Before/Beyond the Relational Model

COMP3211 Advanced Databases

Dr Nicholas Gibbins

2020-2021

The Road Less Travelled

The Road Less Travelled

Lectures so far have concentrated on relational databases

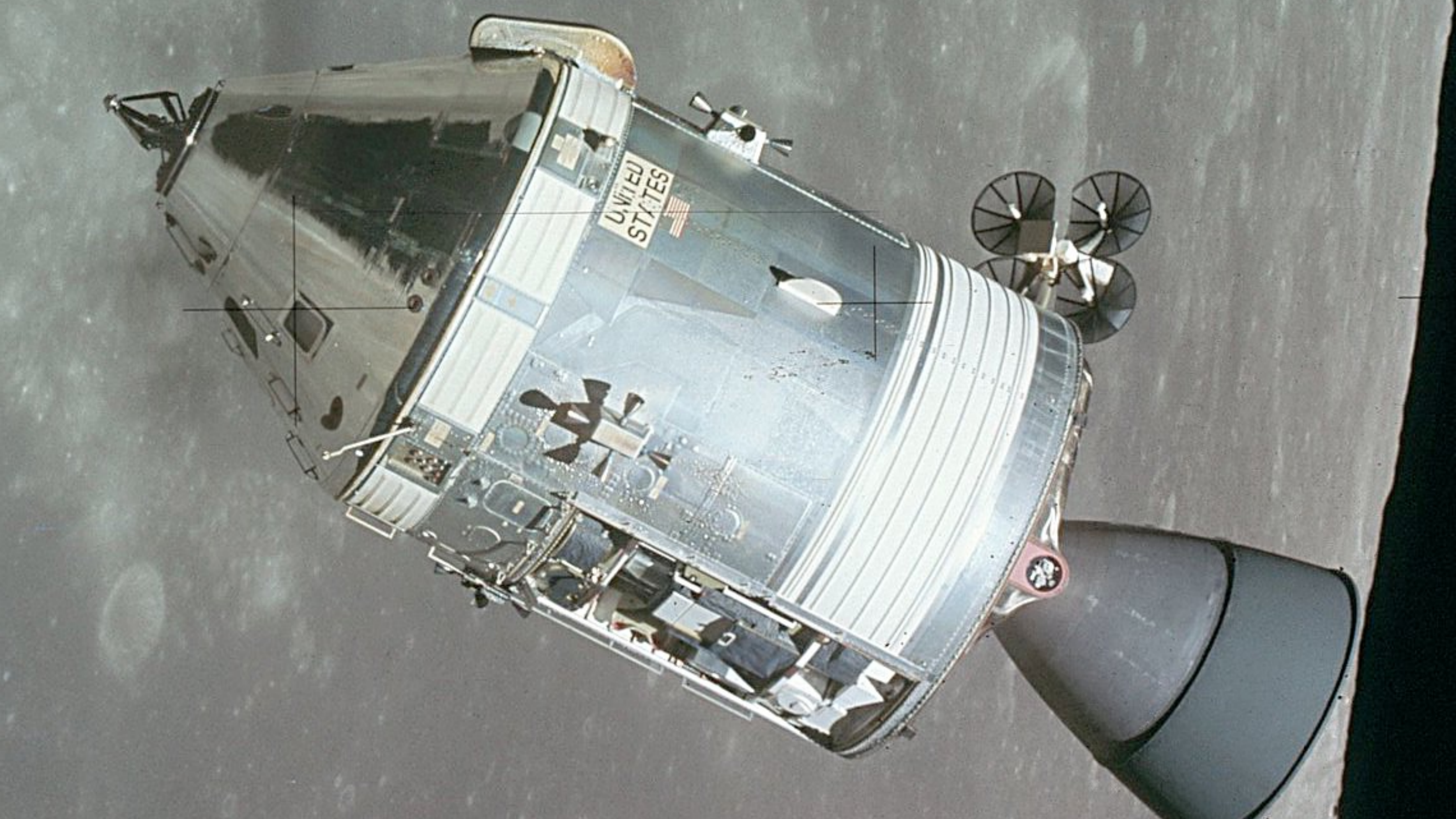
- Proposed by Ted Codd in 1969
- Developed by IBM for System R in the early 1970s
- blahblah SQL blahblah Ingres blahblah Oracle blahblah etc

What came before relational databases?

What can we learn from those systems?

What are the modern equivalents to those systems?

Hierarchical Databases


UNITED
STATES

IBM Information Management System

Development started in 1966 to support the Apollo programme

- Originally IBM Information Control System and Data Language/Interface (ICS/DL/I)
- Used to track the bill of materials for the Saturn V and the CSM

Best known example of a hierarchical database

- Still in use!
- Fast on common tasks that change infrequently – complements DB2 (IBM’s relational database)

Hierarchical Databases

A hierarchy is a natural way to model many real world systems

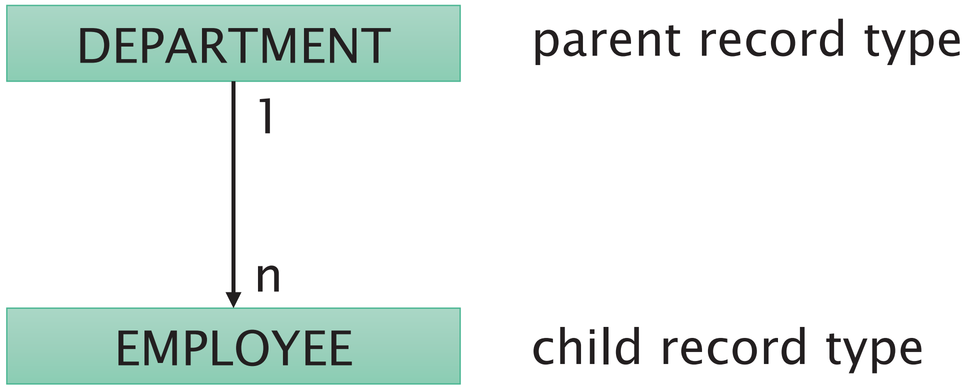
- Taxonomy (“is a kind of”)
- Meronymy (“is a part of”)

