Database Backends

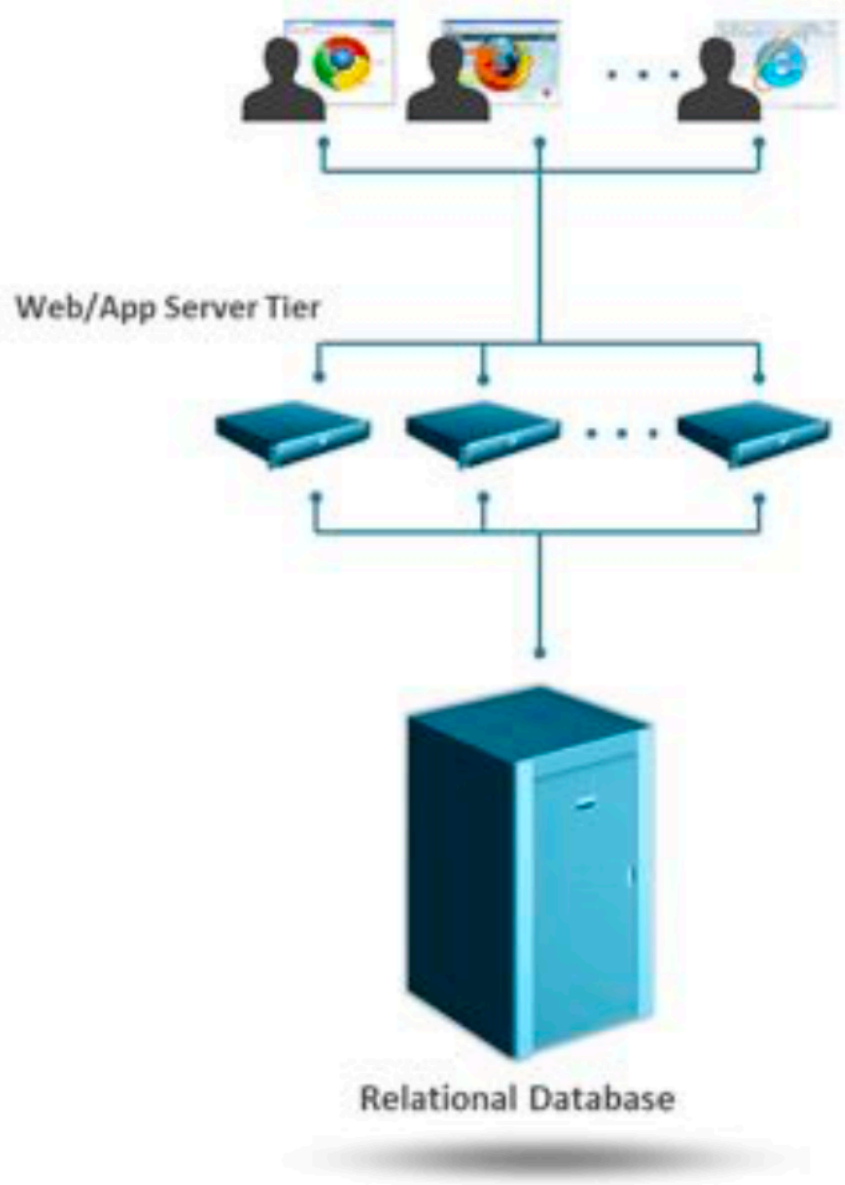
Scalability of Web Systems – Fall 2017

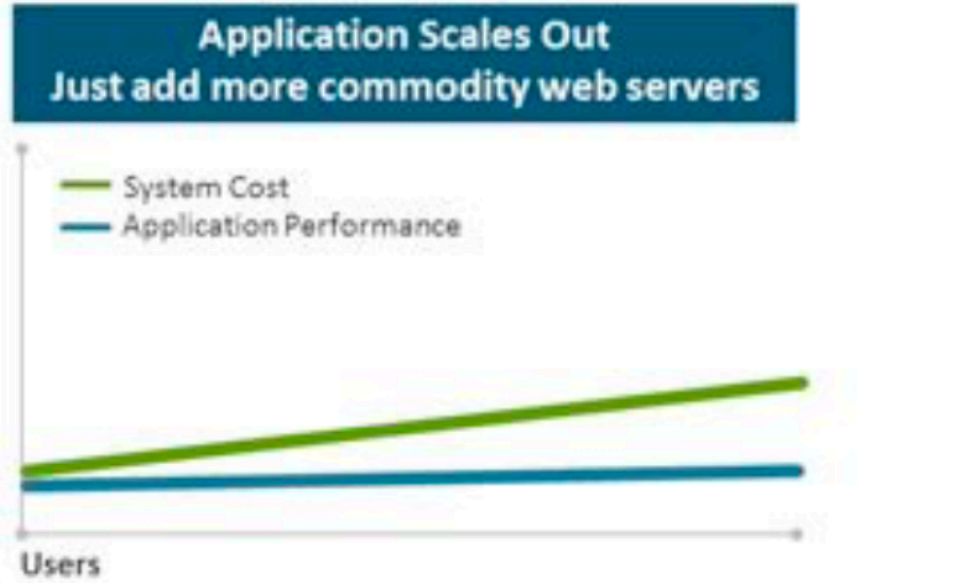
Björn Þór Jónsson

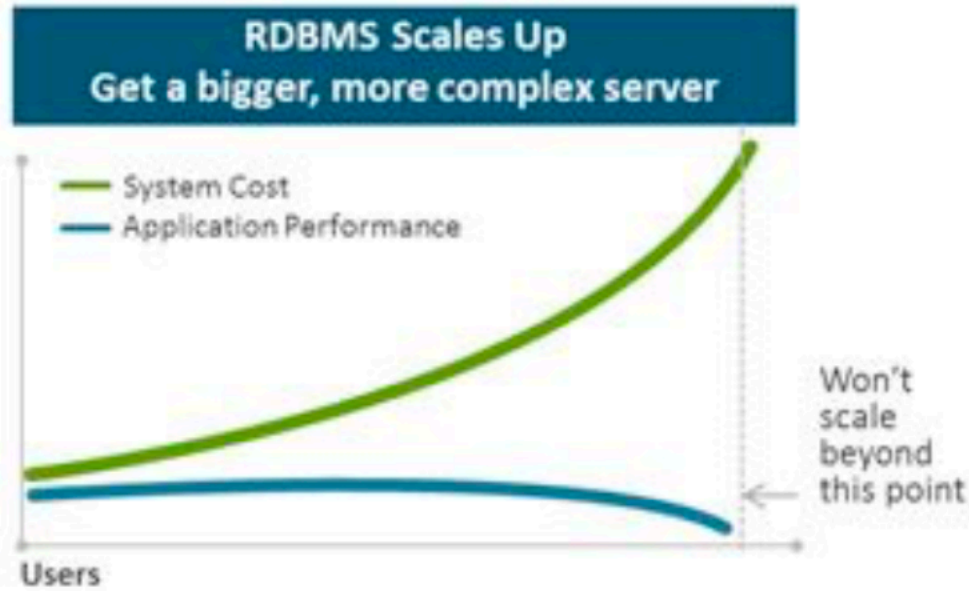
Slides partially adapted from Philippe Bonnet, Grímur Tómasson

