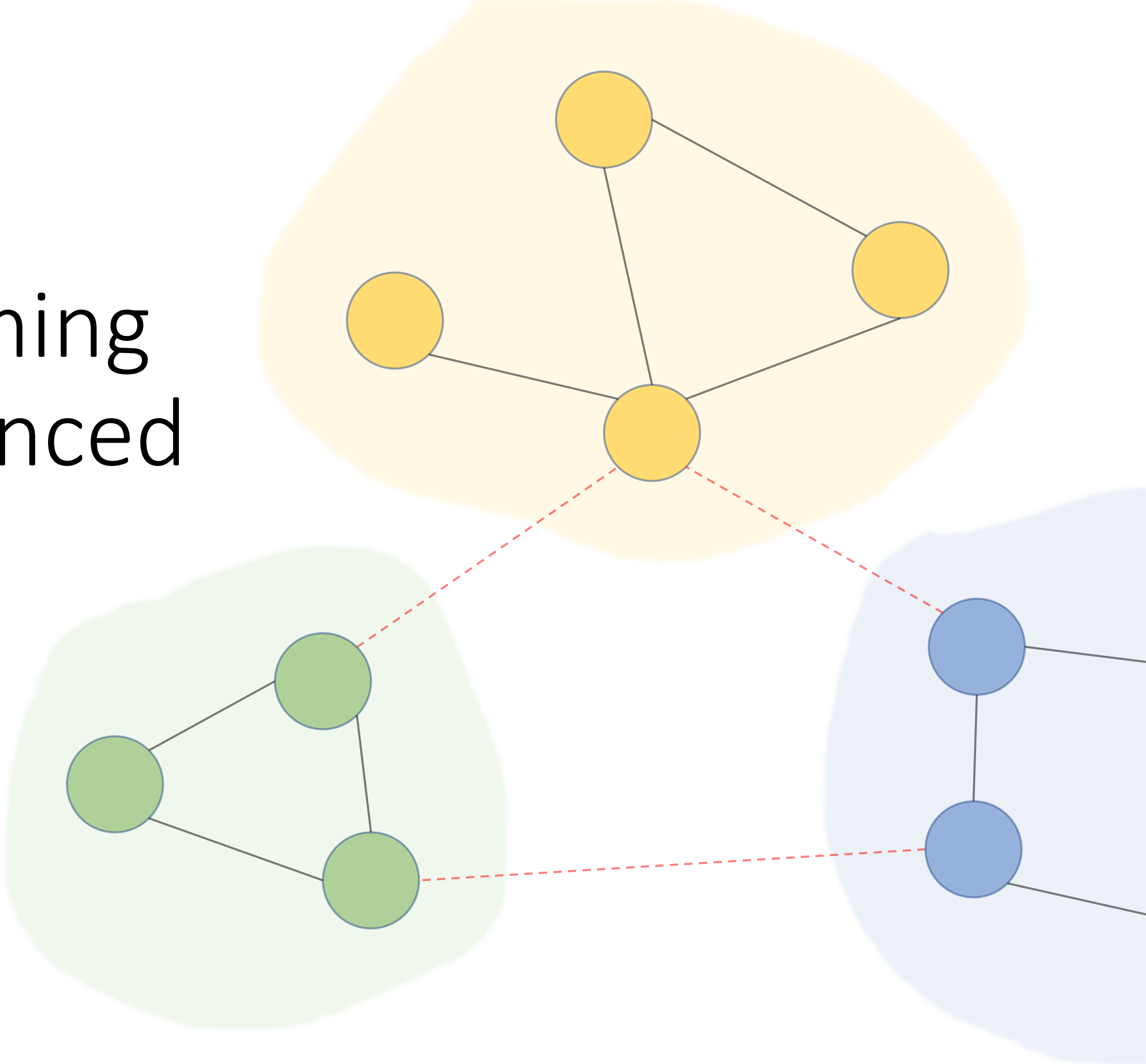
Prioritized Restreaming Algorithms for Balanced Graph Partitioning

Amel Awadelkarim

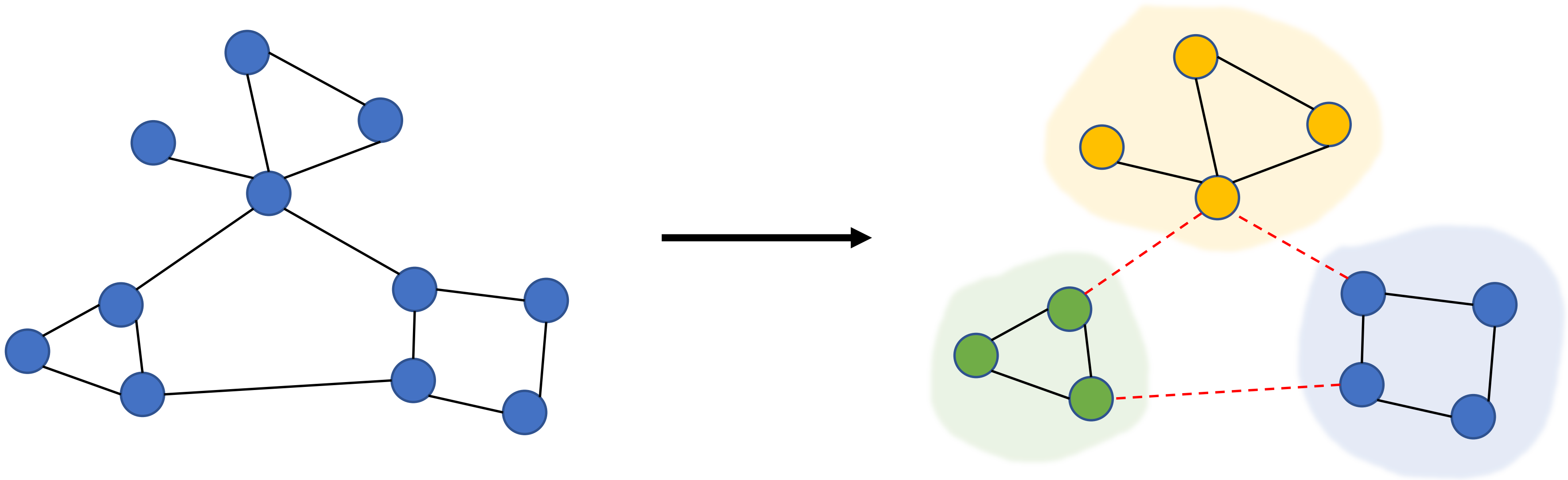
ameloa@stanford.edu

Johan Ugander

jugander@stanford.edu

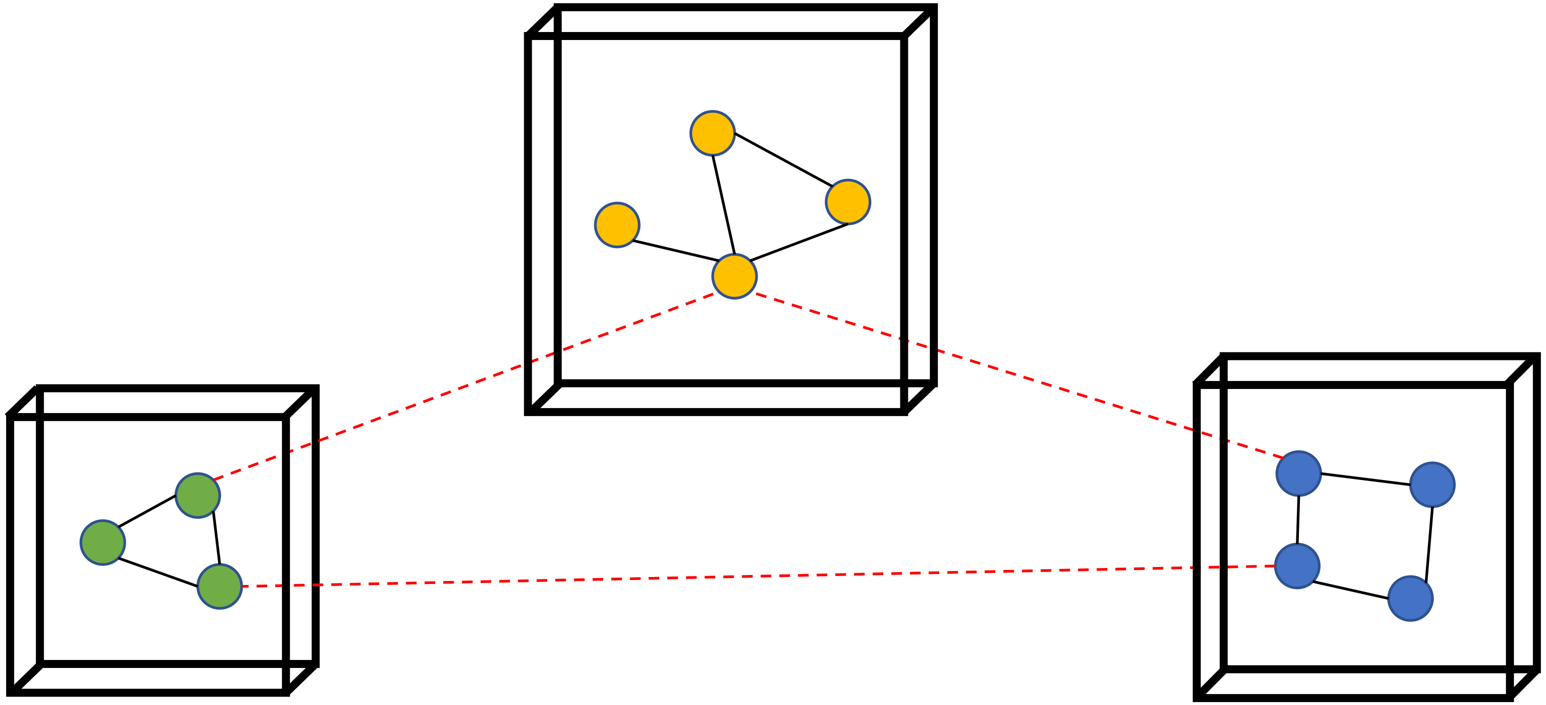


Balanced graph partitioning



We want to partition a graph into node-sets of approximately equal size, while minimizing the number of edges cut.

Balanced graph partitioning



This problem has practical application as an imperative step for large-scale distributed graph computation.

Existing algorithms

Global

Multilevel

Local

The exact solution is infeasible to compute, hence we focus on iterative local heuristics.

Existing algorithms

Global

Multilevel

Local

The exact solution is infeasible to compute, hence we focus on iterative local heuristics.

A new class of algorithms

Global

Multilevel

Local

↳ Streaming

↳ **Prioritized**

Specifically, we explore the role of stream order in (re)streaming algorithms and introduce *prioritized* restreaming algorithms.

