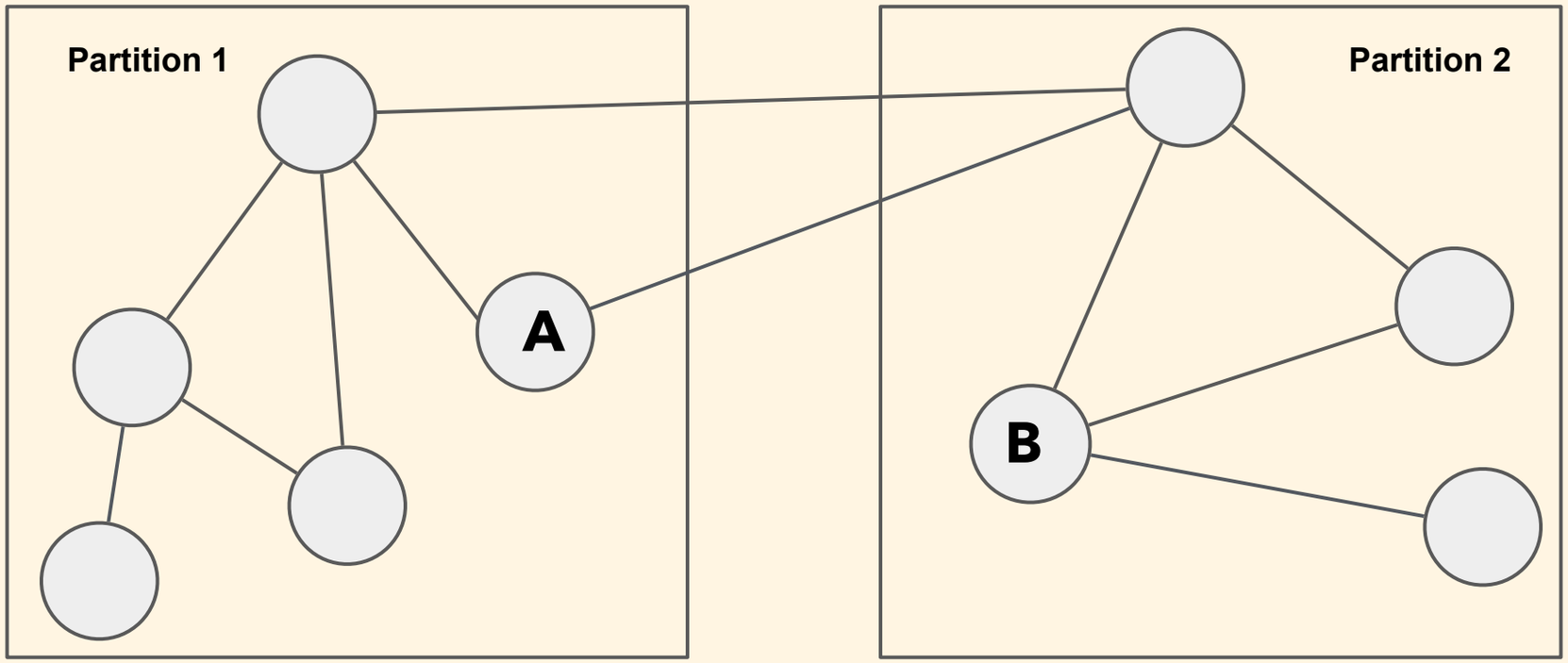
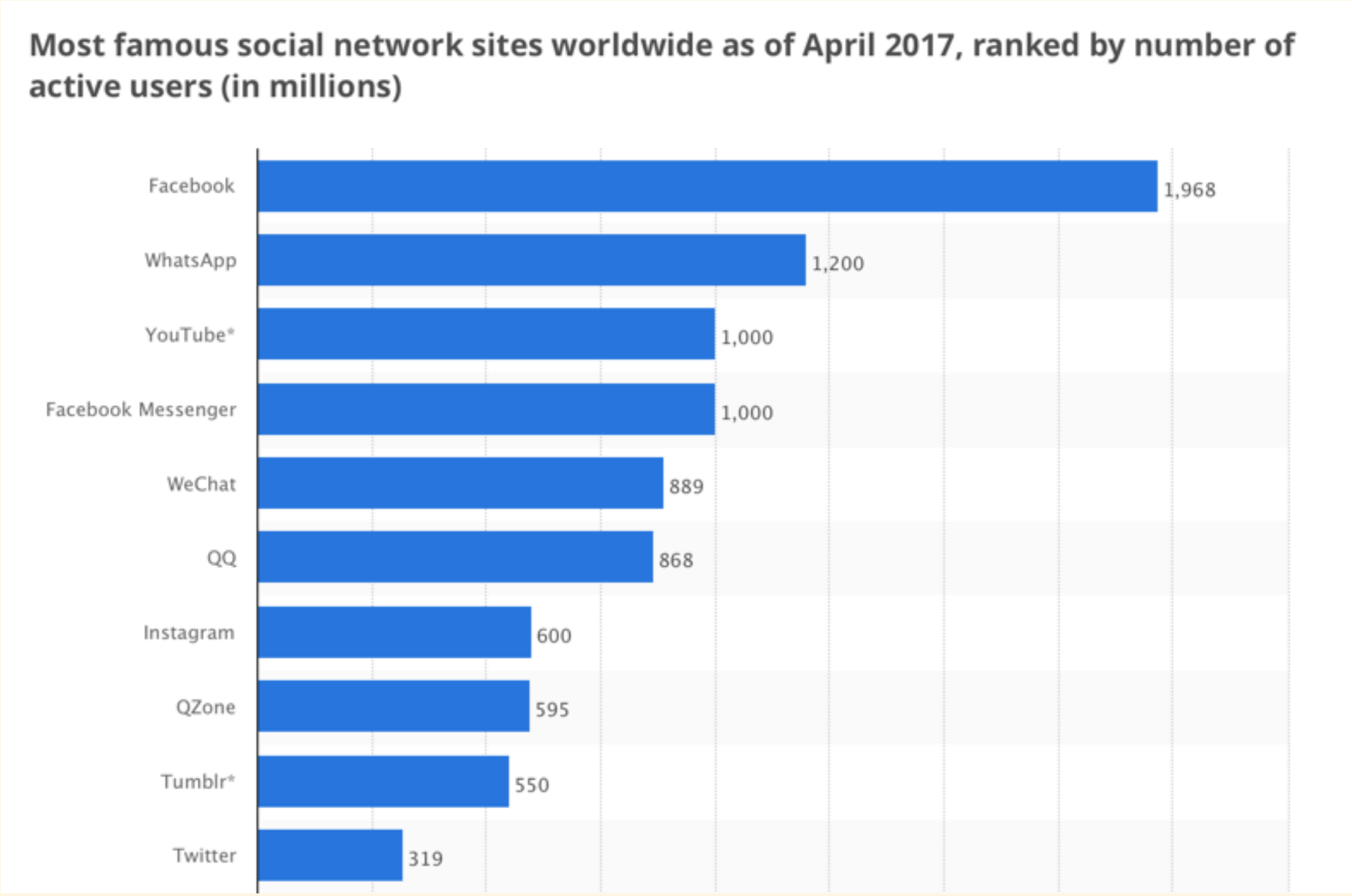
Leopard: Lightweight Edge-Oriented Partitioning and Replication for Dynamic Graphs

presented by Qixuan Li

What is Graph Partitioning?



Why Graph Partitioning?



Social Network

Why Graph Partitioning?

- Large graphs are becoming increasingly prevalent
- Large graph datasets are too large to manage on a single machine

Ideal Graph Partitioning Algorithm

- As few cut as possible
- As balance as possible

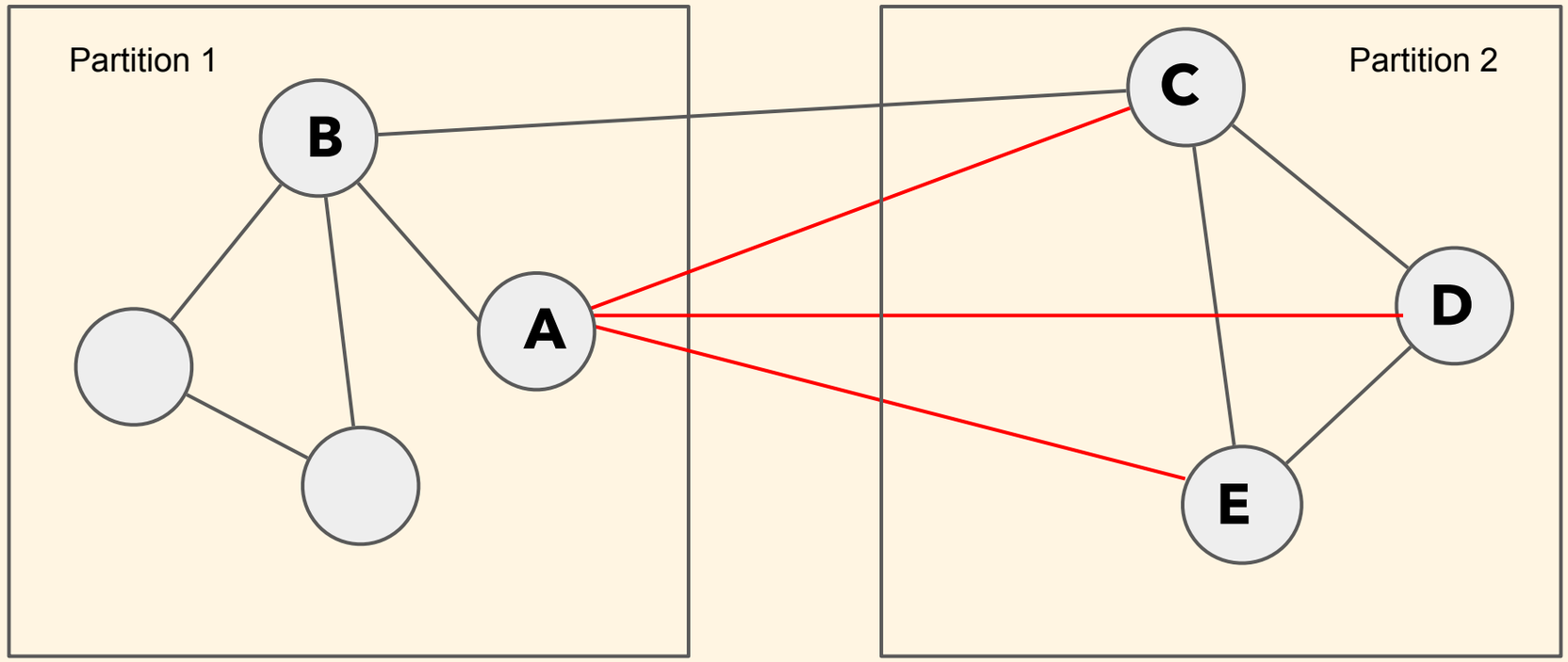
Conflict!

Dynamic Graph Partitioning

- Graph is always changing
- NP-hard

Dynamic Graph Partitioning Example

Is A still in Partition 1?



