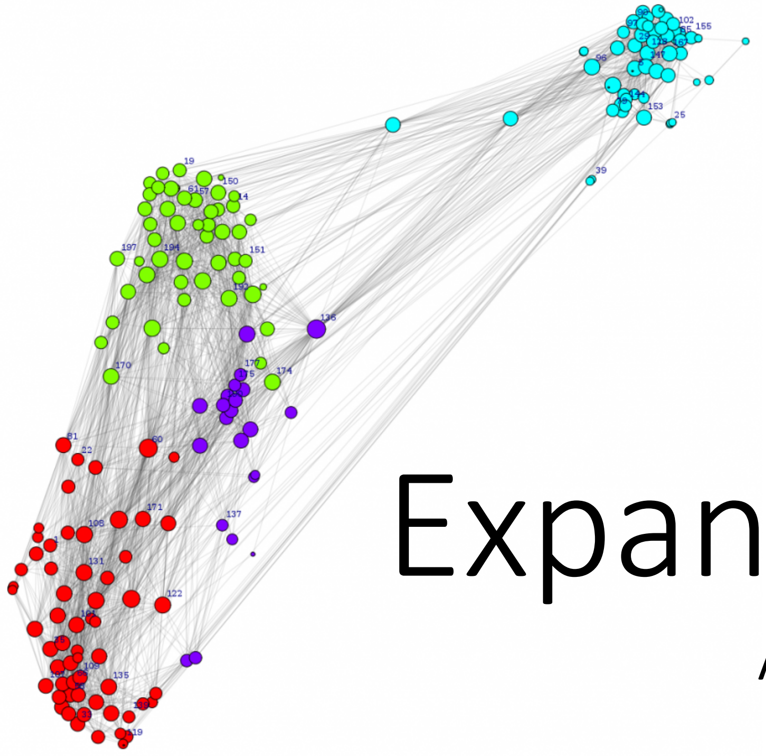
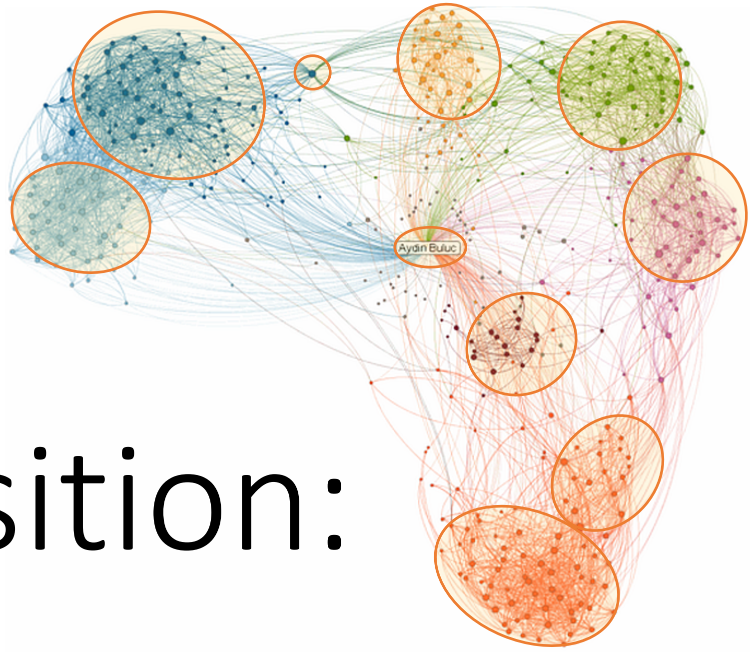




# Expander Decomposition:

## Applications and How to use it


**Thatchaphol Saranurak**

Toyota Technological Institute at Chicago


**ADGA 2019**

# Goal of this talk

1. Motivate **dynamic** algorithms
2. **Expander Decomposition** through **dynamic graph** applications.
3. How it is also used for **centralized** and **distributed** algorithms.
4. Quick **survey of applications**

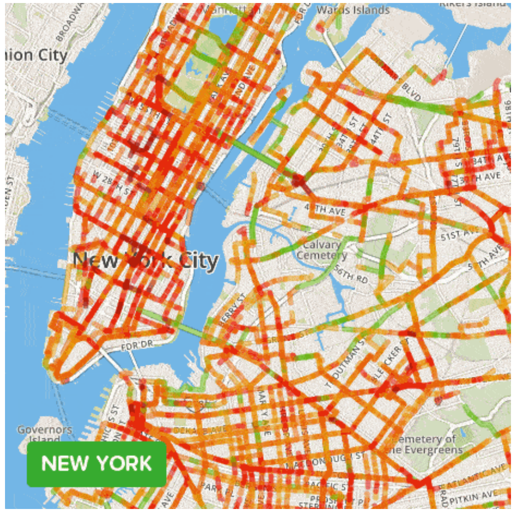
# Part 1

# Dynamic Algorithms:
# What and Why?

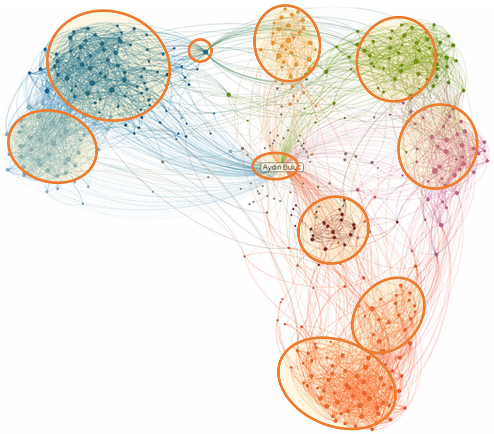
*Also say **Dynamic Data Structures** as well.

## Analyze **dynamic networks**

Road networks

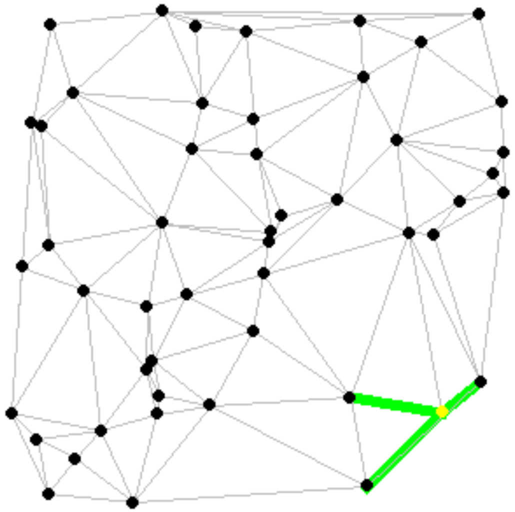



Track communities in social networks

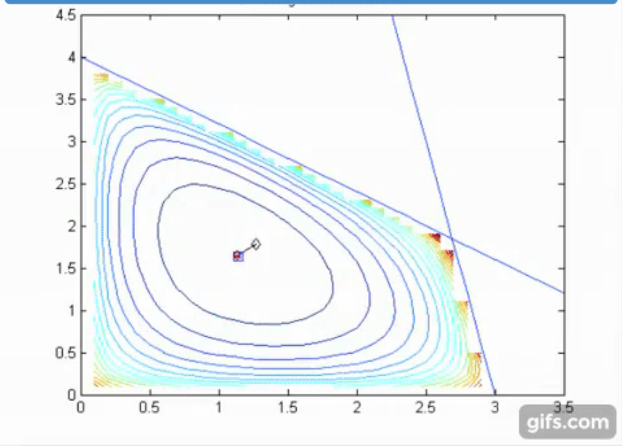


## **Subroutines** in static algorithms

“inside” Shortest Paths

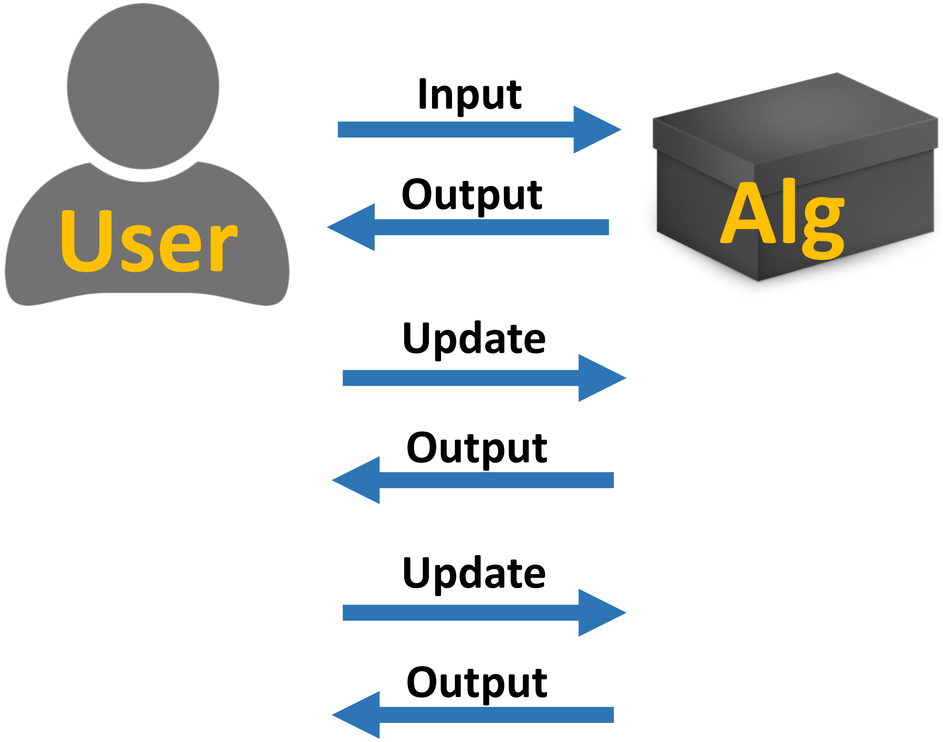


“inside” Linear Programming



# A common theme

We solve the **same** problem **repeatedly** where input keeps **changing**



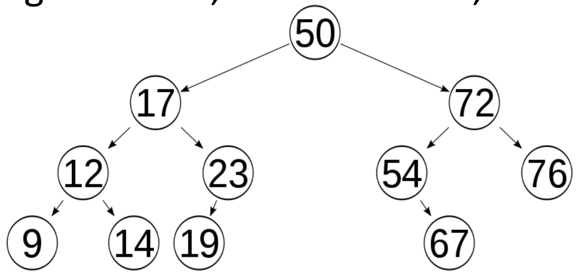
## Dynamic Algorithms

Science of how **not** to compute things from scratch

High-level goal:

"How to Efficiently Prepare for Changes"

# Example: Dynamic Problems

| Level | Textbook | Research |
|---|---|---|
| **Change** | Insert/delete a number in a set $S$ | Insert/delete an edge in a graph $G$ |
| **Maintain** | Minimum number in $S$ | Is $G$ connected? |
| **Recompute** | $\boldsymbol{O(\lvert S \rvert)}$ time | $\boldsymbol{O(\#edges)}$ time |
| **Can do** | $\boldsymbol{O(\log\lvert S \rvert)}$ time<br>**Balanced Binary Search tree**<br>e.g. AVL tree, red-black tree, etc.<br> | Example: $\mathbf{polylog(n)}$ randomized<br>[Sleator Tarjan STOC'81, Frederickson, STOC'83<br>Eppstein et al, FOCS'92, Henzinger King, STOC'95<br>Holm et al, STOC'98, Thorup, STOC'00<br>Patrascu Demaine, STOC'04, Patrascu Thorup, FOCS'07<br>Kapron et al, SODA'13, Wulff-Nilsen, SODA'13<br>Huang et al, SODA'17, Nanongkai S, STOC'17<br>Wulff-Nilsen, STOC'17, Nanongkai S Wulff-Nilsen, FOCS'17]<br>**Many** open questions… |

# More Example: Dynamic Graph Problems (in FOCS/STOC/SODA)

### Dynamic Connectivity / MST

Sleator Tarjan STOC'81
Frederickson, STOC'83
Eppstein et al, FOCS'92
Henzinger King, STOC'95
Holm et al, STOC'98
Thorup, STOC'00
Patrascu Demaine, STOC'04
Patrascu Thorup, FOCS'07
Kapron et al, SODA'13
Wulff-Nilsen, SODA'13
Huang et al, SODA'17
Nanongkai **S**, STOC'17
Wulff-Nilsen, STOC'17
Nanongkai **S** Wulff-Nilsen, FOCS'17

### Dynamic (Directed) Reachability

Even Shiloach, JACM'81
Henzinger King, FOCS'95
King Sagert, STOC'99
Demetrescu Italiano, FOCS'00
Rodditty Zwick, FOCS'02
Sankowski, FOCS'04
Lacki, SODA'11
Henzinger et al, STOC'14
Chechik et al, FOCS'16
Italiano et al, STOC'17
Bernstein et al, STOC'19

### Dynamic Maximum Matching

Sankowski, SODA'07
Onak Rubinfeld, STOC'10
Baswana et al, FOCS'11
Gupta Peng, FOCS'13
Neiman Soloman, STOC'13
Bosek et al, FOCS'14
Gupta et al, SODA'14
Bhattacharya et al, SODA'15
Bernstein Stein, SODA'16
Peleg Solomon, SODA'16
Bhattacharya et al, STOC'16
Solomon, FOCS'16
Bhattacharya et al, SODA'17

### Dynamic Single-Source Shortest Path

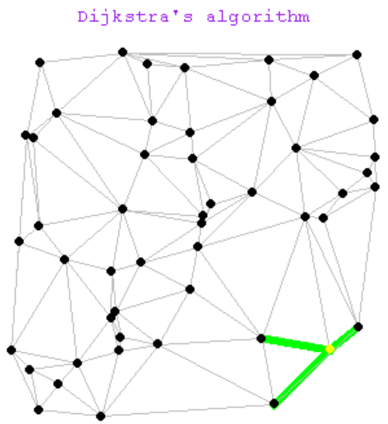
Even Shiloach, JACM'81
Ausiello et al, SODA'90
Rodditty Zwick, FOCS'04
Bernstein Riditty, SODA'11
Bernstein, STOC'13
Henzinger et al, SODA'14
Henzinger et al, STOC'14
Henzinger et al, FOCS'14
Bernstein Chechik, STOC'16
Bernstein Chechik, SODA'17
Chuzhoy Khanna, STOC'19
Probst Wulff-Nilsen, SODA'20
Probst Wulff-Nilsen, SODA'20

### Dynamic All-Pairs Shortest Path

King, FOCS'99
Demetrescu Italiano, FOCS'00
Fakcharoenphol Rao, FOCS'01
Baswana et al, STOC'02
Baswana et al, SODA'03
Roditty Zwick, FOCS'04
Thorup, STOC'05
Bernstein, SODA'09
Abraham et al, STOC'12
Henzinger et al, FOCS'13
Abraham et al, SODA'17
Chechik, FOCS'18
Probst Wulff-Nilsen, SODA'20

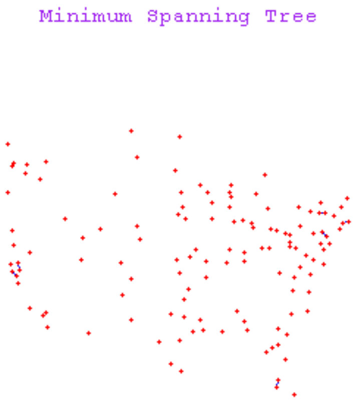
**More problems…**

# Dynamic Alg. Inside Static Alg.



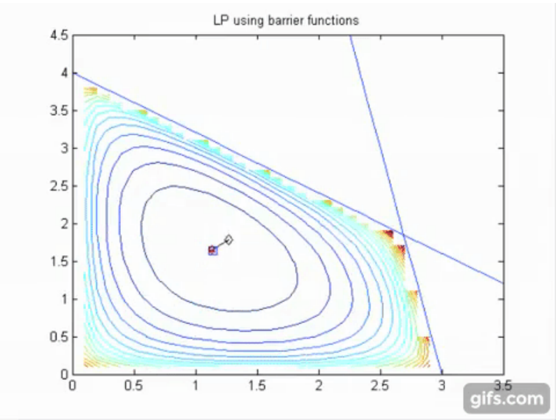

**Shortest path**

Priority Queue
(e.g. Fibonacci Heap)



