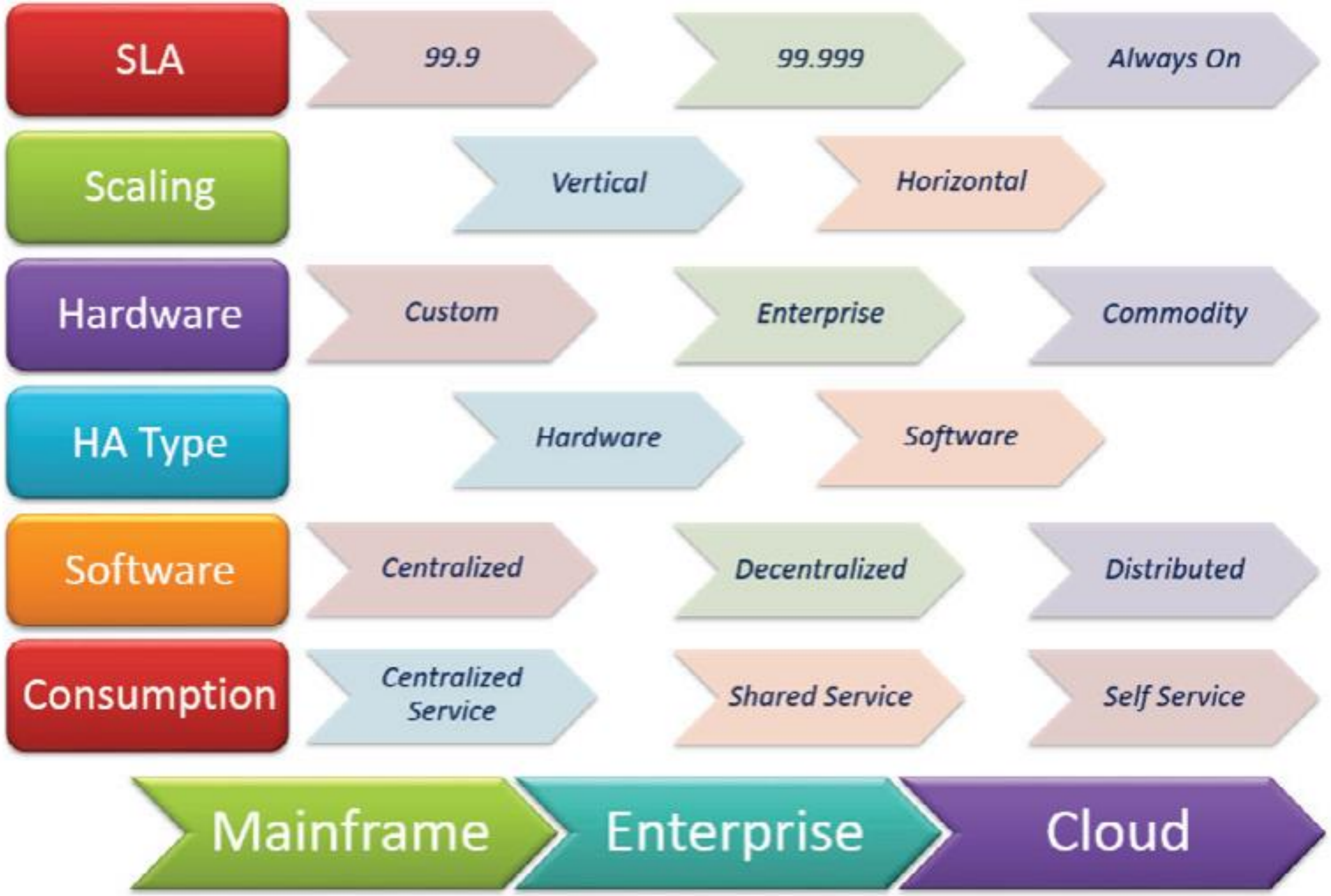
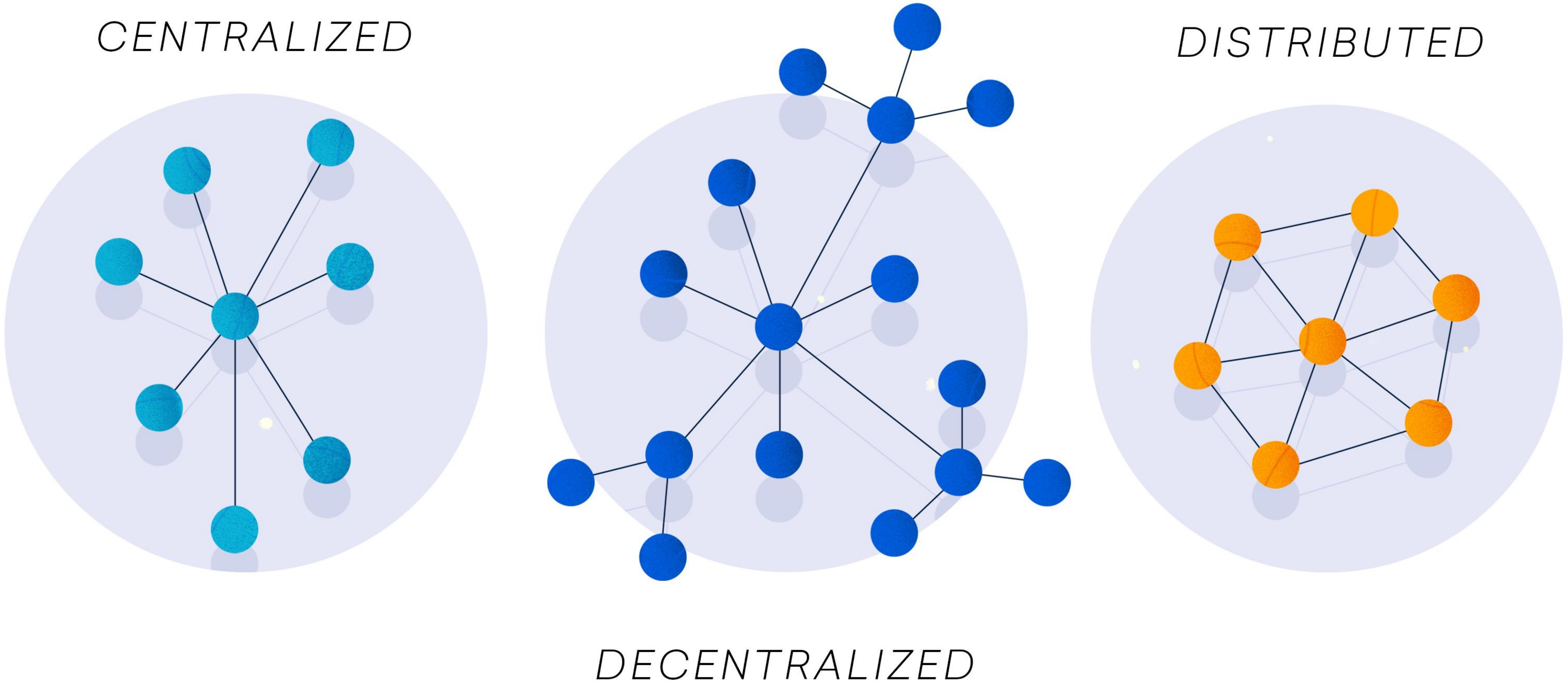
Parralel & Distributed Computing

Trong-Hop Do

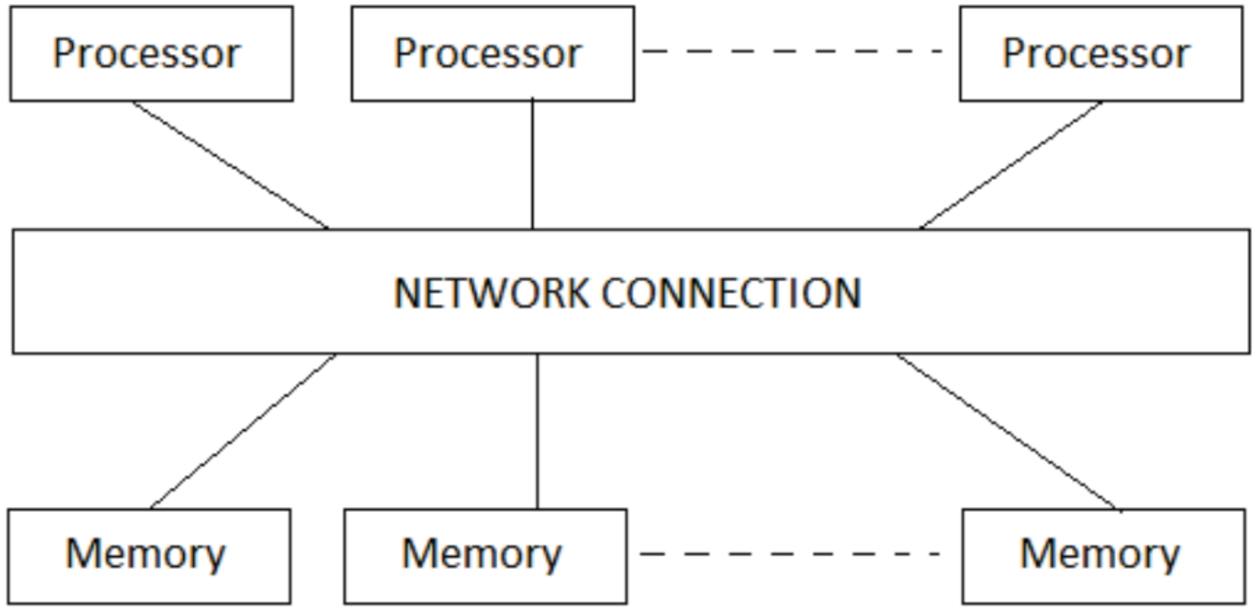
Evolution of Computing Models



	Mainframe	Enterprise	Cloud
SLA	*99.9*	*99.999*	*Always On*
Scaling	*Vertical*	*Horizontal*	
Hardware	*Custom*	*Enterprise*	*Commodity*
HA Type	*Hardware*	*Software*	
Software	*Centralized*	*Decentralized*	*Distributed*
Consumption	*Centralized Service*	*Shared Service*	*Self Service*


CENTRALIZED
DECENTRALIZED
DISTRIBUTED

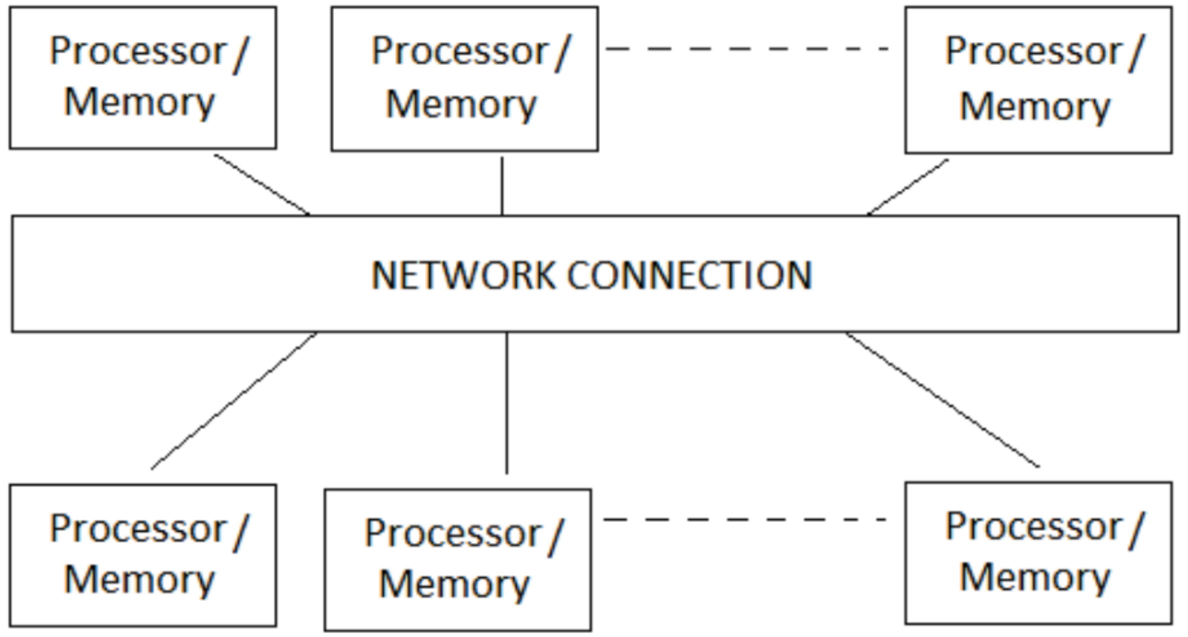
Parallel computing



Uniform Memory Access (UMA) Parallel System

- Multiple processor having direct memory access to the shared memory that can form a common address space.
- Access latency (processing time) for accessing any particular location of a memory from a particular processor is the same
- In general, the processors are homogeneous and are installed within the same container of the shared memory

