course:

Database Systems (A7B36DBS)

lecture 12:

Database Architectures and Models

Doc. RNDr. Irena Holubova, Ph.D.

Acknowledgement:
The slides were kindly lent by Doc. RNDr. Tomas Skopal, Ph.D.,
Department of Software Engineering, Charles University in Prague

Today's Lecture Outline

- architectures of database systems
  - centralized systems
  - client – server systems
  - parallel systems
  - distributed systems
- logical database models
  - relational
  - object-relational
  - object
- types of queries
- NoSQL databases

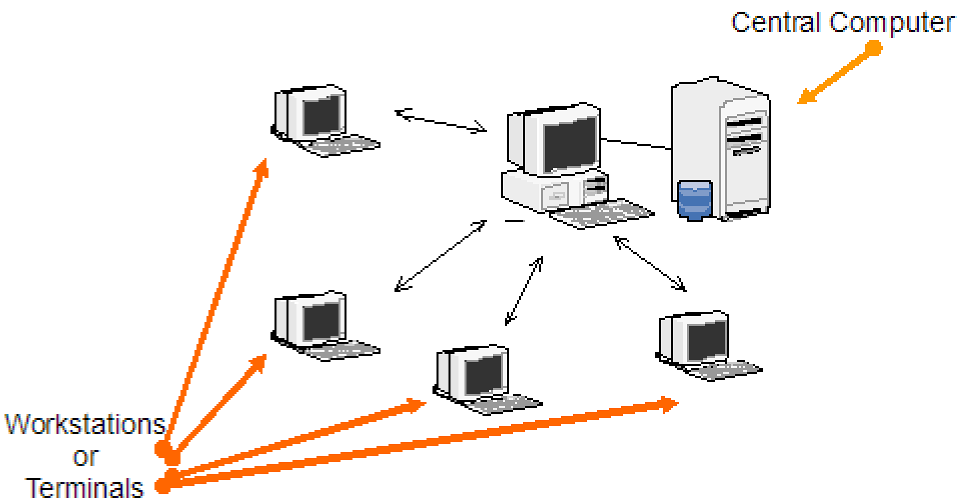
Architectures of Database Systems

- centralized systems
- client – server systems
- parallel systems
- distributed systems

Centralized Systems

- run on a single computer system
- do not interact with other computer systems
- **general-purpose computer system**
  - one to a few CPUs and a number of device controllers
  - connected through a common bus
    - provides access to a shared memory
- **single-user system** (e.g., personal computer or workstation)
  - **desk-top unit**, single user, usually has one or two CPUs and one or two hard disks
  - the OS may support only one user
- **multi-user system**:
  - more disks, more memory, multiple CPUs, and a multi-user OS
  - serve a large number of users who are connected to the system via **terminals**

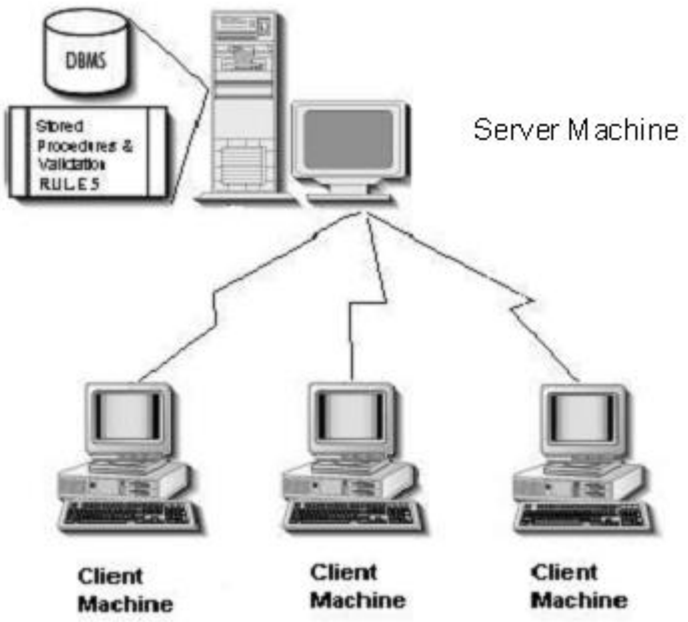
Multi-User Systems



Client-Server Systems

- server systems satisfy requests generated at *m* client systems
- advantages of replacing mainframes with networks of workstations or personal computers connected to back-end server machines:
  - better functionality for the cost
  - flexibility in locating resources and expanding facilities
  - better user interfaces
  - easier maintenance

