CASSANDRA

Looking at the bottom

What are we learning today?

> Why NoSQL like CASSANDRA

> Internal organ of CASSANDRA

> Basics of Distributed System Theory

> Basics of Scalable system design

ThoughtWorks®

Strength of SQL

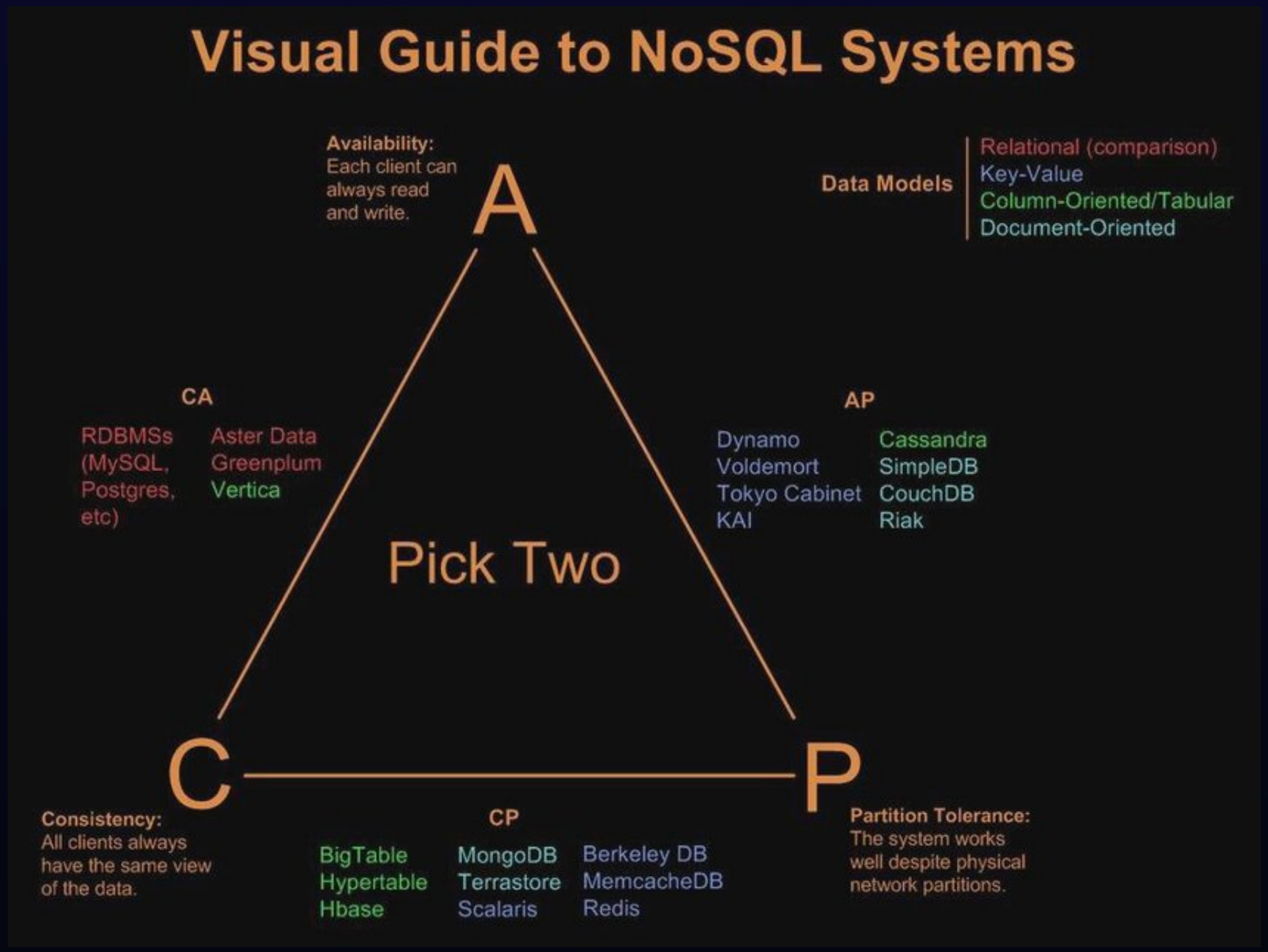
> Consistency via ACID transactions

> Strict constraint checks

> Maturity

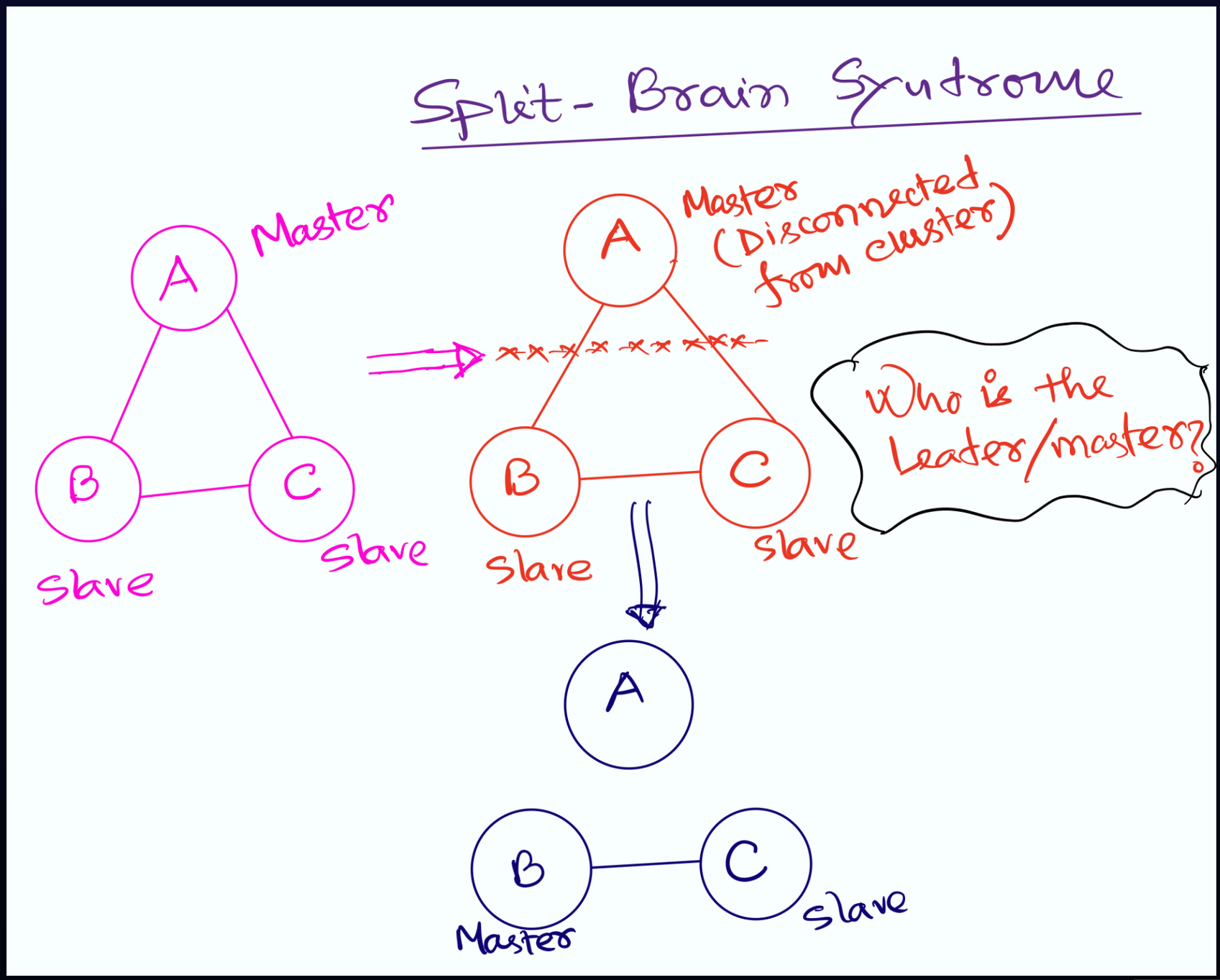
> Huge toolkits

CAP Theorem



