Zookeeper Mutex, Distributed Databases,

10 May 2026
Lecture 8

Slide Credits: Maarten van Steen
Many images copyright © Lena Wiese,
Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases.

Topics for Today

- Mutual Exclusion
  - Using Zookeeper
- Elections
- Distributed Databases
- Cyber: Storage and Naming

Using ZooKeeper basics

Centralized server setup

All client-server communication is nonblocking

- A client immediately gets a response

Maintains a tree-based namespace

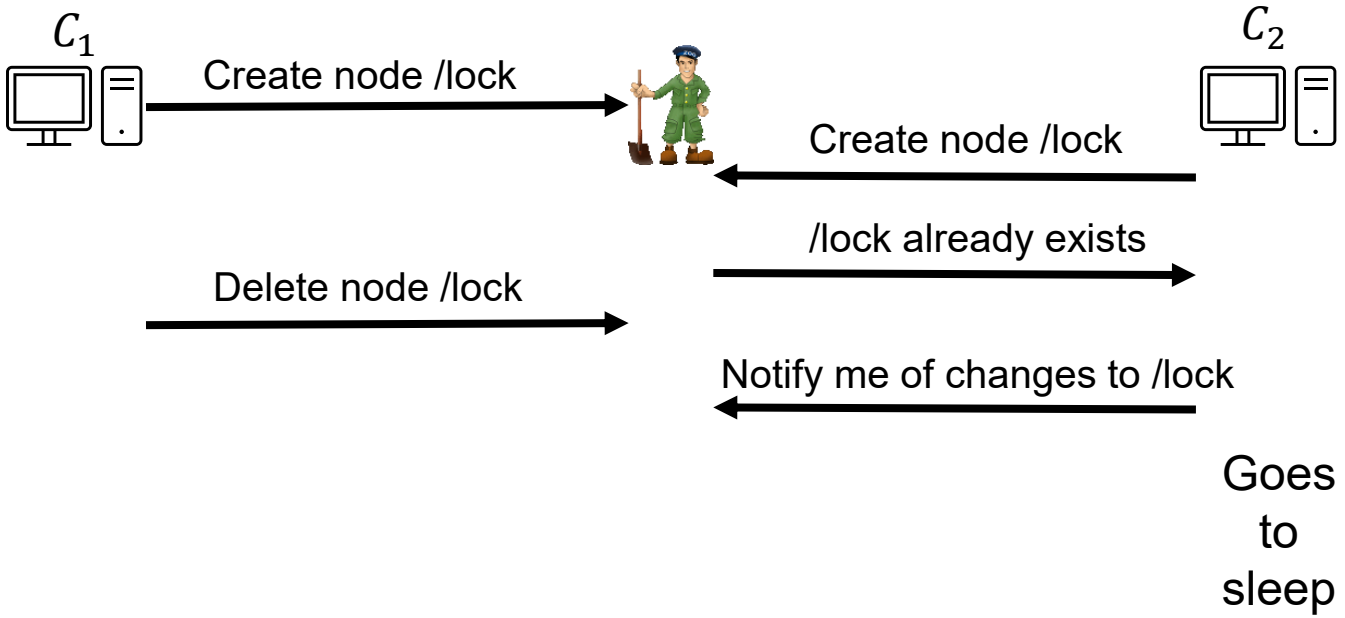
- Like a filesystem
- Example: /lock

Clients can

- create
- delete
- update nodes
- check existence

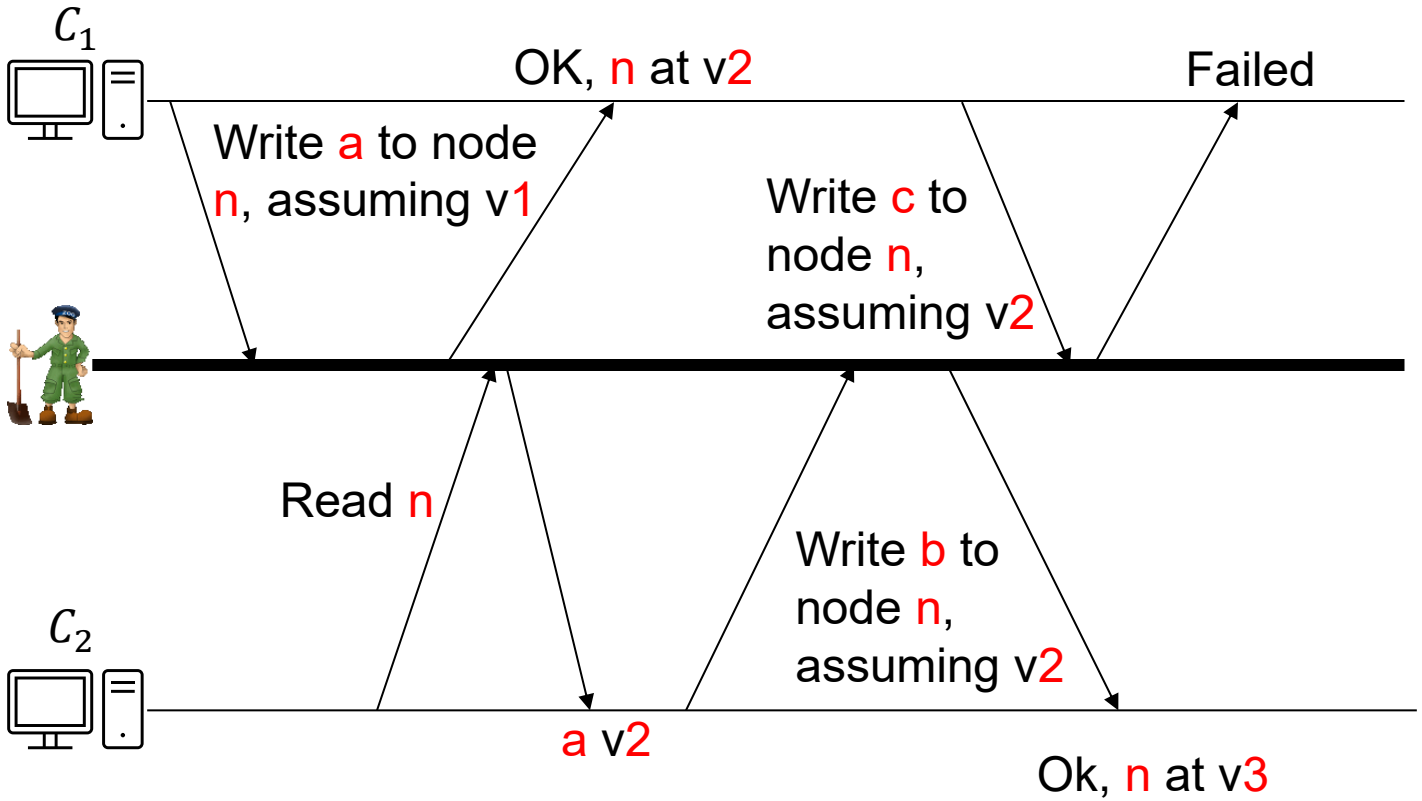
ZooKeeper Race Conditions

- ZooKeeper allows a client to be notified when a node or a branch in the tree changes
- May easily lead to race conditions.