Client-Server Systems



Front-End vs. Back-End

- database functionality can be divided into:
  - **back-end**: manages access structures, query evaluation and optimization, concurrency control and recovery
  - **front-end**: consists of tools such as forms, report-writers, and graphical user interface facilities
- interface between the front-end and the back-end:
  - SQL
  - application program interface

Parallel Systems

- consist of multiple processors and multiple disks connected by a fast interconnection network
  - a **coarse-grain parallel** machine consists of a small number of powerful processors
  - a massively parallel or **fine-grain parallel** machine utilizes thousands of smaller processors
- two main performance measures:
  - **throughput** – the number of tasks that can be completed in a given time interval
  - **latency** (response time) – the amount of time it takes to complete a single task from the time it is submitted

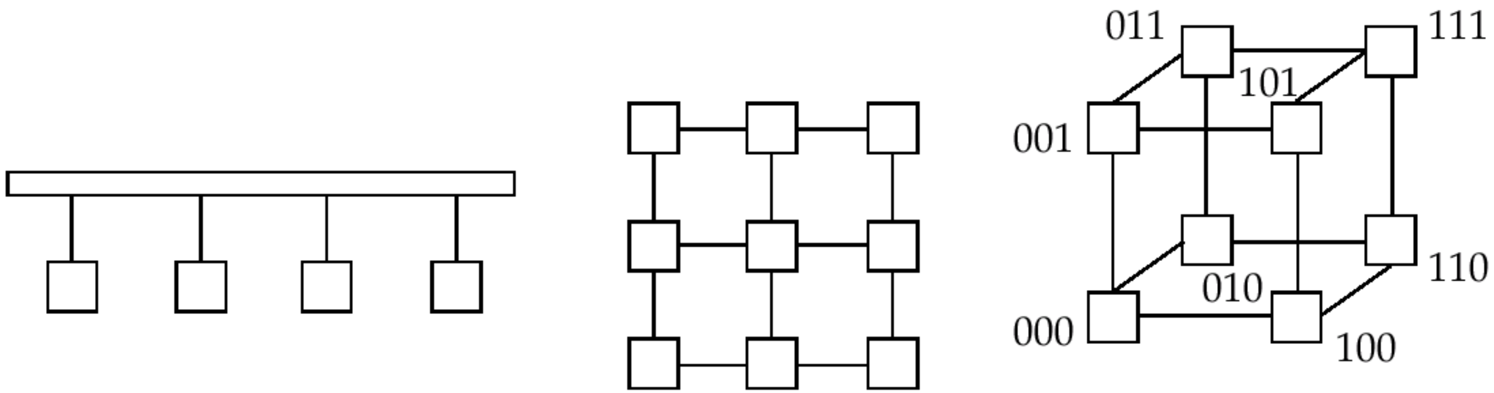
Parallel Systems

- **speed-up**: a fixed-sized problem executing on a small system is given to a system which is N-times larger (more efficient)
- **scale-up**: increase the size of both the problem and the system
  - N-times larger system used to perform N-times larger job
- both often sub-linear due to:
  - Start-up costs: Cost of starting up multiple processes > computation time
    - If the degree of parallelism is high
  - Interference: Processes accessing shared resources (e.g., system bus, disks, or locks) compete with each other $\Rightarrow$ spend time waiting on other processes rather than performing useful work
  - Skew: Increasing the degree of parallelism increases the variance in service times of tasks executed in parallel
    - Overall execution time is determined by the slowest of executing tasks

Interconnection Architectures

- **Bus**: components send data on and receive data from a single communication bus
  - cons: does not scale well with increasing parallelism
- **Mesh**: components are arranged as nodes in a grid, and each component is connected to adjacent components
  - pros: communication links grow with growing number of components
    - scales better
  - cons: may require $2\sqrt{n}$ hops to send message to a node
- **Hypercube**: components are numbered in binary representation ⇒ components are connected to one another if their binary representations differ in exactly one bit.
  - n components are connected to $log(n)$ other components and can reach each other via at most $log(n)$ links
  - reduces communication delays

