Eventual Consistency

Immanuel Trummer
itrummer@cornell.edu
www.itrummer.org

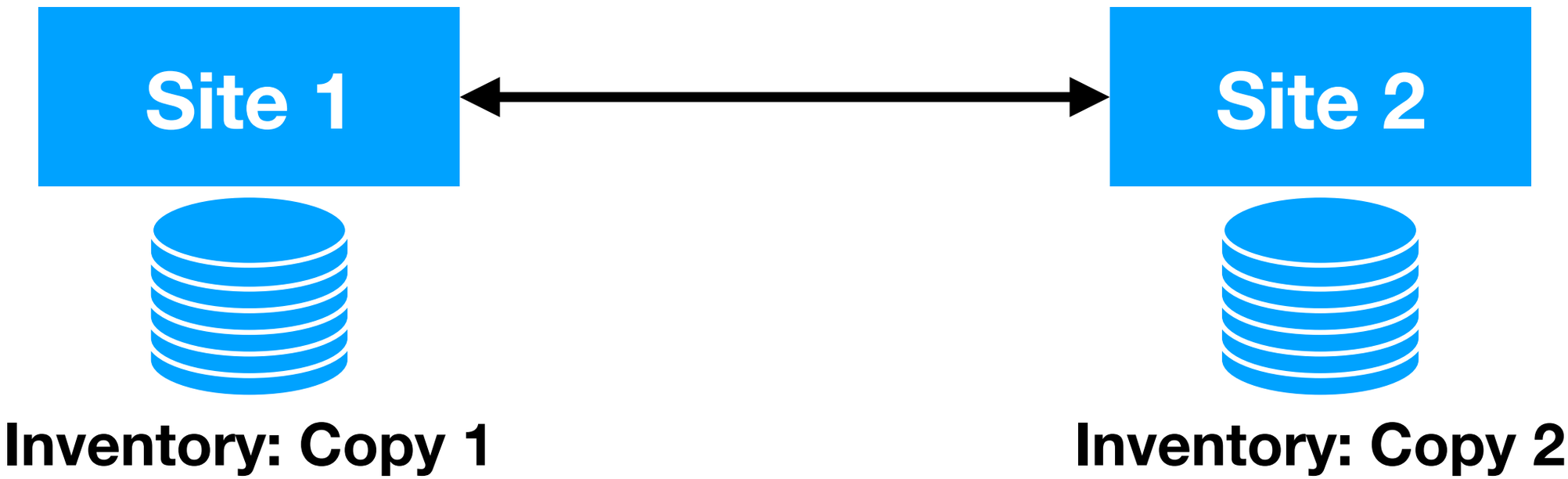
(Optional) Reading List

- https://cassandra.apache.org/
- **Dynamo: Amazon's Highly Available Key-value Store**, DeCandia at al, SOSP 2007
- **Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: the Consumers' Perspective**, H. Wada et al, CIDR 2011

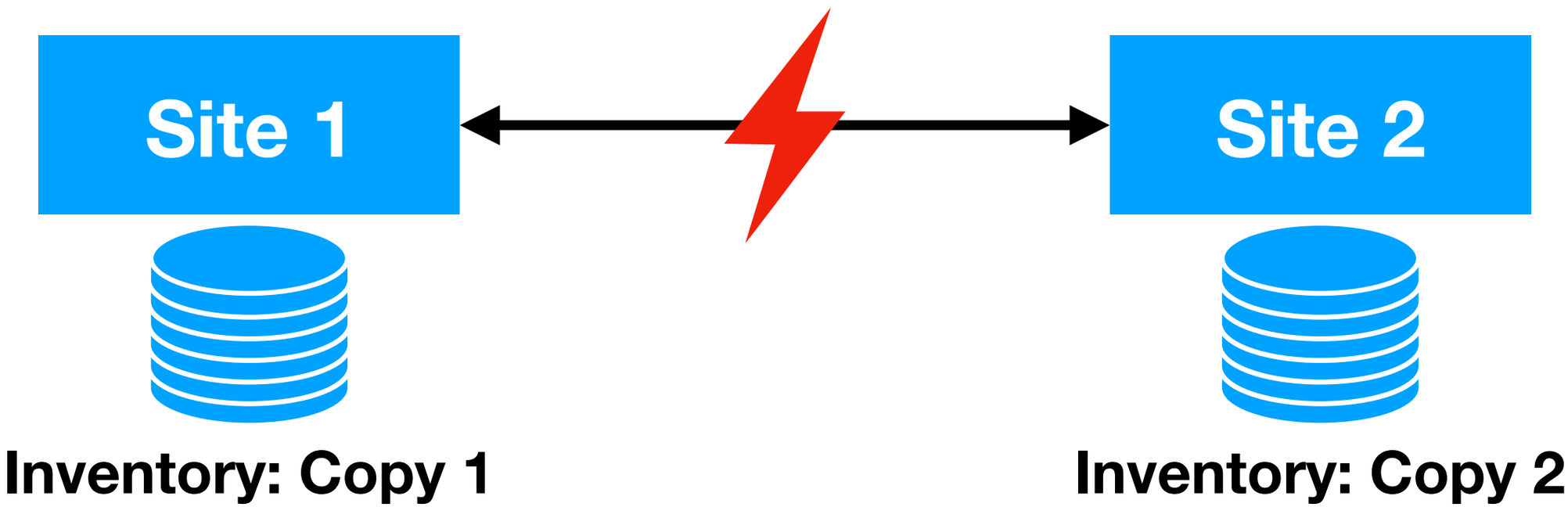
Terminology Warning

- Consistency so far: database satisfies all **constraints**
- Now: consistency means all replicas are in **sync**
- Terminology from **distributed systems** community