Parallel computing



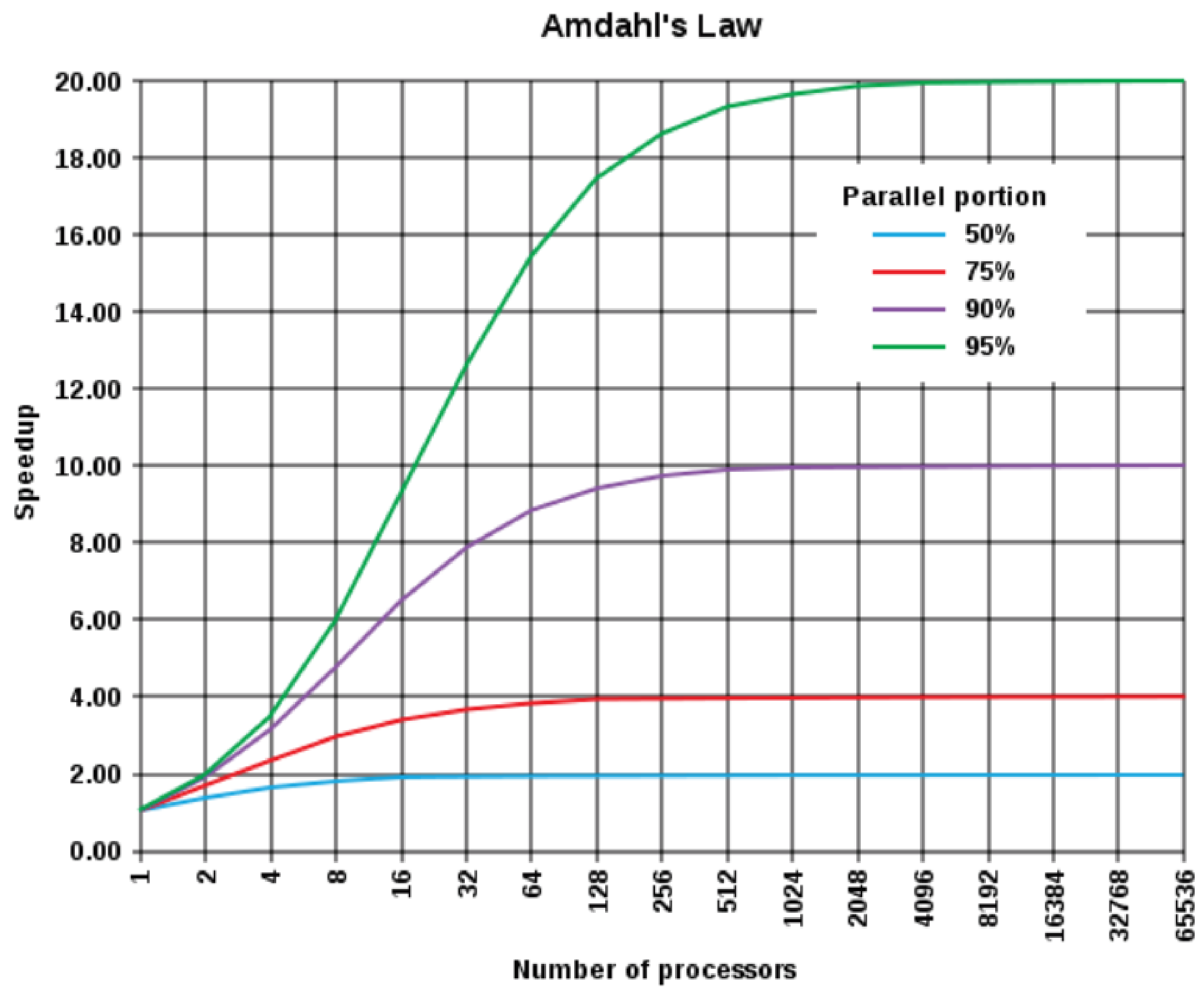
Non-uniform memory access (NUMA) Parallel System

- Multiple processors configured without having a direct accessibility to the shared memory
- Computers are not expected to contain a common clock
- Accessing different memory locations in a shared memory across different processors shows different latency times.

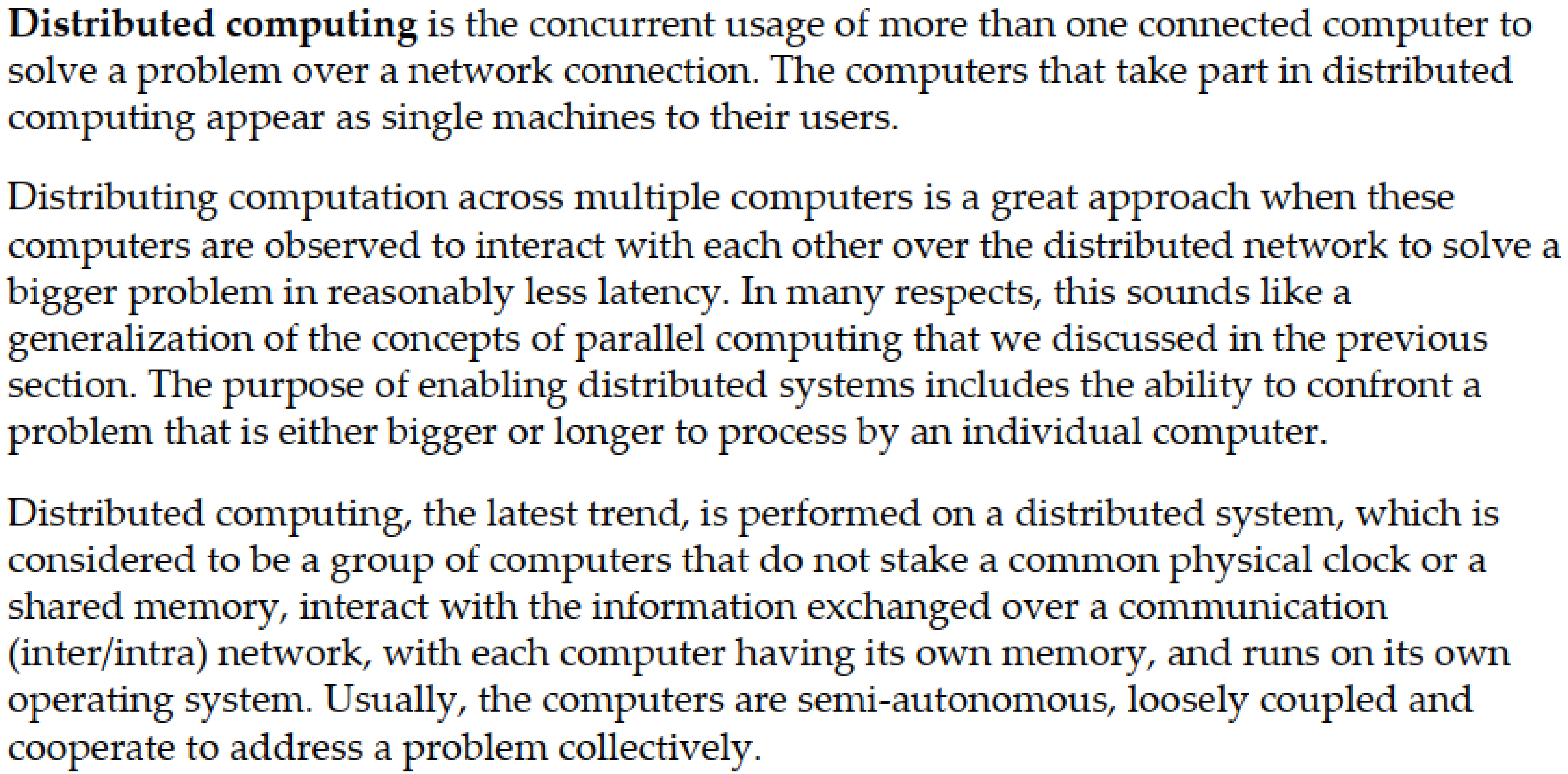
Parallel computing

$$\boxed{S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}}$$

Amdahl's Law



Distributed computing

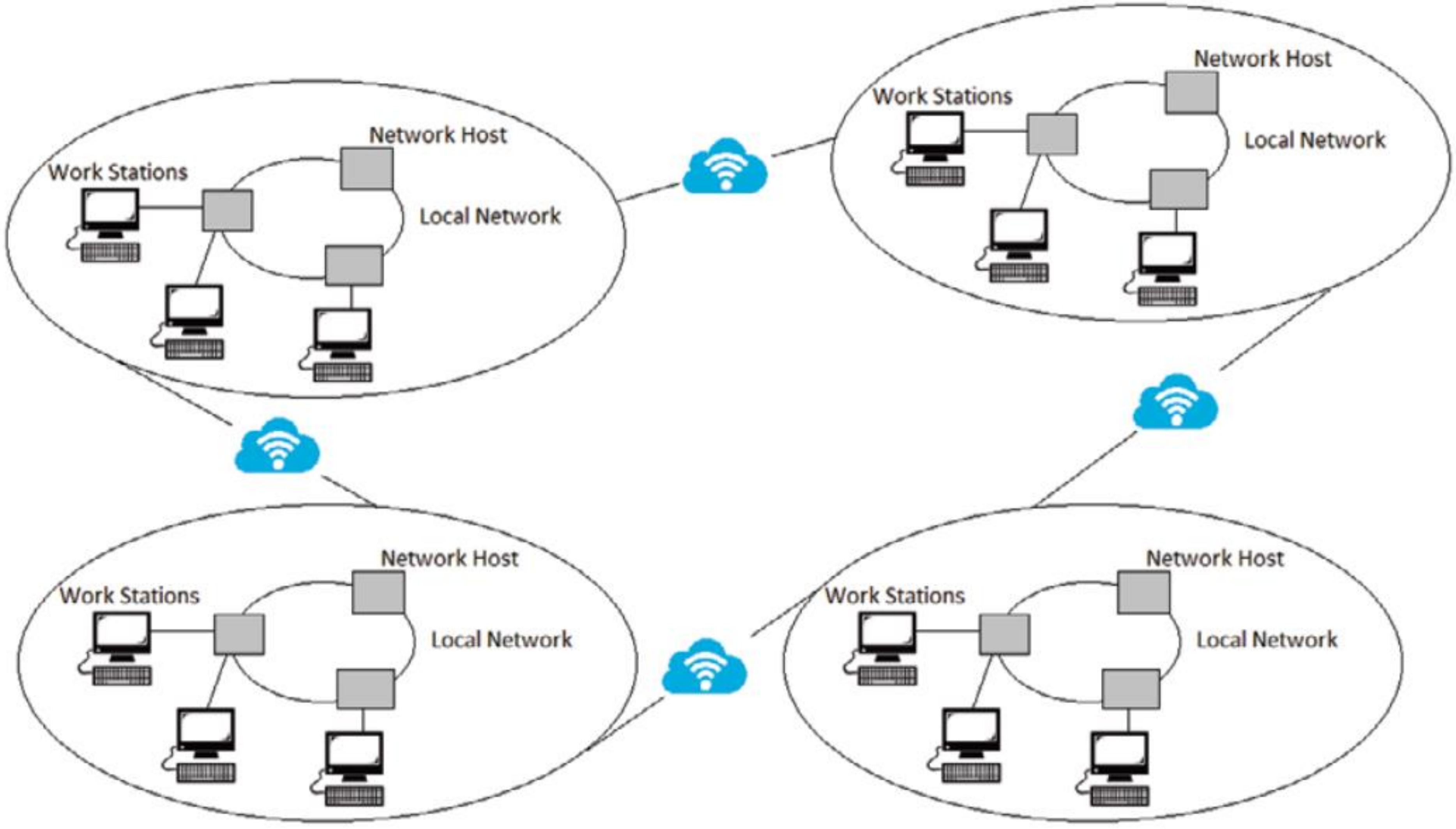


Distributed computing is the concurrent usage of more than one connected computer to solve a problem over a network connection. The computers that take part in distributed computing appear as single machines to their users.

Distributing computation across multiple computers is a great approach when these computers are observed to interact with each other over the distributed network to solve a bigger problem in reasonably less latency. In many respects, this sounds like a generalization of the concepts of parallel computing that we discussed in the previous section. The purpose of enabling distributed systems includes the ability to confront a problem that is either bigger or longer to process by an individual computer.

Distributed computing, the latest trend, is performed on a distributed system, which is considered to be a group of computers that do not stake a common physical clock or a shared memory, interact with the information exchanged over a communication (inter/intra) network, with each computer having its own memory, and runs on its own operating system. Usually, the computers are semi-autonomous, loosely coupled and cooperate to address a problem collectively.

