University of Southampton

Data Storage

COMP3211 Advanced Databases

Dr Nicholas Gibbins - nmg@ecs.soton.ac.uk

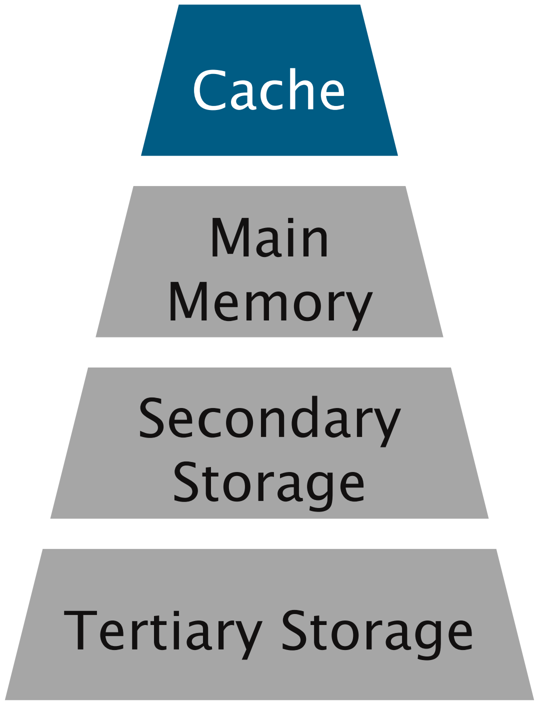
University of Southampton

Overview

- Storage Organisation
- Secondary storage
- Buffer management
- The Five-Minute Rule
- Disk Organisation
  - Data Items
  - Records
  - Blocks

Storage Organisation

The Memory Hierarchy



The Memory Hierarchy: Cache

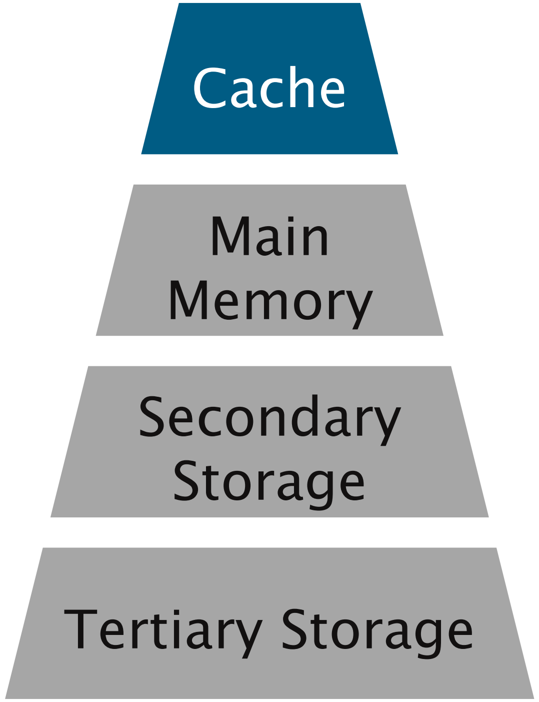
Volatile storage

Very fast, very expensive, limited capacity

Hierarchical

Typical capacities and access times:

- Registers – ~10^1 bytes, 1 cycle
- L1 – ~10^4 bytes, <5 cycles
- L2 – ~10^5 bytes, 5-10 cycles



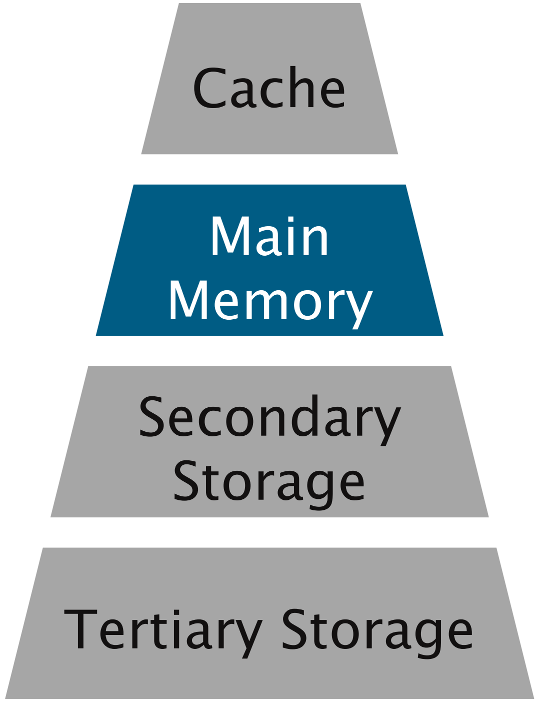
The Memory Hierarchy: Main Memory

Volatile storage

Fast, affordable, medium capacity

Typical capacity: 10^9-10^{10} bytes

Typical access time: 10^{-8} s (20-30 cycles)



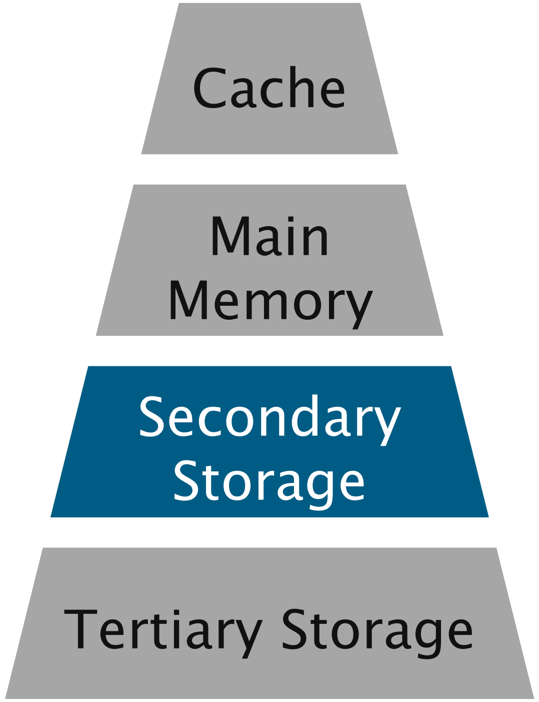
The Memory Hierarchy: Secondary Storage

Non-volatile storage

Slow, cheap, large capacity

Typical capacity: 10^{11}-10^{12} bytes

Typical access time: 10^{-3} s (10^{6} cycles)



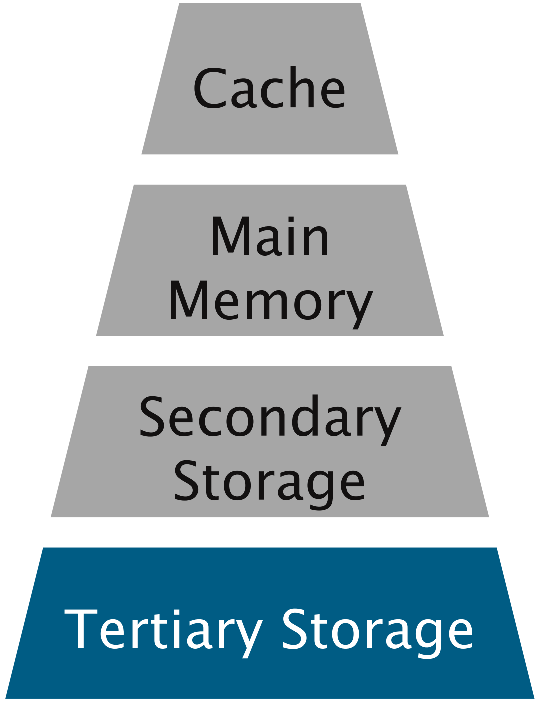
The Memory Hierarchy: Tertiary Storage

Non-volatile storage

Very slow, very cheap, very large capacity

Typical capacity: 10^{13}-10^{17} bytes

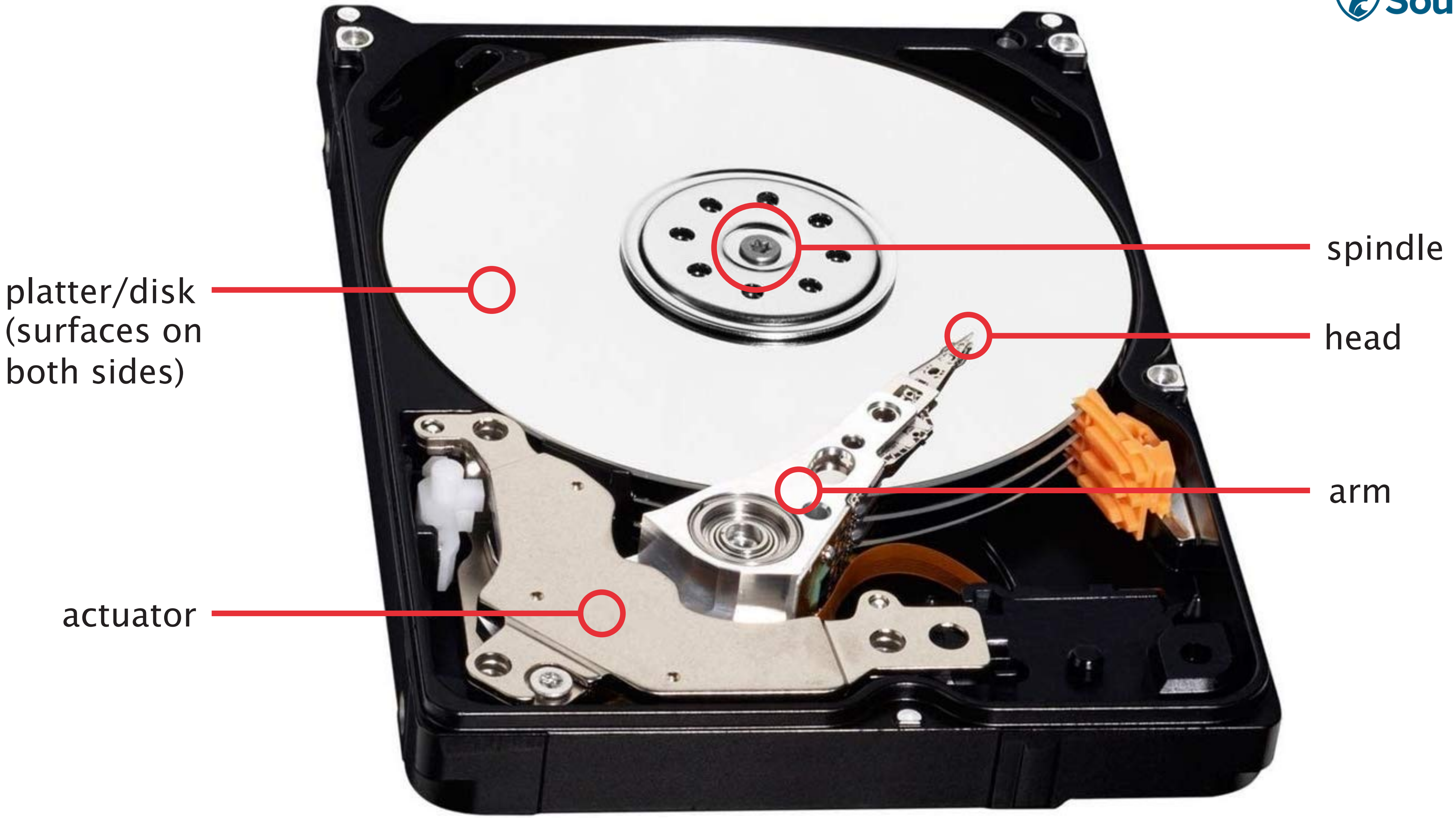
Typical access time: 10^{1}-10^{2} s



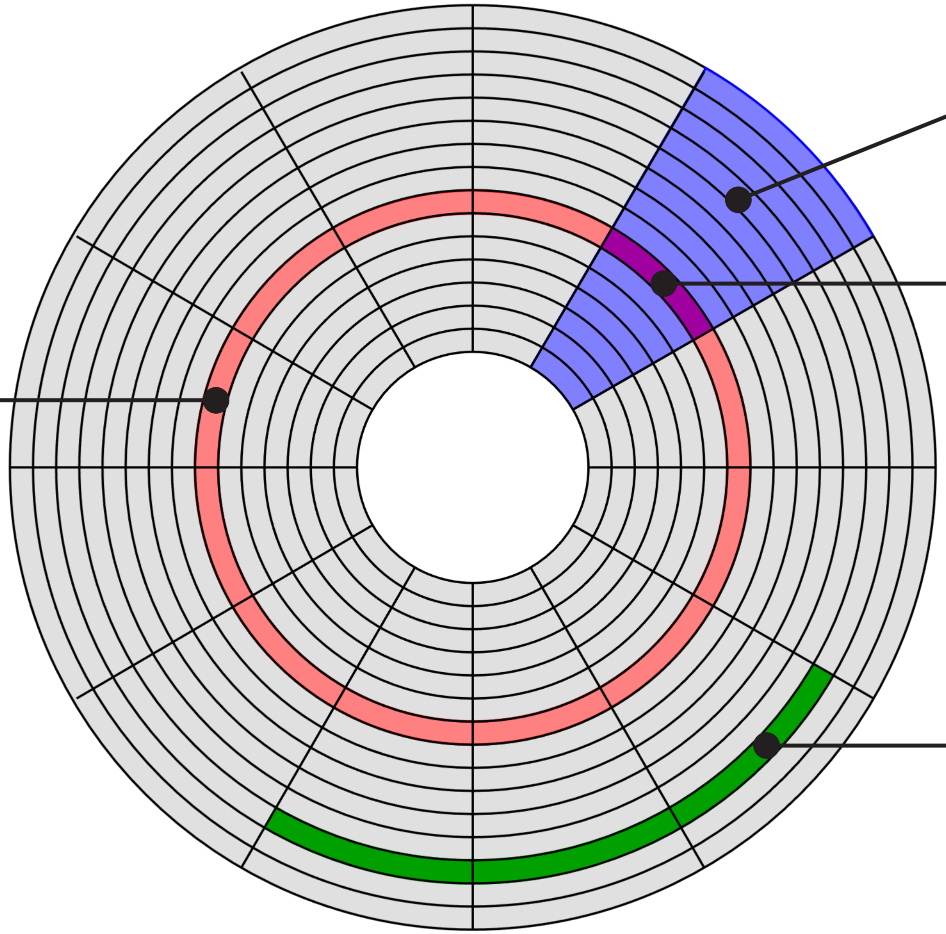
Secondary Storage

Hard Disk Drives

Typical secondary storage medium for databases


platter/disk
(surfaces on
both sides)
actuator
spindle
head
arm

Disk Structure



cylinder is the same
track on all surfaces

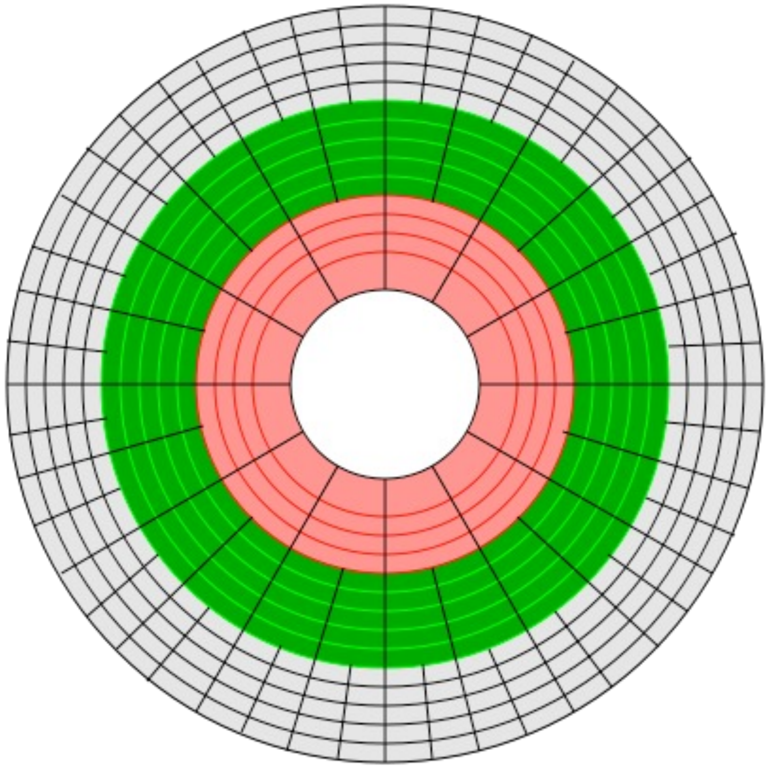
Zone Bit Recording

Tracks closer to the disc edge are longer than those closer to the axis

- Bit densities vary in order to ensure a constant number of bits per sector

Instead, we can vary the number of sectors per track (depending on track location)

