Secondary Storage Management

Outline

- Memory Hierarchy
- Disk, and Disk Access Time
- Storing (Relational) Data on Disks
- Pointer Swizzling
- Variable-Length or Large Records/Fields
- Deletions and Insertions of Records

Memory Hierarchy

Tertiary Storage (PetaBs; secs-minutes)

Non-Volatile

Secondary Storage/Disks (TBs; 10 msec)
(Virtual Memory; File System)

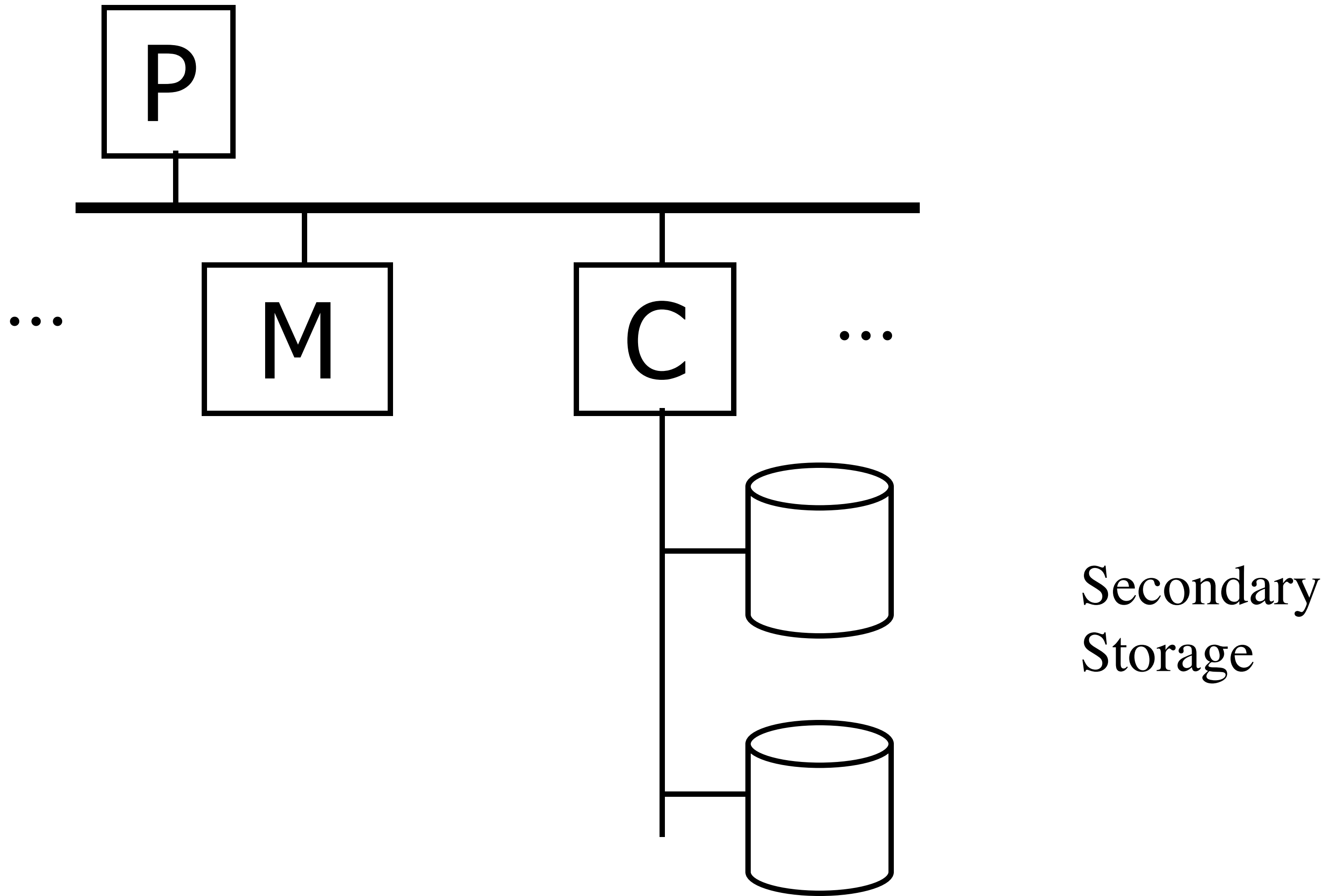
Blocks

Main Memory (GBs; 10-100 nsec)

Volatile

Cache (1 MB; 1-5 nsec)

