Deterministic Min-cut in Poly-logarithmic Max-flows

Jason Li

Joint work with Debmalya Panigrahi (Duke)

Introduction

All graphs are undirected and either unweighted or non-negatively weighted

Introduction

All graphs are undirected and either unweighted or non-negatively weighted

(Global) mincut: given a graph, find a minimum cardinality/weight cut whose removal disconnects the graph

Introduction

All graphs are undirected and either unweighted or non-negatively weighted

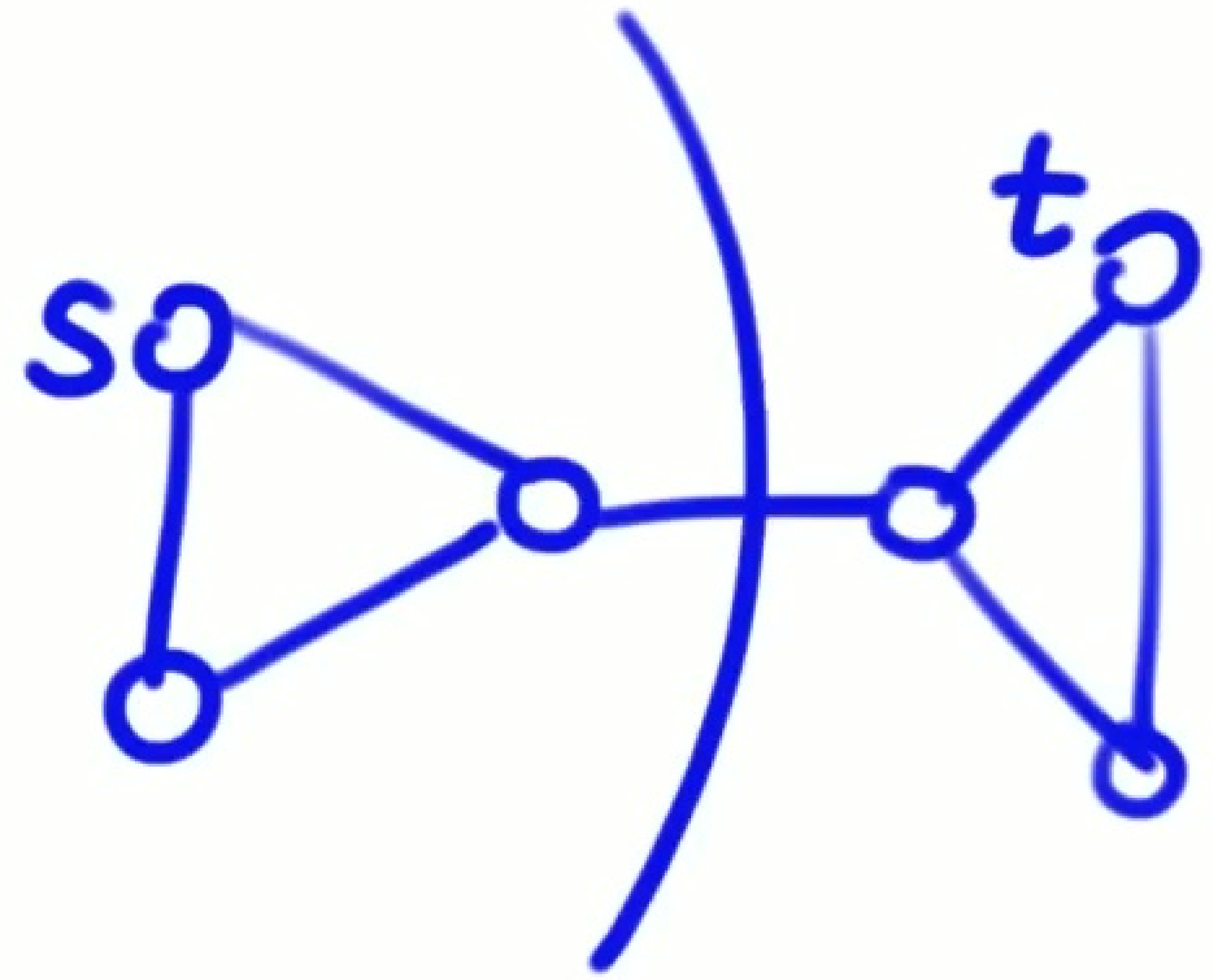
(Global) mincut: given a graph, find a minimum cardinality/weight cut whose removal disconnects the graph

s-t mincut: ...cut whose removal disconnects vertices s and t

Introduction

All graphs are undirected and either unweighted or non-negatively weighted

(Global) mincut: given a graph, find a minimum cardinality/weight cut whose removal disconnects the graph

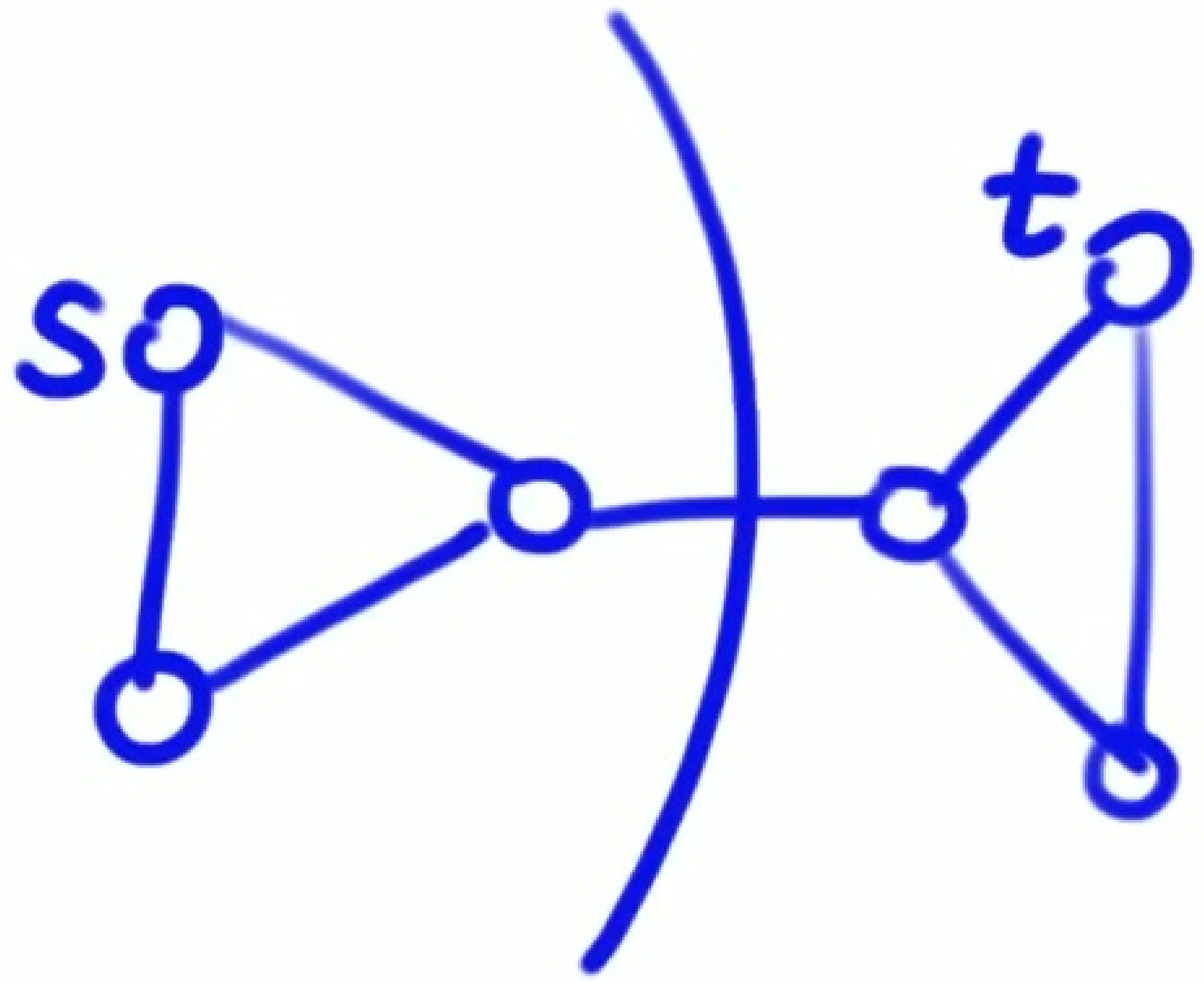


s-t mincut: ...cut whose removal disconnects vertices s and t

Introduction

All graphs are undirected and either unweighted or non-negatively weighted

(Global) mincut: given a graph, find a minimum cardinality/weight cut whose removal disconnects the graph



s-t mincut: ...cut whose removal disconnects vertices s and t

Max-flow min-cut theorem: s-t min-cut = s-t max-flow,
can recover s-t min-cut given s-t max-flow

Global min-cut algorithms

Find s-t max-flow for all pairs s,t: $O(n^2)$ max-flows

Fix any vertex s; find s-t max-flow for all t: $O(n)$ max-flows

Global min-cut algorithms

Find s-t max-flow for all pairs s,t: $O(n^2)$ max-flows

Fix any vertex s; find s-t max-flow for all t: $O(n)$ max-flows

Push-relabel max-flow algorithm: $\tilde{O}(mn)$ time [Hao-Orlin '94]

Global min-cut algorithms

Find s-t max-flow for all pairs s,t: $O(n^2)$ max-flows

Fix any vertex s; find s-t max-flow for all t: $O(n)$ max-flows

Push-relabel max-flow algorithm: $\tilde{O}(mn)$ time [Hao-Orlin '94]

Non max-flow based techniques: $\tilde{O}(mn)$ time

[Nagamochi-Ibaraki '92, Gabow '95, Stoer-Wagner '97]

Global min-cut algorithms

Find s-t max-flow for all pairs s,t: $O(n^2)$ max-flows

Fix any vertex s; find s-t max-flow for all t: $O(n)$ max-flows

Push-relabel max-flow algorithm: $\tilde{O}(mn)$ time [Hao-Orlin '94]

Non max-flow based techniques: $\tilde{O}(mn)$ time
[Nagamochi-Ibaraki '92, Gabow '95, Stoer-Wagner '97]

Random contraction: $\tilde{O}(n^2)$ time [Karger '92, Karger-Stein '94]

Global min-cut algorithms

Find s-t max-flow for all pairs s,t: $O(n^2)$ max-flows

Fix any vertex s; find s-t max-flow for all t: $O(n)$ max-flows

Push-relabel max-flow algorithm: $\tilde{O}(mn)$ time [Hao-Orlin '94]

Non max-flow based techniques: $\tilde{O}(mn)$ time [Nagamochi-Ibaraki '92, Gabow '95, Stoer-Wagner '97]

Random contraction: $\tilde{O}(n^2)$ time [Karger '92, Karger-Stein '94]

Random sampling + tree packing: $\tilde{O}(m)$ time [Karger '95]

Global min-cut algorithms

Find s-t max-flow for all pairs s,t: $O(n^2)$ max-flows

Fix any vertex s; find s-t max-flow for all t: $O(n)$ max-flows

Push-relabel max-flow algorithm: $\tilde{O}(mn)$ time [Hao-Orlin '94]

Non max-flow based techniques: $\tilde{O}(mn)$ time
[Nagamochi-Ibaraki '92, Gabow '95, Stoer-Wagner '97]

Random contraction: $\tilde{O}(n^2)$ time [Karger '92, Karger-Stein '94]

Random sampling + tree packing: $\tilde{O}(m)$ time [Karger '95]

Best deterministic algorithm still $\tilde{O}(mn)$

Global min-cut algorithms

Find s-t max-flow for all pairs s,t: $O(n^2)$ max-flows

Fix any vertex s; find s-t max-flow for all t: $O(n)$ max-flows polylog(n) ?

Push-relabel max-flow algorithm: $\tilde{O}(mn)$ time [Hao-Orlin '94]

Non max-flow based techniques: $\tilde{O}(mn)$ time
[Nagamochi-Ibaraki '92, Gabow '95, Stoer-Wagner '97]

Random contraction: $\tilde{O}(n^2)$ time [Karger '92, Karger-Stein '94]

Random sampling + tree packing: $\tilde{O}(m)$ ✓ time [Karger '95]

Best deterministic algorithm still $\tilde{O}(mn)$ $\tilde{O}(m)$?

Global min-cut algorithms

Open problem 1: Can we solve (global) min-cut using polylog(n) s-t max-flows?

Open problem 2: Can we solve min-cut deterministically in $\tilde{O}(m)$ time?

Global min-cut algorithms

Open problem 1: Can we solve (global) min-cut using polylog(n) s-t max-flows?

Open problem 2: Can we solve min-cut deterministically in $\tilde{O}(m)$ time?

No progress till 5 years back

[Kawarabayashi and Thorup 2015]: deterministic min-cut on simple graphs in $\tilde{O}(m)$ time

[Henzinger, Rao, Wang 2017]: improved to $O(m \log^2 n \log\log^2 n)$

Global min-cut algorithms

Open problem 1: Can we solve (global) min-cut using polylog(n) s-t max-flows?

Open problem 2: Can we solve min-cut deterministically in $\tilde{O}(m)$ time?

No progress till 5 years back

[Kawarabayashi and Thorup 2015]: deterministic min-cut on simple graphs in $\tilde{O}(m)$ time

[Henzinger, Rao, Wang 2017]: improved to $O(m \log^2 n \log\log^2 n)$

[This work]: deterministic min-cut for weighted graphs in $O(m^{1+\varepsilon})$ time plus polylog(n) calls to s-t max-flow

Our Approach

Main insights: local algorithms and expander decomposition

Our Approach

Main insights: local algorithms and expander decomposition

Both popularized by [Spielman and Teng 2004] in their $\tilde{O}(m)$ time algorithm on solving Laplacian systems

Our Approach

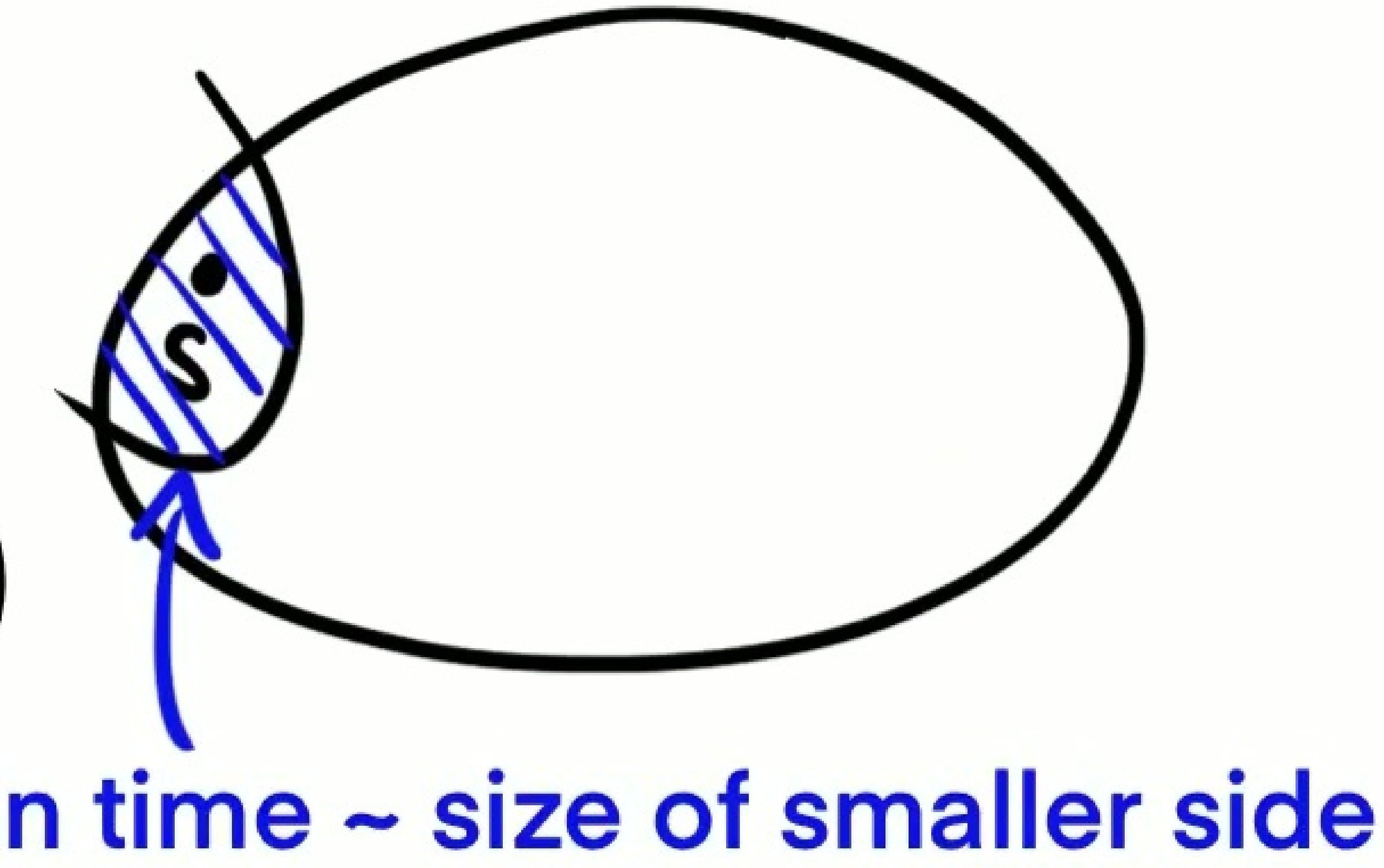
Main insights: local algorithms and expander decomposition

Both popularized by [Spielman and Teng 2004] in their $\tilde{O}(m)$ time algorithm on solving Laplacian systems

