We'll start with a summary of RAID

Then cover table & buffer management

We will start at 2:10 pm

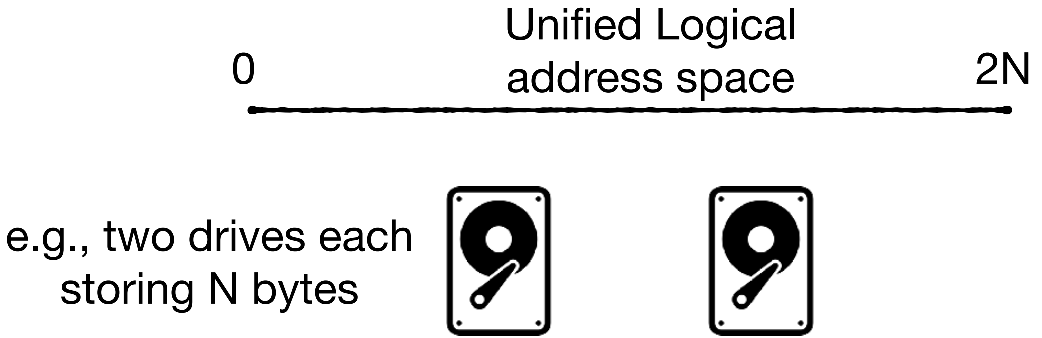
RAID Addresses Three Problems

Our database size exceeds one drive and we need more storage

A drive fails, and we need to recover its data

We want to overcome the limits of one storage device speed

Expose a larger logical address space to OS



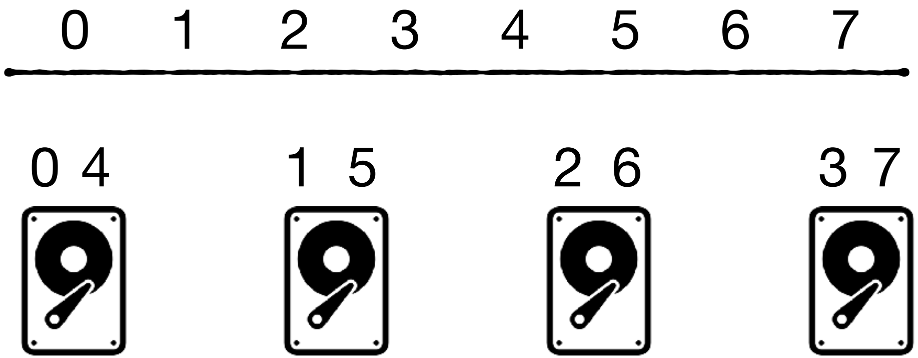
Looks to the OS like one drive, though consists of many

The spectrum of RAID designs

RAID 0 **RAID 1** **RAID 0+1** RAID 4 RAID 5 RAID 6

RAID 0 - Pure striping

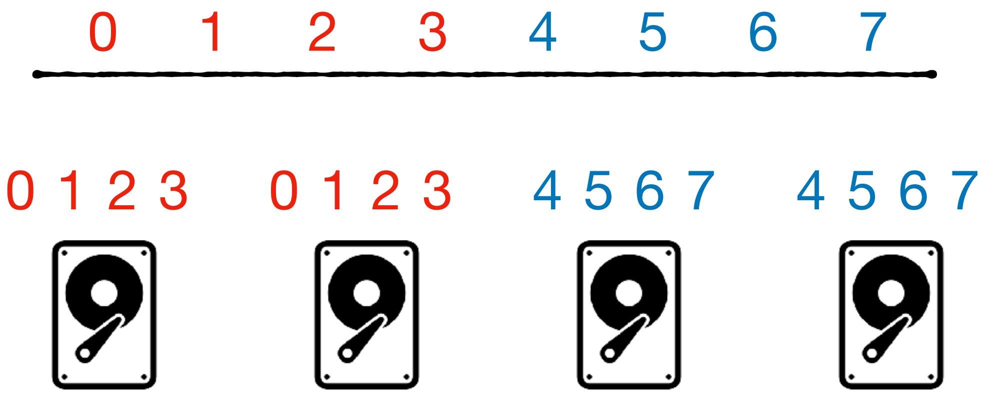
Stripe data in the logical address



1. Much faster sequential writes and reads
2. Also improvement for random writes and reads due to load balancing
3. No redundancy. If one disk fails, we lose data.

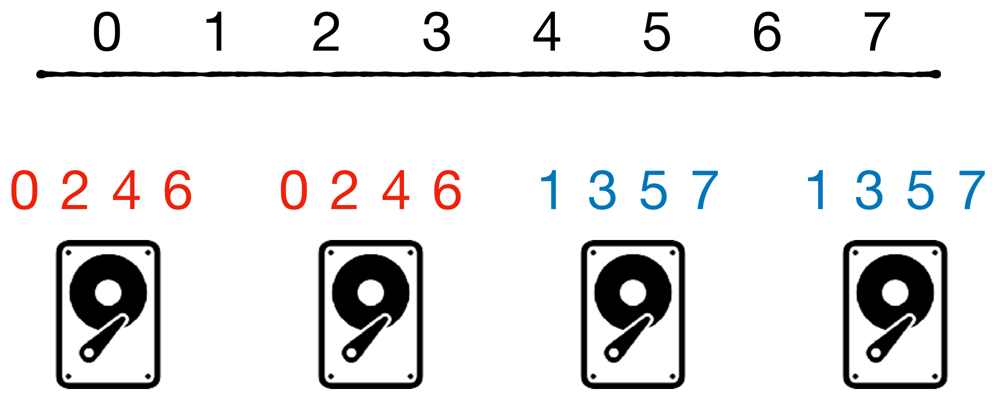
RAID 1 - Mirroring

Each drive has one mirror



1. Slower writes as they must make 2 copies
2. Faster reads as we have a choice to read from a non-busy drive
3. Allows recovery of a disk but costs 50% of storage capacity

