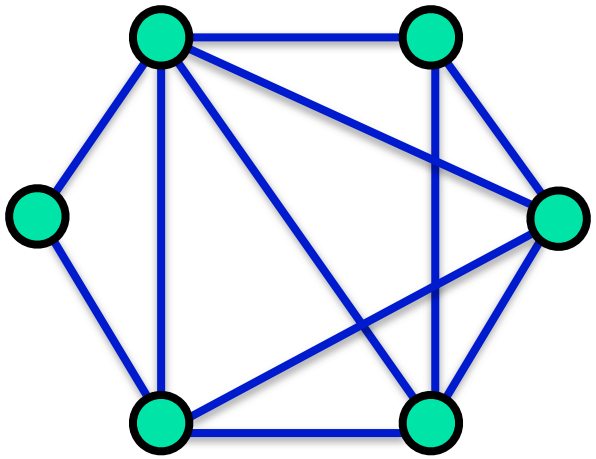
# (More) Recent Advances in Cut Query Algorithms

Aaron (Louie) Putterman
Harvard University

# Recap: Cut-Query Model

Given a graph $G = (V, E)$, can query the cut oracle with any set $S \subseteq V$

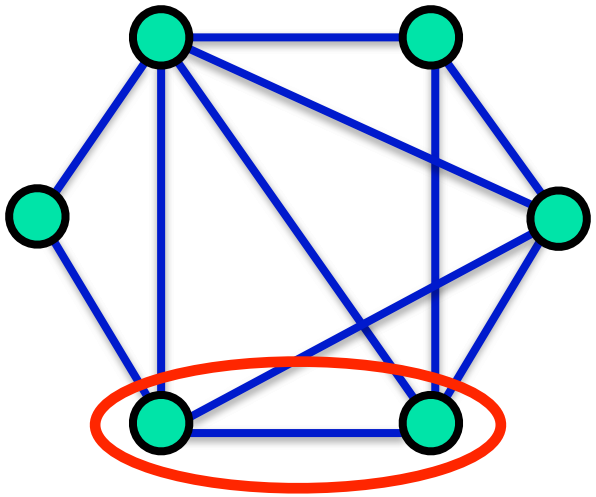
Oracle reports the *weight* of edges cut by $S$



# Recap: Cut-Query Model

Given a graph $G = (V, E)$, can query the cut oracle with any set $S \subseteq V$

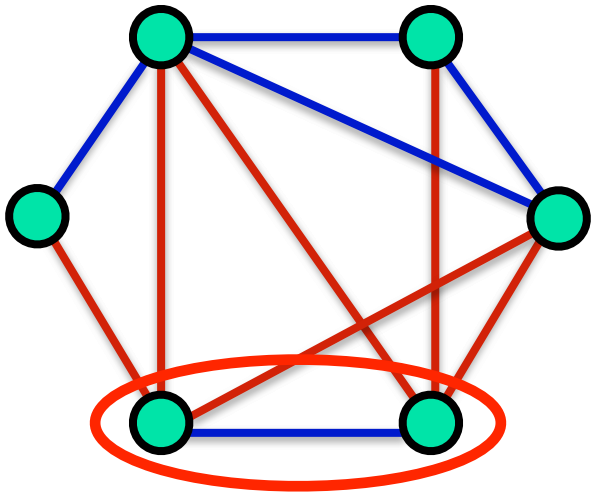
Oracle reports the *weight* of edges cut by $S$



# Recap: Cut-Query Model

Given a graph $G = (V, E)$, can query the cut oracle with any set $S \subseteq V$

Oracle reports the *weight* of edges cut by $S$



# Recap: Cut-Query Model

Given a graph $G = (V, E)$, can query the cut oracle with any set $S \subseteq V$

Oracle reports the *weight* of edges cut by $S$

**Guiding Question:** When can we learn graph properties with a small number of cut queries?

# This Talk

Will mostly discuss the pioneering work of Rubinstein, Schramm, and Weinberg [2018]:

1. Building $(1 \pm \epsilon)$ cut-sparsifiers in $\tilde{O}(n)$ cut queries.
2. Computing the global min-cut of a graph in $\tilde{O}(n)$ cut queries.
3. Computing an s-t min-cut of a graph in $\tilde{O}(n^{5/3})$ cut queries.

# This Talk

Will mostly discuss the pioneering work of Rubinstein, Schramm, and Weinberg [2018]:

1. Building $(1 \pm \epsilon)$ cut-sparsifiers in $\tilde{O}(n)$ cut queries.
2. Computing the global min-cut of a graph in $\tilde{O}(n)$ cut queries.
3. Computing an s-t min-cut of a graph in $\tilde{O}(n^{5/3})$ cut queries.

First key insight: can randomly sample edges in the cut query model!

# Randomly Sampling Edges

Given: graph $G = (V, E)$

Output: random edge $e \in E$

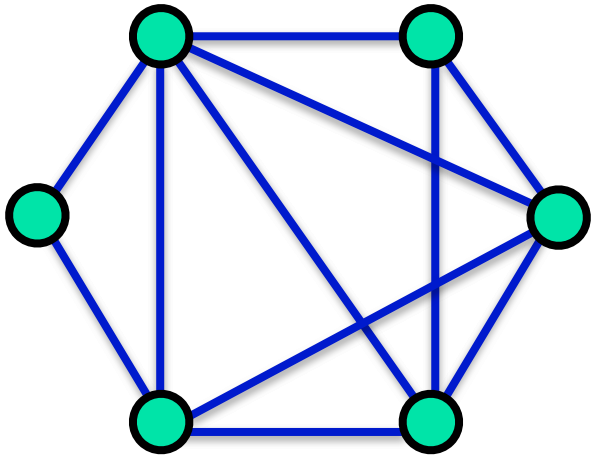
**Algorithm**:

1. Compute all vertex degrees
2. Sample $v \in V$ proportional to degree
3. Let $S = V - \{v\}$
4. Split $S$ into equal parts $S_1, S_2$
5. Compute $|E[v, S_1]|$ and $|E[v, S_2]|$
6. Recurse with $S = S_i$ w.p. proportional to $|E[v, S_i]|$
7. Terminate when $|S| = 1$

