Scheduling with Communication Delay in Near-Linear Time

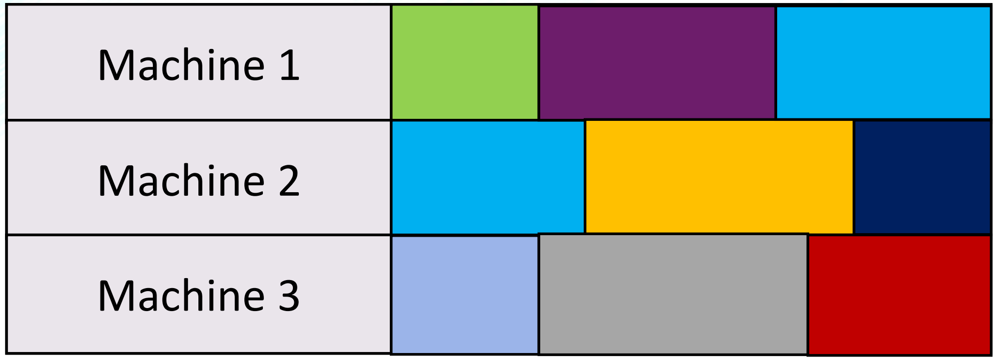
Quanquan C. Liu
Northwestern University


Manish Purohit
Google Research


Zoya Svitkina
Google Research

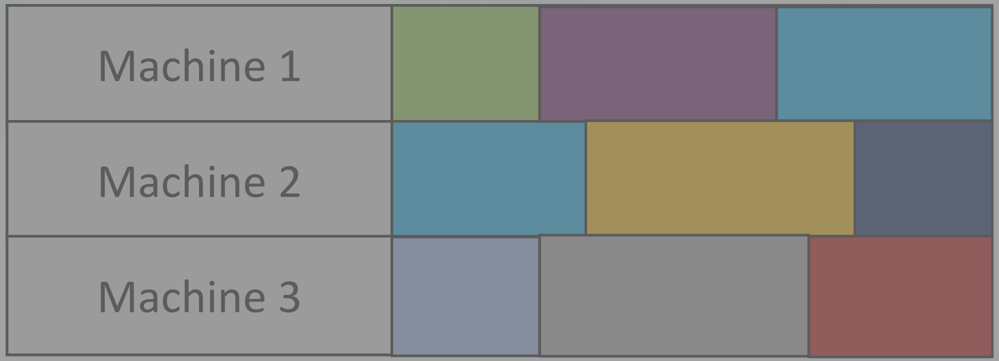

Erik Vee
Google Research


Joshua R. Wang
Google Research



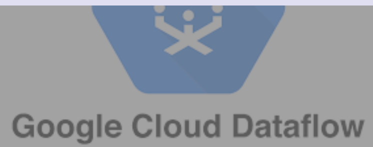
Scheduling is a classical problem in theory and in practice

- Cluster data processing management (Google Cloud Dataflow, Spark, Hadoop, Mesos...etc.)
- Machine learning (scheduling training, e.g., Tensorflow…etc.)



Scheduling is a classical problem in theory and in practice

Efficient Scheduling is Important in Large Data Centers



- Cluster data processing management (Google Cloud Dataflow, Spark, Hadoop, Mesos...etc.)
- Machine learning (scheduling training, e.g., Tensorflow...etc.)

Research Question and Goal

How computationally expensive is it to perform approximately-optimal scheduling?

ρ

ρ

Even simple formulations of exact scheduling are **NP Hard**

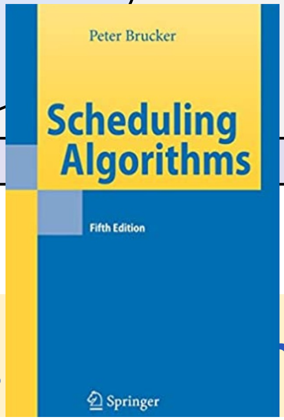
Communication Delay

Rich body of literature for designing good **approximation algorithms** for multiprocessor scheduling

Job

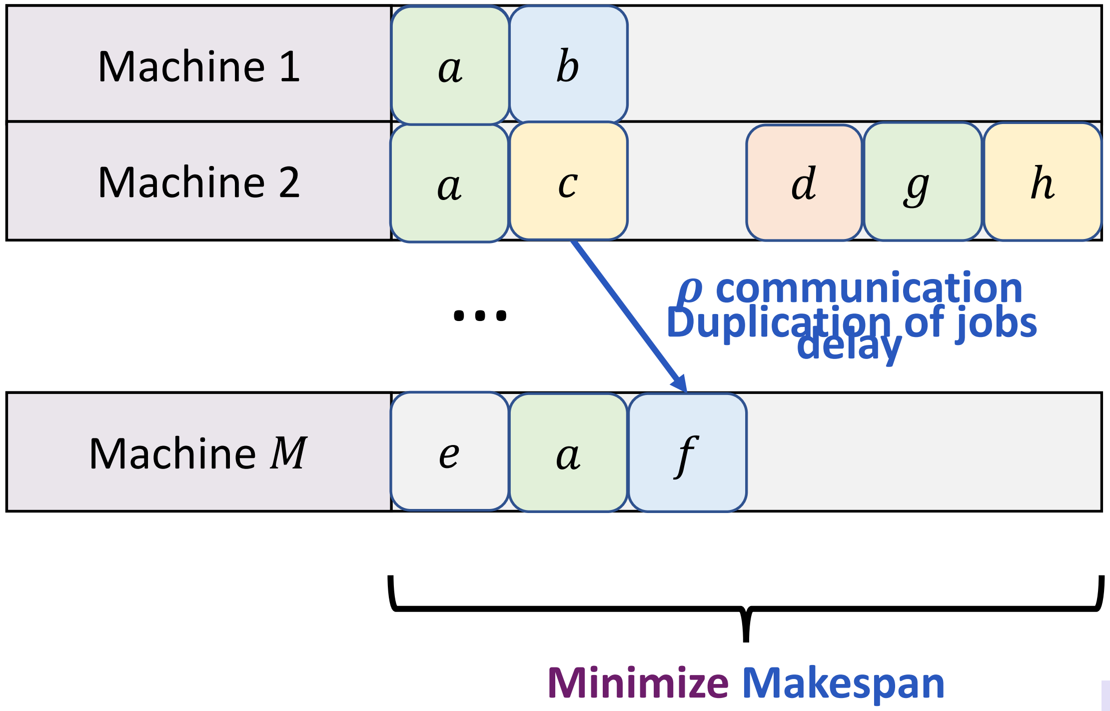
Open: efficient algorithms with good approximations

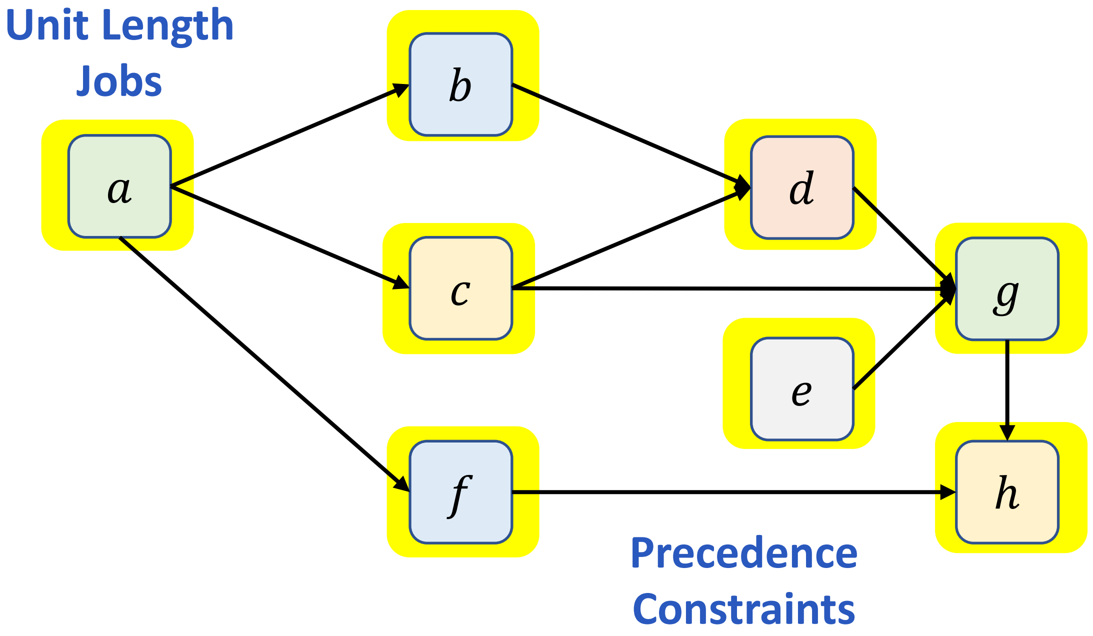
Job Duplication



Goal: Minimiz ne while computing good approx. schedules

Scheduling with Communication Delay





Runtime as Close to Linear as Possible

Previous Results

- **With duplication:**
  - Unit jobs, fixed uniform communication delay, identical machines, duplication:

Previous Results

- **With duplication:**
  - Unit jobs, fixed uniform communication delay, identical machines, duplication:
    - $O(\log \rho \,/\, \log\log \rho)$-approximation [Lepere-Rapine, STACS '02], assuming schedule has length at least ρ

Previous Results

- **With duplication:**
  - Unit jobs, fixed uniform communication delay, identical machines, duplication:
    - $O(\log \rho / \log \log \rho)$-approximation [Lepere-Rapine, STACS '02], assuming schedule has length at least ρ
    - $\Omega(n \ln M + n\rho^2 + m\rho)$ runtime

Previous Results

- **With duplication:**
  - Unit jobs, fixed uniform communication delay, identical machines, duplication:
    - $O(\log \rho / \log \log \rho)$-approximation [Lepere-Rapine, STACS '02], assuming schedule has length at least ρ
    - $\Omega(n \ln M + n\rho^2 + m\rho)$ runtime

Our Result: $O(\log \rho / \log \log \rho)$-approximation, $O(n \ln M + m \ln^3 n \ln \rho / \ln \ln \rho)$ runtime, whp, assuming schedule has length at least ρ

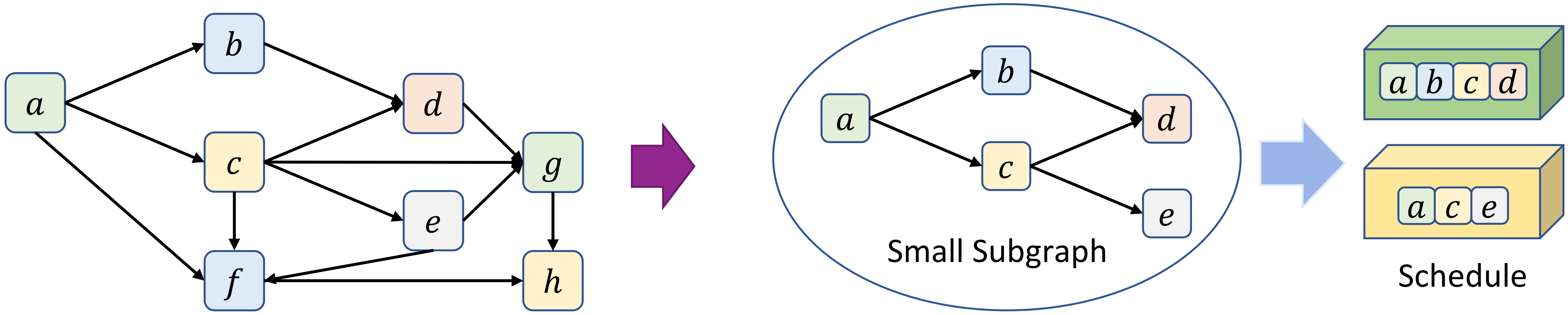
Lepere-Rapine Algorithm

Lepere-Rapine Algorithm (+Modifications)

- Small subgraph: A maximal subgraph of the input where each vertex has at most 2ρ ancestors

Lepere-Rapine Algorithm (+Modifications)

- Small subgraph: A maximal subgraph of the input where each vertex has at most 2ρ ancestors
- Phases:
  - Schedule a small subgraph



Lepere-Rapine Algorithm (+Modifications)

- Small subgraph: A maximal subgraph of the input where each vertex has at most 2ρ ancestors
- Phases:
  - Schedule a small subgraph
    - List schedule subsets of jobs in batches

Lepere-Rapine Algorithm (+Modifications)

- Small subgraph: A maximal subgraph of the input where each vertex has at most 2ρ ancestors
- Phases:
  - Schedule a small subgraph
    - List schedule subsets of jobs in batches
    - Add a delay of ρ to the schedule after each batch

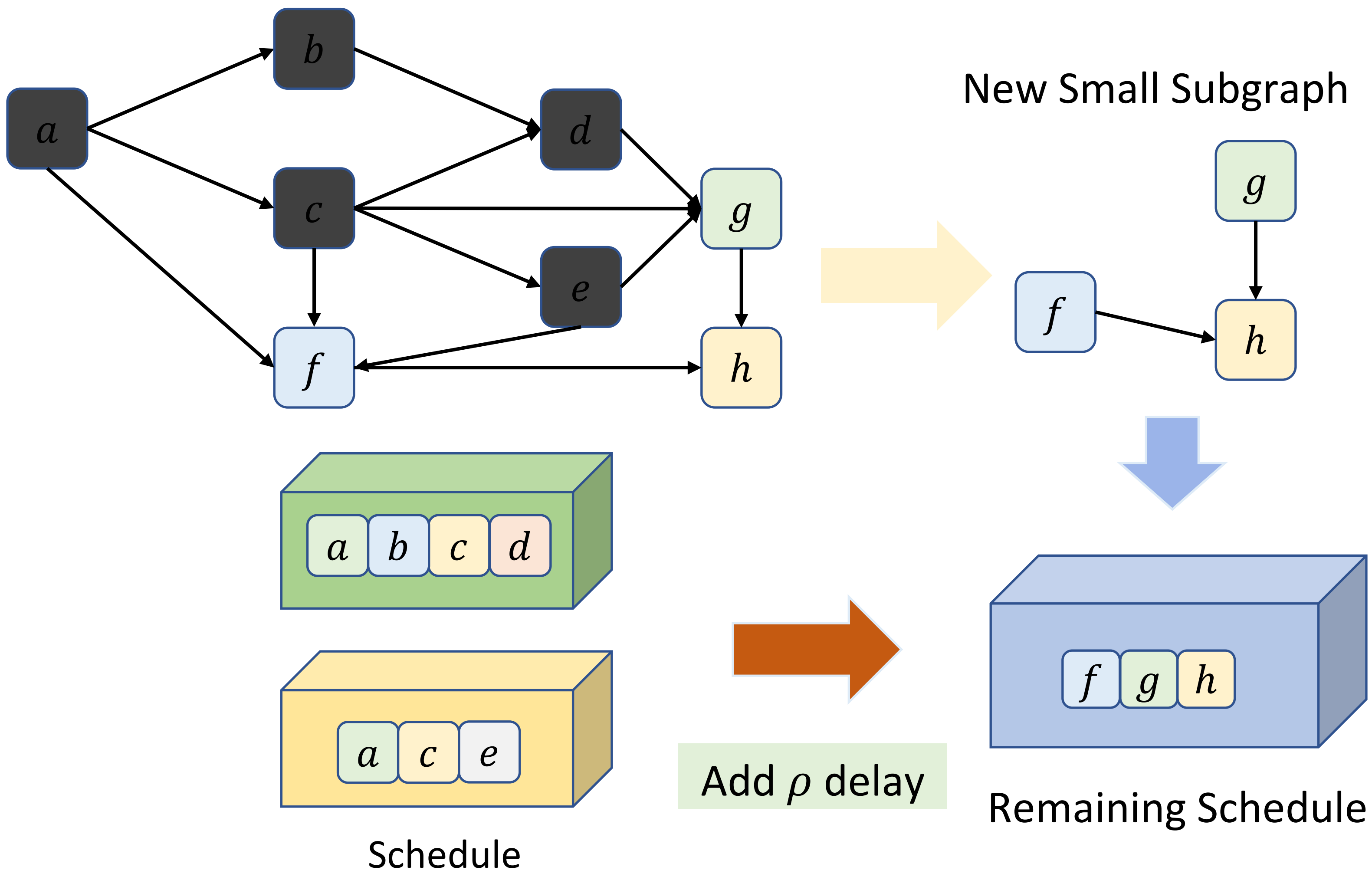
Lepere-Rapine Algorithm (+Modifications)

