CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

Databases Beyond Tables

There‘s A System For Every Use-Case

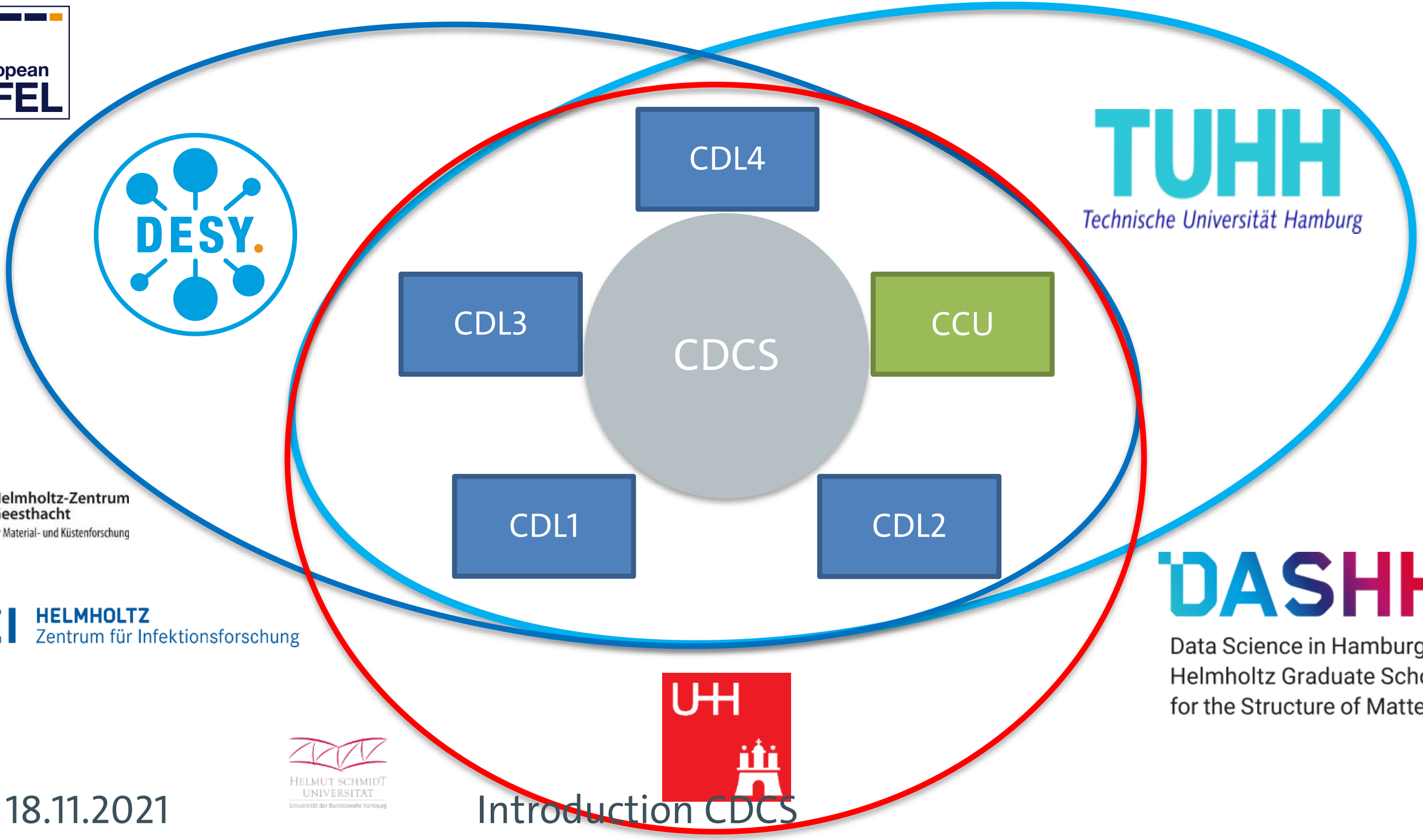
Annett Ungethüm, 06.01.2022

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

DESY.

TUHH
Technische Universität Hamburg-Harburg

CDCS Hamburg-X Project (BWFGB)



Associated Partner

HAW HAMBURG

Helmholtz-Zentrum Geesthacht
Zentrum für Material- und Küstenforschung

HZI HELMHOLTZ Zentrum für Infektionsforschung

Helmut Schmidt Universität
Universität der Bundeswehr Hamburg

DASHH
Data Science in Hamburg
Helmholtz Graduate School
for the Structure of Matter

CDCS Structure

UHH Project Coordinator

Prof. Dr. Matthias Rarey
UHH Computer Science
Spokesperson CDCS

DESY Project Coordinator

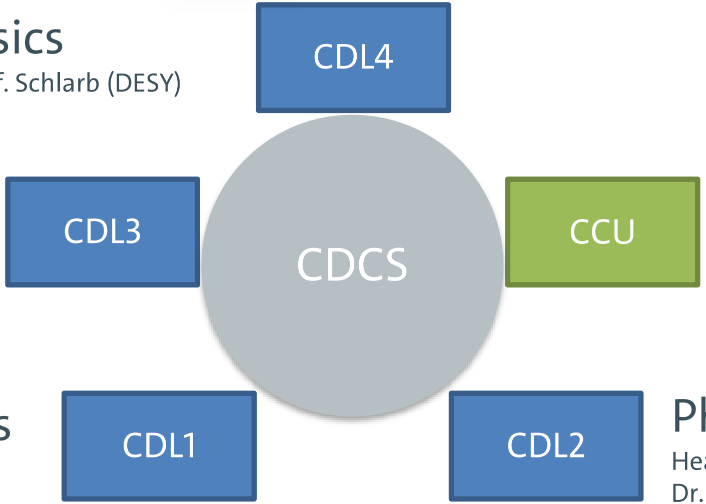
Prof. Dr. Nina Rohringer
DESY / UHH Physics

TUHH Project Coordinator

Prof. Dr. Sabine Le Borne
TUHH Mathematics

Accelerator Physics
Head: Prof. Fey (TUHH) / Prof. Schlarb (DESY)

Systems Biology
Head: Prof. Grünewald (UHH)/
Prof. Baumbach (UHH)



Computational Core Unit
Head: Prof. Rarey (UHH)/ Prof. Knopp (TUHH)

Astro & Particle Physics
Head: Prof. Schleper (DESY)/
Dr. Gaede (UHH)

Photon Science
Head: Dr. Barty (DESY)/
Dr. White (DESY)

CDCS and CDLs in Detail

CDL4
CDL3
CDCS
Miriam Döring
Thomas Merling
CCU
CDL1
CDL2

Accelerator Physics

Head: Prof. Fey (TUHH) / Prof. Schlarb (DESY)

Dr. Antonin Sulc

Dr. Ahmad Al-Zoubi

Systems Biology

Head: Prof. Grünewald (UHH)/
Prof. Baumbach (UHH)

Dr. Karen M.-Cantos

Dr. Khalique Newaz

Computational Core Unit

Head: Prof. Rarey (UHH)/ Prof. Knopp (TUHH)

Dr. Marie Tolkiehn
Scientific-administrative
Management

Dr. Florian Griese

Dr. Annett Ungethüm

Astro & Particle Physics

Head: Prof. Schleper (DESY)/
Prof. Gaede (UHH)

Dr. Patrick Connor

Dr. Janis Kummer

Dr. Lennart Rustige

Photon Science

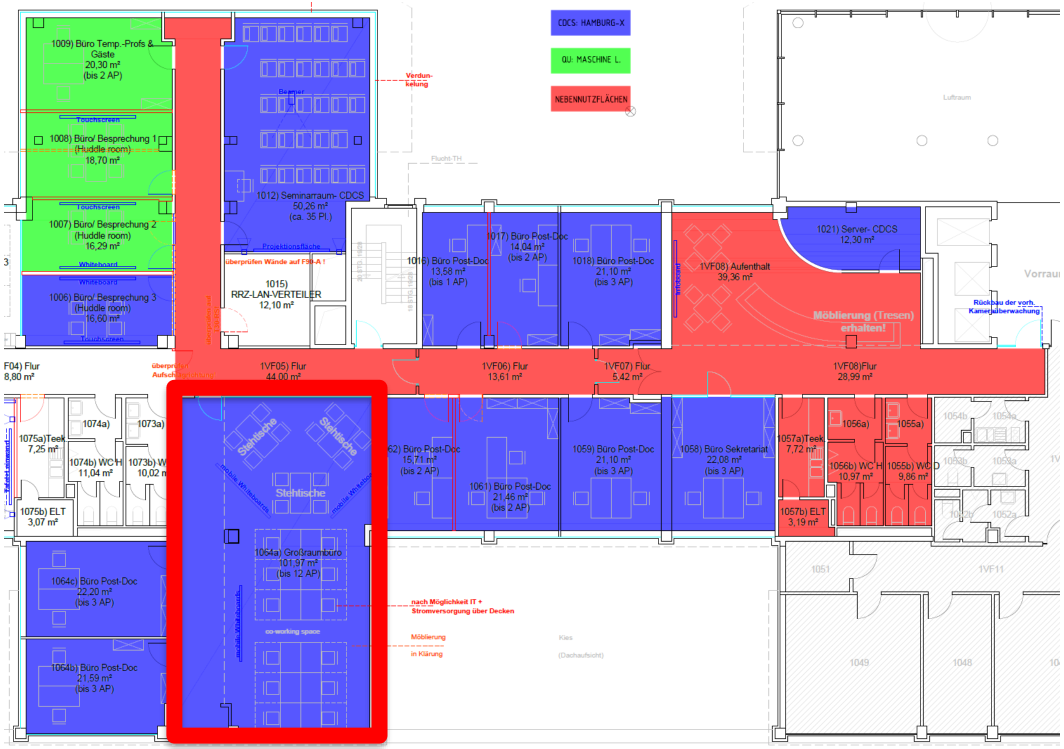
Head: Dr. Barty (DESY)/
Dr. White (DESY)

Dr. Mads Jakobsen

N.N

The CDCS Office Space

As a DASHH student you can get a transponder to the CDCS hot desk office space (room 1064)
Ask our secretary Miriam Döring:
miriam.doering@uni-hamburg.de



Data Science Thursdays: Database Timeline

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

Topic of the month: Databases

You might want to send us your questions in advance to get more sophisticated answers

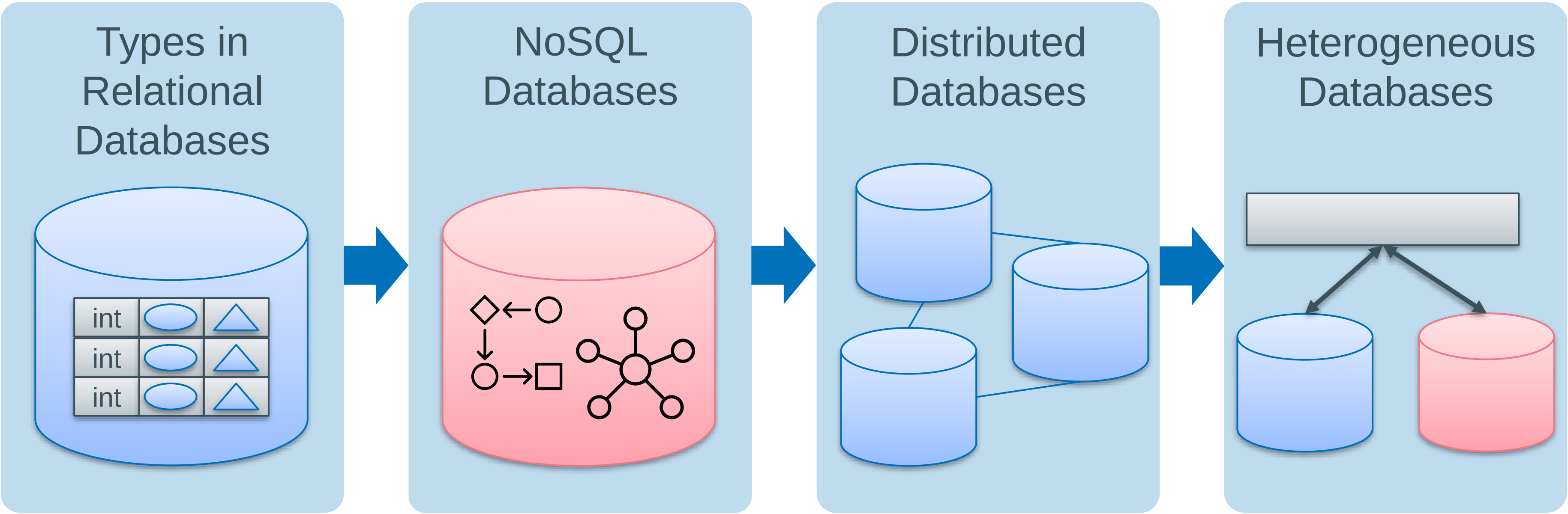
16 Dec 2021
Why and how should I use a database and why it is different from an excel sheet

Holidays!

