Parallel k-Core Decomposition: Theory and Practice

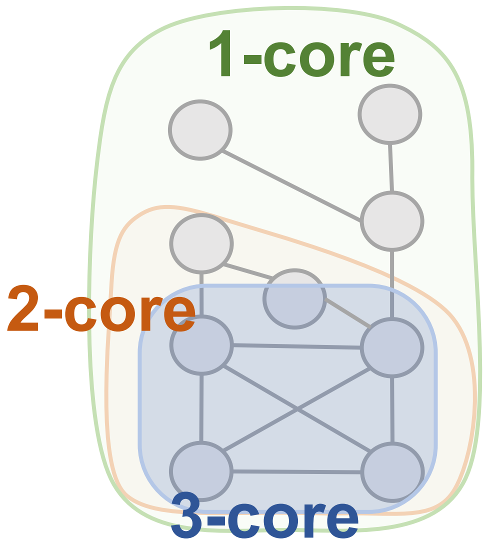
<u>Youzhe Liu</u>, Xiaojun Dong, Yan Gu, Yihan Sun

University of California, Riverside

How to Define Dense Subgraphs (formally)?

- **k-core definition:**

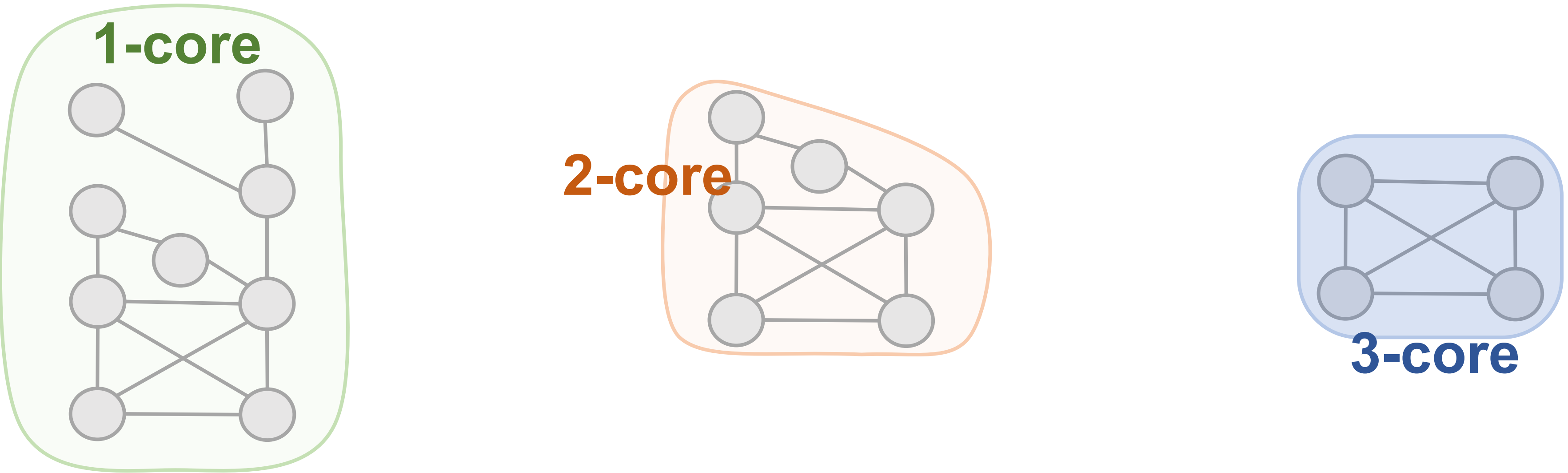
Given a graph G, k-core is the **subgraph** after **removing all** the vertices with degrees smaller than k



How to Define Dense Subgraphs (formally)?

- **k-core definition:**

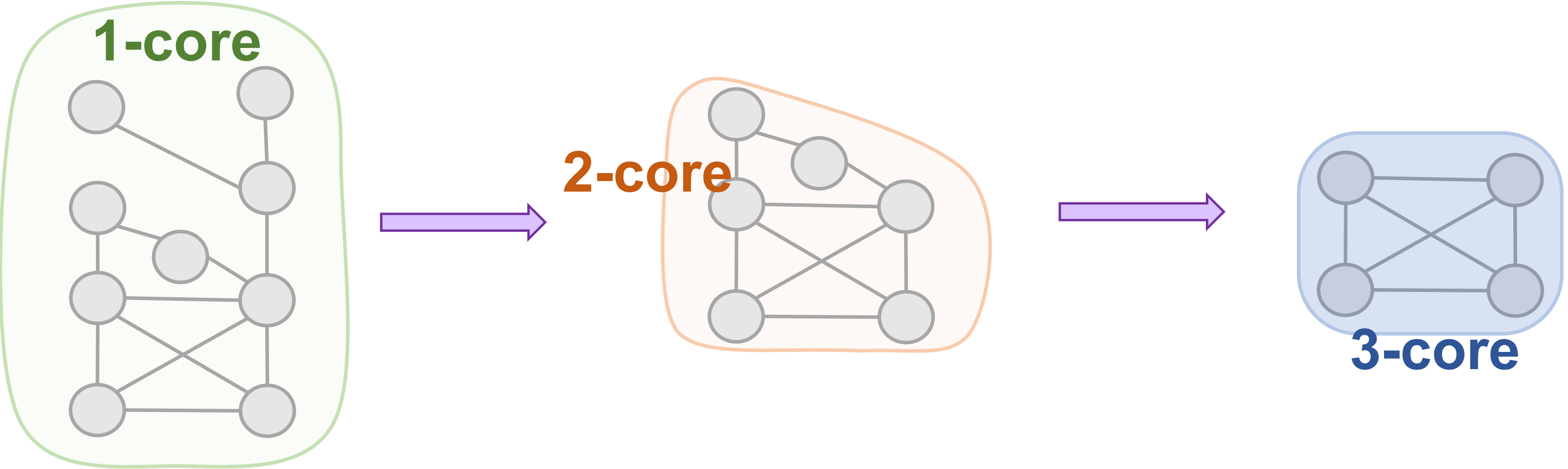
Given a graph G, k-core is the **subgraph** after **removing all** the vertices with degrees smaller than k



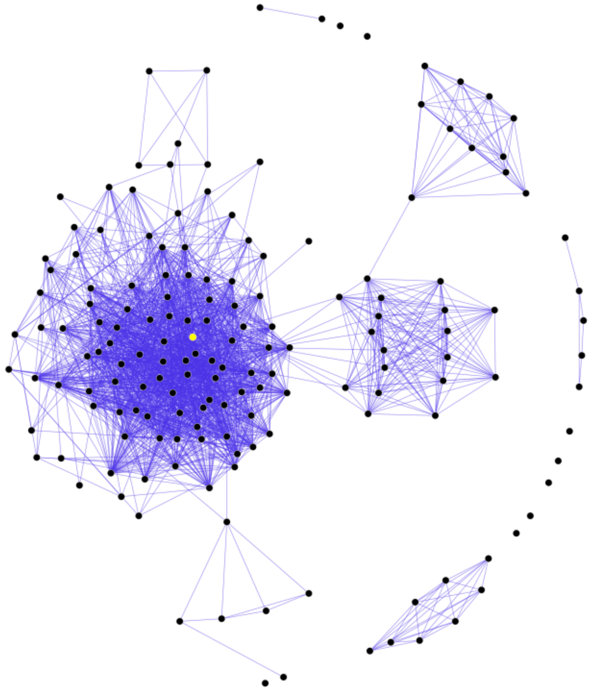
How to Define Dense Subgraphs (formally)?

• k-core decomposition:

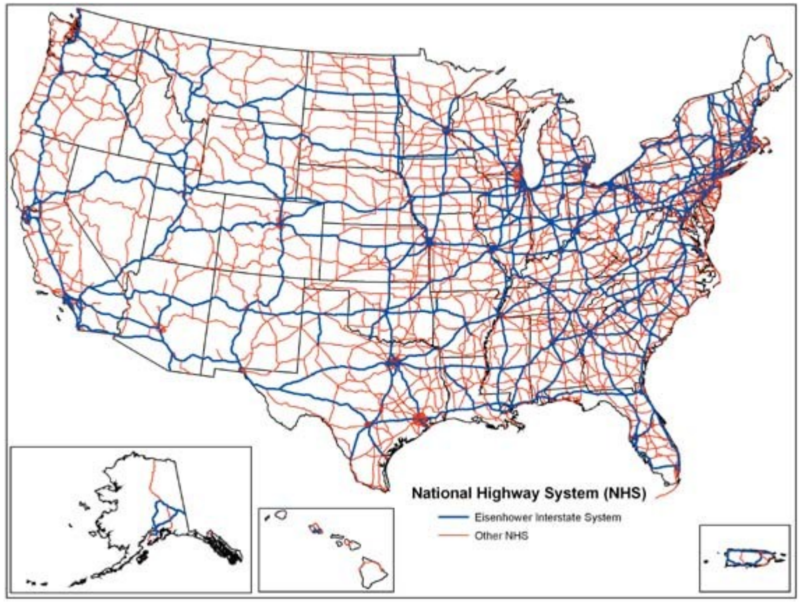
The process of listing all the k-core structures in the graph G



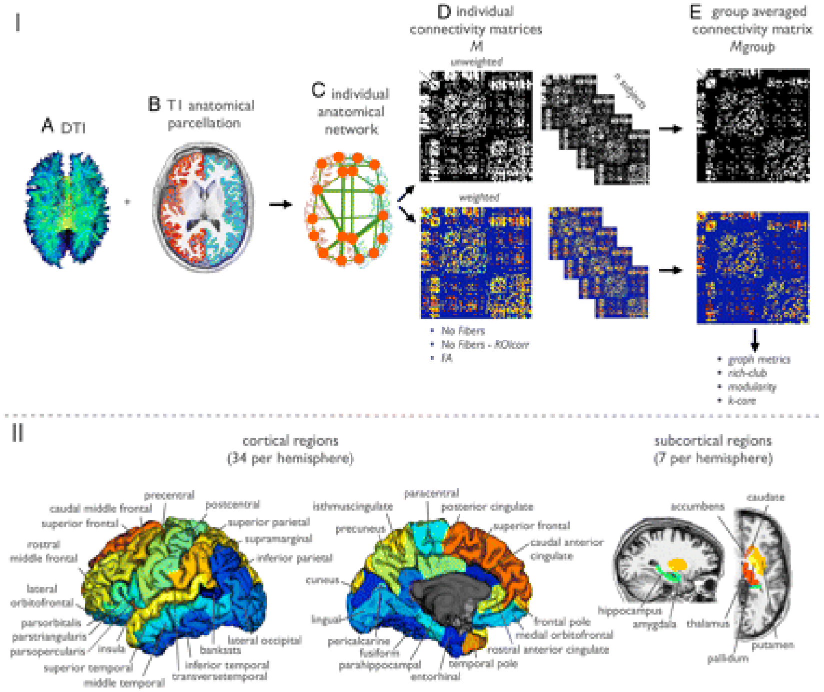
k-core is widely used



Community detection [1,2,3]



Infra-network robustness [4]



Strongly cohesive structure in biology [5,6]

[1] Marián Boguná, Romualdo Pastor-Satorras, Albert Dí az Guilera, and Alex Arenas. 2004. Models of social networks based on social distance attachment. Physical Review E—Statistical, Nonlinear, and Soft Matter Physics 70, 5 (2004)
[2] Christos Giatsidis, Dimitrios M Thilikos, and Michalis Vazirgiannis. 2011. Evaluating cooperation in communities with the k-core structure. In 2011 International conference on advances in social networks analysis and mining.
[3] Maksim Kitsak, Lazaros K Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H Eugene Stanley, and Herná n A Makse. 2010. Identification of influential spreaders in complex networks. Nature physics 6, 11 (2010), 888–893.
[4] Kate Burleson-Lesser, Flaviano Morone, Maria S Tomassone, and Hernán A Makse. 2020. K-core robustness in ecological and financial networks. Scientific reports 10, 1 (2020), 3357.
[5] Yizong Cheng, Chen Lu, and Nan Wang. 2013. Local k-core clustering for gene networks. In 2013 IEEE International Conference on Bioinformatics and Biomedicine. IEEE, 9–15.
[6] Arnold I Emerson, Simeon Andrews, Ikhlak Ahmed, Thasni KA Azis, and Joel A Malek. 2015. K-core decomposition of a protein domain co-occurrence network reveals lower cancer mutation rates for interior cores. Journal of clinical bioinformatics 5 (2015), 1–11.

Figure sources: Wikipedia

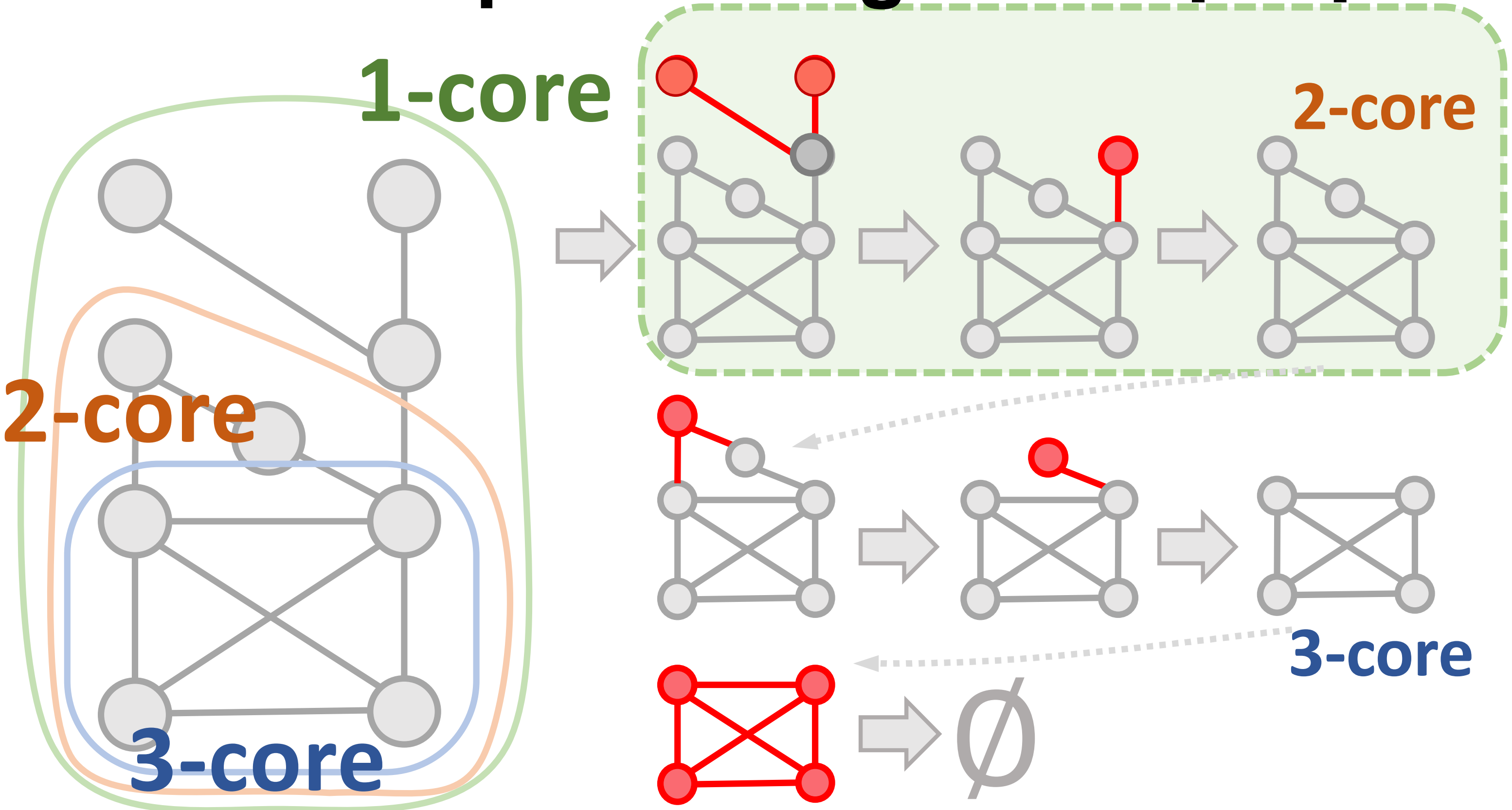
k-core is a subroutine for many problems

- Dense subgraphs discovery [1]
- Hierarchical graph clustering [2]
- Graph degeneracy for graph learning [3]
- Graph coloring [4]
- …

[1] Ahmet Erdem Sariyüce, C. Seshadhri, and Ali Pinar. 2018. Local algorithms for hierarchical dense subgraph discovery. Proc. VLDB Endow. 12, 1 (September 2018), 43–56. https://doi.org/10.14778/3275536.3275540
[2] Christos Giatsidis, Fragkiskos D. Malliaros, Dimitrios M. Thilikos, and Michalis Vazirgiannis. 2014. CORECLUSTER: a degeneracy based graph clustering framework. In Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI'14). AAAI Press, 44–50.
[3] Guillaume Salha, Romain Hennequin, Viet Anh Tran, and Michalis Vazirgiannis. 2019. A degeneracy framework for scalable graph autoencoders. In Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19).
[4] Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh. Graph Coloring via Degeneracy in Streaming and Other Space-Conscious Models. In 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020). Leibniz International Proceedings in Informatics (LIPIc

Sequential Solutions

- **Efficient sequential algorithm [BZ03]: follows the definition**



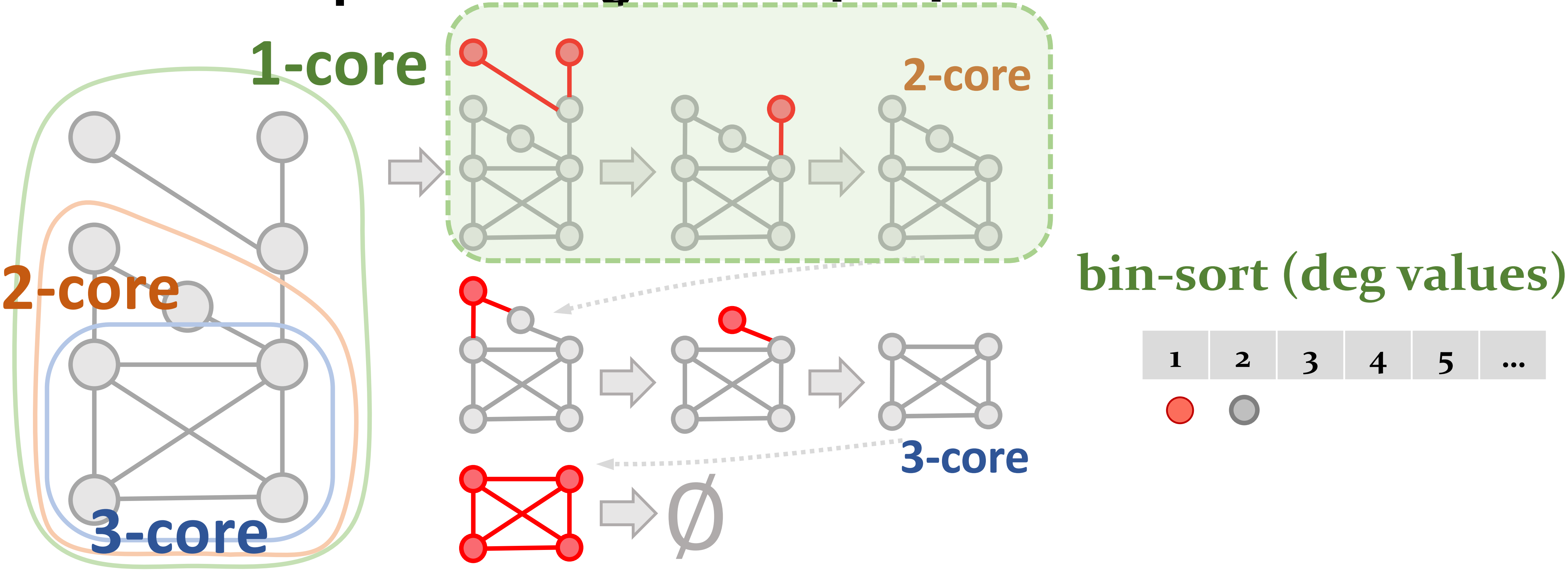
bin-sort (deg values)

1	2	3	4	5	…

[BZ03] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) algorithm for cores decomposition of networks. arXiv preprint cs/0310049 (2003).

Sequential Solutions

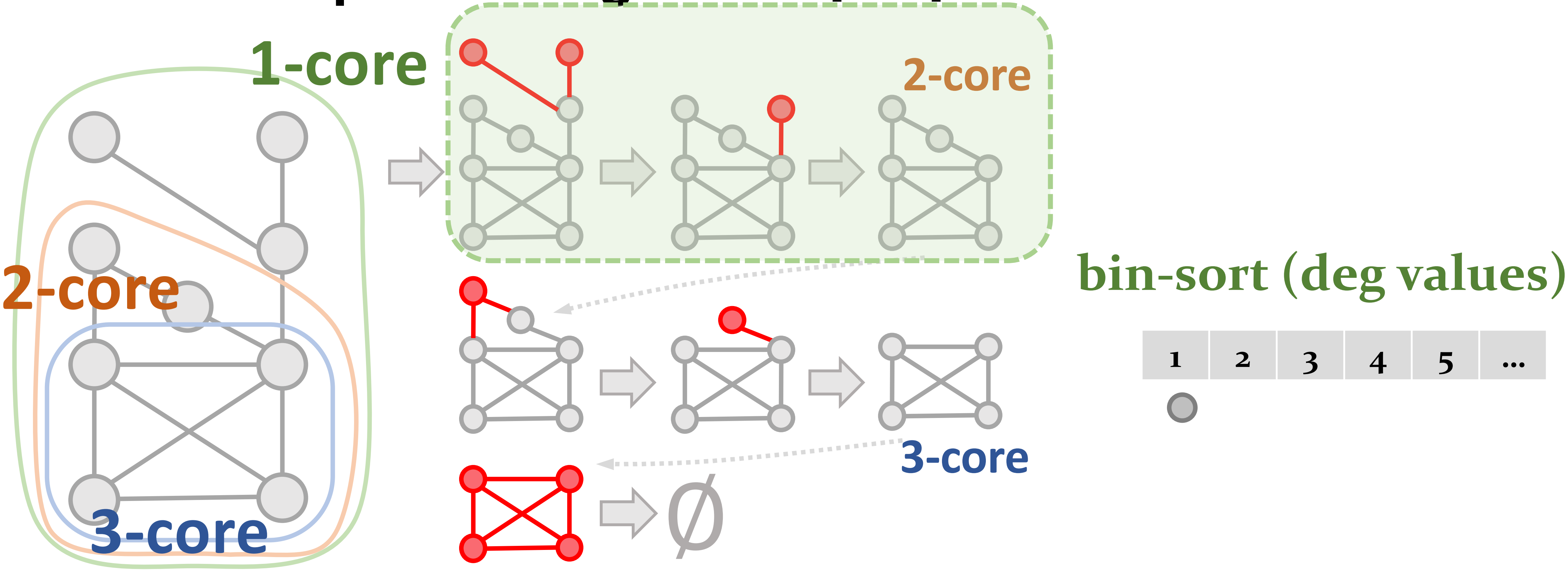
- **Efficient sequential algorithm [BZ03]: follows the definition**



[BZ03] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) algorithm for cores decomposition of networks. arXiv preprint cs/0310049 (2003).

Sequential Solutions

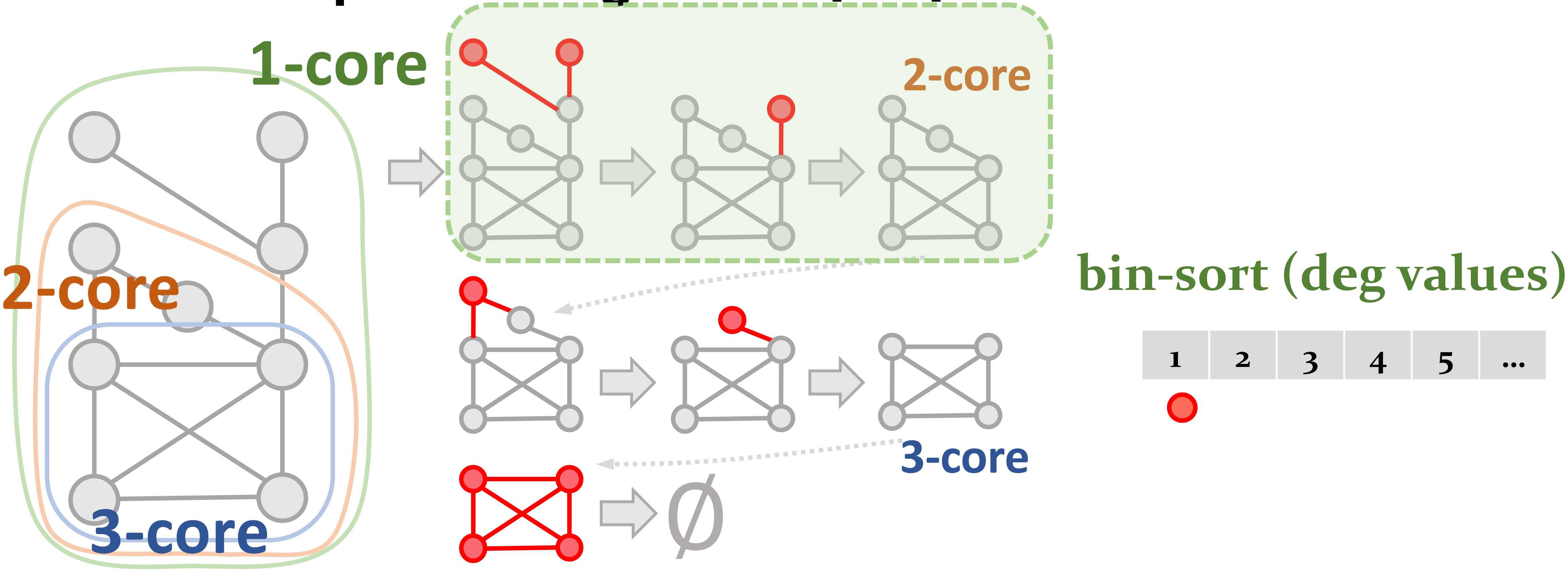
- **Efficient sequential algorithm [BZ03]: follows the definition**



[BZ03] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) algorithm for cores decomposition of networks. arXiv preprint cs/0310049 (2003).