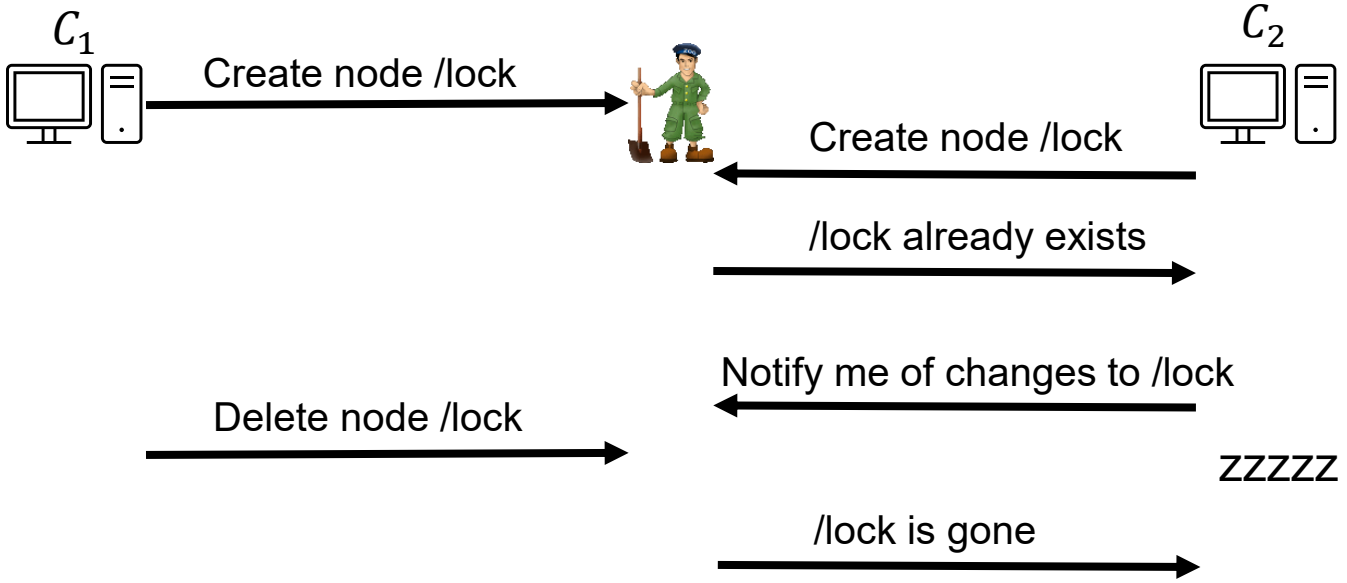


- Solution: Use Version Numbers

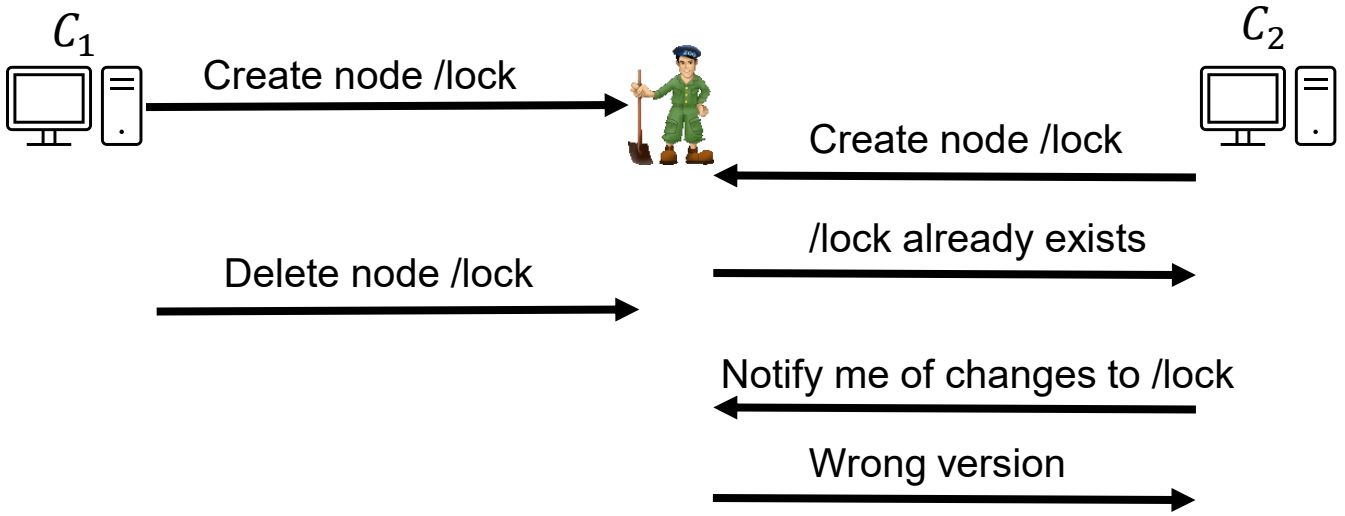
ZooKeeper Versioning



ZooKeeper Locking



ZooKeeper Locking



So Far

- Mutual Exclusion
  - Using Zookeeper
- Elections
- Distributed Databases
- Cyber: Storage and Naming

Election Algorithms

Principle: An algorithm requires some process acts as a coordinator. How to select the special process **dynamically**.

Note: In many systems the coordinator is chosen by hand (e.g. file servers). This leads to centralized solutions → single point of failure.

Question: If a coordinator is chosen dynamically, is it centralized? Distributed?

Question: Is a fully distributed solution, i.e. one without a coordinator, always more robust than any centralized or coordinated solution?

Election by Bullying (1/2)

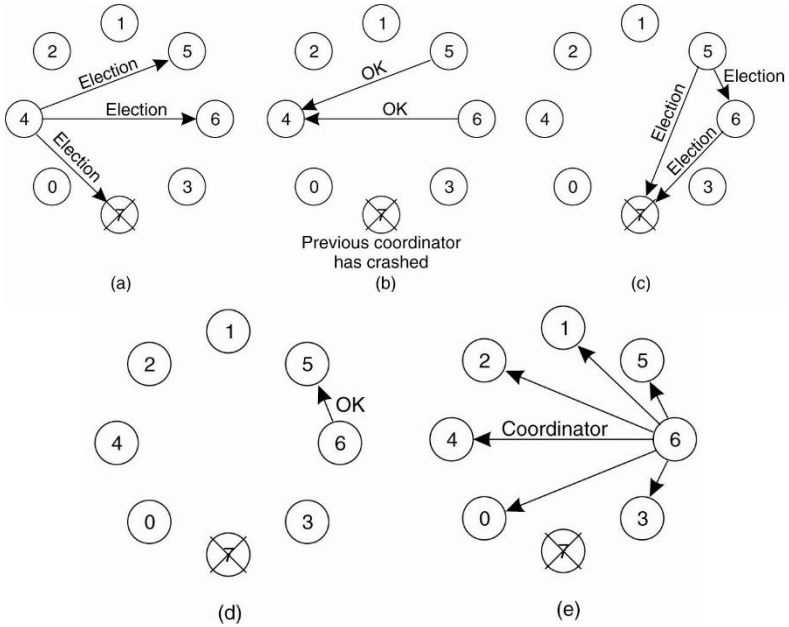
Principle: Each process has an associated priority (weight). Process with the highest priority must be elected as coordinator.

Issue: How to find the heaviest process?

1. Any process can just start an election by sending an election message to all other processes (assuming you don't know the weights of the others).
2. If a process P_{heavy} receives an election message from a lighter process P_{light}, it sends a take-over message to P_{light}. P_{light} is out of the race.
3. If a process doesn't get a take-over message back, it wins, and sends a victory message to all other processes.

Election by Bullying (2/2)

Question: We're assuming something very important here – what?



If you have broadcast – just broadcast to everyone.

Election in a Ring

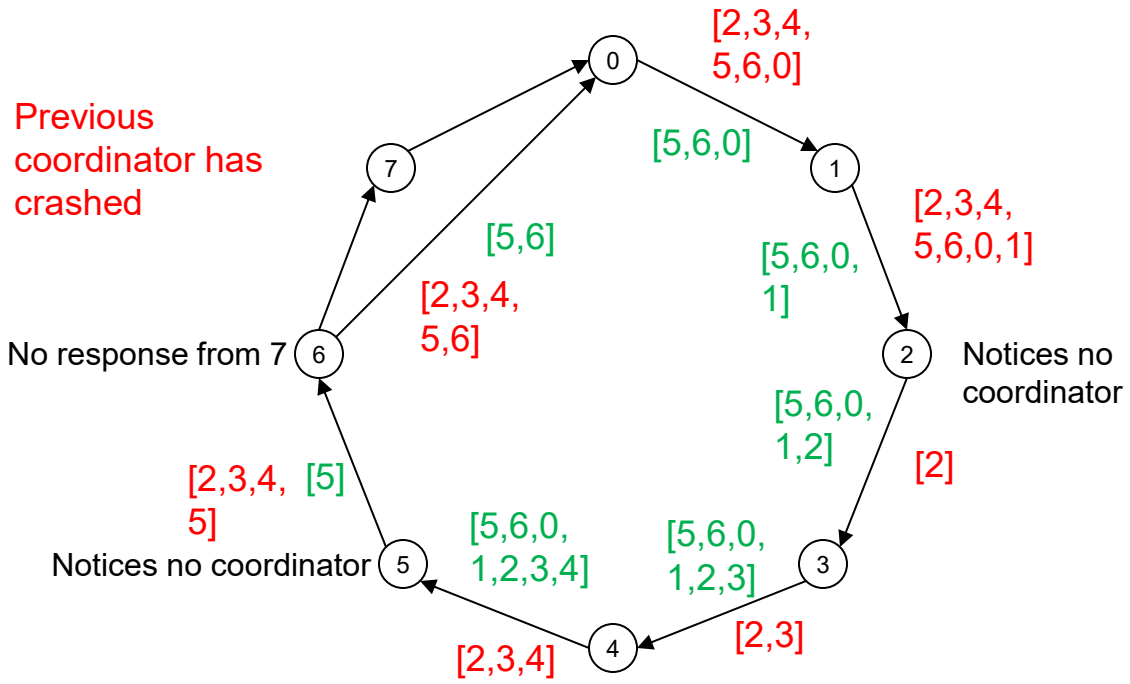
Principle: Process priority is obtained by organizing processes into a (logical) ring. Process with the highest priority should be elected as coordinator.

- Any process can start an election by sending an election message to its successor. If successor is down, the message is passed on to the next successor.
- If a message is passed on, the sender adds itself to the list. When it gets back to the initiator, everyone had a chance to make its presence known.
- The initiator sends a coordinator message around the ring containing a list of all living processes. The one with the highest priority is elected as coordinator.

Question: Does it matter if two processes initiate an election?

Question: What happens if a process crashes *during* the election?

Election in a Ring



Leader election in ZooKeeper server group

- Each server s in the server group has an identifier $id(s)$
- Each server has a monotonically increasing counter $tx(s)$ of the latest transaction it handled (i.e., series of operations on the namespace).
- When follower s suspects leader crashed, it broadcasts an ELECTION message, along with the pair $(voteID, voteTX)$. Initially,
  - $voteID \leftarrow id(s)$
  - $voteTX \leftarrow tx(s)$
- Each server s maintains two variables:
  - $leader(s)$: records the server that s believes may be final leader. Initially, $leader(s) \leftarrow id(s)$.
  - $lastTX(s)$: what s knows to be the most recent transaction. Initially, $lastTX(s) \leftarrow tx(s)$.

Leader election in ZooKeeper server group

- When s^* receives $\langle voteID, voteTX \rangle$
- If $lastTX(s^*) < voteTX$, then s^* just received more up-to-date information on the most recent transaction, and sets
  - $leader(s^*) \leftarrow voteID$
  - $lastTX(s^*) \leftarrow voteTX$
- If $lastTX(s^*) = voteTX$ and $leader(s^*) < voteID$, then s^* knows as much about the most recent transaction as what it was just sent, but its perspective on which server will be the next leader needs to be updated:
  - $leader(s^*) \leftarrow voteID$

Note

- When s^* believes it should be the leader, it broadcasts $\langle id(s^*), tx(s^*) \rangle$.
- Essentially, we're bullying.

