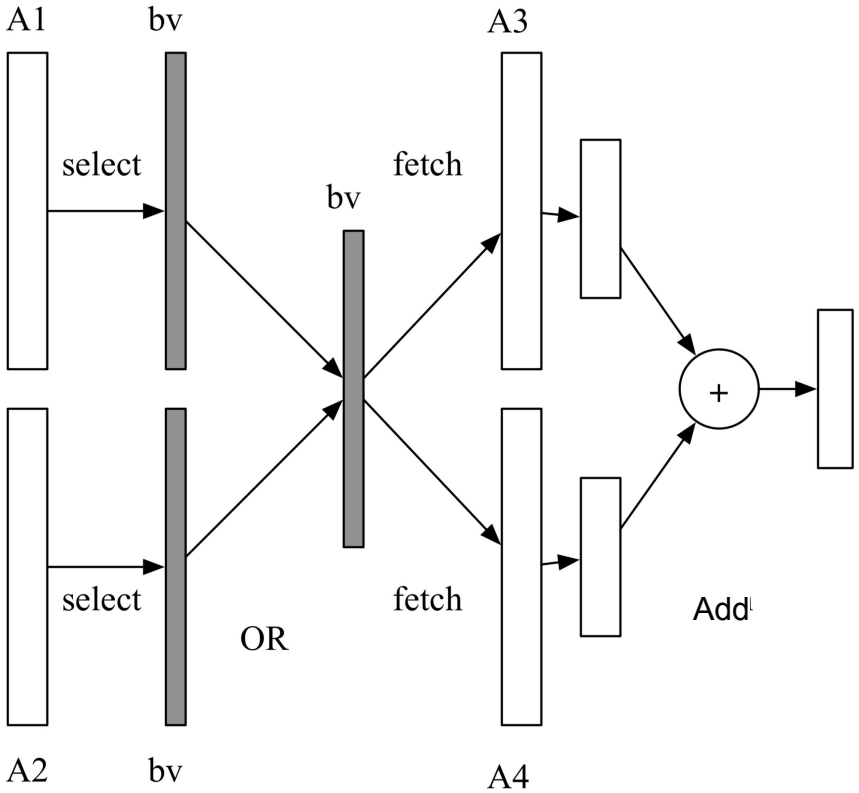
# CS165 Midterm 1 Review

# Part 1

# Query plan





# DSL Operators



- `select(column, left, right)` returns a bit vector such that all positions where `column` values are between `left` and `right` is set to 1.
- `fetch(column, bitVector)` returns a vector of values from `column` that are at positions where `bitVector` is set.
- `or(bitVec1,bitVec2)` returns a bit vector with bitwise or of `bitVec1` and `bitVec2`.
- `add(col1,col2)` returns the vector sum of `col1` and `col2`.

```
def select(c,left,right):

        Initialize bit vector 'res' of size len(c) set to zeros
        i = 0
        For i in range (0,len(c)):
                If c[i] >= left and c[i] < right:
                        res[i] = 1
        return res
```

```
def fetch(bv,c):

        Initialize vector 'res' of size len(c)
        For i in range(0,len(c)):
                if bv[i]:
                        res.append(c[i])
        return res
```

```
def or(bv1,bv2):

        Initialize bit vector 'res' of size len(c) set to zeros
        For i in range(0,len(bv1)):
                res[i] = bv1[i] or bv2[i]

        return res
```

```
def add(c1,c2):

        Initialize res vector of size len(c1)
        For i in range(0,len(c1)):
                res[i] = c1[i]+c2[i]

        return res
```

- Since “A1” has index in the form of a sorted copy, we can use binary search to find the qualifying values in A1.
- Define the select operator with binary search.
- The rest of the plan is the same as before.

# Part 1 - 4

For no index

In the following cost models, let s be the portion of tuples which satisfy A1 < 10 or A2 > 20.

Scan on A1: (A1.width * N + N/8) / M2.block

Scan on A2: (A2.width * N + N/8) / M2.block

Union of A1,A2: (2N/8 + N/8) / M2.block

Fetch on A3 = (N/8 + 2s * A3.width * N) / M2.block

Fetch on A4 = (N/8 + 2s * A4.width * N) / M2.block

Sum = sN * (A3.width + A4.width) + sN * max{A3.width,A4.width} / M2.block

Total cost is sum of all of the above.

