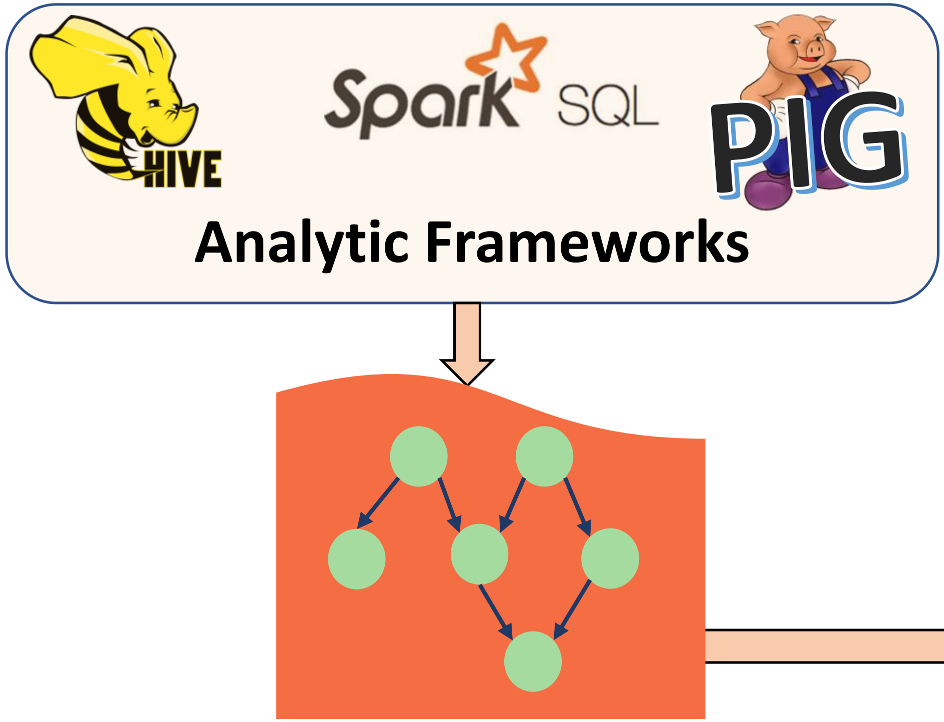
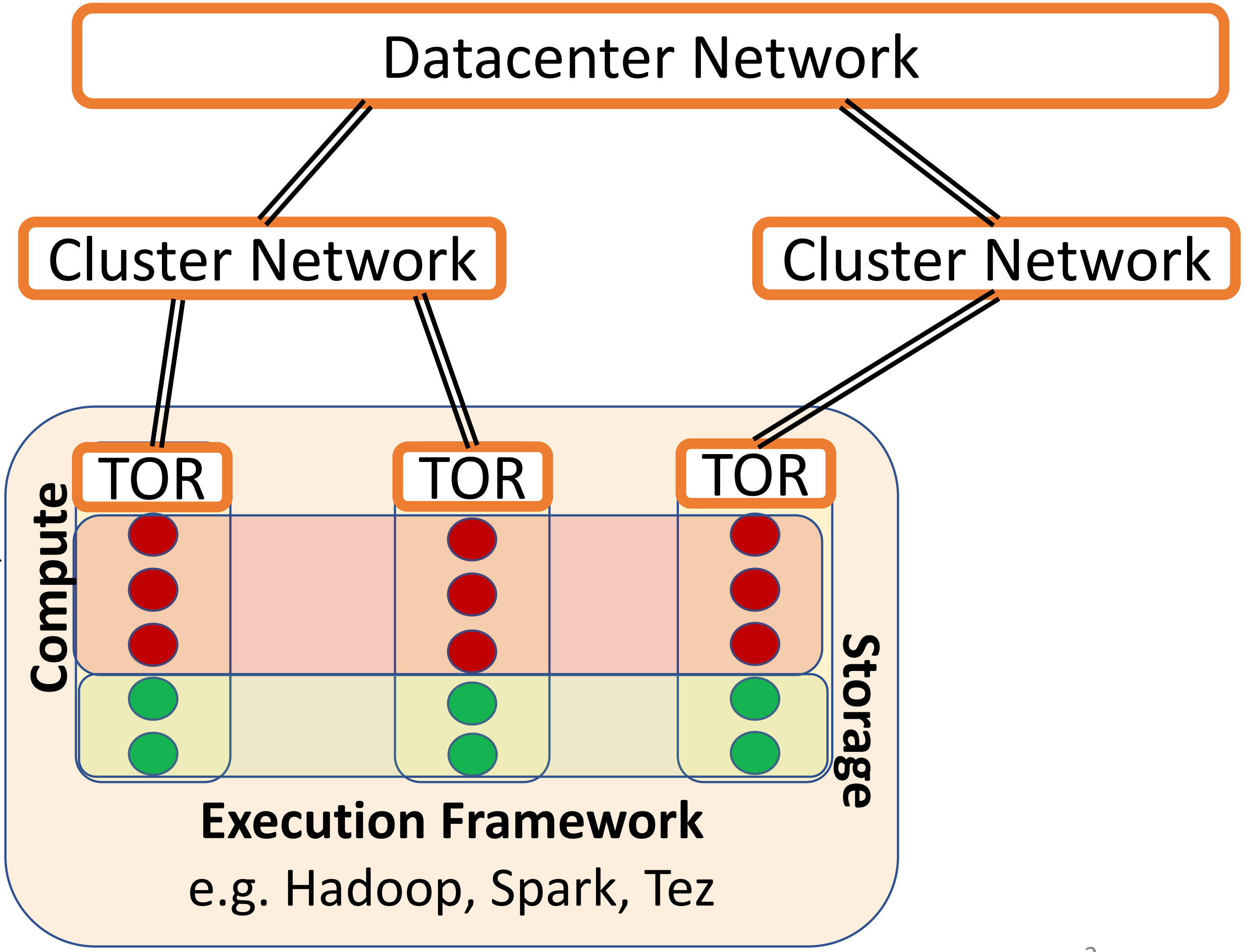
Caching in the multiverse

