Department of Computer Science
University of Cyprus

EPL646 – Advanced Topics in Databases

Lecture 4

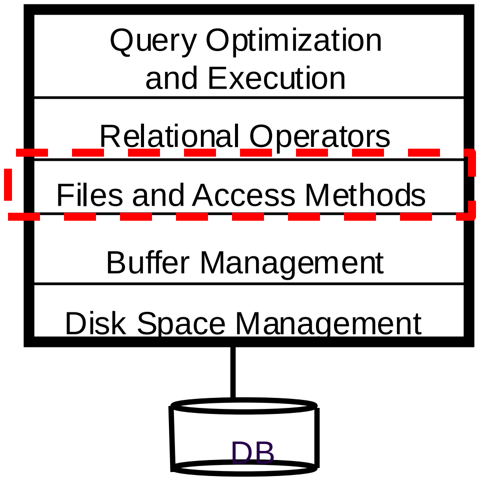
Indexing II: Tree-Structured Indexing and ISAM Indexes

Chap. 10.1-10.8: Ramakrishnan & Gehrke

Demetris Zeinalipour
http://www.cs.ucy.ac.cy/~dzeina/courses/epl446

Lecture Outline
Tree-Structured Indexing

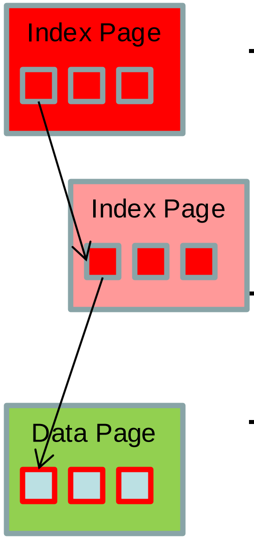
- **Note:** In prior lectures we gave an overview of **Storage and Indexing**. In this and the following lecture we will explore **Indexing** in more detail.
- 10.1) Introduction to Tree Indexes
- 10.2) The ISAM Index
  - Structure of Nodes in Trees,
  - Binary Search over Sorted Files,
  - Binary vs. N-ary Search Trees,
  - ISAM: Indexed Sequential Access Method (Outline, Search, Insert, Delete, Examples)



Indexes (Access Methods)
(Ευρετήρια Δευτερεύουσας Μνήμης)

- An *index* is a data structure that has **index records** that **point to** certain **data records**.
- An index can **optimize** certain kinds of retrieval operations (depending on the index).

- **Definitions**



  - **Index Page (Σελίδες Ευρετηρίου) vs. Data Pages (Σελίδες Δεδομένων):** Index Pages store index records to data records. Both reside on disk because we might have many of these pages!
  - **Data Record (Εγγραφή Δεδομένων):** Stores the actual data e.g., (59,Mike,3.14) .
  - **Index Record (Εγγραφή Ευρετηρίου):** Stores the RID of another index record (then called **index entry**) or a data record (then called **data entry**)

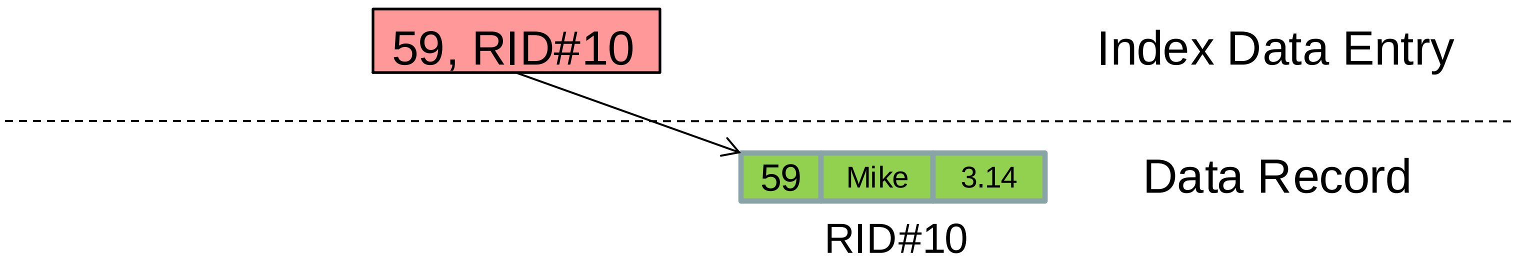
Data Entry **k*** Examples (Παραδείγματα Καταχώρησης **k***)

- **Alternative 1: <k>**

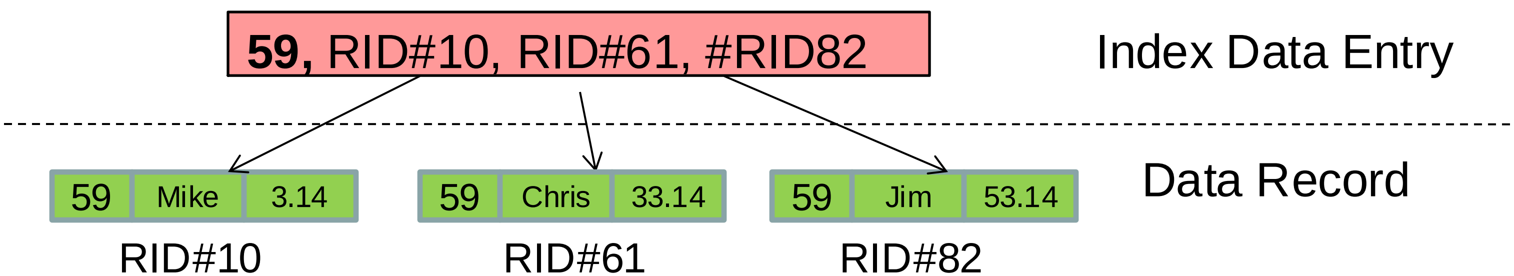
Results in a Index File Organization!

59, Mike, 3.14 — Index Data Entry

- **Alternative 2: <k, RID>**



- **Alternative 3: <k, [RID,…,RID]>**

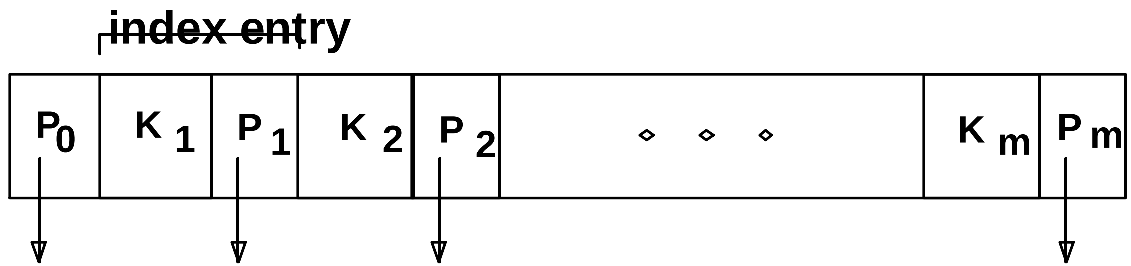


Introduction to Tree Structures (Εισαγωγή σε Δενδρικές Δομές)

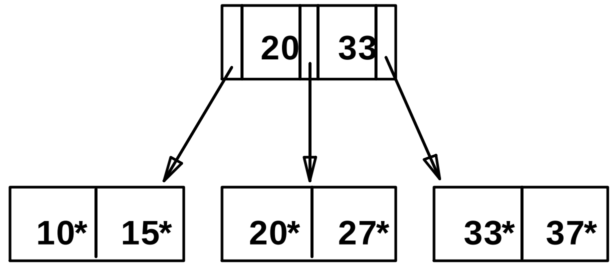
- **We will study two Tree-based structures:**
  - ***ISAM*:** A **static** structure (does not **grow** or **shrink)**.
    - Suitable for situations where the target relation does **not change frequently**;
    - Copes better with **Locking Protocols (explained later),** because the **index/data entries** are statically allocated, thus are not required to be locked during **concurrent access.**
  - ***B+ tree*:** A **dynamic** data structure that adjusts efficiently under **inserts** and **deletes**.
    - Most widely used tree structure in DBMS systems because it copes **efficiently with updates**! and because the cost for range and equality searches is good.
    - **Will be covered subsequently in this lecture!**

Structure of Nodes in Trees
(Δομή Κόμβου σε Δένδρα)

- Same Structure for **ISAM** and **B+Trees** (we shall utilize Alt.1 with keyonly unless otherwise noted)
- **M Keys** and **M+1 Pointers** to **children** (either index entries or data entries)

