Deterministic Max-Flows in Simple Graphs

Tianyi Zhang, Tsinghua Univ.

Max-flows in simple graphs

Graph $G = (V, E)$

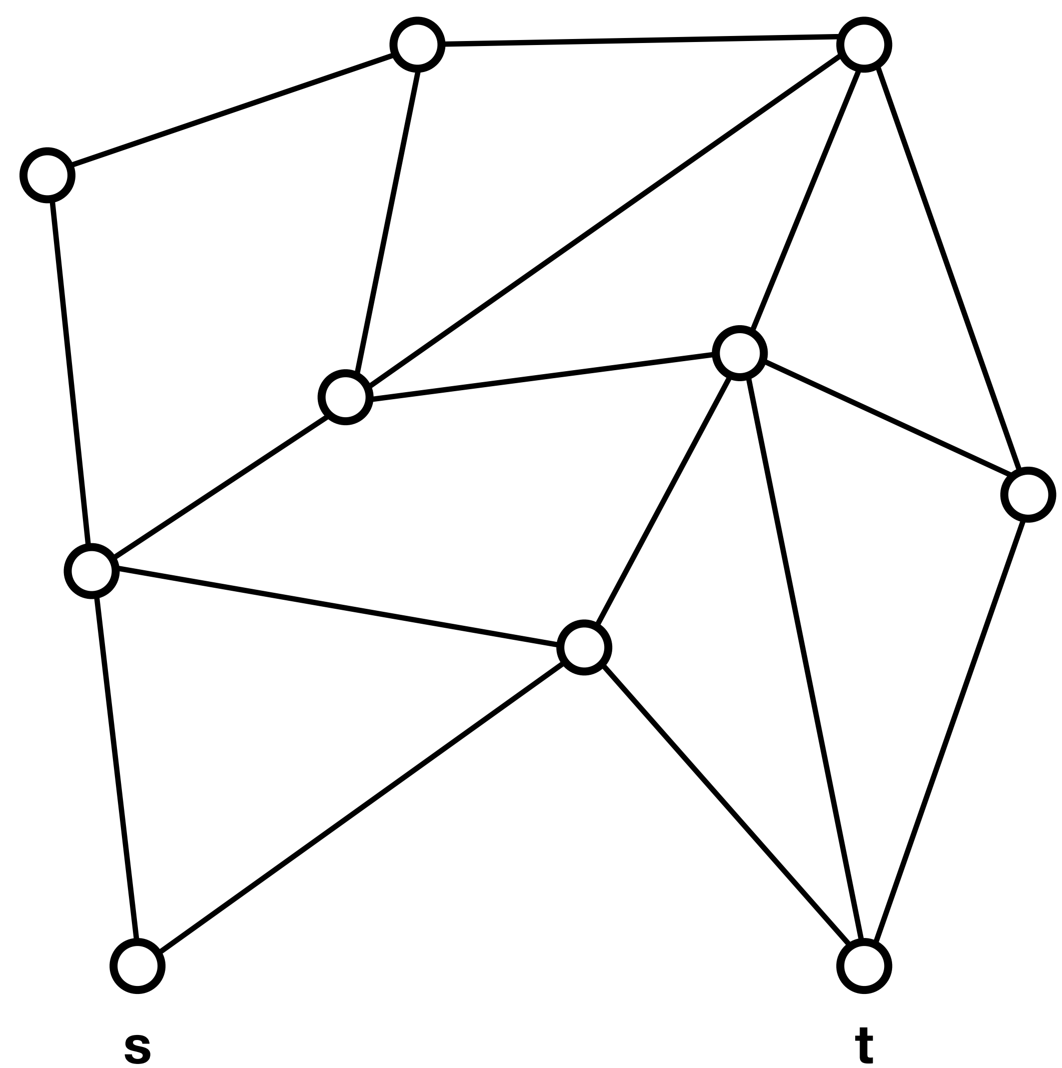
- n vertices
- m edges
- No parallel edges

Capacities

- Unit

Terminals

- $s, t \in V$



Max-flows in simple graphs

Graph $G = (V, E)$

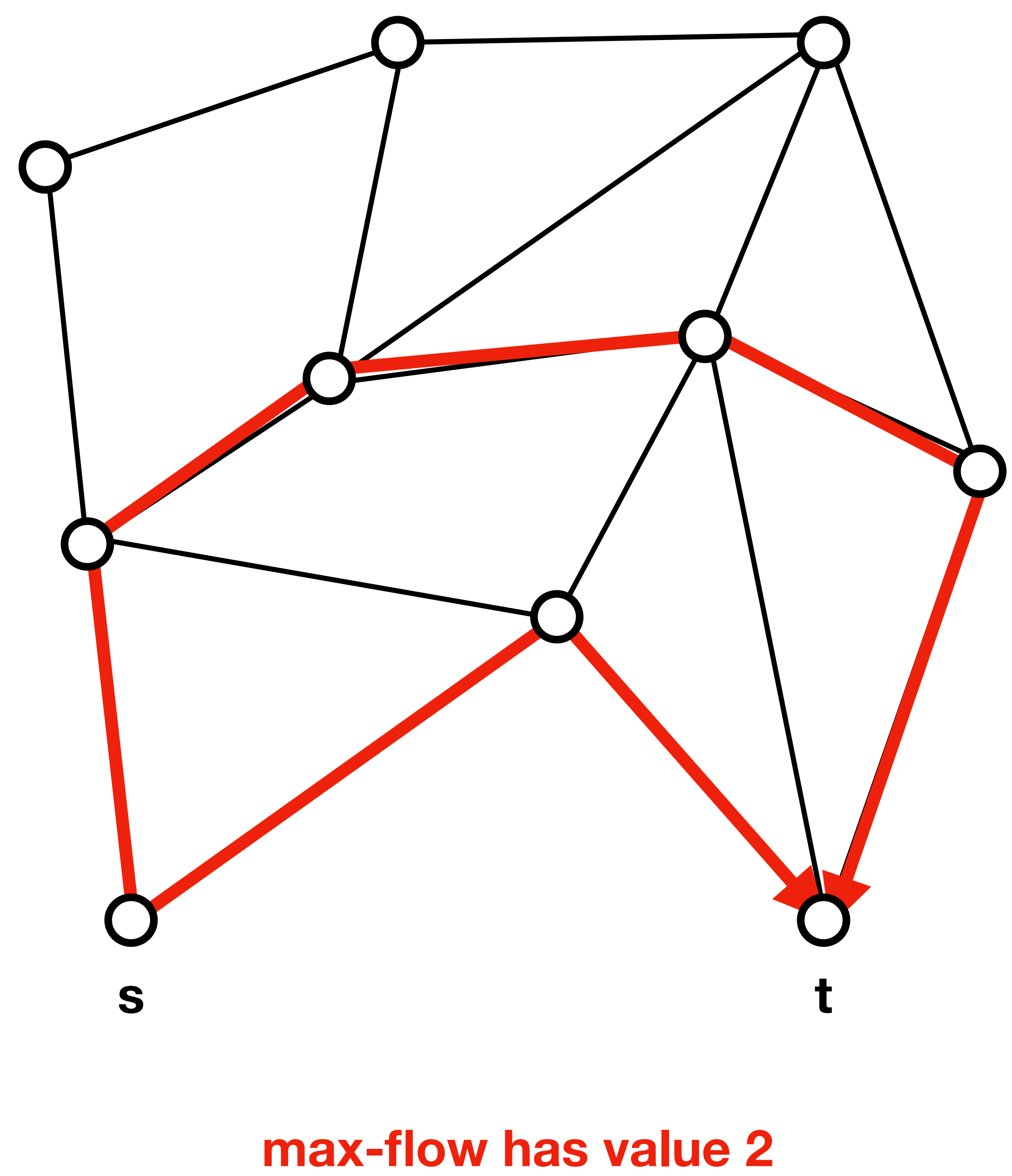
- n vertices
- m edges
- No parallel edges

Capacities

- Unit

Terminals

- $s, t \in V$



History

Reference	Running time	det / rand ?
[KL'98]	$O(m + n\tau^{3/2})$	det
[KL'02]	$\tilde{O}(m + n\tau)$	rand
[Duan'13]	$\tilde{O}(n^{9/4}\tau^{1/8})$	det
Ours	$\tilde{O}(m + n^{5/3}\tau^{1/2})$	det

Fastest when $\tau > n^{0.67}$

n = #vertices, m = #edges

τ = an upper bound on max-flow

Flow decycling

Ford-Fulkerson

- Residual graph G_f of G w.r.t flow f

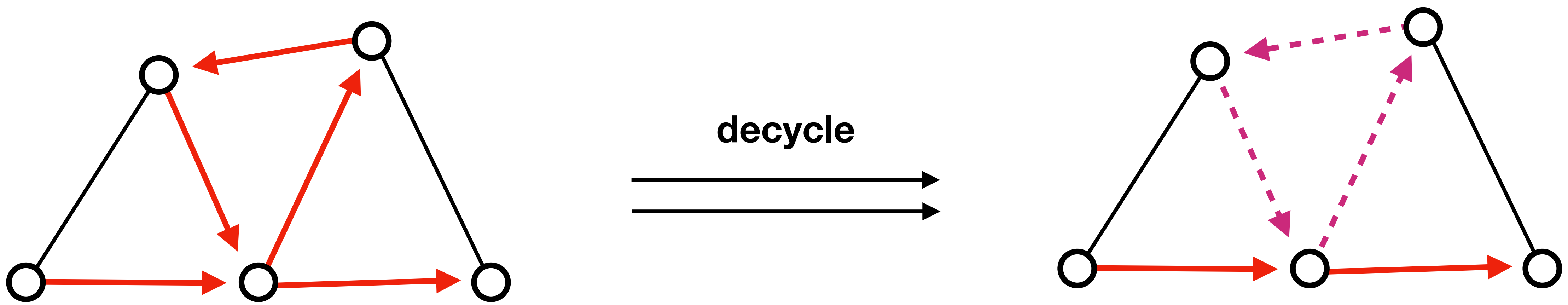
- <u>Ford-Fulkerson:</u>
  Keep finding augmenting paths from s to t in G_f

- Running time = $\tilde{O}(m\tau)$
  τ is a known upper bound on the max-flow value

Flow decycling

- <u>Lemma: [Karger & Levine '98]</u>
  Acyclic flow f with value $|f|$ has $O(n|f|^{1/2})$ flow edges

- G_f has at most $O(n|f|^{1/2})$ directed edges when f is acyclic



many flow edges

at most $O(n|f|^{1/2})$ flow edges

Flow decycling

- Lemma: [Karger & Levine '98]
  Acyclic flow f with value $|f|$ has $O(n|f|^{1/2})$ flow edges

- G_f has at most $O(n|f|^{1/2})$ directed edges when f is acyclic

- Algorithm: [Karger & Levine '98]
  While $\exists$ augmenting path in G_f

  contract all connected components by **undi-**edges in G_f

  BFS on the contracted G_f which contains only $O(n|f|^{1/2})$ di-edges

  augment flow f, then decycle f

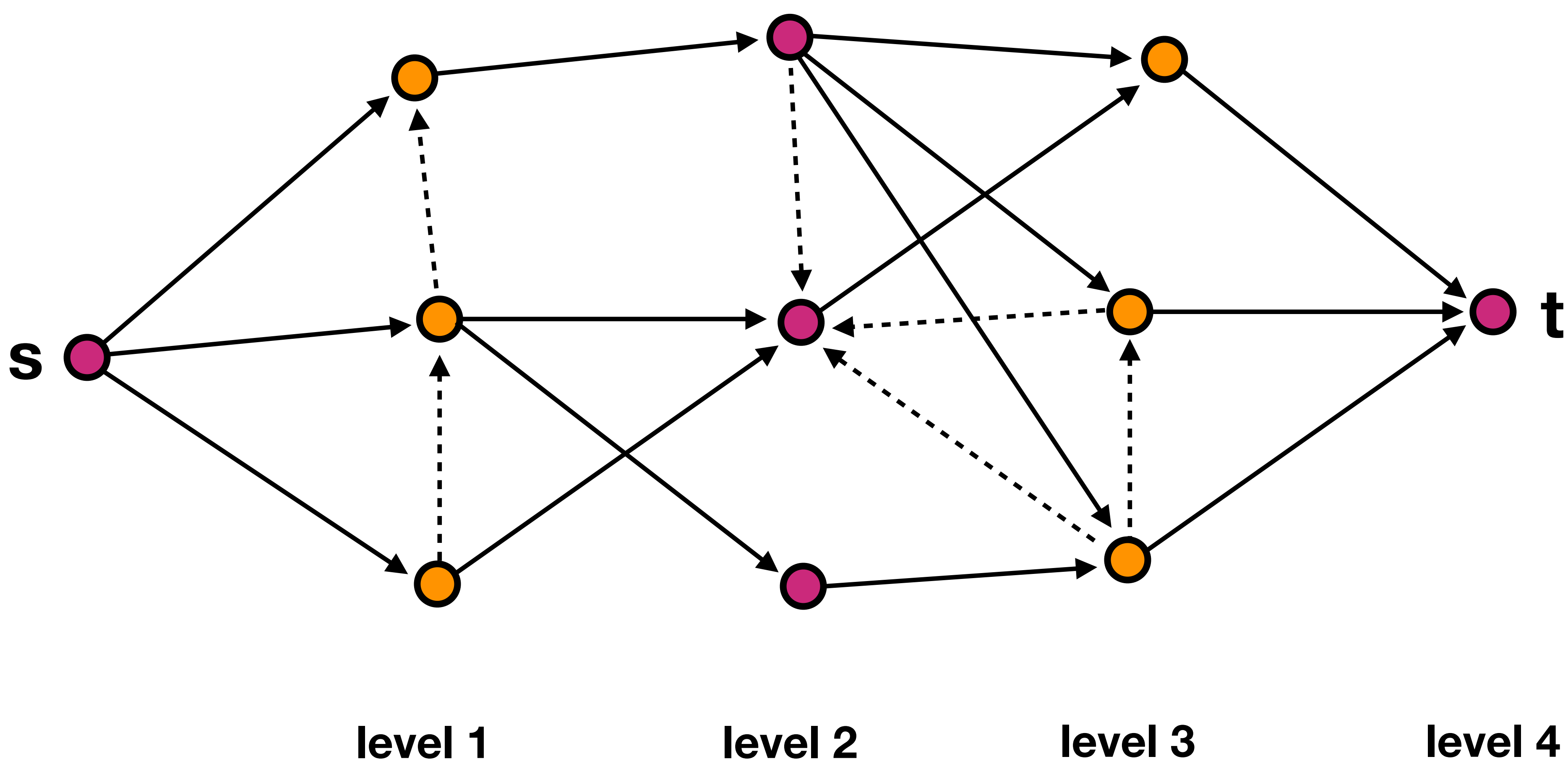
- Running time = $\tilde{O}(m + n\tau^{3/2})$

  G_f always has $O(n\tau^{1/2})$ edges, so total time = $\tilde{O}(m + \tau \cdot n\tau^{1/2})$

