NoSQL & NewSQL

Thomas Heinis

Imperial College
London

Some Data Management History…

The Database

What about…

Reliability?
Security?
Consistency?
Response time?
Scalability?

Relational Databases

- Structured, schema-based organization of data
- Data decomposed into tables, ex. customer data:

Customer		
Name	**Date**	**City**
Dustin	12.1.1971	London
Jack	8.19.1965	Leicester

City	
City	**Country**
London	UK
Leicester	UK

- SQL - General-purpose query language
- Join tables to retrieve all data: e.g., SELECT * FROM Customer, City WHERE City.City = Customer.City
- Focused on strong consistency (ACID)

Transactions – Data Integrity

Why Concurrent Access to Data must be managed?

John and Jane withdraw $50 and $100 from a common account…

John:

1. get balance
2. if balance > $50
3. balance = balance - $50
4. update balance

Jane:

1. get balance
2. if balance > $100
3. balance = balance - $100
4. update balance

Initial balance $300. Final balance=?

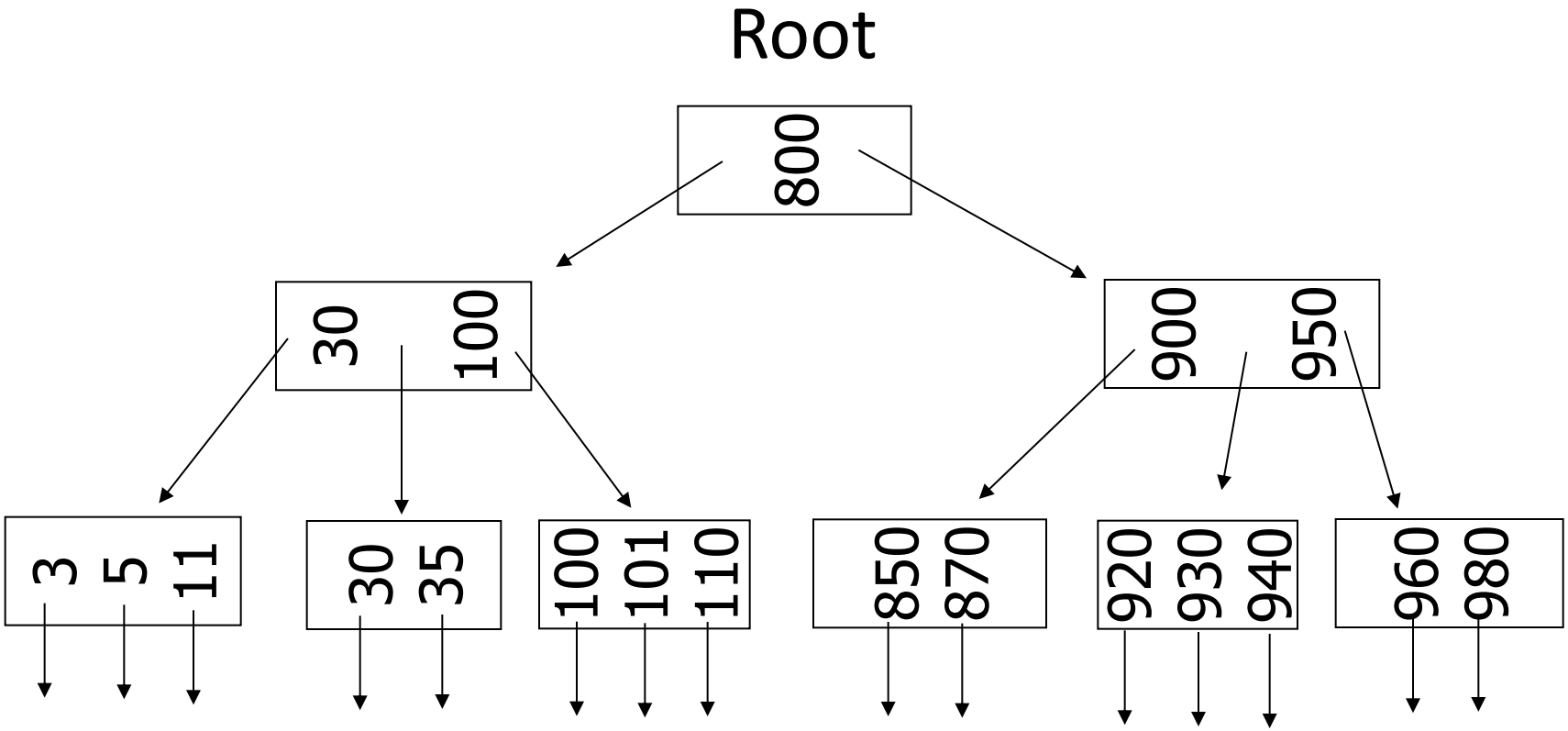
It depends…

Indexing – Efficient Data Retrieval

- How can we answer the query: “Find the account with the balance of 920.-” efficient?

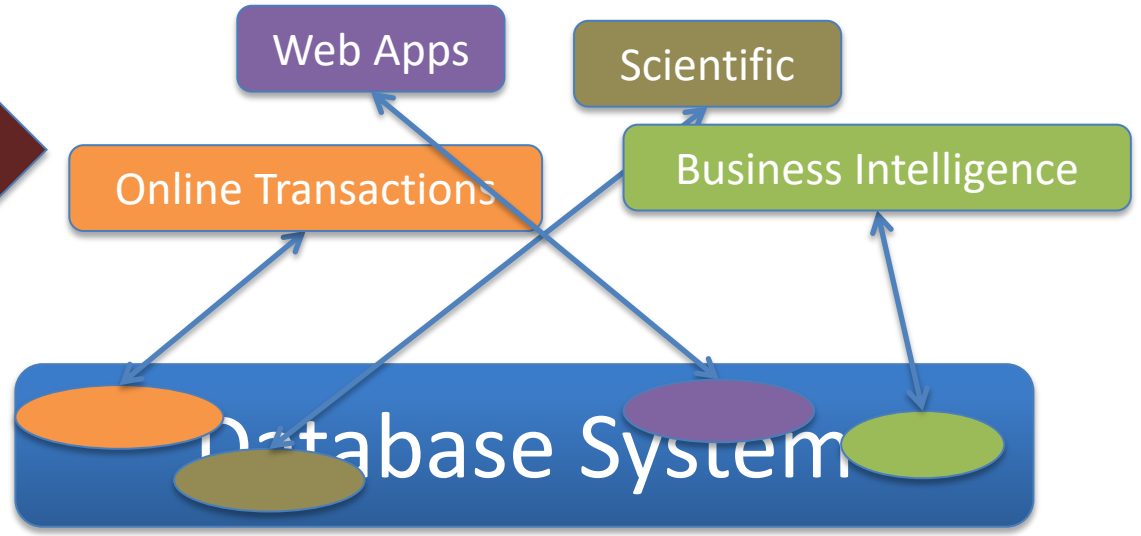
- One approach is to scan the entire customer table, check every customer, return the one with balance = 920 … very slow for large databases

Example Index (B-Tree)



Database Systems Today

New applications, more requirements!



Rapidly changing hardware, new performance hurdles!

Relational Databases: One Size Does Not Fit All…

New applications challenge relational databases:

1. Strong consistency (ACID) limits scalability
2. Schema evolution is challenging
3. Little optimization for novel hardware
4. Cumbersome language
5. Limited data types

NoSQL: Overview

- Main objective: implement distributed state
  - Different objects stored on different servers
  - Same object replicated on different servers
- Main idea: give up some of the ACID constraints to improve performance
- Simple interface:
  - Write (=Put): needs to write all replicas
  - Read (=Get): may get only one
- Eventual consistency ← Strong consistency

NoSQL

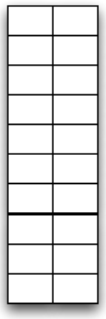
"Not Only SQL" or "Not Relational".

Six key features:

1. Scale horizontally "simple operations"
2. Replicate/distribute data over many servers
3. Simple call level interface (contrast w/ SQL)
4. Weaker concurrency model than ACID
5. Efficient use of distributed indexes and main memory
6. Flexible schema

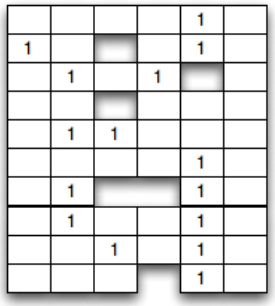
Four NOSQL Categories

Key-Value

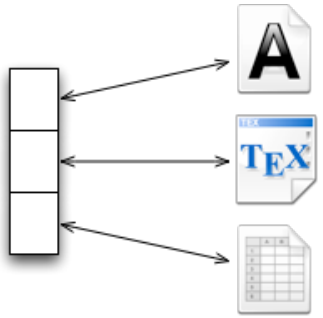


Graph DB

BigTable



Document



Data Model

- Tuple = row in a relational DB
- Document = nested values, extensible records (think XML or JSON)
- Extensible record = families of attributes have a schema, but new attributes may be added
- Object = like in a programming language, but without methods

Key-value Stores

- Think “file system” more than “database”
- Consistent hashing (DHT)
- Only primary index: lookup by key
- No secondary indexes
- Transactions: single- or multi-update

Basic Idea: Key-Value Store

Table T:

key	value
k1	v1
k2	v2
k3	v3
k4	v4

- API:
  - lookup(key) → value
  - lookup(key range) → values
  - getNext → value
  - insert(key, value)
  - delete(key)
- Each row has timestemp
- Single row actions atomic
  No multi-key transactions
- No query language!