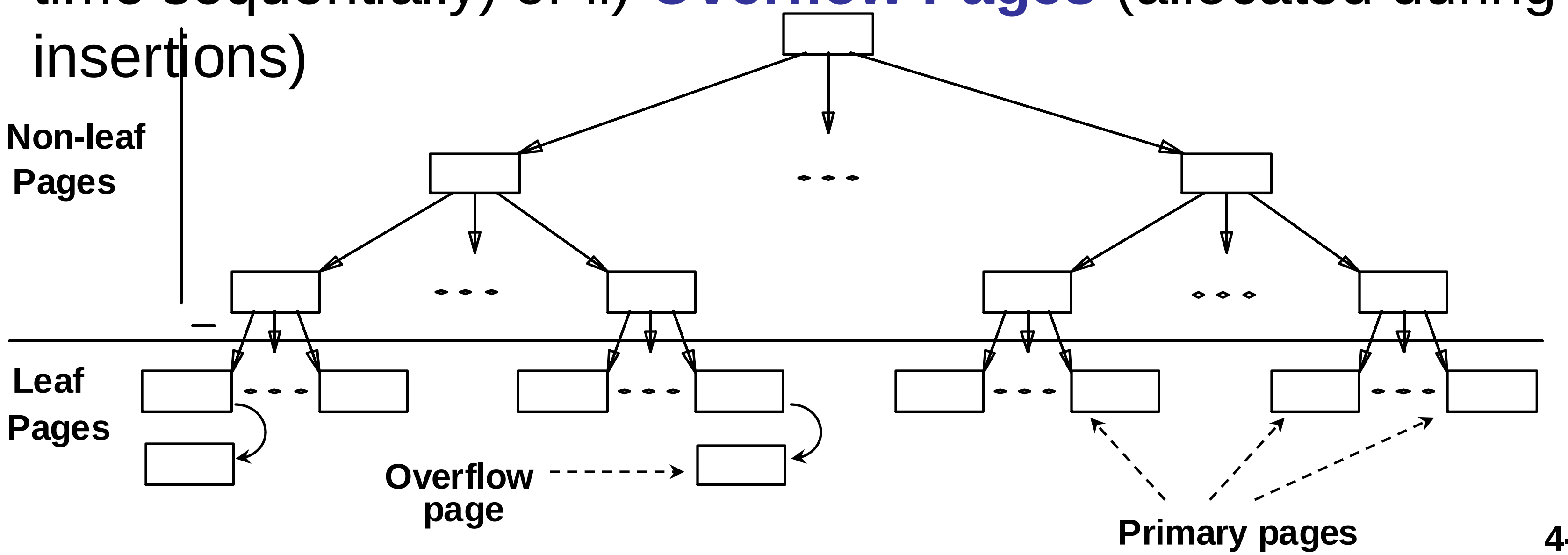


- Example with M=2



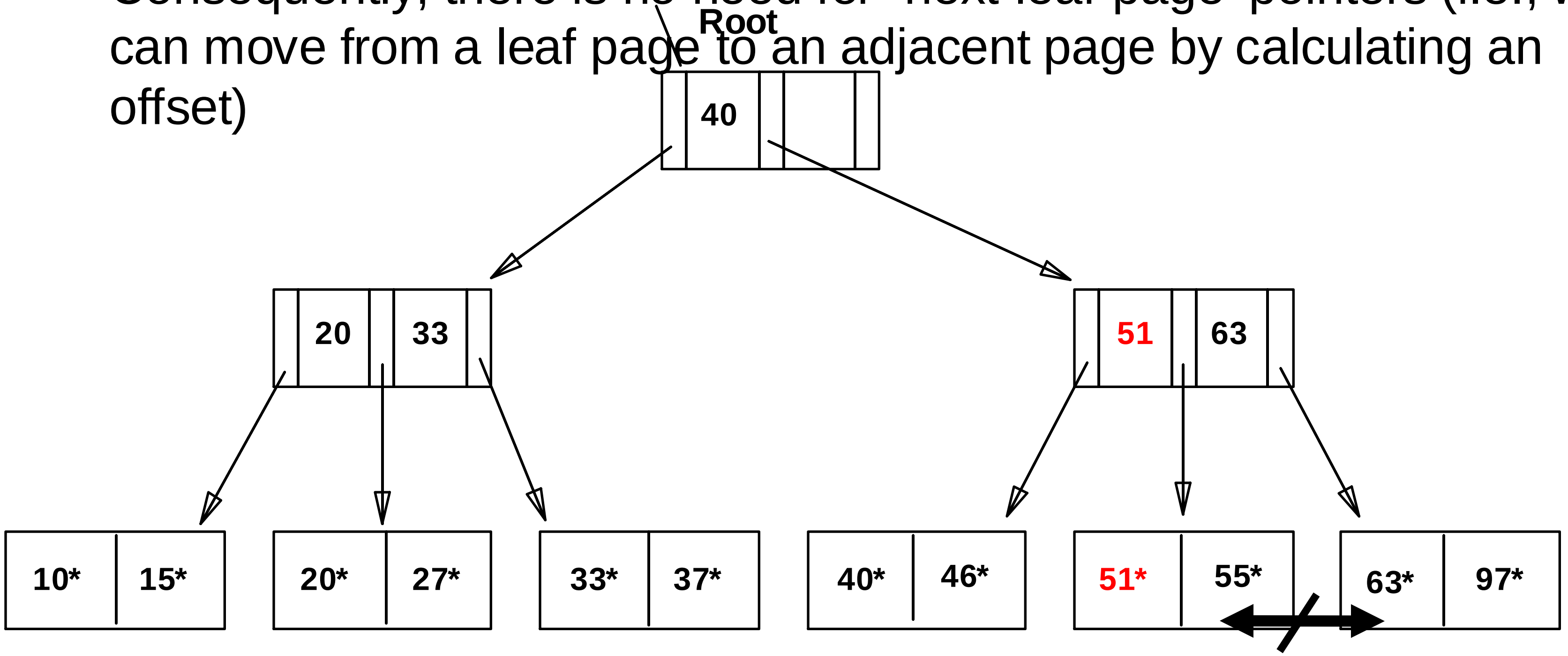
ISAM: Indexed Sequential Access Method

- A simple tree structure utilized by DBMS systems
- Constructed **Statically** at index creation time.
- Consists of **Non-leaf** (**index entries**, allocated at creation time) and **Leaf pages** (**data entries**) – **Alternative 1.**
- **Data Entries** : i) **Primary Pages** (allocated at creation time sequentially) or ii) **Overflow Pages** (allocated during insertions)



Outline of Operation (Ανασκόπηση Λειτουργίας)

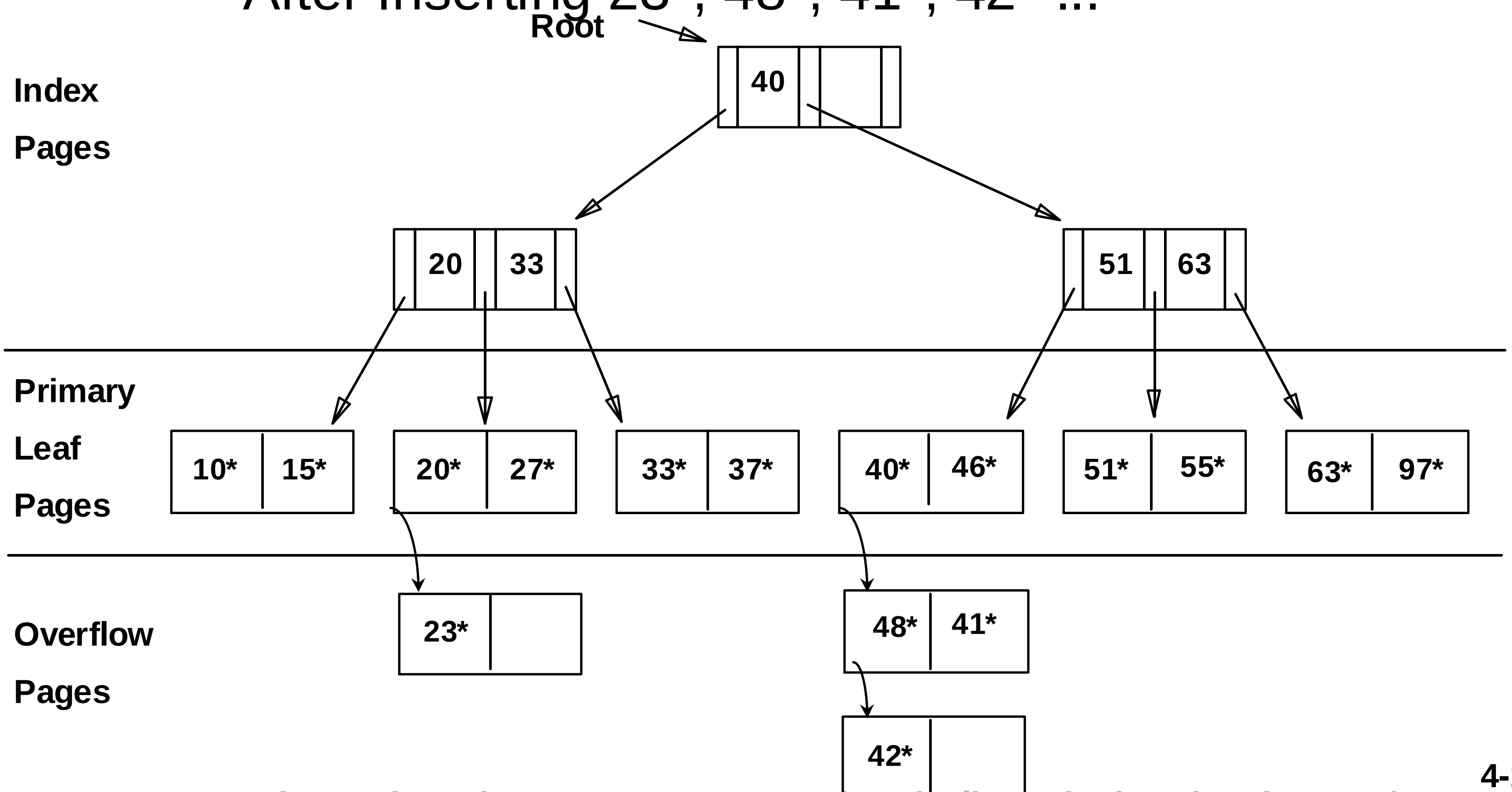
- **Search:** Start at root; use key comparisons to go to leaf. Cost: $\lfloor \log_F N \rfloor$; F=#entries_per_indexPage+1, N=#leafpgs
- Recall that data Entries are allocated sequentially when the tree is created.
  - Consequently, there is no need for `next-leaf-page' pointers (i.e., we can move from a leaf page to an adjacent page by calculating an offset)



Inserting to an ISAM Index
(Εισαγωγές στο Ευρετήριο ISAM)

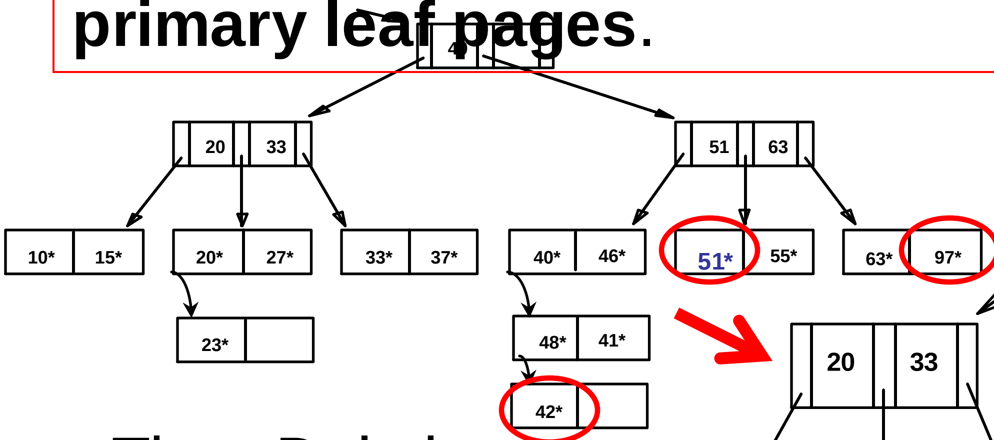
Insert: Find the appropriate **leaf data entry** and assign it to there. If full, allocate an overflow page and put it there

After Inserting 23*, 48*, 41*, 42* ...

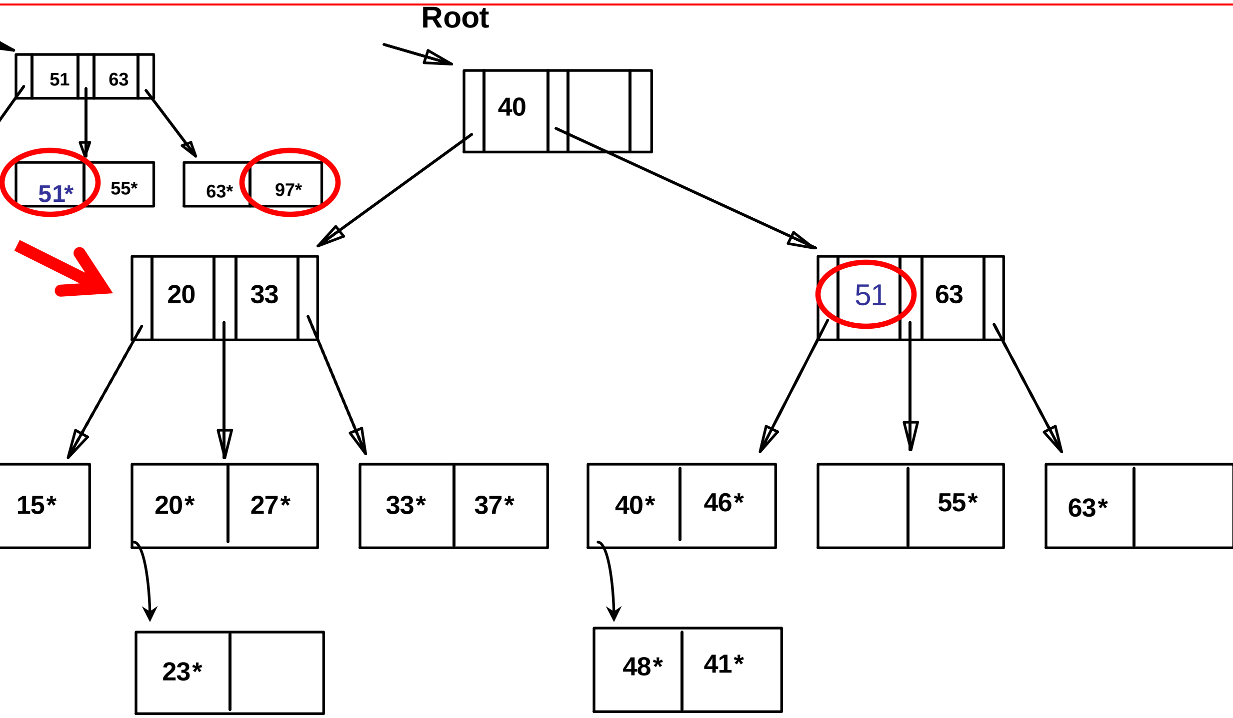


Deletions from an ISAM Index
(Διαγραφές από το Ευρετήριο ISAM)

Delete: Find and remove from leaf; if **overflow page gets empty** then de-allocate then given page. Never deallocate **primary leaf pages**.