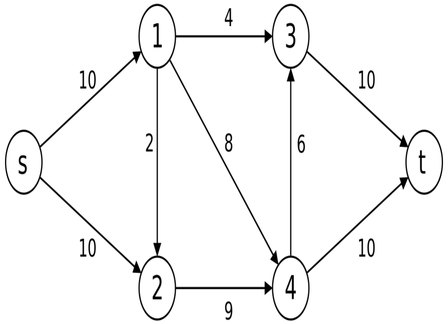

**Kruskal's MST**

Union-find



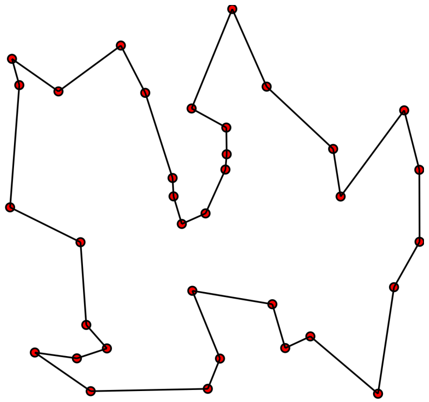

**Linear Program**
[Karmakar'84] [Vaidya'89]

Dynamic Linear
System Solver




**Max flow**
[Sleator Tarjan'82]

Link-cut Tree



**Traveling Salesman**
[Chekuri Quanrud FOCS'17]

Dyn. "Global
Min Cut"

**Many more…**

**Non-adaptive users:**

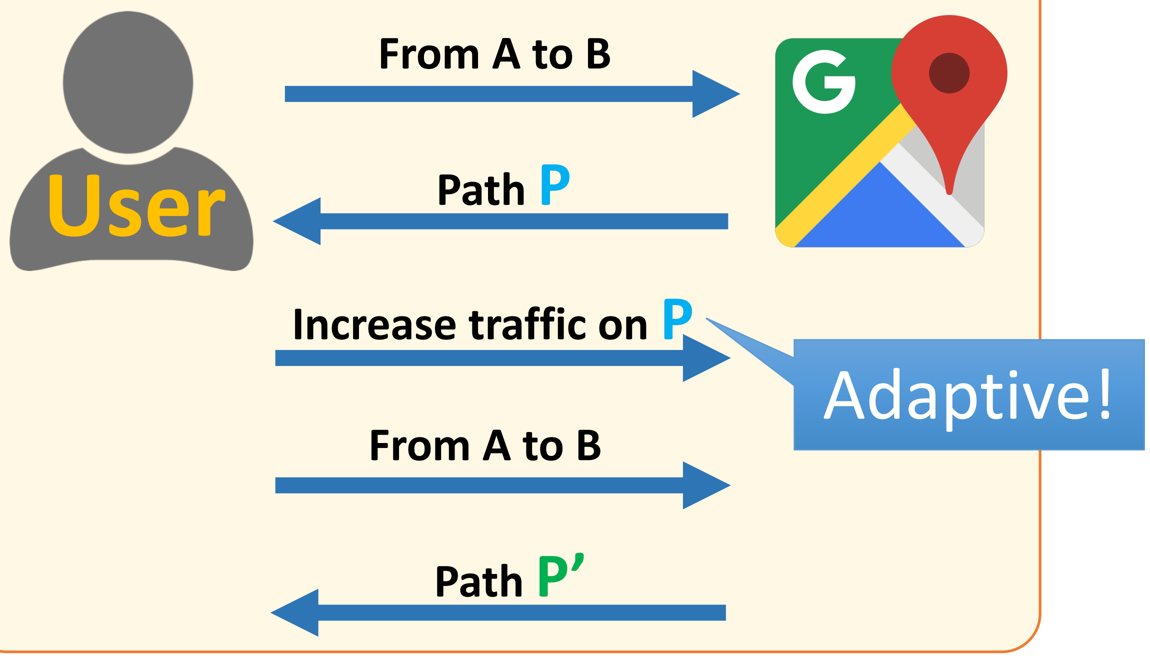
All updates are **fixed from the beginning**.

Usually **cannot** be used as subroutines inside static algo.

**Adaptive users:**

Updates from users can **depend on previous answers**

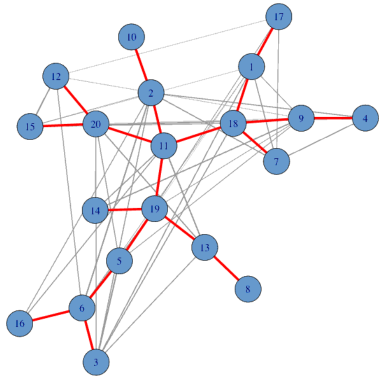
**Example:**

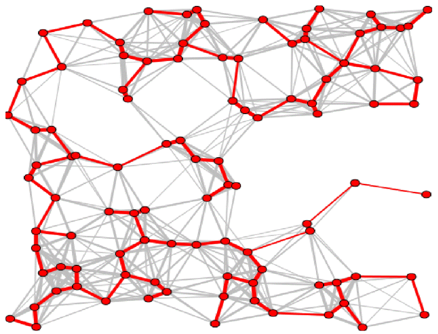


# Dynamic Spanning Forest: Definition and Progress

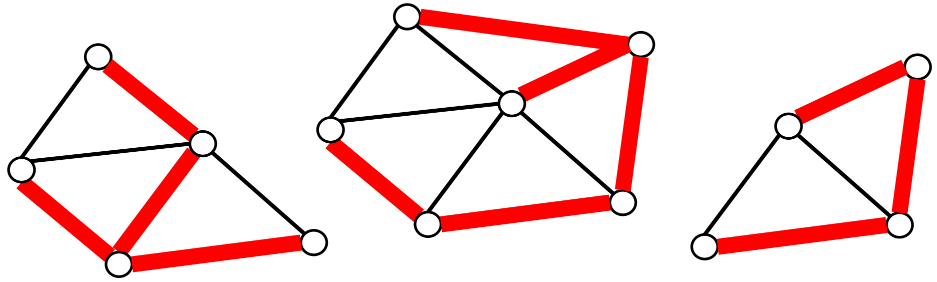
# <u>Definition</u>: Spanning Tree/Forest

**Spanning tree**: a **smallest** sub-network that **connects all** nodes together





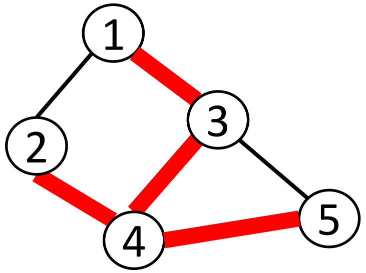
**Spanning forest**: set of **spanning trees** on each connected component



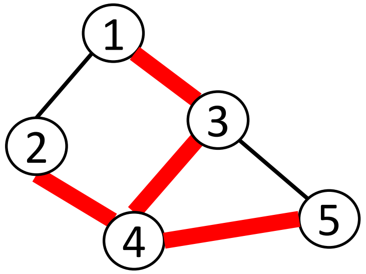
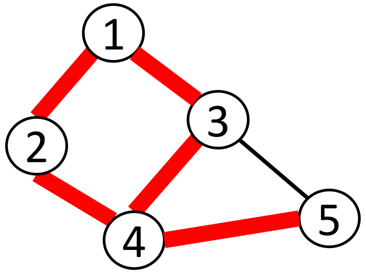
# *Maintaining a **spanning forest** under changes*

*Will say spanning tree and spanning forest interchangeably

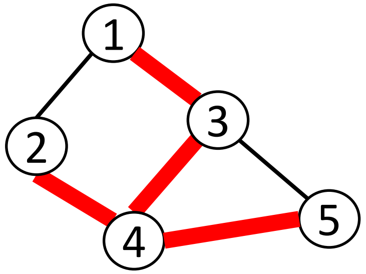
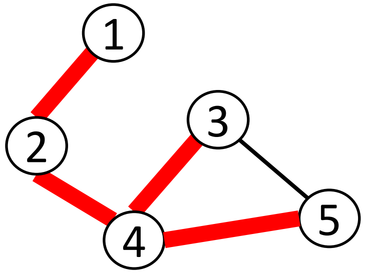
# Example: Dynamic Spanning Forest

| | |
|---|---|
| **Input:**<br>Update in G | |
| **Picture** |  |
| **Output:**<br>Change in F | |

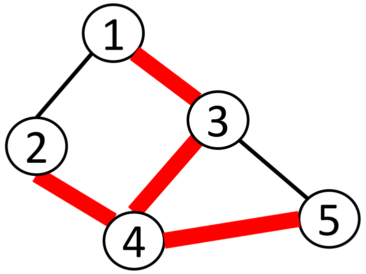
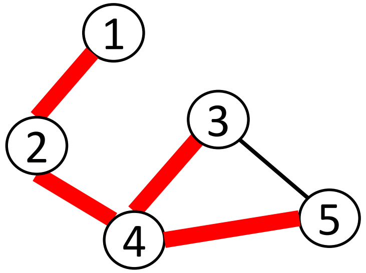
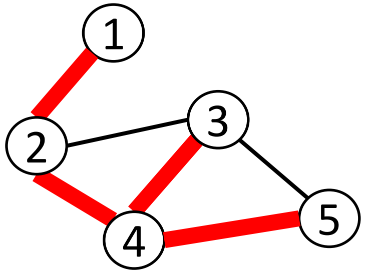
# Example: Dynamic Spanning Forest

| **Input:** Update in G | | Delete(1,3) |
|---|---|---|
| **Picture** |  |  |
| **Output:** Change in F | | |

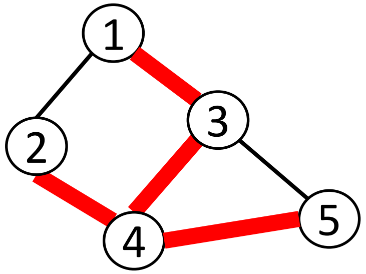
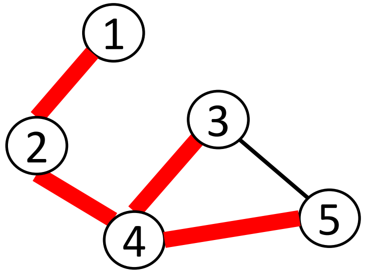
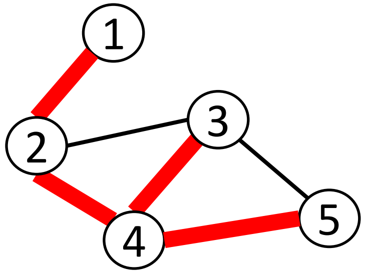
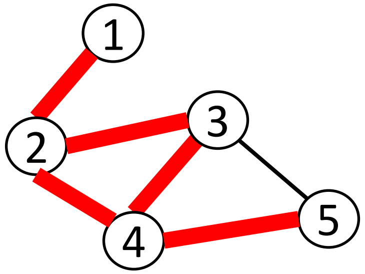
# Example: Dynamic Spanning Forest

| | | |
|---|---|---|
| **Input:**<br>Update in G | | Delete(1,3) |
| **Picture** |  |  |
| **Output:**<br>Change in F | | (1,3) removed<br>(1,2) added |

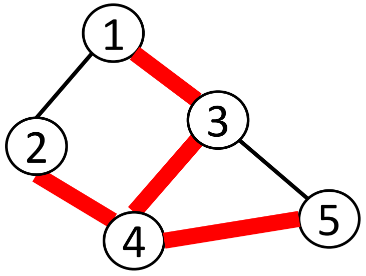
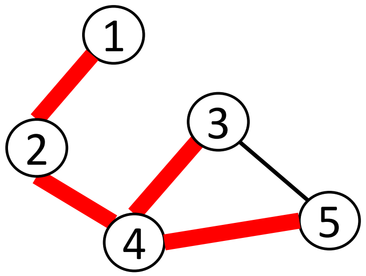
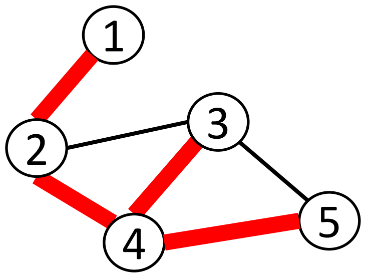
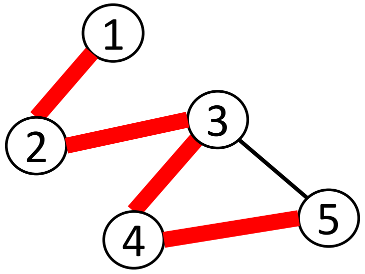
# Example: Dynamic Spanning Forest

| **Input:**<br>Update in G | | Delete(1,3) | Insert(2,3) |
|---|---|---|---|
| **Picture** |  |  |  |
| **Output:**<br>Change in F | | (1,3) removed<br>(1,2) added | |

# Example: Dynamic Spanning Forest

| **Input:** Update in G | | Delete(1,3) | Insert(2,3) | Delete(2,4) |
|---|---|---|---|---|
| **Picture** |  |  |  |  |
| **Output:** Change in F | | (1,3) removed<br>(1,2) added | | |

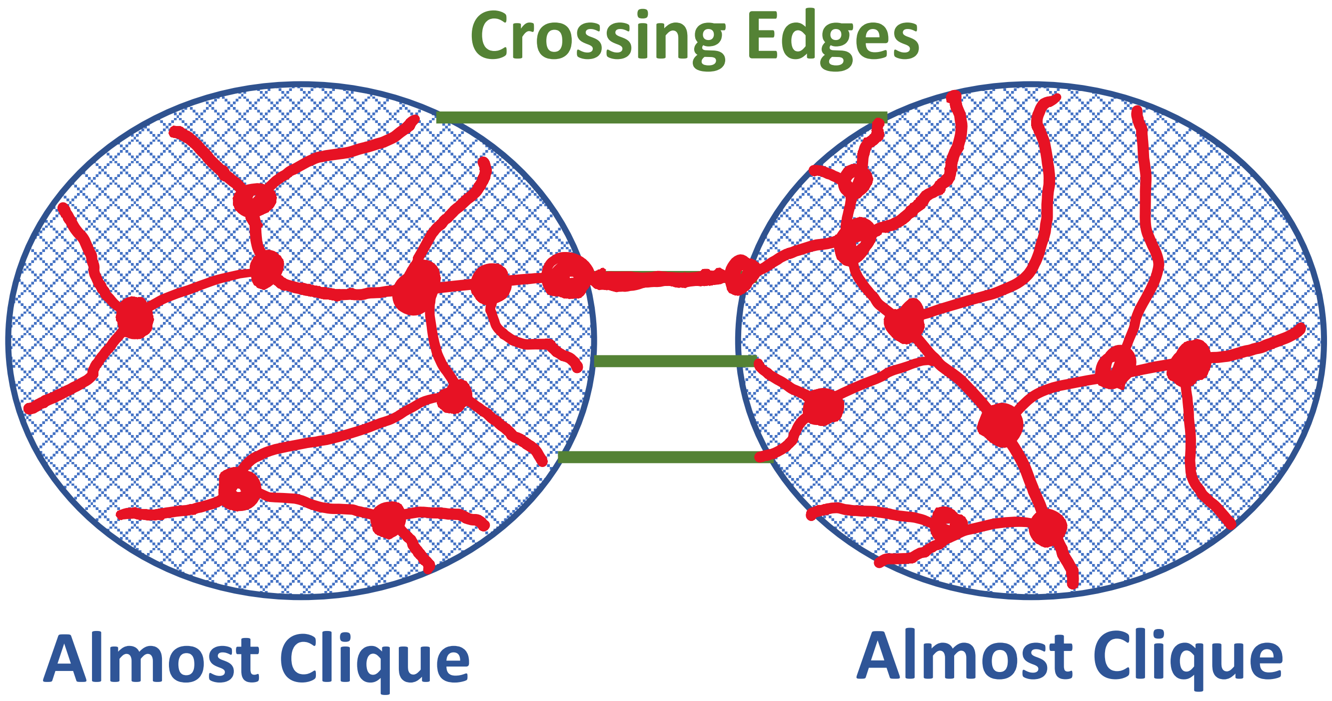
# Example: Dynamic Spanning Forest

| **Input:** Update in G | | Delete(1,3) | Insert(2,3) | Delete(2,4) |
|---|---|---|---|---|
| **Picture** |  |  |  |  |
| **Output:** Change in F | | (1,3) removed<br>(1,2) added | | (2,4) removed<br>(2,3) added |

