Faster Gomory-Hu Trees in Simple Graphs

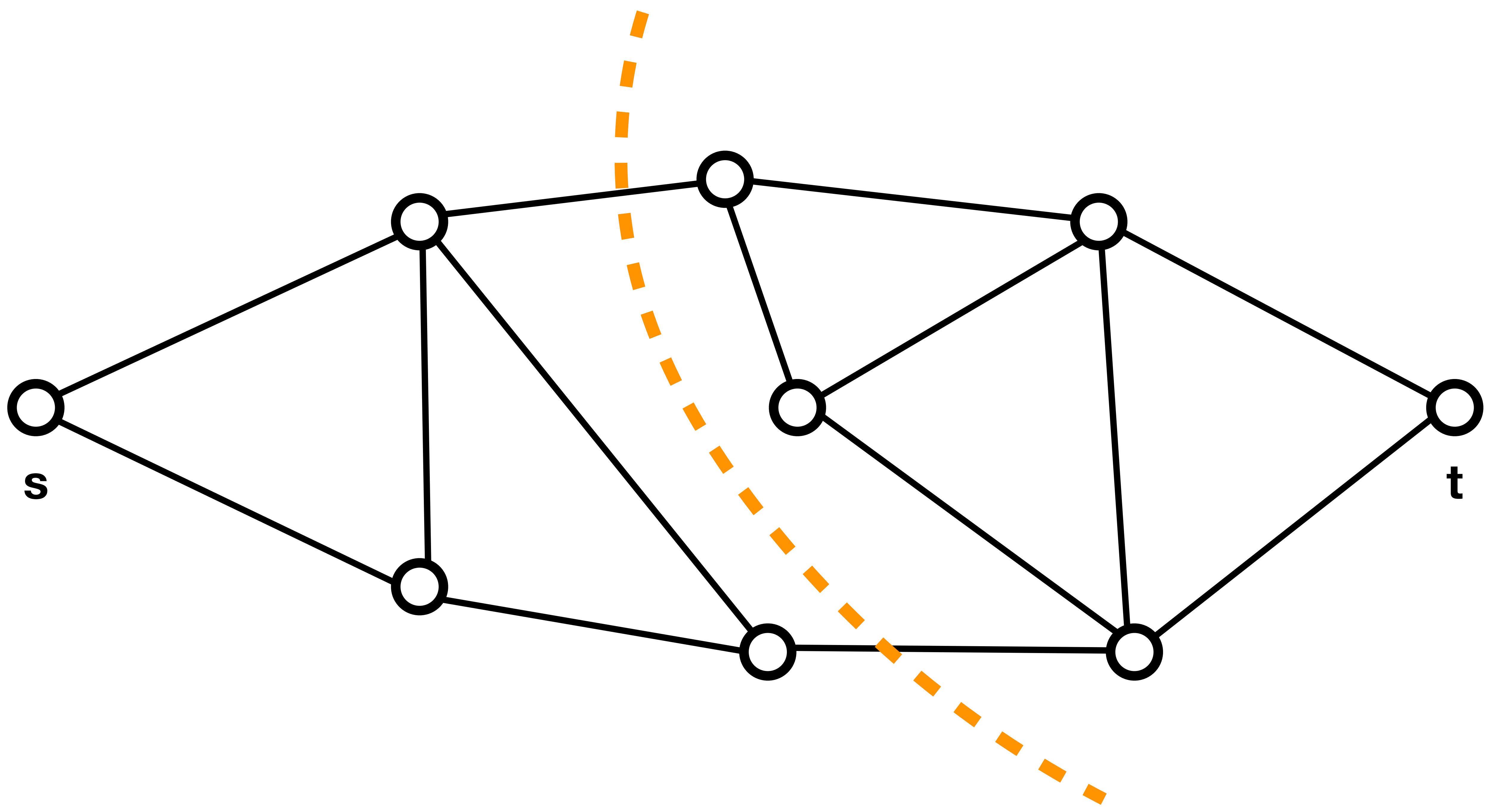
Tianyi Zhang

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

Problem Definition

All-Pairs Minimum Cuts

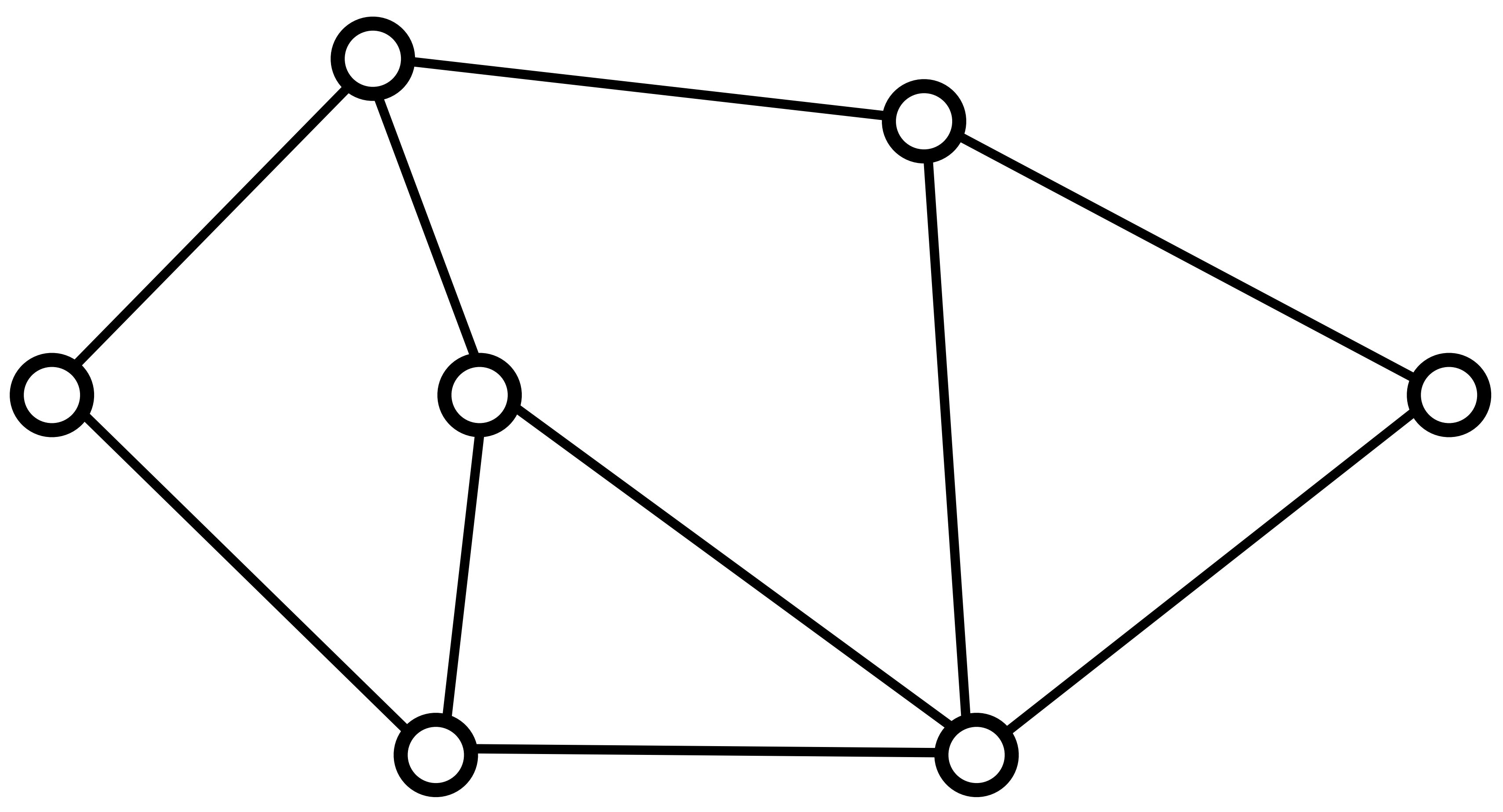
- **Input:** an undirected **simple** graph $G = (V, E)$, n vertices and m edges
- **Output:** for every pair $s, t \in V$, the **s-t min-cut value** in G



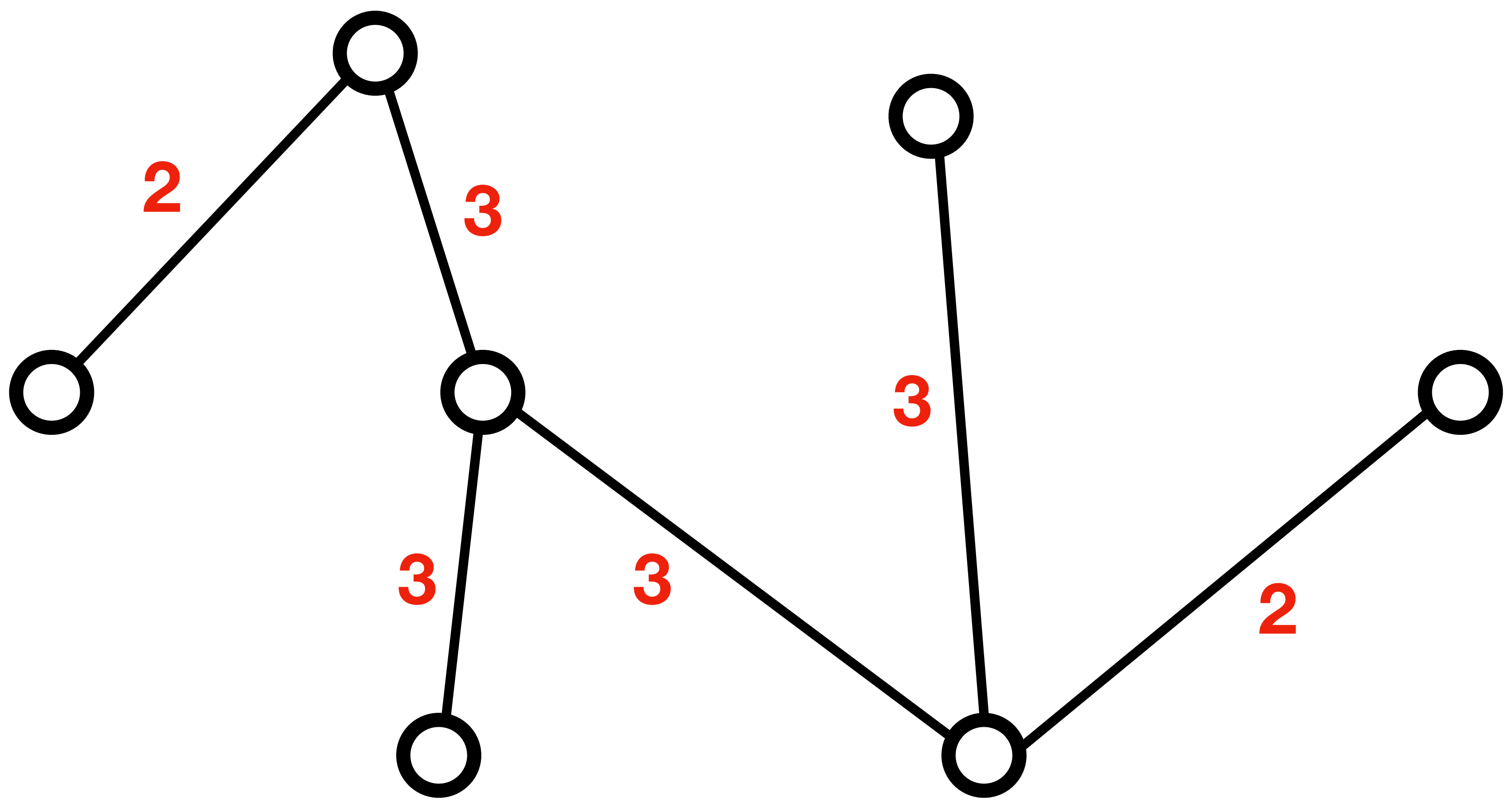
Gomory-Hu Tree

Theorem [GH61]:

Given any $G = (V, E)$, there exists an edge-weighted tree $T = (V, F)$, such that any s-t min-cut in T is also an s-t min-cut in G



$G = (V, E)$

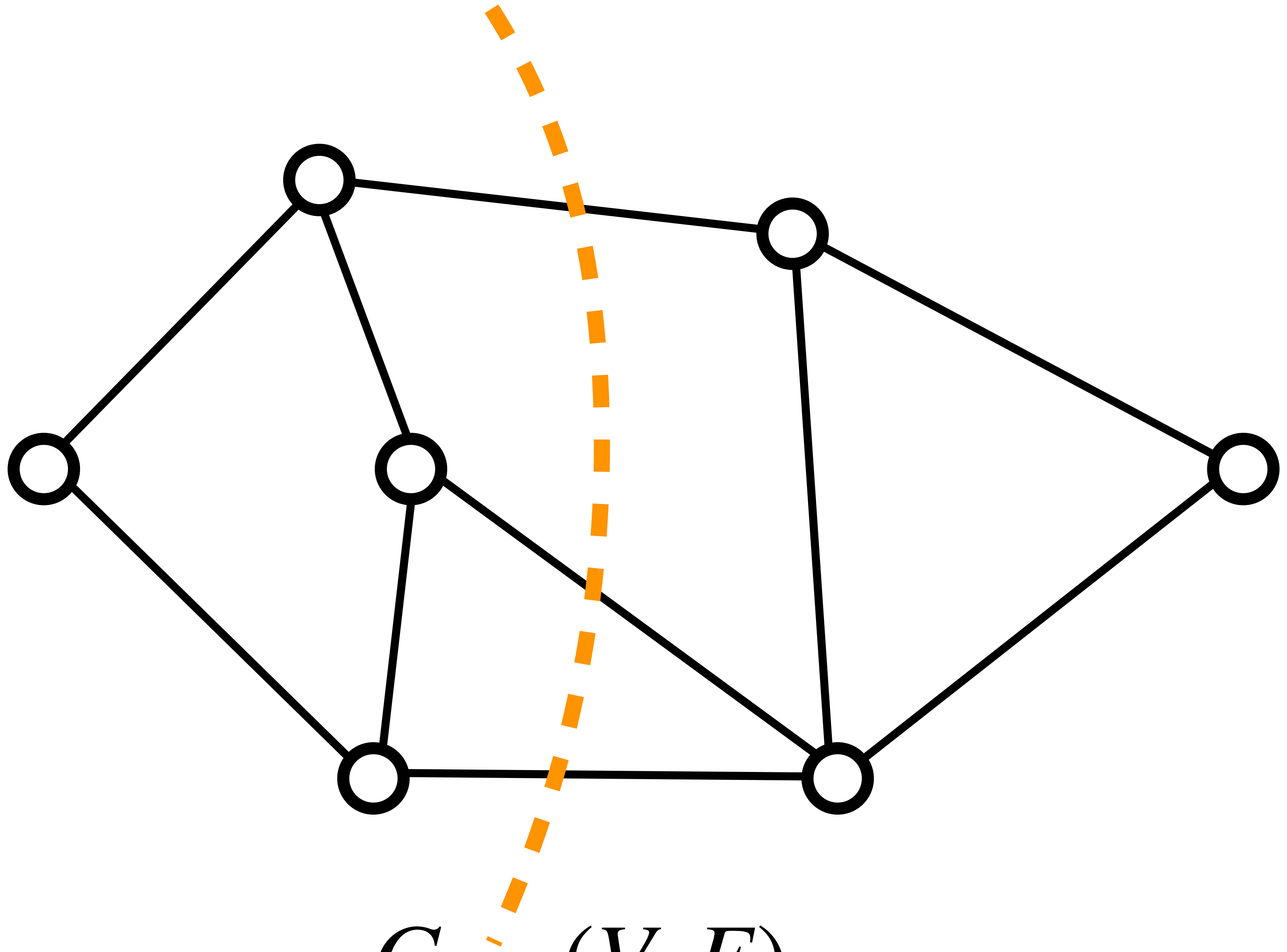


$T = (V, F)$

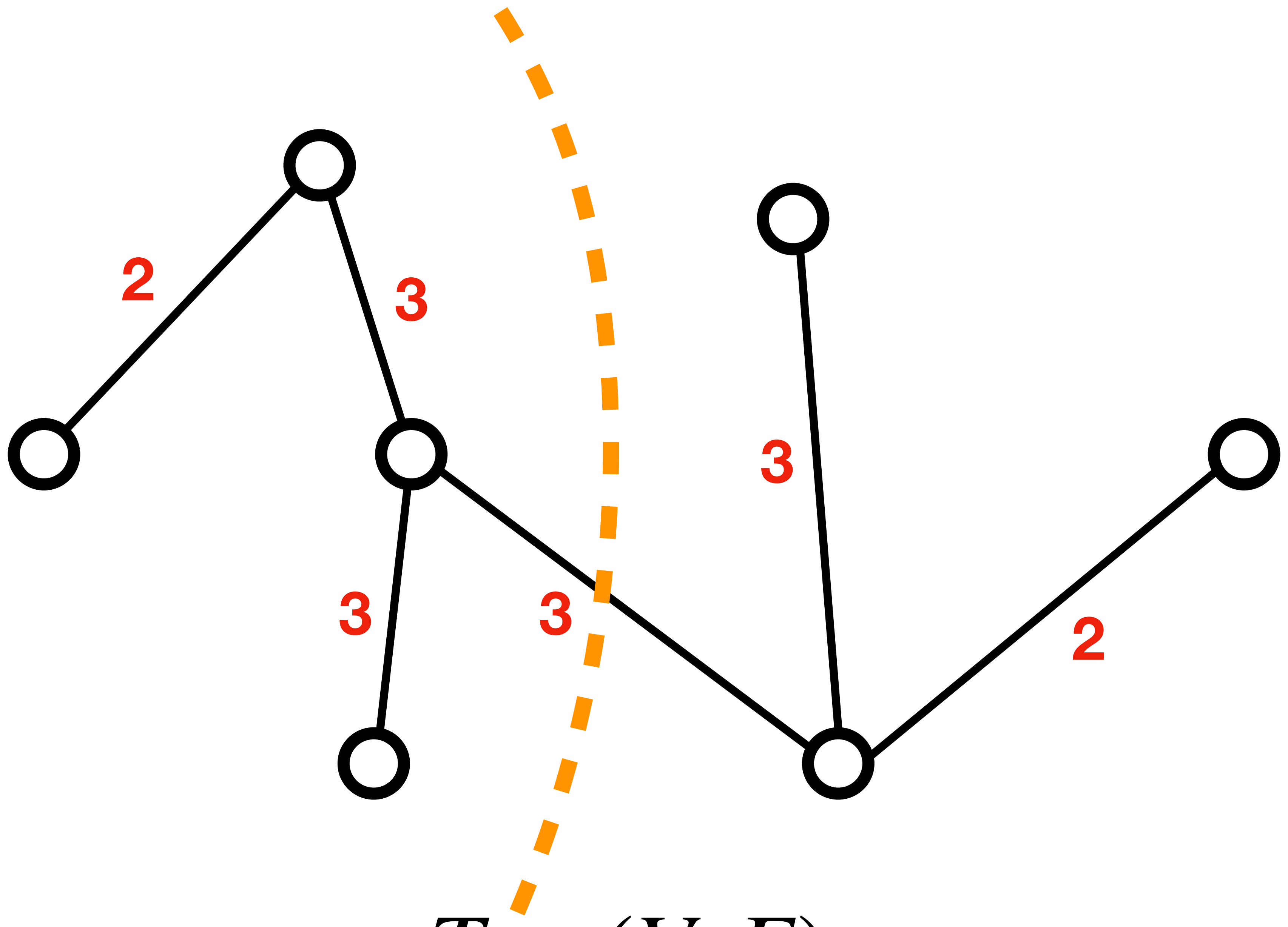
Gomory-Hu Tree

Theorem [GH61]:

Given any $G = (V, E)$, there exists an edge-weighted tree $T = (V, F)$, such that any s-t min-cut in T is also an s-t min-cut in G



$G = (V, E)$

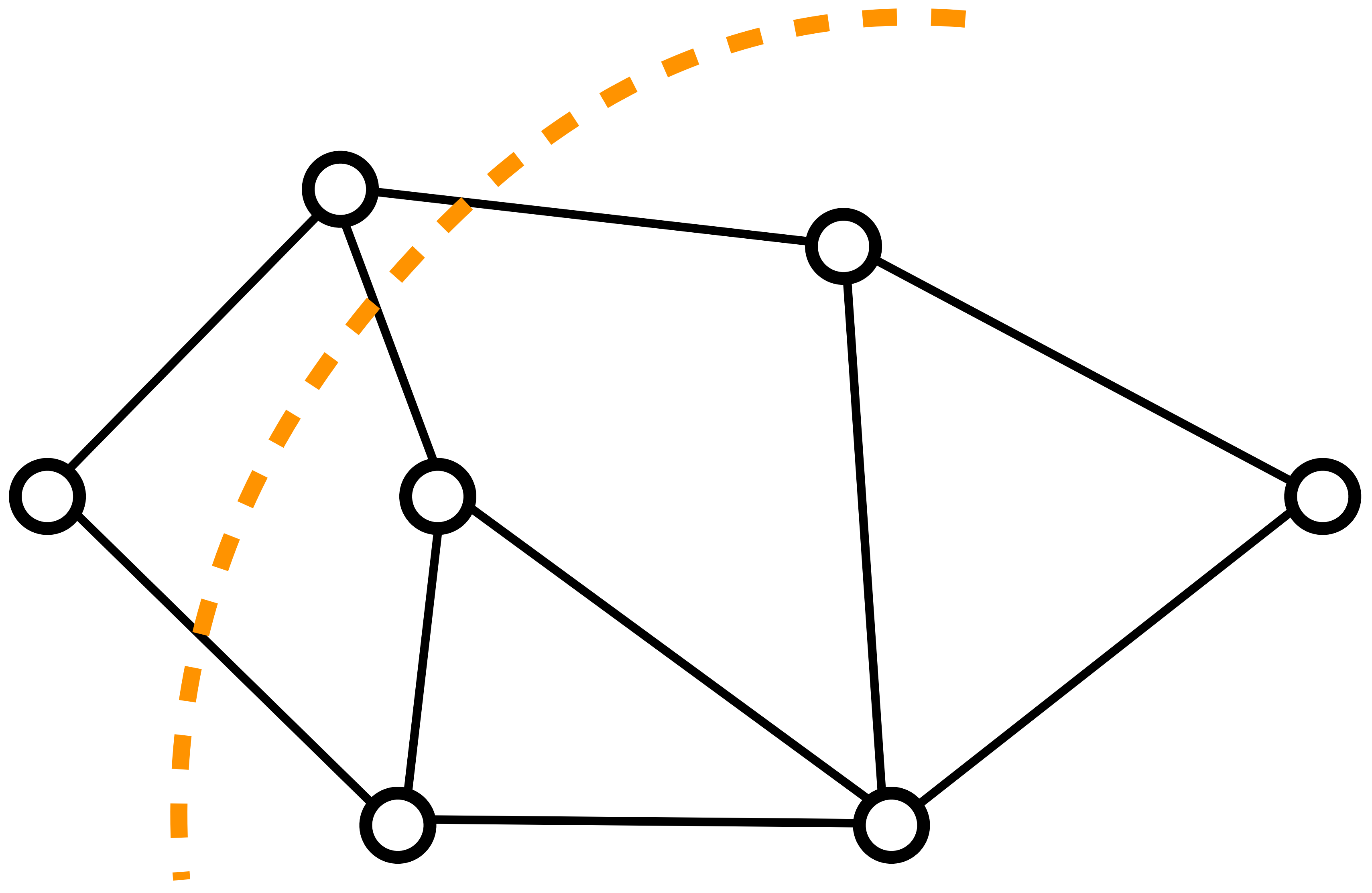


$T = (V, F)$

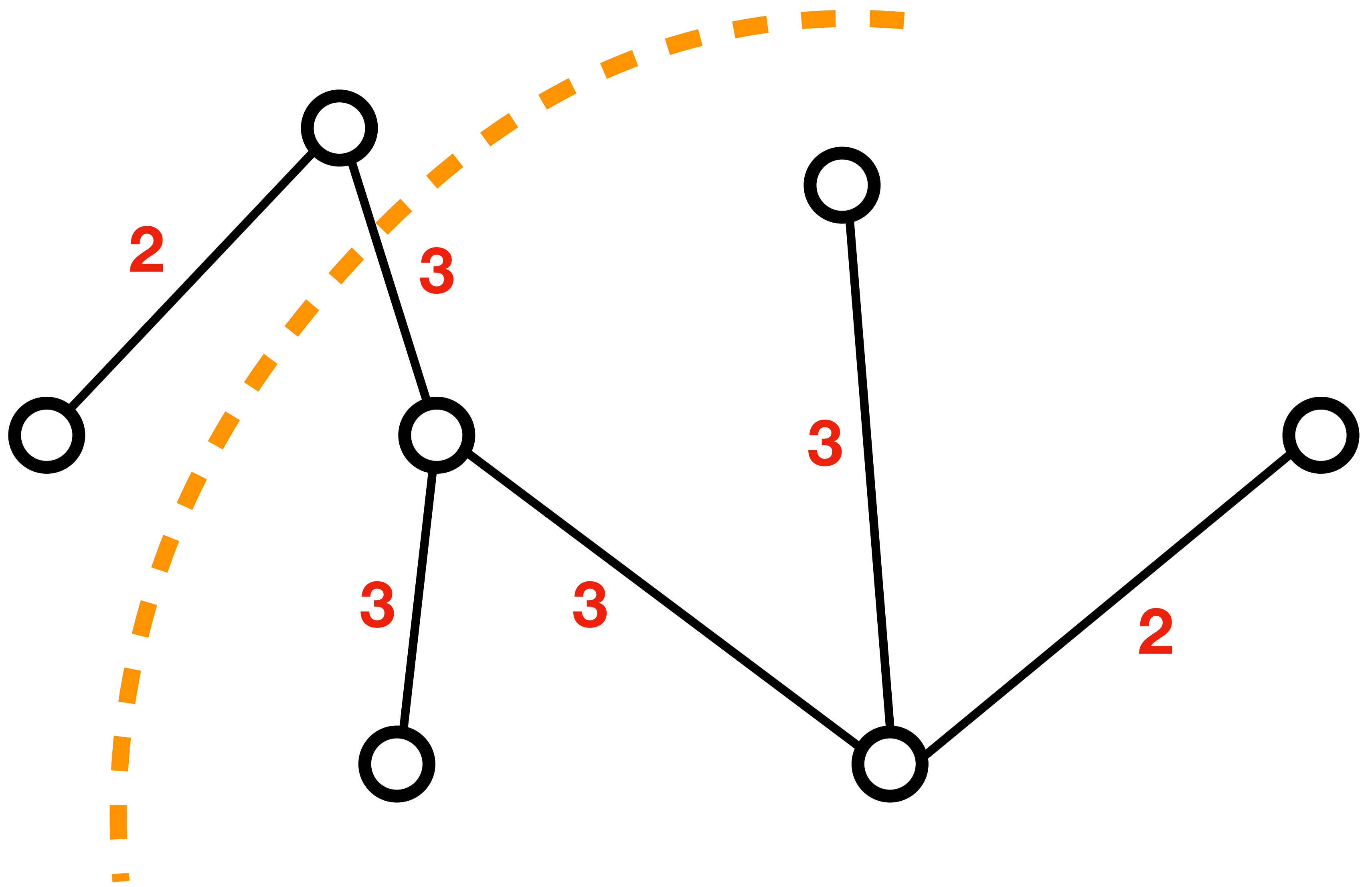
Gomory-Hu Tree

Theorem [GH61]:

Given any $G = (V, E)$, there exists an edge-weighted tree $T = (V, F)$, such that any s-t min-cut in T is also an s-t min-cut in G



$G = (V, E)$



$T = (V, F)$

Gomory-Hu Tree

Theorem [GH61]:

Given any $G = (V, E)$, there exists an edge-weighted tree $T = (V, F)$, such that any s-t min-cut in T is also an s-t min-cut in G

All-Pairs Min-Cuts:

Given a Gomory-Hu tree $T = (V, F)$ of $G = (V, E)$, can query any s-t minimum cut in $\tilde{O}(1)$ time, so total time = runtime of GH + $\tilde{O}(n^2)$

History

reference	total size of max-flow instances	runtime	graph type
Gomory & Hu 1961	mn	$mn + n^{2.5}$	edge-weighted
Hariharan, Kavitha, Panigrahi, Bhalgat 2007		mn	simple
Abboud, Krauthgamer, Trabelsi 2021	$n^{2.5}$	$n^{2.5}$	simple
Abboud, Krauthgamer, Trabelsi 2021	$n^{2+o(1)}$	$n^{2+o(1)}$	simple
Li, Panigrahi, Saranurak 2021	$n^{2+o(1)}$	$n^{2+o(1)}$	simple
Abboud, Krauthgamer, Trabelsi 2022	$(m + n^{1.75})^{1+o(1)}$	$(m + n^{1.9})^{1+o(1)}$	simple
Ours	n^2	$n^{17/8}$	simple
Abboud et al, 2022	m	n^2	edge-weighted

For real runtime, assume MaxFlow(m, n) = $m + n^{1.5}$ [BLL+, 2021]

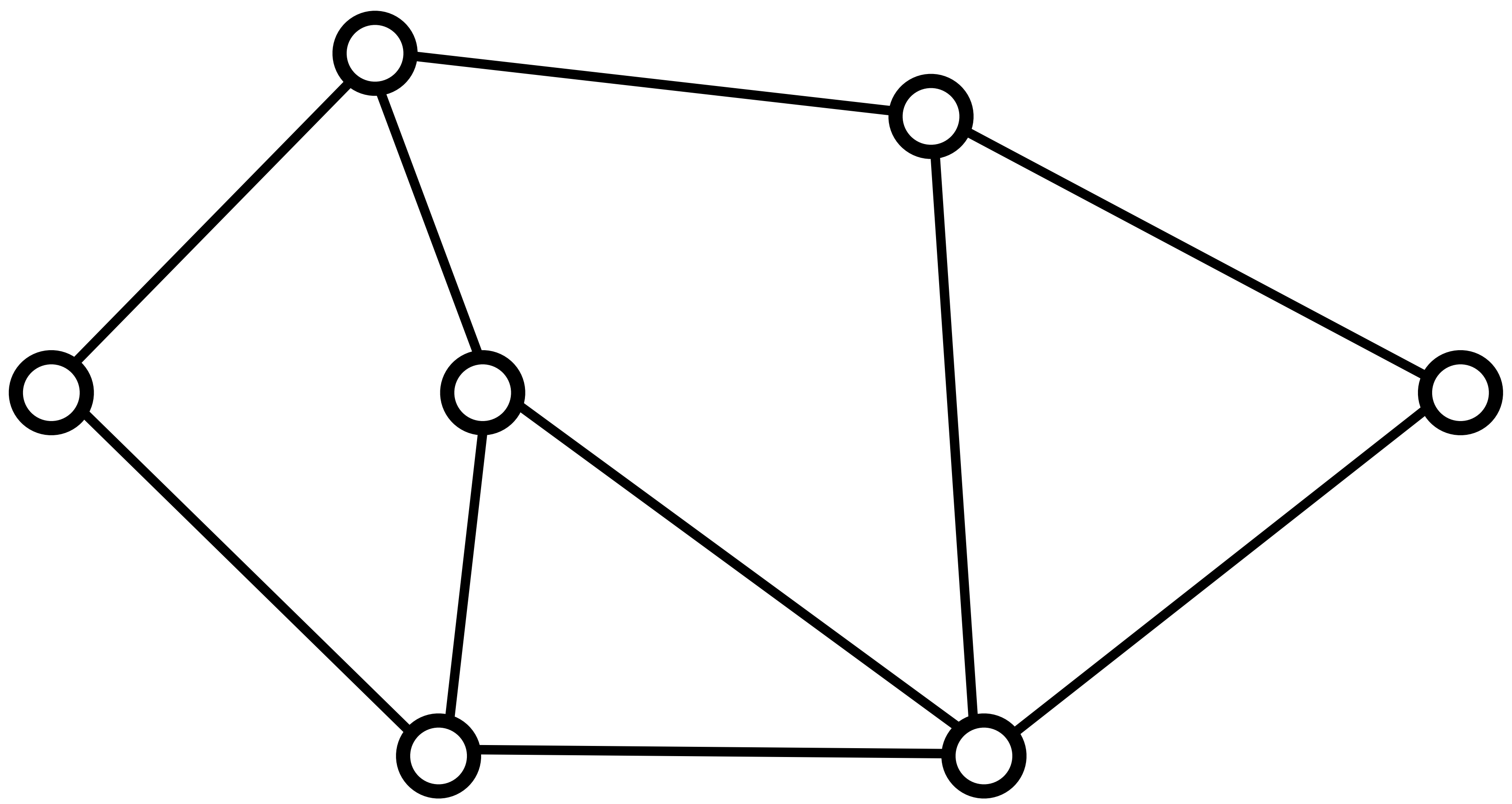
Classic Gomory-Hu Tree Algorithm
[GH, 1961]

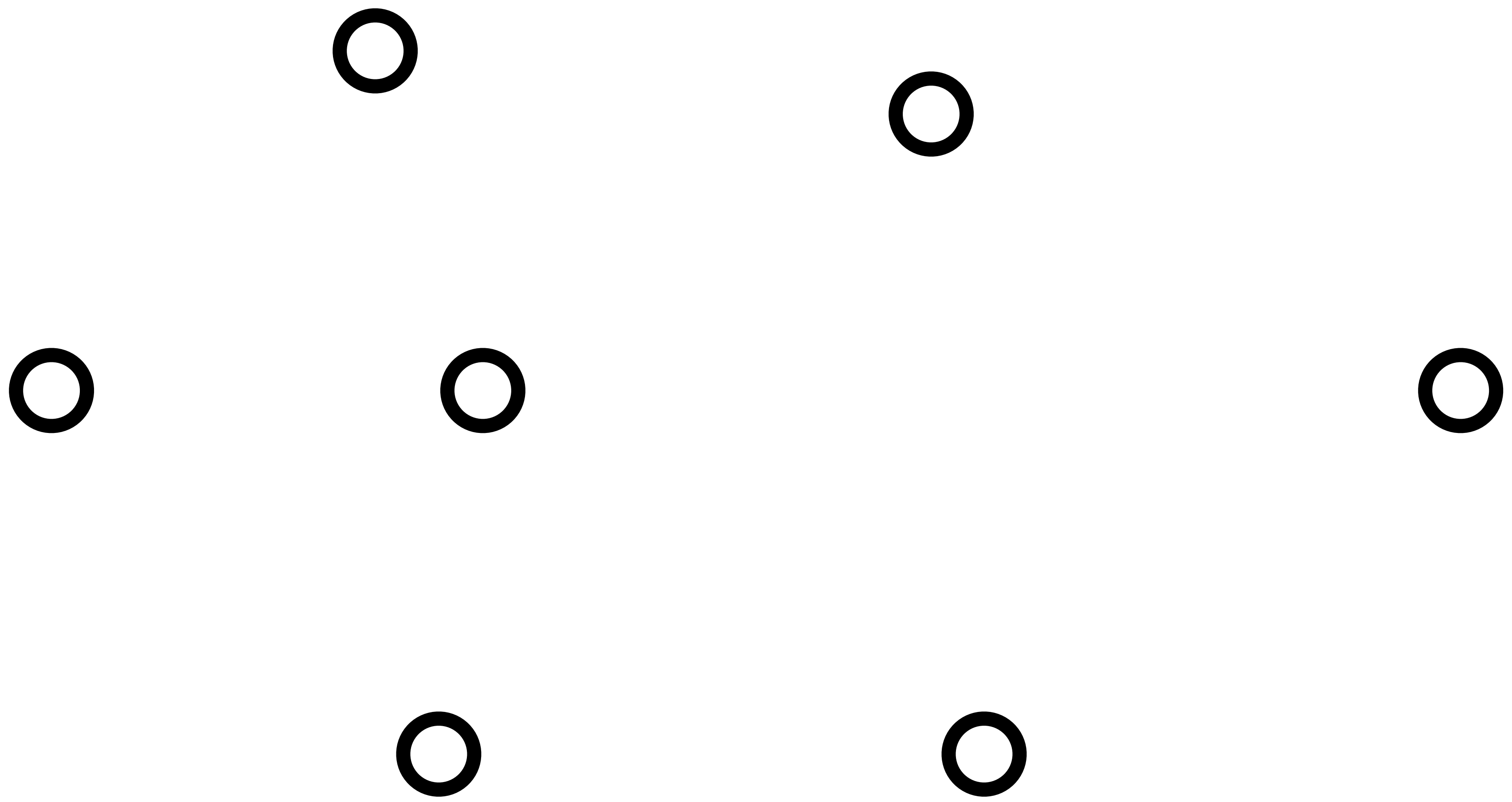
Classic Gomory-Hu Tree

Algorithm [GH'61]

1. Pick arbitrary $s, t \in V$ and compute s-t min-cut $(C, V \backslash C)$

2. Contract one side and recur on the other side



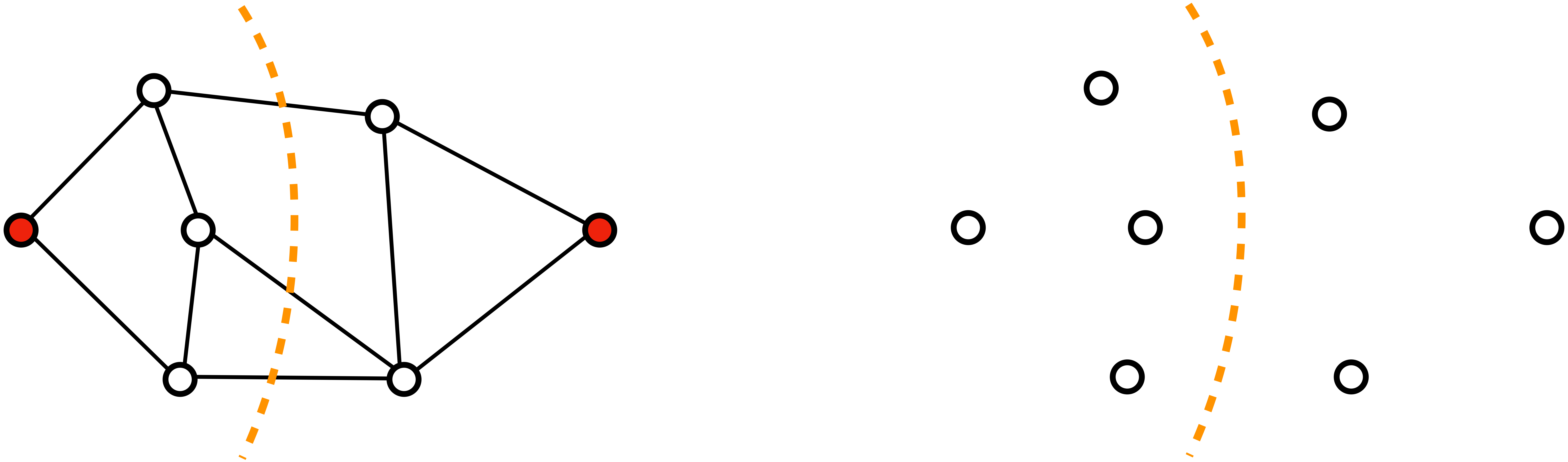


Classic Gomory-Hu Tree

Algorithm [GH'61]

1. Pick arbitrary $s, t \in V$ and compute s-t min-cut $(C, V \backslash C)$

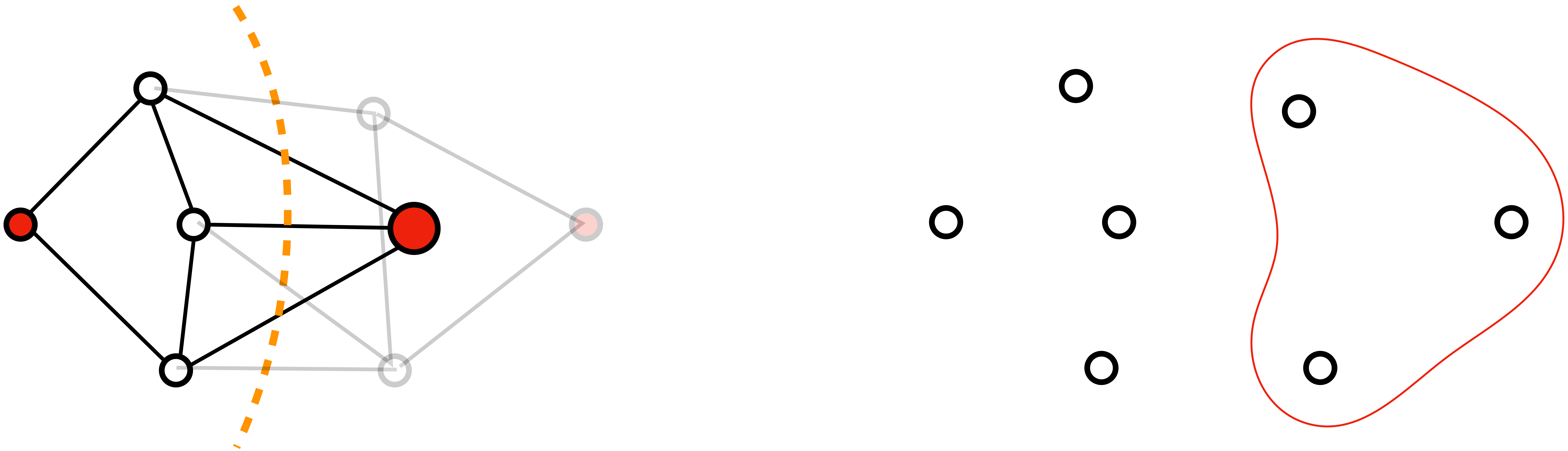
2. Contract one side and recur on the other side



Classic Gomory-Hu Tree

Algorithm [GH'61]

1. Pick arbitrary $s, t \in V$ and compute s-t min-cut $(C, V \backslash C)$
2. Contract one side and recur on the other side

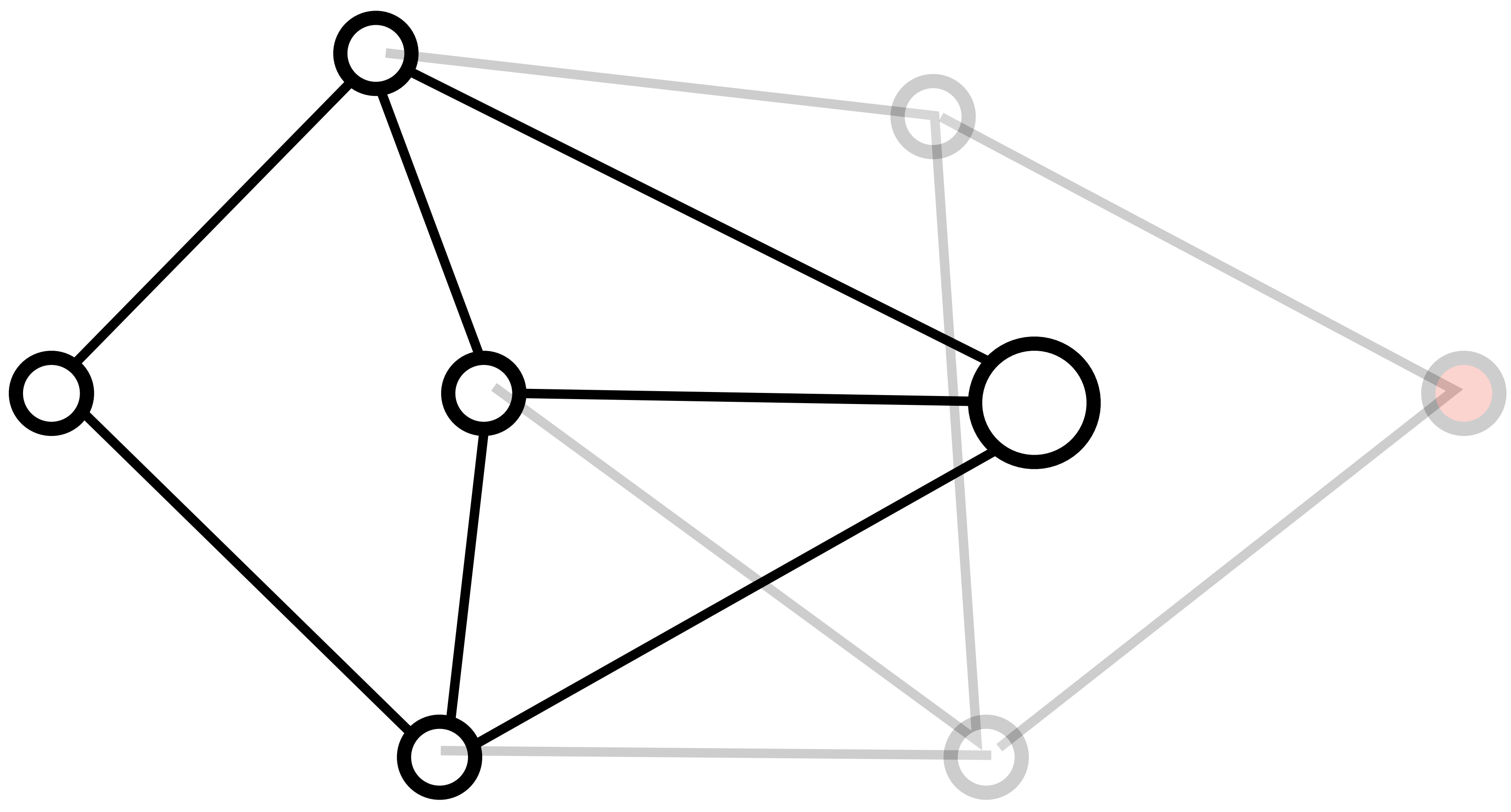


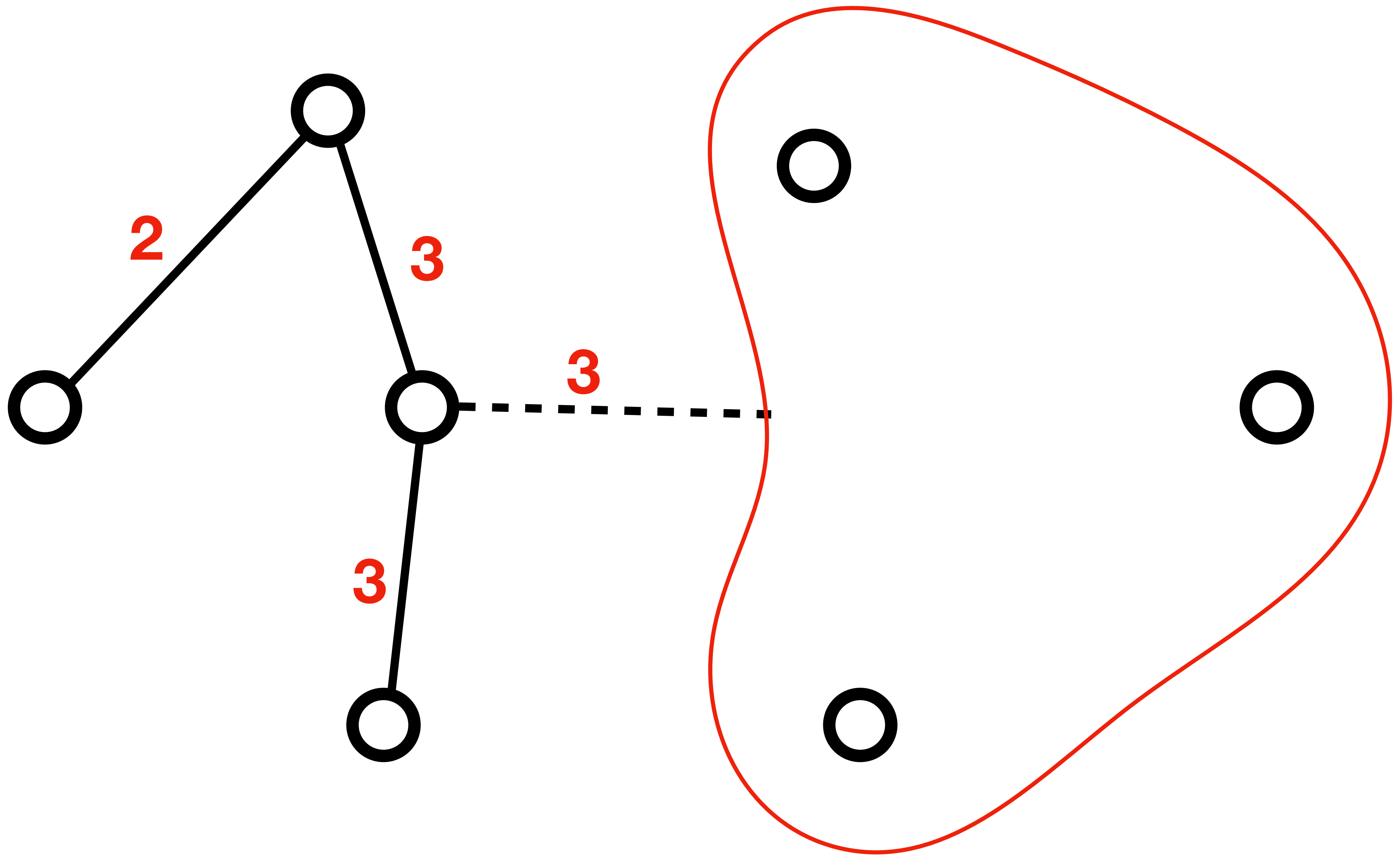
Classic Gomory-Hu Tree

Algorithm [GH'61]

1. Pick arbitrary $s, t \in V$ and compute s-t min-cut $(C, V \backslash C)$

2. Contract one side and recur on the other side

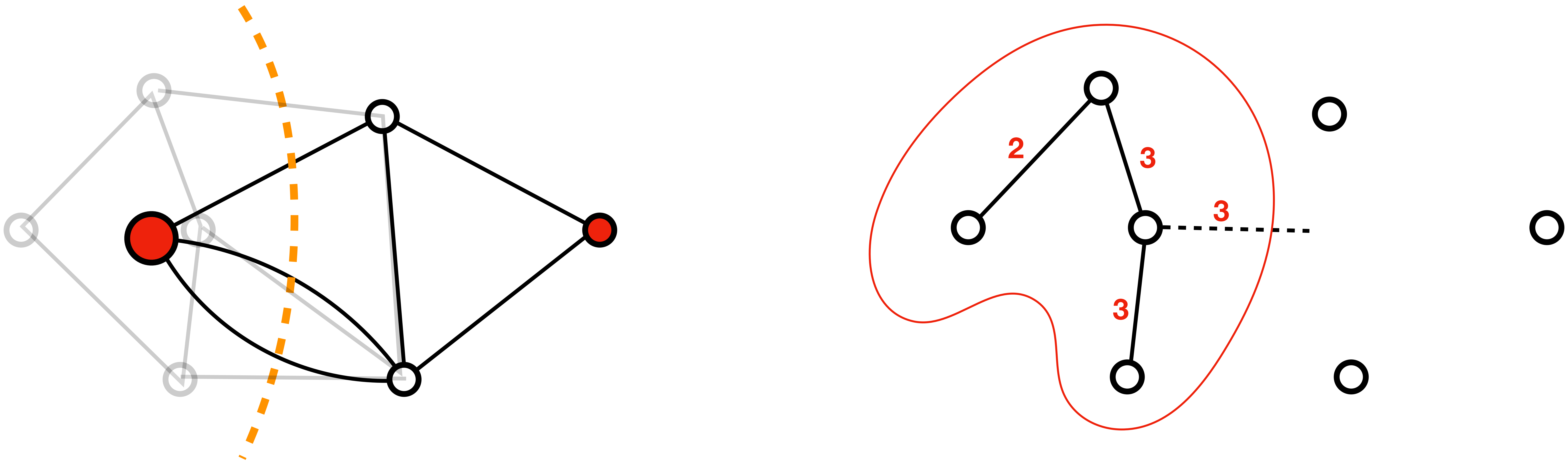




Classic Gomory-Hu Tree

<u>Algorithm [GH'61]</u>

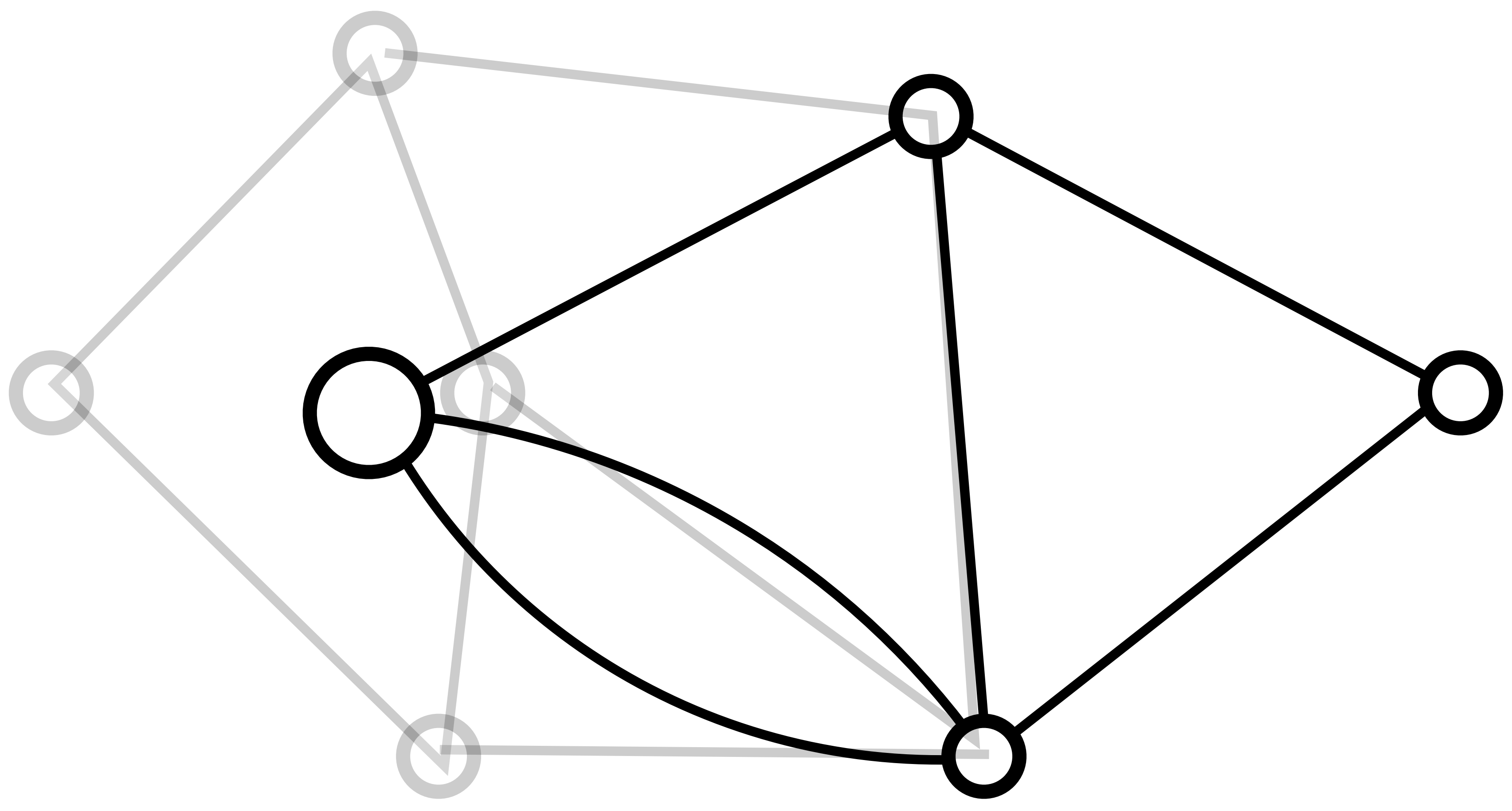
1. Pick arbitrary $s, t \in V$ and compute s-t min-cut $(C, V \backslash C)$
2. Contract one side and recur on the other side