Interconnection Architectures

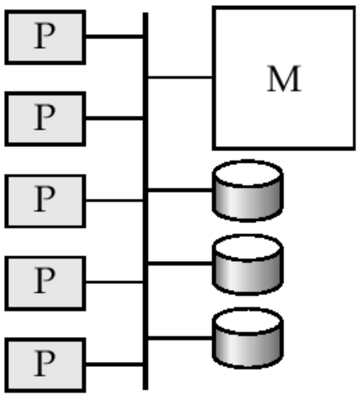


(a) bus (b) mesh (c) hypercube

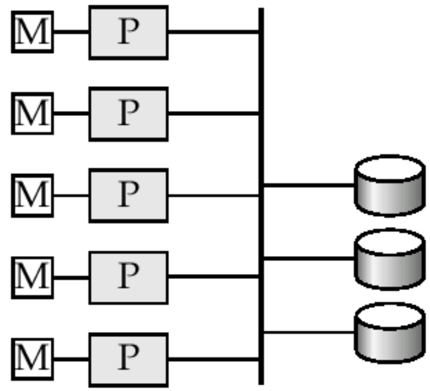
Parallel (Database) Architectures

- **Shared memory** – processors share a common memory
  - efficient communication between processors
  - not scalable much
    - the bus or the interconnection network becomes a bottleneck
- **Shared disk** – processors share a common disk
  - a degree of fault tolerance – if a processor fails, other processors can take over its tasks
    - data are accessible from all processors
  - bottleneck = interconnection to the disk
- **Shared nothing** – processors share neither a common memory nor common disk
  - processors communicate using an interconnection network
  - drawback: cost of communication and non-local disk access
- **Hierarchical** – combination of the above architectures
  - top level is a shared-nothing
  - each node of the system could be a shared-memory sub-system