Distributed computing



Distributed computing

Distributed computing

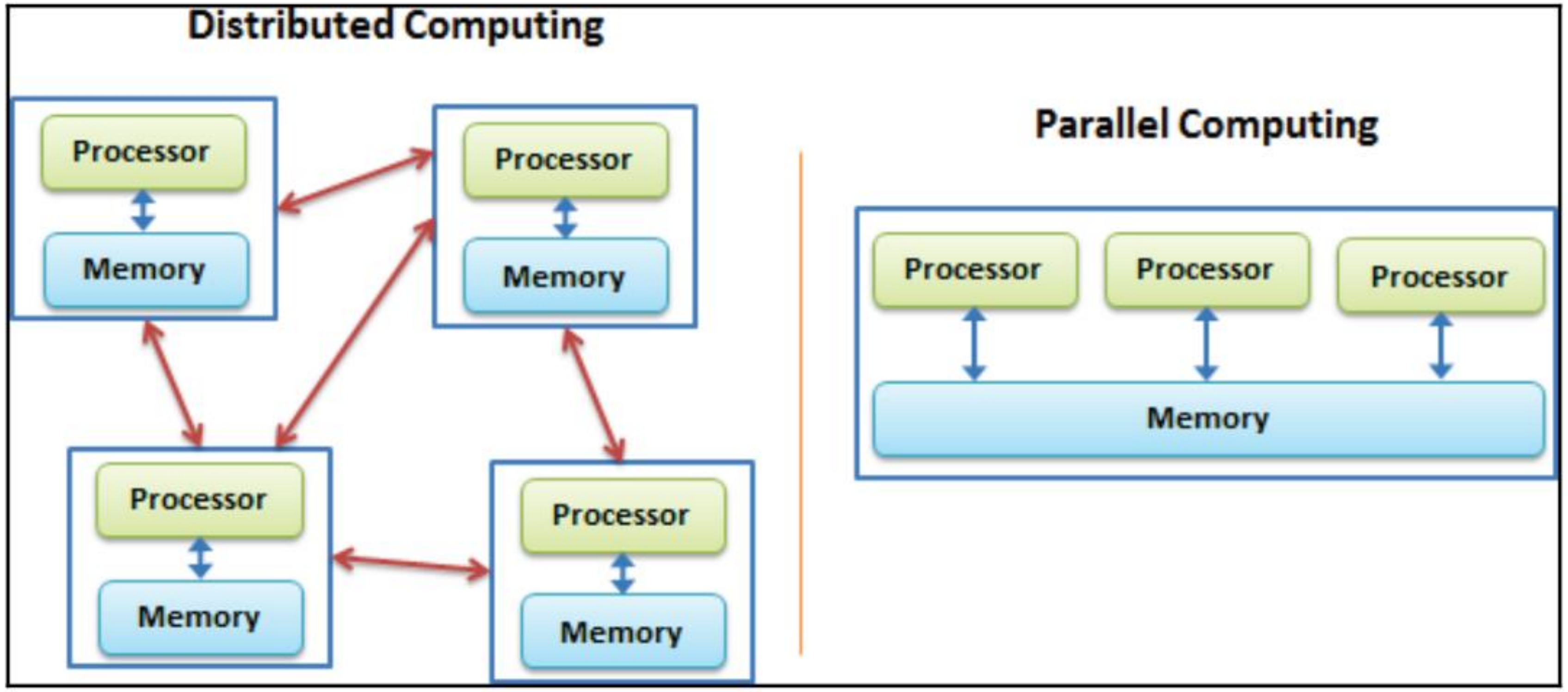
Google

amazon

DreamWorks
Animation SKG

PIXAR

Parallel versus distributed computing



The main difference is that parallel computing allows multiple processors to execute tasks simultaneously while distributed computing divides a single task between multiple computers to achieve a common goal.

Parallel versus distributed computing

- **Number of computers**
  - The number of computers involved is a difference between parallel and distributed computing. Parallel computing occurs in a single computer whereas distributed computing involves multiple computers.
- **Functionality**
  - In parallel computing, multiple processors execute multiple tasks at the same time. However, in distributed computing, multiple computers perform tasks at the same time. Hence, this is another difference between parallel and distributed computing.
- **Memory**
  - Moreover, memory is a major difference between parallel and distributed computing. In parallel computing, the computer can have a shared memory or distributed memory. In distributed computing, each computer has its own memory.

Parallel versus distributed computing

- **Communication**
  - Also, one other difference between parallel and distributed computing is the method of communication. In parallel computing, the processors communicate with each other using a bus. In distributed computing, computers communicate with each other via the network.
- **Usage**
  - Parallel computing helps to increase the performance of the system. In contrast, distributed computing allows scalability, sharing resources and helps to perform computation tasks efficiently. So, this is also a difference between parallel and distributed computing.

The winner in the war between multiprocessing and distributing processing, supercomputing versus network computing seems to be decided.

Why distributed, not just parallel?

- **Scalability**: As distributed systems do not have the problems associated with shared memory, with the increased number of processors, they are obviously regarded as more scalable than parallel systems.
- **Reliability**: The impact of the failure of any single subsystem or a computer on the network of computers defines the reliability of such a connected system. Definitely, distributed systems demonstrate a better aspect in this area compared to the parallel systems.
- **Data sharing**: Data sharing provided by distributed systems is similar to the data sharing provided by distributed databases. Thus, multiple organizations can have distributed systems with the integrated applications for data exchange.
- **Resources sharing**: If there exists an expensive and a special purpose resource or a processor, which cannot be dedicated to each processor in the system, such a resource can be easily shared across distributed systems.

Why distributed, not just parallel?

- **Heterogeneity and modularity**: A system should be flexible enough to accept a new heterogeneous processor to be added into it and one of the processors to be replaced or removed from the system without affecting the overall system processing capability. Distributed systems are observed to be more flexible in this respect.
- **Geographic construction**: The geographic placement of different subsystems of an application may be inherently placed as distributed. Local processing may be forced by the low communication bandwidth more specifically within a wireless network.
- **Economic**: With the evolution of modern computers, high-bandwidth networks and workstations are available at low cost, which also favors distributed computing for economic reasons.

Design considerations for distributed systems

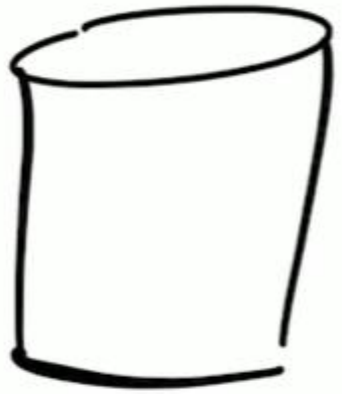
- No global clock
- Geographical distribution
- No shared memory
- Independence and heterogeneity
- Fail-over mechanism
- Security concerns