In terms of grading, few points were taken off if s was ignored in the cost model as long as it was discussed in the text. Additionally, similar scores were given for cost models which took into account the cost of writing results and which did not.

# Part 1 - 4

Indexed plan (Unclustered Index)

In terms of grading, in 3,4,5,6 you were given full points if you answered these questions correctly for clustered or unclustered indices. The solutions given are all in terms of unclustered indices.

All operators besides the scan on A1 are identical.
Index Search on A1: log2 (A1.width * N / M2.block)
Writing Results: Each result requires a random probe into the bitvector to write a result. Thus, each probe costs 1 M2 miss and so if we let s1 equal the proportion of tuples with A1 < 10, we get:
Scanning index and result writing: (2 * s1 * A1.width * N / M2.block) + (s1 * N)

Total cost =
log2 (A1.width * N / M2.block) + (2 * s1 * A1.width * N / M2.block) + (s1 * N)

# Part 1 - 5

In this question, you were supposed to mention selectivity as a key decision point between using the index and using the full column scan. Mentioning so was enough to get most points.

In addition, we wanted you to use the cost models in 4 to mention parameters which affected this decision. Notably:
Part 4 (Column Scan): (A1.width * N / M2.block) + N/8
Part 4 (Index): log2 (A1.width * N / M2.block) + (2 * A1.width * s1 * N / M2.block) + (s1 * N)

Thus the key factor is notably s1, which makes the index cost quickly increase. As a more detailed analysis, larger sizes for M2.block also favor the column scan, as the column scan is divided by this M2.block size, whereas the index cost is dominated by s1 * N which does not contain this division.

# Part 1 - 6

There were three possible answers to this question (for the unclustered case). Near full credit was given for mentioning any 1 of the 3, with full credit for mentioning more than one (with correct analysis as well). In each method, we are estimating the selectivity.

The first answer was to say that we collect statistics and use something like histograms to estimate the selectivity of the query.

The second answer was to use an index probe to get the starting point and end point for the query (without actually scanning the in between data). Then, use the selectivity gathered from these probes to decide between using the index and scan.

The third answer was to use sampling from the column. After sampling data and conducting the predicate over this (small) sample, then a decision would be made.

# CS165 Midterm 1 Review

## Part 2

# Part 2 - 1

Below are adaptations of pseudo-code for shared select, fetch, and addition. Note that the fetch and addition are expressed using the same algorithm, which is one of many ways of getting points on this question.

**shared select.**

```
def select(c,queries):
        Initialize bit vector 'res' of size len(c) set to zeros for each query q
        For i in range (0,len(c)):
                For q in queries:
                        If c[i] >= q.left and c[i] < q.right:
                                res[q][i] = 1
        Return res
```

**shared fetch & addition.**

```
def fetch_add(c1,c2,bv,queries):
        Initialize vector 'res' of size len(c) for each query
        For i in range(0,len(c)):
                For q in queries:
                        If bv[q][i]:
                                res[q][i].append(c1[i]+c2[i])
        Return res
```