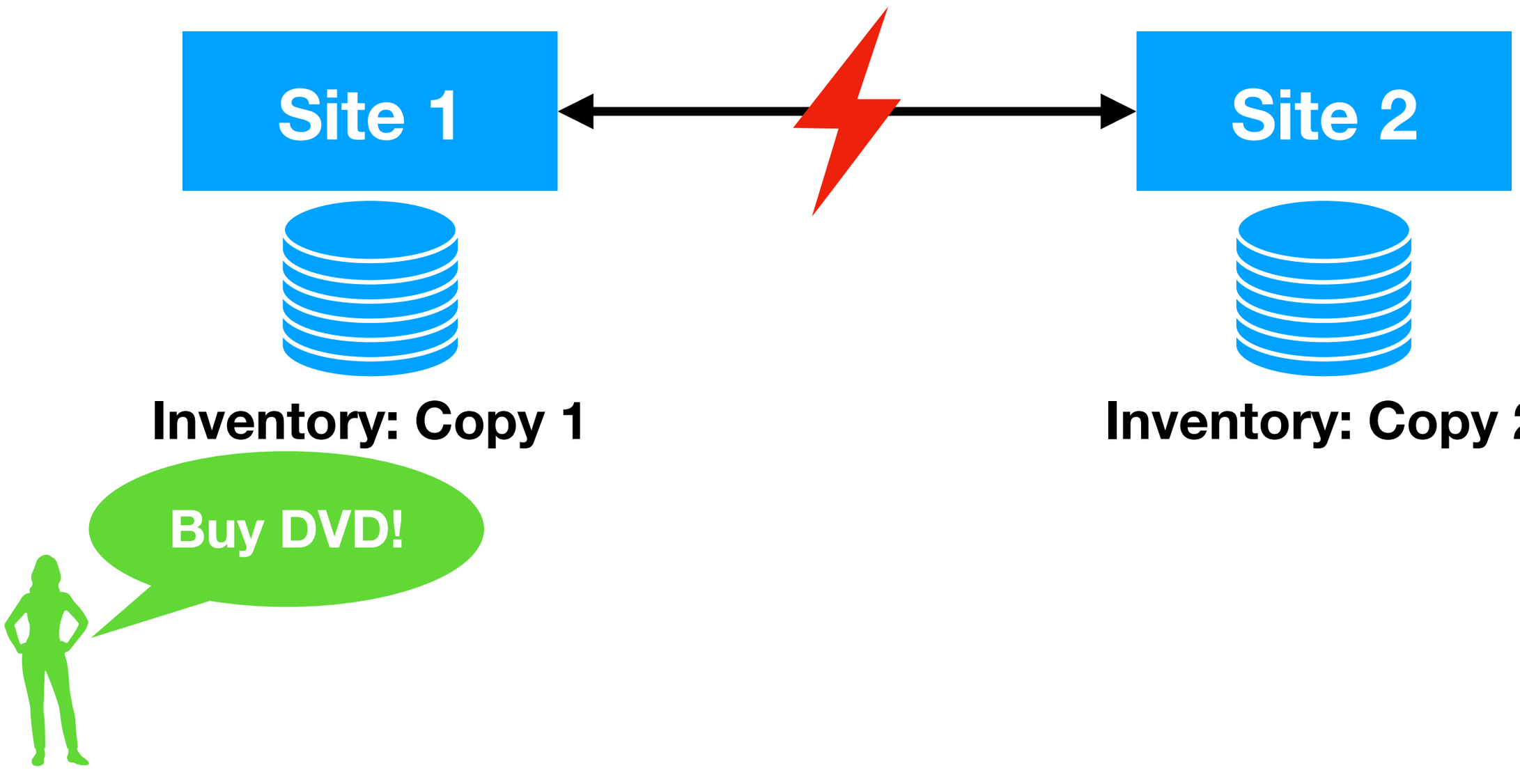
Consistency vs. Availability



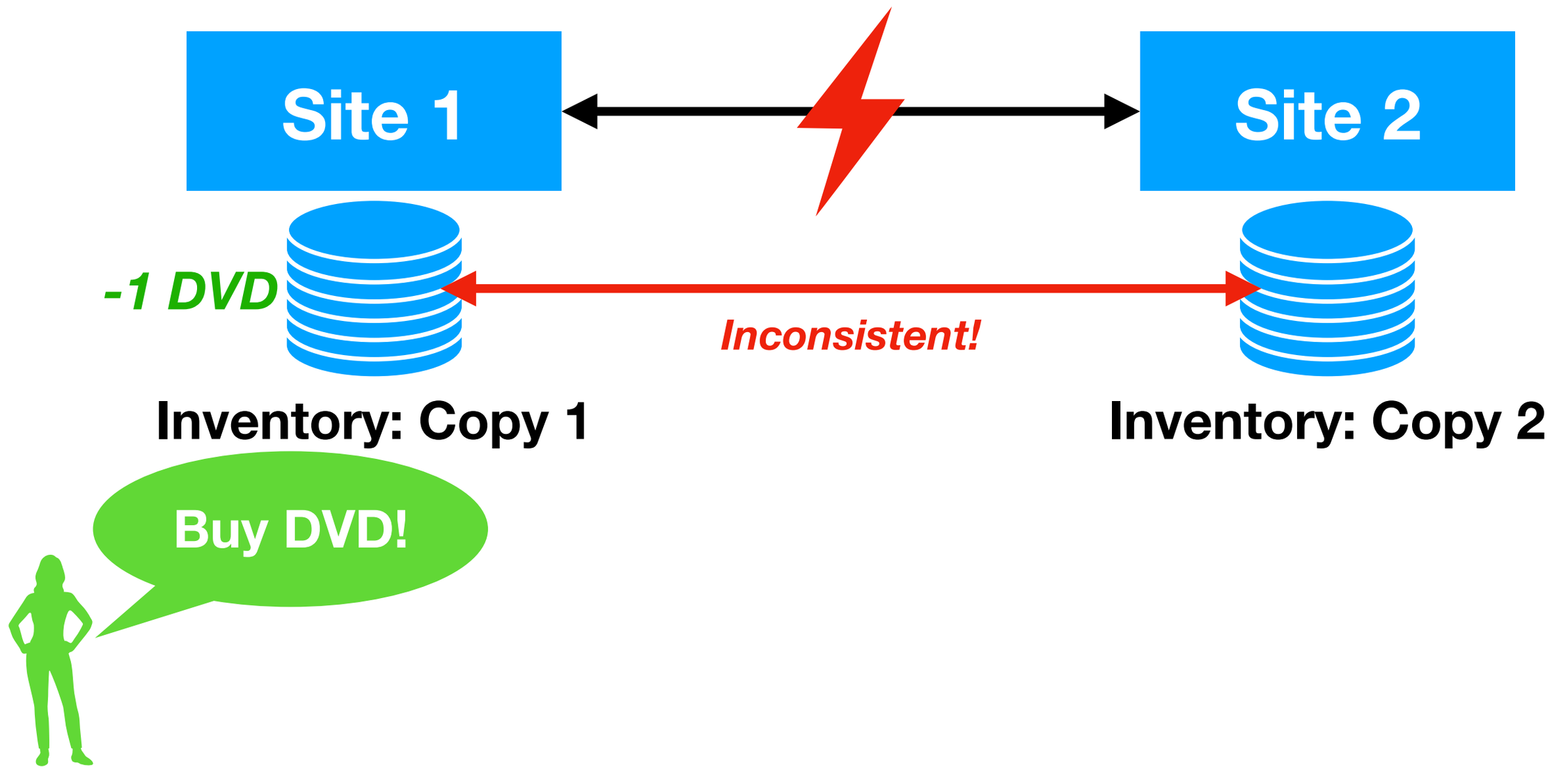
Consistency vs. Availability



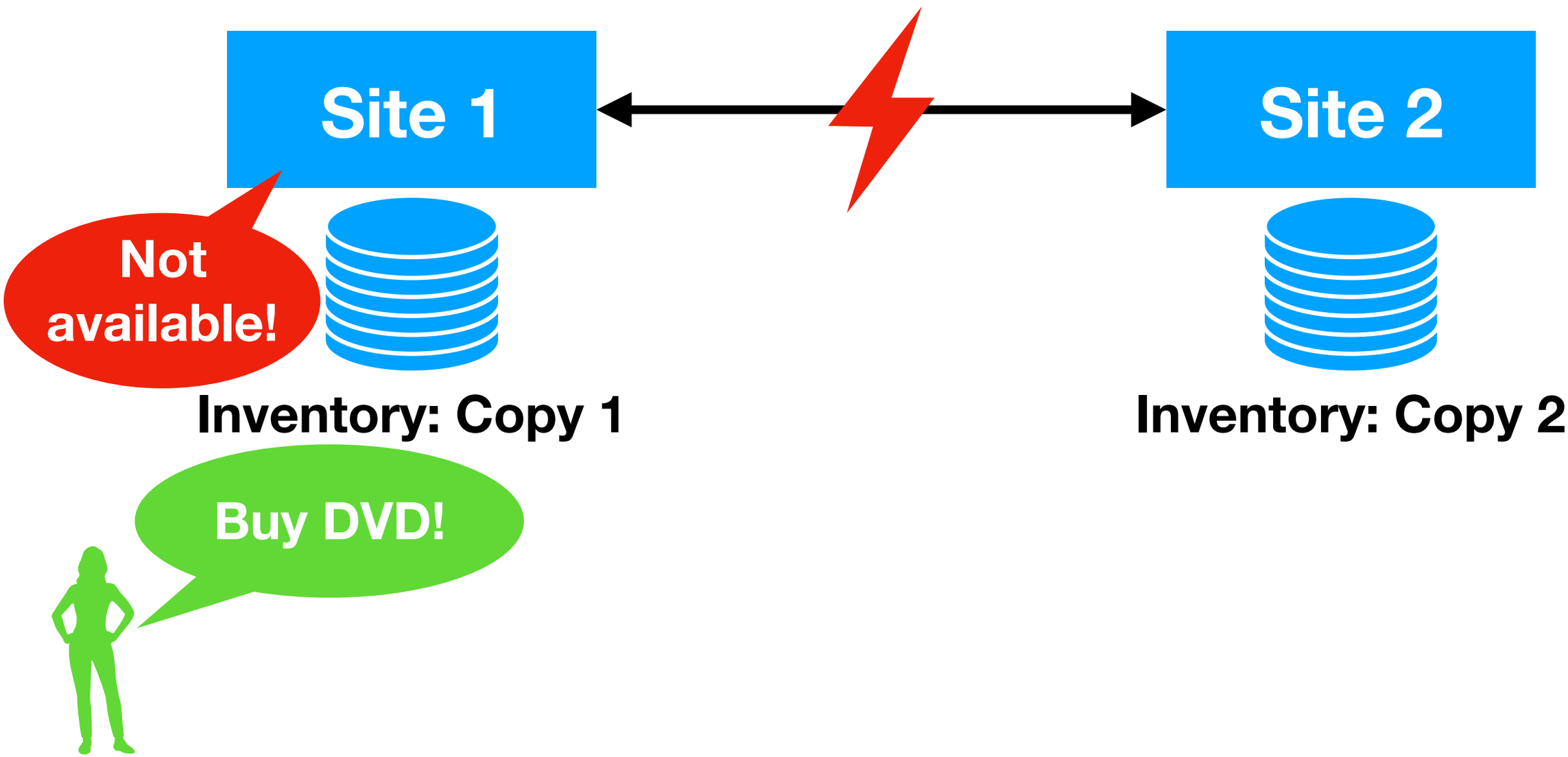
Consistency vs. Availability



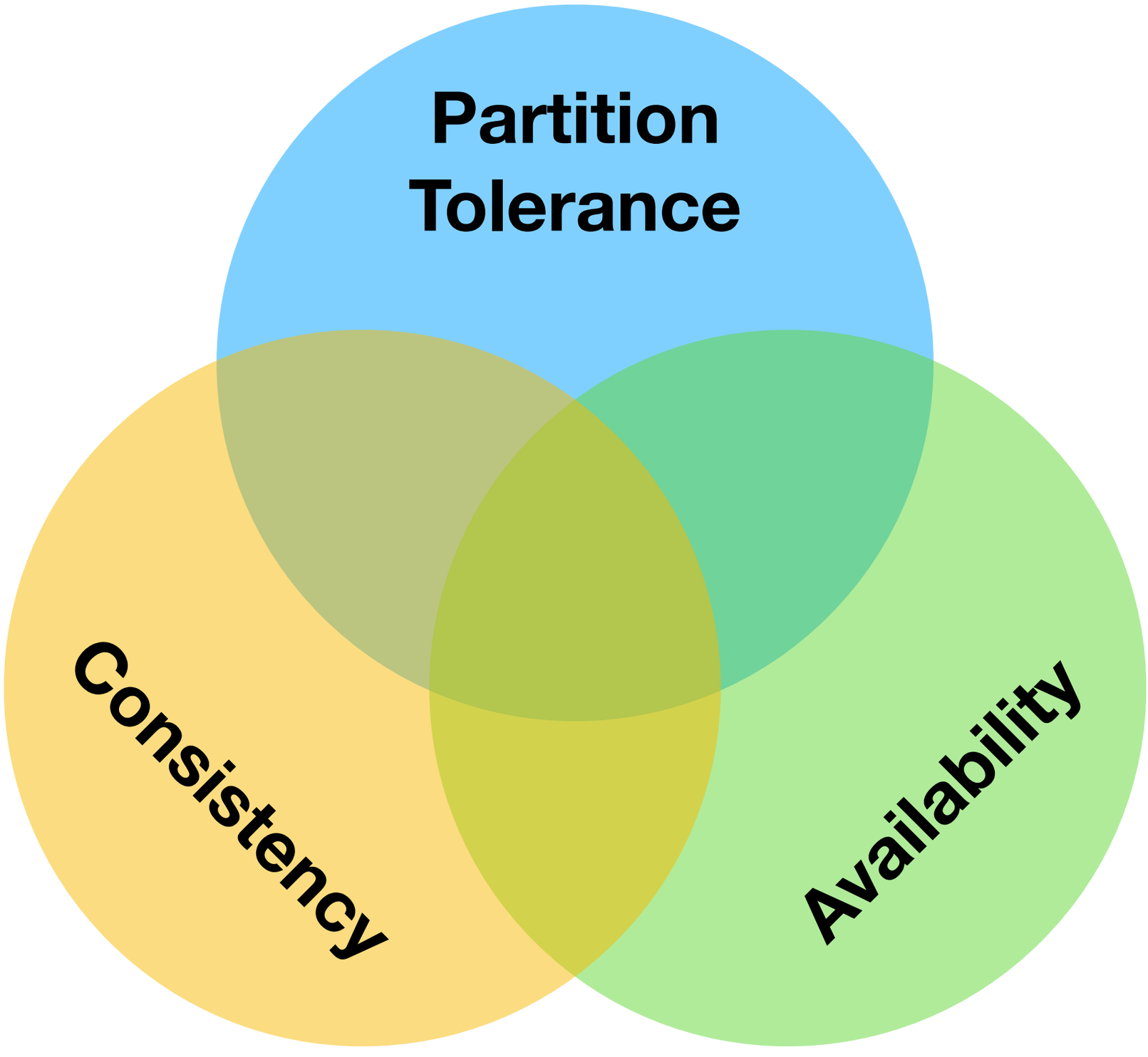
Consistency vs. Availability



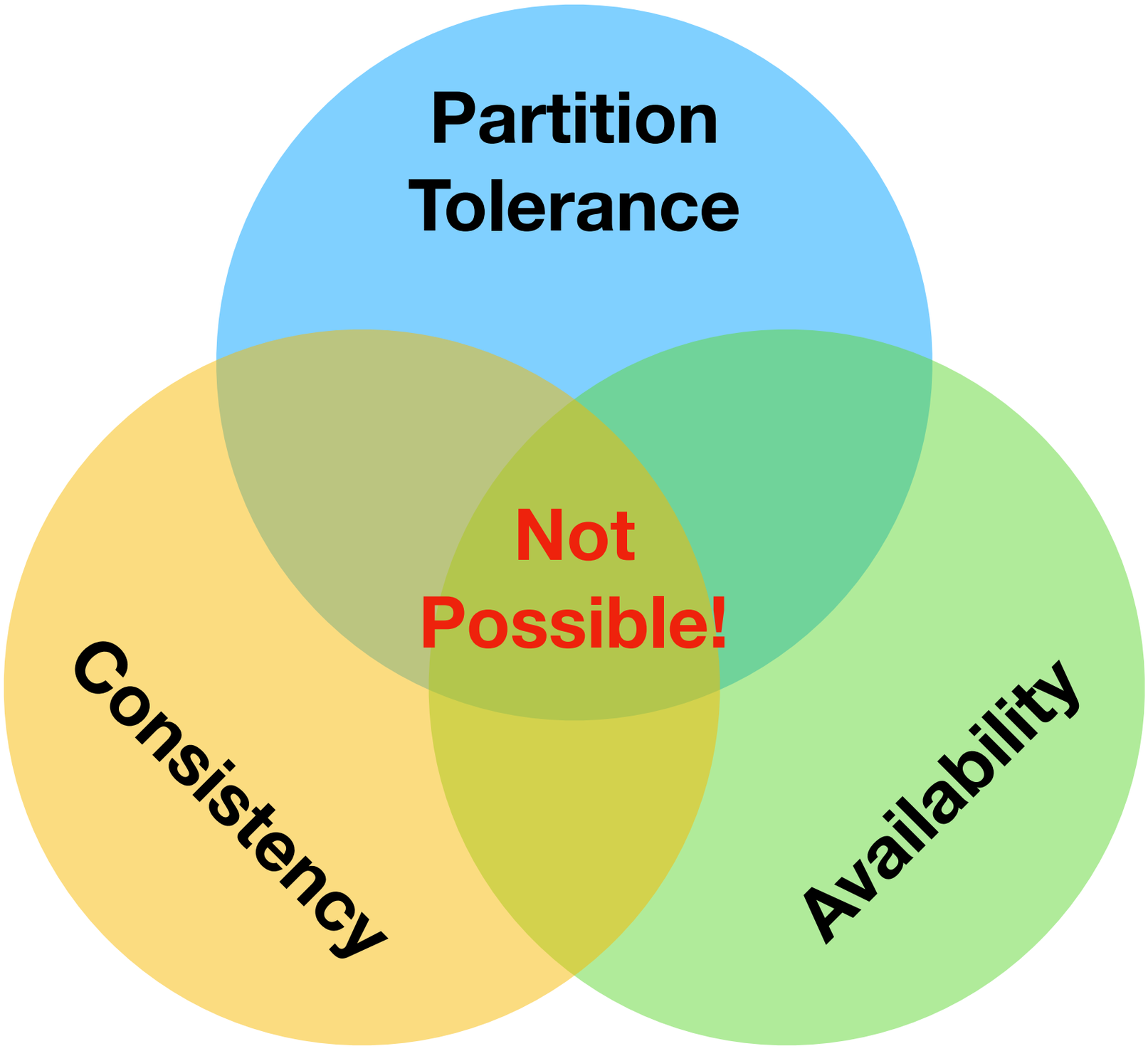
Consistency vs. Availability



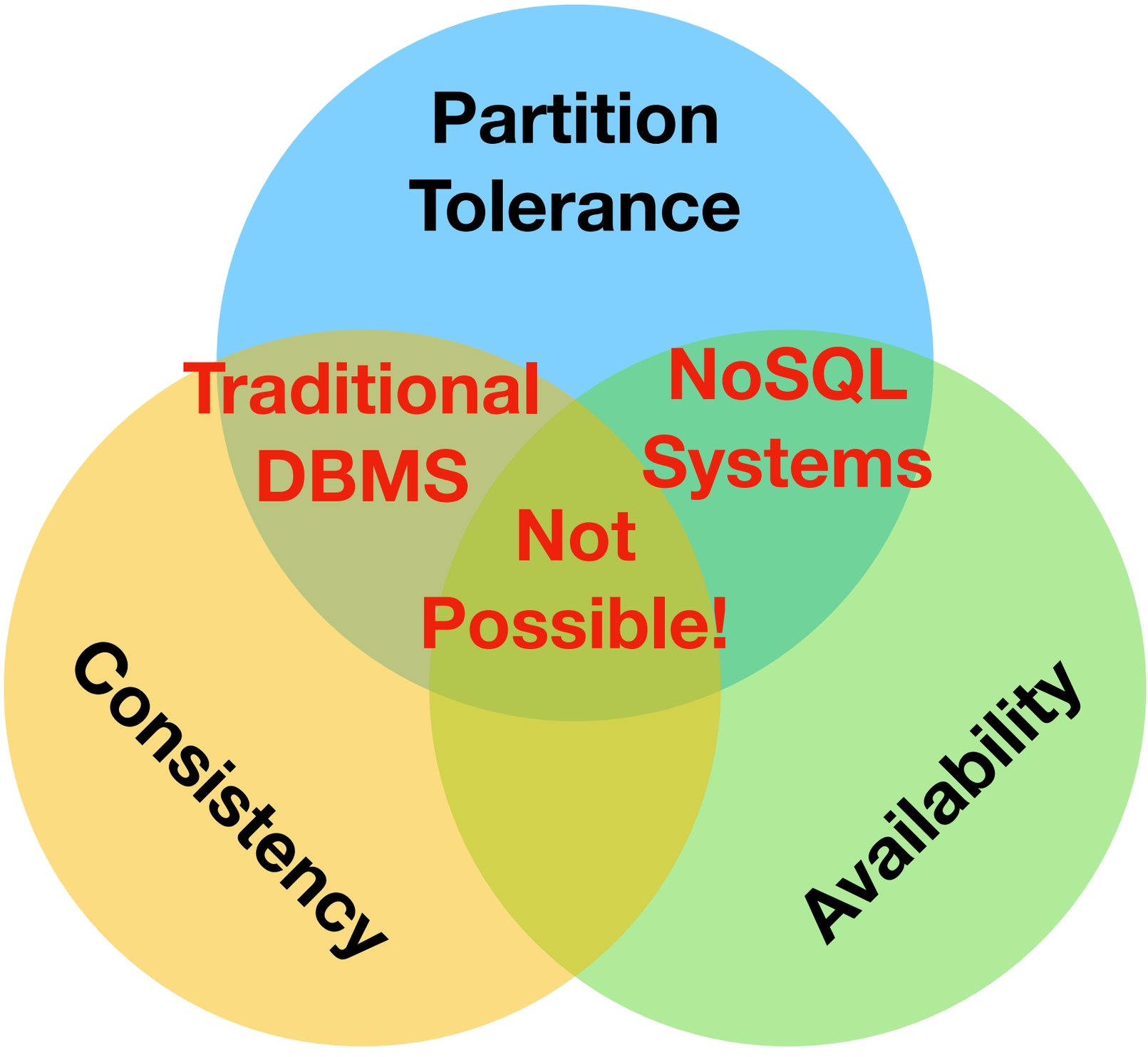
The CAP Theorem



The CAP Theorem



The CAP Theorem



Eventual Consistency

- Traditional DBMS choose **consistency** over availability
