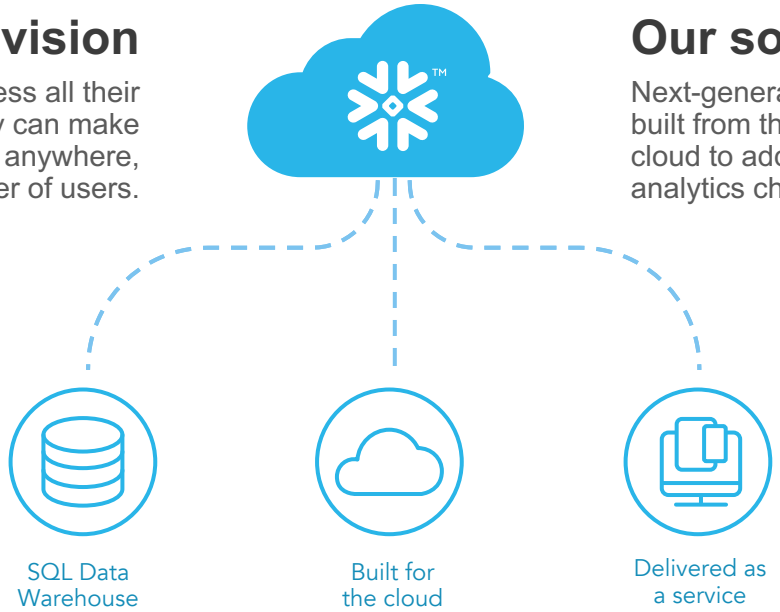
AUTOMATIC CLUSTERING

PRASANNA RAJAPERUMAL I MARCH 2019

SNOWFLAKE

Our vision

Allow our customers to access all their data in one place so they can make actionable decisions anytime, anywhere, with any number of users.



Our solution

Next-generation data warehouse built from the ground up for the cloud to address today's data and analytics challenges.

Automatic

- Memory Management
- Degree of parallelism
- Failure recovery
- Scale Clusters

....

- Clustering

ARCHITECTURE

- Shared disk architecture
- Data stored in S3
- Metadata stored separately
- Compute on-demand

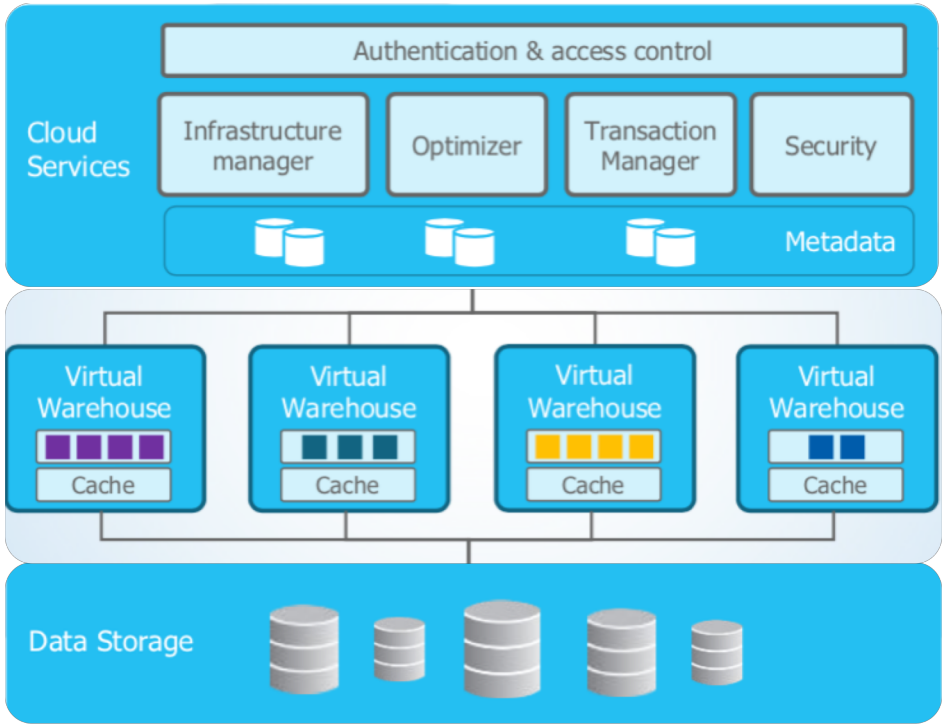


TABLE DATA

- Partitioned horizontally into files (!)
- Columnar (Hybrid) storage (PAX)
  - Column values grouped
  - Compressed
  - Header contains index of column start

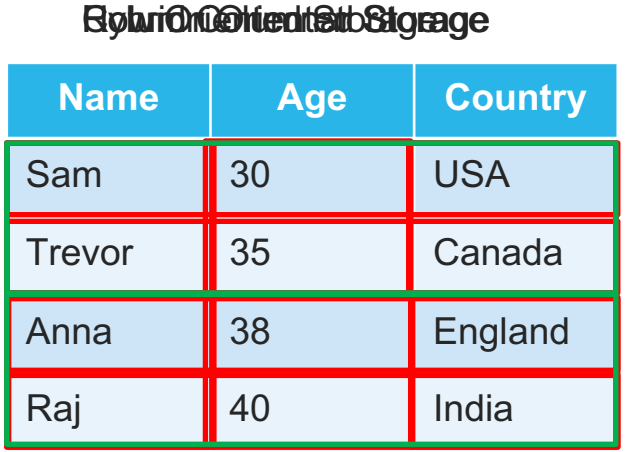


TABLE DATA

- Immutable
- File is a unit of
  - Update
    - Every change is
      files deleted and files added
  - Concurrency
- Sized to be few 10's of MB at the most
- 10's of millions of files

File 1:

Sam	30	USA
Trevor	35	Canada

Update table T set age = 45 where name = "Trevor"

File 2:

Sam	30	USA
Trevor	45	Canada

METADATA

- Stats on every column for every file
  - Zone maps (Netezza)
  - Min/Max, #distinct, #nulls etc
- Improve Query performance
  - Pruning: Reduce scan

File Name	Col	Min	Max
File1	Id date	3 4/1	9 4/3
File2	Id date	1 4/3	10 4/4
File3	Id date	7 4/5	10 4/5
File4	Id date	1 4/5	10 4/6

File1
Id: 3, 5, 9
date: 4/1, 4/2, 4/3

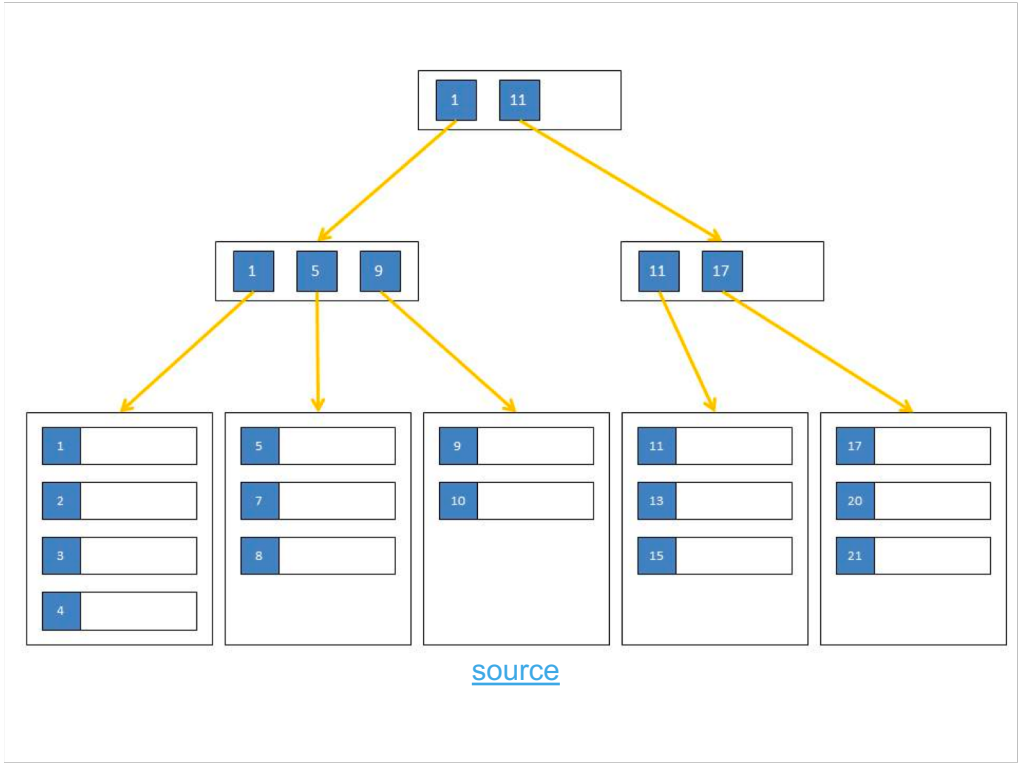
File2
Id: 6, 1, 10
date: 4/4, 4/3, 4/4

File3
Id: 7, 9, 10
date: 4/5, 4/5, 4/5

File4
Id: 1, 9, 10
date: 4/5, 4/6, 4/6

INDEX COMPARISON

- Why not B-Trees?
  - Uni-dimensional
  - Index Maintenance
  - IO Cost on large queries

