Reachability and Strong-connectivity under Failures

Keerti Choudhary
(Weizmann Institute → Tel Aviv University)

Based upon..

- Surender Baswana, Keerti Choudhary, Liam Roditty: Fault tolerant subgraph for single source reachability: generic and optimal. STOC 2016 and SICOMP 2018.

- Surender Baswana, Keerti Choudhary, Liam Roditty: An Efficient Strongly Connected Components Algorithm in the Fault Tolerant Model. ICALP 2017 and and Algorithmica 2019.

Fundamental Graph Problems

Fundamental Graph Problems

Fundamental
Graph
Problems

Fundamental Graph Problems

Reachability

Fundamental Graph Problems

Fundamental Graph Problems

Reachability

Shortest-path

Fundamental
Graph
Problems

Fundamental Graph Problems

Reachability

Shortest-path

Max-flows

Fundamental
Graph
Problems

Fundamental Graph Problems

Reachability

Shortest-path

Max-flows

Fundamental Graph Problems

minimum-cut

Fundamental Graph Problems

Reachability

Shortest-path

Fundamental Graph Problems

Max-flows

minimum-cut

Connectivity

Fundamental Graph Problems

Reachability

Shortest-path

Max-flows

Fundamental Graph Problems

minimum-cut

Connectivity

strong-connectivity

Fundamental Graph Problems

Fundamental Graph Problems

Reachability

Shortest-path

Max-flows

minimum-cut

Connectivity

strong-connectivity

Matching

Fundamental Graph Problems

Reachability

Shortest-path

Fundamental Graph Problems

Max-flows

minimum-cut

Connectivity

strong-connectivity

We already have efficient solutions..

Matching

Fundamental Graph Problems

Reachability

Shortest-path

Max-flows

Fundamental Graph Problems

minimum-cut

Connectivity

strong-connectivity

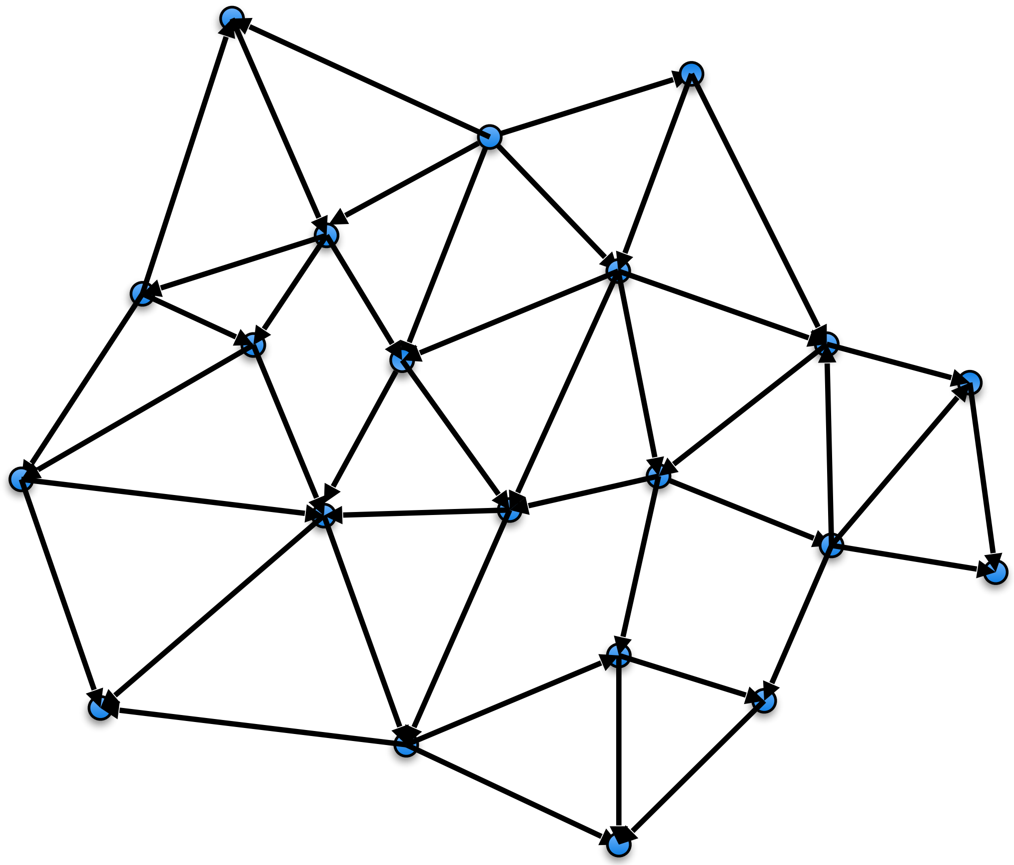
What if there are faults?

Matching

Fault Tolerant Model

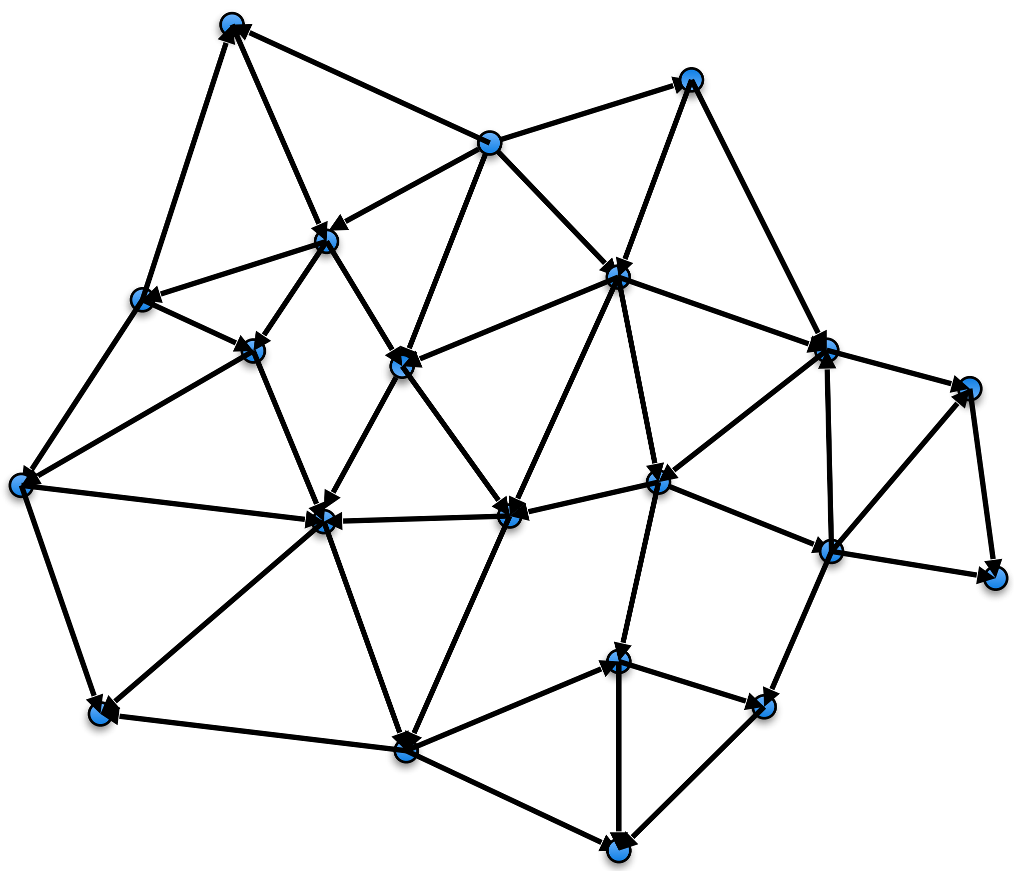
Fault Tolerant Model

G



Fault Tolerant Model

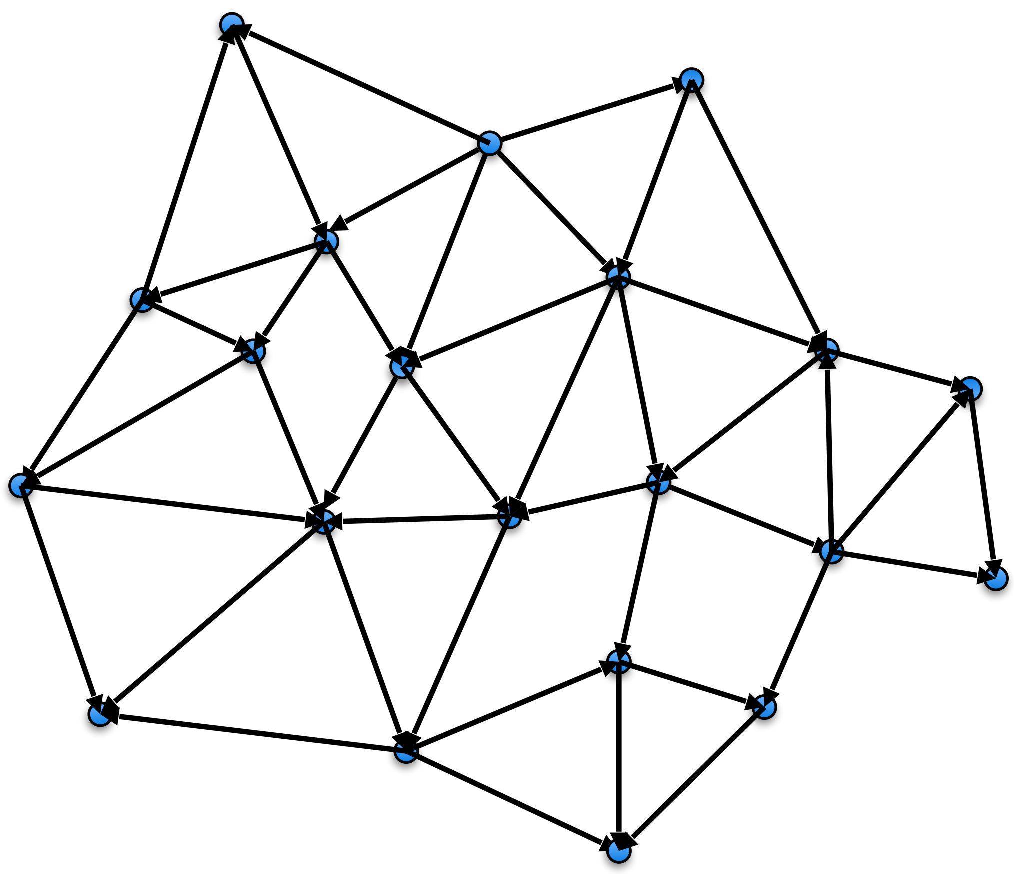
G



$k = 2$(Faults)

