CSCE 313-200
Introduction to Computer Systems
Spring 2024

File System V

Dmitri Loguinov
Texas A&M University

April 5, 2024

Chapter 12: Roadmap

12.1 Overview

12.2 File organization

12.3 Directories

12.4 Sharing

12.5 Record blocking

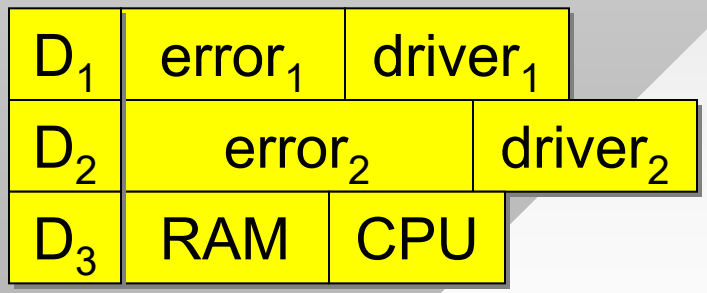
12.6 Secondary storage

12.7 File security

12.8-12.10 Unix, Linux, Windows

File Organization

- As before, a file is just a bunch of bytes
- Our next task is to figure out how to organize these bytes within the file to enable ease of operation
  - Mostly concerned here with data lookup and retrieval
- Assume data is split into items/records
  - Each record has multiple fields (e.g., name, age, SSN)
- 1) Pile is the most general
  - Records dumped into file as they become available to the program, in no particular order, \n separator
  - Different records may have different length or # of fields, typically read by humans
  - e.g., Unix syslog file into which all kernel modules write



File Organization

- 2) Sequential file (sorted or unsorted)
  - One field in each record is the key, everything else is value
  - Search for a given key or range

SSN_1	$salary_1$	age_1
SSN_2	$salary_2$	age_2

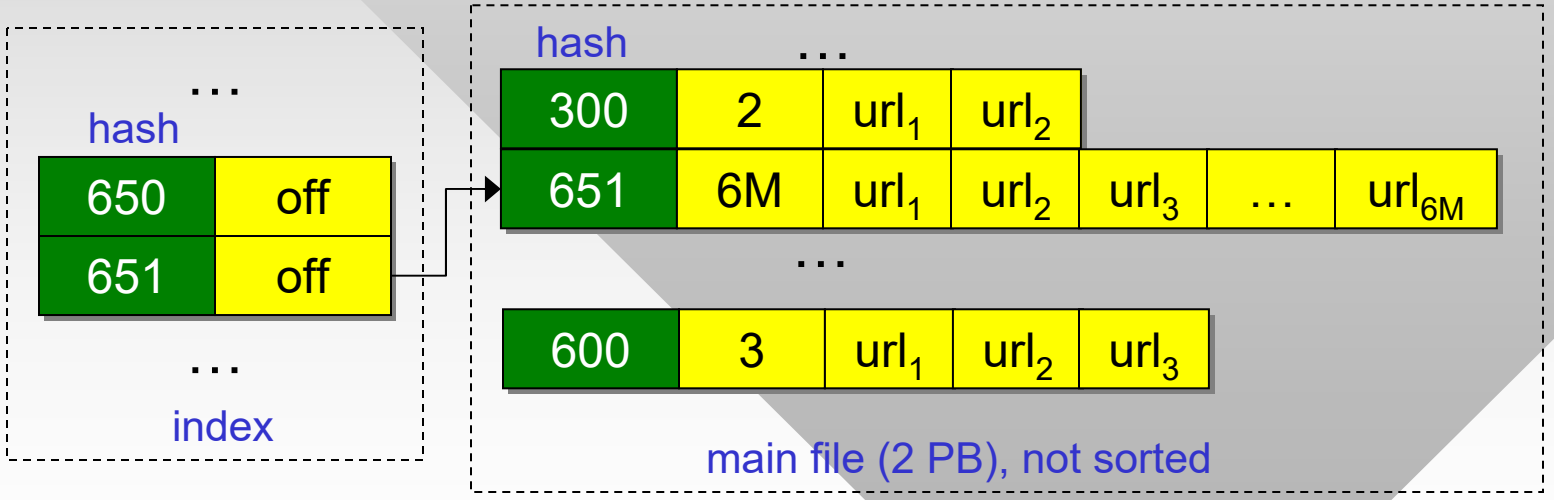
- Fixed-size fields
  - E.g., payroll database with all fields padded to same size
- Variable-size fields
  - E.g., graph (key = nodeID, value = degree + adjacency list)