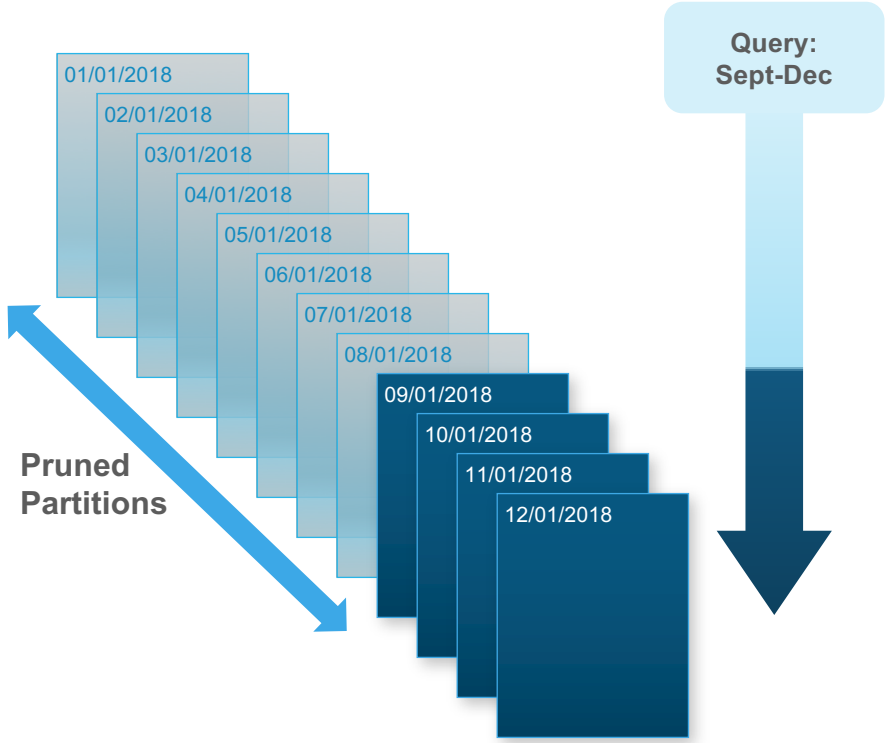


DEFAULT CLUSTERING

- Partitioned on ingestion
  - Based on physical property: size
  - Clustered based on load time
  - Not optimized for other dimensions

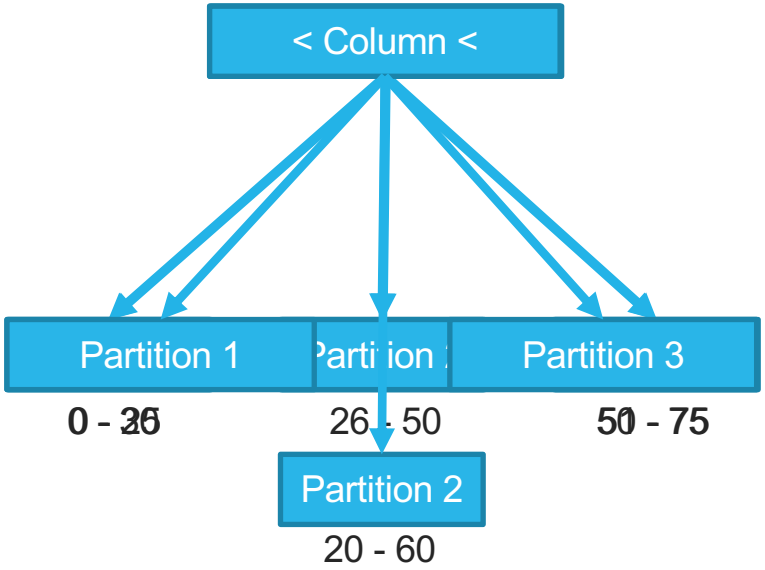
Select … from orders where
date between
’09-01-2018’ and ’12-01-2018’

Select … from orders where
product = ‘IPhone’



PARTITIONING COMPARISON

- Hash based
  - Not efficient for range scans
- Range based
  - Strict boundaries
  - Range Splitting when it gets too big
- Snowflake
  - Key not used for data distribution
  - Overlaps in ranges are okay
  - Achieve good pruning with zone maps (min/max)

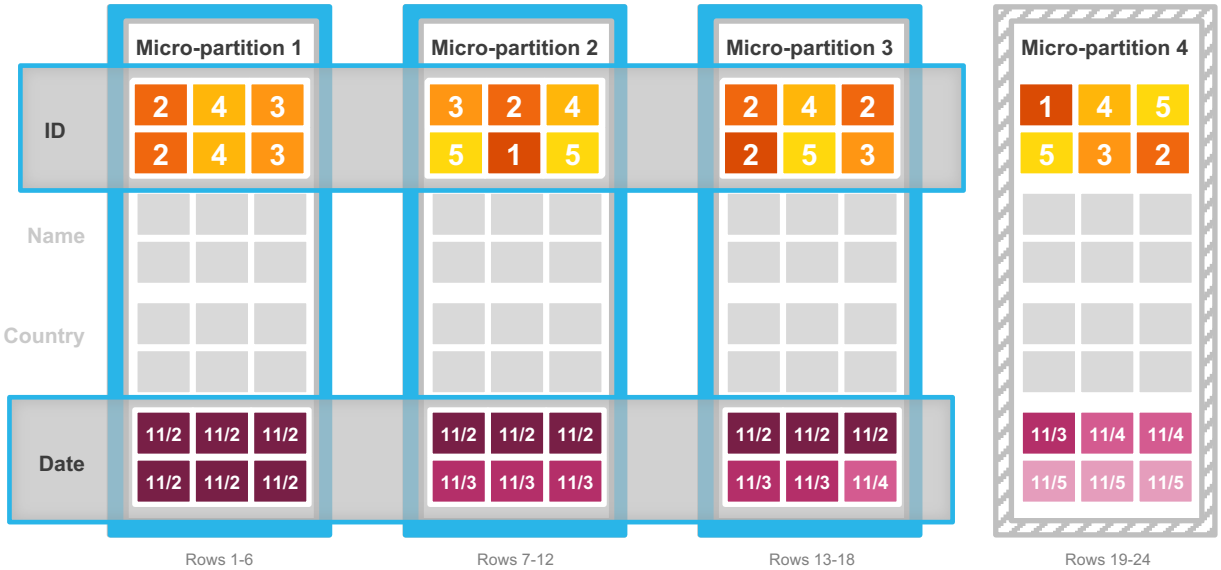


DEFAULT CLUSTERED TABLE

ID	Name	Country	Date
2	A	UK	11/2
4	C	SP	11/2
3	C	DE	11/2
2	B	DE	11/2
3	A	FR	11/2
2	C	SP	11/2
3	Z	DE	11/2
2	B	UK	11/2
4	C	NL	11/2
5	X	FR	11/3
1	A	NL	11/3
5	A	FR	11/3
2	X	FR	11/2
4	Z	NL	11/2
2	Y	SP	11/2
2	B	SP	11/3
5	X	DE	11/3
3	A	UK	11/4
1	C	FR	11/3
4	Z	NL	11/4
5	Y	SP	11/4
5	B	SP	11/5
3	X	DE	11/5
2	Z	UK	11/5