Fault Tolerant Model

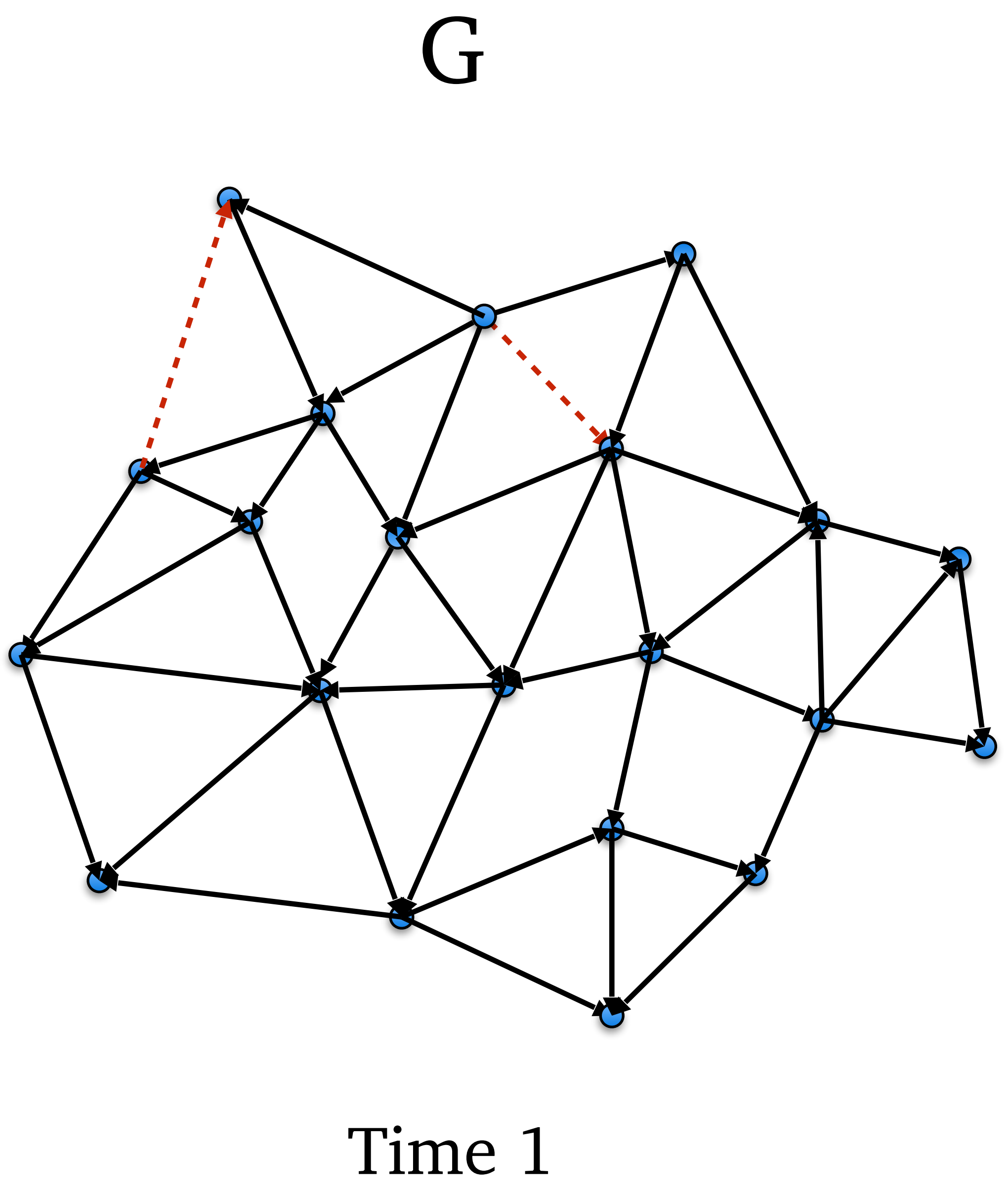
G



Time 0

$k = 2$(Faults)

