Dr Greg Wadley

INFO90002 Database Systems & Information Modelling

Week 08

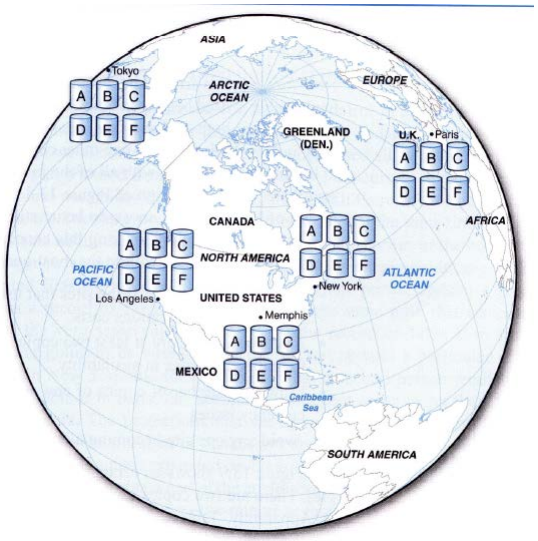
Distributed Databases

Today's lecture

- What is a distributed database?
- Why are they used, and how they work
- Pros and cons of different approaches
  - material in this lecture is drawn from Hoffer et al. (2013) *Modern Database Management* 11th edition, chapter 12, available online at http://wps.prenhall.com/bp_hoffer_mdm_11/230/58943/15089539.cw/index.html
  - pictures on this page are from Gillenson (2005) *Fundamentals of Database Management Systems*

centralized database

distributed database



replicated database

Recent trends

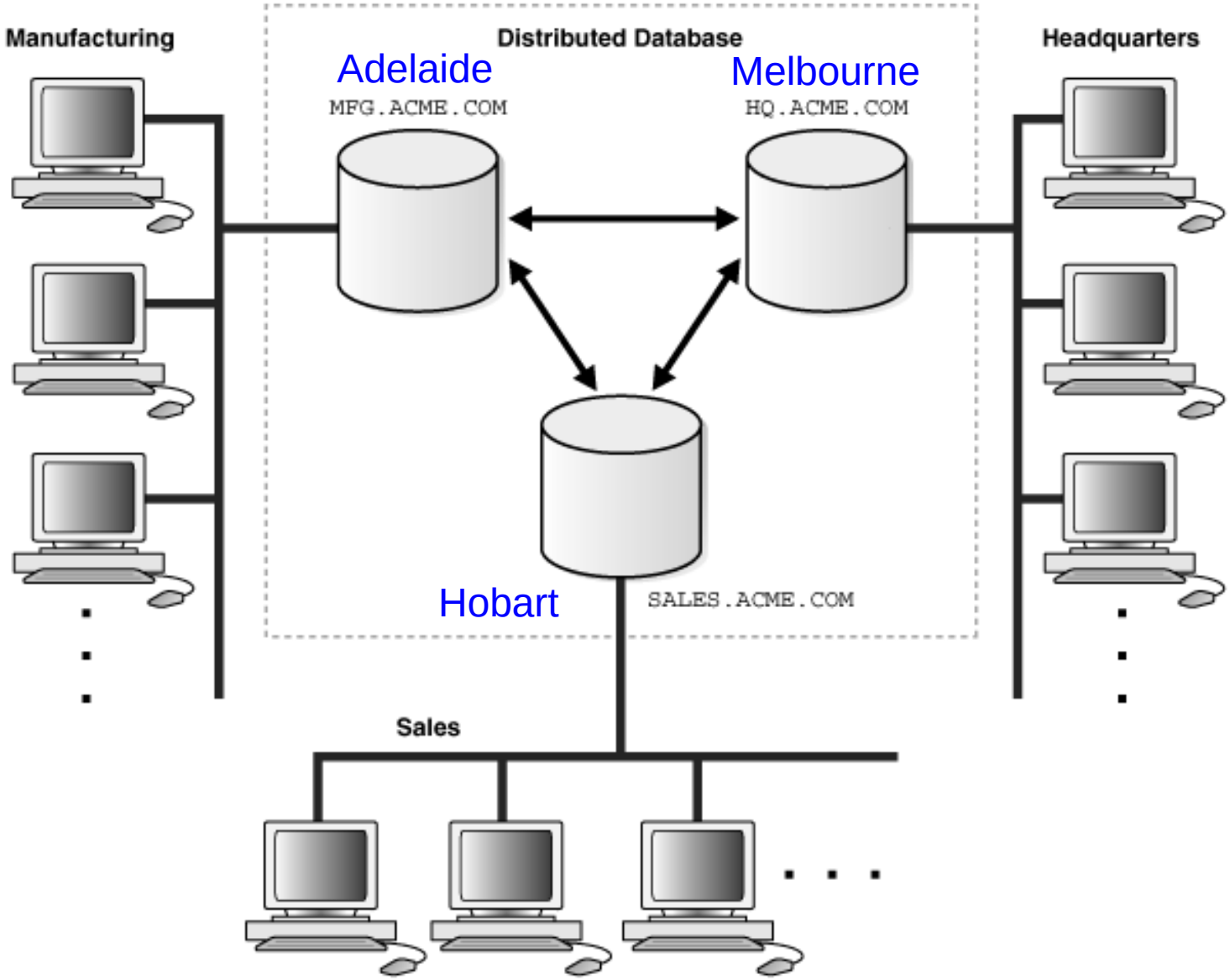
- *Distributed* storage and processing is currently one of the most important research topics in database
- example use scenario: Internet startup
  - faces sudden increase in number and distribution of users
- see industry panel discussion
  - at *Google I/O 2013* (software developer conference)
  - https://youtu.be/zxwsOueJU4Q