Sequential Solutions

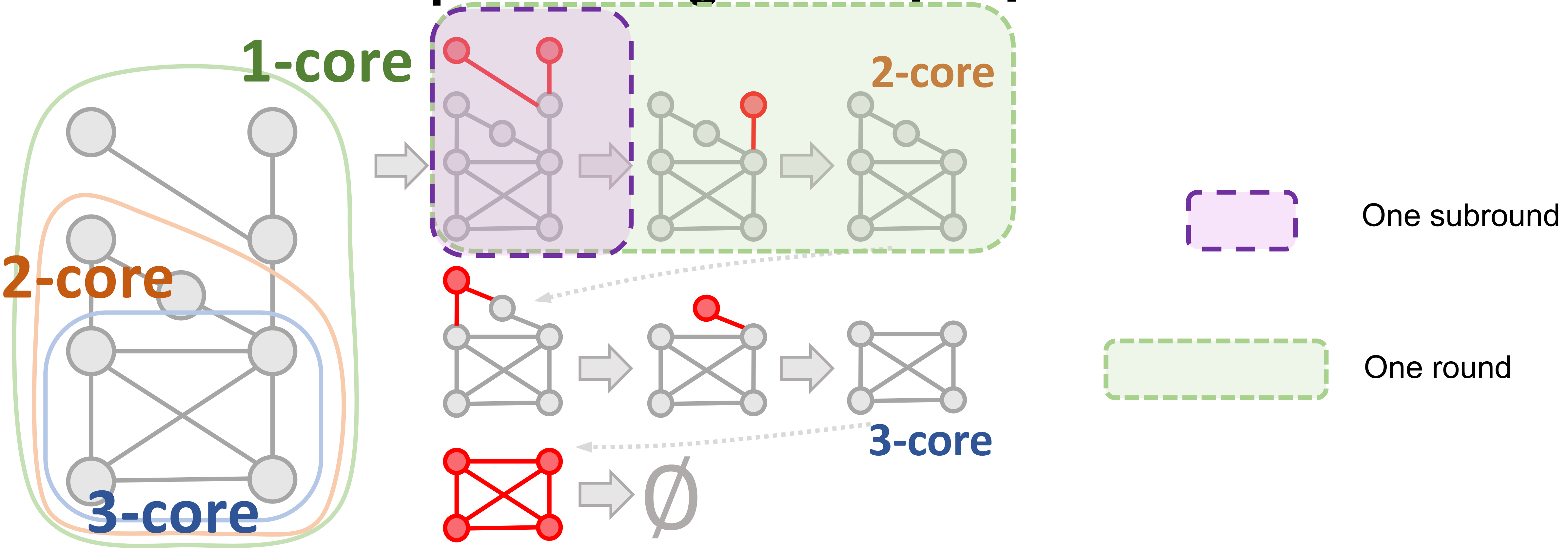
- **Efficient sequential algorithm [BZ03]: follows the definition**



[BZ03] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) algorithm for cores decomposition of networks. arXiv preprint cs/0310049 (2003).

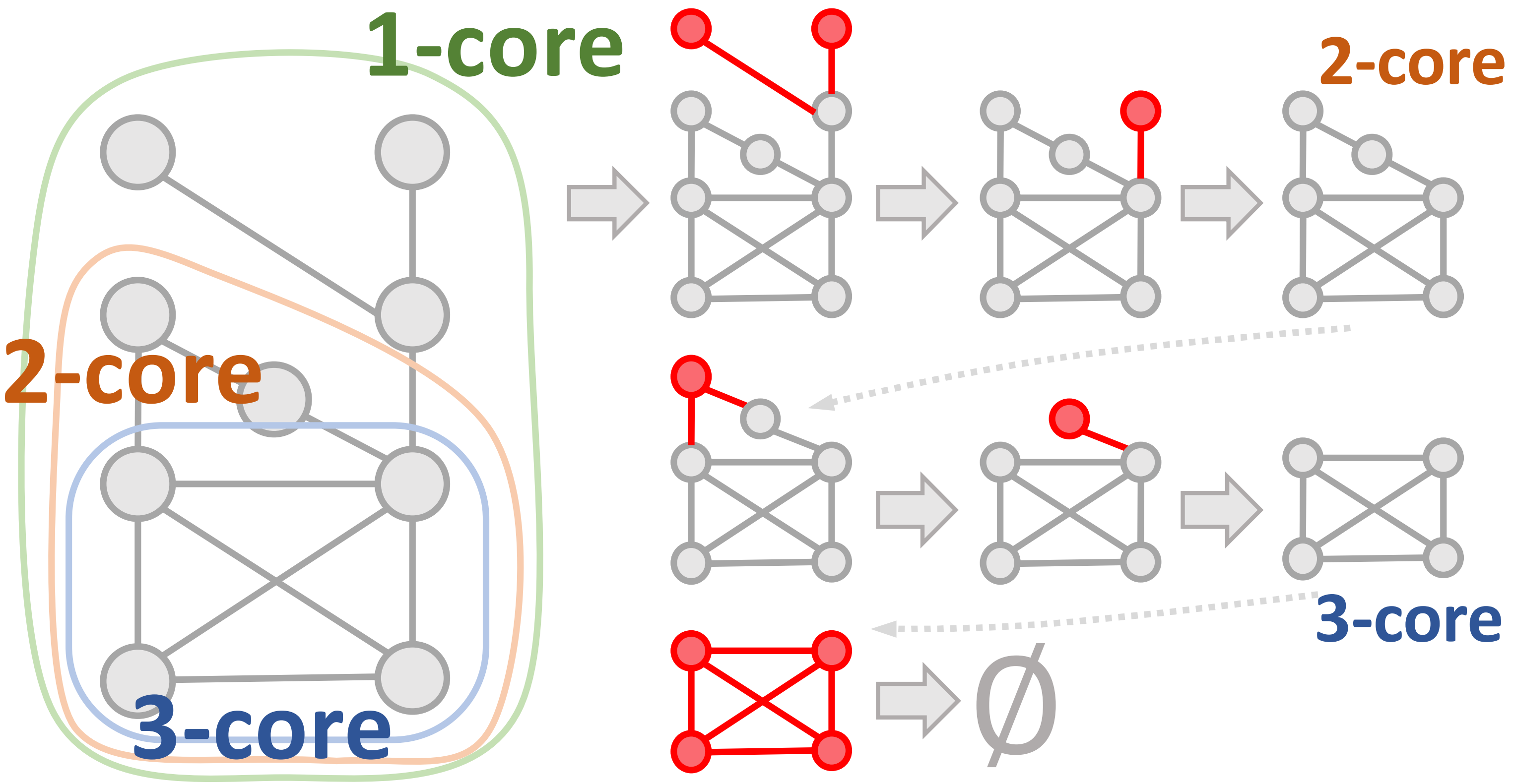
Sequential Solutions

- **Efficient sequential algorithm [BZ03]: follows the definition**



[BZ03] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) algorithm for cores decomposition of networks. arXiv preprint cs/0310049 (2003).

Sequential Solutions



$$O(|V| + |E|)$$

Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) algorithm for cores decomposition of networks. arXiv preprint cs/0310049 (2003).

The sizes of real-world graphs are very large

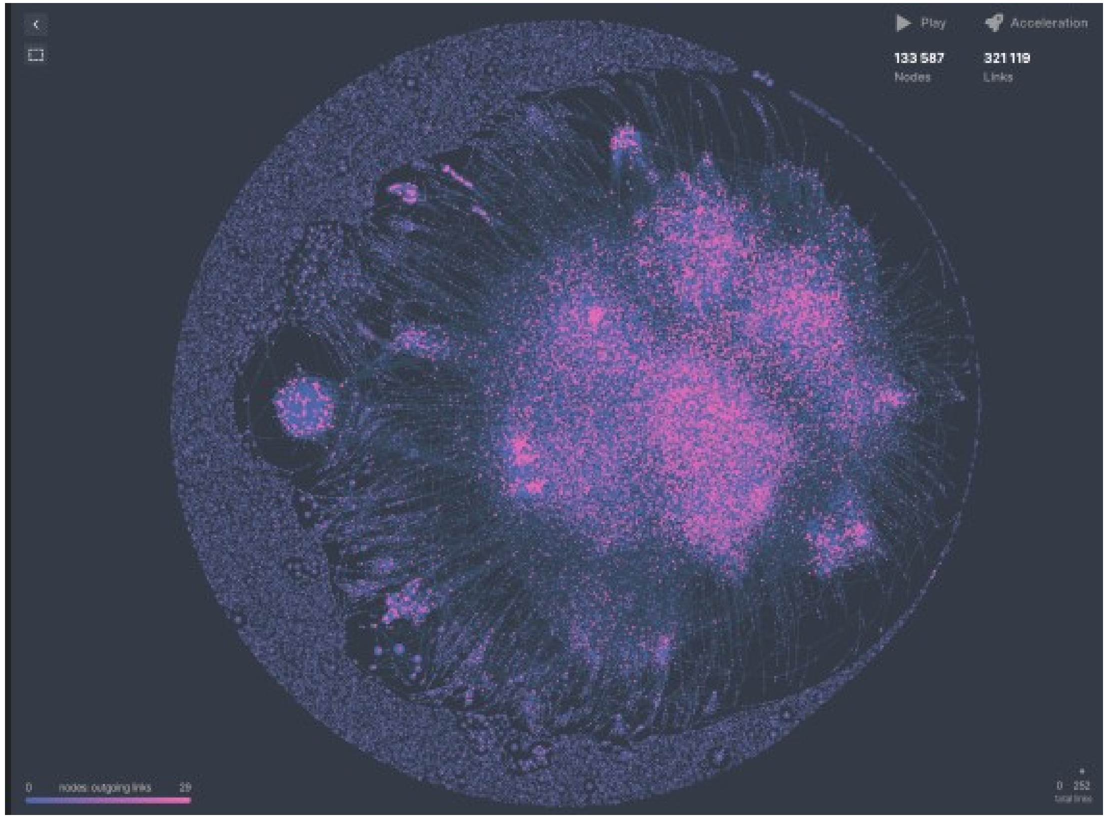


Figure source: https://nightingaledvs.com/how-to-visualize-a-graph-with-a-million-nodes/

Web graphs [1]:
Billions of edges

[1] Robert Meusel, Oliver Lehmberg, Christian Bizer, and Sebastiano Vigna. 2014. Web Data Commons — Hyperlink Graphs. http://webdatacommons.org/ hyperlinkgraph

The sizes of real-world graphs are very large

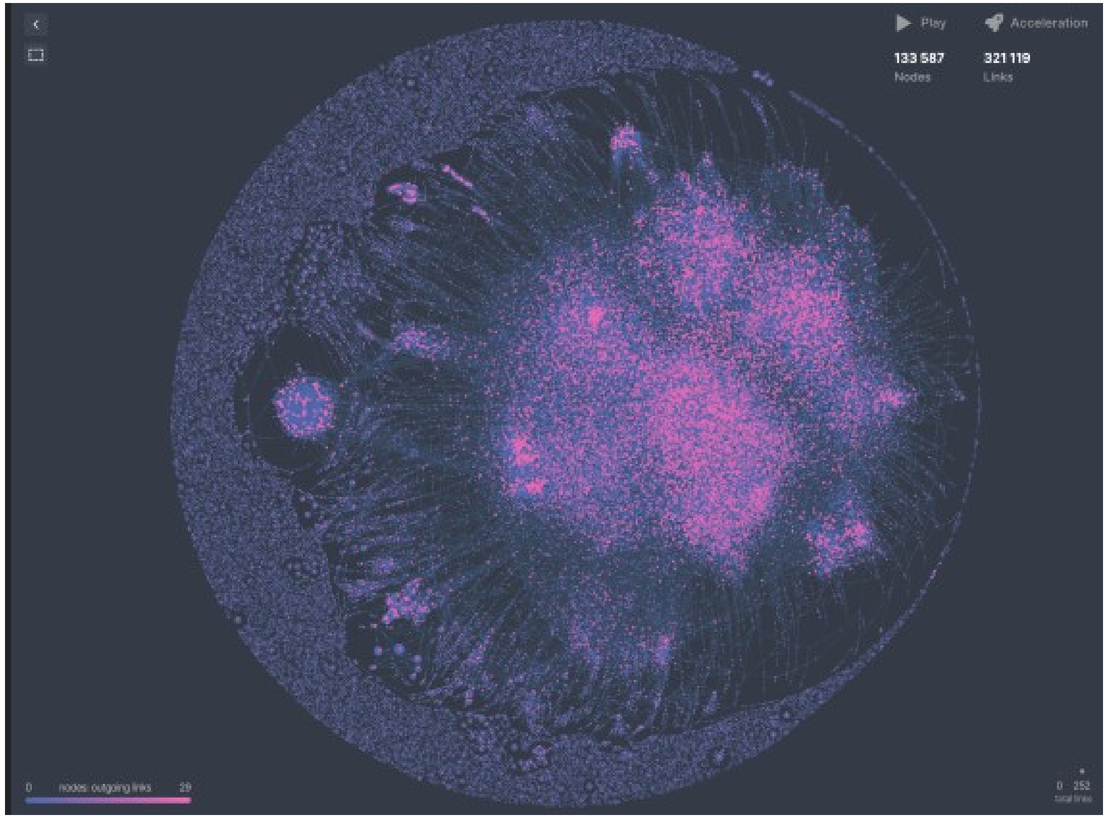


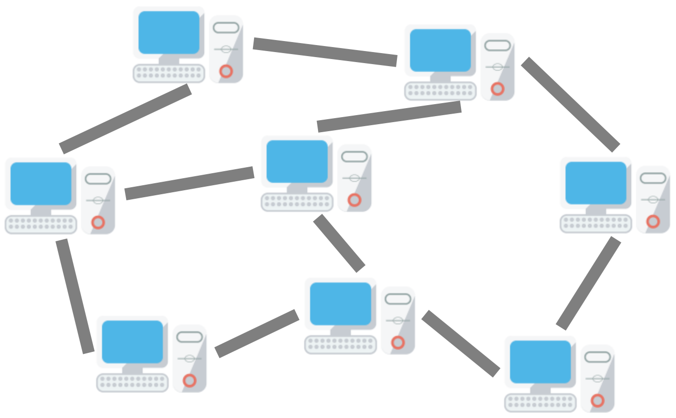
Figure source: https://nightingaledvs.com/how-to-visualize-a-graph-with-a-million-nodes/

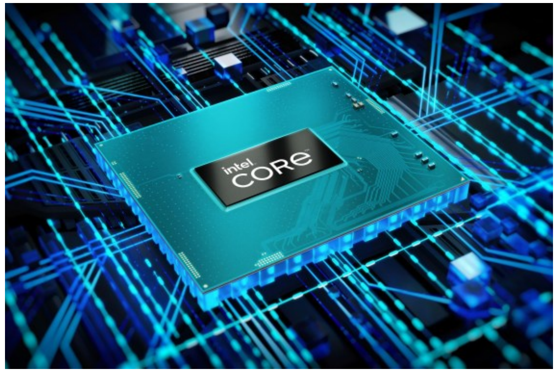
Web graphs [1]:
Billions of edges

Performance is important for *k*-core decomposition!

[1] Robert Meusel, Oliver Lehmberg, Christian Bizer, and Sebastiano Vigna. 2014. Web Data Commons — Hyperlink Graphs. http://webdatacommons.org/ hyperlinkgraph

Multi-core Parallelism





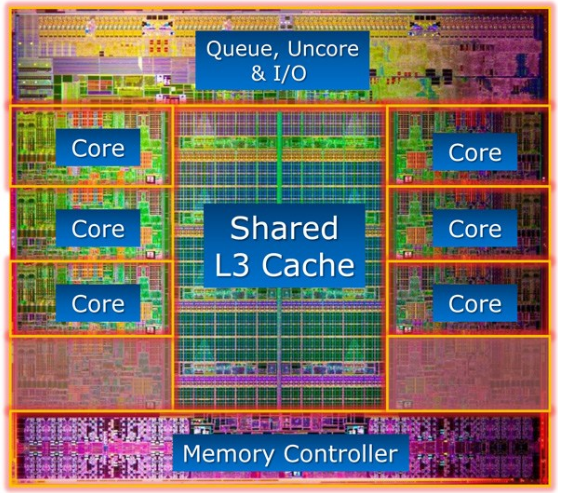


Figure credit: Wikipedia

<u>**Parallelism**</u> is ubiquitous and can be used to accelerate algorithms

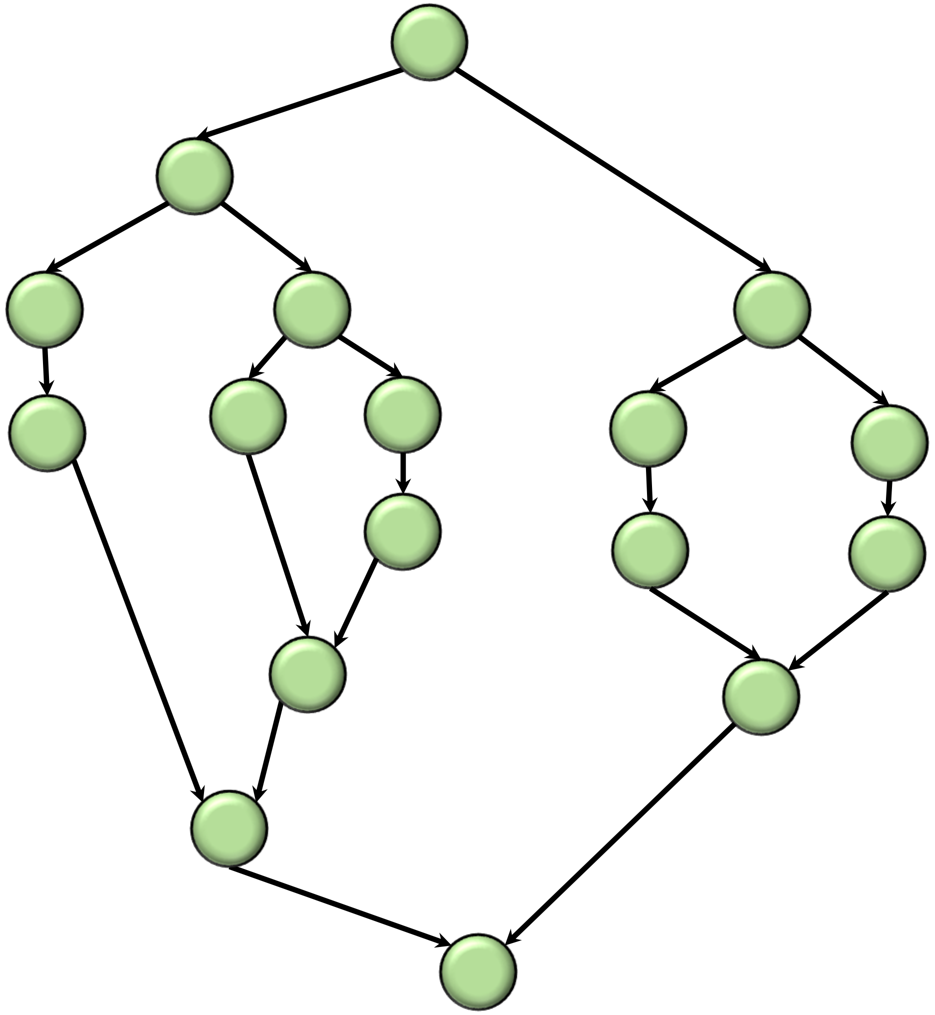
Parallel Computational Model [1]

- Shared-memory multi-core setting

- Work: the total number of operations
  - = running time on one core

- **Work-efficient**: The Work matches the complexity of the best sequential algorithm

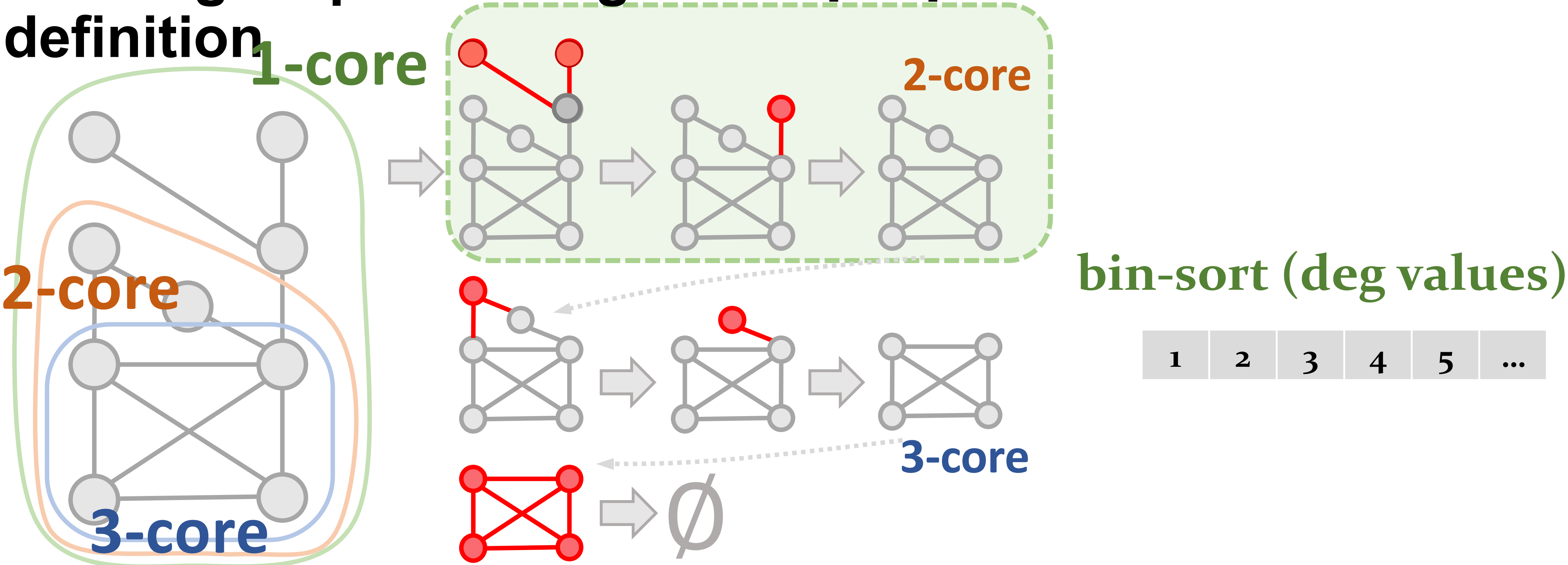
- Work-efficient is a primary goal for parallel algorithm design



[1] Guy E. Blelloch, Jeremy T. Fineman, Yan Gu, and Yihan Sun. 2020. Optimal parallel algorithms in the binary-forking model. In ACM Symposium on Parallelism in Algorithms and Architectures (SPAA). 89–102.

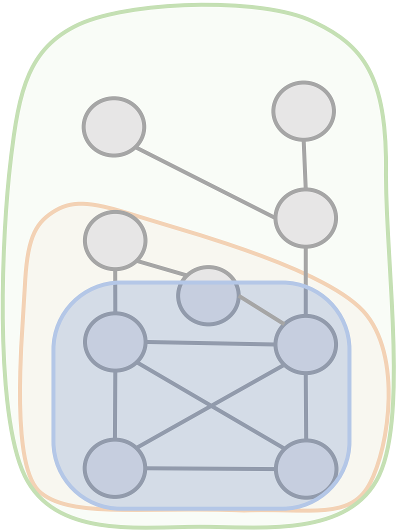
Challenges for Parallel k-core Solutions

- **Existing sequential algorithm [BZ03]: follows the definition**



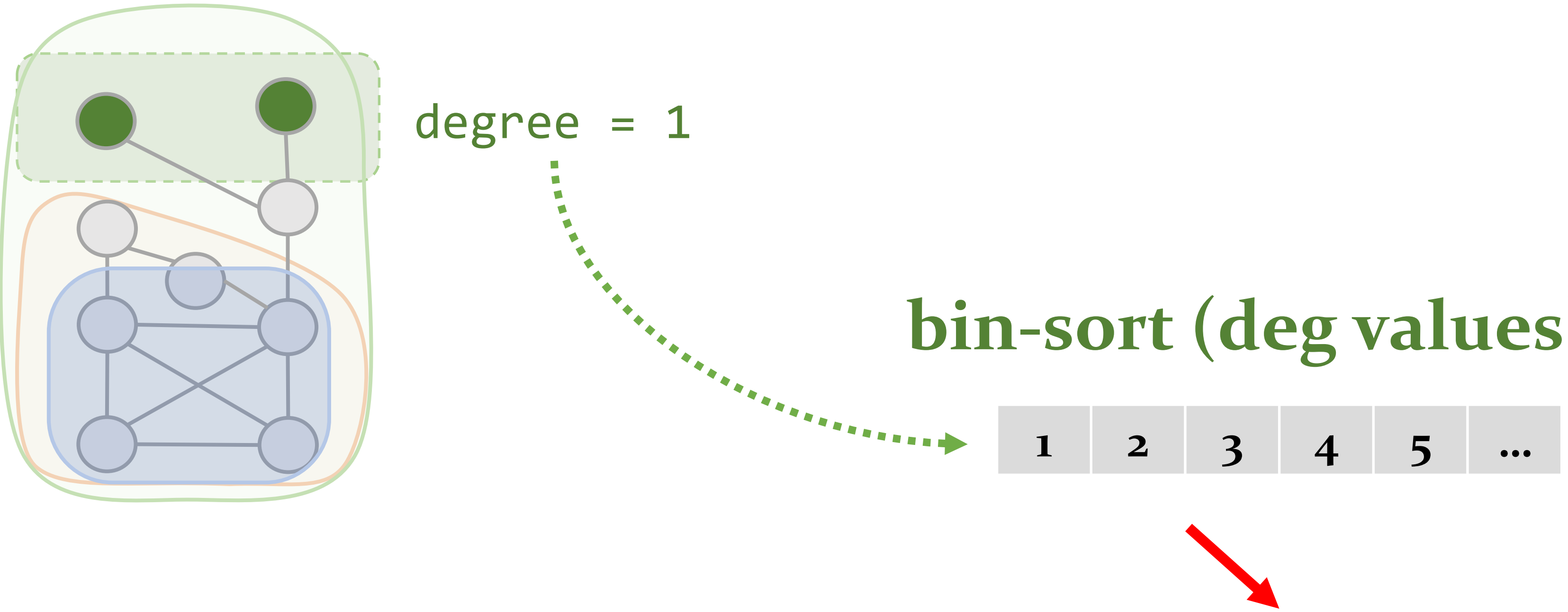
[BZ03] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) algorithm for cores decomposition of networks. arXiv preprint cs/0310049 (2003).

