SCIENCE
PASSION
TECHNOLOGY

Architecture of DB Systems
02 DB System Architectures

Matthias Boehm

Graz University of Technology, Austria
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMK endowed chair for Data Management

Last update: Oct 14, 2020

Announcements/Org

- **#1 Video Recording**
  - Link in TeachCenter & TUbe (lectures will be public)
  - Optional attendance (independent of COVID)

TUbe

- **#2 COVID-19 Restrictions** (HS i5)
  - Max 25% room capacity (TC registrations)

max 18/74

- **#3 Programming Projects**
  - Initial test suite, benchmark, and make file
  - Reference implementation Dexter released (you need to implement ./server.h)

```
cd src; make harness_test
./speed_test 1468 0 0 0 0 4000 16000 100
```

seed IX pop TX/IX #IX

```
Creating 100 indices
Populating indices 100
Time to populate: 29ms
Testing the indices
Time to test: 1106ms
Testing complete.
     NUM_DEADLOCK: 0
     NUM_TXN_FAIL: 0
     NUM_TXN_COMP: 1600000
Overall time to run: 1135ms
```

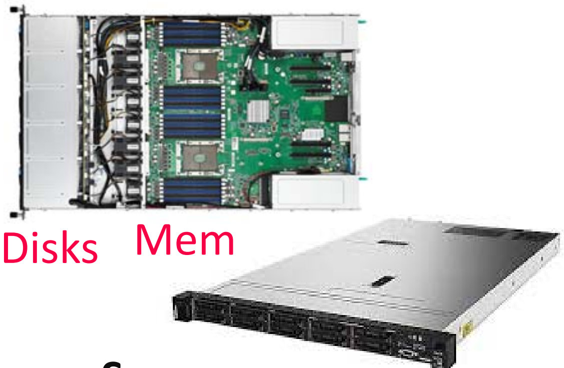
Agenda

- **Basic HW Background**
- **Classification of DB Architectures**

Basic Hardware Background

Anatomy of a Data Center

Commodity CPU:
Xeon E5-2440: 6/12 cores
Xeon Gold 6148: 20/40 cores



Server:
Multiple sockets,
RAM, disks

Rack:
16-64 servers +
top-of-rack switch

Cluster:
Multiple racks + cluster switch

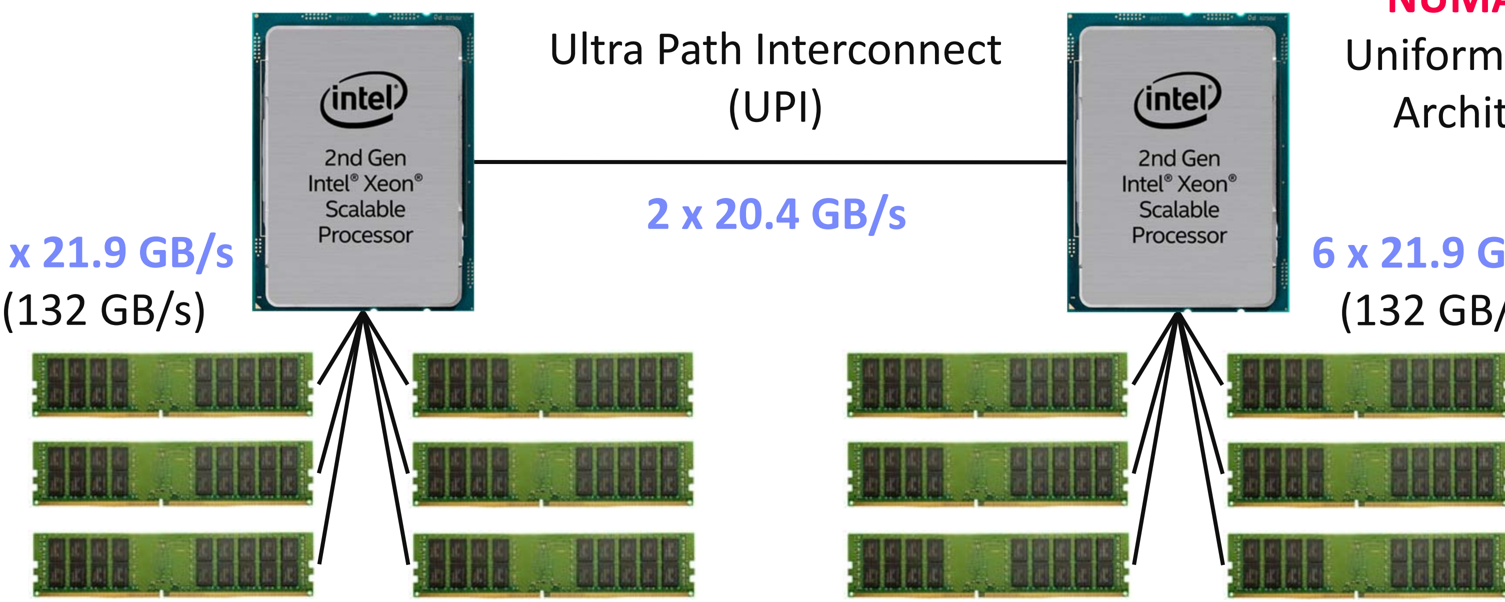
Data Center:
>100,000 servers

[Google Data Center, Eemshaven, Netherlands]

Basic CPU/Memory Architecture

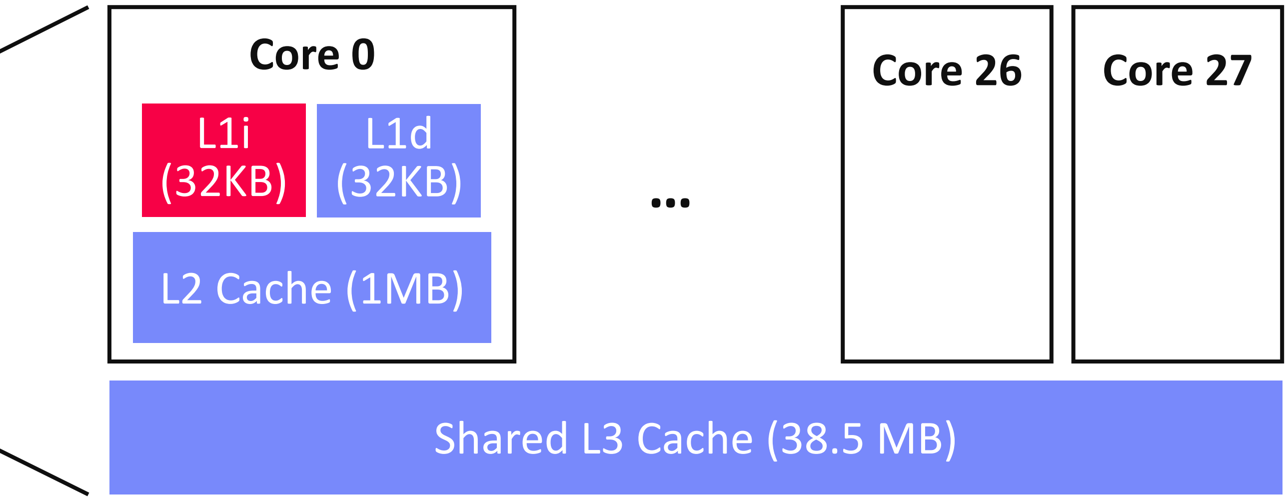
[https://en.wikichip.org/wiki/intel/xeon_gold/6238r]

- **Example DM Cluster** (scale-up)
  - Scale-up Intel Xeon Gold 6238R @ 2.2-4 Ghz (2 x 28 pcores, **2 x 56 vcores**)
  - **768 GB** HPE DDR4 RAM @ 2.933 GHz (12 x 64GB 2Rx4 PC4-2933Y-R)



Basic CPU/Memory Architecture, cont.