ThoughtWorks®

Challenges with RDMS



ThoughtWorks®

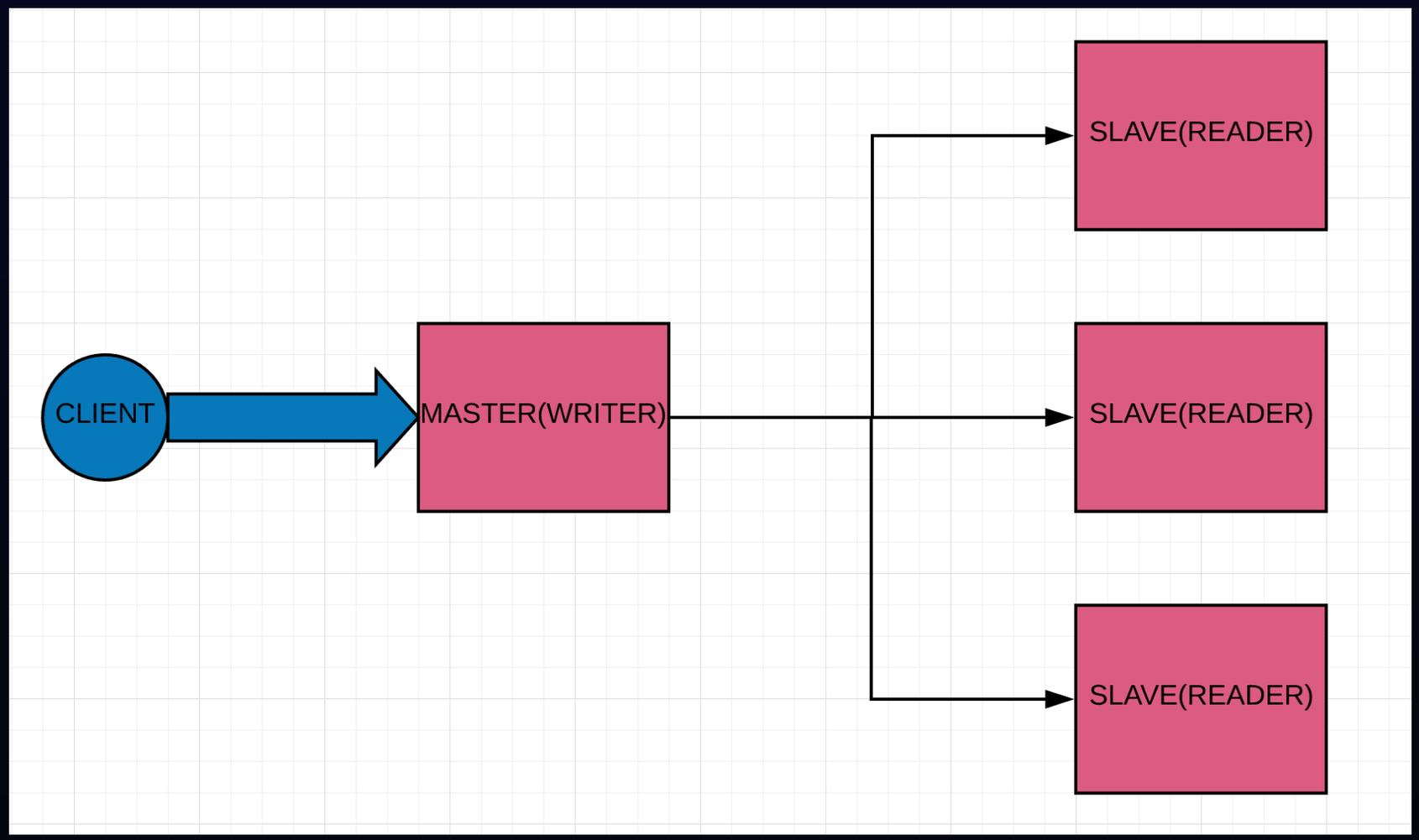
Challenges with RDMS

> Guess how long it might take for aurora for handling network partition or master failure??

