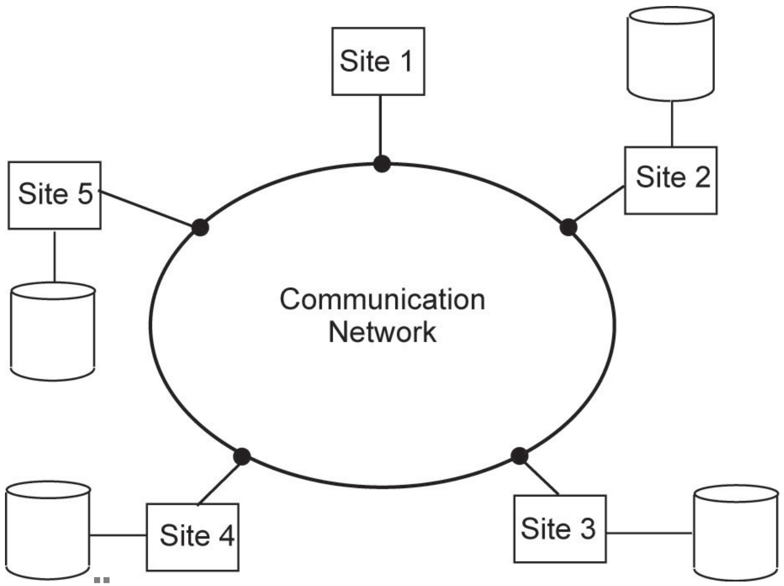
CS4224/CS5424 Lecture 1
Introduction

Distributed Database Systems

- A distributed database is a collection of multiple, logically interrelated databases distributed over a computer network

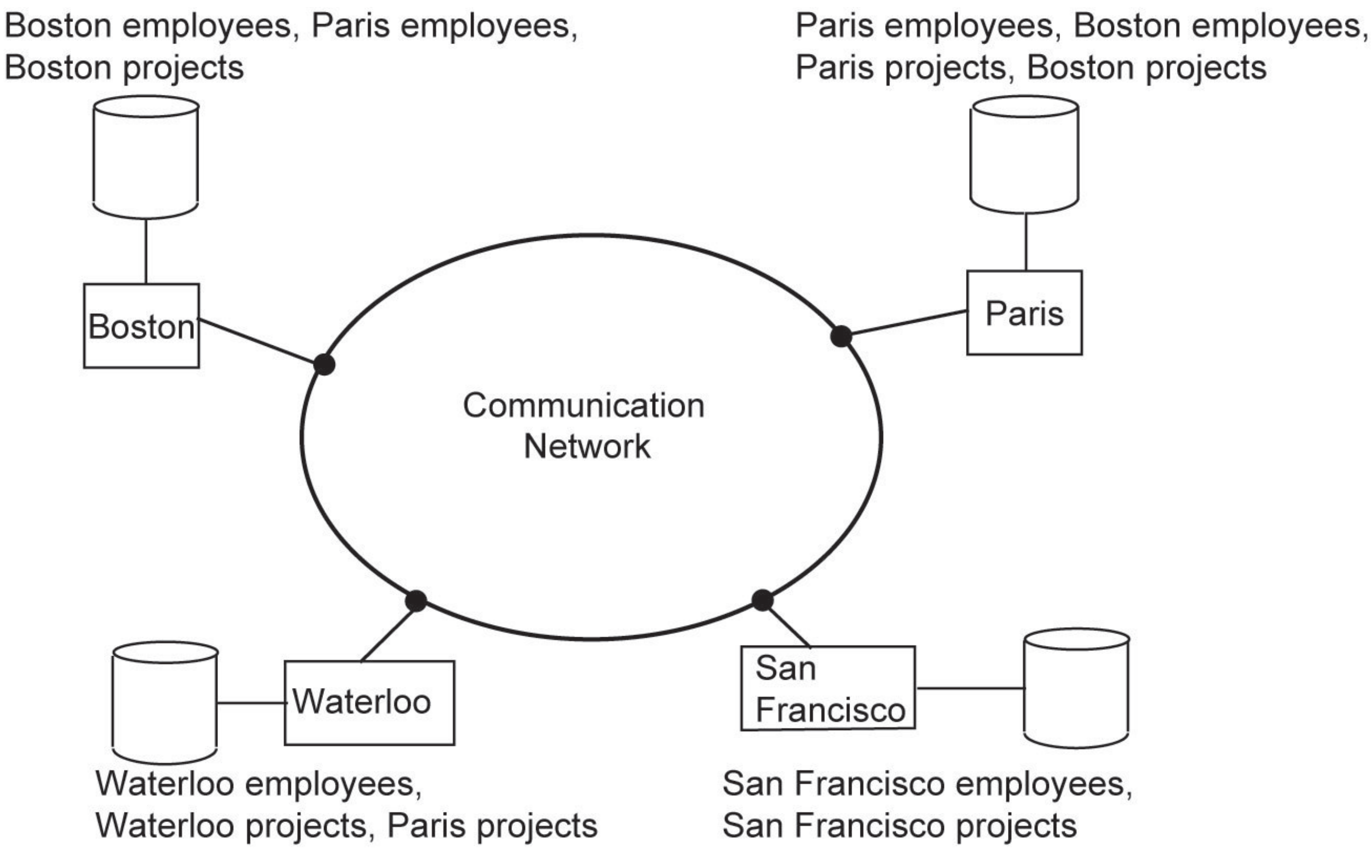
- A distributed database management system (DDBMS) is the software system that manages the distributed database and makes the distribution transparent to the users



(Özsu & Valdureiz, 2011)

Early Distributed DBMS

- Supports the organizational structure of distributed enterprises



(Özsu & Valdureiz, 2011)

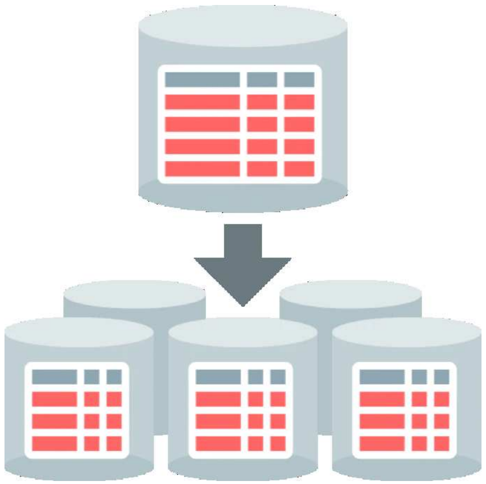
Modern Distributed DBMS

- NoSQL & NewSQL

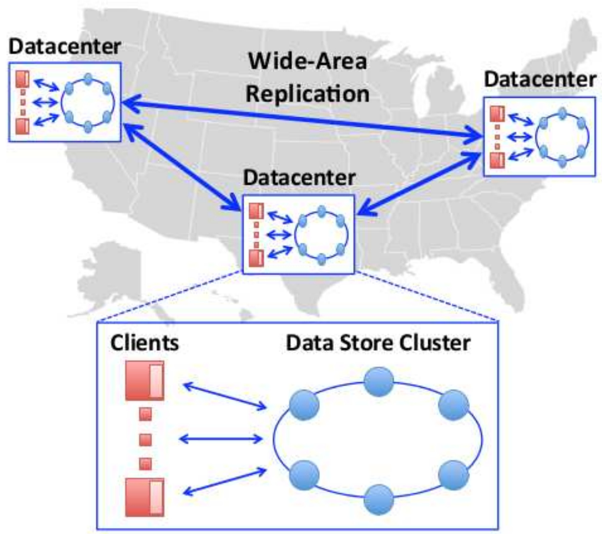
http://www.informationweek.com/big-data/big-data-analytics/16-nosql-newsql-databases-to-watch/d/d-id/1269559