- **Example DM Cluster**
  - Scale-up Intel Xeon Gold 6238R @ 2.2-4 GHz (2 x 28 pcores, **2 x 56 vcores**)
  - **768 GB** HPE DDR4 RAM @ 2.933 GHz (12 x 64GB 2Rx4 PC4-2933Y-R)



Micro Architecture:
Cascade Lake '19

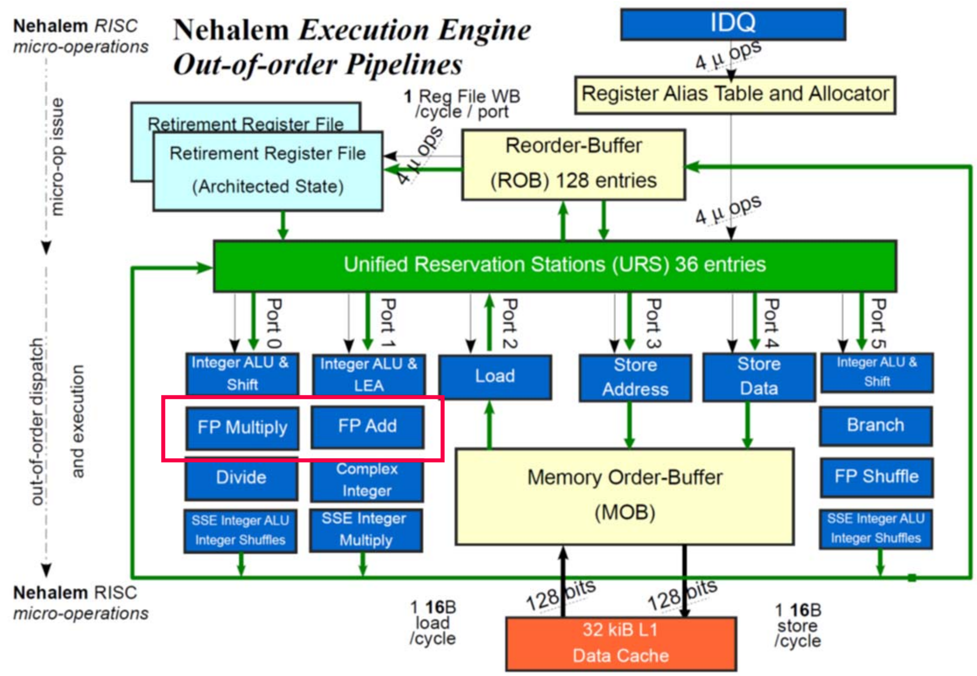
Why do we need a cache hierarchy?

- **Cache Coherence Protocols** (e.g., dictionary, snooping)

CPU (Core) Microarchitecture

- **Example Nehalem**
  - **Frontend:** Instruction Fetch, Pre-Decode, and Decode
  - **Backend:** Rename/Allocate, Scheduler, Execute
  - Out-of-Order Execution Engine (128b FP Mult/Add)

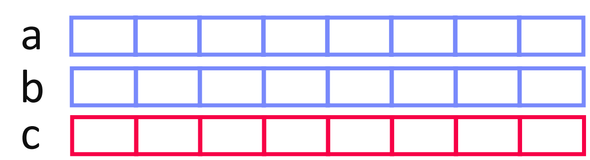
[M. E. Thomadakis: The Architecture of the Nehalem Processor and Nehalem EP SMP Platforms, Report, 2010]



- **SIMD Processing**
  - Single-instruction, multiple data
  - Process the same operation on multiple elements at a time
  - Data/instruction parallelism
  - Example: `VFMADD132PD`

2009 Nehalem: **128b** (2xFP64)
2012 Sandy Bridge: **256b** (4xFP64)
2017 Skylake: **512b** (8xFP64)

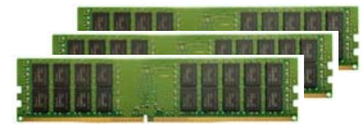
```
c = _mm512_fmadd_pd(a, b);
```



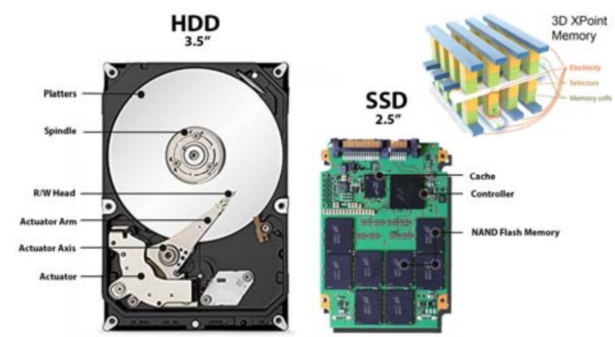
Basic Storage Architecture

Perf ←→ Cost per GB

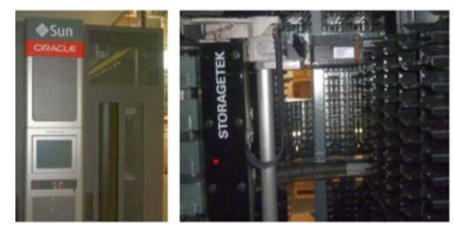
- **Primary Storage**
  - Main Memory (volatile, often charge-based)
  - Capacitors leak charge → periodic refresh (**~64ms**)



- **Secondary Storage** (non-volatile storage)
  - **HDD:** hard disk drive (magnetic, rotating platters)
  - **SSD:** solid-state drives (flash memory)
  - **NVM:** non-volatile memory (flash/resistive)



- **Tertiary Storage** (archival mass storage)
  - Optical disks (special materials), Magneto-optical disks
  - Tape drives: magnetic tape w/ high capacity cartridges



50PB tape library

Why do we need a storage hierarchy?

[Thomas Hahmann, Hans Weber, Erhard Diedrich, Gunter Schreier: SENTINEL-1 AND SENTINEL-3-OLCI PAC AT **DLR**, ESA-SP 722, **2013**]

Basic Network Architecture

- **Example DM Cluster**
  - 2 Racks Inffeldgasse 31
  - Switch: **HPE FlexFabric 5710 48XGT** (48x **10 GbE**, or 6x 40 GbE, 2 x 100 GbE)

[https://www.bechtle.com/at/shop/hpe-flexfabric-5710-48xgt-switch--4288448--p]