# Randomly Sampling Edges

Given: graph $G = (V, E)$

Output: random edge $e \in E$

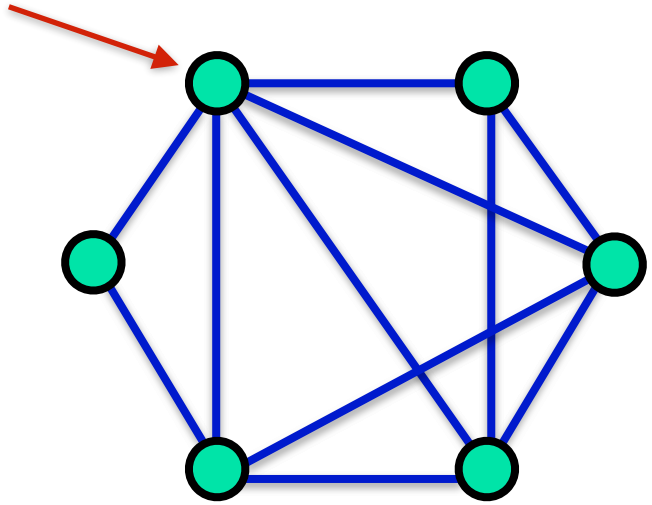


# Randomly Sampling Edges

Given: graph $G = (V, E)$

Output: random edge $e \in E$

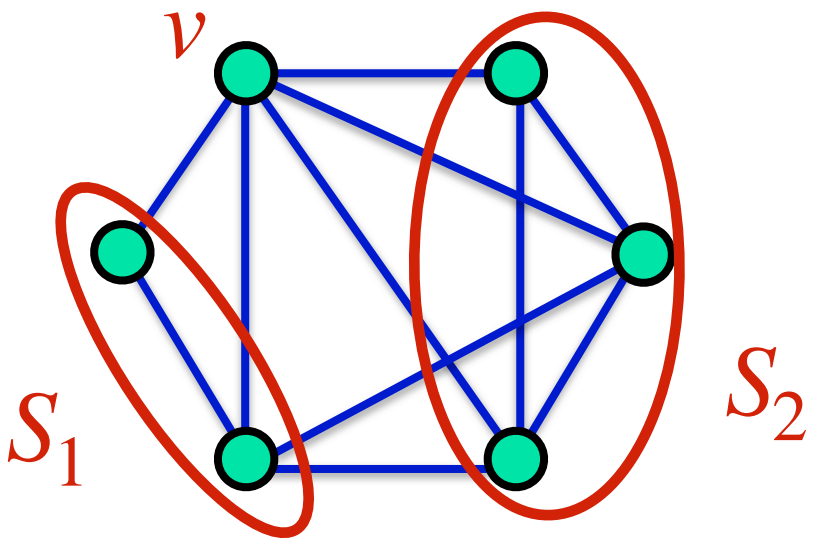
Choose $v$ proportional to degree



# Randomly Sampling Edges

Given: graph $G = (V, E)$

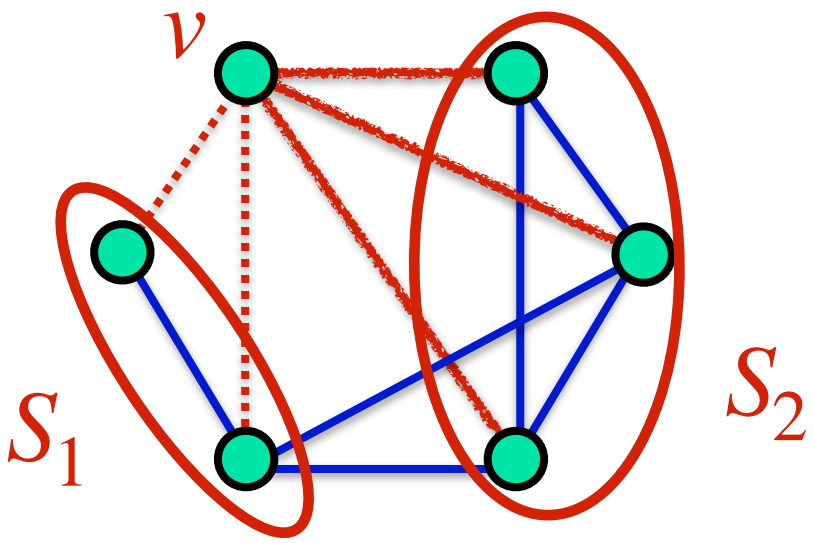
Output: random edge $e \in E$



# Randomly Sampling Edges

Given: graph $G = (V, E)$

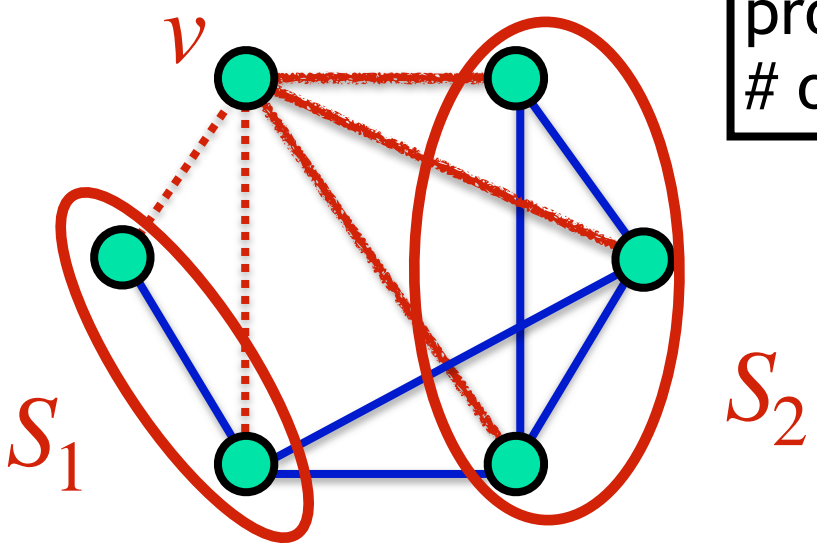
Output: random edge $e \in E$



# Randomly Sampling Edges

Given: graph $G = (V, E)$

Output: random edge $e \in E$

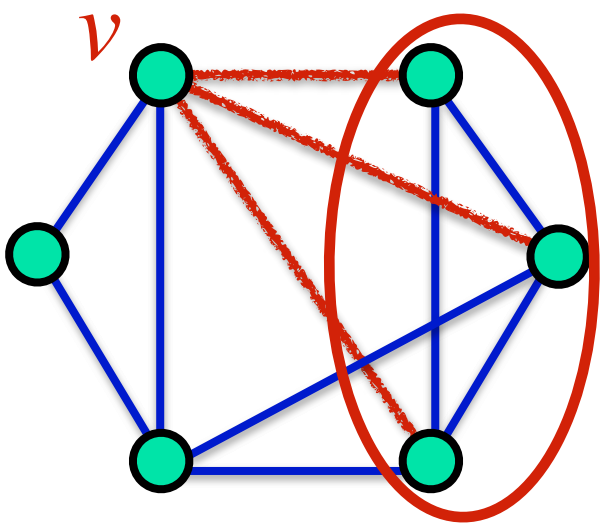



Recurse proportional to # of edges

# Randomly Sampling Edges

Given: graph $G = (V, E)$

Output: random edge $e \in E$

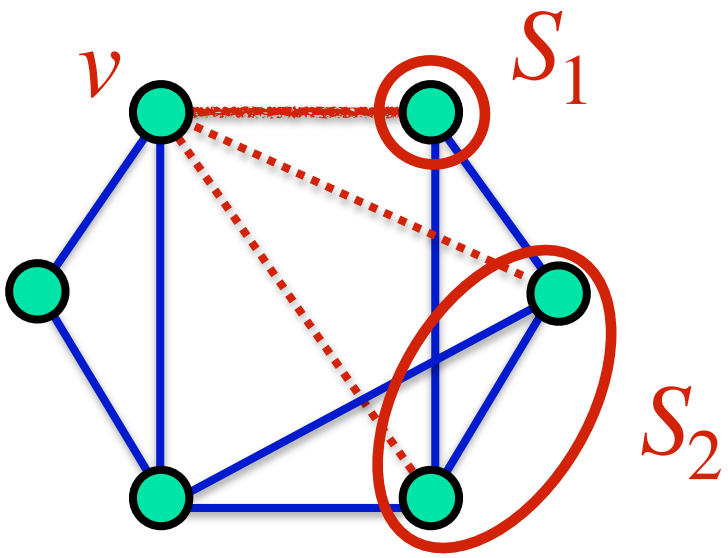



Recurse proportional to # of edges

# Randomly Sampling Edges

Given: graph $G = (V, E)$

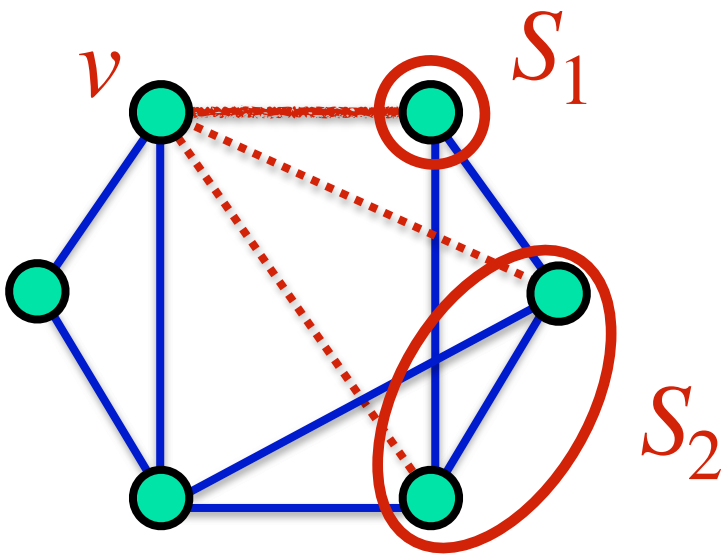
Output: random edge $e \in E$



# Randomly Sampling Edges

Given: graph $G = (V, E)$

Output: random edge $e \in E$




Claim: Once you know all vertex degrees, randomly sampling an edge requires only $O(\log(n))$ many cut queries.

# Global Min-Cut

# Randomly Sampling Edges

**Goal**: given a graph $G = (V, E)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

# Randomly Sampling Edges

**Goal**: given a graph $G = (V, E)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

**Intuition**:

- Random sampling allows us to simulate random contraction of edges

# Randomly Sampling Edges

**Goal**: given a graph $G = (V, E)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

**Intuition**:

- Random sampling allows us to simulate random contraction of edges
- When the contracted graph has few edges, can *learn* the entire graph and compute min-cut explicitly!

# Warm-Up: Small Min-Cuts

**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

**Algorithm**:

1. Initialize contracted components:

$C_1 = \{v_1\}, \ldots C_n = \{v_n\}$

# Warm-Up: Small Min-Cuts

**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

**Algorithm**:

1. Initialize contracted components:

$C_1 = \{v_1\}, \ldots C_n = \{v_n\}$

2. While sum of degrees is $\geq 2n \log(n)$, sample a random edge $(u, v)$, with $u \in C_i, v \in C_j$ and merge $C_i, C_j$

# Warm-Up: Small Min-Cuts

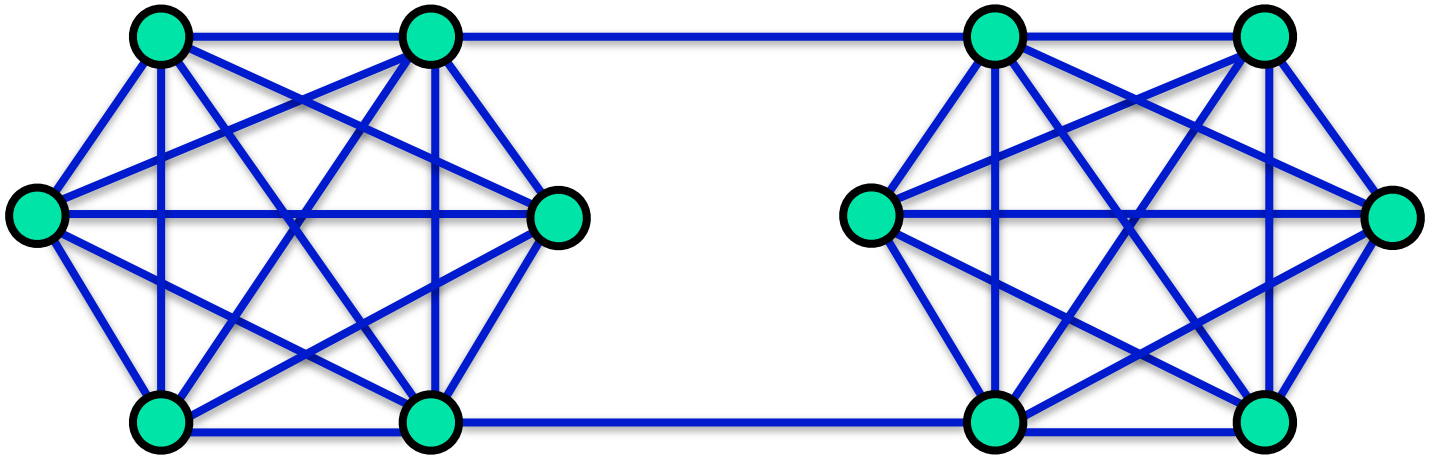
**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

**Algorithm**:

1. Initialize contracted components:
$C_1 = \{v_1\}, \ldots C_n = \{v_n\}$
2. While sum of degrees is $\geq 2n \log(n)$, sample a random edge $(u, v)$, with $u \in C_i, v \in C_j$ and merge $C_i, C_j$
3. Now learn all the edges in $G$ between the contracted components $C_1, \ldots C_\ell$
4. Output Min-Cut of $G[C_1, \ldots C_\ell]$

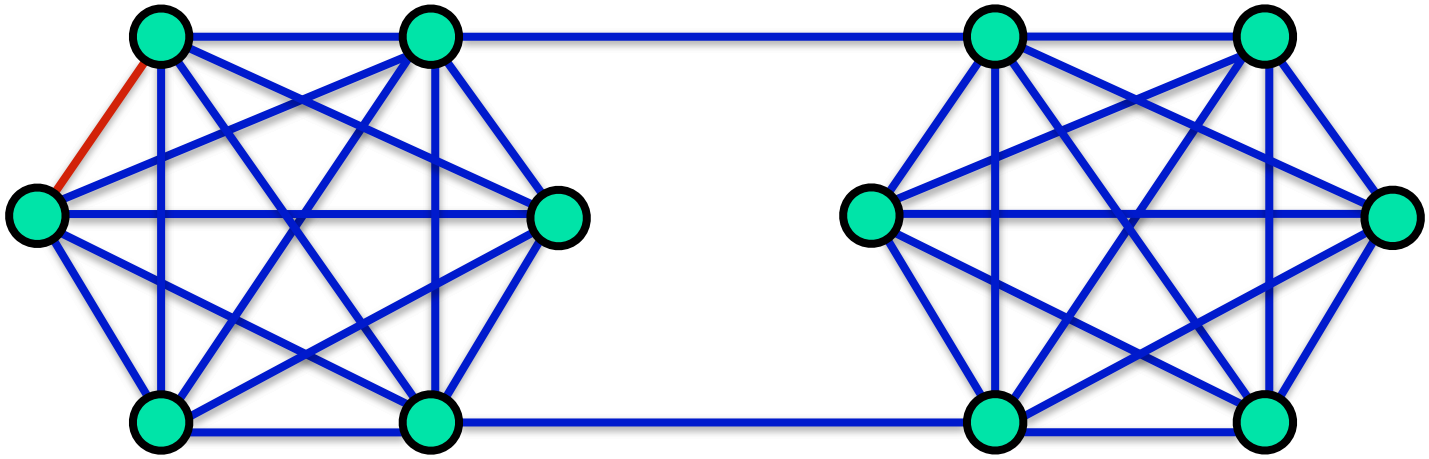
# Warm-Up: Small Min-Cuts

**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)



# Warm-Up: Small Min-Cuts

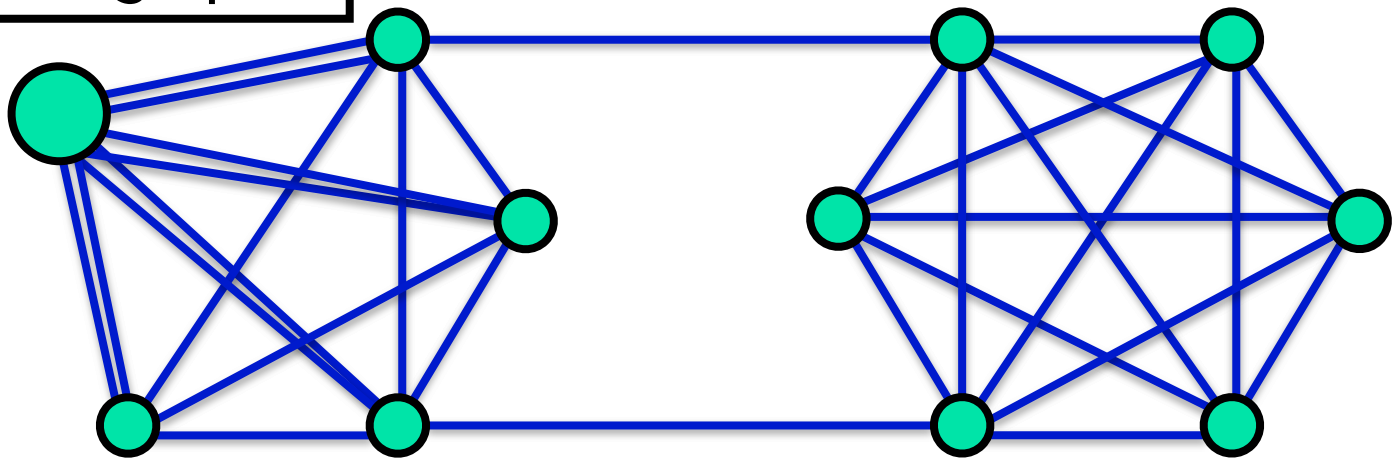
**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)



# Warm-Up: Small Min-Cuts

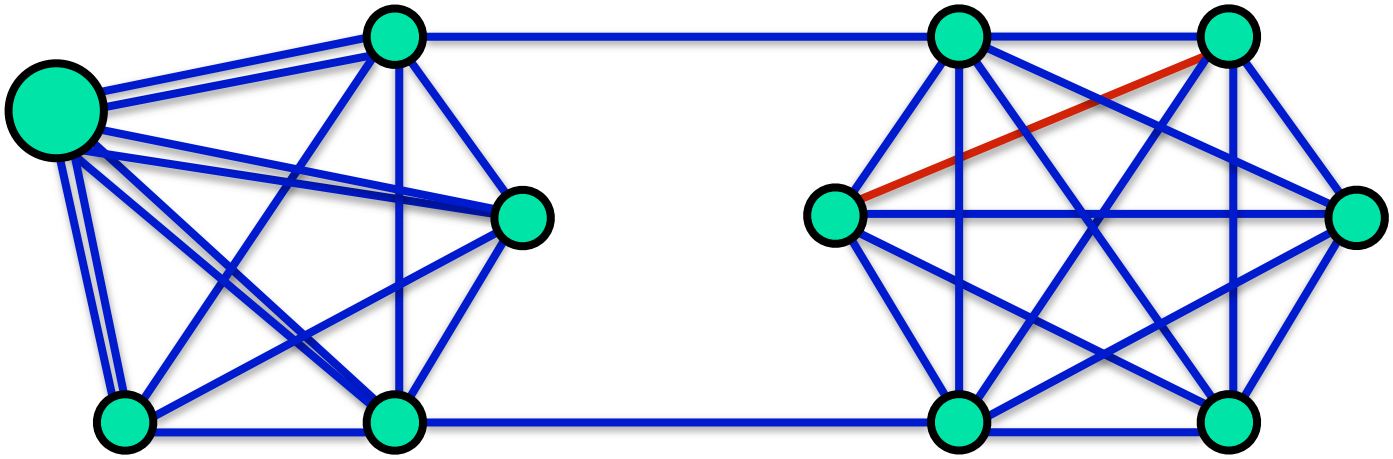
**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

Easy to simulate cut queries in contracted graph!