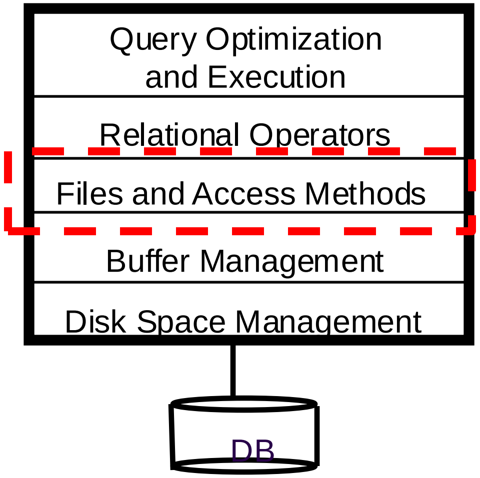
... Then Deleting 42*, 51*, 97*



- Note that 51* appears in index levels, but not in leaf! Static tree structure: inserts/deletes affect only leaf pages! …Will be useful for concurrency control (locking protocol)

Lecture Outline
B+ Trees: Structure and Functions

- 10.3) Introduction to B+ Trees
- 10.4-10.6) B+Tree Functions: Search / Insert / Delete with Examples
- 10.7) B+ Trees in Practice.
  - Prefix-Key Compression (Προθεματική Συμπίεση Κλειδιών)
  - Bulk Loading B+Trees (Μαζική Εισαγωγή Δεδομένων)

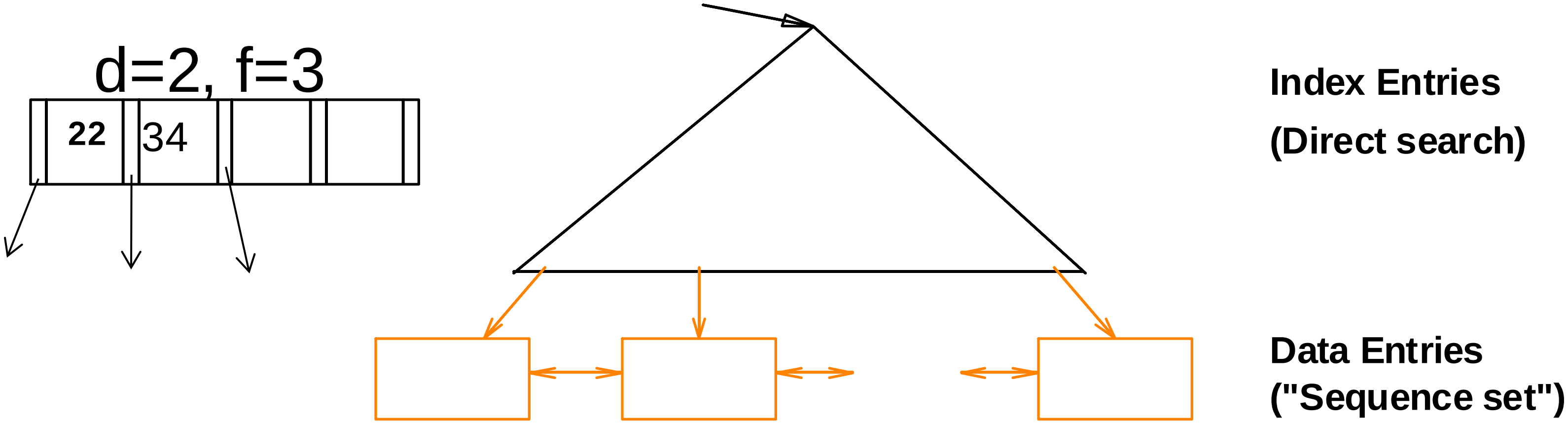


Introduction to Tree Structures
(Εισαγωγή σε Δενδρικές Δομές)

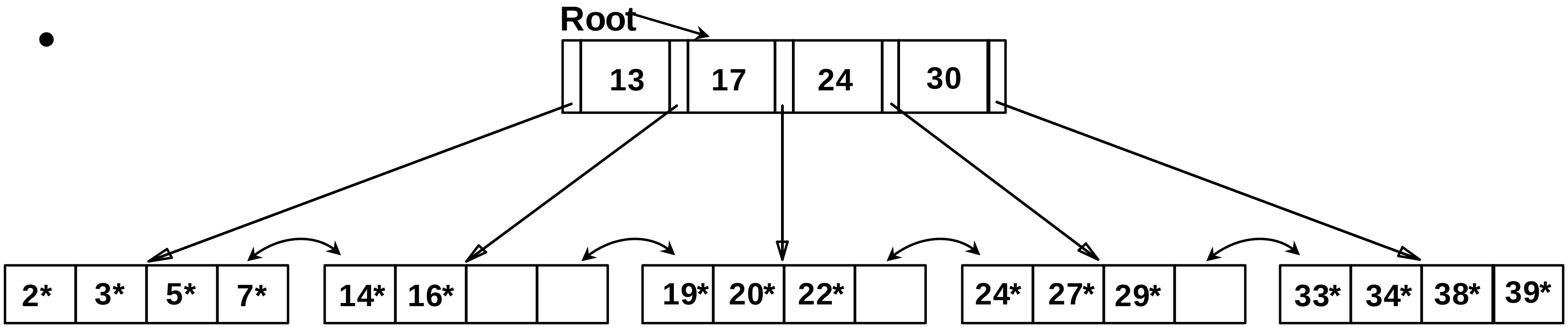
- **We will study two Tree-based structures:**
  - ***ISAM*:** A **static** structure (does not **grow** or **shrink)**.
    - **Suitable when changes are infrequently;**
    - Copes better with **Locking Protocols**
  - ***B+ tree*:** A **dynamic** data structure which adjusts efficiently under **inserts** and **deletes.**
    - Most widely used tree structure in DBMS systems!
    - Has similarly to ISAM, nodes with a high **fan-out (f)** (~133 children per node).
    - Similar to a **Btree** but different…
      - In a B+Tree, **data entries** are stored at the leaf level.
      - **A Btree allows search-key values to appear only once**; eliminates redundant storage of search keys (not suitable for DB apps where more index entries yield better search performance)

B+ Tree: Introductory Notes
(B+Tree: Εισαγωγικές Επισημάνσεις)

- Insert/delete at **log $_F$ N** cost; keep tree ***balanced (ισοζυγισμένο).*** (**F** = fanout, **N** = # leaf pages)
- **Minimum 50% occupancy** (except for root). Each node contains **d** <= *m* <= 2**d** entries. The parameter **d** is called the ***order* of the tree (βαθμός του δένδρου)**.
- Supports **equality** and **range-searches** (αναζητήσεις ισότητας και διαστήματος) efficiently.



Example B+ Tree
(Παράδειγμα B+Tree)

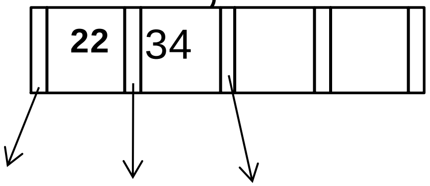
- Search begins at root, and key comparisons direct it to a leaf (as in ISAM).
- Search for 5*, 15*, all data entries >= 24* ...
- 

- Based on the search for 15*, we know its not in the tree!
- Note that **leaf pages** (τερματικοί κόμβοι) **are** linked together in a **doubly-linked list** (as opposed to **ISAM).**
- That happens because ISAM nodes are allocated **sequentially** during **Index construction time**
  - consequently, no need to maintain the next prev-next-pointer.

B+ Trees in Practice
(B+Trees στην Πράξη)

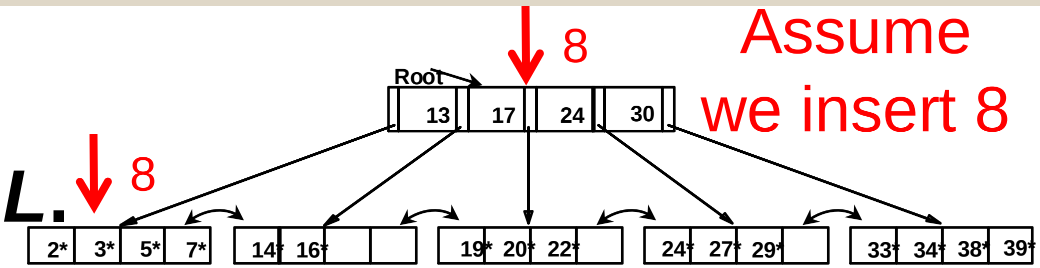
- **Typical order (d):** 100 (ie100<=#children<=**200**)
- **Typical fanout (f)** = 133
  - **Typical fill-factor: 67% (133/200)**

d=2, f=3

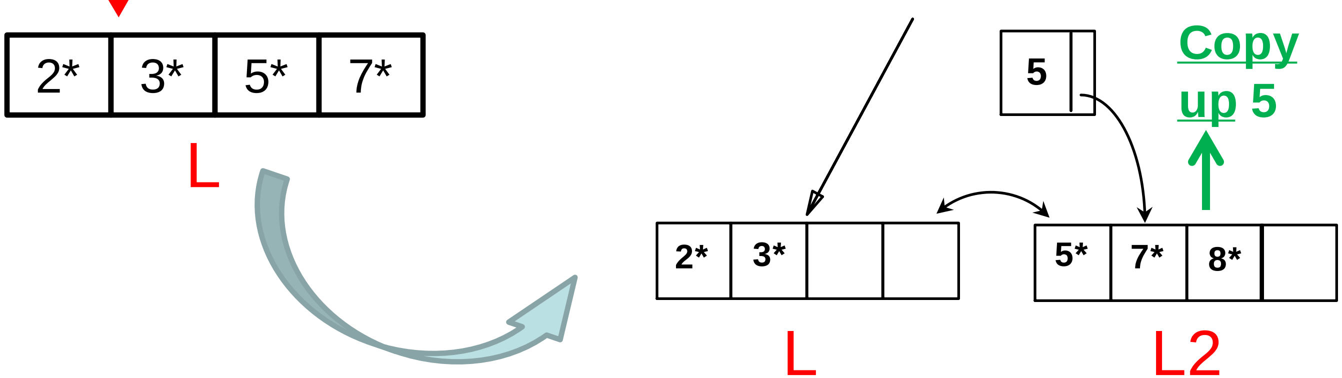


- **Typical capacities:**
  - Height 4: 133^4 = 312,900,700 records
  - Height 3: 133^3 = 2,352,637 records
- **Can often hold top levels in buffer pool:**
  - Level 1 = 133^0 = **1 page = 8 Kbytes**
  - Level 2 = 133^1 = 133 pages = ~1 MB (1064 KB)
  - Level 3 = 133^2 = 17,689 pages = ~133 MB (141,512KB)

B+ Tree Insertion Algorithm
(Αλγόριθμος Εισαγωγής στο B+Tree)



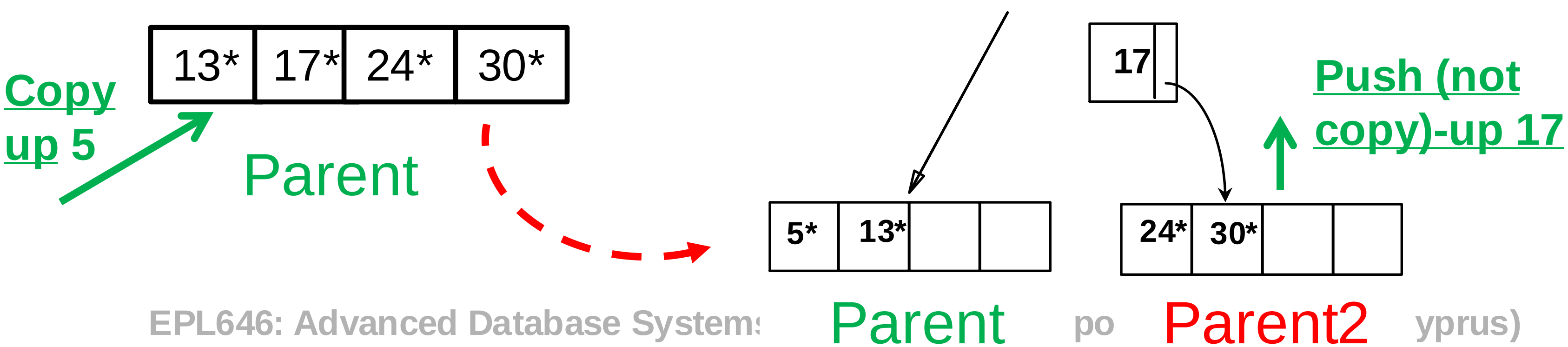
1. **Find** correct **leaf *L.***
2. **Put** data entry **onto *L*.**
   - If *L* has enough space, *done*!
   - Else *split (διαμοίρaση)* ***L*** *(into* ***L*** *and a new node* ***L2****)*
     - Redistribute (Ανακατένειμε) entries evenly between L and L2, **copy up (Αντιγραφή-Πρός-Τα-Πάνω)** middle key.
     - Insert index entry pointing to *L2* into parent of *L*.