06 Jan 2021
Relational DBs, document stores, key-value-stores: There's a system for every use-case

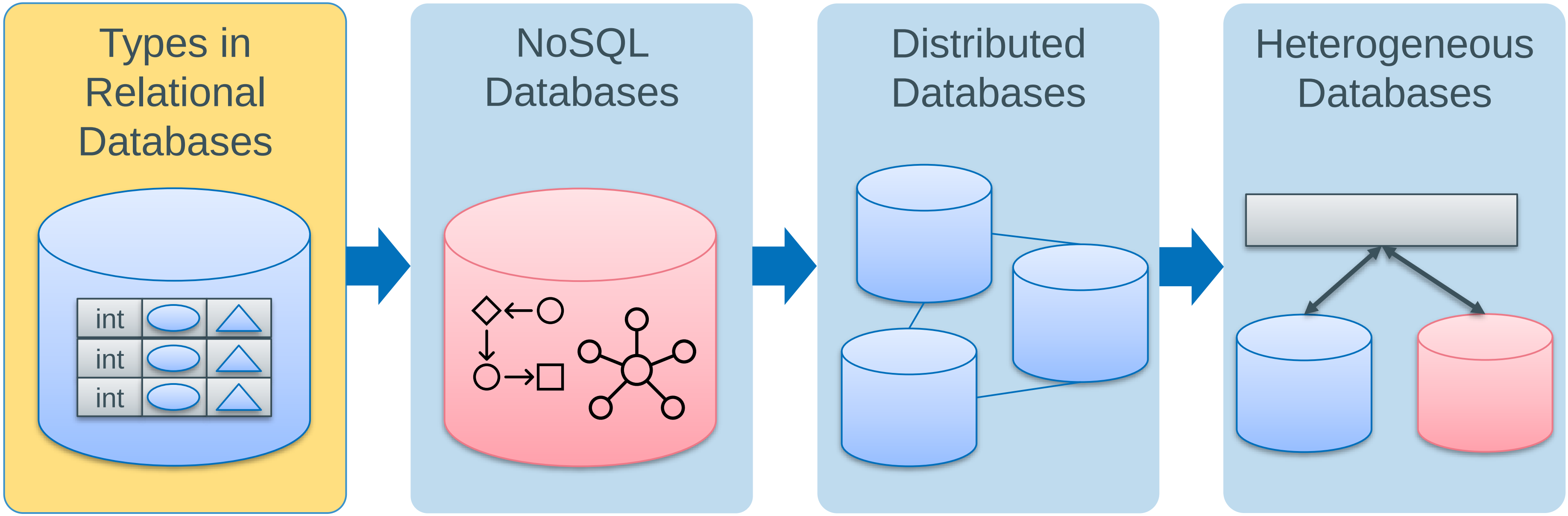
13 Jan 2021
Get your research data into a database!

Roadmap: The Database System Universe



Roadmap: The Database System Universe

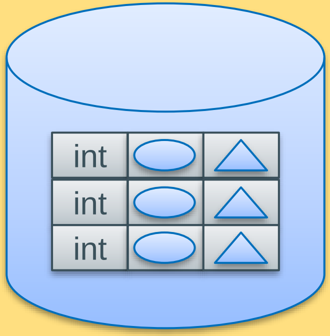
CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES



Roadmap: The Database System Universe

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

Types in Relational Databases



- Recap Relational Databases
- Binary Large Objects
- Multi-Dimensional Data

Recap Relational Databases

This is a **relation**, defined as *MensaMeals(Meal,Price)*

**That's why table based DBs are called relational Databases*

MensaMeals

Meal	Price
Pizza	6,50
Pasta	4,90
Pie	1,20
Potato Salad	5,80
Pannfisch	7,90

SQL is a query language for relational DBs, e.g.
SELECT * FROM MensaMeals WHERE PRICE <5;

Meal	Price
Pasta	4,90
Pie	1,20

Meal and *Price* are the **attributes** of the relation *MensaMeals*

- Each attribute has a type, e.g. integer, text/varchar/string, real
- Types here: *varchar* and *real*

This is a **tuple**, which belongs to the relation *MensaMeals*

Data is not always structured in tables containing plain numbers or text

Fancy data in relational databases (SQL DBs)

Standard types you may know:

CHAR, VARCHAR (aka TEXT or STRING), REAL, INTEGER, SMALLINT, BIGINT, FLOAT, DATE, TIME, INTERVAL,...

Standard types you may not know:

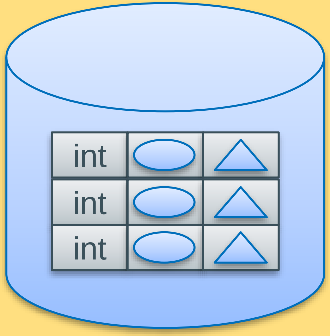
BINARY, BLOB (binary large object), MULTISET (collection of unordered values), ARRAY (array of another standard type), MDarray (multidimensional array, since 2019), JSON, ...

- Some systems support more types or alternative notations, e.g. TINYINT, MAP, LIST, ...
- Nesting of lists, structs and other composite types is supported by many systems

Roadmap: The Database System Universe

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

Types in Relational Databases

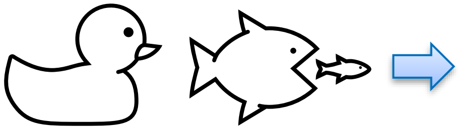


- Recap Relational Databases
- **Binary Large Objects**
- Multi-Dimensional Data

Binary large objects (Blobs)

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

- Arbitrary binary data (e.g. images)
- “large” is a nebulous word → it’s a bad idea to store a file with several GB of arbitrary binary data in a relational database
- Database does not know what your binary data represents
- Only the first x bytes are indexed
- Inserting data via command line might not be feasible → use the API
- Supported features vary heavily between systems → Client-Server systems usually come with a larger set of features



pictures

animal	picture
Duck	x0000…FFFF00FF…
Fish	x0000…FFFF0000…

Tasks:
1. Insert data from file
2. Return something, e.g. blob size
3. Match a pattern, e.g. 

Blob Features

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

<table>
<tr><th>Task</th><th>DuckDB (embedded, free)</th><th>MySQL (Client-Server, free)</th><th>Oracle (Client Server, $$$)</th></tr>
<tr><td>1
insert</td><td>Use undocumented functions of backend or read and construct query yourself

quick and dirty c++

//get size
std::ifstream infile;
infile.open("fish_small.bmp", std::ios::binary | std::ios::in | std::ios::ate);
auto size = infile.tellg();
infile.close();
//get data from file
char x[3000]; //make this large enough
infile.open("fish_small.bmp", std::ios::binary | std::ios::in);
infile.read(&(x[0]), size);
infile.close();
//create and run query
char first[9000] = "INSERT INTO pictures VALUES ('duck', '"; //41
char last[11] = "'::BLOB);"; //11
strcat(first, x);
strcat(first, last);
auto seq2 = con.Query(first);
seq2->Print();</td><td>• Same as DuckDB but with convenience function to escape special characters in strings (mysql_real_escape_string) OR
• Use LOAD_FILE(‘filename‘) → SQL extension</td><td>Oracle SQL extension can be used (PL/SQL)

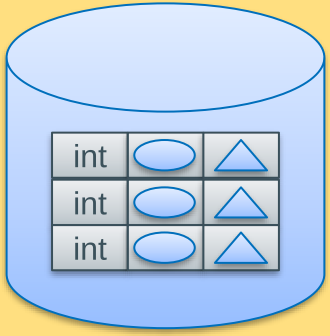
InFile BFILE := BFILENAME(‘directory', ‘fish_small.bmp’);
…
DBMS_LOB.LOADFROMFILE(
DEST_LOB => lnBlob,
SRC_LOB => lnFile,
AMOUNT => DBMS_LOB.GETLENGTH(lnFile));</td></tr>
<tr><td>2
return</td><td>Limited choice of functions → check for equality and count of bytes (=size)

SELECT animal, octet_length(picture)
FROM pictures

animal   octet_length(picture)
VARCHAR BIGINT
[Rows: 1]
duck   2826</td><td>Blobs mostly treated like byte strings
→ use string operations</td><td>Extensive selection of functions in dbms_lob package, e.g. getlength, get_storage_limit, compare,…</td></tr>
<tr><td>3
match</td><td>No support
→ Return blob and do it on your own</td><td>LOCATE (substring, string)</td><td>Multiple ways to do this, e.g.:
• utl_raw .cast_to_raw () → get raw data format
• dbms_lob.instr() → compare content (alternative for LIKE)</td></tr>
</table>

Roadmap: The Database System Universe

Types in Relational Databases



- Recap Relational Databases
- Binary Large Objects
- Multi-Dimensional Data

Multi-Dimensional Data

Nesting of lists, maps, arrays is supported by most systems

Example using DuckDB:

Integer type

List of integers

Nested list of integers (depth=2)

```
CREATE TABLE mylist (id INT, intlist INT[], nestedlist INT[][]);
INSERT INTO mylist VALUES (0, [0,1,2,3,4,5], [[0,1],[2,3],[4,5]]);
```

```
SELECT * FROM mylist;
```

id	intlist	nestedlist
0	[0, 1, 2, 3, 4, 5]	[[0, 1], [2, 3], [4, 5]]

```
INSERT INTO mylist VALUES (1, range(5,10), [range(5,10),range(11,15)]);
INSERT INTO mylist VALUES (2, generate_series(5,10),[generate_series(5,10),
                                                     generate_series(11,15)]);
```

```
SELECT * FROM mylist;
```

id	intlist	nestedlist
0	[0, 1, 2, 3, 4, 5]	[[0, 1], [2, 3], [4, 5]]
1	[5, 6, 7, 8, 9]	[[5, 6, 7, 8, 9], [11, 12, 13, 14]]
2	[5, 6, 7, 8, 9, 10]	[[5, 6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]