Point of Departure: RDBMS

- Services?
  - Data model: Relations
  - Queries: SQL
  - Interaction: Transactions
- Workloads?
  - OLTP
  - OLAP


Web/App Server Tier
Relational Database


Application Scales Out
Just add more commodity web servers
System Cost
Application Performance
Users


RDBMS Scales Up
Get a bigger, more complex server
System Cost
Application Performance
Won't scale beyond this point
Users

Scaling RDBMSs: Distributed Processing

- Distribute data to servers
  - “sharding”
  - Isolate transactions to a single server (when possible)
- Replicate data
  - Log transfers & replay
- Not really what they were designed for…

Architecting RDBMSs for Workload Type

OLTP

Queries = small CRUD operations

- Row-storage
- External indices necessary

OLAP

Queries = large analysis scans

- Column storage
- External indices useless
- Compress each column
  - Disk space savings
  - IO savings

NewSQL – Optimize for OLTP only

Assumptions = "banking"

- Relations = MUST
- Transactions = MUST
- SQL = MUST
- Workload = **OLTP**

Leverage shared-nothing distributed architectures?

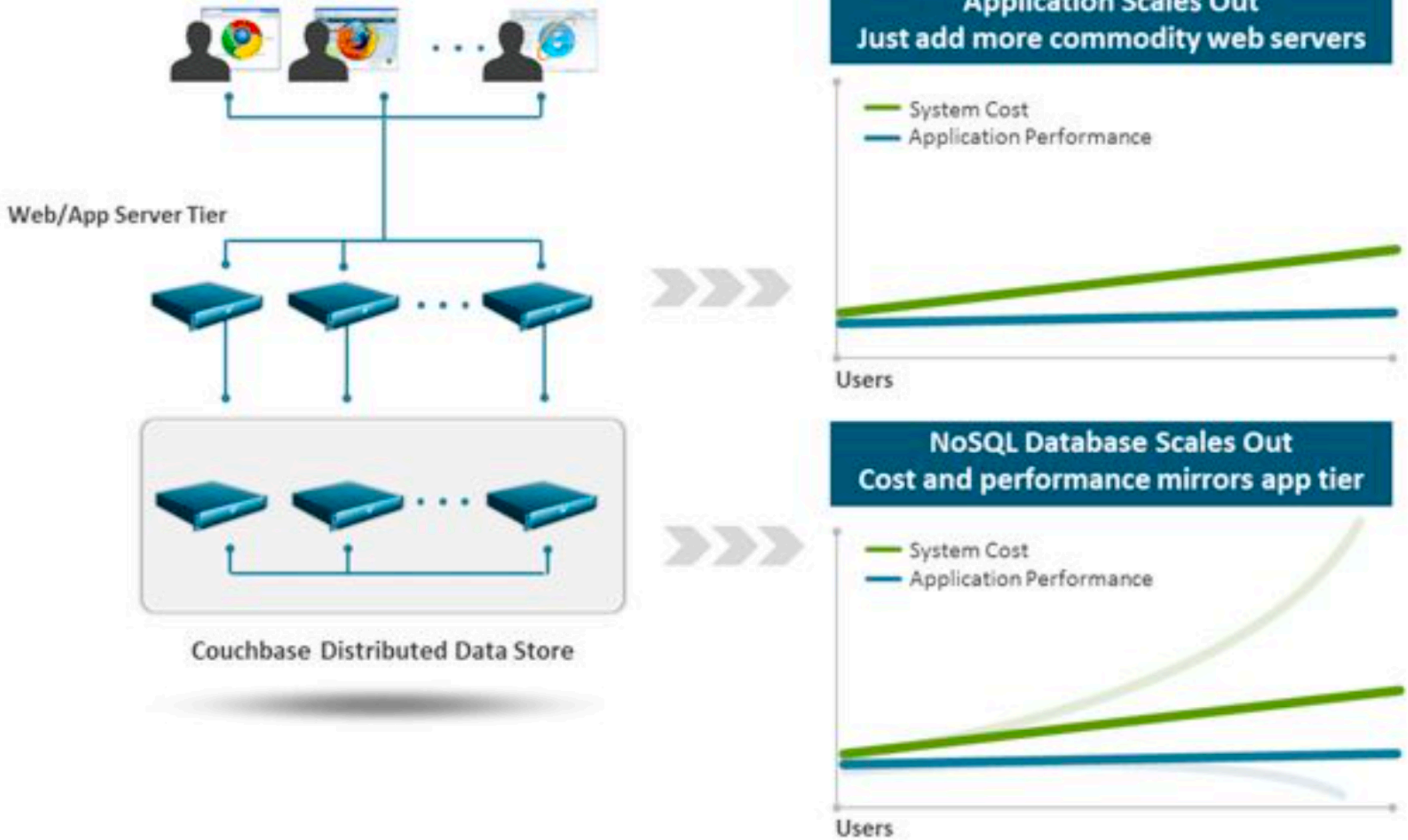
https://en.wikipedia.org/wiki/NewSQL

Example: H-STORE

- Disk oriented → Main memory
- Multithreading → Single thread
  - Multithreading was needed to hide latency
- Locking → Single Thread
  - Concurrency control no longer needed
- Log-based recovery → Peer recovery
  - Redo logging no longer needed

Each transaction takes microseconds!

https://en.wikipedia.org/wiki/H-Store


Web/App Server Tier
Couchbase Distributed Data Store
Application Scales Out
Just add more commodity web servers
System Cost
Application Performance
Users
NoSQL Database Scales Out
Cost and performance mirrors app tier
System Cost
Application Performance
Users

Dimensions of Datastore Space I

Data Model

- Relational
- Graph
- Stream
- RDF
- Key-value
- Semi-structured (JSON, XML)
- Document
- Triples
- Multidimensional data (arrays)

Programming Abstractions

- Declarative programming
- Domain-specific language
- Map-reduce
- Transaction abstractions

