Parallel Batch-Dynamic Algorithms for *k*-Core Decomposition and Related Problems

Quanquan C. Liu
Northwestern University



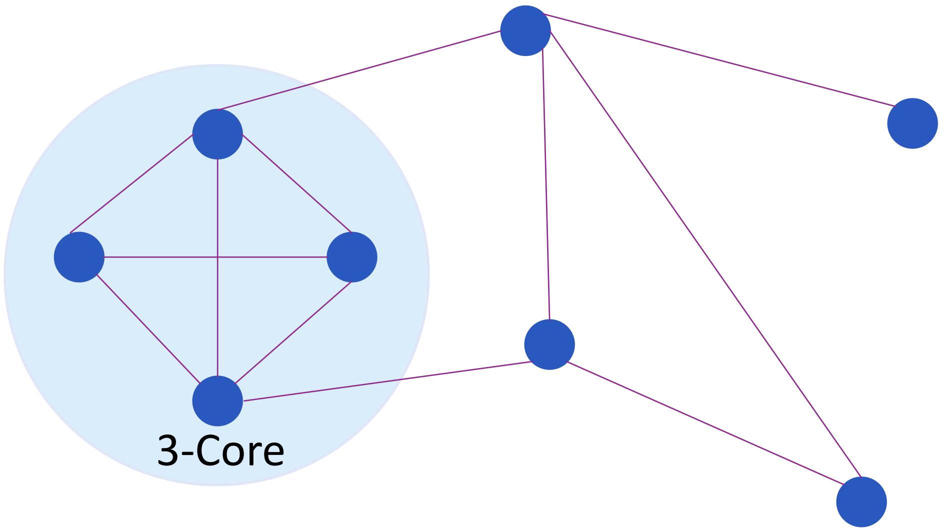

Jessica Shi
MIT

Shangdi Yu
MIT

Laxman Dhulipala
University of Maryland

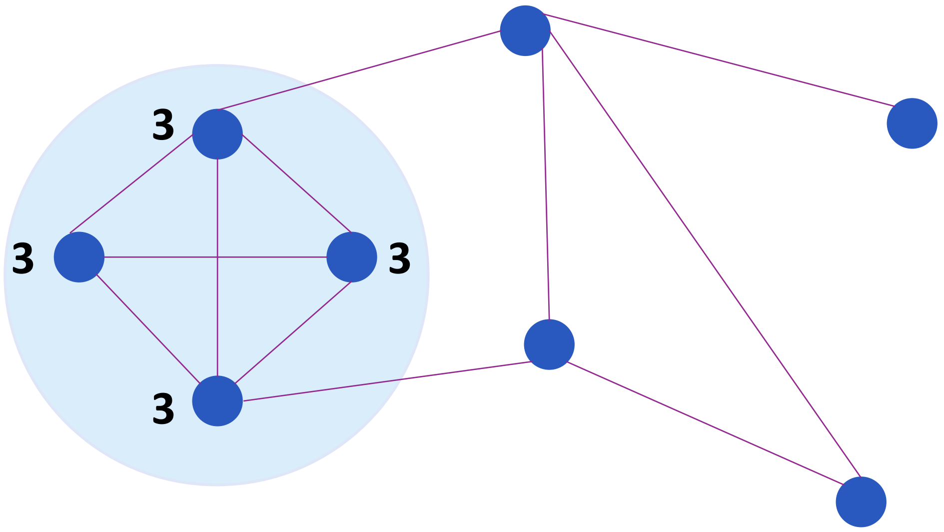
Julian Shun
MIT

k-Core



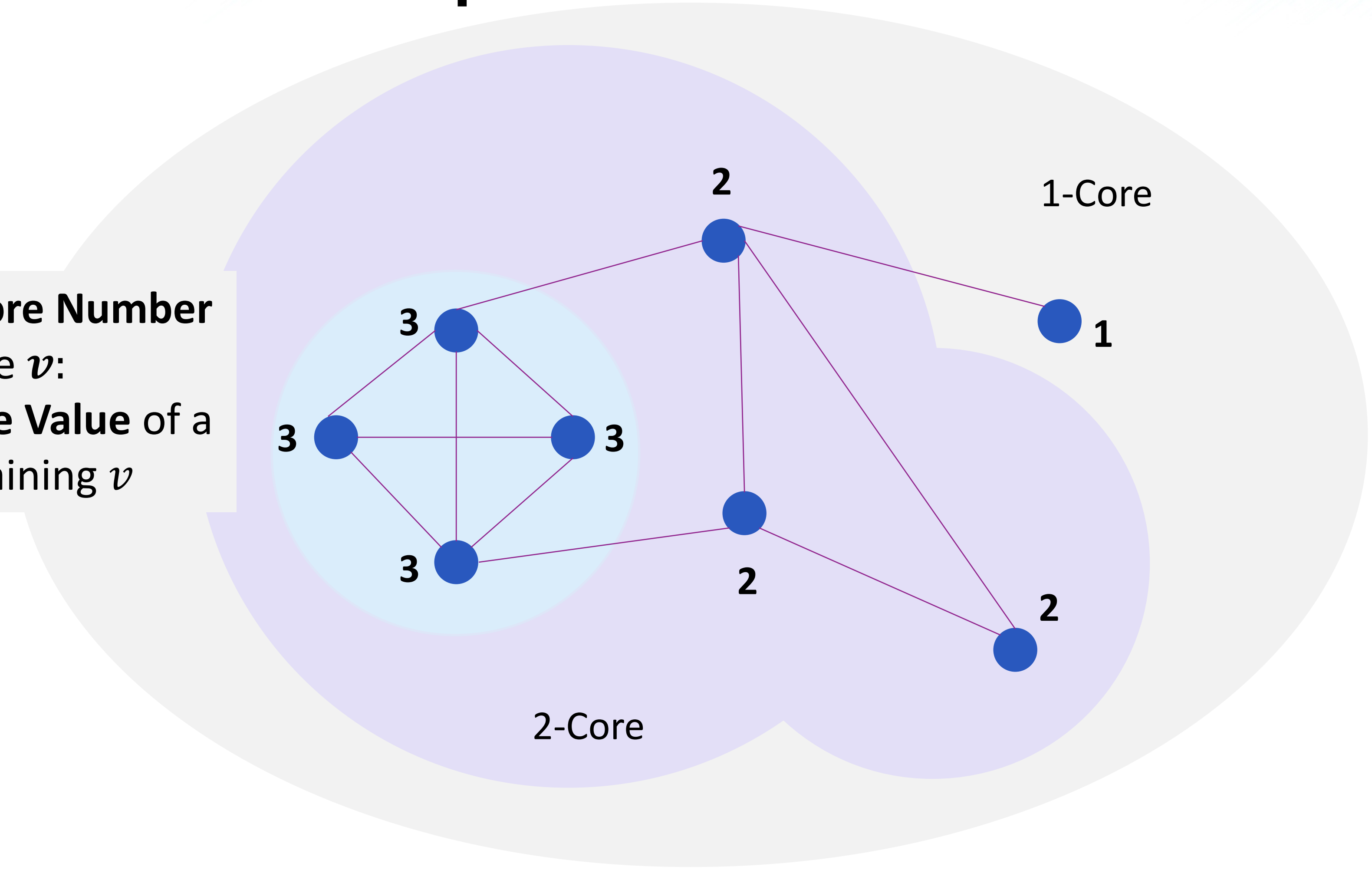
k-Core Decomposition

Coreness or Core Number of Node v:
Maximum Core Value of a Core Containing v



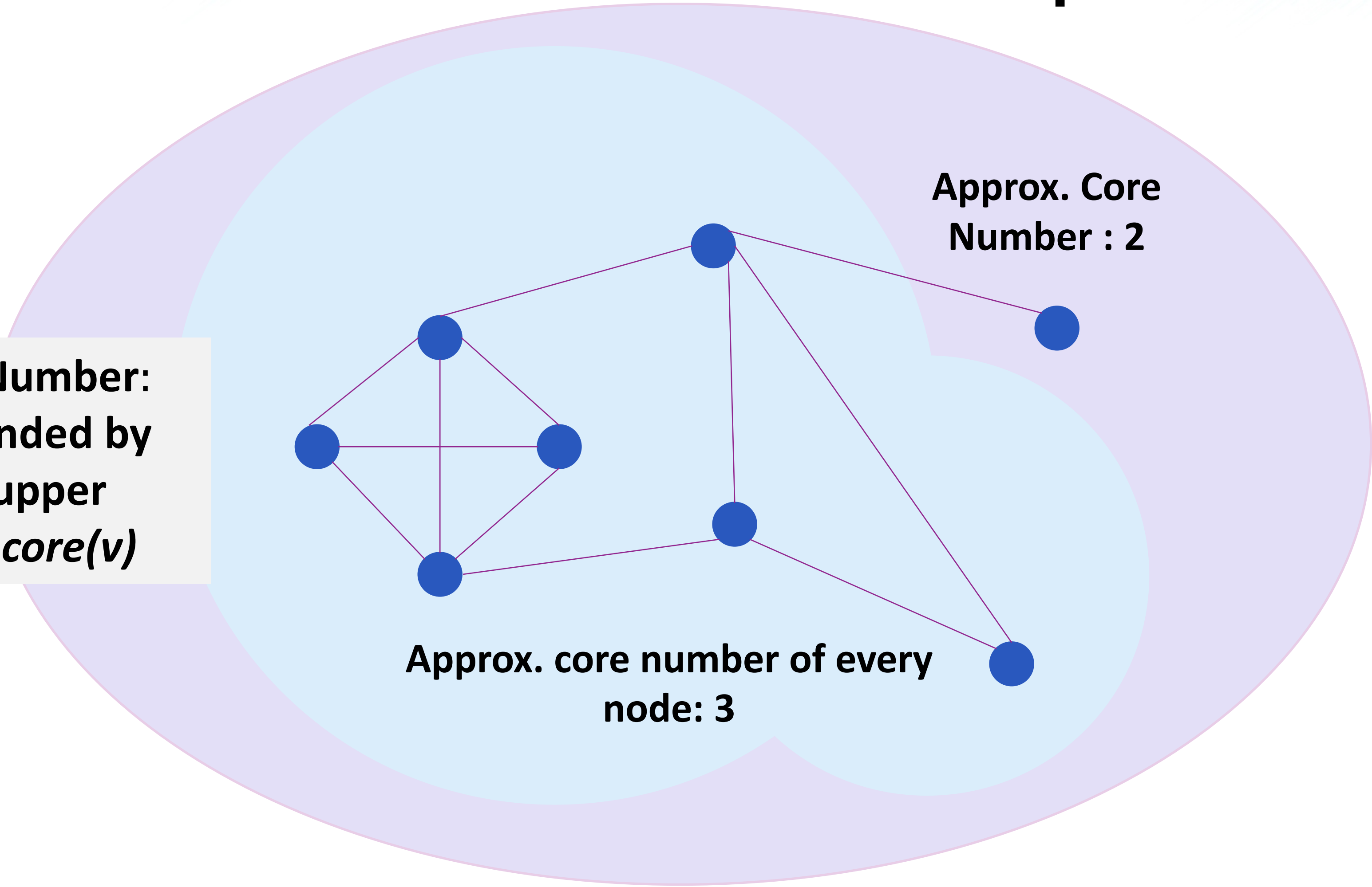
k-Core Decomposition

Coreness or Core Number of Node v: **Maximum Core Value** of a Core Containing v



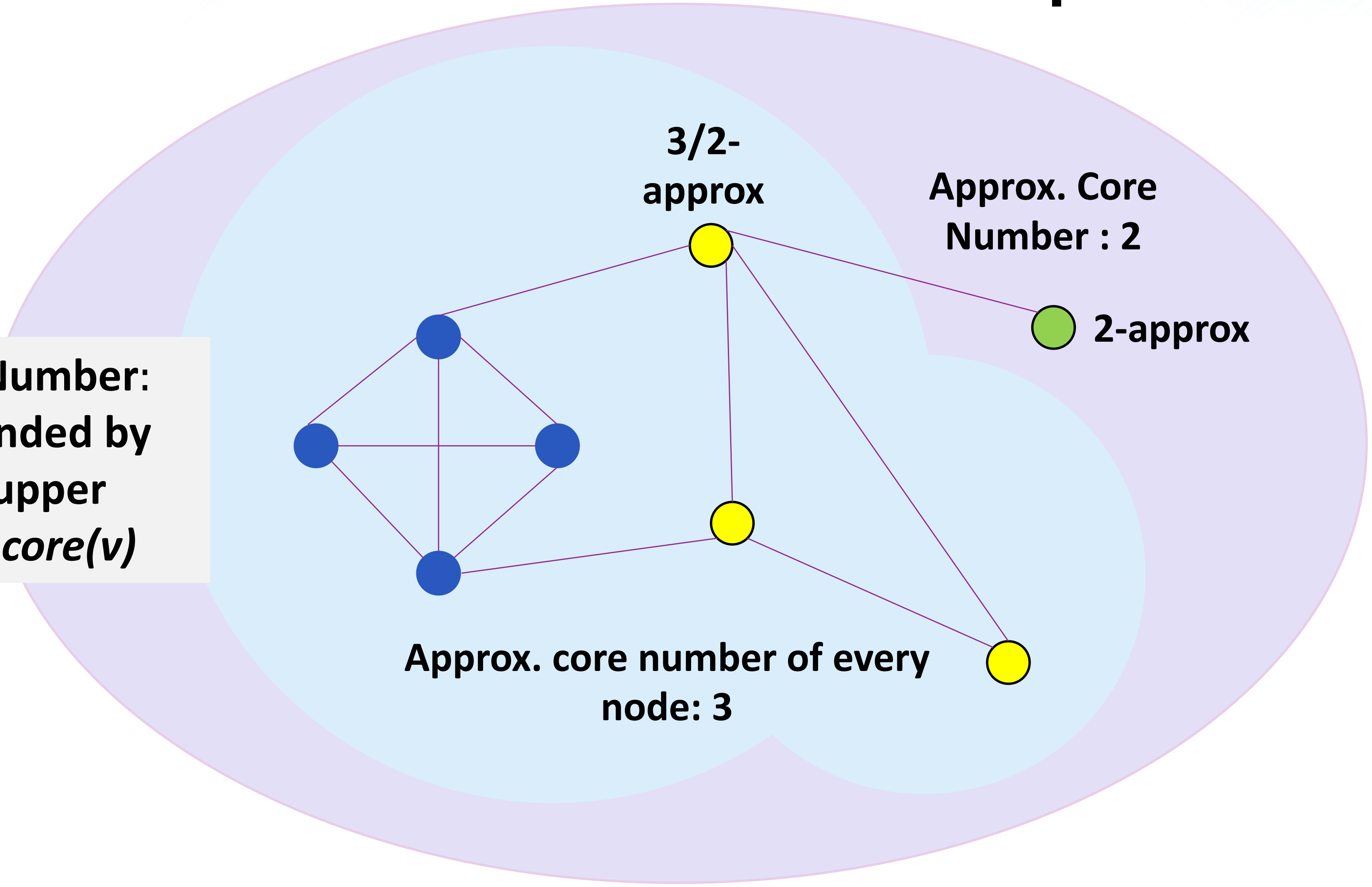
Approximate *k*-Core Decomposition

c-Approx. Core Number: Value **lower bounded by *core(v)/c*** and **upper bounded by *c * core(v)***



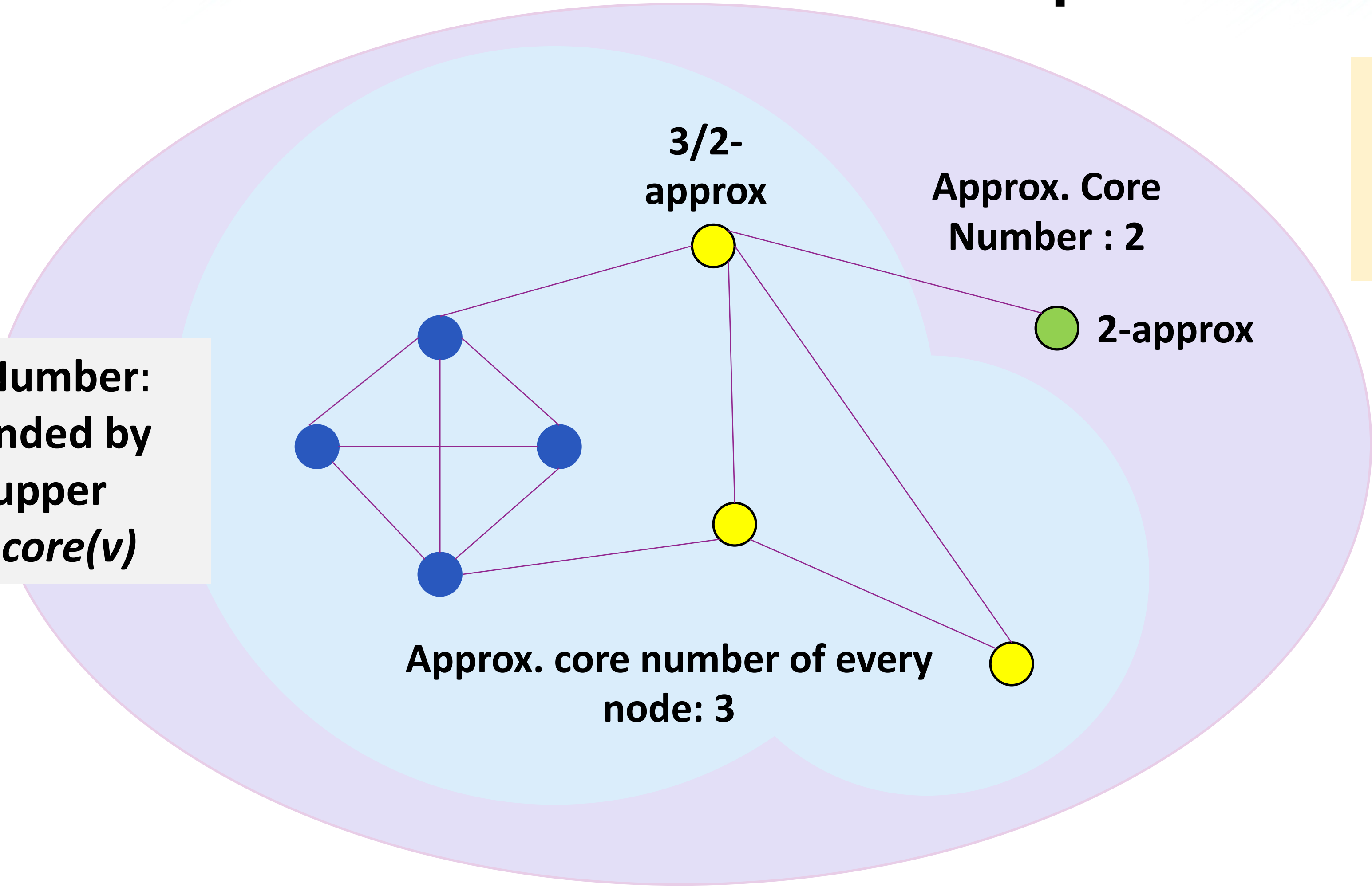
Approximate *k*-Core Decomposition

_c_-Approx. Core Number: Value **lower bounded by _core(v)/c_** and **upper bounded by _c * core(v)_**



Approximate *k*-Core Decomposition

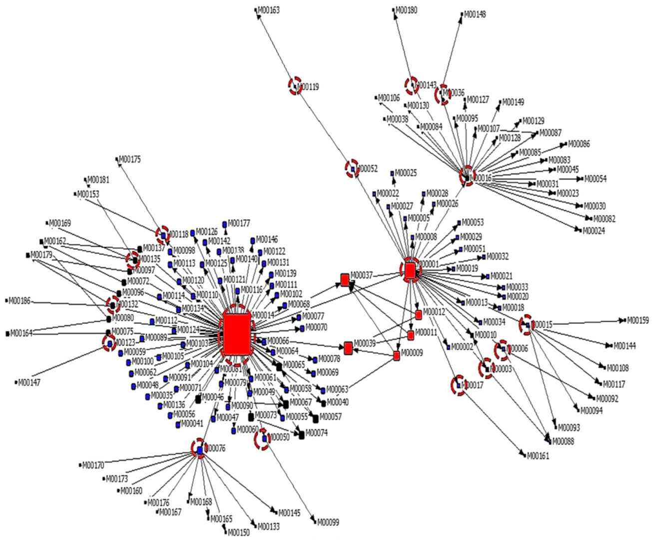
c-Approx. Core Number: Value **lower bounded by** ***core(v)/c*** and **upper bounded by *c * core(v)***

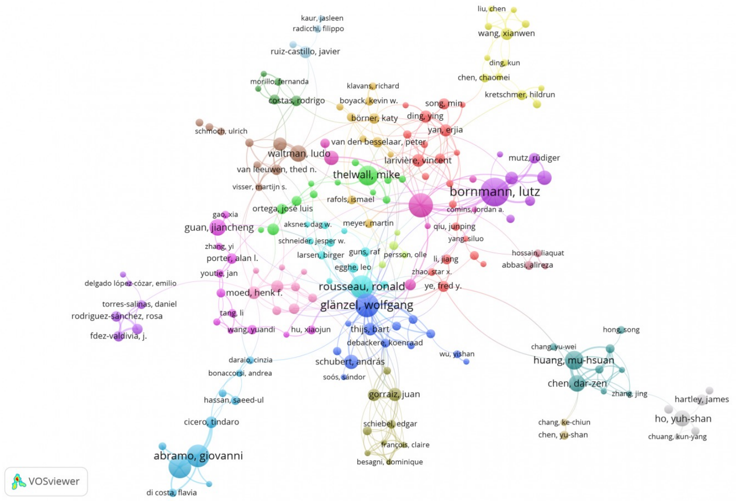


$(2+\varepsilon)$-approximations in this paper

Applications of *k*-Core Decomposition

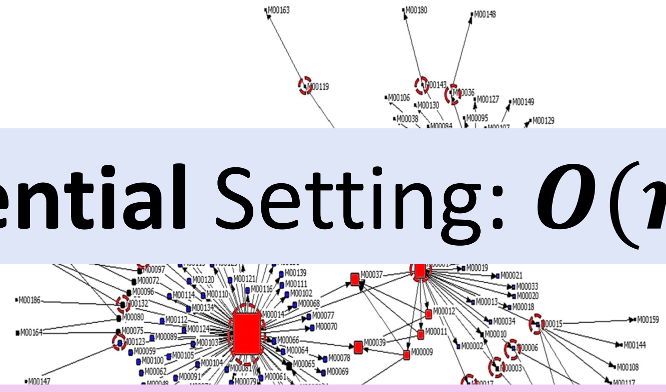
- Graph clustering
- Community detection
- Graph visualizations
- Protein network analysis
- Modeling of disease spread
- Approximating network centrality measures
- Much interest in the machine learning, database, graph analytics, and other communities





Applications of *k*-Core Decomposition

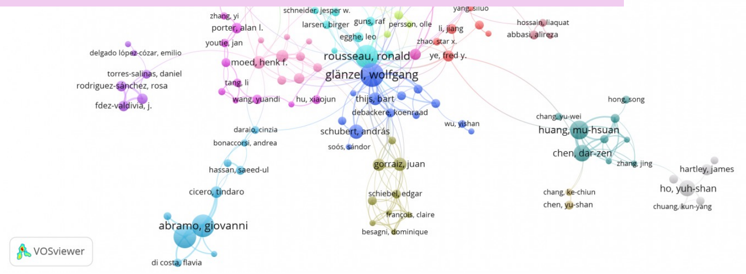
- Graph clustering
- Commu
- Graph visualizations
- Protein netwo
- Modeling of di
- Approximating measures
- Much interest in the machine learning, database, graph analytics, and other communities



Static, Sequential Setting: $O(n)$ time

Billions or Even Trillions of Edges

Too Much Time to **Process Statically and Sequentially**



Large Graphs

facebook ~ 92.5 million edges

friendster ~ 1.8 billion edges

~ 2 billion edges

Common Crawl ~ 128 billion edges

Google ~ 6 trillion edges

Large Graphs

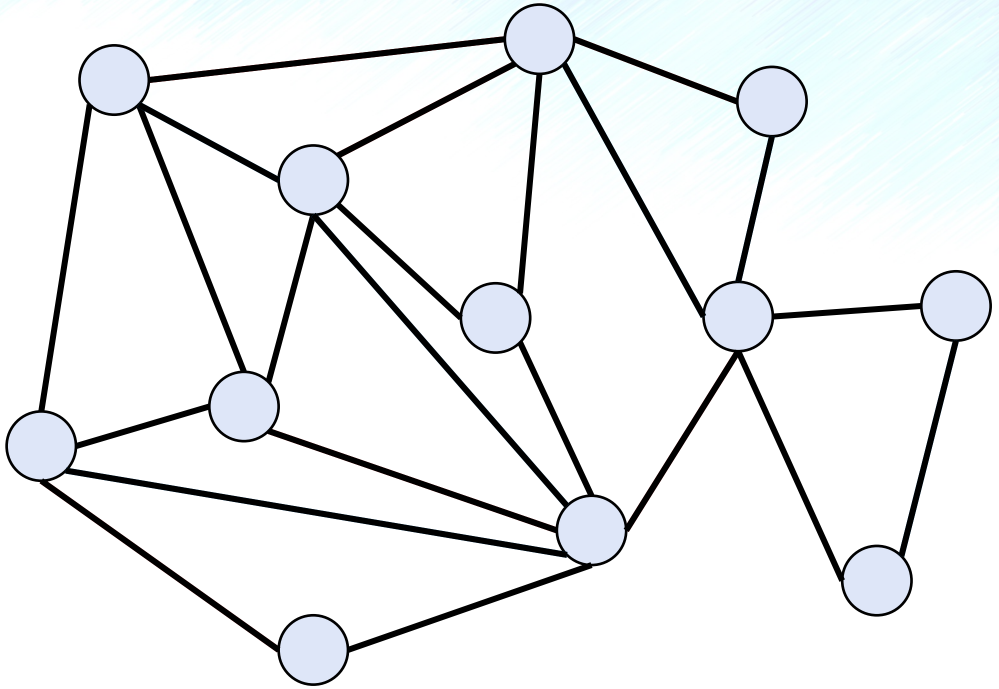
facebook ~ 92.5 million edges

friendster ~ 1.8 billion edges

~ 2 billion edges

Common Crawl ~ 128 billion edges

Google ~ 6 trillion edges



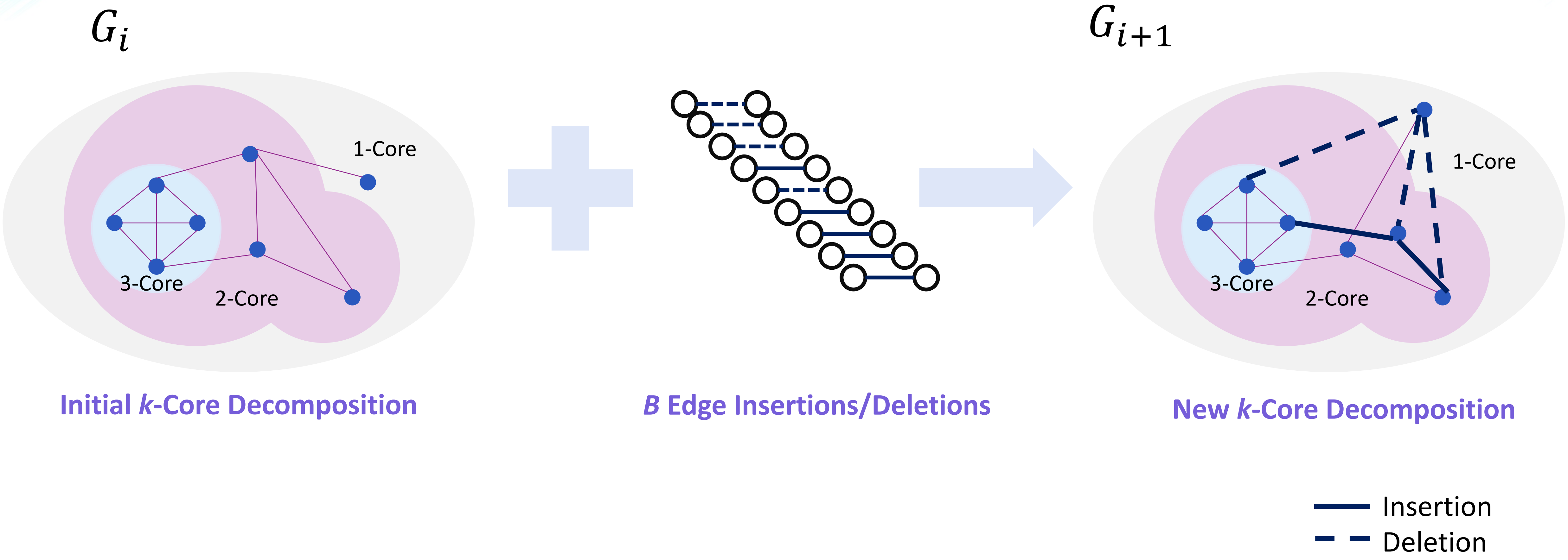
Graphs are rapidly changing:

- 3M emails/sec
- 486K WhatApp messages/sec
- 500M tweets/day
- 547K new websites/day

Work-Depth Model

- **Work:**
  - Total number of operations executed by algorithm
  - ***Work-efficient***: work asymptotically the same as *best-known sequential algorithm*
- **Depth:**
  - Longest chain of sequential dependencies in algorithm
- **Other Characteristics:**
  - Arbitrary forking
  - Concurrent read, concurrent write to the same shared memory

Batch-Dynamic Model Definition



Batch-Dynamic Graph Algorithms

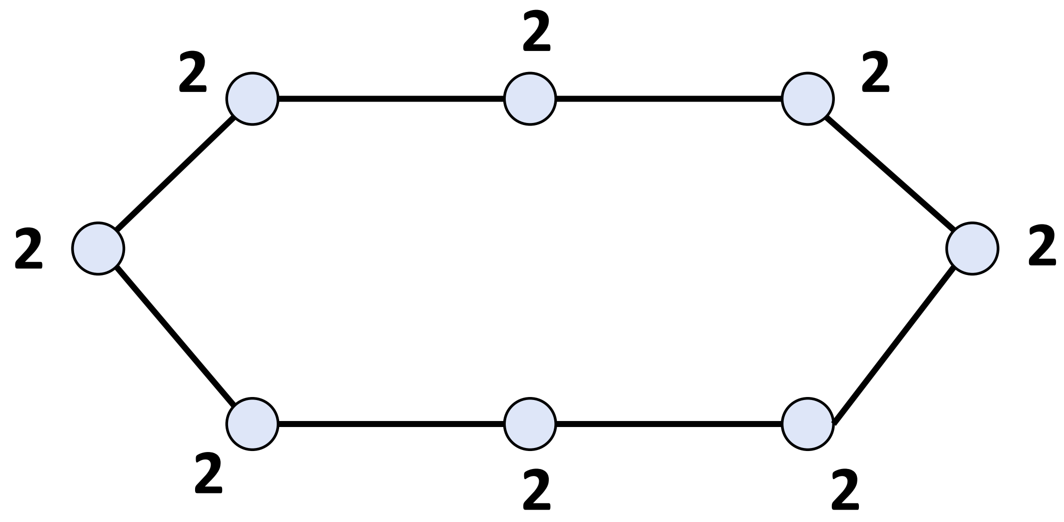
- Triangle counting [Ediger et al. ‘10, Makkar et al. ’17, Dhulipala et al. ‘20]
- Euler Tour Trees [Tseng et al. ‘19]
- Connected Components [Ferragina and Lucio ‘94, McColl et al. ‘13; Acar et al. ’19, Nowicki and Onak ‘21]
- Rake-Compress Trees [Acar et al. ‘20]
- Incremental Minimum Spanning Trees [Anderson et al. ‘20]
- Minimum Spanning Forest/Graph Clustering [Nowiki and Onak ‘21, Tseng et al. ‘22]
- Graph Connectivity [Dhulipala et al. ‘20]
- Maximal Matching [Nowicki and Onak ‘21]

Why Approximate *k*-Core Decomposition

- **Dynamic exact *k*-core decomposition:**
  - **$\Omega(n)$ work, $\Omega(n)$ depth, parallel** [Aridhi et al '16, Gabert et al. '21, Hua et al. '20, Jin et al. '18, Wang '17]
  - One update can cause **$\Omega(n)$ coreness changes**

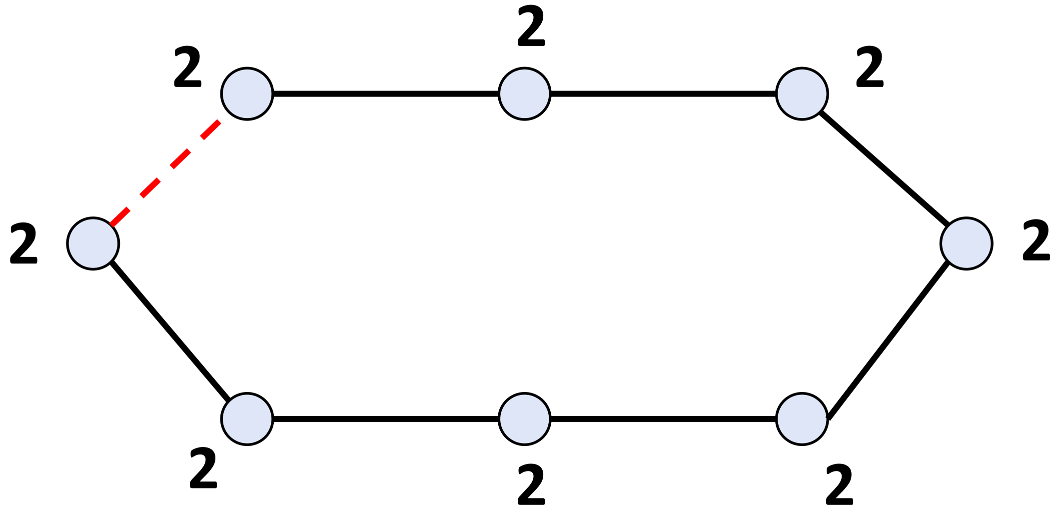
Why Approximate k-Core Decomposition

- **Dynamic exact k-core decomposition:**
  - **$\Omega(n)$ work, $\Omega(n)$ depth, parallel** [Aridhi et al '16, Gabert et al. '21, Hua et al. '20, Jin et al. '18, Wang '17]
  - One update can cause **$\Omega(n)$ coreness changes**



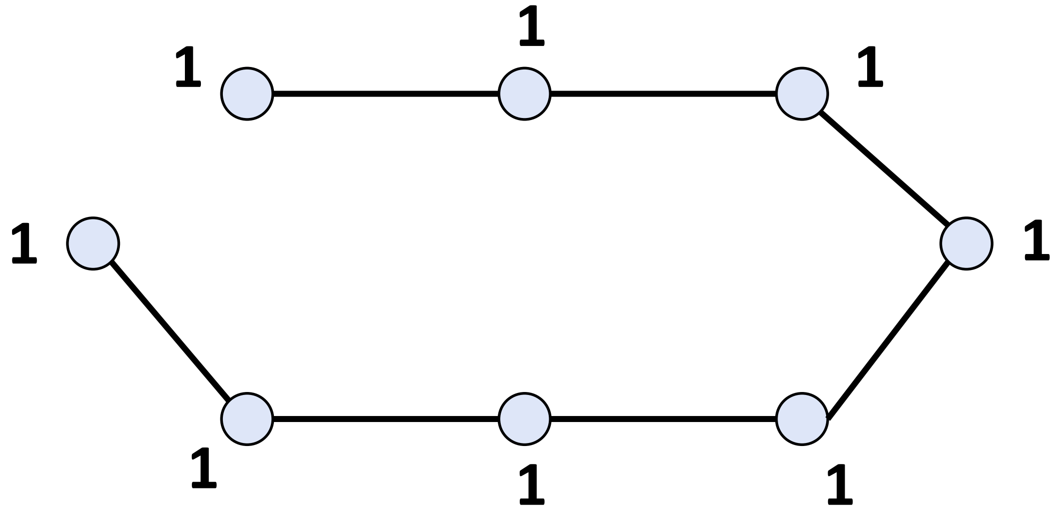
Why Approximate *k*-Core Decomposition

- **Dynamic exact *k*-core decomposition:**
  - **$\Omega(n)$ work, $\Omega(n)$ depth, parallel** [Aridhi et al '16, Gabert et al. '21, Hua et al. '20, Jin et al. '18, Wang '17]
  - One update can cause **$\Omega(n)$ coreness changes**



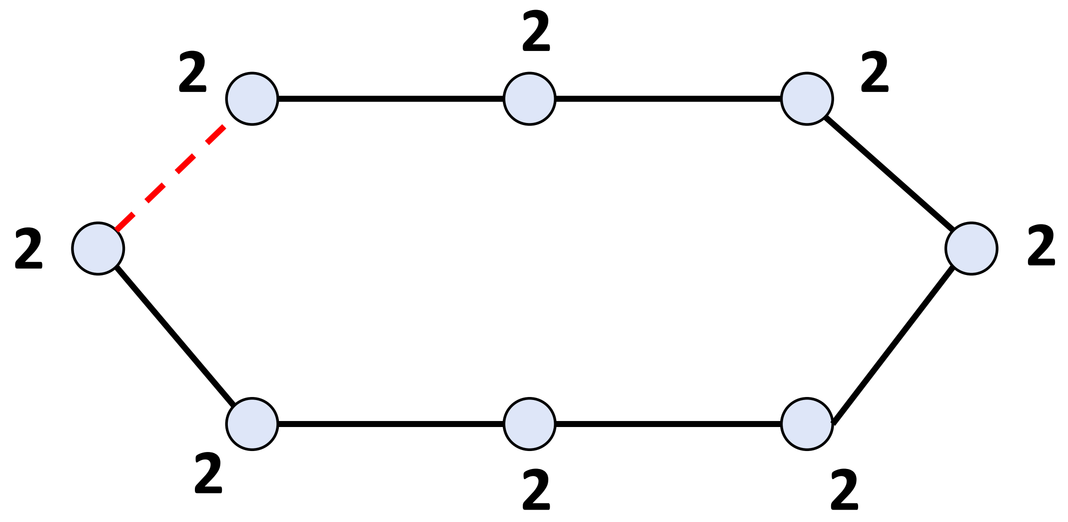
Why Approximate *k*-Core Decomposition

- **Dynamic exact *k*-core decomposition:**
  - **$\Omega(n)$ work, $\Omega(n)$ depth, parallel** [Aridhi et al '16, Gabert et al. '21, Hua et al. '20, Jin et al. '18, Wang '17]
  - One update can cause **$\Omega(n)$ coreness changes**



Why Approximate *k*-Core Decomposition

- **Dynamic exact *k*-core decomposition:**
  - **$\Omega(n)$ work, $\Omega(n)$ depth, parallel** [Aridhi et al '16, Gabert et al. '21, Hua et al. '20, Jin et al. '18, Wang '17]
  - One update can cause **$\Omega(n)$ coreness changes**



Why Approximate *k*-Core Decomposition

- **Dynamic approximate *k*-core decomposition:**
  - $O(\log^2 n)$ **time amortized, sequential,** $(2+\varepsilon)$**-approximation** [Sun et al. '20]
  - Can accumulate error, **charge time to updates**
  - **Threshold peeling** procedure

Why Approximate *k*-Core Decomposition

- **Dynamic approximate *k*-core decomposition:**
  - $O(\log^2 n)$ **time amortized, sequential,** $(2+\varepsilon)$**-approximation** [Sun et al. '20]
  - Can accumulate error, **charge time to updates**
  - **Threshold peeling** procedure

Does not use parallelism

One update at a time

Why Approximate *k*-Core Decomposition

- **Dynamic approximate *k*-core decomposition:**
  - $O(\log^2 n)$ **time amortized, sequential,** $(2+\varepsilon)$**-approximation** [Sun et al. '20]
  - Can accumulate error, **charge time to updates**
  - **Threshold peeling** procedure

Caveat: amortized $O(\log^2 n)$ depth,
worst-case $\Omega(n)$ depth

Want: worst-case $\text{poly}(\log n)$ depth

Batch Dynamic *k*-Core Decomposition

- $(2+\epsilon)$**-approximation** for coreness of every vertex

Batch Dynamic *k*-Core Decomposition

- $(2+\epsilon)$**-approximation** for coreness of every vertex
- $O(B\log^2 n)$ **amortized work** and $O(\log^2 n \log\log n)$ **depth** with high probability, size B batch

Batch Dynamic *k*-Core Decomposition

- $(2+\epsilon)$**-approximation** for coreness of every vertex

- $\boldsymbol{O(B\log^2 n)}$ **amortized work** and $\boldsymbol{O(\log^2 n \log\log n)}$ **depth** with high probability, size $\boldsymbol{B}$ batch

- Is **work-efficient**, matches Sun et al. ‘20

Batch Dynamic *k*-Core Decomposition

- $(2+\epsilon)$**-approximation** for coreness of every vertex
- $O(B\log^2 n)$ **amortized work** and $O(\log^2 n \log\log n)$ **depth** with high probability, size B batch
- Is **work-efficient**, matches Sun et al. '20
- **Based on a parallel level data structure (PLDS)**

Batch Dynamic *k*-Core Decomposition **+ Others!**

- $(2+\epsilon)$**-approximation** for coreness of every vertex
- $O(B\log^2 n)$ **amortized work** and $O(\log^2 n \log\log n)$ **depth** with high probability, size B batch
- Is **work-efficient**, matches Sun et al. ’20
- **Based on a parallel level data structure (PLDS)**

Static *k*-Core Decomposition
Low Out-Degree Orientation
Maximal Matching
Clique Counting
Vertex Coloring

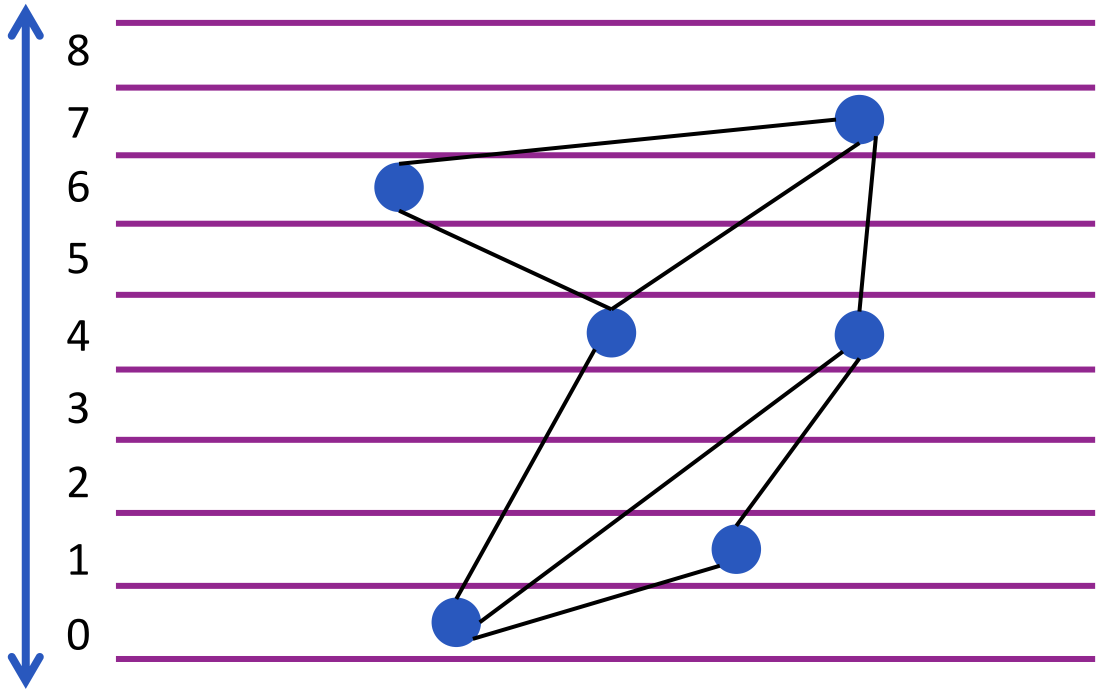
Sequential Level Data Structures for Dynamic Problems

- Maximal Matching [Baswana-Gupta-Sen 18, Solomon ‘16]
- $(\Delta + 1)$-Coloring [Bhattacharya-Chakrabarty-Henzinger-Nanongkai ‘18, Bhattacharya-Grandoni-Kulkarni-**L**-Solomon ‘19]
- Clustering [Wulff-Nilsen ‘12]
- Low out-degree orientation [Solomon-Wein 20, Henzinger-Neumann-Weiss ‘20]
- Densest subgraph [Bhattacharya-Henzinger-Nanongkai-Tsourakakis ‘15]

Sequential Level Data Structure (LDS)

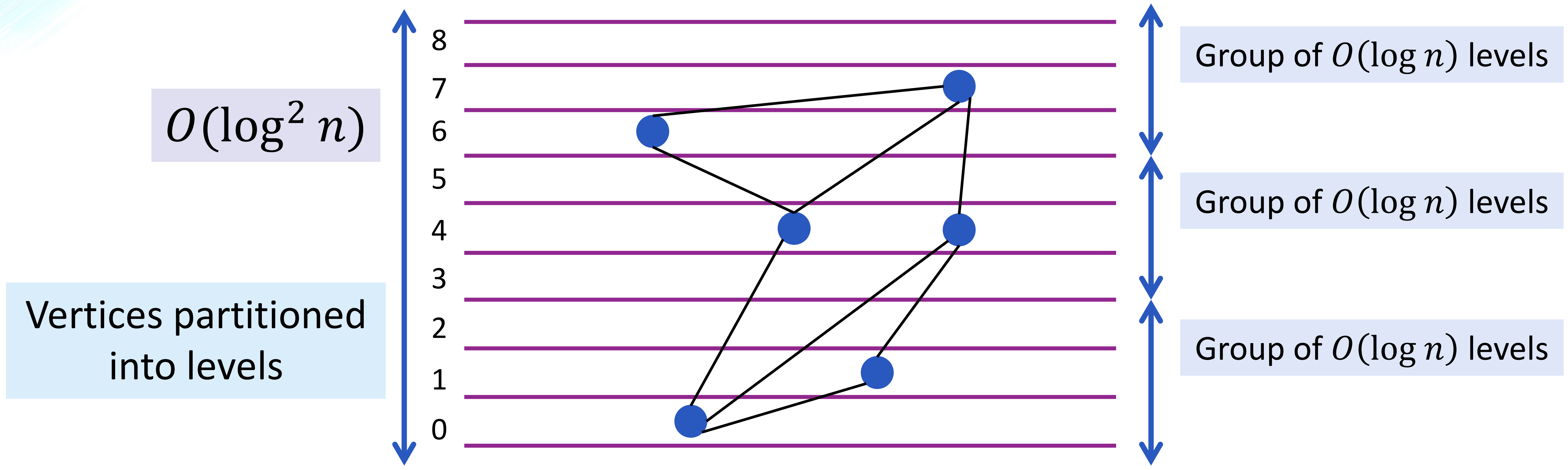
$O(\log^2 n)$

Vertices partitioned into levels



Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015
Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)

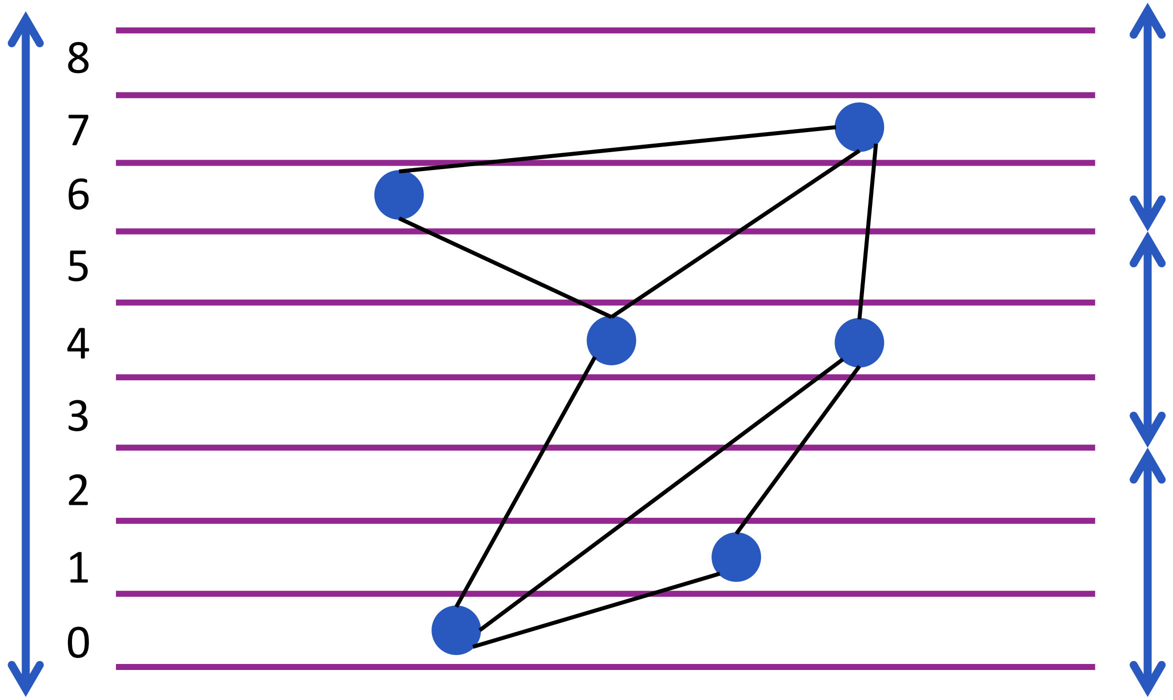


Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015
Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)

$O(\log^2 n)$

Vertices partitioned into levels



Group of $O(\log n)$ levels

Cut-off: $(1+\epsilon)^i$

Group of $O(\log n)$ levels

Cut-off: $(1+\epsilon)^{i-1}$

Group of $O(\log n)$ levels

Cut-off: $(1+\epsilon)^{i-2}$

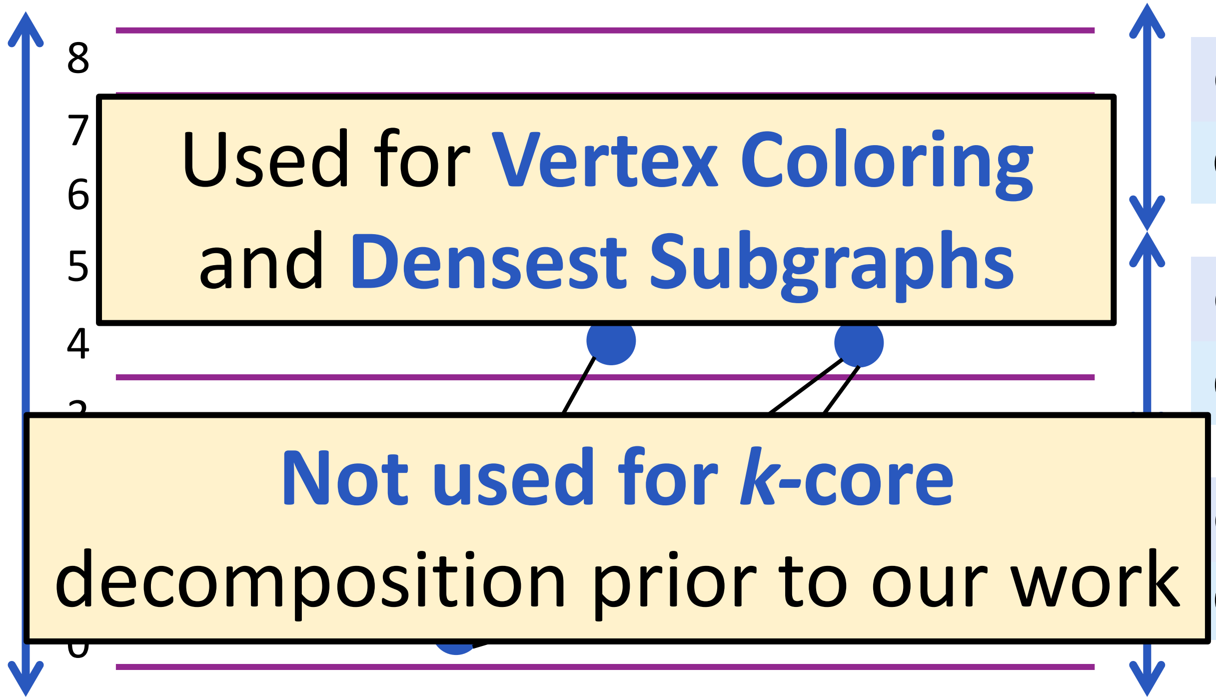
...

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015
Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)

$O(\log^2 n)$

Vertices partitioned into levels



Group of $O(\log n)$ levels

Cut-off: $(1+\epsilon)^i$

Group of $O(\log n)$ levels

Cut-off: $(1+\epsilon)^{i-1}$

Group of $O(\log n)$ levels

Cut-off: $(1+\epsilon)^{i-2}$

…

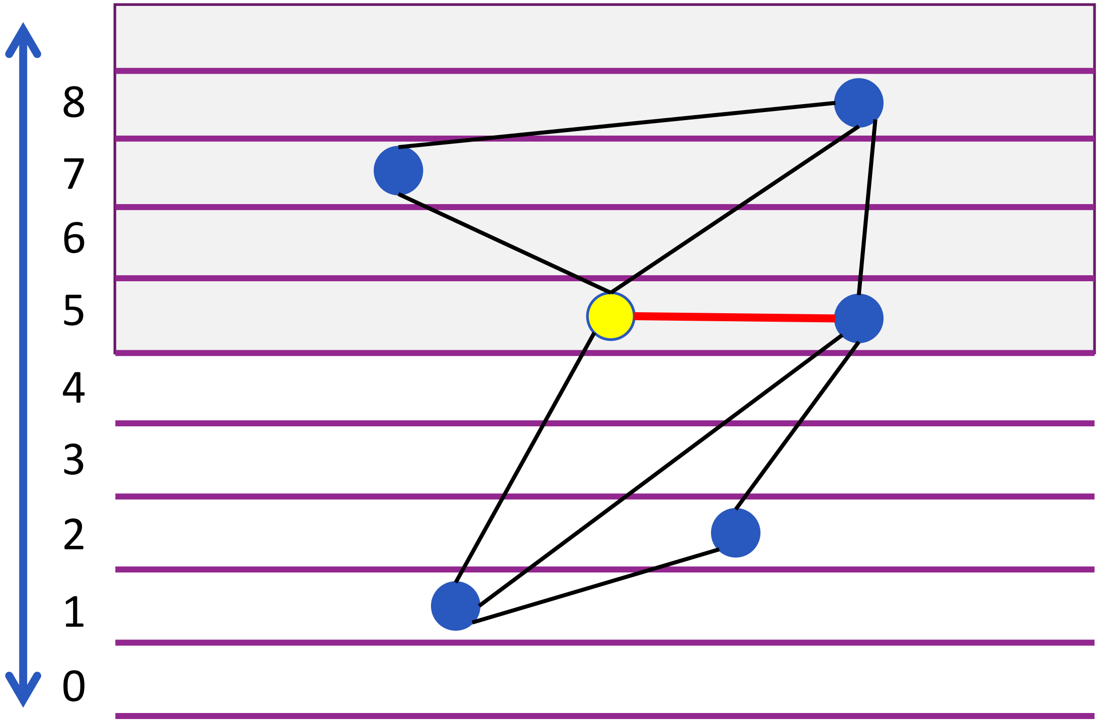
Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015

Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)

$O(\log^2 n)$

Vertices partitioned into levels



neighbors: $> 2.1(1+\epsilon)^i$

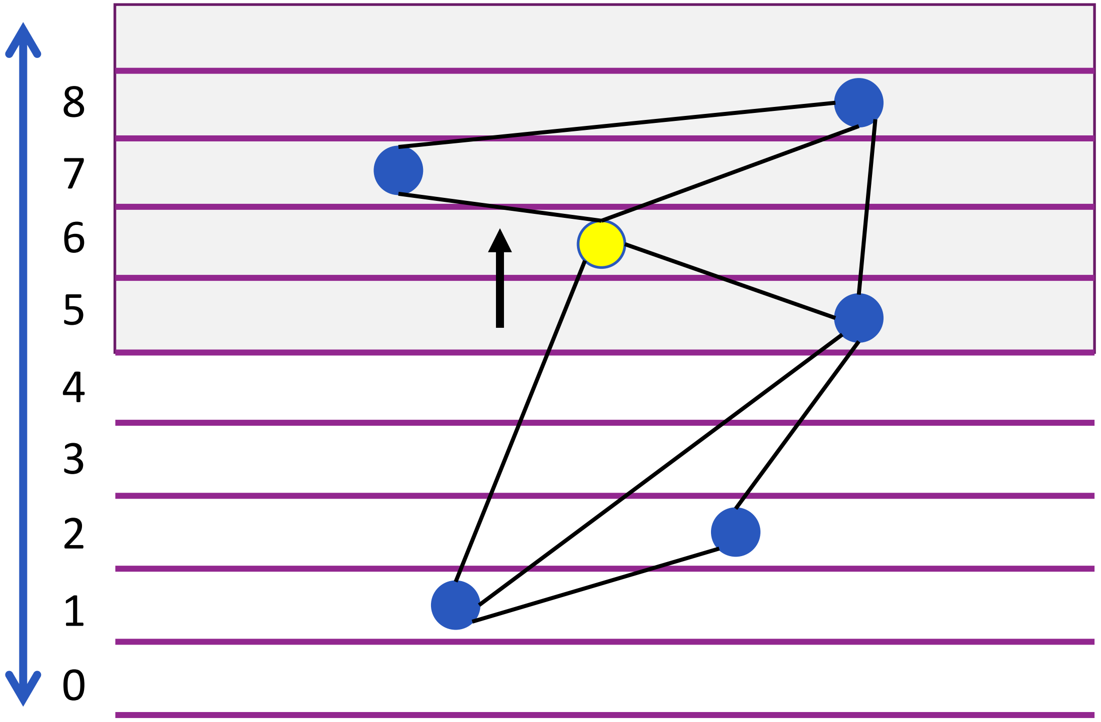
= edge insertion

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)