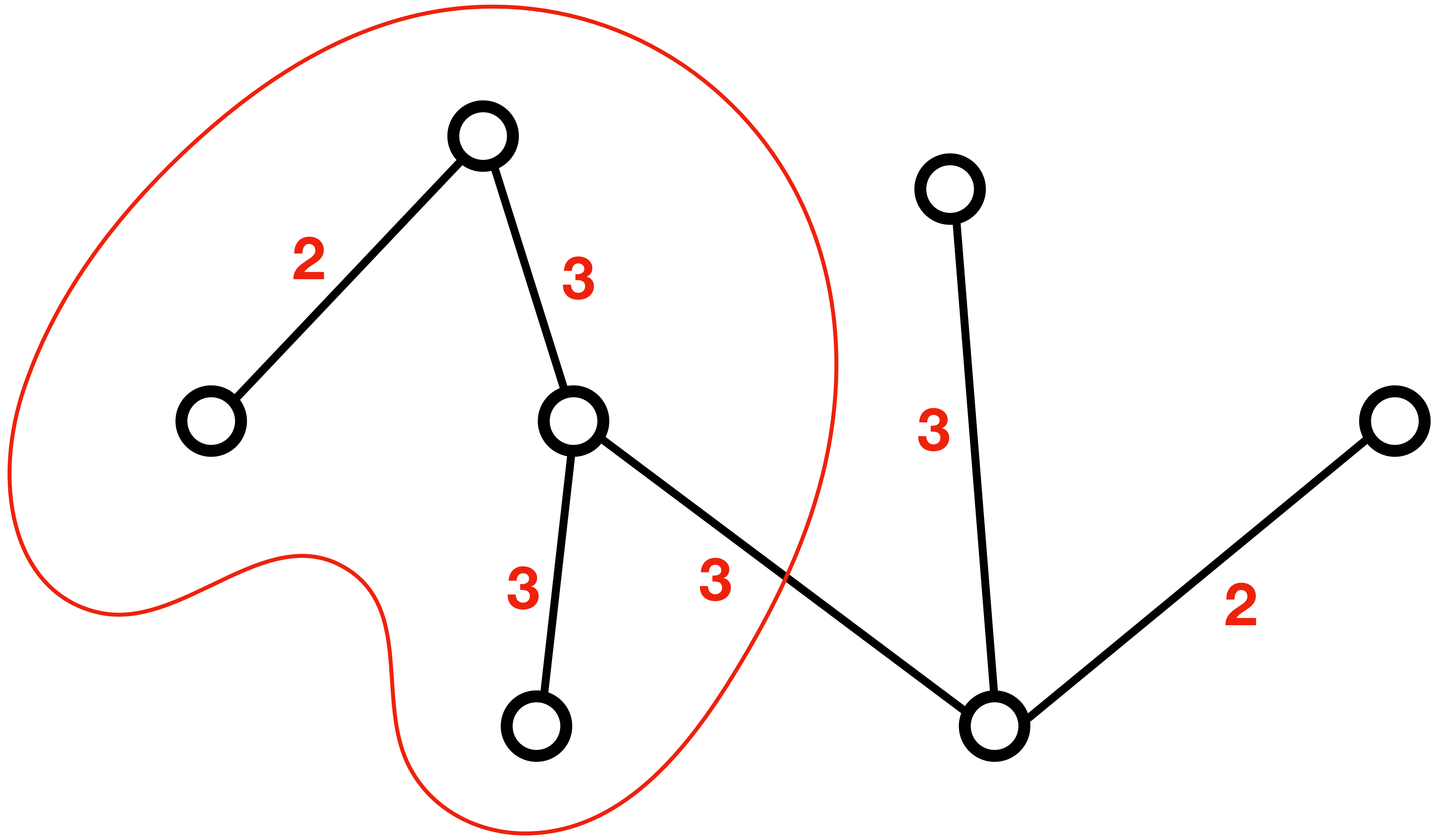


Classic Gomory-Hu Tree

Algorithm [GH'61]

1. Pick arbitrary $s, t \in V$ and compute s-t min-cut $(C, V \backslash C)$
2. Contract one side and recur on the other side

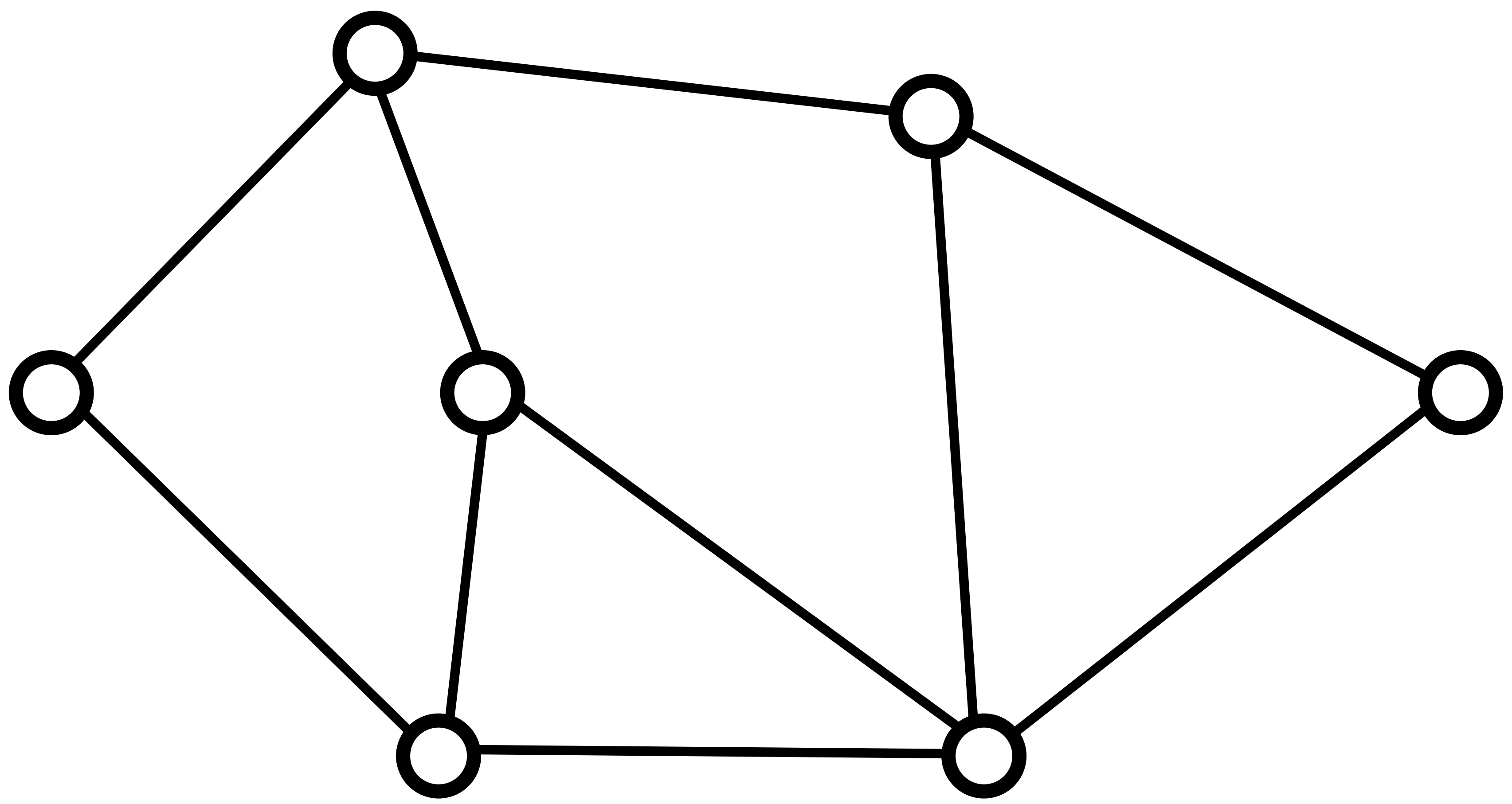


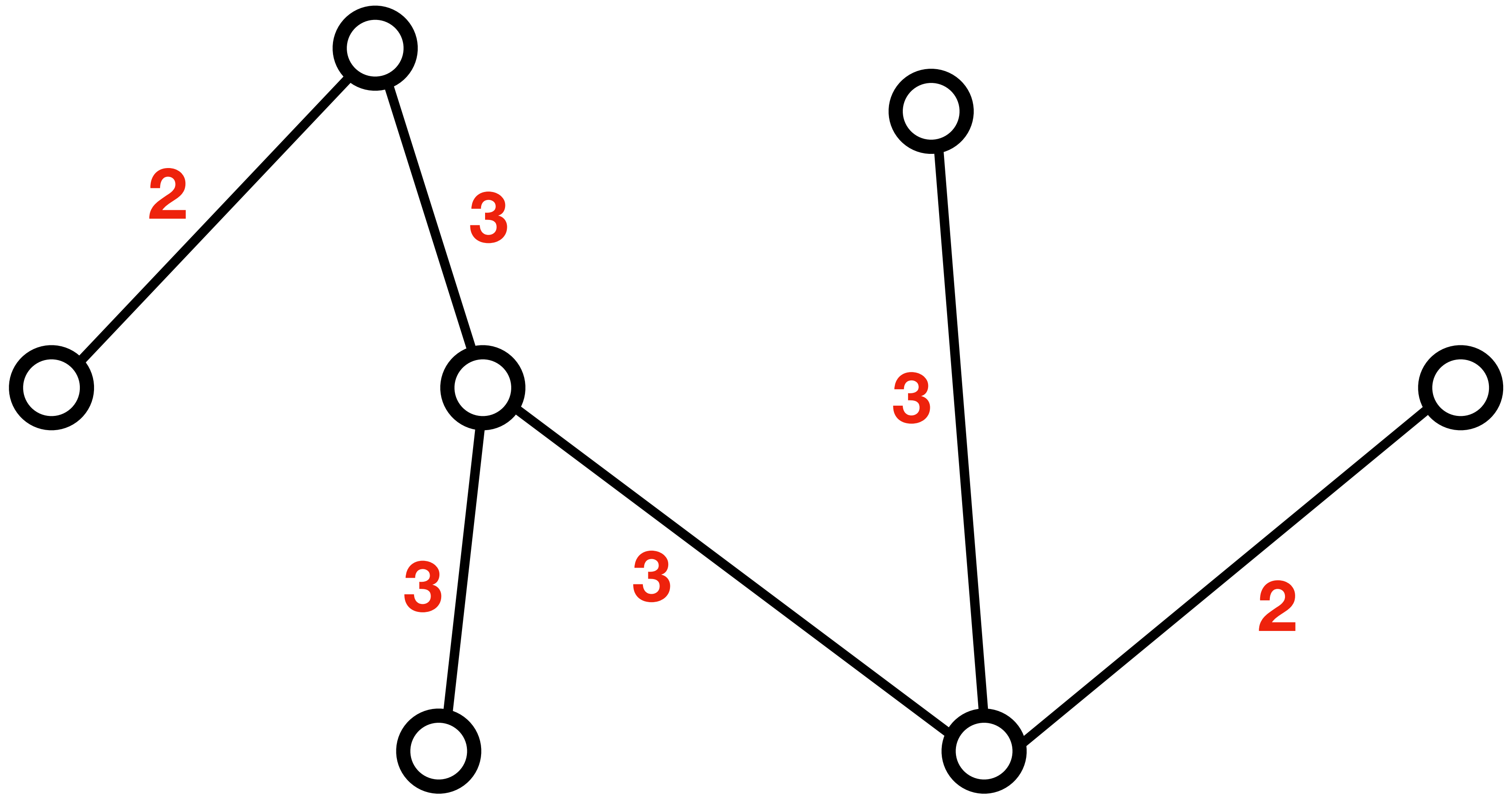


Classic Gomory-Hu Tree

Algorithm [GH'61]

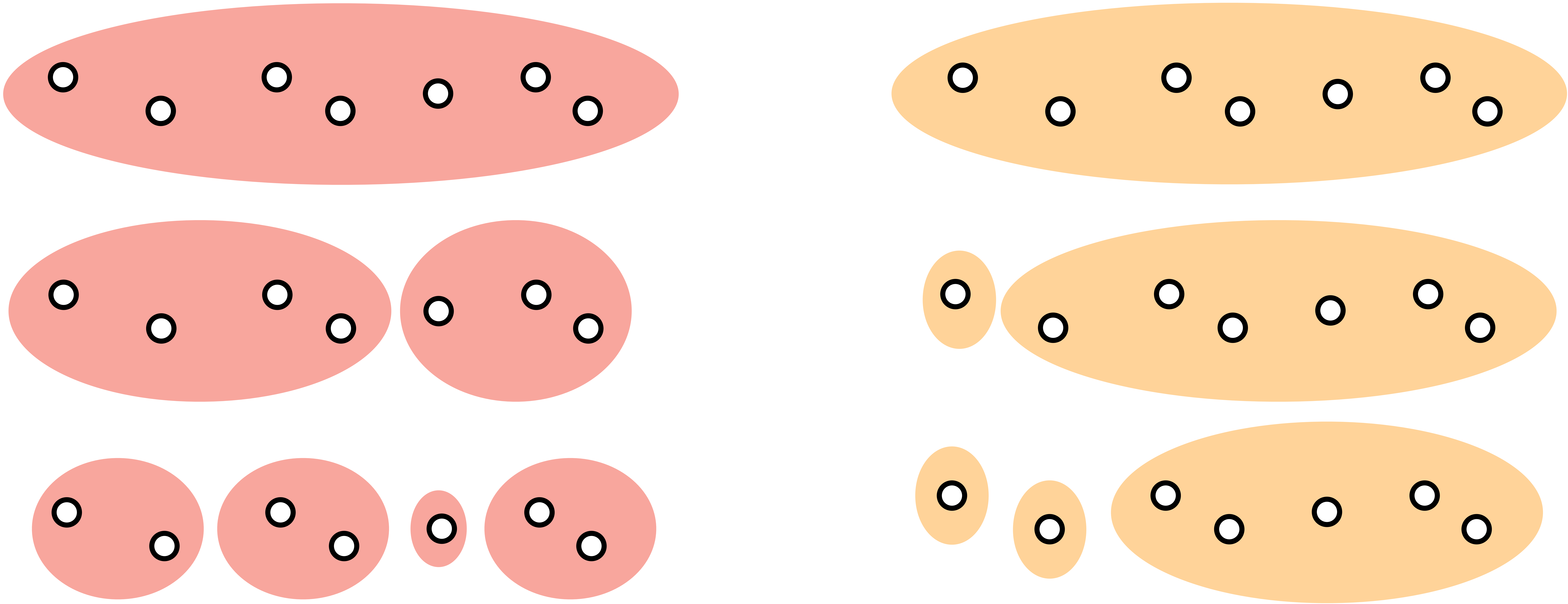
1. Pick arbitrary $s, t \in V$ and compute s-t min-cut $(C, V \backslash C)$
2. Contract one side and recur on the other side





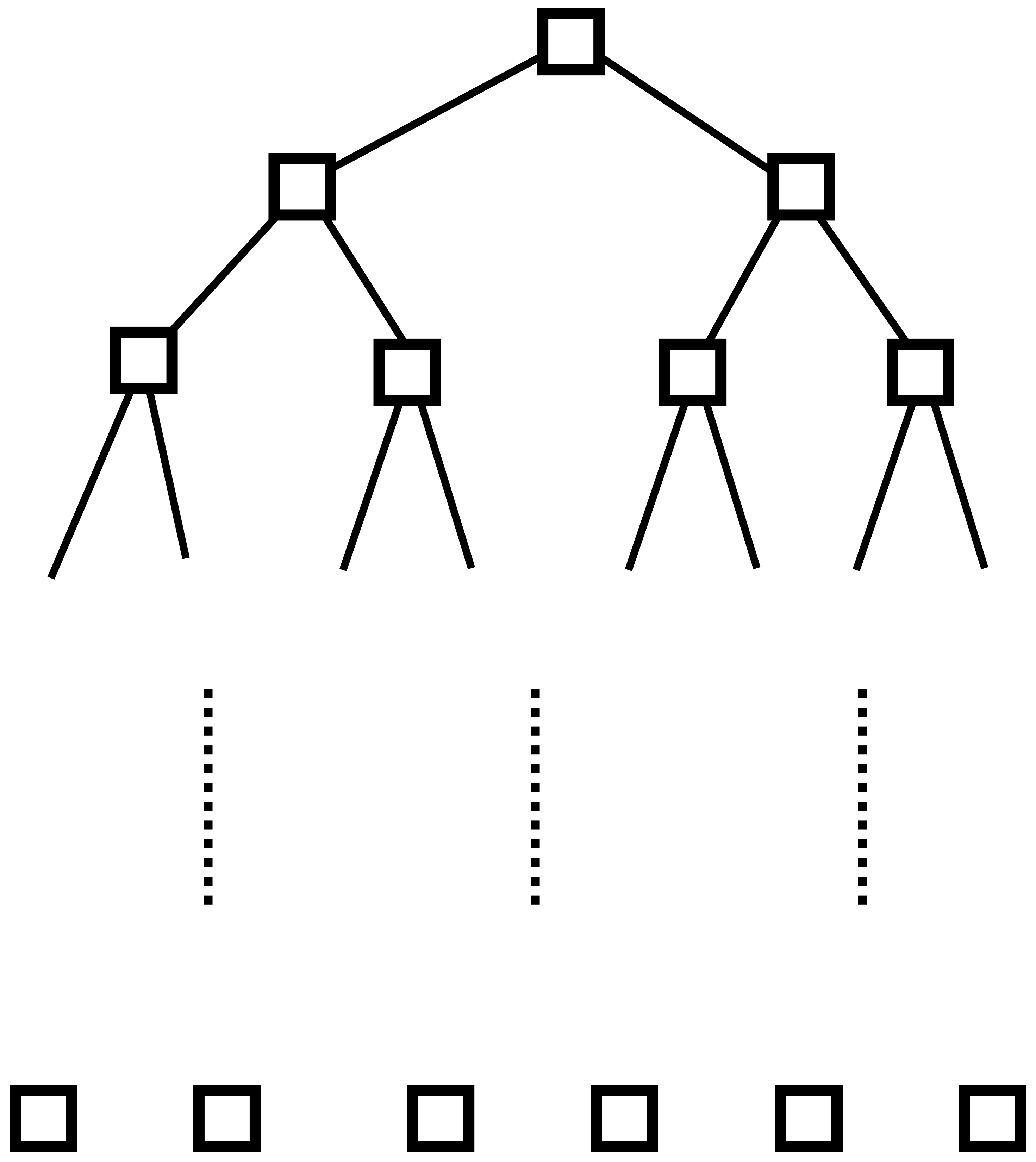
Classic Gomory-Hu Tree

Possible recursions of [GH'61]

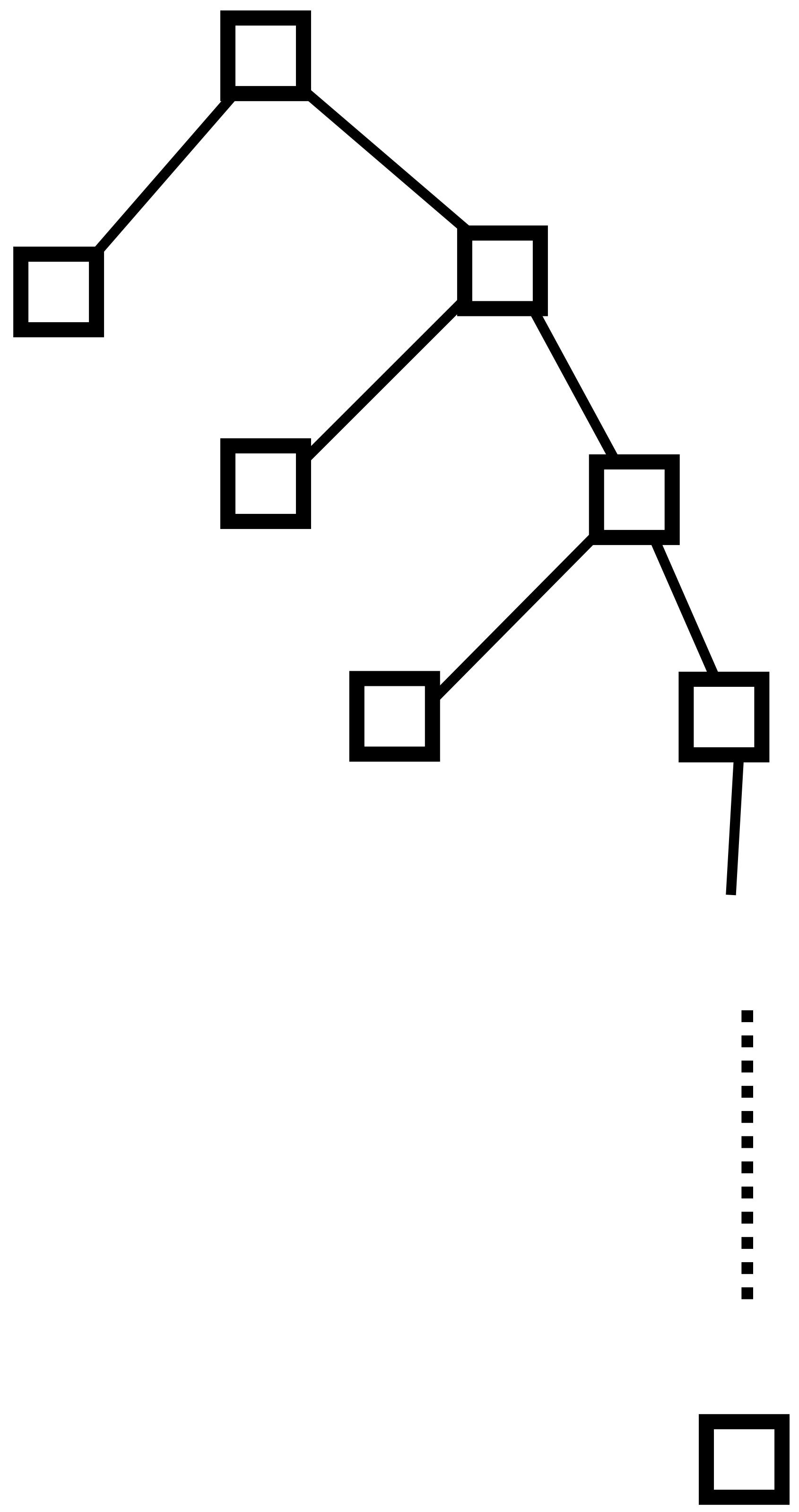


Classic Gomory-Hu Tree

Possible recursion trees of [GH’61]



balanced, runtime = $MF(m \log n)$

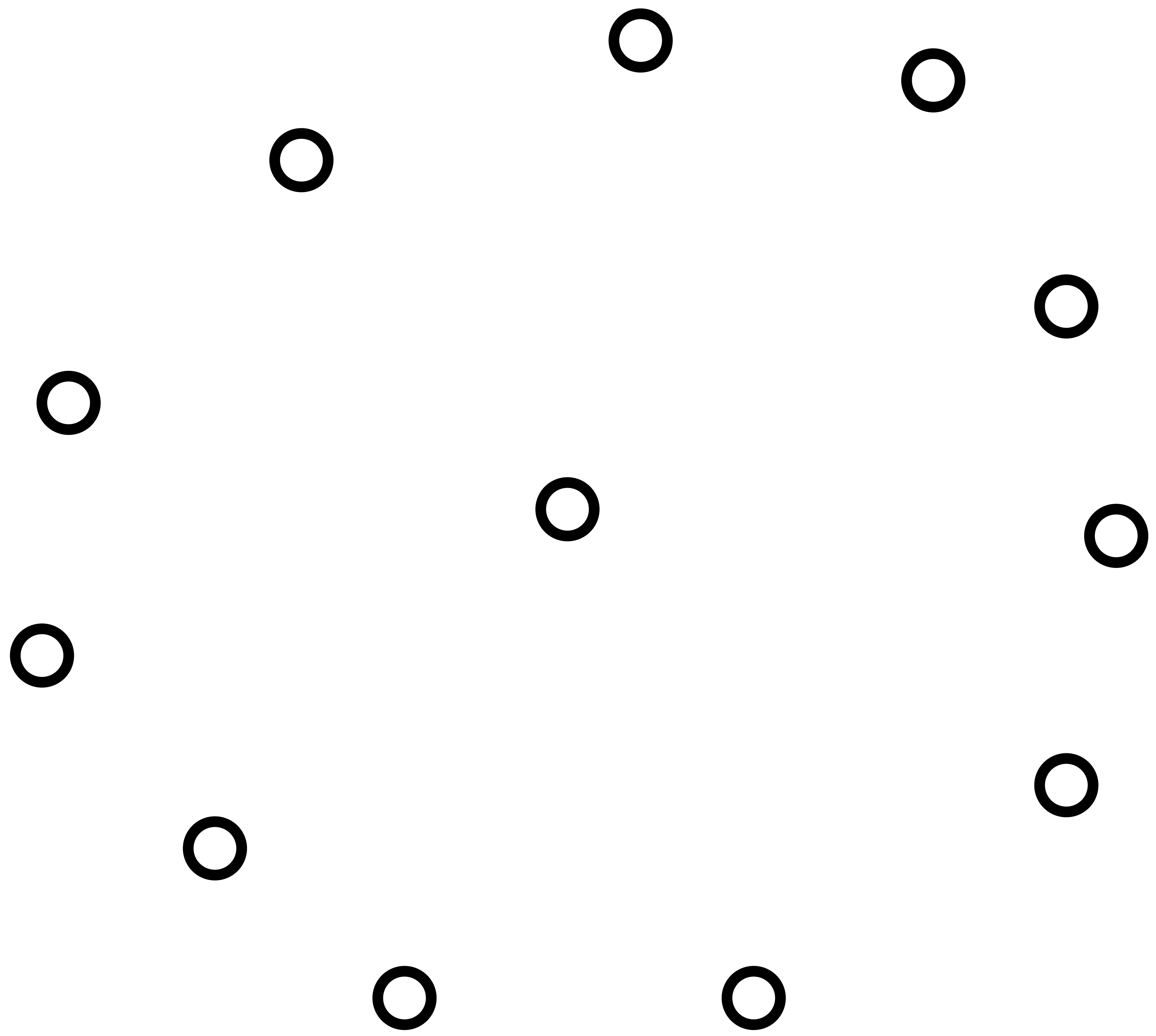


unbalanced, runtime = $MF(mn)$

Subcubic Gomory-Hu Tree
[AKT, 2021]