NoSQL

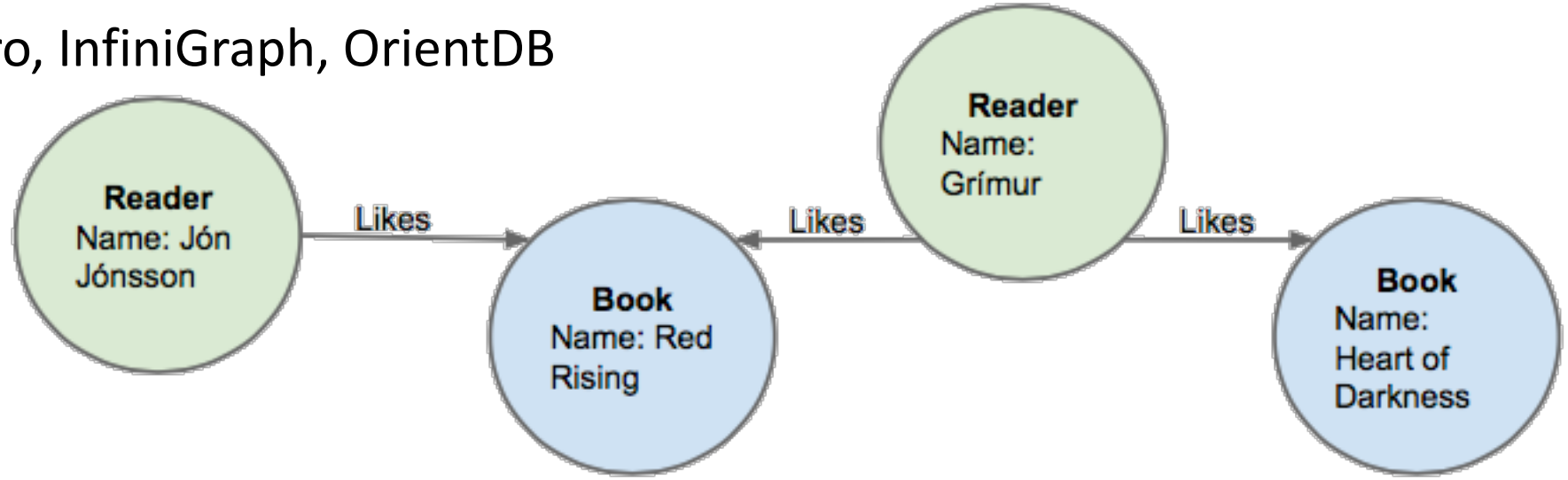
- Data model
  - Not relational
  - No formally described structure (schema-less)
- Interface
  - Not only SQL
  - Proprietary, REST, CQL etc.
- Architecture
  - Usually distributed
- Mostly not ACID compliant
- No Join operation
  - Requires a fixed schema
  - Design your data access up front
- Web-driven
- Mostly open source
- Consistency/Availability tradeoff
  - Later!

Partial NoSQL timeline

- Pre 1970s - noSQL-like databases (network model, hierarchical model)
- 1964 - MultiValue data model developed at TRW (PICK)
- 1979 - DBM released (AT&T, Ken Thompson)
- 1989 - Lotus Notes released (IBM)
- 1998 - noSQL name (Carlo Strozzi)
- 2000 - neo4j released
- 2004 - Google BigTable development starts
- 2005 - CouchDB released
- 2007 - Amazon's Dynamo paper released
- 2008 - Facebook's Cassandra open sourced
- 2012 - Amazon's DynamoDB released

Graph Stores

- Nodes = Entities
- Edges = Relationships, directional
- Properties = Entity descriptors
- Examples
  - neo4j, Allegro, InfiniGraph, OrientDB



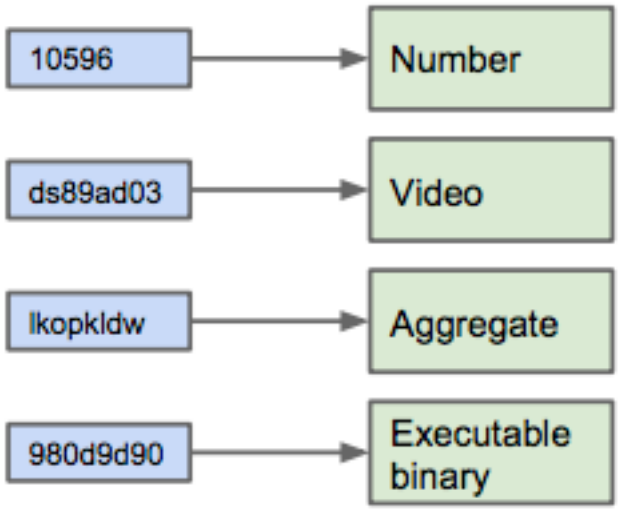
Data Streaming Systems

Streaming vs. Queries

- Database =
  - Data at rest
  - Queries on the move
- Data stream =
  - Queries at rest
  - Data on the move

Key-Value Stores

- Associative Array
  - Unique key points to a value
  - Value contents unknown
- Can not be queried
  - Can be aggregate
- Examples
  - Riak, Voldemort, MemcacheDB

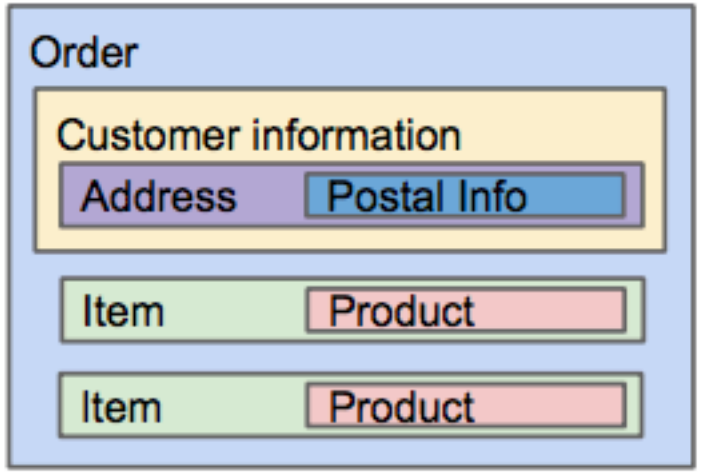


Document Stores

- Each value is a document
  - Most often JSON
  - Unique keys used for retrieval
- You can query into the document
  - More transparent than key-value stores
- The document is an aggregate structure
- Examples
  - Lotus Notes, CouchDB, MongoDB

Document Store Example

```
{
"order_id" : 3294,
"customer" : {
        "ssn" : "1111111119",
        "name" : "Jón Jónsson",
        "address" : {
                "street" : "Skúlagata 18",
                "postal_code" : "101"
        }
},
"line-items" : [
        {"product" : "Sófasett", "price" : 300000, "discount" : 45000},
        {"product" : "Lampi", "price" : 40000 }
]
}
```



