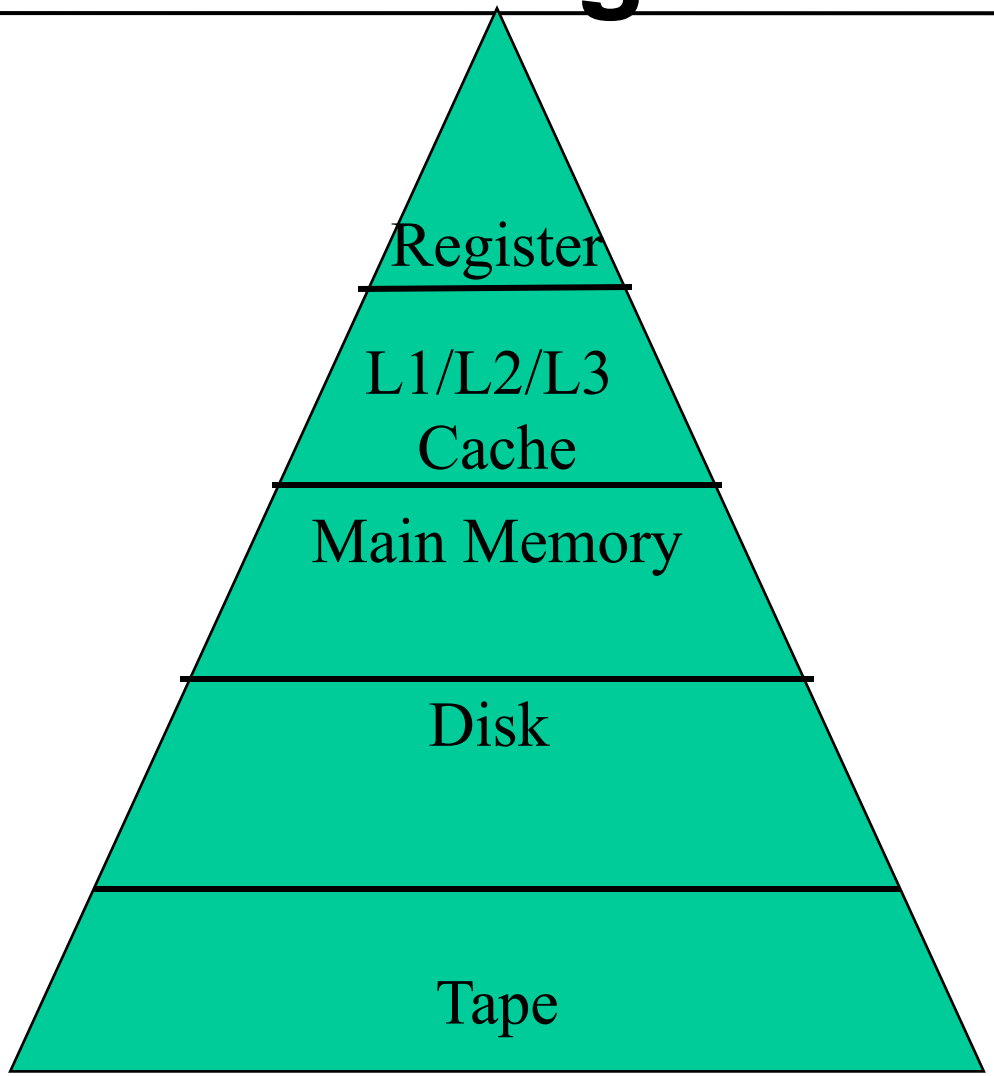
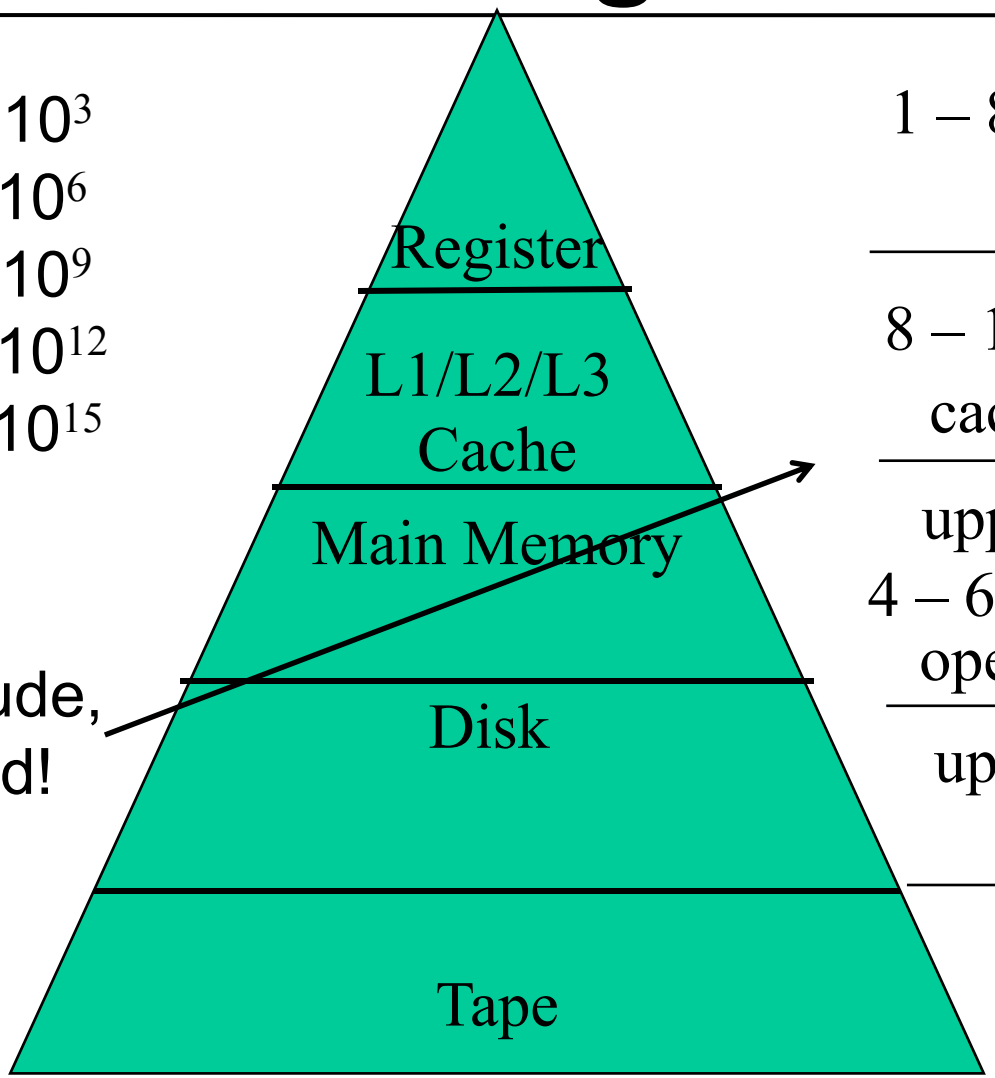
Overview: Storage Hierarchy