RAID 0+1 - Striping and Mirroring



1. Faster sequential reads and writes as they are more distributed
2. Writes still require making two copies, and reads still have flexibility
3. Still requires 50% of storage capacity

RAID 0 RAID 1 RAID 0+1

know these for midterm

RAID 4 RAID 5 RAID 6

we'll cover these later

Tables Management

Database System Technology - Lecture 3, Chapter 9

Niv Dayan

Database Tables

A database consists of multiple tables

How do we store them in storage efficiently?

Customers

ID	Name	email	Addr

Orders

ID	Customer ID	Product ID	Date

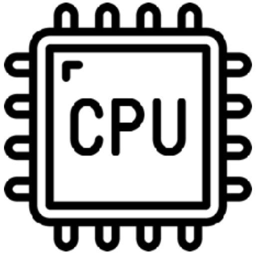
Operations to Efficiently Support

1. Scans e.g., select * from Customers
2. Deletes e.g., delete from Customers where name = “…”
3. Updates e.g., update Customers set email = “…” where name = “”
4. Insertions e.g., Insert into Customers (, , ,)

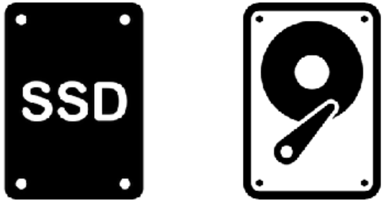
Customers

ID	Name	email	Addr

Optimizing for Data Movement



In previous courses on algorithms & data structures, you learned to optimize CPU cycles for an algorithm.

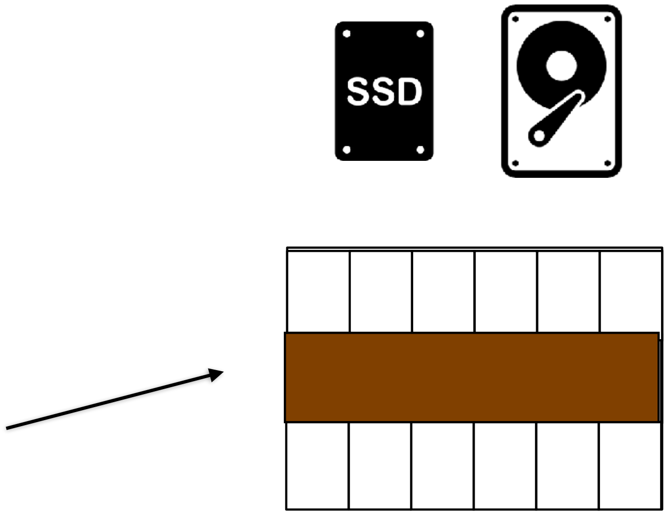


As storage devices are far slower, in this course we focus on optimizing data movement.

First Insight: Database Pages

Reading/writing from storage at units of less than ≈4KB does not pay off.

Reading/writing at very large units consumes memory and is less flexible for applications

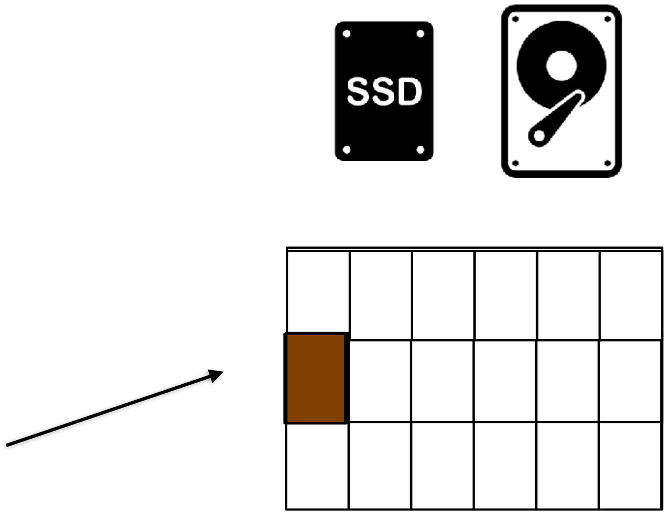


Storage

Database Pages

To balance, DBs use ≈4KB as the read/write unit. This is known as a database page.

An I/O (input/output) is one read or write request of one database page.