"Modern" approach to algorithm design

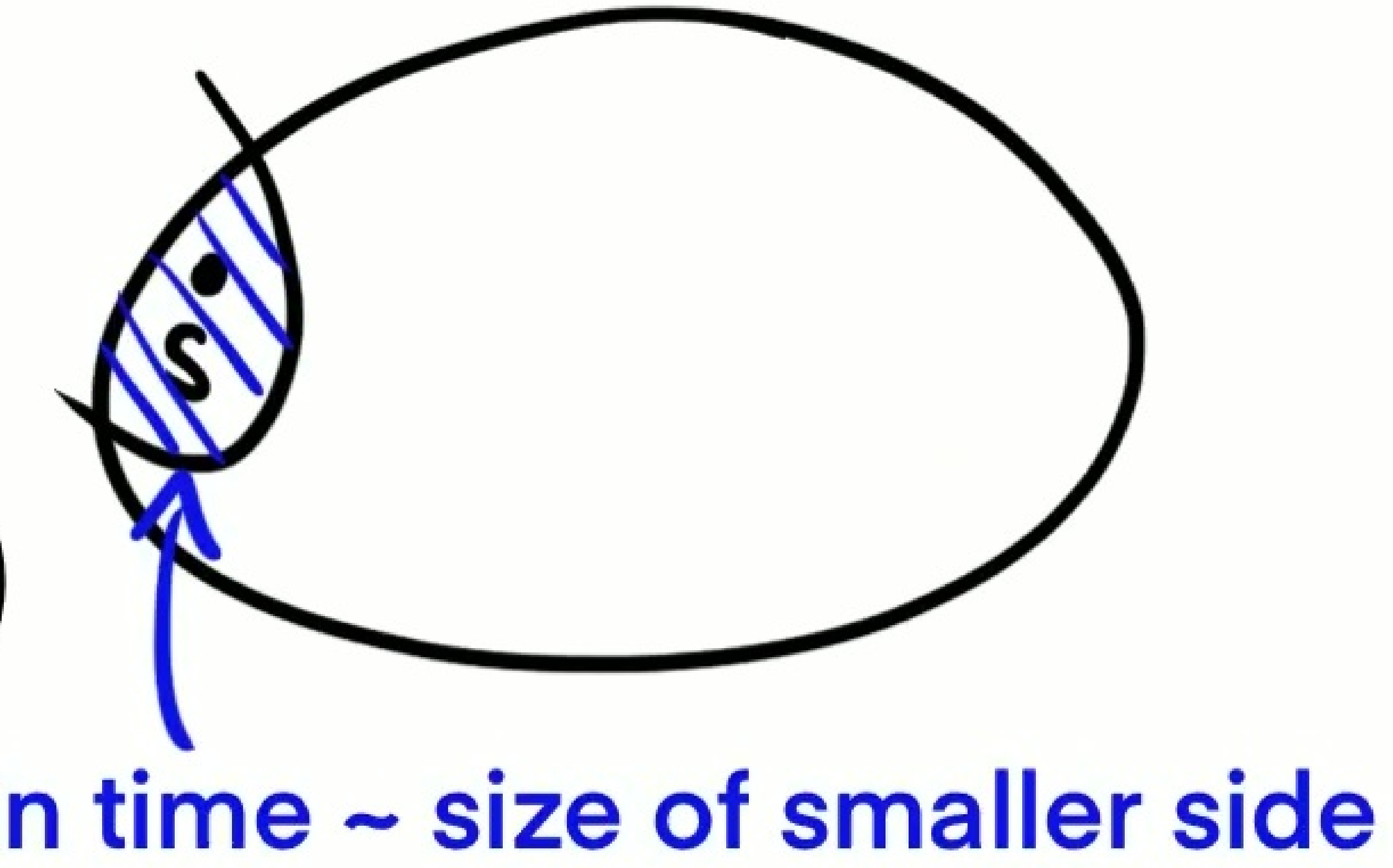
Local Graph Cut Algorithms

Fix a seed vertex s. If there exists a good cut “local” to s, then output in sublinear time (without looking at the whole graph)



Local Graph Cut Algorithms

Fix a seed vertex s. If there exists a good cut "local" to s, then output in sublinear time (without looking at the whole graph)

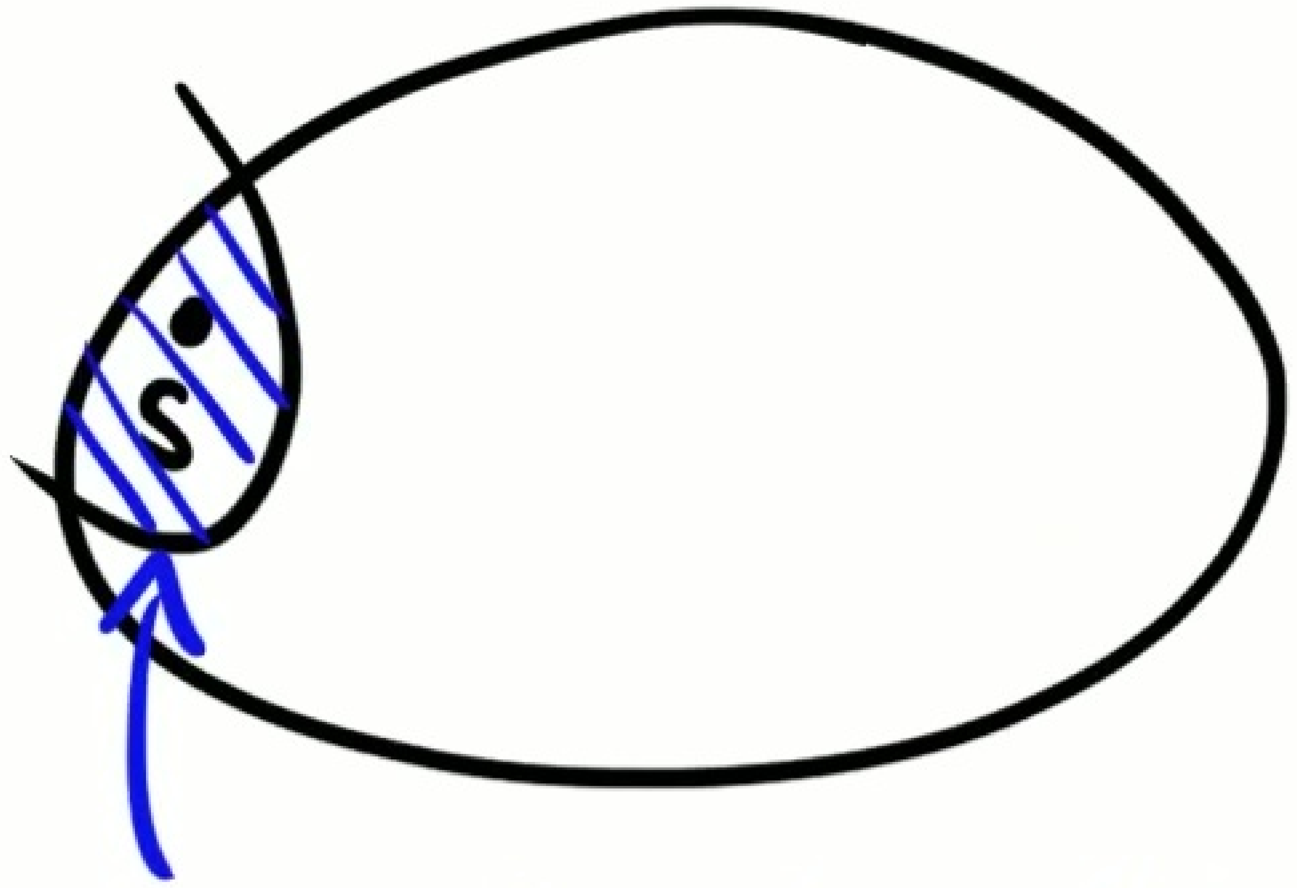


"PageRank Nibble" [Andersen, Chung, Lang '06]:

(approximate) low-conductance cut in time ~ # edges of smaller side

Local Graph Cut Algorithms

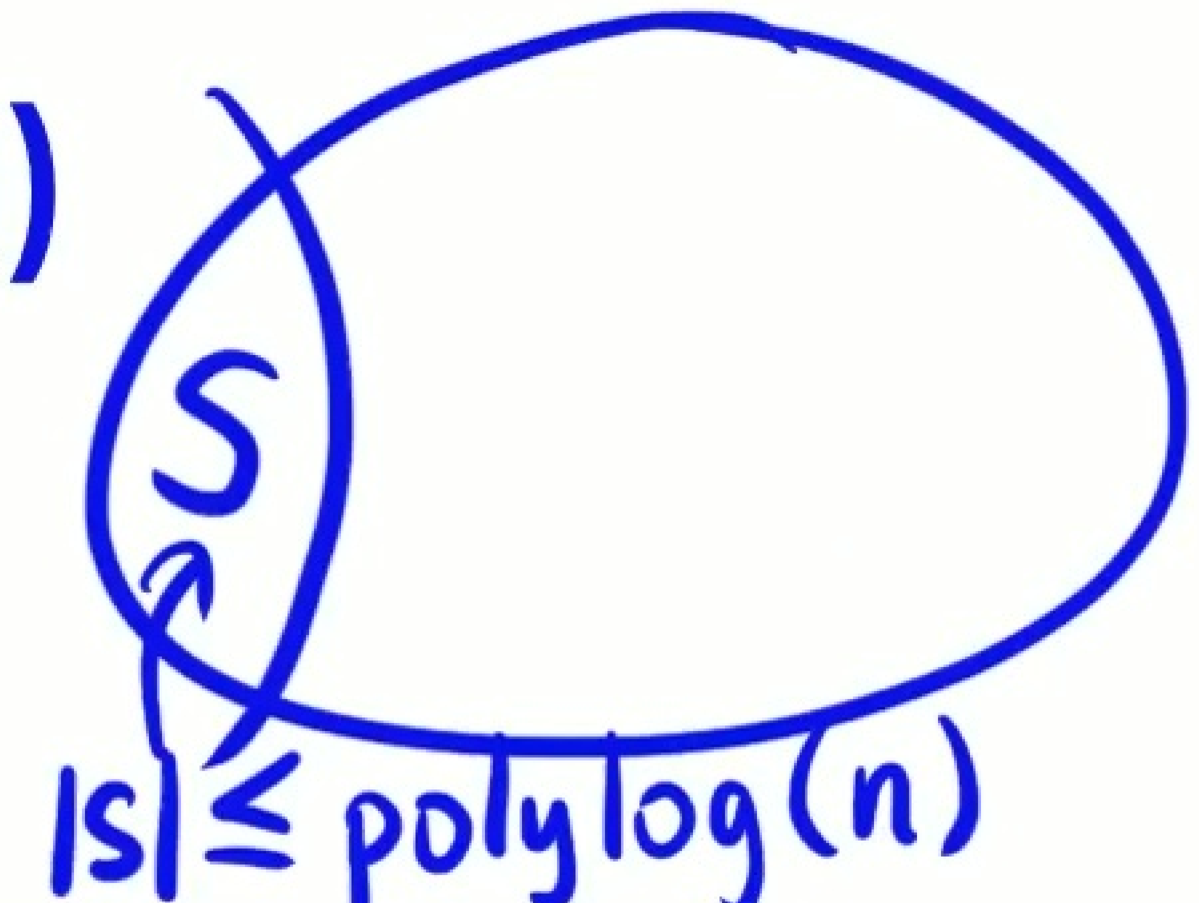
Fix a <u>seed</u> vertex s. If there exists a good cut "local" to s, then output in <u>sublinear</u> time (without looking at the whole graph)



Run in time ~ size of smaller side

"PageRank Nibble" [Andersen, Chung, Lang '06]:
(approximate) low-conductance cut in time ~ # edges of smaller side

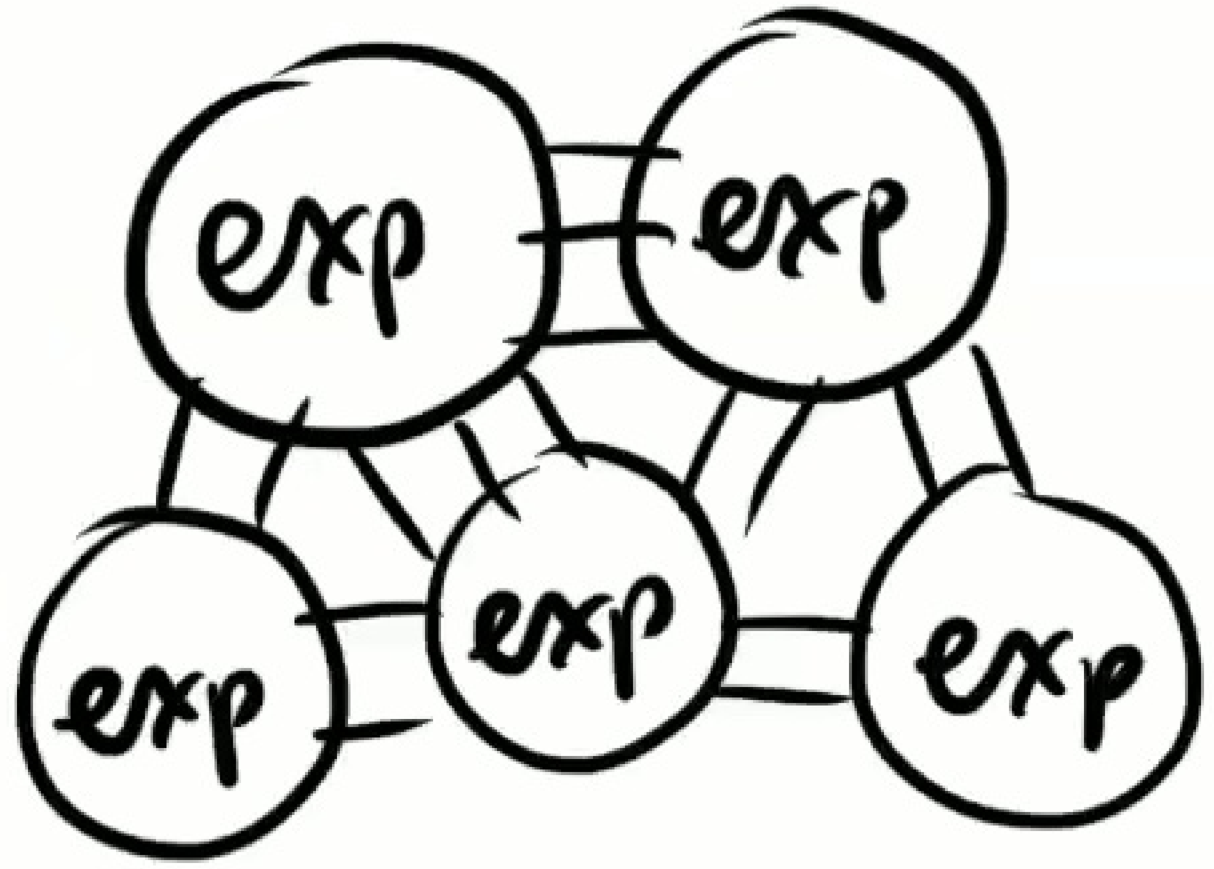
This work: if global min-cut has polylog(n) vertices on smaller side ("unbalanced"), then can find in polylog(n) s-t max-flows



Expander Decomposition

Solve when graph is an expander (easy case)

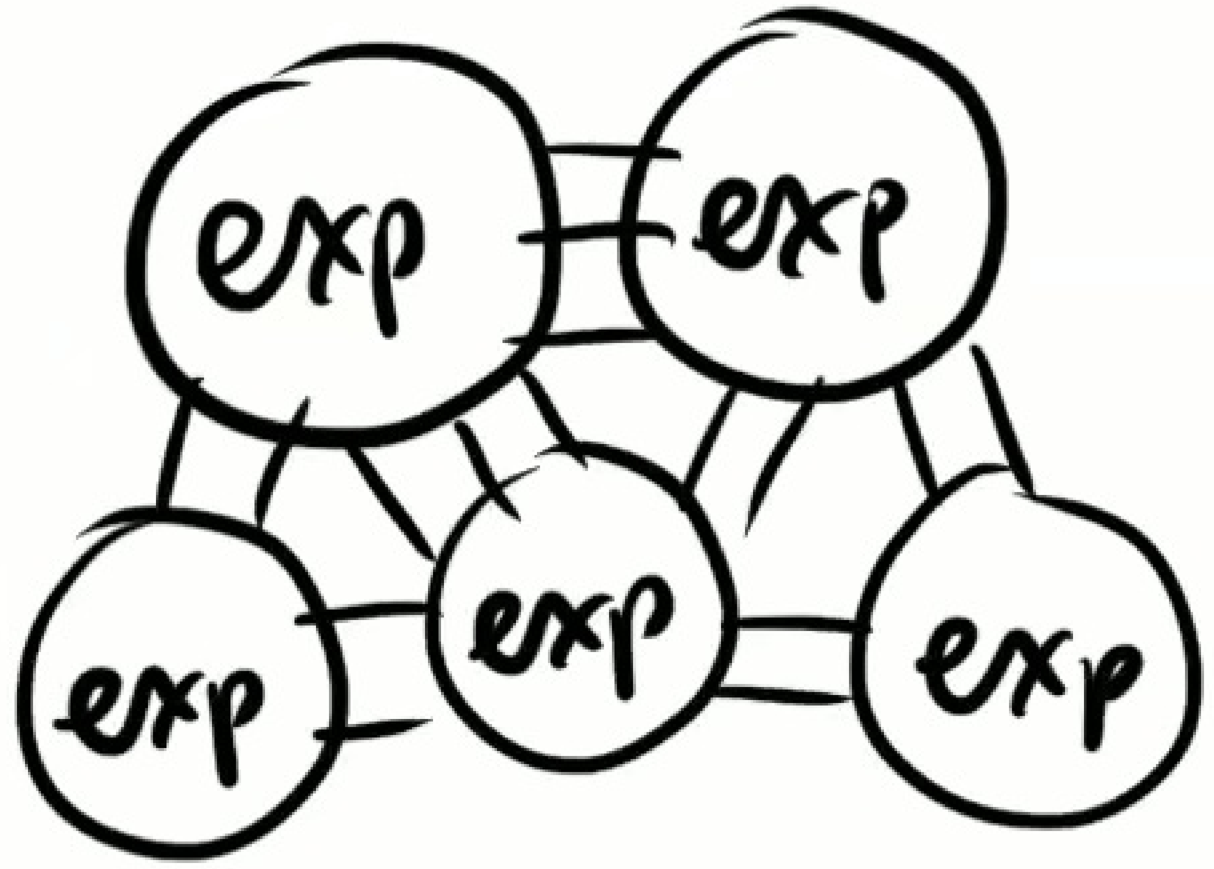
For general graphs, decompose into expanders, solve on each expander, and recurse



