25th February 2021

ISIC 2021

Emergent Hardware for Databases with focus on Semantic Web: Semantic Hybrid Multi-model Multi-platform (HM3P) Databases

Professor Dr. rer. nat. habil. Sven Groppe
https://www.ifis.uni-luebeck.de/index.php?id=groppe

Agenda: Types of Database Management Systems (DBMS)

- Cloud DBMS
- Hardware-Accelerated DBMS (GPU, FPGA, Quantum)
- IoT DBMS
- Mobile DBMS
- Federated DBMS
- Multi-Model DBMS
  - relational, XML, JSON, graph, Semantic Web, unstructured
- Multi-Platform DBMS
  - Examples
  - Multi-Platform Development
- Hybrid Multi-Model Multi-Platform (HM3P) DBMS
  - Challenges

Zoo of Data Formats, for example:

- relational data
  - in relational databases
- XML
  - for exchange
- JSON
  - web data
- Resource Descr. Framework (RDF)
  - Semantic Web
- graph data
  - from social networks
- unstructured data
  - of social media like wikis

➡ Parallel use of different Data Models for storing and processing

Relational:

Primary		Secondary

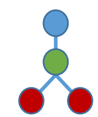
Primary	

XML:

```
<root>
 <child>
  <first>hello</first>
  <sibling>sibling</sibling>
 <child>
</root>
```

JSON:

```
{root:{
    child:{
      first:hello,
      sibling:sibling
    }
}}
```

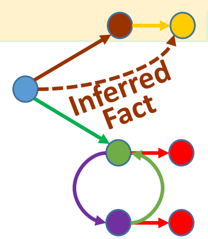


Tree

RDF/Graph Data:

Ontology of Semantic Web

:article	rdfs:subclassOf	bench:doc
:article1	rdf:type	:article .
:article1	dc:creator	:person1 .
:person1	foaf:name	'Martin' .
:person1	:likes	:person2 .
:person2	foaf:name	'Jennifer' .
:person2	:hates	:person1 .

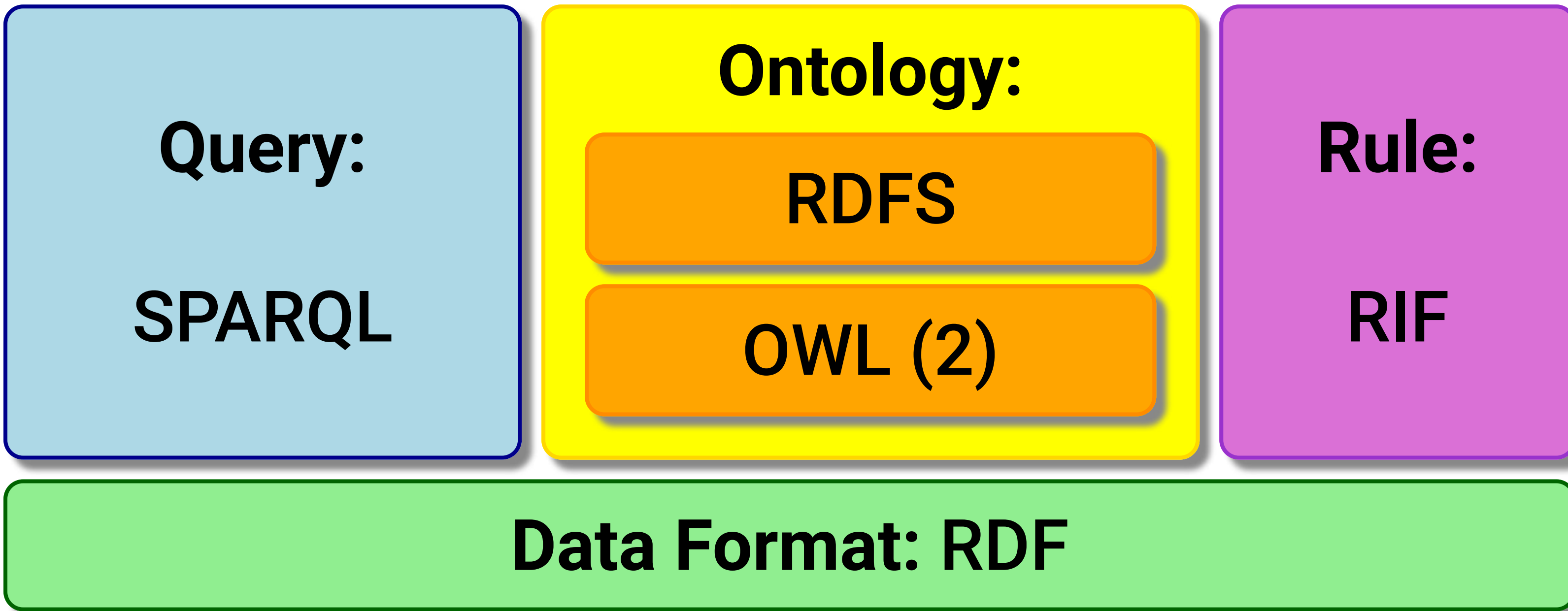


Graph

Unstructured Data:

Title
The following issues are important:
1. Very Importand Persons (VIPs)
2. Very Important Data (VID)

Semantic Web (Core) "Standards"



- Every data model (here Semantic Web) has its own set of languages (data, query, rule, ...)

Semantic Web: Ontology

- Ontology as additional abstraction layer
  - More than schema descriptions:
    - Specification of background knowledge (based on which new facts can be derived)

      $\Rightarrow$ avoids storing of redundant data

      $\Rightarrow$ supports re-use of data

      $\Rightarrow$ supports data integration

      $\Rightarrow$ **increases computational complexity**

Special Concepts 1/2: Open world assumption (OWA)

- Closed World Assumption (CWA) in Databases:

  "The database contains all and anything not contained in the database is presumed to be false/not existent!"

- Open Context like Web
  ➡ CWA is false!

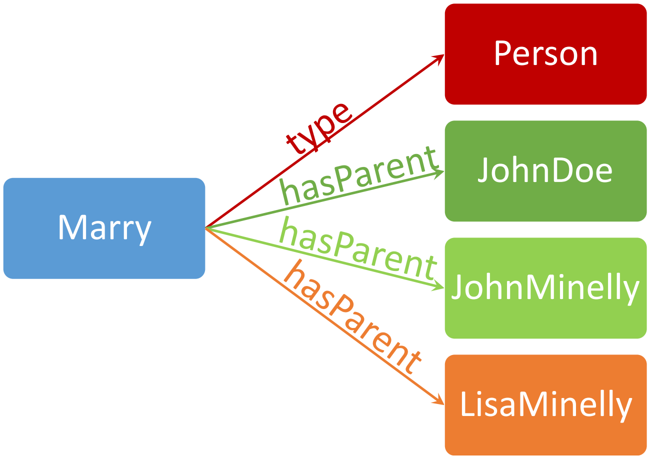
Special Concepts 1/2: Open world assumption (OWA): Example

- Data source 1 contains:
  "There exists a flight at 2pm"
  "There exists a flight at 3pm"
- My query:
  "Is there a flight at 5pm?"
- CWA Result: No!
- OWA Result: unknown!
  - i.e., there could be a data source 2, which contains the information about a flight at 5pm!
    Data source 2 is maybe currently not integrated or currently not available...

Special Concepts 2/2: No unique name assumption

Example:

A child has two parents: (in DL: Person $\sqsubseteq \leq 2$ hasParent.Person), but the following facts seem to be conflicting:

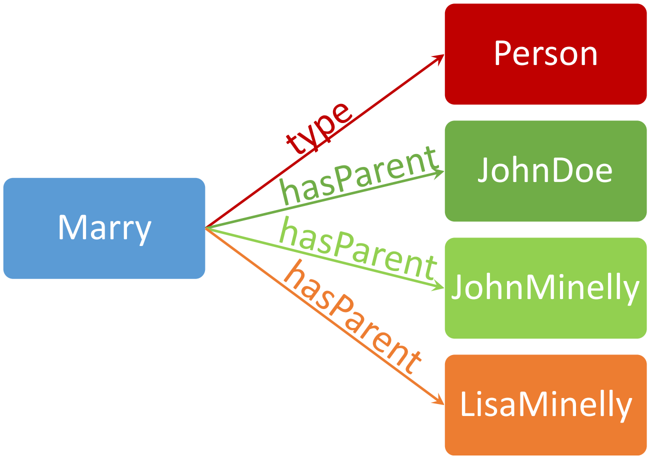


Special Concepts 2/2: No unique name assumption

Example:

A child has two parents: (in DL: Person $\sqsubseteq \leq 2$ hasParent.Person),

but the following facts seem to be conflicting:



No unique names/keys

➡ JohnDoe, JohnMine and LisaMine are **not** necessarily different objects (here persons)

Special Concepts 2/2: No unique name assumption

4 possibilities:

1. JohnDoe $\equiv$ JohnMine
2. JohnDoe $\equiv$ LisaMine
3. JohnMine $\equiv$ LisaMine
4. JohnDoe $\equiv$ JohnMine $\equiv$ LisaMine

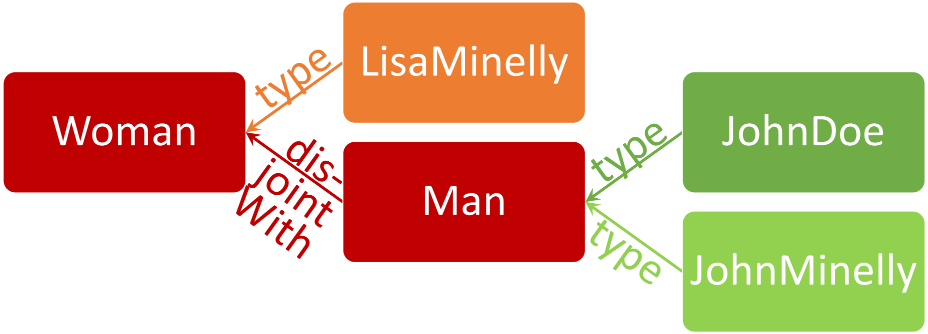
Special Concepts 2/2: No unique name assumption

4 possibilities:

1. $\boxed{\text{JohnDoe} \equiv \text{JohnMine}}$
2. JohnDoe ≡ LisaMine
3. JohnMine ≡ LisaMine
4. JohnDoe ≡ JohnMine ≡ LisaMine

Only $\boxed{1.}$ is intuitive for humans!

Adding following facts and axioms:

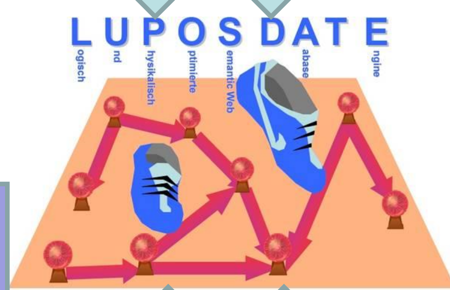


➡ automatic inference of 1. possibility!

Semantic Web DBMS LUPOSDATE

Support of:
- SPARQL Queries
- RIF Rules
- RDF Schema
- OWL (via OWL2RL in RIF)

Indexing:
- Stream Processing
- Main memory for small datasets
- Disk-based for large datasets
    - RDF3X
- Cloud: HBase
- P2P



Visualizations:
- Visual Editor
    - Queries (SPARQL)
    - Rules (RIF)
    - Data (RDF) in
        - 2D and
        - 3D
    - Logical Optimization Rules
- Summaries of RDF Data
- Operator graph
- Processing of Queries and Rules
- Optimization Steps

Extra:
- Parallel Processing
- Distributed Processing
- Cloud Computing
- Mobile Computing
- P2P for Internet of Things
- Compression of RDF Data
- Embedding of SW Languages in Programming Languages
- Speeding up by FPGAs

RDF3X - Indexing Scheme for large-scale RDF triple stores

$\bowtie_{Merge(?car)}$

sorted by ?car

R: ?car rdf:type ont:Car

S: data:Sven ont:Owns ?car

Maximise usage of **merge joins**
⇒ 3!=6 indices for **all possible sort orders** of triples SPO:
SPO, SOP, PSO, POS, OSP, OPS

	R	*S*
Prefix-Search in **Index**:	PSO	SPO
with (Prefix-)**Key**:	rdf:type ont:Car	data:Sven ont:Owns

Result of Triple Pattern:

?car	?	?car
HL 100		**HL 999**
HL 101		
…		
HL 999		
…		

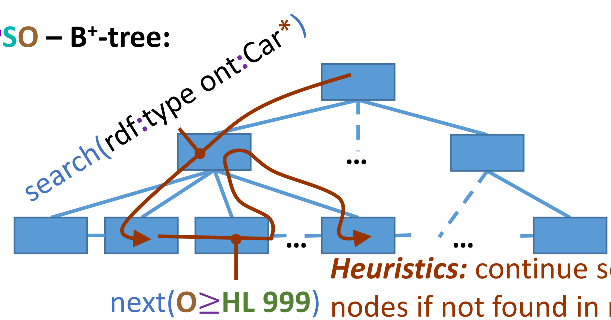
next(?car≥HL 999)

„Sideways **Information** Passing“

Complexity of Merge Join $\bowtie_{Merge}$:

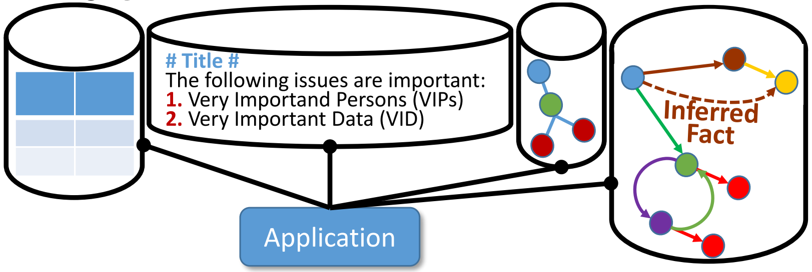
Worst Case (duplicates): $O(|R| \times |S|)$
without duplicates: $O(|R| + |S|)$
with **sideways information passing**: $O(|R \bowtie S|)$
(assuming quasi-constant access in B⁺-tree)

Search in PSO – B⁺-tree:



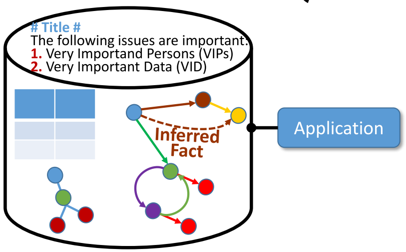
Heuristics: continue search in inner nodes if not found in next leaf node!

Polyglot Persistence



- data sources: integration at application level
- performance of data processing cannot be fully optimized
- fault-tolerance cannot be transparently offered across the different databases
- zoo of query languages
+ features of different types of databases can be used

Multi-Model DBMS (MM-DBMS)

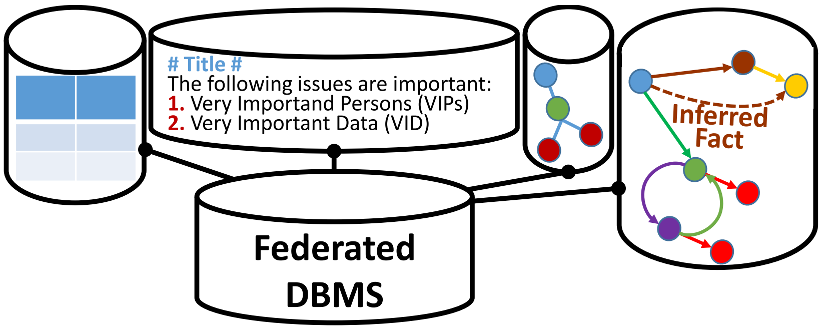


+ full and uniform data integration at database level
+ performance: fully optimized across different data models
+ transparent fault-tolerance
+ SQL standards:

relational ('87), XML ('03), temporal ('11), JSON ('16), Multi-dimensional Arrays ('19), schemaless ('19), streams ('20?), property graphs ('21?)

- features of different types of databases can**not** be used

Federated DBMS



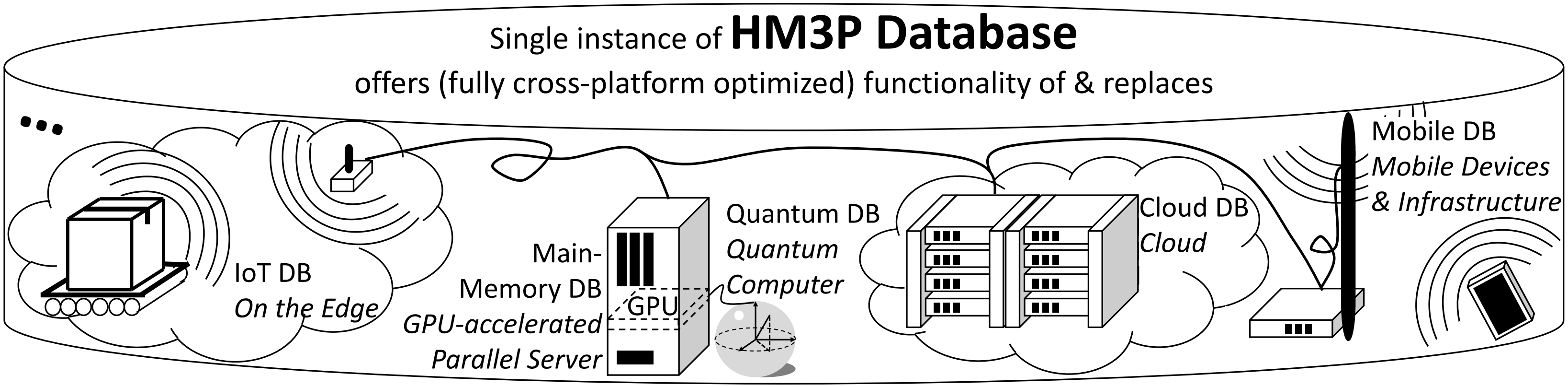
- Bottom-up-integration of existent databases
- mostly independent DBMS with private conceptual database schemes
- partially enabling external accesses (in cooperation)
- heterogeneity of data models and transaction management possible (but relational DBMS in most times)

- problems with semantic heterogeneity
- transparency in distribution only partially achievable

One Size-Approach

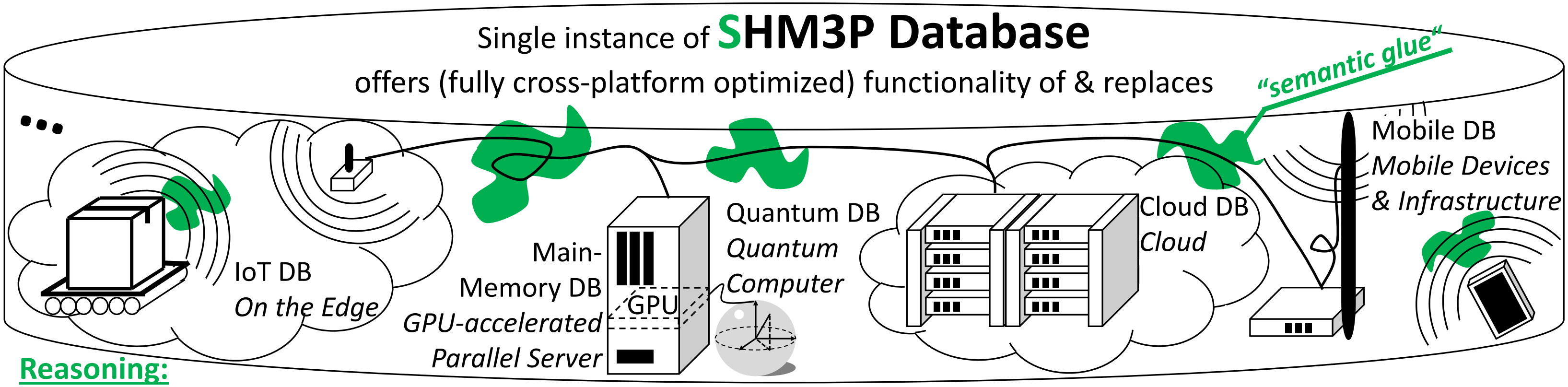
- M. Stonebraker, U. Cetintemel. *"One Size Fits All": An Idea Whose Time Has Come and Gone*. ICDE 2005
  - *The last 25 years of commercial DBMS development can be summed up in a single phrase: "One size fits all".*
  - *...this concept is no longer applicable to the database market...*
- Our approach: **Enlarge the size!**
  - Over the boundaries and limitations of single platforms and their specialized approaches
  - Increase transparency, performance and ease of use