So Far

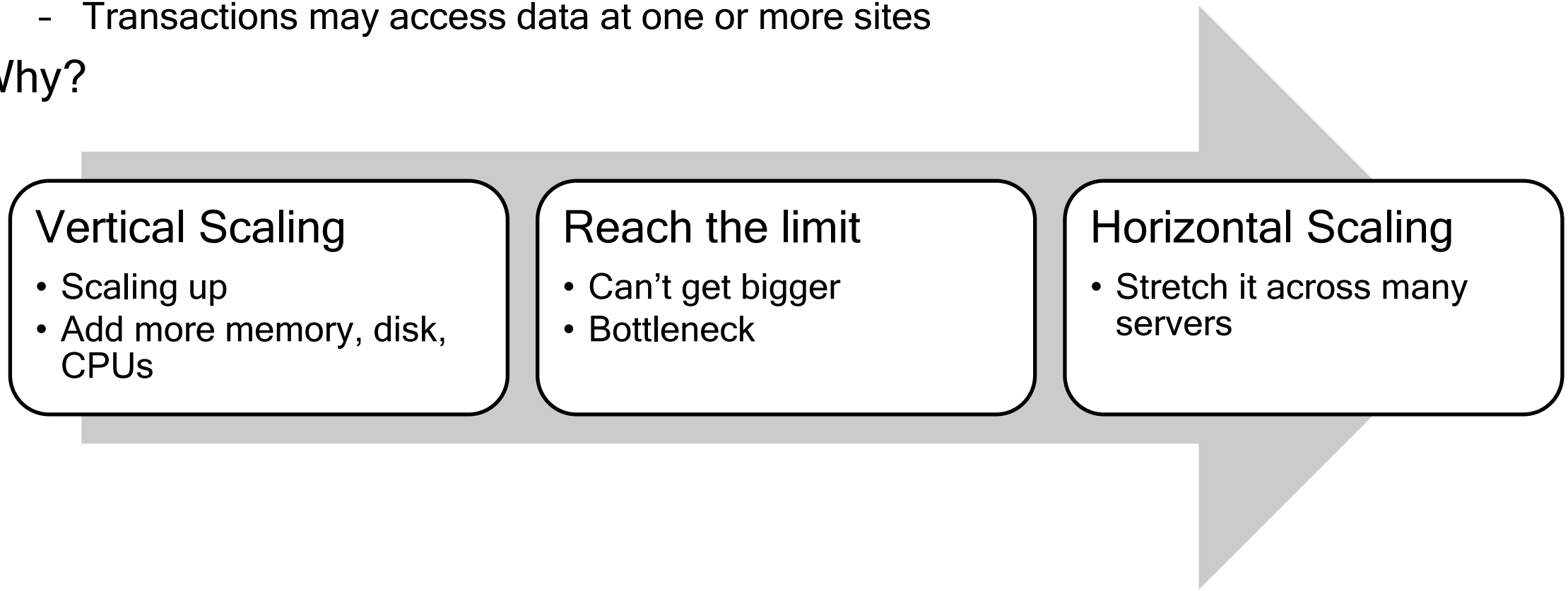
- Mutual Exclusion
  - Using Zookeeper
- Elections
- Distributed Databases
- Cyber: Storage and Naming

Distributed Databases Intro

A distributed database system consists of loosely coupled sites that share no physical components

- DBs that run at each site are independent of each other
- Transactions may access data at one or more sites

Why?



Distribution Advantages

Load balancing

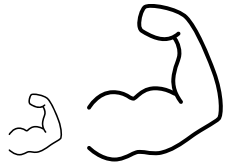
- Divide the traffic and burden

Flexible Scaling

- Add or remove servers as needed

Heterogenous Nodes

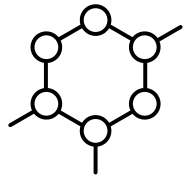
- Combine powerful and cheap servers



Symmetric Configuration

- Nodes can replace each other
- Failure recovery

Decentralized Control



- P2P algorithms for failure tolerance
- No single point of failure

Distributed Database Transparency

Access

- Uniform query and management interfaces

Location

- User can query without specifying where to run it

Replication

- Can query anywhere in a replicated system and get same answer
- Nodes update each other

Fragmentation

- Data may be split up
- Queries are routed to the correct nodes as needed

Migration

- If data moves, user is unaware

Concurrency

- Many users may work at once
- Resolve conflicts

Failure

- Work even in presence of failures
- Recover from missed messages

Distributed Data Storage

Assume relational data model. Major goals of distribution:

- Replication
  - System maintains multiple copies of data (stored in different sites) for fast retrieval and fault tolerance.
- Fragmentation
  - Relation is partitioned into several fragments stored at different sites

---

- Replication + Fragmentation
  - Relation is partitioned into several fragments: system maintains several identical replicas of each such fragment.

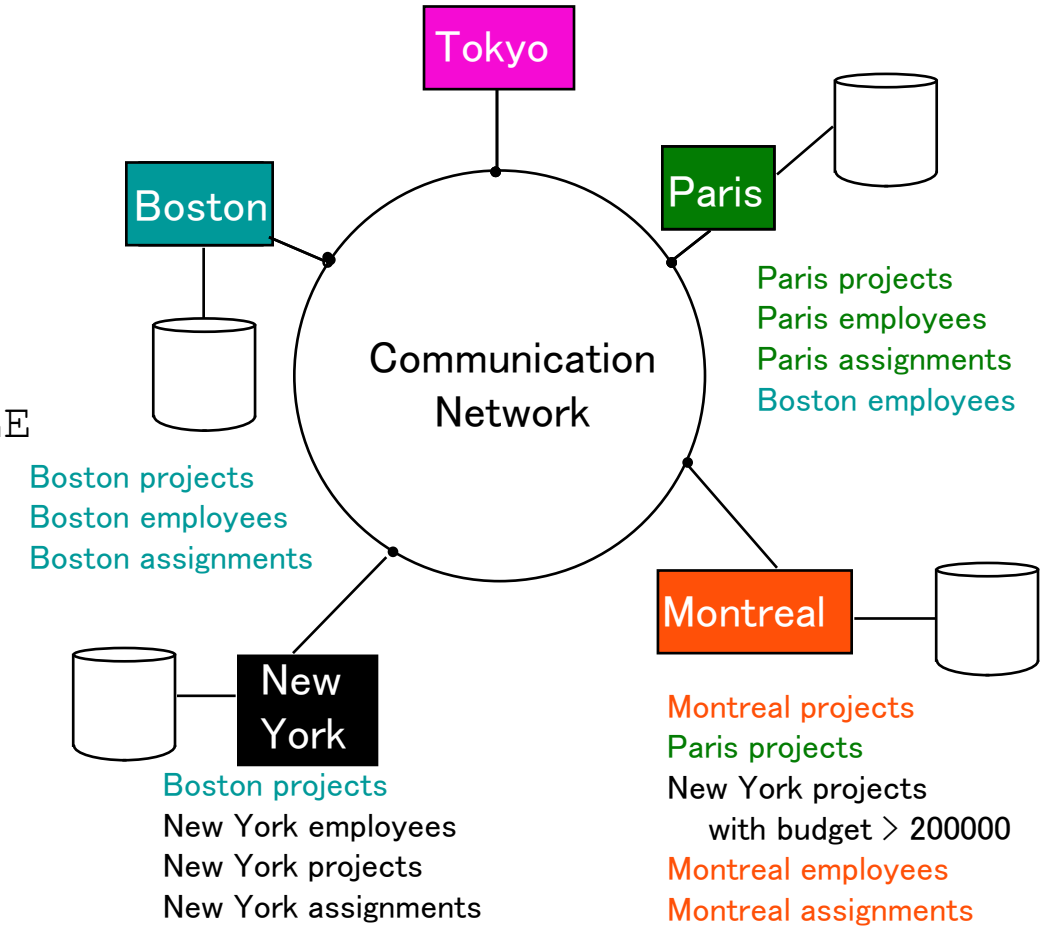
Data Replication

Relation or fragment is **replicated** when it is stored at two or more sites.