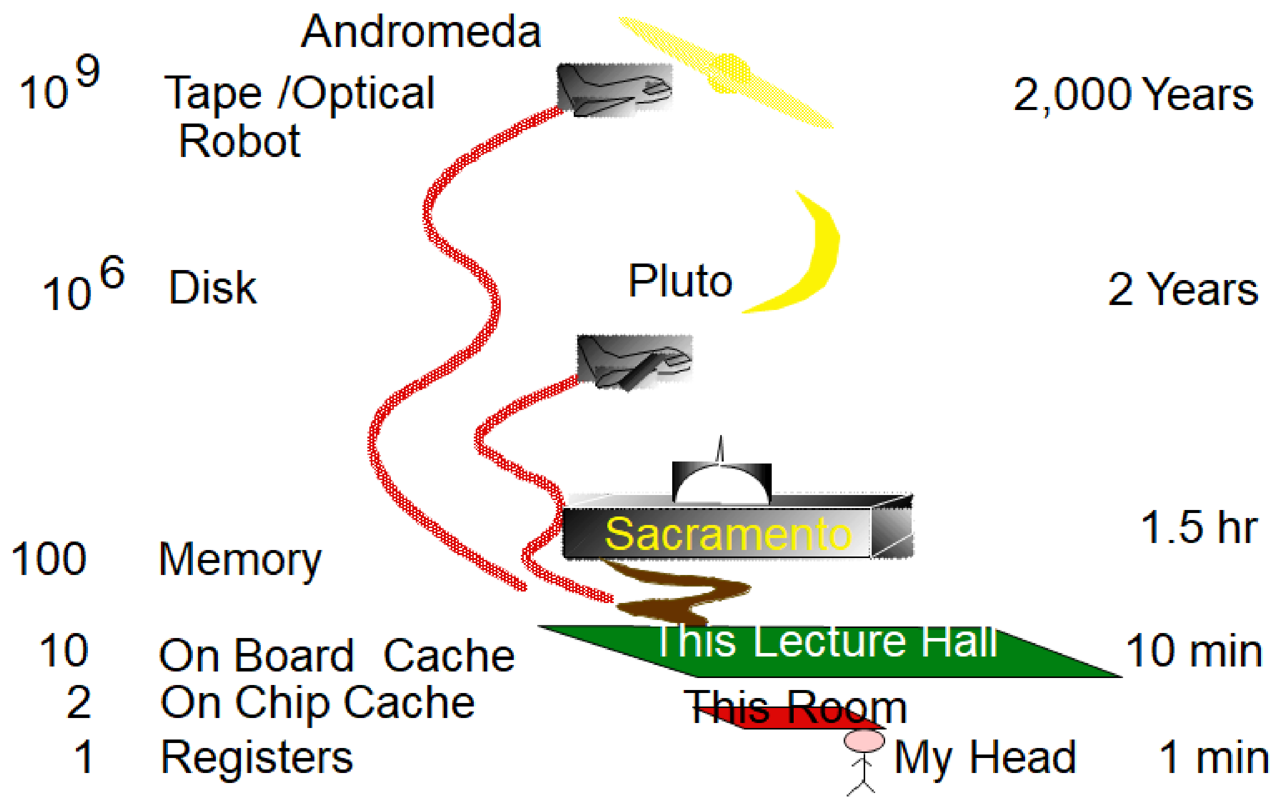
  - 1 Node (scale-up, 2 SSD system, 12 SSD data, T4 GPU)
  - 14 Nodes (scale-out)
    - AMD EPYC 7302 CPU at 3.0-3.3 GHz (16 pcores / **32 vcores**)
    - **128GB** HPE DDR4 RAM @ 2.933 GHz (8x 16GB 1Rx4 PC4-2933Y-R)
    - 2x 480GB SATA SDDs (system), 12x 2TB SATA HDDs (data)
    - **2x 10Gb** Ethernet (2 port adapter)

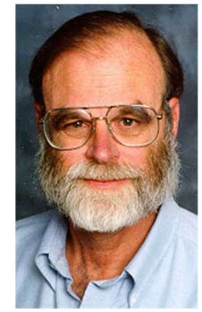
Latency Numbers Every Programmer Should Know

Operation	Time [ns]	Time [us]	Time [ms]
Inst execute / L1 cache reference	0.5		
Branch mispredict	5		
L2 cache reference	7		
Mutex lock/unlock	25		
Main memory reference	100		
Send 1K bytes over 1 Gb Ethn	10,000	10	
Read 4K randomly from SSD	150,000	150	
Read 1 MB sequentially from RAM	250,000	250	
Round trip within same datacenter	500,000	500	
Read 1 MB sequentially from SSD	1,000,000	1,000	1
Disk seek	10,000,000	10,000	10
Read 1 MB sequentially from disk	20,000,000	20,000	20
Send packet US←→Europe	150,000,000	150,000	150

[https://gist.github.com/jboner/2841832, **2012**]

Jim Gray's Storage Latency Analogy





Turing Award '98

[Joseph M. Hellerstein: CS 186: Introduction to Database Systems – Storing Data: Disks and Files, **Fall 2002,** https://dsf.berkeley.edu/jmh/cs186/f02/lecs/lec15_6up.pdf]

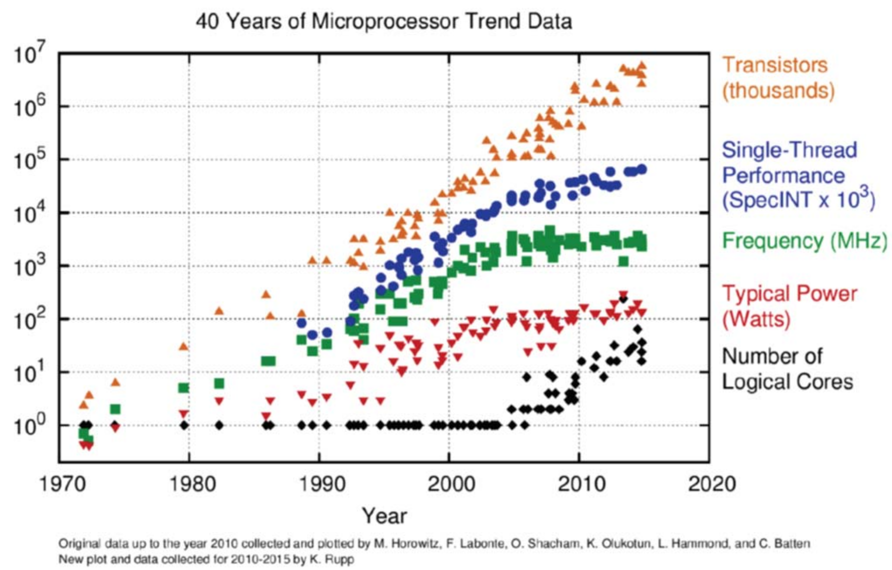
HW Challenges

[S. Markidis, E. Laure, N. Jansson, S. Rivas-Gomez and S. W. D. Chien: Moore's Law and Dennard Scaling]

- **#1 End of Dennard Scaling** (~2005)
  - Law: power stays proportional to the area of the transistor
  - Ignored leakage current / threshold voltage
    → **increasing power density S^2** (power wall, heat) → stagnating frequency

$P = \alpha CFV^2$ (**power density 1**)
(P .. Power, C .. Capacity, F .. Frequency, V .. Voltage)

- **#2 End of Moore's Law** (~2010-20)
  - Law: #transistors/performance/ CPU frequency doubles every 18/24 months
  - Original: # transistors per chip doubles every two years **at constant costs**
  - Now increasing costs