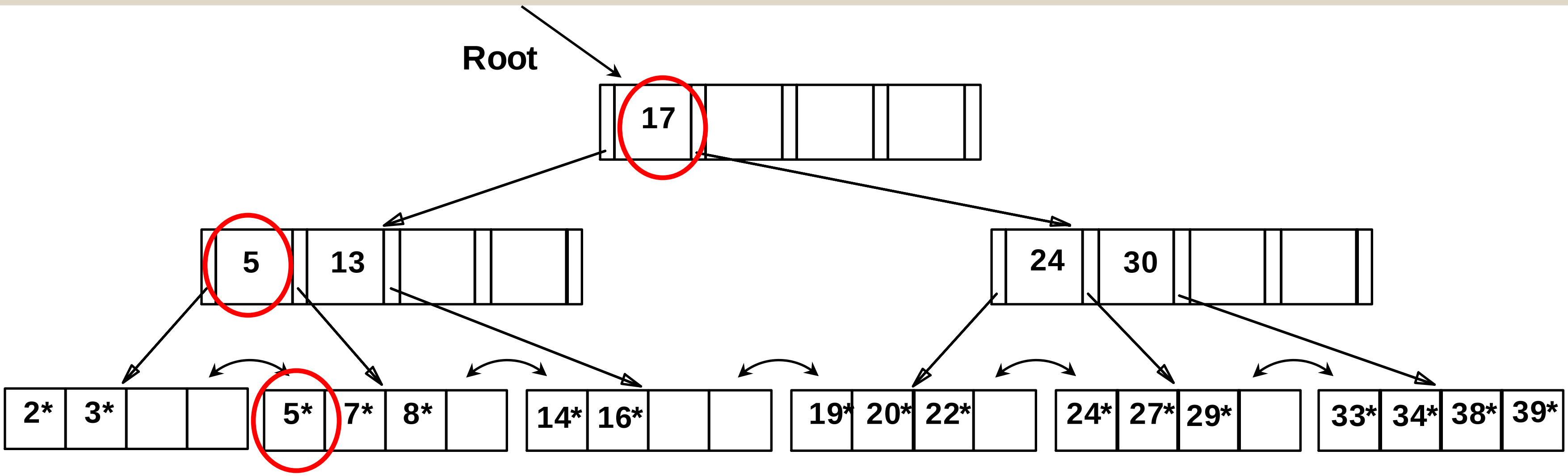
- **Copy up 5:** cannot just push-up 5 as every data entry needs to appear in a leaf node
- **Problem:** 5 won't fit in parent of L2. (see next slide)

B+ Tree Insertion Algorithm (Αλγόριθμος Εισαγωγής στο B+Tree)

3. A parent needs to recursively **Push-Up (Προώθηση-Προς-Πάνω)** the **middle key** until the insertion is successful i.e.,
   - No need to **copy-up** as the latter will generate redundant index entries.
   - If ***Parent*** has enough space, *done*!
   - Else *split (διαμοίραση)* *Parent*
     - Redistribute (Ανακατένειμε) entries evenly, **push up** middle key.
4. Splits "grow" tree; root split increases **height** (ύψος)
   - Tree growth: gets *wider* or *one level taller at top.*



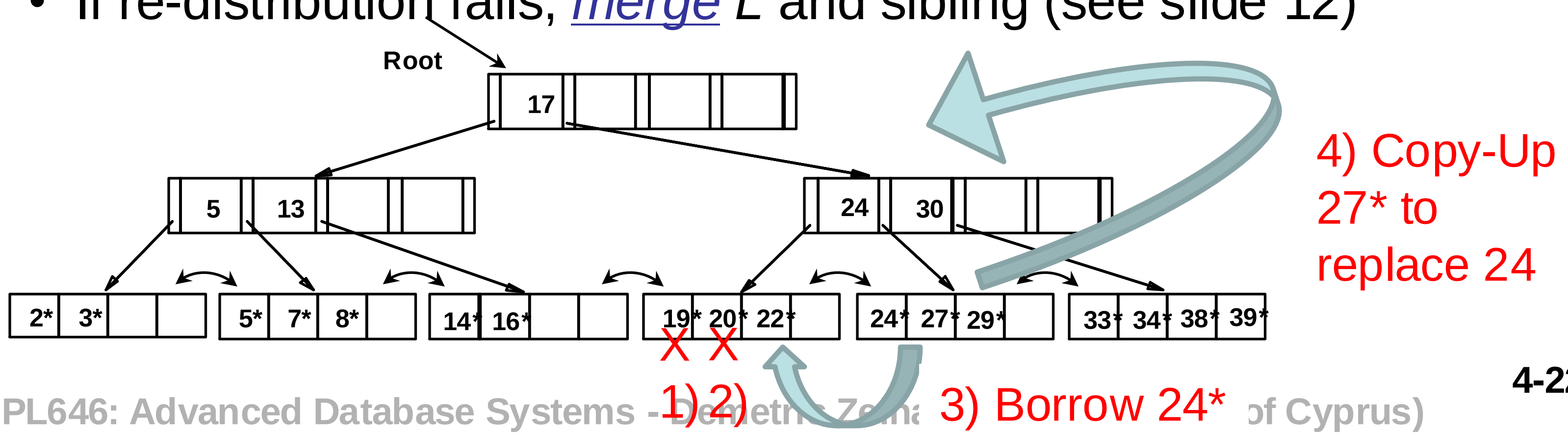
Example B+ Tree After Inserting 8* Αποτέλεσμα Εισαγωγής 8*



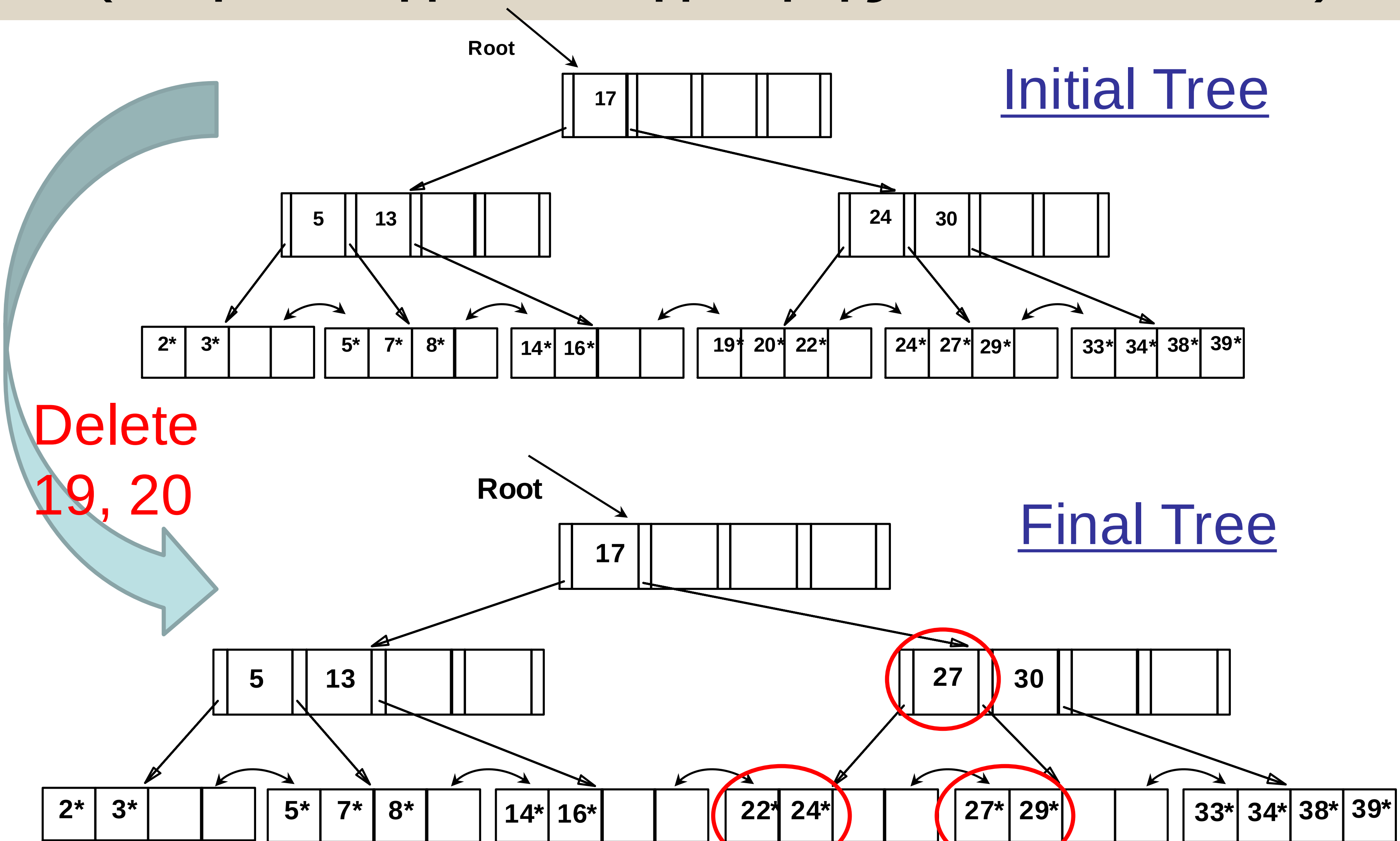
- **Root was split =>** That lead to increase in height from 1 to 2.
- **Minimum occupancy (d, i.e., 50%)** is guaranteed in both leaf and index pages splits (for root page this constraint is relaxed)
  - **Split occurs** when adding 1 key to a node that is full (has **2d entries**). Thus we will end up with two nodes, one with **d** and one with **d+1** entries.
- Can avoid split by **re-distributing entries between siblings** – (αδελφικοί κόμβοι); however, this is usually not done in practice. The borrowing practice is adopted only during deletions (see next).

B+ Tree Deletion Algorithm (Αλγόριθμος Διαγραφής απο B+Tree)

- Start at root, **find leaf *L*** where entry belongs.
  - E.g., deleting 19 then 20
- **Remove the entry K* (not respective index entries)**.
  - If L is **at least half-full**, *done! (e.g., after deleting 19*)*
  - If L has only **d-1** entries, (e.g., after deleting 20*)
    - Try to re-distribute, borrowing from *sibling* *(adjacent node with same parent as L)*. (e.g., borrow 24* and update)
    - If re-distribution fails, *merge* *L* and sibling (see slide 12)

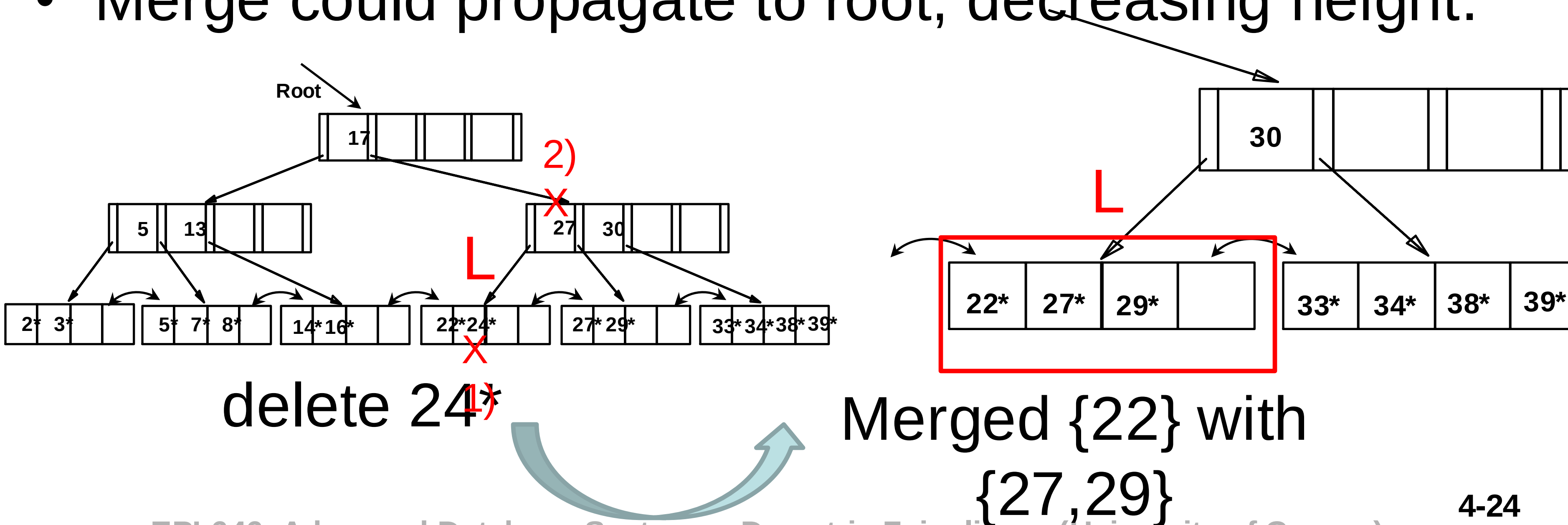


B+ Tree Deletion Example (Παράδειγμα Διαγραφής από B+Tree)



