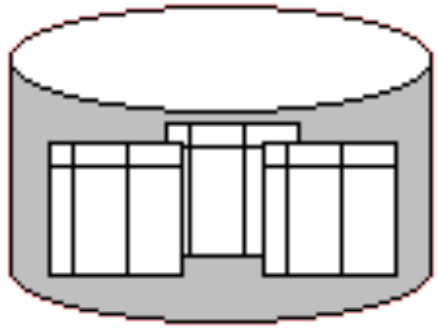
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Συστήματα Διαχείρισης Βάσεων Δεδομένων

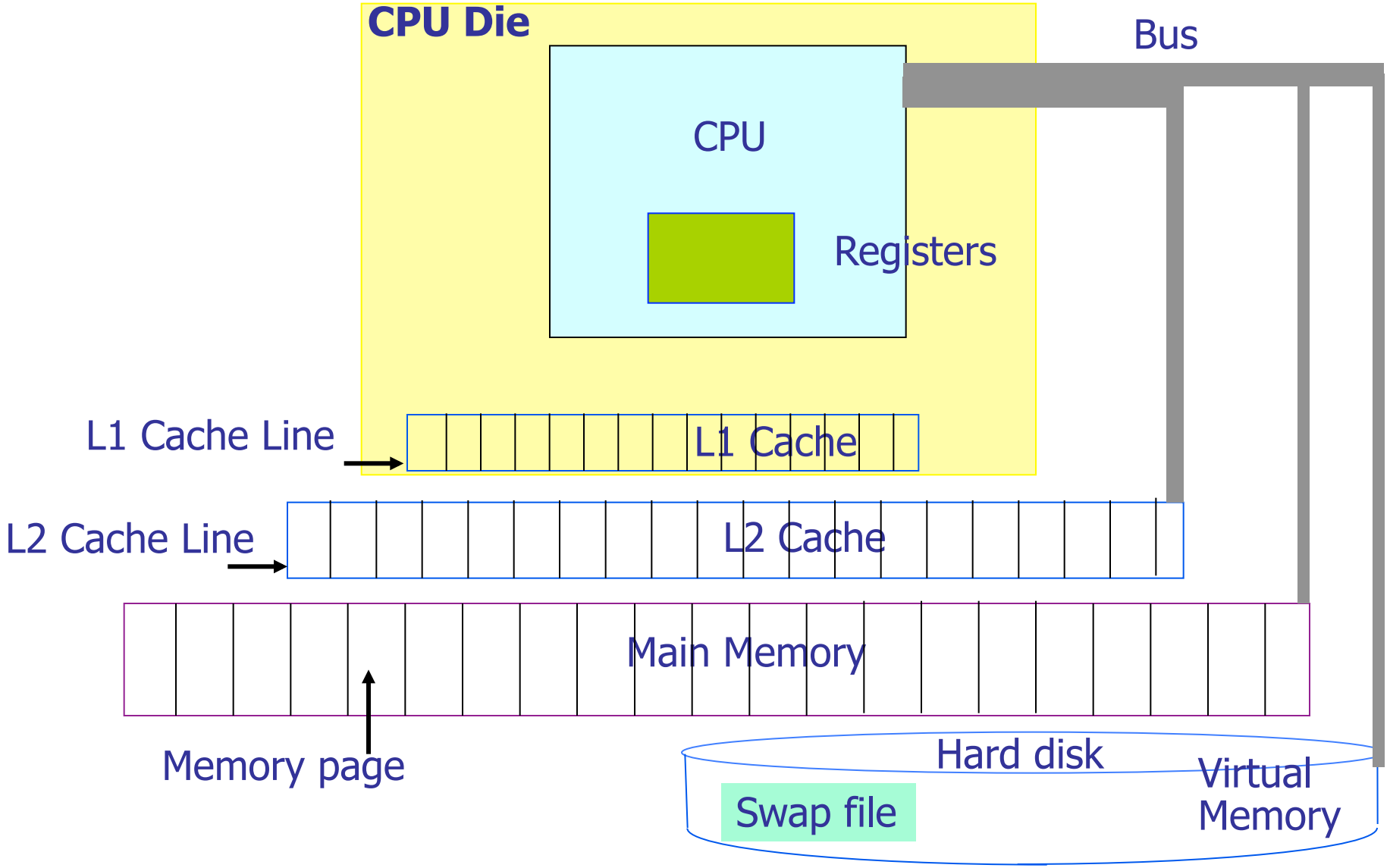
Φροντιστήριο 1: Tutorial on buffer management

