Impression Store:
Compressive Sensing-based Storage for Big Data Analytics

Jiaxing Zhang, **Ying Yan**, Liang Jeff Chen, Minjie Wang,
Thomas Moscibroda & Zheng Zhang

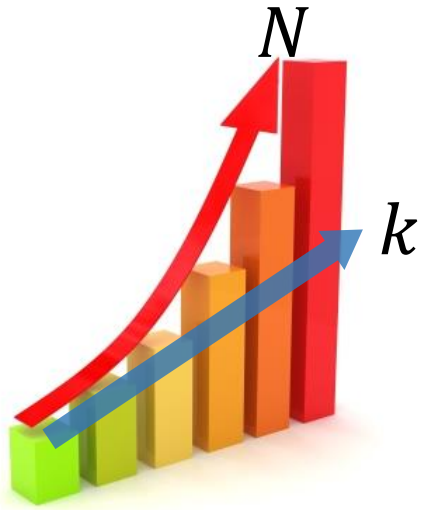
Microsoft Research

The Curse of O(N) in Big Data Era

- In the old days, an $O(N)$ algorithm was efficient
  - But what if N is increasing fast?
  - Parallelism is only a partial solution

$$O(N) \longrightarrow O\left(\frac{N}{k}\right)$$

k: # of machines



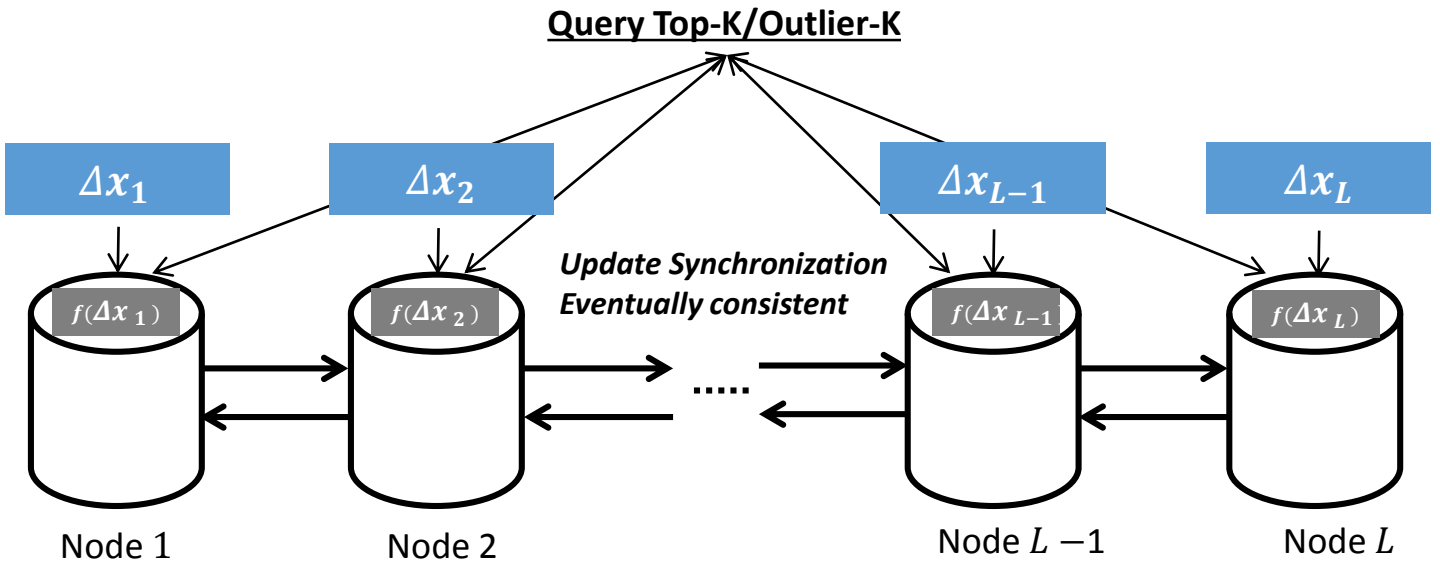
- It's an illusion that we can compute against all the data
  - What we collect is always a sample
  - By the time we finish computing, new data has generated

Approximate Results Suffice!

Impression Store

- Provides an abstraction of big data vectors
  - Support retrieval of big data components
  - Store impression information rather than raw data
- Improvements the performance
  - Save storage capacity
  - Save IO bandwidth
  - High scalability