Disk Controllers

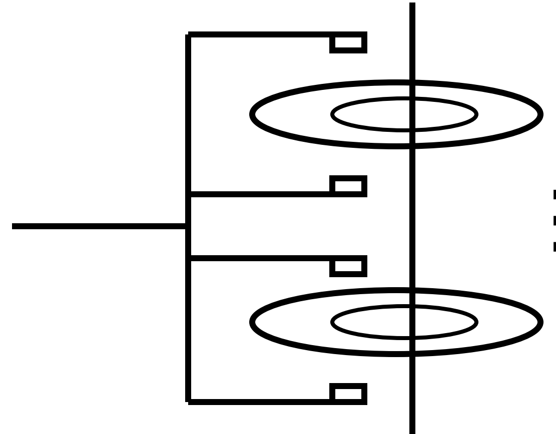


Secondary
Storage

Outline

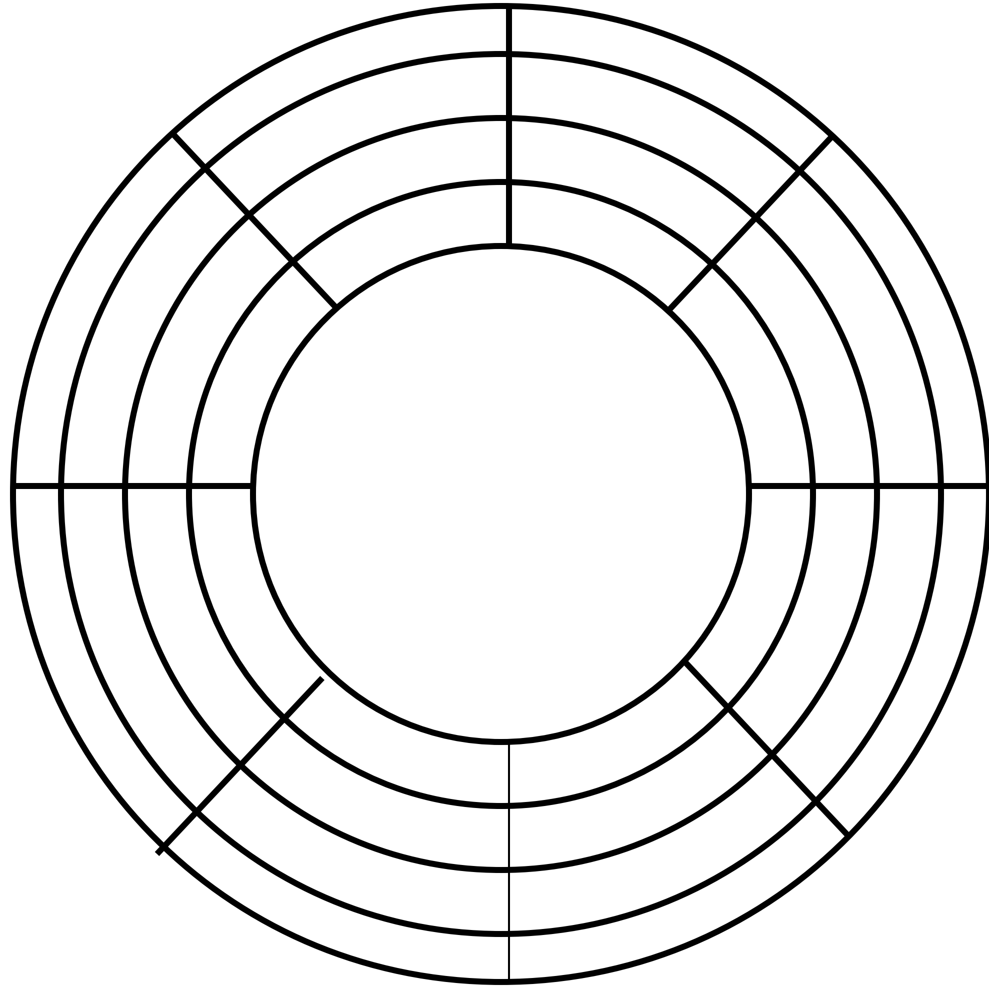
- Memory Hierarchy
- Disk, and Disk Access Time
- Storing (Relational) Data on Disks
- Pointer Swizzling
- Variable-Length or Large Records/Fields
- Deletions and Insertions of Records