Hybrid Multi-Model Multi-Platform (HM3P) Database



- full and uniform data integration at database level
- performance: fully optimized across different data models
- transparent fault-tolerance
- SQL standards: relational ('87), XML ('03), temporal ('11), JSON ('16), Multi-dimensional Arrays ('19), schemaless ('19), streams ('20?), property graphs ('21?)
- **features of different types of databases running on different platforms can be used**

Variant: Semantic HM3P (SHM3P) DB



Reasoning:

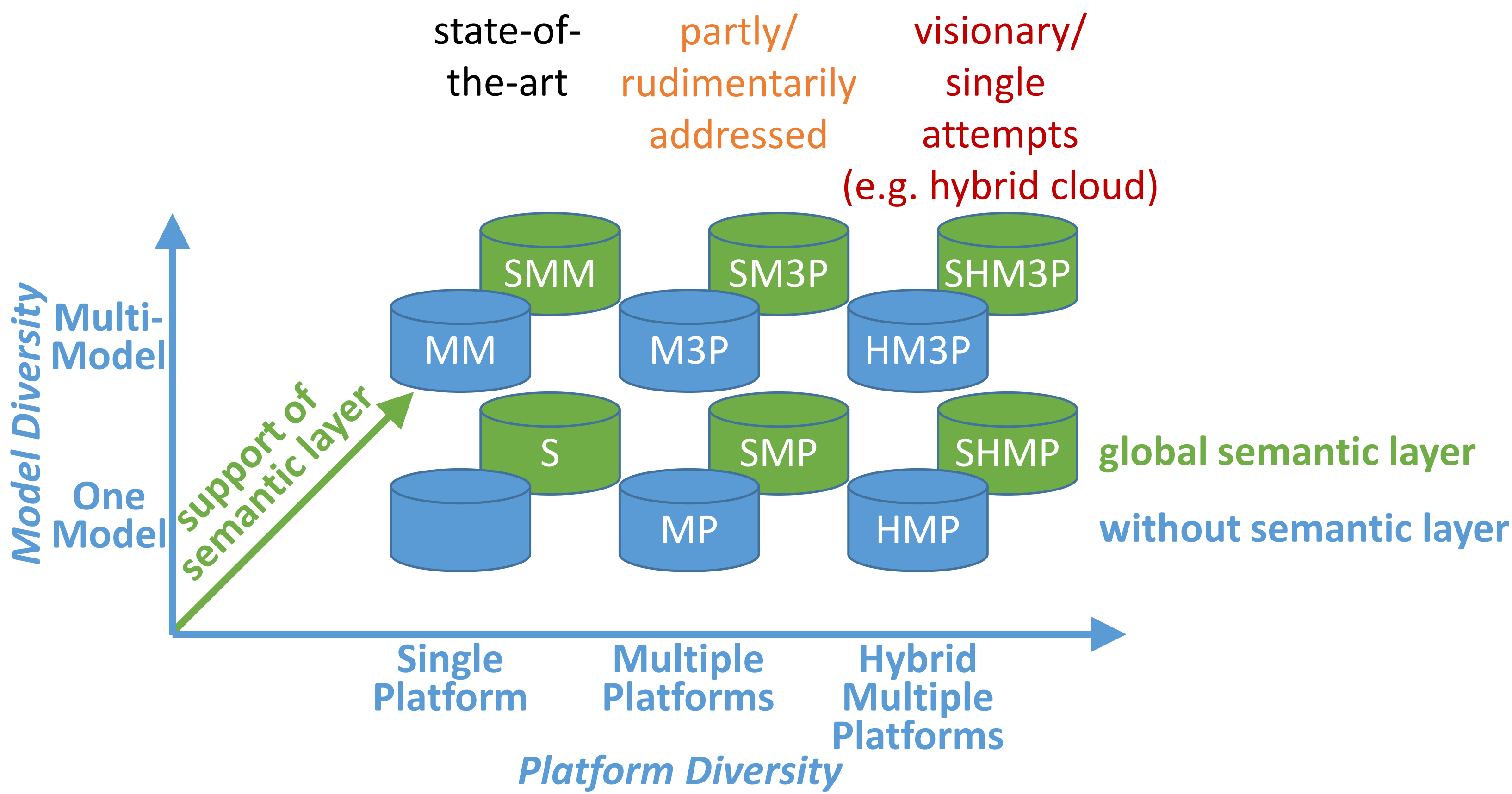
Lightweight reasoning on large data sizes of IoT devices	**Heavyweight reasoning on moderate data sizes**	**Heavyweight reasoning on large data sizes**	**Reasoning on small data sizes of mobile devices**

How to integrate the different reasoning capabilities and requirements into one transparent global reasoner?

- Semantic Layer as glue between other models and platforms
- new challenges like integrating different types of reasoners in a transparent global reasoner

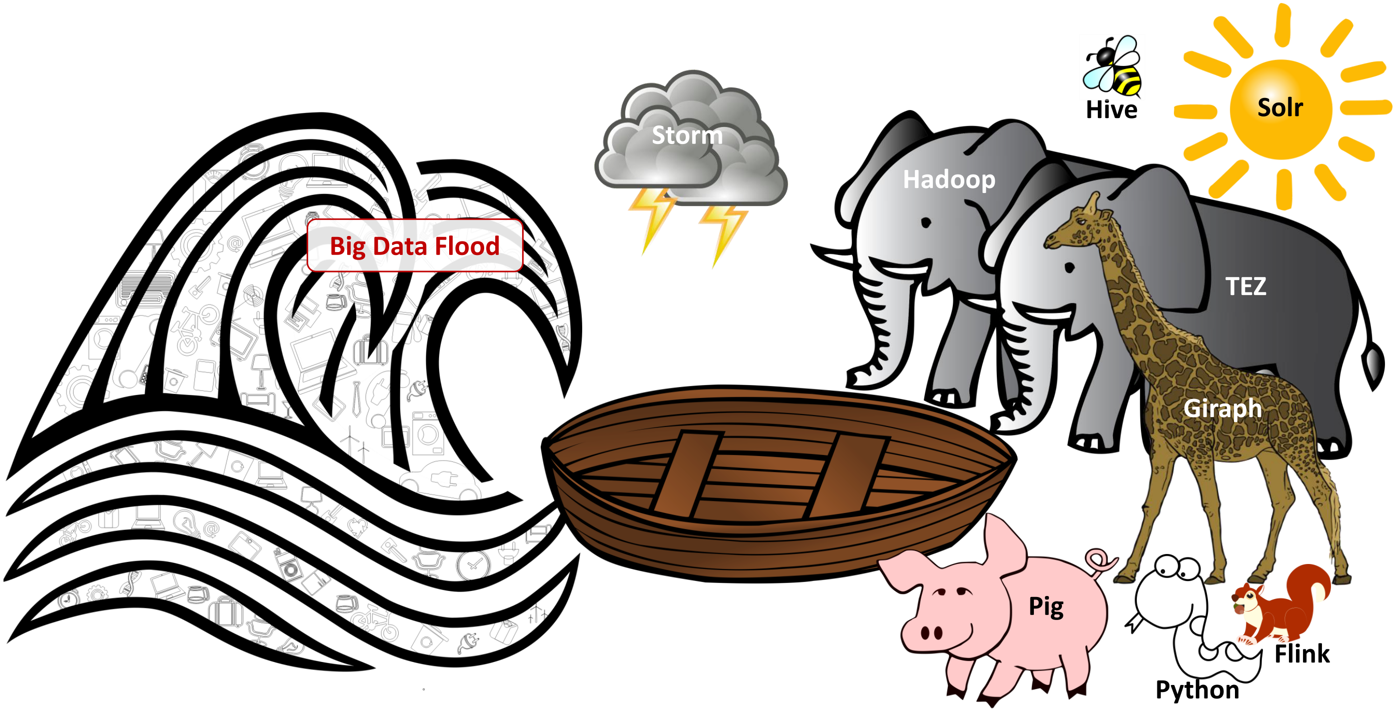
+ Features of HM3P databases
+ Easier data integration
- Performance issues may occur due to semantic layer

Types of DBMS

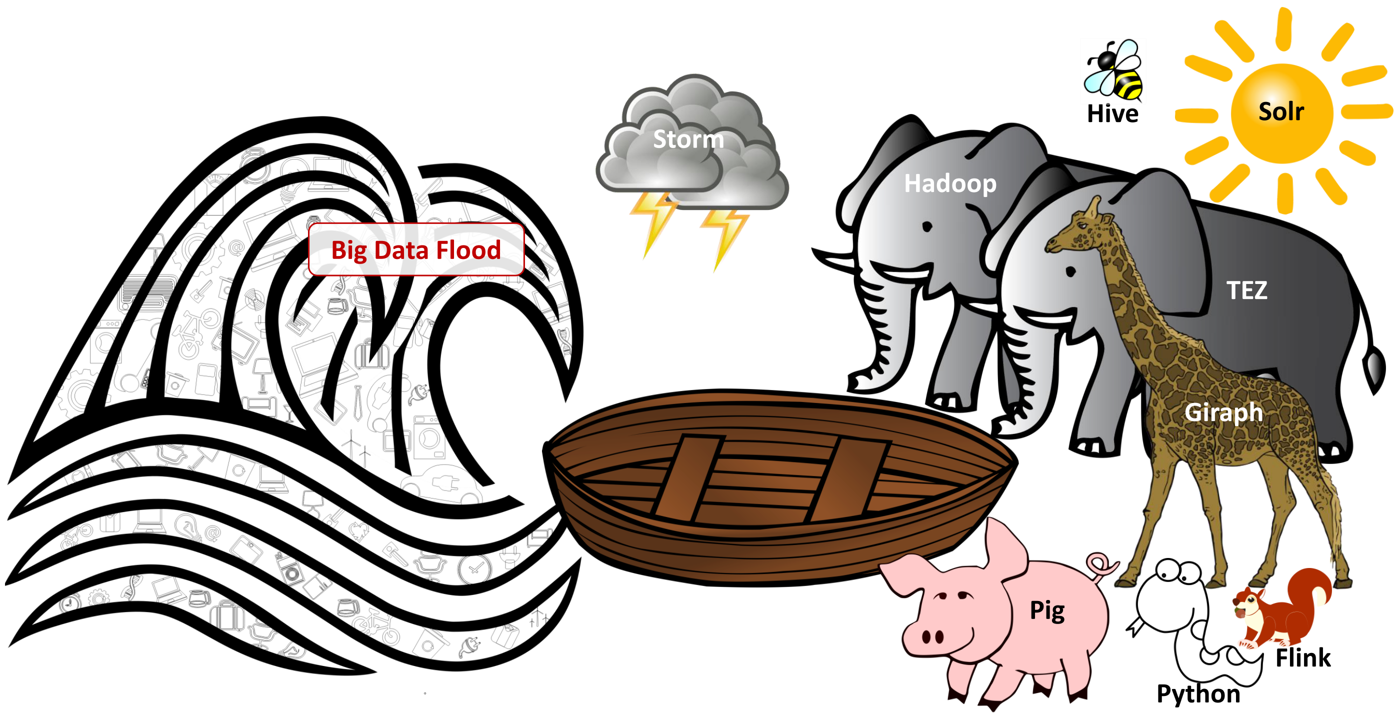


Legend: S: Semantic MP: Multi-Platform MM: Multi-Model M3P: MM MP H: Hybrid

The ark is too small...

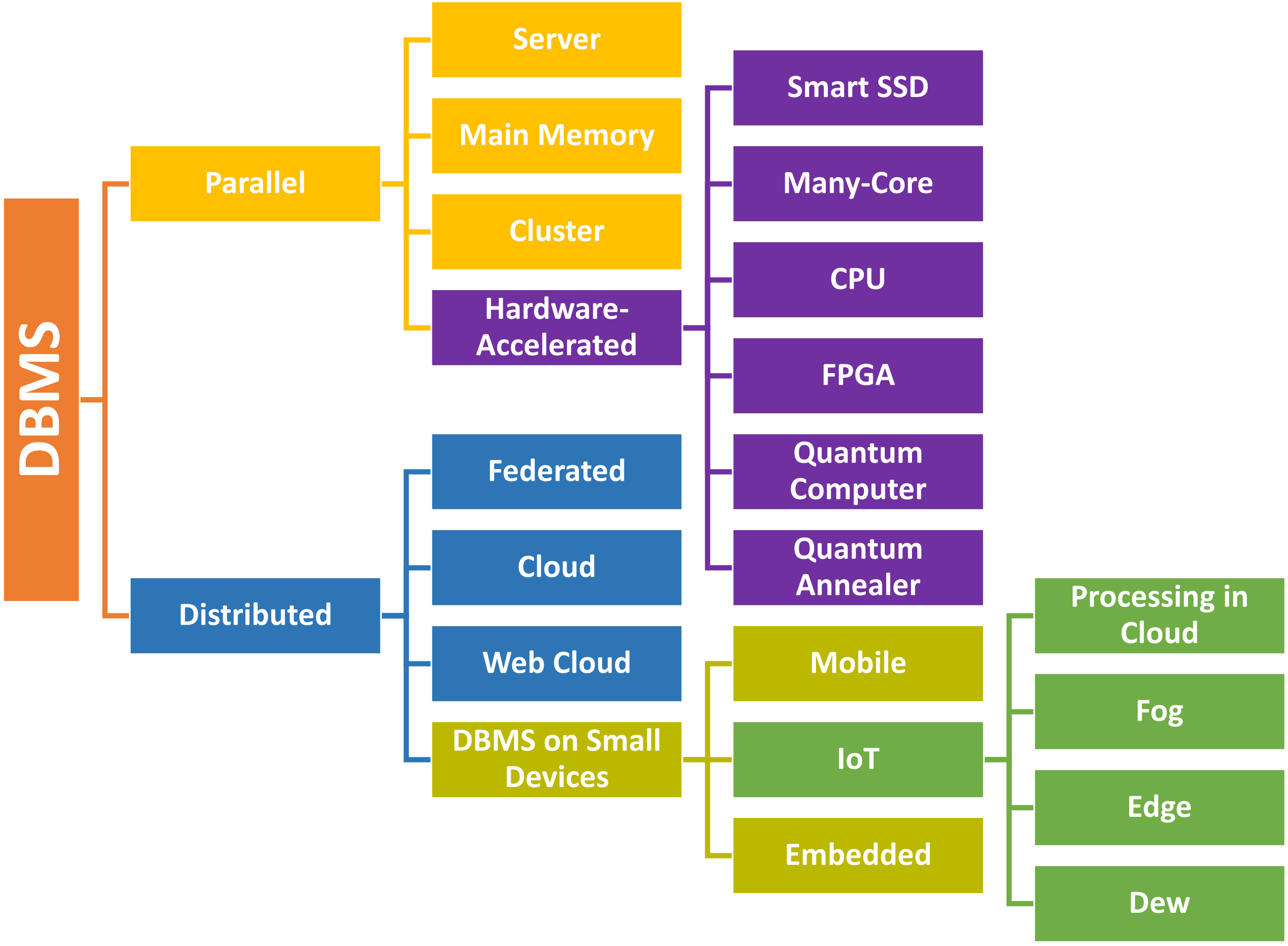


The ark is too small...



- ... but there is always enough space for the own product/research system! → luposdate3000

Platform-specific types of DBMS



Examples of Multi-Platform Databases 1/2

Type	DBMS	Ext.	Models RCKJXGDO	Query Languages	Platforms NJWLUMSZCH
Relational	PostgreSQL	I	R-KJX--O	extended SQL	N-WLUMS-CH
	MS SQL Server	I	R--JXG-O	extended SQL	N-WL----CH
	IBM DB2 LUW	I	R---XGDO	extended SQL/XML	N-WLU-S-C-
	IBM DB2 z/OS	I	R---XGDO	extended SQL/XML	N------Z--
	Oracle DB	I	R--JX-DO	SQL/XML, SQL/JSON	N-WLUMS*CH
	MySQL	II	R-K----O	SQL, memcached API	N-WLUMS-C-
	Sinew[1]	III	R-K-----	SQL	N-WLUMS-CH
Column	Cassandra	I	-C---G-O	SQL-like CQL	-JWLUMS-CH
	CrateDB	I	RC-J-G--	SQL	-JWL-M--C-
	DynamoDB	I	-CKJ-G-O	simple API (get/put/update) + simple queries over indices	-JWLUM--C-
	Vertica	II	-C-J-G--	SQL-like	N--LU---CH

Legend: Ext.: I = adoption of a new storage strategy, II = extension of the original storage strategy, III = creation of a new interface, IV = no change;
Models: R = relational, C = column, K = key/value, J = JSON, X = XML, G = graph, D = RDF, O = object, – = no support;
Platforms: N = Native Machine Code, J = Java/JVM, W = Win, L = Linux, U = Unix (e.g. BSD), M = macOS, S = Solaris, Z = z/OS, C = Cloud, H = Hybrid Cloud, – = no support, * = support for old versions.

[1] D. Tahara, T. Diamond, and D. J. Abadi, "Sinew: A SQL System for Multistructured Data," in SIGMOD, 2014

Examples of Multi-Platform Databases 2/2

Type	DBMS	Ext.	Models RCKJXGDO	Query Languages	Platforms NJWLUMSZCH
Key/value	Riak KV	I	--KJXG--	Solr	N--LUM--CH
	c-treeACE	III	R-K--G--	SQL	N-WLUMS-C-
	Oracle NoSQL DB	III	R-K--GD-	SQL	-JWLUMS-C-
Document	Cosmos DB	I	-CKJ----	SQL-like	N-------C-
	ArangoDB	II	--KJ-G--	SQL-like AQL	N-WL-M--C-
	MongoDB	II	--KJ---O	JSON-based	N-WL-M--C-
	Couchbase	III	--KJ----	SQL-based N_1QL	N-WL-M--CH
	MarkLogic	III	---JX-DO	XPath, XQuery, SQL-like	N-WL-M--CH
Graph	OrientDB	II	--KJ-G--	Gremlin, extended SQL, SPARQL	N-WLUM--CH
Object	InterSystems Caché	III	R--JX--O	SQL with object extensions	N-WLUMS-CH

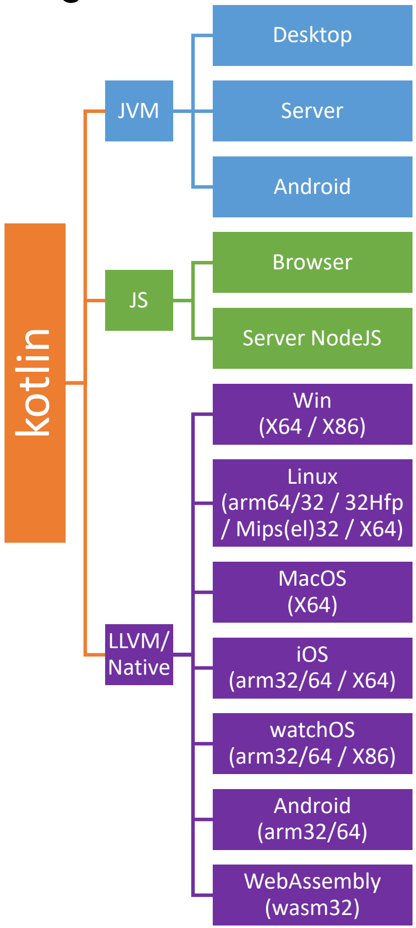
Legend: Ext.: I = adoption of a new storage strategy, II = extension of the original storage strategy, III = creation of a new interface, IV = no change;
Models: R = relational, C = column, K = key/value, J = JSON, X = XML, G = graph, D = RDF, O = object, - = no support;
Platforms: N = Native Machine Code, J = Java/JVM, W = Win, L = Linux, U = Unix (e.g. BSD), M = macOS, S = Solaris, Z = z/OS, C = Cloud, H = Hybrid Cloud, - = no support, * = support for old versions.