- This is **not ideal** in scenarios such as online shopping
- Here, we want to be **available** at all costs
- Need to accept **inconsistency** according to CAP
- This inconsistency is **resolved eventually**

BASE Transactions

- **BASE** =
  - **B**asically **A**vailable
  - **S**oft State
  - **E**ventually Consistent

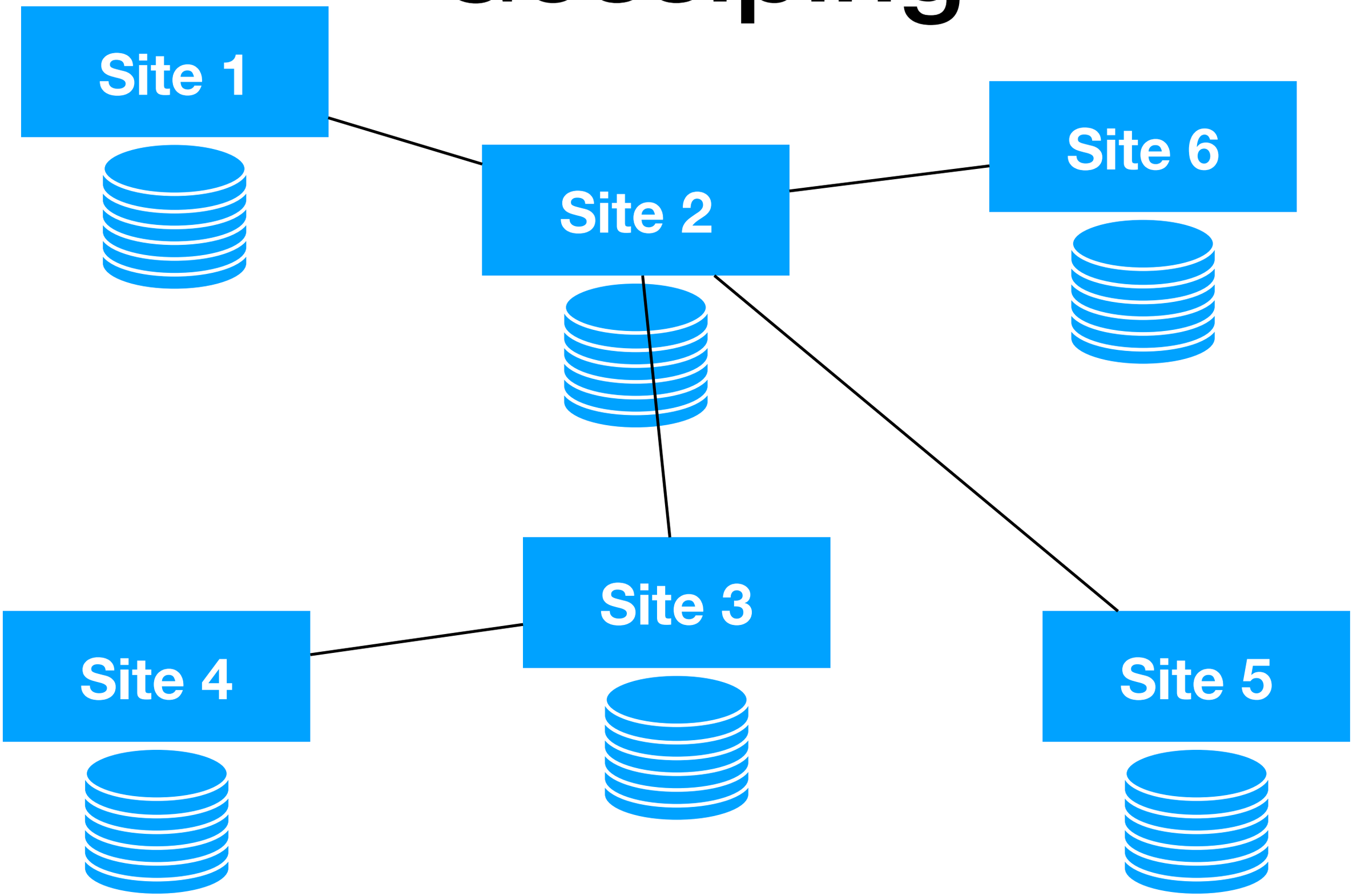
NoSQL

- Systems that **move away** from traditional SQL DBMS
- **Broad** term covering many aspects such as
  - Reduced **consistency** (BASE)
  - Non-SQL query **languages**
  - Non-relational **data models**
  - ...