Overview: Storage Hierarchy

1 K (Kilo) = 10^3
1 M (Mega) = 10^6
1 G (Giga) = 10^9
1 T (Tera) = 10^{12}
1 P (Peta) = 10^{15}



1 – 8 Byte/Register
Compiler

8 – 128 Byte/Cache
cache-controller

upper GB-range,
4 – 64 KB block size
operating system

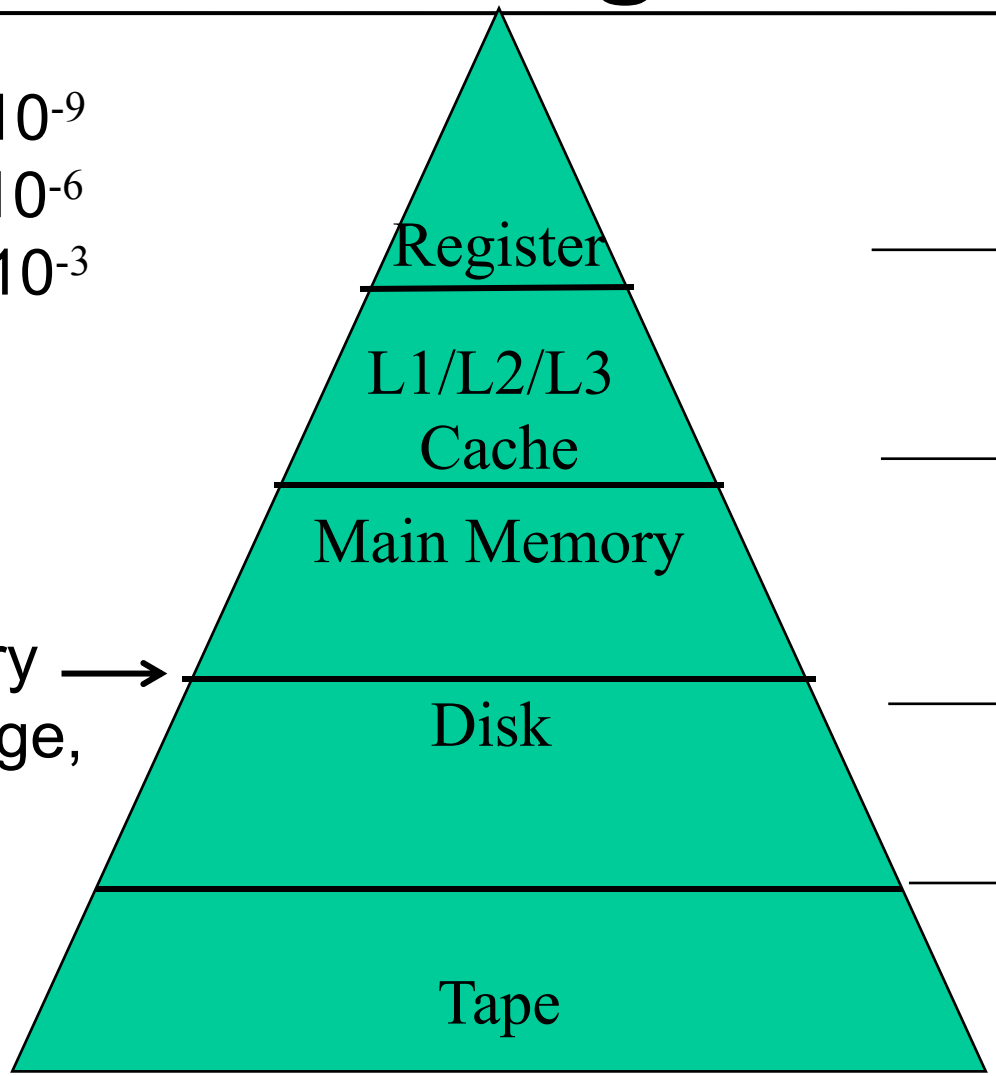
upper TB-range
user

PB-range
user

Rough magnitude,
rapidly outdated!