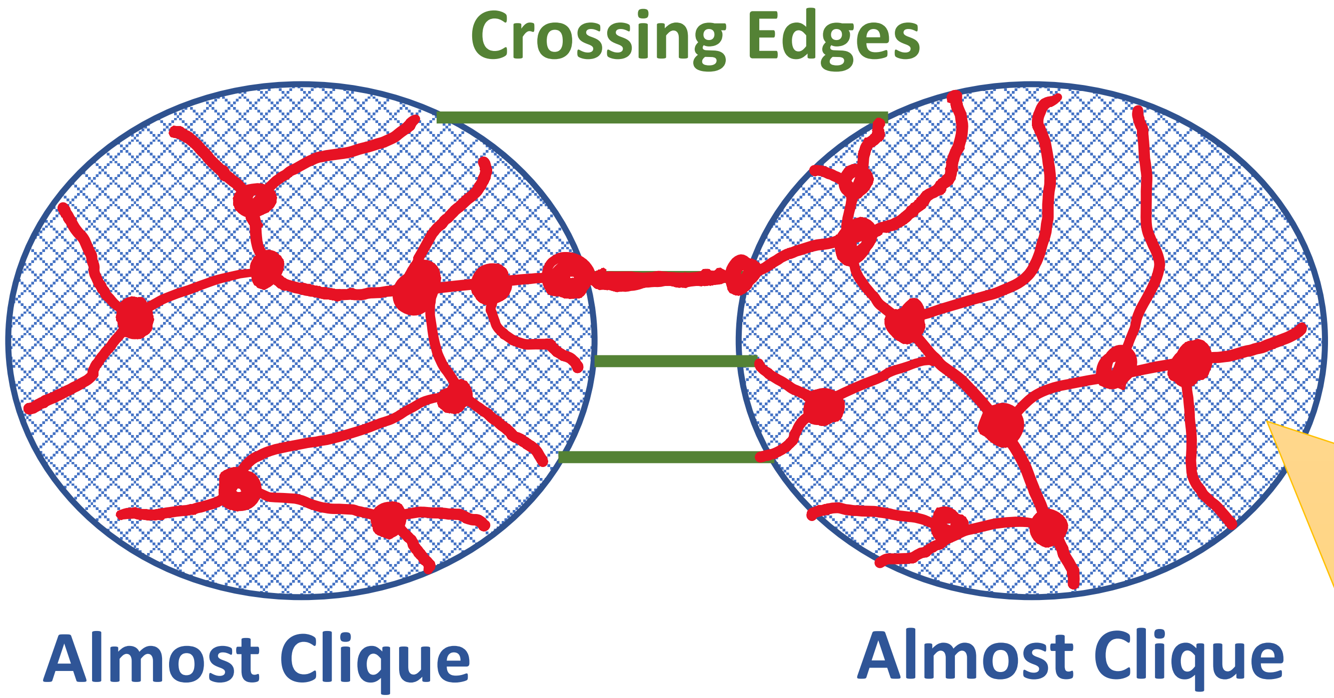
Goal: minimize update time

**Worst-case** time to output
**changes of F** for each update

# Why this problem can be hard?



# Why this problem can be hard?




**Interesting when**:
delete a **tree-edge**

**Want:** Find a **crossing edge**

**Question**:
Must scan all **clique-edges**?
**Scan the whole graph?**

# Progress

$n$ = # of nodes, $m$=# of edges

| Reference | Update time |
|---|---|
| **Naïve** | $m$ |
| **Frederickson** [STOC'83] | $m^{1/2}$ |
| **EGIN** [FOCS'92] | $n^{1/2}$ |

**Hide log factors** from now

# Progress

$n$ = # of nodes, $m$=# of edges

| Reference | Update time |
|---|---|
| **Naïve** | $m$ |
| **Frederickson** [STOC'83] | $m^{1/2}$ |
| **EGIN** [FOCS'92] | $n^{1/2}$ |
| 20-year gap: A lot of successes in closely related settings (amortized update time) | |

**Important development in amortized update time**

**Henzinger King** [STOC'95]
**Holm Lichtenberg Thorup** [STOC'98]
**Thorup** [STOC'00]
**Patrascu Demaine** [STOC'04]
**Wulff-Nilsen** [SODA'13]
**HHKP** [SODA'17]

# Progress

$n$ = # of nodes, $m$=# of edges

**Important development in amortized update time**

**Henzinger King** [STOC'95]
**Holm Lichtenberg Thorup** [STOC'98]
**Thorup** [STOC'00]
**Patrascu Demaine** [STOC'04]
**Wulff-Nilsen** [SODA'13]
**HHKP** [SODA'17]

| **Reference** | **Update time** |
|---|---|
| **Naïve** | $m$ |
| **Frederickson** [STOC'83] | $m^{1/2}$ |
| **EGIN** [FOCS'92] | $n^{1/2}$ |
| 20-year gap: A lot of successes in closely related settings (amortized update time) | |
| **Assume user is not adaptive** | |
| **Kapron King Mountjoy** [SODA'13] | $\mathrm{polylog}\, n$ |

**Non-adaptive users:**

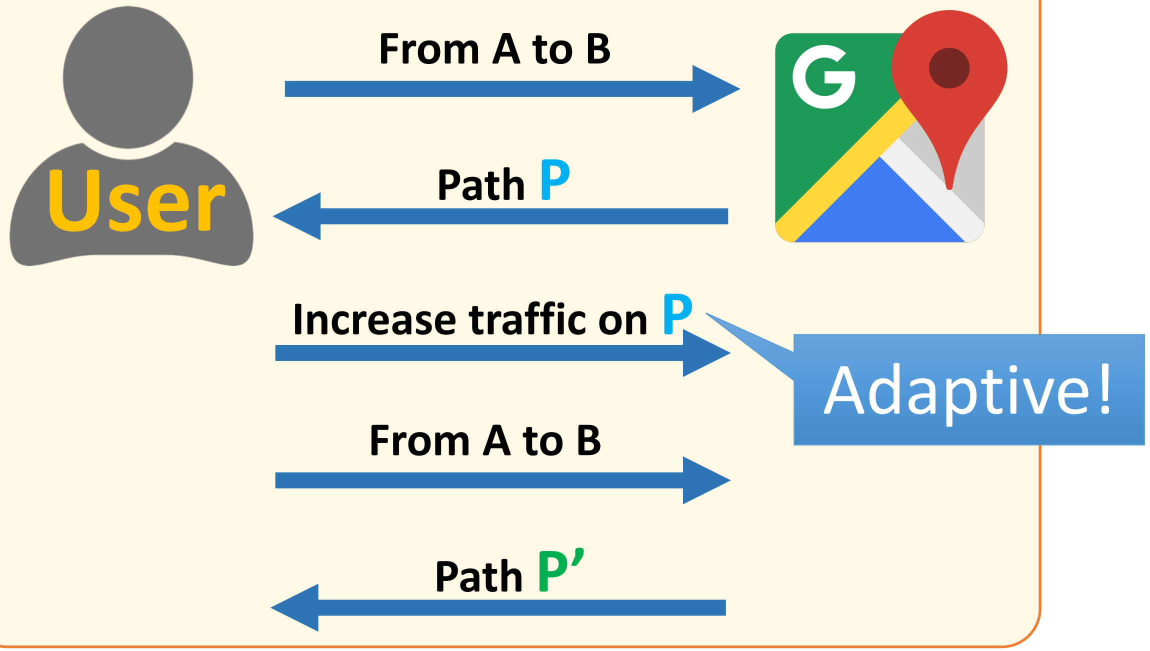
All updates are **fixed from the beginning**.

Usually **cannot** be used as subroutines inside static algo.

**Adaptive users:**

Updates from users can **depend on previous answers**

**Example:**



# Progress

$n$ = # of nodes, $m$=# of edges

**Important development in amortized update time**

**Henzinger King** [STOC'95]
**Holm Lichtenberg Thorup** [STOC'98]
**Thorup** [STOC'00]
**Patrascu Demaine** [STOC'04]
**Wulff-Nilsen** [SODA'13]
**HHKP** [SODA'17]

| Reference | Update time |
|---|---|
| **Naïve** | $m$ |
| **Frederickson** [STOC'83] | $m^{1/2}$ |
| **EGIN** [FOCS'92] | $n^{1/2}$ |
| 20-year gap: A lot of successes in closely related settings (amortized update time) | |
| **Assume user is not adaptive** | |
| **Kapron King Mountjoy** [SODA'13] | $\mathrm{polylog}\, n$ |

# Progress

$n$ = # of nodes, $m$=# of edges

| Reference | Update time |
|---|---|
| **Naïve** | $m$ |
| **Frederickson** [STOC'83] | $m^{1/2}$ |
| **EGIN** [FOCS'92] | $n^{1/2}$ |
| 20-year gap:<br>A lot of successes in closely related settings<br>(amortized update time) | |
| **Assume user is not adaptive** | |
| **Kapron King Mountjoy** [SODA'13] | $\operatorname{polylog} n$ |
| **KKPT** [ESA'16] | $n^{1/2} \cdot \frac{\log \log n}{(\log n)^{1/2}}$ |

# Progress

$n$ = # of nodes, $m$=# of edges

| Reference | Update time |
|---|---|
| **Naïve** | $m$ |
| **Frederickson** [STOC'83] | $m^{1/2}$ |
| **EGIN** [FOCS'92] | $n^{1/2}$ |
| 20-year gap:<br>A lot of successes in closely related settings<br>(amortized update time) | |
| **Assume user is not adaptive** | |
| **Kapron King Mountjoy** [SODA'13] | $\mathrm{polylog}\, n$ |
| **KKPT** [ESA'16] | $n^{1/2} \cdot \frac{\log\log n}{(\log n)^{1/2}}$ |
| **Wulff-Nilsen** [STOC'17] | $n^{0.499}$ |
| **Nanongkai S** [STOC'17] | $n^{0.401}$ |

**Independent works**

# Progress

$n$ = # of nodes, $m$=# of edges

| Reference | Update time |
|---|---|
| **Naïve** | $m$ |
| **Frederickson** [STOC'83] | $m^{1/2}$ |
| **EGIN** [FOCS'92] | $n^{1/2}$ |
| 20-year gap:<br>A lot of successes in closely related settings<br>(amortized update time) | |
| **Assume user is not adaptive** | |
| **Kapron King Mountjoy** [SODA'13] | $\mathrm{polylog}\, n$ |
| **KKPT** [ESA'16] | $n^{1/2} \cdot \frac{\log\log n}{(\log n)^{1/2}}$ |
| **Wulff-Nilsen** [STOC'17] | $n^{0.499}$ |
| **Nanongkai S** [STOC'17] | $n^{0.401}$ |
| **NSW** [FOCS'17] | $\boldsymbol{n^{o(1)}}$ |

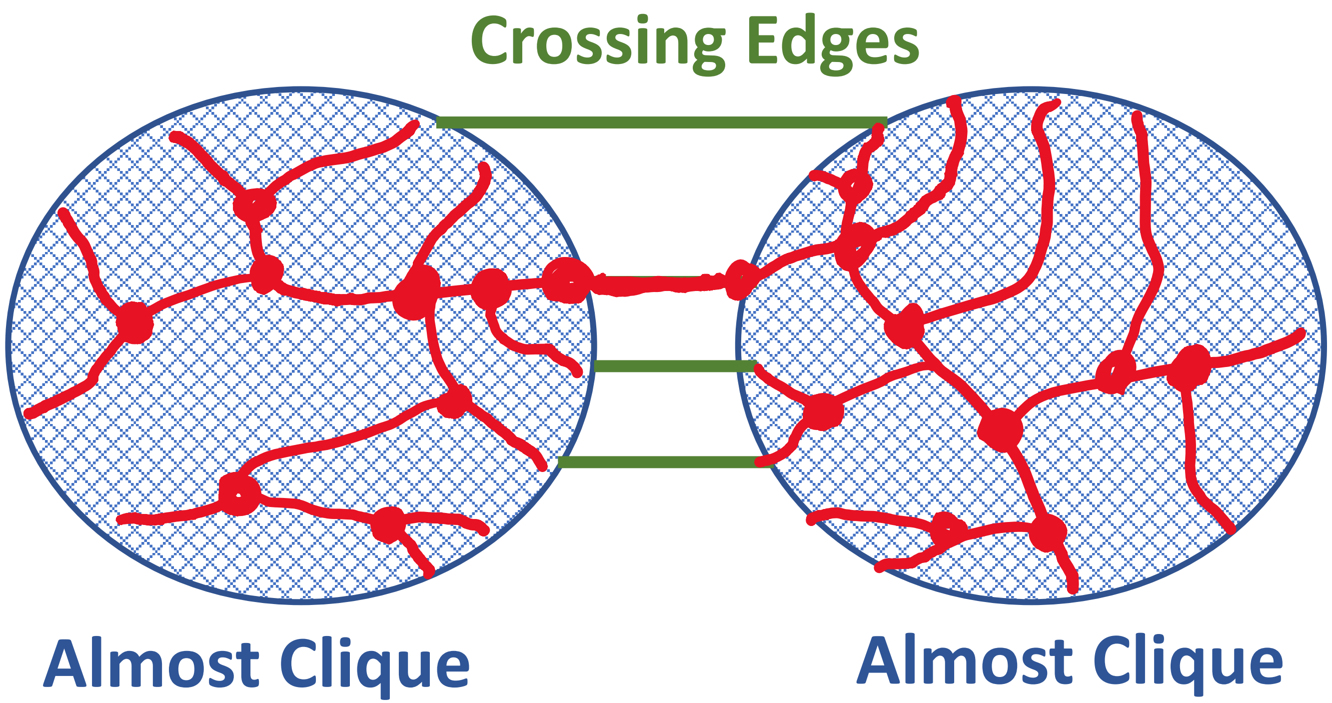
Will explain how to use
Expander decomposition
via (simplification of) this work

# Part 1.2

# Dynamic Spanning Forest:

## How to use Expander Decomposition

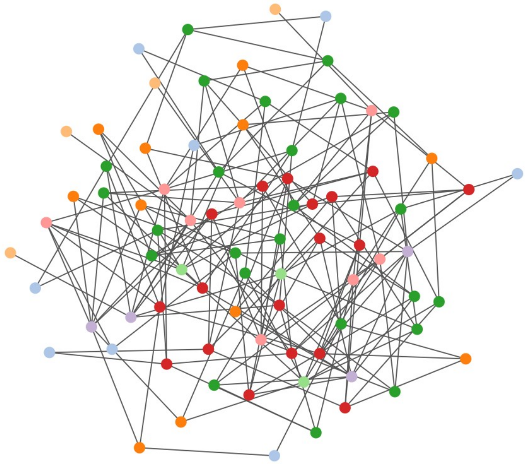
# **Recall**: Why this problem can be hard?