$O(\log^2 n)$

Vertices partitioned into levels



neighbors: $> 2.1(1+\epsilon)^i$

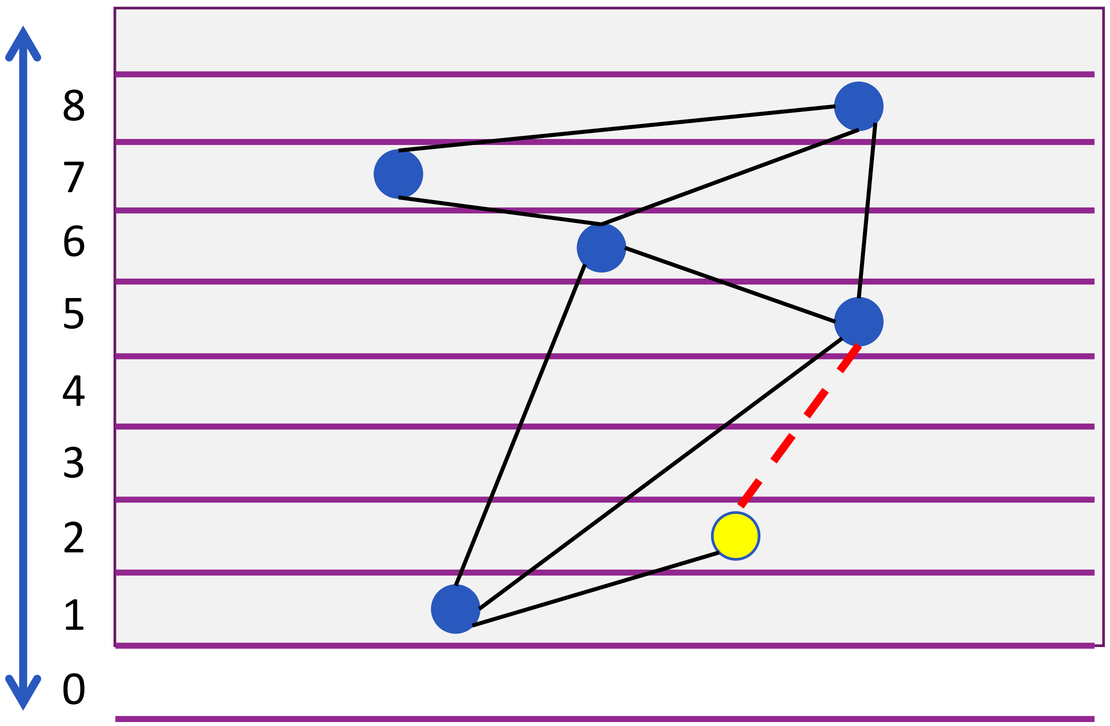
= edge insertion

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)

$O(\log^2 n)$

Vertices partitioned into levels



= edge deletion

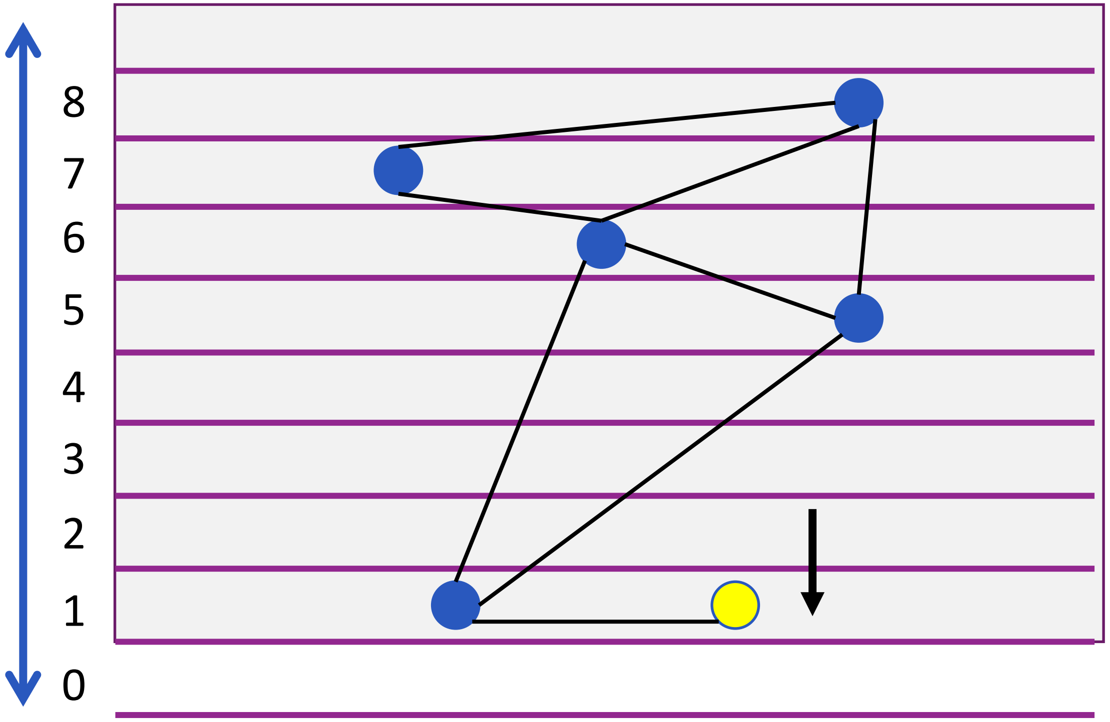
neighbors: $< (1+\epsilon)^i$

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)

$O(\log^2 n)$

Vertices partitioned into levels



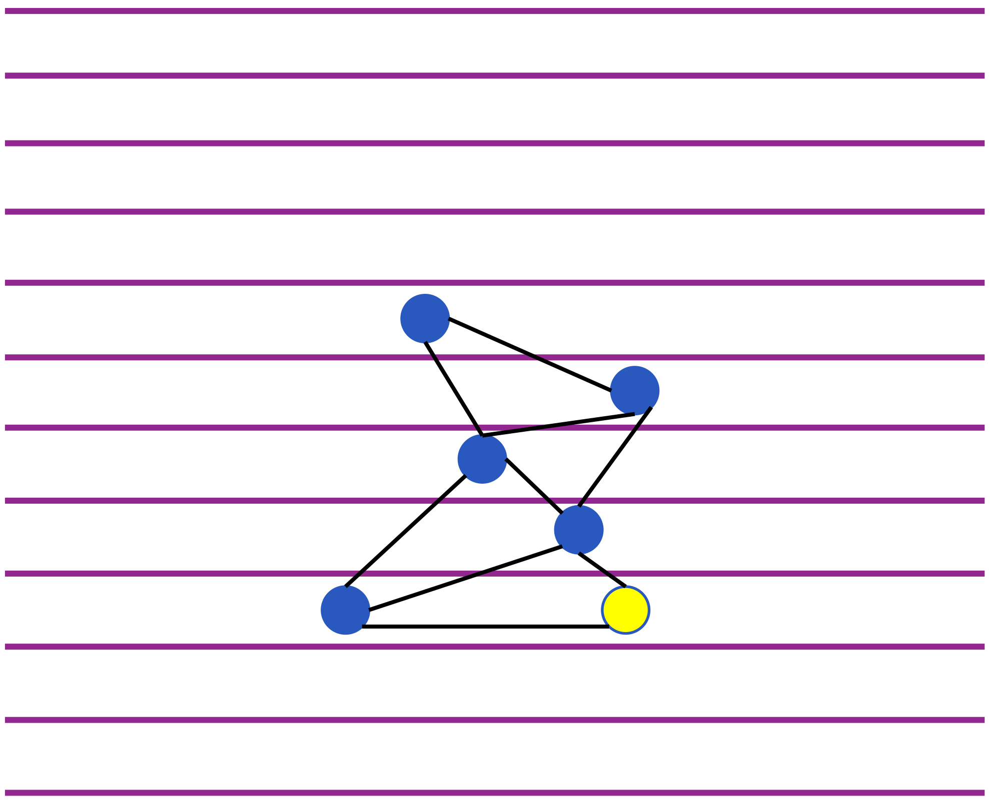
= edge deletion

neighbors: $< (1+\epsilon)^i$

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Difficulties with Parallelization