DASlab @ Harvard SEAS

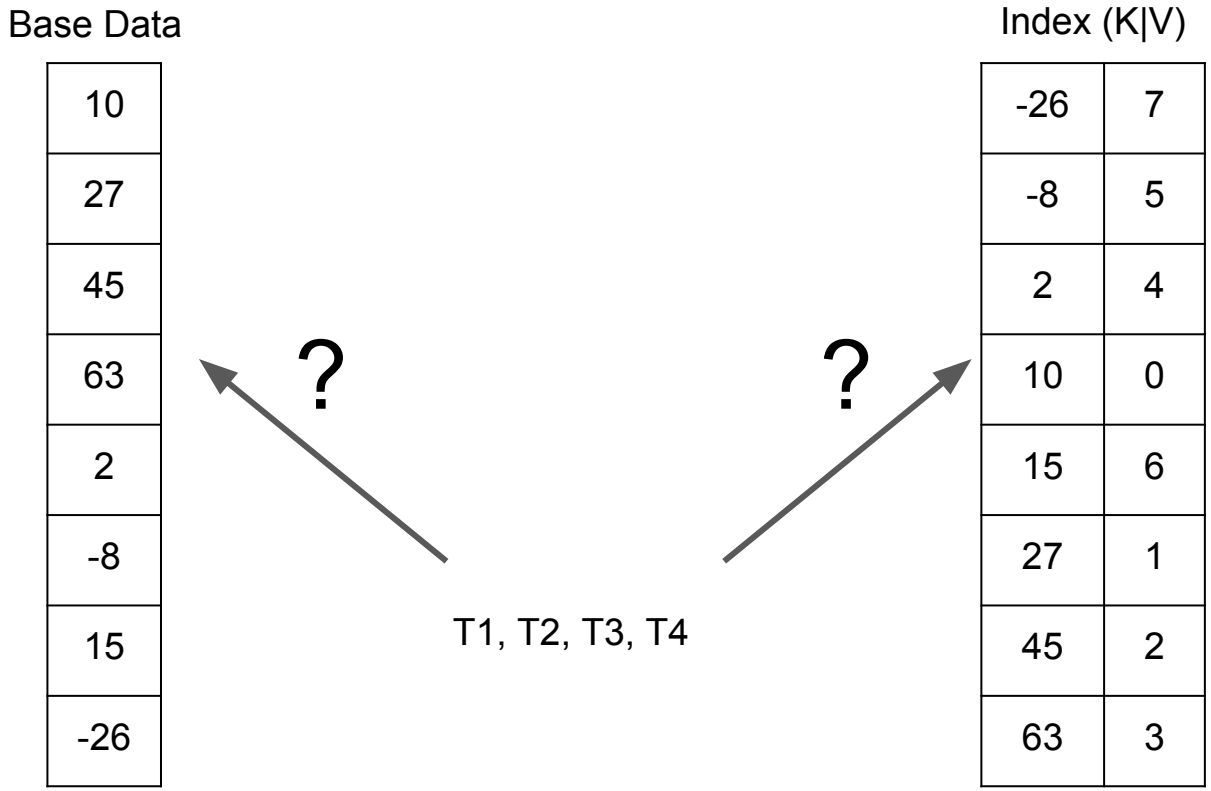
VE RI TAS HARVARD

# Part 2 - 1

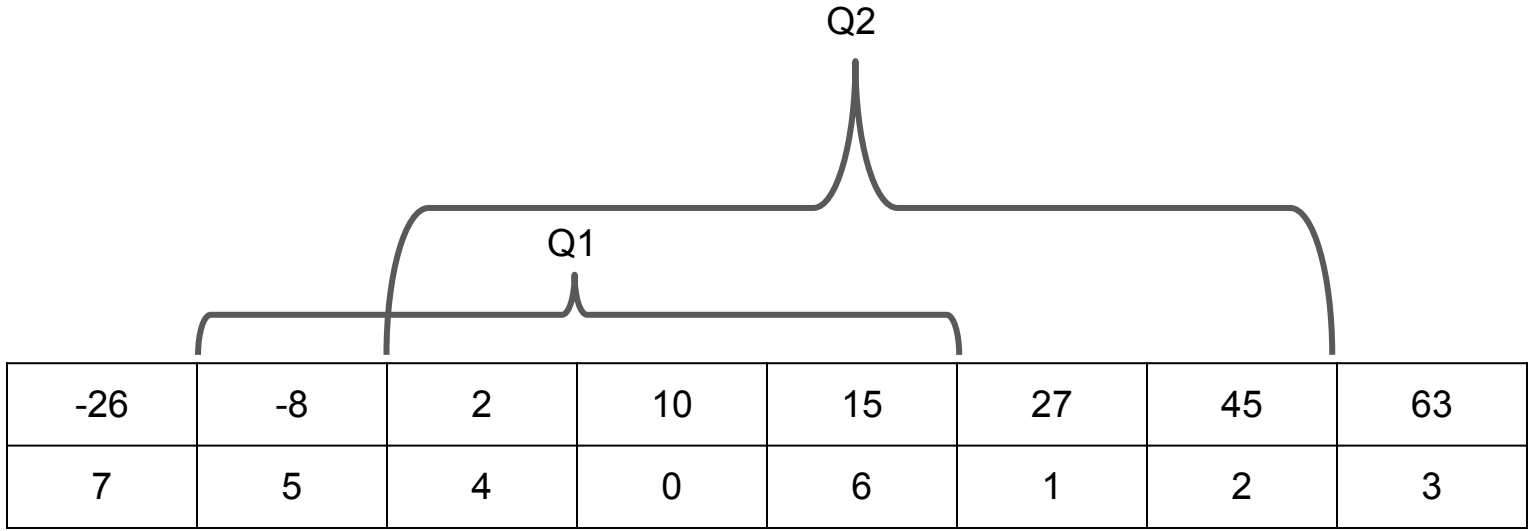
In addition, students got points for pointing out that the number of queries sharing a scan should not exceed the TBL size to avoid thrashing.