→ **Consequences: Dark Silicon and Specialization**

Classification of DB Architectures

Background and Design Dimensions

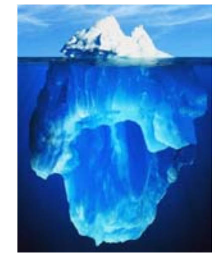
Recap Data Models, Consistency Models

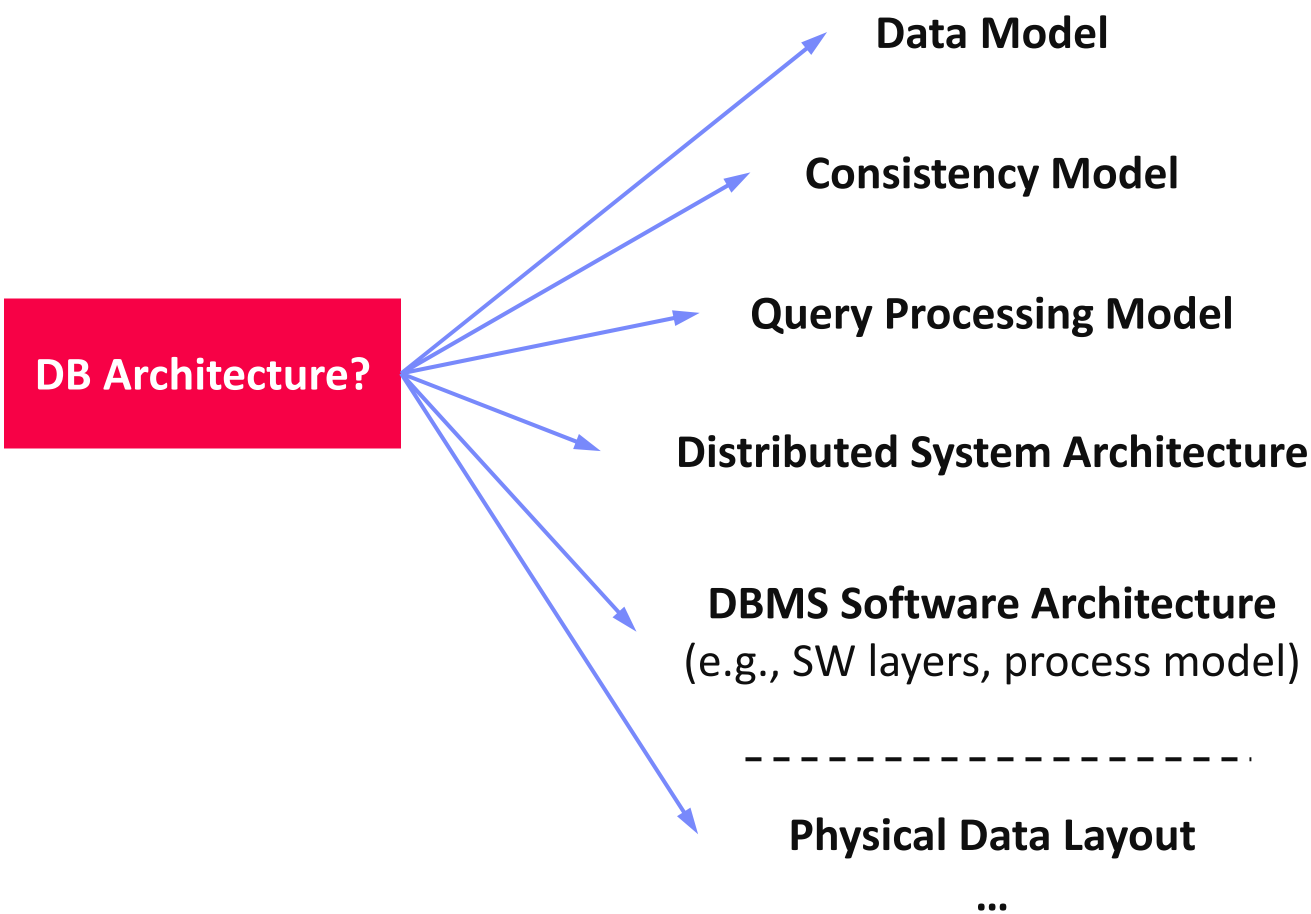
Recap Query Processing Models

Distributed Systems & DBMS Architecture

Row & Column Storage

Classification Dimensions





Recap: Data Models

- **Conceptual Data Models**
  - Entity-Relationship Model (ERM), focus on data, ~1975
  - Unified Modeling Language (UML), focus on data and behavior, ~1990

- **Logical Data Models**
  - Relational (Object/Relational)
  - Key-Value
  - Document (XML, JSON)
  - Graph
  - Time Series
  - Matrix/Tensor
  - Object-oriented
  - Network
  - Hierarchical

  } Mostly obsolete (Object-oriented, Network, Hierarchical)

- **Physical Data Models**
  - Row / column (page layouts)
  - LSM
  - Nested text/binary, flattened
  - Vertex-centric
  - TSM
  - Row-/column-major, tiled, etc

Recap: Relational Data Model

- **Domain D (value domain): e.g., Set S, INT, Char[20]**
- **Relation R**
  - **Relation schema** RS: Set of k attributes $\{A_1,\ldots,A_k\}$
  - **Attribute** A_j: value domain $D_j = dom(A_j)$
  - **Relation:** subset of the Cartesian product over all value domains D_j
    $R \subseteq D_1 \times D_2 \times \ldots \times D_k, k \geq 1$

Attribute

A1 INT	A2 INT	A3 BOOL
3	7	T
1	2	T
3	4	F
1	7	T

Tuple

cardinality: 4
rank: 3

- **Additional Terminology**
  - **Tuple**: row of k elements of a relation
  - **Cardinality** of a relation: number of tuples in the relation
  - **Rank** of a relation: number of attributes
  - **Semantics:** Set := no duplicate tuples (in practice: Bag := duplicates allowed)
  - **Order of tuples and attributes is irrelevant**

Recap: Key-Value Stores

- **Motivation**
  - Basic key-value mapping via simple API (more complex data models can be mapped to key-value representations)
  - Reliability at massive scale on commodity HW (cloud computing)

- **System Architecture**
  - Key-value maps, where values can be of a variety of data types
  - APIs for CRUD operations (create, read, update, delete)
  - Scalability via sharding (horizontal partitioning)

users:1:a	"Inffeldgasse 13, Graz"
users:1:b	"[12, 34, 45, 67, 89]"
users:2:a	"Mandellstraße 12, Graz"
users:2:b	"[12, 212, 3212, 43212]"

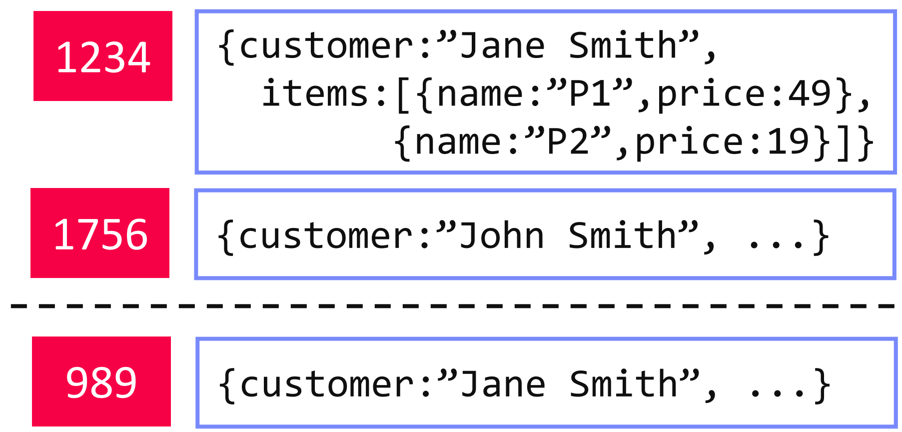
- **Example Systems**
  - Dynamo (2007, AP) → Amazon DynamoDB (2012)
  - Redis (2009, CP/AP)

[Giuseppe DeCandia et al: Dynamo: amazon's highly available key-value store. **SOSP 2007**]

Recap: Document Stores

- **Motivation**
  - Application-oriented management of **structured, semi-structured, and unstructured information** (pay-as-you-go, schema evolution)
  - Scalability via parallelization on commodity HW (cloud computing)

- **System Architecture**
  - Collections of (**key**, **document**)
  - Scalability via sharding (horizontal partitioning)
  - Custom SQL-like or functional query languages

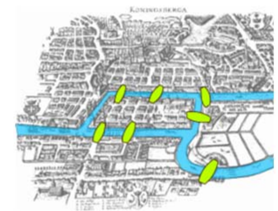


- **Example Systems**
  - **MongoDB** (C++, 2007, **CP**) → **RethinkDB**, **Espresso**, **Amazon DocumentDB** (Jan 2019)
  - **CouchDB** (Erlang, 2005, **AP**) → **CouchBase**