Δημήτρης Πλεξουσάκης
Τμήμα Επιστήμης Υπολογιστών

TUTORIAL ON BUFFER MANAGEMENT



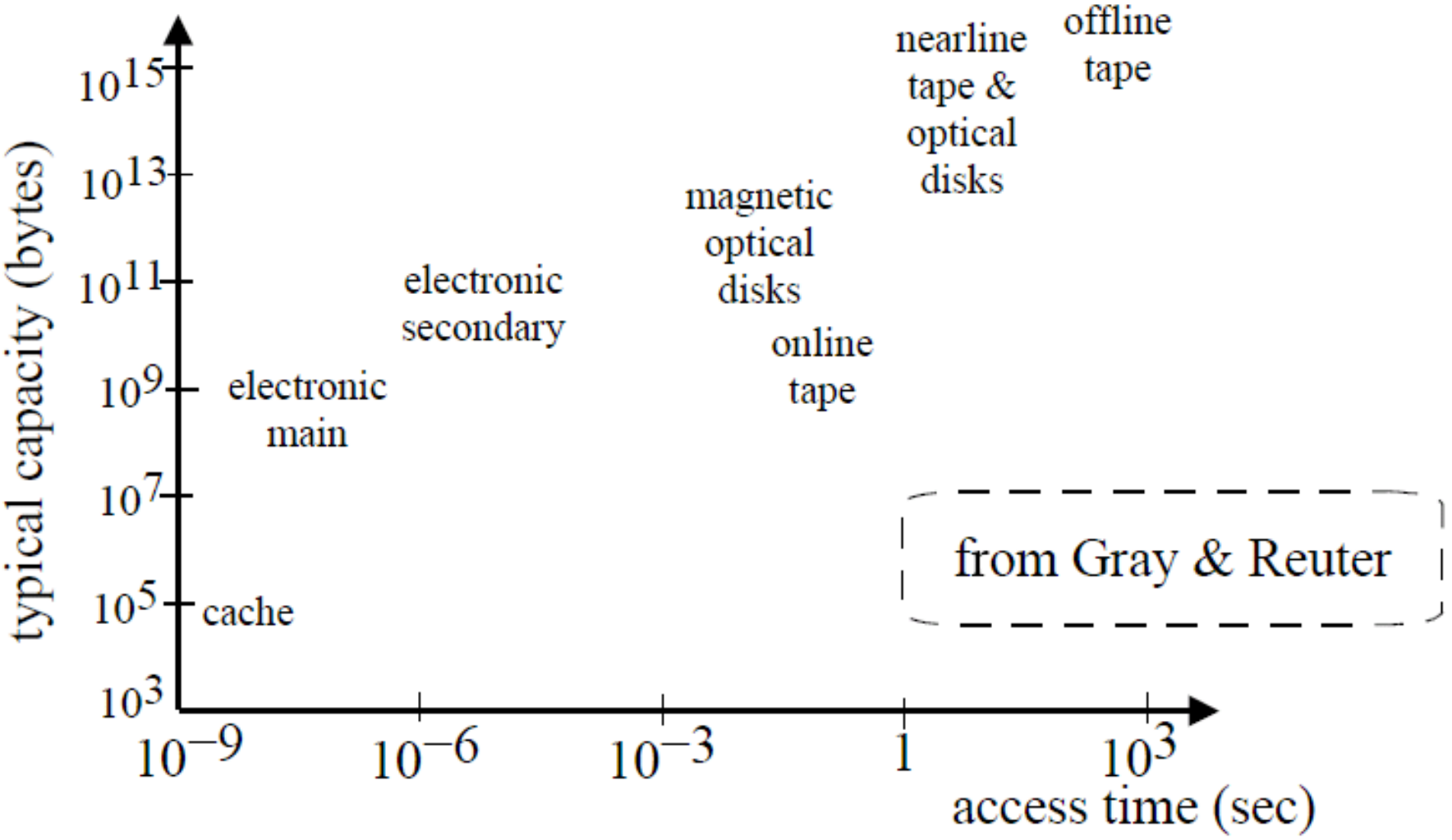
Typical Memory Hierarchy



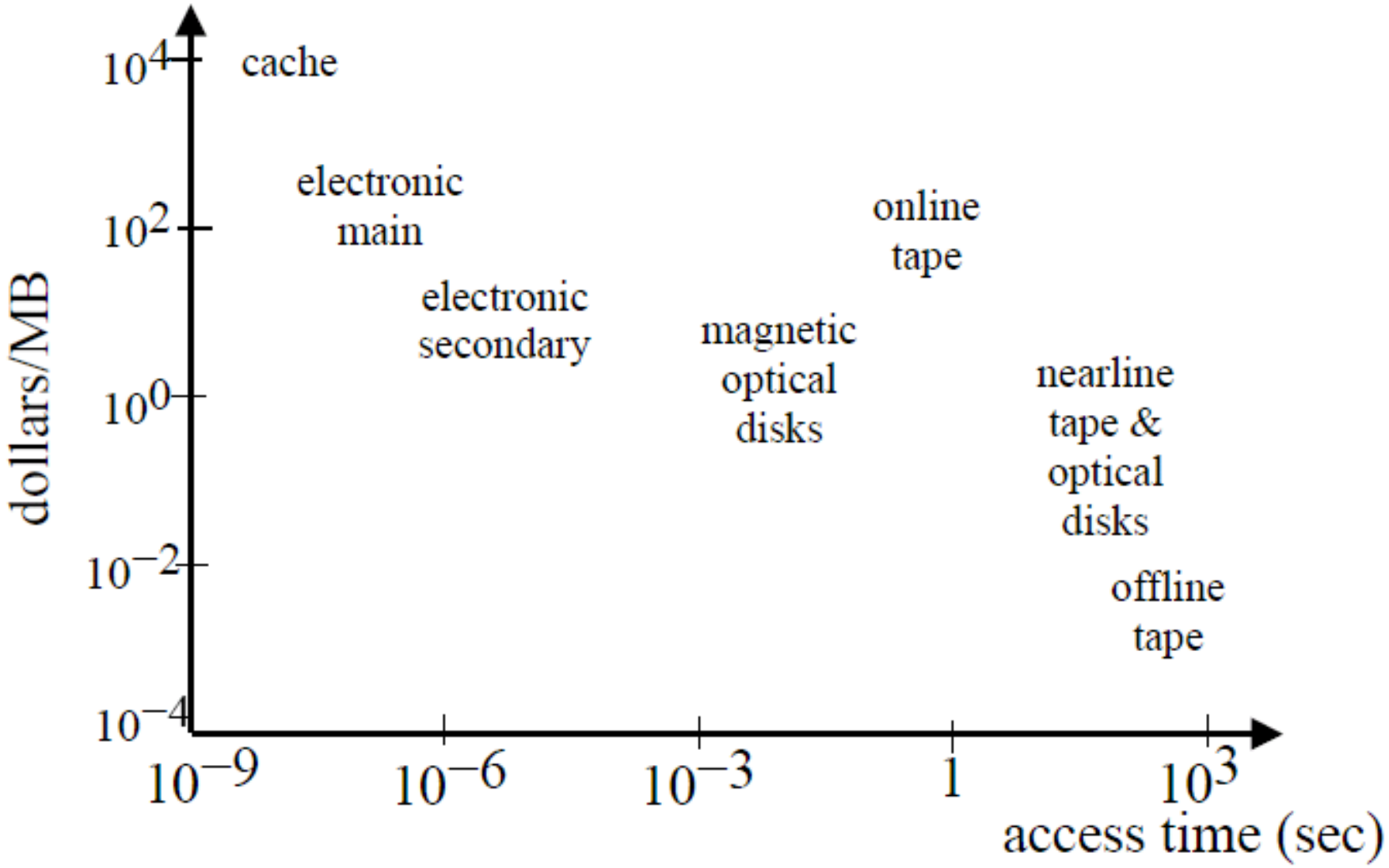
Memory Hierarchy

	L1 Cache	L2 Cache	Main memory
Block size	16--32 bytes	32--64 bytes	4--16 KB
Size	16--64 KB	256KB--8MB	4--16 GB
Hit time	1 Clock Cycle	1--4 Clock Cycles	10--40 Clock Cycles
Backing Store	L2 Cache	Main memory	Disk
Block replacement	Random	Random	Replacement strategies
Miss penalty	4--20 clock cycles	40--200 clock cycles	~6M clock cycles

Storage Capacity

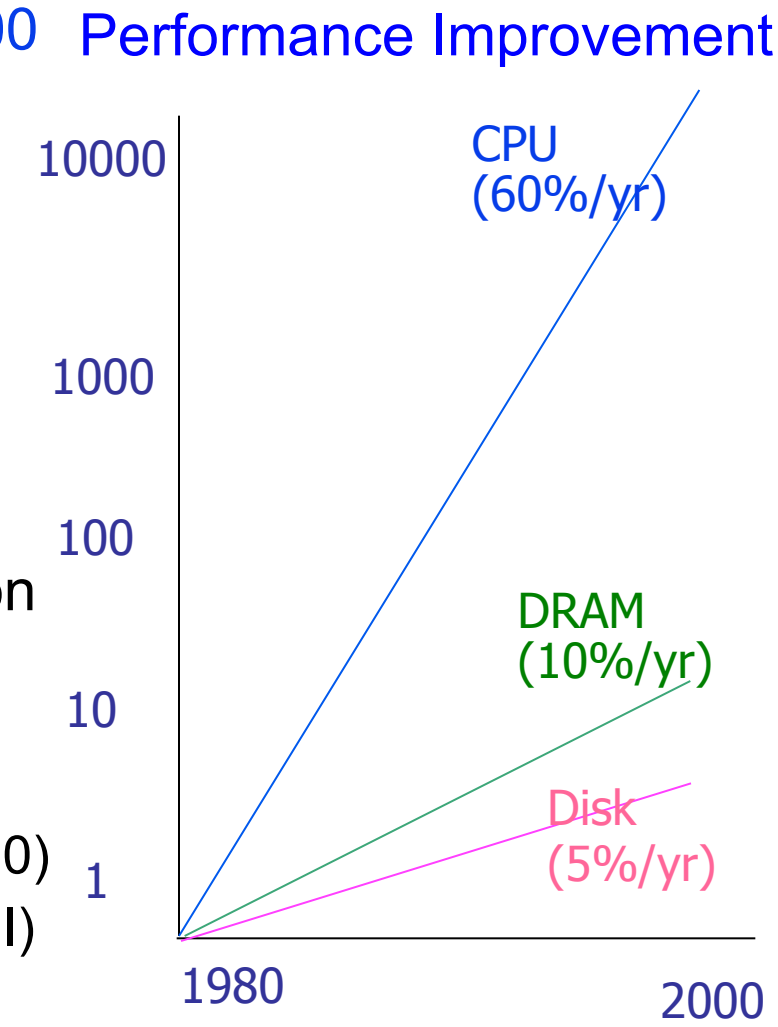


Storage Cost



Latest Disk Performance Measures

- The fastest enterprise disks revolve at 15000 revolutions per minute (rpm) (e.g. Seagate Cheetah 15K)
- By late 2010 the fastest high-performance drives were capable of
  - an average latency of 2ms
  - an average seek time of between 3 and 6ms and
  - maximum data transfer rates in the region of
    - 133 MB/s for Parallel ATA drives (Ultra ATA/133)
    - 600 MB/s for Serial ATA drives (SATA 600)
    - 640 MB/s for SCSI drives (Ultra-640 SCSI)



Pages and Blocks

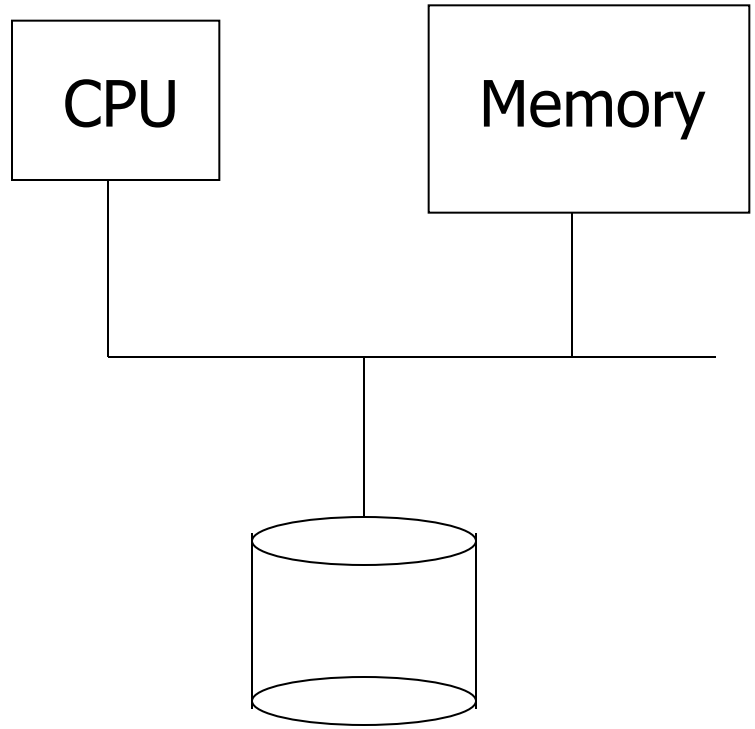
- Data files decomposed into pages
  - Fixed size piece of contiguous information in the file
  - Unit of exchange between disk and main memory
- Disk divided into page size blocks of storage
  - Page can be stored in any block
- Disk-block access methods must take care of some information within each block, as well as information about each block:
  - allocate records (tuples) within blocks
  - support record addressing by address and by value
  - support auxiliary (secondary indexing) file structures for more efficient processing
- Application's request to read item satisfied by the following:
  - Read page containing item to buffer in DBMS
  - Transfer item from buffer to application
- Application's request to change item satisfied by the following:
  - Read page containing item to buffer in DBMS (if it is not already there)
  - Update item in DBMS buffer
  - (Eventually) copy buffer page to page on disk