Multi-Platform Development of DBMS

- Native Binaries via C/C++
  - support of a new platform: porting code is necessary
  - code close to hardware, fast execution
  - direct access to native libraries
  - doesn't run in browser
  - most server DBMS: C/C++ code
- Java/Java Virtual Machine (JVM)
  - runs on many platforms (without porting code)
  - interpreted bytecode, via Just-In-Time compilation comparable speed to native execution
  - no direct access to native libraries
  - does neither run on iPhone nor in browser
  - many NoSQL/NewSQL/Cloud DBMS: Java (or JVM language like Scala) code
- Code generation for query processing via C/C++ or Janino-Compiler (JVM)

Multi-Platform Development with Kotlin

Targets:



- Most target platforms are supported
- Splitting the project in platform-**in**dependent and platform-dependent code
  - Platform-dependent code can be partly coded in the programming language of the target platform (e.g., Java for JVM, JS for Web)
- Enables one code repository for various target platforms
  - Sharing of code between server & (various) clients
- Avoids efforts to port code (into other programming languages)

Multi-Platform Development with Kotlin

- Common Module
  - Code independent of platforms containing declarations for platform dependent code without implementation, e.g.:

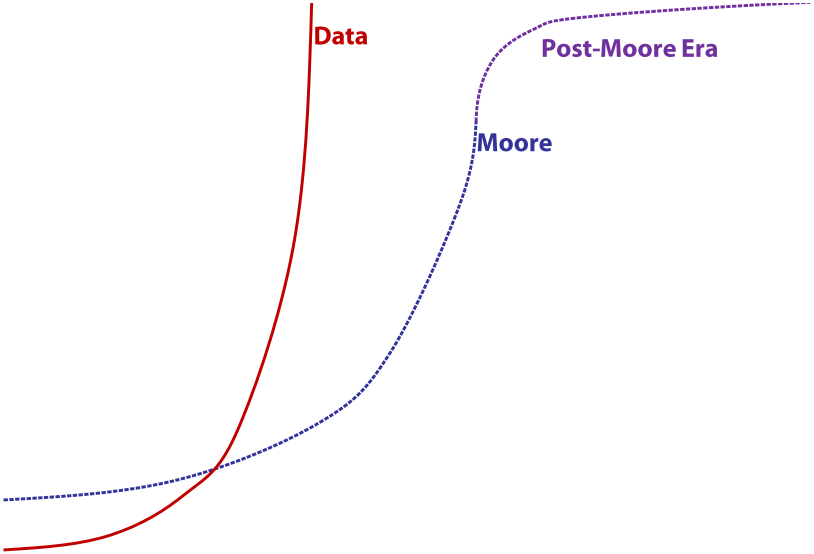
```
expect fun formatString(source: String, vararg args: Any): String
expect annotation class Test
```

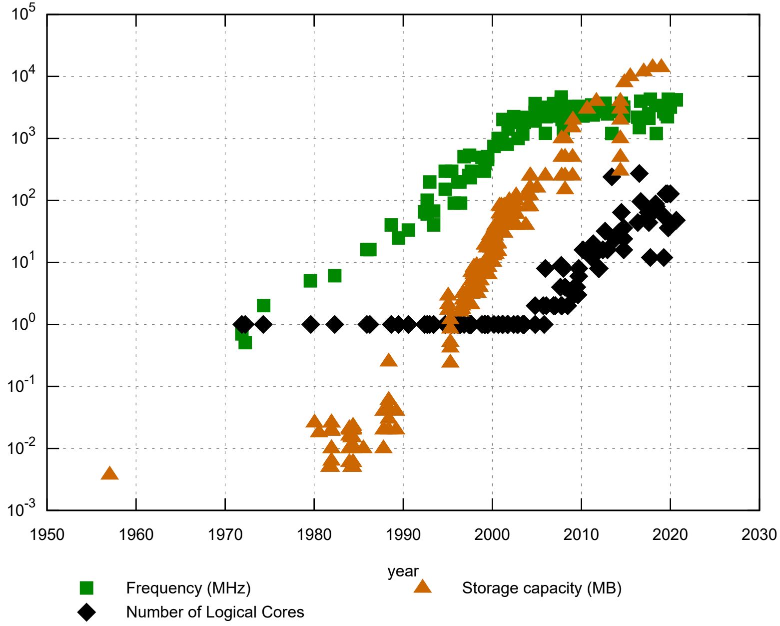
- Platform Module
  - Implementation of within the common module declared platform-dependent code (and other platform-dependent code), e.g.:

```
actual fun formatString(source: String, vararg args: Any) =
                                        String.format(source, args)
actual typealias Test = org.junit.Test
```

- Regular Module
  - depend on platform modules or platform modules depend on this module
- **However: High compilation times, faster**: Including different sets of source code directories for different targets and configurations (e.g., centralized, Cloud, P2P, browser, ...)

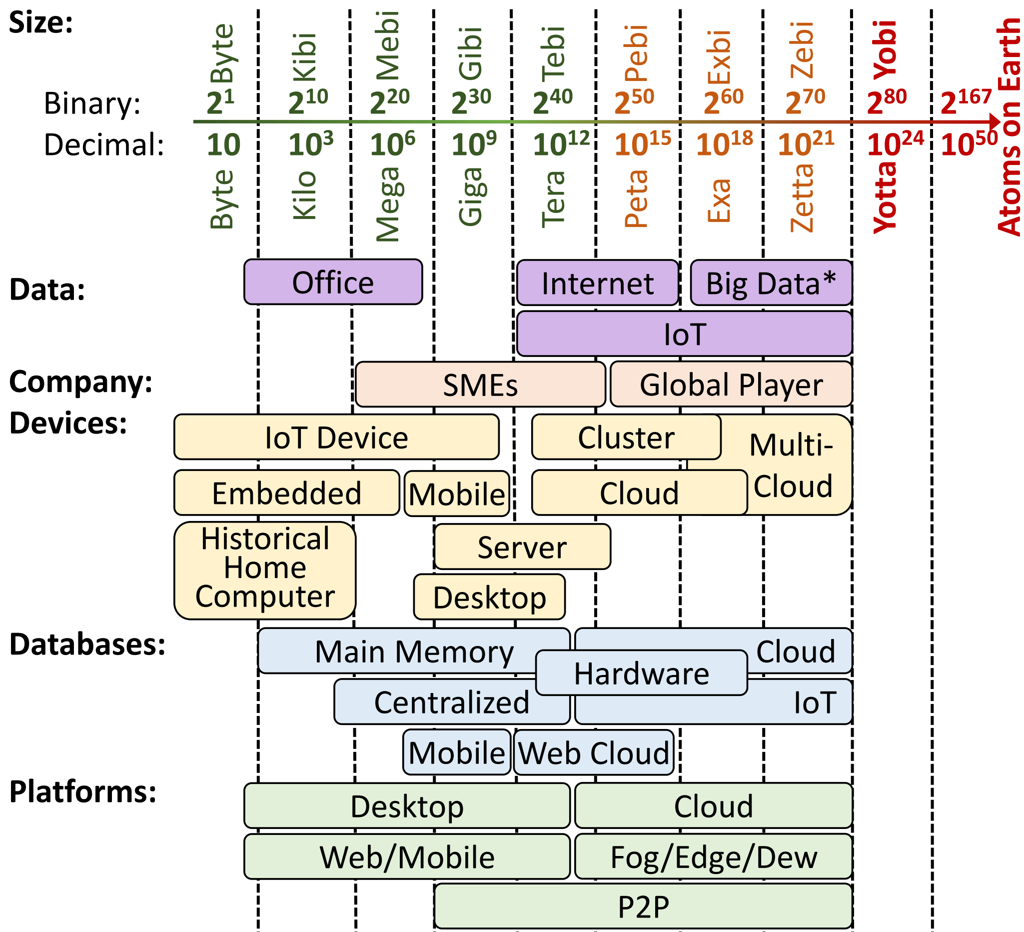
Data versus Moore





- Data sizes are growing faster than computing capacity of single CPU
  ➡ Parallel/distributed computing to overcome limitations of single CPUs

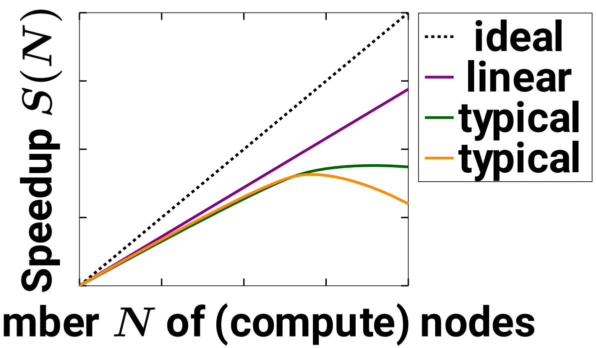
Data Sizes

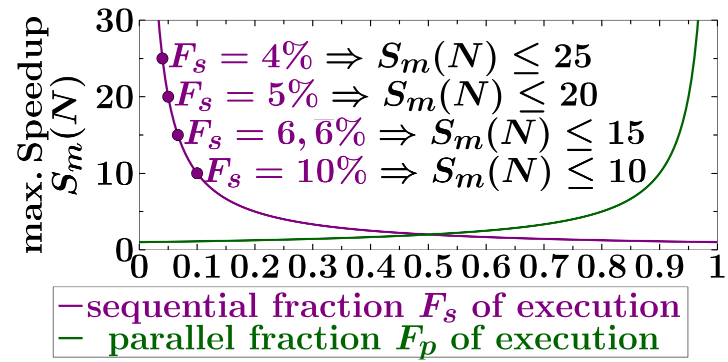


SMEs: Small and medium-sized enterprises * social media, search engines

Amdahl's versus Gustafson‘s law

- Amdahl's law
  - a sequential part of the overall algorithm limits overall speedup (in the context of fixed problem/data size)





- Gustafson‘s law:
  - programmers tend to set the size of problems to fully exploit the computing power that becomes available as the resources improve
  - if faster equipment or more nodes are available, larger problems can be solved within the same time

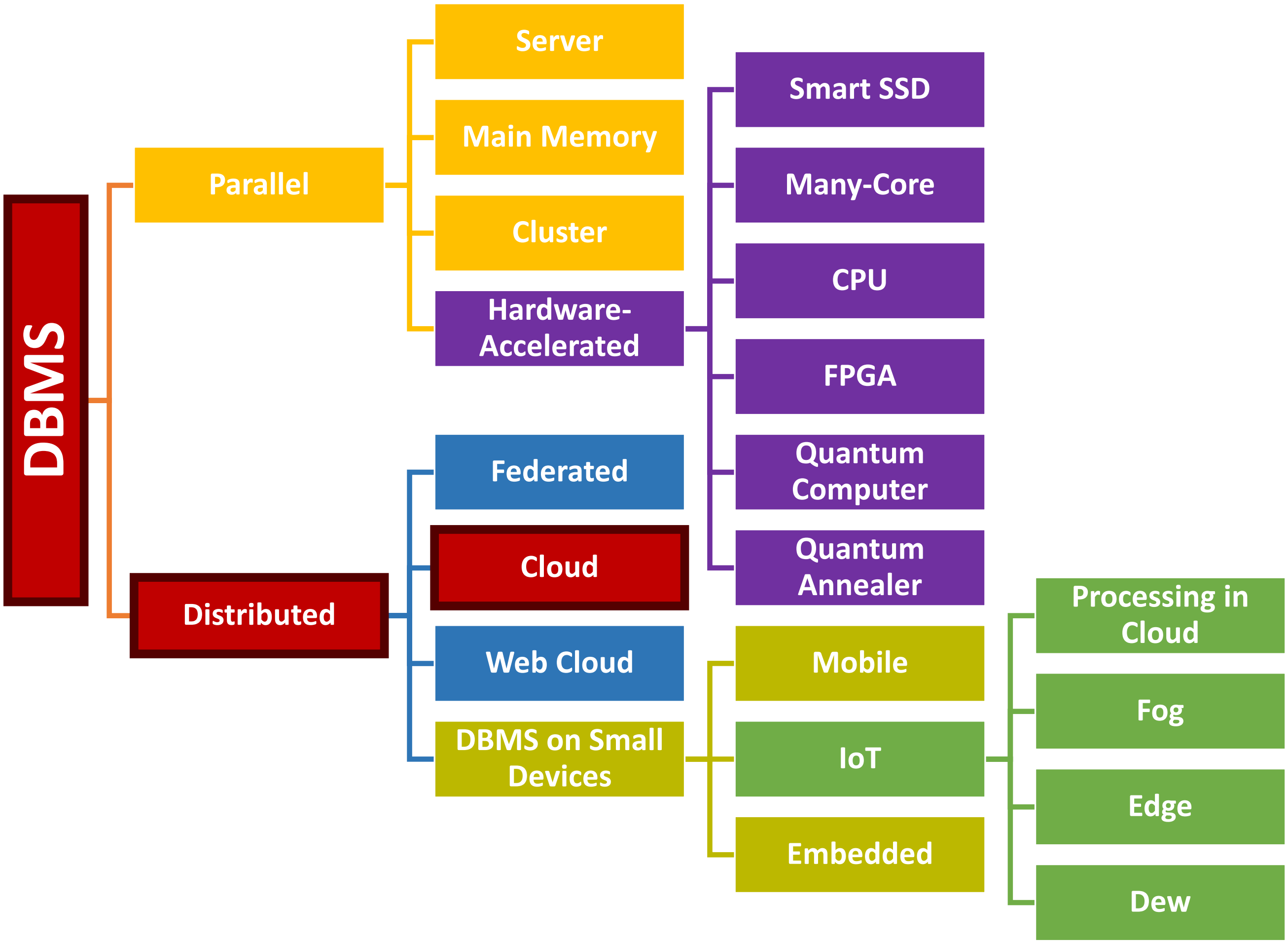
PACELC Theorem as Refinement of CAP

- In case of network partitions (P):
  - Guarantee of either Availability (A) or Consistency (C) (like CAP theorem)
- In normal operation without network partitions errors: "Else (E)"
  - Guarantee of either small Latency (L) or strong Consistency (C)
- NoSQL-DBMS
  - some with several configuration possibilities
  - challenge for hybrid: transparent global approaches supporting different PACELC properties for different partitions at the same time

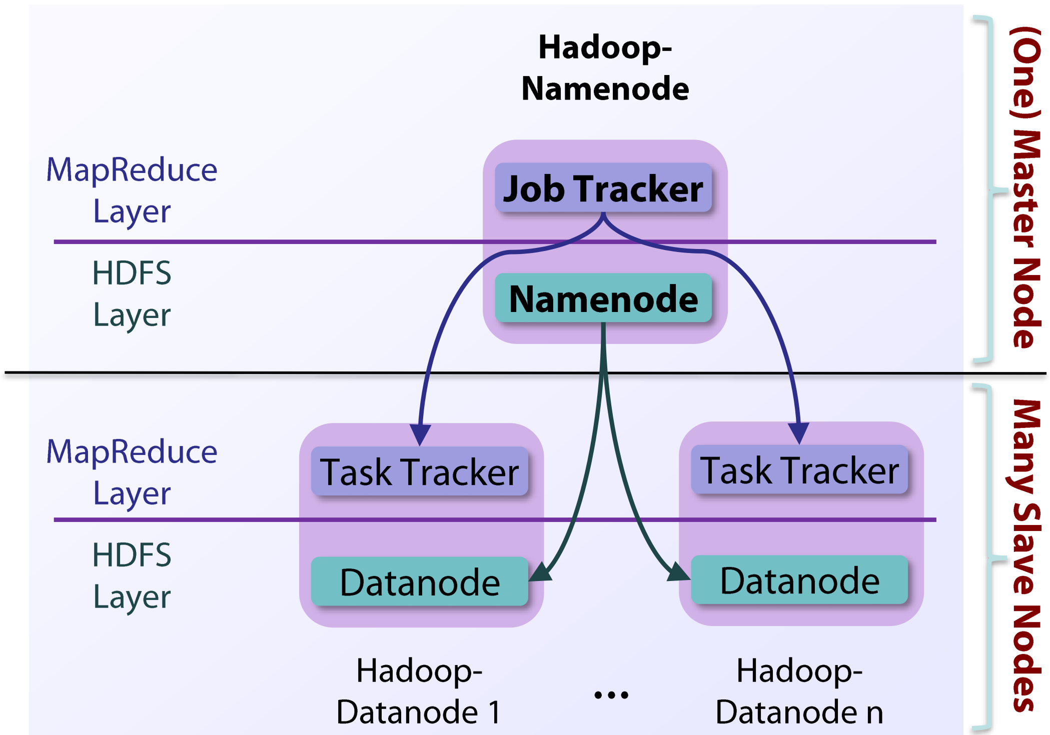
Distributed DBMS	P+A	P+C	E+L	E+C
DynamoDB, Cassandra, Cosmos DB, Riak	✓		✓	
Couchbase, FaunaDB		✓	✓	✓
VoltDB/H-Store, Megastore, BigTable/HBase, MySQL Cluster		✓		✓
MongoDB	✓			✓
PNUTS		✓	✓	
Hazelcast IMDG	✓	✓	✓	✓

D. Abadi: Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. Computer 45(2), 2012.

Platform-specific types of DBMS

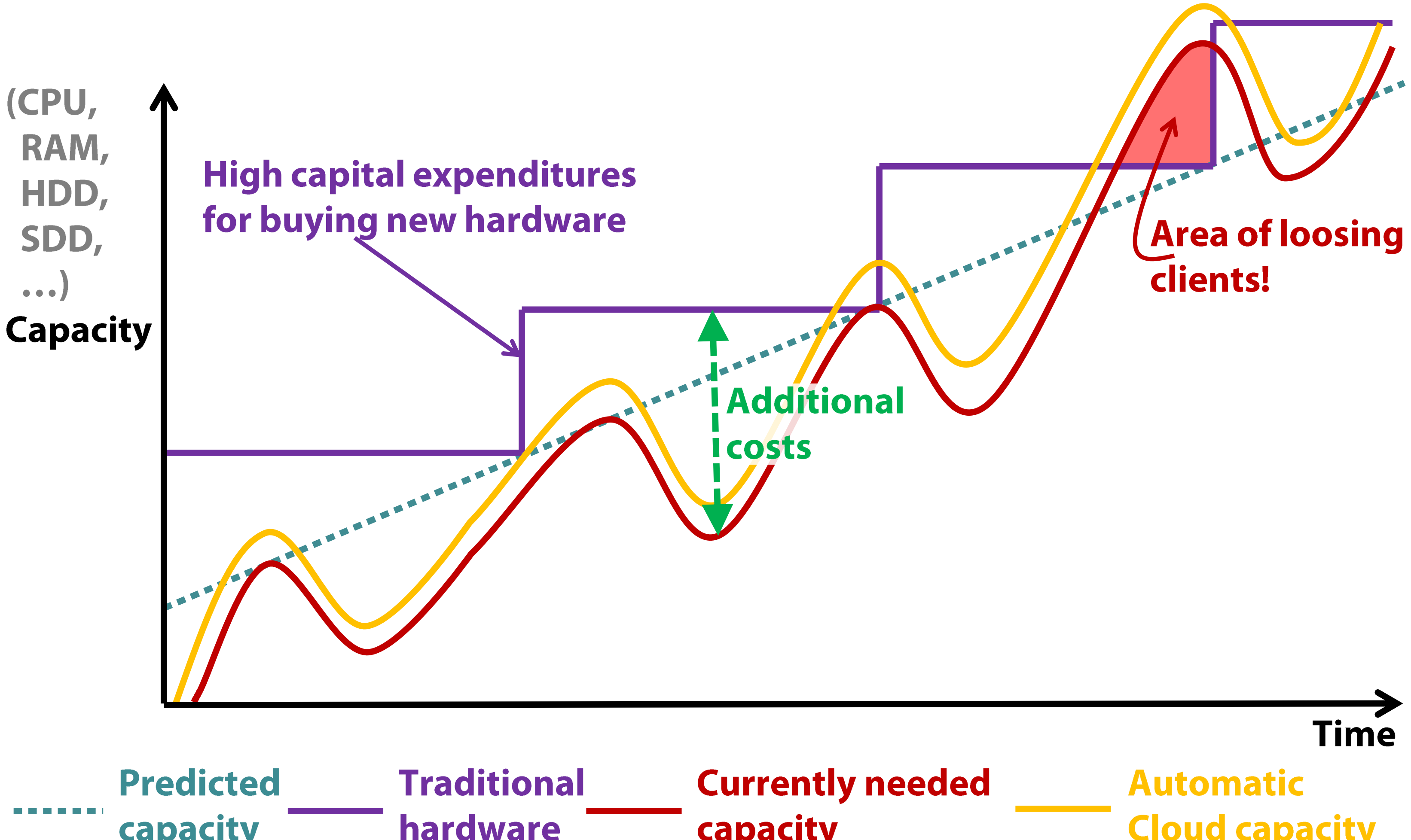


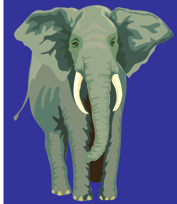
Cloud Computing Architecture



- Large cluster with up to several thousand nodes
- Replication of data blocks (default 3 times)
- Simple error detection and recovery by job repetition