Working With Multidimensional Data

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

WHERE-clause and functions treat the whole element as one unit, e.g. the whole nested list

id	intlist	nestedlist
0	[0, 1, 2, 3, 4, 5]	[[0, 1], [2, 3], [4, 5]]
1	[5, 6, 7, 8, 9]	[[5, 6, 7, 8, 9], [11, 12, 13, 14]]
2	[5, 6, 7, 8, 9, 10]	[[5, 6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]

select * from mylist where nestedlist=2; → Throws an error because elements are not literals

select * from mylist where nestedlist=[[0,1],[2,3],[4,5]]; →

id	intlist	nestedlist
0	[0, 1, 2, 3, 4, 5]	[[0, 1], [2, 3], [4, 5]]

select * from mylist where intlist[2]=7; →

id	intlist	nestedlist
1	[5, 6, 7, 8, 9]	[[5, 6, 7, 8, 9], [11, 12, 13, 14]]
2	[5, 6, 7, 8, 9, 10]	[[5, 6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]

select * from mylist where nestedlist[1]=[11,12,13,14]; →

id	intlist	nestedlist
1	[5, 6, 7, 8, 9]	[[5, 6, 7, 8, 9], [11, 12, 13, 14]]

select * from mylist where nestedlist[1][1]=12; →

id	intlist	nestedlist
1	[5, 6, 7, 8, 9]	[[5, 6, 7, 8, 9], [11, 12, 13, 14]]
2	[5, 6, 7, 8, 9, 10]	[[5, 6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]

More Fun With Nested Lists

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

Unnest nested structures

ID needed to know which tuple the unnested elements belong to

SELECT * FROM (SELECT *unnest* (nestedlist), id FROM mylist) foo;

unnest(nestedlist)	id
[0, 1]	0
[2, 3]	0
[4, 5]	0
[5, 6, 7, 8, 9]	1
[11, 12, 13, 14]	1
[5, 6, 7, 8, 9, 10]	2
[11, 12, 13, 14, 15]	2

Long enough that a view starts making sense (see dec 16, slide 28)

Search in an unnested structure

SELECT * FROM (SELECT *unnest* (nestedlist) AS nr, id FROM mylist) foo WHERE foo.nr=[0,1];

nr	id
[0, 1]	0

Get information about array, e.g. length

SELECT *len* (nestedlist) FROM mylist;

len(nestedlist)
3
2
2

SELECT *len* (nestedlist[0]) FROM mylist;

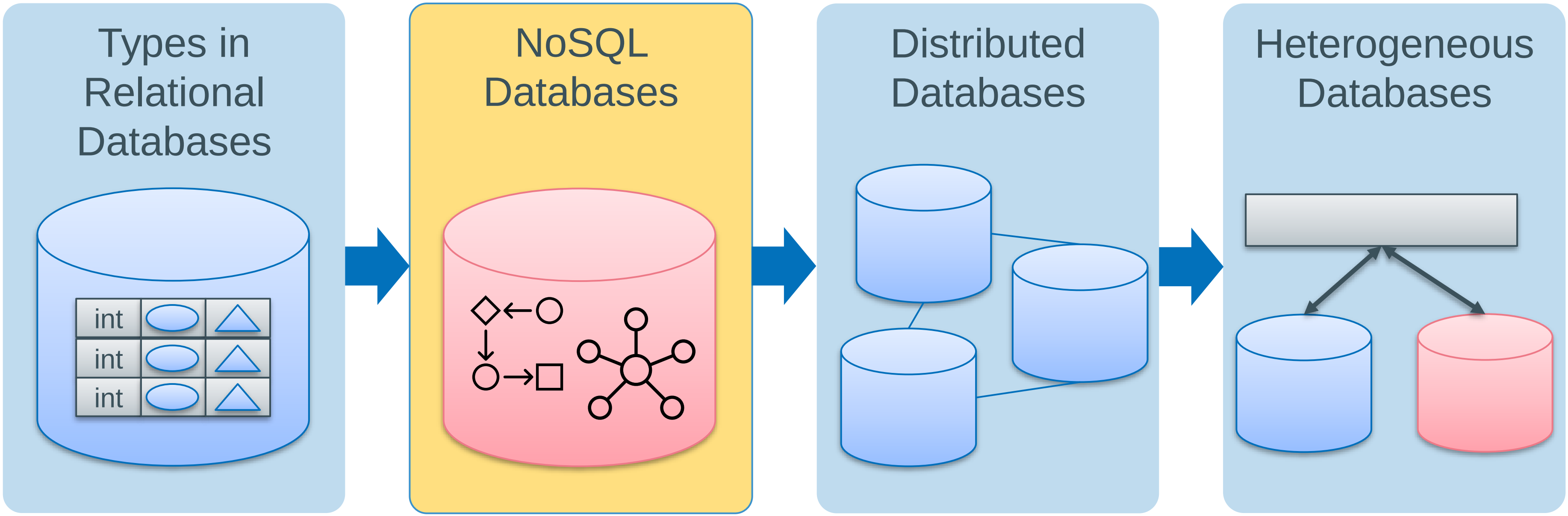
len(ARRAY_EXTRACT(nestedlist, 0))
2
5
6

Element-wise math

SELECT *sin* (*unnest*(intlist)) FROM mylist;

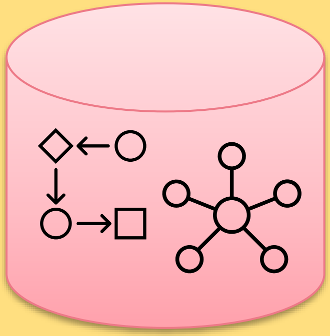
sin(unnest(intlist))
0.0
0.8414709848078965
0.9092974268256817
0.1411200080598672
-0.7568024953079282
-0.9589242746631385
-0.9589242746631385
-0.27941549819892586
0.6569865987187891
0.9893582466233818
0.4121184852417566
-0.9589242746631385
-0.27941549819892586
0.6569865987187891
0.9893582466233818
0.4121184852417566
-0.5440211108893698

Roadmap: The Database System Universe



Roadmap: The Database System Universe

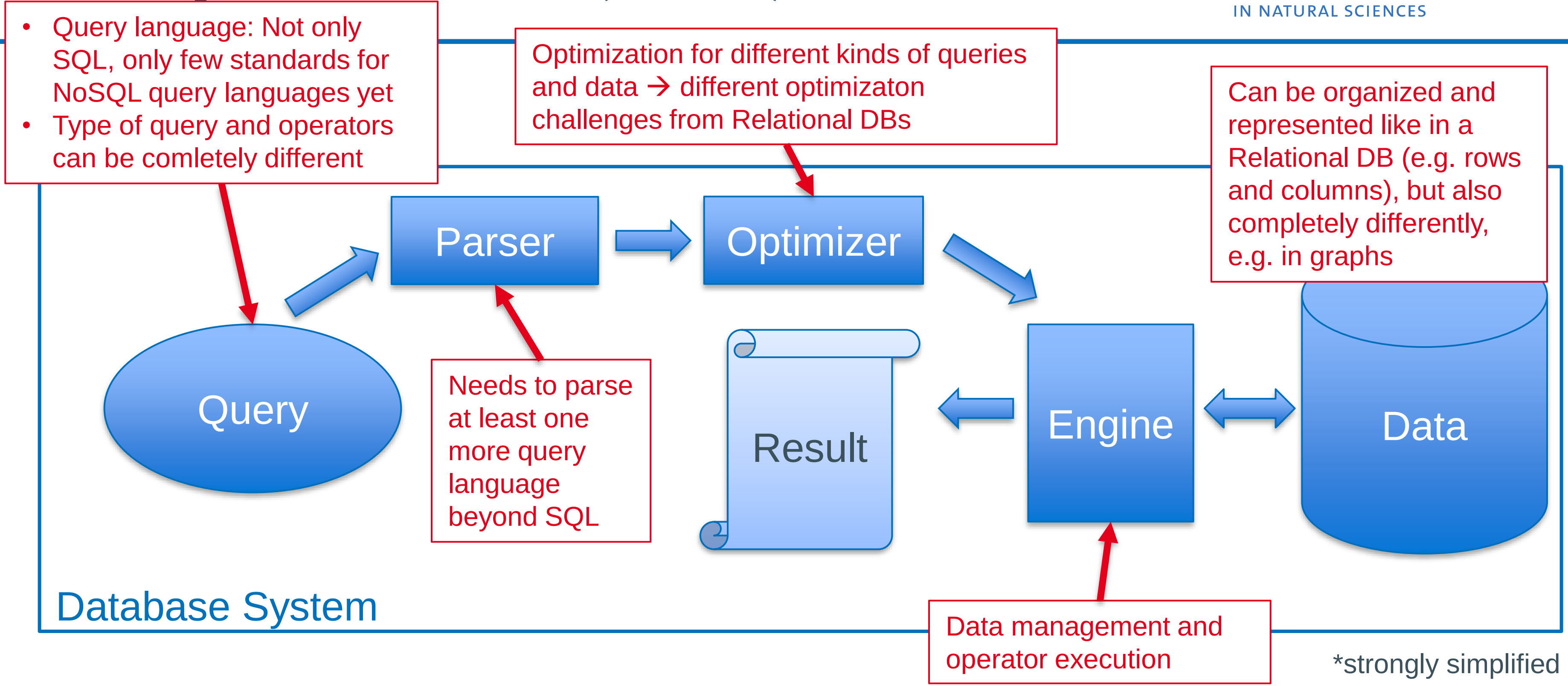
NoSQL
Databases



- **Not Only SQL Databases**
- Graph Databases
- Key-Value Stores
- Document Stores
- Application-Specific Database Systems

Not only SQL Databases (NoSQL)

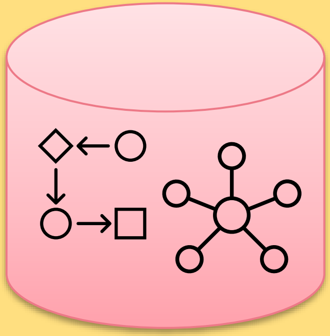
CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES



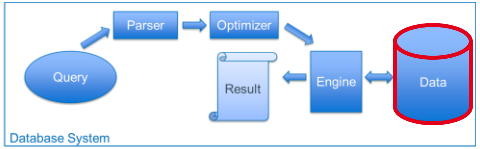
*strongly simplified

Roadmap: The Database System Universe

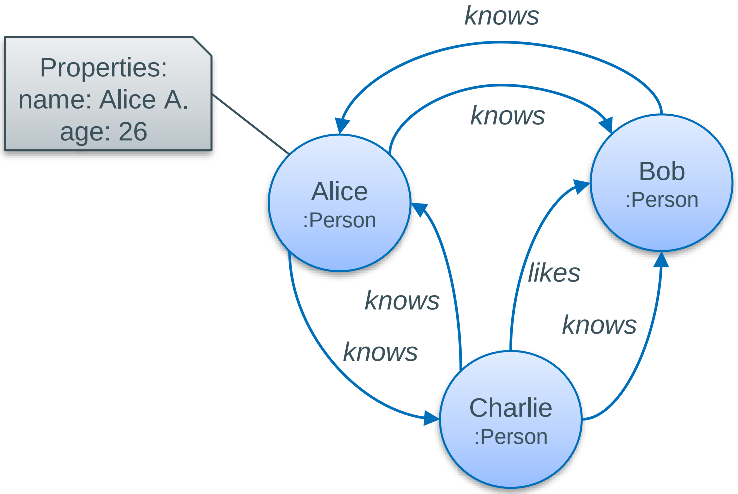
NoSQL
Databases



- **Not Only SQL** Databases
- Graph Databases
- Key-Value Stores
- Document Stores
- Application-Specific Database Systems