- Improves overall storage density
- A hybrid of constant linear velocity (CLV) and constant angular velocity (CAV)



Disk Sector Format

Terms:

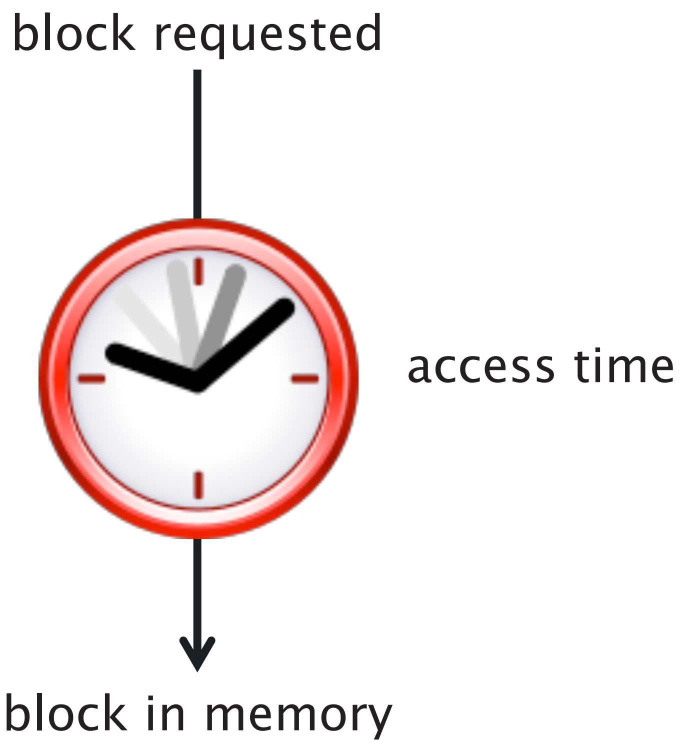
- Gap – separator between sectors
- Sync – indicates start of sector
- Address mark – indicates sector’s number/location
- ECC – error correcting code (may be distributed)

For 4k Advanced Format:

- gap+sync+mark = 15 bytes
- data = 4096 bytes
- ecc = 100 bytes
- 2.7% overhead

Disk Access Time: Reading

Access Time = Seek Time +
Rotational Delay +
Transfer Time

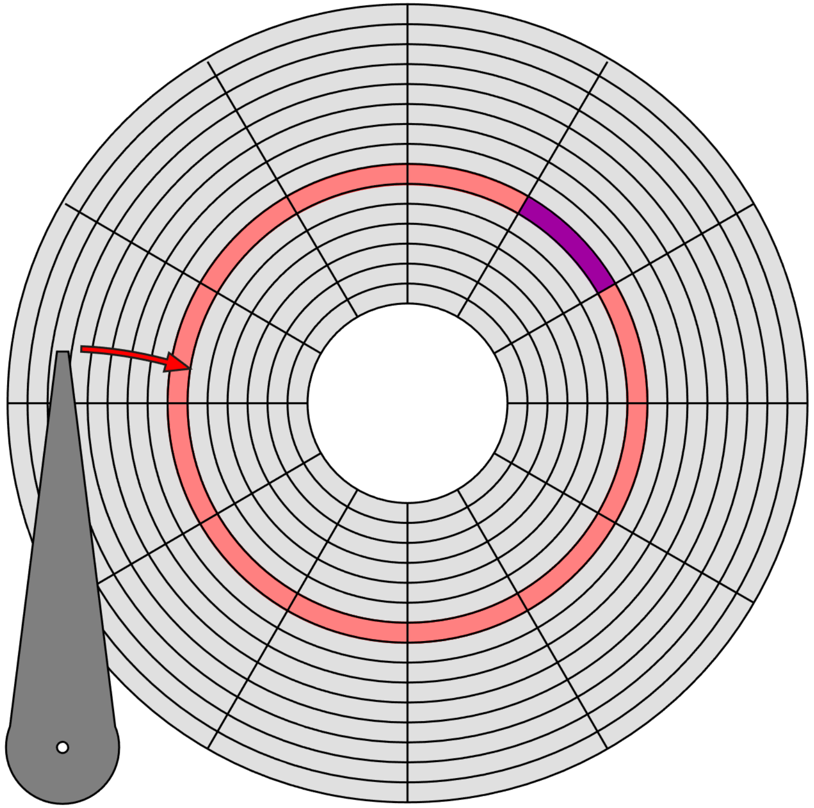


Seek Time

Time taken for head assembly to move to a given track

Average seek time range:

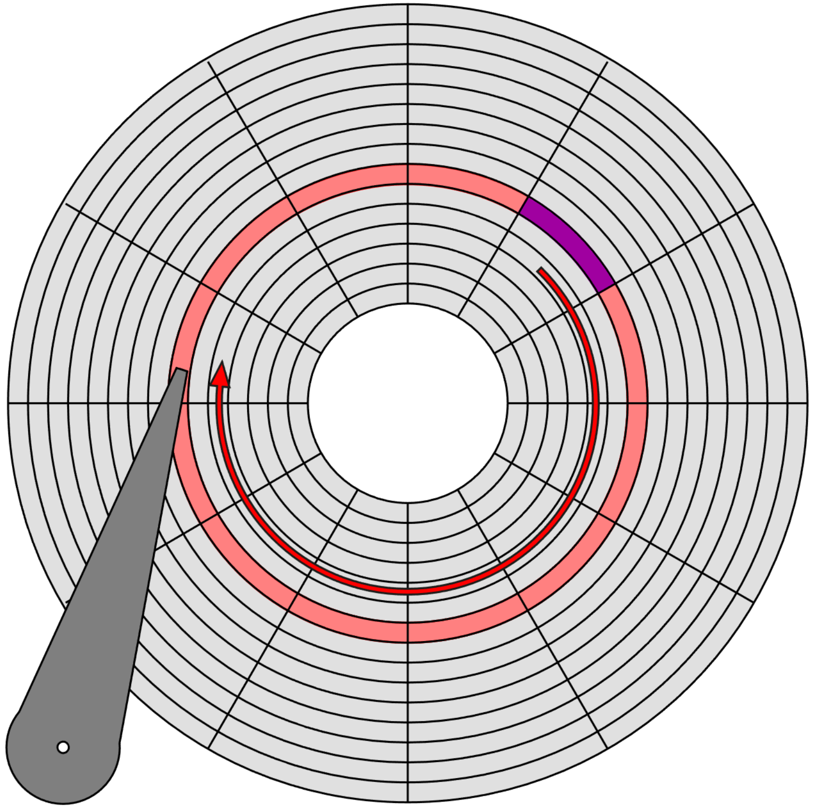
- 4ms for high end drives
- 15ms for mobile devices



Rotational Delay (Latency)

Average delay = time for 0.5 rev

rotational speed [rpm]	average delay [ms]
4,200	7.14
5,400	5.56
7,200	4.17
10,000	3.00
15,000	2.00



Transfer Time

Transfer rate ranges from:

- up to 1000 Mbit/sec
- 432 Mbit/sec 12x Blu-Ray disk
- 1.23 Mbits/sec 1x CD
- for SSDs, limited by interface
  e.g., SATA 3000 Mbit/s

Transfer time = block size / transfer rate

Sequential Access

So far, random access - what about reading “next” block?

Access time = (block size / transfer rate) + negligible costs

Negligible costs:

- skip inter-block gap
- switch track (within same cylinder)
- switch to adjacent cylinder occasionally

- In general, sequential i/o is much less expensive than random i/o

Disk Access Time: Writing

Costs similar to those for reading, unless we wish to verify data

Verifying requires that we read the block we've just written

Access Time = Seek Time +
Rotational Delay (1/2 rotation) +
Transfer Time (for writing) +
Rotational Delay (full rotation) +
Transfer Time (for verifying)

Disk Access Time: Modifying

1. Read Block
2. Modify in Memory
3. Write Block
4. Verify Block (optional)

Disk Access Time: Modifying

Access Time = Seek Time +
Rotational Delay (1/2 rotation) +
Transfer Time (for reading) +
Rotational Delay (full rotation) +
Transfer Time (for writing) +
[Rotational Delay (full rotation) +
Transfer Time (for verifying)
]

Block Addressing

Cylinder-head-sector

- Physical location of data on disk
- ZBR causes problems (sectors vary by tracks)

Logical Block Addressing

- Blocks located by integer index
- HDD firmware maps LBA addresses to physical locations on disk
- Allows remapping of bad blocks

Block Size Selection?

The size of blocks affects I/O efficiency:

Big blocks reduce the costs of access

- Fewer seeks (seek time + rotational delay) for the same amount of data

Big blocks also increase the amount of irrelevant data read

- If you're trying to read a single record in a block, larger blocks force you to read more data

But what about Solid State Drives?

Solid State Drives

- Typically based on NAND flash memory
- More expensive than HDD (~4-5x)
  - Getting cheaper over time
  - Global SSD production was expected to exceed HDD production in 2021
- Typically smaller maximum size than HDD (~1-2TB)
- Considerably higher I/O performance
- Asymmetric read/write performance (writes are slower)
- Limited number of program-erase cycles (~100,000 – wear levelling used)

HDD versus SSD

Random I/Os per second (IOPS) = 1/ (seek + latency + transfer)

	HDD *	SSD **
Random Read IOPS	125-150 IOPS	~50,000 IOPS
Random Write IOPS	125-150 IOPS	~40,000 IOPS

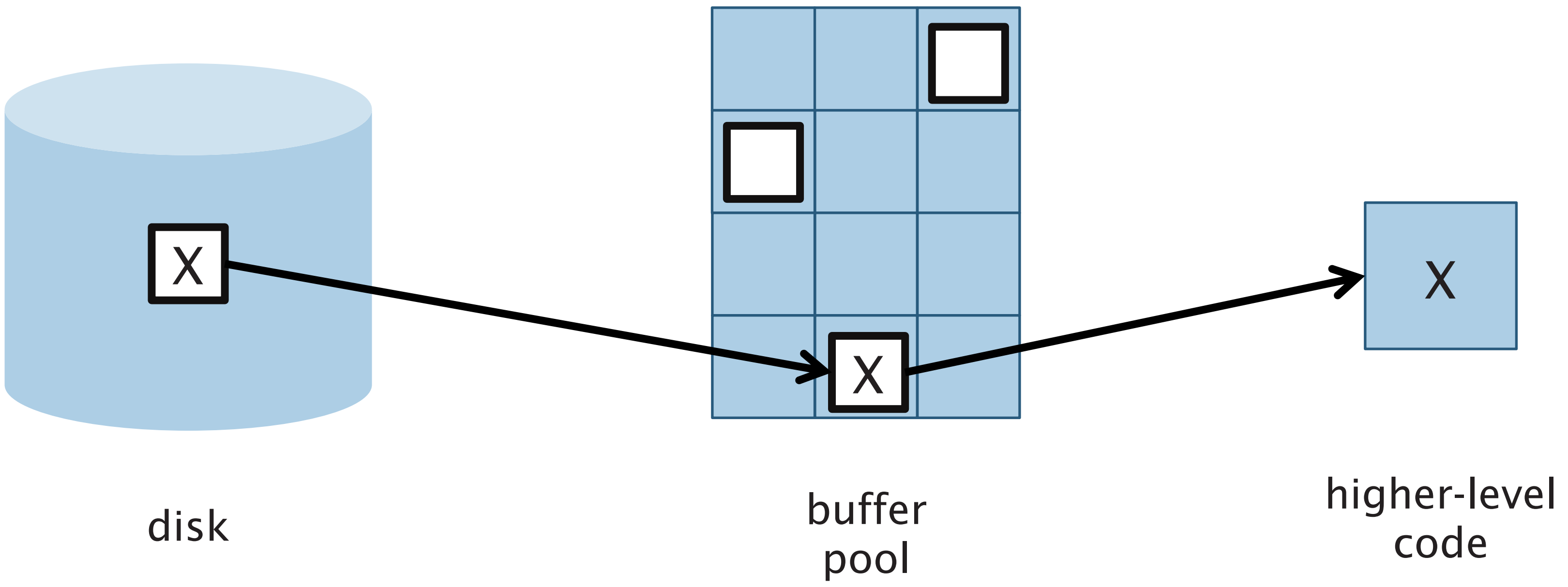
* Assumes 10,000 rpm HDD with SATA 3Gb/s interface
** OCZ 480GB Vertex 3 (c. 2012) with SATA 6Gb/s interface

Buffer Management

The Buffer Pool

Far more blocks of secondary storage than space in main memory – need to be selective about what's kept in memory

Buffer pool organised into frames (size of database block, plus metadata)



Buffer Metadata

Each frame in the buffer pool has:

- a *pin count* (number of current users of the block in that frame)
- a *dirty flag* (1 if the copy in the buffer has been changed, 0 otherwise)
- an *access time* (optional – used for LRU replacement)
- a *loading time* (optional – used for FIFO replacement)
- a *clock flag* (optional – used for Clock replacement)

University of Southampton

Requesting a Block

```
if      buffer pool already has a frame containing the block
then    increment pin count ("pin the block")
else    if      there is an empty frame
        then    read block into empty frame and set pin count to 1
        else    choose a frame to be replaced
                if      dirty bit on the replacement frame is set
                then    write block in replacement frame to disk
                endif
                read block into replacement frame and set pin count to 1
        endif
endif
```