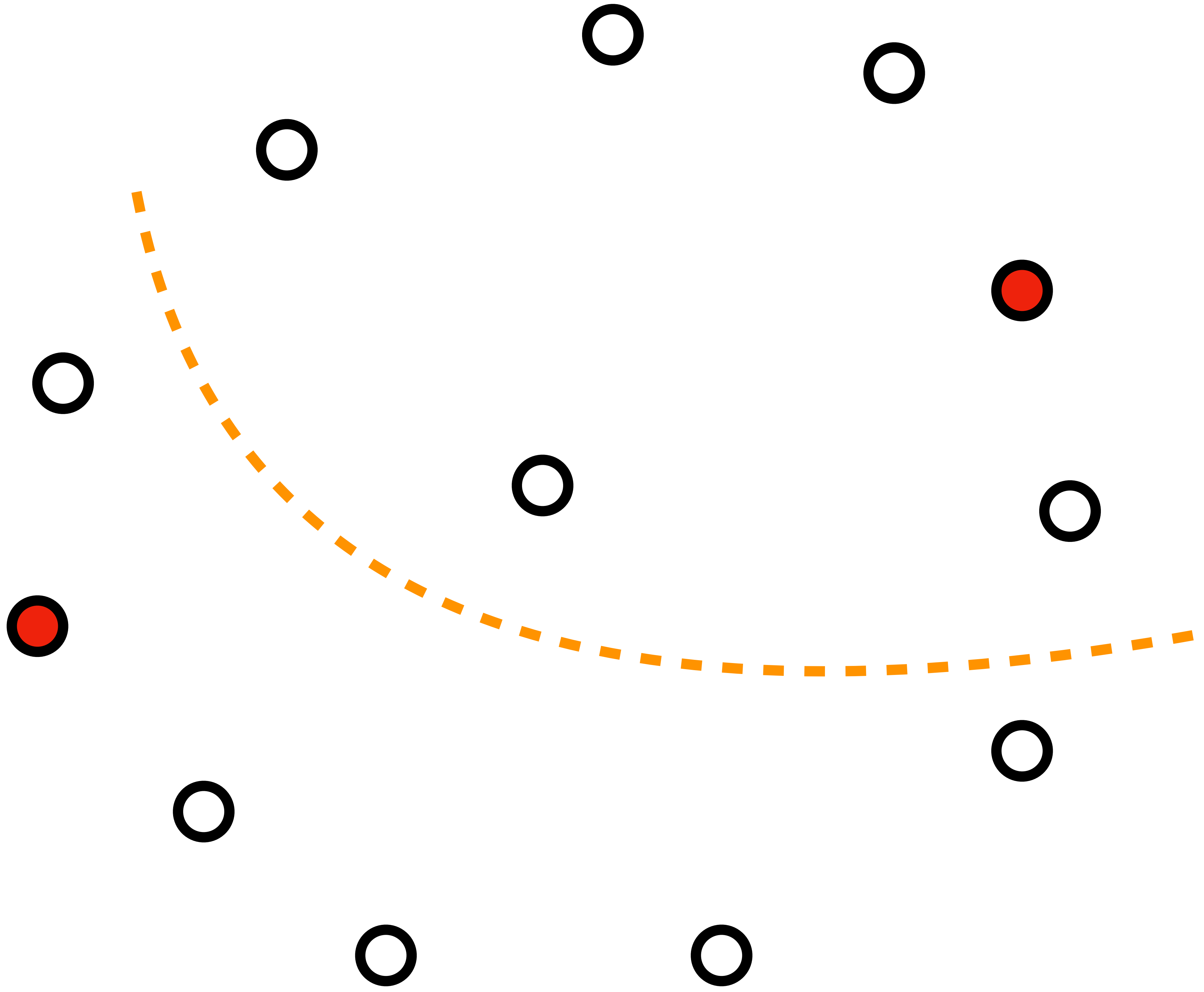
Subcubic Gomory-Hu [AKT'21]

graph **recursion tree**



Subcubic Gomory-Hu [AKT'21]

graph

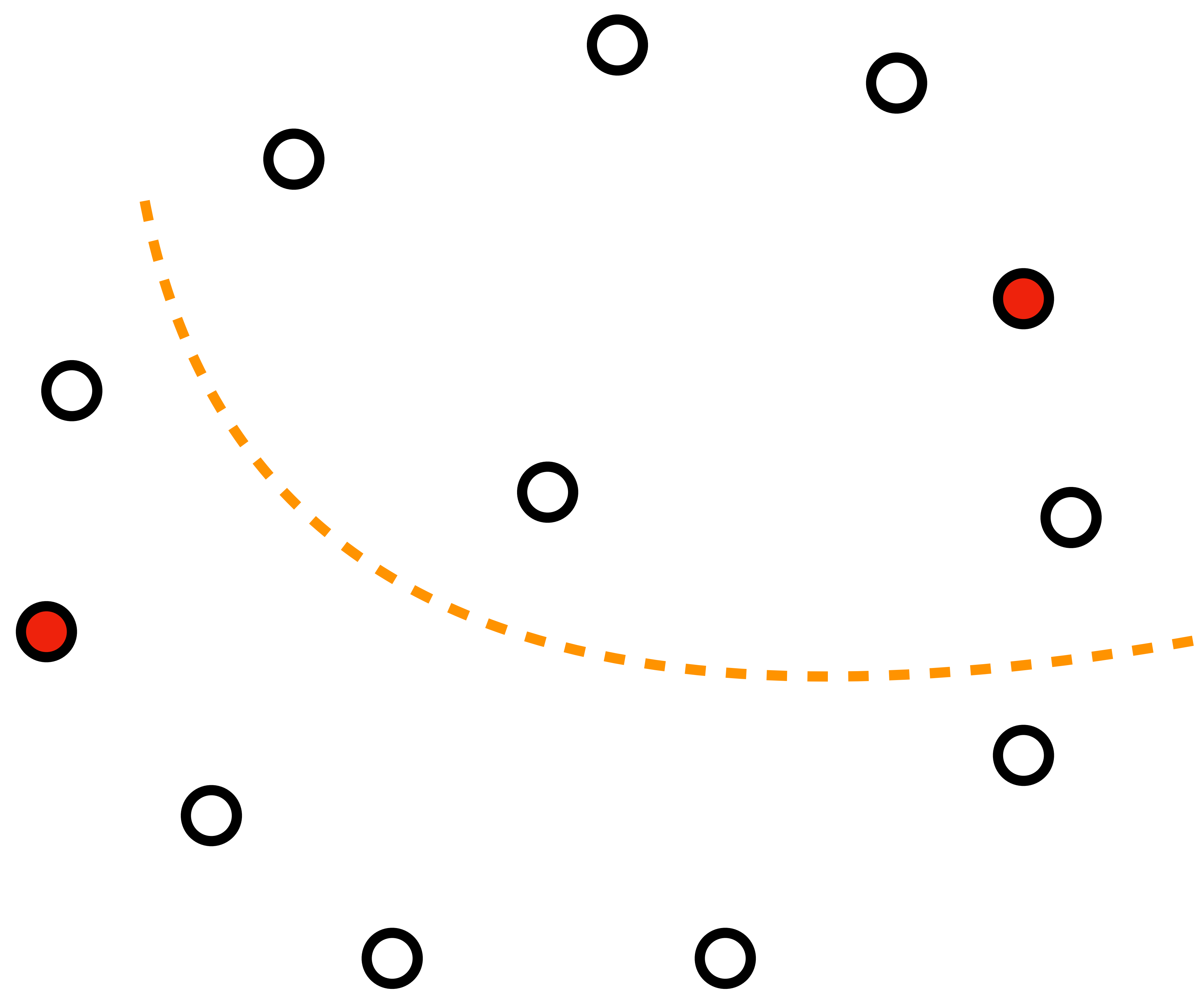


recursion tree

Ideally, each side contains half of the vertices

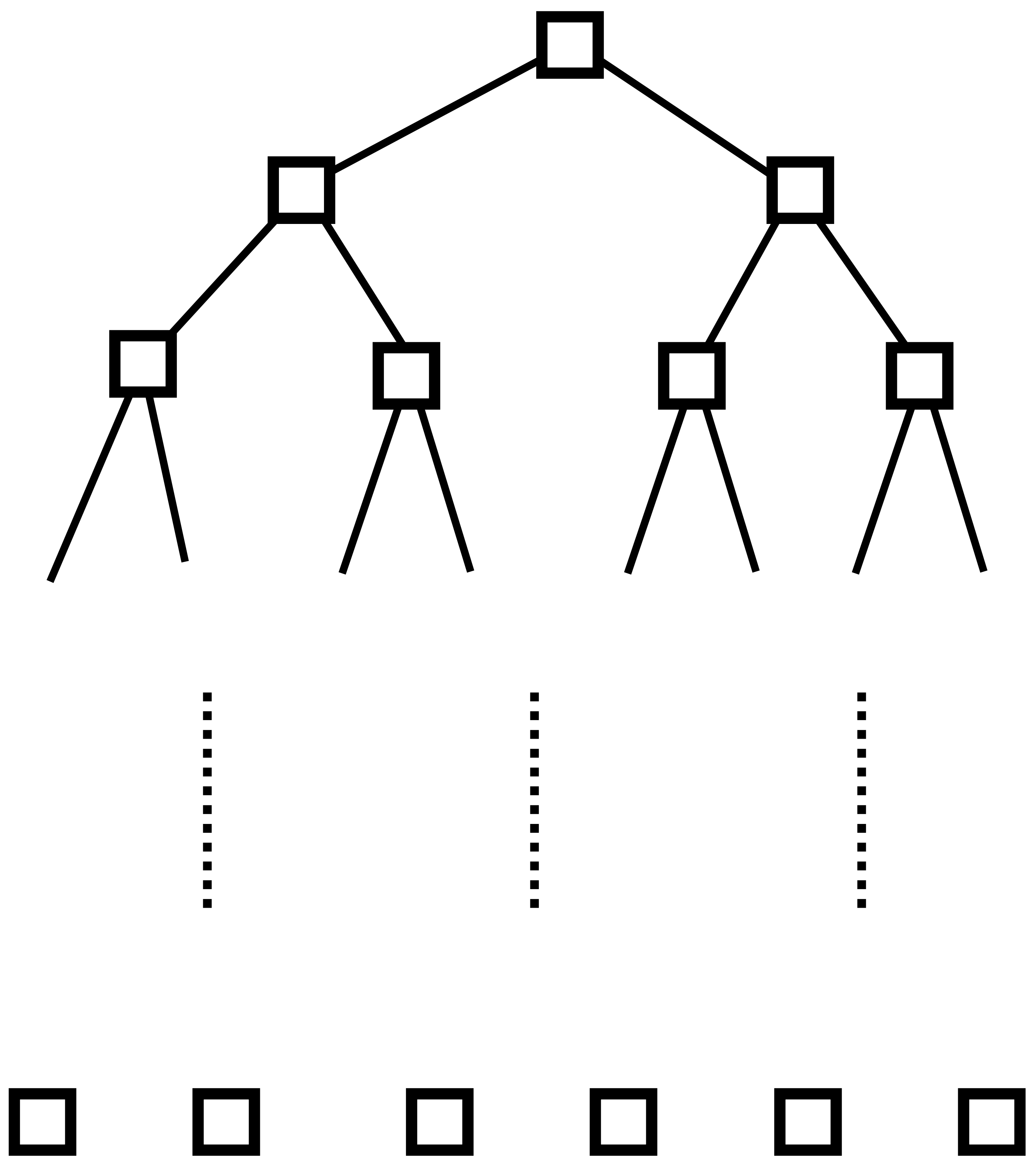
Subcubic Gomory-Hu [AKT'21]

graph



Ideally, each side contains half of the vertices

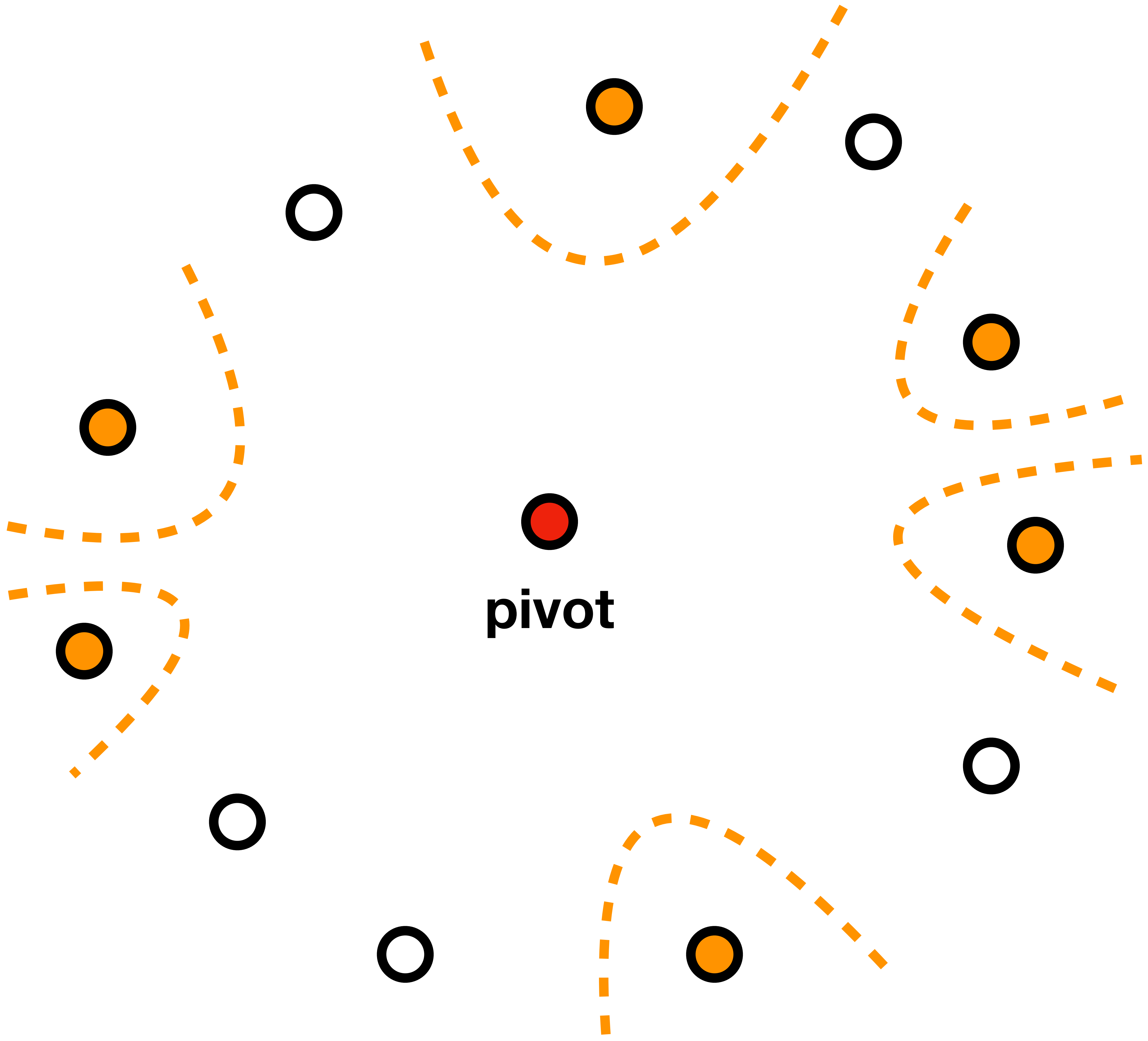
recursion tree



balanced, runtime = $MF(m \log n)$

Subcubic Gomory-Hu [AKT'21]

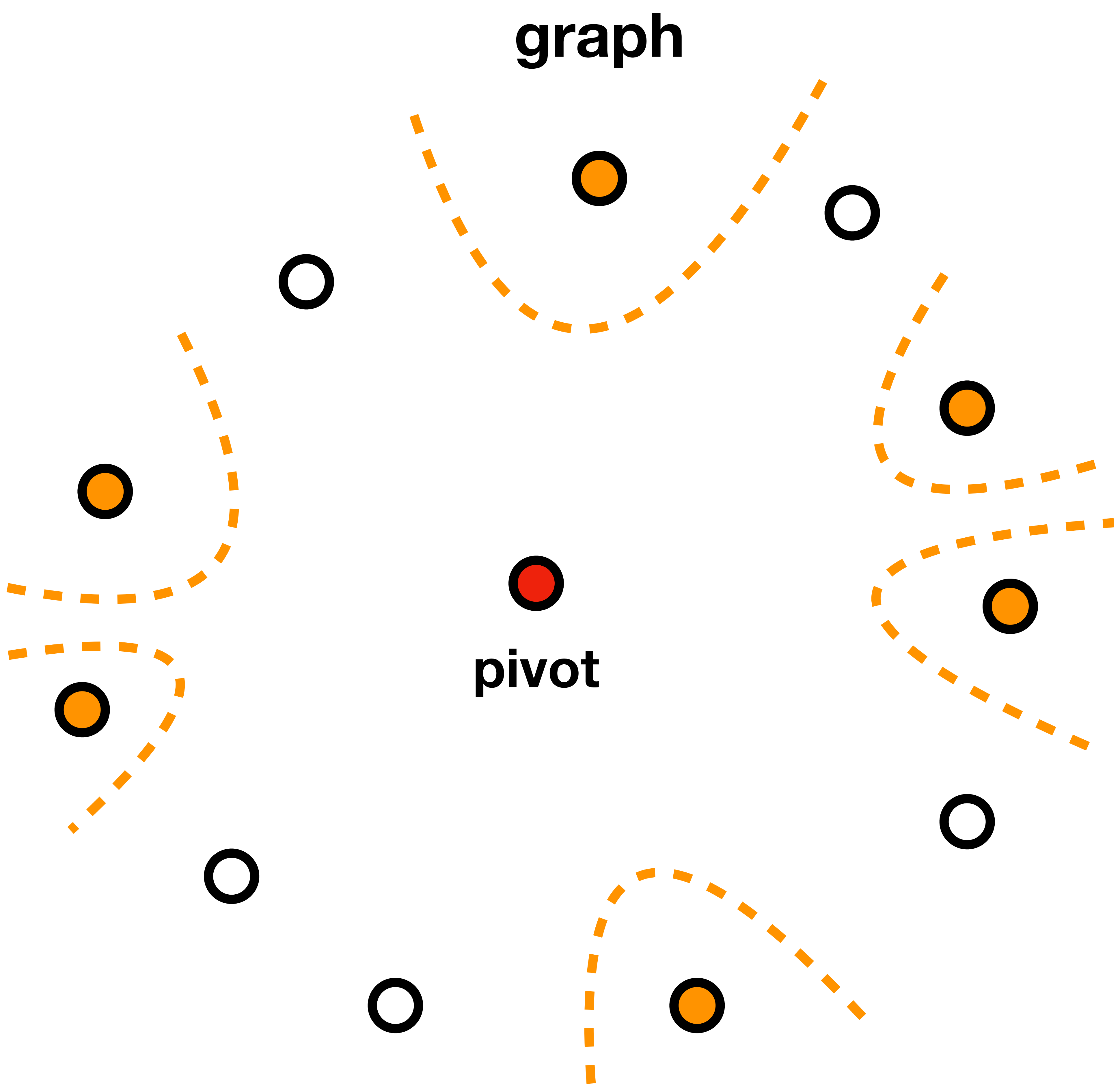
graph

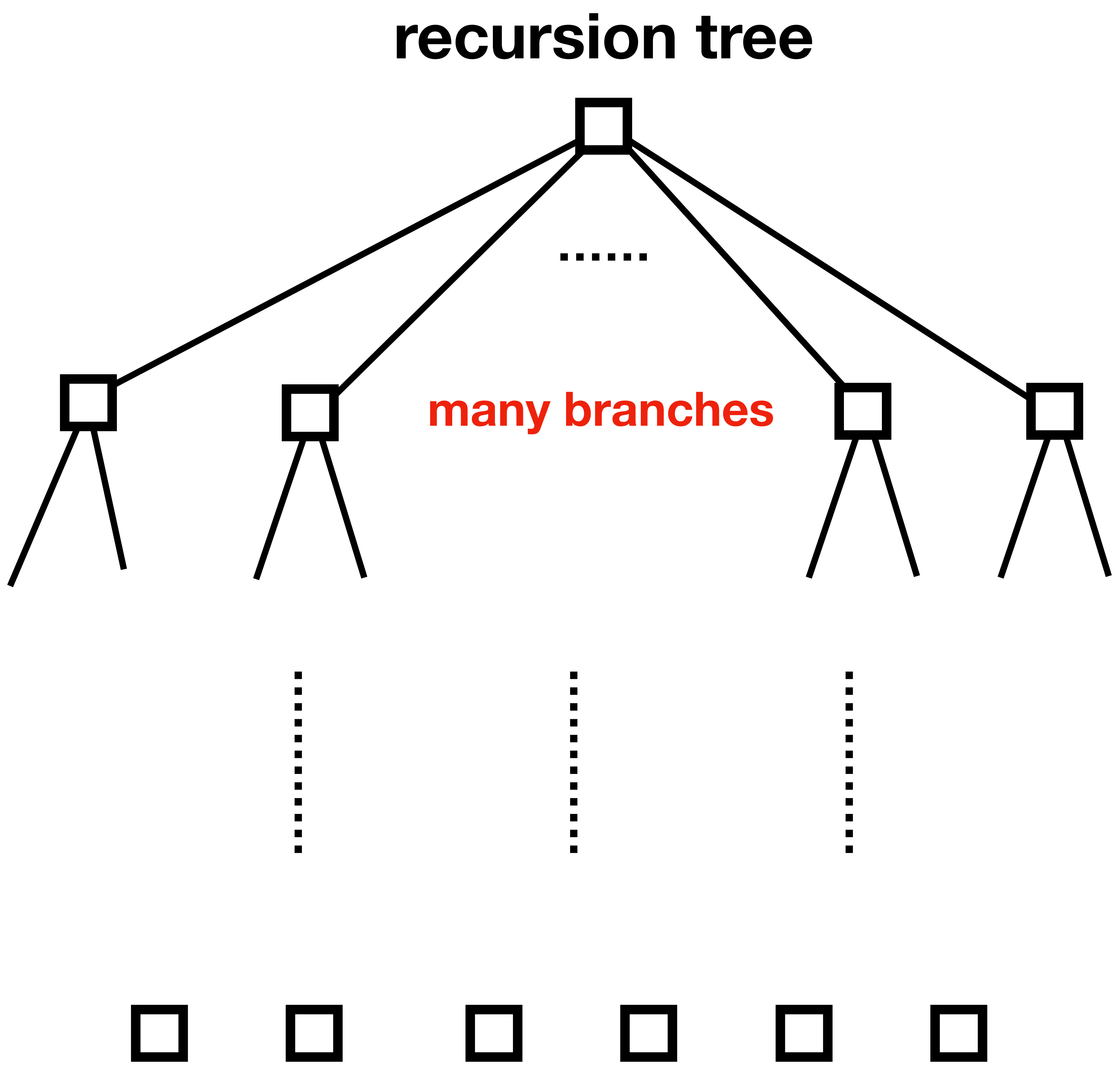


recursion tree

In reality, compute single-source min-cuts

Subcubic Gomory-Hu [AKT'21]

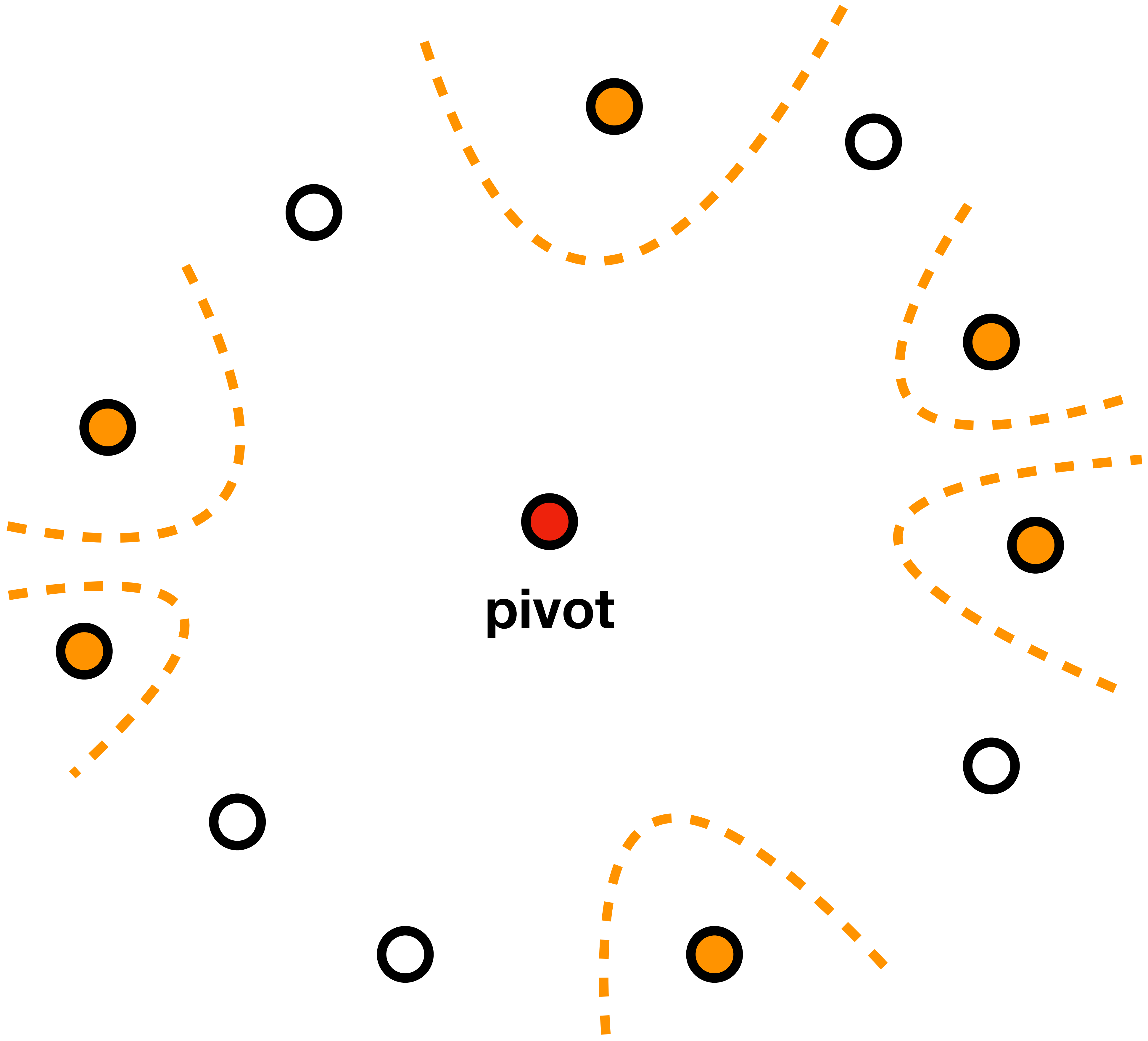




In reality, compute single-source min-cuts

balanced, runtime = $MF(m \log n)$

Single Source Min-Cuts [AKT'21]