Challenges for Parallel k-core Solutions



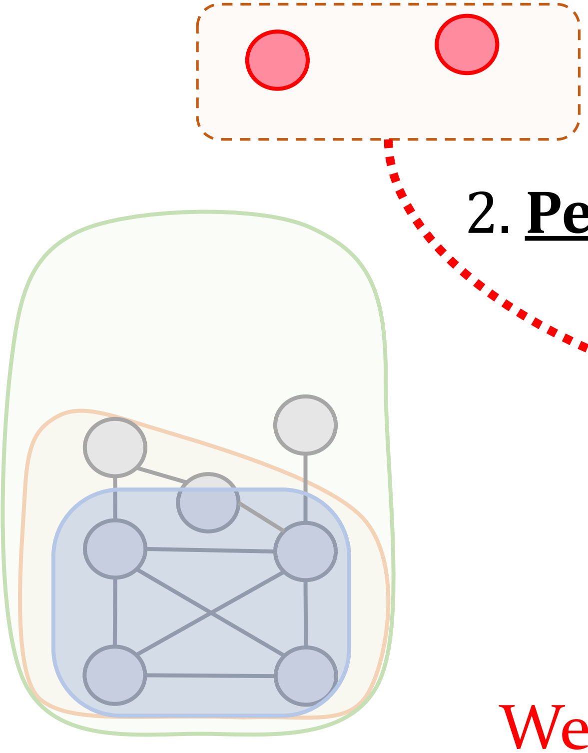
Challenges for Parallel k-core Solutions

1. <u>Framework</u>: How to **arrange & gather** the vertices with target degree values



The bin-sort-based array does not work in parallel

Challenges for Parallel k-core Solutions



2. **<u>Peeling in a subround</u>**: How to **peel** (remove) them **in parallel**

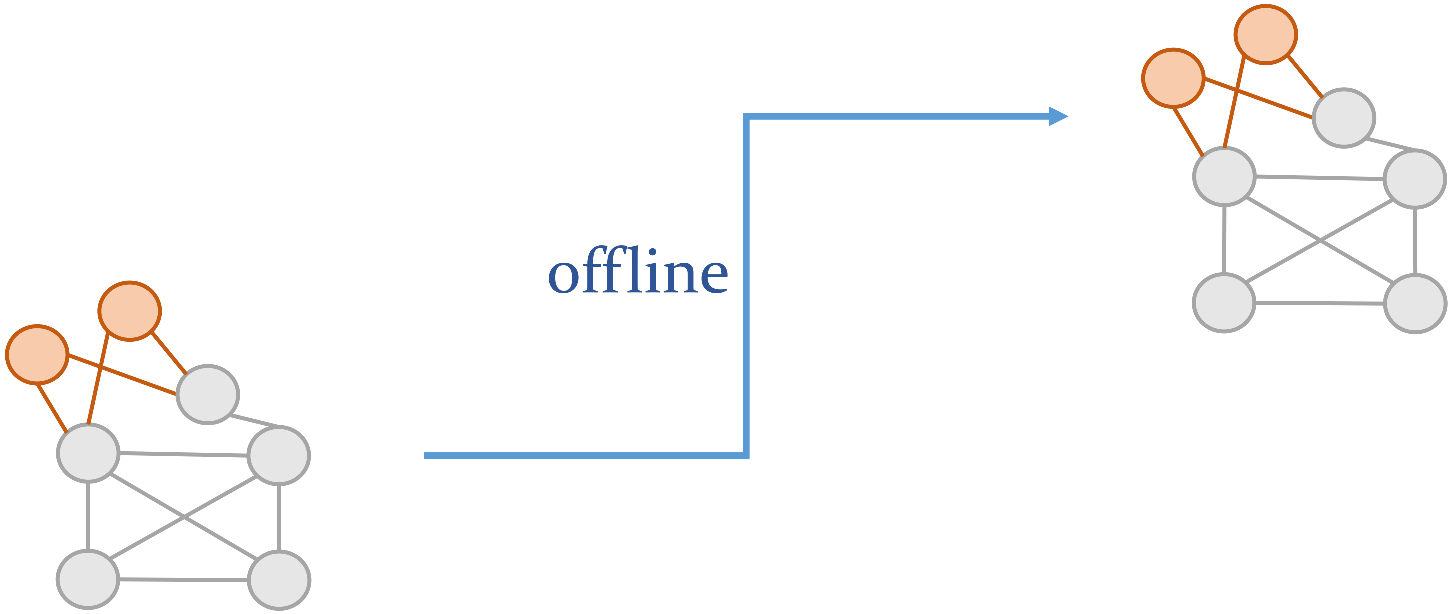
We need to process the **signals** of degree reduction in parallel

There are some existing parallel solutions

- Different parallel peeling strategies
  - Offline-peeling: Julienne[1]
  - Online-peeling: ParK[2], PKC[3]

[1] Laxman Dhulipala, Guy Blelloch, and Julian Shun. Julienne: A Framework for Parallel Graph Algorithms using Work-efficient Bucketing, SPAA ’17
[2]Dasari, Naga Shailaja et al. “ParK: An efficient algorithm for k-core decomposition on multicore processors.” *Big Data,* 2014
[3] H. Kabir and K. Madduri, "Parallel k-Core Decomposition on Multicore Platforms," *IPDPSW*, 2017

Offline & Online Peeling

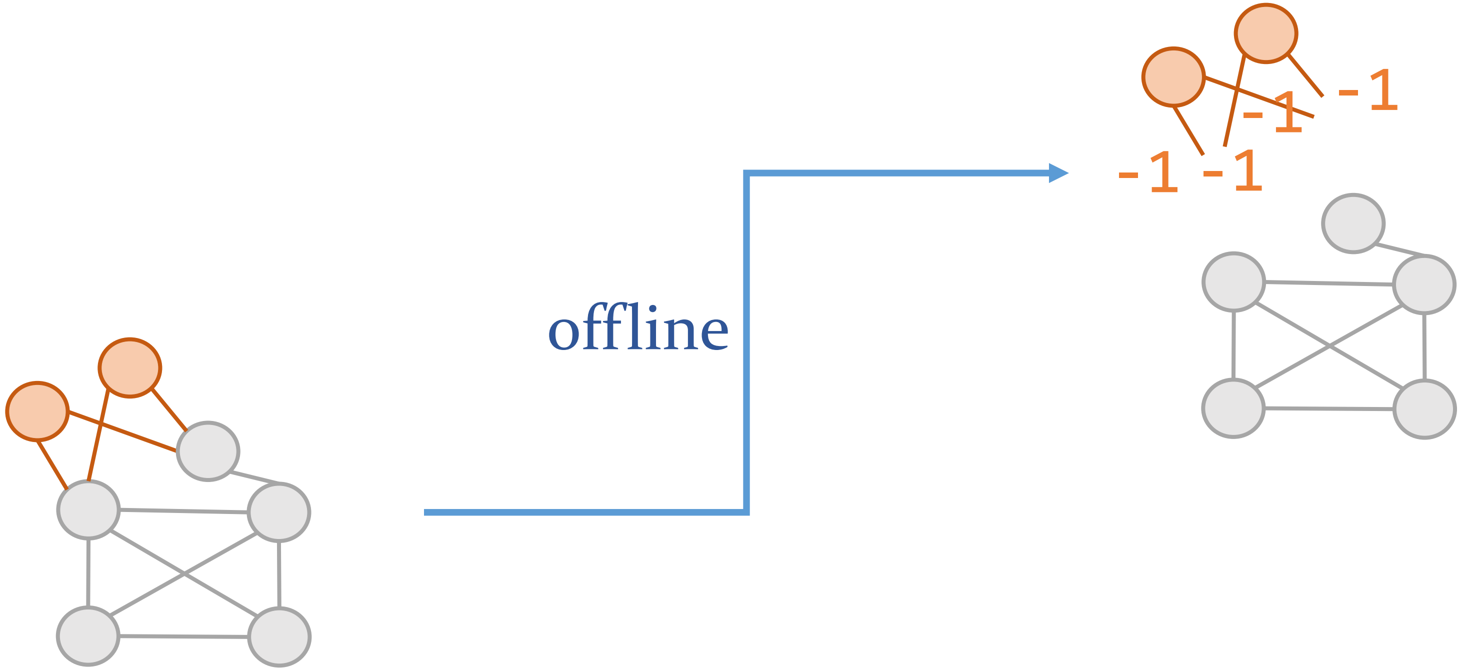


decompose to `3-core`

↓

peel vertices with degree <3

Offline & Online Peeling

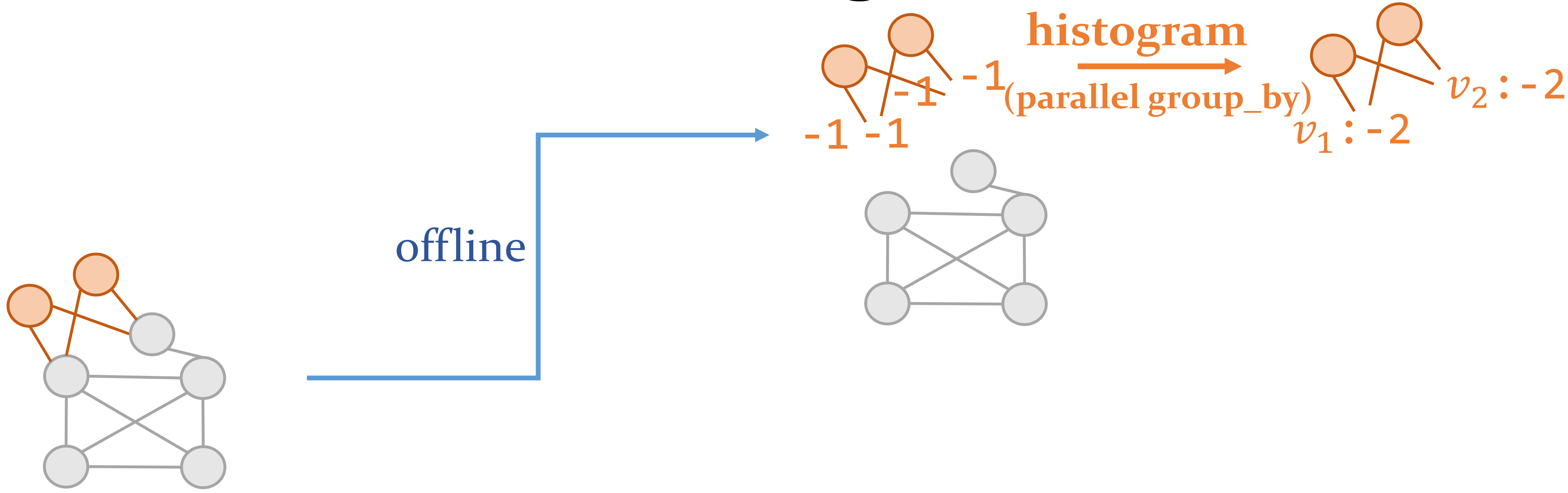


decompose to `3-core`

↓

peel vertices with degree `<3`

Offline & Online Peeling

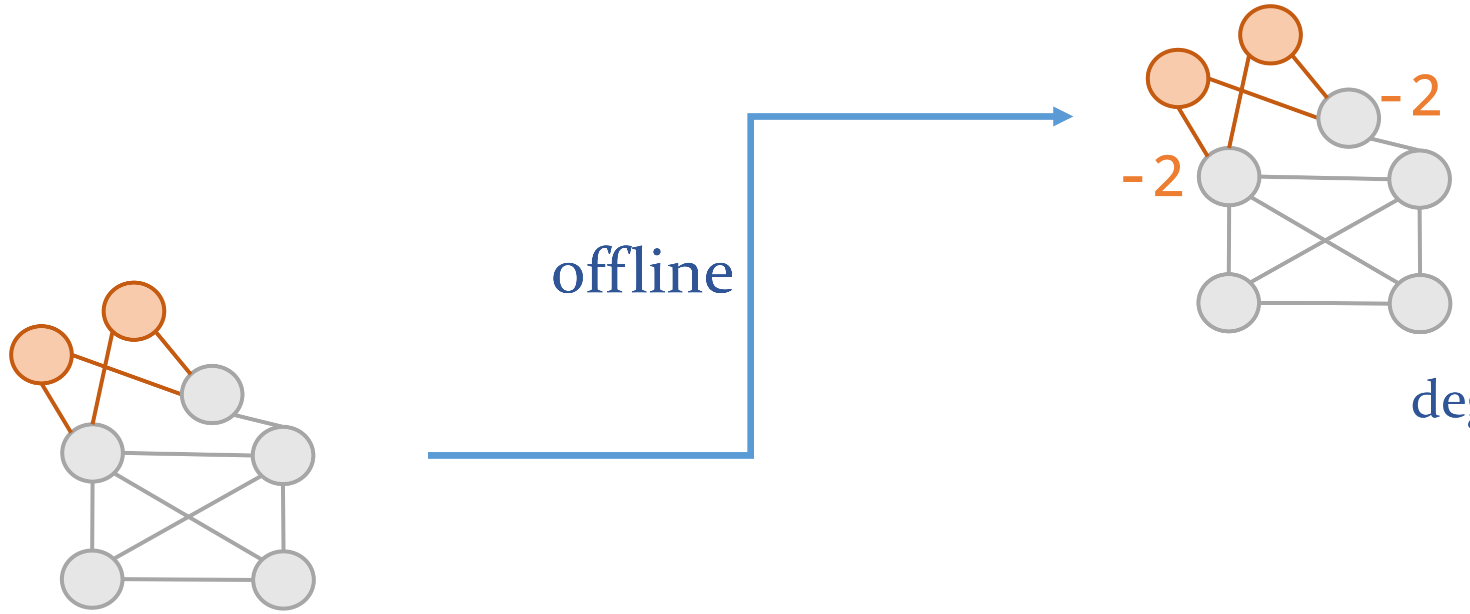


decompose to `3-core`

↓

peel vertices with degree `<3`

Offline & Online Peeling



key-value pairs

↓

parallel histogram
(in a batch)

↓

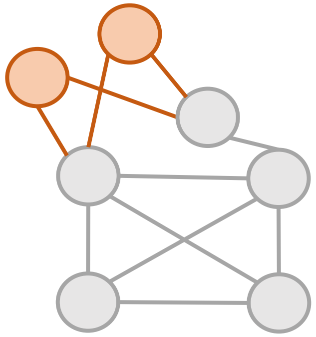
degree reduction of neighbors (keys)

decompose to `3-core`

↓

peel vertices with degree `<3`

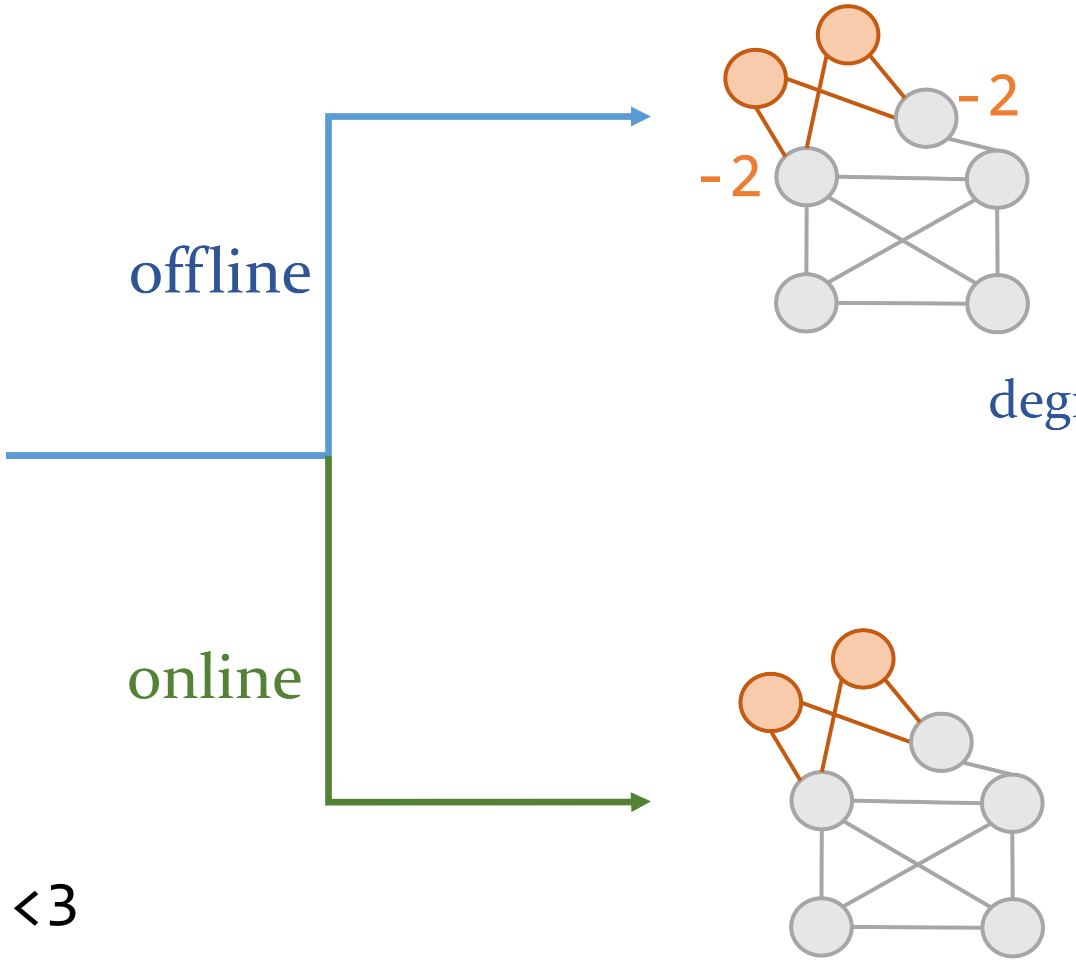
Offline & Online Peeling



decompose to `3-core`

↓

peel vertices with degree `<3`



key-value pairs

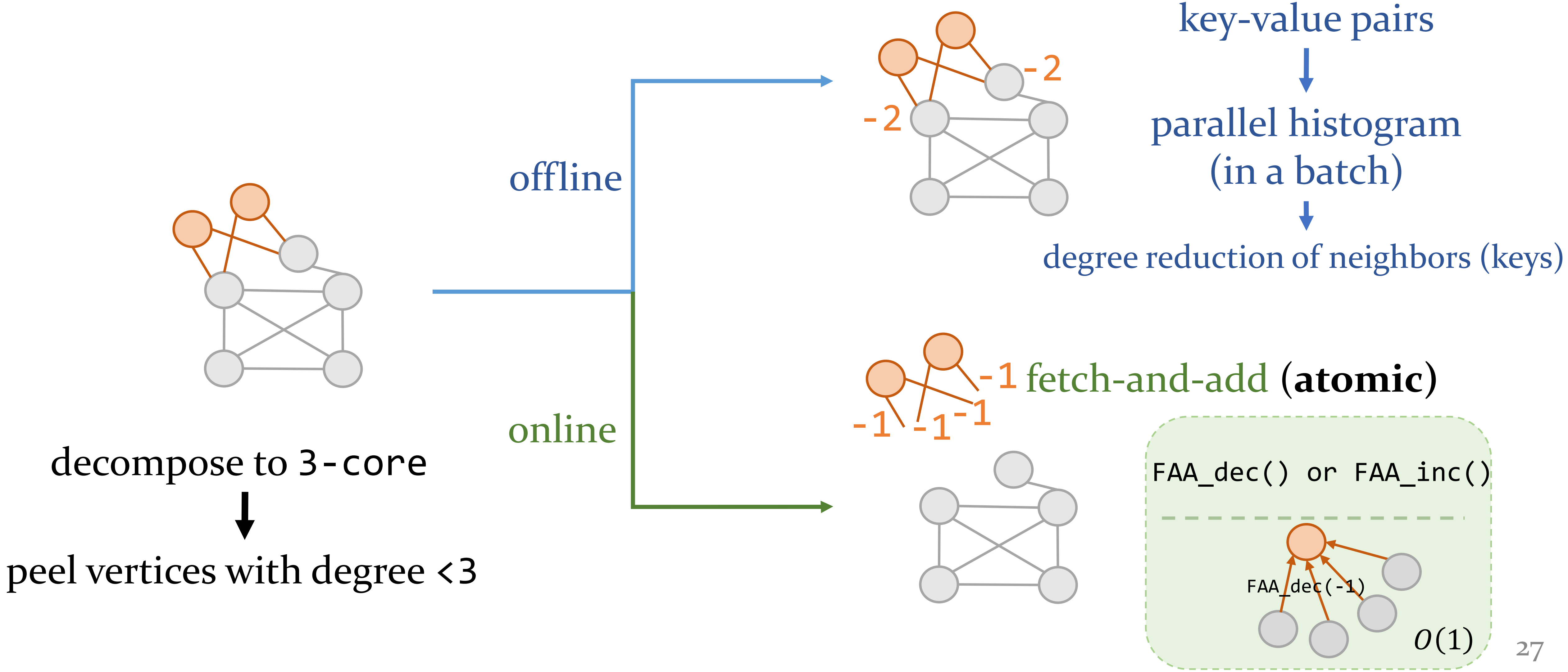
↓

parallel histogram
(in a batch)

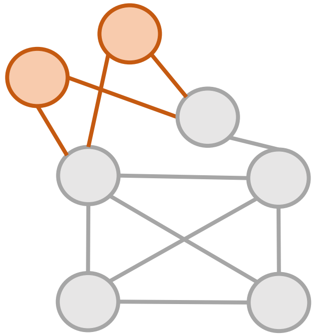
↓

degree reduction of neighbors (keys)

Offline & Online Peeling



Offline & Online Peeling

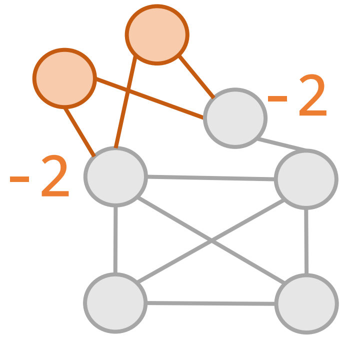


decompose to `3-core`

↓

peel vertices with degree `<3`

offline



key-value pairs

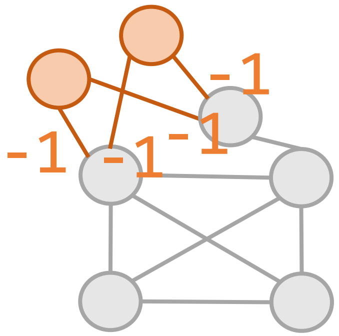
↓

parallel histogram
(in a batch)

↓

degree reduction of neighbors (keys)

online



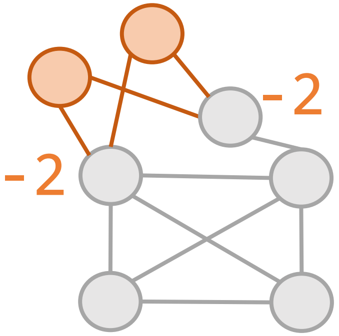
fetch-and-add

↓

decrease degree directly
(using atomic ops)