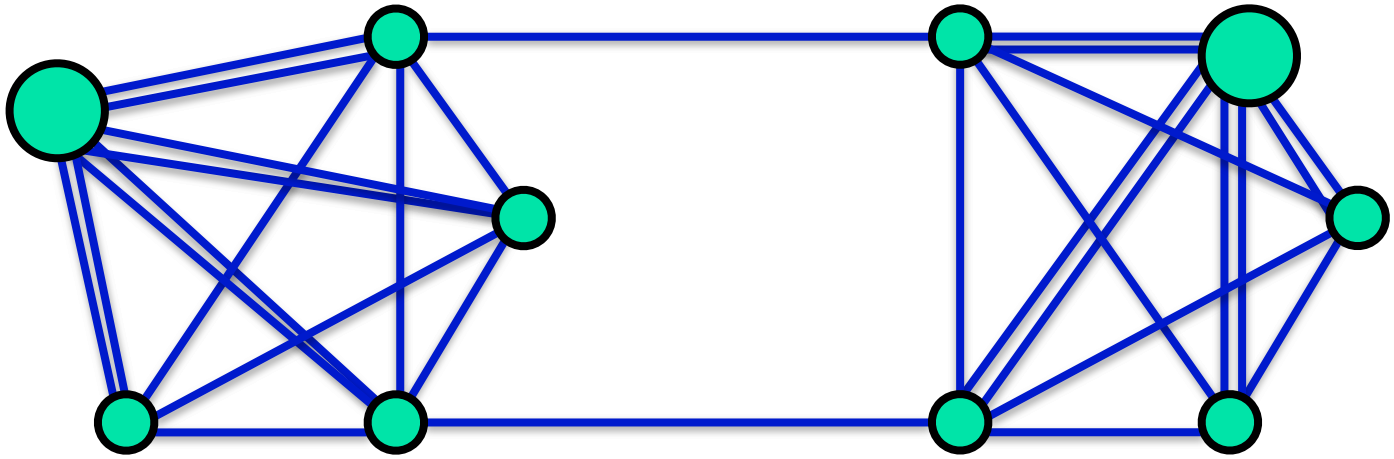
# Warm-Up: Small Min-Cuts

**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)



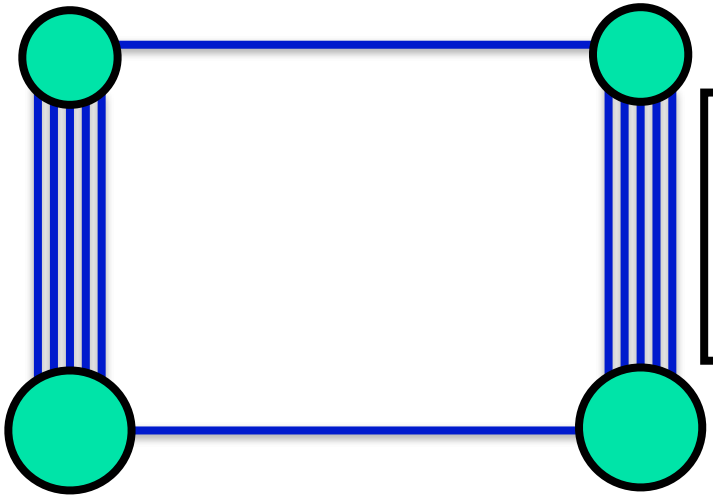
# Warm-Up: Small Min-Cuts

**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)



# Warm-Up: Small Min-Cuts

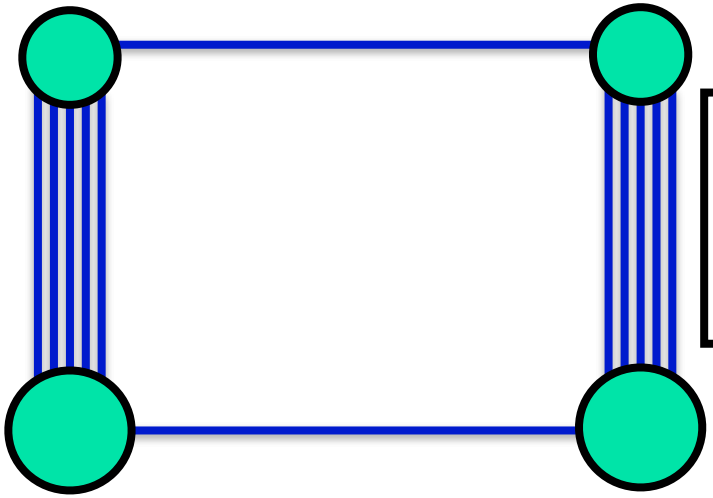
**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)



Stop when $n \log(n)$ edges remain

# Warm-Up: Small Min-Cuts

**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)



Now you can *learn* this entire graph via cut queries!

# Warm-Up: Small Min-Cuts

**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

Must show two things:

Claim: Minimum cut of $G$ survives the contractions (w.h.p.).

Claim: Total number of cut queries is bounded by $\tilde{O}(n)$.

# Warm-Up: Small Min-Cuts

**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

Must show two things:

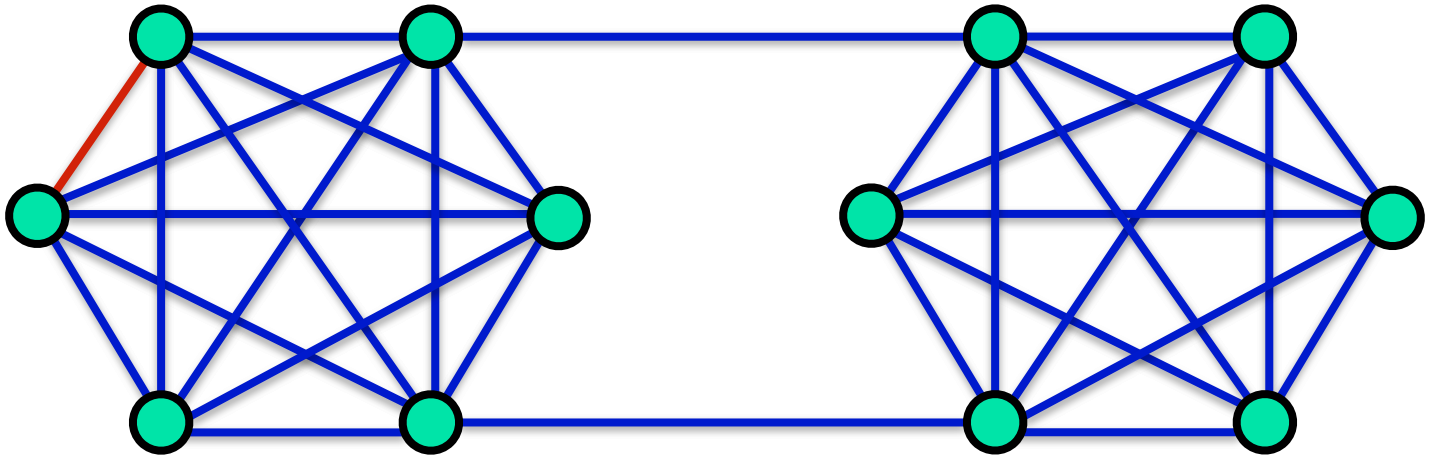
Claim: Minimum cut of $G$ survives the contractions (w.h.p.).

Claim: Total number of cut queries is bounded by $\tilde{O}(n)$.

# Warm-Up: Small Min-Cuts

**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

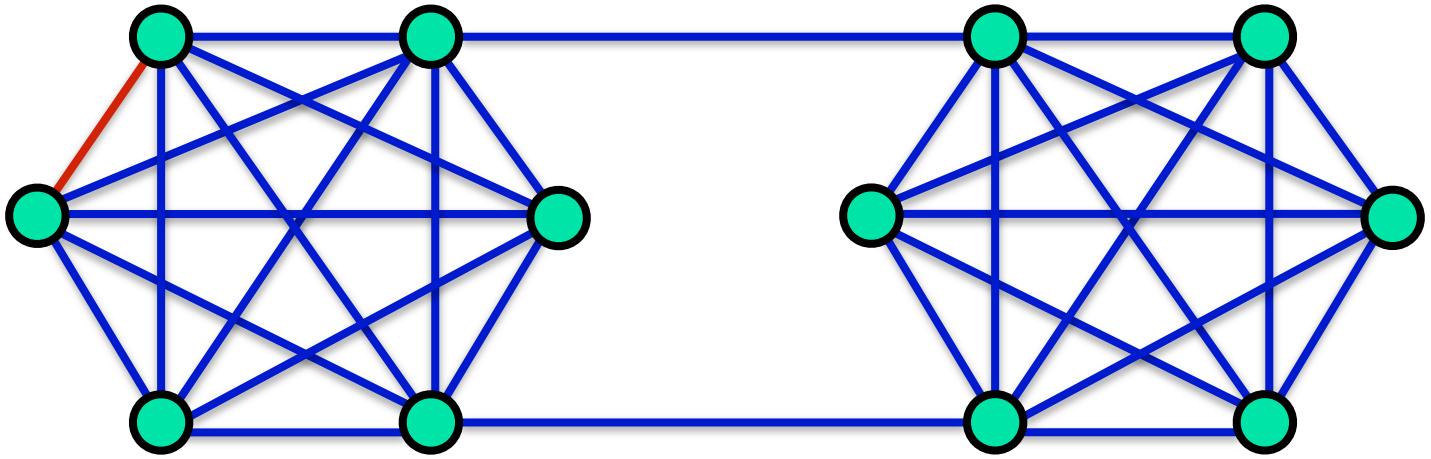
Min-cut survives as long as no edge inside it is contracted



# Warm-Up: Small Min-Cuts

**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

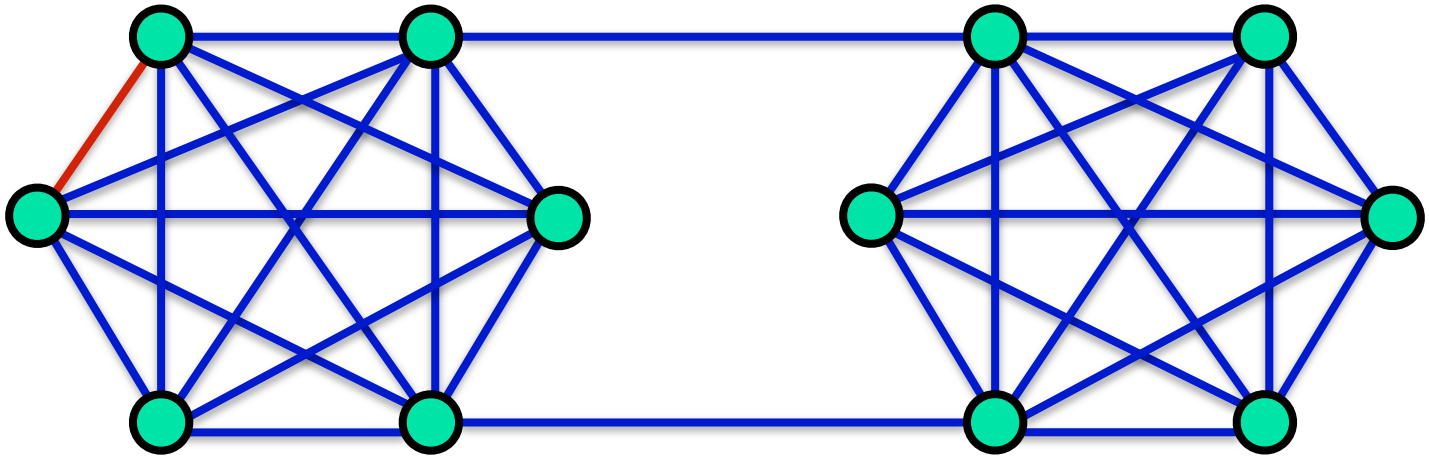
Probability survives one contraction $\geq 1 - \log(n)/(n\log(n))$



# Warm-Up: Small Min-Cuts

**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

Probability survives all
$\geq (1 - 1/n)^n = \Omega(1)$



# Warm-Up: Small Min-Cuts

**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

Must show two things:

Claim: Minimum cut of $G$ survives the contractions (w.h.p.).

Claim: Total number of cut queries is bounded by $\tilde{O}(n)$.

# Warm-Up: Small Min-Cuts

**Goal**: given a graph $G = (V, E)$ with min-cut $\leq \log(n)$, output the global min-cut with only $\tilde{O}(n)$ many cut queries (w.h.p.)

Must show two things:

Claim: Minimum cut of $G$ survives the contractions (w.h.p.).

Claim: Total number of cut queries is bounded by $\tilde{O}(n)$.

Learning degrees: $O(n)$
Sampling $n$ random edges: $O(n \log(n))$
Learning $n \log(n)$ edge graph: $O(n \log^2(n))$

# Generalizing Beyond Small Min-Cut

To generalize to all min-cut values:

- Guess approx. value of min cut $c \in \{1,2,4,\ldots n\}$
- Run contractions until $c \cdot n$ edges
- Now, *subsample* $n \log(n)$ many edges and further contract any components which appear together in all small cuts
- Finally, learn the contracted graph and compute min-cut!