Many real-world examples

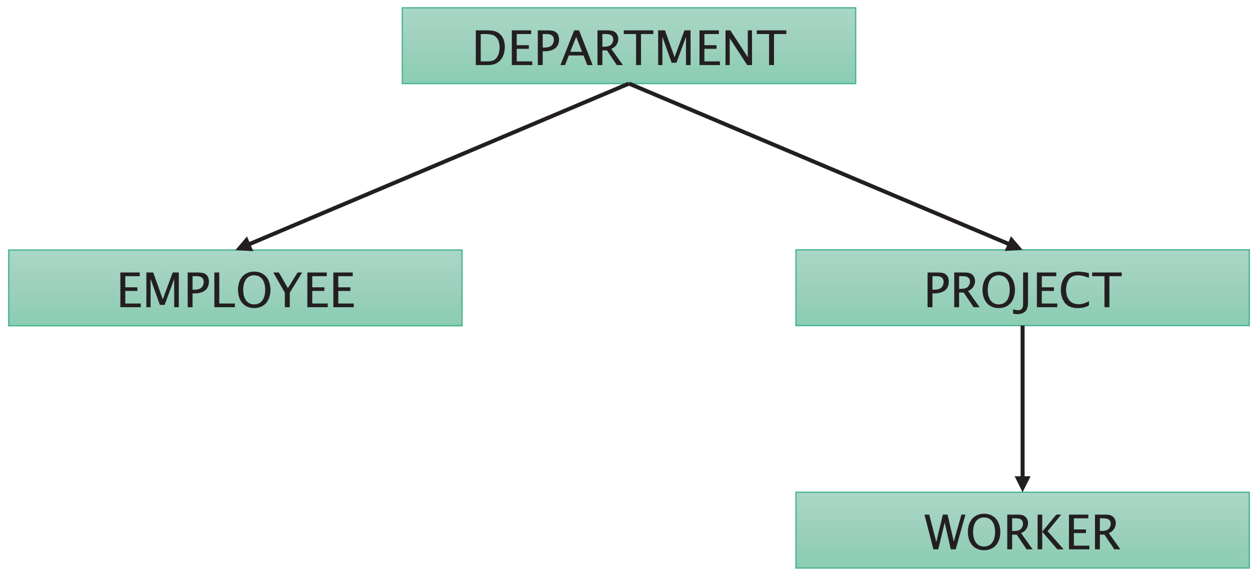
- Organisation charts
- Library classification systems
- Biological taxonomies
- Components of manufactures

Hierarchical DBs are built as trees of related record types connected by parent-child relationships

Parent-Child Relationship Types



Hierarchical Schemas



Occurrences

An *occurrence* or *instance* of the PCR type consists of:

- One record of the parent record type
- Zero or more records of the child record type
- (i.e. an instance is a record and all its children)

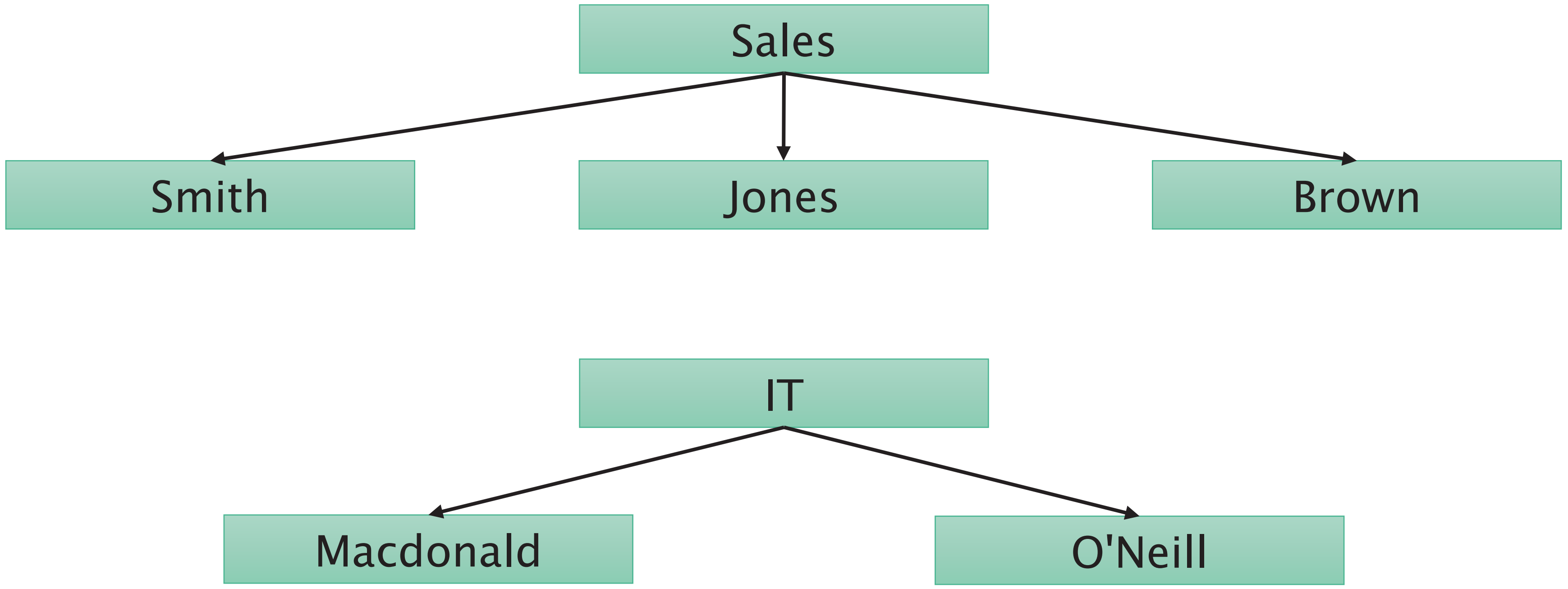
PCR types are referred to by naming the parent record type and child record type

- e.g. (Department, Employee)

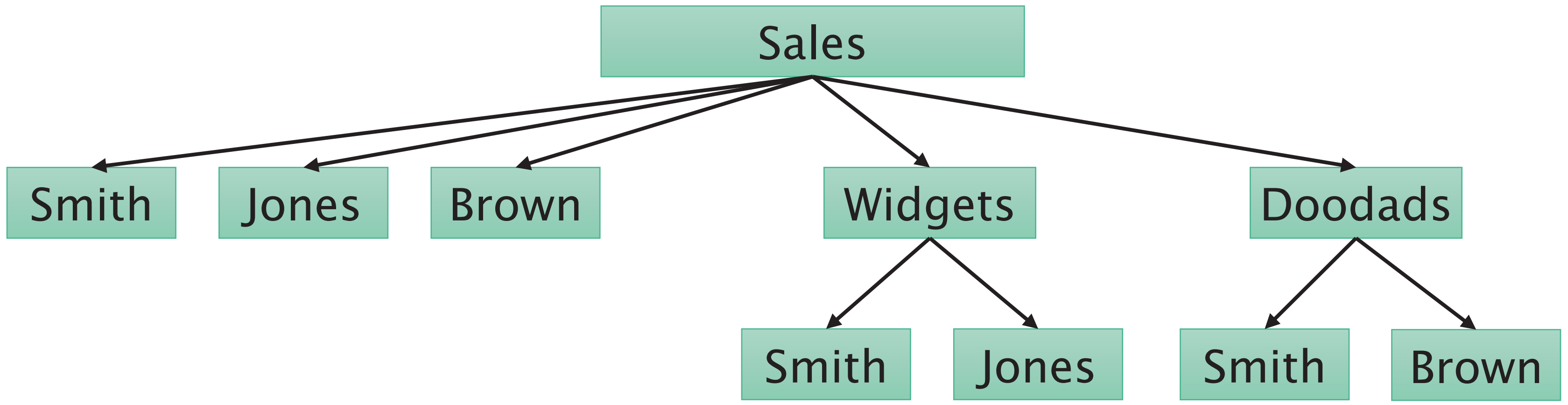
A database may contain many hierarchical occurrences (*occurrence trees*)

