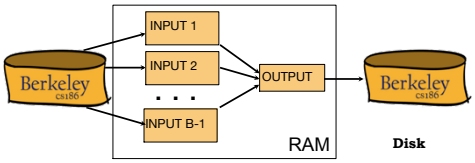
Review Session

**EXTERNAL SORTING**

## General External Merge Sort

- More than 3 buffer pages. How can we utilize them?
- To sort a file with N pages using B buffer pages:
  - Pass 0: use B buffer pages. Produce $\lceil N / B \rceil$ sorted runs of B pages each.
  - Pass 1, 2, …, etc.: merge B-1 runs.




Merging Runs

## Cost of External Merge Sort

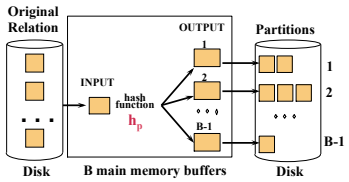
- Number of passes: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
- Cost = 2N * (# of passes)
- E.g., with 5 buffer pages, to sort 108 page file:
  - Pass 0: $\lceil 108 / 5 \rceil$ = 22 sorted runs of 5 pages each (last run is only 3 pages)
  - Pass 1: $\lceil 22 / 4 \rceil$ = 6 sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: 1 run => Sorted file of 108 pages

Formula check: $1 + \lceil \log_4 22 \rceil = 1+3$ → 4 passes √

**EXTERNAL HASHING**

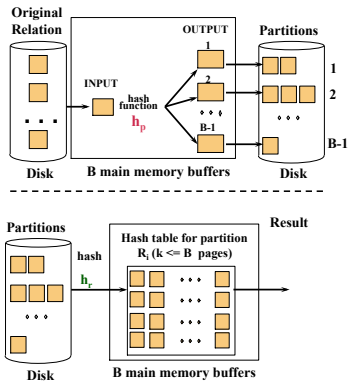
## Two Phases

- Partition: (Divide)



## Two Phases

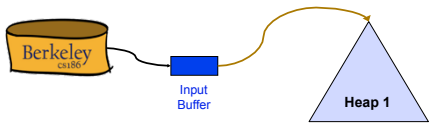
- Partition: (Divide)
- Rehash: (Conquer)




## TOURNAMENT SORT

## Tournament Sort (Heapsort)

First, load a heap with B-2 pages of records

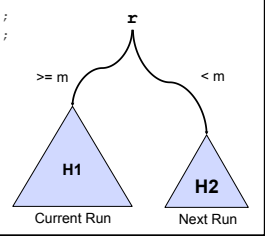



This is a priority heap, so it is in sorted order (hence called Heapsort)

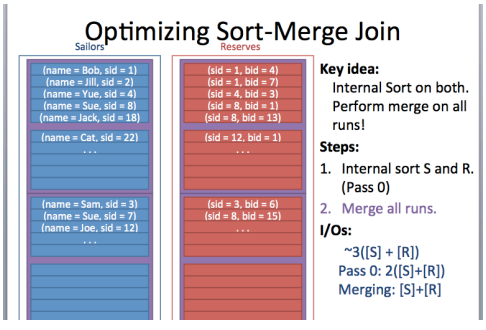
## Tournament Sort (2)

```
while (records left on input) {
      m = H1.removemin(); // get smallest value
      Output(m)           // put m in output buffer;
      if (H1 NOT empty)
         r = InputRecord()
         if (r < m)  H2.insert(r);
         else        H1.insert(r);

      else
         H1 = H2;  H2.reset();
         start new output run;
   }
```



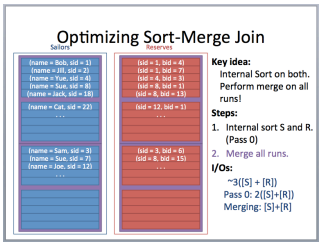

## TRICKY QUESTION




**Using General Merge Sort, when can you do Optimized Sort Merge join? Express with an inequality.**

STEP 1:
- # runs for R – ceil([R]/B)
- # run for S – ceil([S]/B)