Document Stores

- A "document" = a pointerless object = e.g. JSON = nested or not = schema-less
- In addition to KV stores, may have secondary indexes
- SimpleDB, CouchDB, MongoDB, Terrastore
- Scalability:
  - Replication (e.g. SimpleDB, CounchDB – means entire db is replicated),
  - Sharding (MongoDB);
  - Both

Document Store (MongoDB)

```
> db.user.insert({
      first: "John",
      last : "Doe",
      age: 39
})
```

```
> db.user.find ({"first" : "John"})
{
      "_id" : ObjectId("51…"),
      "first" : "John",
      "last" : "Doe",
      "age" : 39
}
```

```
> db.user.update(
      {"_id" : ObjectId("51…")},
      {
          $set: {
              age: 40,
              salary: 7000}
      }
)
```

```
> db.user.remove({
      "first": /^J/
})
```

Column Family

- Most Based on **BigTable**: Google's Distributed Storage System for Structured Data
- Data Model:
  - A big table, with column families
  - Map Reduce for querying/processing
- Examples:
  - HBase, HyperTable, Cassandra

Column Family: Pros and Cons

- Pros:
  - Supports Semi-Structured Data
  - Naturally Indexed (columns)
  - Scalable
- Cons
  - Poor for interconnected data

Graph Databases

- Data Model:
  - Nodes and Relationships
- Examples:
  - Neo4j, OrientDB, InfiniteGraph, AllegroGraph

Graph Databases: Pros and Cons

- Pros:
  - Powerful data model, as general as RDBMS
  - Connected data locally indexed
  - Easy to query
- Cons
  - Sharding (lots of people working on this)
    - Scales UP reasonably well
  - Requires rewiring your brain

What are graphs good for?

- Recommendations
- Business intelligence
- Social computing
- Geospatial
- Systems management
- Web of things
- Genealogy
- Time series data
- Product catalogue
- Web analytics
- Scientific computing (especially bioinformatics)
- Indexing your *slow* RDBMS
- And much more!

What is a Graph?

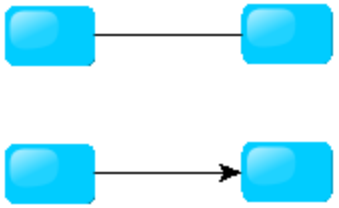
An abstract representation of a set of objects where some pairs are connected by links.

Object (Vertex, Node)

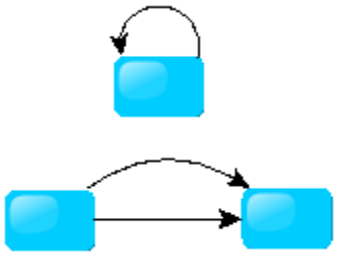
Link (Edge, Arc, Relationship)

Different Kinds of Graphs

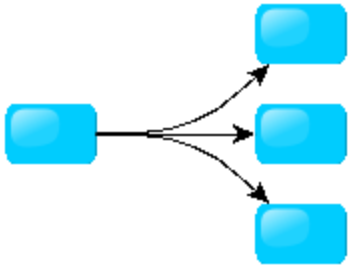
- Undirected Graph
- Directed Graph



- Pseudo Graph
- Multi Graph

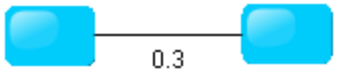


- Hyper Graph

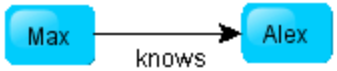


More Kinds of Graphs

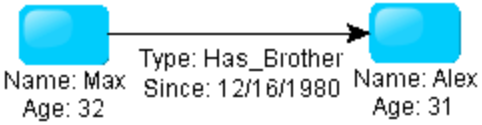
- Weighted Graph



- Labeled Graph



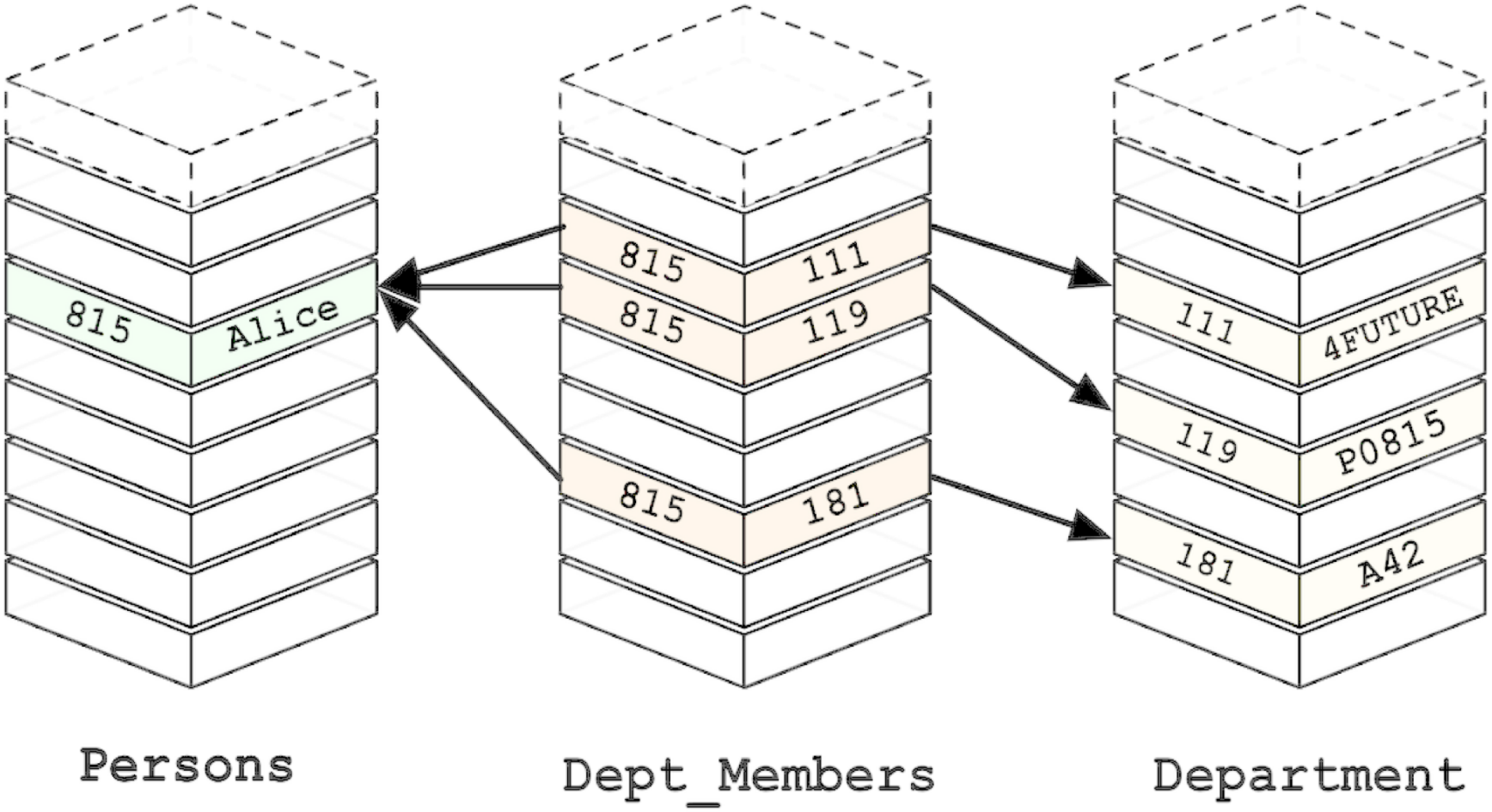
- Property Graph



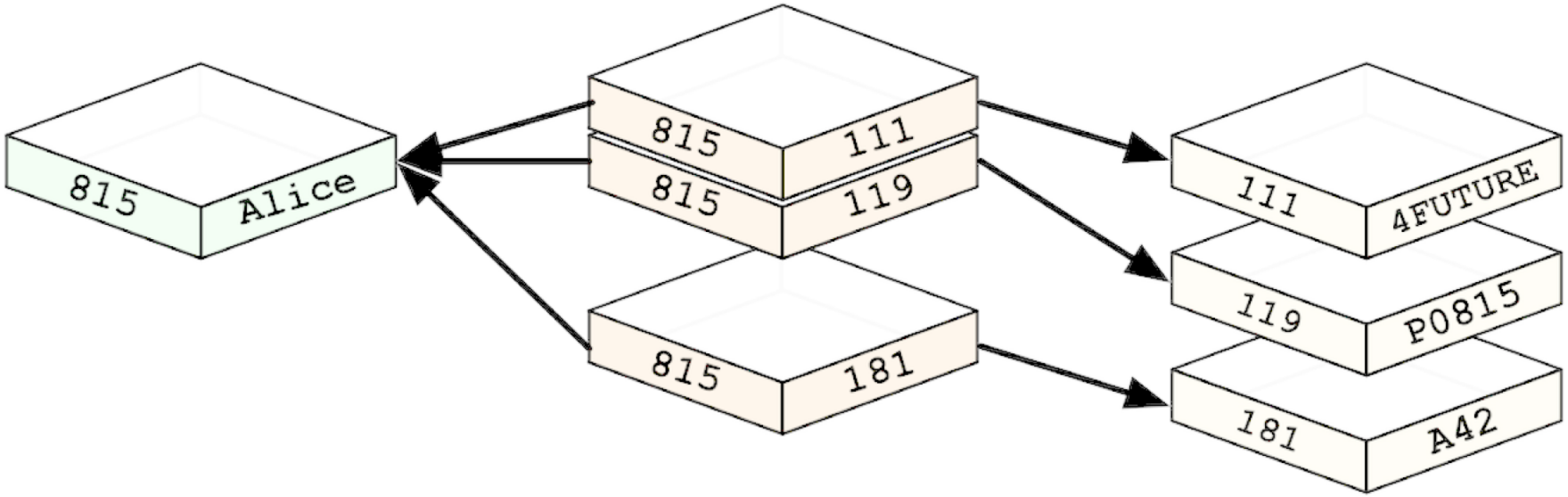
What is a Graph Database?