Recap: Graph Processing

[Grzegorz Malewicz et al: Pregel: a system for large-scale graph processing. **SIGMOD 2010,** (SIGMOD 2020 TTA)]

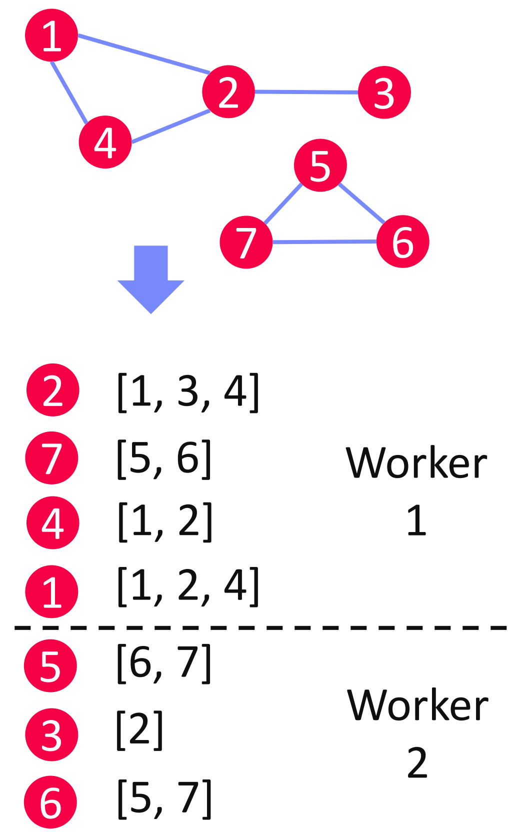
- **Google Pregel**
  - Name: Seven Bridges of Koenigsberg (Euler 1736)
  - **"Think-like-a-vertex"** computation model
  - Iterative processing in super steps, comm.: message passing



- **Programming Model**
  - Represent graph as collection of vertices w/ edge (adjacency) lists
  - Implement algorithms via Vertex API
  - Terminate if all vertices halted / no more msgs

```
public abstract class Vertex {
  public String getID();
  public long superstep();
  public VertexValue getValue();

  public compute(Iterator<Message> msgs);
  public sendMsgTo(String v, Message msg);
  public void voteToHalt();
}
```



Recap: ACID Properties

- **Atomicity**
  - A transaction is executed atomically (**completely or not at all**)
  - If the transaction fails/aborts no changes are made to the database (**UNDO**)
- **Consistency**
  - A successful transaction ensures that all **consistency constraints are met** (referential integrity, semantic/domain constraints)
- **Isolation**
  - Concurrent transactions are executed in isolation of each other
  - **Appearance of serial transaction execution**
- **Durability**
  - **Guaranteed persistence** of all changes made by a successful transaction
  - In case of system failures, the database is recoverable (**REDO**)

Recap: CAP Theorem

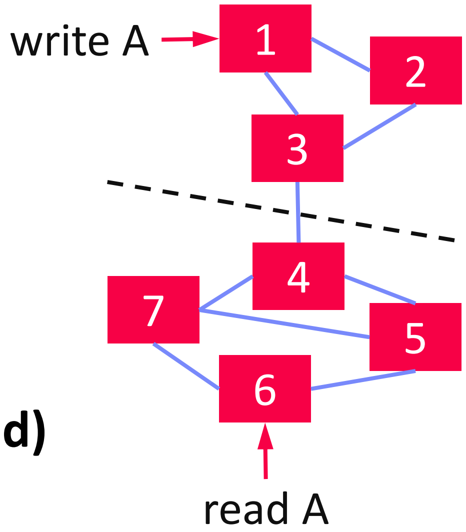
- **Consistency**
  - Visibility of updates to distributed data (atomic or linearizable consistency)
  - Different from ACIDs consistency in terms of integrity constraints
- **Availability**
  - Responsiveness of a services (clients reach available service, **read/write**)
- **Partition Tolerance**
  - Tolerance of temporarily unreachable network partitions
  - System characteristics (e.g., latency) maintained
- **CAP Theorem**
- **Proof**

"You can have AT MOST TWO of these properties for a networked shared-data systems."

[Eric A. Brewer: Towards robust distributed systems (abstract). **PODC 2000**]

[Seth Gilbert, Nancy A. Lynch: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. **SIGACT News 2002**]

Recap: CAP Theorem, cont.



- **CA: Consistency & Availability (ACID single node)**
  - Network partitions cannot be tolerated
  - Visibility of updates (**consistency**) in conflict with **availability** ➔ **no distributed systems**

- **CP: Consistency & Partition Tolerance (ACID distributed)**
  - Availability cannot be guaranteed
  - **On connection failure, unavailable** (wait for overall system to become consistent)

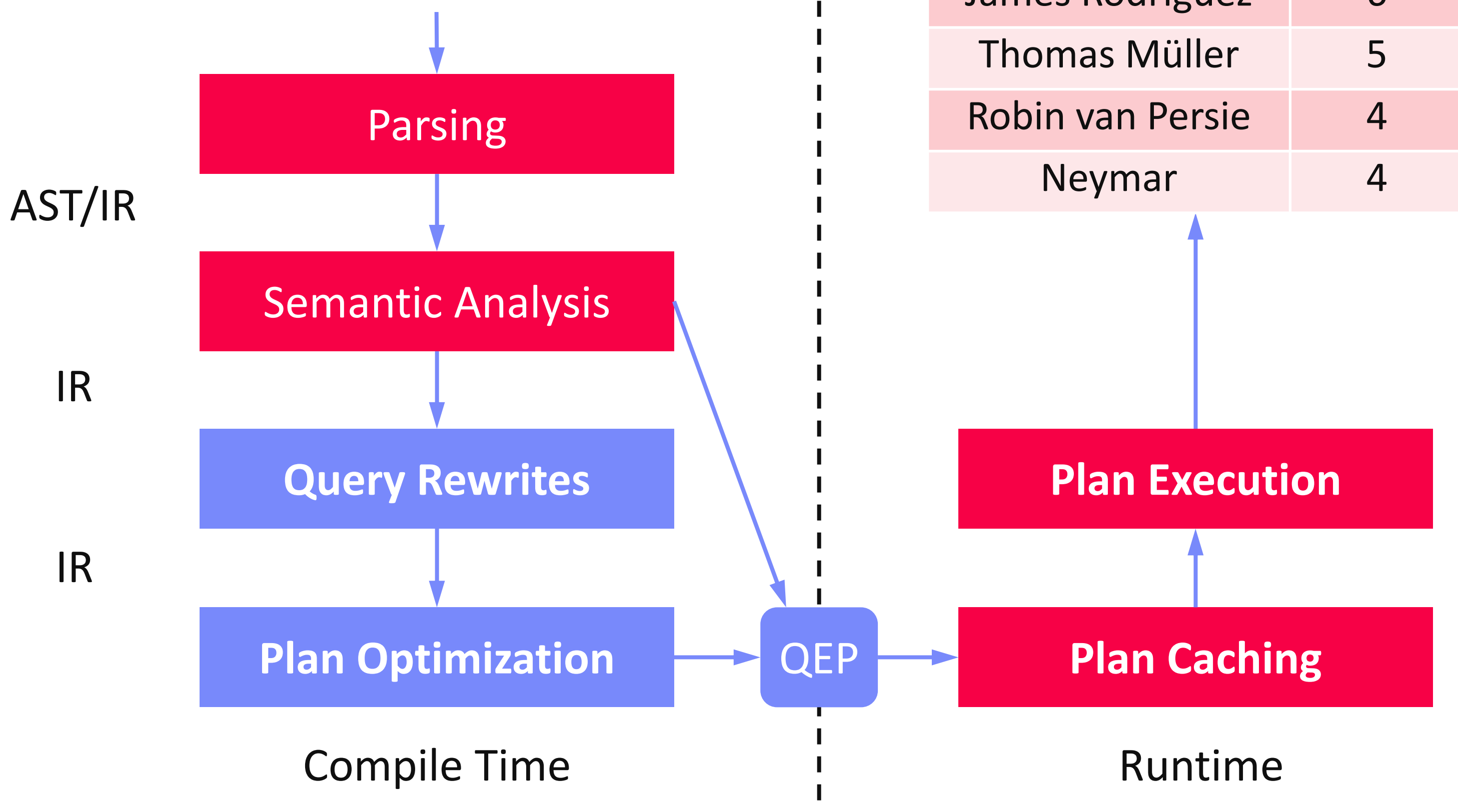
- **AP: Availability & Partition Tolerance (BASE)**
  - Consistency cannot be guaranteed, use of optimistic strategies
  - Simple to implement, main concern: availability to ensure revenue ($$$)