Three topics

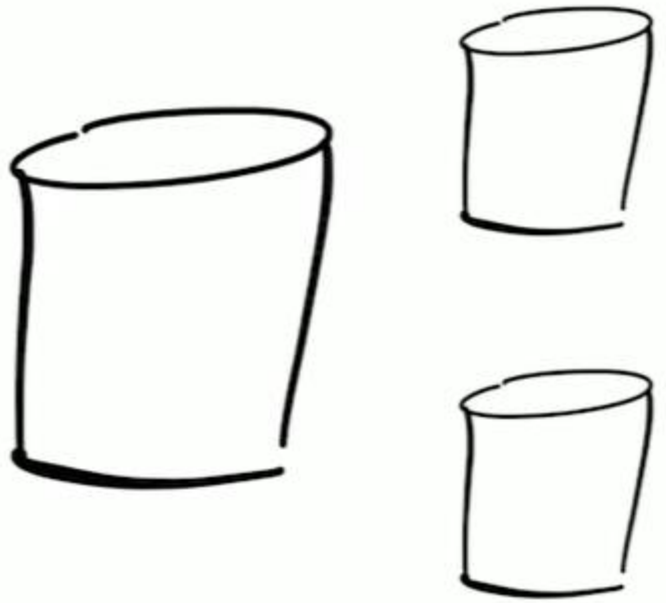
- Storage
- Computation
- Messeging

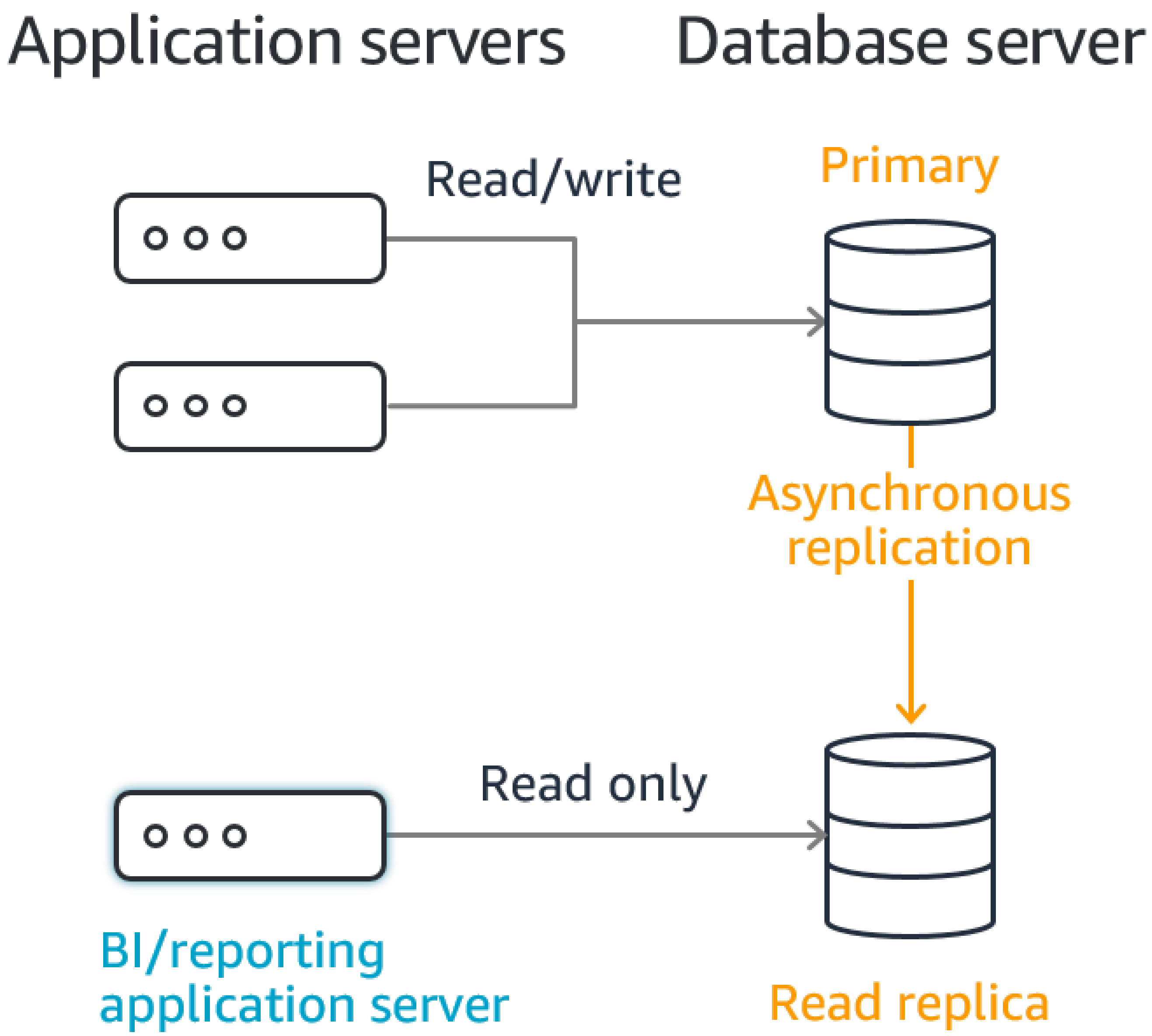
Storage

Single-Master Storage



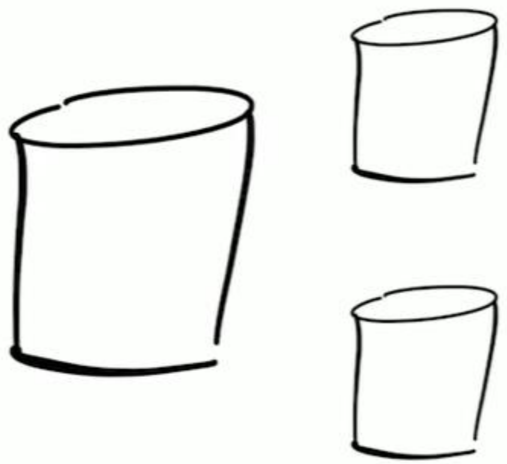
Read Replication

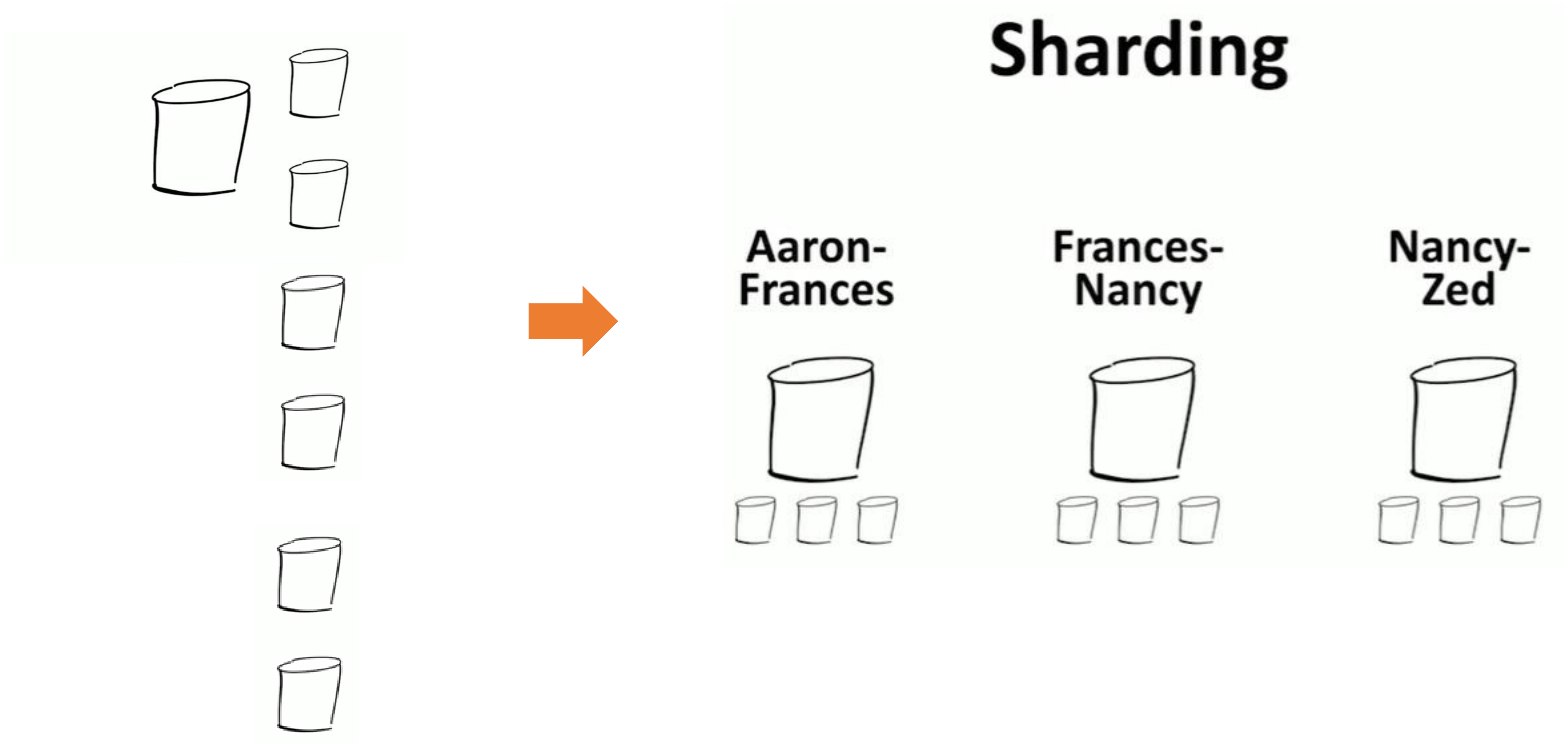



Application servers
Database server
Read/write
Primary
Asynchronous replication
Read only
BI/reporting application server
Read replica