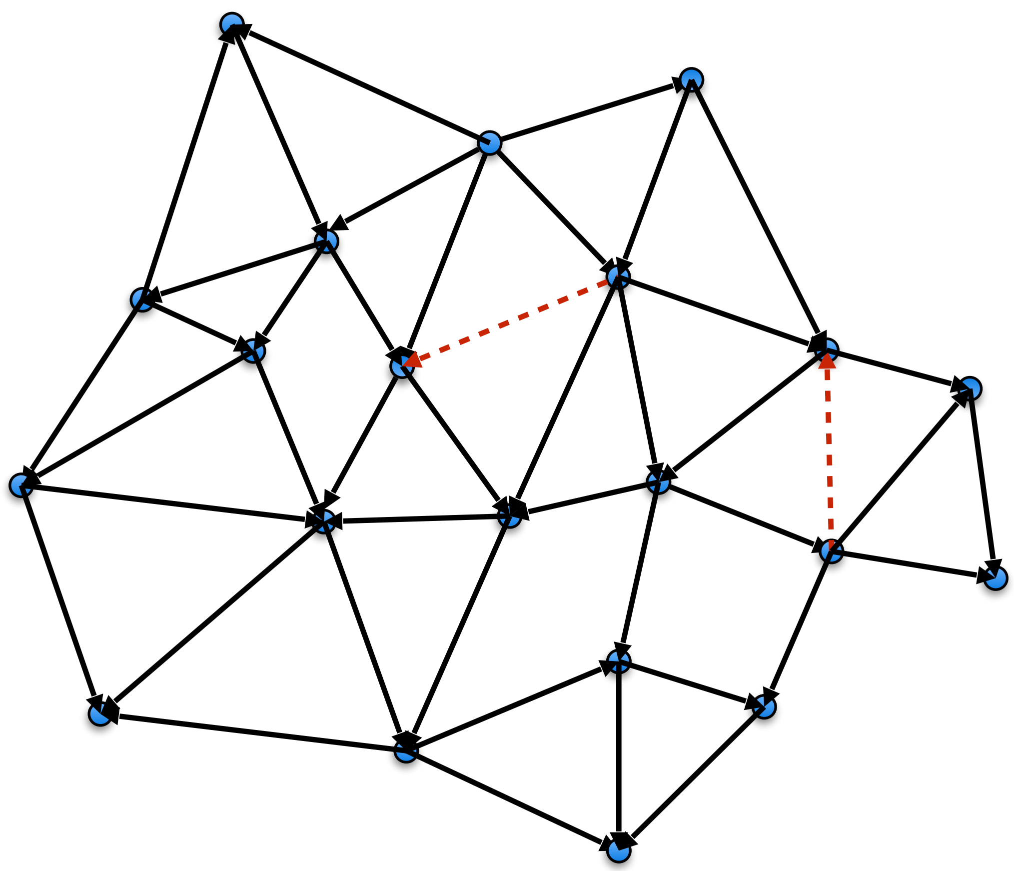
Fault Tolerant Model



$k = 2(\text{Faults})$

Fault Tolerant Model

G

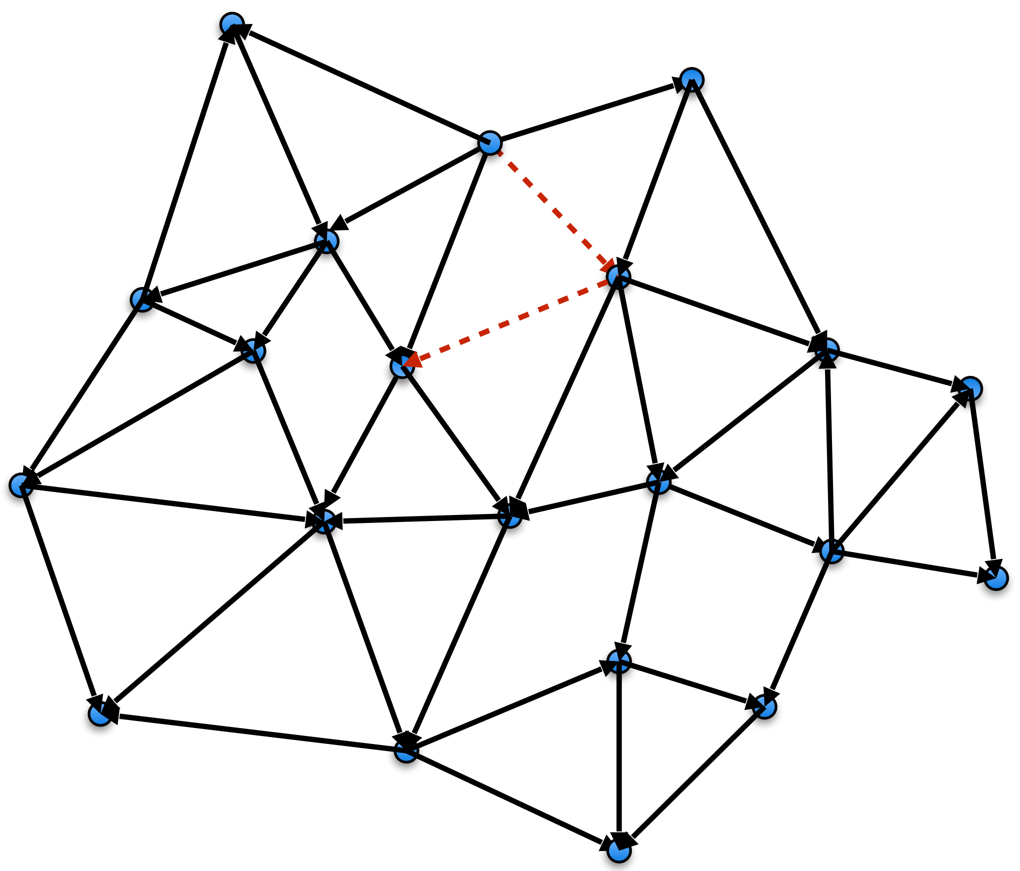


Time 2

$k = 2(\text{Faults})$

Fault Tolerant Model

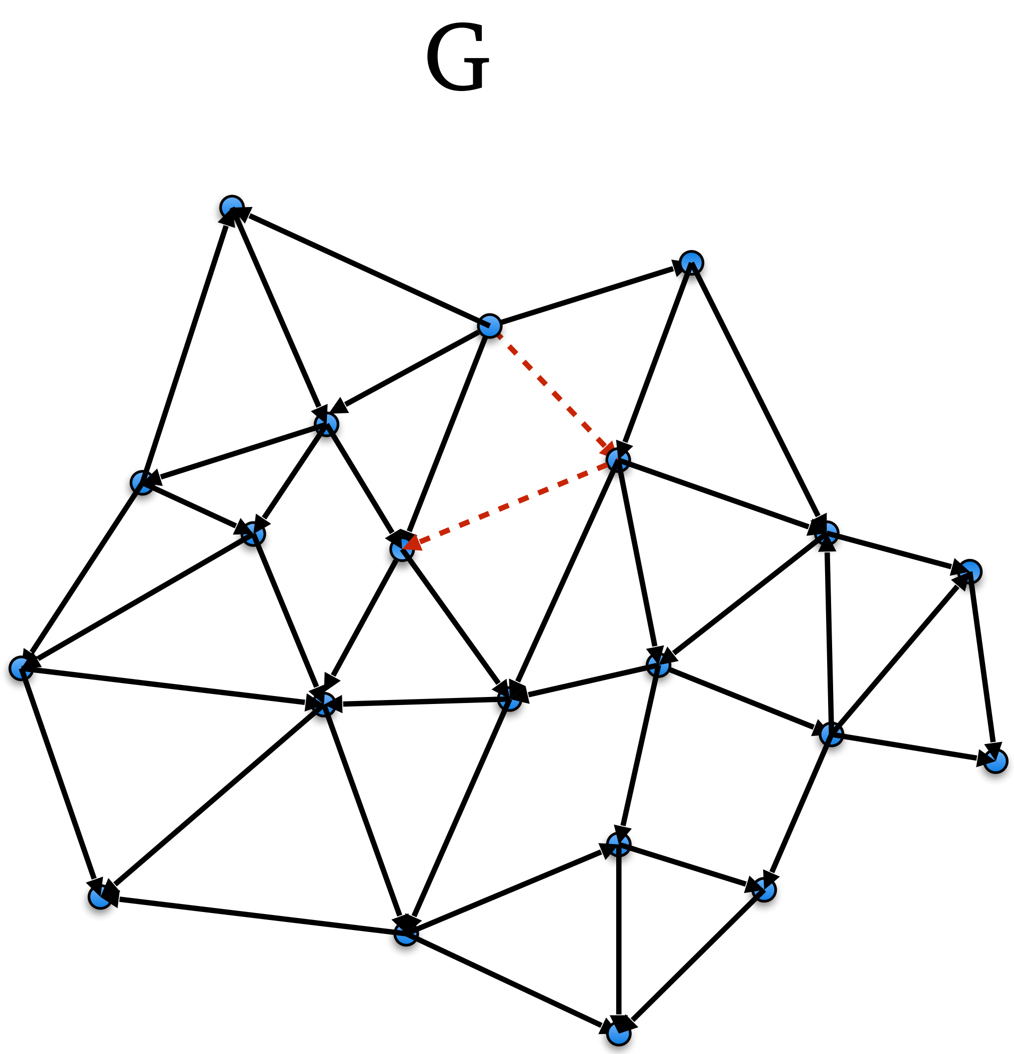
G



Time 3

$k = 2$(Faults)

Fault Tolerant Model



Time 3

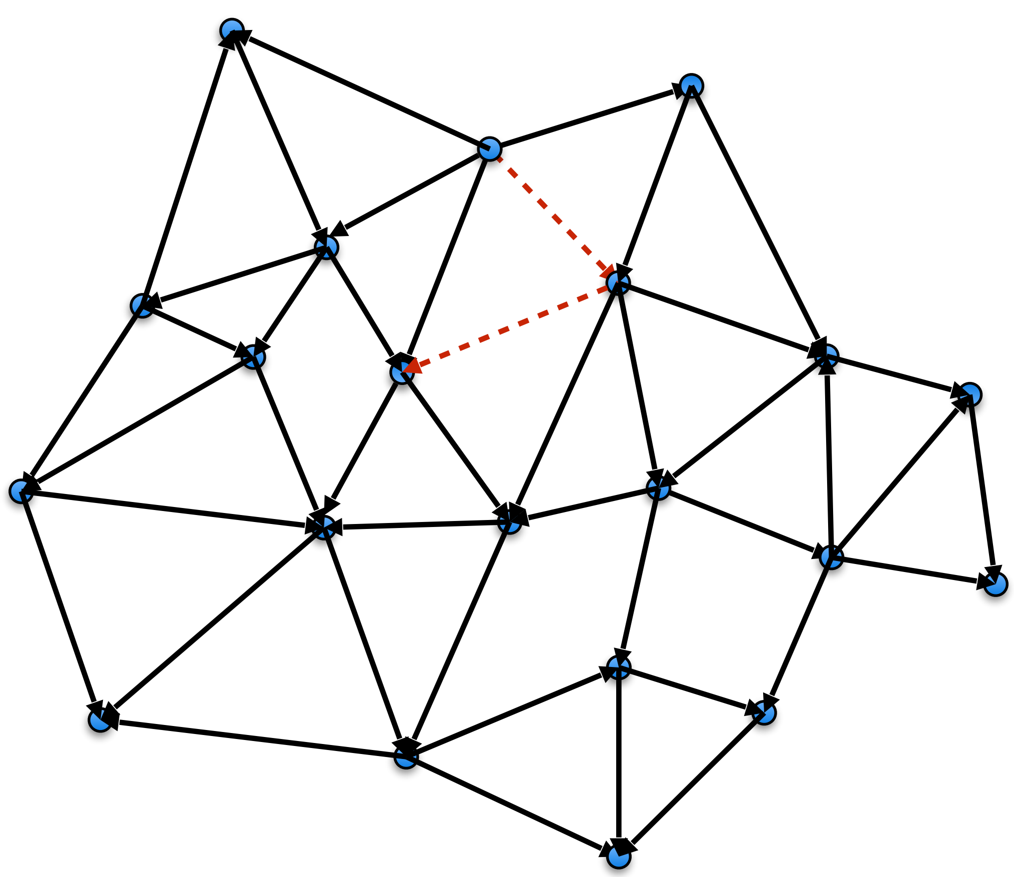
$k = 2(\text{Faults})$

Answer queries of the form:

- Exact/approximate distances
- Maximally Independent Set
- Minimum Spanning-tree

Fault Tolerant Model

G



Time 3

$k = 2(\text{Faults})$

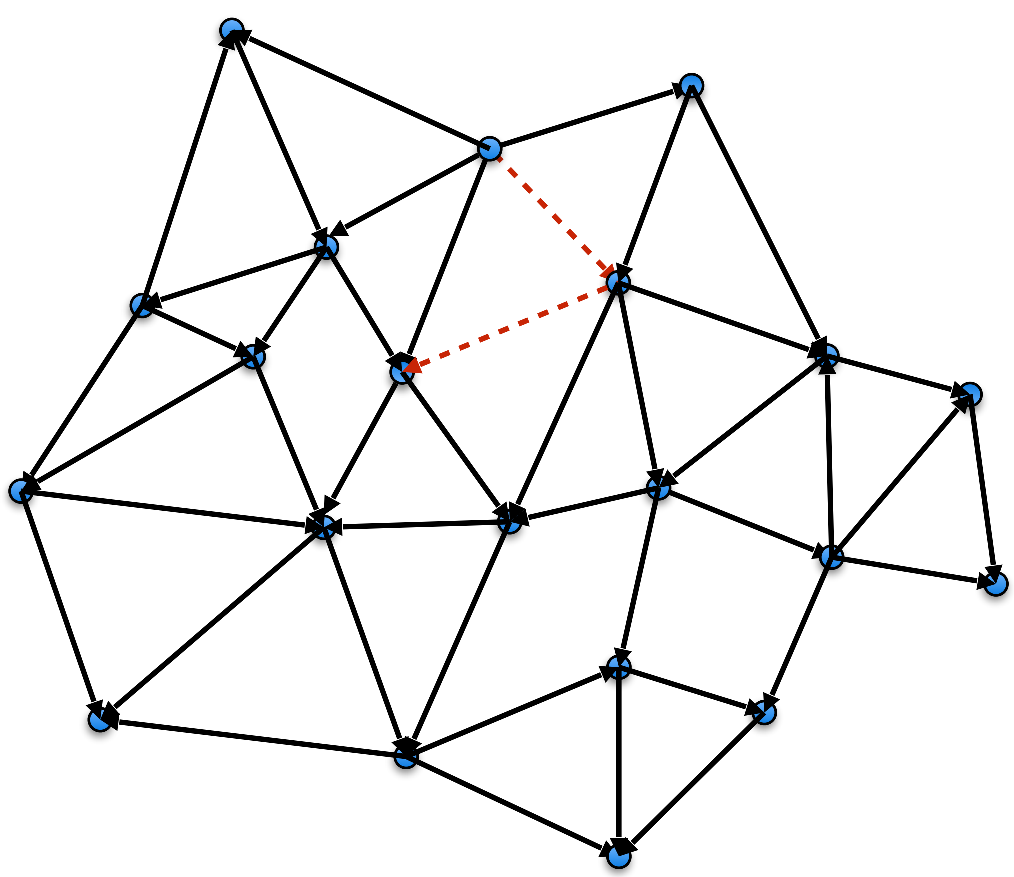
Answer queries of the form:

- Exact/approximate distances
- Maximally Independent Set
- Minimum Spanning-tree

Naive approach
Re-compute the solution
at each time.

Fault Tolerant Model

G



Time 3

$k = 2(\text{Faults})$

Answer queries of the form:
- Exact/approximate distances
- Maximally Independent Set
- Minimum Spanning-tree

Naive approach
Re-compute the solution
at each time.

O(m) at each step!

Fault Tolerant Model

Fault Tolerant Model

Why should we learn this model?

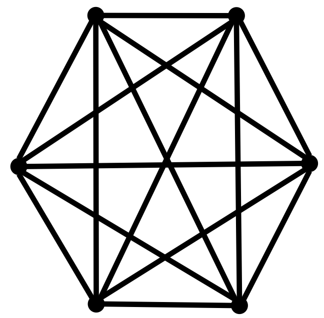
Fault Tolerant Model

Why should we learn this model?