System Design



Key technique:
Compressive Sensing

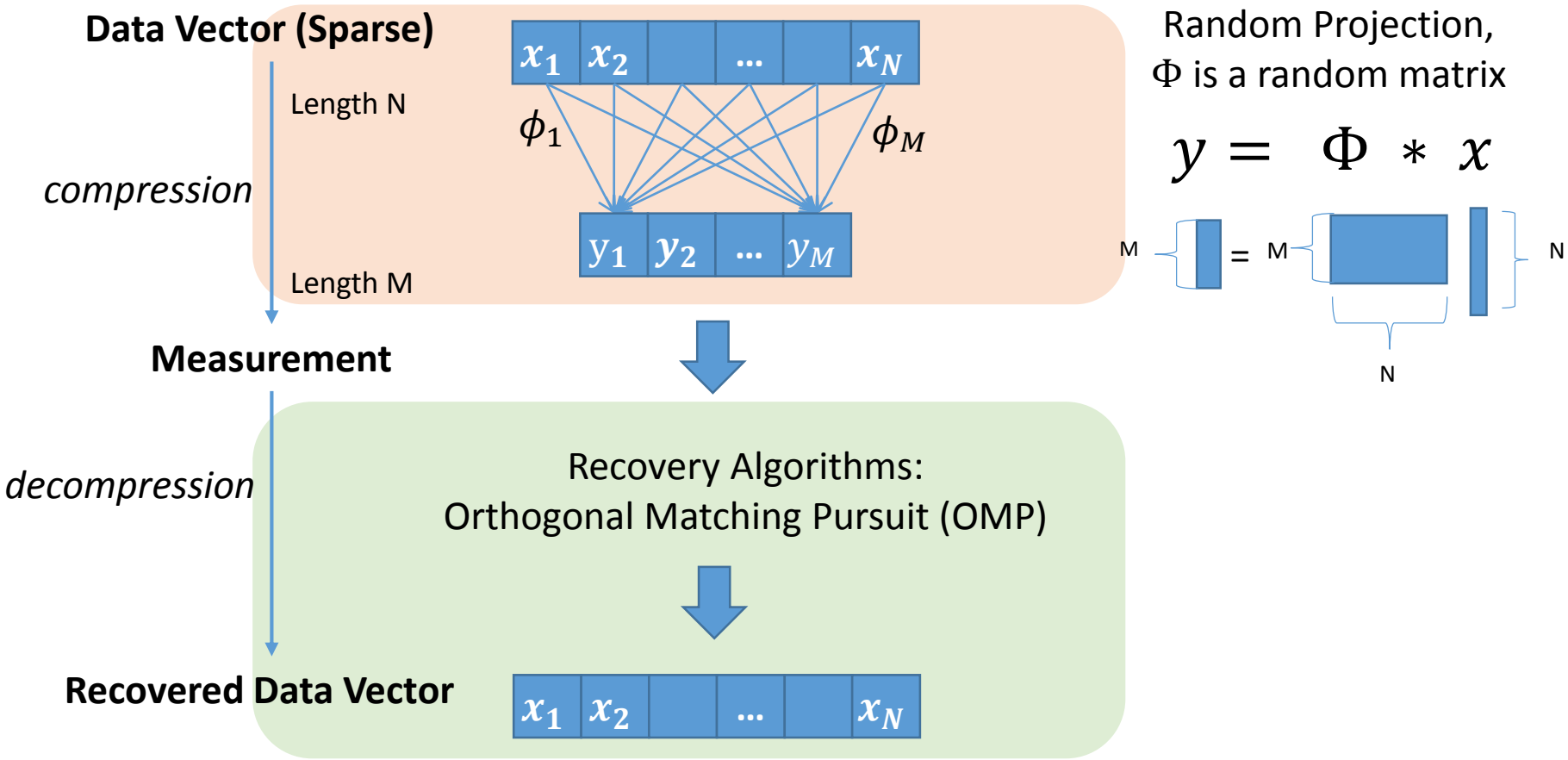
1. High Scalability: Any node can process any update / query → Synchronization

2. Storage, memory, IO and communication cost efficient → Compression

3. High throughput to Impression Query- Top, Outlier & mode → Incremental updates on compression domain

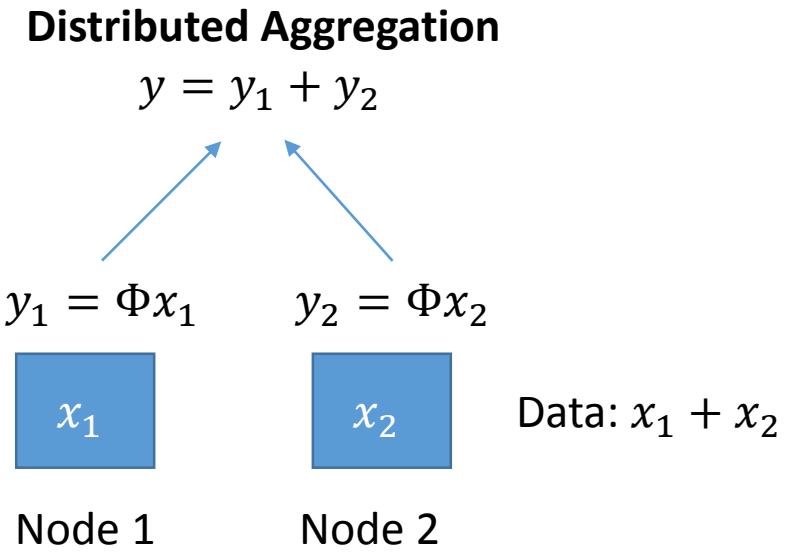
→ Uncompressing big components only

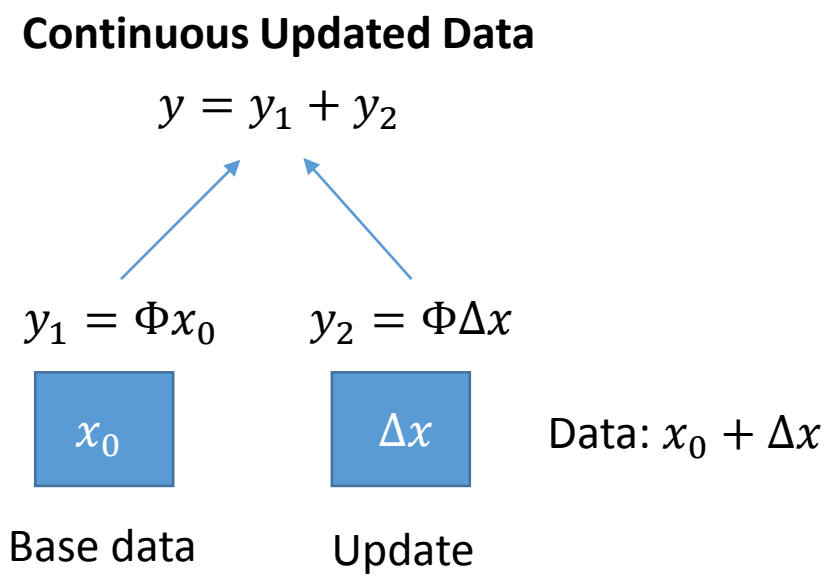
Introduction of Compressive Sensing



Compressive Sensing vs. Compression

- Decomposable Compression:
  - $x = x_1 + x_2$ ➔ $y = \Phi x = \Phi(x_1 + x_2) = \Phi x_1 + \Phi x_2 = y_1 + y_2$

