Big Data Storage

Big Data Management

12-09-2025

photo: https://www.absolutdata.com/blog/five-things-to-know-about-big-data-storage/

Important notice

- ✦ Random fraud check will be done **same day** with the exam
- ✦ From 13.30-15.30 in **Zoom**
- ✦ We will share a link later

Previously …

- AI and Big Data
- 3 V's of Big Data
  - Volume
  - Velocity
  - Variety
- Need for data mining/analysis and their challenges

Previously …

- AI and Big Data
- 3 V's of Big Data
  - Volume
  - Velocity
  - Variety
- Need for data mining/analysis and their challenges

Disclaimer: We will mostly talk about the **volume** aspect

Preview of today's lecture

- Why do we **store** data in **multiple machines** and what happens in this case?
- What are the **challenges** we face and how we **solve** them?

Why use multiple machines?

Why use multiple machines?

- Scalability
  - Ability to cope with **increased load**

Why use multiple machines?

- Scalability
  - Ability to cope with **increased load**

Be aware of the difference with **elasticity**!

Why use multiple machines?

- Scalability
  - Ability to cope with **increased load**
- Fault-tolerance/high-availability
  - When a machine **fails**, we want the system to still be available

Be aware of the difference with **elasticity**!

Why use multiple machines?

- Scalability
  - Ability to cope with **increased load**
- Fault-tolerance/high-availability
  - When a machine **fails**, we want the system to still be available
- Latency
  - Speed up **performance** (especially when having many requests)

Be aware of the difference with **elasticity**!

Why use multiple machines?

- Scalability
  - Ability to cope with **increased load**
- Fault-tolerance/high-availability
  - When a machine **fails**, we want the system to still be available
- Latency
  - Speed up **performance** (especially when having many requests)

Be aware of the difference with **elasticity**!

Quantifying load

- Load parameters
  - requests/sec to a web server
  - ratio of reads to writes in a database
  - # simultaneous users in a chat room
- Different statistical values can be used
  - mean
  - median
  - max

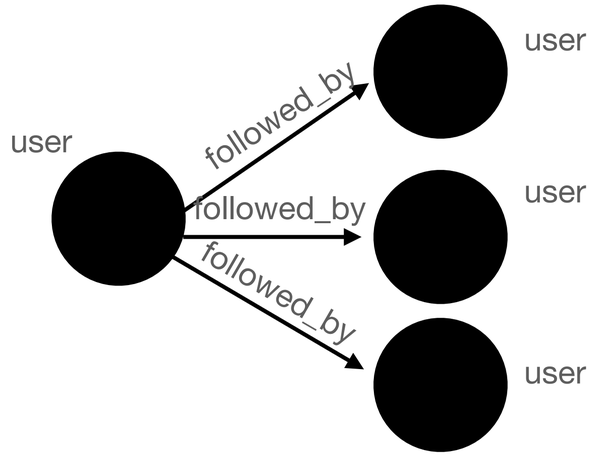
Example from Twitter (X)

- Post tweet (4.6k req/sec on average, over 12k req/sec at peak)
- Home timeline (300k req/sec)

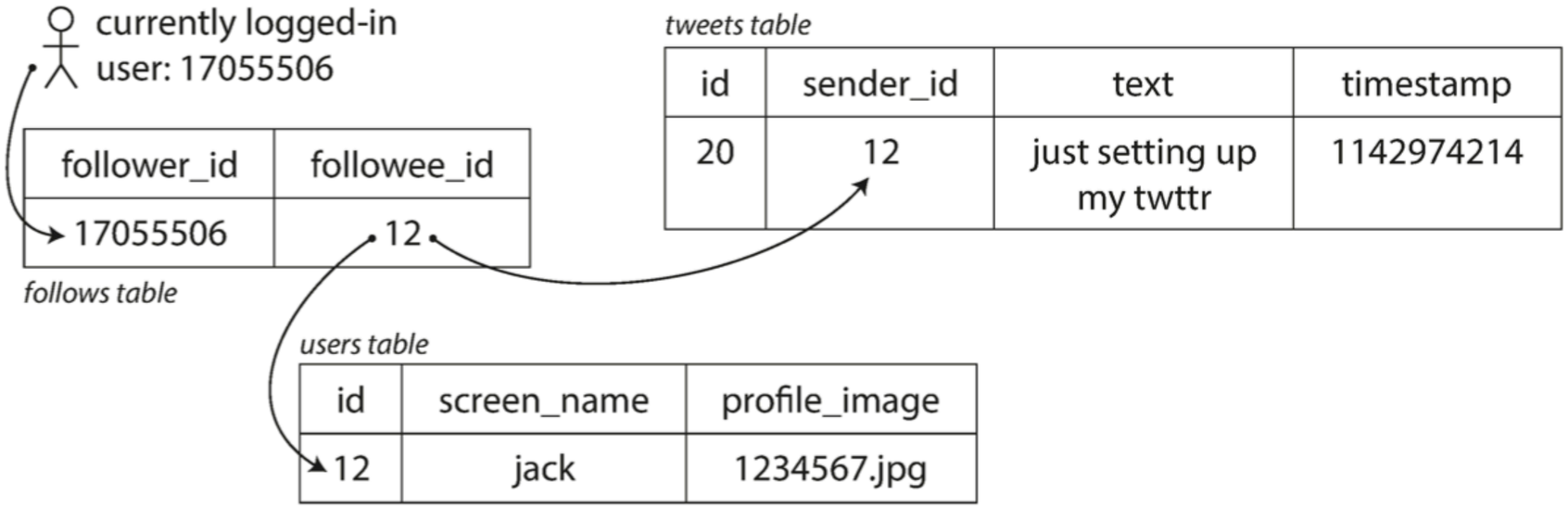
Example from Twitter (X)

- Post tweet (4.6k req/sec on average, over 12k req/sec at peak)
- Home timeline (300k req/sec)

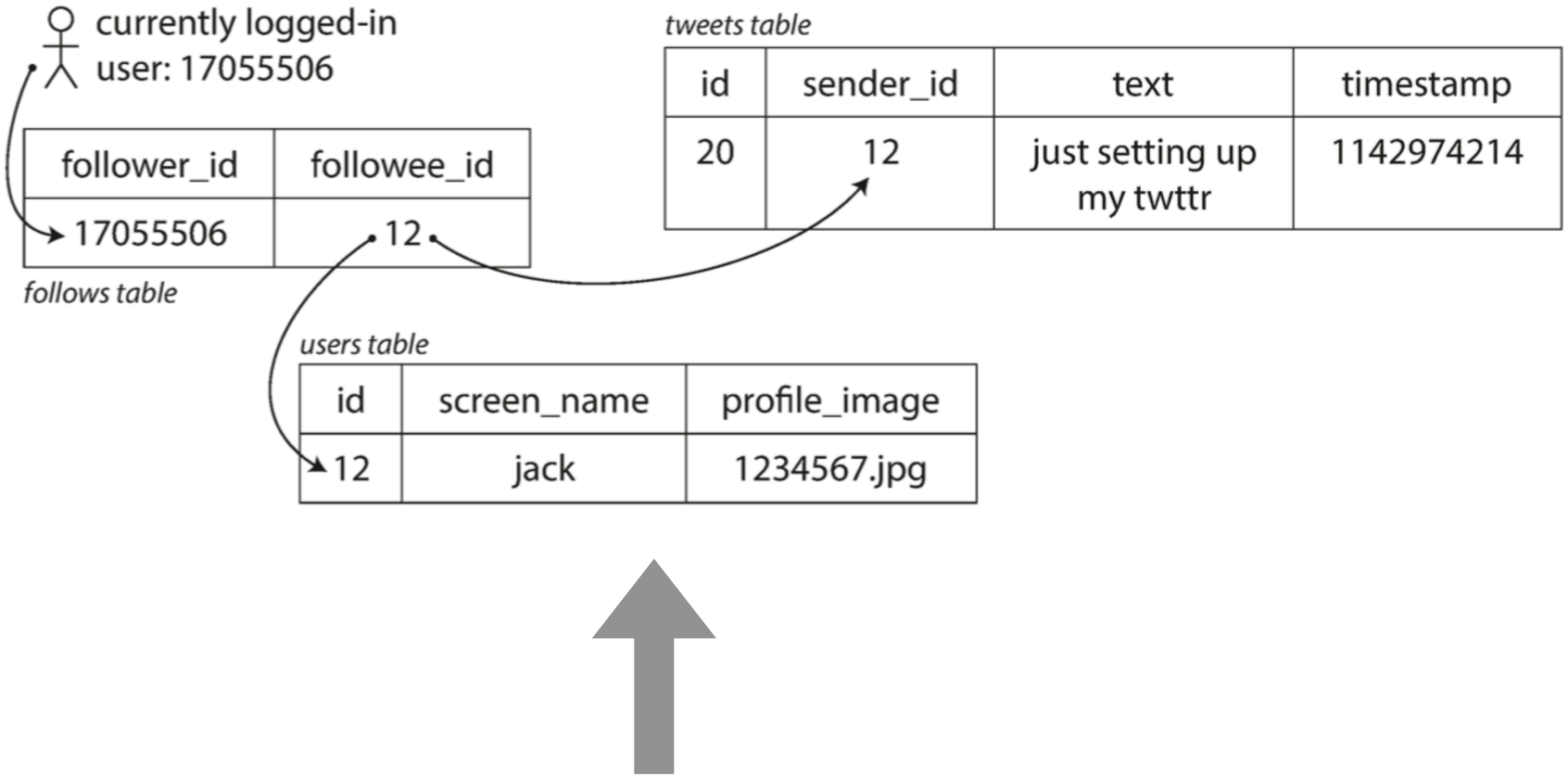
Challenge: not volume, but **fan-out**



Approach 1

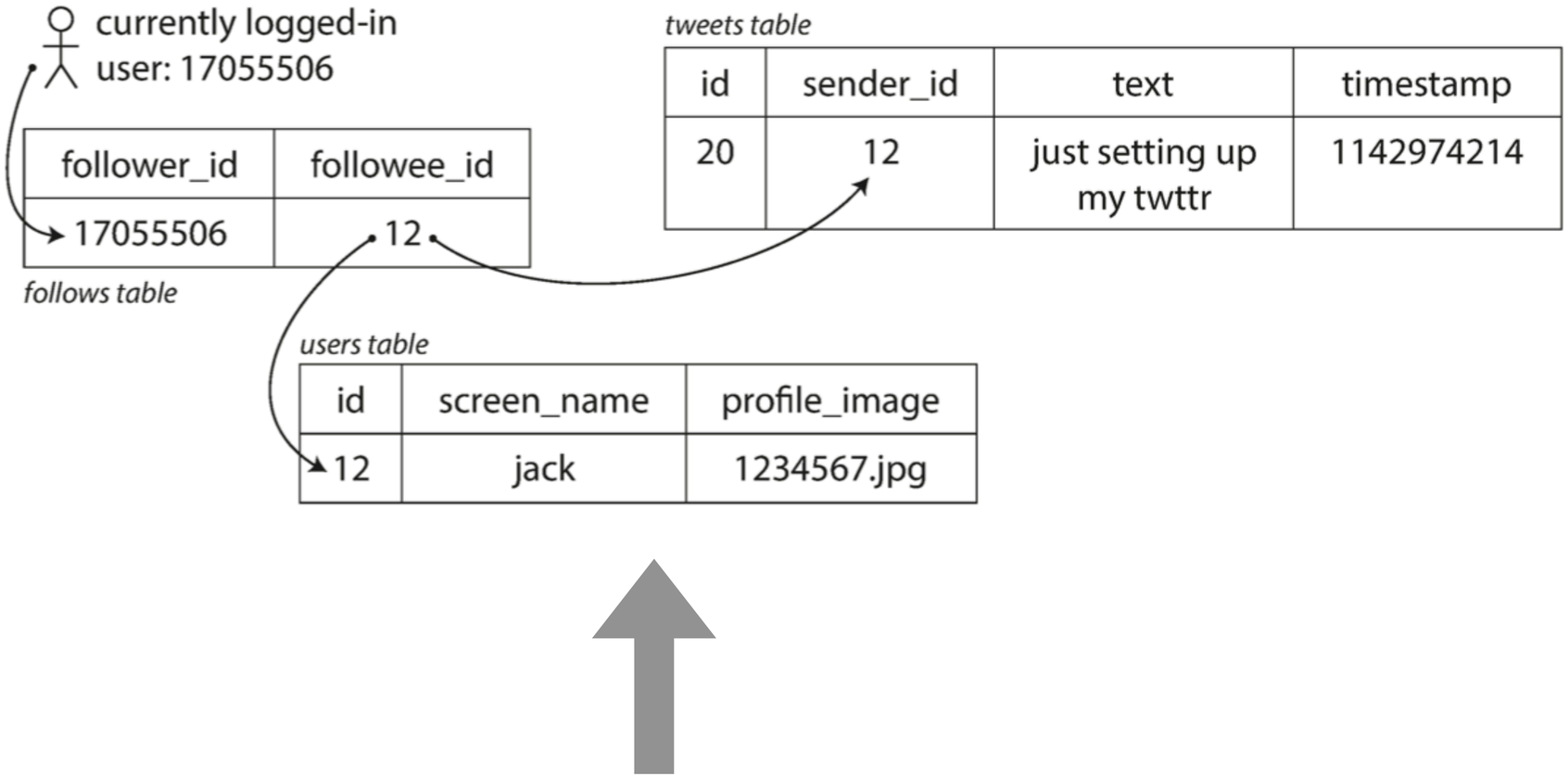

currently logged-in
user: 17055506
follower_id
followee_id
17055506
12
follows table
tweets table
id
sender_id
text
timestamp
20
12
just setting up my twttr
1142974214
users table
id
screen_name
profile_image
12
jack
1234567.jpg

Approach 1




```
SELECT tweets.*, users.* FROM tweets
  JOIN users   ON tweets.sender_id    = users.id
  JOIN follows ON follows.followee_id = users.id
  WHERE follows.follower_id = current_user
```

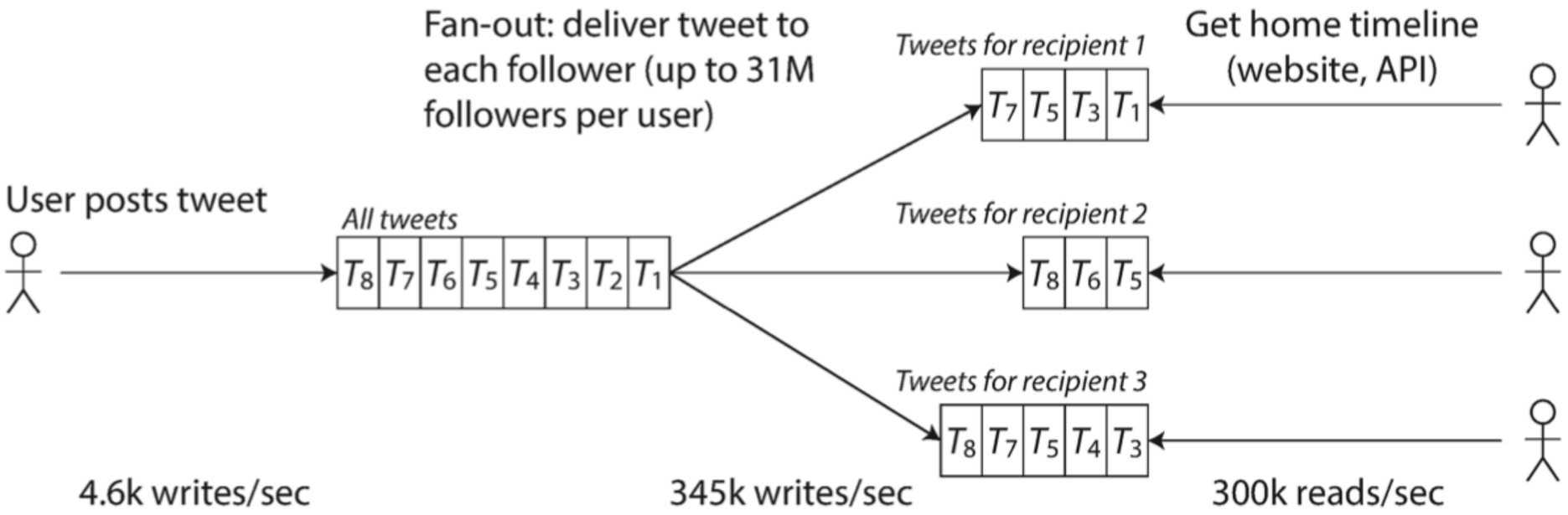
Approach 1



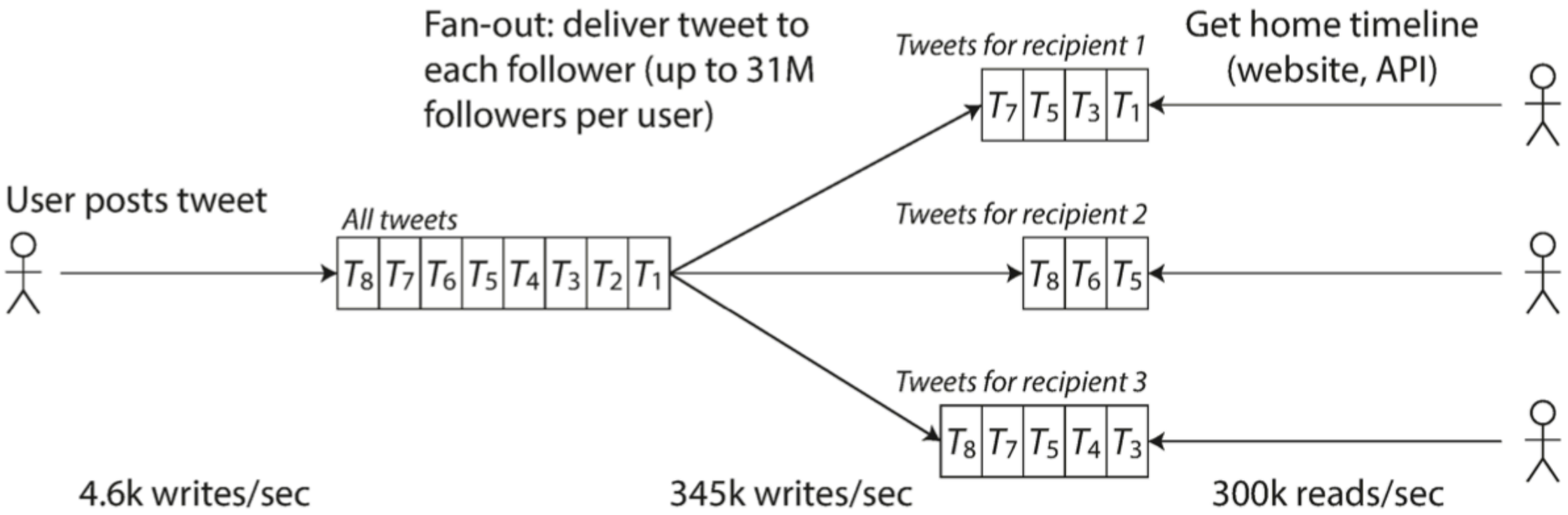

```
SELECT tweets.*, users.* FROM tweets
  JOIN users   ON tweets.sender_id    = users.id
  JOIN follows ON follows.followee_id = users.id
  WHERE follows.follower_id = current_user
```

Difficult to keep up with the load of home timeline queries!

Approach 2


Fan-out: deliver tweet to each follower (up to 31M followers per user)
Tweets for recipient 1
Get home timeline (website, API)
User posts tweet
All tweets
Tweets for recipient 2
Tweets for recipient 3
4.6k writes/sec
345k writes/sec
300k reads/sec

Approach 2

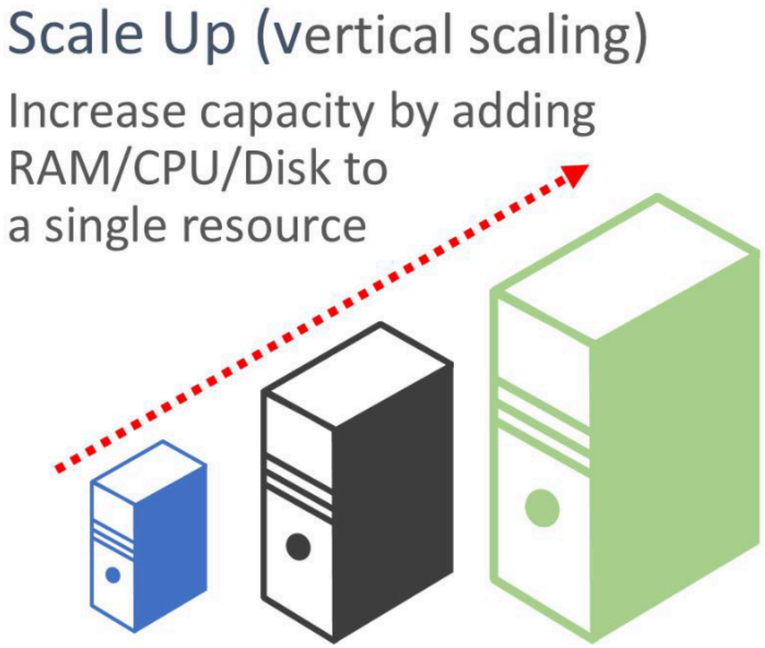


Difficult for users that are **hotspots**: have a very large fan-out degree!

Coping with load

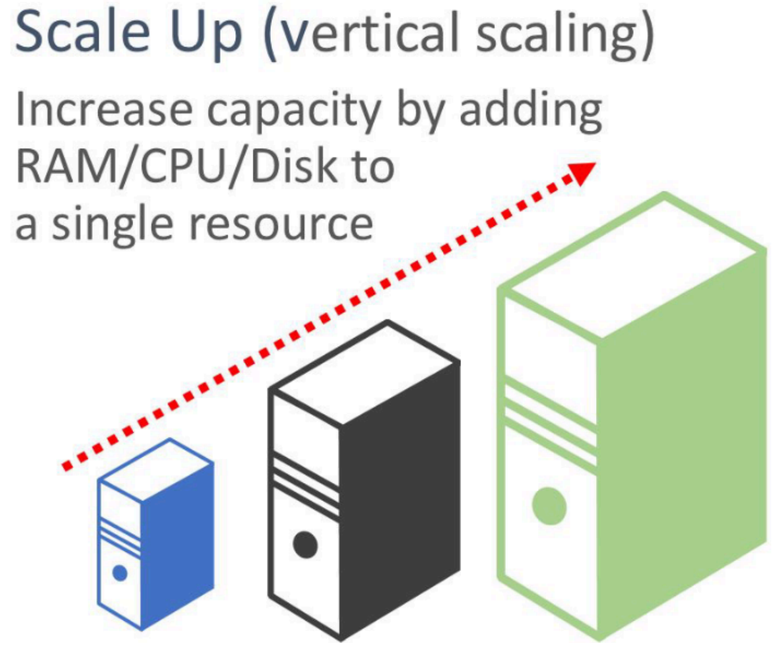
Coping with load

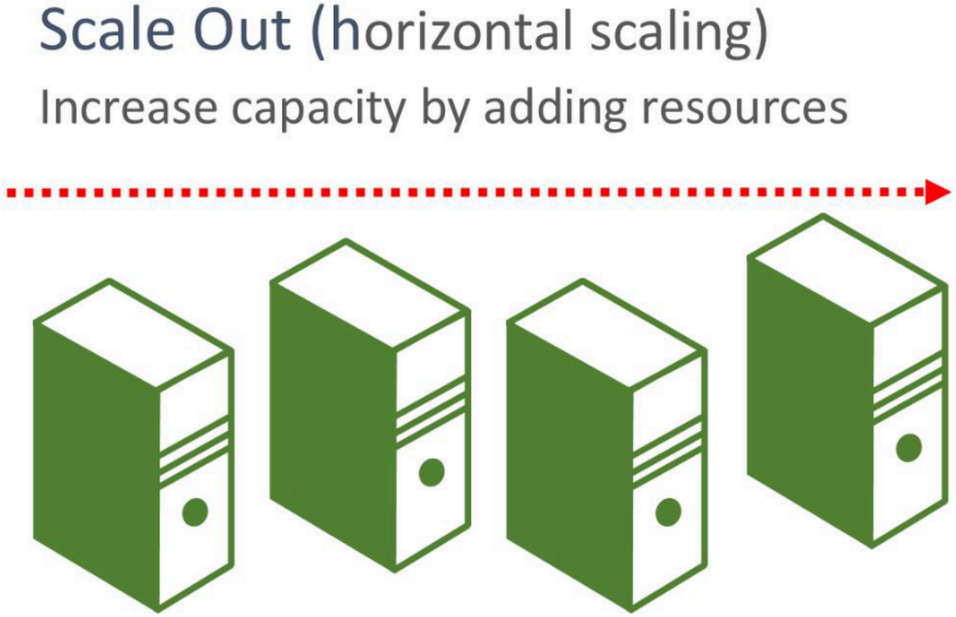
- **Scale up** a.k.a. **vertical scaling**
  - moving to a more powerful machine
  - simplest solution
  - cost grows faster than linearly



Coping with load

- **Scale up** a.k.a. **vertical scaling**
  - moving to a more powerful machine
  - simplest solution
  - cost grows faster than linearly
- **Scale out** a.k.a. **horizontal scaling**
  - adding more (cheap) machines

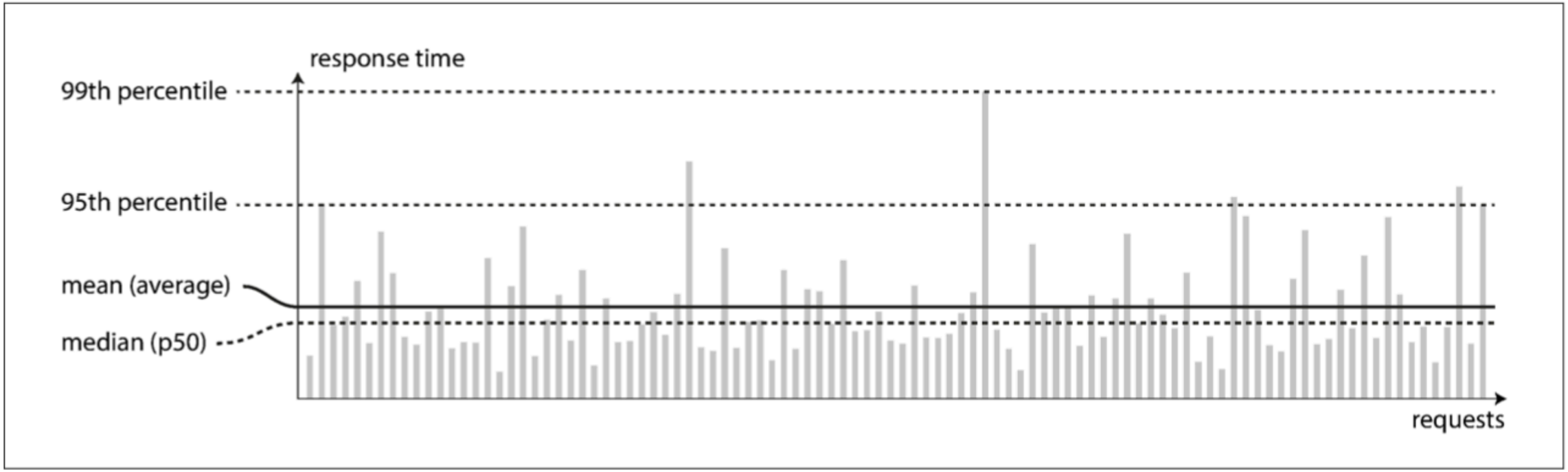




Quantifying performance

- **Throughput**
  - how many units of information a system can process in a given amount of time
  - number of records/data points/queries we can process per second
- **Response time**
  - time between user/client sending request and receiving response
- **Latency**
  - time to process a request

How to measure latency/response time/runtime?



