Database Administration

José Orlando Pereira

Departamento de Informática
Universidade do Minho

Relational systems

- Assume a relational system with the SQL query language
- How is a query executed?
  - Physical data representation
  - Query processing
  - Redundancy: Indexes and views
  - Query optimization

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
  - Compression
- How to change data?
  - Delete rows
  - Change values

Memory hierarchy

execute typical instruction	1/1,000,000,000 sec = 1 nanosec
fetch from L1 cache memory	0.5 nanosec
branch misprediction	5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec

Source: http://norvig.com/21-days.html#answers

Disk blocks

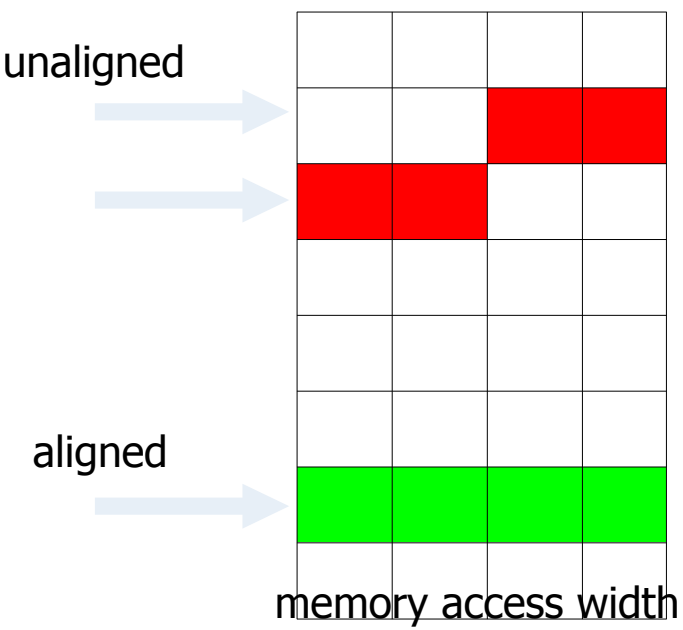
- Typical block sizes: 512B to 4KiB
- Consequences of block I/O:
  - Cost of 1 byte = cost of 1 block
  - Alignment
- Random access vs sequential access
- Still partially true with SSD...

Cache lines

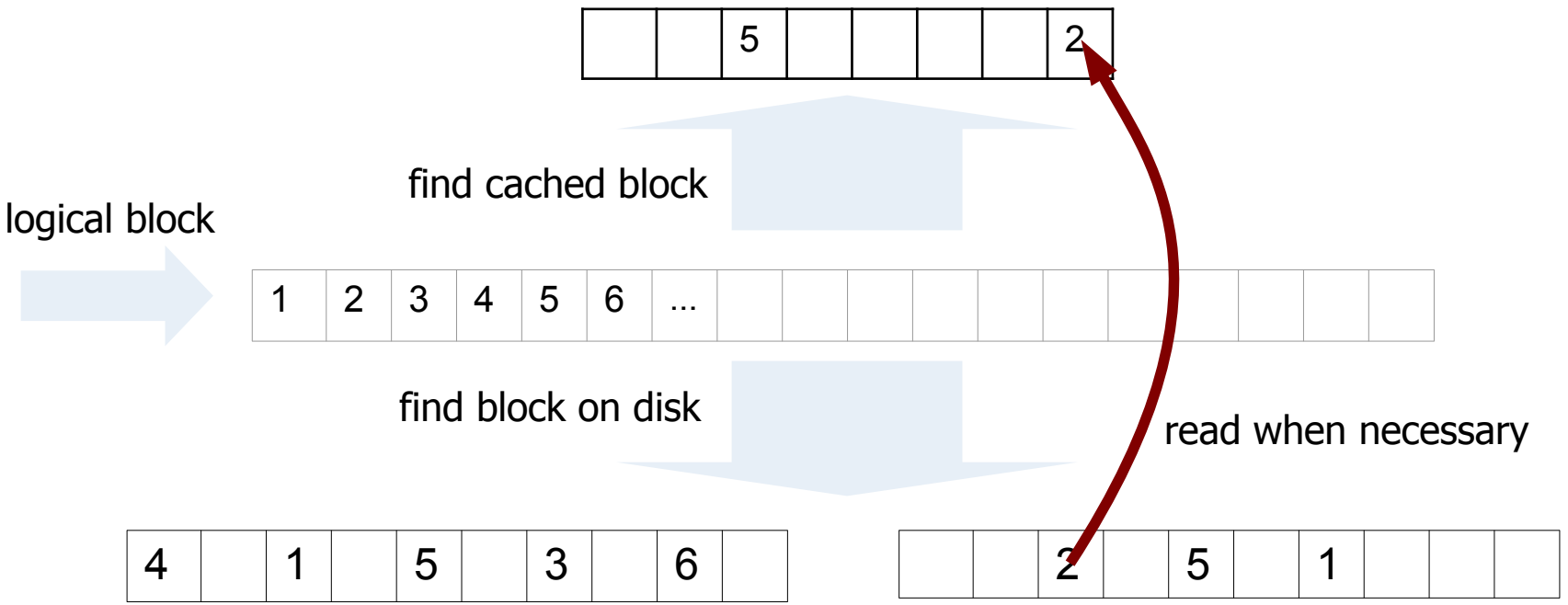
- Typical line size: 64B
- Cache size is very limited
  - Must make use of every byte cached!
- False sharing when updated

Alignment

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



Blocks in disk and in memory



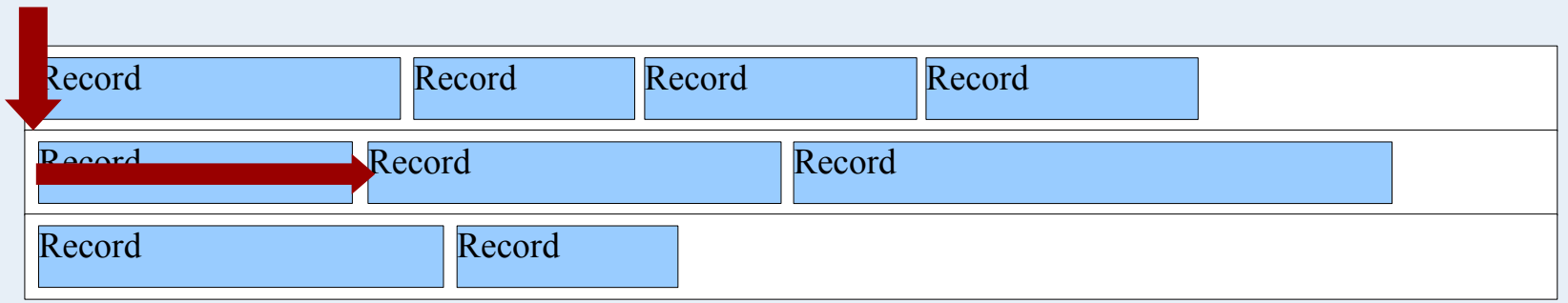
Roadmap

- Layouts:
  - Row oriented
  - Column oriented
- Visible consequences
- Case study: PostgreSQL

Row layout: Records in blocks

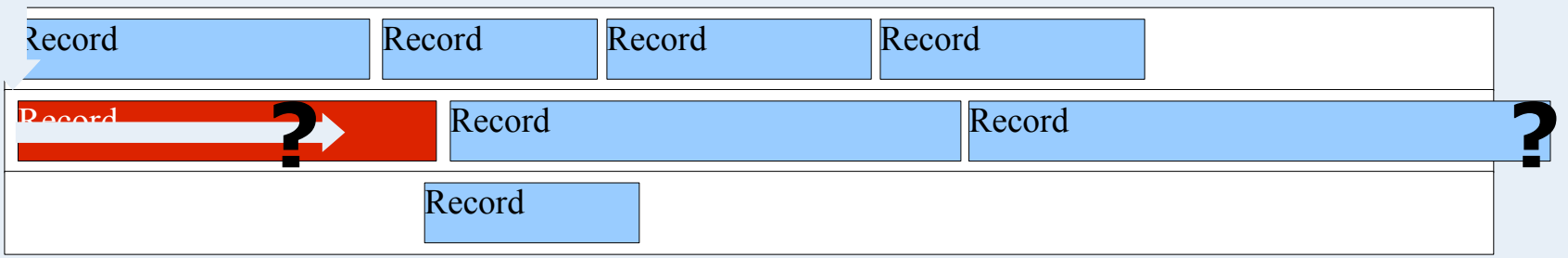
- Sequential:
  - Enumerate blocks
  - Enumerate records in the block

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

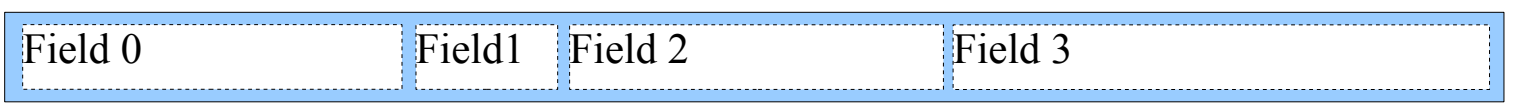
- Space occupied by deleted records is reclaimed
  - No fragmentation
- Records grow without impacting references
- Records can be migrated by leaving the forwarding address

Fields

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

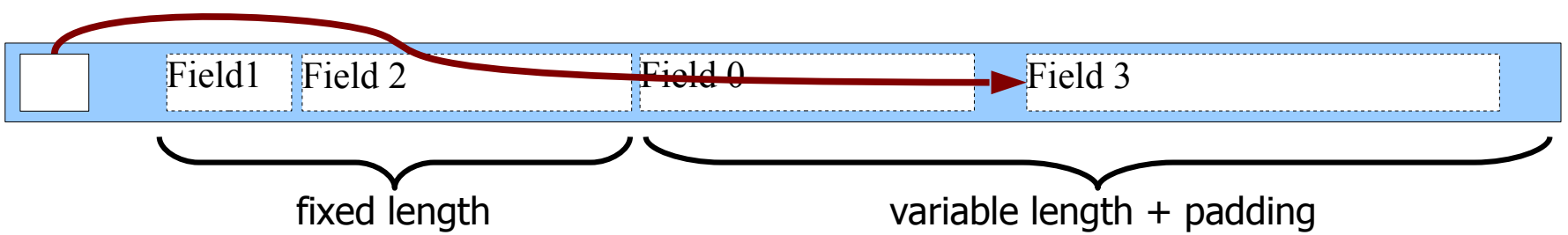
Fields in records

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



Fields in records

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

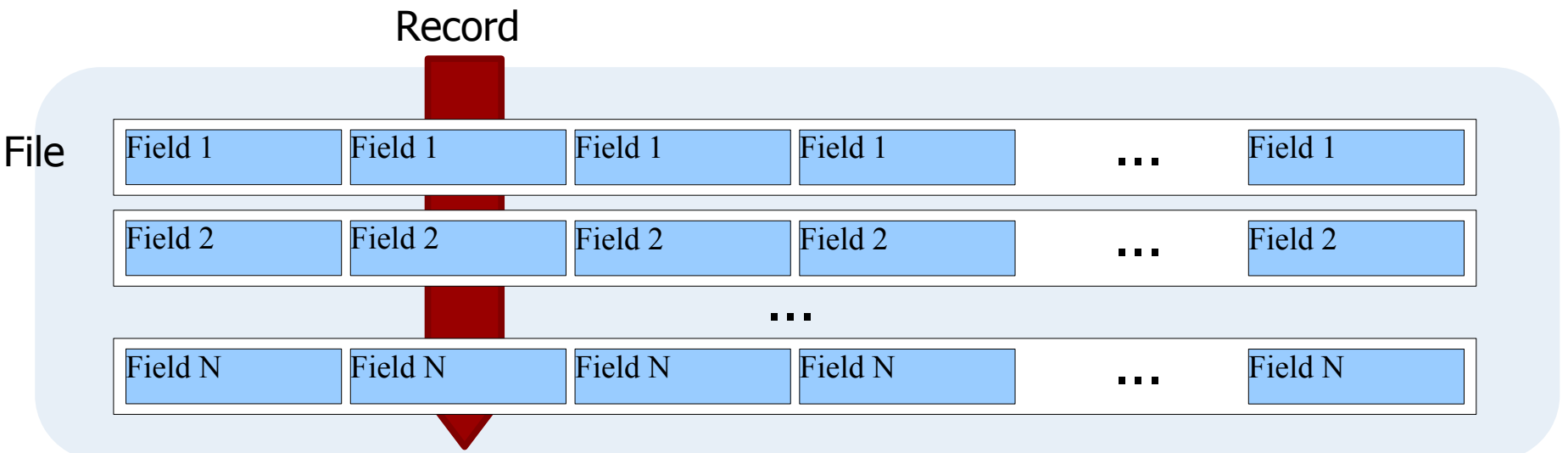


Consequences

- Efficient direct access to individual records
- I/U/D impact only one page
- Cannot avoid reading the full width of the table when scanning a column
  - e.g. select sum(a) from verylargetable;

- Good for transactional processing

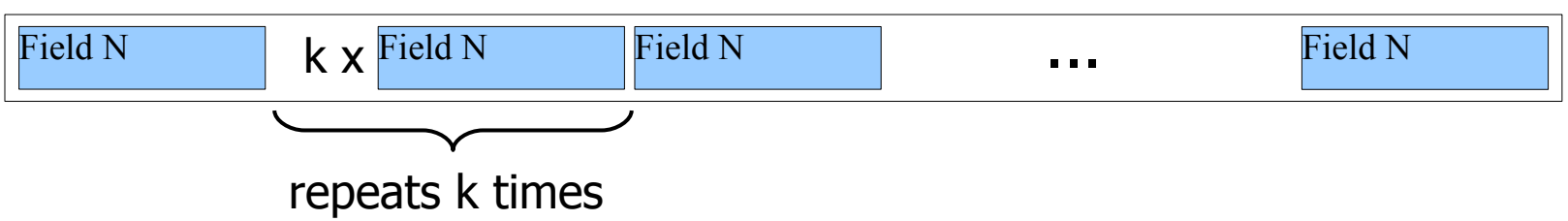
Columnar layout



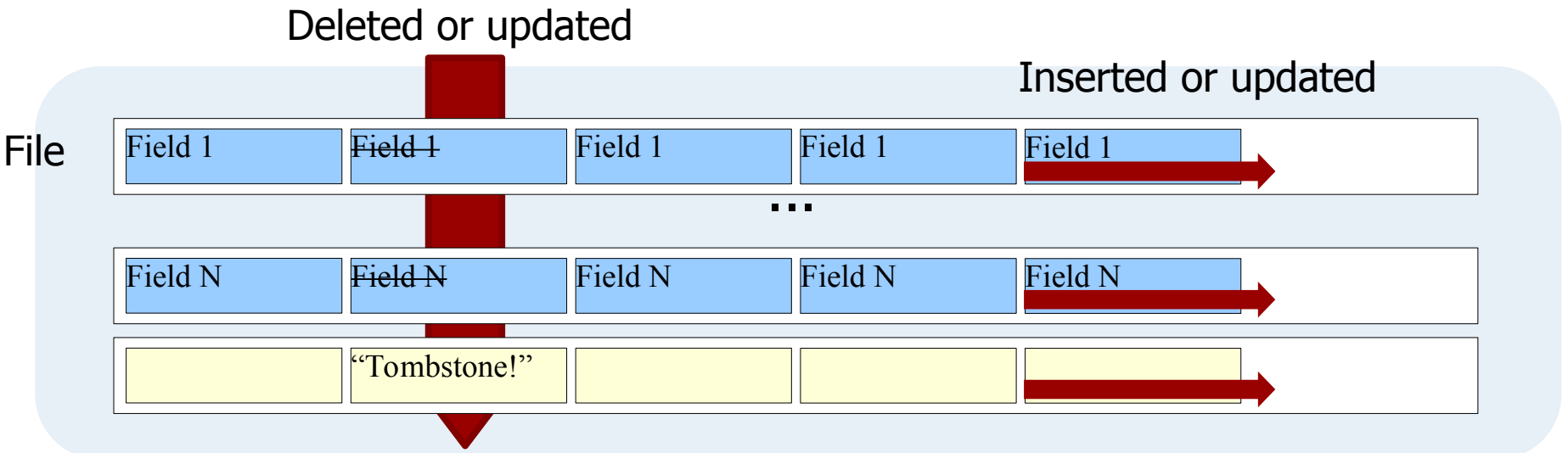
- Each file holds a column (i.e., the same field for all rows)

Columnar layout

- Minimizes data transfer for efficient sequential scan
- Sequences of fields of the same data type can be compressed easily and efficiently:
  - Run length encoding
- Operations on compressed data improve efficiency on all levels of hierarchy:



Columnar layout



- Efficient insert (append data)
- Updates and deletes require:
  - "Tombstone" column
  - Re-writing the file

Consequences

- Column stores: Update/delete are costly operations
- Inserts are cheap by appending to files
- Efficient scan of a subset of columns from a large table
- Data already in a compact vector format

- Better for analytical processing

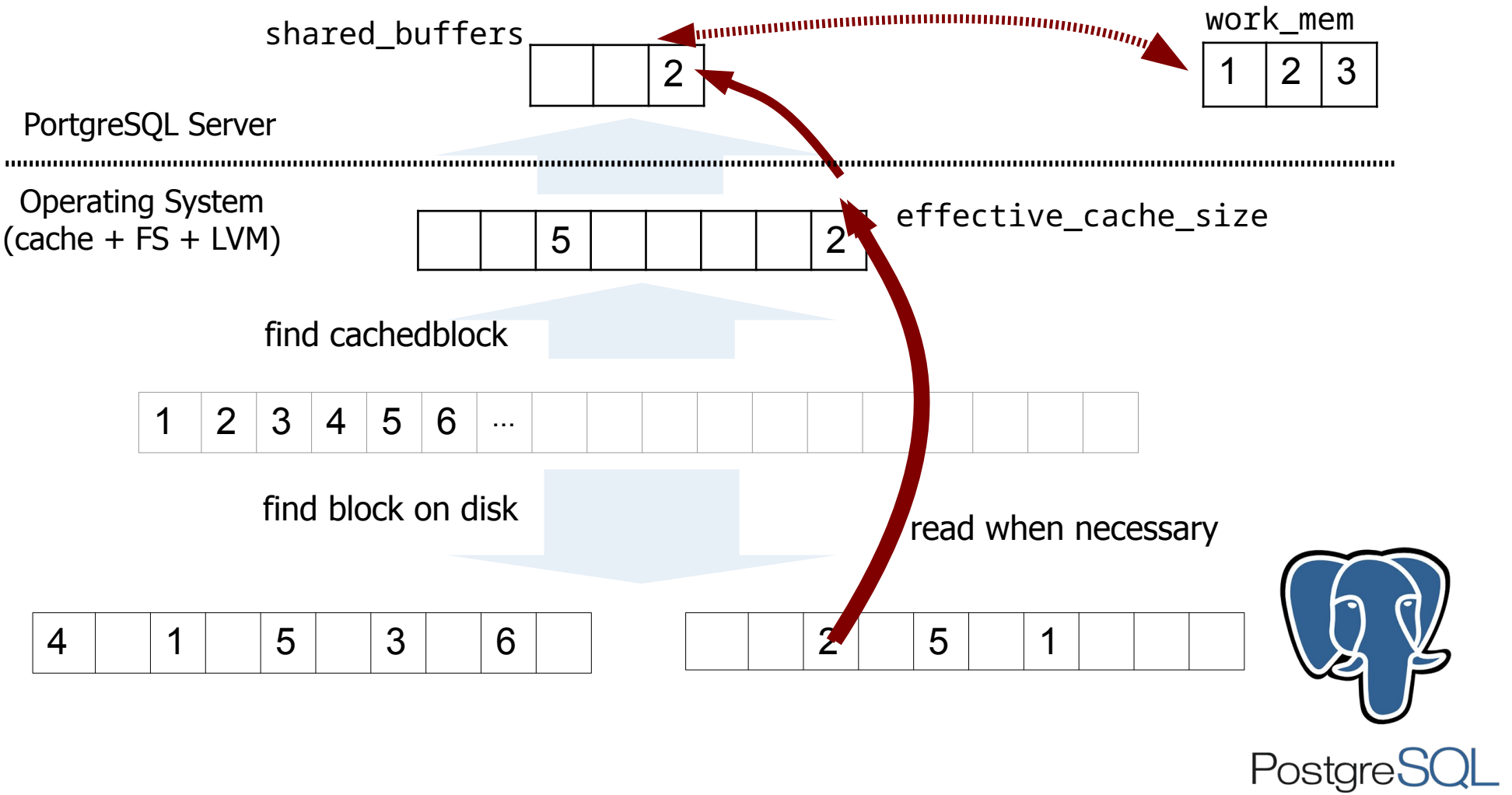
Consequences

- 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
- Row stores: Single column scan in wide tables is a costly operation
- Column stores: Update/delete is a costly operation

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...

Benchmarking

- Repeat workload for a variable number of client threads
- Discard initial and final periods
- Measure:
  - Response time (duration of transactions)
  - Throughput (rate of execution)