INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING
ICPP/2021/CHICAGO/USA
acm In-Cooperation
sighpc
AUGUST 9-12, 2021

Using Vectorized Execution to Boost SQL Query Performance on Spark

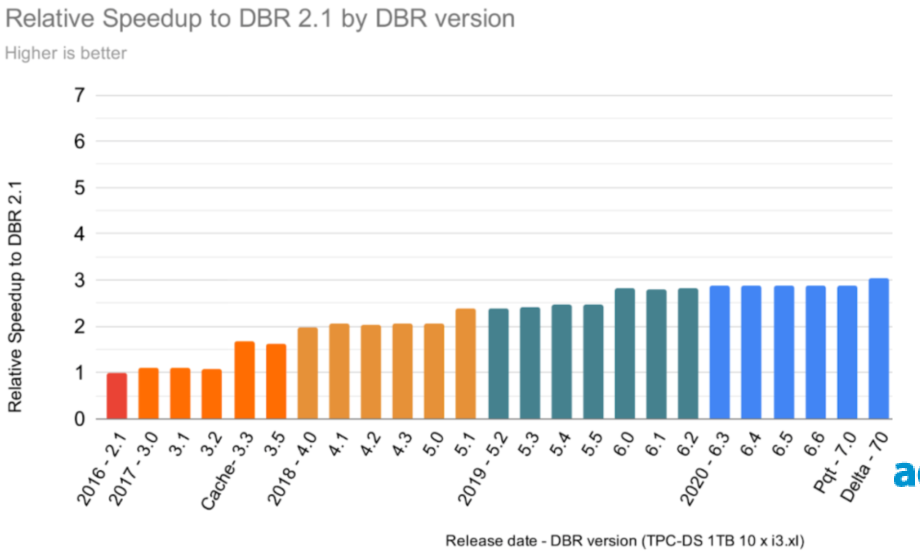
Yijie Shen†‡, Jin Xiong†‡, Dejun Jiang†‡

† Institute of Computing Technology, CAS, Beijing, China
‡ University of Chinese Academy of Sciences, Beijing, China

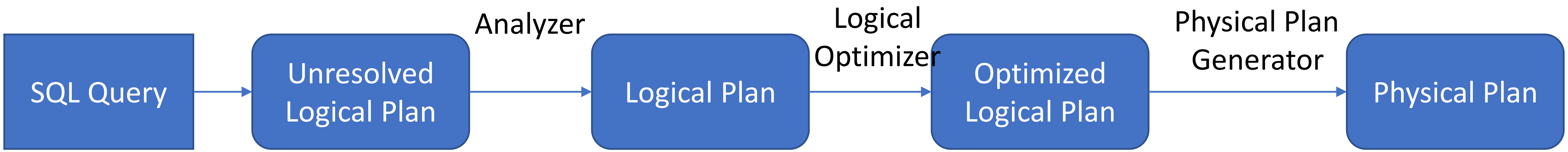
The pursuit of high-performance query processing

- People never get satisfied with data processing speed
  - General-purpose DBMS
  - OLAP (HyPer / MonetDB)
  - SQL-on-Hadoop (MPP / MapReduce-based SQL engines)

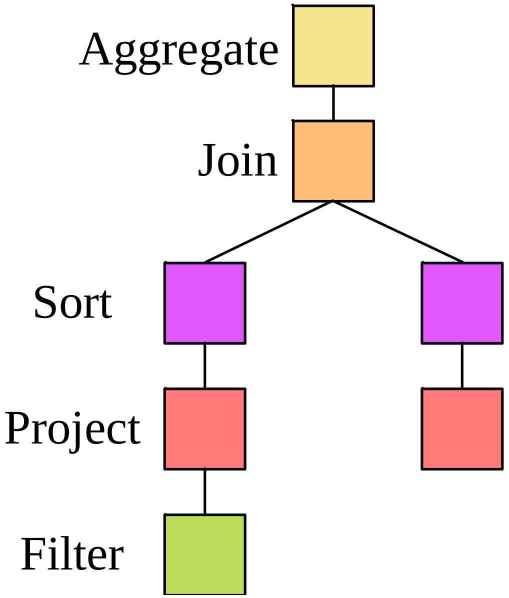
- The speed of the Spark engine itself is also accelerating
  - ~3x speed up since its first release