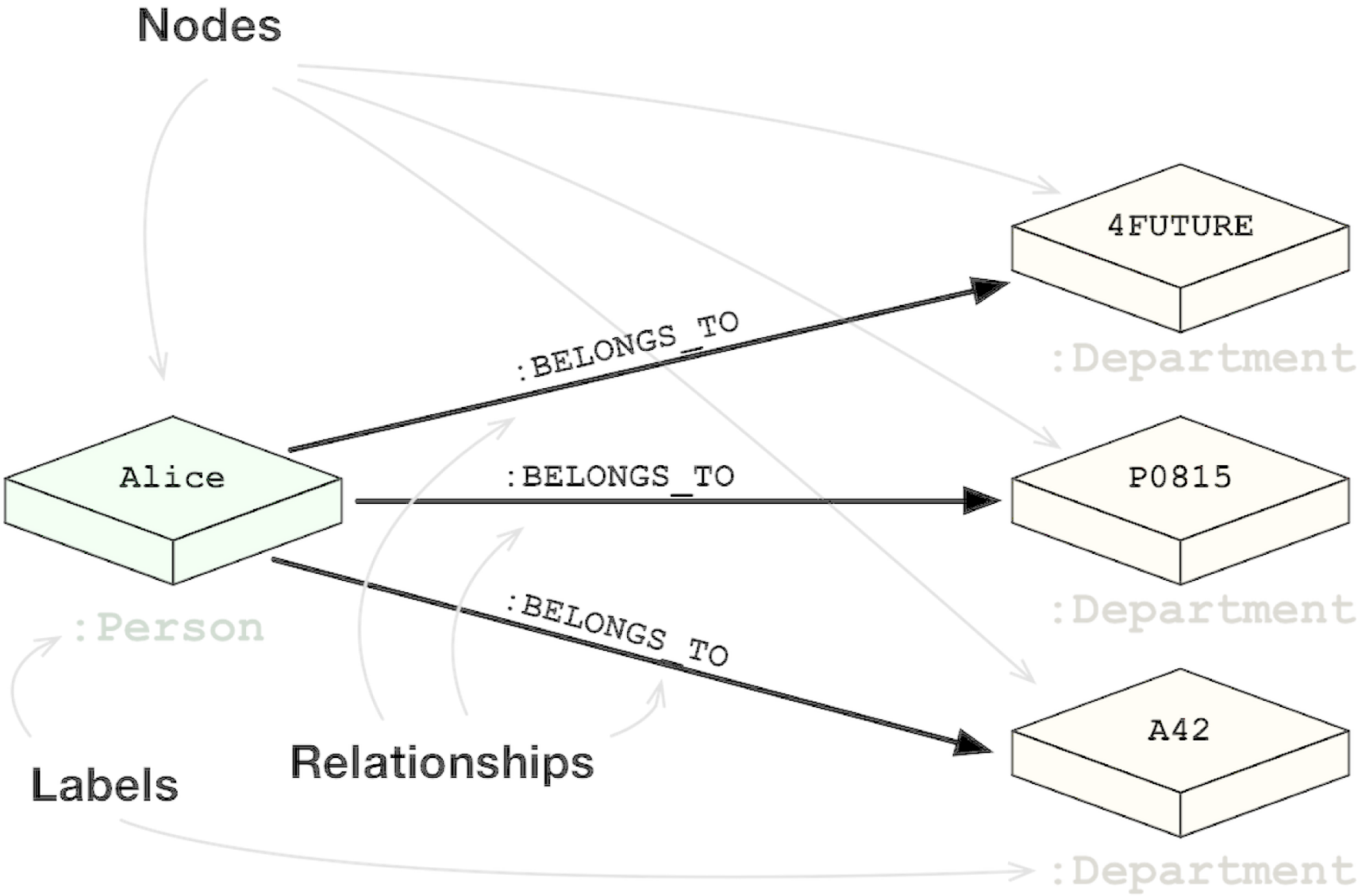
- A database with an explicit graph structure
- Each node knows its adjacent nodes
- As the number of nodes increases, the cost of a local step (or hop) remains the same
- Plus an Index for lookups

Relational Databases



Graph Databases

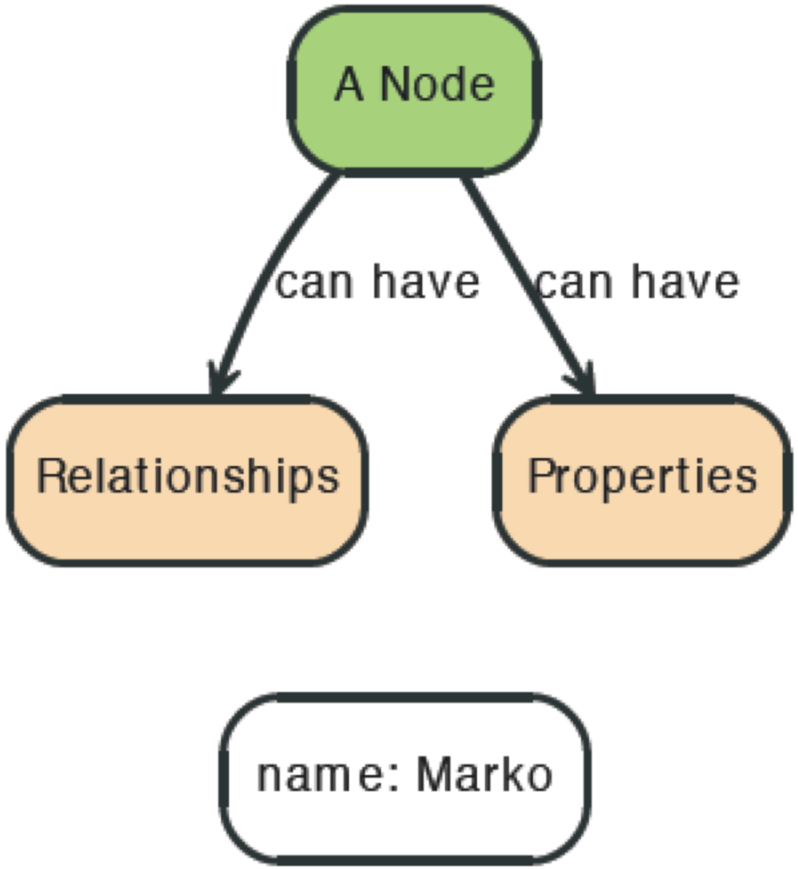



Nodes
4FUTURE
:Department
:BELONGS_TO
Alice
:BELONGS_TO
P0815
:Department
:Person
:BELONGS_TO
A42
Labels
Relationships
:Department