Modern Distributed DBMS

- Supports **large-scale data management** challenges of today's web-based applications
  - Database Scalability, High Availability, Low Latency
  - Schema-less data or data with dynamic schema
- Data being sharded & replicated across a cluster of servers



Data Sharding (Image: Oracle)



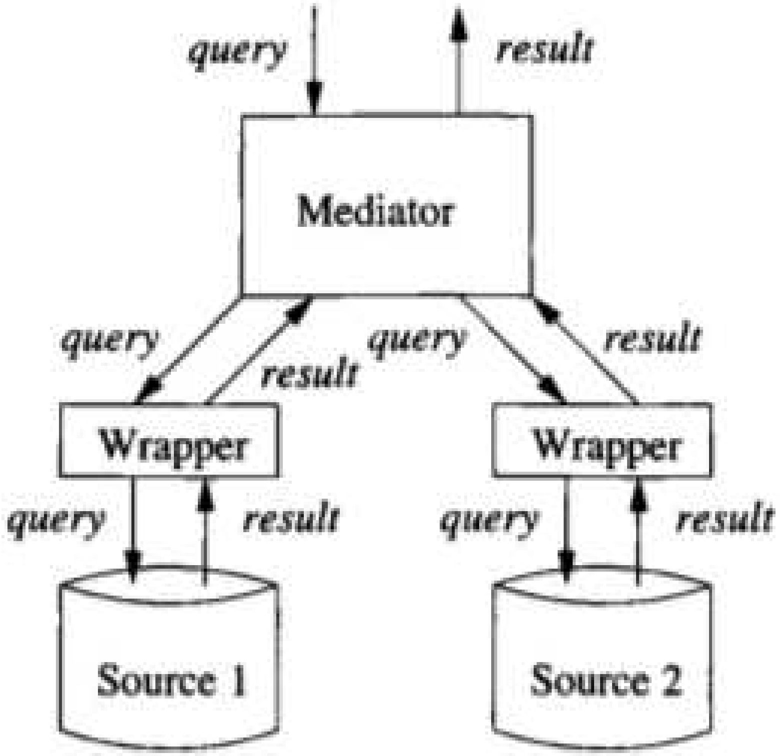
Data Replication (Image: Lloyd, et. al, SOSP 2011)

Federated Databases

- A collection of autonomous, heterogeneous database systems
- **Example**: Consider two databases for used cars
    - Database A
        - Cars (<u>carId</u>, type, model, engine, year, mileage, color, price)
    - Database B
        - Sedan (<u>id</u>, model, engineCapacity, year, mileage, description)
        - Suv (<u>id</u>, model, engineCapacity, year, mileage, description)
        - Sports (<u>id</u>, model, engineCapacity, year, mileage, description)
        - Pricing (<u>id</u>, price, specialPrice)

Federated Databases (cont.)

- Access databases using a multidatabase system
  - Also known as mediator system
- Provides illusion of logical integrated database



(Garcia-Molina, Ullman, & Widom, 2009)

Federated Databases (cont.)

- Database A
  - Cars (carId, type, model, engine, year, mileage, color, price)
- Database B
  - Sedan (id, model, engineCapacity, year, mileage, description)
  - Suv (id, model, engineCapacity, year, mileage, description)
  - Sports (id, model, engineCapacity, year, mileage, description)
  - Pricing (id, price, specialPrice)
- Mediator's Schema
  - Autos (autoId, type, model, engine, year, mileage, price)

Federated Databases (cont.)

- Database A
  - Cars (carId, type, model, engine, year, mileage, color, price)
- Database B
  - Sedan (id, model, engineCapacity, year, mileage, description)
  - Suv (id, model, engineCapacity, year, mileage, description)
  - Sports (id, model, engineCapacity, year, mileage, description)
  - Pricing (id, price, specialPrice)
- Mediator's Schema
  - Autos (autoId, type, model, engine, year, mileage, price)
- Query *Q* on mediator's schema:

SELECT autoId, model, year, price
FROM Autos
WHERE type = "sedan"

Federated Databases (cont.)

- Database A
  - Cars (carId, type, model, engine, year, mileage, color, price)
- Database B
  - Sedan (id, model, engineCapacity, year, mileage, description)
  - Suv (id, model, engineCapacity, year, mileage, description)
  - Sports (id, model, engineCapacity, year, mileage, description)
  - Pricing (id, price, specialPrice)
- Mediator's Schema
  - Autos (autoId, type, model, engine, year, mileage, price)
- Reformulated query Q_A on database A:

```
SELECT carId AS autoId, model, year, price
FROM   Cars
WHERE  type = "sedan"
```

Federated Databases (cont.)

- Database A
  - Cars (carId, type, model, engine, year, mileage, color, price)
- Database B
  - Sedan (id, model, engineCapacity, year, mileage, description)
  - Suv (id, model, engineCapacity, year, mileage, description)
  - Sports (id, model, engineCapacity, year, mileage, description)
  - Pricing (id, price, specialPrice)
- Mediator's Schema
  - Autos (autoId, type, model, engine, year, mileage, price)
- Reformulated query Q_B on database B:

```
SELECT id AS autoId, model, year, price
FROM   Sedan s JOIN Pricing p ON s.id = p.id
```

Federated Databases (cont.)

```
SELECT   autoId, model, year, price
FROM     Autos
WHERE    type = "sedan"
```

```
SELECT   carId AS autoId, model, year, price
FROM     Cars
WHERE    type = "sedan"
UNION
SELECT   id AS autoId, model, year, price
FROM     Sedan s JOIN Pricing p ON s.id = p.id
```