Secondary Storage – Disks

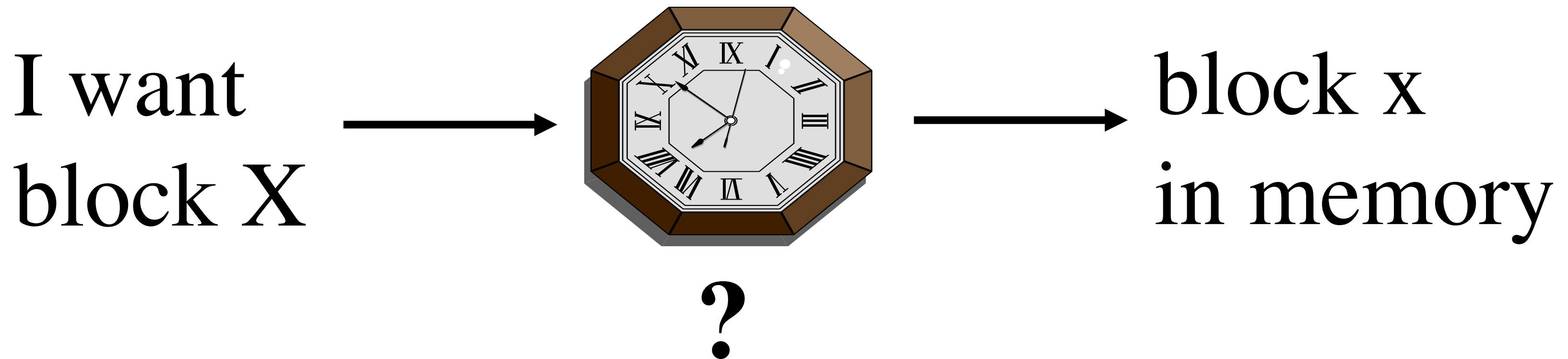


Terms: Head, Platter/Surface,
Cylinder, Track, Sector,
Gap (to delineate sectors)

Top View



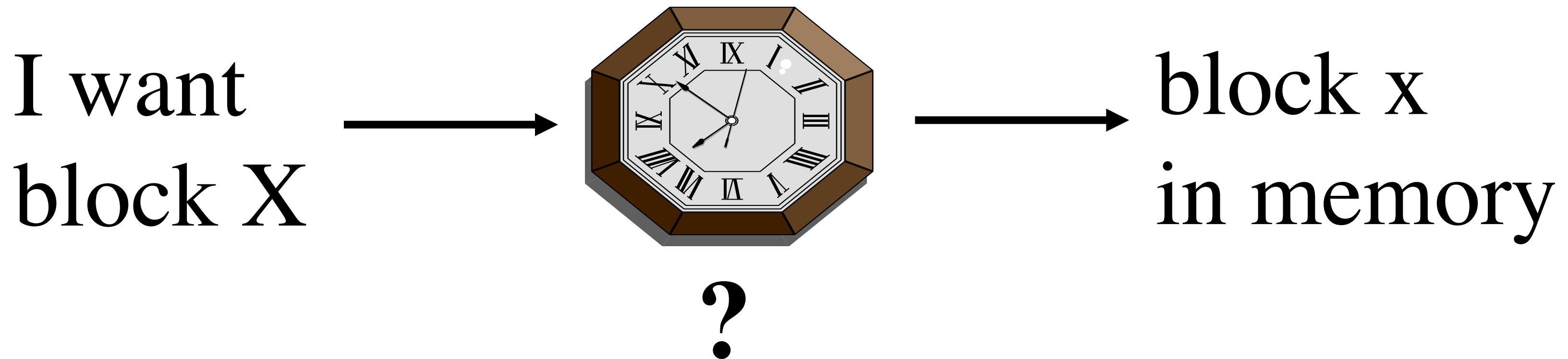
Disk Access Time



Access time for a random block

= Seek Time +
Rotational Delay +
Transfer Time +
Other (contention, CPU time; negligible in comparison)

Disk Access Time



Access time for the next block

= Transfer Time + Negligible

- skip gap
- switch track
- once in a while,
  next cylinder

Rule of Thumb	Random I/O: Expensive Sequential I/O: Much less

- Writing: Similar cost as reading
- Update:
  - (a) Read Block
  - (b) Modify in Memory
  - (c) Write Block

Accelerating Access to Disk Storage

- I/O Cost is dominant (compared to main-memory computation)

Techniques to improve I/O cost:

- Organize data by cylinders
- Using multiple disks (in parallel)
- Elevator algorithm (to serve data requests)
- Pre-fetching, Large-Scale Buffering

Outline

- Memory Hierarchy
- Disk, and Disk Access Time
- Storing (Relational) Data on Disks
- Pointer Swizzling
- Variable-Length or Large Records/Fields
- Deletions and Insertions of Records

Storing Relations on Disks

- To store: Attributes, records, relations
- Physical Data Hierarchy
  - Attrs → Records → Blocks → Relations
- Block is the unit of I/O transfer.
- Database relations map to one or more blocks.