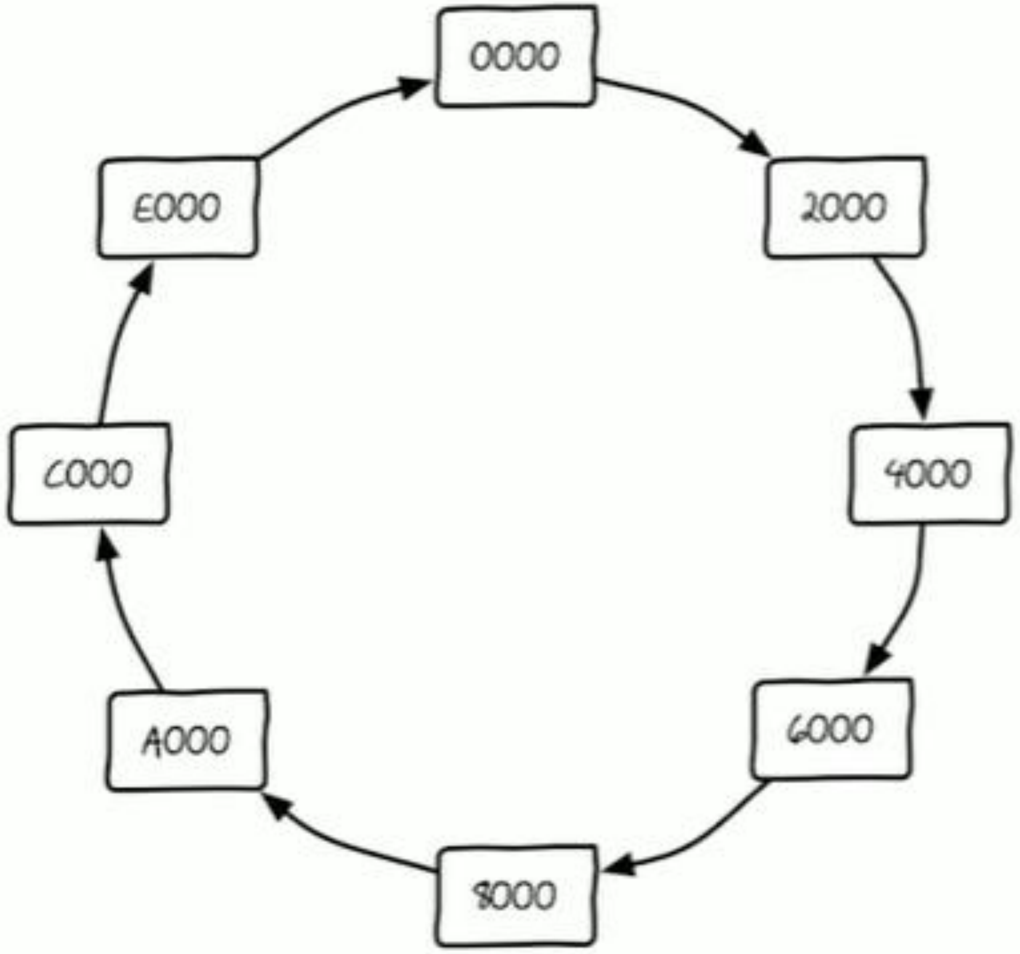
Single-Master Storage

Read Replication

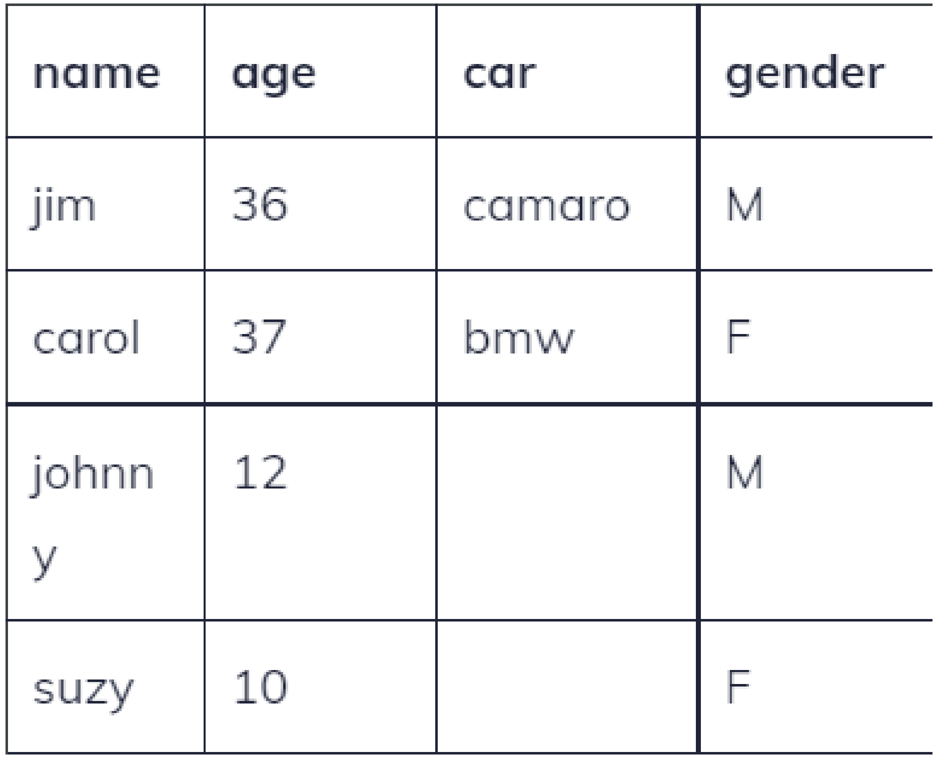



Sharding
Aaron-
Frances
Frances-
Nancy
Nancy-
Zed

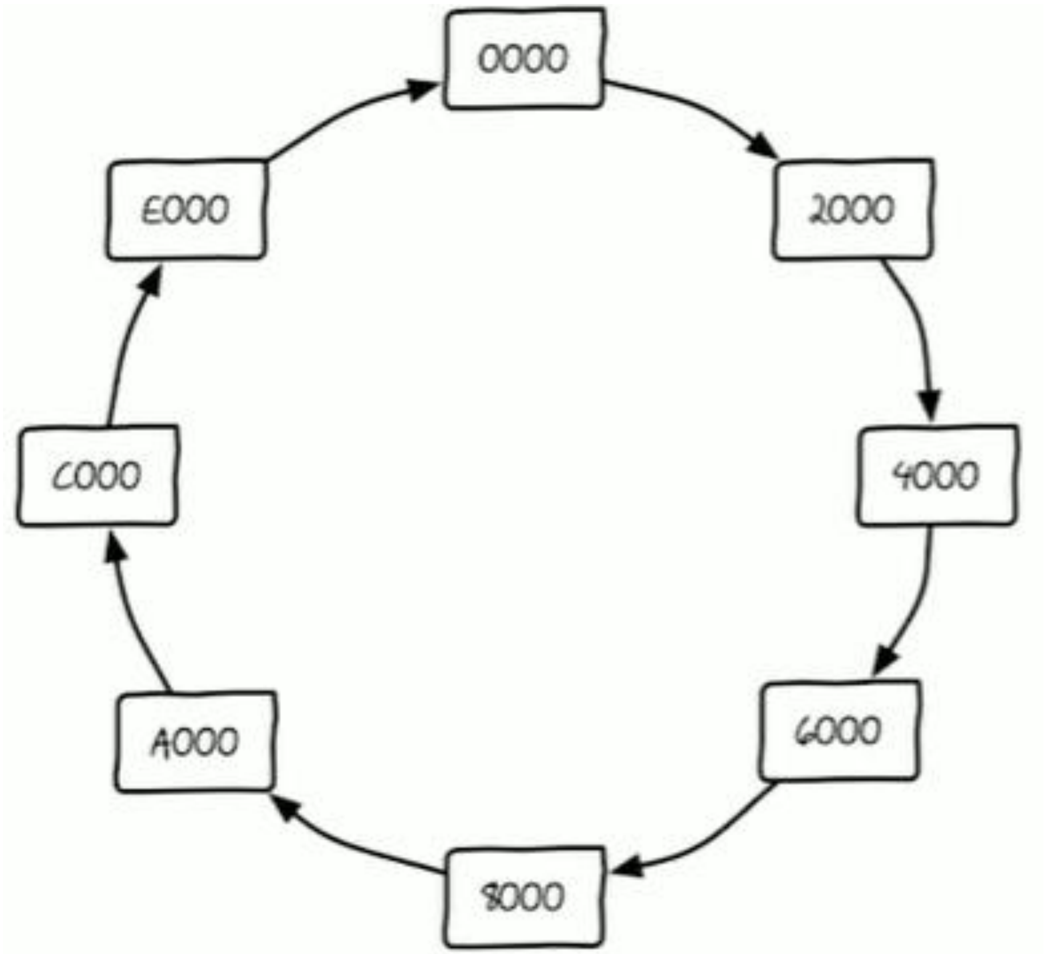
Consistent Hashing



Write

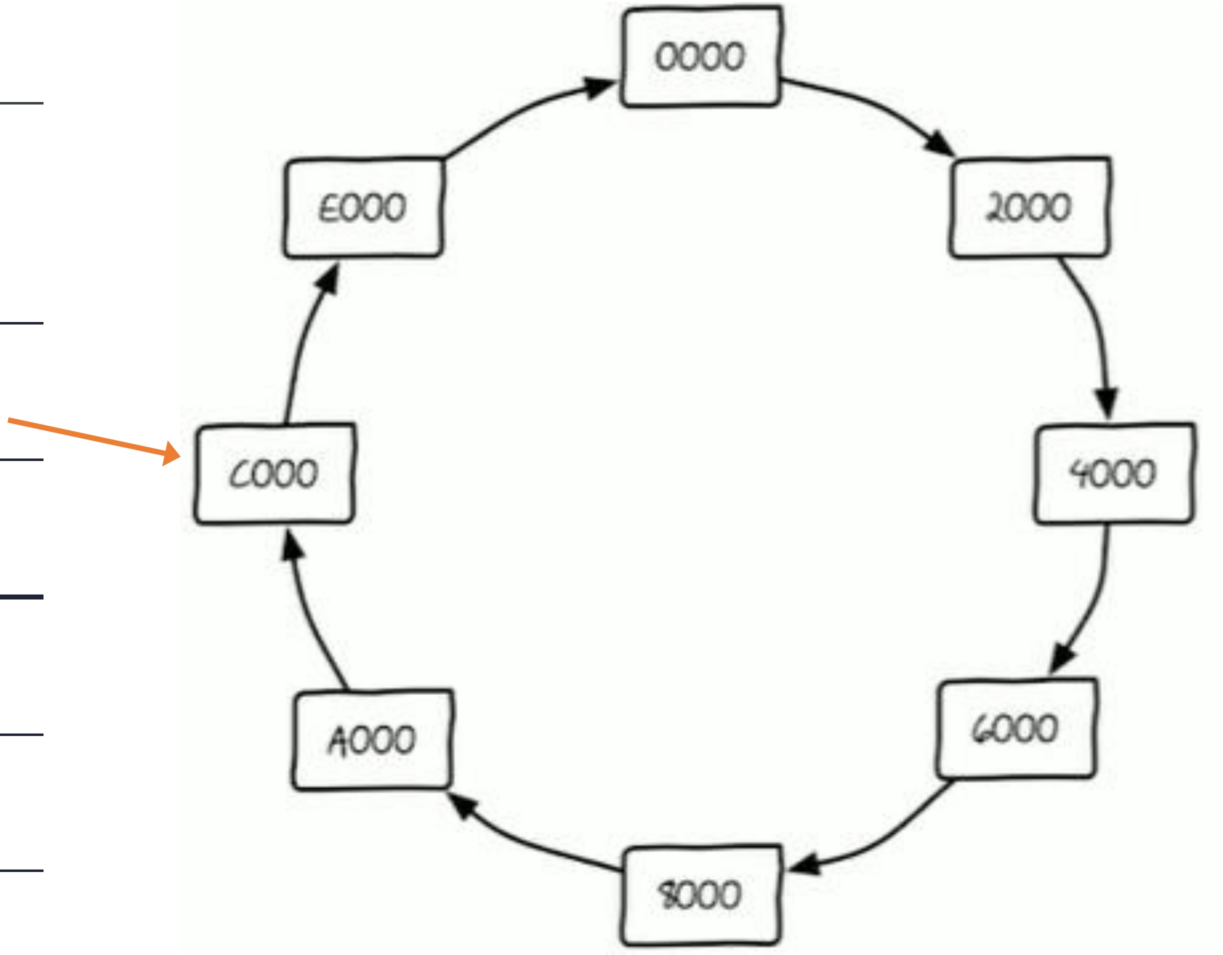


name	age	car	gender
jim	36	camaro	M
carol	37	bmw	F
johnny	12		M
suzy	10		F

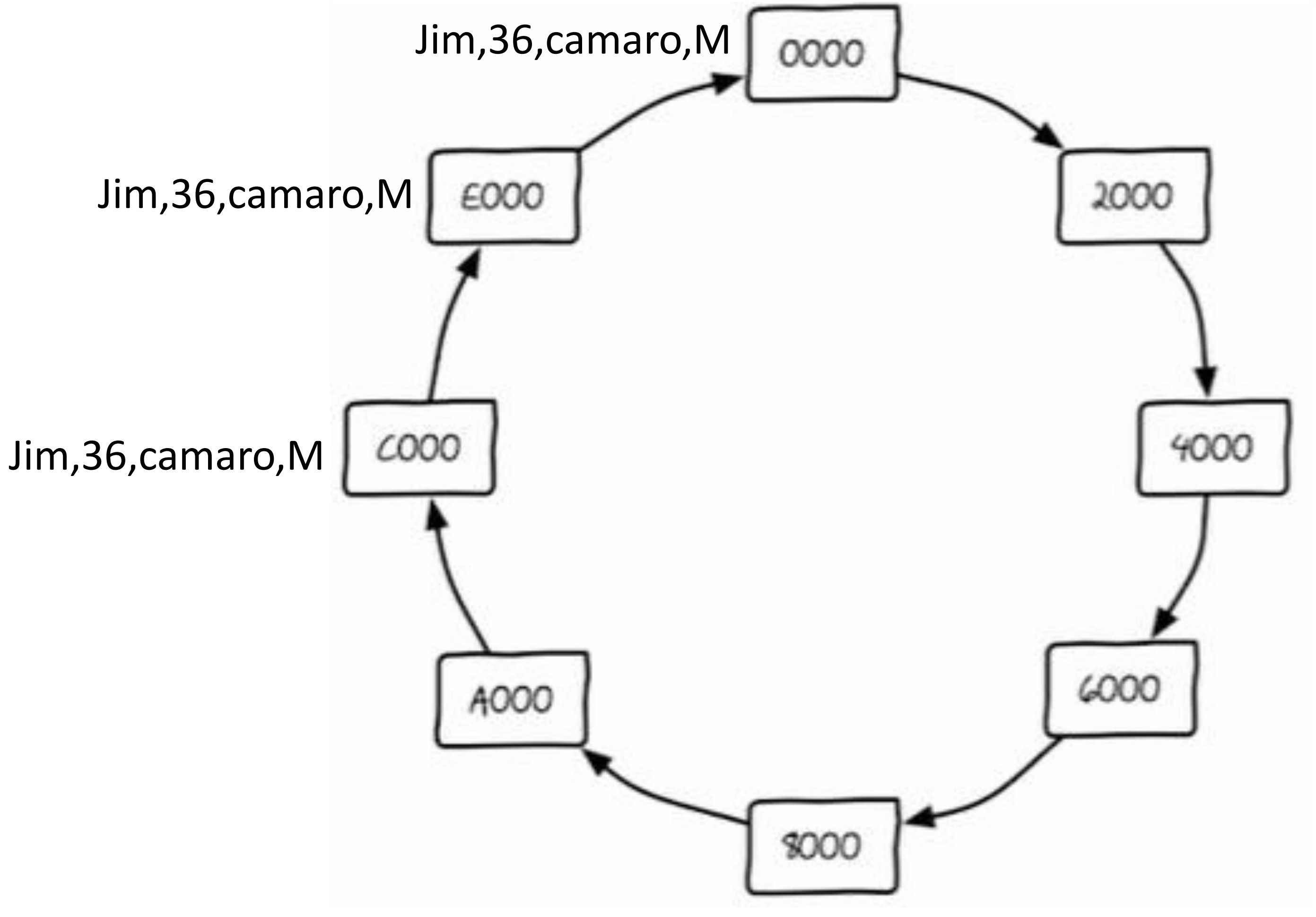


Write

Partition key	Murmur3 hash value
jim	-2245462676723223822
carol	7723358927203680754
johnny	-6723372854036780875
suzy	1168604627387940318

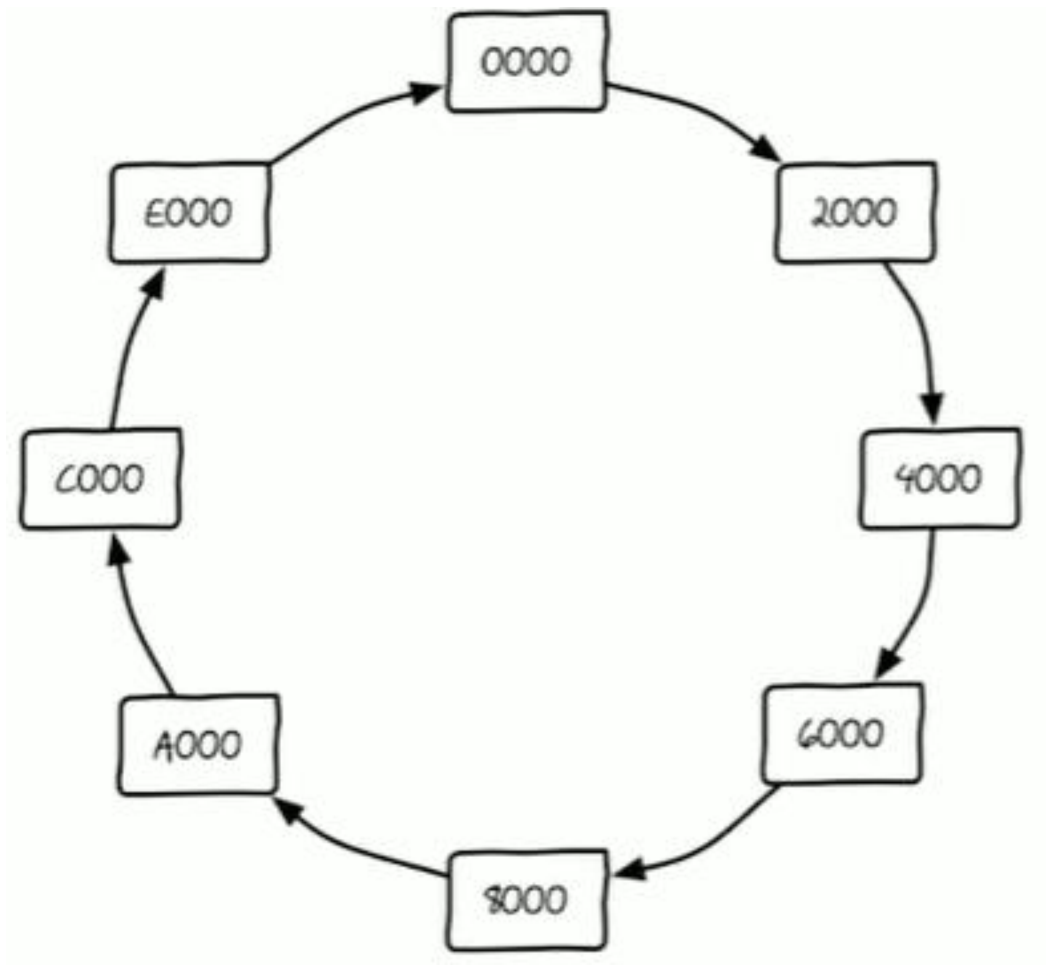


Write


Jim,36,camaro,M
0000
Jim,36,camaro,M
E000
2000
Jim,36,camaro,M
C000
4000
A000
6000
8000