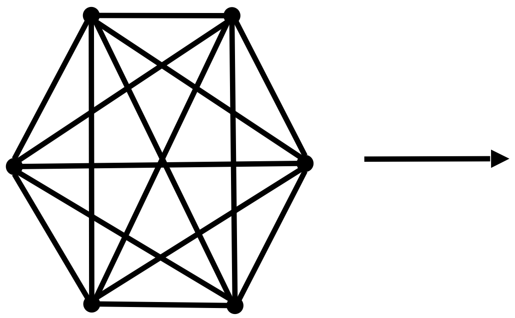
Dynamic graph algorithms models:

- Fully dynamic – An update is an edge insertion or deletion
- Decremental – An update is an edge deletion
- Incremental – An update is an edge insertion

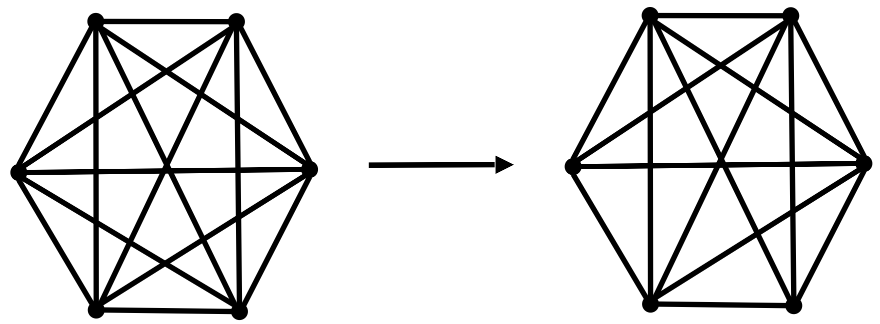
Dynamic Model: An Example



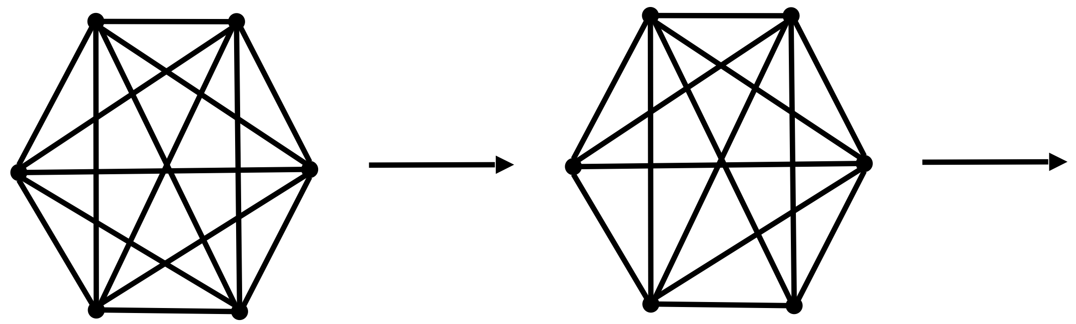
Dynamic Model: An Example



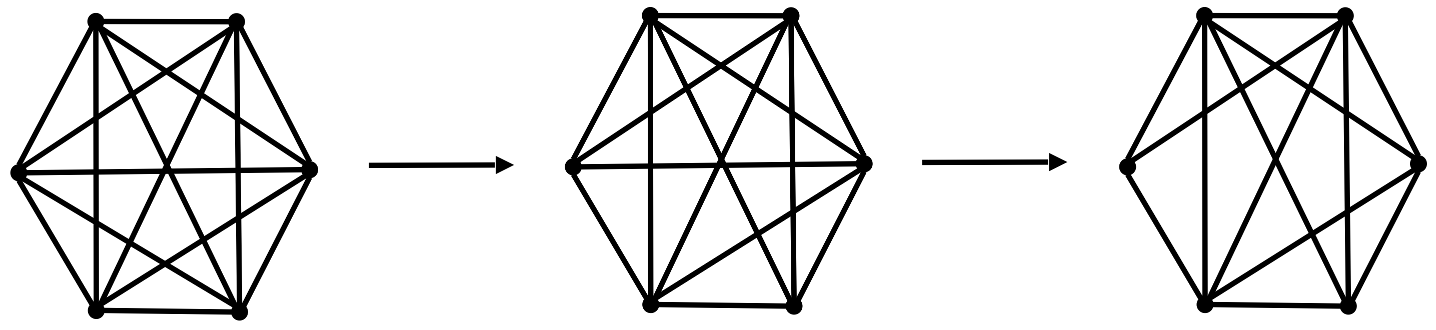
Dynamic Model: An Example



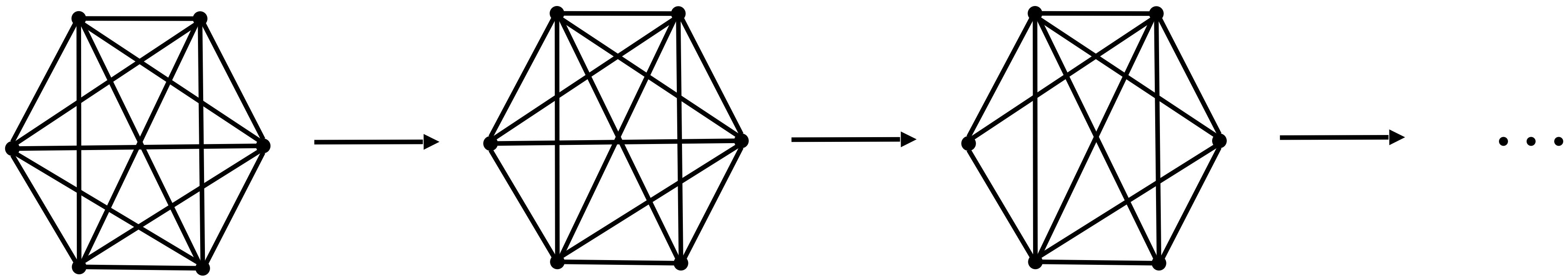
Dynamic Model: An Example



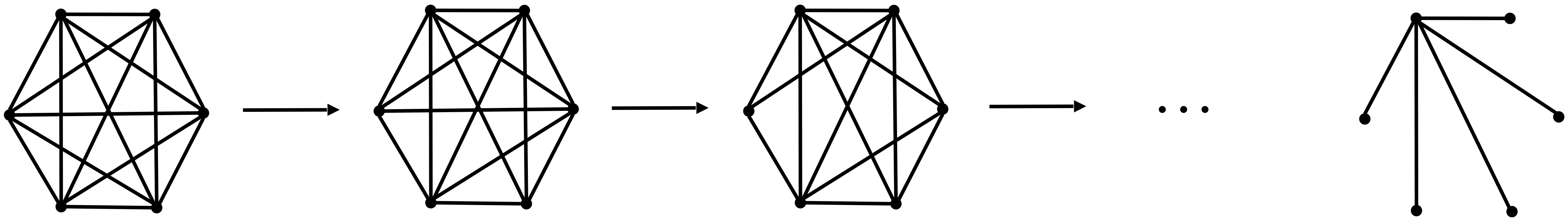
Dynamic Model: An Example



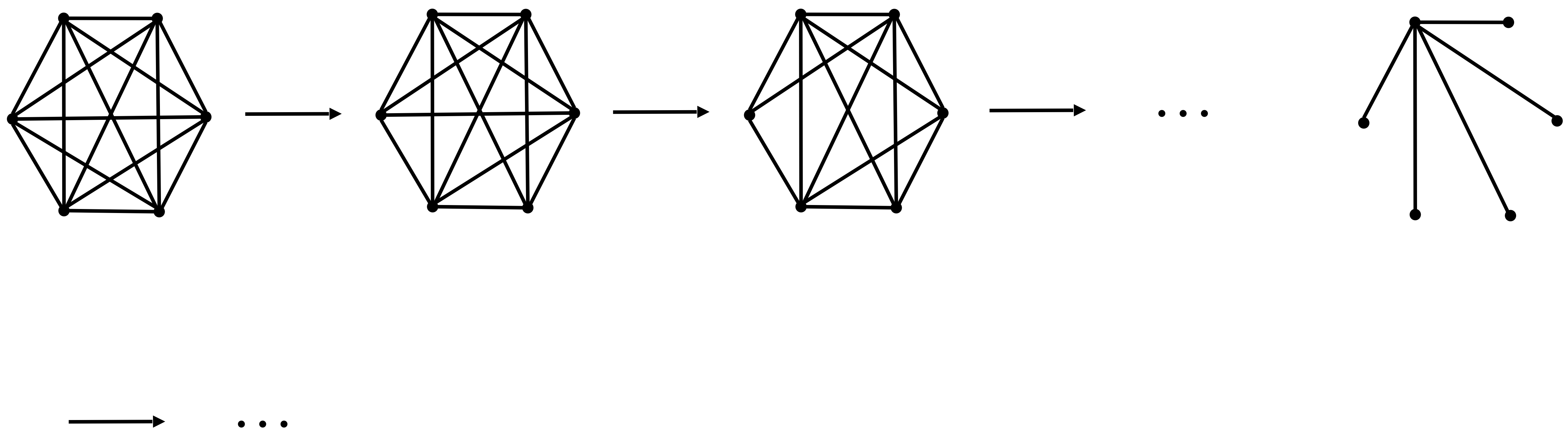
Dynamic Model: An Example



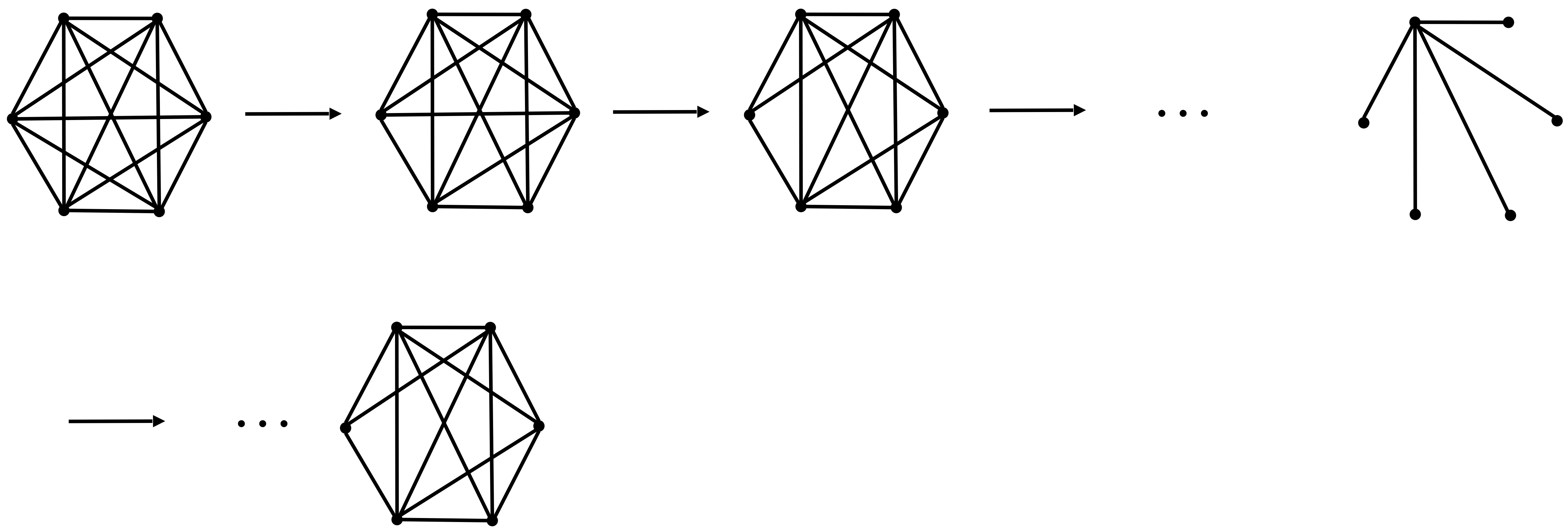