Contributions

1. A taxonomy of existing iterative techniques
2. Informative benchmarking that was absent from the literature
3. A paradigm shift in restreaming partitioning algorithms

Outline

Existing methods

Taxonomy

Prioritized restreaming

Results

- Benchmark existing methods
- Prioritized restreaming results
- Correlation between stream orders

Outline

Existing methods

Taxonomy

Prioritized restreaming

Results

- Benchmark existing methods
- Prioritized restreaming results
- Correlation between stream orders

Outline

Existing methods

Taxonomy

Prioritized restreaming

Results

- Benchmark existing methods
- Prioritized restreaming results
- Correlation between stream orders

Outline

Existing methods

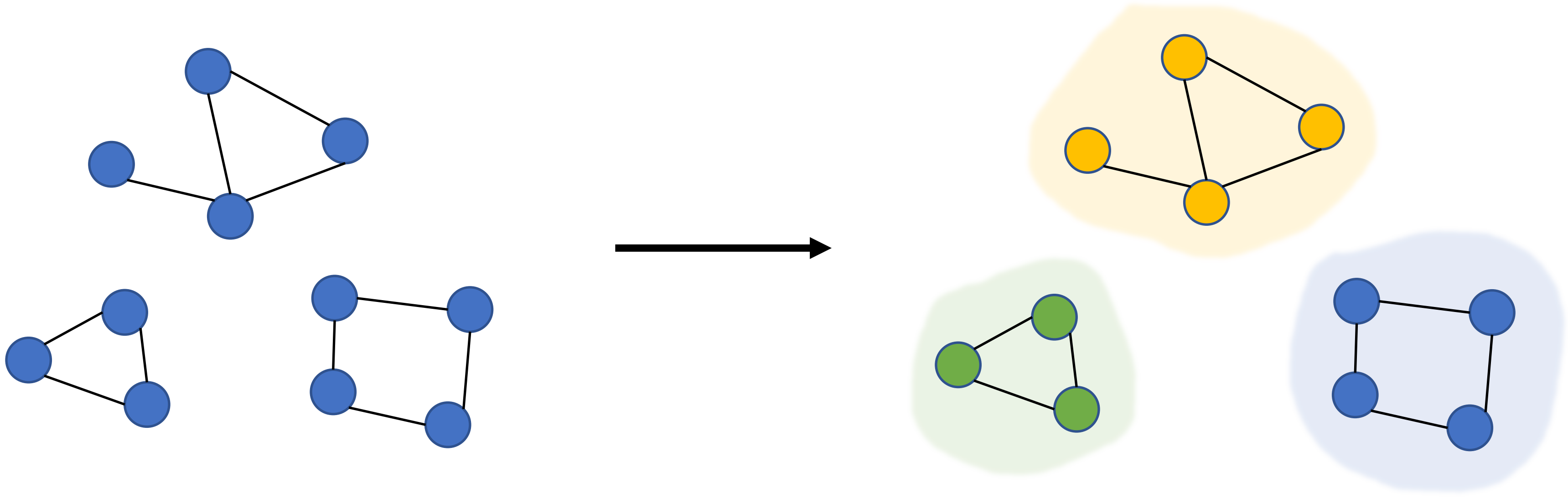
Taxonomy

Prioritized restreaming

Results

- Benchmark existing methods
- Prioritized restreaming results
- Correlation between stream orders

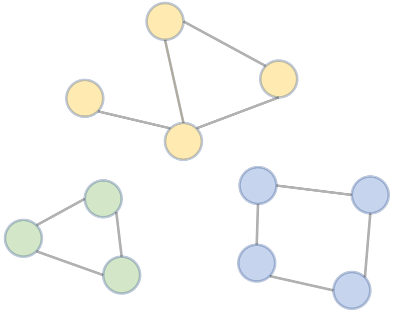
Existing methods

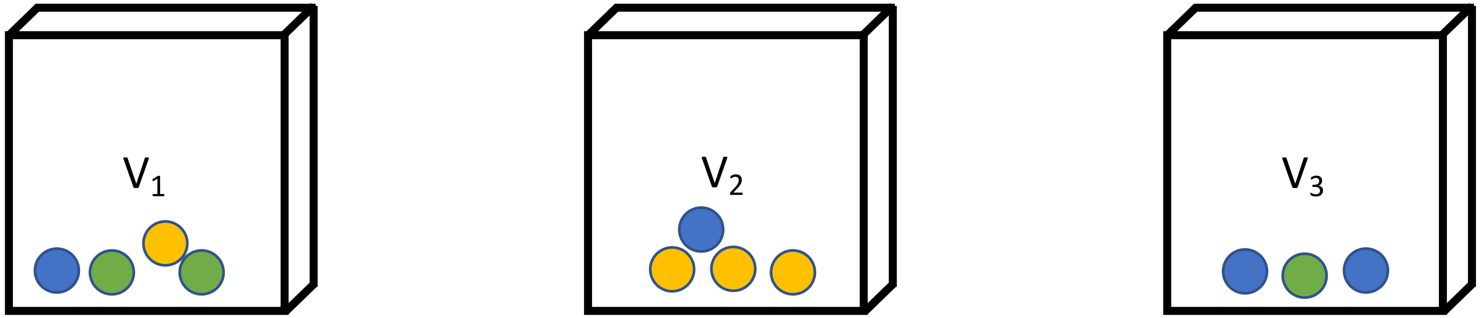


We present three algorithms from the literature – two based on label propagation and one restreaming algorithm.

Balanced label propagation

Ugander and Backstrom. *WSDM. 2013.*

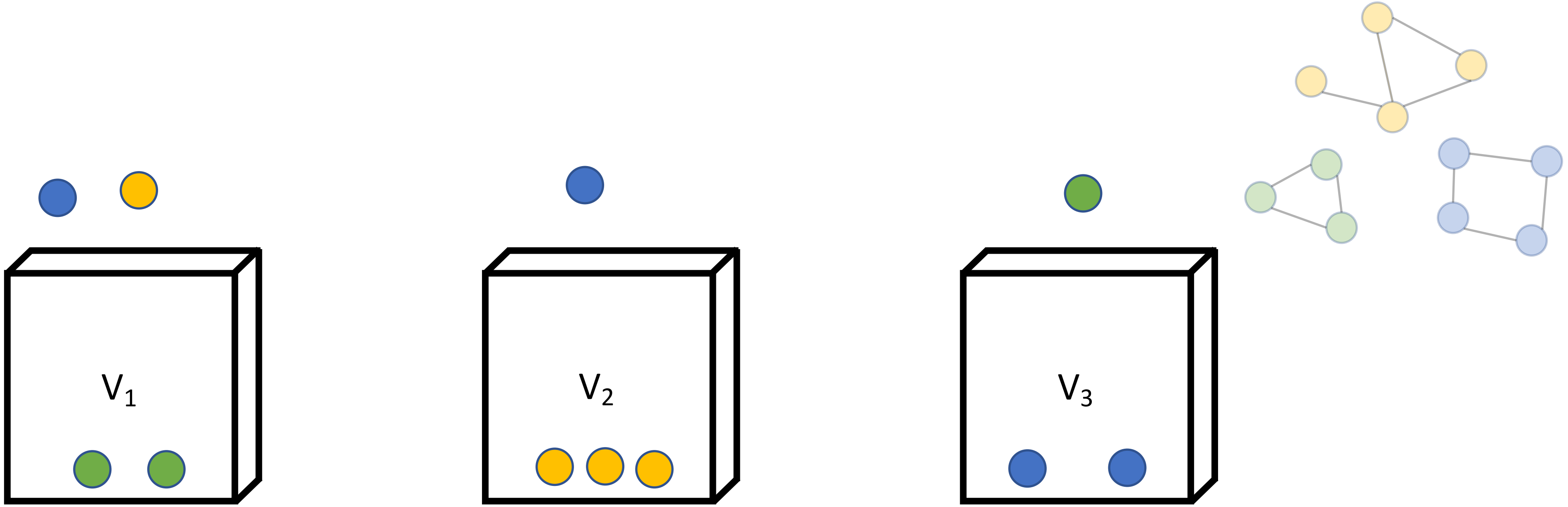




BLP begins from an initial partitioning, iteratively improving upon the edge cut objective.

Balanced label propagation

Ugander and Backstrom. *WSDM. 2013.*



At each iteration, BLP identifies which nodes desire to move and to where,

Balanced label propagation

Ugander and Backstrom. *WSDM. 2013.*

Gain of node *u*

$$g_u = \max_{i \in [k]} N_{u,i} - N_{u,P(u)}$$

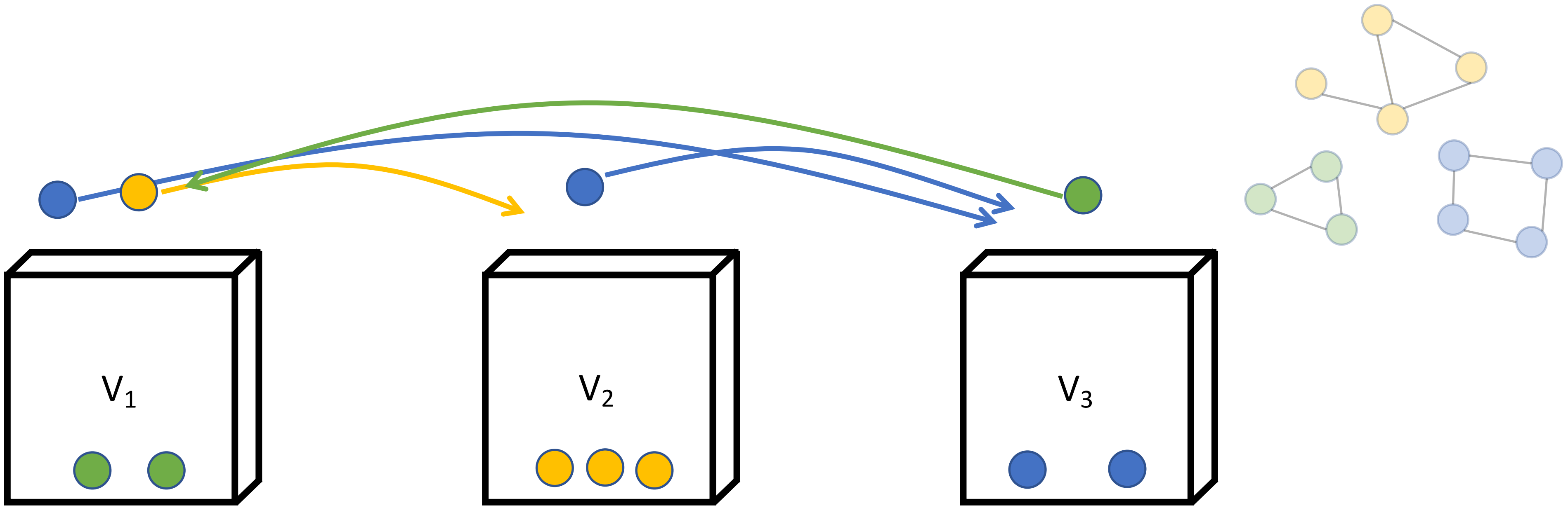
Neighbors in shard *i*

Neighbors in current shard assignment, *P(u)*

places nodes in sorted move queues to their target shards by order of decreasing *gain*,