Block Access Time

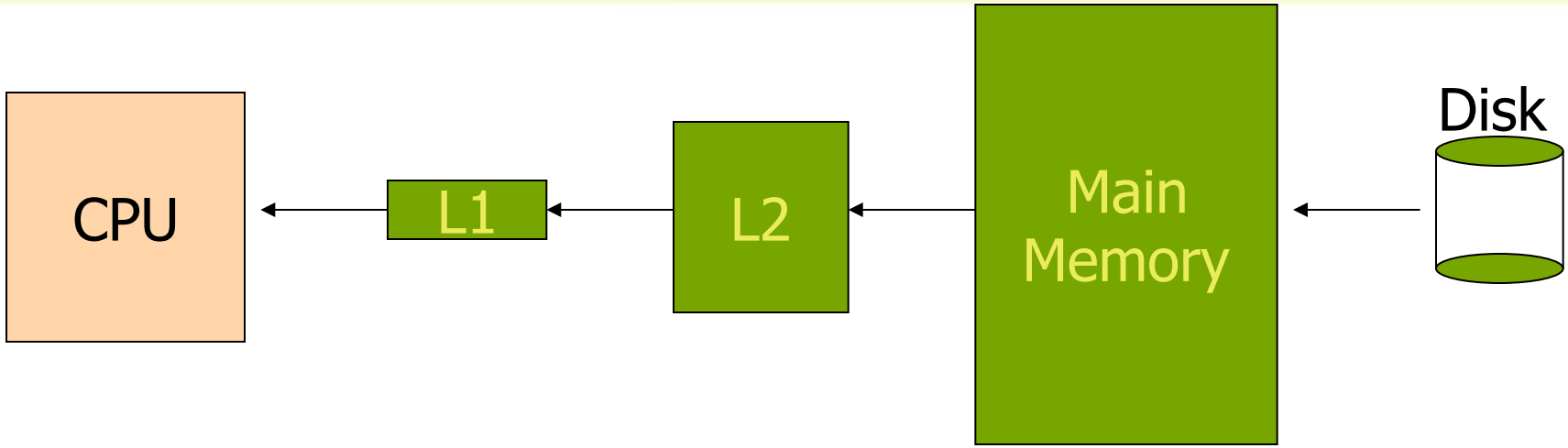
- Time= Seek Time+ Rotational Delay+ Transfer Time+ Other
- Rule of Thumb
  - Random I/O: Expensive
  - Sequential I/O: Much less
- Example:1 KB Block Size
  - Random I/O: ~ 20 ms
  - Sequential I/O: ~ 1 ms
- Improving Access Time of Secondary Storage
  - Organization of data on disk
  - Disk scheduling algorithms
    - e.g., elevator algorithm
  - Multiple disks
  - Mirrored disks
  - Prefetching and large-scale buffering

Complete Model Specification

- Data resides on some **N** disk blocks
  - Implies **N *B** data elements!
- Computation can be performed only on data in main memory
  - Main memory can store **M** blocks
  - Data size much larger than main memory size
  - **N >> M:** interesting case
- Model of Computation
  - Minimum transfer unit: a page = b bytes or **B** records
  - n records -> n/B = **N** pages
  - I/O complexity: in number of pages

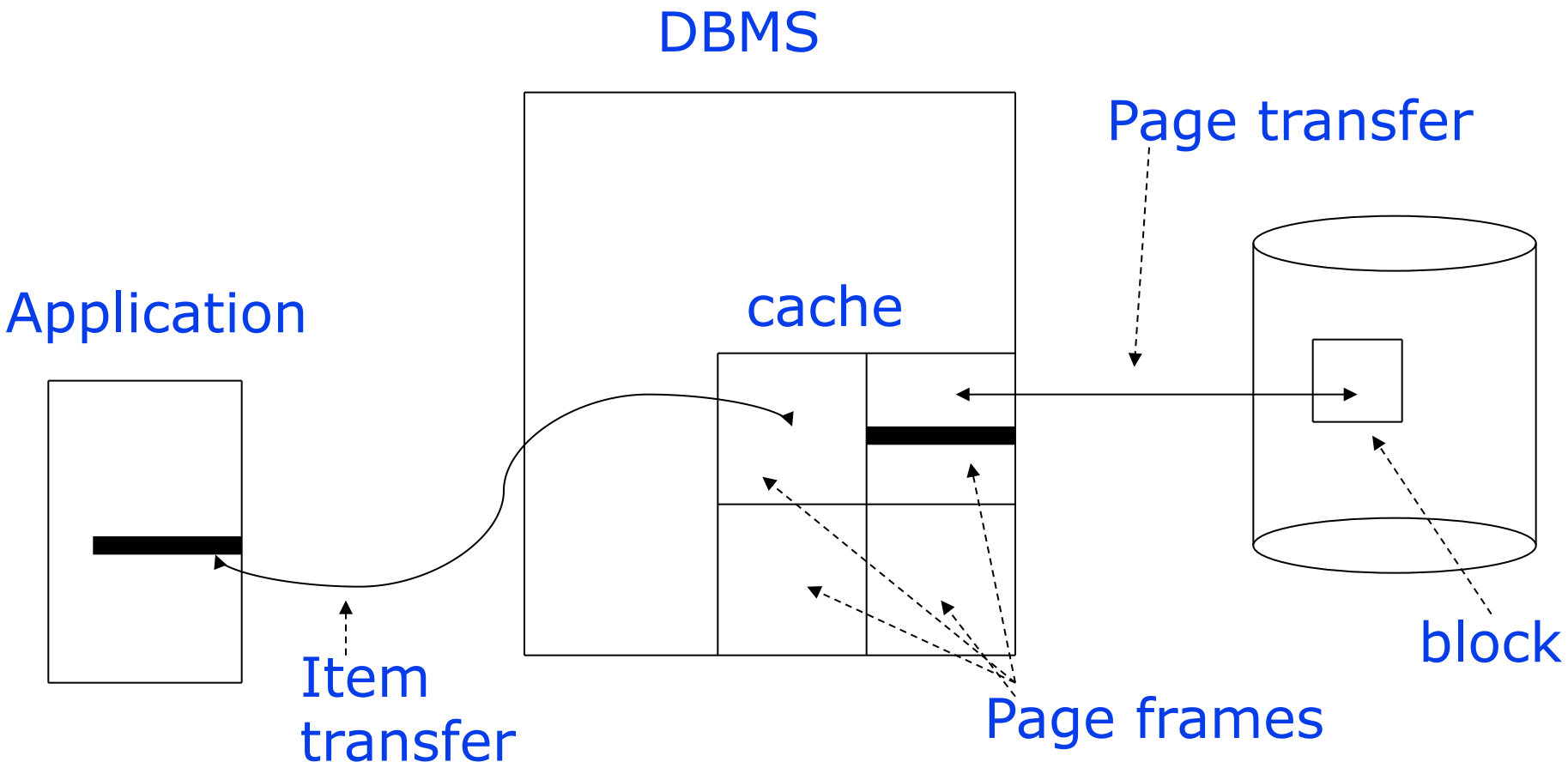


Reducing Number of Page Transfers



- DBMS seek to minimize the number of block transfers between the disk and memory
  - We can reduce the number of disk accesses by keeping as many blocks as possible in main memory
- Keep cache of recently accessed pages in main memory
  - Goal: request for page can be satisfied from cache instead of disk
  - Purge pages when cache is full
    - For example, use LRU algorithm
    - Record clean/dirty state of page (clean pages don't have to be written)

Accessing Data Through Cache



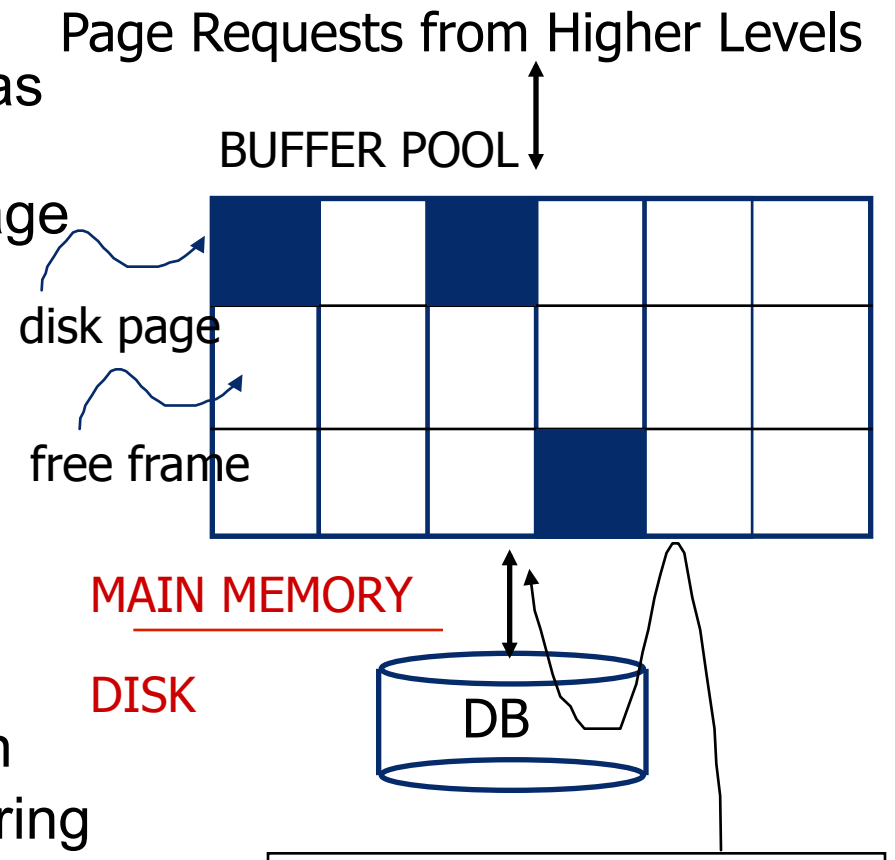
- Buffer - portion of main memory used to store copies of disk blocks
- Buffer manager - subsystem responsible for allocating buffer space in main memory and handling block transfer between buffer and disk