Blocking flows

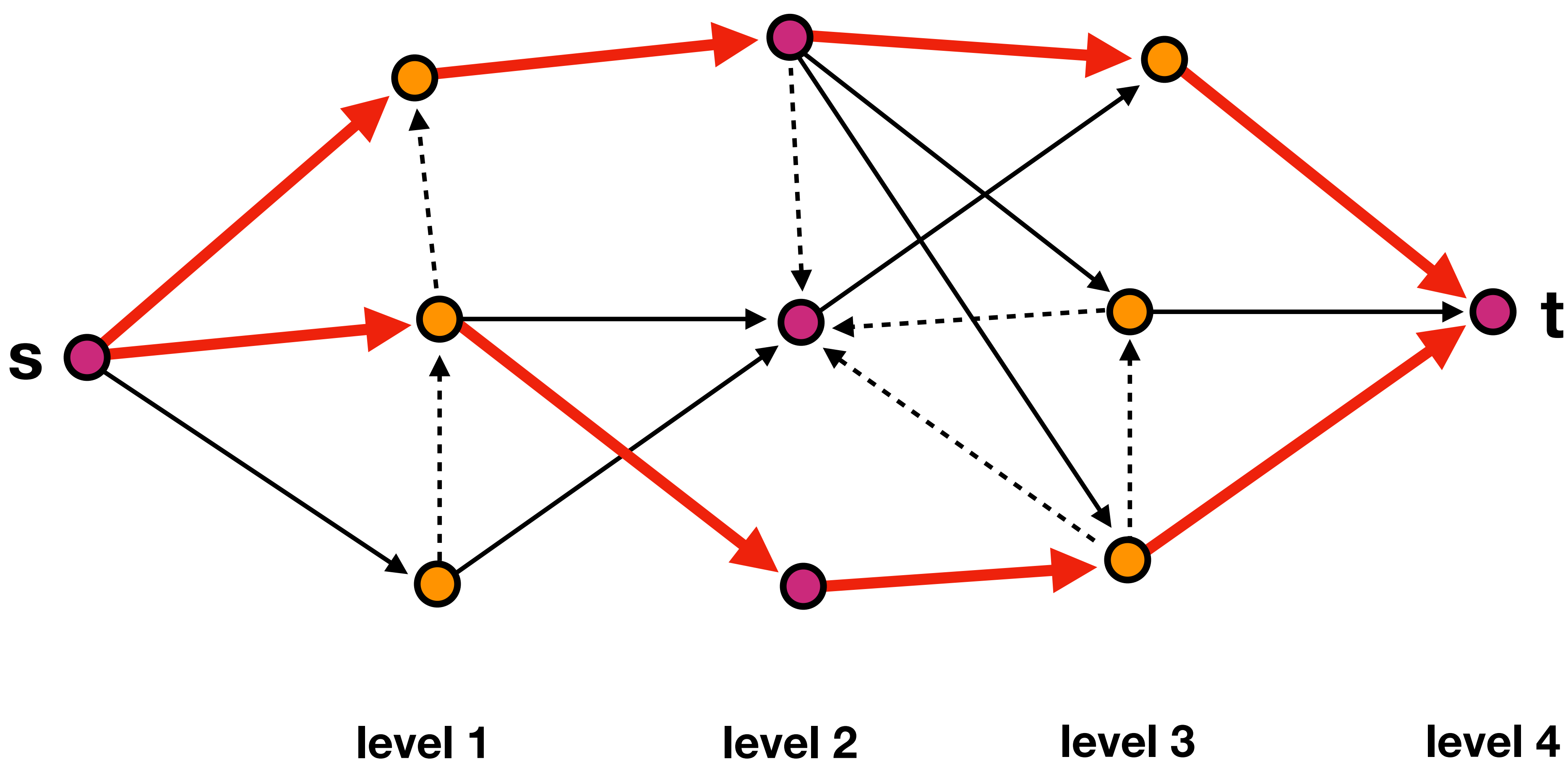
Blocking flows

- Form a level graph by the distance from s in G_f



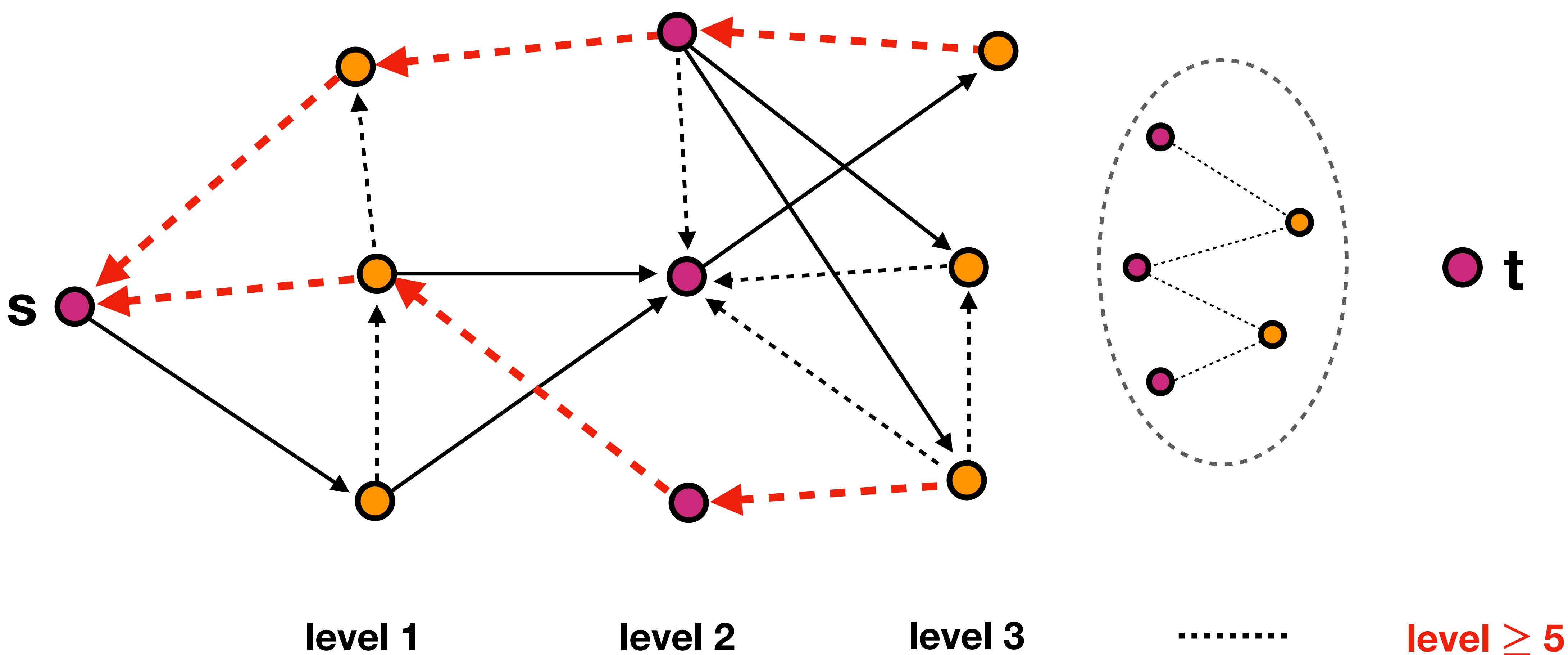
Blocking flows

- Form a level graph by the distance from s in G_f
- Find a maximal set of shortest disjoint aug-paths



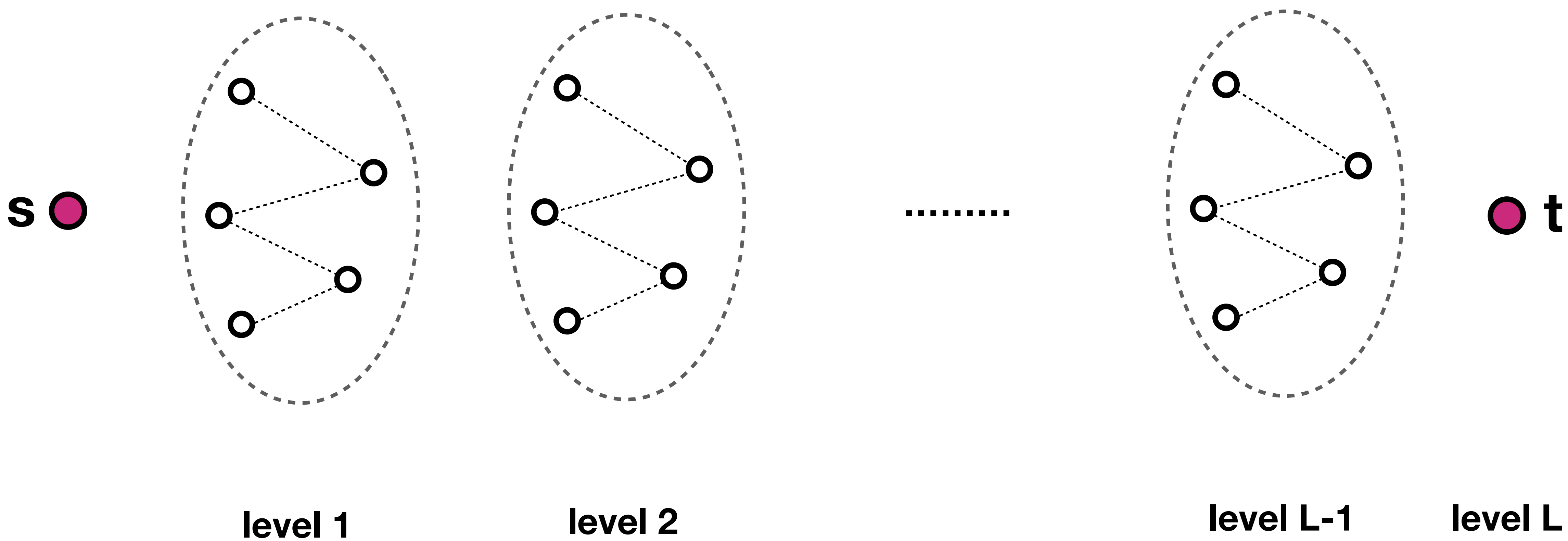
Blocking flows

- Form a level graph by the distance from s in G_f
- Find a maximal set of shortest disjoint aug-paths
- $\mathrm{Dist}(s, t)$ in G_f increases



Blocking flows

- Repeat blocking-flows until $\mathrm{Dist}(s,t) \geq L$
- Residual flow in G_f becomes at most $O(n^2/L^2)$
- Then apply Ford-Fulkerson $O(n^2/L^2)$ times



- Running time = $L \cdot \mathrm{BF} + \frac{n^2}{L^2} \cdot \mathrm{FF} = mn^{2/3}$ [Goldberg & Rao '98]

Decycling + Blocking-flow

- Lemma: [Karger & Levine '98]
  Acyclic flow f with value $|f|$ has $O(n|f|^{1/2})$ flow edges

- Exists subgraph $H \subseteq G$ with $O(n\tau^{1/2})$ edges that contains the max-flow

- If we knew H beforehand, then applying blocking-flow on H can compute max-flow in $\tilde{O}(n^{5/3}\tau^{1/2})$ time

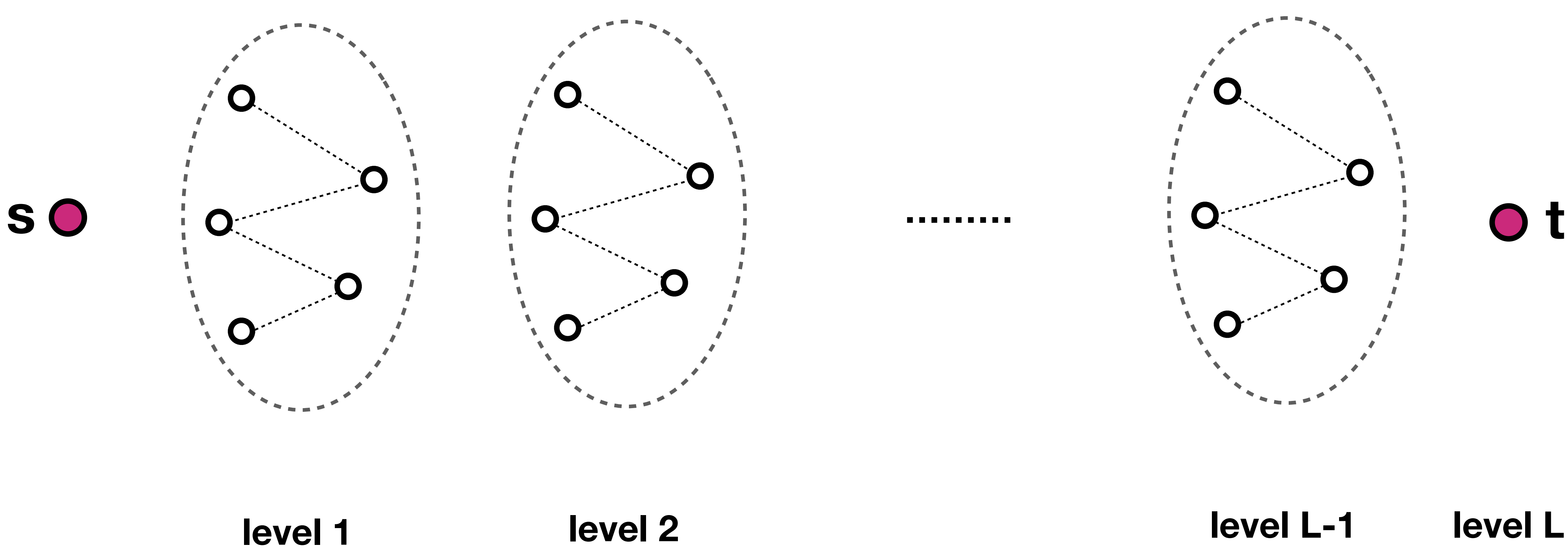
- Ideally, shoot for $\tilde{O}(m + n^{5/3}\tau^{1/2})$

Decycling + blocking-flows
[Duan '13]

Combining two techniques

- Compute a blocking-flow in G_f where f is acyclic

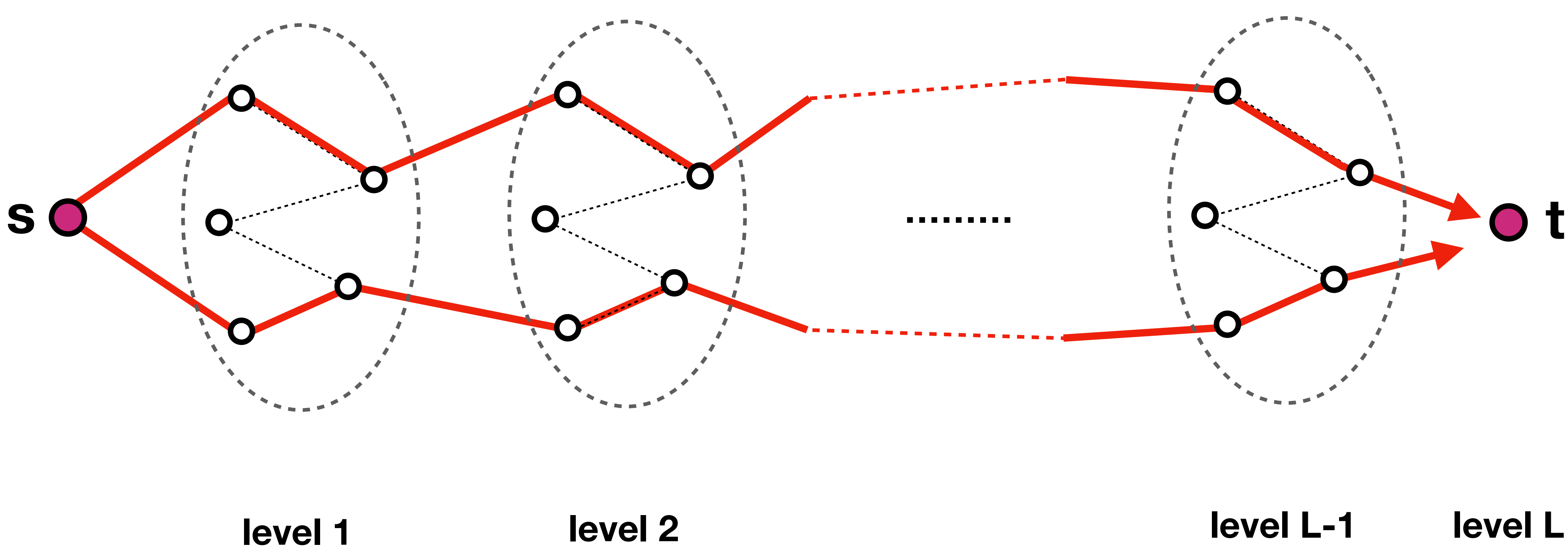
  Take time $\tilde{O}(n|f|^{1/2})$ since undirected edges are contracted



Combining two techniques

- Compute a blocking-flow in G_f where f is acyclic

  Take time $\tilde{O}(n|f|^{1/2})$ since undirected edges are contracted



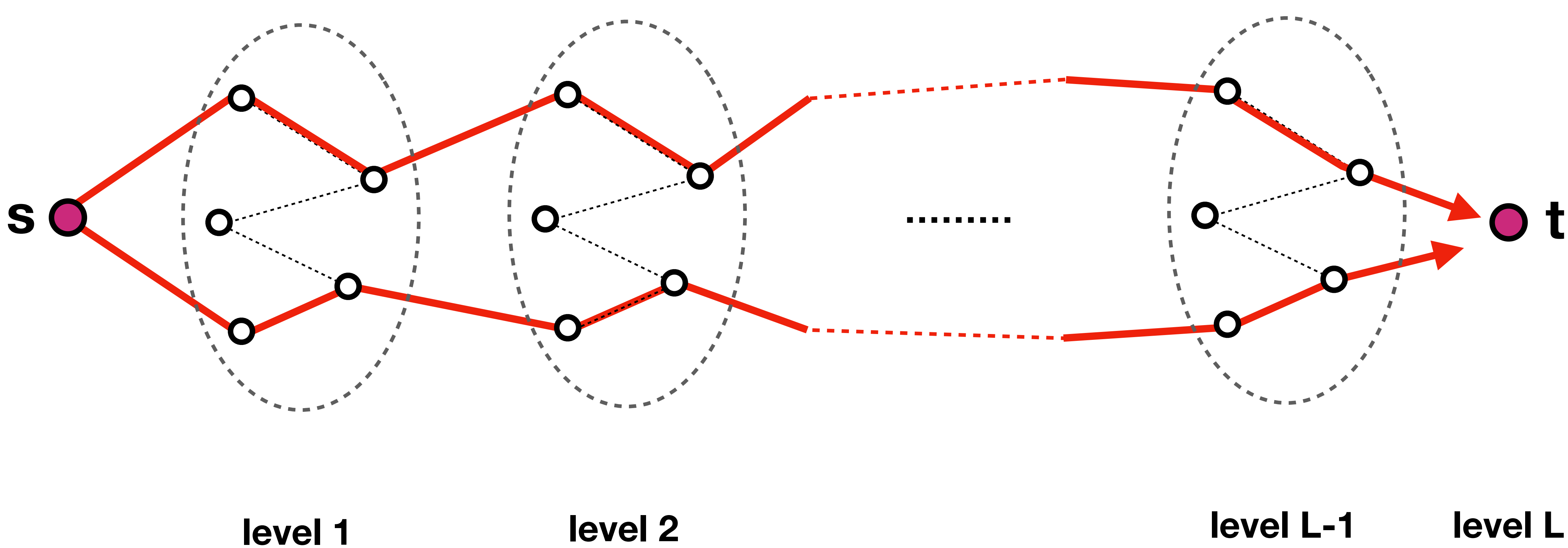
Combining two techniques

- Compute a blocking-flow in G_f where f is acyclic

  Take time $\tilde{O}(n|f|^{1/2})$ since undirected edges are contracted

- Augment flow $f \leftarrow f + \Delta f$, so $\mathrm{Dist}(s,t)$ in G_f increases