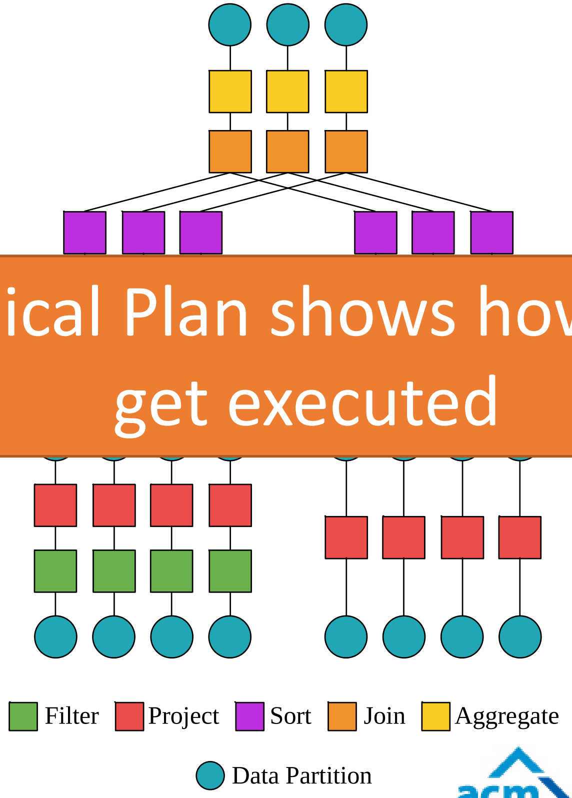


Physical-Op is the basic execution unit



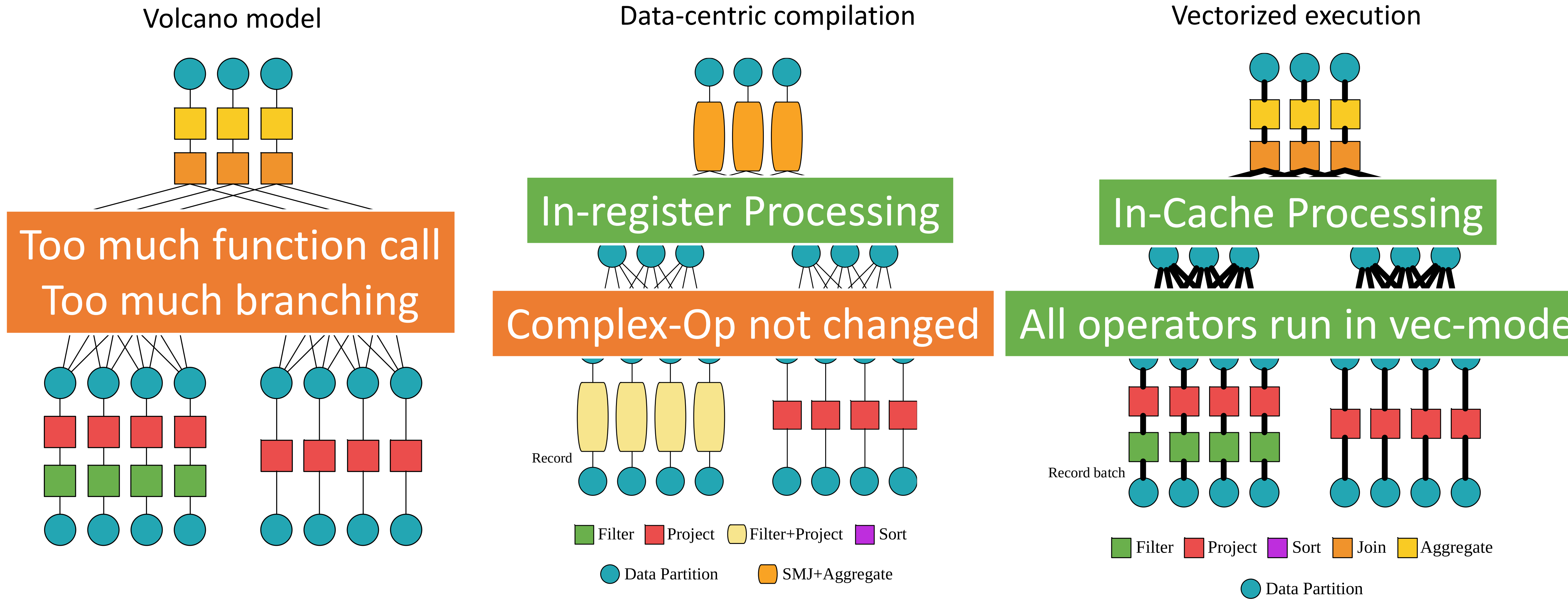

```
SELECT a1, SUM(5 * a2), AVG(b2 + 4)
FROM A, B
WHERE A.a1 = B.b1 AND A. a2 > 10;
```