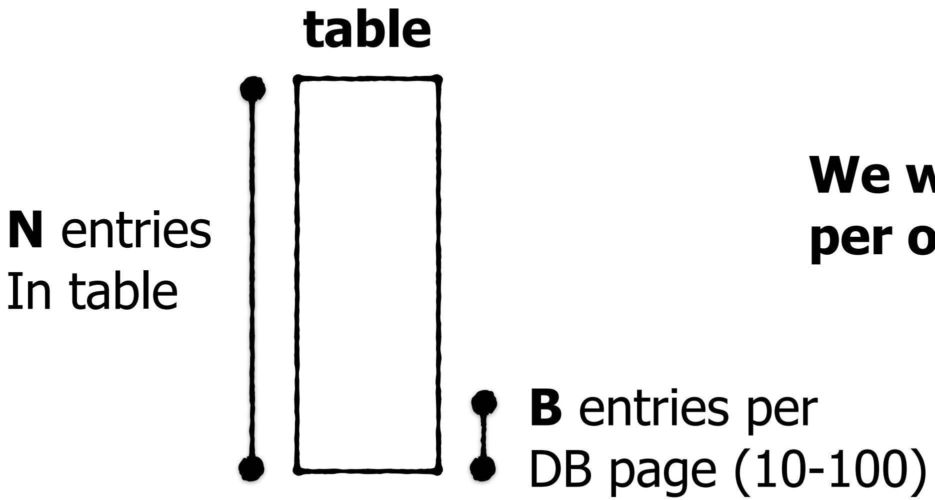


Storage

The Disk Access Model (The DAM Model)

We will shortly propose algorithms to support scans/delete/updates/inserts

To reason about such algorithms, we need a cost model

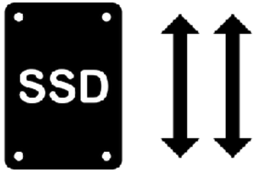


We will count the worst-case number of I/Os per operations with respect to N and B

The Disk Access Model (The DAM Model)

This model is imperfect. It ignores many characteristics of storage.

Ignores that sequential disk reads are more economical



Ignores that SSD asynchronous I/O are faster

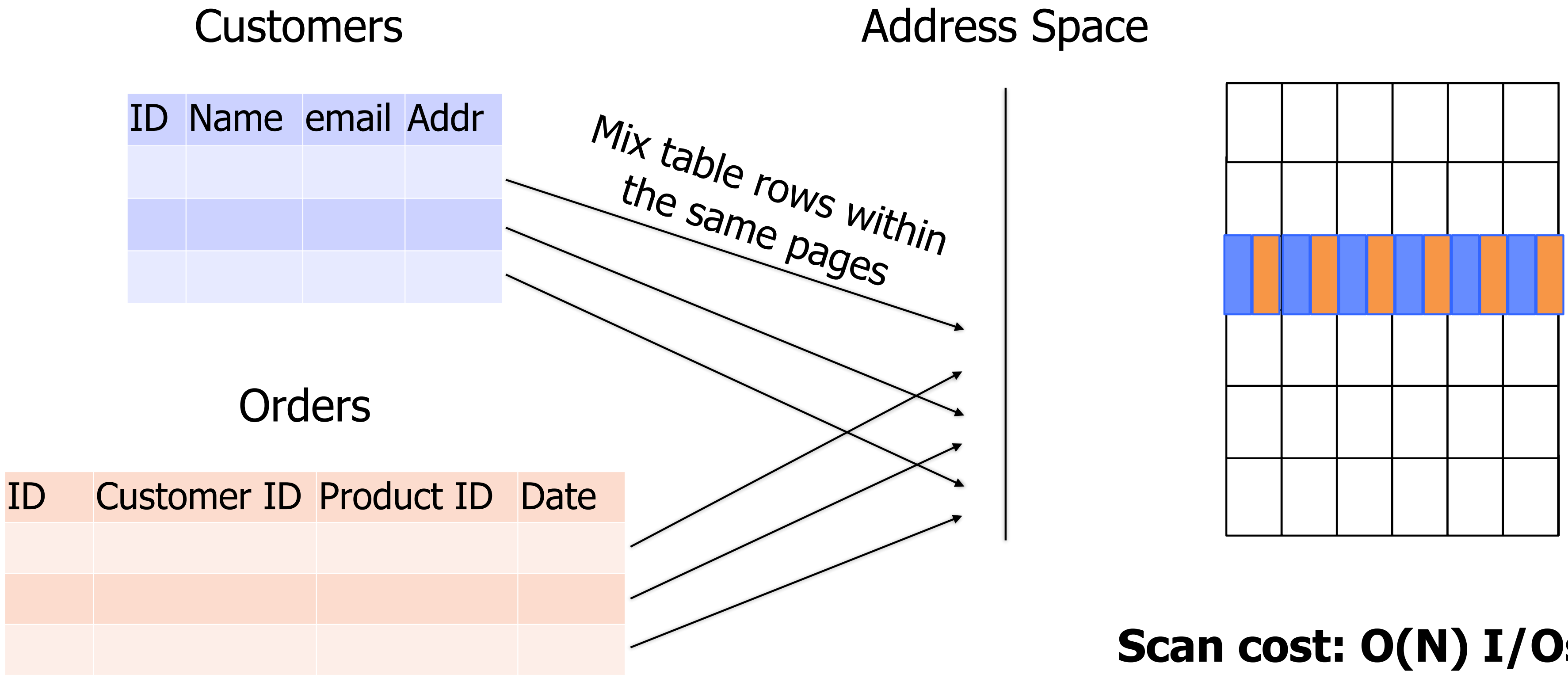
Ignores SSD garbage-collection due to random writes

However, it’s useful due to its simplicity.

