

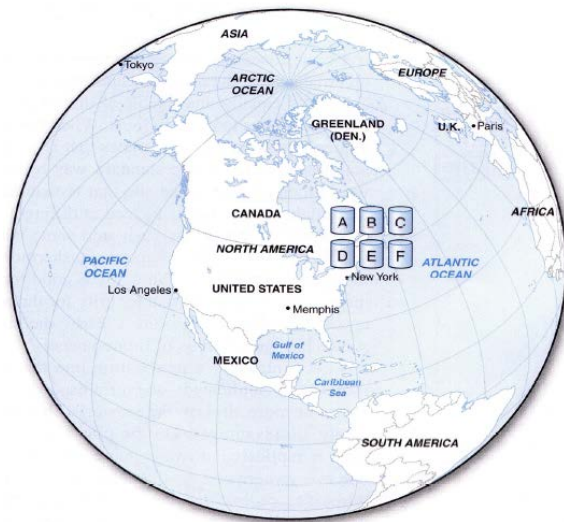
David Eccles



# INFO90002 Database Systems & Information Modelling

Week 08  
Distributed Databases

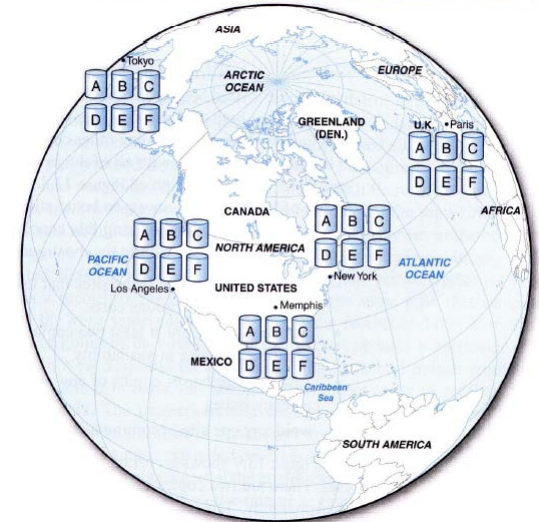
- What is a distributed database?
- Why are they used, and how they work
- Pros and cons of different approaches
  - material in this lecture is drawn from Hoffer et al. (2013) *Modern Database Management* 11<sup>th</sup> edition, chapter 12, available online at [http://wps.prenhall.com/bp\\_hoffer\\_mdm\\_11/230/58943/15089539.cw/index.html](http://wps.prenhall.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