Balanced label propagation

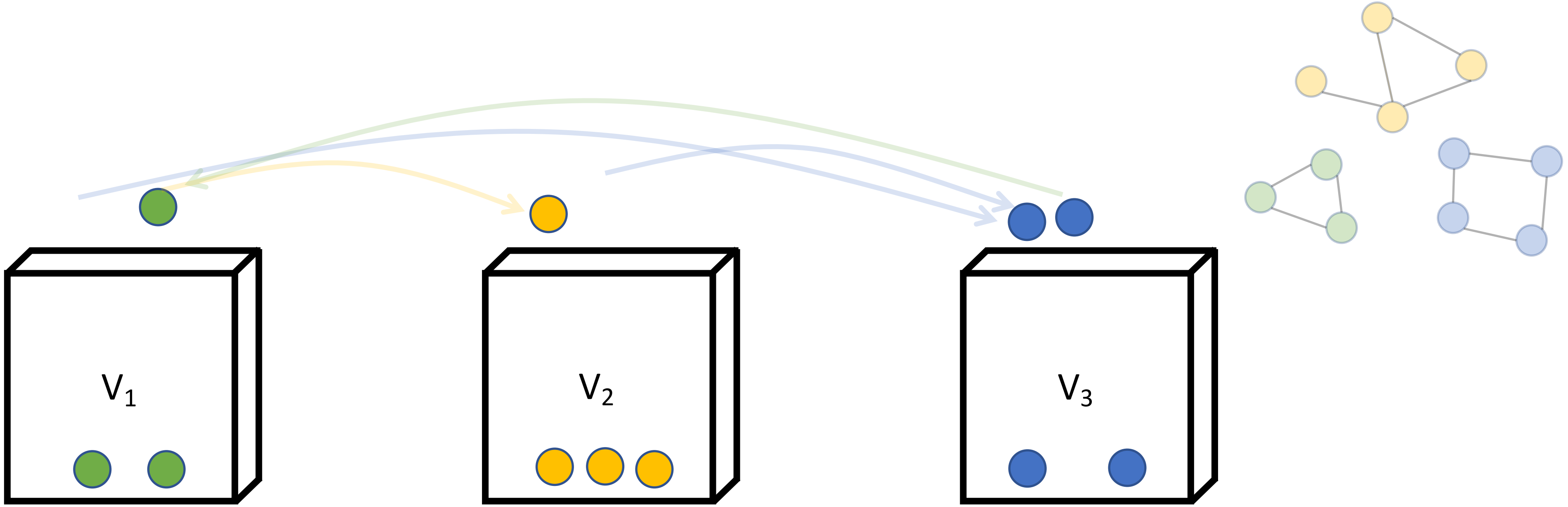
Ugander and Backstrom. *WSDM. 2013.*



then solves a linear program to determine how many top nodes to relocate.

Balanced label propagation

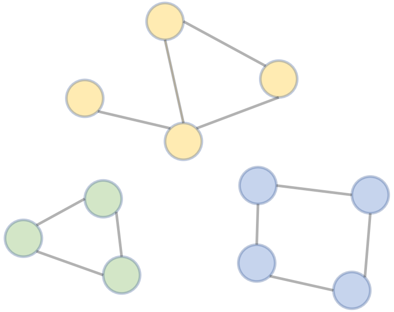
Ugander and Backstrom. *WSDM. 2013.*

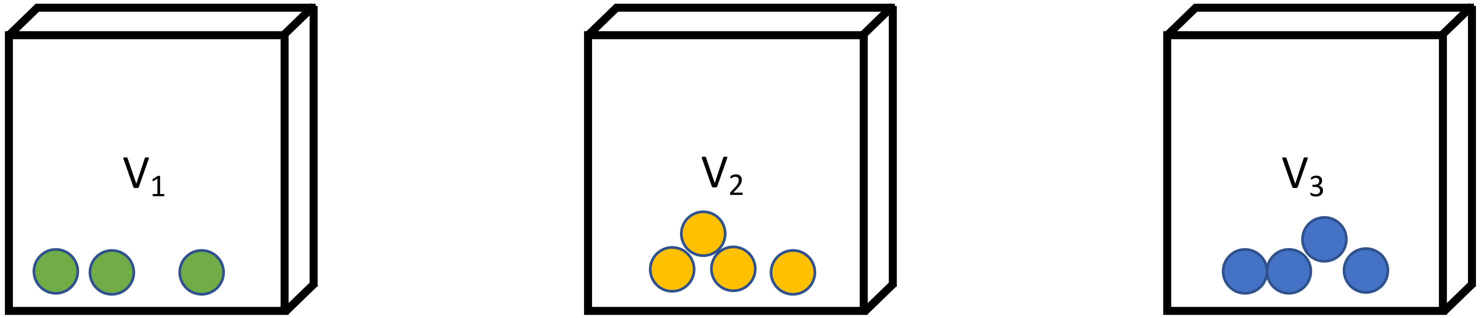


then solves a linear program to determine how many top nodes to relocate.

Balanced label propagation

Ugander and Backstrom. *WSDM. 2013.*

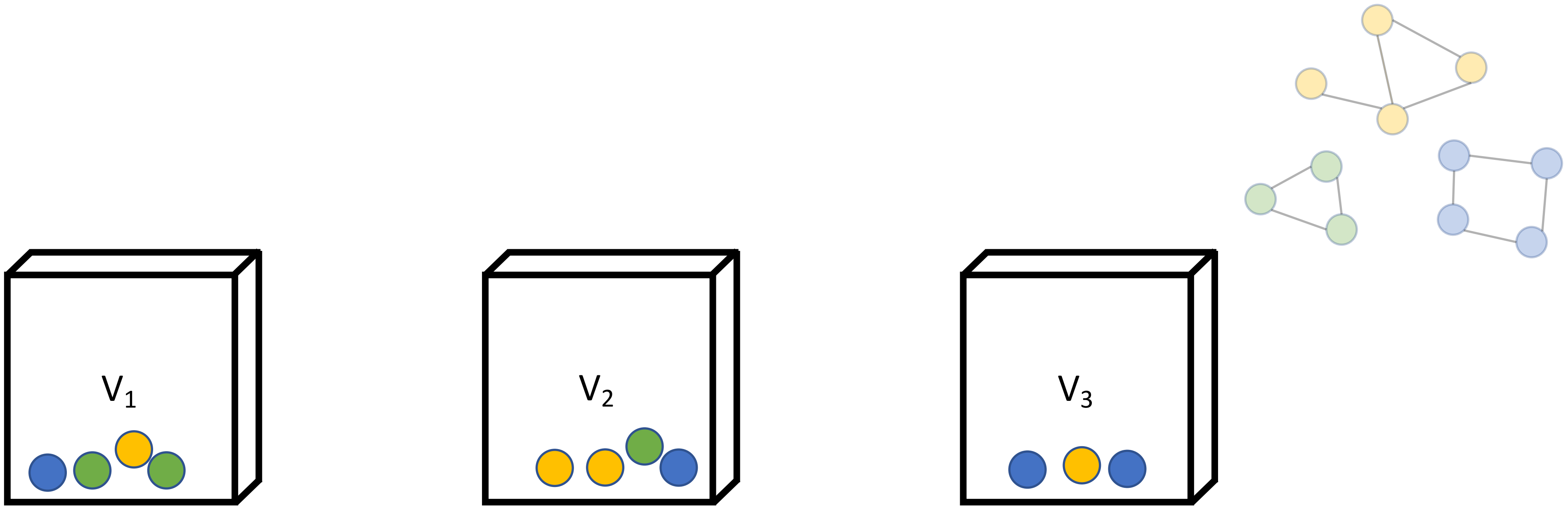




then solves a linear program to determine how many top nodes to relocate.

Social Hash partitioner

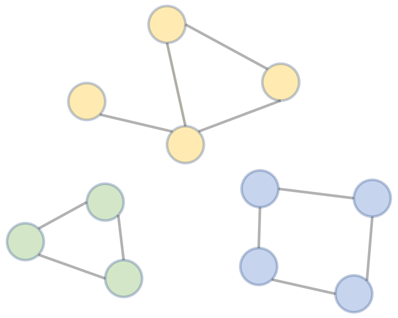
Kabiljo et al. *VLDB.* 2017.
Shalita et al. NSDI. 2016.

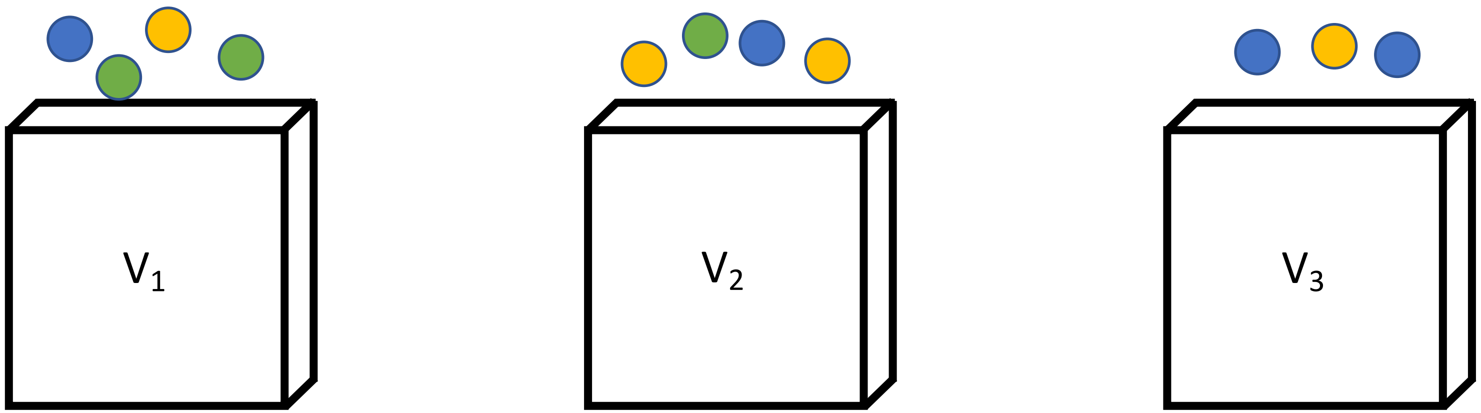


SHP also starts from an initial partitioning.

Social Hash partitioner

Kabiljo et al. *VLDB.* 2017.
Shalita et al. NSDI. 2016.





At each iteration, we place *all* nodes in the move queue of the shard that maximizes a modified form of gain,

Social Hash partitioner

Kabiljo et al. *VLDB.* 2017.
Shalita et al. NSDI. 2016.

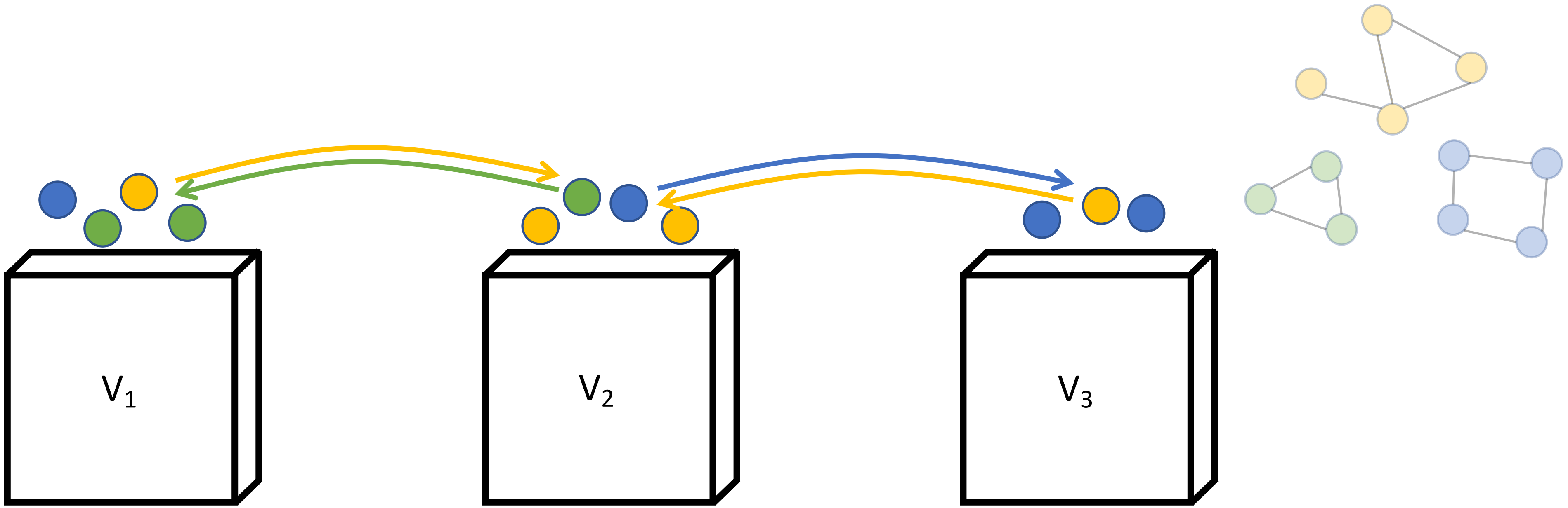
$$g'_u = \max_{i \in [k] \setminus P(u)} N_{u,i} - N_{u,P(u)}$$

Max over external shards

the max gain *outside* of a node's current shard assignment, and sort move queues by this quantity.