Buffering Strategies

- Double Buffering: Two buffers can be used to allow processing and I/O to overlap
  - Suppose that a program is only writing to a disk
  - CPU wants to fill a buffer at the same time that I/O is being performed
  - If two buffers are used and I/O-CPU overlapping is permitted, CPU can be filling one buffer while the other buffer is being transmitted to disk
  - When both tasks are finished, the roles of the buffers can be exchanged
  - The actual management is done by the OS
- Multiple Buffering: instead of two buffers any number of buffers can be used to allow processing and I/O to overlap
- Buffer pooling:
  - There is a pool of buffers
  - When a request for a block is received, OS first looks to see that the block is in some buffer
  - If not there, it brings the block to some free buffer. If no free buffer exists, it must choose an occupied buffer (usually LRU strategy is used)

Basic Buffer Concepts

- Two variables maintained for each frame in buffer pool
  - Pin count
    - Number of times page in frame has been requested but not released
    - Number of current users of the page
    - Set to 0 initially
  - Dirty bit
    - Indicates if page has been modified since it was brought into the buffer pool from disk
    - Turned off initially
- Why Pin a Page?
  - Page is in use by a query/transaction
  - Log/recovery protocol enforced ordering
  - Page is hot and will be needed soon, e.g., root of index trees



When a Page is Requested ...

- If requested page is not in pool:
  - Choose a frame for replacement
    - If a free frame is not available, then choose a frame with pin count 0
      - All requestors of the page in frame have unpinned or released it
  - If dirty bit is on, write page to disk
  - Read requested page into chosen frame
  - Pin the page (increment the pin count)
    - A page in pool may be requested many times
  - Return address of page to requestor

Buffer Management: Parameters

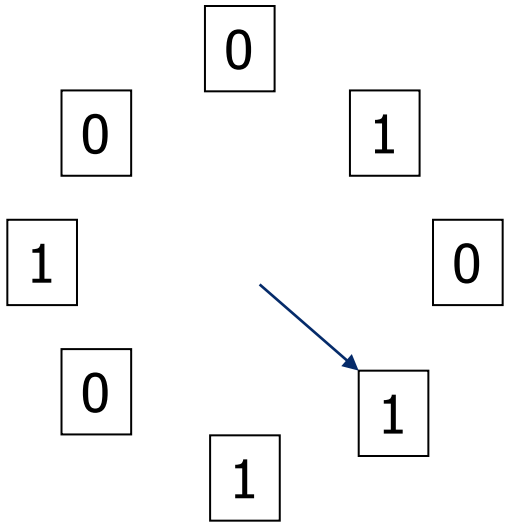
- What are the design parameters that distinguish one BM from another?
  - Buffer allocation: subdividing the pool
    - Who owns a subdivision?
      - Global? Per query? Per relation?
    - How many pages to allocate? (working set)
  - Replacement policy
    - Which page to kick out when out of space?
    - Policy can have big impact on number of I/Os
      - Depends on the access pattern
  - Load control
    - Determine how much load to handle
- Frame is chosen for replacement by a replacement policy:
  - Least-recently-used (LRU)
  - Most-recently-used (MRU)
  - First-In-First-Out (FIFO)
  - Clock / Circular order

Buffer Replacement Policies

- Least-recently-used (LRU)
  - Buffers not used for a long time are less likely to be accessed
  - Rule: Throw out the block that has not been read or written for the longest time
    - Maintain a table to indicate the last time the block in each buffer has been accessed
    - Each database access makes an entry in table
      - Expensive ?
- First-In-First-Out (FIFO)
  - Rule: Empty buffer that has been occupied the longest by the same block
    - Maintain a table to indicate the time a block is loaded into the buffer
    - Make an entry in table each time a block is read from disk
      - Less maintenance than LRU
      - No need to modify table when block is accessed

Buffer Replacement Policies

- Clock Algorithm
  - Buffers arranged in a circle
    - Each buffer associated with a Flag (0 or 1)
    - Flag set to 1 when
      - A block is read into a buffer
      - Contents of a buffer is accessed
  - A "hand" points to one of the buffers
    - Rotate clockwise to find a buffer with Flag=0
    - If it passes a buffer with Flag=1, set it to 0
  - Rule: Throw out a block from buffer if it remains unaccessed when the hand
    - makes a complete rotation to set its flag to 0, and
    - another complete rotation to find the buffer with its 0 unchanged

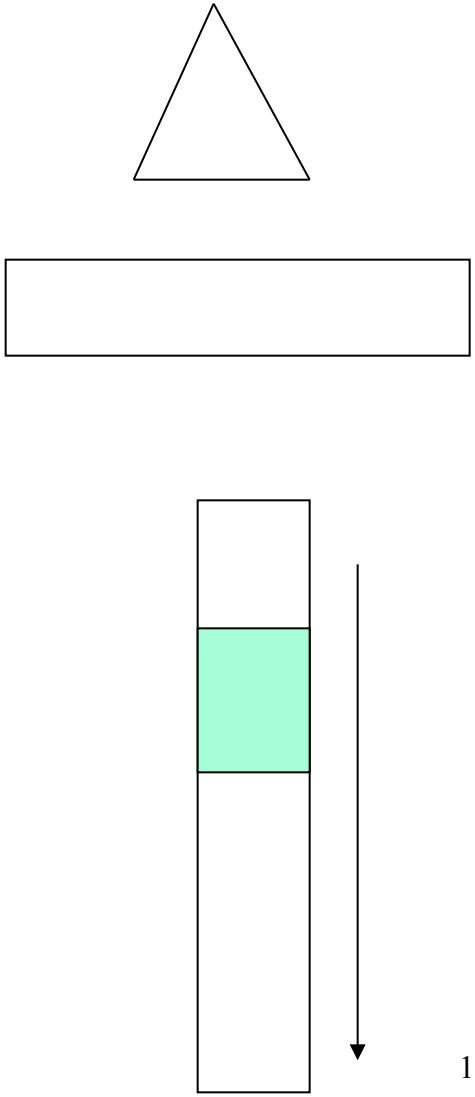


LRU-K

- Self-tuning
  - Approach the behavior of buffering algorithms in which page sets with known access frequencies are manually assigned to different buffer pools of specifically tuned sizes
  - Does not rely on external hints about workload characteristics
  - Adapts in real time to changing patterns of access
  - Provably optimal
- GUESS when the page will be referenced again
  - Problems with LRU?
    - Makes decision based on too little info
    - Cannot tell between frequent/infrequent refs on time of last reference
    - System spends resources to keep useless stuff around

Examples

- Example 1
  - CUSTOMER has 20k tuples
  - Clustered B+-tree on CUST_ID, 20 b/key
  - 4K page, 4000 bytes useful space
  - 100 leaf pages, 10000 data pages
  - Many users (random access)
  - Buffer size: 101
  - References L1,R1,L2,R2,L3,R3,…
  - Probability to ref Li is .005, to ref Ri is .00005
- Example 2
  - R has 1M tuples
  - A bunch of processes ref 5000 (0.5%) tuples
  - A few batch processes do sequential scans



LRU-K Basic Concepts

- Idea: Take into account history – last K references
  - Classic LRU: K = 1 (LRU-1), keeps track of the last reference only
- Parameters:
  - Pages N={1,2,…,n}
  - Reference string $r_1, r_2, \ldots, r_t, \ldots$
  - $r_t = p$ for page p at time t
  - b_p = probability that $r_{t+1}=p$
  - Time between references of p: $I_p = 1/b_p$

LRU-K Algorithm

- Backward K-distance $b_t(p,K)$:
  - #references from t back to the Kth most recent reference to p
- $b_t(p,K)$ = INF if Kth reference doesn't exist
- Algorithm:
  - Drop page p with max Backward K-distance $b_t(p,K)$
- Ambiguous when infinite (use subsidiary policy, e.g., LRU)
- LRU-2 is better than LRU-1
  - Why?