# Generalizing Beyond Small Min-Cut

To generalize to all min-cut values:

- Guess approx. value of min cut $c \in \{1,2,4,\ldots n\}$
- Run contractions until $c \cdot n$ edges
- Now, *subsample* $n \log(n)$ many edges and further contract any components which appear together in all small cuts
- Finally, learn the contracted graph and compute min-cut!

Gives an $\tilde{O}(n)$ query randomized algorithm for min-cut!

# Some Remarks

Lots of natural questions from here:

Can we do better than $\tilde{O}(n)$ queries?

- Yes! Beautiful work of Apers, Efron, Gawrychowski, Lee, Mukhopadhyay, and Nanongkai [FOCS 2022] gets $O(n)$ randomized queries.

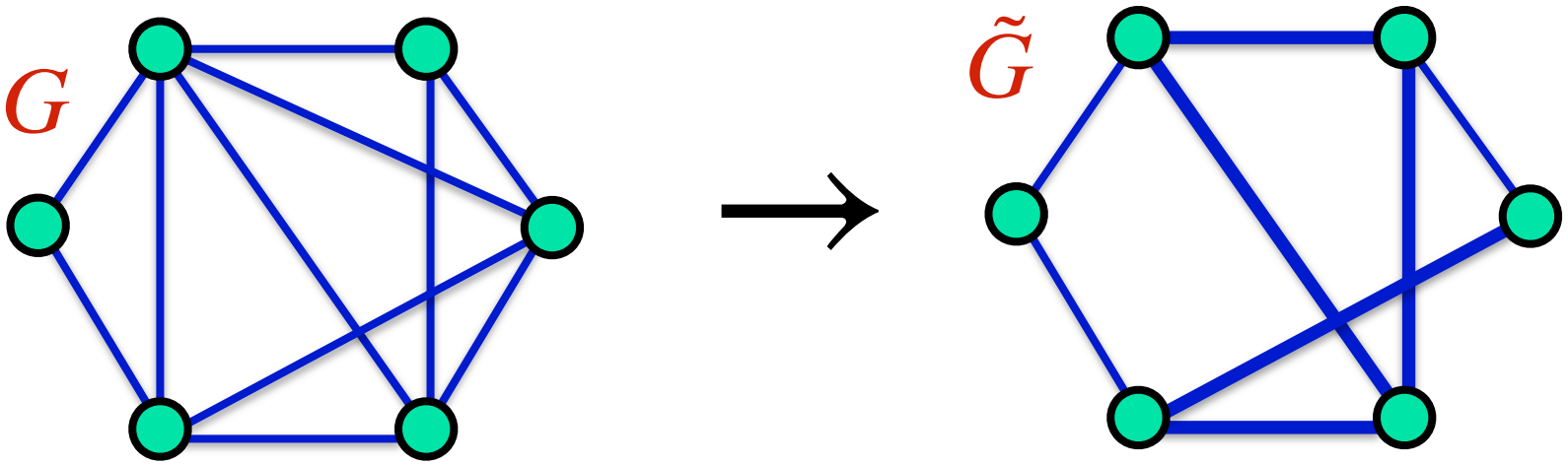
Can we get deterministic algorithms?

- Yes, but $\tilde{O}(n^{5/3})$ queries by Anand, Saranurak, Wang [SODA 2025]. Open question to improve!

# Cut-Sparsifiers

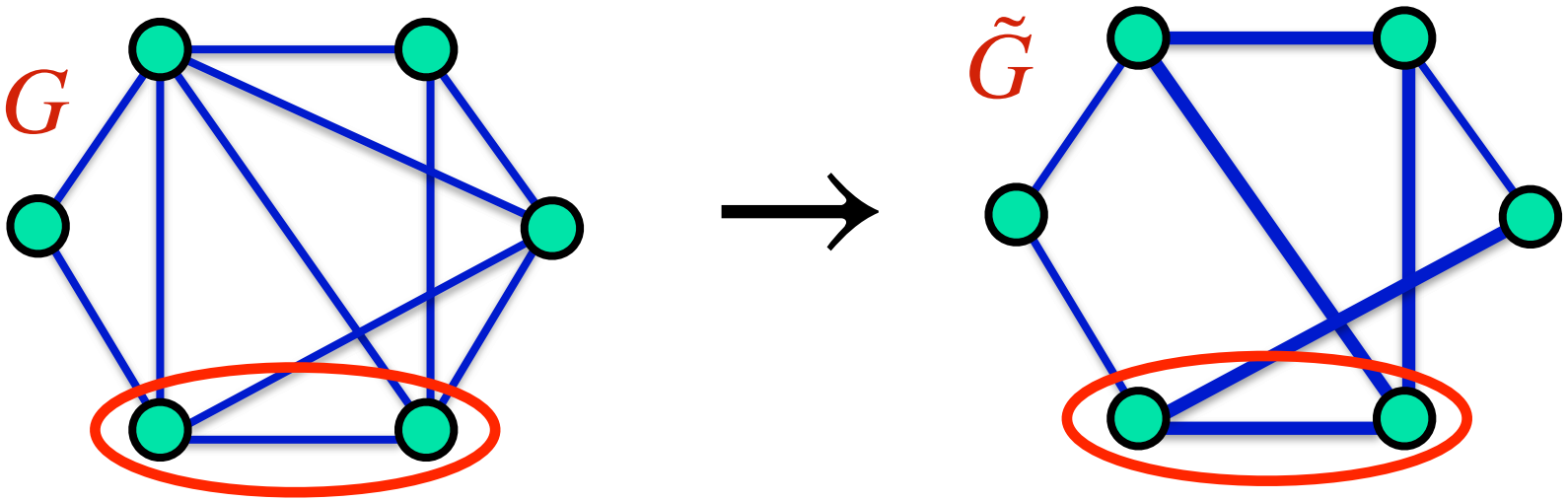
# Cut Sparsifiers

**Definition:** For a graph $G$, a *cut-sparsifier* is a sparse subgraph $\tilde{G}$ which preserves every cut size.



# Cut Sparsifiers

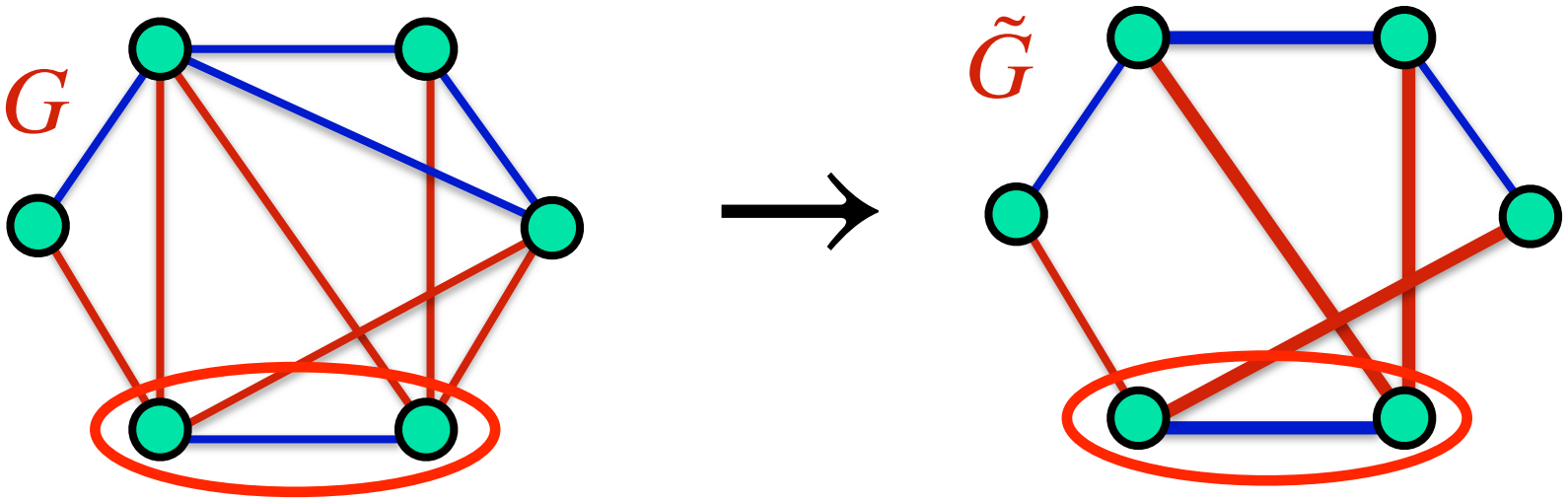
**Definition:** For a graph $G$, a *cut-sparsifier* is a sparse subgraph $\tilde{G}$ which preserves every cut size.



# Cut Sparsifiers

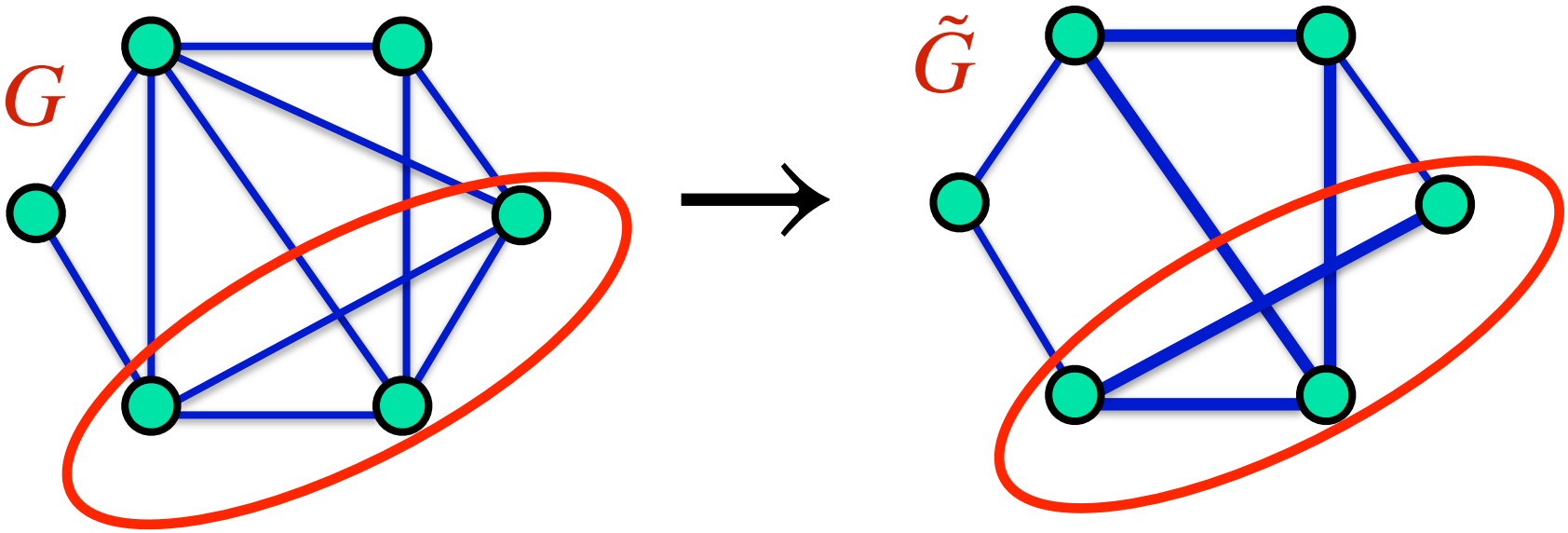
**Definition:** For a graph $G$, a *cut-sparsifier* is a sparse subgraph $\tilde{G}$ which preserves every cut size.

Increase edge weights to compensate



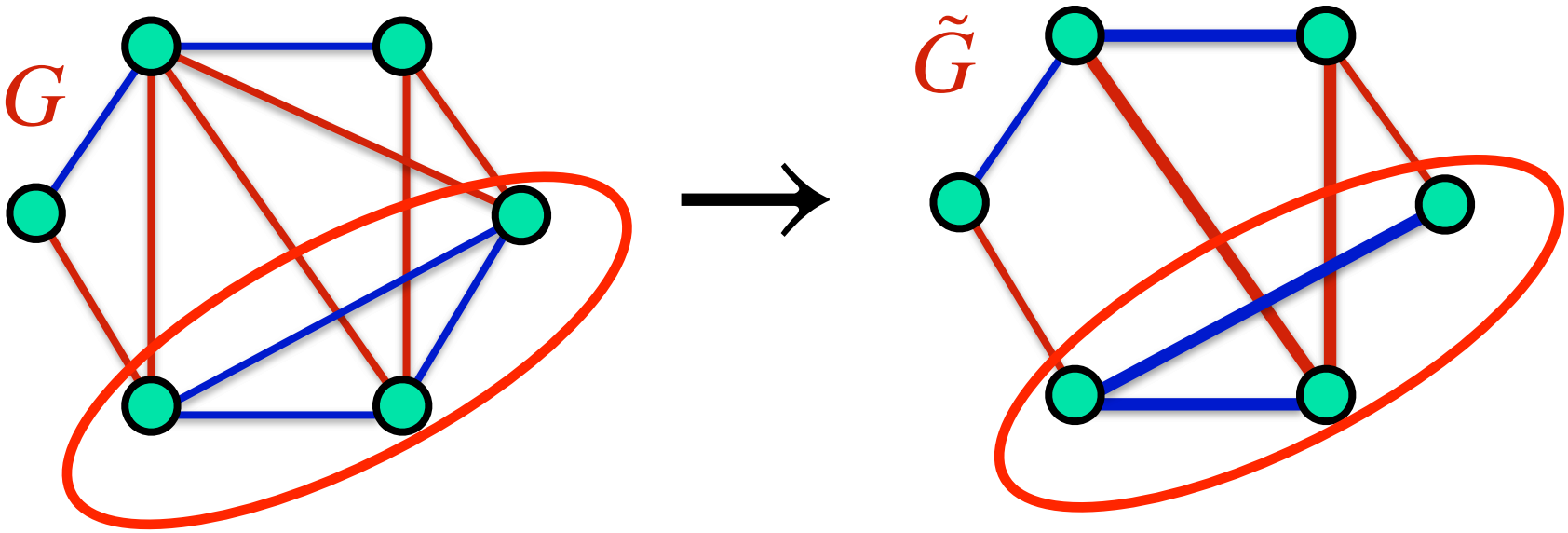
# Cut Sparsifiers

**Definition:** For a graph $G$, a *cut-sparsifier* is a sparse subgraph $\tilde{G}$ which preserves every cut size.



# Cut Sparsifiers

**Definition:** For a graph $G$, a *cut-sparsifier* is a sparse subgraph $\tilde{G}$ which preserves every cut size.



# Building Cut-Sparsifiers

**Definition:** for a graph $G = (V, E)$ and edge $e \in E$, define edge strength $\lambda_e = \max_{S \subseteq V; e \subseteq S} \text{mincut}(G[S])$.