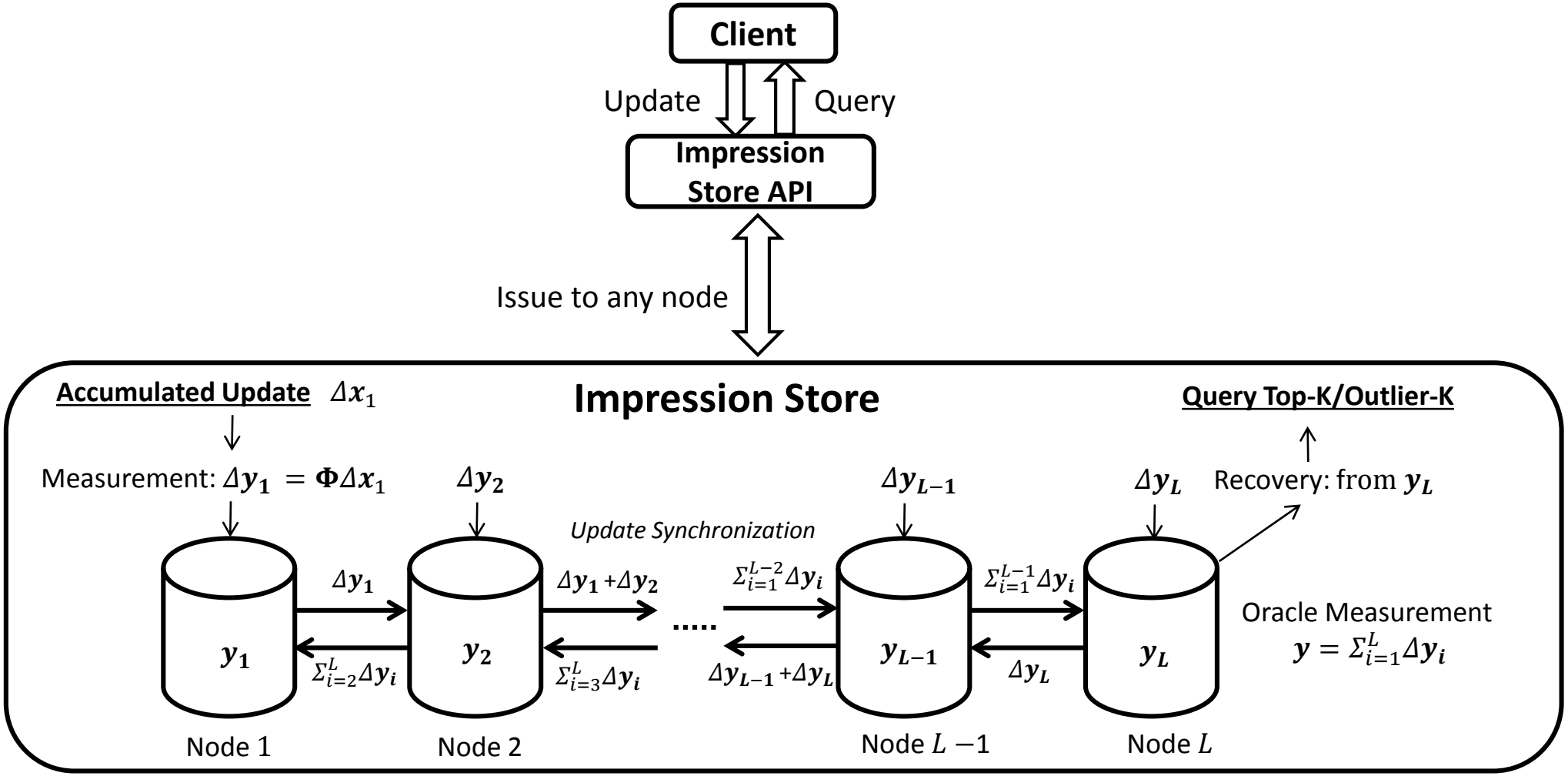




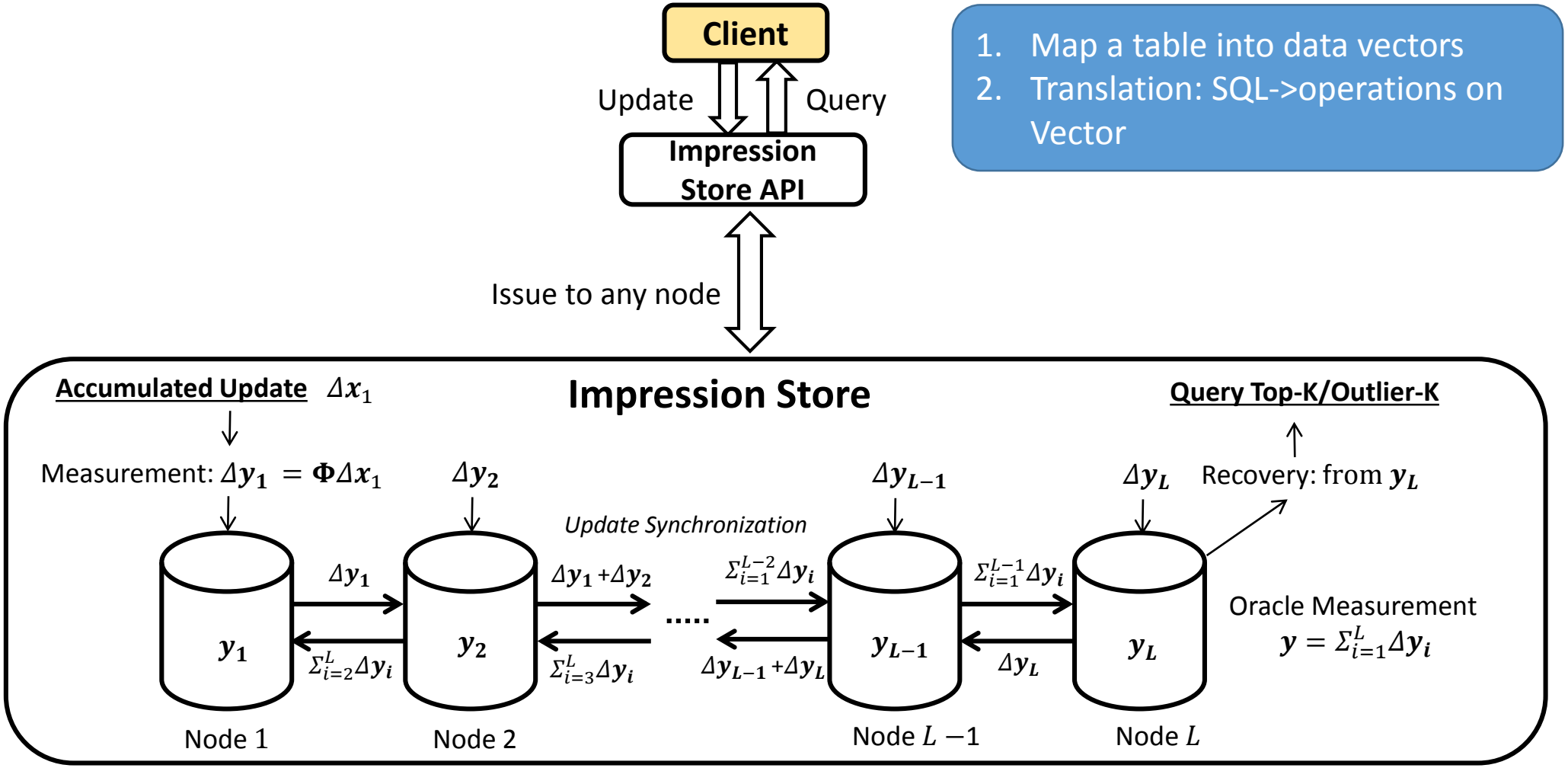
- Recovery Algorithm (OMP)/Our BOMP

  Big components have more precision

Architecture

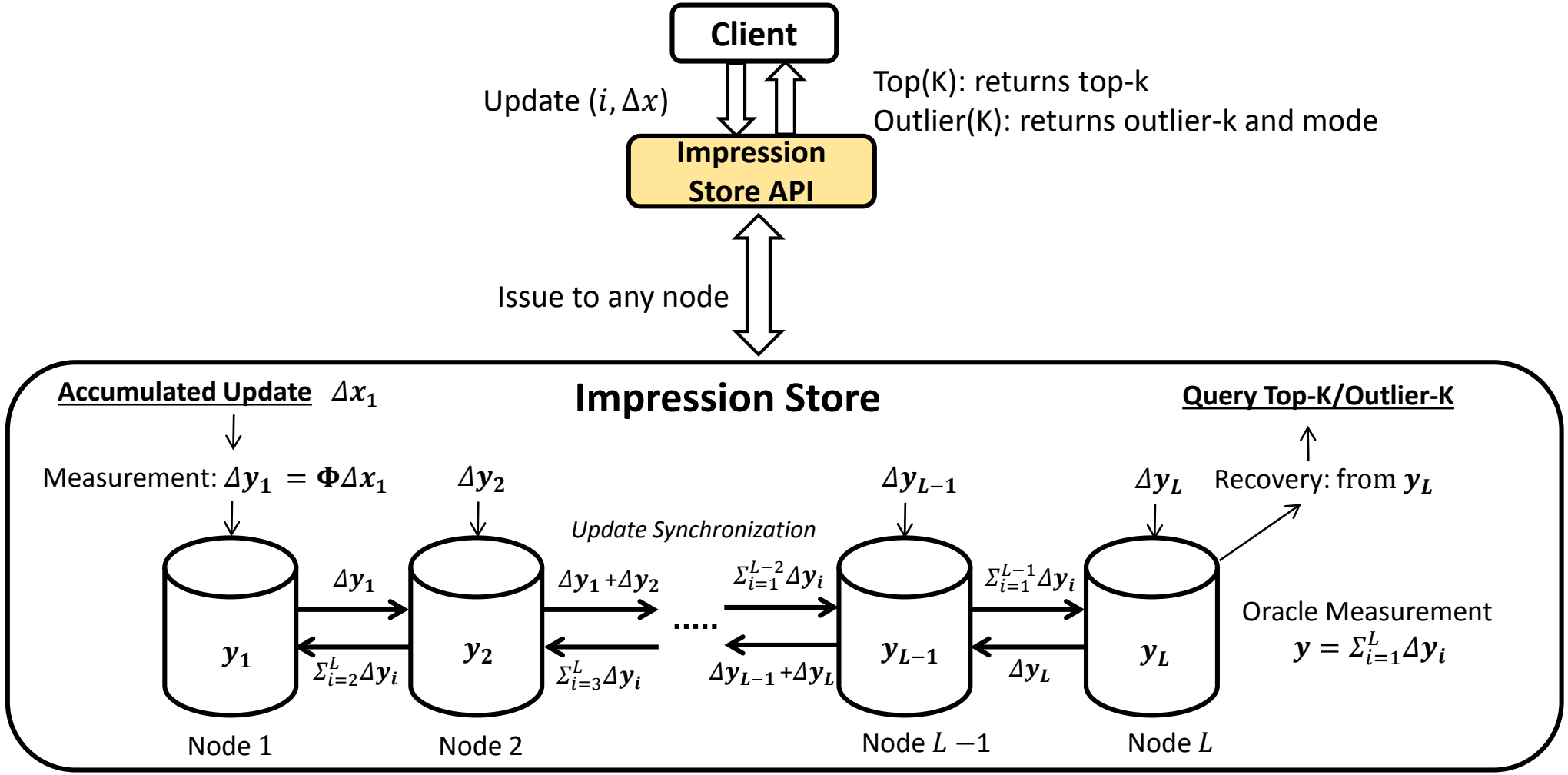


Client



1. Map a table into data vectors
2. Translation: SQL->operations on Vector

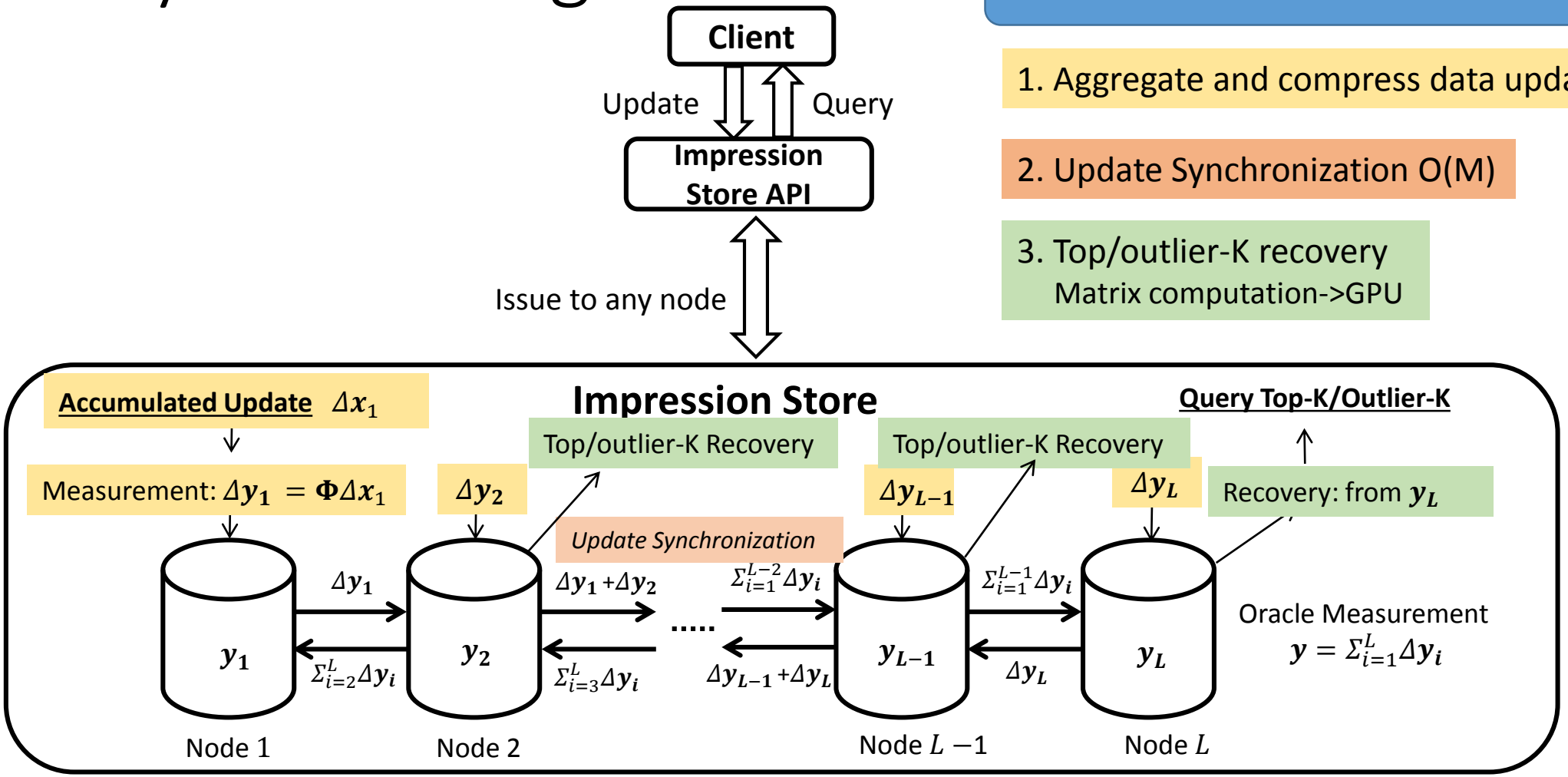
Architecture



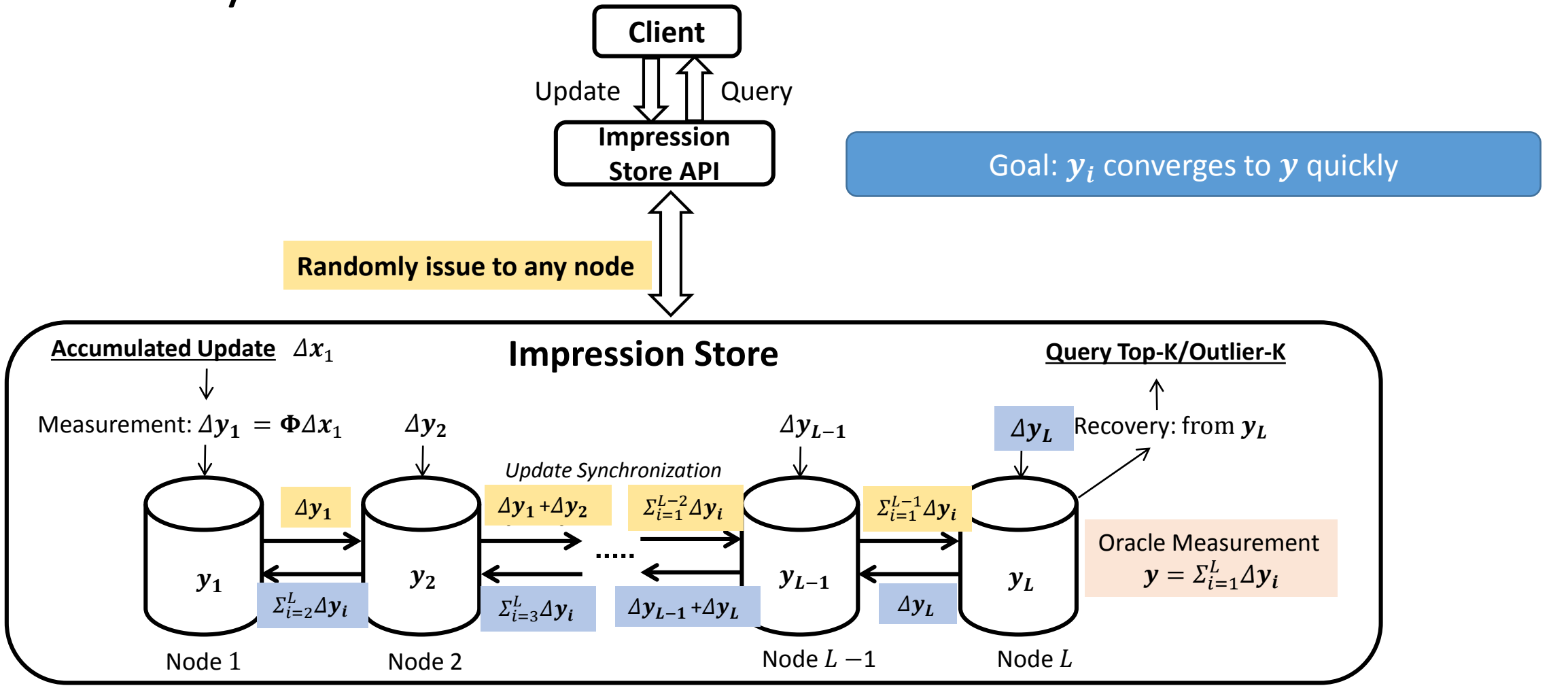
Query Processing

