Goals

- Sequential access:
  - Enumerate (some columns of) all rows
- Random access:
  - Retrieve a previously visited row
  - Assumes a "row reference" data structure
- Insert/Update/Delete

Key Issue:
How much data has to be moved for each operation

Naive row layout: Fixed length records

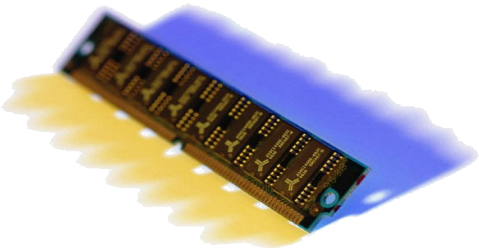
- Sequential access:
  - Offset += len
- Random access:
  - Reference = offset
- Insert:
  - Append at the end
- Update:
  - In place
- Delete?

Challenges

- What if data does not fit in memory?
- How to make the representation more compact?
  - Variable sized columns
  - Null values
- How to change data?
  - Delete rows
  - Change values

Disk blocks

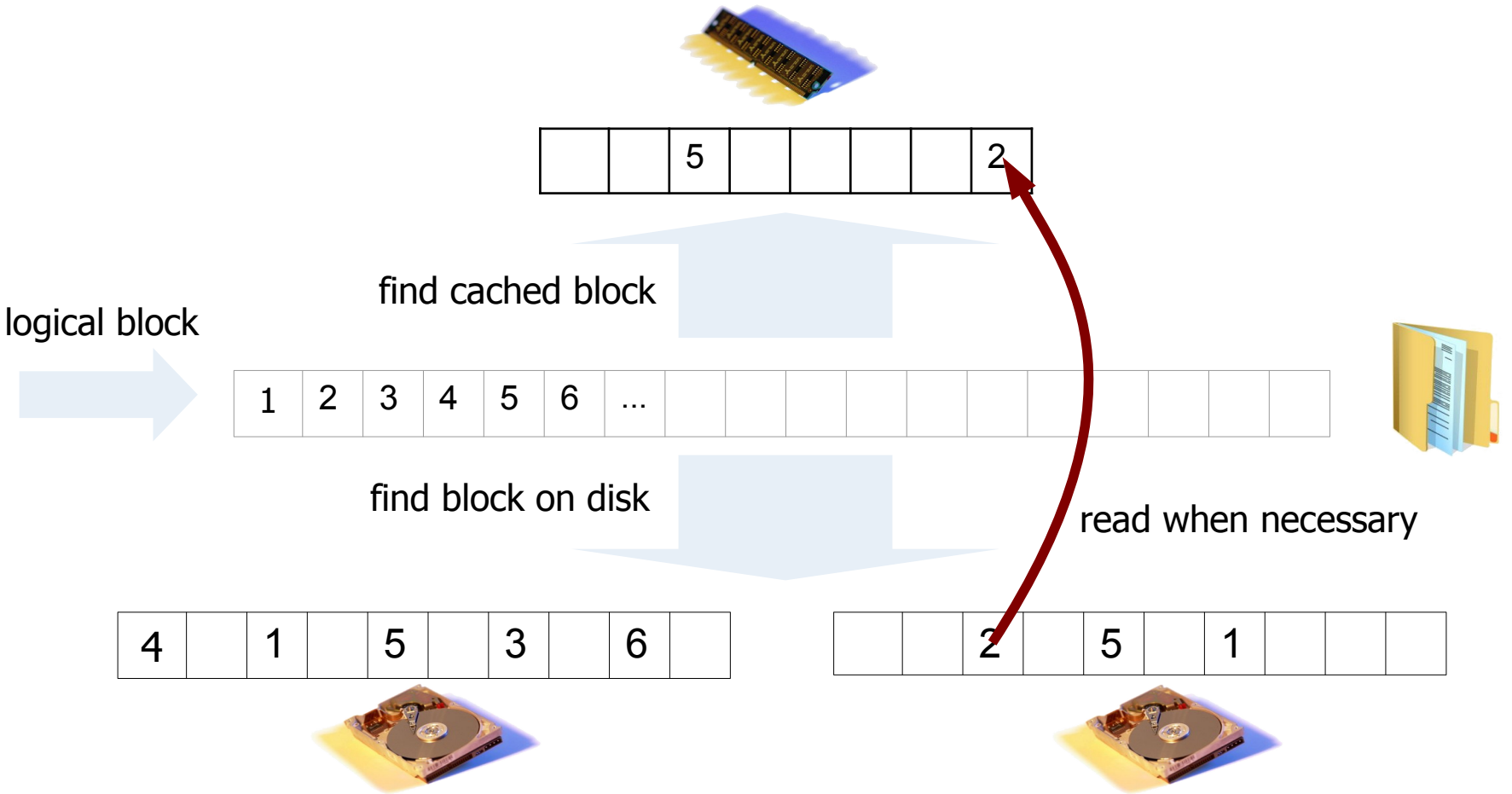
- Consequences of block I/O:
  - Cost of 1 byte = cost of 1 block
  - Alignment
- Random access vs sequential access
- Still partially true with SSD...