- Each occurrence tree is a tree structure whose root is a single record from the root record type of the schema
- The occurrence tree contains all the children (and further descendants) of the root record, all the way to records of the leaf record types

Example Occurrences
(DEPARTMENT, EMPLOYEE)



Example Occurrence Tree



Issues

- Multiple parents (M:N relationships) are not supported – strict hierarchy
  - Can’t represent an employee that works in more than one department
- Record type can't be involved in more than one PCR as child
  - Can't have both (DEPARTMENT, EMPLOYEE) and (PROJECT, EMPLOYEE)
- N-ary relationships (between more than two record types) are not supported
- Querying/update requires the programmer to explicitly navigate the hierarchy – poor data independence

Network Databases

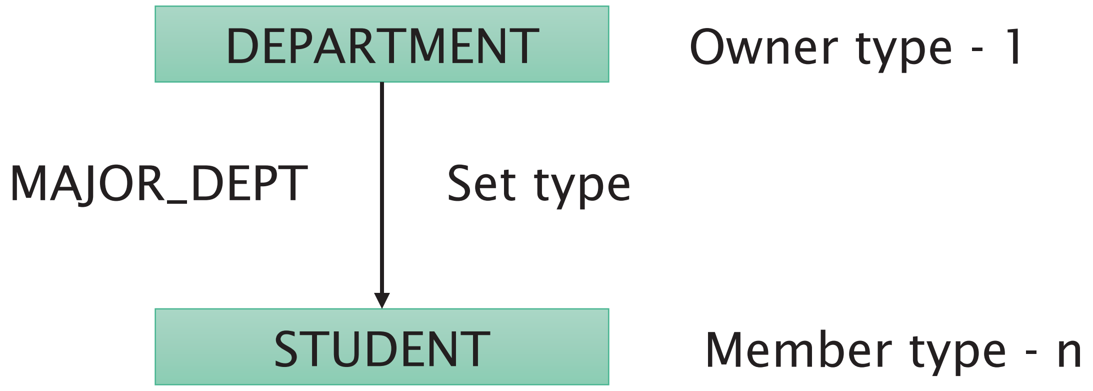
Network Databases

- Standardised by the Conference on Data Systems Languages (CODASYL) committee in 1969
  - (as also was COBOL)
- Addresses limitations of the hierarchical model
- Entities may be related to any number of other entities – no longer limited to a tree
- CA IDMS possibly the best-known example
  - Again, many instances still running worldwide

Using Network Databases

- Record types linked in 1:N relationships
- There are no constraints on the number and direction of links between record types
- No need for a root record type

Set Types


DEPARTMENT
Owner type - 1
MAJOR_DEPT
Set type
STUDENT
Member type - n

Set Occurrences

Set occurrences (set instances) are composed of:

- One owner record from the owner record type
- Zero or more related member records from the member record type

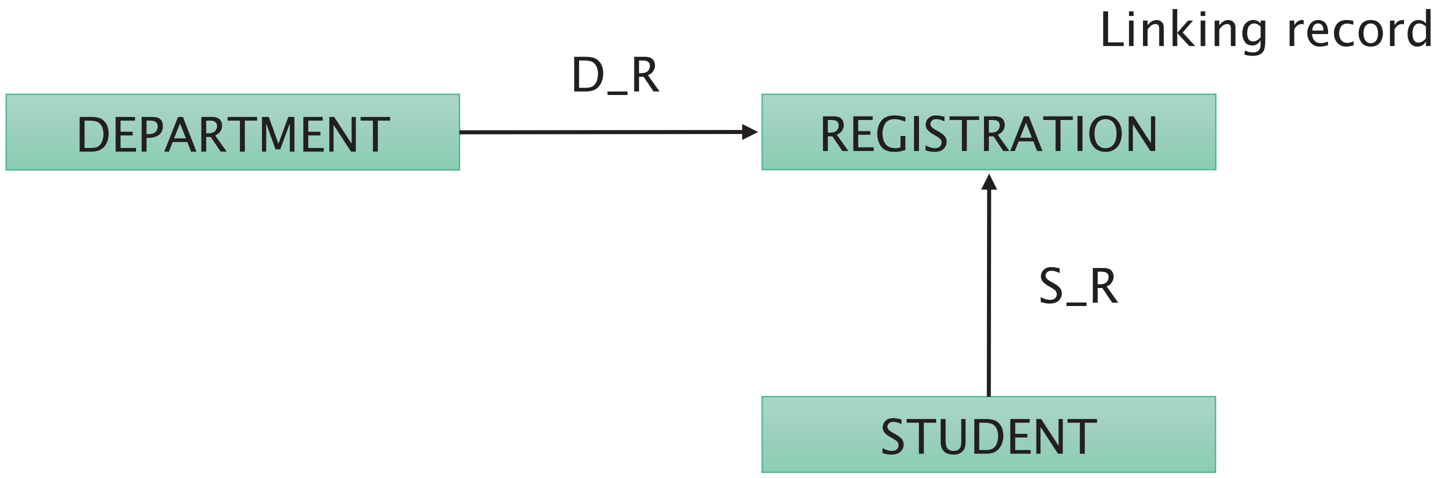
A record from the member record type cannot exist in more than one occurrence of a particular set type

- Maintains 1:N constraint on set types

Representing M:N Relationships

Set types can only represent 1:N relationships, yet many real-world relationships are M:N