Capacity-Cost Performance

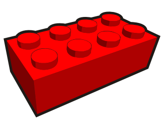




Cloud DBMS

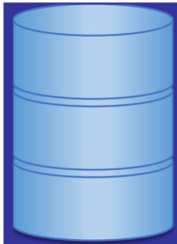
Scalability
(especially for Updates)

- Petabytes of data
- Thousands of Computers



Flexibility

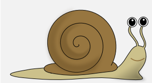
- Processing of any data format
- schemaless/without schema



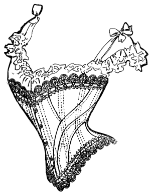
Traditional DBMS

High performance

- only for read-heavy workloads

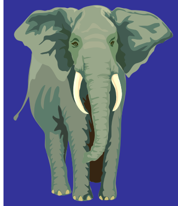


Updates are relatively slow

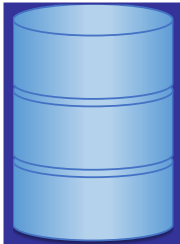


Uniform Data format

- Separation of schema and content

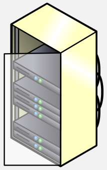


Cloud DBMS



Traditional DBMS

(Relatively) cheap (commodity-) hardware

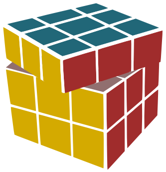


Few high-end server

- few hardware crashes

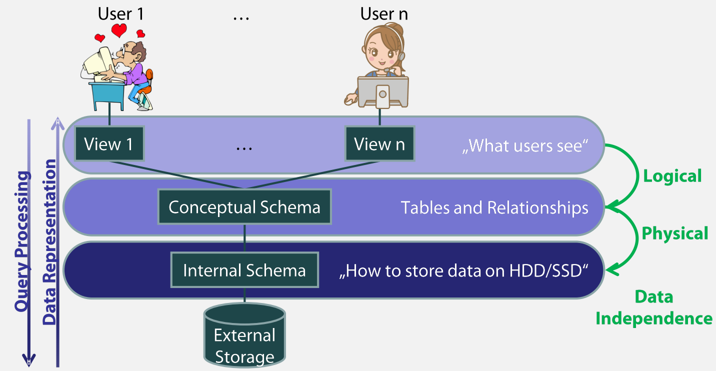
Efficient and simple fault-tolerant mechanisms

- Dealing with frequent errors (hardware/communication)

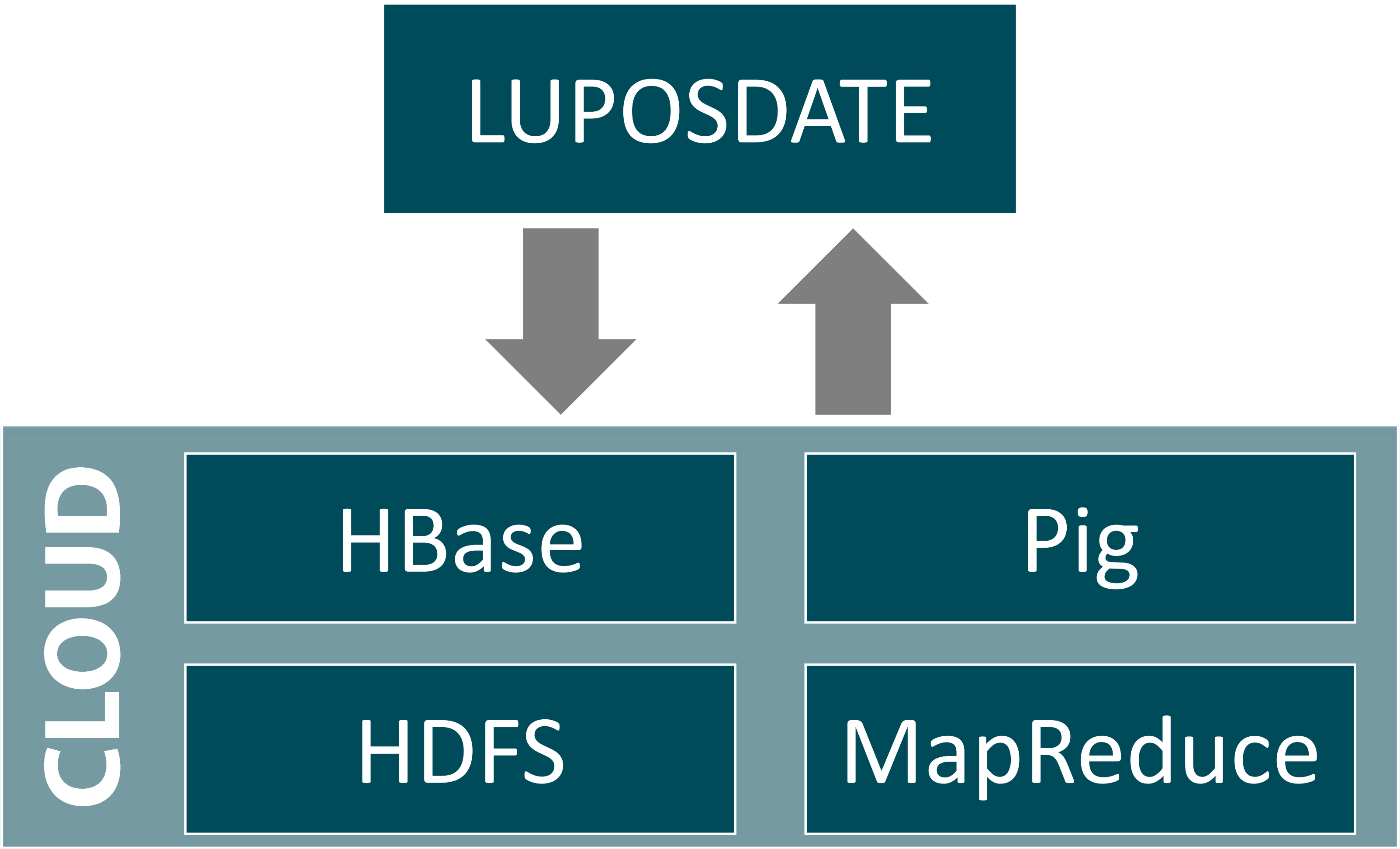


Transactions: Garanty of AtomicityConsistentIsolationDurable properties

- Assumption: Error case is seldom

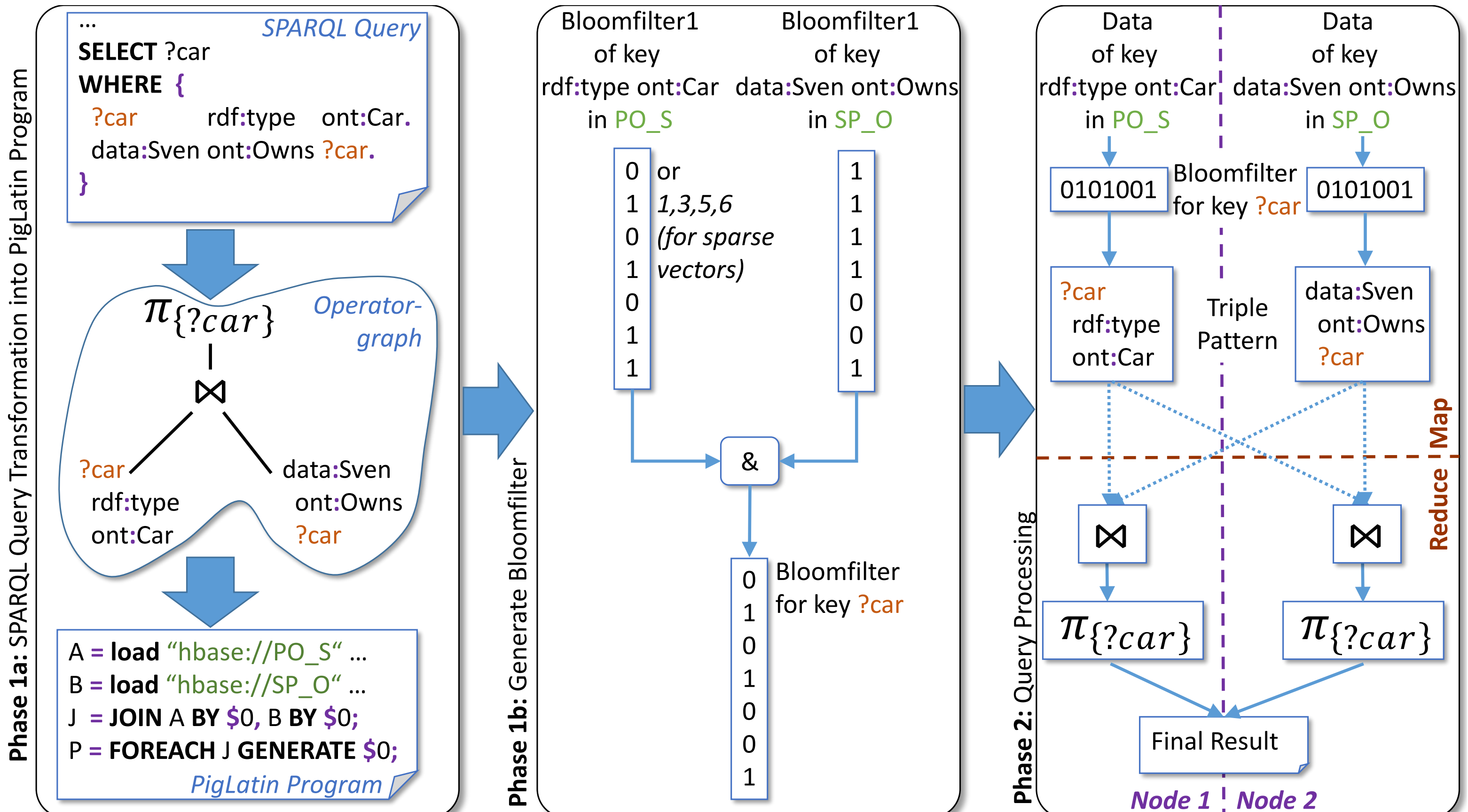


P-LUPOSDATE - Stack

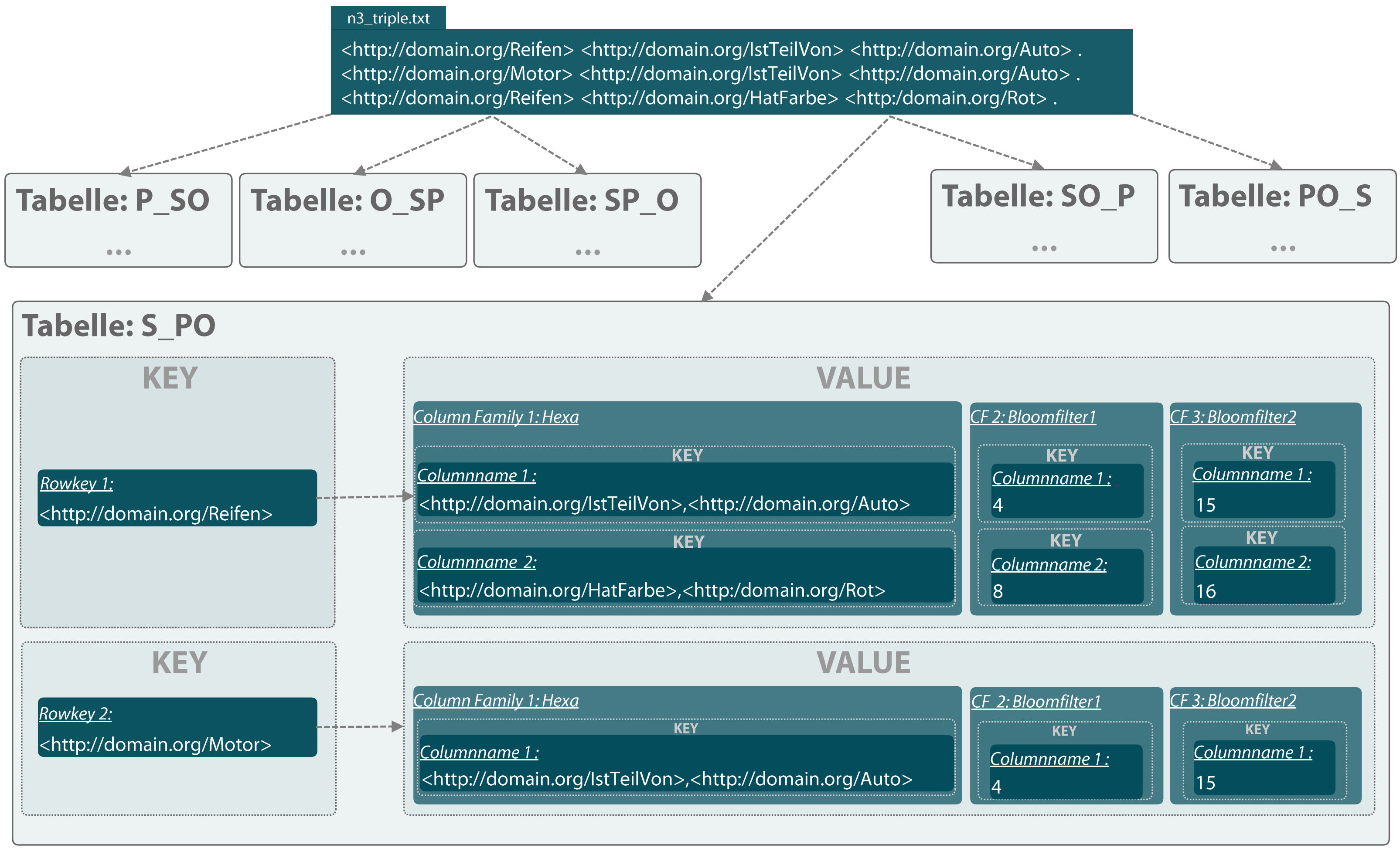


S. Groppe et al., P-LUPOSDATE: Using Precomputed Bloom Filters to Speed Up SPARQL Processing in the Cloud, OJSW, 1(2), 2014

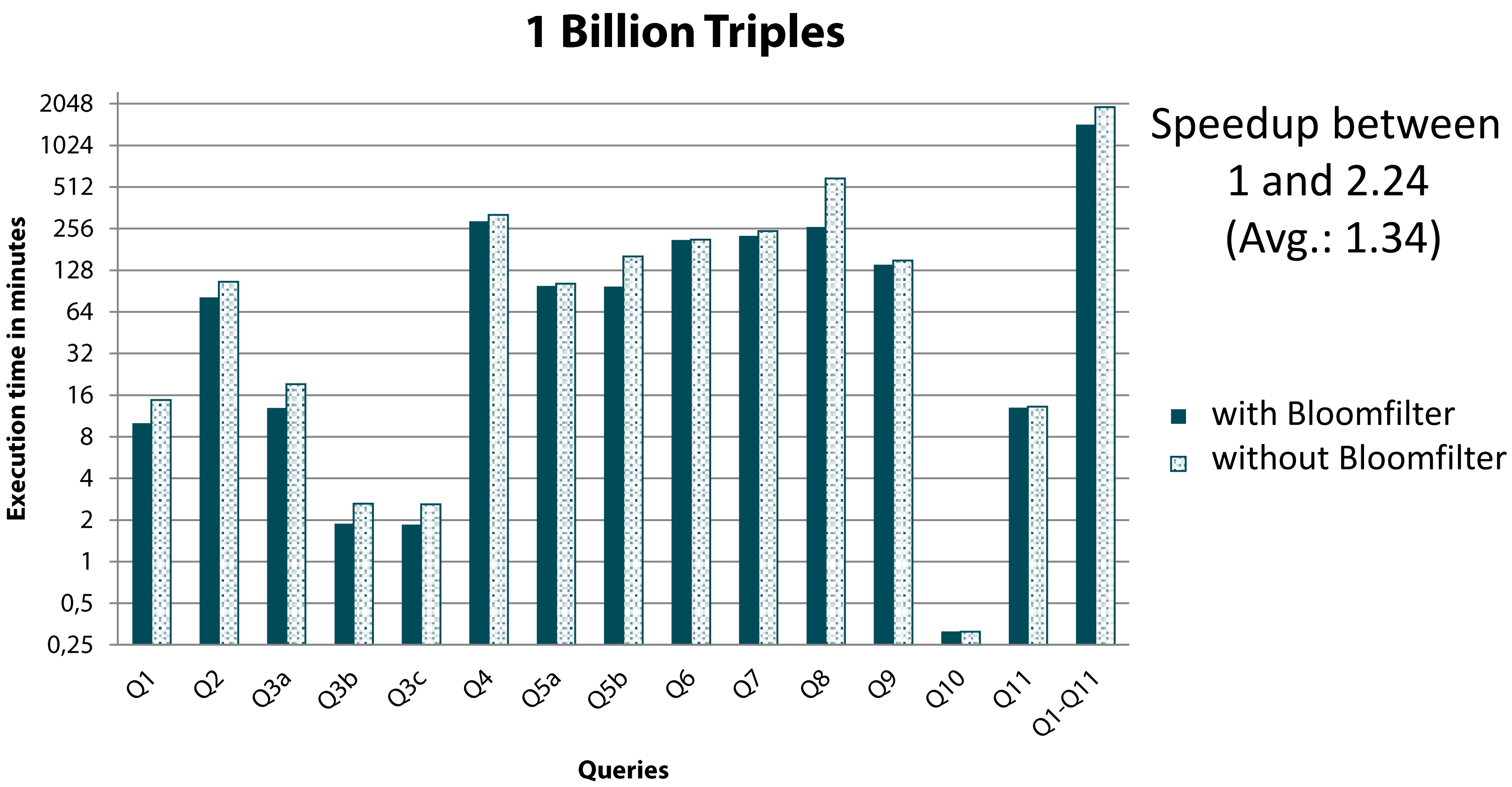
P-LUPOSDATE - Bloomfilter and Query Processing