Overview: Storage Hierarchy

1 n (nano) = 10^{-9}
1 µ (micro) = 10^{-6}
1 m (milli) = 10^{-3}



< 1ns

< 10 ns

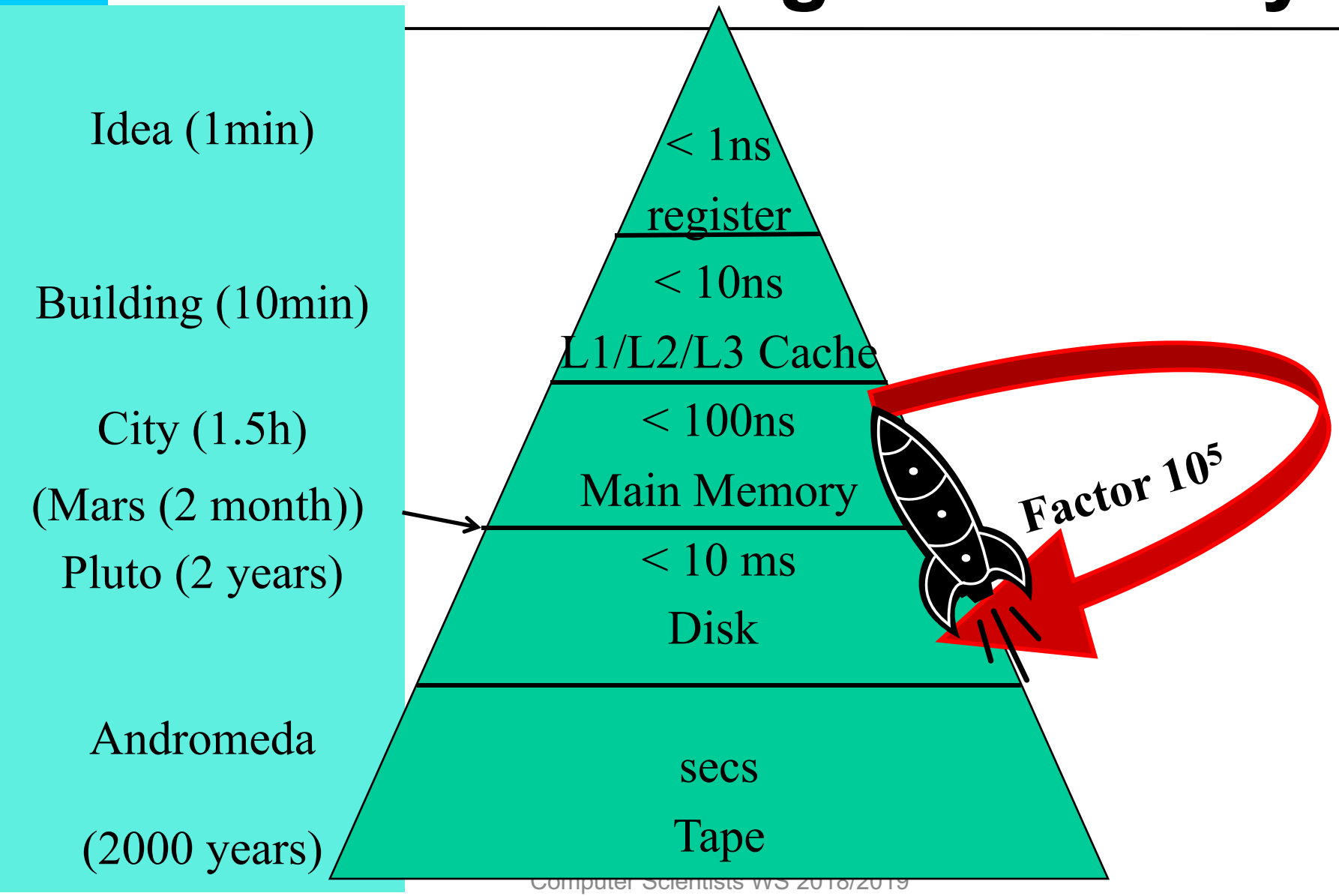
< 100 ns

(Flash-Memory → Lower TB-range, < 100 µs)