Buffer Replacement Strategies

A frame will not be selected for replacement until its pin count is 0

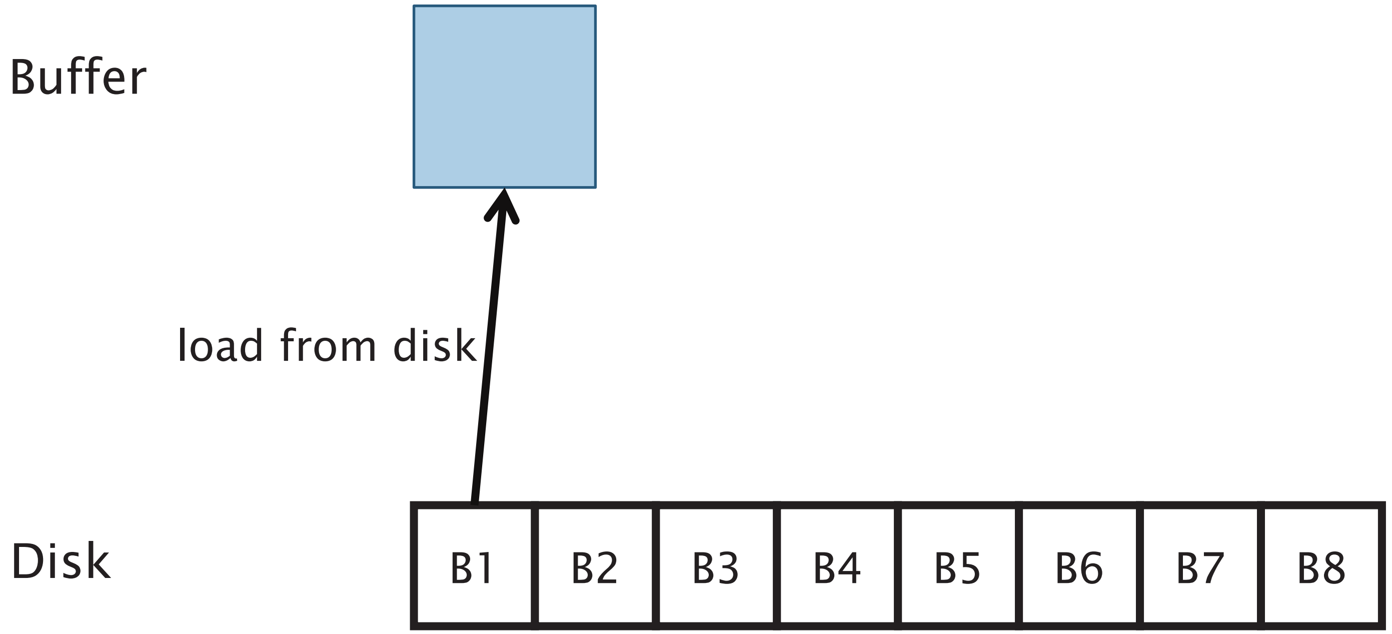
If there's more than one frame with a pin count of 0, use a *replacement strategy* to choose the frame to be replaced

- Least Recently Used (LRU)
  Select the frame with the oldest access time
- First In First Out (FIFO)
  Select the frame with the oldest loading time
- Clock
  Approximation of LRU – cycle through each buffer in turn, if a buffer hasn't been accessed in a full cycle then mark it for replacement

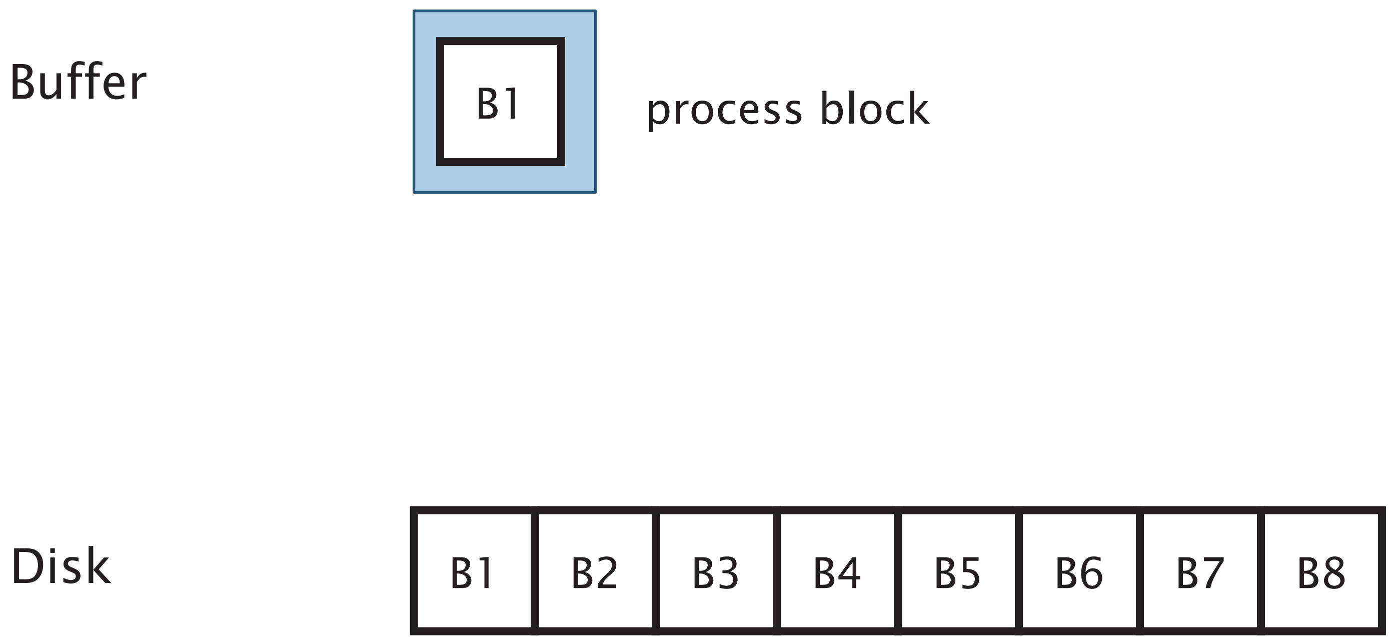
Single Buffering

1. Read B1 → Buffer
2. Process Data in Buffer
3. Read B2 → Buffer
4. Process Data in Buffer ...

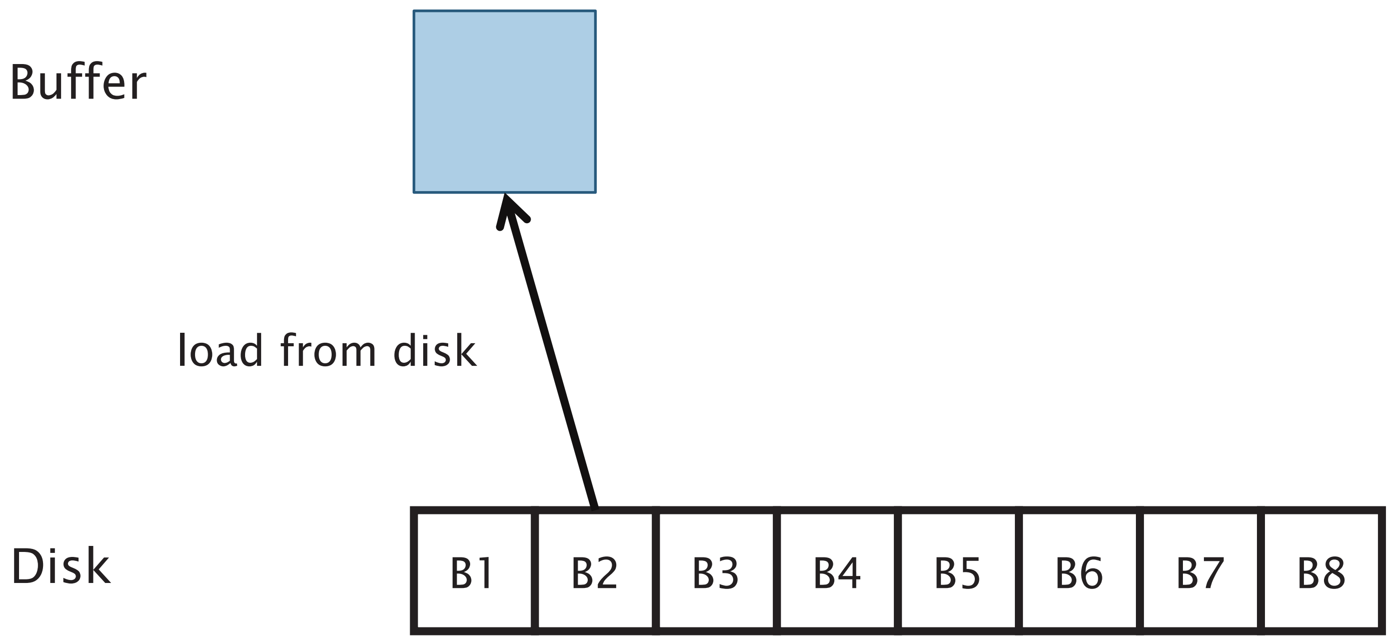
Single Buffering



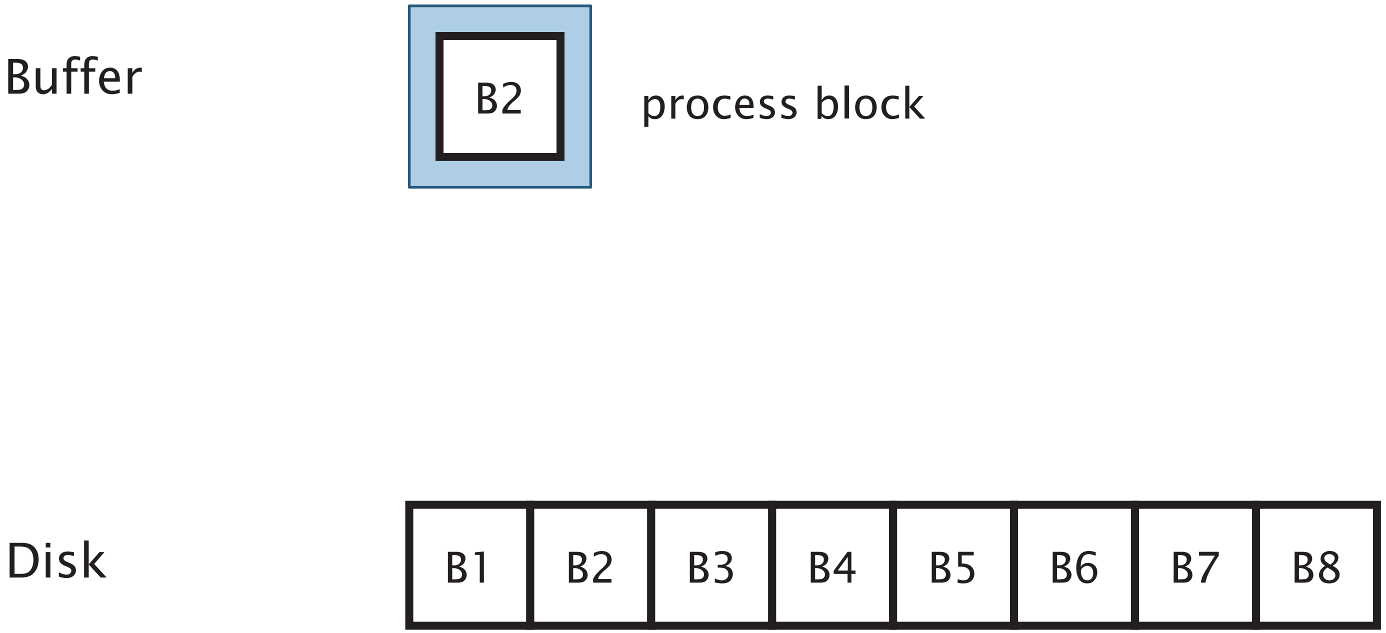
Single Buffering



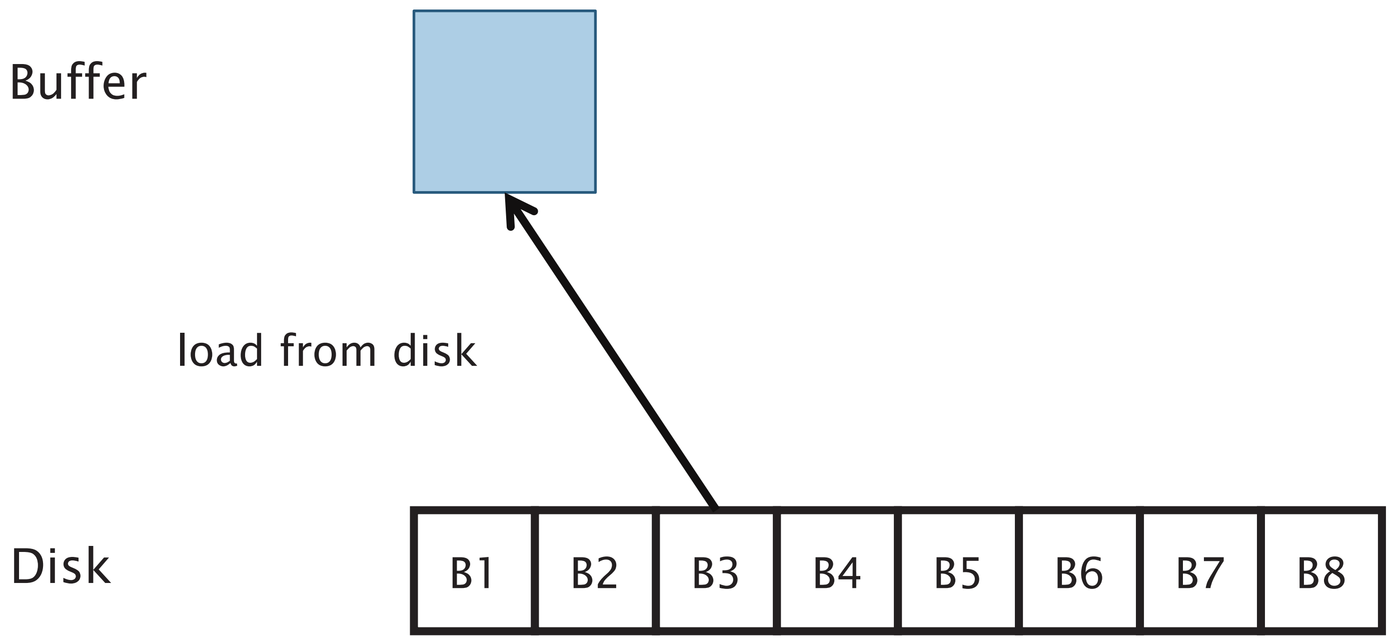
Single Buffering



Single Buffering



Single Buffering



Single Buffering Cost

Single buffer time = $n(P + R)$

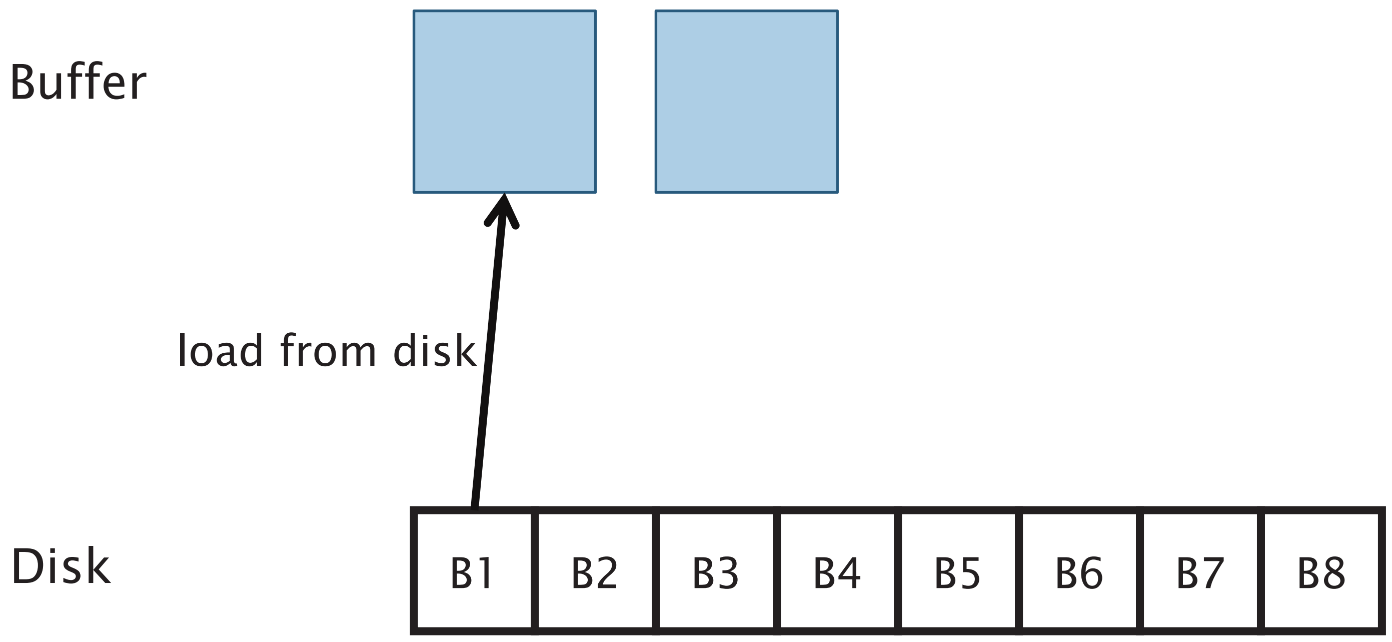
where P = time to process a block
R = time to read a block
n = number of blocks