A brief history of DBMS

Year	Relational DBMS	Distributed DBMS	Parallel DBMS
1970	Codd's paper on relational data model		
1973	System R (IBM Research)		
1974	INGRES (UC Berkeley)		
1976		SDD-1 (Computer Corp. of America)	
1977		Distributed INGRES (UC Berkeley)	
1978	Oracle Version 1		Teradata DBMS
1981	IBM's SQL/DS (aka DB2) Informix's RDBMS	System R^* (IBM Research)	
1982			Super DB Computer (Univ. Tokyo)
1985	POSTGRES (UC Berkeley)		
1986		Oracle DDBMS	Gamma (Univ. Wisconsin) XPRS (UC Berkeley)
1987	Sybase SQL Server		NonStop SQL (Tandem)
1988			Bubba (MCC)
1989	SQL Server 1.0 (Microsoft)		
1990		IBM's DRDA	DB2 Parallel Edition (IBM)
1991			Oracle Parallel Server
1995	Sybase IQ		
2001			Oracle RAC
2004	MonetDB (CWI)		
2005	C-Store (MIT,Yale,Brandeis,Brown,UMass) Vertica		
2006		Bigtable (Google)	
2007		Dynamo (Amazon) H-Store (Brown,CMU,MIT,Yale)	
2008		PNUTS (Yahoo!) Cassandra (Facebook)	
2009		Voldemort (LinkedIn)	
2012			SQL Server PDW (Microsoft)
2014		Azure DocumentDB (Microsoft)	

Relational DBMS

- Initially targeted at business processing applications
  - OLTP = On-Line Transaction Processing
  - Characteristics: small update ACID transactions
- 1970 - Edgar Codd's paper on relational data model
- 1973 - System R (IBM Research)
- 1974 - INGRES (Univ. of California at Berkeley)
- Products: IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, SAP Sybase, etc.

Early Distributed DBMS

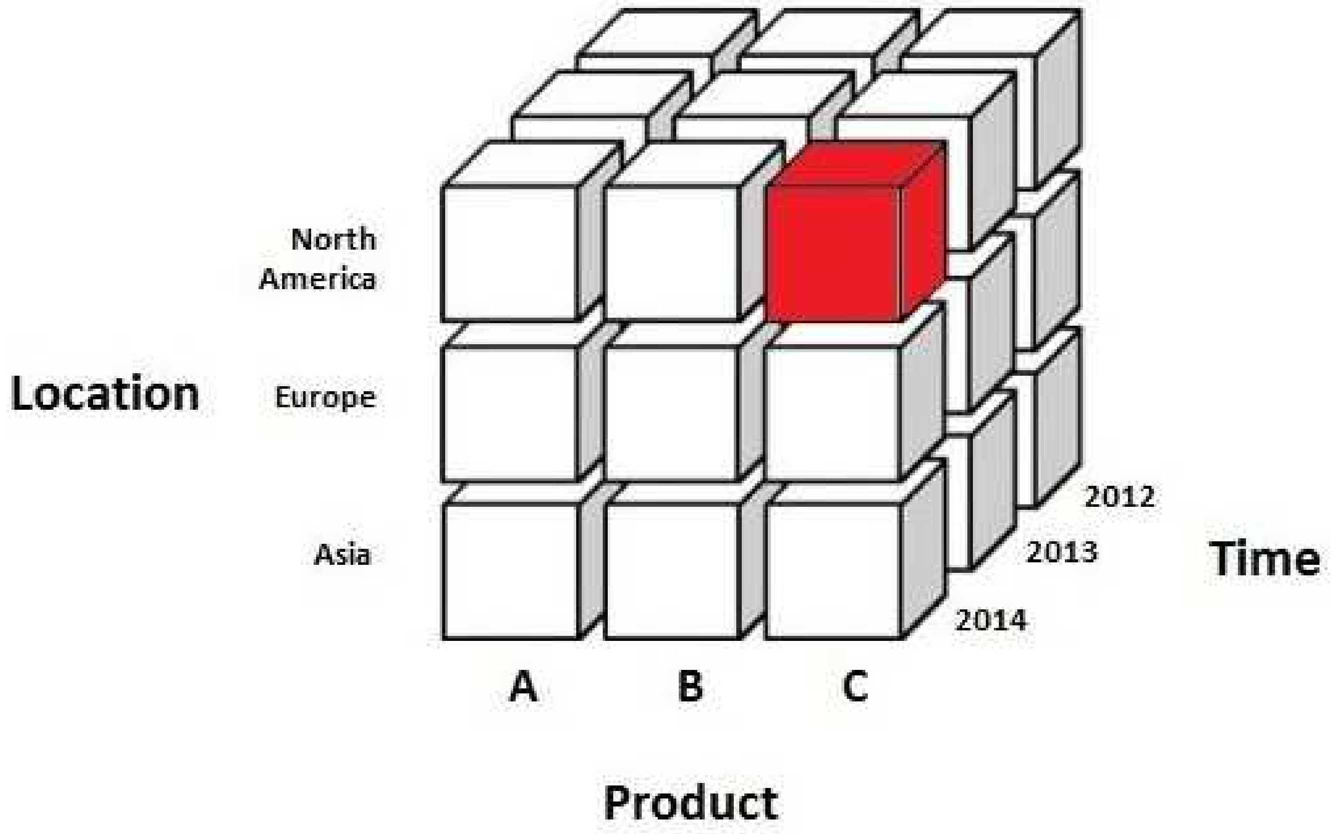
- Targeted to support the organizational structure of distributed enterprises
- 1976 - SDD-1 (Computer Corporation of America)
- 1977 - Distributed INGRES (U.C. Berkeley)
- 1981 - R^* (IBM Research)

Parallel DBMS

- Targeted at decision support systems (DSSs)
  - OLAP = On-line Analytical Processing
  - Characteristics: Complex read-mostly queries on large data
- Early Parallel DBMS
  - 1978 - Teradata DBMS
  - 1982 - Super Database Computer (Univ. Tokyo)
  - 1986 - Gamma (Univ. Wisconsin-Madison), XPRS (UC Berkeley)
  - 1987 - NonStop SQL (Tandem)
  - 1988 - Bubba (MCC)