Dynamic Model: An Example



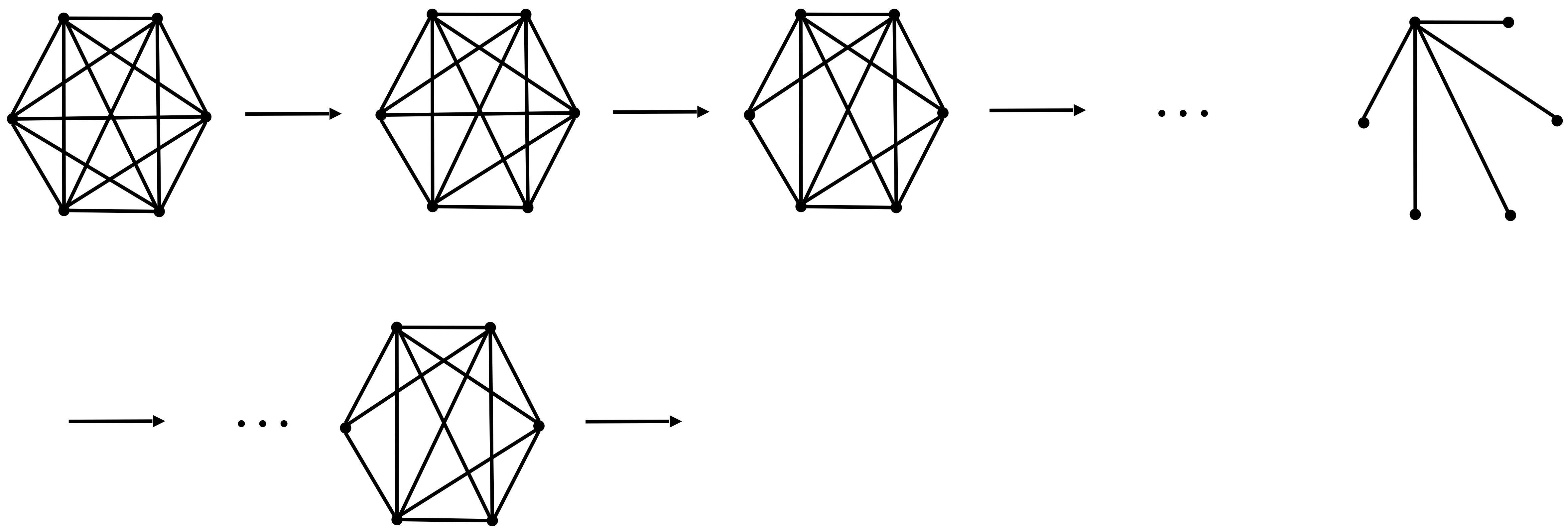
Dynamic Model: An Example



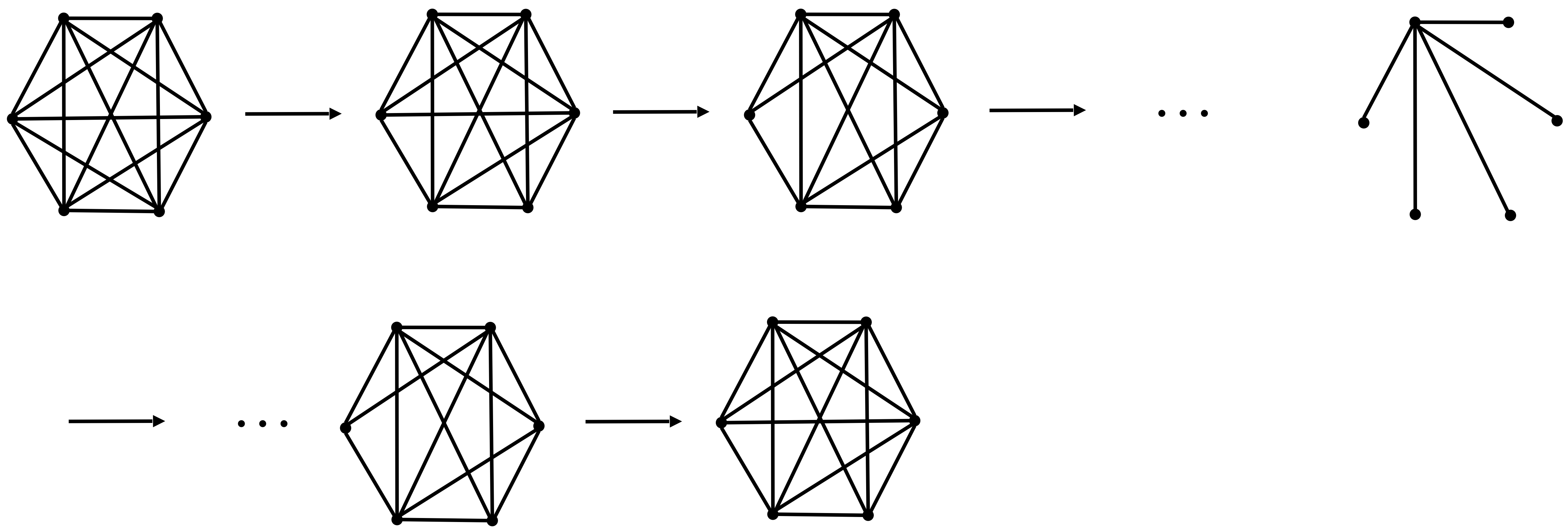
Dynamic Model: An Example



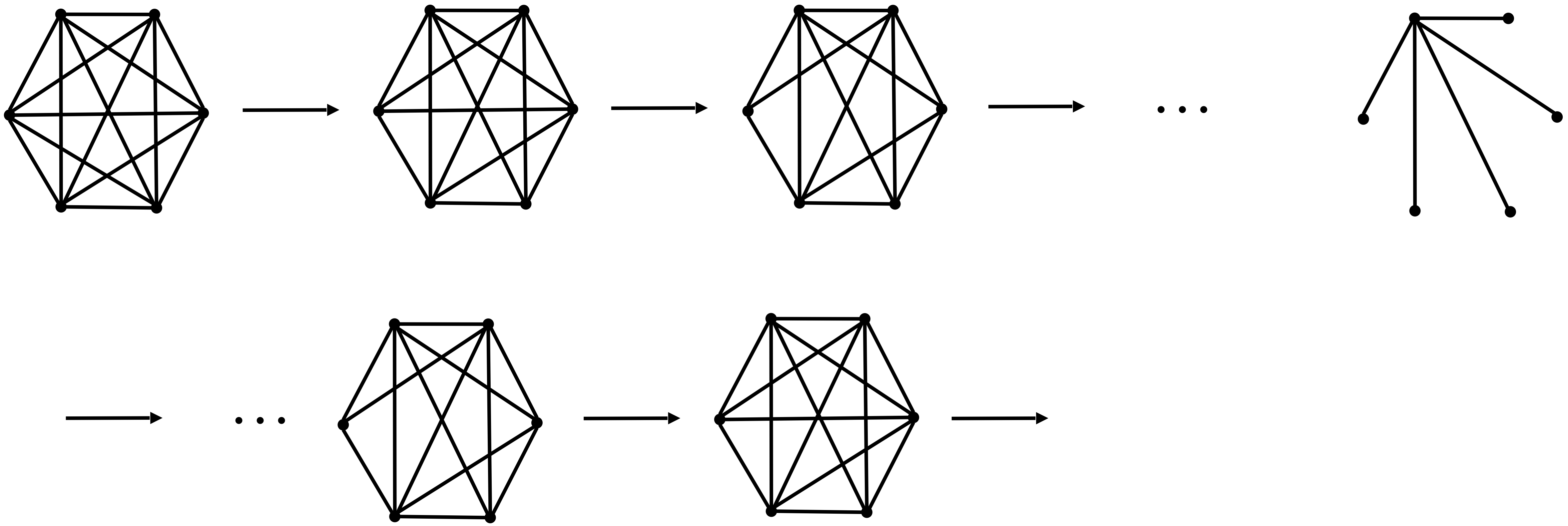
Dynamic Model: An Example



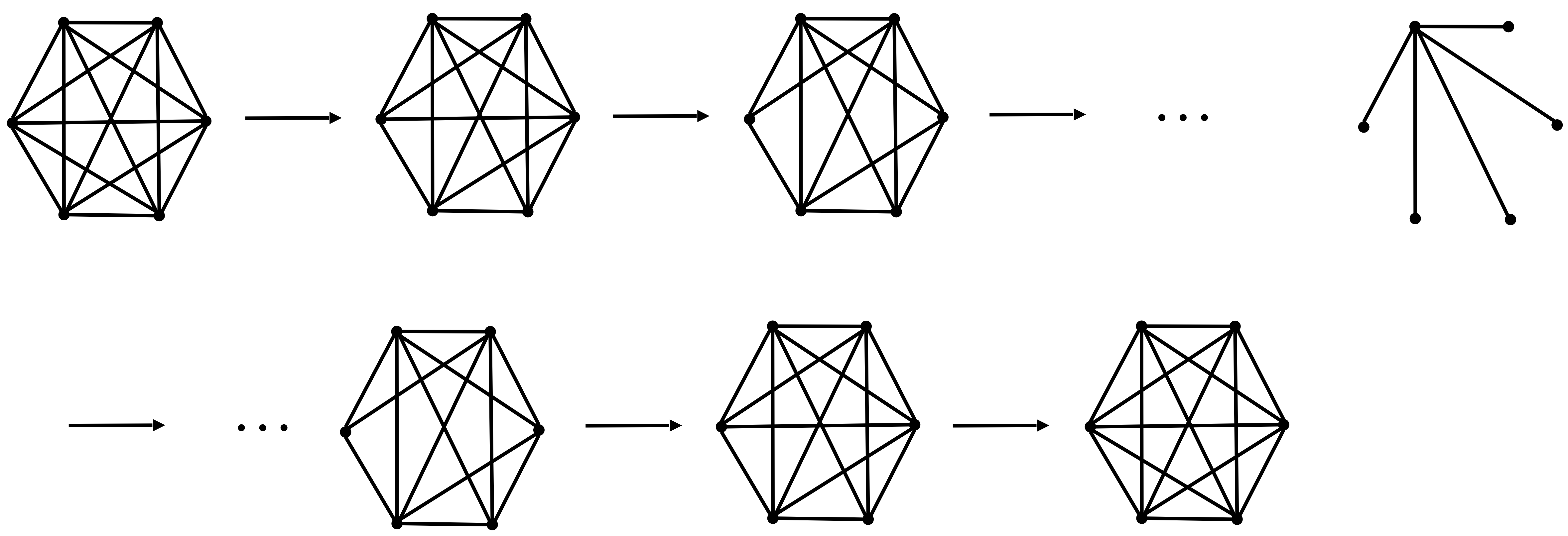
Dynamic Model: An Example



Dynamic Model: An Example



Dynamic Model: An Example



Fault Tolerant Model

Fully dynamic / Dec / Inc model

Fault Tolerant Model

Fully dynamic / Dec / Inc model

Too general

Fault Tolerant Model

Fully dynamic / Dec / Inc model

Too general

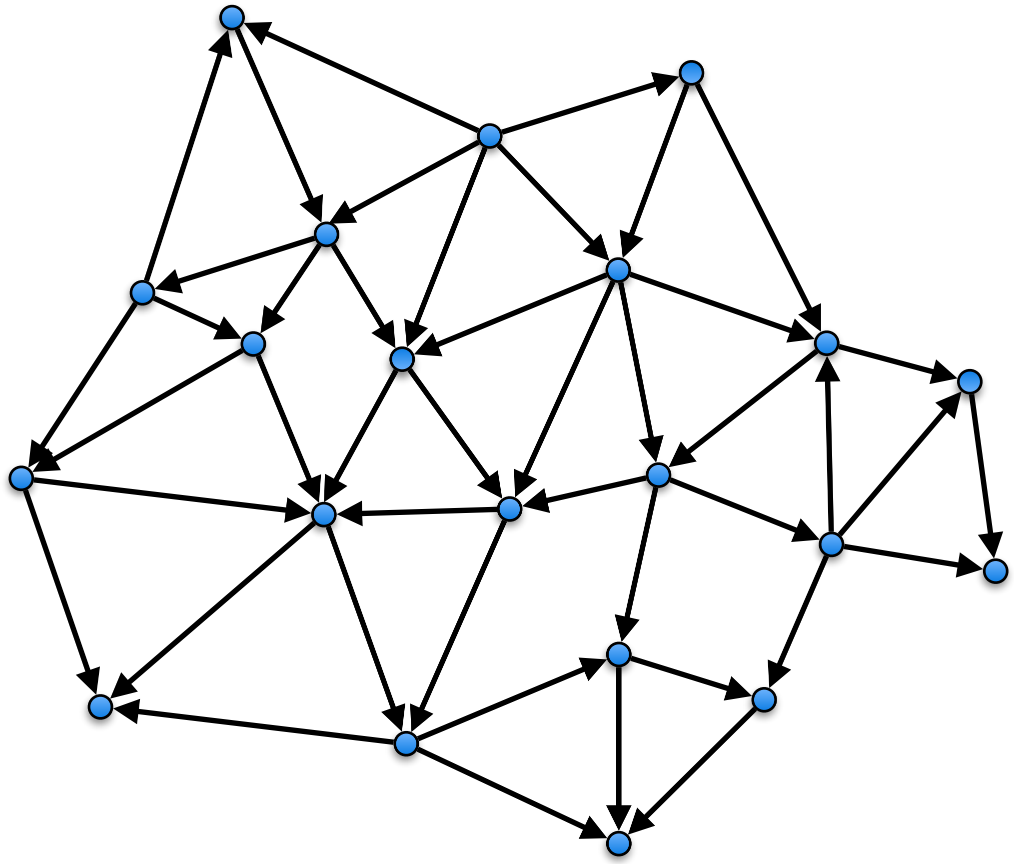
In many real world networks changes are very limited and transient

Road networks, communication networks etc.