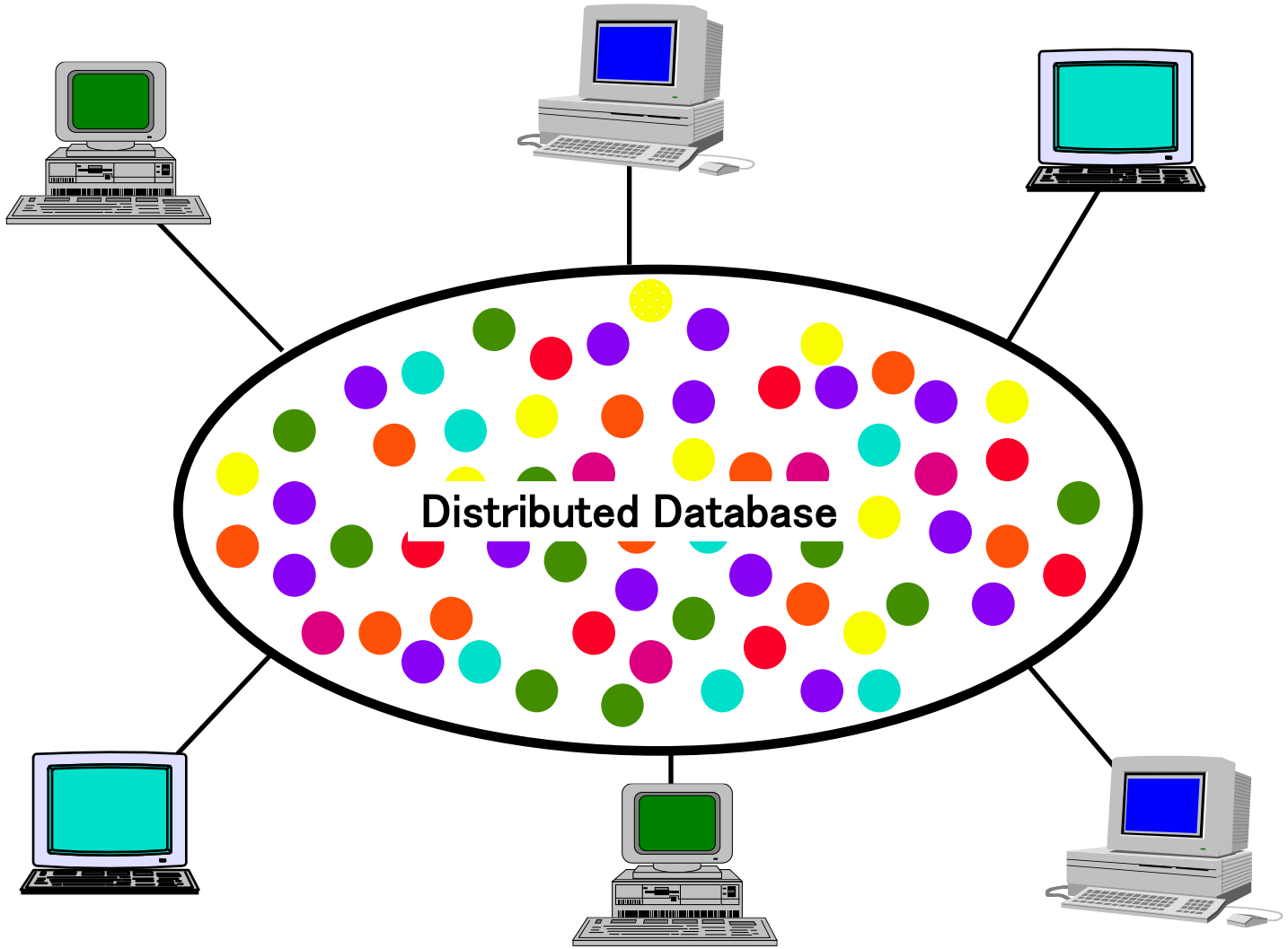
- Full replication → the relation is stored at all sites.
- Fully redundant DBs → every site has a copy of the entire database.

Fragmentation and Transparent Access

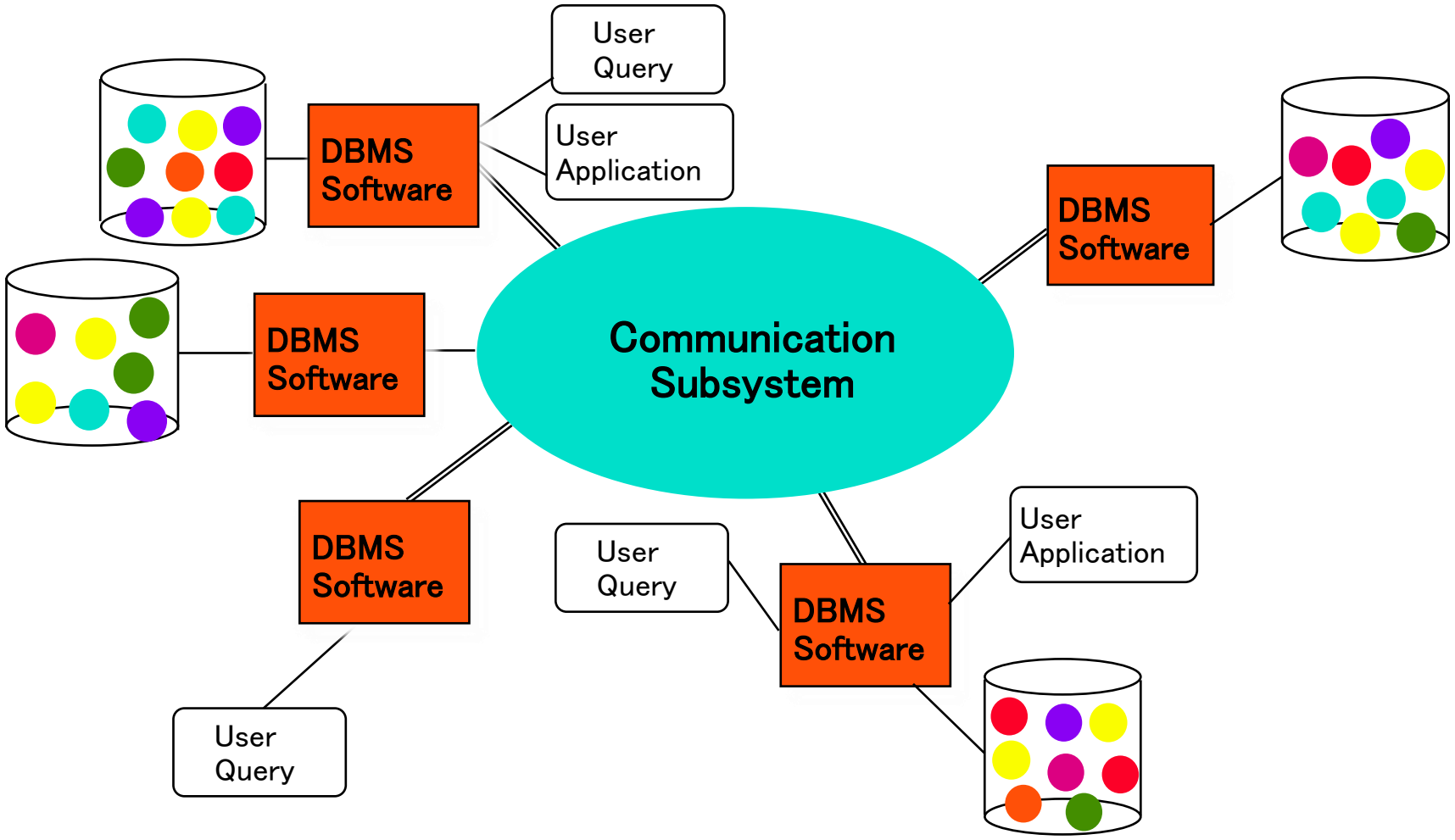
```
SELECT ENAME,SAL
FROM   EMP,ASG,PAY
WHERE  DUR > 12
AND    EMP.ENO = ASG.ENO
AND    PAY.TITLE = EMP.TITLE
```



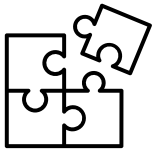
Distributed Database - User View



Distributed DBMS - Reality



Data Fragmentation



Division of r into fragments $r_1, r_2, r_3, \ldots, r_n$ which contain sufficient information to reconstruct it.

Horizontal fragmentation: each tuple of r is assigned to one or more fragments

Vertical fragmentation: the schema (columns) for r is split into several smaller schemas

- All schemas must contain a common candidate key (or superkey) to ensure lossless join (reconstruction).
- A special attribute (a rowId or artificial key) may be added to the schema

Example: relation account with schema:

- `Account = (branch_name, account_number, balance )`

Example: Horizontal Fragmentation

Table `Account(branch_name, account_number, balance)`

$account_1 = \sigma_{branch\_name="Hillside"}(Account)$

branch_name	account_number	balance
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62

branch_name	account_number	balance
Valleyview	A-177	205
Valleyview	A-402	1000
Valleyview	A-408	1123
Valleyview	A-639	750

$account_2 = \sigma_{branch\_name="Valleyview"}(Account)$

Example: Vertical Fragmentation

Table `Customer_Info(tuple_id, account_number, branch_name, customer_name, balance)`

$$deposit_1 = \Pi_{branch\_name,customer\_name,tuple\_id}(Customer\_info)$$

branch_name	customer_name	tuple_id
Hillside	Lowman	1
Hillside	Camp	2
Valleyview	Camp	3
Valleyview	Kahn	4
Hillside	Kahn	5
Valleyview	Kahn	6
Valleyview	Green	7

tuple_id	account_number	balance
1	A-305	500
2	A-226	336
3	A-177	205
4	A-402	10000
5	A-155	62
6	A-408	1123
7	A-639	750

$$deposit_2 = \Pi_{tuple\_id,account\_number,balance}(Customer\_info)$$

Fragmentation Advantages

Horizontal

- Parallel processing on fragments of a relation
- Can split a relation so that tuples are located where they are most frequently accessed

Vertical

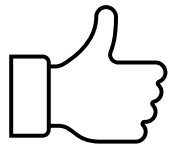
- Columns can be split so that each part of the tuple is stored where it is most frequently accessed
  - tuple-id attribute allows efficient joining of vertical fragments
- Parallel processing on a relation by column splits

Can mix the two (vertical and horizontal)
Fragments may be re-fragmented to an arbitrary depth

Fragmentation Costs

Horizontal

- Division into equal parts
- Hot spots – data in particular demand
- Maintenance over time with data creation and deletion



Vertical

- Joins across columns are costly
- Need to calculate affinity – which columns are more likely to be requested together
- Need to contact many servers to do complete query
  - SELECT * FROM Sailors