Definitions

- Distributed Database
  - a single logical database physically spread across multiple computers in multiple locations that are connected by a data communications link
  - *appears to users as though it is one database*
- Decentralized Database
  - a collection of independent databases which are not networked together as one logical database
  - appears to users as though many databases

- We are concerned with *distributed* databases

Example Distributed Database



Distributed DBMS Advantages

- Good fit for geographically distributed organizations / users
- Data located near site with greatest demand
- Faster data access (to local data)
- Faster data processing
  - workload spread out across many machines
- Allows modular growth
  - add new servers as load increases
- Increased reliability and availability
  - less danger of single-point failure
- Supports database recovery
  - data is replicated across multiple sites

Disadvantages

- Complexity of management and control
  - db or application must stitch together data across sites
- Data integrity
  - additional exposure to improper updating
- Security
  - many server sites -> higher chance of breach
- Lack of standards
  - different DBMS vendors use different protocols
- Increased training costs
  - more complex IT infrastructure
- Increased storage requirements
  - if there are multiple copies of data

Objectives and Trade-offs

- Location transparency
  - a user needn't know where particular data are stored
  - (details next slide)
- Local autonomy
  - a node can continue to function for local users if connectivity to the network is lost
  - (details next slide +1)
- Trade-offs
  - Availability vs Consistency
  - Synchronous vs Asynchronous updates

Location Transparency

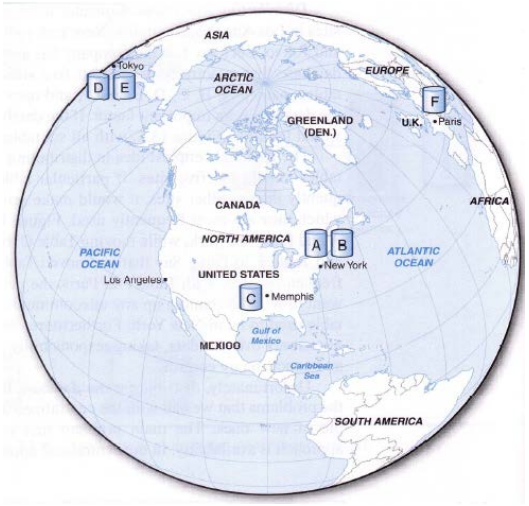
- A user (or program) using data need not know the location of the data in the network of DBMS's
- Requests to retrieve or update data from any site are automatically forwarded by the system to the site or sites related to the processing request
- All data in the network appears as a single logical database stored at one site to the users
- A single query can join data from tables in multiple sites