Mania Abdi[¥], Amin Mosayyebzadeh[ϒ], Mohammad Hossein Hajkazemi[¥], Ata Turk[ǂ], Orran Krieger[ϒ], Peter Desnoyers[¥]

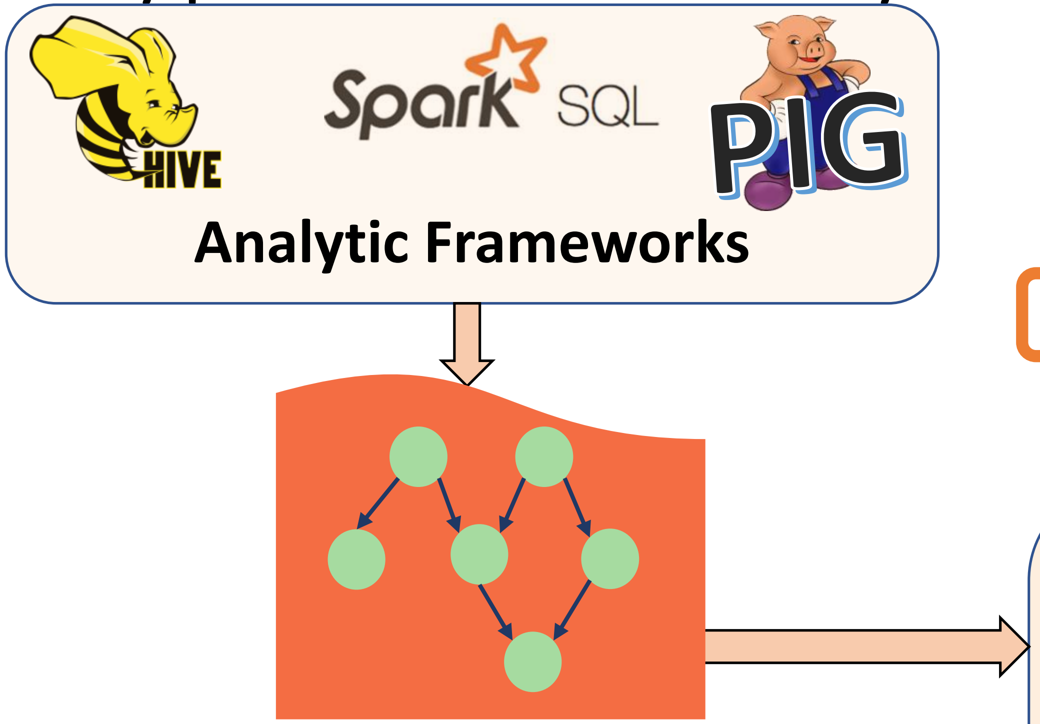
Northeastern University[¥], State Street[ǂ], Boston University[ϒ]