Double Buffering

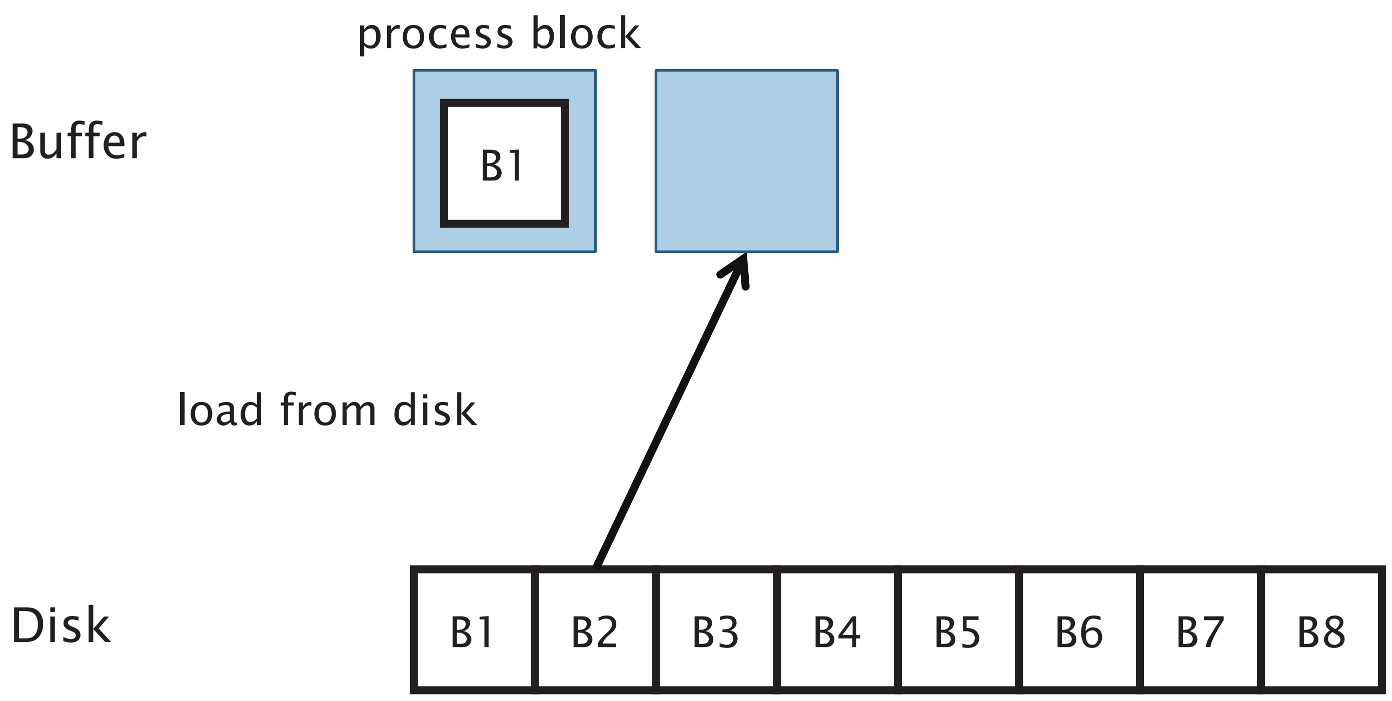
Use a pair of buffers:

- While reading a block and writing into buffer A
- Process block previously read into buffer B
- After block read into A, process A and read next block into B

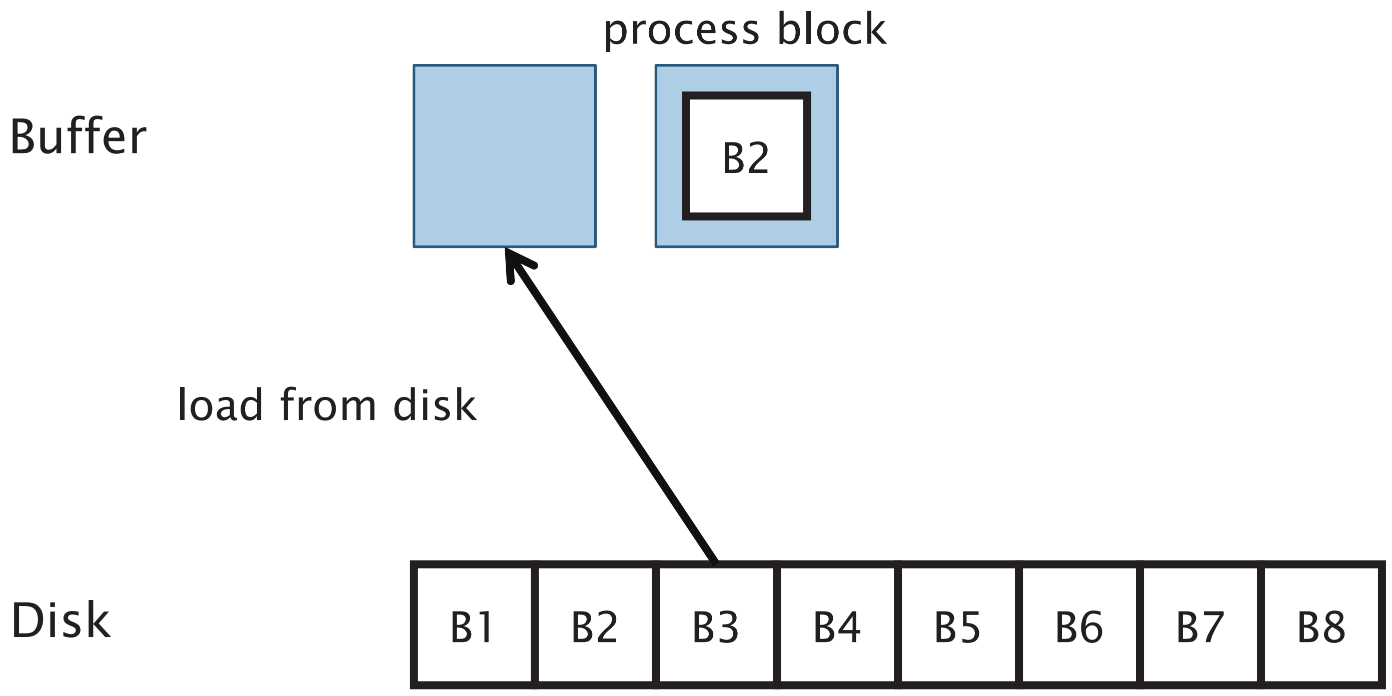
Double Buffering



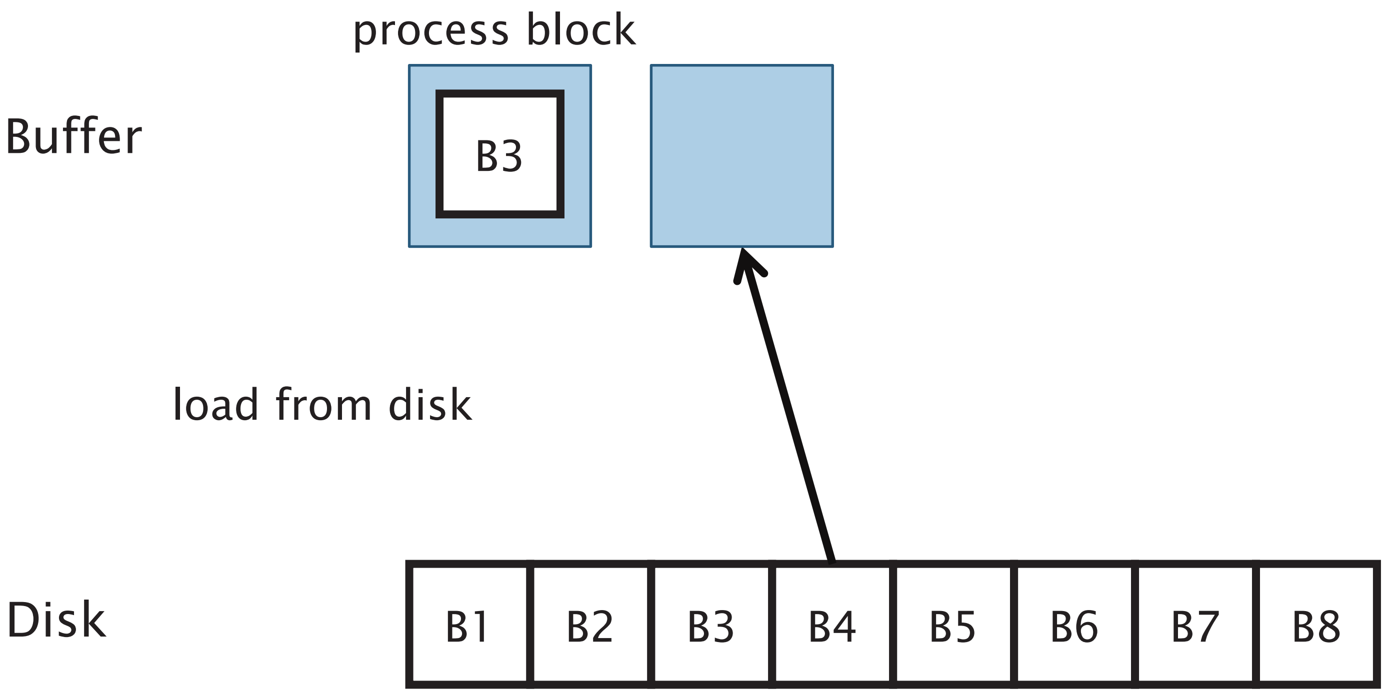
Double Buffering



Double Buffering



Double Buffering



Double Buffering

If time to process a block > time to read a block:

Double buffer time $= R + nP$

Single buffer time $= n(R+P)$

The Five Minute Rule

The Five Minute Rule

Data referenced every five minutes should be memory resident

The Five Minute Rule

The Five Minute Rule for trading memory for disc accesses
Jim Gray & Franco Putzolu
May 1985

The Five Minute Rule, Ten Years Later
Goetz Graefe & Jim Gray
December 1997

The five-minute rule 20 years later (and how flash memory changes the rules)
Goetz Graefe
July 2009

The Five-Minute Rule 30 years later, and its impact on the storage hierarchy
Raja Appuswamy, Goetz Graefe, Renata Borovica-Gajic and Anatasia Ailamaki
November 2019

The Five Minute Rule

Assume a block is accessed every X seconds:

CD = cost if we keep that block on disk

- $D = cost of disk unit
- I = number of IOs that unit can perform per second
- In X seconds, unit can do XI IOs
- So, CD = $D / XI

The Five Minute Rule

Assume a block is accessed every X seconds:

CM = cost if we keep that block in RAM

- \$M = cost of 1MB of RAM
- P = number of pages in 1MB RAM
- So CM = \$M / P

The Five Minute Rule

Assume a block is accessed every X seconds:

If CD is smaller than CM,

- keep block on disk
- else keep in memory

Break even point when CD = CM, or X = ($D P) / (I $M)

Using 1997 numbers

P = 128 blocks/MB (8kB pages)

I = 64 accesses/sec/disk (16ms to read 8kB)

$D = $2000/disk (9GB HDD + controller)

$M = $15/MB of RAM

X = 266 seconds (about 5 minutes)
(did not change much from 1985 to 1997)

Using 2007 numbers

P = 256 blocks/MB (4KB pages)

I = 83 accesses/sec/disk (12ms to read 4KB)

$D = $80/disk (250GB SATA HDD)

$M = $0.047/MB of RAM

X = 5,248 seconds (about 1.5 hours)

Using 2007 numbers

P = 256 blocks/MB (4KB pages)

I = 6,200 accesses/sec/disk (0.16ms to read 4KB)

$D = $999/disk (32GB SSD)

$M = $0.047/MB of RAM

X = 876 seconds (about 15 minutes)

Using 2016 numbers

P = 256 blocks/MB (4KB pages)

I = 64,000 accesses/sec/disk (0.015ms to read 4KB)

$D = $685/disk (240GB SSD)

$M = $0.034/MB of RAM

X = 805 seconds (about 13.5 minutes)

The changing memory hierarchy

The falling price of SSD makes it a viable tier between the performance of DRAM and the capacity of HDDs

- The break-even for DRAM-SSD on modern systems is again ~5 minutes (the DRAM-HDD case is now about 4 hours)
- The break-even for SSD-HDD is now about 1.5 *days*
- The energy costs of DRAM are much greater (>10x) than SSD
- The energy costs of HDD are much greater than tape (idling consumption)
- Likely transition to NVDIMM memory (DRAM+NAND flash)

	1987	1997	2007	2018
DRAM	$5000	$14.6	$0.05	$0.005
HDD	$83	$0.22	$0.0003	$0.00002
SDD			$0.03	$0.0005

Disk Organisation

Overview

- Data Items
- Records
- Blocks
- Files

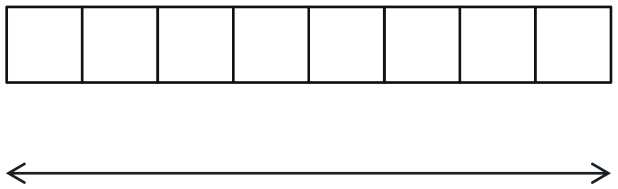
Data Items

Data Items

We might wish to store:

- a salary
- a name
- a date
- a picture

We have: bytes



8 bits

Representing numbers

Integer (short): 2 bytes

- e.g. 57 is

0	0	0	0	0	0	0	0

0	0	1	1	1	0	0	1

Real numbers: IEEE 754 (floating point)

- 1 bit sign, n bits for mantissa, m bits for exponent

Representing characters

Various coding schemes: ASCII, utf-8

• 'A'	0	1	0	0	0	0	0	1
• 'c'	0	1	1	0	0	0	1	1
• CR	0	0	0	0	1	1	0	1

Representing booleans

1 byte per value

- True

1	1	1	1	1	1	1	1

- False

0	0	0	0	0	0	0	0

We can pack more than one value per byte, if we're desperate

Representing dates

Days since a given date (integer)

- 1st Jan 1900
- 1st Jan 1970 (UNIX epoch)

ISO8601 dates

- Calendar dates: YYYYMMDD (8 characters)
- Ordinal dates: YYYYDDD (7 characters)

Representing times

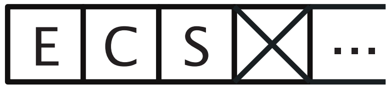
Seconds since midnight (integer)

ISO8601 times

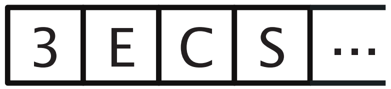
- HHMMSS (6 characters)
- HHMMSSFF (8 characters, to represent fractional seconds)

Representing strings

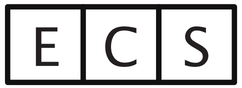
Null terminated



Length given



Fixed length



Representing bit arrays

length	bits

In general...