- Use a linking or dummy record to join two record types in an M:N relationship



Issues

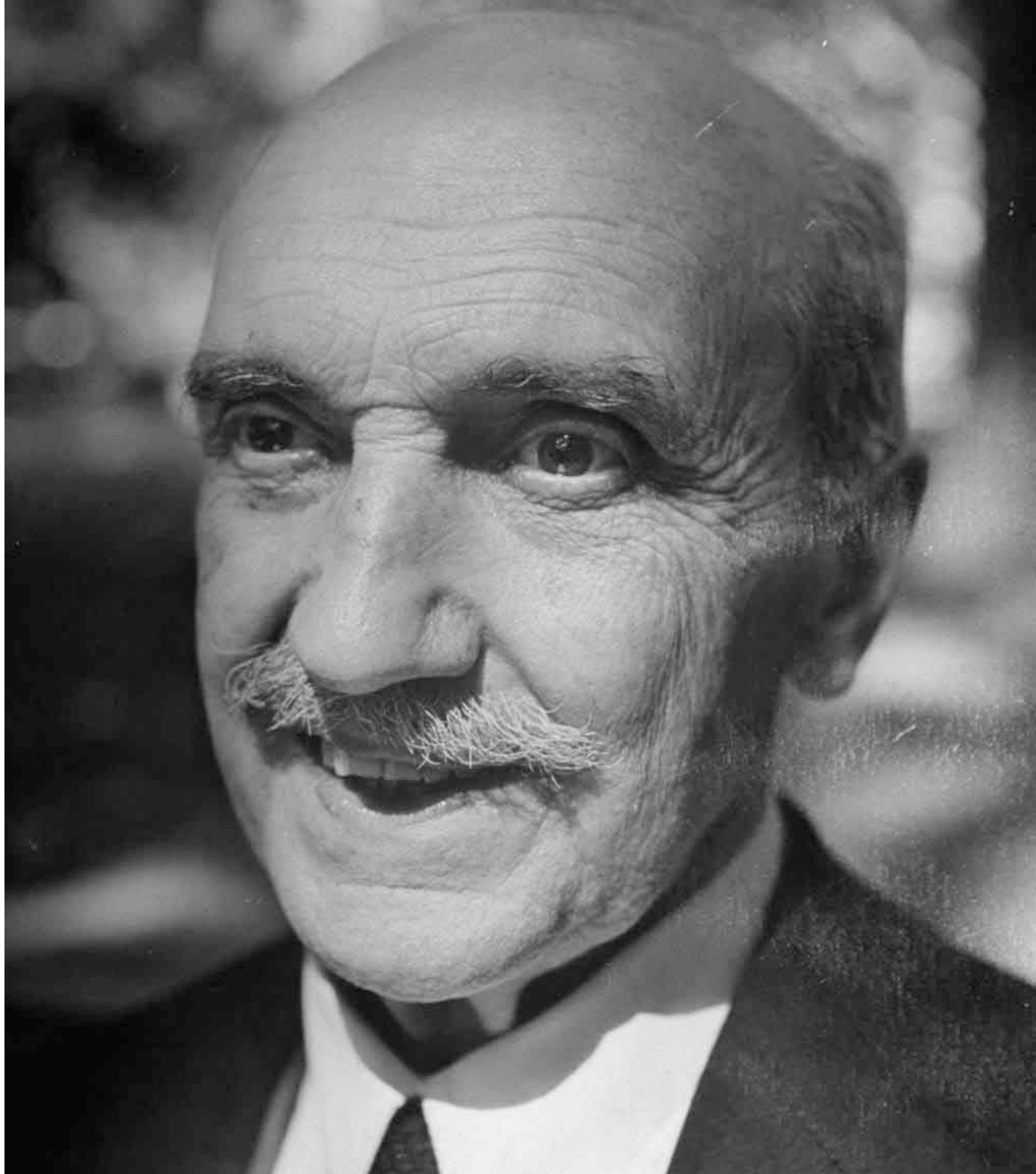
- Easier to model systems with networks than with hierarchies
- Can deal with M:N or N-ary relationships

But

- Querying/update still requires the programmer to explicitly navigate the hierarchy – poor data independence

Why should I care?

Those who cannot remember the past are condemned to repeat it.



George Santayana (1863-1952)

Native XML Databases

- Conceptual descendent of hierarchical DBs
- Define a logical model for an XML document
- Store and retrieve documents according to that model
    - Elements and attributes
    - Plain text content (PCDATA)
    - Ordering of elements (document order)
- Common models
    - XPath data model
    - XML Infoset
    - XML Document Object Model (DOM)

Example XML Database

```
<company>
  <department dname="Sales" mgrname="Smith, J">
    <employee name="Smith, J" birthdate="1969-05-23"/>
    <employee name="Jones, P" birthdate="1961-02-22"/>
    <employee name="Brown, M" birthdate="1973-06-14"/>
    <project pname="Widgets" status="current"
             location="Manchester">
      <worker name="Smith, J" hours="20"/>
      <worker name="Jones, P" hours="40"/>
    </project>
    <project pname="Doodads" status="expired"
             location="London">
      <worker name="Smith, J" hours="20"/>
      <worker name="Brown, M" hours="40"/>
    </project>
  </department>
</company>
```

XQuery example

```
<dl>
{
  for $w in
    document("company.xml")//project[@status="current"]/worker
  where $w/@hours>20
  return <dt>$w/@name</dt><dd>$w/@hours</dd>
}
</dl>
```

XQuery example

```
<dl>
{
  for $w in
    document("company.xml")//project[@status="current"]/worker
  where $w/@hours>20
  return <dt>$w/@name</dt><dd>$w/@hours</dd>
}
</dl>
```

```
<dl>
  <dt>Smith, J</dt><dd>20</dd>
  <dt>Jones, P</dt><dd>40</dd>
</dl>
```

Beyond the Relational Model

So you have some data...

Relational Databases solve most data problems:

- **Persistence**
  - We can store data, and it will remain stored!
- **Integration**
  - We can integrate lots of different apps through a central DB
- **SQL**
  - Standard(ish), well understood, very expressive
- **Transactions**
  - ACID transactions, strong consistency

A few key trends and issues are motivating change in how we store data:

- The impedance mismatch problem
- Increasing volume of data and traffic

The impedance mismatch problem

We typically structure data in memory in an object-oriented fashion

- based on **software engineering** principles of abstraction, encapsulation and inheritance