  ➔ **BASE consistency model** (basically available, soft state, eventual consistency)

Recap: Traditional Query Processing (OLTP/OLAP)

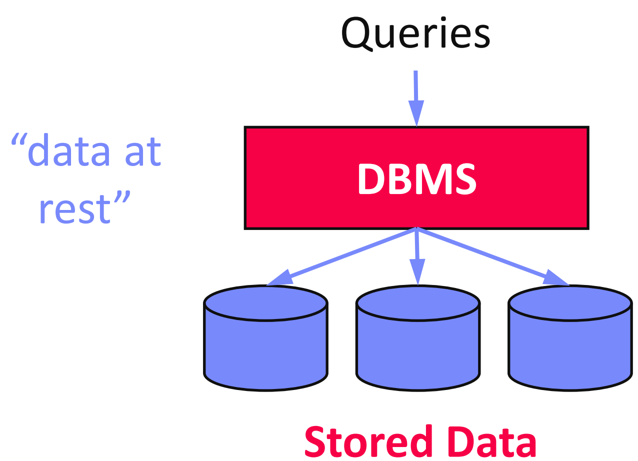
```
SELECT * FROM TopScorer
  WHERE Count>=4
```

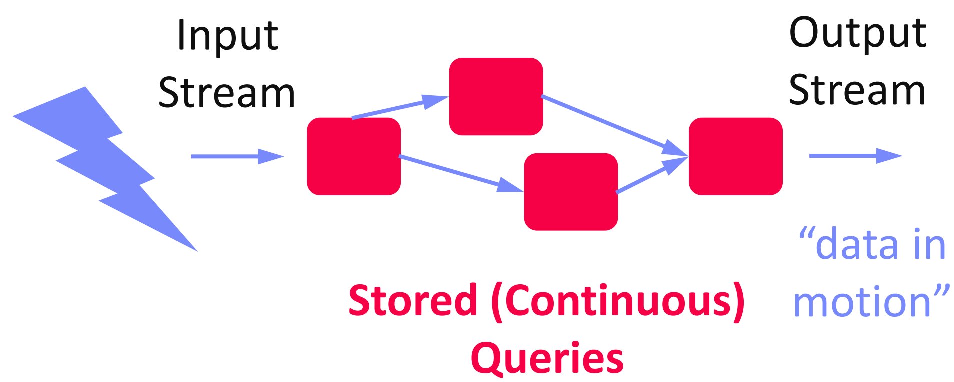
Name	Count
James Rodríguez	6
Thomas Müller	5
Robin van Persie	4
Neymar	4



Continuous Query Processing / Streaming

- **Stream Processing Architecture**
  - **Infinite input streams**, often with window semantics
  - Continuous (aka standing) queries



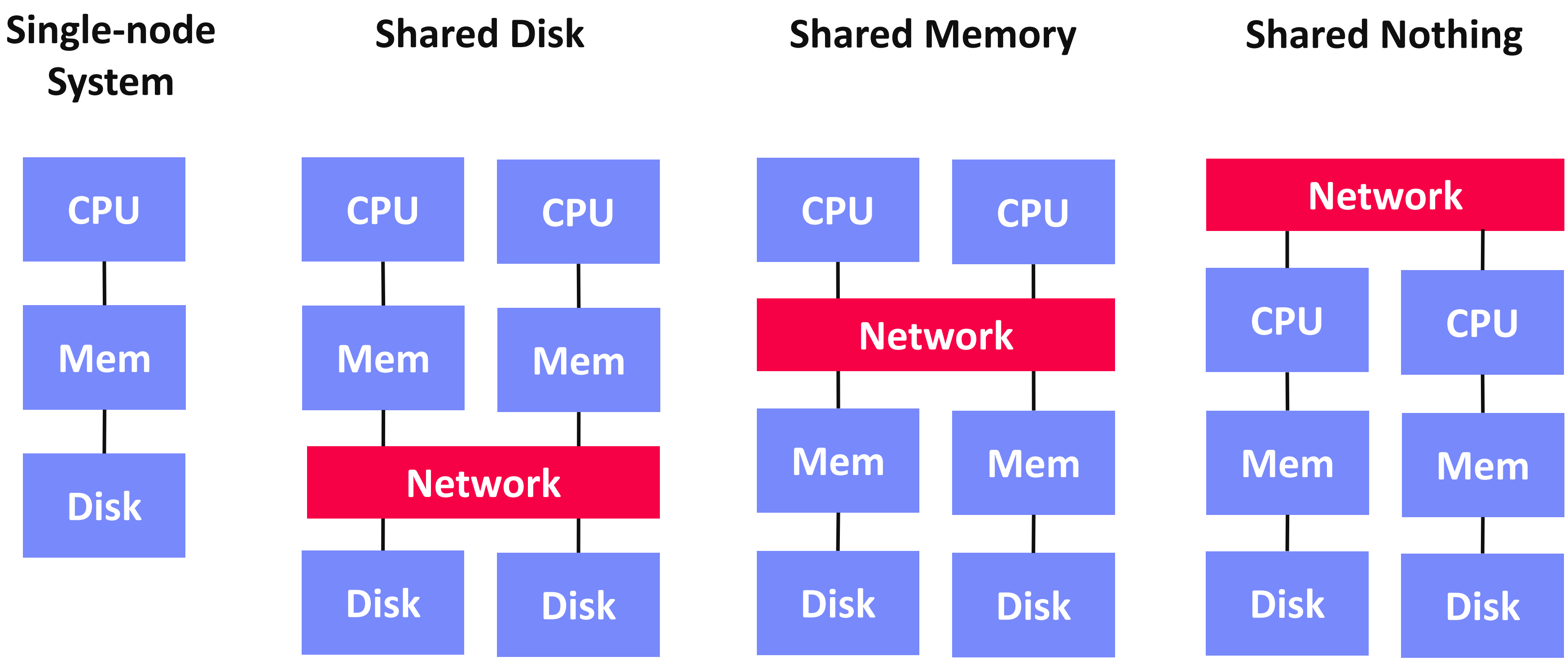


- **Optimizing Continuous Queries**
  - Multi-query optimization (multiple deployed queries)
  - Adaptive query optimization (based on changing workload)