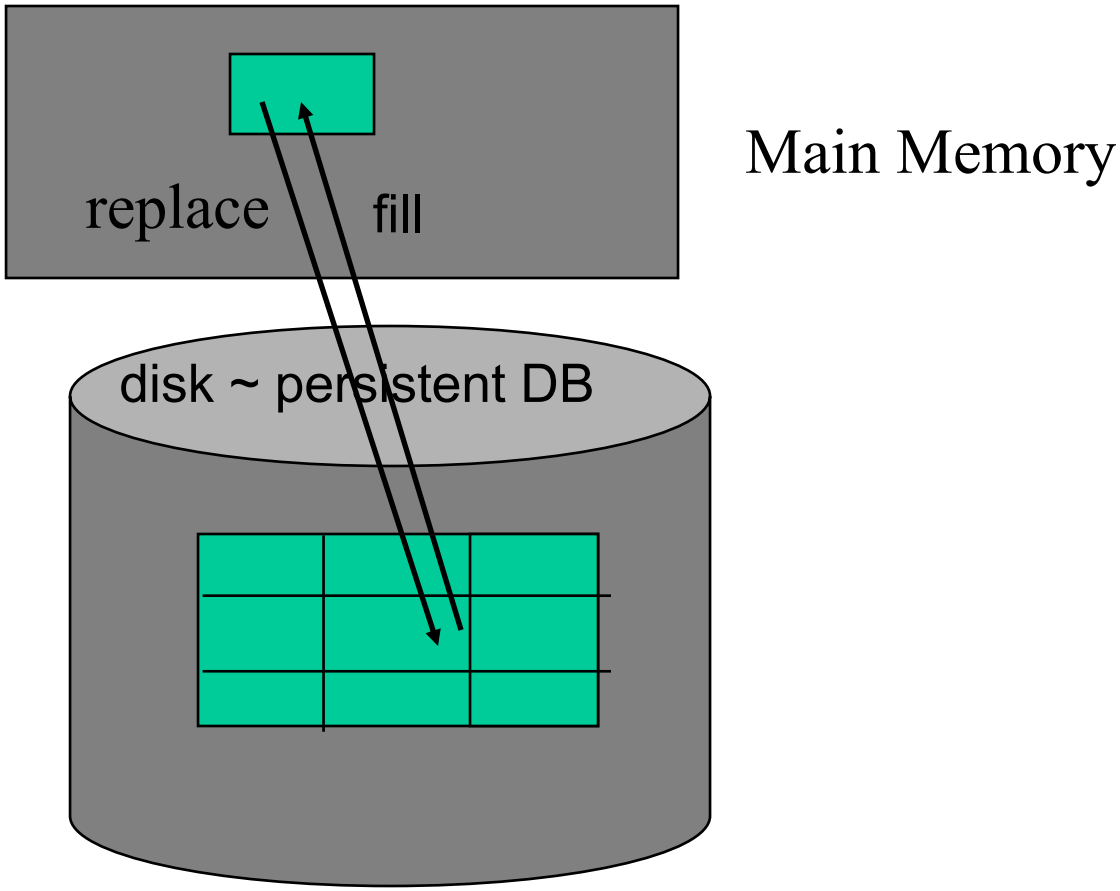
< 10 ms

secs

Overview: Storage Hierarchy

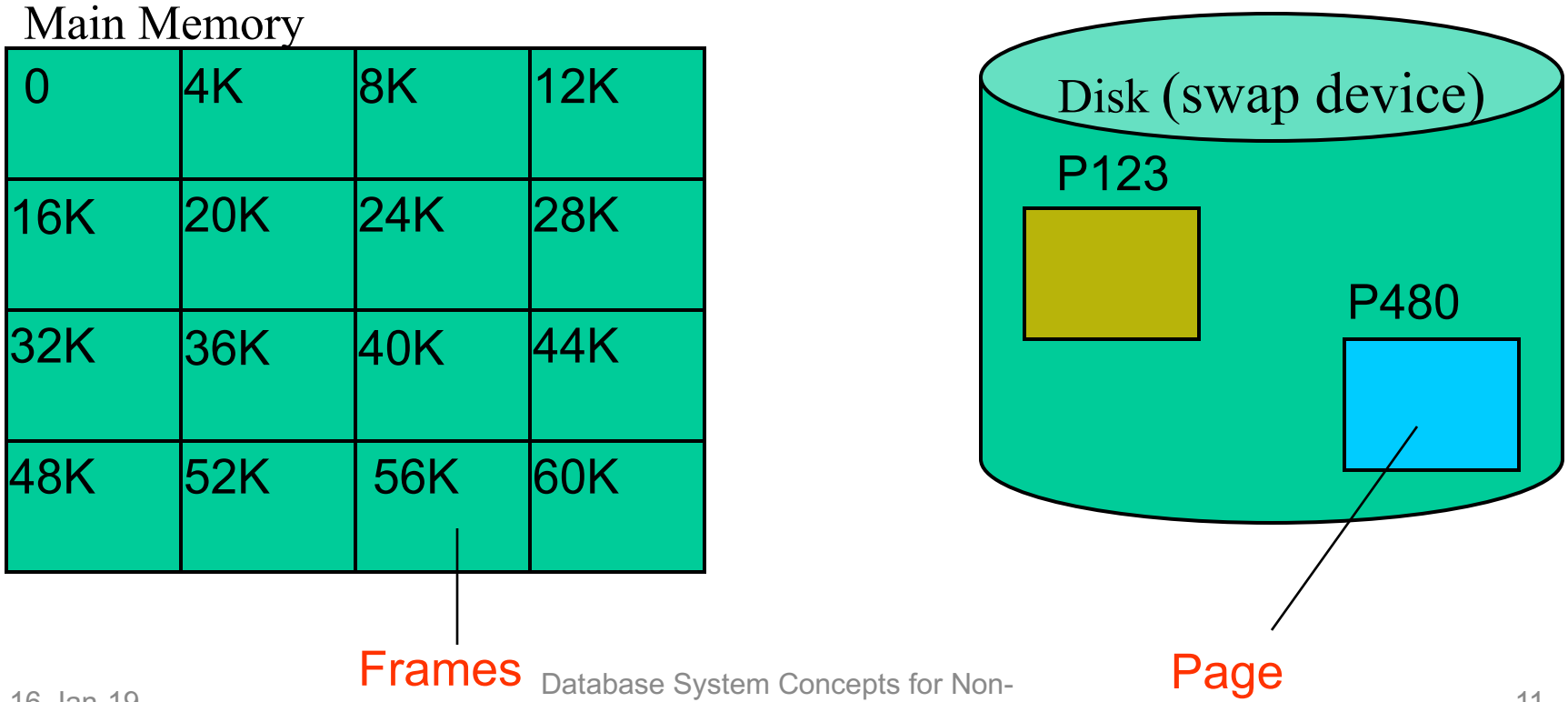


Buffer Management

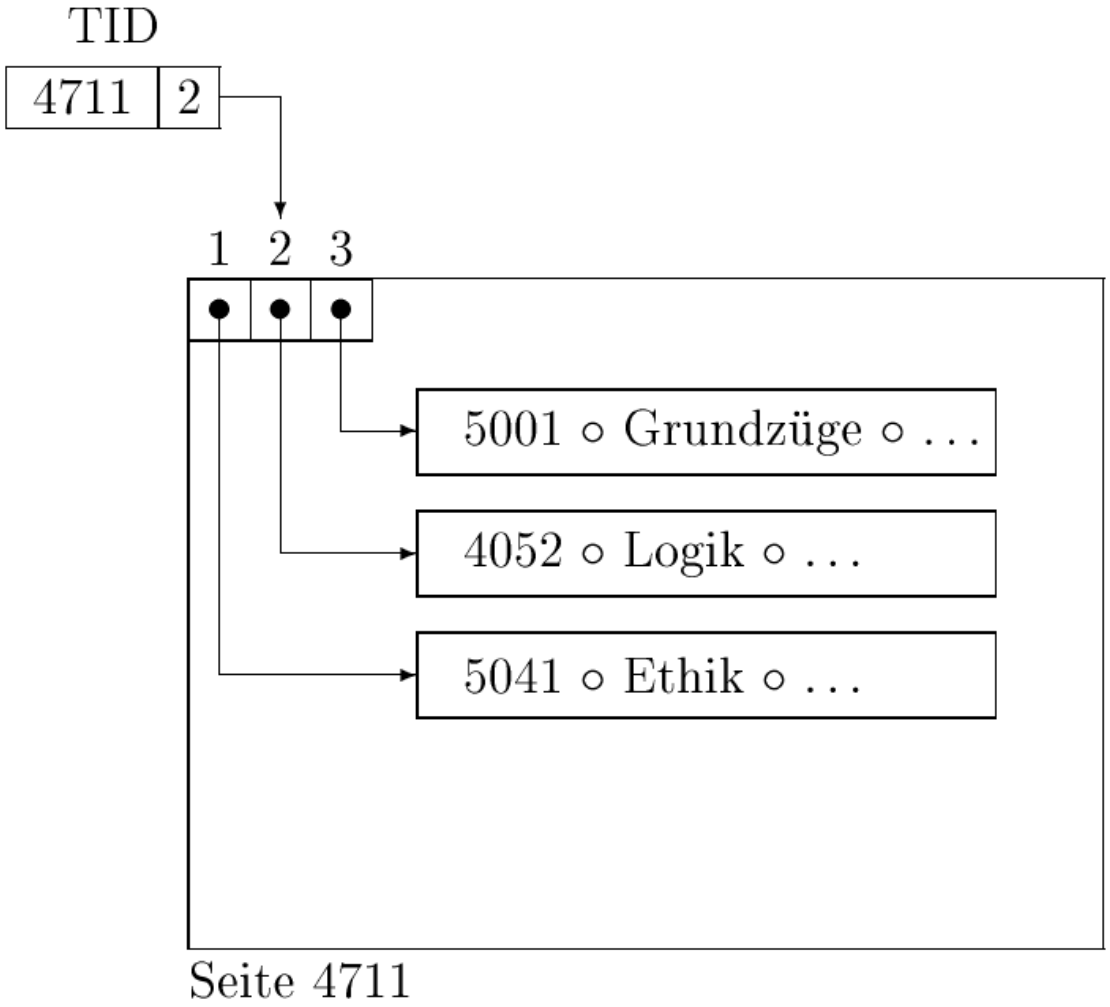


Fill and replace pages

- System buffer is divided in frames of equal size
- A frame can be filled with one page (block, sector)
- Overflow pages are swapped on disk



Addressing tuples on disk



Data Transfer

Simple query execution:

select * **from** students **where** studNr=26120;

Get one tuple after the other to the main memory and evaluate predicates.