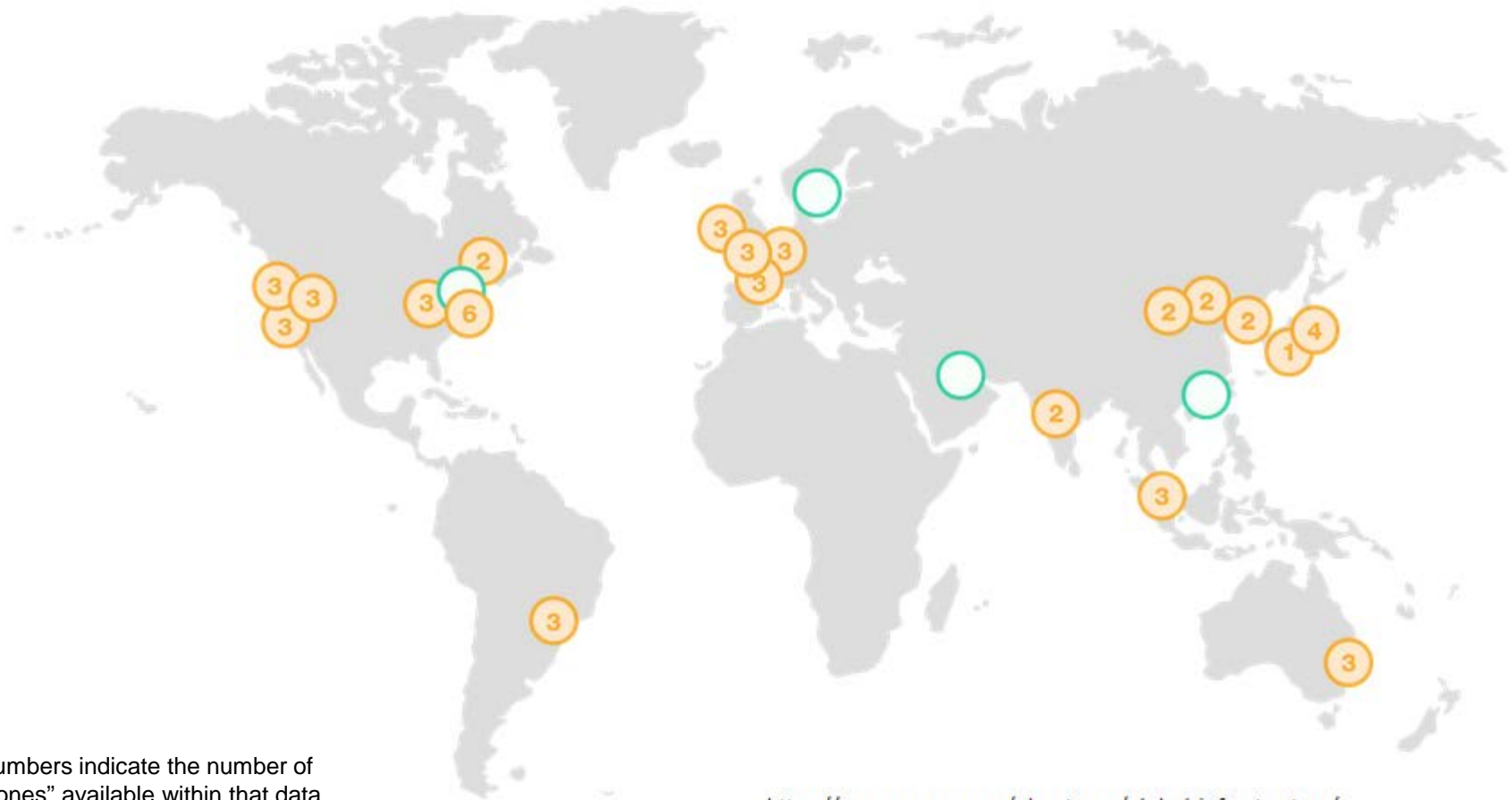


replicated database

- **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 – Amazon AWS

## Global Infrastructure



Numbers indicate the number of “zones” available within that data centre providing redundancy and availability

<https://aws.amazon.com/about-aws/global-infrastructure/>

- Good fit for geographically distributed organizations / users
  - Utilise the internet
- Data located near site with greatest demand
  - ESPN Weekend Sports Scores



AFL - Melbourne



EPL - London



NFL - New York



Hurling - Dublin

- Faster data access (to local data)
- Faster data processing
  - workload is shared between each physical server



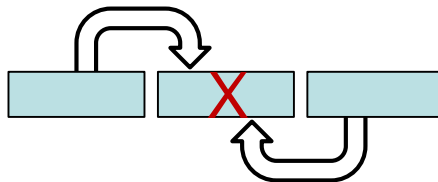
- Allows modular growth
  - add new servers as load increases



- Increased reliability and availability
  - less danger of a single-point of failure (SPOF)



- Supports database recovery
  - data is replicated across multiple sites



- Complexity of management and control
  - db or/and application must stitch together data across sites
    - Who and where is the current version of the record (row & column)?
    - Who is waiting to update that information and where are they?
    - How does the logic display this to the web & application server?
- Data integrity
  - additional exposure to improper updating
    - If two users in two locations update the record at the exact same time who decides which statement should “win”?
  - Solution: Transaction Manager or Master-slave design
- Security
  - many server sites -> higher chance of breach
    - Multiple access sites require protection including network and storage infrastructure from both cyber & physical attacks