Neo4j Approach

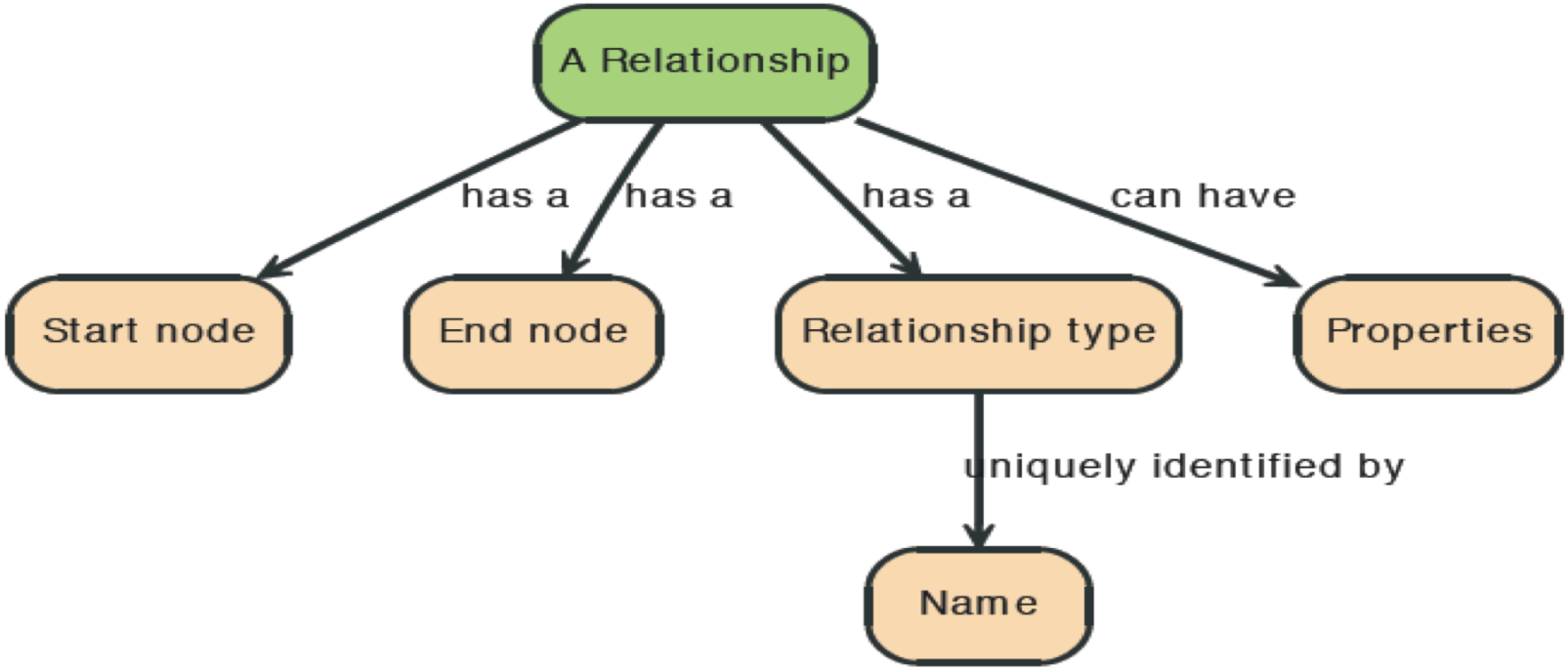
- Each entity table is represented by a label on nodes
- Each row in a entity table is a node
- Columns on those tables become node properties.
- Remove technical primary keys, keep business primary keys
- Add unique constraints for business primary keys, add indexes for frequent lookup attributes

Node in Neo4j

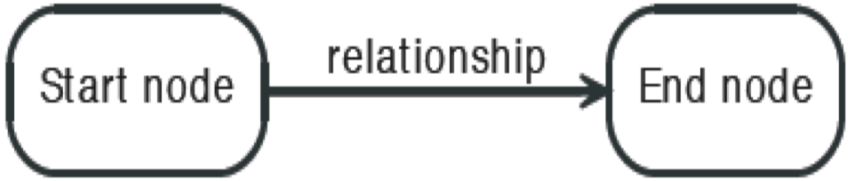


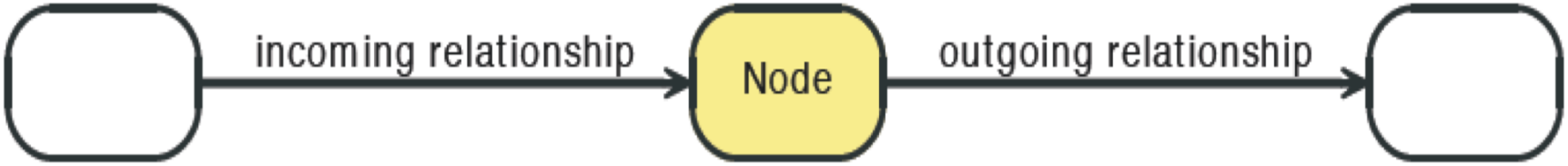
Relationships in Neo4j

- Relationships between nodes are a key part of Neo4j.

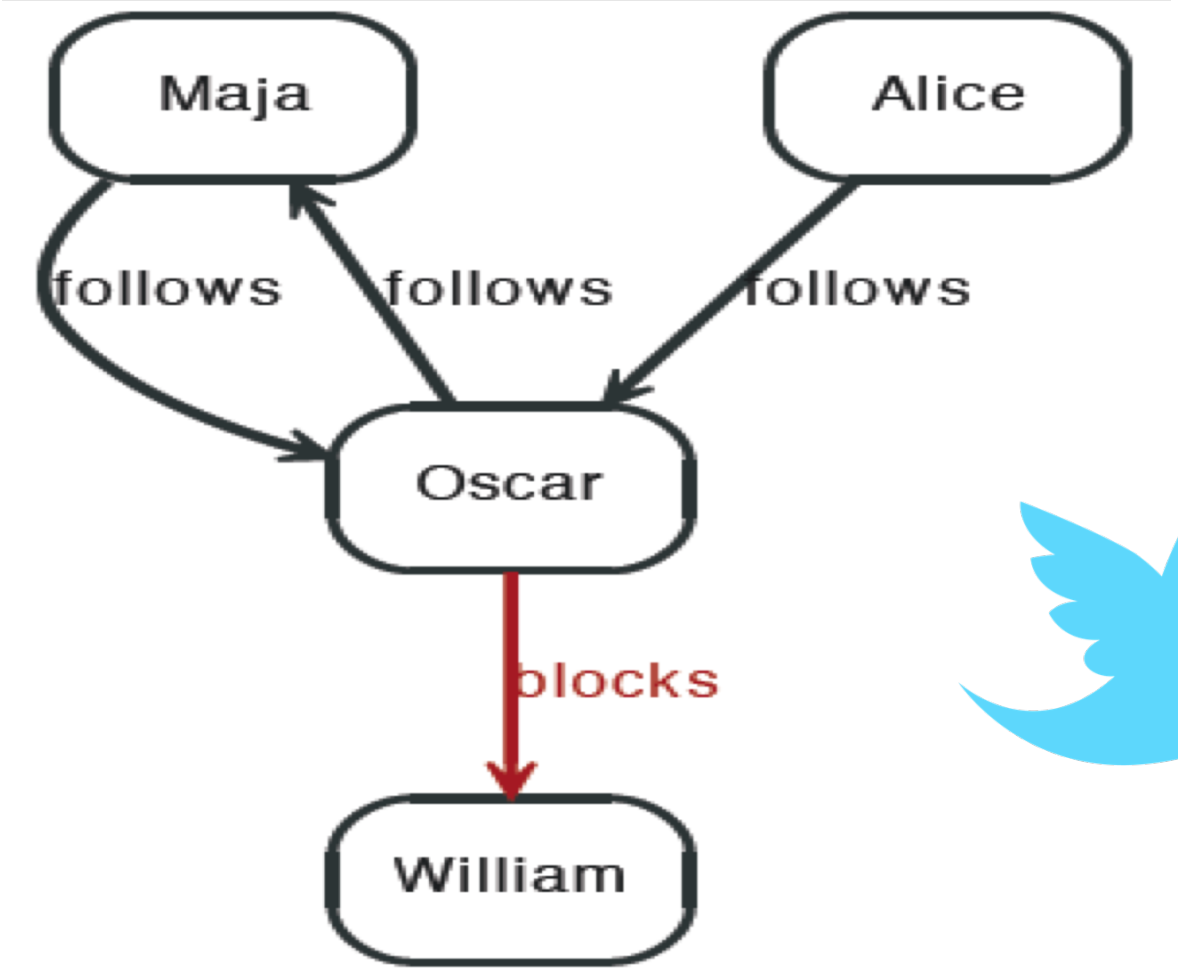


Relationships in Neo4j





Twitter and relationships



(a:Person) – [:follows] -> (b:Person) – [:follows] -> (c:Person)

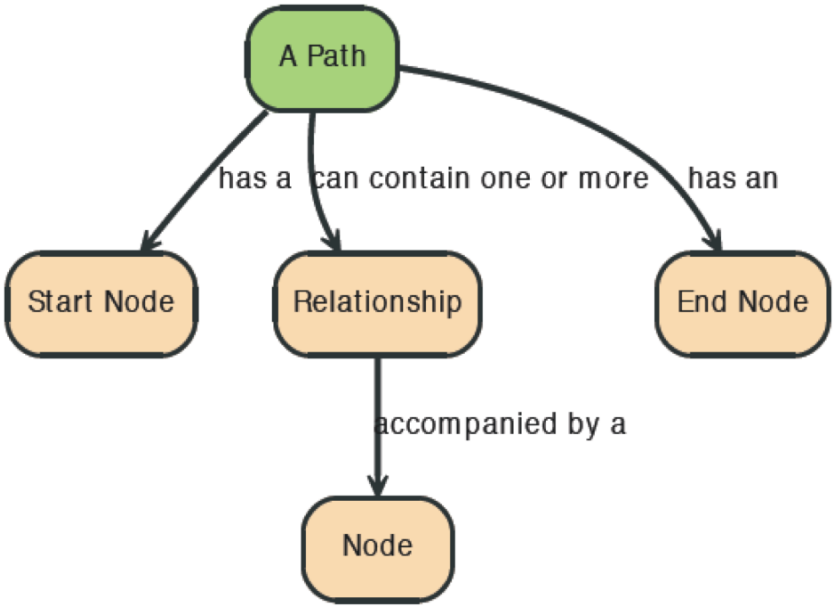
Properties

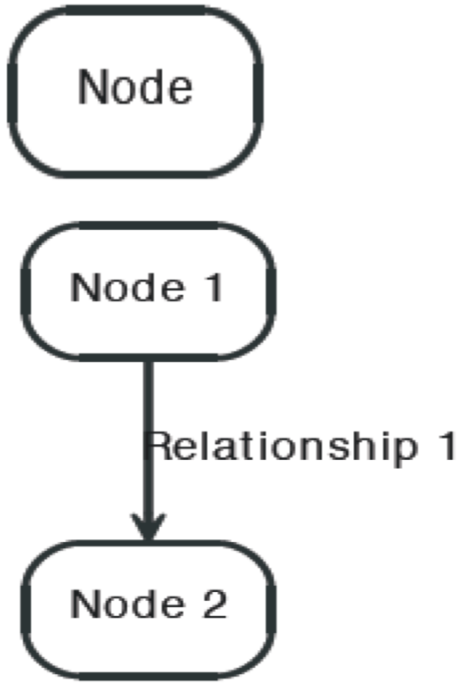
- Both nodes and relationships can have properties.
- Properties are key-value pairs where the key is a string.
- Property values can be either a primitive or an array of one primitive type.

