Stream Queries

```
SELECT A.Month,
       (A.Sales-B.Sales)/B.Sales (x 100%)
FROM
  (SELECT … AS Month, SUM(…) AS Sales
   FROM …) A,
  (SELECT … AS Month, SUM(…) AS Sales
   FROM …) B
WHERE A.Month = B.Month + 1
```

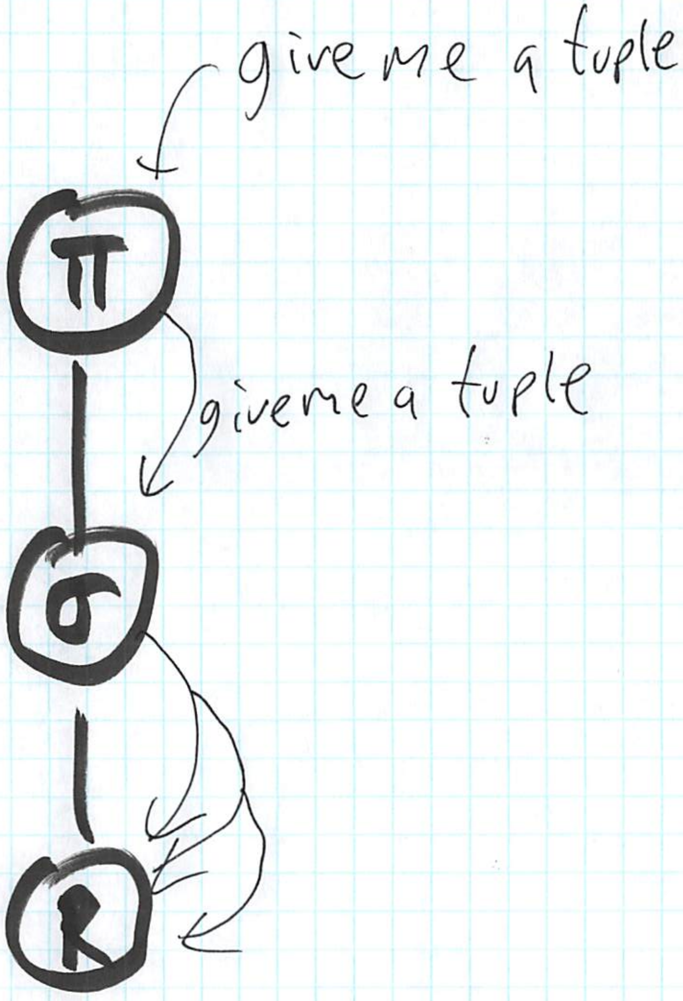
```
SELECT Product, SUM(…) AS Sales
FROM …
WHERE date >= today - ~~1~~3
~~ORDER BY Sales Desc~~
~~LIMIT 5~~

UNION ALL

SELECT Product, SUM(…) AS Sales
FROM …
WHERE date = today - 2
ORDER BY Sales Desc
LIMIT 5

UNION ALL

. . .
```

```
SELECT L.state, T.month,
       AVG(S.sales) OVER W as movavg
FROM   Sales S, Times T, Locations L
WHERE  S.timeid = T.timeid
  AND  S.locid = L.locid
WINDOW W AS (
   PARTITION BY L.state
   ORDER BY T.month
   RANGE BETWEEN INTERVAL ‘1’ MONTH PRECEDING
         AND INTERVAL ‘1’ MONTH FOLLOWING
)
```

