Discussion 9

Joins & Project 4 Intro

EECS 484

Logistics

- **HW 4** due **Today** at 11:45 PM ET
- **HW 5** due **Nov 15th** at 11:45 PM ET
- **Project 4** released, due **Nov 22nd** at 11:45 PM ET

Joins

Joins

- Powerful tool in SQL
  - A join B join C …
  - But how does this actually work?
  - Different ways to represent a join
    - Simple/Stupid Nested Loop
    - Block Nested Loop
    - Index Nested Loop
      - See lecture slides
    - Sort-Merge
    - Grace Hash Join
      - Will be your new best friend

Problem Setup

- We would like to join R with S using the column ID (both share)
- R and S are some relations
- m = Number of tuples in R, n = Number of tuples in S
- M = Number of pages in R, N = Number of pages in S
- Sometimes we use |T| to represent *number of pages* in T and ||T|| to represent *number of tuples* in T
- Will use this to compute number of IOs in each scheme

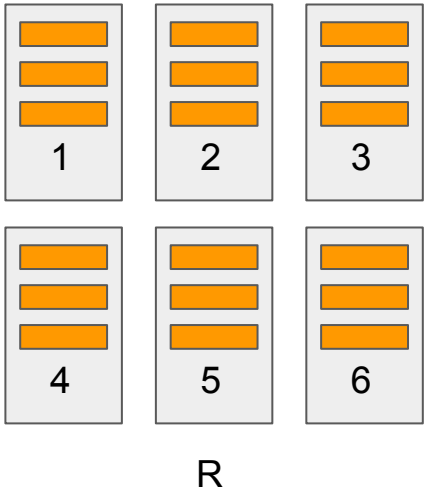
Stupid Nested Loop Join

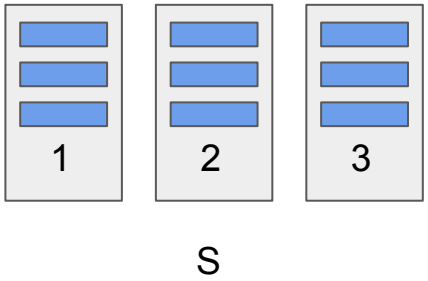
- For each tuple r in R
  - For each tuple s in S
    - If r.ID=s.ID
      - Add <r, s> to the final result
- Super simple!
  - Intuitively makes sense (hopefully)
- Performance
  - IO Cost = Iterate through all M pages of R + Iterate through all N pages of S m times
    - M + N*m

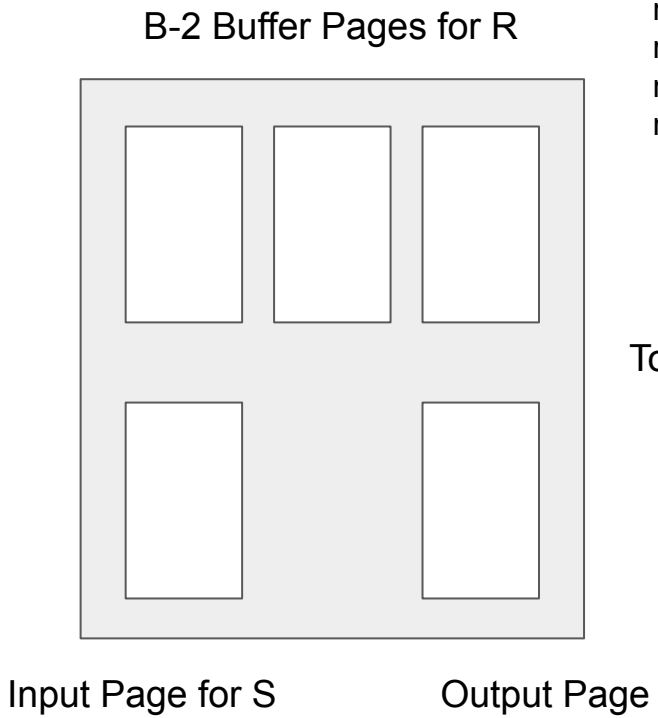
Block Nested Loop Join

- Use B buffer pages
  - Read B-2 pages of R at a time
  - 1 page of S at a time
  - 1 page for output
- For each block of B-2 pages in R
  - For each Page of S
    - For each tuple in the B-2 pages of R
      - For each tuple in the page of S
        - Do the join
- 4 loops???
  - IO Cost = Load all M pages of R but load each N pages of S only (M/(B-2)) times
    - M+N*ceiling(M/(B-2))

```
for(int w=0; w<0; w++)
{
    for(int e=0; e<0; e++)
    {
        for(int l=0; l<0; l++)
        {
            for(int c=0; c<0; c++)
            {
                for(int o=0; o<0; o++)
                {
                    for(int m=0; m<0; m++)
                    {
                        for(int e=0; e<0; e++)
                        {
                            for(int t=0; t<0; t++)
                            {
                                for(int o=0; o<0; o++)
                                {
                                    for(int h=0; h<0; h++)
                                    {
                                        for(int e=0; e<0; e++)
                                        {
                                            for(int l=0; l<0; l++)
                                            {
                                                for(int l=0; l<0; l++)
                                                {
                                                    System.out.pr
```

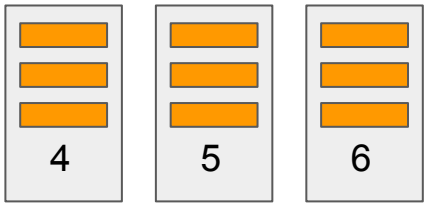




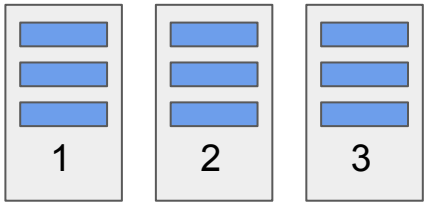


Note: the color of a bar doesn't represent the value of the record. It's used to distinguish records from 2 relations (Blue means it's from S while yellow means it's from R).

Total B=5 Buffer Pages

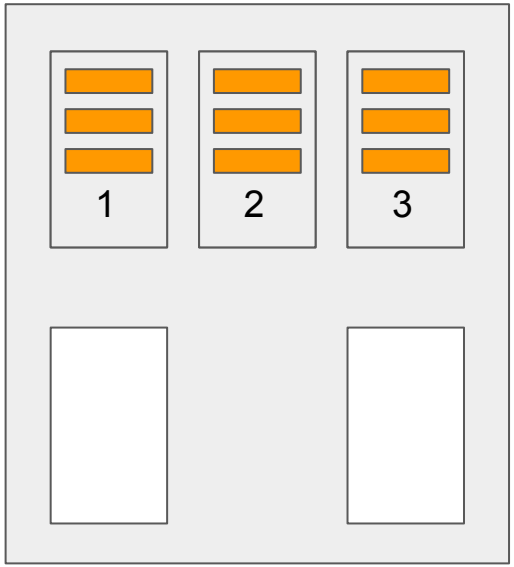


R



S

B-2 Buffer Pages for R

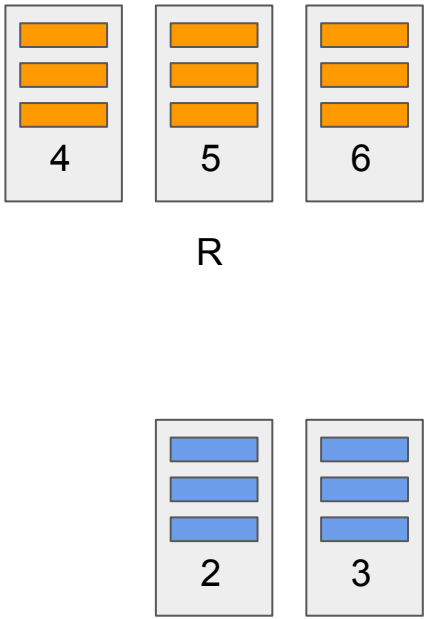


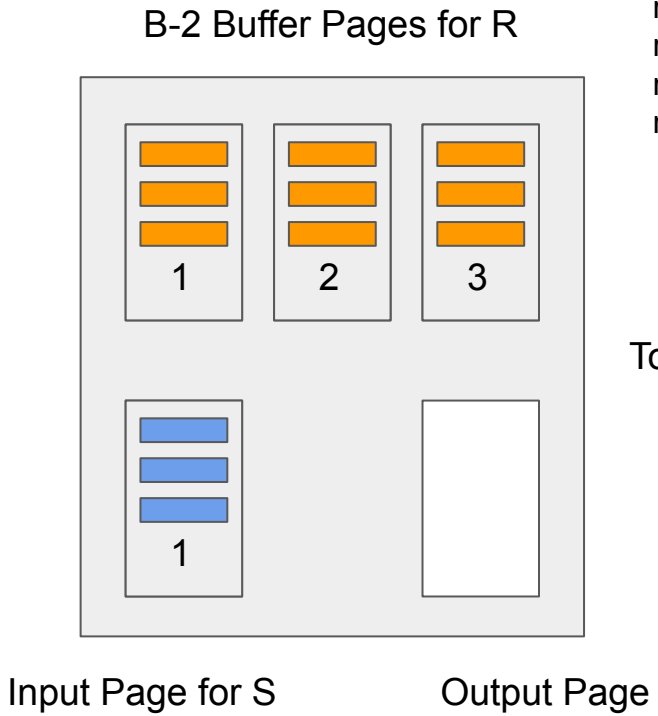
Input Page for S

Output Page

Note: the color of a bar doesn't represent the value of the record. It's used to distinguish records from 2 relations (Blue means it's from S while yellow means it's from R).

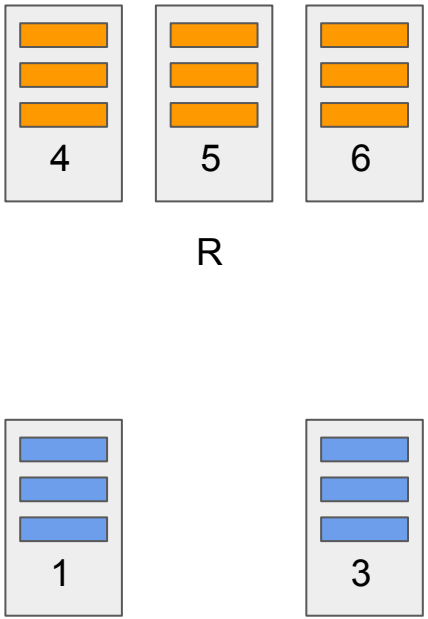
Total B=5 Buffer Pages

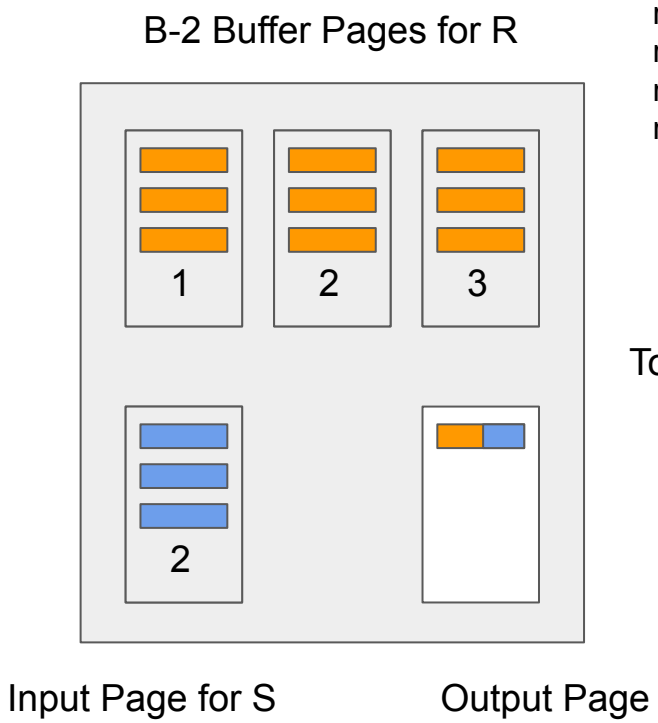




Note: the color of a bar doesn't represent the value of the record. It's used to distinguish records from 2 relations (Blue means it's from S while yellow means it's from R).