Operations

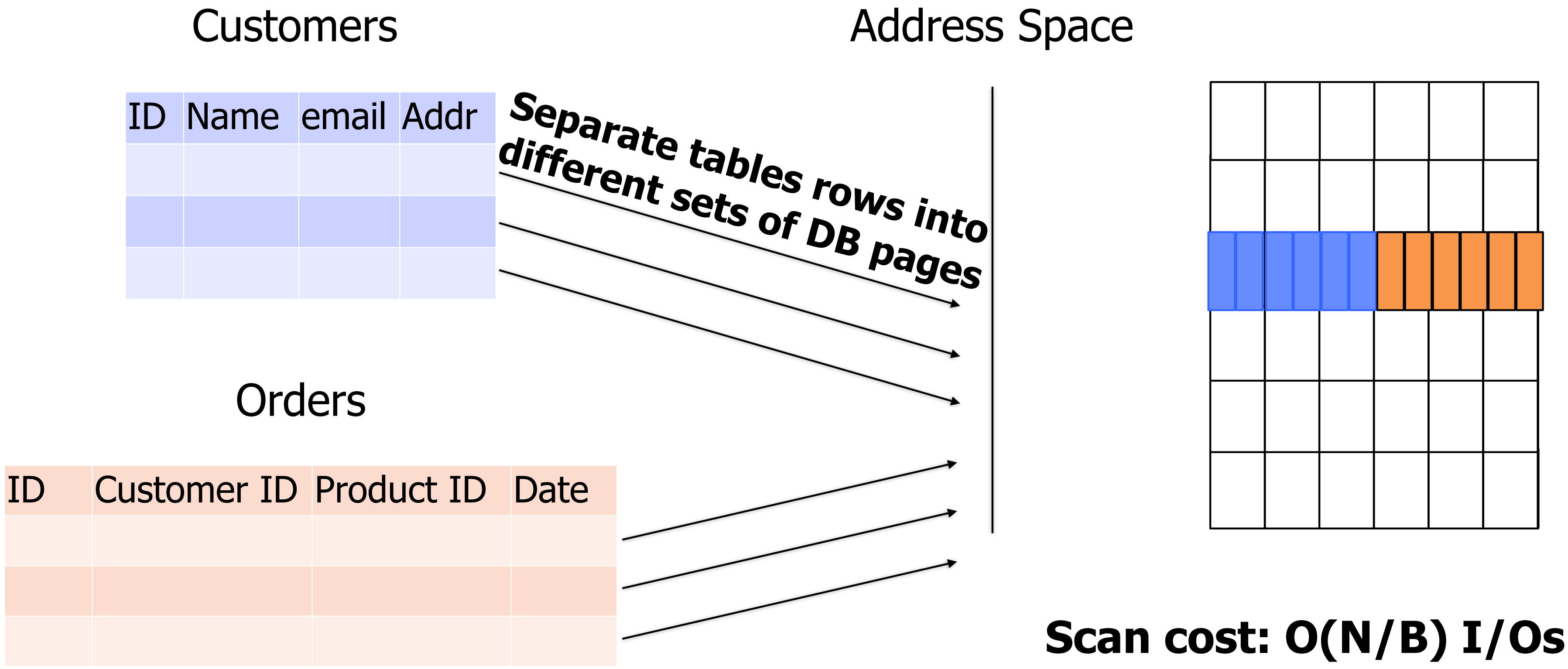
1. Scans e.g., select * from Customers
2. Deletes e.g., delete from Customers where name = "…"
3. Updates e.g., update Customers set email = "…" where name = ""
4. Insertions e.g., Insert into Customers (, , ,)

Scans - How not to Support Them



Scan cost: O(N) I/Os

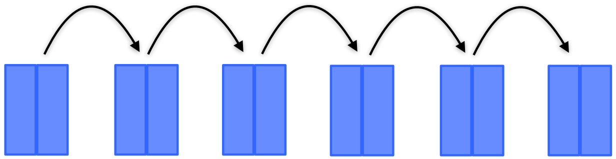
Efficient Scans



Efficient Scans

Which pages belong to which table?

Simplest Solution: Linked List



Problem: **entails synchronous I/Os, which do not exploit SSD parallelism**

Solution:

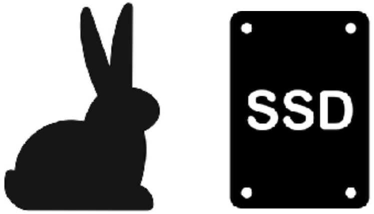
Efficient Scans

Which pages belong to which table?