Offline & Online Peeling: Pros & Cons

offline



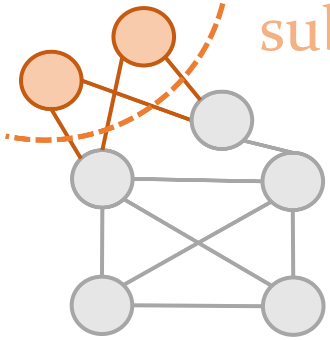
key-value pairs

↓

Parallel histogram

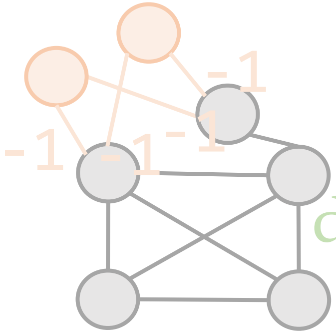
↓

degree reduction of neighbors (keys)



subround 1

online



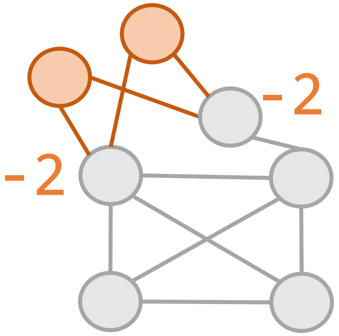
fetch-and-add

↓

decrease degree directly

Offline & Online Peeling: Pros & Cons

offline



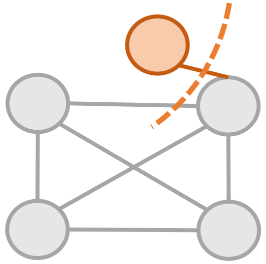
key-value pairs

↓

Parallel histogram

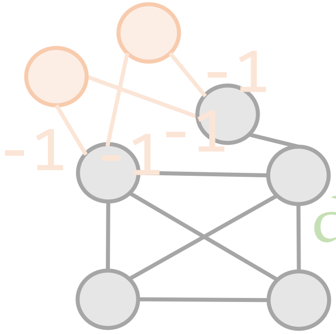
↓

degree reduction of neighbors (keys)



subround 2

online



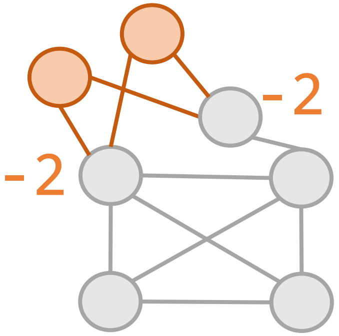
fetch-and-add

↓

decrease degree directly

Offline & Online Peeling: Pros & Cons

offline



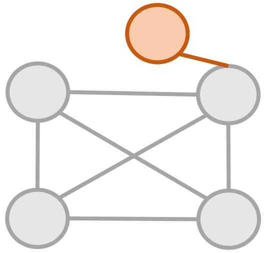
key-value pairs

↓

Parallel histogram

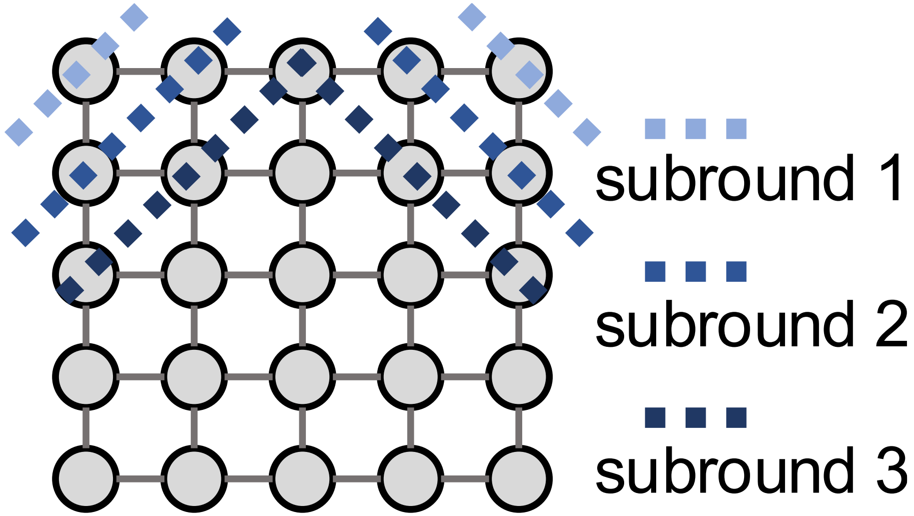
↓

degree reduction of neighbors (keys)

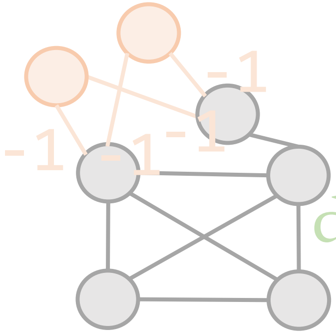


subround 2

can't avoid massive subrounds



online

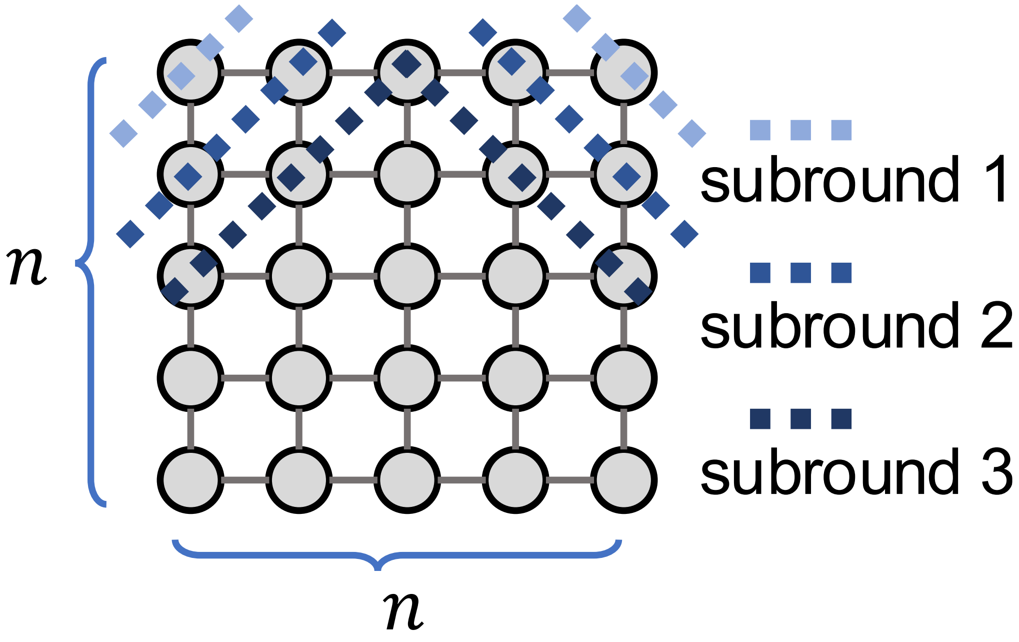


fetch-and-add

↓

decrease degree directly

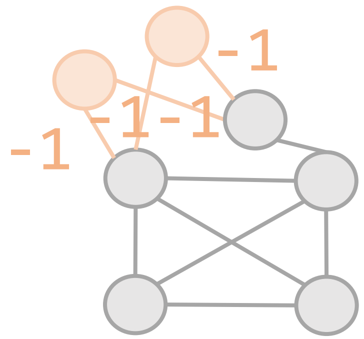
Offline & Online Peeling: Pros & Cons



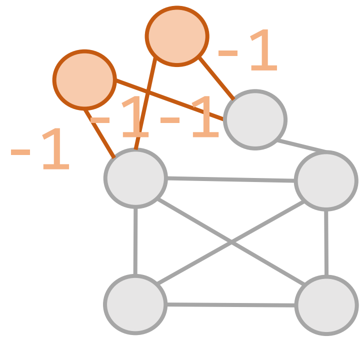
can’t avoid massive subrounds

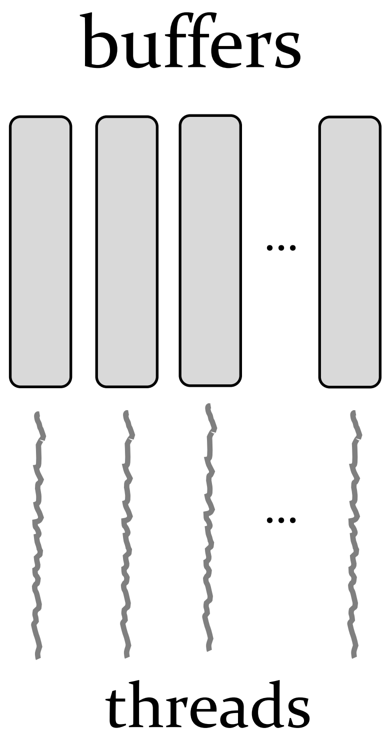
One subround = one round of synchronization overhead

Offline & Online Peeling: Pros & Cons

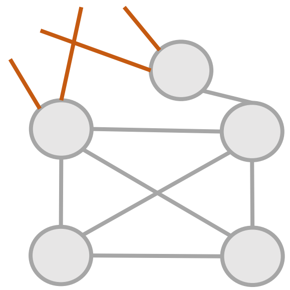


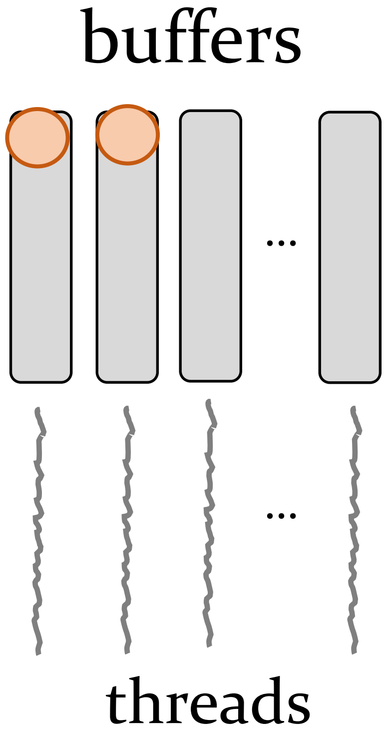
Offline & Online Peeling: Pros & Cons



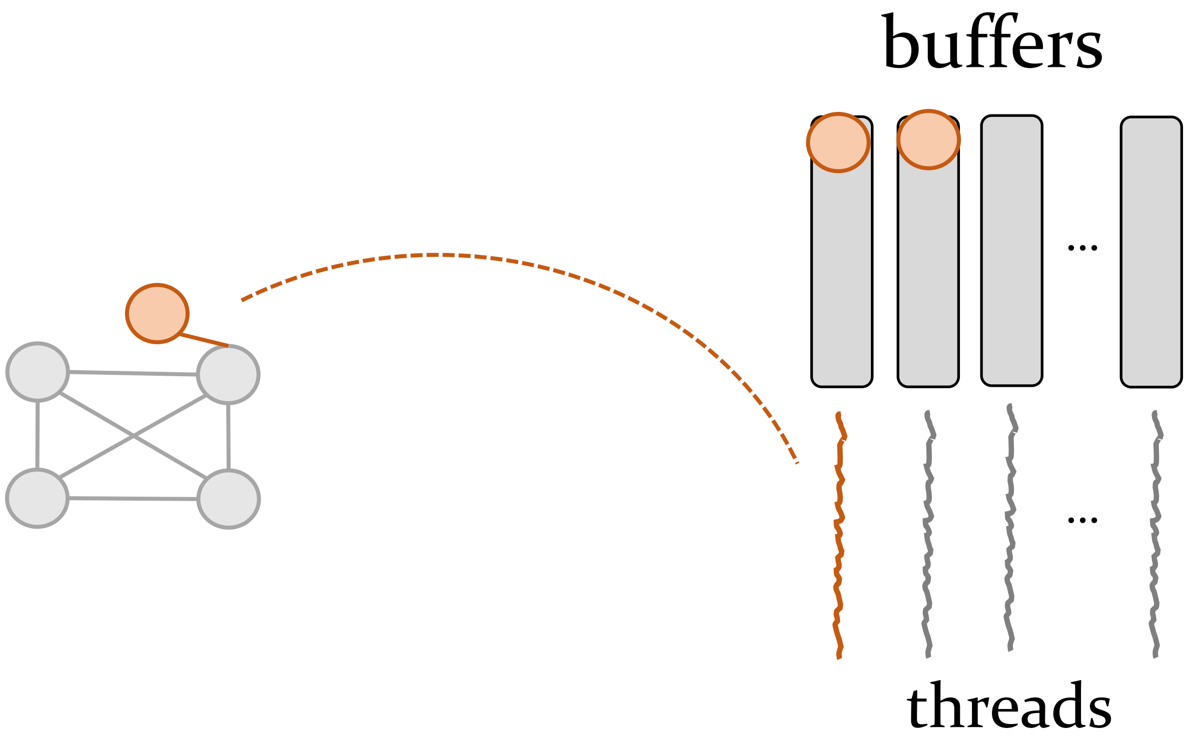


Offline & Online Peeling: Pros & Cons

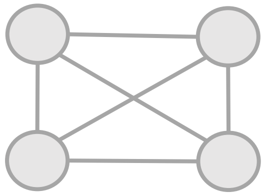


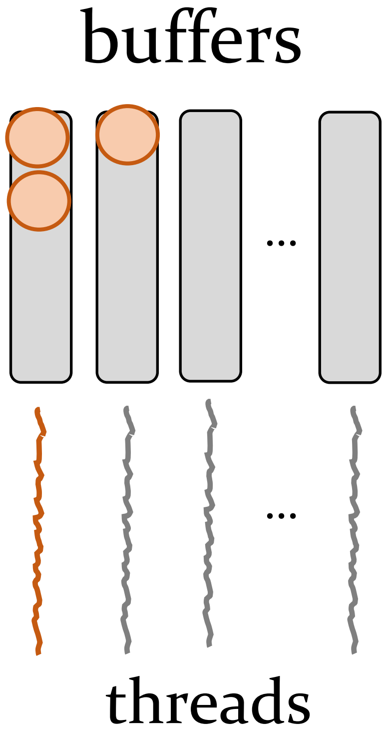


Offline & Online Peeling: Pros & Cons

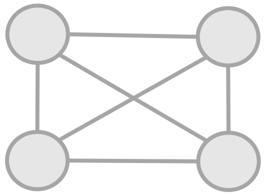


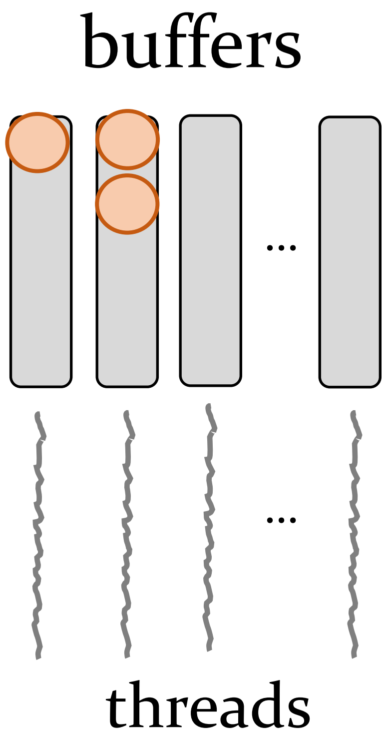
Offline & Online Peeling: Pros & Cons



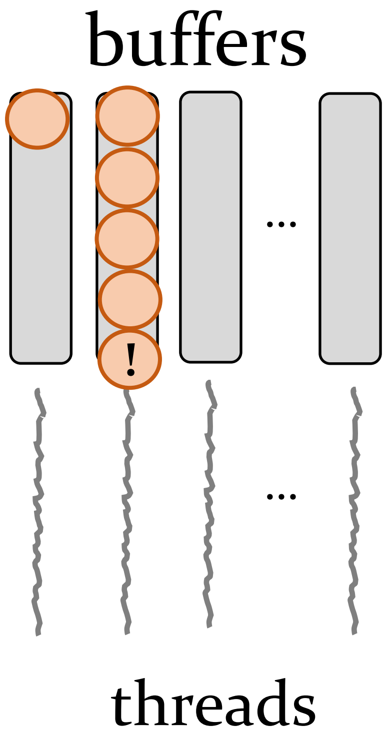


Offline & Online Peeling: Pros & Cons





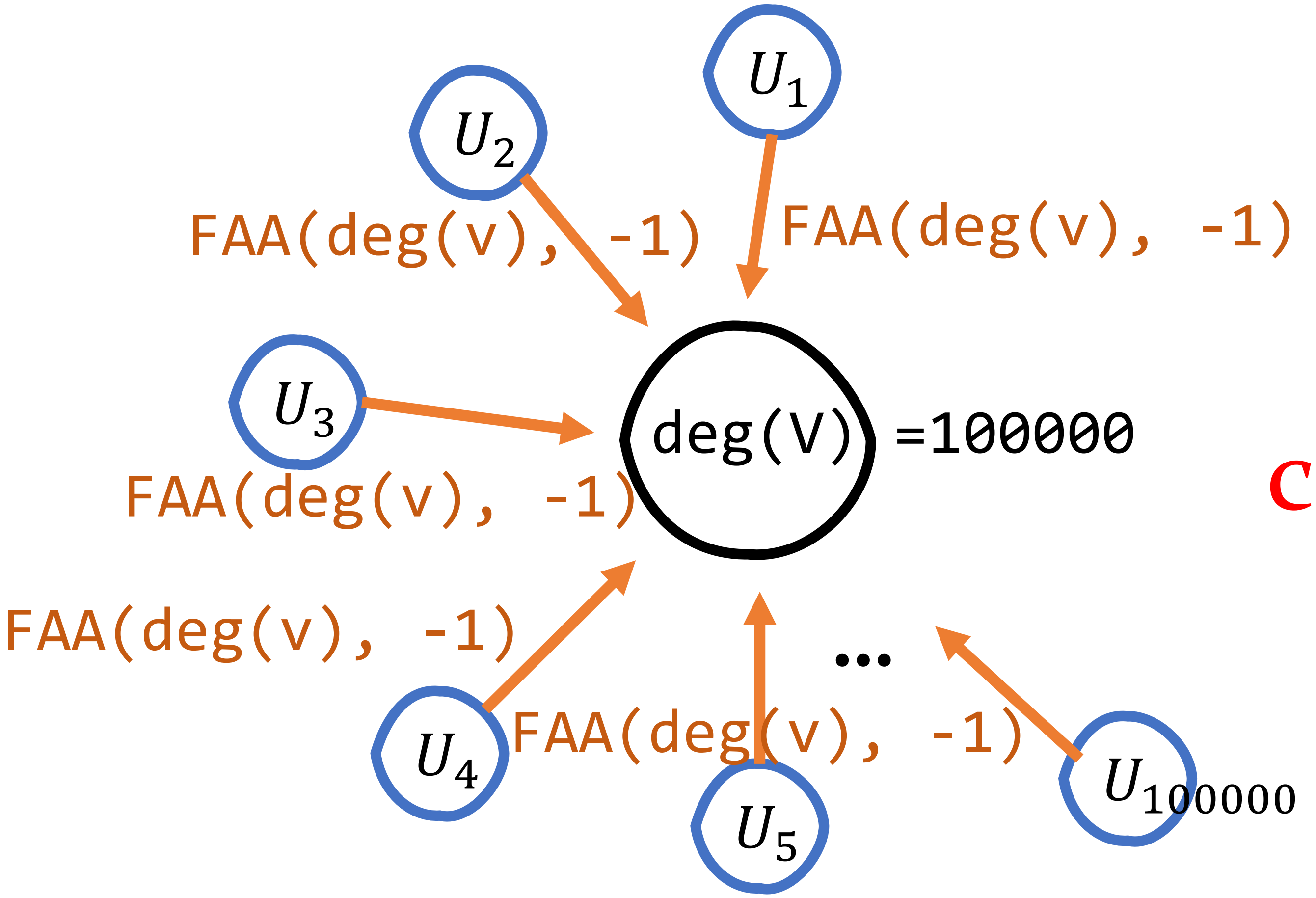
Offline & Online Peeling: Pros & Cons



Issue from the subround reduction

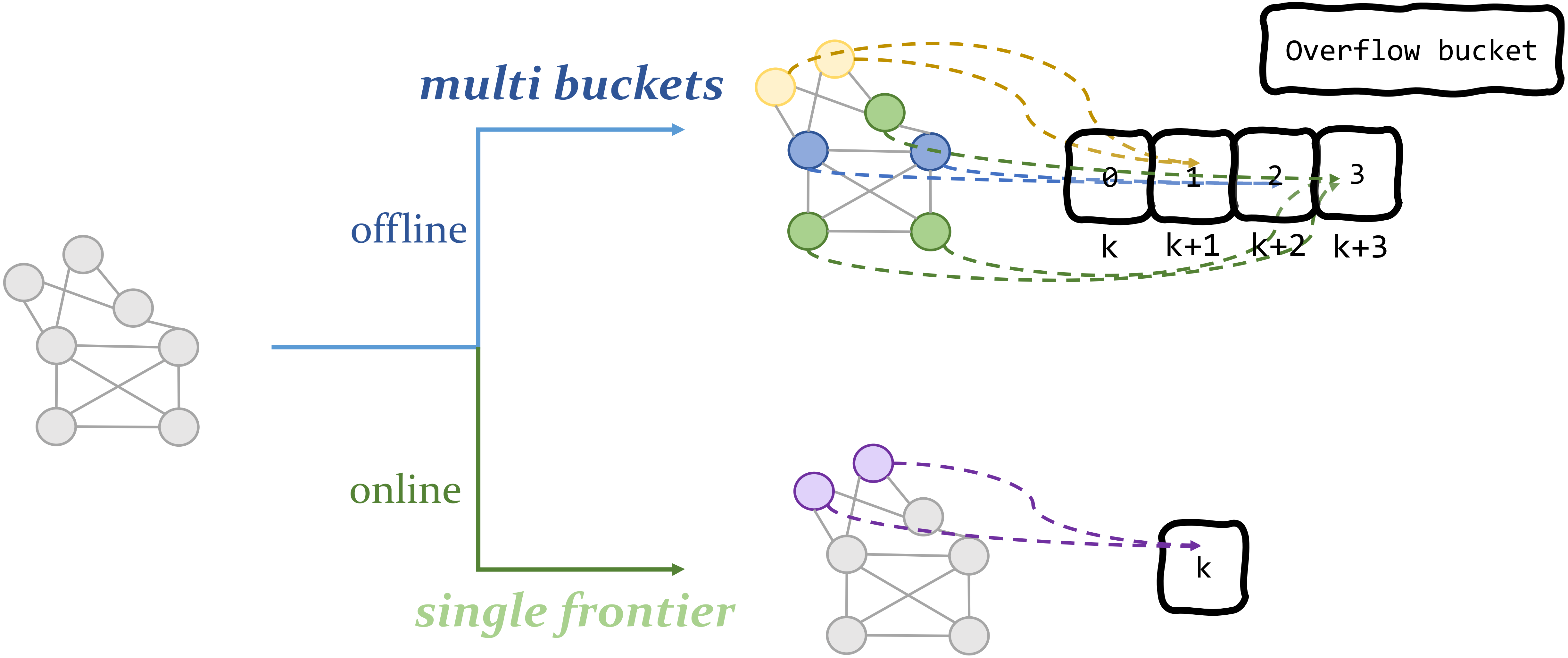
Buffers may be full, and it hurts load balance

Offline & Online Peeling: Pros & Cons

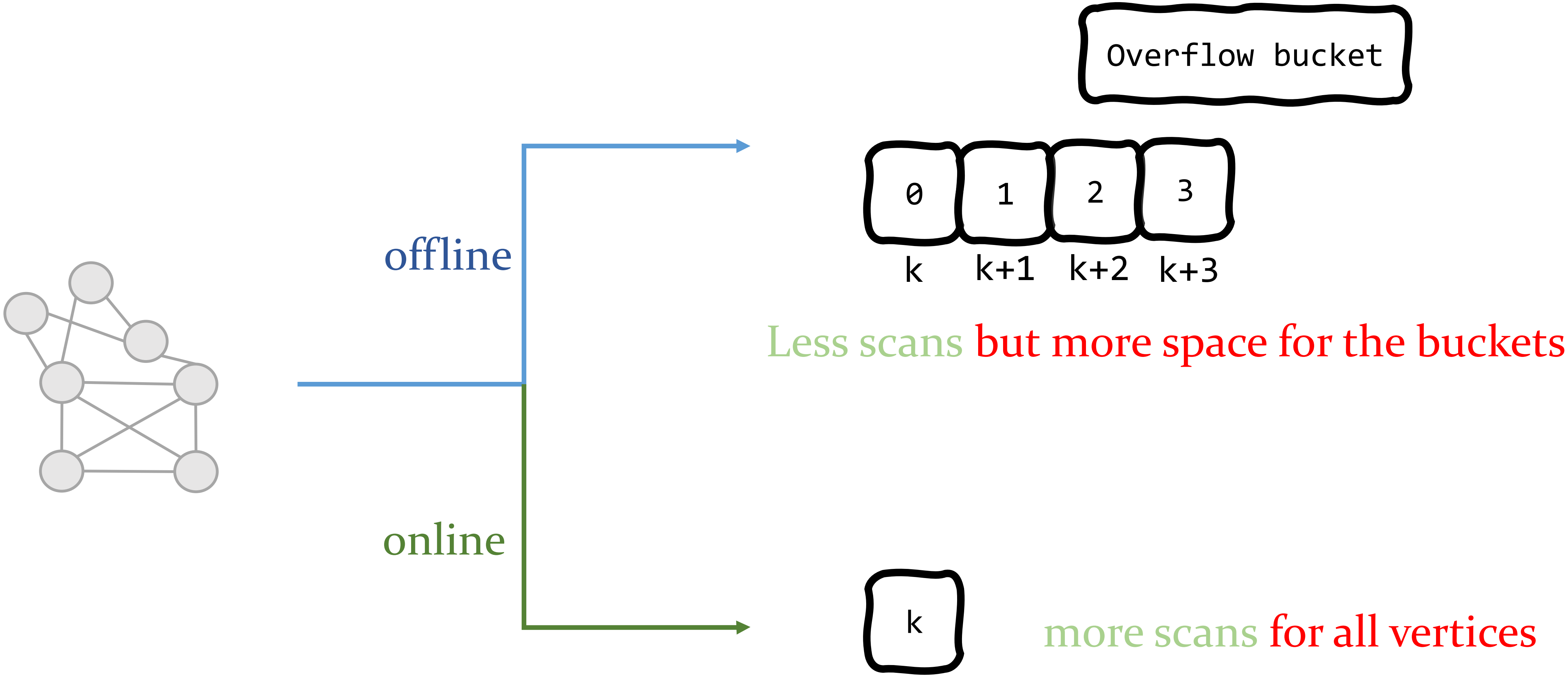


contention

The Second Challenge: How to arrange the vertices?



The Second Challenge: How to arrange the vertices?