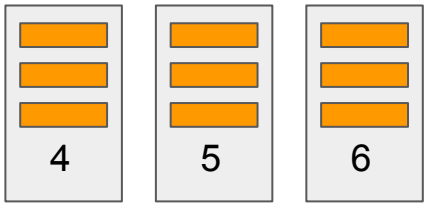
Total B=5 Buffer Pages



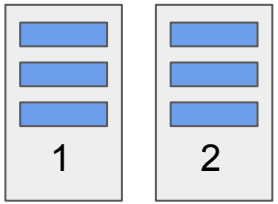


Note: the color of a bar doesn't represent the value of the record. It's used to distinguish records from 2 relations (Blue means it's from S while yellow means it's from R).

Total B=5 Buffer Pages

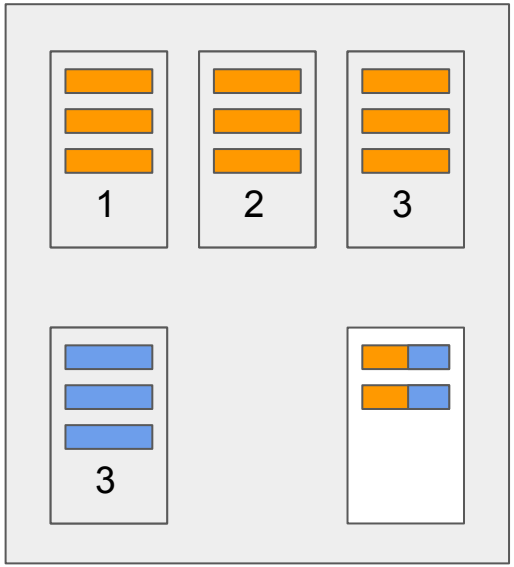


R



S

B-2 Buffer Pages for R

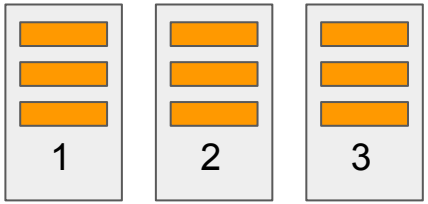


Input Page for S

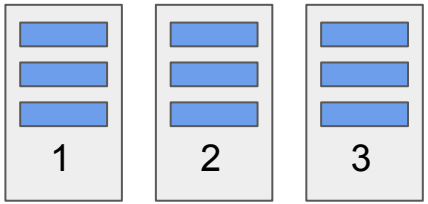
Output Page

Note: the color of a bar doesn't represent the value of the record. It's used to distinguish records from 2 relations (Blue means it's from S while yellow means it's from R).

Total B=5 Buffer Pages

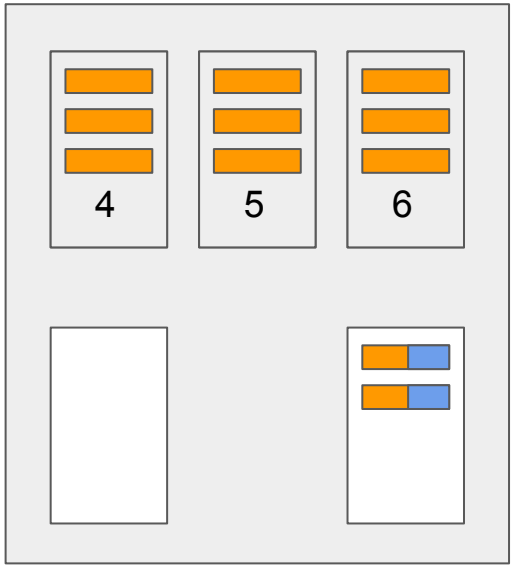


R



S

B-2 Buffer Pages for R



Input Page for S

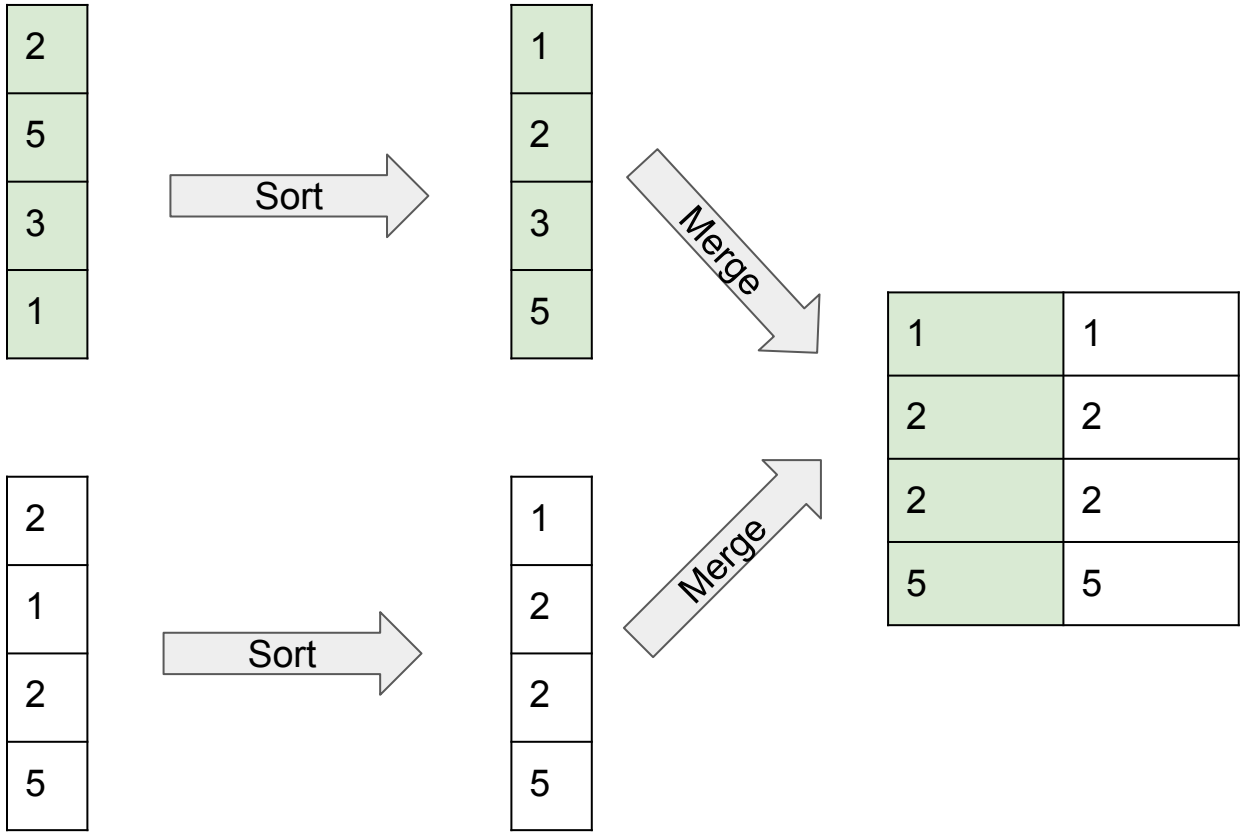
Output Page

Note: the color of a bar doesn't represent the value of the record. It's used to distinguish records from 2 relations (Blue means it's from S while yellow means it's from R).

Total B=5 Buffer Pages

Sort-Merge Join

- **Sort** both relations on the join attribute
  - May omit one final passe if the number of sorted runs from two relations is smaller than the number of buffer pages
- Look for qualifying by **merging** the two relations
- I/O Cost:
  - Sort R: 2M * #passes needed to sort R
  - Sort S: 2N * #passes needed to sort S
  - Typical merge cost: M + N
  - Worst merge cost: M * N
    - When?
  - Typical Sort-Merge Join cost:
    - 2M * 2 + 2N * 2 + M + N = 5(M + N)
    - Possible to finish in 3(M + N)! When?

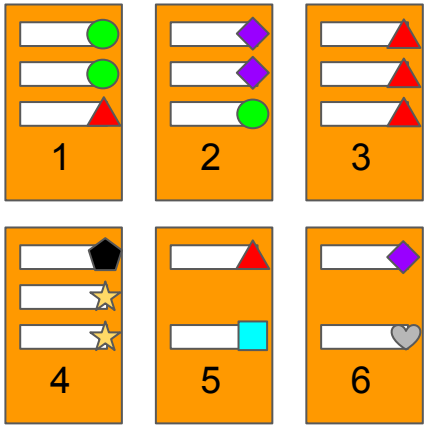

2
5
3
1
Sort
1
2
3
5
Merge
1 1
2 2
2 2
5 5
2
1
2
5
Sort
1
2
2
5
Merge

Grace Hash Join

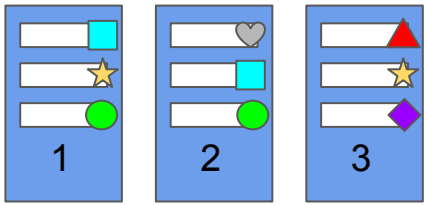
- What if we used advanced data structures(hash tables) to help us sort?
  - If h(r) = h(s) then very likely that r = s!
    - Assume we take the hash on the columns we are joining on
- 2 phases
  - Phase 1: Build
    - Put each tuple r and s into partitions
    - Any two elements in the same partition share a hash
    - Might have multiple partition rounds - need to enforce that each partition fits into buffer space
  - Phase 2: Probe
    - Load in each partition of the smaller relation (assume R for this case) and rehash
    - Read the corresponding partition for the larger relation (assume S for this case) and rehash
    - Compare the tuples and add to output if they match

Grace Hash Join

- IO Cost: (2p+1)(M+N) where p is number of partition rounds
  - Partition = 2p(M+N) - Read + Write each page across both relations once
  - Probe = M+N - Read each page across both relations once more