Why use multiple machines?

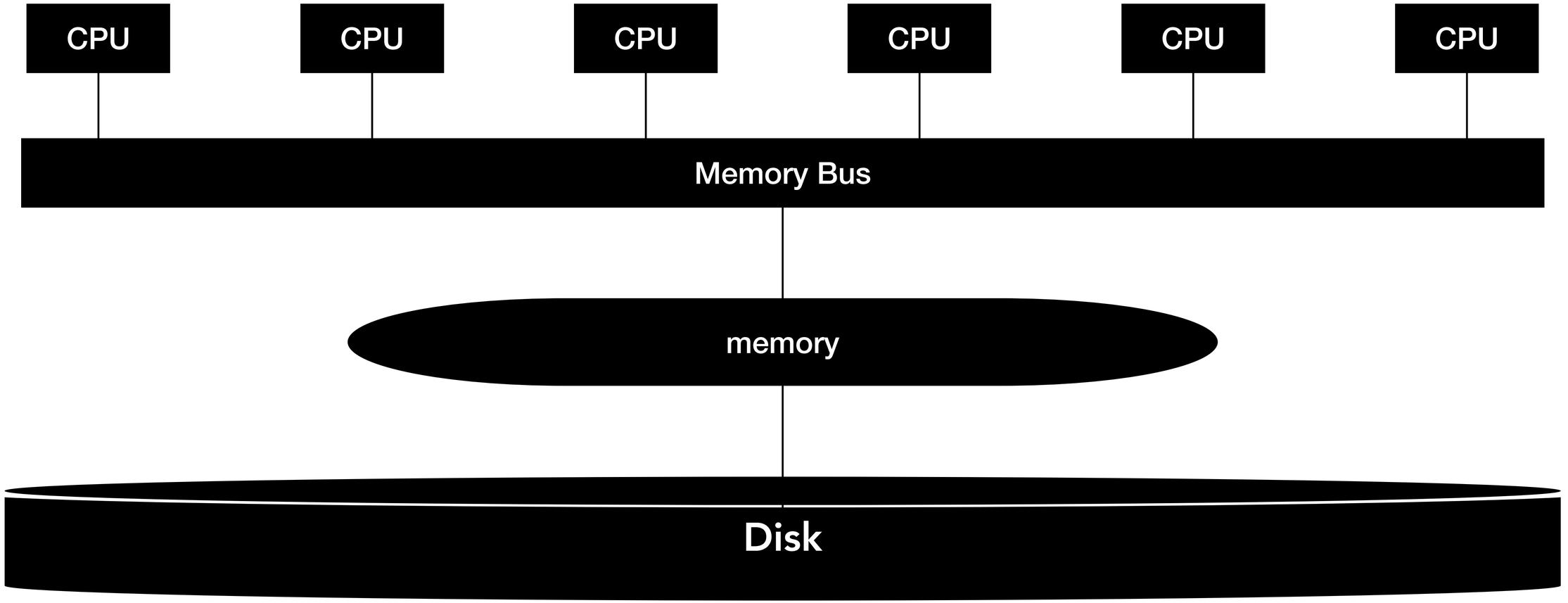
- Scalability
  - Ability to cope with **increased load**
- Fault-tolerance/high-availability
  - When a machine **fails**, we want the system to still be available
- Latency
  - Speed up **performance** (especially when having many requests)

Be aware of the difference with **elasticity**!

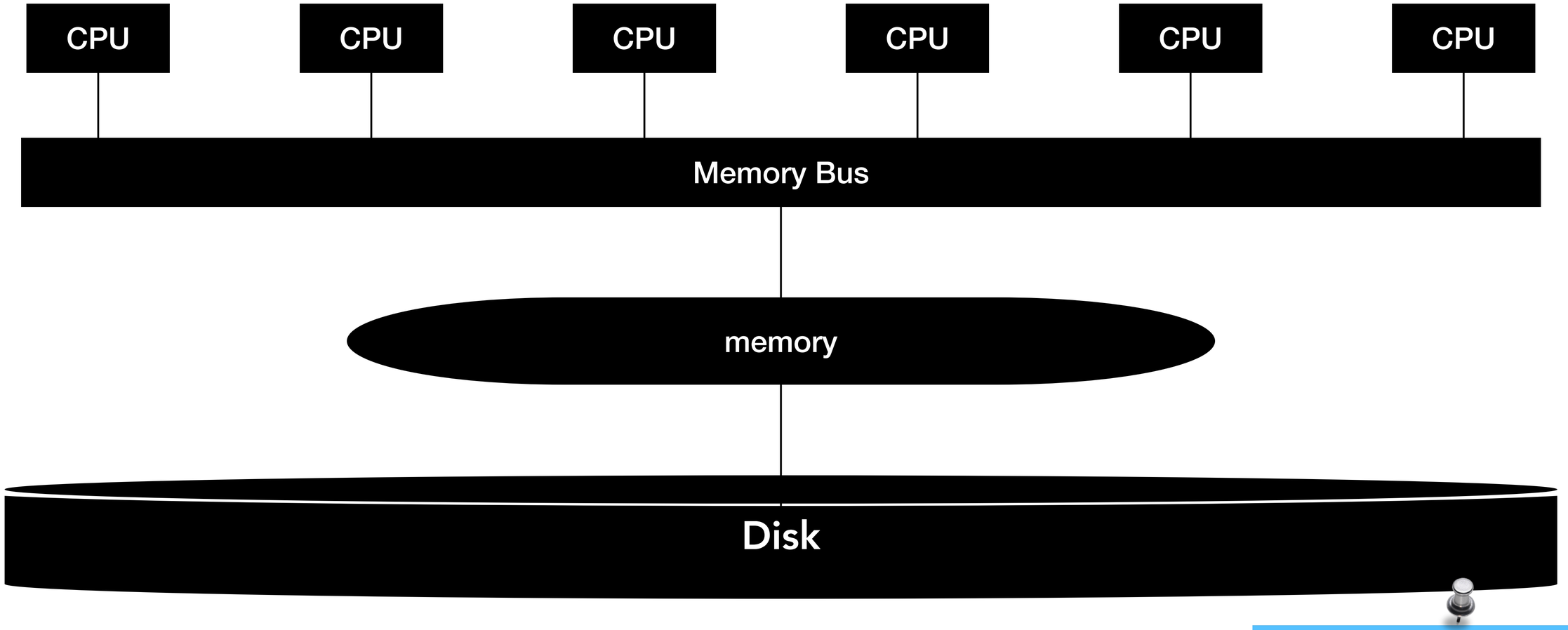
Distributed systems architectures

- Shared-memory
- Shared-disk
- Shared-nothing

Distributed system: Shared-memory architecture

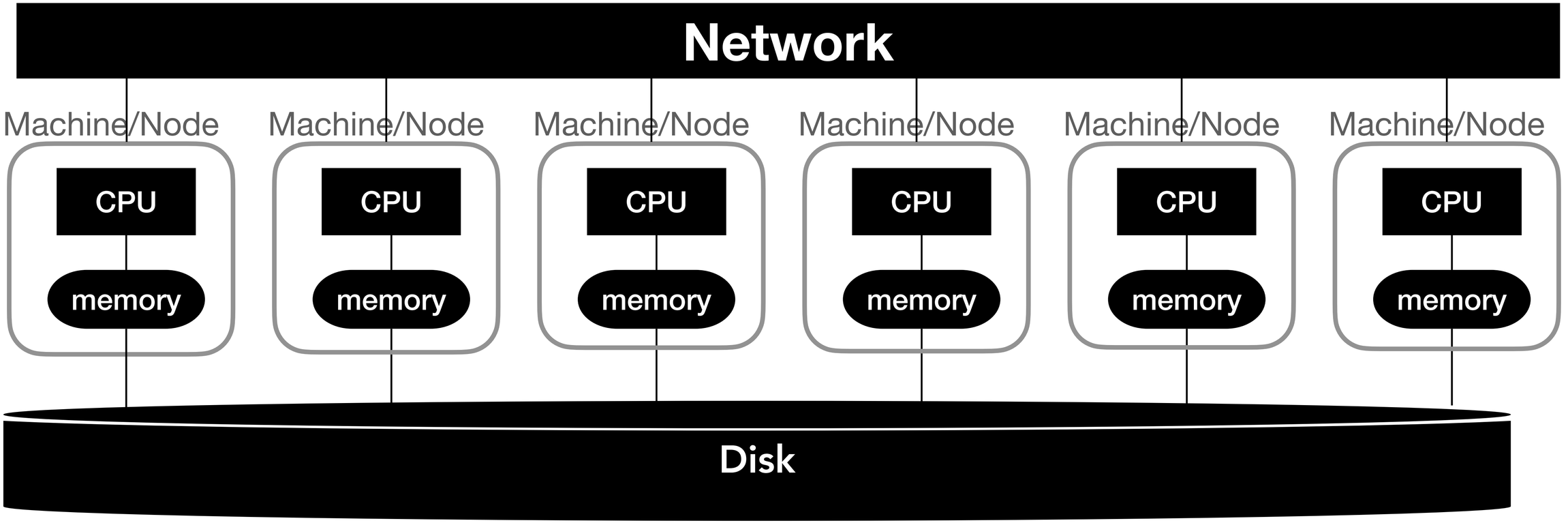


Distributed system: Shared-memory architecture

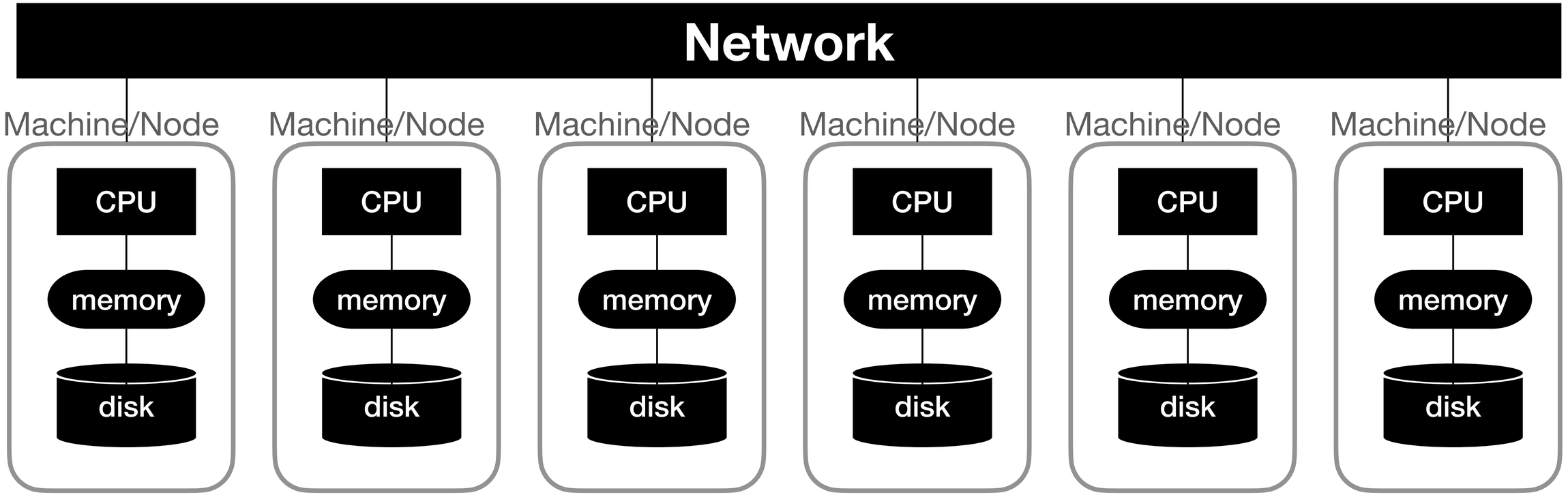


Technically, single machine!

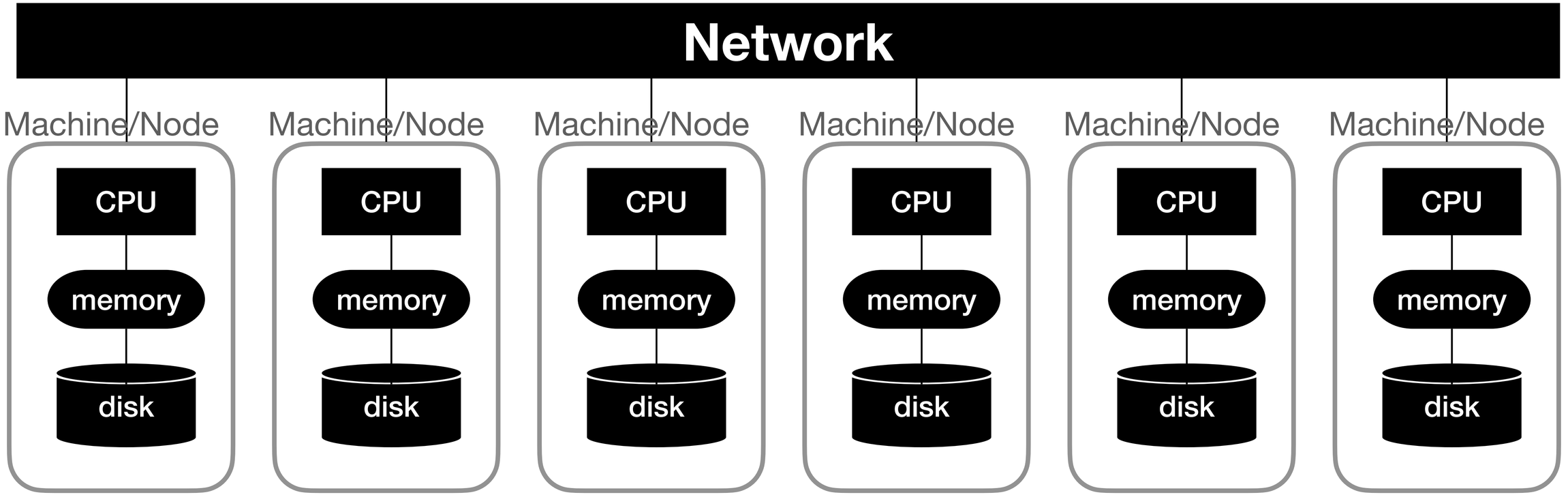
Distributed system: Shared-disk architecture



Distributed system: Shared-nothing architecture



Distributed system: Shared-nothing architecture



Most common architecture for big data systems

Distributed systems infrastructure deployments

Distributed systems infrastructure deployments

- **On premises** 
  - **Cluster computing**: homogeneous commodity machines in a shared-nothing architecture connected via a local network, typically for large scale analysis

Distributed systems infrastructure deployments

- **On premises** 
  - **Cluster computing**: homogeneous commodity machines in a shared-nothing architecture connected via a local network, typically for large scale analysis
- **In the cloud** 
  - **Cloud computing**: machines connected via IP network, typically in geographically different locations, Internet services
    - Private/Public/Hybrid

Distributed systems infrastructure deployments

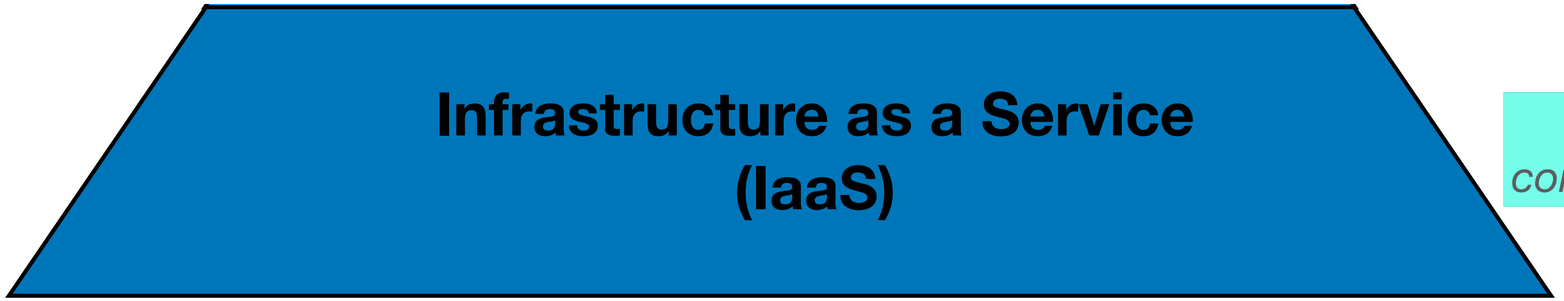
- **On premises** 
  - **Cluster computing**: homogeneous commodity machines in a shared-nothing architecture connected via a local network, typically for large scale analysis
- **In the cloud** 
  - **Cloud computing**: machines connected via IP network, typically in geographically different locations, Internet services
    - Private/Public/Hybrid
- **High-performance computing** (HPC): interconnected supercomputers with multiple CPUs/GPUs in a disk-shared architecture, usually for computationally intensive scientific tasks

Distributed systems infrastructure deployments

- **On premises** 
  - **Cluster computing**: homogeneous commodity machines in a shared-nothing architecture connected via a local network, typically for large scale analysis
- **In the cloud** 
  - **Cloud computing**: machines connected via IP network, typically in geographically different locations, Internet services
    - Private/Public/Hybrid
- **High-performance computing** (HPC): interconnected supercomputers with multiple CPUs/GPUs in a disk-shared architecture, usually for computationally intensive scientific tasks

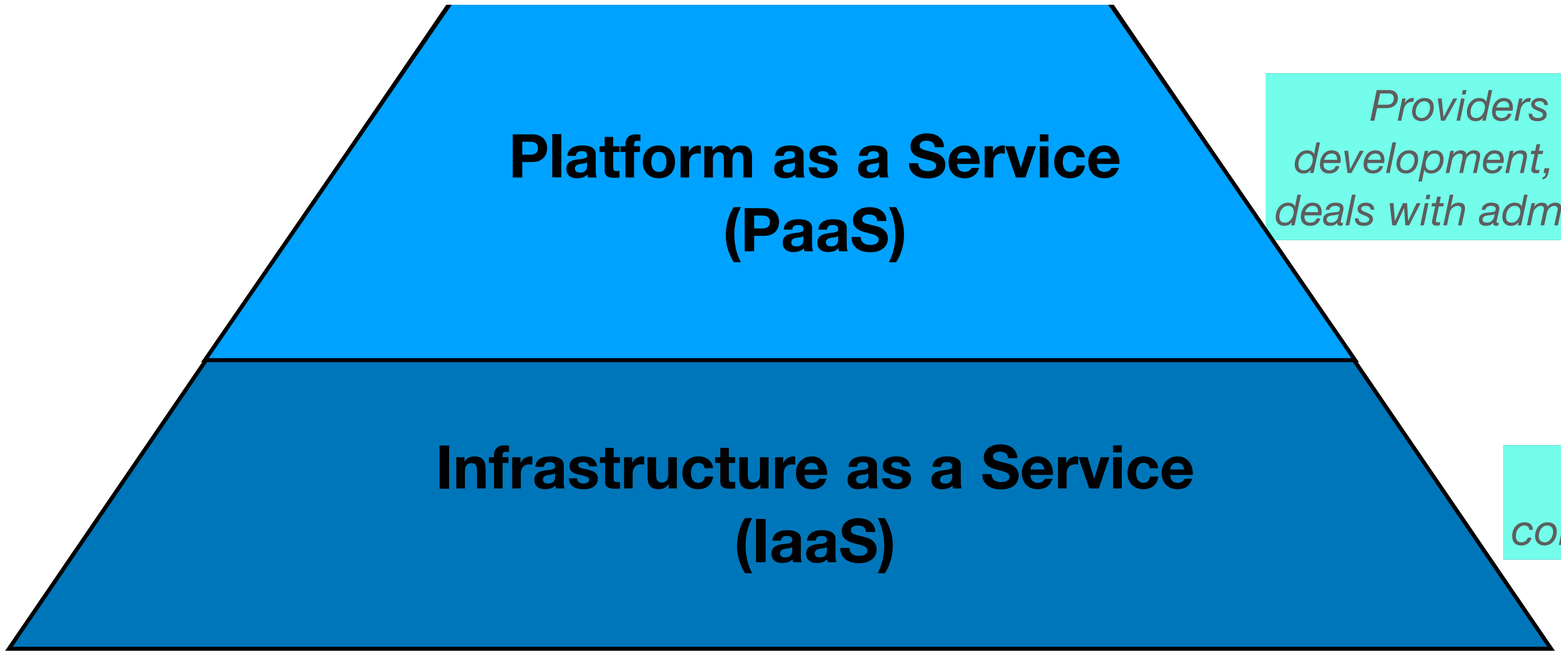
Other terms have appeared throughout the years: grid computing

Cloud computing service models



Providers rent out storage and computing capacities on their servers

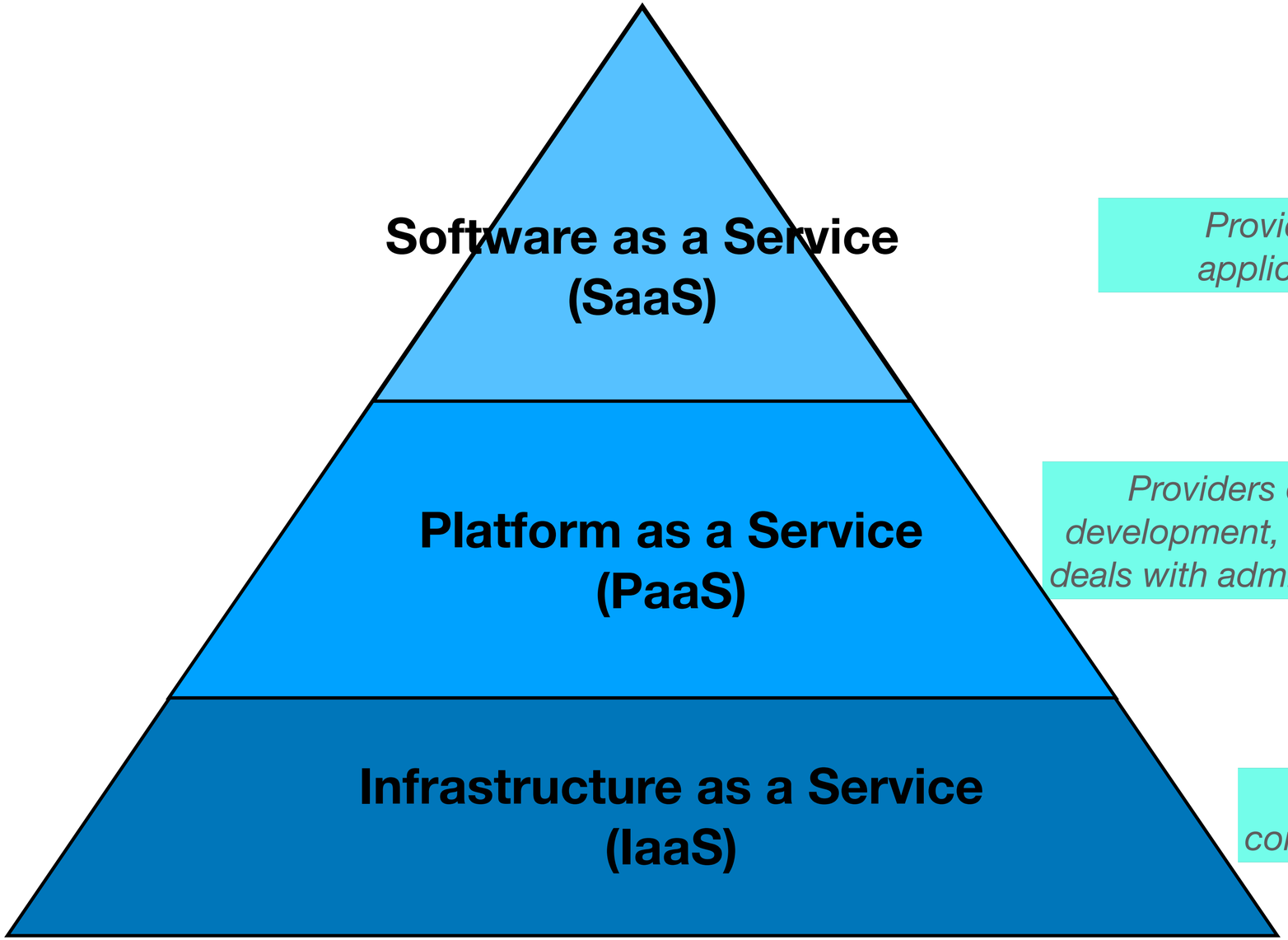
Cloud computing service models



Providers can rent platforms for software development, e.g., Windows Azure. The provider deals with administration of the underlying servers.