In reality, compute single-source min-cuts

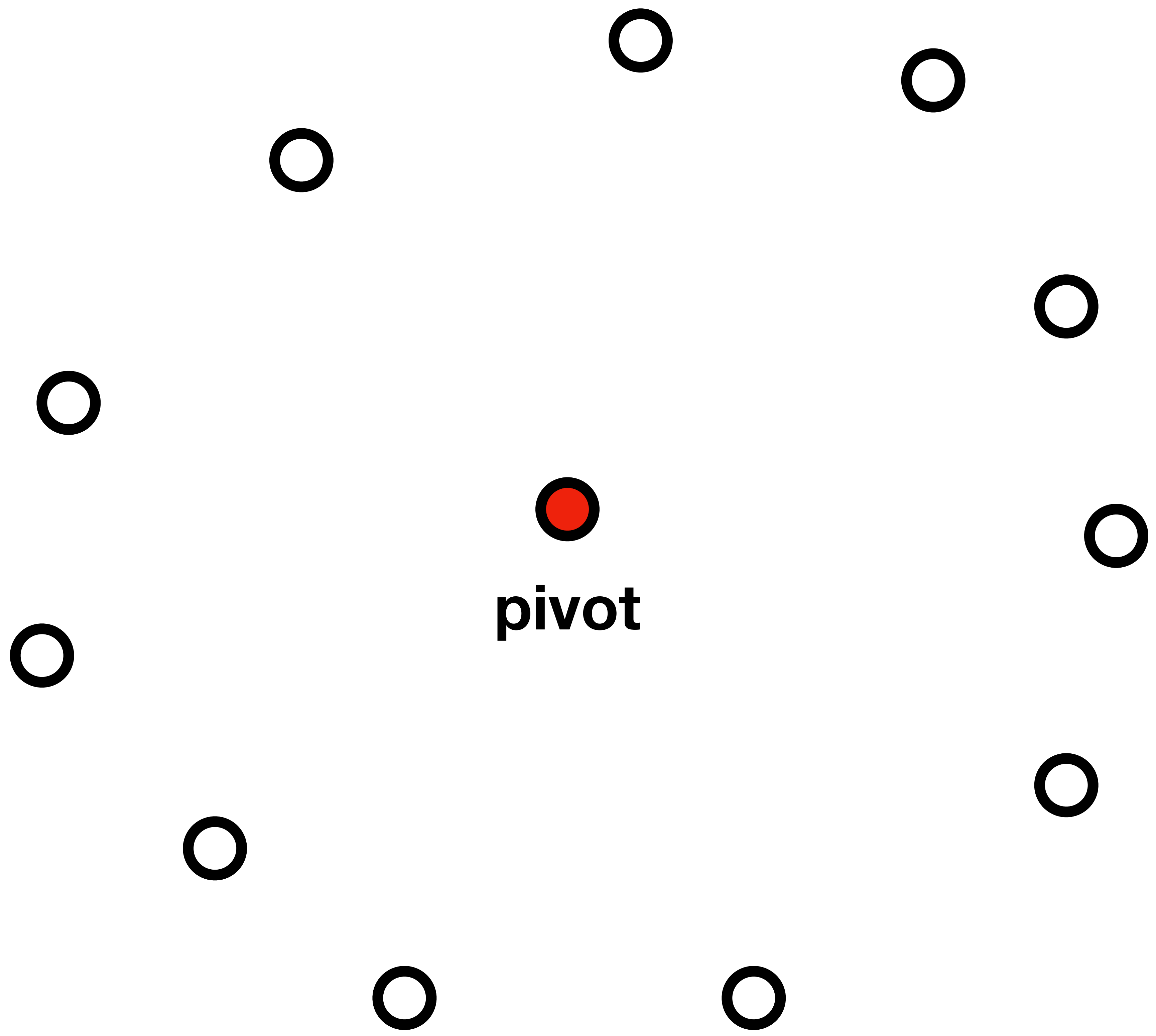
Desired properties:

1. Each part contains $\leq 0.5n$ vertices
2. At least $0.1n$ vertices are cut off

Consequence:

- The recursion tree has $O(\log n)$ depth

Using Expander Decomposition [AKT'21]

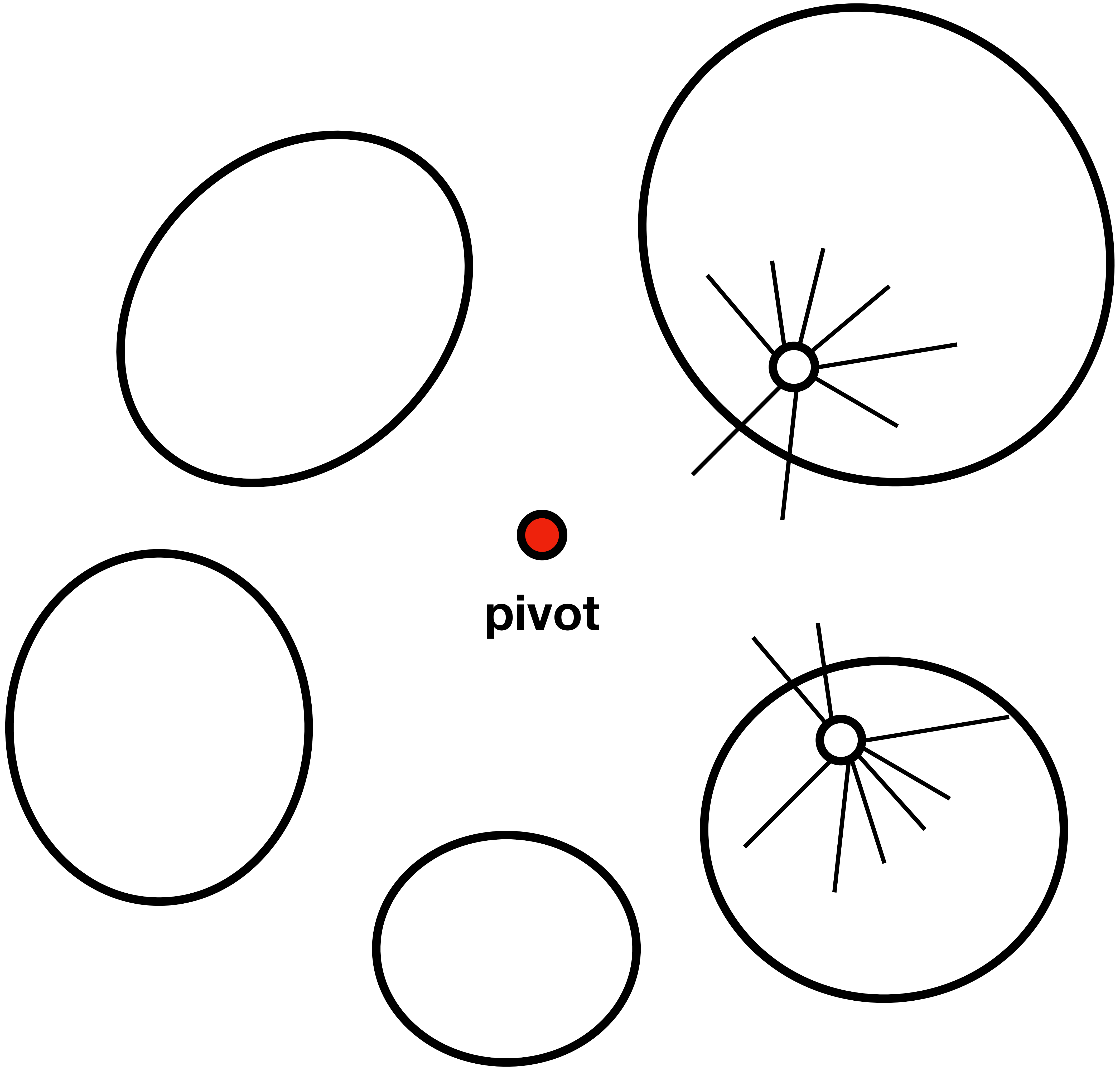


Expander decomposition [SW'19]:

Partition $V = C_1 \cup C_2 \cup \cdots \cup C_k$ s.t.

1. $\partial(C_i) = \tilde{O}(\phi \mathrm{vol}(C_i)), \forall i$
2. Each subgraph $G\{C_i\}$ is a ϕ-expander

Using Expander Decomposition [AKT'21]

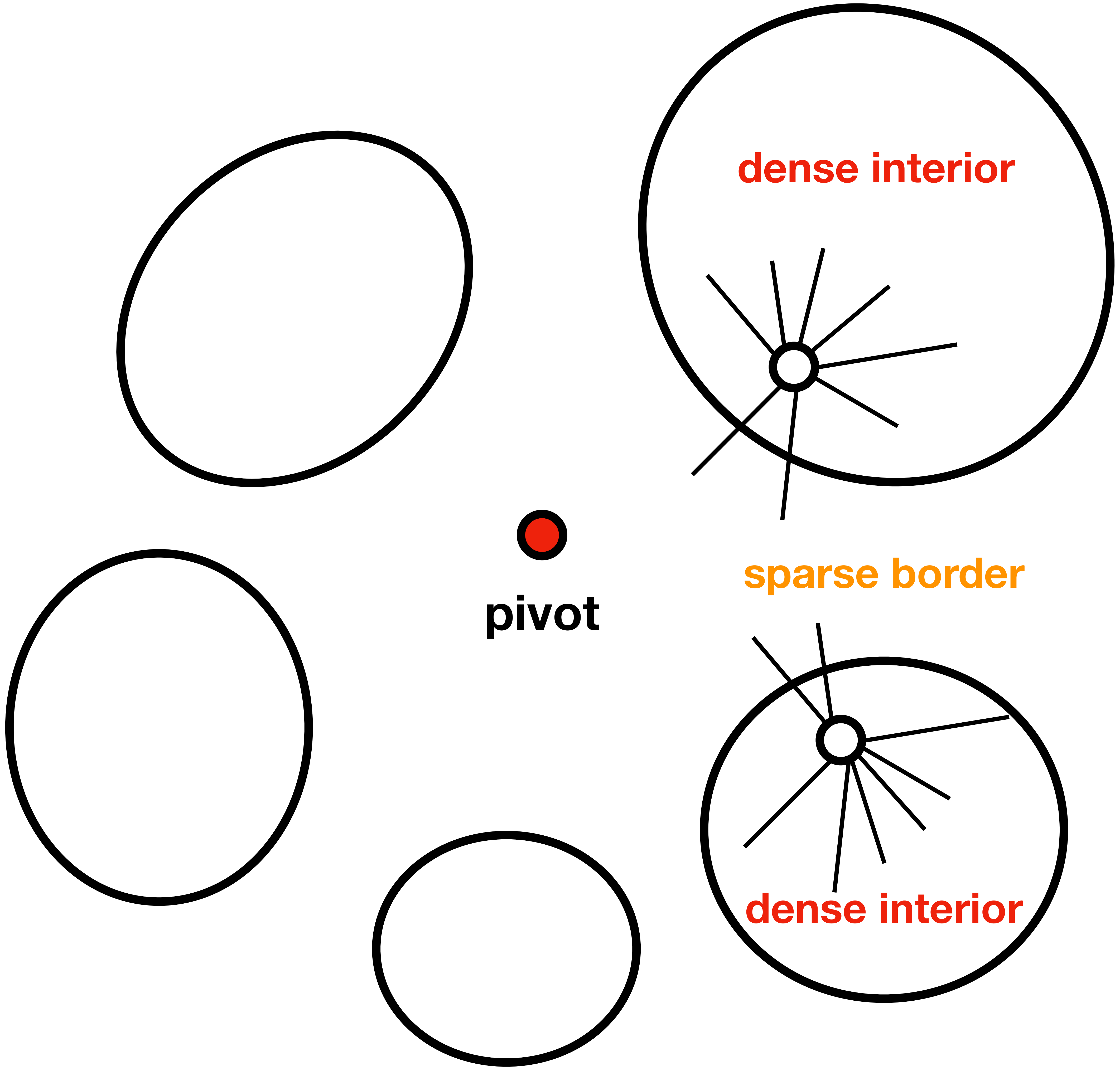


Expander decomposition [SW'19]:

Partition $V = C_1 \cup C_2 \cup \cdots \cup C_k$ s.t.

1. $\partial(C_i) = \tilde{O}(\phi \mathrm{vol}(C_i)), \forall i$
2. Each subgraph $G\{C_i\}$ is a ϕ-expander

Using Expander Decomposition [AKT'21]



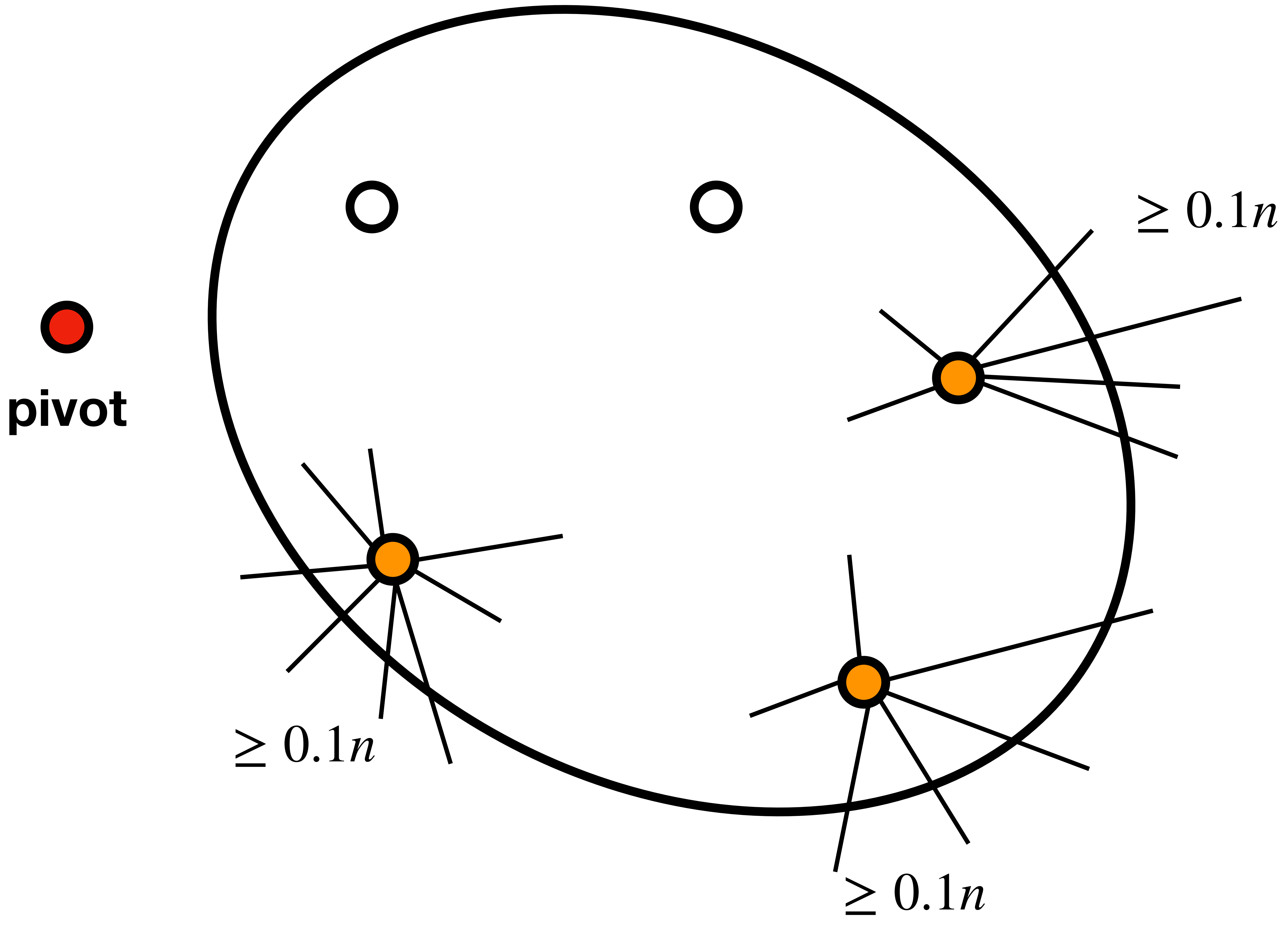
Expander decomposition [SW'19]:

Partition $V = C_1 \cup C_2 \cup \cdots \cup C_k$ s.t.

1. $\partial(C_i) = \tilde{O}(\phi \mathrm{vol}(C_i)), \forall i$
2. Each subgraph $G\{C_i\}$ is a ϕ-expander

Runtime Bottlenecks [AKT'21]

Simplifying assumption: most vertex degrees are at least $0.2n$

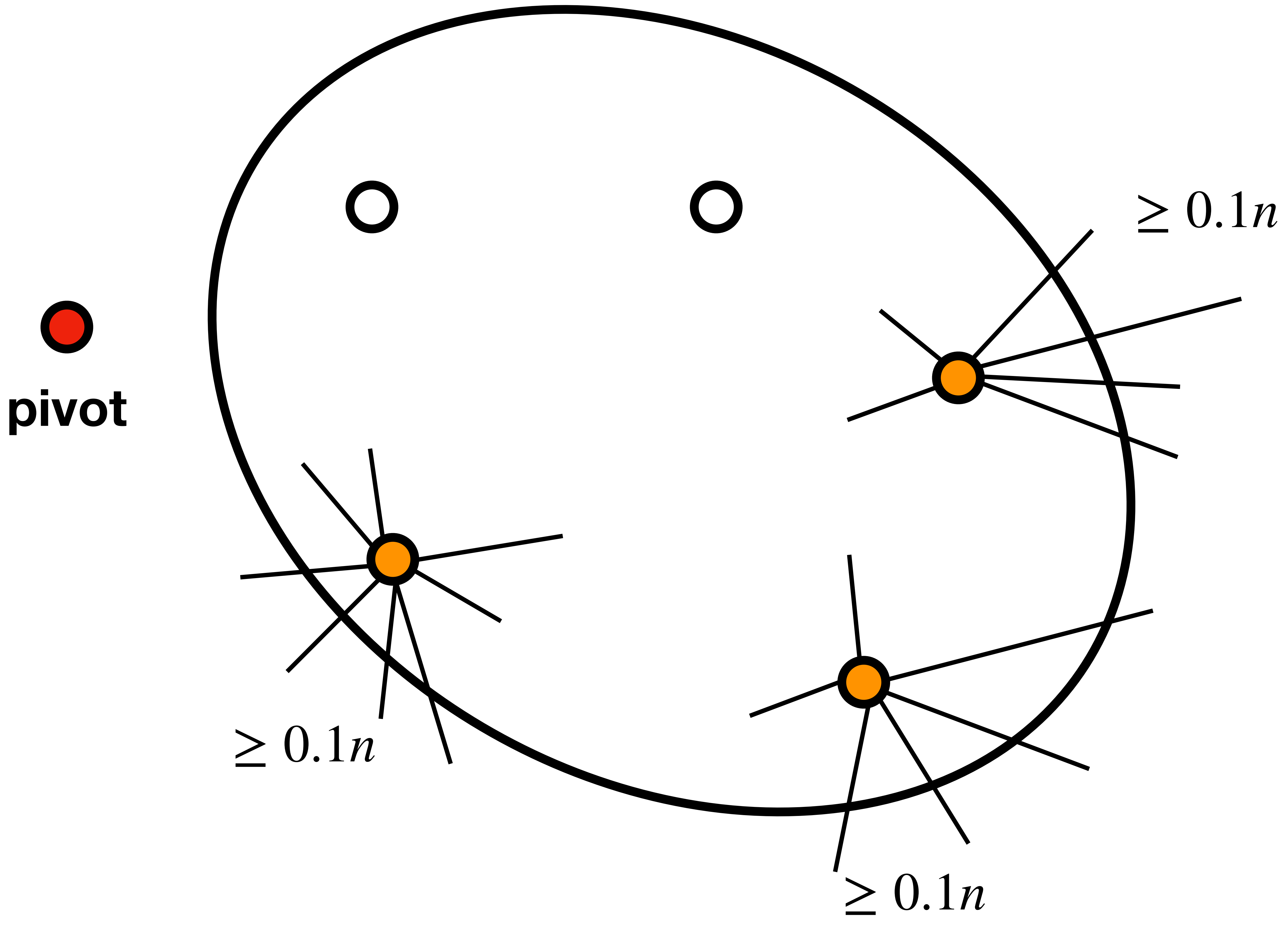


Type 1: small expanders:

- Expander contains less than $0.1n$ vertices
- An average vertex has at least $0.1n$ out-going edges
- Total #average vertices $= O(\phi n)$

Runtime Bottlenecks [AKT'21]

Simplifying assumption: most vertex degrees are at least $0.2n$



Type 1: small expanders:

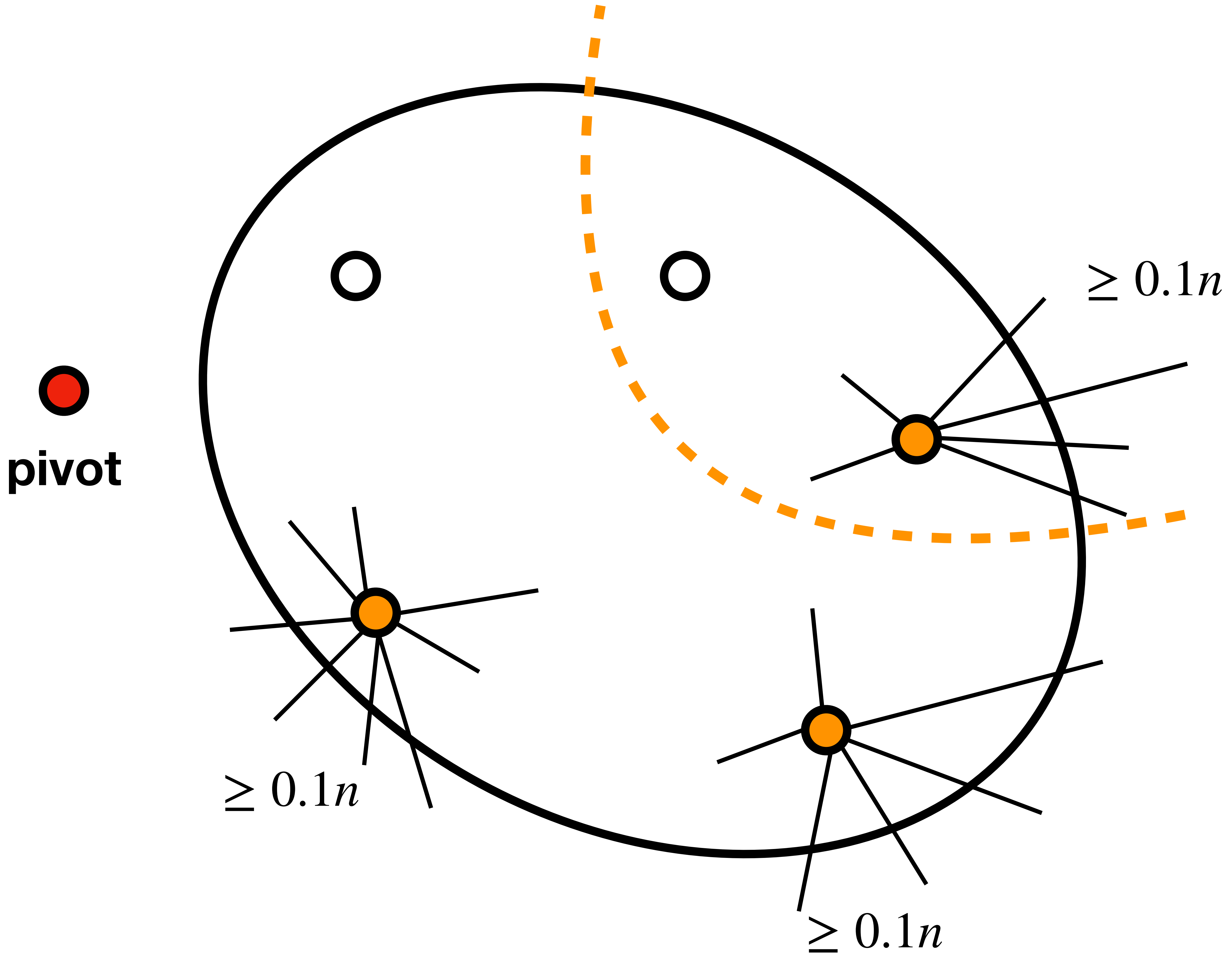
- Expander contains less than $0.1n$ vertices
- An average vertex has at least $0.1n$ out-going edges
- Total #average vertices $= O(\phi n)$

Solution:

- Compute min-cut for each average vertex **one-by-one**

Runtime Bottlenecks [AKT'21]

Simplifying assumption: most vertex degrees are at least $0.2n$



Type 1: small expanders:

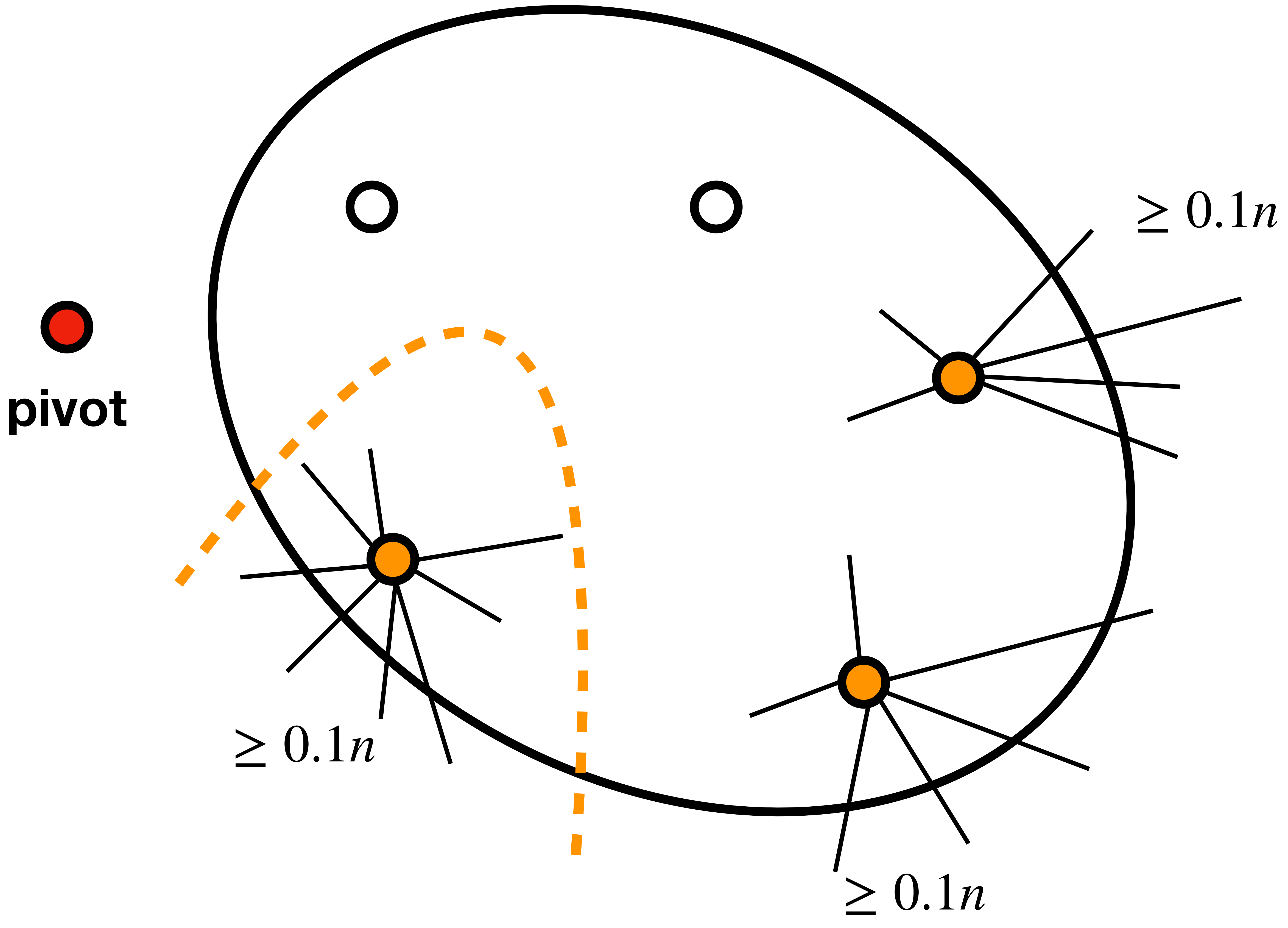
- Expander contains less than $0.1n$ vertices
- An average vertex has at least $0.1n$ out-going edges
- Total #average vertices $= O(\phi n)$