Students also got points for discussing how to process shared scan across multiple cores. For instance, if there are multiple shared scans on different columns we can assign each to a different core, or if there is only one shared scan we can partition it across multiple cores and later reassemble the results from across these cores. Students also got points for describing how to synchronize results from partitioned shared scans using concurrency control techniques.

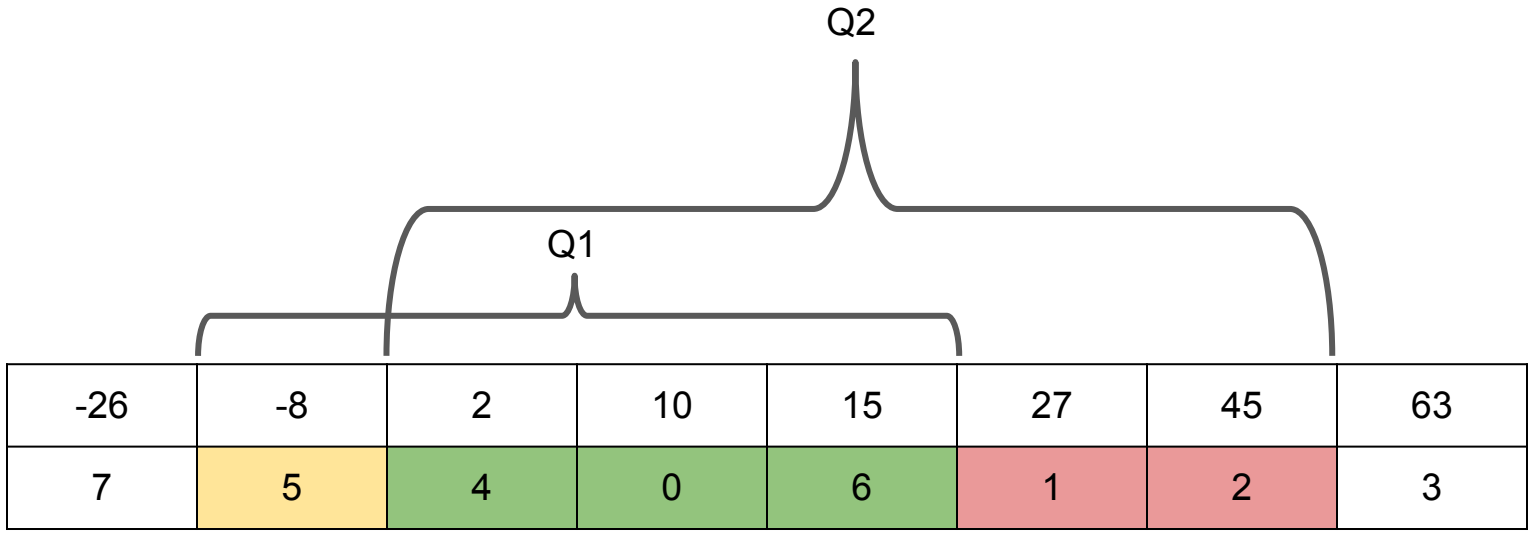
# Part 2 - 2


Base Data
10
27
45
63
2
-8
15
-26
?
?
T1, T2, T3, T4
Index (K|V)
-26 7
-8 5
2 4
10 0
15 6
27 1
45 2
63 3