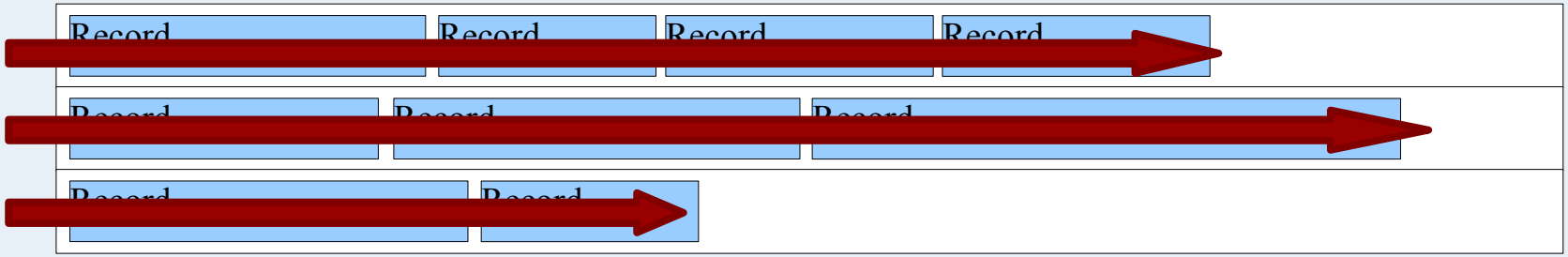
Roadmap

- Three layers:
  - Blocks in disks or memory
  - Records in blocks
  - Fields in records
- Case study
- Visible consequences