Relational tables needed

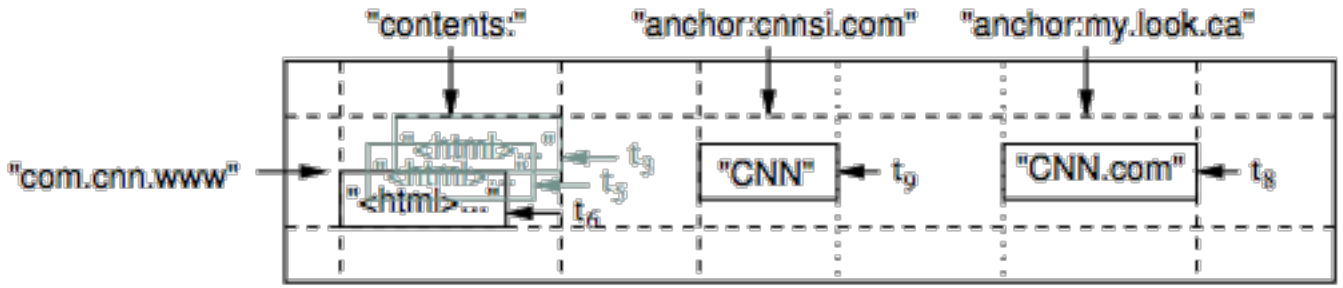
- Customer
- Address
- Postal_info
- Order
- Order_line
- Product

Wide Column Stores

- Similar to Key-Value stores BUT the key is multi-dimensional
- “It uses tables, rows, and columns, but unlike a relational database, the names and format of the columns can vary from row to row in the same table”

https://en.wikipedia.org/wiki/Wide_column_store

- Examples
  - BigTable, Cassandra, …



Data Model Summary

When to Consider NoSQL?

- When data model fits problem
- With huge amounts of data
  - Distributed NoSQL system!
- One paradigm end-to-end
  - No impedance mismatch
  - Shorter time to market
- Consistency/Availability tradeoff works for the business domain
  - Later!

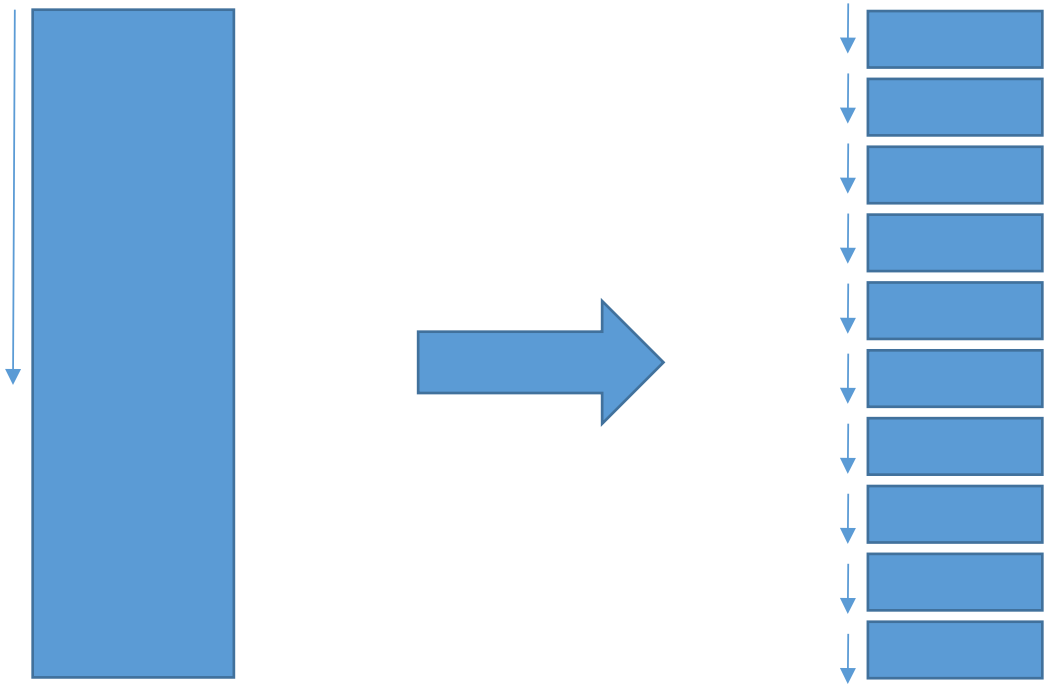
In Conclusion

- Must choose the right tool for the problem
  - There is no one-size-fits-all solution
- NoSQL is not a RDBMS replacement
  - In some cases, it is a better alternative
- **Scaling always requires distributed processing!**

Dimensions of Datastore Space II

Partition Management

- Partitioning (sharding)
  - Defining data partitions + data distribution
- Resource management
  - OS support on each node + cluster file system
- Job scheduling
  - Scheduling and distribution of programs to be executed on partitions
- Failure handling / Recovery
  - Handling node/network failure



Resource Management

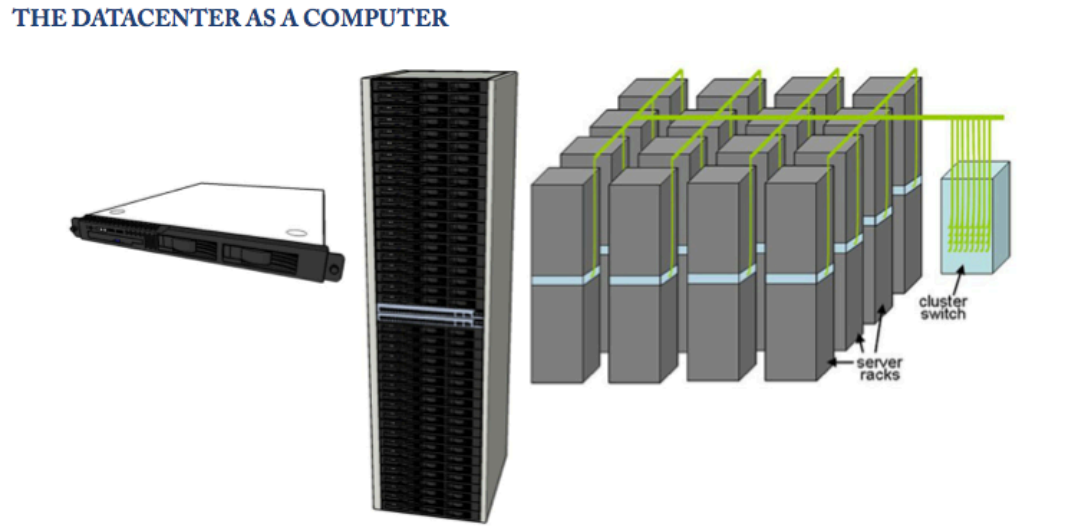


FIGURE 1.1: Typical elements in warehouse-scale systems: 1U server (left), 7′ rack with Ethernet switch (middle), and diagram of a small cluster with a cluster-level Ethernet switch/router (right).