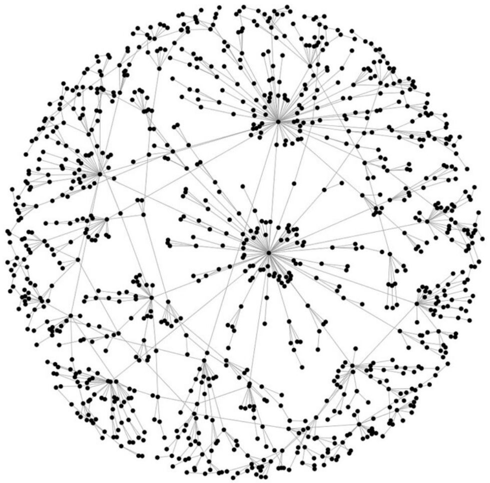

Let’s solve the problem on *graphs* that this situation **cannot happen…**

# Expanders

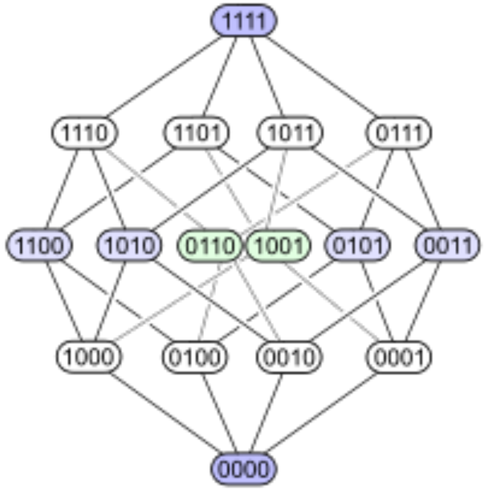
**Intuition**
- Well-connected
- Hard to separate into two equal sides



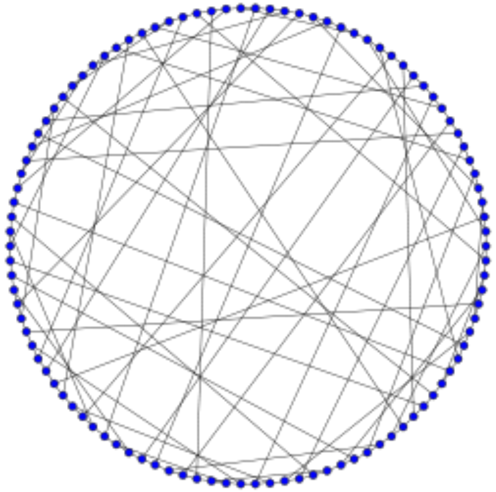
Random Graphs
(Erdös-Rényi)



Power-law Graphs
(preferential attachment)
[Gkantsidis, Mihail, Saberi SIGMETRICS'03]
[Mihail, Papadimitriou, Saberi FOCS'03]




Hypercubes



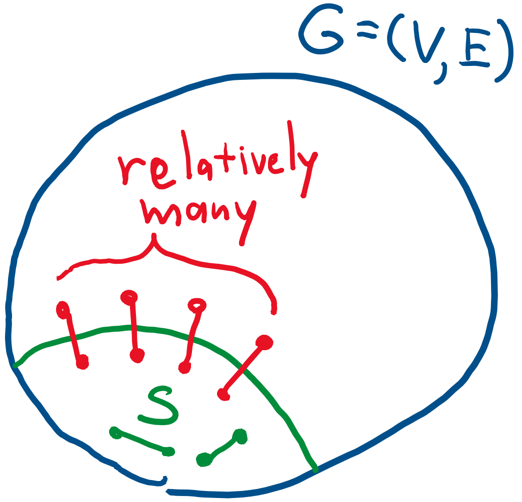
$\mathbb{F}_p$-cycles
with inverse chords

# **Definition:** Expanders

$G = (V, E)$ is an **expander** if

$$\forall S \subset V \quad \frac{|E(S, \bar{S})|}{\min\{vol(S), vol(\bar{S})\}} \geq \frac{1}{\mathrm{polylog}(n)}$$

Sum of degree:
$vol(S) = \sum_{u \in S} \deg_G u$




In general,
$\phi$-expander: $\frac{|E(S,\bar{S})|}{\min\{vol(S),vol(\bar{S})\}} \geq \phi$

# Expander Paradigm

1. Solve it on **expanders**.

2. **Combine** the solutions.

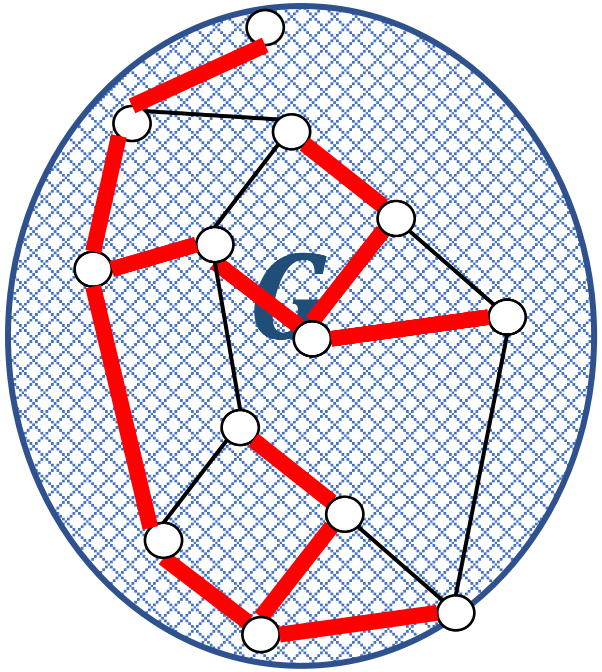
# Expander Paradigm

1. Solve it on **expanders**.

# **Warm-up**: One update to Expander

Suppose that $\boldsymbol{G}$ is an expander, and there is **one** update.

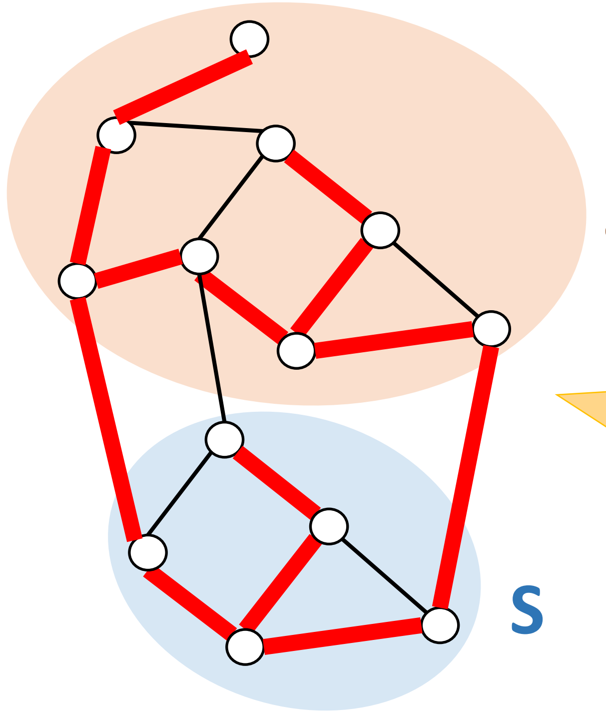
**Goal**: maintain a **spanning tree** $\boldsymbol{T}$ of $\boldsymbol{G}$.



# **Warm-up**: One update to Expander

Suppose that $\boldsymbol{G}$ is an expander, and there is **one** update.

**Goal**: maintain a **spanning tree** $\boldsymbol{T}$ of $\boldsymbol{G}$




**Interesting only when**: delete a tree-edge

**Want**: edge crossing $S$ to reconnect

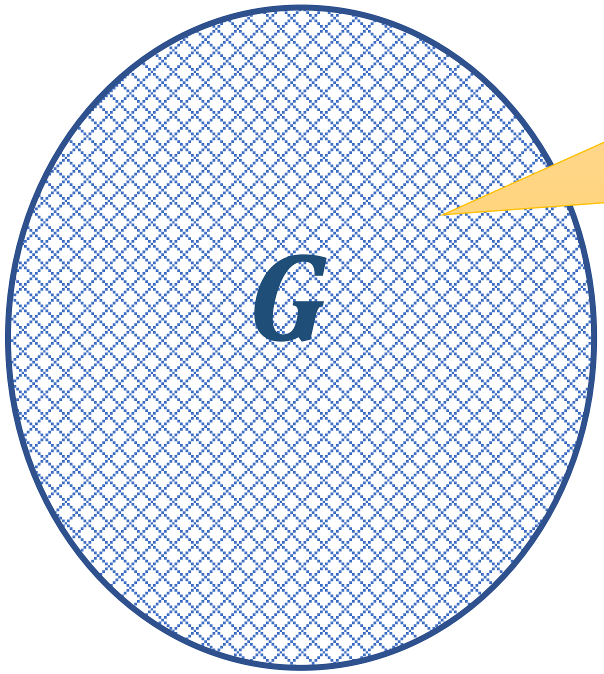
**Alg: sample an edge with an endpoint in $\boldsymbol{S}$**

(can do fast)

**By expansion**: get edge crossing $S$ w.p. $\frac{1}{\text{polylog}(n)}$

**Repeat**: $\tilde{O}(1)$ times. Done w.h.p.

# What if there are more updates?




after many edge deletions,
**not expander** anymore!

Let’s “repair” the expander

# Expander Paradigm

1. Solve it on **expanders**.

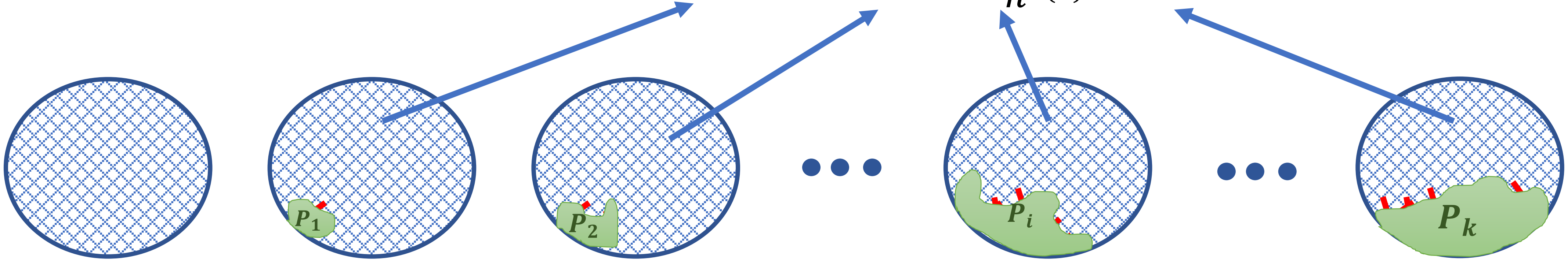
Problem specific:
**e.g. Random Sampling**

General tool:
**Expander Pruning**

2. **Combine** the solutions.

# <u>Expander Pruning</u> [NSW'17]

$G_i[V - \boldsymbol{P_i}]$ is a $(\frac{1}{n^{o(1)}})$-**expander***




$\boldsymbol{G_0}$: expander

$\boldsymbol{G_1 = G_0 - e_1}$

$\boldsymbol{G_2 = G_1 - e_2}$

$\boldsymbol{G_i = G_{i-1} - e_i}$

$\boldsymbol{G_k = G_{k-1} - e_k}$

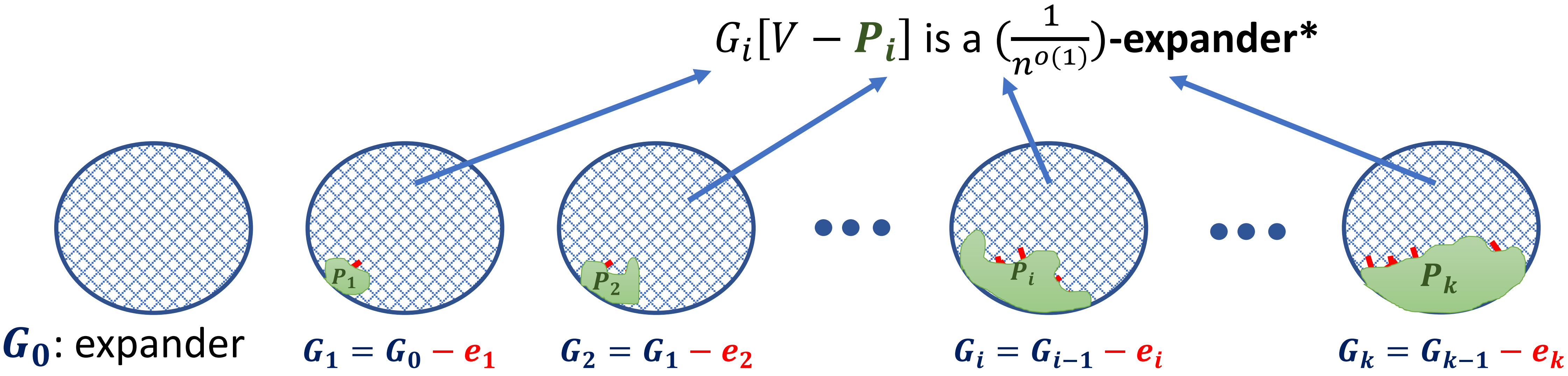
where $\boldsymbol{k \leq m/n^{o(1)}}$

**Guarantee:**

1. Time to update $\boldsymbol{P_{i-1}}$ to $\boldsymbol{P_i}$ is $\boldsymbol{n^{o(1)}}$
2. So $vol(\boldsymbol{P_i}) = \boldsymbol{i} \cdot \boldsymbol{n^{o(1)}}$
3. $G_i[V - \boldsymbol{P_i}]$ is a $\frac{1}{n^{o(1)}}$-**expander**

*We show something slightly weaker

# Expander Pruning [NSW'17]




**Expanders** can be **quickly "repaired"** under edge updates.

# Expander Paradigm

1. Solve it on **expanders.**

Problem specific:

**e.g. Random Sampling**

General tool:

**Expander Pruning**

# Expander Paradigm

1. Solve it on **expanders.**

Problem specific:

**e.g. Random Sampling**

General tool:

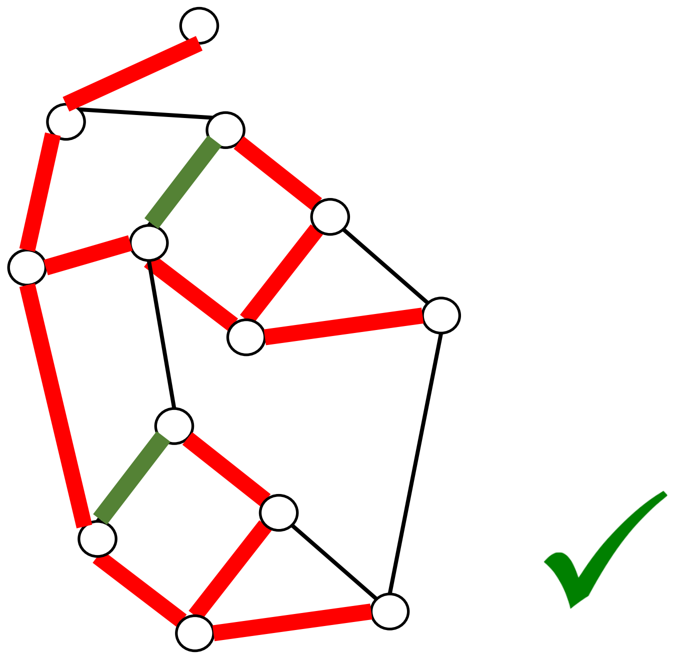
**Expander Pruning**

In this talk, will only show
how to solve a **relaxed problem**
(contains all conceptual ideas)