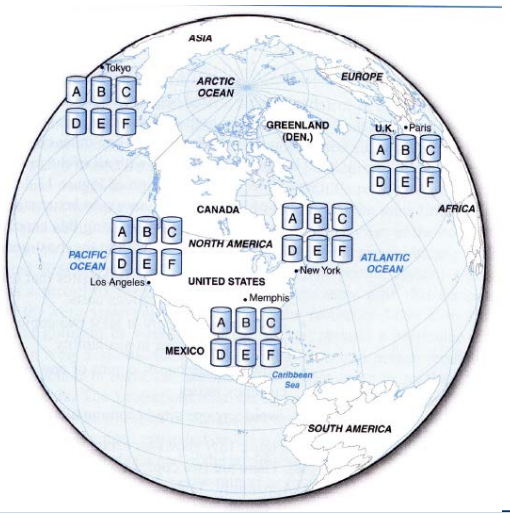
Local Autonomy

- Users can administer their local database
  - control local data
  - administer security
  - log transactions
  - recover when local failures occur
  - provide full access to local data
- Being able to operate locally when connections to other databases fail

Distribution options

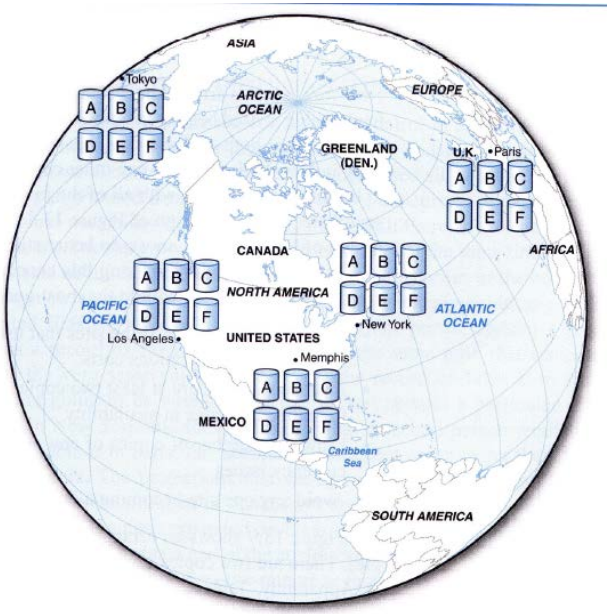
- Data replication
  - Data copied across sites
- Horizontal partitioning
  - Table rows distributed across sites
- Vertical partitioning
  - Table columns distributed across sites
- Combinations of the above





Replication advantages

- High reliability due to redundant copies
- Fast access to local data
- May avoid complicated distributed integrity routines
  - if replicated data is refreshed at scheduled intervals
- Decouples nodes
  - transactions proceed even if some nodes are down
- Reduced network traffic at prime time
  - if updates can be delayed
- This is currently popular as a way of achieving high availability for global systems. Most NoSQL databases offer replication.



Replication disadvantages

- Need more storage space
- Integrity: can retrieve incorrect data if updates have not arrived
- Takes time for update operations
  - high tolerance for out-of-date data may be required
  - updates may cause performance problems for busy nodes
- Network communication capabilities
  - updates place heavy demand on telecommunications

Better for non-volatile (read-only) data

Replication: Synchronous updates

- Data is continuously kept up to date
  - users anywhere can access data and get the same answer.
- If any copy of a data item is updated anywhere on the network, the same update is immediately applied to all other copies or it is aborted.
- Ensures data integrity and minimizes the complexity of knowing where the most recent copy of data is located.
- Can result in slow response time and high network usage
  - the DDBMS spends considerable time checking that an update is accurately and completely propagated across the network.

Replication: Asynchronous updates

- Some delay in propagating data updates to remote databases
  - some degree of at least temporary inconsistency is tolerated
  - may be ok it is temporary and well managed
- Tends to have acceptable response time
  - updates happen locally and data replicas are synchronized in batches and predetermined intervals
- May be more complex to plan and design
  - need to ensure the right level of data integrity and consistency
- Suits some information systems more than others
  - compare commerce/finance systems with social media

Horizontal partitioning

- Different rows of a table at different sites
- Advantages
  - data stored close to where it is used
    - efficiency
  - local access optimization
    - better performance
  - only relevant data is stored locally
    - security
  - unions across partitions
    - ease of query
- Disadvantages
  - accessing data across partitions
    - inconsistent access speed
  - no data replication
    - backup vulnerability