Expander Decomposition

Solve when graph is an expander (easy case)

For general graphs, decompose into expanders, solve on each expander, and recurse

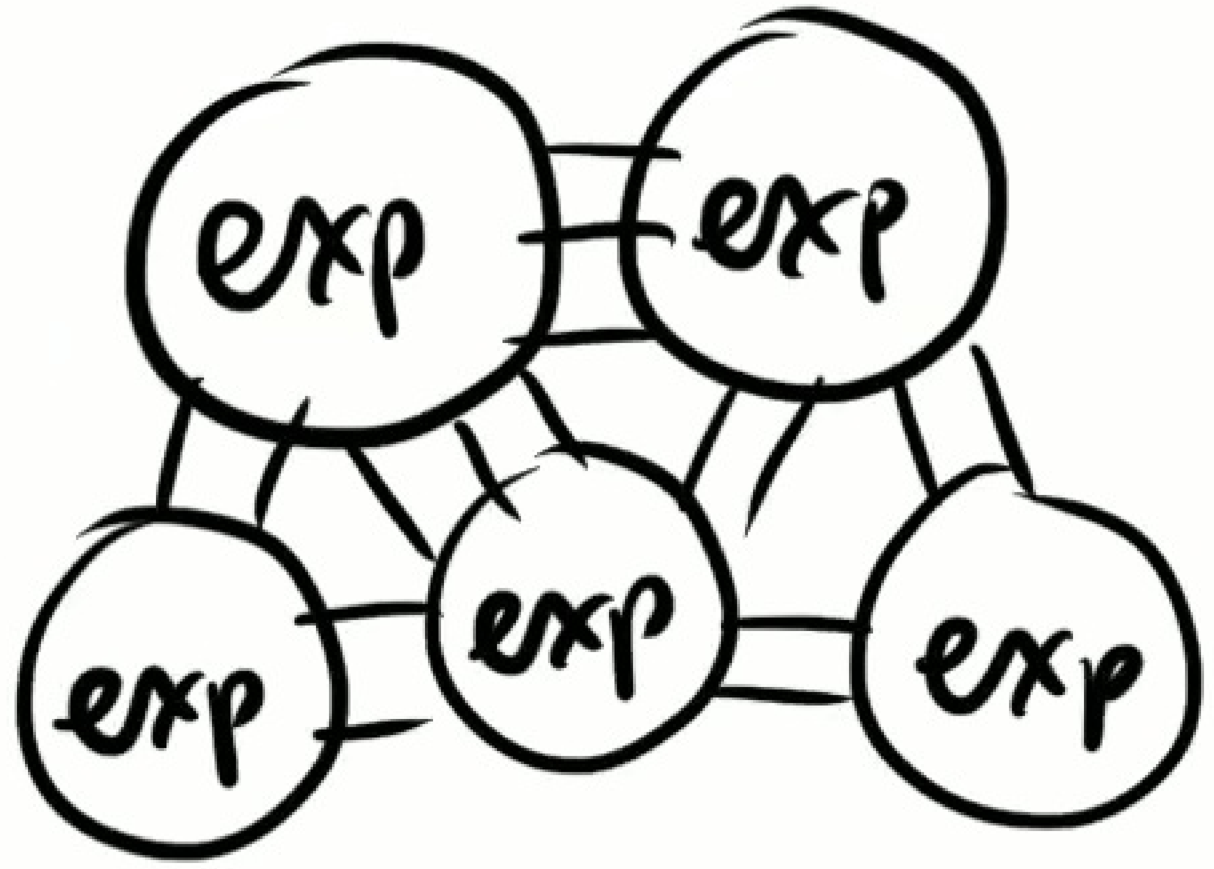


[Chuzhoy, Gao, L., Nanongkai, Peng, Saranurak]: can compute an expander decomposition deterministically in $m^{1+o(1)}$ time

Expander Decomposition

Solve when graph is an expander (easy case)

For general graphs, decompose into expanders, solve on each expander, and recurse



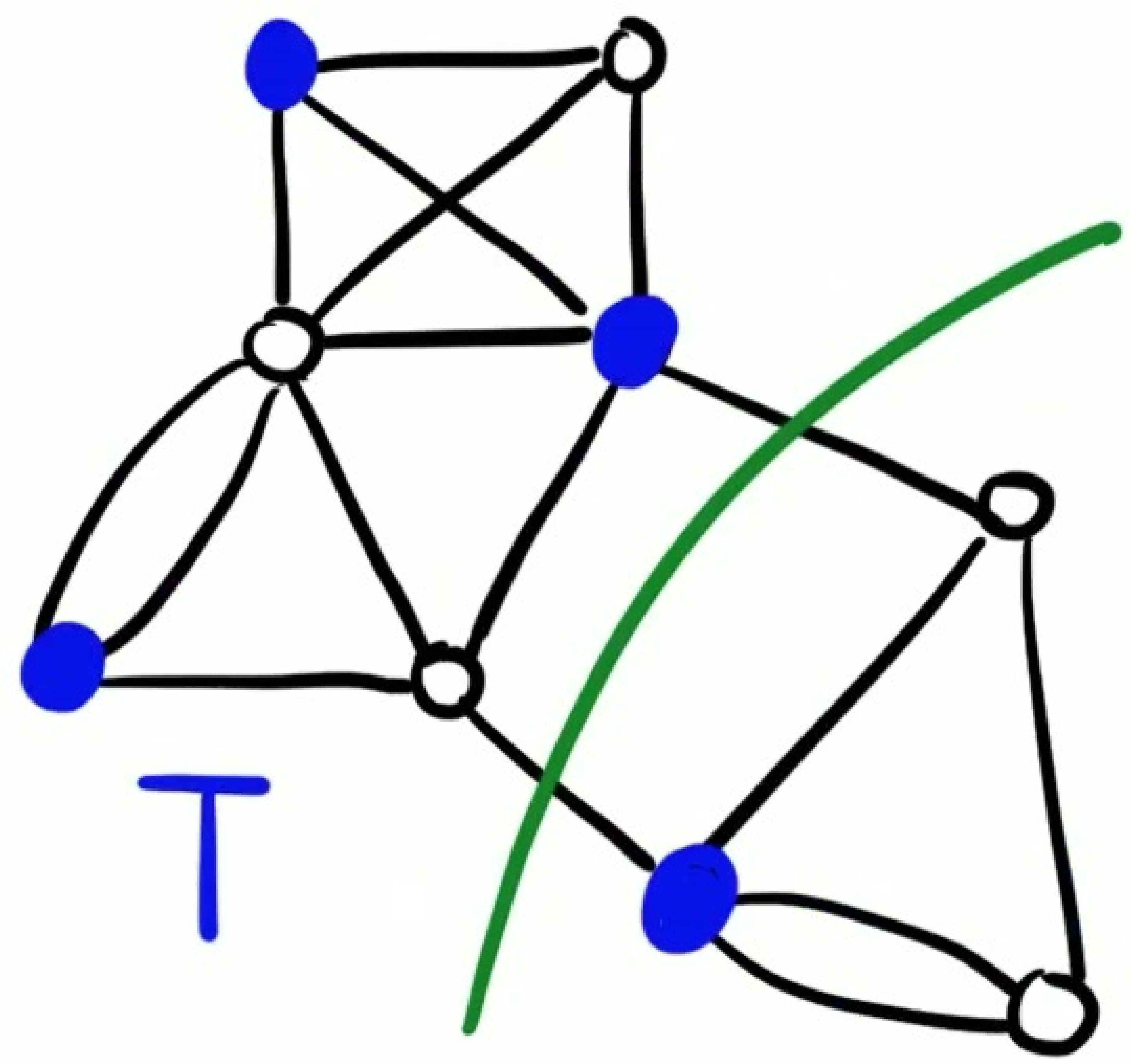
[Chuzhoy, Gao, L., Nanongkai, Peng, Saranurak]: can compute an expander decomposition deterministically in $m^{1+o(1)}$ time

For min-cut: when graph is an expander, the min-cut must be unbalanced! (polylog(n) vertices on smaller side.) So local algorithm works.

Local algorithm: Isolators

Suppose (S, V\S) is a min-cut.

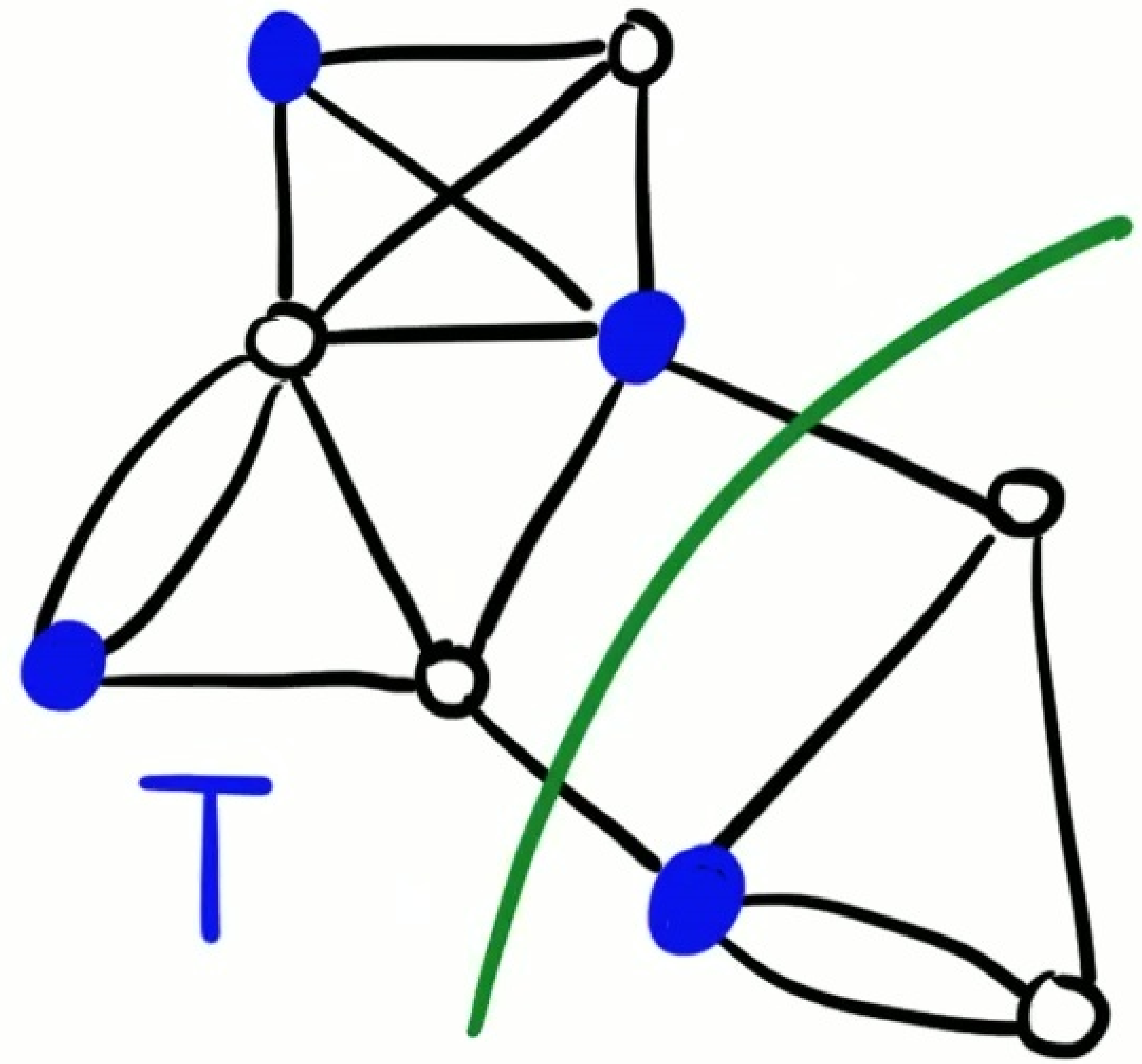
$T \subseteq V$ is an isolator if $|S \cap T| = 1$



Local algorithm: Isolators

Suppose $(S, V \setminus S)$ is a min-cut.

$T \subseteq V$ is an <u>isolator</u> if $|S \cap T| = 1$

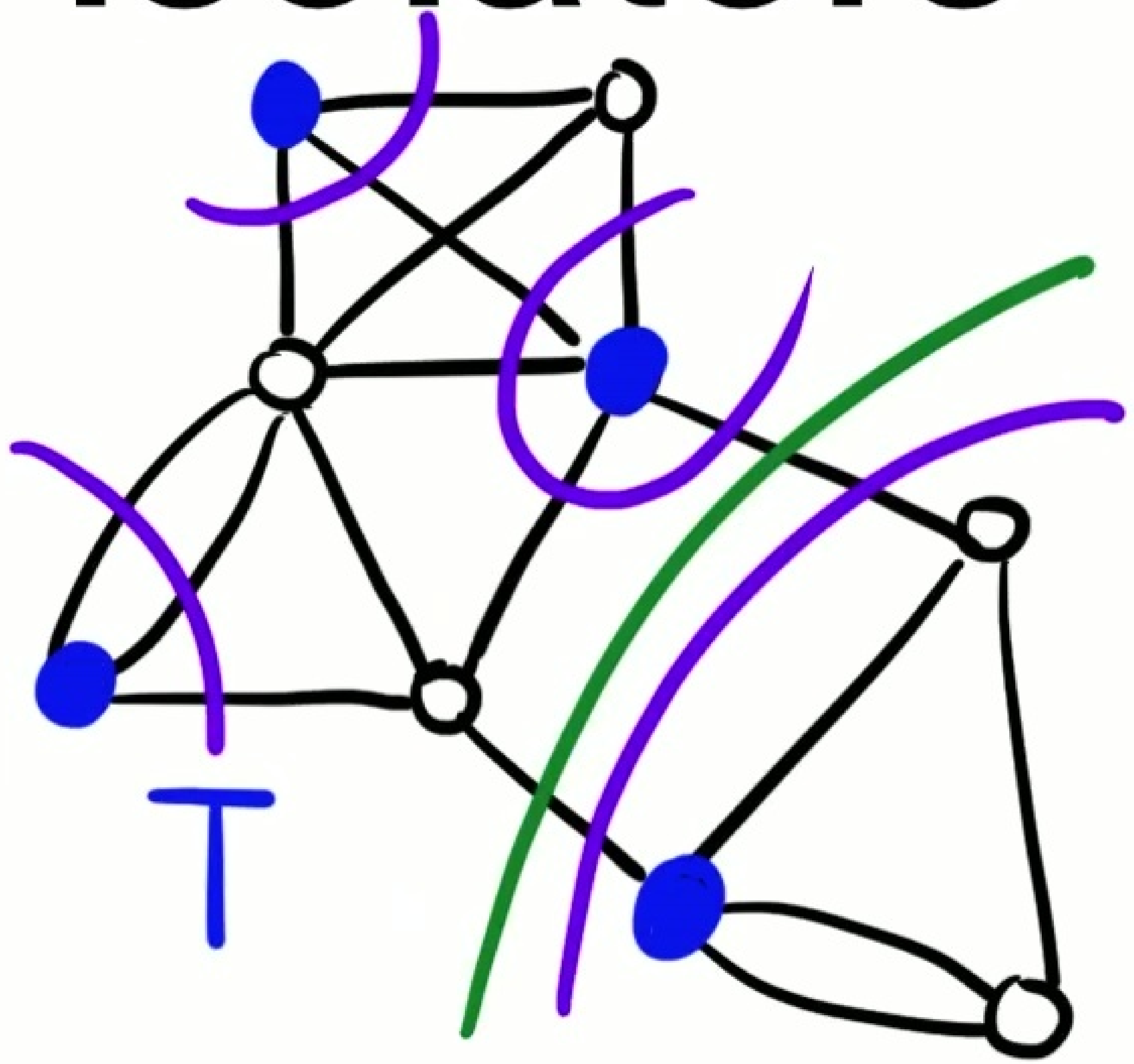


Given T, can compute $(t, T \setminus t)$-min-cut for all $t \in T$ in $O(\log |T|)$ total max-flow calls

Local algorithm: Isolators

Suppose $(S, V\setminus S)$ is a min-cut.