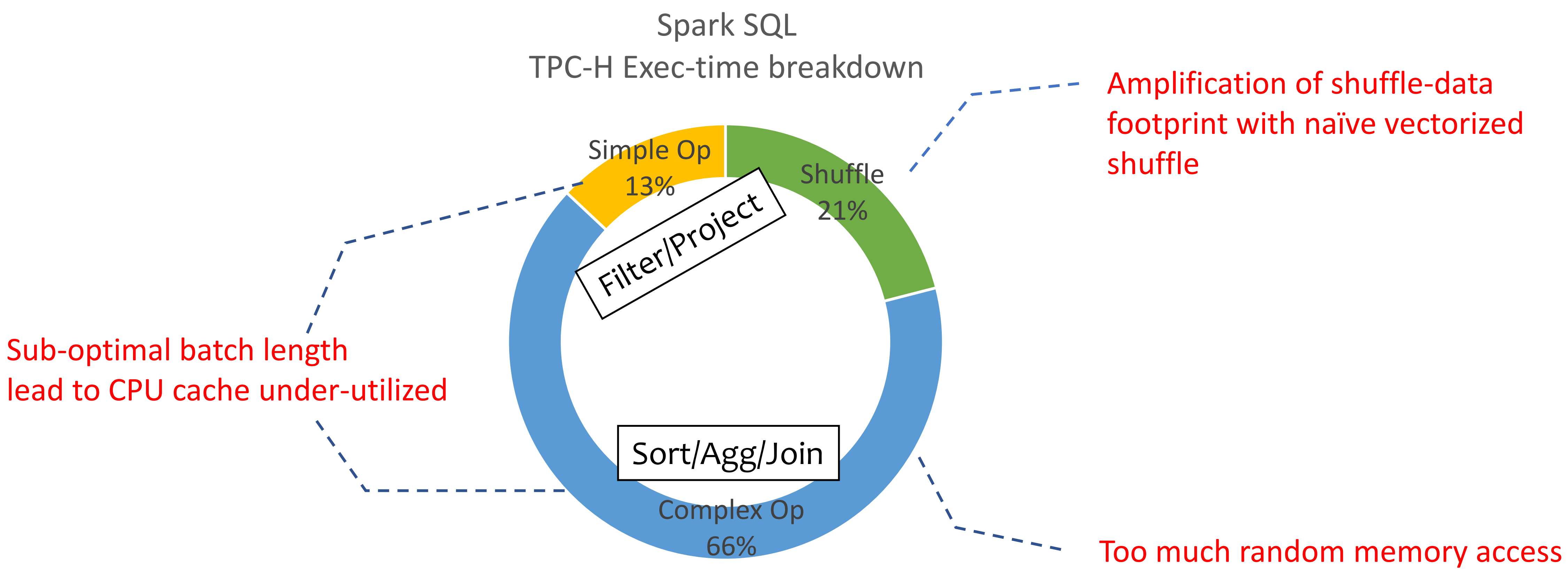


Physical Plan shows how SQL get executed

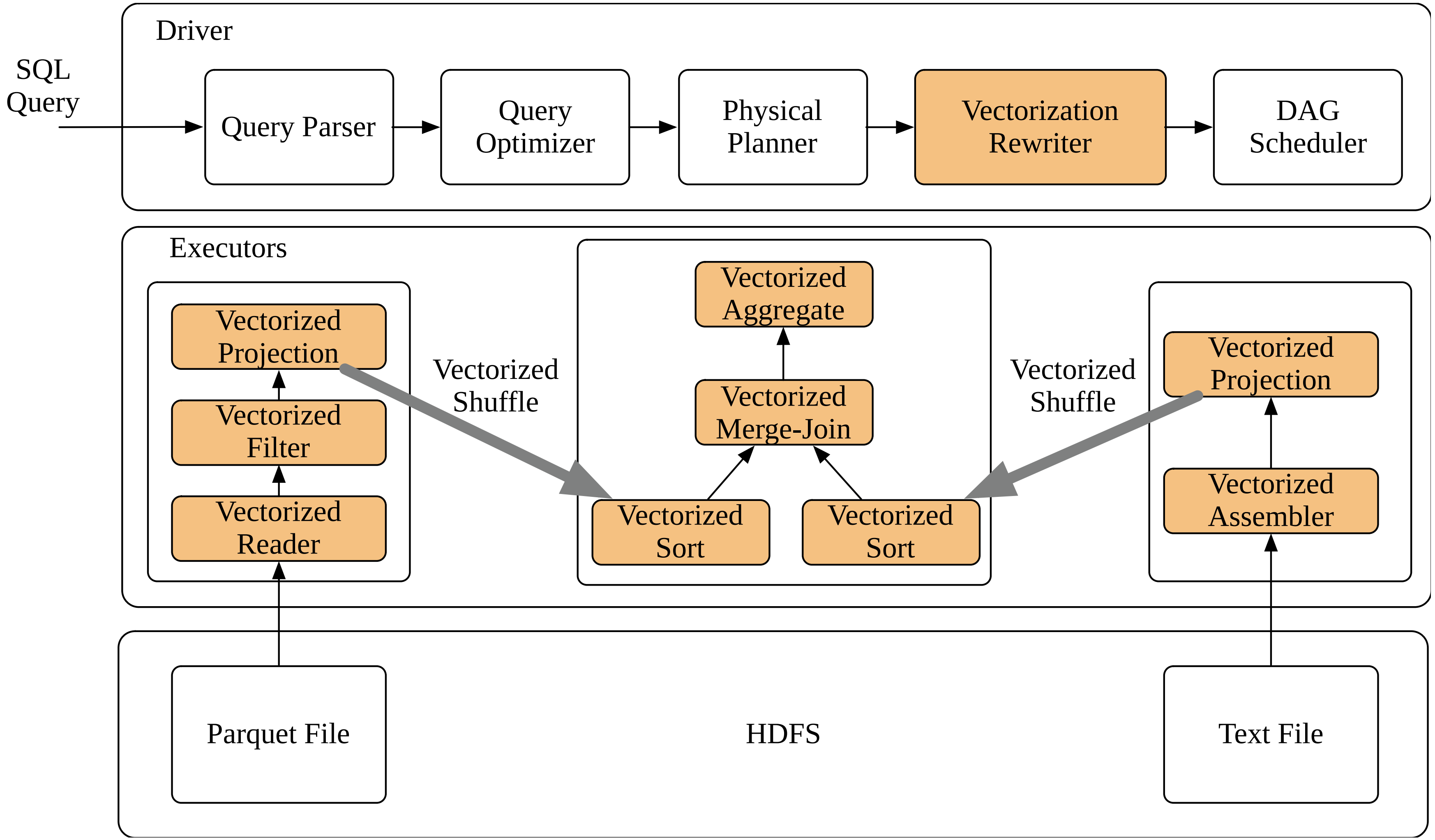
The execution mode for Physical-Op matters