Providers rent out storage and computing capacities on their servers

Cloud computing service models

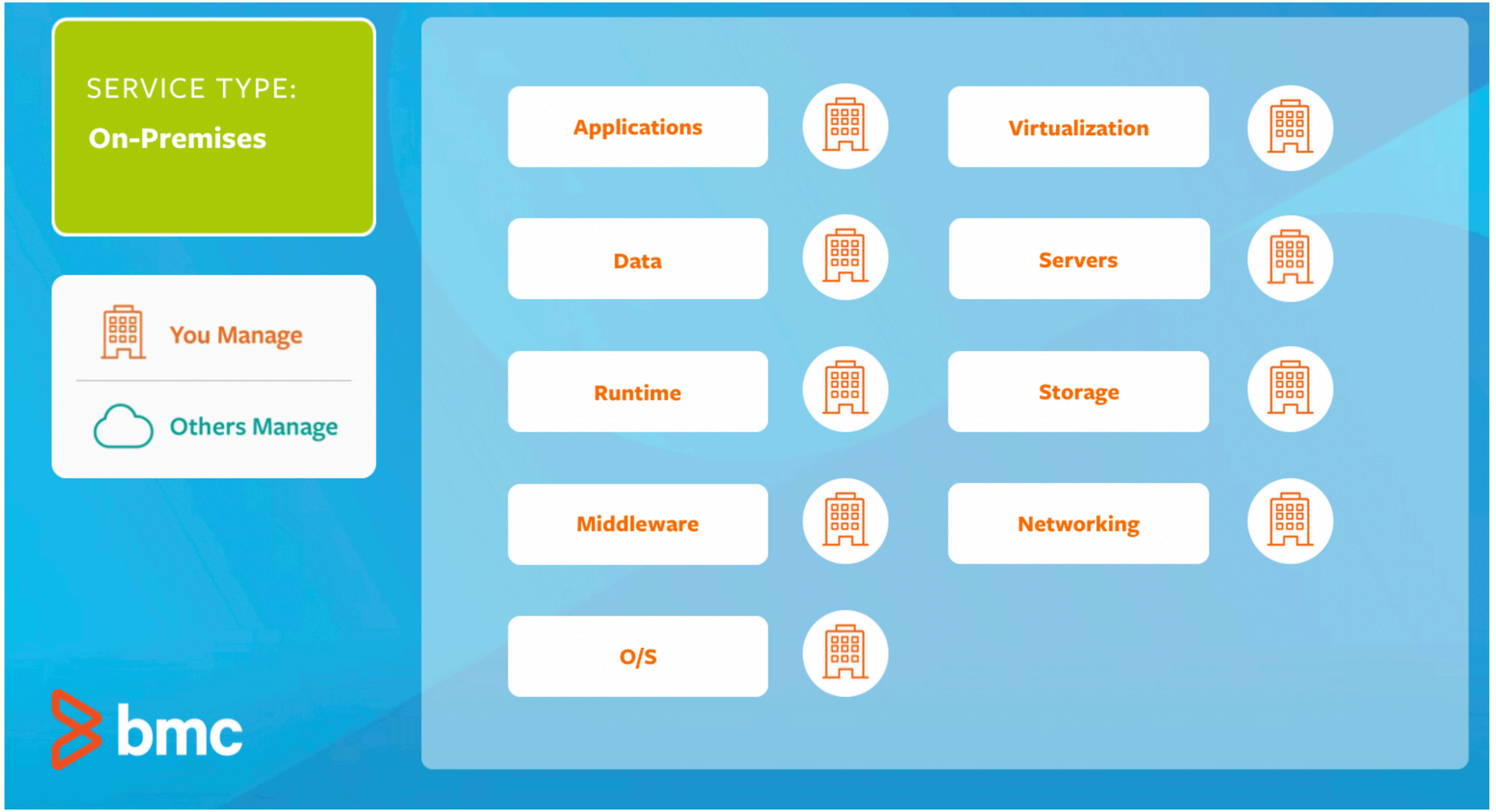


Providers can rent already built applications, e.g., Google docs.

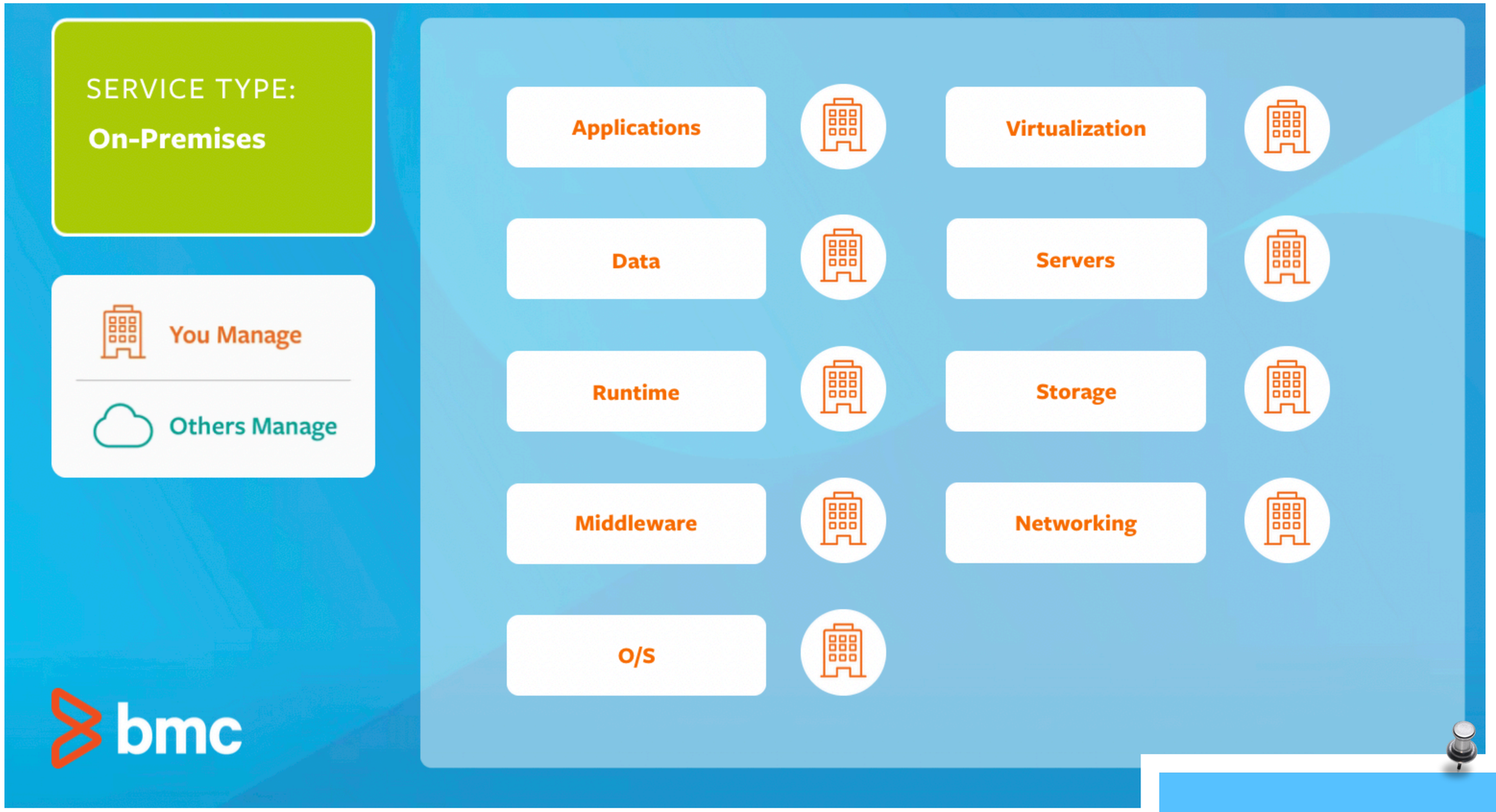
Providers can rent platforms for software development, e.g., Windows Azure. The provider deals with administration of the underlying servers.

Providers rent out storage and computing capacities on their servers

Benefits of cloud computing

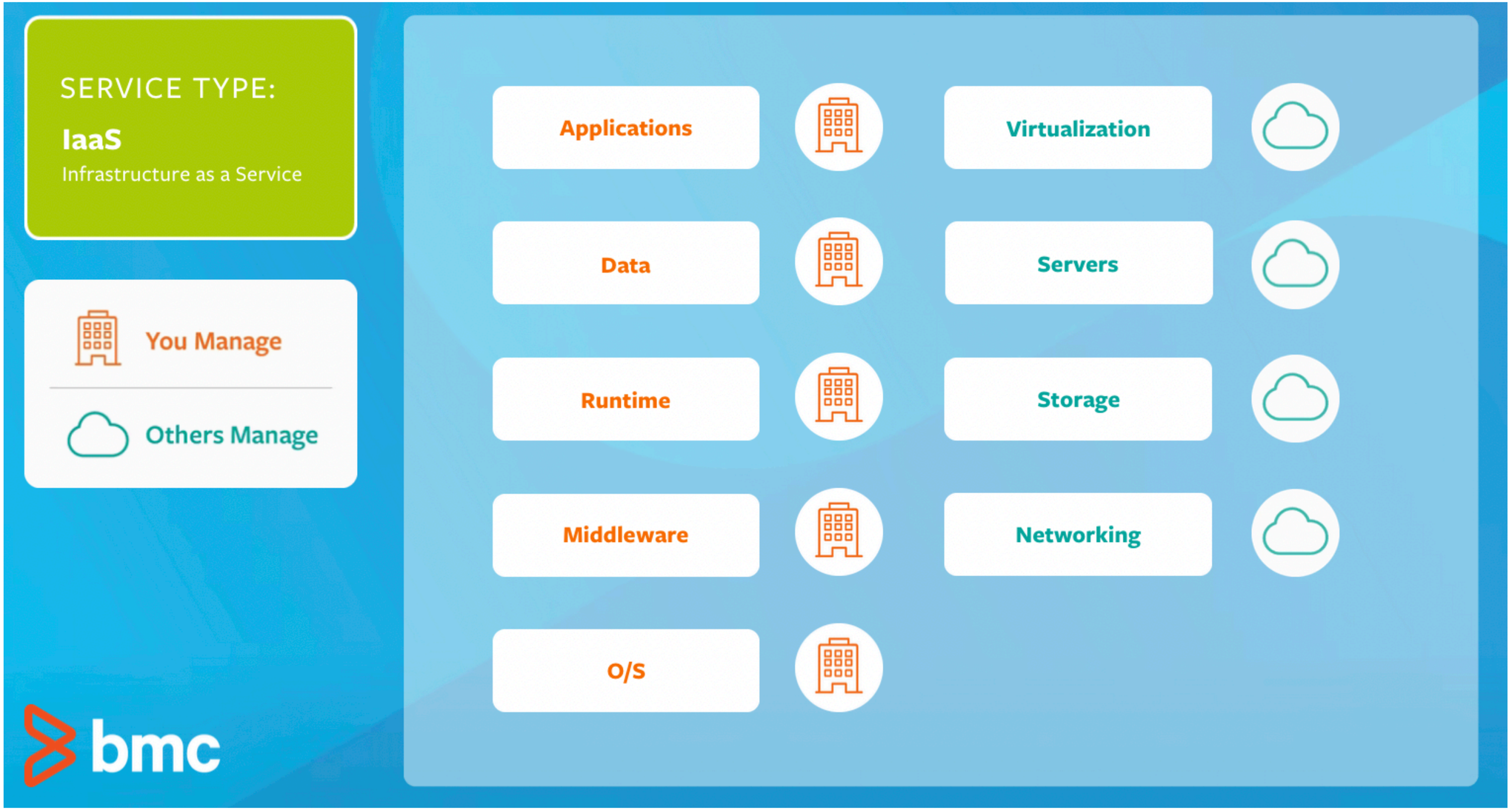


Benefits of cloud computing

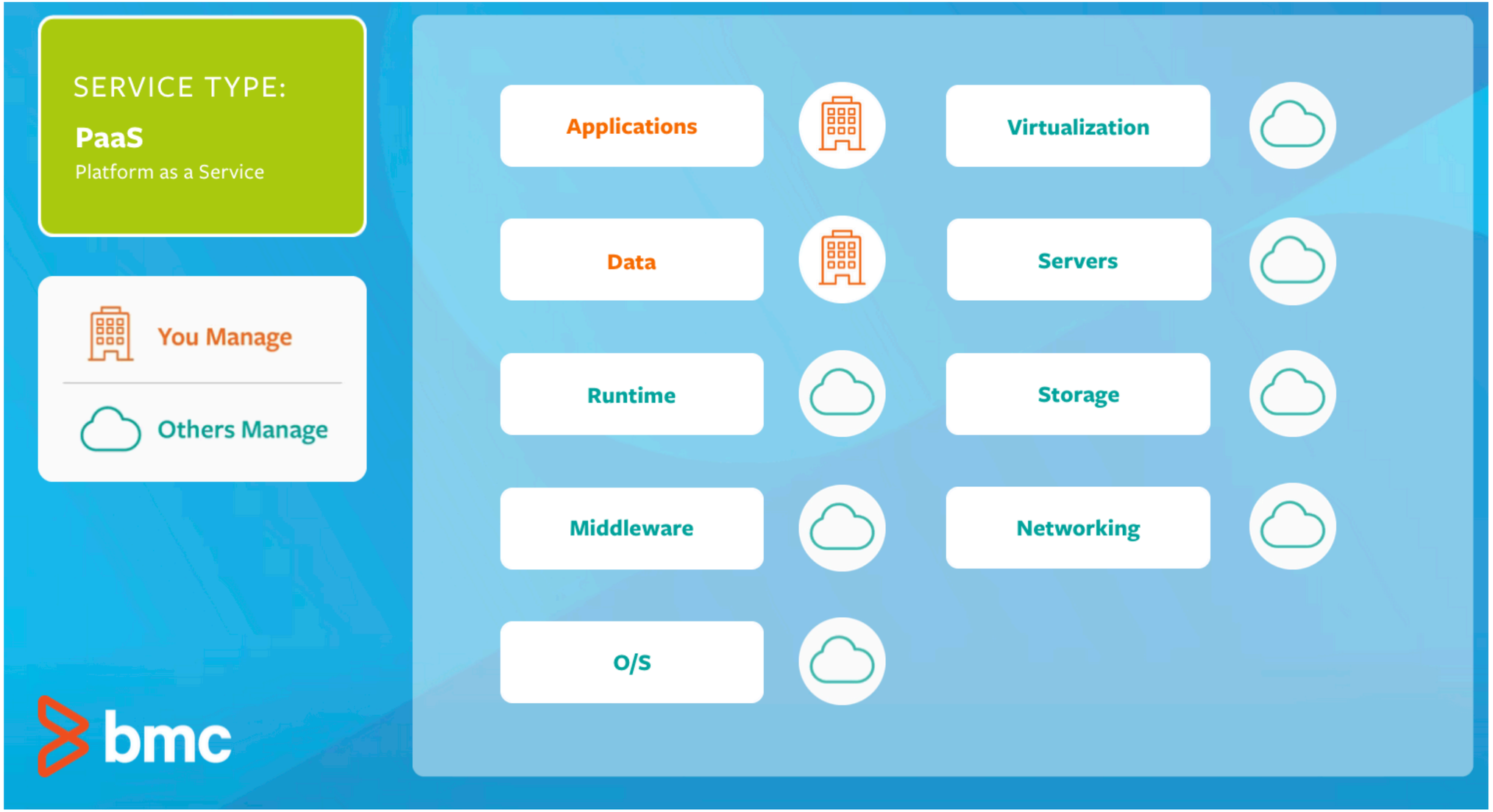


Everything should be managed by the company/organization

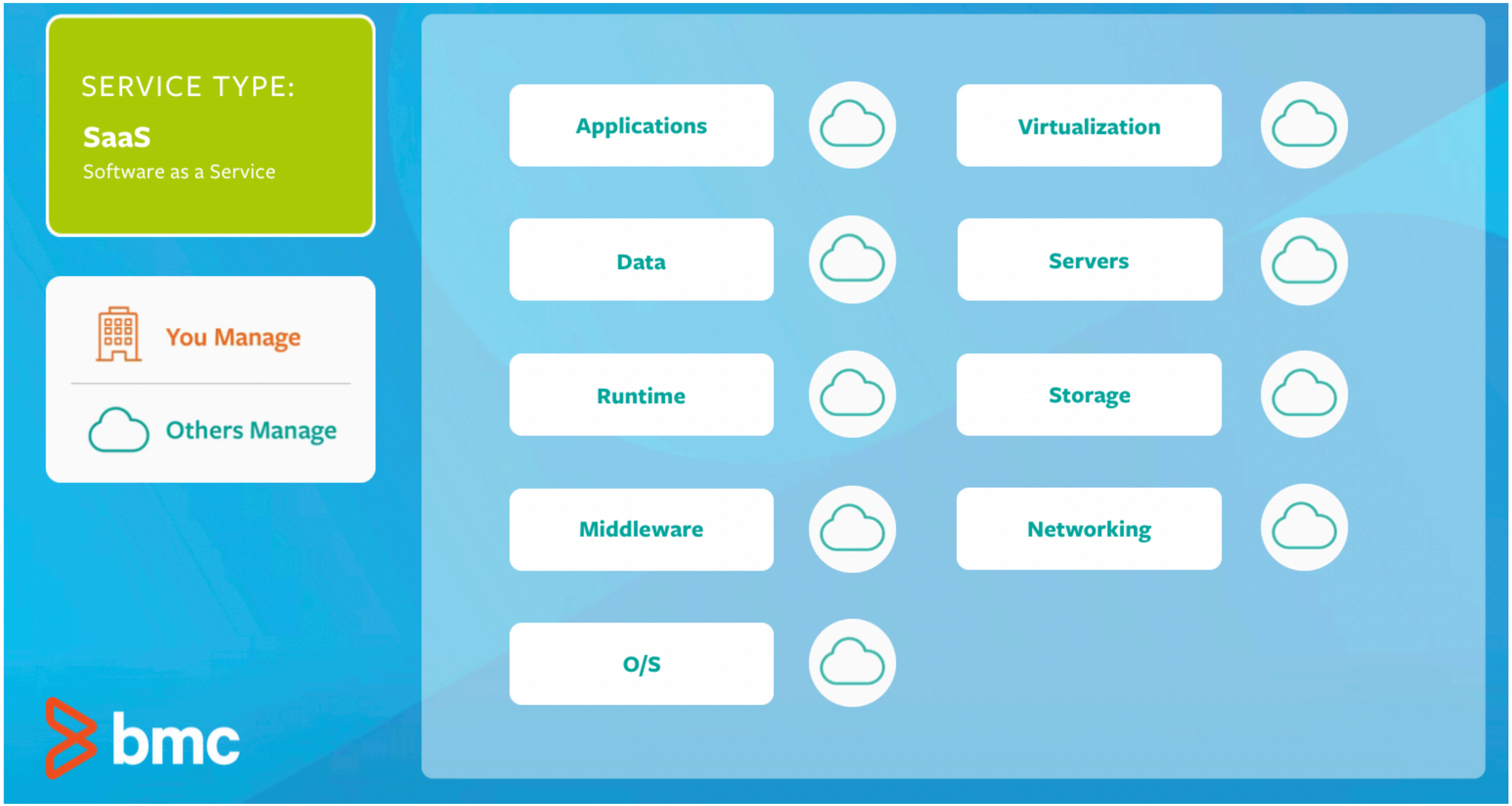
Infrastructure as a Service (IaaS)



Platform as a Service (PaaS)

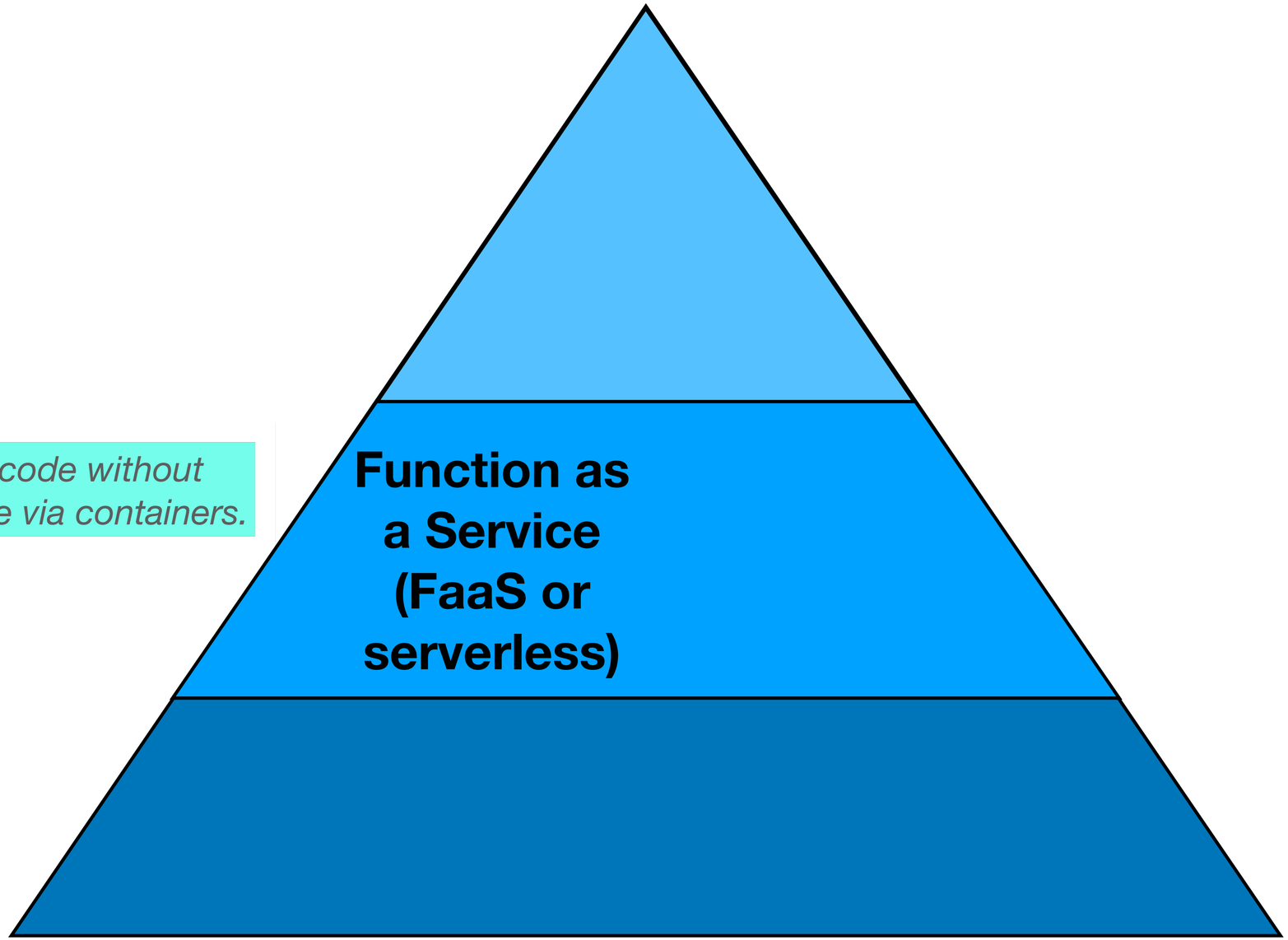


Software as a Service (SaaS)

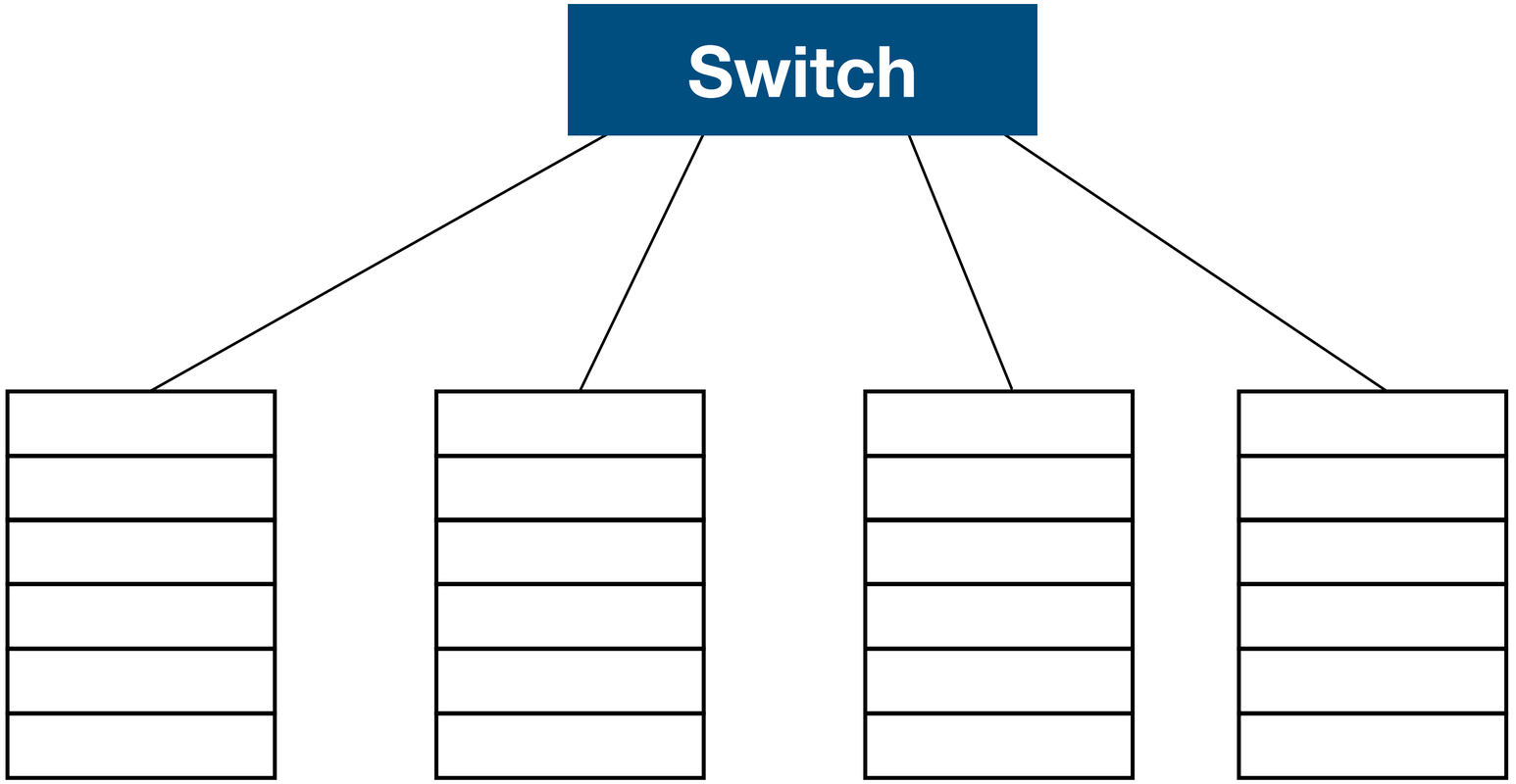


Cloud computing service models

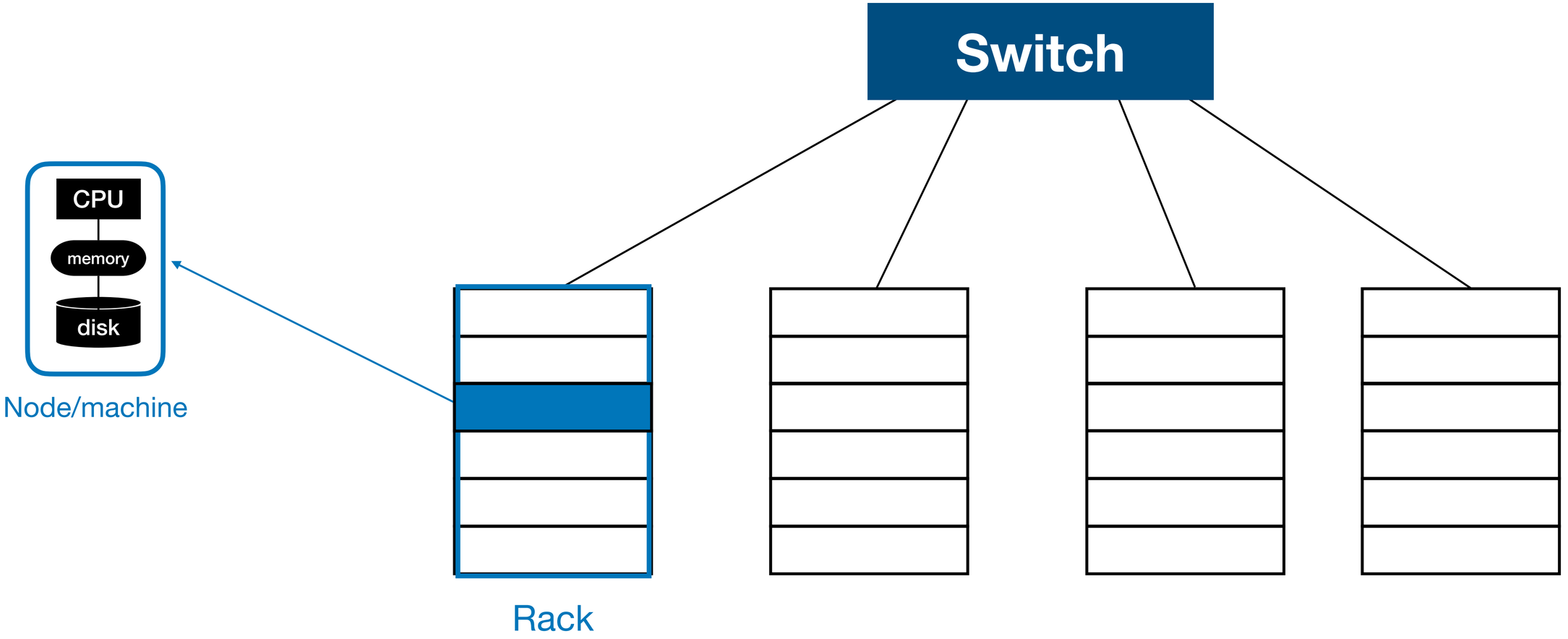
Consumers can deploy and run code without having to manage any infrastructure via containers.