  For example String, int and int[] values are valid for properties.

Paths in Neo4j

A path is one or more nodes with connecting relationships, typically retrieved as a query or traversal result.





Scalable Relational Systems (NewSQL)

- Means RDBMS that offer sharding
- **Key difference**:
  - NoSQL difficult or impossible to perform large-scope operations and transactions (to ensure performance)
  - Scalable RDBMS do not **preclude** these operations, but users pay a price only when they need them
- MySQL Cluster, VoltDB, Clusterix, ScaleDB, Megastore (the new BigTable)
- Many more **NewSQL** systems becoming available…

Scalable Data Processing

- Parallel execution achieves greater efficiency
- But, parallel programming is hard
  - Parallelization
  - Fault Tolerance
  - Data Distribution
  - Load Balancing

MapReduce (Hadoop and others)

- "*MapReduce is a programming model and an associated implementation for processing and generating large data sets*"
- Programming model
  - Abstractions to express simple computations
- Library
  - Takes care of the gory stuff: Parallelization, Fault Tolerance, Data Distribution and Load Balancing

NoSQL Systems

- Key Value Stores

Key	Value
SW3 3TB	London
B18 4BJ	Birmingham

- Document Stores

- Scalable SQL Systems

- Data Processing Systems

NoSQL Challenges

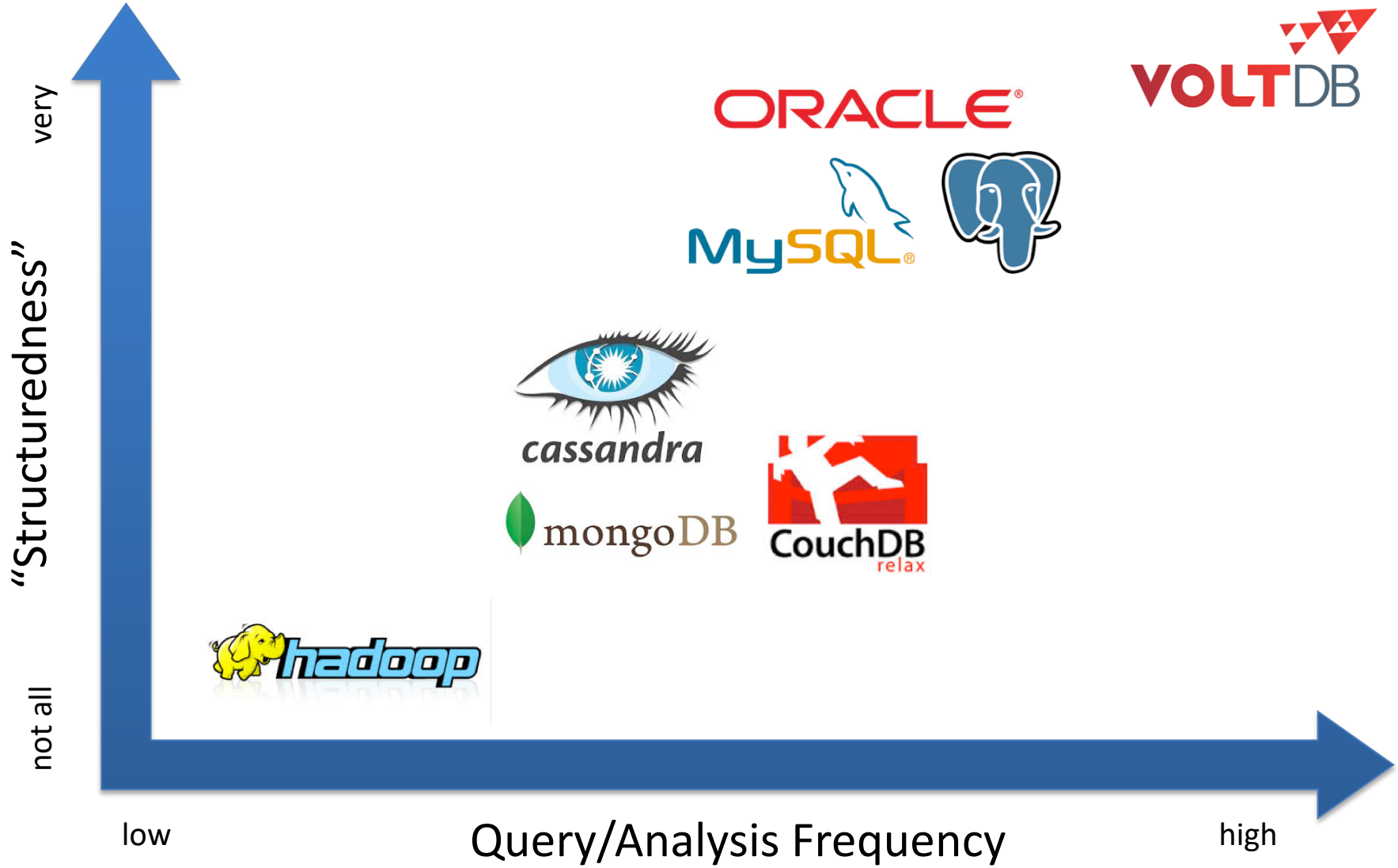
E.g., Flexible Schemas in MongoDB

```
db.inventory.insert(
  {
    category: ”vacuum",
    details: {
      model: "14Q3",
      manufacturer: "XYZ Company"
    },
    stock: [ { size: "S", qty: 25 }],
    category: "clothing"
  }
)
```

```
db.inventory.insert(
  {
    category: ”vacuum",
    details: {
      model: "14Q3",
      manufacturer: "XYZ Company"
    },
    color: ”blue"
  }
)
```

What fields does db.user.find ({”category" : ”vacuum”}) have?

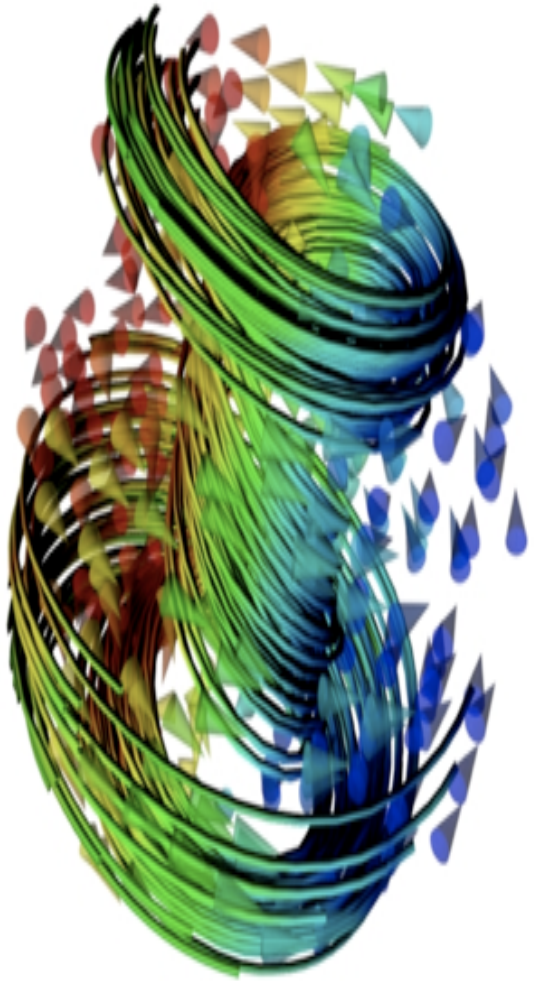
Data Management Landscape*



* according to me

What about Scientific Data?