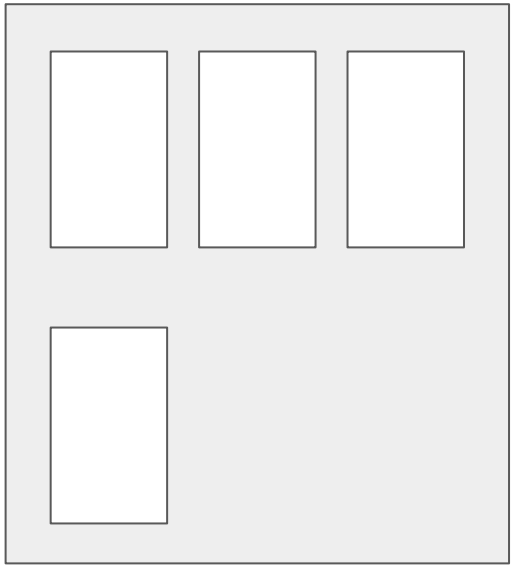


R



S

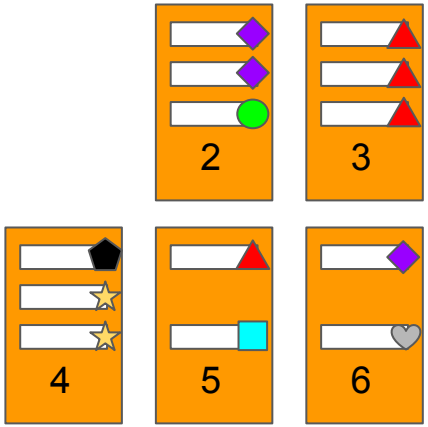
B-1 Buffer Pages for Partitions



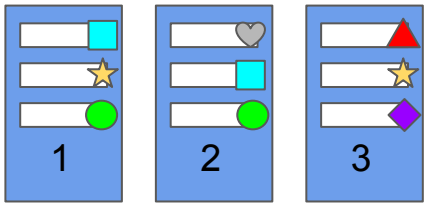
Input Page

Total B=4 Buffer Pages

Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

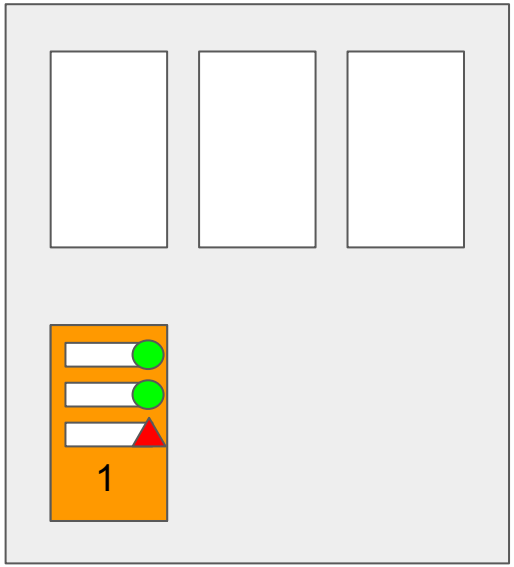


R



S

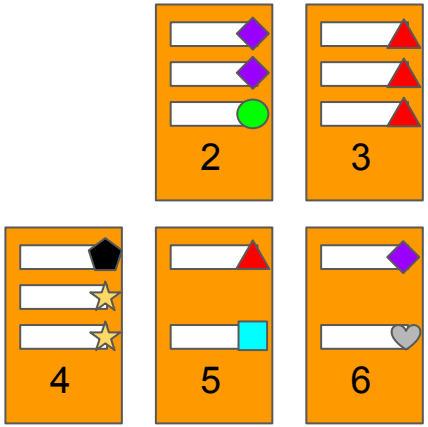
B-1 Buffer Pages for Partitions



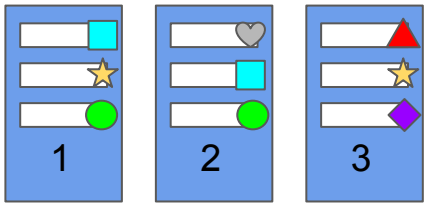
Input Page

Total B=4 Buffer Pages

Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

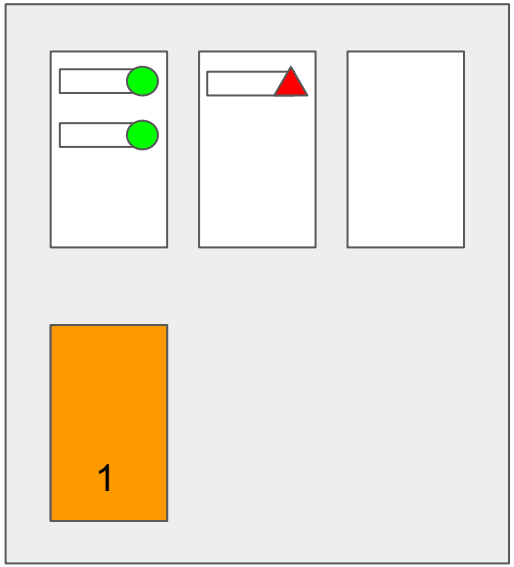


R



S

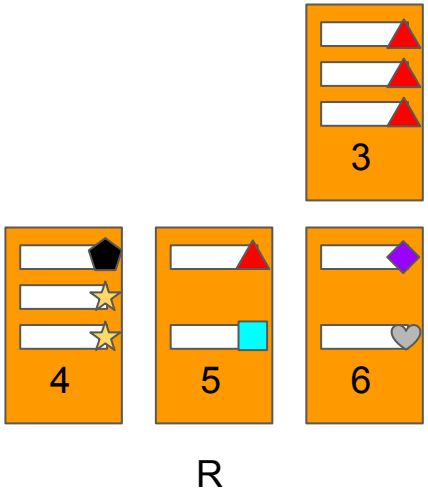
B-1 Buffer Pages for Partitions

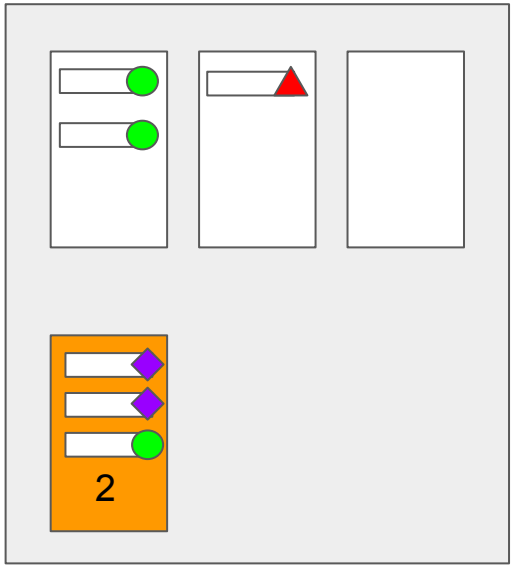


Input Page

Total B=4 Buffer Pages

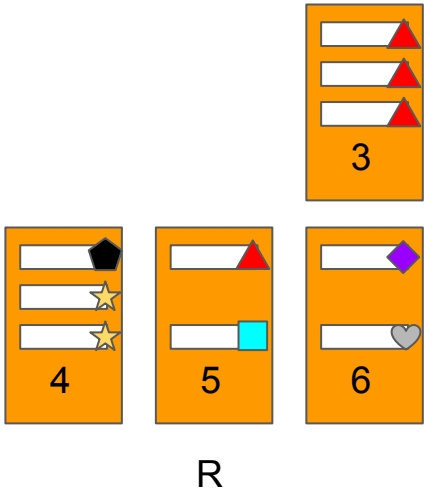
Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

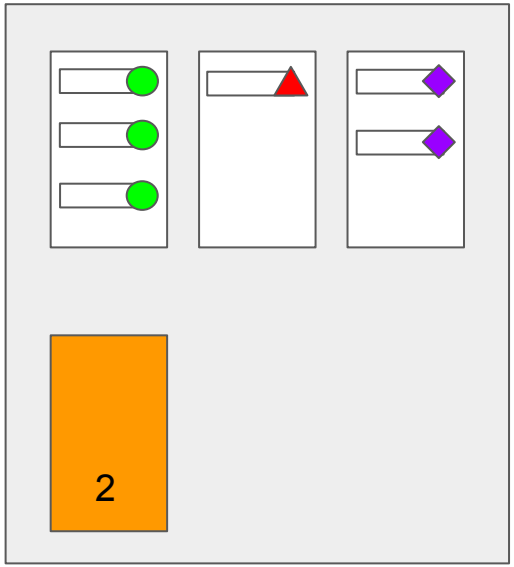




Total B=4 Buffer Pages

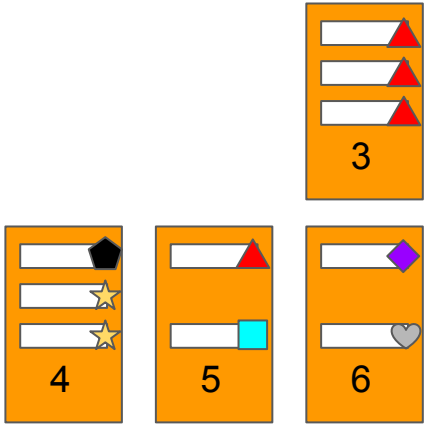
Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

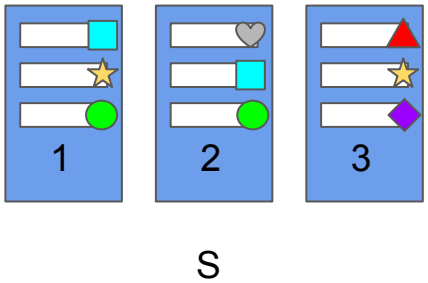




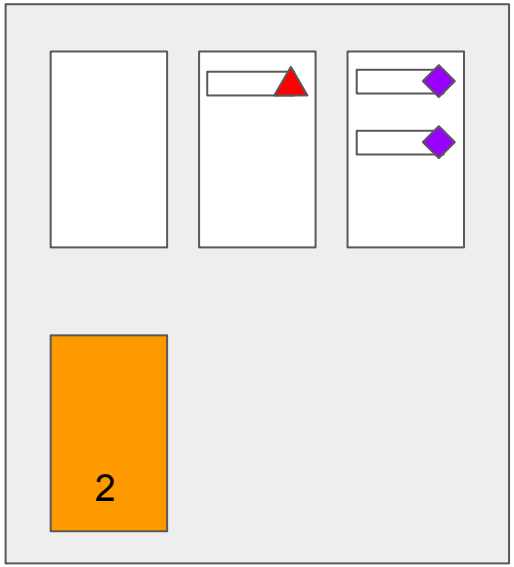
Total B=4 Buffer Pages

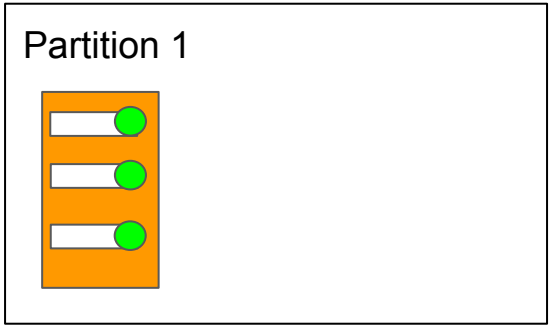
Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley





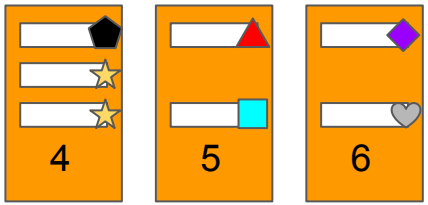
B-1 Buffer Pages for Partitions



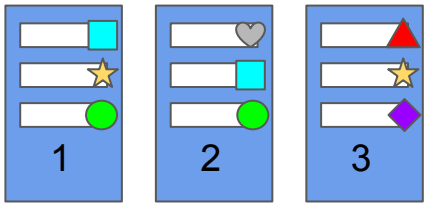


Total B=4 Buffer Pages

Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

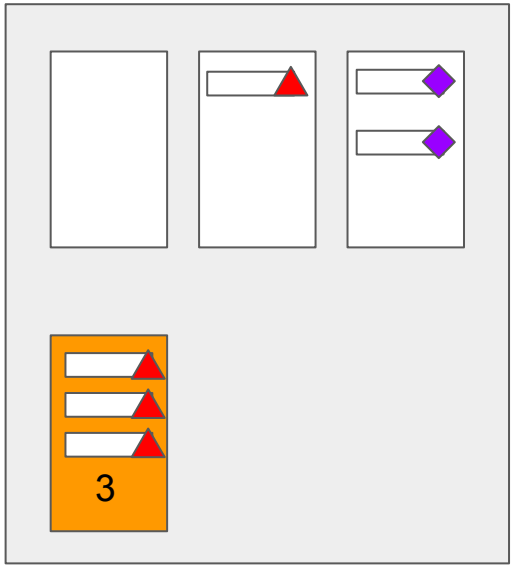


R



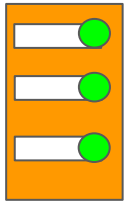
S

B-1 Buffer Pages for Partitions



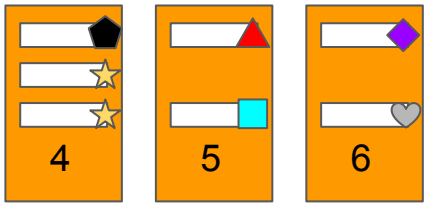
Input Page

Partition 1

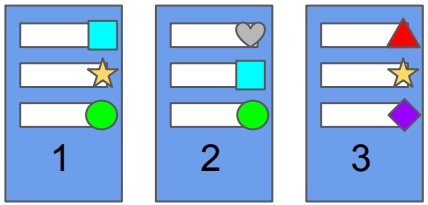


Total B=4 Buffer Pages

Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

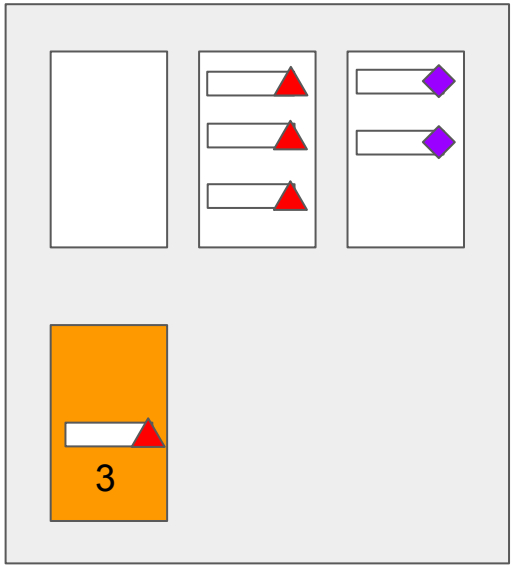


R



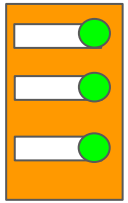
S

B-1 Buffer Pages for Partitions



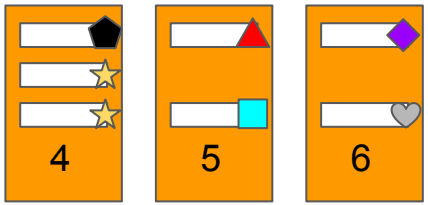
Input Page

Partition 1

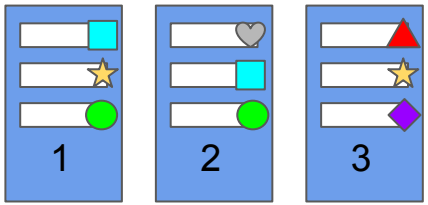


Total B=4 Buffer Pages

Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

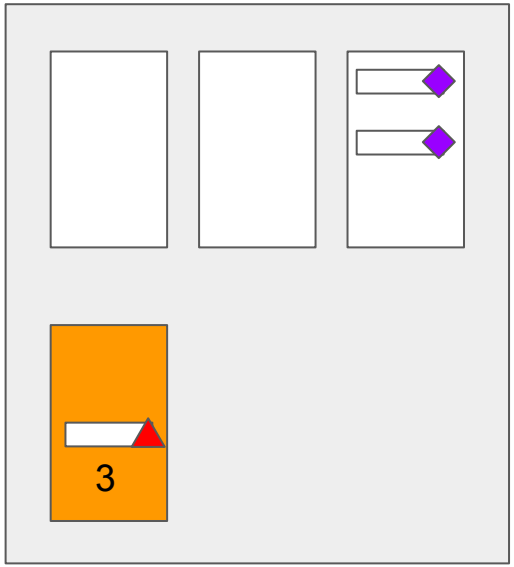


R

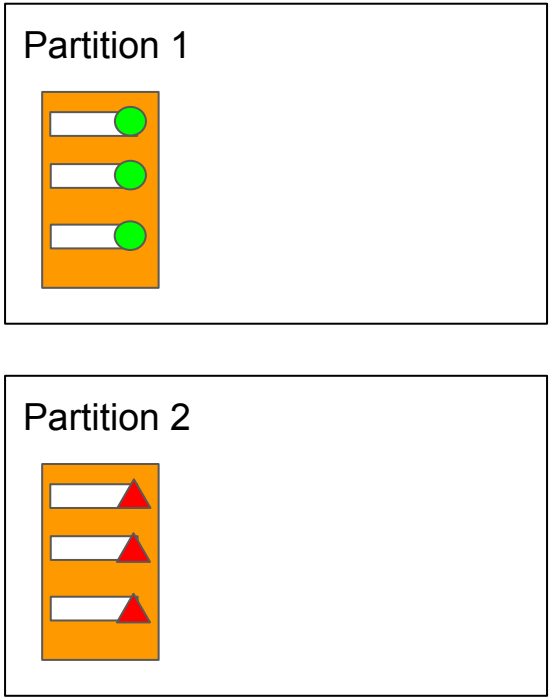


S

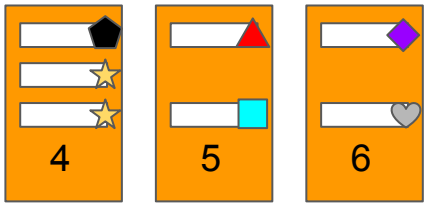
B-1 Buffer Pages for Partitions



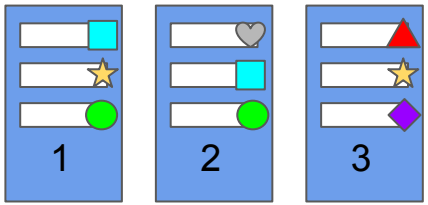
Input Page



Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

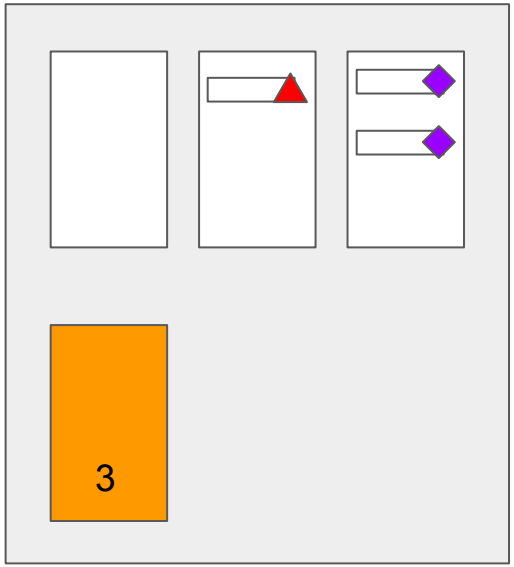


R

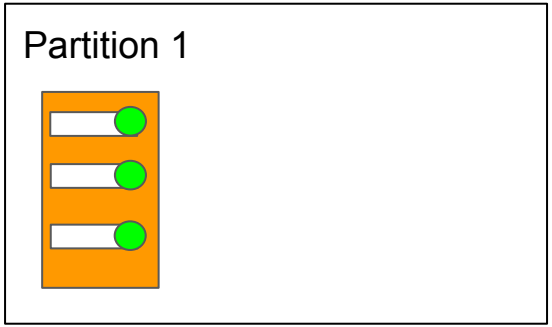


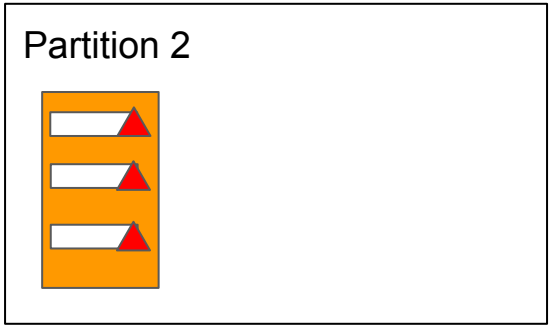
S

B-1 Buffer Pages for Partitions

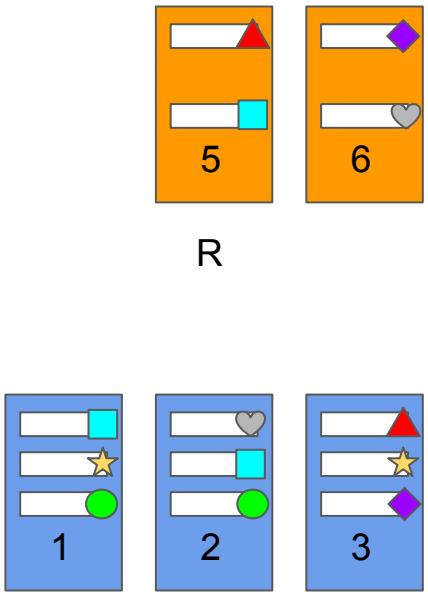


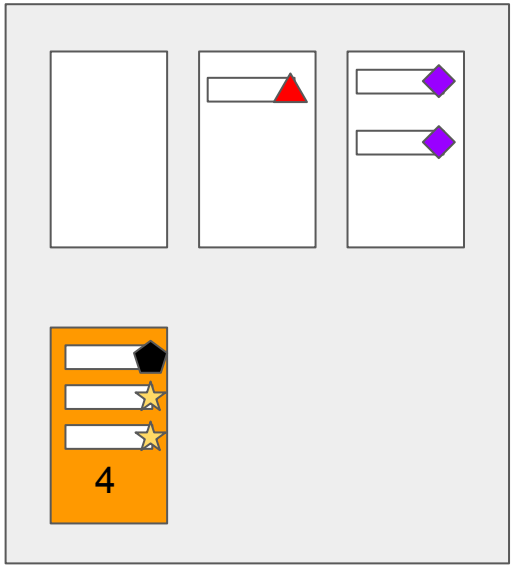
Input Page

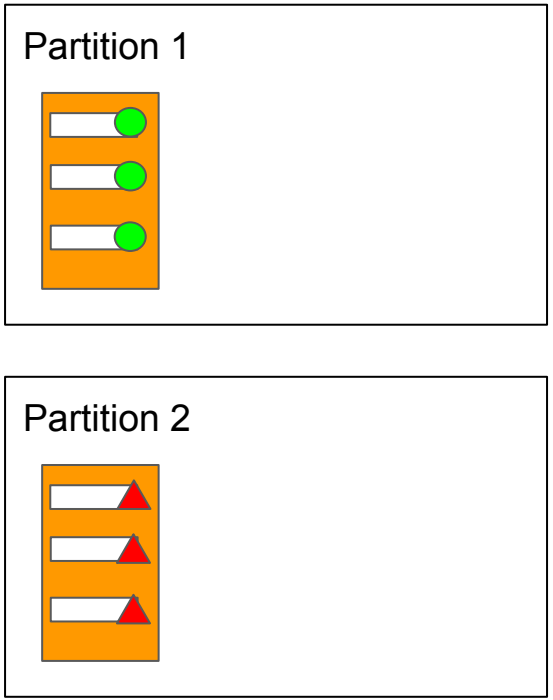




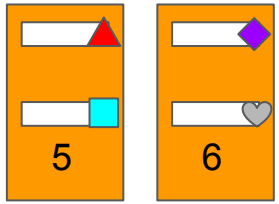
Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley



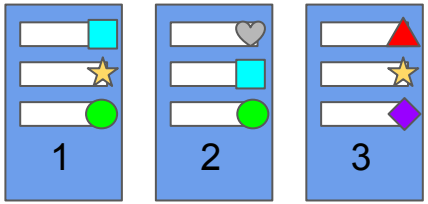




Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

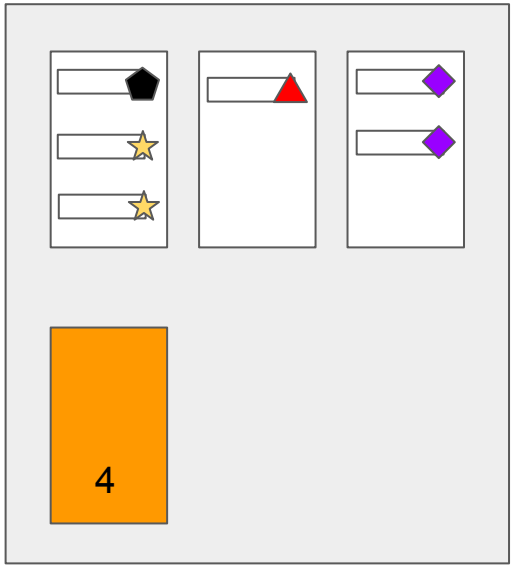


R

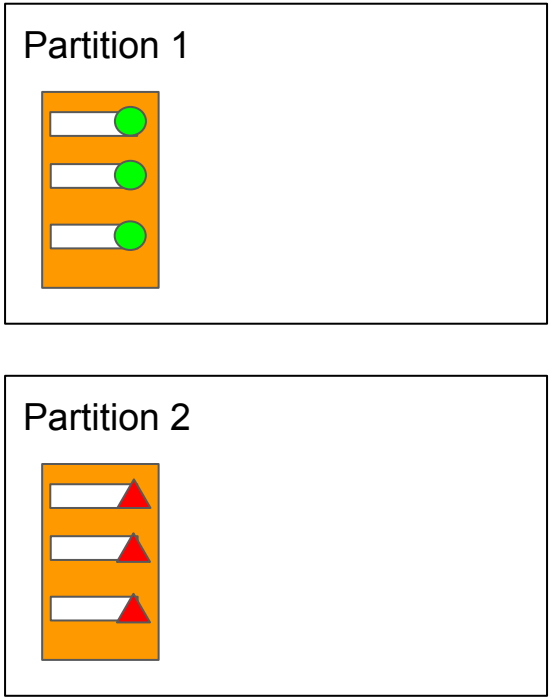


S

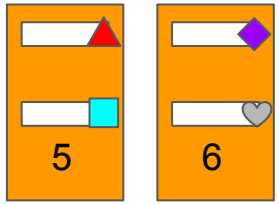
B-1 Buffer Pages for Partitions



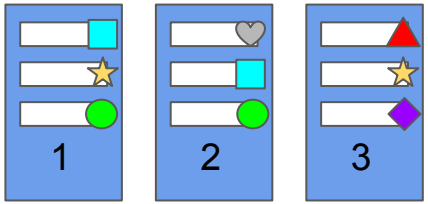
Input Page



Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

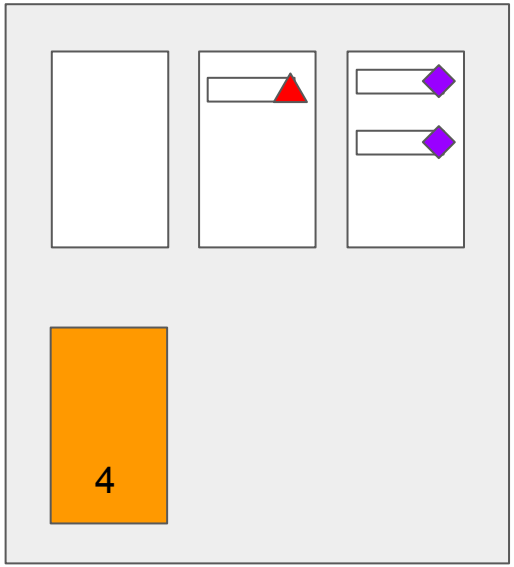


R

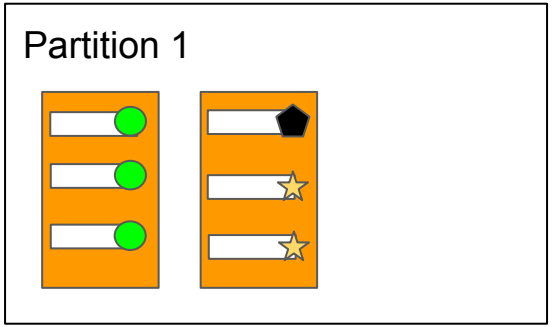


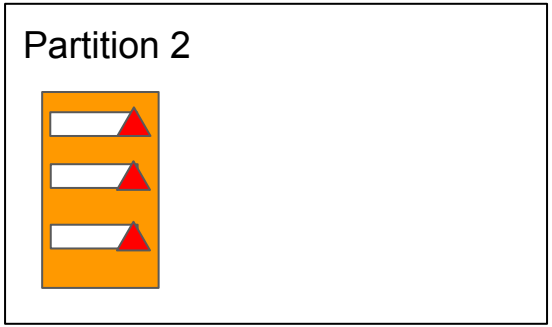
S

B-1 Buffer Pages for Partitions

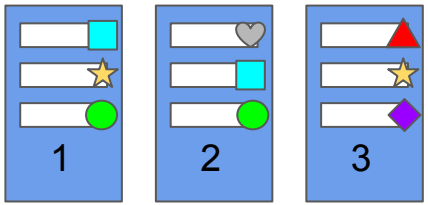


Input Page



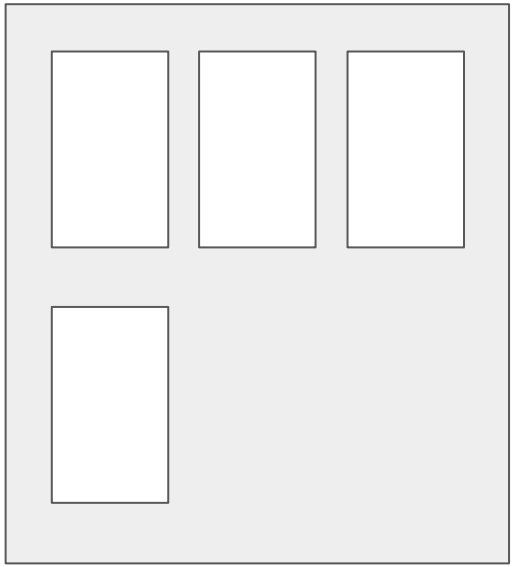


Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

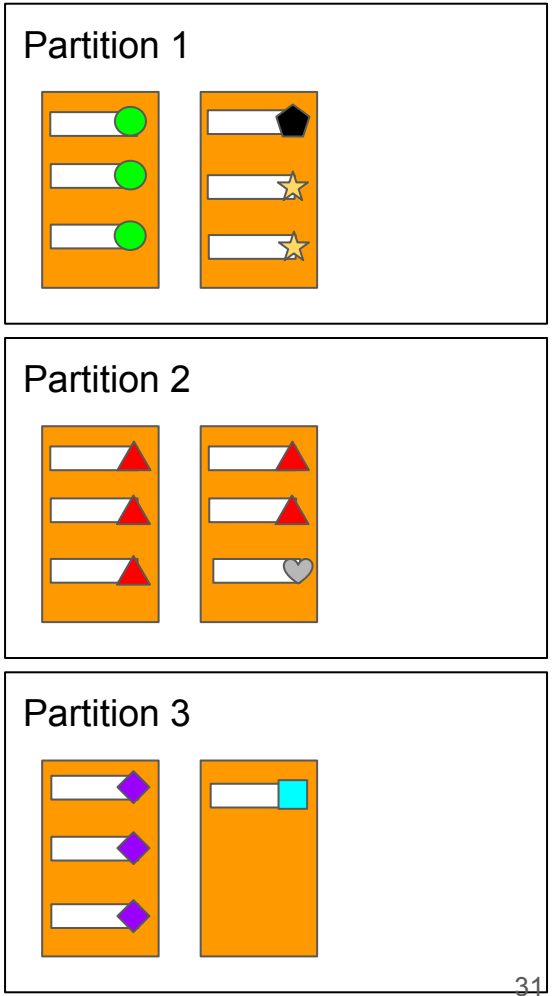


S

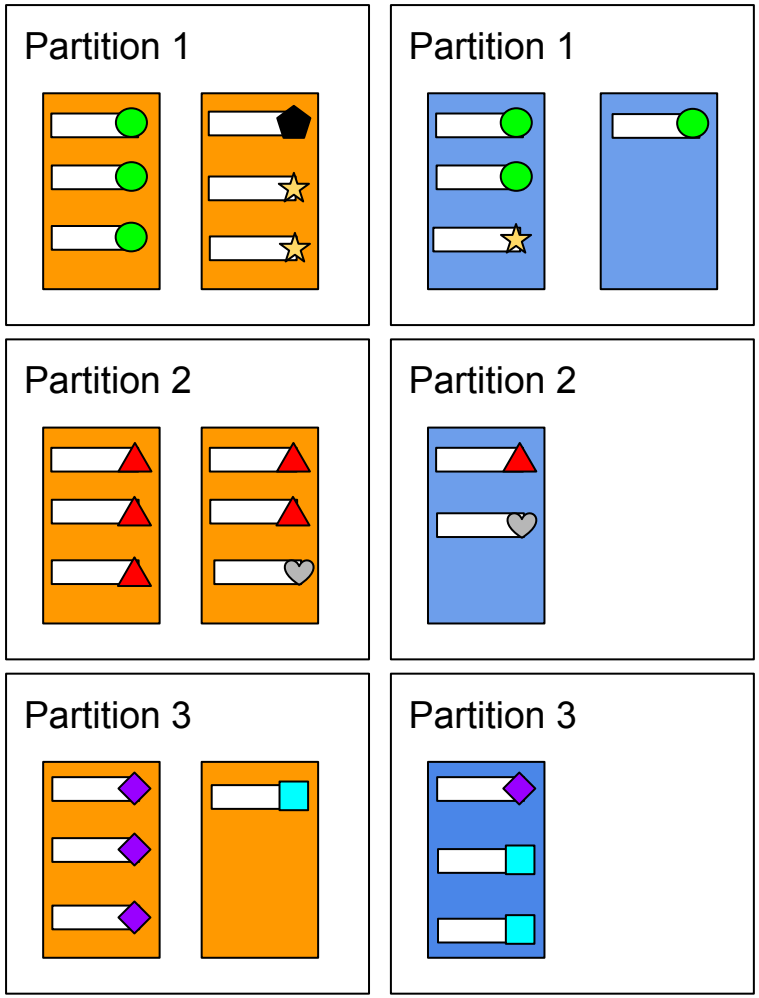
B-1 Buffer Pages for Partitions

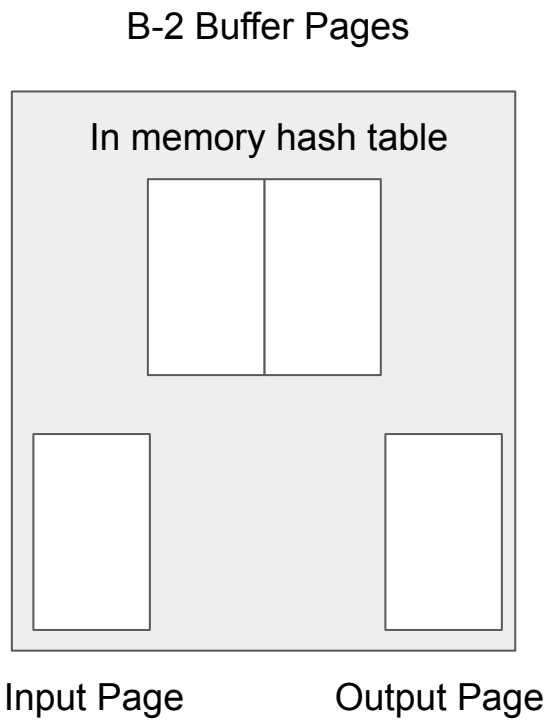


Input Page

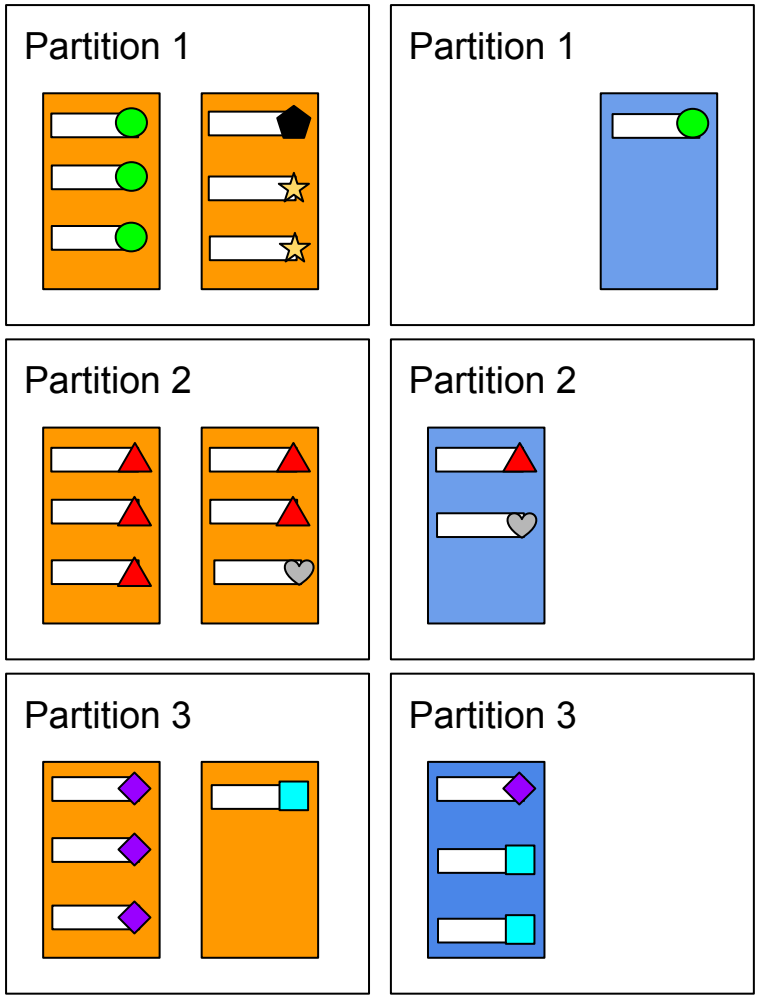


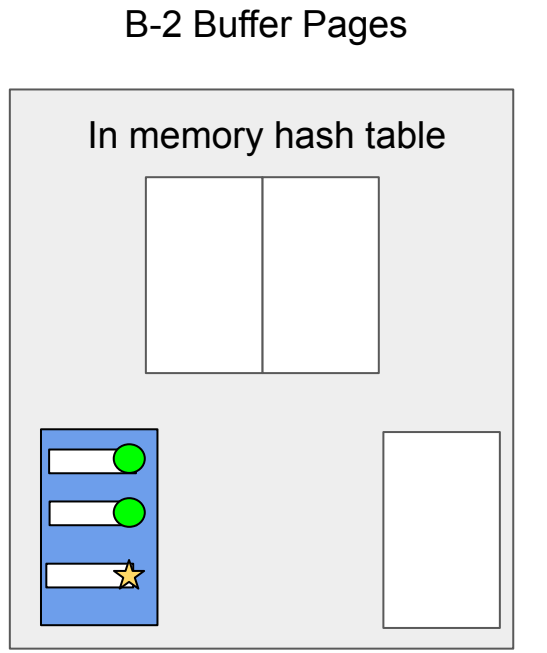
Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley



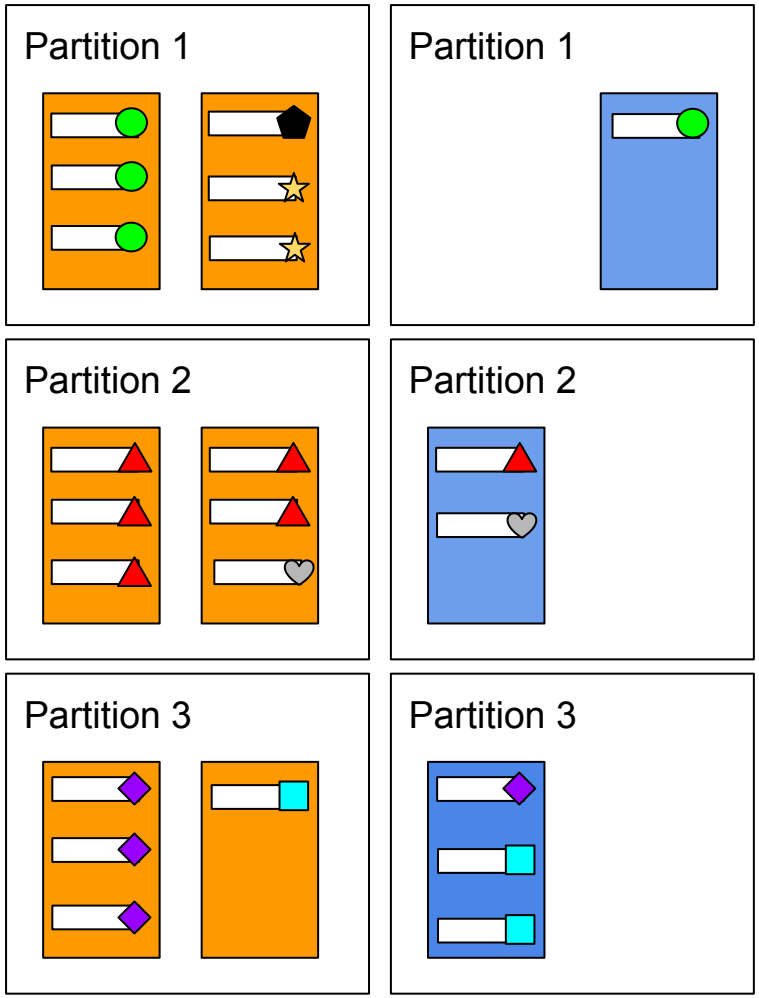


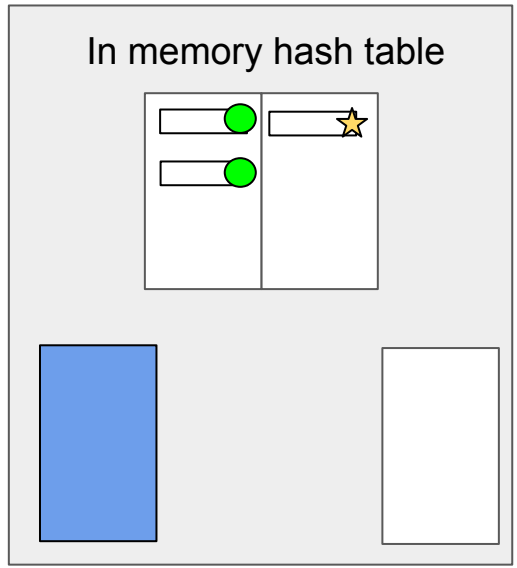
Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley





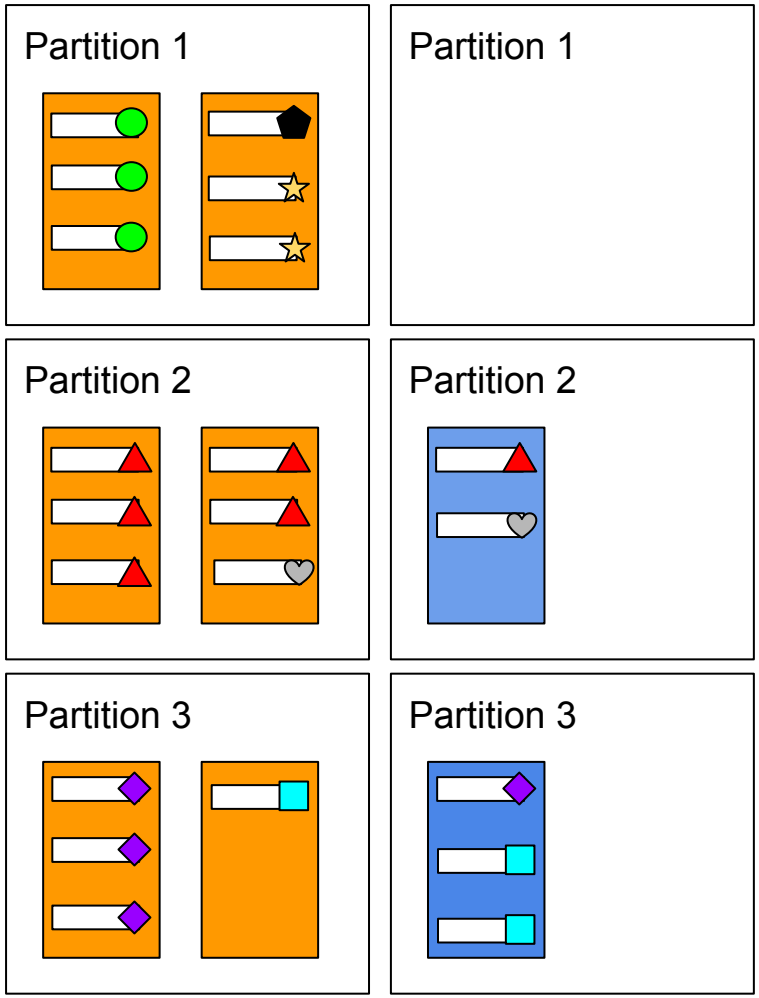
Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

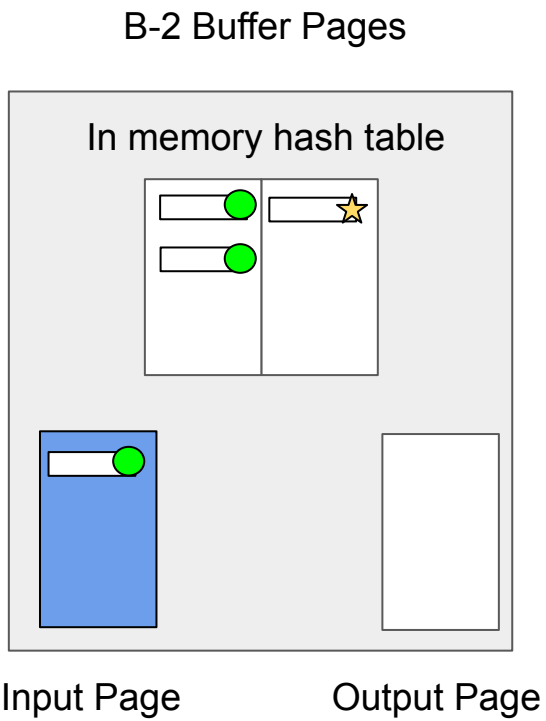




Note: the sections in the hash table represent buckets, rather than pages.

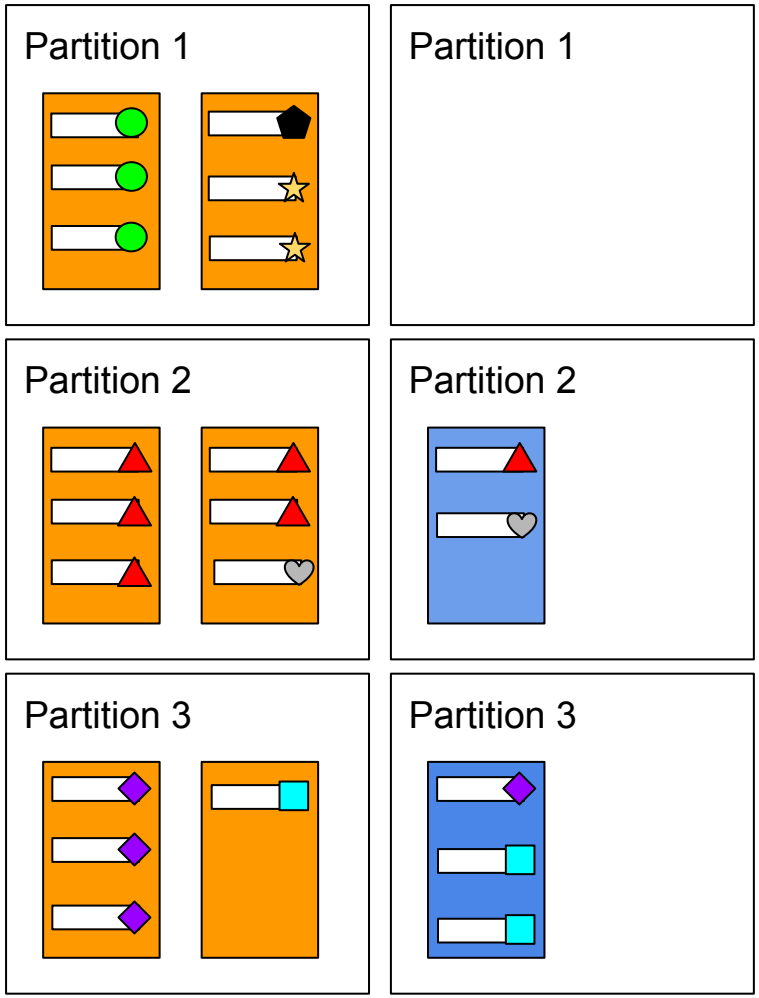
Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

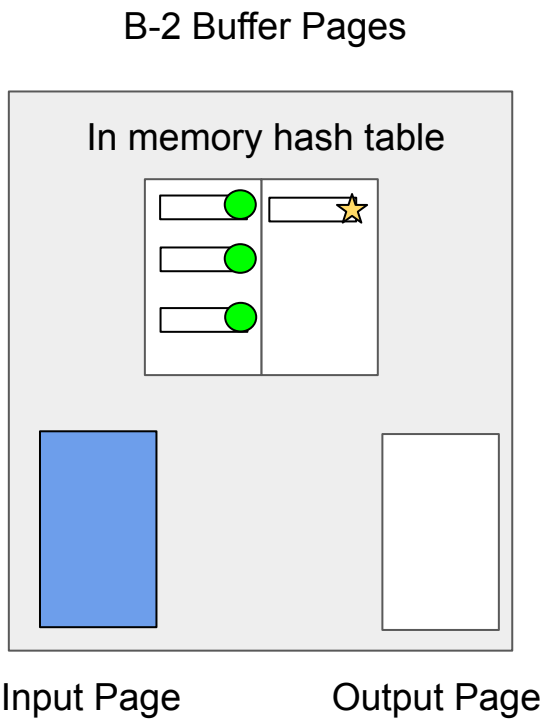




Note: the sections in the hash table represent buckets, rather than pages.