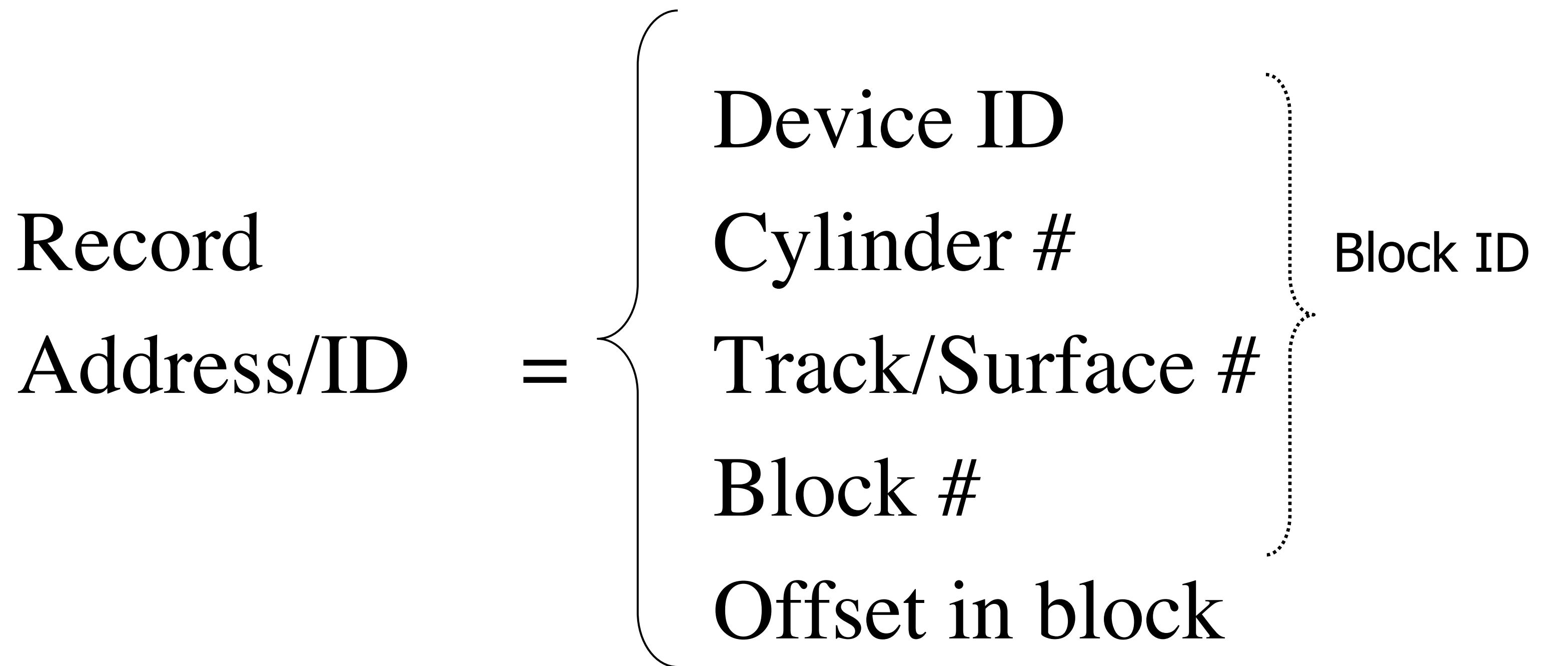
Fixed-Length Records

- Records with fixed-length fields:
  - [Record header, <attr1>, <attr2>, ….]
  - Record header = (schema, length, timestamp). Schema (type and bytes for each attribute) lets us access the attributes. Can be substituted by attribute offsets/pointers.
- Packing records into block
  - [Block header, <rec1>, <rec2>, …]
  - Block header = (pointers to other blocks, schema, record offsets, timestamps, other info)

