration of tablet servers
  - Balancing tablet-server load
  - Garbage collection
- Many tablet servers
  - Tablet servers handle read and write requests to its table
  - Splits tablets that have grown too large

ไม่คุยกับ client เลย คุยแต่ Chubby กับ tablet server หลาย

ตรวจดูว่า Chubby ว่ามี tablet ไหนบ้างที่ไม่ถูก lock
เพื่อรู้ว่า tablet ที่ไม่มีคนรับผิดชอบจะได้ไปดูว่าจะให้ใครมา lock tablet ต่อ

Tablet Servers

- Communication
  - Clients communicate directly with tablet servers for reads and writes (not master)
  - Most clients never communicate with the master server, leaving it lightly loaded in practice

Tablets

- Large tables broken into tablets at row boundaries
  - Tablet holds contiguous range of rows: choosing good row keys to achieve locality
  - Aim for ~100MB to 200MB of data per tablet
- Each tablet server responsible for ~100 tablets
  - Fast recovery: 100 servers each pick up 1 tablet for 1 failed server
  - Fine-grained load balancing:
    - Migrate tablets away from overloaded machine
    - Master makes load-balancing decisions

Tablet Assignment

ดูแลโดย master

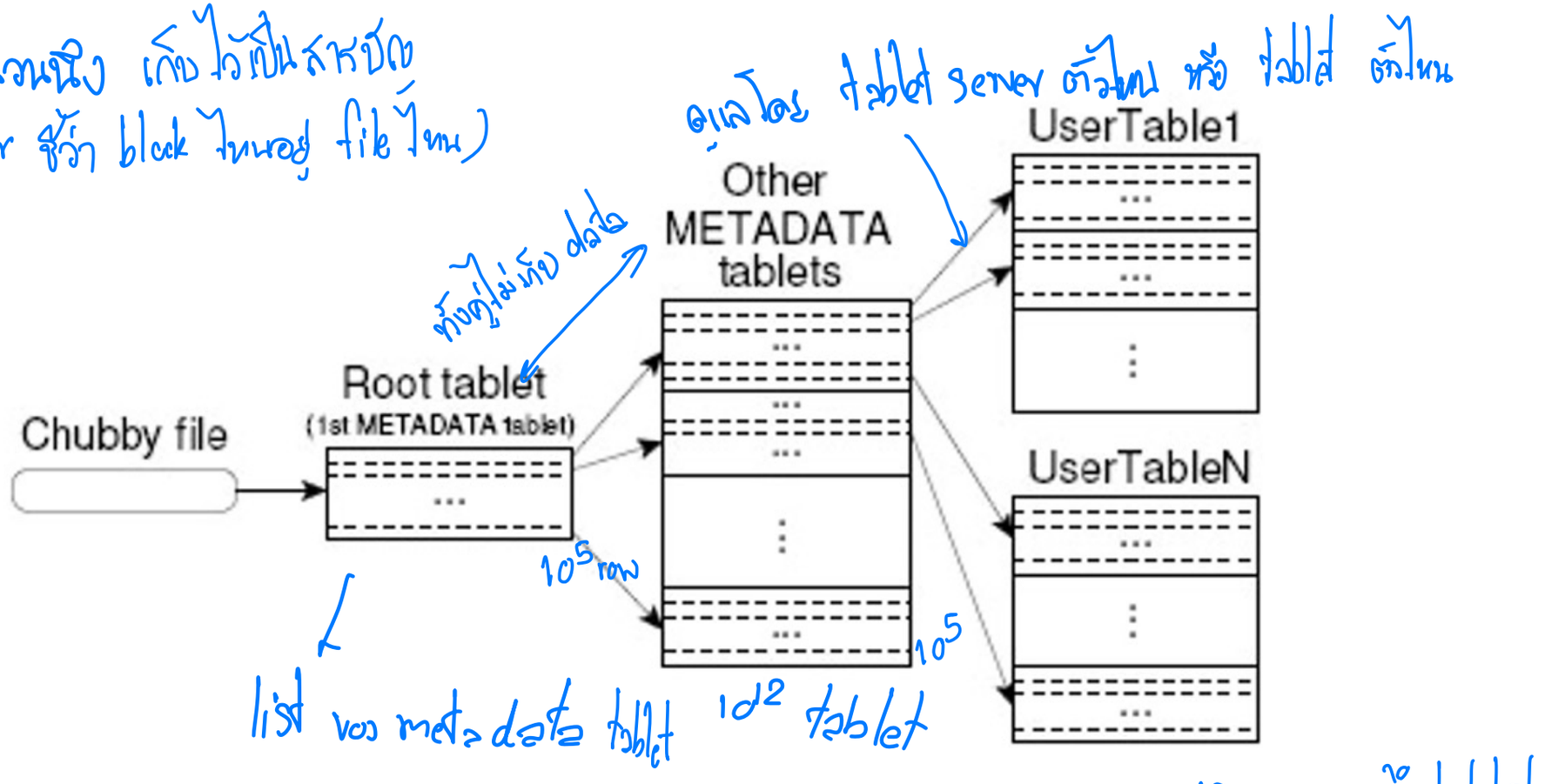
- Each tablet is assigned to one tablet server at a time
- Master server keeps track of the set of live tablet servers and current assignments of tablets to servers. Also keeps track of unassigned tablets
- When a tablet is unassigned, master assigns the tablet to an tablet server with sufficient room

API request อ่านยาก access row key นั้นจะรู้ได้ไง ?

Tablet Location - Two-Level Index Scheme

- Since tablets move around from server to server, given a row, how do clients find the right machine?
- Need to find tablet whose row range covers the target row

ขนาด tablet หนึ่งเก็บไว้เป็นสหปัจ
(File system ที่มี pointer ชี้ว่า block ไหนอยู่ file ไหน)



อาจใช้ tablet server ตัวไหน หรือ tablet ตัวไหน

ตัวนี้ไม่เก็บ data

10^5 row

list ของ metadata tablet

10^2 tablet

10^5

10^{10} row = 10^{10} tablets

บอกช่วงหยาบๆ ว่า row key อยู่ metadata ไหน

Performance Refinements

- Locality Groups - group multiple column families into a locality group for more efficient reads when accessing together

- Compression - reduce storage consumptions (10-to-1 ratio) with minimal speed losses

- Bloom Filter - reduce number of disk accesses by checking if an SSTable might contain data for a specified row/column pair

- Read assignment reading for more details

Reading Assignment

- A. Lakshman, P. Malik, “Cassandra - A Decentralized Structured Storage System”, ACM SIGOPS operating systems review, 44.2 (2010), pp. 35-40.

- F. Chang et al, “Bigtable: A Distributed Storage System for Structured Data”, ACM Transactions on Computer Systems (TOCS), 26.2 (2008), pp.1-26.

References

- Simplilearn, “Apache Cassandra Architecture From The Ground-Up”, https://www.simplilearn.com/tutorials/big-data-tutorial/cassandra-architecture

- J. Carpenter and E. Hewitt, Cassandra: The Definitive Guide: Distributed Data at Web Scale, 3rd Edition, 2022

- Google, “Bigtable overview”, https://cloud.google.com/bigtable/docs/overview

-