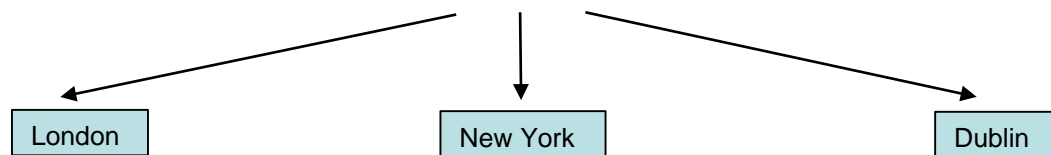
- Lack of standards
  - different Relational DDBMS vendors use different protocols
- Increased training & maintenance costs
  - more complex IT infrastructure
    - Increased Disk storage (\$)
    - Fast intra and inter network infrastructure (\$\$)
    - Clustering software (\$)
    - Network Speed (\$\$)
- Increased storage requirements
  - Replication model



- Location transparency
  - a user does not need to know where particular data are stored
- Local autonomy
  - a node can continue to function for local users if connectivity to the network is lost
- Trade-offs
  - Availability vs Consistency
  - Synchronous vs Asynchronous updates

- A user (or program) accessing data do not need to 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

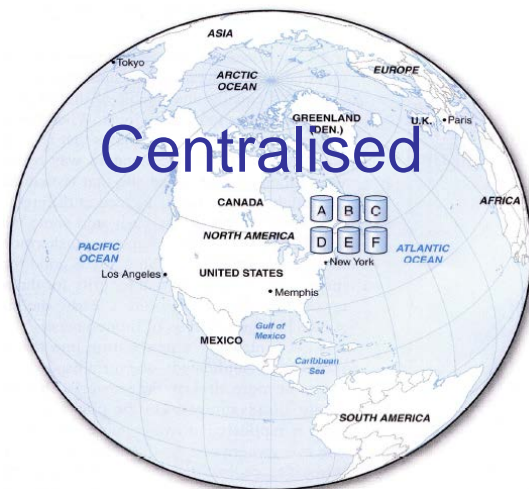
```
SELECT hometeam, homescore, awayteam, awayscore  
FROM results inner join codes  
ON results.codeid = codes.codeid  
WHERE sportscode in ('NFL', 'Hurling', 'EPL');
```





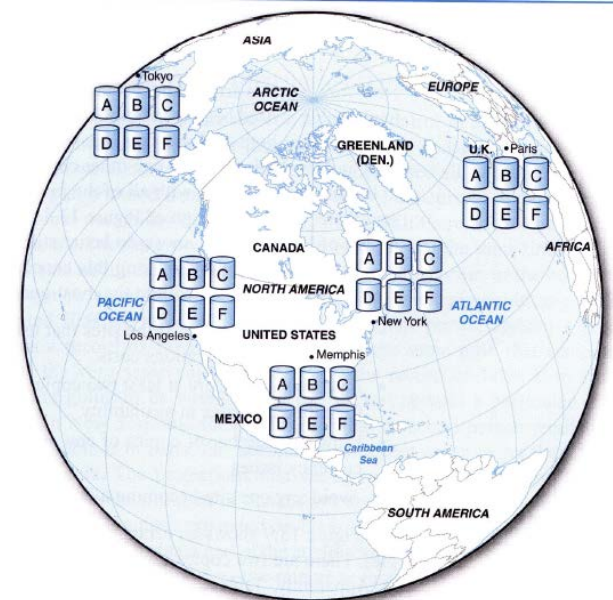
- Users can administer their local database
  - control local data (e.g Hurling results)
  - administer security
  - log transactions
  - recover when local failures occur
  - provide full access to local data
- Being able to operate locally when connections to other databases fail