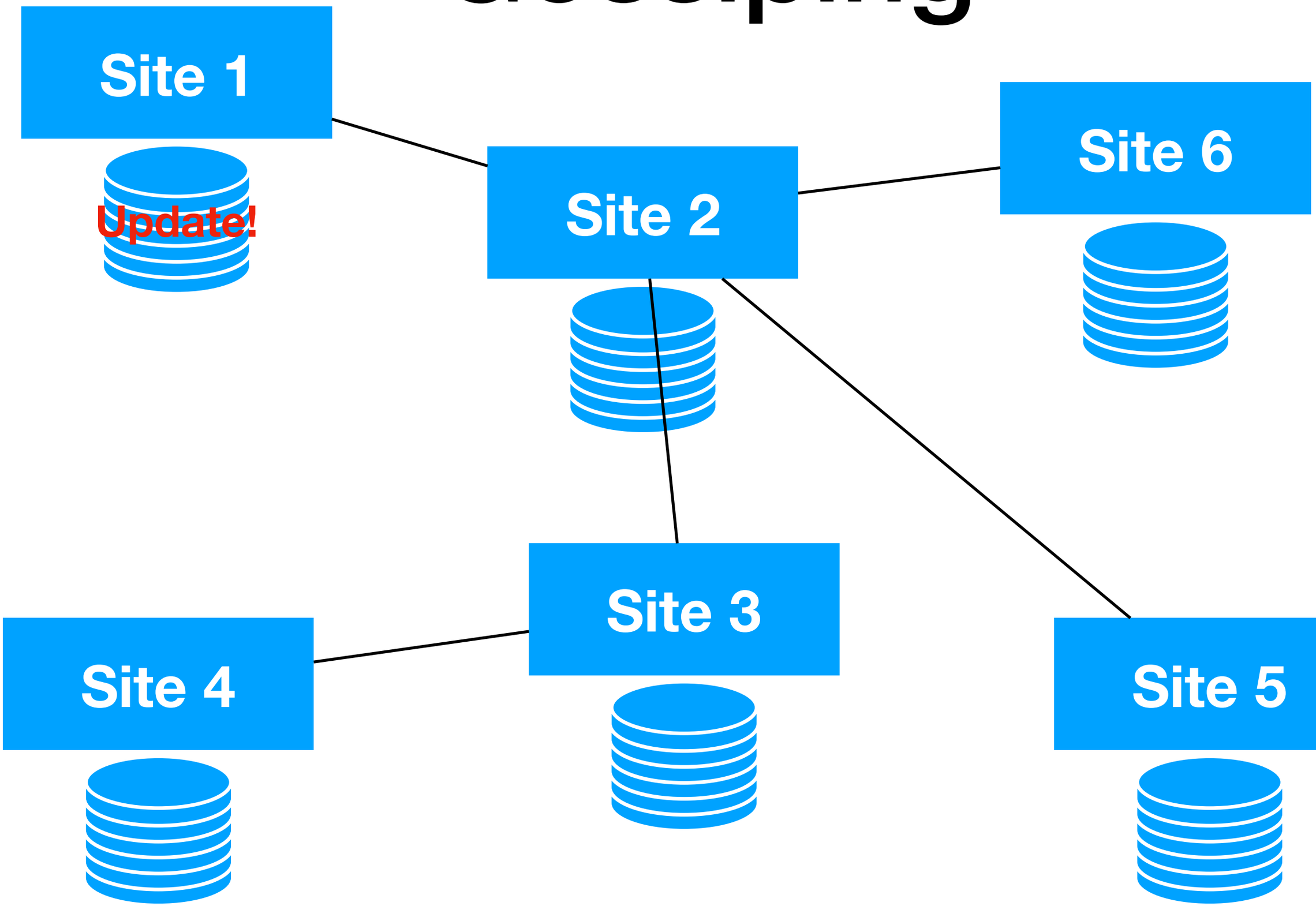
Apache Cassandra

- Distributed system, every node has the **same role**
- **Wide column** store ~ rows have different columns
- Supports **CQL**, simpler than SQL (e.g., no joins)
- Supports replication for **fault tolerance**
- Goal: **scale linearly** when adding new nodes
- Eventually consistent with **tunable consistency**