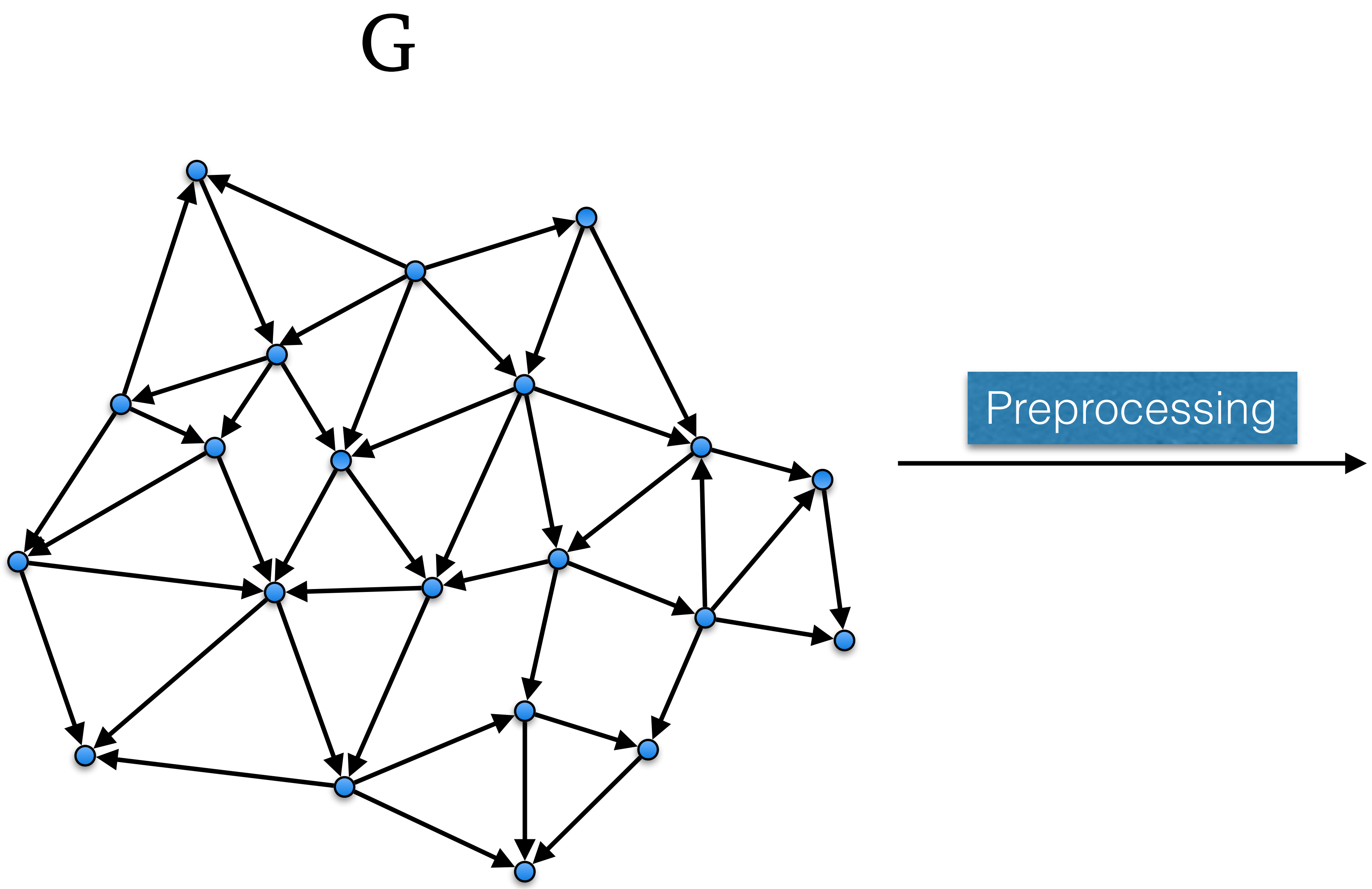
Fault Tolerant Oracle

Fault Tolerant Oracle

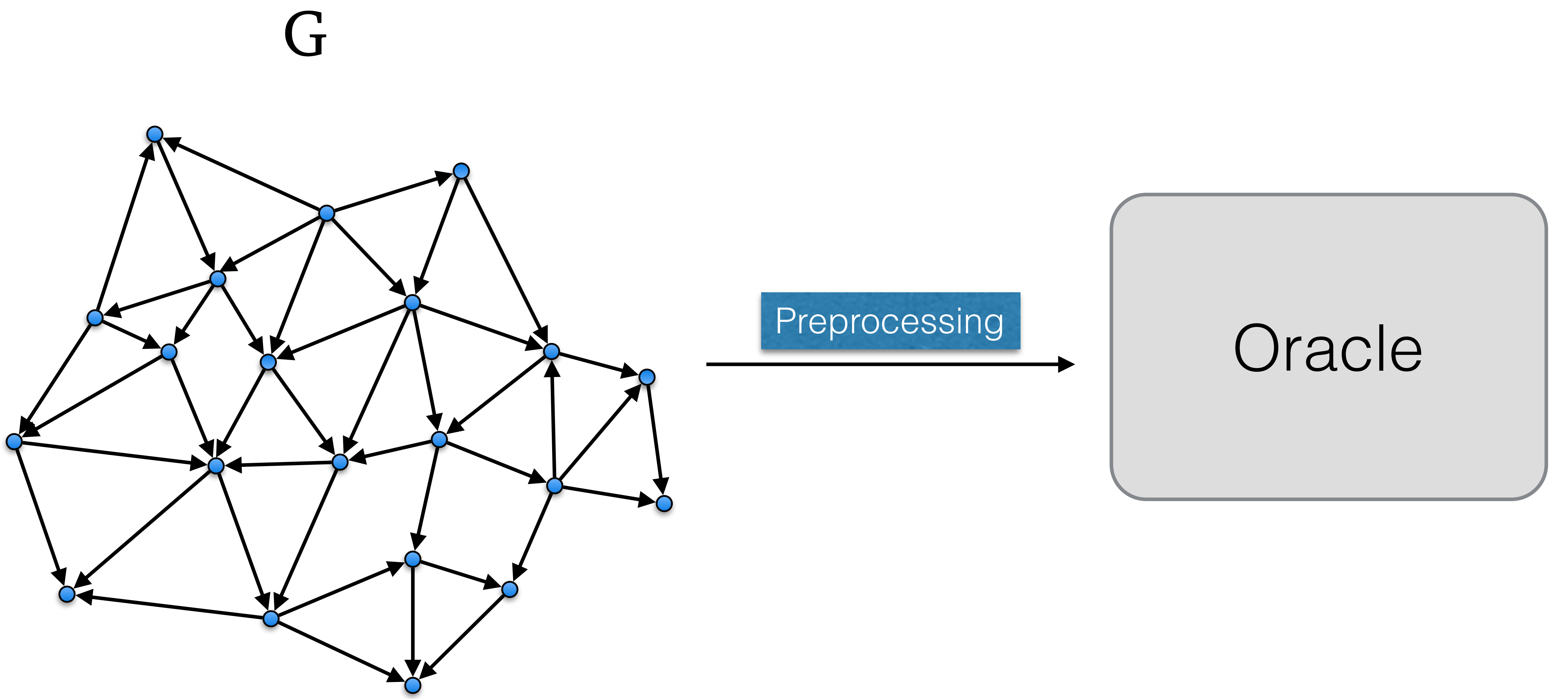
G



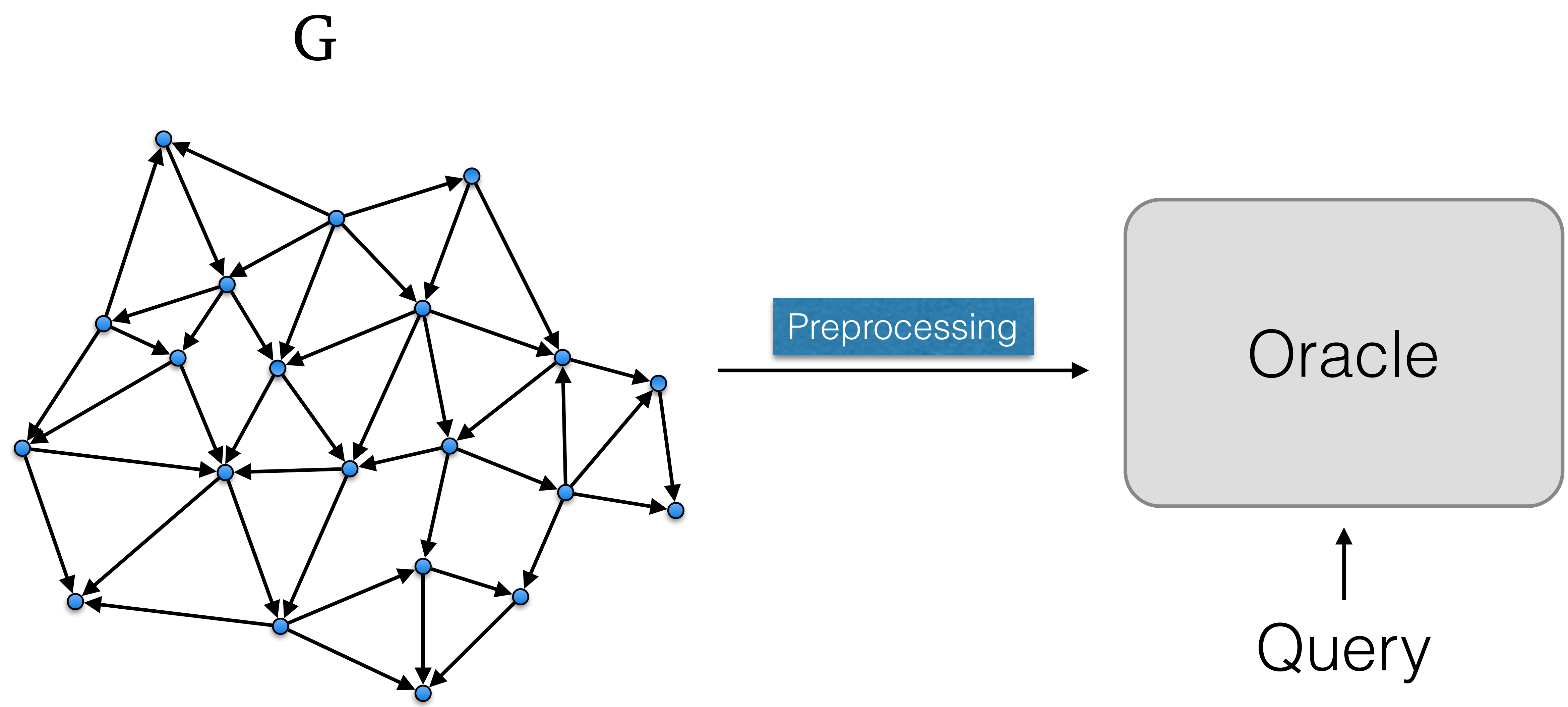