- Processing abstraction
  - Per node processing
    - Single vs. multi-threaded
    - With our without HW acceleration (GPU)
- Communication abstraction
  - Within a rack
  - Across racks
  - Across data centers
- Storage abstraction
  - Single address space
    - Sharding is hidden from programmer

  Vs.
  - Multiple address space
    - Sharding is explicitly performed by the programmer
- Cluster Management
  - Resource allocation (RAM/Disk/CPU/Network)
  - Admission control
  - Elasticity

Key Consideration: Organization

- Master-Slave
  - Simple
  - Known data locations
  - Single point of failure
- Peer-to-Peer
  - Complex
  - Unknown data locations
  - Robust

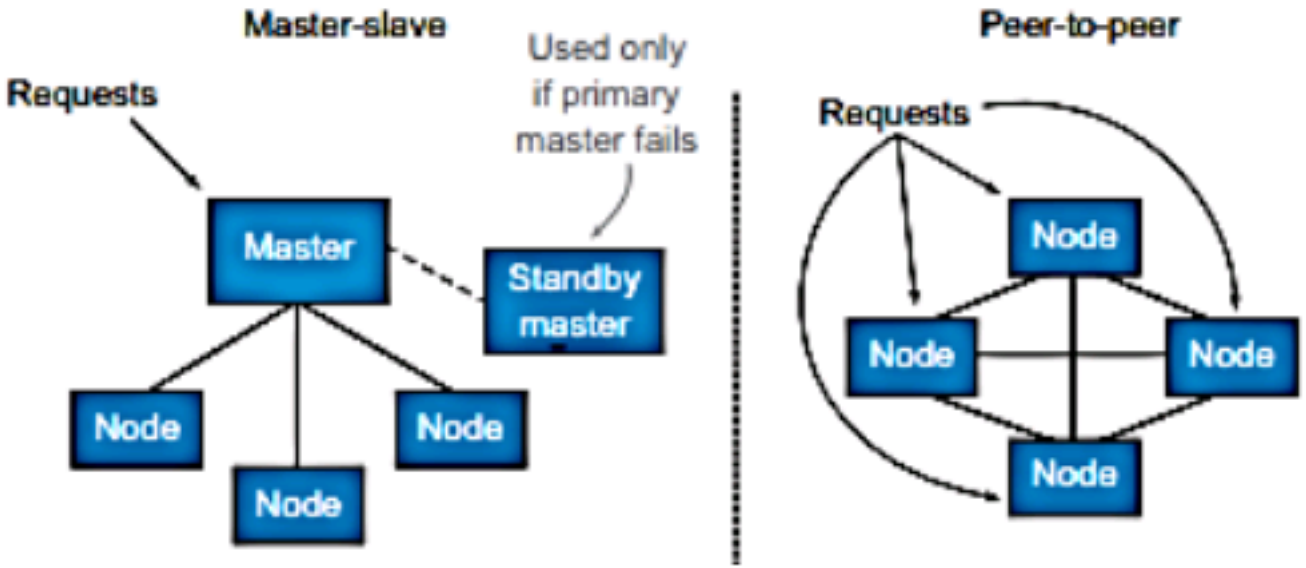


Figure: Master-slave versus peer-to-peer

http://www.ques10.com/p/2826/explain-master-slave-vs-peer-to-peer-distributio-1/

Partitioning/Sharding

Three ways to partition data:

1. Round Robin
2. Hash partitioning
3. Range partitioning

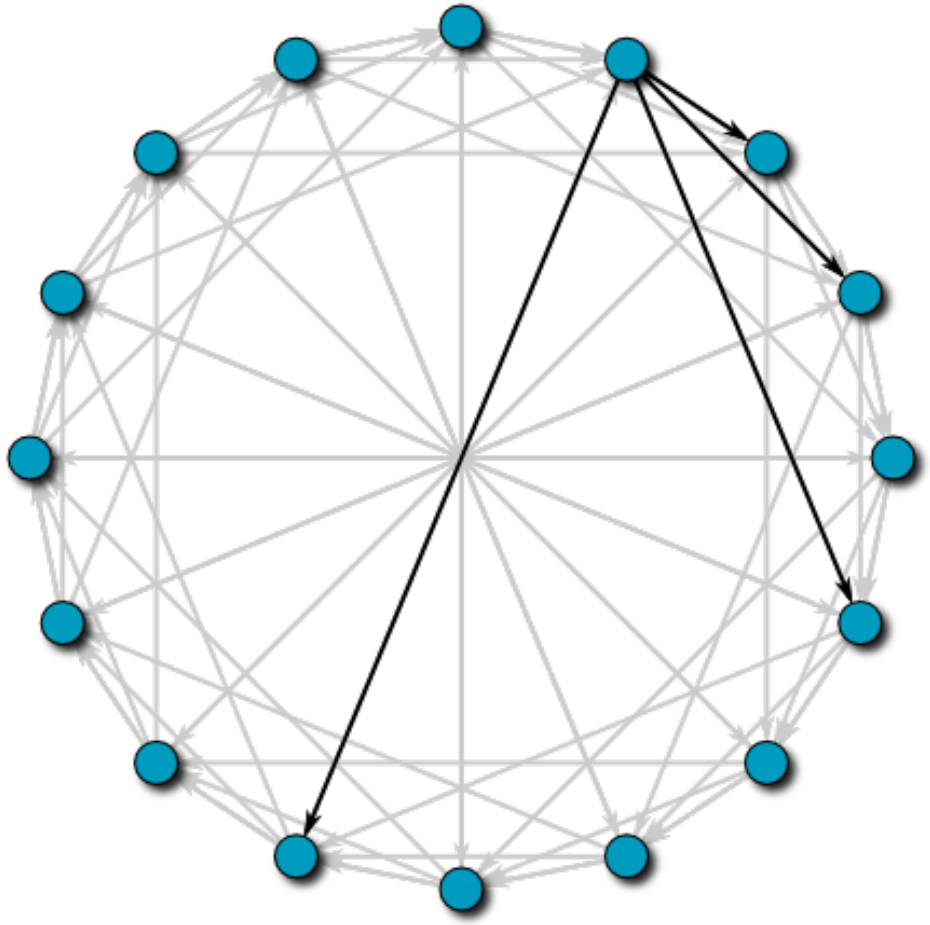
	Fast	Skew	Scan	Point	Range
Round Robin	+	+	+	-	-
Hash	+	-	+	+	-
Range	~	-	+	~	+

Objectives:

- Fast partitioning
- Minimize skew (data + access)
- Efficient support for scan/point/range queries

Distributed Hash Tables

- Consistent hashing
  - Supports a Peer-to-Peer model
  - Assigns equal “space” to nodes
  - Each node responsible for a range
- Example: **CHORD**
  - Servers can easily join/leave
  - Finding data is O(log *n*)
  - Can cope with heterogeneous servers



Failure Handling

- Google Data:
  - 2009: DRAM
    - 25K to 70k errors per billion device hours per Mbit
    - 8% of DIMMS affected by errors
  - 2007: Disk drives
    - Annualized failure rate: 1.7 % for 1 year old device to 8.6% for 3 years old devices
  - Failures are the rule, not the exception!
- Types of failures:
  - HW, SW, maintenance, poor job scheduling, overload
  - Permanent / transient