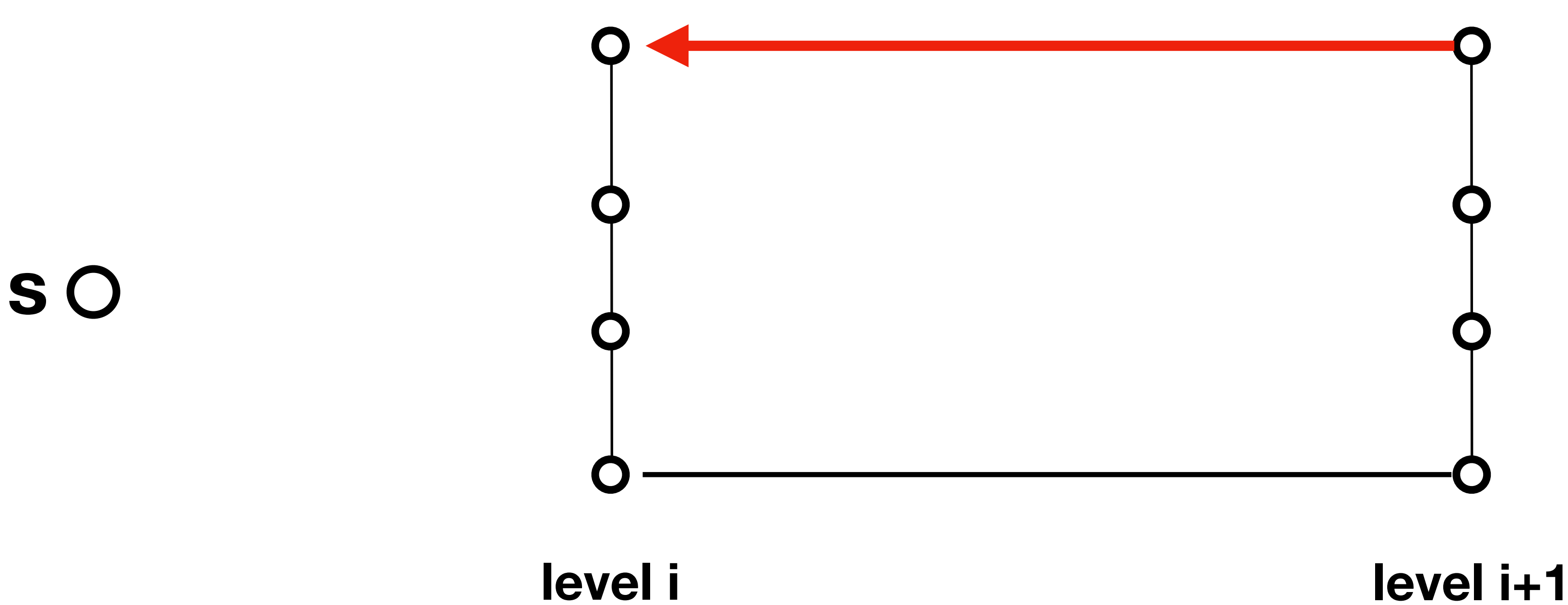
  But now, f might contain cycles



Combining two techniques

- Augment flow $f \leftarrow f + \Delta f$, so $\mathrm{Dist}(s, t)$ in G_f increases

  But now, f might contain cycles



Combining two techniques

- Augment flow $f \leftarrow f + \Delta f$, so $\mathrm{Dist}(s, t)$ in G_f increases

  But now, f might contain cycles

Combining two techniques

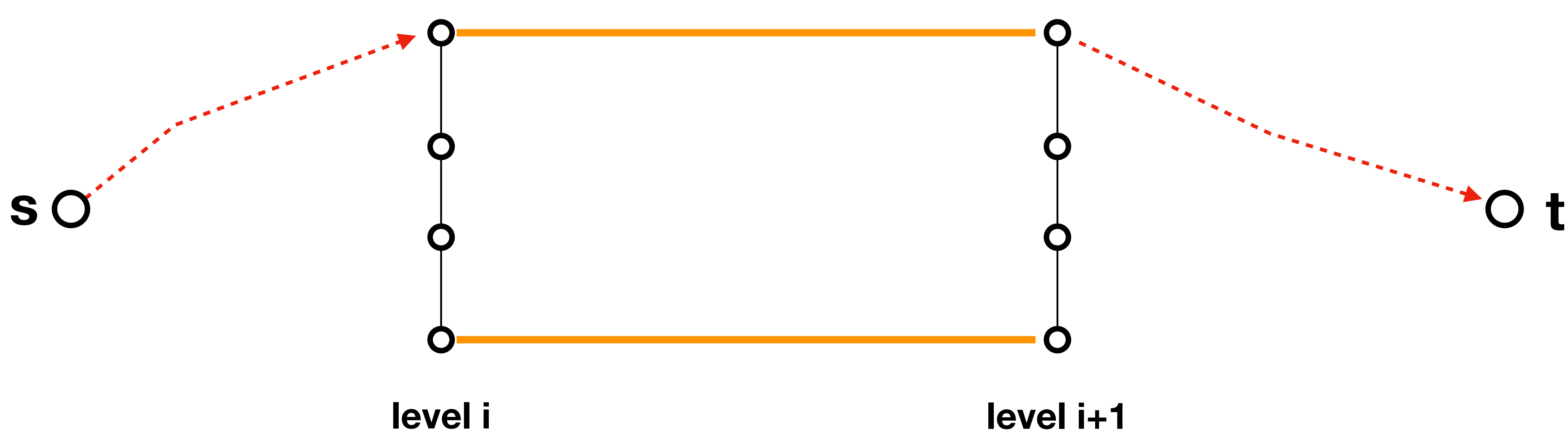
- Augment flow $f \leftarrow f + \Delta f$, so $\mathrm{Dist}(s, t)$ in G_f increases

  But now, f might contain cycles

- Decycling adds undirected edges between level i & (i+1)

- Blocking-flows becomes costly as #undi-edges grows

  Cannot contract these undirected edges

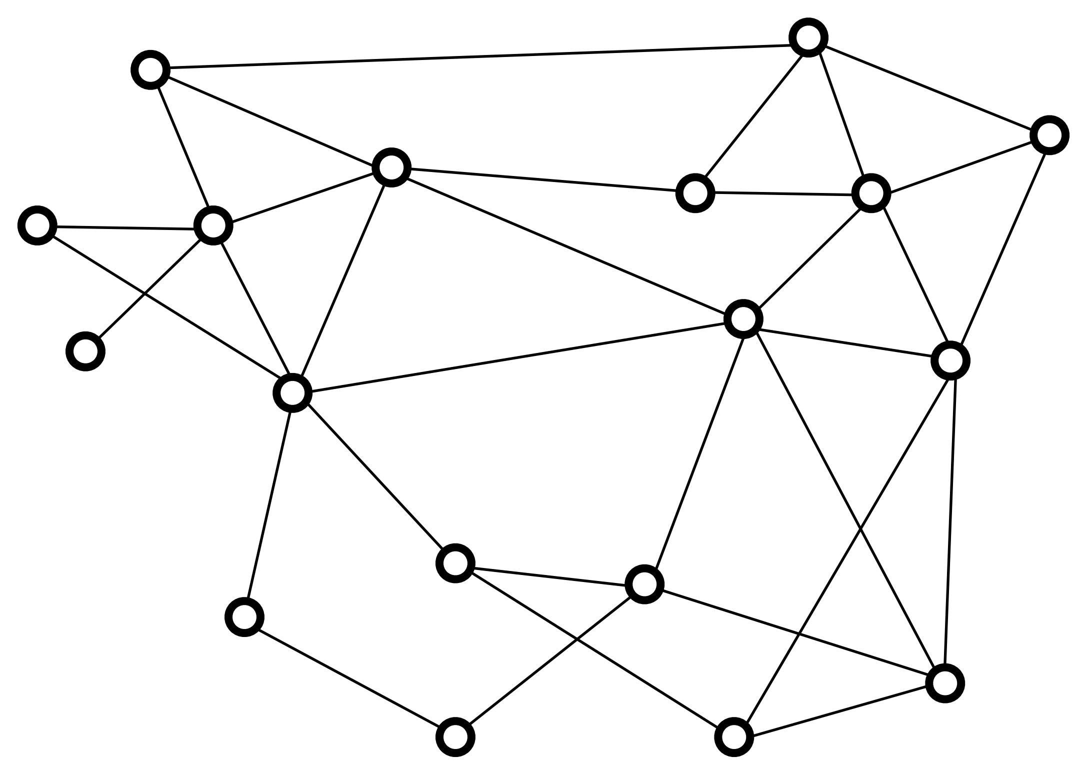


Clustering

- Trouble: #undi-edges grows larger than $n\tau^{1/2}$
  Computing blocking-flows becomes costly

- <u>Key idea: [Duan'13]</u> partition into star-subgraphs

- Partition vertices into $V_1 \cup V_2$
- V_1 is a union of star-graphs, each of size $\geq h$
- Edges between V_2 is at most $O(nh)$

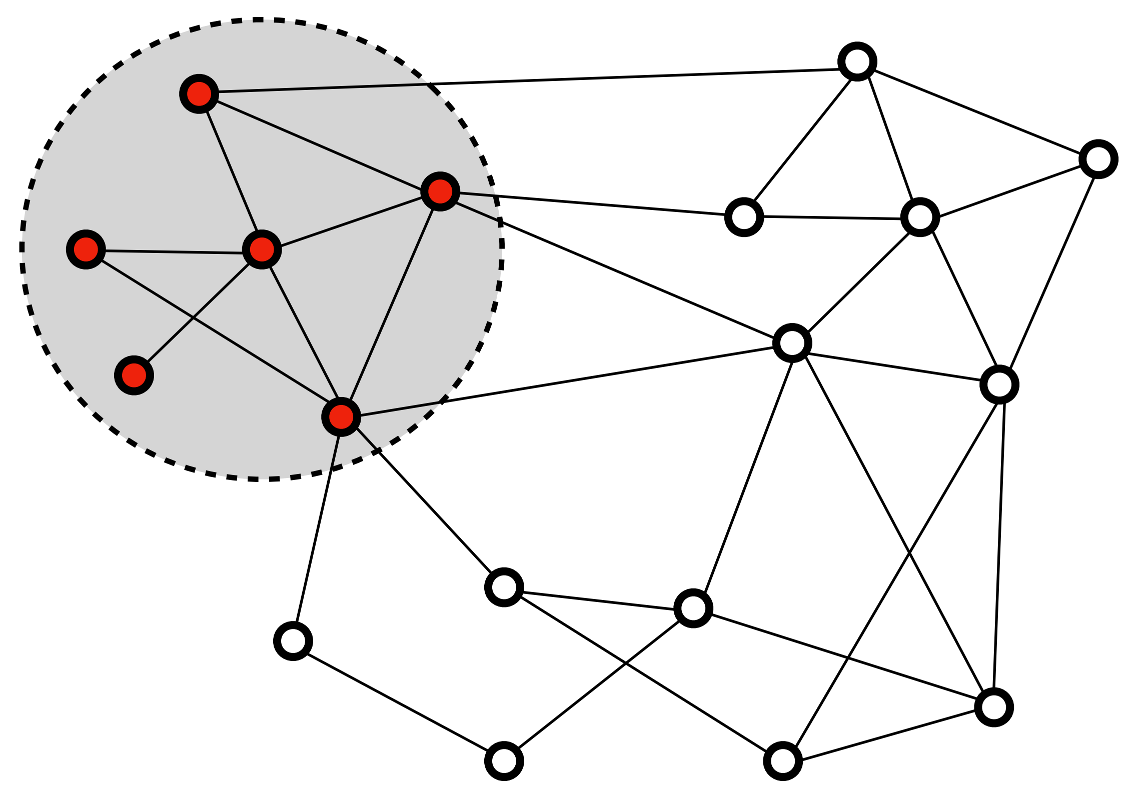


Clustering

- Trouble: #undi-edges grows larger than $n\tau^{1/2}$
  Computing blocking-flows becomes costly

- <u>Key idea: [Duan'13]</u> partition into star-subgraphs

> - Partition vertices into $V_1 \cup V_2$
> - V_1 is a union of star-graphs, each of size $\geq h$
> - Edges between V_2 is at most $O(nh)$



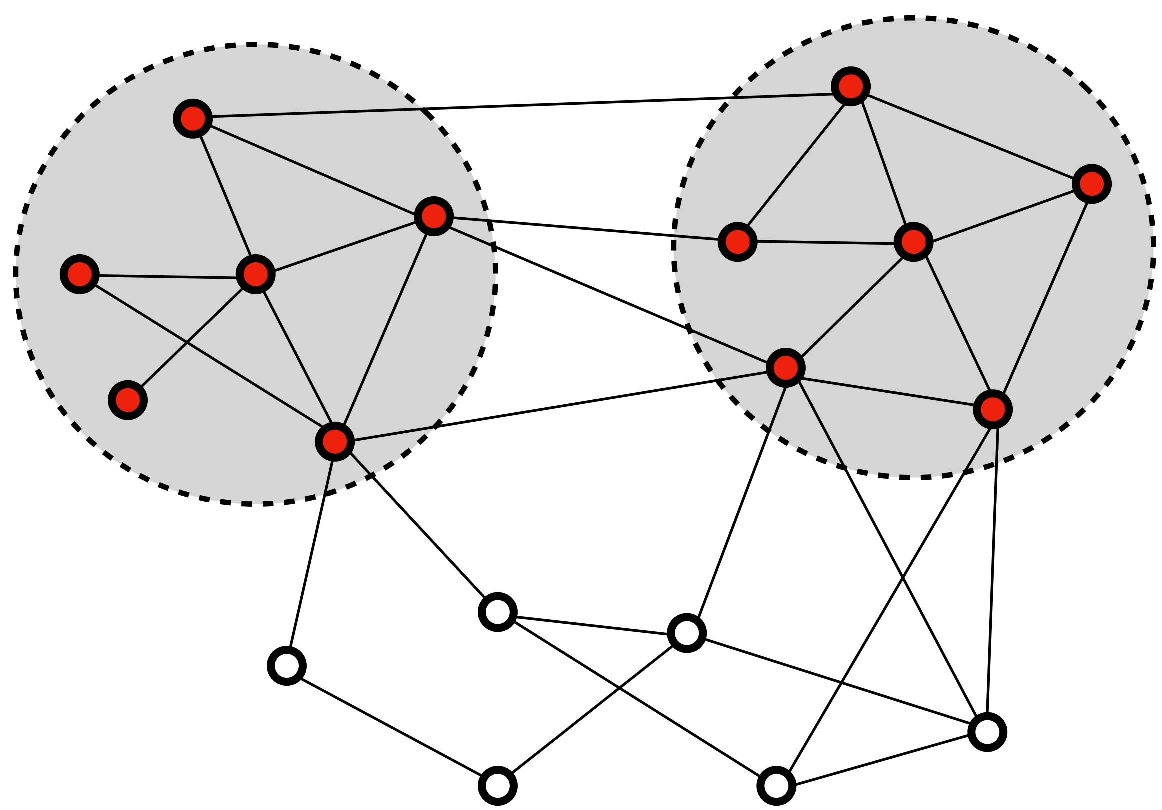
greedy clustering

Clustering

- Trouble: #undi-edges grows larger than $n\tau^{1/2}$
  Computing blocking-flows becomes costly

- Key idea: [Duan'13] partition into star-subgraphs

> - Partition vertices into $V_1 \cup V_2$
> - V_1 is a union of star-graphs, each of size $\geq h$
> - Edges between V_2 is at most $O(nh)$



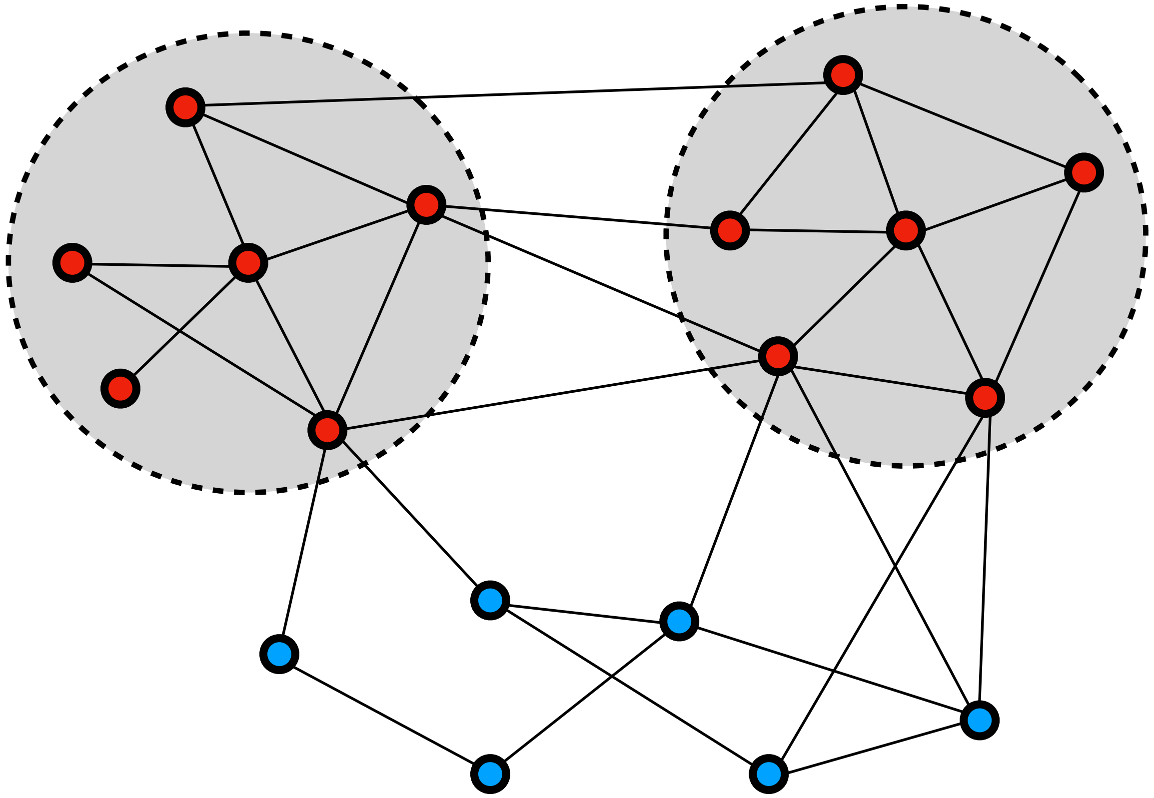
greedy clustering

Clustering

- Trouble: #undi-edges grows larger than $n\tau^{1/2}$
  Computing blocking-flows becomes costly