Leopard

- Algorithm Overview
  - Vertex Assignment
  - Vertex Reassignment
  - Computation Skipping
- Integration with Replication

Algorithm Overview

- Incrementally maintain a quality graph partitioning, dynamically adjusting as new vertices and edge are added to the graph
- Integrates consideration of replication with partitioning

Objective Function

FENNEL Scoring Function:

$$\underset{1 \leq i \leq k}{\operatorname{argmax}} \{ |N(v) \cap P_i| - \alpha \frac{\gamma}{2} (|P_i|)^{\gamma - 1} \}$$

N(v): the set of neighbors of vertex v
Pi: partition i
k: # partitions

Leopard

- Algorithm Overview
  - Vertex Assignment
  - Vertex Reassignment
  - Computation Skipping
- Integration with Replication

Vertex Assignment

- Given a vertex u:
  - compute the best partition for u using the objective function and put u into that partition

Observation: Vertex Assignment

- Graph is changing
- Need to consider adds/deletes

Leopard

- Algorithm Overview
  - Vertex Assignment
  - Vertex Reassignment
  - Computation Skipping
- Integration with Replication

Vertex Reassignment v1

Slow!

- When add/delete an edge (u, v)
  - Go through every vertex and examine for reassignment

Observation: Vertex Reassignment v1

- Most vertices are minimally affected
- u may move to v's partition or vice versa
- ripple effect

Vertex Reassignment v2

- When an edge (u, v) is added/deleted:
  - u and v are chosen as the initial candidates for examination of potential reassignment
  - if either one of them is reassigned, add the immediate neighbors of the moved vertex to the candidate set
  - check for reassignment of every vertex in the candidate set

Still too slow!

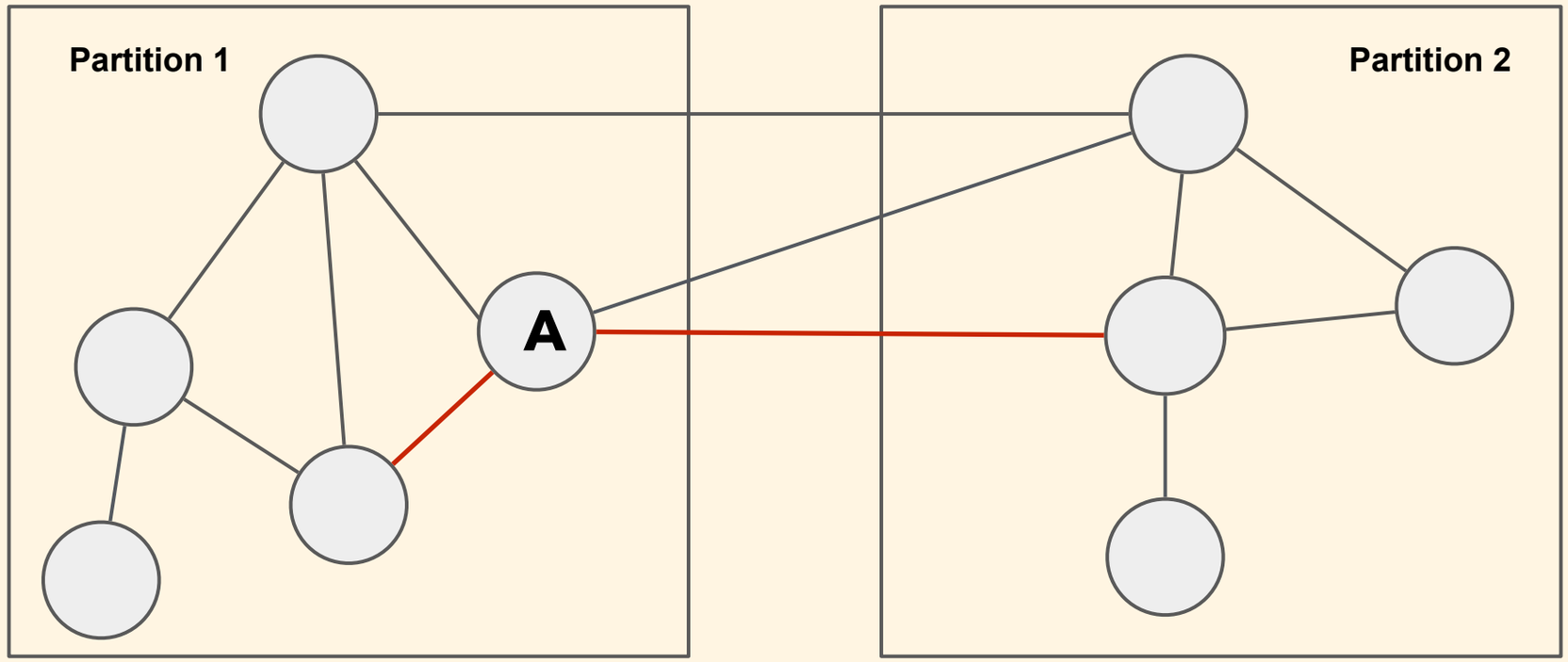
Observation: Vertex Reassignment v2

- As a vertex's number of neighbors increases, the influence of a new neighbor decreases
- Accumulate the changes until they exceed the threshold

Leopard

- Algorithm Overview
  - Vertex Assignment
  - Vertex Reassignment
  - Computation Skipping
- Integration with Replication

Computation Skipping

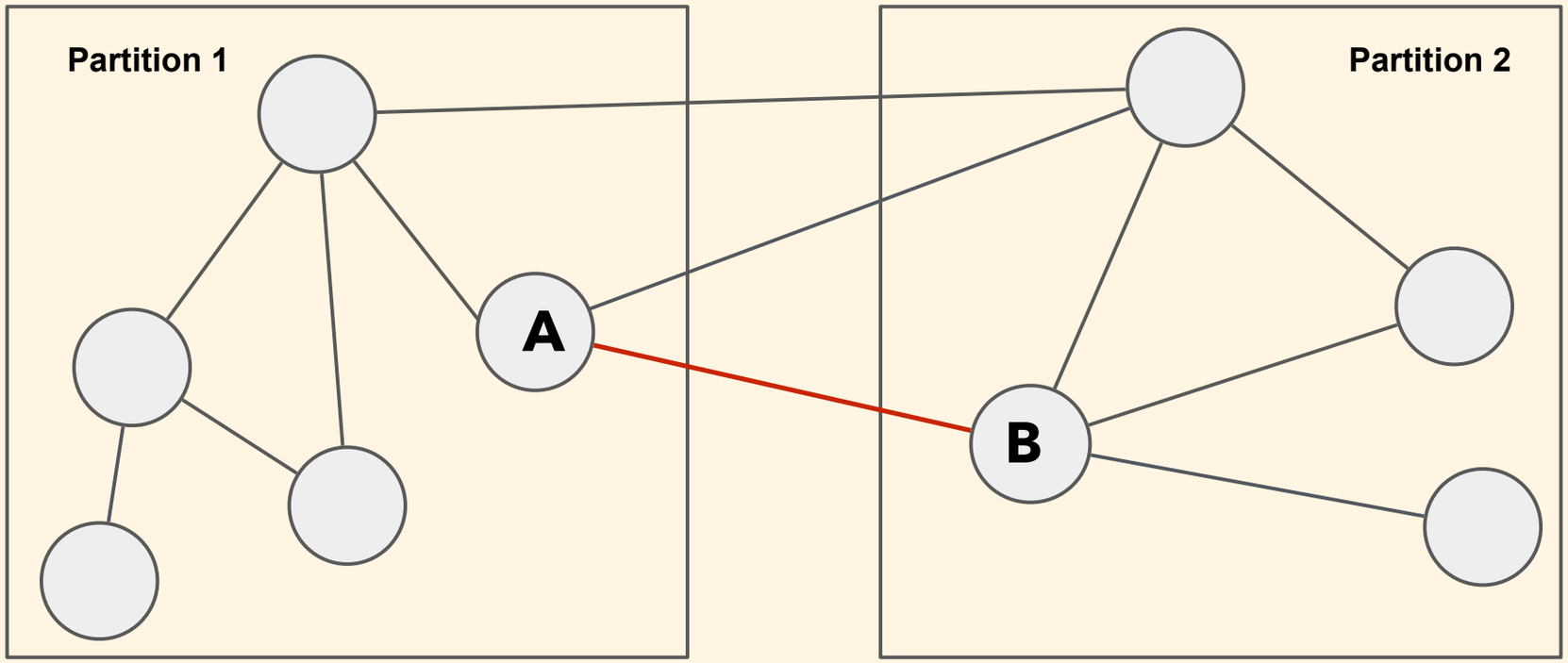


threshold = 0.5
ratio = # accumulated changes / # neighbors
(1) 1 edge is added to A, 1 / 3 = 0.33 < 0.5. Don't recompute
(2) When 1 more new edge is added for A: 2 / 4 = 0.5 = 0.5. Recompute the partition for A.
(3) Reset # accumulated changes to 0.

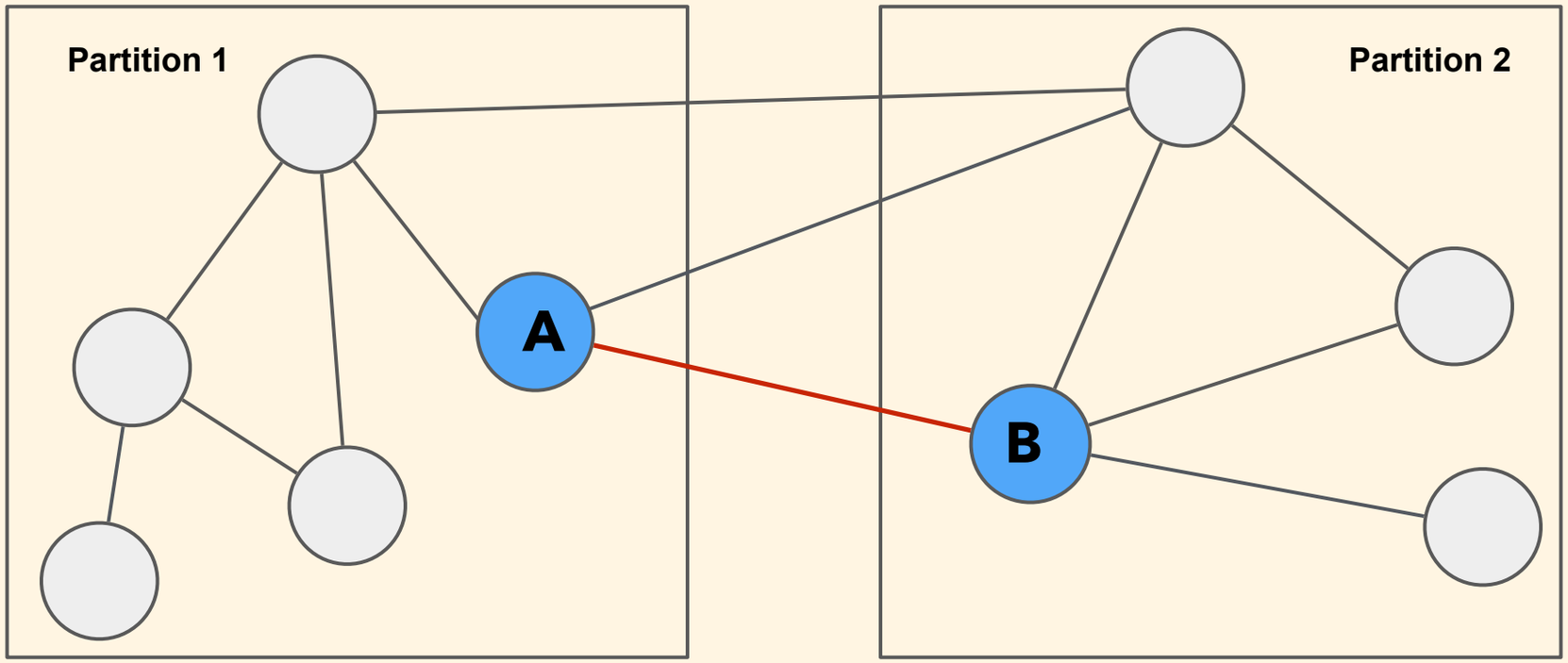
Vertex Reassignment (Final)