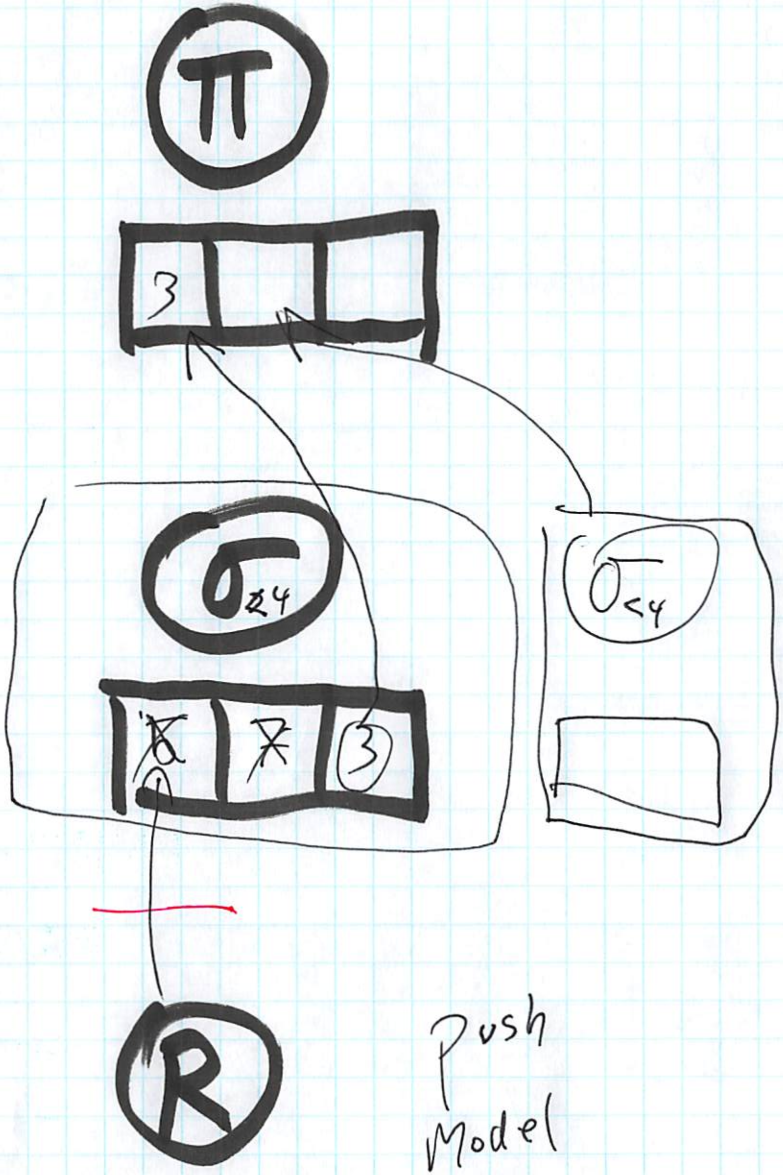
Range between

1 following

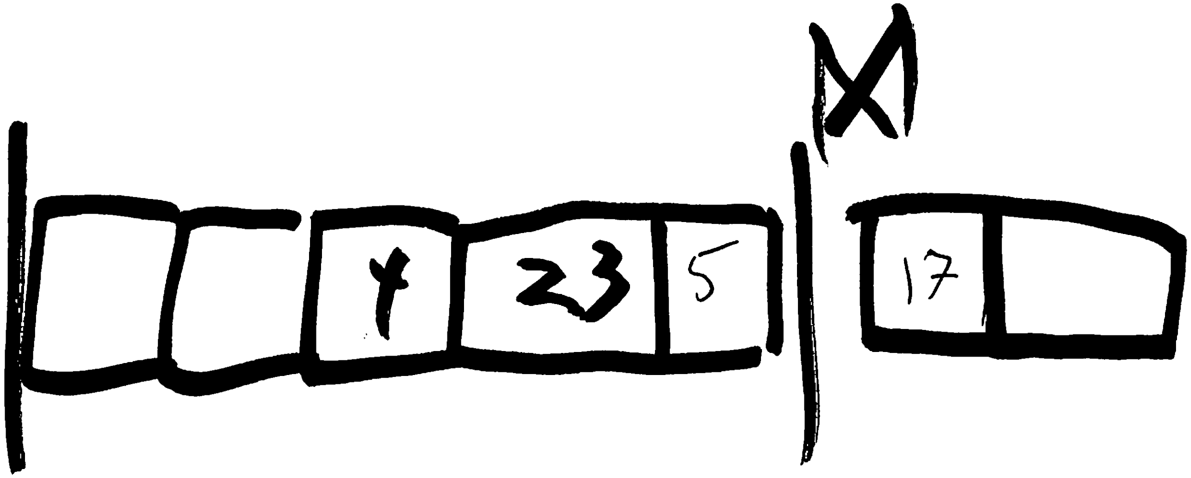
~~0 preceding~~ Optional



Pull Model