Social Hash partitioner

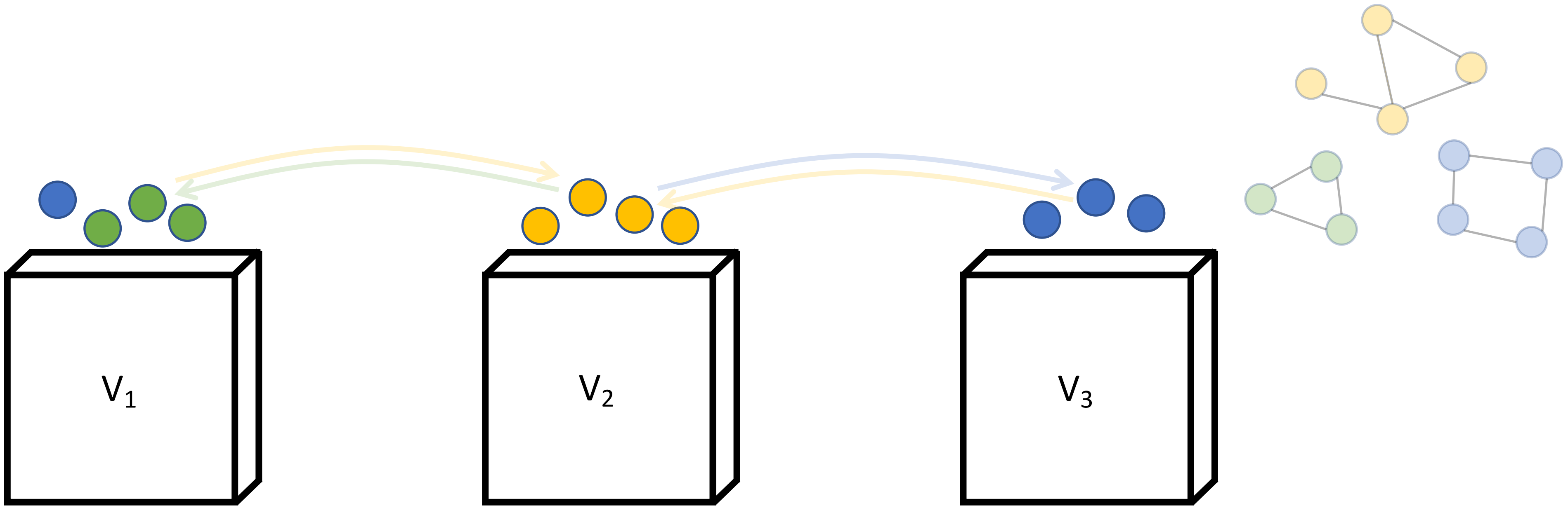
Kabiljo et al. *VLDB.* 2017.
Shalita et al. NSDI. 2016.



Balance is maintained by *swapping* nodes between shard pairs, only doing so when the net gain is positive.

Social Hash partitioner

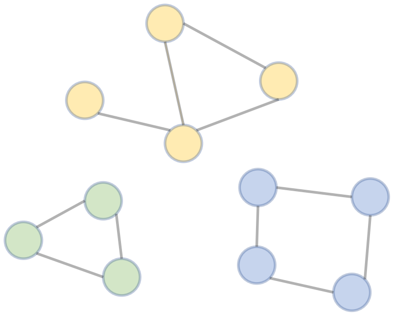
Kabiljo et al. *VLDB.* 2017.
Shalita et al. NSDI. 2016.

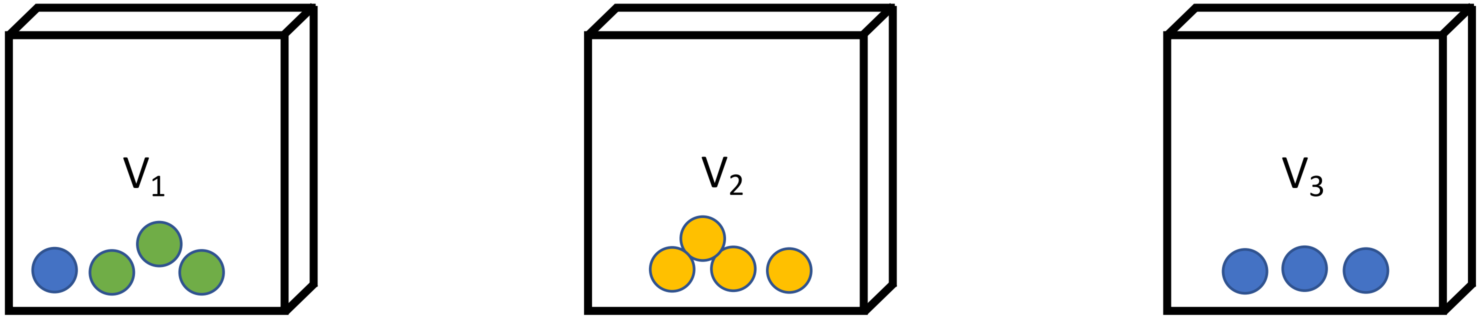


Balance is maintained by *swapping* nodes between shard pairs, only doing so when the net gain is positive.

Social Hash partitioner

Kabiljo et al. *VLDB.* 2017.
Shalita et al. NSDI. 2016.

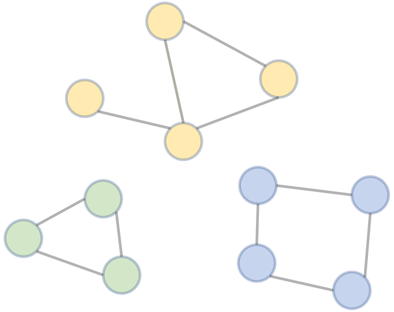


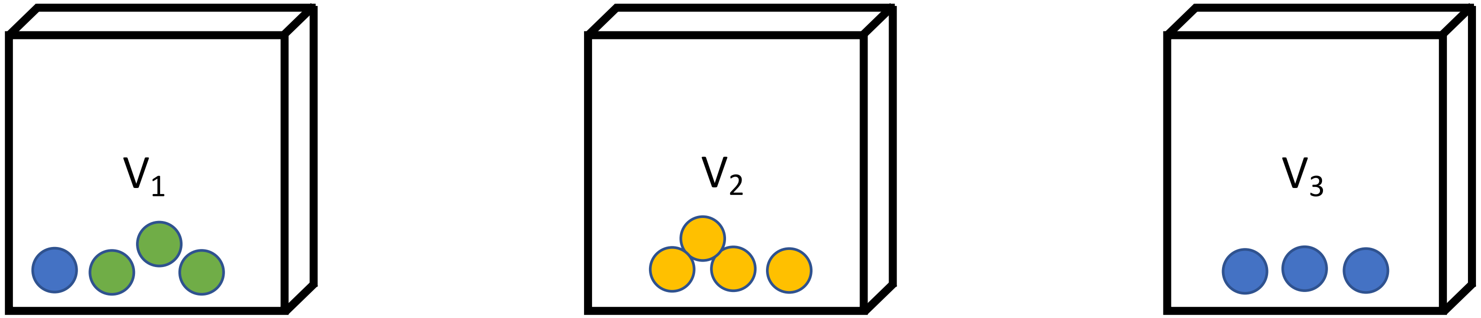


Balance is maintained by *swapping* nodes between shard pairs, only doing so when the net gain is positive.

Social Hash partitioner

Kabiljo et al. *VLDB.* 2017.
Shalita et al. NSDI. 2016.

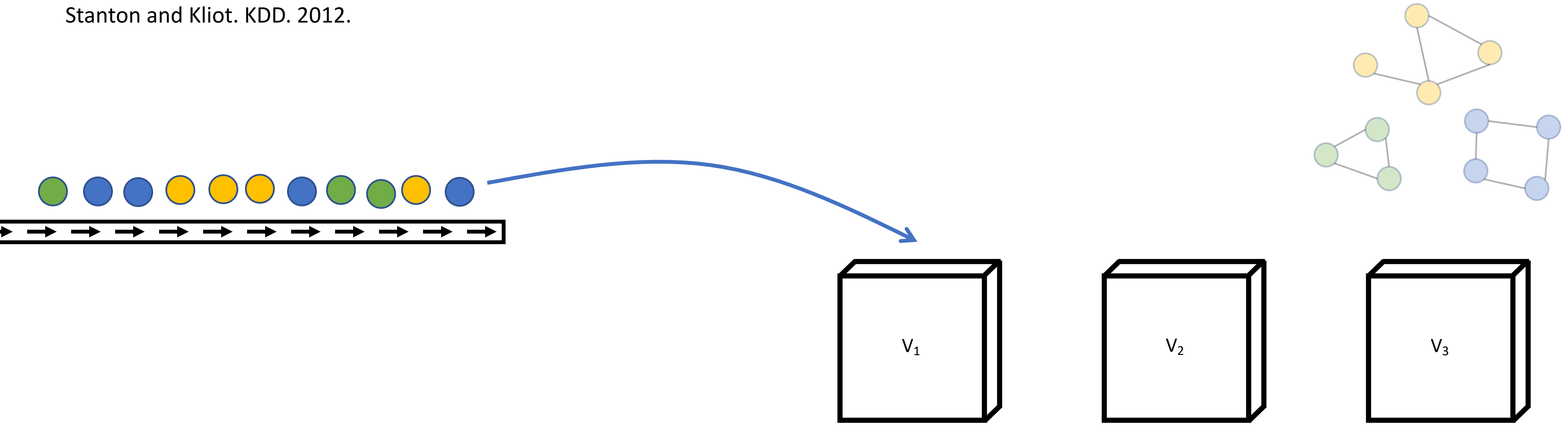




The SHP algorithm boasts many bells and whistles. We denote this version *KL-SHP* and also study two restricted forms, *SHP-I* and *SHP-II*.