ThoughtWorks®

Challenges with RDMS

> Challenge with Single Master(Tough Heavy writing)



ThoughtWorks®

Solution for Heavy writing

SHARDING

ThoughtWorks®

Tackle these topics

> Making high availability

> Remove split brain syndrome

> Faster Write

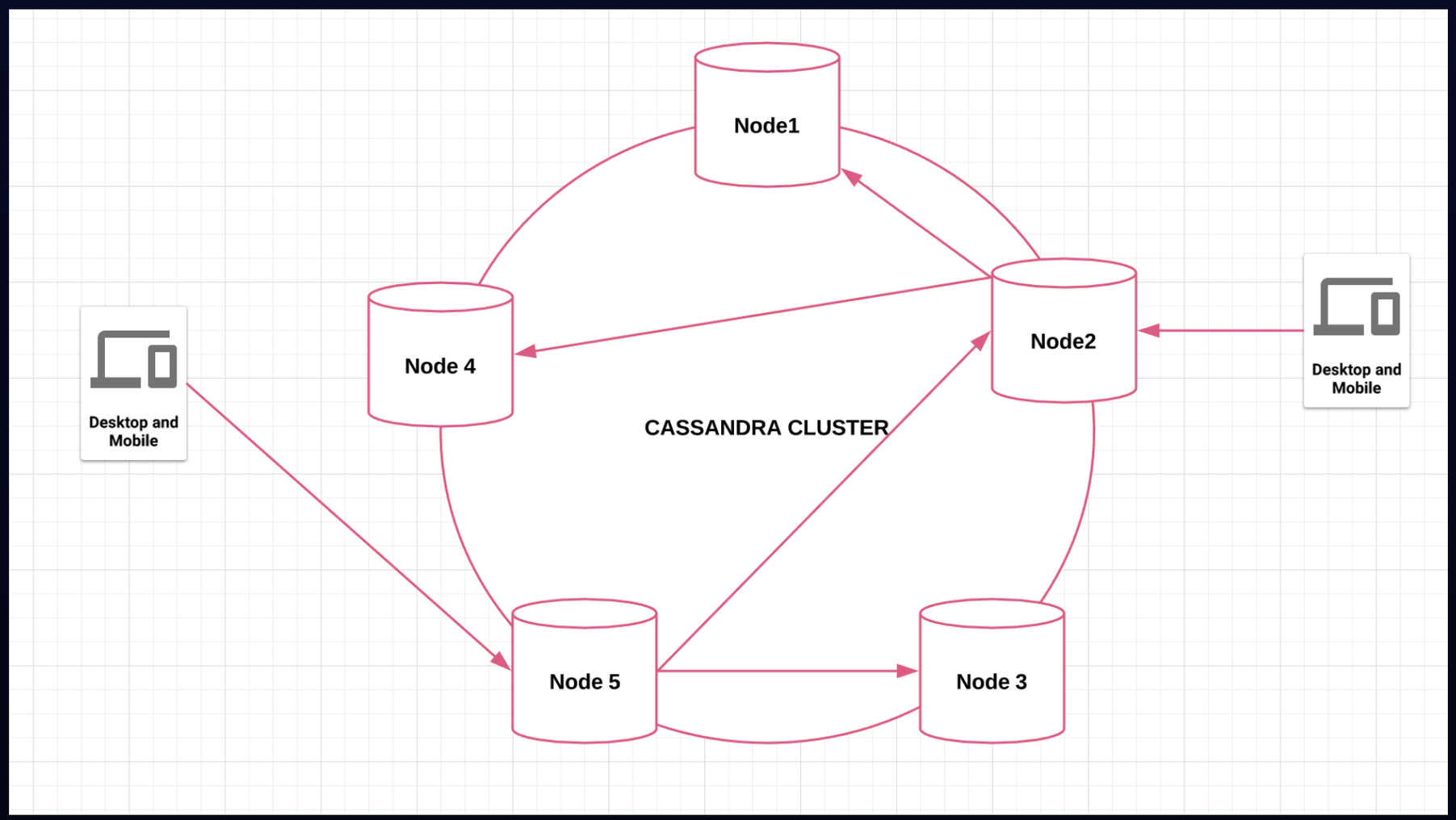
ThoughtWorks®

History of CASSANDRA

Cassandra uses a synthesis of well known techniques to achieve scalability and availability. Cassandra was designed to fulfill the storage needs of the Inbox Search problem. In-box Search is a feature that enables users to search through their Facebook Inbox. At Facebook this meant the system was required to handle a very high write throughput, billions of writes per day, and also scale with the number of users. Since users are served from data centers that are geographically distributed, being able to replicate data across data centers was key to keep search latencies down. Inbox Search was launched in June of 2008 for around 100 million users and today we are at over 250 million users and Cassandra has kept up the promise so far. Cassandra is now deployed as the backend storage system for multiple services within Facebook.