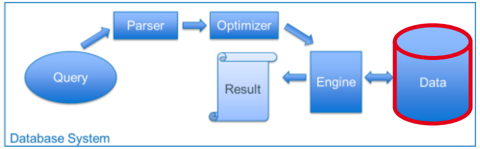


Most common logical representation: Property Graph Model

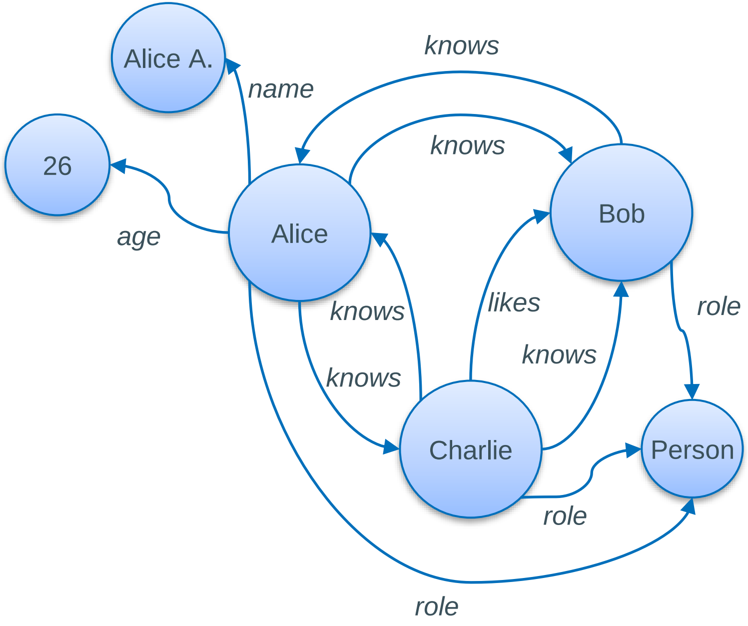


Parts of the Property Graph Model

- Nodes (here: Alice, Bob, Charlie)
- Node properties/attributes (here: Name, Age)
- Node Label: describe the role of a node (here: Person)
- Directed and named edges between nodes (here: knows, likes)
- Edge properties/attributes (here: none)



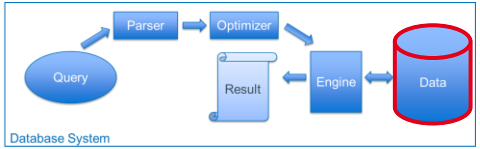
Alternative representation: RDF (Resource Description Framework) → Triples



Parts of the RDF Model

- Source – Predicate – Object
  e.g. Charlie – knows – Alice
- No Properties → properties have to be expressed via triples,
  e.g. Alice – age – 26
- Each node and edge is just a unique label

Graph Databases



Node and Edge creation depends on System and preferred query language

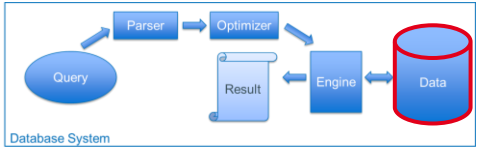
- SPARQL uses INSERT DATA statement

```
INSERT DATA{   <eve>   role < Person>;
                       <name> "Eve V.";
                       <age> 45; }
```

- Cypher uses CREATE statement

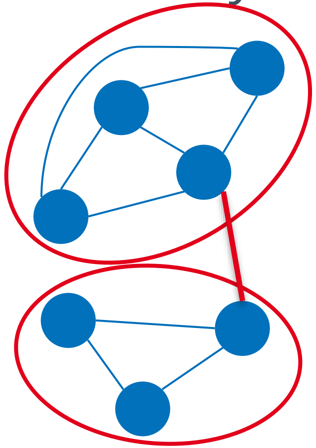
```
CREATE (eve:Person {
      name: „Eve V.“,
      age: 45})
```

> Physical representation in memory can be very different, e.g. as XML, in turtle syntax, a table in main memory, as a relational database (in all its variants),…
> →Neo4j uses different databases for nodes, properties, edges, and indexes



Partitioning of graphs when they become too big for one system

→ Sharding is an NP-hard problem → You won‘t always find the perfect solution, but you can find good solutions



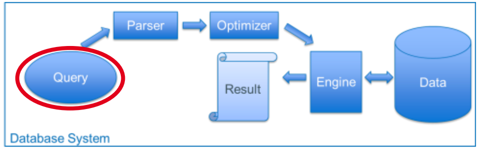
Questions to answer

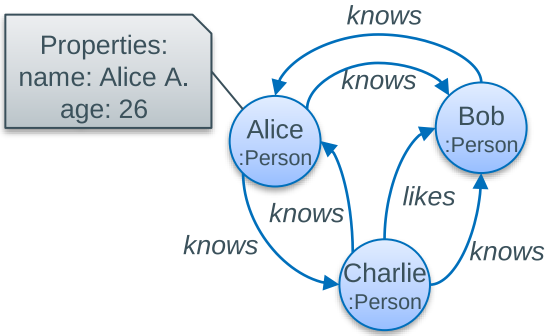
- Where to cut the graph? (equal size, balance load, minimize inter-node edges, keep subgraphs at same node,…)
- How to cut the graph? (Through edges or through vertices)
- How to store the data, i.e. in which format

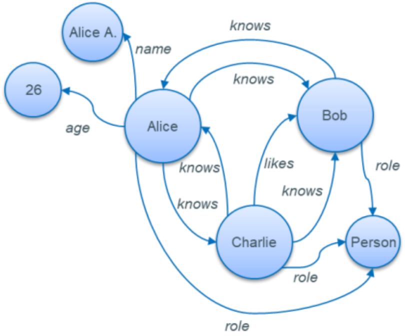
Trivia

Sounds familiar? → Blockchains are basically graphs. The blockchains of some crypto currencies have become really large, but a node still has to hold and synchronize the whole graph. → Sharding with the added necessary security layer is the next big challenge for cryptos

Graph Databases







Query languages:

- Cypher (Neo4j)
- SPARQL (standardized by W3C)
- DSLs (e.g. Green-Marl), ...

Example: Find all people Alice knows

Cypher: MATCH (:Person {name: ‚Alice A.‘}) –[:KNOWS]->(p:Person)
Return p

SPARQL: PREFIX foaf: http://xmlns.com/foaf/0.1/
SELECT ?name WHERE {
?p foaf:name “Alice A.“ ;
foaf:knows ?o .
?o foaf:name ?name . }

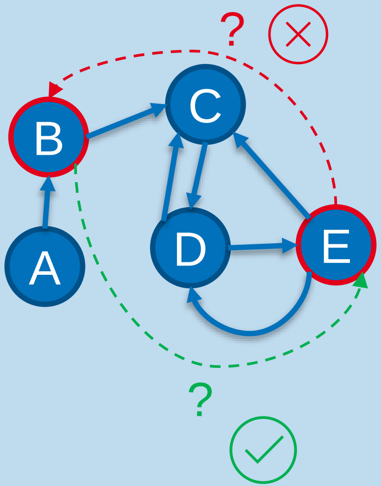
foaf:
- An ontology definition
- Short for friend of a friend
- Intentionally ignored in triple creation for comprehensibility

Different Queries in Graph DBs (Selection)

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

Reachability

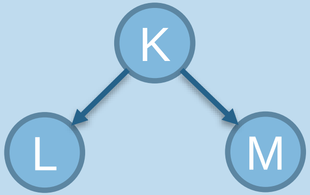
Can a node be reached from another node?



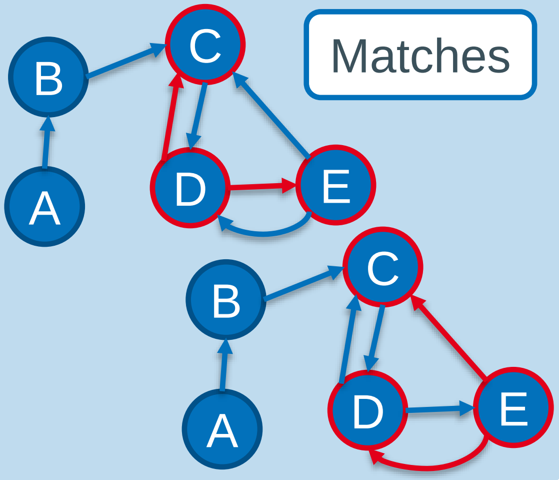
Pattern matching

Find occurrences of a pattern in a graph.

Pattern

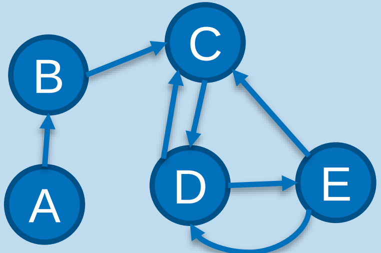


Matches



Betweenness centrality

Score based on number/weight of shortest paths through each node

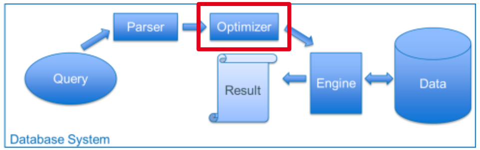


~~AB: A->B~~
AC: ~~A~~->B->~~C~~
AD: ~~A~~->B->C->~~D~~
AE: ~~A~~->B->C->D->~~E~~
~~BC: B->C~~
BD: ~~B~~->C->~~D~~
BE: ~~B~~->C->D->~~E~~
~~CD: C->D~~
CE: ~~C~~->D->~~E~~
~~DC: D->C~~
~~DE: D->E~~
~~EC: E->C~~
~~ED: E->D~~

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

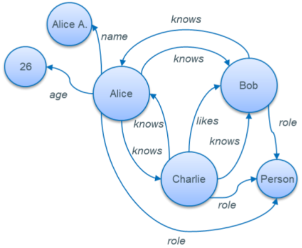
~~A~~
B: 3
$\boxed{\text{C: 4}}$
D: 3
~~E~~

Graph Databases



CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

Common optimization goal with Relational Databases: reduce runtime
Common approach: Reduce work to do as early as possible
Different challenge: in relational DBs, number of results is reduced as early as possible → in Graph DBs communication between vertexes/edges, nodes or clusters is reduced as early as possible



Query: Who likes Bob?
Query Plan: Go through all triples or edges and find one with „likes“ as poperty/label and „Bob“ as objective

Performance Bottleneck:
Broadcast the query to all triples

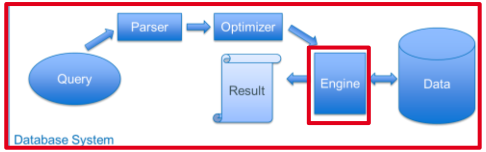
Possible optimizations:

1. Create inverted edges
2. Create and use an index on Bob‘s incoming edges
3. Store entities with many connections close to each other (e.g. on the same node)
4. …

Other common optimizations:

- Breadth First Search (BFS) or Depth First Search (DFS)
- Selection of starting node(s)
- Degree of parallelism
- …

Graph Databases



CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

Stand alone engines implementing a DSL, e.g. GreenMarl → no or very limited query optimization, requires additonal compiler to compile query

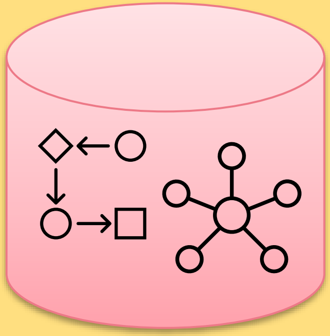
- Can (relatively) easily be integrated into own project, i.e. no system installation
- Rollout of own Software: No dependency on a fully fledged (potentially expensive) database system

Full Graph Database System, e.g. Neo4j, Pregel,...