- Cannot be represented as a table → different data models
- Do not need the abstraction of transaction: typically read-only
- Updates rare
- Massive in size → Indexing considerable overhead

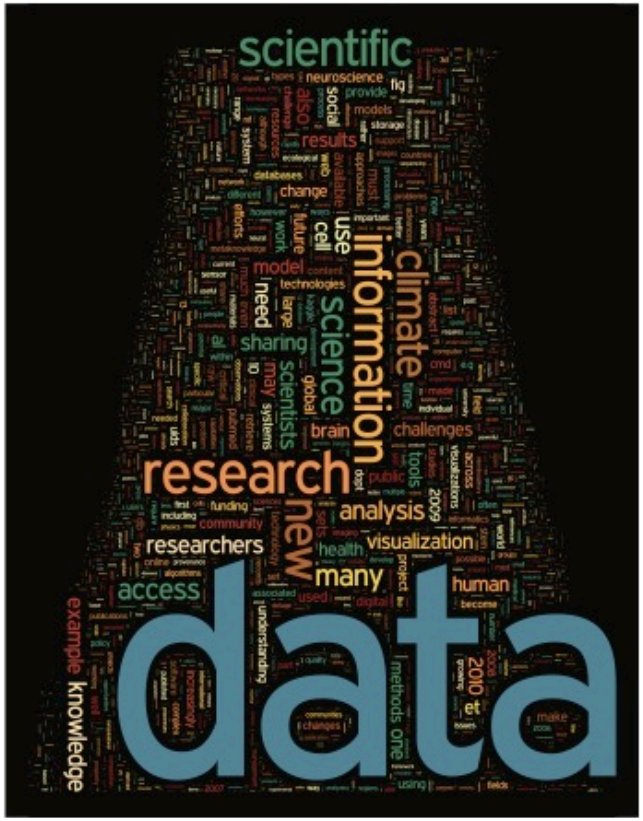


Many database optimizations irrelevant!

Data-driven Science

Past:

- theory
- simulation
- experiments



The “fourth paradigm”

scientific breakthrough through computing on massive data

Scientific Data Management

Today:

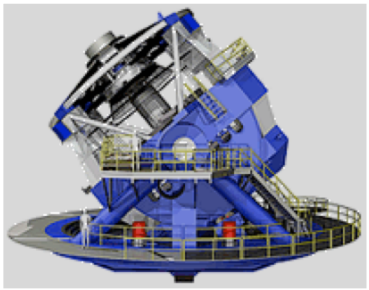
- legacy software
- in main memory of supercomputers
- databases too rigid to use

But as data grows, problem *changes:*

- difficult and slow
- some data discarded

Astronomy

~30,000 movies,



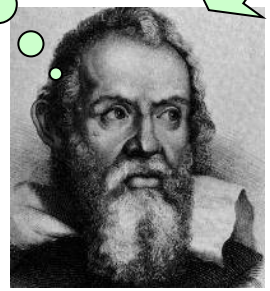
20 TB/day!

Table of Observations

	TYPE	CX	CY	E1	E2	E3	…

Large Synoptic Survey Telescope (www.lsst.org) Due 2017

Find galaxy clusters



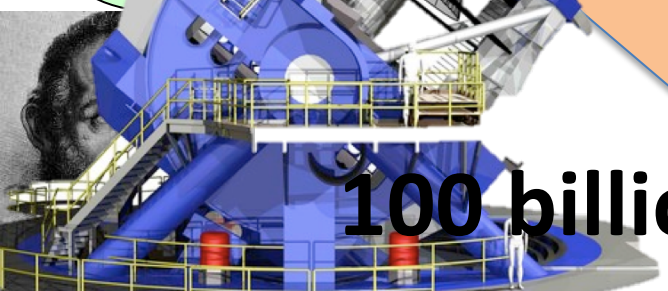
…in a second?

Large Synoptic Survey Telescope (LSST)

date	time	type	category	Cx	Cy	Cz	…
mon	22:00	star	*proto*	…	…	…	…
mon	22:15	star	*red giant*	…	…	…	…
mon	22:2[illegible]	galaxy	*spiral*	…	…	…	…
mon	22:2[illegible]	star	*dwarf*	…	…	…	…
[illegible]	23:00	galaxy	*elliptical*	…	…	…	…
tue	22:1[illegible]	galaxy	*spiral*	…	…	…	…
tue	22:25	star	*proto*	…	…	…	…
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…

index on "type"

Fast r[illegible] vi[illegible] g[illegible]es



100 billion objects, 20 PB/night

Astronomy DBs

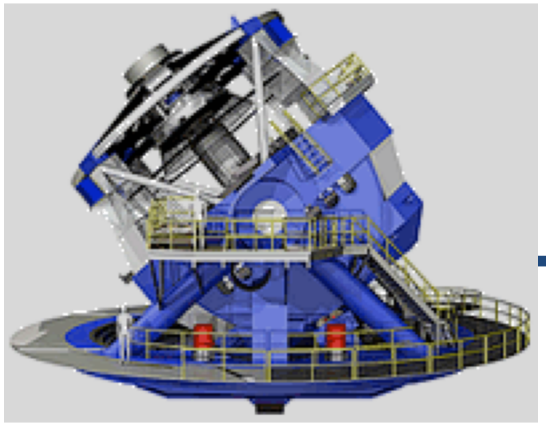


Table with Observations

	TYPE	CX	CY	E1	E2	E3	…

Unused part

Q1

Q2

REPLACE!

	TYPE	CX	CY

	E1	E2	E3

	…

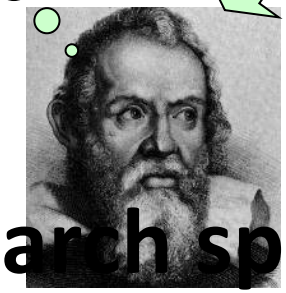
Index

	TYPE	CX	CY

	E1

	E2	E3

Find galaxy clusters



Search space exponentially growing

Designing Astronomy DBs

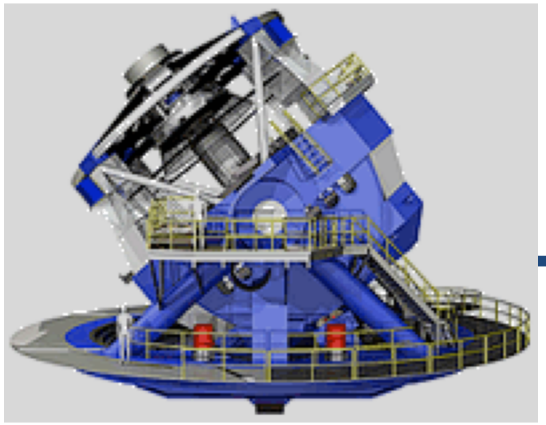


Table with Observations

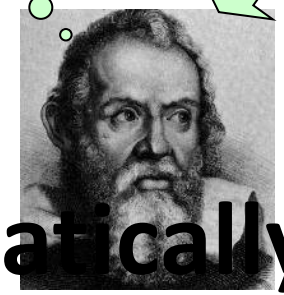
	TYPE	CX	CY	E1	E2	E3	…

Billions of possible ways to organize data on the disk!

Unused part

REPLACE!