ThoughtWorks®

High Availability



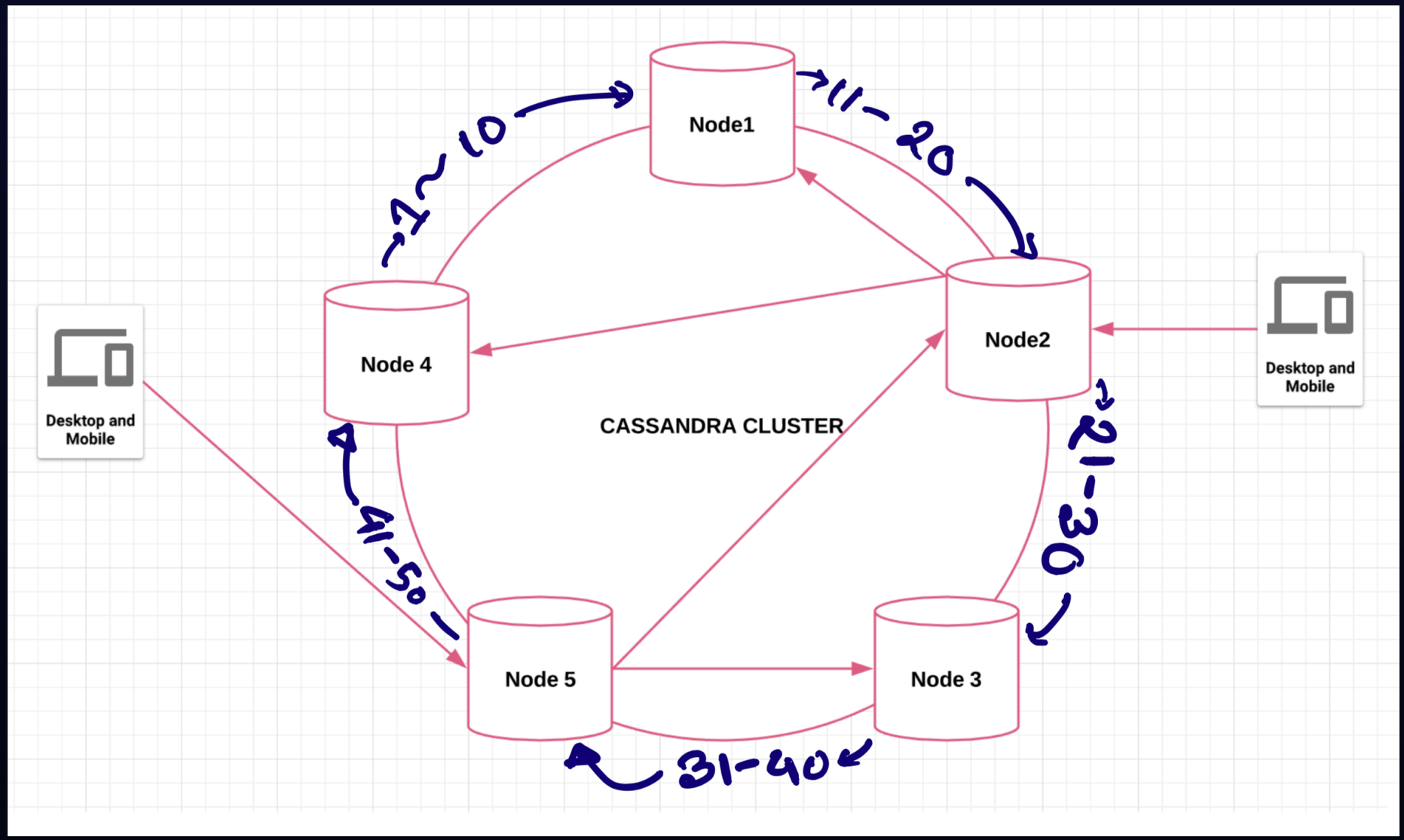
ThoughtWorks®

How node know other node?