Query

```
Select count(*) from T where id = 2 and date = '11/2';
```



- Natural ordering by date
- **Scans 3 partitions**

EXPLICIT CLUSTERING

Clustering Keys

- Identify interesting expressions
- Should be order preserving

AUTOMATIC CLUSTERING

Automatically reorganizes table storage from natural order to Clustering keys order

PROBLEM DEFINTION

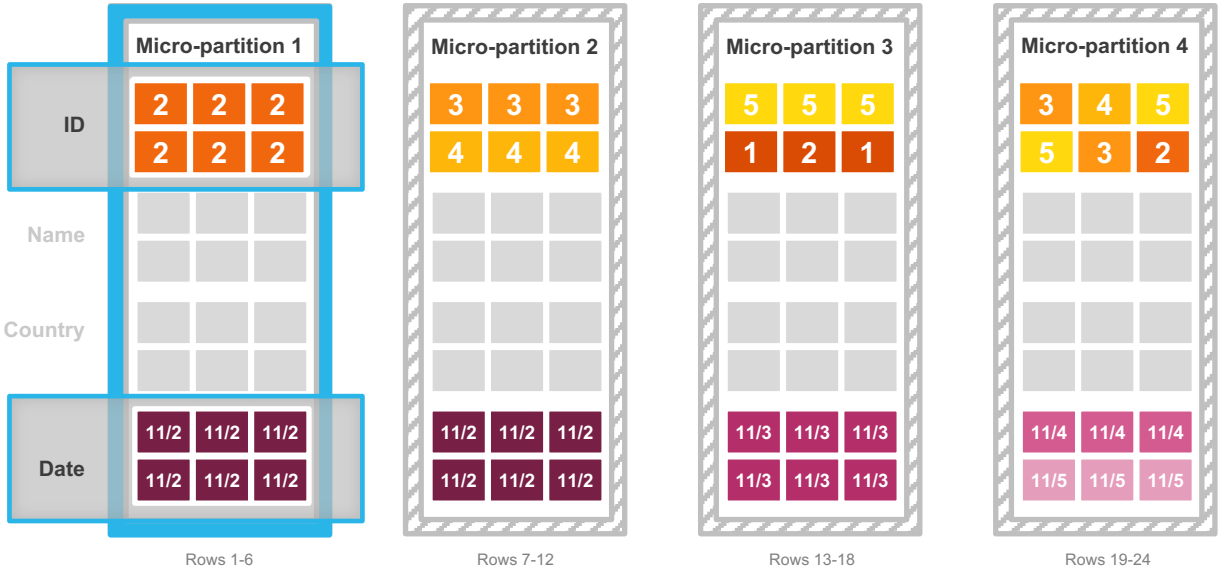
CONTINIOUS SORTING AT PETABYTE SCALE

EXPLICITLY CLUSTERED TABLE

ID	Name	Country	Date
2	A	UK	11/2
4	C	SP	11/2
3	C	DE	11/2
2	B	DE	11/2
3	A	FR	11/2
2	C	SP	11/2
3	Z	DE	11/2
2	B	UK	11/2
4	C	NL	11/2
5	X	FR	11/3
1	A	NL	11/3
5	A	FR	11/3
2	X	FR	11/2
4	Z	NL	11/2
2	Y	SP	11/2
2	B	SP	11/3
5	X	DE	11/3
3	A	UK	11/4
1	C	FR	11/3
4	Z	NL	11/4
5	Y	SP	11/4
5	B	SP	11/5
3	X	DE	11/5
2	Z	UK	11/5

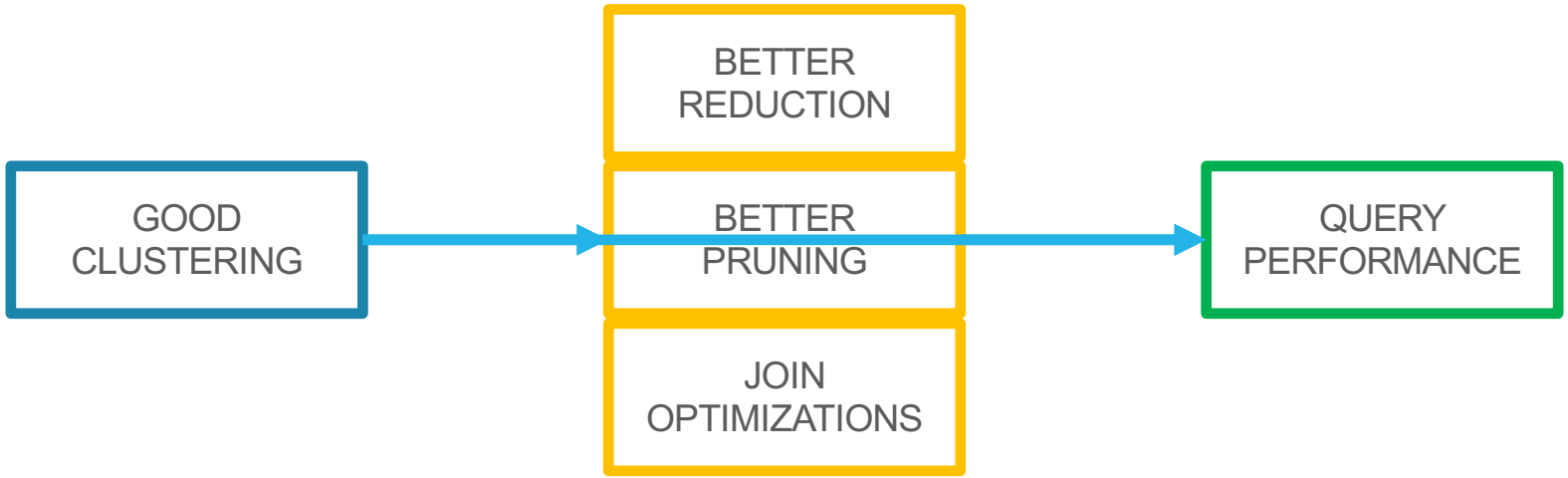
Query

```
Select count(*) from T where id = 2 and date = '11/2';
```

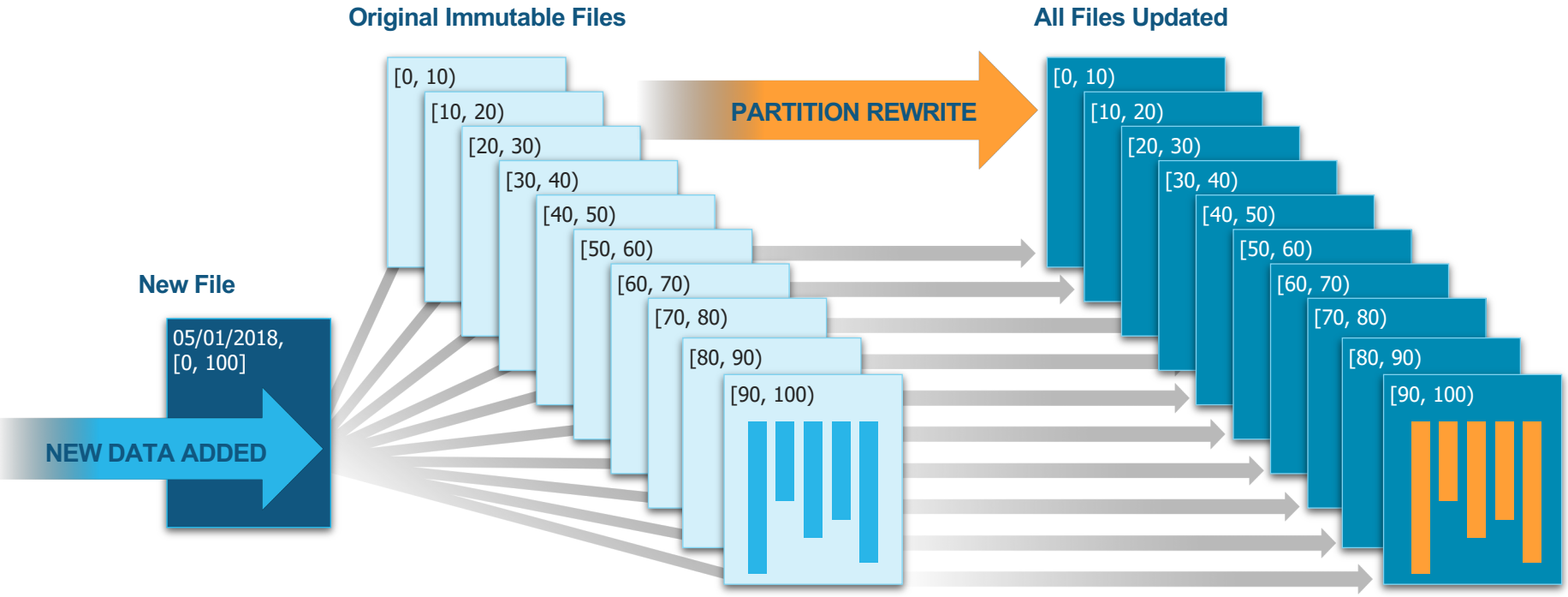


- Clustering keys (date, id)
- Scans only 1 partition

WHY EXPLICIT CLUSTERING?