Data items are either

- Fixed length (integers, characters, etc)
- Variable length (strings, bit arrays) usually with length given at start

May also include type of data item

- Tells us how to interpret the item
- Tells us size, if fixed

Records

Records

Collection of related data items (*fields*)

e.g. Employee record consists of:

- name field
- salary field
- employment start date field

Record types

Records may have fixed or variable formats

Records may have fixed or variable lengths

Fixed format records

Schema describes the structure of records:

- number of fields
- types of fields
- order in record
- meaning of each field

Example: Fixed format record

Employee record structure:

1. e#, 2 byte integer
2. name, 10 char
3. dept, 2 byte code

schema

5	5	s	m	i	t	h						0	2
8	3	j	o	n	e	s						0	1

records

Variable format records

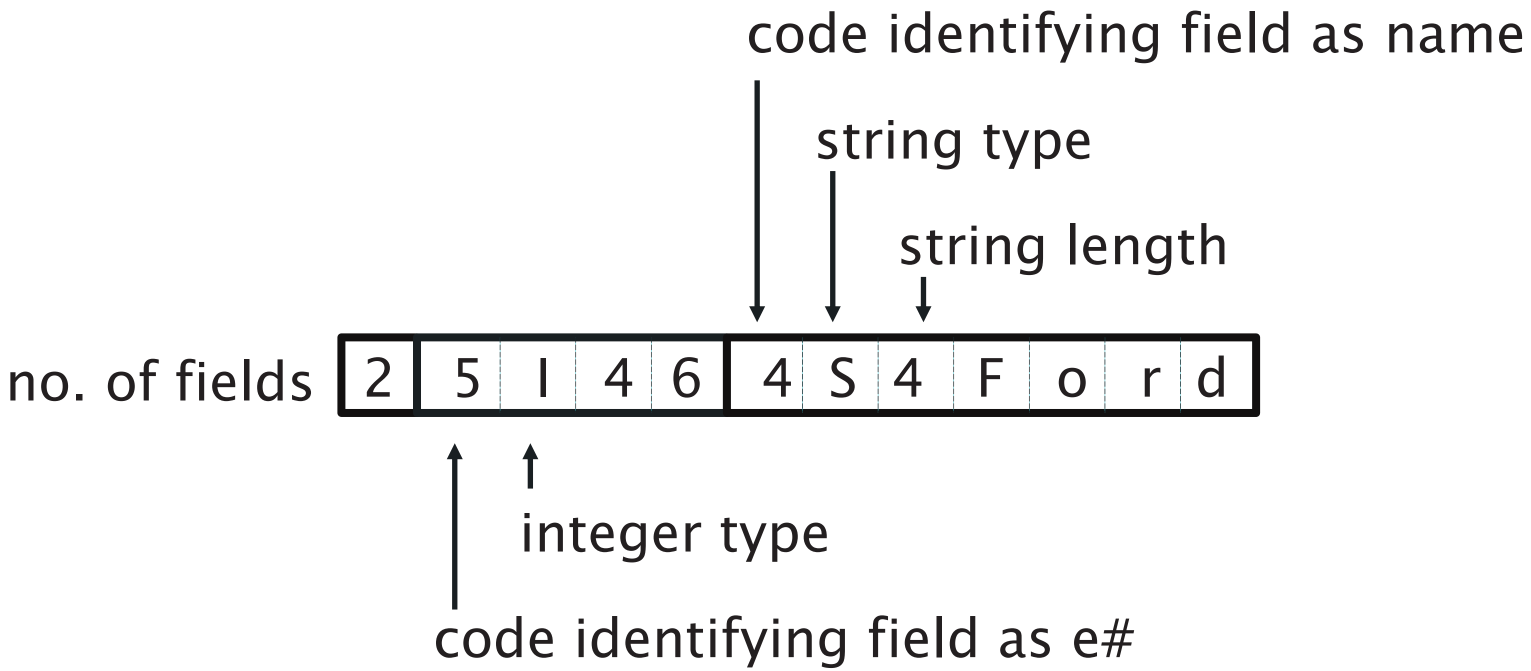
Schema-less format

- Record itself contains format: “self-describing”

Useful for sparse records, repeating fields, evolving formats

May waste space compared to a fixed format record

Example: Variable format record



Record headers

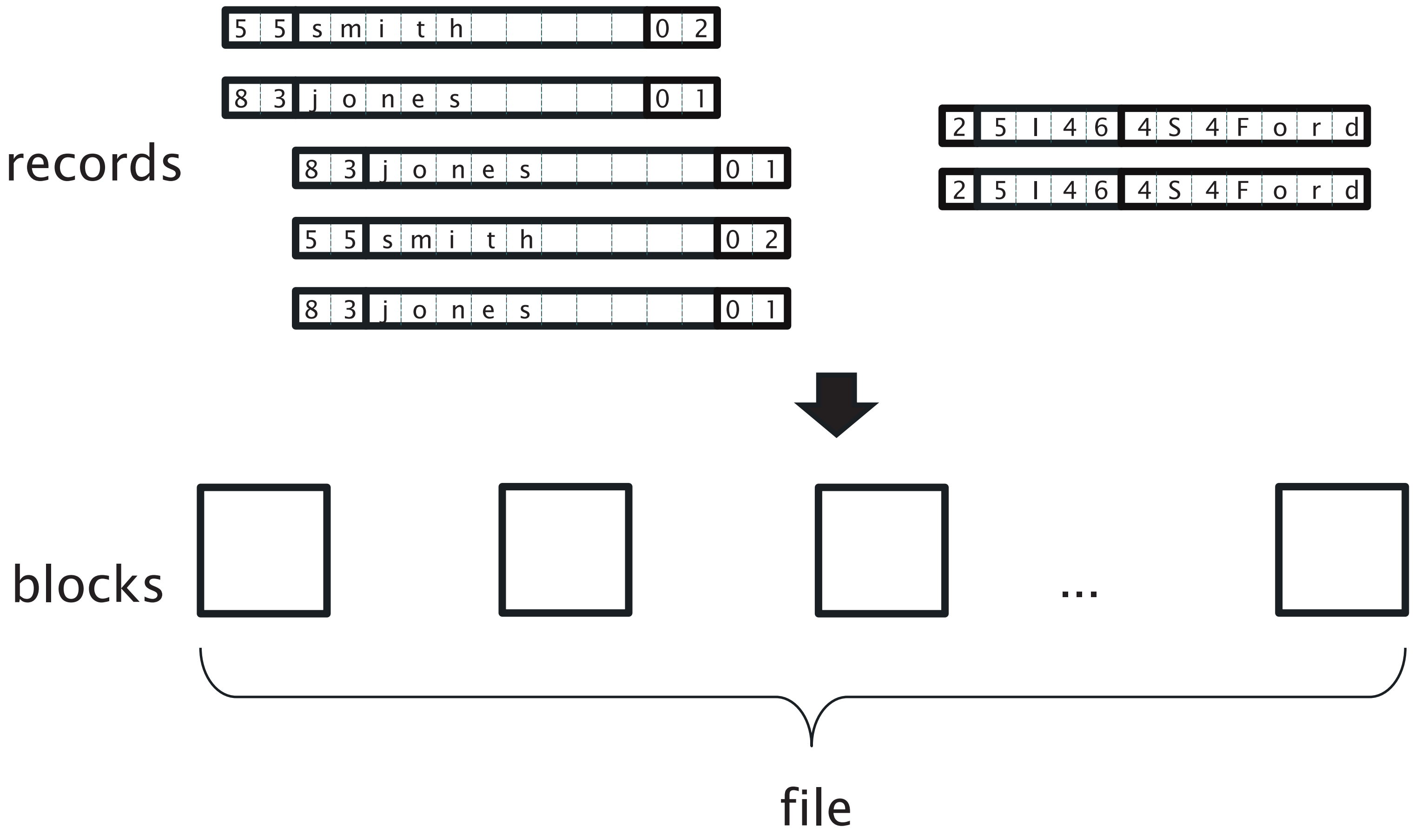
Data at beginning of record that describes record:

- record type (points to schema)
- record length
- timestamp

Intermediate between fixed and variable format

Blocks

Storing records in blocks



Block header

Data at beginning that describes block

May contain:

- File ID (or RELATION or DB ID)
- This block ID
- Record directory
- Pointer to free space
- Type of block (e.g. contains recs type 4; is overflow, …)
- Pointer to other blocks “like it”
- Timestamp ...

Placing records in blocks

Considerations:

- separating records
- spanned vs. unspanned
- sequencing
- indirection

Separating records in a block

Three approaches:

1. use fixed length records - no need to separate
2. use a special marker to indicate record end
3. give record lengths (or offsets)
   - within each record
   - in block header

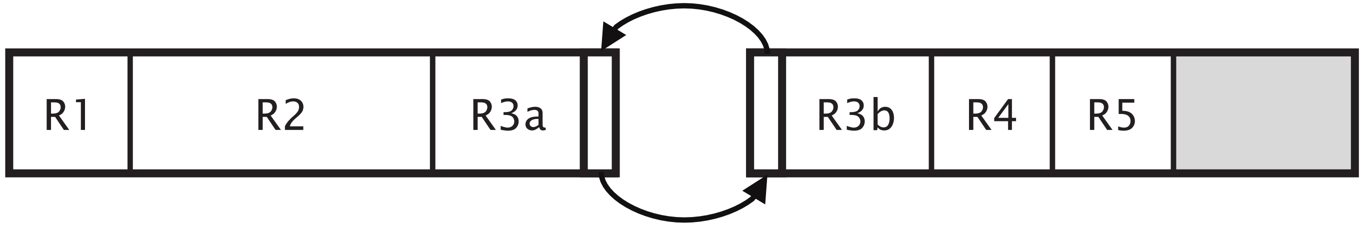
Spanned vs. Unspanned

Unspanned: each record must fit within a single block

Spanned: records may be split between blocks

Spanned records

need indication
of continuation
(from where?)



need indication
of partial record
“pointer” to rest

Spanned vs. Unspanned

Unspanned records are much simpler, but may waste space…

Spanned records are essential if record size > block size

Sequencing

Sequencing: ordering records in file (and block) by some key value

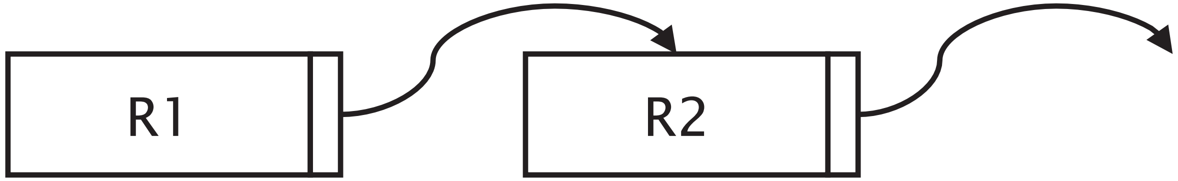
Makes it possible to efficiently read records in order

- e.g., to do a merge-join — discussed later in module

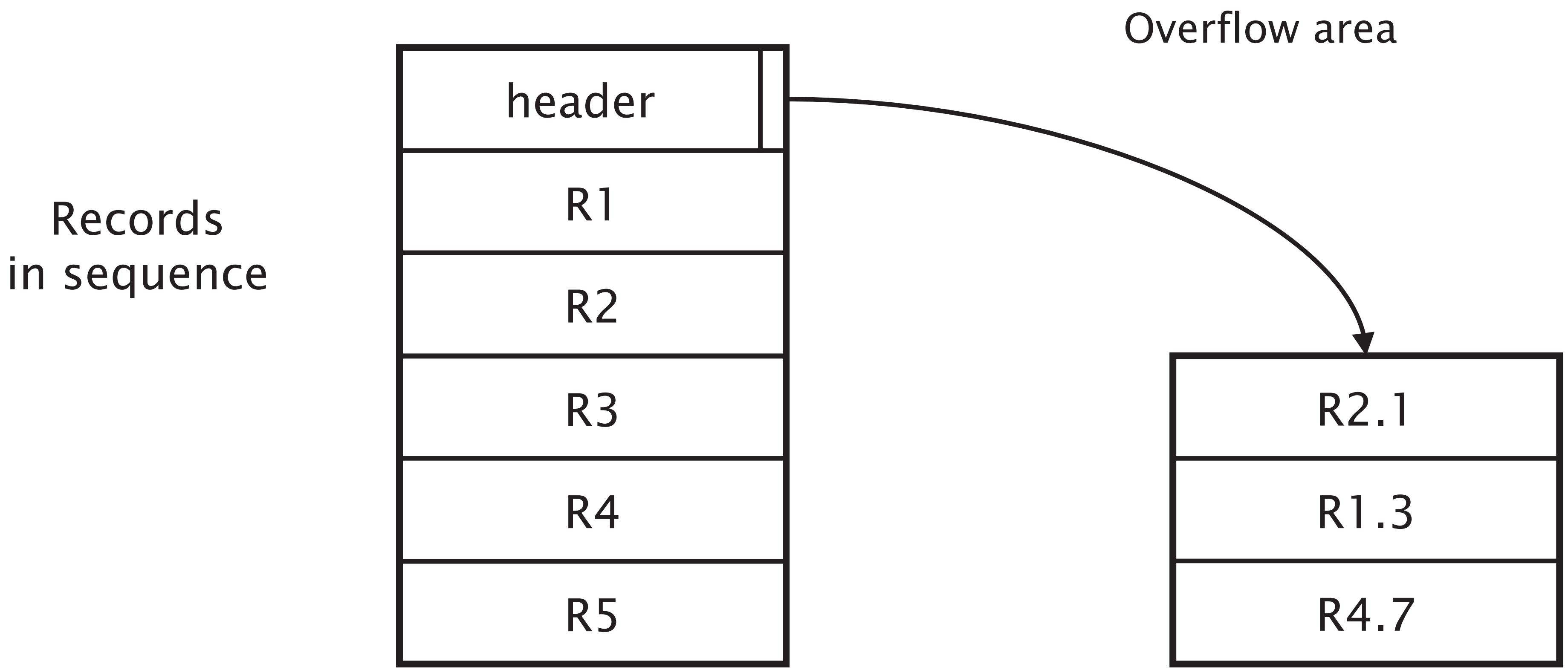
Sequencing Options

Next record physically contiguous:

Linked records:



Sequencing Options



Indirection

How do we refer to records?