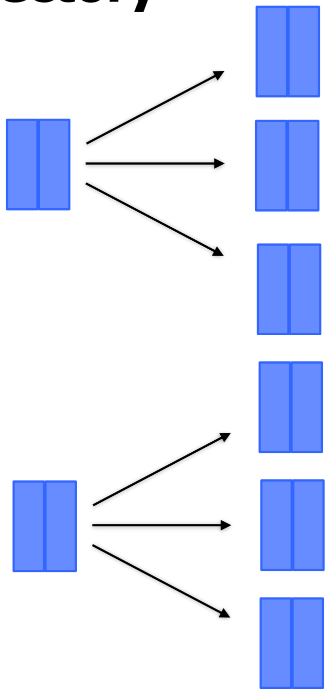
Simplest Solution: Linked List

Problem: entails synchronous I/Os, which do not exploit SSD parallelism

Solution: **Employ directory to allow reading many pages asynchronously**



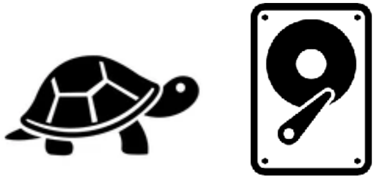
Directory

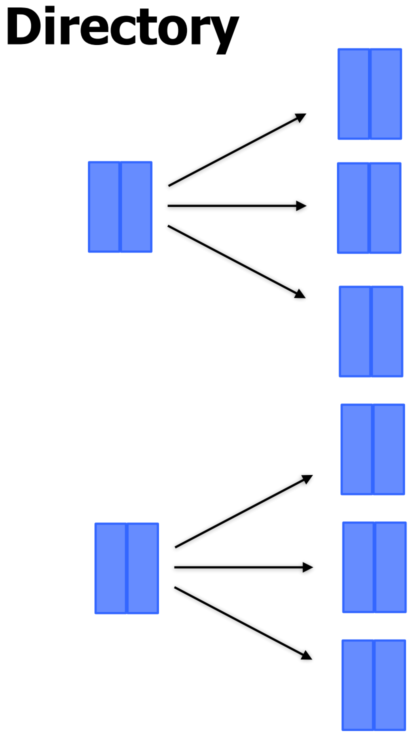


Efficient Scans

Which pages belong to which table?

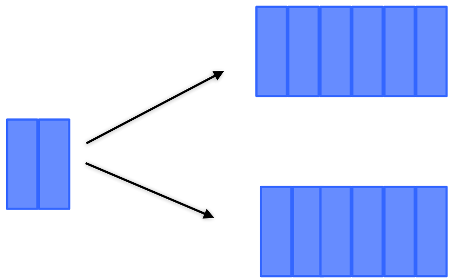
Problem: small I/Os, which do not saturate a disks's sequential bandwidth





Efficient Scans

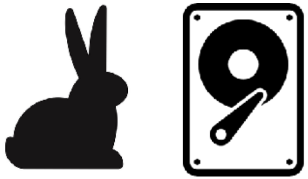
Directory



Which pages belong to which table?

Problem: small I/Os, which do not saturate a disks's sequential bandwidth

Solution: **Store multiple database pages contiguously along "extents"** (8-64 pages)



Efficient Scans

Which pages belong to which table?

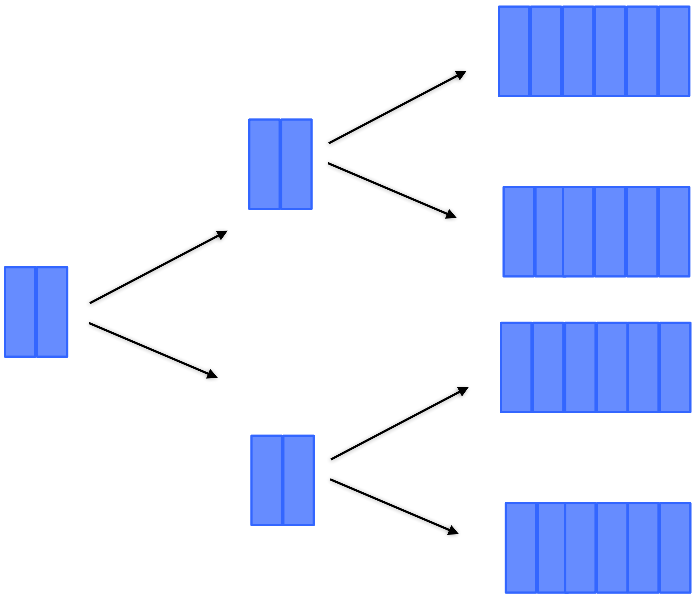
Problem: small I/Os, which do not saturate a disks's sequential bandwidth

Solution: Store multiple database pages contiguously along "extents" (8-64 pages)

Bonus: Saves some metadata

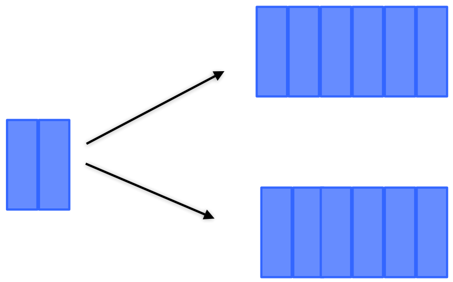
File can grow as a tree if it gets large

Directory



Efficient Scans

How to keep track of directories of all files?