Cluster architecture

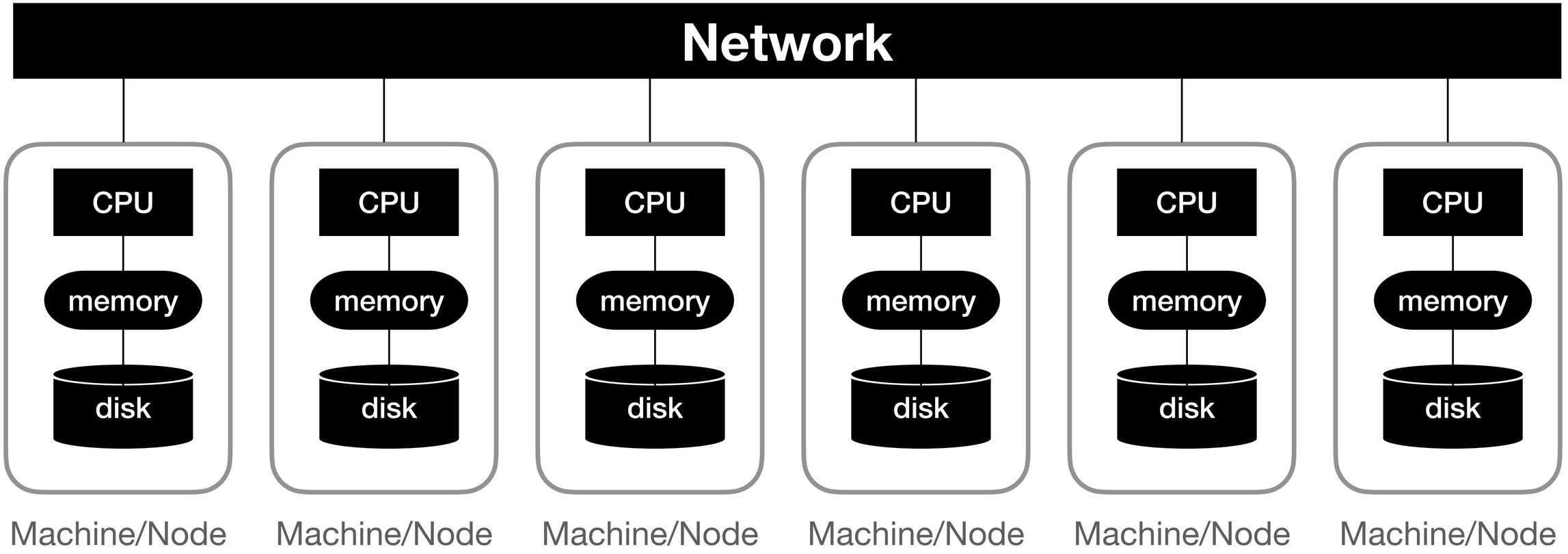


Cluster architecture

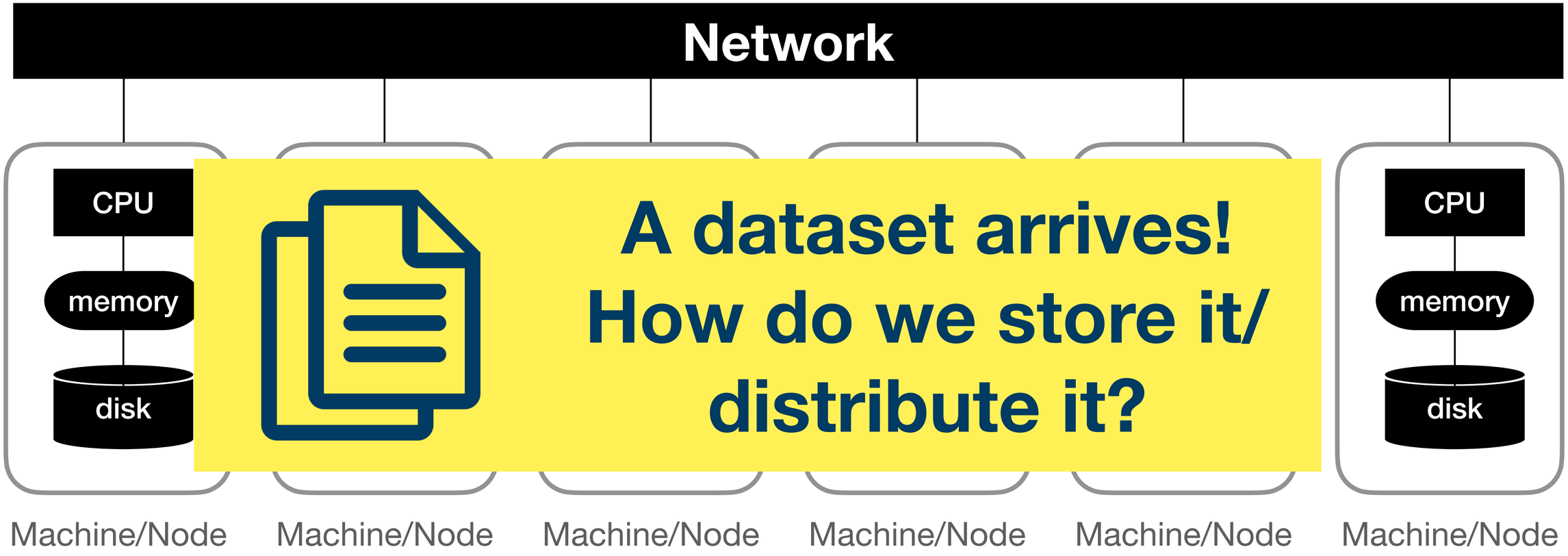


A typical cluster configuration in a data center

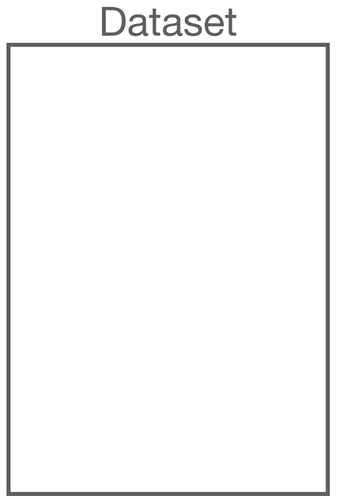
Distributed system: Shared-nothing architecture

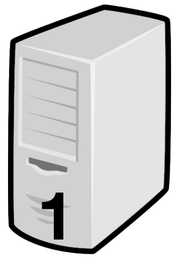


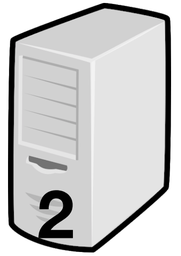
Distributed system: Shared-nothing architecture



Data distribution across multiple nodes

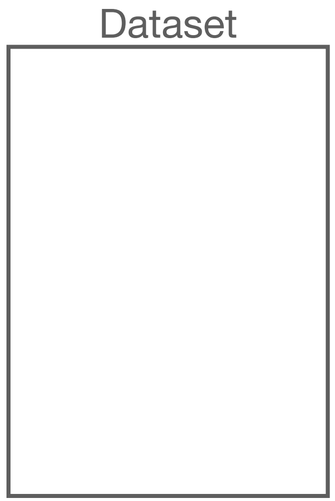


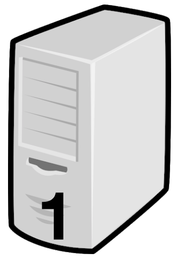


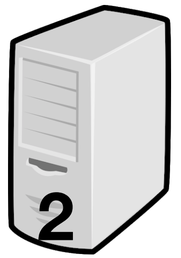


Data distribution across multiple nodes

- **Partitioning**
  - 2 steps process:

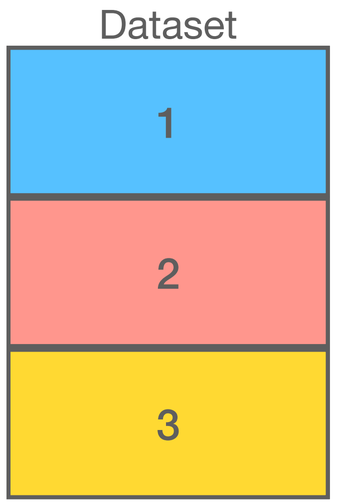


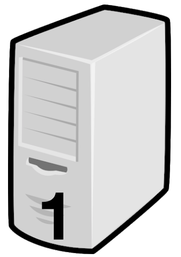


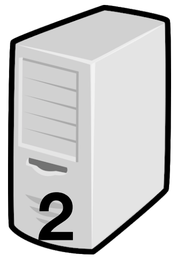


Data distribution across multiple nodes

- **Partitioning**
  - 2 steps process:
    - Split data into (typically disjoint) subsets —> **partitions**

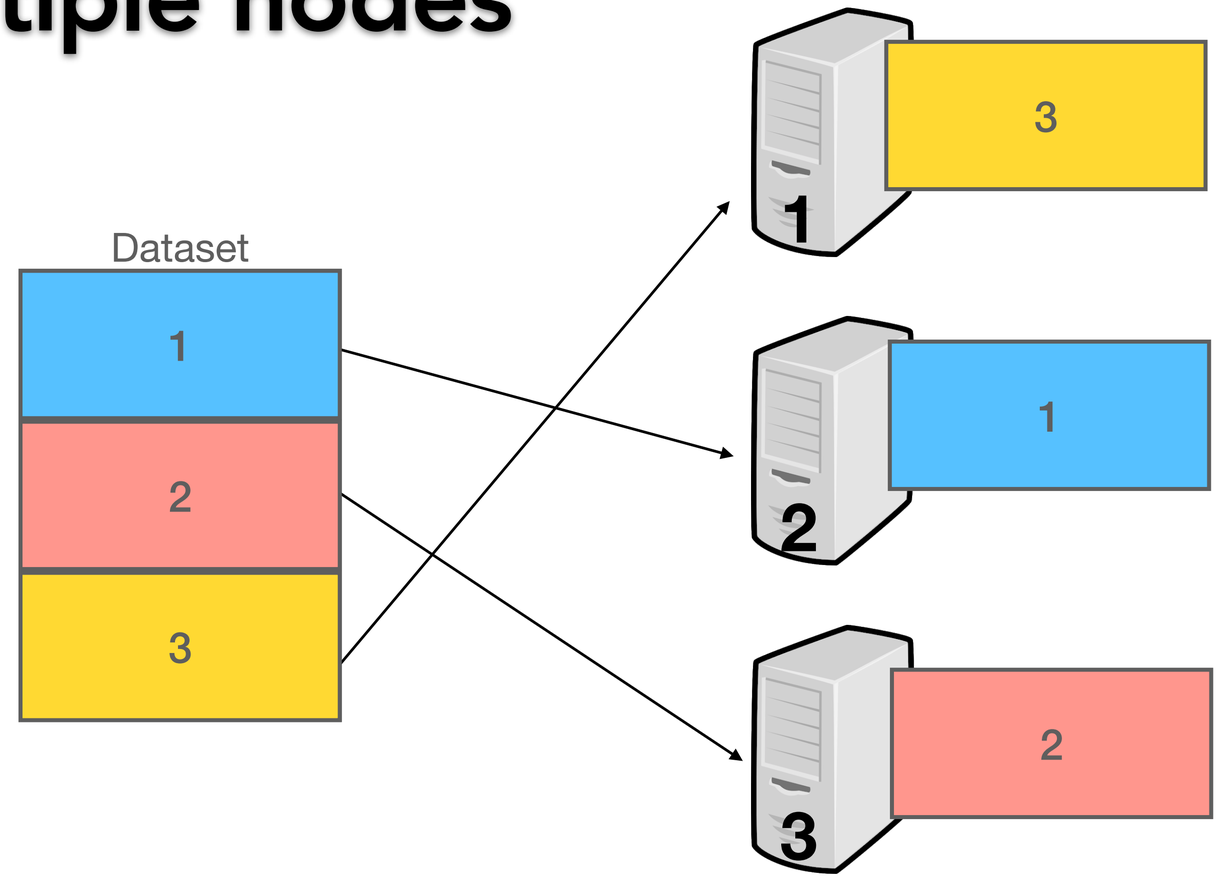






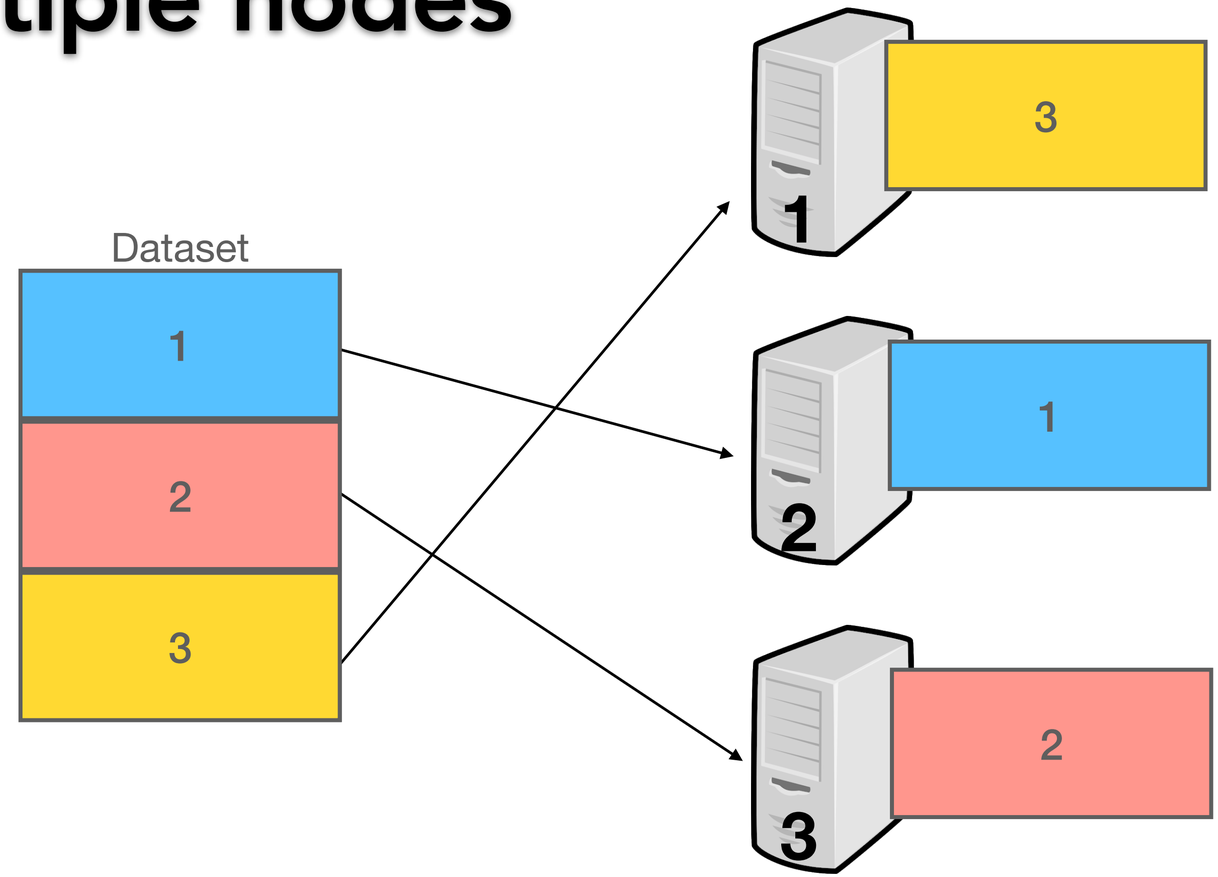
Data distribution across multiple nodes

- **Partitioning**
  - 2 steps process:
    - Split data into (typically disjoint) subsets —> **partitions**
    - Assign each partition to a different node (aka sharding)



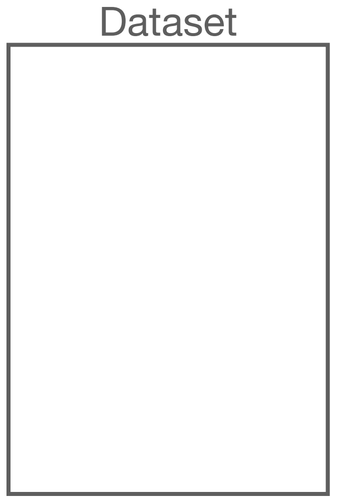
Data distribution across multiple nodes

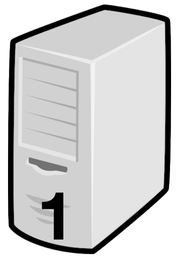
- **Partitioning**
  - 2 steps process:
    - Split data into (typically disjoint) subsets —> **partitions**
    - Assign each partition to a different node (aka sharding)

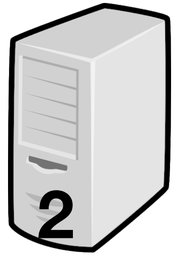


Main reason for partitioning is **scalability**

Data distribution across multiple nodes

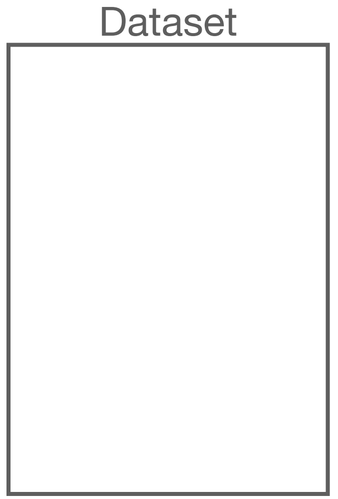


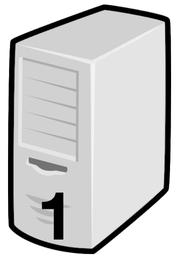


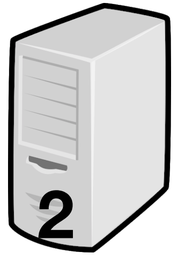


Data distribution across multiple nodes

- Replication

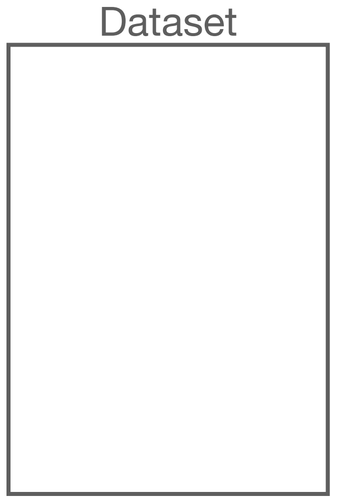


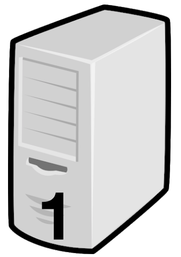


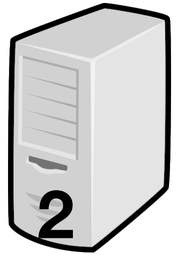


Data distribution across multiple nodes

- **Replication**
  - Keeping a copy of the same data on different nodes (potentially in different locations)

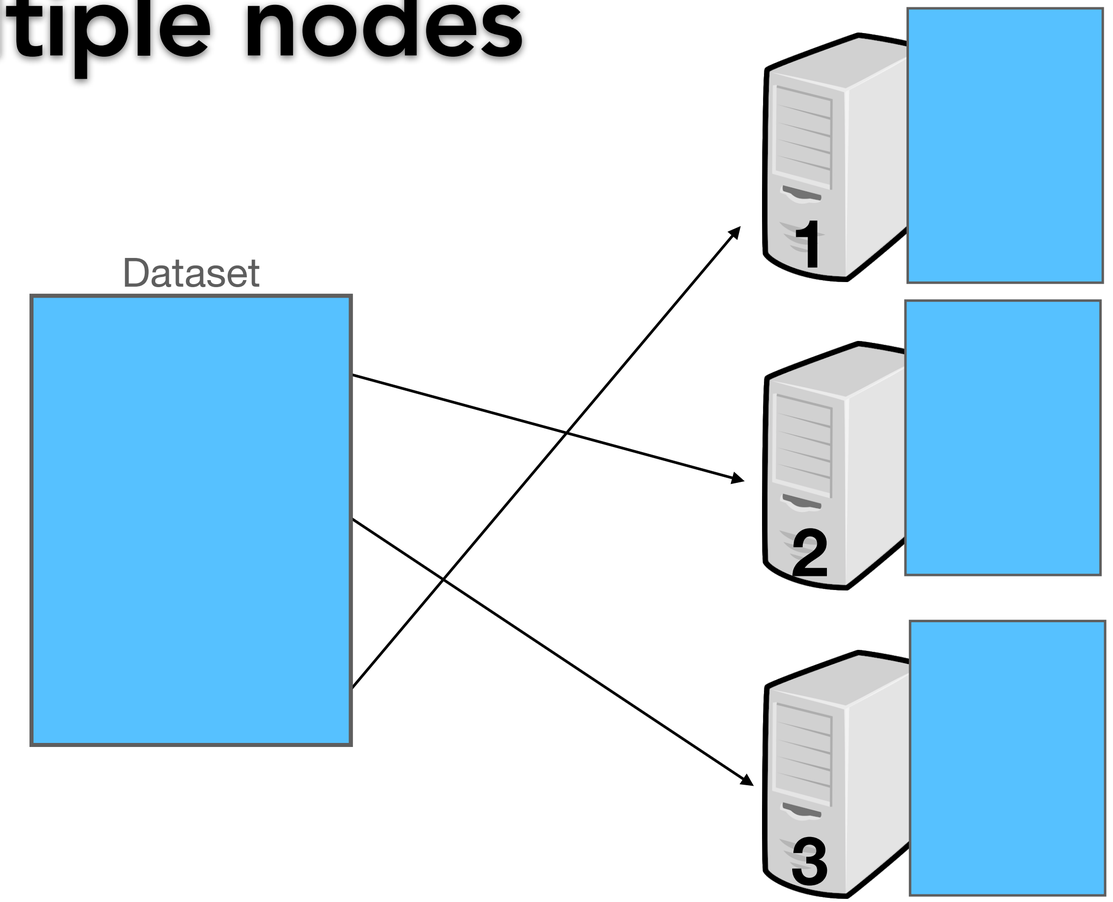






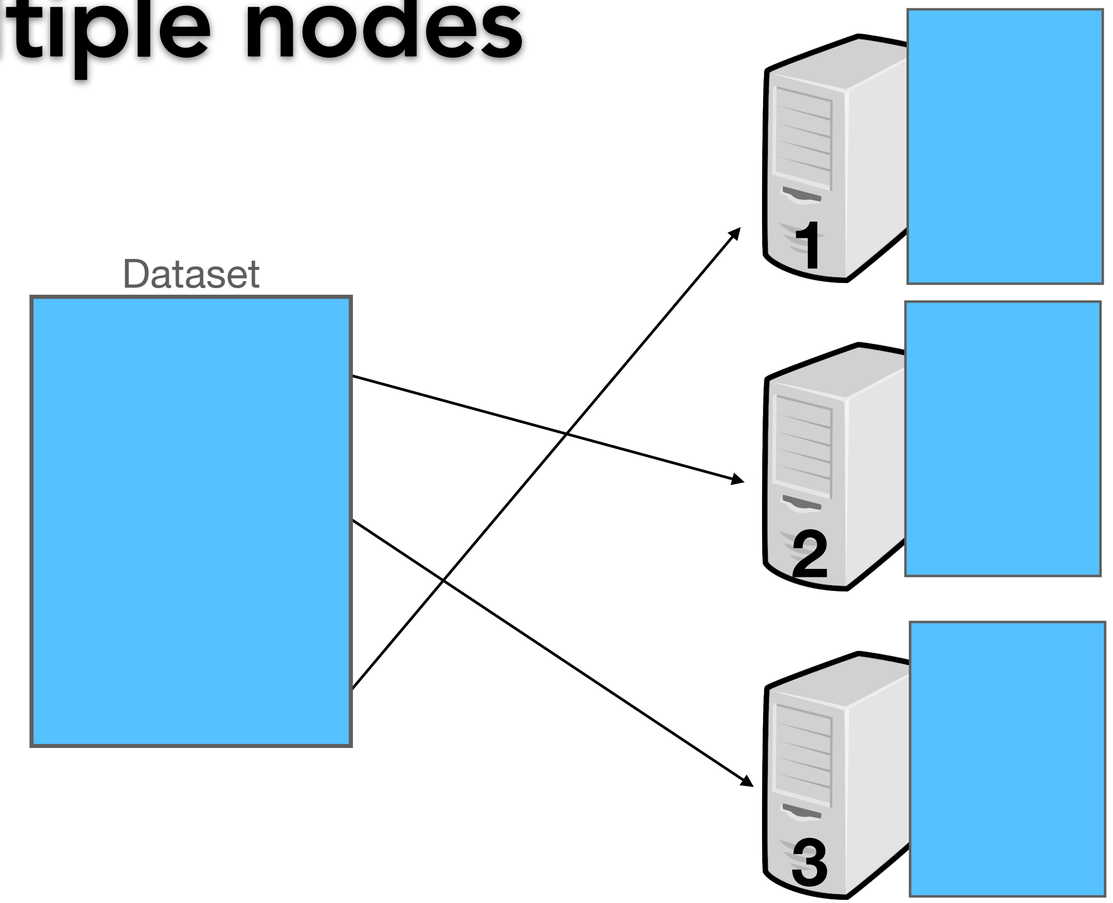
Data distribution across multiple nodes

- **Replication**
  - Keeping a copy of the same data on different nodes (potentially in different locations)