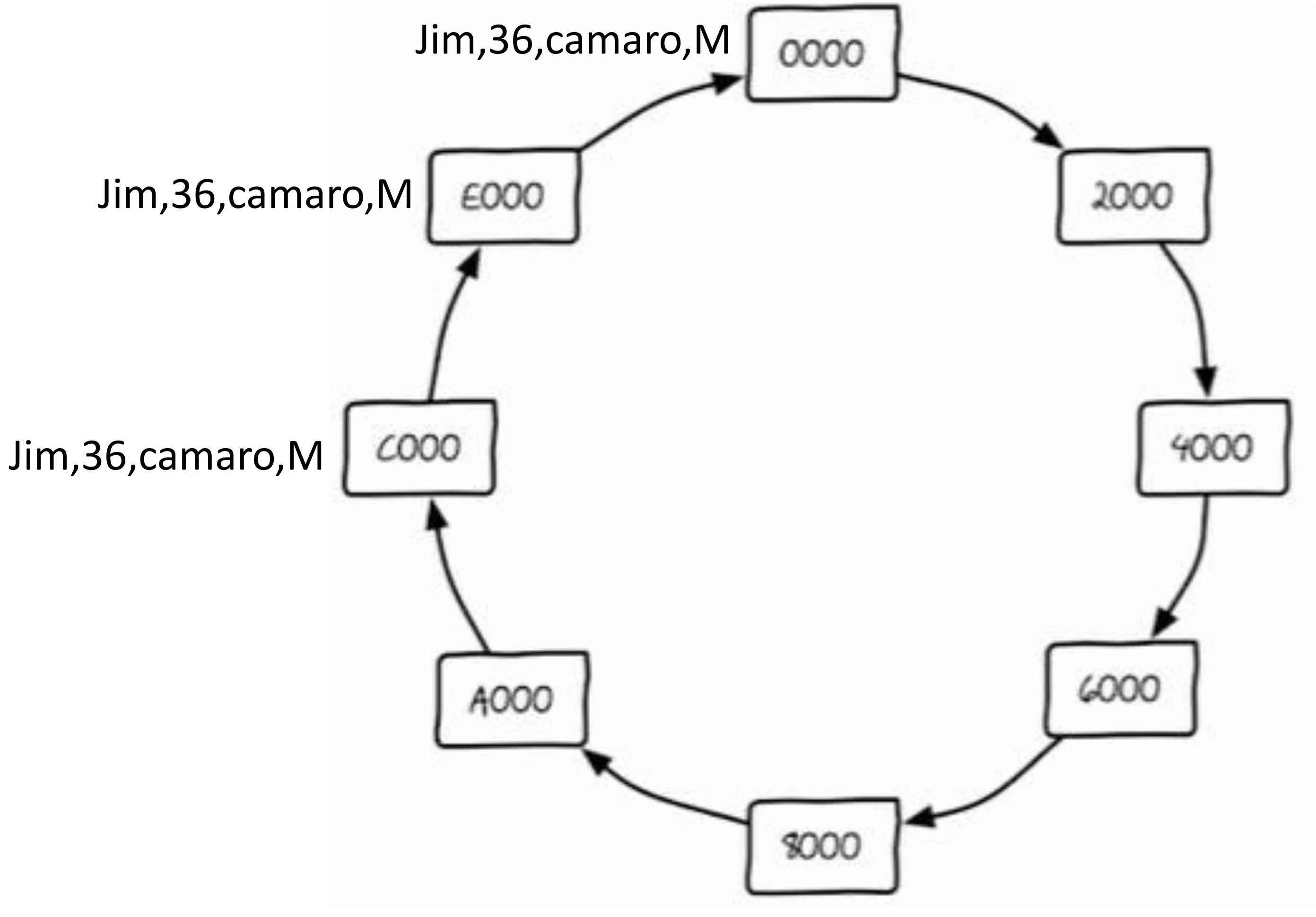
Read

jim → -2245462676723223822 →



Consistency

$R+W>N$



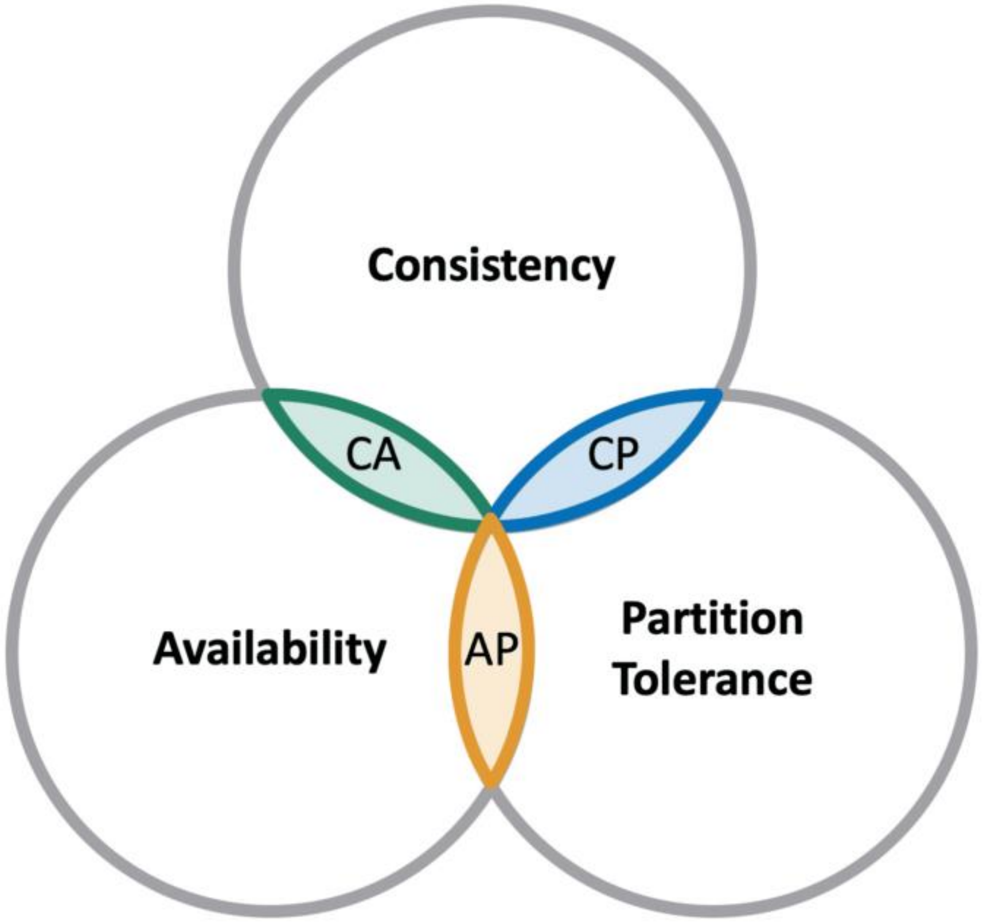
CAP theorem

Ex: Communication between X and Y break, so they can't sync updates. At this point you can either:

AP: Allow the nodes to get out of sync (giving up consistency),

or

CP: Consider the cluster to be "down" (giving up availability)