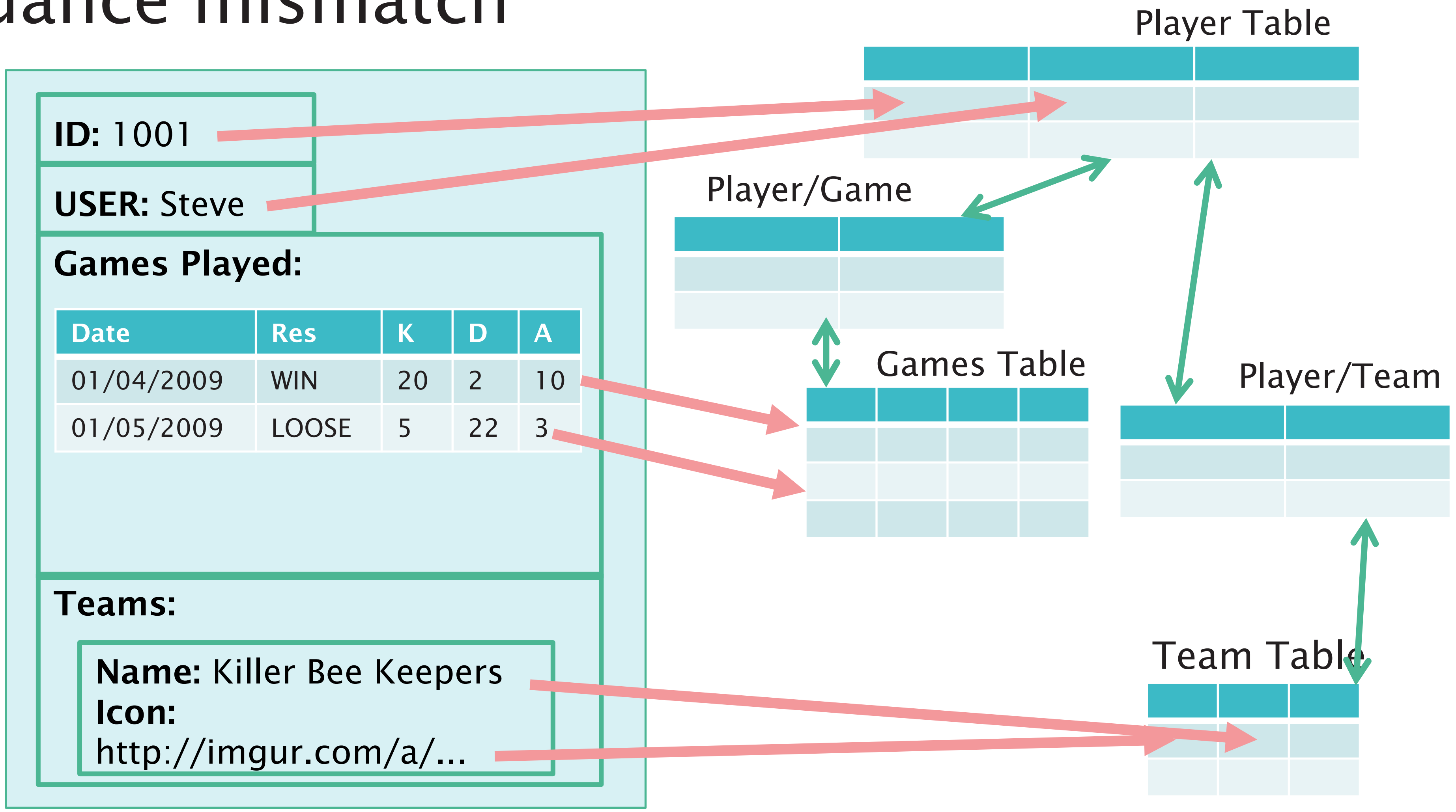
We typically use RDBMSes for persistent storage on disc

- relational model based on **set theory**

These are fundamentally different approaches to structuring data

Mapping from one world to the other has problems

Impedance mismatch



Date	Res	K	D	A
01/04/2009	WIN	20	2	10
01/05/2009	LOOSE	5	22	3

Increased data volume

We are creating/storing/processing more data than ever before

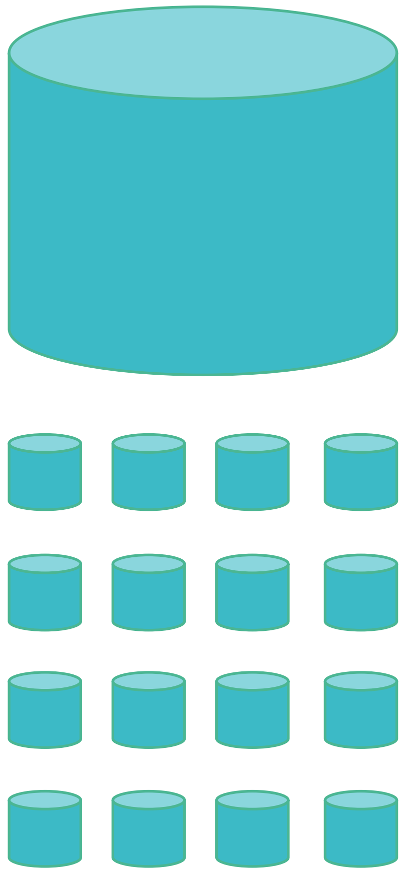
“From 2005 to 2020, the digital universe will grow by a factor of 300, from 130 exabytes to 40,000 exabytes, or 40 trillion gigabytes (more than 5,200 gigabytes for every man, woman, and child in 2020). From now until 2020, the digital universe will about double every two years.”

http://www.emc.com/leadership/digital-universe/index.htm

Dealing with increased data volume

Two options when dealing with these trends:

1. Build bigger database machines
   - This can be expensive
   - Fundamental limits to machine size
2. Build clusters of smaller machines
   - Lots of small machines (commodity machines)
   - Each machine is cheap, potentially unreliable
   - Needs a DBMS which understands clusters



Relational Databases suck…

There is a common perception that RDBMSes have **fundamental issues:**

In dealing with (horizontal) scale

- Designed to work on **single, large machines**
- Difficult to **distribute** effectively

In dealing with the impedance mismatch

- We create data structures in memory and then rip them apart to stick them in an RDBMS
- Relational data models often seem "unnatural" (normalisation seems unintuitive)
- Uncomfortable to program with (joins and ORM etc.)

UNIVERSITY OF Southampton

The NoSQL Movement

NoSQL – A movement

NoSQL came to address

- “**web-scale** problems”
- … **impedance mismatch** on the way

Key attributes include:

- **Non-Relational** (though they can be, but aren’t good at it)
- **Schema-Free** (except the implicit schema, application side)
- Inherently **Distributed** (in different ways, some moreso than others)
- **Open Source** (mostly… e.g. Oracle’s NoSQL)

Defining NoSQL

Quite hard to define a movement based around a **negative**

Is a CSV file NoSQL?

(How about a turnip?)

How about a non-relational database from the 60s/70s/80s/90s

(IMS, IDMS, MUMPS, CLOB, XMLDB etc.)

NoSQL is not easy to define

...but many folks have certainly tried!

Some NoSQL Definitions

"Next Generation Databases mostly addressing some of the points:
being **non-relational**, **distributed**, **open-source** and **horizontally scalable**."

- Stefan Edlich (nosql-database.org)

Some NoSQL Definitions

"NoSQL: a broad class of data management systems where
the data is partitioned across a set of servers,
where no server plays a privileged role."