Breaking-down the benchmark execution time

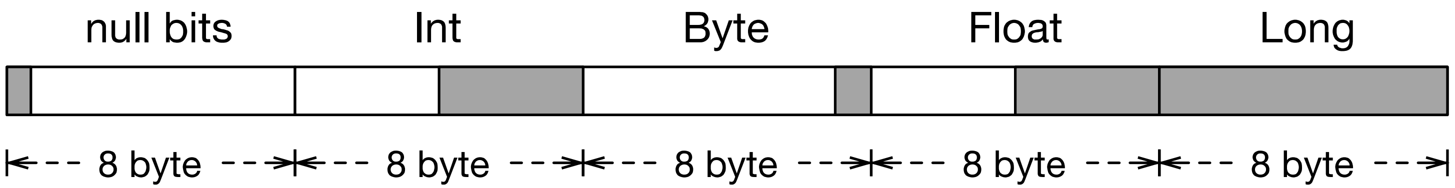


VEE architecture

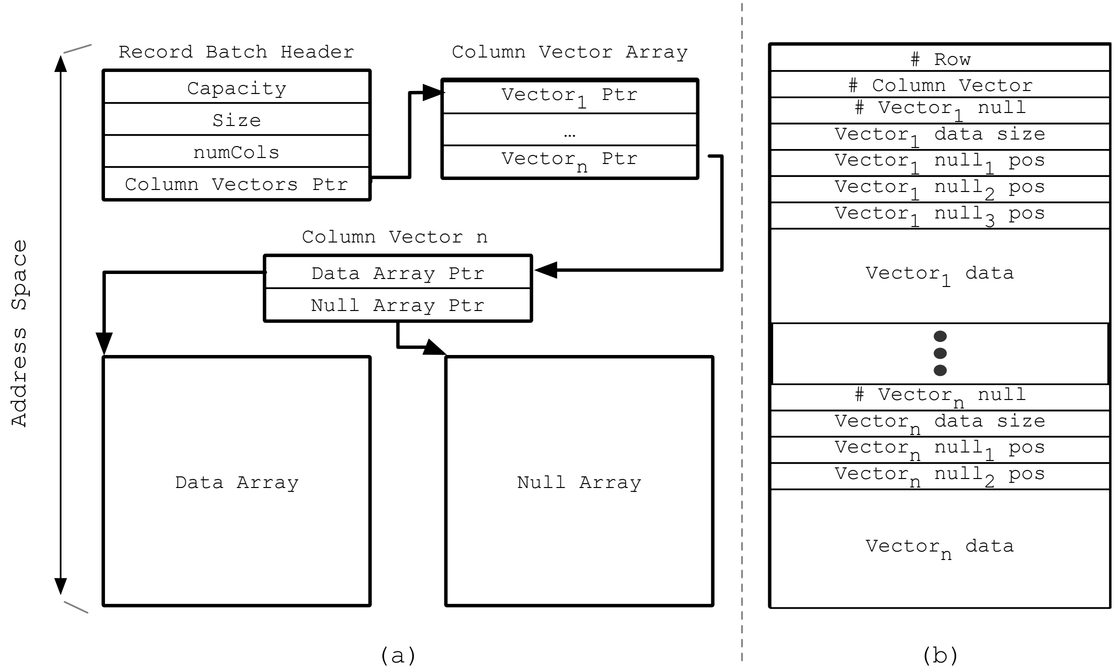


Vectorized Data Structures

UnsafeRow in vanilla Spark



Record batch
Runtime Repr



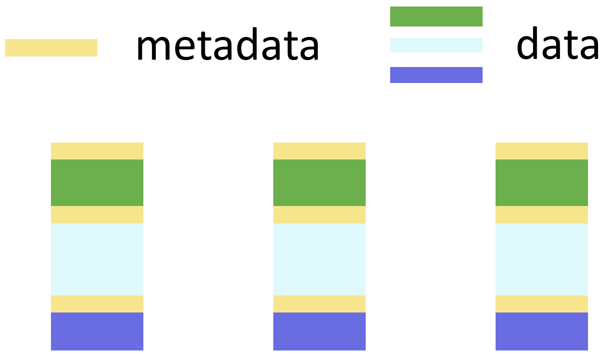
Record batch
In memory & disk

INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

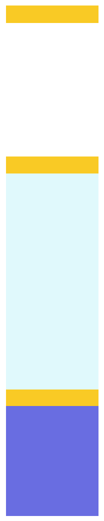
acm In-Cooperation sighpc

Vectorized Shuffle based on Serialization-aware assembling

- Naïve vectorized shuffle
  - Metadata takes too much space
  - Shuffle data amplification
    - CPU, I/O inefficiency