OLAP: Multidimensional Data Model

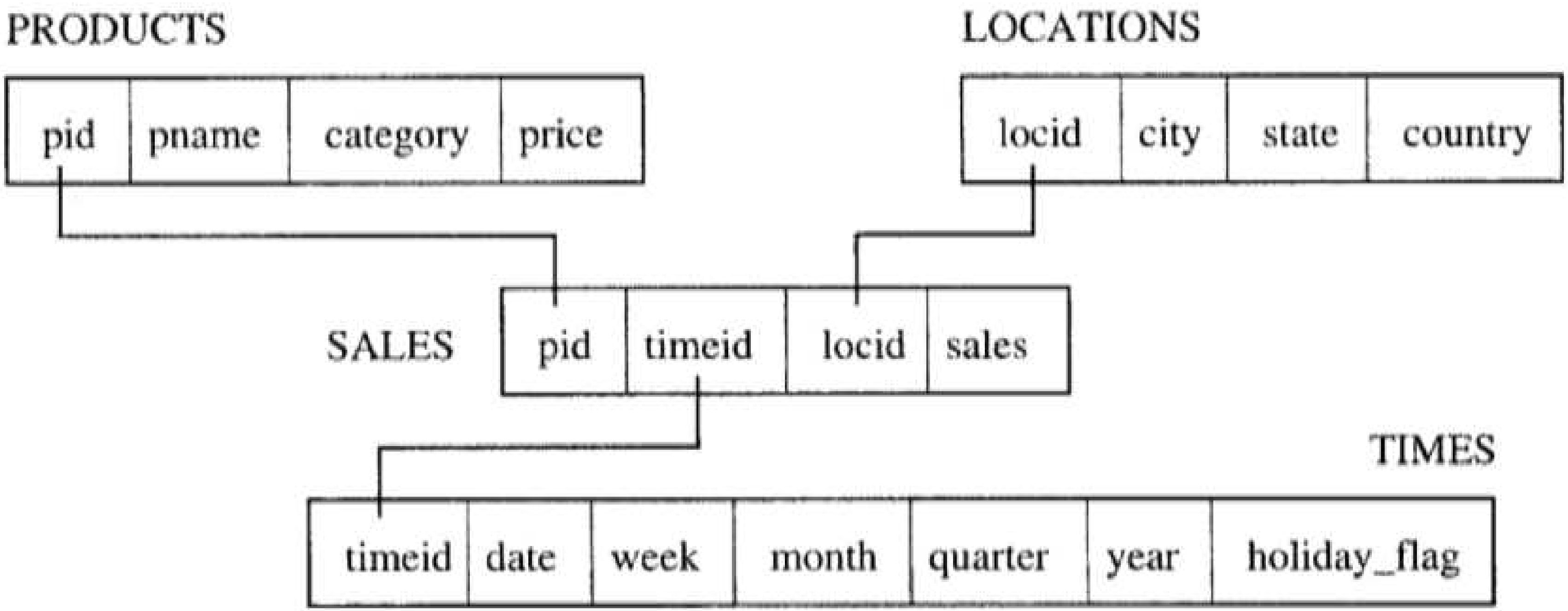
- Stores a collection of numeric measures
- Each measure depends on a set of dimensions



http://www.openit.com/faster-analysis-with-olap/

OLAP: Star Schema

- Data is modeled using a fact table & dimension tables



(Ramakrishnan & Gehrke, 2003)

OLAP: Multidimensional Aggregation

Find the total sales

SELECT SUM(sales) FROM Sales

Find the total sales for each state

SELECT L.state, SUM(S.sales)
FROM Sales S JOIN Locations L ON S.locid = L.locid
GROUP BY L.state

Find the total sales for each city and year

SELECT L.city, T.year, SUM(S.sales)
FROM Sales S JOIN Locations L ON S.locid = L.locid
JOIN Times T ON S.timeid = T.timeid
GROUP BY L.city, T.year

Find the total sales for each city, year, category

SELECT L.city, T.year, P.category, SUM(S.sales)
FROM Sales S JOIN Locations L ON S.locid = L.locid
JOIN Times T ON S.timeid = T.timeid JOIN Products P ON S.pid = P.pid
GROUP BY L.city, T.year, P.category

OLAP: Multidimensional Aggregation

Find the total sales for each city, year, category
Find the total sales for each city, year
Find the total sales for each city, category
Find the total sales for each year, category
Find the total sales for each city
Find the total sales for each year
Find the total sales for each category
Find the total sales

```
SELECT      L.city, T.year, P.category, SUM(S.sales)
FROM        Sales S JOIN Locations L ON S.locid = L.locid
            JOIN Times T ON S.timeid = T.timeid
            JOIN Products P ON S.pid = P.pid
GROUP BY    CUBE (L.city, T.year, P.category)
```

OLAP: Analytic Window Functions

For each state and month, compute its moving average sales over three months (specifically, average sales over the current and two preceding months)

state	month	sales
CA	2019-01-01	100
CA	2019-02-01	200
CA	2019-03-01	300
NY	2019-01-01	200
NY	2019-02-01	400
NY	2019-03-01	600
NY	2019-04-01	800

state	month	movingavg
CA	2019-01-01	100.0000
CA	2019-02-01	150.0000
CA	2019-03-01	200.0000
NY	2019-01-01	200.0000
NY	2019-02-01	300.0000
NY	2019-03-01	400.0000
NY	2019-04-01	600.0000

OLAP: Analytic Window Functions (cont.)

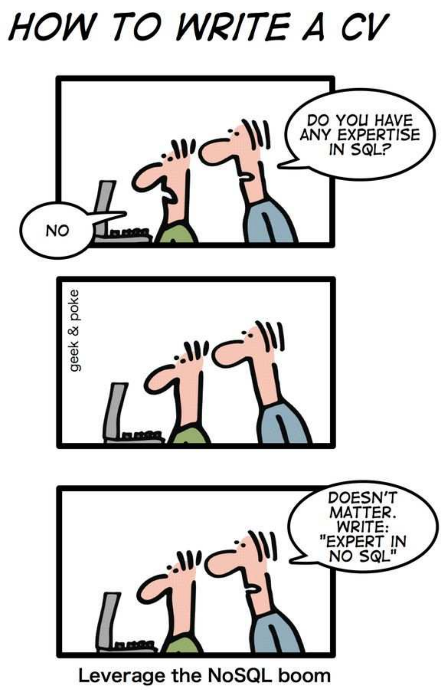
state	month	sales
CA	2019-01-01	100
CA	2019-02-01	200
CA	2019-03-01	300
NY	2019-01-01	200
NY	2019-02-01	400
NY	2019-03-01	600
NY	2019-04-01	800

state	month	movingavg
CA	2019-01-01	100.0000
CA	2019-02-01	150.0000
CA	2019-03-01	200.0000
NY	2019-01-01	200.0000
NY	2019-02-01	300.0000
NY	2019-03-01	400.0000
NY	2019-04-01	600.0000

```
SELECT   L.state, T.month, AVG(S.sales) OVER W AS movingAvg
FROM     Sales S NATURAL JOIN Times T NATURAL JOIN Location L
WINDOW   W AS (
         PARTITION BY L.state
         ORDER BY T.month
         ROWS 2 PRECEDING
)
```