$T \subseteq V$ is an <u>isolator</u> if $|S \cap T| = 1$

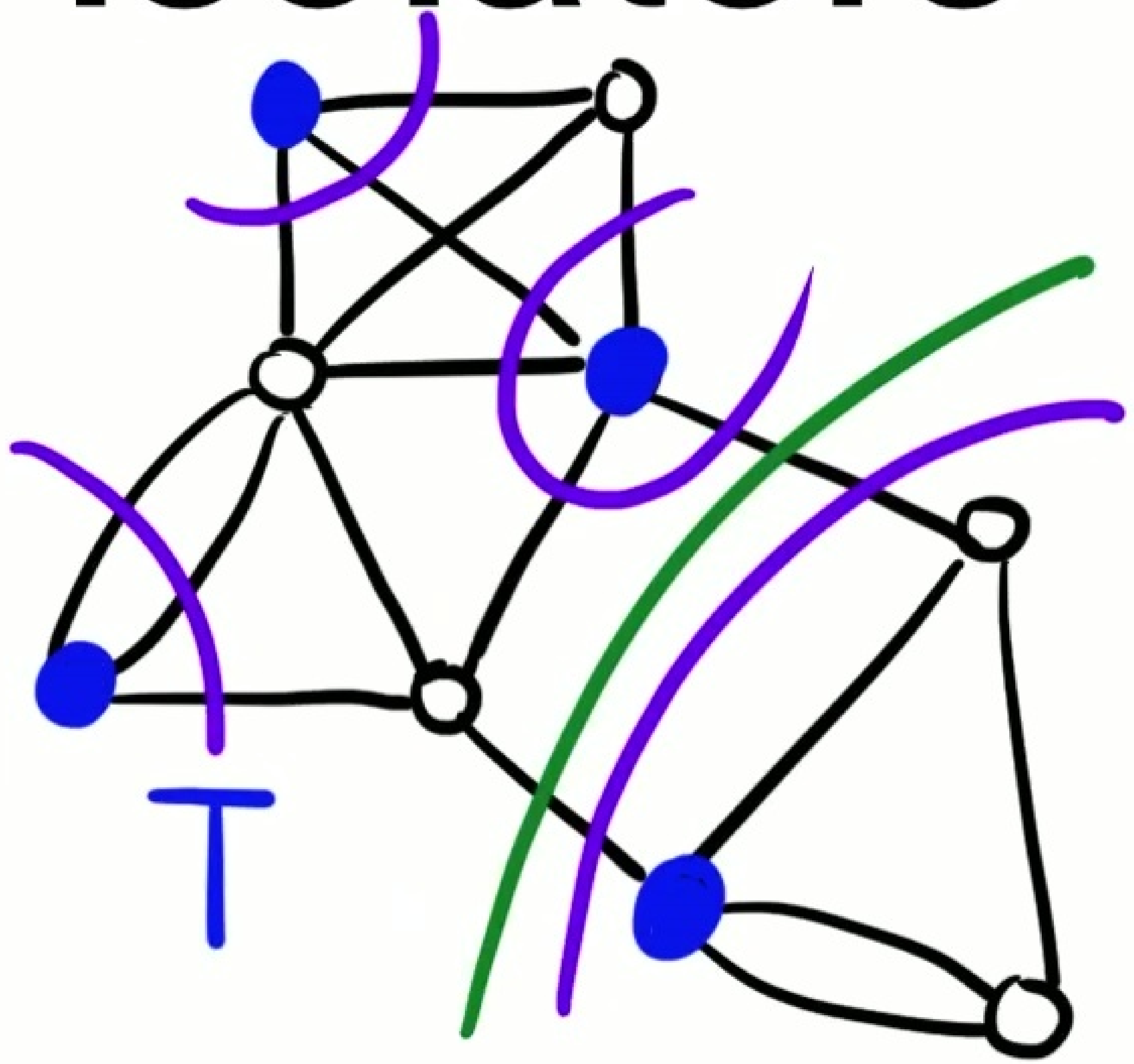


Given T, can compute $(t, T\setminus t)$-min-cut for all $t \in T$ in $O(\log |T|)$ total max-flow calls

Local algorithm: Isolators

Suppose $(S, V \setminus S)$ is a min-cut.

$T \subseteq V$ is an <u>isolator</u> if $|S \cap T| = 1$



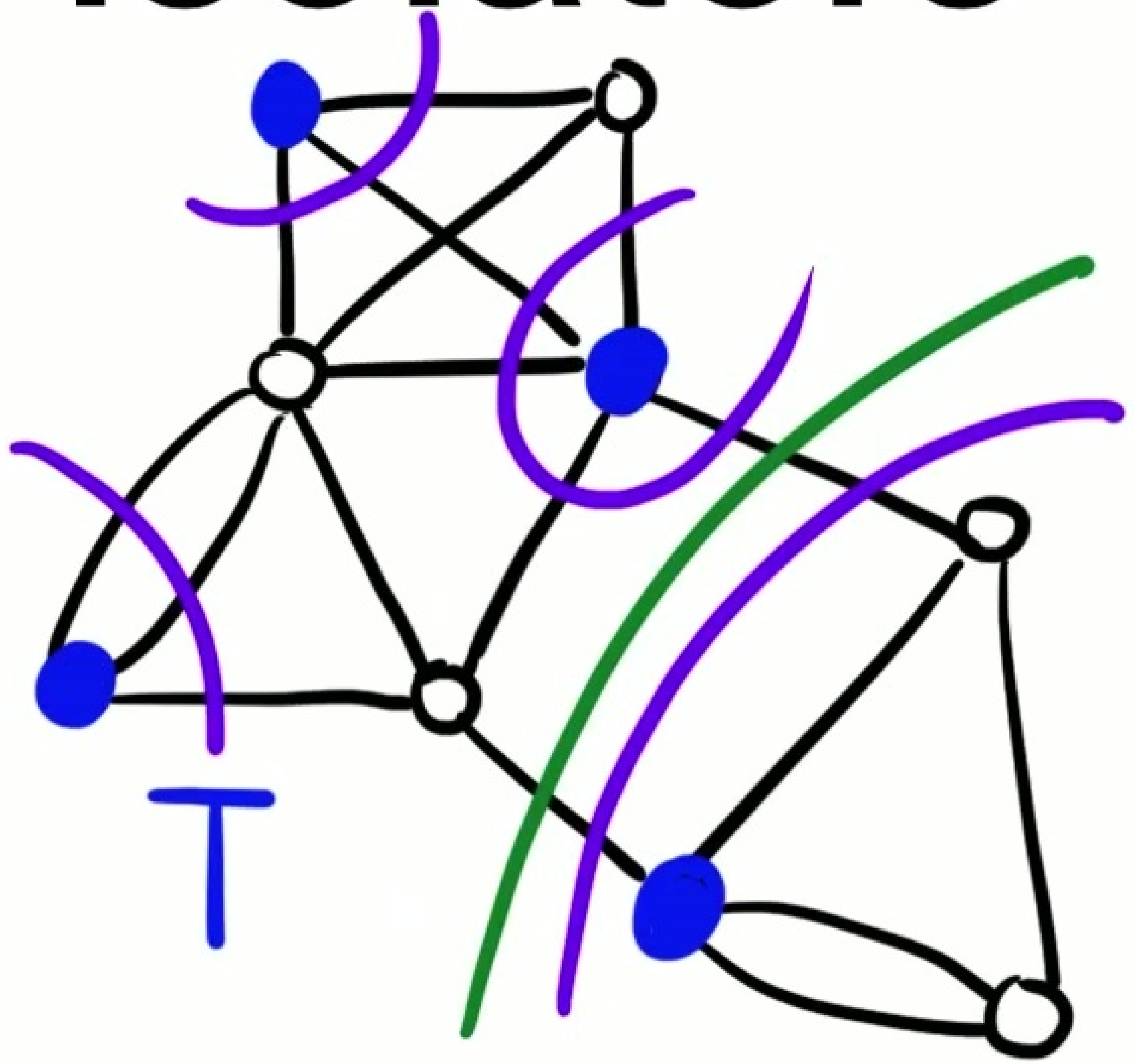
Given T, can compute $(t, T \setminus t)$-min-cut for all $t \in T$ in

$O(\log |T|)$ total max-flow calls "amortized sublinear" over all $t \in T$

Local algorithm: Isolators

Suppose (S, V\S) is a min-cut.

$T \subseteq V$ is an isolator if $|S \cap T| = 1$



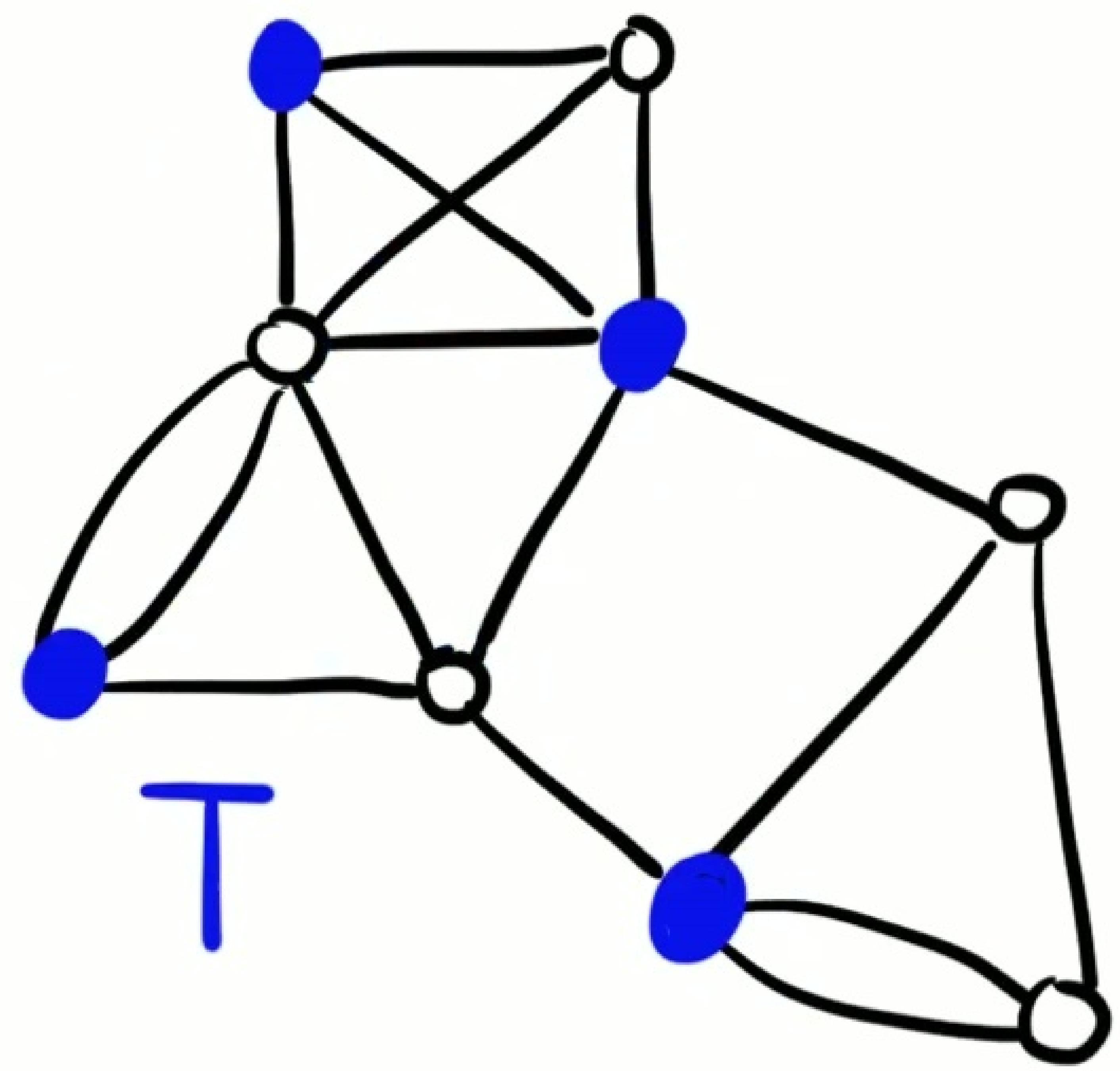
Given T, can compute (t, T\t)-min-cut for all $t \in T$ in

O(log |T|) total max-flow calls "amortized sublinear" over all $t \in T$

If T is an isolator, then one of the cuts returned is the min-cut!

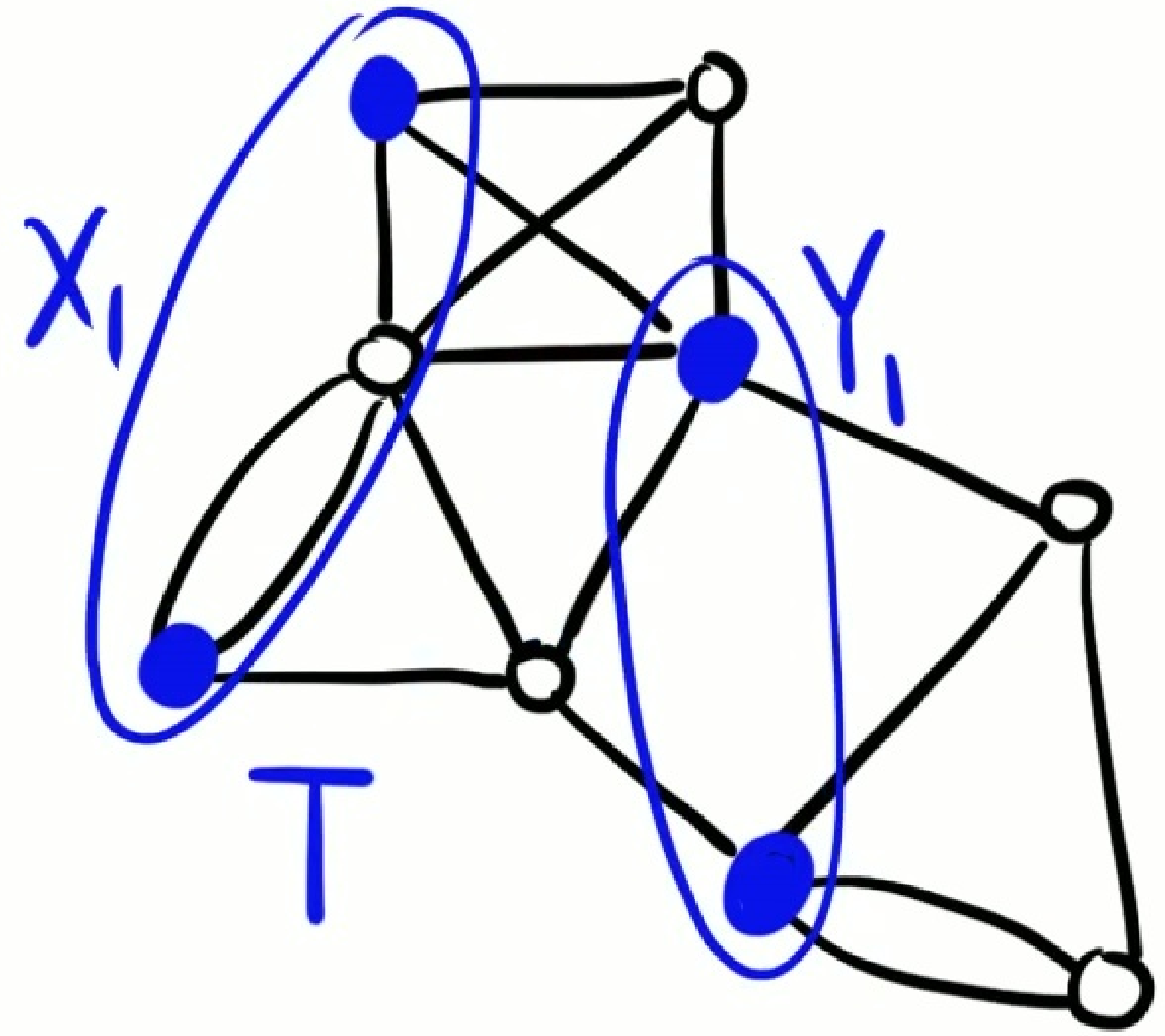
Local algorithm: Isolators

- Compute log |T| bipartitions of T, (X_k, Y_k)
  - Want: each pair s,t in T is separated in at least one of them
  - Encode vertices in T using $\log_2|T|$ bits; for each bit position k, X_k = vertices with k^{th} bit 0, Y_k = vertices with k^{th} bit 1