Each node continuously works on three tasks:

1. Aggregate and compress data updates
2. Update Synchronization O(M)
3. Top/outlier-K recovery
   Matrix computation->GPU

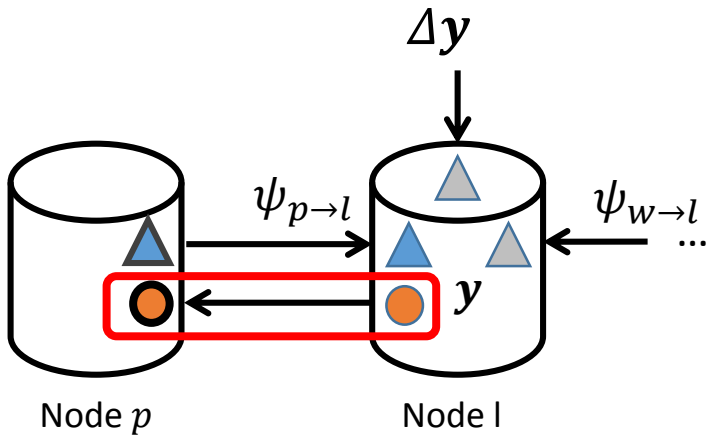


Update Synchronization



Update Synchronization

- Synchronization policy

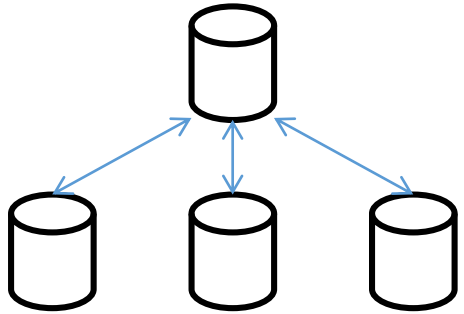


$$\boldsymbol{y} = \Delta \boldsymbol{y} + \sum_{q \in N(l)} \psi_{q\to l}$$