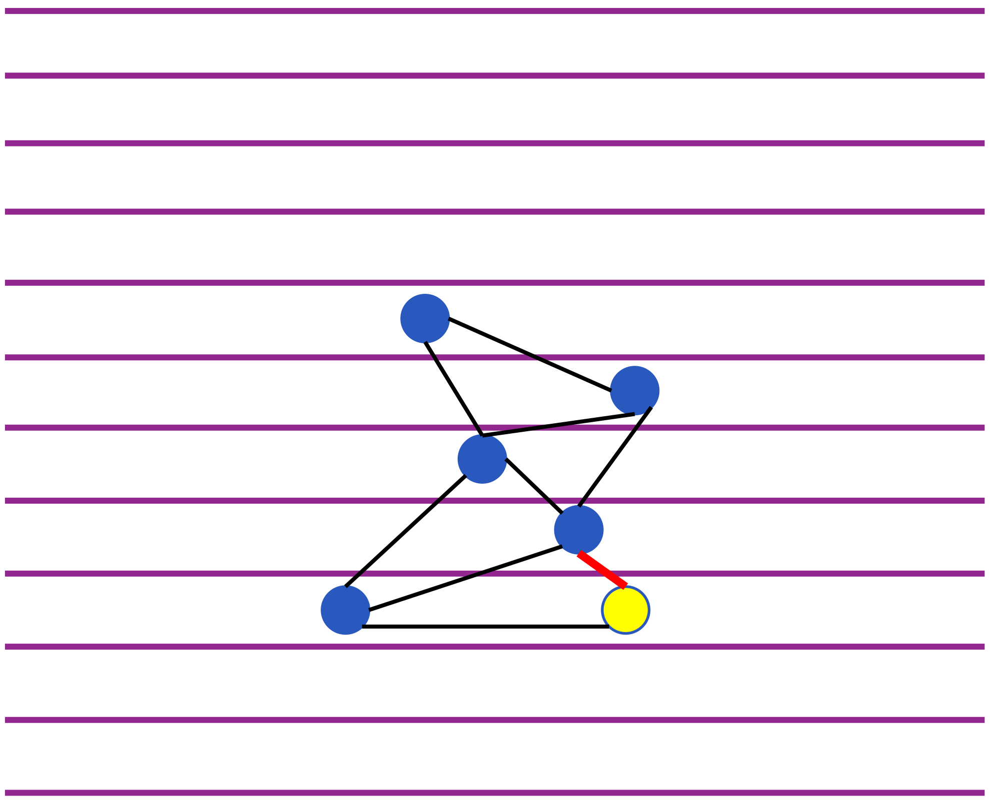
Large sequential dependencies



Large depth

Difficulties with Parallelization

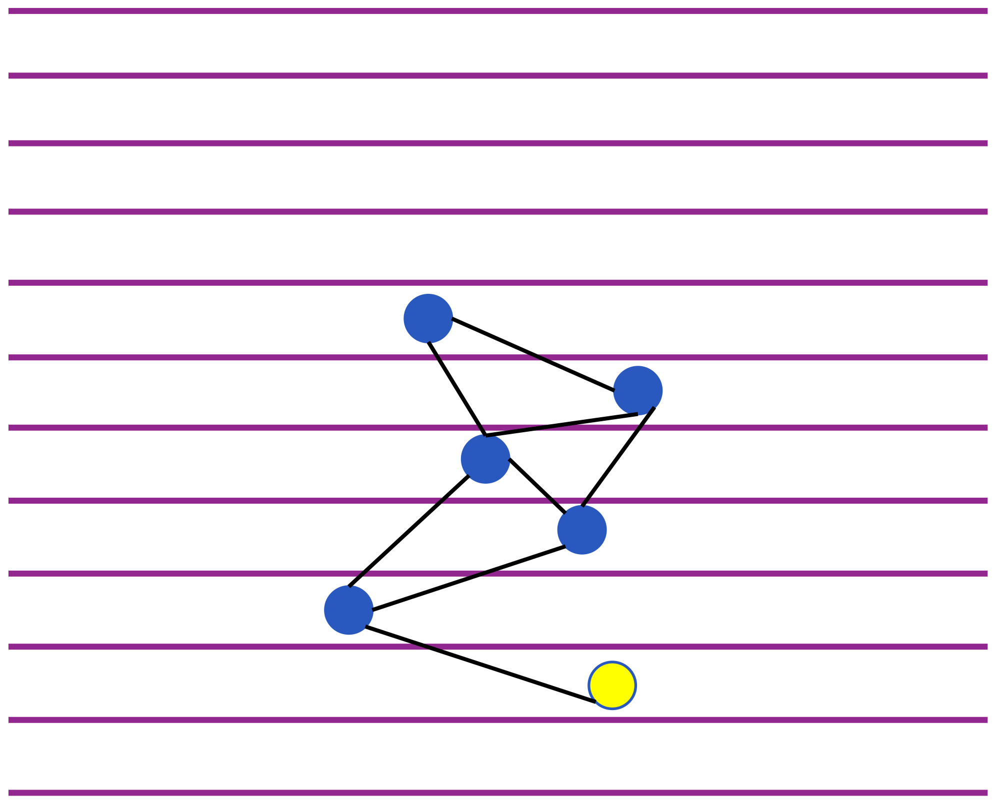
Large sequential dependencies



Large depth

Difficulties with Parallelization

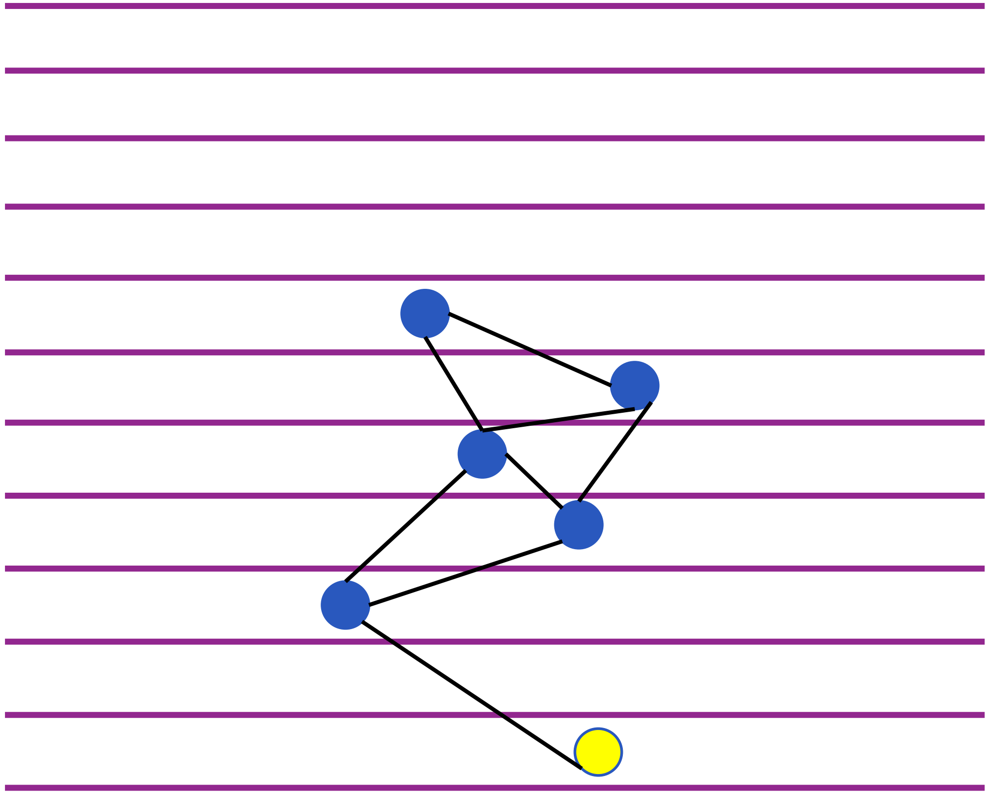
Large sequential dependencies



Large depth

Difficulties with Parallelization

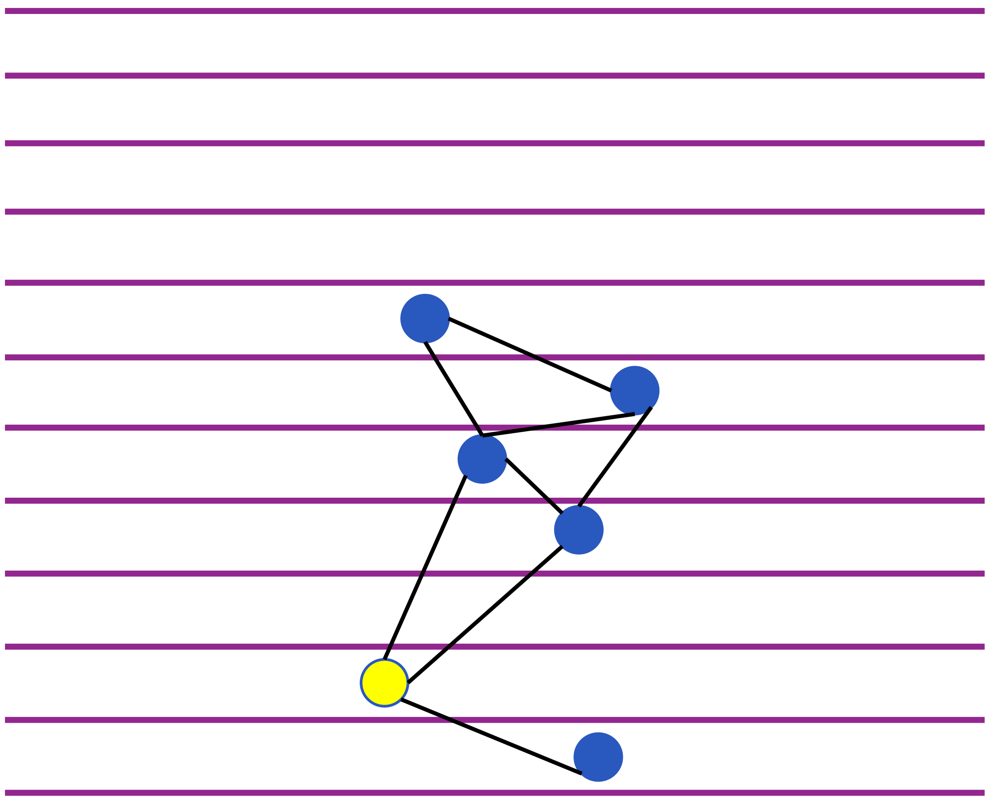
Large sequential dependencies



Large depth

Difficulties with Parallelization

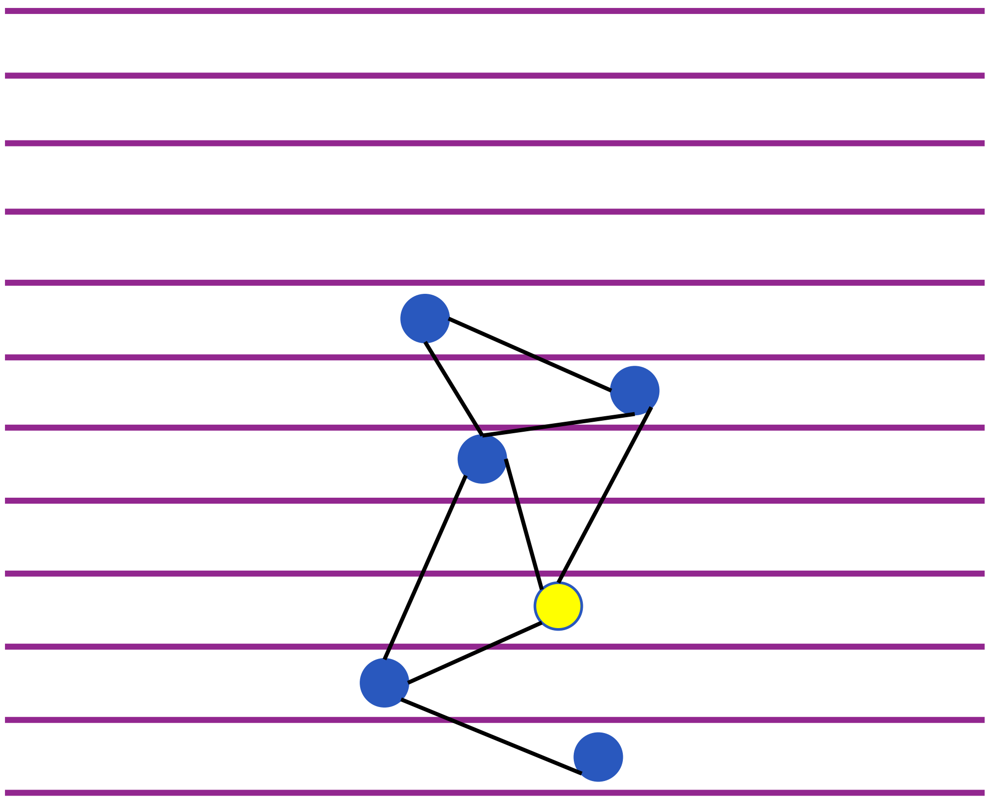
Large sequential dependencies



Large depth

Difficulties with Parallelization

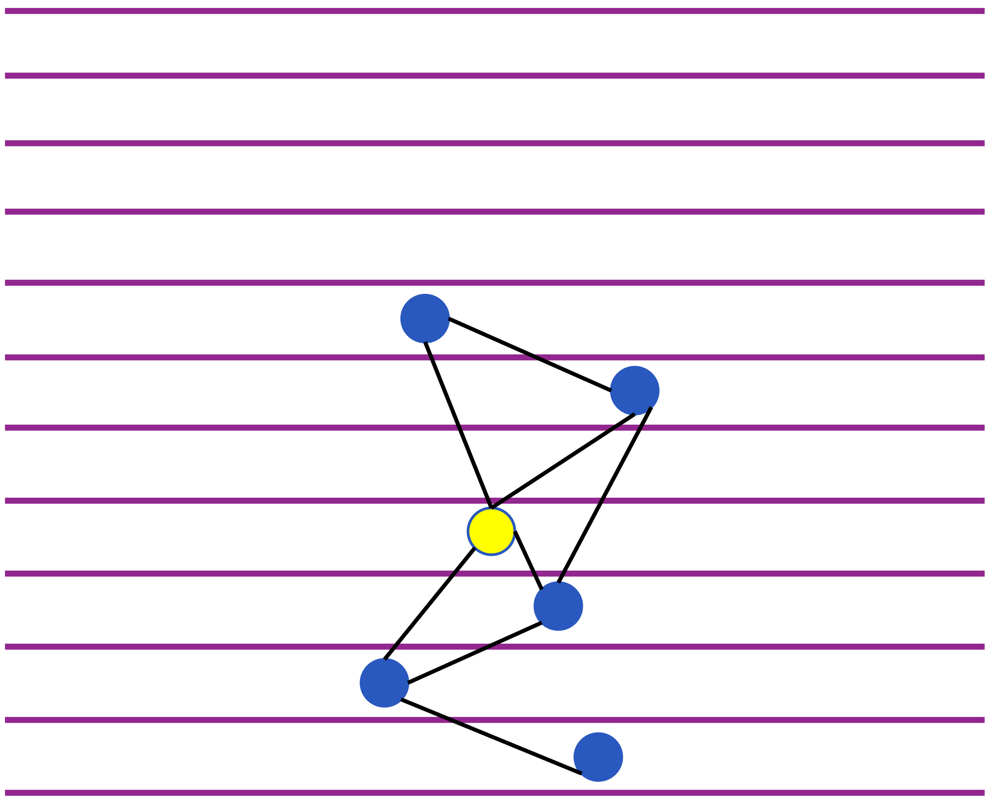
Large sequential dependencies



Large depth

Difficulties with Parallelization

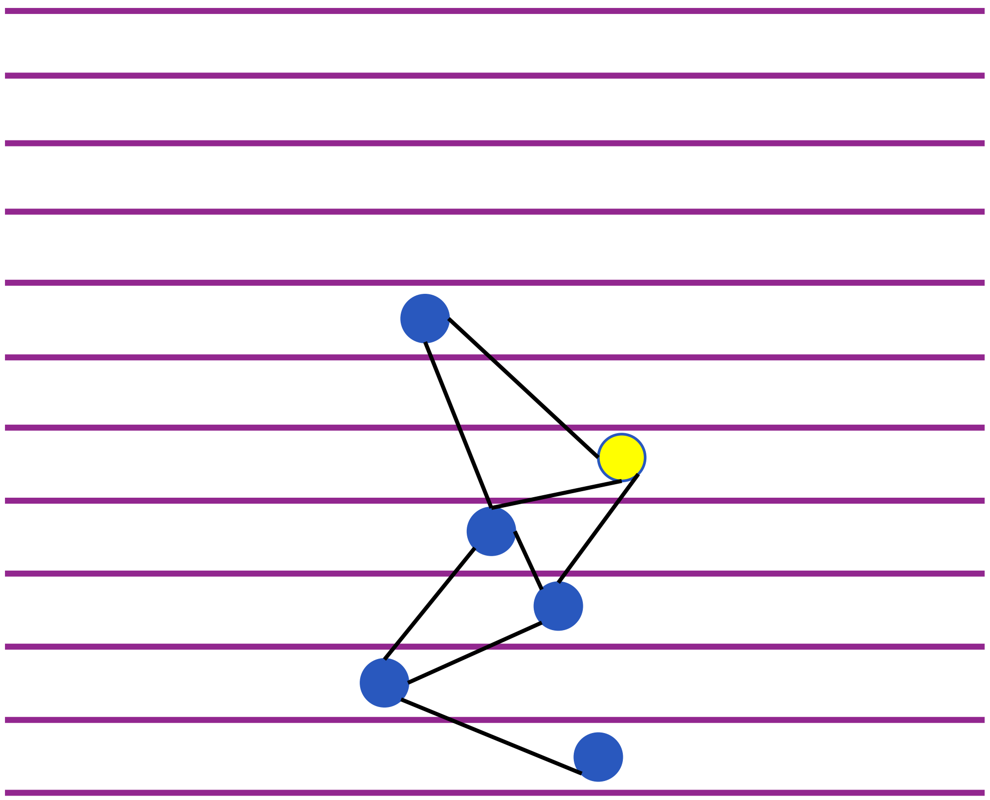
Large sequential dependencies



Large depth

Difficulties with Parallelization

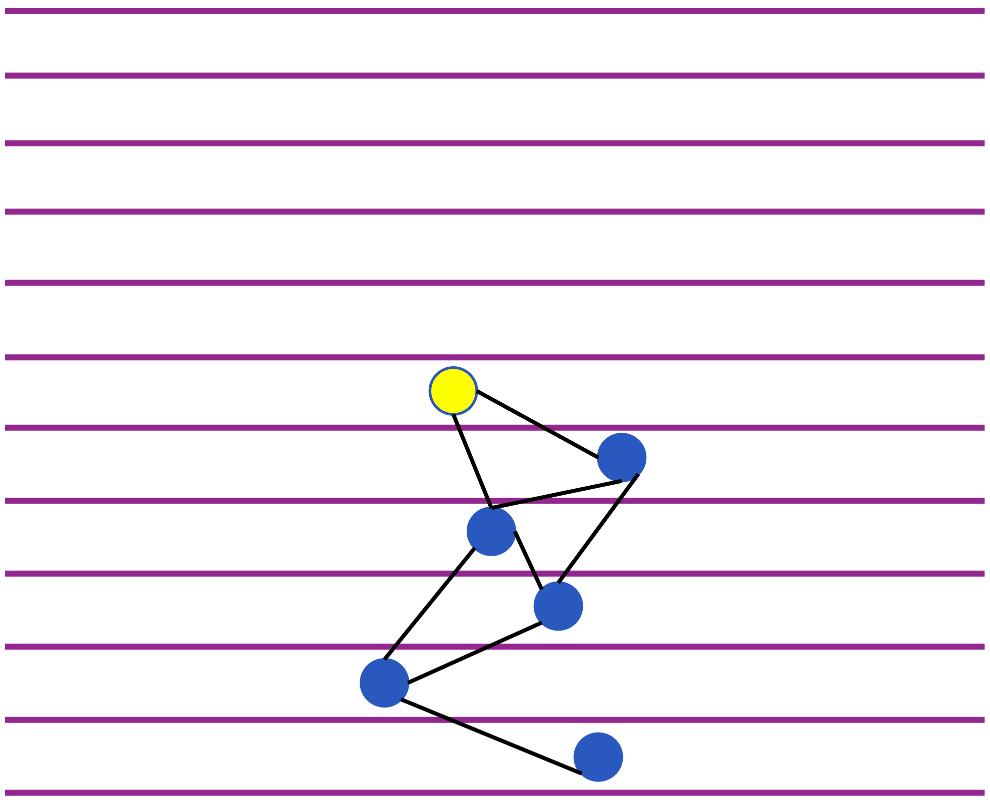
Large sequential dependencies



Large depth

Difficulties with Parallelization

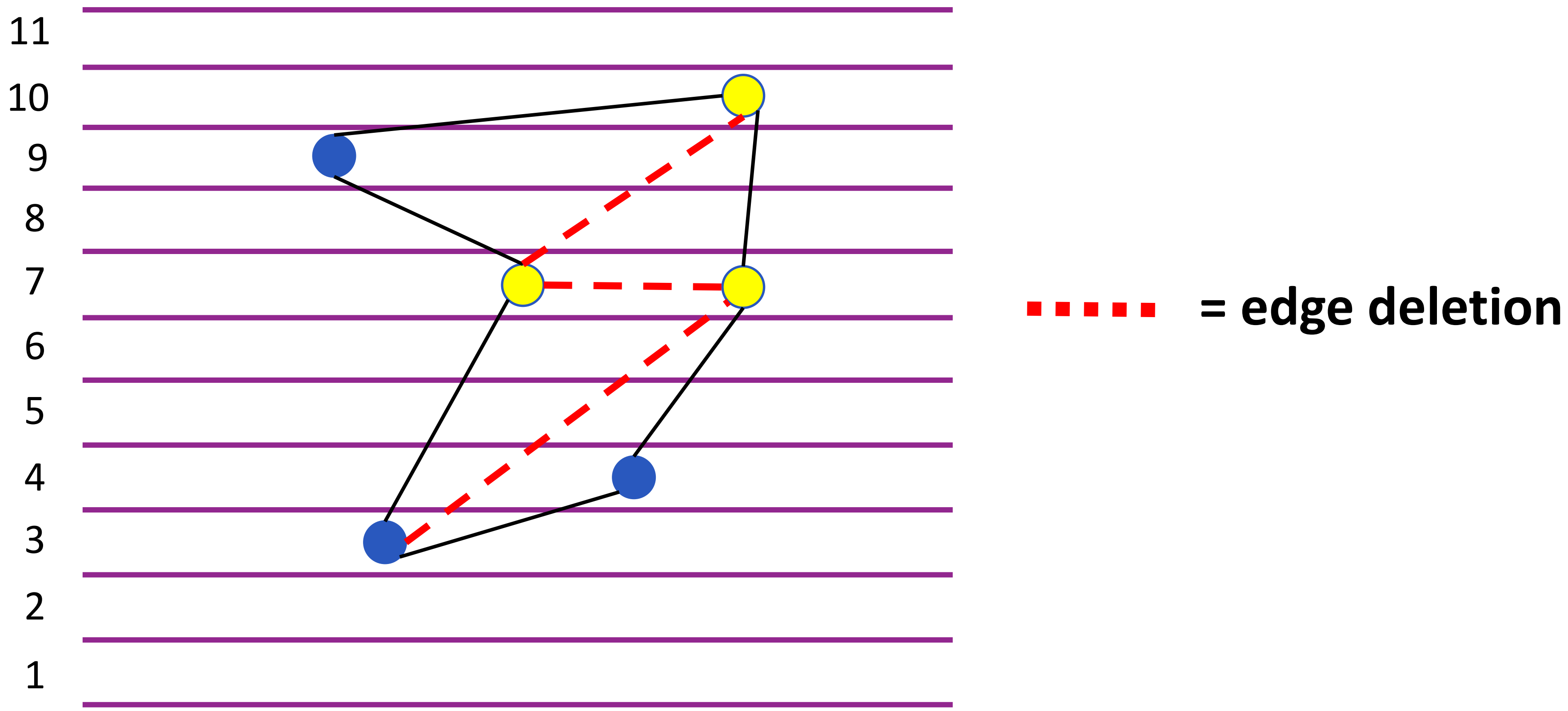
Large sequential dependencies



Large depth

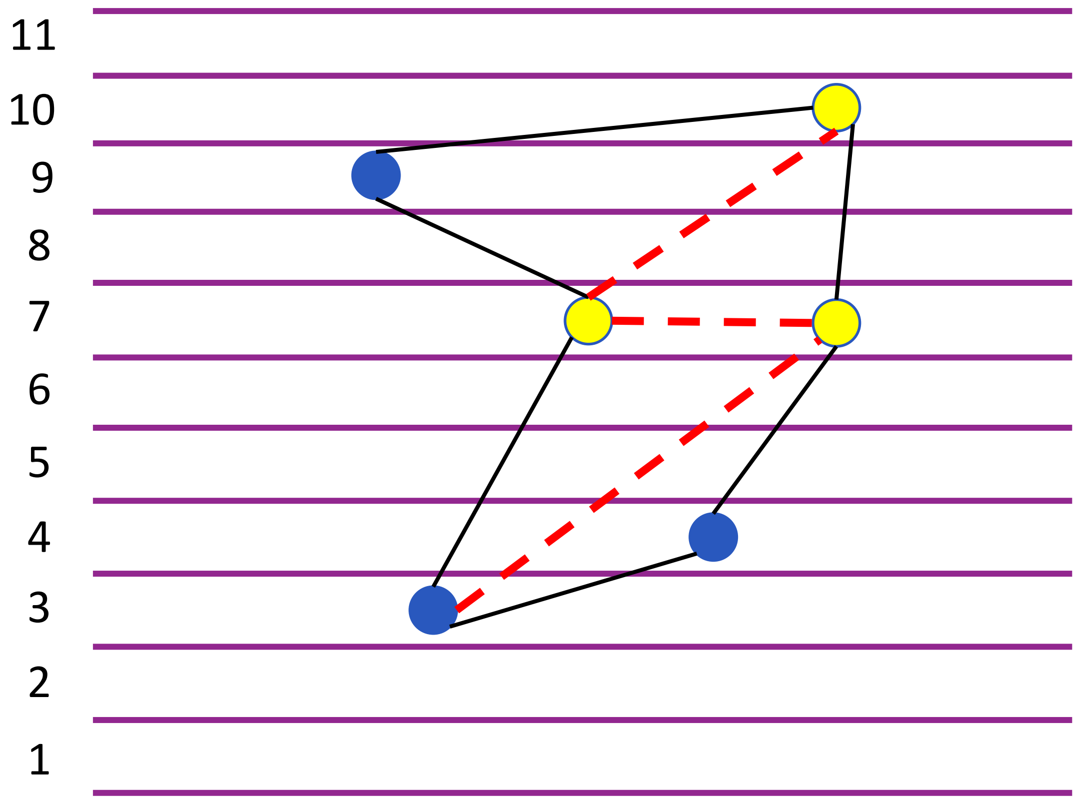
Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions



Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions



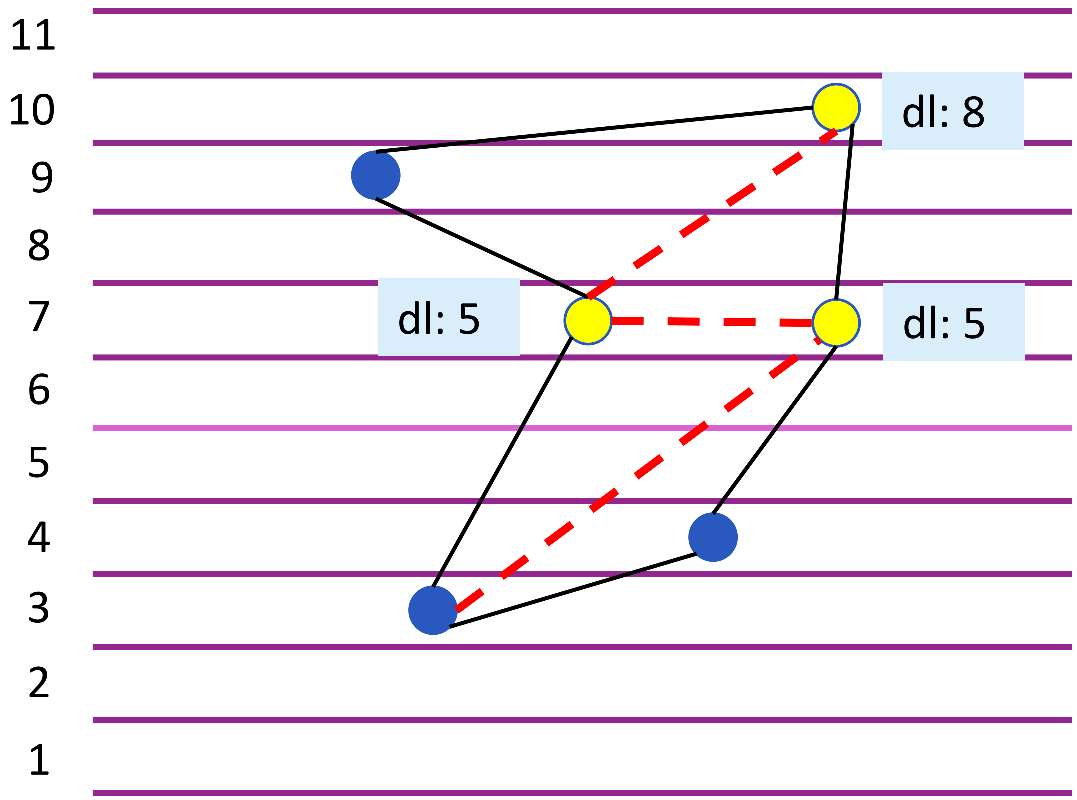
= edge deletion

Only lower bound cutoff, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions

Calculate *desire-level*: **closest level that satisfies cutoffs**



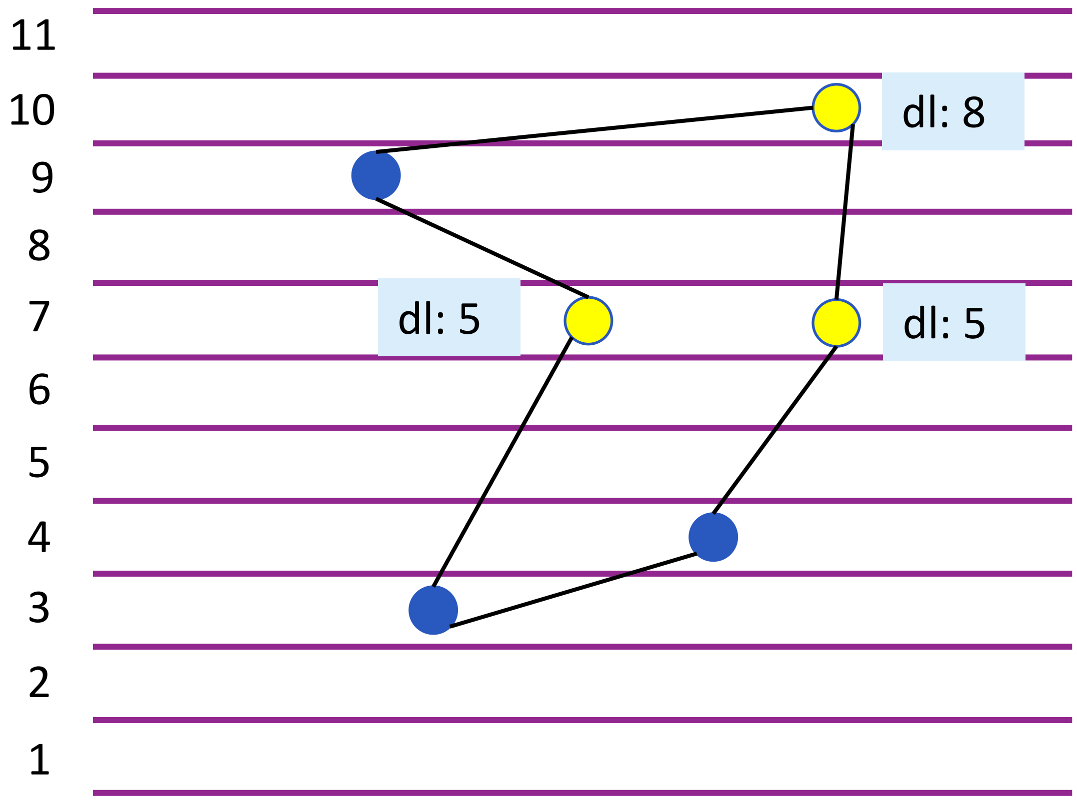
= **edge deletion**

Only lower bound cutoff, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions

Calculate *desire-level*: **closest level that satisfies cutoffs**



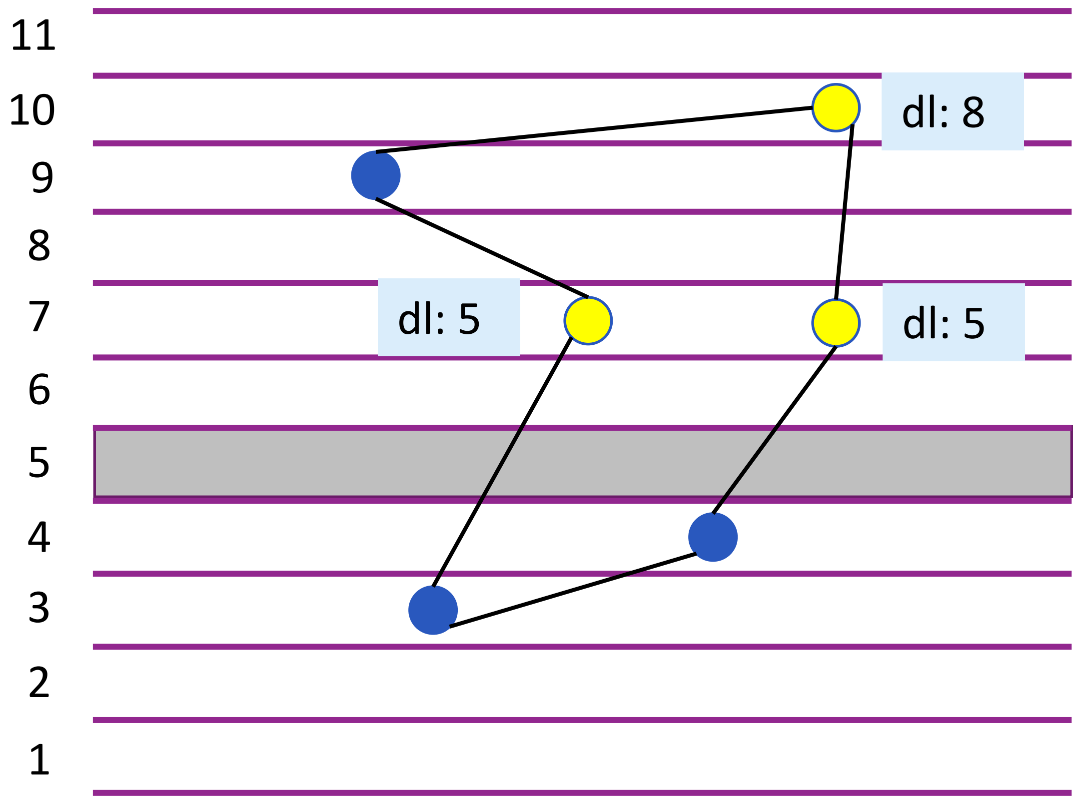
Iterate from **bottommost level to top level** and move vertices to desire-level

Only lower bound cutoff, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions

Calculate *desire-level*: **closest level that satisfies cutoffs**



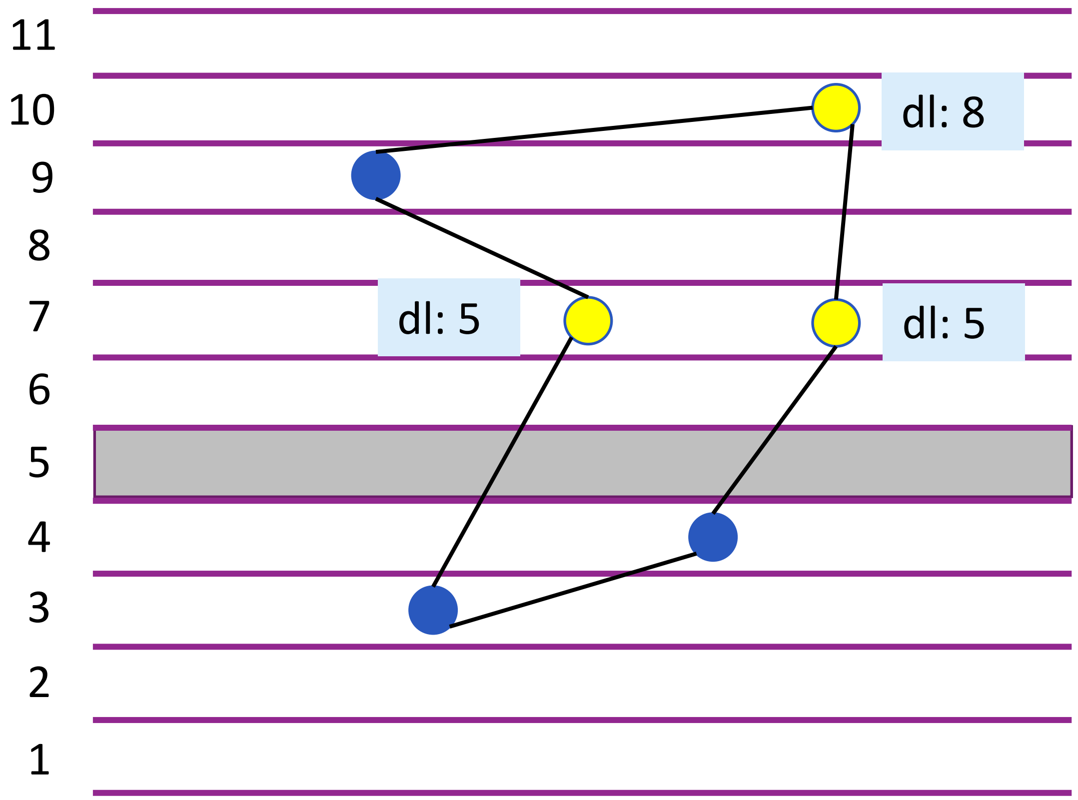
Iterate from **bottommost level to top level** and move vertices to desire-level

Only lower bound cutoff, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions

Calculate *desire-level*: **closest level that satisfies cutoffs**



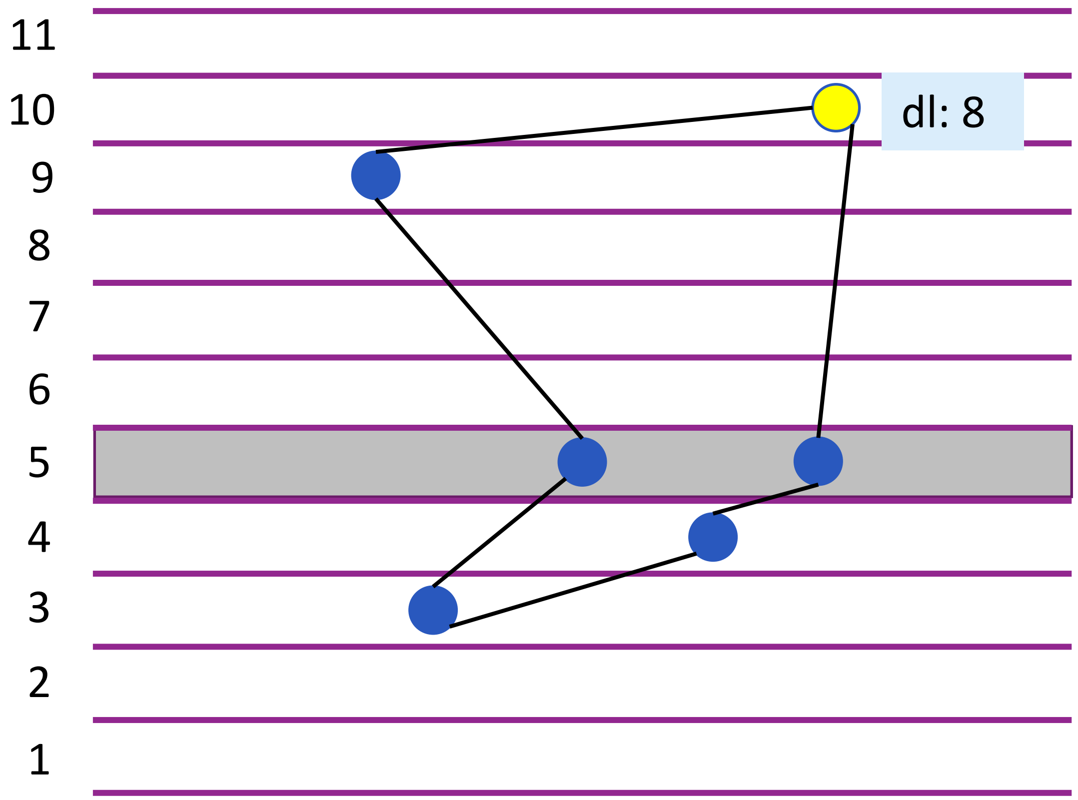
Iterate from **bottommost level to top level** and move vertices to desire-level

Only lower bound cutoff, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions

Calculate *desire-level*: **closest level that satisfies cutoffs**



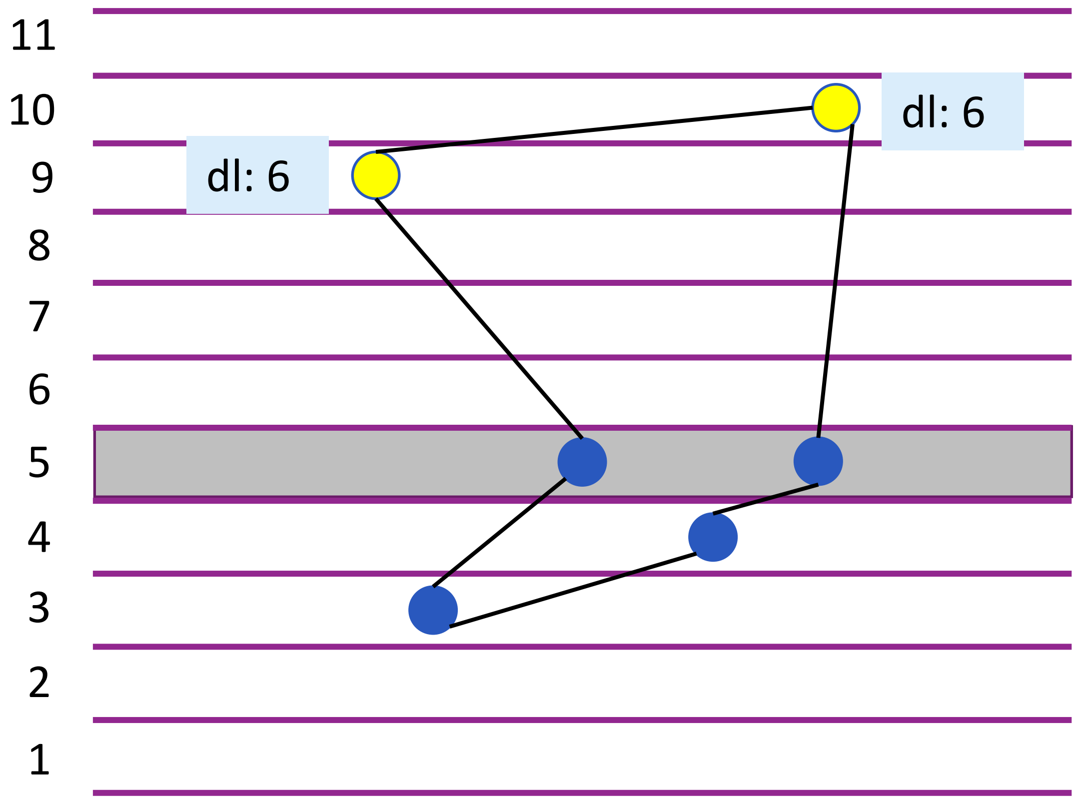
Iterate from **bottommost level to top level** and move vertices to desire-level

Only lower bound cutoff, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions

Calculate *desire-level*: **closest level that satisfies cutoffs**



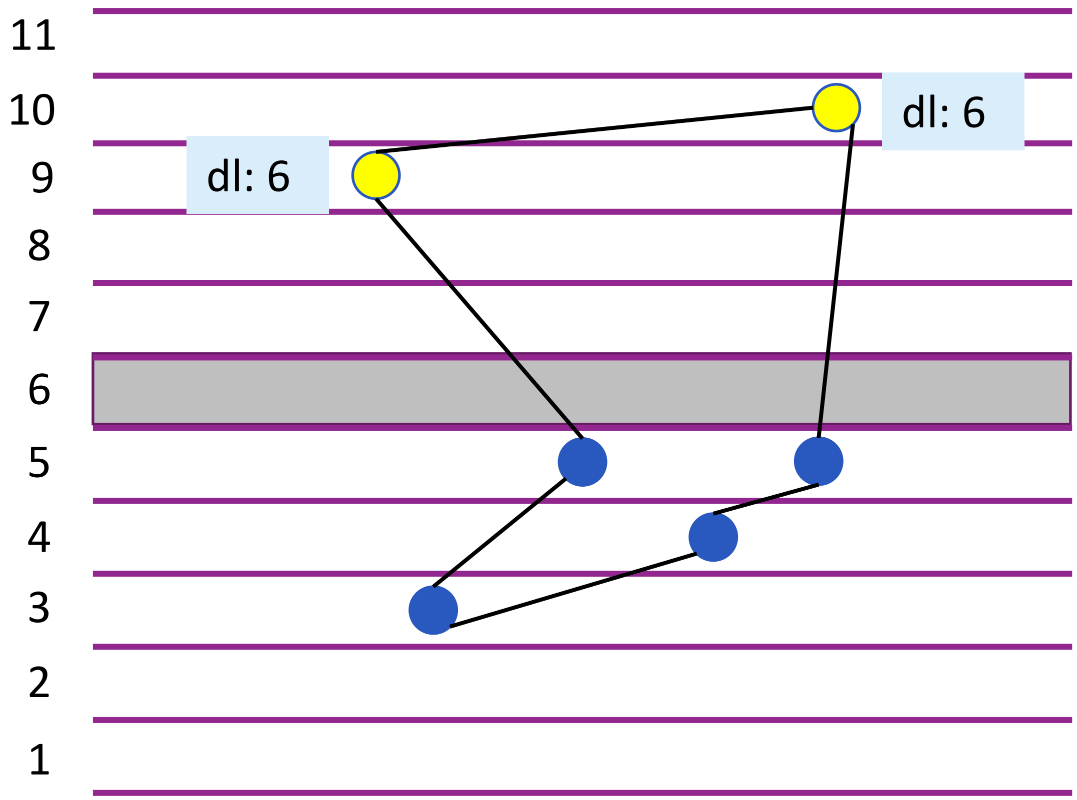
Iterate from **bottommost level to top level** and move vertices to desire-level

Only lower bound cutoff, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions

Calculate *desire-level*: **closest level that satisfies cutoffs**



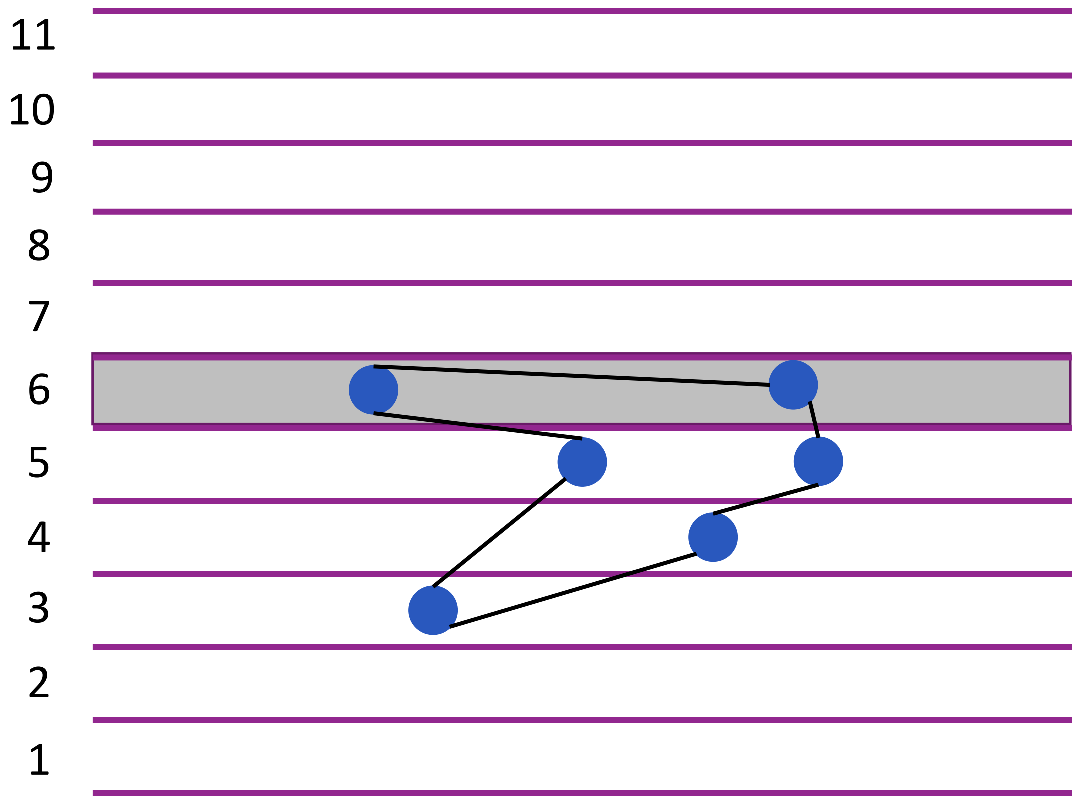
Iterate from **bottommost level to top level** and move vertices to desire-level

Only lower bound cutoff, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions

Calculate *desire-level*: **closest level that satisfies cutoffs**



Iterate from **bottommost level to top level** and move vertices to desire-level

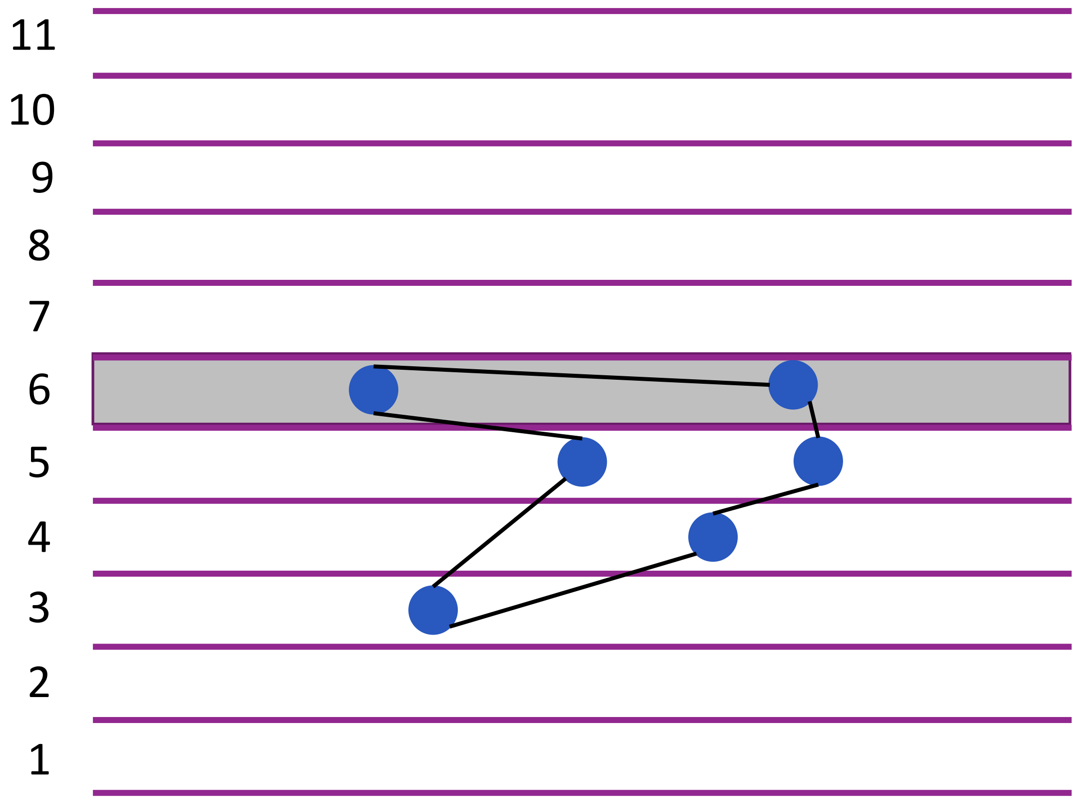
Only lower bound cutoff, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions

Vertices need to move at most ONCE, unlike sequential LDS!

Calculate *desire-level*: **closest level that satisfies cutoffs**



Iterate from **bottommost level to top level** and move vertices to desire-level

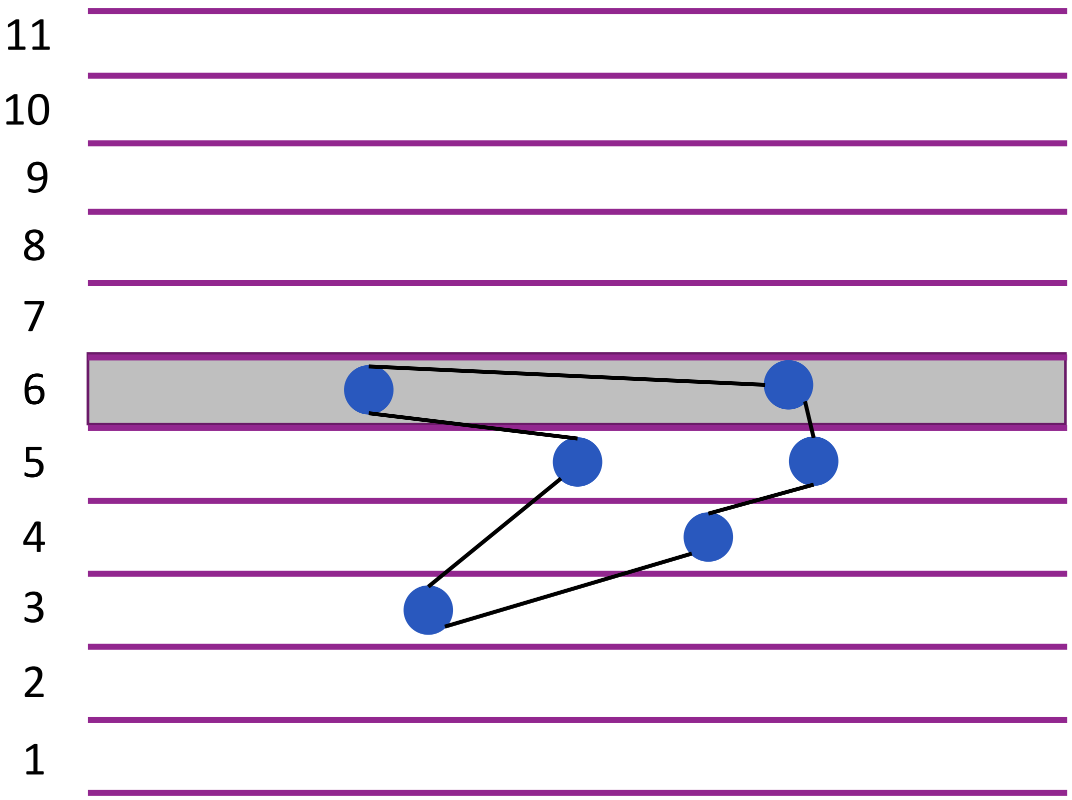
Only lower bound cutoff, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Vertices need to move at most ONCE, unlike sequential LDS!

Calculate *desire-level*: **closest level that satisfies cutoffs**

Deletions



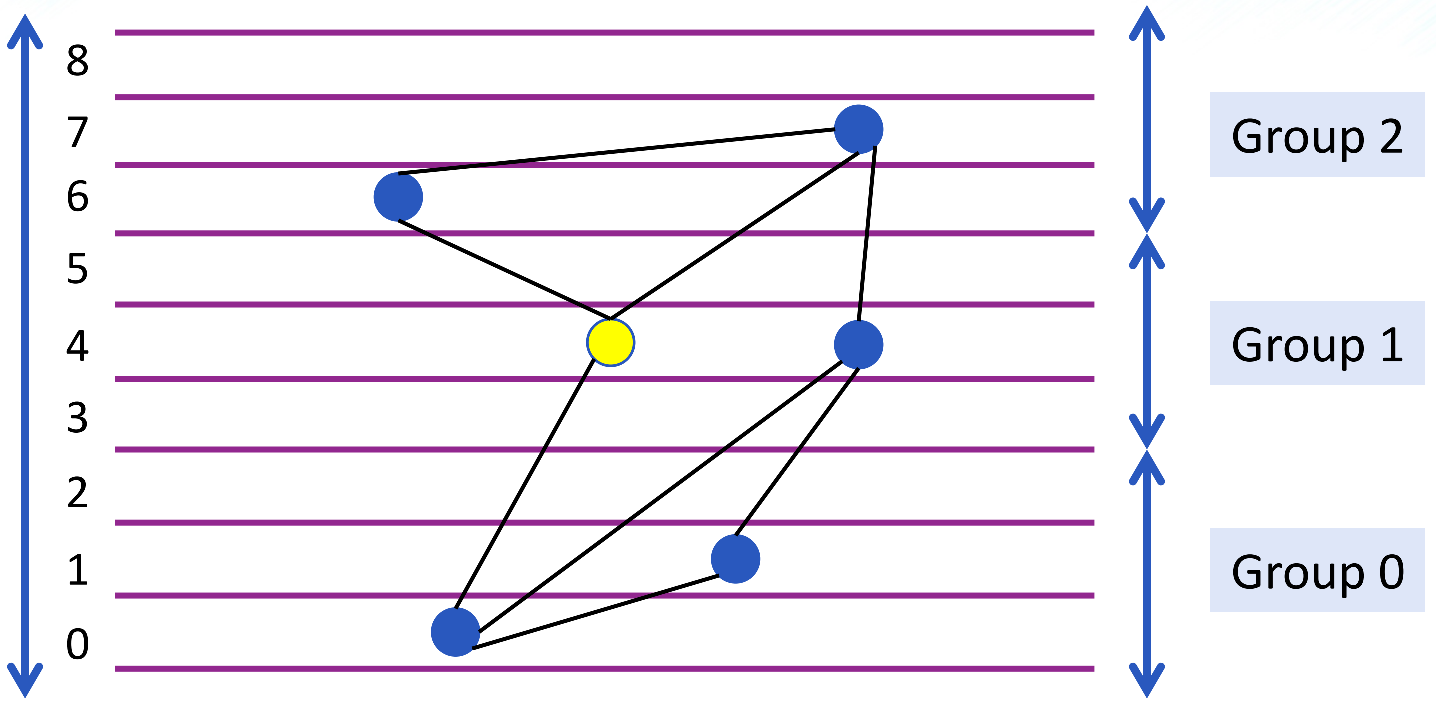
$O(\log^2 n \log\log n)$ depth w.h.p

Iterate from **bottommost level to top level** and move vertices to desire-level

Only lower bound cutoff, $(1+\epsilon)^i$, ever violated.

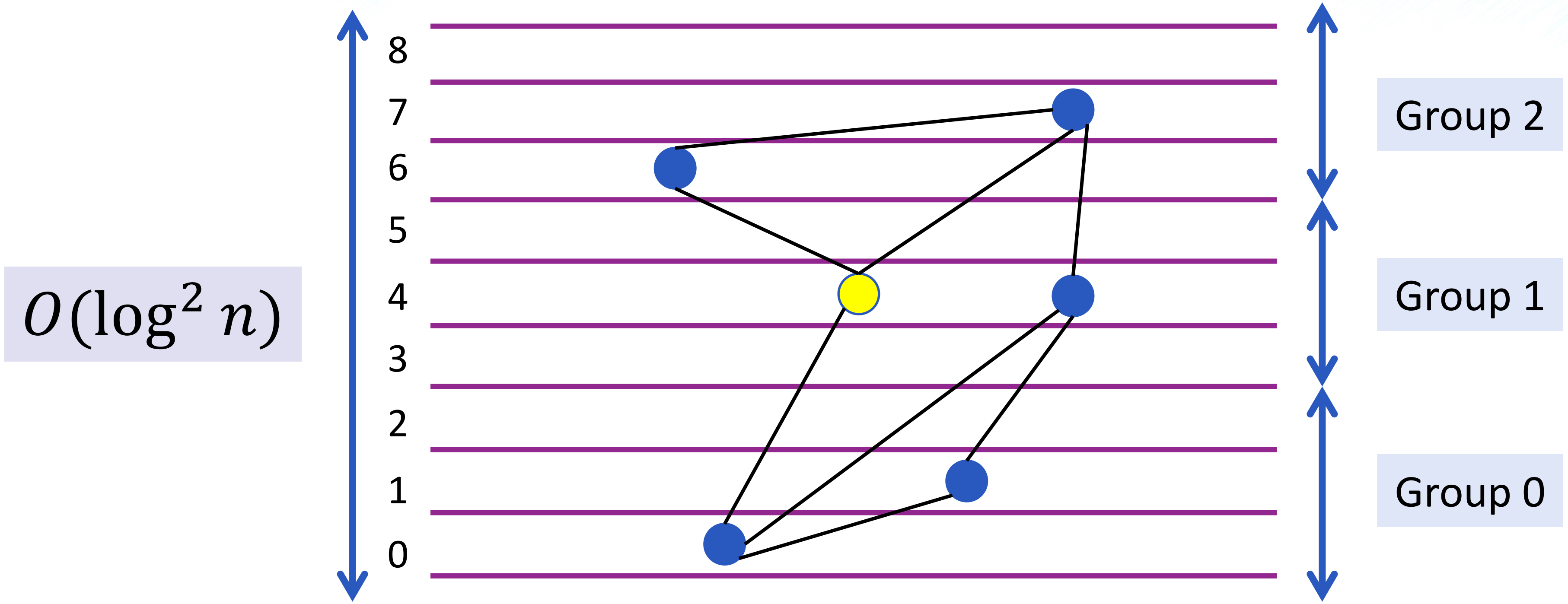
Obtaining the Coreness Estimate

$O(\log^2 n)$



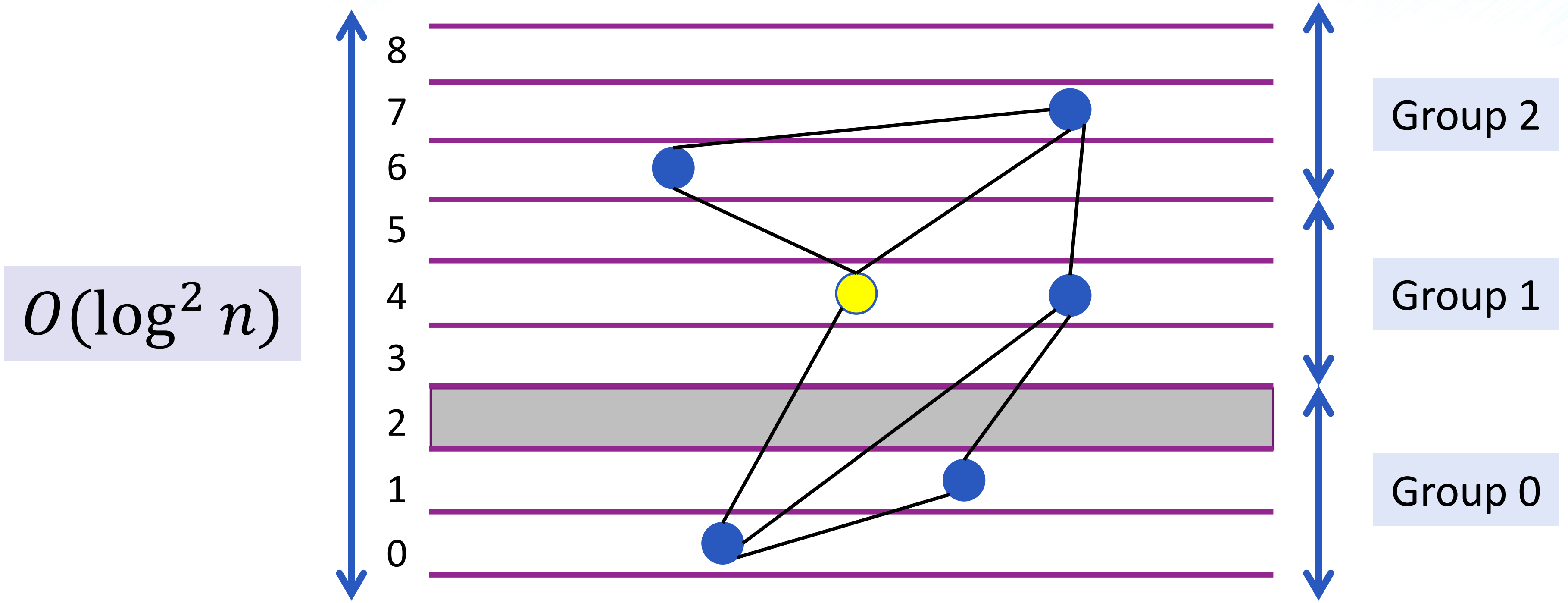
- Set the coreness estimate: $\mathbf{(1+\delta)}^{\boldsymbol{max(\lfloor (level(v)+1)/\,(4\lceil \log_{1+\delta} n\rceil)\rfloor - 1, 0)}}$
- Each group has $4\lceil\log_{1+\delta} n\rceil$ levels

Obtaining the Coreness Estimate



- Set the coreness estimate: $\mathbf{(1+\delta)^{max(\lfloor(level(v)+1)/(4\lceil\log_{1+\delta} n\rceil)\rfloor-1,0)}}$
- Each group has $4\lceil\log_{1+\delta} n\rceil$ levels
- Intuitively, exponent is group number of **highest group where node above topmost level**

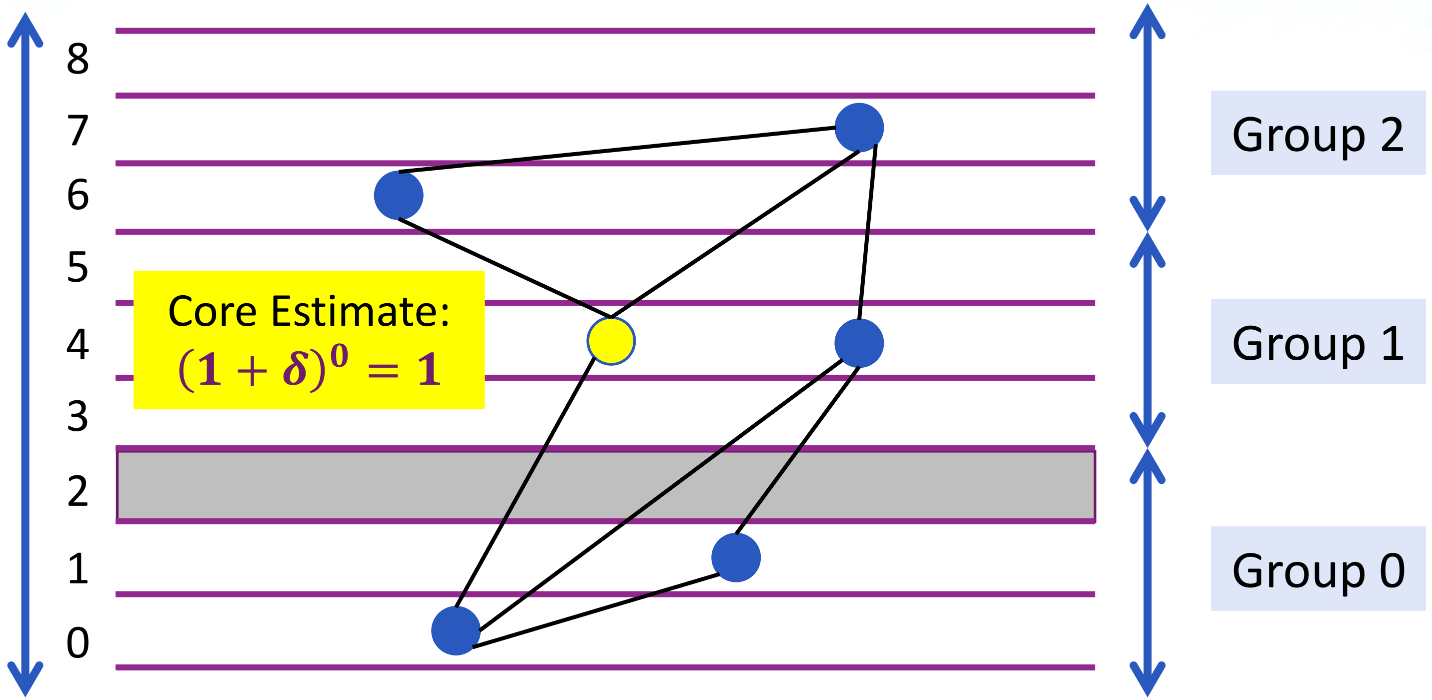
Obtaining the Coreness Estimate



- Set the coreness estimate: $\mathbf{(1+\delta)^{max(\lfloor (level(v)+1)/\,(4\,\lceil \log_{1+\delta} n \rceil)\rfloor - 1, 0)}}$
- Each group has $4\lceil \log_{1+\delta} n \rceil$ levels
- Intuitively, exponent is group number of **highest group where node above topmost level**

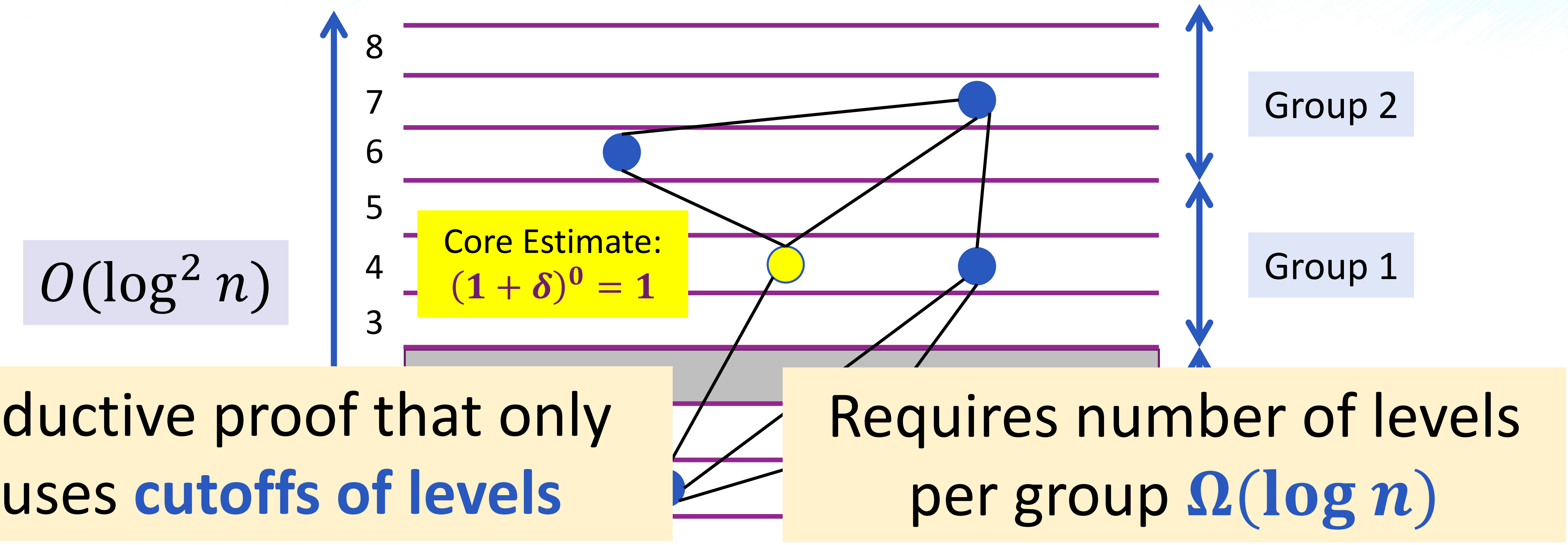
Obtaining the Coreness Estimate

$O(\log^2 n)$



- Set the coreness estimate: $(1+\delta)^{max(\lfloor (level(v)+1)/(4\lceil \log_{1+\delta} n \rceil)\rfloor - 1, 0)}$
- Each group has $4\lceil \log_{1+\delta} n \rceil$ levels
- Intuitively, exponent is group number of **highest group where node above topmost level**

Obtaining the Coreness Estimate



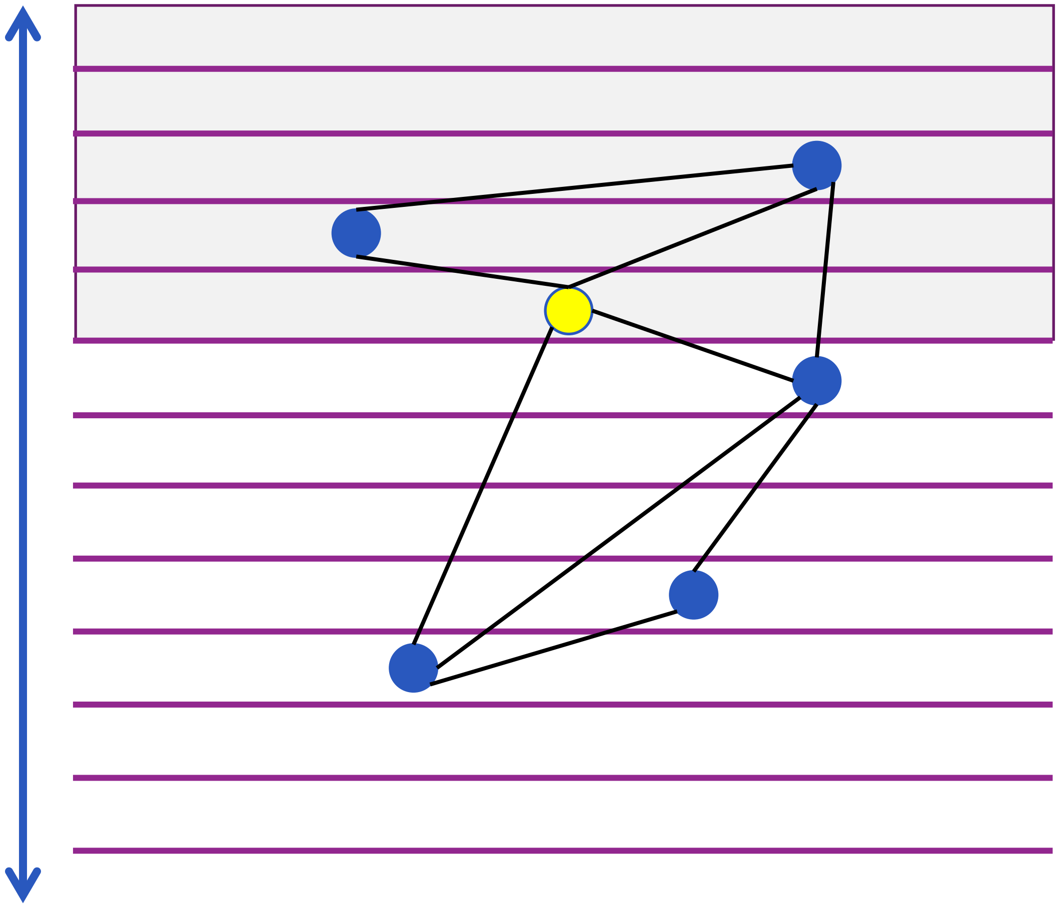
Inductive proof that only uses **cutoffs of levels**

Requires number of levels per group $\mathbf{\Omega(\log n)}$

- Set the coreness estimate: $(1+\delta)^{\max\left(\left\lfloor (level(v)+1)/\left(4\lceil \log_{1+\delta} n \rceil\right)\right\rfloor - 1,\, 0\right)}$
- Each group has $4\lceil \log_{1+\delta} n \rceil$ levels
- Intuitively, exponent is group number of **highest group where node above topmost level**