- <u>Key idea: [Duan'13]</u> partition into star-subgraphs

- Partition vertices into $V_1 \cup V_2$
- V_1 is a union of star-graphs, each of size $\geq h$
- Edges between V_2 is at most $O(nh)$



greedy clustering

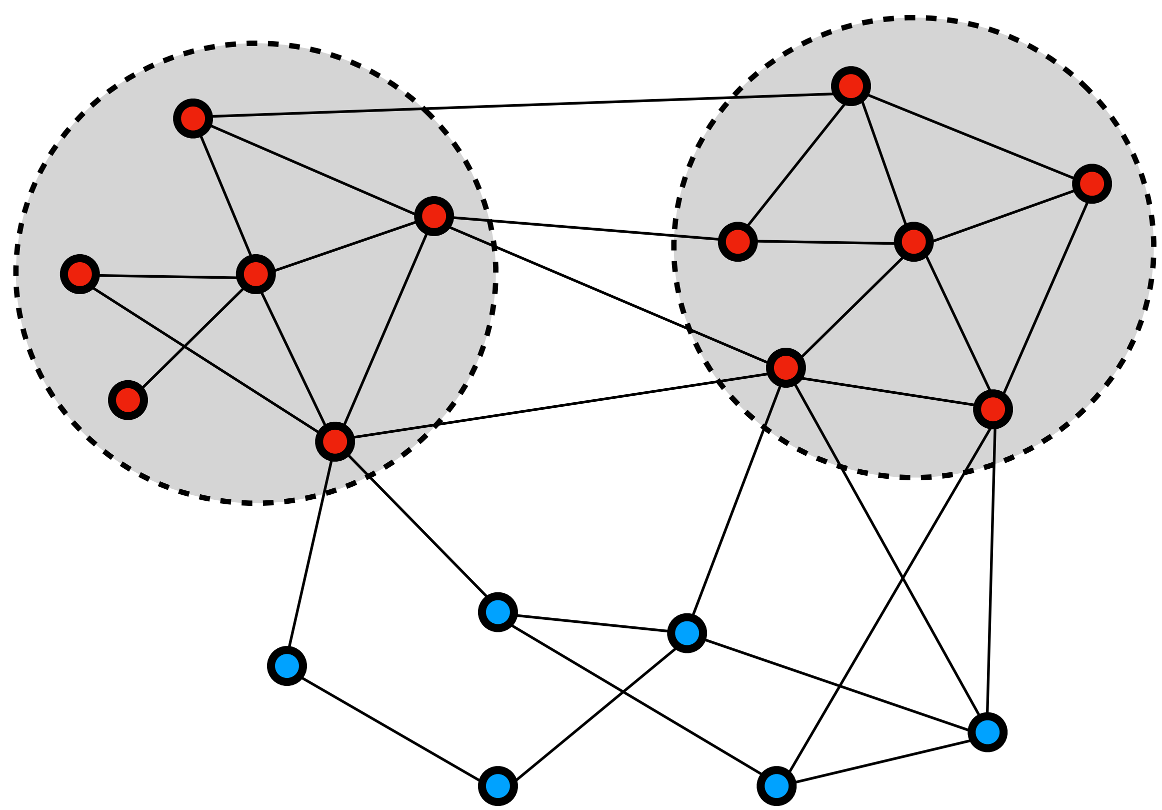
Dynamic maintenance

- Updates: augmentations turn undi-edgs to di-edges
  Delete di-edges from the clustering structure

- Need dynamic maintenance of the clustering structure

Deletion of star edges:

- Disconnects the vertex, and move it downward
- Increase total degree by at most $O(n)$
- Rebuild if total degree exceeds nh



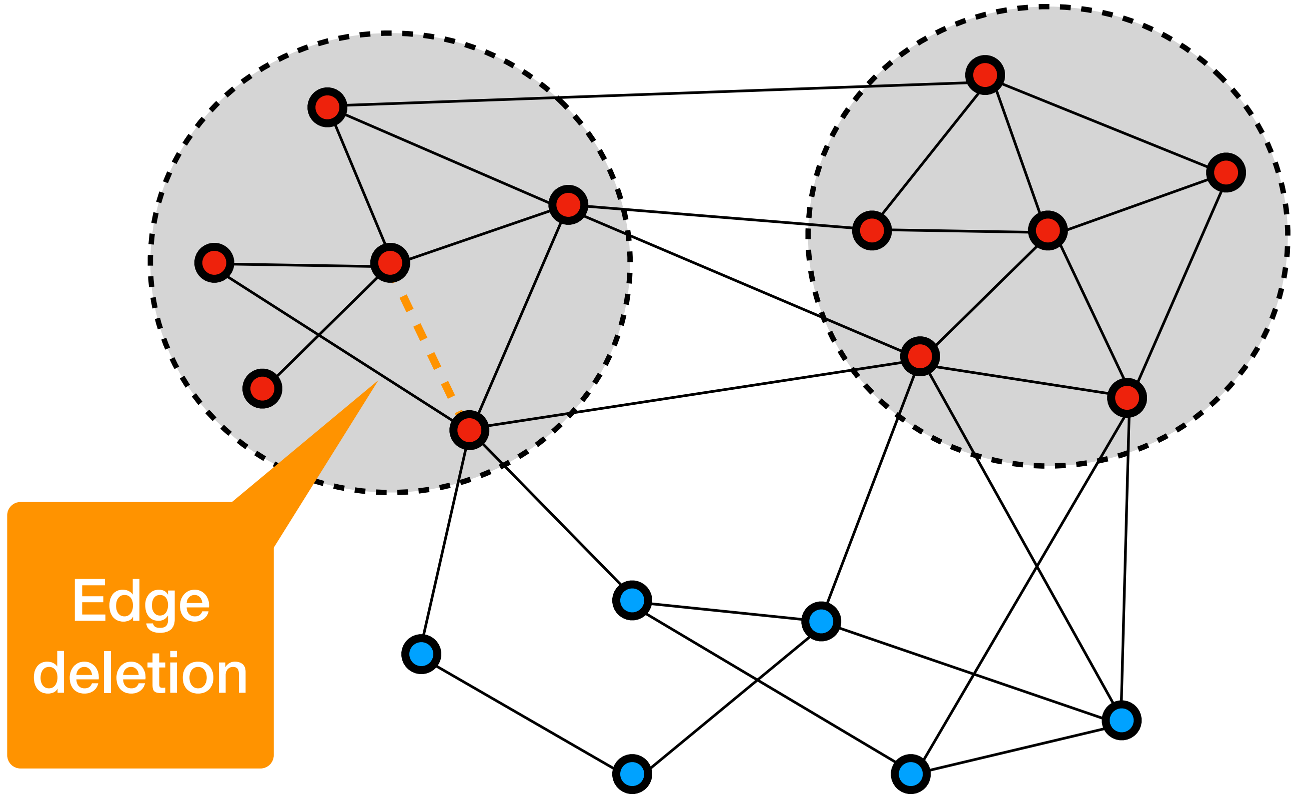
Dynamic maintenance

- <u>Updates:</u> augmentations turn undi-edgs to di-edges
  Delete di-edges from the clustering structure

- Need dynamic maintenance of the clustering structure

Deletion of star edges:

- Disconnects the vertex, and move it downward
- Increase total degree by at most $O(n)$
- Rebuild if total degree exceeds nh



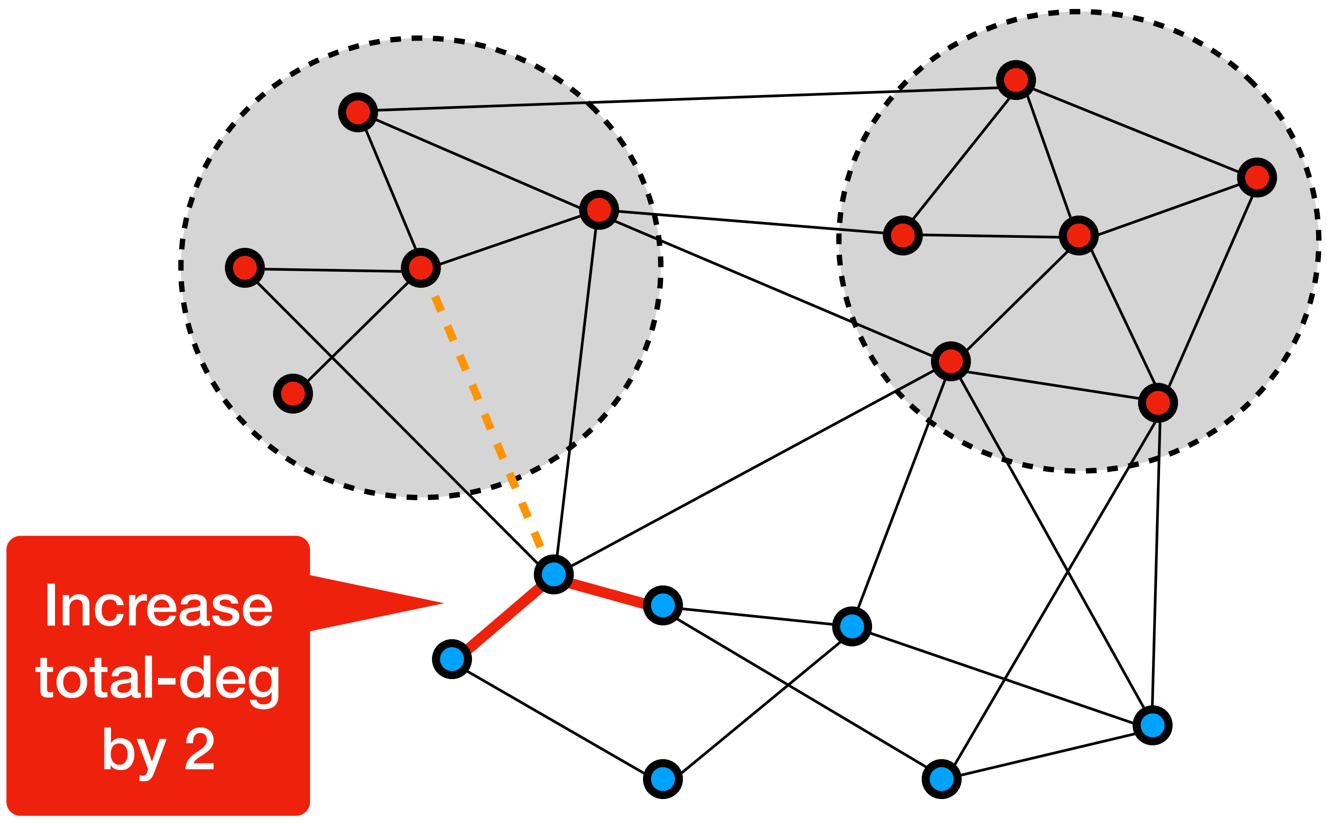
Dynamic maintenance

- Updates: augmentations turn undi-edgs to di-edges
  Delete di-edges from the clustering structure

- Need dynamic maintenance of the clustering structure

Deletion of star edges:

- Disconnects the vertex, and move it downward
- Increase total degree by at most $O(n)$
- Rebuild if total degree exceeds nh