In Cassandra, cluster nodes communicates between each other by using a protocol called “Gossip”

ThoughtWorks®

Consistent Hashing



ThoughtWorks®

Fault Tolerance

> Fault tolerance gained by duplication

> Scale out or scale down can be handled by consistent hashing very easily

Achieving faster write

Internal data structure to achieve massive write faster

Number to remember

Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Google™

ThoughtWorks®

How writes work?

commit_log(disc) -> write to slave -> response to client

What happened at the time to read like below query after inserting billions of row?

```
select * from activities where
activity_id=875858858 and timestamp>x and
timestamp<y
```

Answer

INDEXING

(add more details and challenges)

ThoughtWorks®

LSM tree

What is log structured merge tree?

Answer

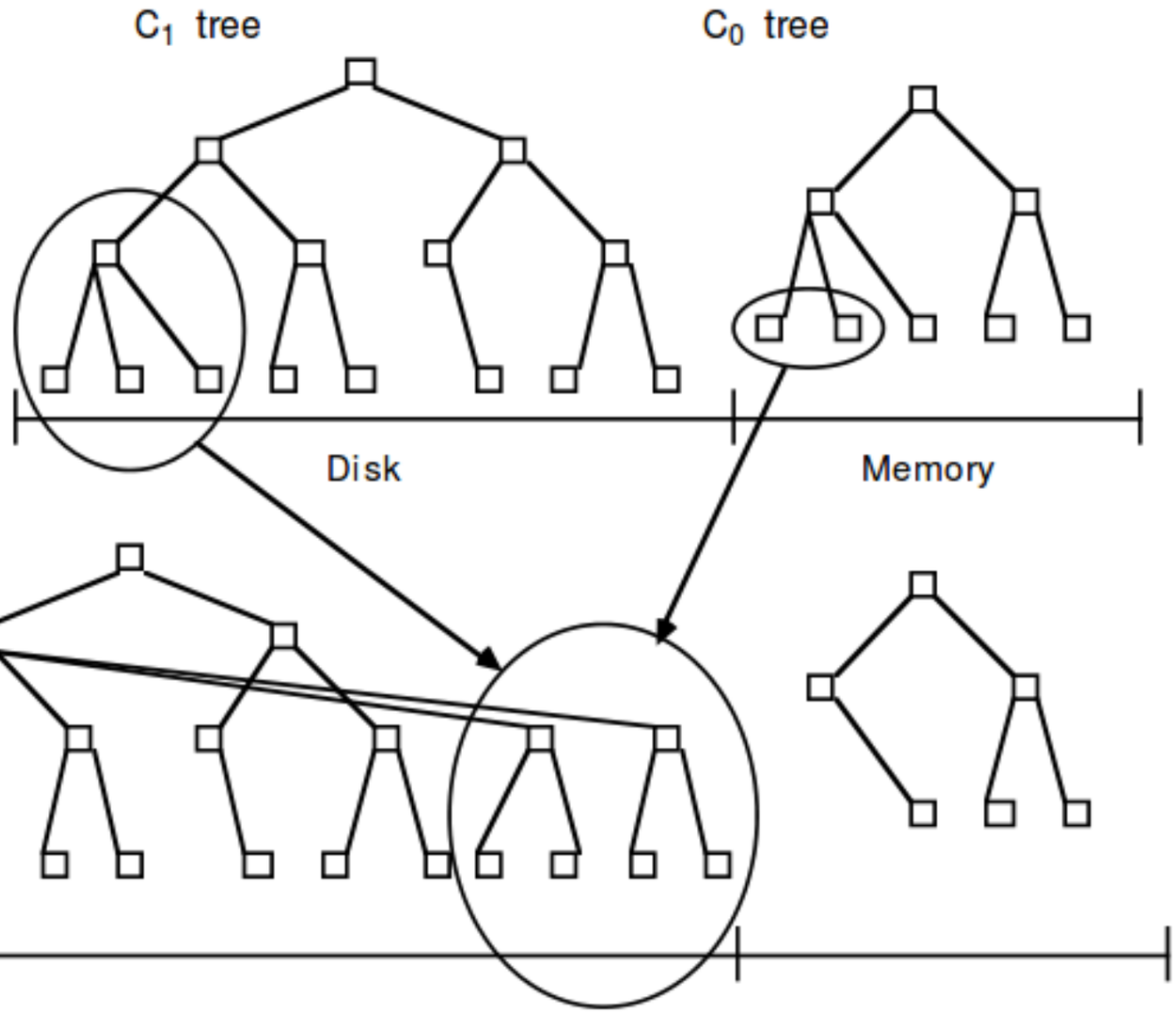


Figure 2.2. Conceptual picture of rolling merge steps, with result written back to disk

ThoughtWorks®

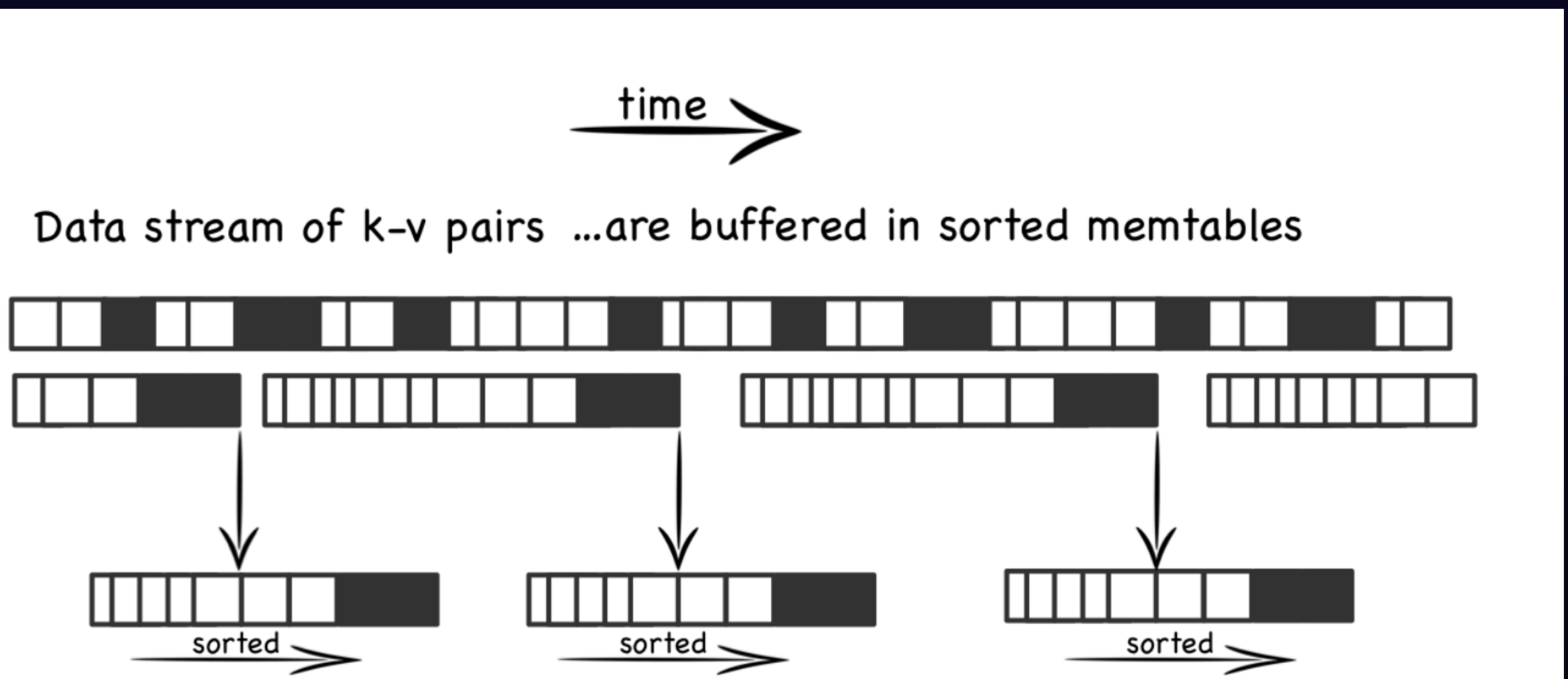
How it works?