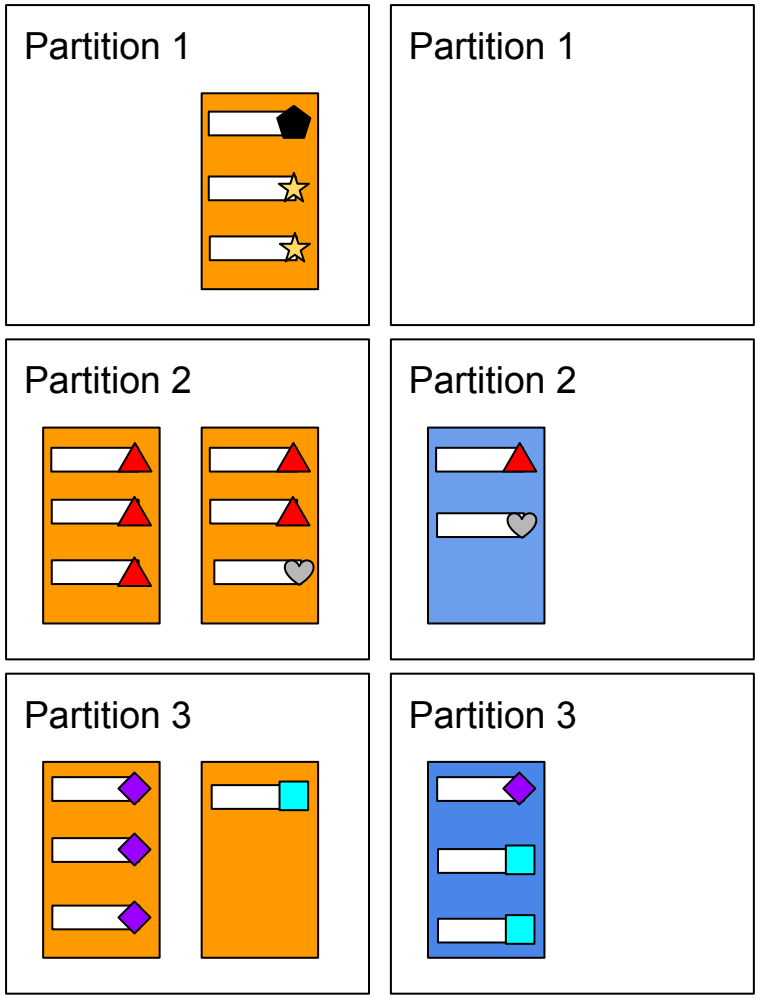
Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

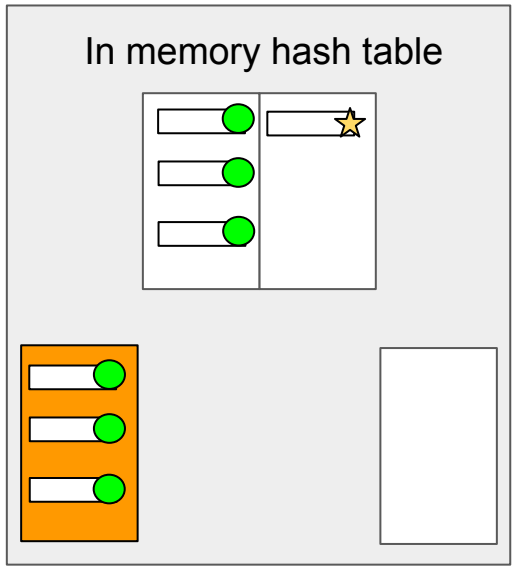




Note: the sections in the hash table represent buckets, rather than pages.

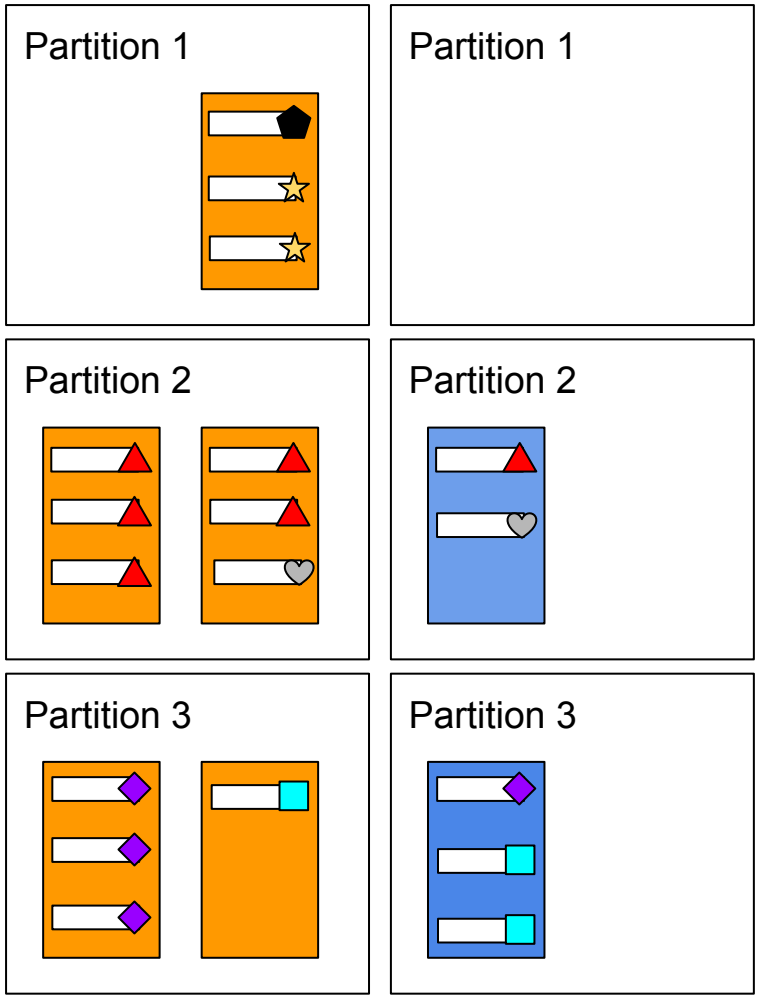
Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

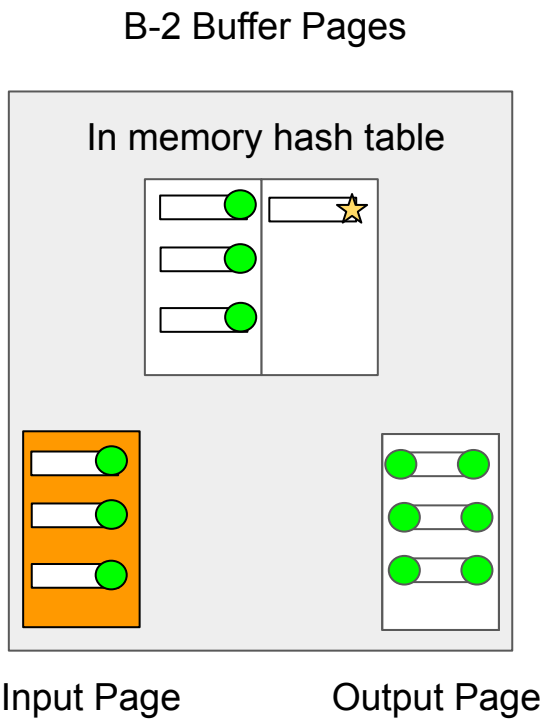




Note: the sections in the hash table represent buckets, rather than pages.

Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

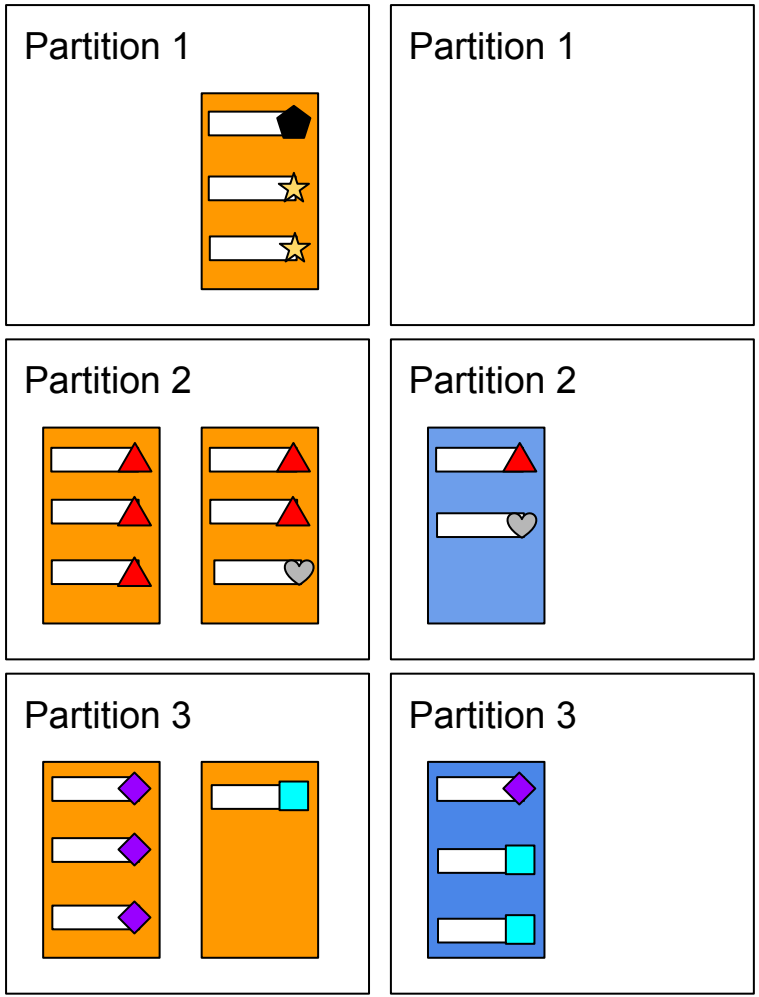


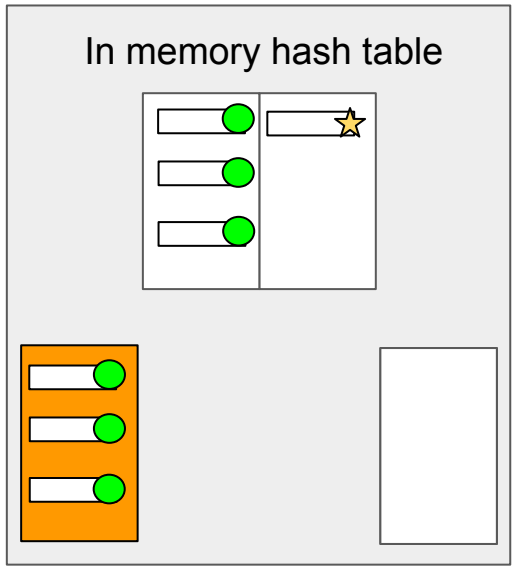


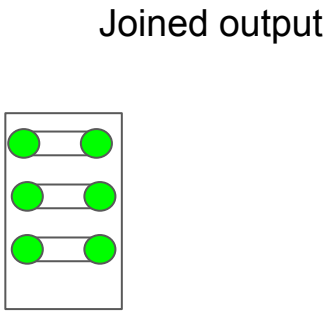
Note: the sections in the hash table represent buckets, rather than pages.

Note: if each page can only hold three records, you could not actually hold 3 joined pairs of records in a single page like this slide is showing. We kept it like this for the sake of convenience.