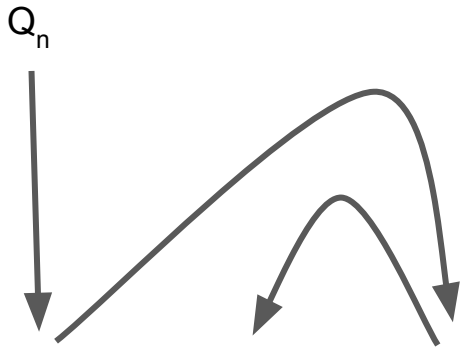
# Part 2 - 2



# Part 2 - 2

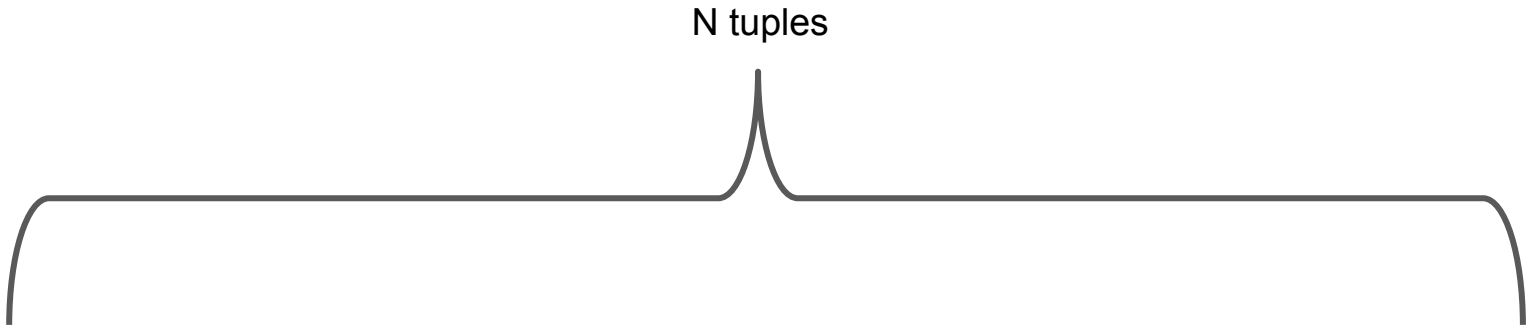


# Part 2 - 3




| -26 | -8 | 2 | 10 | 15 | 27 | 45 | 63 |
|---|---|---|---|---|---|---|---|
| 7 | 5 | 4 | 0 | 6 | 1 | 2 | 3 |

# Part 2 - 3




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -26 | -8 | 2 | 10 | 15 | 27 | 45 | 63 |
| 7 | 5 | 4 | 0 | 6 | 1 | 2 | 3 |

# Part 2 - 3

$$2N * \left( \left\lceil \log_{B-1} \left\lceil \frac{N}{B} \right\rceil \right\rceil + 1 \right)$$

# Part 2 - 3

$$2 * \left(\frac{n * s}{m_2.size}\right) * \left(\left\lceil \log_{\left(\frac{m_3.size}{m_2.size}\right)} \left\lceil \frac{\left(\frac{n*s}{m_2.size}\right)}{\left(\frac{m_3.size}{m_2.size}\right)} \right\rceil \right\rceil + 1\right)$$

# Part 2 - 3

(A1.width * N + Q * (N/8)) / M2.block

# Part 2 - 3

$$Q * (2N/8 + N/8) / M2.block$$

# Part 2 - 3

$$(Q * (N/8) + A3.width * N + A4.width * N) / M2.block$$

# Part 2 - 4

The answer to 1.5 does not fundamentally change: whether scan or index access performs best still depends on selectivity. Shared scans on multiple cores, however, tilt the balance towards scans as more work can be done with a single scan.

# Part 2 - 5

Many students did not address the crux of the problem, which is how to schedule queries with different latency requirements (priorities), but instead gave other design ideas for improving performance in general. Such answers received between 25% and 50% for missing the crux of the question yet demonstrating course knowledge.

There are many possible ways of getting full points on this question. A well developed answer might use both an index and a shared scan: the shared scan for low-priority queries in order to maximize throughput, and the index for high-priority queries with low selectivity. If selectivity is high for a high-priority query, it is less beneficial to use an index due to random access, and so we we can issue a scan for it on a dedicated core/s to minimize latency by avoiding interference with other queries of lesser priority.

There are many variants and technical details for which students received points, including which metadata structures to use to keep track of the queries based on their deadlines, and how to deal with starvation (i.e., a query never getting served due to higher priority queries incessantly arriving).