- Small subgraph: A maximal subgraph of the input where each vertex has at most 2ρ ancestors
- Phases:
  - Schedule a small subgraph
    - List schedule subsets of jobs in batches
    - Add a delay of ρ to the schedule after each batch
    - Remove batch from small subgraph

Lepere-Rapine Algorithm (+Modifications)

- Small subgraph: A maximal subgraph of the input where each vertex has at most 2ρ ancestors
- Phases:
  - Schedule a small subgraph
    - List schedule subsets of jobs in batches
    - Add a delay of ρ to the schedule after each batch
    - Remove batch from small subgraph

Lepere-Rapine Algorithm (+Modifications)



Lepere-Rapine Algorithm (+Modifications)

- Small subgraph: A maximal subgraph of the input where each vertex has at most 2ρ ancestors

$\Omega(n\rho^2 + m\rho)$

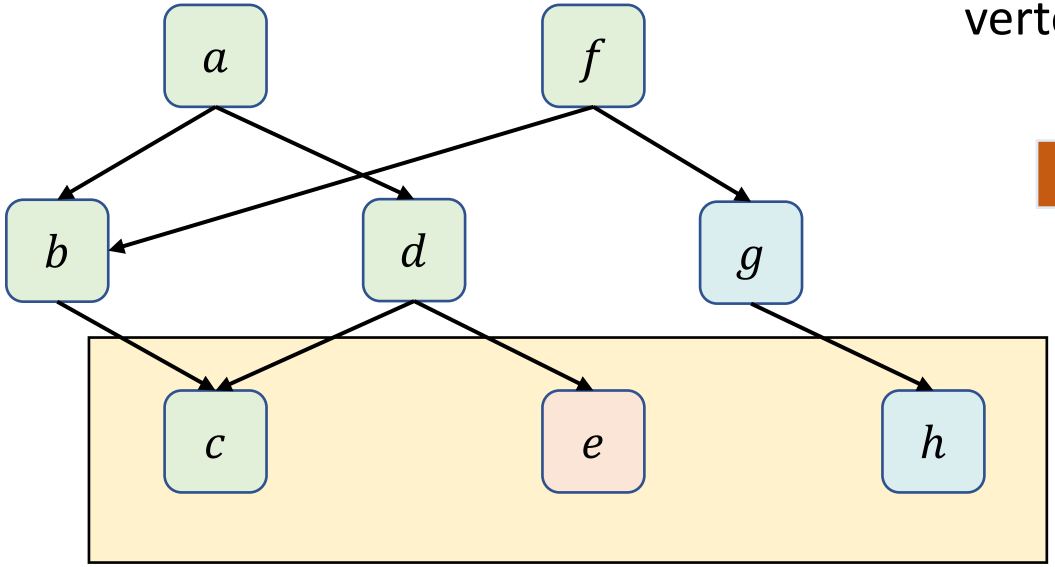
- Phases:
  - Schedule a small subgraph
    - List schedule subsets of jobs in batches
    - Add a delay of ρ to the schedule after each batch
    - Remove batch from small subgraph

Lepere-Rapine Algorithm (+Modifications)

- Small subgraph: A maximal subgraph of the input where each vertex has at most 2ρ ancestors
- Phases:
    - Schedule a small subgraph
        - List schedule subsets of jobs in batches
        - Add a delay of ρ to the schedule after each batch
        - Remove batch from small subgraph

Lepere-Rapine Algorithm (+Modifications)

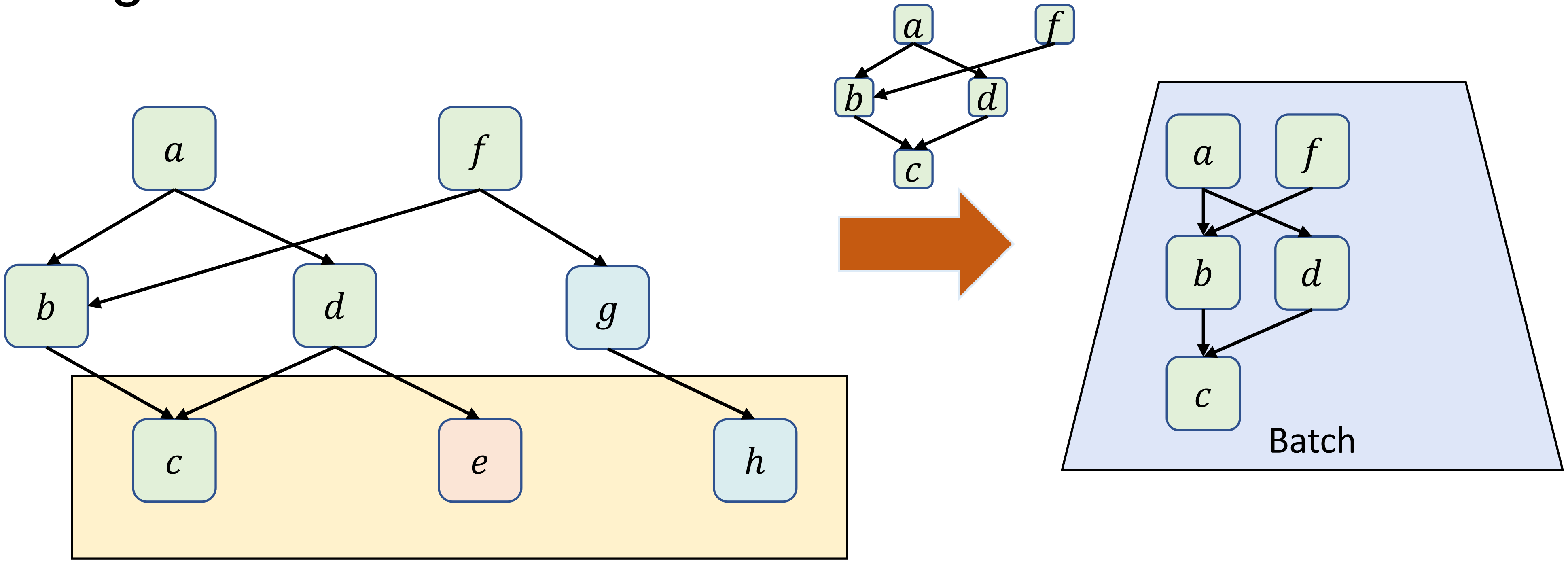
- Finding a batch to schedule



> ½ ancestors + vertex are **not** in batch

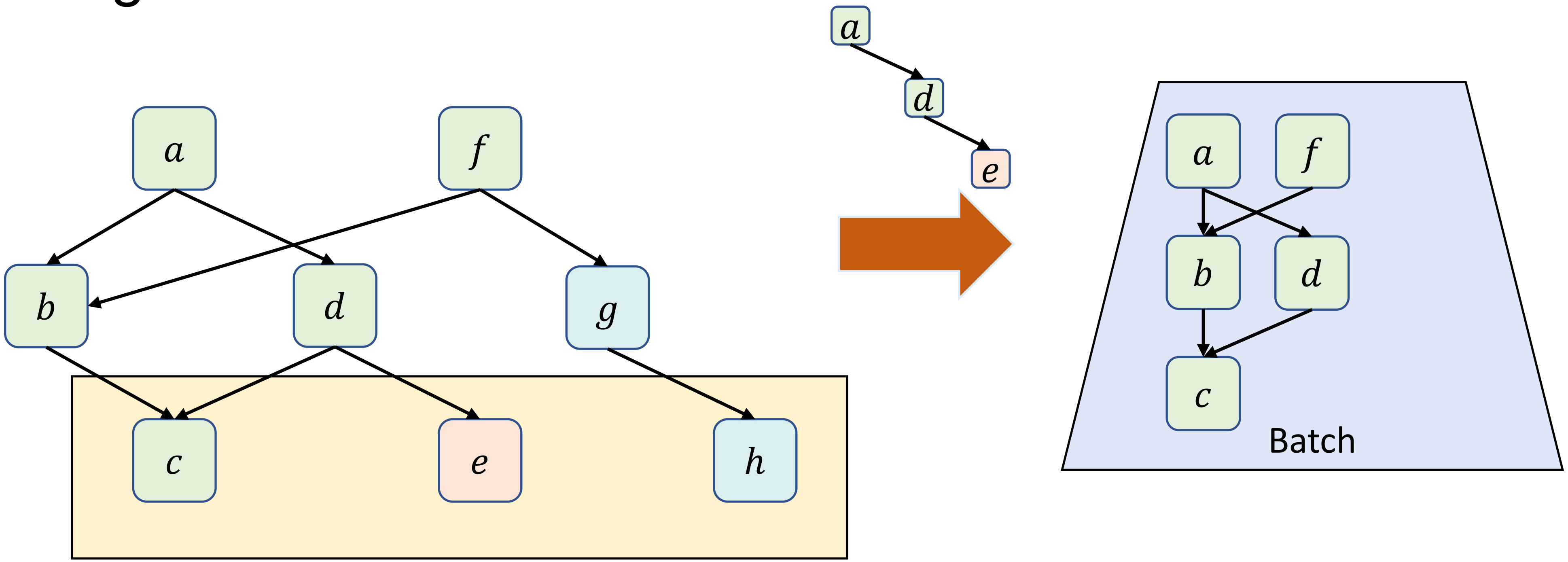
Lepere-Rapine Algorithm (+Modifications)

- Finding a batch to schedule



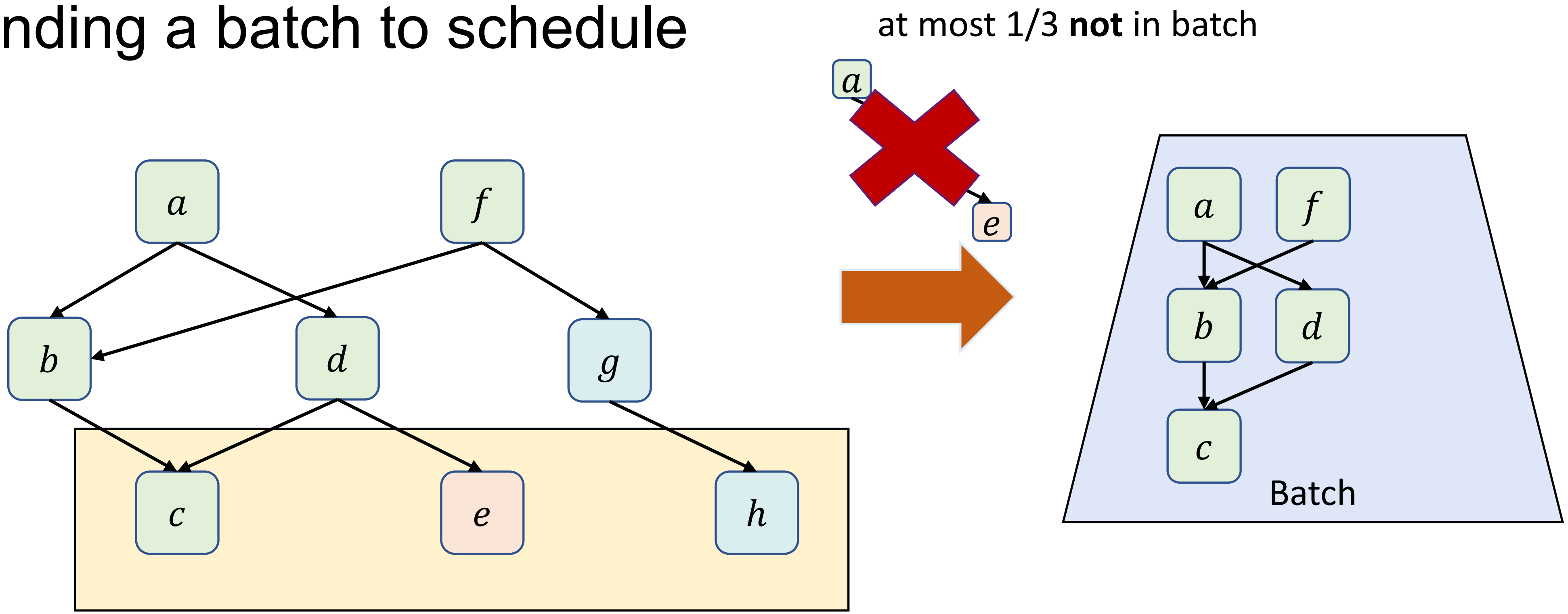
Lepere-Rapine Algorithm (+Modifications)