Push Model


4
23
5
17

Logical

Based on data

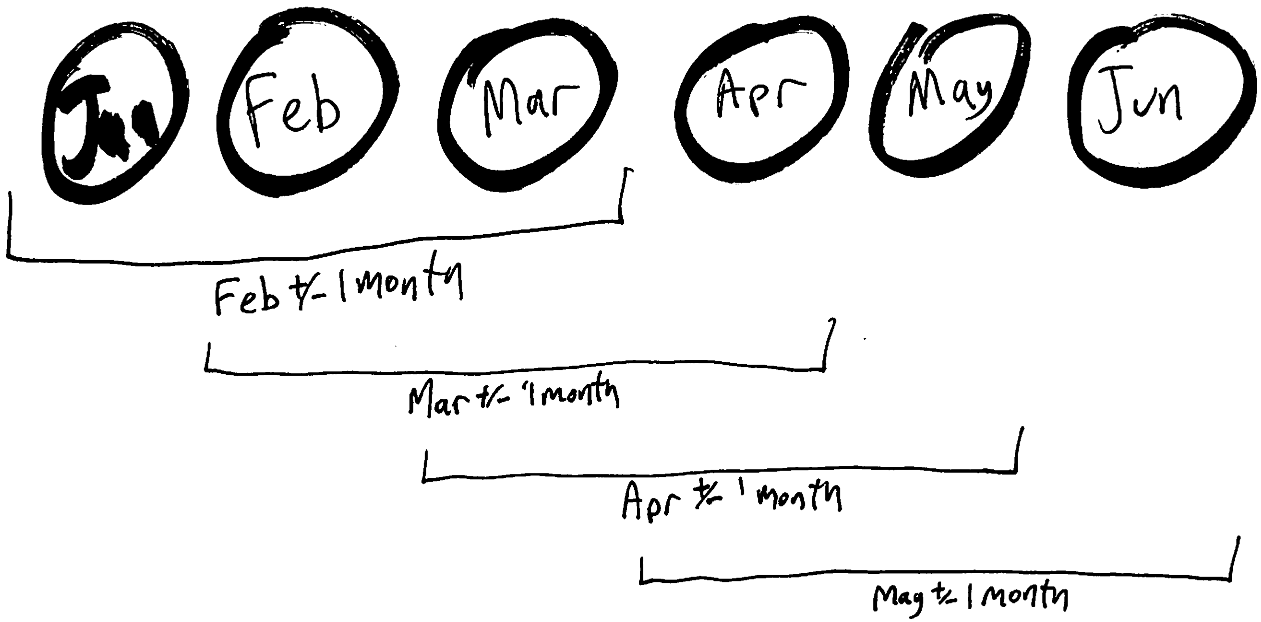
e.g 3 month period

rectangular region of 0.01 lat/long