Data distribution across multiple nodes

- **Replication**
  - Keeping a copy of the same data on different nodes (potentially in different locations)

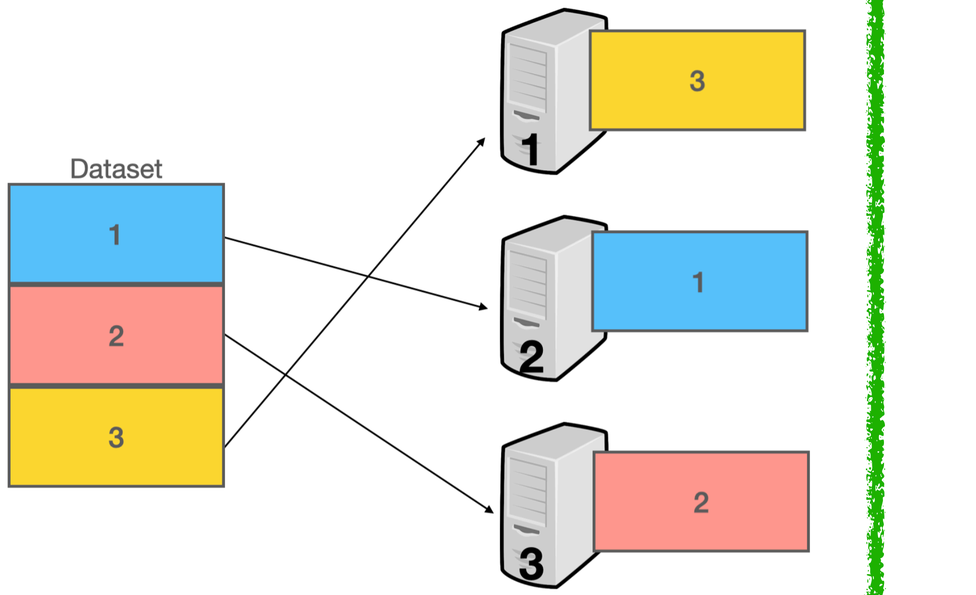


Main reason for replication is **availability**

Data distribution across multiple nodes

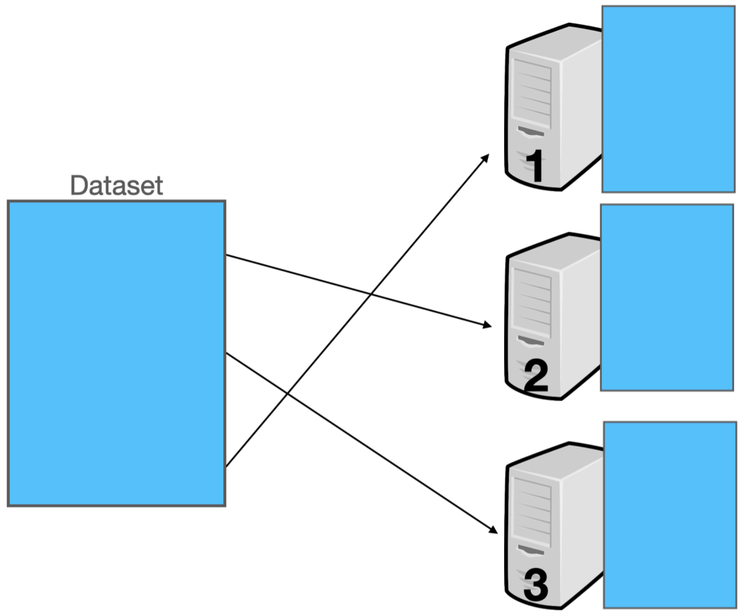
✦ Partitioning

- ✦ Split data into (typically disjoint) subsets —> **partitions**
- ✦ Assign each partition to a different node (aka sharding)

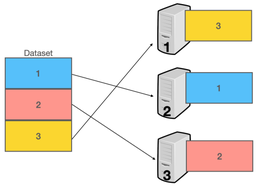


✦ Replication

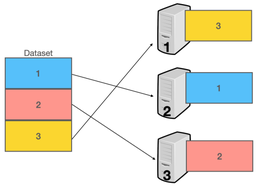
- ✦ Keep a copy of the same data on different nodes (potentially in different locations)



Partitioning

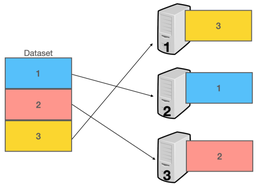


Partitioning



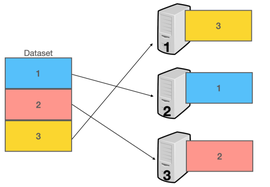
- Goal of partitioning:
  - Typically, we want to distribute the load **evenly**
  - If a node receives much more data (**skewed partitioning**) than others, performance is decreased
  - **Hot spot**: a node/partition with disproportionately high load

Partitioning



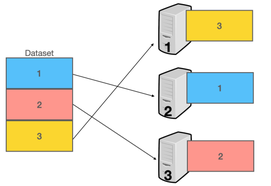
- Goal of partitioning:
  - Typically, we want to distribute the load **evenly**
  - If a node receives much more data (**skewed partitioning**) than others, performance is decreased
  - **Hot spot**: a node/partition with disproportionately high load
- Simple idea: randomly assign each data point to a node
  - If you want to read a certain data point, how do you know where it is located?

Partitioning

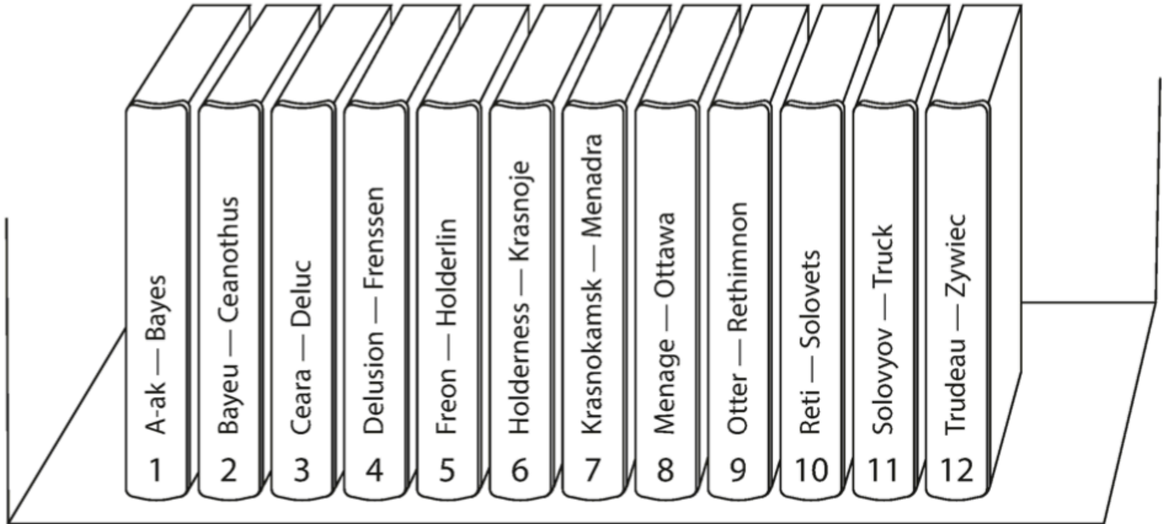


- Goal of partitioning:
  - Typically, we want to distribute the load **evenly**
  - If a node receives much more data (**skewed partitioning**) than others, performance is decreased
  - **Hot spot**: a node/partition with disproportionately high load
- Simple idea: randomly assign each data point to a node
  - If you want to read a certain data point, how do you know where it is located?

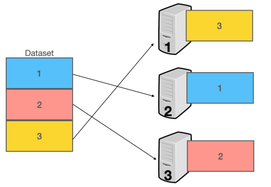
Range partitioning



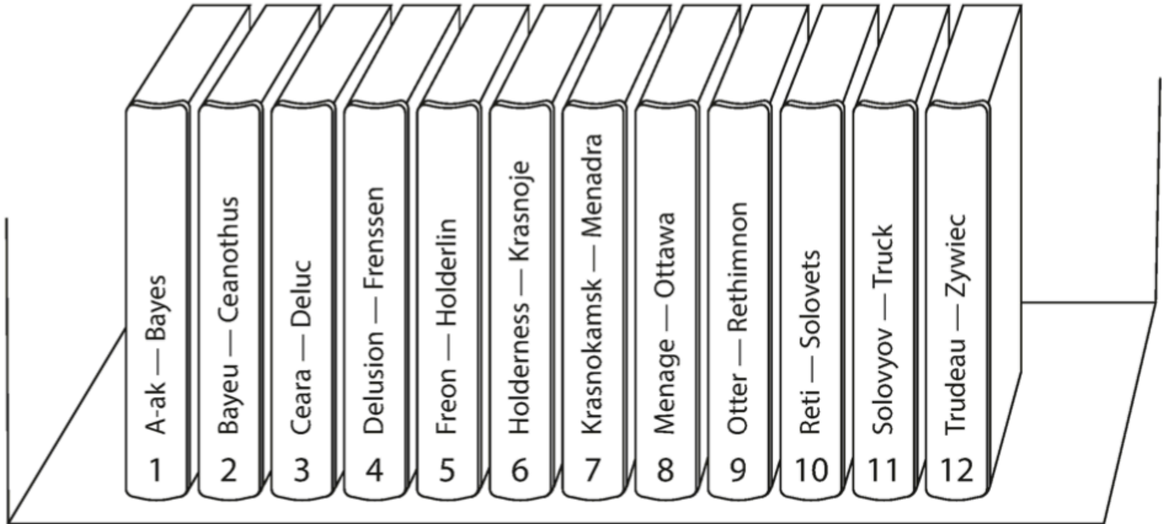
- Assume you can have unique keys (**key-value** data model)
- Assign a continuous range of keys to each partition
- Sort keys within each partition
- Data may not be evenly distributed —> ranges might not be equally spaced because —> **hotspots**
- Boundaries can be chosen manually or automatically



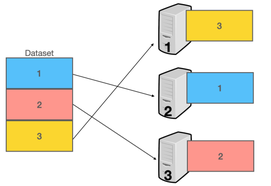
Range partitioning



- Assume you can have unique keys (**key-value** data model)
- Assign a continuous range of keys to each partition
- Sort keys within each partition
- Data may not be evenly distributed —> ranges might not be equally spaced because —> **hotspots** 
- Boundaries can be chosen manually or automatically

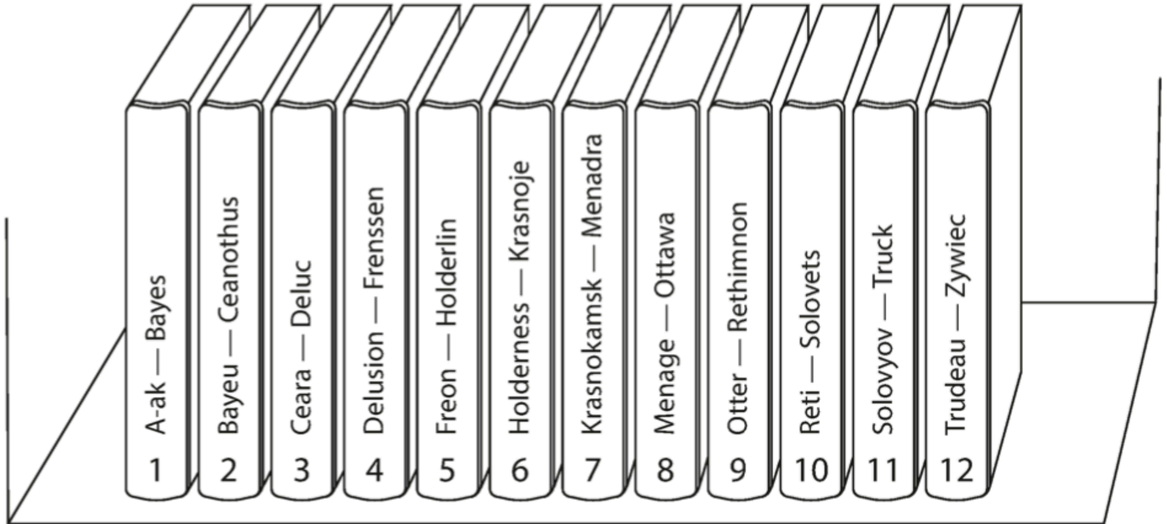


Range partitioning

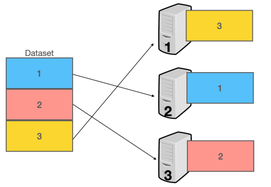


- Assume you can have unique keys (**key-value** data model)
- Assign a continuous range of keys to each partition
- Sort keys within each partition
- Data may not be evenly distributed —> ranges might not be equally spaced because —> **hotspots**
- Boundaries can be chosen manually or automatically

A key-value model is commonly used in NoSQL databases a.k.a. key-value stores

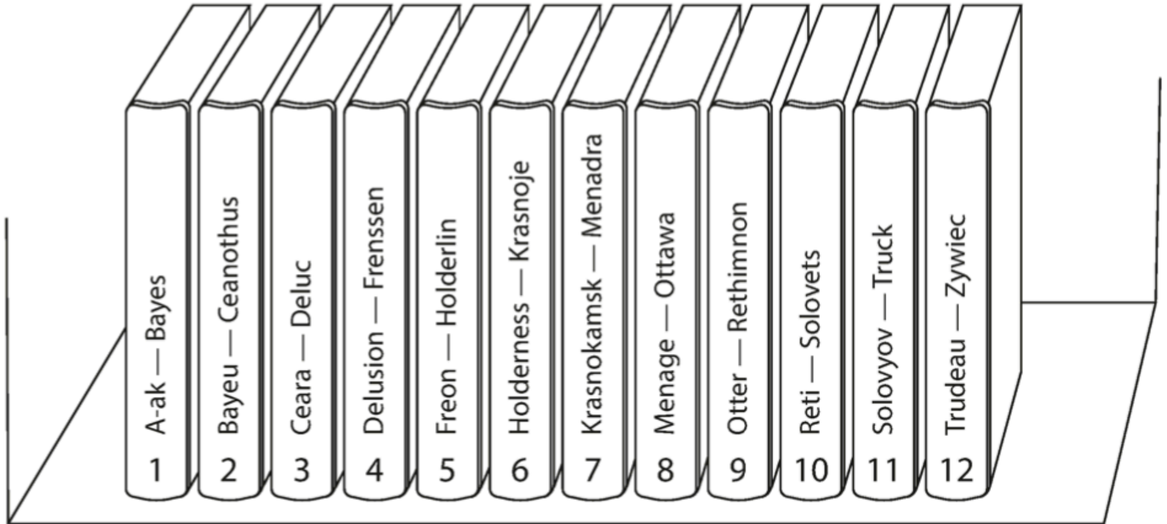


Range partitioning

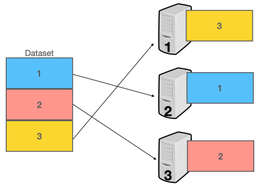


- Assur
- Assig
- Sort keys within each partition
- Ranges might not be equally spaced because data is not equally distributed —> **hotspots**
- Boundaries can be chosen manually or automatically

What would happen if we use a timestamp as the key?

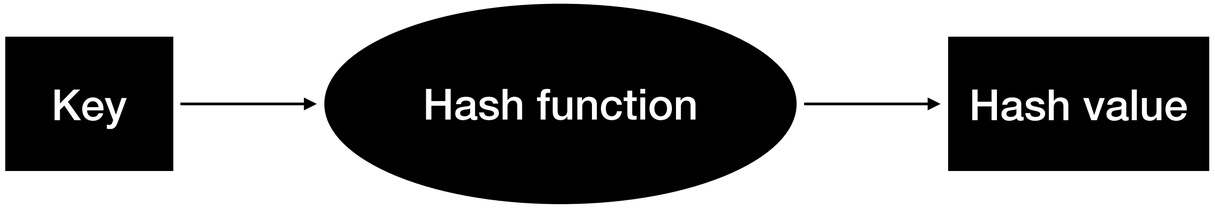


Hash partitioning



- Use a hash function on a key to determine the partition

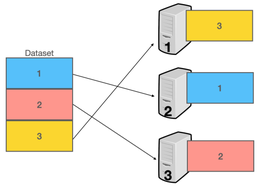
Hash function



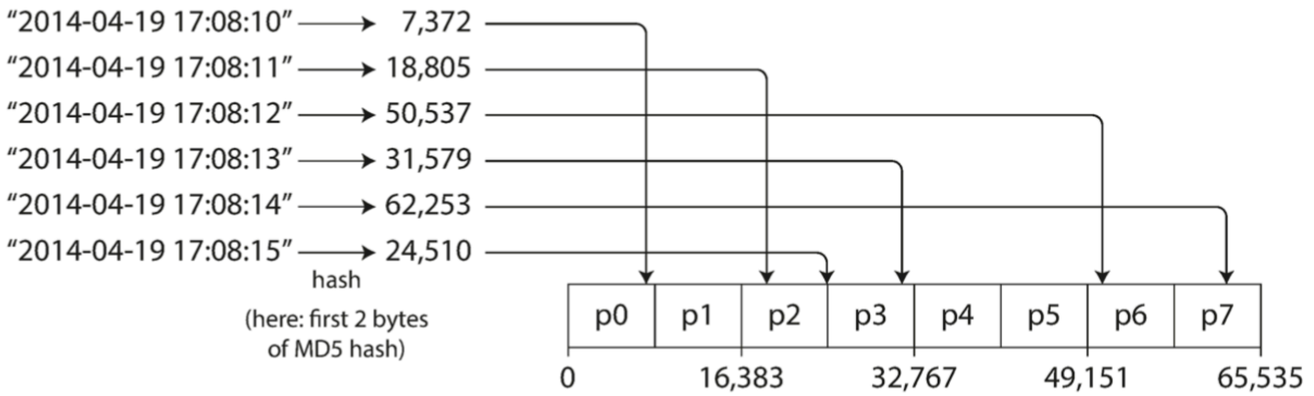
- Assume a 32-bit hash function
- Input: “abcd” —> Output: number from [0, 2^{32}-1]
- Input: “abce” —> Output: number from [0, 2^{32}-1]
- Common hash function: MD5[1] used in Cassandra, MongoDB

[1] https://en.wikipedia.org/wiki/MD5

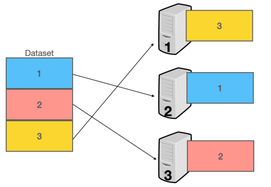
Hash partitioning



- Use a hash function on a key to determine the partition
- A good hash function will **distributed keys evenly**, even in the case of skewed data



Hash partitioning

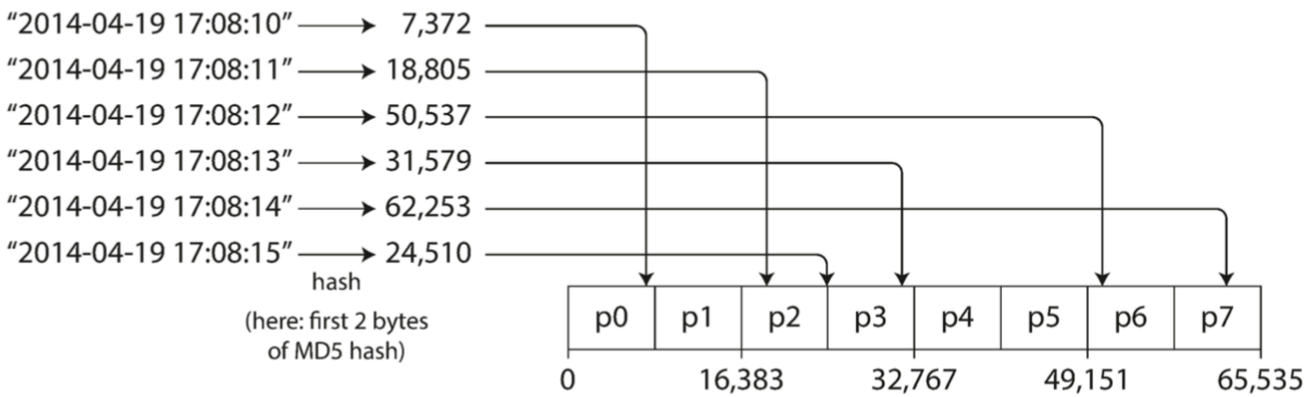


- Use a hash function on a key to determine the partition
- A go… in the c…

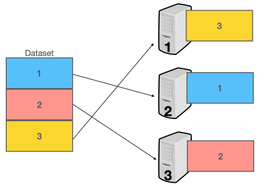
Assume a range query

```
SELECT temperature
FROM sensorDB
WHERE "2014-04-19 17:08:00" < timestamp < "2014-04-19 17:08:59"
```

What is the problem?



Hash partitioning



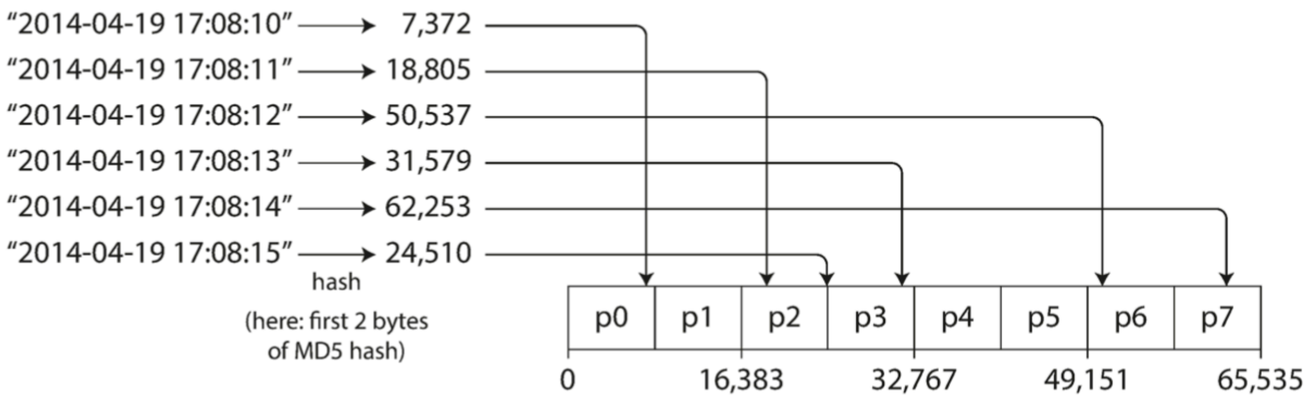
- Use a hash function on a key to determine the partition
- A go… in the c…

Assume a range query

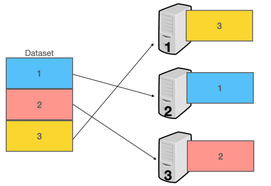
```
SELECT temperature
FROM sensorDB
WHERE "2014-04-19 17:08:00" < timestamp < "2014-04-19 17:08:59"
```

What is the problem?

- Inefficient for range queries



Hash partitioning



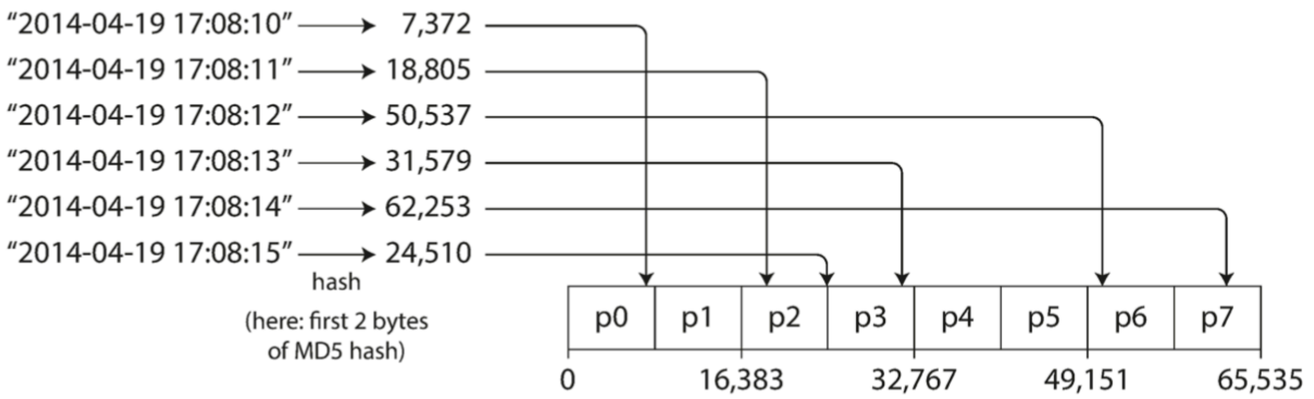
- Use a hash function on a key to determine the partition
- A go… in the c…

Assume a range query

```
SELECT temperature
FROM sensorDB
WHERE "2014-04-19 17:08:00" < timestamp < "2014-04-19 17:08:59"
```

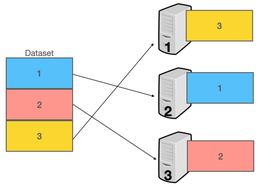
What is the problem?

- Inefficient for range queries



Hashing helps to deal with skew. However, if reads/writes are skewed your can still have hotspots.

Partitioning in Cassandra



- Hybrid model:
  - Use composite key: (k1,k2)
  - Hash partition on k1
  - Index on k2 within a partition

Index for sorting within the partition

Hash to determine the partition

User_id	Tweet	Timestamp

How to assign partitions to nodes

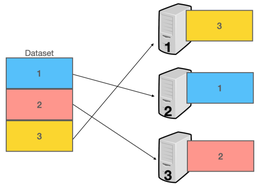
- 0 ≤ hash(key) < b0 —> partition 1, b0 ≤ hash(key) < b1 —> partition 2, etc.
- hash(key) mod N ?

How to assign partitions to nodes

- 0 ≤ hash(key) < b0 —> partition 1, b0 ≤ hash(key) < b1 —> partition 2, etc.
- hash(key) mod N ?

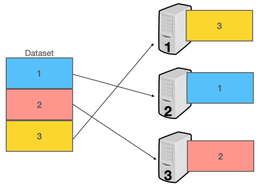
What if the number of nodes N changes?