- All-in-one solution including optimization
- May require root to install
- Can become expensive
- Licensing often more restrictive

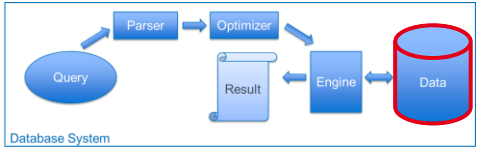
Roadmap: The Database System Universe

NoSQL
Databases



- **Not Only SQL** Databases
- Graph Databases
- Key-Value Stores
- Document Stores
- Application-Specific Database Systems

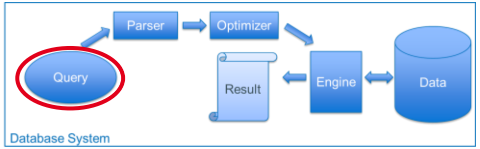
Key-Value Stores



- Simple system for storing and retrieving (key,value)-pairs
- Extensions: sorting keys, two-dimensional (key,value)-pairs (aka wide-column-stores),...
- Additional structures needed to find data fast, e.g. index on keys (e.g. as a tree structure), filters (e.g. bloom filters), ...
- Often used as embedded database
- Data can be organized write-optimized (as they are received) **or** read-optimized (in an organized structure, e.g. a tree or a sorted linked list)

→Data is often written into a write-optimized store and later migrated into a read optimized-store, e.g. when the system load is low

Key-Value Stores



CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

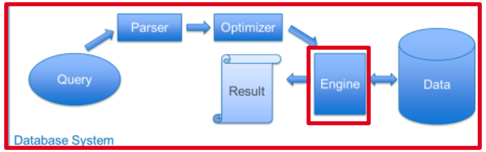
- 3 types of queries: put (add new pair), get (retrieve a pair), delete
- Query language depends on system, usually there is a system-specific API

Example: Memcached via telnet → It has come a long way since ist time as a simple caching tool

Add a new KV-pair
set AgeAlice 0 120 1 [Press Enter]
26 [Press Enter]
Retrieve a KV-pair
get AgeAlice
Delete a KV-pair
delete AgeAlice

Syntax of set:
set KEY META_DATA
EXPIRATION_TIME_IN_S
LENGTH_IN_BYTES
[Press Enter]
VALUE
[Press Enter]

Key-Value Stores



Query execution = Lookup of a key **or** add a key **or** remove a key

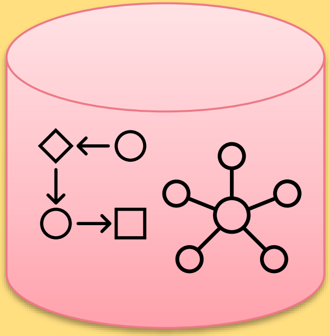
→ Stand-alone engine without optimization layer is not useful

→ Performance of operations depends on used storage layout and index structure

Example systems:

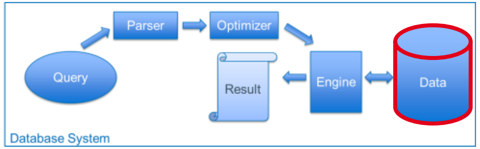
- Memcached (https://www.memcached.org/, simple, open source, many supported languages, e.g. C/C++, Java, Python, Perl, Lisp,...)
- Redis (https://redis.io/, open source, many supported languages, e.g. C/C++, C#, Python, R, VB, Haskell, Prolog, Scala...)

NoSQL
Databases



- **Not Only SQL** Databases
- Graph Databases
- Key-Value Stores
- Document Stores
- Application-Specific Database Systems

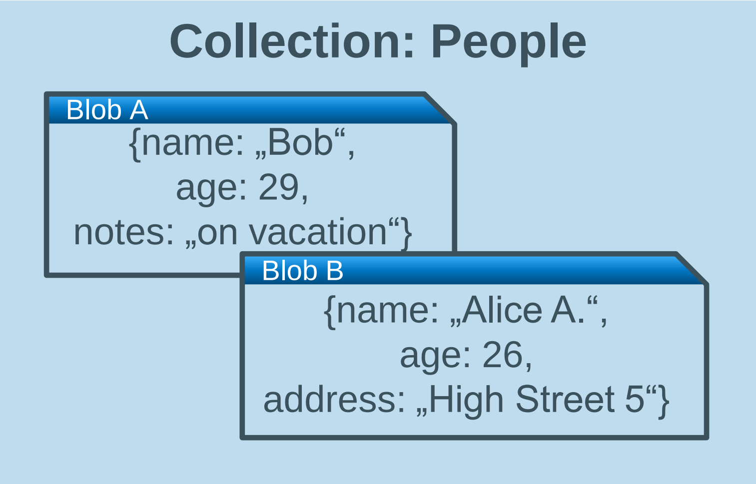
Document Stores



Document stores are fancy key-value-stores:

→ Values are blobs of data

→ Keys are usually assigned automatically



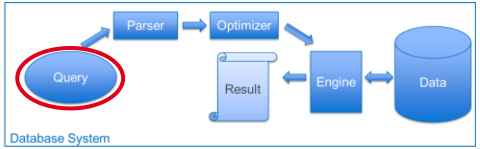
Collection of semi-structured data (e.g. JSON, XML)
→ Data is structured within each document
→ Structure can differ between documents

Insert data with MongoDB

Create an empty collection with the name ‚people‘: *use people*

Insert data into collection people:
db.people.insert({name: „Bob“, age: 29, notes: „on vacation“})

Document Stores



CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

Query language depends on system, e.g. SPARQL, MongoDB Query Language (MQL), XQuery (for xml documents), DSLs, …

Example: MQL

Show active collection:

db

Show all documents in ‘people’ collection:

db.people.find({})

How many documents are in the ‘people’ collection?

db.people.find({}).count()

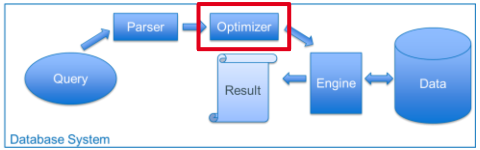
Show all documents in the ‘people’ collection where the name is ‘Bob’:

db.people.find({“name”: “Bob”})

Sort all people called ‘Bob’ by their age in descending order:

db.people.find({“name”: “Bob”}).sort({age: -1})

Document Stores



CDCS
CENTER FOR DATA AND COMPUTING IN NATURAL SCIENCES

- There are the usual optimizations (index usage, choice of operator implementation, ...)
- And there is a speciality of systems supporting full text search: the Inverted Index

Example: „This is a guy called Charlie. He knows this other guy called Bob."

position	word
1	This
2	is
3	a
4	guy
5	called
6	Charlie
7	He
8	knows
9	this
10	other
11	guy
12	called
13	Bob

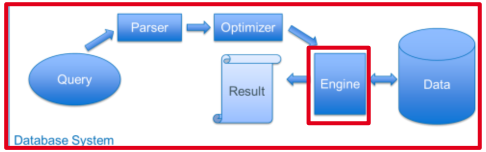
Normal index
Mapping **to** the content of a document
Here: position → word

Inverted index
Mapping **from** the content of a document (from words, sentences, terms, whole documents,…)
Here: word → position

word	position
This	1, 9
is	2
a	3
guy	4, 11
called	5, 12
Charlie	6
He	7
knows	8
other	10
Bob	13

- Not a new idea → oldest papers I found are from the 80s
- Progress during the last decades: compressed versions, support for different data types, inverted multi-indexes,…
- Data science is still slow when it comes to adaptation, e.g. "Real-time structural motif searching in proteins using an inverted index strategy", Bittrich et al., 2020

Document Stores



Usually rolled out as a full system or an extension for a relational database

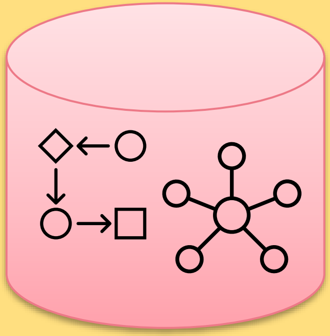
Most popular stand-alone document store: MongoDB

Some other systems:

- Microsoft Azure Cosmos DB and Amazon DynamoDB ($$$, supports JSON documents, cloud-only, supports other models, e.g. key-value, weird combination of table and document concept)
- Couchbase (open source, claims to scale better than MongoDB for large systems and many users)
- Oracle NoSQL ($$$$$, supports other models, does not require a cloud)

Roadmap: The Database System Universe

NoSQL
Databases



- **N**ot **O**nly **SQL** Databases
- Graph Databases
- Key-Value Stores
- Document Stores
- Application-Specific Database Systems

Application-Specific Database Systems

Example 1: SciDB

- Focuses on life sciences and healthcare
- Manages data as multidimensional arrays stored in columns
- Comes with ist own query language, e.g. *scan* instead of *select* *
- Used to be open source, now you have to contact the company for them to install it on your vm (might not be free)
- Initial system paper: https://ieeexplore.ieee.org/document/6461866

Example 2: Oracles Spatial Database

- Focuses on geospatial data and according tools, e.g. mapping services
- Part of Oracle‘s converged DB

→ Like we learned last time, Oracle can deal with different types of data, but it comes at a high cost

Application-Specific Database Systems

Chances are there is already a system doing exactly what you need
→ Try to get funding to pay for it
OR
→ Read the system paper so you do not have to invent everything by yourself

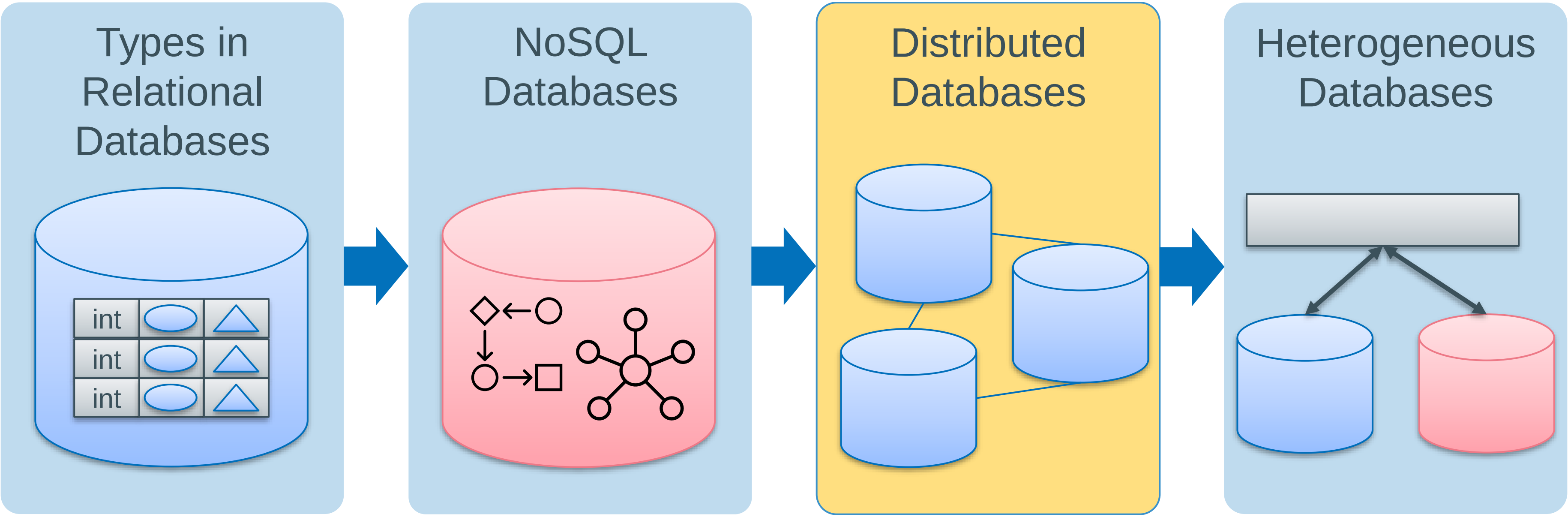
Many systems are forked from or inspired by open source and/or free systems.
Try them before starting from scratch.