- When an edge (u, v) is added/deleted:
  - u and v are chosen as the initial candidates for examination of potential reassignment
  - if either one of them is reassigned, add the immediate neighbors of the moved vertex to the candidate set
  - examine vertex in the candidate set that has accumulated changes exceeding the threshold

Vertex Reassignment (Final)



Vertex Reassignment (Final)

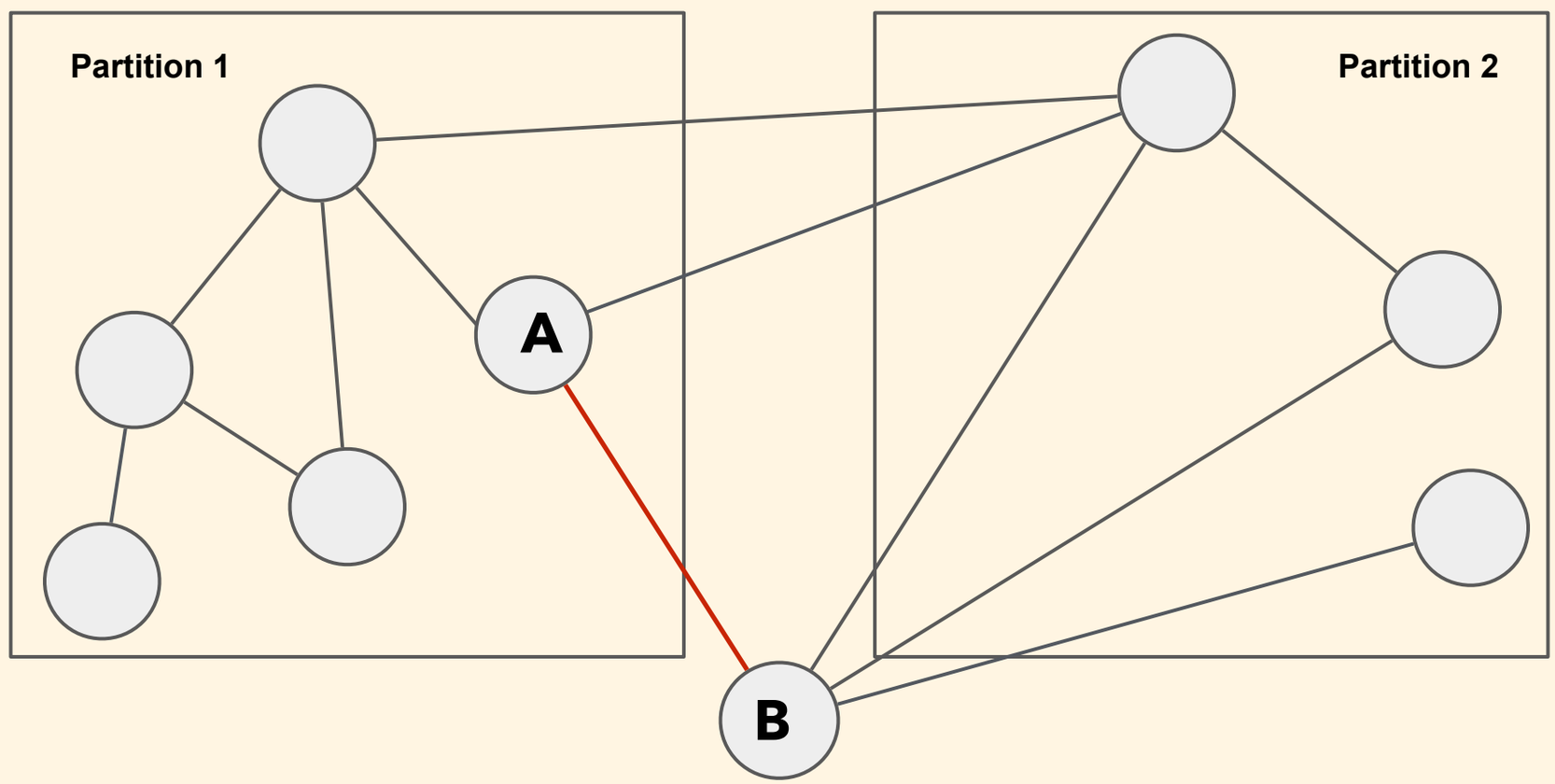


Candidate Set: {A, B}

Vertex Reassignment (Final)

neighbors: 1
vertices: 5
score: 1 * (1 - 5/6) = 0.17

neighbors: 3
vertices: 5
score: 3 * (1 - 3/6) = 1.5



Goals: (1) few cuts and (2) balanced

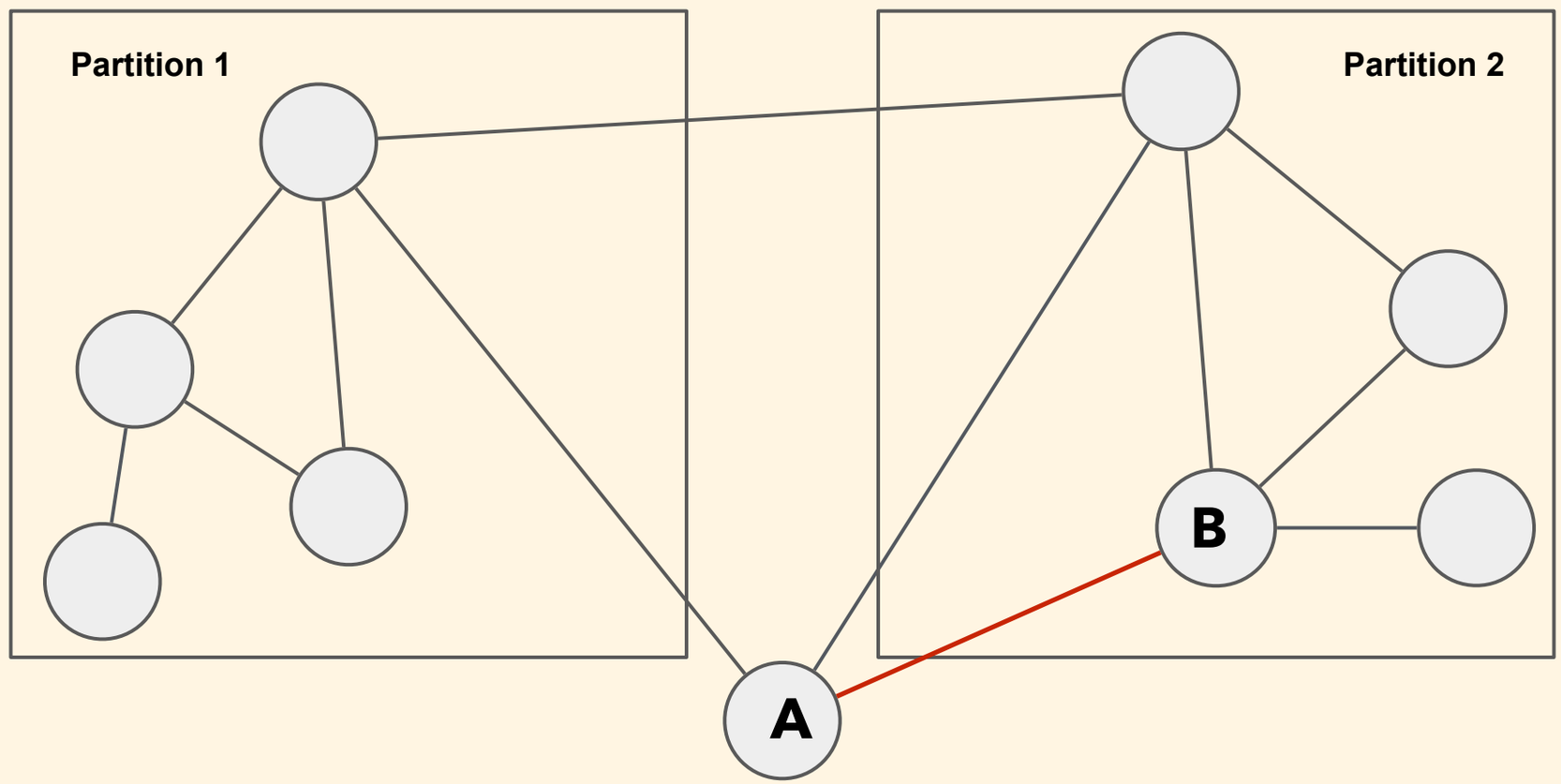
Objective Function: # neighbors * (1 - #vertices/capacity)

Partition Capacity: 6

Vertex Reassignment (Final)