NoSQL Systems

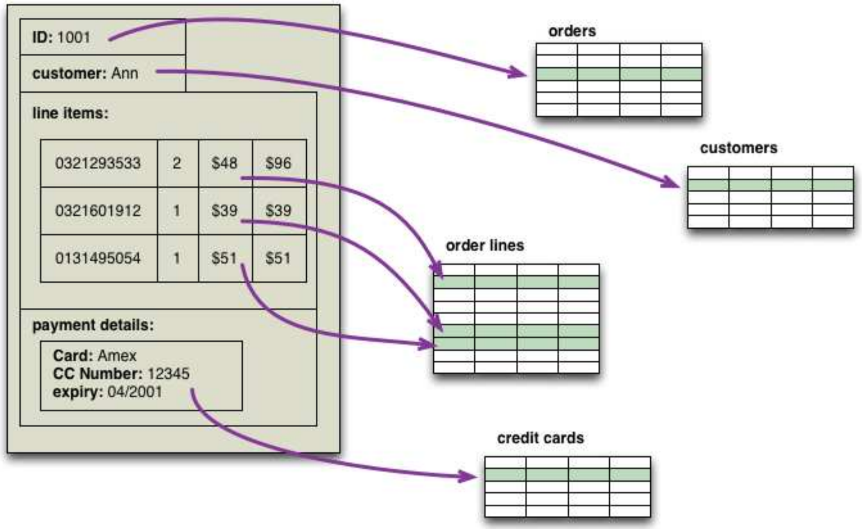
- Early NoSQL Systems
  - Google's Bigtable
  - Amazon's Dynamo
  - Yahoo!'s PNUTS
- Data Models:
  - Key-value
  - Column family
  - Document
  - Graph



- Features of many early NoSQL systems
  - Schema-less data
  - Simple access API (put & get) instead of query language
  - Limited/No ACID transactional support
  - Weak consistency for replicated data

NoSQL Database Systems

- **Key-value stores** (e.g., Dynamo, Redis)
- **Column-family stores** (e.g., BigTable, Cassandra, HBase)
- **Document stores** (e.g., MarkLogic, MongoDB)



(Martin Fowler, 2012)

- **Graph database systems** (e.g., JanusGraph, Neo4j)

NewSQL Database Systems

- Targeted at OLTP workloads
- Features
  - Relational data model
  - SQL query language
  - ACID transactions
  - Runs on distributed cluster of shared-nothing nodes
- Some examples:
  - Clustrix
  - CockroachDB
  - Google’s Cloud Spanner
  - MemSQL
  - VoltDB

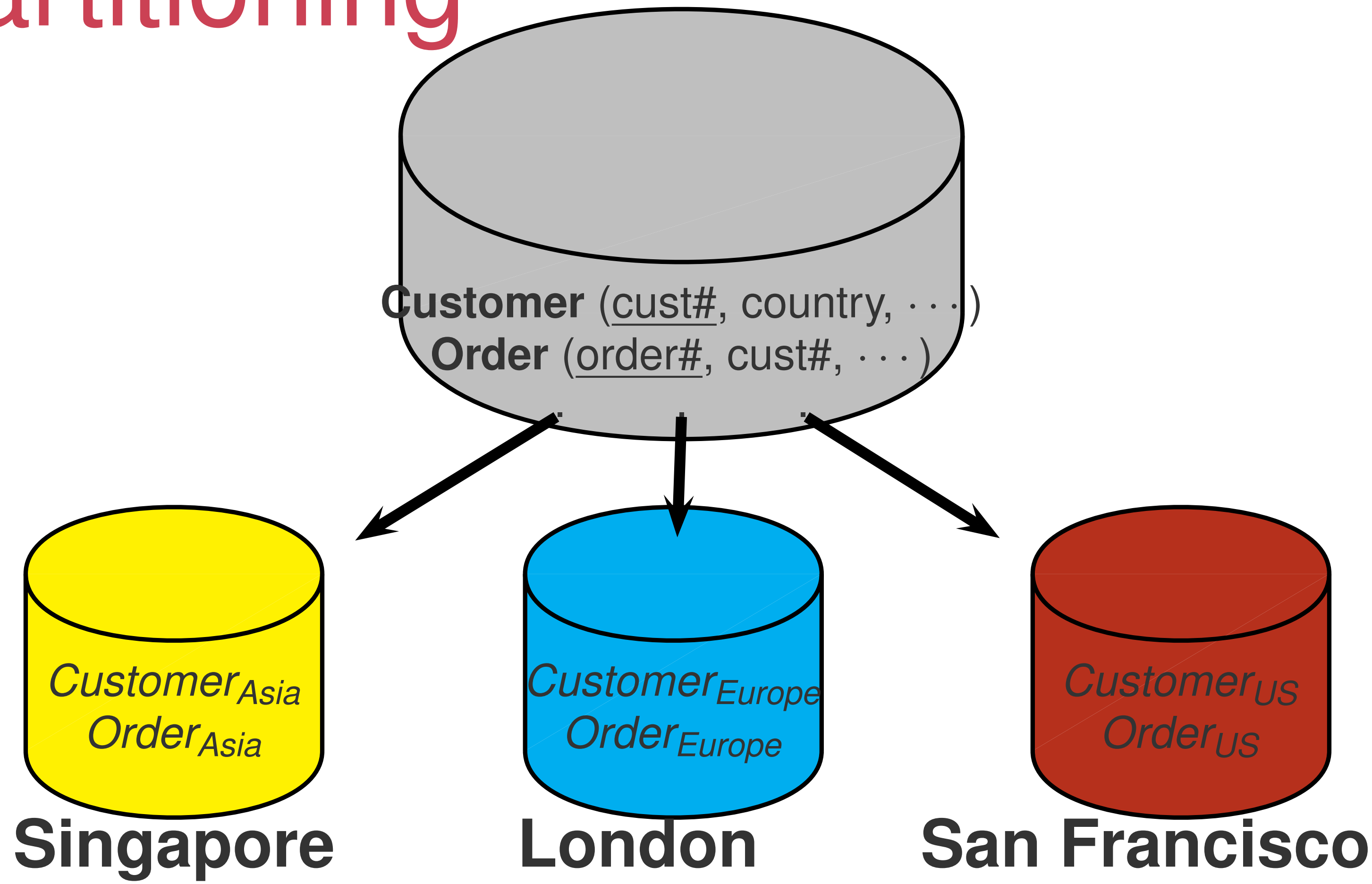
Topics in Distributed DBMS

- Database Design
  - Data partitioning / fragmentation
- Storage & Indexing
- Query Processing
- Transaction Management
  - Atomicity, isolation, & durability of distributed transactions
- Data Replication

Review of Relational Algebra

$\sigma_{A>5}(R)$	SELECT * FROM R WHERE A $>$ 5
$\pi_{X,Y,Z}(R)$	SELECT DISTINCT X, Y, Z FROM R
$R \bowtie_{R.A=S.A} S$ $R \bowtie_A S$	SELECT * FROM R JOIN S ON R.A = S.A
$R \ltimes_A S$	SELECT * FROM R WHERE EXISTS (SELECT * FROM S WHERE R.A = S.A)

Data Partitioning



- $Customer_{Asia} = \sigma_{country \in \{Singapore, Malaysia, \cdots\}}(Customer)$
- $Order_{Asia} = Order \ltimes_{cust\#} Customer_{Asia}$

Indexing

Customers

cust#	cname	city
1	Alice	Singapore
2	Bob	Jarkata
3	Carol	Bangkok
4	Dave	Jarkata
5	Eve	Singapore
6	Fred	Penang
7	George	Hanoi
8	Hal	Bangkok
9	Ivy	Singapore
10	Joe	Penang
11	Kathy	Singapore
12	Larry	Jarkata

Index on Customers.city

Bangkok	3, 8
Hanoi	7
Jarkata	2, 4, 12
Penang	6, 10
Singapore	1, 5, 9, 11

Indexing (cont.)