Sharding – NoSQL Databases

No tables, just record IDs

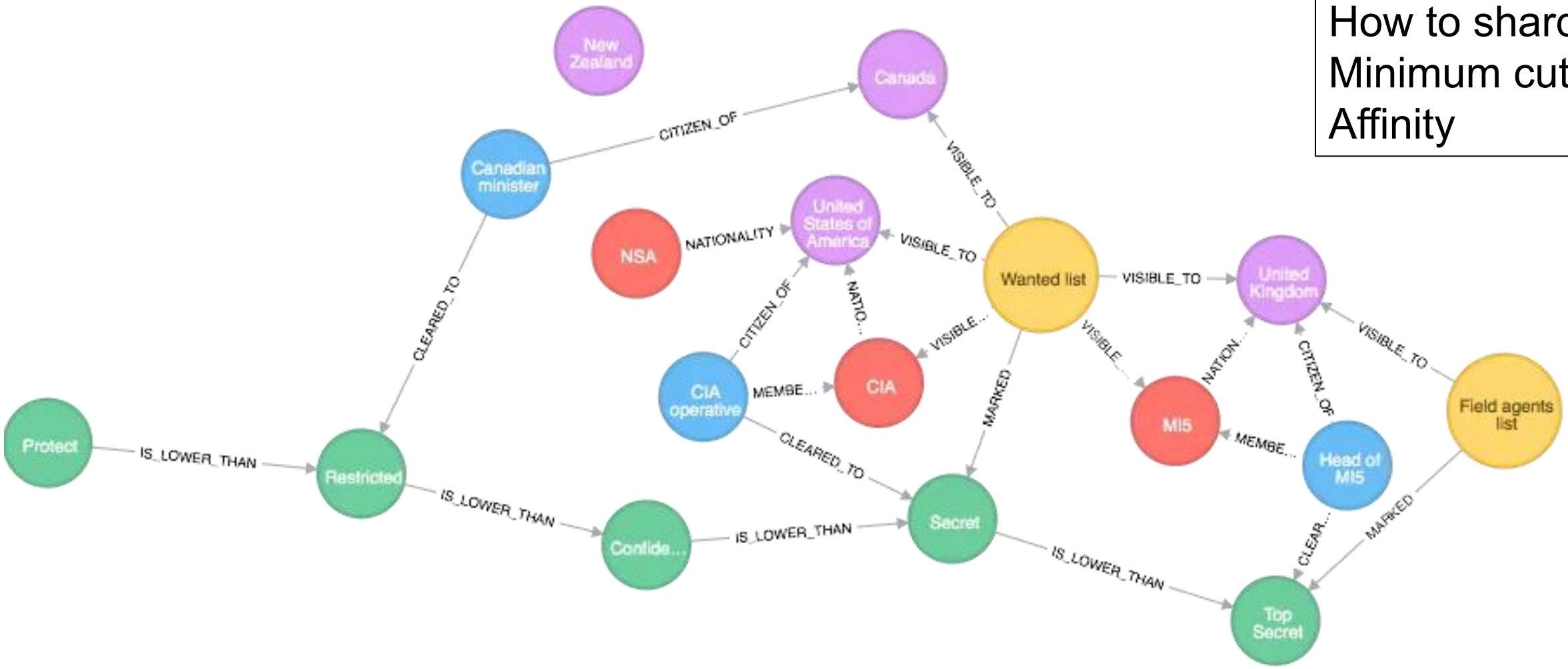
Divide up responsibility for records among servers

No joins, so simpler

Consider which records are commonly retrieved with which

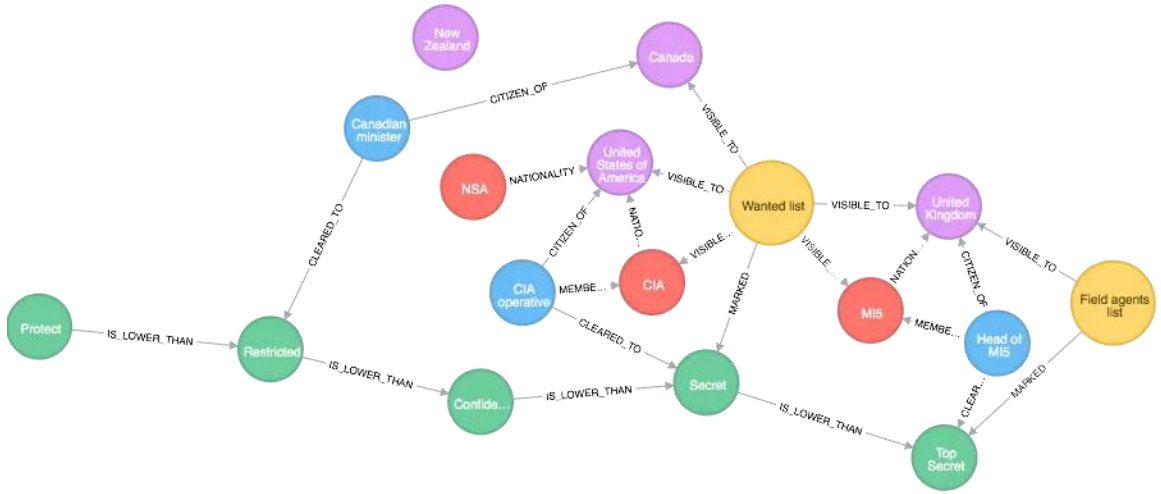
Sharding Graph Databases

How to shard?
Minimum cut
Affinity



https://singhnaveen.medium.com/what-are-graph-databases-and-different-types-of-graph-databases-369e5040a9d0

Sharding Graph Databases: Options



https://singhnaveen.medium.com/what-are-graph-databases-and-different-types-of-graph-databases-369e5040a9d0

Manual

- Place nodes near where they are used

Random

- Place nodes randomly

Hash based

- Assign range of nodes based on hash values

Workload driven

- Reduce cuts traversed in each transaction
- Keep track of usage and history

Issues:

- How to find nodes at the end of a connection?
- Quickly query nodes in a string?

Naming Data Items - Criteria

Uniqueness	Search	Migrate	Autonomy
Every data item must have a system-wide unique name.	• Must be able to find the location of data items efficiently.	• Must be able to change the location of data items transparently.	• Each site should be able to create new data items autonomously.

- I have 100 tables, 500 columns, 1,000,000 data rows
- How do I identify them?

Solution 1: Centralized (Name Server)

Unique

Search

Migrate

Autonomy

Structure:

- Name Server assigns all names
- Each site maintains a record of local data items
- Sites ask Name Server to locate non-local data items

Advantages:

- Satisfies criteria Unique, Search, Migrate

Disadvantages:

- Does not satisfy criterion Autonomy
- Name Server is a potential performance bottleneck
- Name Server is a single point of failure

Solution 2: Aliases and Local

Unique

Search

Migrate

Autonomy

Structure:

- Each site prefixes its own site identifier to any name that it generates (i.e., *site*17.a*ccount)*
  - ✓Gives a unique identifier
  - ✓Avoids problems with central control.
  - ✗ Does not give network transparency.

Solution: Create local **aliases** for data items; Store the mapping of aliases to the real names at each site.

Results:

- User can be unaware of the physical location of a data item
- User is unaffected if the data item is moved from one site to another.

Consistent Hashing

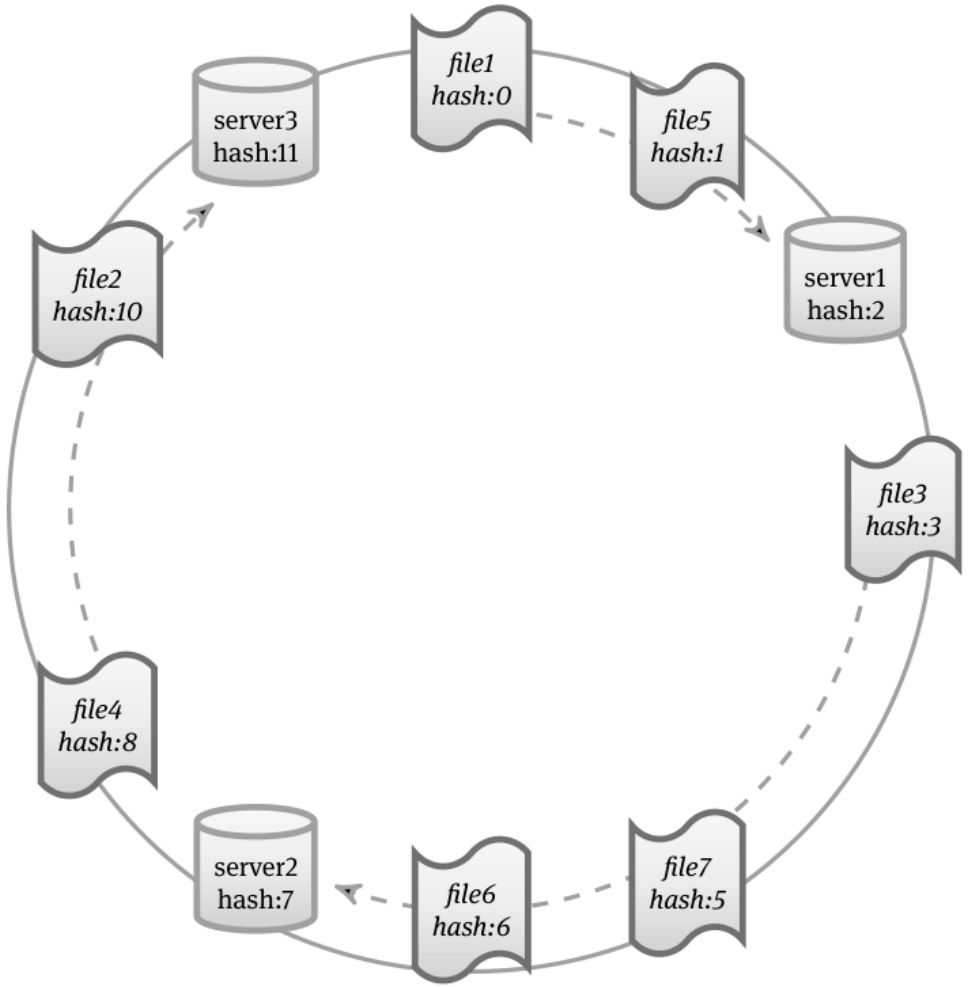


Fig. 11.3. Data allocation with consistent hashing

- Each Server gets a random ID
  - Should put things randomly in the name space
- Each Entity gets a random ID
  - Should put things randomly in the name space
- Result: Even distribution!