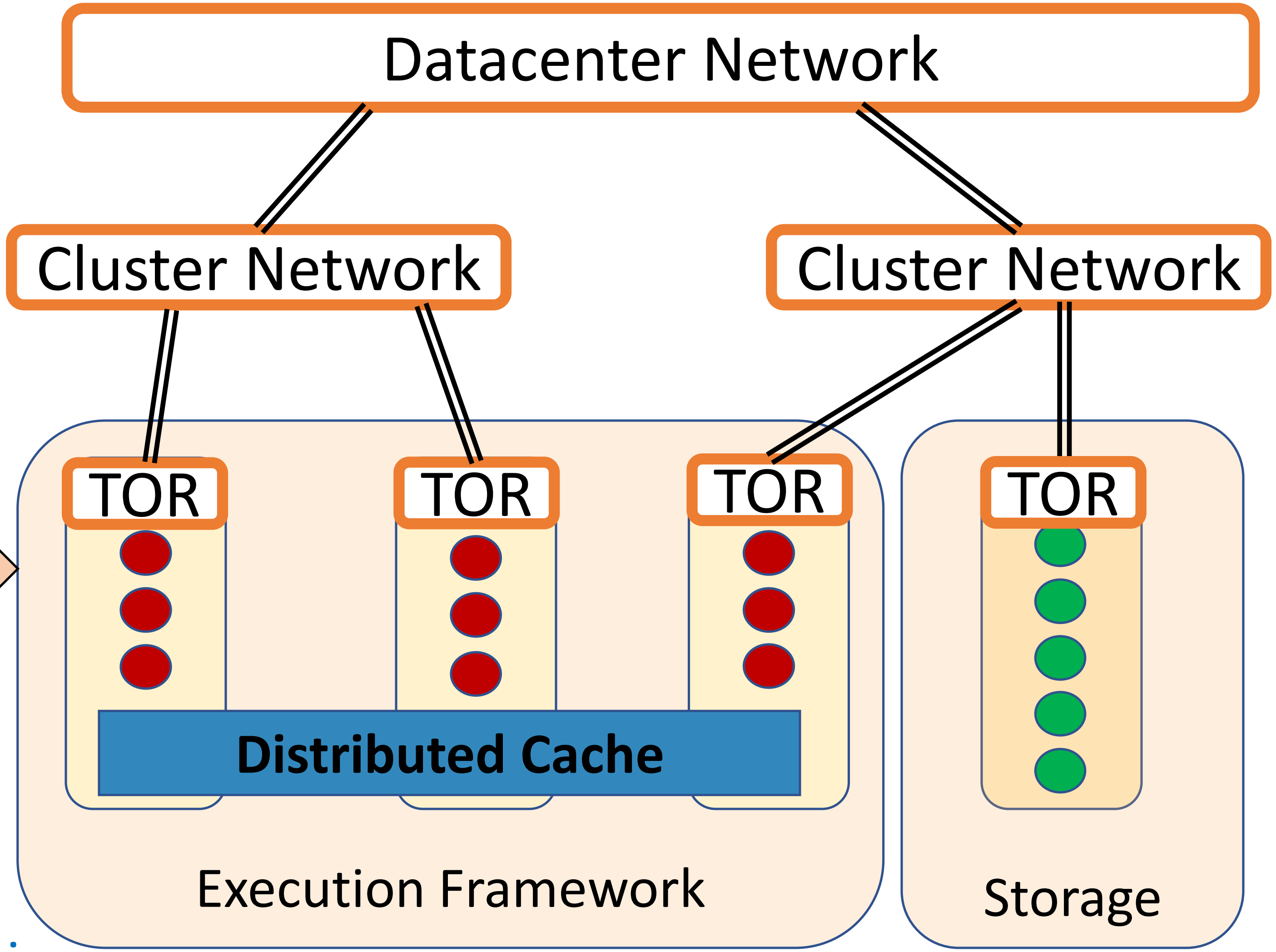
Typical Data analytic Cluster





Typical Data analytic Cluster



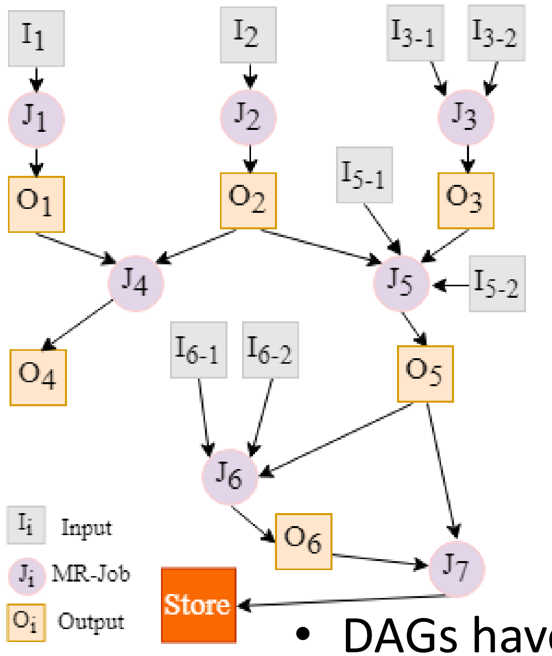


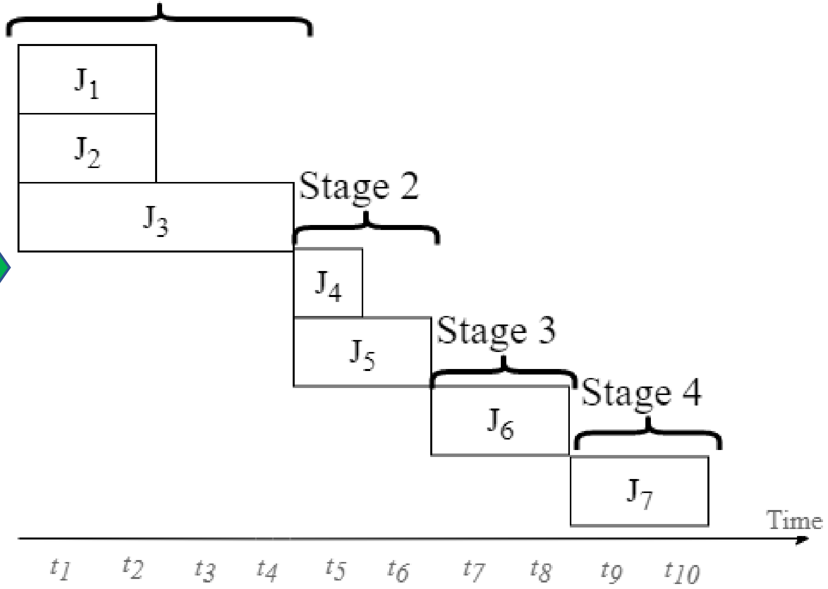
- Storage disaggregation enables:
  - Scalability
  - Flexibility
- Execution frameworks perform better when data is local to cluster
- A distributed cache can help, e.g. Alluxio

Data analytic executions

User queries -> Query optimizer -> Job DAG e.g. PIG

```
J1 { region = load I1
     O1 = filter region

J2 { nation = load I2
     O2 = filter nation

J3 { lineitem = load I3-1
     part = load I2
     fpart = filter part
     O3 = join fpart, lineitem

J4 { O4 = join O1 and O2

J5 { n = load I5-1
     s = load I5-2
     O5 = join s and n

J6 { customer = load I6-1
     supply = load I6-2
     s1 = join supply, O2 and O3
     O6 = join customer and s1

J7 { gr = groupby O5 and O6
     store gr
```





- DAGs have complex and deep structure, e.g. 200 or more nodes for complex jobs. In DAGs:
  - vertices represent analytic jobs
  - Edges represent dependencies between jobs.
- Dependency within a DAG + run time → estimated critical path.

Query #8 from TPC-H benchmark

Taxonomy of caching

Frequent Management Approaches

History based	Informed hint based
LRU Most practice today e.g. Alluxio	**Application hints** e.g. TIP (SOSP95), MC2 (TOCS11)
	Deadline aware e.g. NetCo (SOCC18)
All-or-nothing e.g. Pacman (NSDI12)	**DAG aware** MRD (ICPP18), LRC (InfoComm 17)