In words: induced subgraph containing $e$ with largest mincut

# Building Cut-Sparsifiers

**Definition**: for a graph $G = (V, E)$ and edge $e \in E$, define edge strength $\lambda_e = \max_{S \subseteq V; e \subseteq S} \text{mincut}(G[S])$.

In words: induced subgraph containing $e$ with largest mincut

**Key fact** [Benczur-Karger '95]: sample each edge $e \in E$ with probability $p_e = \frac{\log(n)}{\lambda_e \epsilon^2}$, and give weight $\frac{1}{p_e}$ if sampled.

Retains only $O(n \log(n)/\epsilon^2)$ edges, and produces a $(1 \pm \epsilon)$ cut-sparsifier with high probability!

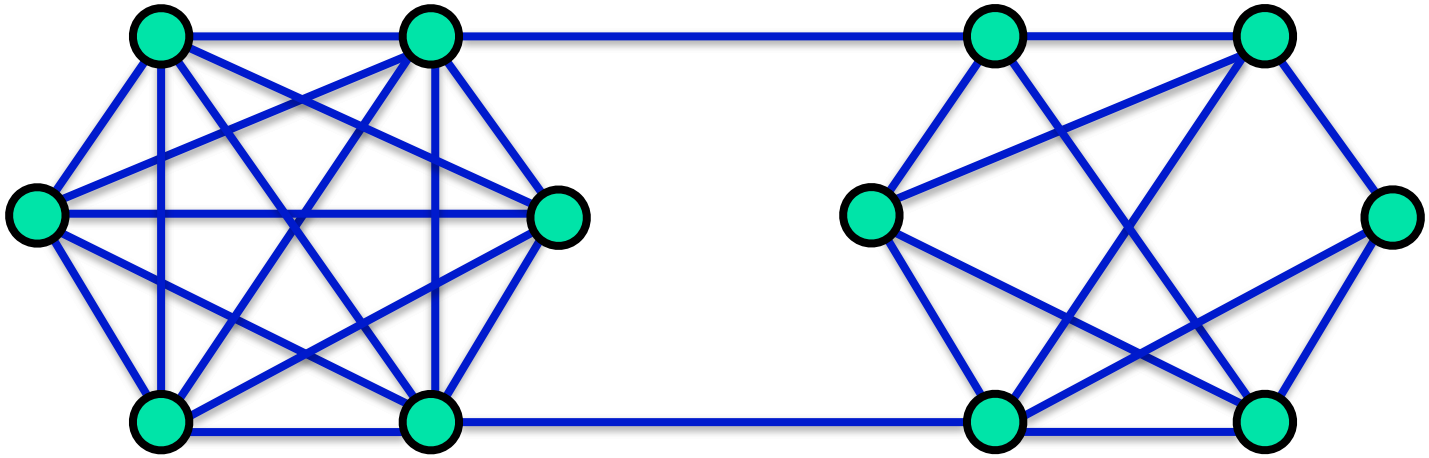
# Building Cut-Sparsifiers with Cut Queries

Theorem [RSW 18]: can build $(1 \pm \epsilon)$ cut-sparsifiers with only $O(n \log(n)/\epsilon^2)$ edges using only $\widetilde{O}(n/\epsilon^2)$ many cut queries.

**Intuition**: simulate the strength-sampling scheme of Benczur and Karger!

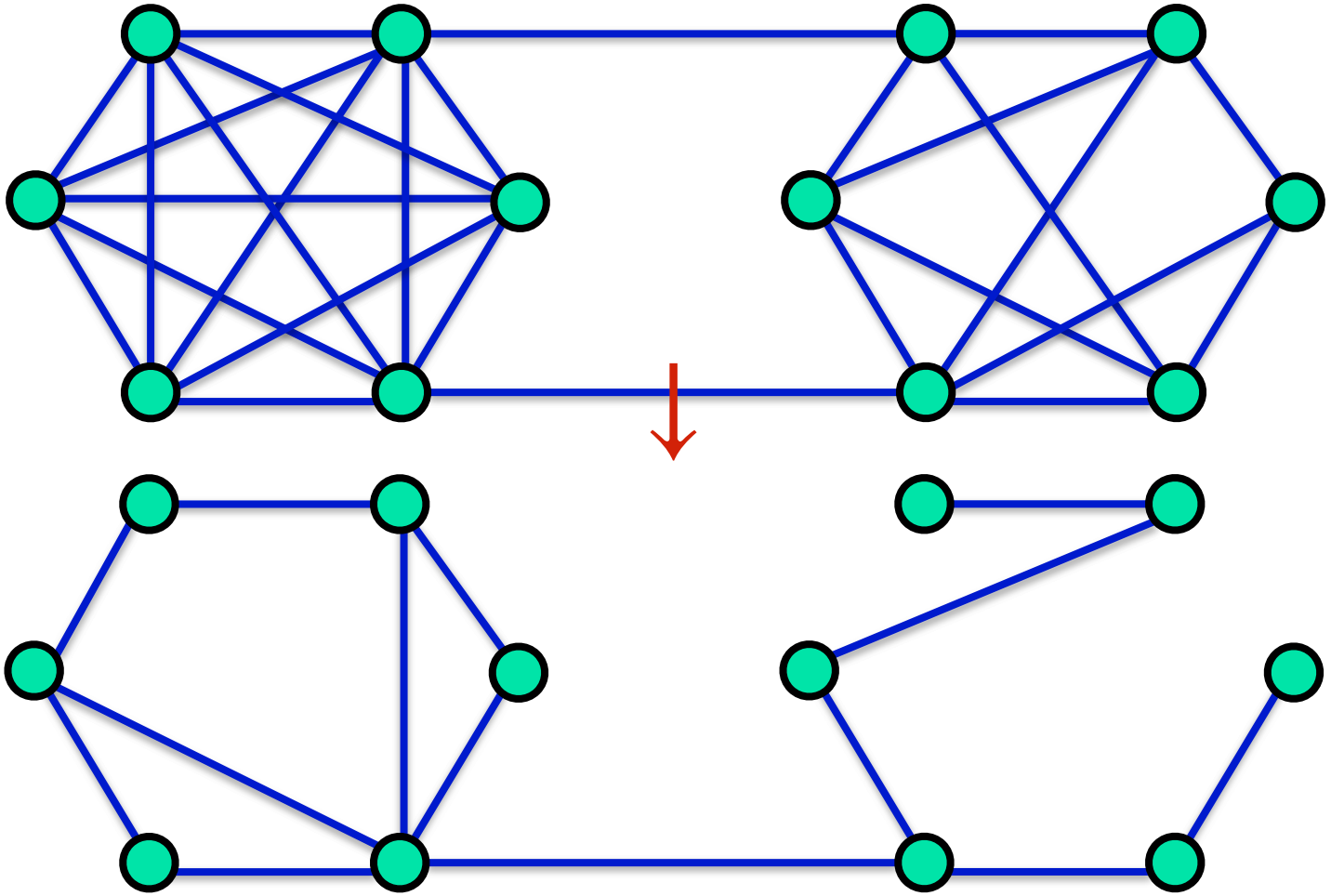
# Building Cut-Sparsifiers with Cut Queries

**Algorithm:** first find the *highest* strength components by subsampling at rate $\log(n)/n$



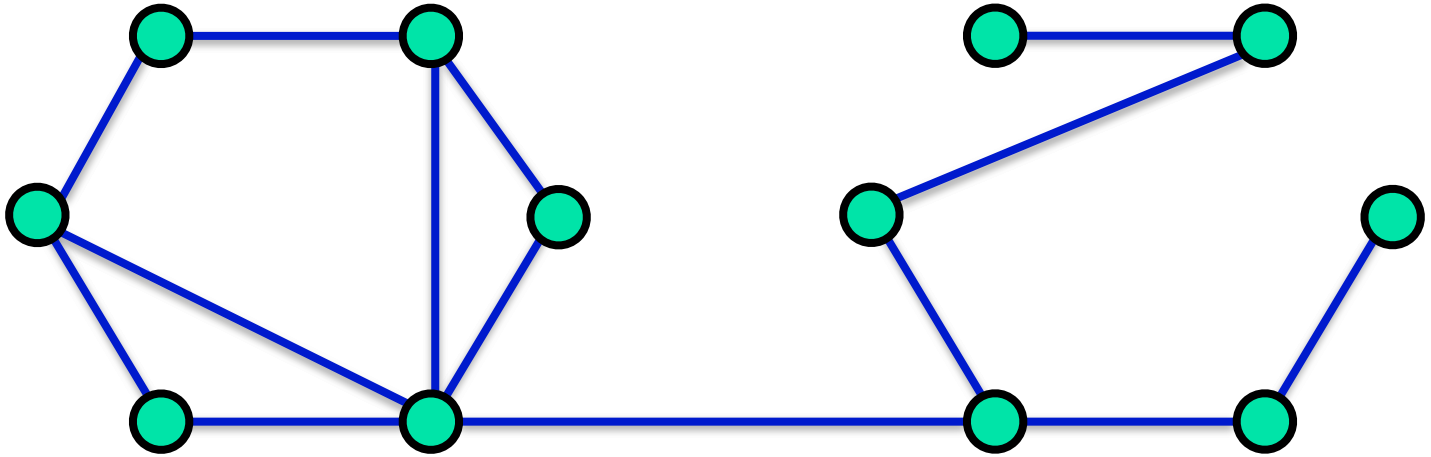
# Building Cut-Sparsifiers with Cut Queries

**Algorithm:** first find the *highest* strength components by subsampling at rate $\log(n)/n$



# Building Cut-Sparsifiers with Cut Queries

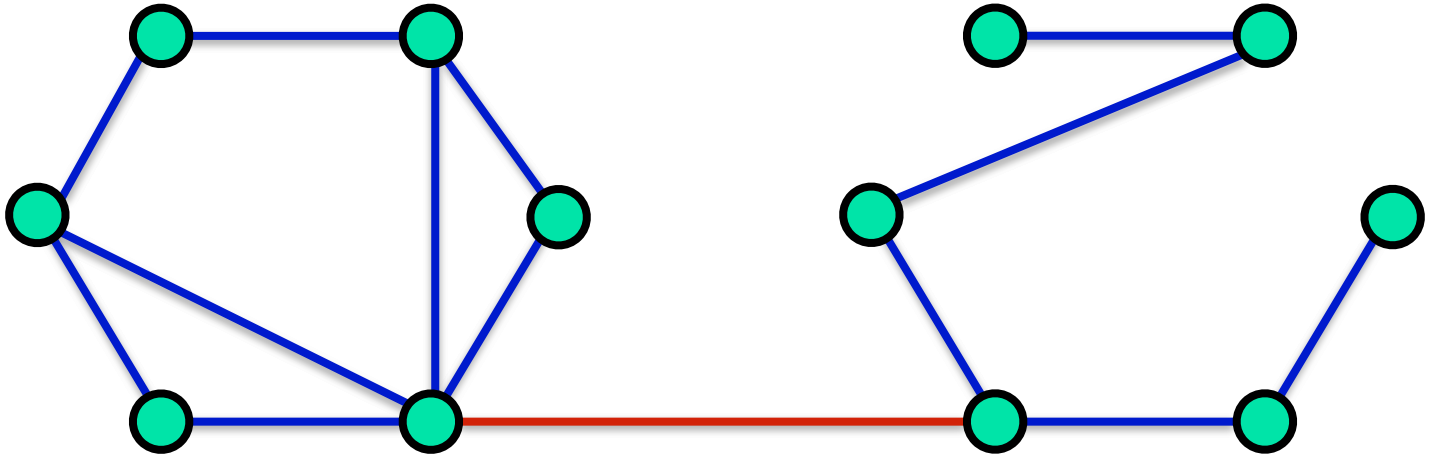
**Algorithm:** first find the *highest* strength components by subsampling at rate $\log(n)/n$