# Relaxed Problem: Dynamic Spanning Subgraphs

1. Maintain **Any Spanning Subgraph** with $\widetilde{O}(n)$ edges
   (Easier than **Spanning Forest**.)

2. There are only $n^{1-o(1)}$ updates
   (can assume w.l.o.g. by standard techniques.)

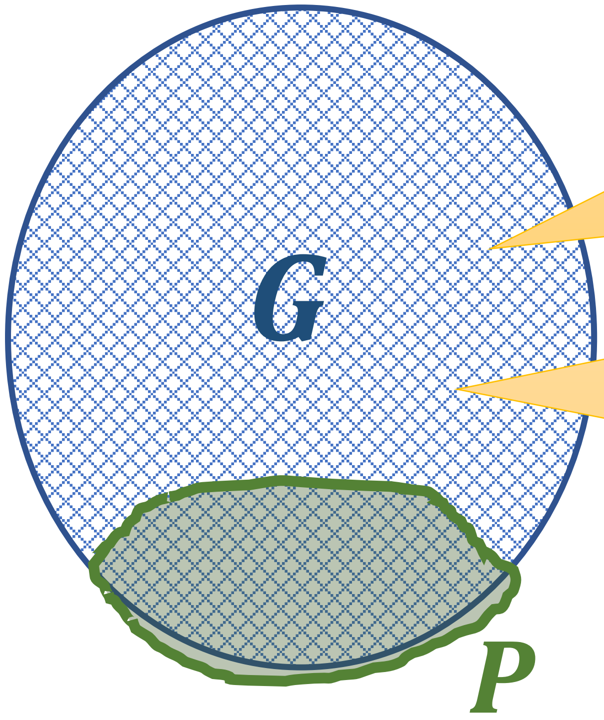


# What if there are more updates?

**Expander Pruning:**

1. Time to update $\boldsymbol{P_i}$ is $\boldsymbol{n^{o(1)}}$
2. So $vol(\boldsymbol{P_i}) = \boldsymbol{i} \cdot \boldsymbol{n^{o(1)}}$
3. $G_i[V - \boldsymbol{P_i}]$ is a $\frac{1}{n^{o(1)}}$**-expander**

Suppose that $\boldsymbol{G}$ is an expander, but there are **many** updates.




after many edge deletions, **not expander** anymore

**Expander Pruning**

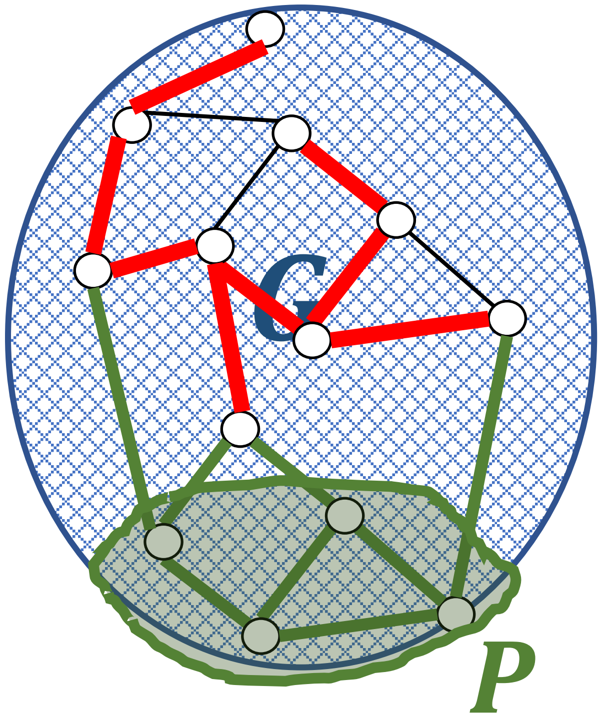
maintains $\boldsymbol{P}$ where

$\boldsymbol{G}[\boldsymbol{V} - \boldsymbol{P}]$ is $\frac{1}{n^{o(1)}}$-expander

# What if there are more updates?

**Expander Pruning:**
1. Time to update $\boldsymbol{P_i}$ is $\boldsymbol{n^{o(1)}}$
2. So $vol(\boldsymbol{P_i}) = \boldsymbol{i} \cdot \boldsymbol{n^{o(1)}}$
3. $G_i[V - \boldsymbol{P_i}]$ is a $\frac{1}{n^{o(1)}}$**-expander**

Suppose that $\boldsymbol{G}$ is an expander, but there are **many** updates.
**Algo**: maintain **spanning tree $\boldsymbol{T}$ of $\boldsymbol{G[V-P]}$** union with $\boldsymbol{E(P,V)}$



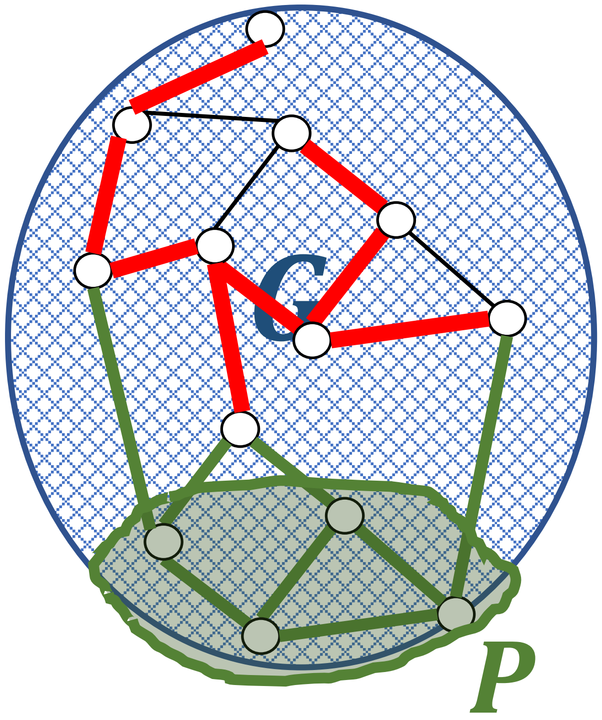

**Update time:** $n^{o(1)}$
- Updating $E(P,V)$: $n^{o(1)}$ by **Expander Pruning**.
- Updating $T$: $n^{o(1)}$ by **Random Sampling**
  - $G[V-P]$ is $\frac{1}{n^{o(1)}}$-expander at any time.

Work with adaptive users!

# What if there are more updates?

**Expander Pruning:**
1. Time to update $\boldsymbol{P_i}$ is $\boldsymbol{n^{o(1)}}$
2. So $vol(\boldsymbol{P_i}) = \boldsymbol{i} \cdot \boldsymbol{n^{o(1)}}$
3. $G_i[V - \boldsymbol{P_i}]$ is a $\frac{1}{n^{o(1)}}$**-expander**

Suppose that $\boldsymbol{G}$ is an expander, but there are **many** updates.
**Algo**: maintain **spanning tree $\boldsymbol{T}$ of $\boldsymbol{G[V-P]}$** union with $\boldsymbol{E(P,V)}$




**Correctness:**
- $T \cup E(P,V)$ spans $G$
- $|T \cup E(P,V)| = O(n)$
  - $|T| \leq n$
  - $|E(P,V)| = vol(P) = O(n)$
    - **Recall**: #updates is $n^{1-o(1)}$

# Expander Paradigm

1. Solve it on **expanders.**

Done on expanders

Problem specific:

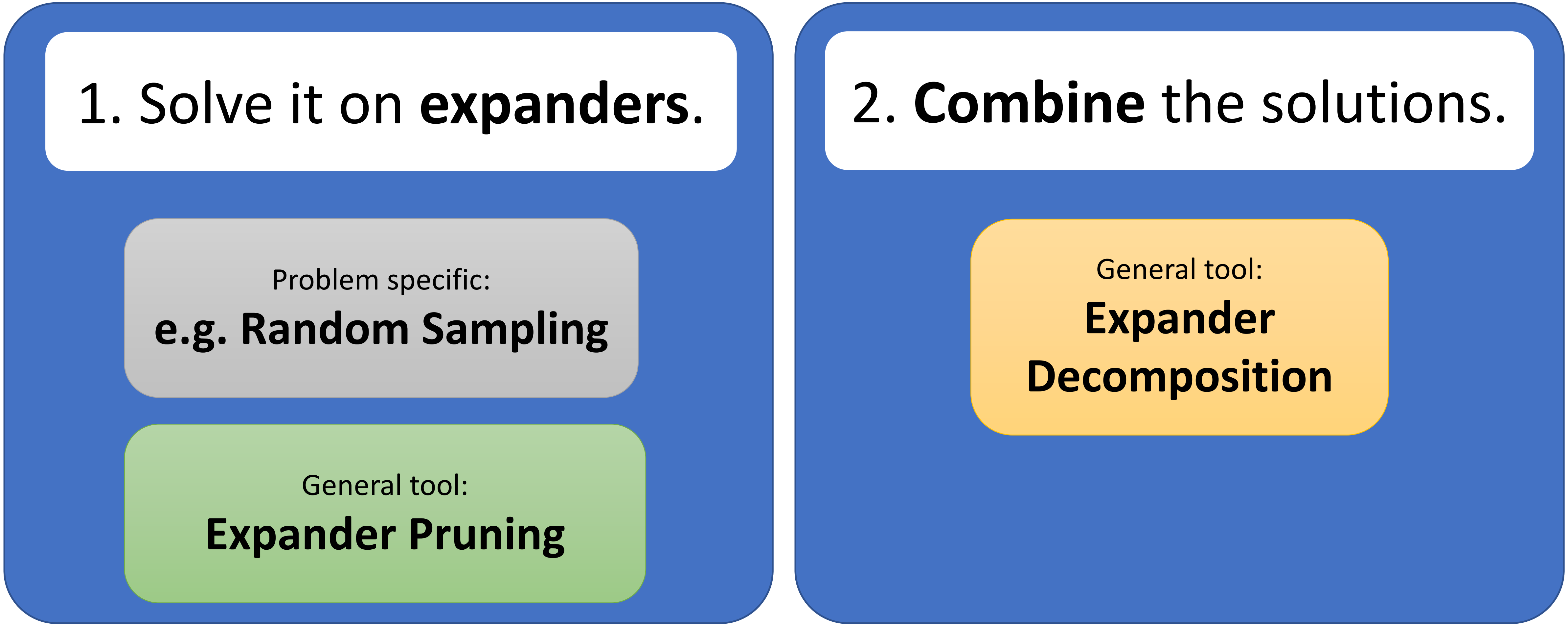
**e.g. Random Sampling**

General tool:

**Expander Pruning**

How to work with general graphs?

# Expander Paradigm

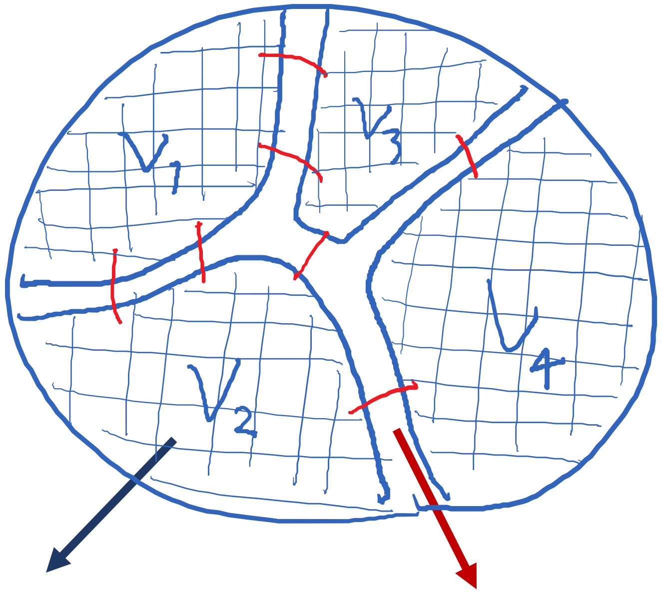


# Expander Decomposition

[**S** Wang SODA'19]:
$\widetilde{O}(m)$-time w.h.p.

**Input:** $G = (V, E)$

**Output:** A partition $(V_1, \ldots V_k)$ of $V$

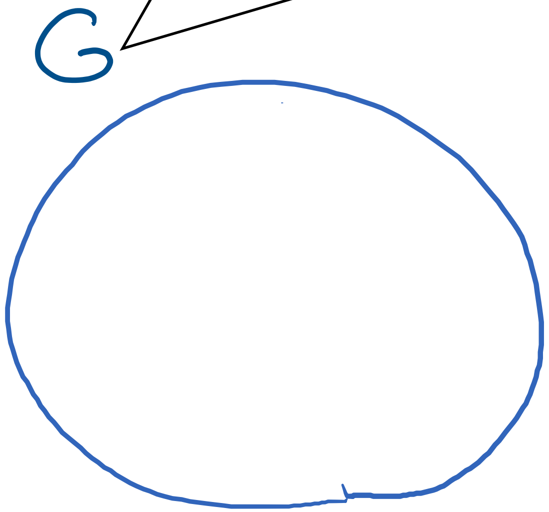



"**Graph** = Disjoint **Expanders** + **Few Edges**"