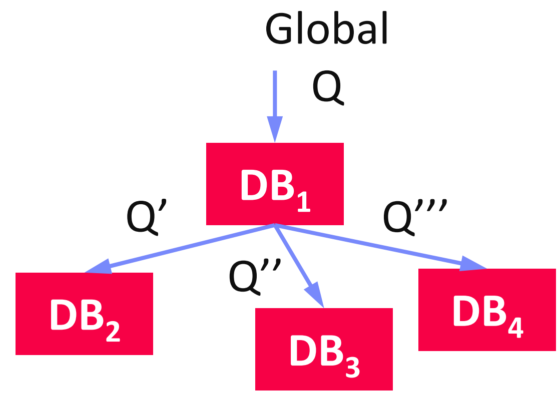
Network System Architectures

[David J. DeWitt, Jim Gray: Parallel Database Systems: The Future of High Performance Database Systems. Commun. ACM 35(6), **1992**]

- **Parallel DBS**
  - Goal: parallel query processing



Distributed Database Systems



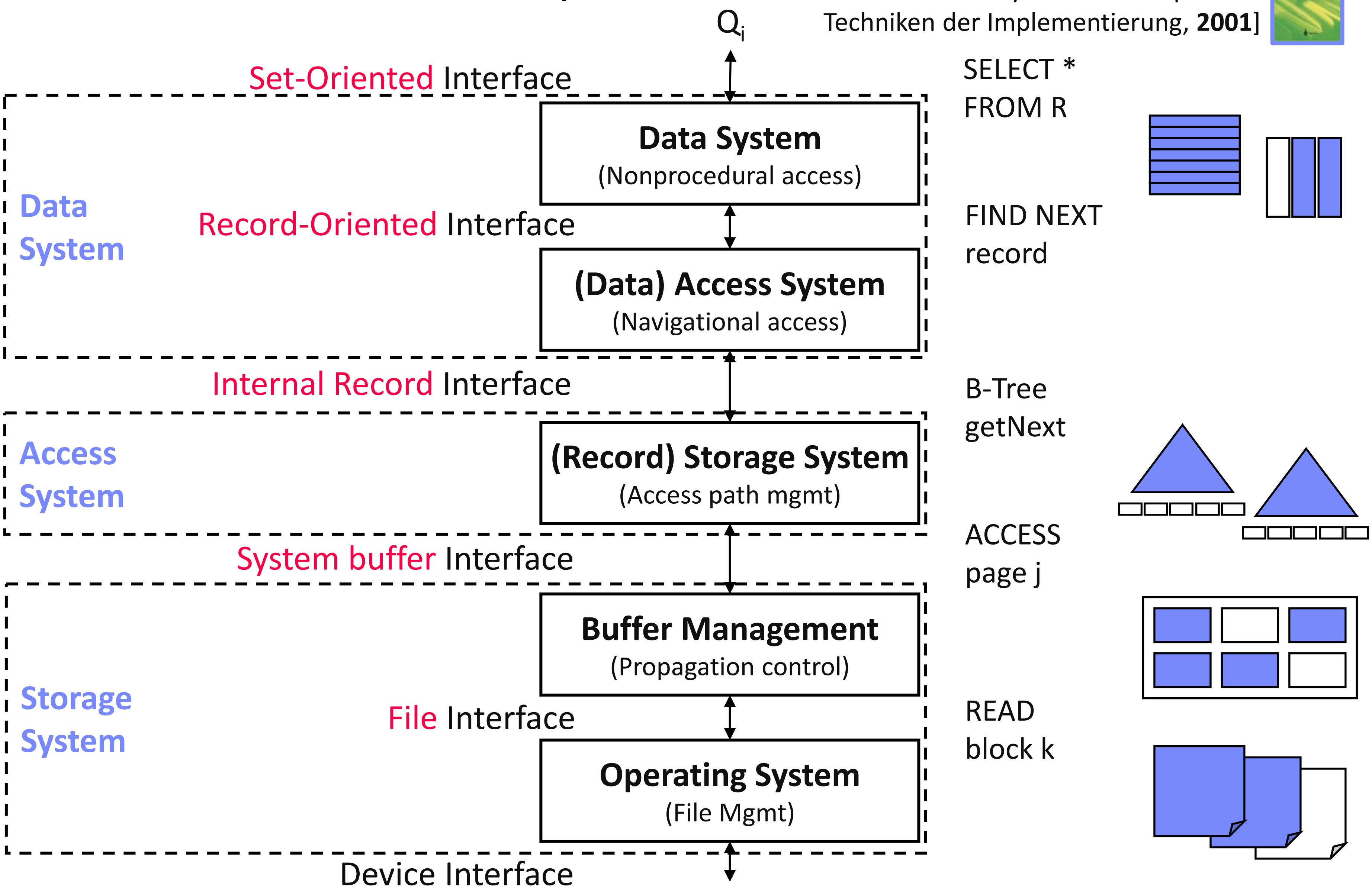
- **Distributed DBS**
  - Distributed database: Virtual (logical) database that appears like a local database but consists of multiple physical databases
  - Multiple local DBMS, components for global query processing
  - **Terminology:** **virtual DBS** (homogeneous), **federated DBS** (heterogeneous)

- **Challenges**
  - **Tradeoffs:** Transparency – autonomy, **consistency – efficiency/fault tolerance**
  - **#1** Global view and query language → schema architecture
  - **#2** Distribution transparency → global catalog
  - **#3** Distribution of data → data partitioning
  - **#4** Global queries → distributed join operators, etc
  - **#5** Concurrent transactions → **2PC**
  - **#6** Consistency of copies → **replication**

Beware: Meaning of "Transparency" (invisibility) here

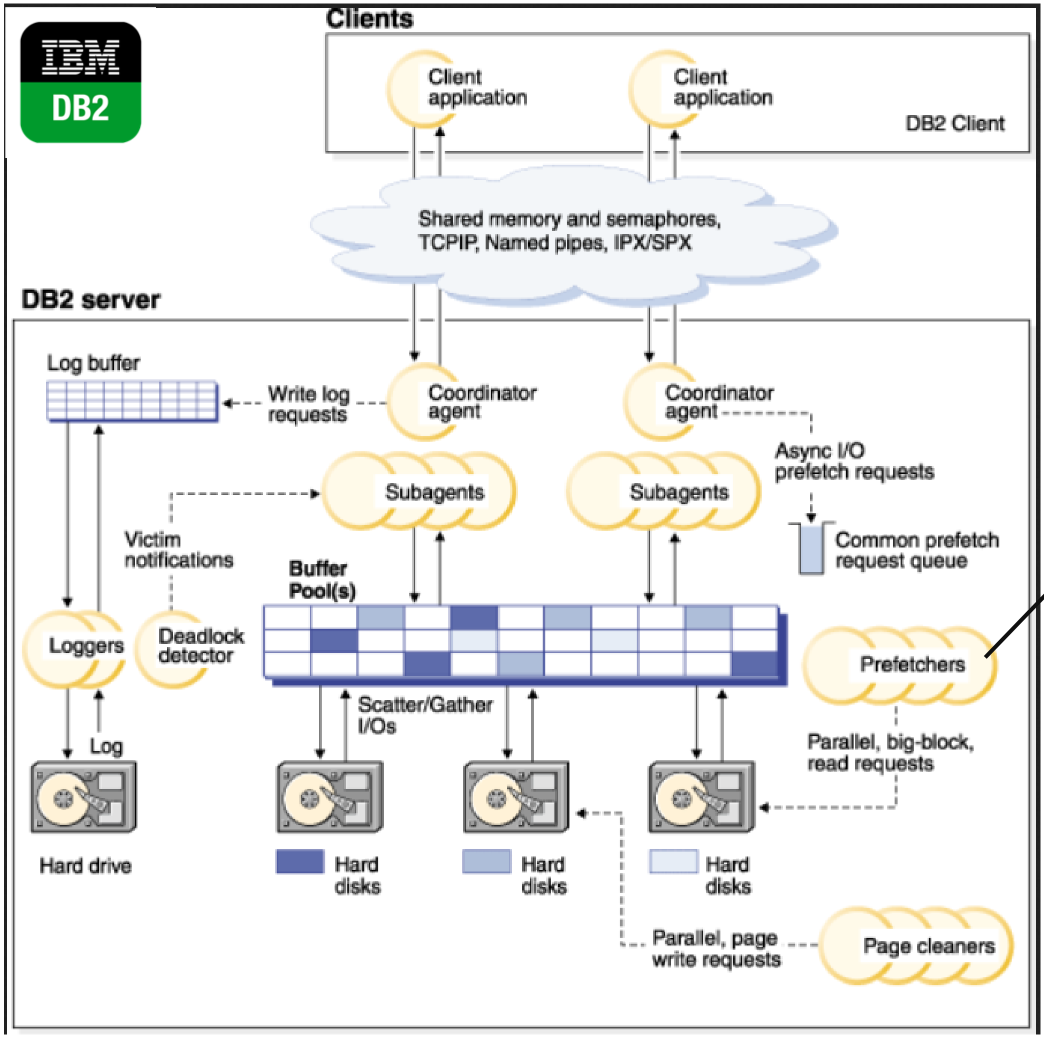
DBMS Architecture, cont.