Blocks in disk and in memory

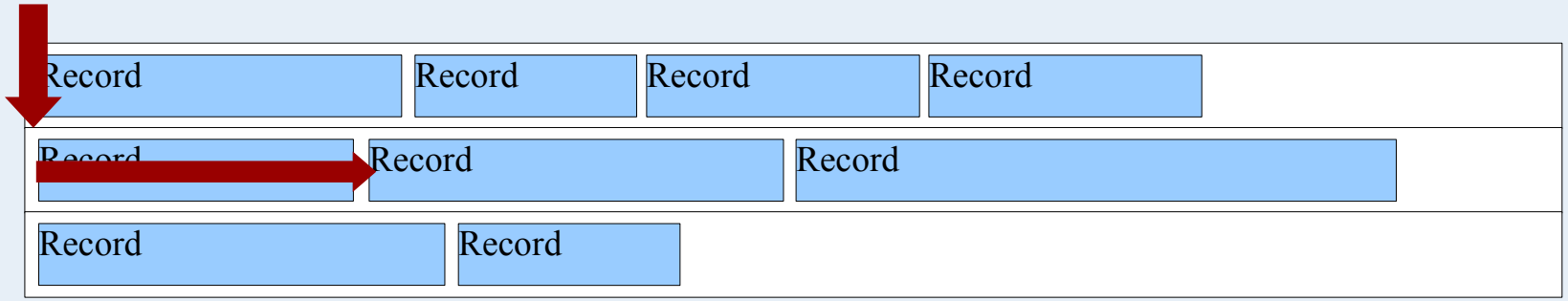


Row layout: Records in blocks



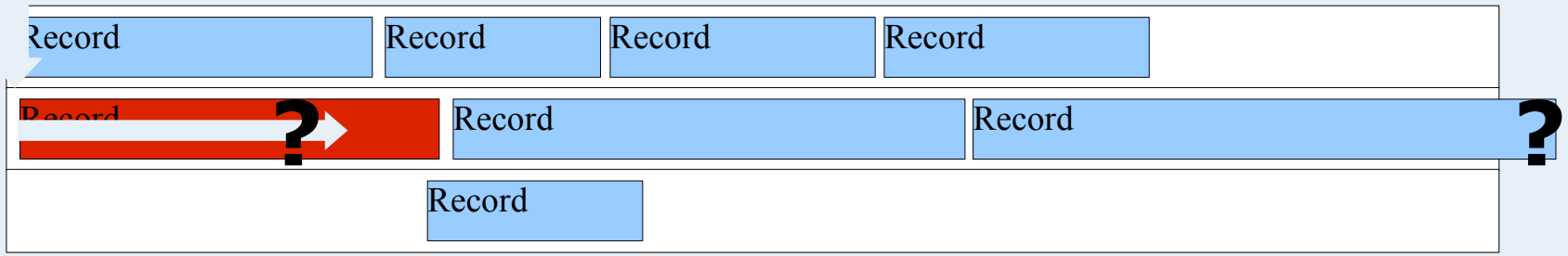
- Sequential:
  - Enumerate blocks
  - Enumerate records in the block

Row layout: Records in blocks



- Random access:
  - (block offset , record offset)
- Insert:
  - append to some block

Row layout: Records in blocks



- Breaks when a record grows:
  - Pushes others forward
  - A block fills up
- Inefficient when a record shrinks / is deleted
  - Fragmentation

Records in blocks

- Reference:
  - (Block offset , Record index)
- Two stacks:
  - Offset table
  - Records

Records in blocks

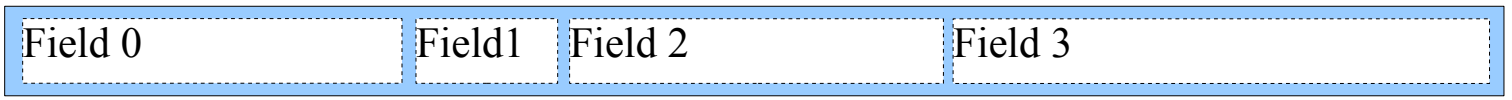
- Space occupied by deleted records is reclaimed
  - No fragmentation
- Records grow without impacting references
- Records can be migrated by leaving the forwarding address

Fields

- Efficient direct access:
  - e.g. for "select column3 from table"
- Reduce space used
- Can represent null values

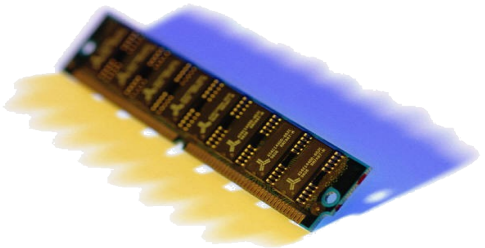
Fields in records

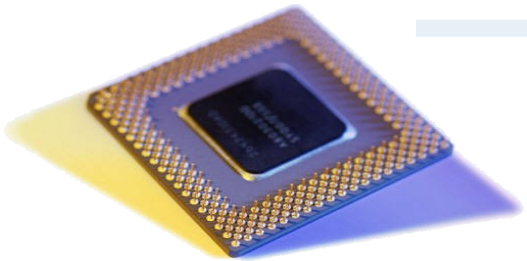
- Packed tightly:
  - Access needs iteration
  - Nulls? Zero size not enough