Restreaming linear deterministic greedy

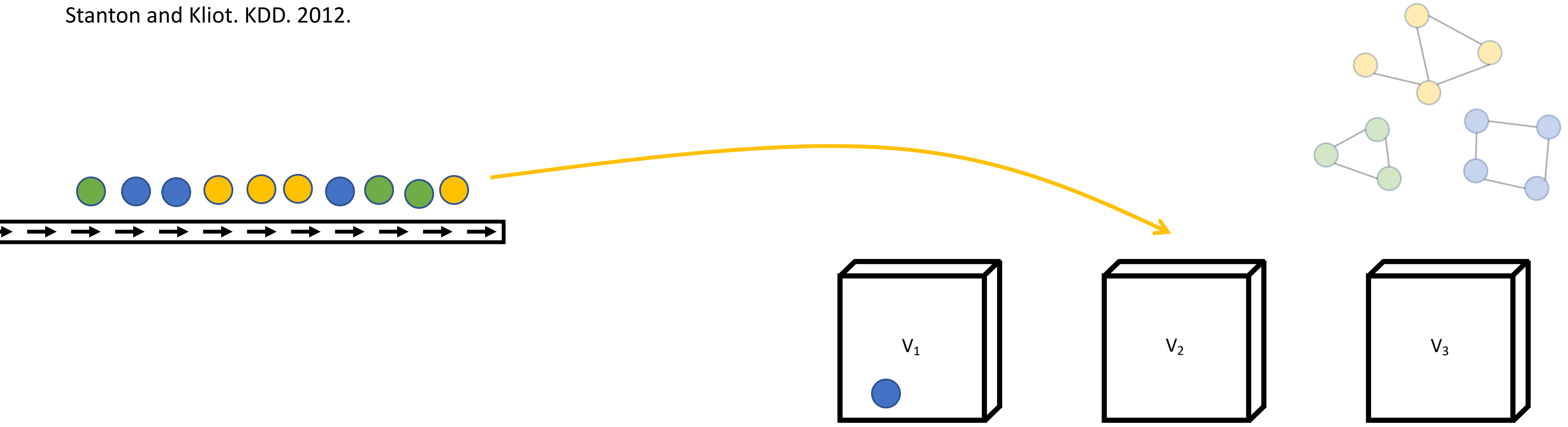
Nishimura and Ugander. *KDD*. 2013.
Stanton and Kliot. KDD. 2012.



ReLDG is a streaming algorithm, and does not require an initial partitioning.

Restreaming linear deterministic greedy

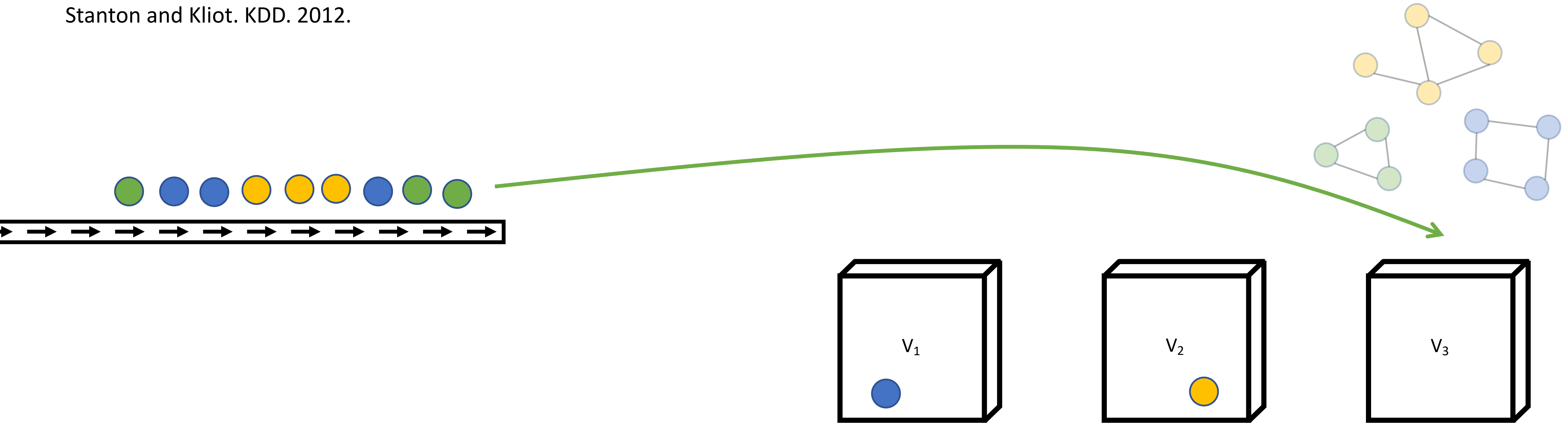
Nishimura and Ugander. *KDD*. 2013.
Stanton and Kliot. KDD. 2012.



It repeatedly streams nodes one at a time to the shard that satisfies the given assignment mechanism.

Restreaming linear deterministic greedy

Nishimura and Ugander. *KDD*. 2013.
Stanton and Kliot. KDD. 2012.



It repeatedly streams nodes one at a time to the shard that satisfies the given assignment mechanism.

Restreaming linear deterministic greedy

Nishimura and Ugander. *KDD*. 2013.
Stanton and Kliot. KDD. 2012.

$$\arg\max_{i \in [k]} |V_i^{(t)} \cap N(u)| \cdot \left(1 - \frac{x_i^{(t)}}{C}\right)$$

It repeatedly streams nodes one at a time to the shard that satisfies the given assignment mechanism.

Restreaming linear deterministic greedy

Nishimura and Ugander. *KDD*. 2013.
Stanton and Kliot. KDD. 2012.

$$\arg\max_{i \in [k]} |V_i^{(t)} \cap N(u)| \cdot \left(1 - \frac{x_i^{(t)}}{C}\right)$$

Share of *u*'s neighbors
in shard *i*, $N_{u,i}$

It repeatedly streams nodes one at a time to the shard that satisfies the given assignment mechanism.

Restreaming linear deterministic greedy

Nishimura and Ugander. *KDD*. 2013.
Stanton and Kliot. KDD. 2012.

$$\arg\max_{i \in [k]} |V_i^{(t)} \cap N(u)| \cdot \left(1 - \frac{x_i^{(t)}}{C}\right)$$

Share of *u*'s neighbors in shard *i*, $N_{u,i}$

Multiplicative weight on emptiness of shard *i*

It repeatedly streams nodes one at a time to the shard that satisfies the given assignment mechanism.

Algorithmic taxonomy

	BLP	KL-SHP	reLDG
Reassignment mechanism	Synchronous	Synchronous	Streaming
Constraint handling	Flow-based (LP)	Pairwise (swaps)	Multiplicative weight
Incumbency	✓	✗	✗
Priority	✓	✓	✗

Algorithmic taxonomy

	BLP	KL-SHP	reLDG
Reassignment mechanism	Synchronous	Synchronous	Streaming
Constraint handling	Flow-based (LP)	Pairwise (swaps)	Multiplicative weight
Incumbency	✓	✗	✗
Priority	✓	✓	✗

Algorithmic taxonomy

	BLP	KL-SHP	reLDG
Reassignment mechanism	Synchronous	Synchronous	Streaming
Constraint handling	Flow-based (LP)	Pairwise (swaps)	Multiplicative weight
Incumbency	✓	✗	✗
Priority	✓	✓	✗

Algorithmic taxonomy

	BLP	KL-SHP	reLDG
Reassignment mechanism	Synchronous	Synchronous	Streaming
Constraint handling	Flow-based (LP)	Pairwise (swaps)	Multiplicative weight
Incumbency	✓	✗	✗
Priority	✓	✓	✗

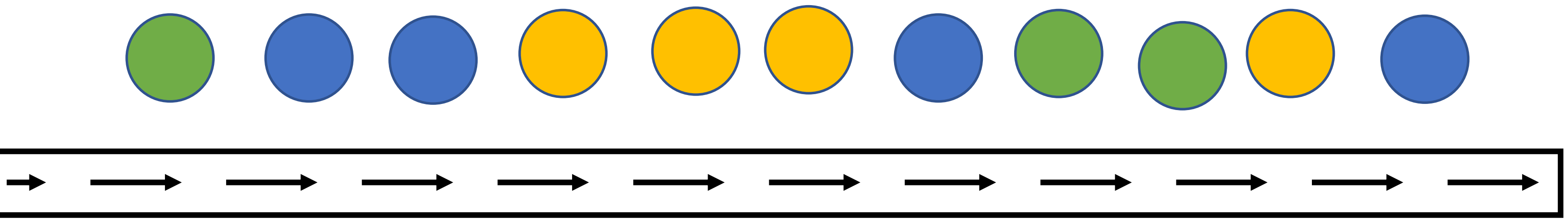
Algorithmic taxonomy

	BLP	KL-SHP	reLDG
Reassignment mechanism	Synchronous	Synchronous	Streaming
Constraint handling	Flow-based (LP)	Pairwise (swaps)	Multiplicative weight
Incumbency	✓	✗	✗
Priority	✓	✓	✗

Algorithmic taxonomy

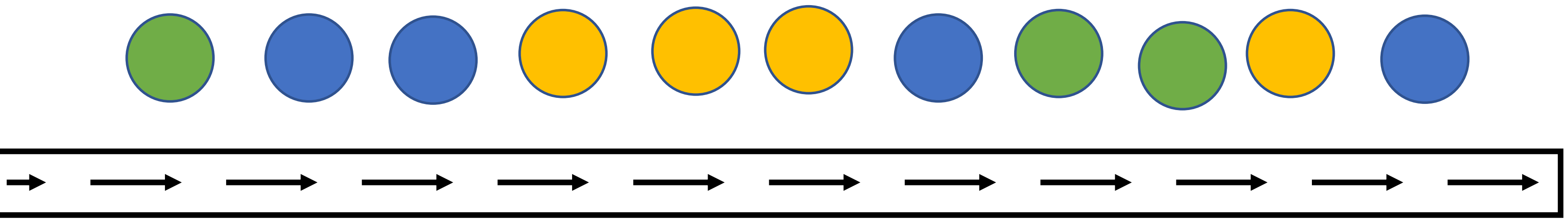
	BLP	KL-SHP	reLDG
Reassignment mechanism	Synchronous	Synchronous	Streaming
Constraint handling	Flow-based (LP)	Pairwise (swaps)	Multiplicative weight
Incumbency	✓	✗	✗
Priority	✓	✓	?

Priority in stream order



The order in which we choose to stream nodes is an obvious avenue for injecting priority into reLDG.

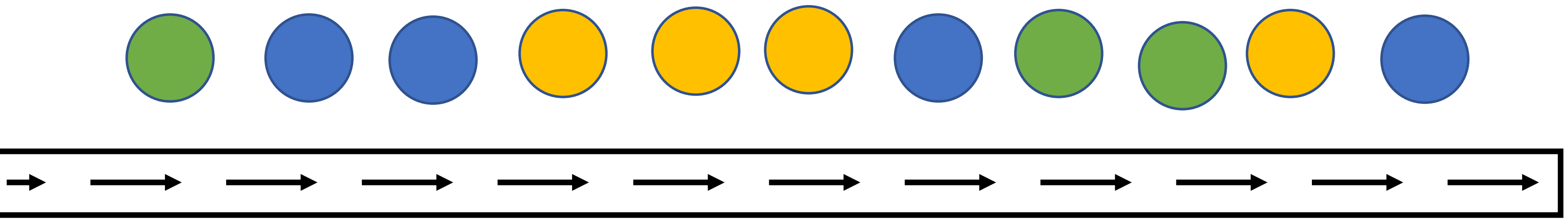
Priority in stream order



Previously studied orders
Random
BFS/DFS (from random node)

So far, only random, BFS and DFS (from a random node) orders are discussed in the literature.