- Finding a batch to schedule



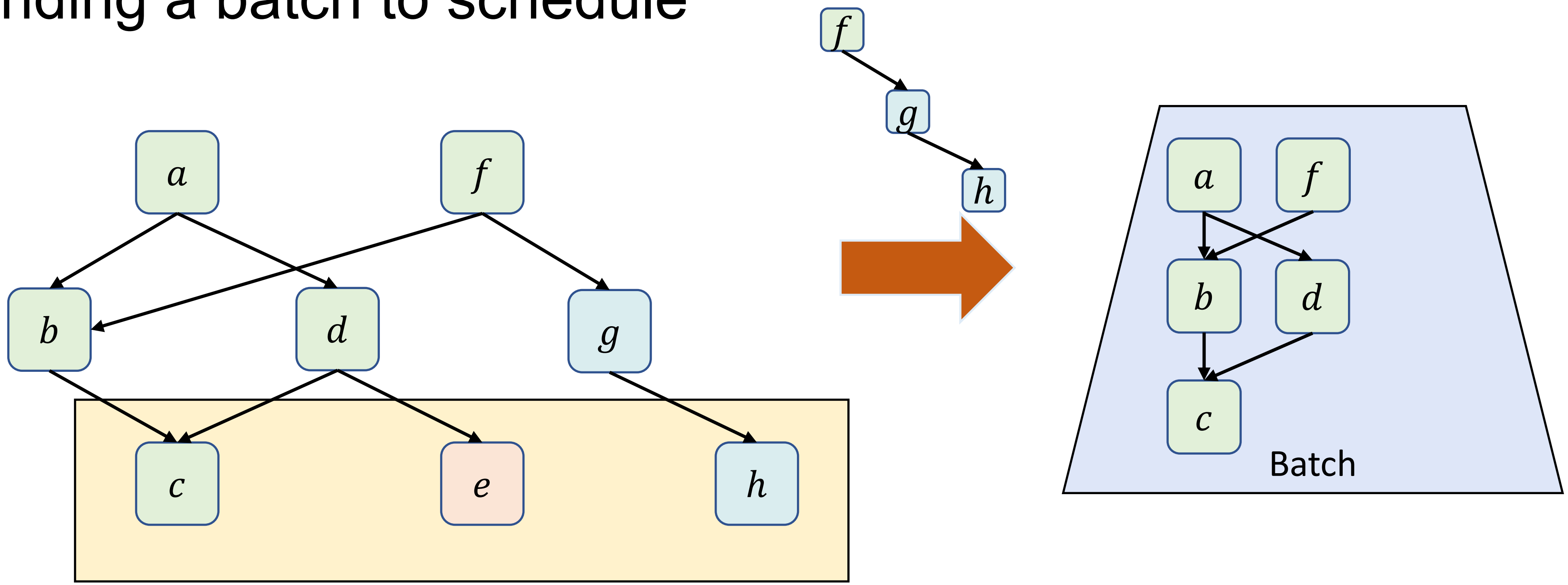
Lepere-Rapine Algorithm (+Modifications)

- Finding a batch to schedule



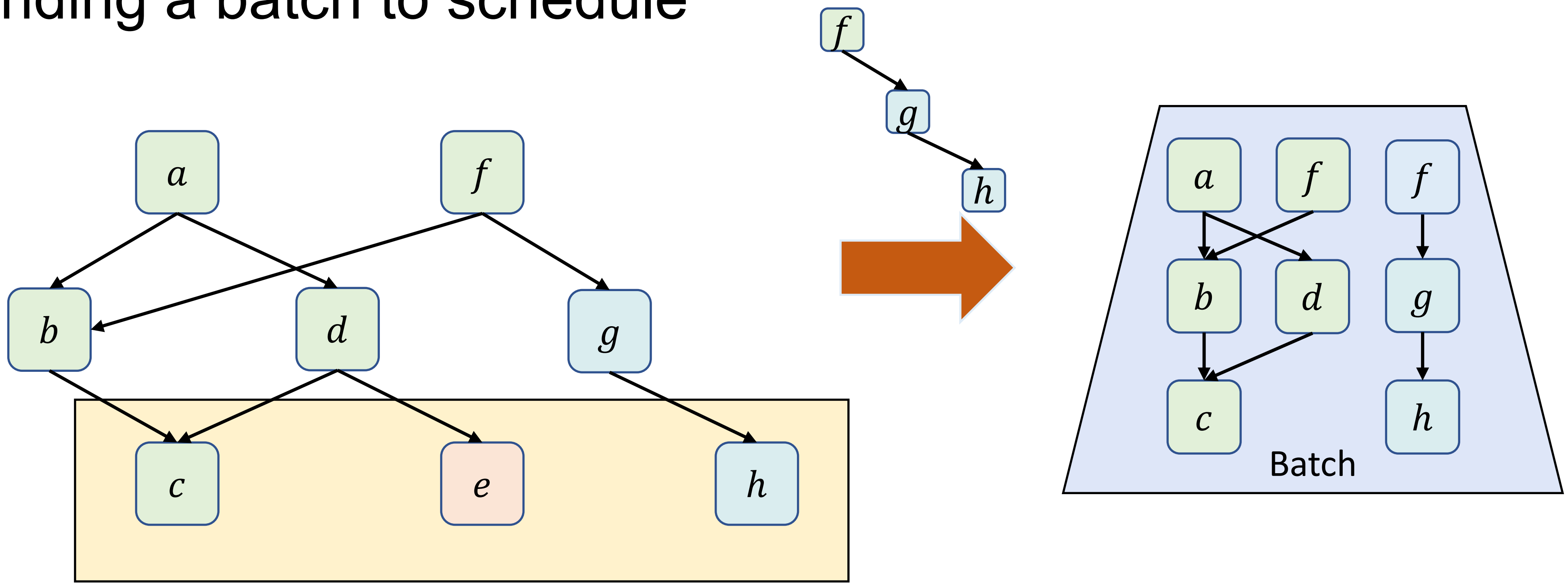
Lepere-Rapine Algorithm (+Modifications)

- Finding a batch to schedule



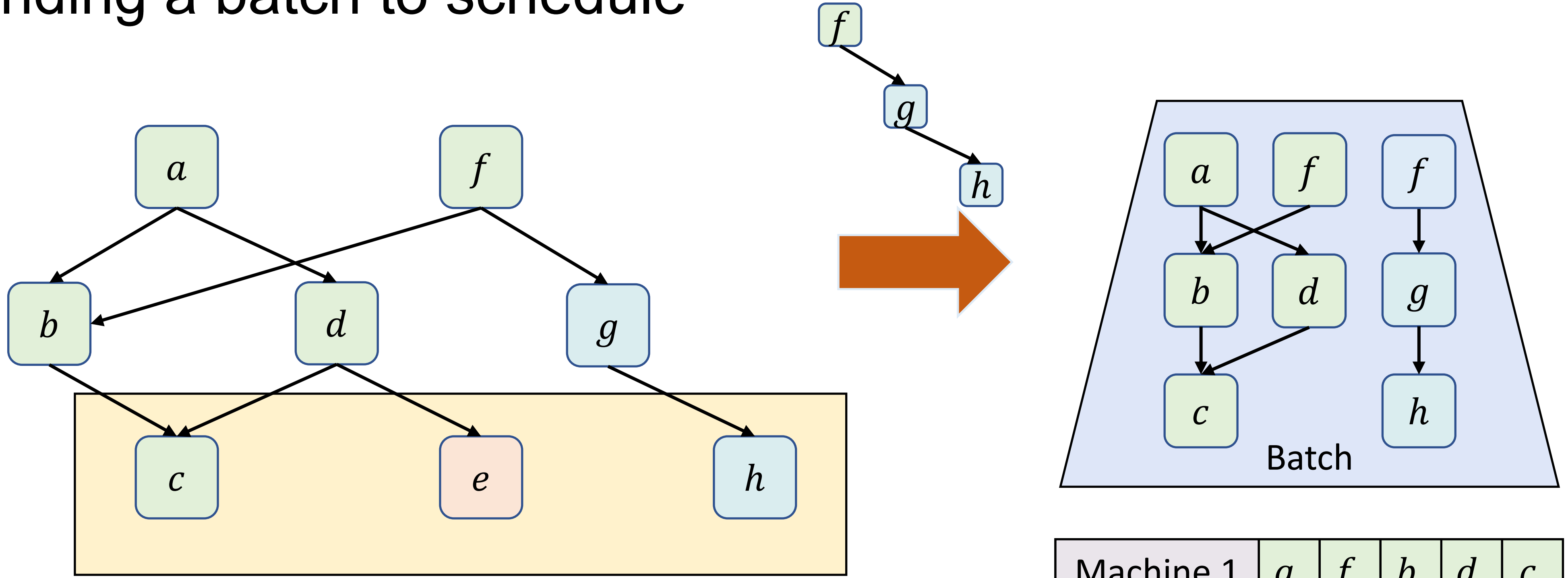
Lepere-Rapine Algorithm (+Modifications)

- Finding a batch to schedule



Lepere-Rapine Algorithm (+Modifications)

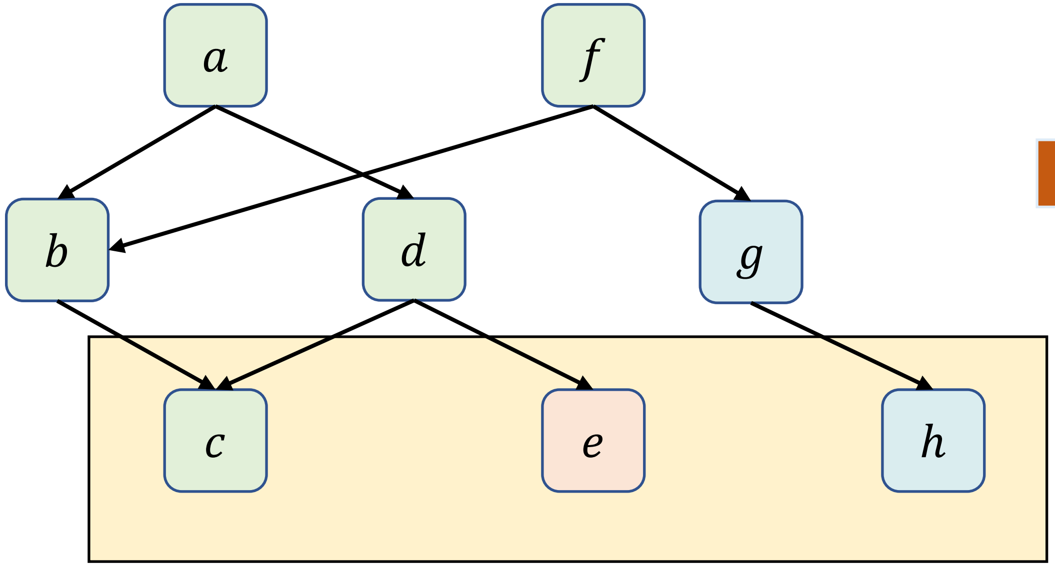
- Finding a batch to schedule

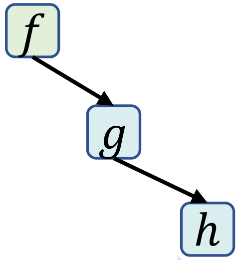


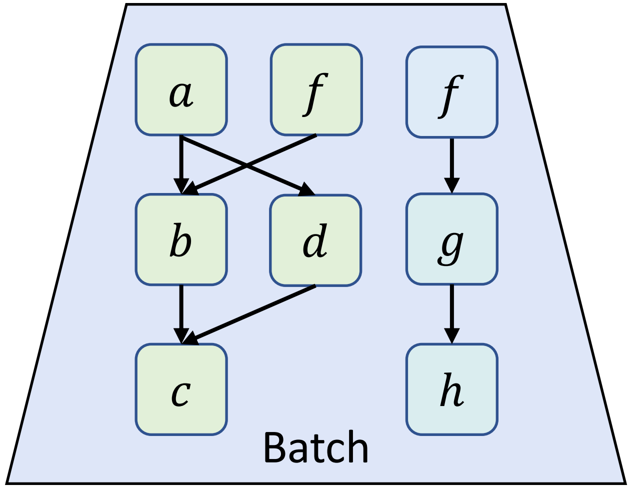
Machine 1	a	f	b	d	c
Machine 2	f	g	h		

Lepere-Rapine Algorithm (+Modifications)

- Finding a batch to schedule







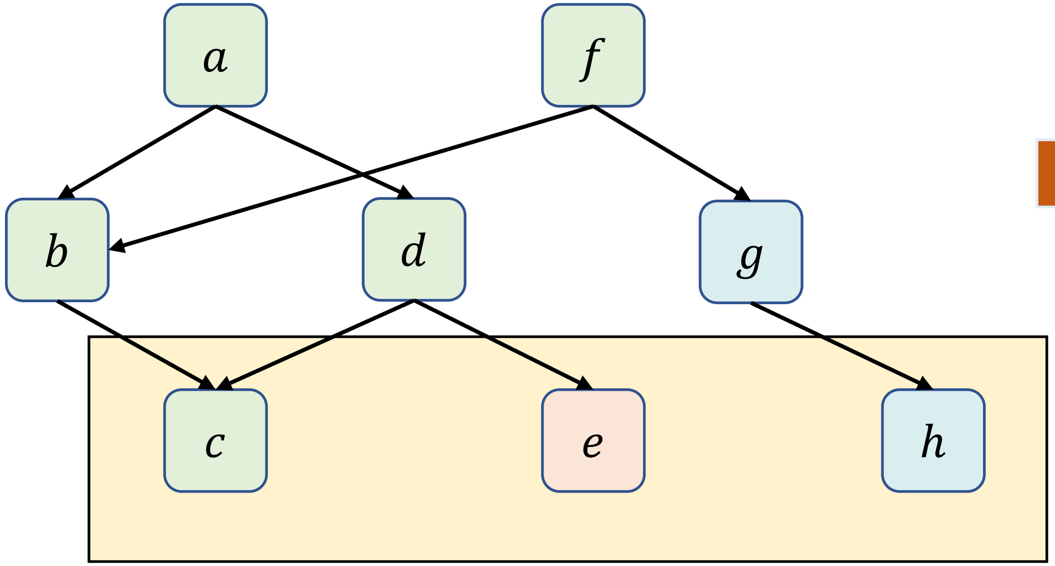
Machine 1	a	f	b	d	c
Machine 2	f	g	h		

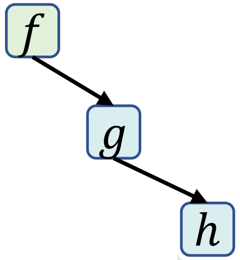
Graham's list scheduling, 1971

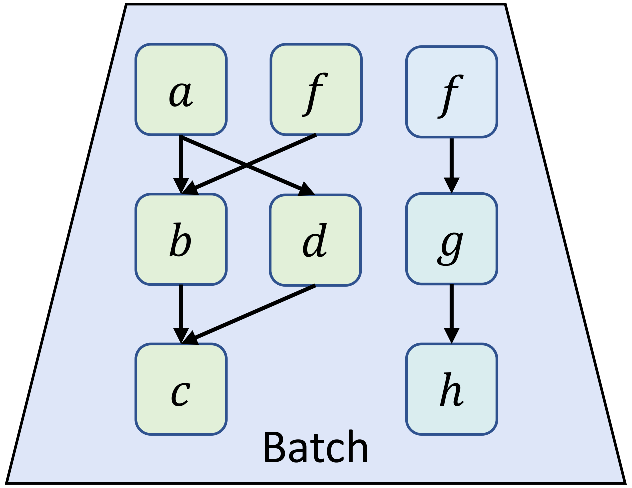
Lepere-Rapine Algorithm (+Modifications)