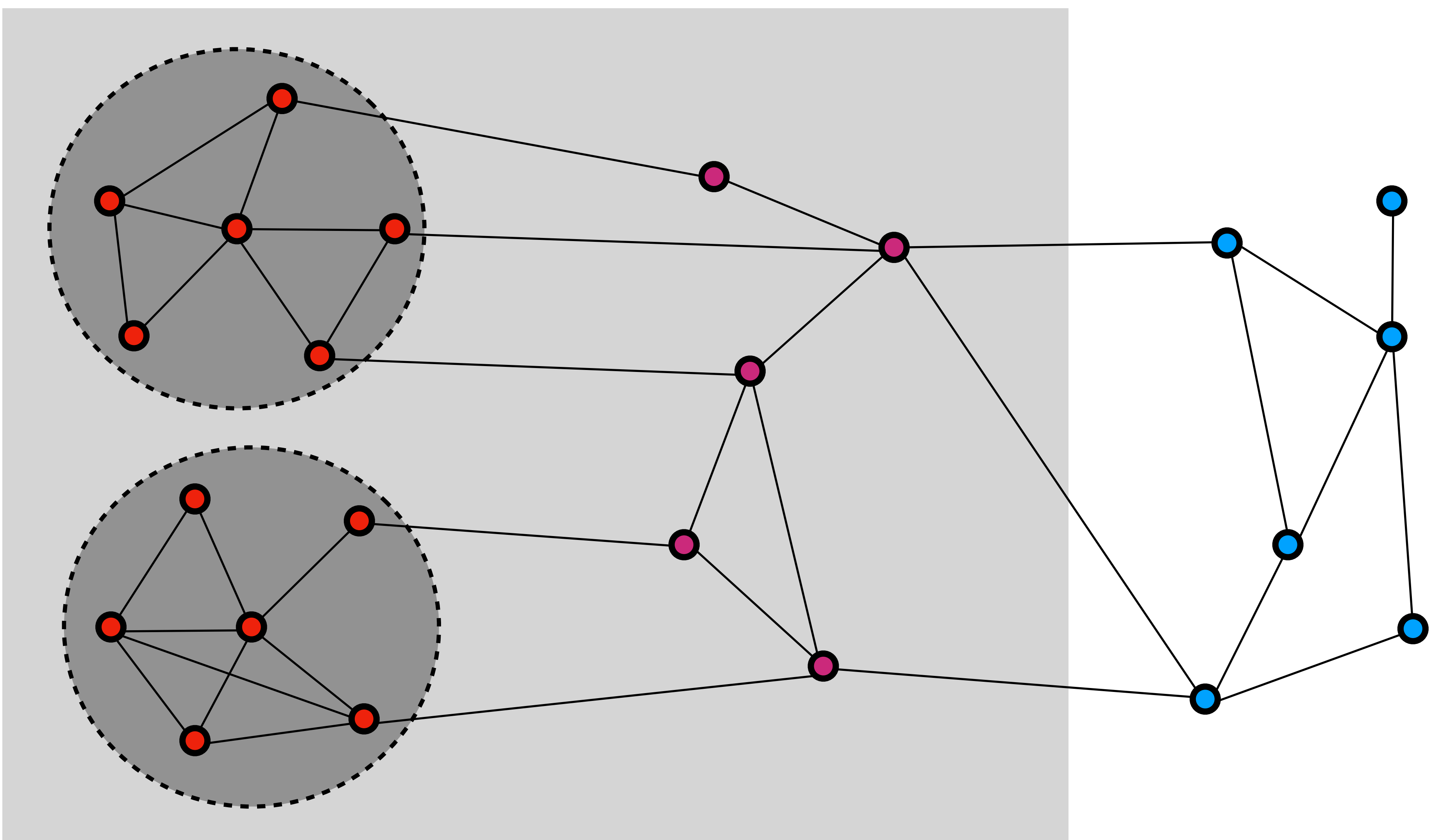


Our technique

a more careful clustering scheme

Avoid rebuilding

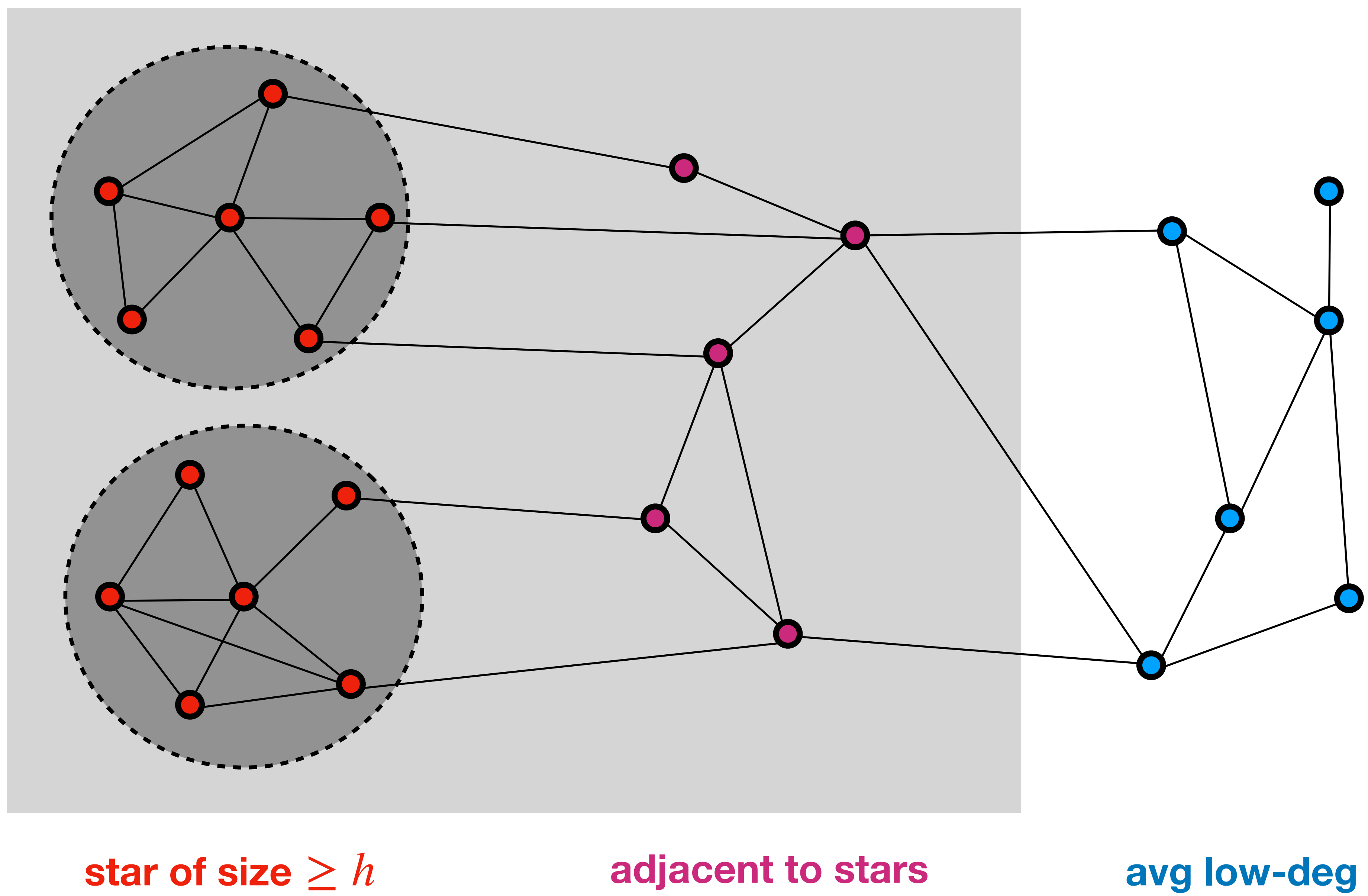
- Previously in [Duan'13], the clustering scheme needs frequent rebuilding procedures
- <u>Solution:</u> try to avoid rebuilding entirely



star of size $\geq h$ adjacent to stars avg low-deg $< h$

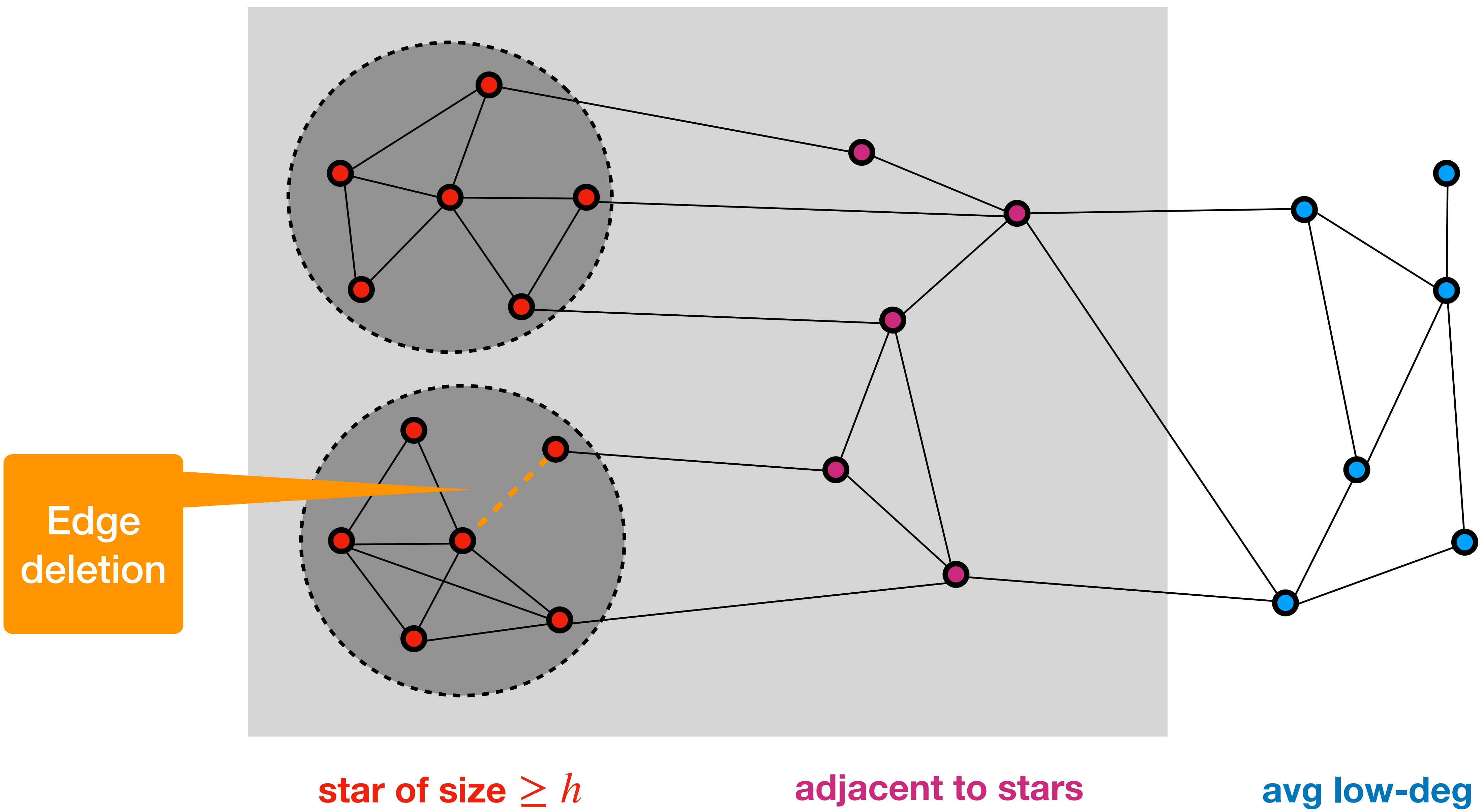
Avoid rebuilding

- Solution: try to avoid rebuilding entirely
- When a star edge is deleted, if it is not adjacent to stars, either collect a new star, or move to low-degree part



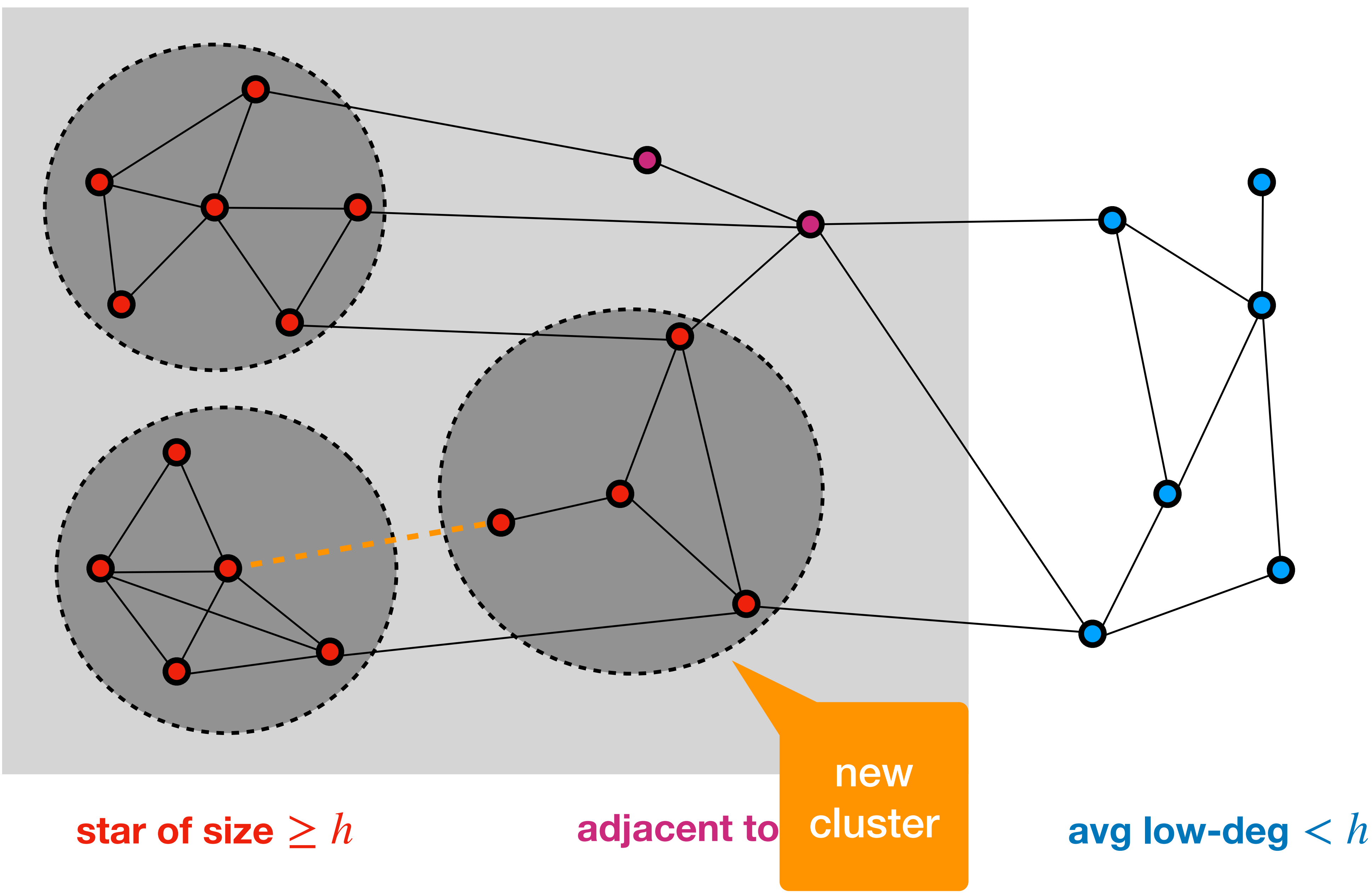
Avoid rebuilding

- <u>Solution:</u> try to avoid rebuilding entirely
- When a star edge is deleted, if it is not adjacent to stars, either collect a new star, or move to low-degree part



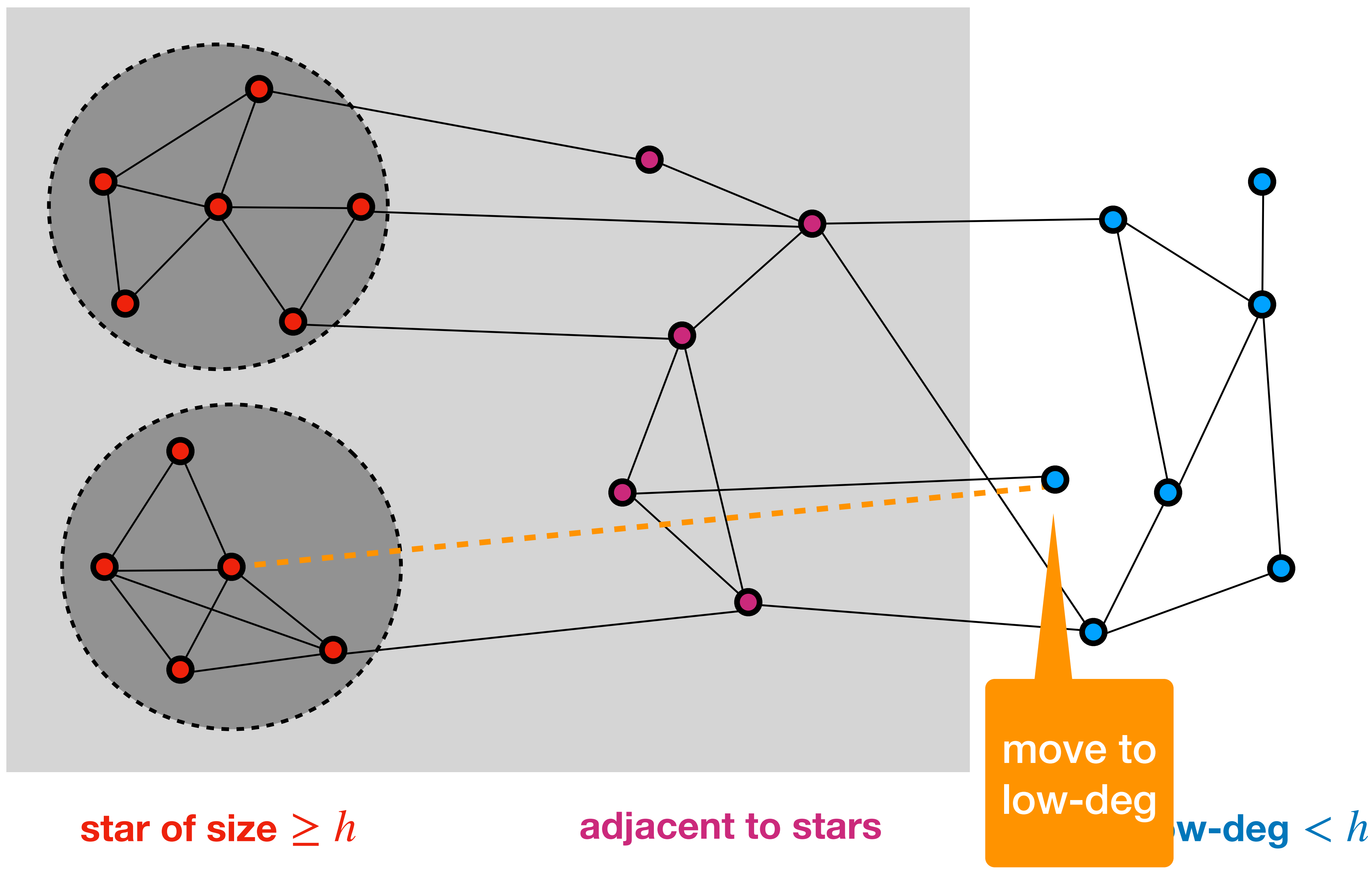
Avoid rebuilding

- <u>Solution:</u> try to avoid rebuilding entirely
- When a star edge is deleted, if it is not adjacent to stars, either collect a new star, or move to low-degree part



Avoid rebuilding

- Solution: try to avoid rebuilding entirely
- When a star edge is deleted, if it is not adjacent to stars, either collect a new star, or move to low-degree part



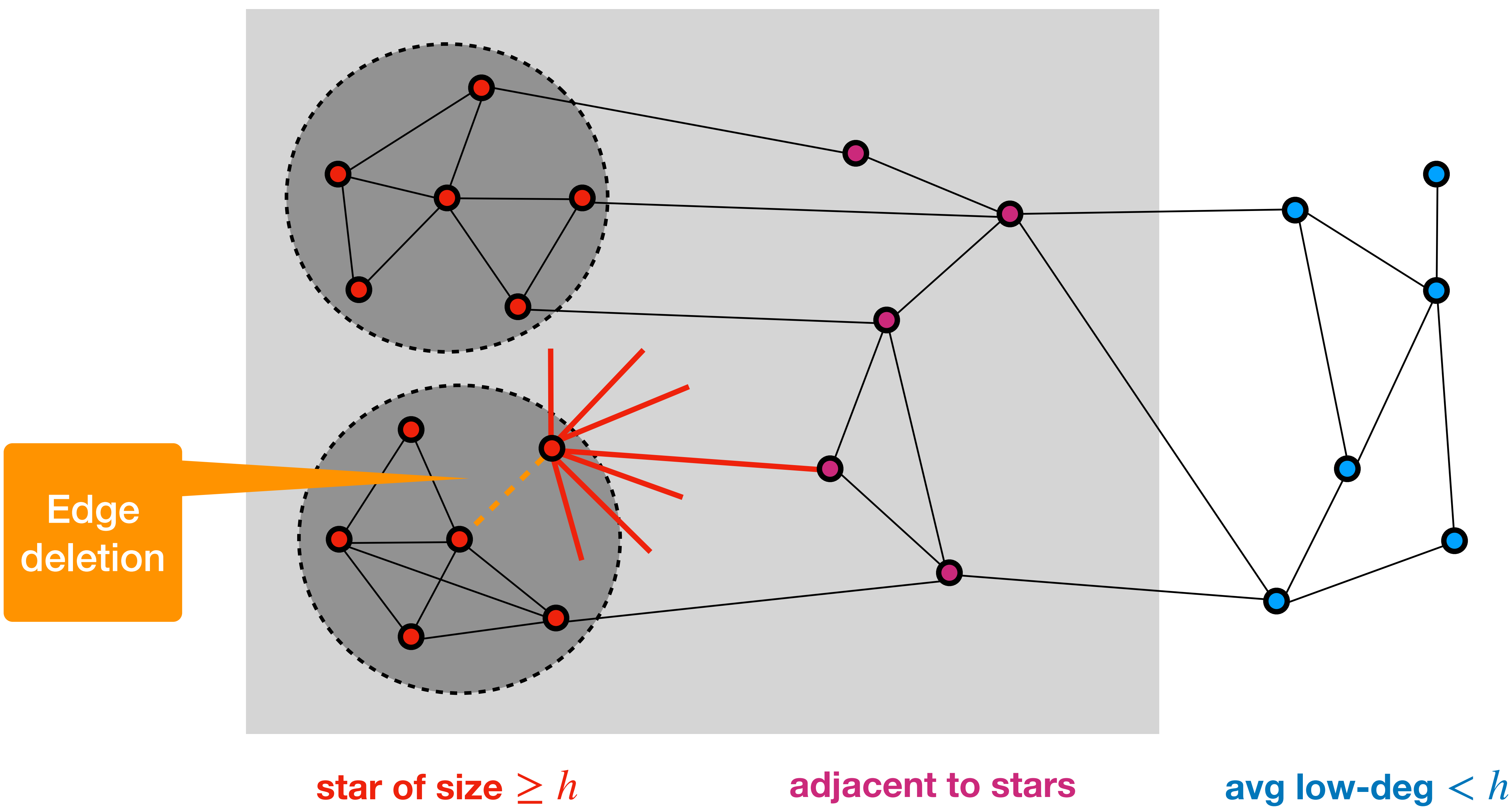
Proof of concept

When $\tau = \Theta(n)$

- [Duan'13] has running time $\tilde{O}(n^{2.375})$ for max-flow
- New clustering scheme already improves to $\tilde{O}(n^{2.25})$