$G[V_i]$ **is expander** $\le m/2$ **inter-cluster edges**

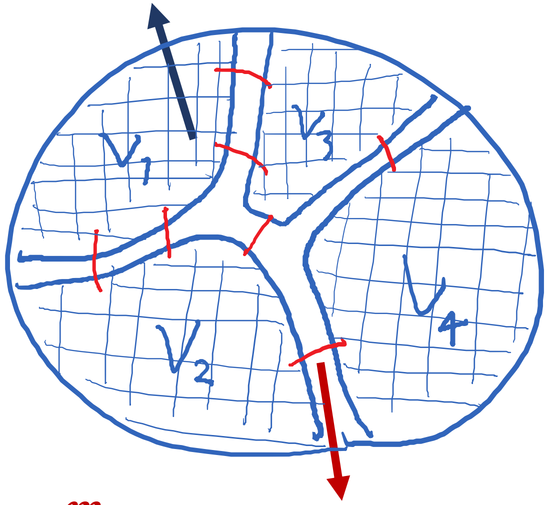
# Repeated Expander Decomposition

Expander decomposition



# Repeated Expander Decomposition

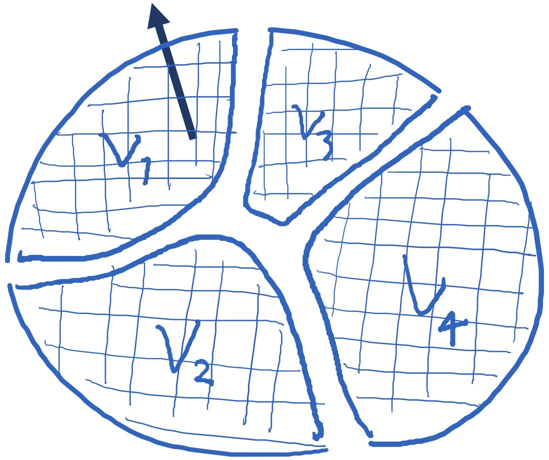
**expander**



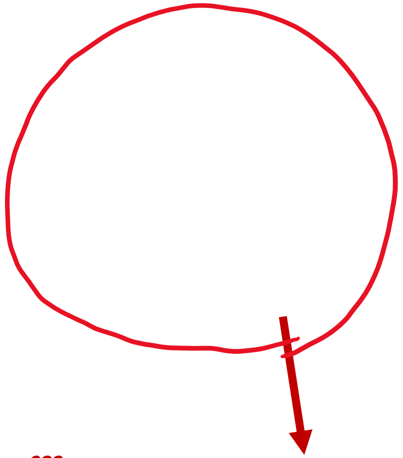

**$\leq \frac{m}{2}$ inter-cluster edges**

# Repeated Expander Decomposition

**expander**

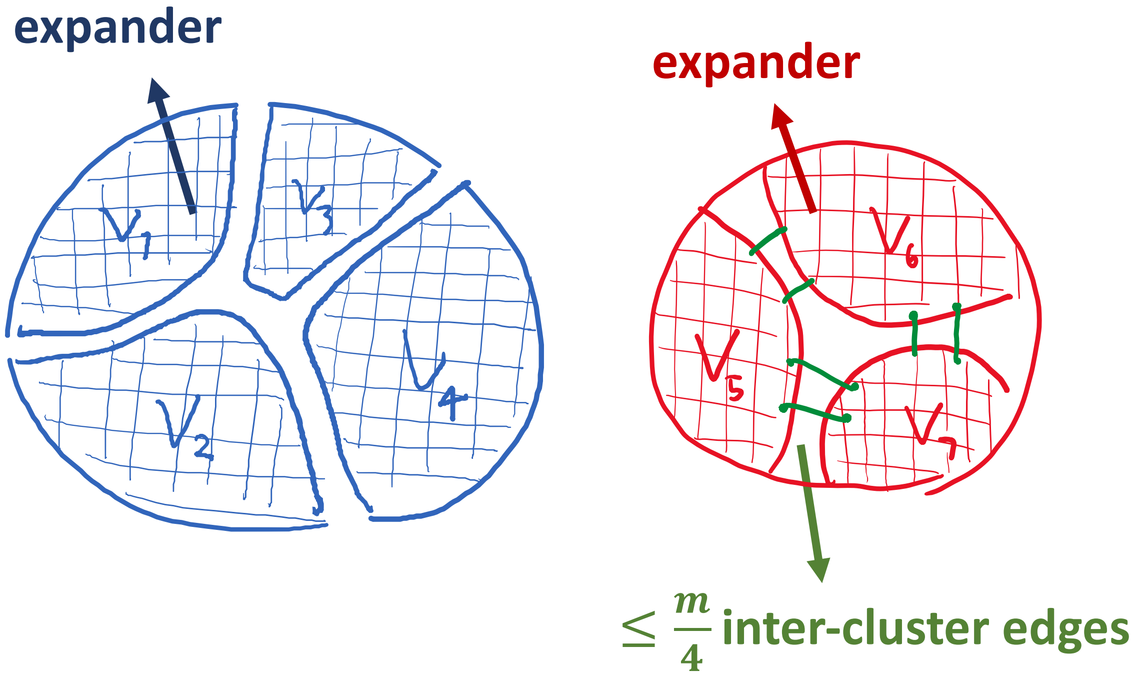



Expander decomposition

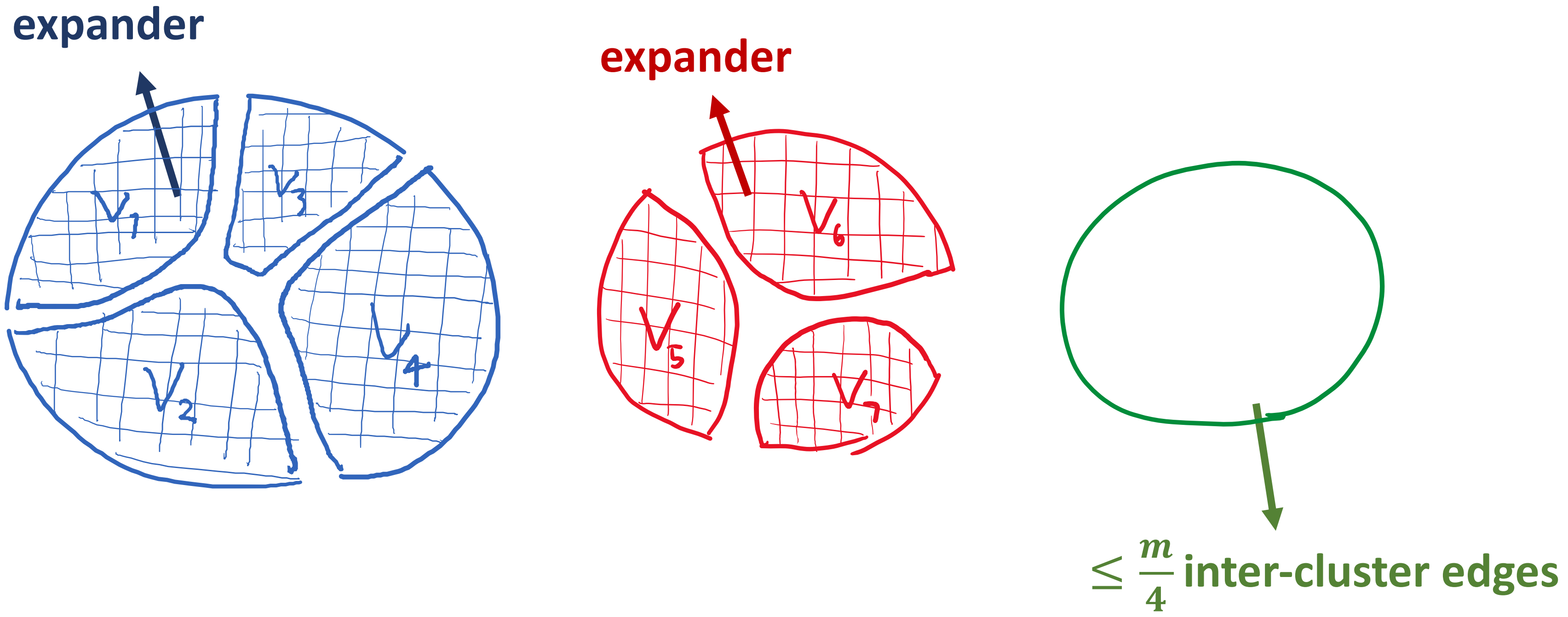


$\leq \frac{m}{2}$ **inter-cluster edges**

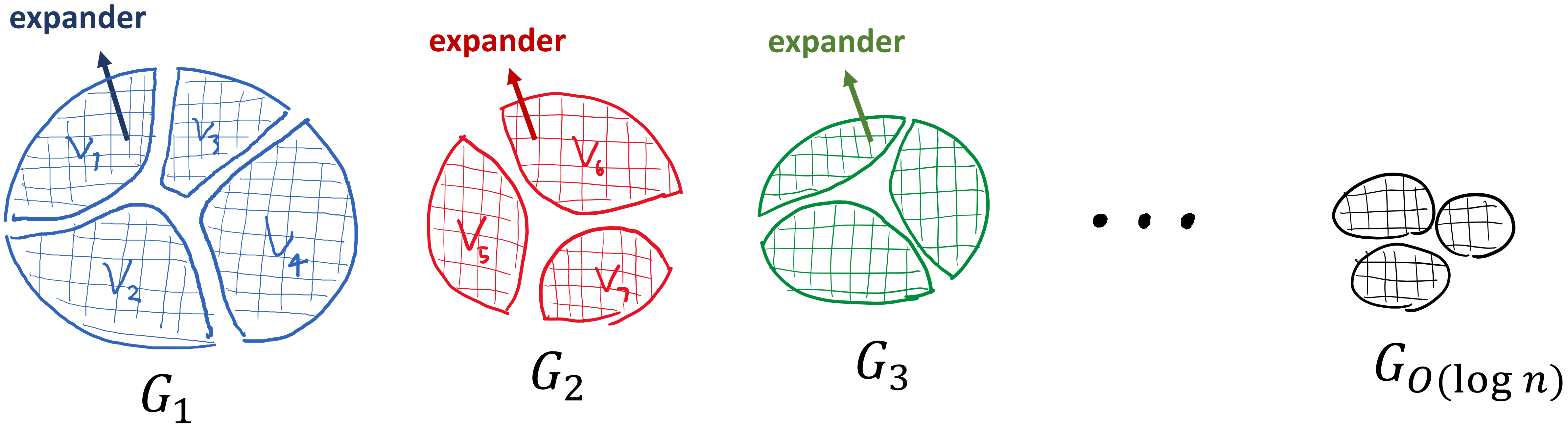
# Repeated Expander Decomposition



# Repeated Expander Decomposition



# Repeated Expander Decomposition



# Repeated Expander Decomposition

**Input:** $G = (V, E)$

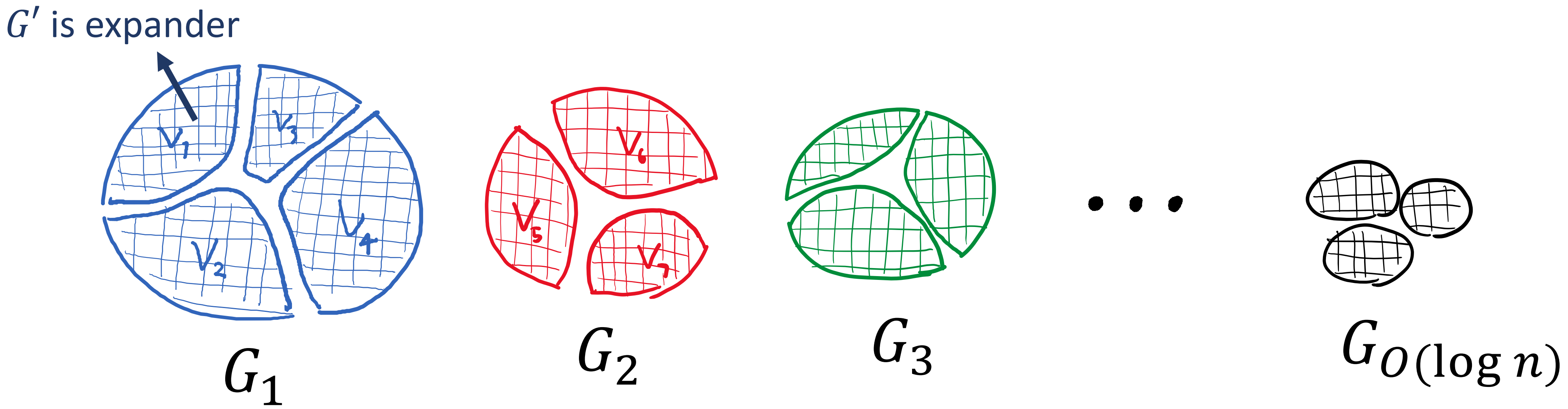
**Output:** $(G_1, \ldots, G_{O(\log n)})$ such that

- $G_i$ = disjoint union of expanders
- $E = E(G_1) \,\dot{\cup}\, \ldots \,\dot{\cup}\, E\left(G_{O(\log n)}\right)$

**Time:** $\tilde{O}(m)$

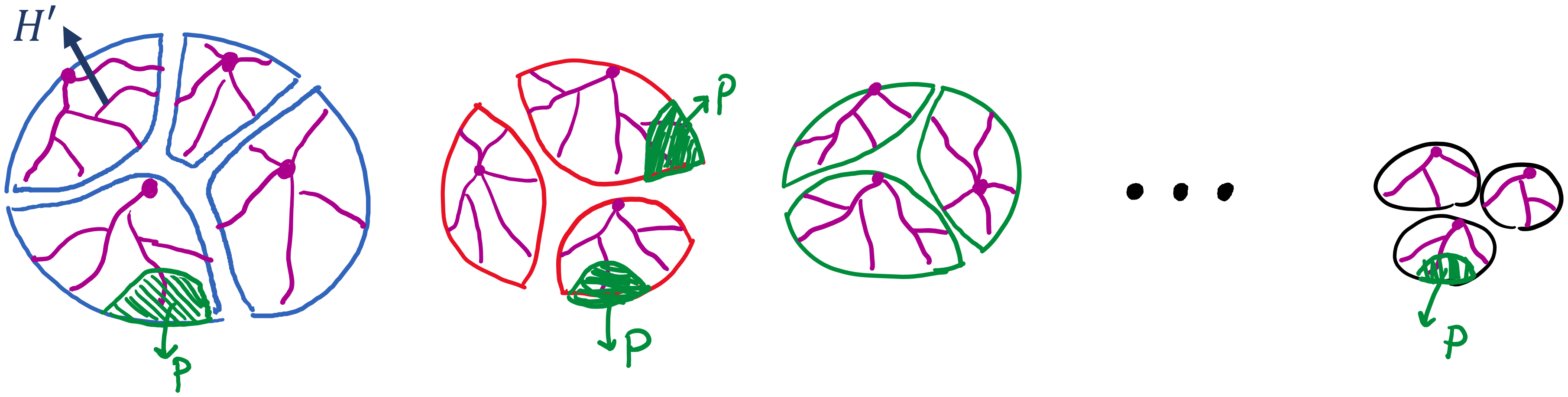
# Dynamic Spanning Subgraph: General Graphs

1. **Preprocess**: Compute repeated expander decomposition $(G_1, \dots, G_{O(\log n)})$
2. **Algo**: Each expander $G'$, maintain spanning subgraph $H'$ of $G'$
   - $H'$ has $O(|V(G')|)$ edges
   - Update time $n^{o(1)}$ (if the update is on $G'$)



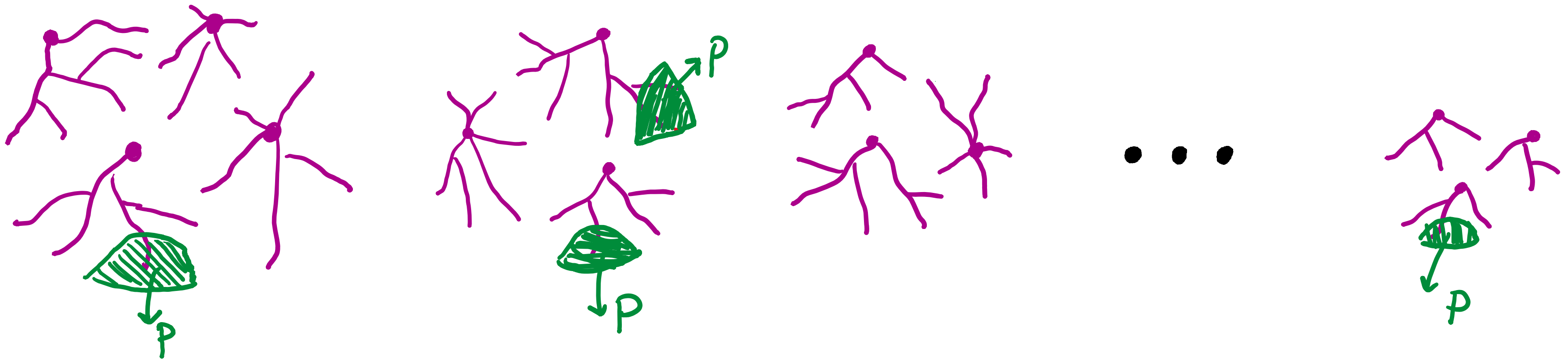
# Dynamic Spanning Subgraph: General Graphs

1. **Preprocess**: Compute repeated expander decomposition $(G_1, \dots, G_{O(\log n)})$
2. **Algo**: Each expander $G'$, maintain spanning subgraph $H'$ of $G'$
   - $H'$ has $O(|V(G')|)$ edges
   - Update time $n^{o(1)}$ (if the update is on $G'$)
3. **Claim**: Union of all $H'$ is a spanning subgraph of $G$ with $O(n \log n)$ edges.