Many options:

- physical addressing
- indirect addressing
- other options in between

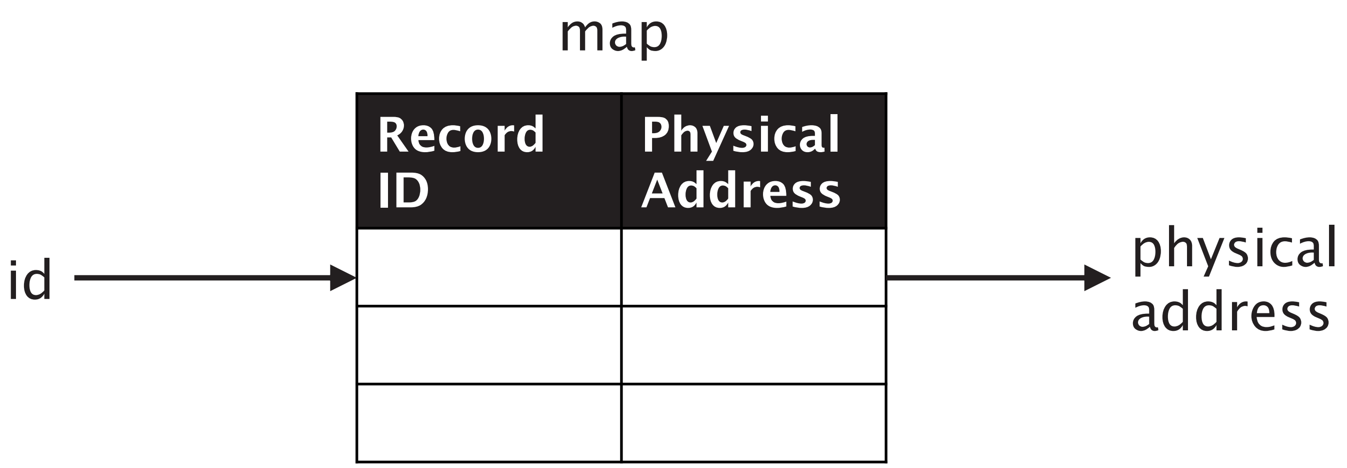
Tradeoff between:

- flexibility (easier to move records on insertion/deletion)
- cost (of maintaining indirection)

Physical Addressing

$$\text{Record ID} = \left\{ \begin{array}{l} \left.\begin{array}{l}\text{Device}\\ \text{Cylinder}\\ \text{Head}\\ \text{Sector}\end{array}\right\} \text{Block ID} \\ \text{Offset in block} \end{array} \right.$$

Indirect Addressing

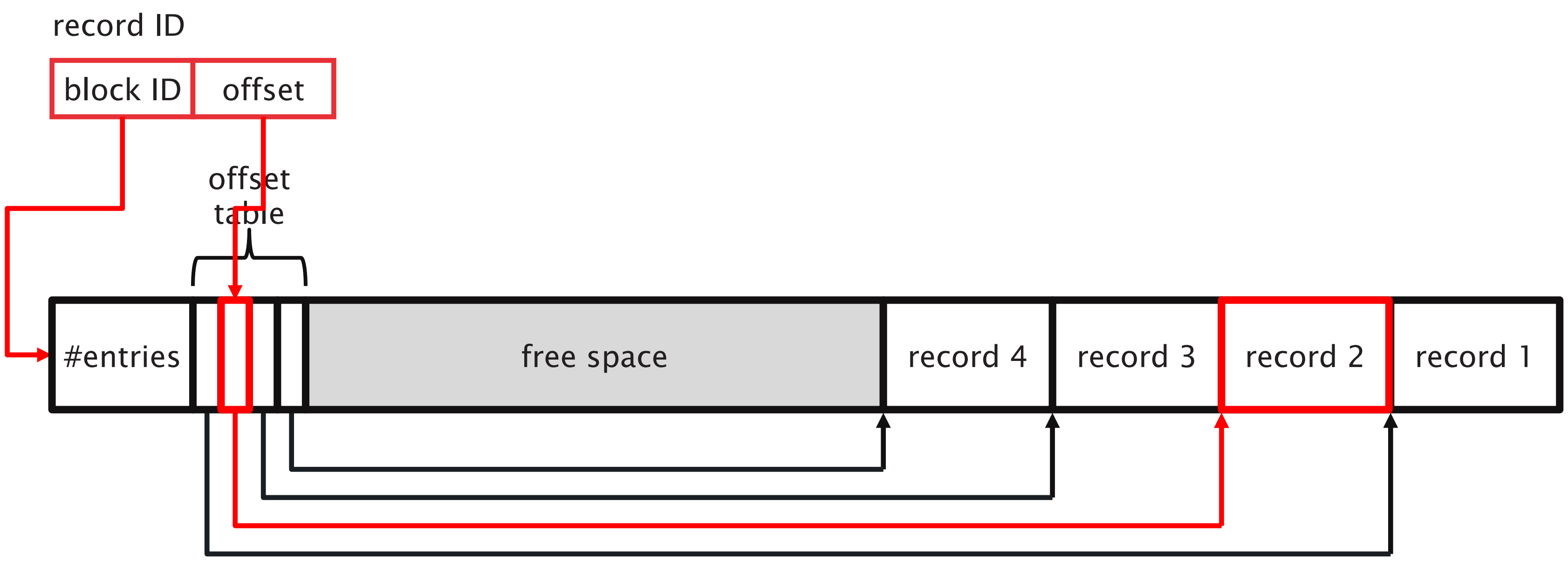


Record ID is arbitrary bit string

Indirection in block

Typical implementation

- Records can be shifted within block without changing record ID
- Access to a given record ID is fast – only a single block access needed



Address Management

Every block and record has *two* addresses:

- a database address (when in secondary storage)
- a memory address (when copied into a buffer)

Translation table records mapping from database addresses to memory addresses:

DB address	Memory address

When in a buffer, using only memory addresses (= pointers) is more efficient

Pointer Swizzling

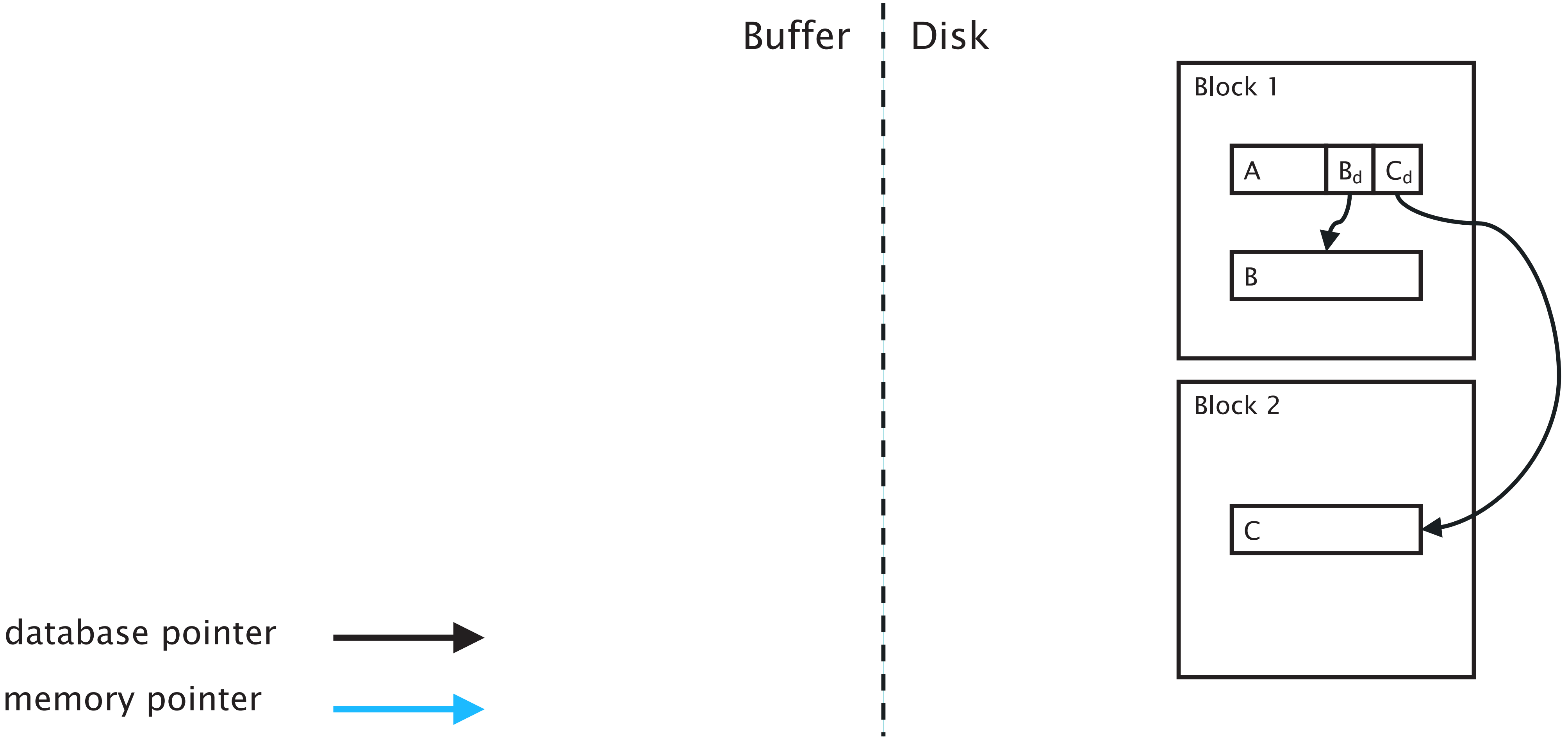
General term for techniques used to translate database address space to virtual memory address space

Swizzled pointers typically consist of

- One bit to indicate whether the pointer is a database address or a memory address
- A database or memory pointer, as appropriate

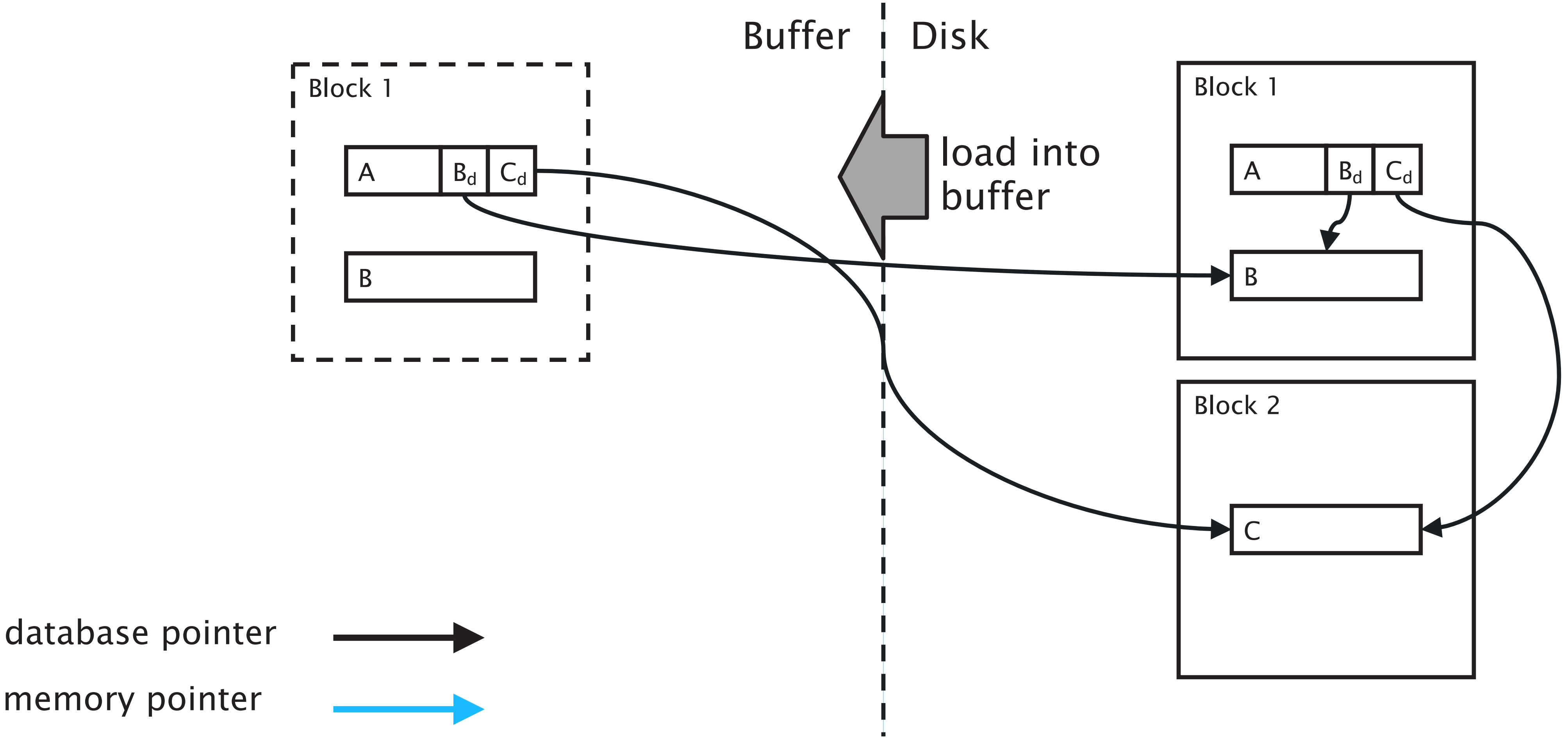
Translation table is used to convert pointers (and to record the conversion)