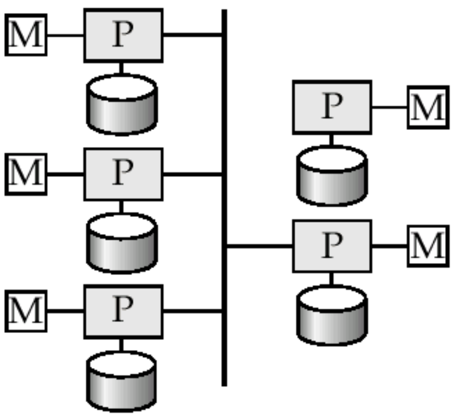
Parallel Database Architectures



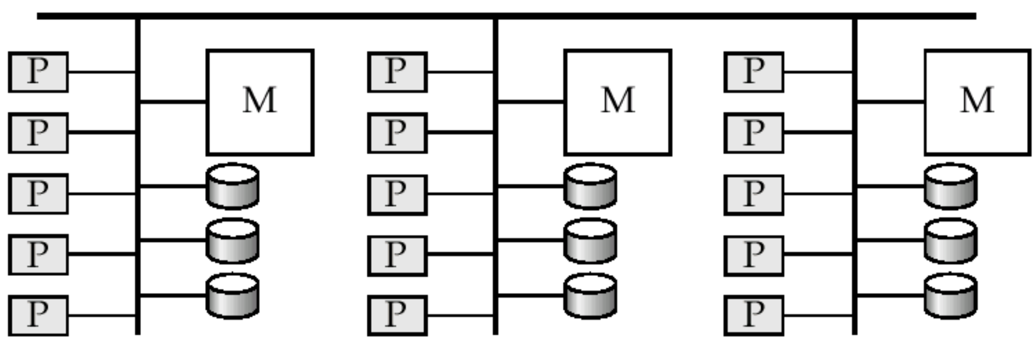
(a) shared memory



(b) shared disk



(c) shared nothing



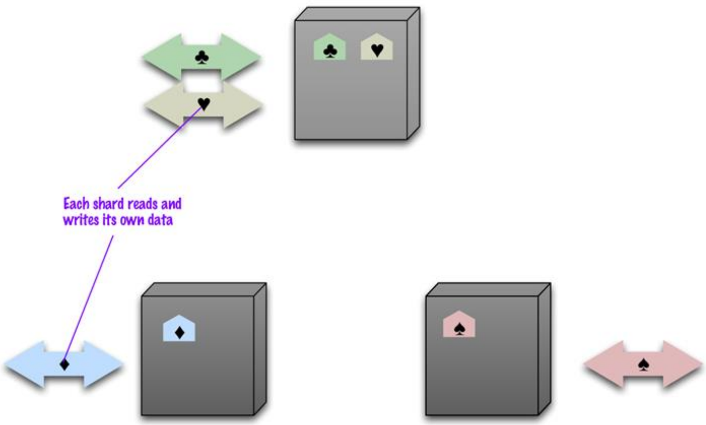
(d) hierarchical

Distributed Systems

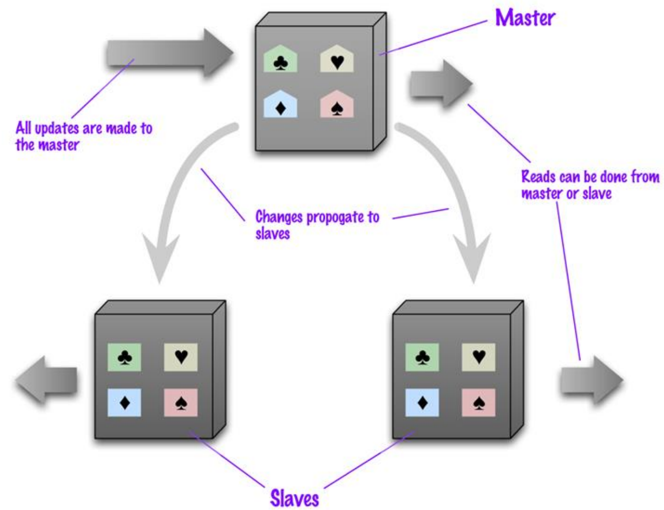
- **scale-out**:data are **distributed** (spread) over multiple machines = nodes
- data are **replicated**
  - system can work even if a node fails
- **homogeneous** distributed databases
  - same software/schema on all nodes, data may be partitioned among nodes
  - goal: provide a view of a single database, hiding details of distribution
- **heterogeneous** distributed databases
  - different software/schema on different nodes
  - goal: integrate existing databases to provide useful functionality

Distribution Models

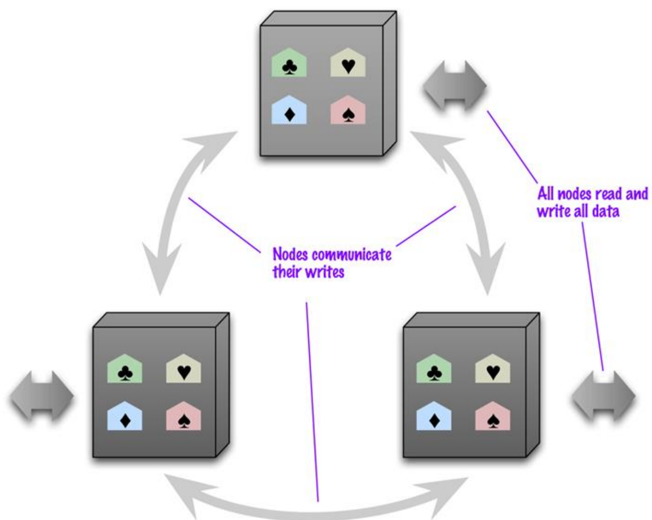
- **single server** – no distribution
- **sharding** – putting different parts of the data onto different servers
  - too many data to be stored on a single node
- **master/slave replication** – master provides reads/writes, slaves provide reads
  - no scalability of writes
- **peer-to-peer replication** – all replicas have equivalent weight
  - each node is a master
- often: **combination** of sharding and replication



sharding = **distribution**



master/slave **replication**



peer-to-peer **replication**

Logical Database Models

- current common models:
  - **relational databases**
  - **object databases**
  - **object-relational databases**
- old, outdated database models:
  - still used on mainframes
  - **hierarchical**
    - tree data structure
    - a record can have one ancestor and multiple descendants
  - **network databases**
    - allows also multiple ancestors for a record (tree $\Rightarrow$ graph)
  - currently replaced by XML databases (trees) or (in general) object databases (general graphs)