Steps:

* Log the record(for recovery in case of crash)
* Add to C0 component(in memory writes)
* Merge merge.. into C1, compaction compaction and add to SSTable in the disc.

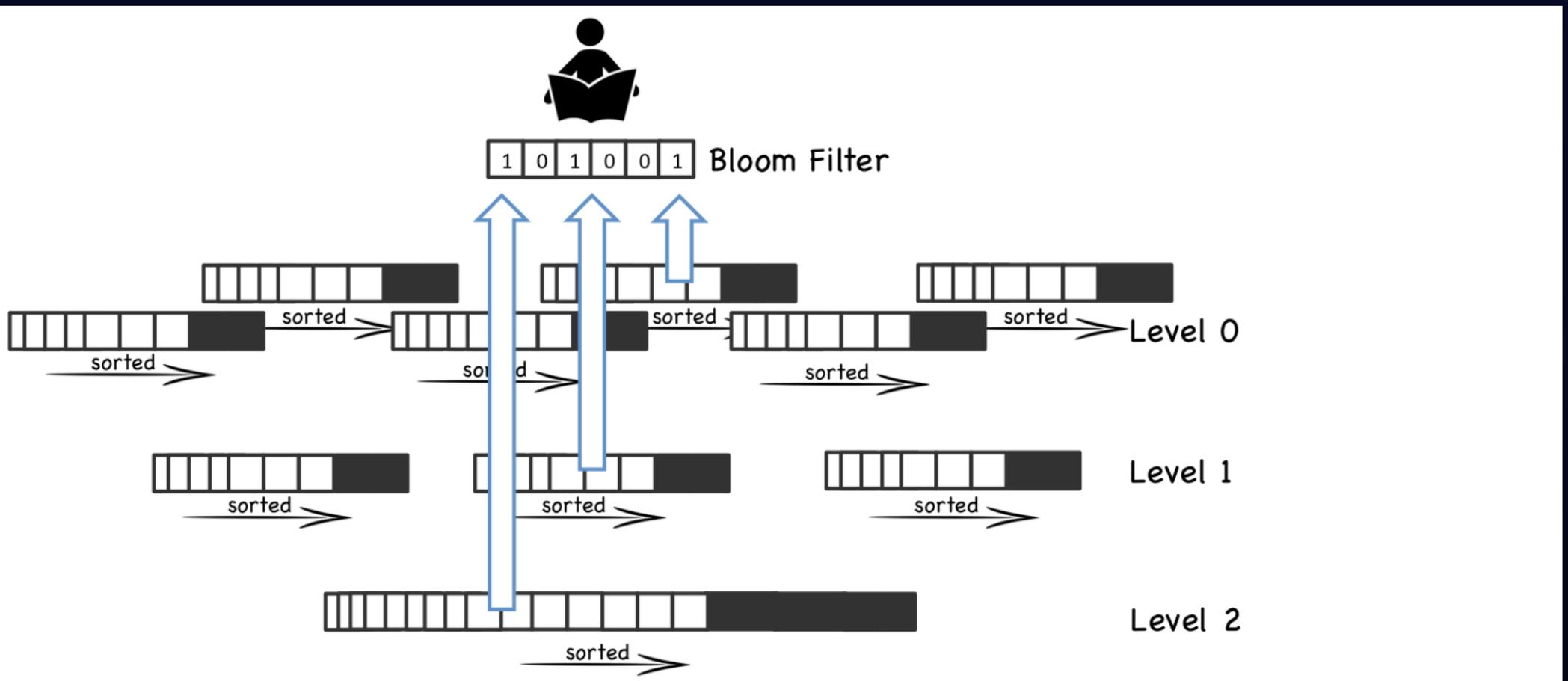
ThoughtWorks®

How does it look like?



ThoughtWorks®

How read work?



ThoughtWorks®

Thanks

The Omar
Future Principle Software Engineer