P-LUPOSDATE - Indexing Scheme



P-LUPOSDATE - Experimental Evaluation



Speedup between 1 and 2.24 (Avg.: 1.34)

Typical Big Data Analytics Stack (e.g. Spark, Flink, Storm)

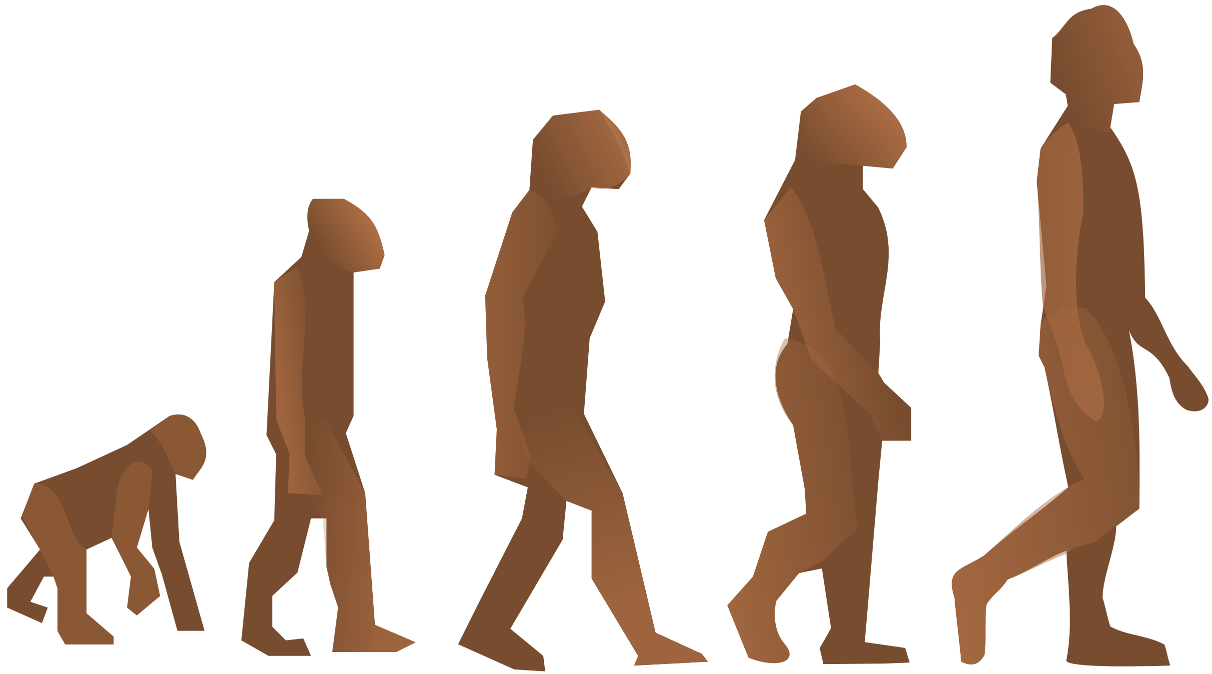
Streaming Processing	Scripting Language/ Applications	Machine Learning	Graph Processing
Spark: Streaming **Flink: DataStream**	**Spark: Spark SQL** **Flink: Table API & SQL**	**Spark: MLib** **Flink: FlinkML**	**Spark: GraphX** **Flink: Gelly**

Execution Engine

Resource Management

Database | Storage | Streams

Evolution of Big Data Analytics Engines

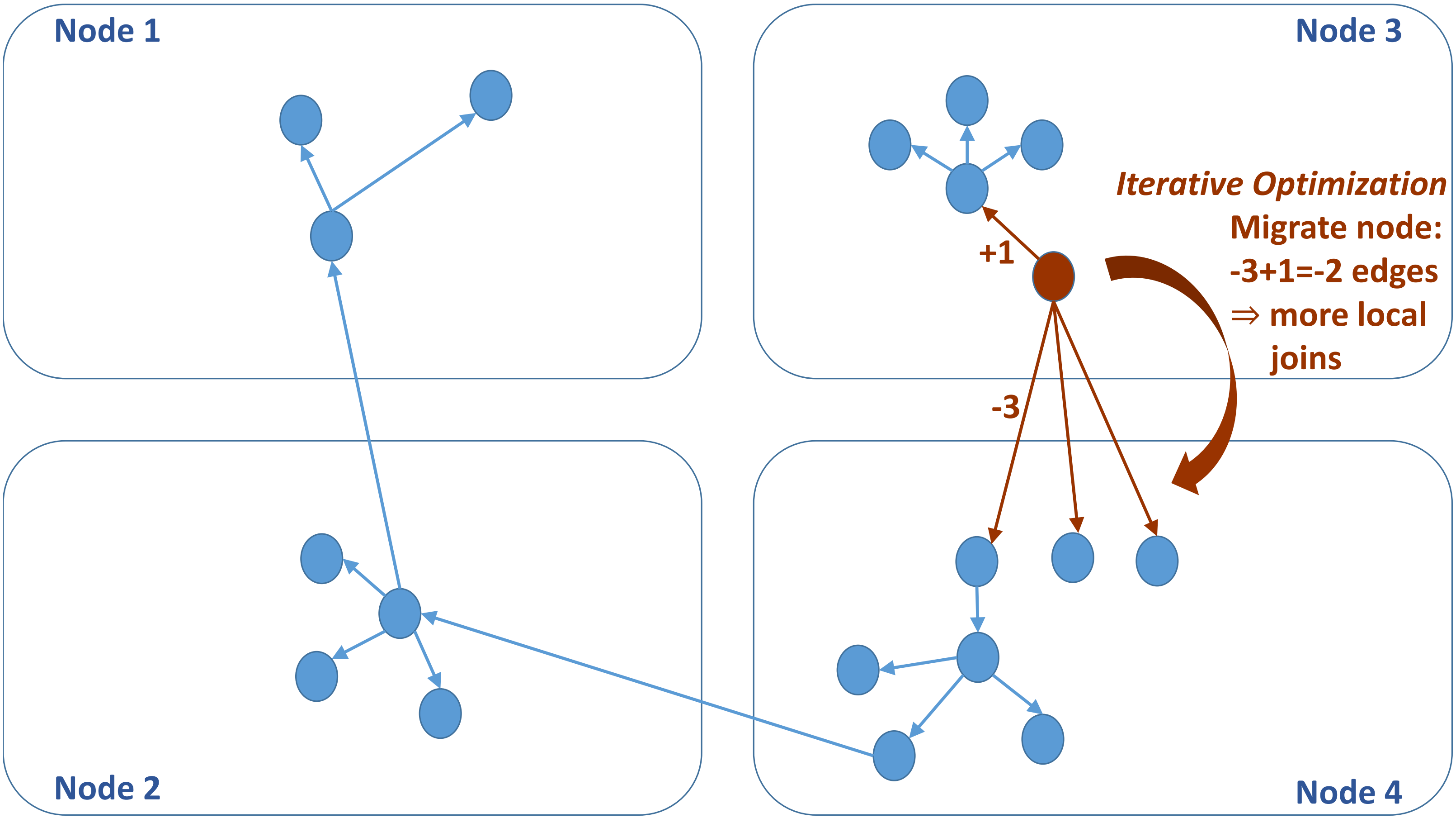


	1. Generation	2. Generation	3. Generation	4. Generation	5. Generation
Features	Batch	+ Interactive	+ Near-Real-Time[1] + Iterative Processing	+ Real-Time Streaming + Native It. Processing	?
Processing Model	MapReduce	DAG Dataflows	Resilient Distributed Datasets (RDD)	Cyclic Dataflows	?
Engine	Hadoop	TEZ	Spark	Flink	?

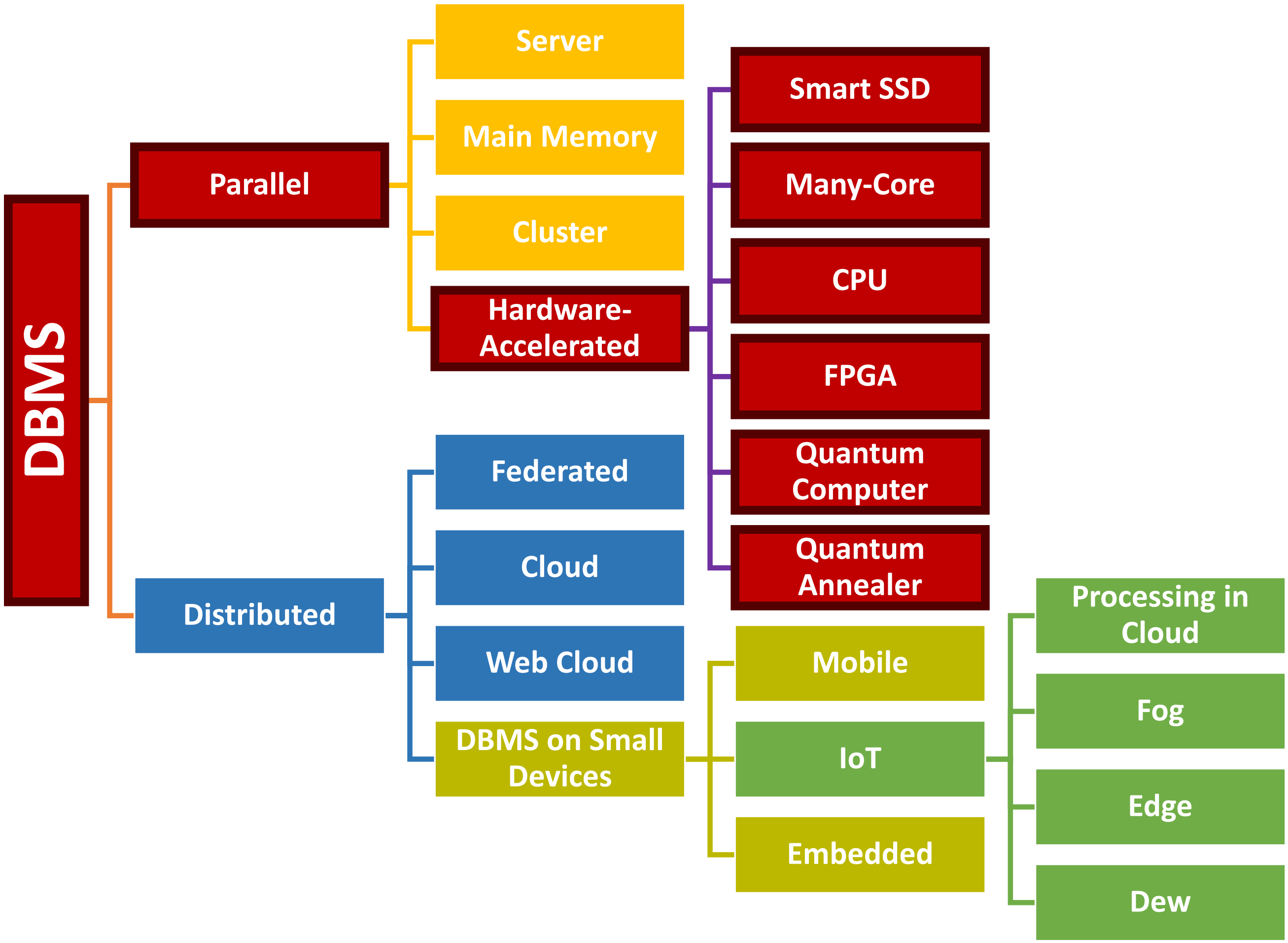
[1] via Mini-Batches

S. Groppe: Emergent models, frameworks, and hardware technologies for Big data analytics. The Journal of Supercomputing, Vol.76(3), 2020.

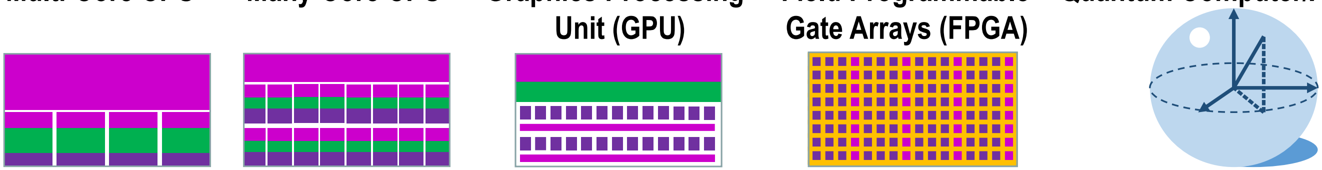
What is missing: Maximizing local Joins



Platform-specific types of DBMS

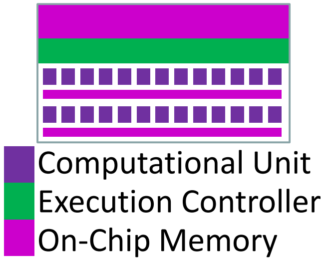


Architectures of Emergent Hardware



	Multi-Core CPU	Many-Core CPU	Graphics Processing Unit (GPU)	Field Programmable Gate Arrays (FPGA)	Quantum Computer/Annealer
Cores	~10	~100	~1000	~100 000	~50 / ~2000 qubits
Core Complexity	Complex (optimized for single thread performance)			Simple	Complex
Computational Model	MIMD + SIMD		SIMD	Data-Flow	Universal/Adiabatic Quantum computing
Parallelism	Thread and Data Parallel		Data Parallel	Arbitrary	Quantum Logic Gates/ Fluctuation (glob. Min.)
Memory Model	Shared	Distributed			Quantum Superposition
Power	150 W	200 W	250 W	50 W	25 KW
Database Op.	Query Optimization (Enumeration of Plans), Concurrency Control				
	Efficient Processing Of				
	Query Processing				

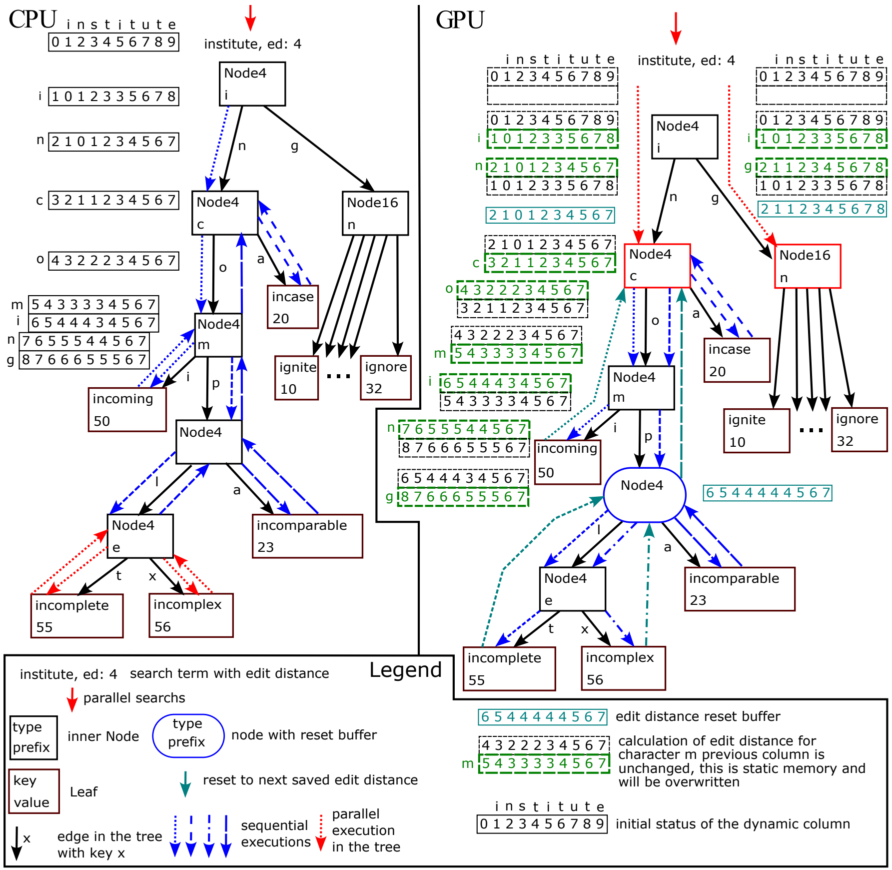
Legend: Computational Unit; Execution Controller; Interconnection Network; On-Chip Memory



General Purpose Graphics Processing Unit (GPGPU)

- turns the massive computational power of a modern graphics accelerator's shader pipeline into general-purpose computing power
- Single instruction, multiple data (SIMD)
- Up to several thousand computing cores
- Programming languages for SIMD computations
  - Open Computing Language (OpenCL): Vendor-independent programming standard
  - CUDA (formerly Compute Unified Device Architecture): NVIDIA-dependent parallel computing platform and API model
  - Open Graphics Library (OpenGL): mainly cross-language, cross-platform API for rendering 2D and 3D vector graphics

Approximate Search in Adaptive Radix Tree (ART) on GPGPUs



- **Levenshtein-distance**: number of operations to transform one string into another:

		i	n	s	t	i	t	u	t	e
	0	1	2	3	4	5	6	7	8	9
i	1	0	1	2	3	3	5	6	7	8
n	2	1	0	1	2	3	4	5	6	7
c	3	2	1	1	2	3	4	5	6	7
o	4	3	2	2	2	3	4	5	6	7
m	5	4	3	3	3	3	4	5	6	7
i	6	5	4	4	4	3	4	5	6	7
n	7	6	5	5	5	4	4	5	6	7
g	8	7	6	6	6	5	5	5	6	7

e.g.
5 operations are needed to transform "institu" into "incom" or vice versa

- Speedup over 4 dependent on ART properties (1.43 for real-world BTC data)