Find galaxy clusters



Index

	TYPE	CX	CY

	E1	E2	E3

	…

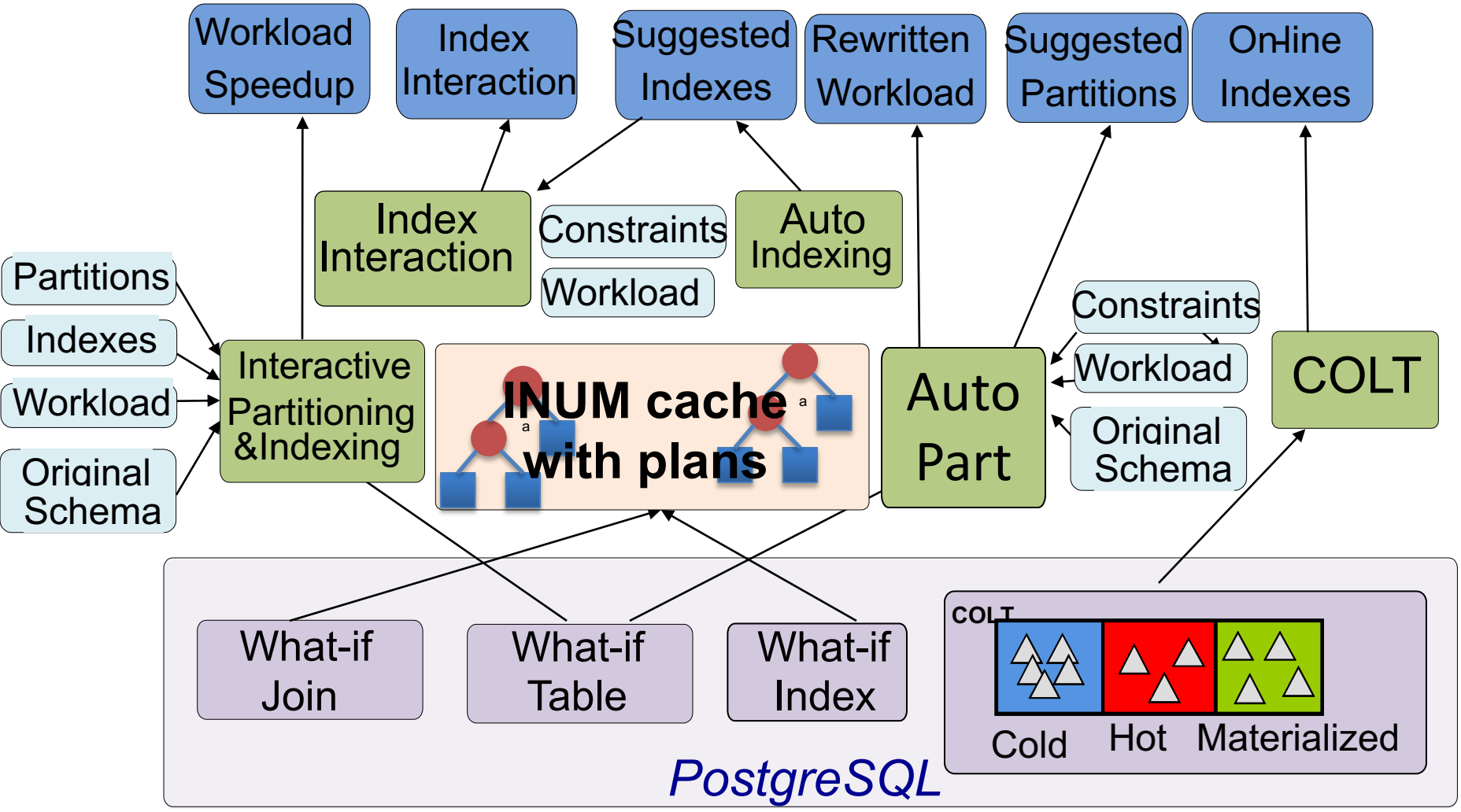
	TYPE	CY

	E1

	E2	E3

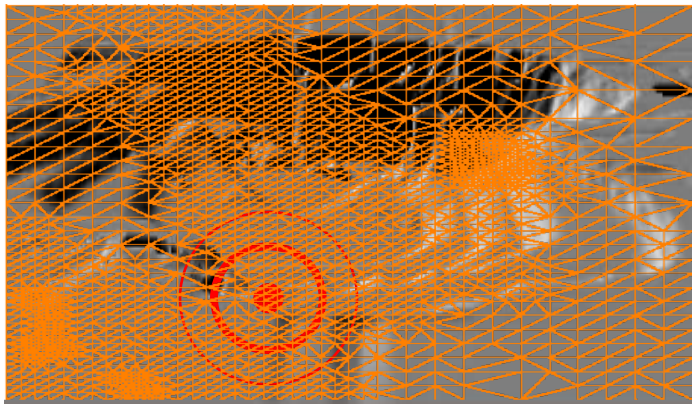
Automatically design database structures

Database Designer

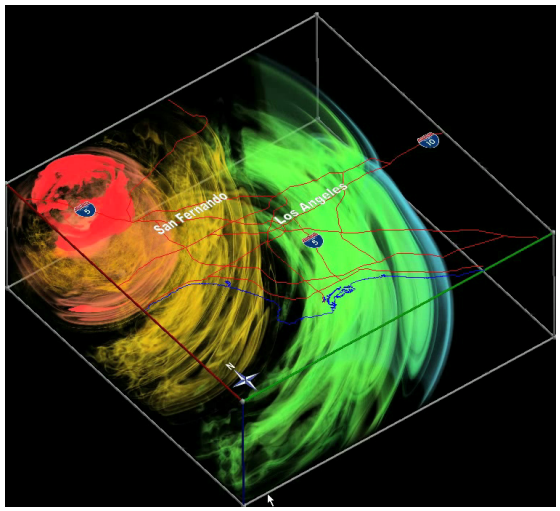


Earthquake Simulation/Analysis

Scientist

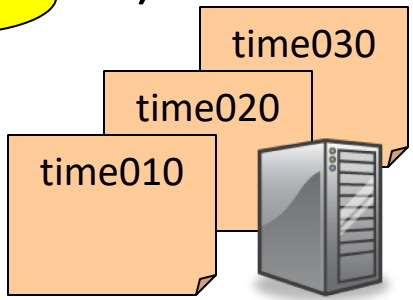


Simulation Model



Simulation Output

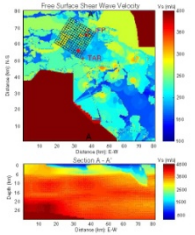
Anal… fopen('time050.dat') …esh)



100GB per time-step, 20000 time-steps

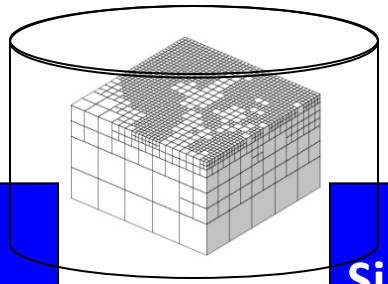
performance? access to complex structures?

Earthquake Simulation



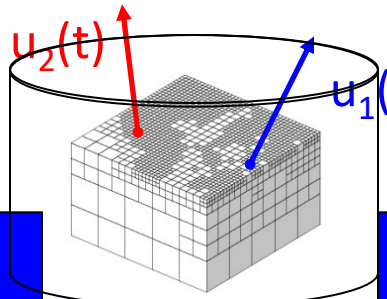
Ground model

Mesh generation



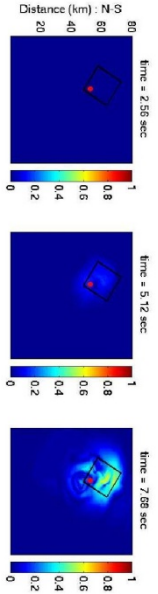
Mesh

Simulation

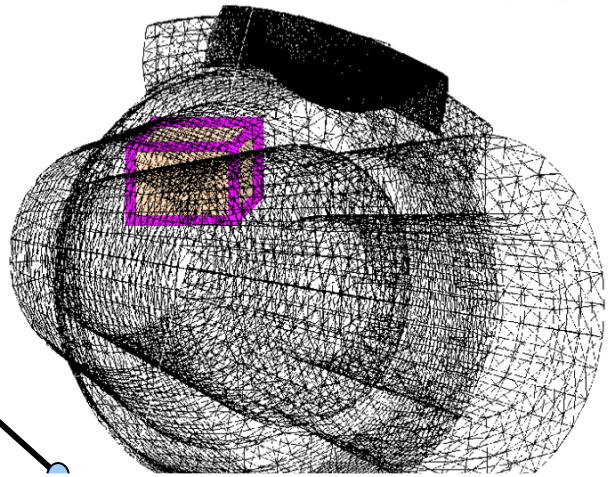


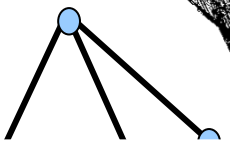
Time-varying output

Analysis Visualization



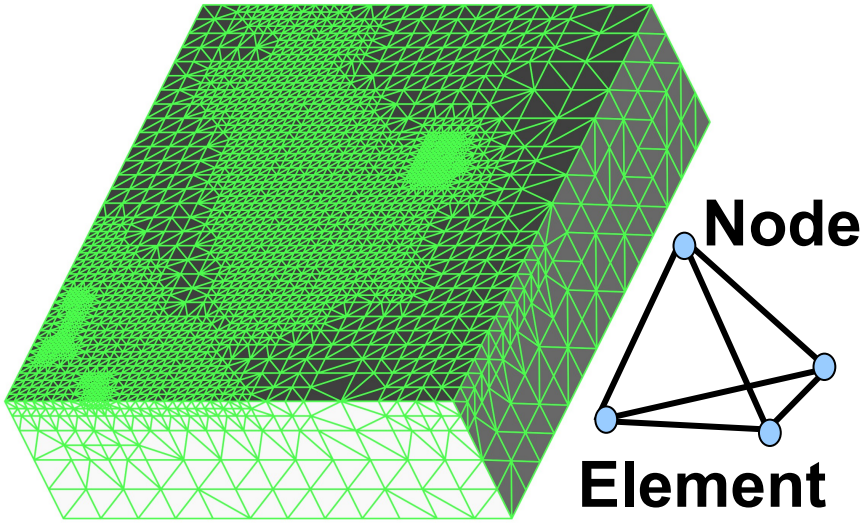
- Huge models, huge outputs
- Not scalable, need database systems
- Mesh representation for fast queries?



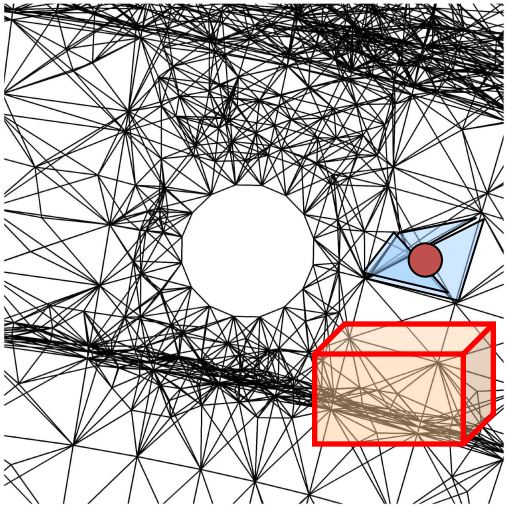


Search entire mesh for regions (not scalable)