# Dynamic Spanning Subgraph: General Graphs

1. **Preprocess**: Compute repeated expander decomposition $(G_1, \dots, G_{O(\log n)})$
2. **Algo**: Each expander $G'$, maintain spanning subgraph $H'$ of $G'$
   - $H'$ has $O(|V(G')|)$ edges
   - Update time $n^{o(1)}$ (if the update is on $G'$)
3. **Claim**: Union of all $H'$ is a spanning subgraph of $G$ with $O(n \log n)$ edges.



# Dynamic Spanning Subgraphs

## Conclusion:

Given $G$ undergoing edge updates,

- maintain spanning subgraph
- with $O(n \log n)$ edges
- in $n^{o(1)}$ update time

# Part 2

# Centralized Algorithms

# Expander Paradigm

## 1. Solve it on **expanders**.

Problem specific:

**e.g. Random Sampling**

~~eral t~~

~~**Exp der P ing**~~

## 2. **Combine** the solutions.

General tool:

**Expander Decomposition**

# Definition: Spanner

**Informal**: Subgraph that preserves all distances.

# **Definition**: Spanner

Let $G = (V, E)$.

$H = (V, E')$ is a $\boldsymbol{k}$**-spanner** of $G$ if

1. $E' \subset E$
2. $\forall (u, v) \in E, \mathrm{dist}_{\mathrm{H}}(\mathrm{u}, \mathrm{v}) \leq k$

# Spanners of Expanders

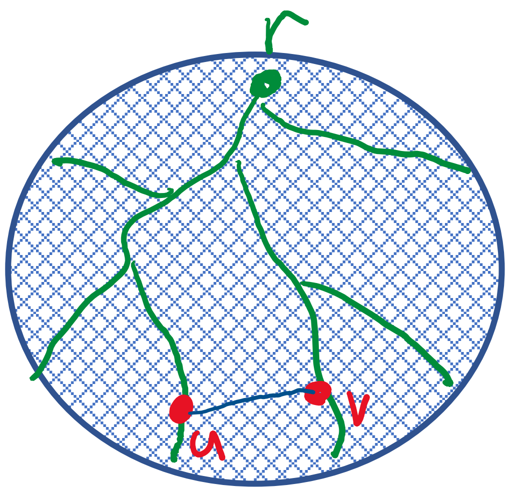
$G$: expander

$T$: a shortest path tree in $G$ (rooted at an arbitrary node $r$).

**Observe**: $T$ is a $\text{polylog}(n)$-spanner of $G$

**Proof**: $\forall (u, v) \in E, \text{dist}_T(u, v) \leq \text{dist}_T(u, r) + \text{dist}_T(r, v)$

$$= \text{polylog}(n)$$



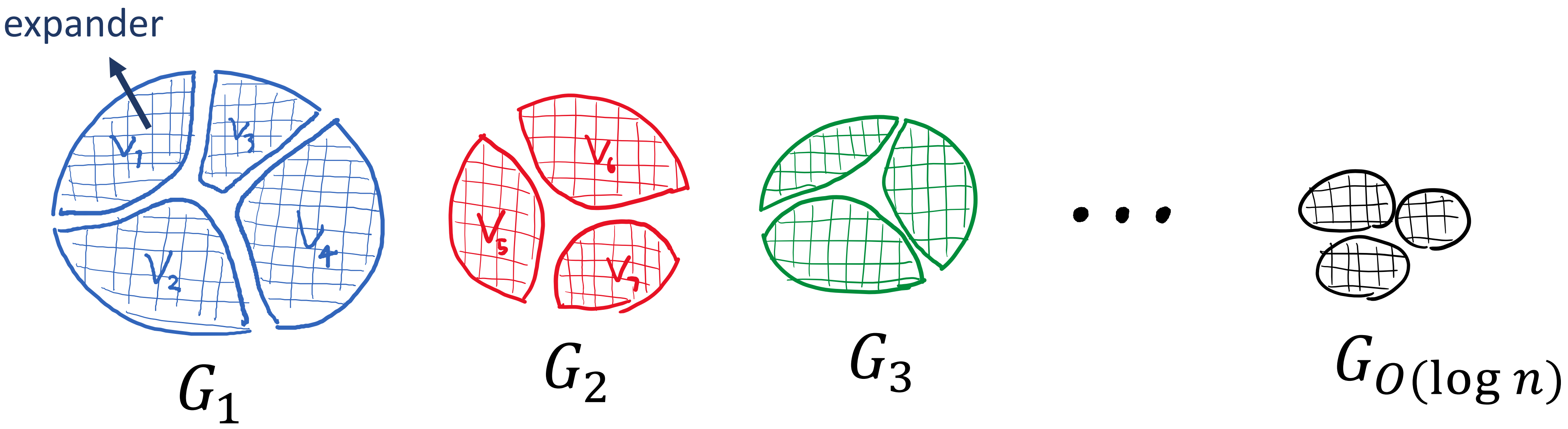

$G$: $\phi$-expander

**Fact**: Diameter of expanders is $\text{polylog}(n)$.

# Spanners of General Graphs

Spanner(G):

1. Compute repeated expander decomposition: $(G_1, \dots, G_{O(\log n)})$
2. $H_i$ = Shortest path tree on each expander of $G_i$



# Spanners of General Graphs

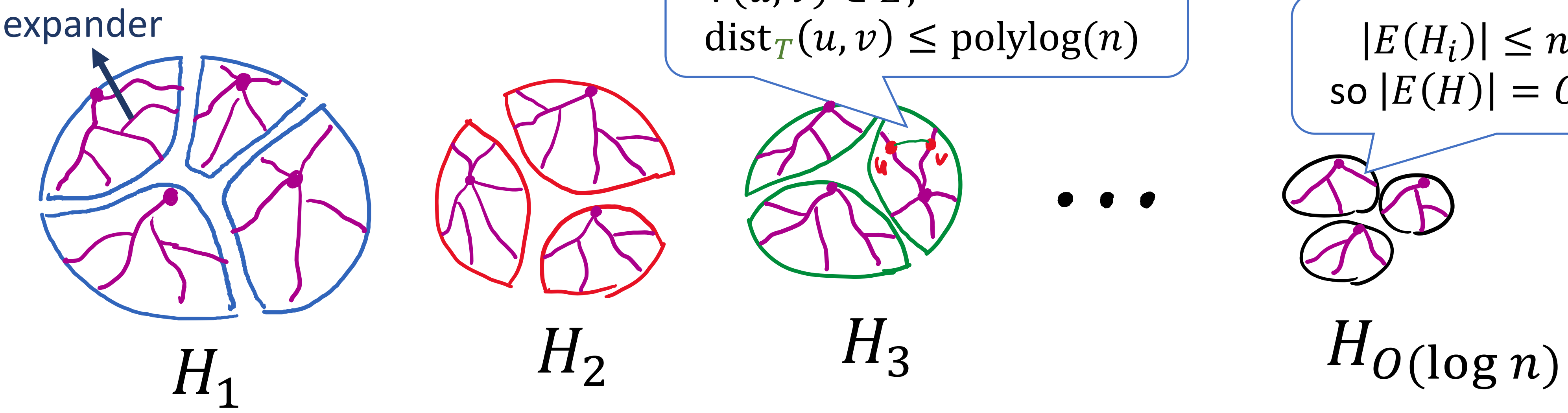
Spanner(G):

1. Compute repeated expander decomposition: $(G_1, \ldots, G_{O(\log n)})$
2. $H_i$ = Shortest path tree on each expander of $G_i$
3. Return $H = \cup_i H_i$

Total time: $\tilde{O}(m)$

$\forall (u, v) \in E$,
$\mathrm{dist}_T(u, v) \leq \mathrm{polylog}(n)$

$|E(H_i)| \leq n$ (forest),
so $|E(H)| = O(n \log n)$.



# Spanners of General Graphs

## Conclusion:

Given $G$,

- a $\text{polylog}(n)$-spanner
- with $O(n \log n)$ edges
- in $\tilde{O}(m)$ time

# Expander Paradigm

## 1. Solve it on **expanders**.

Problem specific:
**Shortest path tree**

Problem specific:
**Random sampling**

## 2. **Combine** the solutions.

General tool:
**Expander Decomposition**

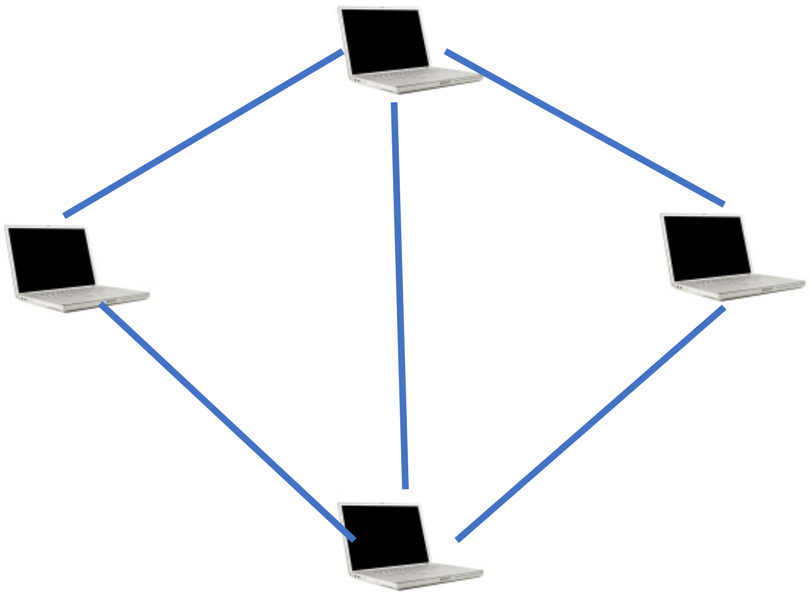
**More applications:**

- **Cut sparsifiers**: preserve cut sizes
- **Spectral sparsifiers**: preserve eigenvalues

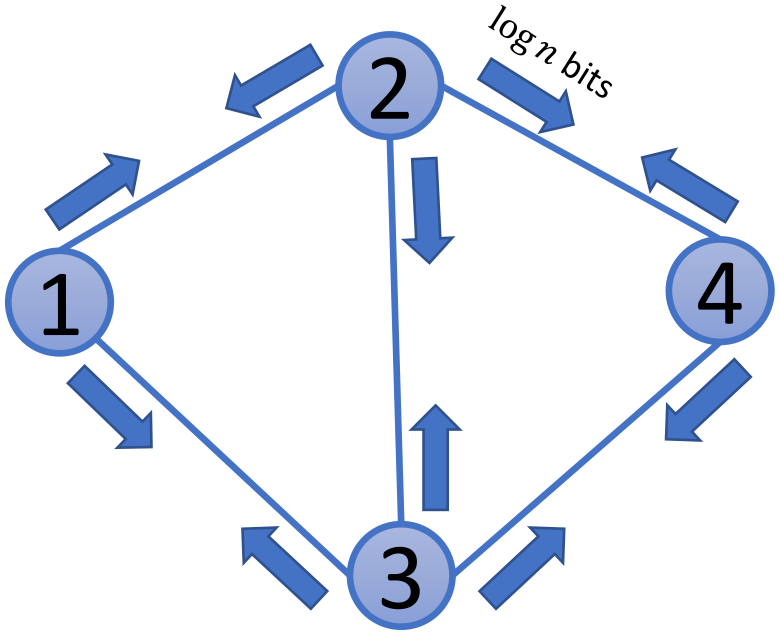
Part 3

# Distributed Algorithms

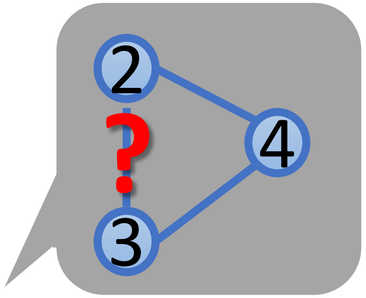
# Definition: CONGEST model



# Definition: CONGEST model




- **Local knowledge:**
  A node know only its neighbors




- **Local communication**:
  A node can send messages to only its neighbors in each *round*
- **Bounded Bandwidth:**
  Each message has size $O(\log n)$-bit

**Goal:**

- Compute something about the underlying network
- Minimize the number of rounds

# Expander Paradigm (Distributed)

## 1. Solve it on **expanders**.

Problem specific:

**e.g. Random Sampling**

General tool:

**Expander Routing**

## 2. **Combine** the solutions.

General tool:

**Expander Decomposition**

# Expander Routing (Informal)

[Ghaffari Kuhn Su PODC'17] [Ghaffari Li DISC'18]

A node $u$ can exchange $\deg_G(u)$ messages
with **any set of nodes**
in $n^{o(1)}$ rounds in an **expander**

**Expanders** allow **global communication**
with small overhead

**Local** communication

In **any** graph,
can exchange with
**only neighbors**
in 1 round

# Expander Routing

[Ghaffari Kuhn Su PODC'17] [Ghaffari Li DISC'18]

**Input:** *underlying* graph $G = (V, E)$ and *demand* graph $D = (V, E')$

- $G$: expander
- $\deg_D(u) \leq \deg_G(u)\ \forall\, u \in V$

**Output:**

- for all $(u, v) \in E'$ **simultaenously**,
- $u$ and $v$ can exchange a message in $n^{o(1)}$ rounds (in $G$)

**Expanders** allow **global communication**

with small overhead

# Expander Paradigm (Distributed)

1. Solve it on **expanders**.

Problem specific:
**e.g. Random Sampling**

General tool:
**Expander Routing**

Can import ideas from algorithms in **CONGESTED-CLIQUE** model

2. **Combine** the solutions.

General tool:
**Expander Decomposition**

**Round complexity**:

- $n^{1-\epsilon}$ [Chang Pettie Zhang SODA'19] (with caveat)
- $n^{\epsilon}$ [Chang **S** PODC'19]
- $\mathrm{polylog}(n)$ [Chang **S**]

# Part 4

# Conclusion:
# Survey and Open Problems

# Centralized Setting

# Expander Paradigm (Centralized)

1. Solve it on **expanders**.

Problem specific:

**e.g. Random Sampling**

2. **Combine** the solutions.

General tool:

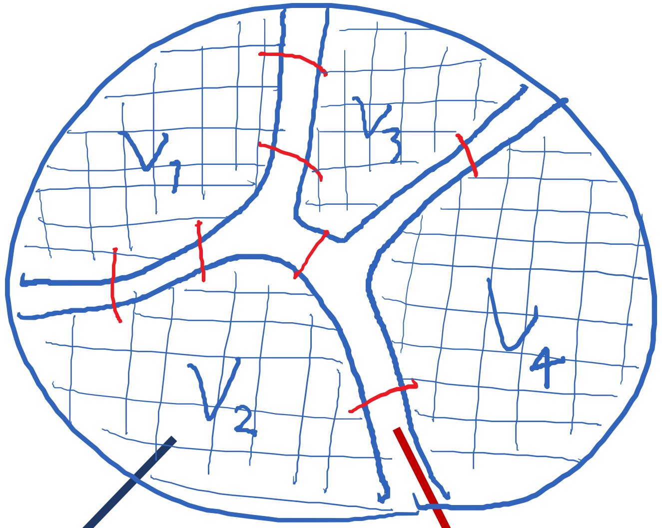
**Expander Decomposition**

# Expander Decomposition

[S Wang SODA'19]:
$\tilde{O}(m)$-time w.h.p.