High-End Parallel GPU System: DGX-2

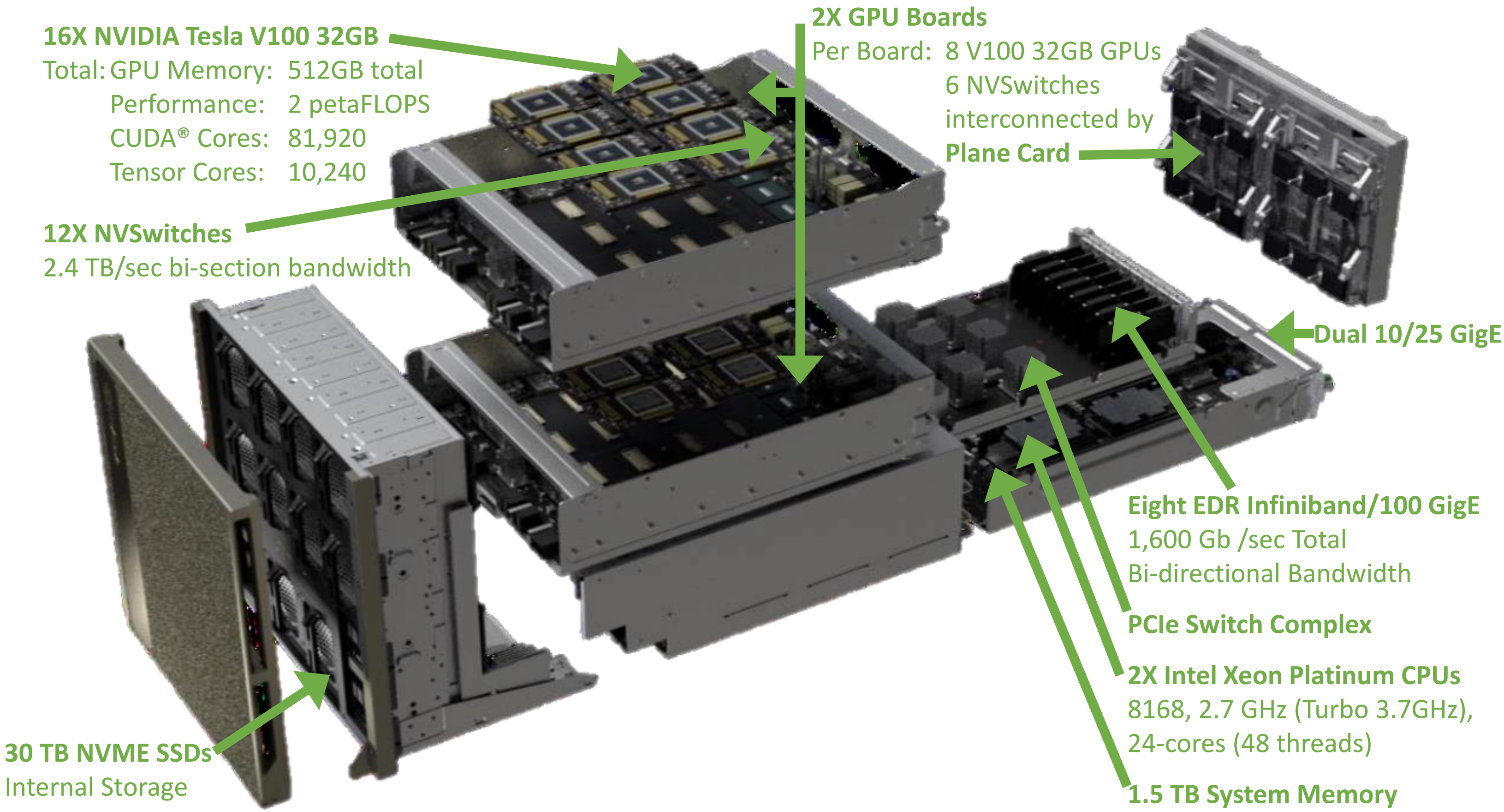
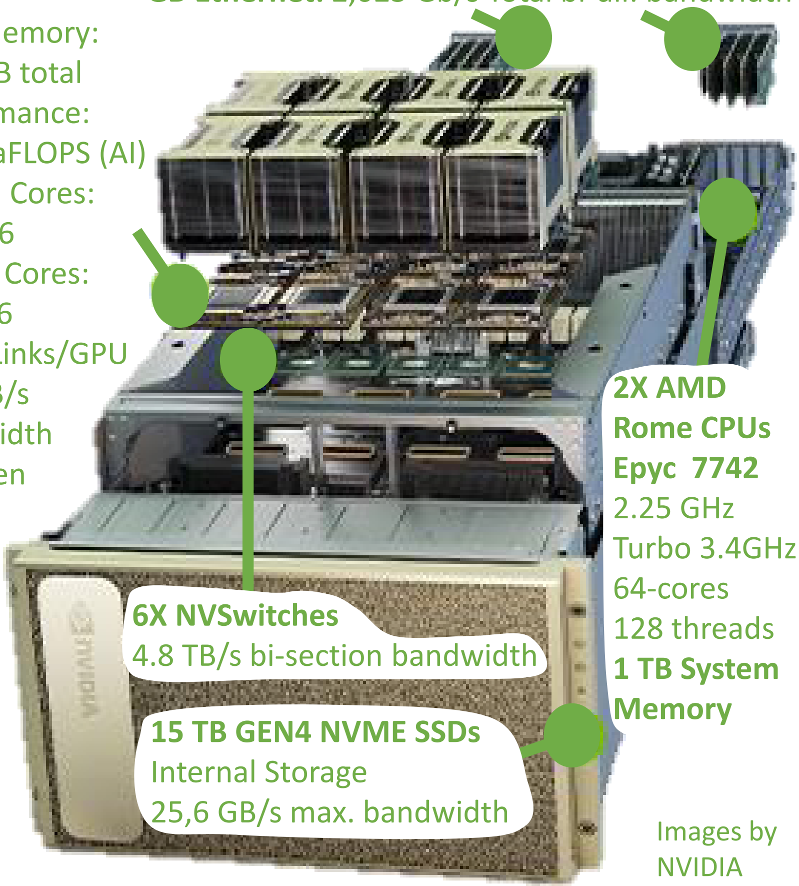


Image by NVIDIA

High-End Parallel GPU System: DGX A100

8X NVIDIA A100
Total:
GPU Memory:
320GB total
Performance:
5 petaFLOPS (AI)
CUDA® Cores:
55,296
Tensor Cores:
3,456
12 NVLinks/GPU
600 GB/s bandwidth between GPUs

9 Mellanox ConnectX-6 VPI HDR InfiniBand/200 GB Ethernet: 2,025 Gb/s Total bi-dir. bandwidth

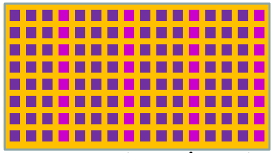


6X NVSwitches
4.8 TB/s bi-section bandwidth

15 TB GEN4 NVME SSDs
Internal Storage
25,6 GB/s max. bandwidth

2X AMD Rome CPUs Epyc 7742
2.25 GHz
Turbo 3.4GHz
64-cores
128 threads
1 TB System Memory

Images by NVIDIA

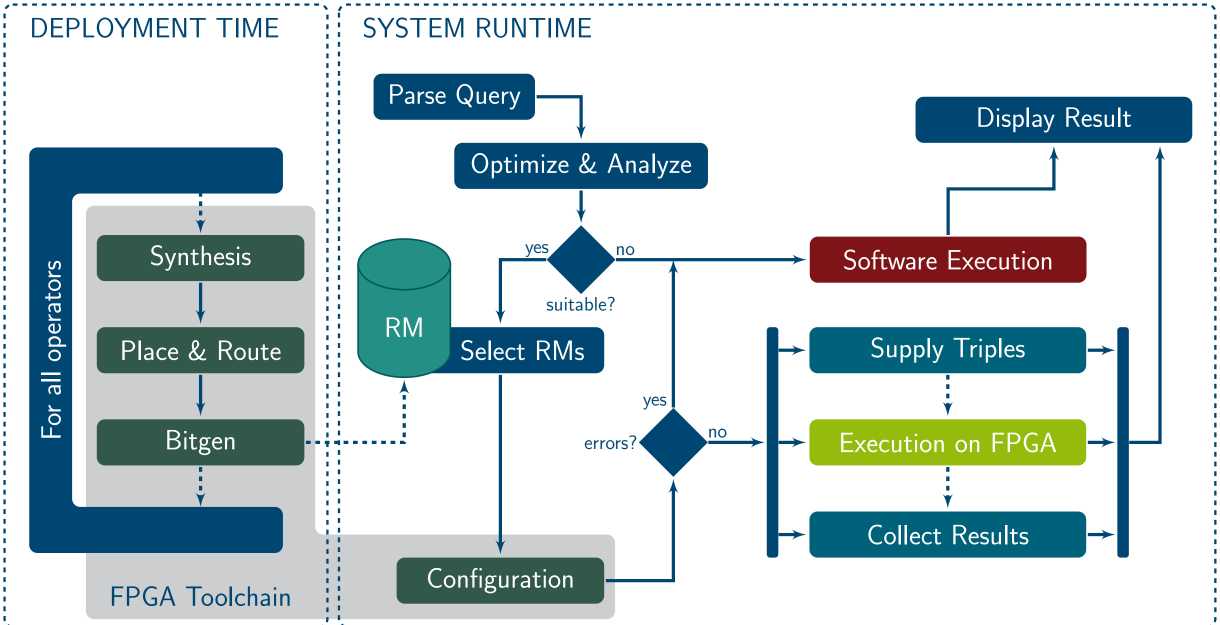


Computational Unit
Interconnection Network
On-Chip Memory

Field-Programmable Gate Array (FPGA)

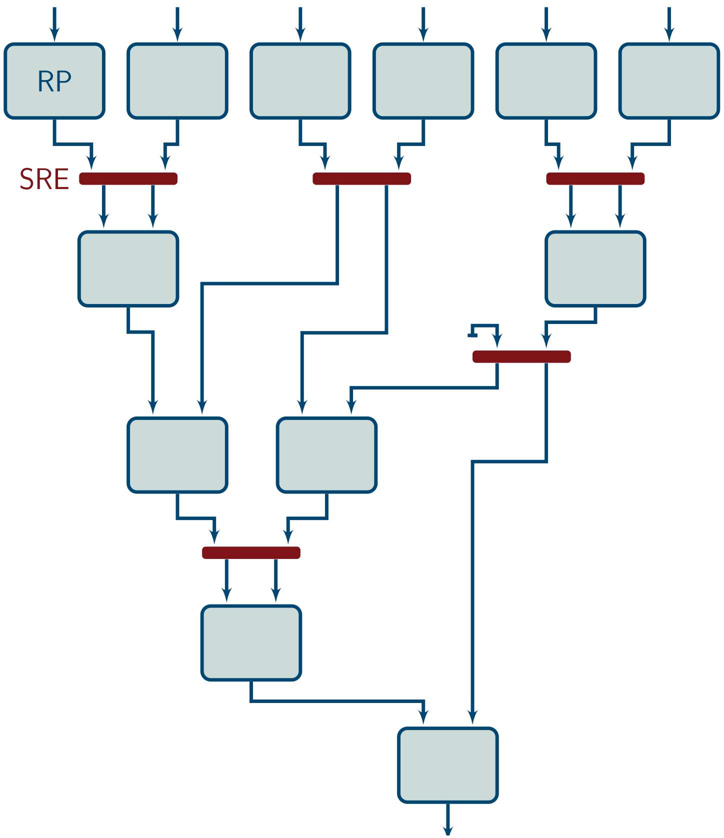
- contains an array of programmable logic blocks and a hierarchy of reconfigurable interconnects
- Specification of configuration typically by hardware description language (HDL)
- Recently High Level Synthesis (e.g., OpenCL) more mature (but still performance-critical parts should not be implemented in OpenCL)
- Long synthesis time

LUPOSDATE on FPGA – Query Processing



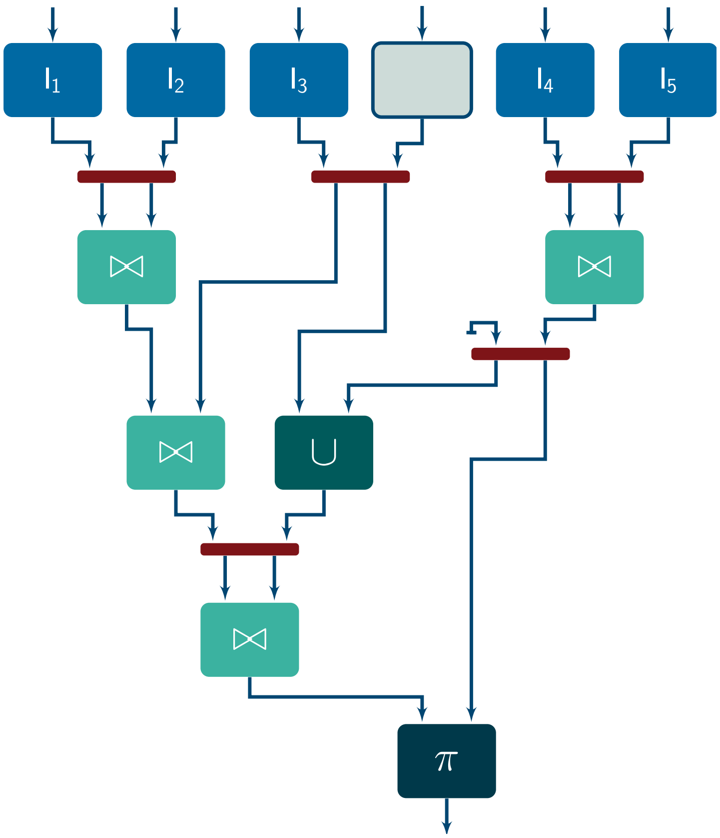
- Generation of Reconfigurable Modules (RMs) at system deployment time
- Selection of RMs and configuration into Reconfigurable Partitions at system runtime ⇝ avoids long synthesis time

Configuring the Semi-Static Operator Graph



SP^{2}B

Query 4

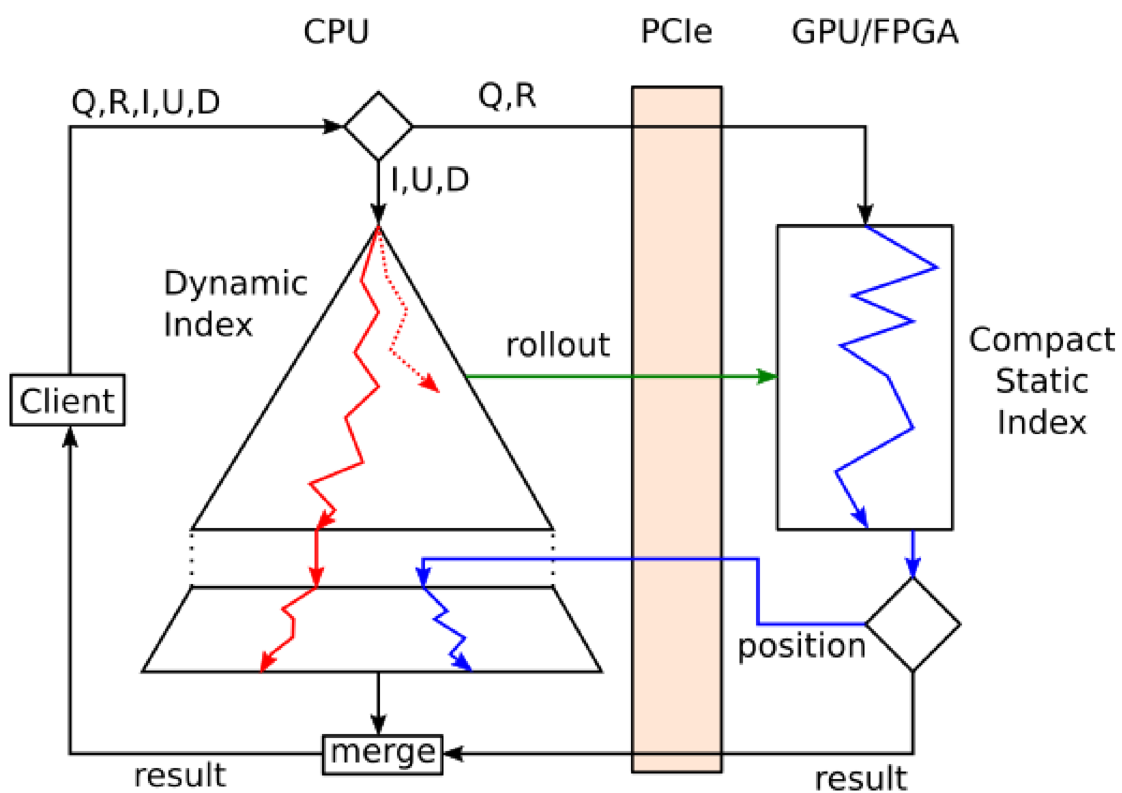


RP: Reconfigurable Partition
SRE: Semi-static Routing Element

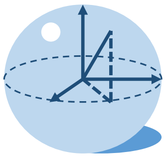
LUPOSDATE on FPGA – Benchmark Results

- Reconfiguration reduced from about half hour to few milliseconds (< 20 ms for all queries) when using semi-static operator graphs
- SP^{2}B Benchmark
  - Dataset sizes from 66 to 262 million triples
  - **Speedups between 4 and 32 times** (dependent on query and dataset size)

Hybrid Index - FPGA Accelerated Index



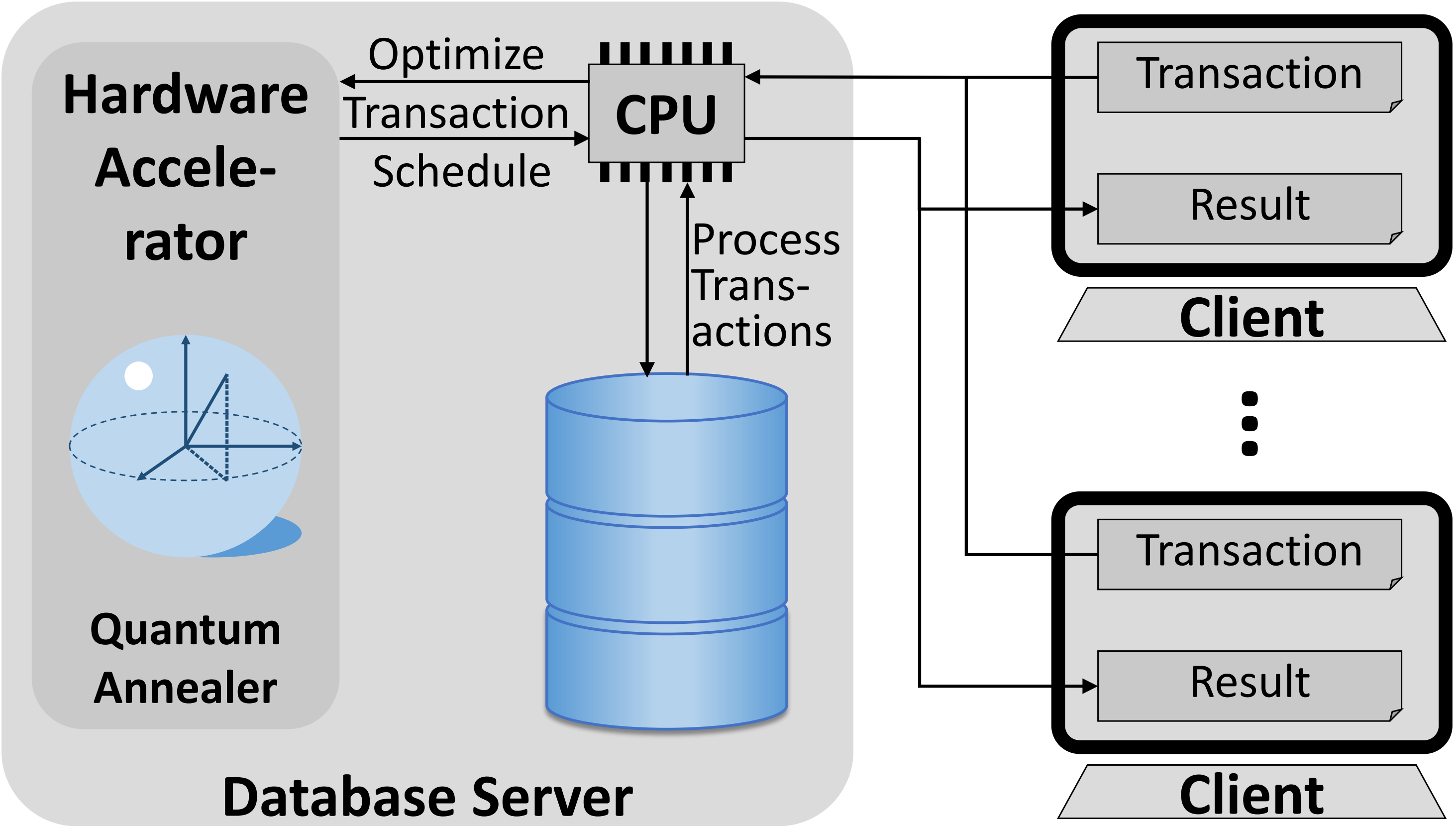
- B^+-Tree (compact static index: CSB^+-Tree): Speedup of 2.3
  Larger speedups possible via pipelining and
  usage of memory hierarchies (currently only BRAM)