Tetrahedral Mesh Models



- Queries
  - Point
  - Range
  - Feature

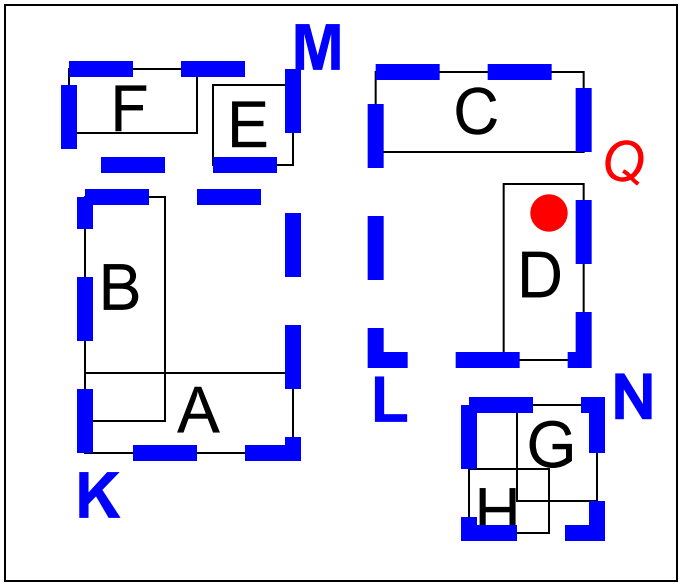


- Example: Visualization
  - Show ground velocity at **Q**
  - Draw the temperature of **R**

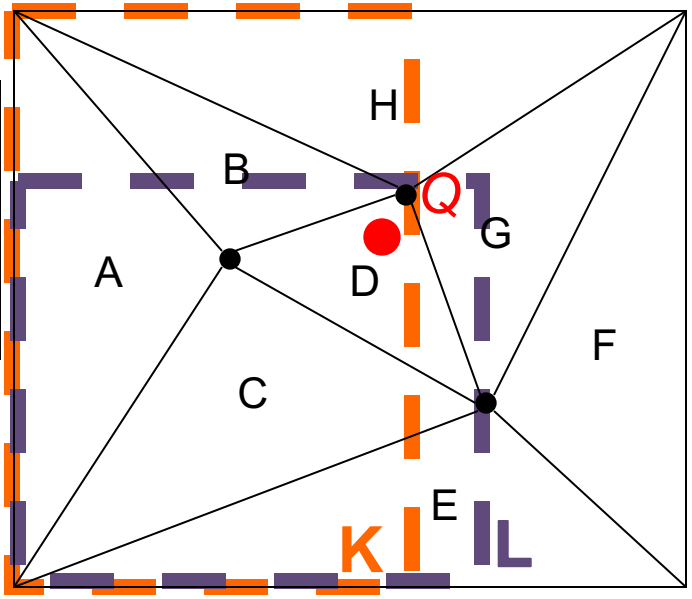
Courtesy Cornell Fracture Group

Goal: efficiently process *geometric* queries

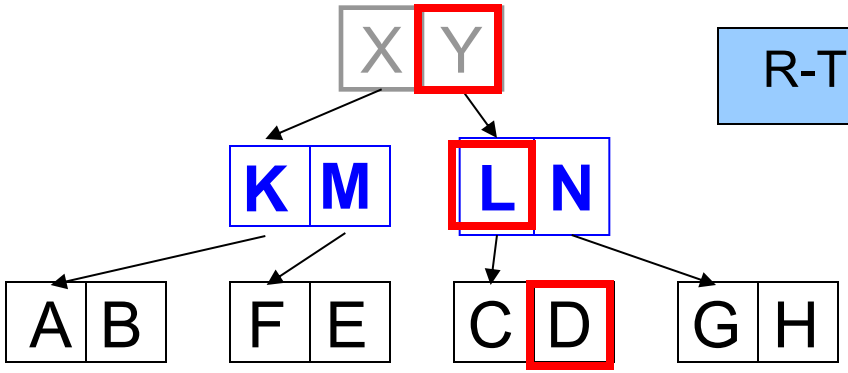
R-Tree Indexing

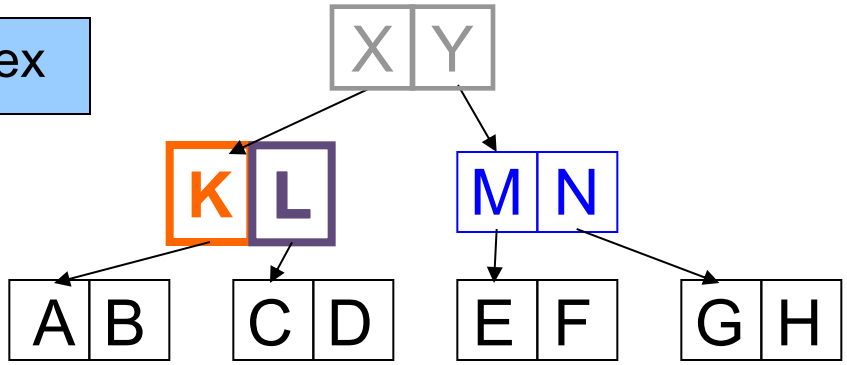


Minimum Bounding Rectangle (MBR)



R-Tree Index





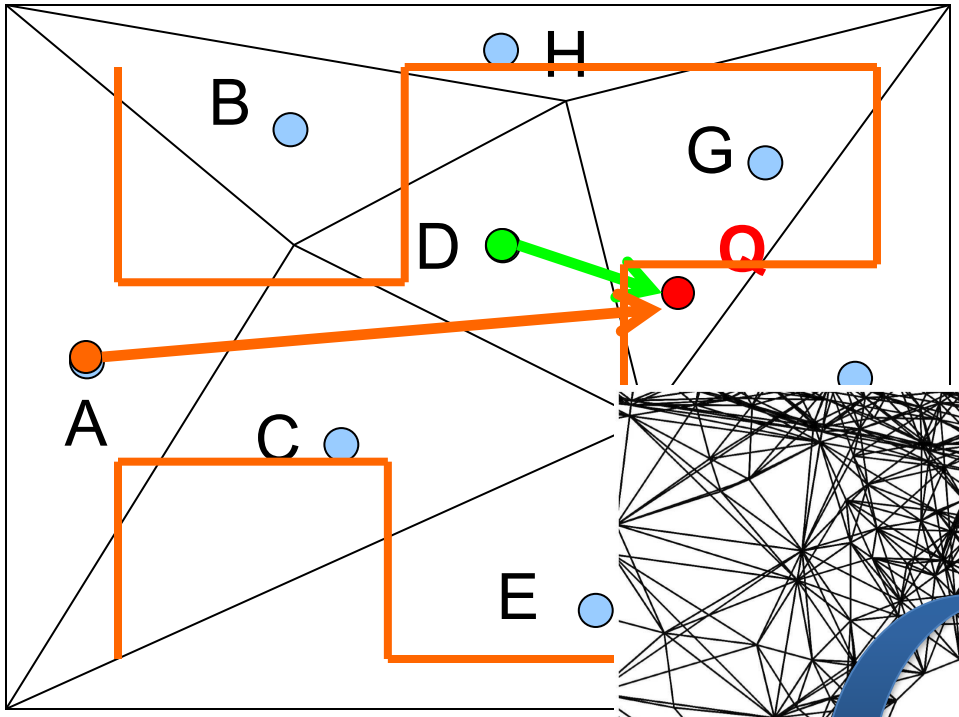
Disk page

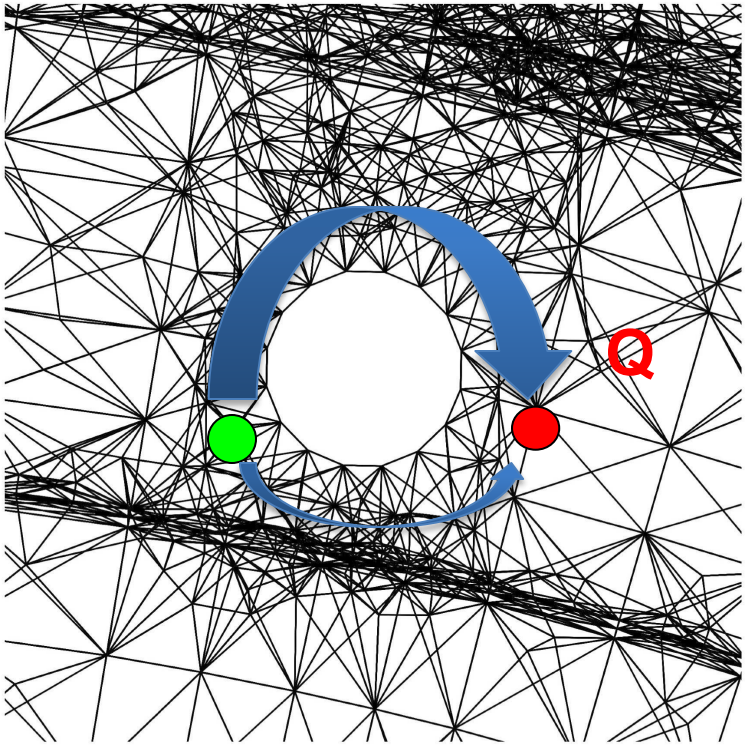
Tight mesh connectivity hurts performance

Solution: Fast Indexing of meshes

- Tetrahedral meshes
  - Index first the centers of the tetrahedra
  - Try to land somewhere near point of interest
  - Use triangulation walking to reach target
- Brain simulation
  - Index an overlay of the brain model
  - Use Voronoi tesselation to fill empty spaces
  - Enable triangulation walking to target

Directed Local Search





- Point queries
  - Step 1: Find a “nearby” element eff
  - Step 2: Follow path of adjacent ele
- Range queries are similar

Conclusions

- Data Management has undergone tremendous changes:
  - New requirements
  - New use cases
- Current needs of scientific applications still not served well
  - Spatial data
  - Large scale analysis