**Input:** $G = (V, E)$

**Output:** A partition $(V_1, \ldots V_k)$ of $V$




"**Graph** = Disjoint **Expanders** + **Few Edges**"

$G[V_i]$ **is expander**

$\leq m/2$ **inter-cluster edges**

# Fast **Centralized** Algorithms

**New!** [Chuzhoy Gao Li Nanongkai Peng S]:
Expander decomposition in
$m^{1+o(1)}$ **deterministic** time

| | Time (Randomized) |
|---|---|
| Laplacian system solvers<br>[Spielman Teng STOC'04] | $\tilde{O}(m)$ |
| Spectral sparsifiers<br>[Spielman Teng STOC'04] | $\tilde{O}(m)$ |
| Approx. max flow<br>[Kelner Lee Orecchia Sidford SODA'14] | $\tilde{O}(m)$ |
| Approx. vertex max flow<br>[Chuzhoy Khanna STOC'19] | $\tilde{O}(n^2)$ |
| Bipartite Matching, Shortest Path, Max flow<br>[Cohen Madry Sankowski Vladu SODA'17] | $\tilde{O}(m^{10/7})$ |

**Expander Paradigm is the key to all these results**

# Fast **Centralized** Algorithms

**Open:** Expander decomposition in $\tilde{O}(m)$ **deterministic** time (would remove all $m^{o(1)}$ below)

| | Time (Randomized) | Time (Deterministic) [CGLNPS] |
|---|---|---|
| Laplacian system solvers [Spielman Teng STOC'04] | $\tilde{O}(m)$ | $m^{1+o(1)}$ |
| Spectral sparsifiers [Spielman Teng STOC'04] | $\tilde{O}(m)$ | $m^{1+o(1)}$ |
| Approx. max flow [Kelner Lee Orecchia Sidford SODA'14] | $\tilde{O}(m)$ | $m^{1+o(1)}$ |
| Approx. vertex max flow [Chuzhoy Khanna STOC'19] | $\tilde{O}(n^2)$ | $n^{2+o(1)}$ |
| Bipartite Matching, Shortest Path, Max flow [Cohen Madry Sankowski Vladu SODA'17] | $\tilde{O}(m^{10/7})$ | $m^{10/7+o(1)}$ |

**Expander Paradigm is the key to all these results**

# Dynamic Setting

**<u>Non-adaptive users:</u>**

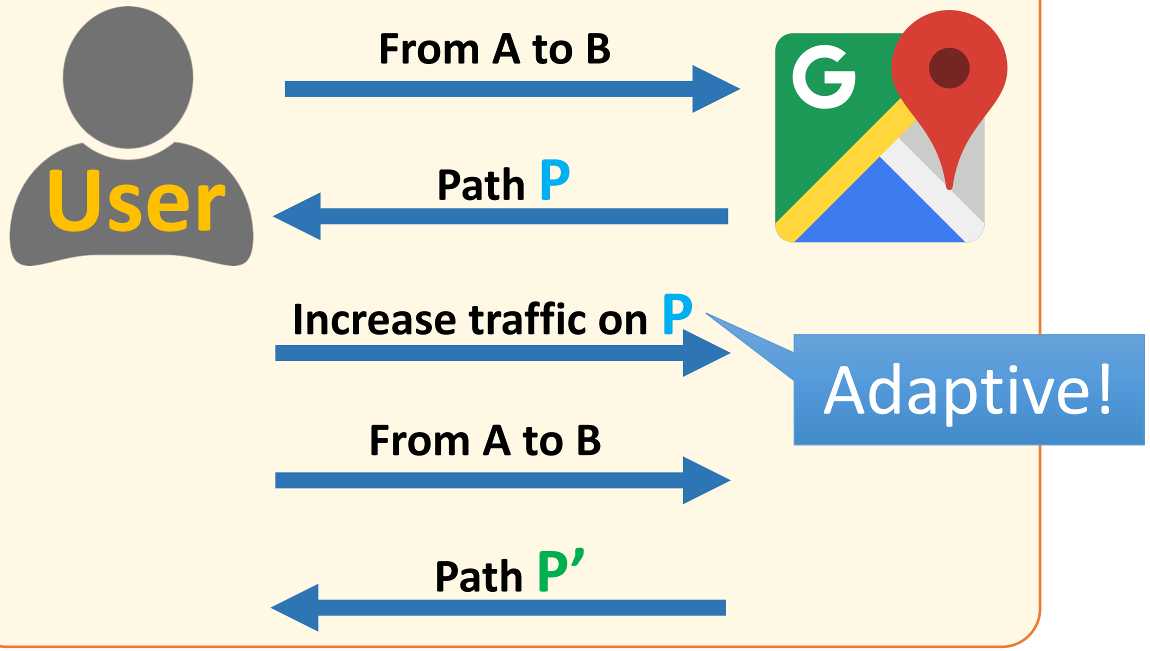
All updates are **fixed from the beginning**.

Usually **cannot** be used as subroutines inside static algo.

**<u>Adaptive users:</u>**

Updates from users can **depend on previous answers**

**Example:**



# Frontier of Dynamic Graph Algorithms

## We DON'T know how to serve adaptive users!

| Problems | Non-adaptive users | Adaptive users | |
|---|---|---|---|
| Spanning Forests<br>(worst case) | $\mathrm{polylog}\, n$<br>[Kapron King Mountjoy SODA'13] | $\sqrt{n}$<br>[EGIN FOCS'92] | |

# Frontier of Dynamic Graph Algorithms

## We DON'T know how to serve adaptive users!

| Problems | Non-adaptive users | Adaptive users | |
|---|---|---|---|
| Spanning Forests (worst case) | $\text{polylog}\, n$ [Kapron King Mountjoy SODA'13] | $\sqrt{n}$ [EGIN FOCS'92] | |
| Spanners (amortized) | $\text{polylog}\, n$ [BKS ESA06, SODA'08] | $m$ [trivial] | |
| Single Source Shortest Paths (decremental approximate amortized) | $m^{1+o(1)}$ [HKN FOCS'14] | $mn$ [Even Shiloah'81] | |
| Single Source Reachability (decremental amortized) | $m$ [BPW STOC'19] | $mn$ [Even Shiloah'81] | |
| Cut Sparsifiers (worst-case) | $\text{polylog}\, n$ [ADKKP FOCS'16] | $m$ [trivial] | |
| Maximal Matching | $O(1)$ [Solomon FOCS'16] | $\sqrt{m}$ [Neiman Solomon STOC'13] | |

# Frontier of Dynamic Graph Algorithms

**Expander Paradigm can help in many cases!**

| Problems | Non-adaptive users | Adaptive users | Adaptive users (by Expander Decomposition) |
|---|---|---|---|
| Spanning Forests (worst case) | $\mathrm{polylog}\, n$ [Kapron King Mountjoy SODA'13] | $\sqrt{n}$ [EGIN FOCS'92] | $n^{o(1)}$ [NSW FOCS'17] |
| Spanners (amortized) | $\mathrm{polylog}\, n$ [BKS ESA06, SODA'08] | $m$ [trivial] | $n^{o(1)}$ [BNSSS FOCS'17] |
| Single Source Shortest Paths (decremental approximate amortized) | $m^{1+o(1)}$ [HKN FOCS'14] | $mn$ [Even Shiloah'81] | $n^{2+o(1)}$ [Bernstein Chechik STOC'16] [CS] |
| Single Source Reachability (decremental amortized) | $m$ [BPW STOC'19] | $mn$ [Even Shiloah'81] | - |
| Cut Sparsifiers (worst-case) | $\mathrm{polylog}\, n$ [ADKKP FOCS'16] | $m$ [trivial] | - |
| Maximal Matching | $O(1)$ [Solomon FOCS'16] | $\sqrt{m}$ [Neiman Solomon STOC'13] | - |

We saw this (simplified)

# Expander Paradigm (Dynamic)

1. Solve it on **expanders**.

Problem specific:
**e.g. Random Sampling**

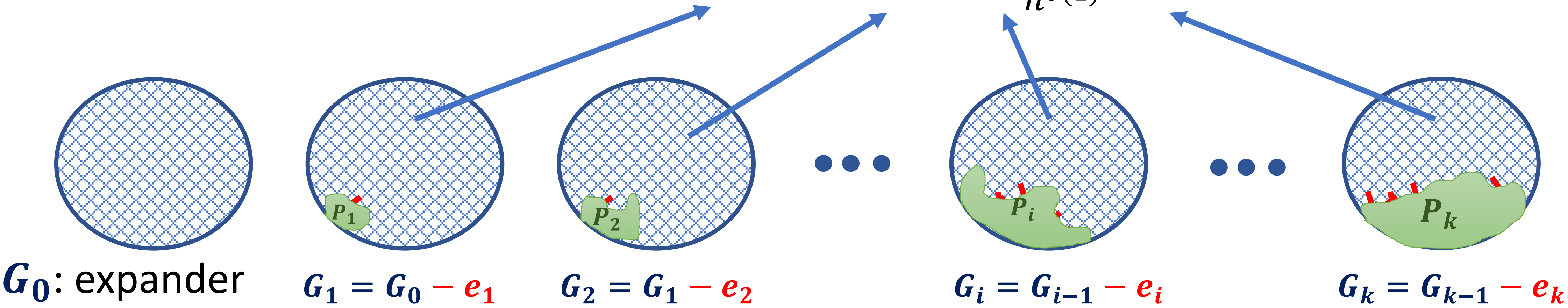
General tool:
**Expander Pruning**

2. **Combine** the solutions.

General tool:
**Expander Decomposition**

# Expander Pruning [NSW'17]

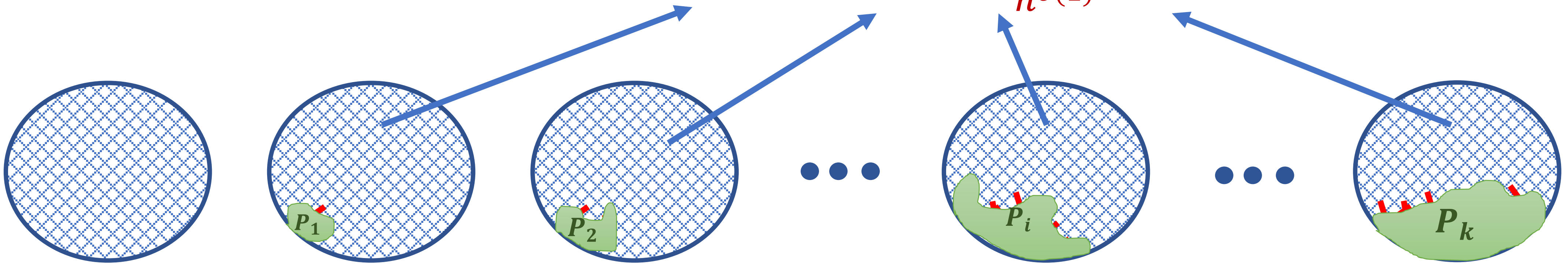
$G_i[V - \boldsymbol{P_i}]$ is a $(\frac{1}{n^{o(1)}})$-**expander**




**Expanders** can be **quickly "repaired"** under edge updates.

# Expander Pruning [NSW'17]

$G_i[V - P_i]$ is a $(\frac{1}{n^{o(1)}})$-expander




$G_0$: expander  $G_1 = G_0 - e_1$  $G_2 = G_1 - e_2$  $G_i = G_{i-1} - e_i$  $G_k = G_{k-1} - e_k$

where $k \le m/n^{o(1)}$

**Guarantee:**

1. Time to update $P_{i-1}$ to $P_i$ is $n^{o(1)}$
2. So $vol(P_i) = i \cdot n^{o(1)}$
3. $G_i[V - P_i]$ is a $\frac{1}{n^{o(1)}}$-expander

**Open:**

Improve $n^{o(1)}$ to $\text{polylog}(n)$

imply $\text{polylog}(n)$ worst-case update time for many problems (e.g. spanning subgraphs, spectral sparsifiers)

# Distributed Setting

# Expander Paradigm (Distributed)

1. Solve it on **expanders**.

Problem specific:
**e.g. Random Sampling**

General tool:
**Expander Routing**

2. **Combine** the solutions.

General tool:
**Expander Decomposition**

# Expander Routing

[Ghaffari Kuhn Su PODC'17] [Ghaffari Li DISC'18]

A node $u$ can exchange $\deg_G(u)$ messages
with **any set of nodes**
in $n^{o(1)}$ rounds in an **expander**

**Expanders** allow
**global communication**
with small overhead

**Open:**
Improve $n^{o(1)}$ to $\mathrm{polylog}(n)$
(Many applications even in
centralized setting (chat offline))

# Distributed CONGEST algorithm

| | Upper bound | Lower bound |
|---|---|---|
| Triangle (3-clique) listing | $\tilde{O}(n^{1/3})$ [Chang Pettie Zhang SODA'18] [Chang **S** PODC'19] | $\tilde{\Omega}(n^{1/3})$ [Izumi LeGall PODC'17] |
| 4-clique listing | $\tilde{O}(n^{5/6})$ [Eden Fiat Fischer Kuhn Oshman DISC'19] | $\tilde{\Omega}(n^{1/2})$ [Fischer Gonen Kuhn Oshman SPAA'18] |
| 5-clique listing | $\tilde{O}(n^{21/22})$ [Eden et al. DISC'19] | $\tilde{\Omega}(n^{3/5})$ [Fischer et al. SPAA'18] |
| $k$-vertex subgraph detection | $n^{2-\Omega(1/k)}$ [Eden et al. DISC'19] | $n^{2-O(1/k)}$ [Fischer et al. SPAA'18] |

**Expander Paradigm used in all upper bounds**

# Distributed CONGEST algorithm

| | Upper bound | Lower bound |
|---|---|---|
| Triangle (3-clique) listing | $\tilde{O}(n^{1/3})$ [Chang Pettie Zhang SODA'18] [Chang **S** PODC'19] | $\tilde{\Omega}(n^{1/3})$ [Izumi LeGall PODC'17] |
| 4-clique listing | $\tilde{O}(n^{5/6})$ [Eden Fiat Fischer Kuhn Oshman DISC'19] | $\tilde{\Omega}(n^{1/2})$ [Fischer Gonen Kuhn Oshman SPAA'18] |
| 5-clique listing | $\tilde{O}(n^{21/22})$ [Eden et al. DISC'19] | $\tilde{\Omega}(n^{3/5})$ [Fischer et al. SPAA'18] |
| $k$-vertex subgraph detection | $n^{2-\Omega(1/k)}$ [Eden et al. DISC'19] | $n^{2-O(1/k)}$ [Fischer et al. SPAA'18] |
| $k$-clique enumeration | ? | $\tilde{\Omega}(n^{1-2/k})$ [Fischer et al. SPAA'18] |

**Open:**

Application which is not *subgraph detection/listing*

# History: Distributed Expander Decomposition

| Reference | Rounds | Note |
|---|---|---|
| [Chang Pettie Zhang SODA'19] | $n^{1-\delta}$ | Output an extra part:<br>a subgraph with arboricity $n^{\delta}$ |
| [Chang **S** PODC'19] | $n^{\epsilon}$ | |
| [Chang **S** in progress] | $\mathrm{polylog}(n)$ | |
| [Chang **S** in progress] | $n^{\epsilon}$ | **Deterministic** |
| **Open:** | $\mathrm{polylog}(n)$ | **Deterministic** |