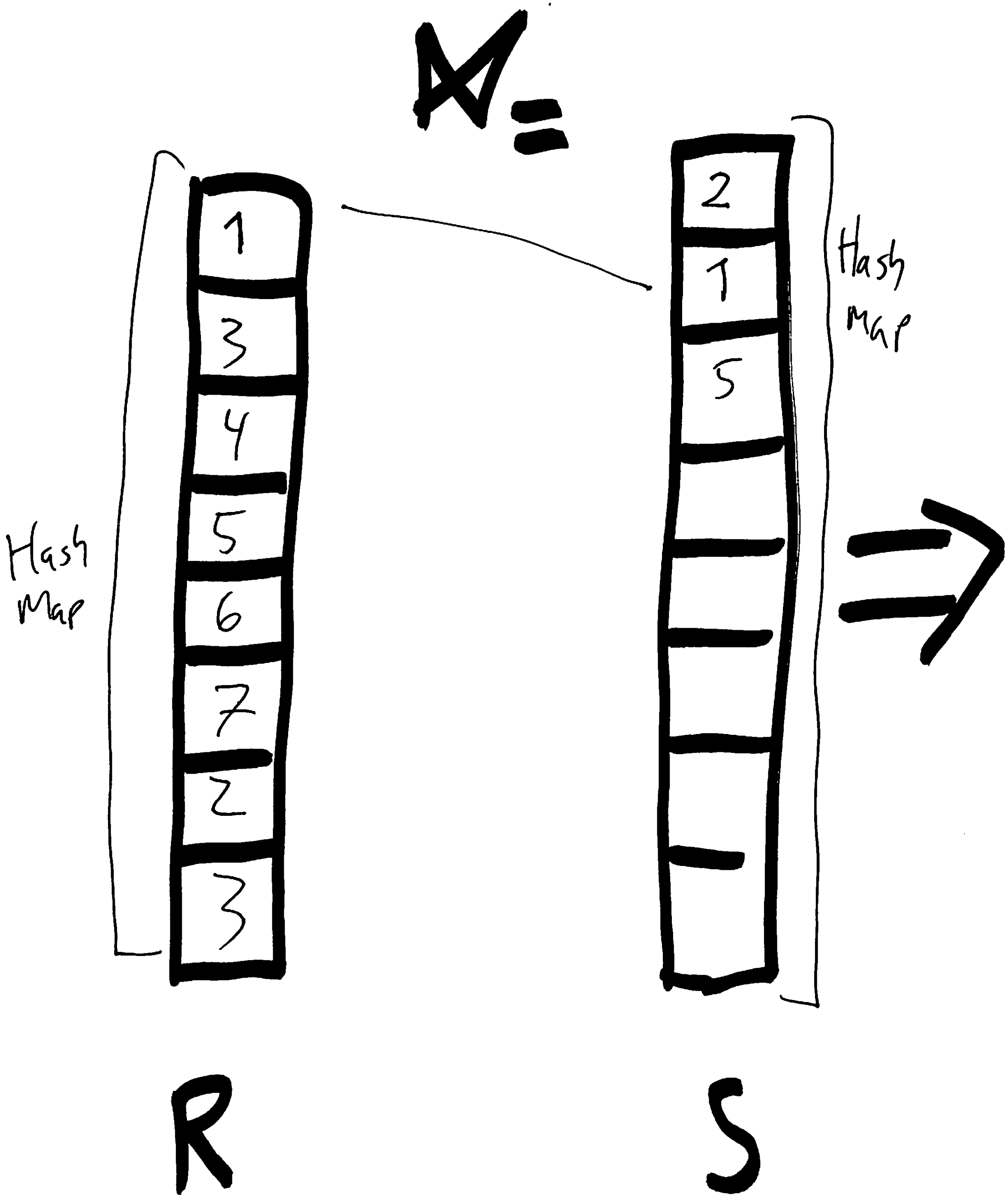
Physical

Based on # of tuples

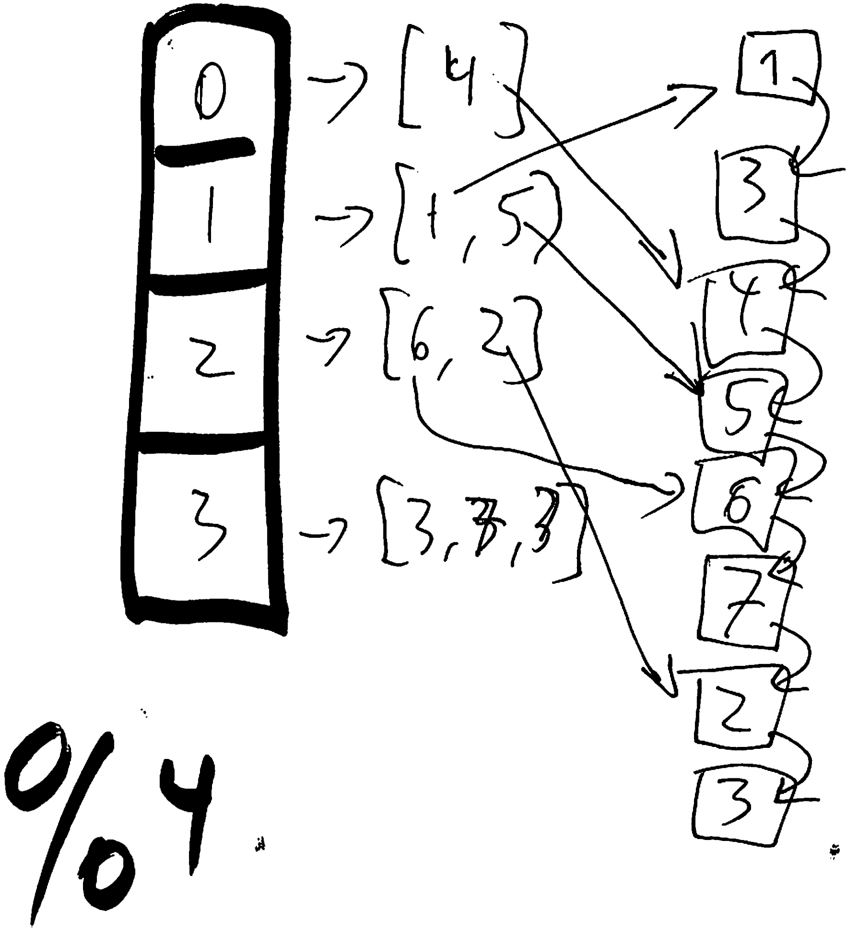
e.g. the last 100 sales


Jan
Feb
Mar
Apr
May
Jun
Feb +/- 1 month
Mar +/- 1 month
Apr +/- 1 month
May +/- 1 month

R(1)
S(2)
S(1)
R(3)
R(4)
R(5)
S(5)
R(6)
R(7)
R(2)
R(3)
R(7)