Summary of Existing Solutions

- Existing **offline-peeling** solutions

Offline peeling (Julienne [DBS'17]**)**

Memory limitation for multi-bucket structure

Cannot avoid massive surrounds

- Existing **online-peeling** solutions

Online peeling (PKC [KM'17]**)**

Not work-efficient

Heavy contention

Load imbalance (for unbounded buffers)

[KM] H. Kabir and K. Madduri, "Parallel k-Core Decomposition on Multicore Platforms," *IPDPSW*, 2017
[DBS] Laxman Dhulipala, Guy Blelloch, and Julian Shun. Julienne: A Framework for Parallel Graph Algorithms using Work-efficient Bucketing, SPAA '17

Summary of Existing Solutions

Offline peeling (Julienne [DBS'17]**)**

- Memory limitation for multi-bucket structure
- Cannot avoid massive surrounds

→ Bad Performance on large-diameter graphs

Online peeling (PKC [KM'17]**)**

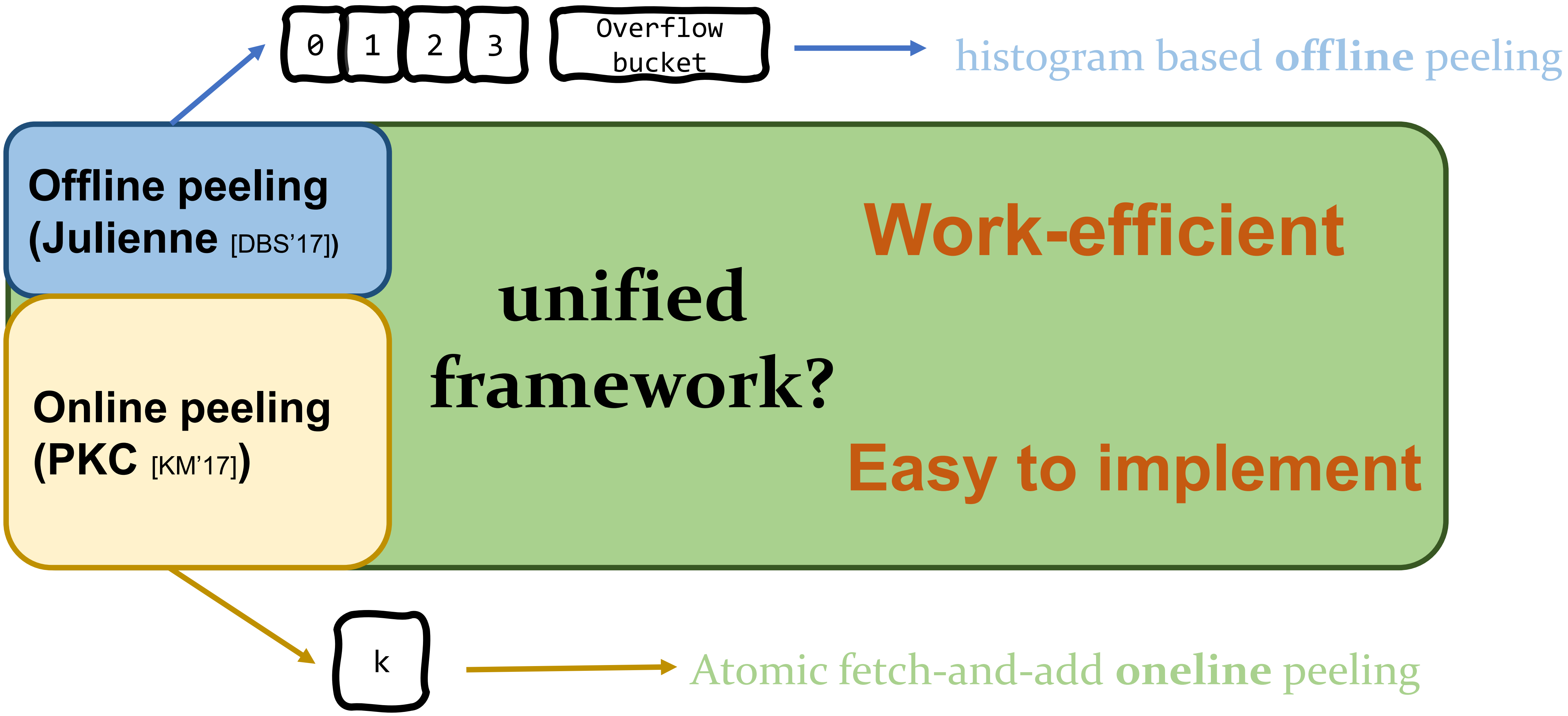
- Not work-efficient
- Heavy contention
- Load imbalance (for unbounded buffers)

→ Bad Performance on dense graphs

Can we get good performance on all types of graphs?

[KM] H. Kabir and K. Madduri, "Parallel k-Core Decomposition on Multicore Platforms," *IPDPSW*, 2017
[DBS] Laxman Dhulipala, Guy Blelloch, and Julian Shun. Julienne: A Framework for Parallel Graph Algorithms using Work-efficient Bucketing, SPAA '17

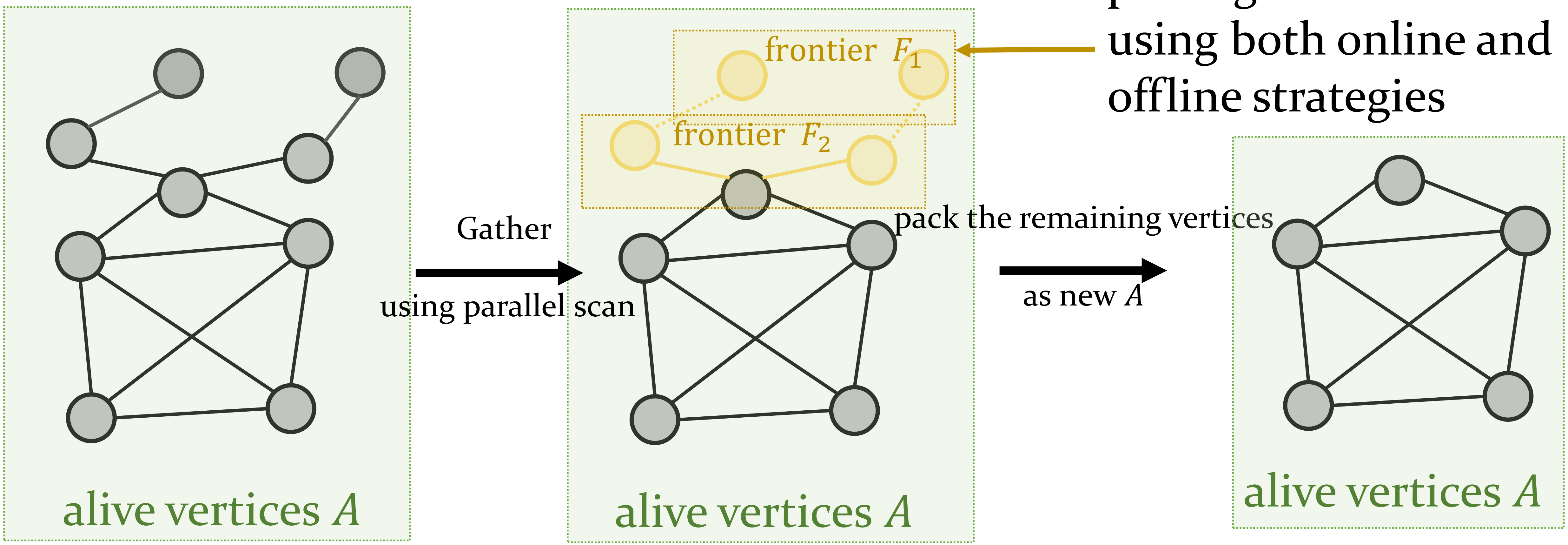
Summary of Existing Solutions



[KM] H. Kabir and K. Madduri, "Parallel k-Core Decomposition on Multicore Platforms," *IPDPSW*, 2017
[DBS] Laxman Dhulipala, Guy Blelloch, and Julian Shun. Julienne: A Framework for Parallel Graph Algorithms using Work-efficient Bucketing, SPAA '17

Our k-core decomposition framework

peeling subrounds using both online and offline strategies



Our Framework: adaptive for both peeling strategies

Our framework

Previous online peeling
(`FAA`)

high-volume concurrent
atomic operations

1. Heavy contention
2. Load imbalance

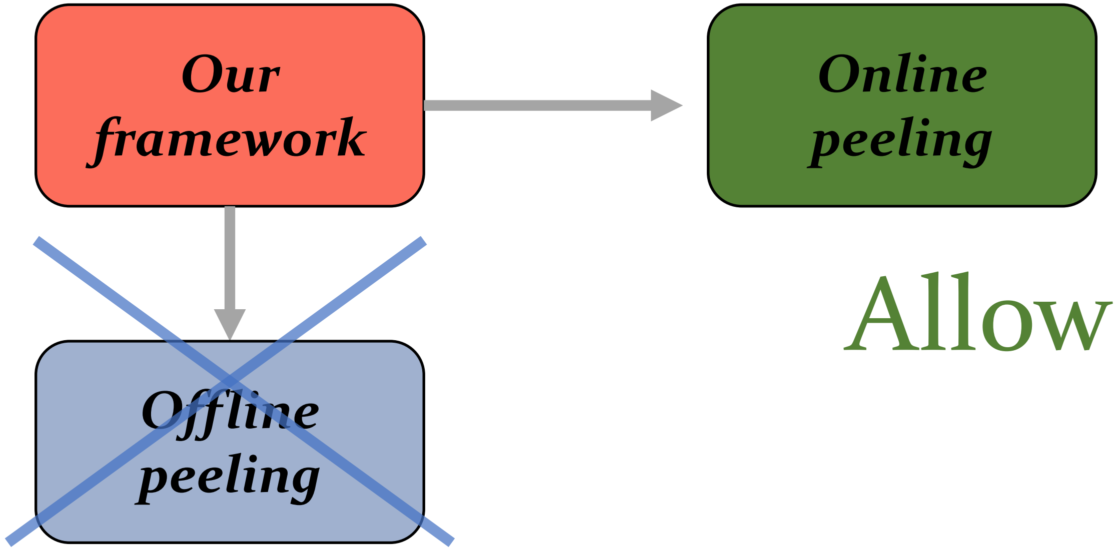
(for unbounded buffer)

Previous offline peeling
(`histogram`)

cannot avoid massive subrounds

Bad performance on graphs with a large number subrounds

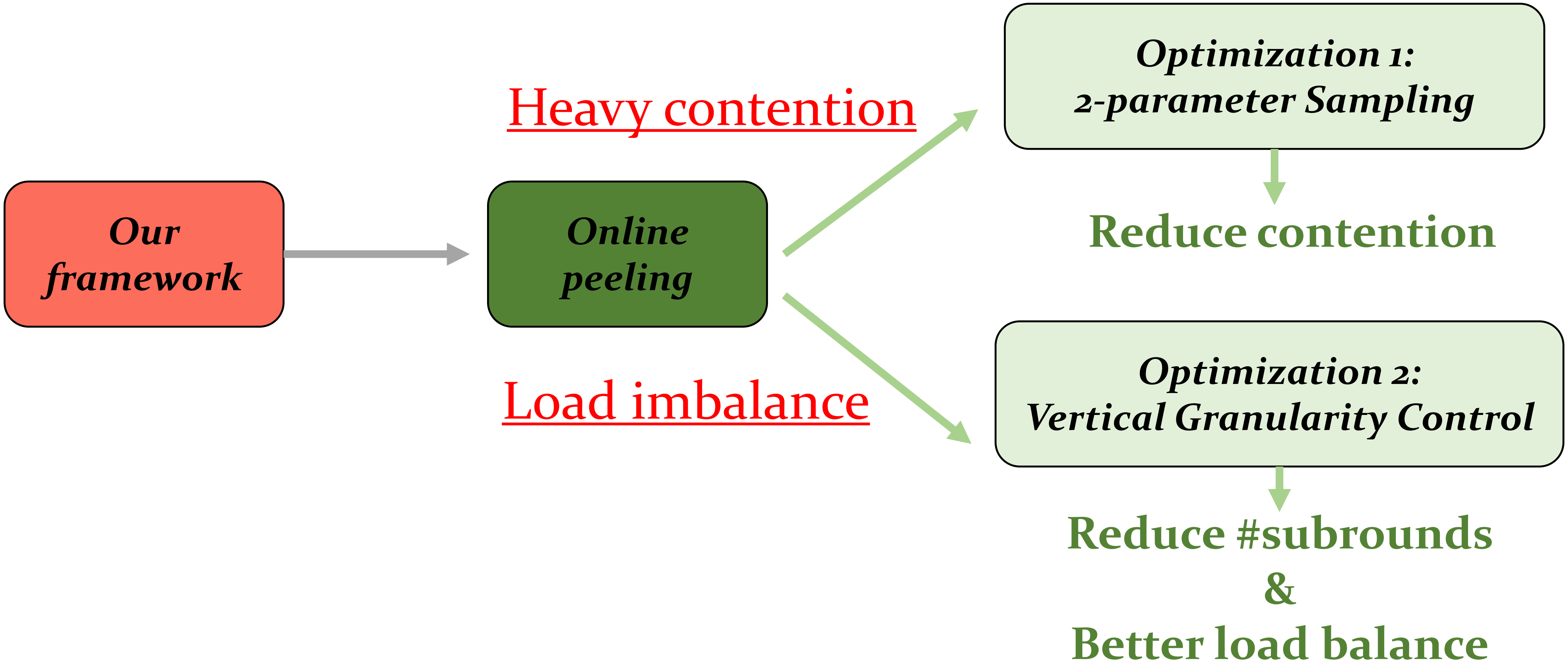
Our Algorithm: Optimized Online-peeling



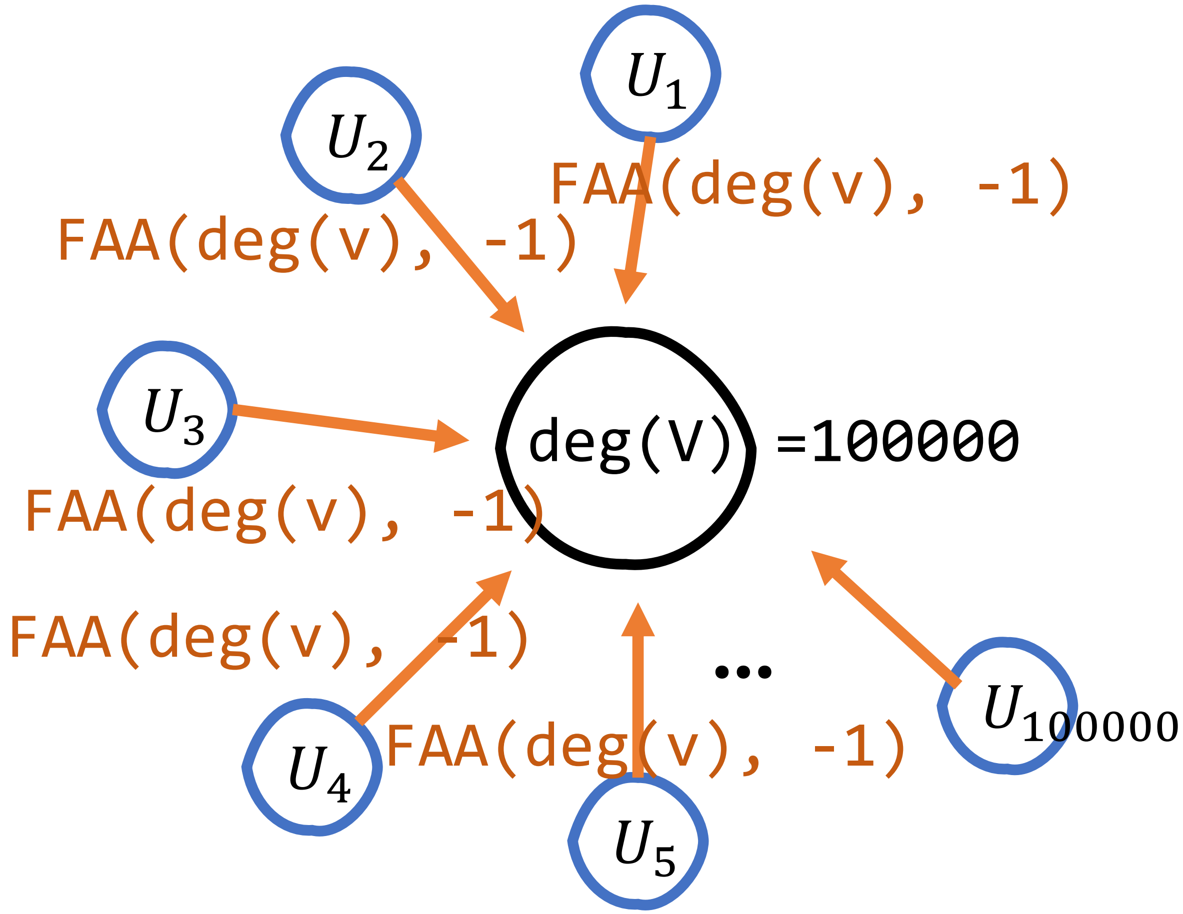
Allow multiple optimizations!

Inherited issue: a large number of subrounds

Our Algorithm: Optimized Online-peeling

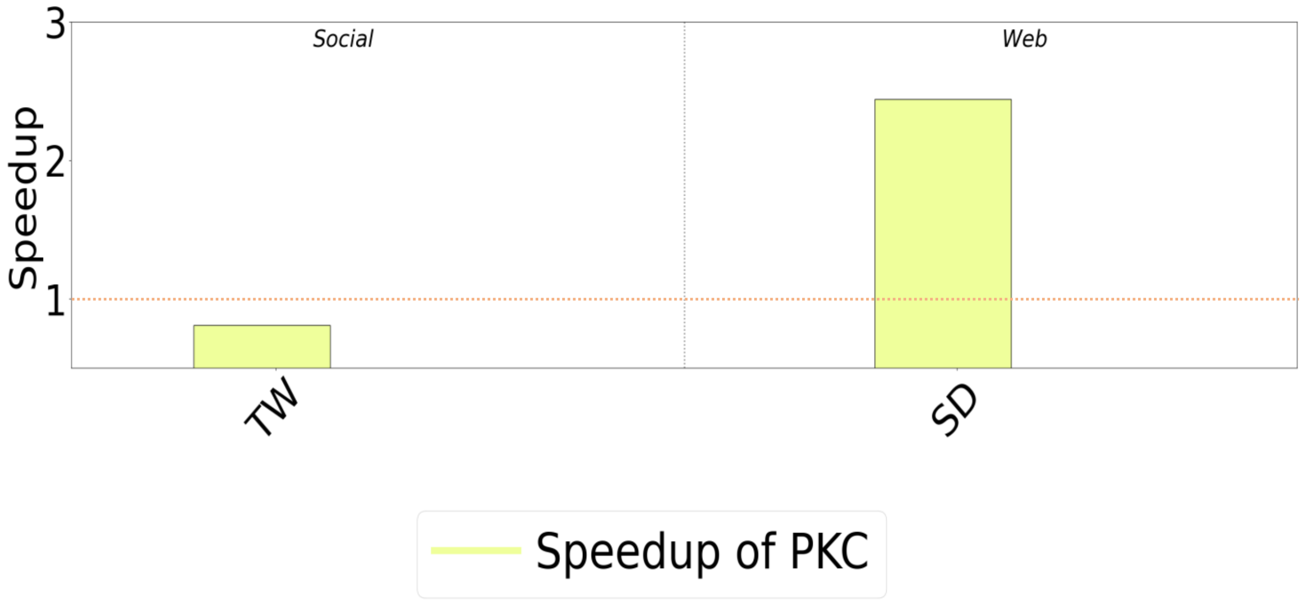


How can we reduce the contention?