neighbors: 1
vertices: 4
score: 1 * (1 - 4/6) = 0.33

neighbors: 2
vertices: 4
score: 2 * (1 - 4/6) = 0.66

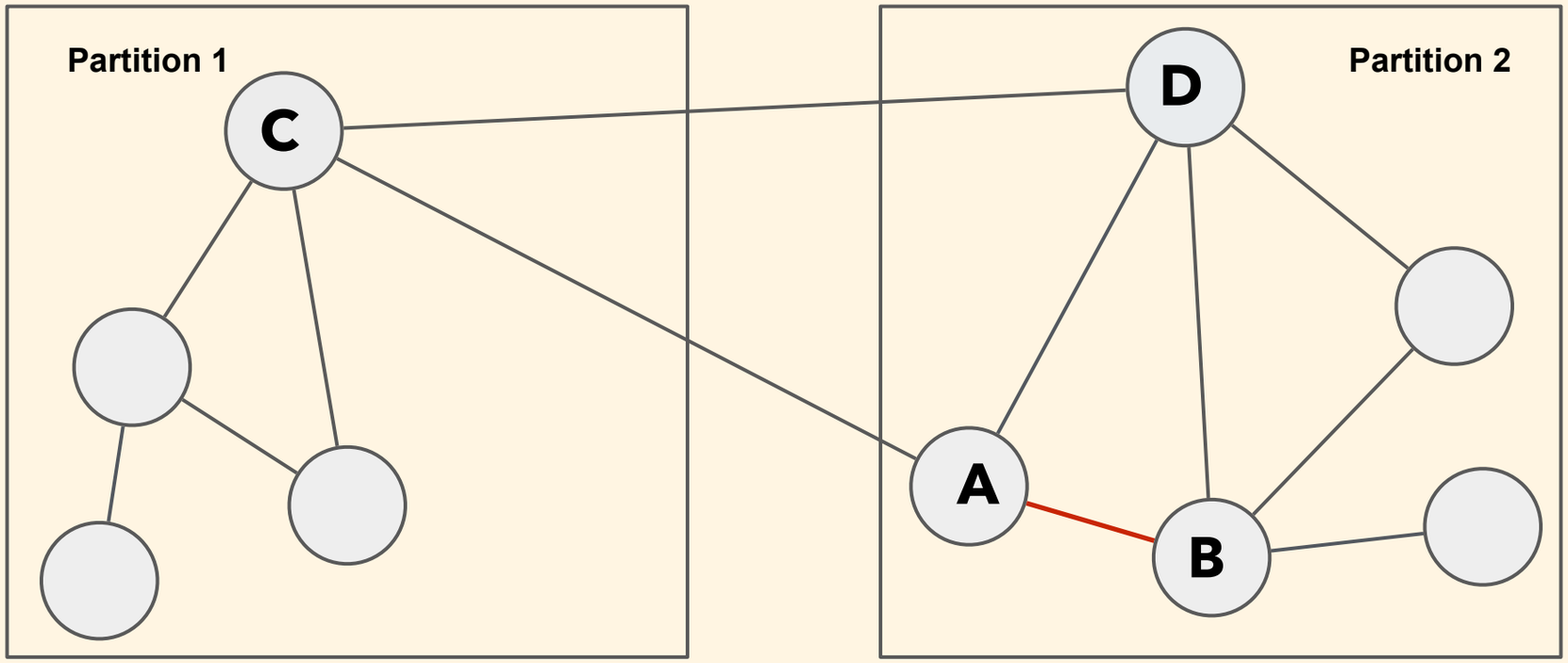


Goals: (1) few cuts and (2) balanced

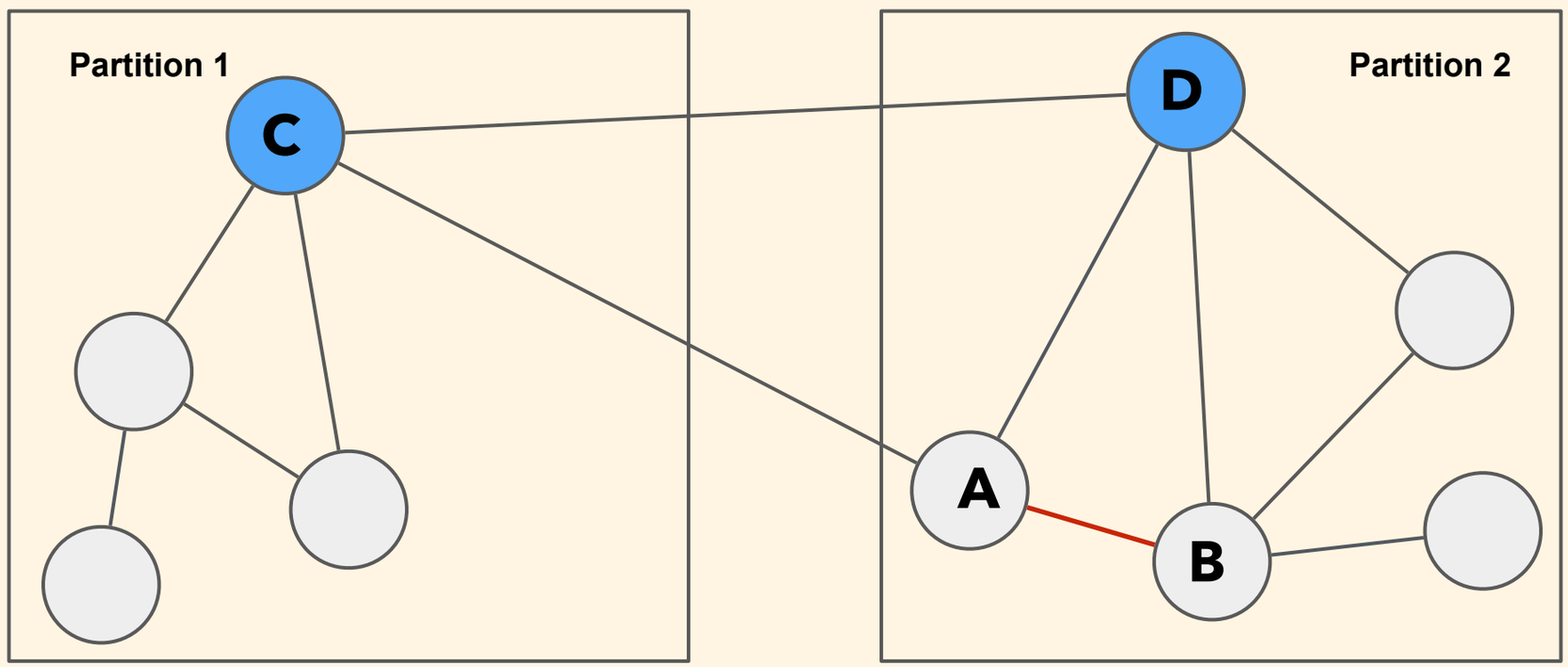
Objective Function: # neighbors * (1 - #vertices/capacity)

Partition Capacity: 6

Vertex Reassignment (Final)



Vertex Reassignment (Final)



Candidate Set: {C, D}

Leopard

- Algorithm Overview
  - Vertex Assignment
  - Vertex Reassignment
  - Computation Skipping
- Integration with Replication

Replication

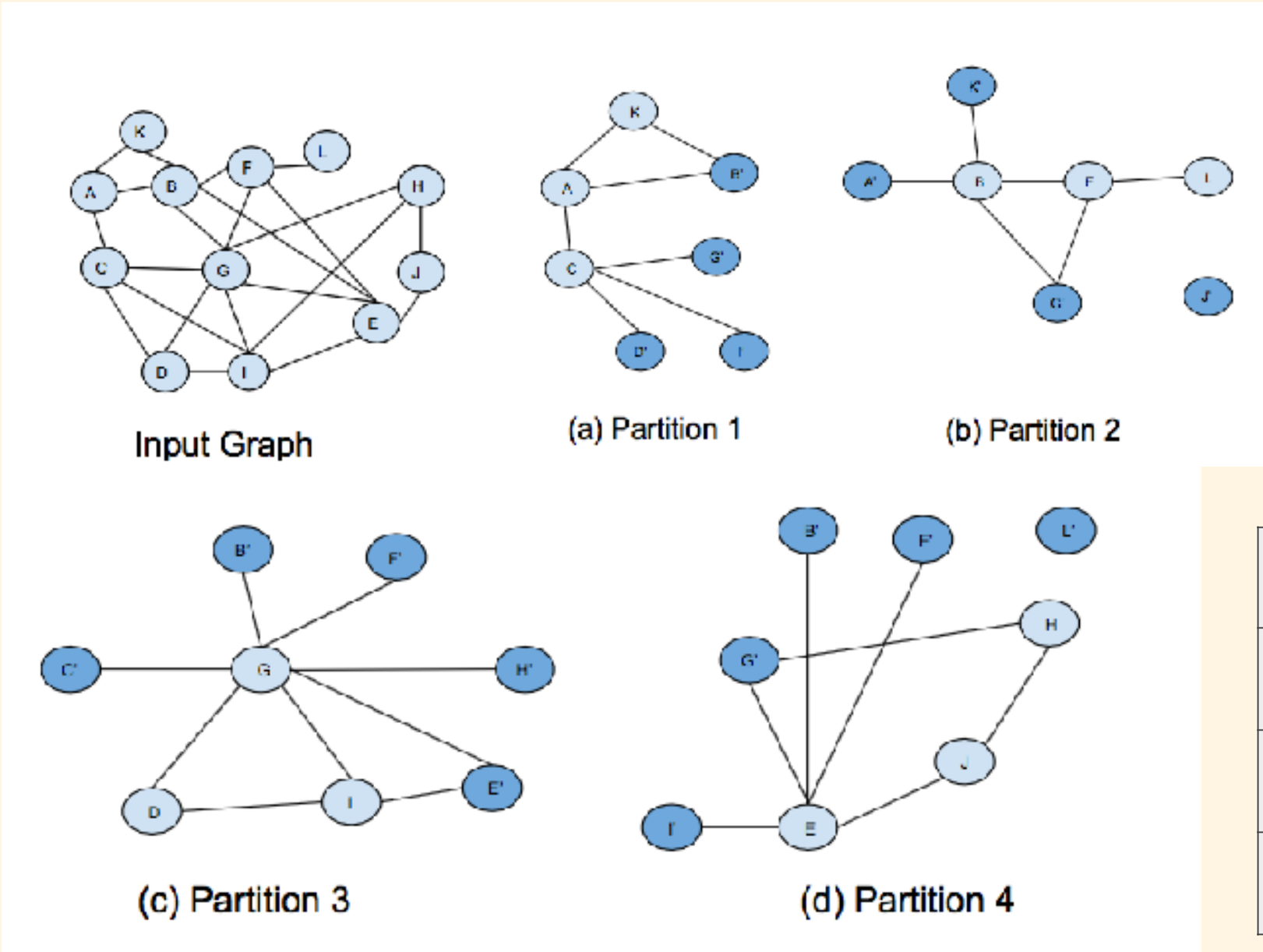
- Fault tolerance
- Access locality

Minimum-Average Replication

- Take two parameters: minimum and average number of copies
- Minimum: fault tolerance
- Average: additional access locality
- Decide the number and location of copies

Minimum-Average Replication

Min: 2
avg: 2.5

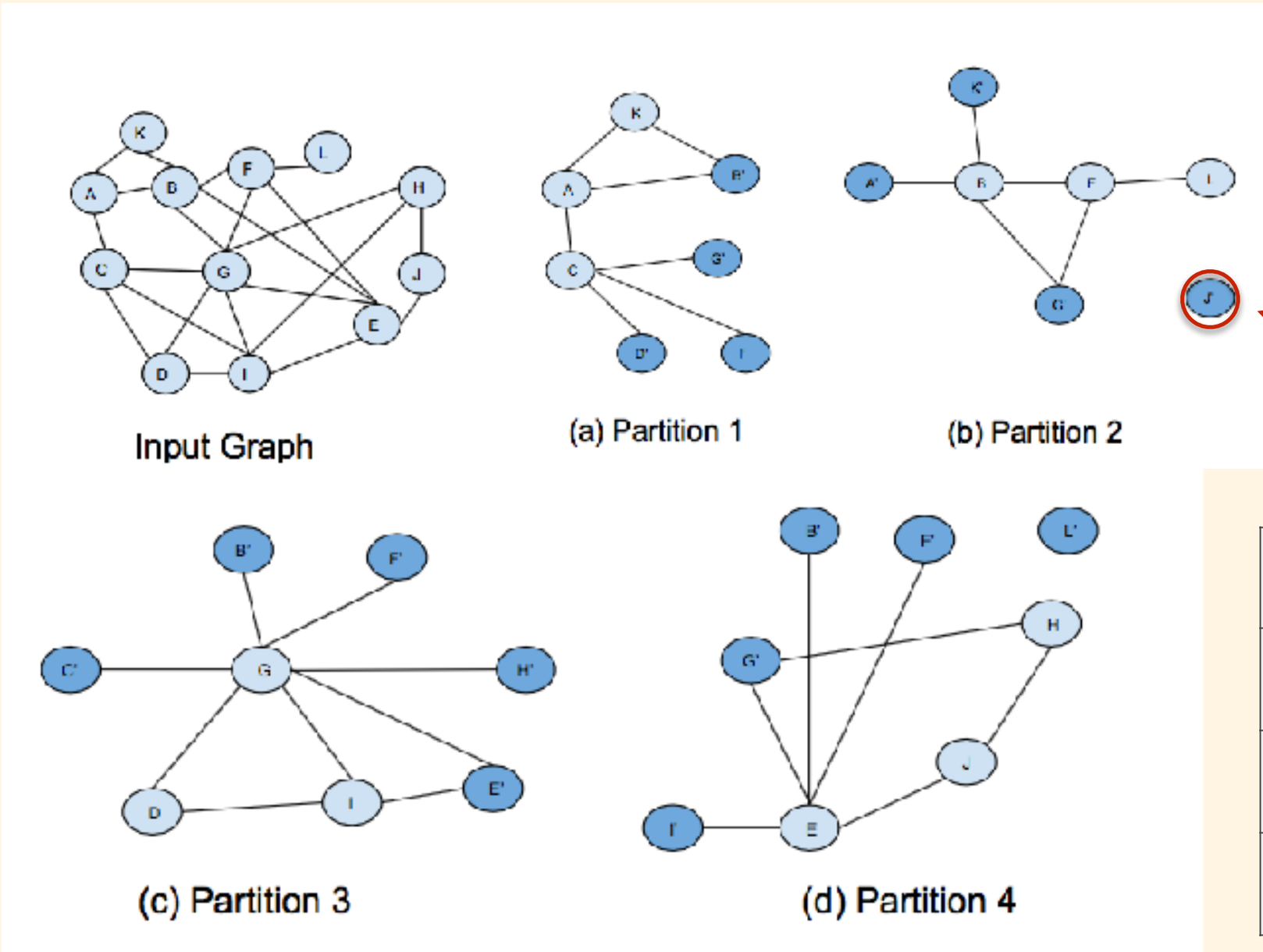


# copies	vertices
2	A,C,D,E,H,J,K,L
3	F,I
4	B,G

example taken from the paper

Minimum-Average Replication

Min: 2
avg: 2.5



only fault tolerance

# copies	vertices
2	A,C,D,E,H,J,K,L
3	F,I
4	B,G

Recall: Objective Function

FENNEL Scoring Function:

$$\underset{1 \leq i \leq k}{\operatorname{argmax}} \{ |N(v) \cap P_i| - \alpha \frac{\gamma}{2} (|P_i|)^{\gamma - 1} \}$$