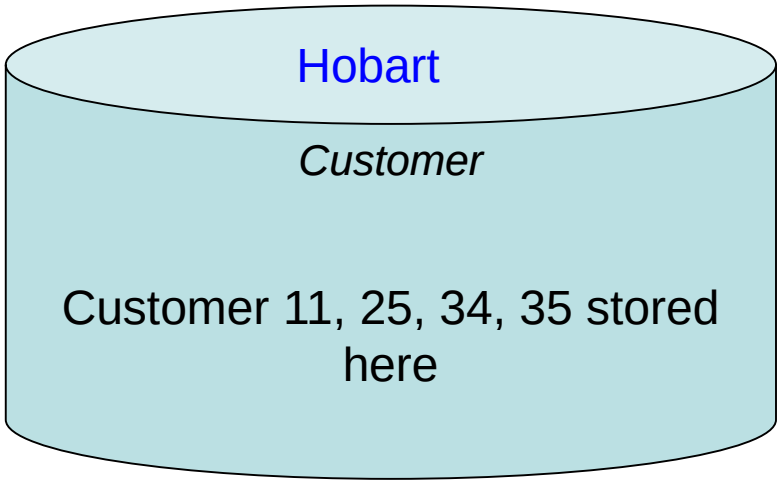
AcctNumber	CustomerName	BranchName	Balance
200	Jones	Lakeview	1000
426	Dorman	Lakeview	796
683	McIntyre	Lakeview	1500
252	Elmore	Lakeview	330

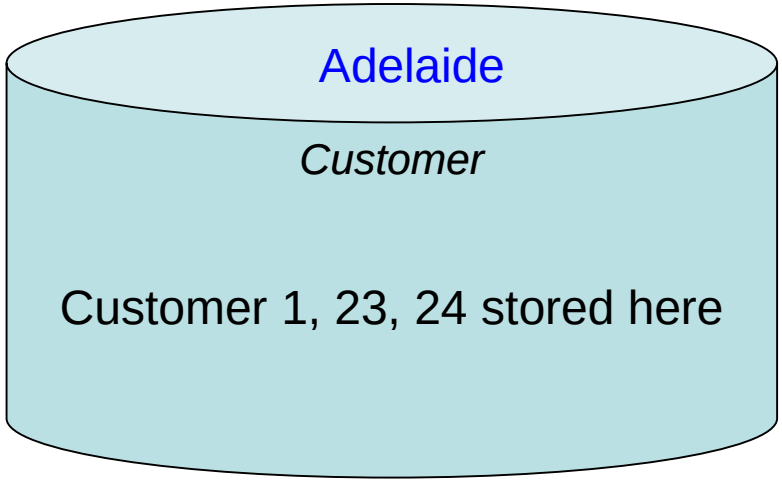
AcctNumber	CustomerName	BranchName	Balance
324	Smith	Valley	250
153	Gray	Valley	38
500	Green	Valley	168

Example horizontal partitioning

CustNbr	Name	Address	State	Limit	Balance
1	Jon	Smith st	SA	100	10
11	Harry	Jones St	TAS	200	5
23	Pete	Fred Place	SA	500	20
24	Jane	Arc Crt	SA	100	35
25	Mary	Home Lane	TAS	410	0
34	Liu	Busy Rd	TAS	125	100
35	Agreda	First St	TAS	220	1

Horizontal Partitioning based on State





Vertical partitioning

- Different columns of a table at different sites
- Advantages and disadvantages are the same as for horizontal partitioning
  - except
    - combining data across partitions is more difficult because it requires joins (instead of unions)

(a) Engineering

PartNumber	DrawingNumber
P2	123-7
P7	621-0
P3	174-3
P1	416-2
P8	321-2
P9	400-1
P6	129-4