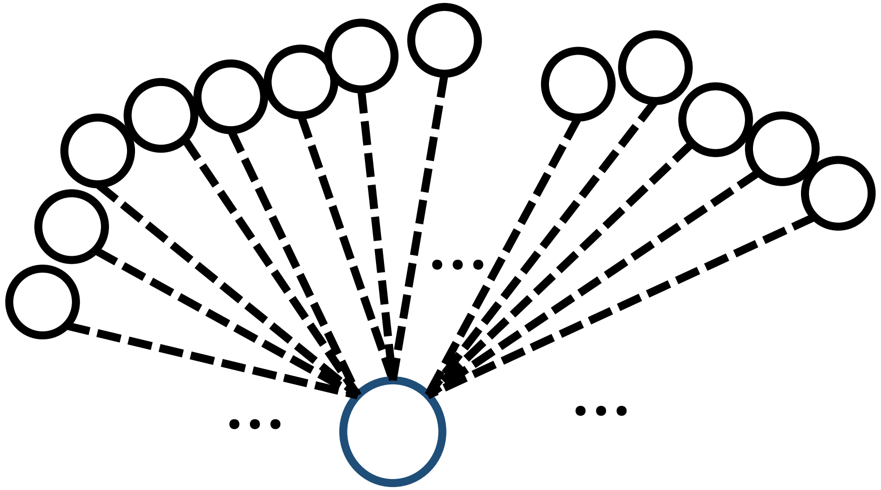
contention

Over the best sequential



Technique 1: Sampling

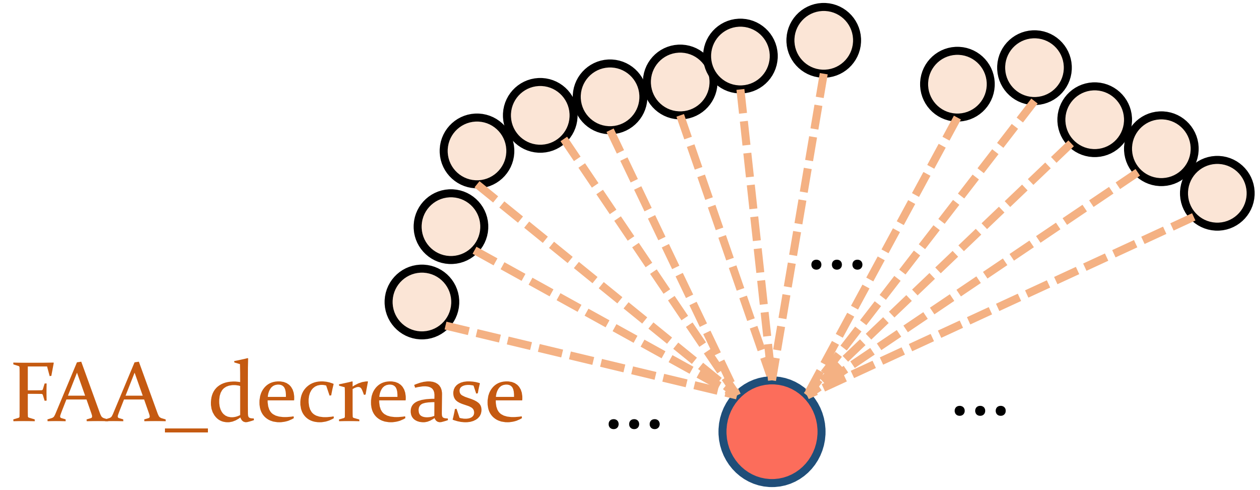
Optimization 1:
Sampling



High-volume contention while `FAA_dec`

Technique 1: Sampling

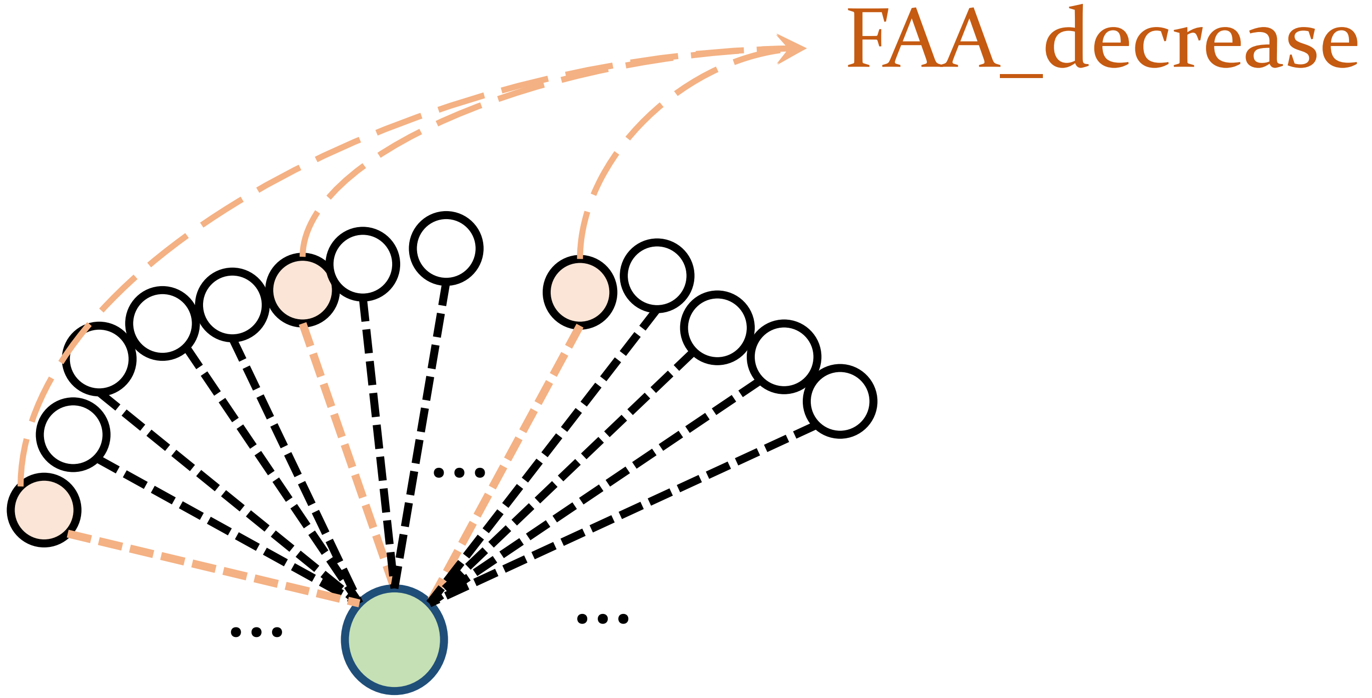
Optimization 1: Sampling



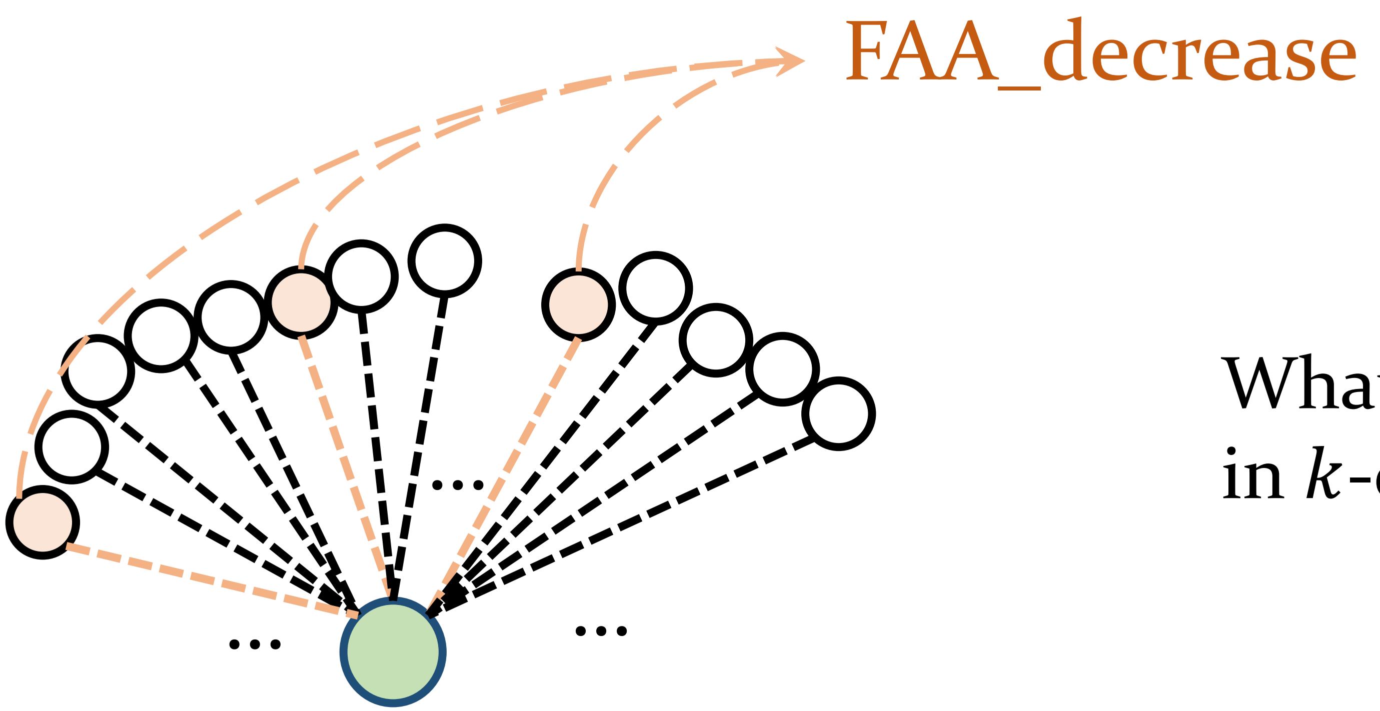
High-volume contention

Technique 1: Sampling

Optimization 1: Sampling

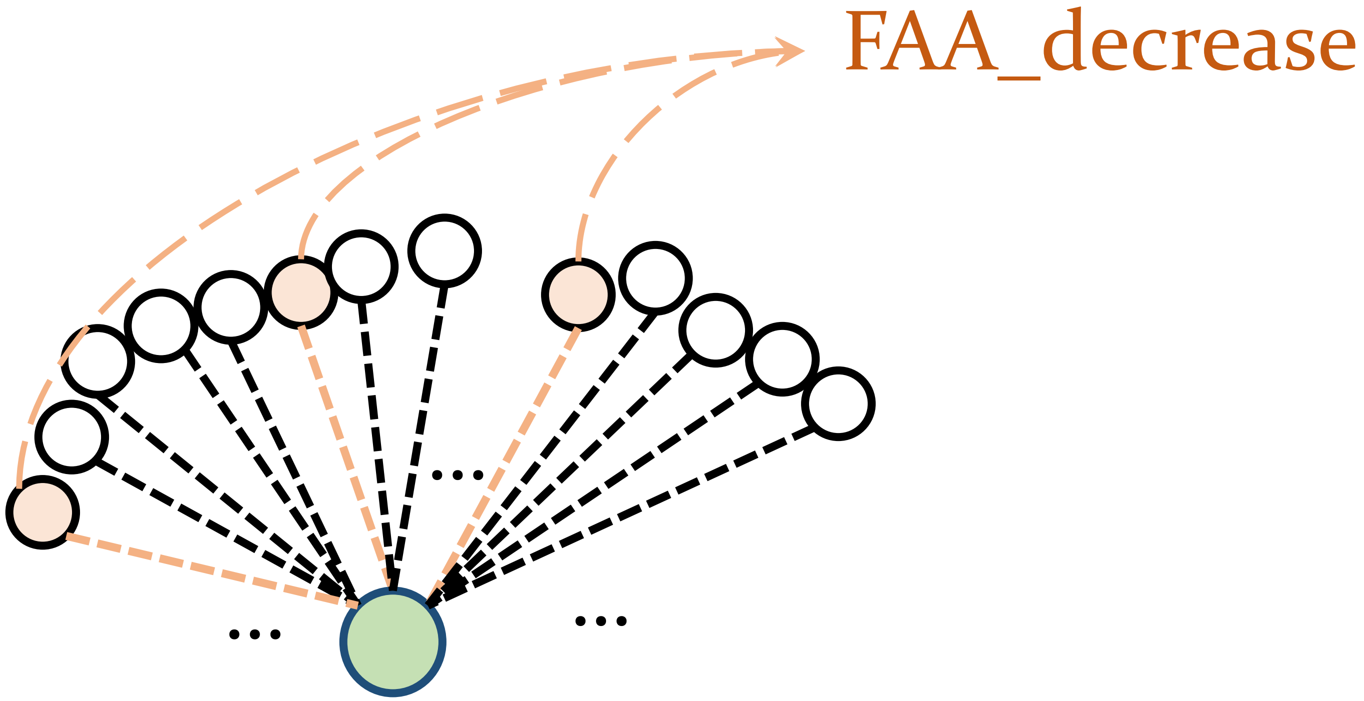


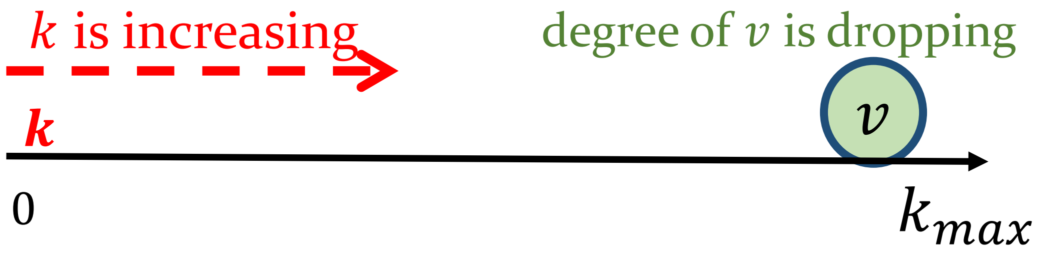
Technique 1: Sampling



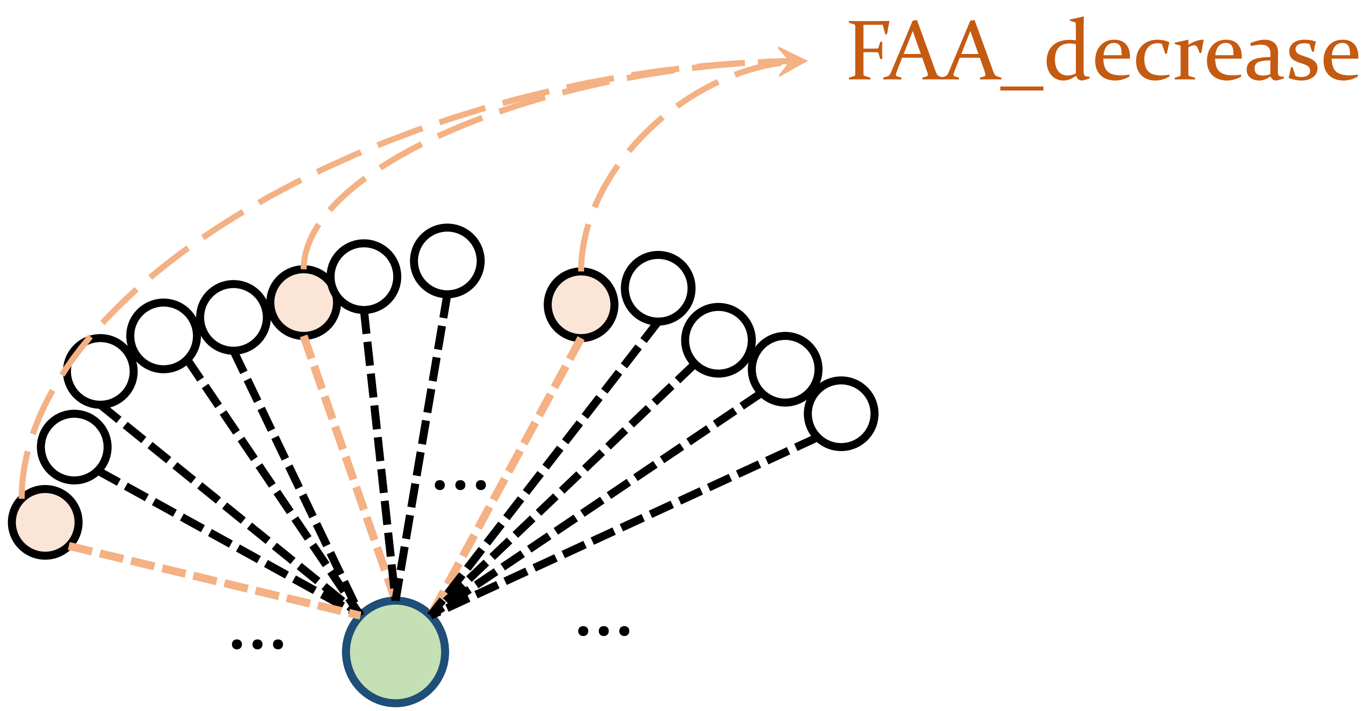
What is the difficulty of sampling in k-core decomposition?

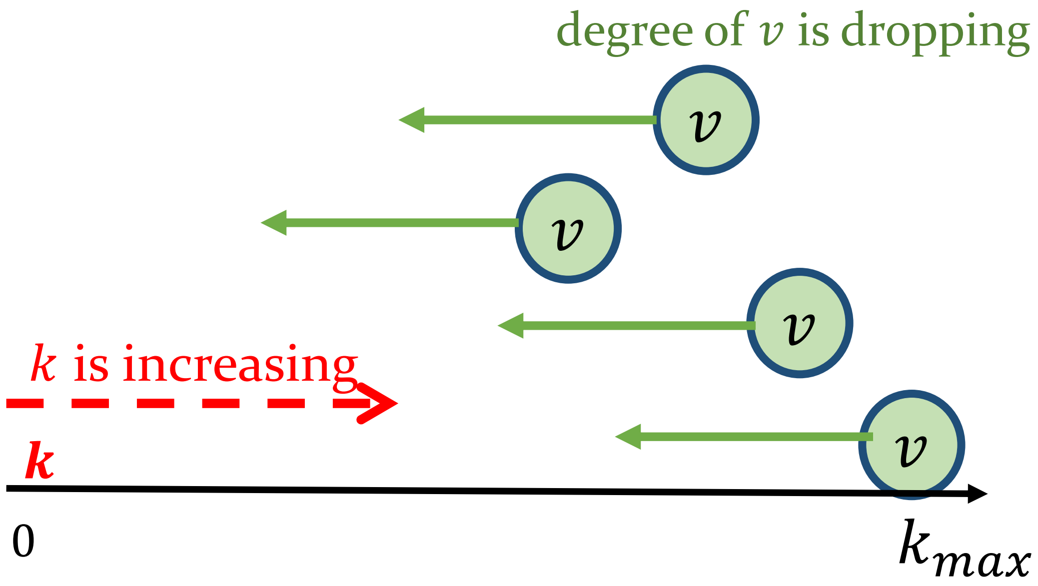
Technique 1: Sampling





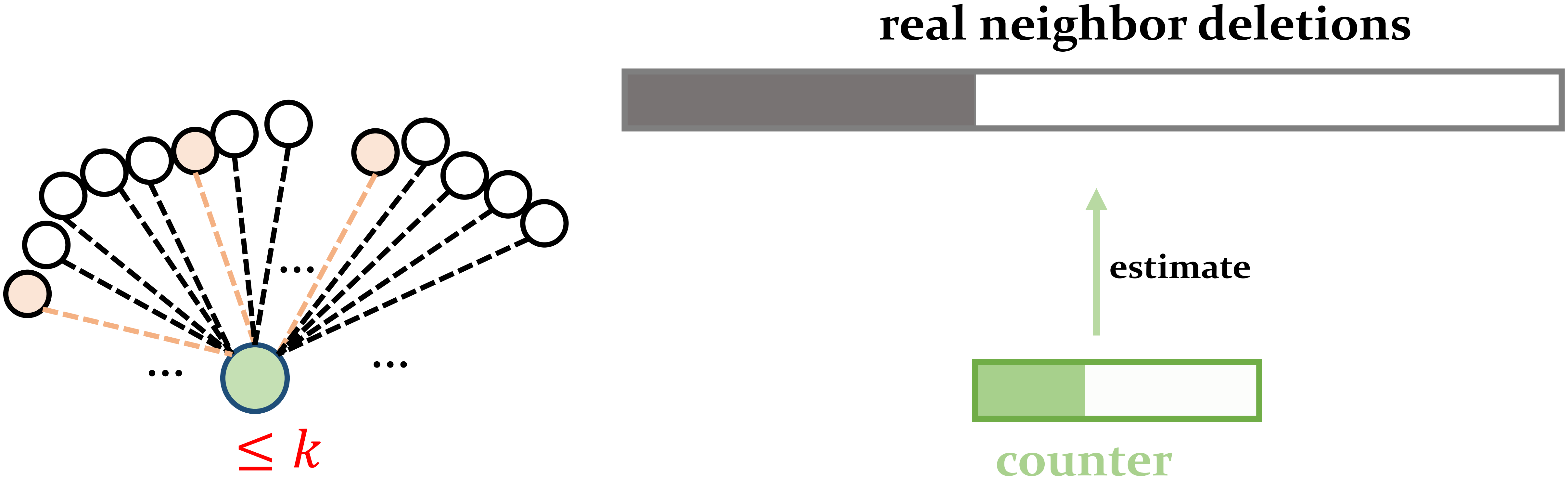
Technique 1: Sampling





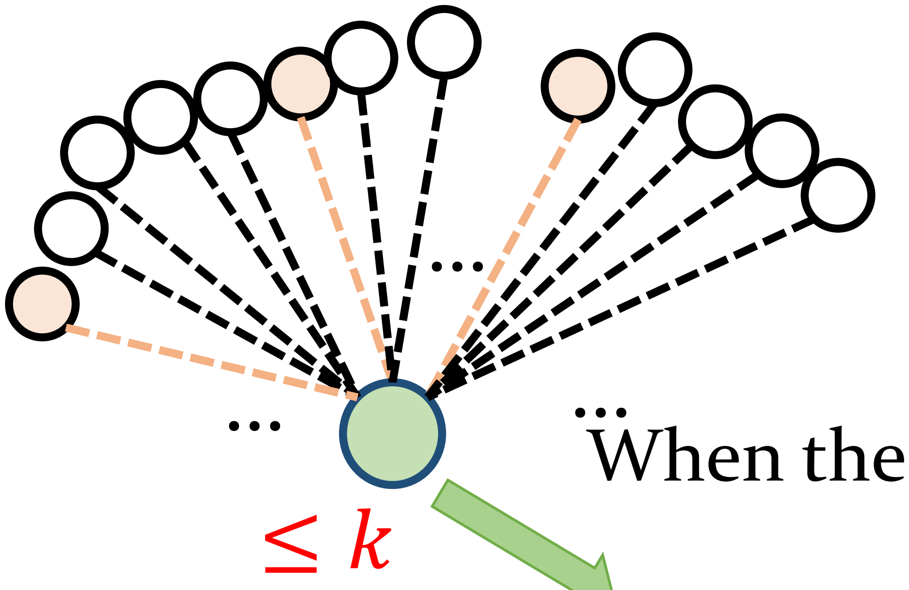
Technique 1: 2-parameter Sampling

When will the error occur?



counter value does not hit the threshold (still under sampling)

Technique 1: 2-parameter Sampling

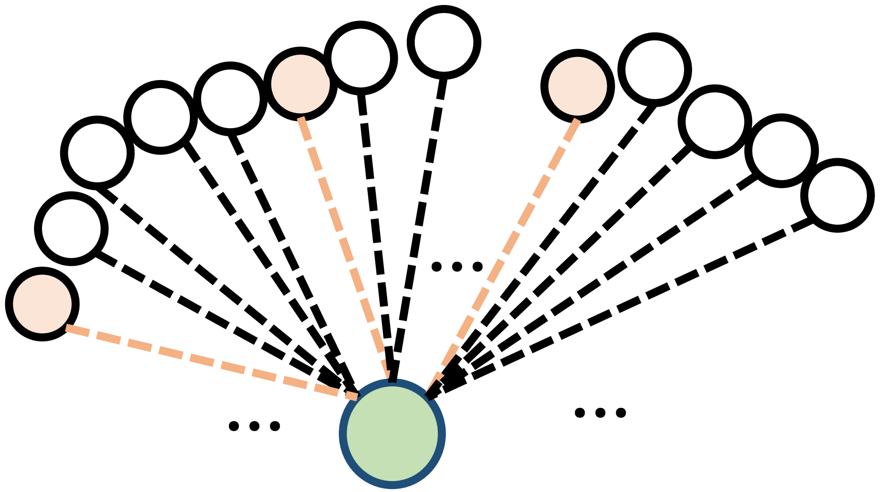


When the counter value is above a ratio w.r.t. k

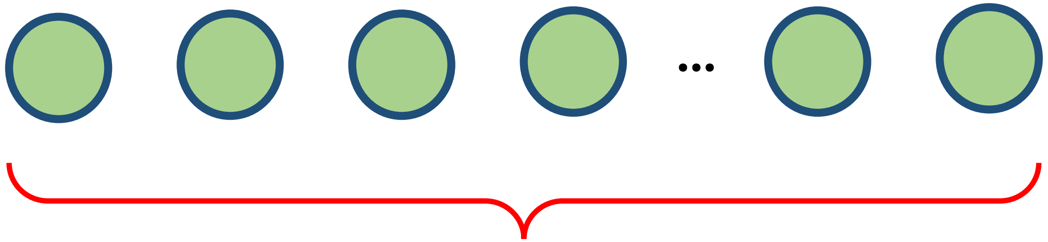
STOP sampling & Check the real induced degree (counting alive neighbors)

Can we make sure the sampling process is correct
<u>*with high probability* regarding $|V|$?</u>

Technique 1: 2-parameter Sampling



$\leq k$



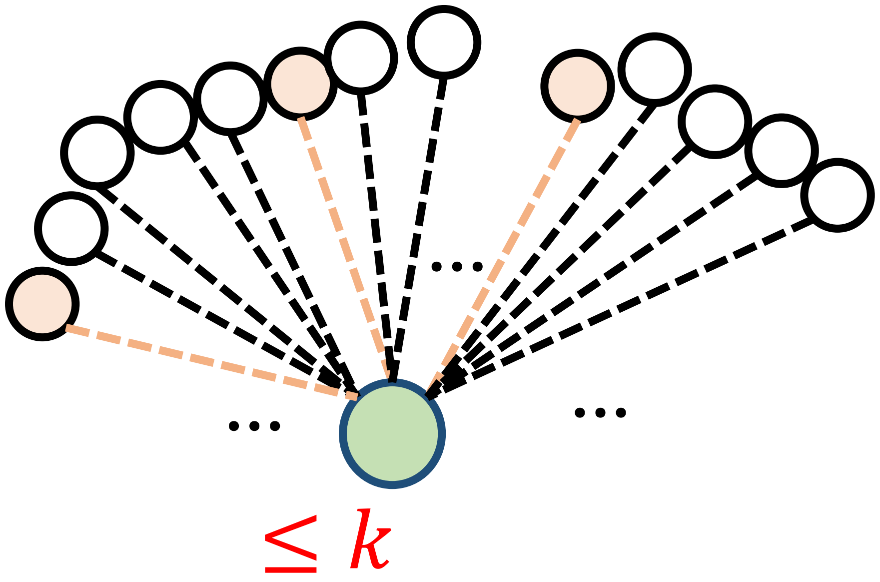
n sampled vertices

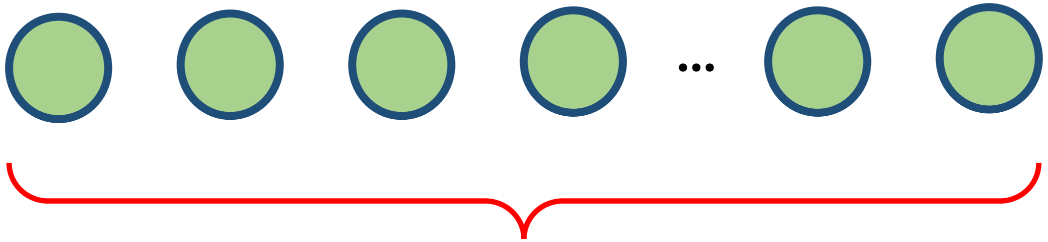
Different *starting* and *exiting* k value

Union-bound for the correctness of sampled vertices

$$\Pr[\text{stop before error}] = \Pr\left[s < \frac{tp}{4}\right] < n^{-\frac{tp}{4\ln n}}$$

Technique 1: 2-parameter Sampling





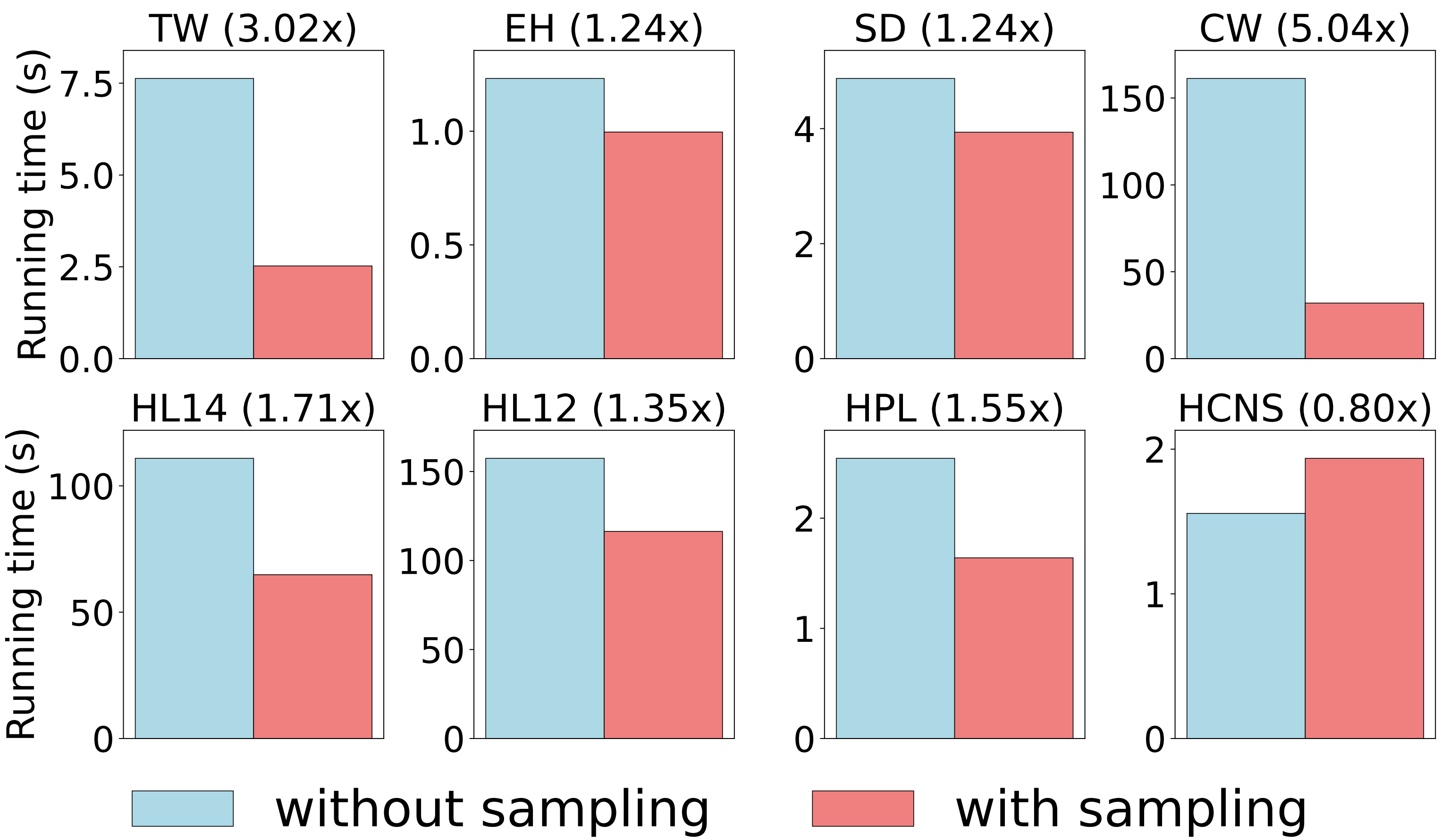
$$\Pr[\text{stop before error}] = \Pr\left[s < \frac{tp}{4}\right] < n^{-\frac{tp}{4 \ln n}}$$

Errors indeed occur

We can detect the errors

Restart the entire algorithm with other parameters

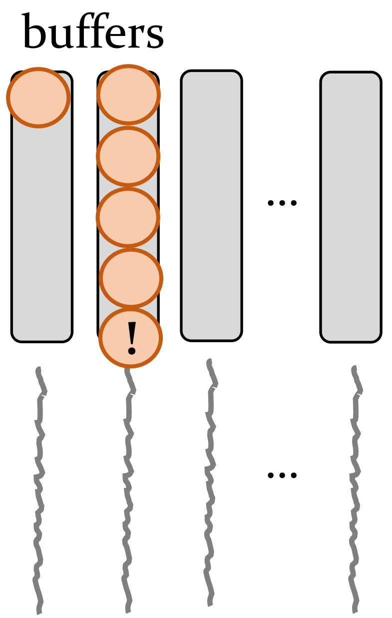
Performance Comparison for Sampling



96-core machine
(192 hyperthreads)

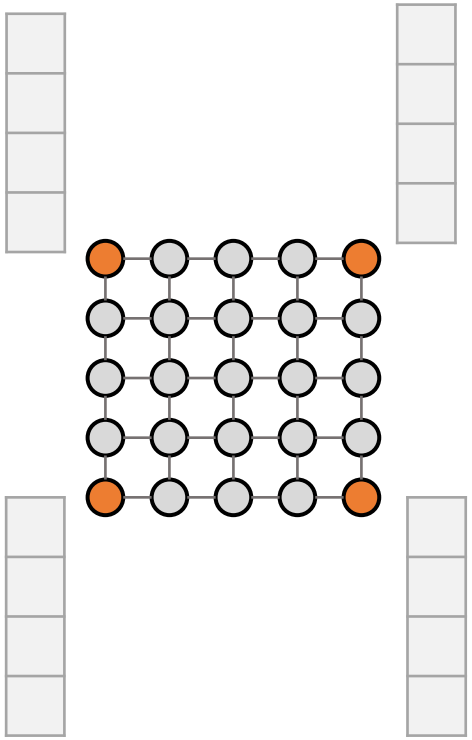
Lower is better

Reduce #Subround: A Better Design?