Fault Tolerant Oracle



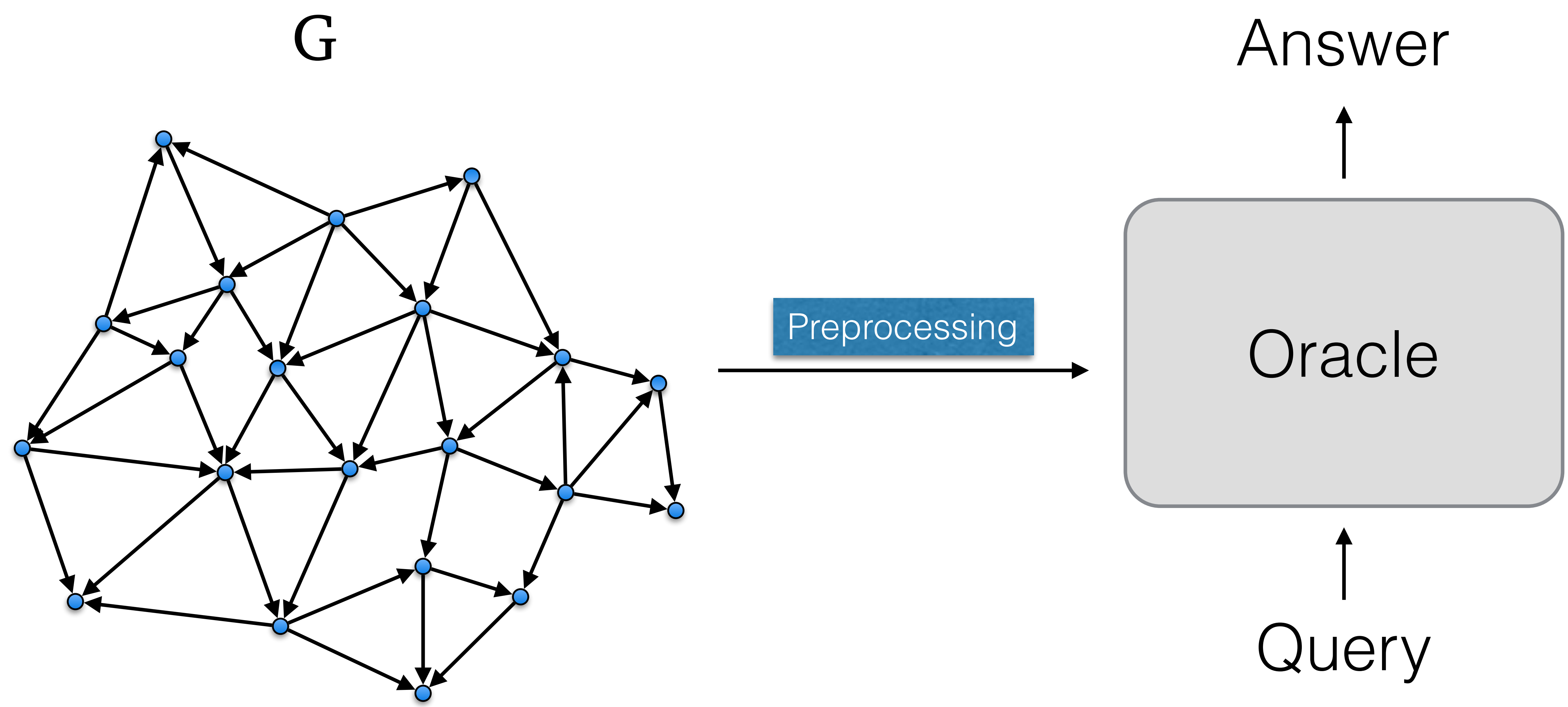
Fault Tolerant Oracle



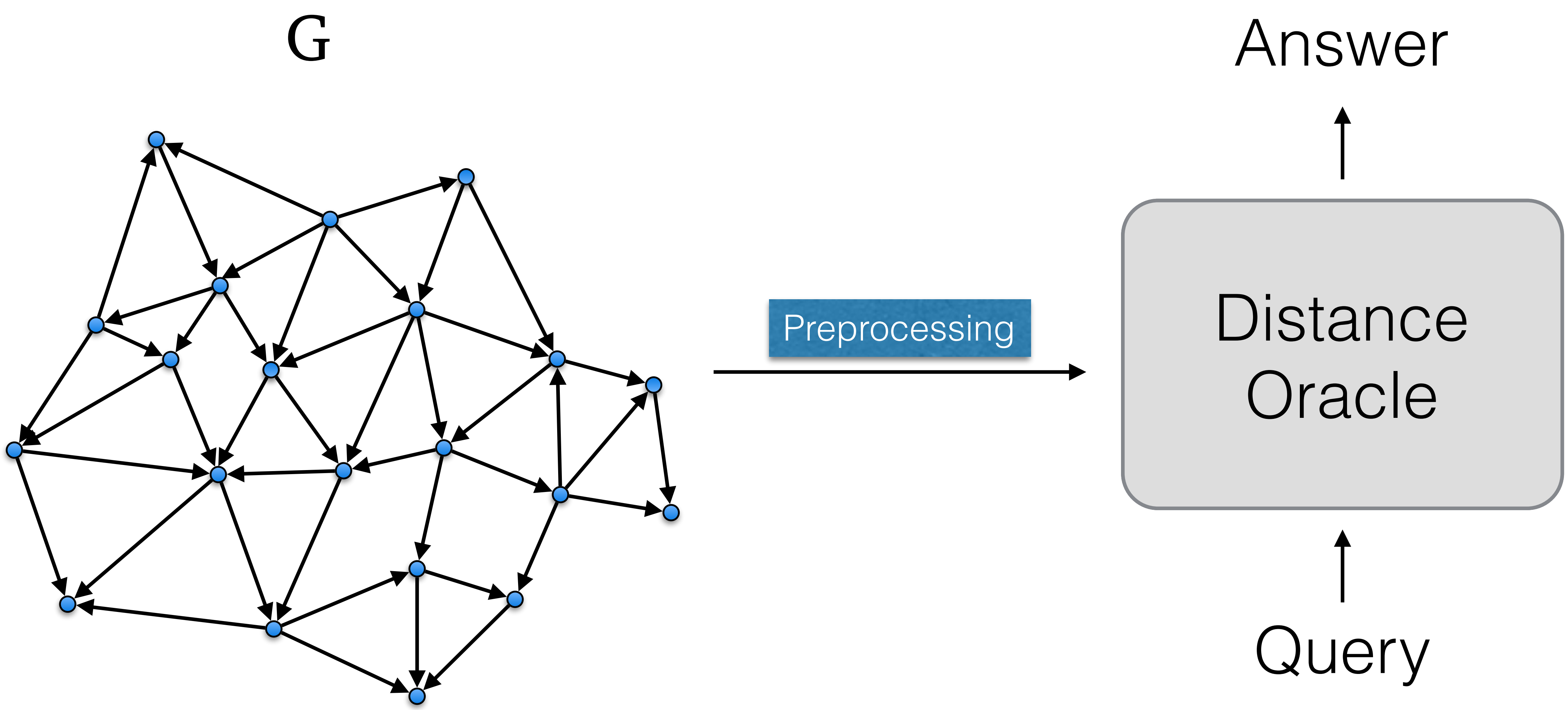
Fault Tolerant Oracle



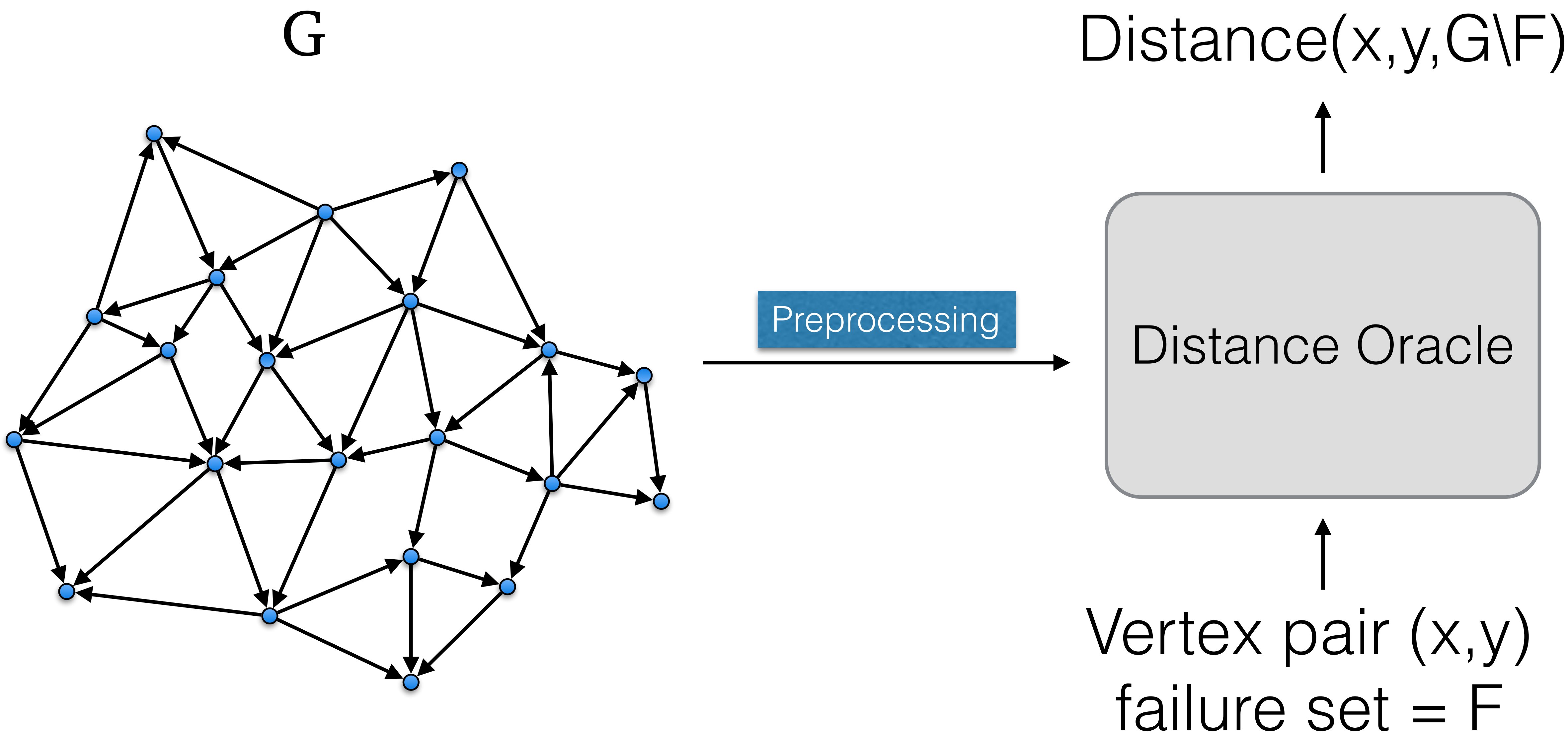