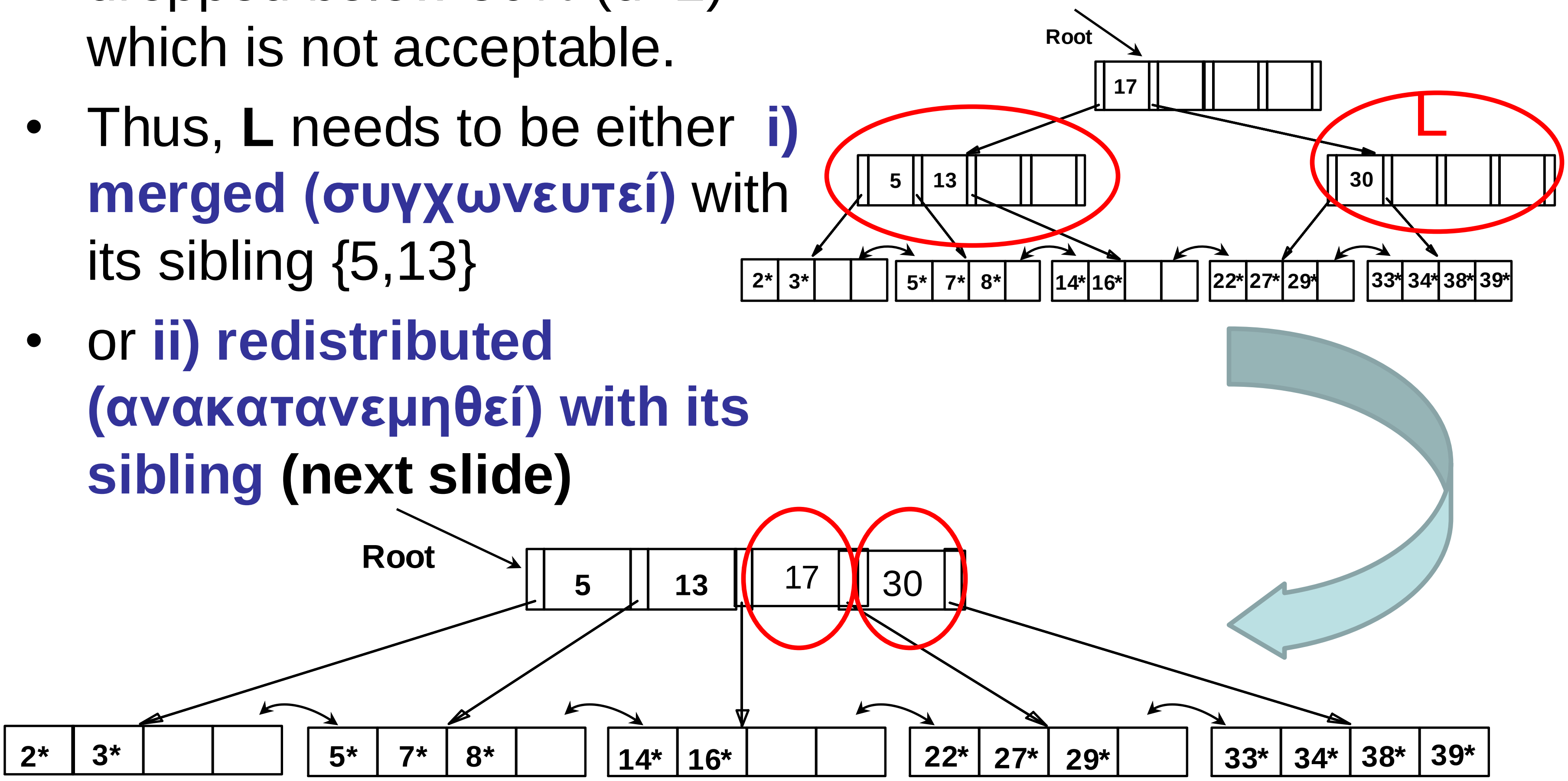
B+ Tree Deletion Algorithm (Αλγόριθμος Διαγραφή απο B+Tree)

- If re-distribution after delete fails then *merge* *L* **and sibling** (e.g., **delete 24 =>** can't borrow => merge)
- Now we also need to adjust **parent of *L*** (pointing to *L* or sibling). **(i.e., delete 27)**
- Merge could propagate to root, decreasing height.



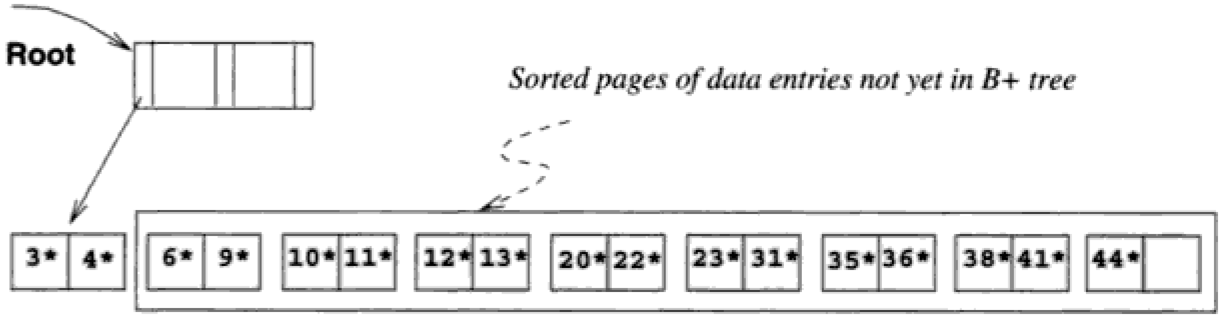
Merging propagates to sink
(Η Συγχώνευση διαδίδεται μέχρι τη ρίζα)

- But … occupancy Factor of **L** dropped below 50% (d=2) which is not acceptable.
- Thus, **L** needs to be either **i) merged (συγχωνευτεί)** with its sibling {5,13}
- or **ii) redistributed (ανακατανεμηθεί) with its sibling (next slide)**

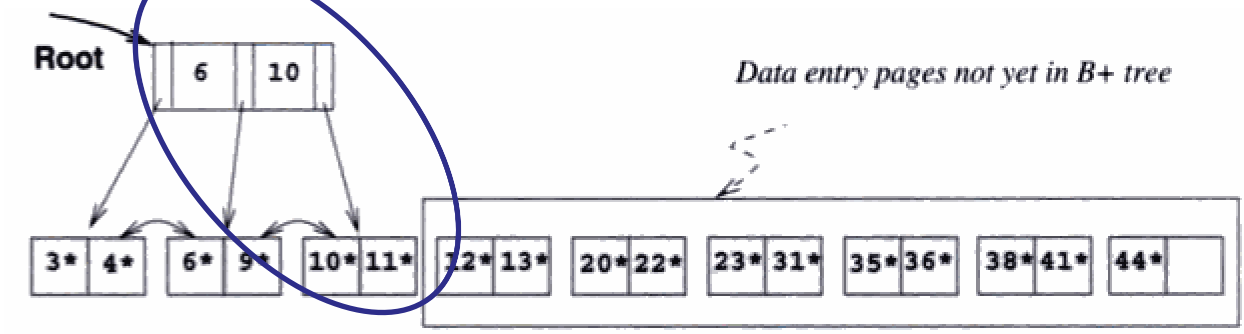


Summary of Bulk Loading (Μαζική Εισαγωγή Δεδομένων)

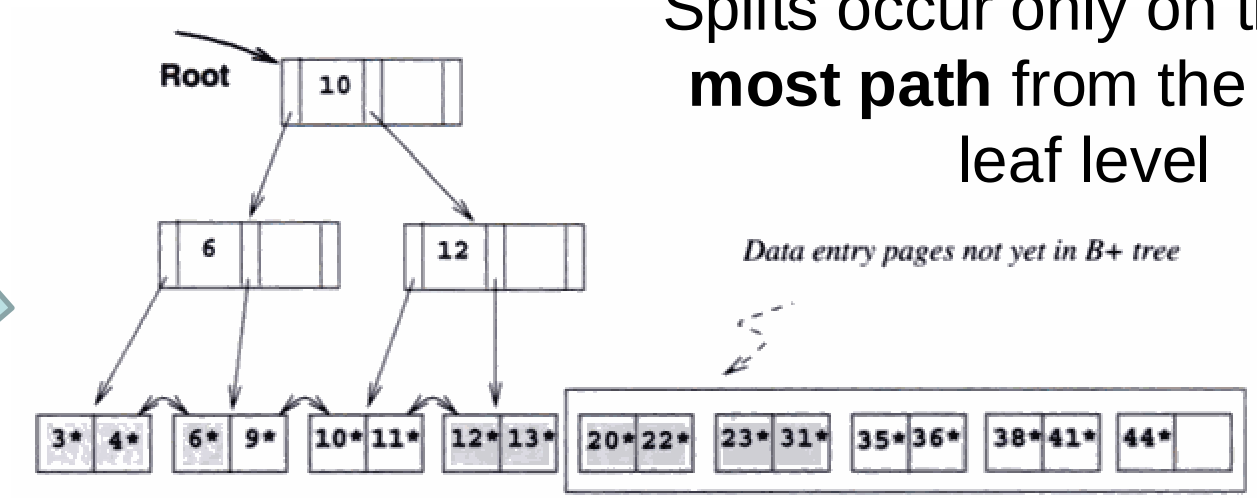
- **Scenario:** We want to **construct** a B+Tree on a **pre-existing collection (υφιστάμενη συλλογή)** of records
- **Option 1: multiple (individual) inserts.**
  - Slow and does not give sequential storage of leaves.
- **Option 2: *Bulk Loading* *(Μαζική Εισαγωγή)*.**
  - **Idea: Sort** all data entries, insert pointer to **first (leaf)** page **in a new (root)**.
  - **Effect:** Splits occur only on the **right-most path** from the root to leaves.
  - **Advantages: i**) **Fewer I/Os** during build and ii) **Leaves** will be **stored sequentially** (and linked, of course).



Bulk Loading with Example (Μαζική Εισαγωγή με Παράδειγμα)



Main Idea of Bulk Loading:
Splits occur only on the **right-most path** from the root the leaf level

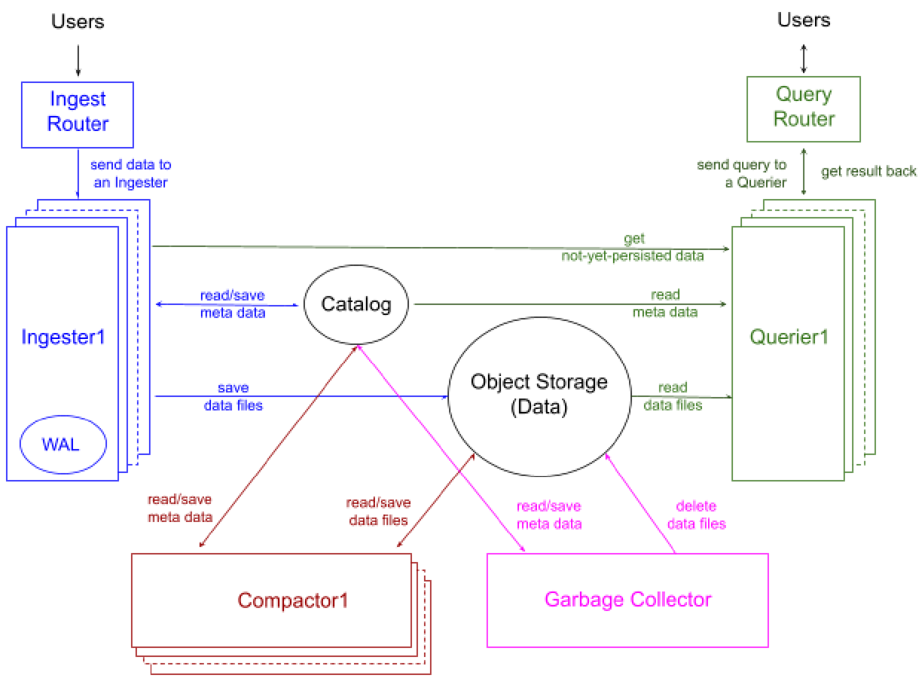


LSM Tree (Log-Structured Merge Tree)

- LSM is a data structure optimized for write-heavy workloads
  - Applications: IoT, DevOps monitoring, cybersecurity analytic stacks / analytics require **superfast & efficient data ingestion.**
  - commonly used in key-balue stores (**LevelDB, RocksDB, and Cassandra)**
  - Commonly used in **relational time series databases (TSDB)** like InfluxDB, Apache IoTDB, TimescaleDB

Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The log-structured merge-tree (LSM-tree). Acta Inf. 33, 4 (Jun 1996), 351–385. https://doi.org/10.1007/s002360050048

LSM Trees (in action)



[InfluxDB 3.0]

Written in Rust (successor for C/C++ with performance, type safety, memory safety, and concurrency.)