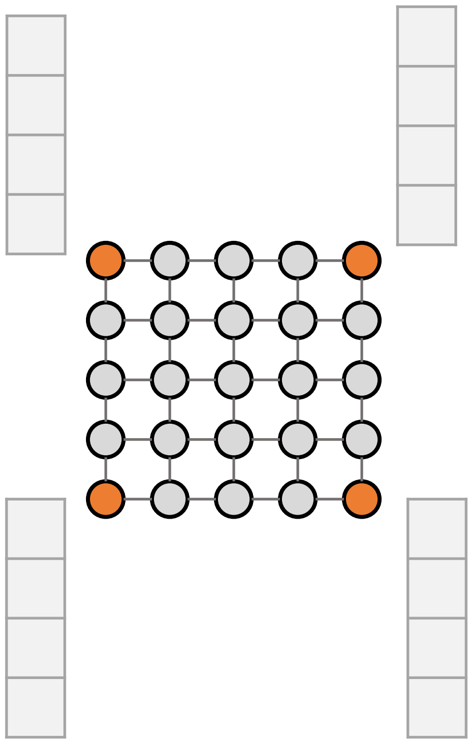
Unbounded buffers -> load imbalance & overflow

Technique 2: Vertical Granularity Control

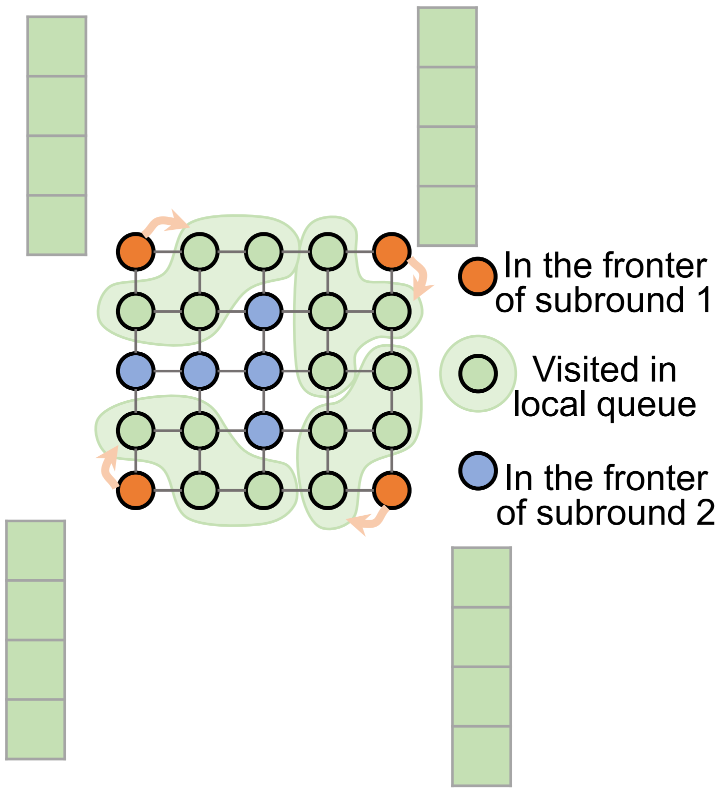


Local queues with fixed sizes

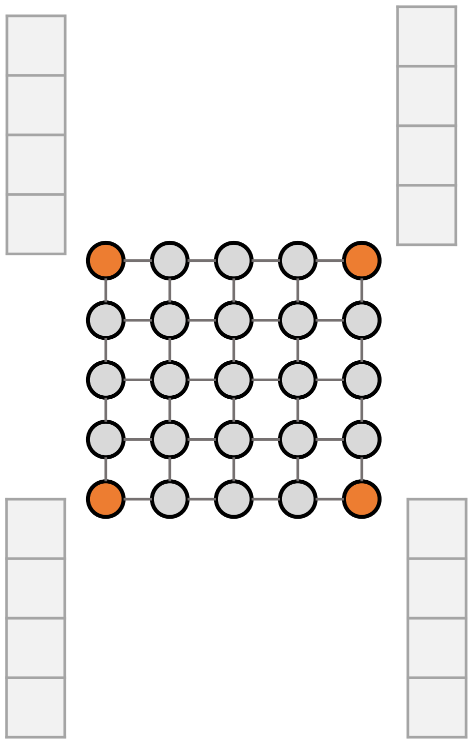
Technique 2: Vertical Granularity Control



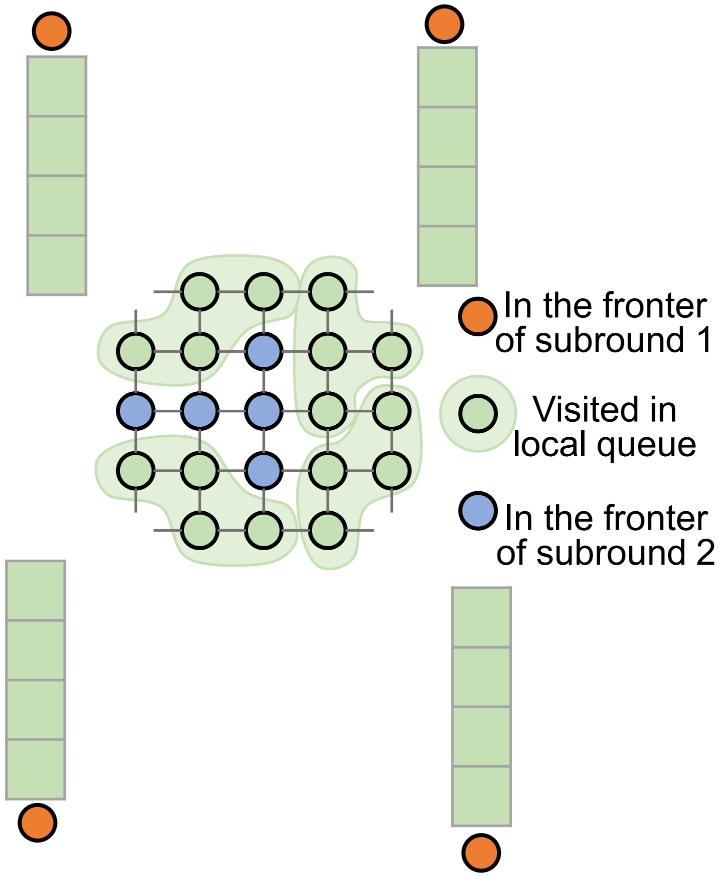
Local queues with fixed sizes



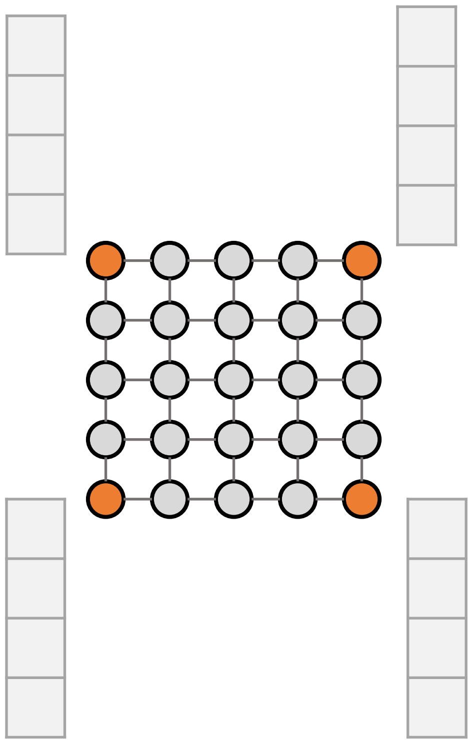
Technique 2: Vertical Granularity Control



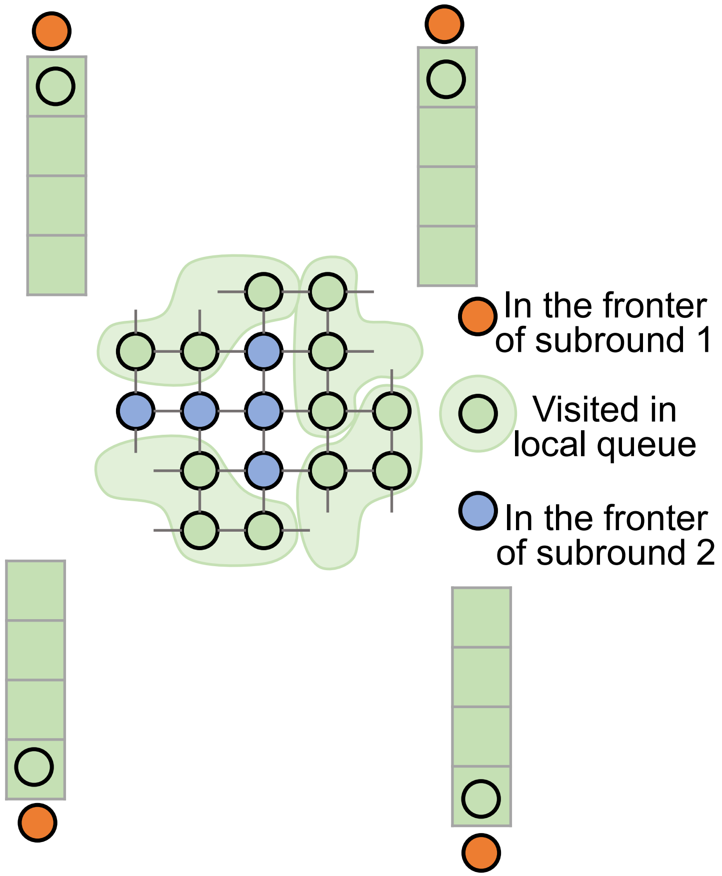
Local queues with fixed sizes



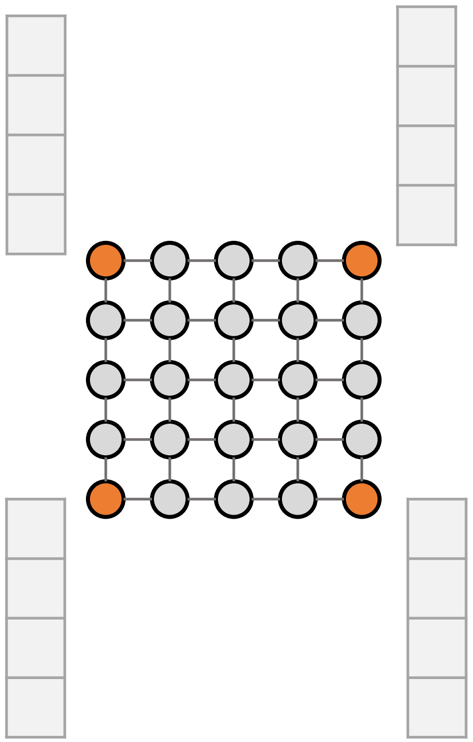
Technique 2: Vertical Granularity Control



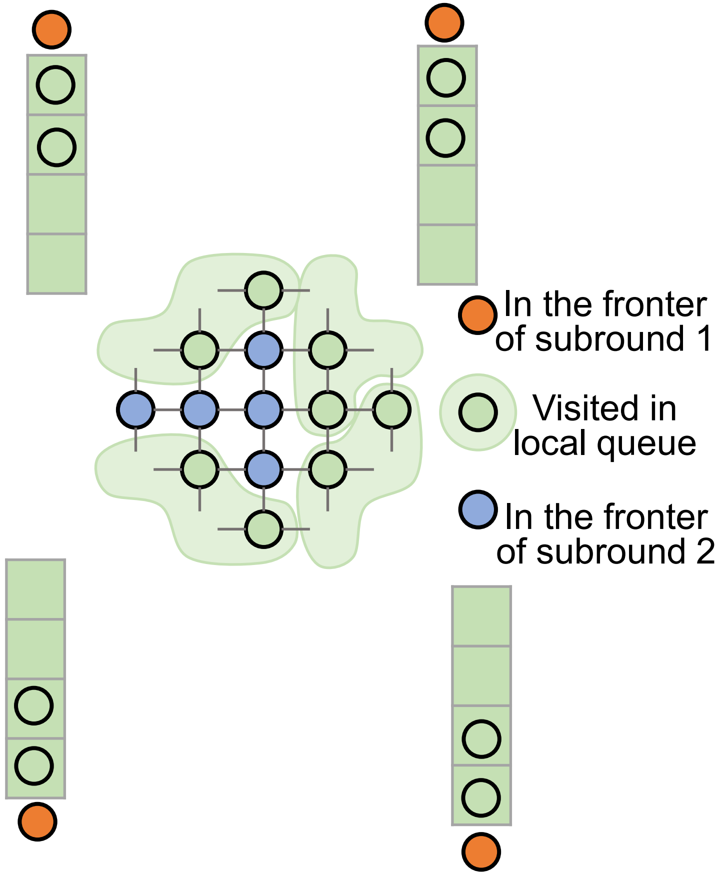
Local queues with fixed sizes



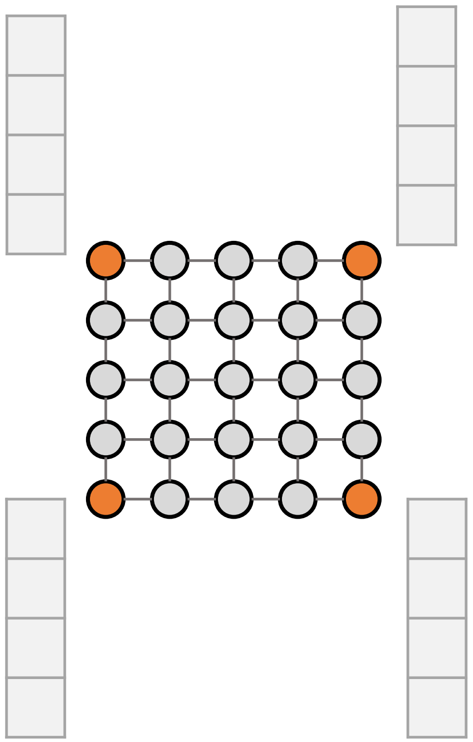
Technique 2: Vertical Granularity Control



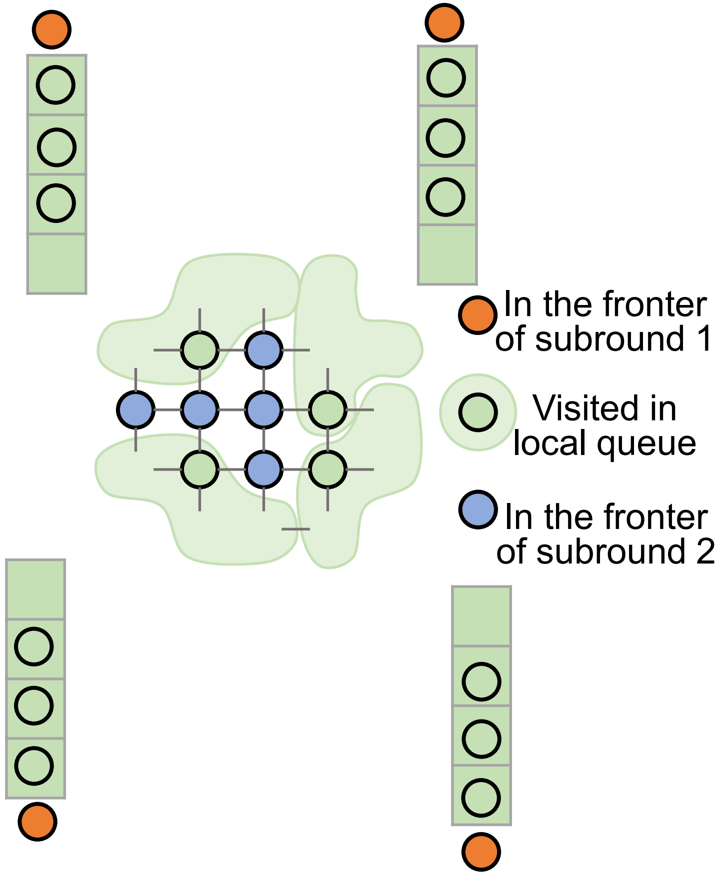
Local queues with fixed sizes



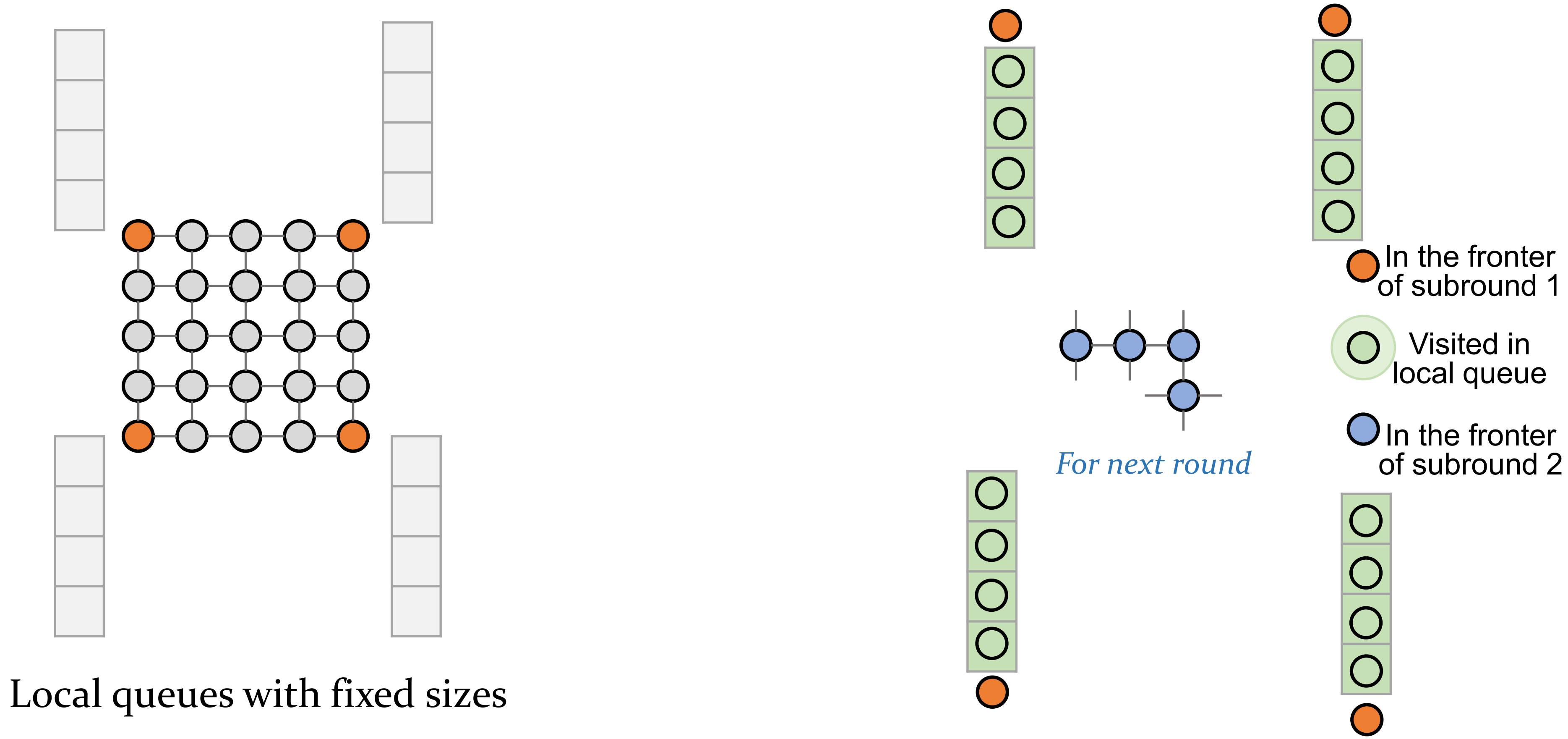
Technique 2: Vertical Granularity Control



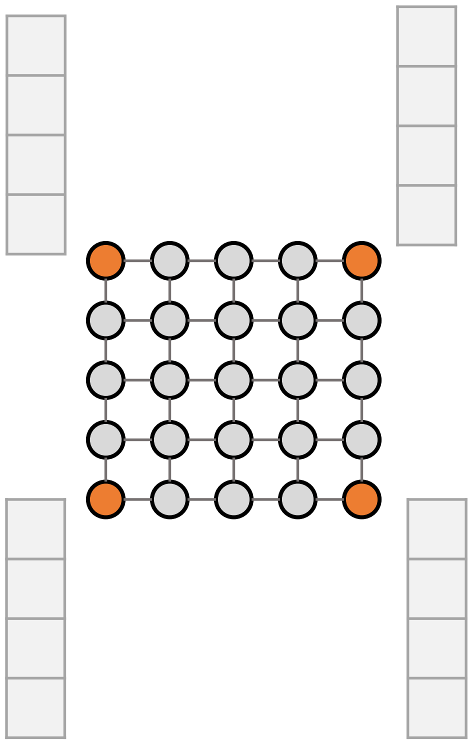
Local queues with fixed sizes



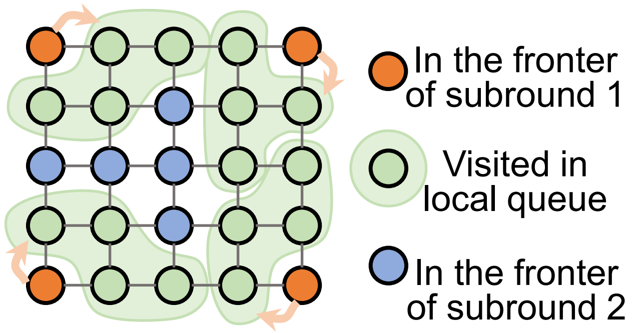
Technique 2: Vertical Granularity Control