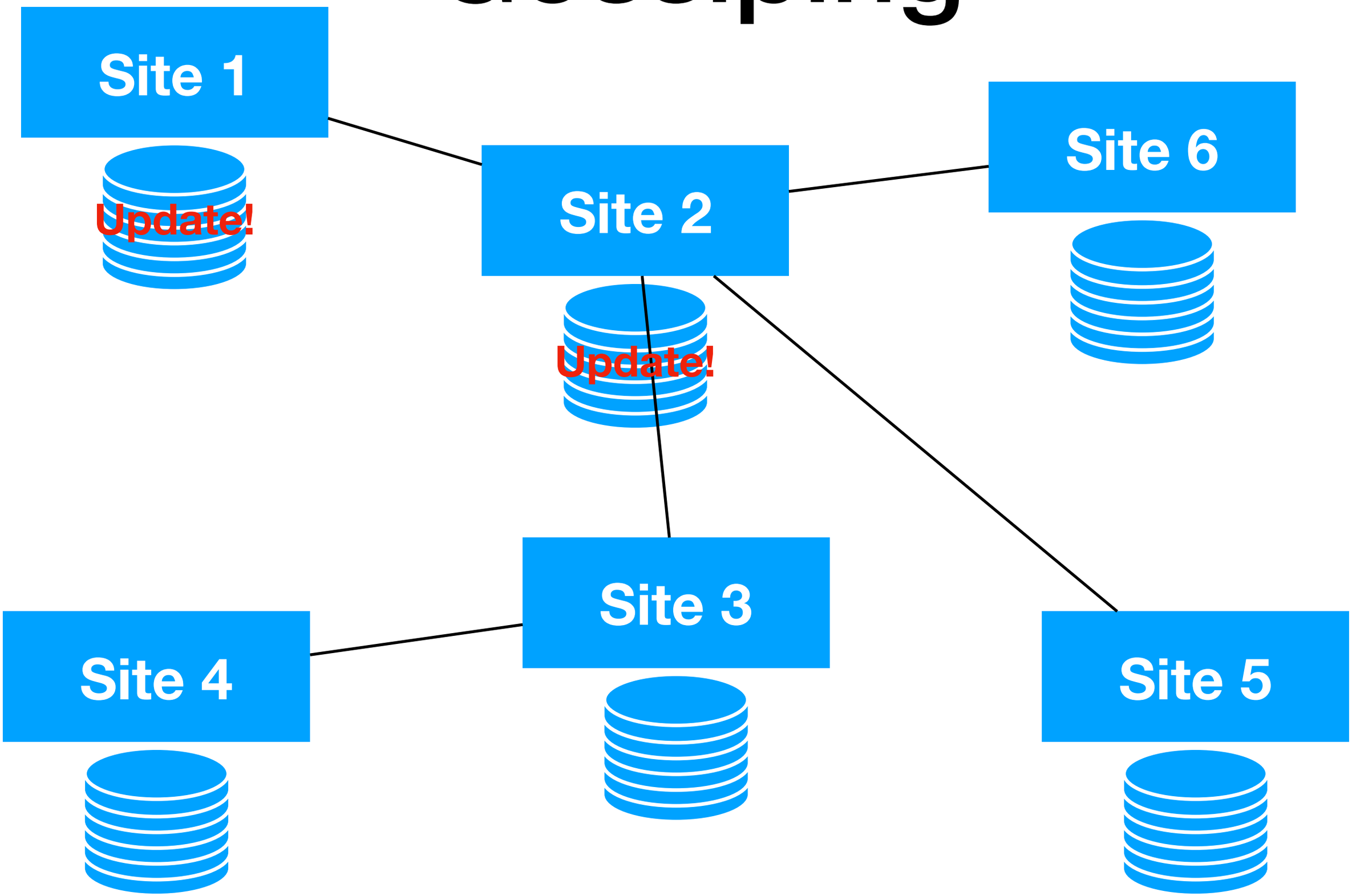
Gossiping



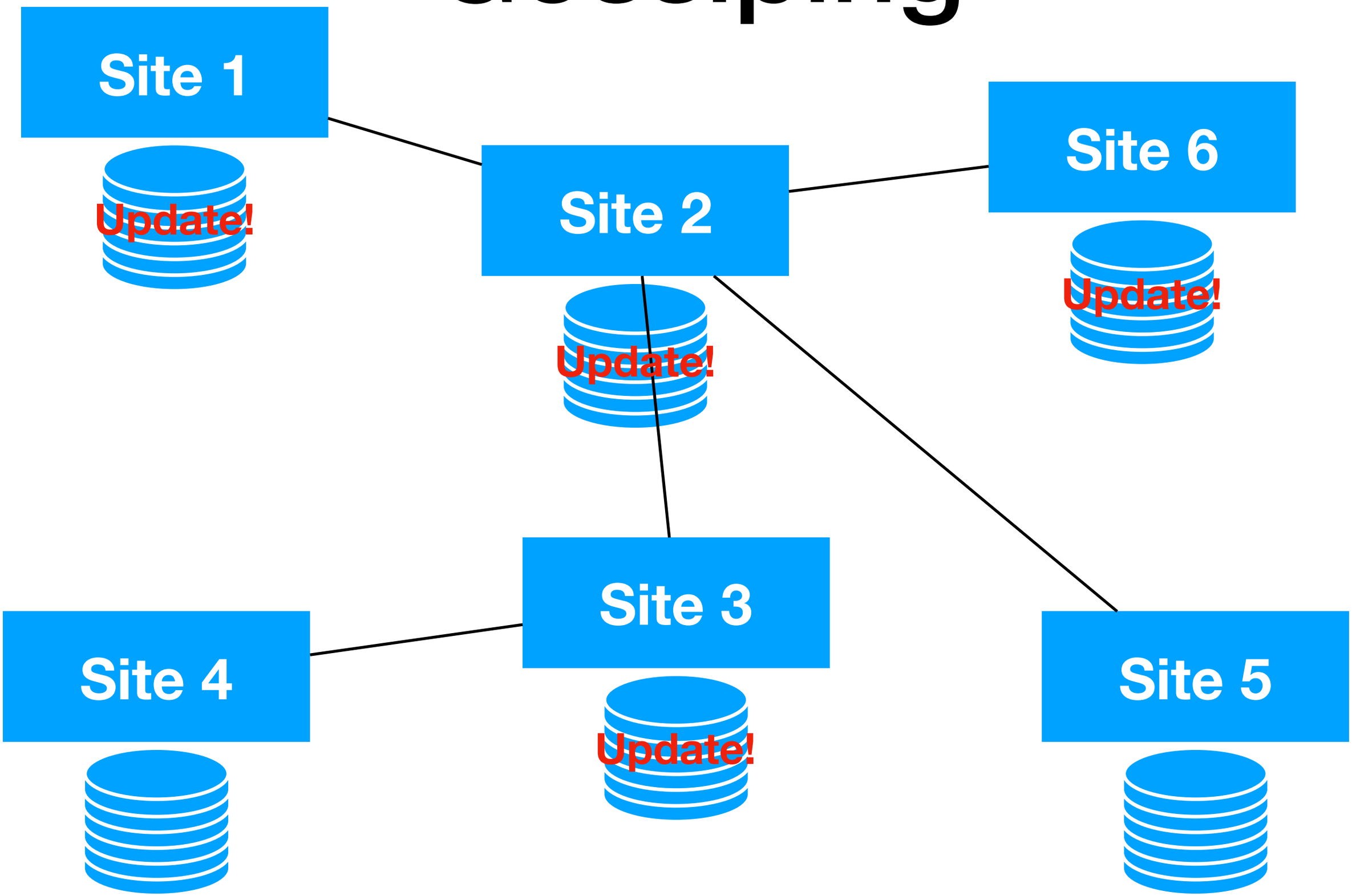
Gossiping



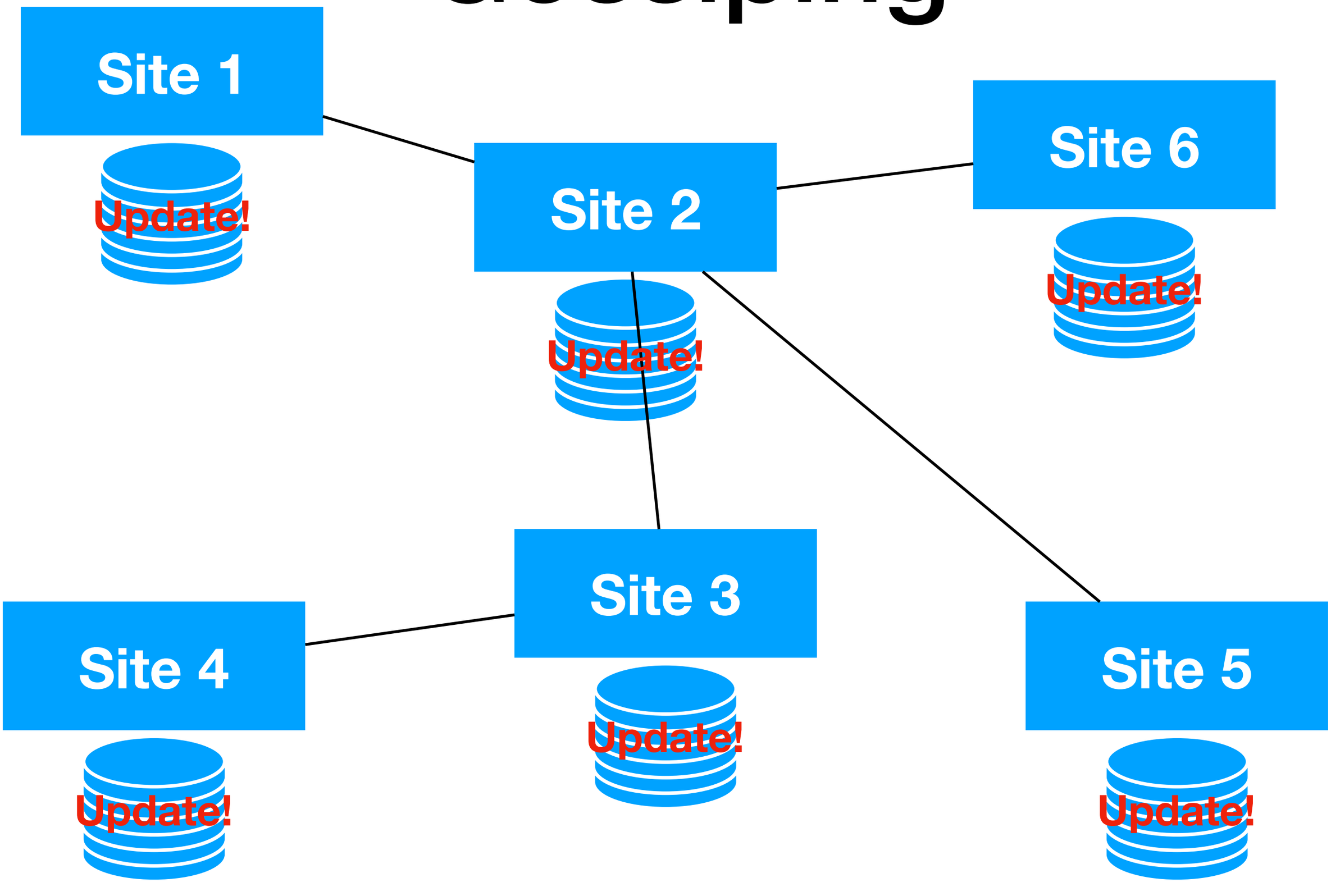
Gossiping



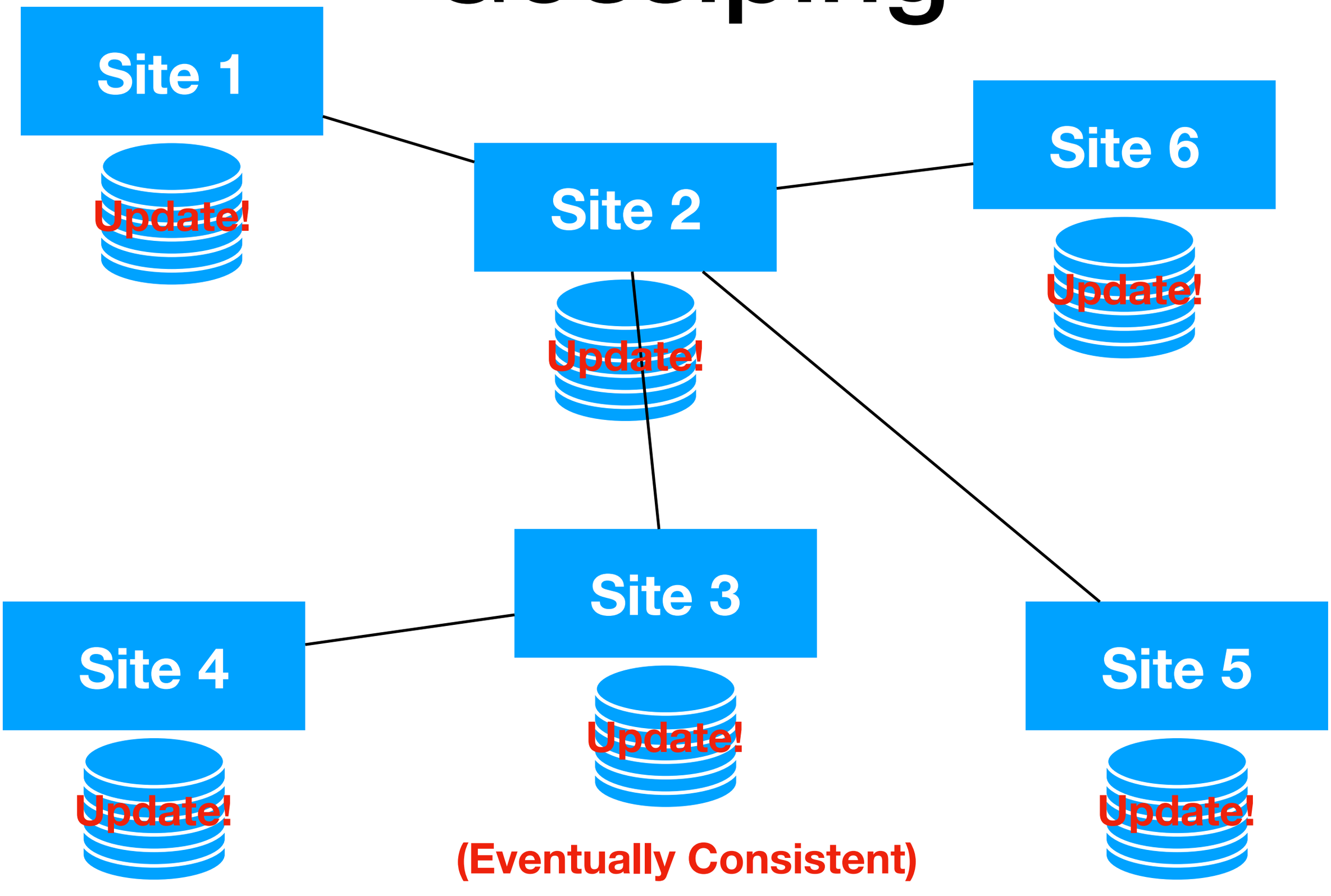
Gossiping


Site 1
Update!
Site 2
Update!
Site 6
Update!
Site 3
Update!
Site 4
Site 5

Gossiping


Site 1
Update!
Site 2
Update!
Site 6
Update!
Site 3
Update!
Site 4
Update!
Site 5
Update!