Iteratively compute min-cut, and delete while of size $< \log(n)$

# Building Cut-Sparsifiers with Cut Queries

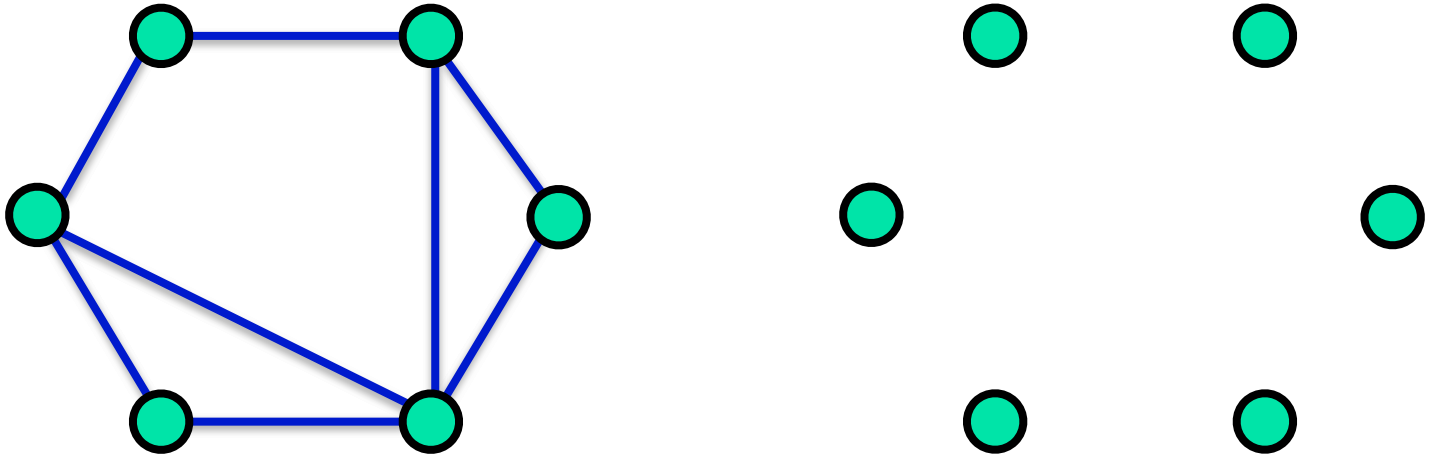
**Algorithm:** first find the *highest* strength components by subsampling at rate $\log(n)/n$



Iteratively compute min-cut, and delete while of size $< \log(n)$

# Building Cut-Sparsifiers with Cut Queries

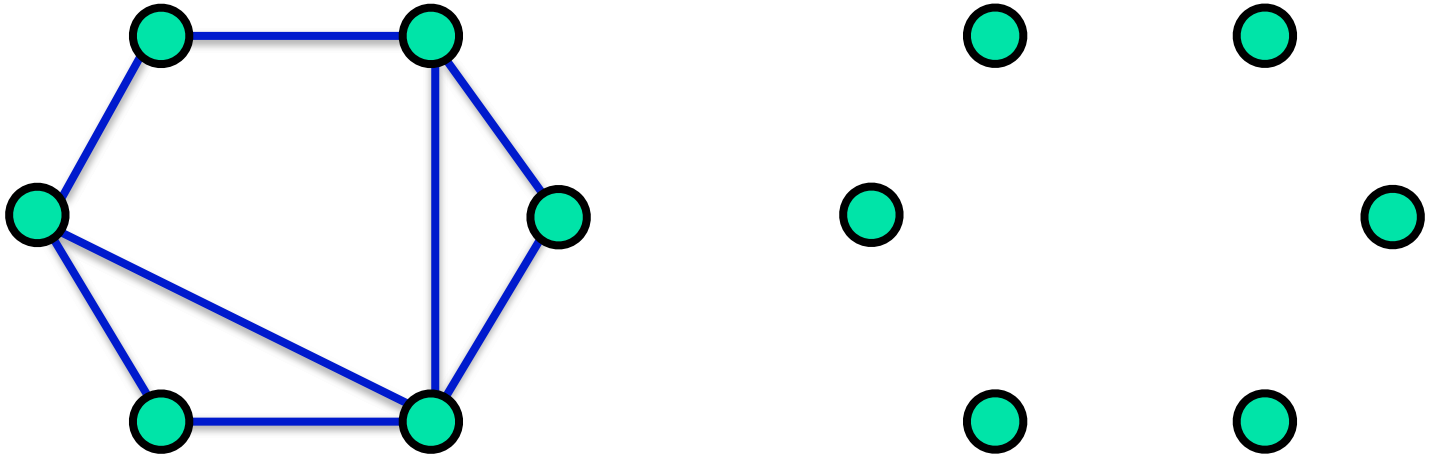
**Algorithm:** first find the *highest* strength components by subsampling at rate $\log(n)/n$



Remaining components were high strength $\geq n/2$ in original graph!

# Building Cut-Sparsifiers with Cut Queries

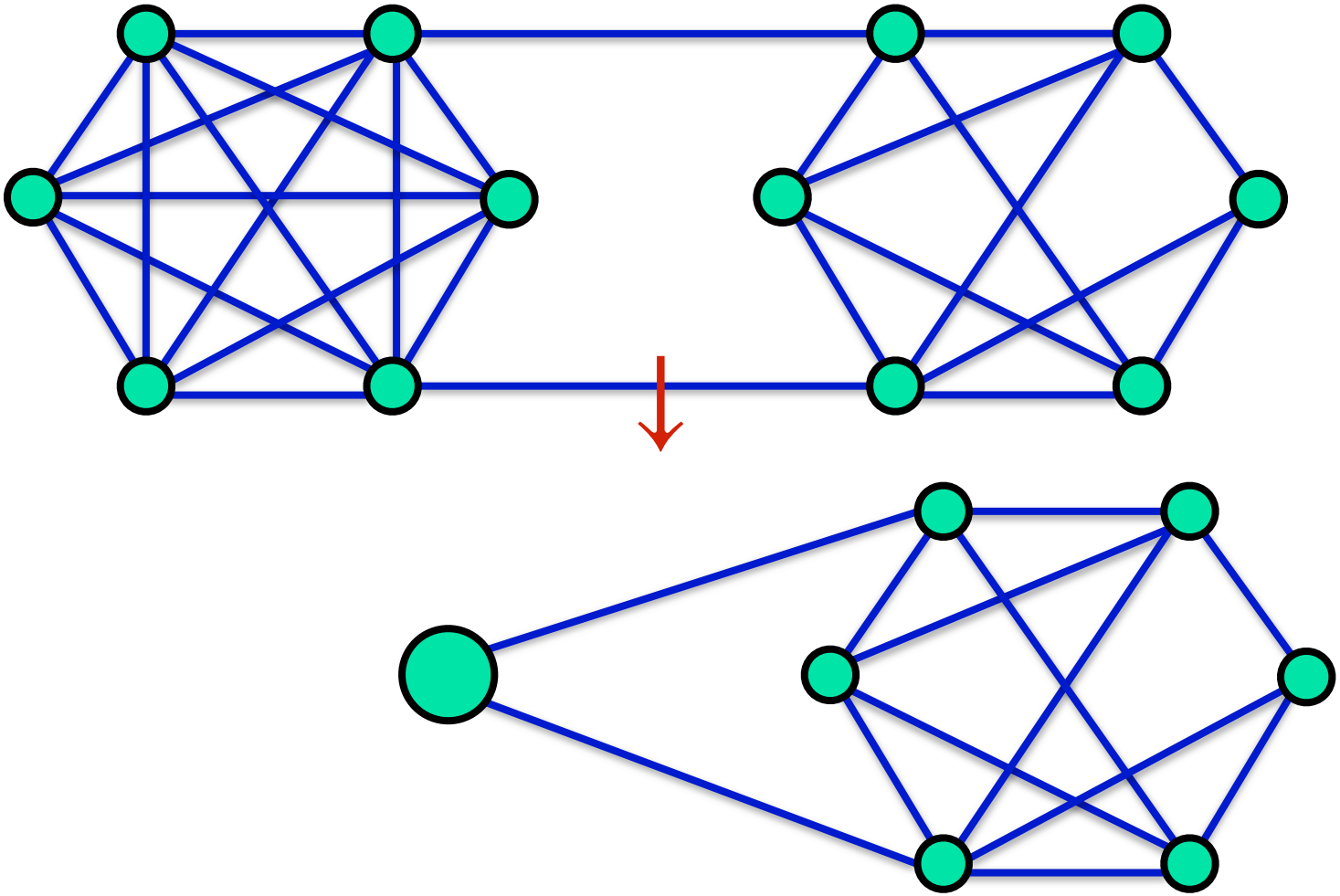
**Algorithm:** first find the *highest* strength components by subsampling at rate $\log(n)/n$



Contract these components in the original graph, and repeat

# Building Cut-Sparsifiers with Cut Queries

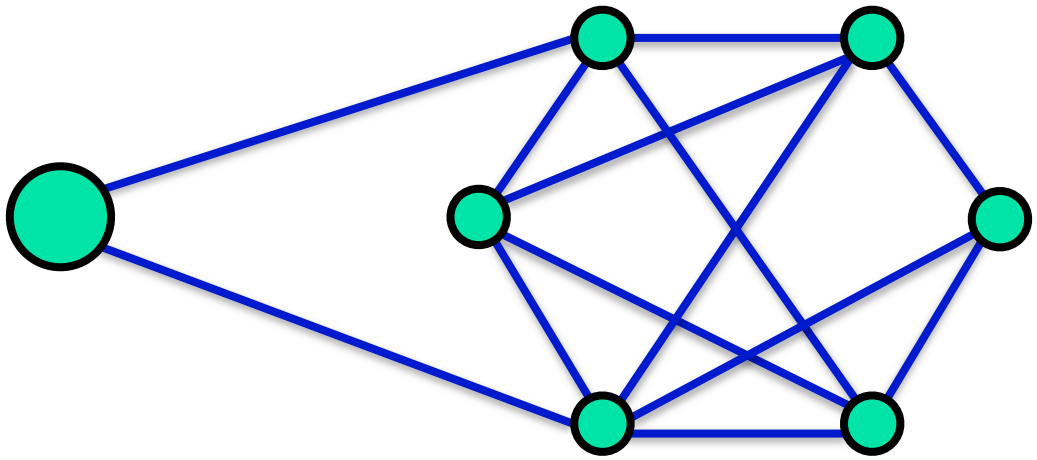
**Algorithm:** contract *highest* strength components



# Building Cut-Sparsifiers with Cut Queries

**Algorithm**: contract *highest* strength components

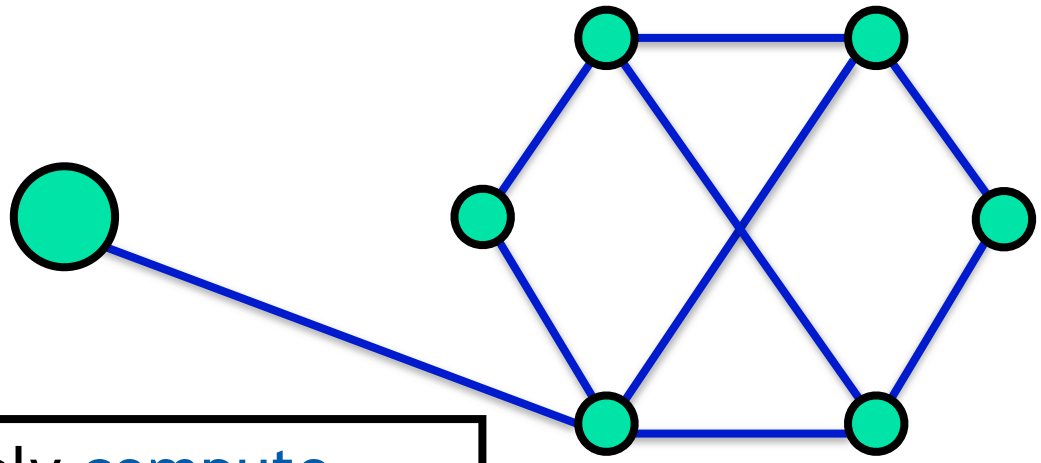
Now repeat to find *slightly lower* strength components:
subsample at rate $2\log(n)/n$



# Building Cut-Sparsifiers with Cut Queries

**Algorithm**: contract *highest* strength components

Now repeat to find *slightly lower* strength components:
subsample at rate $2\log(n)/n$

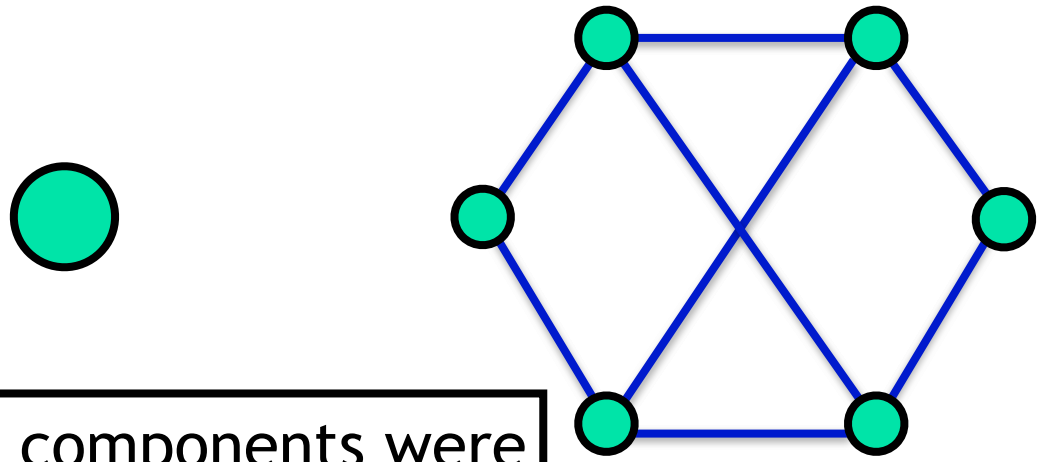


Iteratively compute min-cut, and delete while of size $< \log(n)$

# Building Cut-Sparsifiers with Cut Queries

**Algorithm**: contract *highest* strength components

Now repeat to find *slightly lower* strength components: subsample at rate $2\log(n)/n$



Remaining components were strength $\in [n/4, n/2]$ in original graph!