Modified Vertex Assignment

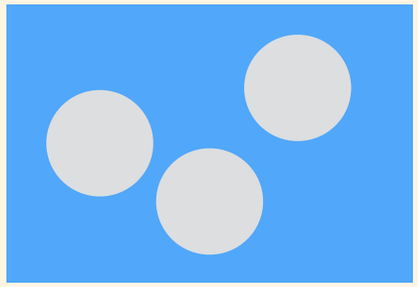
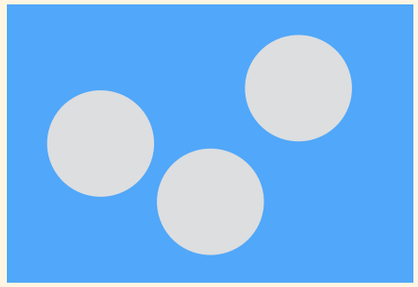
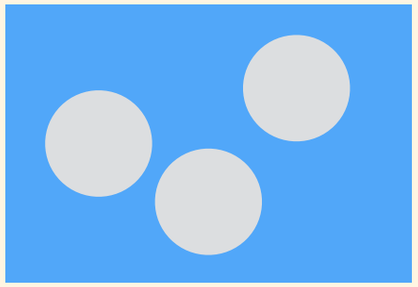
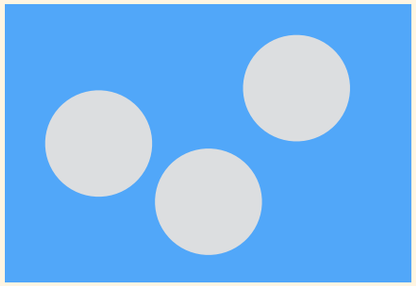
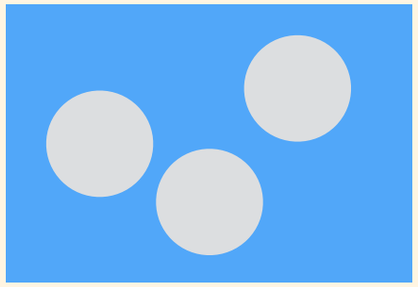
- def: primary copy of v = p(v)

  secondary copy of v = s(v)

- p(v): u is v's neighbor; s(u) or p(u) is in the partition
- s(v): u is v's neighbor; p(u) is in the partition
- Assign two scores for each partition, one for the p(v) and one for s(v)

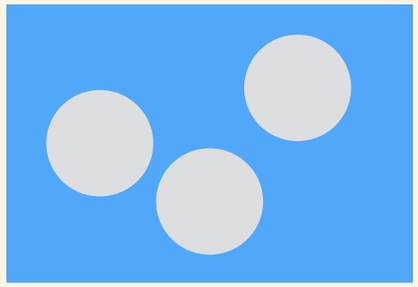
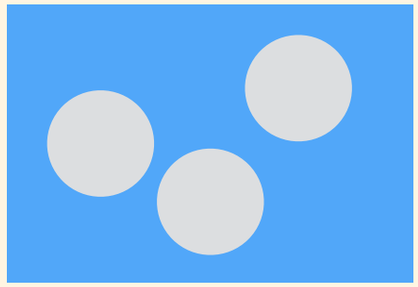
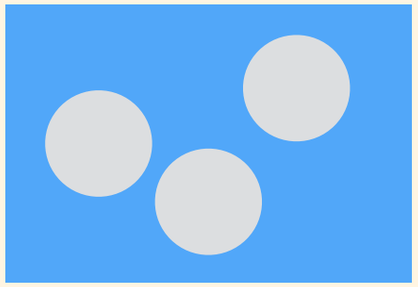
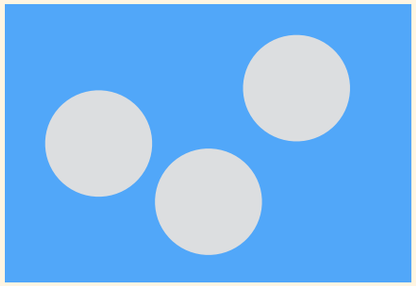
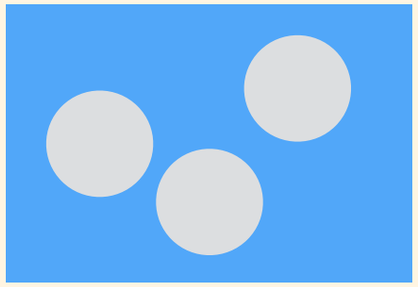
Minimum-Average Replication

min: 2
avg: 3

	Partition 1	Partition 2	Partition 3	Partition 4	Partition 5
					
primary:	0.15	0.25	0.35	0.45	0.55
secondary:	0.1	0.2	0.3	0.4	X

Minimum-Average Replication

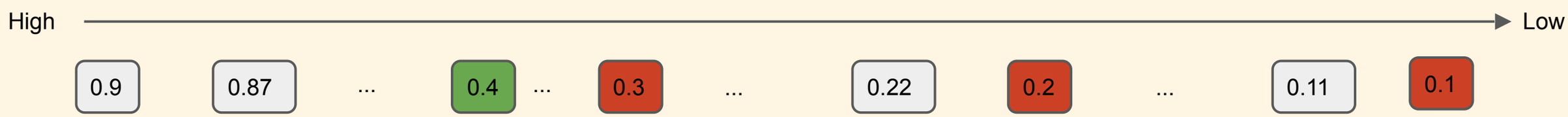
min: 2
avg: 3

	Partition 1	Partition 2	Partition 3	Partition 4	Partition 5
					
primary:	0.15	0.25	0.35	0.45	0.55
secondary:	0.1	0.2	0.3	0.4	X

Minimum-Average Replication

min: 2
avg: 3

keep the last n sorted computed scores

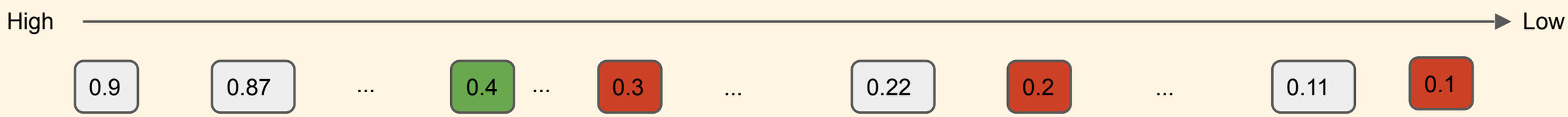


cutoff: top avg-1 / k-1 = 2 / 4 = 50% of scores
k: # partitions

Minimum-Average Replication

min: 2
avg: 3

keep the last n sorted computed scores

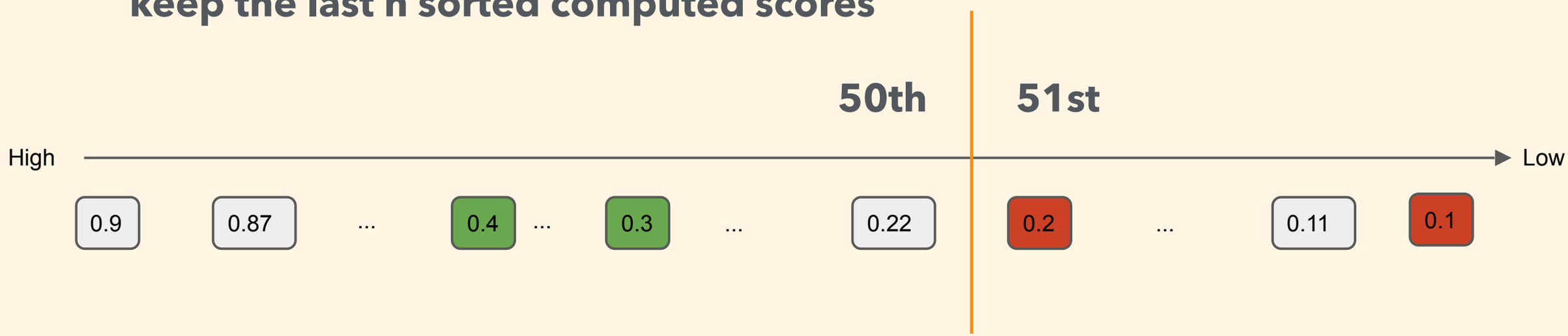


cutoff: 50th highest score

Minimum-Average Replication

min: 2
avg: 3

keep the last n sorted computed scores

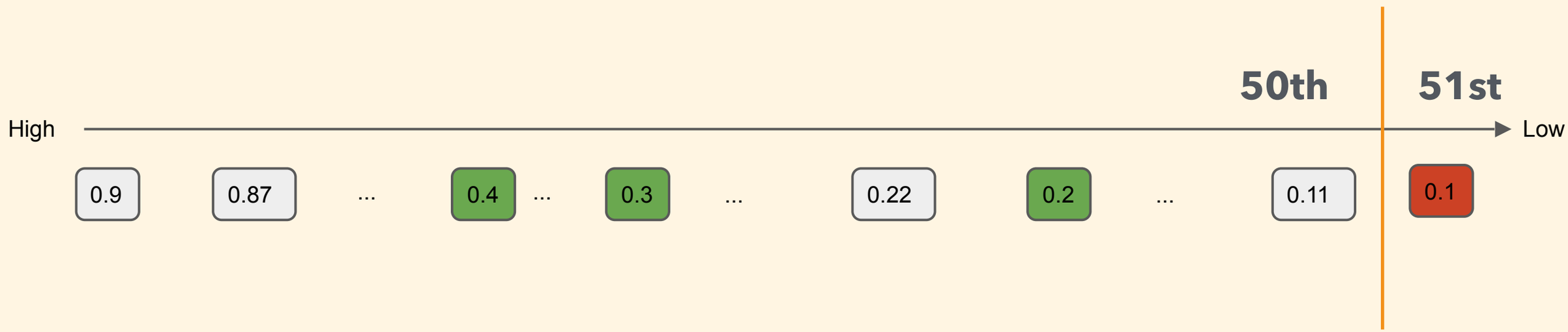


cutoff: 50th highest score

Minimum-Average Replication

min: 2
avg: 3

keep the last n sorted computed scores



cutoff: 50th highest score

Evaluation

- Machine: 4th Generation Intel Core i5 and 16 GB memory
- Comparison Points:
  - Leopard (FENNEL as objective function)
  - One-pass FENNEL (no vertex reassignment)
  - METIS (static graphs)
  - ParMetis (repartitioning for dynamic graphs)
  - Hash Partitioning