[Theo Härder, Erhard Rahm: Datenbanksysteme: Konzepte und Techniken der Implementierung, **2001**]



IBM DB2 11.5 Architecture

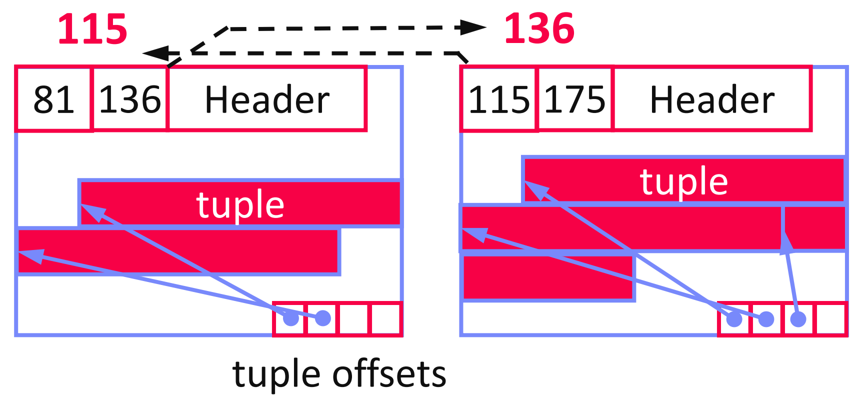
[https://www.ibm.com/support/knowledgecenter/SSEPGG_11.5.0/com.ibm.db2.luw.admin.perf.doc/doc/c0005418.html]



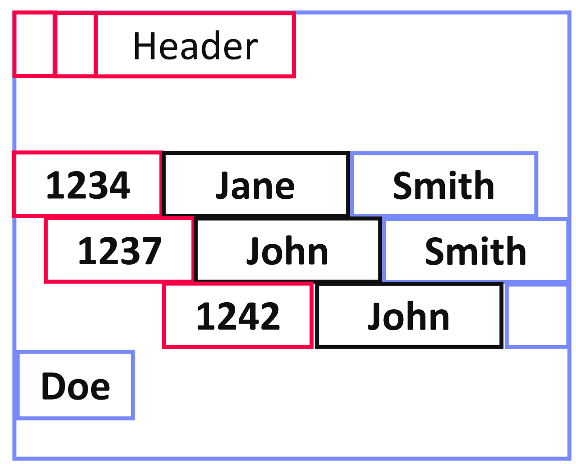
Engine Dispatchable Units (EDUs, e.g., db2 agents), implemented as OS threads

Row and Column Stores

- **Background: Storage System**
  - Buffer and storage management (incl. I/O) at granularity of **pages**
  - PostgreSQL default: **8KB**
  - Different table/page layouts

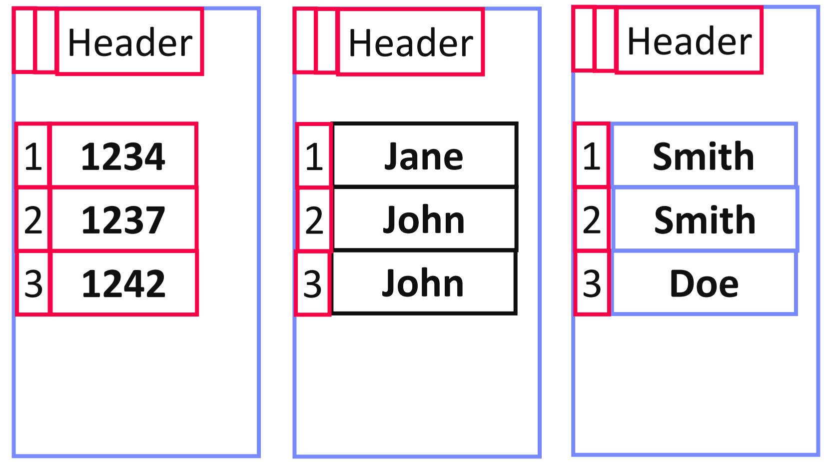


- **Row Storage**
  - **NSM** (nary storage model)
  - Store tuple attributes in contiguous form
  - Fast get/insert/delete
  - Slow column aggregates



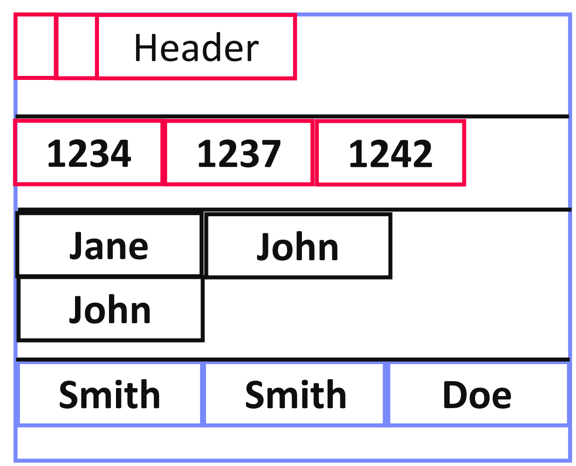
Row and Column Stores, cont.

- **Column Storage**
  - **DSM** (decomposed storage model) [**SIGMOD'85**, **ICDE'87**]
  - Store attribute values contiguously
  - Good compression, fast aggregates
  - Fast get/insert/delete (reconstruction needed)



- **Hybrid**
  - **PAX** (partition attributes across)
  - Combine advantages of NSM+DSM
  - **Cache-friendly page processing**
  - Variants in many modern systems

[Anastassia Ailamaki, David J. DeWitt, Mark D. Hill, Marios Skounakis: Weaving Relations for Cache Performance. **VLDB 2001**]



Summary and Q&A

- **Basic HW Background**
- **Classification of DB Architectures**
  - Data Model, Consistency Model, Query Processing Model,
  - Distributed System Architecture, DBMS Software Architecture,
  - Physical Data Layout

- **Programming Projects**
  - Initial test suite, benchmark, make file, and reference implementation
  - Try compiling it, and start **your own implementation** in next weeks

- **Next Lectures**
  - 03 **Data Layouts and Bufferpool Management** [Oct 21]
  - 04 **Index Structures and Partitioning** [Oct 28]
  - 05 **Compression Techniques** [Nov 04]