Problem 1

- Early page replacement
  - Page $b_t(p,K)$ is infinite, so drop
  - What if it is a rare but "bursty" case?
    - Introduce a time-out period, say 5 sec
  - What if there are Correlated References?
    1. *Intra-transaction*, e.g., read tuple, followed by update
    2. Transaction Retry
    3. *Intra-process*, i.e., a process references page via 2 transactions, e.g., update RIDs 1-10, commit, update 11-20, commit, …
    - In contrast, Uncorrelated References are *inter-process* pairs of references, i.e., two processes reference the same page independently

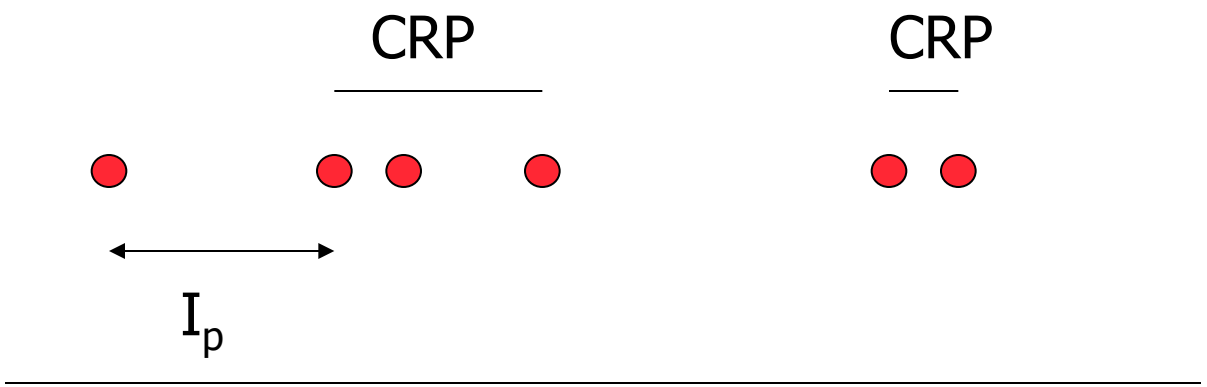
Example

- With or without timeouts, invalid conclusions may result, e.g., assume the intra-transaction correlated reference pair – read/update
  - Without
    - Algorithm sees p (read)
    - Drops it (infinite $b_t(p,K)$) (wrong decision)
  - With
    - Algorithm sees p (read)
    - Sees it again before timeout (update)
    - Keeps it around (wrong decision again)

- Timeout should still be used
  - But inter-arrival time should be computed based on uncorrelated access

Addressing Correlation

- Correlated Reference Period (CRP)
  - No penalty or credit for refs within CRP
  - I_p: interval from end of one CRP to begin of the next



Problem 2

- Reference Retained Information
  - Algorithm needs to keep info for pages that may not be resident anymore. Consider the following LRU-2 example where p is referenced periodically, but in periods longer than t:
    - p is referenced and comes in for the first time
    - $b_t(p,2)$ = INF, p is dropped
    - p is referenced again
    - If no info on p is retained, p may be dropped again, then referenced again, then dropped again etc.
    - Though the page is frequently referenced, we would have no history about it to recognize this fact

Solution to Problem 2

- Retained Information Period (RIP)
  - Period after which we drop information about page p
    - If a page p that has never been referenced before, suddenly becomes popular enough to be kept in buffer, we should recognize this fact as long as two references to the page are no more than the RIP apart
  - Upper bound for RIP: max Backward K-distance of all pages we want to ensure to be memory resident

Data Structures and Algorithm for LRU-K

- HIST(p) – history control block of page p = Time of K most recent references to p - correlated
- LAST(p) – time of most recent ref to page p, correlated references OK
- Maintained for all pages p: $b_t(p,K) < RIP$
- Algorithm

```
If p is in the buffer { // update history of p
  if (t - LAST(p)) > CRP { // uncorrelated ref
      // close correlated period and start new
      for i = K-1 to 1
            move HIST(p,i) into slot HIST(p,i
  +1)
      HIST(p,1) = t
  }
  LAST(p) = t
}
```

LRU-K Algorithm (Cont)

```
else { // select replacement victim
  min = t
  for all pages q in buffer {
      if (t - LAST(p) > CRP // eligible for replacement
            and HIST(q,K) < min { // max Backward-K
            victim = q
            min = HIST(q,K)
      }
  if victim dirty, write back before dropping
Fetch p into the victim's buffer
  if no HIST(p) exists {
      allocate HIST(p)
      for i = 2 to K HIST(p,i) = 0
  } else {
      for i = 2 to K HIST(p,i) = HIST(p,i-1)
  }
  HIST(p,1) = t // last non-correlated reference
  LAST(p) = t // last reference
}
```

Domain Separation

- Classify pages as types
  - Each type has a domain of buffers
- LRU within domain
- Example: B+-tree index
  - One domain per index level
  - One domain per leaf/data pages
- Problems
  - Static domains, i.e., dynamics of page references may vary in different queries
  - Does not differentiate the relative importance between different types of pages
- Extensions
  - Priority ranking for domains to find free pages
  - Dynamically vary domain sizes

"New" Algorithm in INGRES

- Each relation needs a working set
  - Buffer pool is subdivided and allocated on a per-relation basis
  - Each active relation is assigned a resident set which is initially empty
  - The resident sets are linked in a priority list; unlikely reused relations are near the top
  - Ordering of relation is pre-determined, and may be adjusted subsequently
  - Search from top of the list
  - With each relation, use MRU
- Pros
  - A new approach that tracks the locality of a query through relations
- Cons
  - MRU is not always good
  - How to determine priority (especially in multi-user context)?
  - Costly search of list under high loads

Hot-Set Model

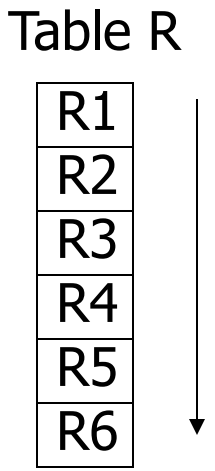
- Hot set: set of pages over which there is a looping behavior
  - Hot set in memory implies efficient query processing
  - #page faults vs size of buffers – points of discontinuities called hot points
- Key ideas
  - Give query |hot set| pages
  - Allow <=1 deficient query to execute
  - Hot set size computed by query optimizer (provides more accurate reference pattern)
  - User LRU within each partition
- Problems
  - LRU not always best and allocate more memory
  - Over-allocates pages for some phases of query

DBMIN: DBMS Reference Patterns

- Based on "Query Locality Set Model"
  - DBMS supports a limited set of operations
  - Reference patterns exhibited are regular and predictable
  - Complex patterns can be decomposed into simple ones
- Reference pattern classification
  - Sequential
    - Straight sequential (SS)
    - Clustered sequential (CS)
    - Looping sequential (LS)
  - Random
    - Independent random (IR)
    - Clustered random (CR)
  - Hierarchical
    - Straight hierarchical (SH)
    - Hierarchical with straight sequential (H/SS)
    - Hierarchical with clustered sequential (H/CS)
    - Looping hierarchical (LH)

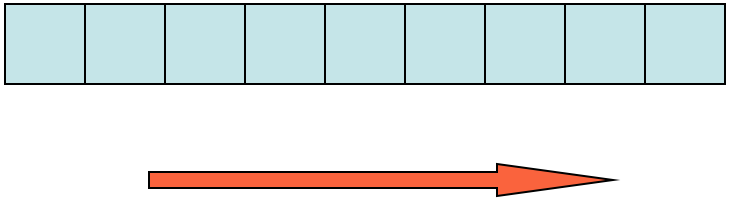
Sequential Patterns

- Straight sequential (SS)
  - File scan without repetition
    - E.g., selection on an unordered relation



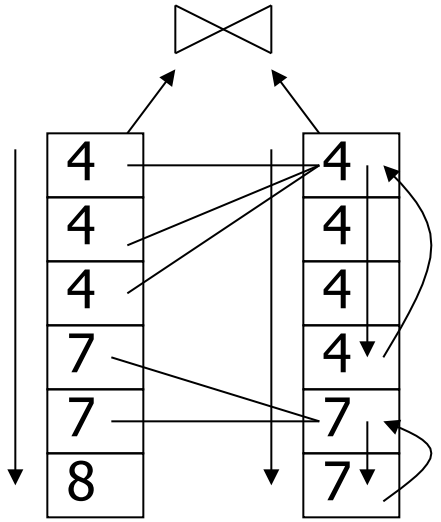
  - #pages? 1
  - Replacement algorithm?