A multi-layer approach

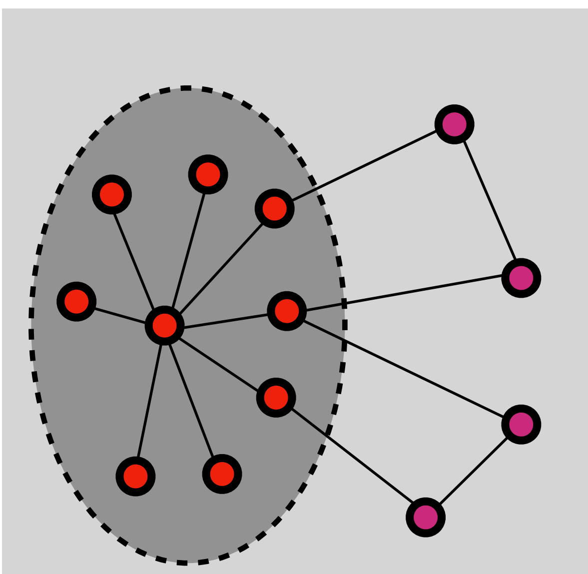
- Handling a deletion needs to scan the adjacency list
  Adjacency list can be as large as $O(n)$
- If adjacency list is large, then can have larger stars
  Use multiple layers to handle different vertex degrees