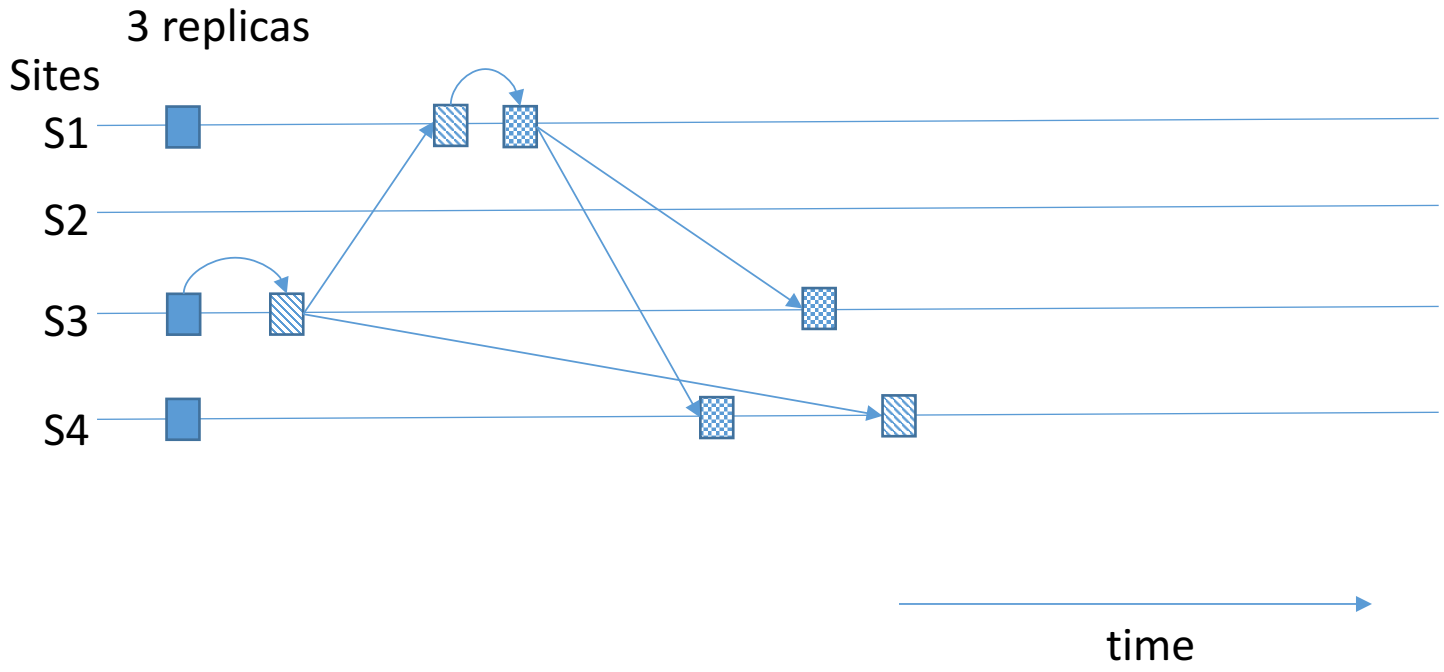
Failure Handling – Redundancy

- HW redundancy in the data center
  - Might reduce, but does not eliminate failure
- Replication
  - Partition replicated within/across racks/data centers
    - Impacts both availability and performance
    - Job can be performed on several replicas in parallel, completes when fastest is done
    - Introduces a problem in terms of **consistency**

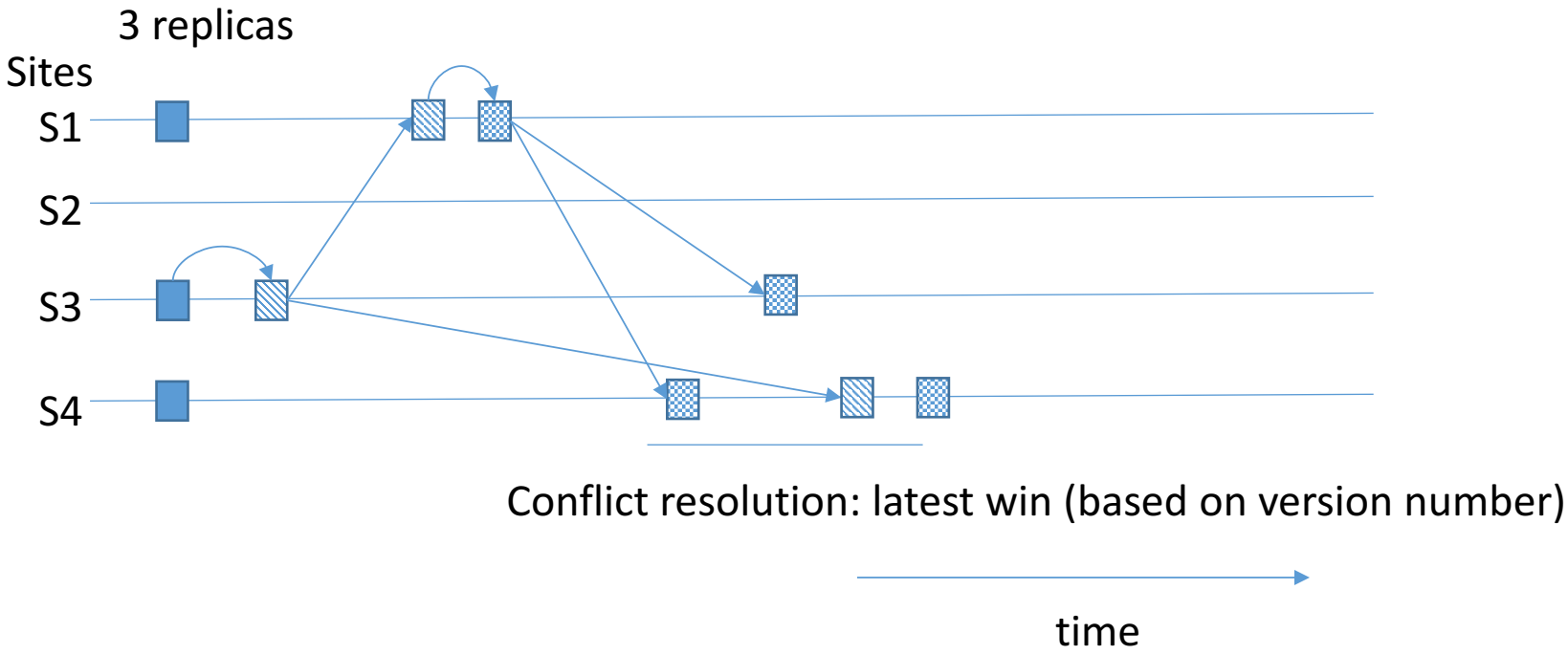
Replica Consistency

- **Sequential (or strong) consistency**: All updates are seen by all processes in the same order. As a result, the effects of an update are seen by all observers. There is no inconsistency.
  - **Atomic (or external or strict) consistency:** The effects of updates are "immediately" seen by all processes in the same sequence. There is no inconsistency, but reads of a replica might be delayed.
- **Weak consistency**: Observers might see inconsistencies among replicas
  - **Eventual consistency**: A form of weak consistency, where at some point, in case there is no failure, all replicas will reflect the last update.