More specialized DBs

- Time series: InfluxDB (https://github.com/influxdata/influxdb, https://www.influxdata.com/)
- Search engine: Elasticsearch (https://github.com/elastic/elasticsearch, www.elastic.co)
- Spatial data:
  - Postgis (https://postgis.net/) → Extension for PostgreSQL
  - SpatiaLite (https://www.gaia-gis.it/) → Extension for SQLite (Postgis shows better benchmark results)

Roadmap: The Database System Universe

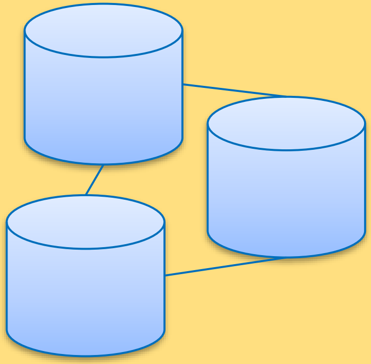
CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES



Roadmap: The Database System Universe

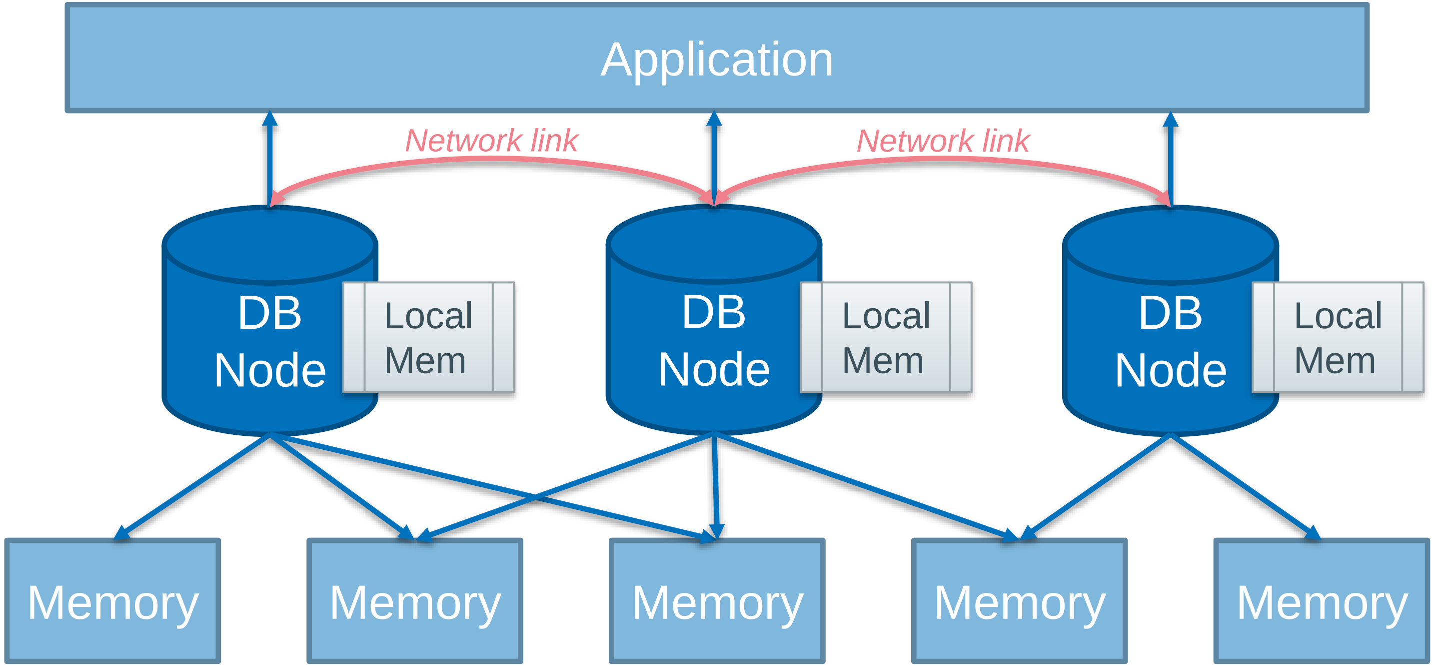
CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

Distributed Databases



- **Shared Disk and Shared Nothing**
- Cloud Databases
- Federated Databases
- Best Practices

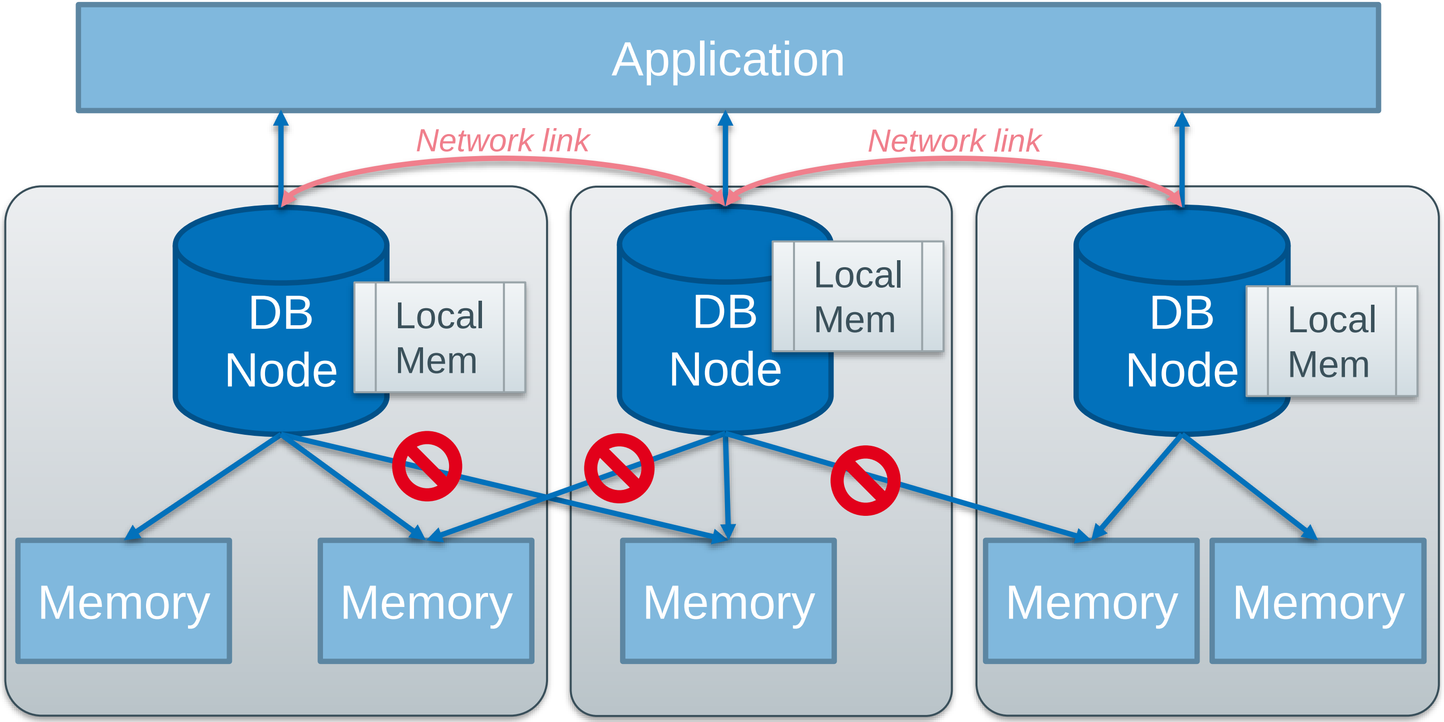
Distributed Databases: Shared Disc



Shared (Disk) Cluster

- Distributed Database
- One or more servers have equal access to memory (e.g. to discs)
- Servers don‘t share their own memory

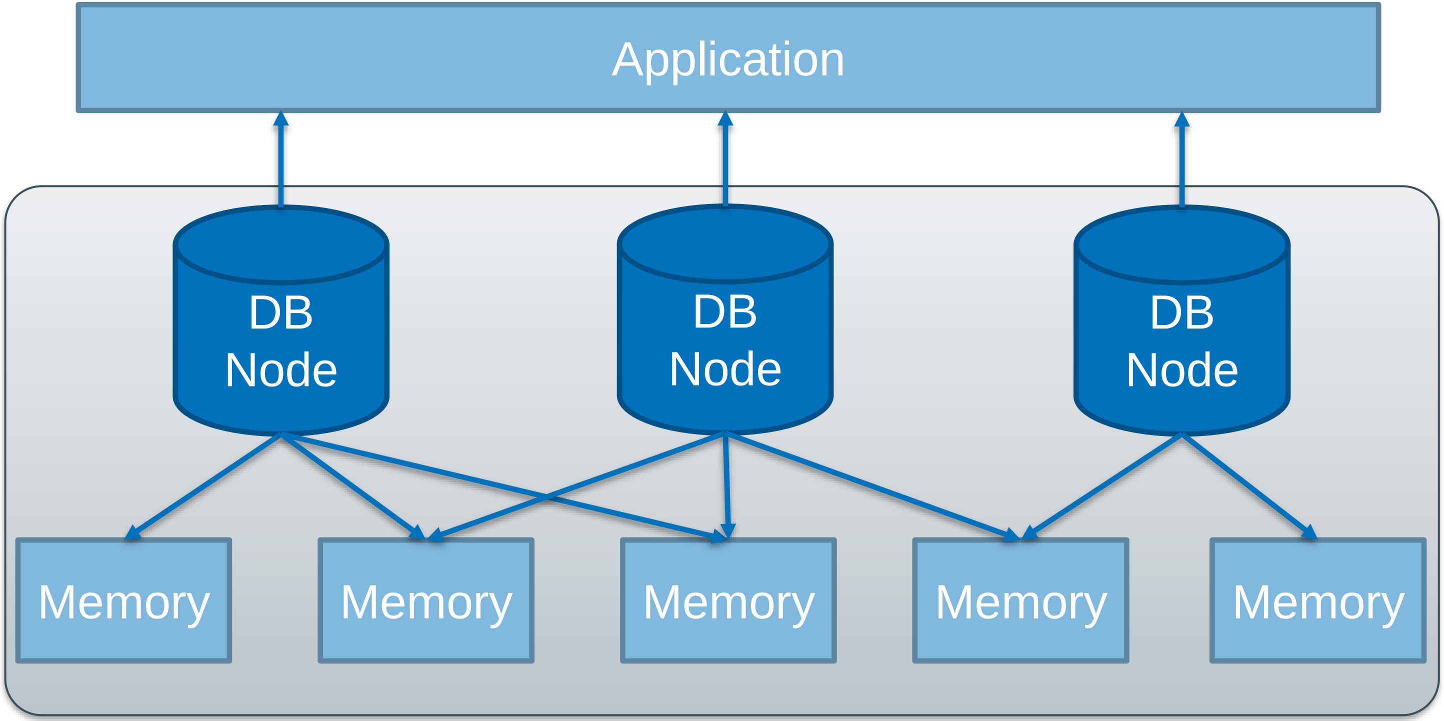
Distributed Databases: Shared Nothing



Shared Nothing

- Distributed Database
- Each node has exclusive access to its memory
- Servers don‘t share their own memory

Distributed Databases: Share Everything



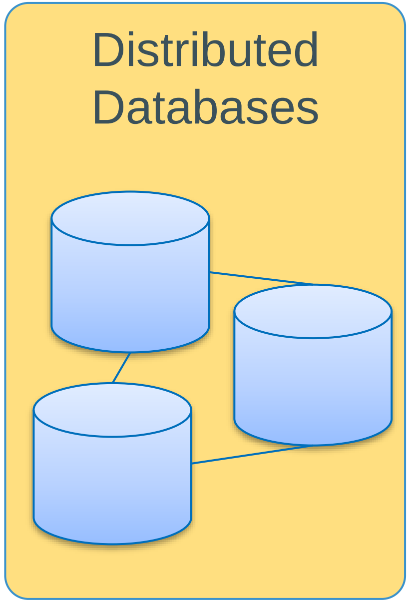
Shared Everything

- One big system instead of multiple (small) systems → scale-up instead od scale-out
- Disk and main memory is shared (NUMA system)