Rebalancing partitions

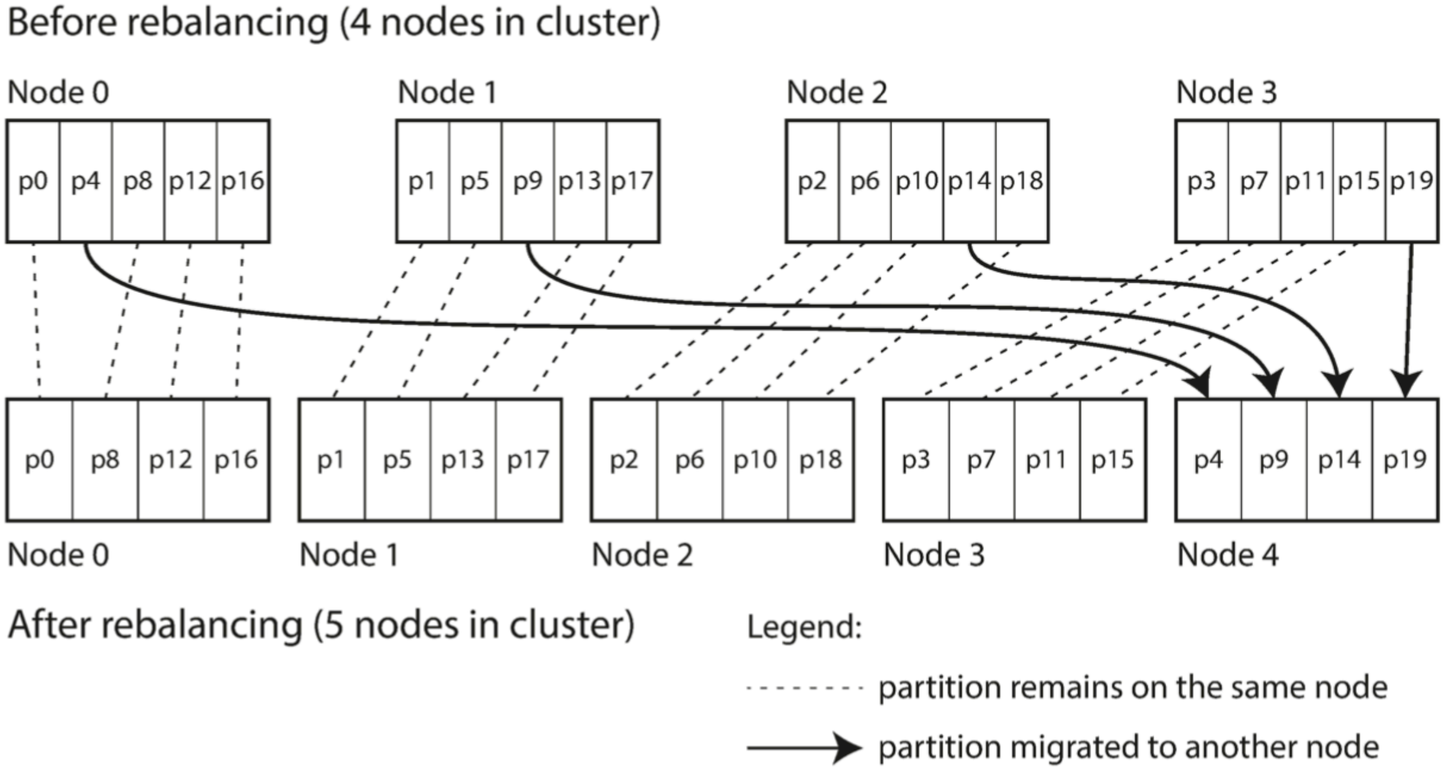


- Things may change
  - Query throughput increases —> add more resources
  - Dataset size increases —> add more resources
  - Machines fail —> add new machines
- Move data from one node to another —> rebalancing/repartitioning
- Typical requirements
  - After rebalancing, load should be evenly distributed
  - While rebalancing, system should continue to operate
  - Minimize the data movement, to minimise network and disk I/O load

Rebalancing with fixed number of partitions

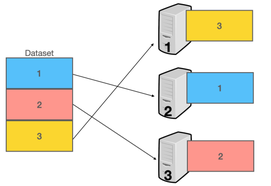


- # partitions >> # nodes
- Each node has > 1 partition
- e.g., cluster with 10 nodes, 100 partitions per node
- If a new node is added, move partitions from other nodes until evenly distributed

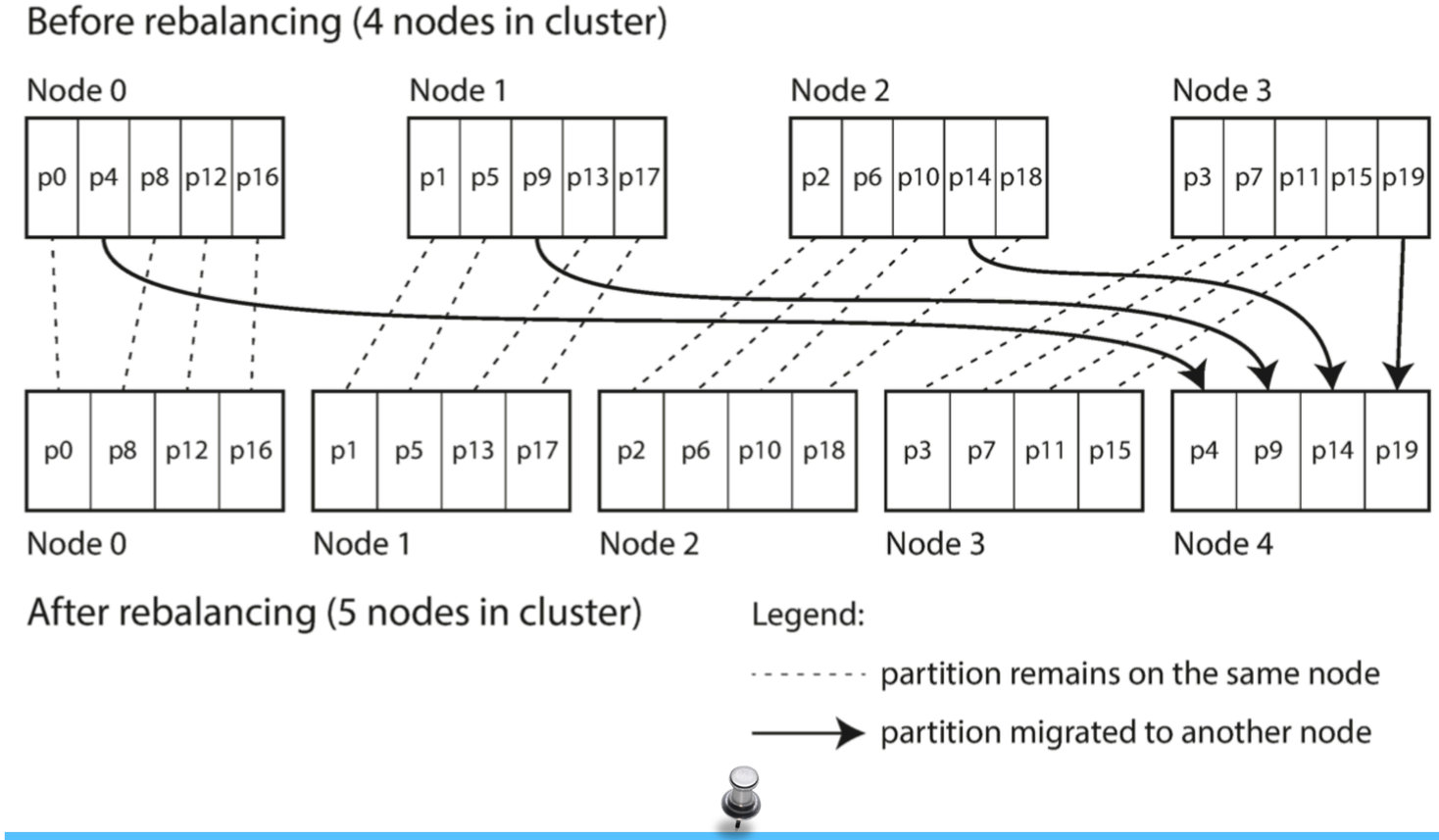


elastic

Rebalancing with fixed number of partitions

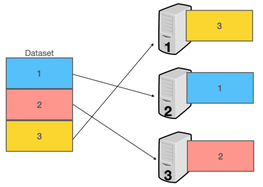


- # partitions >> # nodes
- Each node has > 1 partition
- e.g., cluster with 10 nodes, 100 partitions per node
- If a new node is added, move partitions from other nodes until evenly distributed



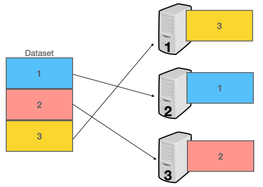
The number of partitions does not change, nor does the assignment of keys to partitions. The only thing that changes is the assignment of partitions to nodes.

Rebalancing with dynamic partitioning



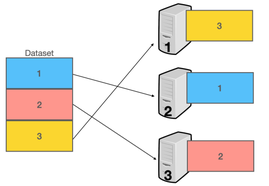
- Fits well for systems using **range partitioning**
- When a partition exceeds a configured fixed size, split it in two partitions
- Similarly, if a partition becomes too small (size below a configured threshold), merge it with another partition
- Nodes may sit idle when dataset too small to fit in 1 partition
  - Solution: pre-splitting

Rebalancing with dynamic partitioning



- Fits well for systems using **range partitioning**
- When a partition exceeds a configured fixed size, split it in two partitions
- Similarly, if a partition becomes too small (size below a configured threshold), merge it with another partition
- Nodes may sit idle when dataset too small to fit in 1 partition 👎
  - Solution: pre-splitting

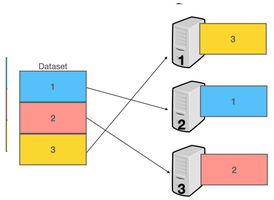
Rebalancing with dynamic partitioning



- Fits well for systems using **range partitioning**
- When a partition exceeds a configured fixed size, split it in two partitions
- Similarly, if a partition becomes too small (size below a configured threshold), merge it with another partition
- Nodes may sit idle when dataset too small to fit in 1 partition 👎
  - Solution: pre-splitting

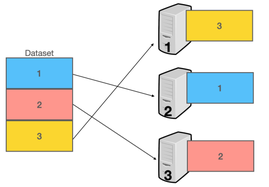
partitions follows data volume

Rebalancing proportionally to nodes



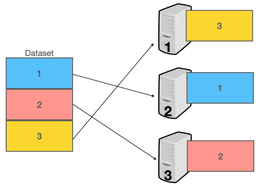
- Fixed number of partitions per node
- For stable number of nodes, the size of partitions increases proportionally to the dataset size
- When adding a new node, the node chooses randomly partitions from other nodes to split and assigns half of them to itself
  - the size of partitions decreases

Request routing: how to get access to the right node

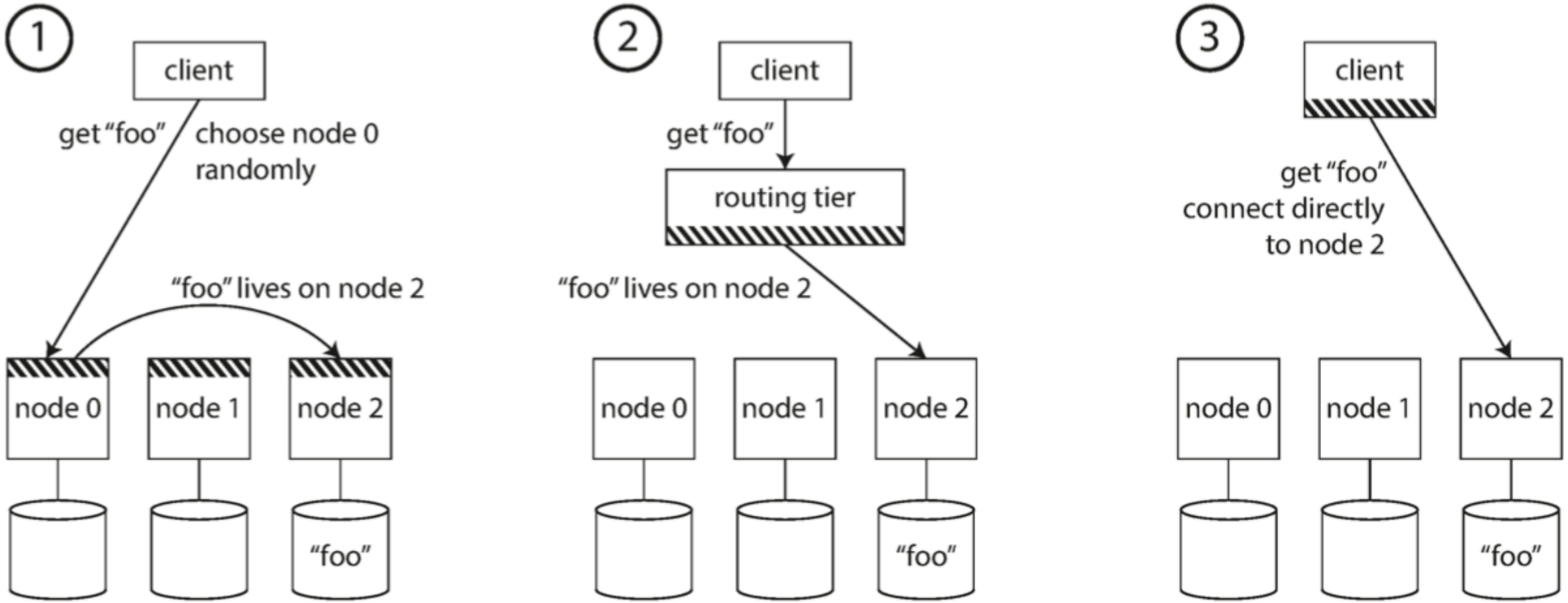


- ✦ 3 main strategies

Request routing: how to get access to the right node

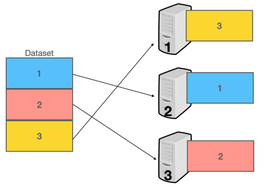


- 3 main strategies

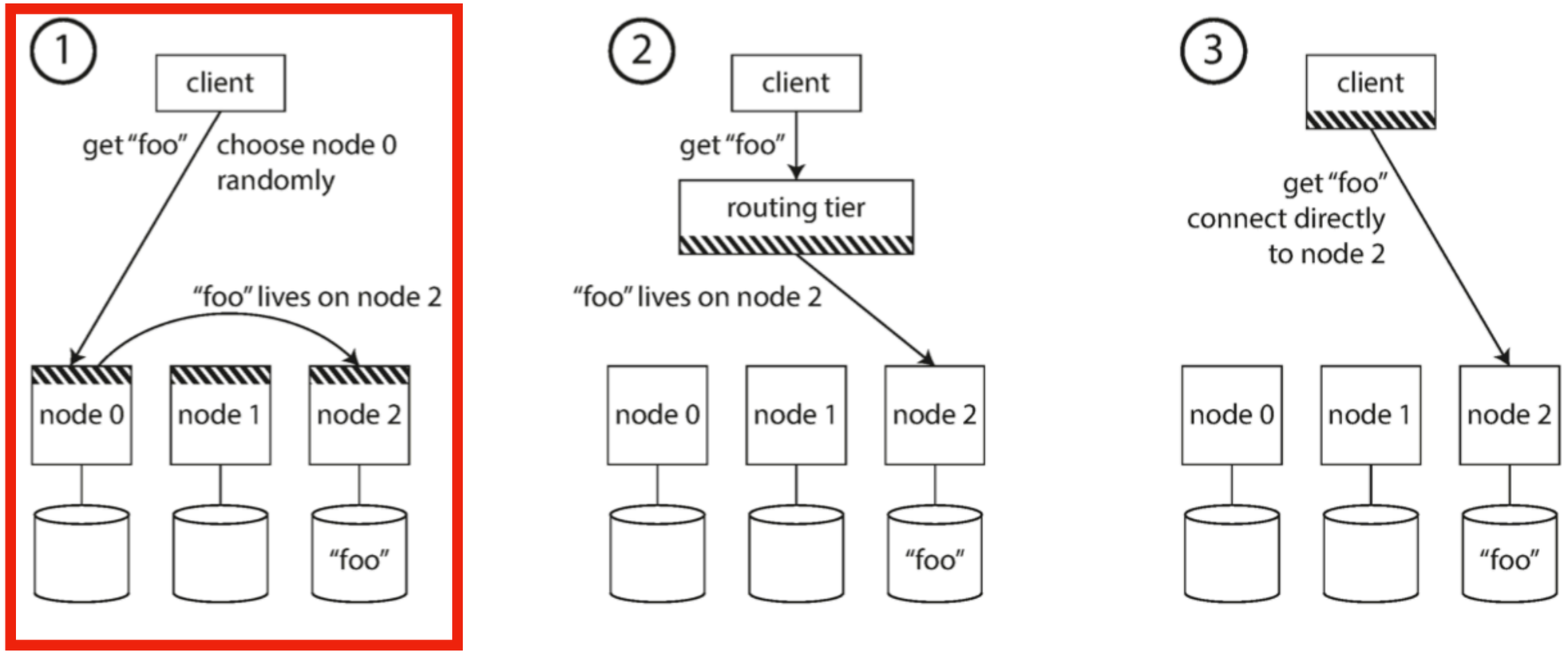


= the knowledge of which partition is assigned to which node

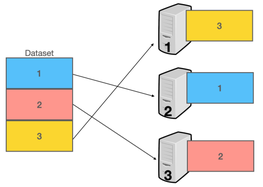
Request routing: how to get access to the right node



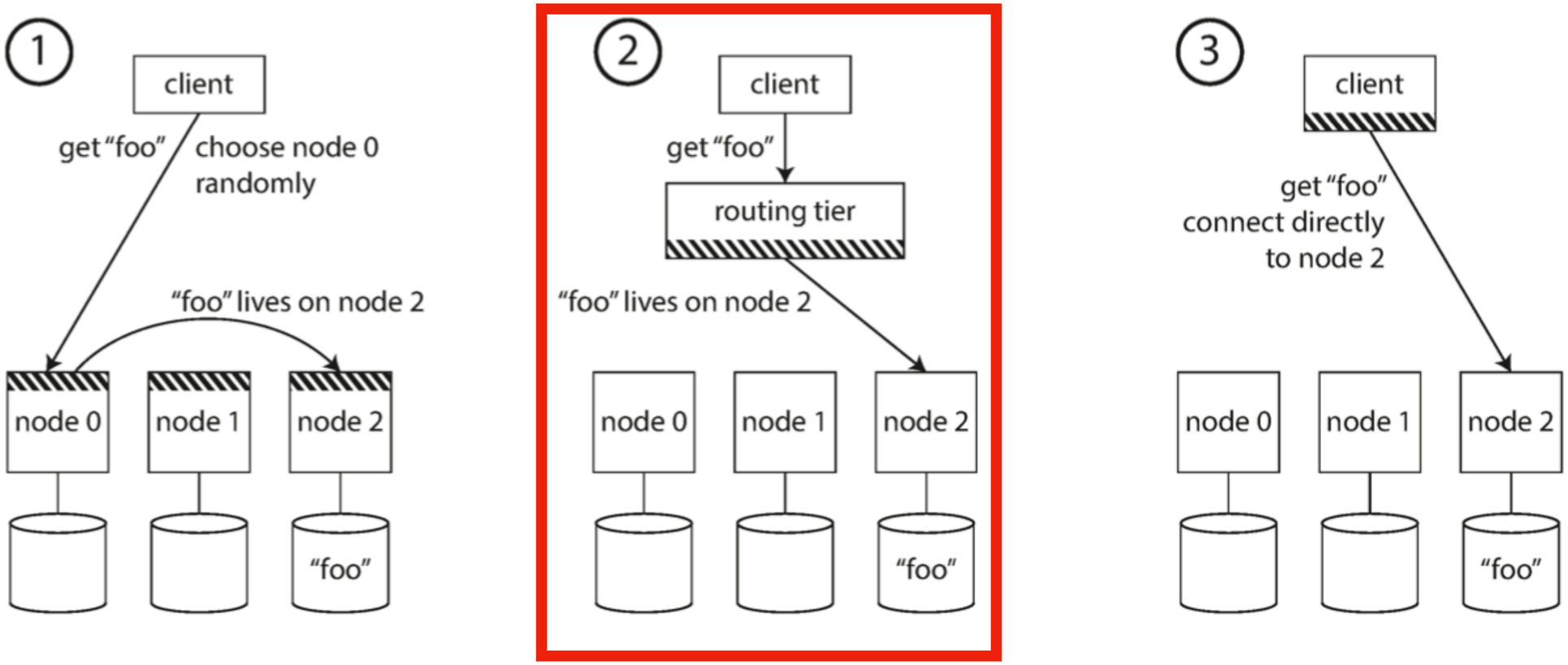
- 3 main strategies



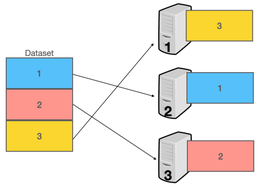
Request routing: how to get access to the right node



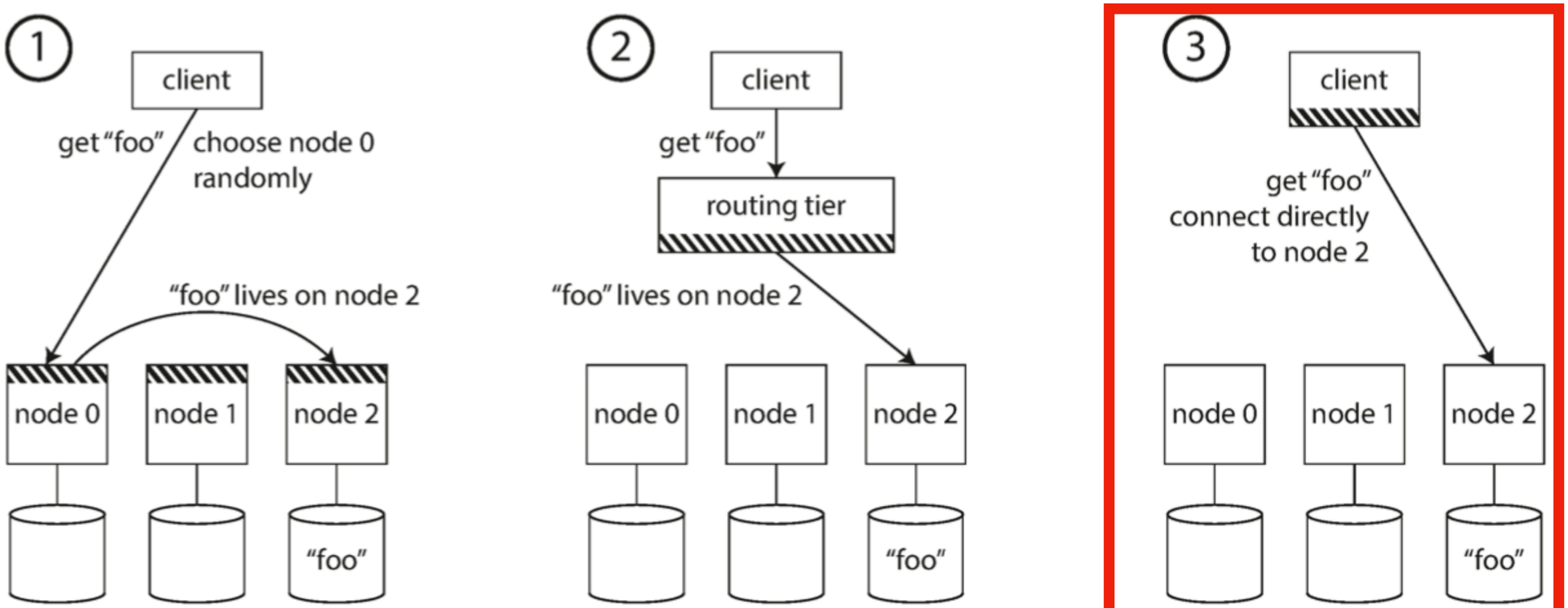
- 3 main strategies



Request routing: how to get access to the right node

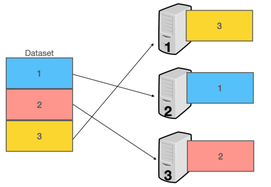


- 3 main strategies

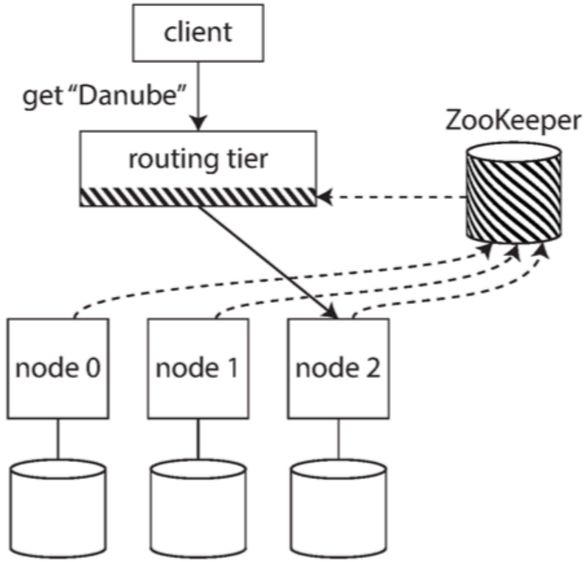


= the knowledge of which partition is assigned to which node

How to keep up with changes? An example with Zookeeper



- Keeps cluster metadata
- Each node registers in Zookeeper
- Keeps mapping of partitions to nodes
- Services subscribe to Zookeeper



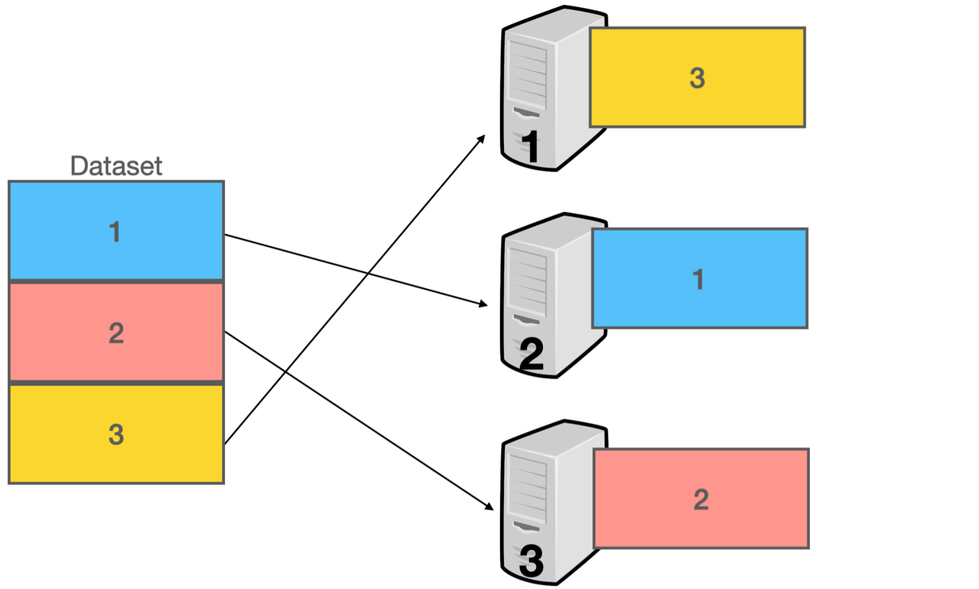
Key range	Partition	Node	IP address
A-ak — Bayes	partition 0	node 0	10.20.30.100
Bayeu — Ceanothus	partition 1	node 1	10.20.30.101
Ceara — Deluc	partition 2	node 2	10.20.30.102
Delusion — Frenssen	partition 3	node 0	10.20.30.100
Freon — Holderlin	partition 4	node 1	10.20.30.101
Holderness — Krasnoje	partition 5	node 2	10.20.30.102
Krasnokamsk — Menadra	partition 6	node 0	10.20.30.100
Menage — Ottawa	partition 7	node 1	10.20.30.101
Otter — Rethimnon	partition 8	node 2	10.20.30.102
Reti — Solovets	partition 9	node 0	10.20.30.100
Solovyov — Truck	partition 10	node 1	10.20.30.101
Trudeau — Zywiec	partition 11	node 2	10.20.30.102