CHALLENGES: KEEPING UP WITH CHANGES



APPROACHES TO KEEP UP

Re-cluster inline with the changes

- Full table re-write
- High write amplification
- Bound by re-cluster speed
- Not practical for TB/PB tables

Batch re-cluster

- Extremely expensive
- Block other changes
- Huge variation in Query performance
- Not practical on PB tables

REQUIREMENTS

- Actual sort order is not a requirement
  - Goal – Good pruning with min/max
  - Overlap of value ranges are acceptable
- Background Service
- Find incremental work to improve query performance
- Should not interfere with other changes
- Can change clustering keys

PROBLEM DEFINTION

Approximate

CONTINIOUS SORTING AT PETABYTE SCALE

CLUSTERING METRICS - WIDTH

- **Width of a partition**

  Width of the line connecting the min and max value in the clustering values domain range

File1 : Wide File

Sam	18	USA
Trevor	78	Canada

File2 : Narrow File

Anna	30	England
Raj	35	India

CLUSTERING METRICS - DEPTH

- **Depth at a single value (or range)**

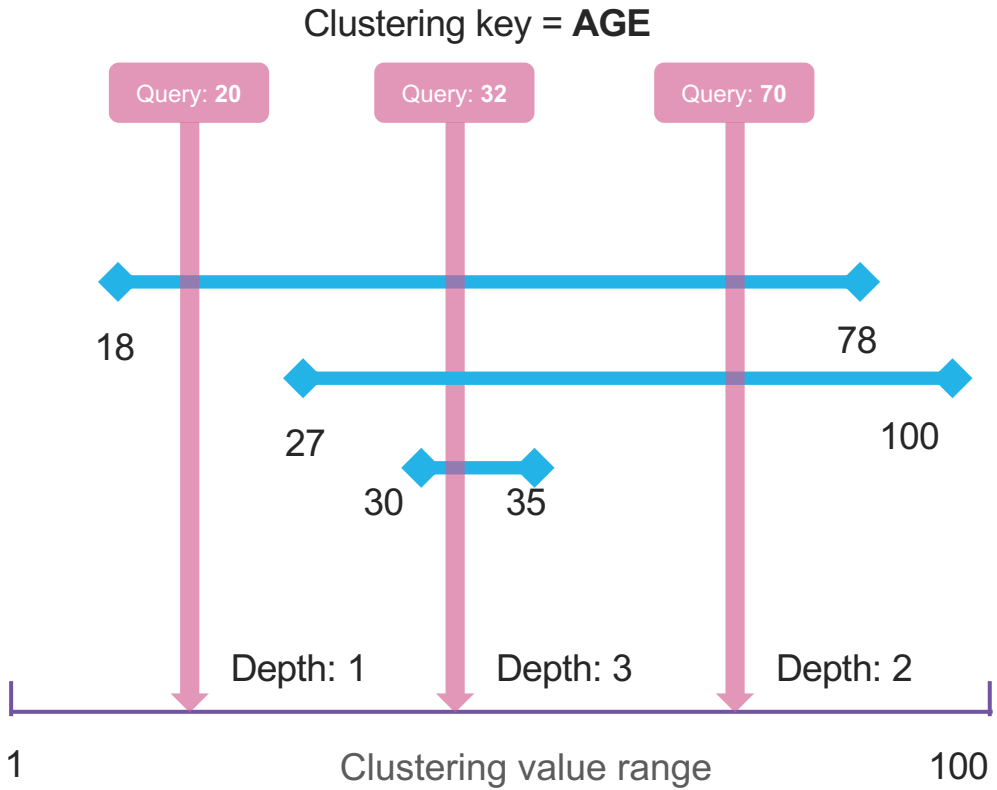
  Number of partitions overlapping at a certain value in the clustering key domain value range