STEP 2:
- We need to fit EACH run into buffer

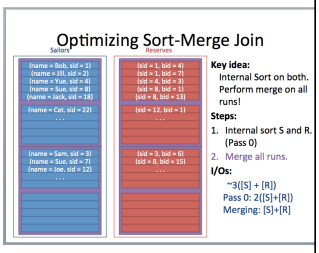



$$\text{ceil}([S]/B) + \text{ceil}([R]/B) <= B - 1$$

**Using tournament sort, when can you do Optimized Sort Merge join? Express with an inequality.**

STEP 1:
- # runs for R – ceil([R]/2(B-2))
- # run for S – ceil([S]/2(B-2))

STEP 2:
- We need to fit EACH run into buffer

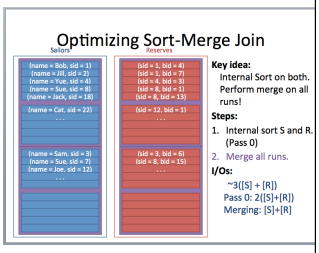



ceil([S]/2(B-2)) + ceil([R]/2(B-2)) <= B - 1

**NOW, let's assume [R] is equal to [S]. Express the same inequality with [R] and [B].**

STEP 1:
- # runs for R – ceil([R]/2(B-2))
- # runs for S – ceil([R]/2(B-2))

STEP 2:
- We need to fit EACH run into buffer




ceil([R]/2(B-2)) + ceil([R]/2(B-2)) <= B - 1

## Tournament Sort + Optimized Sort Merge

ceil([R]/(B-2)) < B

[R] < (B)(B-2)

As B becomes very large, we can approximate this:
SQRT([R]) < B

## SQL

## Vitamin Question

```
SELECT sname
FROM
    (SELECT sid
    FROM Reserves
    EXCEPT
        (SELECT sid
        FROM
            (SELECT Reserves.sid, PinkBoats.bid
            FROM Reserves,
            (SELECT bid FROM Boats WHERE color='pink') PinkBoats
            EXCEPT
                (SELECT sid, bid FROM Reserves)
            )
        )
    ) R, Sailors S
WHERE R.sid = S.sid;
```

## Vitamin Question Step 1

```
### STEP 1

SELECT sname
FROM
    (SELECT sid
    FROM Reserves
    EXCEPT
        (SELECT sid
        FROM
            (SELECT Reserves.sid, PinkBoats.bid
            FROM Reserves, [All possible pink boats] PinkBoats
            EXCEPT
                [All (sid, bid) of reservations]
            )
        )
    ) R, Sailors S
WHERE R.sid = S.sid;
```

```
#### STEP 1b
...
    (SELECT Reserves.sid, PinkBoats.bid
    FROM Reserves, [All possible pink boats] PinkBoats
    EXCEPT
        [All (sid, bid) of reservations]
    )
...
```

[All sailors in the reservation table X All pinkboats ]

- [All (sailors, pink boat) existing reservations]

= [All (sailor_R, pink boats) combos that do not exist]

...(sailor_R are all sailors that have made a reservation)

## Step 2

```
#### STEP 2

SELECT sname
FROM
    (SELECT sid
    FROM Reserves
    EXCEPT
        (SELECT sid
        FROM [All (sailor_R, pink boats) combos that do not exist]
        )
    )
)
```

*do not exist in Reservations

## Step 3

```
#### STEP 3

SELECT sname
FROM
    (SELECT sid
    FROM Reserves
    EXCEPT
        All sailors_R that have not reserved all pink boats
    )
)
```

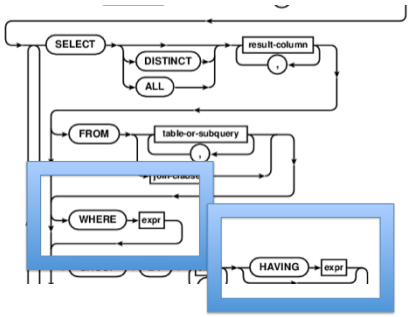
## Step 4

```
#### STEP 4

SELECT sname
FROM
    All Sailors that have reserved all pink boats
)
```

This actually only true sometimes (it has some implicit assumptions) – what happens when certain tables are NULL?