Record Addressing

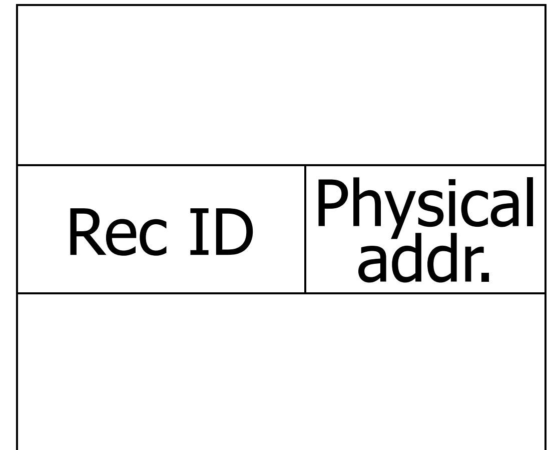
- How does one refer to records?
  - Physical/Direct Addressing
  - Logical/Indirect Addressing
  - Mixed/Structure Approach

Purely Physical Address



Logical/Indirect Addressing

E.g., Record ID is arbitrary bit string

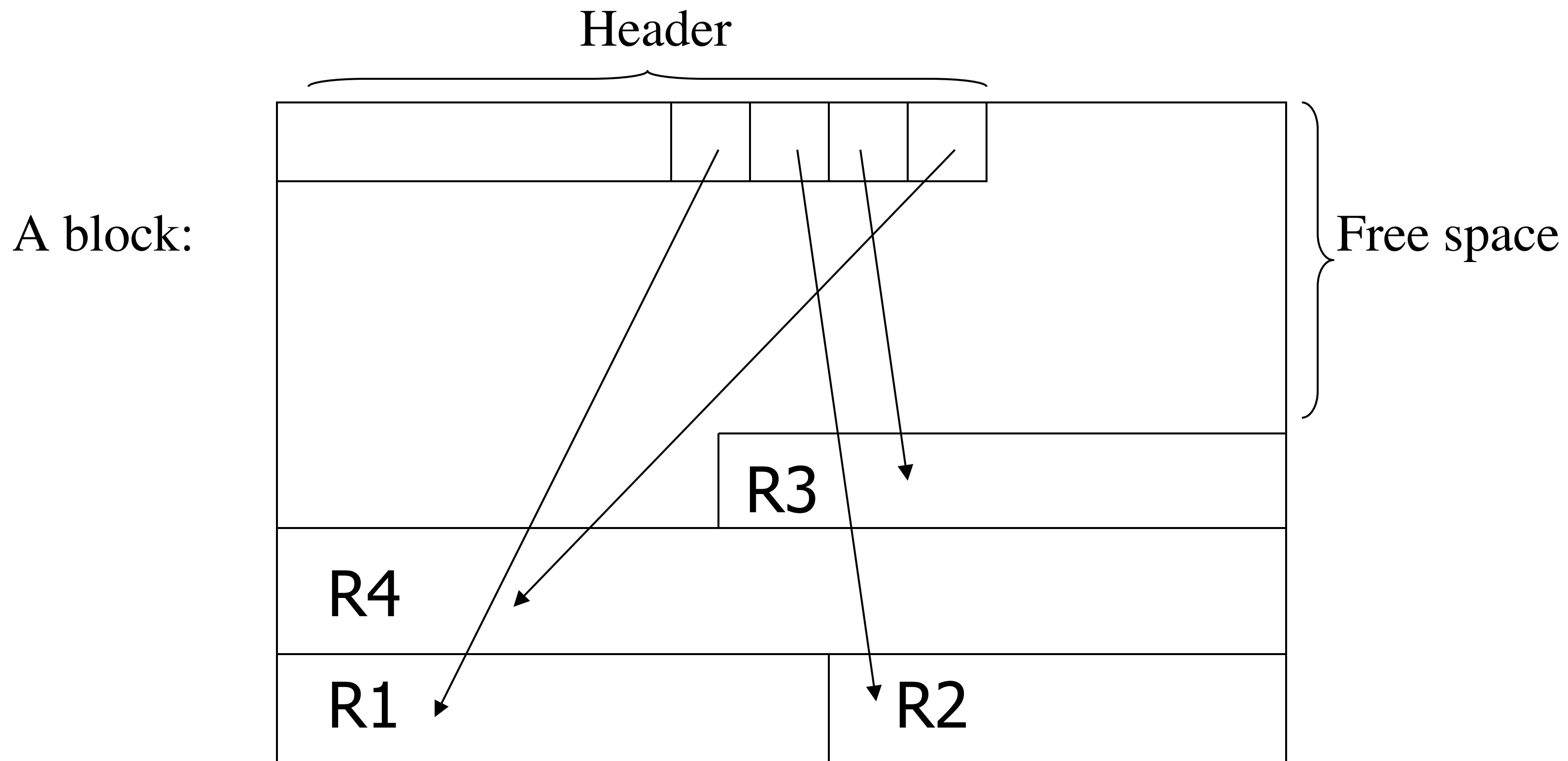


Advantages:

Records are "mobile," deletion doesn't give "dangling" pointers.