    Replaced with next one



Sequential Patterns (Cont)

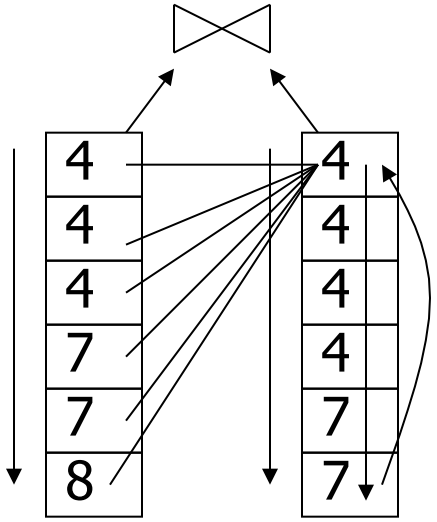
- Clustered sequential (CS)
  - Like inner S for merge-join (sequential with backup)
  - Local rescan in SS
  - Join condition: R.a = S.a



  - #Pages? #pages in largest cluster
  - Replacement algo? FIFO/LRU

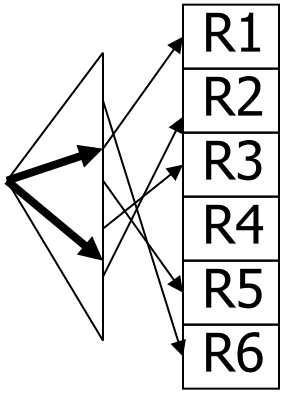
Sequential Patterns (Cont)

- Looping sequential (LS)
  - Sequential reference be repeated several times
    - e.g., Like inner S for nested-loop-join
  - #Pages? As many as possible
  - Replacement algo? MRU



Random Pattterns

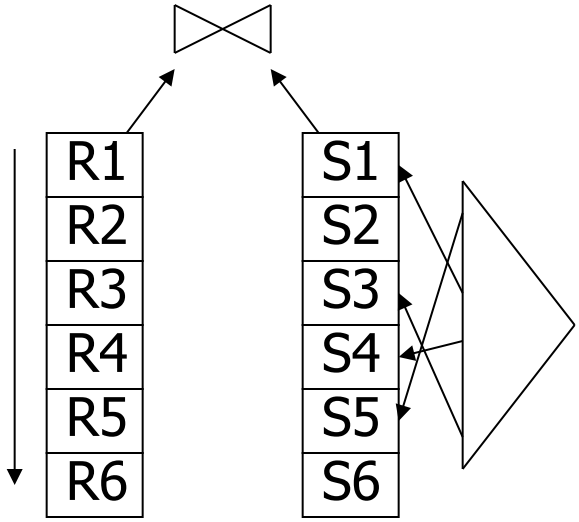
- Independent Random (IR)
  - Genuinely random accesses
    - e.g., non-clustered index scan



  - #Pages? 1 page (assuming low prob. of reaccesses)
  - Replacement algo? Any!

Random Patterns

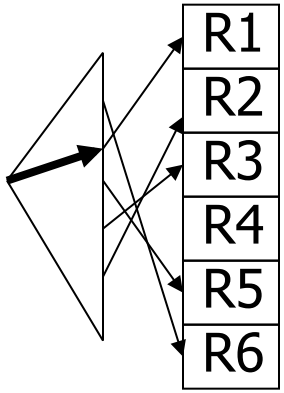
- Clustered Random (CR)
  - Random accesses which demonstrate locality
    - e.g., join with inner, non-clustered, non-unique index on join column



  - #Pages? #pages in largest cluster
  - Replacement algo? FIFO/LRU

Hierarchical Patterns

- Straight Hierarchical (SH)
  - Access index pages ONCE (retrieve a single tuple)
    - Like SS
  - Followed by straight sequential scan (H/SS)
    - Like SS
  - Followed by clustered scan (H/CS)
    - Like CS



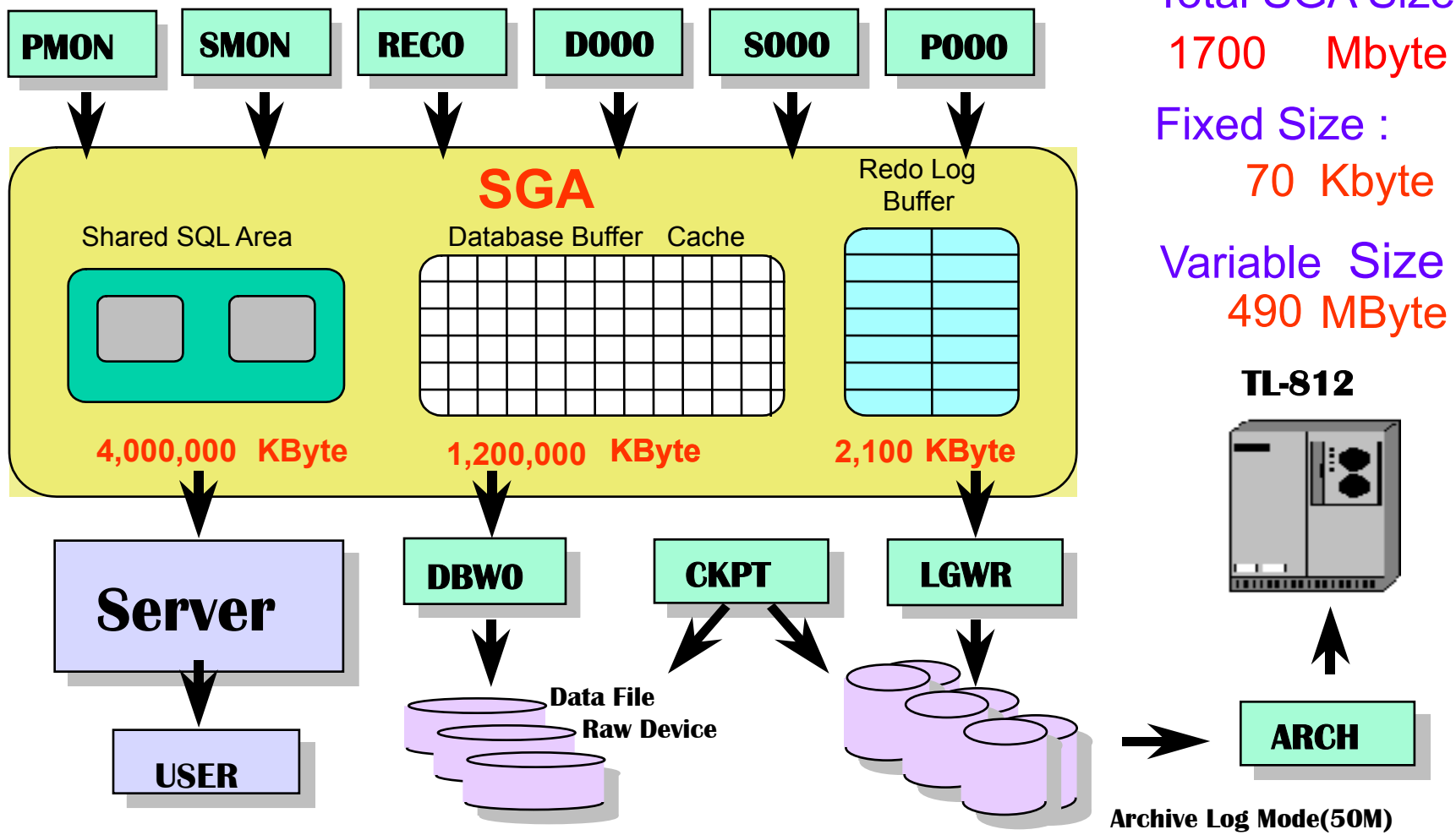
Hierarchical Patterns

- Looping Hierarchical (LH)
  - Repeatedly traverse an index, e.g., when inner index in join is repeatedly accessed
  - #Pages? Height of tree
  - Replacement algo? LIFO (to keep the root)

What's Implemented in DBMS?

- DB2 & Sybase ASE
  - Named pools to be bound to tables or indexes
  - Each pool can be configured to use clock replacement or LRU (ASE)
  - Client can indicate pages to replace
- Oracle
  - A table can be bound to 1 to 2 pools, one with higher priority to be kept
- Others
  - Global pools with simple policies