# Building Cut-Sparsifiers with Cut Queries

**Algorithm**: contract *highest* strength components

Now repeat to find *slightly lower* strength components:
subsample at rate $2\log(n)/n$

Keep repeating: in $i$th round, sample at rate $2^i \cdot \log(n)/n$,
discover components of strength $\in [n/2^i, n/2^{i-1}]$.

# Building Cut-Sparsifiers with Cut Queries

**Algorithm**: contract *highest* strength components

Now repeat to find *slightly lower* strength components: subsample at rate $2\log(n)/n$

Keep repeating: in $i$th round, sample at rate $2^i \cdot \log(n)/n$, discover components of strength $\in [n/2^i, n/2^{i-1}]$.

Lemma [RSW]: Entire procedure can be done in $\tilde{O}(n)$ many cut queries.

Information now suffices for strength-based subsampling!

# Building Cut-Sparsifiers with Cut Queries

**Algorithm**: contract *highest* strength components

Now repeat to find *slightly lower* strength components: subsample at rate $2\log(n)/n$

Keep repeating: in $i$th round, sample at rate $2^i \cdot \log(n)/n$, discover components of strength $\in [n/2^i, n/2^{i-1}]$.

Lemma [RSW]: Entire procedure can be done in $\tilde{O}(n)$ many cut queries.

Theorem [RSW]: Can build $(1 \pm \epsilon)$ cut sparsifiers in $\tilde{O}(n/\epsilon^2)$ many cut queries.

# Minimum s-t Cuts

# Computing s-t Min-Cuts

**Goal**: given a graph $G = (V, E)$, vertices $s, t$, compute smallest cut separating $s, t$.

# Computing s-t Min-Cuts

**Goal**: given a graph $G = (V, E)$, vertices $s, t$, compute smallest cut separating $s, t$.

**Algorithm**:

1. Compute a $(1 \pm \epsilon)$ sparsifier $H$ of $G$ with $\epsilon = 1/n^{1/3}$.

# Computing s-t Min-Cuts

**Goal**: given a graph $G = (V, E)$, vertices $s, t$, compute smallest cut separating $s, t$.

**Algorithm**:

1. Compute a $(1 \pm \epsilon)$ sparsifier $H$ of $G$ with $\epsilon = 1/n^{1/3}$.
2. Compute a maximum flow $F$ from $s$ to $t$ in $H$. Let $H' = H - F$.

# Computing s-t Min-Cuts

**Goal**: given a graph $G = (V, E)$, vertices $s, t$, compute smallest cut separating $s, t$.

**Algorithm**:

1. Compute a $(1 \pm \epsilon)$ sparsifier $H$ of $G$ with $\epsilon = 1/n^{1/3}$.
2. Compute a maximum flow $F$ from $s$ to $t$ in $H$. Let $H' = H - F$.
3. Find all components in $H'$ with strength $\geq 3\epsilon n$, call this collection $C$.

# Computing s-t Min-Cuts

**Goal**: given a graph $G = (V, E)$, vertices $s, t$, compute smallest cut separating $s, t$.

**Algorithm**:

1. Compute a $(1 \pm \epsilon)$ sparsifier $H$ of $G$ with $\epsilon = 1/n^{1/3}$.
2. Compute a maximum flow $F$ from $s$ to $t$ in $H$. Let $H' = H - F$.
3. Find all components in $H'$ with strength $\geq 3\epsilon n$, call this collection $C$.
4. Learn the graph $G/C$, and compute the exact $s, t$ min-cut.

# Computing s-t Min-Cuts

**Goal**: given a graph $G = (V, E)$, vertices $s, t$, compute smallest cut separating $s, t$.

**Algorithm**:

1. Compute a $(1 \pm \epsilon)$ sparsifier $H$ of $G$ with $\epsilon = 1/n^{1/3}$.
2. Compute a maximum flow $F$ from $s$ to $t$ in $H$. Let $H' = H - F$.
3. Find all components in $H'$ with strength $\geq 3\epsilon n$, call this collection $C$.
4. Learn the graph $G/C$, and compute the exact $s, t$ min-cut.

Theorem [RSW]: Algorithm can be executed with $\tilde{O}(n^{5/3})$ cut queries, and finds the $s, t$ min-cut w.h.p.!

# Computing s-t Min-Cuts

**Algorithm:**

1. Compute a $(1 \pm \epsilon)$ sparsifier $H$ of $G$ with $\epsilon = 1/n^{1/3}$.
2. Compute a maximum flow $F$ from $s$ to $t$ in $H$. Let $H' = H - F$.
3. Find all components in $H'$ with strength $\geq 3\epsilon n$, call this collection $C$.
4. Learn the graph $G/C$, and compute the exact $s, t$ min-cut.

**Key Lemma:** In $G/C$,

1. There are $\tilde{O}(n^{5/3})$ many edges
2. The exact $s, t$ min-cut survives

# Computing s-t Min-Cuts

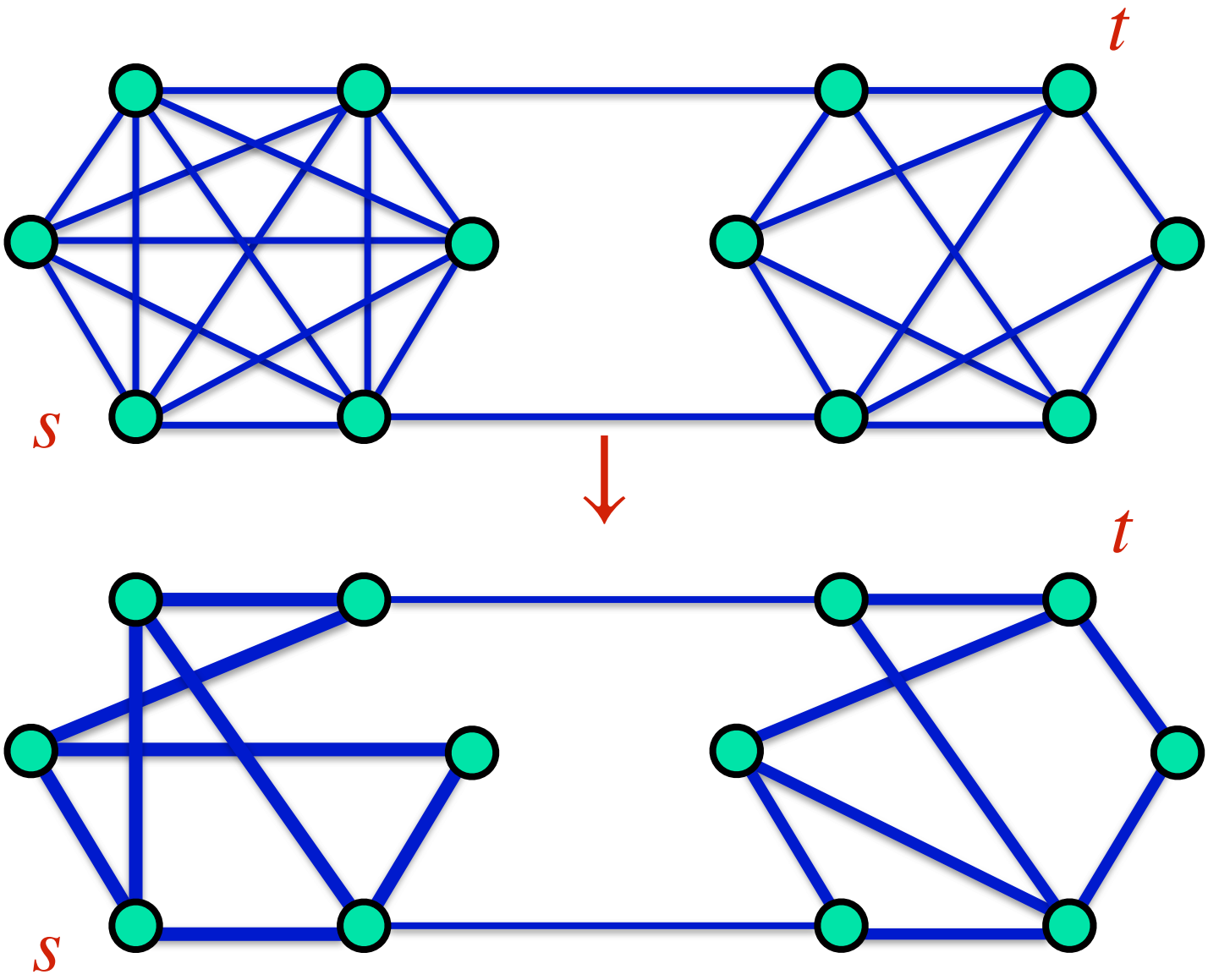
**Algorithm:**

1. Compute a $(1 \pm \epsilon)$ sparsifier $H$ of $G$ with $\epsilon = 1/n^{1/3}$.
2. Compute a maximum flow $F$ from $s$ to $t$ in $H$. Let $H' = H - F$.
3. Find all components in $H'$ with strength $\geq 3\epsilon n$, call this collection $C$.
4. Learn the graph $G/C$, and compute the exact $s, t$ min-cut.