Structured/Mixed Approach

Record Address: (Physical Block Address, Record-ID in block)



Outline

- Memory Hierarchy
- Disk, and Disk Access Time
- Storing (Relational) Data on Disks
- Pointer Swizzling
- Variable-Length or Large Records/Fields
- Deletions and Insertions of Records

Pointer Swizzling

- Consider a block B read into the memory. What happens to the pointers to B? Note that these pointers are disk-addresses.
- Option 1: Create a "translation table" in memory, that maps disk-addresses to memory-addresses (for blocks currently in memory).
- Option 2: Change the disk-addresses to memory-addresses (in other blocks in memory containing pointers to B). This is called pointer-swizzling (still need the "translation table").

Pointer Swizzling: Options and Issues

1. Automatic: Whenever a block is read: update the table, and swizzle all relevant pointers.
2. On demand: Swizzle only after the first try (table is still updated, when a block is read).

- Unswizzling: when blocks are written back
- “Pinned” blocks (due to recovery system, or sizzling mechanism). To unpin, we first need to unswizzle pointers to it.

Outline

- Memory Hierarchy
- Disk, and Disk Access Time
- Storing (Relational) Data on Disks
- Pointer Swizzling
- Variable-Length or Large Records/Fields
- Deletions and Insertions of Records

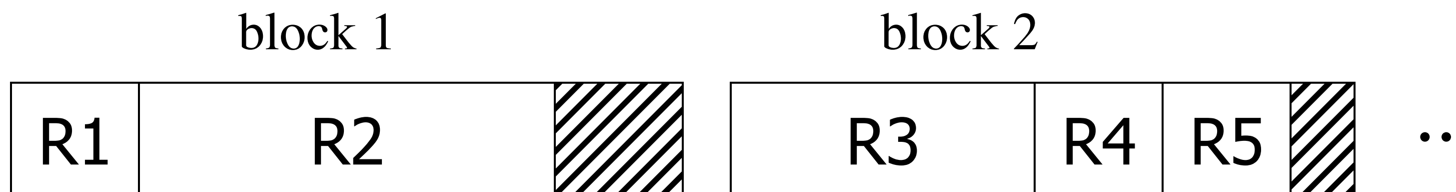
Variable-Length/Format Fields/Records

- Keep appropriate information in the record header:
  - Pointers to attribute positions. Works for variable-length fields.
  - Number/size of elements. Works for attributes that are list of values.
- "Tagged fields": Indicate type, length, etc. at the start of the attribute. Works for variable-format records.

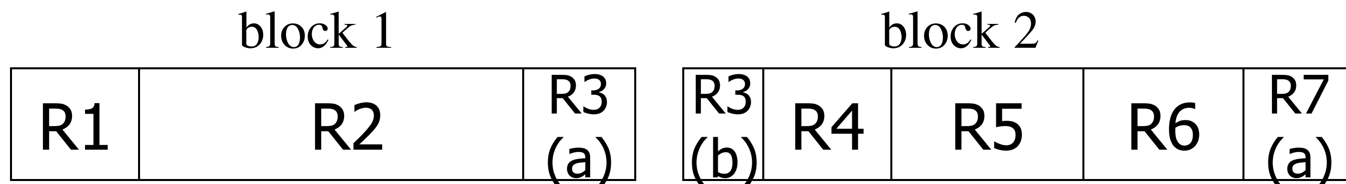
Large Records

- Some records may not fit in a single block
- Need to be "spanned" across blocks