→ Most expensive way ☹

→ Mostly only a small fraction of the tuples fulfills the query

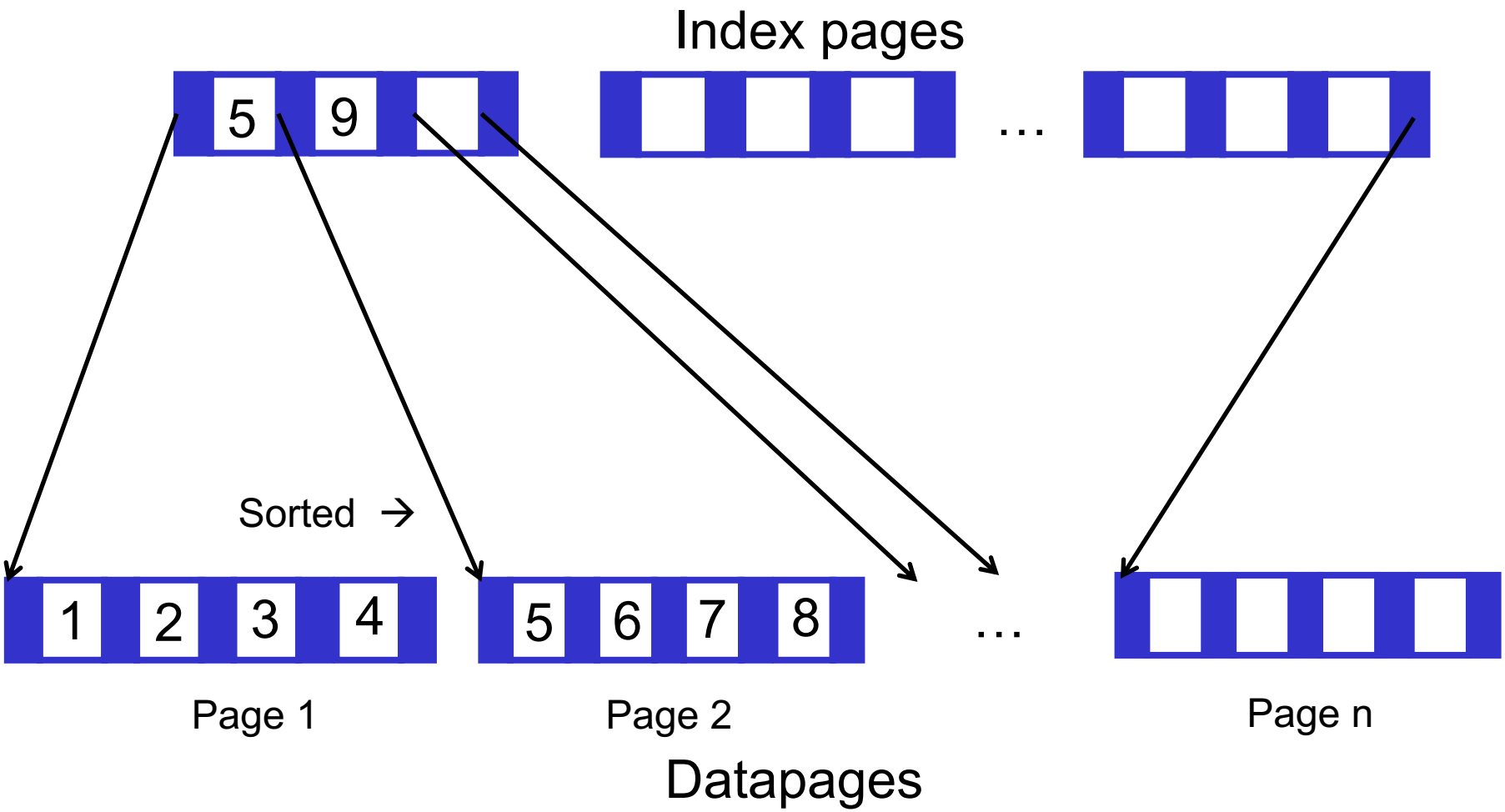
Hierarchical Indexes

We consider two hierarchical index structures:

- ISAM (Index-Sequential Access Method)
- B-Trees

- ISAM is the predecessor of B-Trees
- Main idea: sort tuples on the indexed attribute and create an index file on it
- Similar to a thumb index in a book

Example


Index pages
5
9
…
Sorted →
1
2
3
4
5
6
7
8
…
Page 1
Page 2
Page n
Datapages

Example cont.

- Student with student number 13542 is searched
- During query execution you go through the **index pages** and look for the place where 13542 fits
- From there you get the referenced **data page**
- **Advantage**: Number of index pages is much less than number of data pages, i.e. you save I/O
- You can also answer **range queries**, e.g. all StudNr between 765 and 1232: find as a start the first fitting data page for 765 and from there on you can go **sequentially** through the data pages until StudNr 1232