File1

Sam	18	USA
Trevor	78	Canada

File2

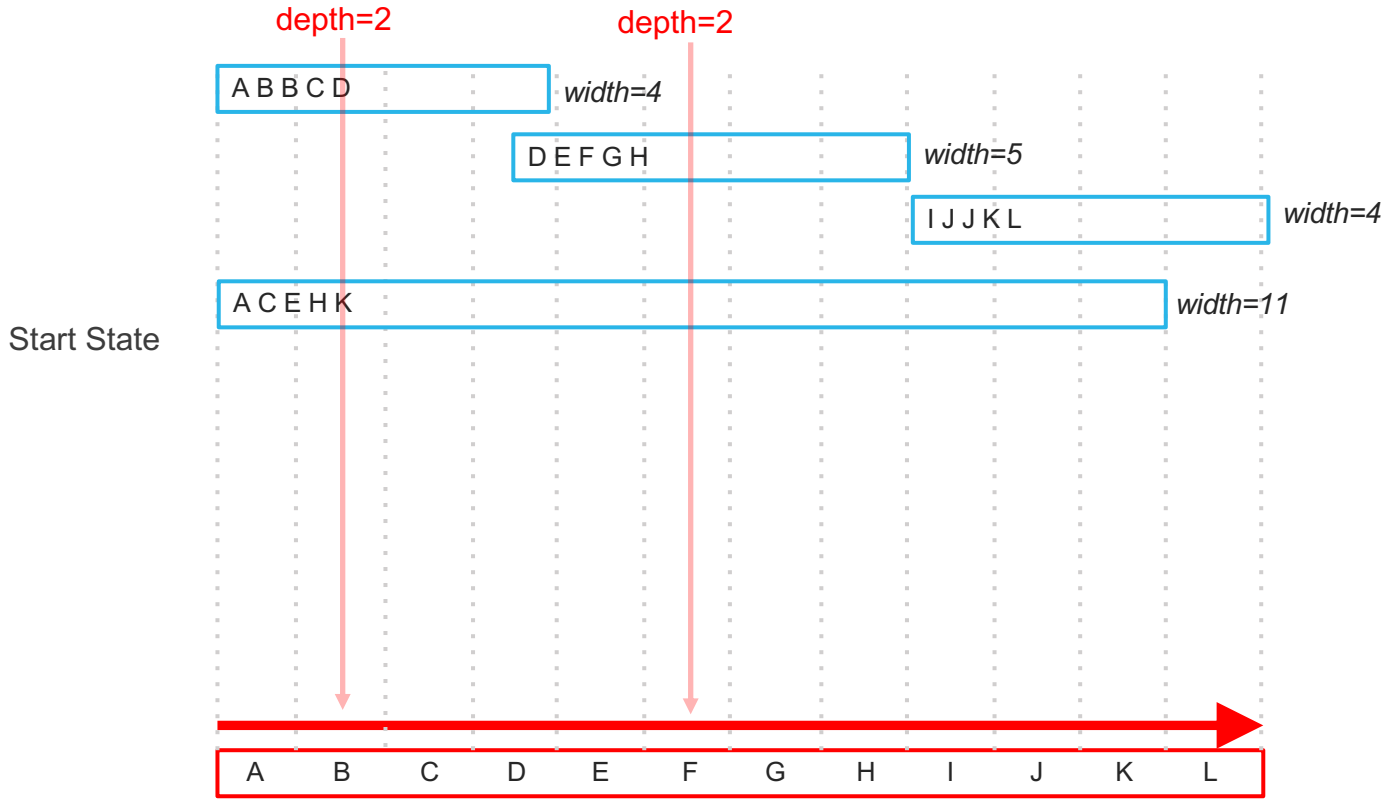
Anna	30	England
Raj	35	India



GOAL

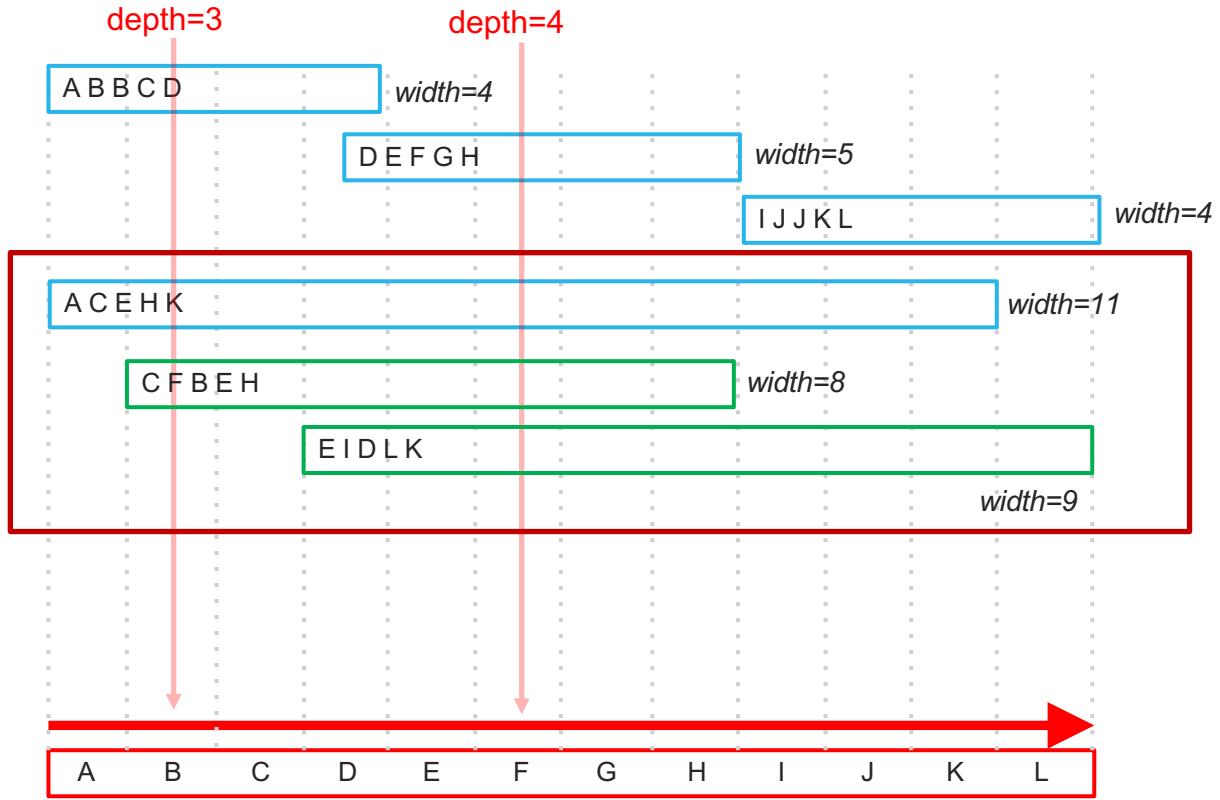
Reduce **Worst** Clustering Depth
below an
acceptable threshold
to get
Predictable Query Performance