- Assign responsibility to the Server that succeeds the Entity's ID

Consistent Hashing Entry and Exit

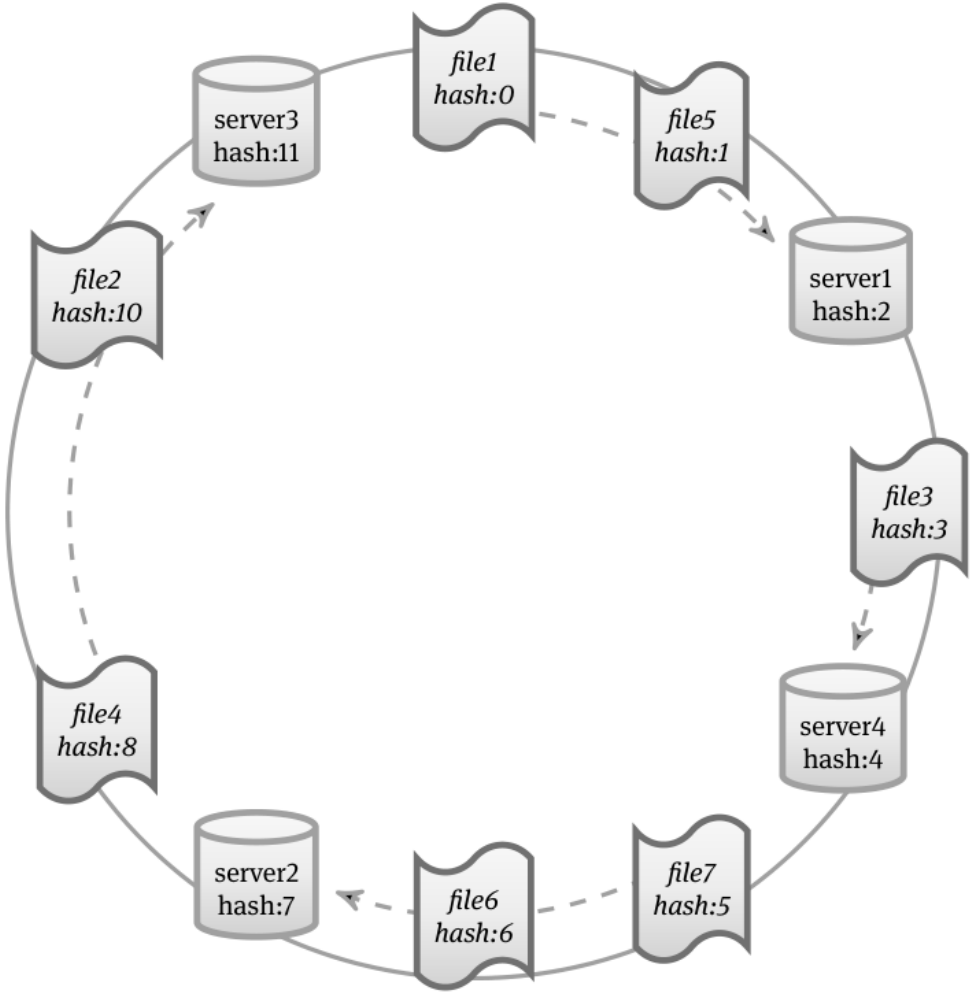


Fig. 11.5. Server addition with consistent hashing

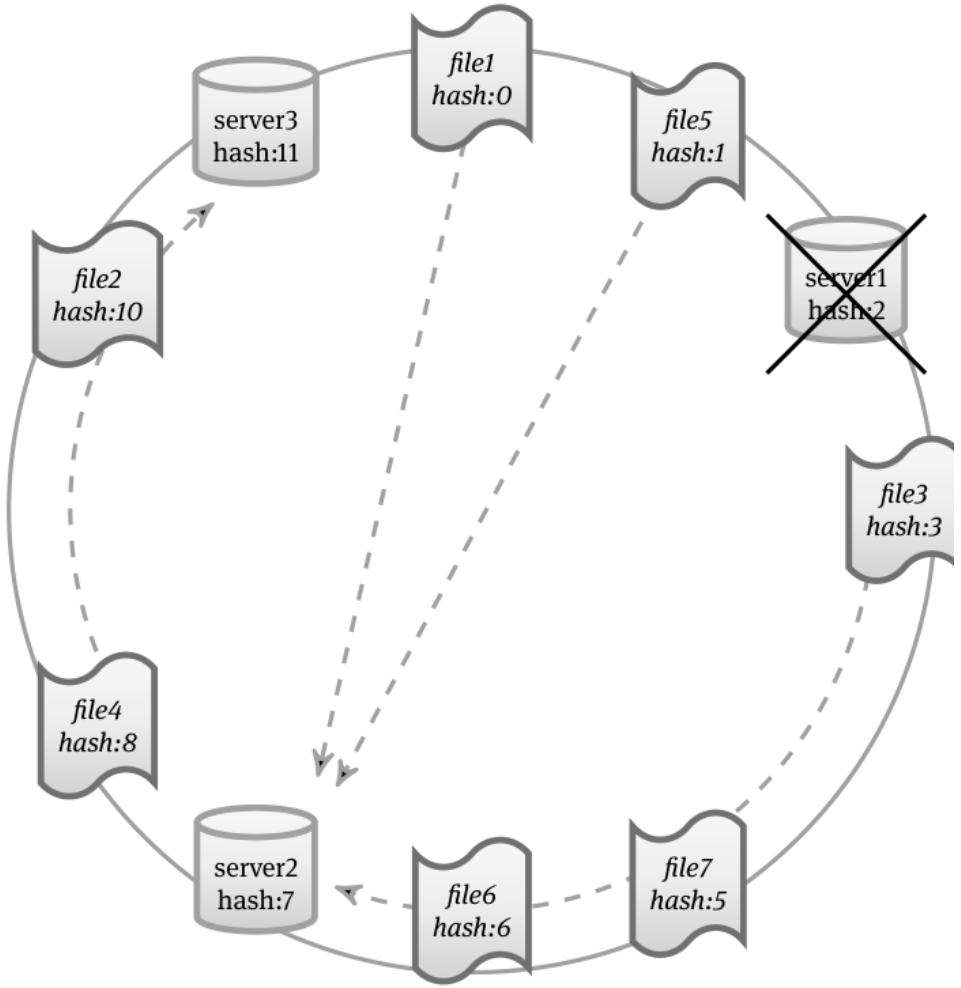


Fig. 11.4. Server removal with consistent hashing

Consistent Hashing Variations

"Virtual Servers"

- Each server has multiple IDs
- Each ID leads to responsibilities
- Stronger server → More IDs
- Weaker server → Less IDs

- Gradual entry → A new node takes IDs up bit by bit until it's full

Dynamic Reassignment

- Divide the ring into X partitions for X servers
- Server enters → Reallocate partitions for all
- Server exits → Reallocate partitions for remaining

Backups

- Multiple assignments of IDs per entity
- Server nodes with overlapping responsibilities

Replication Techniques: Master-Slave

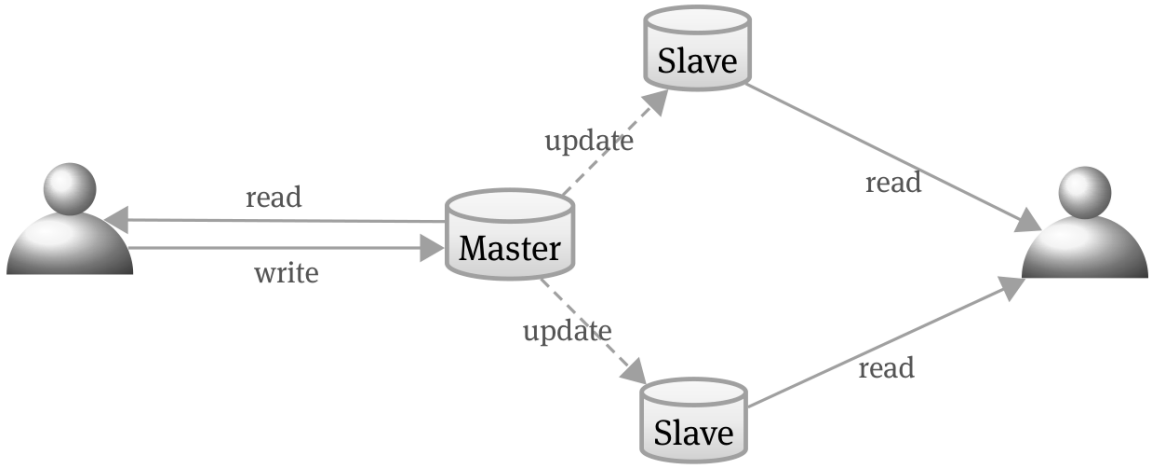


Fig. 12.1. Master-slave replication

- A single node handles all writes (master)
- It updates all replicas (slaves)
- Read from any node