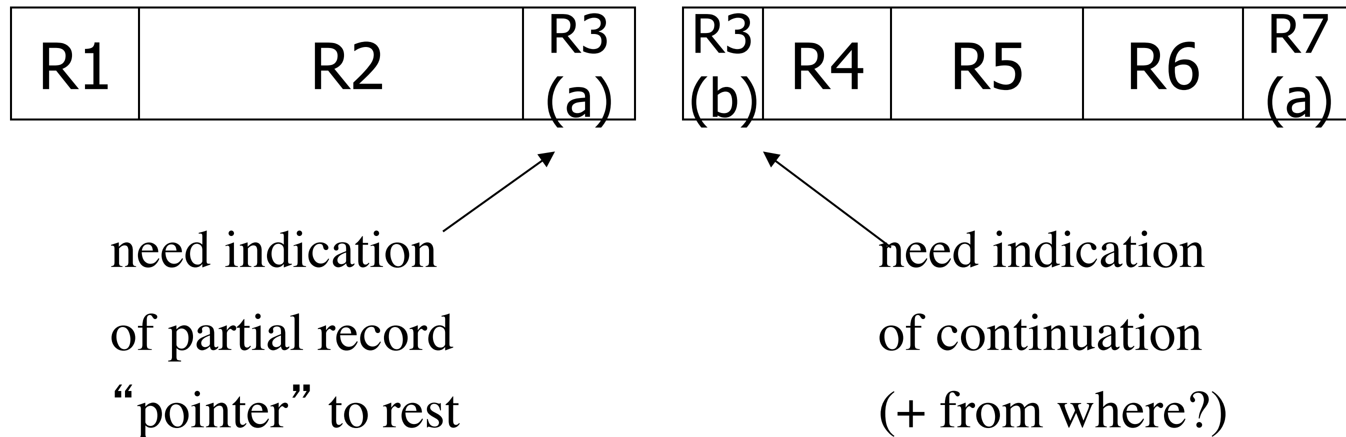
Unspanned: records must be within one block



Spanned



With spanned records:



- Spanned essential if record size > block size
- Unspanned is simpler, but may waste space

Very Large Fields (BLOBs)

- Some fields may be very large (e.g., videos or pictures). May make sense to store them:
  - Separate from the rest of the record,
  - Across disks for efficient parallelized access.
  - Indexed, so that any portion of it can be retrieved (e.g., 45-50 mins of the movie).

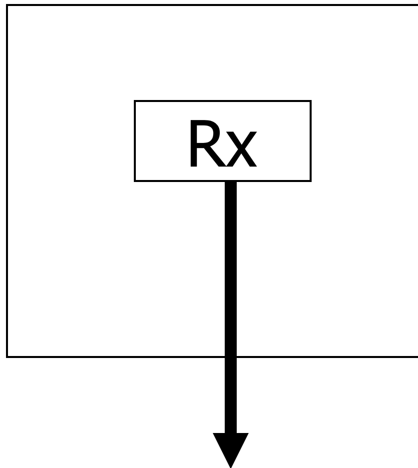
Other Storage Options

- Split Records: Rather than storing rows, we can store columns!
  - Will require keeping the "ID" with each attribute.
  - Could be helpful, if we need some statistics for an attribute.
- Ordering the records:
  - Sorted by some attribute value.
  - Physical or logical (linked list)

Outline

- Memory Hierarchy
- Disk, and Disk Access Time
- Storing (Relational) Data on Disks
- Pointer Swizzling
- Variable-Length or Large Records/Fields
- Deletions and Insertions of Records

Record Deletion

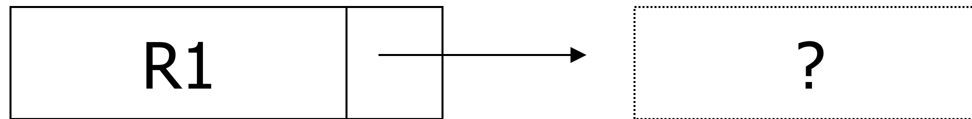


(a) Immediately reclaim space

(b) Dangling pointers? Next.