Solution:

- Compute min-cut for each average vertex **one-by-one**

Runtime Bottlenecks [AKT'21]

Simplifying assumption: most vertex degrees are at least $0.2n$



Type 1: small expanders:

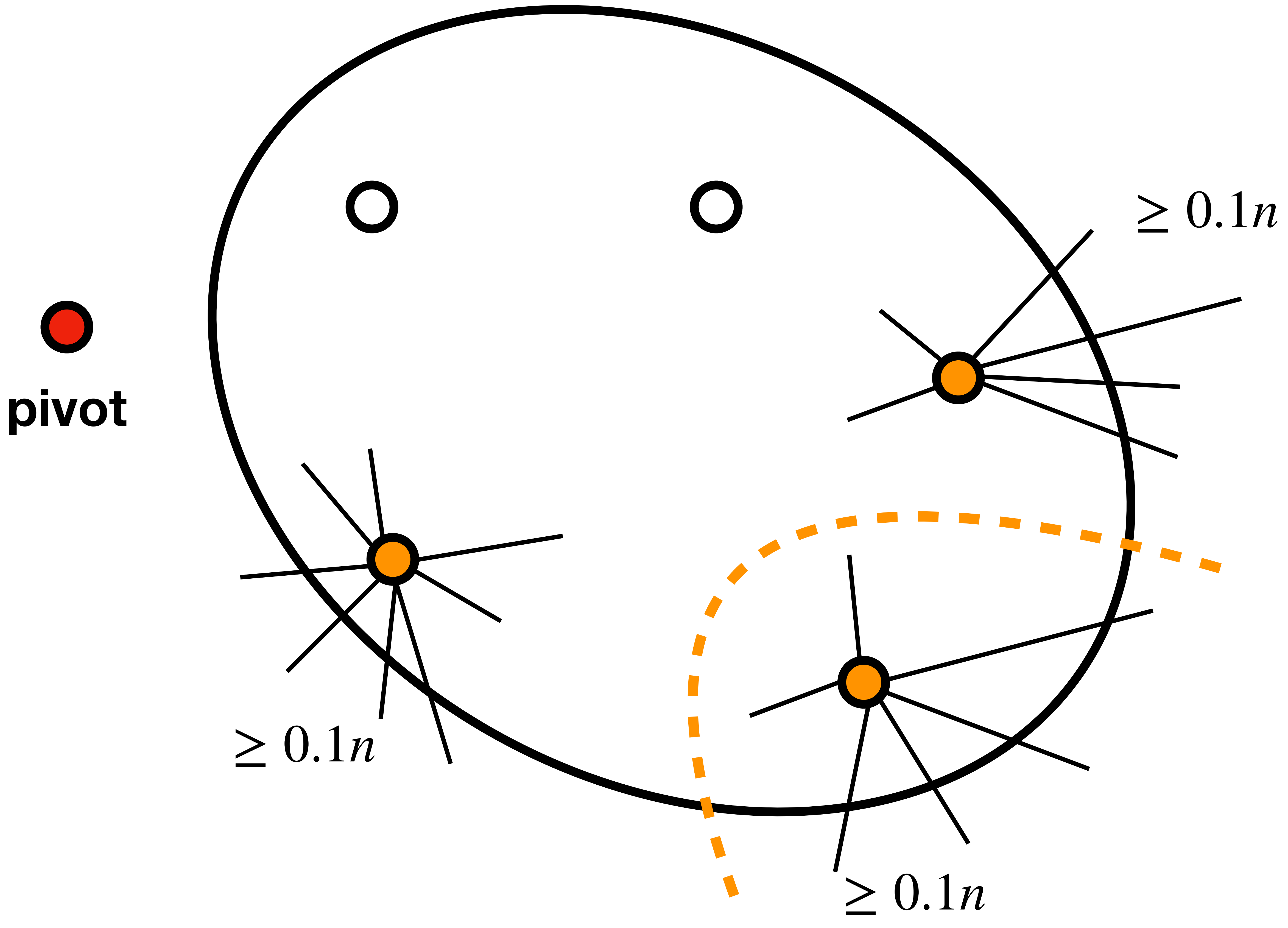
- Expander contains less than $0.1n$ vertices
- An average vertex has at least $0.1n$ out-going edges
- Total #average vertices $= O(\phi n)$

Solution:

- Compute min-cut for each average vertex **one-by-one**

Runtime Bottlenecks [AKT'21]

Simplifying assumption: most vertex degrees are at least $0.2n$



Type 1: small expanders:

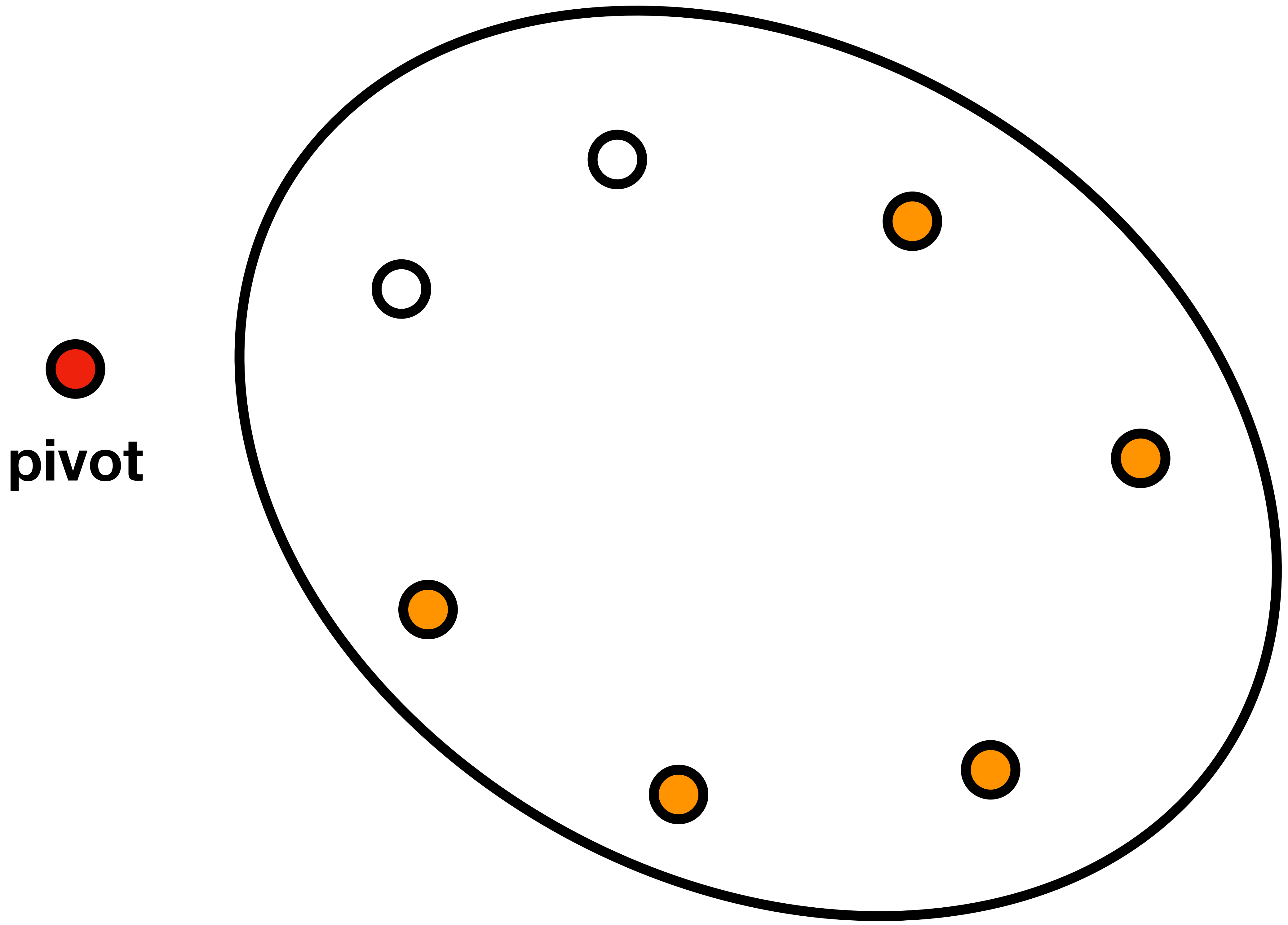
- Expander contains less than $0.1n$ vertices
- An average vertex has at least $0.1n$ out-going edges
- Total #average vertices $= O(\phi n)$

Solution:

- Compute min-cut for each average vertex **one-by-one**

Runtime Bottlenecks [AKT'21]

Simplifying assumption: most vertex degrees are at least $0.2n$

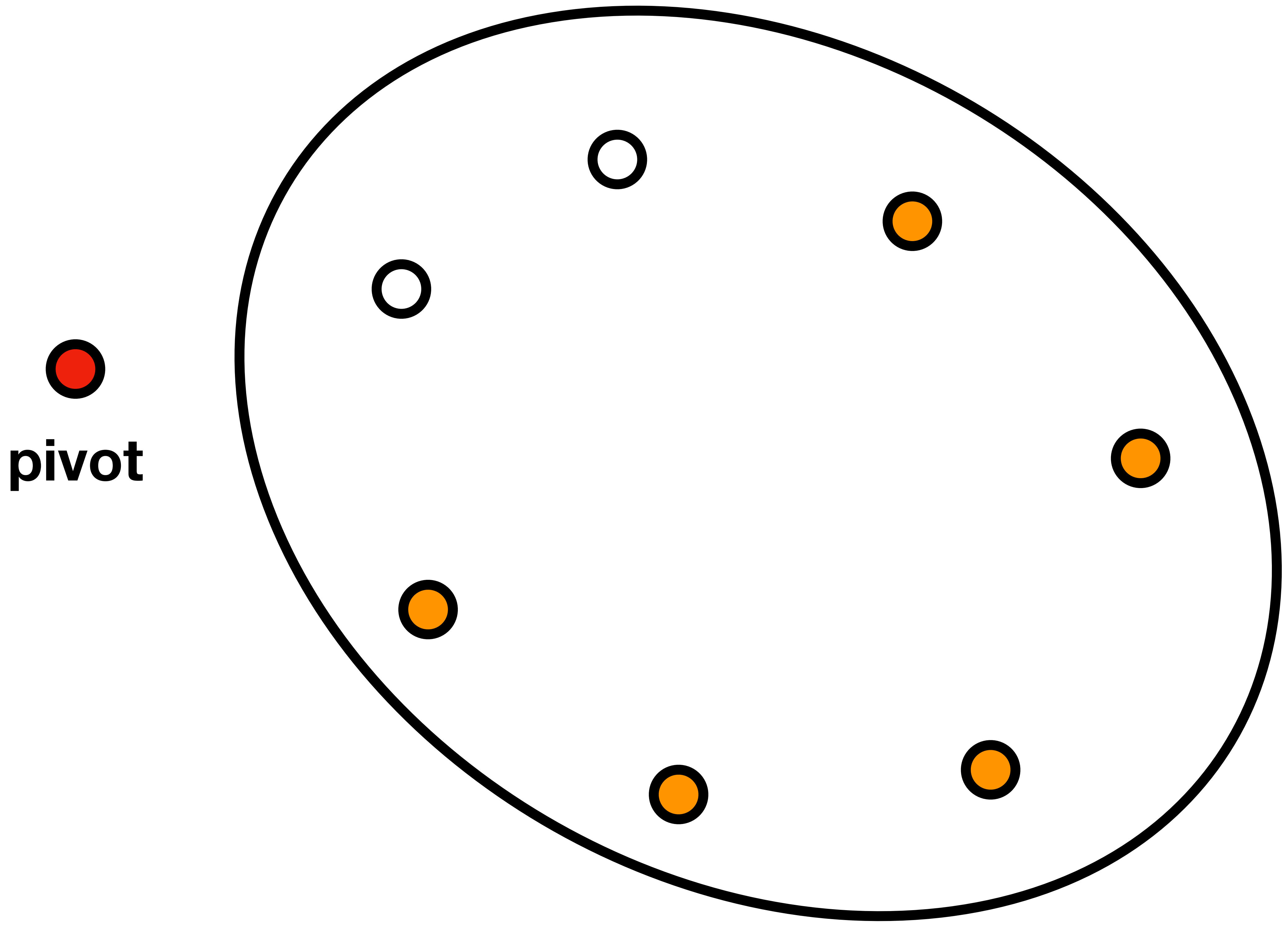


Type 2: large expanders & small cuts:

- Expander contains $\geq 0.1n$ vertices
- Cuts in expander have size at most $\tilde{O}(1/\phi)$

Runtime Bottlenecks [AKT'21]

Simplifying assumption: most vertex degrees are at least $0.2n$



Type 2: large expanders & small cuts:

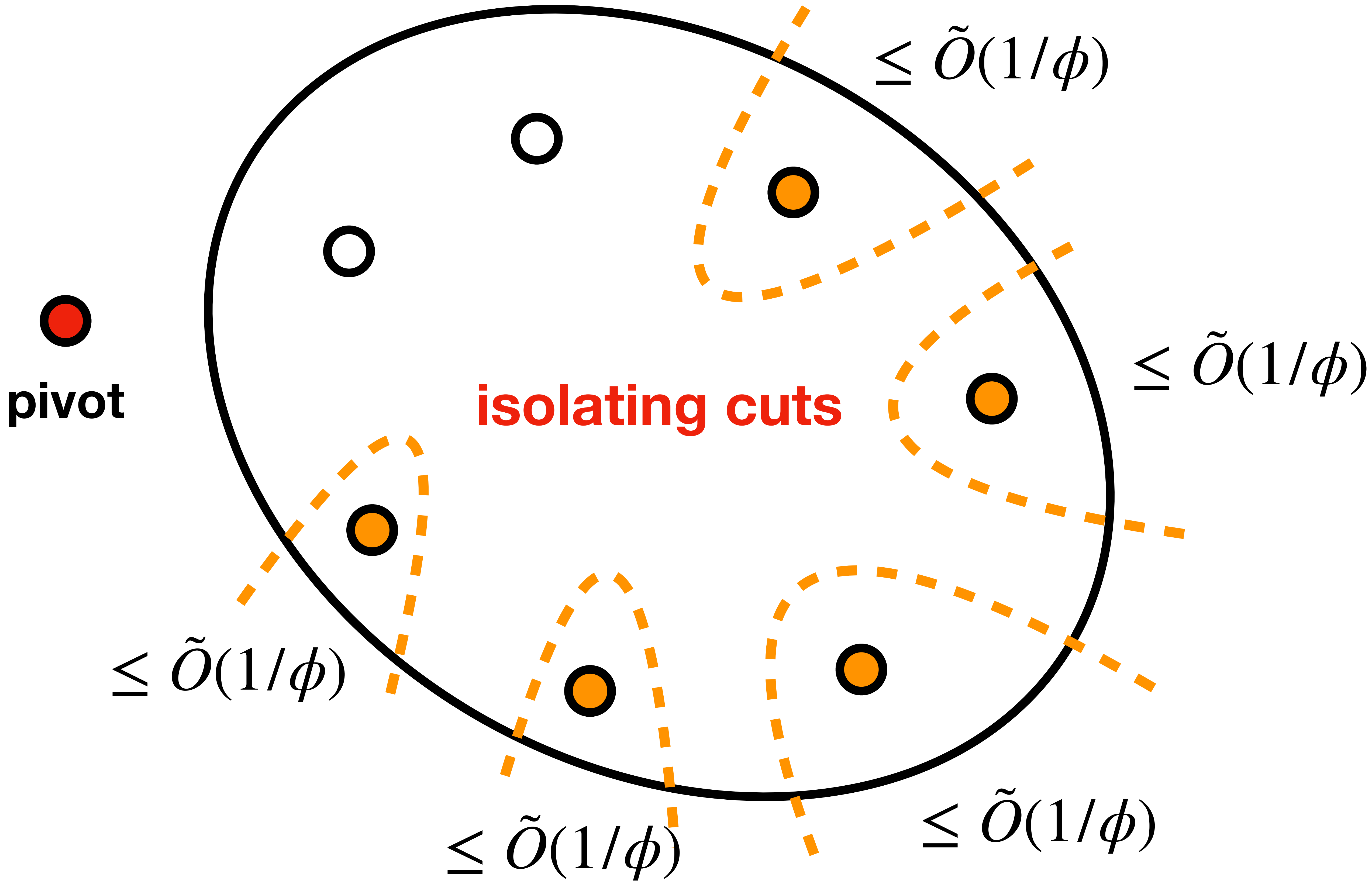
- Expander contains $\geq 0.1n$ vertices
- Cuts in expander have size at most $\tilde{O}(1/\phi)$

Solution:

- Compute **isolating cuts** [AKT'21, LP'20] in each large expander

Runtime Bottlenecks [AKT'21]

Simplifying assumption: most vertex degrees are at least $0.2n$



Type 2: large expanders & small cuts:

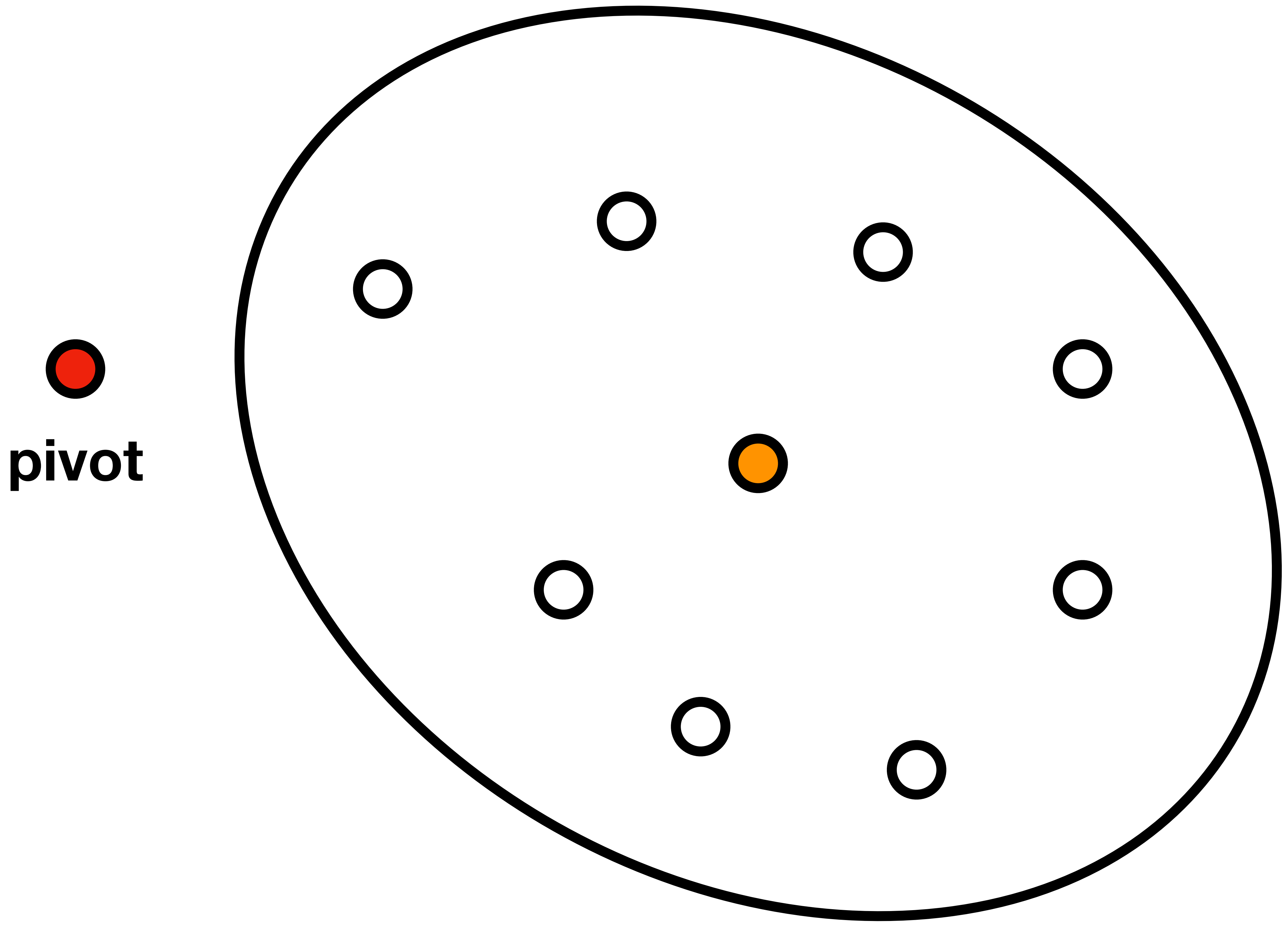
- Expander contains $\geq 0.1n$ vertices
- Cuts in expander have size at most $\tilde{O}(1/\phi)$

Solution:

- Compute **isolating cuts** [AKT'21, LP'20] in each large expander

Runtime Bottlenecks [AKT'21]

Simplifying assumption: most vertex degrees are at least $0.2n$

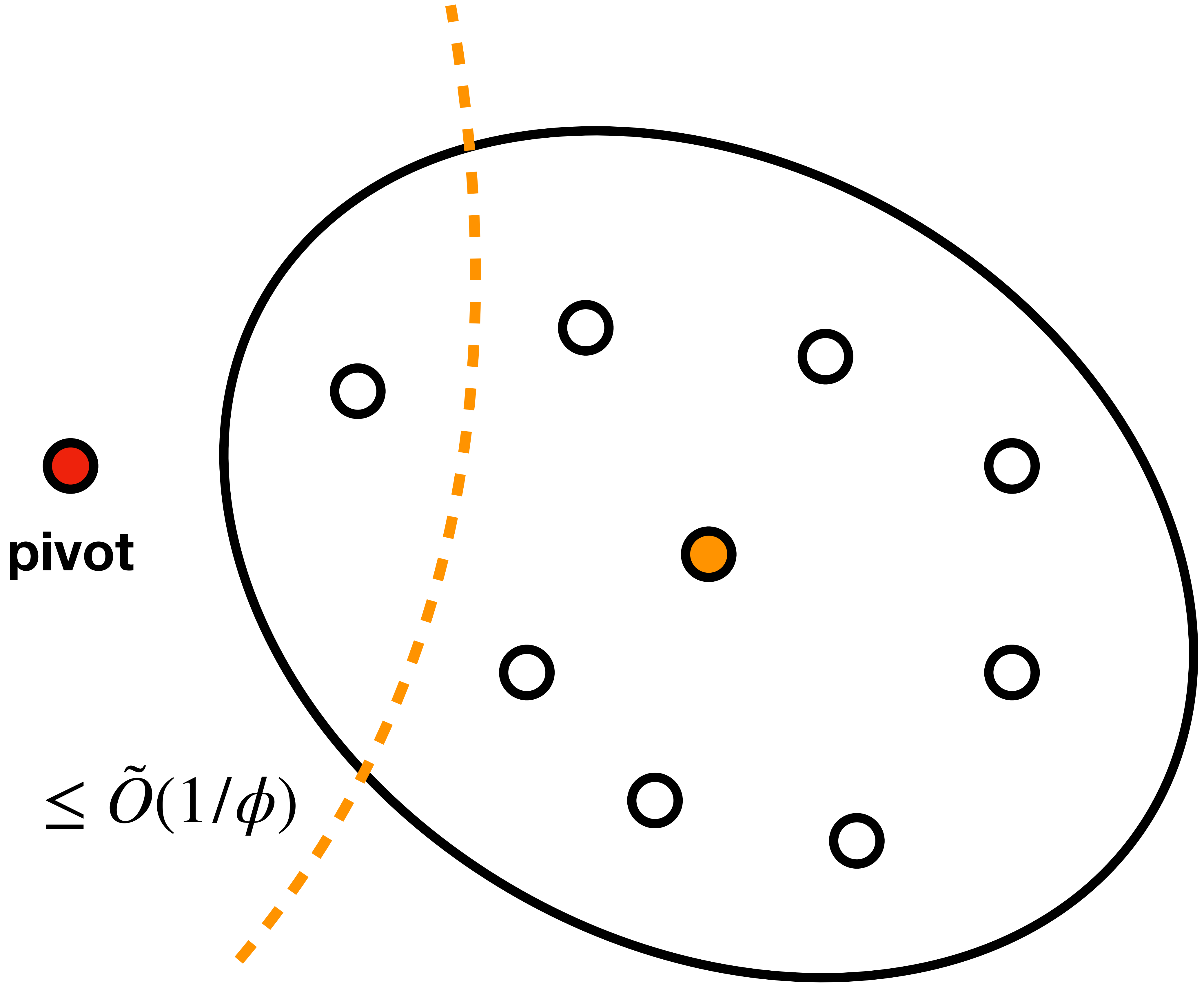


Type 3: large expanders & large cuts:

- Expander contains $\geq 0.1n$ vertices
- Cuts "minus" expander have size at most $\tilde{O}(1/\phi)$

Runtime Bottlenecks [AKT'21]