- Emin Gün Sirer (hackingdistributed.com)

Some NoSQL Definitions

“[To organise a meetup in the late 2000s]… **you need a twitter #hashtag**…That’s all #nosql was ever meant to be, a twitter hashtag to organise a single meetup at one point in time”

- Martin Fowler (goto; 2012)

ACID, BASE and CAP

ACID – A Recap

In an ideal world, database transactions should be:

- **Atomic**
  Entire transaction succeeds or the entire transactions rolls back
- **Consistent**
  A transaction must leave the database “valid” re: some defined rules
- **Isolated**
  Concurrent interactions behave as though they occurred serially
- **Durable**
  Once committed, transactions survive power loss, acts of god etc.

Considered a key requirement for RDBMSes

The CAP Theorem – a Recap

You can only ever have two of the following three:

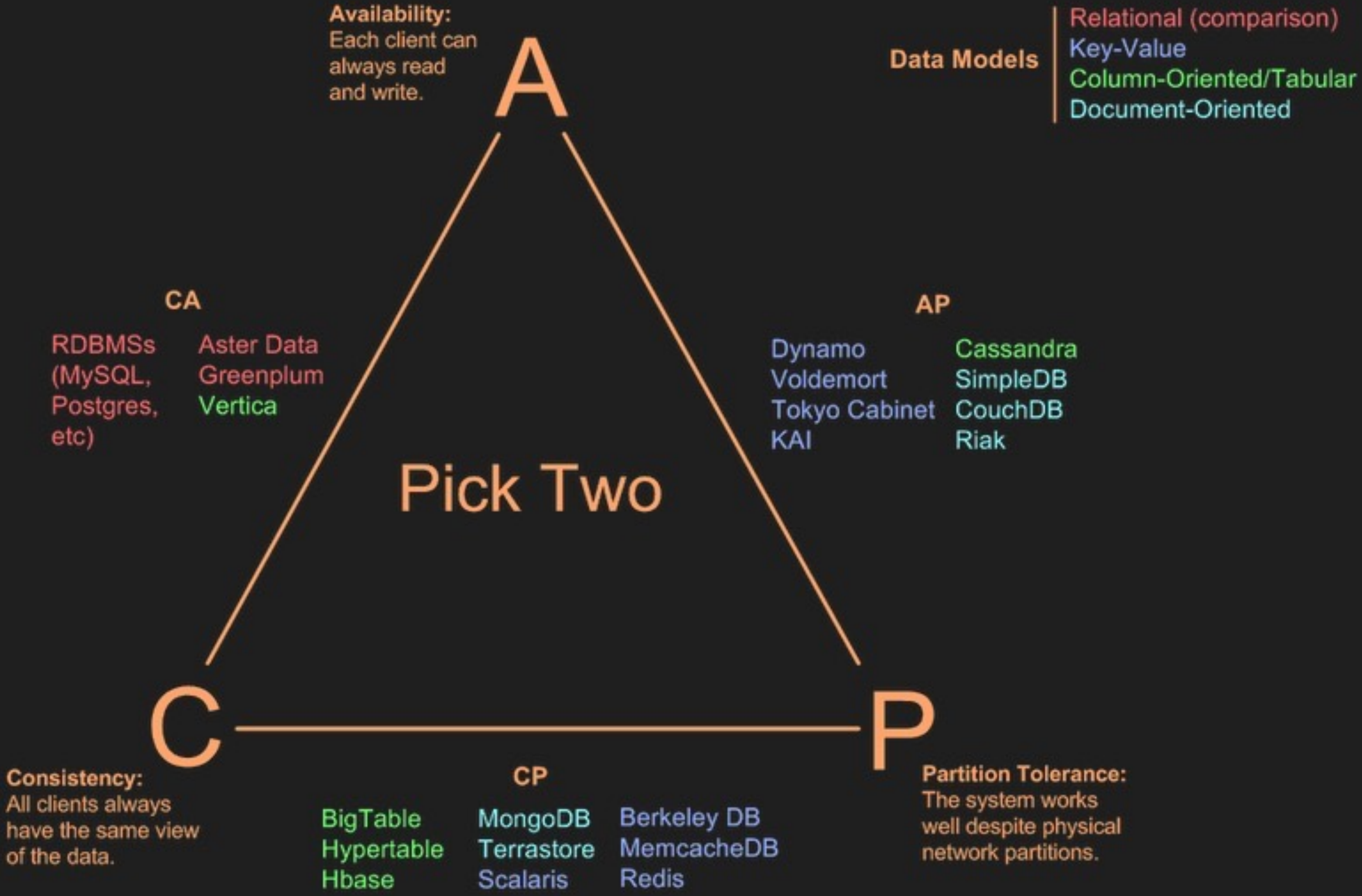
- **Consistent**: writes are atomic, all subsequent requests retrieve the new value
- **Available**: the database will always return a value so long as the server is running
- **Partition Tolerant**: the system will still function even if the cluster network is partitioned (i.e. the cluster loses contact with parts of itself)

To put it a different way, partitions force us to choose one of:

- Consistency (i.e. we disallow writes during the partition)
- Availability (i.e. we allow writes during a partition)

Gilbert, S. and Lynch, N. (2002) Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News 33(2), pp.51-59.

Visual Guide to NoSQL Systems



BASE – An alternative to ACID

A gratuitous backronym:

- **B**asic **A**vailability
  The application works basically all the time
- **S**oft-state
  Does not have to be consistent all the time
- **E**ventual consistency
  But will be in some known state eventually

Eventual Consistency (weak mutual consistency)

If we write to a replicated data item, updates to the replicas need not happen during the lifetime of the transaction, but should happen eventually

From Amazon's Dynamo paper:

"the storage system guarantees that if no new updates are made to the object, **eventually all accesses** will return the last updated value."

Two common approaches:

- MVCC
- Vector Clocks

Multi-Version Concurrency Control

Commonly used by NoSQL document databases

- Like a version control system
- Writes without locks
- Multiple versions of documents

Distributed Incremental Replication

- Different versions on different machines
- Collisions detected during replication
- App developer can be informed/decide on collisions

Used by: CouchDB

Vector Clocks

An extension of Lamport timestamps

Represent the order of events in a distributed system

Vector clocks can be used to:

- Identify the provenance of an item of data
- Decide order in which data was changed
- Help resolve conflicts
- Flag inconsistencies for app specific decisions

Used by: Amazon’s Dynamo and Riak

NoSQL Databases

UNIVERSITY OF Southampton

NoSQL Databases

mongoDB

{name: "mongo", type:"DB"}

APACHE HBASE

Cassandra

hadoop

basho

Neo4j
the graph database

redis

NoSQL Varieties

- Key-Value stores (Amazon Dynamo)
- Document Oriented (Lotus notes? Bit of a stretch! Still cool)
- Column Oriented (Google's BigTable)
- Graph DBs (Triples! SPARQL!)