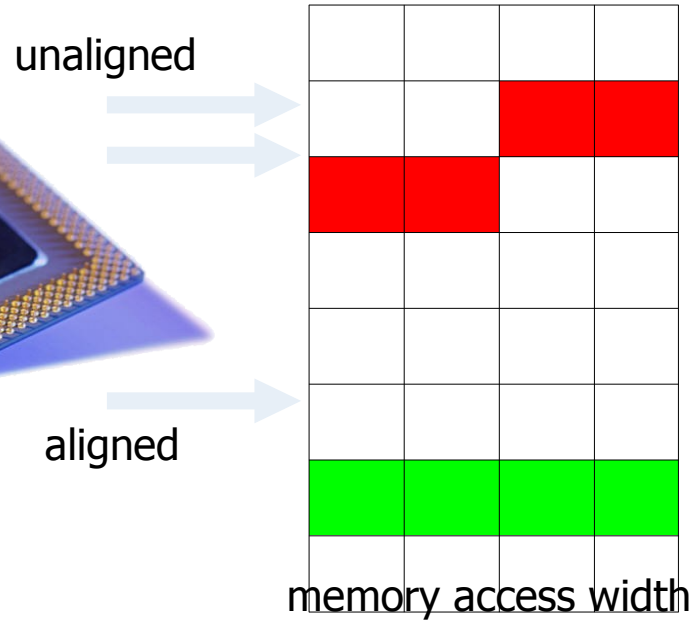


Alignment

- Memory is addressed at byte offsets
- But accessed at multi-byte offsets
- Unaligned accesses are:
  - Costly
  - Disallowed