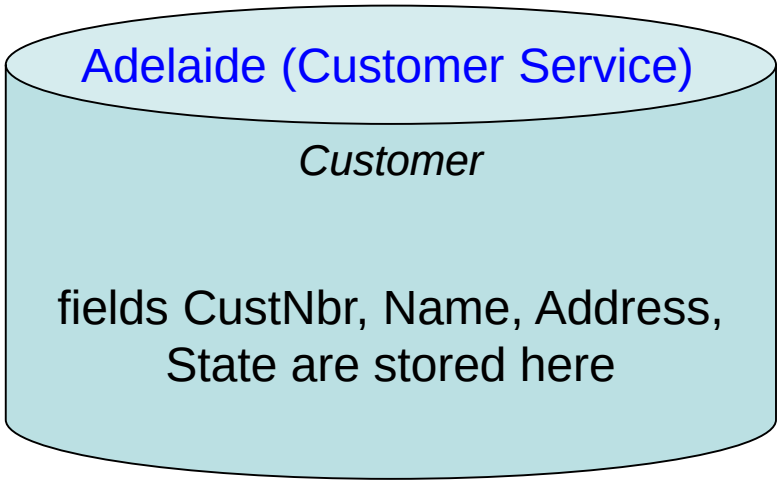
(b) Manufacturing

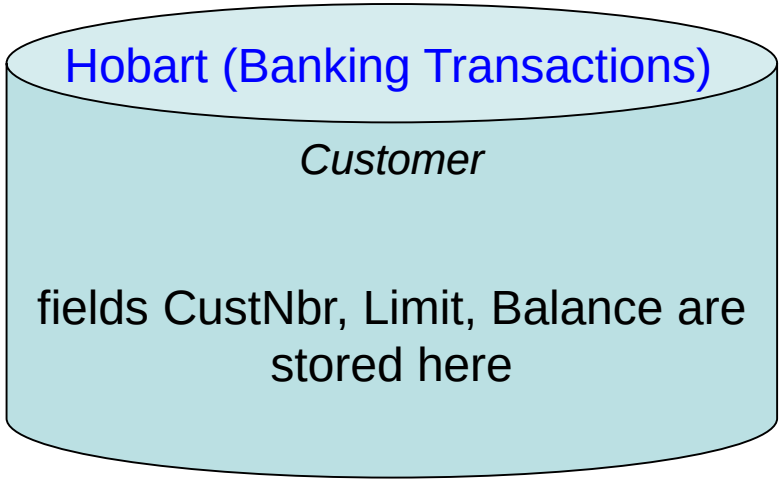
PartNumber	Name	Cost	QtyOnHand
P2	Widget	100	20
P7	Gizmo	550	100
P3	Thing	48	0
P1	Whatsit	220	16
P8	Thumzer	16	50
P9	Bobbit	75	200
P6	Nailit	125	200

Example vertical partitioning

CustNbr	Name	Address	State	Limit	Balance
1	Jon	Smith st	SA	100	10
11	Harry	Jones St	TAS	200	5
23	Pete	Fred Place	SA	500	20
24	Jane	Arc Crt	SA	100	35
25	Mary	Home Lane	TAS	410	0
34	Liu	Busy Rd	TAS	125	100
35	Agreda	First St	TAS	220	1

Vertical Partitioning based on column requirements





Comparing 5 configurations

- Centralized database, distributed access
  - DB is at one location, and accessed from everywhere
- Replication with periodic snapshot update
  - many locations, each data copy updated periodically
- Replication with near real-time synchronization of updates
  - many locations, each data copy updated in near real time
- Partitioned, integrated, one logical database
  - data partitioned across at many sites, within a logical database, and a single DBMS
- Partitioned, independent, nonintegrated segments
  - data partitioned across many sites.
  - independent, non-integrated segments
  - multiple DBMS, multiple computers

Comparison: Centralised

- Reliability
  - POOR: Highly dependent on central server
- Expandability
  - POOR: Limitations are barriers to performance
- Communications Overhead
  - VERY HIGH: Traffic from all locations goes to one site
- Manageability
  - VERY GOOD: One monolithic site requires little coordination
- Data Consistency
  - EXCELLENT: All users always access the same data

Comparison: Synchronised Replication

- Reliability
  - EXCELLENT: Redundancy and minimal delays
- Expandability
  - VERY GOOD: Cost of additional copies may be low and synchronization work only linear
- Communications Overhead
  - MEDIUM: Messages are constant, but some delays are tolerated
- Manageability
  - MEDIUM: Collisions add some complexity to manageability
- Data Consistency
  - VERY GOOD: Close to precise consistency