Distributed storage

HDFS

amazon S3

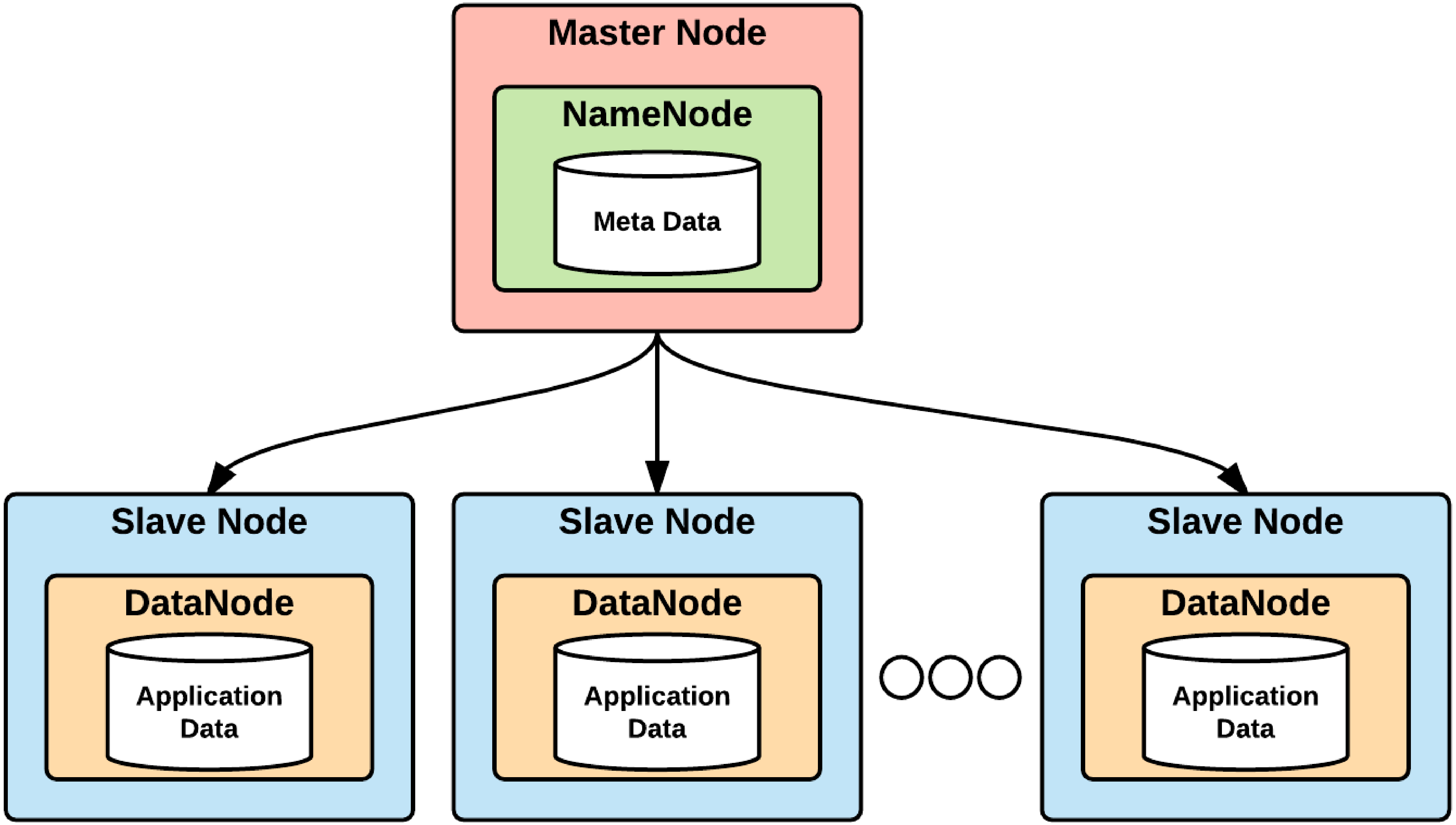
APACHE HBASE

BLOCKCHAIN

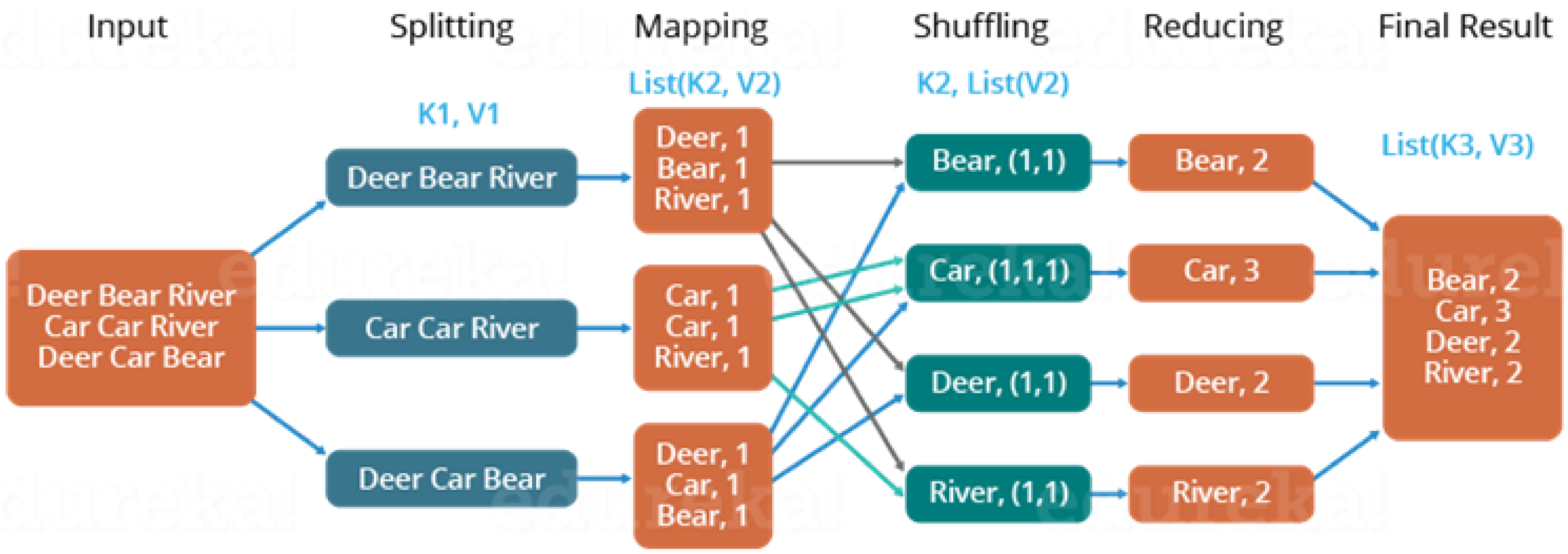
Computation

hadoop

Map Reduce

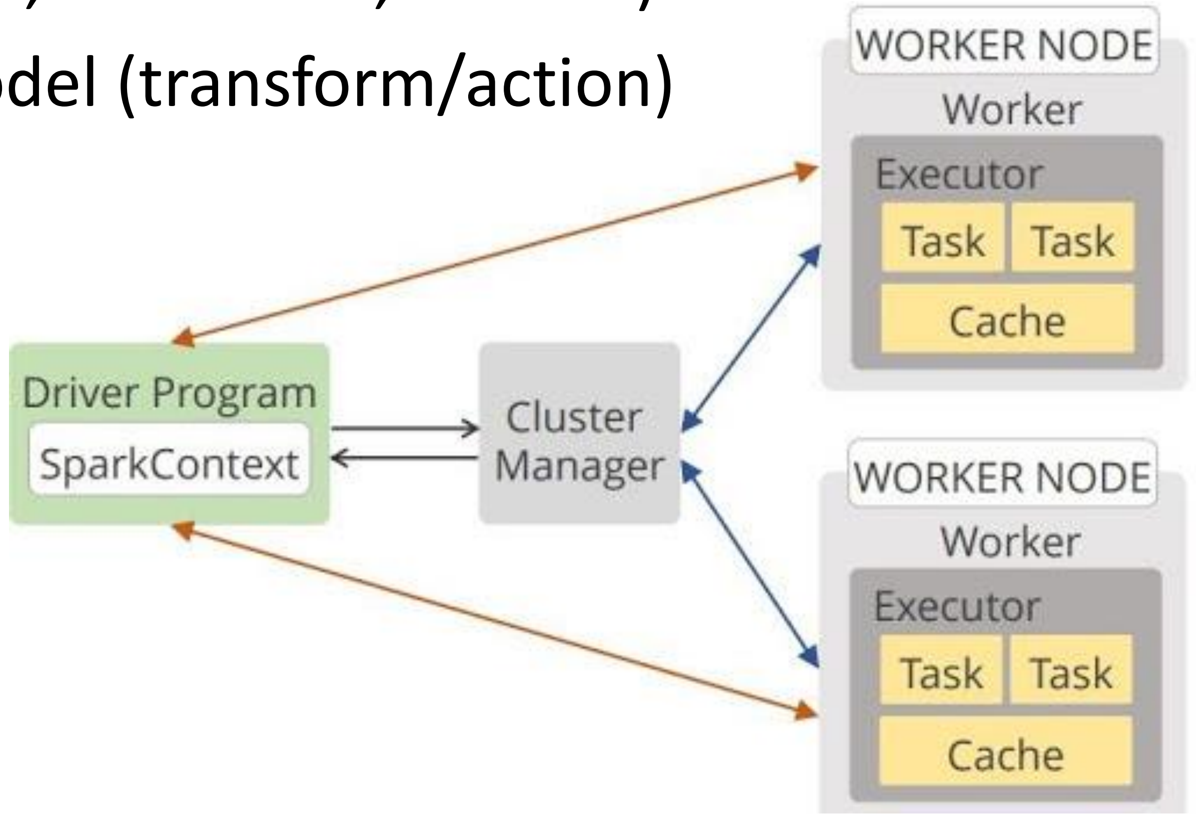

Master Node
NameNode
Meta Data
Slave Node
DataNode
Application Data
Slave Node
DataNode
Application Data
Slave Node
DataNode
Application Data

The Overall MapReduce Word Count Process



APACHE Spark™

- Scatter/gather paradigm (similar to MapReduce)
- More general data model (RDDs, dataframe, dataset)
- More general programming model (transform/action)
- Storage agnostic

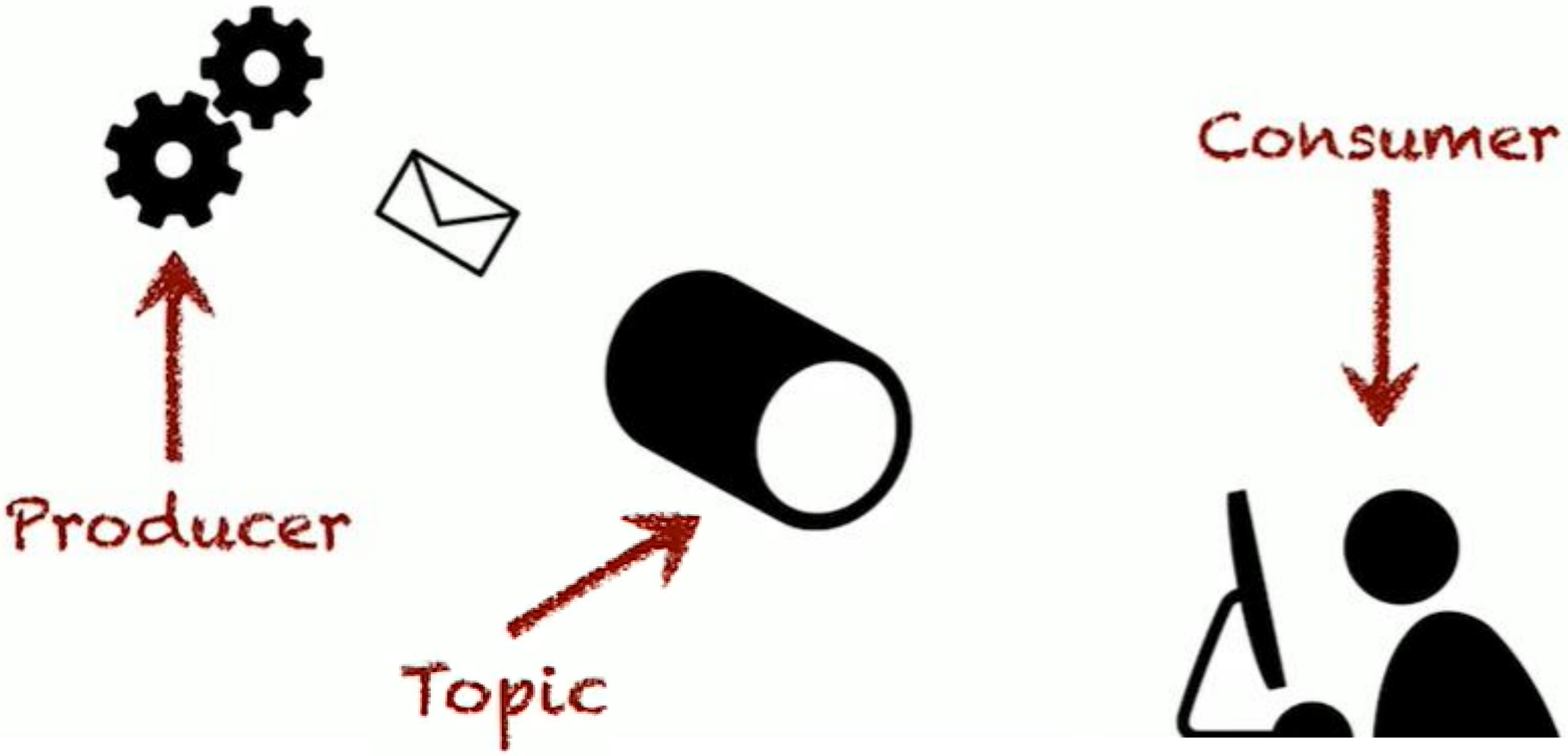


Messeging

Messaging

- Means of loosely coupling subsystems
- Messages consumed by *subscribers*
- Created by one or more *producers*
- Organized into *topics*
- Processed by *brokers*
- Usually persistent over the short term

Messaging



Single server messaging

- Can do
  - Guarantee messages are delivered exactly once
  - Guarantee messages are in ordered

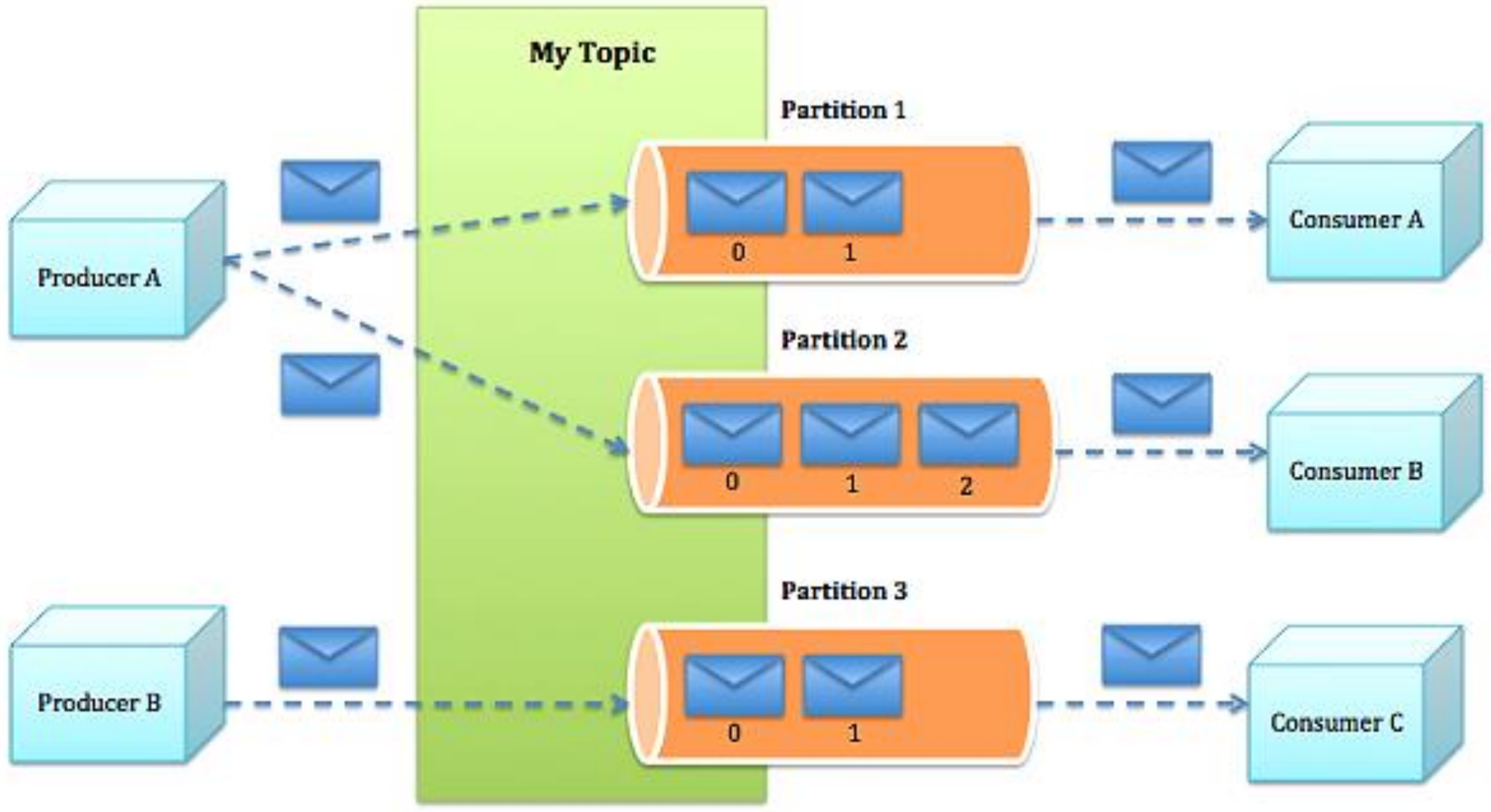
- Can’t do
  - Resilient to failure or scalable

Messaging Problems

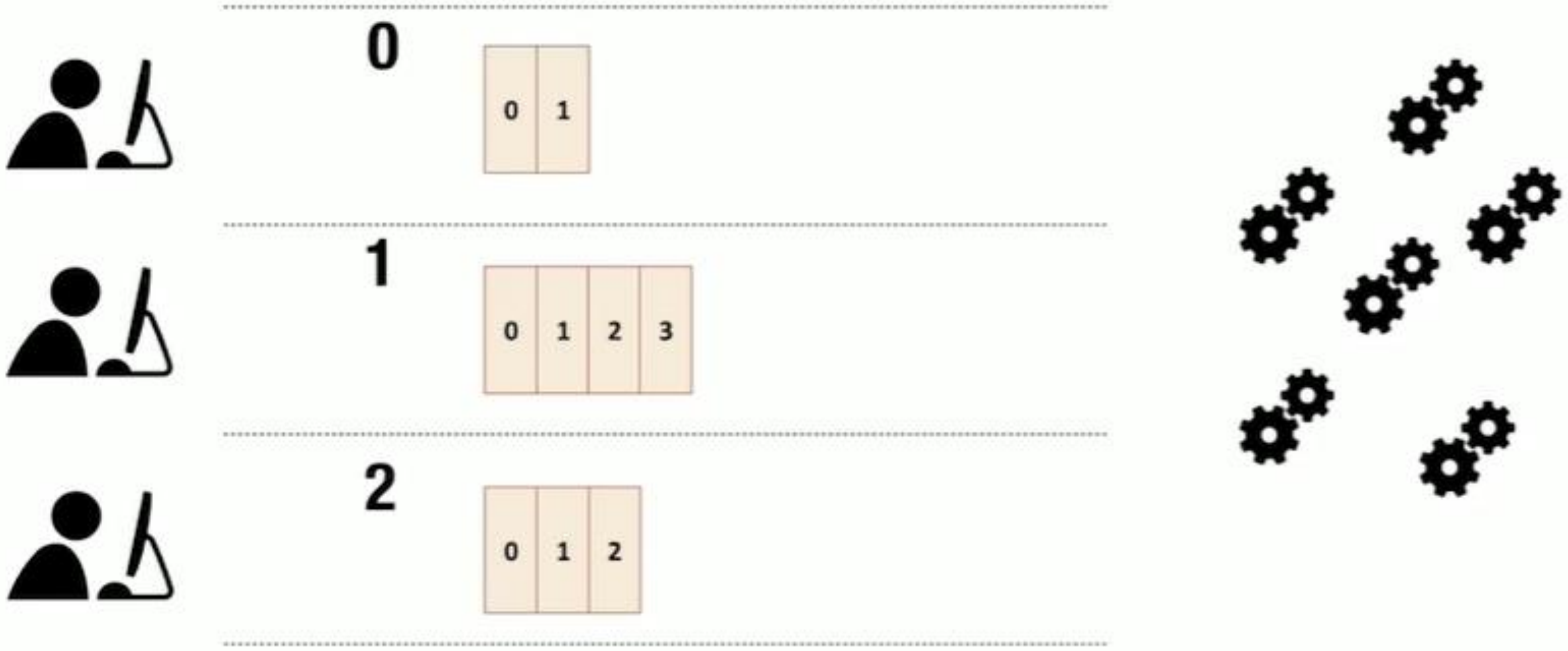
- What if a topic gets too big for one computer?
- What is one computer is not reliable enough?
- How strongly can we guarantee delivery?