Send to p: $\psi_{l\to p} = \Delta \boldsymbol{y} + \sum_{q\in N(l)} \psi_{q\to l} - \psi_{p\to l} = y - \psi_{p\to l}$

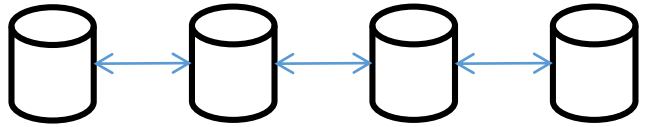
Each pair of Send-Receive copies my not be the same all the time!
The policy is proved to achieve eventual consistency.

- Loop-free topology



Master-Slave tree structure
- Small latency
- Load is not balanced

Topology in between - trade off



Line structure
- Long latency
- Load is balanced

Optimizations

- Speed up the recovery $\mathrm{O}(M^2N)$
  - GPU: 30~40X speed-up
  - For continuous updates
    - Optimizing the recover algorithm by keeping the positions of the last recovery
    - Reduce the complexity to $\mathrm{O}(M^3)$

Experiment Setup

- Error metrics

$$\textbf{Error on Position}: E_P = 1 - |I \cap \tilde{I}|/K.$$
$$\textbf{Error on Value}: E_V = \|\tilde{\mathbf{x}}_{\tilde{I}} - \mathbf{x}_I\|_2 / \|\mathbf{x}_I\|_2.$$