- Finding a batch to schedule

$$\Omega(n\rho^2 + m\rho)$$



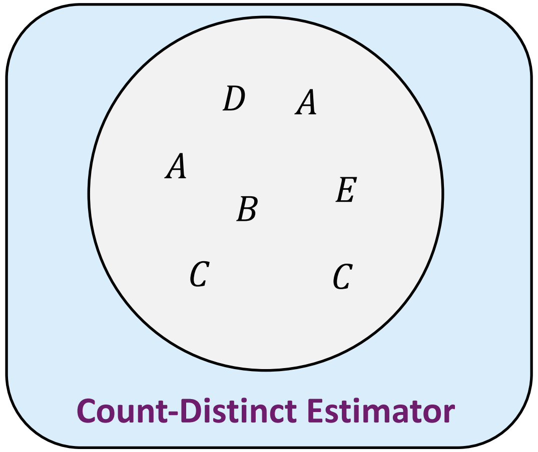


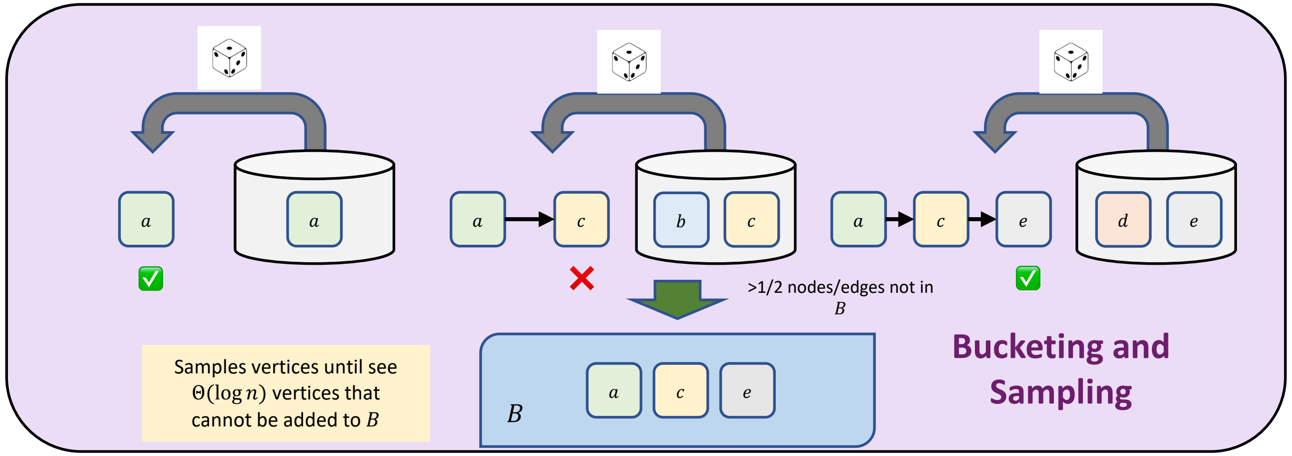


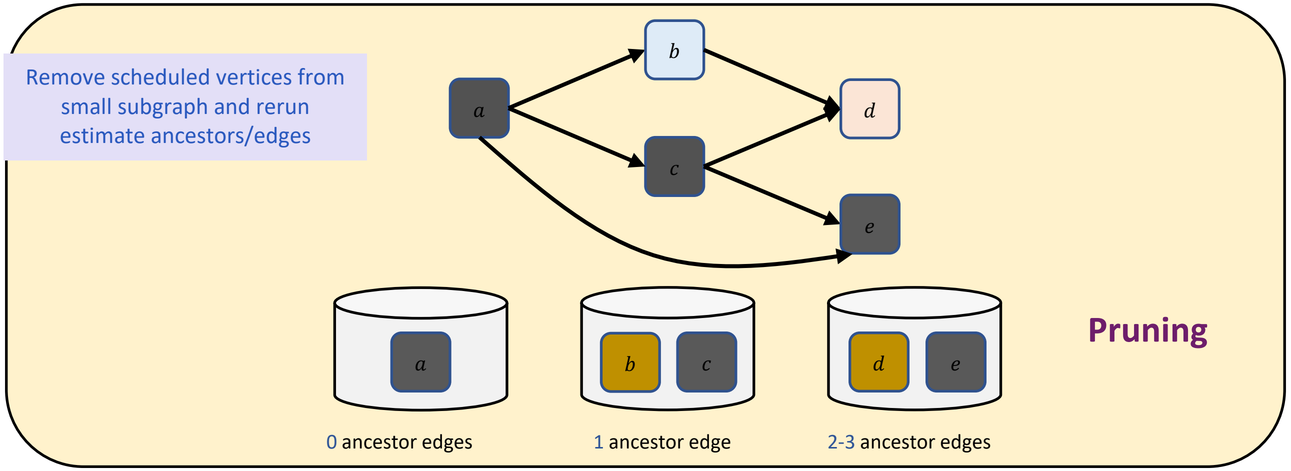
Graham's list scheduling, 1971

Machine 1	a	f	b	d	c
Machine 2	f	g	h		

Our Result: $O(\log\rho\,/\log\log\rho)$-approximation, $\tilde{O}(n+m)$ runtime, whp, assuming schedule has length at least ρ







Near-Linear Time Scheduling

- Estimating the Number of Ancestors

Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)

Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator

Near-Linear Time Scheduling

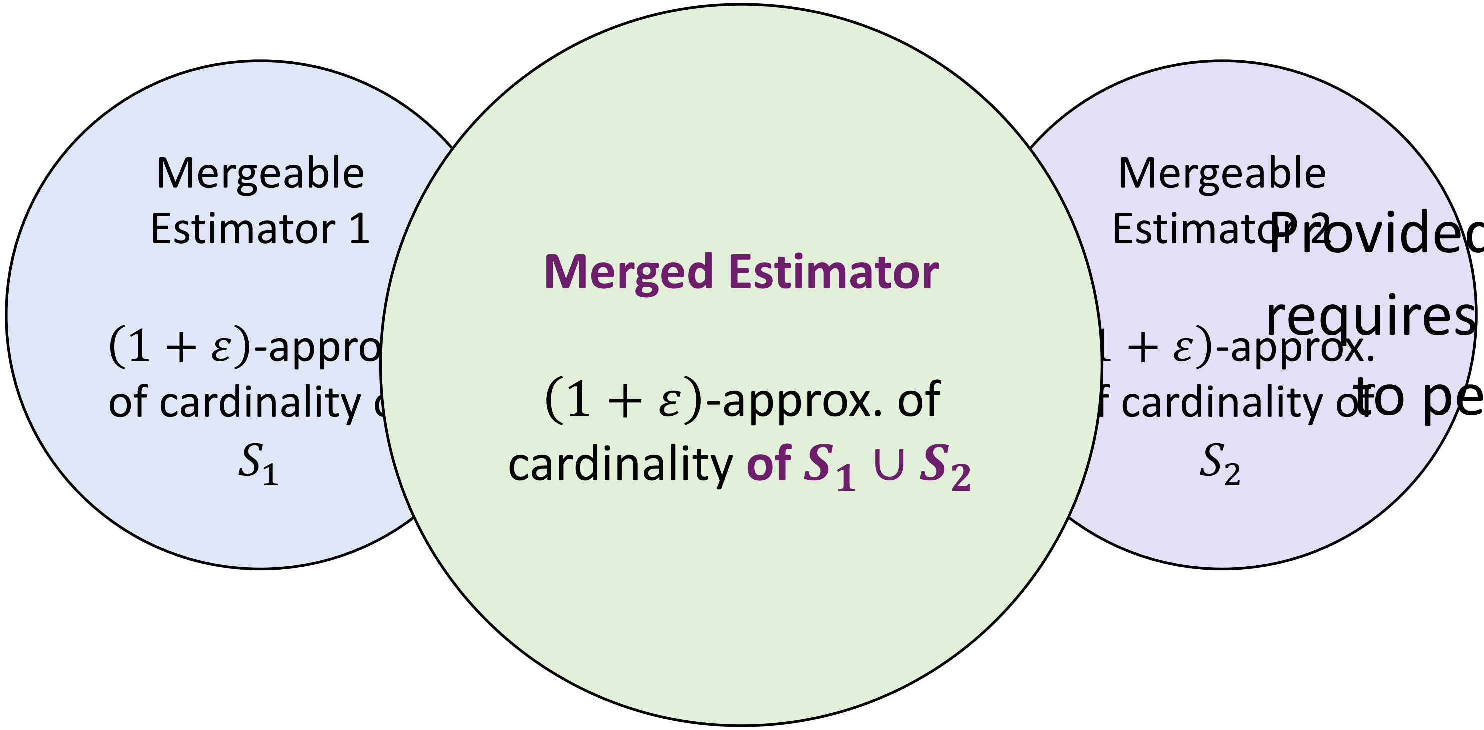
- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random '02]

Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random '02] ⟵ **Mergeable estimator**

Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random '02] ⬅ **Mergeable estimator**



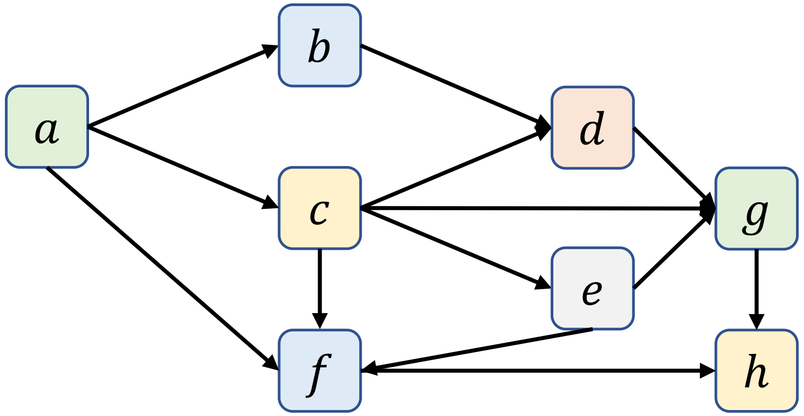
Provided $|S_1 \cup S_2| = n$, requires $O(\log^2 n)$ time to perform merge

Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random '02] ⟵ **Mergeable estimator**
    - Topsort graph and update estimators in every node

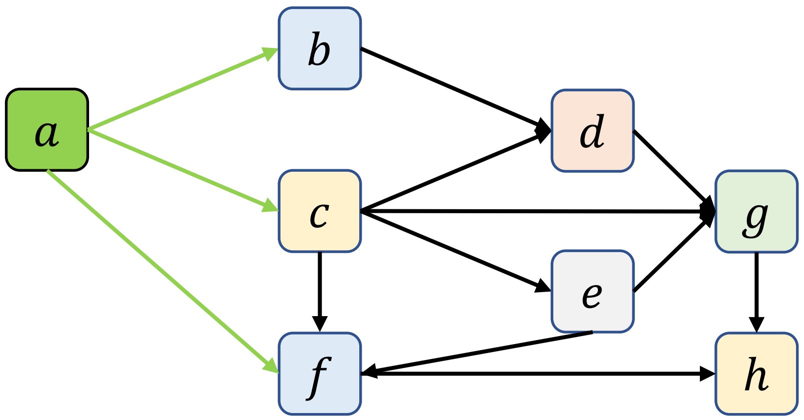
Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random ‘02] **Mergeable estimator**
    - Topsort graph and update estimators in every node



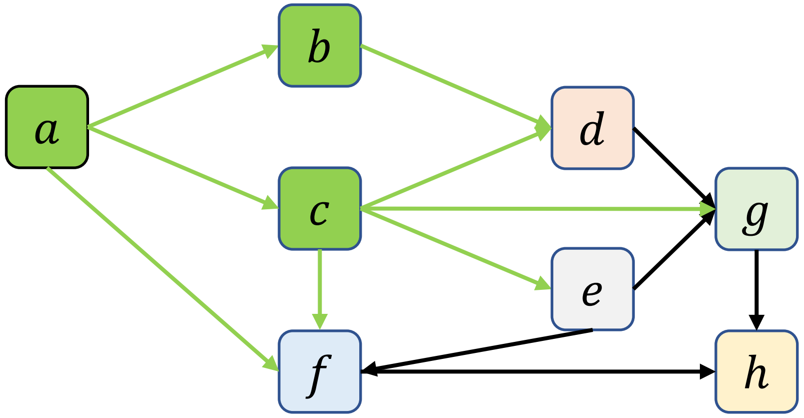
Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random '02] **Mergeable estimator**
    - Topsort graph and update estimators in every node



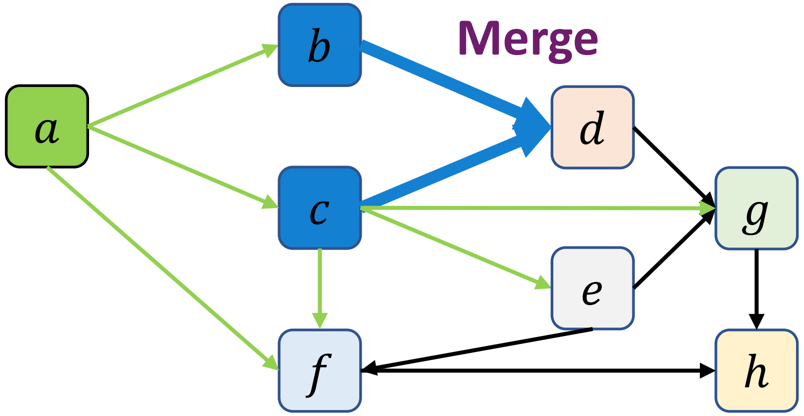
Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random '02] ⟵ **Mergeable estimator**
    - Topsort graph and update estimators in every node



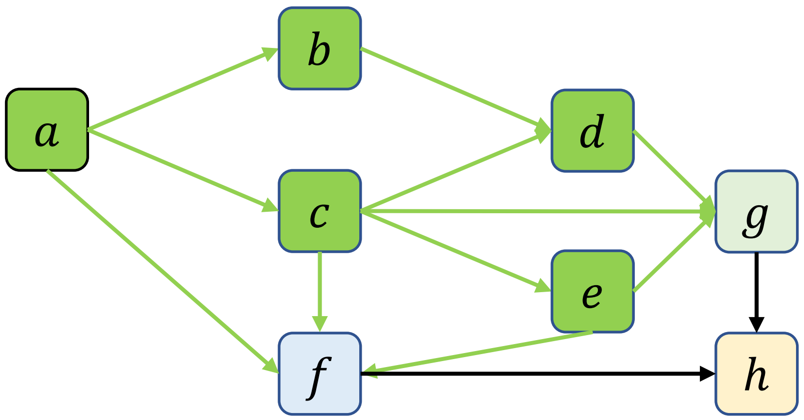
Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random '02] ⟸ **Mergeable estimator**
    - Topsort graph and update estimators in every node