☐ include secondary database models — 39 systems in ranking, May 2022

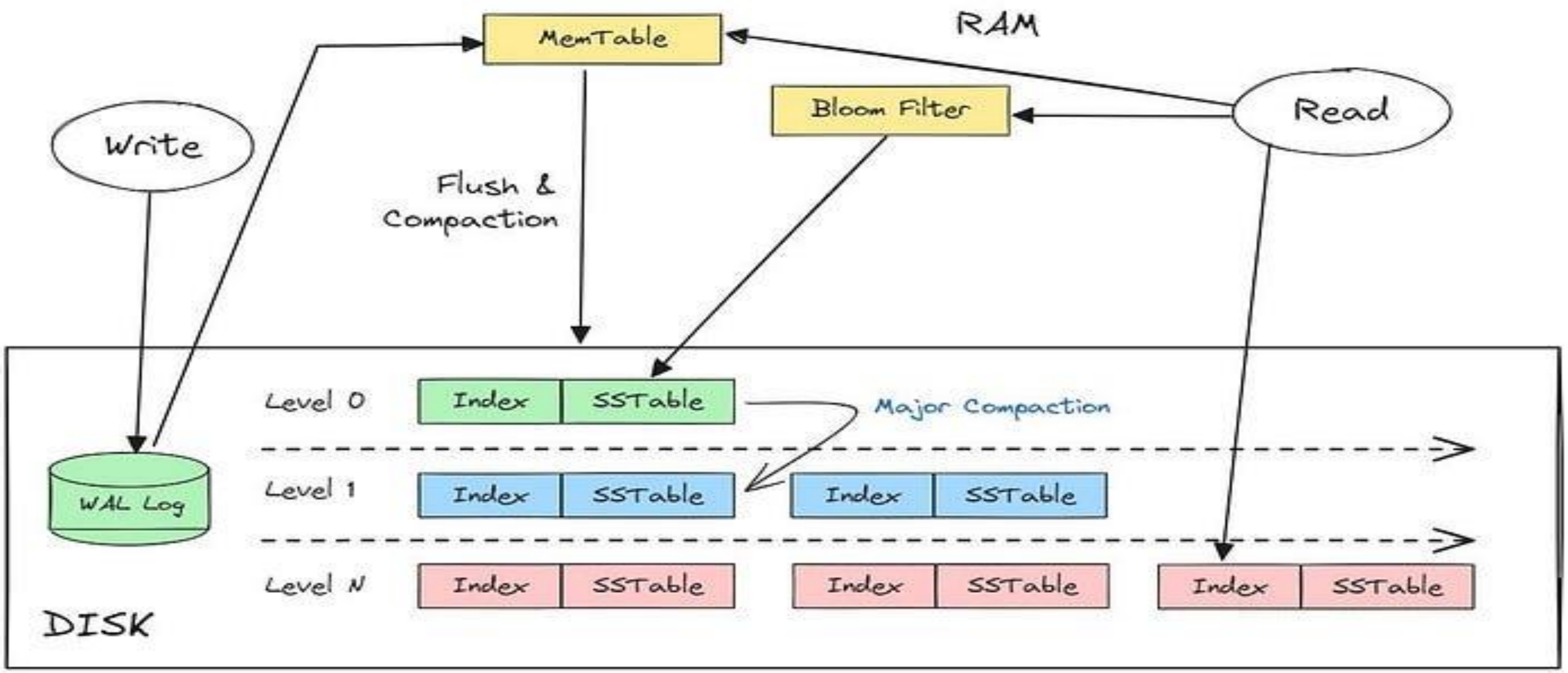
Rank May 2022	Rank Apr 2022	Rank May 2021	DBMS	Database Model	Score May 2022	Score Apr 2022	Score May 2021
1.	1.	1.	InfluxDB	Time Series, Multi-model	29.55	-0.47	+2.38
2.	2.	2.	Kdb+	Time Series, Multi-model	8.98	+0.21	+0.72
3.	3.	3.	Prometheus	Time Series	6.13	-0.18	+0.37
4.	4.	4.	Graphite	Time Series	5.46	+0.10	+0.90
5.	5.	5.	TimescaleDB	Time Series, Multi-model	4.70	+0.14	+1.80
6.	6.	6.	Apache Druid	Multi-model	3.00	-0.17	+0.33
7.	7.	7.	RRDtool	Time Series	2.50	-0.08	+0.04
8.	8.	8.	OpenTSDB	Time Series	1.84	+0.02	+0.03
9.	9.	↑ 11.	DolphinDB	Time Series, Multi-model	1.65	+0.03	+0.75
10.	10.	↓ 9.	Fauna	Multi-model	1.36	-0.05	-0.12
11.	11.	↓ 10.	GridDB	Time Series, Multi-model	1.23	-0.05	+0.20
12.	12.	↑ 16.	QuestDB	Time Series, Multi-model	1.19	+0.03	+0.74

LSM Trees: Overview

- Write:
  - **MemTable** + WAL
  - Flush/Compact in **SSTable**

Read:

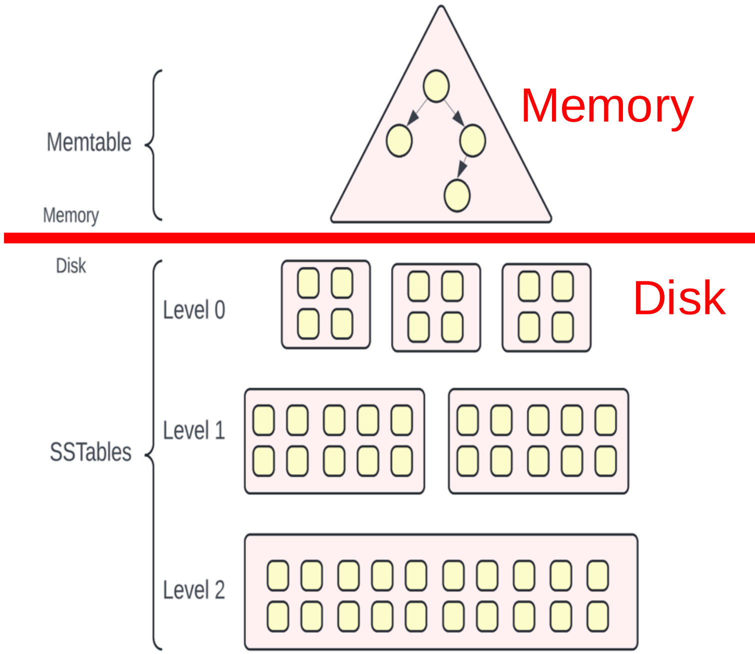
Bloom Filter + MemTable



https://medium.com/@abhay.nimmagadda/lsm-trees-a-deep-dive-into-a-fundamental-data-structure-used-in-databases-e2622919ec6b

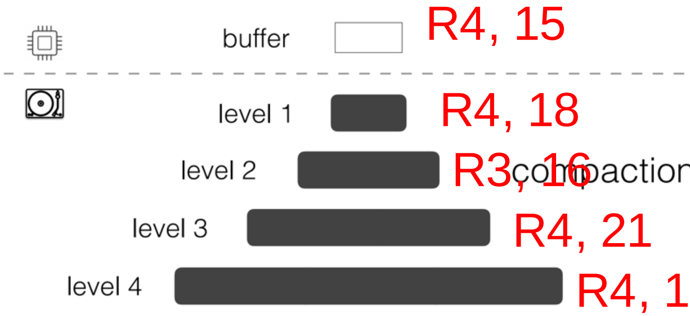
LSM Trees: Writes

- **Memtable**
  - **in-memory balanced tree** (usually a **Red-Black Tree** or **Skip List**)
  - **sorted order** to facilitate fast reads and efficient merging later.
- **Write-Ahead Log (WAL) for Durability**
  - Part of every DB offering recovery!
  - We will focus later in the course on this extensively!
- When the memtable reaches a predefined size (e.g., 64MB), it is flushed to disk as an immutable **Sorted String Table (SSTable).**
  - The SSTable is:**Sorted** → No need for additional indexing.
  - **Immutable** → No in-place updates (only compaction merges them).
  - Stored alongside **a Bloom filter** (for fast existence checks) and **an index** (for range lookups).



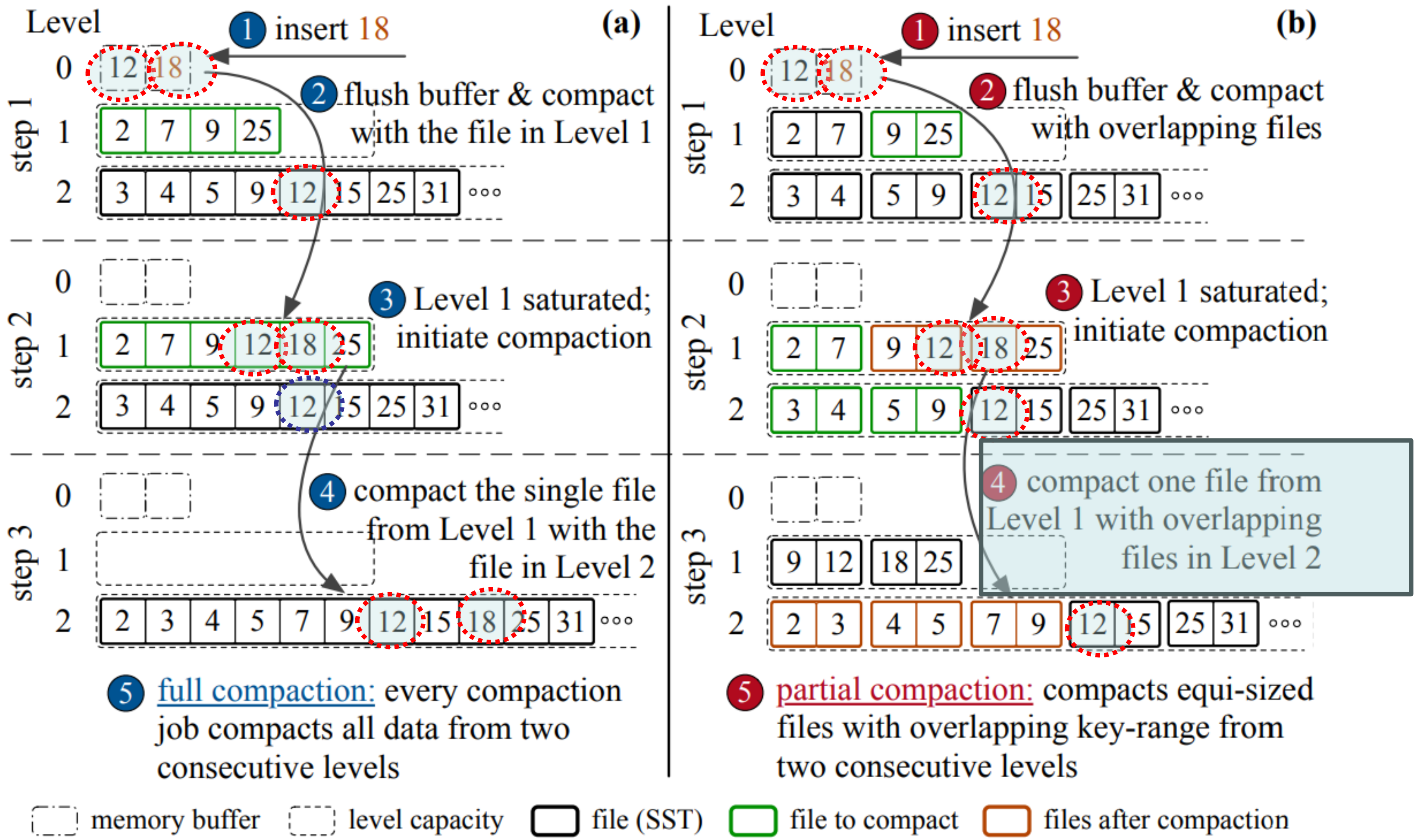
https://kflu.github.io/2018/02/09/2018-02-09-lsm-tree/