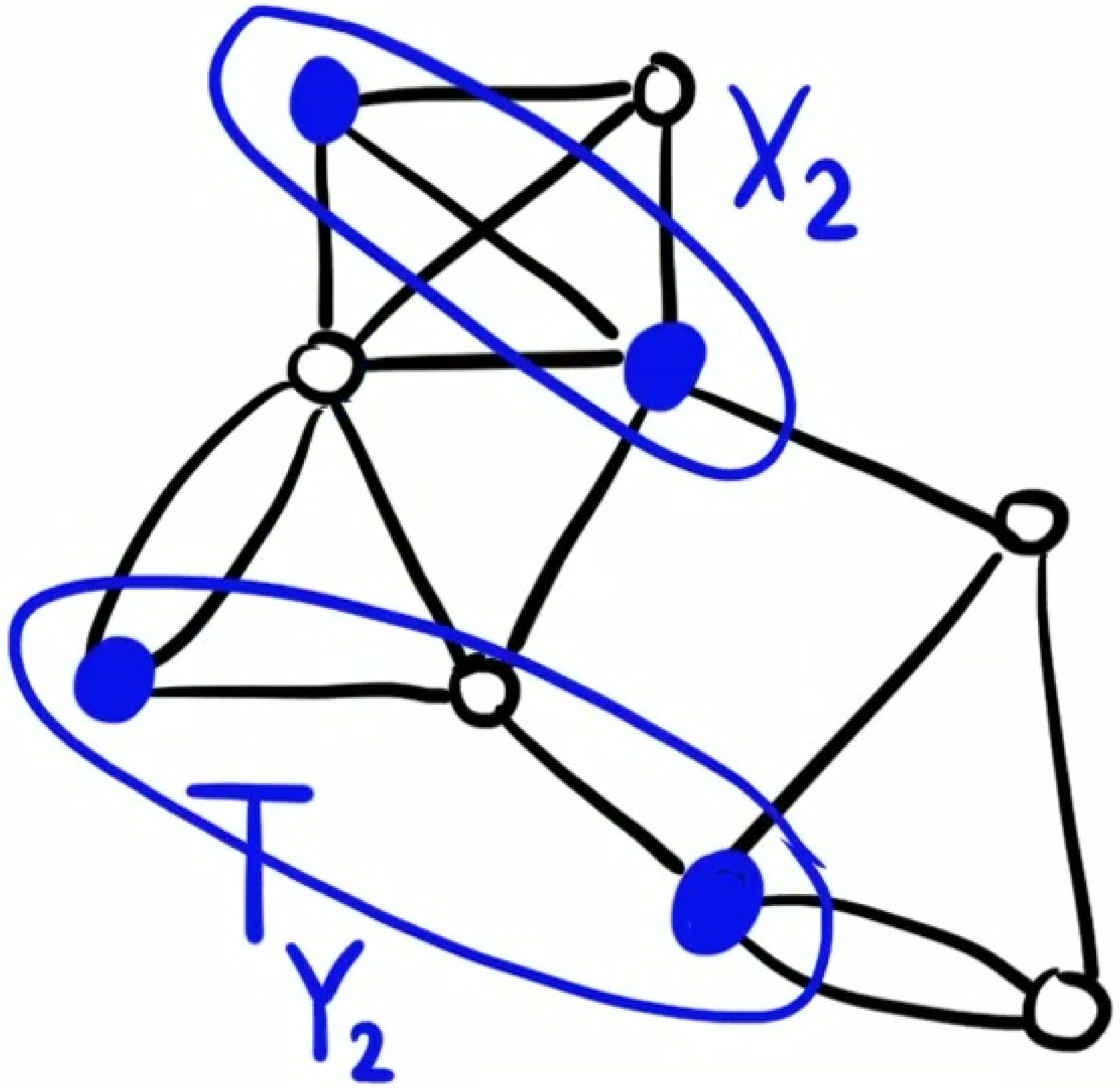
Local algorithm: Isolators

- Compute log |T| bipartitions of T, (X_k, Y_k)
  - Want: each pair s,t in T is separated in at least one of them
  - Encode vertices in T using $\log_2|T|$ bits; for each bit position k, X_k = vertices with k^{th} bit 0, Y_k = vertices with k^{th} bit 1



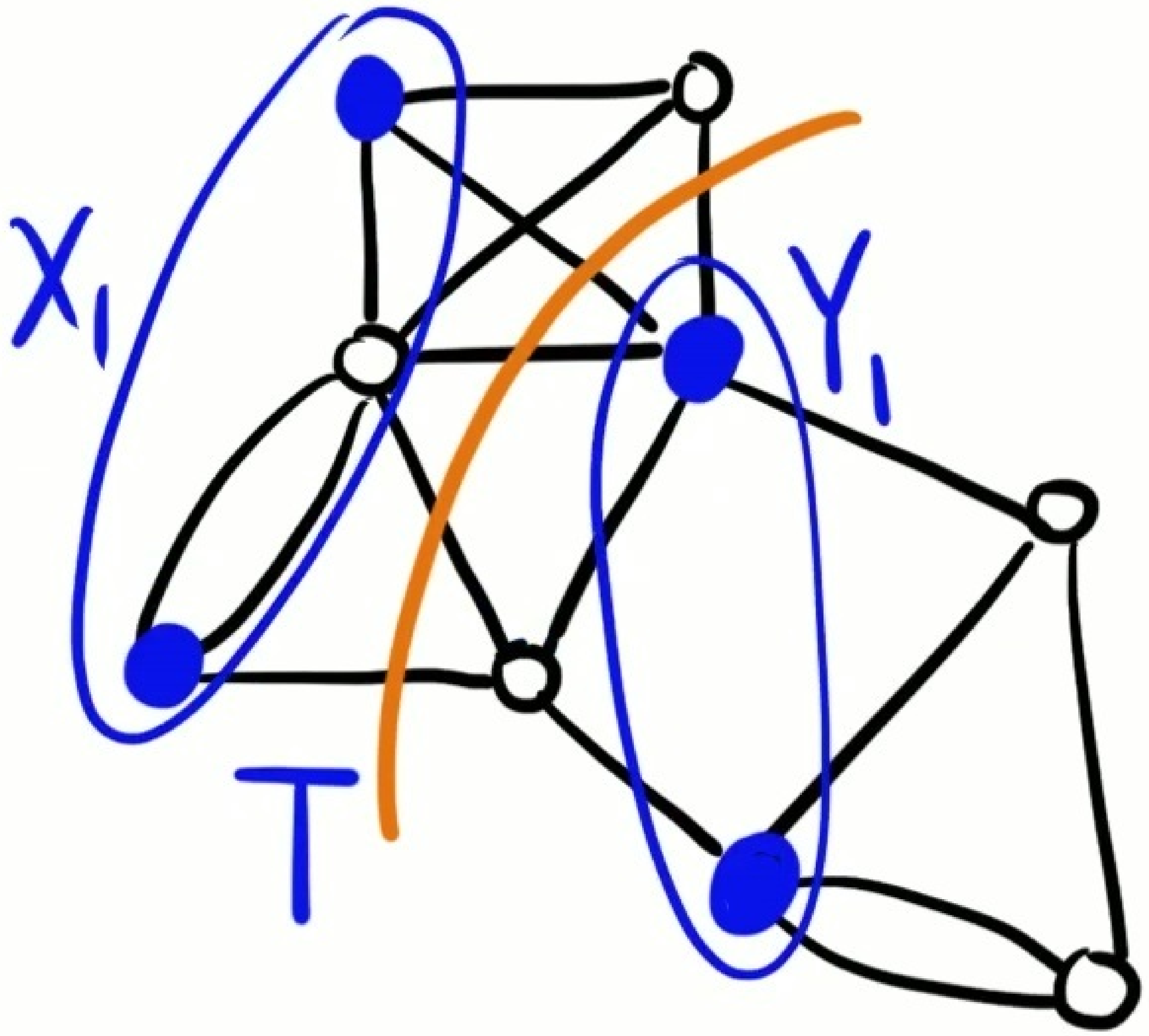
Local algorithm: Isolators

- Compute log |T| bipartitions of T, (X_k, Y_k)
  - Want: each pair s,t in T is separated in at least one of them
  - Encode vertices in T using $\log_2|T|$ bits; for each bit position k, X_k = vertices with k^{th} bit 0, Y_k = vertices with k^{th} bit 1



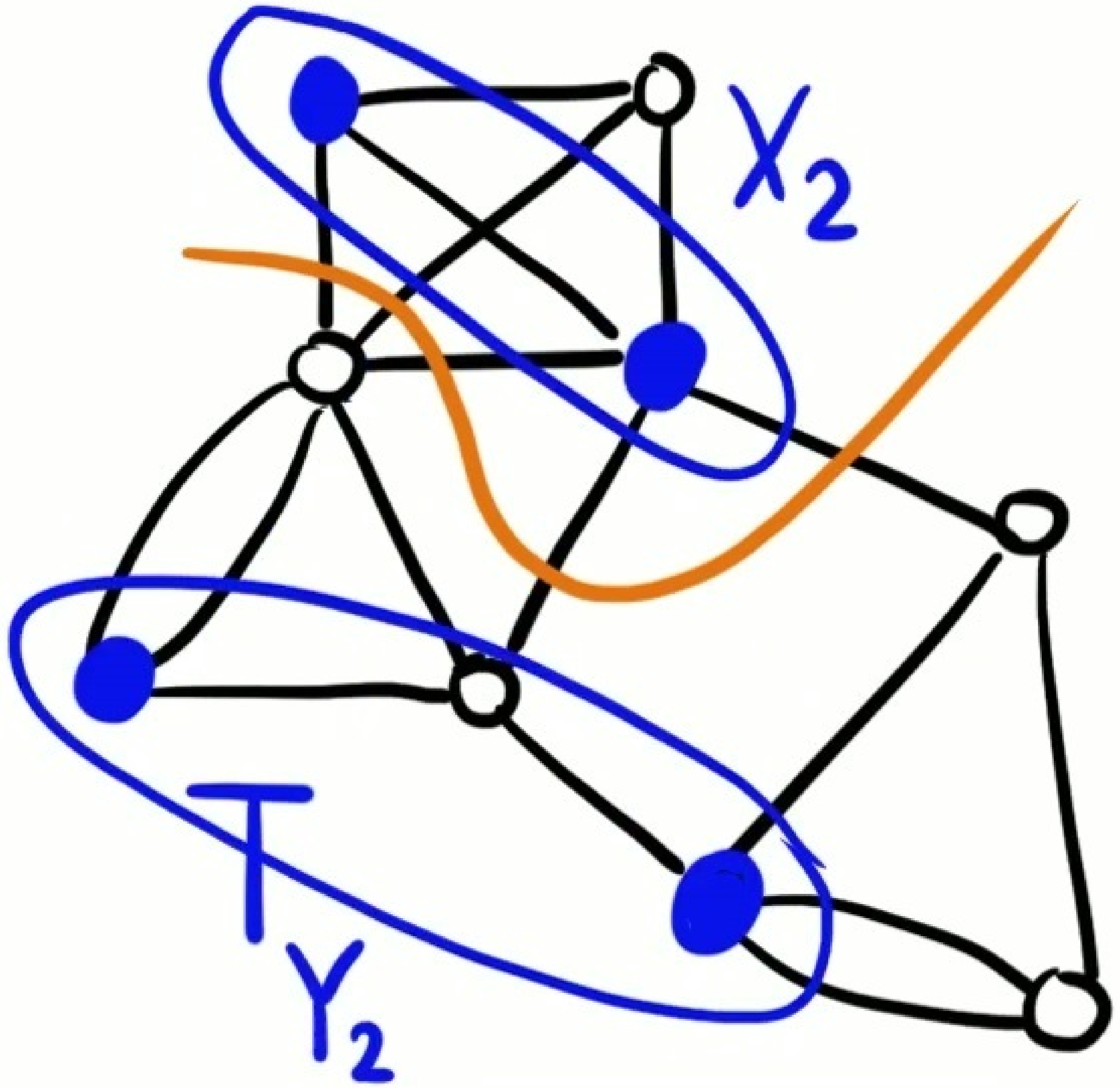
Local algorithm: Isolators

- Compute log |T| bipartitions of T, (X_k, Y_k)
  - Want: each pair s,t in T is separated in at least one of them
  - Encode vertices in T using $\log_2 |T|$ bits; for each bit position k, X_k = vertices with k^{th} bit 0, Y_k = vertices with k^{th} bit 1
- For each k, compute (X_k, Y_k)-min-cut



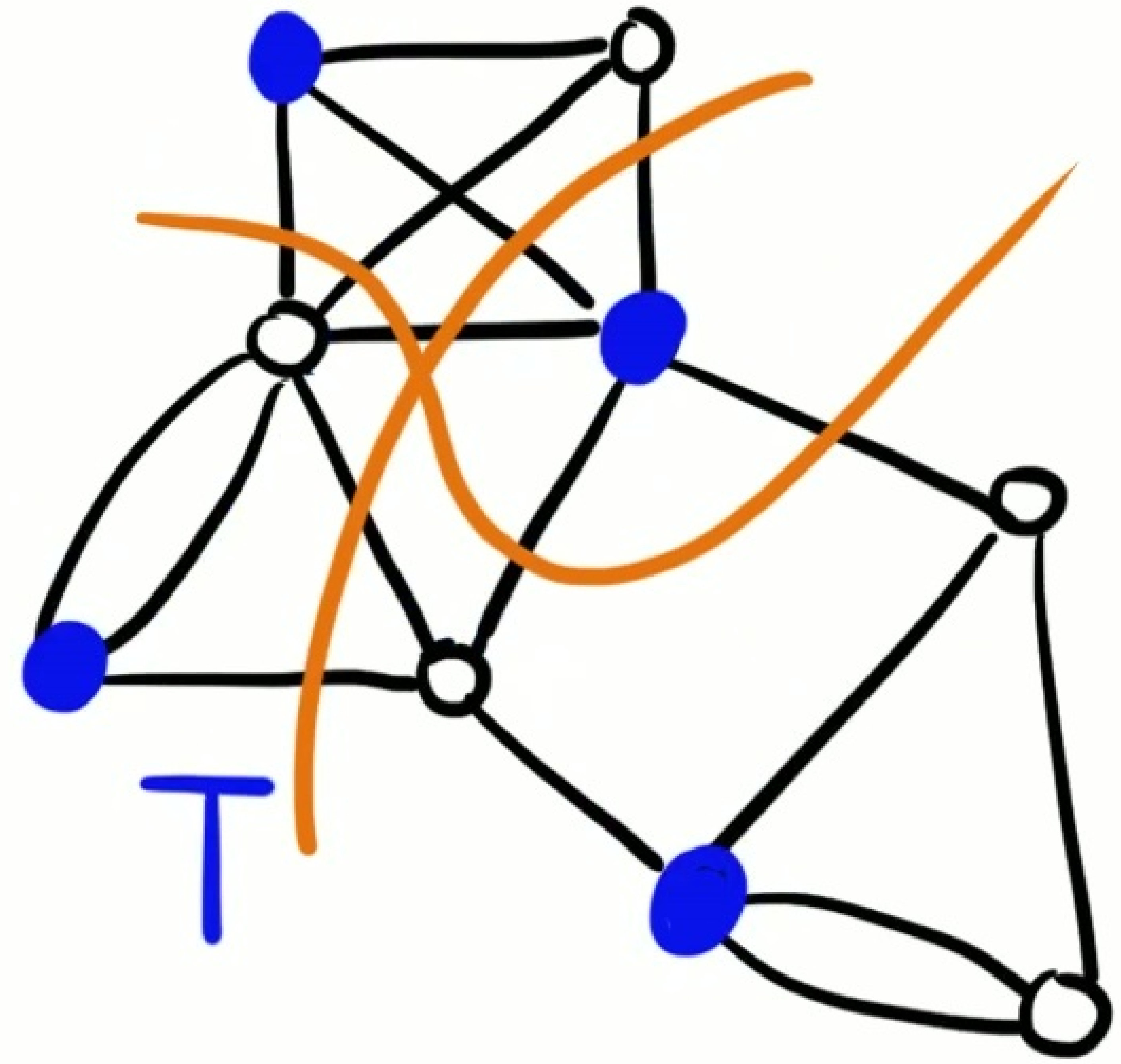
Local algorithm: Isolators

- Compute log |T| bipartitions of T, (X_k, Y_k)
  - Want: each pair s,t in T is separated in at least one of them
  - Encode vertices in T using $\log_2|T|$ bits; for each bit position k, X_k = vertices with k^{th} bit 0, Y_k = vertices with k^{th} bit 1
- For each k, compute (X_k, Y_k)-min-cut



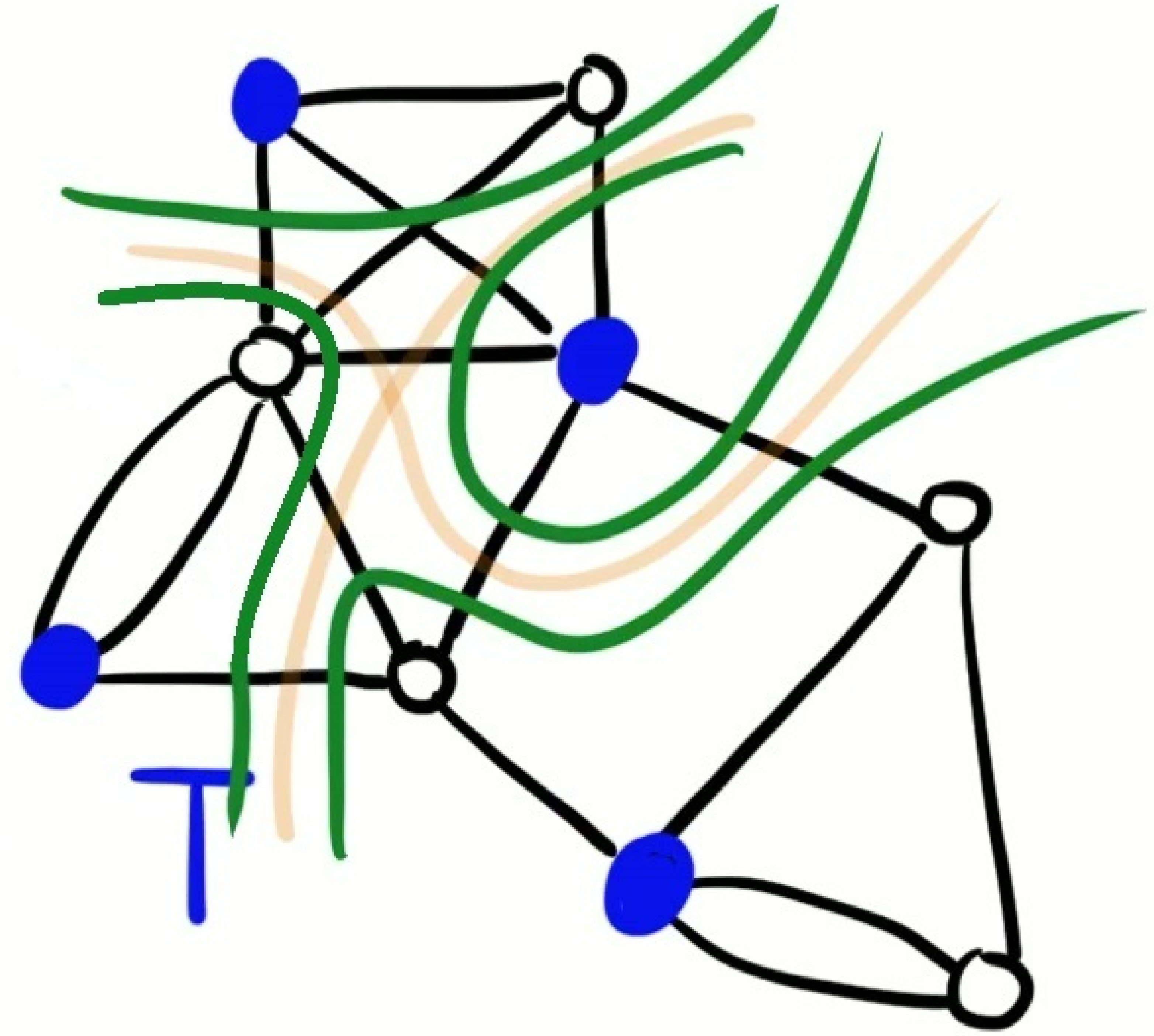
Local algorithm: Isolators

- Compute log |T| bipartitions of T, (X_k, Y_k)
  - Want: each pair s,t in T is separated in at least one of them
  - Encode vertices in T using $\log_2|T|$ bits; for each bit position k, X_k = vertices with k^{th} bit 0, Y_k = vertices with k^{th} bit 1
- For each k, compute (X_k, Y_k)-min-cut
- Union of min-cuts separates all of T