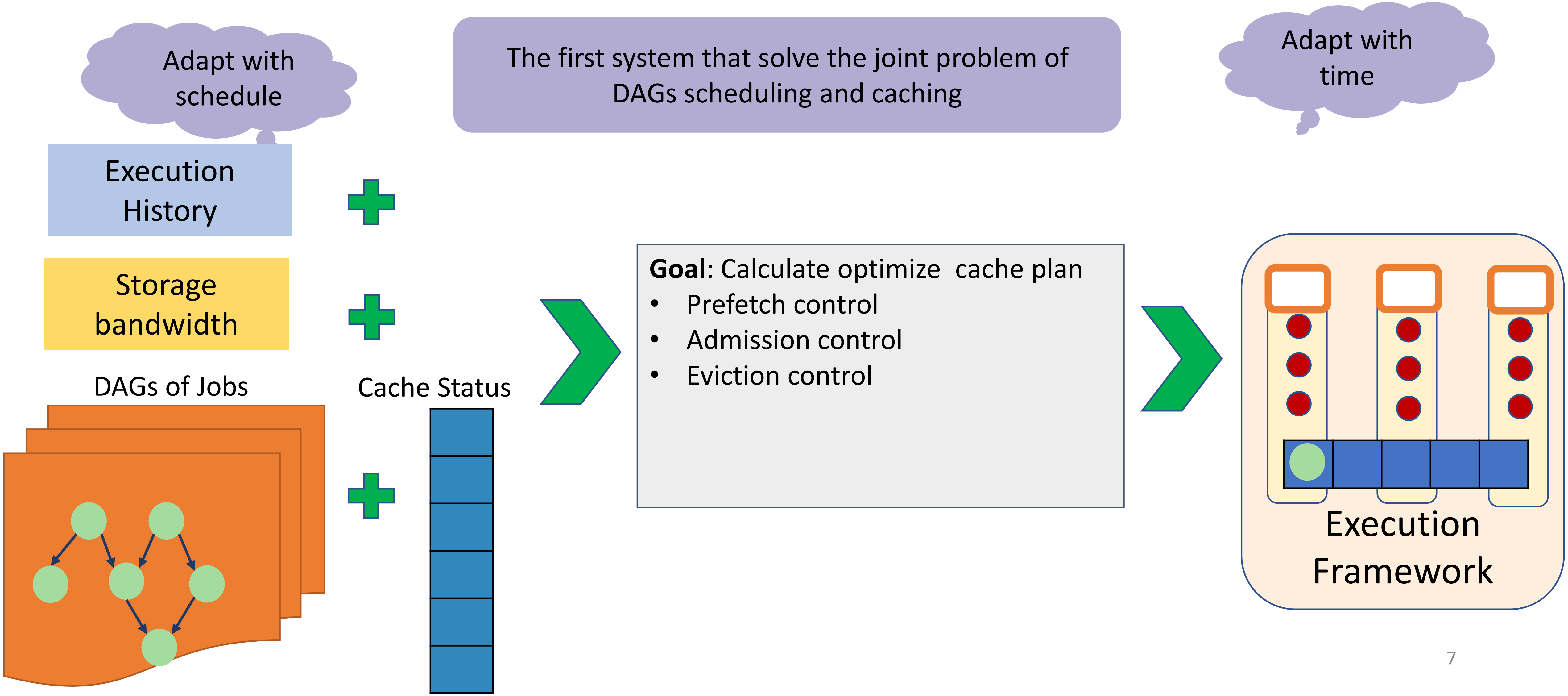
Optimizing Data Analytic caches

Cache performance metric:

Query completion time = $\mathrm{T_{job\ finish}} - \mathrm{T_{job\ submission}}$

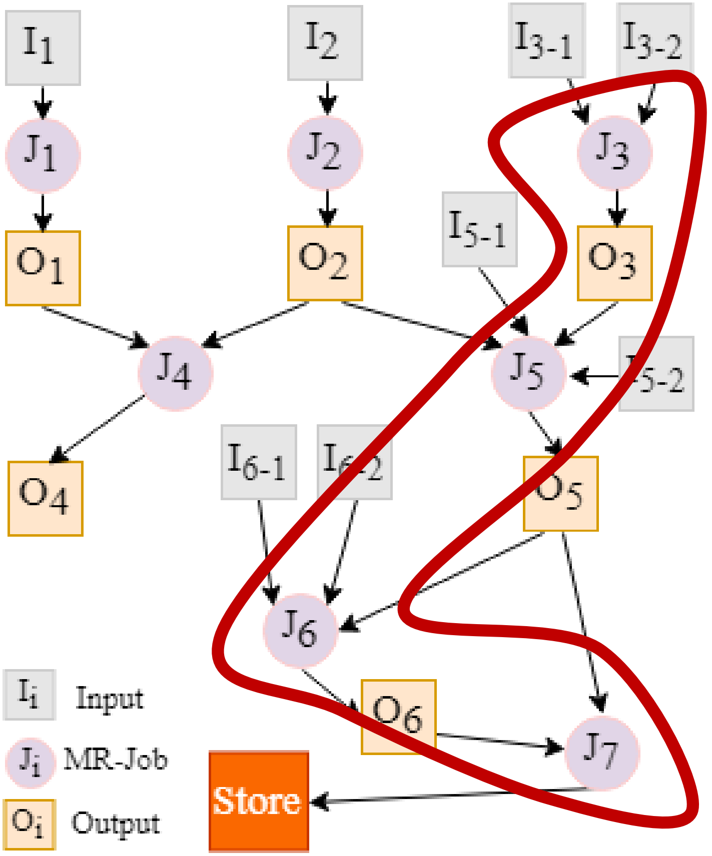
Goal: minimize query completion time

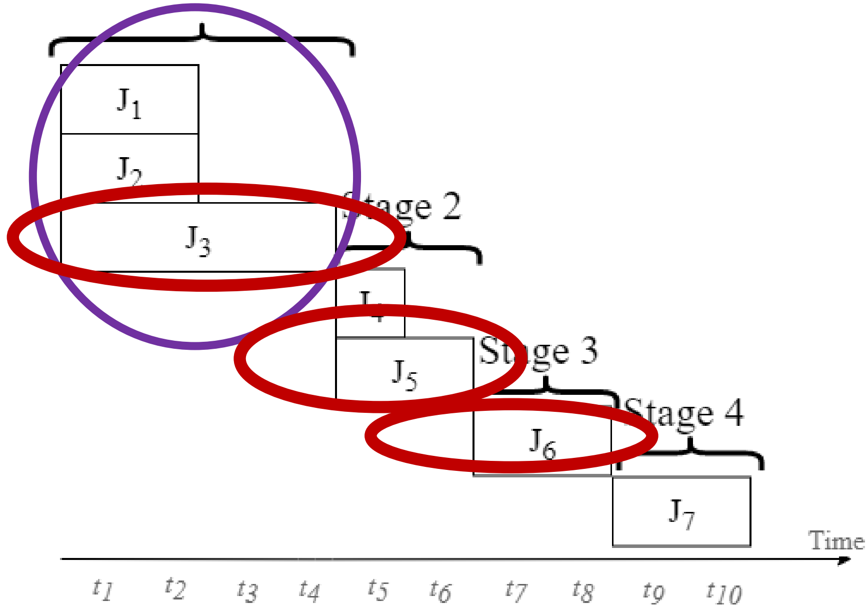
Our approach



How to find cache/prefetch plan?