Example

Ground truth

Key	Value
1	100
2	80
3	50
4	30
5	10

Approximation

Key	Value
1	100
2	80
3	48
4	25
6	11

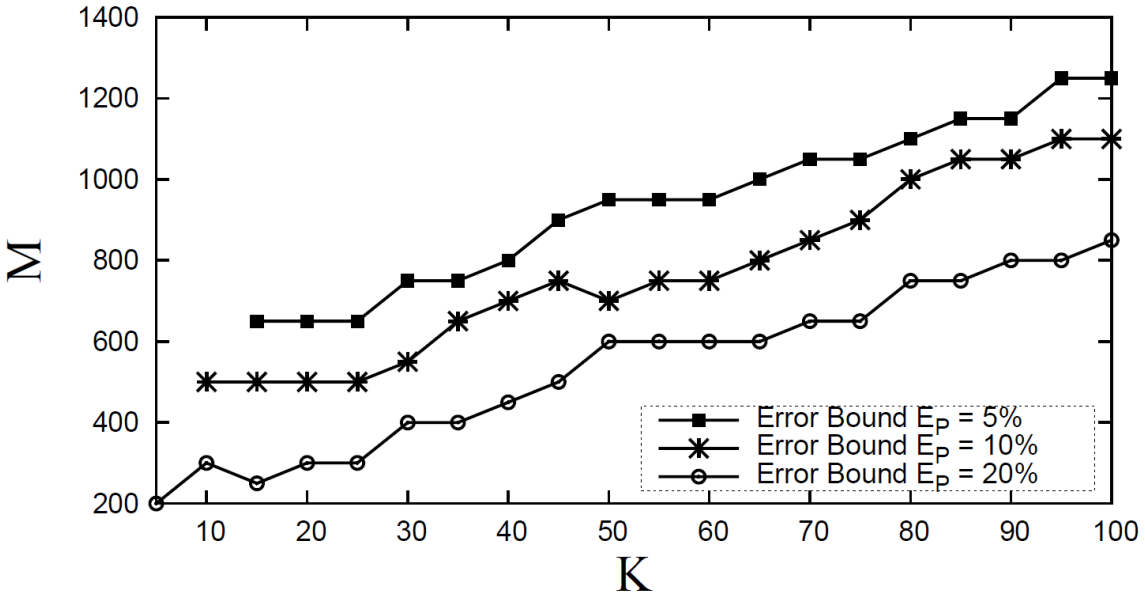
$$E_p = 1 - \frac{4}{5} = 20\%$$

$$E_v = 3.88\%$$

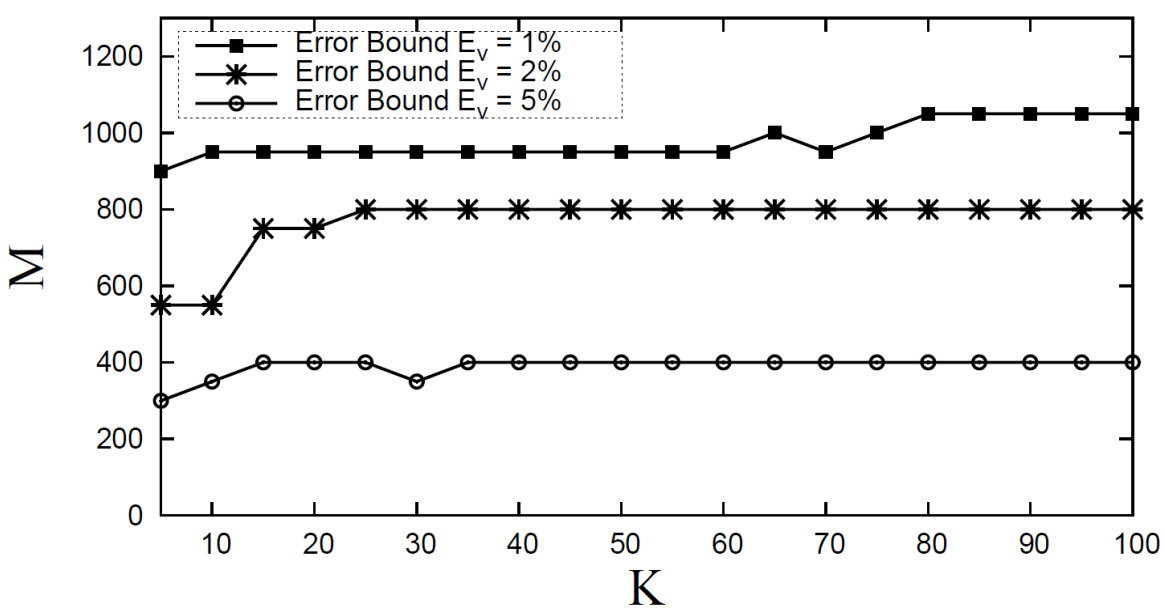
- Workload:
  - Revenue on Ads entries in Bing Search engine Group by 6 attributes (Market, Vertical, QueryClass…)
  - Totally N=12,891 user-interested entries in the vector

Preliminary Results (1)