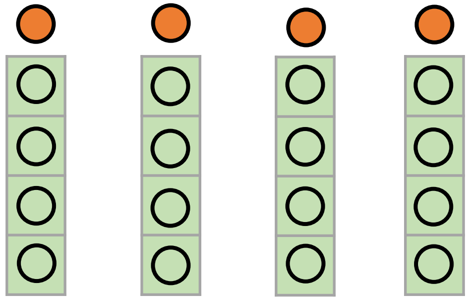


Technique 2: Vertical Granularity Control



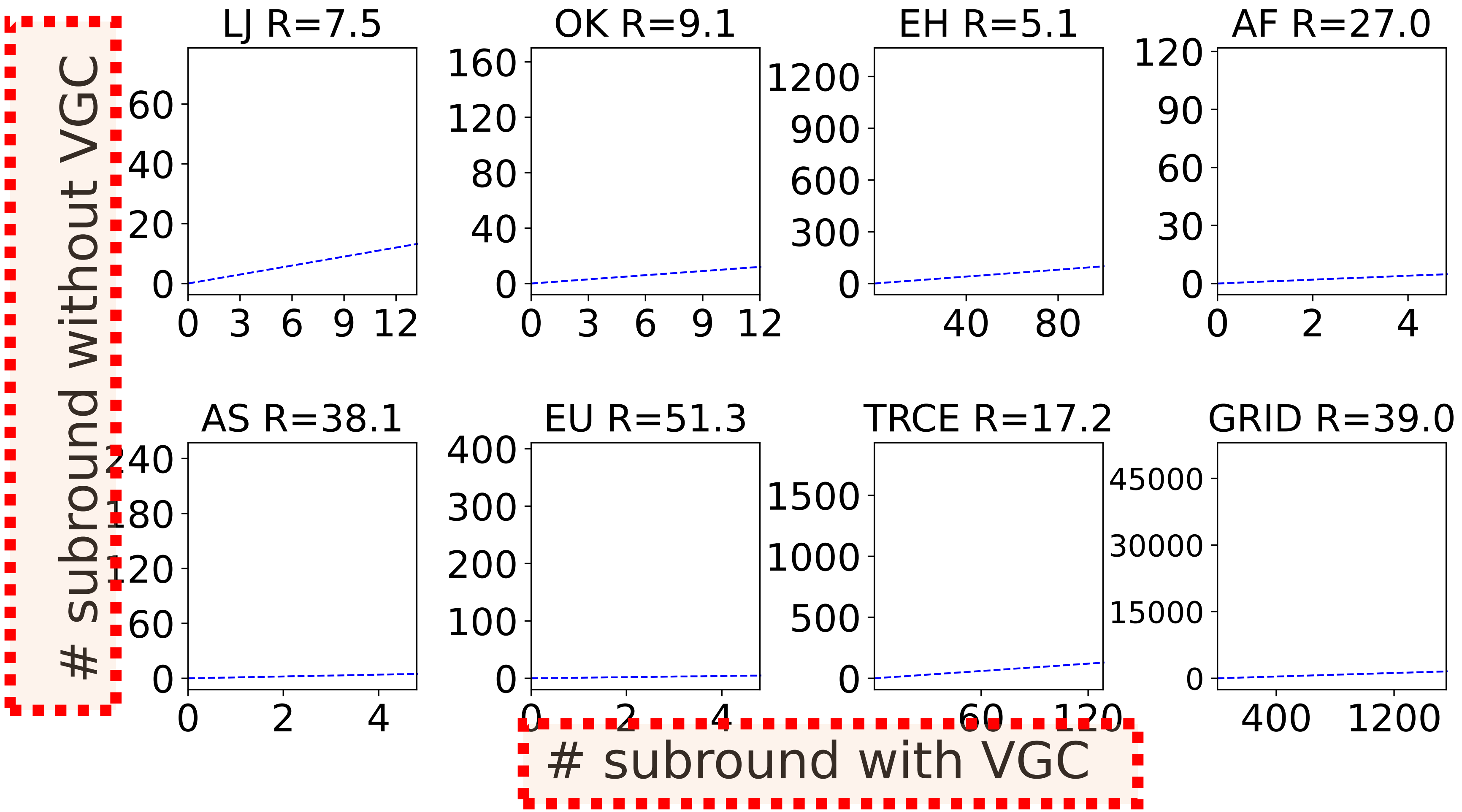
Local queues with fixed sizes



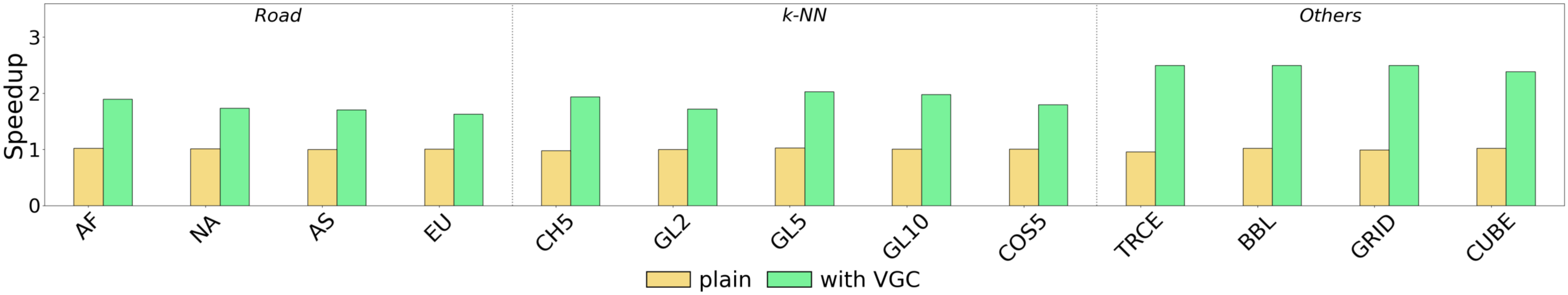


- Reduce #subrounds (for a ratio)
- Load balance
- Avoid overflow

Subround Reduction Ratio with VGC



VGC Provides Significant Speed-Up



Higher is better

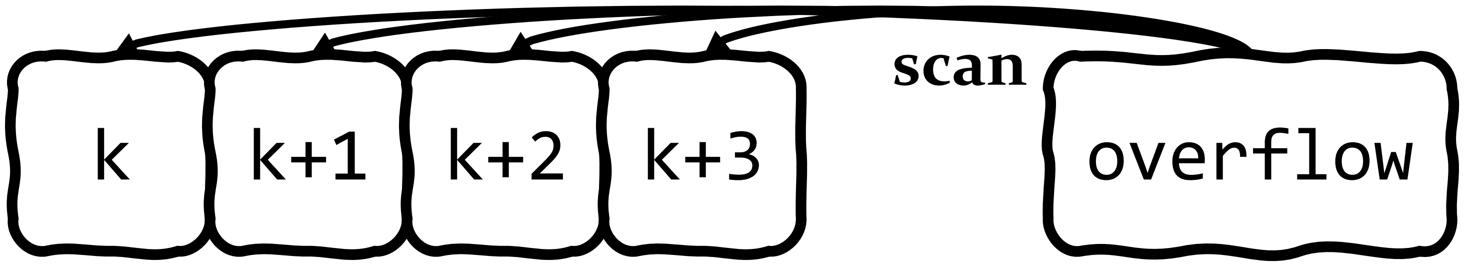
Summary: Two Optimizations

Optimization 1:
2-parameter Sampling → **Reduce contention**

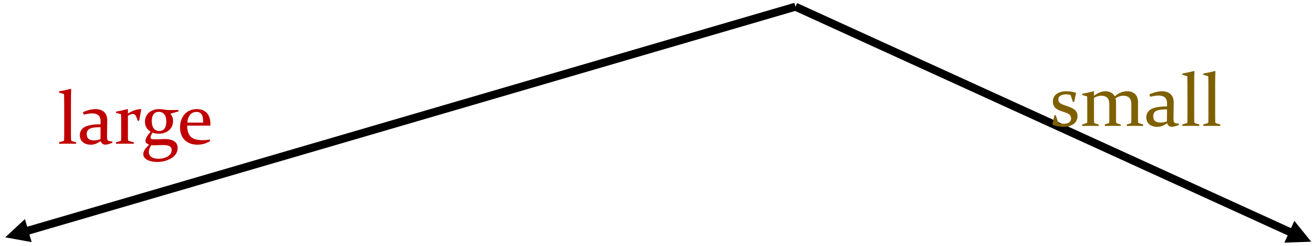
Optimization 2:
Vertical Granularity Control → **Reduce #subrounds & Better load balance**

Further improvements?
Maintain vertices in a better structure?

Why do we need a better bucketing structure?



Fixed number of buckets



- **Space** issue;
- ***Movements*** overhead

- Large **scan** overhead

Our Hierarchical Bucketing Structure (HBS)

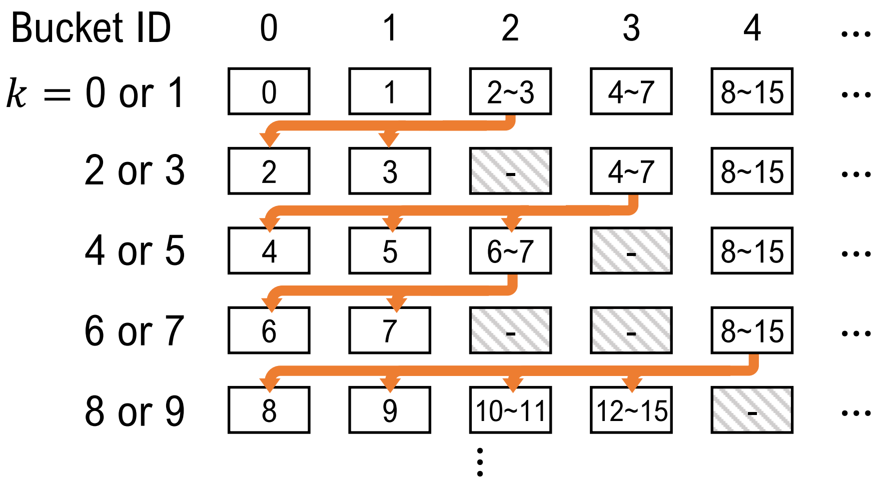
Each bucket holds a range of degrees instead of a cetain one

Bucket ID	0	1	2	3	4	⋯
k = 0 or 1	0	1	2~3	4~7	8~15	⋯

Bucketing structure in a **prefix-doubling** manner

Range size `1, 1, 2, 4, 8,` …

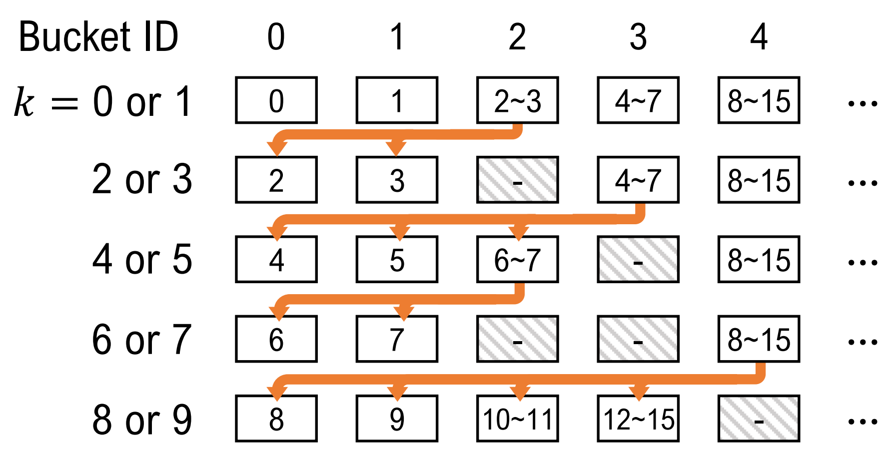
Our Hierarchical Bucketing Structure (HBS)



The new HBS can efficiently handle and balance all cases

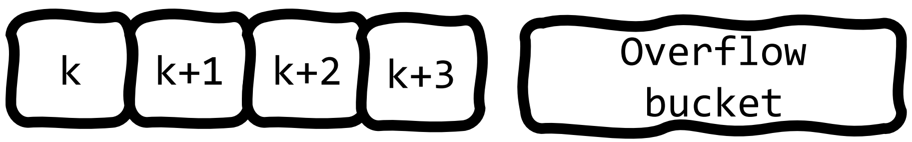
Our Hierarchical Bucketing Structure (HBS)

HBS



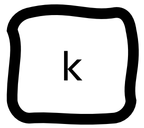
$O(\log d(v))$

fix-number buckets



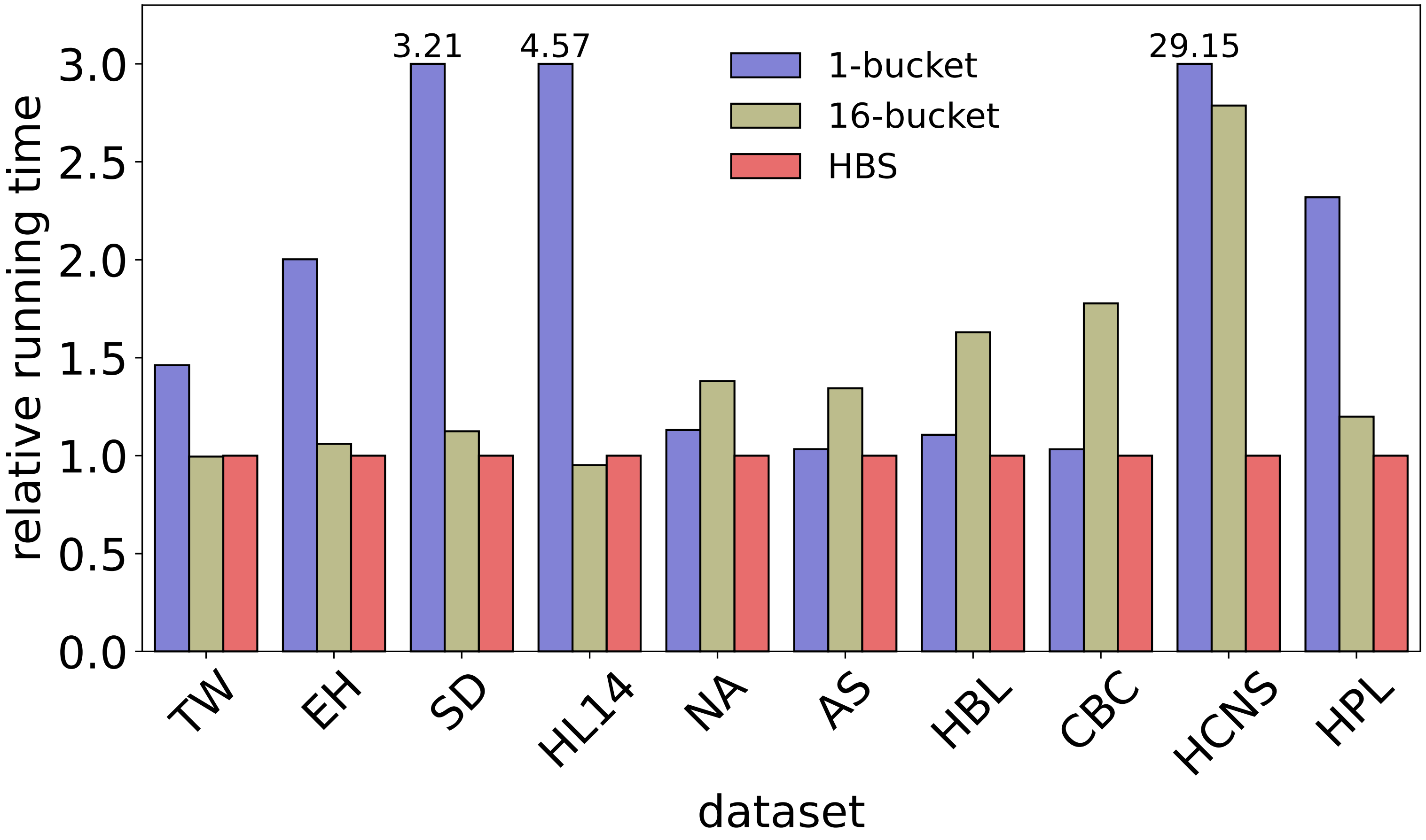
$O(\sqrt{m/n})$

single frontier (1 bucket)



$O(d(v))$

HBS is overall better than the other two structures

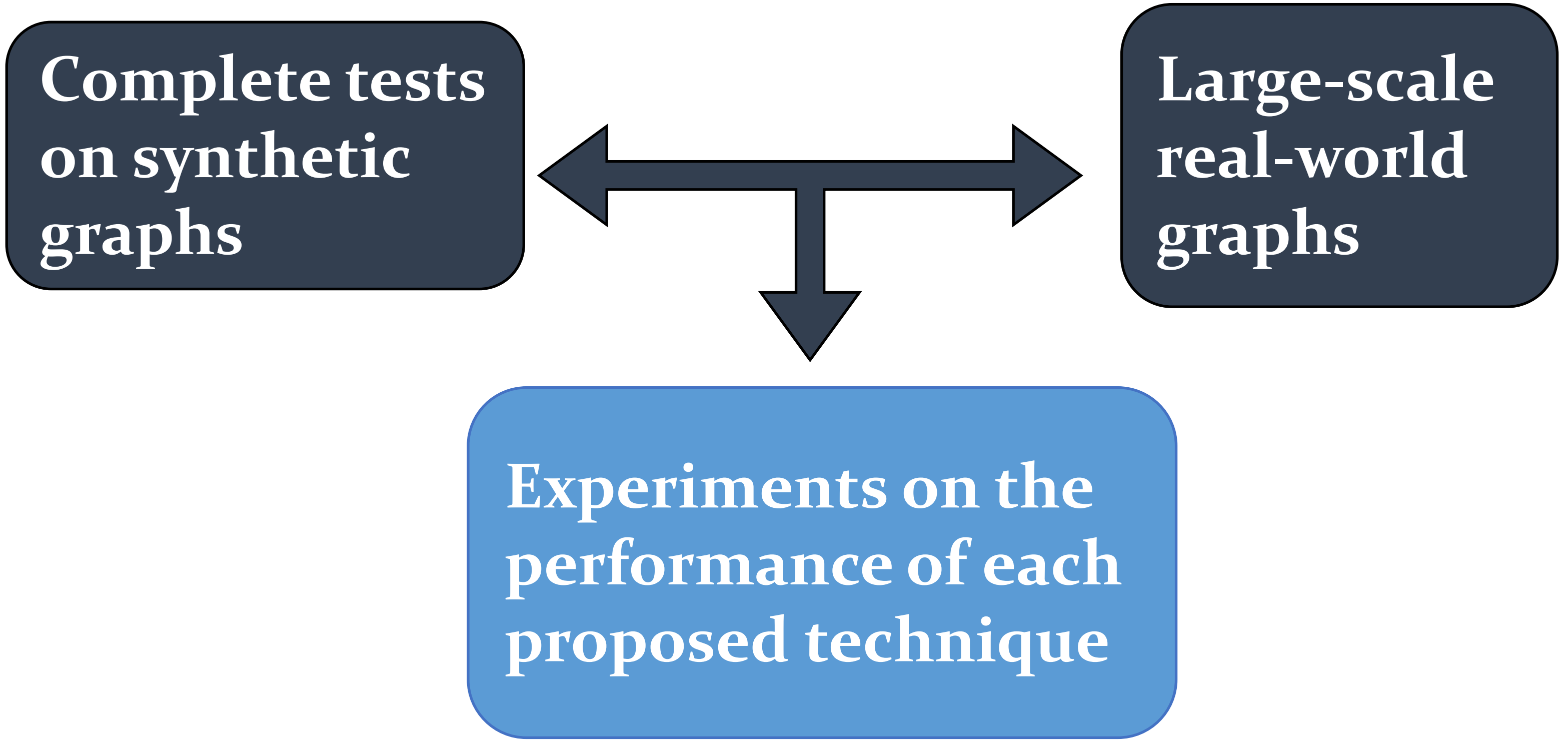


Lower is better

Experiments Setups

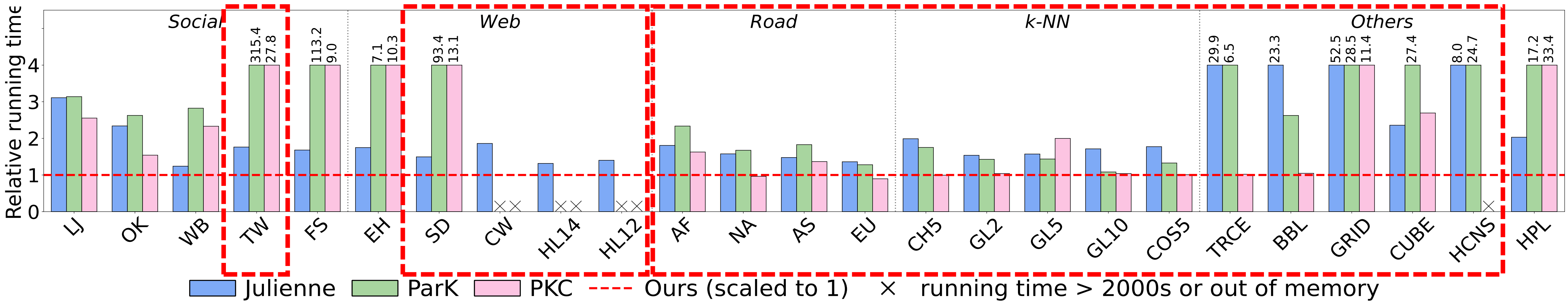
Experiments setup

- ParlayLib [1] for **fork-join parallelism** and **primitives**
- **96-core** (**192 hyperthreads**) machine with four Intel Xeon Gold 6252 CPUs



Guy E. Blelloch, Daniel Anderson, and Laxman Dhulipala. 2020. ParlayLib - A Toolkit for Parallel Algorithms on Shared-Memory Multicore Machines. In Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '20)

Our framework with all the techniques: Better Overall Performance

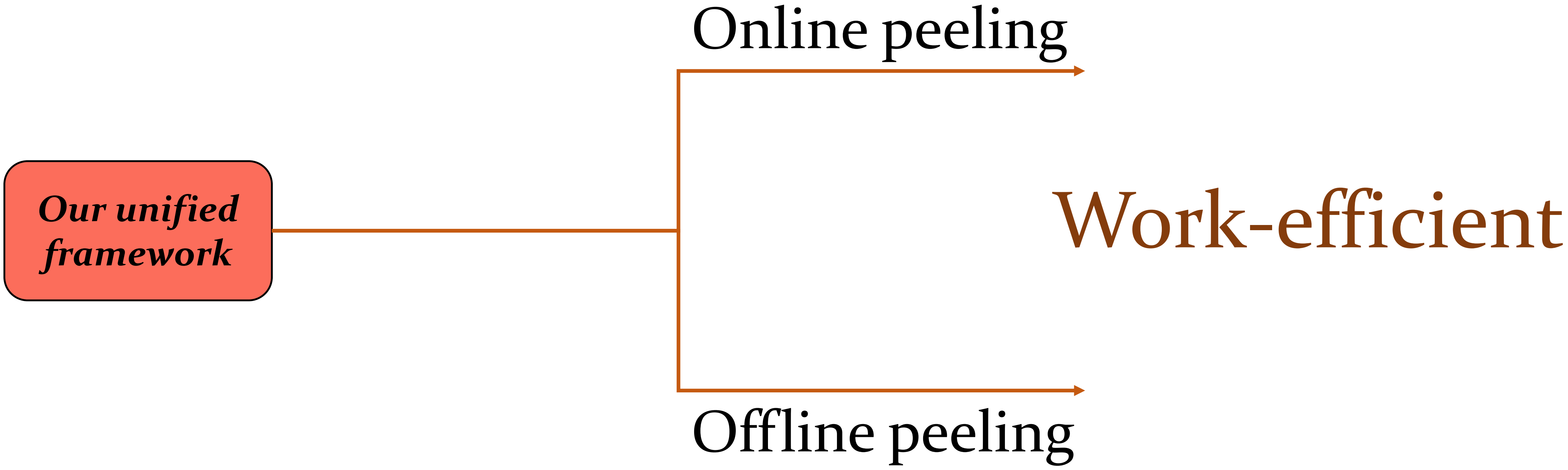


Lower is better

96-core machine
(192 hyperthreads)

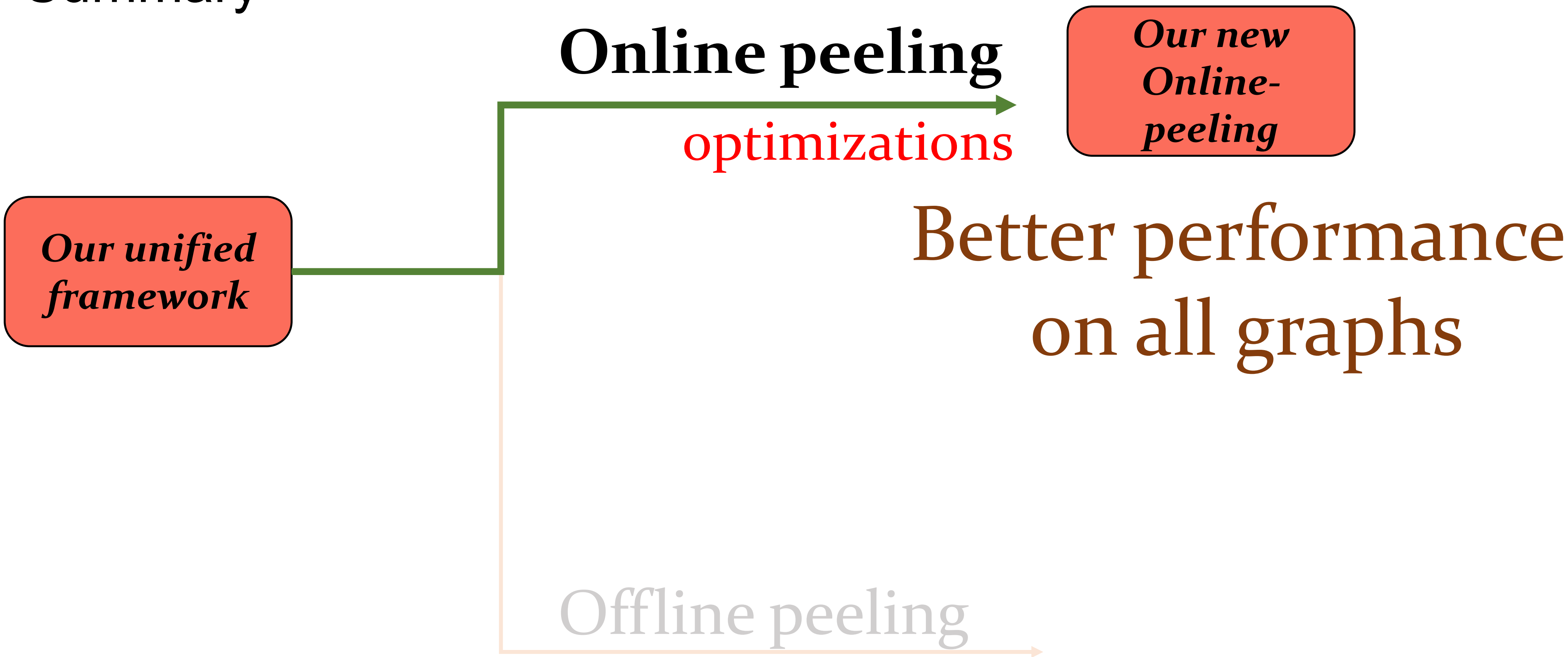
Conclusions: Work-efficient Framework

- Summary



Conclusions: Improvements on Our Framework

- Summary



Conclusions: Two Optimizations

Our new Online-peeling

For dense graphs

For all types of graphs

Technique 1: sampling

Technique 2: Vertical granularity control

Improve the contention issue

Reduce #subrounds with better load balance

Conclusions: HBS – A Better Structure

Our new Online-peeling

0 | 1 | 2~3 | 4~7 | 8~15

A better bucketing structure

For dense graphs

Technique 1: sampling

Improve the contention issue

For all types of graphs

Technique 2: Vertical granularity control

Reduce #subrounds with better load balance

Contact

- Code on GitHub:
  - https://github.com/ucrparlay/PASGAL

- Contact
  - Youzhe Liu (*yliu908@ucr.edu*)