Simplifying assumption: most vertex degrees are at least $0.2n$

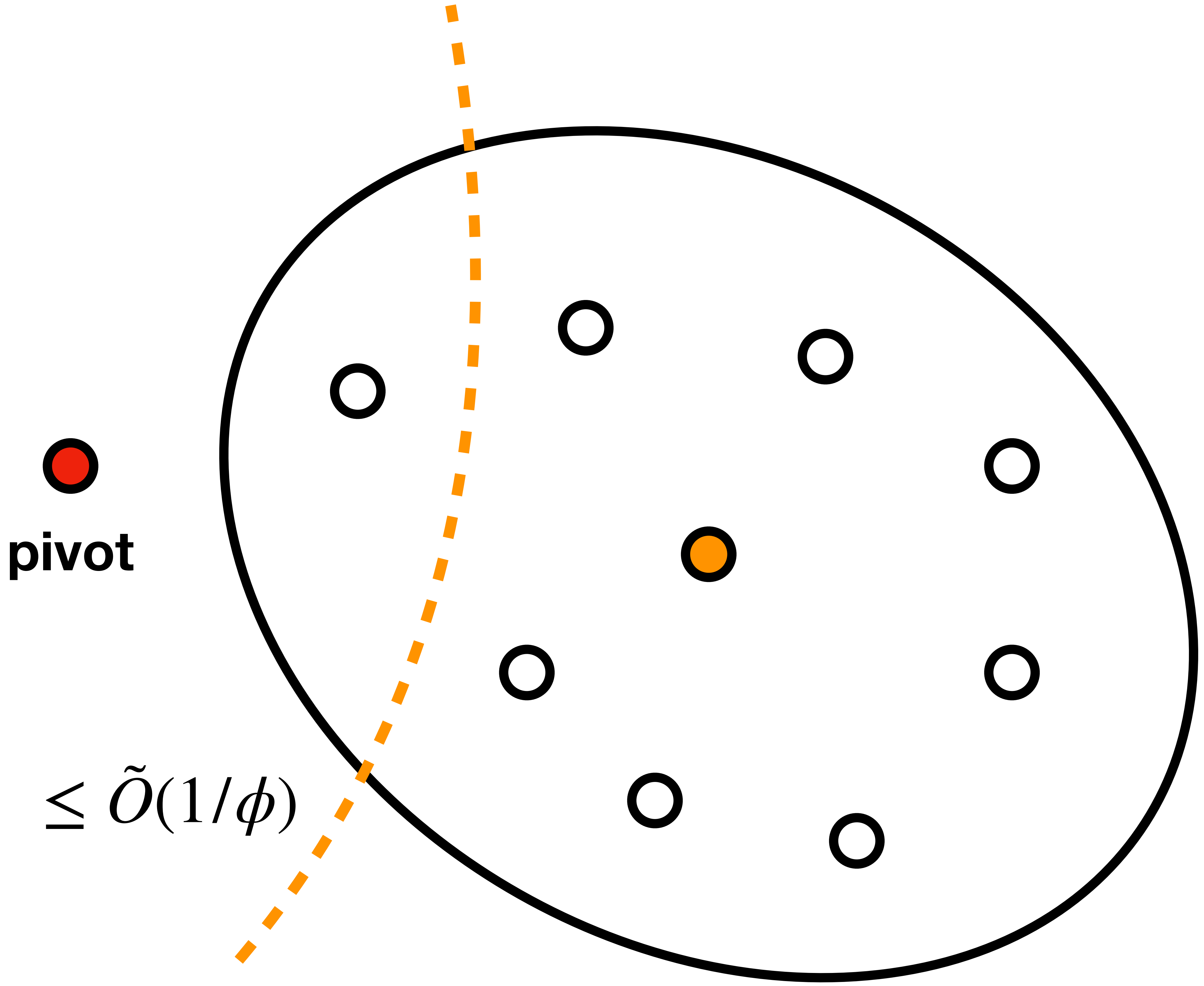


Type 3: large expanders & large cuts:

- Expander contains $\geq 0.1n$ vertices
- Cuts "minus" expander have size at most $\tilde{O}(1/\phi)$

Runtime Bottlenecks [AKT'21]

Simplifying assumption: most vertex degrees are at least $0.2n$



Type 3: large expanders & large cuts:

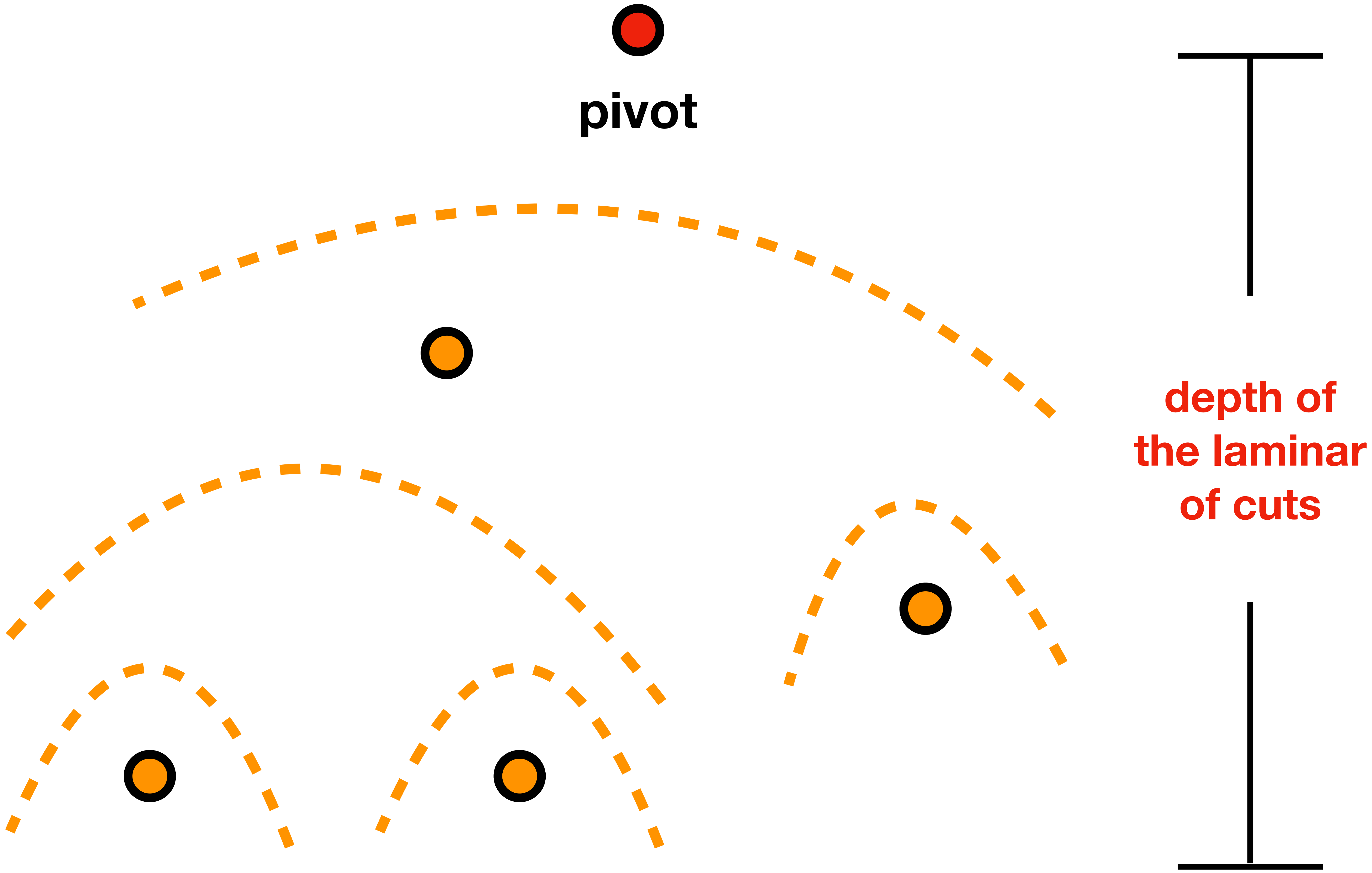
- Expander contains $\geq 0.1n$ vertices
- Cuts "minus" expander have size at most $\tilde{O}(1/\phi)$

Solution:

- Need to bound the **depth of the laminar family** of all such cuts

Runtime Bottlenecks [AKT'21]

Simplifying assumption: most vertex degrees are at least $0.2n$



Type 3: large expanders & large cuts:

- Expander contains $\geq 0.1n$ vertices
- Cuts "minus" expander have size at most $\tilde{O}(1/\phi)$

Solution:

- Need to bound the **depth of the laminar family** of all such cuts

The First Bottleneck

Simplifying assumption: most vertex degrees are at least $0.2n$

Running time:

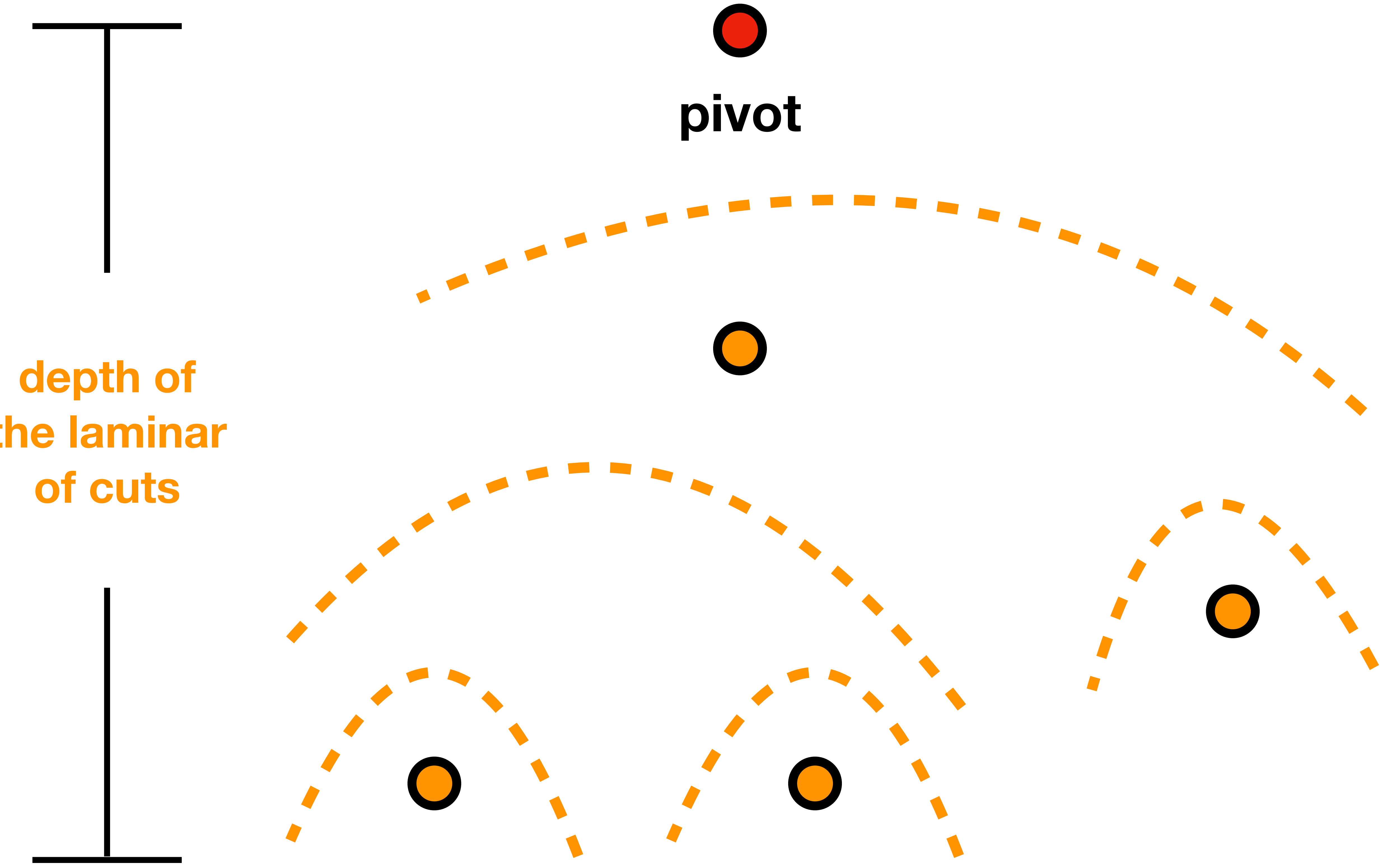
- Assume MaxFlow(m, n) = m+n
- **Type 1** min-cuts cost ϕn instances of **max-flow**
  runtime = ϕn^3
- **Type 2** min-cuts cost $1/\phi$ instances of **isolating cuts**
  runtime = n^2/ϕ
- Overall runtime = $\phi n^3 + n^2/\phi \geq n^{2.5}$

The Second Bottleneck

Simplifying assumption: most vertex degrees are at least $0.2n$

Running time:

- **Type 3** min-cuts costs $d =$ depth instances of **max-flow**
- Before that, need n/d instances of **max-flow** to make the laminar depth bounded by d
- Overall runtime = $(d + n/d) \cdot n^2 \geq n^{2.5}$



Attack on the Runtime Bottlenecks

The First Bottleneck

Runtime bottleneck:

- Type 1 min-cuts cost ϕn instances of **max-flow**
  runtime = ϕn^3
- Type 2 min-cuts cost $1/\phi$ instances of **isolating cuts**
  runtime = n^2/ϕ
- Overall runtime = $\phi n^3 + n^2/\phi \geq n^{2.5}$

Solution:

- Instead of doing type 1 **and** 2, just do type 1 **or** 2
- If type 2 involves more vertices, then only do type 2
- If type 1 involves more vertices, then the **graph is sparse**
  So **max-flow** should be cheaper

The First Bottleneck

Solution:

- Instead of doing type 1 **and** 2,
  just do type 1 **or** 2

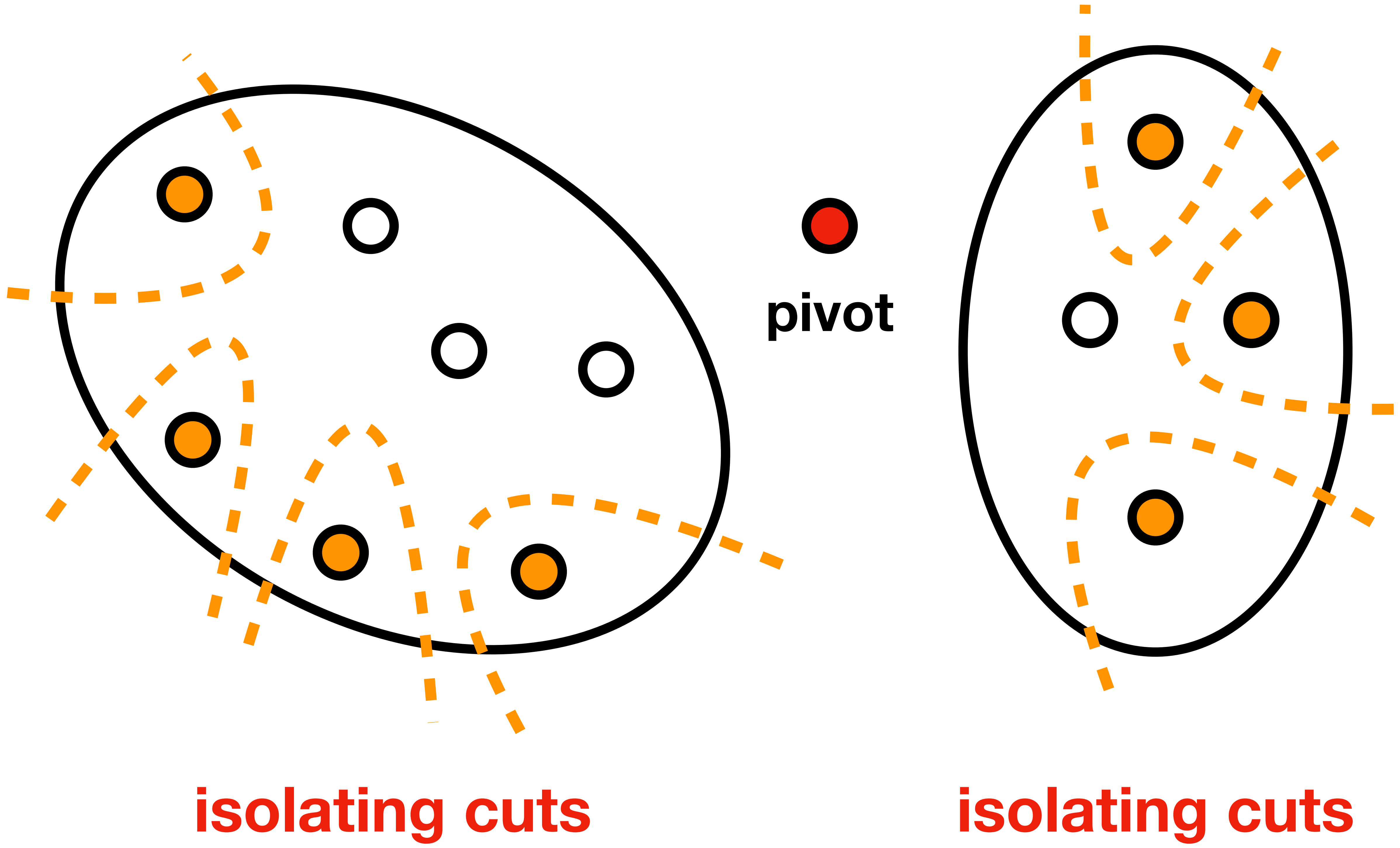
- If type 2 involves more vertices,
  then only do type 2

- If type 1 involves more vertices,
  then the graph is sparse
  So max-flow should be cheaper

The First Bottleneck

Solution:

- Instead of doing type 1 **and** 2, just do type 1 **or** 2
- If type 2 involves more vertices, then only do type 2
- If type 1 involves more vertices, then the graph is sparse
  So max-flow should be cheaper



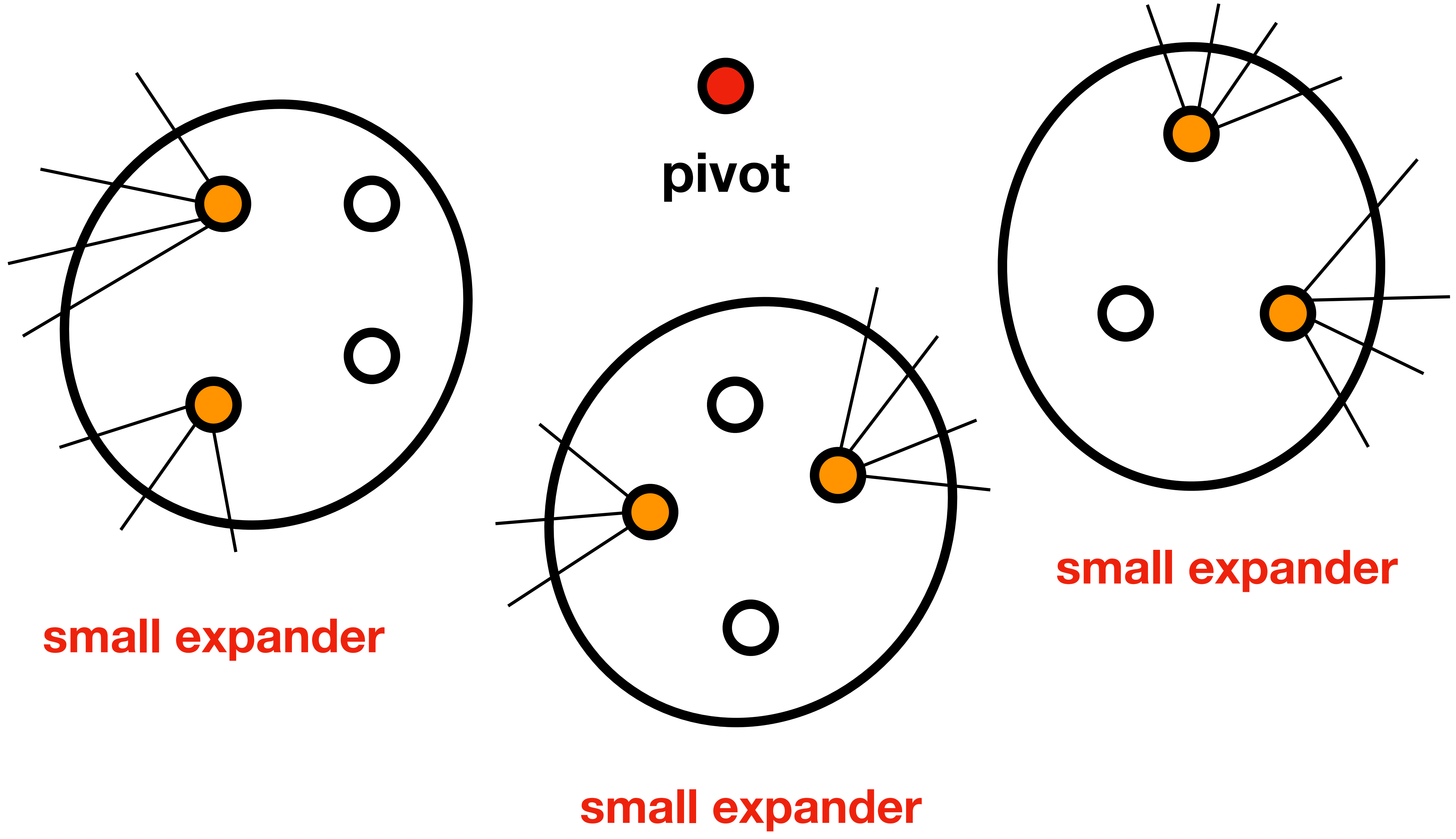
Recursion tree:

- **More than half** of the vertices are cut-off, so the recursion tree depth is still logarithmic

The First Bottleneck

Solution:

- Instead of doing type 1 **and** 2, just do type 1 **or** 2
- If type 2 involves more vertices, then only do type 2
- If type 1 involves more vertices, then the graph is sparse
  So max-flow should be cheaper



Total #edges:

- Most incident edges of yellow vertices cross the border
- Since total #inter-cluster edges $\leq \phi n^2$ in an expander decomposition, total degree is $\leq \phi n^2$

The First Bottleneck

Solution:

- Instead of doing type 1 **and** 2, just do type 1 **or** 2
- If type 2 involves more vertices, then only do type 2
- If type 1 involves more vertices, then the graph is sparse
  So max-flow should be cheaper

Sanity check:

- New runtime = $n^2/\phi + \phi^2 n^3 = n^{8/3 < 2.5}$

The First Bottleneck

Solution:

- Instead of doing type 1 **and** 2, just do type 1 **or** 2

- If type 2 involves more vertices, then only do type 2

- If type 1 involves more vertices, then the graph is sparse
  So max-flow should be cheaper

Sanity check:

- New runtime = $n^2/\phi + \phi^2 n^3 = n^{8/3 < 2.5}$

General cases:

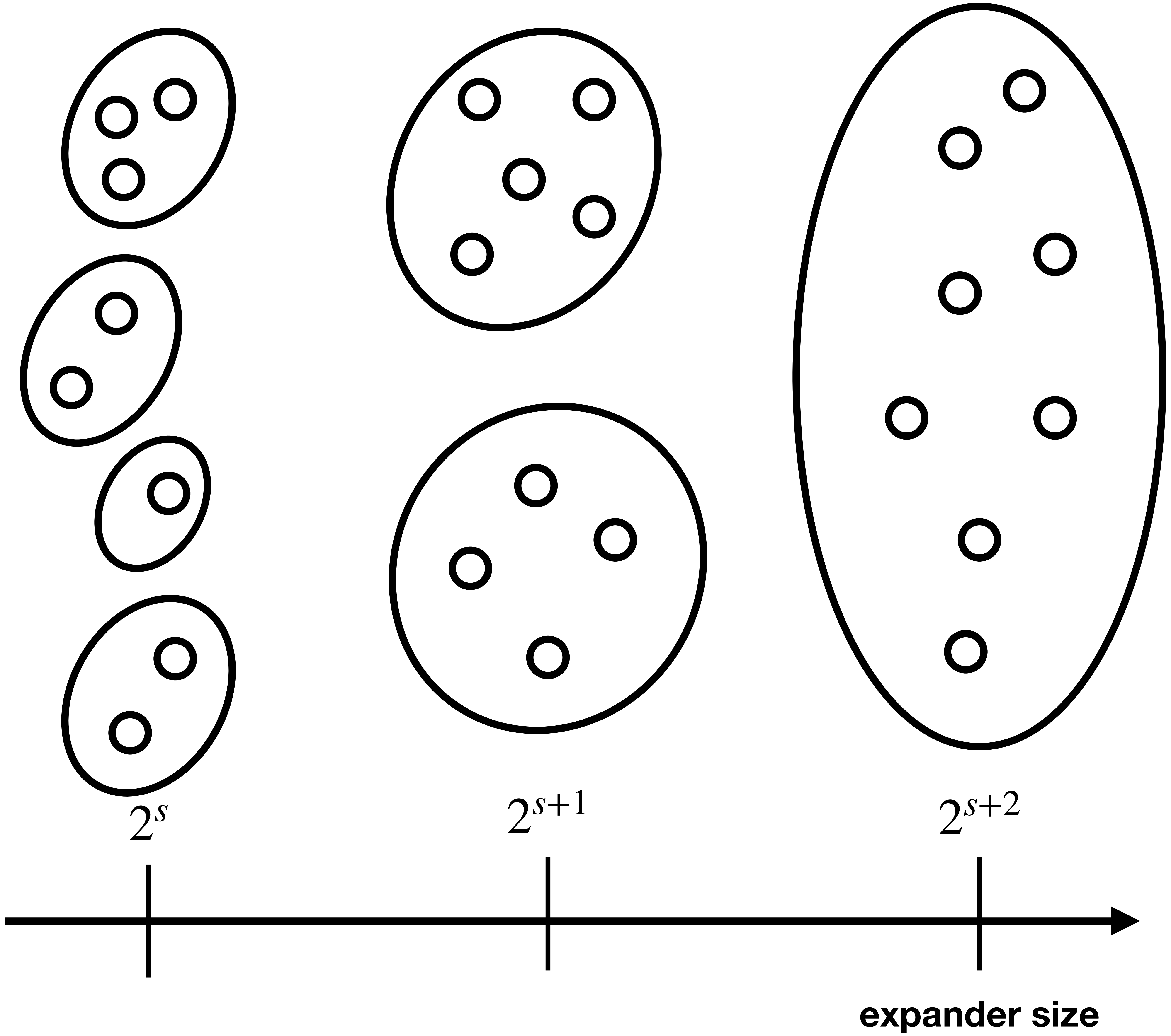
- How to **remove the assumption** that most vertex degrees are $\geq 0.2n$

- How to **achieve** n^2 instead of $n^{8/3}$

The First Bottleneck

General cases:

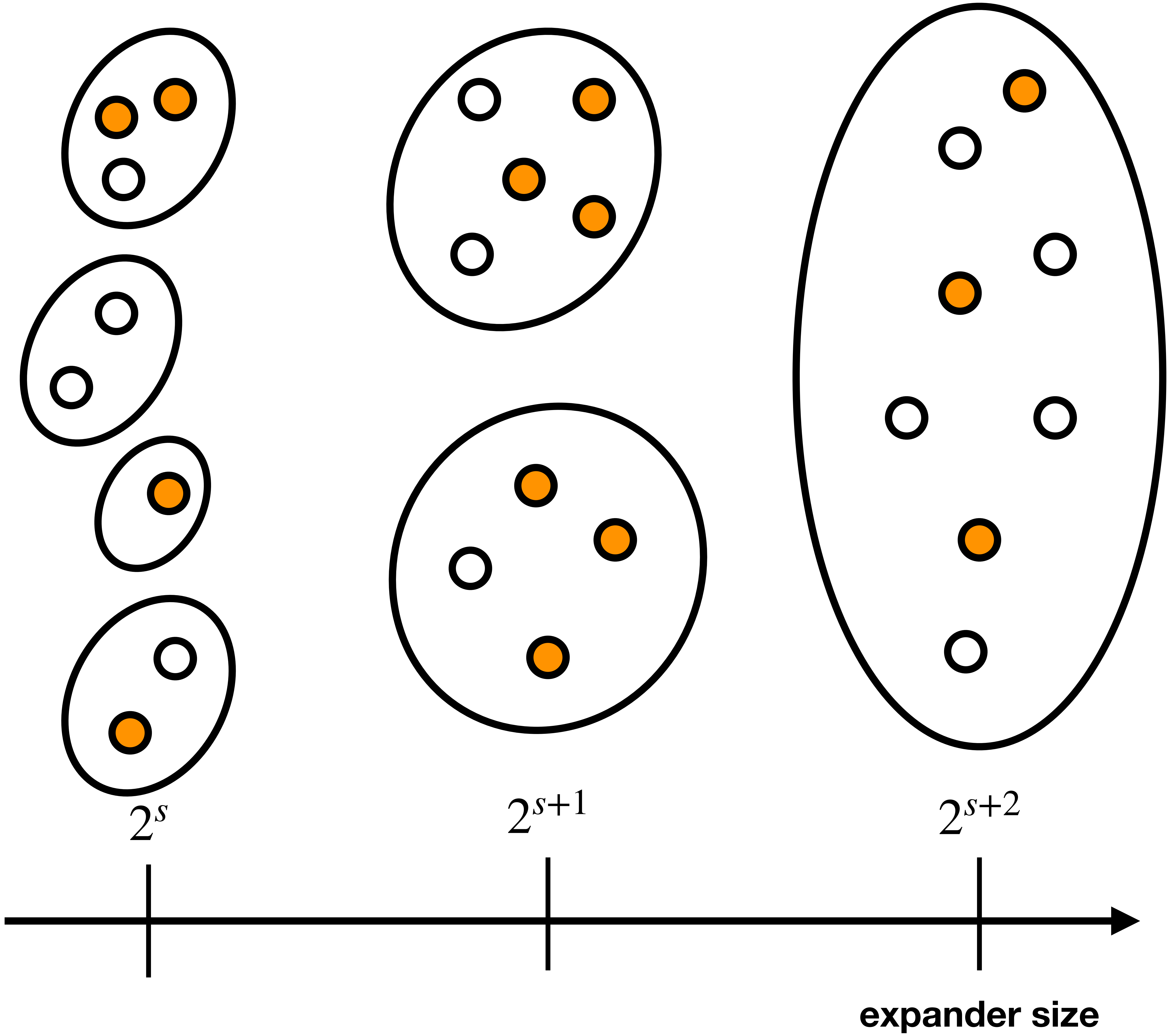
1. Define $U_i = \{v \mid \deg(v) \in [2^i, 2^{i+1})\}$
2. Pick i such that $2^i |U_i|$ is maximized
3. Define subset of expanders $\mathscr{C}_j = \{C \mid |C| \in [2^j, 2^{j+1})\}$
4. Pick j such that $|\mathscr{C}_j \cap U_i|$ is maximized
5. Compute isolating cuts for each $C \cap U_i$ where $C \in \mathscr{C}_j$