Partitioned Data	Local Index	Global Index

Customers$_1$

cust#	cname	city
3	Carol	Bangkok
6	Fred	Penang
9	Ivy	Singapore
12	Larry	Jarkata

Index I_1 on Customers$_1$.city

Bangkok	3
Jarkata	12
Penang	6
Singapore	9

Index I_1

Jakarta	2, 4, 12
Singapore	1, 5, 9, 11

Customers$_2$

cust#	cname	city
1	Alice	Singapore
4	Dave	Jarkata
7	George	Hanoi
10	Joe	Penang

Index I_2 on Customers$_2$.city

Hanoi	7
Jakarta	4
Penang	10
Singapore	1

Index I_2

Penang	6, 10

Customers$_3$

cust#	cname	city
2	Bob	Jarkata
5	Eve	Singapore
8	Hal	Bangkok
11	Kathy	Singapore

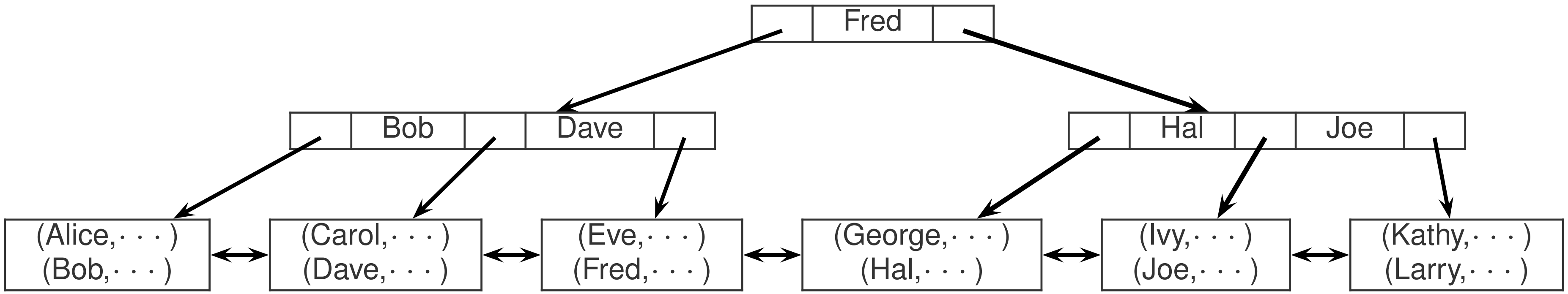
Index I_3 on Customers$_3$.city

Bangkok	8
Jakarta	2
Singapore	5, 11

Index I_3

Bangkok	3, 8
Hanoi	7

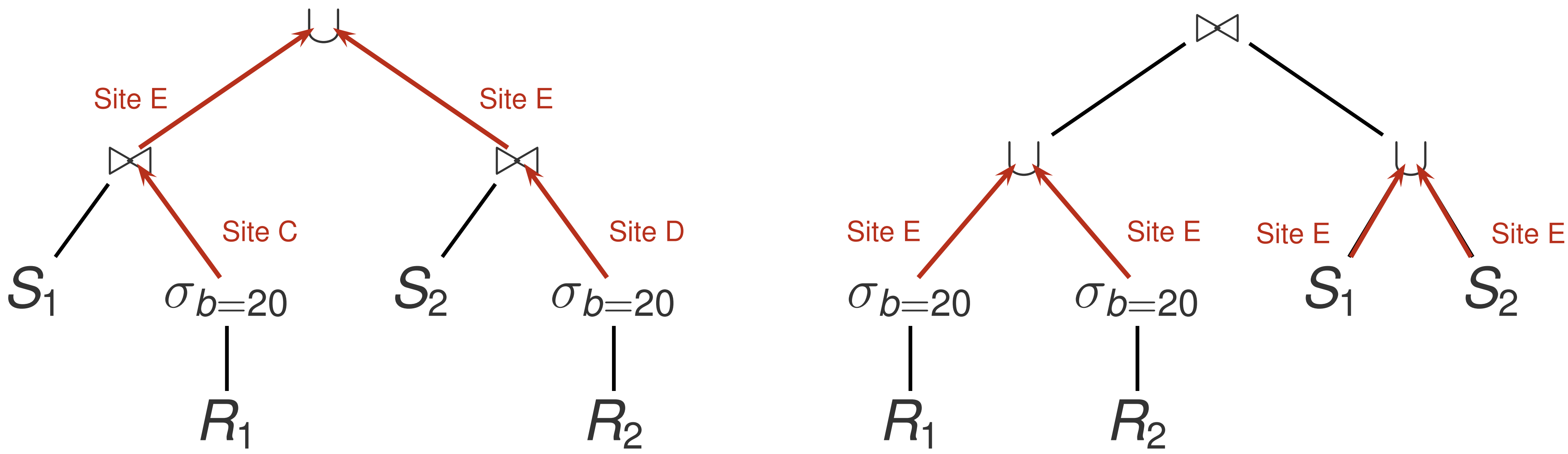
B^+-tree Index



Fred
Bob
Dave
Hal
Joe
(Alice,· · ·)
(Bob,· · ·)
(Carol,· · ·)
(Dave,· · ·)
(Eve,· · ·)
(Fred,· · ·)
(George,· · ·)
(Hal,· · ·)
(Ivy,· · ·)
(Joe,· · ·)
(Kathy,· · ·)
(Larry,· · ·)

Distributed Query Processing

- **Site A**: $R_1 = \sigma_{a \leq 10}(R)$ **Site B**: $R_2 = \sigma_{a > 10}(R)$
- **Site C**: $S_1 = \sigma_{a \leq 10}(S)$ **Site D**: $S_2 = \sigma_{a > 10}(S)$
- **Site E**: Query $Q = \sigma_{b=20}(R) \bowtie_a S$