Evaluation

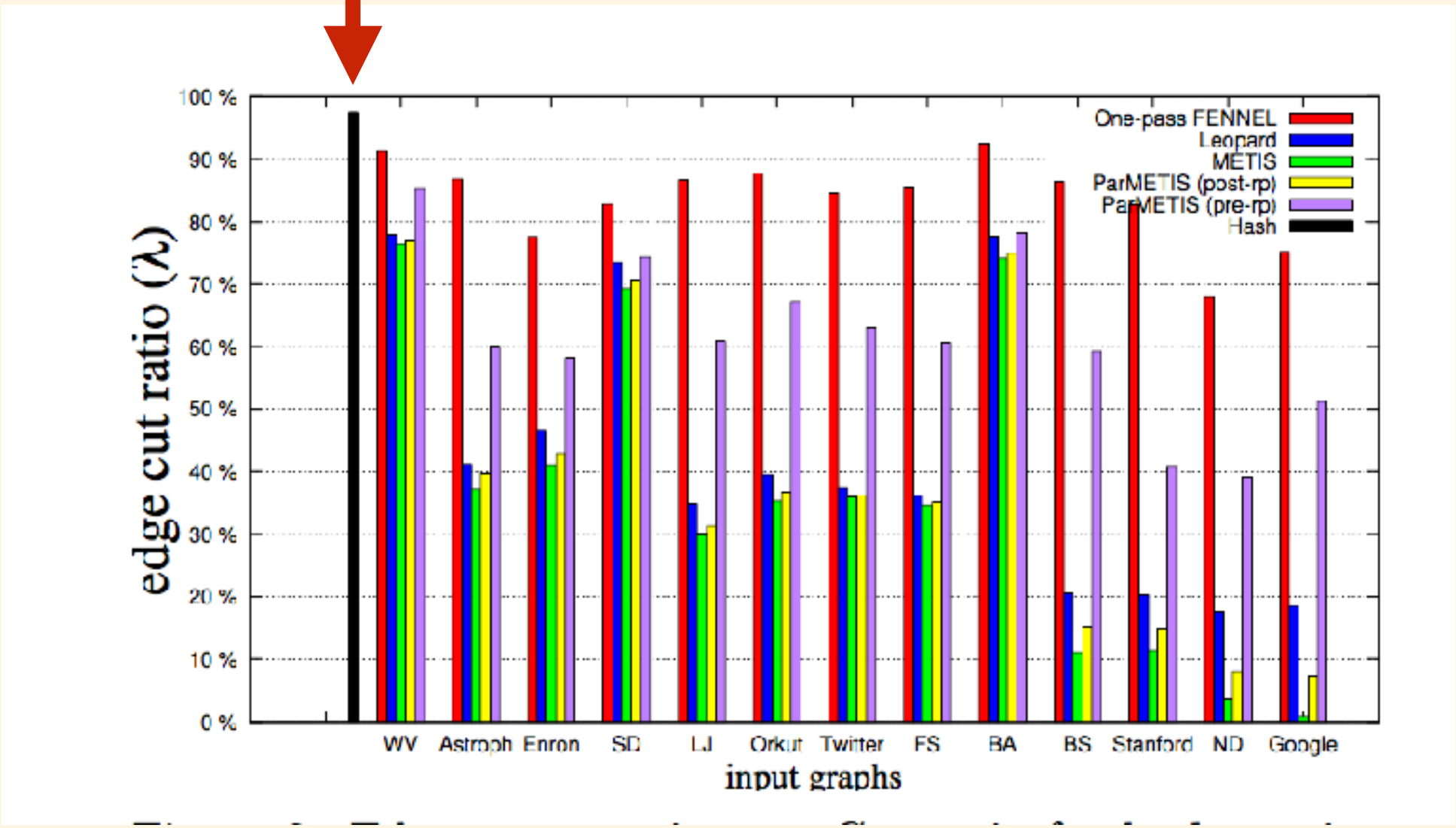
- Partition graphs into 40 partitions
- Parameters: $\gamma = 1.5 \text{ and } \alpha = \frac{\sqrt{k}|E|}{|V|^{1.5}}$
- cut ratio: # edge cuts / # edges

Dataset

Graph	|V|	|E|	Density	Clustering Coef.	Diameter [20]	Type
Wiki-Vote (WV)	7,115	100,762	$3.9 * 10^{-3}$	0.1409	3.8	Social
Astroph	18,771	198,050	$1.1 * 10^{-3}$	0.6306	5.0	Citation
Enron	36,692	183,831	$2.7 * 10^{-4}$	0.4970	4.8	Email
Slashdot (SD)	77,360	469,180	$1.6 * 10^{-4}$	0.0555	4.7	Social
NotreDame (ND)	325,729	1,090,108	$2.1 * 10^{-5}$	0.2346	9.4	Web
Stanford	281,903	1,992,636	$5.0 * 10^{-5}$	0.5976	9.7	Web
BerkStan (BS)	685,230	6,649,470	$2.8 * 10^{-5}$	0.5967	9.9	Web
Google	875,713	4,322,051	$1.1 * 10^{-5}$	0.5143	8.1	Web
LiveJournal (LJ)	4,846,609	42,851,237	$3.7 * 10^{-6}$	0.2742	6.5	Social
Orkut	3,072,441	117,185,083	$2.5 * 10^{-5}$	0.1666	4.8	Social
BaraBasi-Albert graph (BA)	15,000,000	1,800,000,000	$1.6 * 10^{-5}$	0.2195	4.6	Synthetic
Twitter	41,652,230	1,468,365,182	$1.7 * 10^{-6}$	0.1734	4.8	Social
Friendster (FS)	65,608,366	1,806,067,135	$8.4 * 10^{-7}$	0.1623	5.8	Social

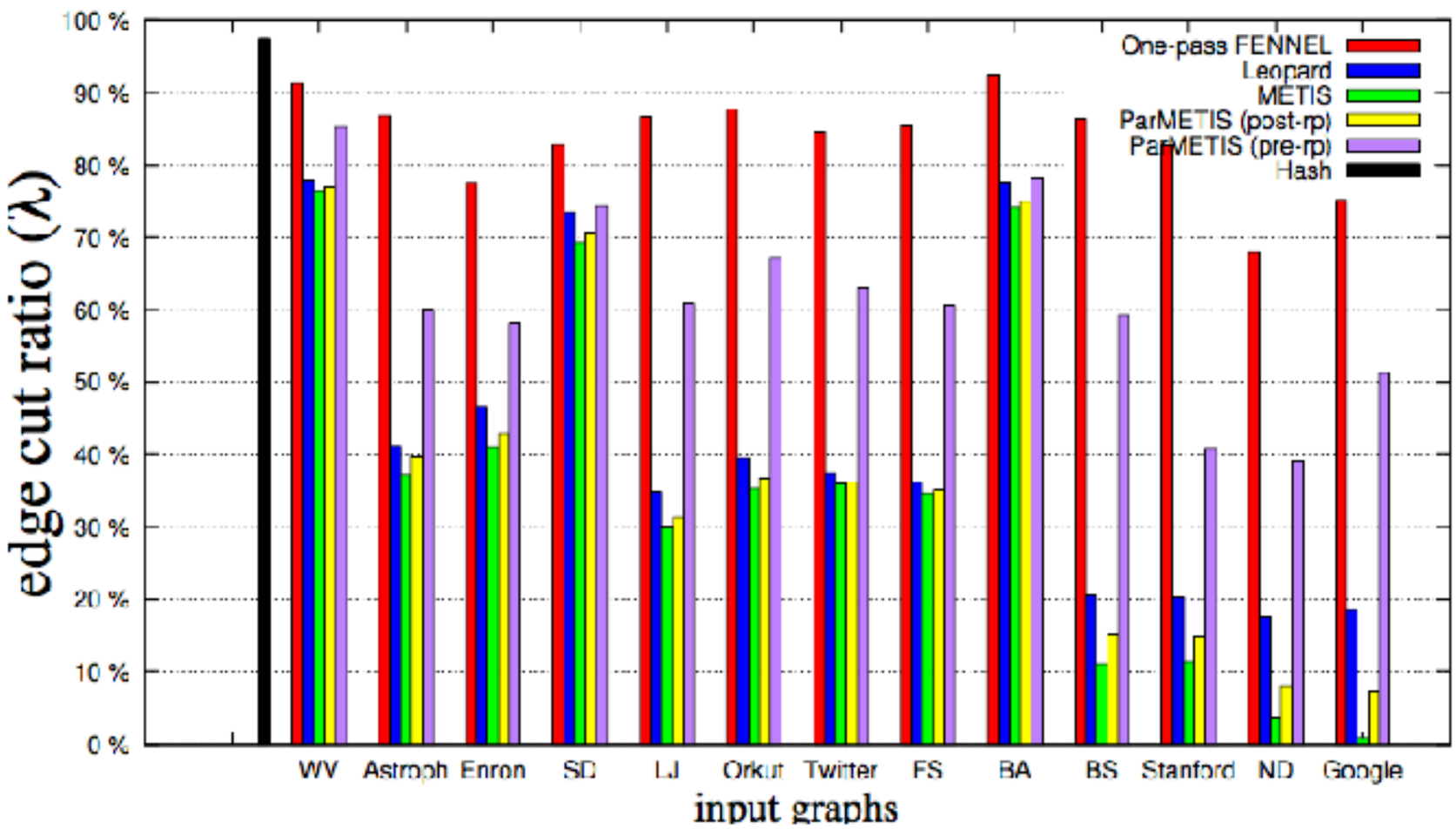
Figure 7: Statistics of the graphs used in the experiments. Diameter is reported at 90th-percentile to eliminate outliers.