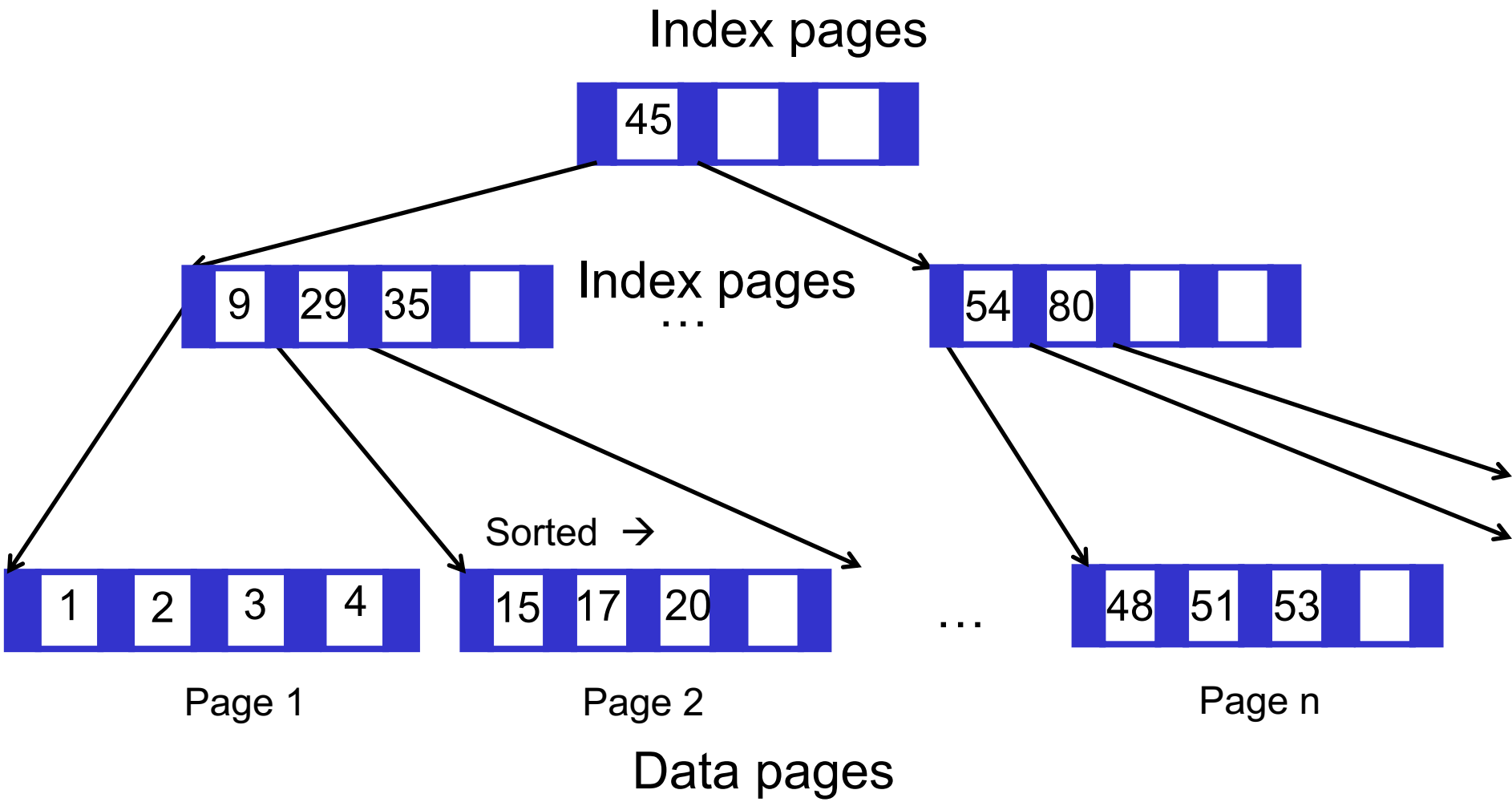
Advancement

Idea:

Why not have **index pages for the index pages**?

→ This is in principle the idea of a **B-Tree**

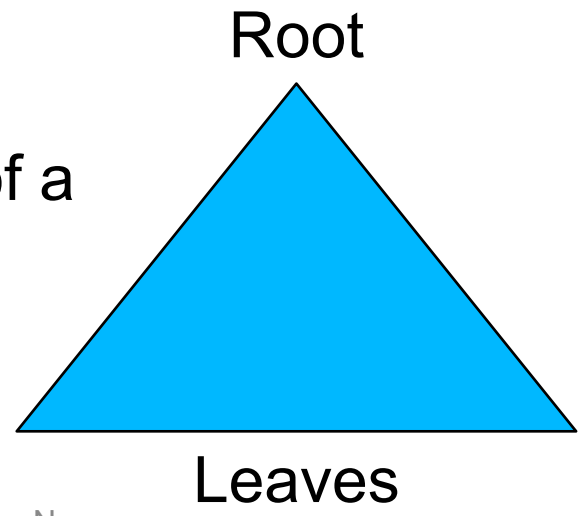
B-Tree


Index pages
45
9
29
35
Index pages
…
54
80
Sorted →
1
2
3
4
15
17
20
…
48
51
53
Page 1
Page 2
Page n
Data pages