- Slows things down a lot – can make a Master per partition
  - Smaller area of responsibility

Master-Slave Replication

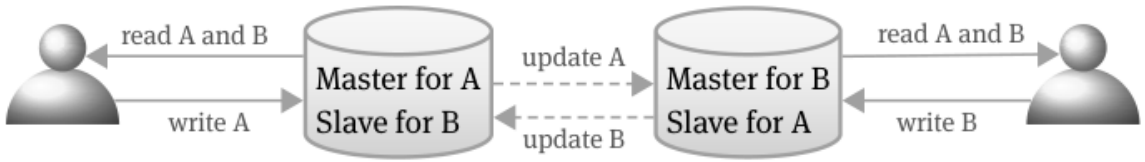


Fig. 12.2. Master-slave replication with multiple records

- Here A and B are Masters of different parts
- Can break responsibility down at different data levels
  - Table
  - Row
  - Data element

Multi-Master Replication

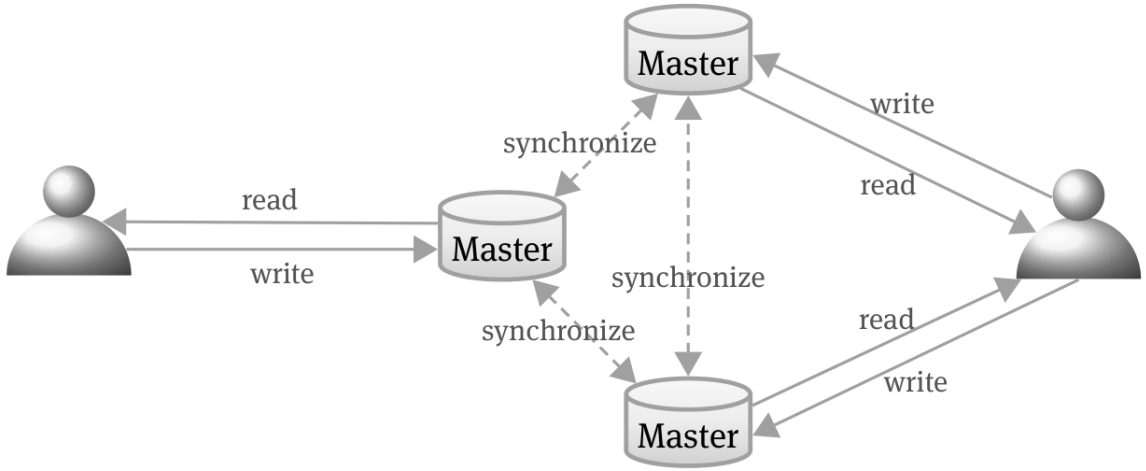


Fig. 12.3. Multi-master replication

- Peer-to-peer replication
- Any where can read or write
- Consistency?
- How do nodes update each other to prevent stale reads?
  - Remember epidemics?

Synchronization Problems

Master-Slave

Master fails

Master fails before it updates all slaves

Slave takes over, the master returns

Master fails, Slave takes over, Slave fails, Master returns

Multi-Master

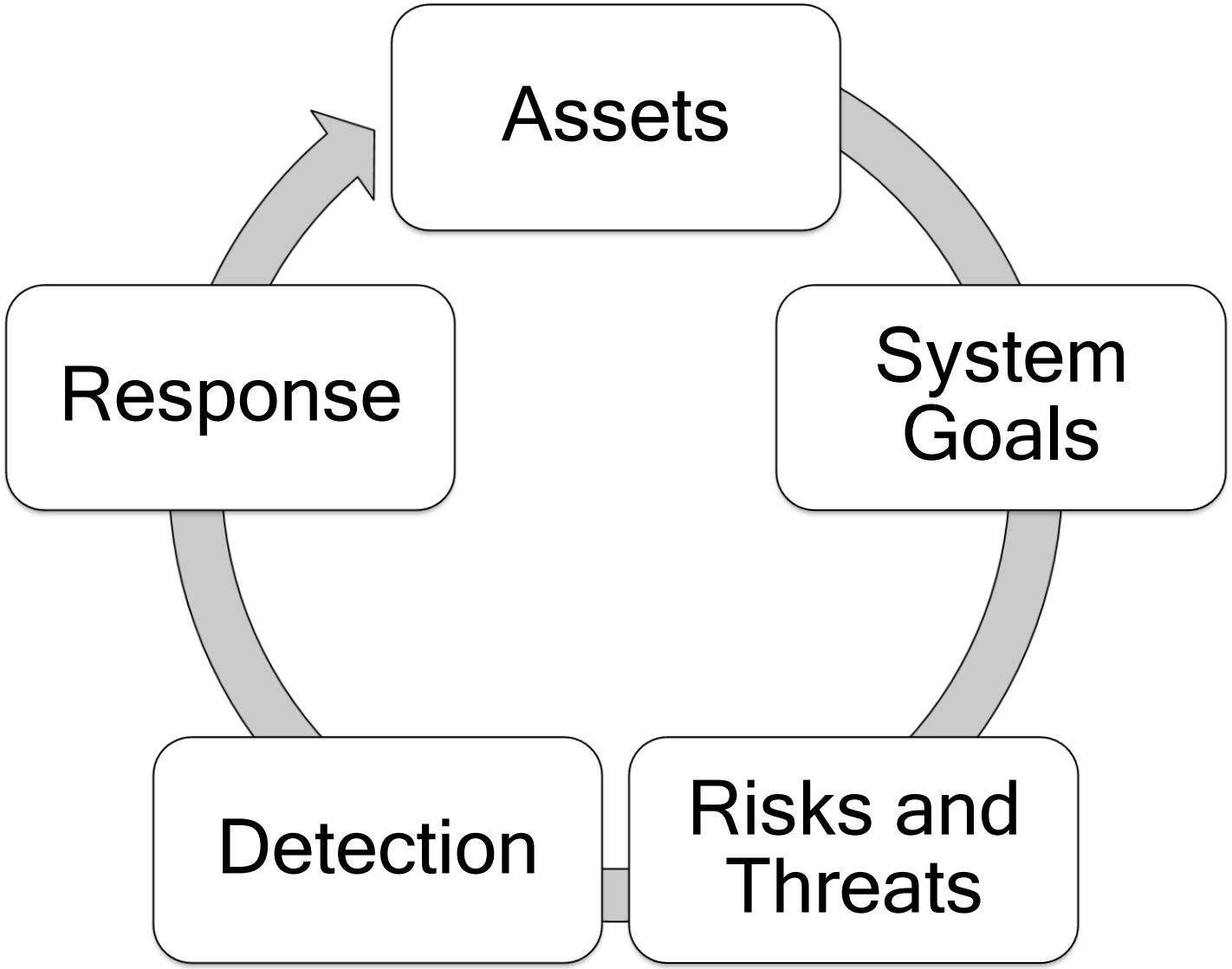
Master 1 receives update, fails before Master 2 gets update

Master 1 and Master 2 receive updates at same time

So Far

- Mutual Exclusion
  - Using Zookeeper
- Elections
- Distributed Databases
- Cyber: Storage and Naming

Distributed Data Storage Cyber Security



Assets – Distributed Databases

Database Servers	Network Attached Storage (NAS)	Laptop hard drives	Desktop hard drives
Cloud databases	Hosted databases	File servers	Cloud document servers
	Image and media storage	Protected personal information (PII)	

Etc.

Assets – Naming Services

Name storage	Name resolution mechanisms	Name resolution servers
Mappings from names to resources	Trackers of resources and name update tools	Naming policies enforcers