Comparison: Replicated with Snapshots

- Reliability
  - GOOD: Redundancy and tolerated delays
- Expandability
  - VERY GOOD: Cheap to add new servers
- Communications Overhead
  - LOW to MEDIUM: Not constant, but periodic snapshots can cause bursts of network traffic
- Manageability
  - VERY GOOD: Each copy is like every other one
- Data Consistency
  - MEDIUM: Fine as long as update delays are tolerable

Comparison: Integrated Partitions

- Reliability
  - GOOD: Effective use of partitioning and redundancy
- Expandability
  - VERY GOOD: New nodes get only data they need without changes in overall database design
- Communications Overhead
  - LOW to MEDIUM: Most queries are local, but queries that require data from multiple sites can cause a temporary load
- Manageability
  - DIFFICULT: Especially difficult for queries that need data from distributed tables, and updates must be tightly coordinated
- Data Consistency
  - VERY POOR: Considerable effort; and inconsistencies not tolerated

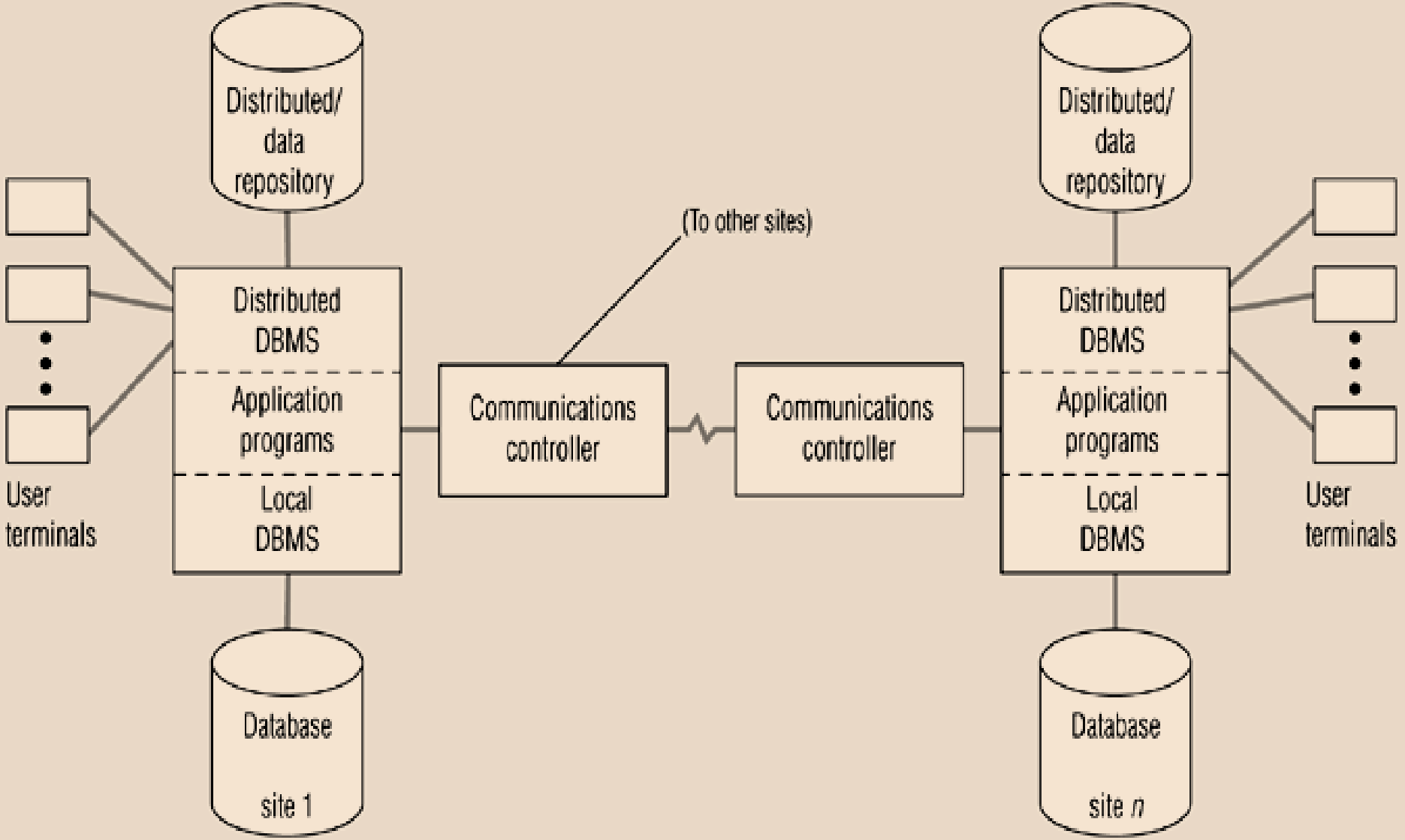
Comparison: Decentralised, Independent Partitions