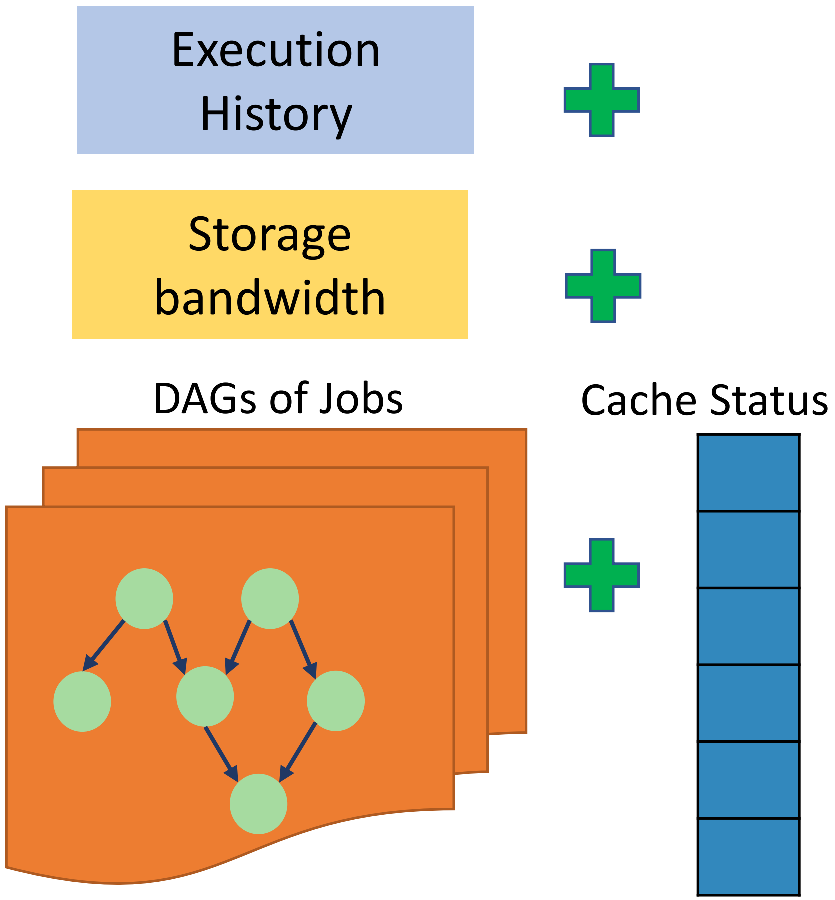
```
J1  region = load I1
    O1 = filter region
J2  nation = load I2
    O2 = filter nation
J3  lineitem = load I3-1
    part = load I2
    fpart = filter part
    O3 = join fpart, lineitem
J4  O4 = join O1 and O2
J5  n = load I5-1
    s = load I5-2
    O5 = join s and n
J6  customer = load I6-1
    supply = load I6-2
    s1 = join supply, O2 and O3
    O6 = join customer and s1
J7  gr = groupby O5 and O6
    store gr
```





How to find cache/prefetch plan?

- Predict job run time with and without prefetching.
- Find critical path based on dependencies and history of execution.
- Incorporate dependencies, bandwidth to storage, current cache status to choose:
  - Dataset to be cached
  - Dataset to be prefetched
  - Dataset to be evicted