CLUSTERING SCENARIO



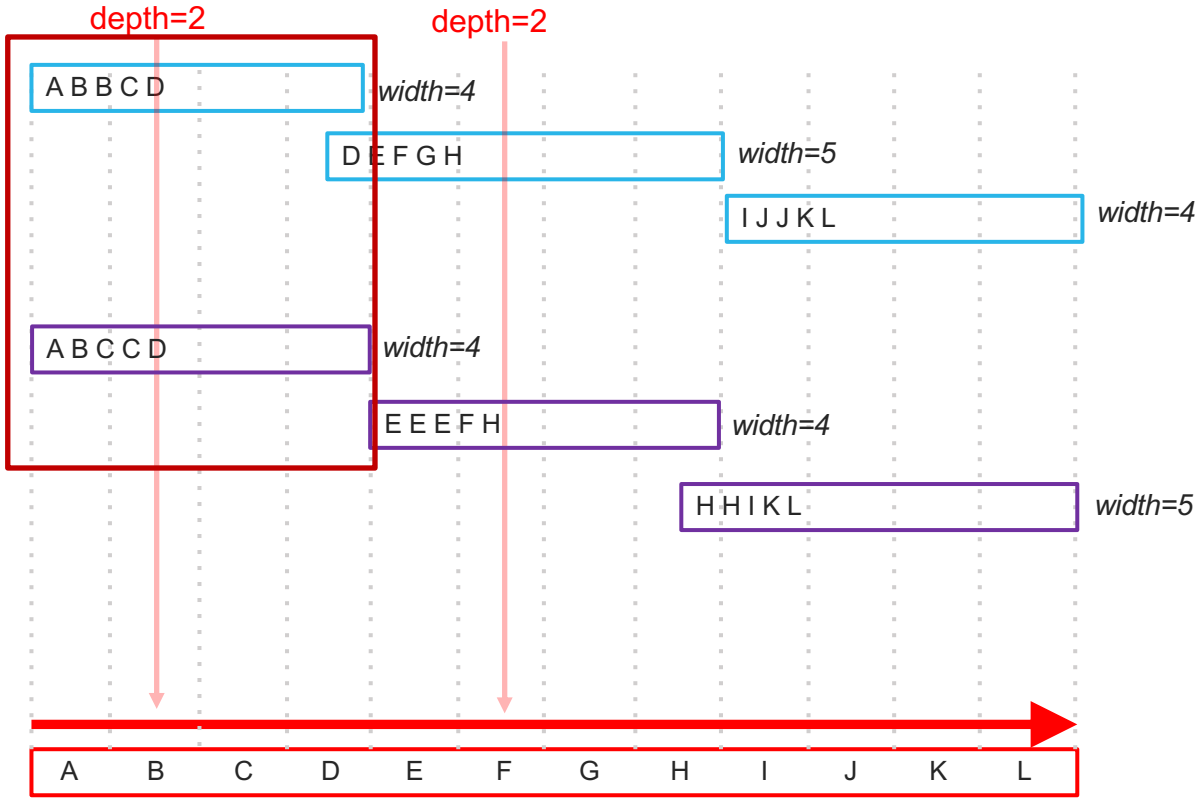
CLUSTERING SCENARIO

Iteration 1: Pick 3 files to recluster



CLUSTERING SCENARIO

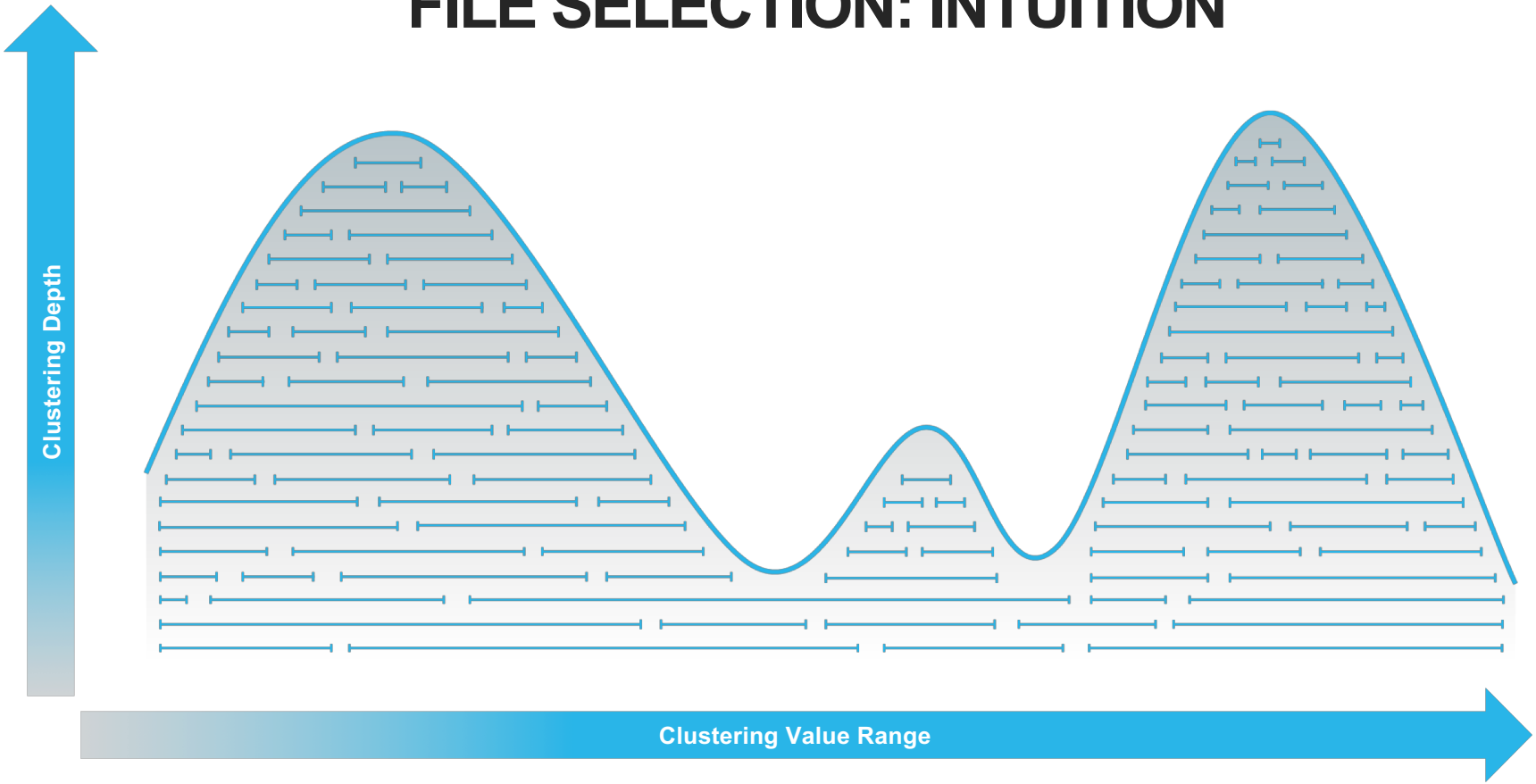
Improved Clustering state
Iteration 2:



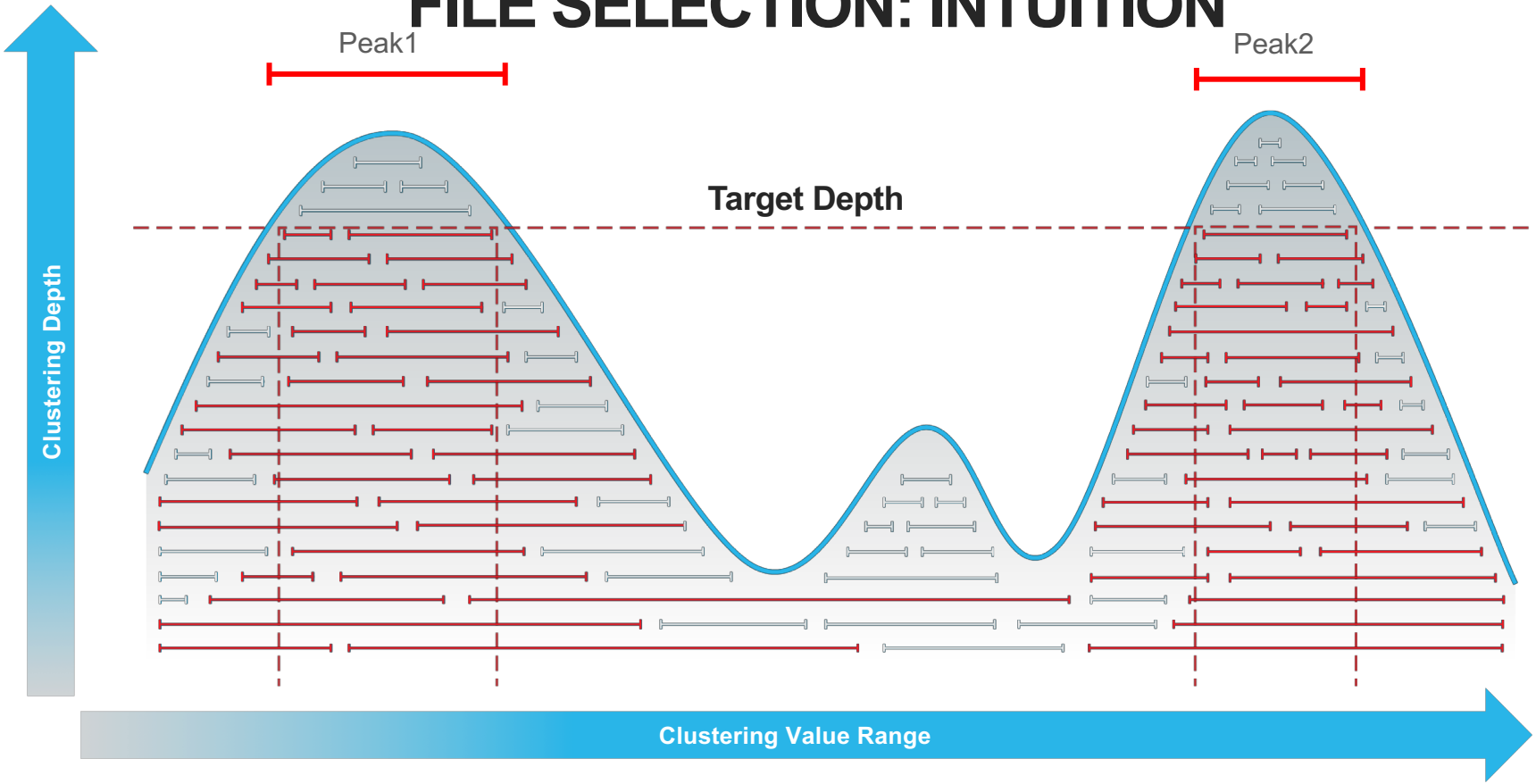
INSIGHT

Reduce Worst Depth by Reducing overlapping (reduce width)