Priority in stream order

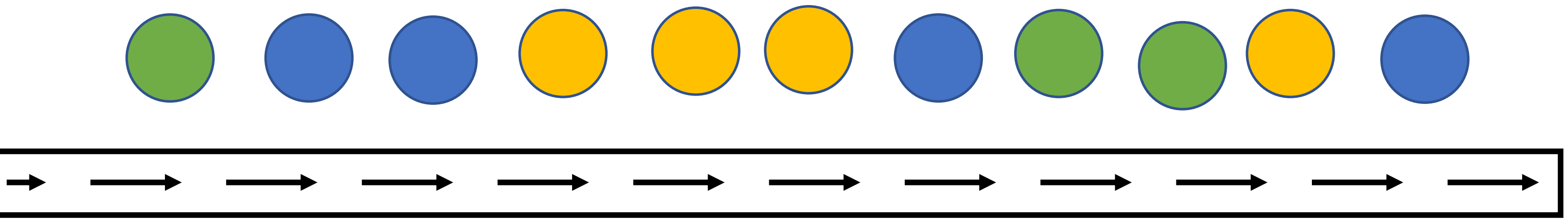


<u>Prioritized orders</u>

BFS (from highest degree)
Local clustering coefficient
Degree
Gain, g_u

In the offline setting, we can choose more strategic *static* and *dynamic* orderings.

Priority in stream order



Prioritized orders

BFS (from highest degree)

Local clustering coefficient

Degree

Gain, g_u

Ambivalence, a_u

In the offline setting, we can choose more strategic *static* and *dynamic* orderings.

New metric for sorting order

Ambivalence of node *u*

$$a_u = -\max_{i \in [k] \setminus P(u)} |N_{u,i} - N_{u,P(u)}|$$

New metric for sorting order

Ambivalence of node *u*

$$a_u = - \max_{i \in [k] \setminus P(u)} |N_{u,i} - N_{u,P(u)}|$$

Negative max over
external shards

New metric for sorting order

Ambivalence of node *u*

$$a_u = - \max_{i \in [k] \setminus P(u)} \left| N_{u,i} - N_{u,P(u)} \right|$$

Negative max over external shards

Absolute difference in co-located neighbor count

New metric for sorting order

Ambivalence of node *u*

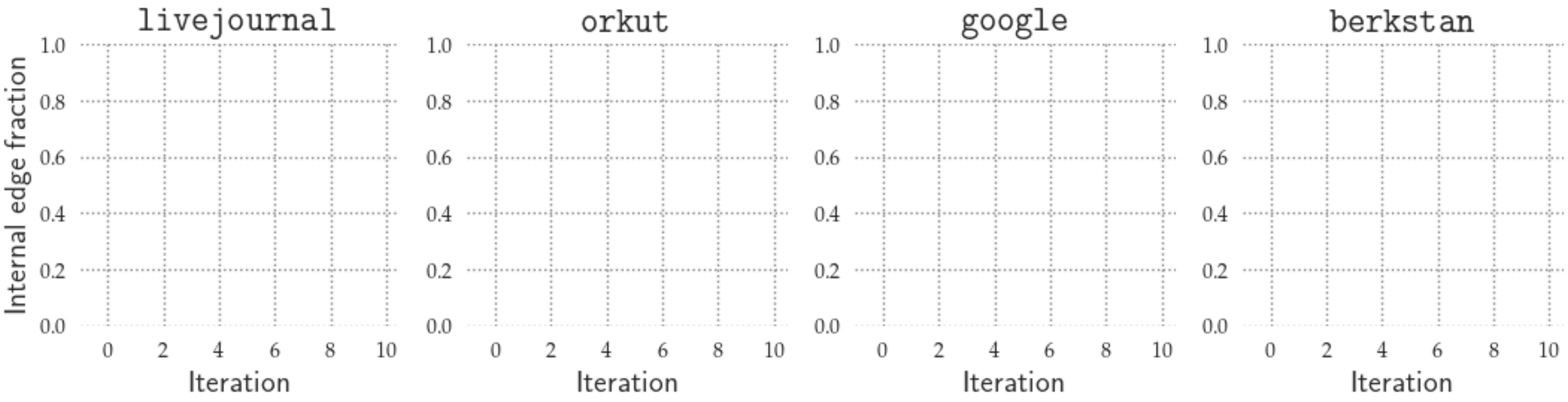
$$a_u = -\max_{i \in [k] \setminus P(u)} \left| N_{u,i} - N_{u,P(u)} \right|$$

Negative max over external shards

Absolute difference in co-located neighbor count

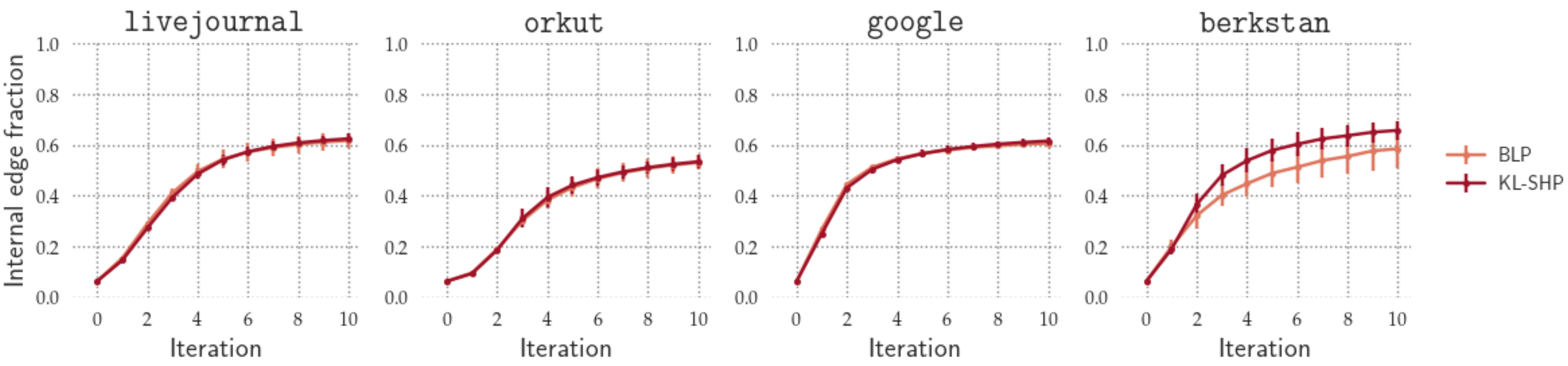
The larger the *magnitude* of the difference, the more negative the value, the less “ambivalent” the node is to relocation.

Benchmarking base methods



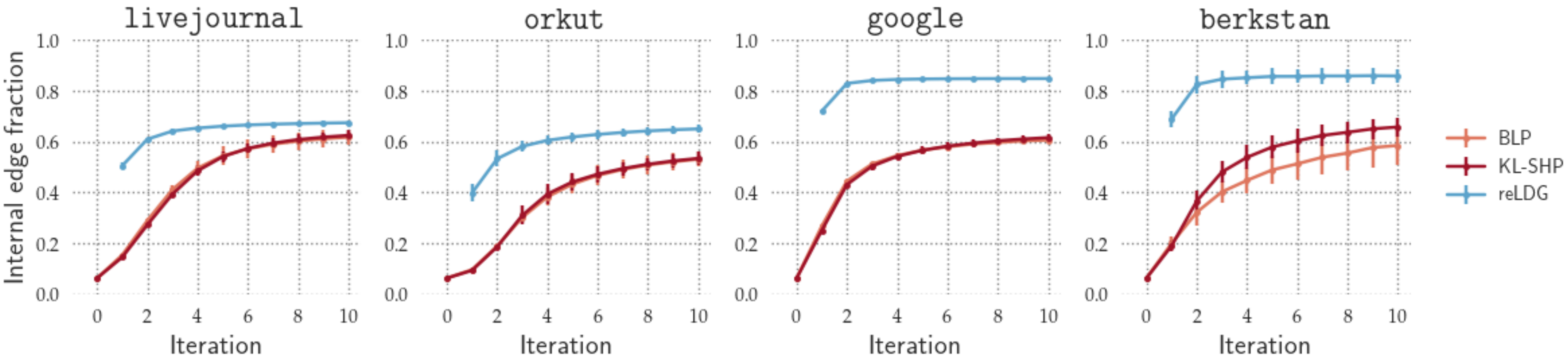
First, we plot internal edge fraction of BLP, KL-SHP, and reLDG as a function of iteration on 4 datasets.

Benchmarking base methods



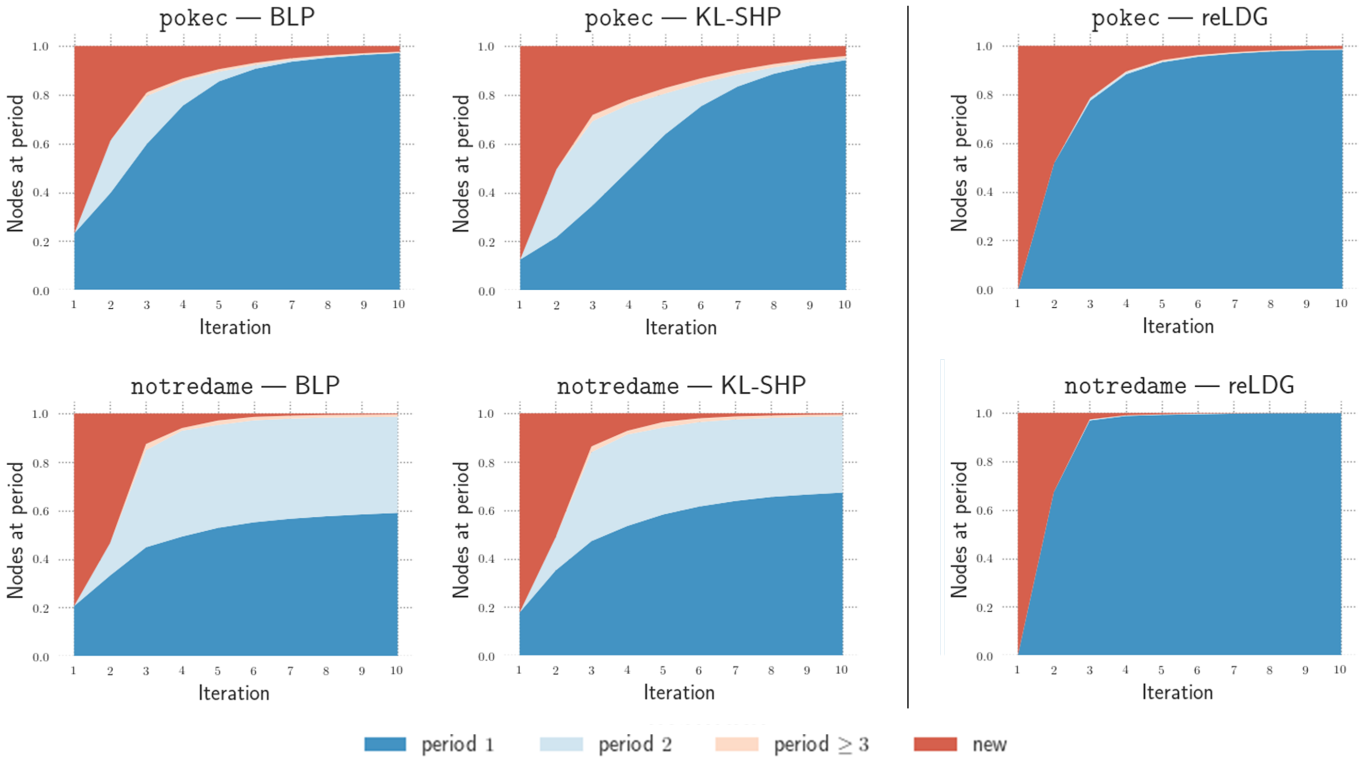
BLP and KL-SHP display similar performance, with KL-SHP winning out on all tested networks.