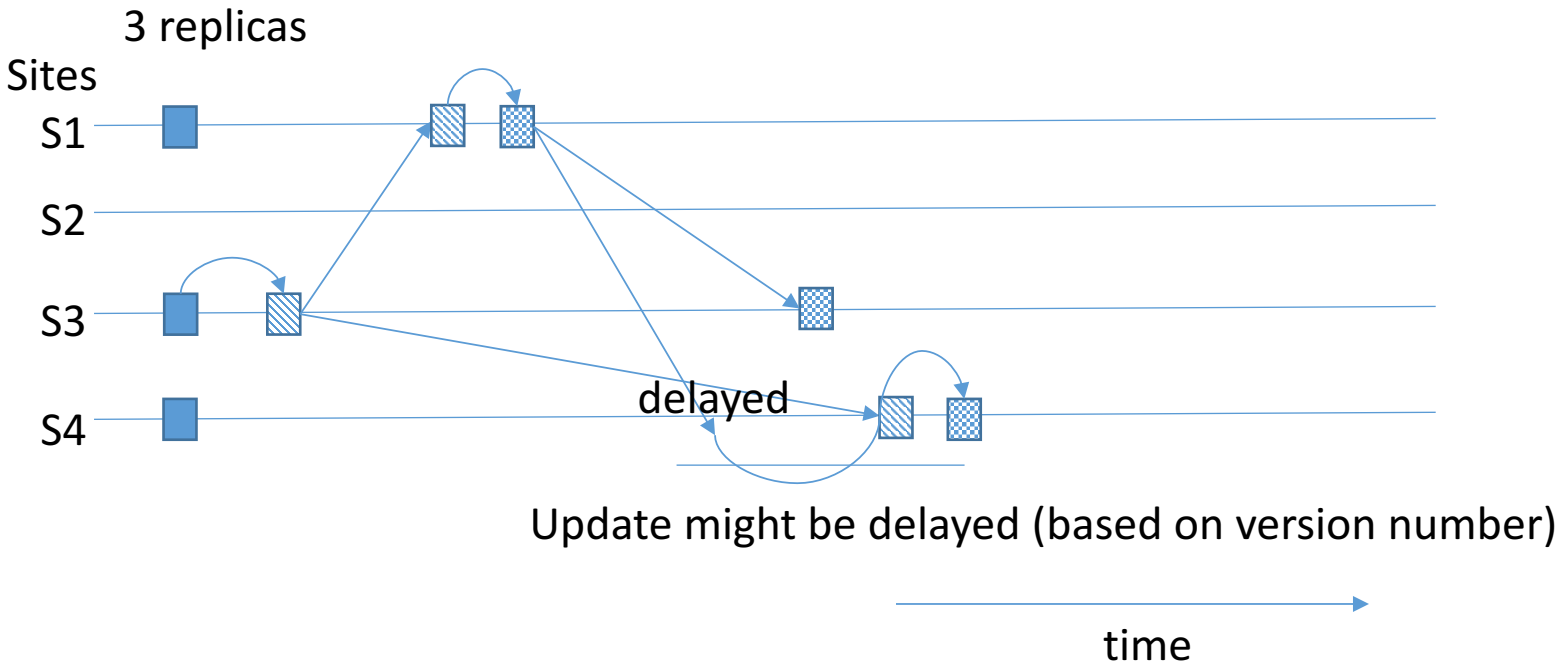
Replica Consistency – Weak Consistency



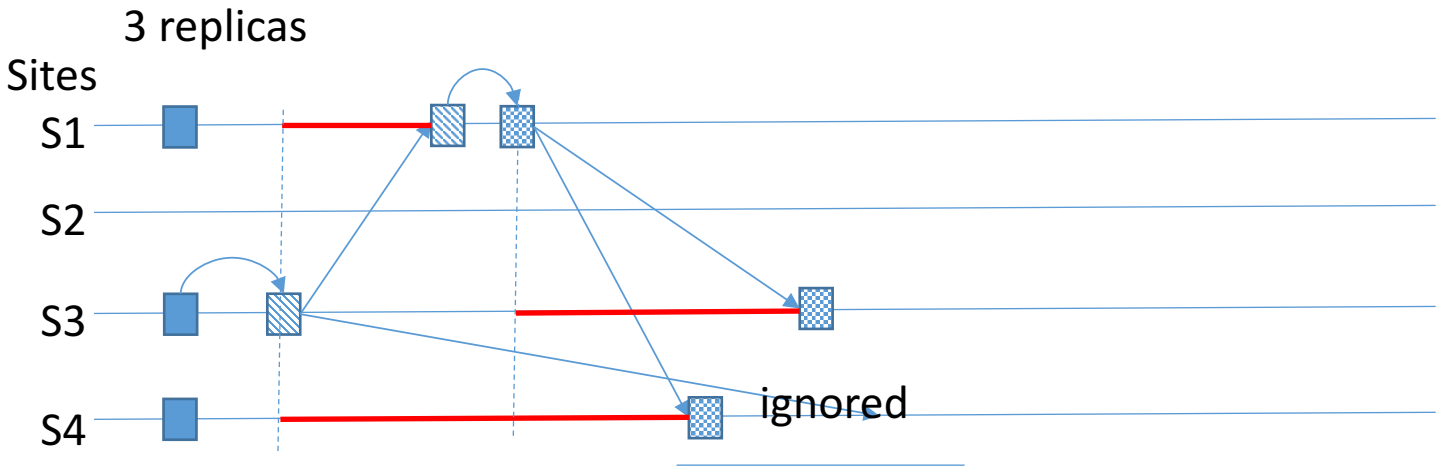
Replica Consistency – Eventual Consistency



Replica Consistency – Strong Consistency



Replica Consistency – Strict Consistency



Reads not allowed on outdated version of a replica
Only latest update accepted

time

How to ensure synchronized decision on availability of a new value?

Paxos Algorithm

- Distributed consensus algorithm
  - Peer-to-peer architecture
  - Messages can be lost
  - Machines can fail

- A Paxos node
  - Proposer: proposes a value it wants agreement upon (e.g., new value)
  - Acceptor: chooses a value – it might receive several proposals and sends its decision to learners
  - Learner: determine whether a value has been accepted.