Proof of Our Approximation Factor: Upper Bound

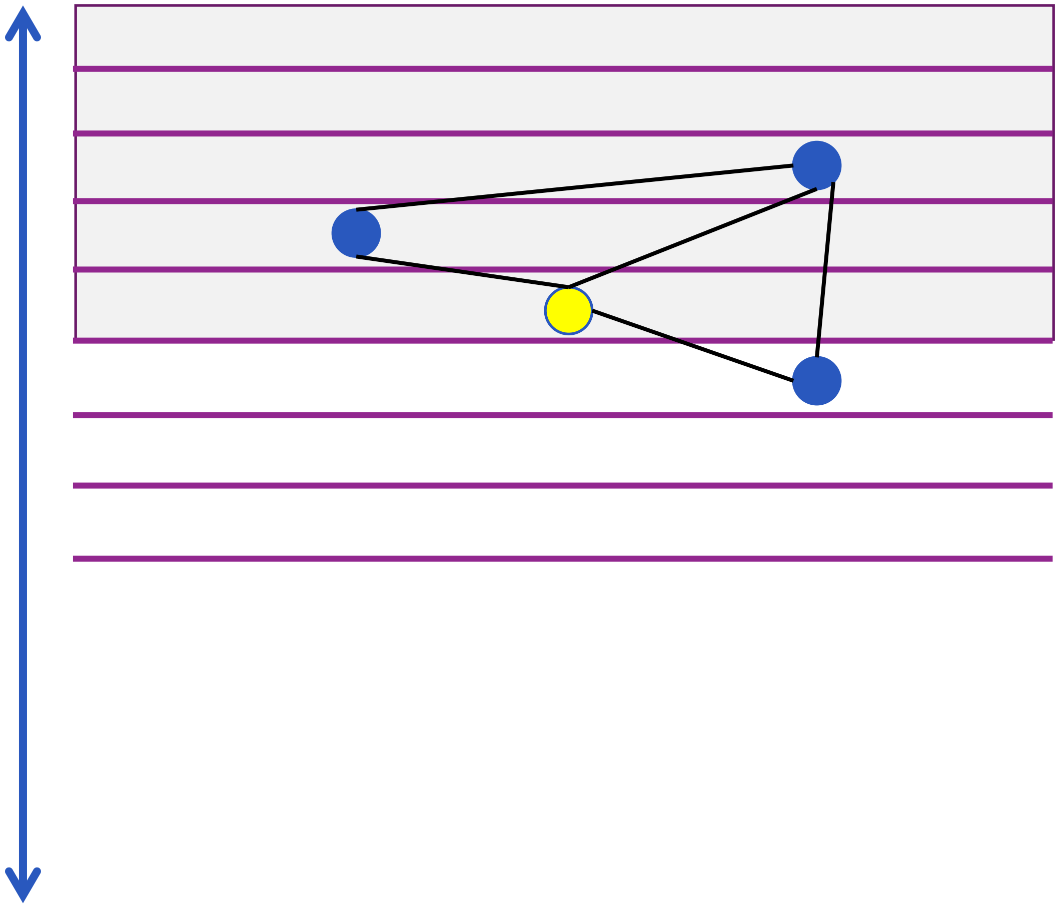
Coreness Estimate:
$(1+\epsilon)^i$



Invariant: $\leq 2.1(1+\epsilon)^i$

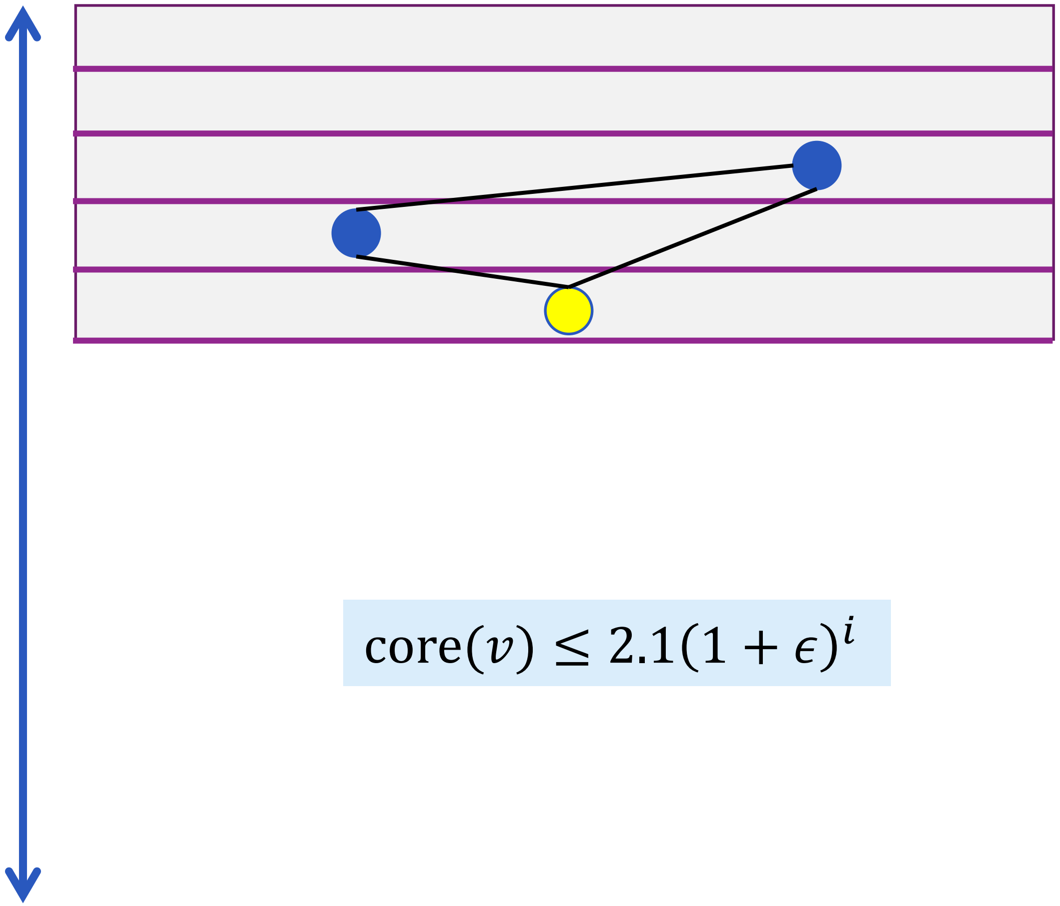
Proof of Our Approximation Factor: Upper Bound

Coreness Estimate:
$(1+\epsilon)^i$



Invariant: $\leq 2.1(1+\epsilon)^i$

Proof of Our Approximation Factor: Upper Bound

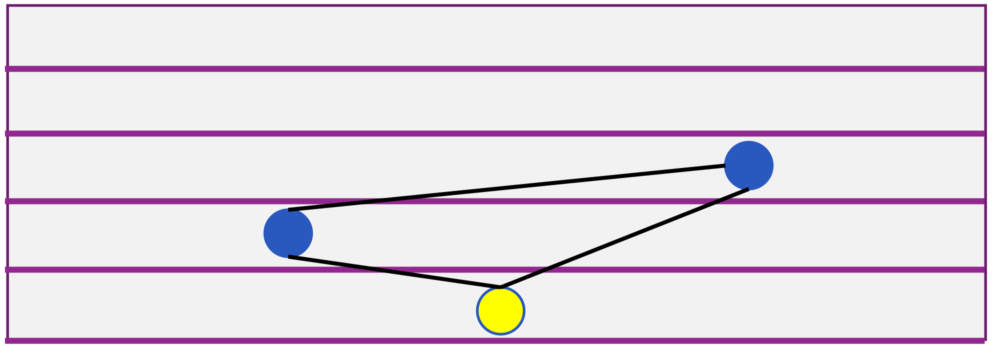


Estimate: $(1+\epsilon)^i$

Invariant: $\leq 2.1(1+\epsilon)^i$

$\text{core}(v) \leq 2.1(1+\epsilon)^i$

Proof of Our Approximation Factor: Upper Bound



Estimate: $(1+\epsilon)^i$

Invariant: $\leq 2.1(1+\epsilon)^i$

$\mathrm{core}(v) \leq 2.1(1+\epsilon)^i$

Key Proof: Lower bound proof only requires lower bound invariant and definition of *k-core.*

Complexity Analysis

- $O(\log^2 n)$ levels

Complexity Analysis

- $O(\log^2 n)$ levels
  - $O(\log\log n)$ depth per level to calculate **desire-levels** using **doubling search**

Complexity Analysis

- $O(\log^2 n)$ levels
  - $O(\log \log n)$ depth per level to calculate **desire-levels** using **doubling search**
  - $O(\log^* n)$ depth with high probability for **hash table operations**

Complexity Analysis

- $O(\log^2 n)$ levels
  - $O(\log \log n)$ depth per level to calculate **desire-levels** using **doubling search**
  - $O(\log^* n)$ depth with high probability for **hash table operations**
- **Total depth: $\boldsymbol{O\left(\log^2 n \log \log n\right)}$**

Complexity Analysis

- $O(\log^2 n)$ levels
  - $O(\log\log n)$ depth per level to calculate **desire-levels** using **doubling search**
  - $O(\log^* n)$ depth with high probability for **hash table operations**
- **Total depth: $O(\log^2 n \log\log n)$**

- **$O(B \log^2 n)$ amortized work** is based on potential argument
  - Vertices and edges store potential based on their levels

Experimental Implementation Details

- Designed an optimized multicore implementation

Experimental Implementation Details

- Designed an optimized multicore implementation
- Used parallel primitives and data structures from the Graph Based Benchmark Suite [Dhulipala et al. ‘20]

Experimental Implementation Details

- Designed an optimized multicore implementation
- Used parallel primitives and data structures from the Graph Based Benchmark Suite [Dhulipala et al. '20]
- Maintain concurrent hash tables for each vertex v
  - One for storing neighbors on levels ≥ level(v)
  - One for storing neighbors on every level i in [0, level(v)-1]

Experimental Implementation Details

- Designed an optimized multicore implementation
- Used parallel primitives and data structures from the Graph Based Benchmark Suite [Dhulipala et al. ‘20]
- Maintain concurrent hash tables for each vertex v
  - One for storing neighbors on levels ≥ level(v)
  - One for storing neighbors on every level i in [0, level(v)-1]
- Moving vertices around in the PLDS requires carefully updating these hash tables for work-efficiency

Tested Graphs

Graphs from Stanford SNAP database, DIMACS Shortest Paths challenge, and Network Repository—including some temporal

Graph	Num. Vertices	Num. Edges	Max *k*
dblp	425,957	2,099,732	101
brain-network	784,262	267,844,669	1200
wikipedia	1,140,149	2,787,967	124
youtube	1,138,499	5,980,886	51
stackoverflow	2,601,977	28,183,518	163
livejournal	4,847,571	85,702,474	329
orkut	3,072,627	234,370,166	253
usa-central	14,081,816	16,933,413	2
usa-road	23,072,627	28,854,312	3
twitter	41,652,231	1,202,513,046	2484
friendster	65,608,366	1,806,067,135	304

Tested Graphs

Graphs from Stanford SNAP database, DIMACS Shortest Paths challenge, and Network Repository—including some temporal

Graph	Num. Vertices	Num. Edges	Max *k*
dblp	425,957	2,099,732	101
brain-network	784,262	267,844,669	1200
wikipedia	1,140,149	2,787,967	124
youtube	1,138,499	5,980,886	51
stackoverflow	2,601,977	28,183,518	163
livejournal	4,847,571	85,702,474	329
orkut	3,072,627	234,370,166	253
usa-central	14,081,816	16,933,413	2
usa-road	23,072,627	28,854,312	3
twitter	41,652,231	1,202,513,046	2484
friendster	65,608,366	1,806,067,135	304

Experiments

- c2-standard-60 Google Cloud instances
  - 30 cores with two-way hyper-threading
  - 236 GB memory
- m1-megamem-96 Google Cloud instances
  - 48 cores with two-way hyperthreading
  - 1433.6 GB memory
- Timeout: 3 hours
- 3 different types of batches:

Experiments

- c2-standard-60 Google Cloud instances
  - 30 cores with two-way hyper-threading
  - 236 GB memory
- m1-megamem-96 Google Cloud instances
  - 48 cores with two-way hyperthreading
  - 1433.6 GB memory
- Timeout: 3 hours
- 3 different types of batches:
  - All Batched Insertions
  - All Batched Deletions
  - Mixed Batches of Both Insertions and Deletions

Experiments

- c2-standard-60 Google Cloud instances
  - 30 cores with two-way hyper-threading
  - 236 GB memory
- m1-megamem-96 Google Cloud instances
  - 48 cores with two-way hyperthreading
  - 1433.6 GB memory
- Timeout: 3 hours
- 3 different types of batches:
  - All Batched Insertions
  - All Batched Deletions
  - Mixed Batches of Both Insertions and Deletions

Improvements across **all** experiments!

Runtimes/Accuracy Against State-of-the-Art Algorithms

Benchmarks

- **Sun et al. TKDD**: **sequential, approx., dynamic** algorithm
- **LDS**: **sequential, approx., dynamic** LDS of Henzinger et al.
- **Zhang and Yu SIGMOD**: **sequential, exact, dynamic** algorithm
- **Hua et al. TPDS: parallel, exact, dynamic** algorithm

Versions of PLDS

- **PLDS: exact theoretical** algorithm
- **PLDSOpt: code-optimized** PLDS

Runtimes/Accuracy Against State-of-the-Art Algorithms

Benchmarks

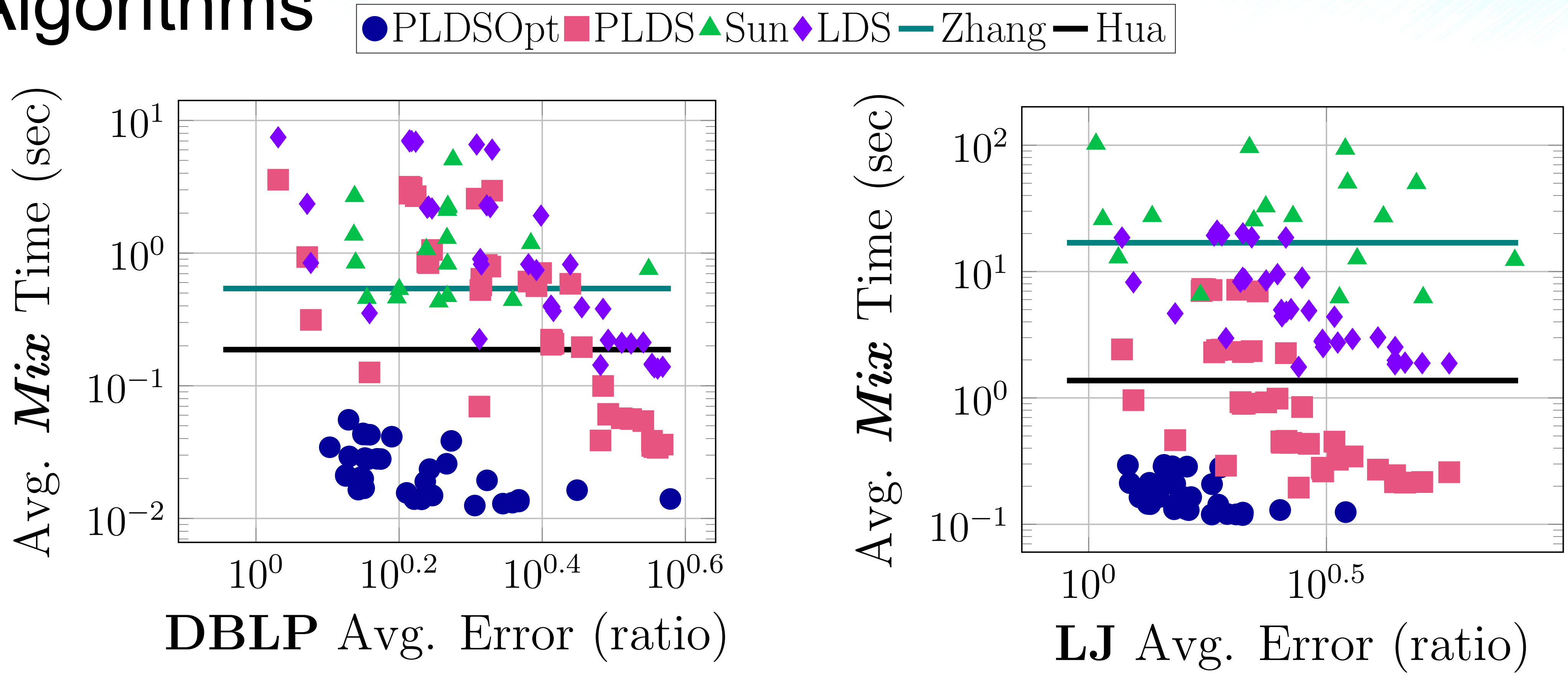
- **Sun et al. TKDD**: **sequential, approx., dynamic** algorithm
- **LDS**: **sequential, approx., dynamic** LDS of Henzinger et al.
- **Zhang and Yu SIGMOD**: **sequential, exact, dynamic** algorithm
- **Hua et al. TPDS: parallel, exact, dynamic** algorithm

Versions of PLDS

- **PLDS: exact theoretical** algorithm
- **PLDSOpt: code-optimized** PLDS

Key Optimization Feature:
Reduce number of levels per group

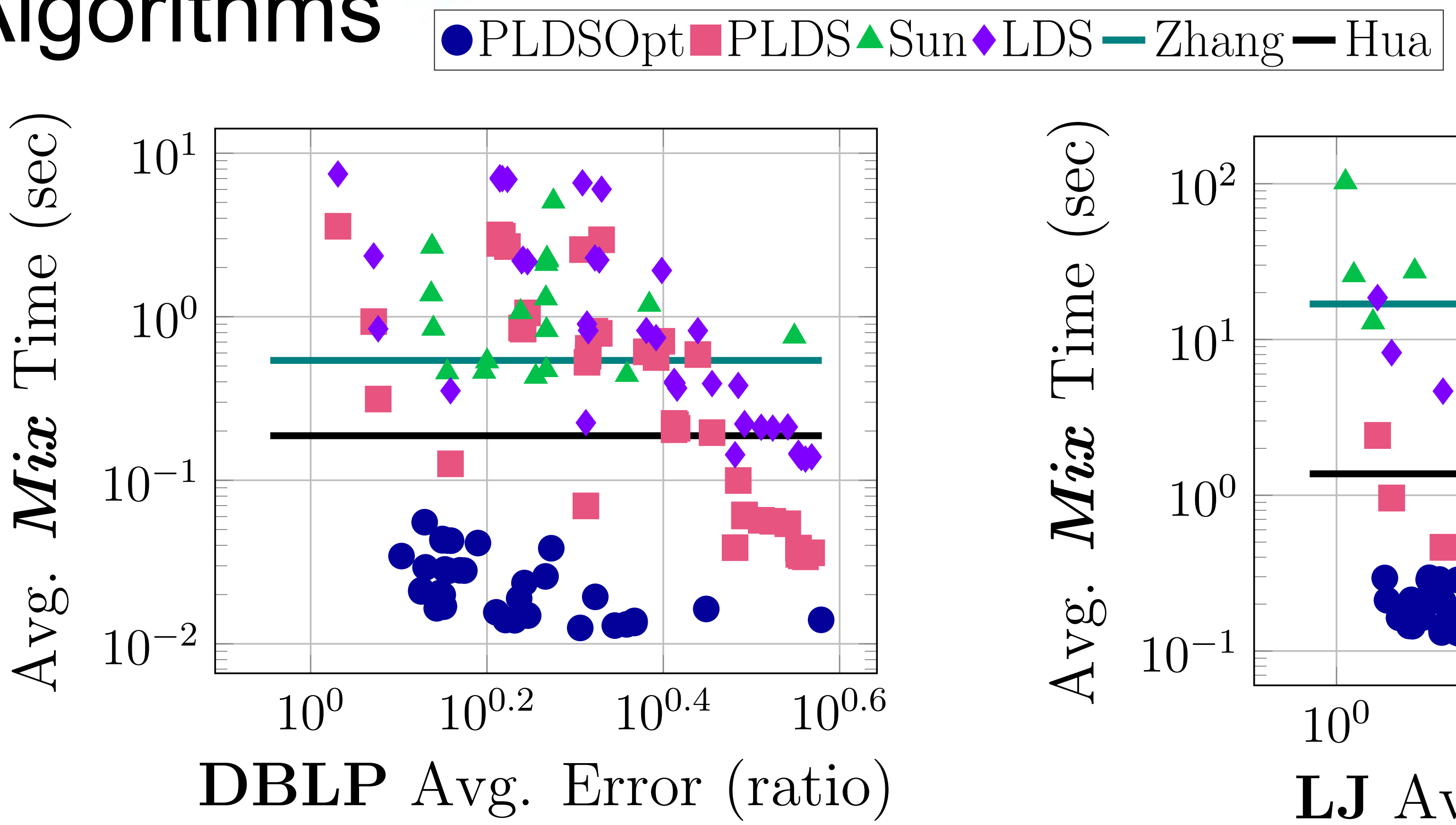
Runtimes/Accuracy Against State-of-the-Art Algorithms

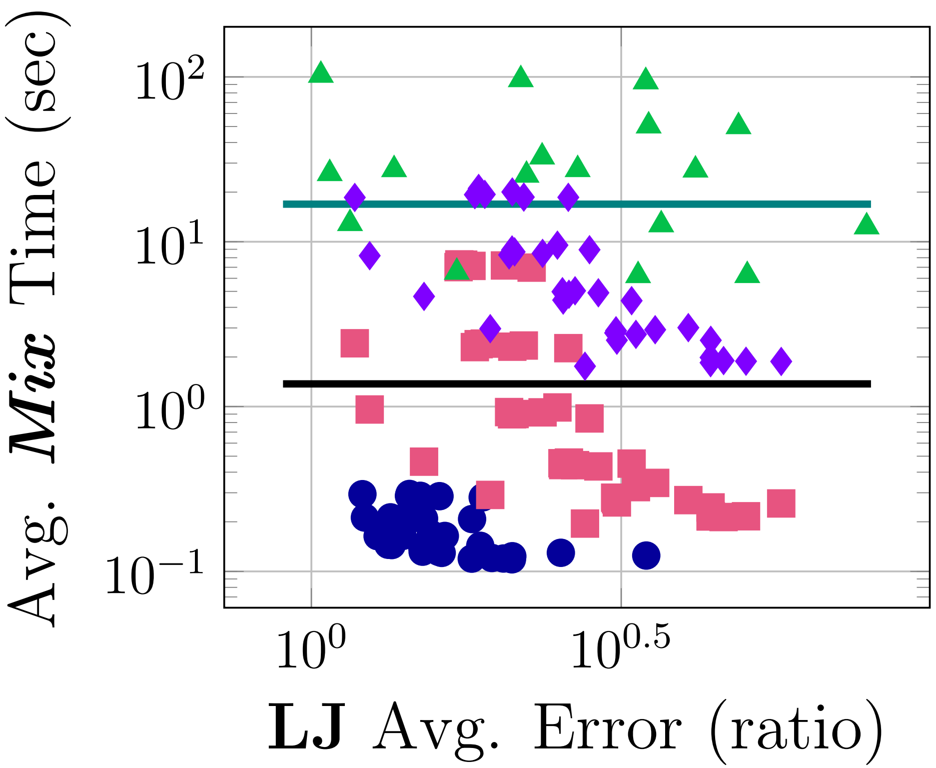


DBLP: 425K vertices, 2.1M edges

LJ (LiveJournal): 4.8M vertices, 85.7M edges

Runtimes/Accuracy Against State-of-the-Art Algorithms





PLDSOpt: 19.04–544.22x
speedup over Sun

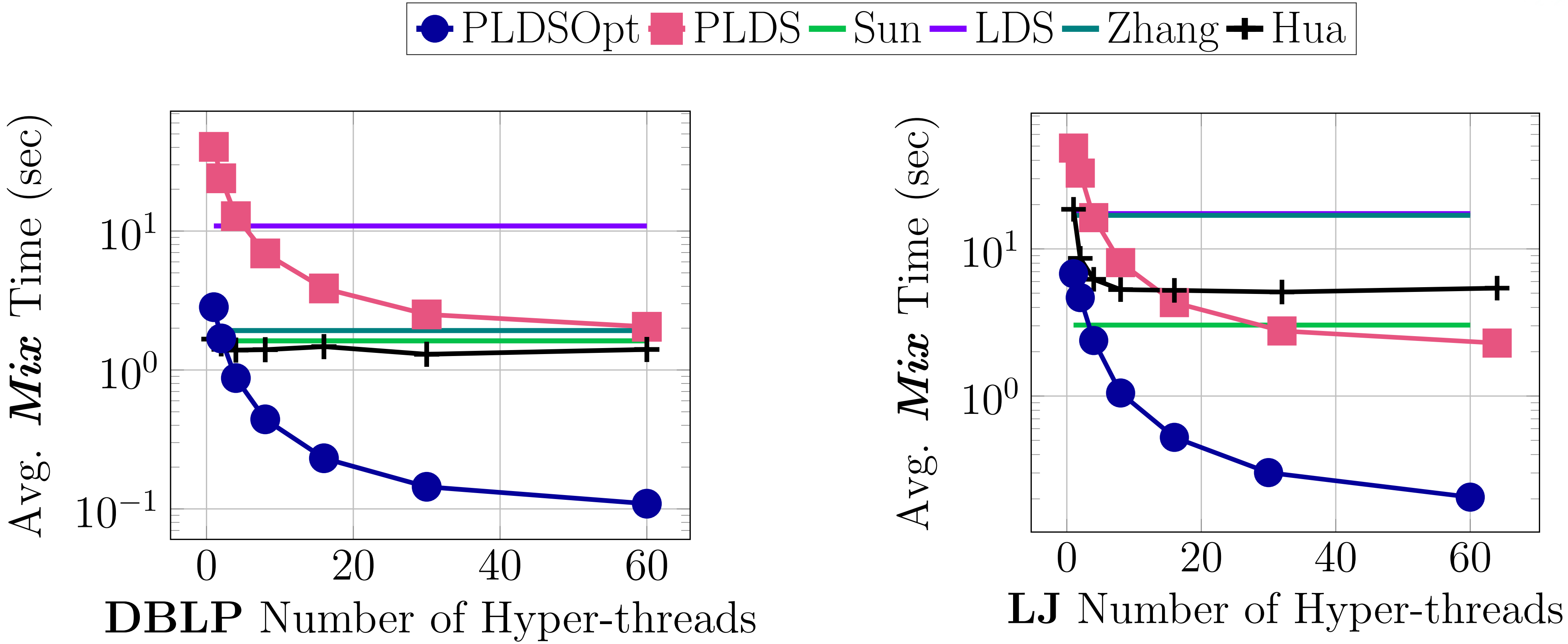
LJ (LiveJ

PLDSOpt: 2.49–24.41x
speedup over Hua

edges

Number of Hyper-Threads

Faster than all other algorithms at 4 cores!