Gossiping



(Eventually Consistent)

Eventual Consistency in Cassandra

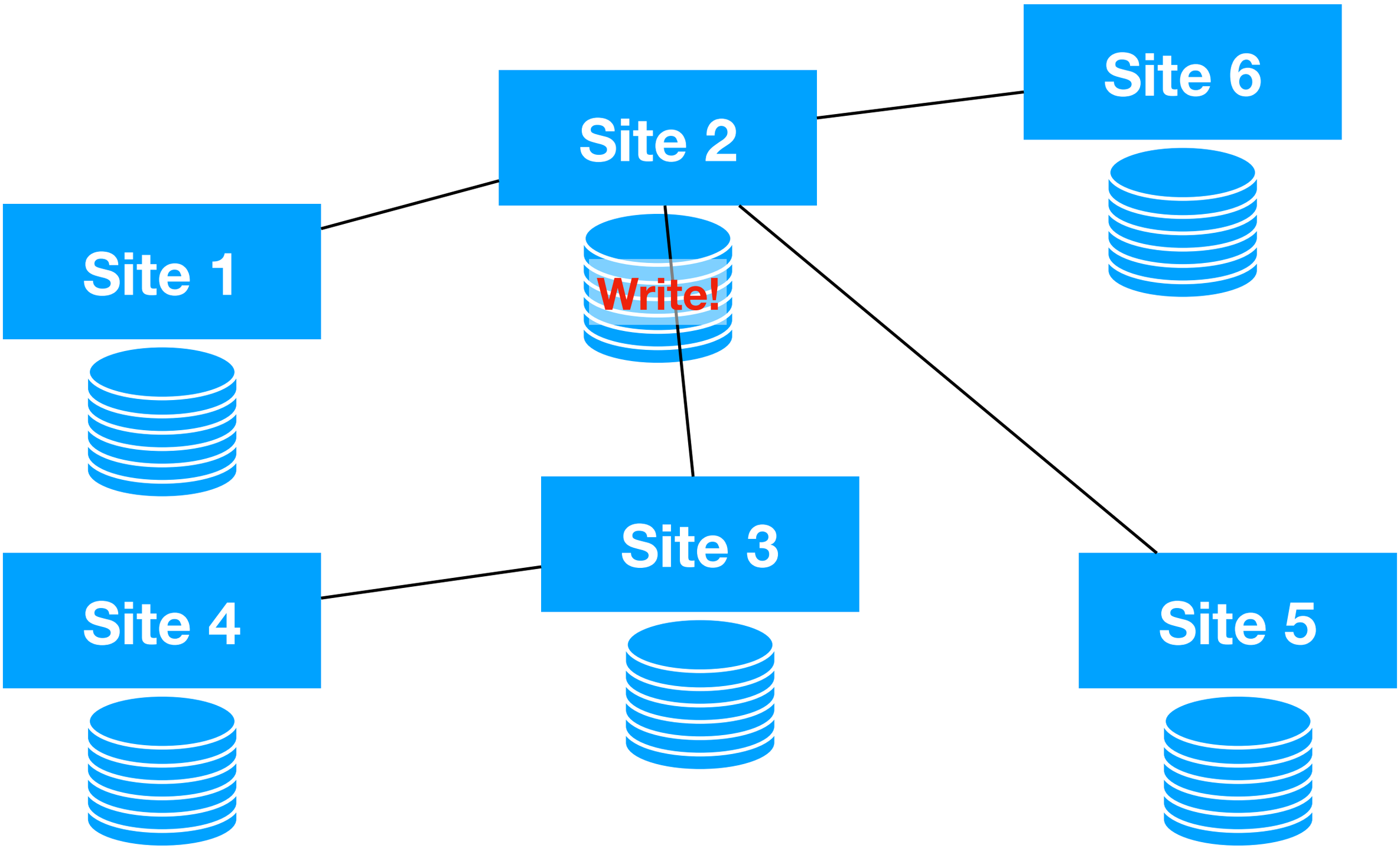
- Nodes periodically **exchange** information on updates
- Each node communicates with a **limited** number of peers
- **Eventually**, updates are propagated to all nodes
- ***Question: how should we handle deletions ... ?***

Tuning Consistency

- Cassandra allows to tune several **parameters**:
  - **Replication** factor (i.e., how many copies): N
    - Number of replicas for **successful read**: R
    - Number of replicas for **successful write**: W
- ***What must hold for N, R, and W for full (i.e., non-eventual) consistency ... ?***