To Share Or Not To Share

	Scale-out		Scale-up
	Shared Disk	**Shared Nothing**	**Share Everything**
Advantages	- Robust in case of node failure (discs can be accessed by another node) - Usually easy to set up (if there is already a shared file system)	- Robust in case of disk failure (frequently used data can be replicated across nodes) - No distributed locking necessary - High performance if query is executed on node where most of the data is	- Comes for free with many systems → no additional setup - Faster than accessing remote memory
Disadvantages	- Simultaneous disk access is a potential bottleneck - Overhead to maintain cache consistency - Requires complex locking mechanisms for updates	- Partitioning (sharding) of data needs additional care to get optimal performance (store data where it is processed)	- Limited scaling possibilities - Hardware for large systems becomes expensive
Example System(s)	- Add-on/Feature of data management systems, e.g. Microsoft Azure shared disk, Oracle RAC - Hybrid Systems (SD & SN), e.g. Snowflake	Couchbase*, MariaDB SkySQL,…	Each system working with NUMA architectures, e.g. MonetDB, PostgreSQL, SQL Server,…
Comments	Requires shared file system	Can be used for backup servers if full replication is enabled	Usually not noticed by the user

*Couchbase white paper: http://info.couchbase.com/rs/northscale/images/Couchbase_Architectural_Document_Whitepaper_2015.pdf



- Shared Disk and Shared Nothing
- **Cloud Databases**
- Federated Databases
- Best Practices

Cloud DBs

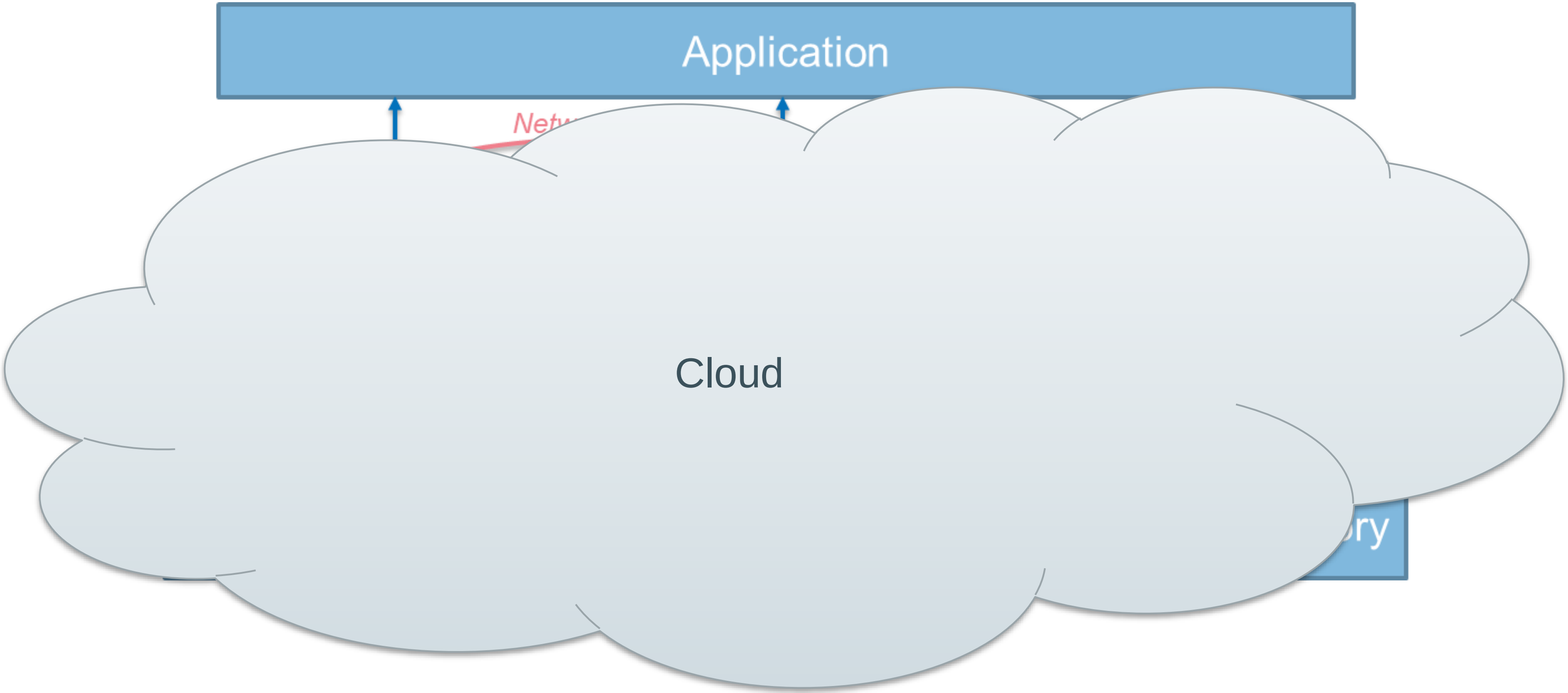
Cloud storages can be regarded as a special kind of shared disc storages with user management and some additional features:

- The cloud provider is an additional layer between the database and the application
  - Advantages: Cloud provider has to fix bugs, usually high availability due to large system with redundancies
  - Disadvantage: usually no physical access, provider must be trusted with security issues
- Cloud storage is usually remote whereas shared discs can be local
- A lot of marketing

Edge cloud → process data close to its location (at the device of the end user) and make it available in the cloud and/or outsource work to the cloud

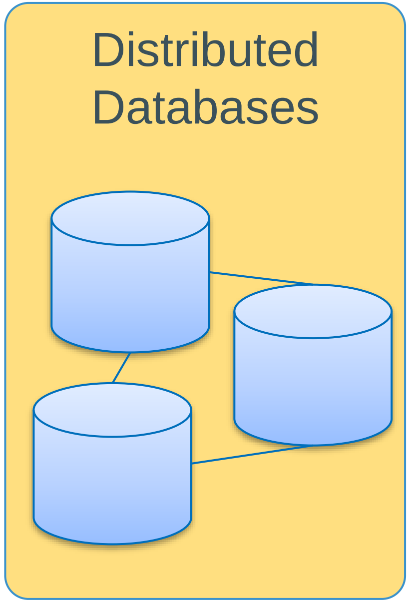
Cloud DBs

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES



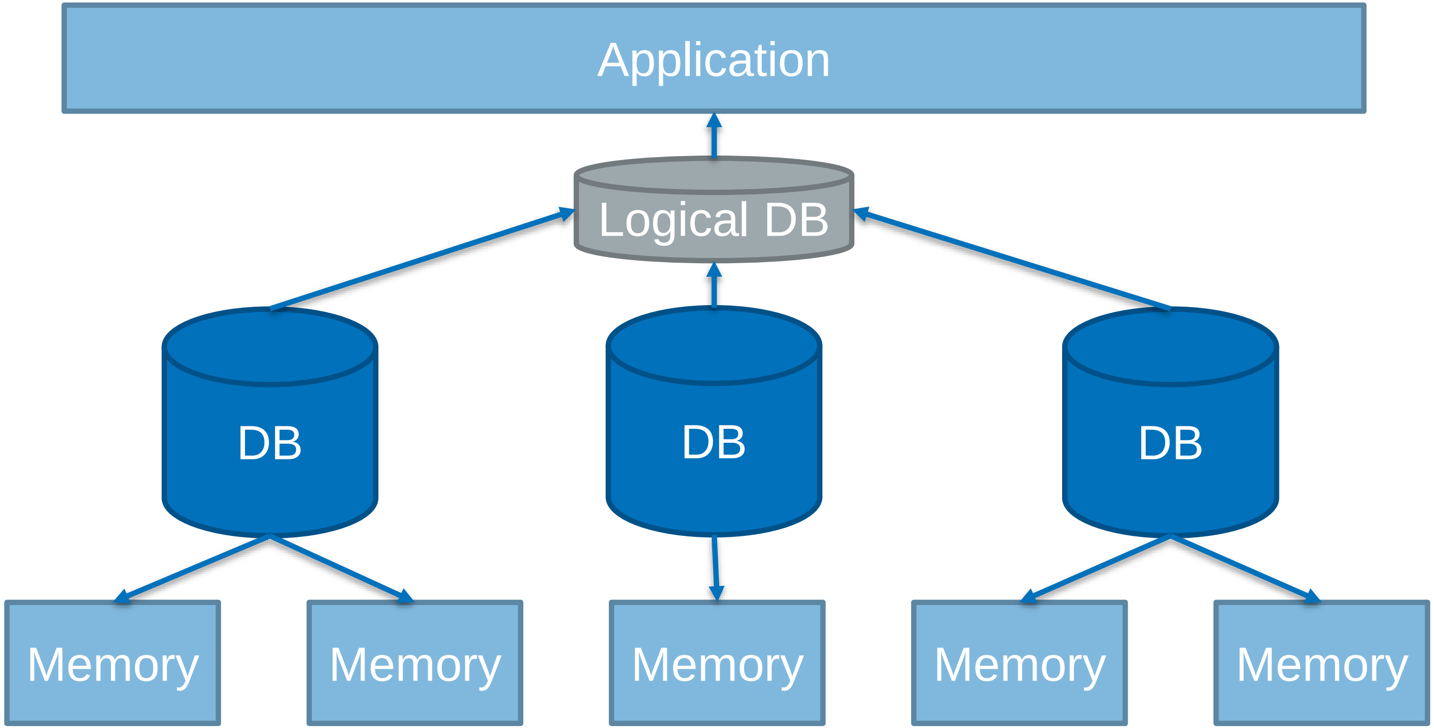
Roadmap: The Database System Universe

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES



- Shared Disk and Shared Nothing
- Cloud Databases
- Federated Databases
- Best Practices

Federated Databases



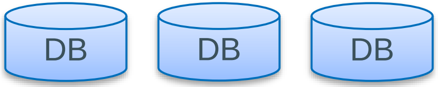
Federated Database

- Multiple Databases
- Databases are connected to one logical view
- Databases do not directly share data

Federated Databases

Federated Database

Multiple independent databases



Explicit distinction between local and remote data

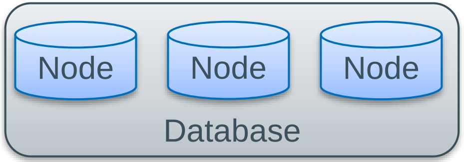
→ Not truely distributed, just coupled

→ Remote databases must be explicitly connected

→ No inconsistent states if individual databases are disconnected

Shared Nothing

One large distributed database



Data can become inconsistent if a node fails

→ Different nodes may have replicated different parts of the data and cannot synchronize with the base data when the node if offline

Example: FederatedX (MariaDB)

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

Step 1: Define the connection details of your (remote) database server

```
create server 'myserver' foreign data wrapper 'mysql' options (HOST '127.0.0.1', DATABASE 'mydb', USER 'root', PASSWORD '', PORT 3306, SOCKET '', OWNER 'root');
```

- Expects a local MySQL database called mydbat port 3306
- Connection will be available as ‚myserver‘

Step 2: Send your queries using the defined connection

Your normal SQL statement

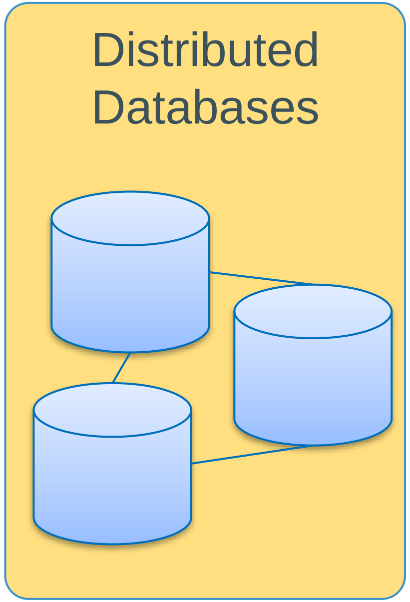
```
CREATE TABLE mynewtable (
`id` int (20) NOT NULL,
`name` int (64) NOT NULL )
ENGINE="FEDERATED" DEFAULT CHARSET=latin1
CONNECTION=‘myserver';
```