For a roundup see:
http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis

Key-Value Stores

From an API perspective, a hash table with persistence

Use a key (usually a string), ask the database for the corresponding value

The value can be anything (text, structure, an image etc.)

Database often unaware of value content (but not always)

Examples

Riak

- Buckets/Keys/Values/Links
- Query with key, process with map-reduce
- Secondary Indexes (metadata)
- “Loves the Web” (but they all say this)

Redia

- More understanding of value types (strings, integers, lists, hashes)
- In memory (very fast)

Document Databases

Database as storage of a **mass of different documents**

A document…

- … is a **complex data structure**
- … can contain **completely different** data from other documents

Document data stores **understand** their documents

- Queries can run against **values** of document fields
- **Indexes** can be constructed for document fields

Document Databases

```
{
"_id": "1",
  "name": "steve",
  "games_owned": [
    {"name":"Super Meat Boy"},
    {"name":"FTL"},
  ],
}v
```

```
{
  "_id": "2",
  "name": "darren",
  "handle":"zerocool",
  "games_owned": [
    {"name":“FTL"},
    {"name":“Assassin’s Creed 3“, “dev”: “ubisoft”},
  ],
}
```

Examples

MongoDB

- Master/Slave design
- .find() queries like ORM
- Geo-spatial indexing

CouchDB

- Master/master
- Only map-reduce queries
  - Weird but pretty cool, see: http://sitr.us/2009/06/30/database-queries-the-couchdb-way.html
- Favours availability to consistency (more on this in a bit)

Column Databases

Data is held in rows

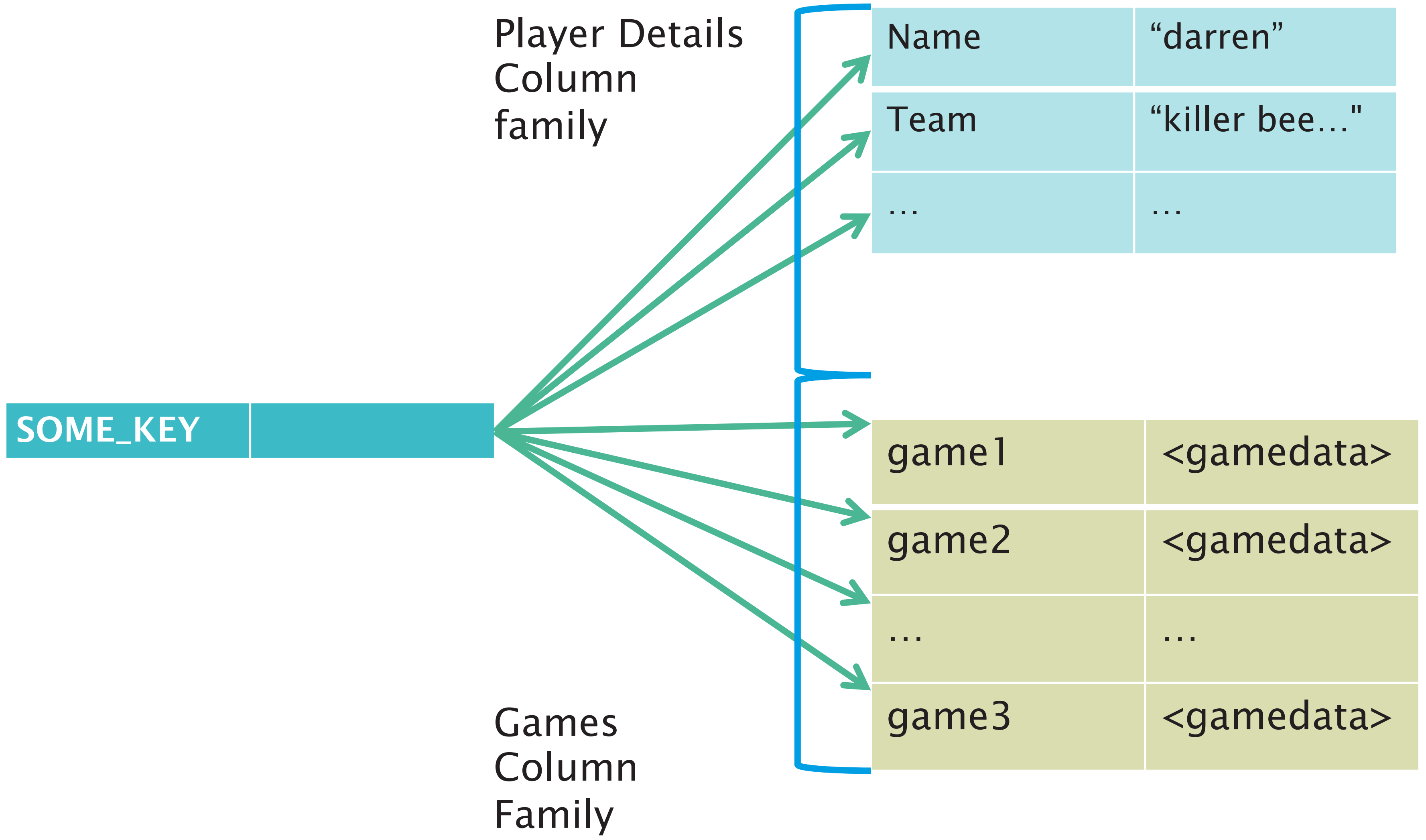
- Rows have keys associated
- Rows contain “column families”
- Column families contain the actual columns, thus data

No Schema (Columns in a family change per row)

On Querying:

- Key lookup is fast
- Batch processing via map-reduce
- All else involves row scans

Column Databases



Examples

Hbase

- Uses HDFS for storage, Hadoop for processing
- Built to treasure consistency over availability

Cassandra

- Supports key ranges
- Works over a variety of processing architectures (Hadoop, Storm, etc.)