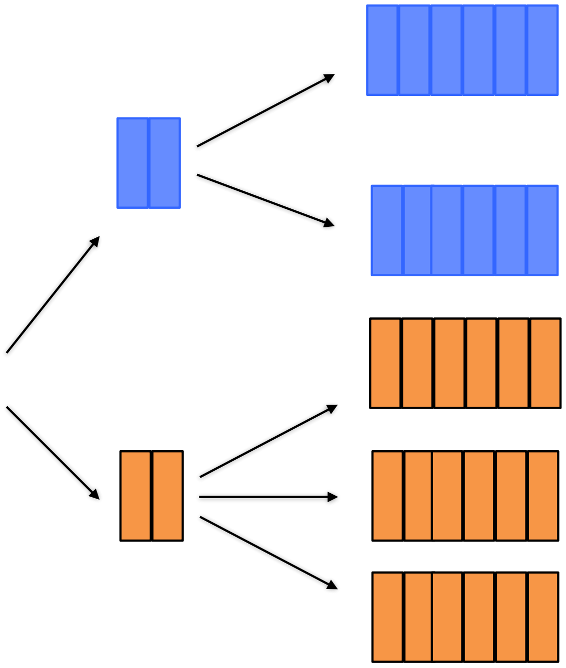


Efficient Scans

How to keep track of directories of all files?

Catalog

Name	Data type	Size (Bytes)	Start
Customers			
Orders			

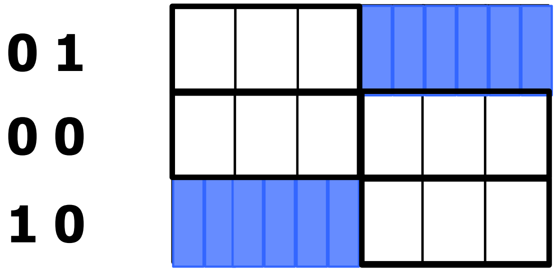


Efficient Scans

How to keep track of free pages/extents?

Solution 1: linked list (slower)

Solution 2: bitmap (takes space)



Operations

1. Scans e.g., select * from Customers
2. Deletes e.g., delete from Customers where name = “…”
3. Updates e.g., update Customers set email = “…” where name = “”
4. Insertions e.g., Insert into Customers (, , ,)

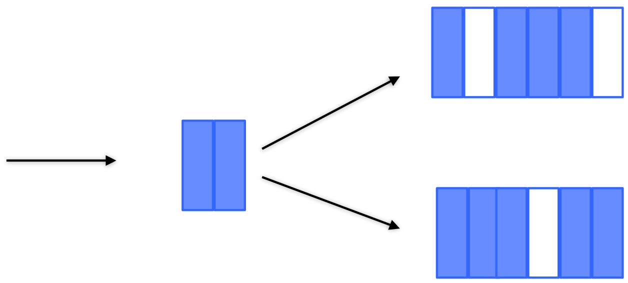
Supporting Deletes

e.g., delete from Customers where name = "…"

Simplest solution? Scan of the table. Creates "holes".

Catalog

Name	Data type	Size (Bytes)	Start
Customers			
Orders			



Cost: O(1) write and O(N/B) reads.

Operations

1. Scans
2. Deletes
3. **Updates**
4. Insertions

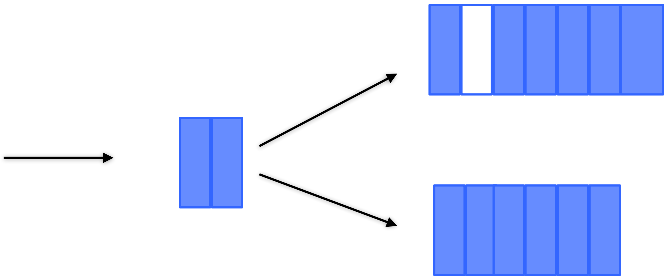
Supporting Updates

e.g., update Customers set email = “…” where name = “”

Scan and update. If newer version is too large, delete & reinsert

Catalog

Name	Data type	Size (Bytes)	Start
Customers			
Orders			



Cost: O(1) write and O(N/B) reads

Operations

1. Scans
2. Deletes
3. Updates
4. **Insertions**

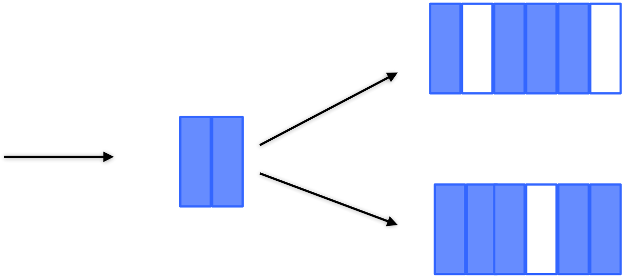
Supporting Insertions

e.g., Insert into Customers (, , ,)

Solutions?

Catalog

Name	Data type	Size (Bytes)	Start
Customers			
Orders			

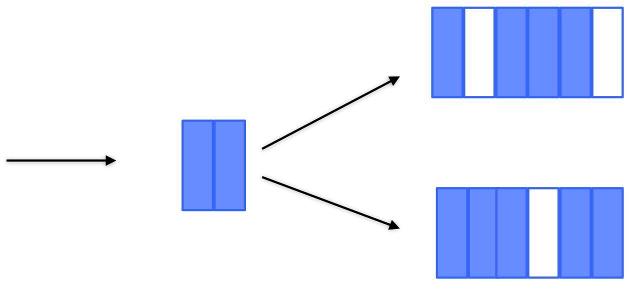


Supporting Insertions

(1) Scan & find space. Cost: O(N/B) reads and O(1) write.

Catalog

Name	Data type	Size (Bytes)	Start
Customers			
Orders			



Supporting Insertions

(1) Scan & find space. Cost: O(N/B) reads and O(1) write.