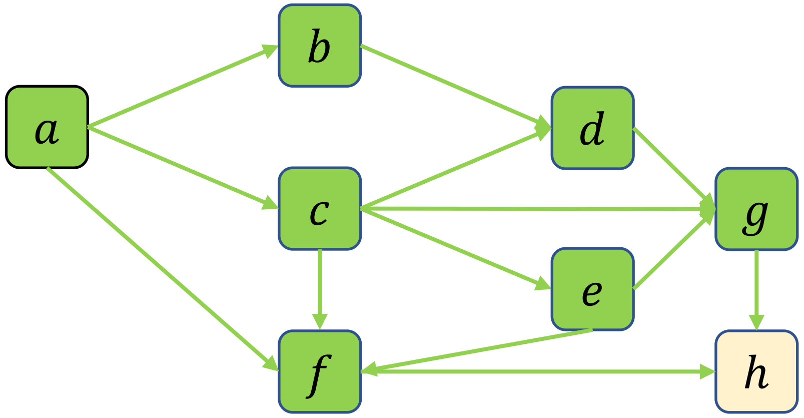
Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random '02] ⟵ **Mergeable estimator**
    - Topsort graph and update estimators in every node



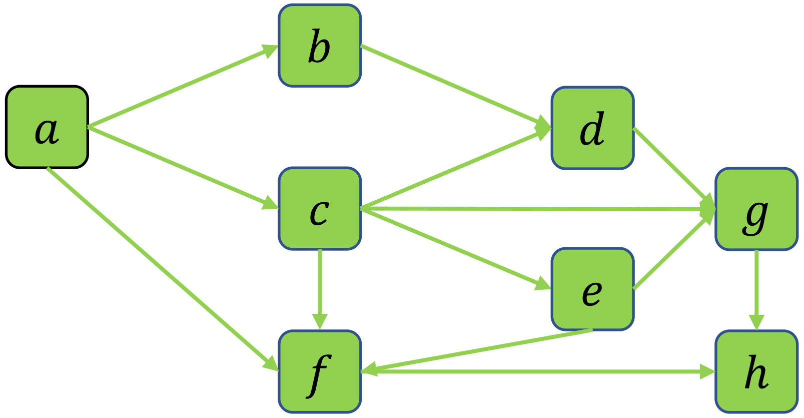
Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random '02] ⟵ **Mergeable estimator**
    - Topsort graph and update estimators in every node



Near-Linear Time Scheduling

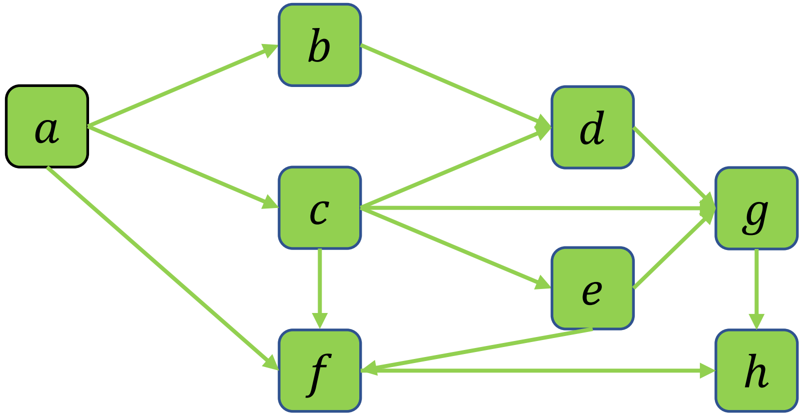
- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random '02] ← **Mergeable estimator**
    - Topsort graph and update estimators in every node



Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random '02] **Mergeable estimator**
    - Topsort graph and update estimators in every node

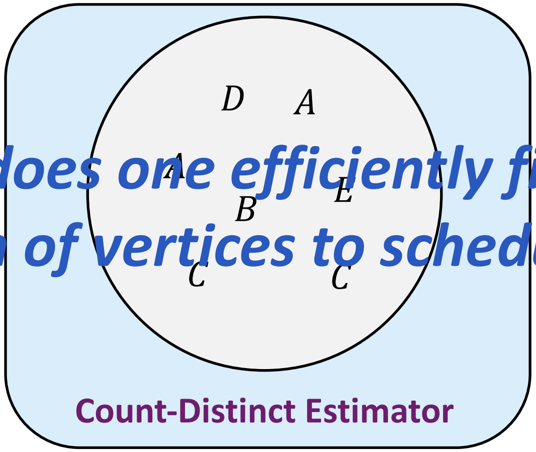
$(1 \pm \epsilon)$-approx. on number of **ancestors and edges**

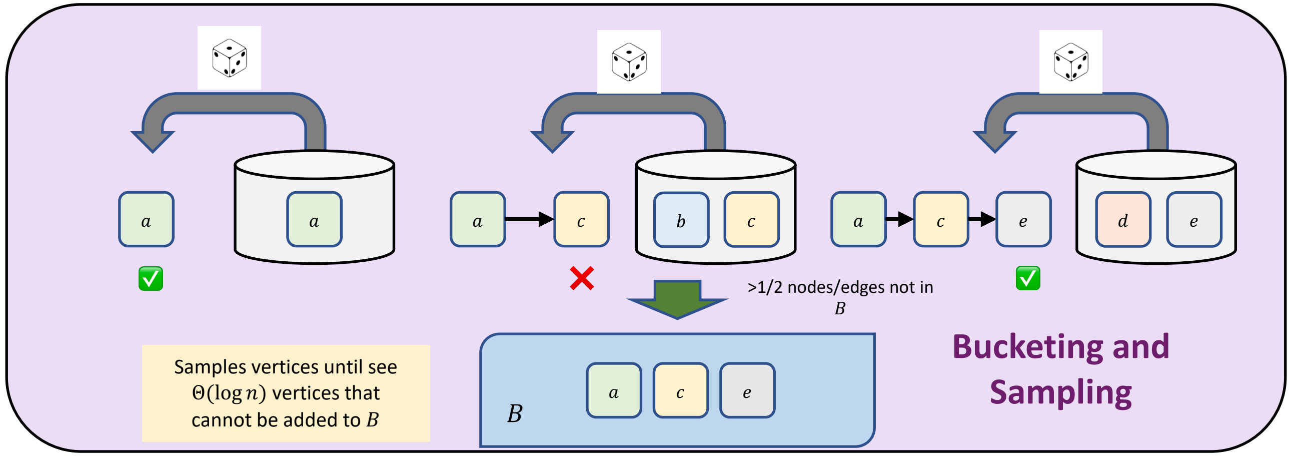


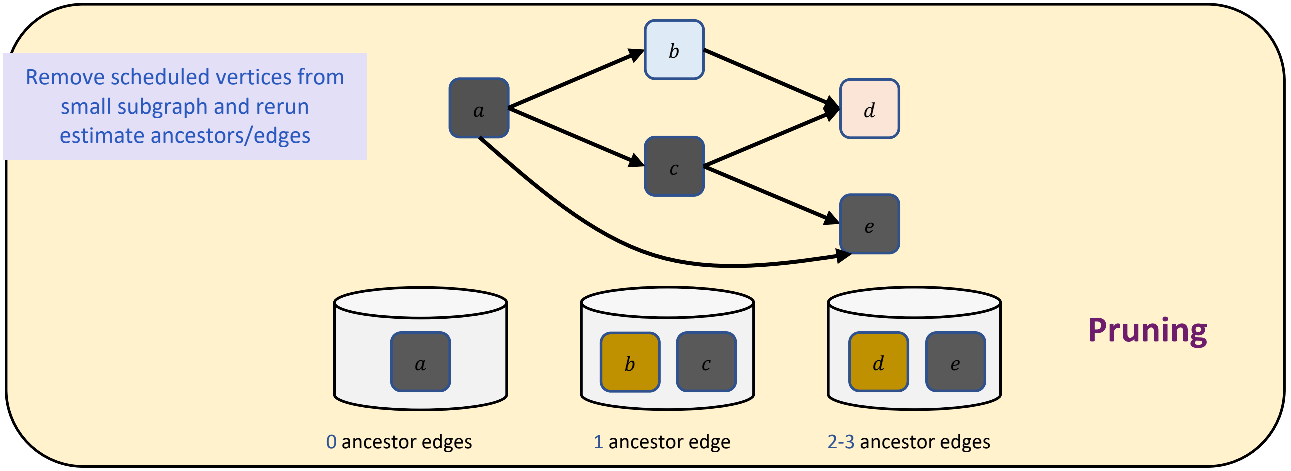
$O\left((n+m)\log^2 n\right)$

Our Result: $O(\log \rho \,/\, \log\log \rho)$-approximation, $\tilde{O}(n+m)$ runtime, whp, assuming schedule has length at least ρ

How does one efficiently find a batch of vertices to schedule?

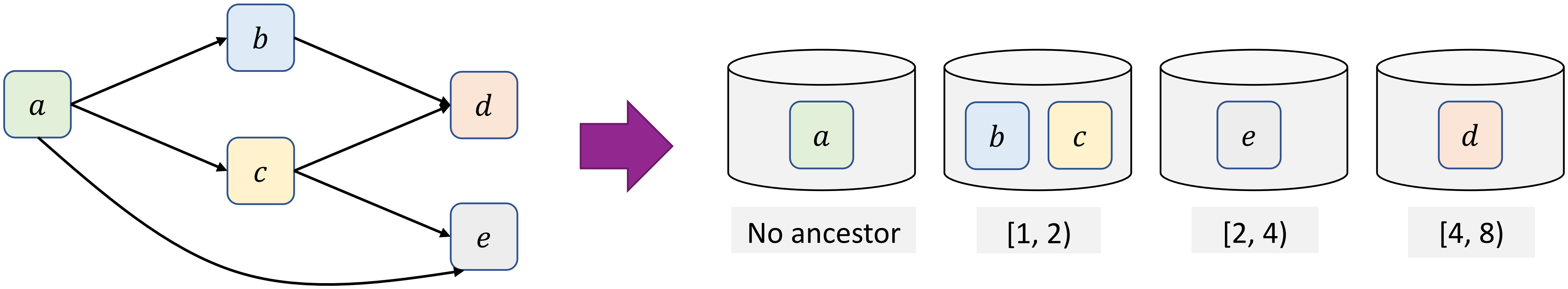






Scheduling Small Subgraph

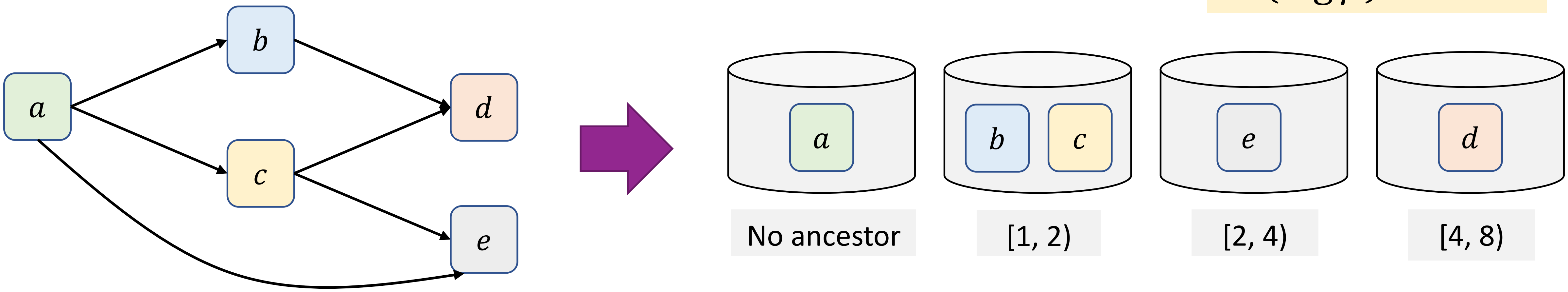
- Partition vertices into buckets where v is in bucket i if the number of ancestor edges is in $[2^i, 2^{i+1})$ (starting with $i = 0$, first bucket for nodes with no ancestors)



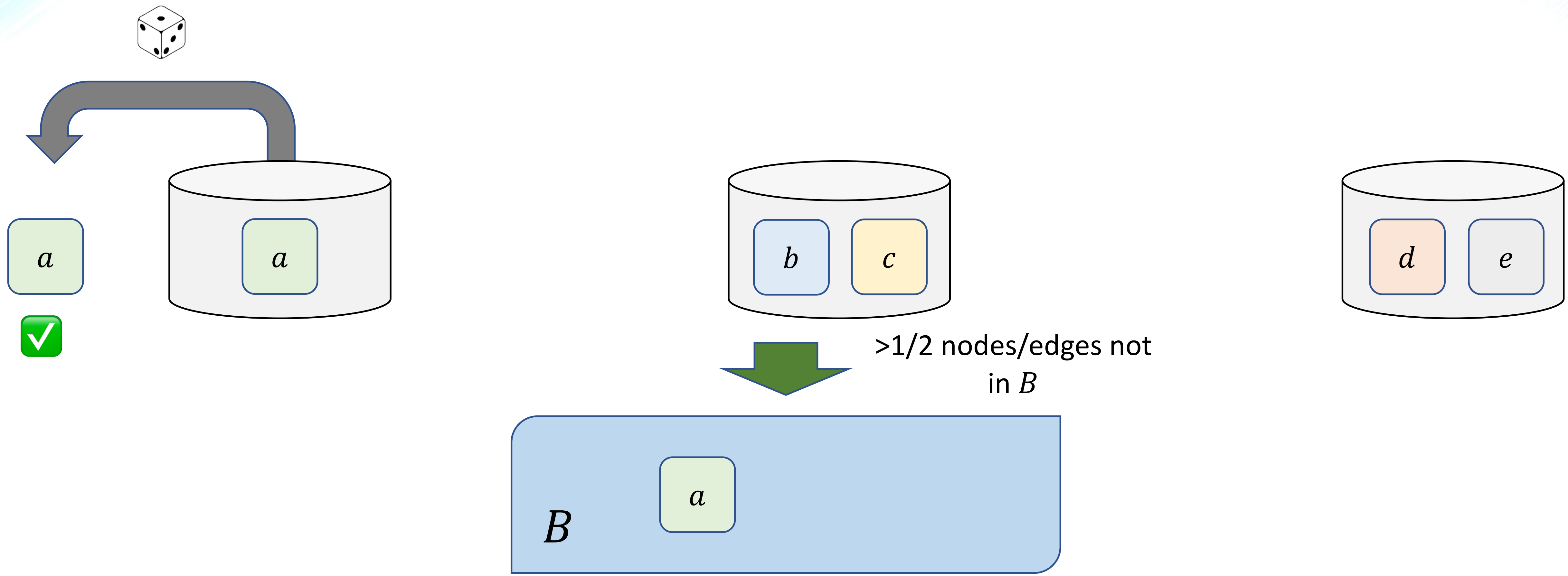
Scheduling Small Subgraph

- Partition vertices into buckets where v is in bucket i if the number of ancestor edges is in $[2^i, 2^{i+1})$ (starting with $i = 0$, first bucket for nodes with no ancestors)