- Reliability
  - GOOD: Depends on only local database availability
- Expandability
  - GOOD: New sites independent of existing ones
- Communications Overhead
  - LOW: Little if any need to pass data or queries across the network (if one exists)
- Manageability
  - VERY GOOD: Easy for each site, until there is a need to share data across sites
- Data Consistency
  - LOW: No guarantees of consistency; in fact, pretty sure of inconsistency

Functions of a distributed DBMS

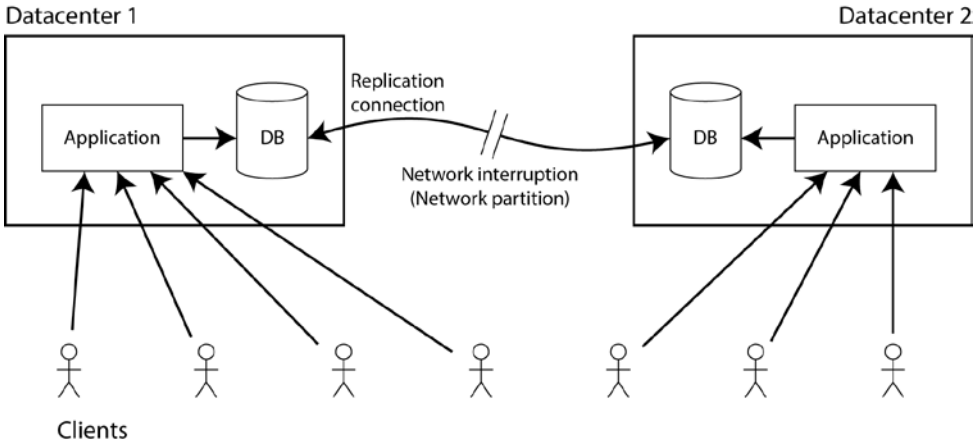
- Locate data with a distributed data dictionary
- Determine location from which to retrieve data and process query components
- DBMS translation between nodes with different local DBMSs (using middleware)
- Data consistency (via multiphase commit protocols)
- Global primary key control
- Scalability
- Security, concurrency, query optimization, failure recovery

Distributed DBMS architecture



Problem: Network partitions

- Imagine you have a synchronously-updating, replicated database – this is a common setup in industry today
- Now imagine that the link between 2 nodes is interrupted



- What are your choices?
  - shut down the system (to avoid inconsistency)
  - keep it available to users (and accept inconsistency)
- More about this in the NoSQL lecture …

(diagram from https://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html)

Date's 12 Commandments for Distributed Databases

- Local Site Independence
  - Each local site can act as an independent, autonomous, centralised DBMS. Each site is responsible for security, concurrency, control, backup and recovery.
- Central Site Independence
  - No site in the network relies on a central site or any other site. All sites have the same capabilities.
- Failure Independence
  - The system is not affected by node failures. The system is in continuous operation even in the case of a node failure or an expansion of the network.
- Location Transparency
  - The user does not need to know the location of the data to retrieve those data.

Date's 12 Commandments for Distributed Databases

- Fragmentation Transparency
  - Data fragmentation is transparent to the user, who sees only one logical database. The user does not need to know the name of the database fragments to retrieve them.
- Replication Transparency
  - The user sees only one logical database. The DDBMS transparently selects the database fragment(s) to access. To the user, the DDBMS manages all fragments transparently.
- Distributed Query Processing
  - A distributed query may be executed at several different sites. Query optimization is performed transparently by the DDBMS.
- Distributed Transaction Processing
  - A transaction may update data at several different sites, and the transaction is executed transparently.

Date's 12 Commandments for Distributed Databases

- Hardware Independence
  - The system must run on any hardware platform.
- Operating System Independence
  - The system must run on any operating system platform.
- Network Independence
  - The system must run on any network platform.
- Database Independence
  - The system must support any vendor's database product