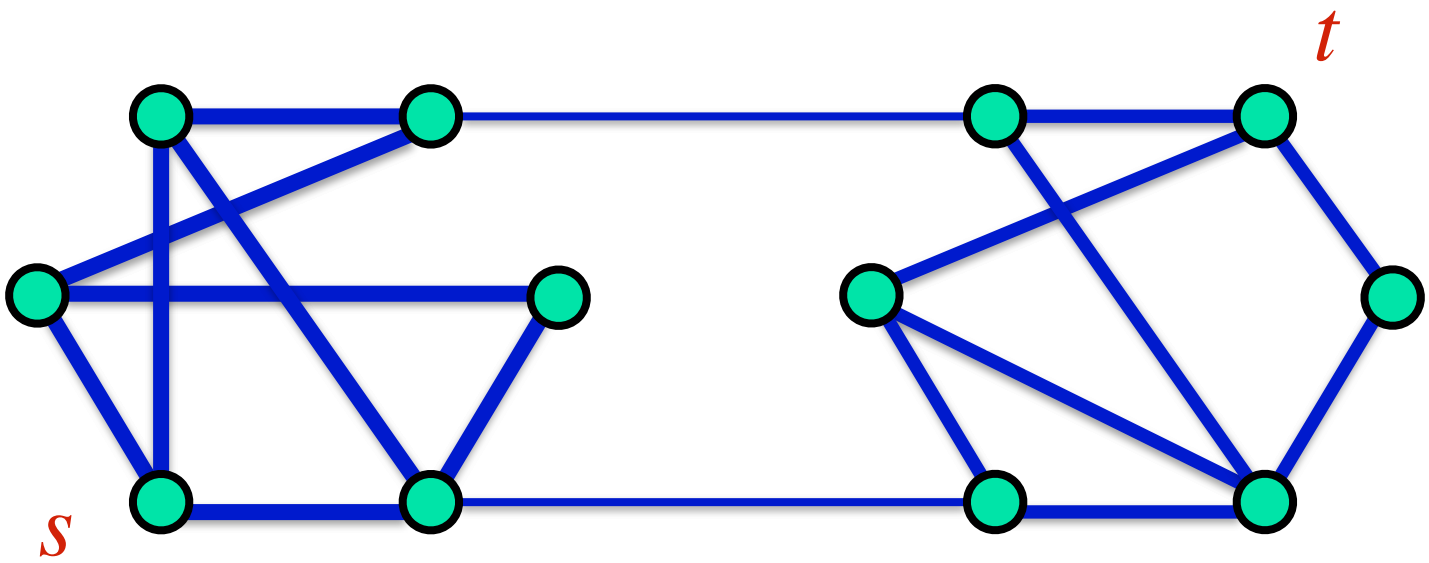
**Key Lemma:** In $G/C$,

1. There are $\tilde{O}(n^{5/3})$ many edges
2. The exact $s, t$ min-cut survives

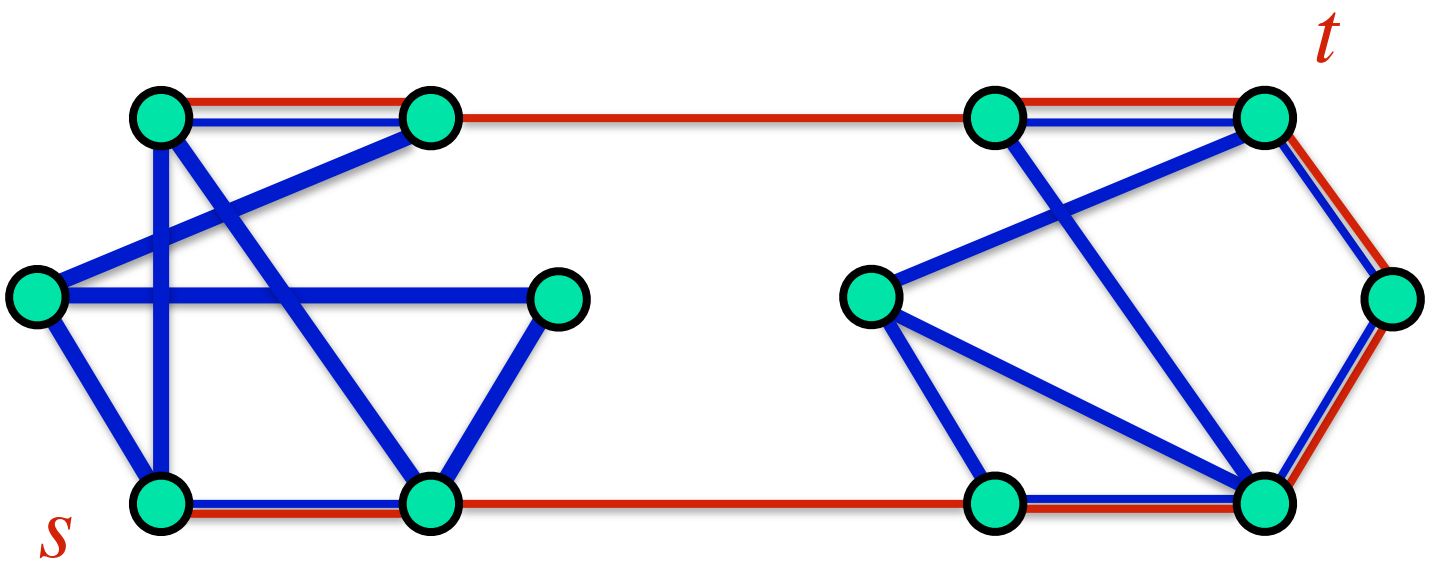
# Computing s-t Min-Cuts



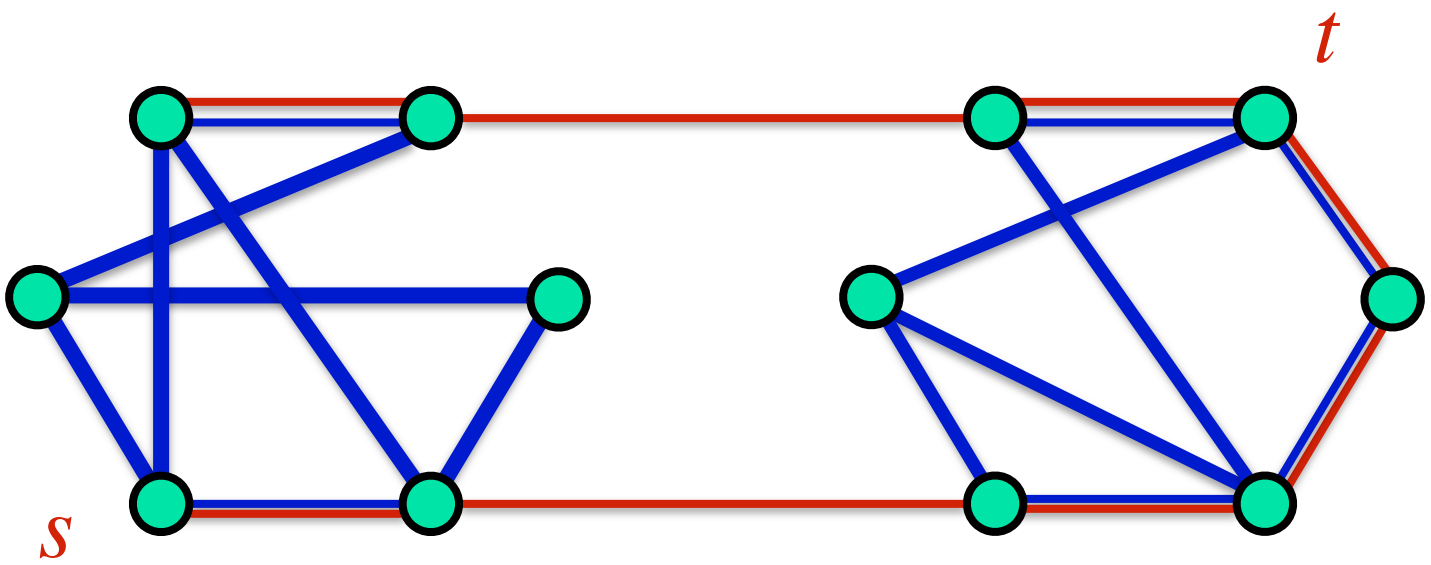
# Computing s-t Min-Cuts



# Computing s-t Min-Cuts

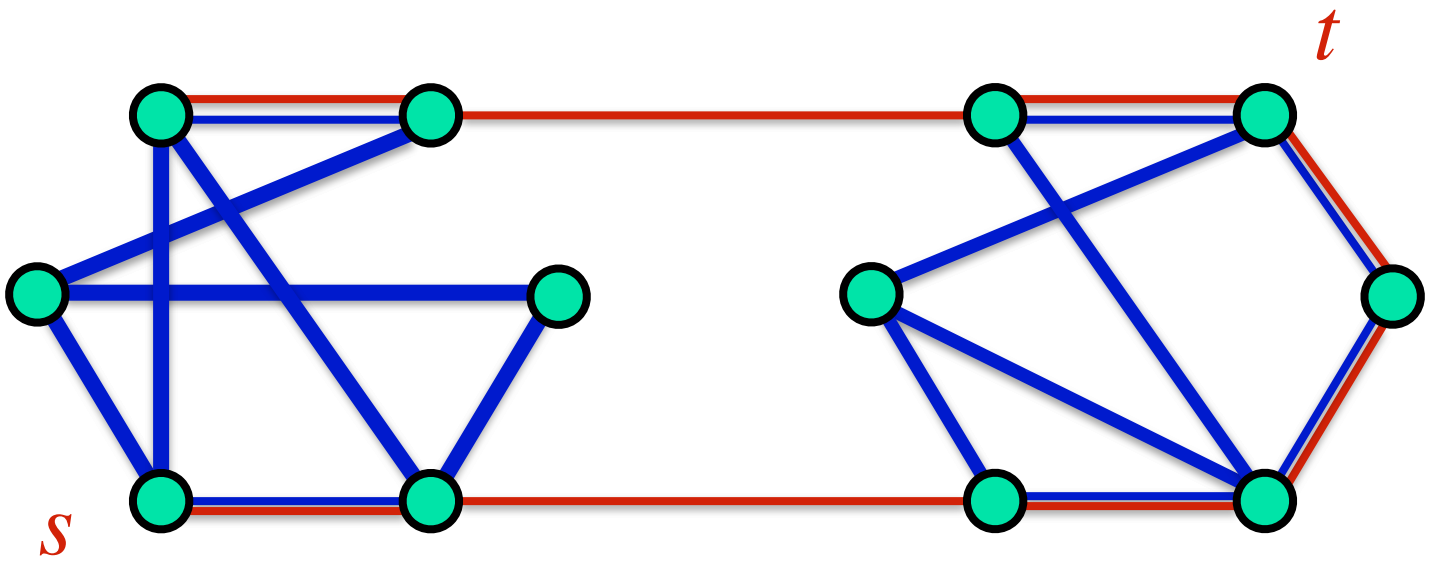


# Computing s-t Min-Cuts



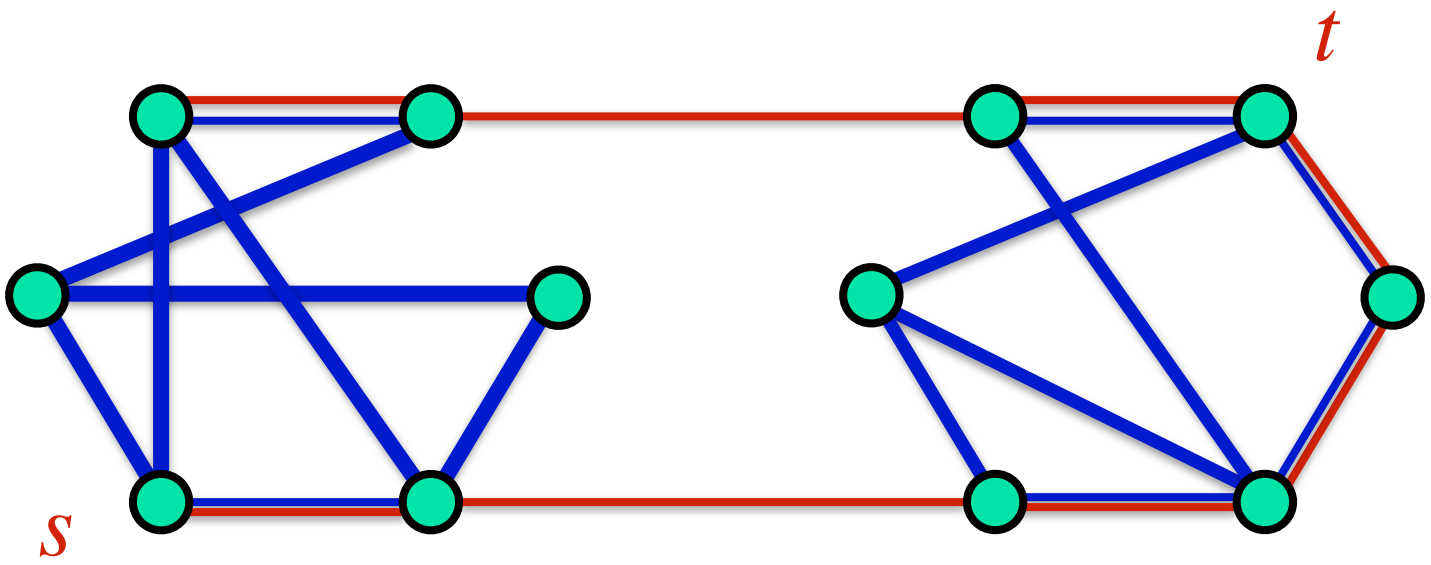

Because graph is a sparsifier, the value of the flow is within $(1 \pm \epsilon)$ of the optimal s-t cut size.

# Computing s-t Min-Cuts




**Intuition:** no s-t mincut will cut blue components that are $3\epsilon n$-connected:

# Computing s-t Min-Cuts




**Intuition:** no s-t mincut will cut blue components that are $3\epsilon n$-connected:
Such s-t cuts are of size $\text{flow} + 3\epsilon n$!

# Computing s-t Min-Cuts

**Algorithm:**

1. Compute a $(1 \pm \epsilon)$ sparsifier $H$ of $G$ with $\epsilon = 1/n^{1/3}$.
2. Compute a maximum flow $F$ from $s$ to $t$ in $H$. Let $H' = H - F$.
3. Find all components in $H'$ with strength $\geq 3\epsilon n$, call this collection $C$.
4. Learn the graph $G/C$, and compute the exact $s, t$ min-cut.

**Key Lemma:** In $G/C$,

1. There are $\tilde{O}(n^{5/3})$ many edges
2. The exact $s, t$ min-cut survives

With sparsifier theorem, get main theorem immediately!

# Some Remarks

Can we do better than $\tilde{O}(n^{5/3})$ queries?

- $\tilde{O}(n^{8/5})$ queries by Jiang, Nanongkai, and Sawettamalya [SODA 2026]
- Recently $\tilde{O}(n^{3/2})$ queries by Kenneth-Mordoch and Krauthgamer [STOC 2026]

# Some Remarks

Can we do better than $\tilde{O}(n^{5/3})$ queries?

- $\tilde{O}(n^{8/5})$ queries by Jiang, Nanongkai, and Sawettamalya [SODA 2026]
- Recently $\tilde{O}(n^{3/2})$ queries by Kenneth-Mordoch and Krauthgamer [STOC 2026]
- **Barrier** here: if algorithm recovers the max-flow, there can be as many as $n^{3/2}$ many edges.

# Some Remarks

Can we do better than $\tilde{O}(n^{5/3})$ queries?

- $\tilde{O}(n^{8/5})$ queries by Jiang, Nanongkai, and Sawettamalya [SODA 2026]
- Recently $\tilde{O}(n^{3/2})$ queries by Kenneth-Mordoch and Krauthgamer [STOC 2026]
- **Barrier** here: if algorithm recovers the max-flow, there can be as many as $n^{3/2}$ many edges.

Can we get deterministic algorithms?

- Best is $\tilde{O}(n^{5/3})$ queries by Anand, Saranurak, Wang [SODA 2025]. Open question to improve!

# Thanks!