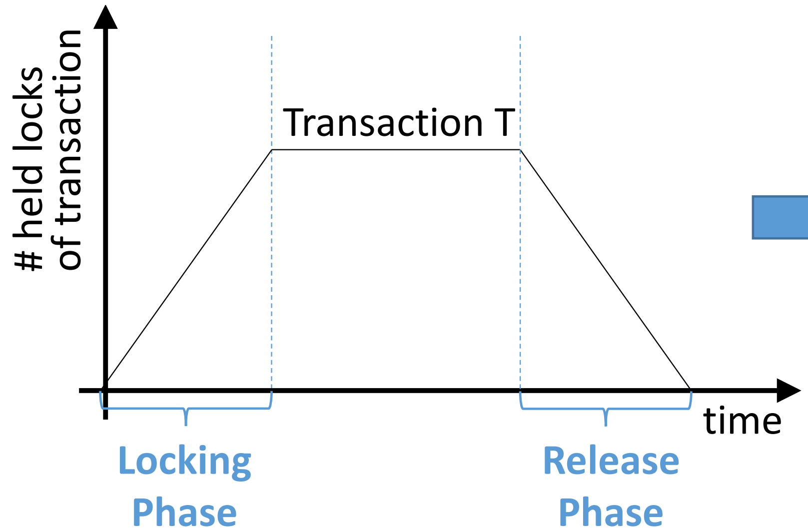
Quantum Computer

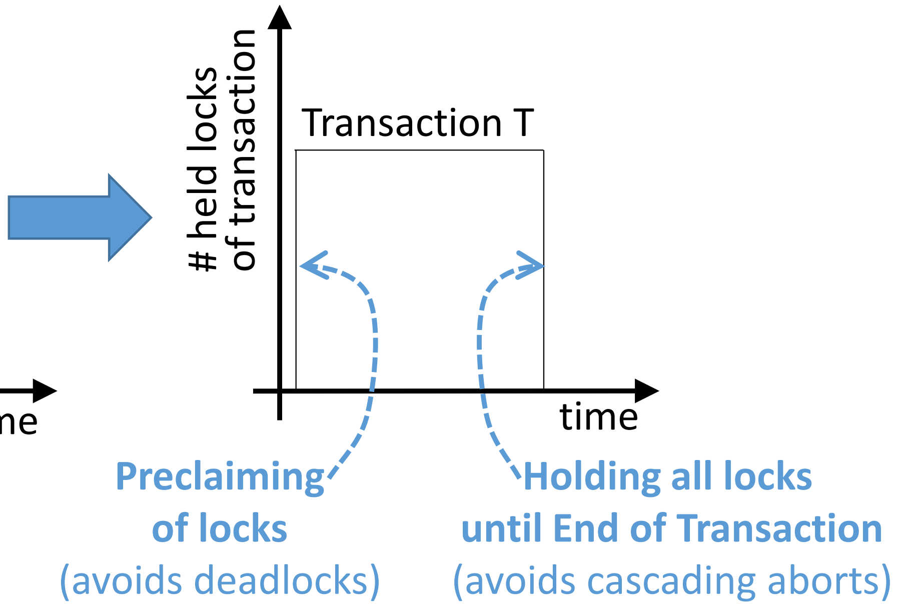
- use of quantum-mechanical phenomena such as superposition and entanglement to perform computation
- Different types of quantum computer, e.g.
  - Digital Quantum Computer
    - uses quantum logic gates to do computation
    - measurement (sometimes called observation) assigns the observed variable to a single value
  - Quantum Annealing
    - metaheuristic for finding the global minimum of a given objective function over a given set of candidate solutions
    - i.e., some way to solve a special type of mathematical optimization problem

Using Hardware Accelerator for optimizing Transaction Schedules



2 Phase Locking (2PL) versus Strict Conservative 2PL

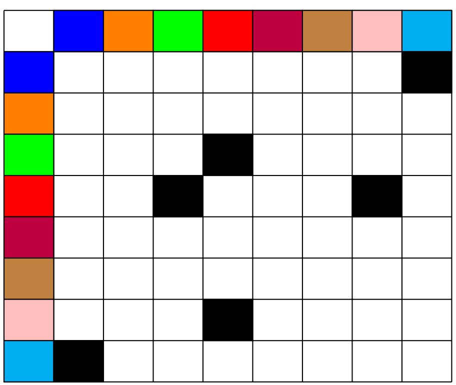




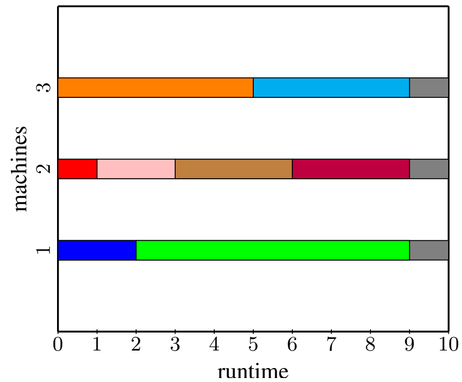
- required locks to be determined by
  - static analysis of transaction, or if static analysis is not possible:
  - an additional phase at runtime before transaction processing
    - A. Thomson et al., "Calvin: Fast distributed transactions for partitioned database systems", SIGMOD 2012.

Optimizing Transaction Schedules

- Job shop schedule problem (JSSP):
  - Multi-Core CPU
    - Process whole job (here transaction) on core X
  - Schedule: $\forall$ cores: Sequence of jobs to be processed
  - What is the optimal schedule for minimal overall processing time?
- Additionally to JSSP:
  Blocking transactions not to be processed in parallel
- Example:



Black: Blocking transactions

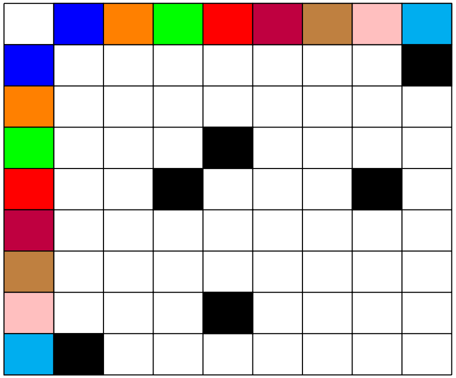


Transaction schedule

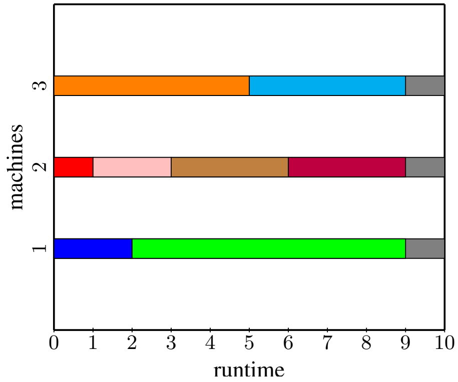
- JSSP is among the hardest combinatorial optimizing problems*
- $\Rightarrow$ Hardware accelerating the optimization of transaction schedules

Optimizing Transaction Schedules via Quantum Annealing

- Scenario: Strict conservative 2-Phase Locking
  - Preclaiming of all locks at *Begin of Transaction* (avoids deadlocks)
  - Holding all locks until *End of Transaction* (avoids cascading aborts)
- Solution formulated as set of binary variables
  - $X_{i,j,s}$ is 1 iff transaction t_i is started at time s on machine m_j, otherwise 0
- Example:



Black: Blocking transactions



Transaction schedule

- Solution:
  $X_{1,1,0}$, $X_{3,1,2}$, $X_{4,2,0}$, $X_{7,2,1}$, $X_{6,2,3}$, $X_{5,2,6}$, $X_{2,3,0}$, $X_{8,3,5}$

Optimizing Transaction Schedules via Quantum Annealing

- Transaction Model
  - T: set of transactions with |T| = n
  - M: set of machines with |M| = k
  - $O \subseteq T \times T$: set of blocking transactions
  - l_i: length of transaction i
  - R: maximum execution time
  - upper bound $r_i = R - l_i$ for start time of transaction i
- Example
  - $T = \{t_1, t_2, t_3\}$, n=3
  - $M = \{m_1, m_2\}$, k=2
  - $O = \{(t_2, t_3)\}$
  - $l_1 = 2, l_2 = 1, l_3 = 1$
  - R = 2
  - $r_1 = 0, r_2 = 1, r_3 = 1$
- Quadratic unconstrained binary optimization (QUBO) problems (solving is NP-hard)
  - A QUBO-problem is defined by N weighted binary variables $X_1, ..., X_N \in 0, 1$, either as linear or quadratic term to be minimized:
    $\sum_{0<i\leq N} w_i X_i + \sum_{i\leq j\leq N} w_{ij} X_i X_j$, where $w_i, w_{ij} \in \mathbb{R}$

Optimizing Transaction Schedules via Quantum Annealing

- Valid Solution
  - A: each transaction starts exactly once

$$A = \underbrace{\sum_{i=1}^{n}}_{\text{transactions}} \left(\underbrace{\sum_{j=1}^{k}}_{\text{machines}} \underbrace{\sum_{s=0}^{r_i}}_{\text{start times}} X_{i,j,s} - 1 \right)^2$$

  - B: transactions cannot be executed at the same time on the same machine

$$B = \underbrace{\sum_{j=1}^{k}}_{\text{machines}} \overbrace{\sum_{i_1=1}^{n-1}}^{\text{transactions without } t_n} \underbrace{\sum_{s_1=0}^{r_{i_1}}}_{\text{start times}} \overbrace{\sum_{i_2=i_1+1}^{n}}^{\text{remaining transactions}} \underbrace{\sum_{s_2=q}^{p}}_{\text{invalid start times}} X_{i_1,j,s_1} X_{i_2,j,s_2} \quad \text{for } q = \max\{0, s_1 - l_{i_2} + 1\}, p = \min\{s_1 + l_{i_1}, r_{i_2}\}$$

  - C: transactions that block each other cannot be executed at the same time

$$C = \underbrace{\sum_{\{t_{i_1}, t_{i_2}\} \in O}}_{\text{blocking transactions}} \overbrace{\sum_{j_1=1}^{k}}^{\text{machines}} \underbrace{\sum_{s_1=0}^{r_{i_1}}}_{\text{start times}} \overbrace{\sum_{j_2 \in J}}^{\substack{\text{remaining} \\ \text{machines}}} \underbrace{\sum_{s_2=q}^{p}}_{\text{invalid start times}} X_{i_1,j_1,s_1} X_{i_2,j_2,s_2} \quad \text{for } J = \{1, ..., k\} \setminus \{j_1\}, q = \max\{0, s_1 - l_{i_2} + 1\}, p = \min\{s_1 + l_{i_1}, r_{i_2}\}$$

Optimizing Transaction Schedules via Quantum Annealing

- Optimal Solution
  - D: minimizing the maximum execution time

$$D = \sum_{i=1}^{n} \sum_{j=1}^{k} \sum_{s=0}^{r_i} w_{s+l_i} X_{i,j,s}, \text{ where } w_{s+l_i} = \frac{(k+1)^{s+l_i-1}}{(k+1)^R} < 1$$

  - Increasing weights: Weight of step n is larger than of all preceding steps 1 to n-1 $\Rightarrow$ preferring transactions ending earlier
  - Weigths in A, B and C $\geq$ 1
    $\Rightarrow$ first priority is validity, second priority is optimality
- Overall Solution
  - Minimize $P = A + B + C + D$

Optimizing Transaction Schedules via Quantum Annealing

- Experiments on real Quantum Annealer (D-Wave 2000Q cloud service)
  - first minute free (afterwards too much for our budget)
- Versus Simulated Annealing on CPU
- Preprocessing time/Number of QuBits: $O((n \cdot k \cdot R)^2)$

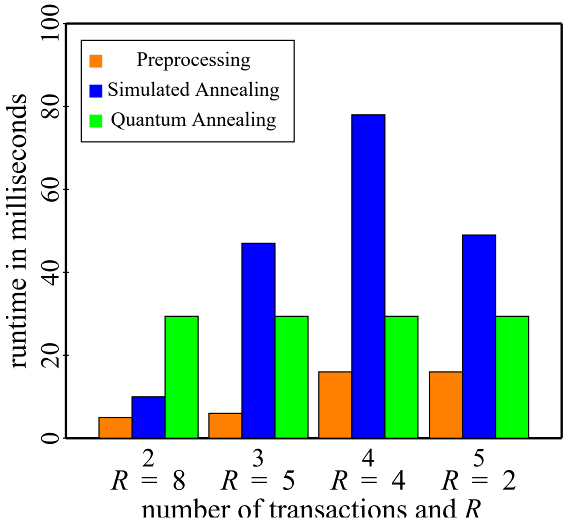
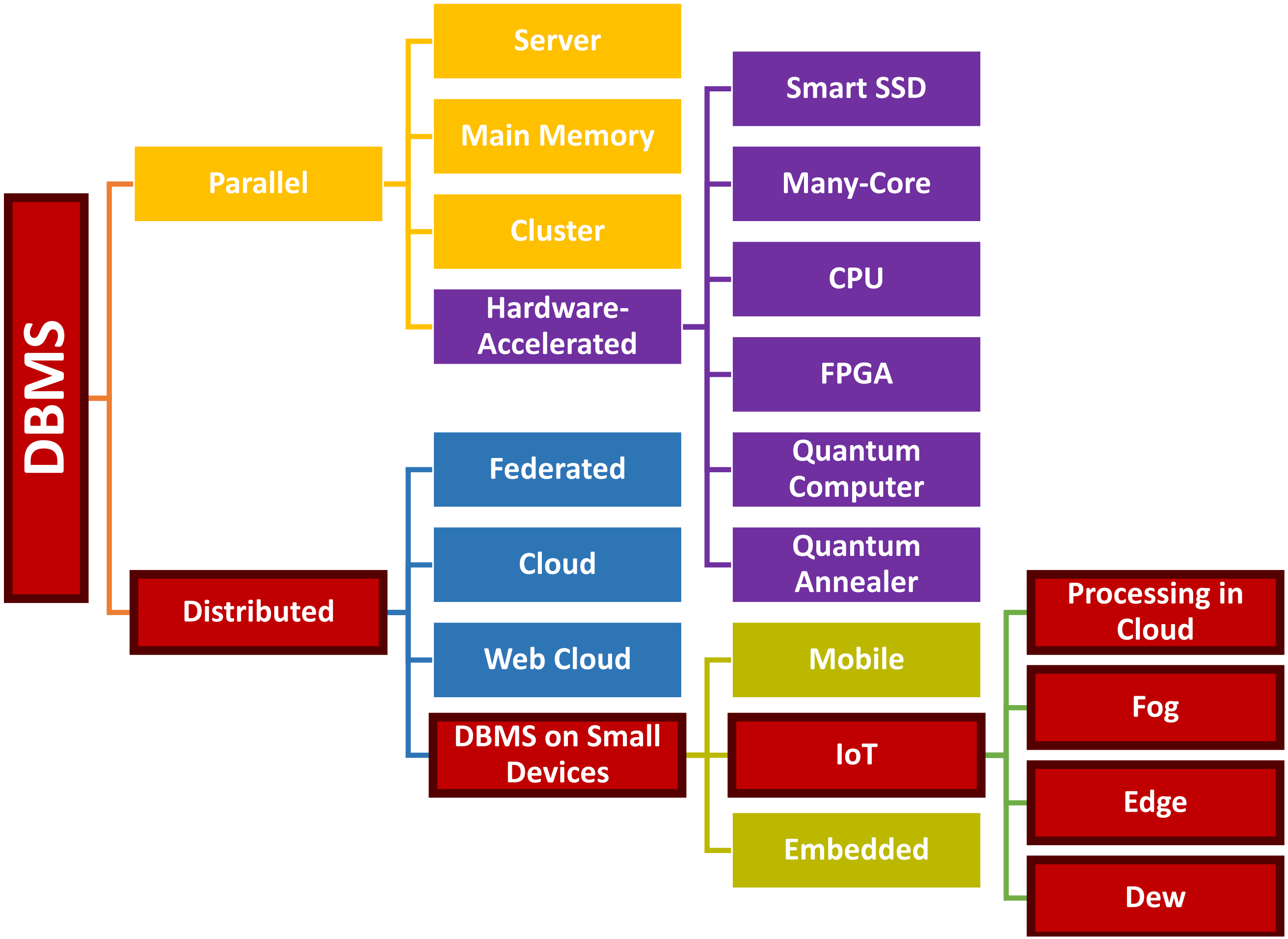
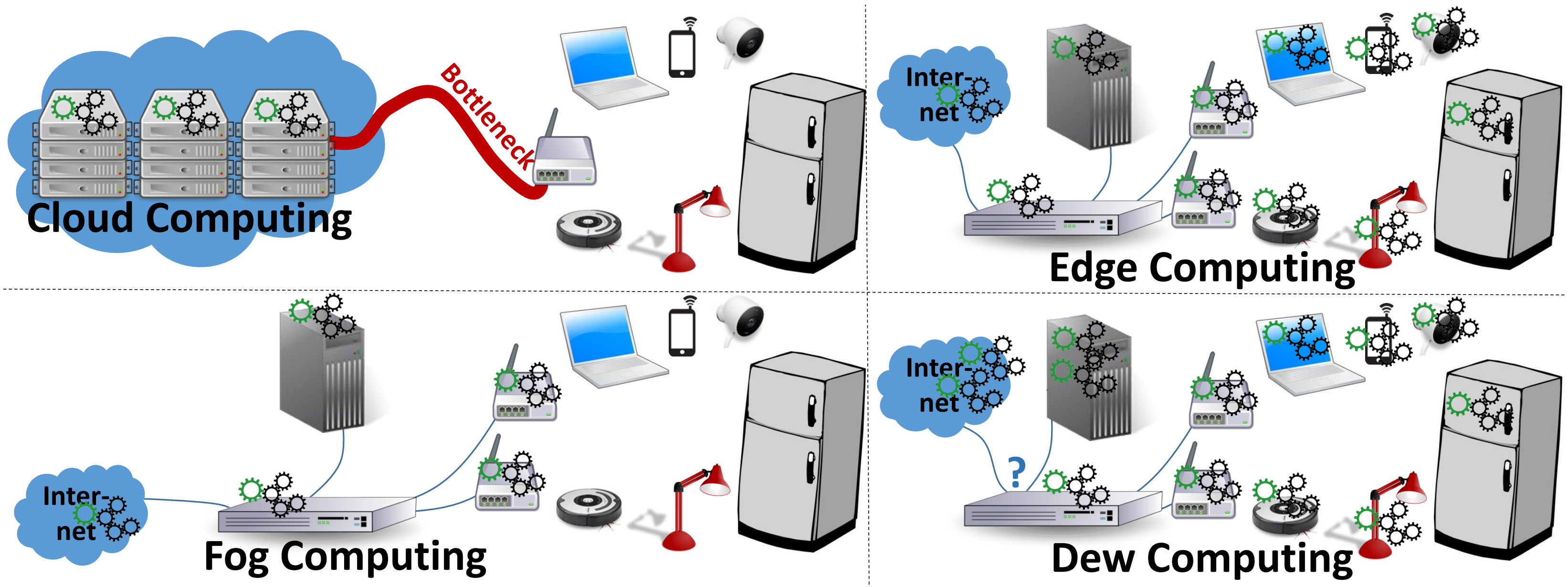


Fig.	k	n	R	O	$l_1, \ldots, l_n$	$r_1, \ldots, r_n$	req. var.
11	2	2	8	$\{\}$	8, 4	0, 4	8
		3	5	$\{(t_1, t_3)\}$	4, 5, 1	1, 0, 4	10
		4	4	$\{(t_2, t_4)\}$	3, 2, 1, 2	1, 2, 3, 2	16
		5	2	$\{(t_1, t_2), (t_4, t_5)\}$	1, 1, 1, 1, 1	1, 1, 1, 1, 1	10