System Goals

Data accessibility by users
- on-premises
- Off-premises

Data access
- From apps
- From web

Data integrity
- Who updated
- Where updated
- Why updated

Tracking of data access

Name resolution to resources

Compliance with
- Privacy laws and rules
- Internal organizational rules

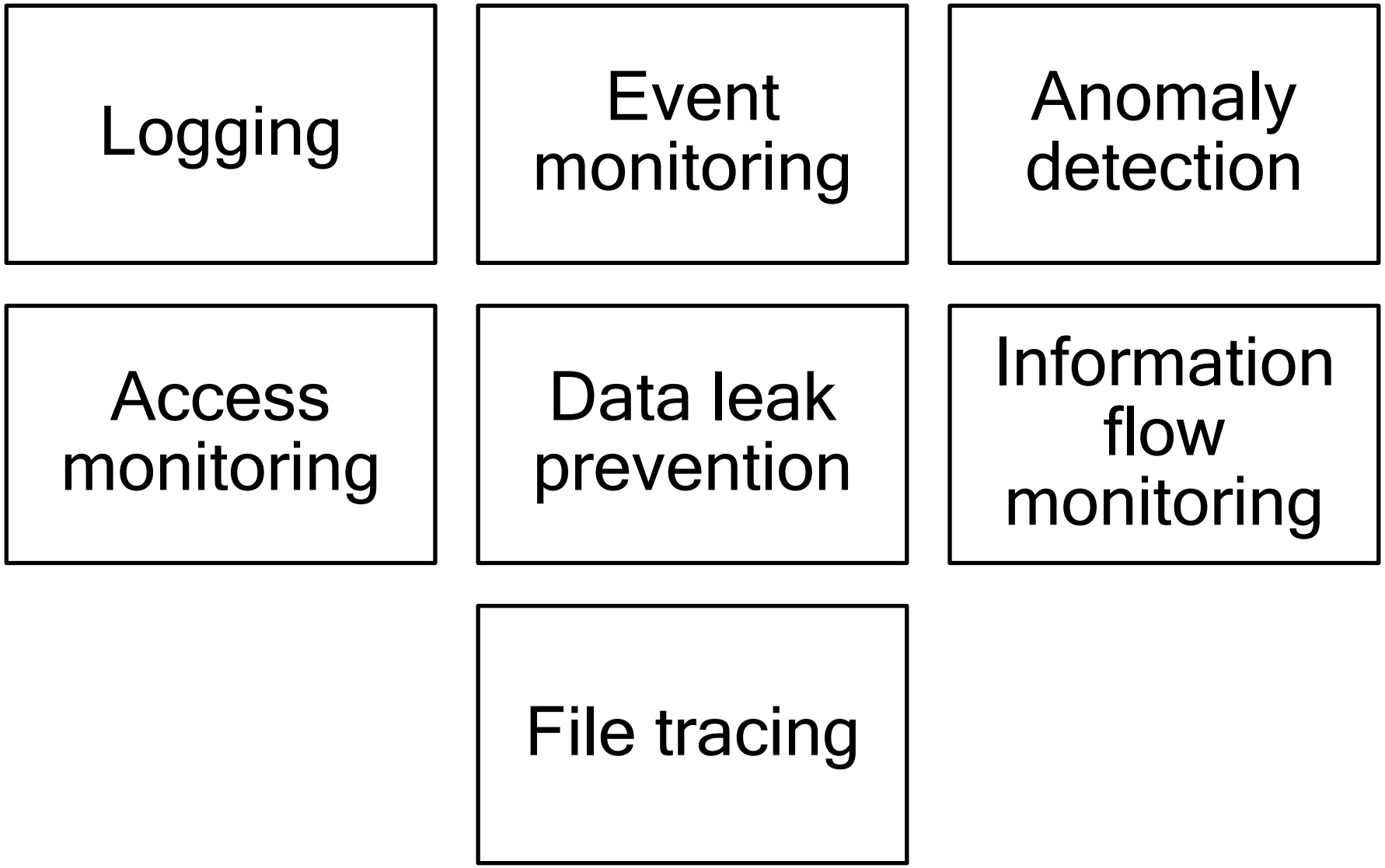
Auditing
- Access by authorized users
- Access by non-authorized users

Logging
- Event detection
- Anomaly detection
- Warnings

Risks and Threats

Unauthorized data access	Data exfiltration	Data corruption • Data encryption • Ransomware
Insider access to protected data	Inconsistent data protection policies	Insufficient data protection/encryption
Loss of data control	Loss of data access	Naming services failure

Detection


Logging
Event monitoring
Anomaly detection
Access monitoring
Data leak prevention
Information flow monitoring
File tracing

Response

Monitor network anomalies

- Close unknown connection endpoints
- Prevent large data transfers to unknown targets
- Deep packet inspection to find data exiting

Data fingerprinting to watch for exfiltration

User monitoring

Access monitoring

Automatic event notification and log notification

Some stories

+1

Akamai

Products Solutions Why Akamai Resources Partners Contact Us

Blog > Security > Ransomware Devastating MySQL Servers

Ransomware Devastating MySQL Servers

Ophir Harpaz

https://www.akamai.com/blog/security/please-read-me-opportunistic-ransomware-devastating-mysql-servers

Ransomware attacks on storage happen all the time

https://spin.ai/resources/ransomware-tracker/

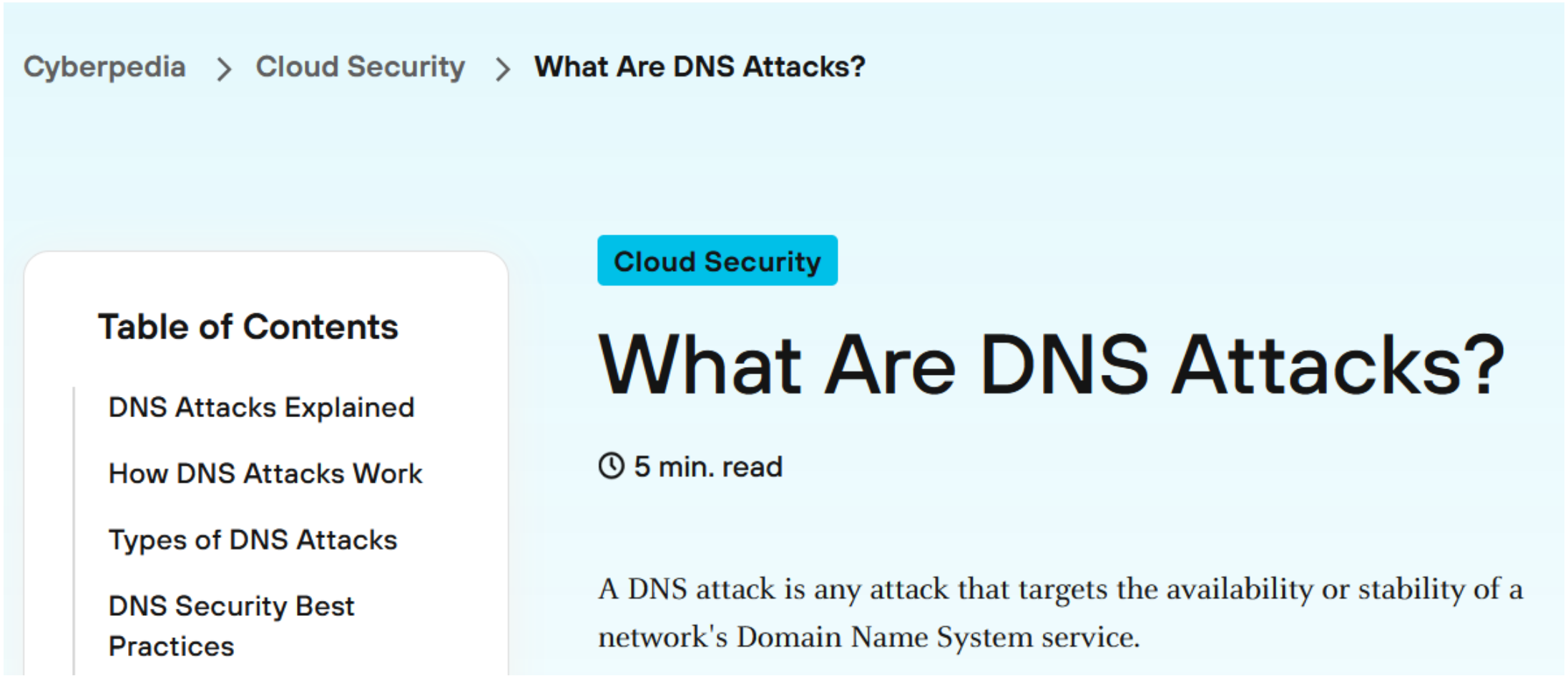
Spin.ai Platform Pricing Solutions Resources Partners Company

Ransomware Tracker (2014 – 2025)

Name	State or Country	Date	Type
SRP Federal Credit Union	South Carolina	12/23/2024	Finances
Telecom Namibia	Namibia	12/16/2024	Communication Technologies
Compass Communications	New Zealand	12/13/2024	Communication Technologies
Rhode Island State Benefits System	Rhode Island	12/13/2024	Government
Taylor Regional Hospital	Georgia	12/13/2024	Healthcare
Ainsworth Game Technology	Australia	12/12/2024	Manufacturing
WACER	Australia	12/12/2024	Cleaning
Fresh Produce Safety Centre Australia & New Zealand	Australia	12/12/2024	Food
National Museum of the Royal Navy	UK	12/12/2024	Museum

DNS under attack

https://www.paloaltonetworks.com/cyberpedia/what-is-a-dns-attack



DNS under attack

Bright Product Resources Partners Company

Resources > Blog > What is DNS Attack and How To Prevent Them

Published: May 29, 2022

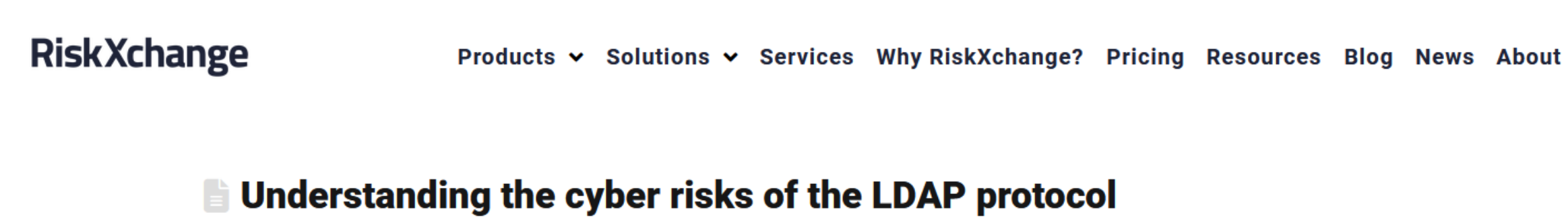
What is DNS Attack and How To Prevent Them

Admir Dizdar

AppSec Testing

https://brightsec.com/blog/dns-attack/

LDAP Security Risks



https://riskxchange.co/4821/cyber-risks-of-ldap-protocol/

https://www.upguard.com/blog/ldap

Conclusion

- Mutual Exclusion
  - Using Zookeeper
- Elections
- Distributed Databases
- Cyber: Storage and Naming