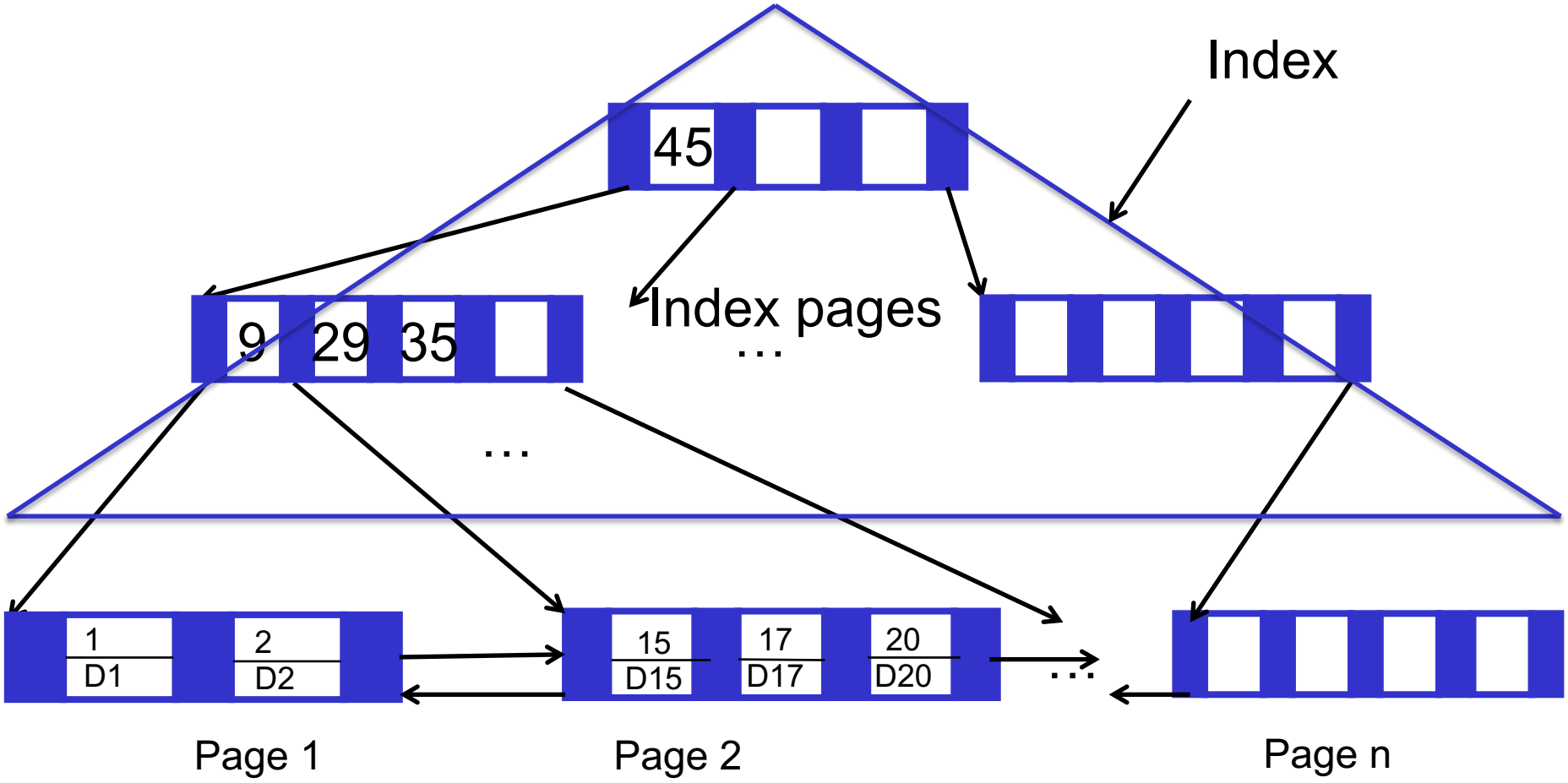
B-Trees

Trees in Informatics
… have nodes
… have edges
… have a root (at the top!)
… have leaves (at the bottom!)
… are often balanced
(otherwise in extreme cases rather a chain)

Schematic depiction of a balanced tree:



Structure B+-Tree



Data pages, sorted, bidirectionally linked

Several Indexes on the same Data

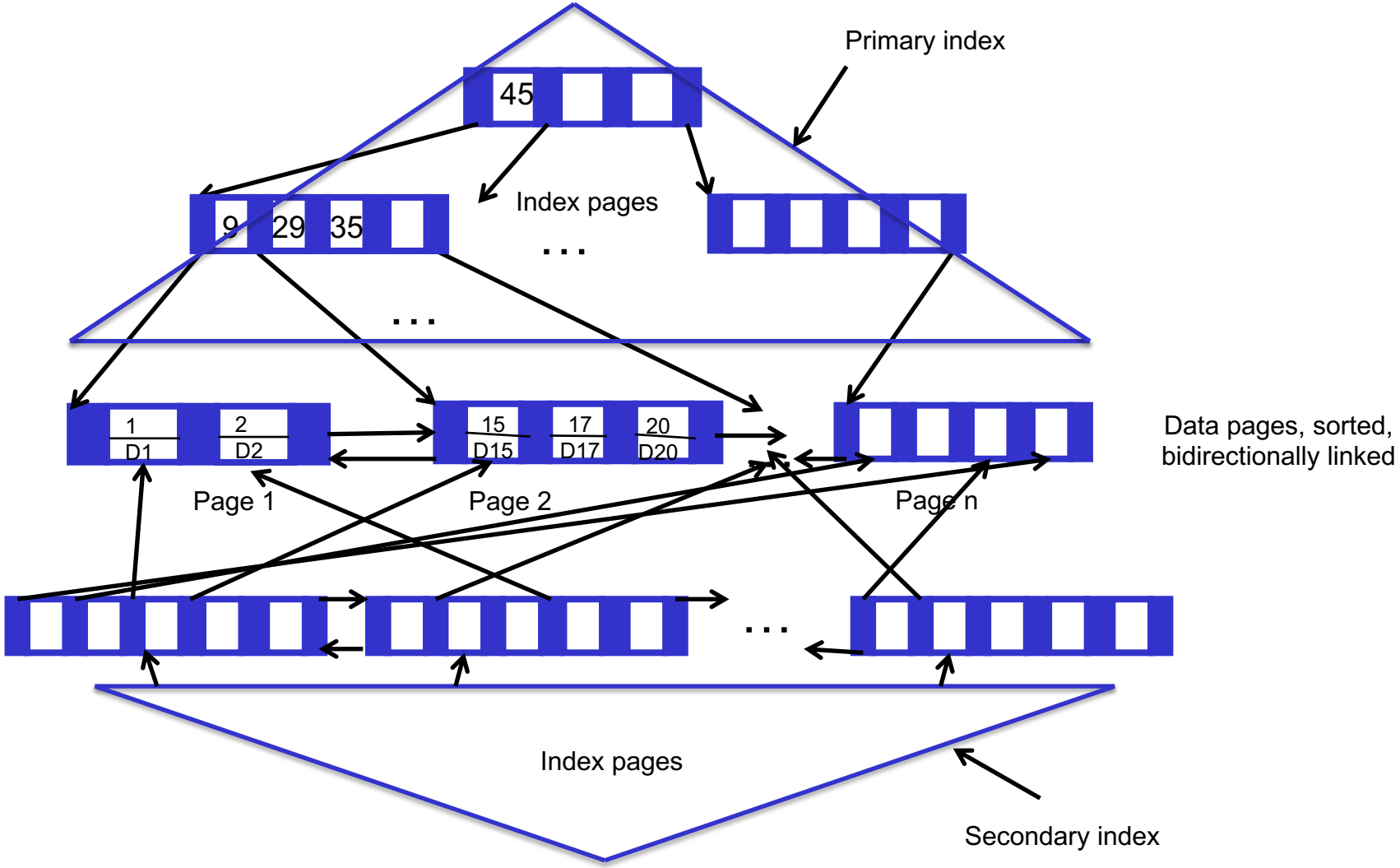
Primary index – Secondary index

Students		
StudNr	Name	Semester
25403	Jonas	12
29120	Theophrastos	2
29555	Feuerbach	2
27550	Schopenhauer	6
⋮	⋮	⋮

When

- Index on StudNr?
- Index on Name?
- Index on Semester?

Secondary indexes



DDL: Create Index

CREATE [UNIQUE] INDEX index_name
ON table_name (column_name1 [, column_name2, …])

Example:

CREATE INDEX full_name
ON Person (Last_Name, First_Name)