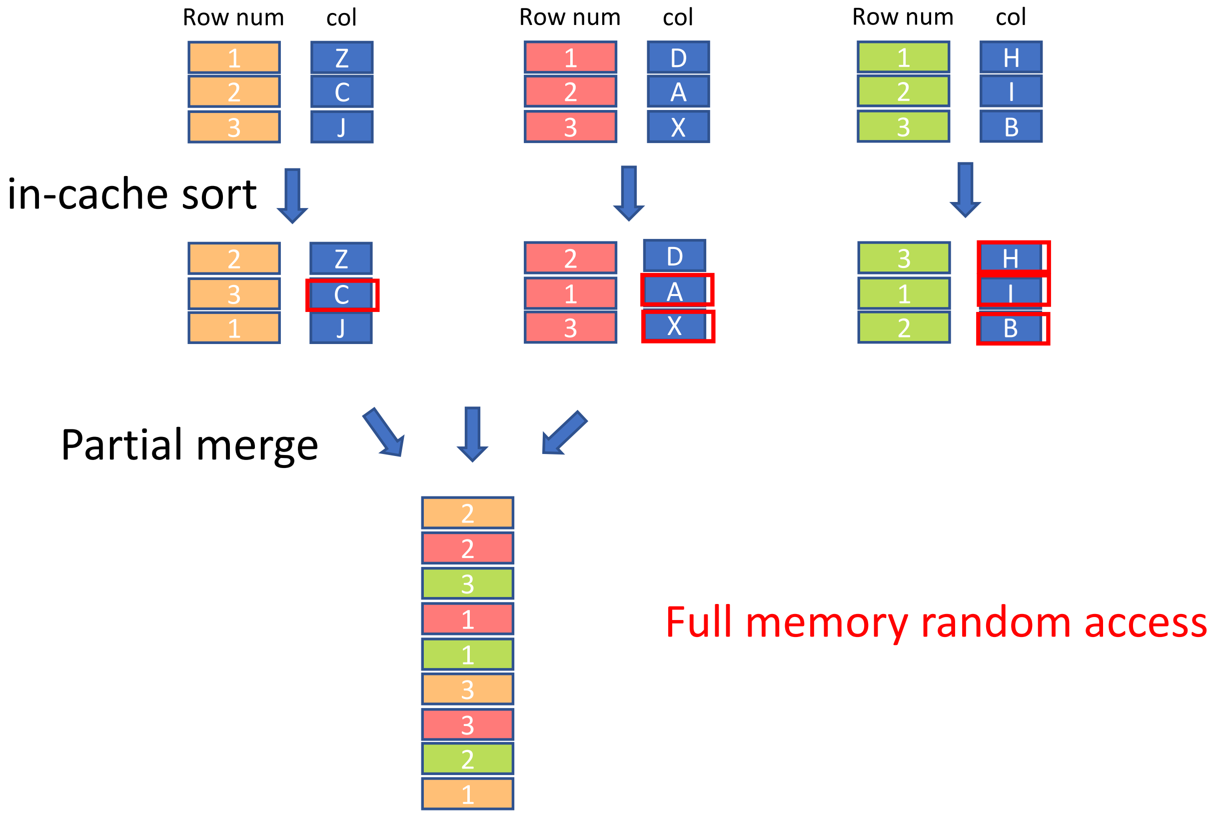
- **Serialization-aware assembling**
  - assembling overhead is significantly reduced
  - the advantages of vectorized shuffle is maintained.