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

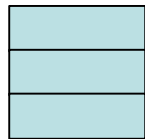


# Replication - advantages

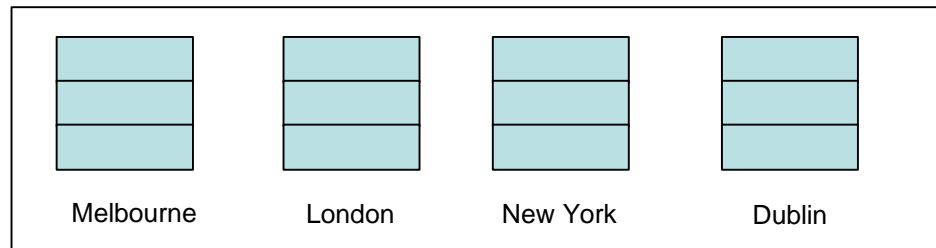
- High reliability due to redundant copies of data
- Fast access to data at the location where it is most accessed
- May avoid complicated distributed integrity routines
  - replicated data is refreshed at scheduled intervals
- Decoupled nodes don't affect data availability
  - transactions proceed even 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 SQL & NoSQL databases offer replication



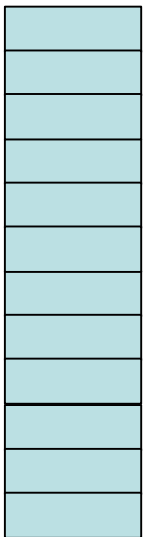
- Need more storage space
  - Each server stores a copy of the row
- Data Integrity:
  - high tolerance for out-of-date data may be required
  - updates may cause performance problems for busy nodes
  - retrieve incorrect data if updates have not arrived



Centralised Database  
One database in one server  
(1 copy of data)



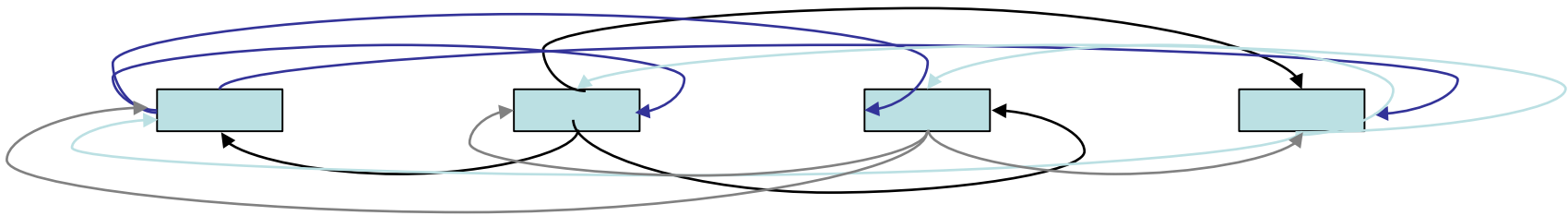
Distributed Database  
One database in 4 physical servers  
(4 copies of data)



Data Size

# Replication - disadvantages

- 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 can place heavy demand on telecommunications/networks

- 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 time checking that an update is accurately and completely propagated across the network.
  - The committed updated record must be identical in all servers



- 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
- 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

- 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 (SPOF)