Graph Databases

Focus on modelling the data’s structure

Graphs are composed of **Vertices** and **Edges**

- **Vertices** are connected by **edges**
- **Edges** have labels and direction
- **Both** have properties

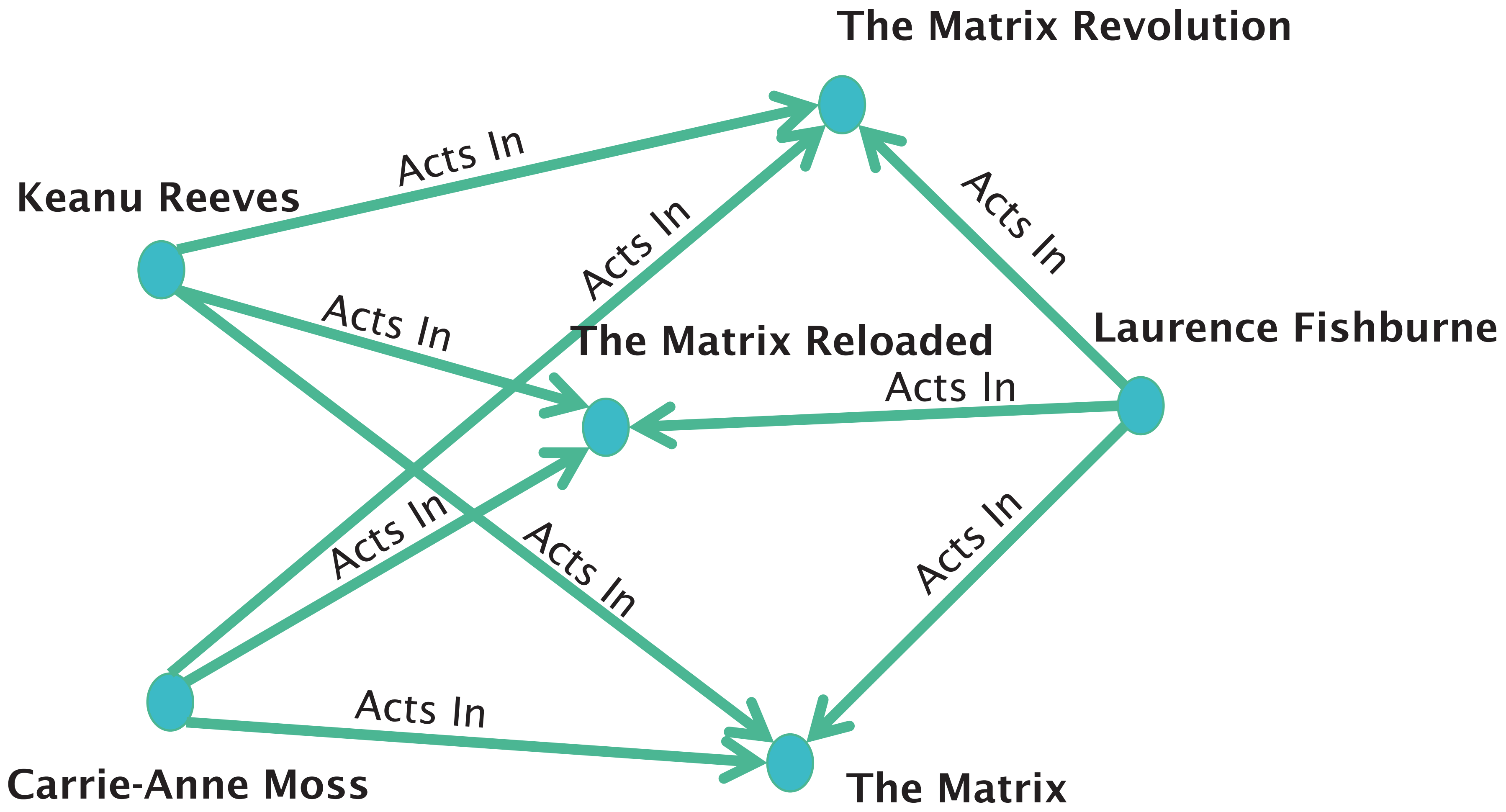
Queried with graph traversal API or graph query language

- Cypher, SPARQL

Can be much faster at querying graph like data structures

- **Like** friends of friends or web links

Graph Databases



Examples

Neo4j

- Not distributed
- ACID transactions

From NoSQL to NewSQL

The NoSQL discussion has nothing to do with SQL

Michael Stonebraker

The NoSQL performance argument

1. I use MySQL to store my data
2. MySQL’s performance isn’t adequate
3. Partitioning my data across multiple sites is hard!
4. I don’t want to pay license fees for an enterprise RDBMS

∴

NoSQL is the way to go!

The NoSQL Performance Argument

Transaction cost in OLTP database consists largely of:

- Logging (write to database, write to log)
- Locking (recording locks in lock table)
- Latching (updating shared data structure: B-trees, lock table, etc)
- Buffer Management (buffer pool containing cached disk pages)

“The single-node performance of a NoSQL, disk-based, non-ACID, multithreaded system is limited to be a modest factor faster than a well-designed stored-procedure SQL OLTP engine” – the overhead isn’t due to SQL

NoSQL in the Enterprise

No ACID equals No Interest

- Stored data is mission critical, inconsistency is dangerous

A Low-Level Query Language is Death

- Record-at-a-time processing (c.f. IMS, CODASYL) requires far greater programming effort - declarative languages like SQL are preferable

NoSQL means No Standards

- Many different NoSQL databases, each with a different interface, data model, etc – how do you migrate from one to another?

Tick-tock, tick-tock...

...and back to relational databases again!

NewSQL

- The scale-out OLTP performance of NoSQL...
- ...with the SQL support and ACID guarantees of RDBMS

Further Reading

Some further reading…

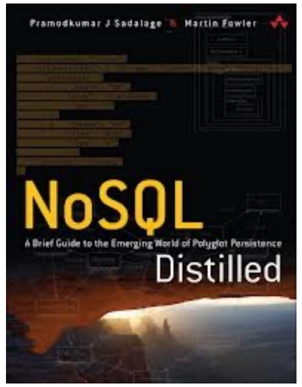
The structure/content of these slides are covered in greater depth in:

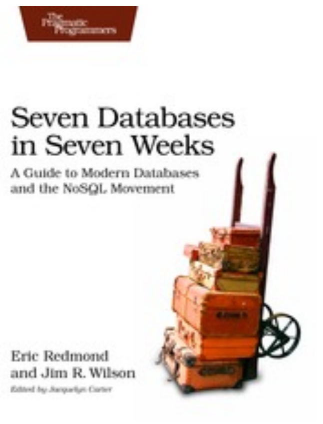
“Seven Databases in Seven Weeks” by Eric Redmond

“NoSQL Distilled” by Martin Fowler

Mike Stonebraker’s blogs for CACM

https://cacm.acm.org/blogs/blog-cacm/?author=Michael+Stonebraker




The Pragmatic Programmers
Seven Databases in Seven Weeks
A Guide to Modern Databases and the NoSQL Movement
Eric Redmond and Jim R. Wilson

… and some watching!

“**Introduction to NoSQL**” – Martin Fowler @ goto; 2012
http://www.youtube.com/watch?v=qI_g07C_Q5I

“**The People vs. NoSQL Databases**” – Panel Discussion @ goto; 2012
http://www.youtube.com/watch?v=191kCkNya5Q (NSFW language)

“**MongoDB: It’s Not Just About Big Data**” – Will Shulman
http://www.youtube.com/watch?v=b1BZ9YFsd2o

Next Lecture: Data Streams