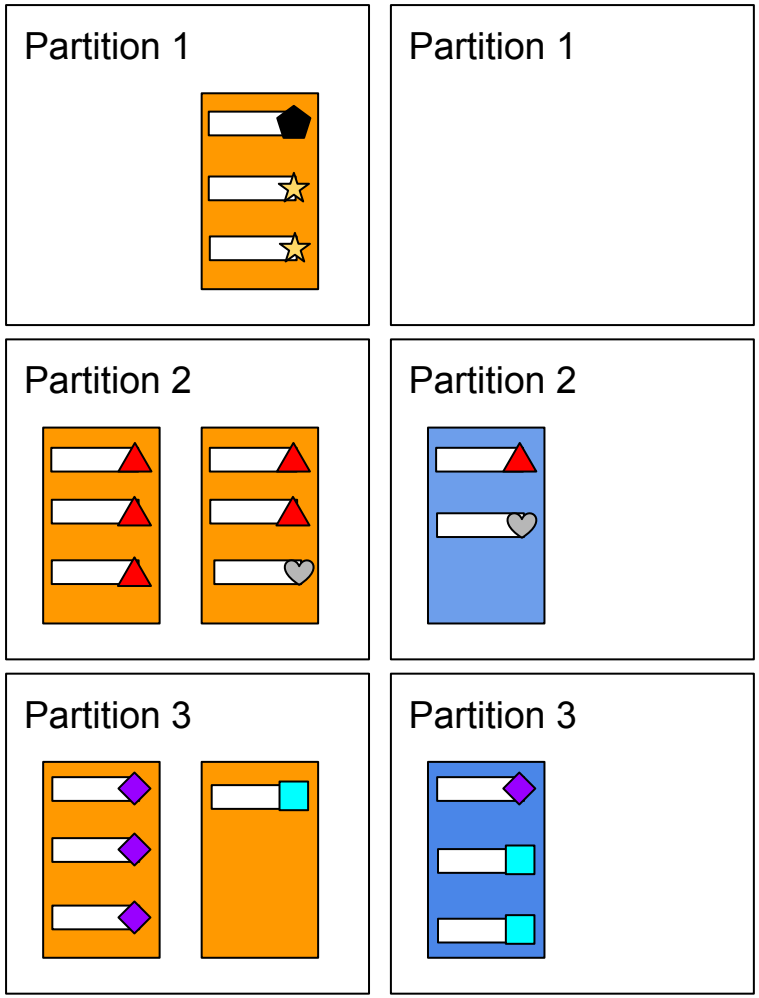
Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

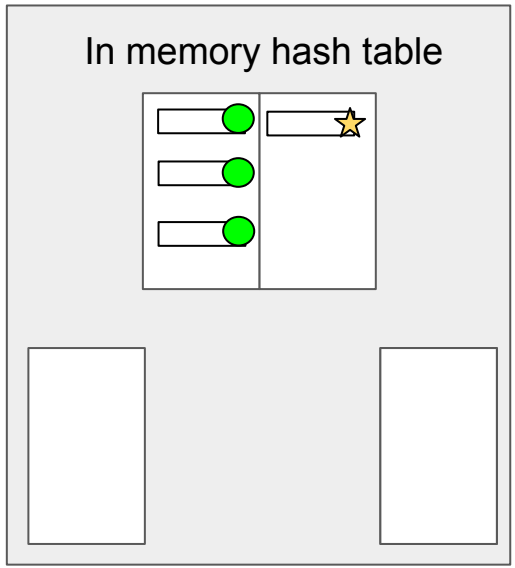


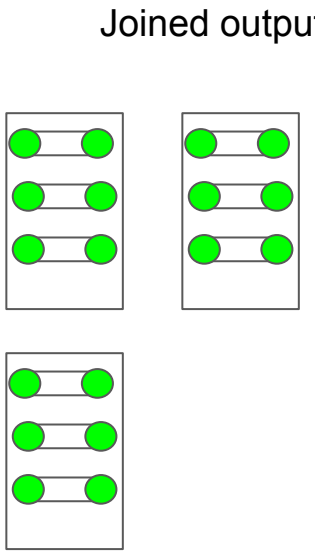




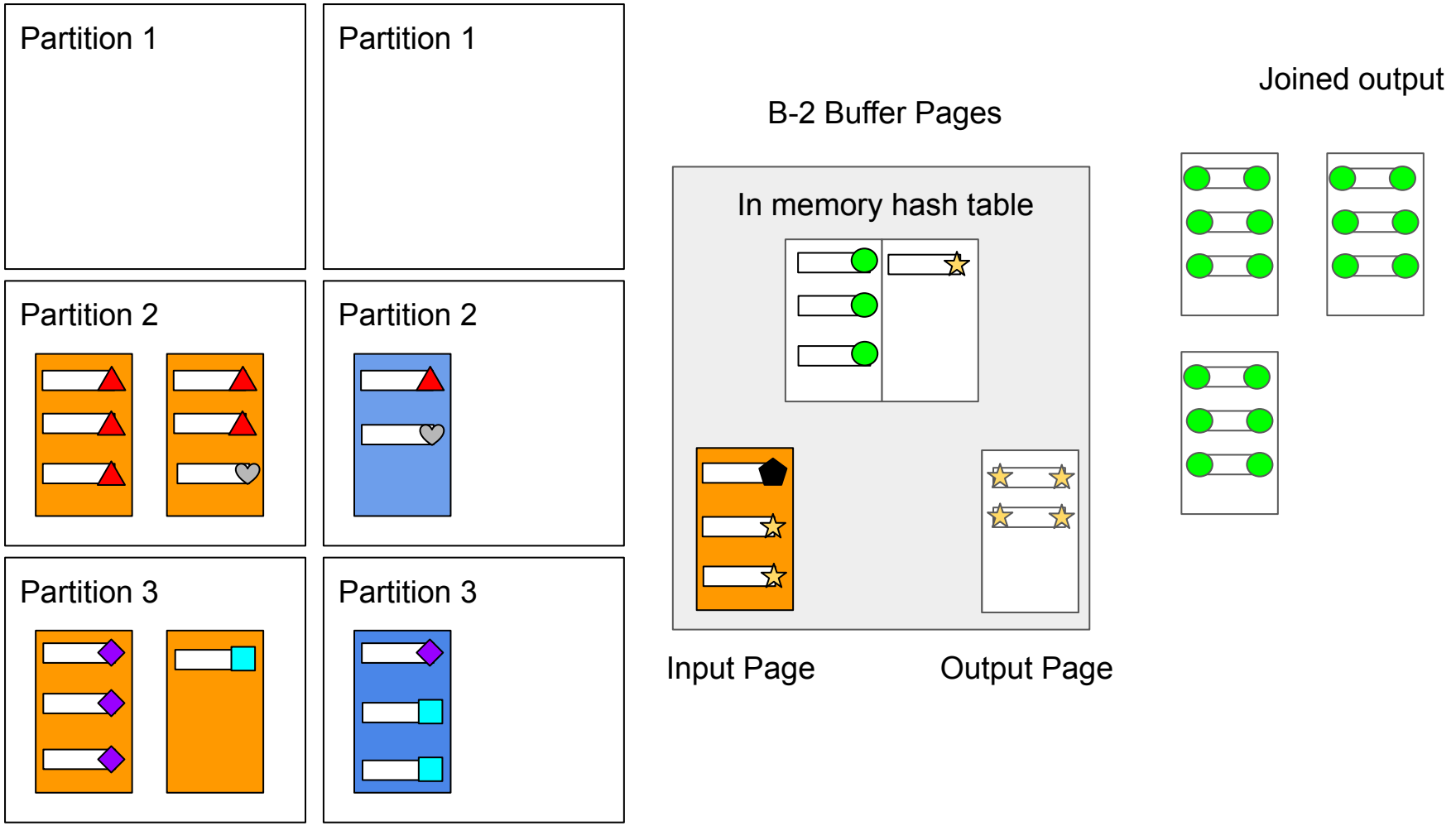
Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley







Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley



Grace hash join example borrowed from Prof. Joseph Hellerstein's CS186 course at UC Berkeley

Practice Problems

- Given Relation X that spans 500 pages, Relation Y that spans 200 pages
  - Both relations have 8 records per page
  - Have access to 12 buffer pages
- Evaluate the number of IO costs for each of the following algorithms:
  - Simple Nested Loop Join
  - Block Nested Loop Join
  - Grace Hash Join
    - Show IOs necessary for both probing and partitioning

Practice Problems

- Given Relation X that spans 500 pages, Relation Y that spans 200 pages
  - Both relations have 8 records per page
  - Have access to 12 buffer pages
- |X|=500 pages, |Y|=200 pages
- ||X|| = |X|*8 records/page = 4000 records
- ||Y|| = |Y|*8 records/page = 1600 records
- B = 12 buffer pages

Practice Problems

- |X|=500 pages, |Y|=200 pages
- ||X|| = |X|*8 records/page = 4000 records
- ||Y|| = |Y|*8 records/page = 1600 records
- B = 12 buffer pages
- Evaluate the number of IO costs for each of the following algorithms:
  - Simple Nested Loop Join
    - |R| + ||R||*|S| = |Y| + ||Y||*|X| = 200 + 1600*500 = 800,200 IOs
  - Block Nested Loop Join
    - |R|+|S|*ceiling(|R|/(B-2)) = |Y|+|X|*ceiling(|Y|/(B-2)) =
      200 + 500*ceiling(200/(12-2)) = 10,200 IOs

Practice Problems

- |X|=500 pages, |Y|=200 pages
- ||X|| = |X|*8 records/page = 4000 records
- ||Y|| = |Y|*8 records/page = 1600 records
- B = 12 buffer pages
- Evaluate the number of IO costs for each of the following algorithms:
  - Grace Hash Join
    - Cost = (2p+1)(M+N) where p is the number of rounds of partitioning
    - Partition size of Y after 1 round of partitioning: 200/ (B-1) = 18.18... > B-2 = 10
    - Partition size of Y after 2 round of partitioning: 200/ $(B-1)^2$ = 1.65... ≤ B-2 = 10
    - Therefore, we need 2 rounds of partitioning.
    - Partition= 2p*(M+N) = 2*2*(500 + 200) = 2800 IOs
    - Probing = (M+N) =500 + 200 = 700 IOs
    - Total = 2800 + 700 = 3500 IOs

Project 4 Intro

Grace Hash Join and You

- Clark Huckelburg has returned from his business trip
  - Personally wants to help optimize Fakebook's join algorithms
- Implement Grace Hash Join
  - Want to optimize to handle join between any relations
  - Have access to a memory and disk simulator
    - Also helper classes for storing partitions(buckets) and records
- Two main methods to implement
  - Partition phase - put data into partitions
    - Each relation has same number of partitions
    - Make sure to simulate memory accesses!
      - Need to write to disk when partition is full
  - Probe phase - compare each partition across relations
    - Look at each partition of the smaller relation and join the other relation when there is a match
      - Each partition of smallest relation is guaranteed to fit into a single page
      - Only one level of recursive hashing (H1 for partition and H2 for probing)

General Tips

- Read the starter code carefully!
- Read the spec carefully!
  - Lots of assumptions that make your life easier
- Start with partitioning then move onto probing
- Remember you have to handle paging and memory accesses
  - All pages live on disk
  - You need to load each page into mem when necessary
    - Load page in by accessing it’s disk page ID
    - Page in mem has different mem page ID
    - Only room for B pages in mem! (Disk unlimited)
  - You need to write each page back to disk when the time comes
    - Use mem page ID
    - Will be given back a disk page ID listing where it lives in disk

Partition Phase

- Create a vector of Buckets!
  - Remember, each bucket gets its own buffer page in memory
  - Need to reserve one page for input from disk
    - B-1 buckets total
- Go one relation at a time
  - Use same hash function for both relation 1 and relation 2
  - Load each relation page by page
    - Need to iterate over records in each page as well
    - Hash each record to correct bucket and add to corresponding page
    - If a page is full you need to write to disk
      - Also need to write to disk all partially full pages once the relation is exhausted!
    - Remember to clear the pages after writing to disk!

Probe Phase

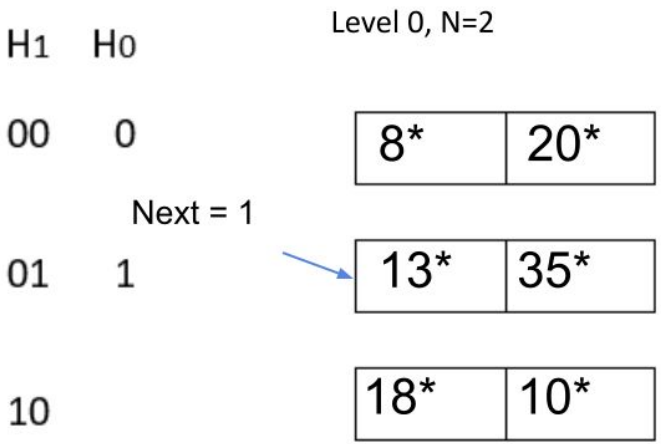
- Use the vector of Buckets!
  - Iterate over the partitions created in the partition phase
  - Repartition each partition into sub partitions
    - Still only have B buffer pages
    - 1 page for input (reading in partition pages from partition page)
    - 1 page for output (output of join)
    - B-2 pages for sub partitions
  - Repartition a partition of one relation, then repartition a partition of the other relation
    - Hint: look at spec to see which one to repartition first to avoid overflows
  - Add any matches to output page
    - Write output page to disk whenever full
    - Accumulate all disk page IDs and return in vector
    - Remember to write output page back to disk at end, even if not full!
  - Remember to clean buffer pages between partitions!

HW4 Q2.2

The hash function that will be used:

$$h_i(value) = value \bmod (2^i N)$$

2.4. Insert 14 = $(1110)_2$ (4 points)



Get started on Homework 5!

We’re here if you need any help!!

- Office Hours: Schedule is here, both virtual and in person offered
- Piazza
- Next week’s discussion!!!