Local algorithm: Isolators

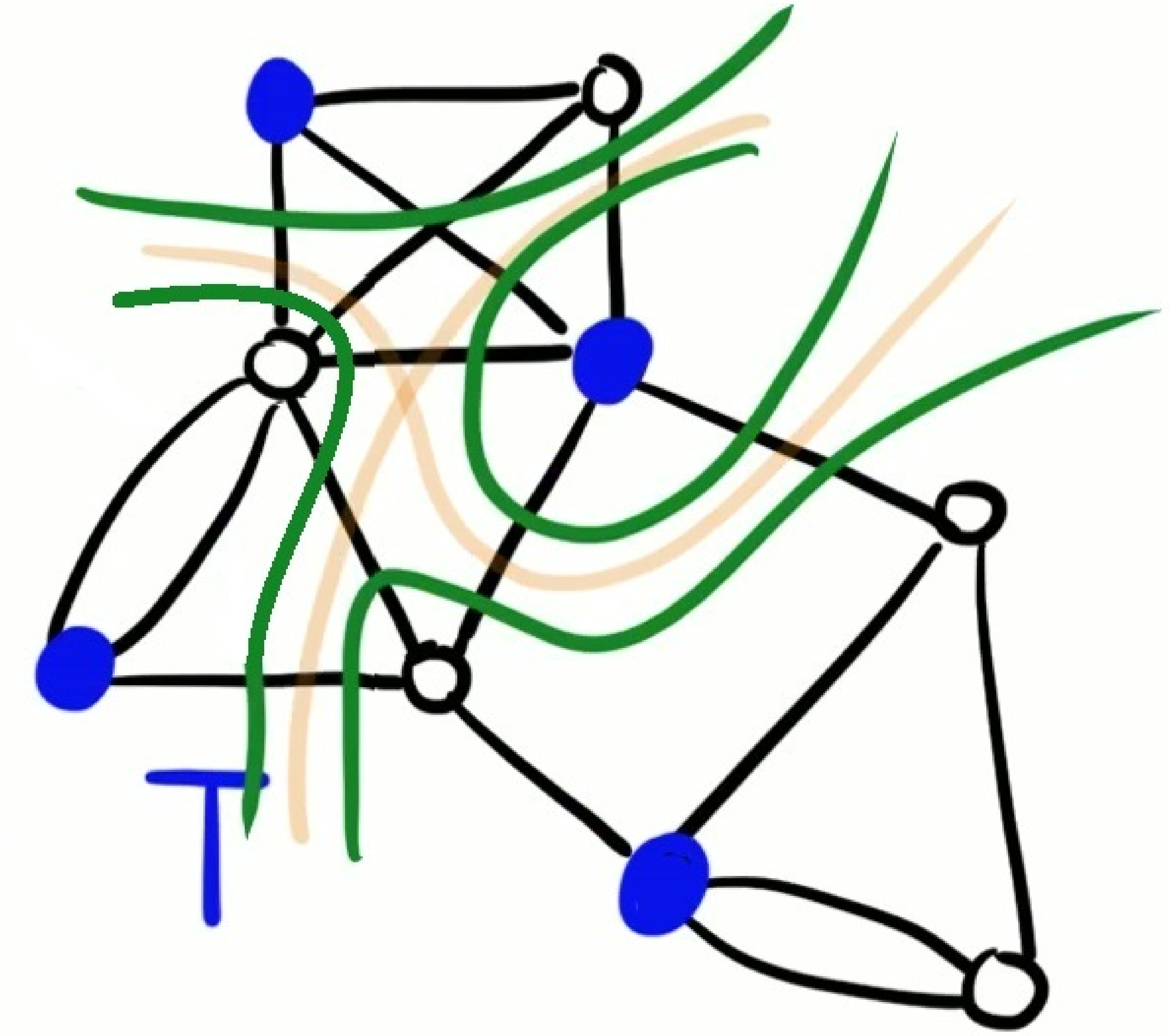
- Compute log |T| bipartitions of T, (X_k, Y_k)
  - Want: each pair s,t in T is separated in at least one of them
  - Encode vertices in T using $\log_2 |T|$ bits; for each bit position k, X_k = vertices with k^{th} bit 0, Y_k = vertices with k^{th} bit 1
- For each k, compute (X_k, Y_k)-min-cut
- Union of min-cuts separates all of T



Local algorithm: Isolators

- Compute log |T| bipartitions of T, (X_k, Y_k)
  - Want: each pair s,t in T is separated in at least one of them
  - Encode vertices in T using $\log_2|T|$ bits; for each bit position k, X_k = vertices with k^{th} bit 0, Y_k = vertices with k^{th} bit 1
- For each k, compute (X_k, Y_k)-min-cut
- Union of min-cuts separates all of T

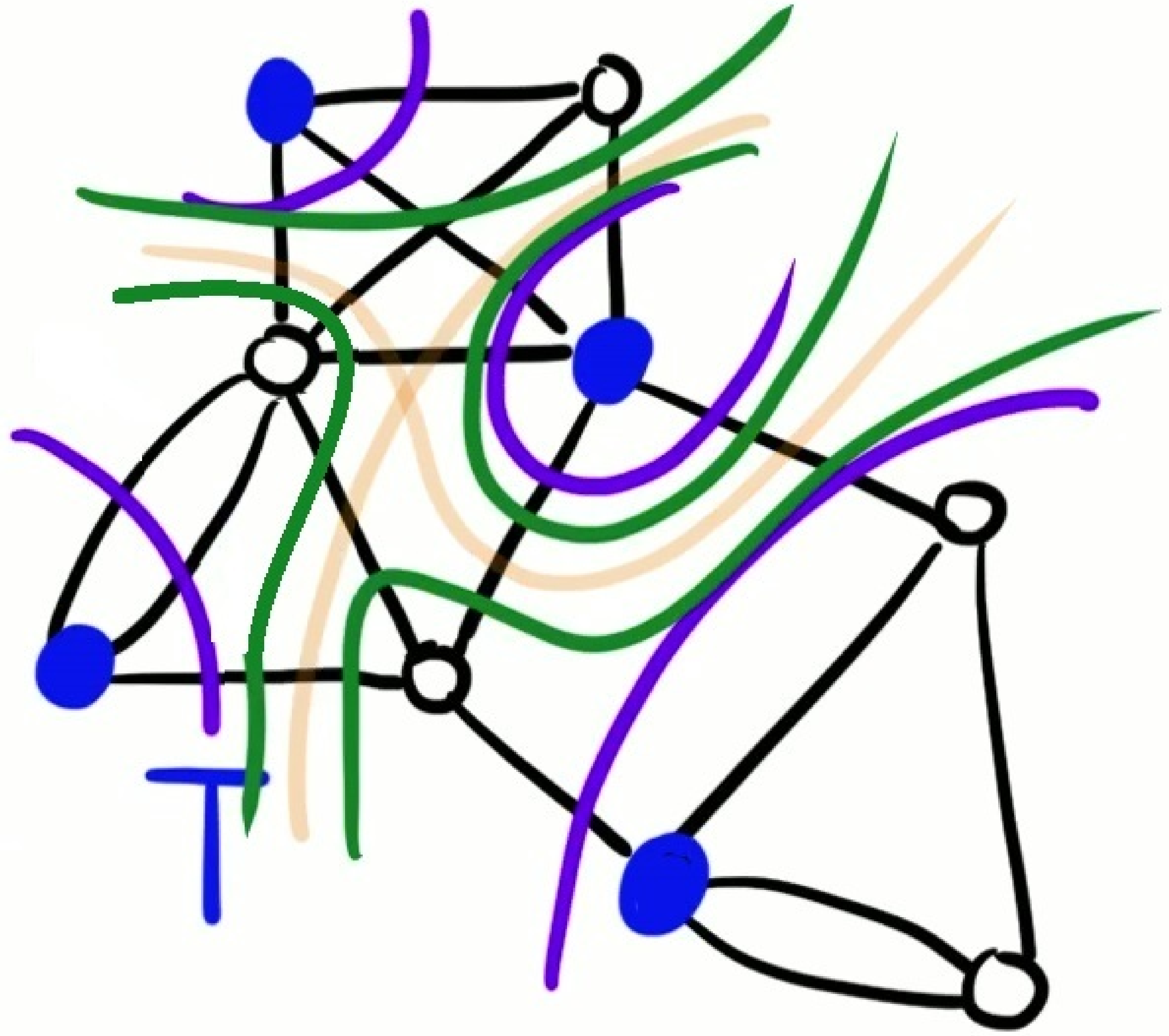
Claim: (t, T\t)-min-cut "contained" in t's piece ("uncrossing" argument)



Local algorithm: Isolators

- Compute log |T| bipartitions of T, (X_k, Y_k)
  - Want: each pair s,t in T is separated in at least one of them
  - Encode vertices in T using $\log_2|T|$ bits; for each bit position k, X_k = vertices with k^{th} bit 0, Y_k = vertices with k^{th} bit 1
- For each k, compute (X_k, Y_k)-min-cut
- Union of min-cuts separates all of T

Claim: (t, T\t)-min-cut "contained" in t's piece ("uncrossing" argument)

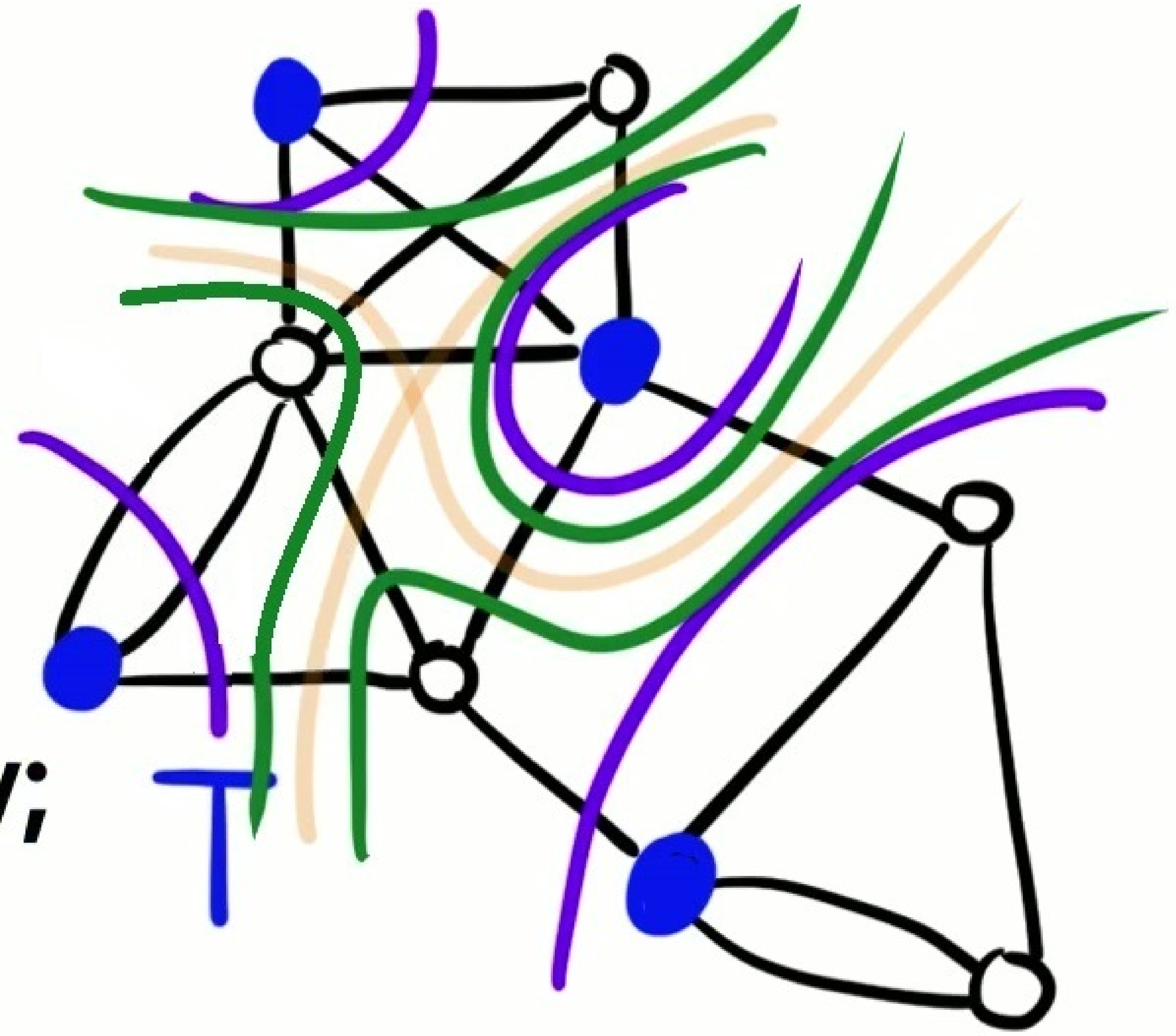


Local algorithm: Isolators

- Compute log |T| bipartitions of T, (X_k, Y_k)
  - Want: each pair s,t in T is separated in at least one of them
  - Encode vertices in T using $\log_2|T|$ bits; for each bit position k, X_k = vertices with k^{th} bit 0, Y_k = vertices with k^{th} bit 1
- For each k, compute (X_k, Y_k)-min-cut
- Union of min-cuts separates all of T

Claim: (t, T\t)-min-cut "contained" in t's piece ("uncrossing" argument)

- Contract outside of t's piece, run max-flow; total O(m) edges over all t

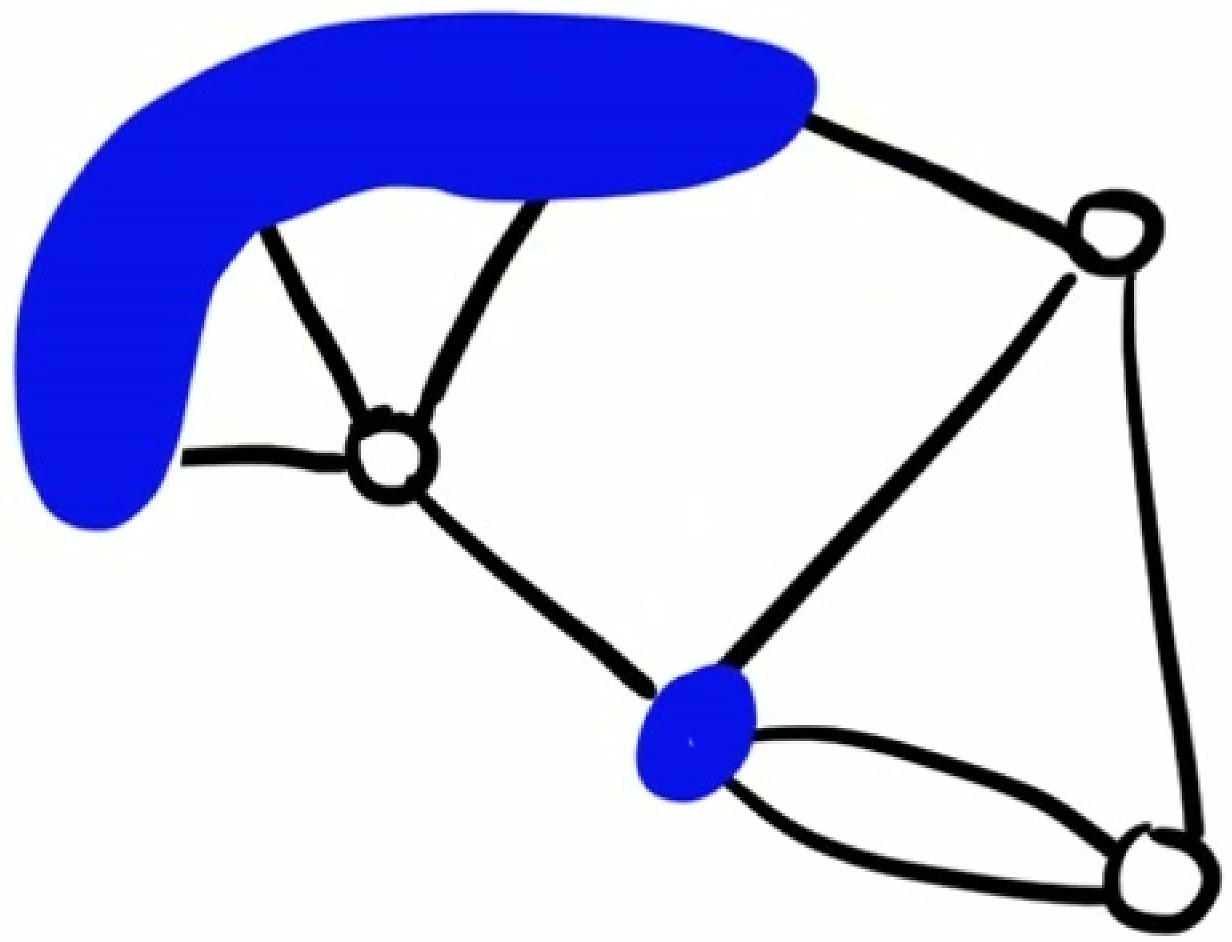


Local algorithm: Isolators

- Compute log |T| bipartitions of T, (X_k, Y_k)
  - Want: each pair s,t in T is separated in at least one of them
  - Encode vertices in T using $\log_2|T|$ bits; for each bit position k, X_k = vertices with k^{th} bit 0, Y_k = vertices with k^{th} bit 1
- For each k, compute (X_k, Y_k)-min-cut
- Union of min-cuts separates all of T