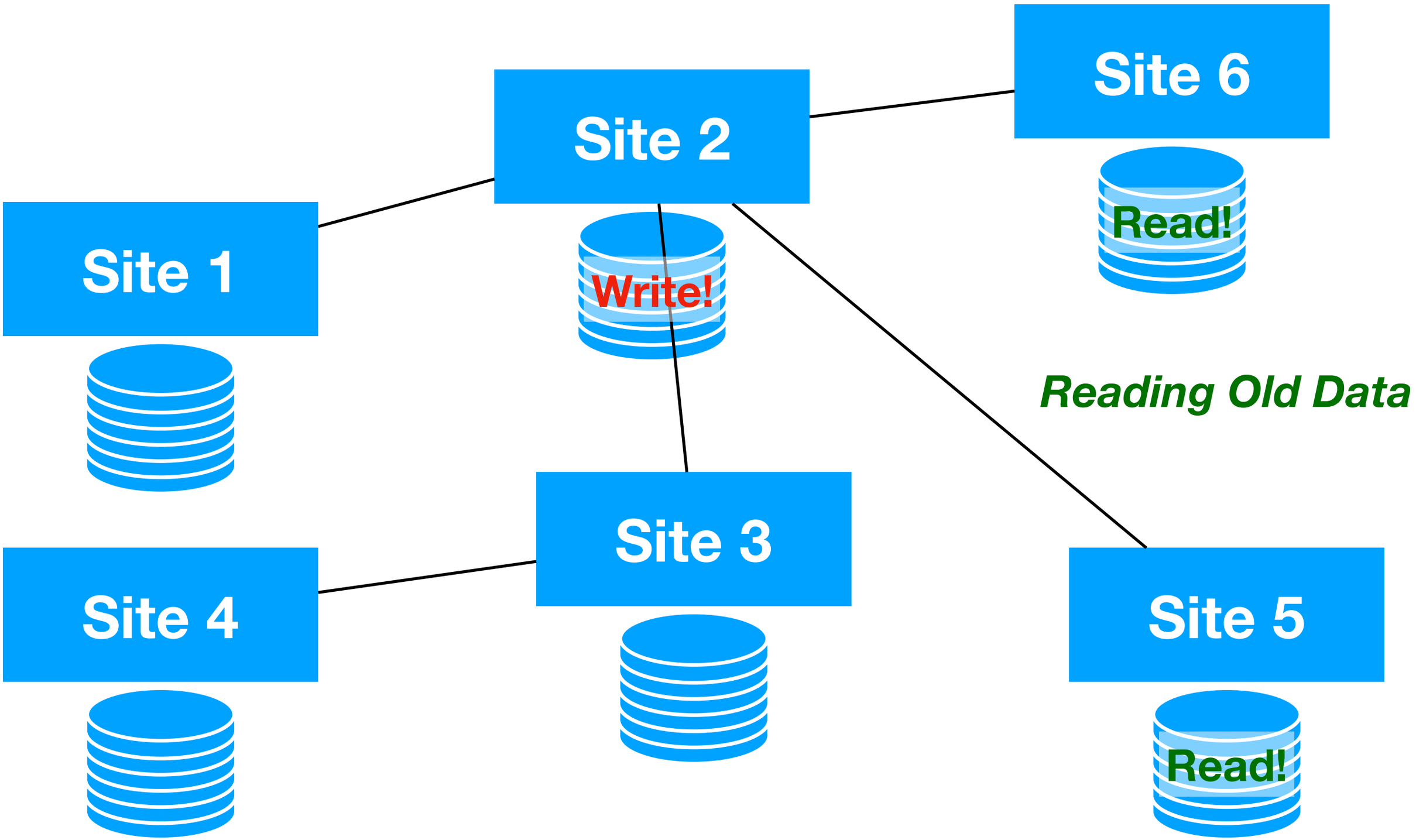
N=6; R=2; W=1

Example



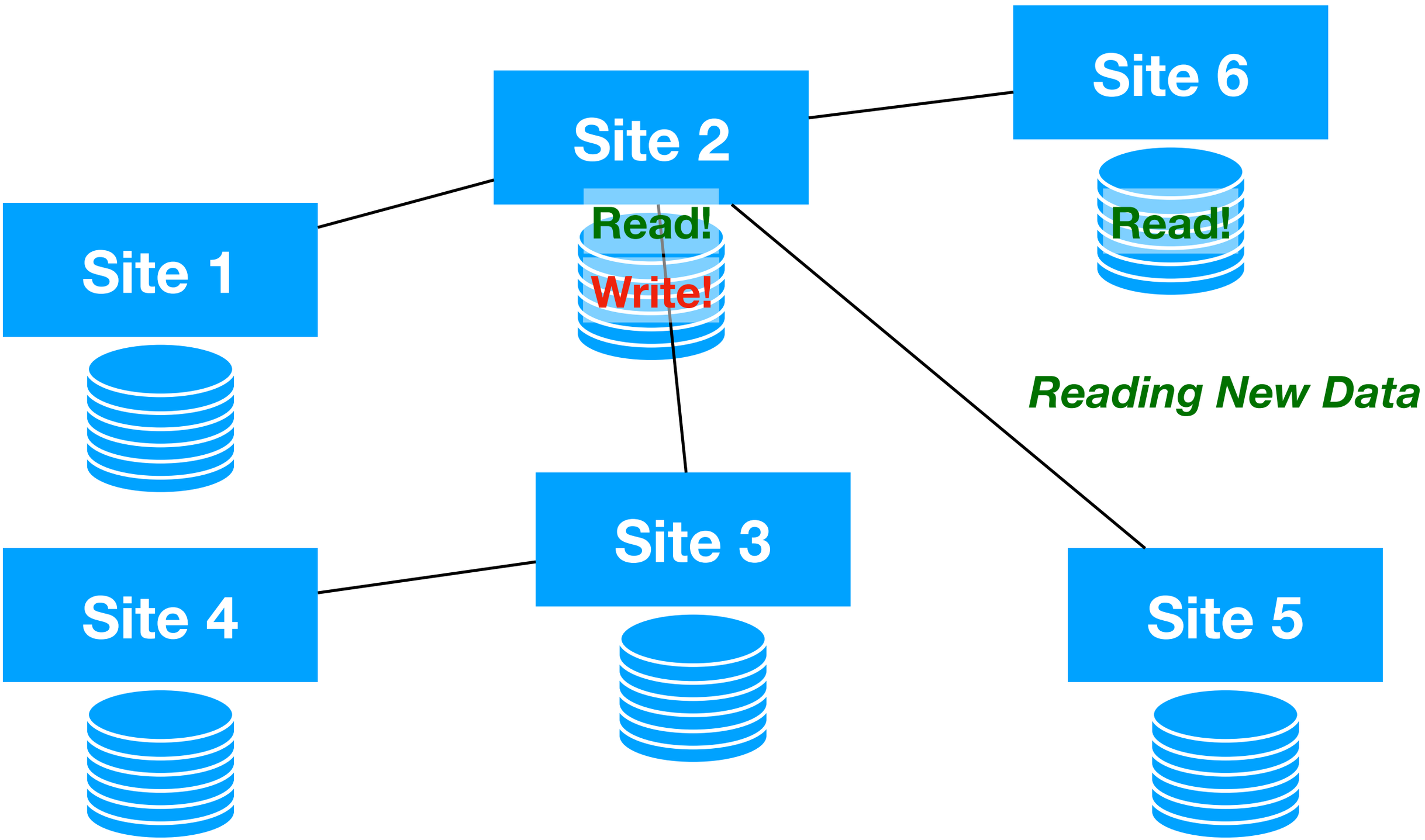
N=6; R=2; W=1

Example



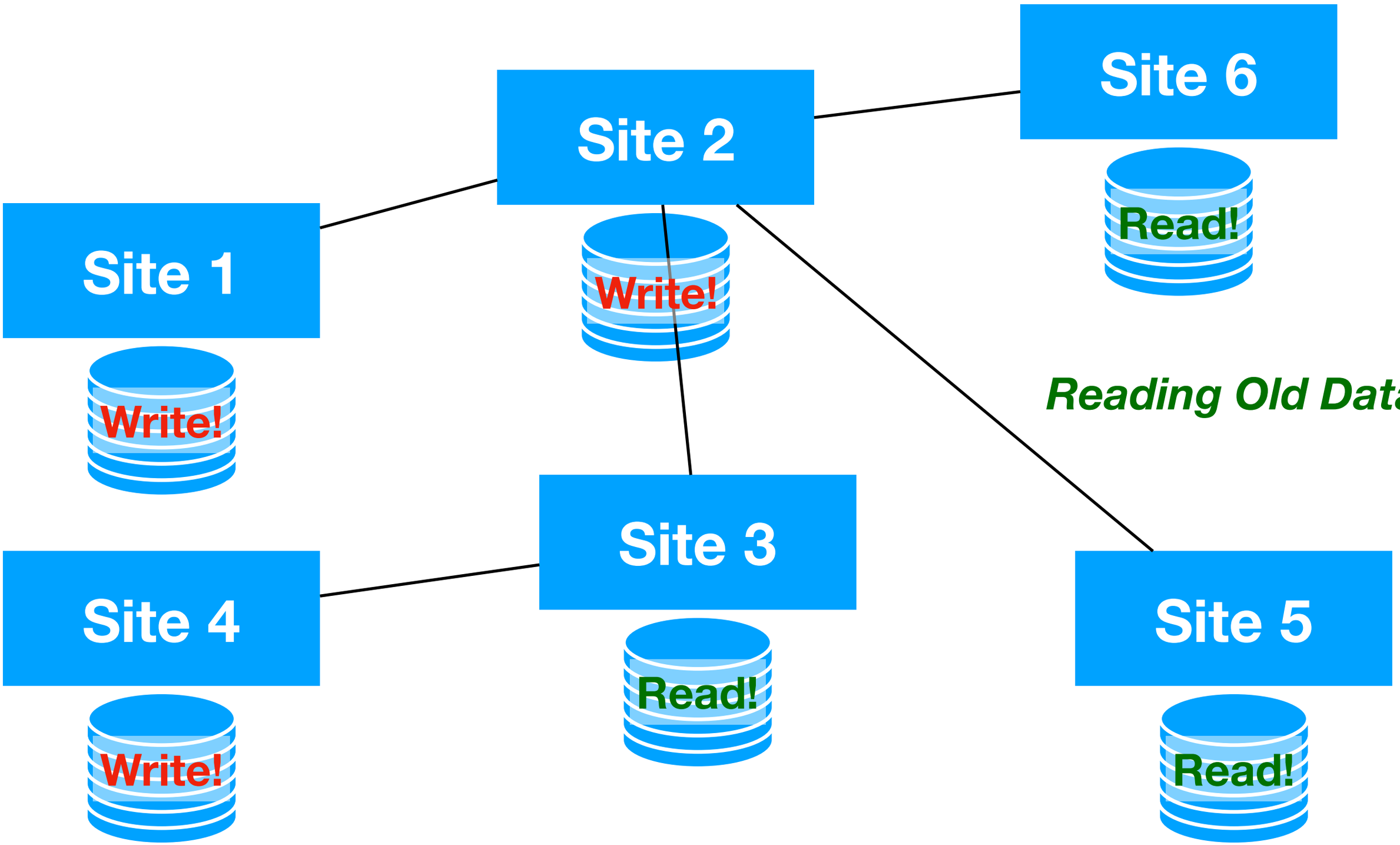
N=6; R=2; W=1

Example



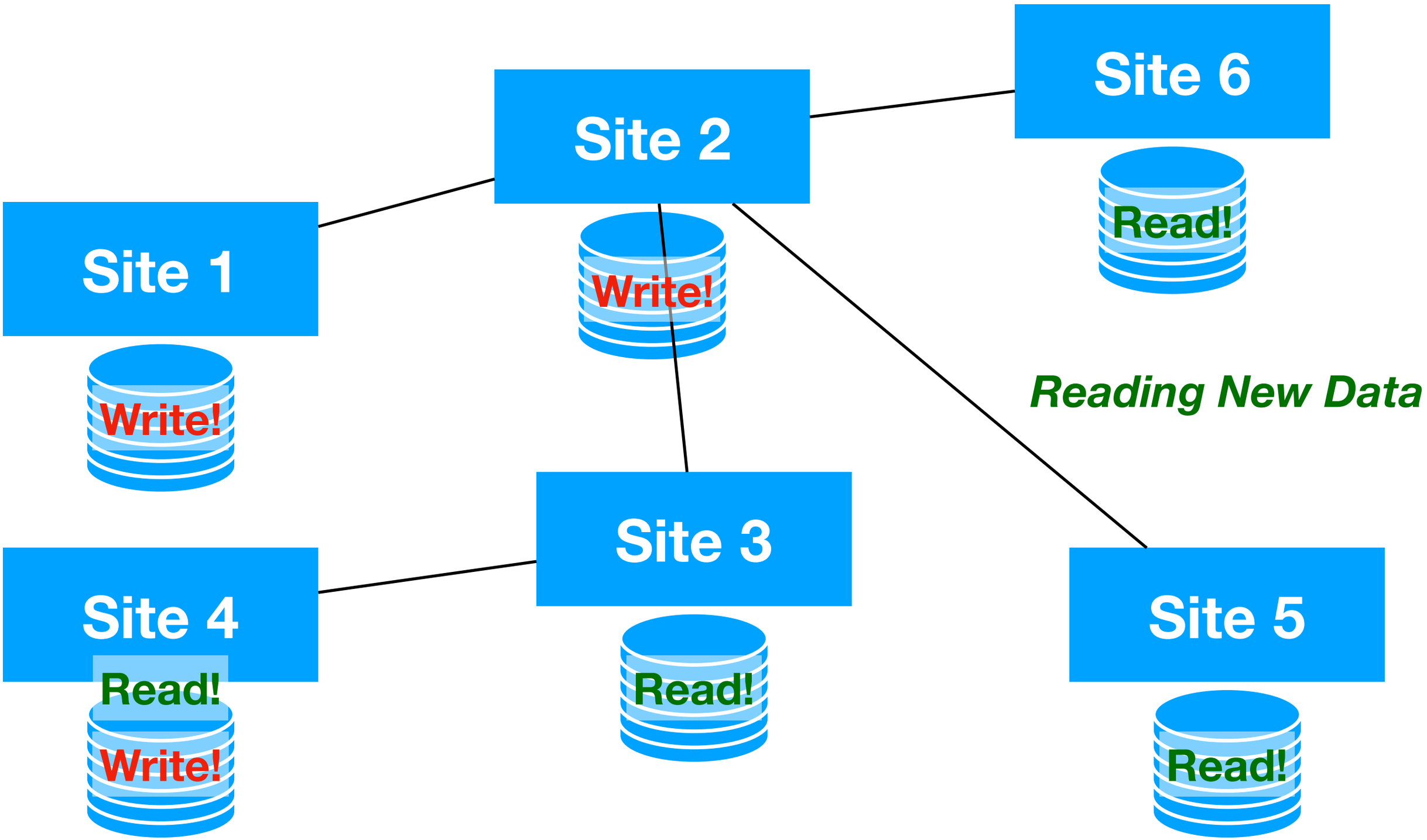
N=6; R=3; W=3

Example



N=6; R=4; W=3

Example



Condition for Consistency

- Strong consistency if **R + W > N**
- If so, we read at least one **updated** replica
- Use timestamps to identify **most recent** version

Amazon Dynamo

- **Eventually consistent** data storage system
- Specialized to the needs of **Amazon**
  - "**always writeable**" - otherwise, sales are lost
  - Customized **conflict resolution** strategies

Writing Data

- Updates should **always** be possible and fast
- Only require **updating few replicas** for update to return
- Means we may have **temporarily inconsistent** data