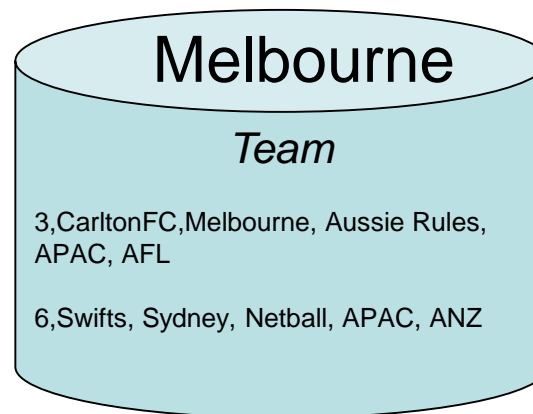
Team table

| ID | Team       | City      | Code         | Region   | League |
|----|------------|-----------|--------------|----------|--------|
| 1  | Arsenal    | London    | Football     | Europe   | EPL    |
| 2  | Jets       | NYC       | Grid Iron    | Americas | NFL    |
| 3  | Carlton FC | Melbourne | Aussie Rules | APAC     | AFL    |
| 4  | Racing92   | Paris     | Rugby        | Europe   | Top14  |
| 5  | Yankees    | NYC       | Baseball     | Americas | MLB    |
| 6  | Swifts     | Sydney    | Netball      | APAC     | ANZ    |

# Example horizontal partitioning

| ID | Team       | City      | Code         | Region   | League |
|----|------------|-----------|--------------|----------|--------|
| 1  | Arsenal    | London    | Football     | Europe   | EPL    |
| 2  | Jets       | NYC       | Grid Iron    | Americas | NFL    |
| 3  | Carlton FC | Melbourne | Aussie Rules | APAC     | AFL    |
| 4  | Racing92   | Paris     | Rugby        | Europe   | Top14  |
| 5  | Yankees    | NYC       | Baseball     | Americas | MLB    |
| 6  | Swifts     | Sydney    | Netball      | APAC     | ANZ    |




## Horizontal Partitioning based on Region



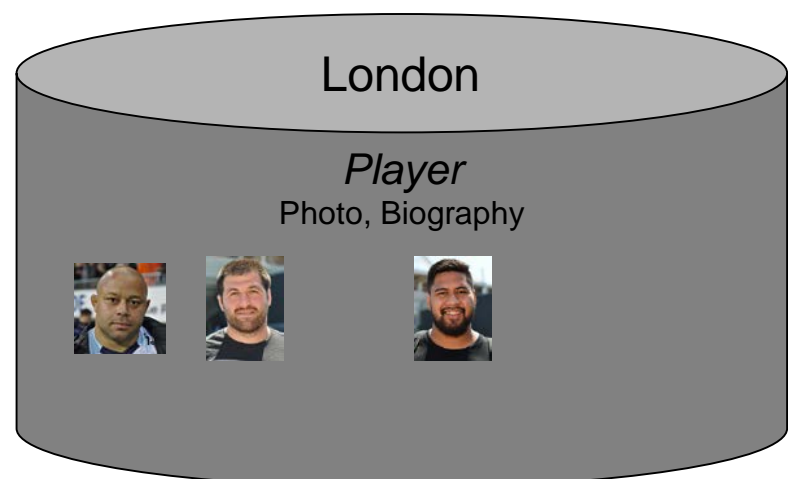
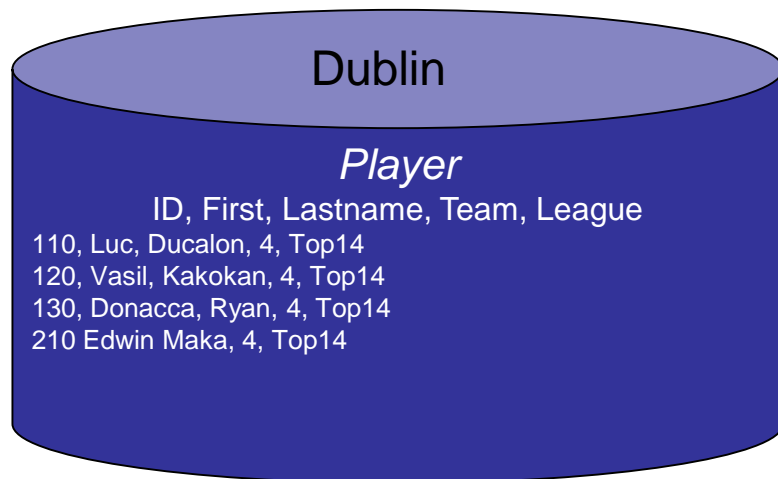
- Different columns of a table at different sites
- Advantages and disadvantages are the same as for horizontal partitioning
  - except
    - combining data across partitions is more difficult because it requires joins (instead of unions)