Overview of Oracle Architecture

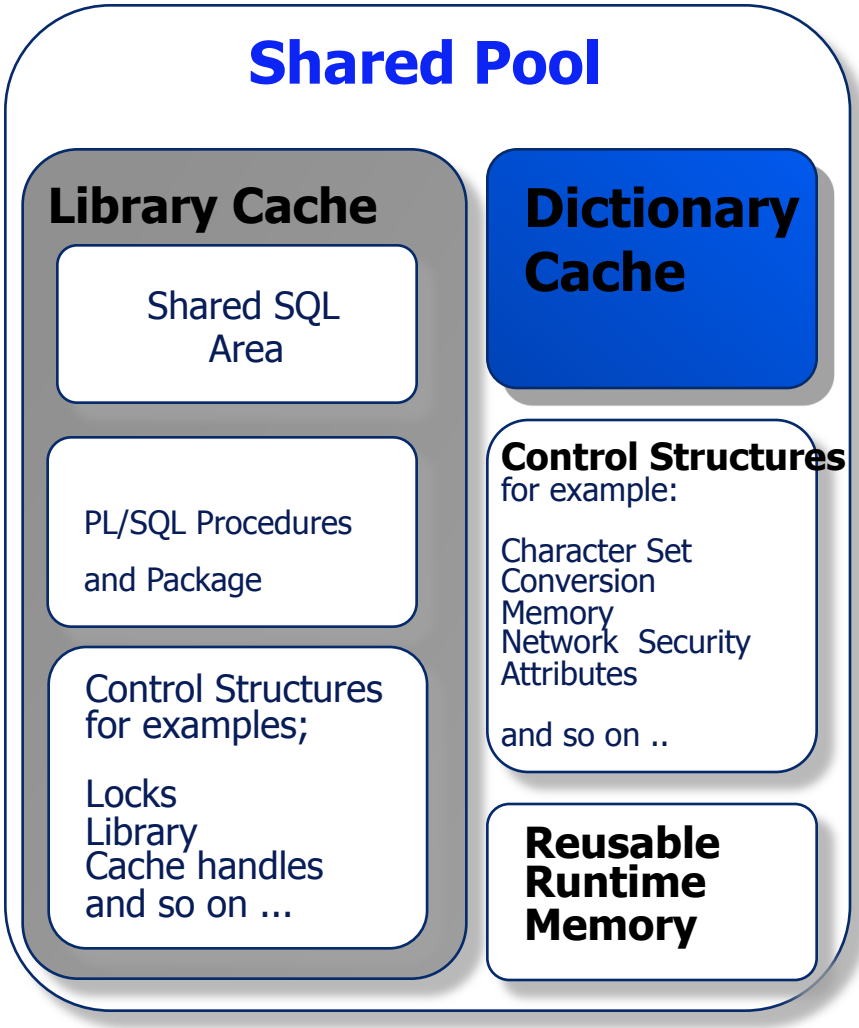


Total SGA Size :
1700 Mbyte

Fixed Size :
70 Kbyte

Variable Size :
490 MByte

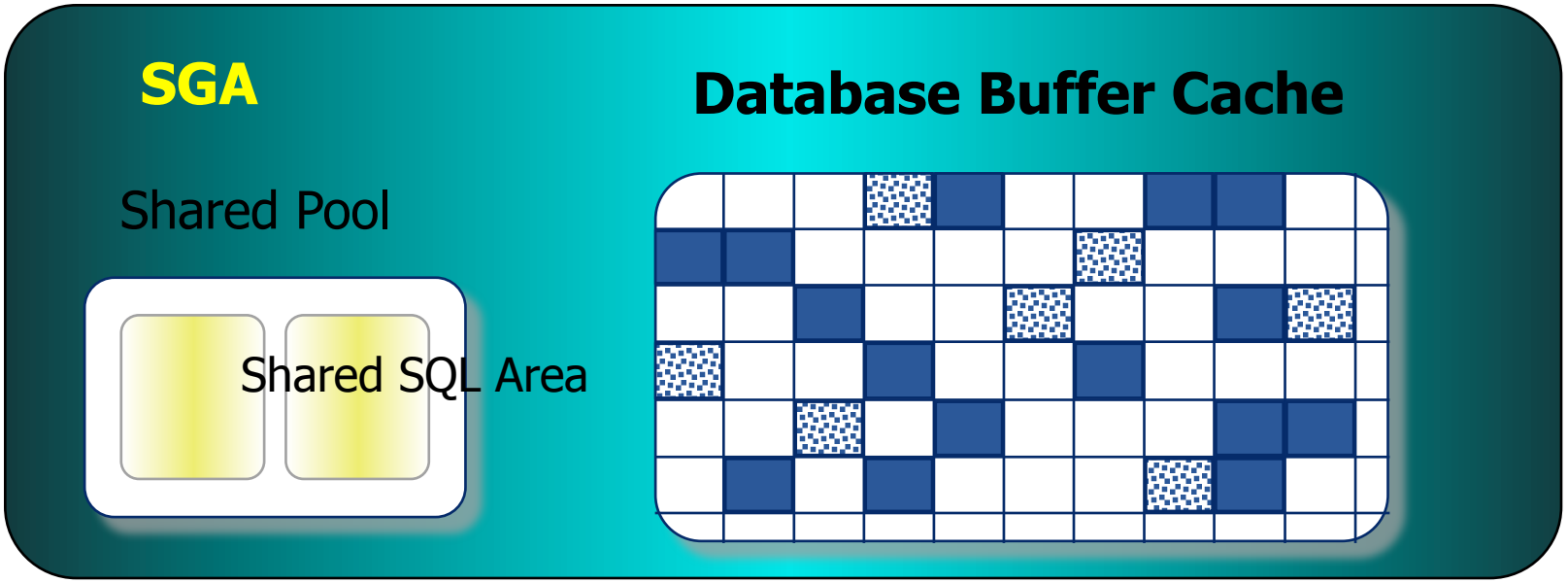
Memory Structure : Shared Pool



- Shared Pool Contents
  - Text of the SQL or PL/SQL statement
  - Parsed form of the SQL or PL/SQL statement
  - Execution plan for the SQL or PL/SQL statements
  - Data dictionary cache containing rows of data dictionary information
- Library Cache
  - shared SQL area
  - private SQL area
  - PL/SQL procedures and package
  - control structures : lock and library cache handles
- Dictionary Cache
  - names of all tables and views in the database
  - names and datatypes of columns in database tables
  - privileges of all Oracle users
- SHARED_POOL_SIZE

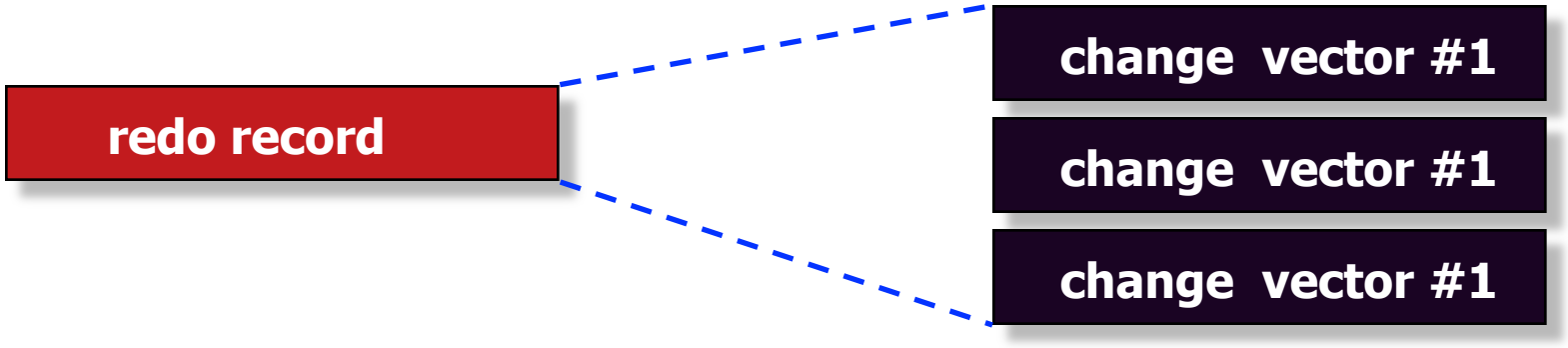
Memory Structure :Database Buffer Cache

- Database Buffer Cache holds copies of data blocks read from disk
- All users concurrently connected to the system share access to the buff er cache
- Dirty List
- LRU List
- `Size = DB_BLOCK_SIZE * DB_BLOCK_BUFFERS`



Memory Structure :Redo Log Buffer

- Circular buffer containing information about changes made to the databa se
- Saves Redo entries
- Redo entries are used in Database Recovery
- DBWR writes contents of Redo Log Buffer to Online Redo Log
- LOG_BUFFER



Overall Cache Hit Ratio

- DBMS buffer tuning (Oracle 9i)
- Cache hit ratio = (# logical read - # physical read) / # logical read
- Ideally, hit ratio > 80%
- Overall buffer cache hit ratio for entire instance:

```
SELECT (P1.value+P2.value-P3.value)/(P1.value+P2.value)
FROM v$sysstat P1, v$sysstat P2, v$sysstat P3
WHERE P1.name = 'db block gets‘
AND P2.name = 'consistent gets‘
AND P3.name = 'physical reads'
```

Session Cache Hit Ratio

- Buffer cache hit ratio for one specific session:

```
SELECT (P1.value+P2.value-P3.value) / (P1.value+P2.value)
FROM v$sesstat P1, v$statname N1, v$sesstat P2,
     v$statname N2, v$sesstat P3, v$statname N3
WHERE N1.name = 'db block gets'
AND P1.statistic# = N1.statistic#
AND P1.sid = <enter SID of session here>
AND N2.name = 'consistent gets'
AND P2.statistic# = N2.statistic#
AND P2.sid = P1.sid
AND N3.name = 'physical reads'
AND P3.statistic# = N3.statistic#
AND P3.sid = P1.sid
```

Adjust Buffer Cache Size

- `Buffer size = db_block_buffers * db_block_size`
  - `db_block_size` is set at database creation; cannot tune
  - Change the `db_block_buffers` parameter

Should Buffer Cache Be Larger?