Swizzling

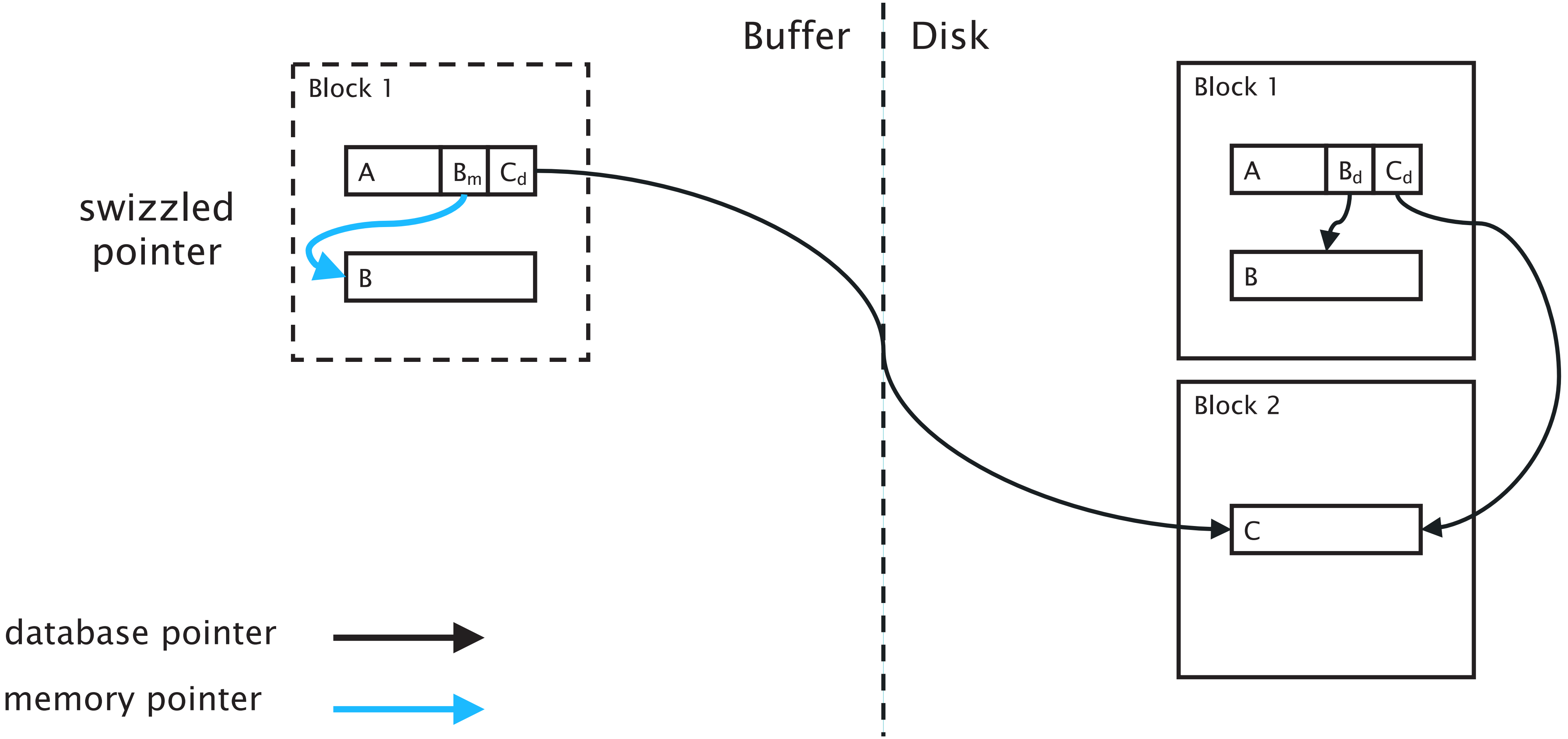


Buffer
Disk
Block 1
A
B_d
C_d
B
Block 2
C
database pointer
memory pointer

Swizzling

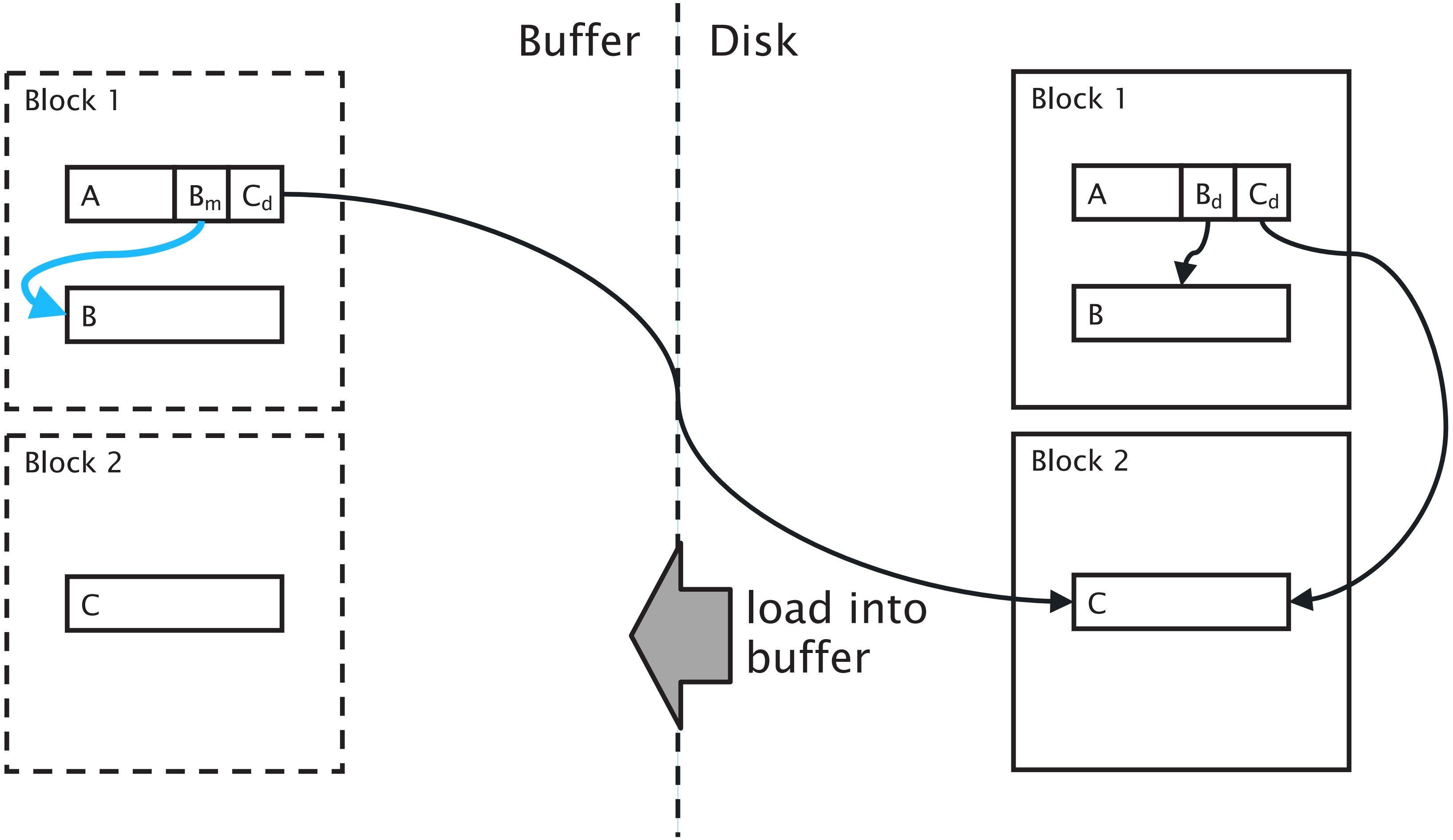

Buffer
Disk
Block 1
A
B_d
C_d
B
load into buffer
Block 1
A
B_d
C_d
B
Block 2
C
database pointer
memory pointer

Swizzling



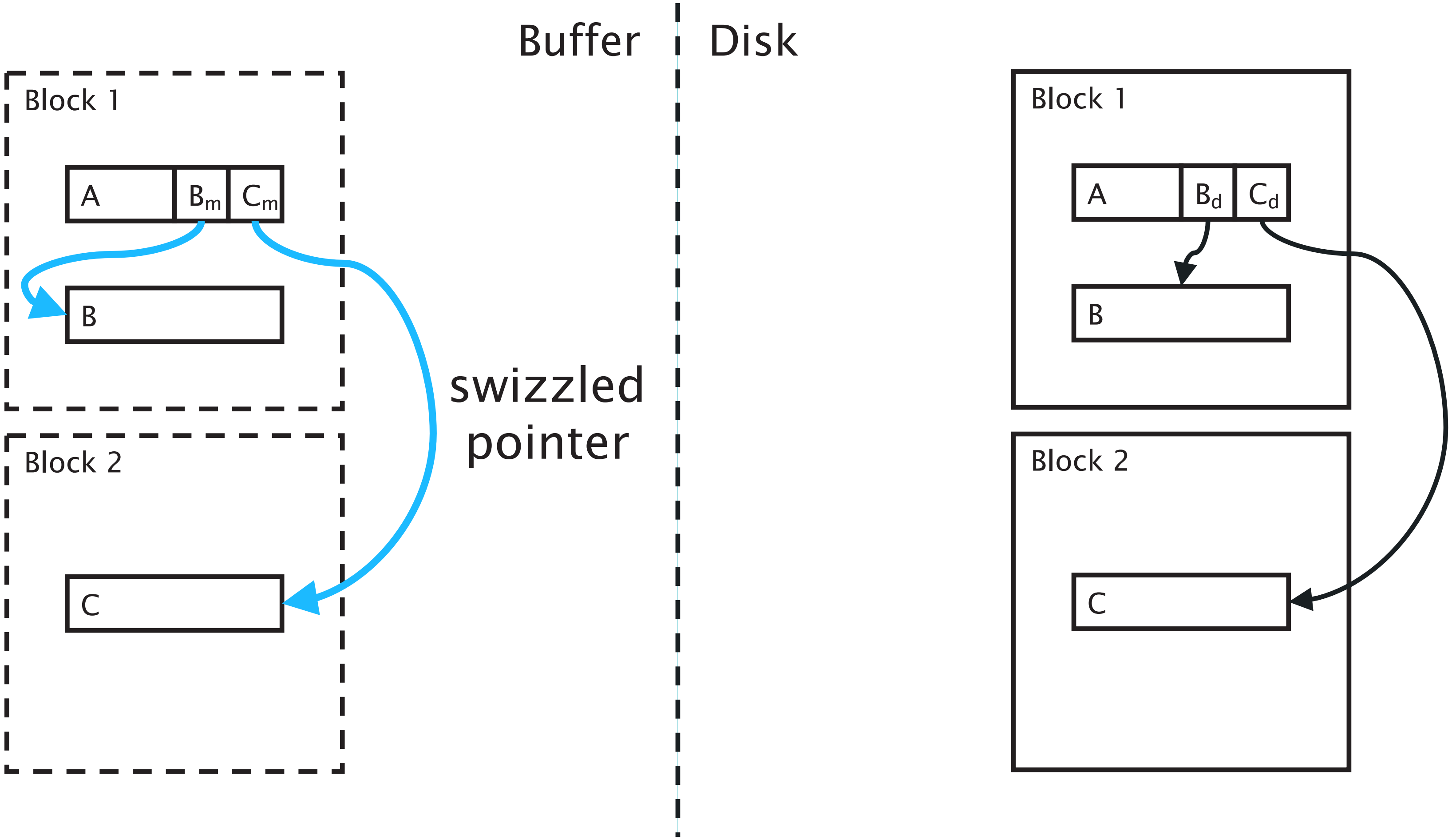
Buffer
Disk
Block 1
A
B_m
C_d
swizzled
pointer
B
Block 1
A
B_d
C_d
B
Block 2
C
database pointer
memory pointer

Swizzling



Buffer
Disk
Block 1
A
B_m
C_d
B
Block 2
C
load into buffer
Block 1
A
B_d
C_d
B
Block 2
C

Swizzling



Buffer
Disk
Block 1
A
B_m
C_m
B
swizzled pointer
Block 2
C
Block 1
A
B_d
C_d
B
Block 2
C

Swizzling Strategies

Automatic

- As soon as block brought into memory, locate all pointers and addresses and enter them into translation table
- Replace pointers in blocks with new entries

On Demand

- Leave all pointers unswizzled when block in brought into memory
- Swizzle pointers only when dereferenced

No swizzling

- Use translation table to map pointers on each dereference

Unswizzling

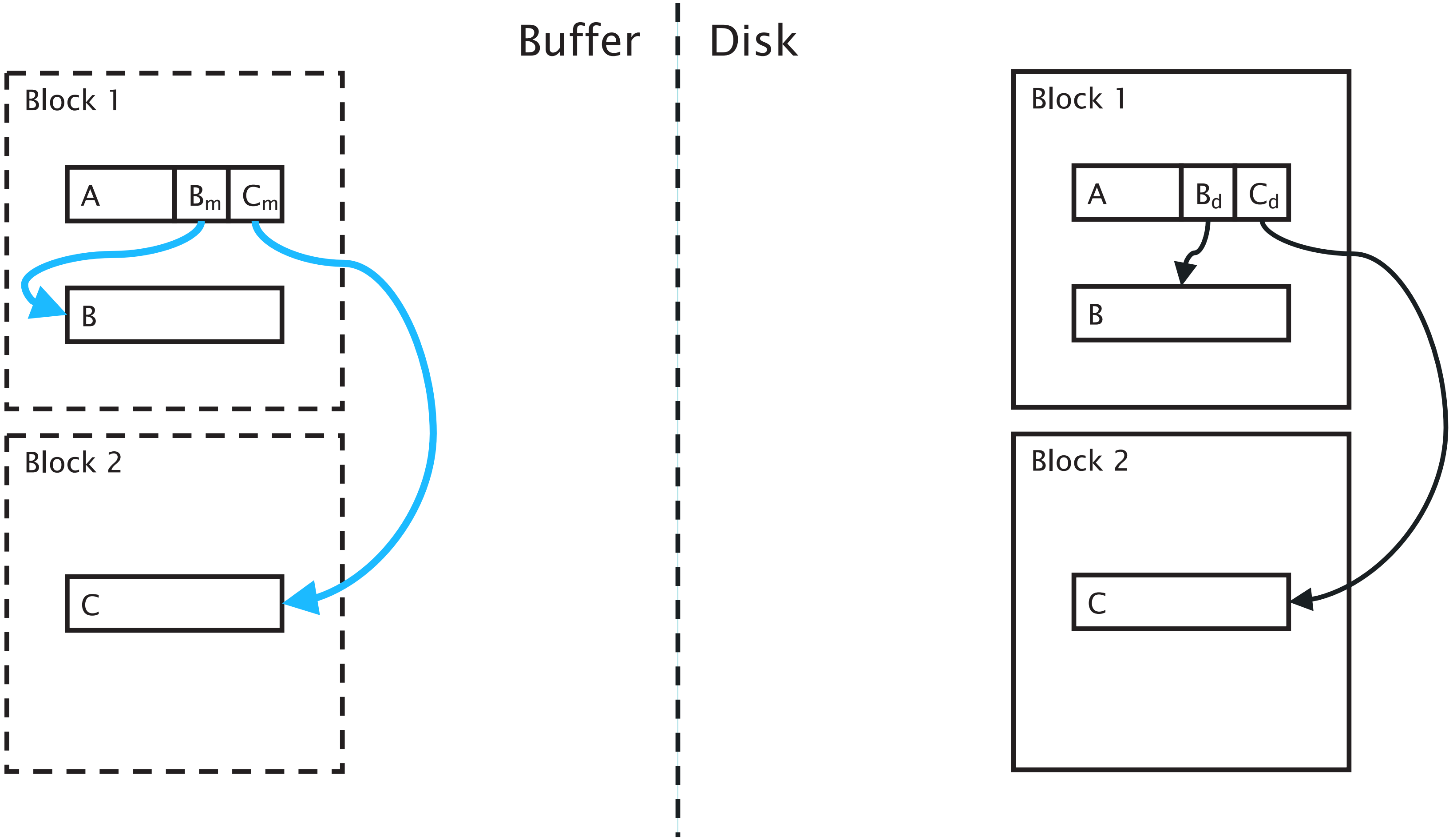
Reverse of the swizzling operation – rewrite memory addresses as database addresses

- Use the translation table
- Translation table is designed to map from DB address to memory address – need an index