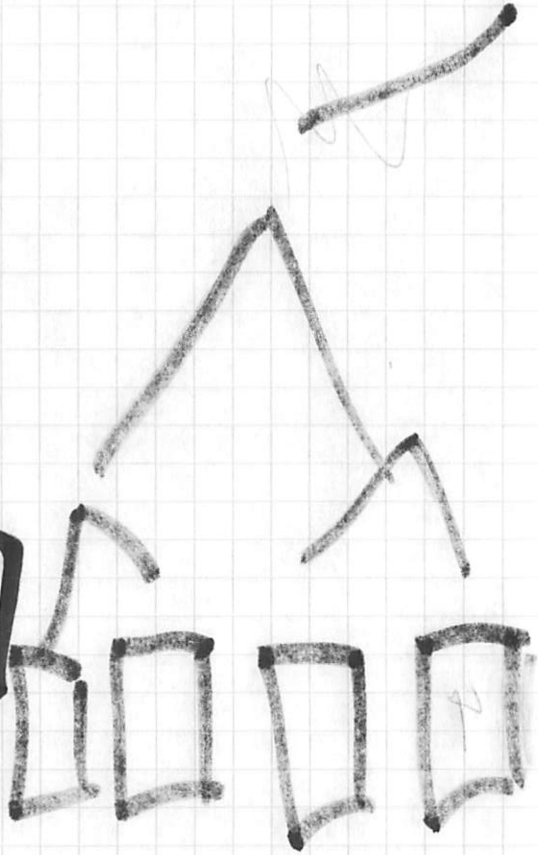


<1, 1>
<5, 5>
<2, 2>


0
1
2
3
[4]
[1, 5]
[6, 2]
[3, 7, 3]
1
3
4
5
6
7
2
3
%4

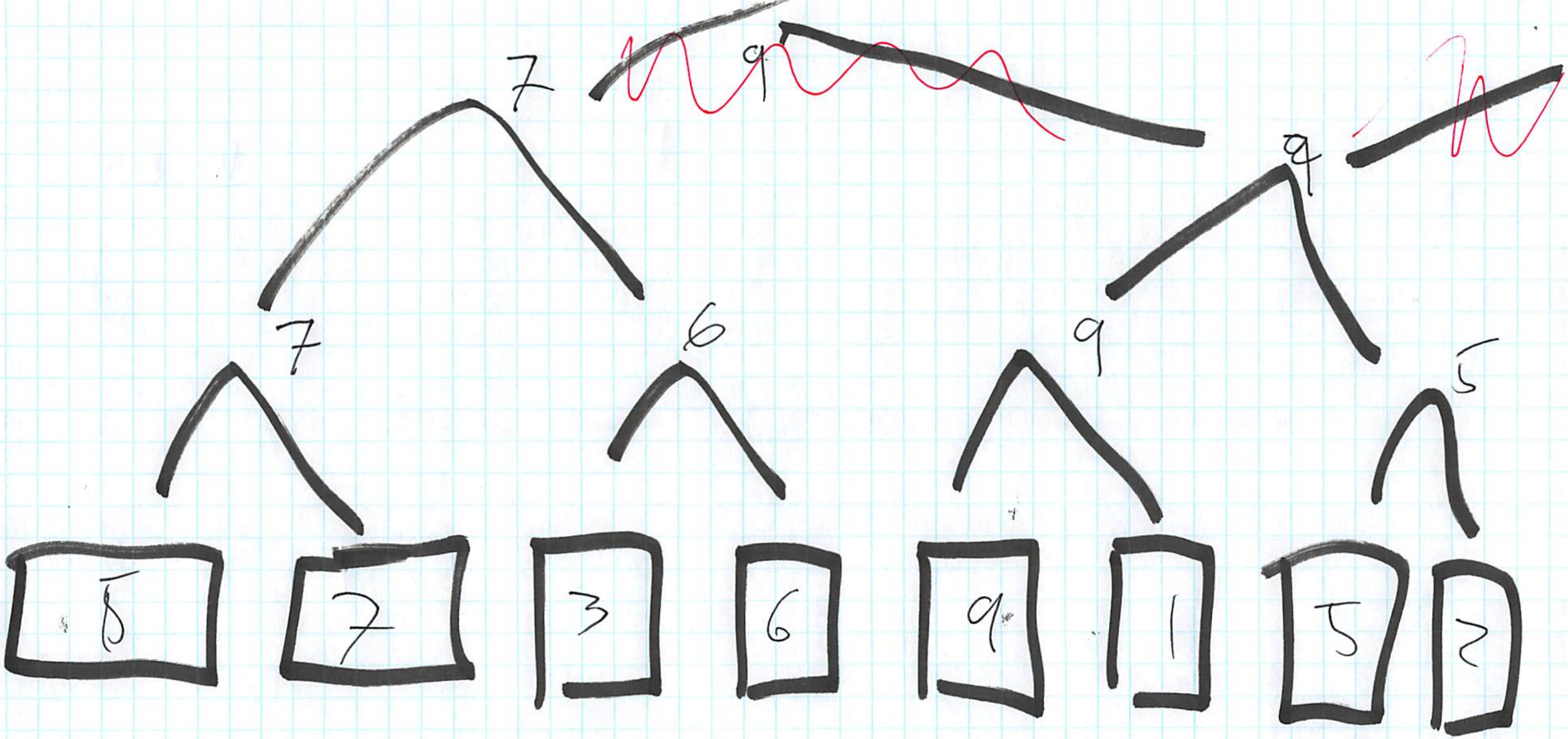
PhysicalWindow : Size of 4

5	7	8	6	9	1	5	2	4



21 25 19

7 9 9 9 9


9
7
9
7
6
9
5
8
7
3
6
9
1
5
2

Sequential Data

Types of data

- **Temporal (focusing on this one today)**
- **Bi-Temporal (Physical Time vs Registered/Recorded Time)**
- **Spatial (2d, 3d)**
- **Spatio-Temporal (3-4d)**