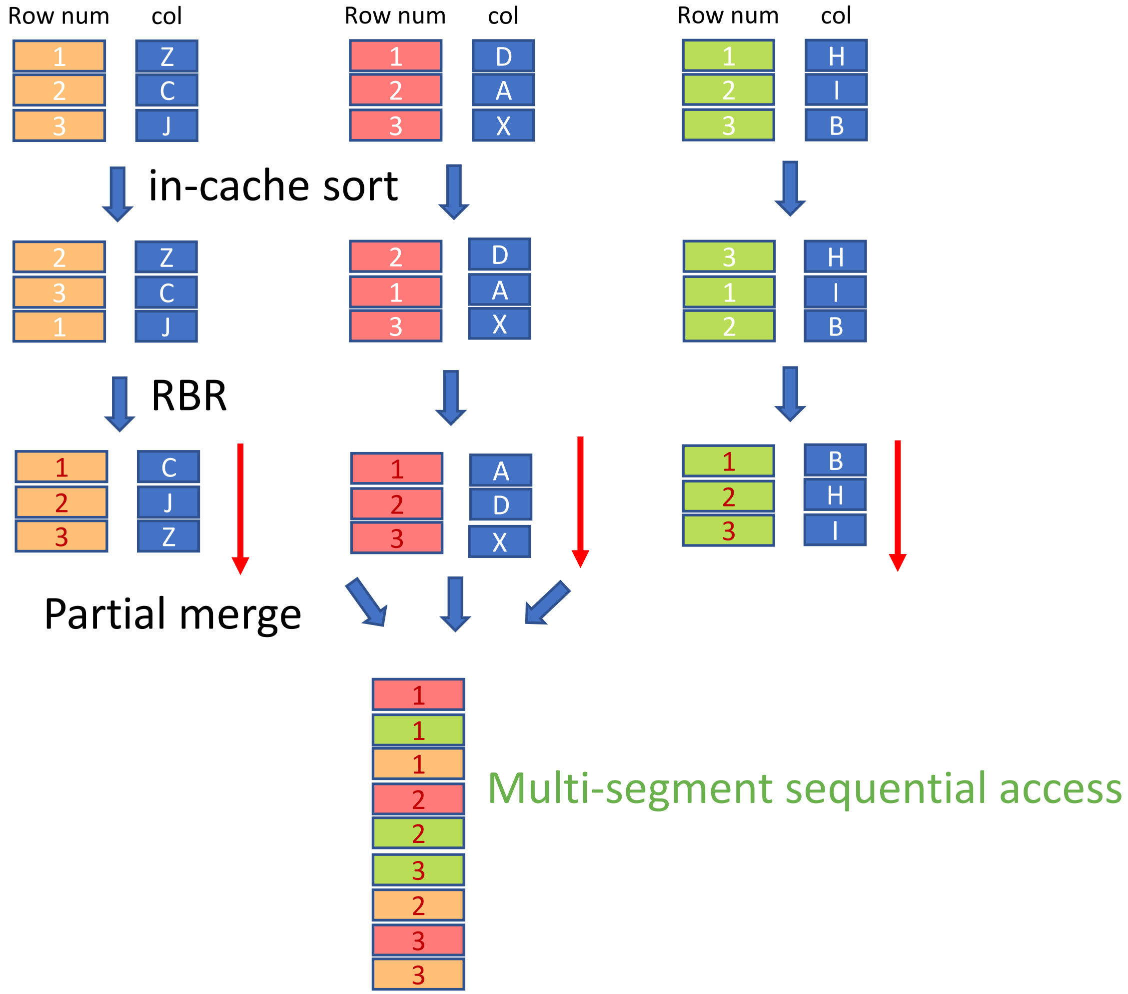


INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

acm In-Cooperation sighpc

Record Batch Rearrangement in Sort





Random memory accesses are greatly reduced

Operator-aware batch length

Existing method:

Keeping the whole batch in cache

$$BL_{rb} = \frac{LLC\ Size}{\sum_i rb.Vec[i].FieldSize}$$

Observation:

Only part of the vectors are used at each step during the computation of complex-op

Only need to guarantee each step's vectors cache-residential

$$BL_{rb} = \frac{LLC\ Size}{4 + Max_j(\sum_i rb.KeyVec[i].FieldSize, rb.Vec[j].FieldSize)} \quad (2)$$

Aggressive batch length

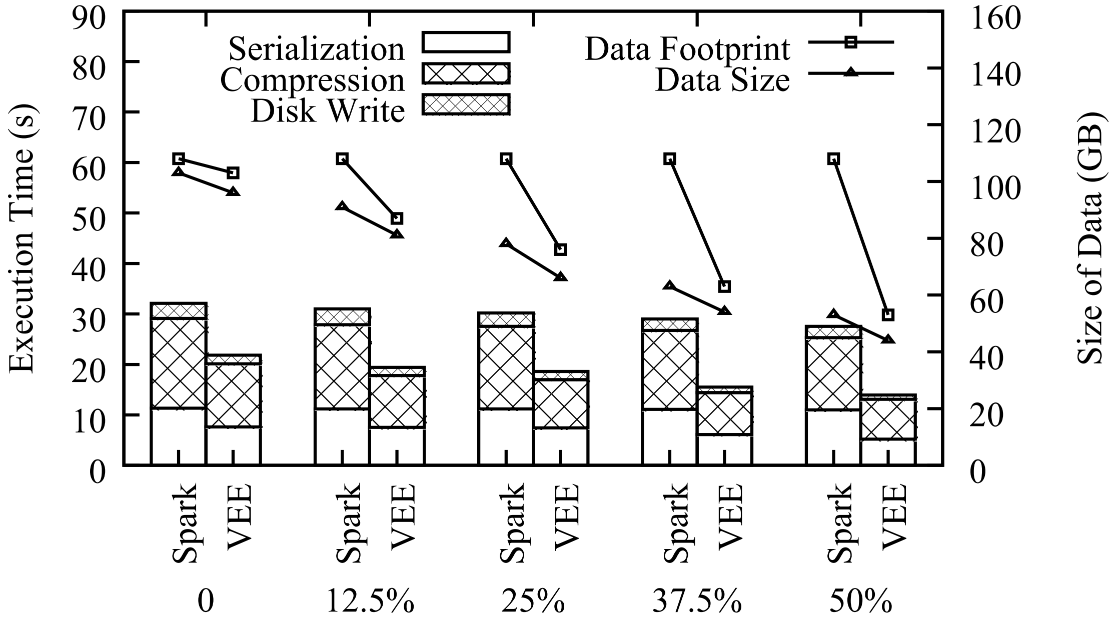
Example: Sort

- Step 1: in-cache sort
  - Row number vec. and
  - Key vecs
- Step 2: partial merge
  - Row number vec. and
  - one vec.