Plan 1

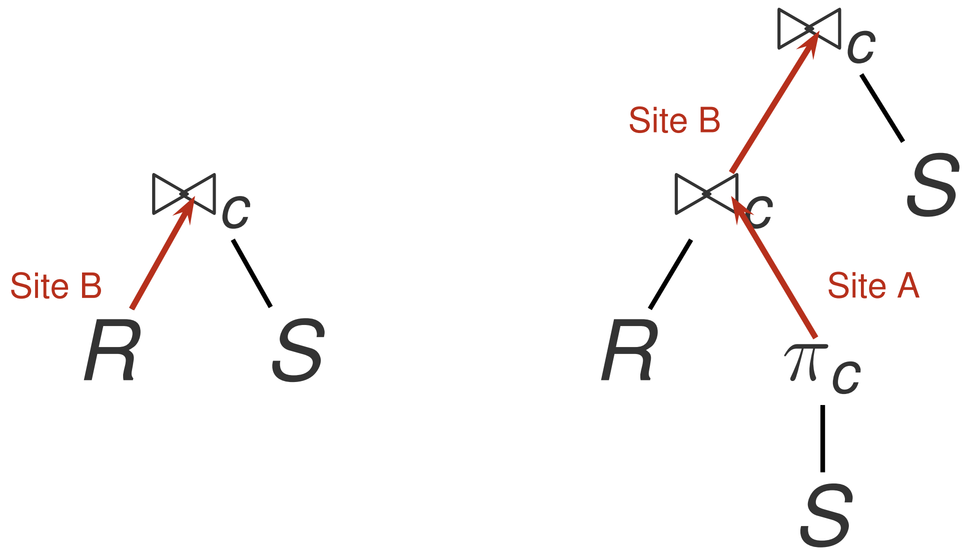
$(S_1 \bowtie_a \sigma_{b=20}(R_1)) \cup (S_2 \bowtie_a \sigma_{b=20}(R_2))$

Plan 2

$(\sigma_{b=20}(R_1) \cup \sigma_{b=20}(R_2)) \bowtie_a (S_1 \cup S_2)$

Distributed Query Processing (cont.)

- **Site A**: R **Site B**: S
- Query $Q = R \bowtie_c S$



Plan 1 **Plan 2**

Review of ACID Transactions

- Atomicity: Xact is either executed completely or not at all
- Consistency: Xact preserves database consistency
- Isolation: Execution of a Xact is isolated from other Xacts
- Durability: If a Xact commits, its effects persist

Review of ACID Transactions (cont.)

- **Transactions**:

T_1:	Read(x)	T_2:	Read(x)
	x = x - 100		Read(y)
	Write(x)		
	Read(y)		
	y = y + 100		
	Write(y)		

- **Serial schedules**:
  - $R_1(x), W_1(x), R_1(y), W_1(y), R_2(x), R_2(y)$
  - $R_2(x), R_2(y), R_1(x), W_1(x), R_1(y), W_1(y)$

- A transaction schedule is **serializable** if it is **view equivalent** to a serial schedule

- Serializable schedule: $R_2(x), R_1(x), W_1(x), R_1(y), R_2(y), W_1(y)$

- Non-serializable schedule: $R_1(x), W_1(x), R_2(x), R_1(y), R_2(y), W_1(y)$

Distributed Transactions

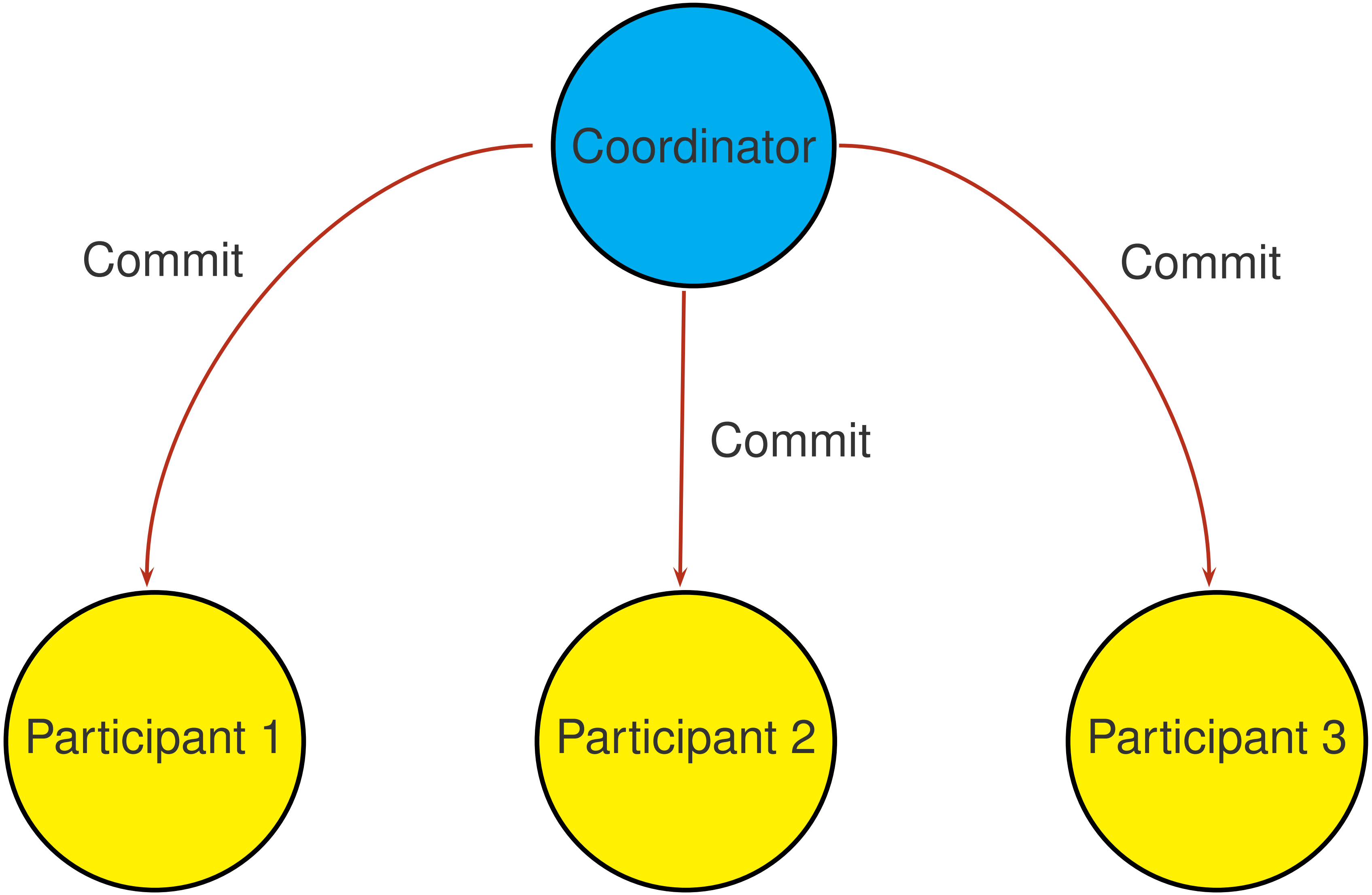
T_1: Read(x)
Write(y)