Claim: (t, T\t)-min-cut “contained” in t’s piece (“uncrossing” argument)

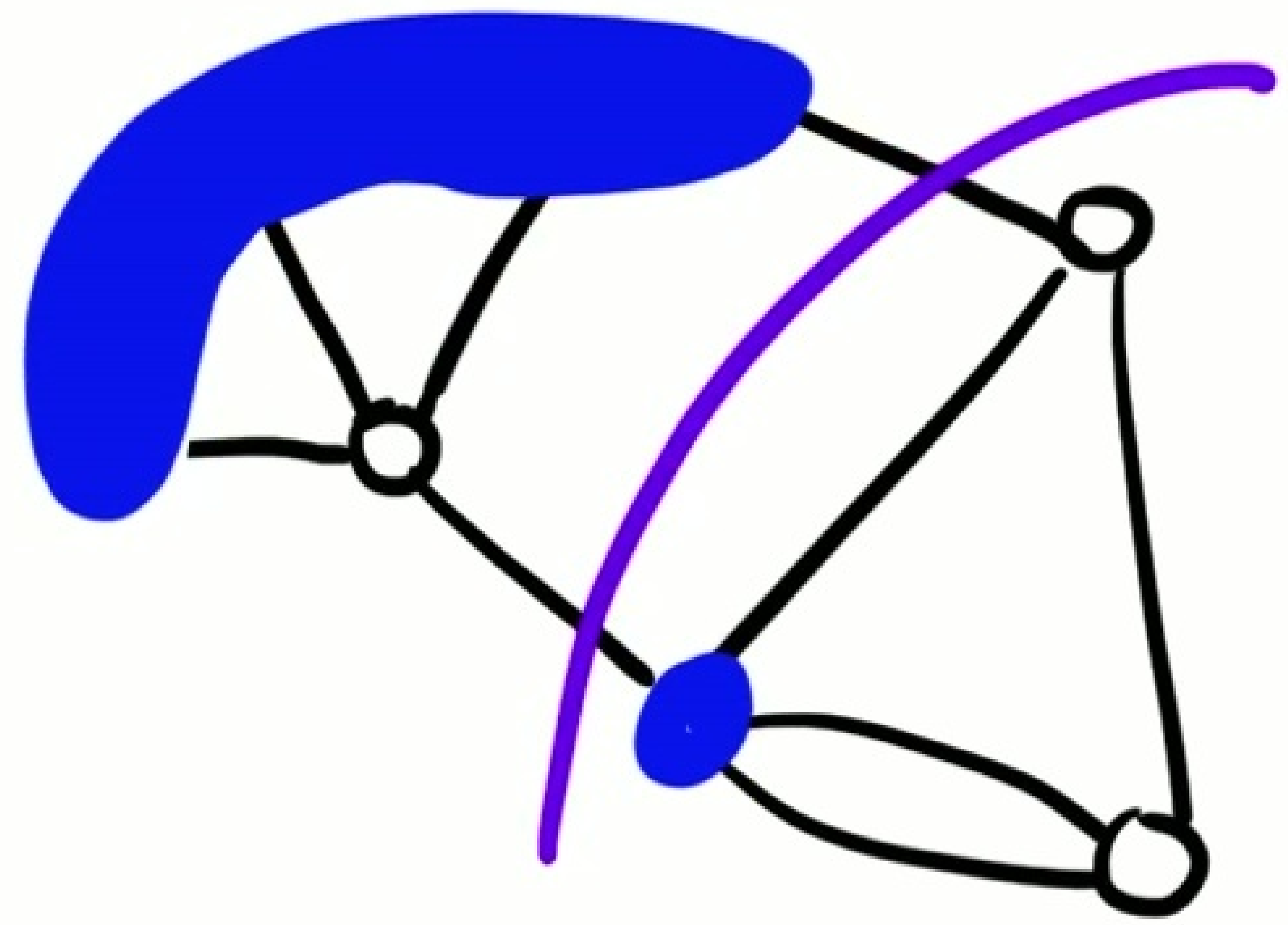
- Contract outside of t’s piece, run max-flow; total O(m) edges over all t



Local algorithm: Isolators

- Compute log |T| bipartitions of T, (X_k, Y_k)
  - Want: each pair s,t in T is separated in at least one of them
  - Encode vertices in T using $\log_2|T|$ bits; for each bit position k, X_k = vertices with k^{th} bit 0, Y_k = vertices with k^{th} bit 1
- For each k, compute (X_k, Y_k)-min-cut
- Union of min-cuts separates all of T

Claim: (t, T\t)-min-cut "contained" in t's piece ("uncrossing" argument)



- Contract outside of t's piece, run max-flow; total O(m) edges over all t

Simple Randomized Min-cut

Let $(S, V \setminus S)$ be the min-cut

- Guess $|S| \sim 2^i$ for some i

Simple Randomized Min-cut

Let (S, V\S) be the min-cut

- Guess |S| ~ 2^i for some i
- Sample each vertex with probability $1/2^i$
  - With constant probability, sample exactly one in S

    => sampled set T is an isolator

Simple Randomized Min-cut

Let (S, V\S) be the min-cut

- Guess |S| ~ 2^i for some i
- Sample each vertex with probability $1/2^i$
  - With constant probability, sample exactly one in S
    => sampled set T is an isolator
- Run the isolator algorithm for all i, output smallest (t, T\t)-min-cut found

Almost-Isolator

$(S, V \setminus S)$ is the min-cut

T is an almost-isolator if $|S \cap T| < k$ (k = polylog(n))

Almost-Isolator

$(S, V\setminus S)$ is the min-cut

T is an almost-isolator if $|S \cap T| < k$ (k = polylog(n))

Given an almost-isolator, subsampling with probability $1/k$ $O(\log n)$ times produces an isolator

Almost-Isolator

$(S, V \setminus S)$ is the min-cut

T is an almost-isolator if $|S \cap T| < k$ ($k = \text{polylog}(n)$)

Given an almost-isolator, subsampling with probability $1/k$ $O(\log n)$ times produces an isolator

Derandomization: overhead of $k^{O(1)} \log n$

Almost-Isolator

(S, V\S) is the min-cut

T is an <u>almost-isolator</u> if $|S \cap T| < k$ ($k = \text{polylog}(n)$)

Given an almost-isolator, subsampling with probability 1/k O(log n) times produces an isolator

Derandomization: overhead of $k^{O(1)} \log n$

Unbalanced case: T = V is an almost-isolator if $|S| < \text{polylog}(n)$. Algorithm calls polylog(n) max-flows

Balanced case: det. sparsification

Suppose $|S \cap T|$, $|(V \setminus S) \cap T)| >$ polylog(n) [initially T=V]

Balanced case: det. sparsification

Suppose $|S \cap T|$, $|(V \setminus S) \cap T| >$ polylog(n) [initially T=V]

Goal: find T' s.t. $|T'| < |T|/2$ (sparsification)

$|S \cap T'| > 0$, $|(V \setminus S) \cap T'| > 0$

(still hit both sides)

Balanced case: det. sparsification

Suppose $|S \cap T|$, $|(V \setminus S) \cap T)| >$ polylog(n) [initially T=V]

Goal: find T' s.t. $|T'| < |T|/2$ (sparsification)

$|S \cap T'| > 0$, $|(V \setminus S) \cap T'| > 0$

(still hit both sides)

Algorithm: start with T=V

run unbalanced case, then sparsify T,

repeat until $|T| = 1$

return smallest $(t, T \setminus t)$-min-cut found

Conductance and Expanders

Conductance of an (unweighted) graph: $\Phi(G) = \min_{\substack{S \subseteq V \\ vol(S) \leq vol(V \setminus S)}} \frac{|E(S, V \setminus S)|}{vol(S)}$

$vol(S)$: sum of degrees in S

G is a ϕ-expander if $\Phi(G) > \phi$

Conductance and Expanders

Conductance of an (unweighted) graph: $\Phi(G) = \min_{\substack{S \subseteq V \\ vol(S) \leq vol(V \setminus S)}} \frac{|E(S, V \setminus S)|}{vol(S)}$

↑ sum of degrees in S

G is a ϕ-expander if $\Phi(G) > \phi$

Why are ϕ-expanders easy? [$\phi = \frac{1}{\text{polylog}(n)}$]

Claim: in a ϕ-expander, then $|S| \leq 1/\phi$

Conductance and Expanders

Conductance of an (unweighted) graph: $\Phi(G) = \min_{\substack{S \subseteq V \\ vol(S) \leq vol(V \setminus S)}} \frac{|E(S, V \setminus S)|}{vol(S)}$

↑ sum of degrees in S

G is a ϕ-expander if $\Phi(G) > \phi$

Why are ϕ-expanders easy? [$\phi = \frac{1}{\text{polylog}(n)}$]

Claim: in a ϕ-expander, then $|S| \leq 1/\phi$

Proof: Suppose $vol(S) \leq vol(V \setminus S)$

$$vol(S) = \sum_{v \in S} deg(v) \geq \sum_{v \in S} \lambda = \lambda |S| \quad [\lambda = \text{min-cut}]$$

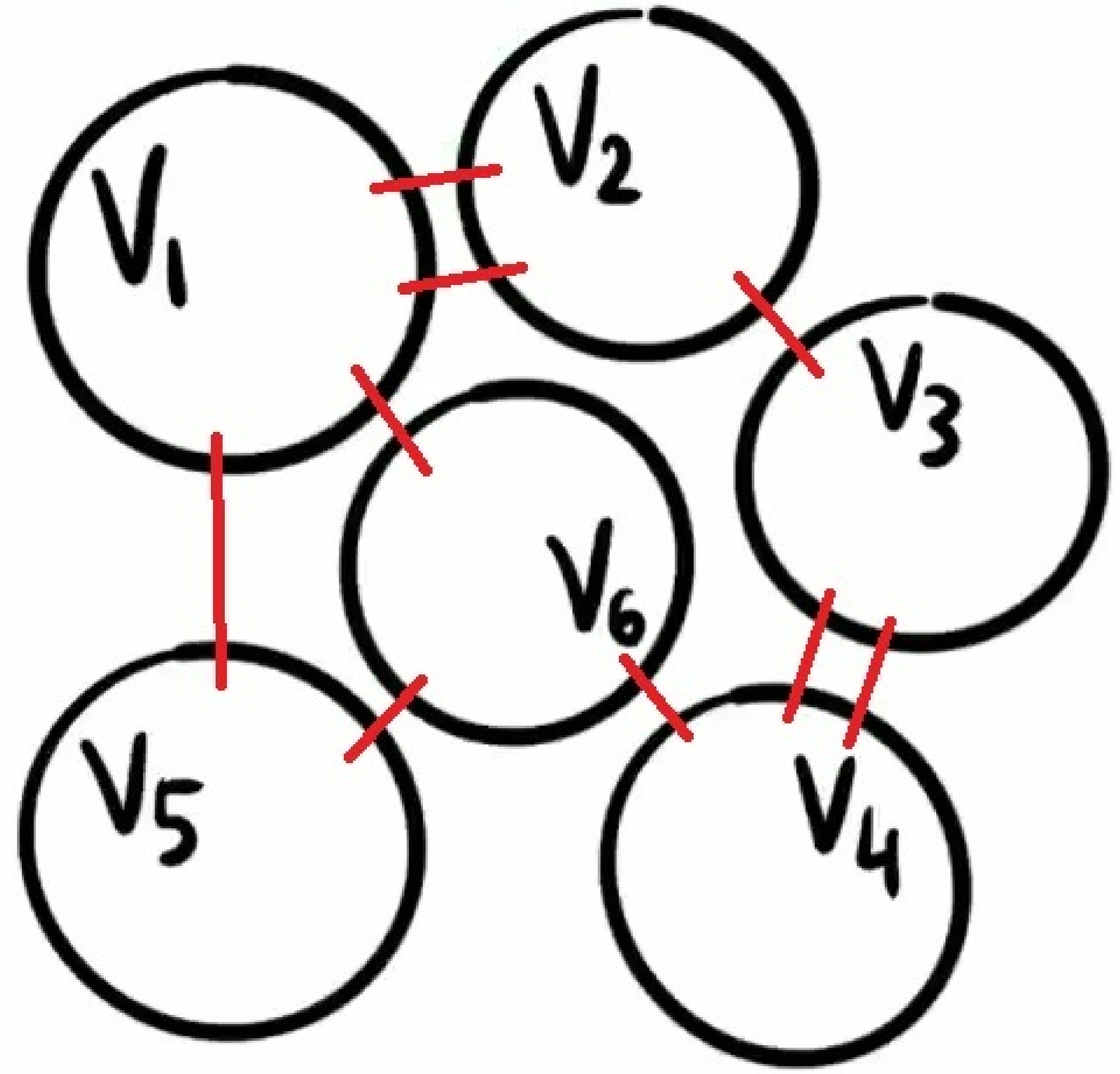
$$\phi \leq \Phi(G) \leq \frac{|E(S, V \setminus S)|}{vol(S)} \leq \frac{\lambda}{\lambda |S|} \iff |S| \leq 1/\phi$$

Deterministic sparsification of T (T=V)

Expander decomposition: partition V into $V_1, \ldots, V_k$ s.t.

(1) Each induced graph $G[V_i]$ is a ϕ-expander

(2) At most half of edges go between expanders



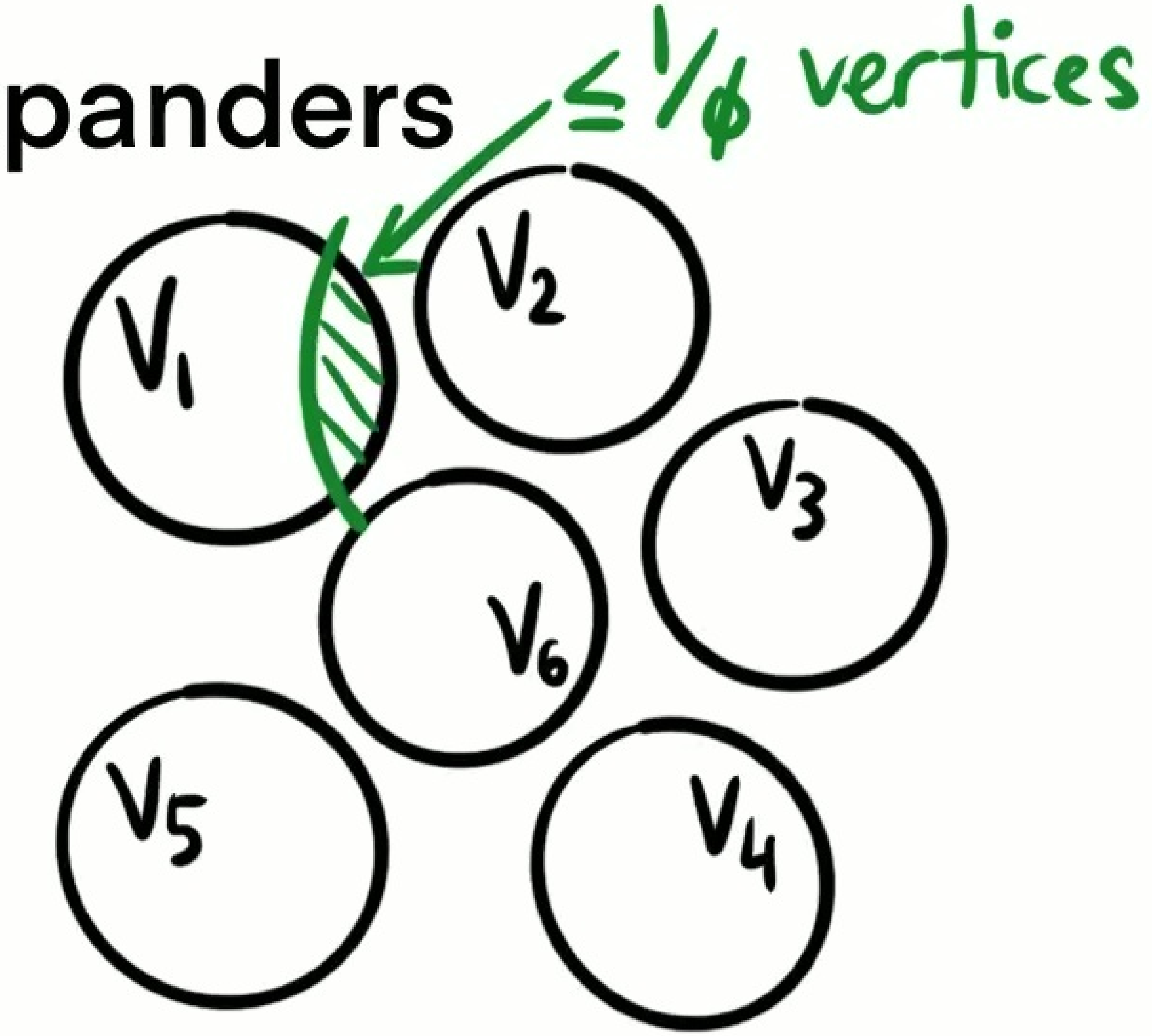
Deterministic sparsification of T (T=V)

<u>Expander decomposition</u>: partition V into $V_1, \ldots, V_k$ s.t.