## HAVING



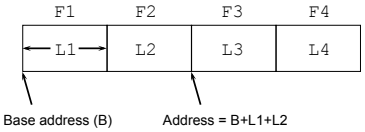

*Record id = <page id, slot #>.*

**FORMATS**

Fixed vs Variable Length

## RECORD FORMATS

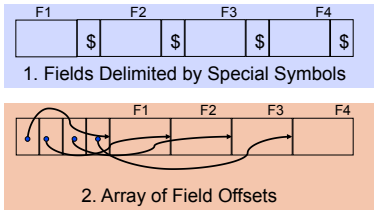
## Record Formats: Fixed Length




- Field types same for all records in a file.
  - Type info stored separately in *system catalog*.
- Finding *i'th* field done via arithmetic like arrays

## Record Formats: Variable Length

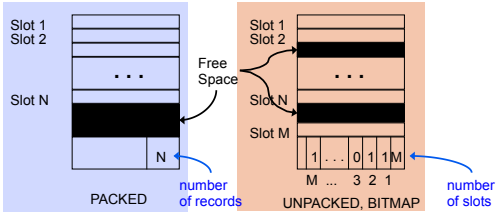
- Two alternative formats (# fields is fixed):




Second offers direct access to i'th field, efficient storage of *nulls* (special *unknown* value); small directory overhead.

## PAGE FORMATS

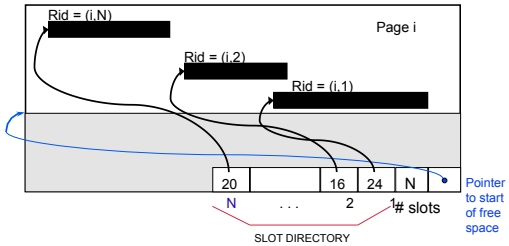
## Page Formats: Fixed Length Records




*Record id = <page id, slot #>.*

In first alternative, moving records for free space management *changes rid*; may be problematic!

## "Slotted Page" Format: Variable Length Records

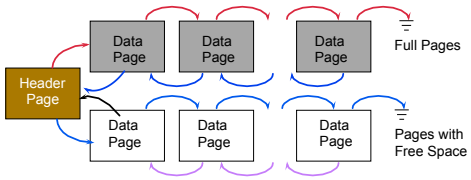



Can move records on page without changing rid! So, attractive for fixed-length records too.

## Unordered (Heap) Files

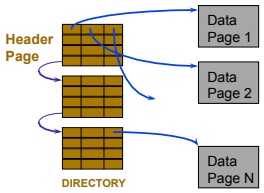
- Collection of records in no particular order.
- As file shrinks/grows, disk pages (de)allocated
- To support record level operations, we must:
  - keep track of the *pages* in a file
  - keep track of *free space* on pages
  - keep track of the *records* on a page
- There are many alternatives for keeping track of this.
  - We'll consider 2

## Heap File Implemented as a List




- Header page ID and Heap file name stored elsewhere
  - Database "catalog"
- Each page contains 2 "pointers" plus data
  - Problem for multi-page objects (blobs) – how to read blobs?

## Better: Use a Page Directory




- Directory entries include #free bytes on the page.
- Directory is a collection of pages; linked list implementation is just one alternative.
  - *Much smaller than linked list of all HF pages*!
  - Can also point to groups of pages (say 64k chunks)