T_2: Read(y)
Write(x)

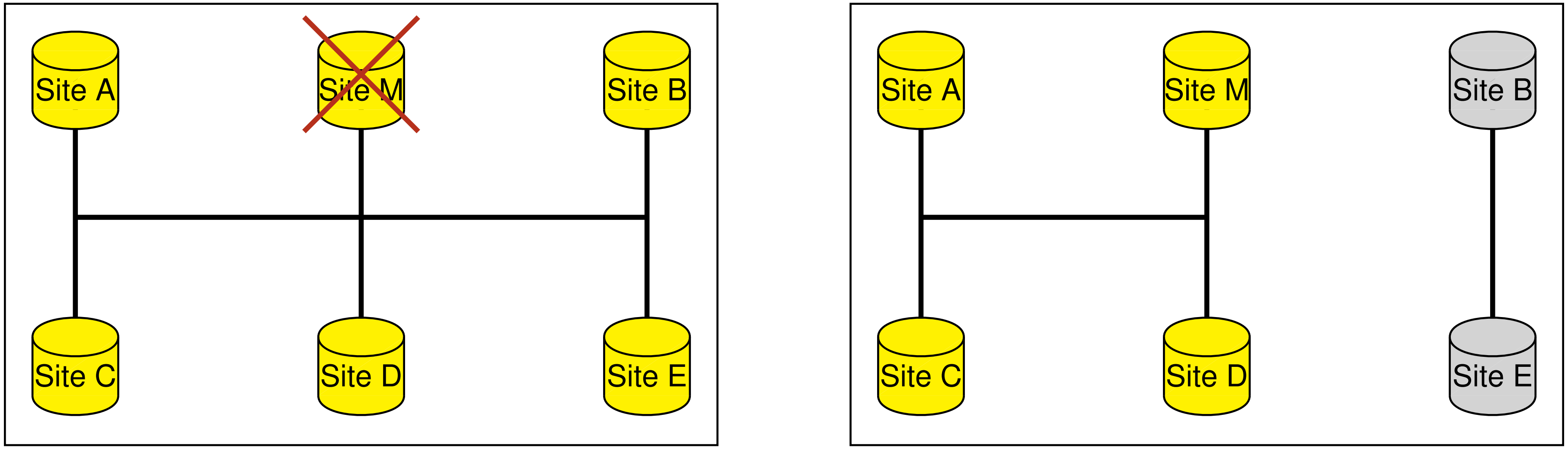
Suppose x is stored at Site A & y is stored at Site B

Schedule at Site A: $R_1(x), W_2(x)$
Schedule at Site B: $R_2(y), W_1(y)$

Distributed Commit



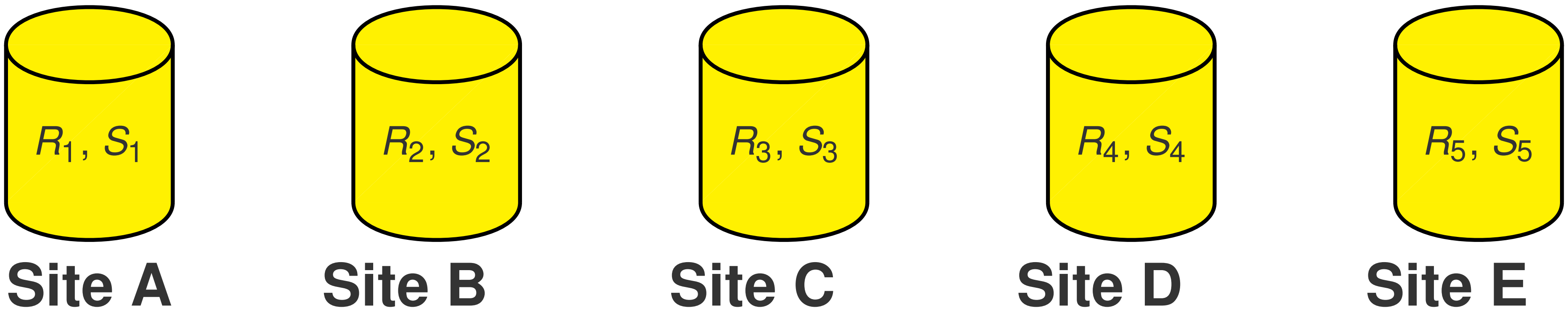
Availability



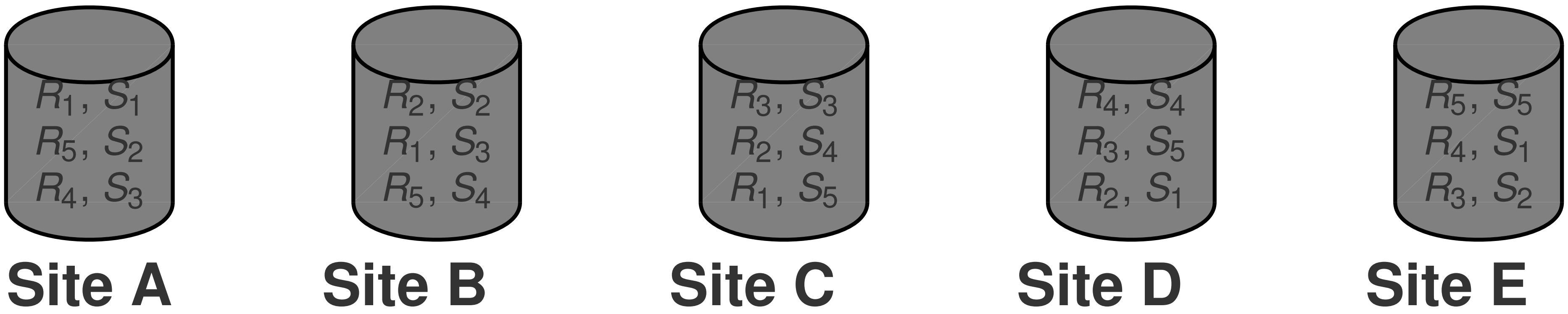
Availability (cont.)

Availability %	Downtime per year
99	3.65 days
99.9	8.77 hours
99.99	52.60 minutes
99.999	5.26 minutes

Data Replication



(a) Non-replicated Database



(b) Replicated Database (replication factor = 3)

References

- T. Özsu & P. Valdureiz, *Introduction*, Chapter 1, Principles of Distributed Database Systems, 4^{th} Edition, 2020
- M. Stonebraker, R. Cattell, *10 rules for scalable performance in 'simple operation' datastores*, CACM 54(6), 2011, 72-80