Object Databases (ODBMS)

- motivation: success of object-oriented programming (OOP)
- data modelled by classes
  - instances = objects
- advantages similar to OOP:
  - **encapsulation**
  - conceptual model is merged with logical model
  - direct associations among objects (**pointers**)
    - native modelling of graphs
  - the model can be **directly used by OOP**
- disadvantages:
  - persistency of objects and related operations are **non-trivial to implement**
    - complexity incomparable to relational databases
  - suitable for navigational queries but **not for declarative queries** (i.e., SQL-like)

Object-Relational Databases (ORDBMS)

- idea: a relational database extended with object-oriented features
- typically:
  - relation (table) is a basis as in RDBMS
  - object types are allowed
    - object tables
    - attributes as object

    $\Rightarrow$ tables are **not in first normal form**
    - nested classes
- since SQL:1999 it is a standard
- currently **the most popular compromise**
  - advantages of both approaches
  - e.g., MS SQL Server, Oracle DB, IBM DB2, …

Types of Queries

- **declarative**
  - we describe the **data we want**, but not how to get it
  - e.g., DRC, TRC
- **procedural**
  - we describe **how to get the data** we want
    - i.e., what operations should be done
  - e.g., relational algebra (partially)
- SQL has both the features

- QBE (Query by Example)
  - graphical query language from mid 70-ies (IBM)
    - developed as an alternative to SQL
  - many graphical front-ends for databases re-use the idea today

QBE

Sailors (<u>*sid:* integer</u>, *sname:* string, *rating:* integer, *age:* real)
Boats (<u>*bid:* integer</u>, *bname:* string, *color:* string)
Reserves (<u>*sid:* integer, *bid:* integer, *day:* dates</u>)

Sailors	*sid*	*sname*	*rating*	*age*
P.			10	

Sailors with rating 10

Sailors	*sid*	*sname*	*rating*	*age*
		P._N		P._A

Names and ages of all sailors

Sailors	*sid*	*sname*	*rating*	*age*
	_Id	P._S		> 25

Reserves	*sid*	*bid*	*day*
	_Id		'8/24/96'

Sailors who have reserved a boat for 8/24/96 and who are older than 25

Sailors	*sid*	*sname*	*rating*	*age*
	_Id			> 25

Reserves	*sid*	*bid*	*day*
	_Id	_B	'8/24/96'

Boats	*bid*	*bname*	*color*
	_B	Interlake	P.

Colors of boats Interlake reserved by sailors who have reserved a boat for 8/24/96 and who are older than 25

NoSQL Databases

- since 2009 (approx.)
- NoSQL movement: “the whole point of seeking alternatives is that you need to solve a problem that relational databases are a bad fit for”
- not „no to SQL“, not „not only SQL“
  - Oracle or Postgres would fit the definition
- „Next generation databases mostly addressing some of the points: being **non-relational**, **distributed**, **open-source** and horizontally **scalable**. The original intention has been modern web-scale databases. Often more characteristics apply as: **schema-free**, easy **replication support**, simple API, **eventually consistent** (not ACID), a **huge data amount**, and more“