LSM Trees: Write (Compaction)

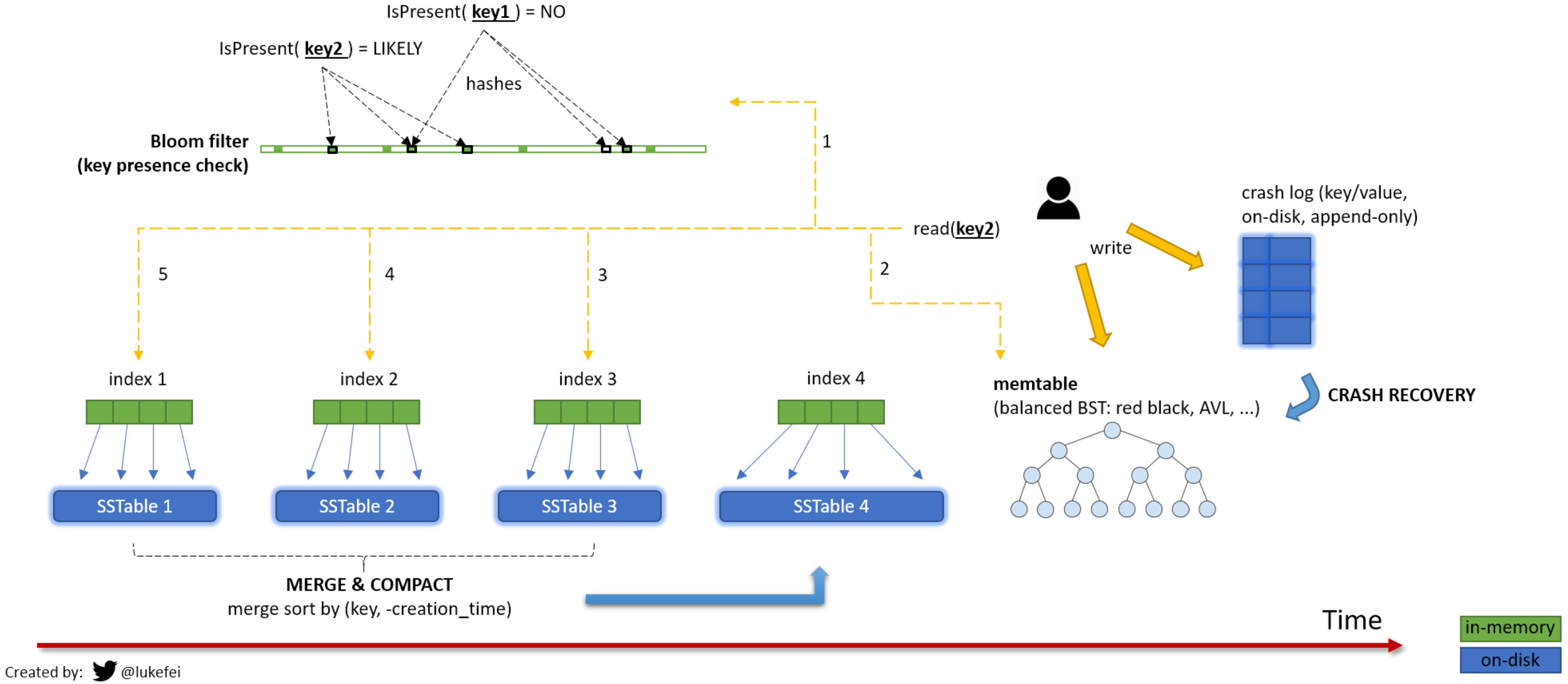


https://disc-projects.bu.edu/compactionar ckground.html

LSM Trees: Write (Compaction)



LSM Trees: Read



Time Series Database (TSDB)

- What is **time series data?**
  - Time Series is an ordered sequence of values of a variable (e.g.temperature) at equally spaced time intervals (e.g. hourly).
- What is **Time Series Databases?**
  - A Time Series Database (TSDB) is a database type which is optimized for time series or time-stamped data.
- Benefits:
  - ingestion performance, range queries by timestamp, compression, scalability, sql support

The following query covers a 12-minute time range and groups results into 12-minute time intervals,

- **SELECT COUNT**("water_level") **FROM** "h2o_feet" **WHERE** "location"='coyote_creek' **AND** time >= '2015-08-18T00:06:00Z' **AND** time < '2015-08-18T00:18:00Z' **GROUP BY** time(12m)
- name: h2o_feet
- time **count**
- ---- -----
- 2015-08-18T00:00:00Z 1 <----- *Note that this timestamp occurs before the start of the query's time range* 2015-08-18T00:12:00Z 1

Source: http://tiny.cc/0jt8001

Time Series Database (TSDB)

Rank			DBMS	Database Model	Score		
Aug 2024	Jul 2024	Aug 2023			Aug 2024	Jul 2024	Aug 2023
1.	1.	1.	InfluxDB	Time Series, Multi-model	22.63	-0.97	-7.24
2.	2.	2.	Kdb	Multi-model	7.97	+0.39	-0.46
3.	3.	3.	Prometheus	Time Series	7.17	-0.17	-0.68
4.	4.	4.	Graphite	Time Series	5.35	+0.30	-0.25
5.	5.	5.	TimescaleDB	Time Series, Multi-model	4.07	+0.17	-1.05
6.	↑ 7.	↑ 8.	Apache Druid	Multi-model	2.95	+0.02	-0.34
7.	↑ 8.	↑ 10.	QuestDB	Time Series, Multi-model	2.87	+0.21	+0.34
8.	↓ 6.	↓ 6.	DolphinDB	Multi-model	2.77	-0.85	-0.72
9.	9.	9.	TDengine	Time Series, Multi-model	2.51	+0.06	-0.17
10.	10.	↑ 12.	GridDB	Time Series, Multi-model	1.98	-0.01	-0.21
11.	11.	↓ 7.	RRDtool	Time Series	1.72	-0.02	-1.58
12.	12.	↓ 11.	OpenTSDB	Time Series	1.60	-0.02	-0.66
13.	13.	13.	Fauna	Multi-model	1.57	+0.08	-0.22
14.	14.	↑ 20.	Apache IoTDB	Time Series	1.28	+0.04	+0.46

TSM (Time-Structured Merge Tree)

InfluxDB is based on **TSM (Time-Structured Merge Tree),** very similar to LSM only more optimized for timeseries data not key-values

name=passengers				
time	**minors**	**adults**	**location**	**driver**
2015-08-18T00:00:00Z	1	2	1	doe
2015-08-18T00:00:00Z	2	2	1	jones
2015-08-18T00:06:00Z	1	1	1	doe
2015-08-18T00:06:00Z	0	1	1	jones
2015-08-18T05:54:00Z	0	2	2	doe
2015-08-18T06:30:00Z	2	2	2	doe
2015-08-18T06:06:00Z	3	1	2	jones
2015-08-18T06:30:00Z	0	4	2	jones

Table 1: Sample time series dataset.

⚡ Core Differences: TSM vs. LSM

Aspect	TSM (Time-Structured Merge Tree)	LSM (Log-Structured Merge Tree)
Primary Use Case	Optimized for **time-series data** (e.g., metrics, logs, IoT).	General-purpose for **key-value stores** (e.g., Cassandra, RocksDB).
Data Model	Time-stamped data with a focus on **time ranges.**	Key-value pairs optimized for **random reads/writes.**
Compaction Strategy	**Time-based compaction** to merge data based on time intervals.	**Level-based or size-tiered compaction** to manage write amplification.
Compression	Advanced **time-based compression** (e.g., Gorilla, delta encoding).	General compression algorithms without time optimizations.
Indexing	**TSM index** maps time ranges efficiently.	Indexes keys for fast lookups (often via bloom filters).
Read Optimization	Optimized for **range queries** over time (e.g., last 24 hours).	Optimized for **point lookups** (specific key retrieval).
Write-Ahead Log (WAL)	WAL is tightly coupled with time-series data ingestion patterns.	WAL handles general transactional durability.
Deletion Strategy	Data expiration via **retention policies** (automatic pruning).	Requires manual deletion or TTL mechanisms.

Retention Policies in InfluxDB

A **retention policy (RP)** in **InfluxDB** defines how long data is kept in a database before it is automatically deleted. It also controls how many copies of the data are stored (replication factor, mainly for InfluxDB Enterprise/Cluster setups).

Key Components of a Retention Policy:

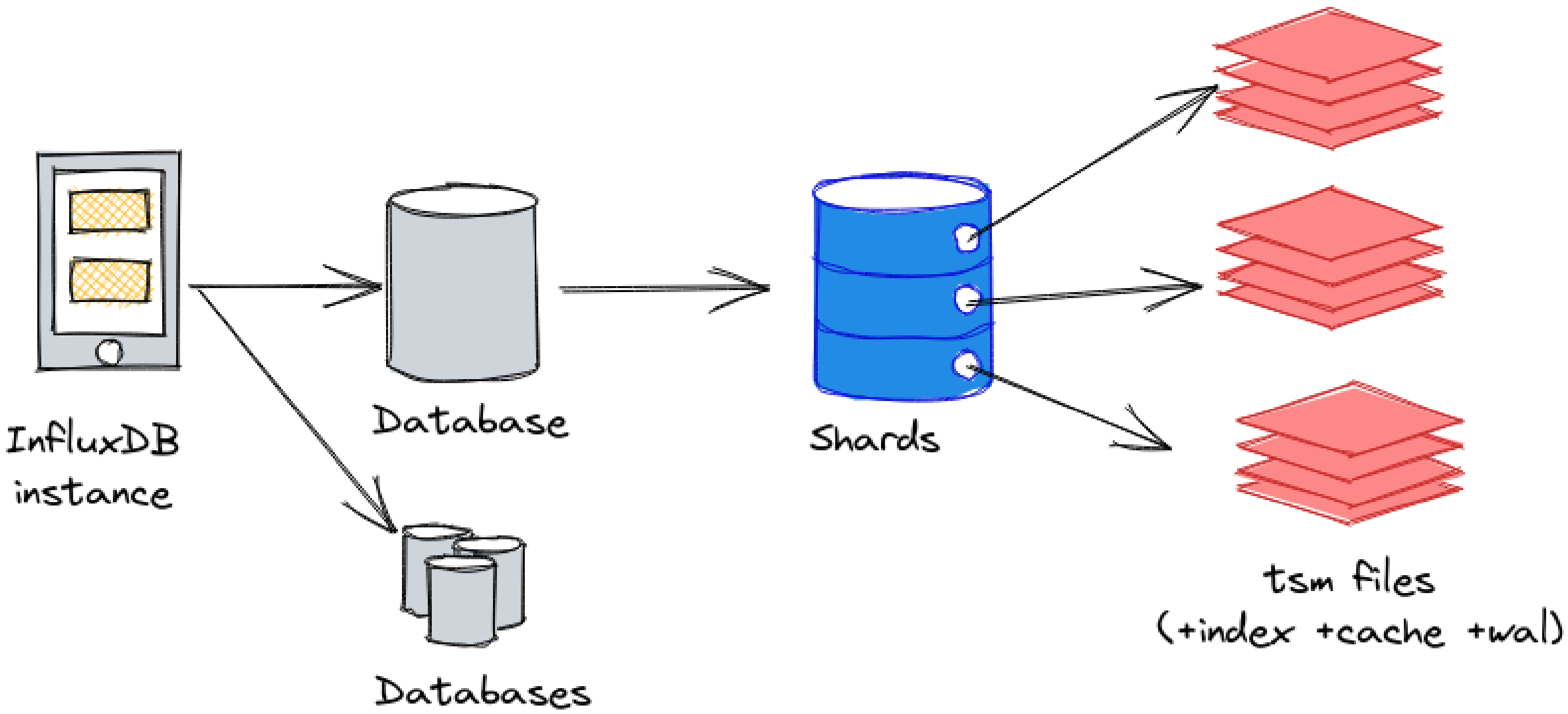
1. **Duration**: How long InfluxDB keeps the data (e.g., 30d, 90d, INF for infinite).
2. **Replication Factor**: Number of copies of the data (relevant in clustered setups).
3. **Shard Duration**: Defines the time range covered by each shard. InfluxDB manages shards internally, but you can customize this if needed.
4. **Default Policy**: One RP can be set as the default for a database. If no RP is specified in a query, the default RP is used.