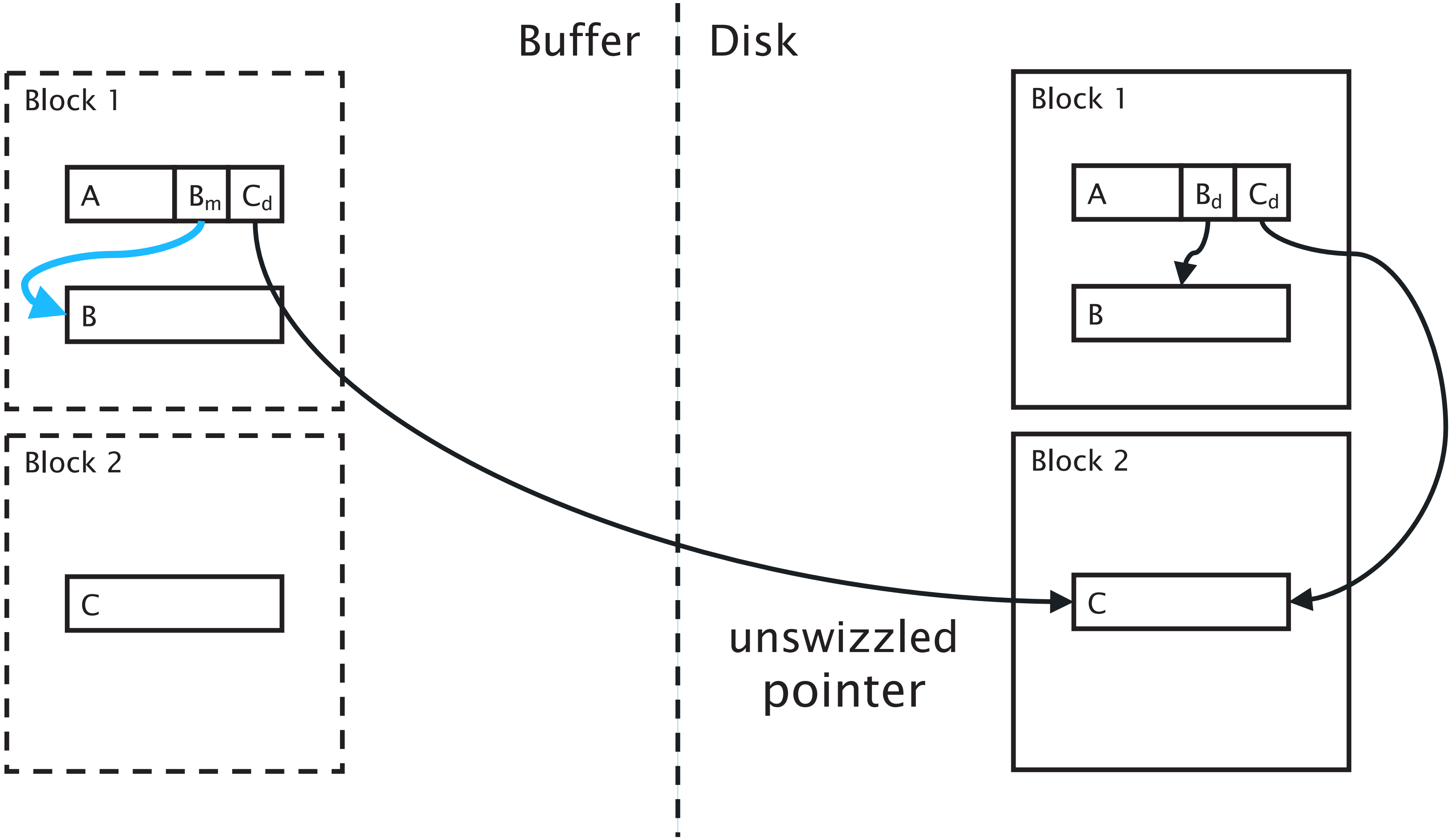
Need to be aware of the relationship between unswizzling and unpinning

- Blocks in the buffer pool are *pinned* to indicate that some part of the DBMS is using their contents
- However, a block may be pinned if there are swizzled pointers that point to that block
- In order to unpin the block (to allow the frame to be reused), we need to unswizzle any pointers to that block

Swizzling

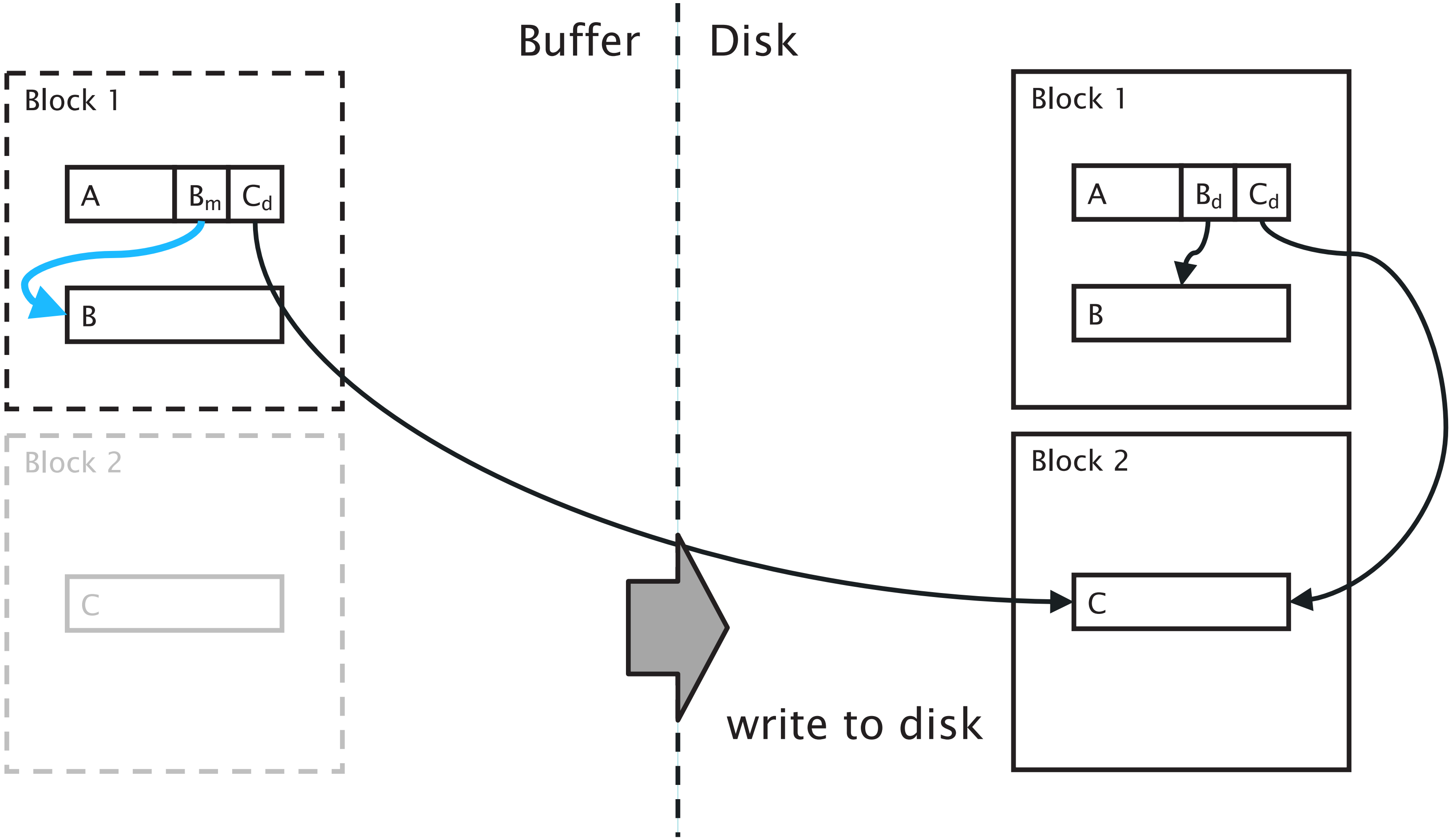


Swizzling



block to unpin and write

Swizzling



Buffer
Disk
Block 1
A
B_m
C_d
B
Block 2
C
write to disk
Block 1
A
B_d
C_d
B
Block 2
C

Insertion and Deletion

Insertion: the easy case

Records not in sequence

- Insert new record at end of file or in deleted slot
- If records are variable size, not as easy...

Insertion: the hard case

Records in sequence

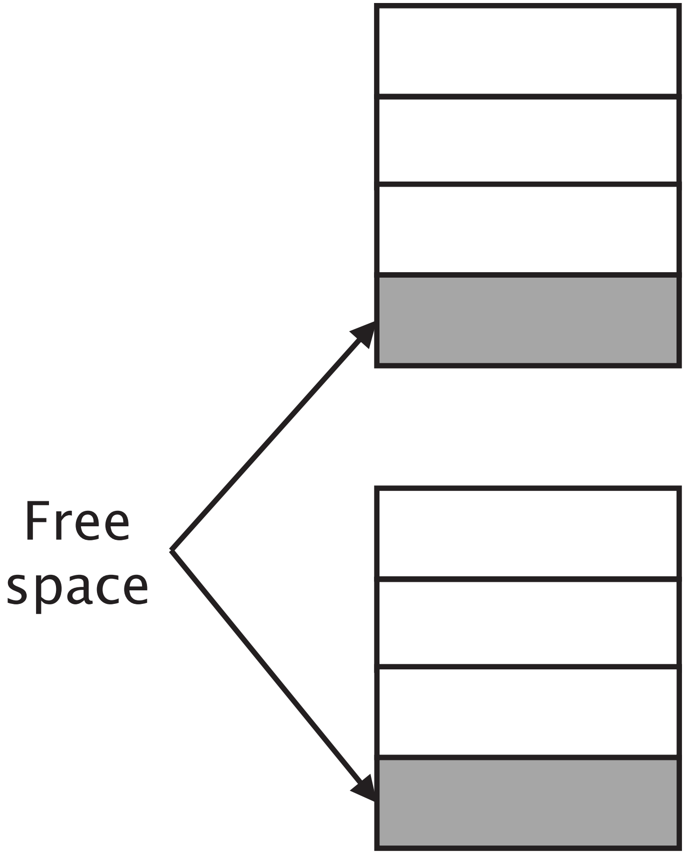
- If free space “close by”, not too bad...
- Or use overflow idea...

Insertion considerations

How much free space should we leave:

- In each block?
- In each track?
- In each cylinder?

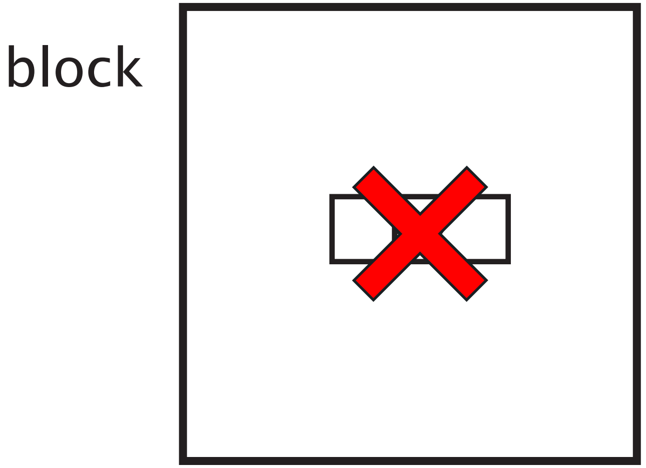
How often should we reorganise files?



Deletion

Two main options:

- Immediately reclaim space
- Mark space as deleted



Deletion marking

May need a chain of deleted records (for re-use)

Need a way to mark deleted records:

- special characters
- delete field
- in map

Deletion tradeoffs

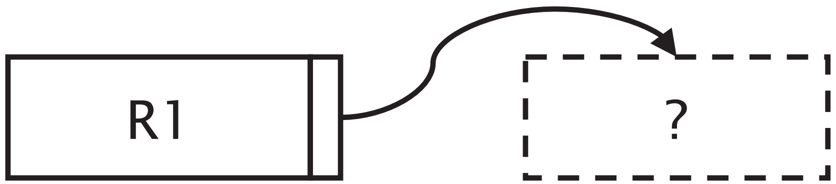
How expensive is it to move valid record to free space for immediate reclaim?

How much space is wasted?

- e.g., deleted records, delete fields, free space chains,...

Deletion considerations

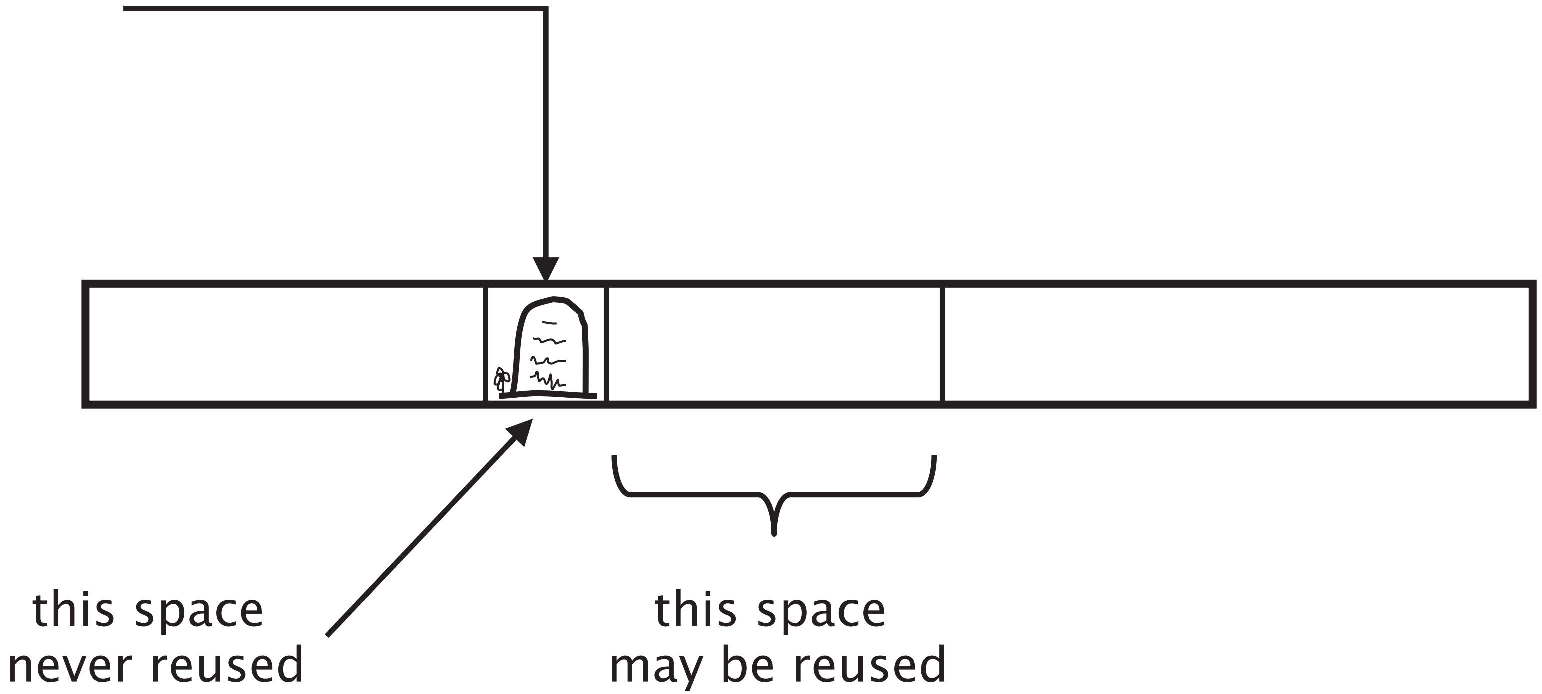
How do we deal with dangling pointers?



Tombstones

Leave “MARK” in map or old location

Physical IDs



Tombstones

Leave “MARK” in map or old location

Logical IDs

map

ID	LOC
7788	

Never reuse
ID 7788 nor
space in map...

Further Reading

Further Reading

- Chapter 13 of Garcia-Molina et al
- Gray, J. and Putzolu, F. 1987. The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for CPU time. Proceedings of SIGMOD 1987, 395-398.
- Gray, J. and Graefe, G. 1997. The five-minute rule ten years later, and other computer storage rules of thumb. *SIGMOD Record*. 26(4), 63-68.
- Graefe, G. 2009. The five-minute rule 20 years later (and how flash memory changes the rules). *Communications of the ACM*. 52(7), 48-59.
- Appuswamy, R., Graefe. G., Borovica-Gajic. R. and Ailamaki. A. 2019. The Five-Minute Rule 30 years later, and its impact on the storage hierarchy. Communications of the ACM 62(11), pp. 114-120.

Next Lecture: Access Structures