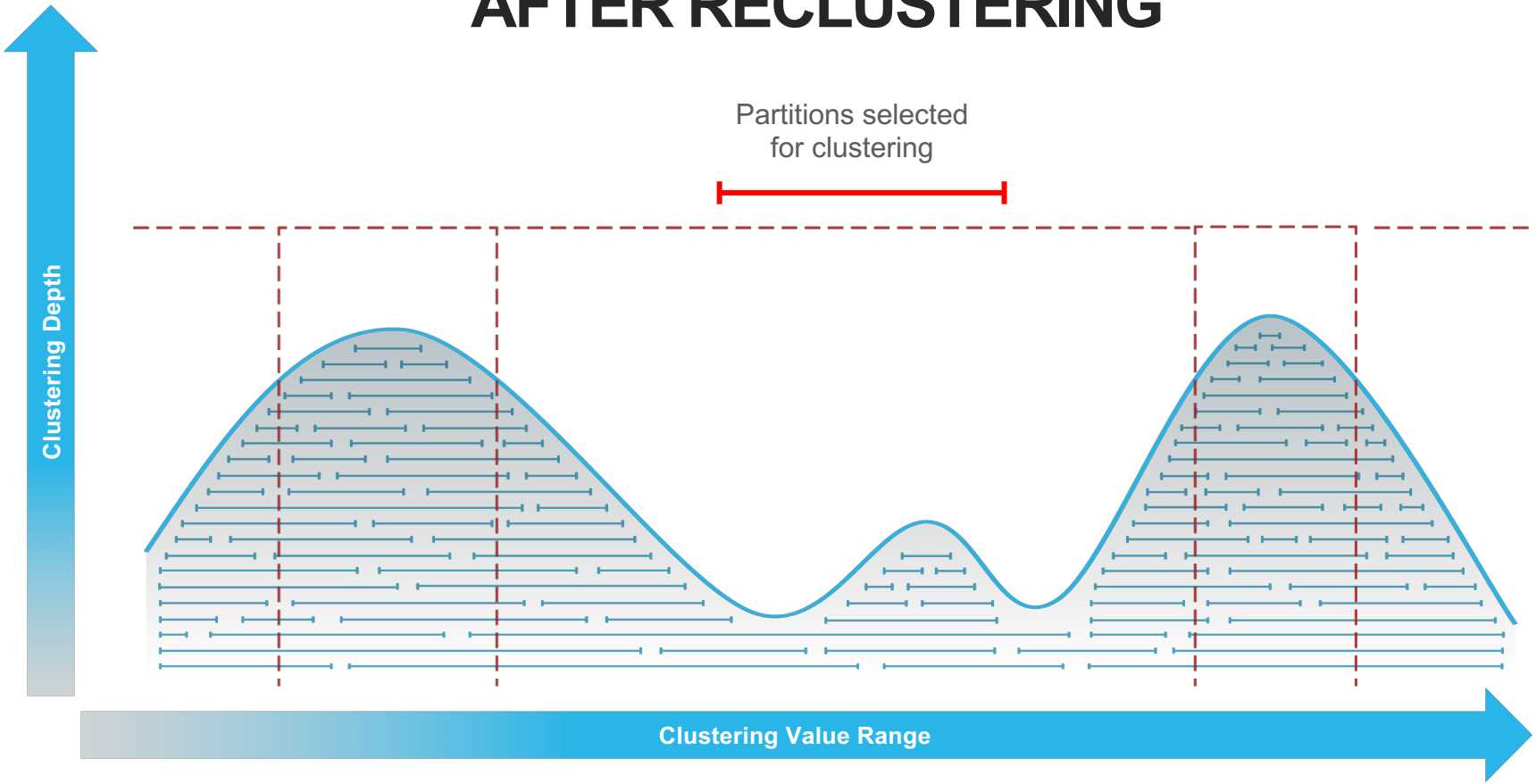
FILE SELECTION: INTUITION



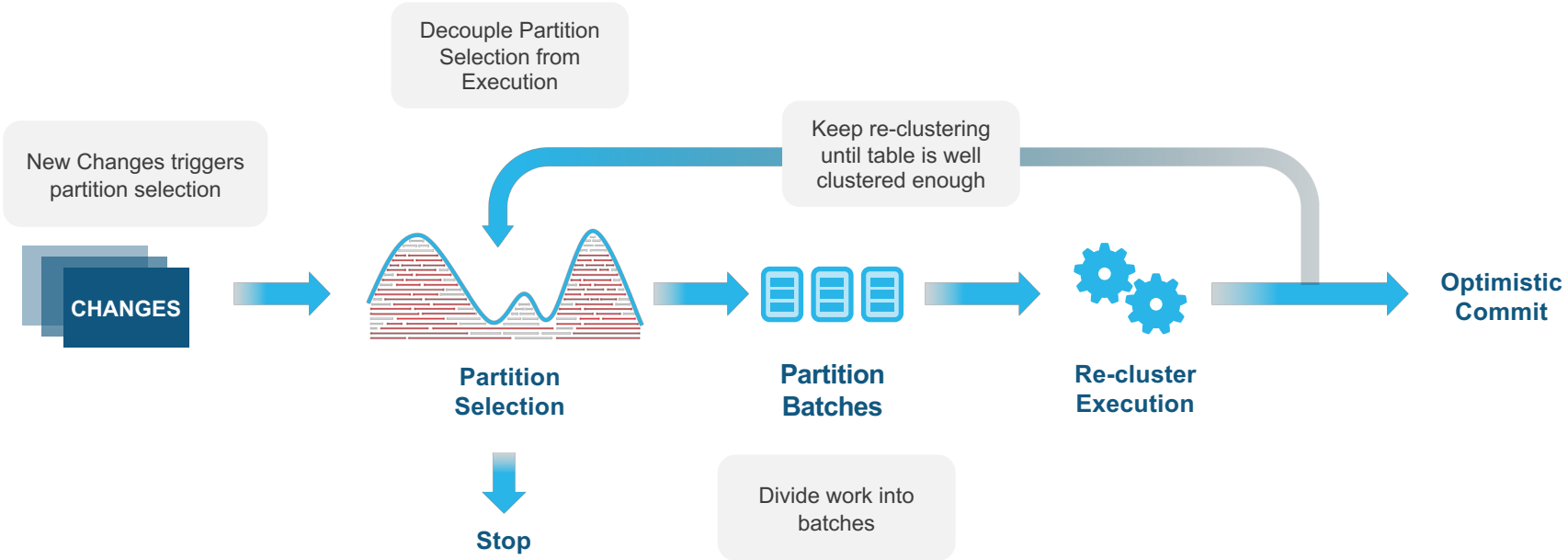
FILE SELECTION: INTUITION



AFTER RECLUSTERING



ALGORITHM OUTLINE

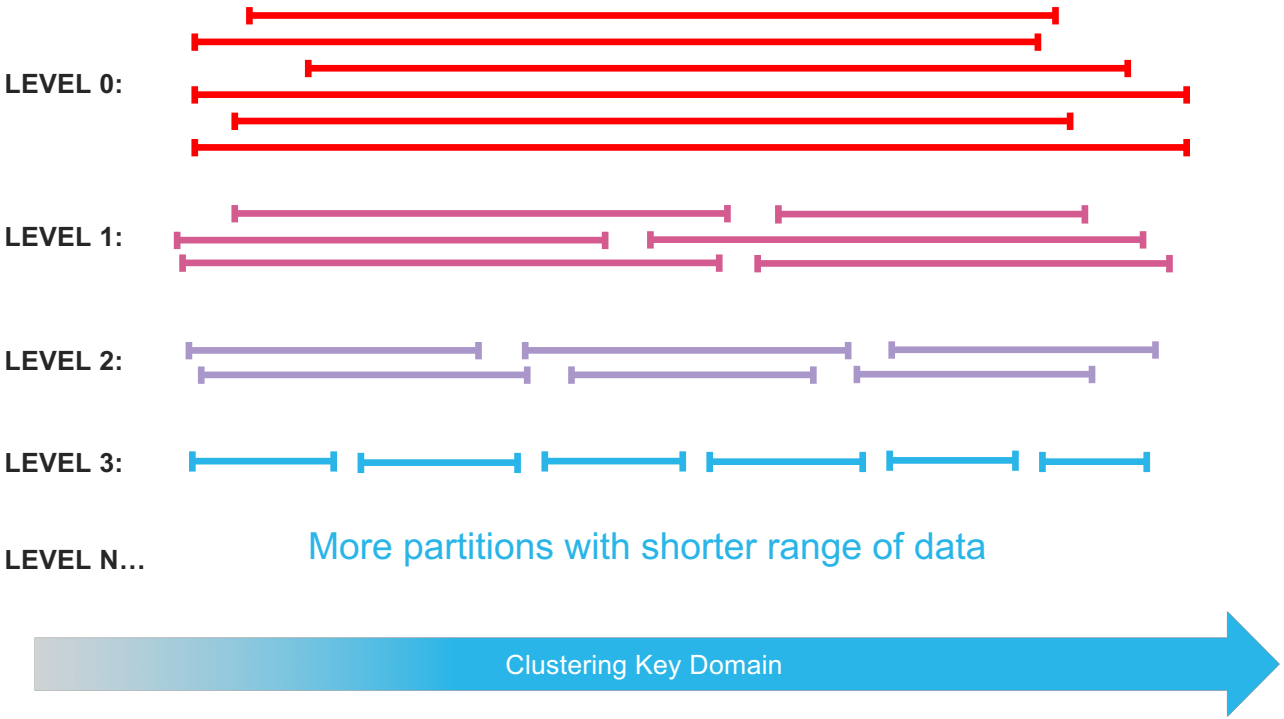


FILE SELECTION ALGORITHM INTERNALS

- Intuition: Work on the peaks
  - Sort the micro-partition endpoints
  - First pass: Compute peak ranges and calculate the number of overlapping micro-partitions
    - Stabbing count array
  - Second pass: For the peak ranges – compute list of partitions ordered by depth
    - MinMaxPriorityQueue