Create a Retention Policy:

```
CREATE RETENTION POLICY "30d_policy" ON "mydb" DURATION 30d REPLICATION 1 DEFAULT
```

- **"30d_policy"**: Name of the retention policy.
- **"mydb"**: Target database.
- **DURATION 30d**: Data will be kept for 30 days.
- **REPLICATION 1**: One copy of the data (default for single-node setups).
- **DEFAULT**: Makes this the default RP for the database.

Sharding in InfluxDB



InfluxDB Operators

Functions and Operators

Aggregation
1. MEAN
2. INTEGRAL
3. MODE
4. STANDARD DEVIATION

Selectors
1. PERCENTILE
2. SAMPLE
3. TOP
4. BOTTOM

Transformations
1. HISTOGRAM
2. MOVING AVERAGE
3. DERIVATIVE

Predictors
1. HOLT WINTERS
2. HOLT WINTERS WITH-FIT

Influx Query Language (InfluxQL)

To be covered in laboratory

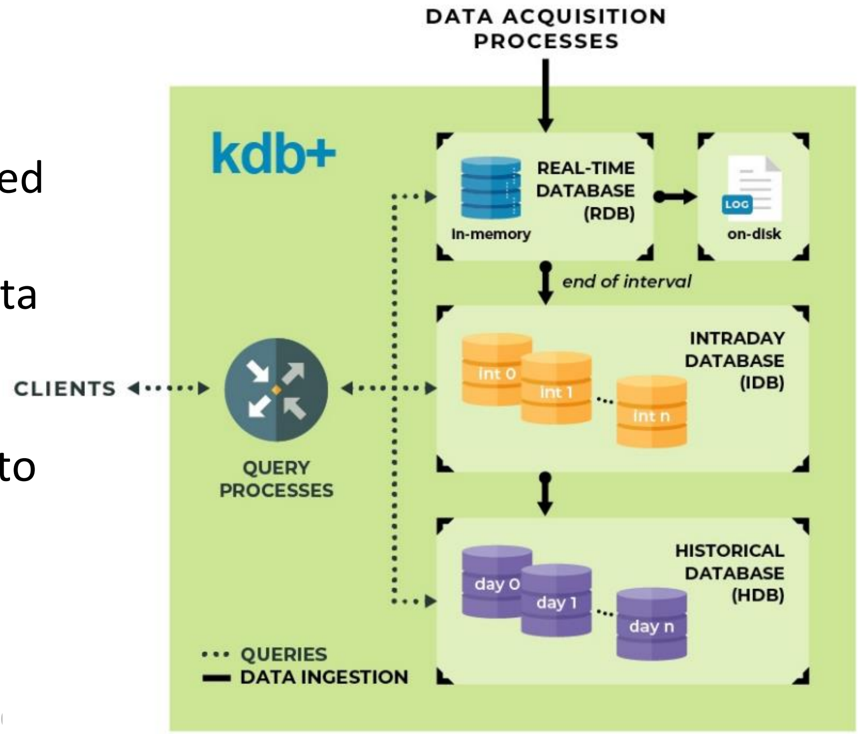
> Like SQL but optimized for **continuous / sliding window queries**

https://docs.influxdata.com/influxdb/v1/query_language/functions/

kdb+ Time series DB

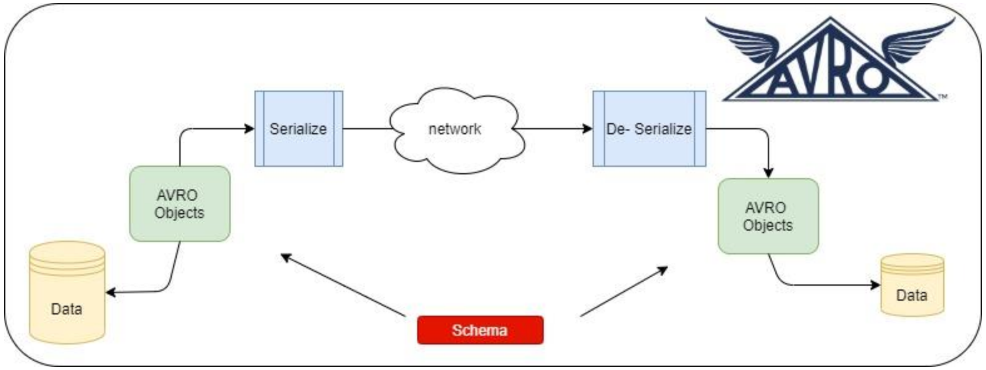
- **kdb+** is a column-based relational time series database (TSDB) with in-memory (IMDB) abilities, developed and marketed by KX.[1] The database is commonly used in high-frequency trading (HFT) to store, analyze, process, and retrieve large data sets at high speed.[2] kdb+ has the ability to handle billions of records and analyzes data within a database.[3]

The **intraday historical database (*IHDB*)** is used to store intraday data of the current day. Generally this is used for tables with large data volumes. Data saved in the *IHDB* will already be partitioned by sym and sorted by time. At the end of day, all data will be written down to the *HDB* and deleted from the *IHDB*.



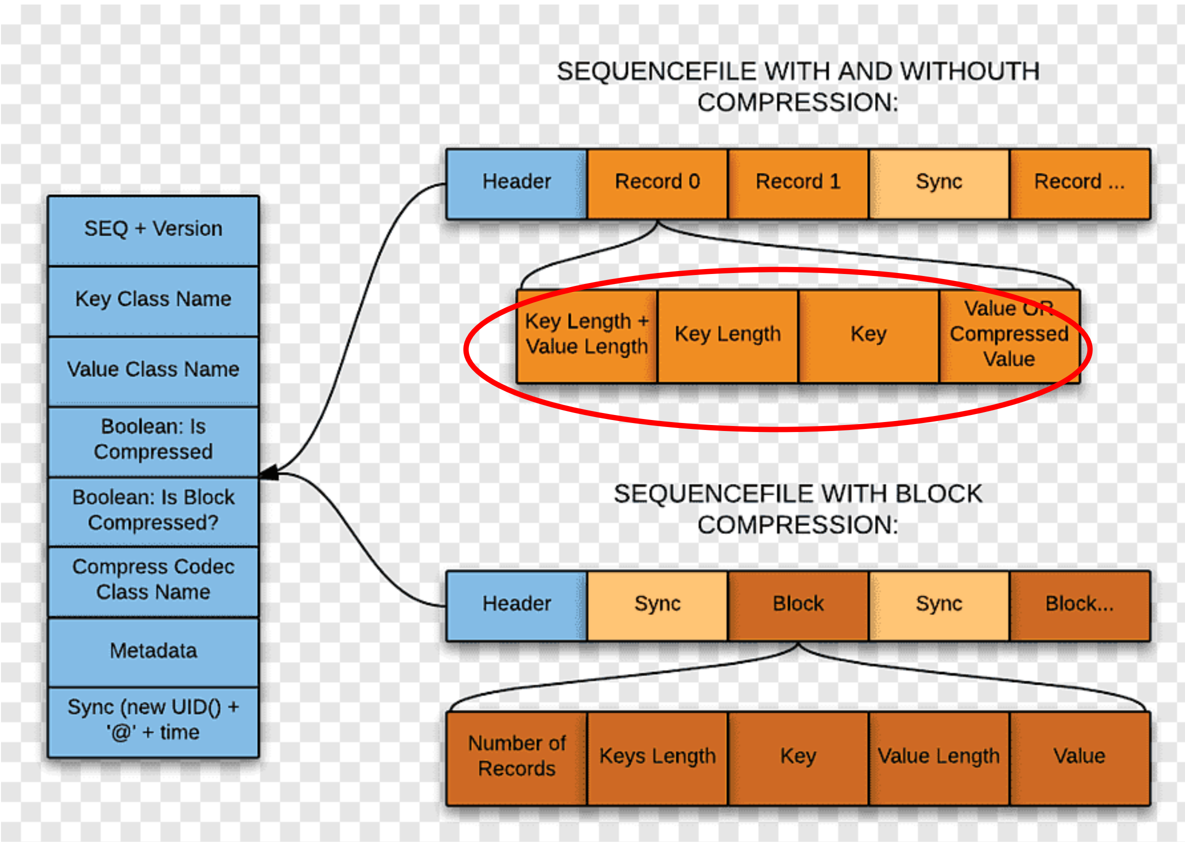
Row Serialization

- Apache Avro™ is the **leading serialization format for record data, and first choice for streaming data pipelines**. It offers excellent schema evolution, and has implementations for the JVM (Java, Kotlin, Scala, …), Python, C/C++/C#, PHP, Ruby, Rust, JavaScript, and even Perl.



Avro

AVRO Internals (Row Serialization)



Apache Avro is widely supported across various systems, including:

Big Data & Storage Systems

- **Apache Hadoop** (HDFS, MapReduce)
- **Apache Hive**
- **Apache HBase**
- **Apache Kafka** (often used for schema-based messaging)
- **Apache Flink**
- **Apache Spark**
- **Apache NiFi**

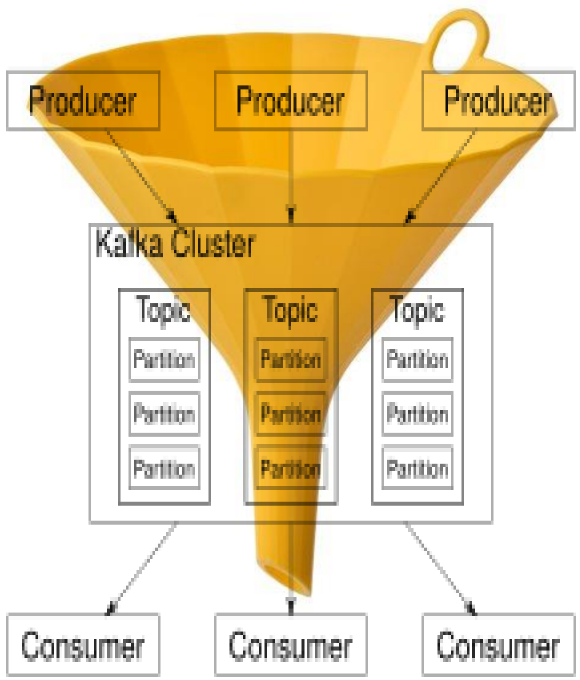
Databases

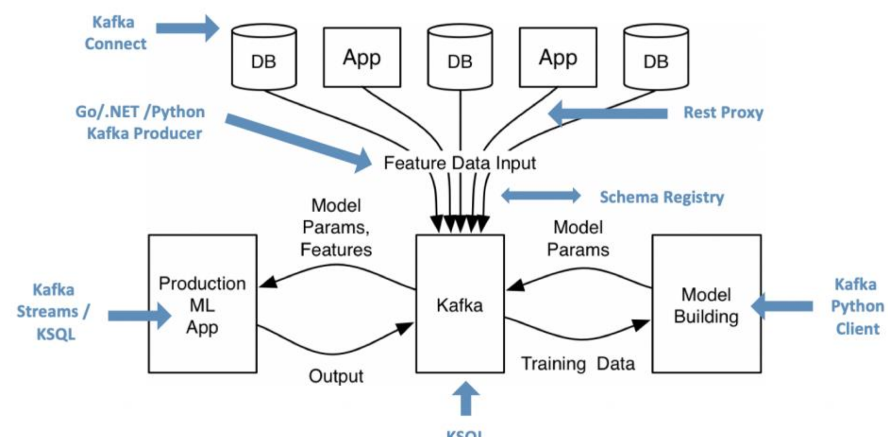
- **Google BigQuery**
- **AWS Glue**
- **Azure Data Lake**
- **Snowflake** (supports reading Avro)
- **PostgreSQL** (via extensions like avro_fdw)

```
// PostgreSQL

CREATE SERVER avro_server FOREIGN D
WRAPPER avro_fdw;

CREATE FOREIGN TABLE avro_data (
   id INTEGER,
   name TEXT,
   age INTEGER
) SERVER avro_server
OPTIONS (filename '/path/to/data.avro');
```

Row Ingestion (Data Funnels - χωνιά)

Apache Kafka is an open-source **distributed event streaming platform** used by thousands of companies for high-performance **data pipelines, streaming analytics, data integration, and mission-critical applications.**





Will be discussed again later in the course