APACHE kafka®

Topic partitioning



Topic Partitioning

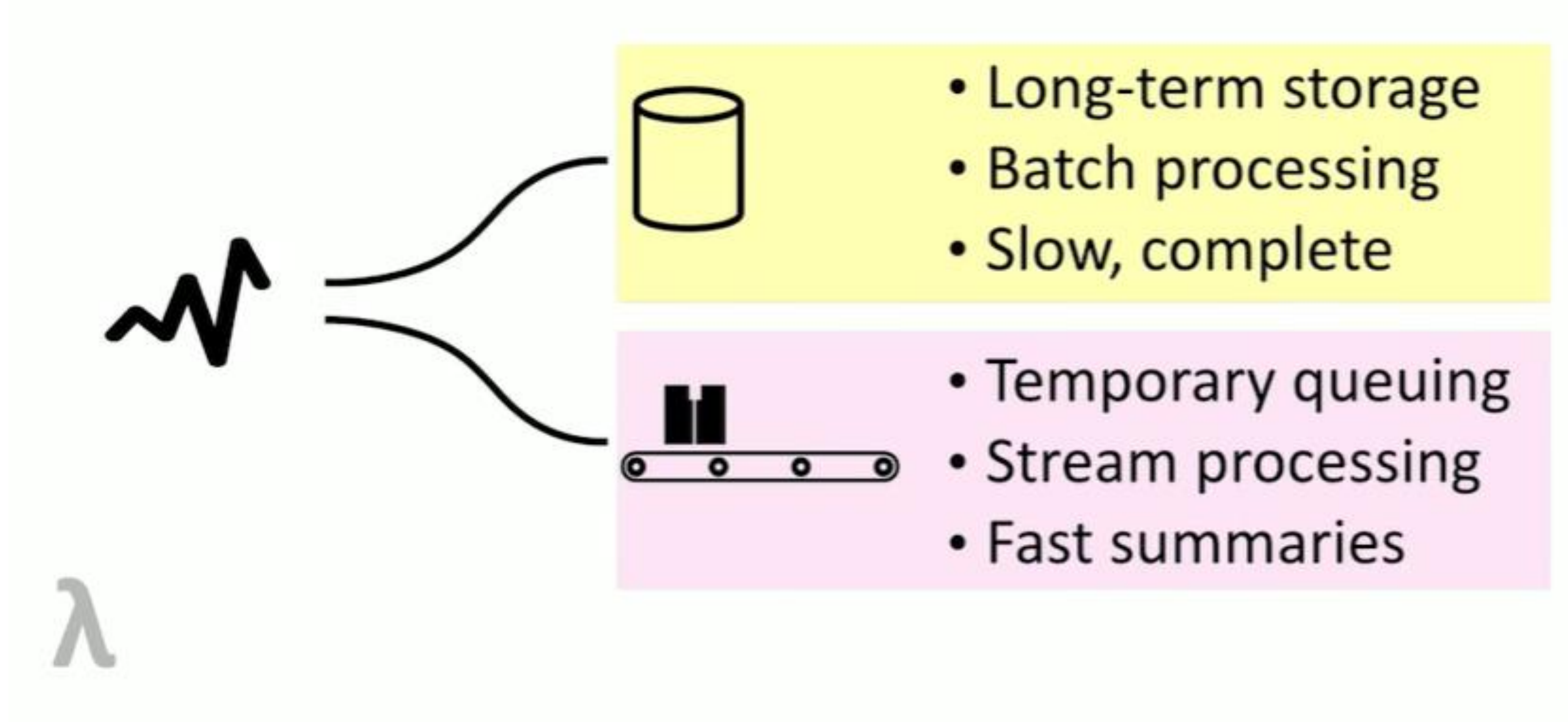


How to achieve ordering?

- If all messages must be ordered within one topic, use one partition.
- If messages can be ordered per a certain property (true in most cases), set a consistent message key and use multiple partitions.

- Yesterday you turned events into rows-in-place
- Tomorrow maybe you’ll just compute events

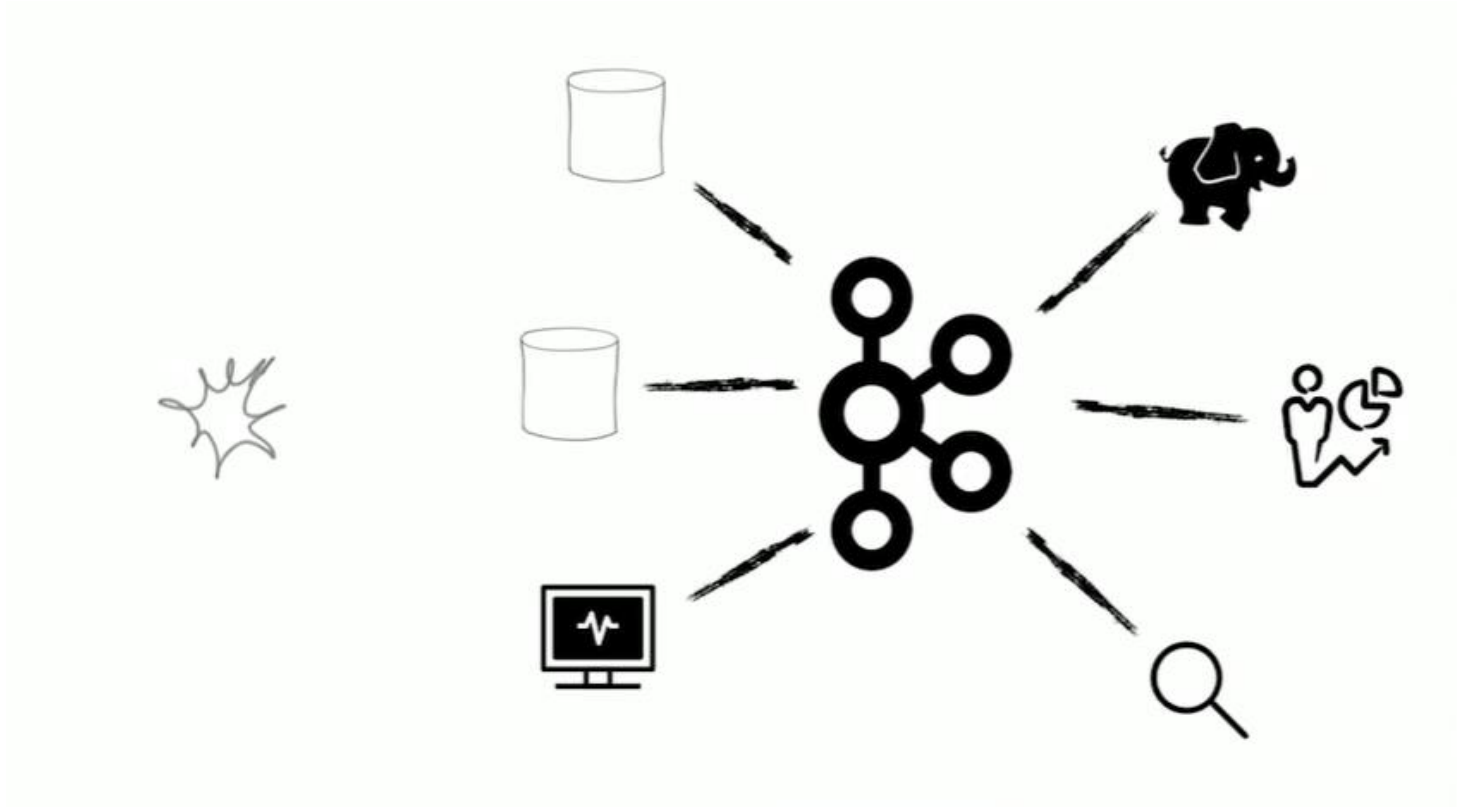
Without streaming

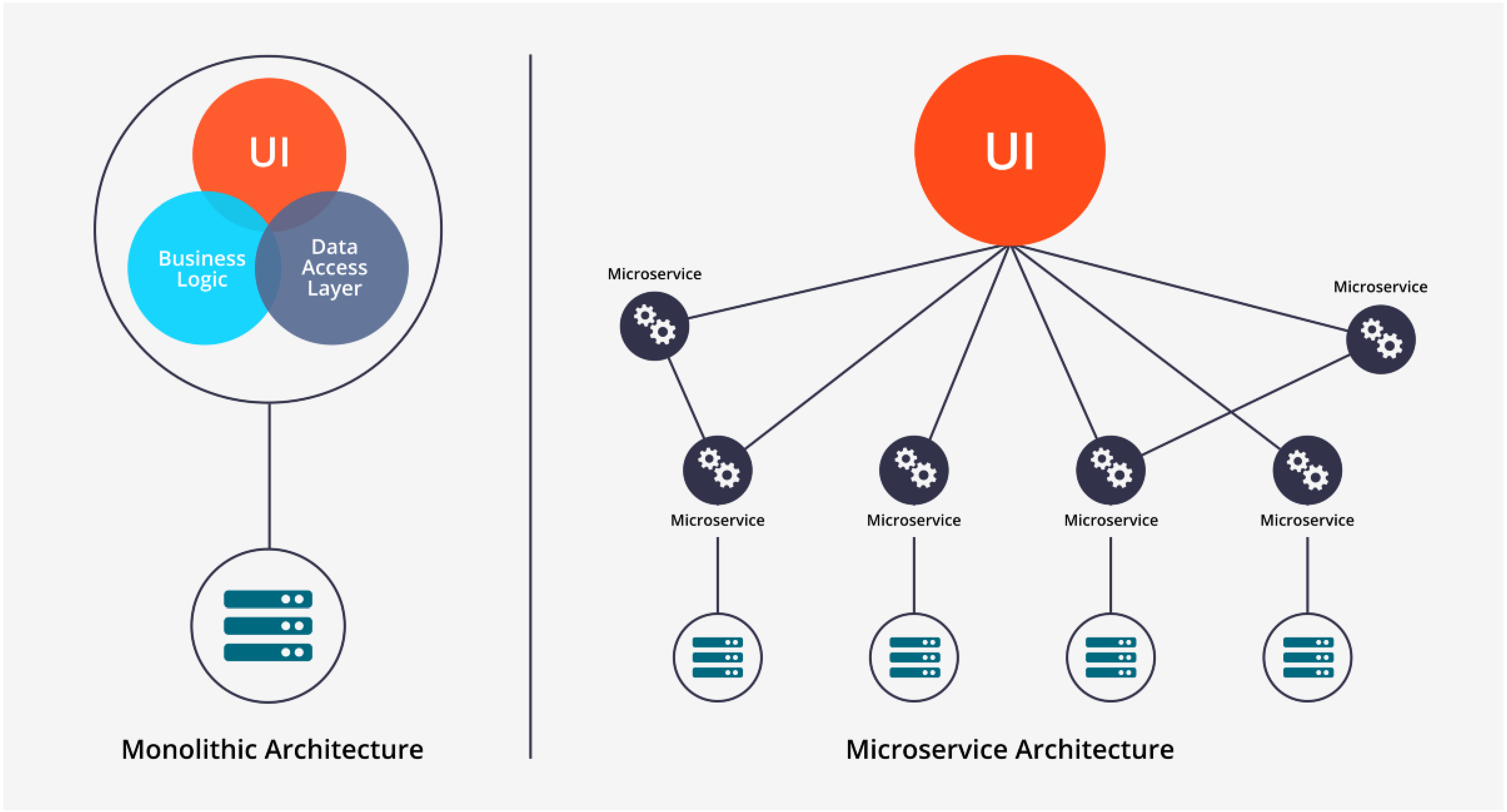


- Long-term storage
- Batch processing
- Slow, complete

- Temporary queuing
- Stream processing
- Fast summaries

With streaming




UI
Business Logic
Data Access Layer
Monolithic Architecture
UI
Microservice
Microservice
Microservice
Microservice
Microservice
Microservice
Microservice Architecture

THANK YOU