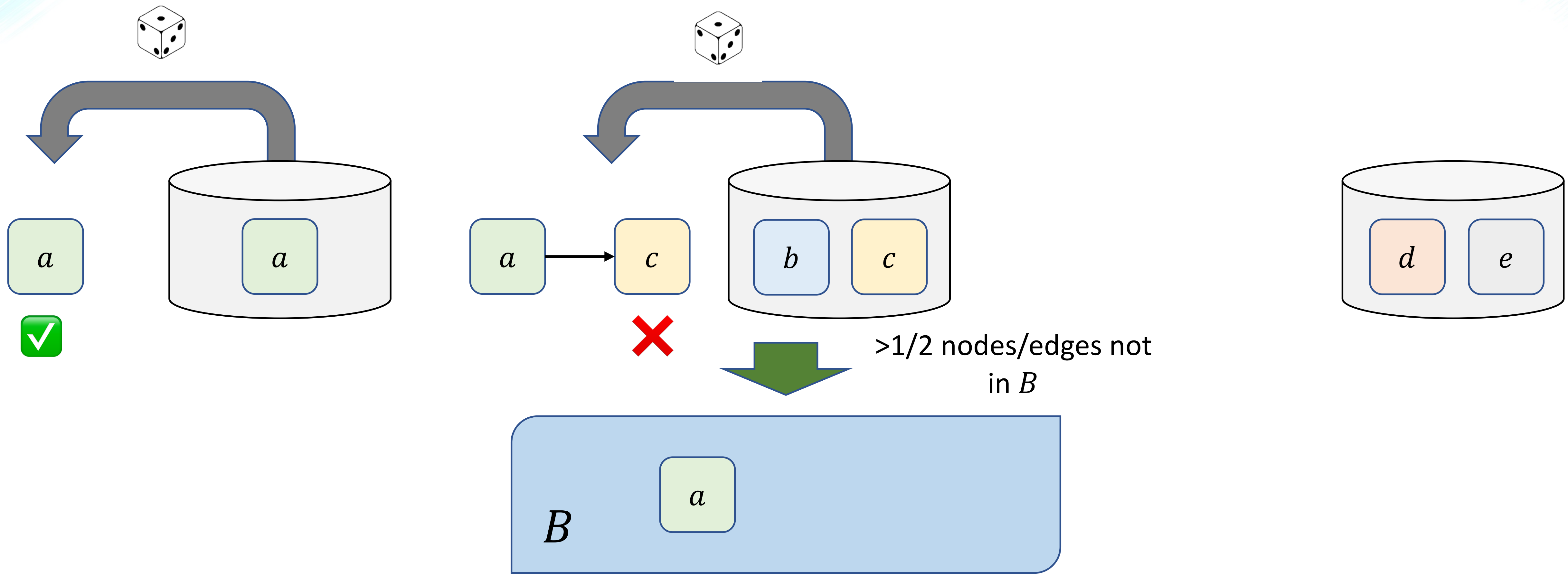
$O(\log \rho)$ buckets



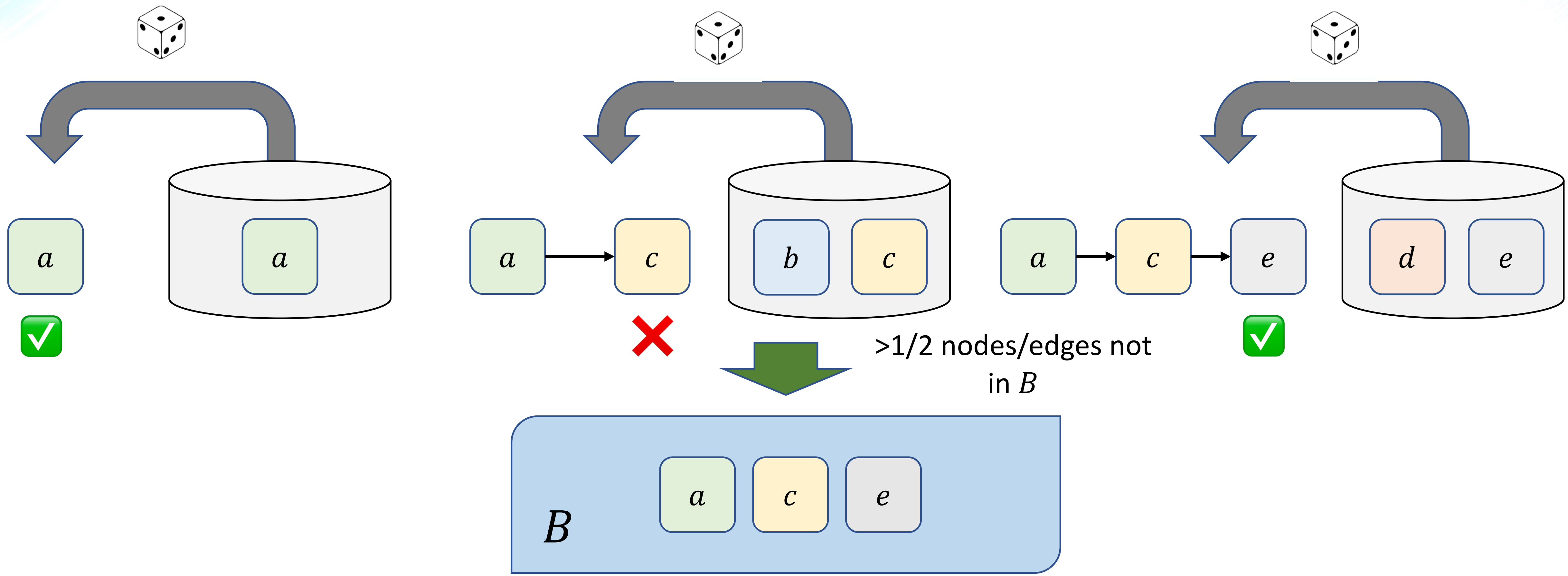
Sample Vertices from Buckets



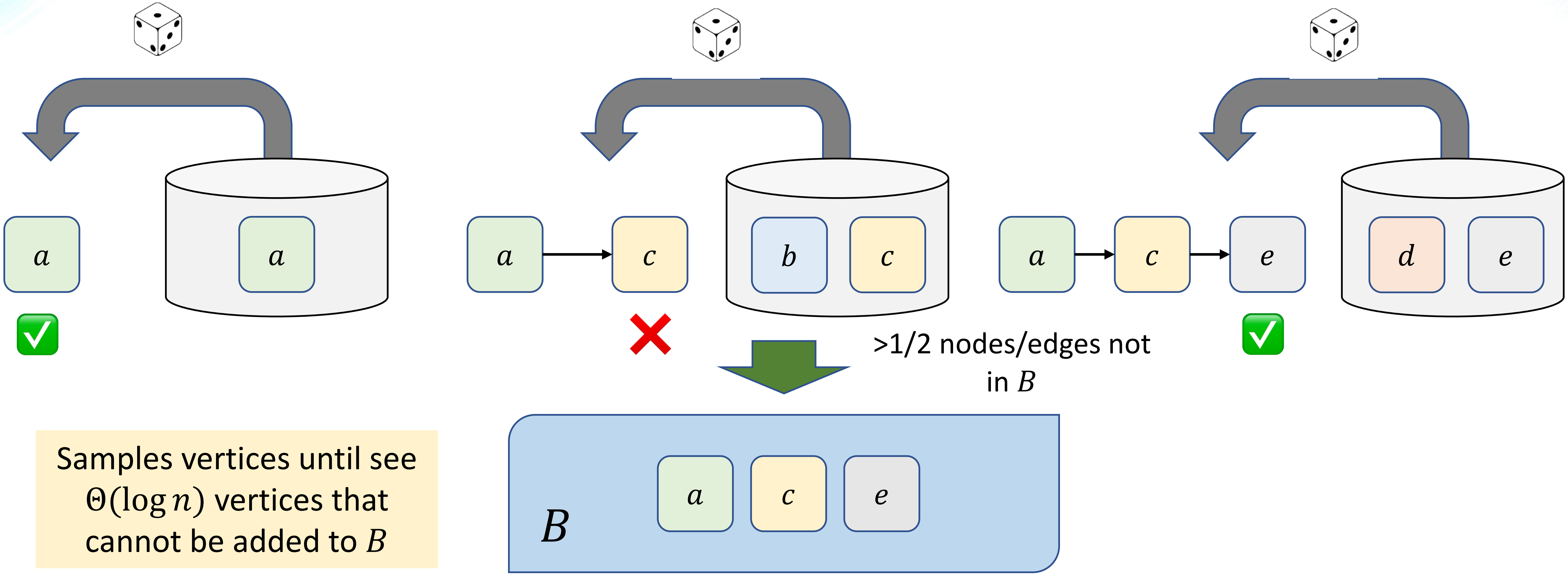
Sample Vertices from Buckets



Sample Vertices from Buckets

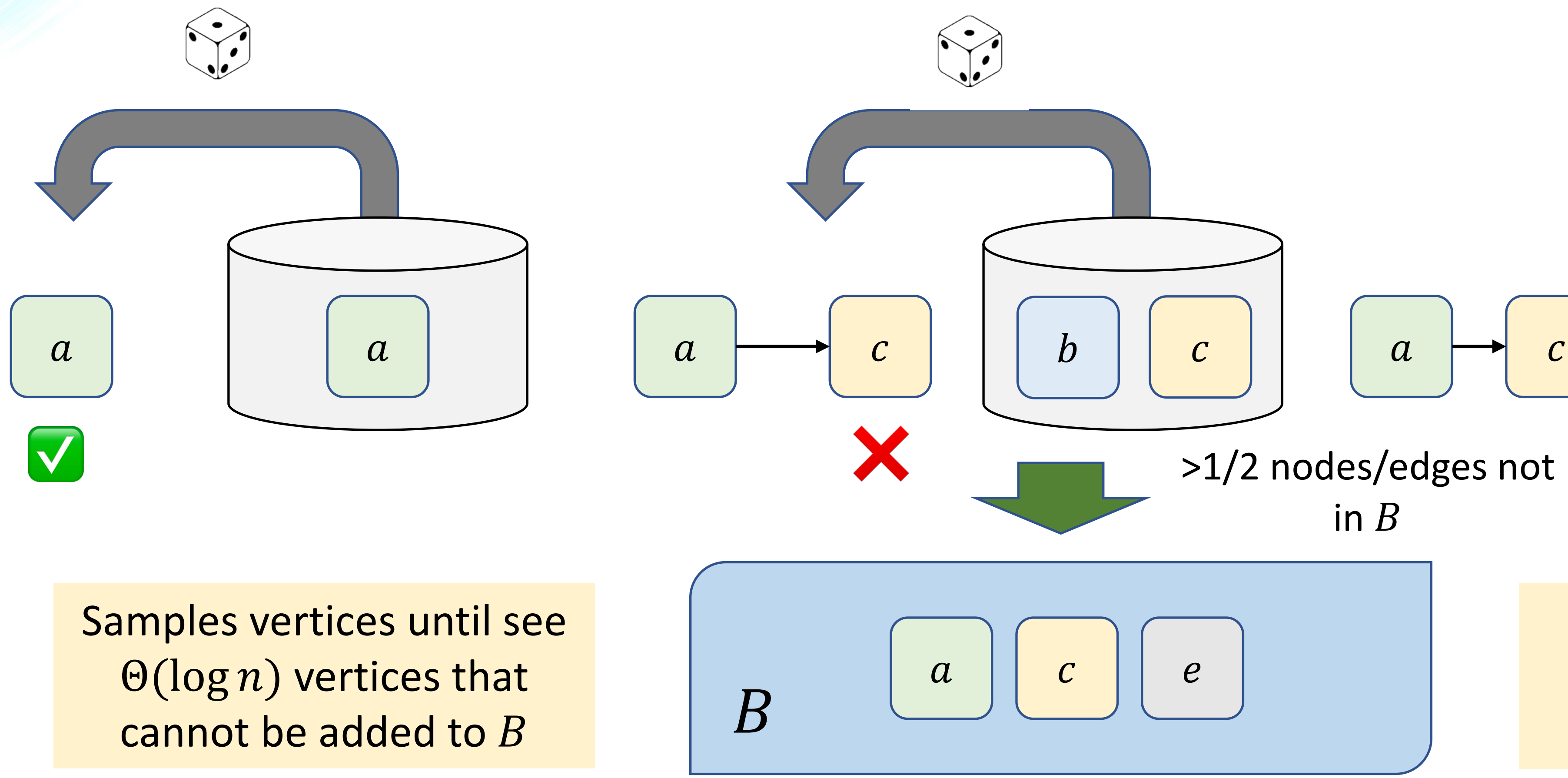


Sample Vertices from Buckets



Samples vertices until see $\Theta(\log n)$ vertices that cannot be added to B

Sample Vertices from Buckets

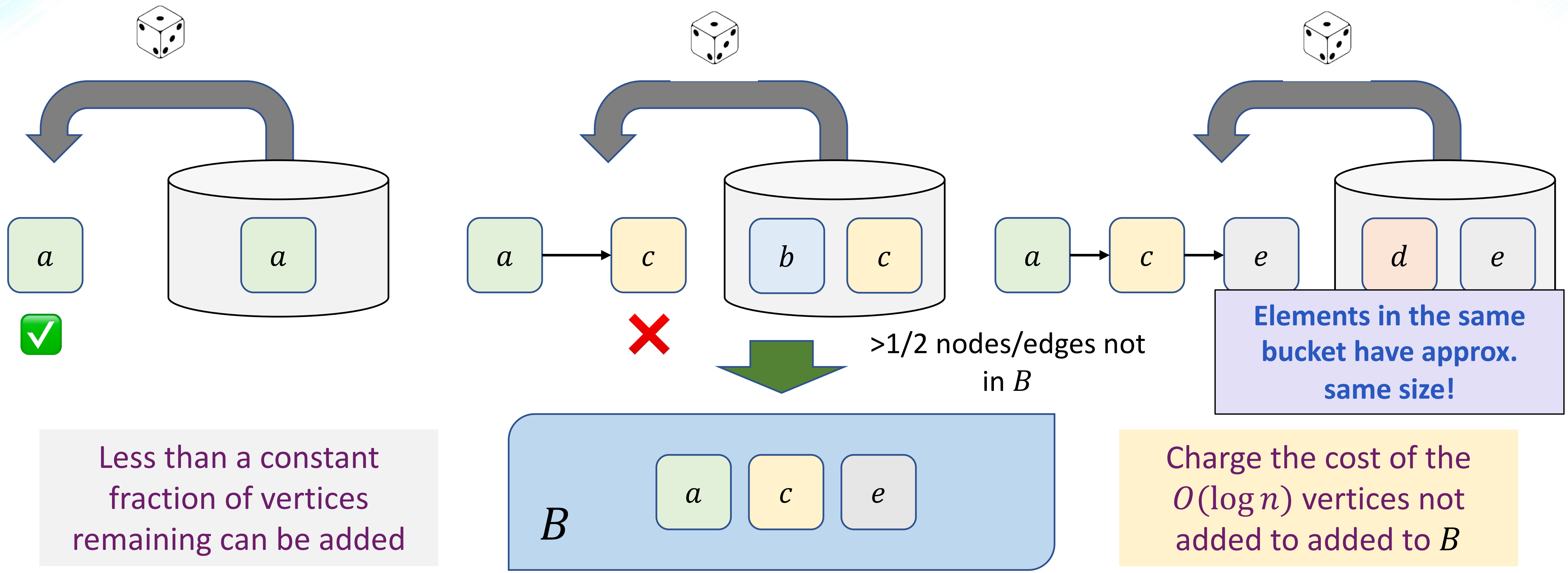


Elements in the same bucket have approx. same size!