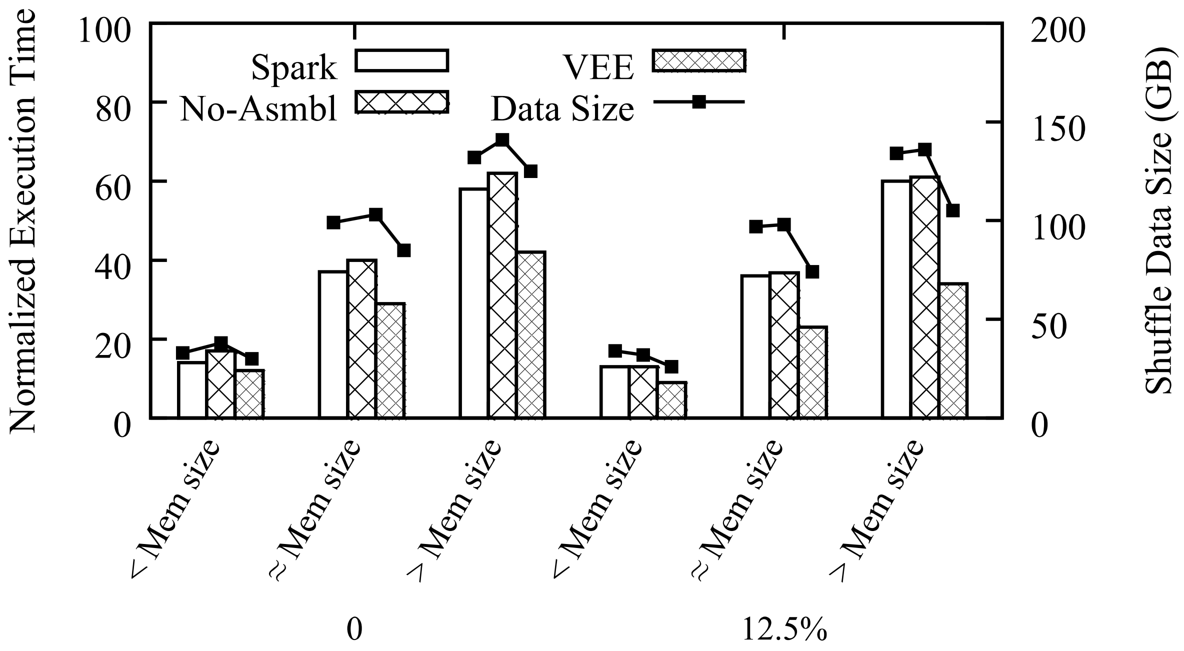
Experiment Setup

- Workload: TPC-H all 22 queries
- Configuration
  - Four-machine Spark cluster
  - Two Intel Xeon E5645 processors, 32GB of memory and eight 1TB HDDs
  - Each processor has 6 cores, 12 hyper threads and 12MB LLC
  - Linux CentOS 7.3 with kernel 3.10.0
- TPC-H 1TB data

Shuffle performance



Improves execution time by up to 1x and 63.9% on average than Spark



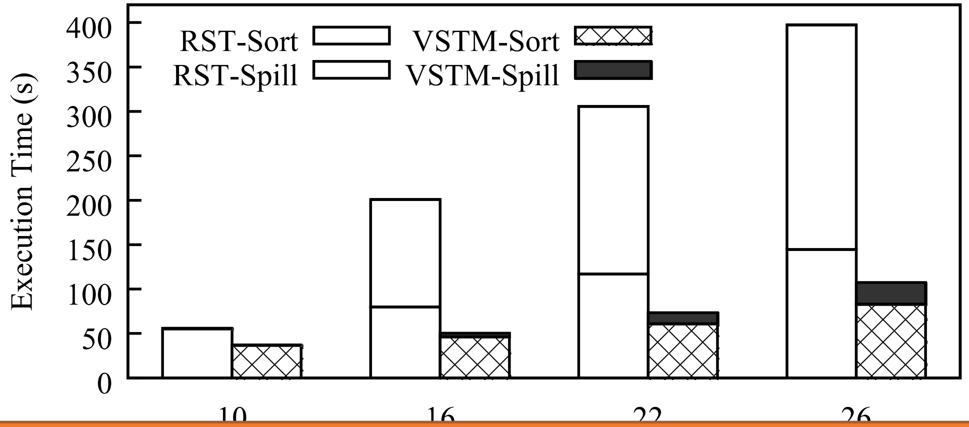
VEE greatly reduces the data footprint by up to 51.8% and 29.6% on average compared to Spark due to SAA

Sort performance

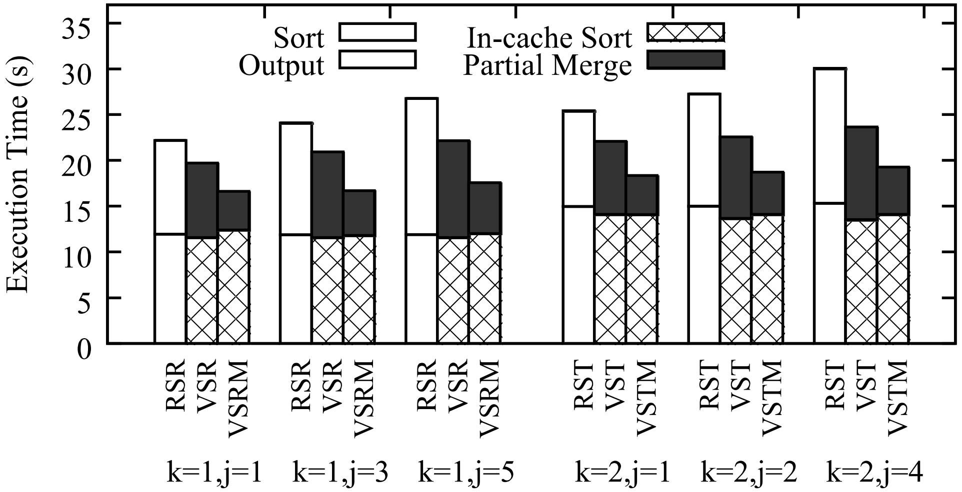
Select SC1,⋯ ,SCk, OC1,⋯ ,OCj

from range(start = 0, end = num, p = 24)