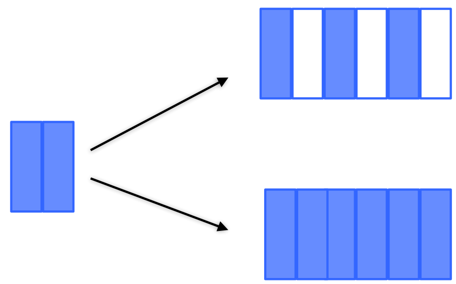
(2) Separate Linked list of pages with free space.

Cost: O(1) reads & O(1) write for fixed-sized entries

Cost: O(N/B) reads & O(1) write for variable-sized entries

Catalog

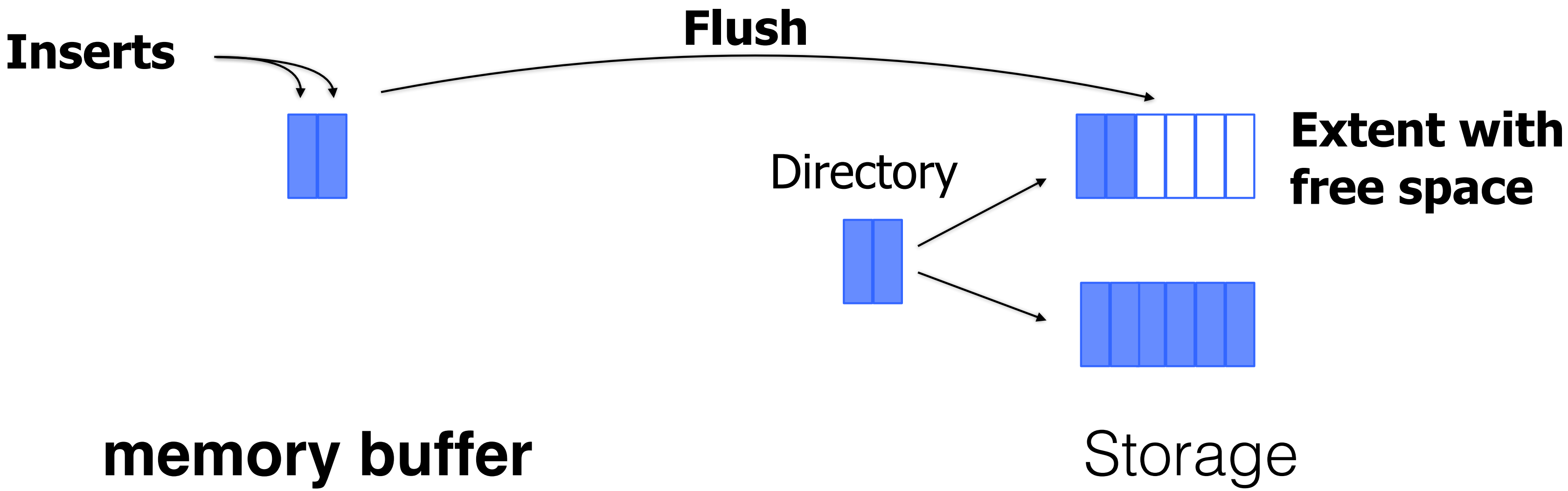
Name	Data type	Size (Bytes)	Start
Customers			
Orders			



Supporting Insertions

(3) buffer insertions in memory until a page fills up & append to extent

Cost: No reads and O(1/B) of a write



Supporting Insertions

(1) Scan & find space. Cost: O(N/B) reads and O(1) write.

(2) Separate Linked list of pages with free space.

Cost: O(1) reads & O(1) write for fixed-sized entries

Cost: O(N/B) reads & O(1) write for variable-sized entries

(3) buffer insertions in memory until a page fills up & append to extent

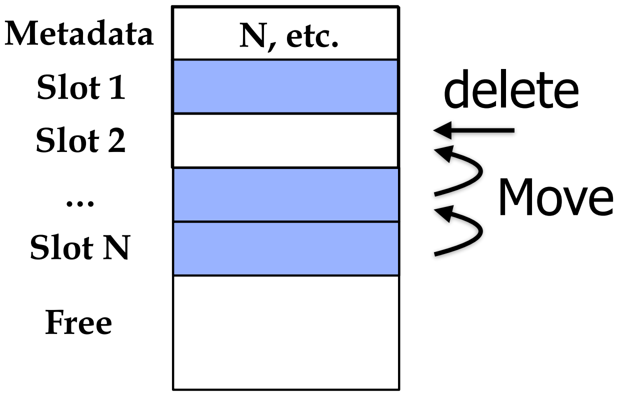
Cost: No reads and O(1/B) of a write

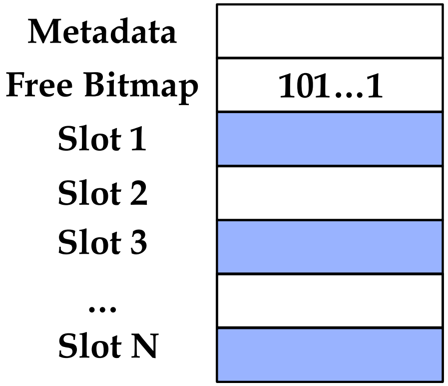
Internal Page Organization

Recall each page is 4-8 KB

Suppose rows are fixed-sized

How to organize rows within a slot?