(1) Each induced graph $G[V_i]$ is a ϕ-expander

(2) At most half of edges go between expanders

Insight 1: a min-cut can't cut too "deeply" into a component

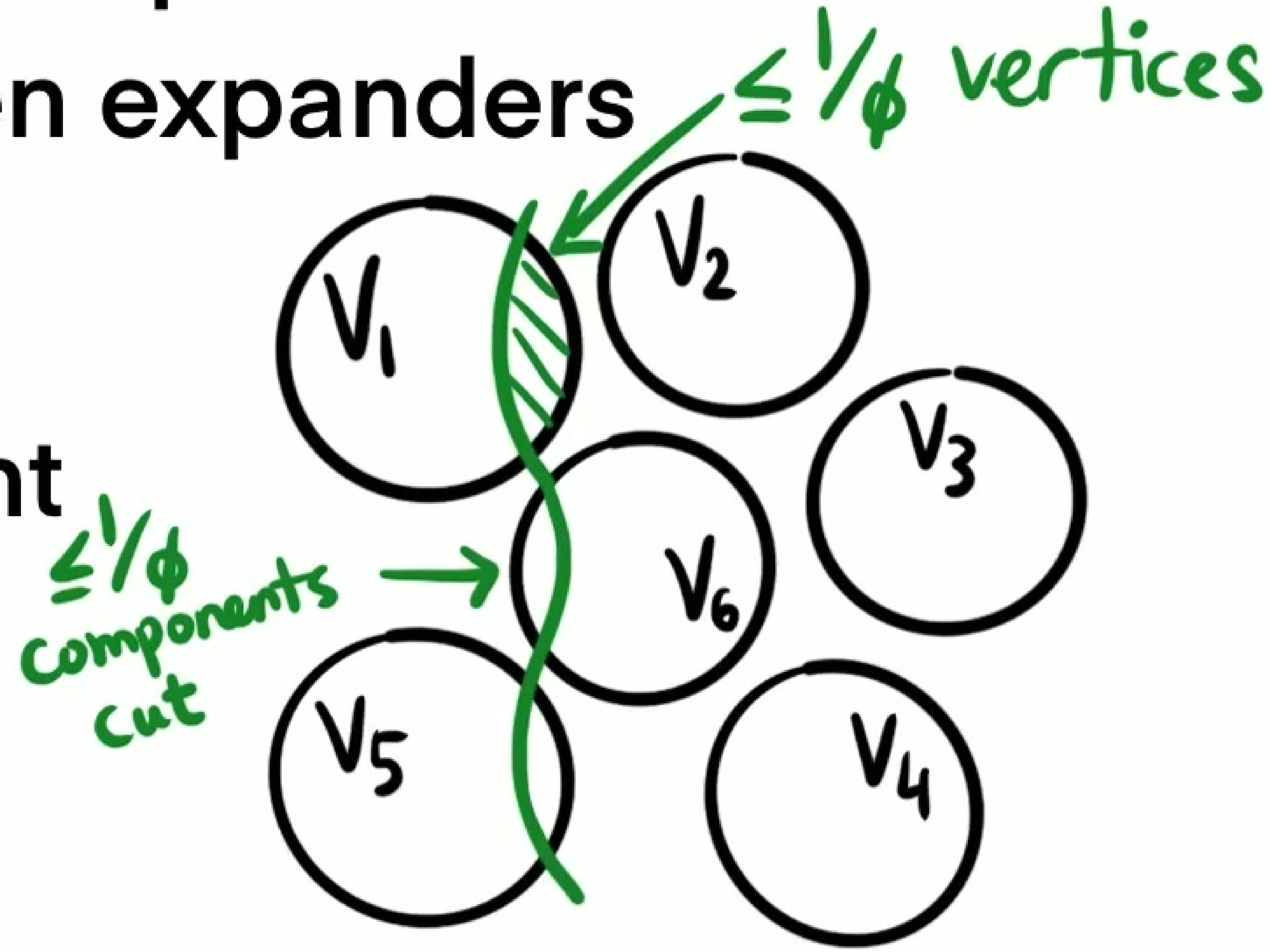


Deterministic sparsification of T (T=V)

Expander decomposition: partition V into $V_1, \ldots, V_k$ s.t.

(1) Each induced graph $G[V_i]$ is a ϕ-expander

(2) At most half of edges go between expanders



Insight 1: a min-cut can't cut too "deeply" into a component

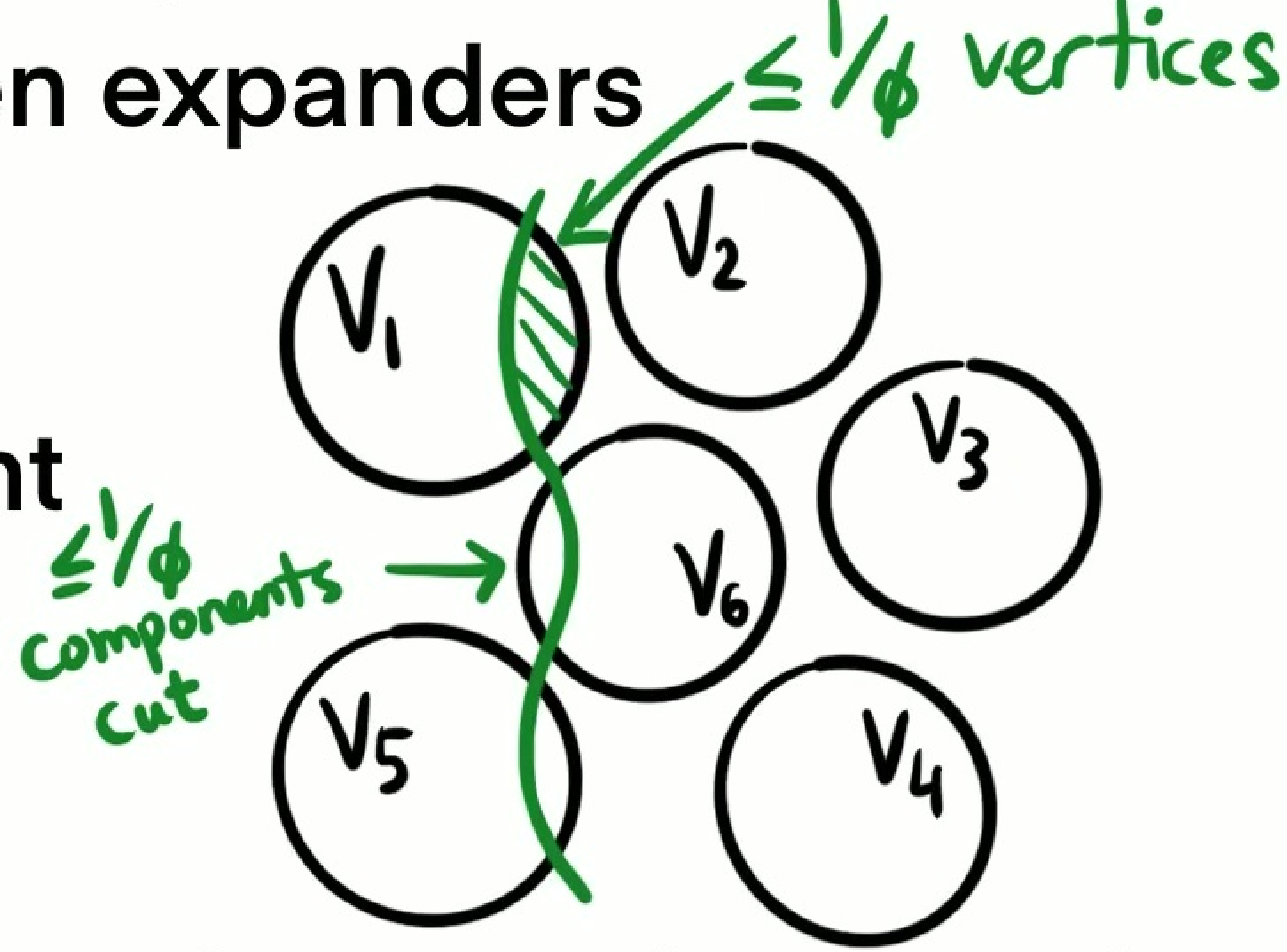
Insight 2: a min-cut can't cut into too many components

Deterministic sparsification of T (T=V)

<u>Expander decomposition</u>: partition V into $V_1, \ldots, V_k$ s.t.

(1) Each induced graph $G[V_i]$ is a ϕ-expander

(2) At most half of edges go between expanders



Insight 1: a min-cut can't cut too "deeply" into a component

Insight 2: a min-cut can't cut into too many components

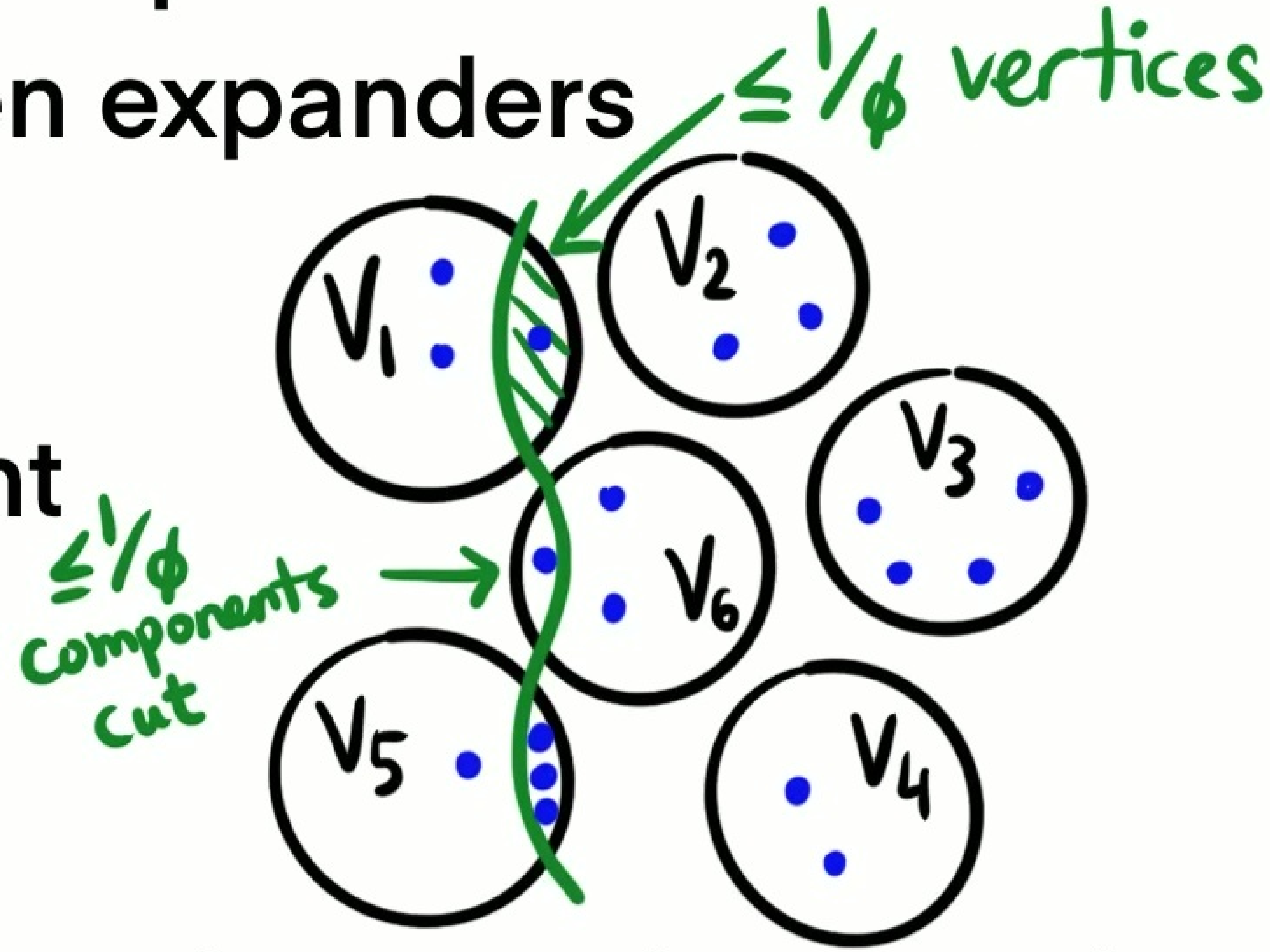
Select an arbitrary small set of vertices from each expander; that's our sparsified set T'

Deterministic sparsification of T (T=V)

<u>Expander decomposition</u>: partition V into $V_1, \dots, V_k$ s.t.

(1) Each induced graph $G[V_i]$ is a ϕ-expander

(2) At most half of edges go between expanders



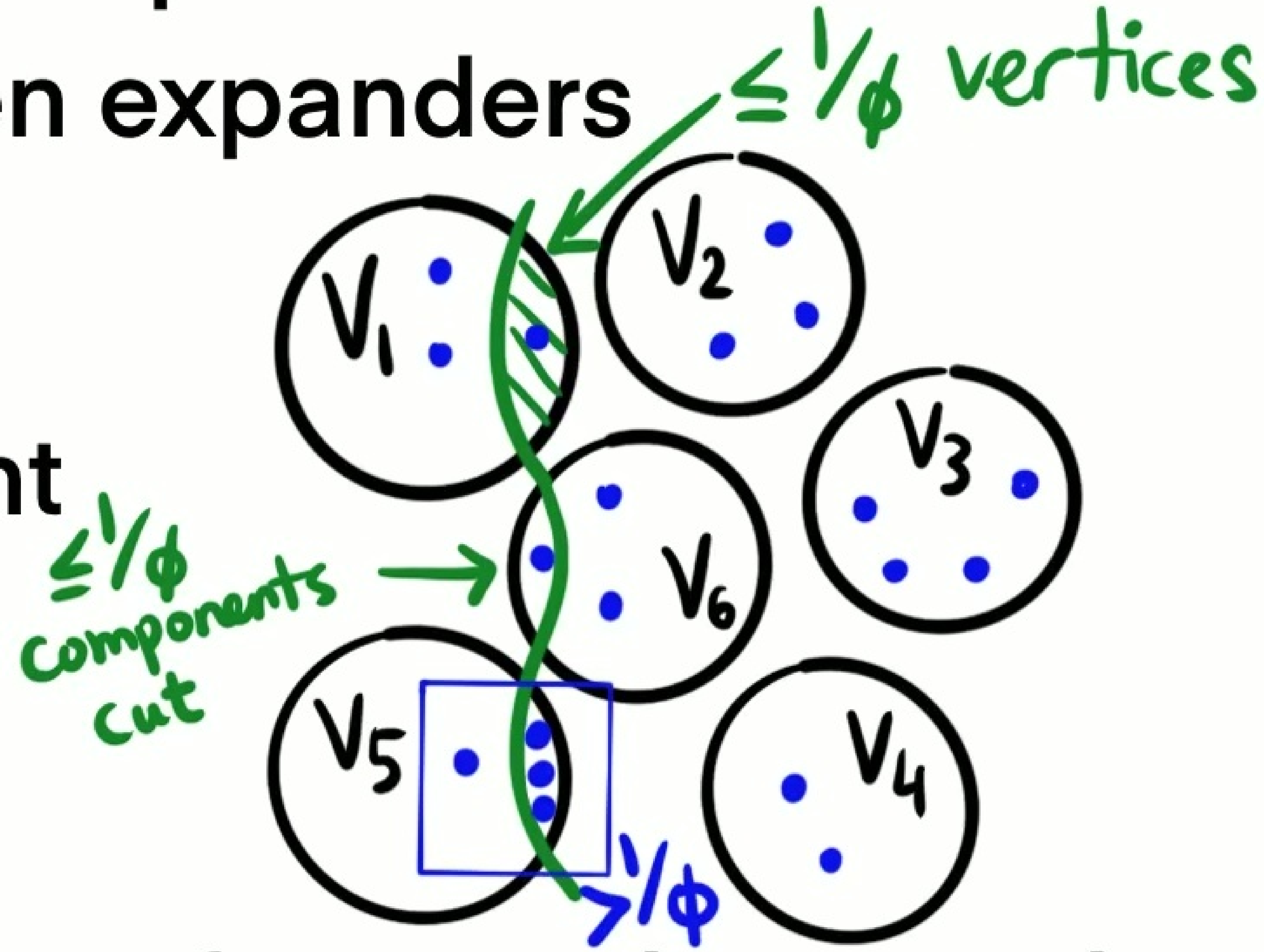
Insight 1: a min-cut can't cut too "deeply" into a component

Insight 2: a min-cut can't cut into too many components

Select an arbitrary small set of vertices from each expander; that's our sparsified set T'

Deterministic sparsification of T (T=V)

<u>Expander decomposition</u>: partition V into $V_1, \ldots, V_k$ s.t.
(1) Each induced graph $G[V_i]$ is a ϕ-expander
(2) At most half of edges go between expanders



Insight 1: a min-cut can't cut too "deeply" into a component

Insight 2: a min-cut can't cut into too many components

Select an arbitrary small set of vertices from each expander; that's our sparsified set T'

Summary & Future Work

Deterministic min-cut algorithm in $m^{1+\varepsilon}$ plus polylog(n) max-flow calls

Summary & Future Work

Deterministic min-cut algorithm in $m^{1+\varepsilon}$ plus polylog(n) max-flow calls

Main open question: $\tilde{O}(m)$ time?

Summary & Future Work

Deterministic min-cut algorithm in $m^{1+\varepsilon}$ plus polylog(n) max-flow calls

Main open question: $\tilde{O}(m)$ time?

Other applications of isolators

- Steiner min-cut in polylog(n) max-flows
- More applications?