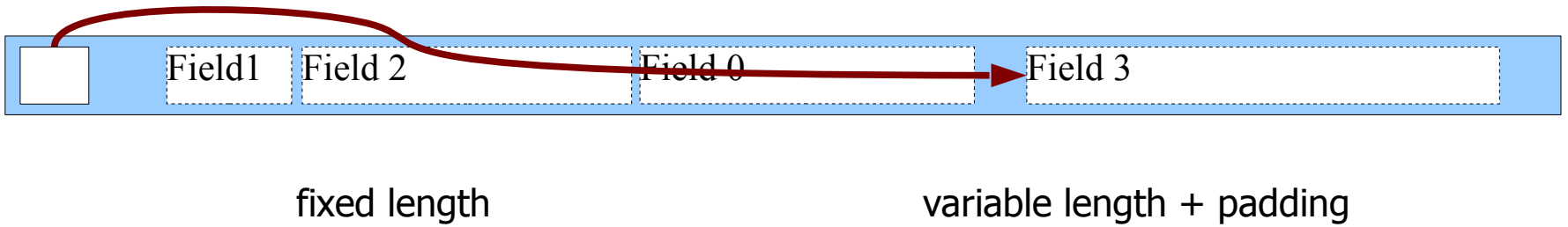




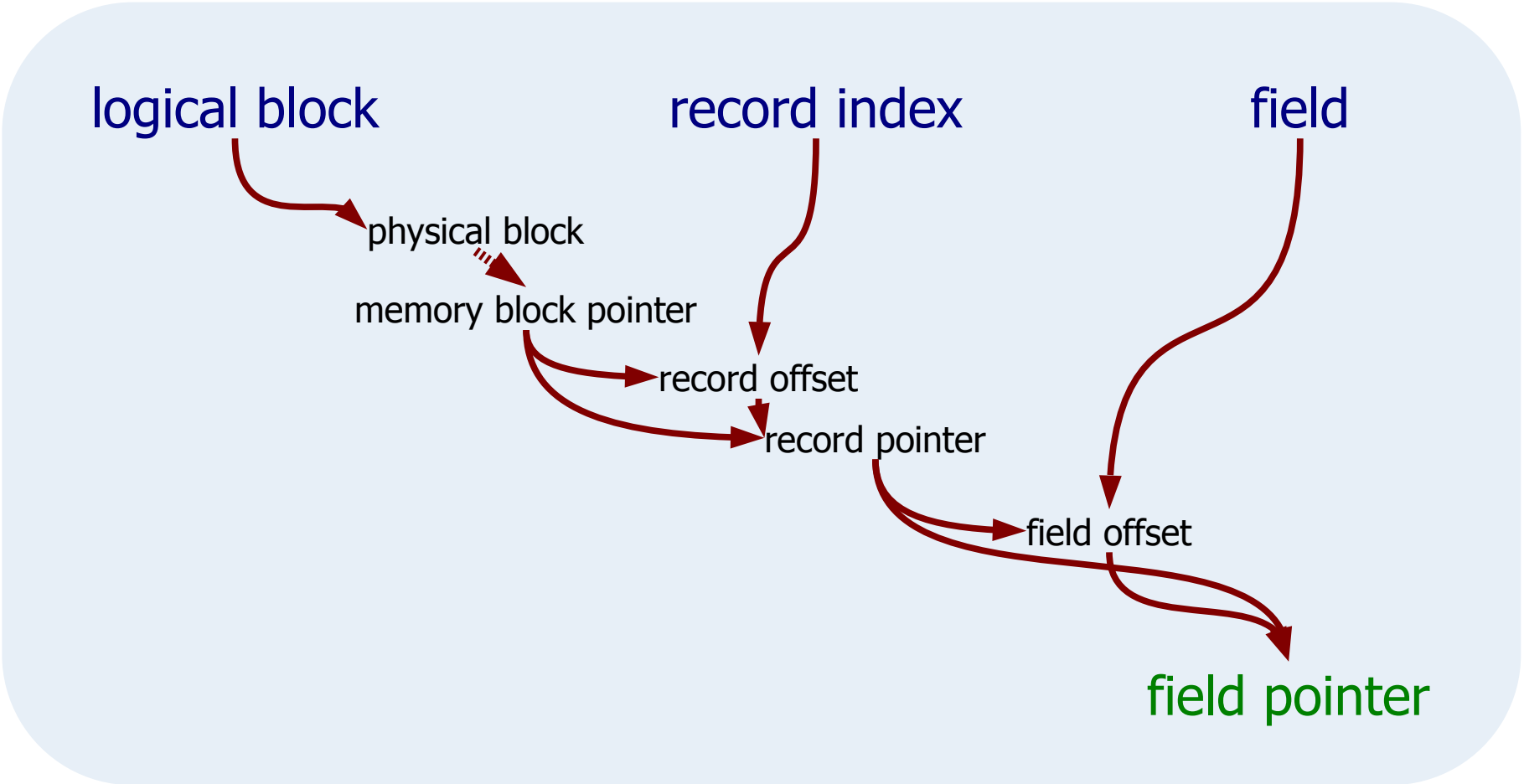


Fields in records

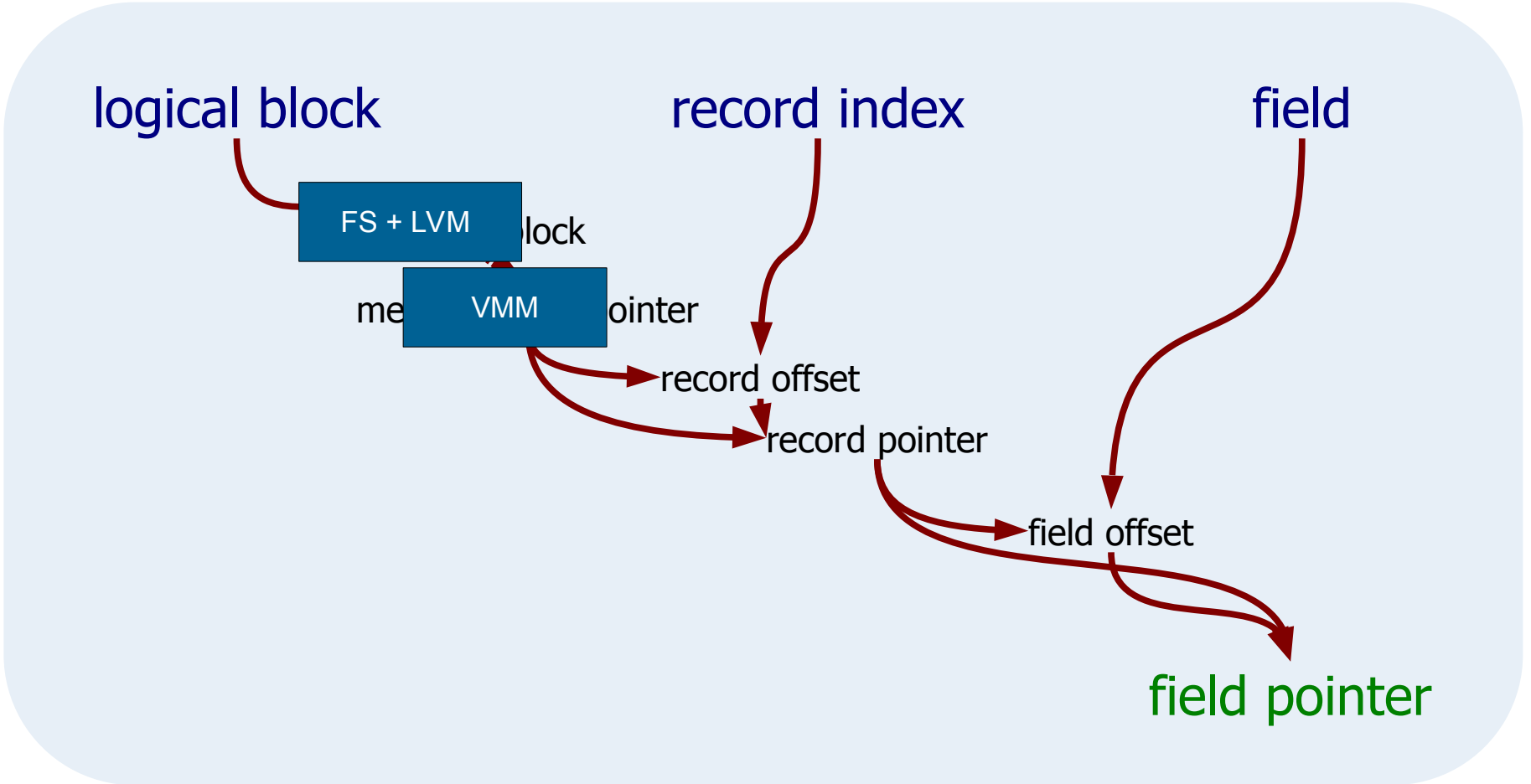
- Pointers at the start of the records:
  - All fixed size fields first
  - Pointers to variable sized fields
- Bitmap of nullable fixed fields