Need to reorganize due to deletes

No reorganization, requires more space

Internal Page Organization

Recall each page is 4-8 KB

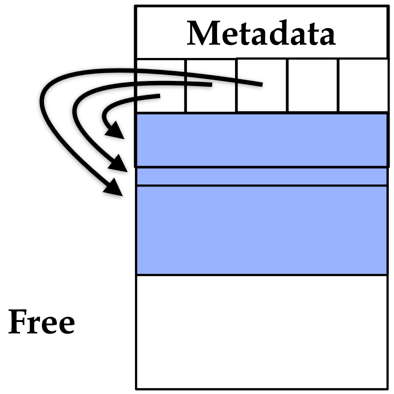
Suppose rows are **variable-length**

Solutions?

Internal Page Organization

Recall each page is 4-8 KB

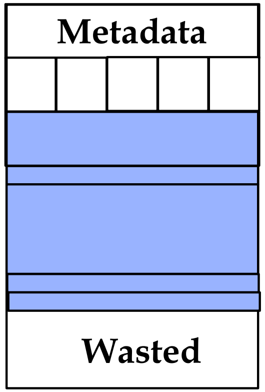
Suppose rows are **variable-length**



Internal Page Organization

Recall each page is 4-8 KB

Suppose rows are **variable-length**

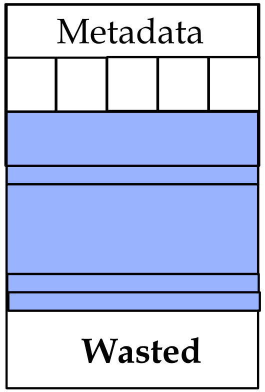


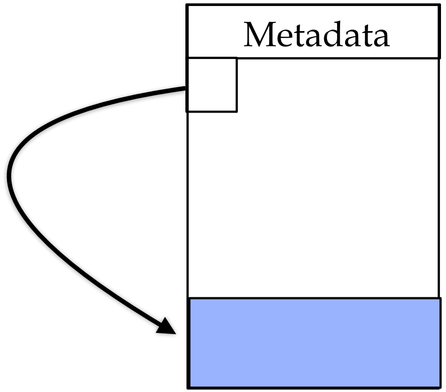
If entries are small, we waste
space at the end, or we must push
all content up to clear space

Internal Page Organization

Recall each page is 4-8 KB

Suppose rows are **variable-length**



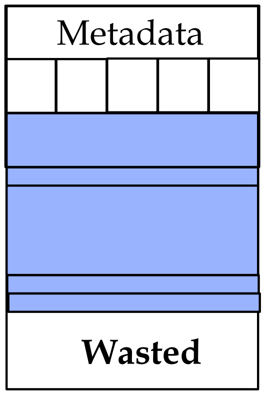


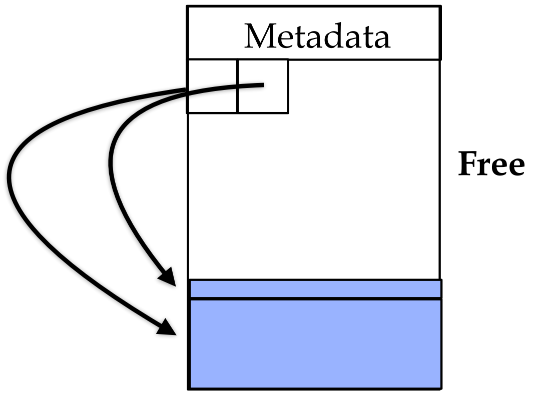
Store data from end of page

Internal Page Organization

Recall each page is 4-8 KB

Suppose rows are **variable-length**

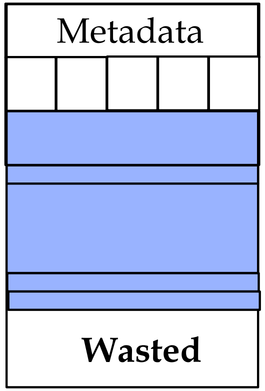


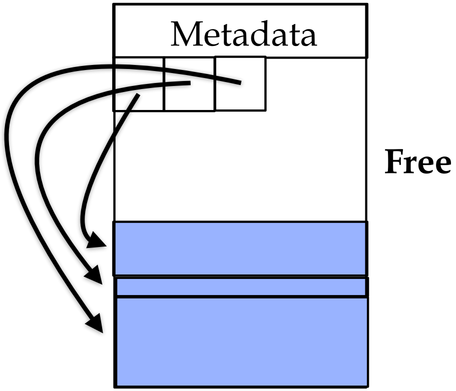


Internal Page Organization

Recall each page is 4-8 KB

Suppose rows are **variable-length**

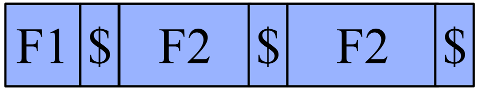




Minimal space wastage,
and no need to move data

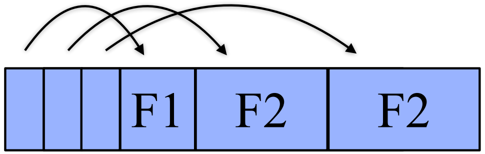
Variable-Sized Record Organization

Delimiters



Smaller
No random access

Pointers



More space
Random access (faster)