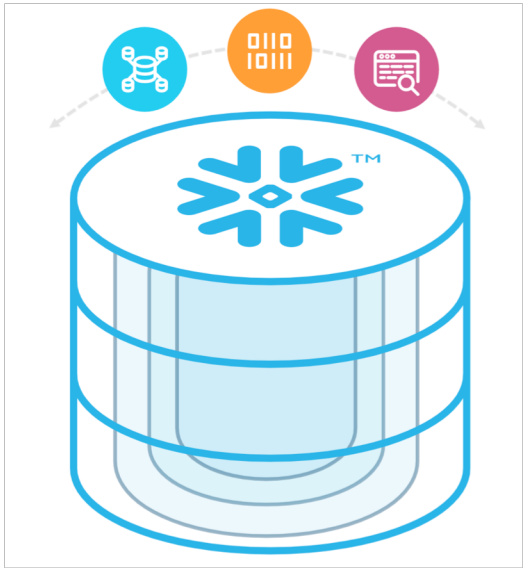
CLUSTERING LEVELS

Reduces clustering width with levels

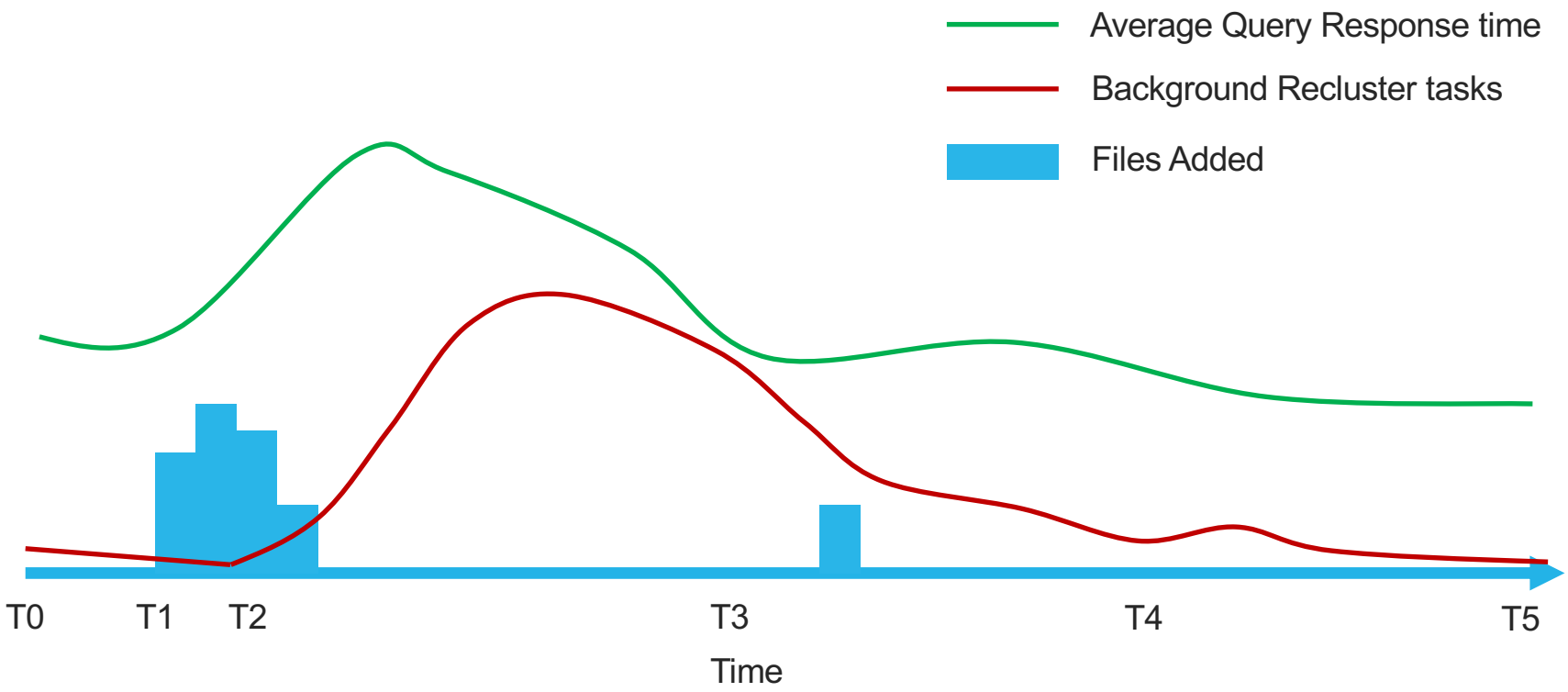


REVIEW

- Wake up when data arrives
- Split the table files into levels (clustering state)
- Run clustering partition selection algorithm on each level
- Queue up “recluster task” for execution
- Scale up and execute re-cluster tasks
  - Sized just right – optimized not to spill
  - Non-blocking – optimistic commit
- If clustering depth is not “good enough” repeat
- Else sleep



EFFECT ON QUERY PERFORMANCE



FUTURE WORK

- **Multi Dimension Keys**
  - “Fake wide partition” problem
  - High cardinality column renders subsequent columns useless
  - Avoid excessive churn
- **Manual cluster keys**
- **Partition selection based on usage**
- **Other linearization functions**
  - Z-order, Grey-order, Hilbert curves
- **Analyze workload to pick best clustering keys automatically**
- **Partition selection Phase 2**

JOIN US!

Ambitious Projects > Smart People > Immediate Impact > Fun Culture

One of the “Best Places to Work” every year

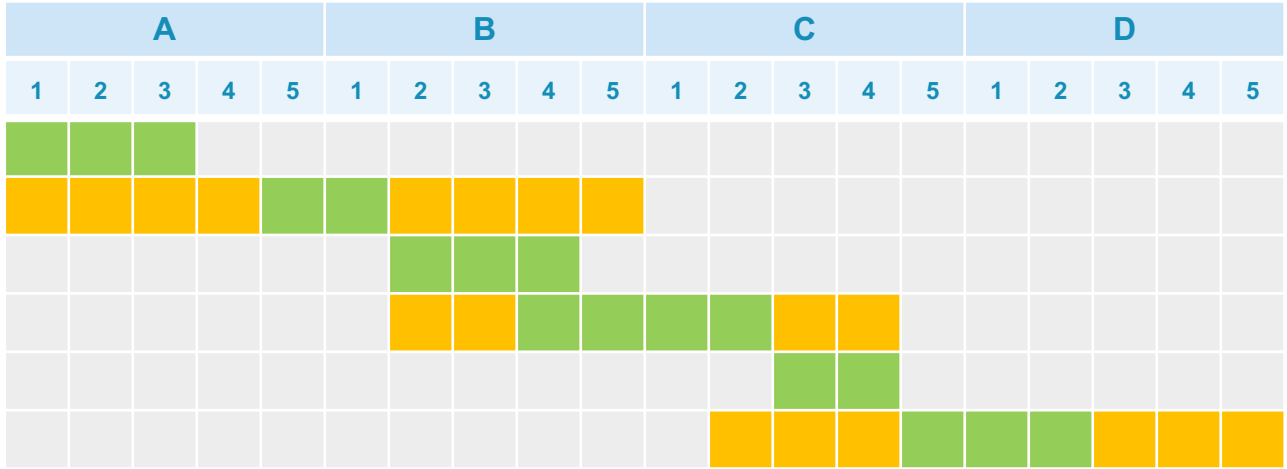
Where else can you compete with Amazon, Google and Microsoft? ☺

THANK YOU

Jiaqi Yan

“FAKE WIDE PARTITIONS”

Actual Min/Max		Column Metadata Min/Max	
A-1	A-3	A-A	1-3
A-5	B-1	A-B	1-5
B-2	B-4	B-B	2-4
B-4	C-2	B-C	2-4
C-3	C-4	C-C	3-4
C-5	D-2	C-D	2-5



CASE STUDY

- > 1PB table, trickle load every 3 minutes
- Provides dashboard service to customers
- Sub-second query time SLA
- Query by application IDs
- Huge skews cross applications
- Cluster by (app_id, time, name)

DASHBOARD