= the knowledge of which partition is assigned to which node

Data distribution across multiple nodes

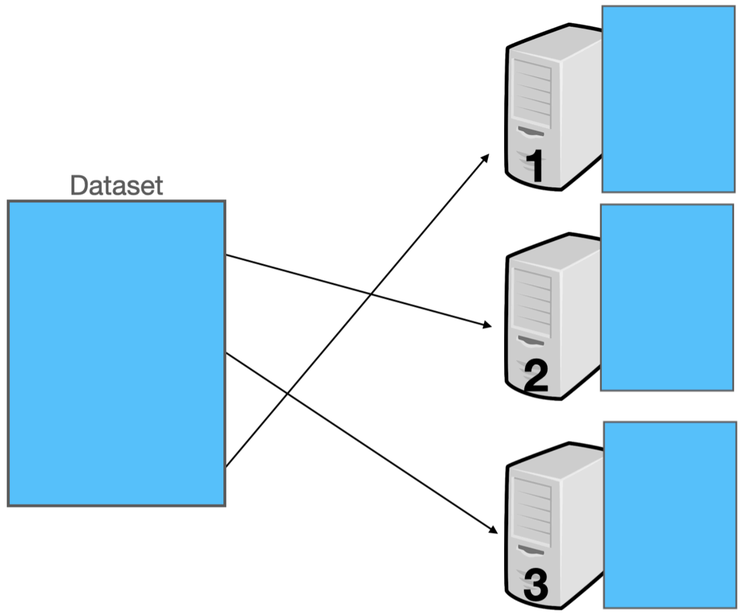
✦ Partitioning

- ✦ Splitting data into (typically disjoint) subsets —> **partitions**
- ✦ Each partition is assigned to a different node (aka sharding)



✦ Replication

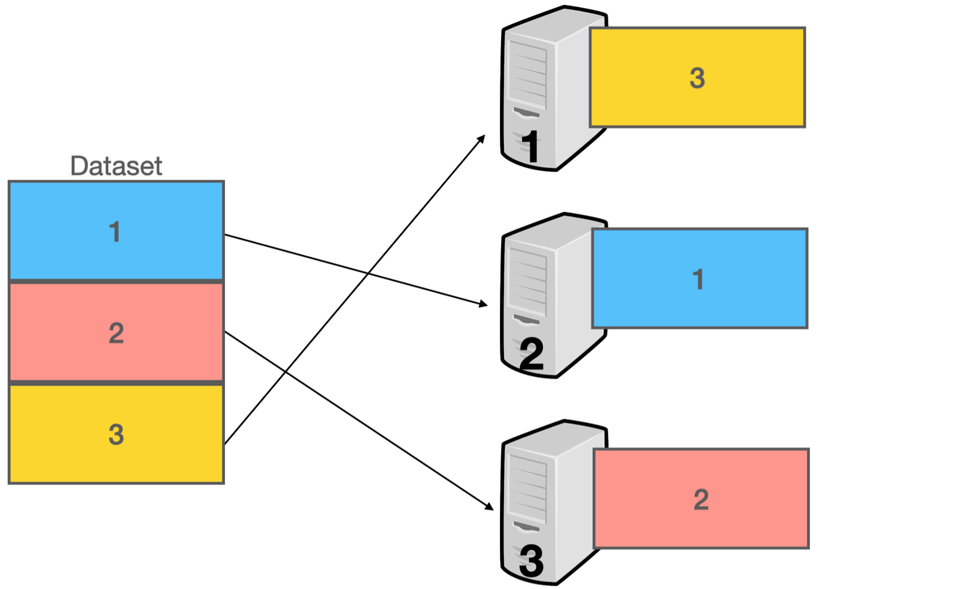
- ✦ Keeping a copy of the same data on different nodes (potentially in different locations)



Data distribution across multiple nodes

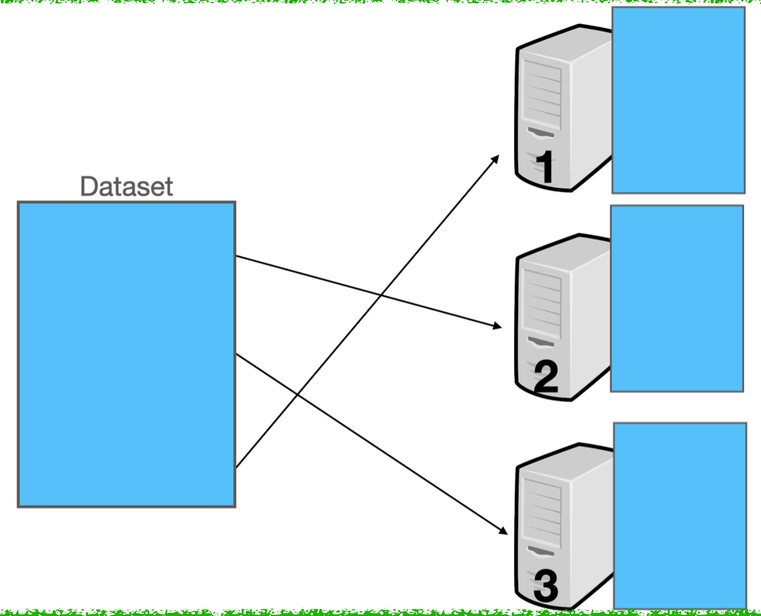
✦ Partitioning

- ✦ Splitting data into (typically disjoint) subsets —> **partitions**
- ✦ Each partition is assigned to a different node (aka sharding)

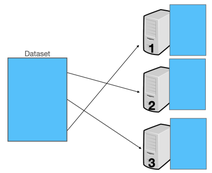


✦ Replication

- ✦ Keeping a copy of the same data on different nodes (potentially in different locations)

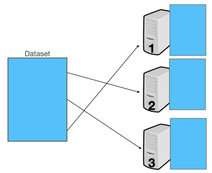


Replication



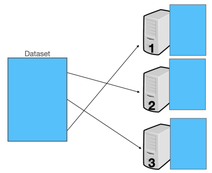
- Keeping a copy of the same data on multiple machines connected via a network
- Why?
- Assumption: dataset fits in 1 node

Replication



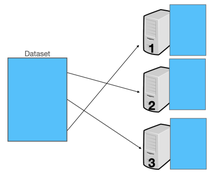
- Keeping a copy of the same data on multiple machines connected via a network
- Why?
  - Reduce **latency**: keep data geographically close to your users
  - Increase **availability**: system continues working even if some nodes have failed
  - Increase **throughput**: scale out #nodes that can serve read queries
- Assumption: dataset fits in 1 node

Replication

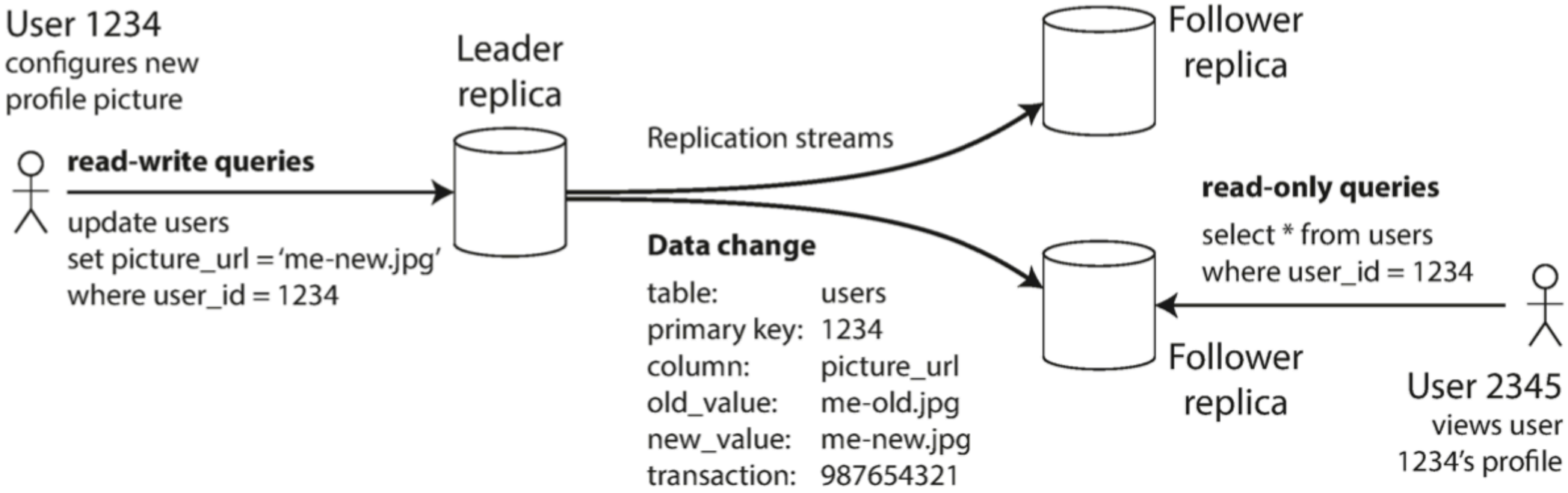


- Keeping a copy of the same data on multiple machines connected via a network
- Why?
  - Reduce
  - Increas have failed
  - Increase **throughput**: scale out #nodes that can serve read queries
- Assumption: dataset fits in 1 node

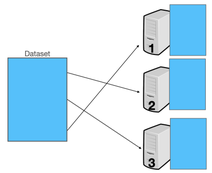
Leader-based replication



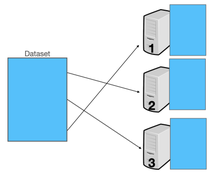
- **Replica**: a copy of the data
- How to ensure that all data ends up to all replicas?

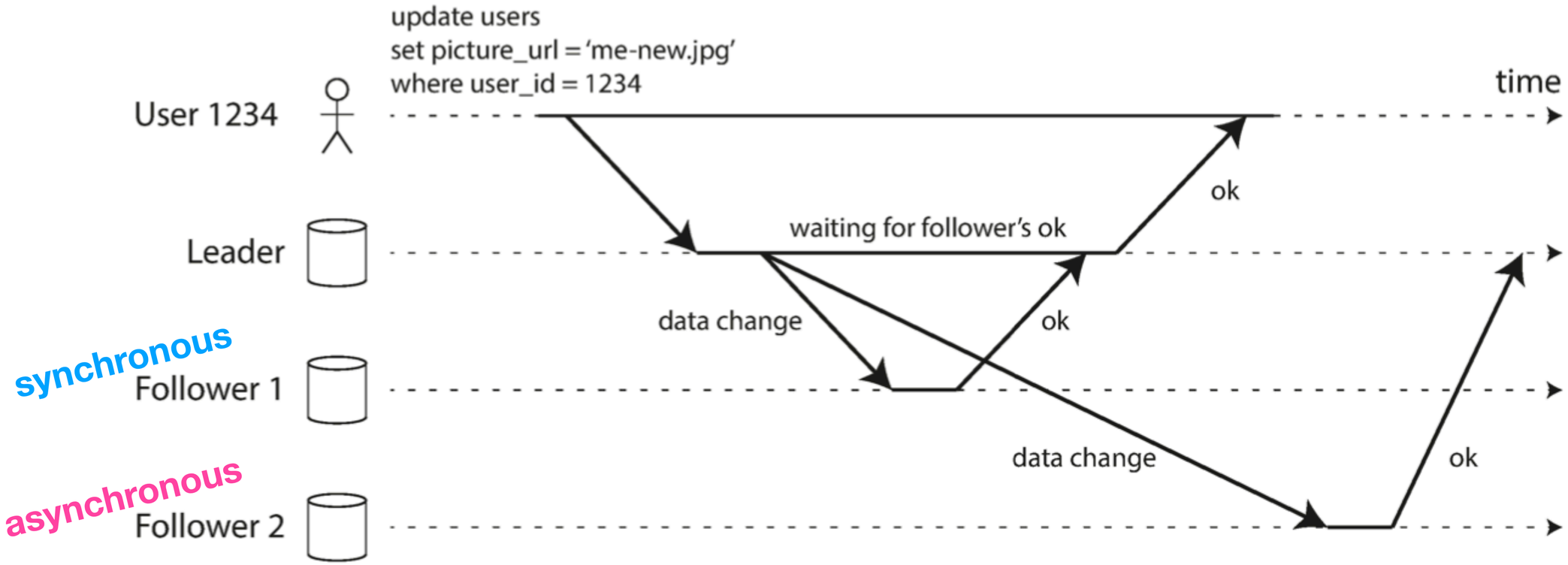


Synchronous vs asynchronous replication

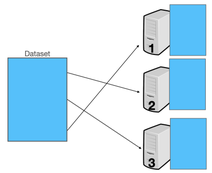


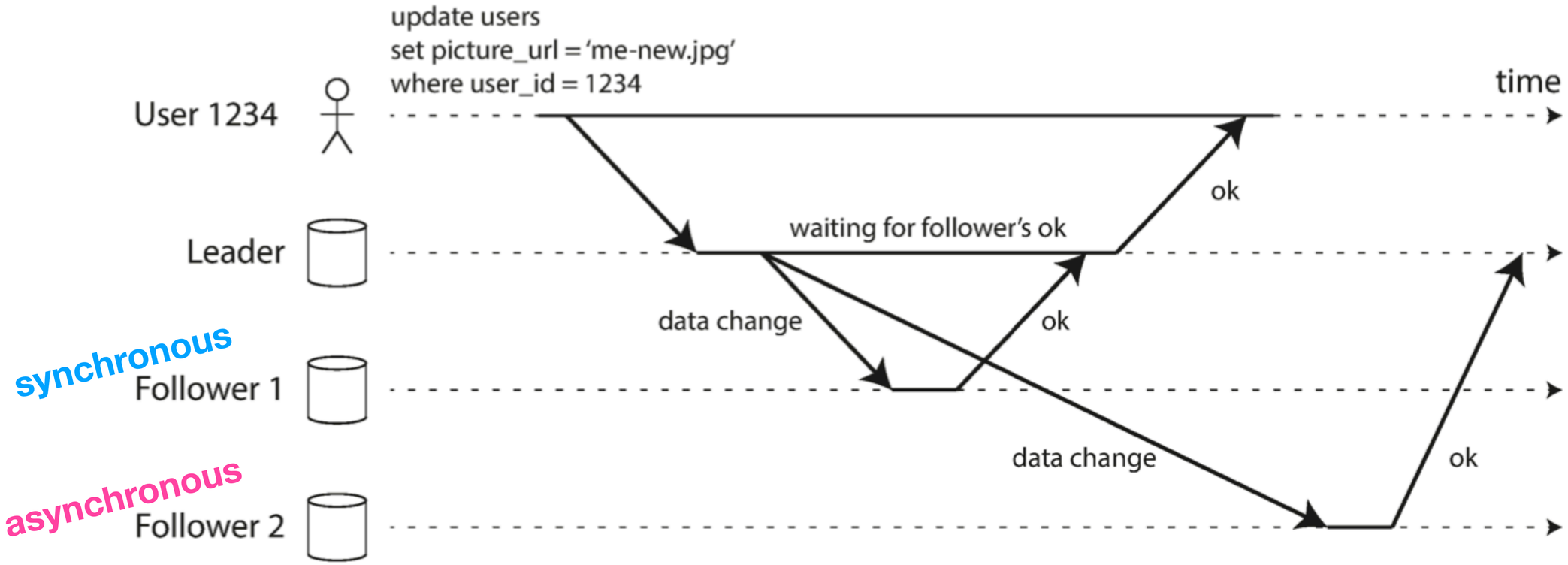
Synchronous vs asynchronous replication





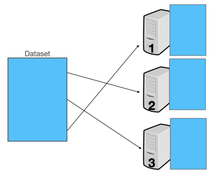
Synchronous vs asynchronous replication

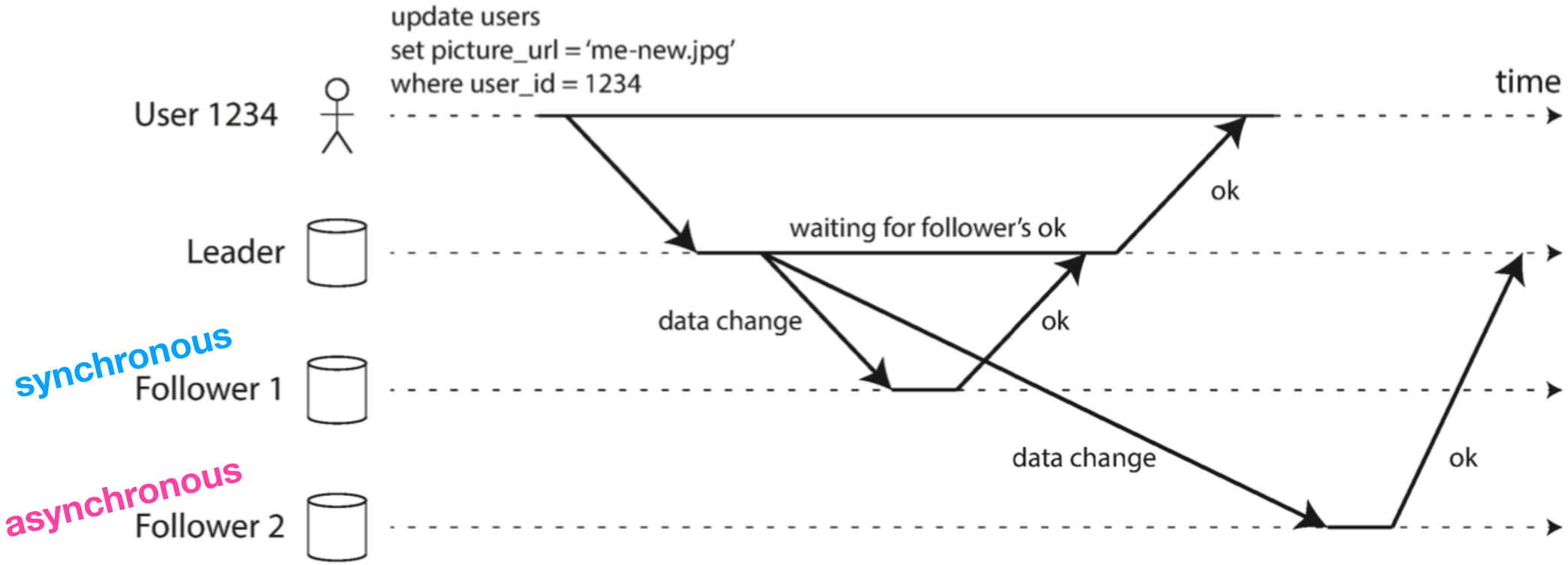




- **Synchronous**:
  - Follower has an up-to-date copy
  - Bottleneck if the follower does not response (e.g., crashed)

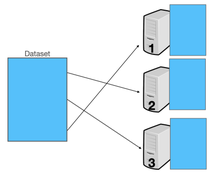
Synchronous vs asynchronous replication

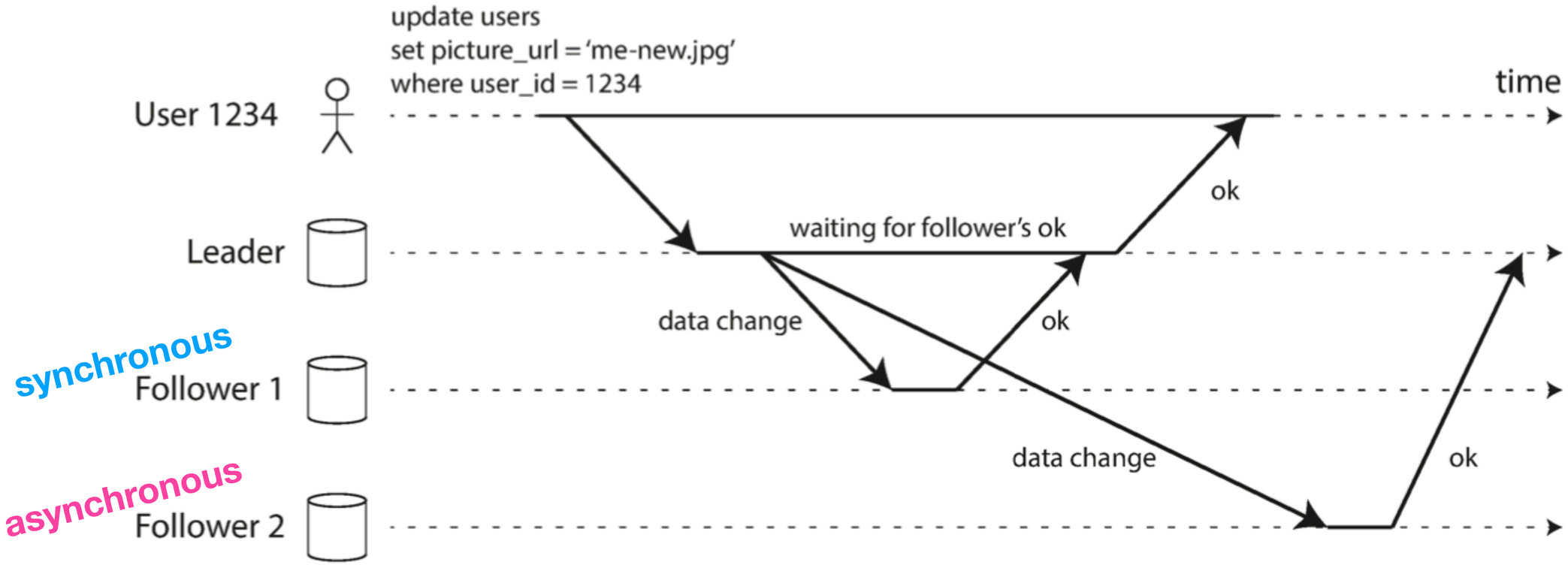




- **Asynchronous**:
  - Higher write throughput
  - Weak durability (a write may be lost)

Synchronous vs asynchronous replication

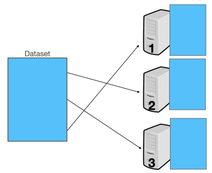


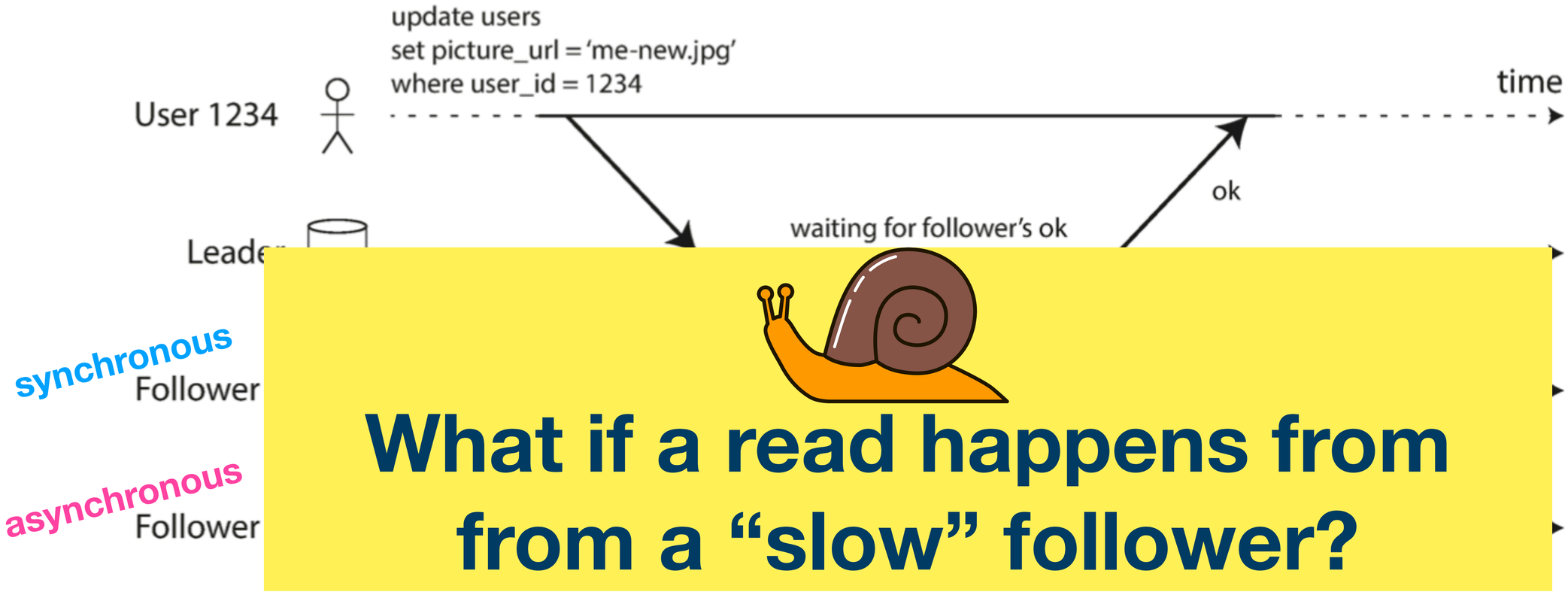


- **Asynchronous**:
  - Higher write throughput
  - Weak durability (a write may be lost)

Usually one follower is synchronous and the rest asynchronous (semi-synchronous)

Synchronous vs asynchronous replication



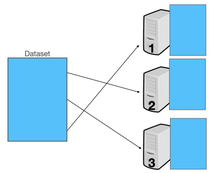


What if a read happens from from a "slow" follower?