http://nosql-database.org/

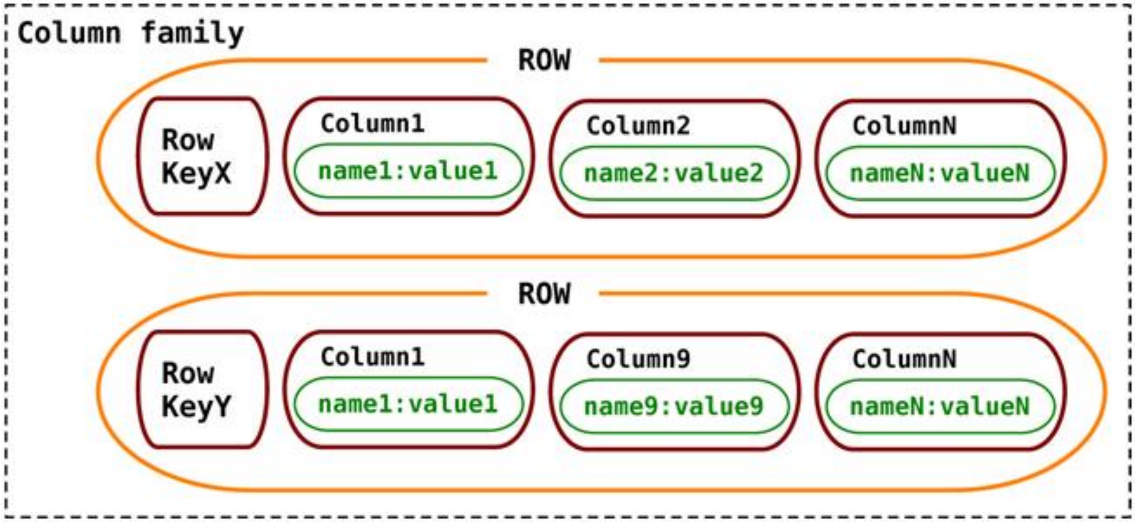
Types of NoSQL Databases

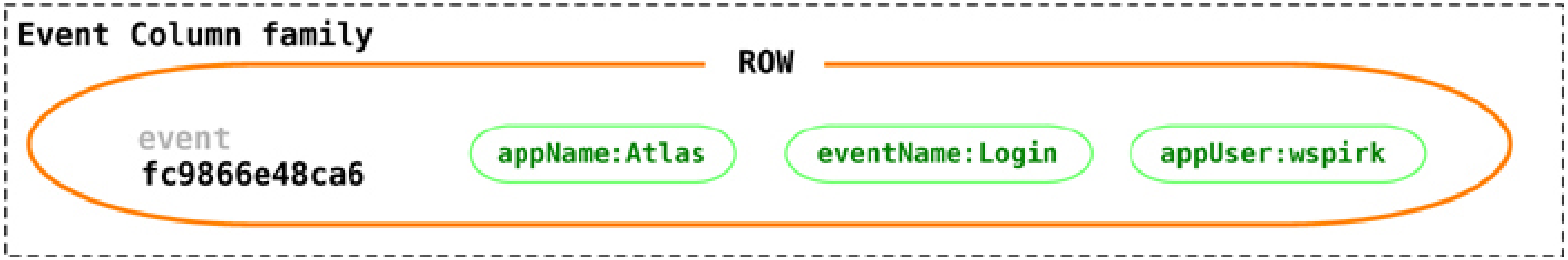
- **Key-value databases**
  - a table with two columns, such as **ID and NAME**
    - ID column being the key
    - NAME column storing the value = a blob that the data store just stores
  - basic operations: get the value for the key, put a value for a key, delete a key from the data store
- **Document databases**
  - document databases store documents in the value part of the key-value store
    - e.g., JSON, XML, …
  - key-value stores where the **value is examinable**
    - hierarchical tree data structures
    - can consist of maps, collections, scalar values, nested documents, …

Types of NoSQL Databases

- **Column-family (column-oriented/columnar) stores**
  - column families = **rows that have many columns** associated with a row key
    - groups of related data that is often accessed together
    - rows do not have to have the same columns
- **Graph databases**
  - to store **entities and relationships** between these entities
    - node = an instance of an object
      - nodes have properties (e.g., name)
    - edges have directional significance
      - edges have types (e.g., likes, friend, …)
  - allow to find interesting patterns
    - e.g., "get all nodes employed by Big Co that like NoSQL Distilled"