The First Bottleneck

General cases:

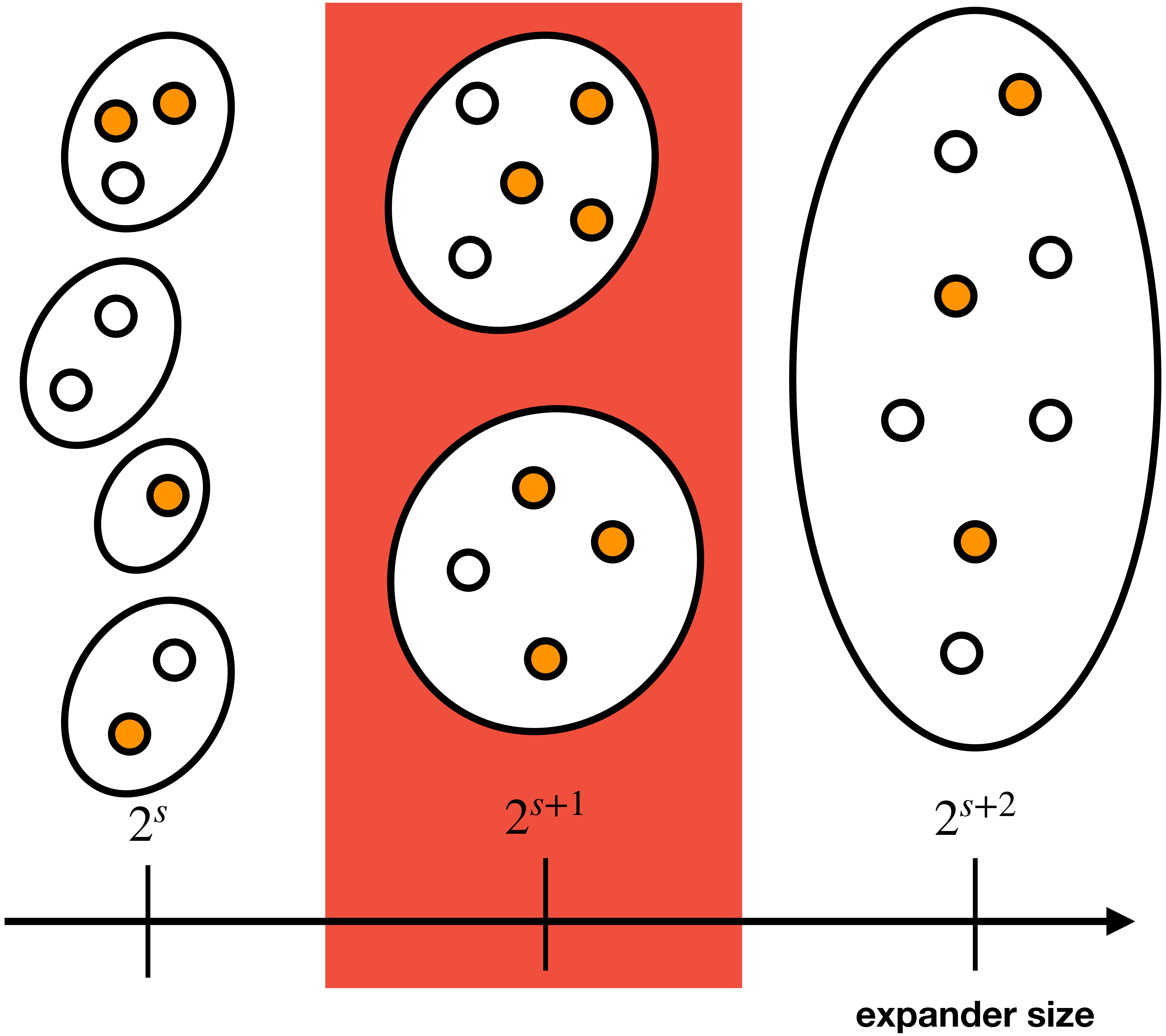
1. Define $U_i = \{v \mid \deg(v) \in [2^i, 2^{i+1})\}$
2. Pick i such that $2^i |U_i|$ is maximized
3. Define subset of expanders $\mathscr{C}_j = \{C \mid |C| \in [2^j, 2^{j+1})\}$
4. Pick j such that $|\mathscr{C}_j \cap U_i|$ is maximized
5. Compute isolating cuts for each $C \cap U_i$ where $C \in \mathscr{C}_j$



The First Bottleneck

General cases:

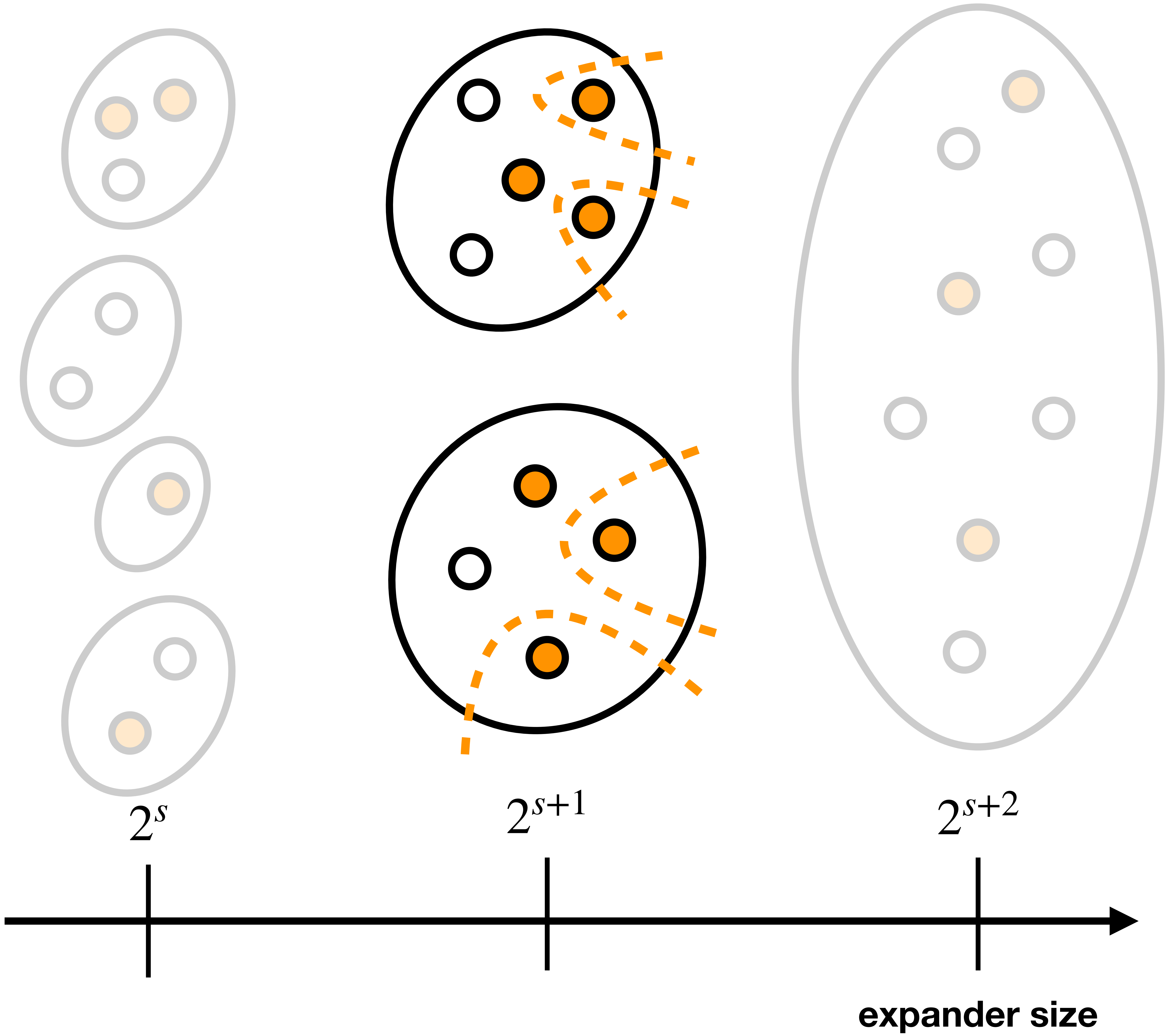
1. Define $U_i = \{v \mid \deg(v) \in [2^i, 2^{i+1})\}$
2. Pick i such that $2^i |U_i|$ is maximized
3. Define subset of expanders $\mathscr{C}_j = \{C \mid |C| \in [2^j, 2^{j+1})\}$
4. Pick j such that $|\mathscr{C}_j \cap U_i|$ is maximized
5. Compute isolating cuts for each $C \cap U_i$ where $C \in \mathscr{C}_j$



The First Bottleneck

<u>General cases:</u>

1. Define $U_i = \{v \mid \deg(v) \in [2^i, 2^{i+1})\}$
2. Pick i such that $2^i |U_i|$ is maximized
3. Define subset of expanders $\mathscr{C}_j = \{C \mid |C| \in [2^j, 2^{j+1})\}$
4. Pick j such that $|\mathscr{C}_j \cap U_i|$ is maximized
5. Compute isolating cuts for each $C \cap U_i$ where $C \in \mathscr{C}_j$



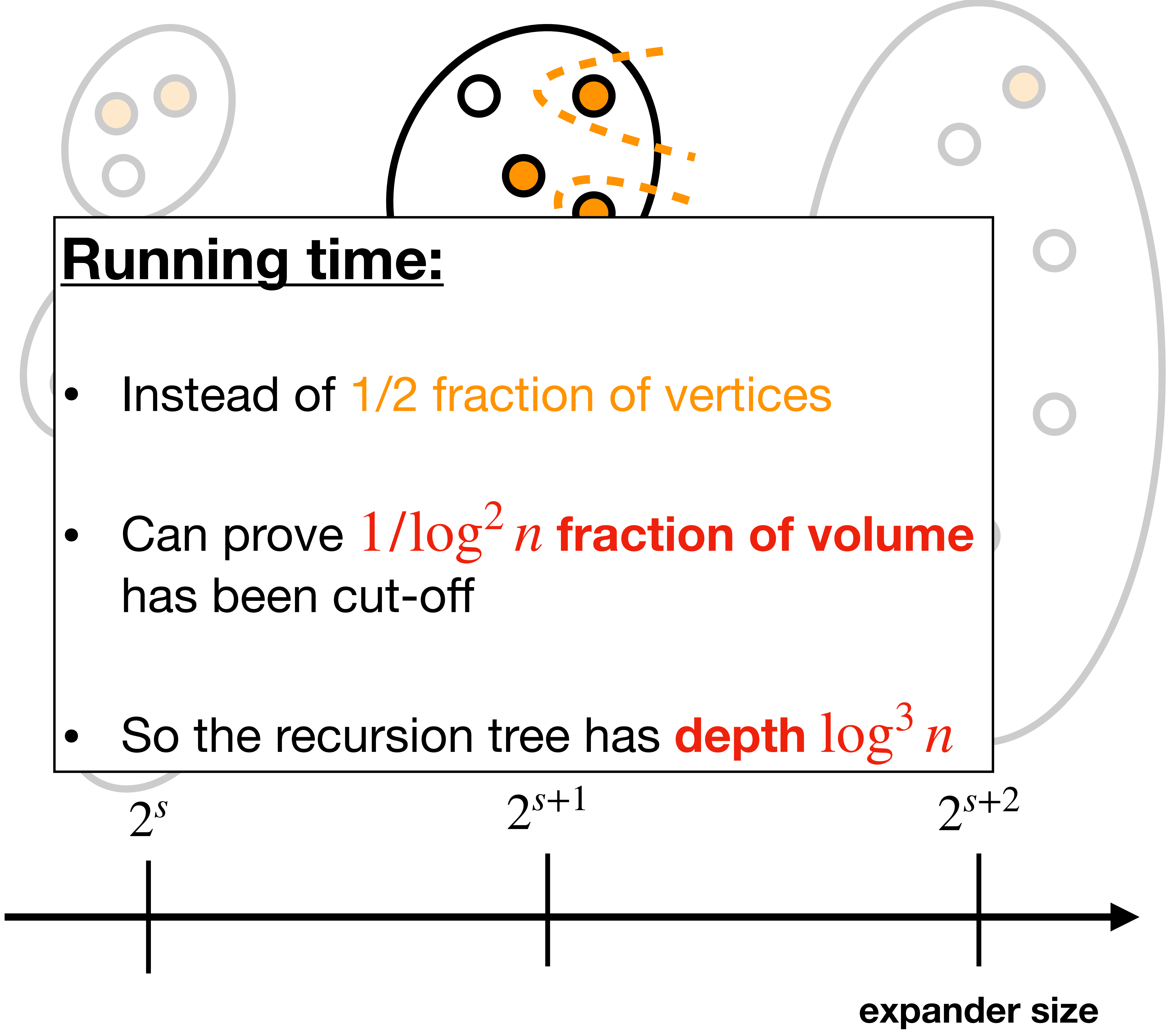
The First Bottleneck

General cases:

1. Define $U_i = \{v \mid \deg(v) \in [2^i, 2^{i+1})\}$
2. Pick i such that $2^i |U_i|$ is maximized
3. Define subset of expanders
   $\mathscr{C}_j = \{C \mid |C| \in [2^j, 2^{j+1})\}$
4. Pick j such that $|\mathscr{C}_j \cap U_i|$ is maximized
5. Compute isolating cuts for each $C \cap U_i$ where $C \in \mathscr{C}_j$

Running time:

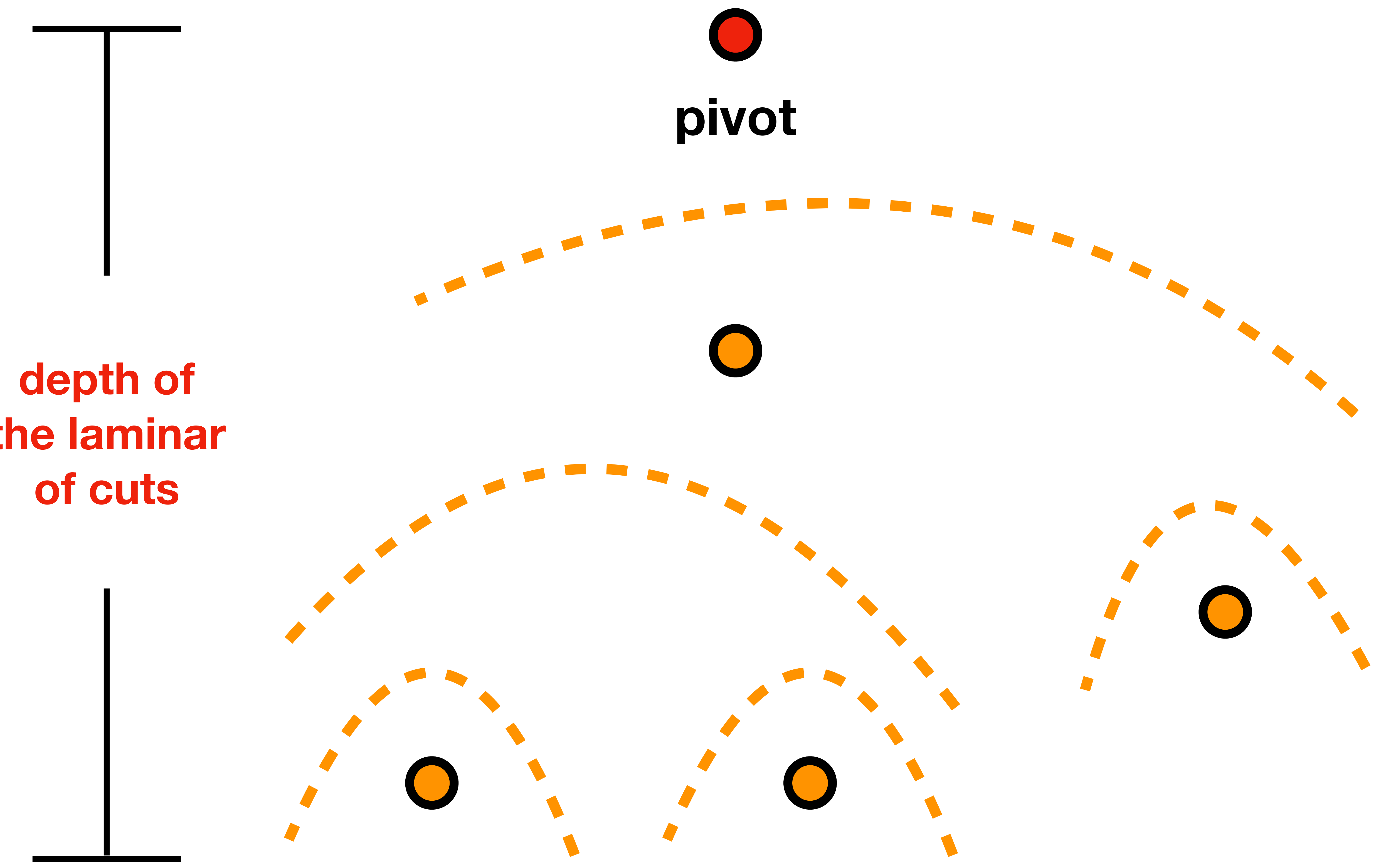
- Instead of 1/2 fraction of vertices
- Can prove $1/\log^2 n$ **fraction of volume** has been cut-off
- So the recursion tree has **depth** $\log^3 n$



The Second Bottleneck

Previous runtime:

- Type 3 min-cuts costs $d =$ depth instances of **max-flow**
- Runtime = $(d + n/d) \cdot n^2$



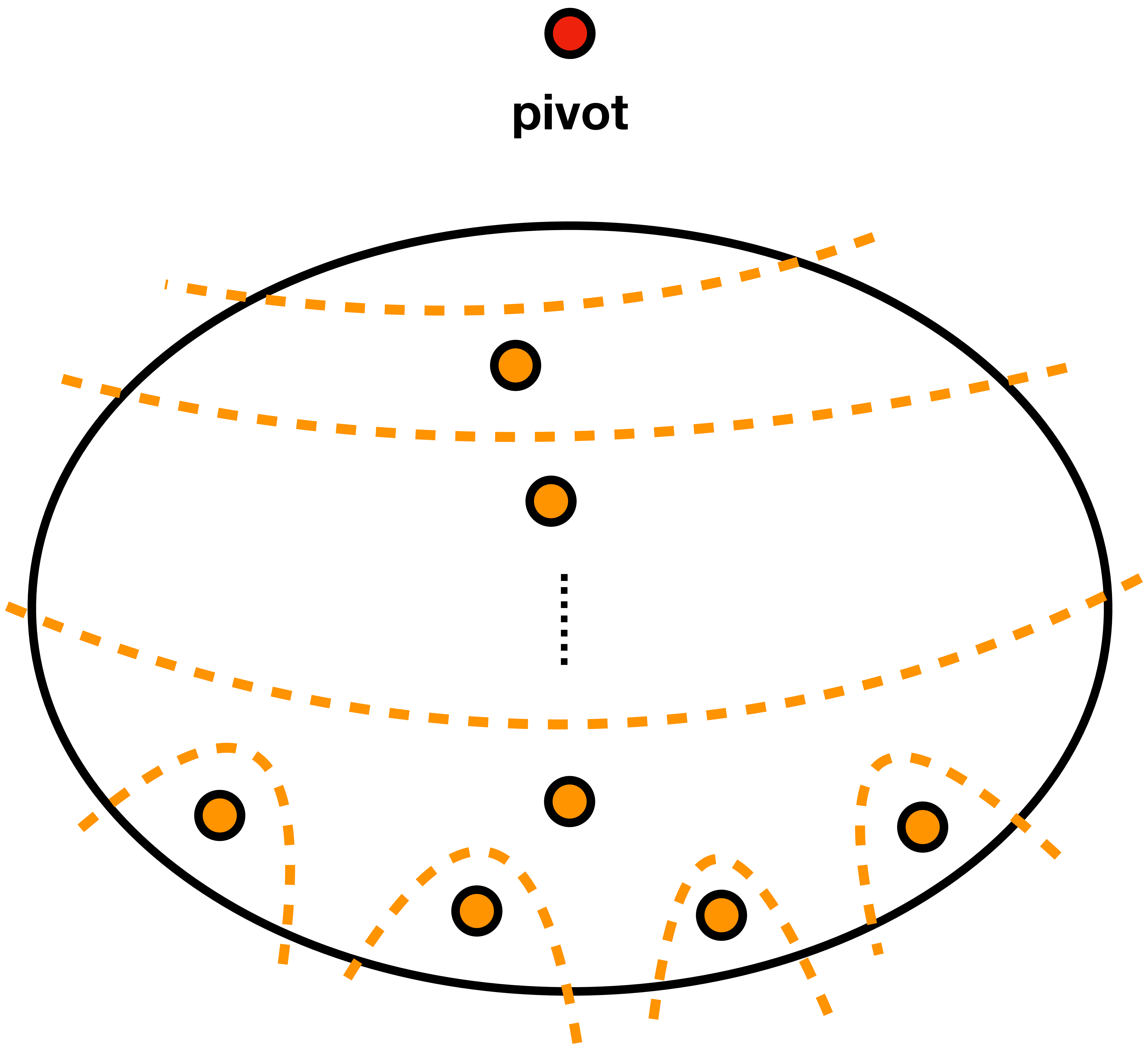
The Second Bottleneck

Previous runtime:

- Type 3 min-cuts costs
  $d =$ depth instances of **max-flow**
- Runtime = $(d + n/d) \cdot n^2$

New runtime:

- Take advantage of expanders



The Second Bottleneck

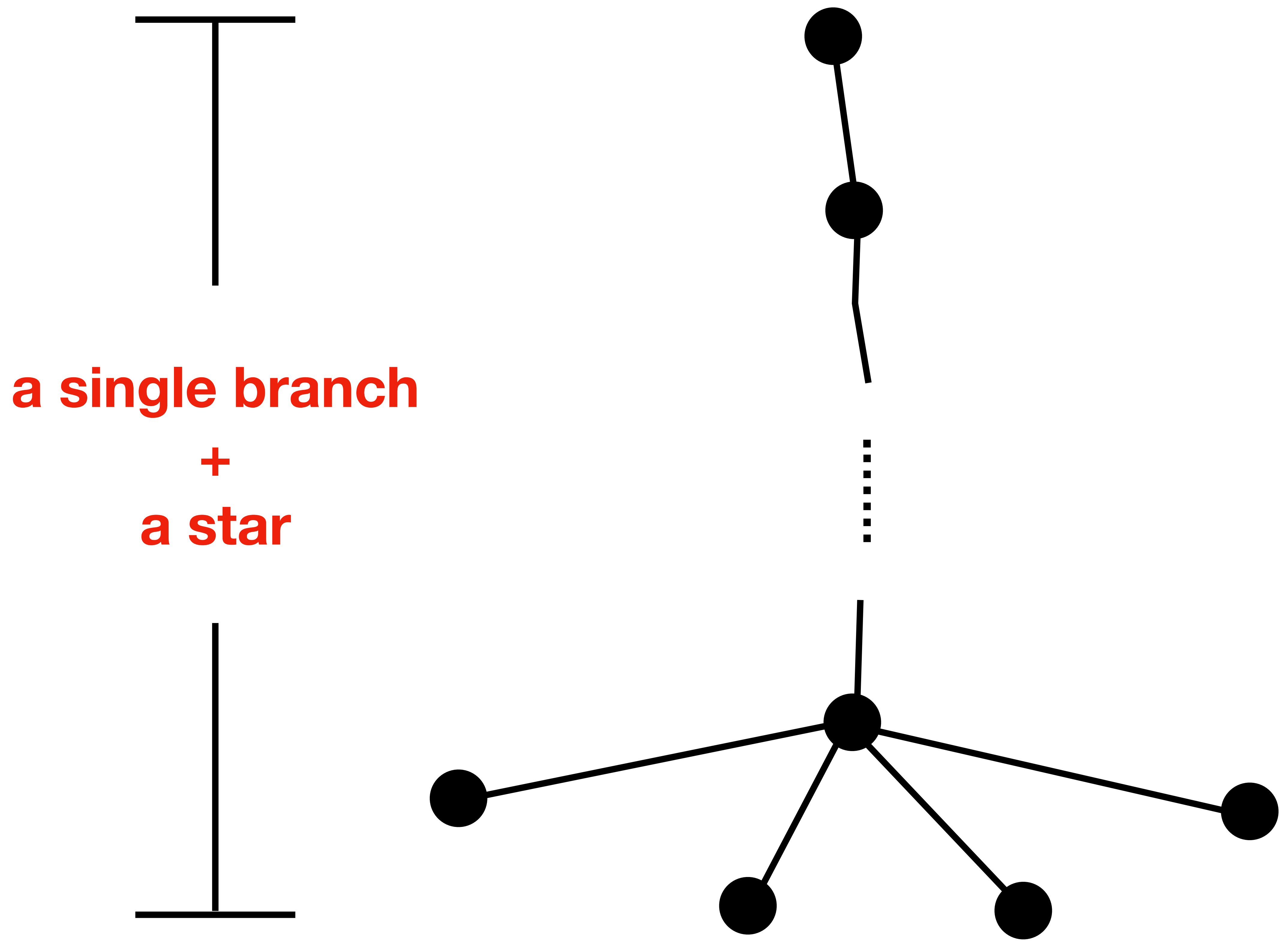
Previous runtime:

- Type 3 min-cuts costs
  $d =$ depth instances of **max-flow**
- Runtime = $(d + n/d) \cdot n^2$

New runtime:

- Take advantage of expanders
- Runtime = n^2/ϕ

laminar structure of min cuts



Further Directions

1. Sub-quadratic Gomory-Hu trees in weighted graphs?
2. Deterministic sub-cubic Gomory-Hu trees in weighted graphs?