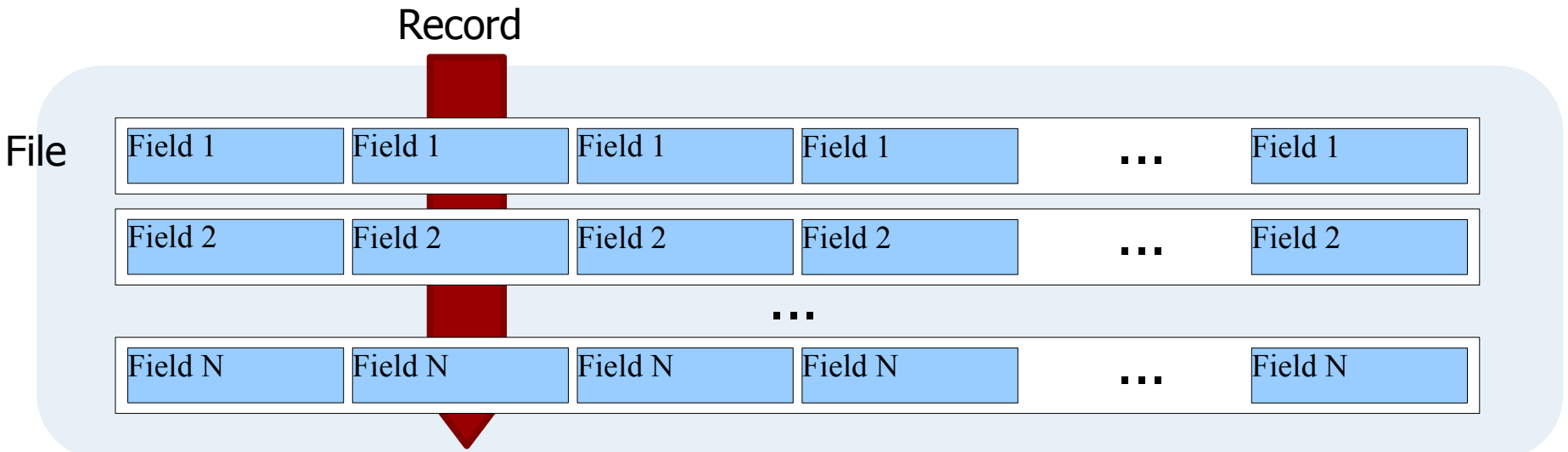
The path to a data item



Shortening the path



Columnar layout

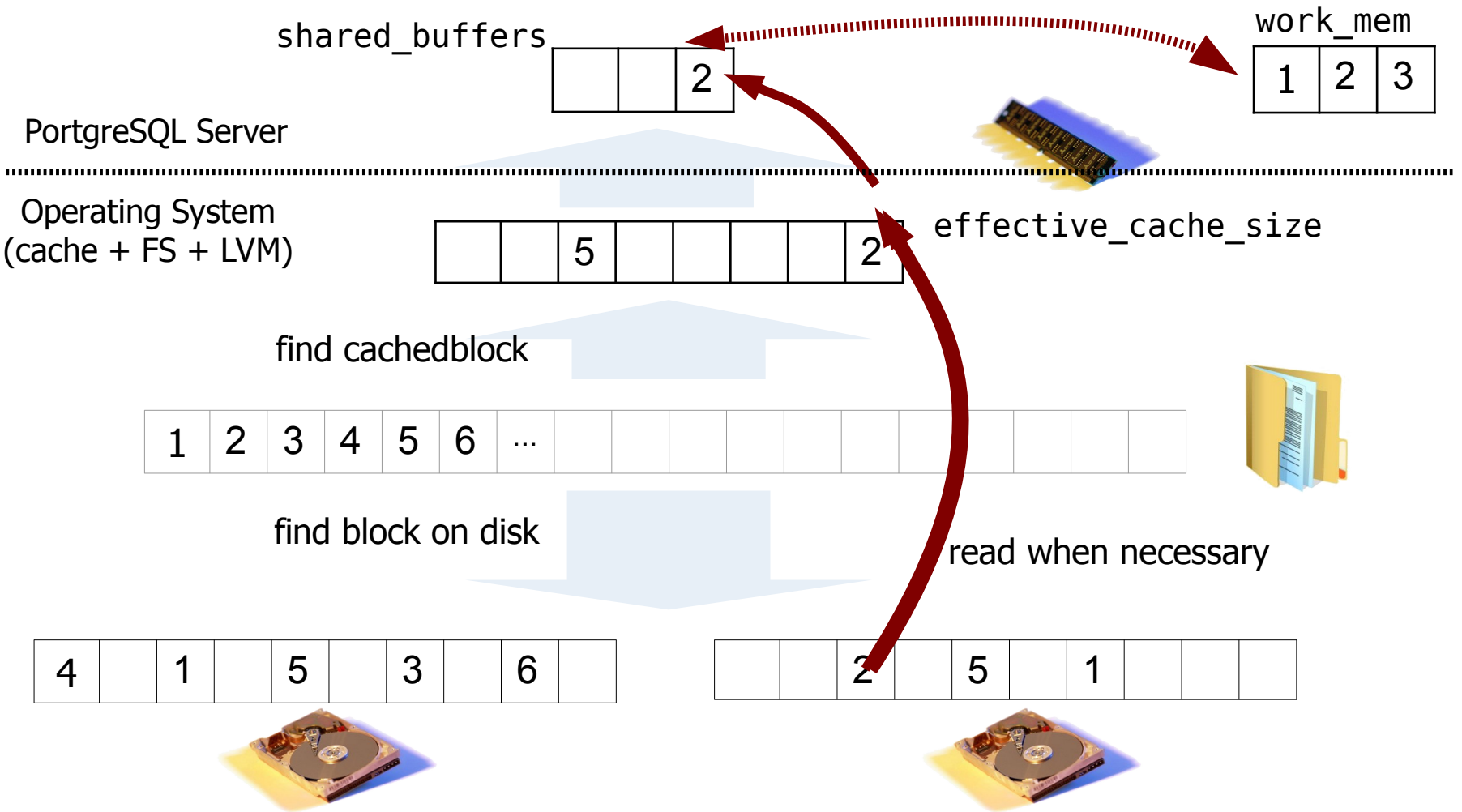


- Each file holds a column (i.e., the same field for all rows)
- Efficient sequential scan
  - Minimizes data transfer, better compression
- Efficient insert (append data)

Blocks in PostgreSQL

- Logical to physical translation performed by the operating system
- Blocks in memory:
  - Part is managed explicitly (shared buffer cache)
  - Relies on operating system cache

Blocks in PostgreSQL



Records in PostgreSQL

- Mostly standard, except...
- Deleted records:
  - Not immediately reclaimed
  - Blocks are periodically “vacuumed”
- Updated records:
  - Records are not updated in place
  - A new version is created
  - Old versions are considered deleted

Will be explained later...

Consequences

- Eliminating duplicates is a costly operation
  - By default SQL deals with bags, not sets
  - Duplicates are allowed on storage and in results
  - Set operations are provided, but costly
    - DISTINCT
    - UNION, INTERSECTION, DIFFERENCE
- Single column scan in wide tables is a costly operation