- **Asynchronous**:
  - Higher write throughput
  - Weak durability (a write may be lost)

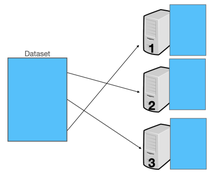
Usually one follower is synchronous and the rest asynchronous (semi-synchronous)

Replication lag

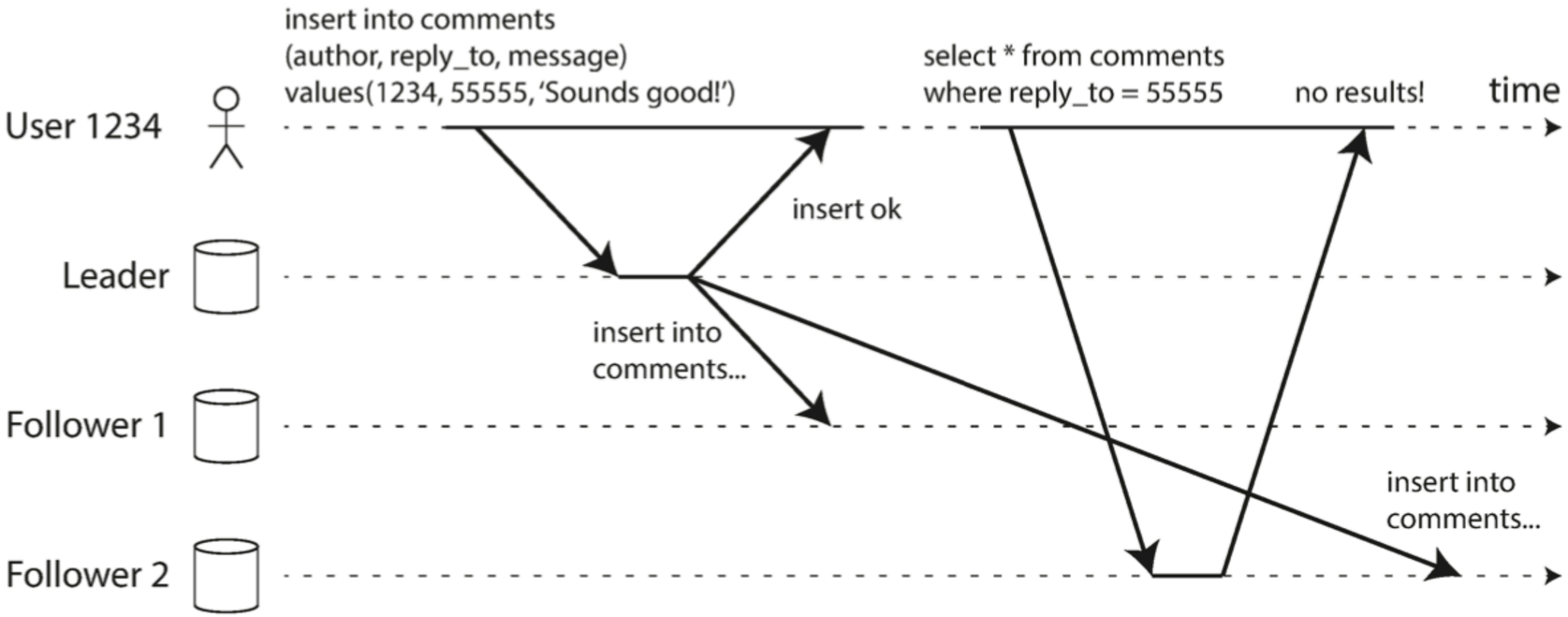


- Asynchronous follower that has fallen behind —> **stale** results —> **inconsistency**
- **Eventual consistency**: database will be consistent at some time
- Stricter consistency models exist

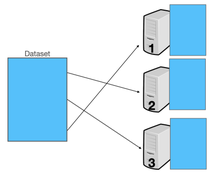
Read-your-own-writes consistency



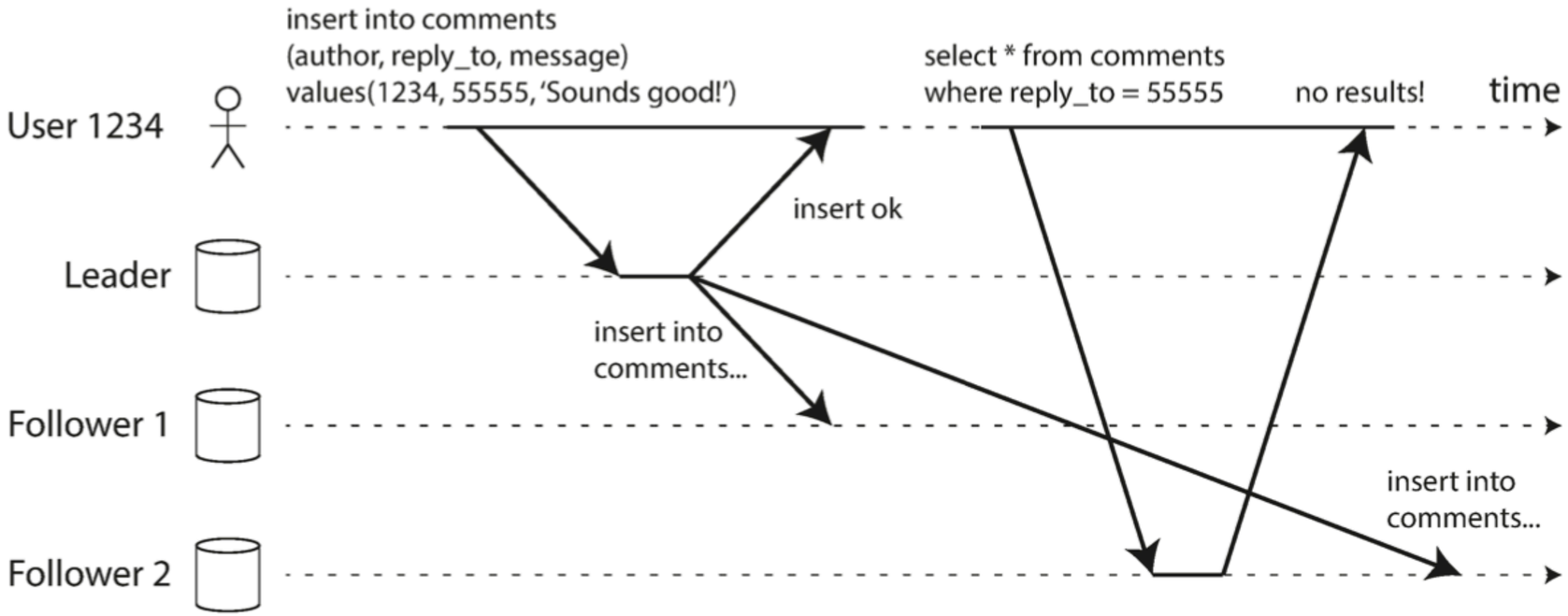
- Problem: A user makes a write, followed by a read from a stale replica



Read-your-own-writes consistency

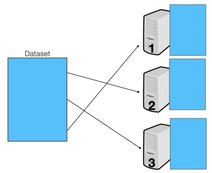


- Problem: A user makes a write, followed by a read from a stale replica

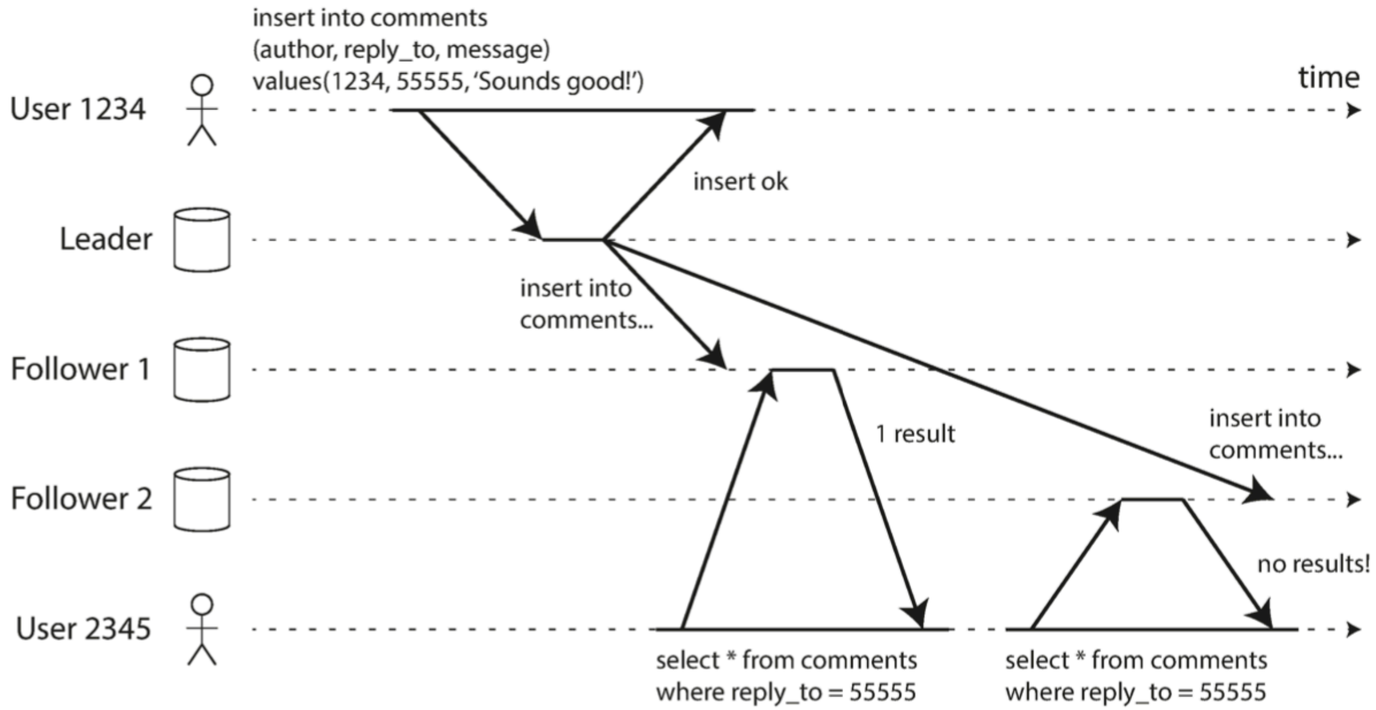


- Possible solution: Read from leader
- Other?

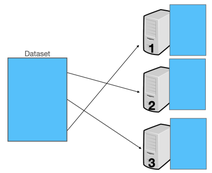
Monotonic reads consistency



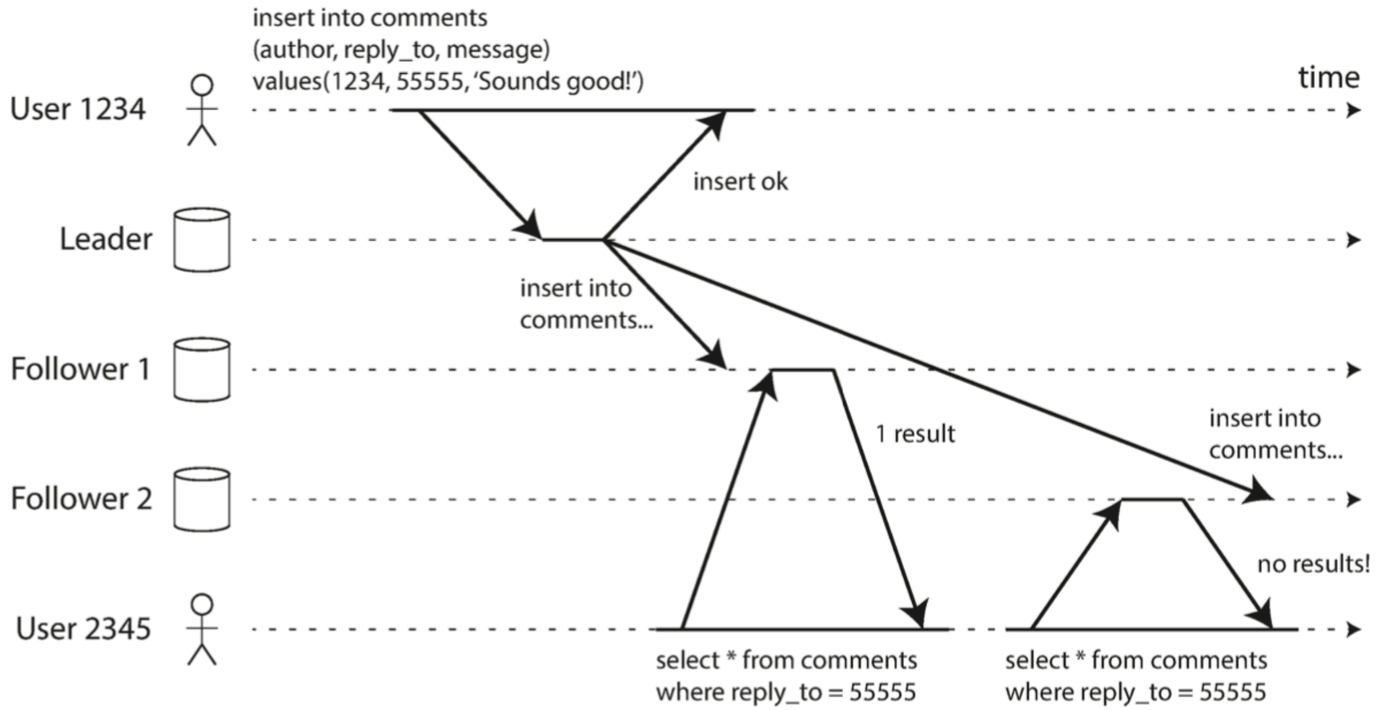
- Problem: A user first reads from a fresh replica, then from a stale replica



Monotonic reads consistency



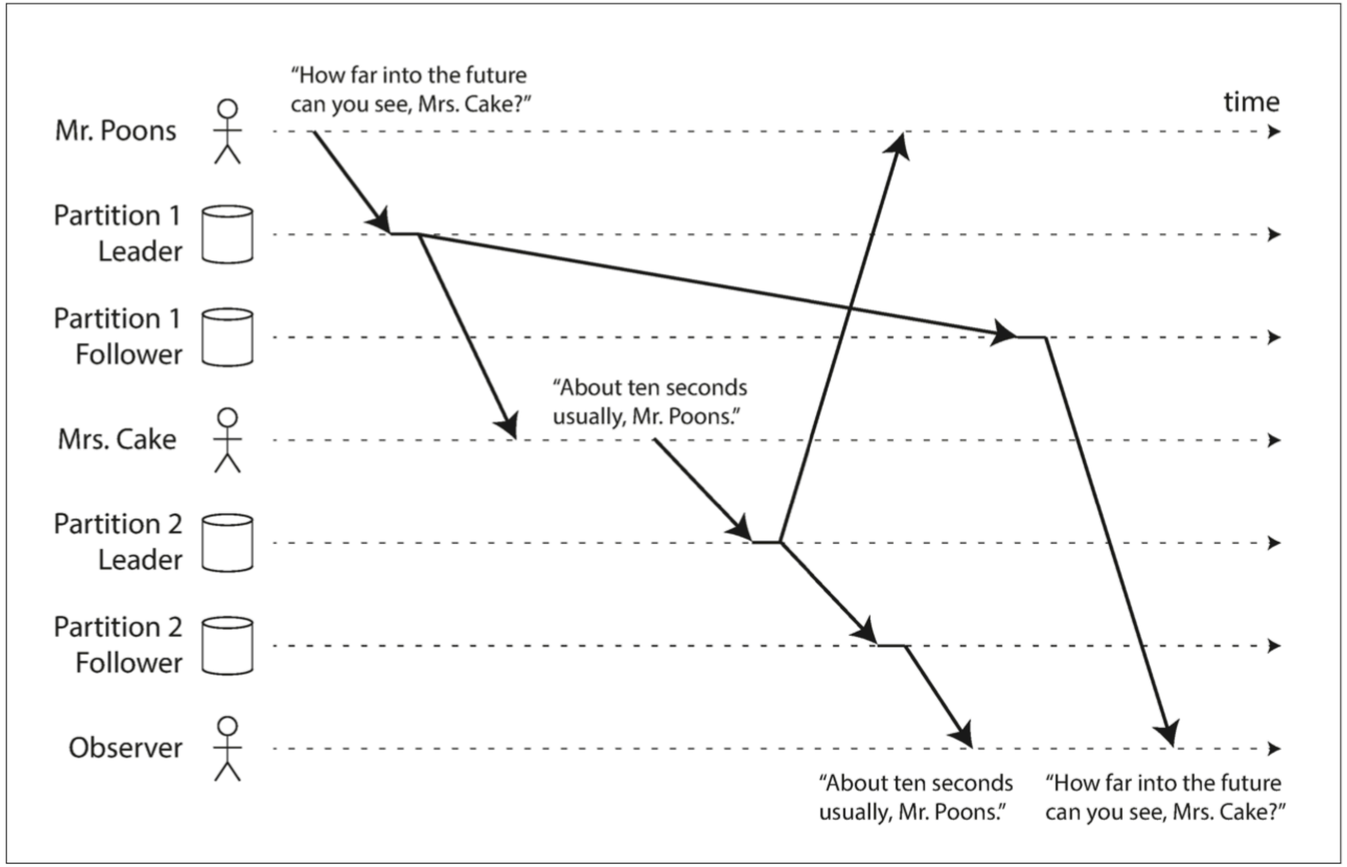
- Problem: A user first reads from a fresh replica, then from a stale replica



- Possible solution: Read from the same replica

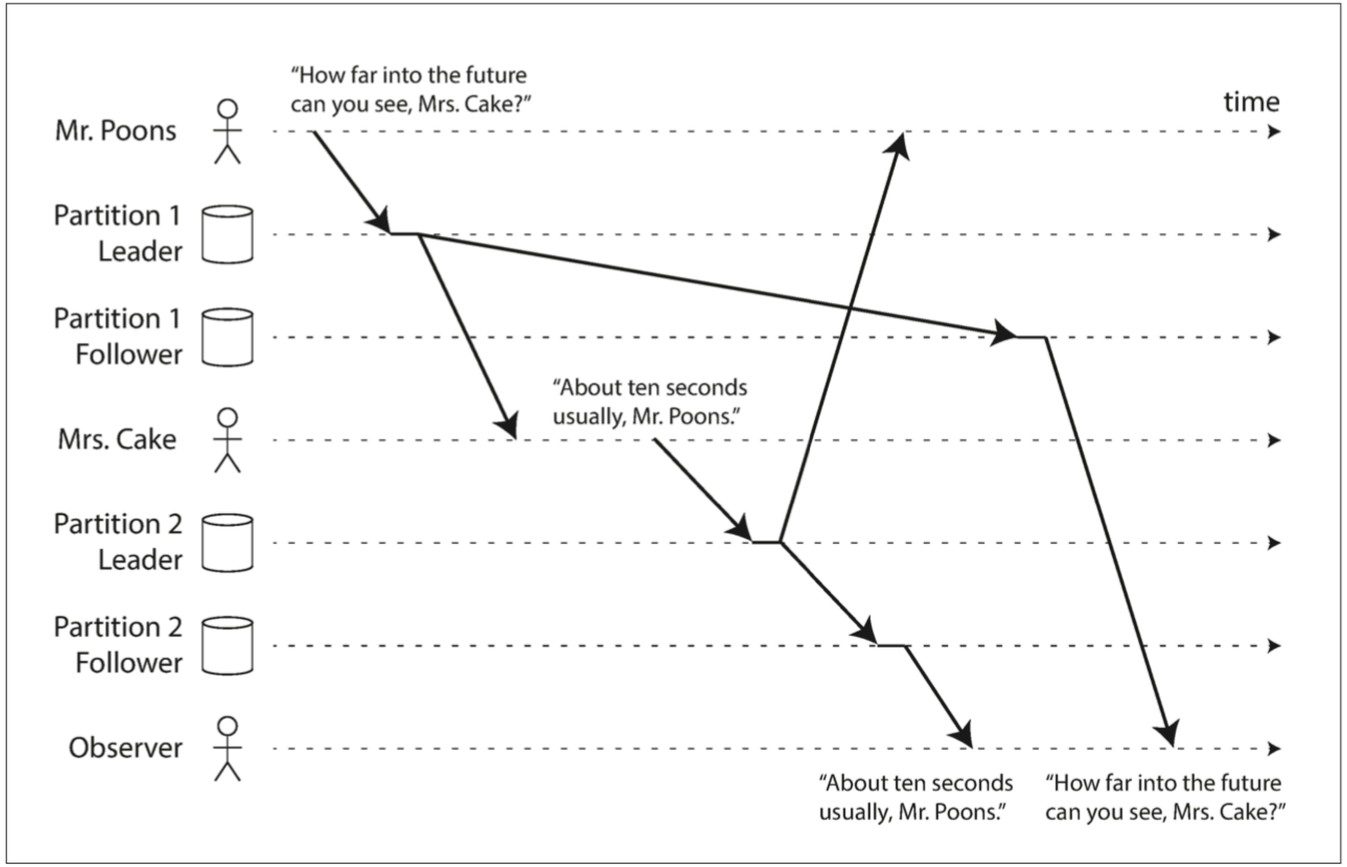
Consistent Prefix Reads

- Problem: The order of events seems twisted



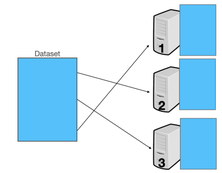
Consistent Prefix Reads

- Problem: The order of events seems twisted

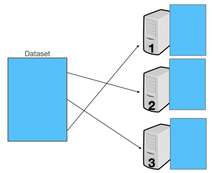


- Possible solution: Write writes that are related to the same follower

Approaches to replication

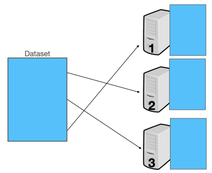


Approaches to replication

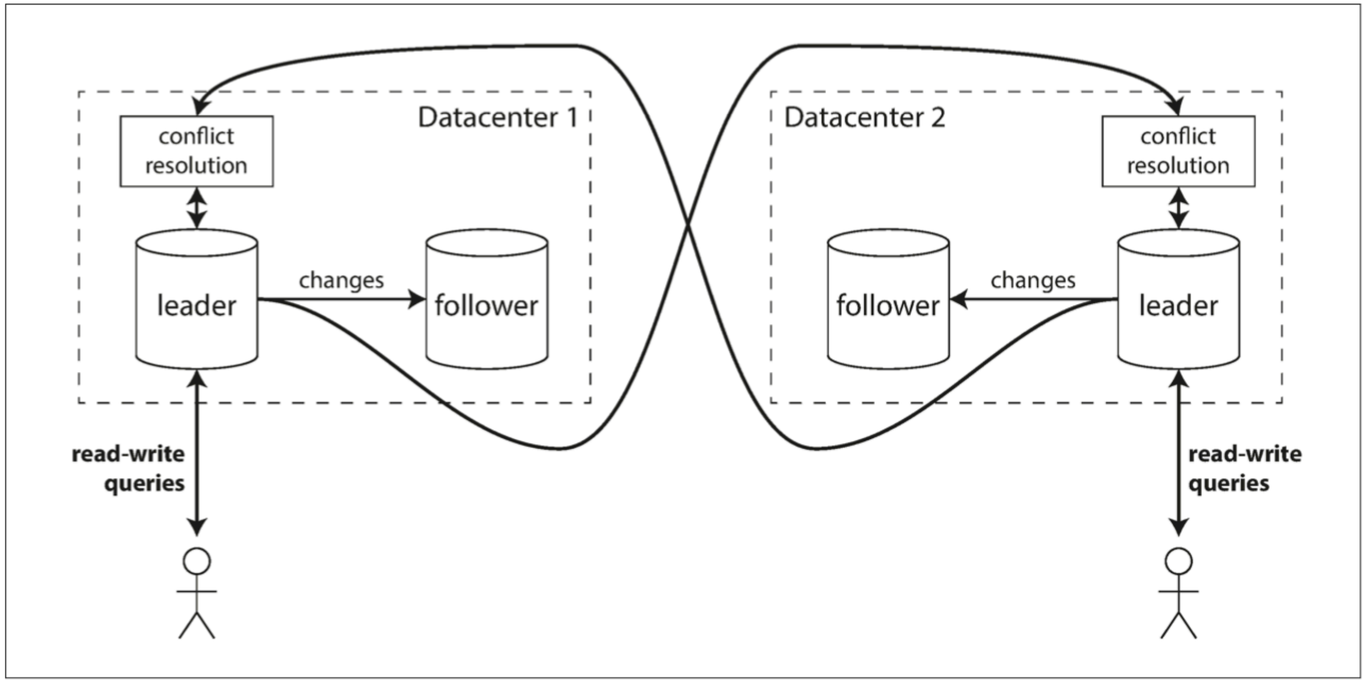


- Single-leader replication
  - Writes to the leader
  - Reads from any replica

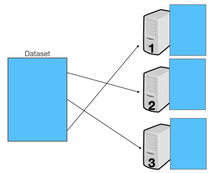
Approaches to replication



- Single-leader replication
  - Writes to the leader
  - Reads from any replica
- Multi-leader replication
  - Writes to any of the multiple leaders
  - Reads from any replica



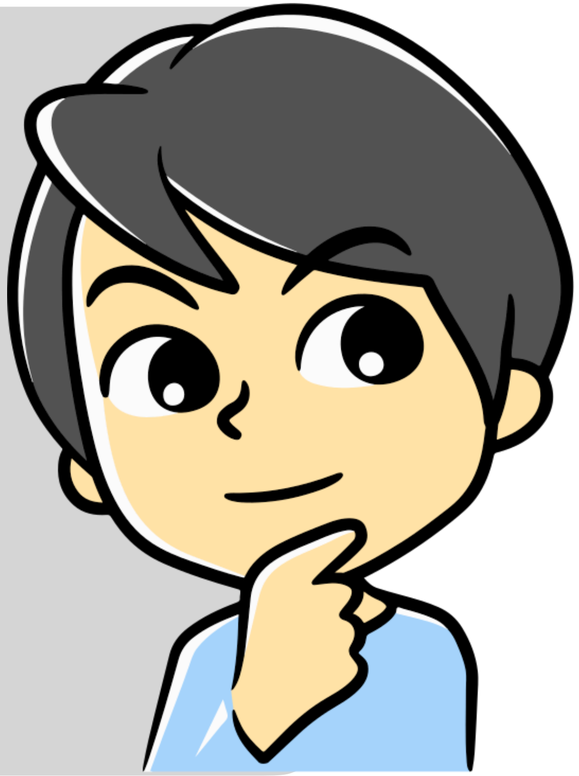
Approaches to replication



- Single-leader replication
  - Writes to the leader
  - Reads from any replica
- Multi-leader replication
  - Writes to any of the multiple leaders
  - Reads from any replica
- Leaderless replication
  - Write to several nodes
  - Read from several nodes in parallel to update stale data

Reflection time

- Would it be possible to **combine** partitioning and replication?
  - If yes, how and what would be the reason?
  - If not, why?



Distributed storage summary

- **Partitioning**: split data across different nodes, preferably evenly (i.e., avoid hotspots)
  - Main approaches: range partitioning or hash partitioning
  - Rebalancing: fixed, dynamic, based on # nodes

- **Replication**: create replicas (copies) of data and send them to all nodes
  - Most common strategy: leader-follower
  - Synchronous vs. asynchronous
  - Keep in mind replication lag: different consistency models

In the next lecture…

- Distributed file systems
  - GFS, HDFS
- Distributed computation/query processing
  - Mapreduce, Hadoop

Readings

- Chapters 1, 5 and 6 from “Designing data-intensive applications” book