Samples vertices until see $\Theta(\log n)$ vertices that cannot be added to B

Charge the cost of the $O(\log n)$ vertices not added to added to B

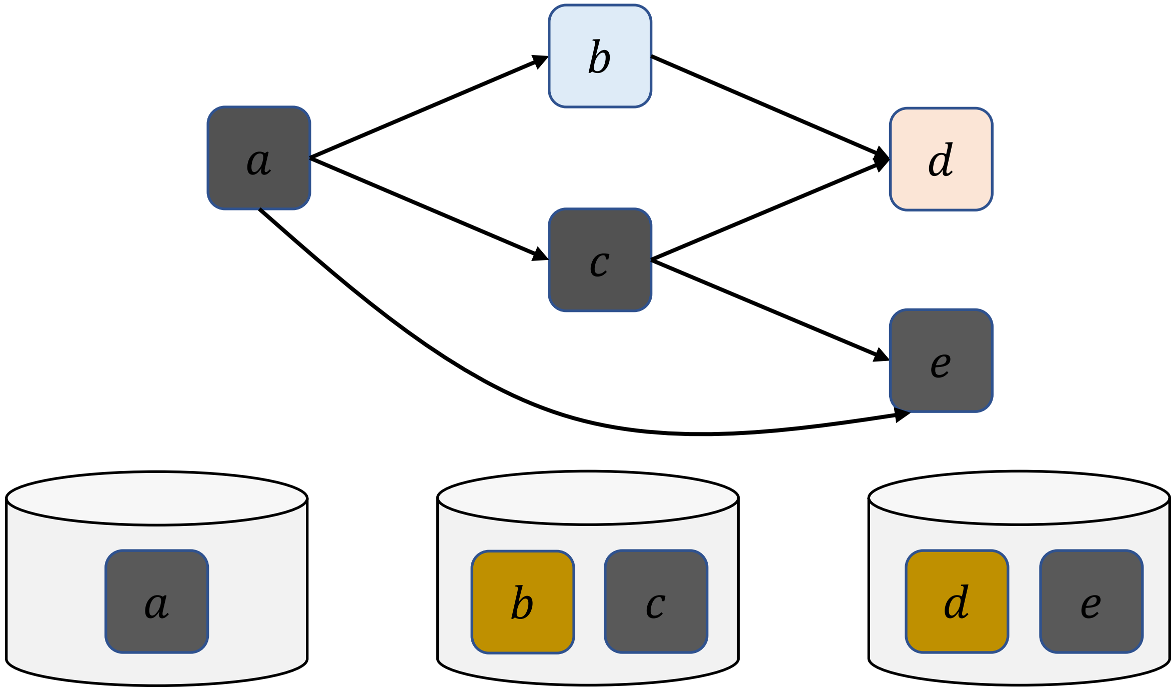
Sample Vertices from Buckets



Less than a constant fraction of vertices remaining can be added

Charge the cost of the $O(\log n)$ vertices not added to added to B

Pruning Vertices



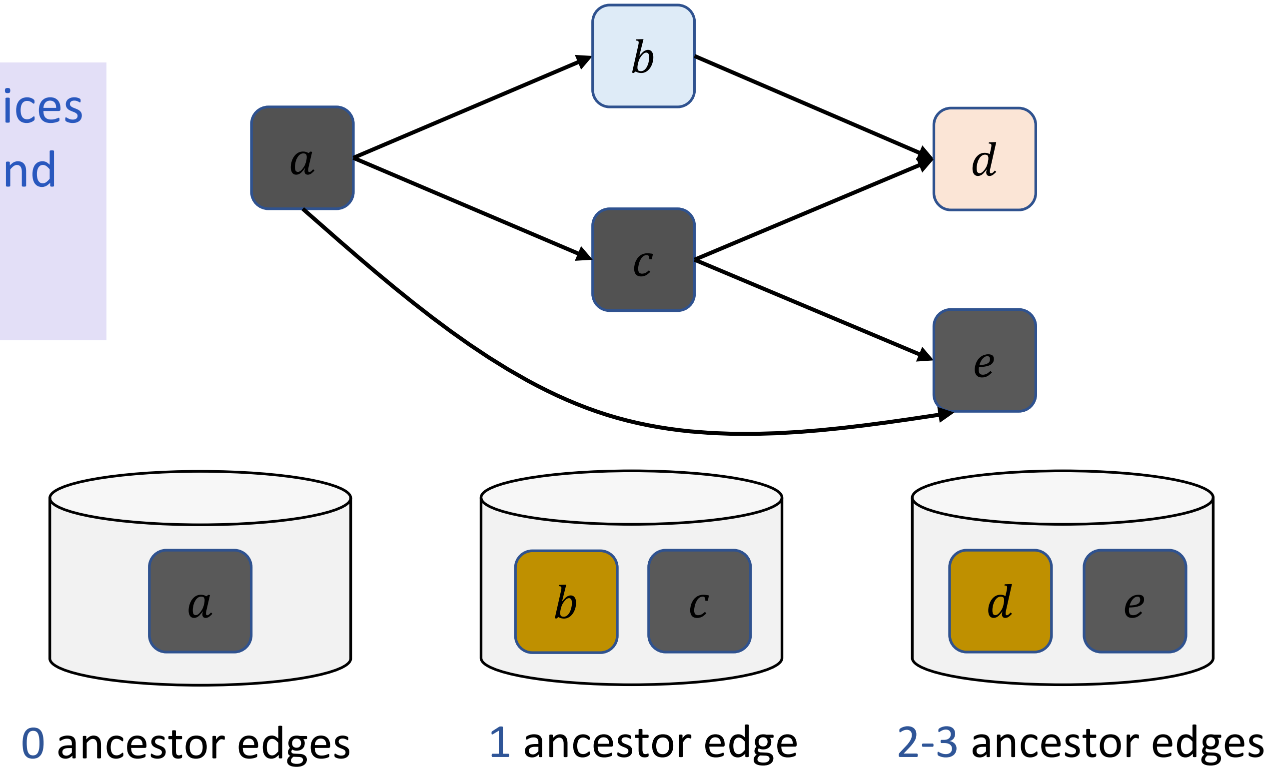
0 ancestor edges

1 ancestor edge

2-3 ancestor edges

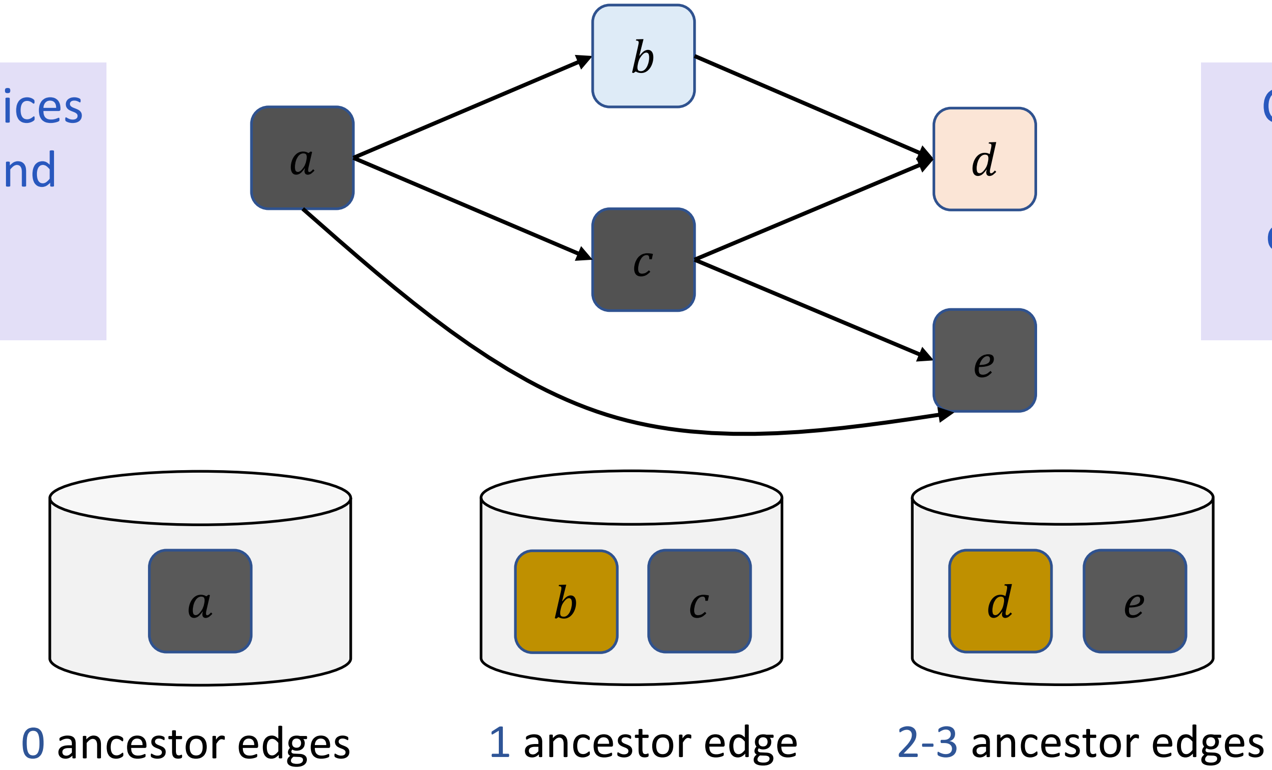
Pruning Vertices

Remove scheduled vertices from small subgraph and rerun estimate ancestors/edges



Pruning Vertices

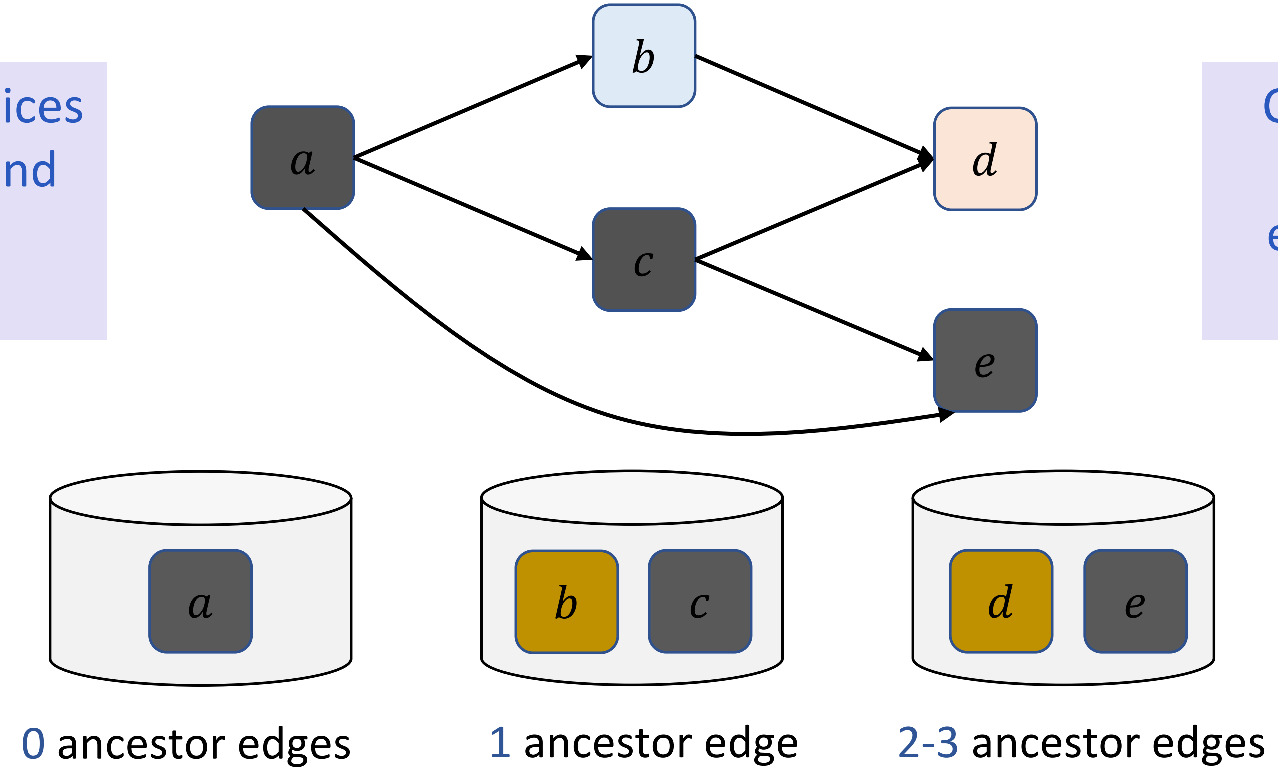
Remove scheduled vertices from small subgraph and rerun estimate ancestors/edges