Cut Ratio


100 %
90 %
80 %
70 %
60 %
50 %
40 %
30 %
20 %
10 %
0 %
edge cut ratio (λ)
One-pass FENNEL
Leopard
METIS
ParMETIS (post-rp)
ParMETIS (pre-rp)
Hash
WV
Astroph
Enron
SD
LJ
Orkut
Twitter
FS
BA
BS
Stanford
ND
Google
input graphs

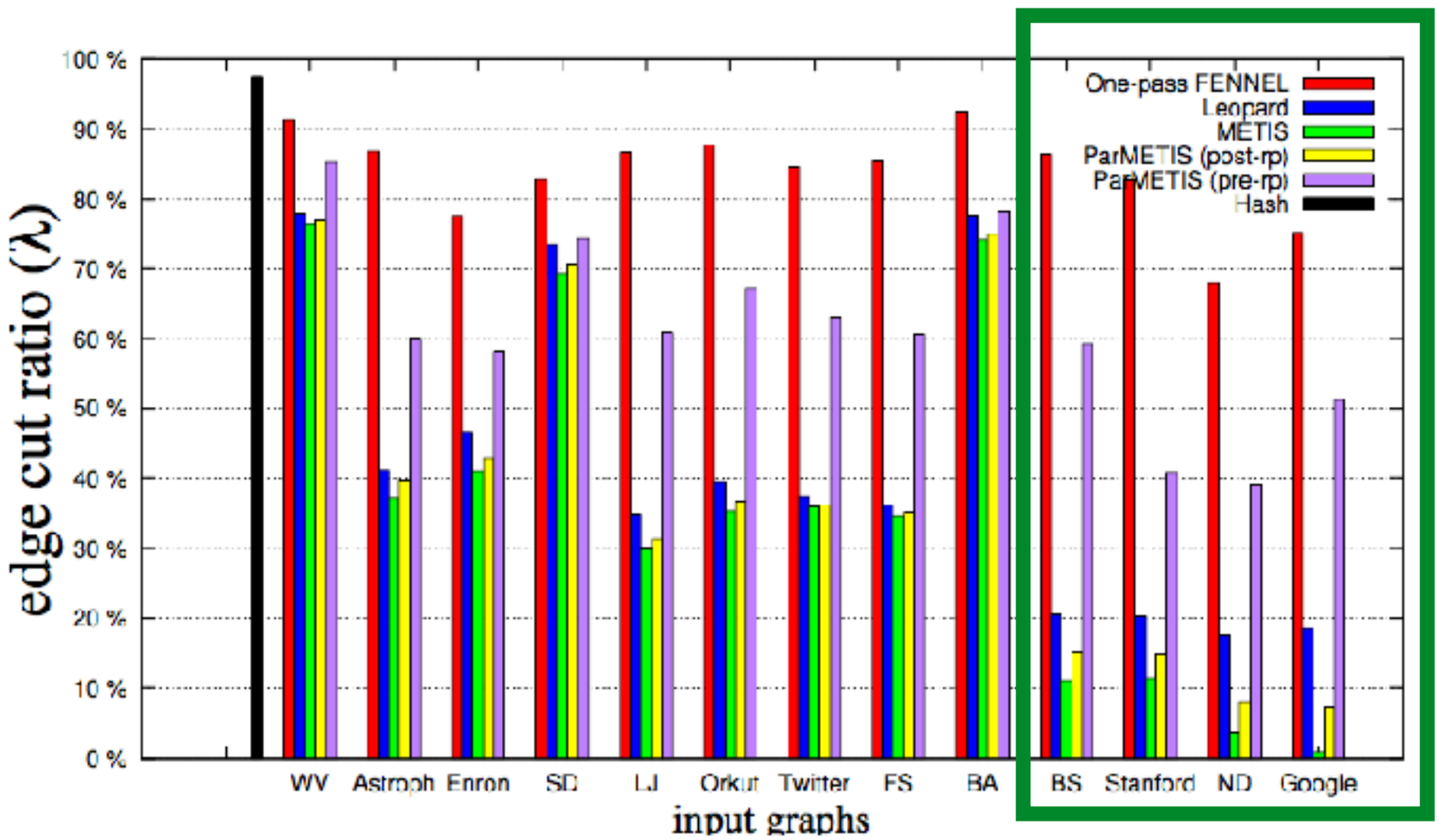
Cut Ratio

METIS: upper bound

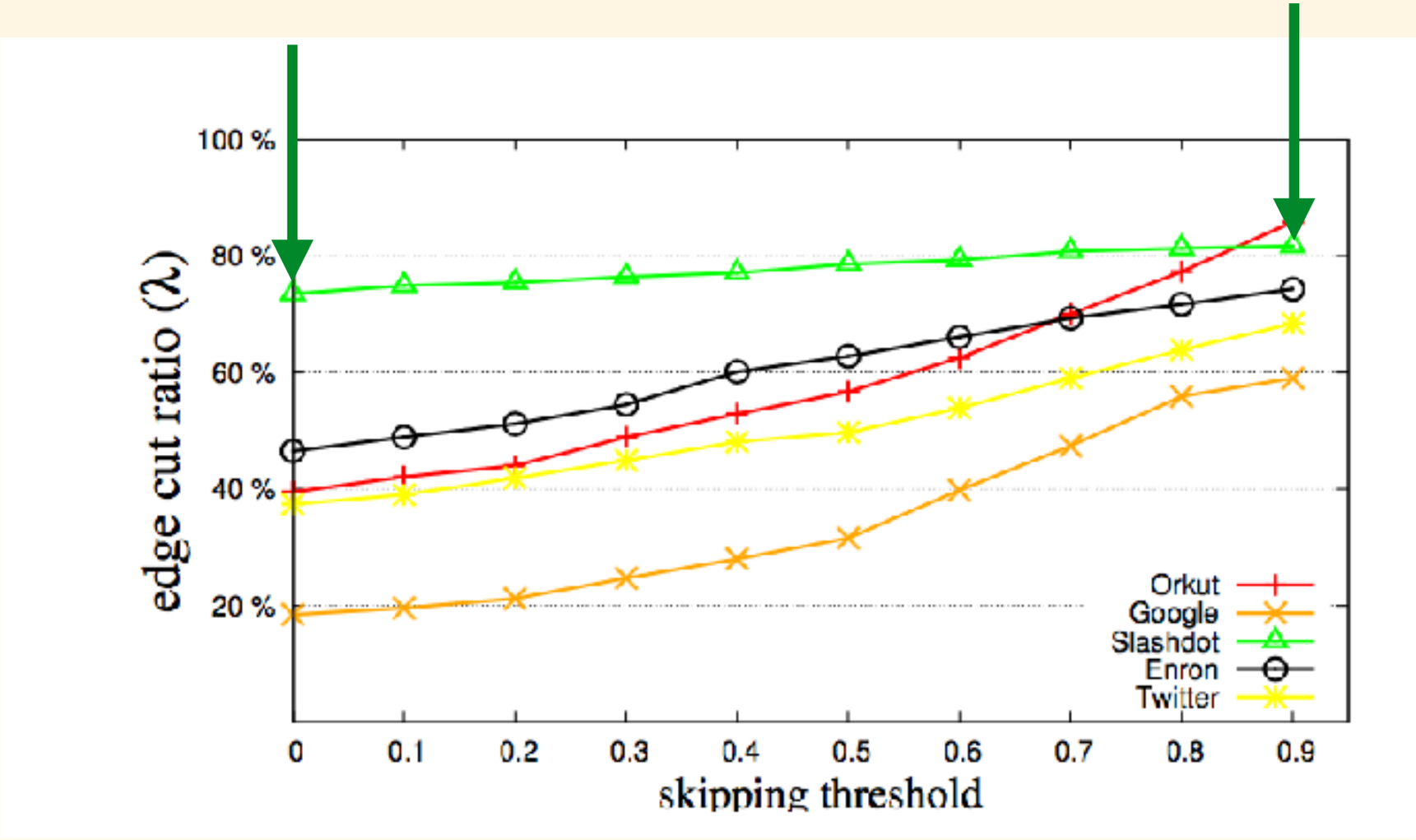


Cut Ratio

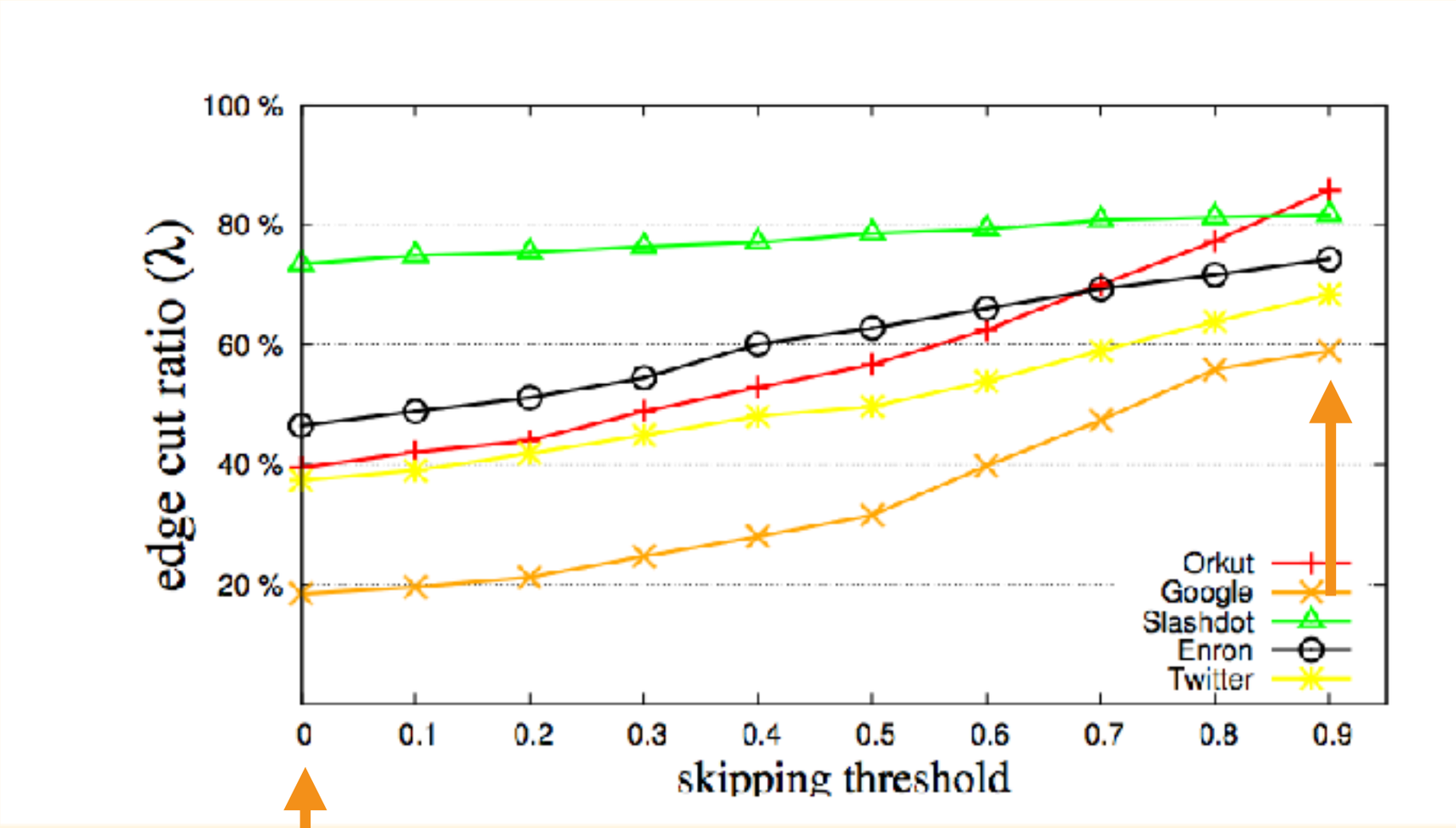
Web graphs



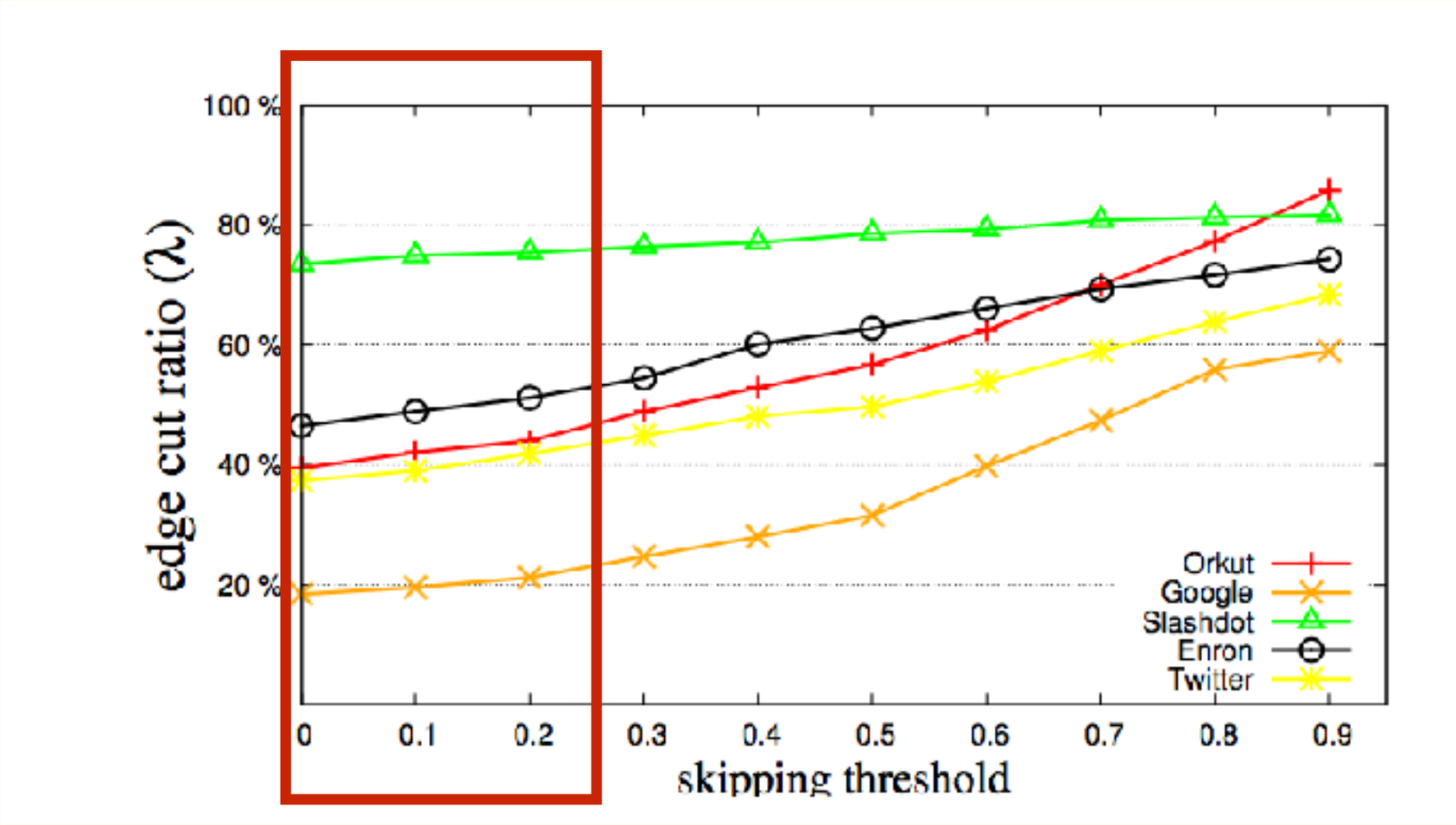
Computation Skipping



Computation Skipping

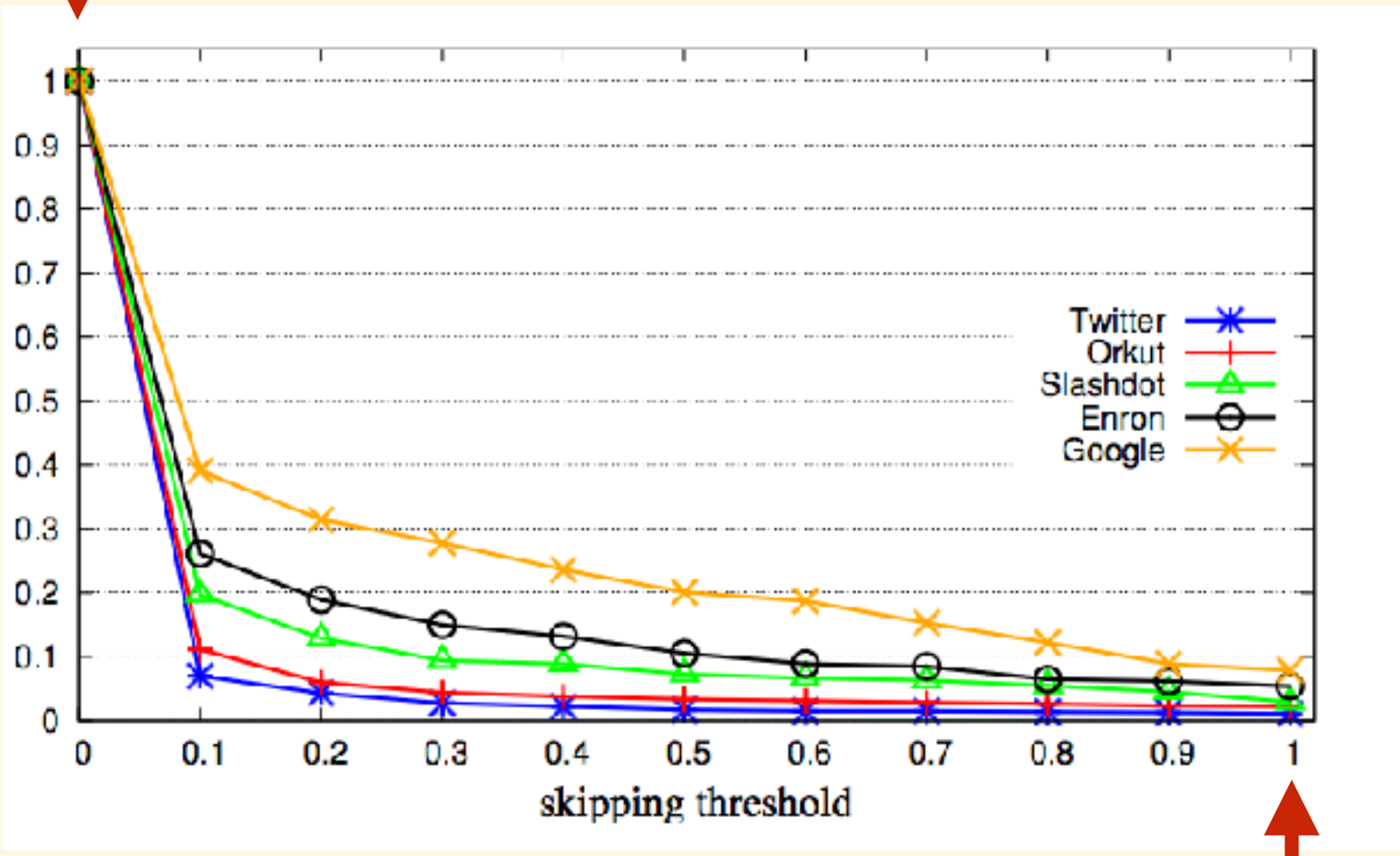


Computation Skipping



Computation Skipping

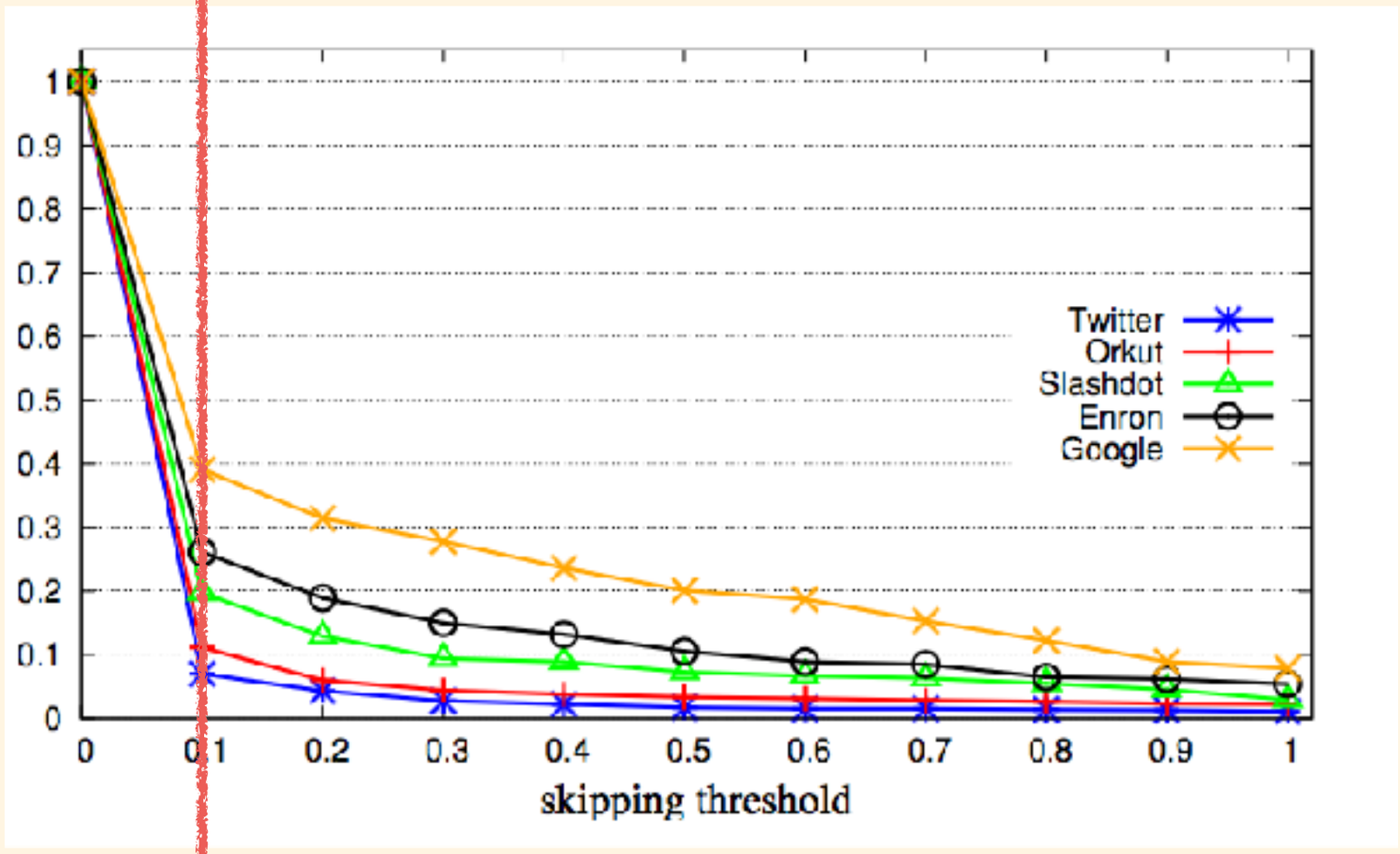
always examine



never examine

Computation Skipping

Sparse



Dense

Summary

- core idea: vertex assigned to partition with with most of its neighbors, large partition should be penalized to prevent it from becoming too large
- focuses on read-only workload
- first to integrate replication with partitioning
- perform poorly on web graph

Discussion

- How the presented approach is different from heuristic method and how well it performs in maintaining accuracy?
- The experiment and research are mainly for theoretical analysis like edge-cut ratio. Then how about the running time?
- How the solution presented (map vertices to machines in a distributed hash table) solves the fundamental problem with hash partitioning stated in the introduction?
- How are frequent updates to the skipped computation counter propagated in the presence of replication?