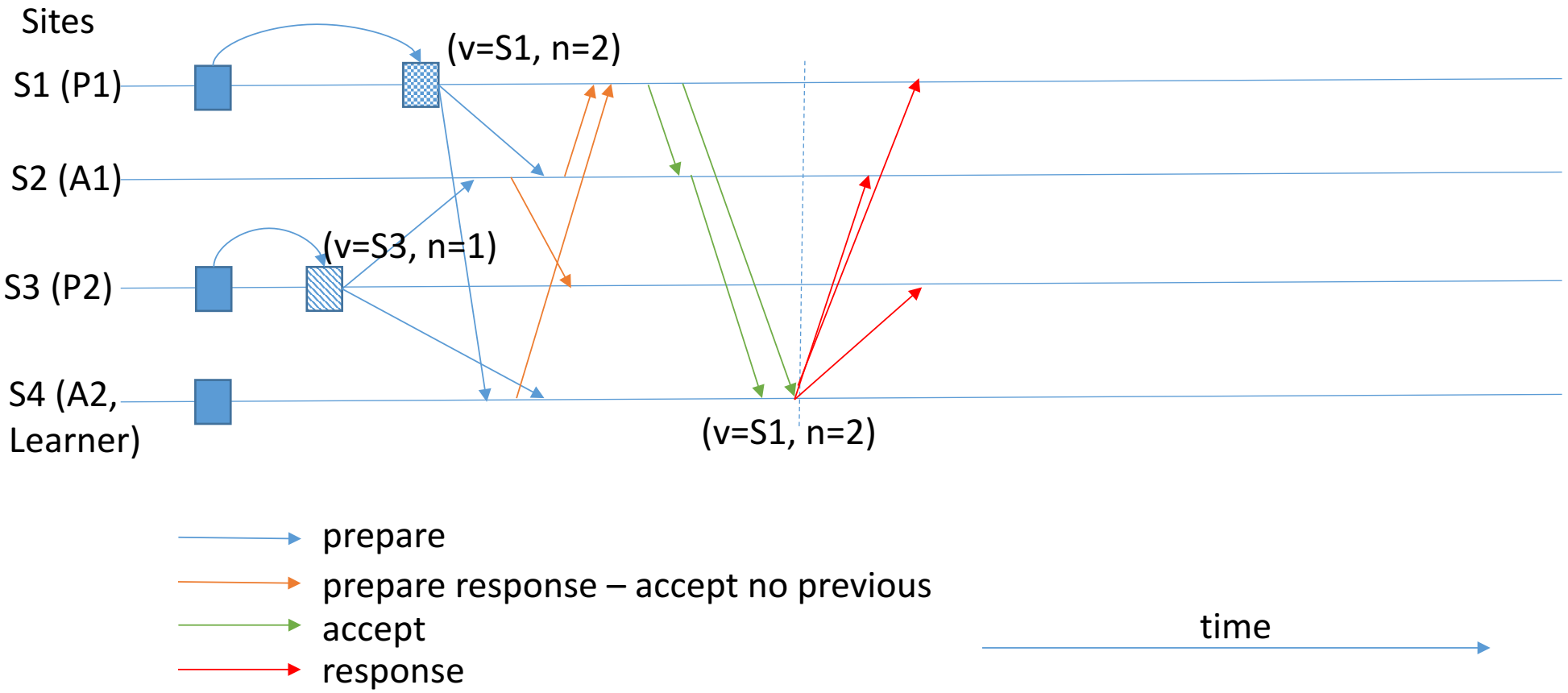
- In Paxos, a value is accepted if:
  a **majority** of acceptors **choose** the same value

https://angus.nyc/2012/paxos-by-example/

Paxos Algorithm

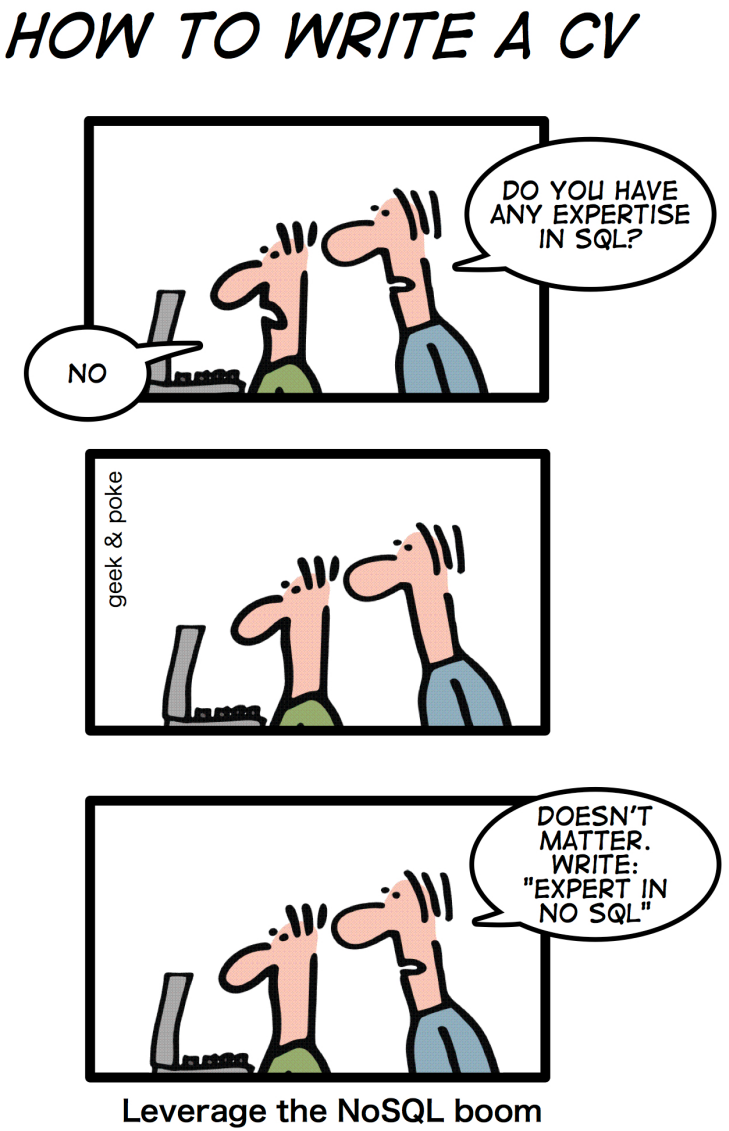
- Proposers send
  - Prepare request
    - Proposed value (p) and proposed number (n)
  - Accept request
    - Proposed value (p) and proposed number (n)
  - Proposed numbers must be unique, monotonically increasing across all proposers!

Paxos – Simple Example



Take Away

- Data models abound
  - Use one that fits your application
- Partitioned processing is key to handling large volumes of data
  - Sharding
  - Resource management
  - Job scheduling
  - Failure handling
- Consistency/Availability/Latency tradeoff
  - Later!



http://geekandpoke.typepad.com/geekandpoke/2011/01/nosql.html