*Player table*

| ID  | Firstname | Lastname | Team | League | Photo   | Biography         |
|-----|-----------|----------|------|--------|---|-------------------|
| 110 | Luc       | Ducalon  | 4    | Top14  |    | Ipso locum        |
| 120 | Vasil     | Kakokan  | 4    | Top14  |   | Ipso locum<br>est |
| 130 | Donacca   | Ryan     | 4    | Top14  | <null>  |                   |
| 210 | Edwin     | Maka     | 4    | Top14  |  |                   |
|     |           |          |      |        |   |                   |
|     |           |          |      |        |   |                   |

| ID  | Firstname | Lastname | Team | League | Photo   | Biography      |
|-----|-----------|----------|------|--------|---|----------------|
| 110 | Luc       | Ducalon  | 4    | Top14  |  | Ipso locum     |
| 120 | Vasil     | Kakokan  | 4    | Top14  |  | Ipso locum est |
| 130 | Donacca   | Ryan     | 4    | Top14  | <null>  |                |
| 210 | Edwin     | Maka     | 4    | Top14  |  |                |

Vertical Partitioning based on column requirements



- 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

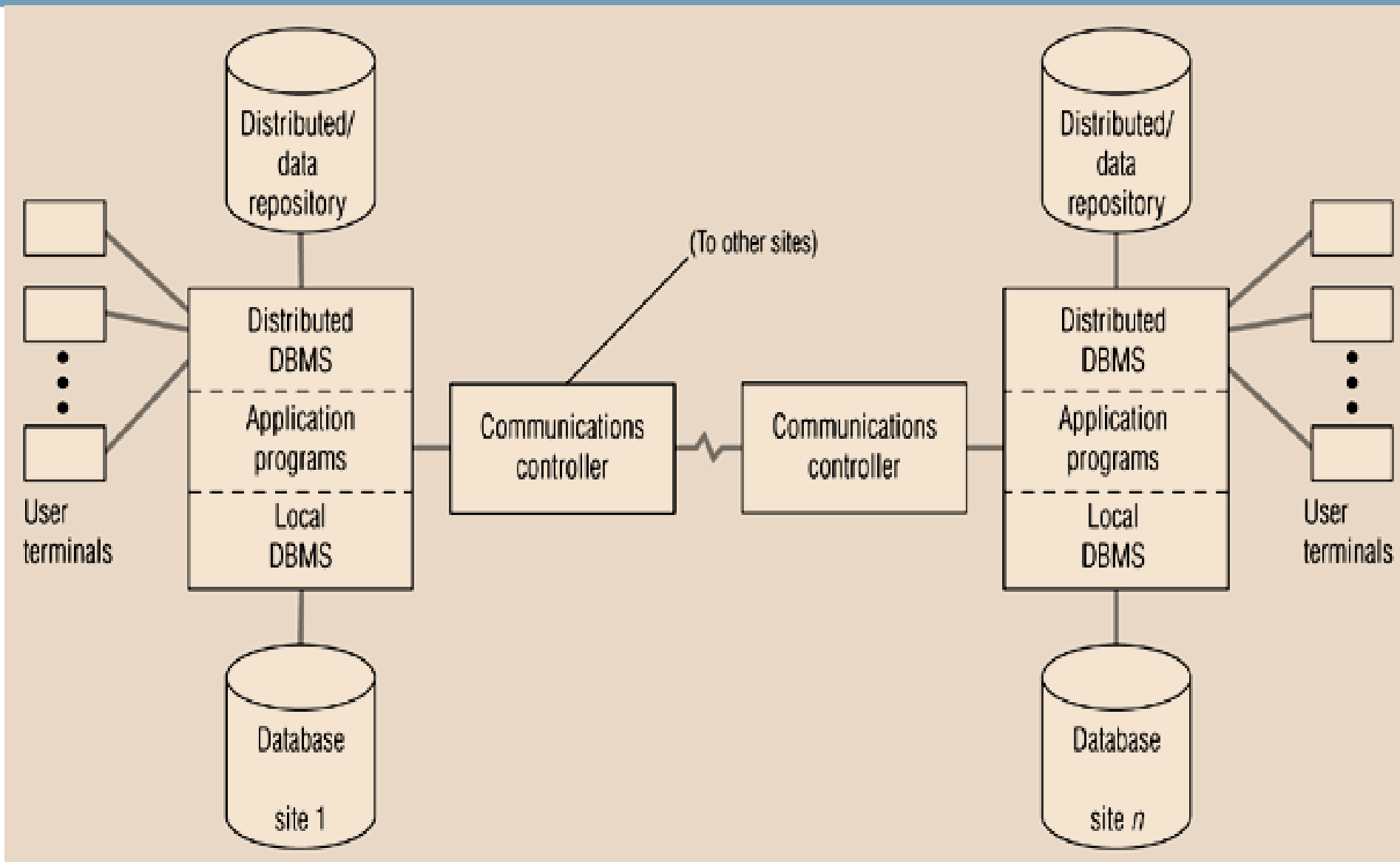
# Comparing Configurations

|   | Reliability  | Expandability   | Communication Overhead  | Management   | Data Consistency  |
|---|--|---|---|--|---|
| <b>Centralised</b>                          | POOR<br>Depends on central server.                     | POOR<br>Single Server is limited by memory & storage maximums.                      | VERY HIGH<br>Traffic heads to one centralised location.   | EXCELLENT<br>One very large site is easier to manage.  | EXCELLENT<br>All users always see the same data.                                    |