Types of queries

- **Find the % change in monthly sales, each month**
    - SELECT A.Month, A.Sales-B.Sales / B.Sales FROM (SELECT … AS Month, SUM(…) AS Sales FROM …) A, (SELECT … AS Month, SUM(…) AS Sales FROM …) B WHERE A.Month = B.Month + 1
- **Find the daily top-5 products by sales in the last week**
    - SELECT Product, SUM(…) AS Sales FROM … WHERE date = today - 1 ORDER BY Sales Desc LIMIT 5 UNION ALL SELECT Product, SUM(…) AS Sales FROM … WHERE date = today - 2 ORDER BY Sales Desc LIMIT 5, …
- **Find the trailing n-day moving average of sales.**
    - … almost impossible to express if n is a parameter (query size depends on N)

The WINDOW Operator

Semantics:

- **Define a sequence (by sorting the relation)**
- **Generate all subsequences of fixed size**
    - Fixed Physical Size: N records exactly
    - Fixed Logical Size: e.g., Events within N hours of one another
- **Compute an aggregate over each subsequence (like a group-by query)**
- **In-Class Example**
- **Semantics**
    -
      ```
      SELECT L.state, T.month,
             AVG(S.sales) OVER W as movavg
      FROM   Sales S, Times T, Locations L
      WHERE  S.timeid = T.timeid
        AND  S.locid = L.locid
      WINDOW W AS (
         PARTITION BY L.state
         ORDER BY T.month
         RANGE BETWEEN INTERVAL ‘1’ MONTH PRECEDING
               AND INTERVAL ‘1’ MONTH FOLLOWING
      )
      ```
    - Partition By is like Group By
    - Order By Required
    - Range Between Required to define the size of the window (logical vs physical)

        - Aggregates defined OVER W

Stream Queries

Stream vs OLAP vs OLTP

- **OLAP: Fixed Data, Changing Query**
- **OLTP: Changing data, minimal queries**
- **Stream: Fixed Queries, Changing data**
    - Views on steroids
    - View: after a ~10% data update, just rerun the query from scratch

Streams

- **Key Goal: Query Performance >> all**
    - Allowed to discard/defer showing results
    - Allowed to approximate results
    - Allowed to restrict language
        - No nested subqueries
        - All queries must be WINDOW queries (CEP allows hybrid Stream/OLAP queries)
- **Push Model**
    - Each operator is its own processing component with a work queue
    - Operators push records from input to output, requiring per-operator input buffer(s)
    - Operator execution must be scheduled (multi-core execution permitted)
- **“Real-Time” streaming**
    - Operators are given a “fair” amount of scheduled resources to process everything they can
    - Pushes into queues that are full drop the pushed tuples on the floor.
- **Stream Join Data Structures**
    - Stream Join Algo
        - Like view, for R x S:
            - On new record r into R: Join r x S, Index r
            - On new record s into S: Join R x s, Index s
    - Requirements:
        - Push records to the head.
        - Pull records from the tail
        - Be able to look-up records for equi/range joins
    - Implementation
        - Linked Hash-Map, Linked Tree Map
- **Window Aggregate Data Structures**
    - SUM/AVG/COUNT (ring aggregates)
        - Linked List + Aggregate

  - O(1) update cost

- MIN/MAX (semiring aggregates)
  - Linked List + Merkle-ish Trees
  - O(logN) update cost