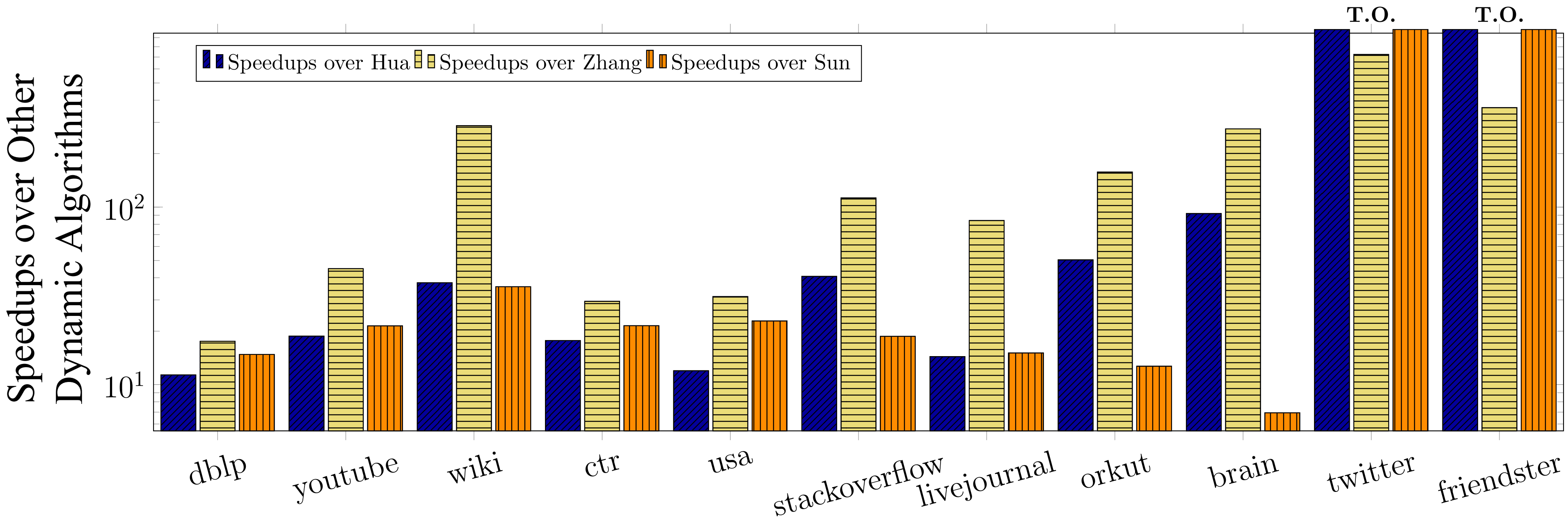
PLDSOpt: 33.02x
self-relative speedup

PLDS: 26.46x
self-relative speedup

Hua: 3.6x
self-relative speedup

Speedups On a Variety of Graphs

- Speedups against dynamic benchmarks: Hua, Zhang, and Sun



Batch size = 10^6

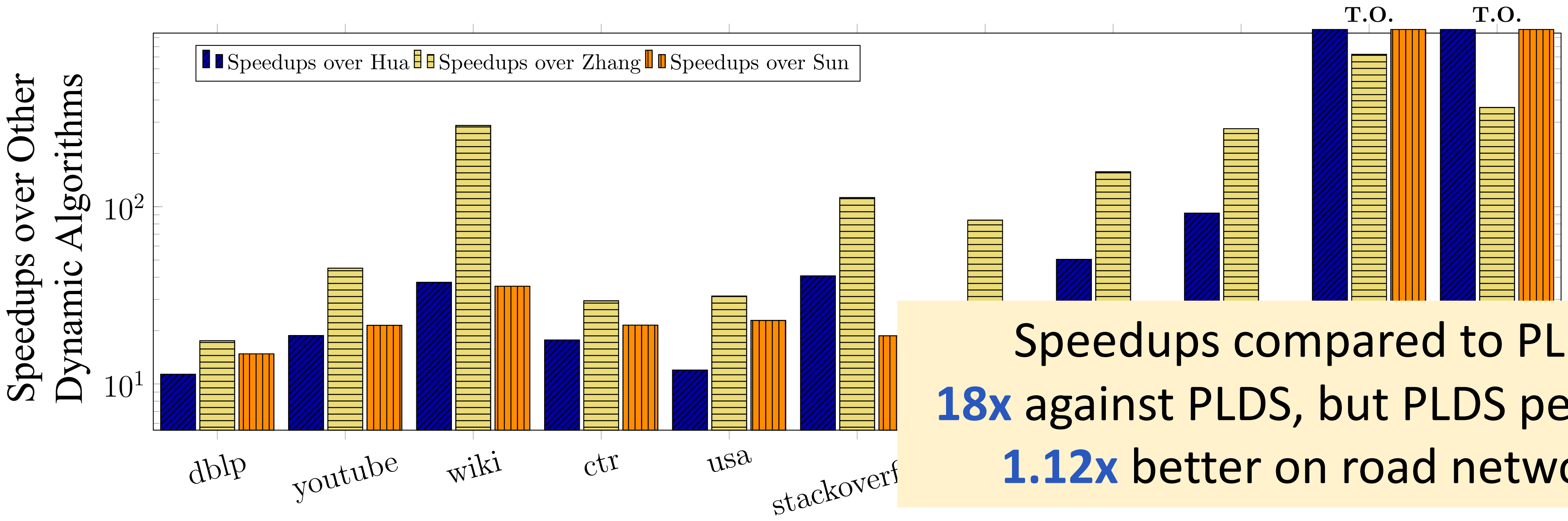
Graphs ordered by size (left to right)

Speedups on **all graphs** against **all benchmarks**

Speedups up to: **91.95x** for Hua, **35.59x** for Sun, **723.72x** for Zhang

Speedups On a Variety of Graphs

- Speedups against dynamic benchmarks: Hua, Zhang, and Sun



Speedups compared to PLDS:
18x against PLDS, but PLDS performs **1.12x** better on road networks

Batch size = 10^6

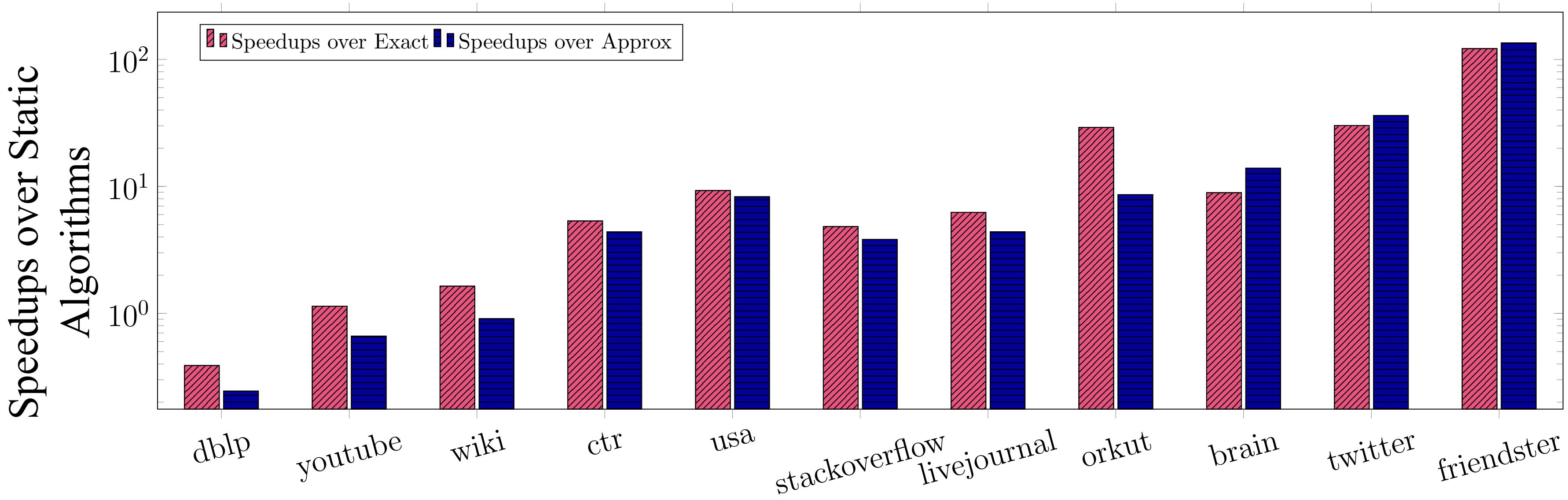
Graphs ordered by size (left to right)

Speedups on **all graphs** against **all benchmarks**

Speedups up to: **91.95x** for Hua, **35.59x** for Sun, **723.72x** for Zhang

Speedups Against Parallel Static Algorithms

- Parallel exact *k*-core decomposition [Dhulipala et al. ‘18]
- Parallel $(2+\epsilon)$-approximate *k*-core decomposition

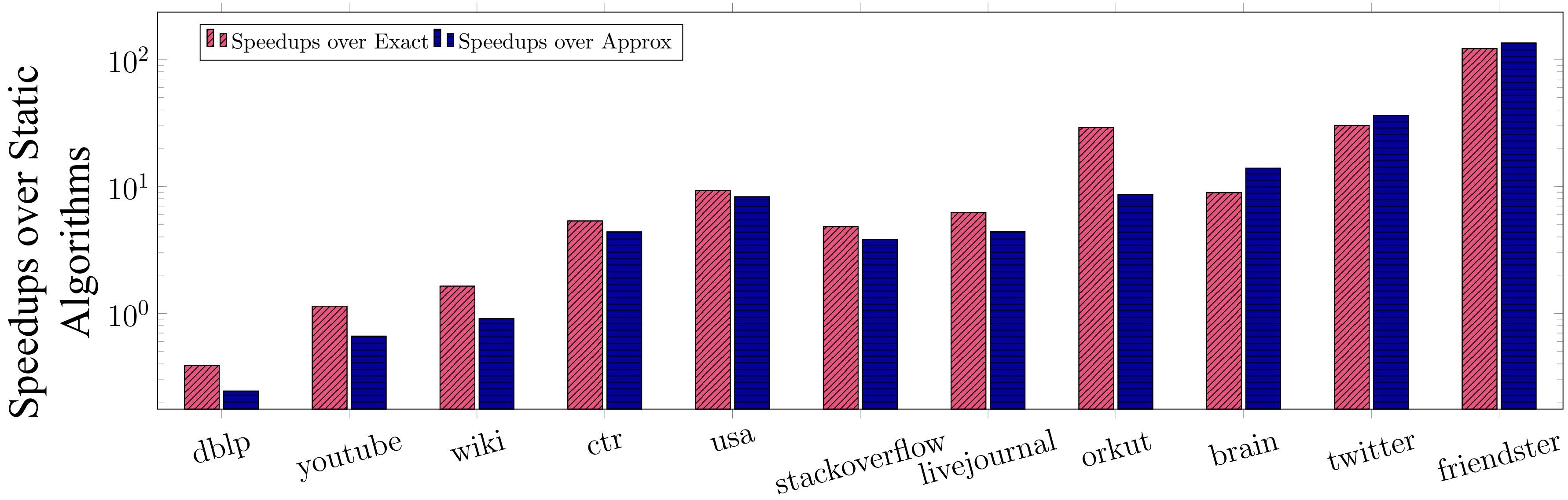


Batch size = 10^6

Graphs ordered by size (left to right)

Speedups Against Parallel Static Algorithms

- Parallel exact *k*-core decomposition [Dhulipala et al. '18]
- Parallel $(2+\epsilon)$-approximate *k*-core decomposition



Batch size = 10^6

Graphs ordered by size (left to right)

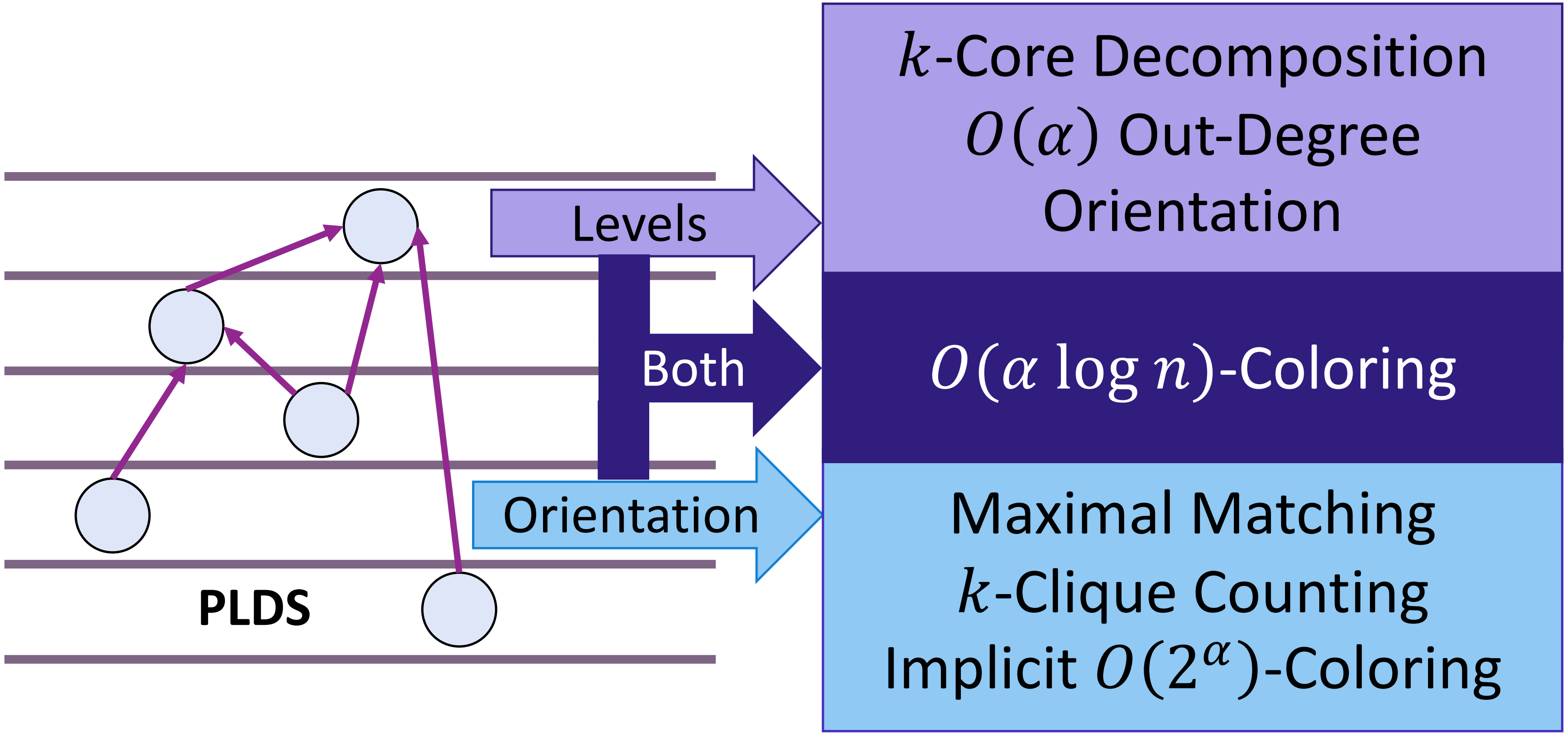
We achieve speedups for all but the smallest graphs

Speedups of up to **122x** for Twitter (1.2B edges) and Friendster (1.8B edges)

Other Results

Problem	Approx	Work	Depth
Static k-Core	$2+\varepsilon$	$O(m+n)$	$\tilde{O}(\log^2 n)$
Low Out-Degree	$4+\varepsilon$	$O(\lvert B\rvert \log^2 n)$	$\tilde{O}(\log^3 n)$
Maximal Matching	Maximal	$O(\lvert B\rvert\,(k+\log^2 n))$	$\tilde{O}(\log\Delta \log^2 n)$
Clique Counting	Exact	$O(\lvert B\rvert\,(k^{c-2}+\log^2 n))$	$\tilde{O}(\log^2 n)$
Explicit Coloring	$O(k \log n)$	$O(\lvert B\rvert \log^2 n)$	$\tilde{O}(\log^2 n)$
Implicit Coloring	$O(2^k)$	$O(\lvert B\rvert \log^3 n)$	$\tilde{O}(\log^2 n)$

PLDS to Other Results



Other Results + Future Work Implementations!

Problem	Approx	Work	Depth
Static k-Core	$2+\varepsilon$	$O(m+n)$	$\tilde{O}(\log^2 n)$
Low Out-Degree	$4+\varepsilon$	$O(\lvert B\rvert \log^2 n)$	$\tilde{O}(\log^3 n)$
Maximal Matching	Maximal	$O(\lvert B\rvert\,(k+\log^2 n))$	$\tilde{O}(\log\Delta \log^2 n)$
Clique Counting	Exact	$O(\lvert B\rvert\,(k^{c-2}+\log^2 n))$	$\tilde{O}(\log^2 n)$
Explicit Coloring	$O(k \log n)$	$O(\lvert B\rvert \log^2 n)$	$\tilde{O}(\log^2 n)$
Implicit Coloring	$O(2^k)$	$O(\lvert B\rvert \log^3 n)$	$\tilde{O}(\log^2 n)$

Conclusion

- New parallel level data structure (PLDS)
- Parallel batch-dynamic algorithms for *k*-core decomposition and related problems (low out-degree orientation, maximal matching, clique counting, graph coloring)
- Our *k*-core algorithm achieves significant improvements over state-of-the-art solutions in practice
- Source code available at https://github.com/qqliu/batch-dynamic-kcore-decomposition

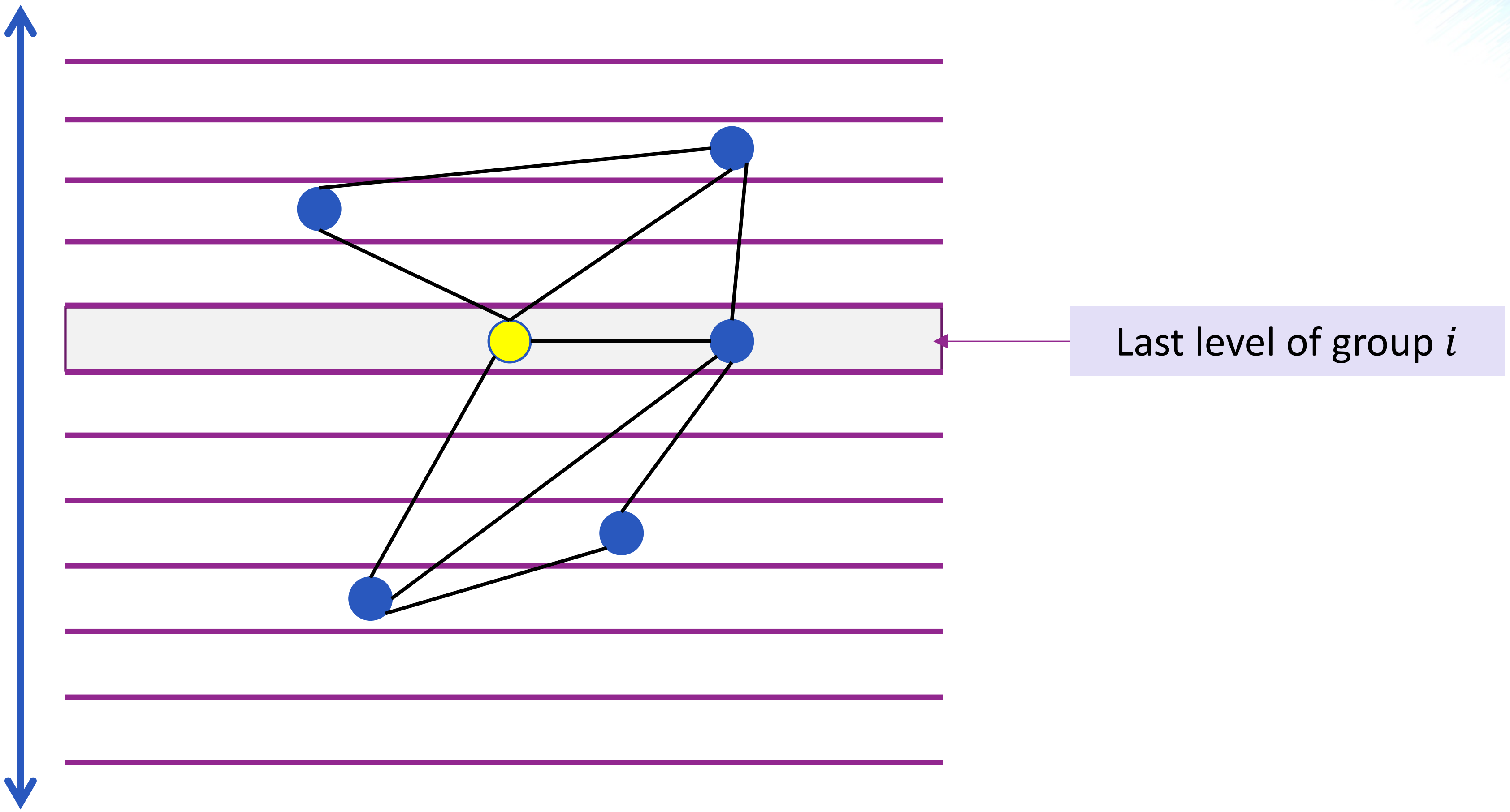
Extra Slides

Proof of Our Approximation Factor: Lower Bound

Estimate: $(1+\epsilon)^i$

Assume for Contradiction:

$$c(v) < \frac{(1+\epsilon)^i}{2.1(1+\epsilon)}$$

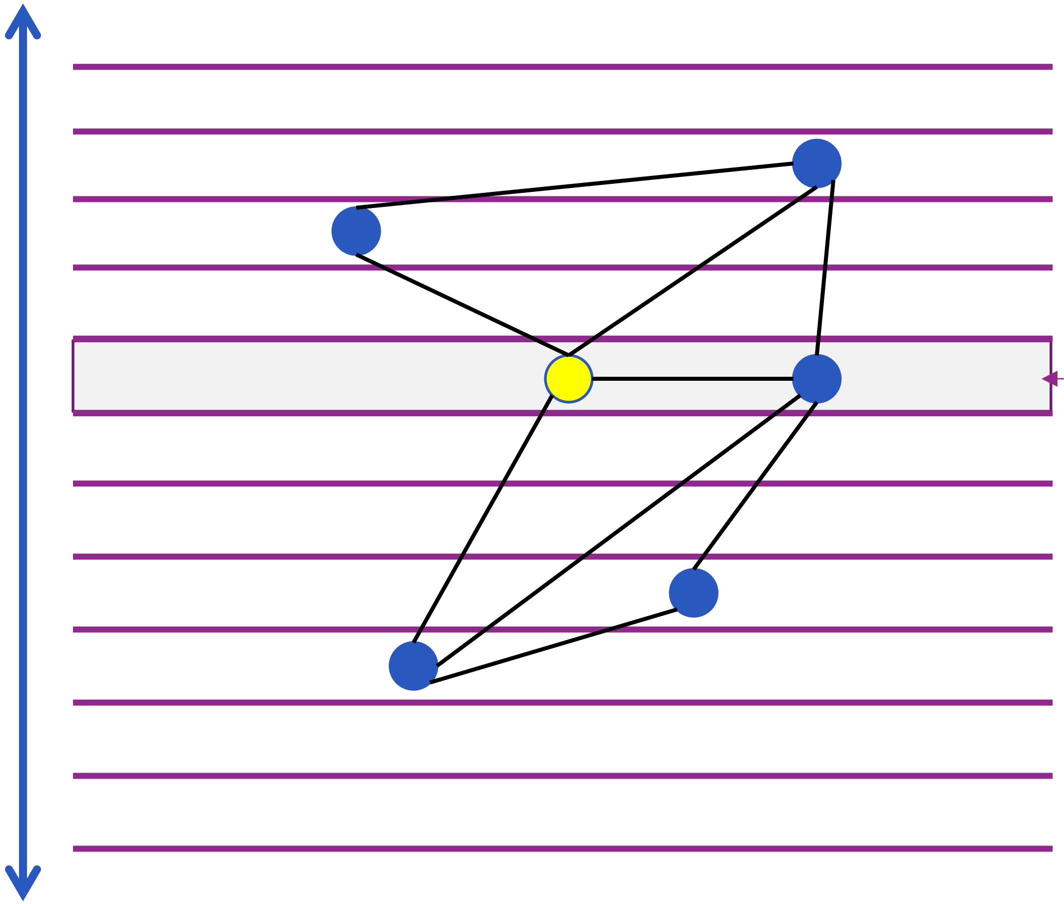


Proof of Our Approximation Factor: Lower Bound

Estimate: $(1+\epsilon)^i$

Assume for Contradiction:

$$c(v) < \frac{(1+\epsilon)^i}{2.5}$$



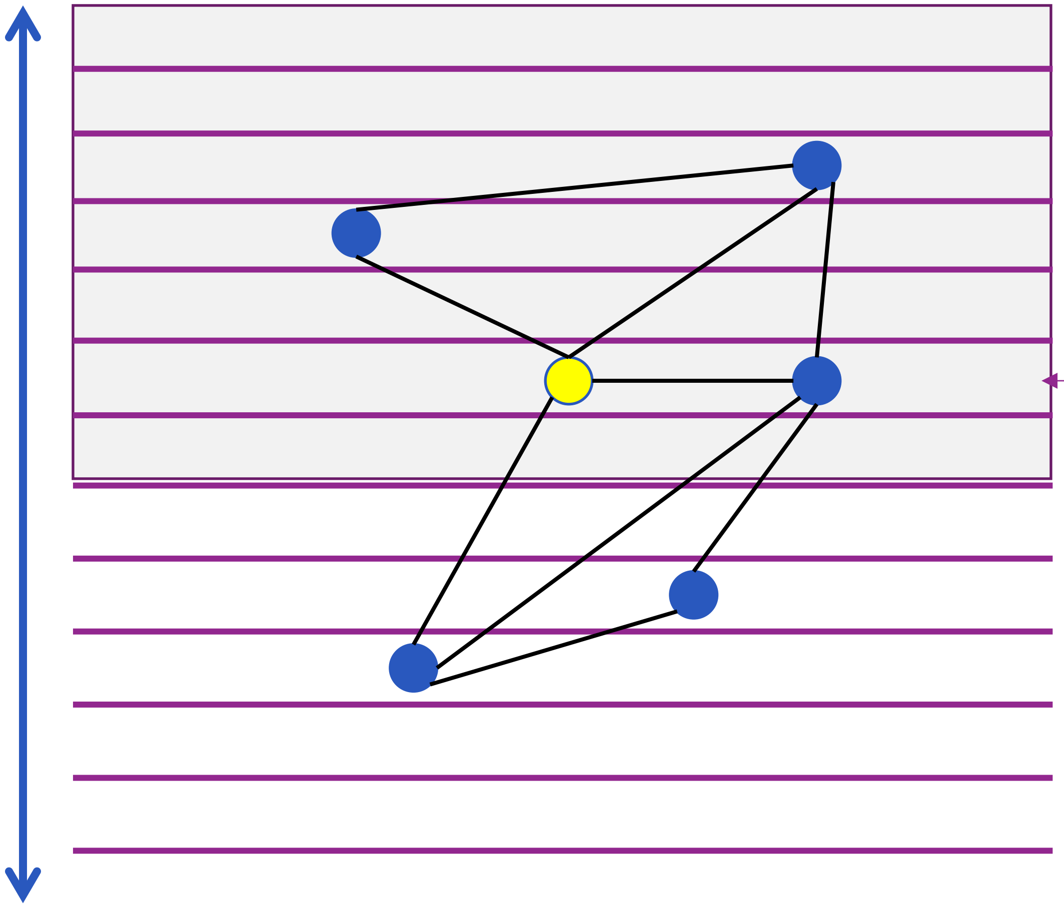
Last level of group i

Proof of Our Approximation Factor: Lower Bound

Estimate: $(1+\epsilon)^i$

Assume for Contradiction:

$$c(v) < \frac{(1+\epsilon)^i}{2.5}$$



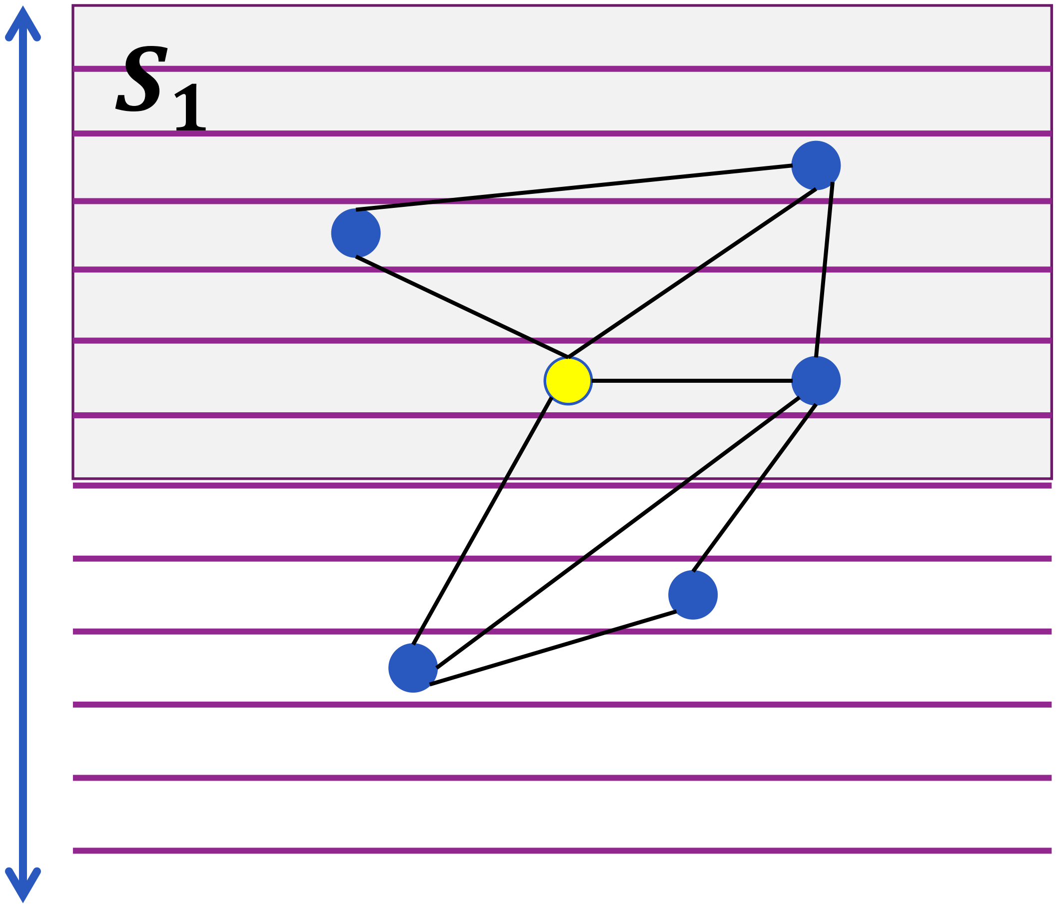
Last level of group i

nodes at or above level below v is: $\geq (1+\epsilon)^i$

Proof of Our Approximation Factor: Lower Bound

Assume for Contradiction:

$$c(v) < \frac{(1+\epsilon)^i}{2.5}$$



Pruning Procedure

Remove all w where

$$d_{S_1}(w) < \frac{(1+\epsilon)^i}{2.5}$$

nodes at or above level of v is: $\geq (1+\epsilon)^i$

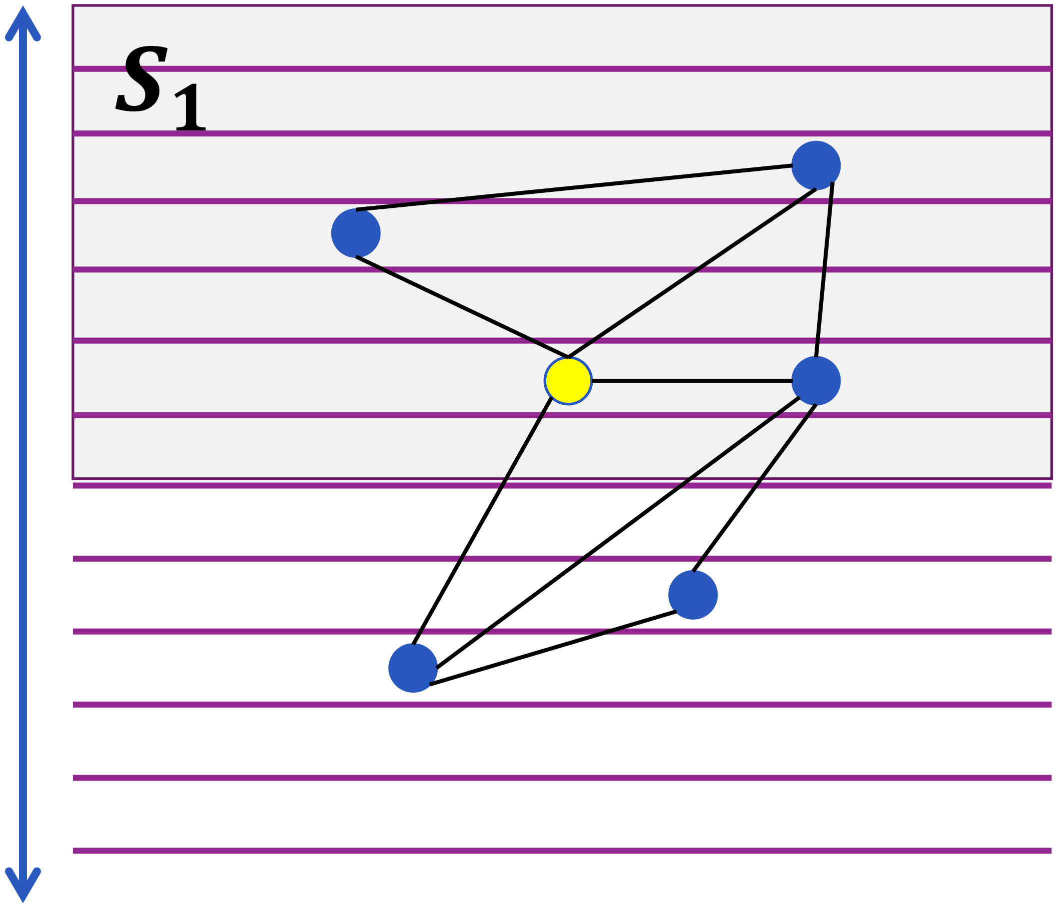
Proof of Our Approximation Factor: Lower Bound