Experimental environment

Analytic framework: Pig

Execution framework: MapReduce

- 4 bare metal nodes:
  - 60GB RAM
  - 16 CPU
  - 10 Gbps Ethernet

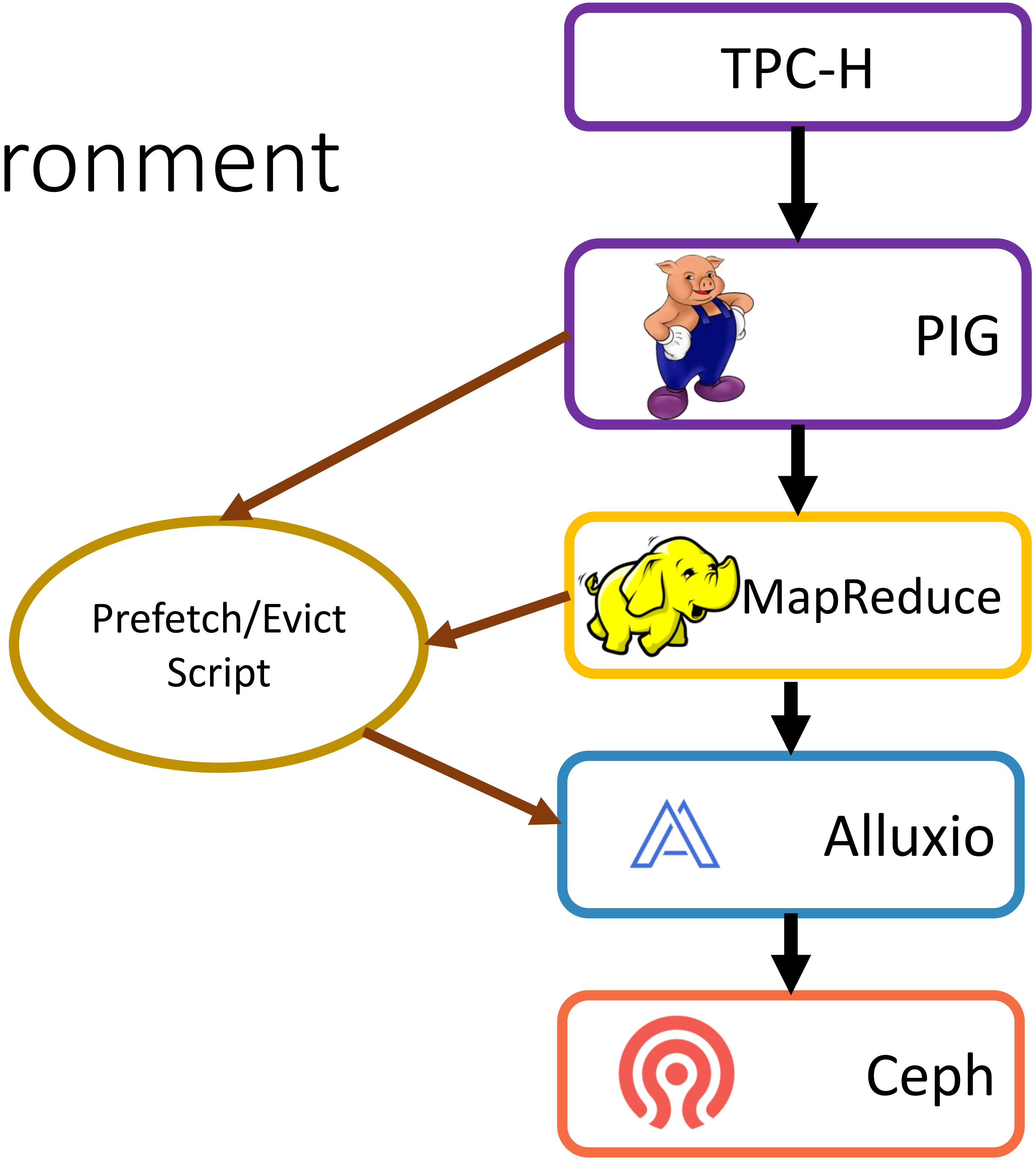
Distributed cache: Alluxio

- 4 bare metal nodes:
  - Collocate with Hadoop nodes.
  - 6BG per cache = 24GB cache