- Set `db_block_lru_extended_statistics` to 1000

```
SELECT 250 * TRUNC (rownum / 250) + 1 || ' to ' ||
               250 * (TRUNC (rownum / 250) + 1) "Interval",
               SUM (count) "Buffer Cache Hits“
FROM v$recent_bucket
GROUP BY TRUNC (rownum / 250)

Interval        Buffer Cache Hits
--------------- ----------------------
1 to 250                     16083
251 to 500                   11422
501 to 750                     683
751 to 1000                    177
```

- Incurs overhead! Set back to 0 when done

Should Buffer Cache Be Smaller?

- Set `db_block_lru_statistics` to true

```
SELECT 1000 * TRUNC (rownum / 1000) + 1 || ' to ' ||
              1000 * (TRUNC (rownum / 1000) + 1) "Interval",
              SUM (count) "Buffer Cache Hits“
FROM v$current_bucket
WHERE rownum > 0
GROUP BY TRUNC (rownum / 1000)

Interval   Buffer Cache Hits
---------- ----------------------
1 to 1000              668415
1001 to 2000           281760
2001 to 3000           166940
3001 to 4000            14770
4001 to 5000             7030
5001 to 6000              959
```

I/O Intensive SQL Statements

- `v$sqlarea` contains one row for each SQL statement currently in the system global area

```
SELECT executions, buffer_gets, disk_reads,
       first_load_time, sql_text
FROM v$sqlarea
ORDER BY disk_reads
```

- Executions: # times the statement has been executed since entering SGA
- `Buffer_gets`: total # logical reads by all executions of the statement
- `Disk_reads`: total # physical reads by all executions of the statement

Swapping of Data Pages

- Monitoring tools: `sar` or `vmstat`
- If system is swapping
  - Remove unnecessary system daemons and applications
  - Decrease number of database buffers
  - Decrease number of UNIX file buffers

Paging of Program Blocks

- Monitoring tools: `sar` or `vmstat`
- To reduce paging
  - Install more memory
  - Move some programs to another machine
  - Configure SGA to use less memory
- Compare paging activities during fast versus slow response

SAR – Monitoring Tool

- `vmstat -S 5 8`

procs			memory				page					disk				faults		cpu			
r	b	w	swap	free	si	so	pi	po	fr	de	sr	f0	s0	s1	s3	in	sy	cs	us	sy	id
0	0	0	1892	5864	0	0	0	0	0	0	0	0	0	0	0	90	74	24	0	0	99
0	0	0	85356	8372	0	0	0	0	0	0	0	0	0	0	0	46	25	21	0	0	100
0	0	0	85356	8372	0	0	0	0	0	0	0	0	0	0	0	47	20	18	0	0	100
0	0	0	85356	8372	0	0	0	0	0	0	0	0	0	0	2	53	22	20	0	0	100
0	0	0	85356	8372	0	0	0	0	0	0	0	0	0	0	0	87	23	21	0	0	100
0	0	0	85356	8372	0	0	0	0	0	0	0	0	0	0	0	48	41	23	0	0	100
0	0	0	85356	8372	0	0	0	0	0	0	0	0	0	0	0	44	20	18	0	0	100
0	0	0	85356	8372	0	0	0	0	0	0	0	0	0	0	0	51	71	24	0	0	100

1 = swapped out processes

swap-in, swap-out per sec

page-in, page-out per sec

Example Buffer Replacement Policies

- Assume a page reference pattern, which executes 3 consecutive scans in a set of five (disk) pages
- Assume that you begin with an empty pool of 3 frames

1. Calculate the following:
   - #page faults if LRU is used
   - #page faults if MRU is used
   - #page faults if Clock algorithm is used
2. What happens if the buffer (frame) pool is not empty (for LRU and MRU)? For example, assume that the first frame buffer is not empty
3. Repeat first part with an arbitrary random page reference pattern
4. Is LRU approximated by the Clock Algorithm?

Consecutive Scans with Empty Pool

- LRU (15 page faults)

Frames V	Read Page	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1		1	1	1	4	4	4	2	2	2	5	5	5	3	3	3
2			2	2	2	5	5	5	3	3	3	1	1	1	4	4
3				3	3	3	1	1	1	4	4	4	2	2	2	5
Page fault?		y	y	y	y	y	y	y	y	y	y	y	y	y	y	y

- MRU (9 page faults)

Frames V	Read Page	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1		1	1	1	1	1	1	1	1	1	1	1	2	3	3	3
2			2	2	2	2	2	2	3	4	4	4	4	4	4	4
3				3	4	5	5	5	5	5	5	5	5	5	5	5
Page fault?		y	y	y	y	y	n	n	y	y	n	n	y	y	n	n

- Clock algorithm (15 page faults)

Frames V	Read Page	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1		1 1 -	1 1 -	1 1 -	4 1	4 1	4 1 -	2 1	2 1	2 1 -	5 1	5 1	5 1 -	3 1	3 1	3 1 -
2			2 1	2 1	2 0 -	5 1	5 1	5 0 -	3 1	3 1	3 0 -	1 1	1 1	1 0 -	4 1	4 1
3				3 1	3 0	3 0 -	1 1	1 0	1 0 -	4 1	4 0	4 0 -	2 1	2 0	2 0 -	5 1
Page fault?		y	y	y	y	y	y	y	y	y	y	y	y	y	y	y

Consecutive Scans with Non-Empty Buffer Pool

- We consider two cases:
  - Case 1: First frame buffer does not contain any of the five pages we have to scan
  - Case 2: First frame buffer contains one of the five pages we have to scan. Assume it contains the first one

Case 1: First Frame Contains Page X

- LRU (15 page faults)

Frames\ Page Read	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1	X	X	3	3	3	1	1	1	4	4	4	2	2	2	2
2	1	1	1	4	4	4	2	2	2	5	5	5	3	3	3
3		2	2	2	5	5	5	3	3	3	1	1	1	4	4
Page fault?(y or n)	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y

- MRU (12 page faults)

Frames\ Page Read	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
2	1	1	1	1	1	1	2	3	4	4	4	4	4	4	5
3		2	3	4	5	5	5	5	5	5	1	2	3	3	3
Page fault? (y or n)	y	y	y	y	y	n	y	y	y	n	y	y	y	n	y

Case 2: First Frame Contains Page 1

- LRU (14 page faults)

Frames\ Page Read	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1	1	1	1	4	4	4	2	2	2	5	5	5	3	3	3
2		2	2	2	5	5	5	3	3	3	1	1	1	4	4
3			3	3	3	1	1	1	4	4	4	2	2	2	5
Page fault?(y or n)	n	y	y	y	y	y	y	y	y	y	y	y	y	y	y

- MRU (8 page faults)

Frames\ Page Read	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	1	1	2	3	3	3
2		2	2	2	2	2	2	3	4	4	4	4	4	4	4
3			3	4	5	5	5	5	5	5	5	5	5	5	5
Page fault? (y or n)	n	y	y	y	y	n	n	y	y	n	n	y	y	n	n

Random Page Reference Pattern

- We consider the following arbitrary and random page reference pattern: [2,4,1,2,4,3,5,3,2,1,1,4,4,5,3]
- LRU (10 page faults)

Frames\ Page Read	2	4	1	2	4	3	5	3	2	1	1	4	4	5	3
1	2	2	2	2	2	2	5	5	5	1	1	1	1	1	3
2		4	4	4	4	4	4	4	2	2	2	2	2	5	5
3			1	1	1	3	3	3	3	3	3	4	4	4	4
Page fault? (y or n)	y	y	y	n	n	y	y	n	y	y	n	y	n	y	y

Random Page Reference Pattern

- MRU (8 page faults)

Frames\ Page Read	2	4	1	2	4	3	5	3	2	1	1	4	4	5	3
1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2		4	4	4	4	3	5	3	3	3	3	3	3	3	3
3			1	1	1	1	1	1	1	1	1	4	4	5	5
Page fault? (y or n)	y	y	y	n	n	y	y	y	n	n	n	y	n	y	n

- Clock algorithm (10 page faults)

Frames\ Page Read	2	4	1	2	4	3	5	3	2	1	1	4	4	5	3
1	2 1 -	2 1 -	2 1 -	2 1	2 1	2 0 -	5 1	5 1 -	5 0	1 1	1 1	1 1 -	1 1 -	1 0	3 1
2		4 1	4 1	4 1 -	4 1	4 0	4 0 -	4 0	2 1	2 1 -	2 1 -	2 0	2 0	5 1	5 1 -
3			1 1	1 1	1 1 -	3 1	3 1	3 1	3 1 -	3 0	3 0	4 1	4 1	4 1 -	4 0
Page fault? (y or n)	y	y	y	n	n	y	y	n	y	y	n	y	n	y	y

LRU Approximated by Clock Algorithm?

- Yes, considering the previous example, they have the same behavior and performance
- From theoretical perspective, the Clock algorithm is a cheap implementation of LRU
  - This leads to the Clock being adopted instead of LRU most times

Summary

- Monitor cache hit ratio
- Increase/reduce buffer cache size
- Pay attention to I/O intensive SQL statements
- Avoid swapping
- Check for excessive paging

Τέλος Ενότητας

Χρηματοδότηση

•Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.

•Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.

•Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημειώματα

Σημειώματα

Σημείωμα αδειοδότησης

.

Σημείωμα Αναφοράς