Benchmarking base methods



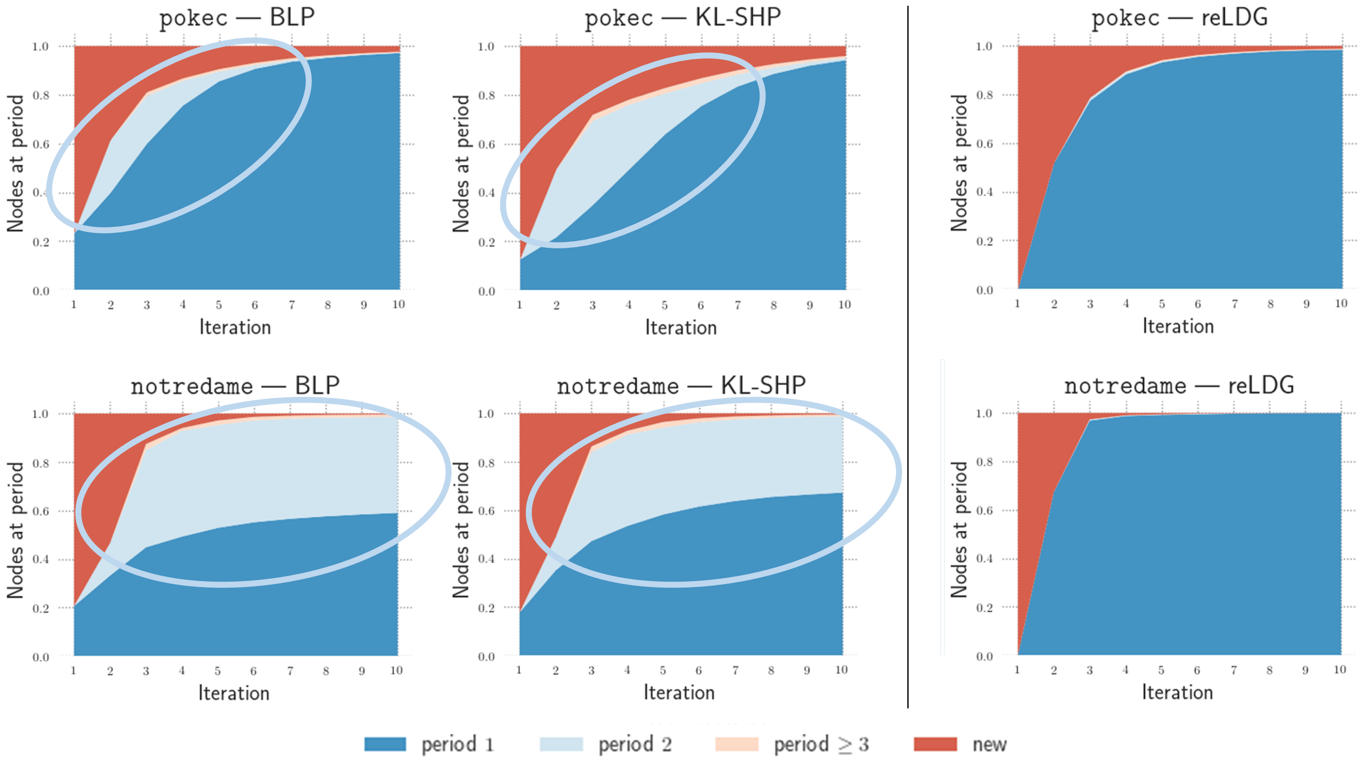
reLDG with random stream order results in higher quality partitions in fewer iterations than both synchronous ones.

Periodic reassignment of synchronous class



To investigate the assignment behavior of nodes under our three base methods, we plot their *periodicity*.

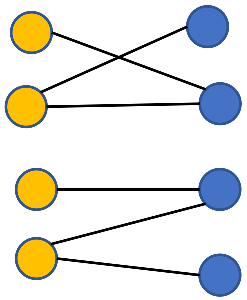
Periodic reassignment of synchronous class



We find that many nodes get assigned to the shard they were assigned to two iterations prior under the synchronous algorithms.

Periodic reassignment of synchronous class

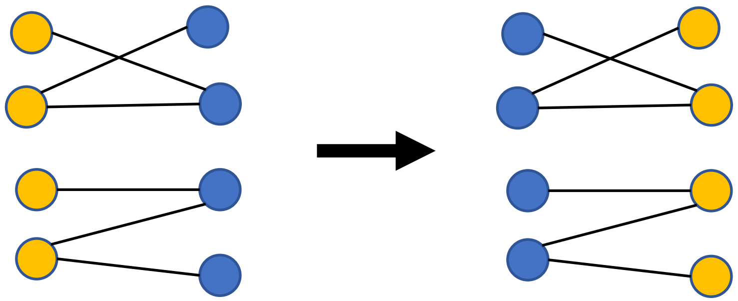
Synchronous:



The phenomenon is illustrated well by a bipartite network and provides intuition for streaming's superior performance.

Periodic reassignment of synchronous class

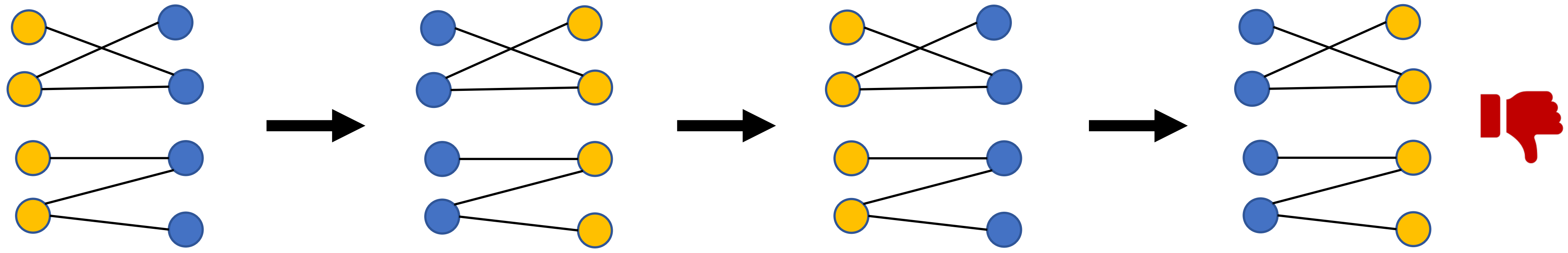
Synchronous:



The phenomenon is illustrated well by a bipartite network and provides intuition for streaming’s superior performance.

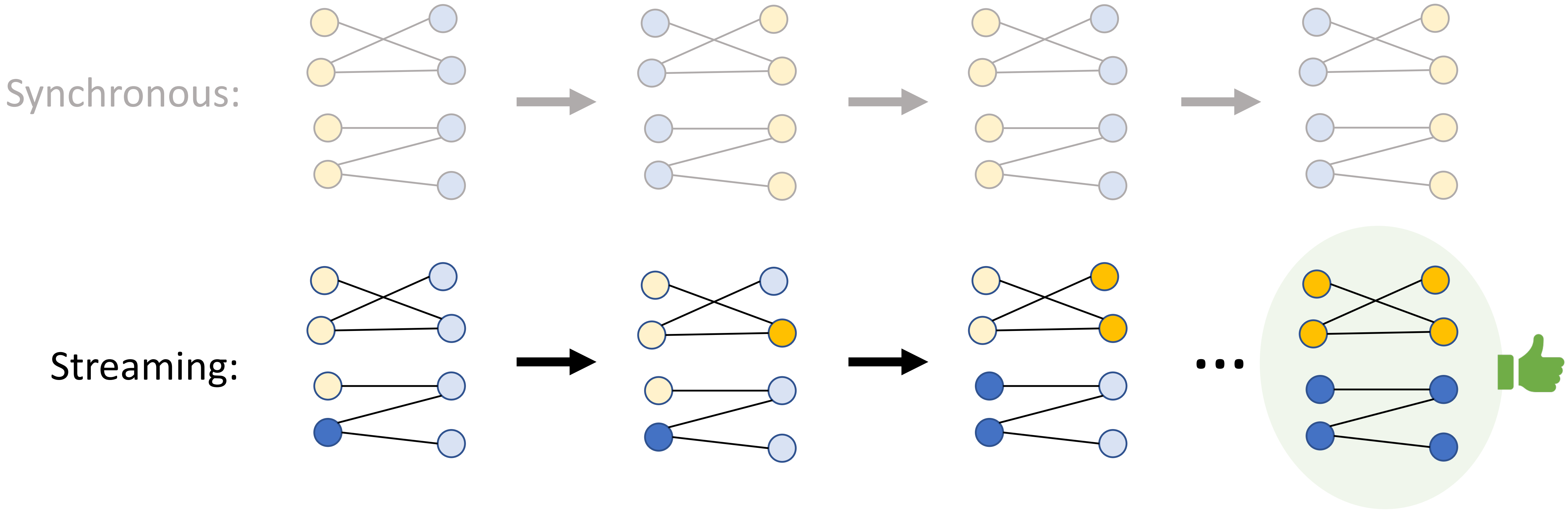
Periodic reassignment of synchronous class

Synchronous:



The phenomenon is illustrated well by a bipartite network and provides intuition for streaming's superior performance.

Periodic reassignment of synchronous class



The phenomenon is illustrated well by a bipartite network and provides intuition for streaming's superior performance.

Synchronous vs. streaming performance

Graph	Synchronous				Streaming (reLDG)					
	SHP-I	SHP-II	KL-SHP	BLP	Random	CC	BFS	Degree	Ambivalence	Gain
pokec	0.578	0.595	0.585	0.532	0.675	0.681	0.698	**0.716**	0.712	0.618
livejournal	0.626	0.648	0.625	0.617	0.674	0.666	0.731	0.745	**0.749**	0.671
orkut	0.535	0.555	0.534	0.531	0.650	0.628	0.665	**0.689**	0.679	0.626
notredame	0.783	0.635	0.652	0.612	0.882	0.864	**0.929**	0.902	0.924	0.878
stanford	0.737	0.711	0.697	0.629	0.856	0.844	0.891	0.900	**0.916**	0.793
google	0.670	0.603	0.616	0.606	0.848	0.814	0.868	0.959	**0.964**	0.799
berkstan	0.701	0.652	0.658	0.585	0.858	0.805	0.895	0.913	**0.918**	0.766

Internal edge fraction of 16-shard partitioning after 10 iterations, averaged over 10 trials.

Synchronous vs. streaming performance

Best

Graph	Synchronous				Streaming (reLDG)					
	SHP-I	SHP-II	KL-SHP	BLP	Random	CC	BFS	Degree	Ambivalence	Gain
pokec	0.578	0.595	0.585	0.532	0.675	0.681	0.698	**0.716**	0.712	0.618
livejournal	0.626	0.648	0.625	0.617	0.674	0.666	0.731	0.745	**0.749**	0.671
orkut	0.535	0.555	0.534	0.531	0.650	0.628	0.665	**0.689**	0.679	0.626
notredame	0.783	0.635	0.652	0.612	0.882	0.864	**0.929**	0.902	0.924	0.878
stanford	0.737	0.711	0.697	0.629	0.856	0.844	0.891	0.900	**0.916**	0.793
google	0.670	0.603	0.616	0.606	0.848	0.814	0.868	0.959	**0.964**	0.799
berkstan	0.701	0.652	0.658	0.585	0.858	0.805	0.895	0.913	**0.918**	0.766

Internal edge fraction of 16-shard partitioning after 10 iterations, averaged over 10 trials.