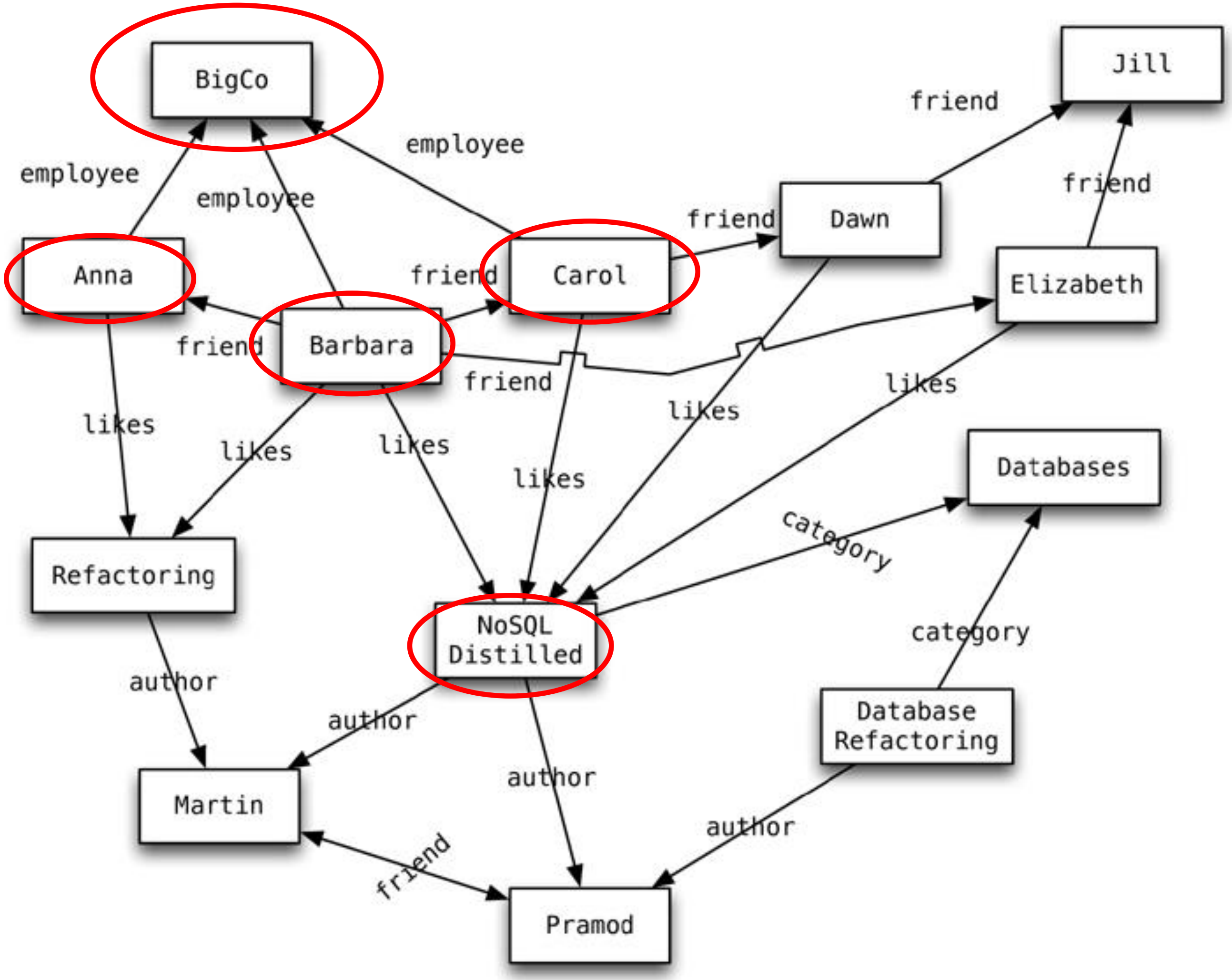
Column Family Examples





typical use case: logging of events in a system (their parameters are similar but not same)

Graph Database Example



“Get all nodes employed by Big Co that like NoSQL Distilled”

NoSQL Databases – the End of Relational Databases?

- relational databases are <u>not</u> going away
- still have compelling arguments for most projects
  - familiarity, stability, feature set, and available support
- we should see relational databases as one option for data storage
  - **polyglot persistence** – using different data stores in different circumstances
- problems NoSQL databases solve:
  - huge amounts of data are now handled in **real-time**
  - both data and use cases are getting more and more **dynamic**
  - social networks (relying on **graph data**) have gained impressive momentum
  - ...

Example: FaceBook

Statistics from 2010

- 500 million users
- 570 billion page views per month
- 3 billion photos uploaded per month
- 1.2 million photos served per second
- 25 billion pieces of content (updates, comments) shared every month
- 50 million server-side operations per second
- 2008: 10,000 servers; 2009: 30,000, …

=> One RDBMS may not be enough to keep this going on!

http://royal.pingdom.com/2010/06/18/the-software-behind-facebook/