At least $(1+\epsilon)^i - \frac{(1+\epsilon)^i}{2.5}$

edges must be pruned

Assume for Contradiction:

$$c(v) < \frac{(1+\epsilon)^i}{2.5}$$



Pruning Procedure

Remove all w where

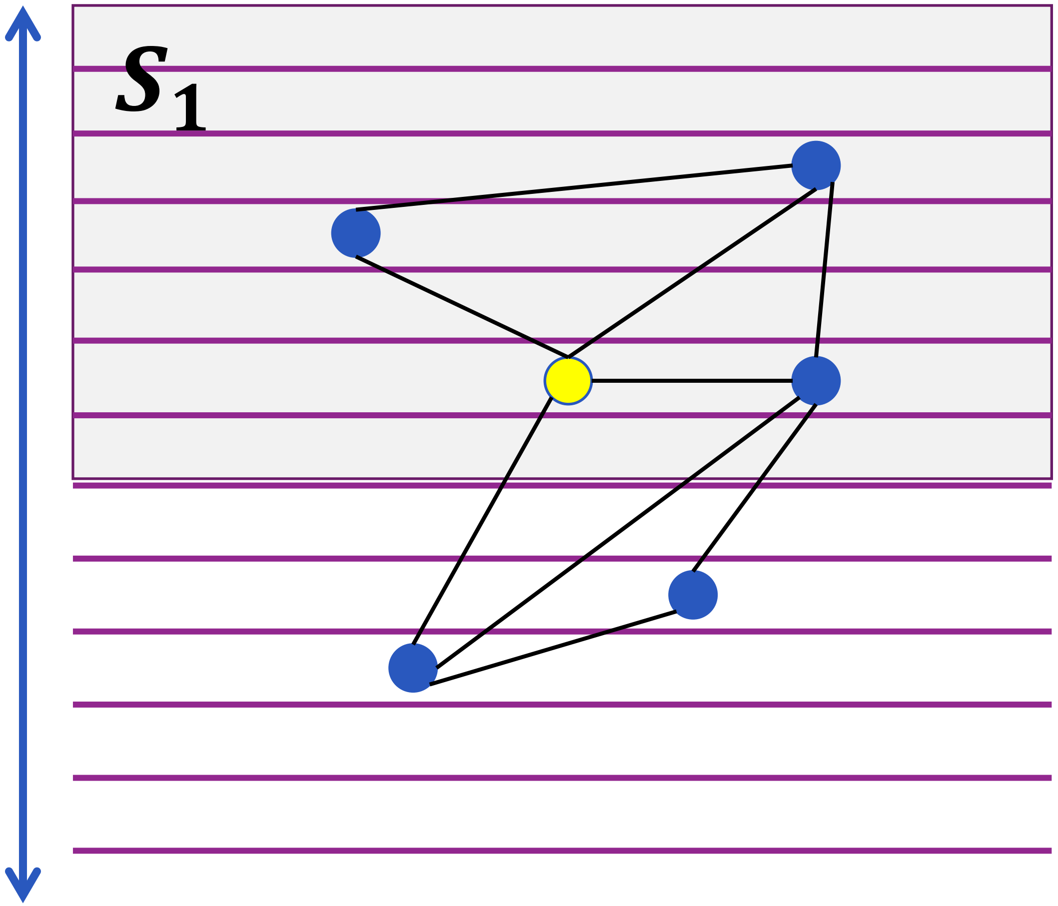
$$d_{S_1}(w) < \frac{(1+\epsilon)^i}{2.5}$$

\# nodes at or above level of v is: $\geq (1+\epsilon)^i$

Proof of Our Approximation Factor: Lower Bound

At least $\frac{(1+\epsilon)^i}{2}$
edges must be pruned

Assume for Contradiction:
$$c(v) < \frac{(1+\epsilon)^i}{2.5}$$



Pruning Procedure
Remove all w where
$$d_{S_1}(w) < \frac{(1+\epsilon)^i}{2.5}$$

nodes at or above level of v is: $\geq (1+\epsilon)^i$

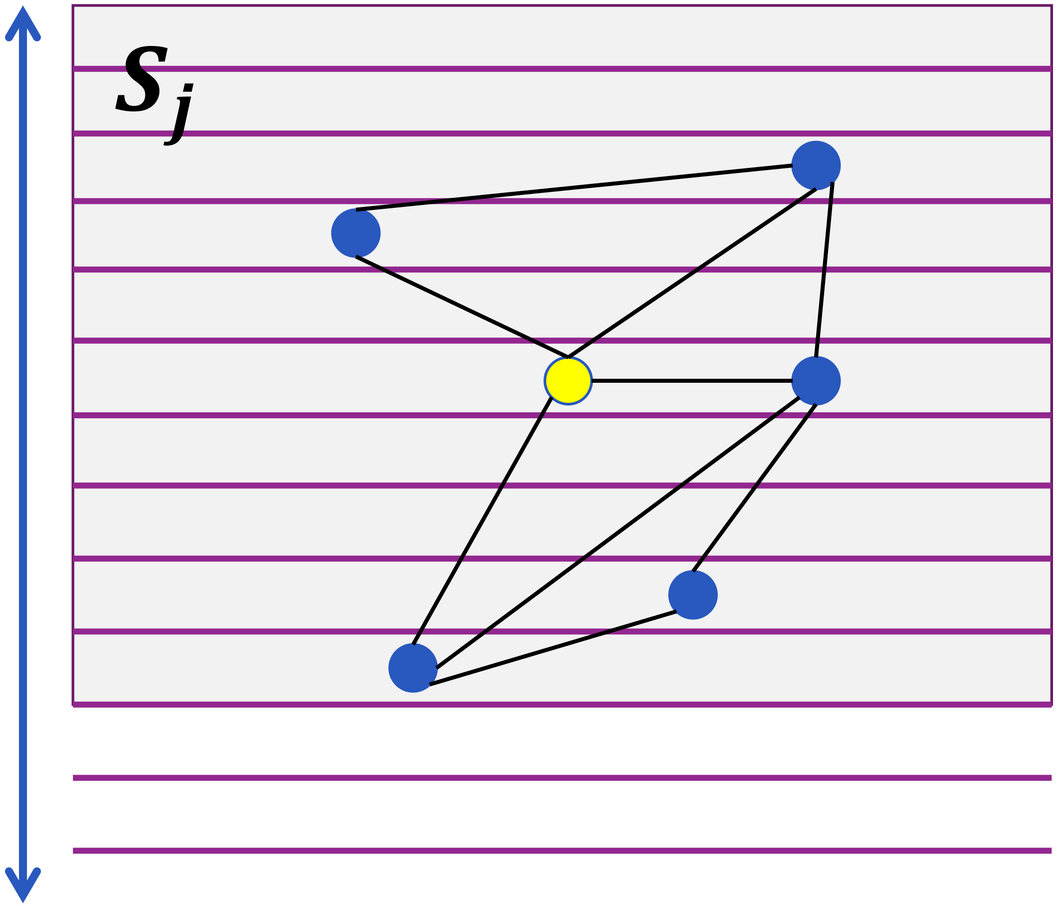
Proof of Our Approximation Factor: Lower Bound

By Induction:

At least $\left(\frac{(1+\epsilon)^i}{2}\right)^j$ edges must be pruned

Assume for Contradiction:

$$c(v) < \frac{(1+\epsilon)^i}{2.5}$$



Pruning Procedure

Remove all w where

$$d_{S_j}(w) < \frac{(1+\epsilon)^i}{2.5}$$

nodes at or above level of v is: $\geq (1+\epsilon)^i$

Proof of Our Approximation Factor: Lower Bound

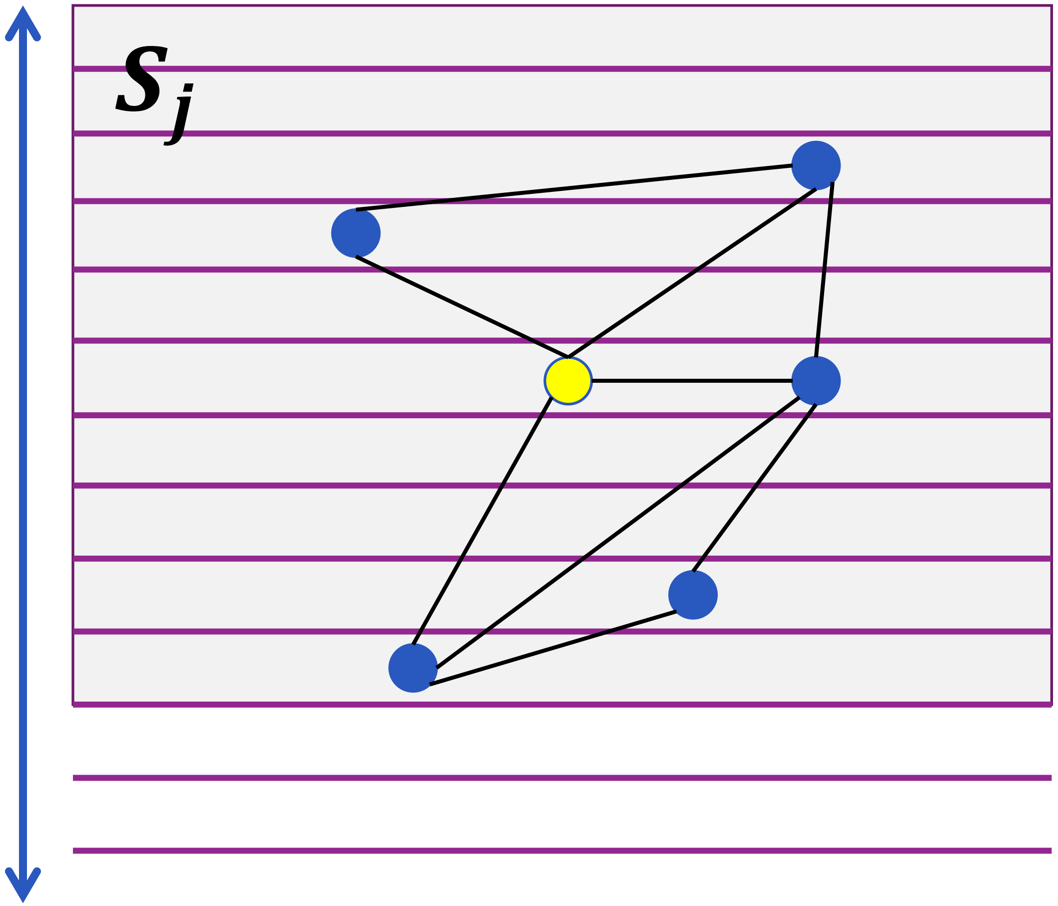
By Induction:

At least $\left(\frac{(1+\epsilon)^i}{2}\right)^j$ edges must be pruned

At least $\frac{\left(\frac{(1+\epsilon)^i}{2}\right)^j}{\frac{(1+\epsilon)^i}{2.5}}$ **nodes** must be pruned

Assume for Contradiction:

$$c(v) < \frac{(1+\epsilon)^i}{2.5}$$



Pruning Procedure

Remove all w where

$$d_{S_j}(w) < \frac{(1+\epsilon)^i}{2.5}$$

nodes at or above level of v is: $\geq (1+\epsilon)^i$

Proof of Our Approximation Factor: Lower Bound

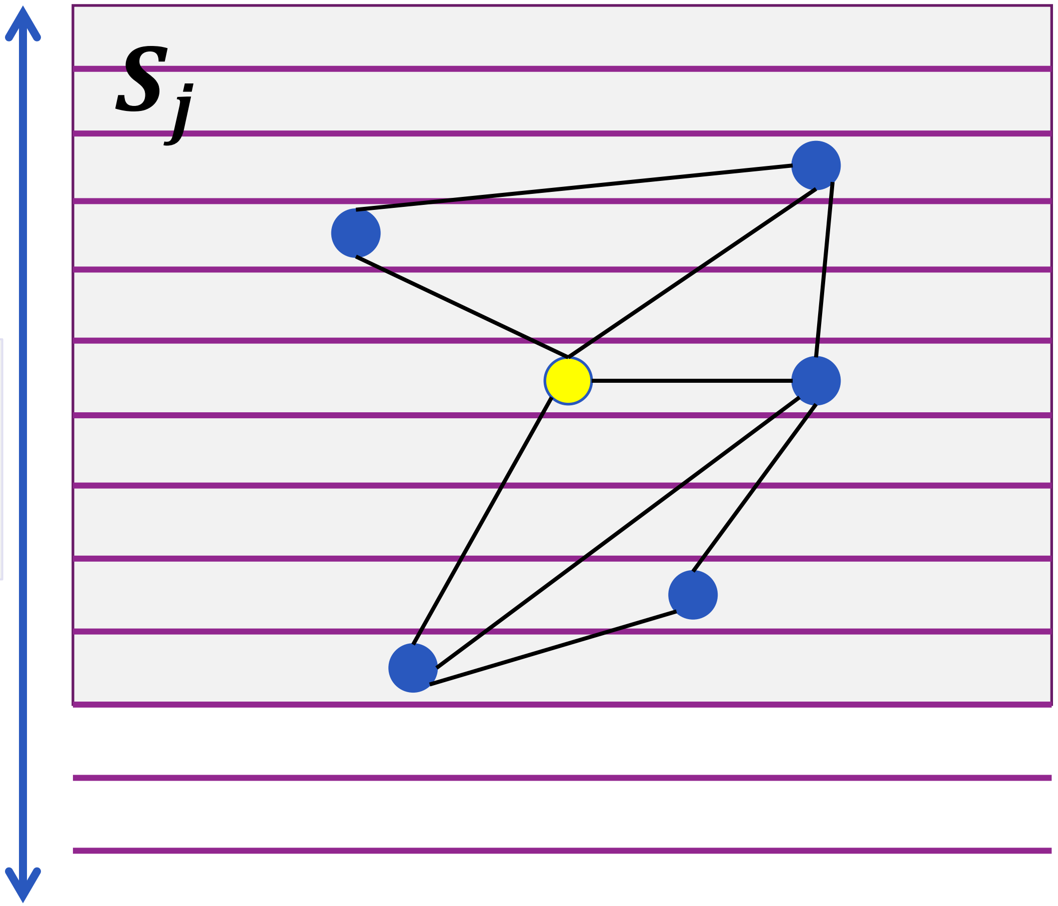
By Induction:

At least $\left(\frac{(1+\epsilon)^i}{2}\right)^j$ edges must be pruned

At least $\left(\frac{(1+\epsilon)^i}{2}\right)^{j-1}$ **nodes** must be pruned

Assume for Contradiction:

$$c(v) < \frac{(1+\epsilon)^i}{2.5}$$



Pruning Procedure

Remove all w where

$$d_{S_j}(w) < \frac{(1+\epsilon)^i}{2.5}$$

\# nodes at or above level of v is: $\geq (1+\epsilon)^i$

Proof of Our Approximation Factor: Lower Bound

By Induction:

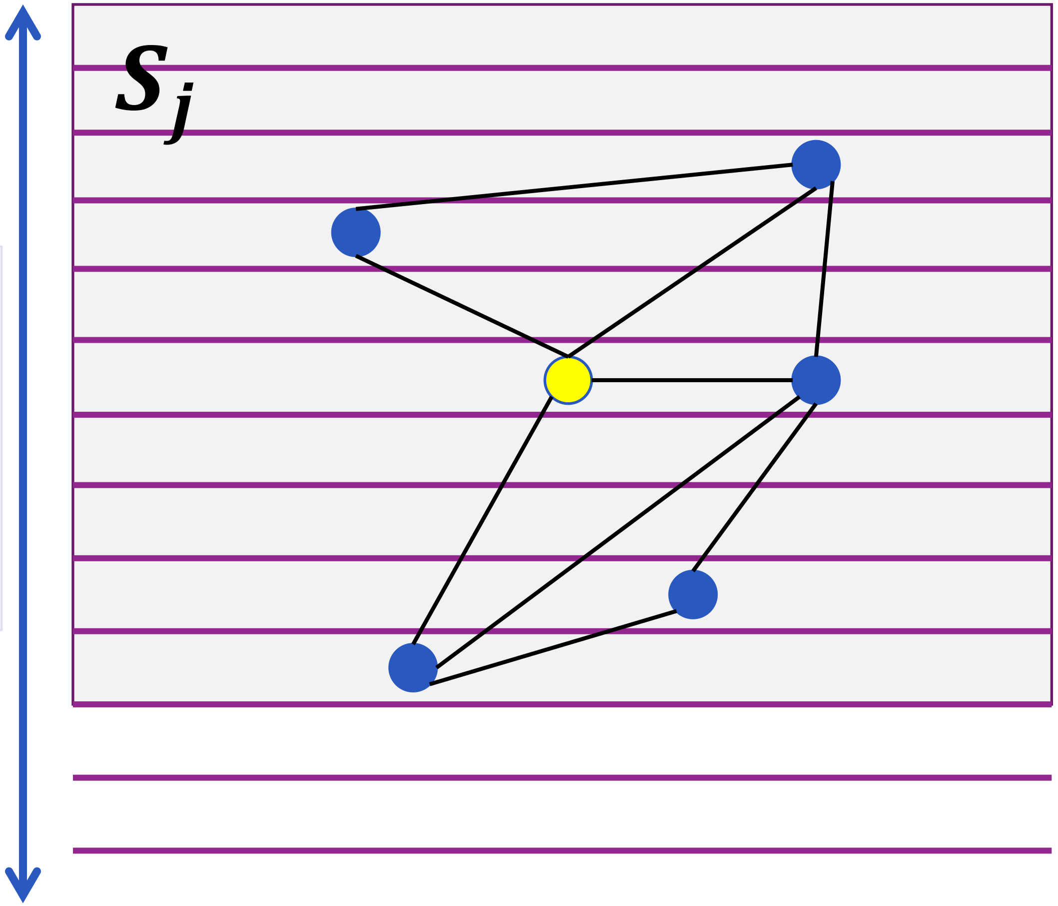
At least $\left(\frac{(1+\epsilon)^i}{2}\right)^j$ edges must be pruned

$$\left(\frac{(1+\epsilon)^i}{2}\right)^{j-1} \leq n$$

$$j \leq \log_{(1+\epsilon)^i/2}(n)$$

Assume for Contradiction:

$$c(v) < \frac{(1+\epsilon)^i}{2.5}$$



Pruning Procedure

Remove all w where

$$d_{S_j}(w) < \frac{(1+\epsilon)^i}{2.5}$$

nodes at or above level of v is: $\geq (1+\epsilon)^i$

Proof of Our Approximation Factor: Lower Bound

By Induction:

At least $\left(\frac{(1+\epsilon)^i}{2}\right)^j$

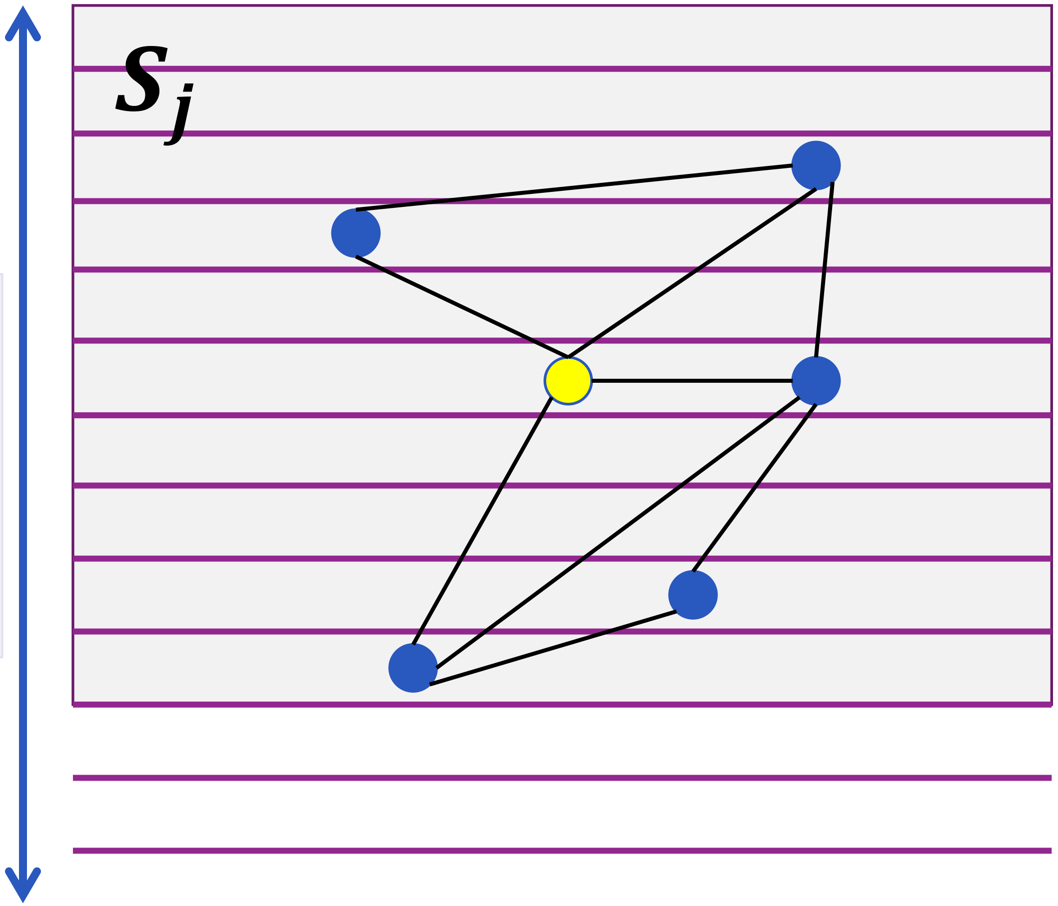
edges must be pruned

$$\left(\frac{(1+\epsilon)^i}{2}\right)^{j-1} \leq n$$

$$j \leq \log_{(1+\epsilon)^i/2}(n)$$

Assume for Contradiction:

$$c(v) < \frac{(1+\epsilon)^i}{2.5}$$



Pruning Procedure

Remove all w where

$$d_{S_j}(w) < \frac{(1+\epsilon)^i}{2.5}$$

nodes at or above level of v is: $\geq (1+\epsilon)^i$

Run out of vertices before first level of the group.

Proof of Our Approximation Factor: Lower Bound

By Induction:

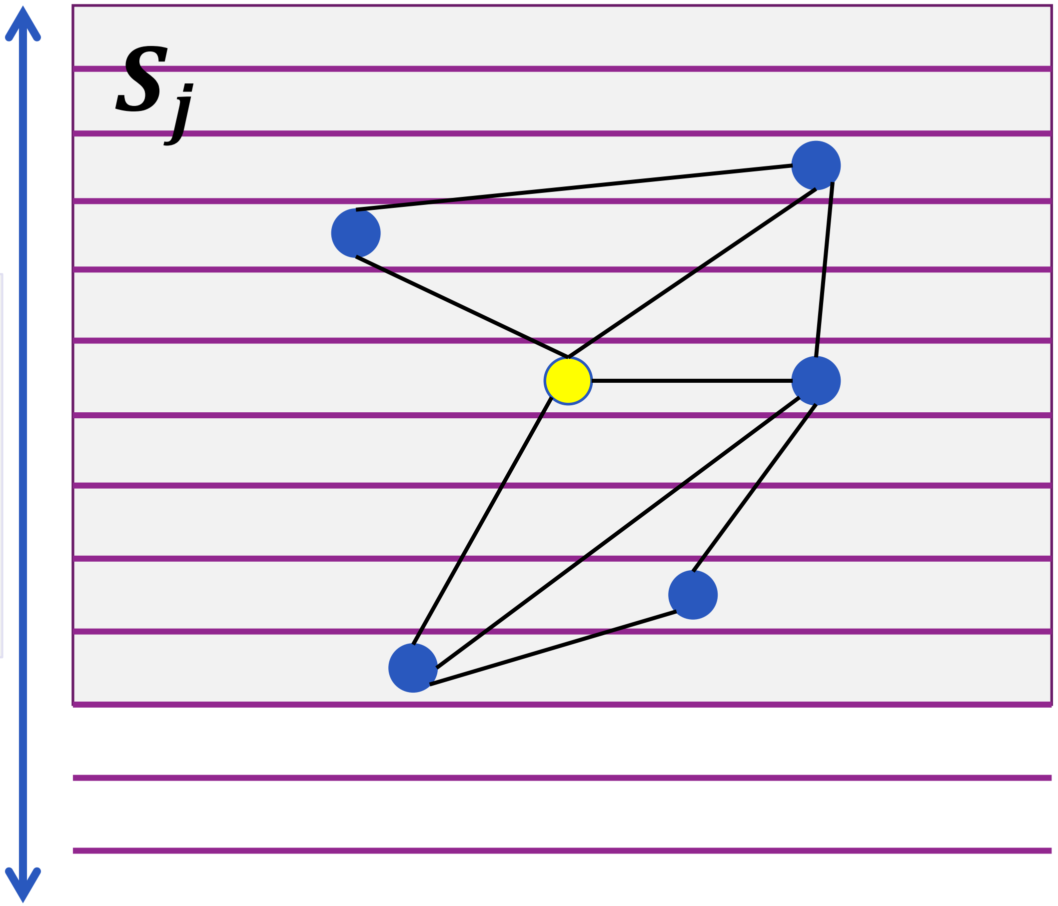
At least $\left(\frac{(1+\epsilon)^i}{2}\right)^j$ edges must be pruned

$$\left(\frac{(1+\epsilon)^i}{2}\right)^{j-1} \leq n$$

$$j \leq \log_{(1+\epsilon)^i/2}(n)$$

Must be the case that:

$$c(v) \geq \frac{(1+\epsilon)^i}{2.5}$$



Pruning Procedure

Remove all w where

$$d_{S_j}(w) < \frac{(1+\epsilon)^i}{2.5}$$

nodes at or above level of v is: $\geq (1+\epsilon)^i$

Run out of vertices before first level of the group.