Synchronous vs. streaming performance

Worst

Graph	Synchronous				Streaming (reLDG)					
	SHP-I	SHP-II	KL-SHP	BLP	Random	CC	BFS	Degree	Ambivalence	Gain
pokec	0.578	0.595	0.585	0.532	0.675	0.681	0.698	**0.716**	0.712	0.618
livejournal	0.626	0.648	0.625	0.617	0.674	0.666	0.731	0.745	**0.749**	0.671
orkut	0.535	0.555	0.534	0.531	0.650	0.628	0.665	**0.689**	0.679	0.626
notredame	0.783	0.635	0.652	0.612	0.882	0.864	**0.929**	0.902	0.924	0.878
stanford	0.737	0.711	0.697	0.629	0.856	0.844	0.891	0.900	**0.916**	0.793
google	0.670	0.603	0.616	0.606	0.848	0.814	0.868	0.959	**0.964**	0.799
berkstan	0.701	0.652	0.658	0.585	0.858	0.805	0.895	0.913	**0.918**	0.766

Internal edge fraction of 16-shard partitioning after 10 iterations, averaged over 10 trials.

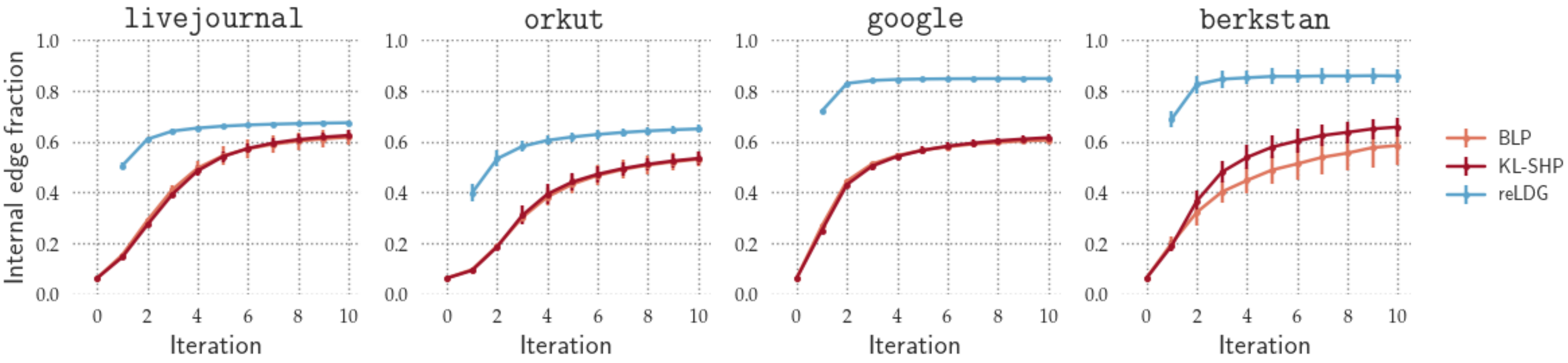
Synchronous vs. streaming performance

Graph	Synchronous				Streaming (reLDG)					
	SHP-I	SHP-II	KL-SHP	BLP	Random	CC	BFS	Degree	Ambivalence	Gain
pokec	0.578	0.595	0.585	0.532	0.675	0.681	0.698	**0.716**	0.712	0.618
livejournal	0.626	0.648	0.625	0.617	0.674	0.666	0.731	0.745	**0.749**	0.671
orkut	0.535	0.555	0.534	0.531	0.650	0.628	0.665	**0.689**	0.679	0.626
notredame	0.783	0.635	0.652	0.612	0.882	0.864	**0.929**	0.902	0.924	0.878
stanford	0.737	0.711	0.697	0.629	0.856	0.844	0.891	0.900	**0.916**	0.793
google	0.670	0.603	0.616	0.606	0.848	0.814	0.868	0.959	**0.964**	0.799
berkstan	0.701	0.652	0.658	0.585	0.858	0.805	0.895	0.913	**0.918**	0.766

Internal edge fraction of 16-shard partitioning after 10 iterations, averaged over 10 trials.

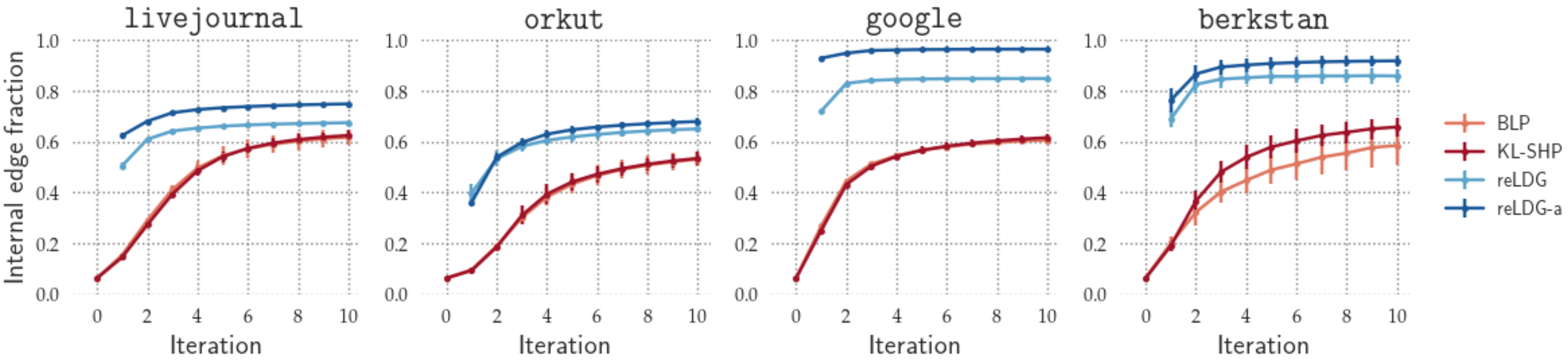
Note that the worst performing stream order outperforms the best performer of the synchronous class, a truly remarkable result.

Prioritized restreaming



Furthermore, streaming nodes in order of increasing ambivalence can significantly improve the quality of the resulting partition.

Prioritized restreaming



Furthermore, streaming nodes in order of increasing ambivalence can significantly improve the quality of the resulting partition.

Prioritized restreaming

Prioritized orders

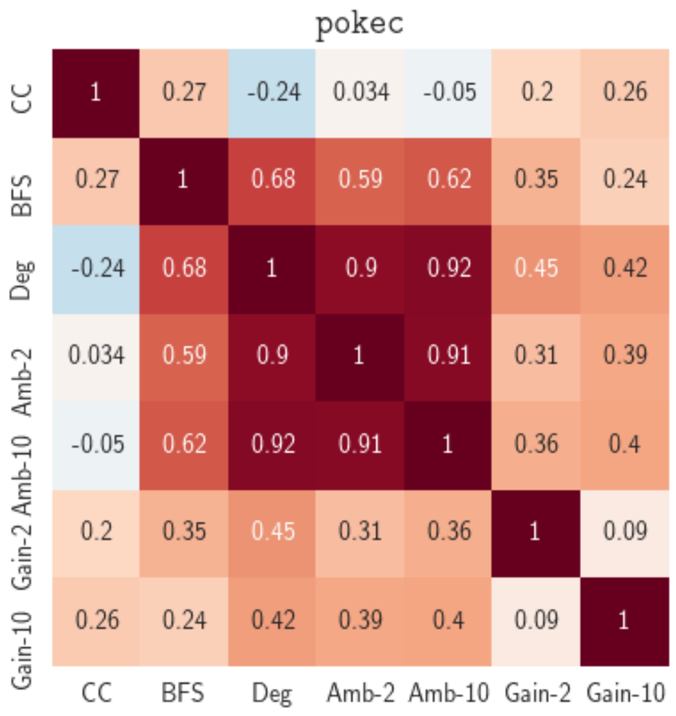
Graph	Synchronous				Streaming (reLDG)					
	SHP-I	SHP-II	KL-SHP	BLP	Random	CC	BFS	Degree	Ambivalence	Gain
pokec	0.578	0.595	0.585	0.532	0.675	0.681	0.698	**0.716**	0.712	0.618
livejournal	0.626	0.648	0.625	0.617	0.674	0.666	0.731	0.745	**0.749**	0.671
orkut	0.535	0.555	0.534	0.531	0.650	0.628	0.665	**0.689**	0.679	0.626
notredame	0.783	0.635	0.652	0.612	0.882	0.864	**0.929**	0.902	0.924	0.878
stanford	0.737	0.711	0.697	0.629	0.856	0.844	0.891	0.900	**0.916**	0.793
google	0.670	0.603	0.616	0.606	0.848	0.814	0.868	0.959	**0.964**	0.799
berkstan	0.701	0.652	0.658	0.585	0.858	0.805	0.895	0.913	**0.918**	0.766

Internal edge fraction of 16-shard partitioning after 10 iterations, averaged over 10 trials.

The top performer in each row lies in the prioritized restreaming category, showing up to 12% improvement in objective from random.

Correlation between stream orders

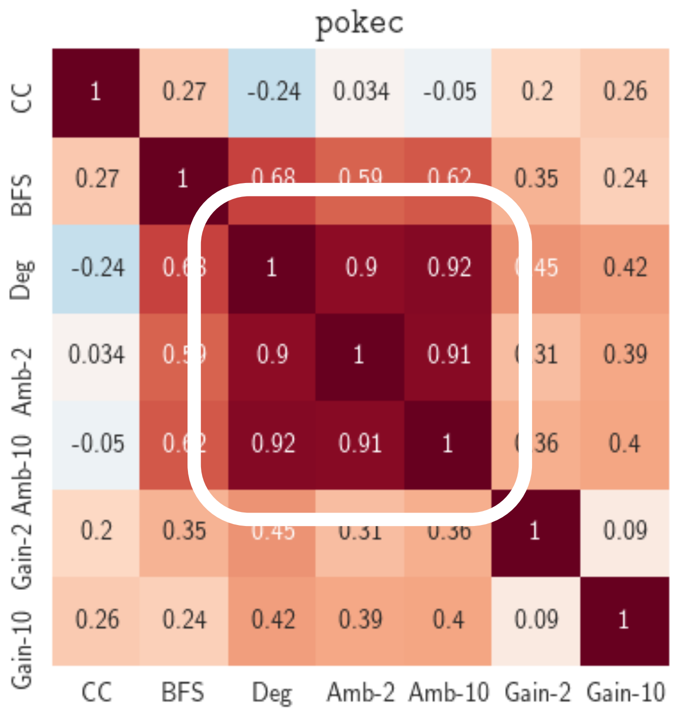
Vigna. WWW. 2015.



To quantify their differences, we plot the weighted Kendall’s tau correlation between all tested stream orders.

Correlation between stream orders

Vigna. WWW. 2015.



Decreasing-degree and increasing-ambivalence are highly correlated orderings.

Ambivalence and degree

$$\frac{2}{k} \cdot d_u \leq \mathbb{E}[\tilde{a}_u] \leq \frac{2(k-1)}{k} \cdot d_u$$

Further, ambivalence is upper and lower bounded by a linear function of degree, relative to a random partitioning.

Takeaways

From this talk
Streaming > synchronous.
Prioritized orders show significant improvement over random.
Ambivalence and degree are most promising orders and are highly correlated.

Takeaways

From paper
“Less is more” within synchronous.
Incumbency exploration shows that methods are good as is regarding the option.
reLDG outperforms previously benchmarked methods with increasing *k*.

Thank you!

Awadelkarim and Ugander. “Prioritized Restreaming Algorithms for Balanced Graph Partitioning”.