Creates a table with two integer type columns in the federated database

Step 3: Repeat with as many connections and/or queries as you like

Roadmap: The Database System Universe

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES



- Shared Disk and Shared Nothing
- Cloud Databases
- Federated Databases
- **Best Practices**

Best Practices For Using Distributed DBs

Send one big query instead of several small queries

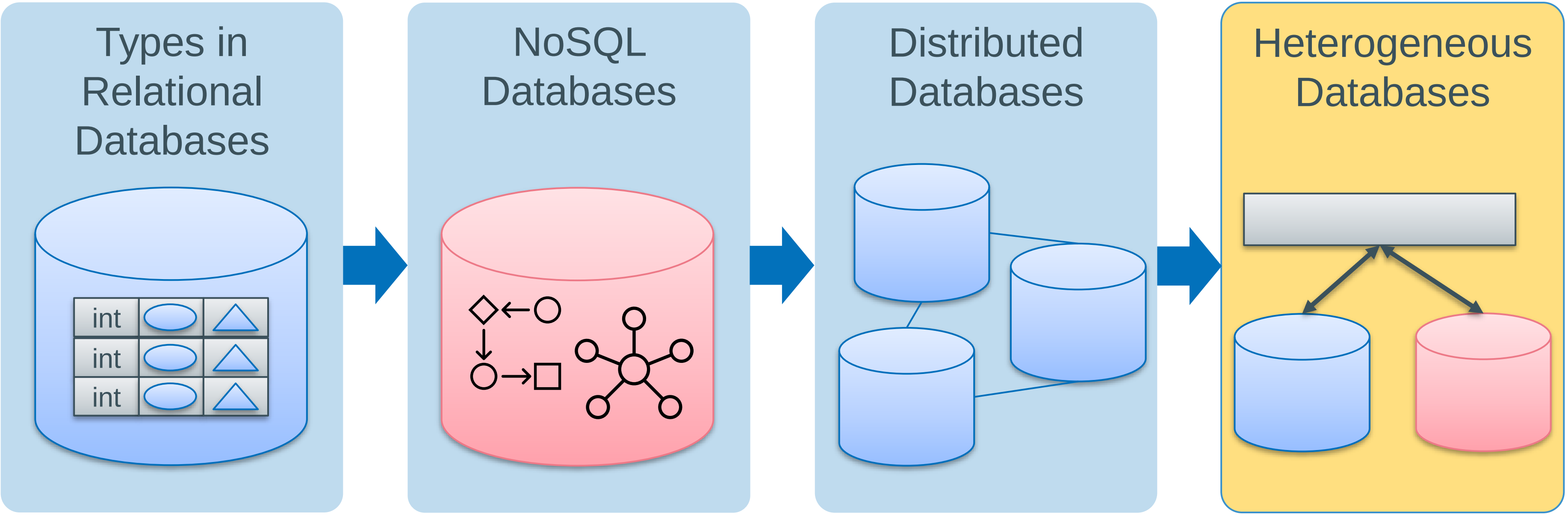
- Your optimizer will find a fast way to execute it, most probably better than you could do this
- You save time for transfering results between client and server

If possible, save a copy of your data where you want to process it

- Eliminates transfer between servers, especially useful during peak times
- Hard to control when using a cloud

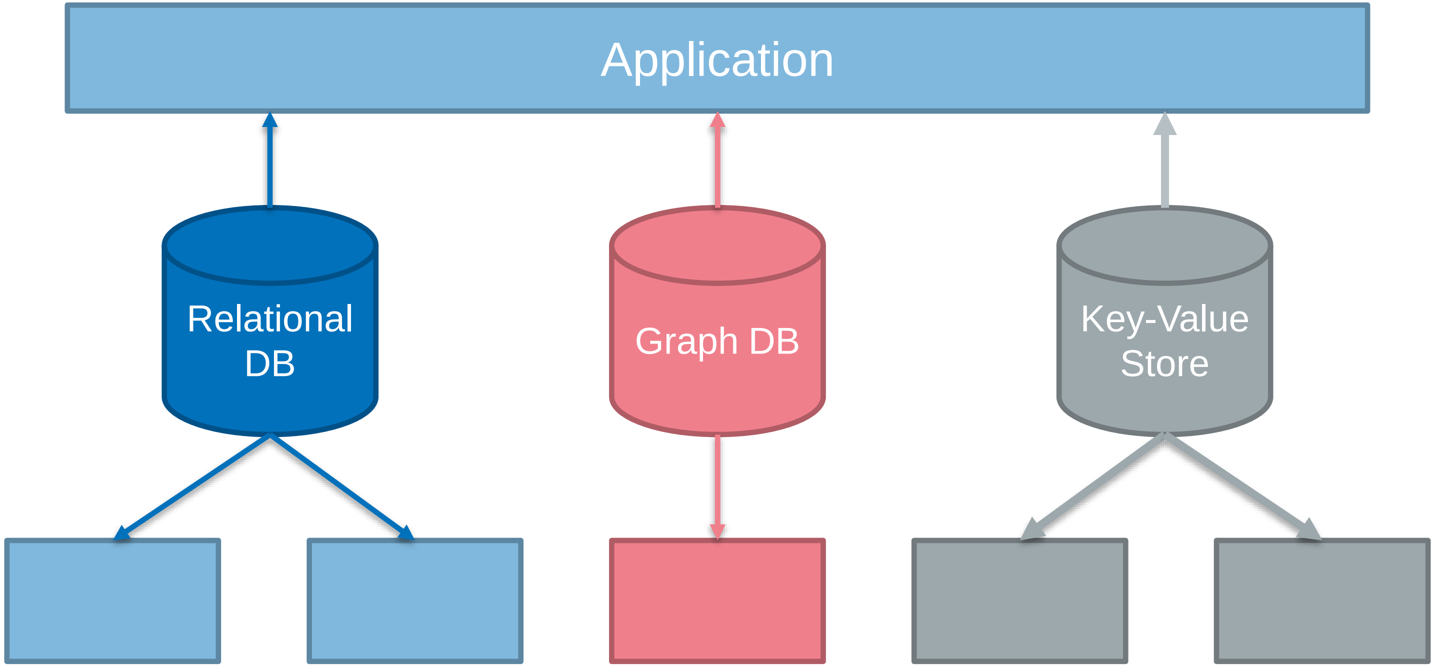
Roadmap: The Database System Universe

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES



Heterogeneous Databases

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES

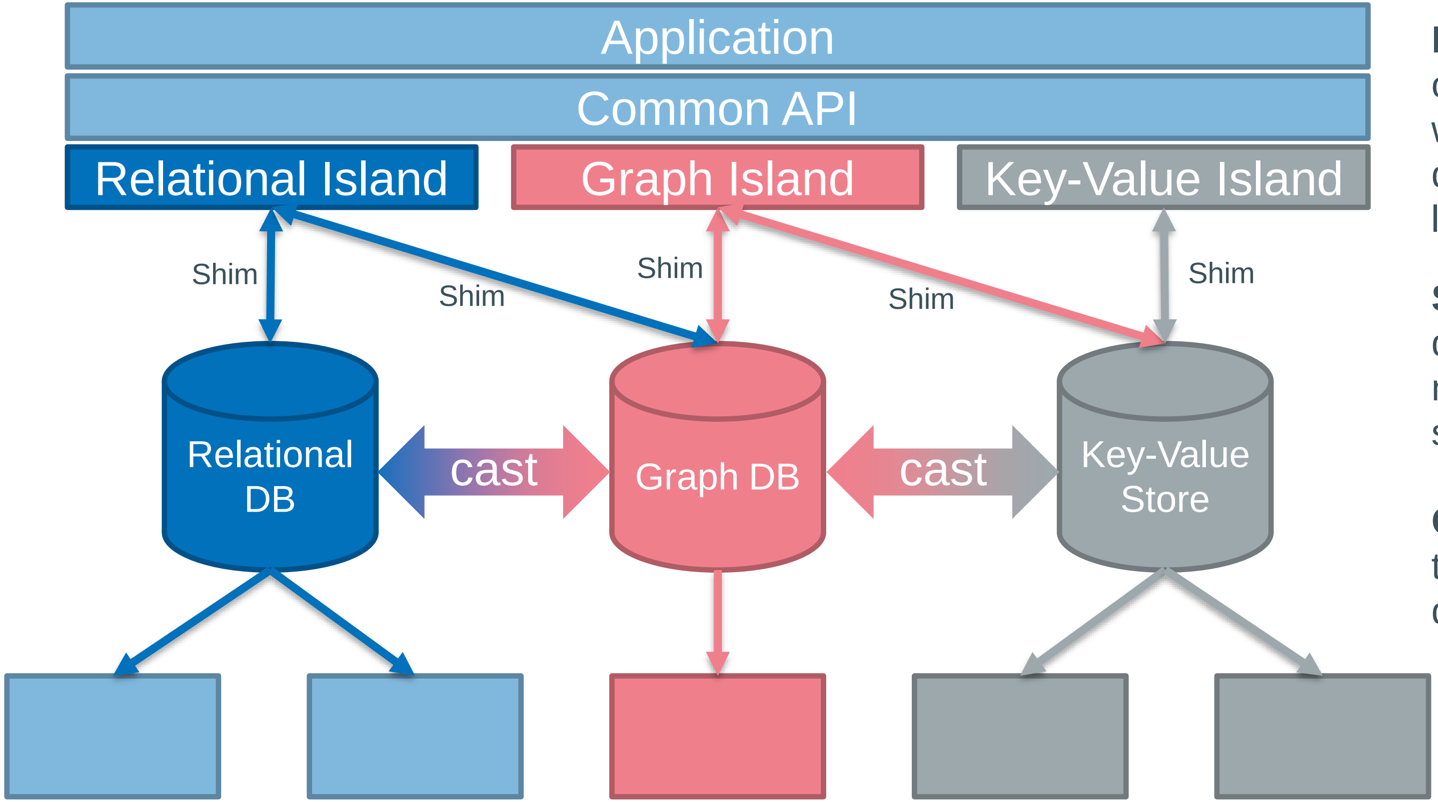


Polystore

- May or may not be distributed
- Different kinds of databases (e.g. relational and graph DBs)
- Challenges: different query languages, data models, query optimization,…

Interesting blog entry of Michael Stonebraker about Polystores: http://wp.sigmod.org/?p=1629

Polystore



Island: An asbtraction of database systems which feature the same data model and query language

Shim: Query written in a query language but might be intended for a separate island

Cast: data transformation between different data models

Polystore Example: BigDawg

- Currently supports PostgreSQL, SciDB, and Accumulo
- Other systems can be added with a bit of work as long as they fit into one of the existing islands (else, there is more work included)
- To get started, there is a tutorial which uses docker images

→ https://bigdawg-documentation.readthedocs.io/en/latest/getting-started.html

MensaMeals

Meal	Price
Pizza	6,50
Pasta	4,90
Pie	1,20
Potato Salad	5,80
Pannfisch	7,90

Relational island, e.g. PostgreSQL (bdrel):

bdrel(SELECT meal FROM MensaMeals WHERE price<5)

Array island, e.g. SciDB (bdarray):

bdarray(filter(MensaMeals, price<5))

Selection on data on array island, projection on relational island:

bdrel(SELECT meal FROM bdcast(bdarray(filter(MensaMeals, price<5)), mytable, ‘meal varchar, price REAL‘, relational))

Cast to relational island and provide new table name (mytable), attributes(meal and price), and attribute types (varchar and real)

Roadmap: The Database System Universe

CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES