Storage Model

- Database on secondary storage (disk, flash)
  - block-oriented
  - relatively slow I/O (milliseconds for disk)
- Copies of some pages are kept in a database buffer cache, managed by the DBMS buffer manager
  - All requests for data go to the buffer manager.
  - Buffer manager ensures that requested data are buffered, copying them from secondary storage if necessary.

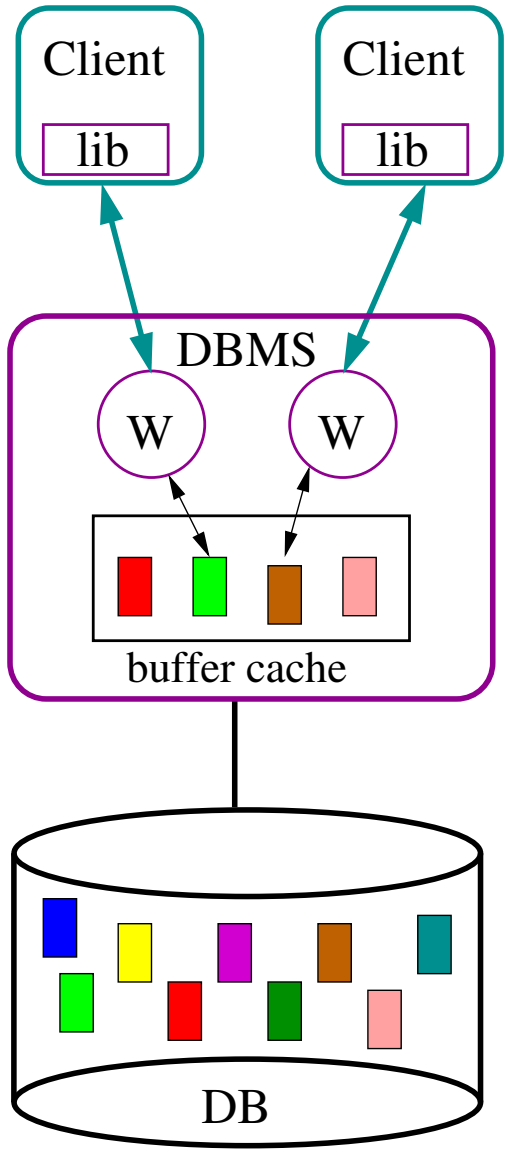
Secondary Storage

- magnetic disk drives - most common
  - organized as an array of fixed-size blocks
  - larger blocks $\Rightarrow$ more efficient access (why?)
  - sequential access faster than non-sequential (why?)
- solid-state drives (SSDs) - becoming more common
  - flash memory used like a magnetic disk drive
  - lower latency, higher I/O bandwidth than disks, but also more expensive ($/gigabyte)
  - sequential and non-sequential access equally fast

"Random" Access

Non-sequential access is often called "random" access.

Database and Buffer Cache



Why Two Tiers?

- Q: Why not keep everything in memory?
- A1: Cost
  - secondary storage much slower but also much cheaper than memory
  - idea: balance performance and cost by keeping frequently accessed data in memory
  - this same tradeoff repeats at many levels of the memory hierarchy!
- A2: Volatility
  - (most) databases are expected to survive failures

Memory-Resident Databases

- though the two-tier storage model is the common case, there are DBMS designed to manage memory-resident databases
- use cases:
  - transient/temporary databases
  - high-performance transactional applications
- database persistence can be achieved by
  - backing database up to secondary storage
  - replicating the database
- single-tier storage model can simplify the DBMS

The 5 Minute Rule (1/2)

- Question: how much memory for the database?
- idea: constraining factor for memory is space, constraining factor for secondary storage (e.g., disk) is bandwidth
- originally proposed by Jim Gray and Franco Putzolu in 1985, updated several times since then

The 5 Minute Rule (2/2)

- consider a page P and suppose
  - P is accessed λ times per second by the workload
  - page size (in MB) is S
  - cost of memory ($/MB) is C_M
  - secondary storage bandwidth is B IOPS (I/O per second)
  - secondary storage cost is C_D
- cost of putting P on secondary storage: $\frac{\lambda}{B} C_D$
- cost of putting P in memory: SC_M
- Put P in memory if $SC_M < \frac{\lambda}{B} C_D$, i.e., if

$$\lambda > B \frac{SC_M}{C_D}$$

The 5 Minute Rule in 1985 and Today

1985

- $C_M = \$15/\mathbf{MB}$
- $S = 8\mathbf{KB}$
- $C_D = \$2000$
- $B = 64$ IOPS
- $\lambda \approx \frac{1}{260\mathbf{seconds}}$ (about once every 5 minutes)

Today

- $C_M = \$0.01/\mathbf{MB}$
- $S = 32\mathbf{KB}$
- $C_D = \$200$
- $B = 70$ IOPS
- $\lambda \approx \frac{1}{9000\mathbf{seconds}}$ (about once every 150 minutes)

5 Minute Rule: Additional Reading

- Jim Gray and Franco Putzolu. *The 5 Minute Rule for Trading Memory for Disk Accesses and the 5 Byte Rule for Trading Memory for CPU Time.* Technical Report 86.1, Tandem Computers, May 1986.

  `http://www.hpl.hp.com/techreports/tandem/TR-86.1.pdf`
- Jim Gray and Goetz Graefe. "The Five-Minute Rule Ten Years Later, and Other Computer Storage Rules of Thumb." *SIGMOD Record*, 26(4):63-68, December 1997.
- Goetz Graefe. *The Five-Minute Rule Twenty Years Later, and How Flash Memory Changes the Rules.* In Proc. Int'l Workshop on Data Management on New Hardware. pp. 1-9, 2007.

Storage Management: Key Issues

- DBMS must lay out logical objects (relations, tuples) into blocks for storage.
  - tuples may be inserted and deleted
  - tuple schema may change over time
  - tuples may include variable-length fields
- DBMS must place blocks on secondary storage
- DBMS must move blocks between secondary storage and the buffer cache, and manage the DBMS buffer cache to maximize performance
- DBMS must be able to find and retrieve logical objects during query evaluation.
  - get tuples based on a search condition e.g., retrieve an employee with a specified employee ID
  - get tuples in some order, e.g, retrieve employee tuples in order or employee ID

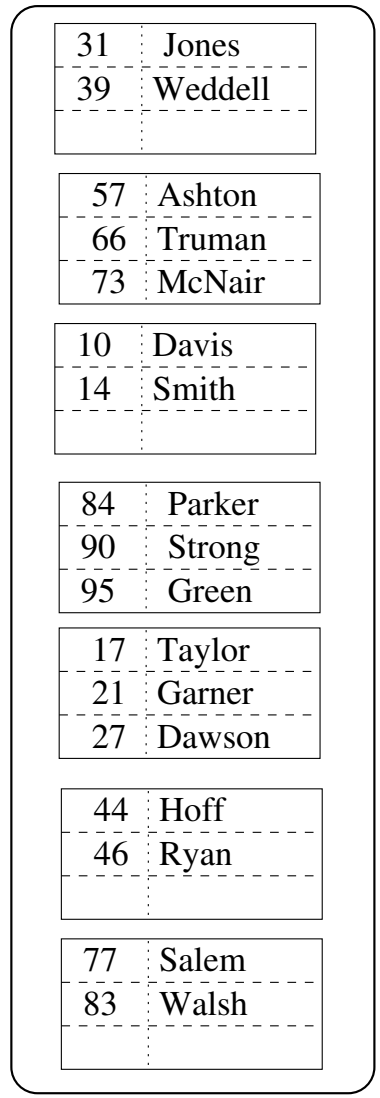
About Blocks

- a disk block is the smallest unit of data that can be read from or written to the device
  - 512 bytes is a typical size
  - a single disk I/O operation will retrieve one or more consecutive blocks
- A DBMS will typically choose its own block size
  - size of each DBMS block is a multiple of the disk block size, e.g., 8KB
  - larger blocks mean
    - increased I/O efficiency
    - reduced retrieval precision

Storing Tuples in Blocks

ID	Surname
10	Davis
14	Smith
17	Taylor
21	Garner
27	Dawson
31	Jones
39	Weddell
44	Hoff
46	Ryan
57	Ashton
66	Truman
73	McNair
77	Salem
83	Walsh
84	Parker
90	Strong
95	Green

"People"
RELATION



DEVICE or FILE

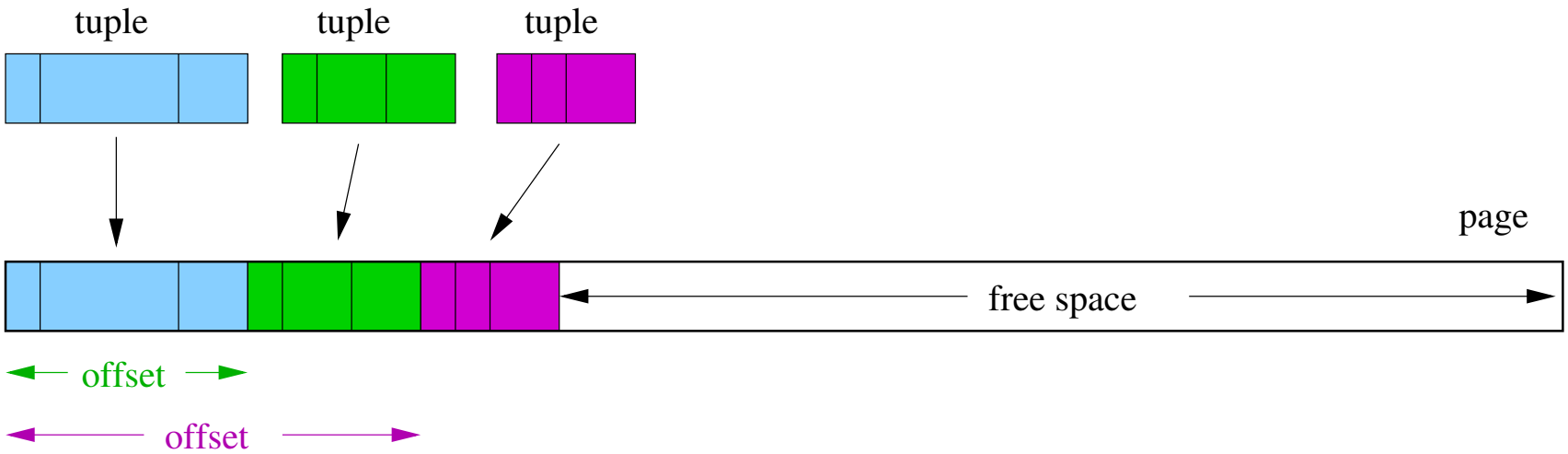
Tuple Layout

- how to organize each page to store tuples?
- some issues:
  - tuples may be inserted and deleted
  - variable length fields
  - tuple schema may change
  - tuple/record identifiers

RIDs

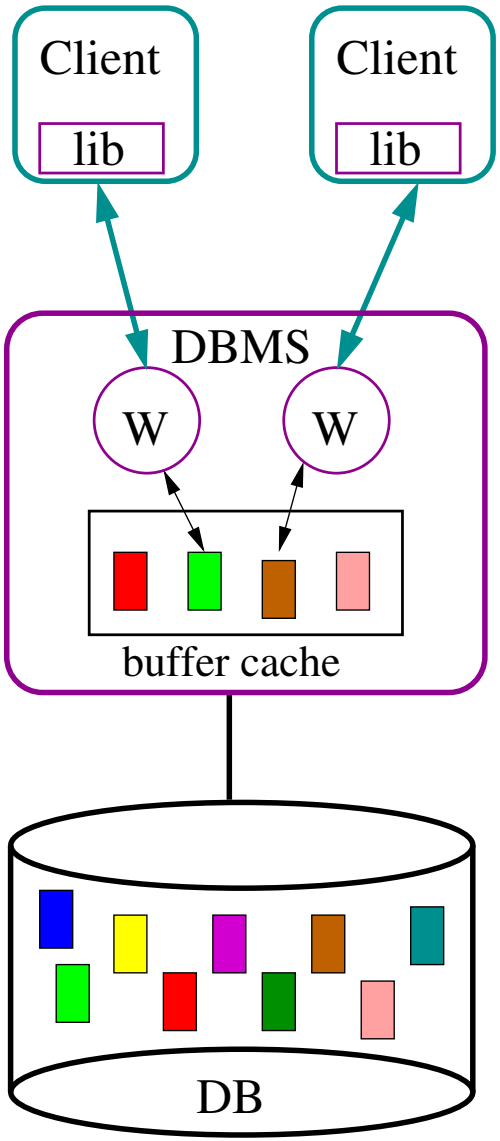
Tuple/record identifiers, sometimes known as RIDs or TIDs, are used, e.g., in indexes, to refer to tuples. They are similar to tuple pointers in that they can be used to determine the location of a stored tuple.

A Simple Packed Page Organization



- RID is page number plus offset within page
- updates to variable-length fields may force tuple moves
- tuple insertions and deletions will result in fragmentation of free space
- moving tuples within page (e.g., because of update or defragementation) will change tuples' RIDs

Buffer Management



- all page access are made through the DBMS buffer manager
- on read miss retieve page from disk/file, evicting another page if necessary
- on write, update cached page, mark it dirty

Pinning and Unpinning

- When a DBMS worker requests a page, the buffer manager retrieves the page (if necessary), then increments a pin count in the page's buffer frame
- A frame's pin count indicates how many workers are currently using the page in that frame.
- Workers must explicitly unpin pages (causing the pin count to be decremented) when they are done using them
- Pinning pages prevents the buffer manager from evicting them while they are being used by DBMS workers.
- Pinning/unpinning are necessary because workers read and update pages directly in the buffer cache

DBMS Buffer Cache Replacement Policies

- DBMS buffer cache manager, like other cache managers, exploits temporal locality to guide page replacement
- DBMS may also be able to exploit knowledge of workers' page access patterns to further guide replacement decisions
  - example: consider a table scan of a large table
- some examples of "DBMS-friendly" replacement policies:
  - generalized CLOCK
  - LRU-*K* and 2Q

The CLOCK Replacement Policy (review)

- Each buffer cache frame has a `use` bit, set whenever the frame's page is used by a worker
- To choose a victim, the buffer manager cycles through frames until it finds an unpinned frame whose `use` bit is not set. Frames with set `use` bits are skipped, but their bits are cleared as the buffer manager cycles through.

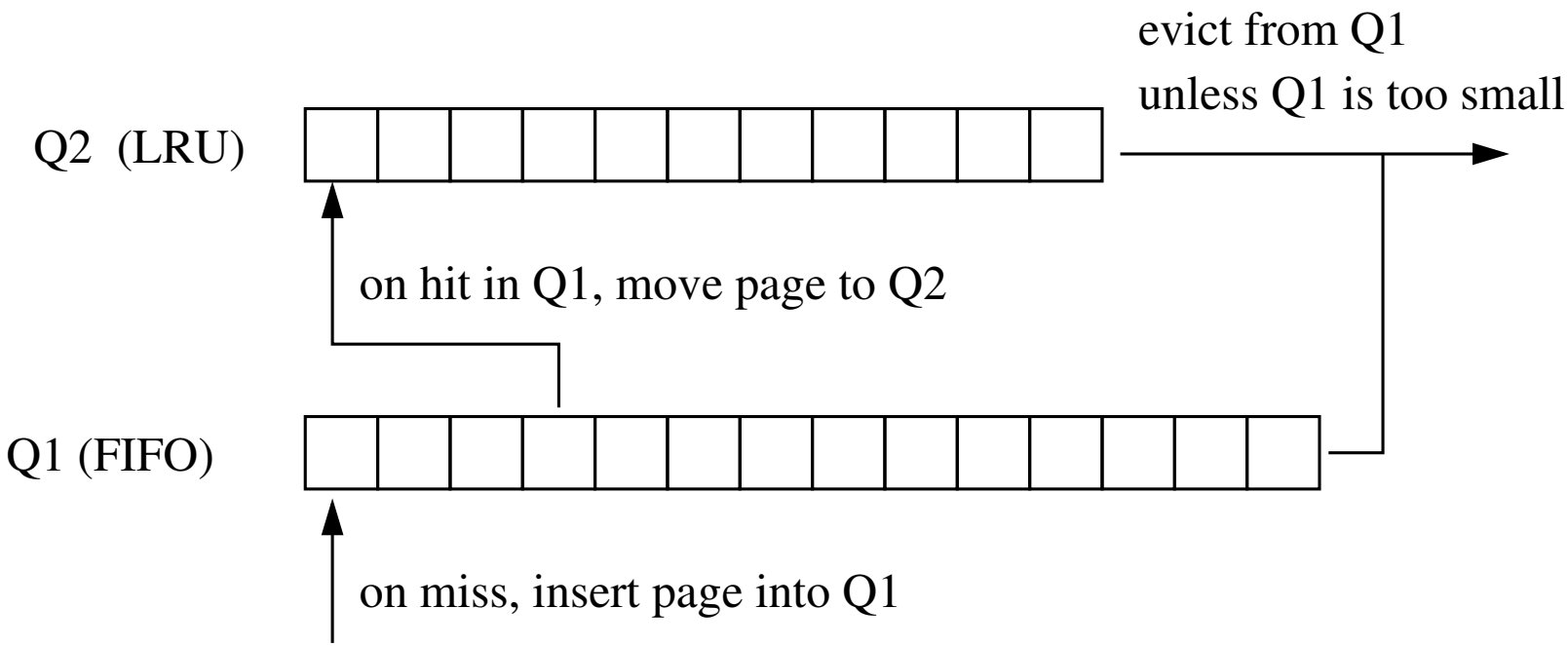
Generalized CLOCK

- generalized CLOCK is similar to clock, but the `use` bits are replaced by counters, set when the frame's page is used by a worker
- To choose a victim, the buffer manager cycles through frames until it finds one whose counter value is zero. Frames with non-zero counters are skipped, but their counters are decremented as the buffer manager cycles through.
- generalized CLOCK allows the DBMS buffer manager to affect page replacement by controlling the counter value that is set when the page is used by a worker. For example:
  - set counters to low values during table scans
  - set counters to high values when accessing index pages

LRU-*K* Replacement

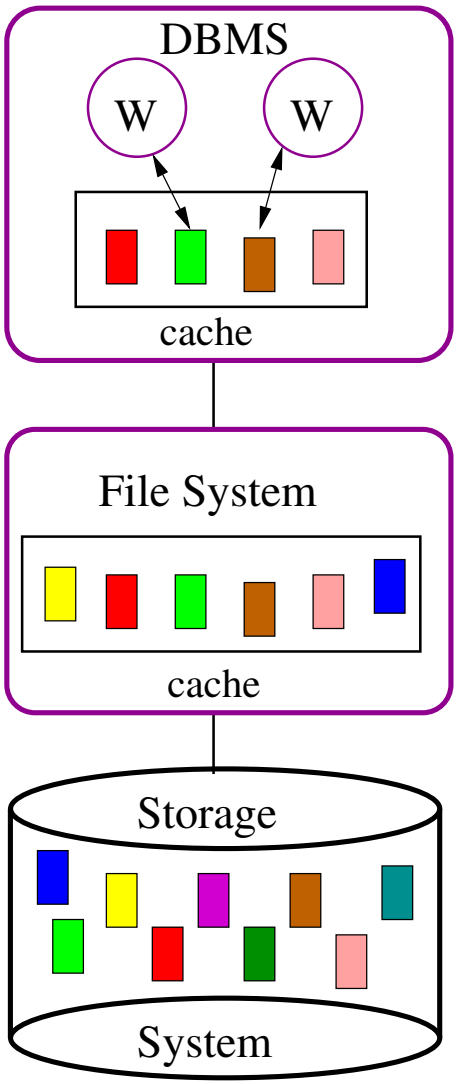
- The LRU policy evicts the page whose most recent use is the oldest (among all cached pages)
- The LRU-*K* policy evices the page whose *K*th most recent use is the oldest (among all cached pages)
- LRU-*K* is actually a family of policies, one for each value of *K*. The LRU-1 policy is the same as LRU.
- LRU-*K* ($K \geq 2$) policies require the buffer manager to track reference histories for pages that are not in the buffer cache (as well as those that are cached).

2Q Replacement



- Q2 holds pages that are referenced a second time before falling off of Q1
- Sizes of Q1 and Q2 will vary

The World Beneath the DBMS



- storage system may be a single device or a more elaborate system with many devices
- storage system may cache blocks, reorder requests

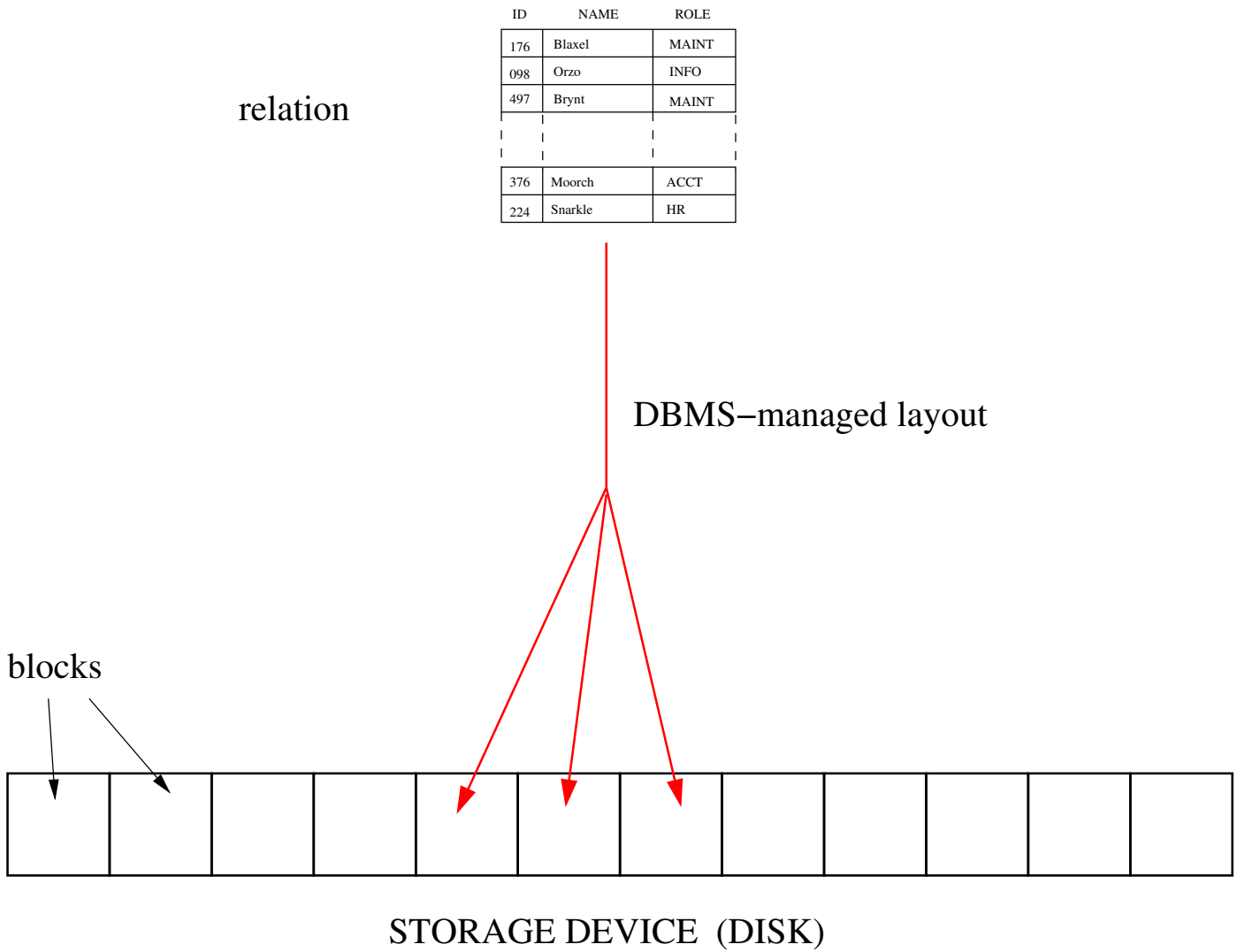
Why Buffer Pages in the DBMS?

- DBMS wants direct access to buffered pages
  - uses block pinning and unpinning operations
- DBMS buffer manager can exploit its knowledge to decide what to buffer
  - DBMS has information about queries' data access patterns
  - DBMS has knowledge of what logical objects are stored in each page
- DBMS needs to control when blocks are written to properly implement transactions (more on this later)
  - must know when a block is stored persistently
  - must control the order in which blocks are made persistent

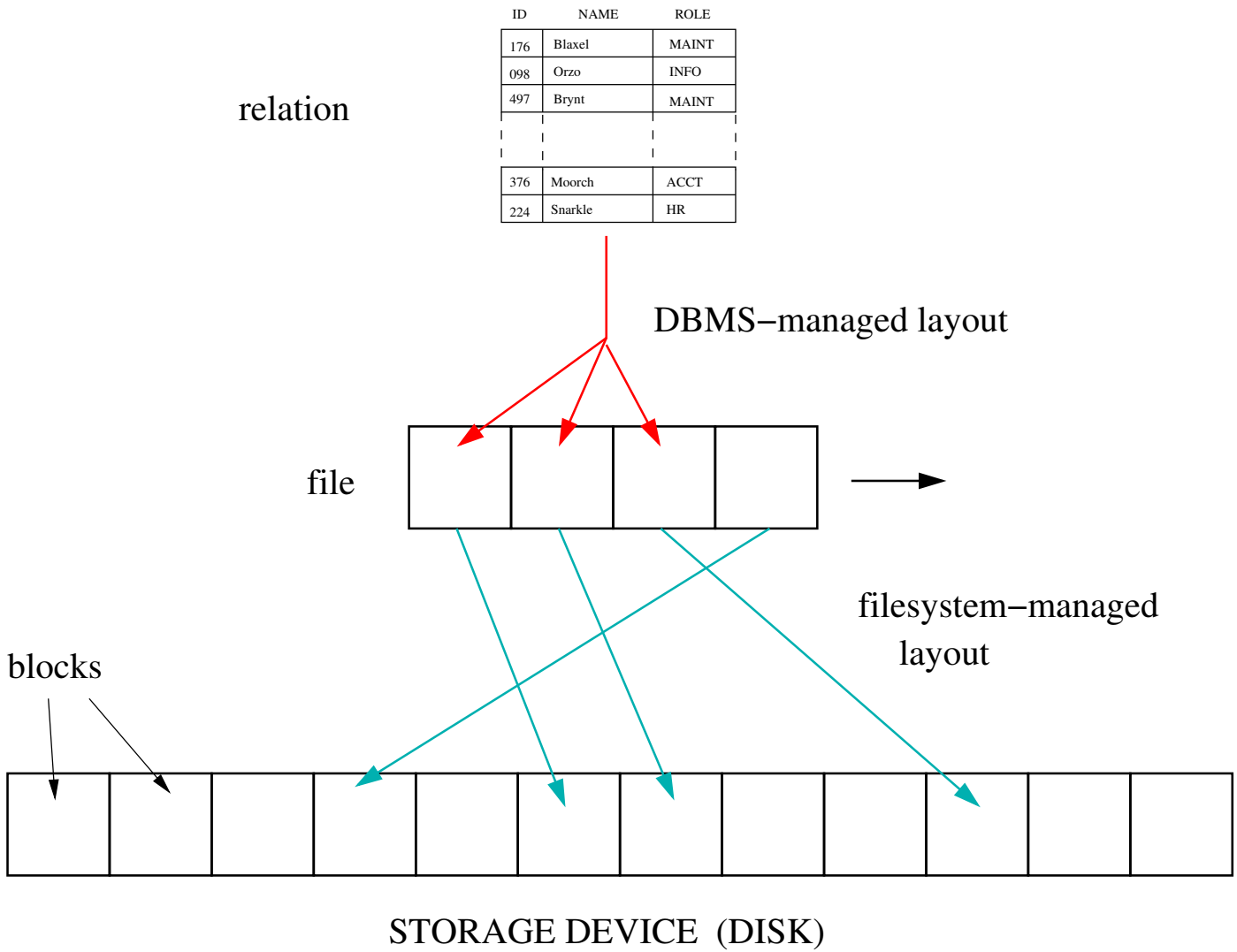
Impact of File and Storage Systems

- abstractions provided by the file system can simplify the DBMS
  - DBMS can use one or more files to store the database, e.g., file-per-database or file-per-relation.
  - files can grow, move - file system takes care of managing space on the storage device
- underlying file and storage Systems also present some potential hazards
  - DBMS loses control of physical placement of blocks
  - multi-level caching
    - problem of cache inclusion
  - DBMS loses control of block persistence and ordering of I/O operations

Data Layout with Direct Device Management



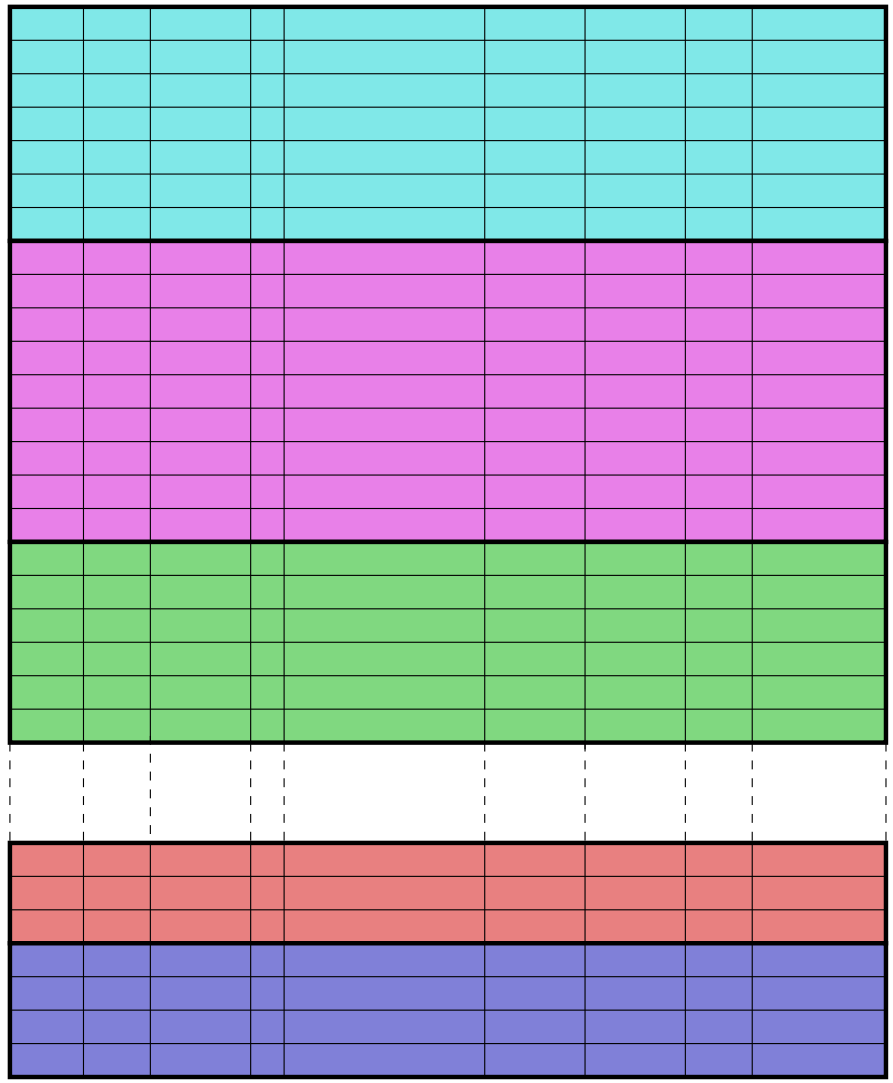
Data Layout with a File System



Coping with the World Beneath

- DBMS Option 1: use file system, but explicitly force writes when necessary
  - use `fsync` or similar file system operation
  - guarantees the DBMS that a page is persistently stored
  - allows DBMS to control the order in which pages are made consistent
- DBMS Option 2: use file system, but bypass file system cache completely
  - use files or file system in "direct I/O" mode
  - all writes are forced
  - avoids cache inclusion problem and double copying
- DBMS Option 3: bypass the file system completely
  - store data directly on the underlying storage device or system, using raw I/O interface
  - allows DBMS to control block placement

Data Layout: Row Stores and Column Stores



Column Store Applications

- column stores are best-suited to applications with
  - wide relations
  - queries that access few columns, and that are not very selective
  - high query/update ratio (since update/insert/delete are tuple-oriented)
- analytic workloads often have suitable properties, e.g.,
  - large aggregation queries
  - infrequent, batch updates (e.g., in data warehouses)

Data Warehouse Example

- consider a database with a large table containing 2 years of sales data for a retailer
  - `tranId,prodId,custId,time,qty,price,...`
  - suppose 100 bytes of data per sale, 1000 stores, 10 registers per store, 10 trans/register/min for 10 hours per day
  - 10 MB/min, 6 GB/day, 2.2 TB/year, 4.4 TB in two years
- example queries
  - total sales per Ontario store, by day of week
  - average daily sales of Shreddies, by quarter
- Suppose system has 40 disk drives, 50MB/sec per disk drive, 2GB/sec total
  - total time to read entire table is about 2200 seconds (more than half an hour)
- goal of column store is to read only the necessary columns, not the whole table

Column Stores

- For "long, narrow" queries, advantage of column stores is that they do not need to access irrelevant columns.
- For large queries that are I/O-bound, reducing the amount of data that has to be accessed can improve performance substantially
- Column store DBMS can also exploit other column-oriented optimizations, such as
  - column-specific compression techniques to further reduce the amount of data to be retrieved
  - batch iteration - pass column values from multiple rows as a batch from one operator to the next in a query plan
- in this class, we focus primarily on row stores