Concern with deletions

Dangling pointers

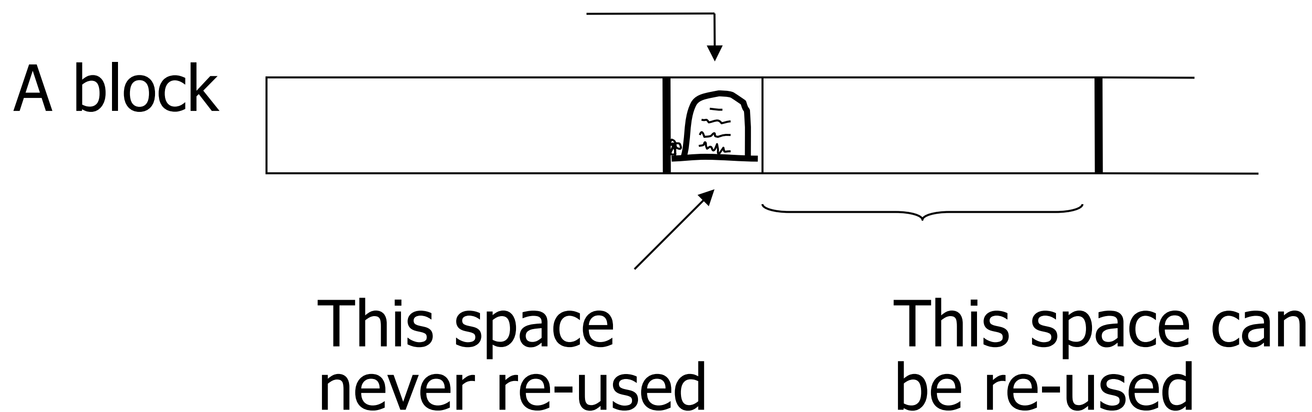


Solutions

1. Do not worry
2. Use "Tombstone"

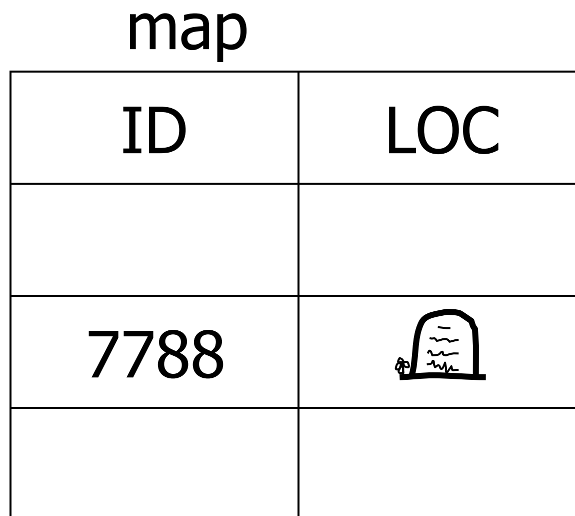
Tombstones for Physical IDs

E.g., Leave "MARK" in old location



Tombstones for Logical IDs

E.g., Leave “MARK” in the map



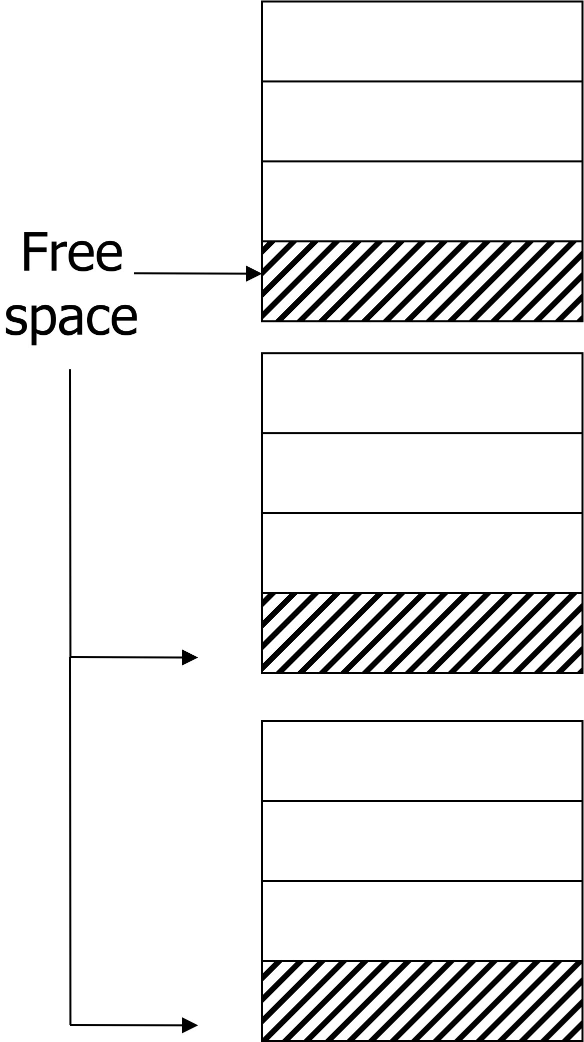
map

ID	LOC
7788	

Never reuse ID 7788 nor space in map...

Insertions

- If records NOT in sequence (easy case)
  - Insert at end of file or in deleted slot
  - If records are variable size, not as easy...
- If records in sequence
  - If free space "close by", not too bad...
  - Or use overflow idea...


Free
space

Given a key, how to find a record quickly