Calculate ratio between new estimate and old estimate—**prune if less than 1/2**

Pruning Vertices

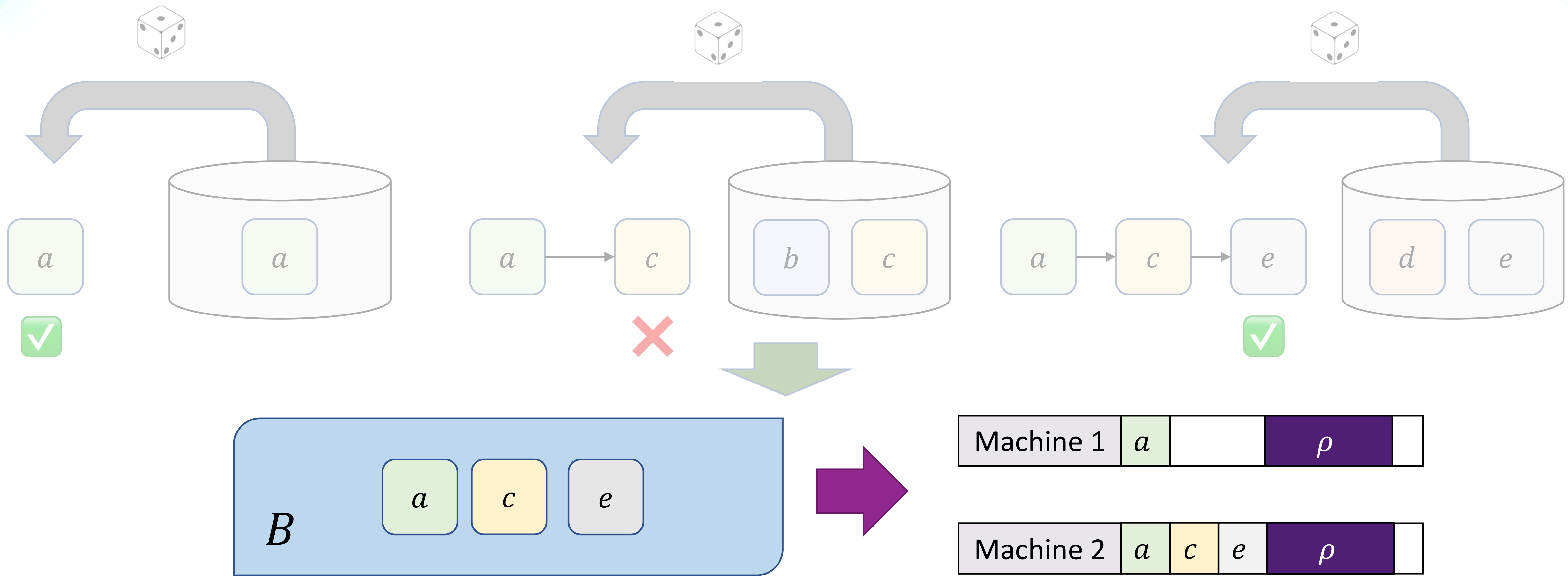
Remove scheduled vertices from small subgraph and rerun estimate ancestors/edges



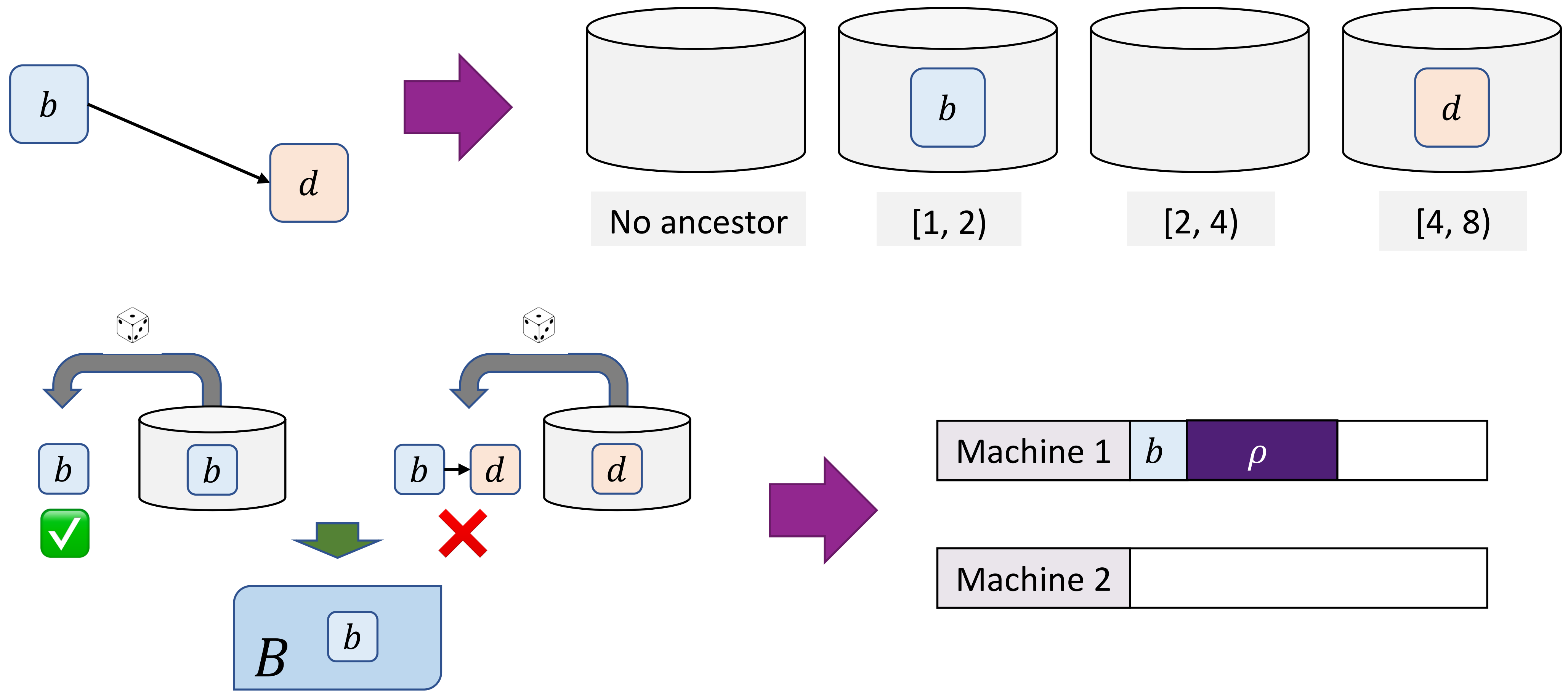
Calculate ratio between new estimate and old estimate—**prune if less than 1/2**

$O((n_S + m_S)\log^2 n)$

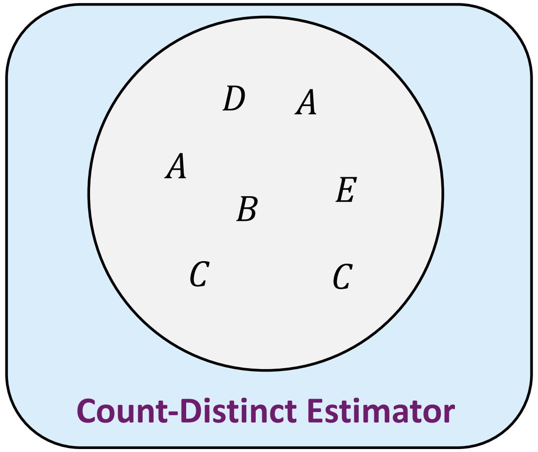
Schedule Bucket

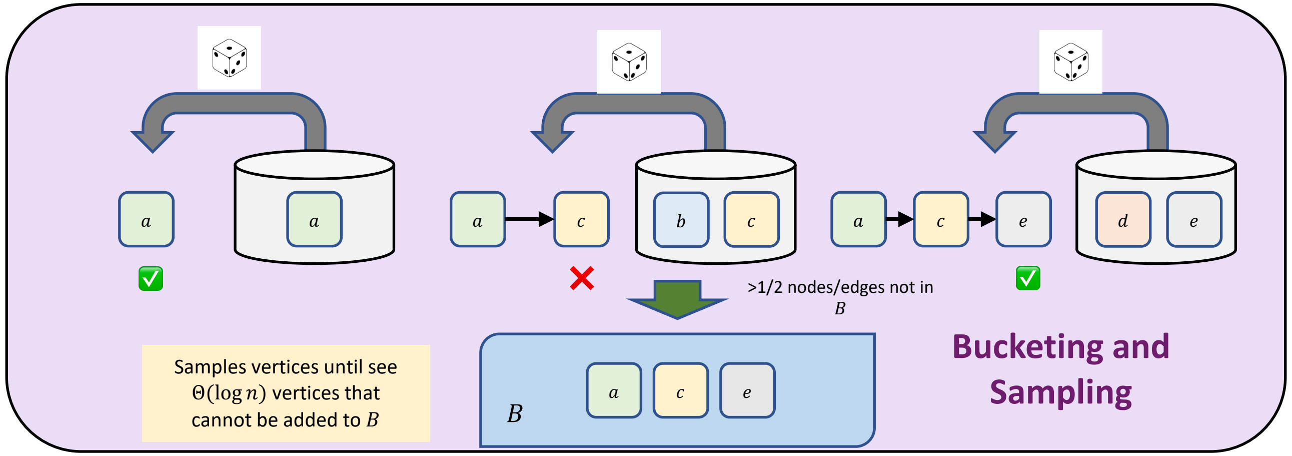


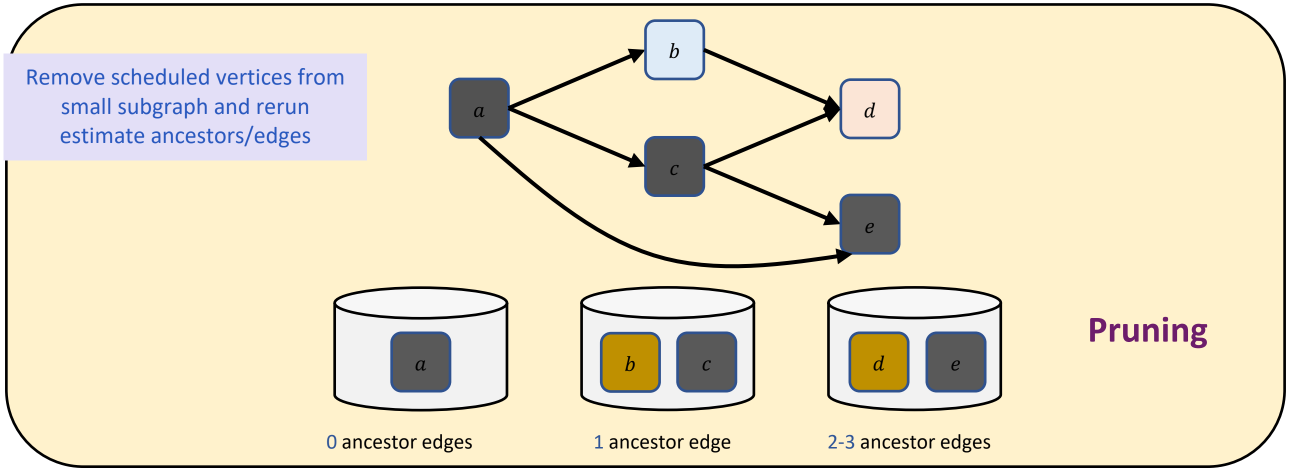
Schedule Remaining Vertices



Our Result: $O(\log \rho \,/\, \log\log \rho)$-approximation, $\tilde{O}(n+m)$ runtime, whp, assuming schedule has length at least ρ







Runtime

- Estimate the number of ancestors/edges: $O(m \ln^2 n)$

Runtime

- Estimate the number of ancestors/edges: $O(m \ln^2 n)$
- Scheduling Small Subgraph Sampling Runtime:

Runtime

- Estimate the number of ancestors/edges: $O(m \ln^2 n)$
- Scheduling Small Subgraph Sampling Runtime:
  - $O(\ln \rho)$ buckets, each charge at most $O(\ln n)$ not added vertices to an added vertex

Runtime

- Estimate the number of ancestors/edges: $O(m \ln^2 n)$
- Scheduling Small Subgraph Sampling Runtime:
  - $O(\ln \rho)$ buckets, each charge at most $O(\ln n)$ not added vertices to an added vertex
  - At most $O(\ln n \cdot \ln \rho)$ charged to each element of a batch

Runtime

- Estimate the number of ancestors/edges: $O(m \ln^2 n)$
- Scheduling Small Subgraph Sampling Runtime:
  - $O(\ln \rho)$ buckets, each charge at most $O(\ln n)$ not added vertices to an added vertex
  - At most $O(\ln n \cdot \ln \rho)$ charged to each element of a batch
  - $O(\ln \rho)$ iterations of scheduling batches

Runtime

- Estimate the number of ancestors/edges: $O(m \ln^2 n)$
- Scheduling Small Subgraph Sampling Runtime:
  - $O(\ln \rho)$ buckets, each charge at most $O(\ln n)$ not added vertices to an added vertex
  - At most $O(\ln n \cdot \ln \rho)$ charged to each element of a batch
  - $O(\ln \rho)$ iterations of scheduling batches
  - $O(|E_S| \cdot \ln n \cdot \ln \rho \cdot \ln \rho)$ total cost over all iterations

Runtime

- Estimate the number of ancestors/edges: $O(m \ln^2 n)$
- Scheduling Small Subgraph Sampling Runtime:
  - $O(\ln \rho)$ buckets, each charge at most $O(\ln n)$ not added vertices to an added vertex
  - At most $O(\ln n \cdot \ln \rho)$ charged to each element of a batch
  - $O(\ln \rho)$ iterations of scheduling batches
  - $O(|E_S| \cdot \ln n \cdot \ln \rho \cdot \ln \rho)$ total cost over all iterations
- Pruning Runtime: $O(|E_S| \cdot \ln^3 n \cdot \ln \rho)$

Runtime

- Estimate the number of ancestors/edges: $O(m \ln^2 n)$
- Scheduling Small Subgraph Sampling Runtime:
  - $O(\ln \rho)$ buckets, each charge at most $O(\ln n)$ not added vertices to an added vertex
  - At most $O(\ln n \cdot \ln \rho)$ charged to each element of a batch
  - $O(\ln \rho$ …
  - $O(|E_S|$ … total cost over all iterations
- Pruning Runtime: $O(|E_S| \cdot \ln^3 n \cdot \ln \rho)$

Total: $O(m \ln^3 n \ln \rho + n \ln M)$

Conclusion

Main challenge: efficiently determining which jobs to schedule in a batch of jobs

Solution: size-estimation via sketching, sampling and pruning, and work charging argument

Open Questions:

1. Can we get a **linear time** algorithm?
2. Near-linear time algorithm for **non-uniform machines and non-unit jobs**.
3. Can we obtain a **linear-time transformation** for a result *without* duplication?