A multi-layer approach

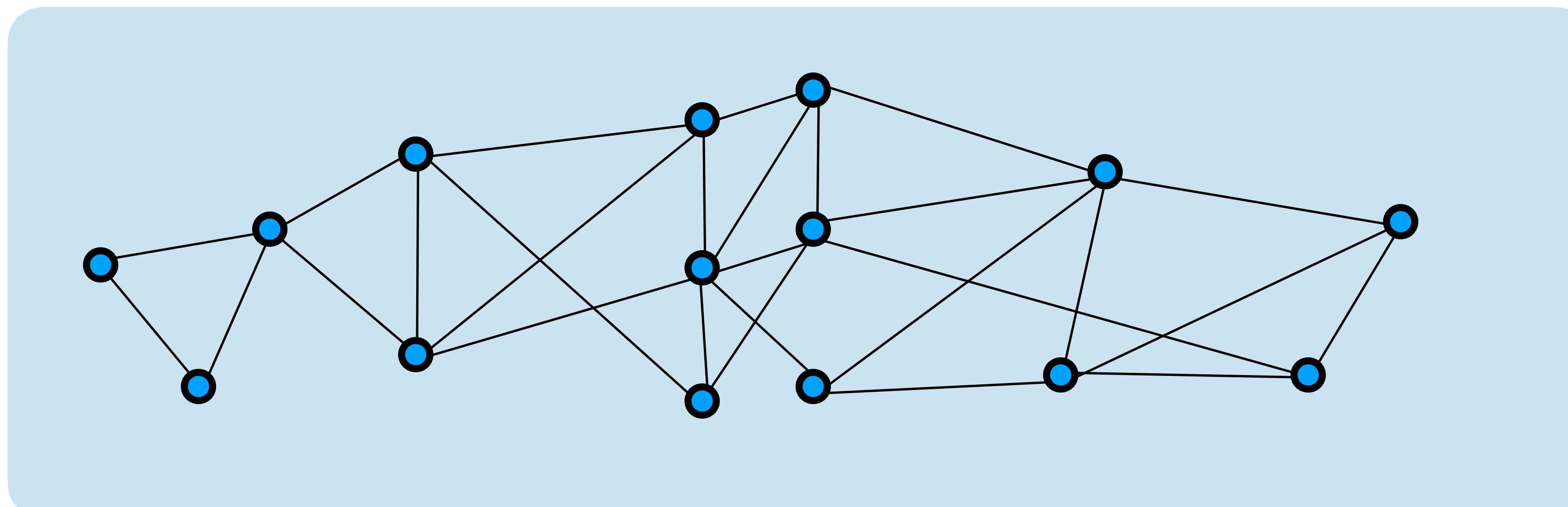
stars

adjacent to stars



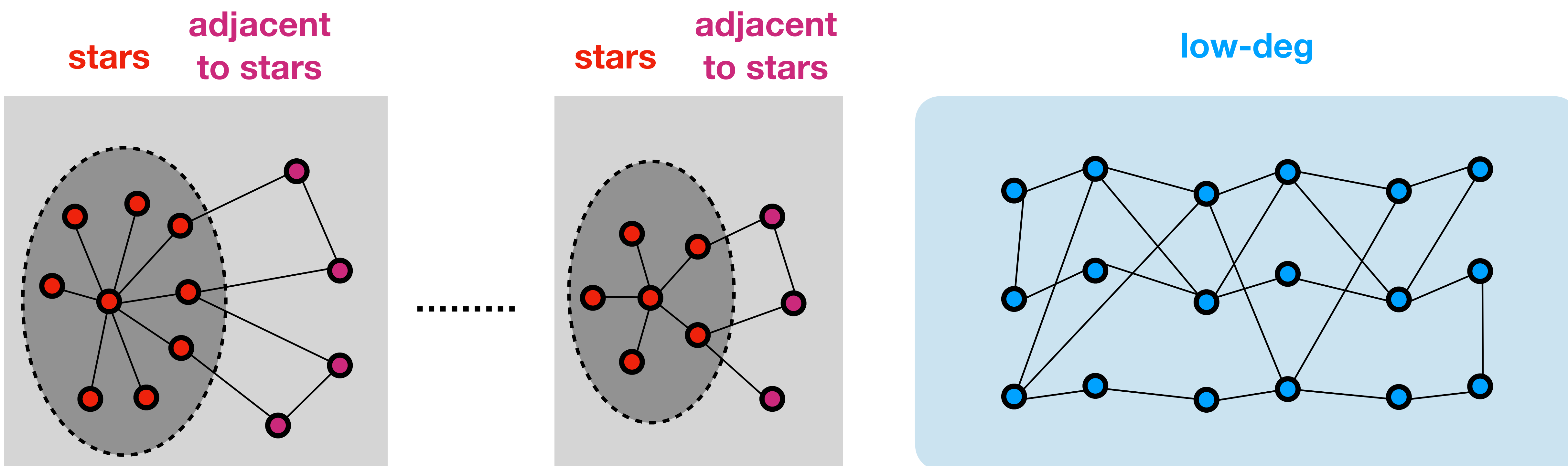
$\deg \leq n$

low-deg



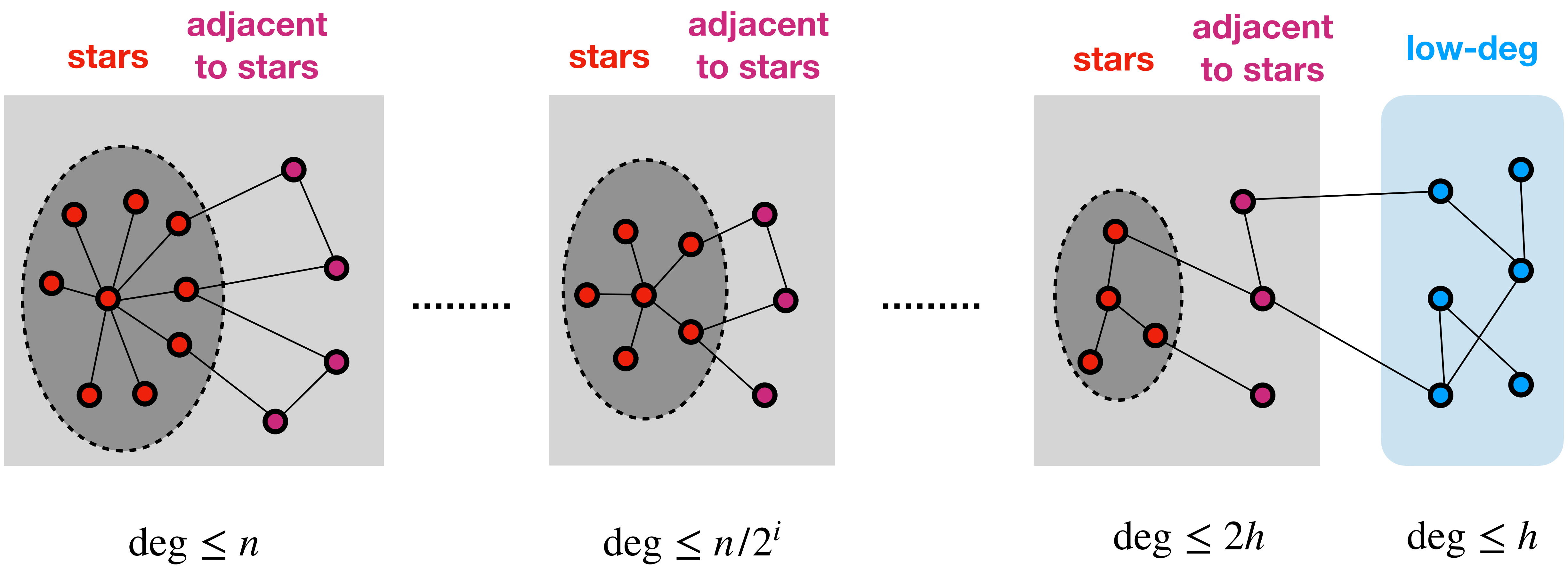
$\deg \leq n/2$

A multi-layer approach

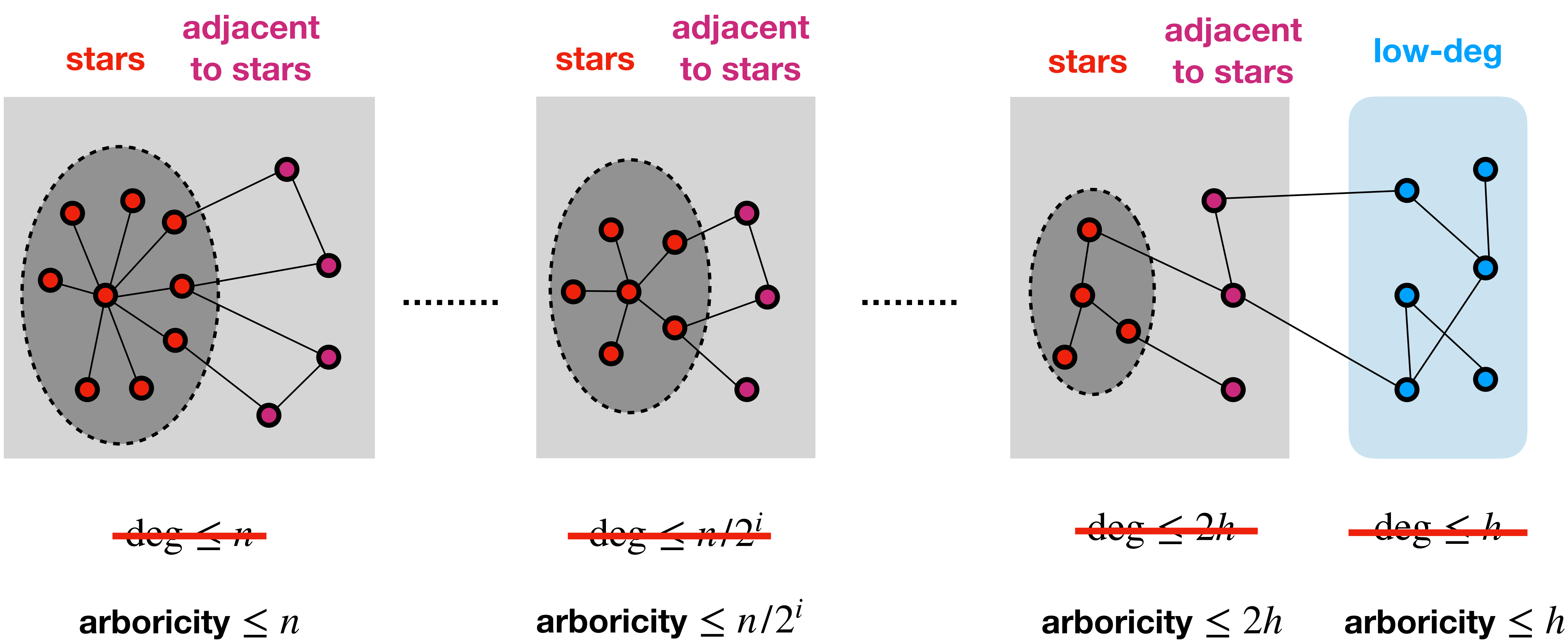


$\deg \leq n$ .... $\deg \leq n/2^i$ $\deg \leq n/2^{i+1}$

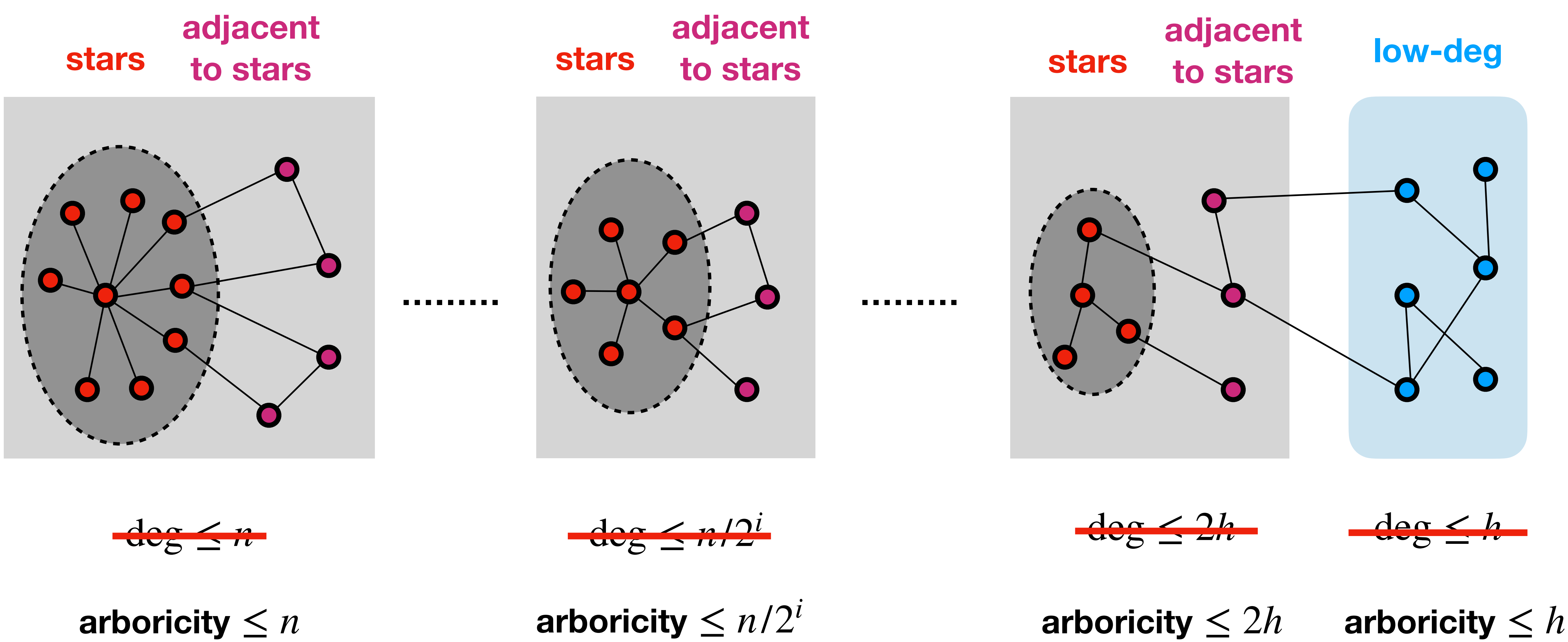
A multi-layer approach



A multi-layer approach

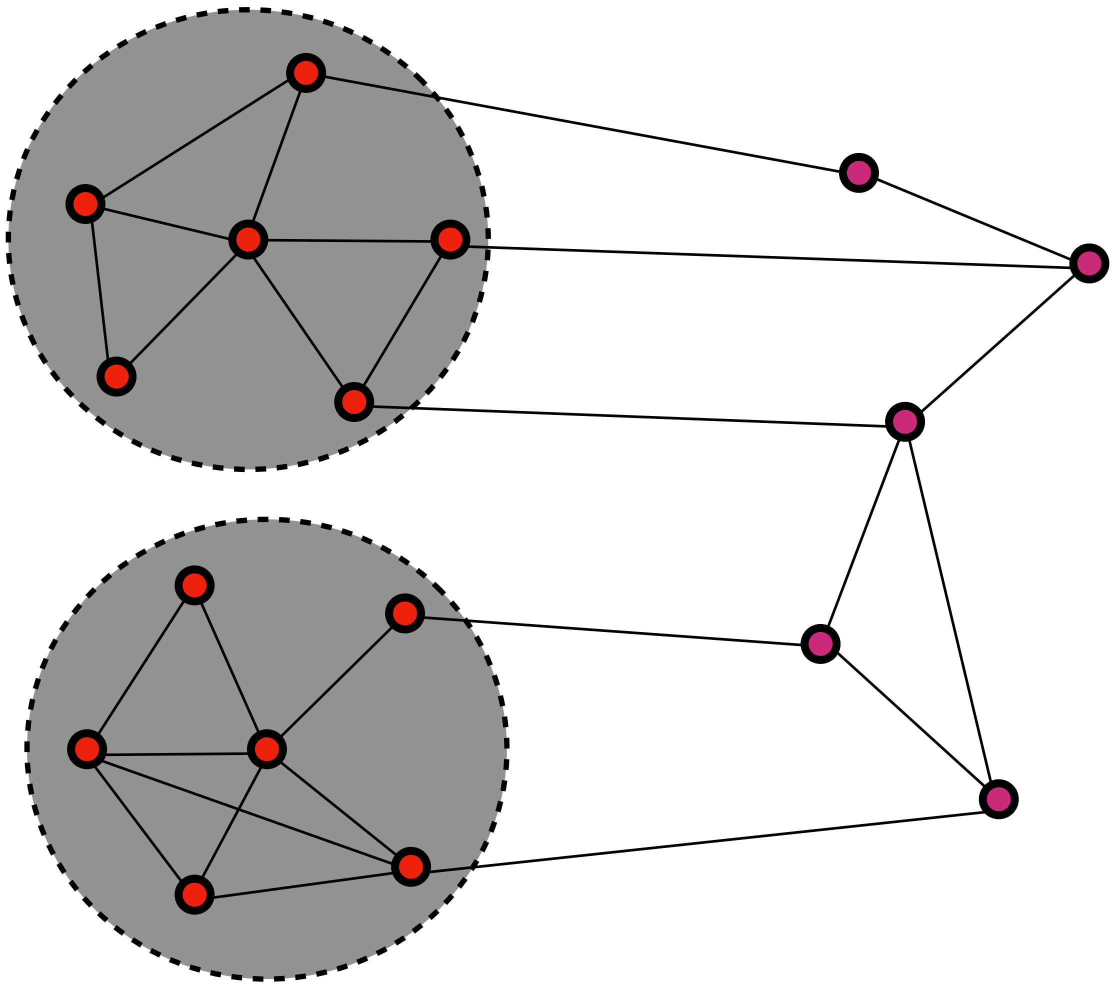


A multi-layer approach



How to utilize low arboricity?

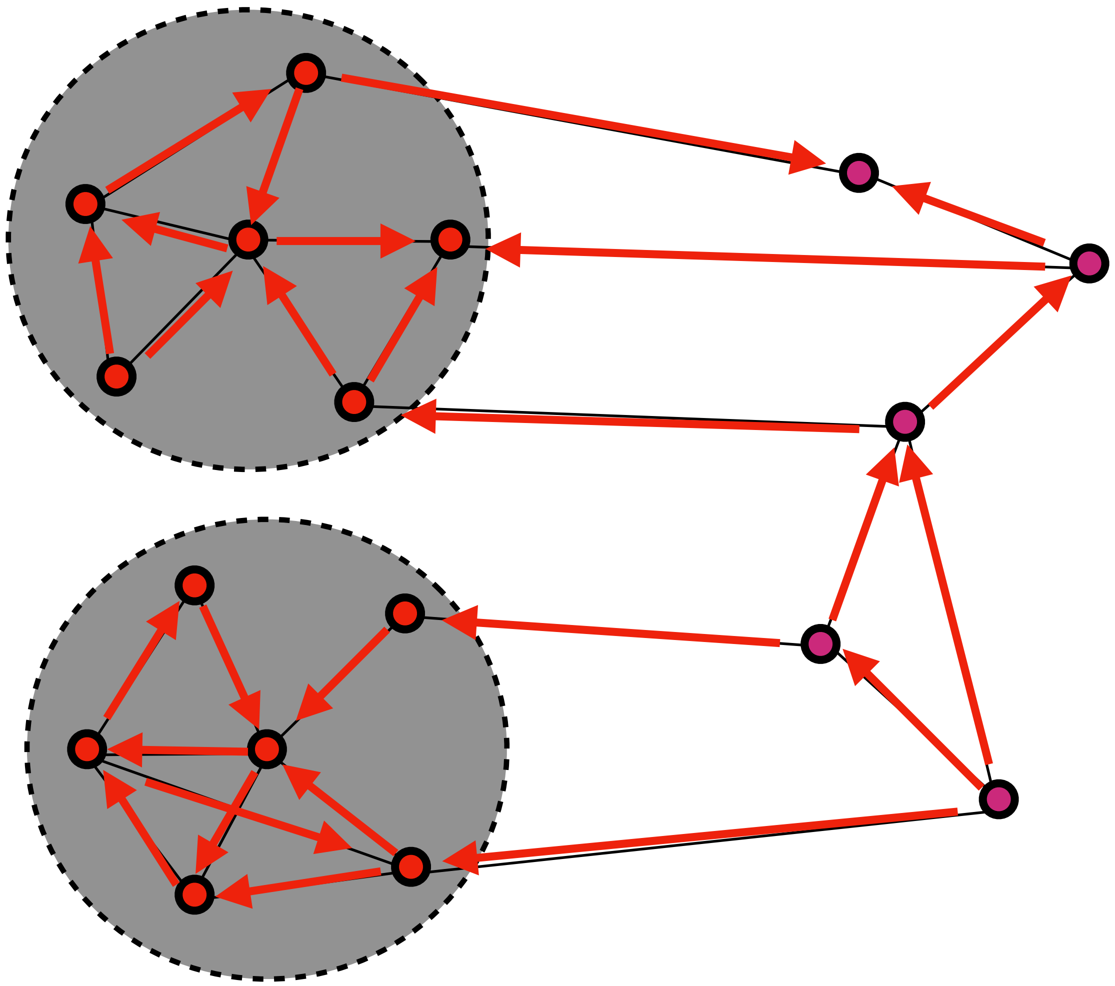
Low arboricity



star of size $\geq n/2^i$

adjacent to stars

Low arboricity

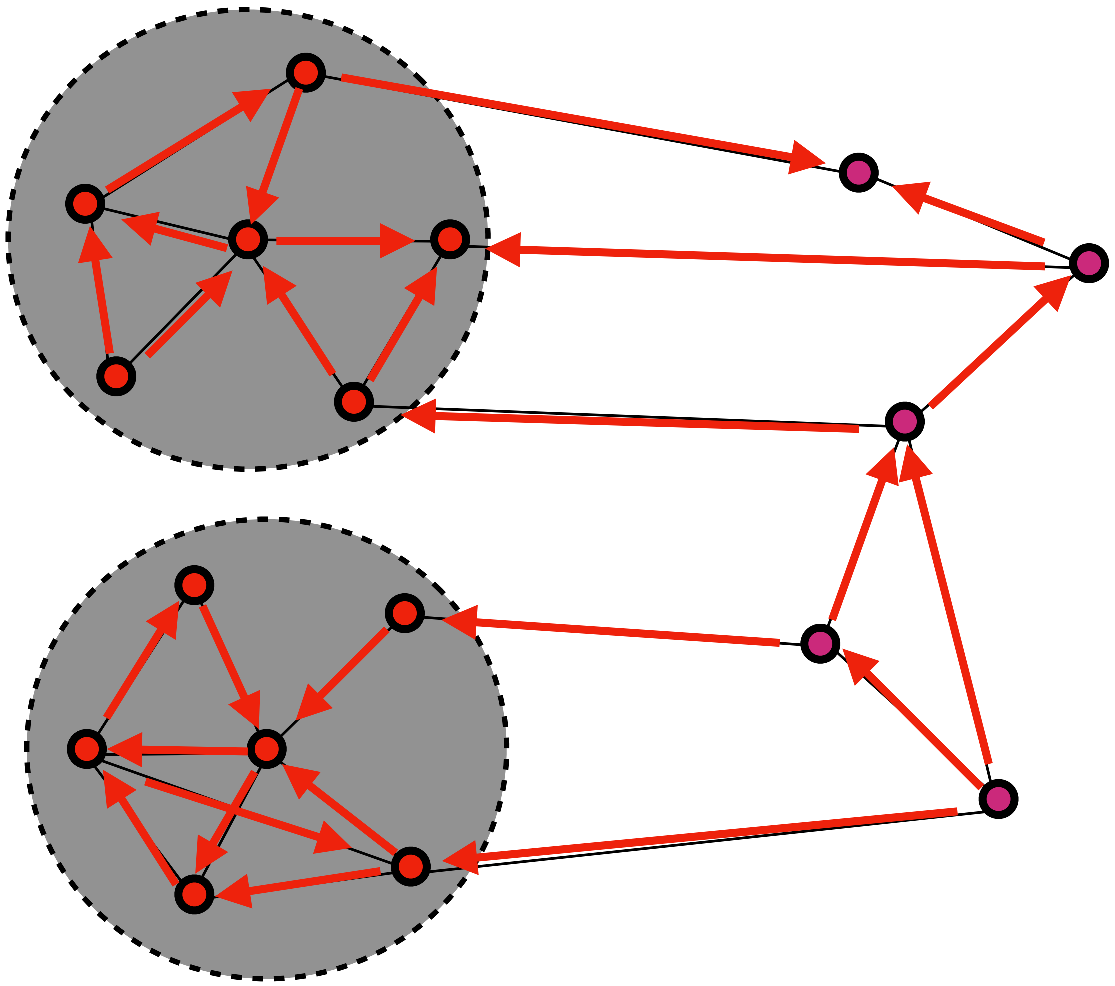


star of size $\geq n/2^i$

adjacent to stars

Edge orientation such that out-degrees $\leq n/2^{i-1}$

Low arboricity



Two kinds of operations:

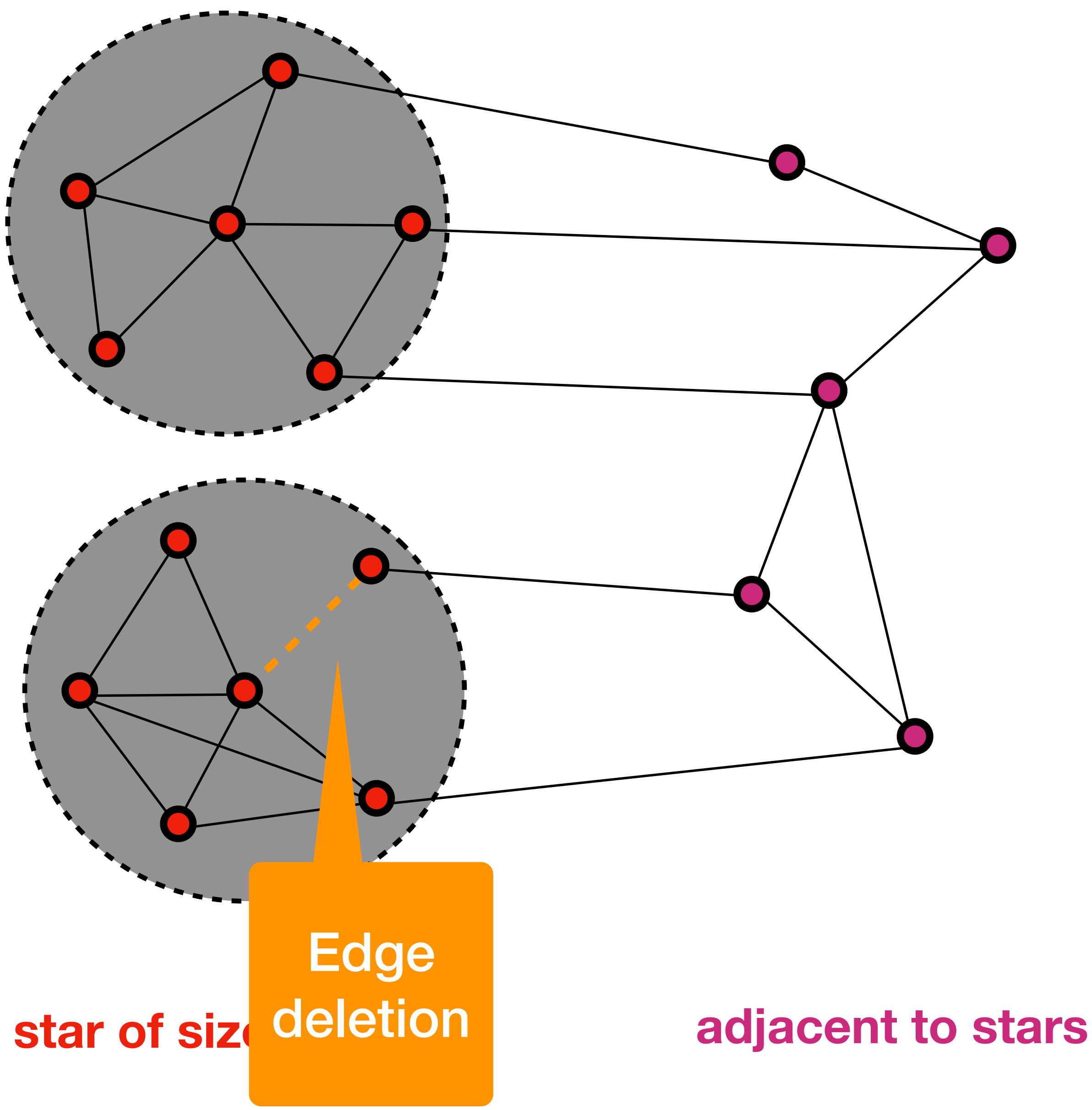
- Turn red vertices into purple vertices
- Find stars among purple vertices, and turn them red ones

star of size $\geq n/2^i$

adjacent to stars

Edge orientation such that out-degrees $\leq n/2^{i-1}$

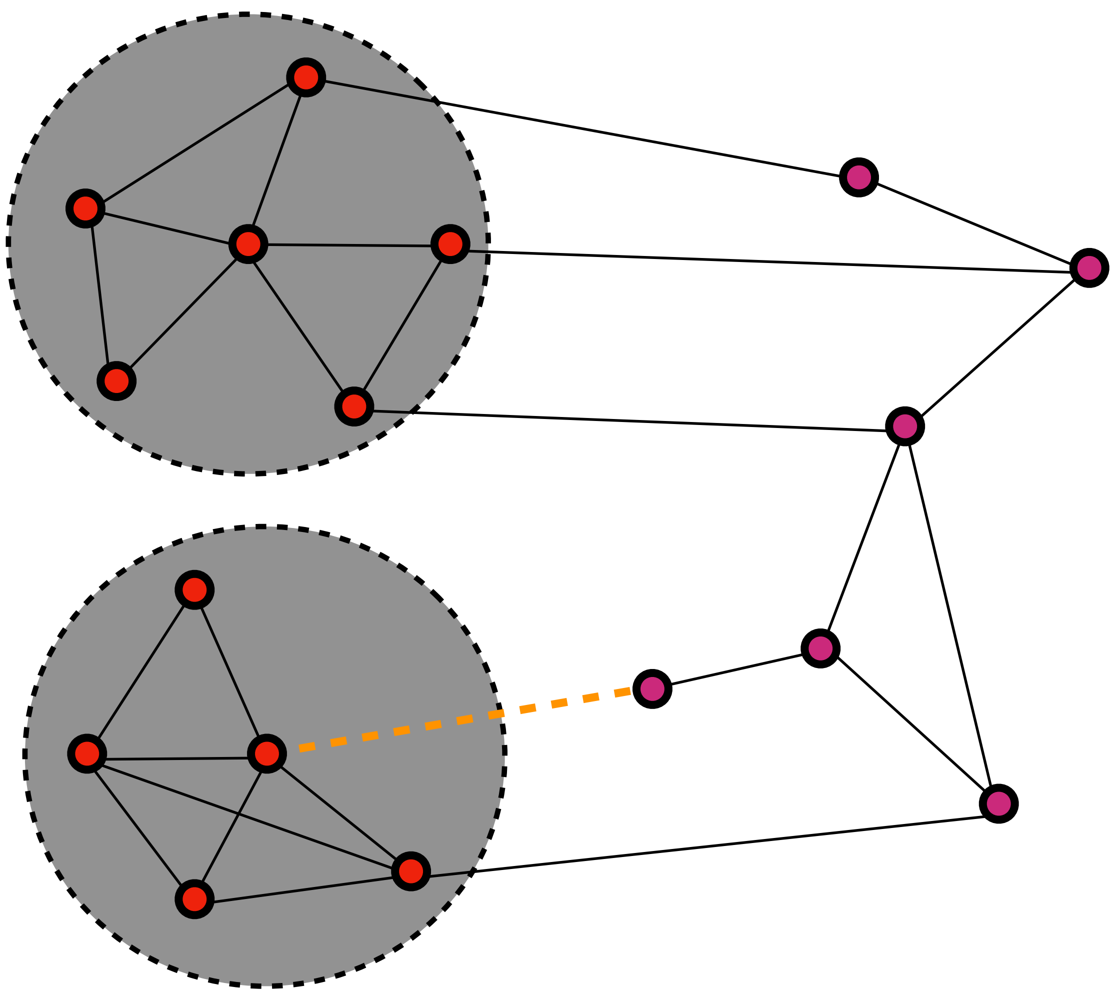
Low arboricity



Two kinds of operations:

- Turn red vertices into purple vertices
- Find stars among purple vertices, and turn them red ones

Low arboricity



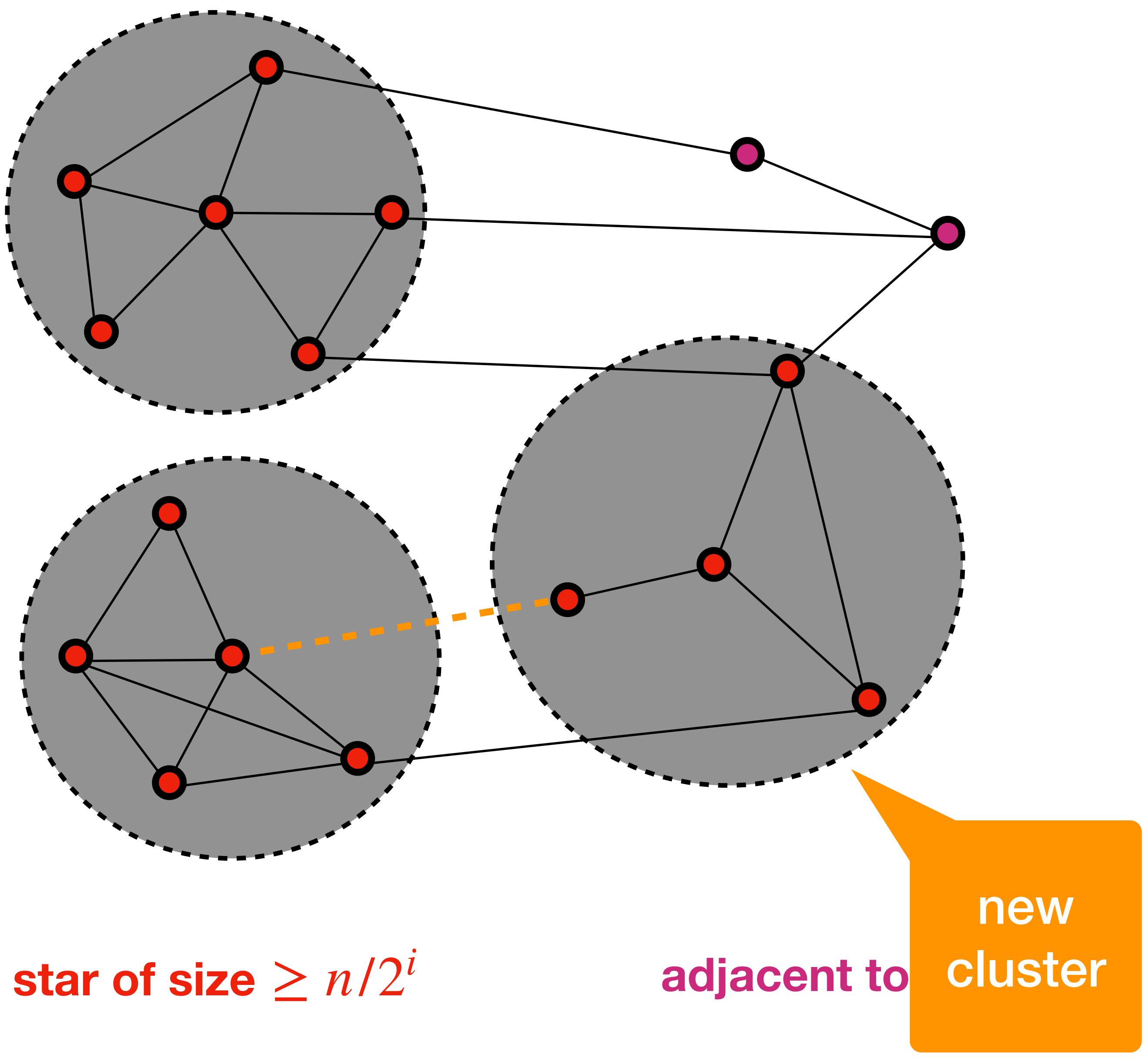
star of size $\geq n/2^i$

adjacent to stars

Two kinds of operations:

- Turn red vertices into purple vertices
- Find stars among purple vertices, and turn them red ones

Low arboricity



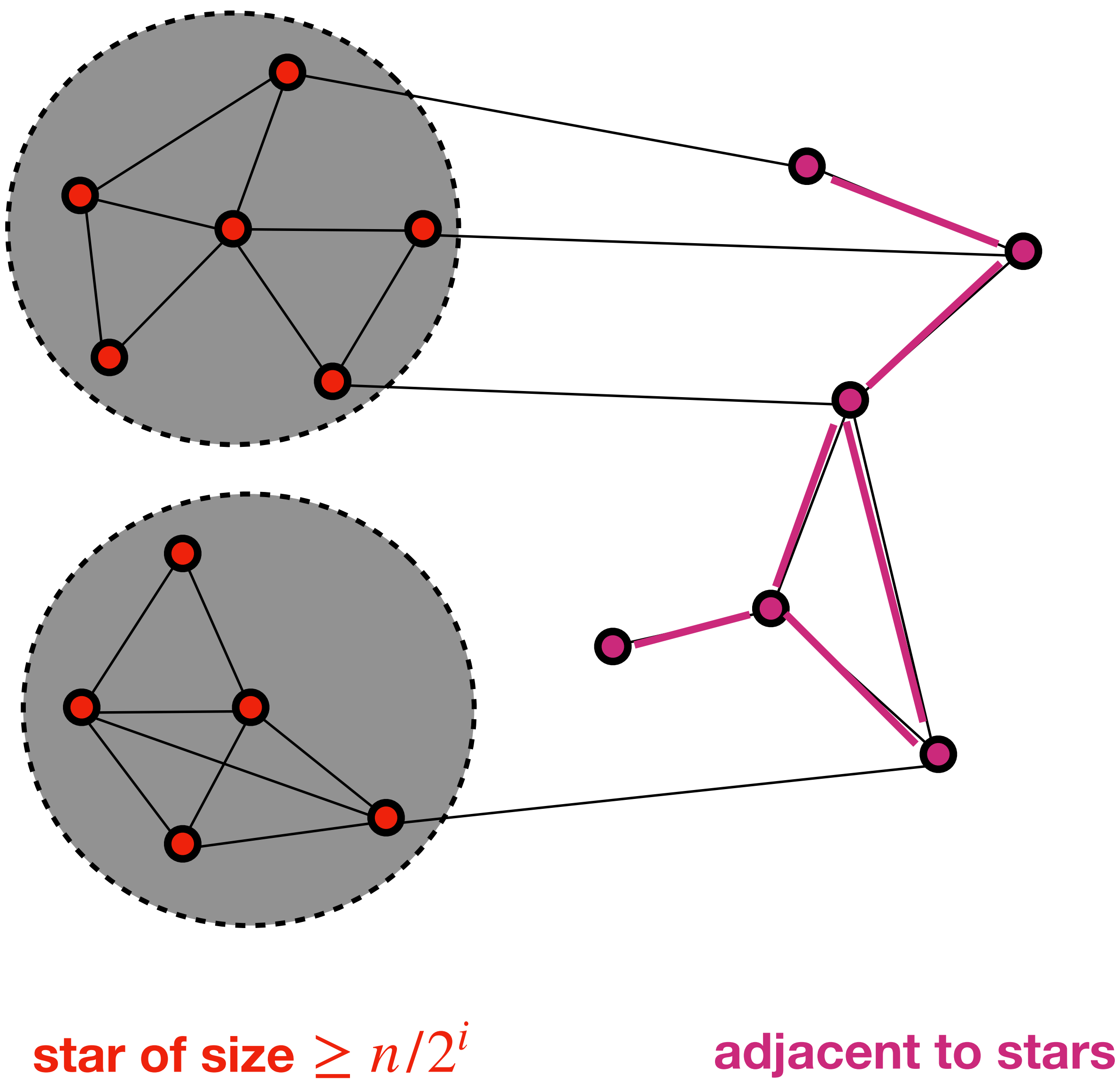
star of size $\geq n/2^i$

adjacent to

Two kinds of operations:

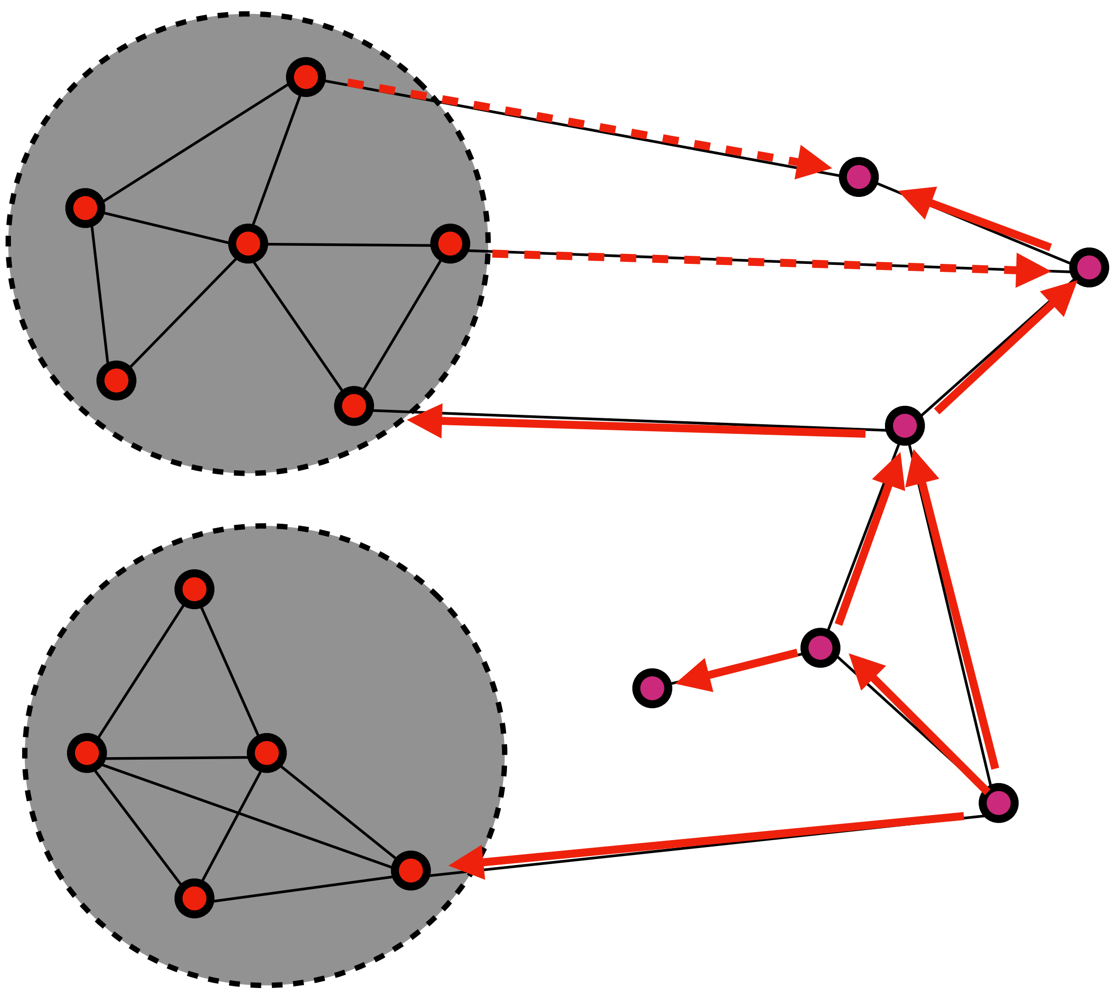
- Turn red vertices into purple vertices
- Find stars among purple vertices, and turn them red ones

Low arboricity



- To detect large stars, need to explicitly store the induced subgraph on purple vertices
- When **red** —> **purple**
  Need to scan adjacency-list to find all purple neighbors which is costly

Low arboricity

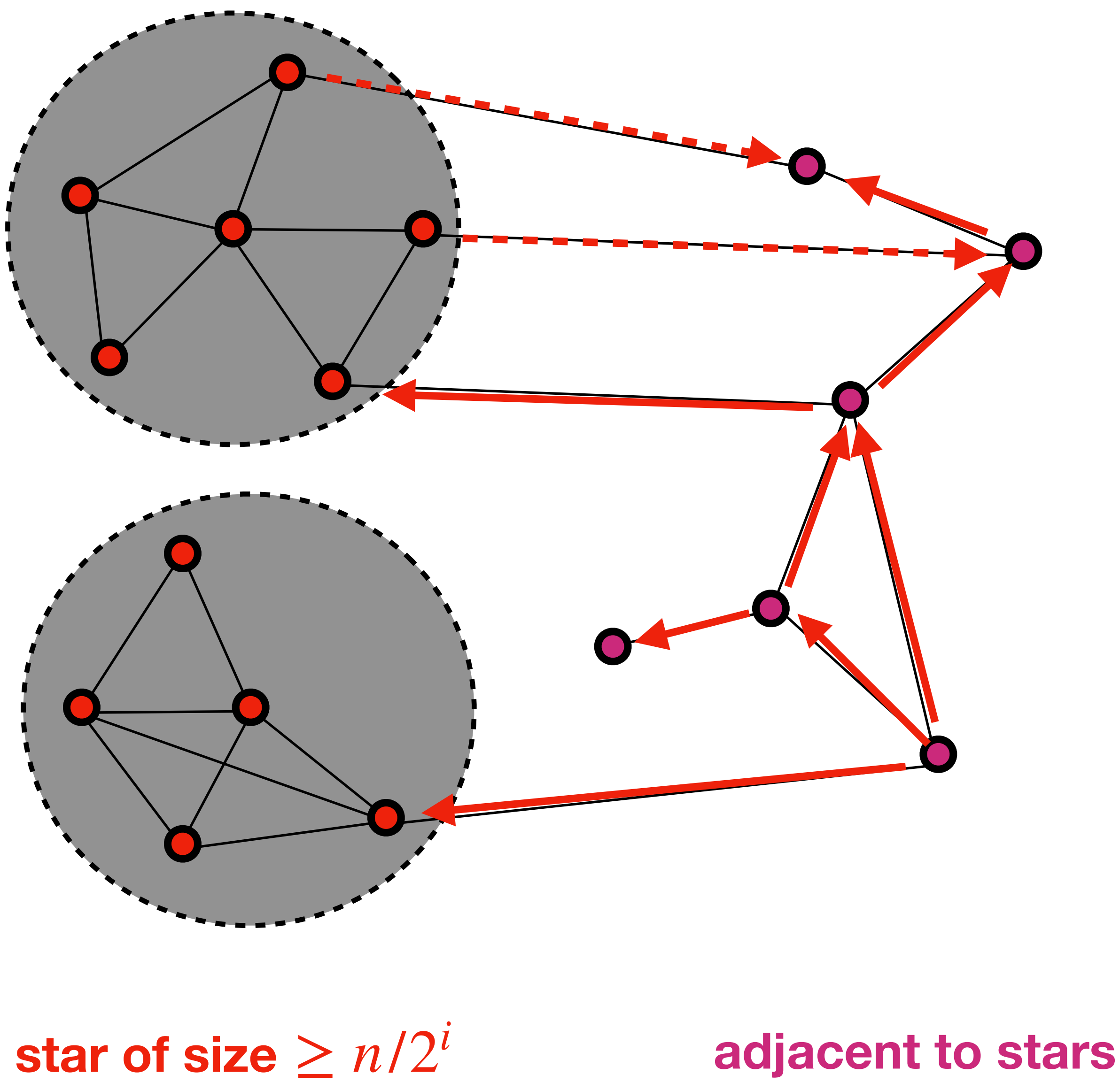


star of size $\geq n/2^i$

adjacent to stars

- When **red** —> **purple**
  Need to scan adjacency-list to find all purple neighbors which is costly
- **Arboricity helps!**
- **Purple** vertices store **in/out-**purple neighbors
- **Red** vertices only store **in-**purple neighbors

Low arboricity



- When **red** —> **purple**
  Need to scan adjacency-list to find all purple neighbors which is costly
- **Arboricity helps!**
- **Purple** vertices store **in/out-**purple neighbors
- **Red** vertices only store **in-**purple neighbors
- When **red** <—> **purple**
  Only need to scan out-neighbors, which is at most $n/2^{i-1}$

Further questions

- How about deterministic max-flow in multi-graphs?
- …. in weighted graphs?

Further questions

- How about deterministic max-flow in multi-graphs?
- …. in weighted graphs?

Thanks for listening