$node_1$	deg_1	$list_1$
$node_2$	deg_2	$list_2$

- If sorted by key
  - If fixed-size values, binary search to find records
  - If variable-size, need unambiguous record separators
  - Painful to add elements as resorting the file is expensive

File Organization

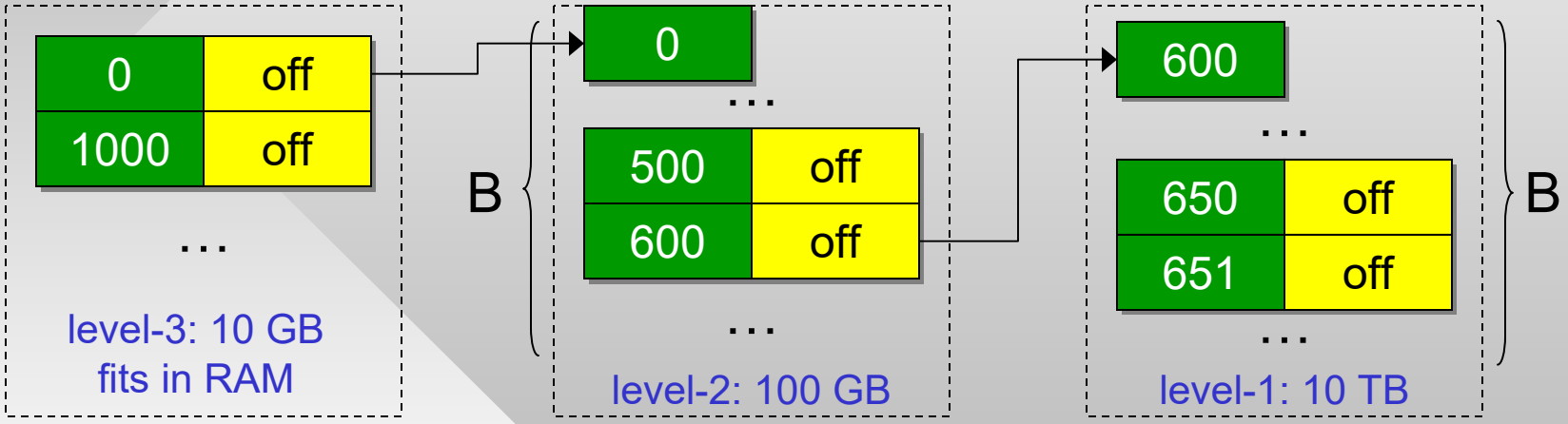
- 3) Indexed Sequential
  - File structure that has the main file with data (usually huge) and a separate file containing the index for keys
- Suppose the main file is Google's word→URL mapping
  - Query maps hashes of words to pages with them



- Binary search on the index, find offset in main file

File Organization

- If index is too big to fit in RAM and binary search is inefficient, a k-level index is possible



- Assume level-1 index size F, read I/O block size B
  - Binary search needs $\log_2(F/B)$ seeks
  - On the other hand, k-level index needs k-1 seeks
- F = 10 TB file, B = 1 MB block size $\rightarrow$ 23 seeks, while multi-index above does it in k-1 = 2 seeks

File Organization

- 4) Indexed
  - Separate index for every possible field, allows database-like operations on fields
- Main challenge for indexed files is keeping the index updated when it doesn't fit in RAM
- 5) Hashed file
  - Treat file contents as RAM, hash items directly to some offset

```
uint64 N;           // hash table size
// preallocate file of size N * sizeof(item)
void Hash (Item x) {
    off = HashFunction (x.key) % N;
    file.Seek (off * sizeof(Item));
    file.Write (&x, sizeof(Item));
}
```

- What to do with collisions?