- Effect of M and N on Recover Quality



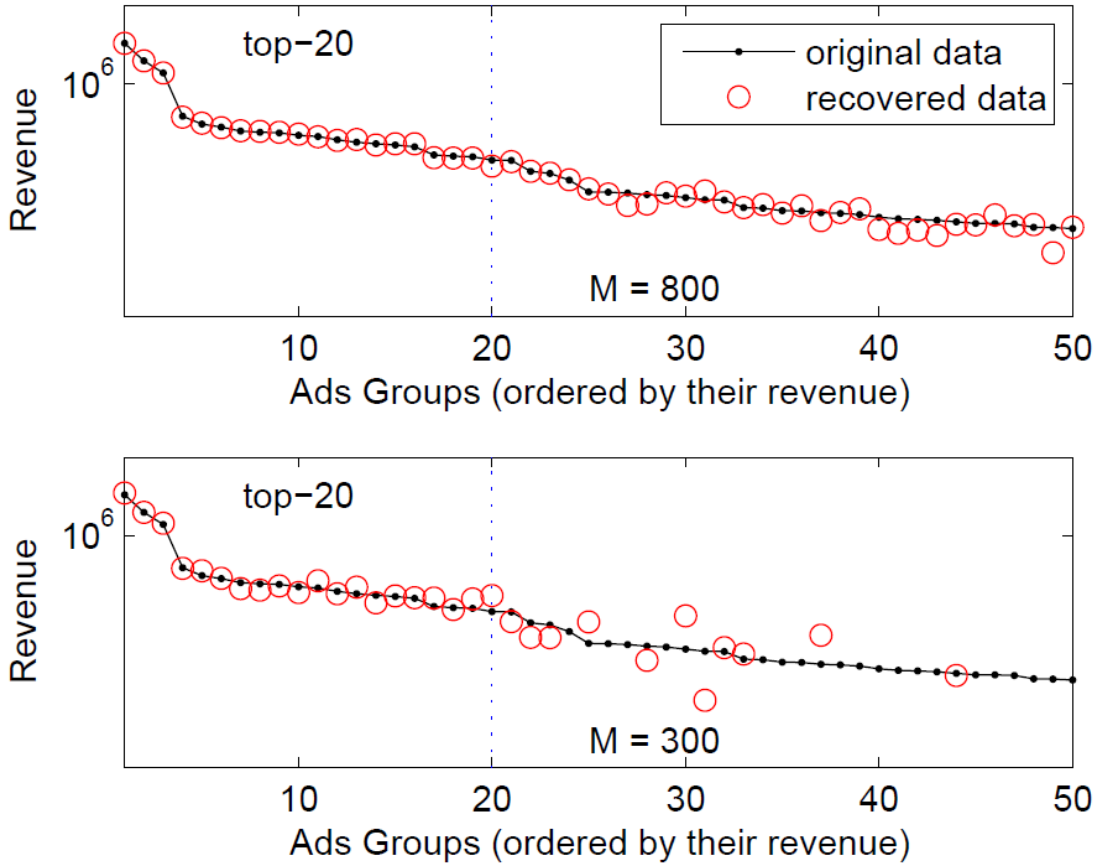
E_p



E_v

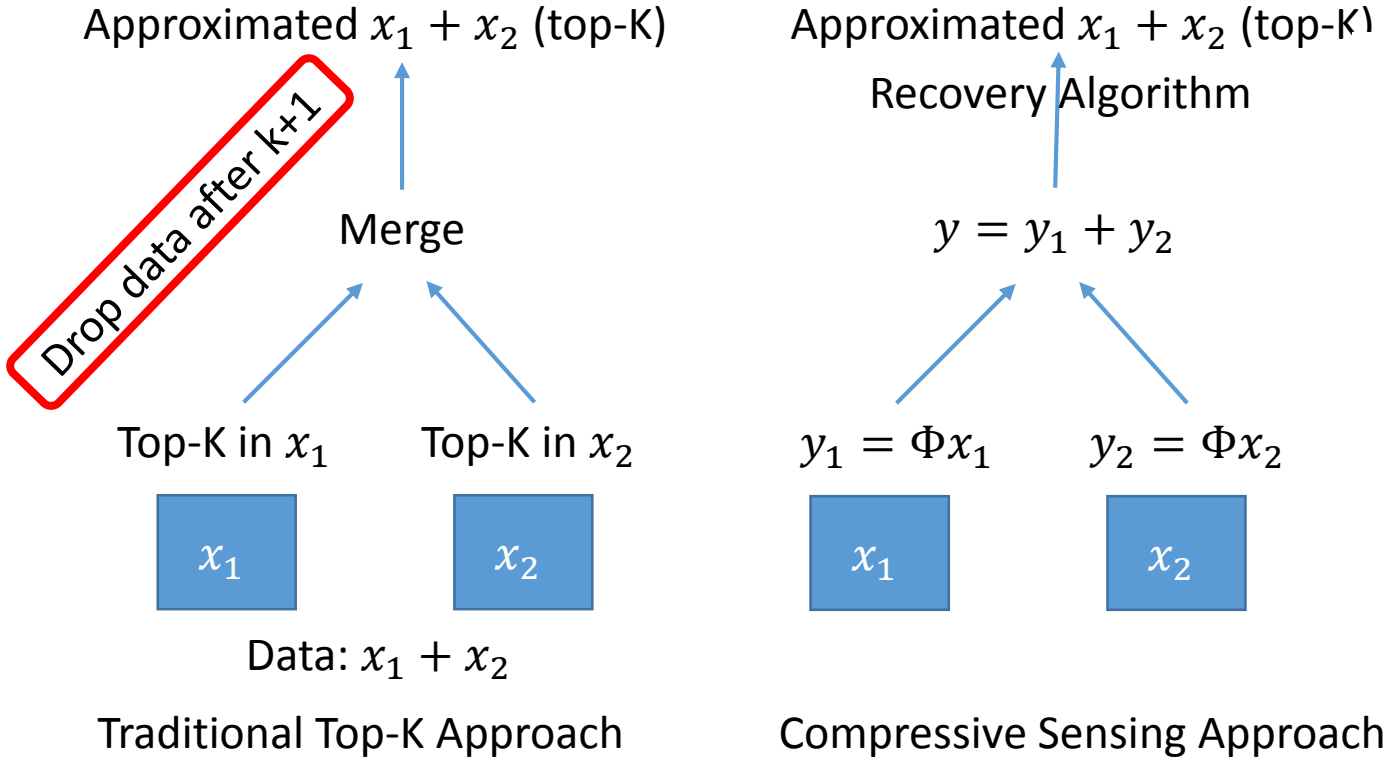
Preliminary Results (2)

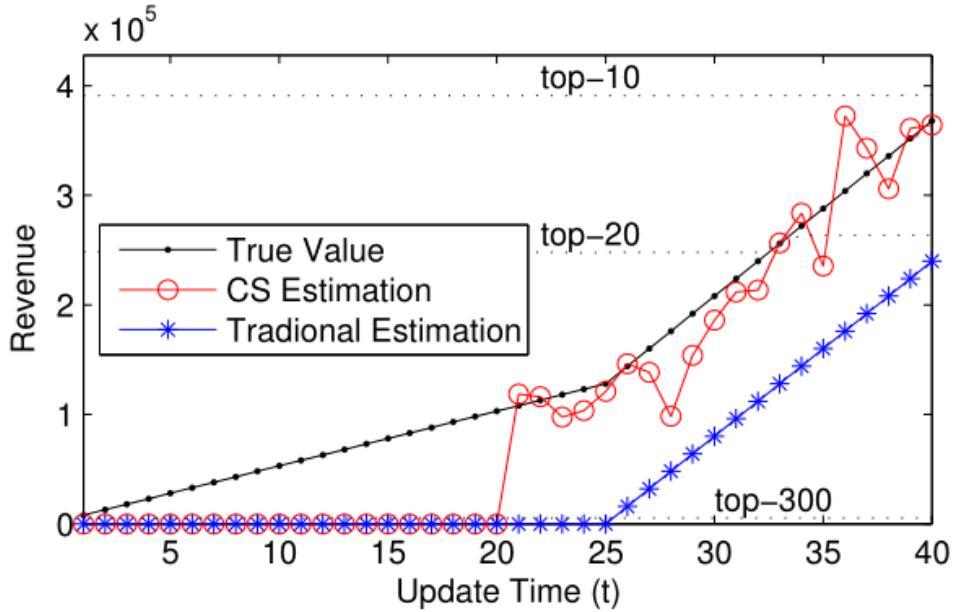
- Bigger value can be recovered much more accurate with smaller M



Preliminary Results (3)

- Compare with traditional Top-K only approach (K=M)





Ongoing and Future work

- Support more sophisticated queries
  - Exploring CS and other techniques
- Work together with sampling
  - Multiple parallel queries to different nodes can improve confidence

Thanks!