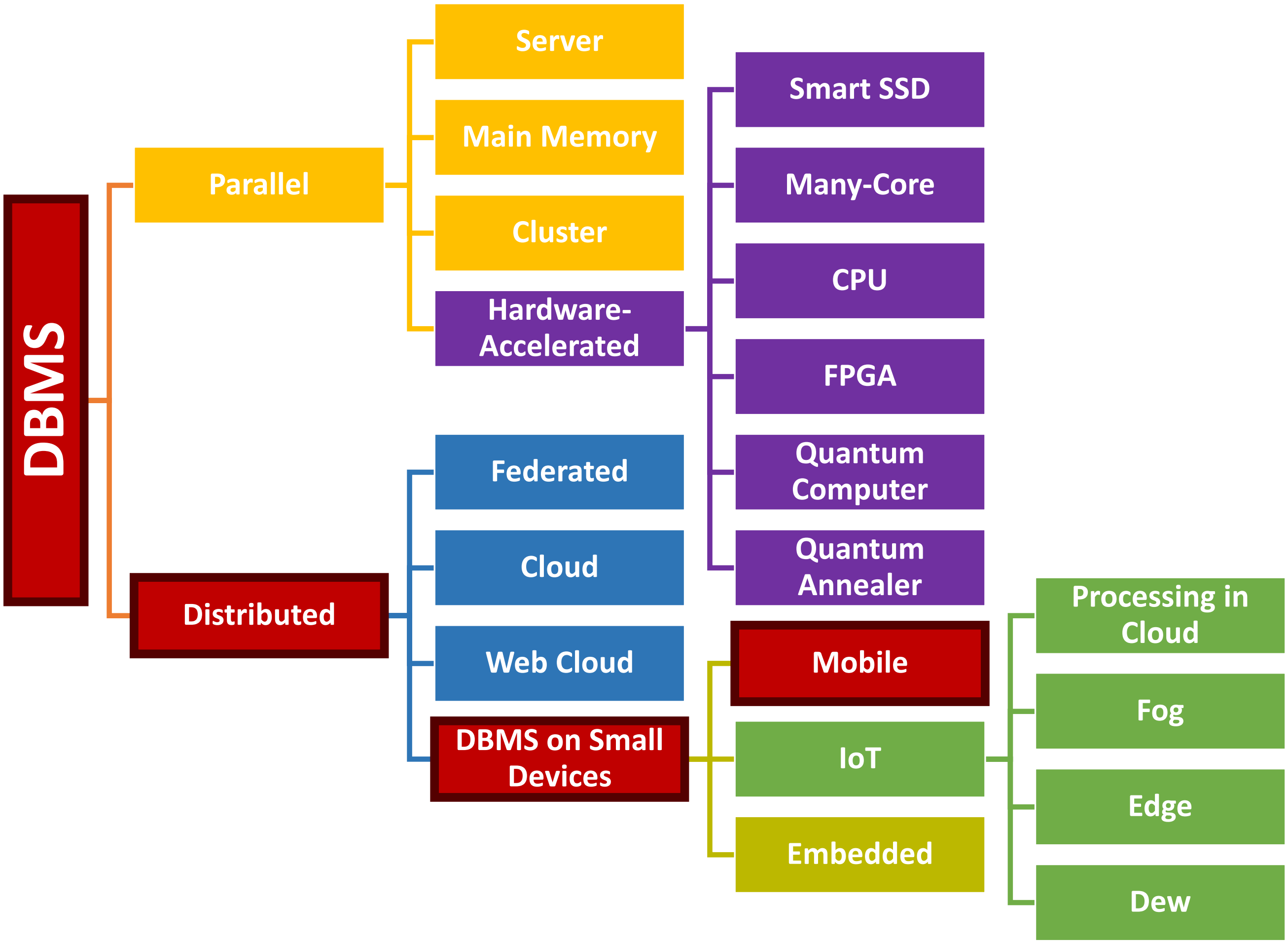
Platform-specific types of DBMS



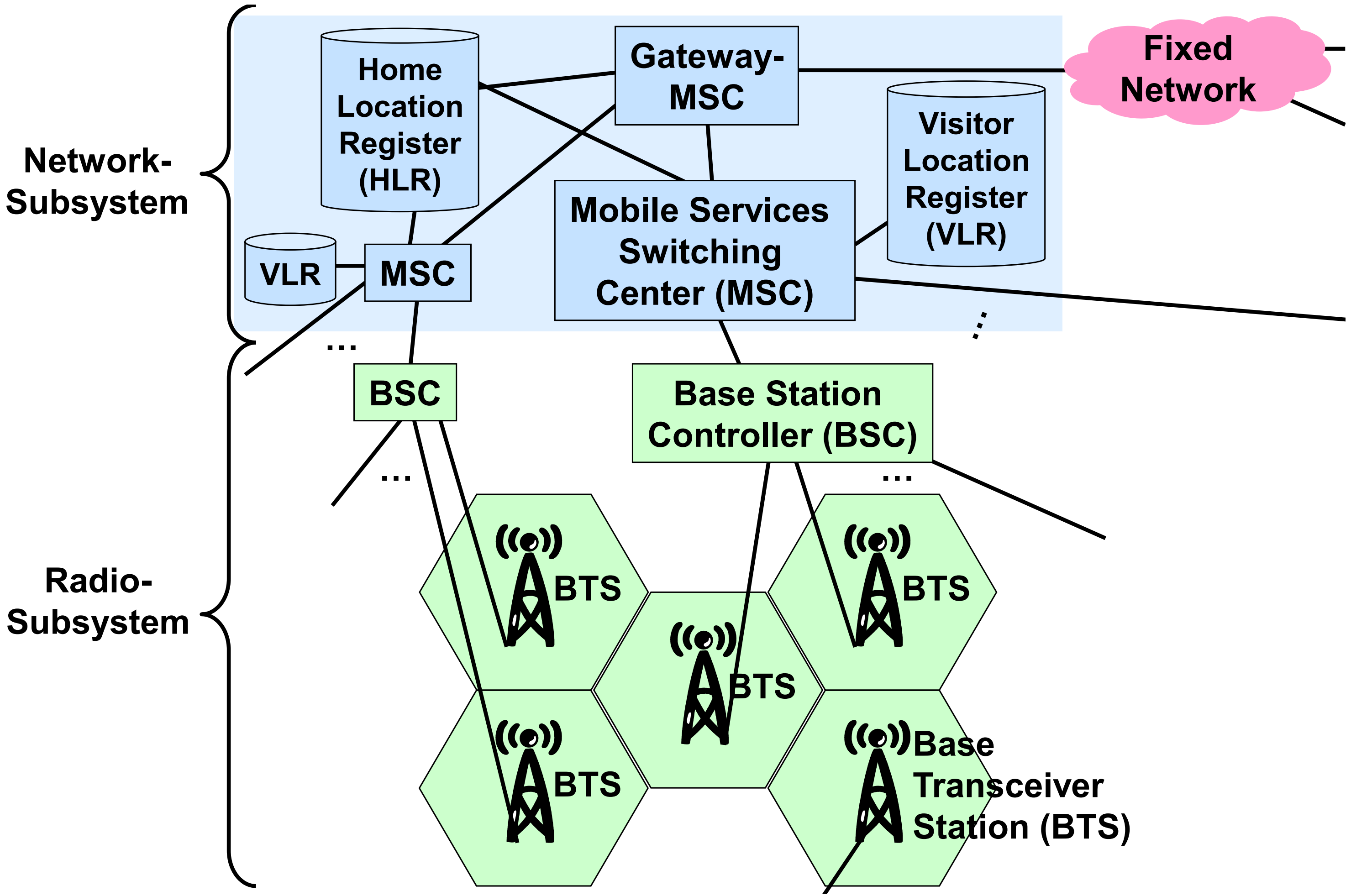
IoT Architectures



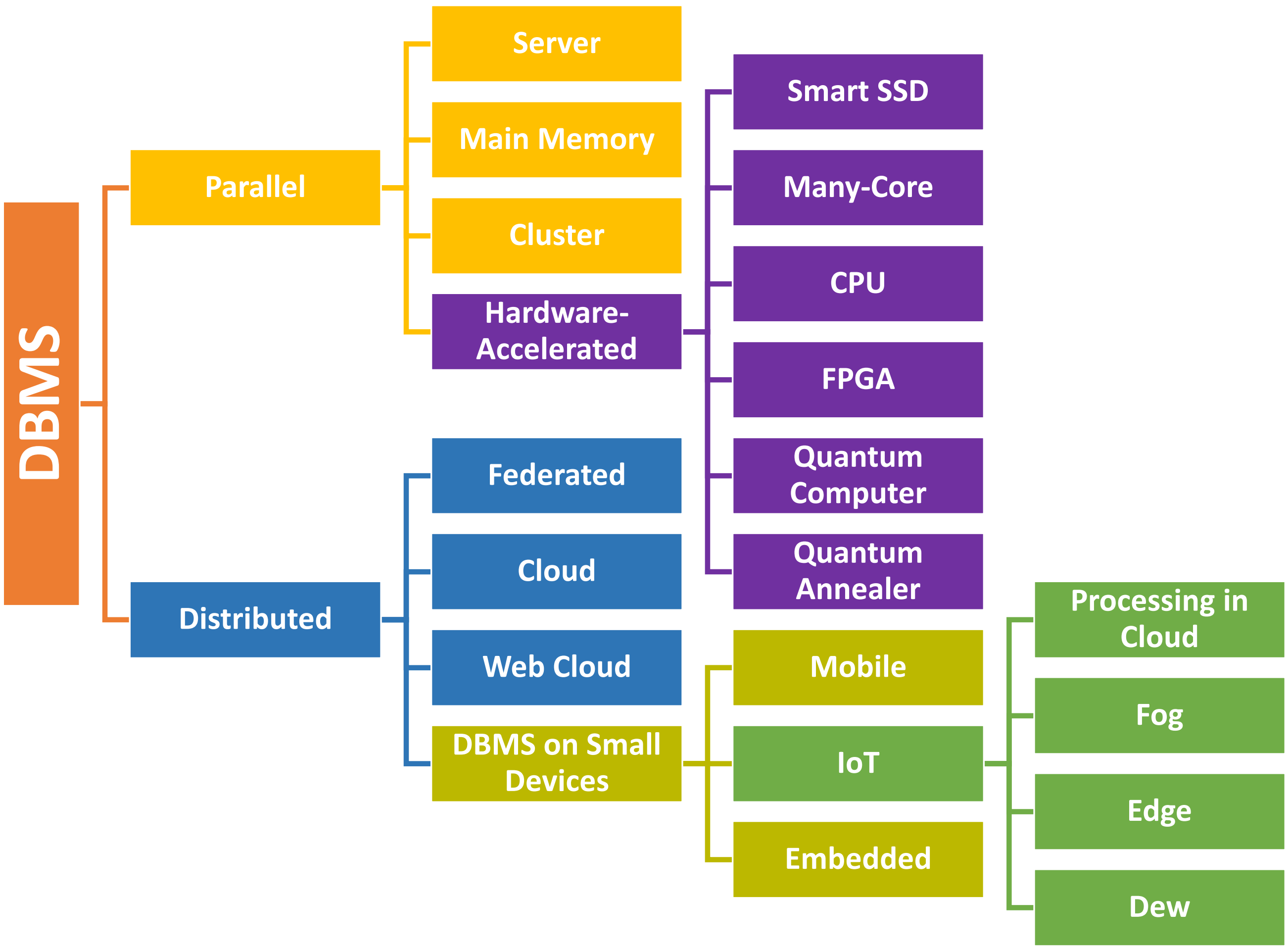
Platform-specific types of DBMS



Mobile DBMS integrated into Architecture for Mobile Phones



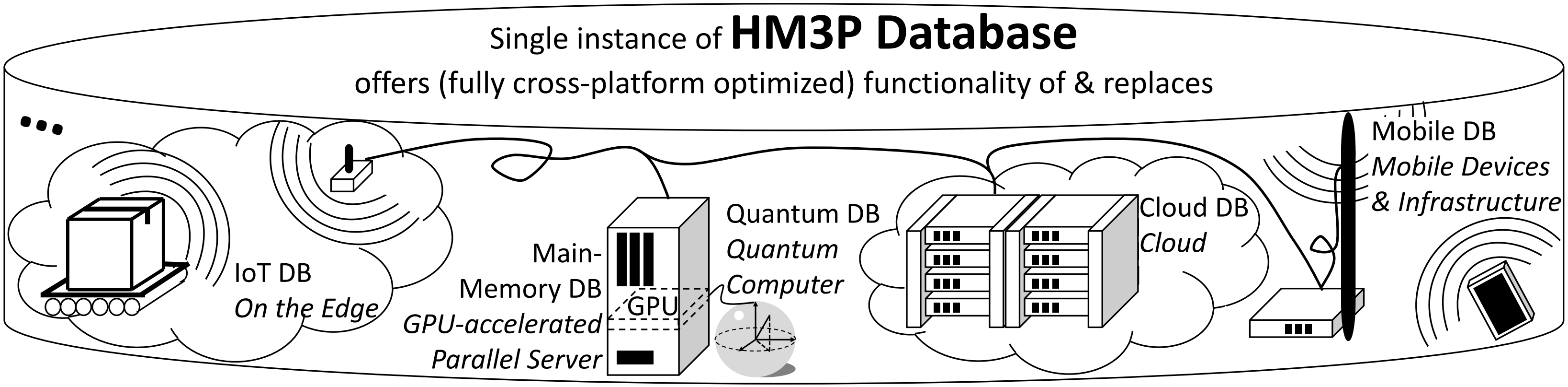
Platform-specific types of DBMS



Features of different types of databases

Feature \ DBMS	Main Memory	Parallel	Distributed	Federated	Cloud	Web Cloud	Mobile	IoT
Scalability	−	o	+	+	++	+++	+	++
Transaction rates	+++	++	o / +	o	++	+	−	−−
Intra-Transaction Parallelism	+++	++	o / +	− / o	+	o	−	−
Atomicity	+++	+++	++	+	+	+	+	+
Durability	+	+	+++	++	++	−	o	−
Consistency	+++	+++	++	+	+	+	+	+
Extensibility	−	+	o / +	o	++	+++	−	+++
Schemaless	−−−	−−−	−−−	−	+++	+++	+	+++
Availability	++	+	+	−	−	−−−	−−	−−−
Transparency of Distribution	++	++	+	o	++	−	−	−−
Geographical Distribution	−−	−	+	+	++	+++	++	++
Mobility	−	−	−	o	o	o	++	+
Node Autonomy	−−	−	o	+	o	−−	++	+
Heterogeneity of DBMS	−−	−	−	+	−	−	++	+++
Administration	o	o	−	− / −−	−	++	−−	−−−
Hardware Costs	−	−−	−	−	++	+++	−	+++
Reasoning	+++	+++	+	−−	++	+	−−	−−−

Hybrid Multi-Model Multi-Platform (HM3P) Database



- How to integrate the features of different types of databases into one single database running also on different platforms?

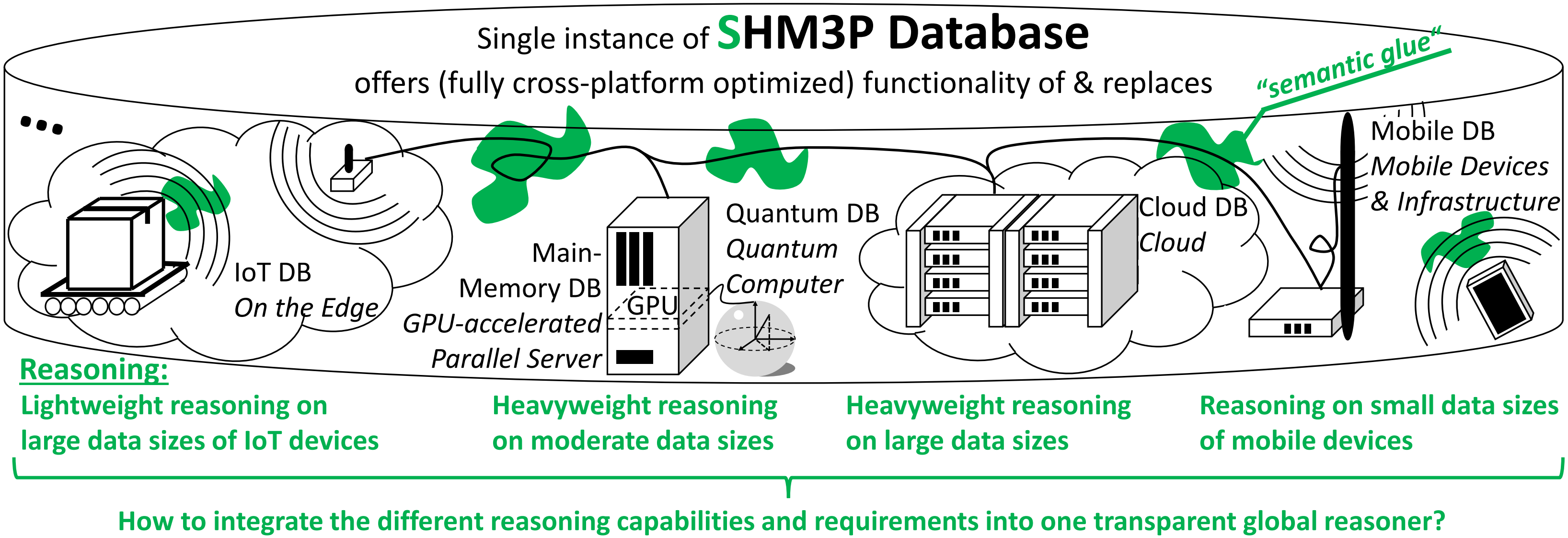
Challenges for HM3P Databases 1/2

- developing only one code base for the different platforms, but not introducing performance overhead in comparison to single platform databases
- identifying common properties of several platforms and reusing those approaches (like fault tolerance mechanisms) in different combinations, which are best suitable for these considered platforms
- data distribution among different platforms (applying different data distribution approaches as well)
- data distribution strategies considering overall the different properties of used platforms and models (like fast reads on parallel servers (using relational databases) and fast updates in cloud databases)

Challenges for HM3P Databases 2/2

- query optimization and other database tasks across different platforms, which apply different database approaches
- dealing with and integrating different privacy and security mechanisms supporting different privacy and security levels in the different platforms (with research e.g. on querying heterogeneous encrypted data)
- concurrency control approaches of different type have to be combined and work in cooperation (like 2 phase locking for server platforms and optimistic concurrency control for P2P networks)
- combining different types of databases (on different platforms) to offer the best of these databases and platforms *under one hood* to applications and users transparently or via intelligent integration into query language and API, e.g.,
    - guaranteeing atomicity and isolation in transactions for the data stored on a parallel server, but not for those data in the cloud supporting fast updates

Semantic Hybrid Multi-Model Multi-Platform (SHM3P) Database



- How to integrate the semantic layer between different types of databases and support semantic processing specialities like reasoning over the boundaries of different platforms?

Challenges for **S**HM3P Databases

- integrating different data models in a semantic layer on top of the underlying data models
- efficient transformations from and to the semantic model in an operational system
- developing efficient semantic querying and reasoning over the integrated data of different models
- global reasoning over reasoners running on different platforms supporting some kind of distributed heterogeneous reasoning
- developing a combination of stream reasoning over streaming data (e.g. of IoT devices) with static reasoning over large-scale data sets (stored e.g. in clouds)
- supporting transactions over semantic data by integrating the reasoner in transaction synchronization

Proposals for Cooperation & Collaborations

- Contributions to luposdate3000 are welcome: https://github.com/luposdate3000/luposdate3000
  - current status: SMP DBMS, soon SHMP DBMS
- Any other computer science topic in my expertise area
- Please contact me: groppe@ifis.uni-luebeck.de

Summary and Conclusions

- Different data models and their special features
  - ➡ Multi-Model Databases
- Different platforms and a need for different types of databases
  - Different features
  - ➡ Multi-Platform Databases
- Databases spanning over different platforms in operation (supporting multiple data models)
  - ➡ Hybrid Multi-Model Multi-Platform (HM3P) Databases

CfPs: My workshops at best-ranked DB conferences

- https://www.ifis.uni-luebeck.de/~groppe/bidede/
- Submission: March 18, 2021
- Workshop (online): June 20, 2021

- https://www.ifis.uni-luebeck.de/~groppe/vliot/
- Submission: April 5, 2021
- Workshop (hybrid): August 16, 2021