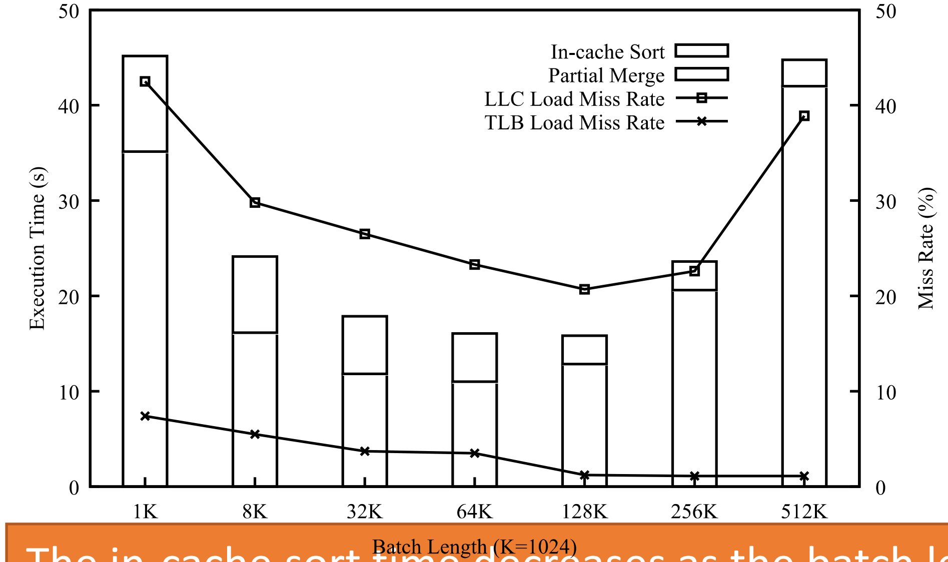
order by SC1,⋯ ,SCm;



VSTM increase much less (by 37%, 96% and 1.9x respectively), and spilling only takes 16% of total time on average, benefiting from compact batch layout.

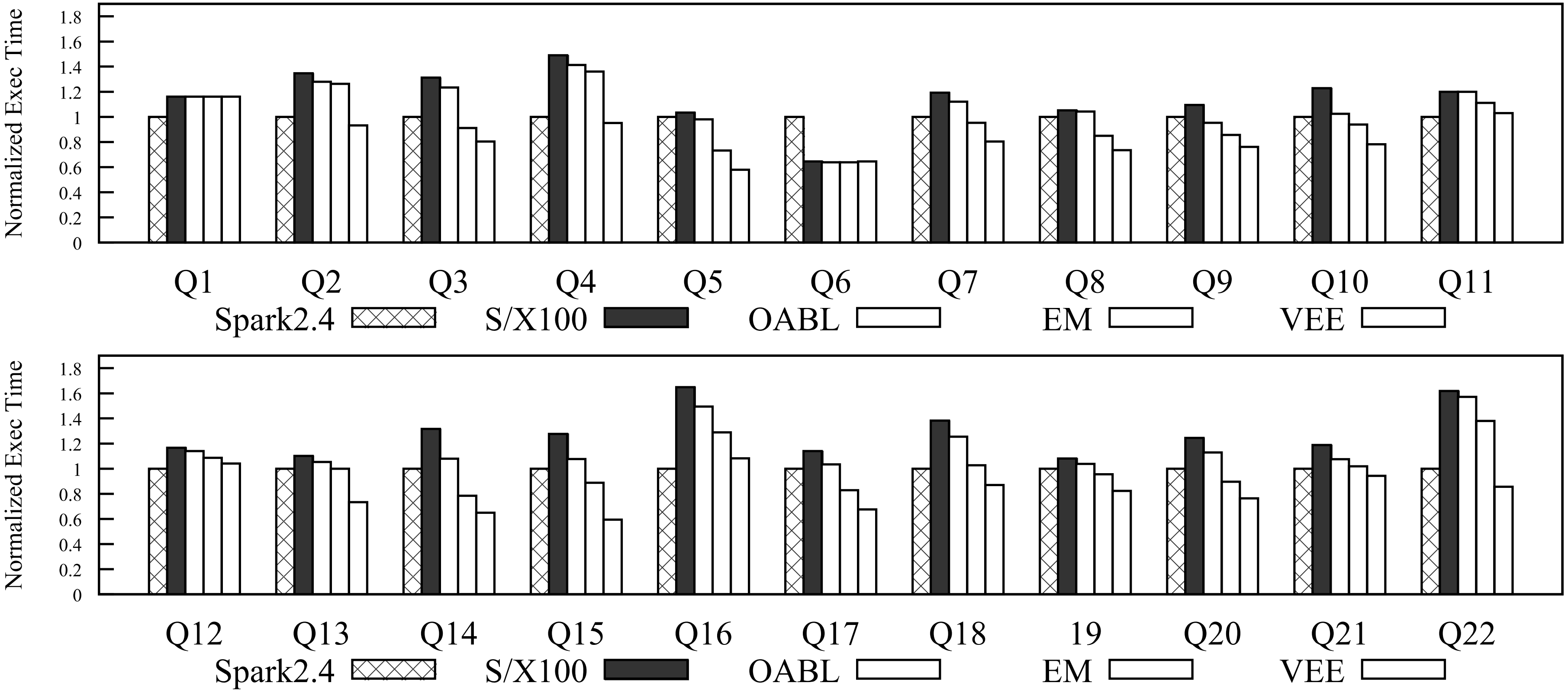


Partial merges with rearrangement save 47.8% time on average compared to their counterparts without rearrangement.



The in-cache sort time decreases as the batch length grows until the batch length reaches 128 K records

Overall TPC-H performance



The performance speedup of VEE against Spark is up to 72.7% and 25.0% on average

Thanks Q&A

Using Vectorized Execution to Boost SQL Query Performance on Spark

Yijie Shen