| <b>Replicated with Snapshots</b>            | GOOD<br>Redundancy and tolerated delays in data synch. | VERY GOOD<br>Cheap to scale up with new servers.                                    | LOW to MEDIUM<br>Intermittent bursts of network traffic (but not constant flooding of network). | VERY GOOD<br>Each copy is alike.   | MEDIUM<br>Update delays are tolerable with snapshot catch ups for data consistency. |
| <b>Synchronised Replication</b>             | EXCELLENT<br>Redundancy and minimal delays.            | VERY GOOD<br>Low cost and only linear growth in synchronisation.                    | MEDIUM<br>Constant messages to maintain synchronisation.  | MEDIUM<br>Data collusions need to be resolved and need good design and management.                 | VERY GOOD<br>Close to precise consistency.  |
| <b>Integrated Partitions</b>                | GOOD<br>Effective use of partitioning and redundancy.  | VERY GOOD<br>New nodes only get the data they need and no need to change DB design. | LOW to MEDIUM<br>Most queries are local, but global queries to create temporary comms load.     | DIFFICULT<br>Distributed table updates require tight precise coordination.                         | VERY POOR<br>Requires considerable effort and inconsistencies are not tolerated.    |
| <b>Decentralised Independent Partitions</b> | GOOD<br>Depends on local DB availability.              | GOOD<br>New sites are independent of all other sites.                               | LOW<br>Little or no traffic needs to be communicated across the network.                        | VERY GOOD<br>Easy – as each site is independent of the other sites and minimal need to share data. | LOW<br>No guarantee of consistency – therefore high chance of consistency.          |