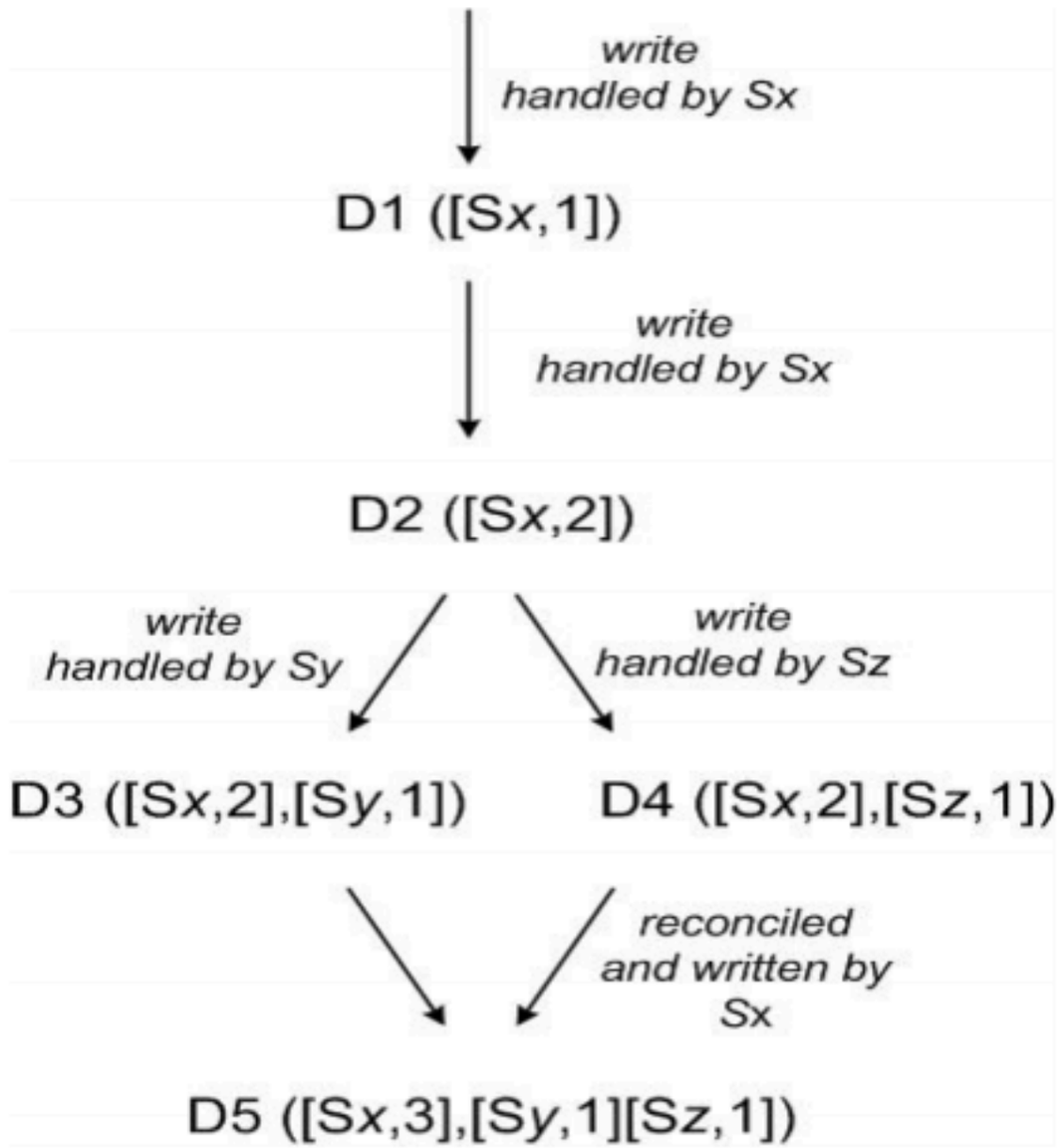
Reading Data

- We may have temporarily **inconsistent** data
- Reading multiple replicas may return **multiple versions**
- First try to identify and discard **outdated** versions
- For others: apply customized **reconciliation** strategy

Identifying Outdated Versions

- We associate each object version with a **vector clock**
- A vector clock stores **one counter for each node**
- Node counters **increase** monotonically for updates
- Each node **updates its counter** along with an object
- **Tracks** which version was generated from which version

Vector Clocks Example



Source: Dynamo: Amazon's Highly Available Key-value Store

Customized Reconciliation

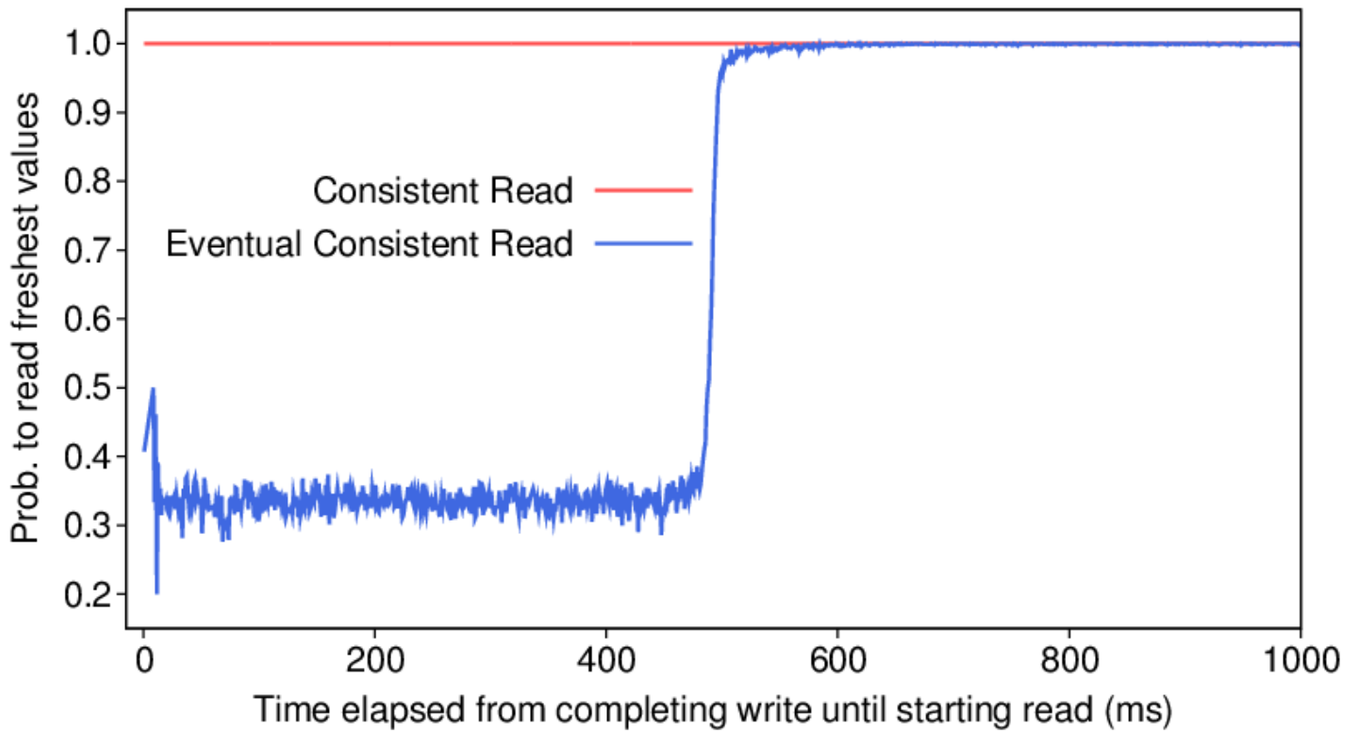
A typical example of a collapse operation is “merging” different versions of a customer’s shopping cart. Using this reconciliation mechanism, an “add to cart” operation is never lost. However, deleted items can resurface ...

Dynamo: Amazon’s Highly Available Key-value Store

Consistency Guarantees

- **Strong** consistency: reads after update return new version
- **Weak** consistency: reads may return outdated values
  - **Eventual** consistency: specializes weak consistency
- **Read-your-writes** consistency: process reads its updates
  - **Session** consistency: read-your-writes within session
- **Monotonic read** consistency:
  process never sees older value than seen before

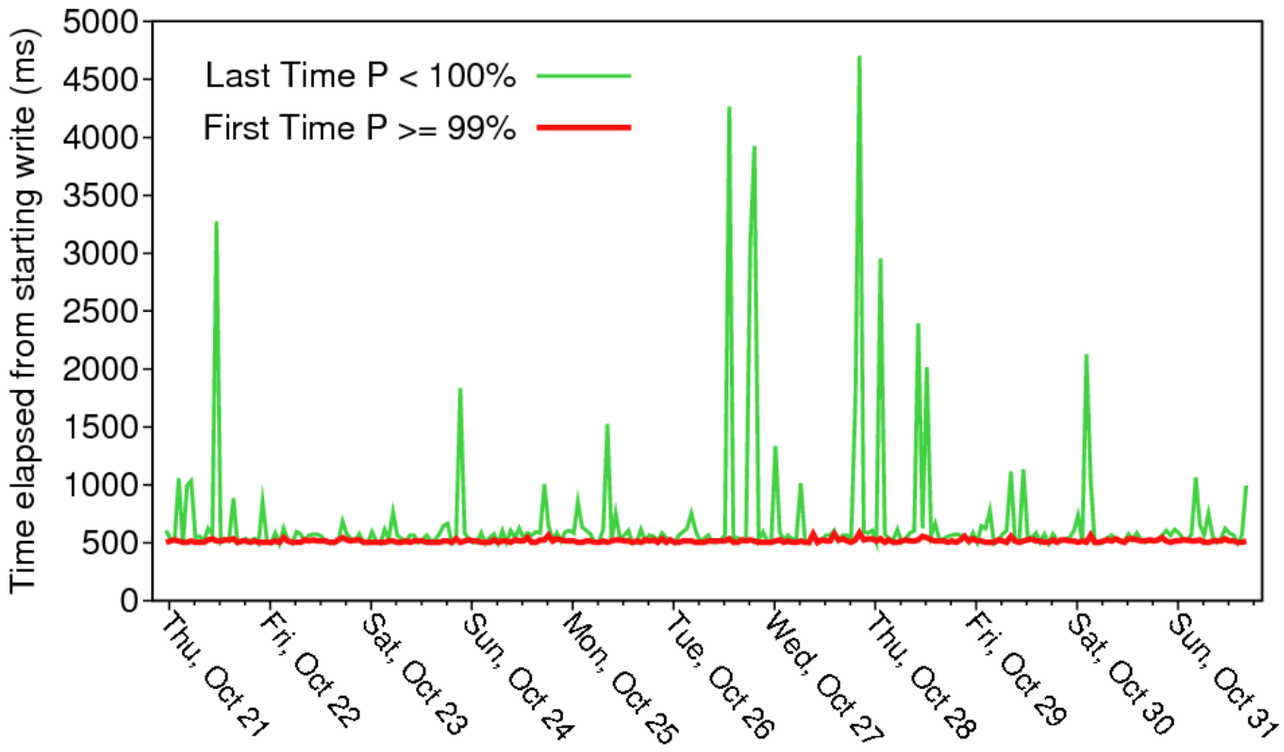
Analyzing Amazon SimpleDB I



Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: the Consumers' Perspective, H. Wada et al, CIDR 2011

Which conclusions can we draw from this?

Analyzing Amazon SimpleDB II



Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: the Consumers' Perspective, H. Wada et al, CIDR 2011

Analyzing Amazon SimpleDB III

	Second Stale	Second Fresh
First Stale	39.94%	21.08%
First Fresh	23.36%	15.63%

Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: the Consumers' Perspective, H. Wada et al, CIDR 2011

Analyzing Amazon SimpleDB III

	Second Stale	Second Fresh
First Stale	39.94%	21.08%
First Fresh	23.36%	15.63%

Non-Monotone Reads!

Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: the Consumers' Perspective, H. Wada et al, CIDR 2011