Remote storage: Ceph

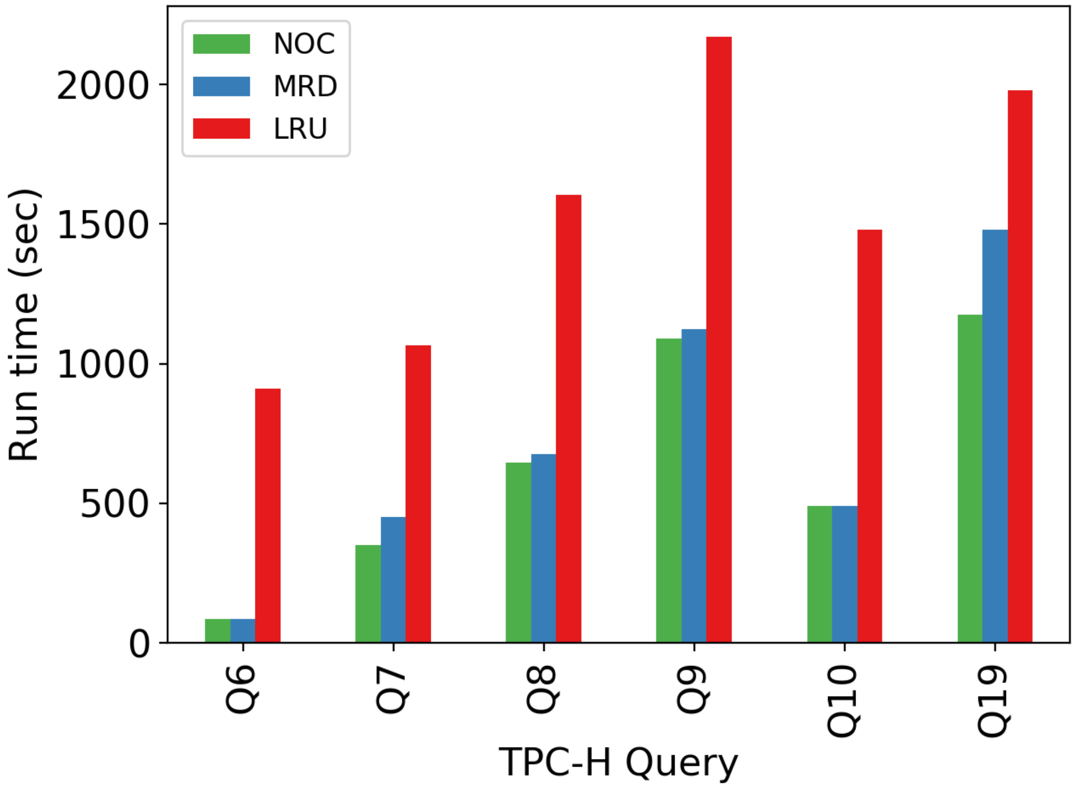
Benchmark: TPC-H queries

- 30 GB dataset size

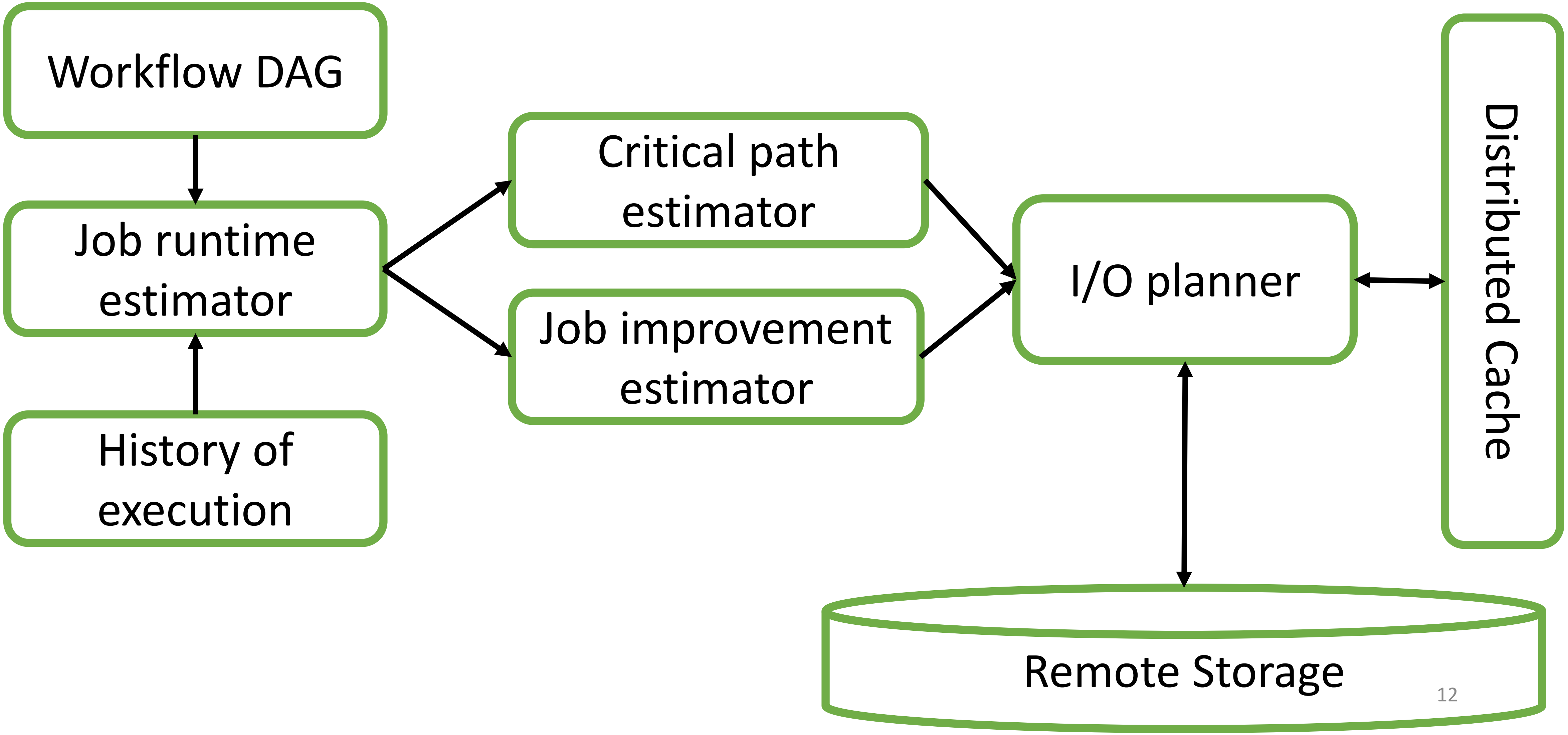


Results

- 2X improvement over LRU.
- Up to 22% improvement over MRD



Current work: implementation



Discussion / Challenges

- Benchmark:
    - New benchmark to evaluate our approach.
- Multi-query execution:
    - Approaches: two step cache management
        - Create plan for a single query
        - Create plan for multiple queries.
    - Competition queries with contradicting requests
        - Initial approaches: prioritize nearest future access.
    - Bandwidth allocation for multiple queries
        - Initial approaches: prioritize shortest job first.
- Where to prefetch?

Conclusion

Goal: minimize end-to-end latency of query execution

Approach: scheduling aware cache management policy

Key insight: incorporate execution history and current cache status to optimize the critical path through caching and prefetching.

Results: 2X improvement over LRU