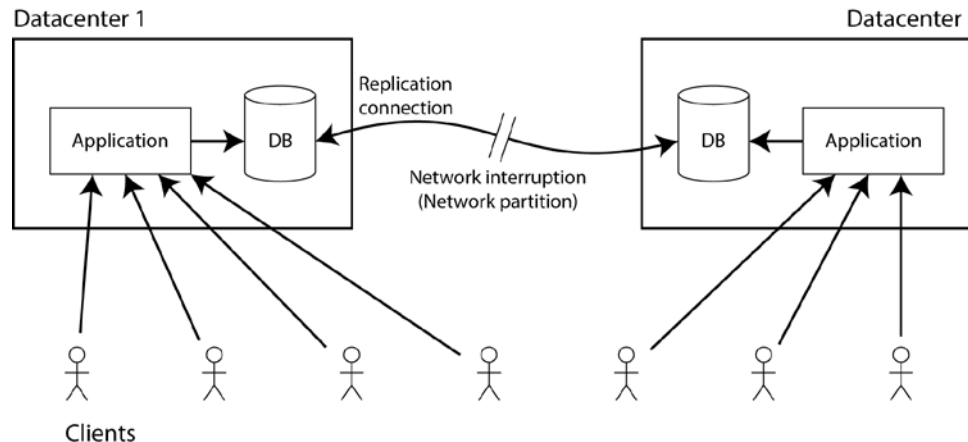
- Locate data with a distributed data dictionary
- Determine location from which to retrieve data and process query components
- DBMS translation between nodes with different local DBMSs (using middleware)
- Data consistency (via multiphase commit protocols)
- Global primary key control
- Scalability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



# Distributed DBMS architecture



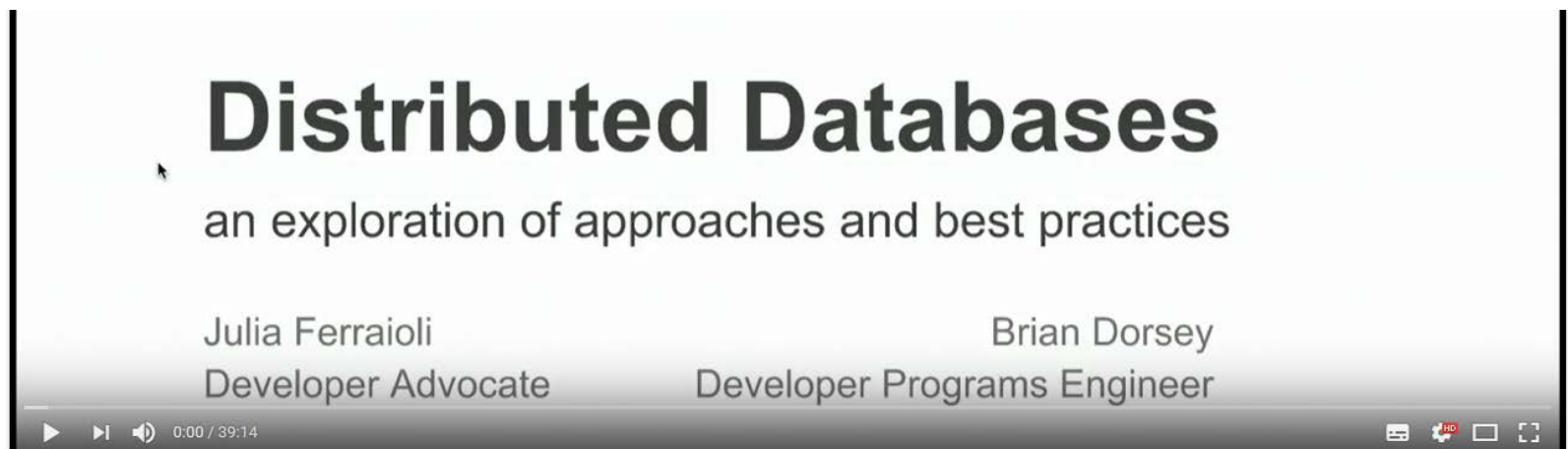
- Imagine you have a synchronously-updating, replicated database –
- Now imagine that the link between 2 nodes is interrupted



- 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>)

- *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>



# 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.

- **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



THE UNIVERSITY OF  
MELBOURNE

# Week 08 Distributed Databases