Fault Tolerant Oracle



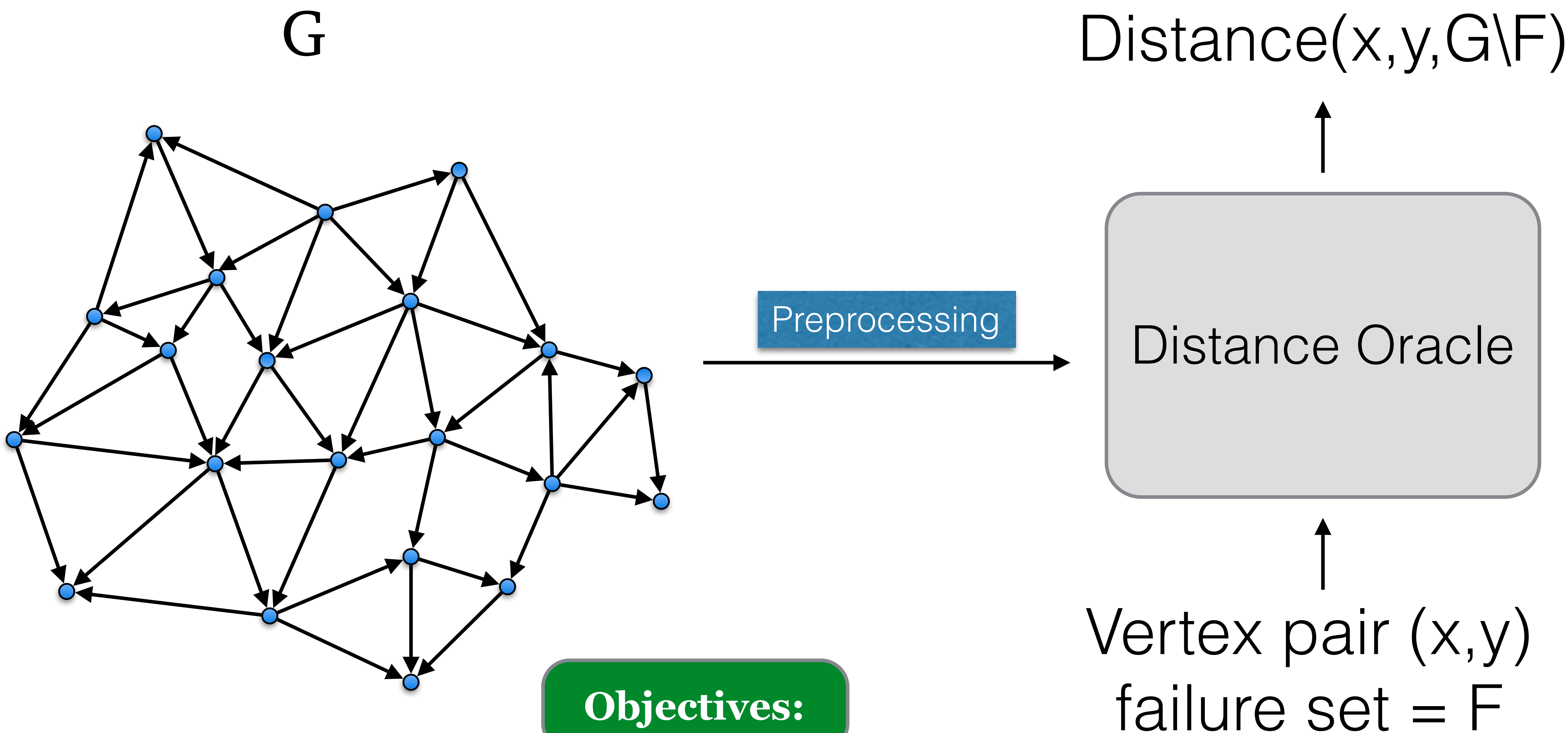
Fault Tolerant Oracle



Fault Tolerant Oracle

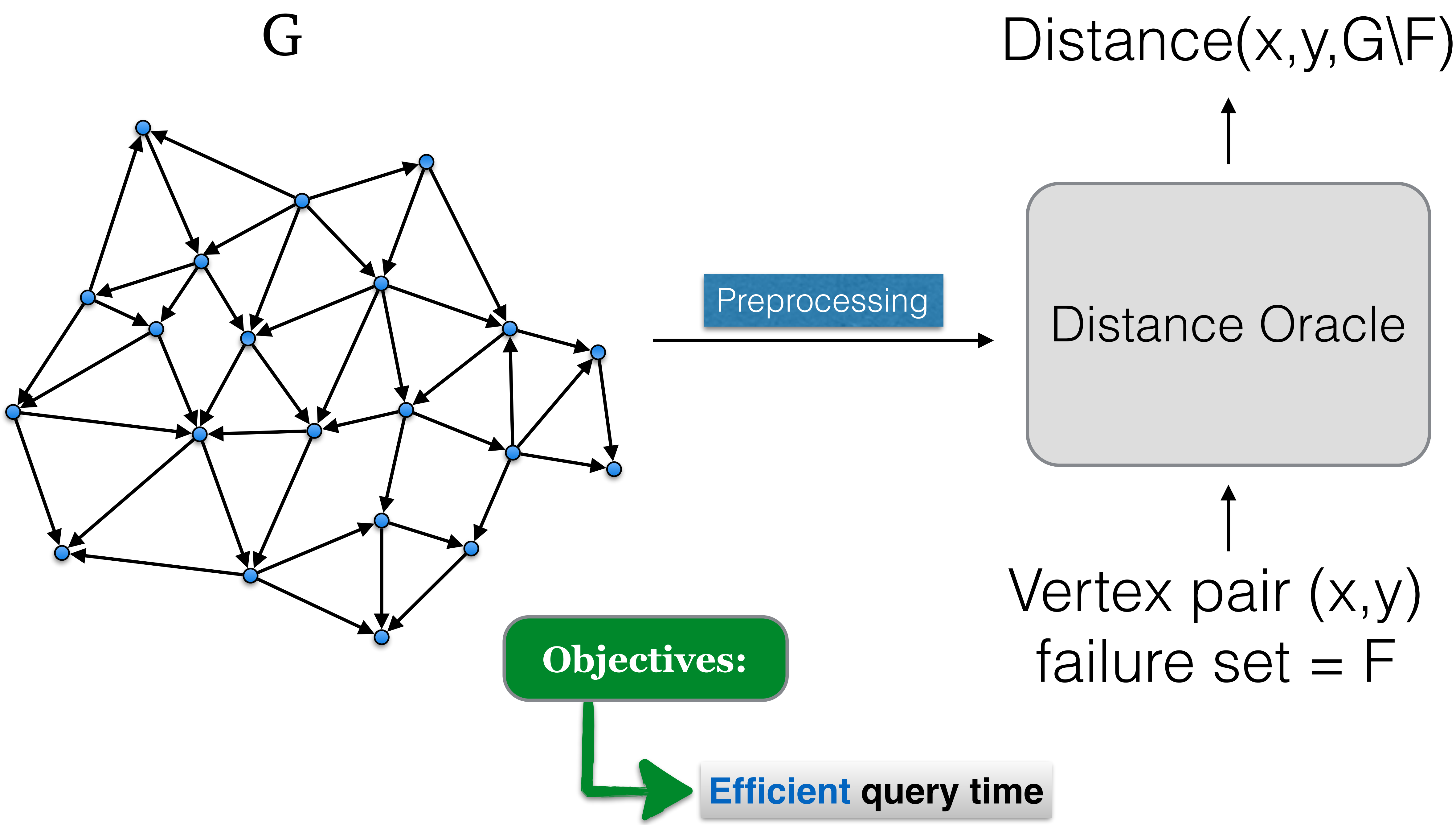


Fault Tolerant Oracle

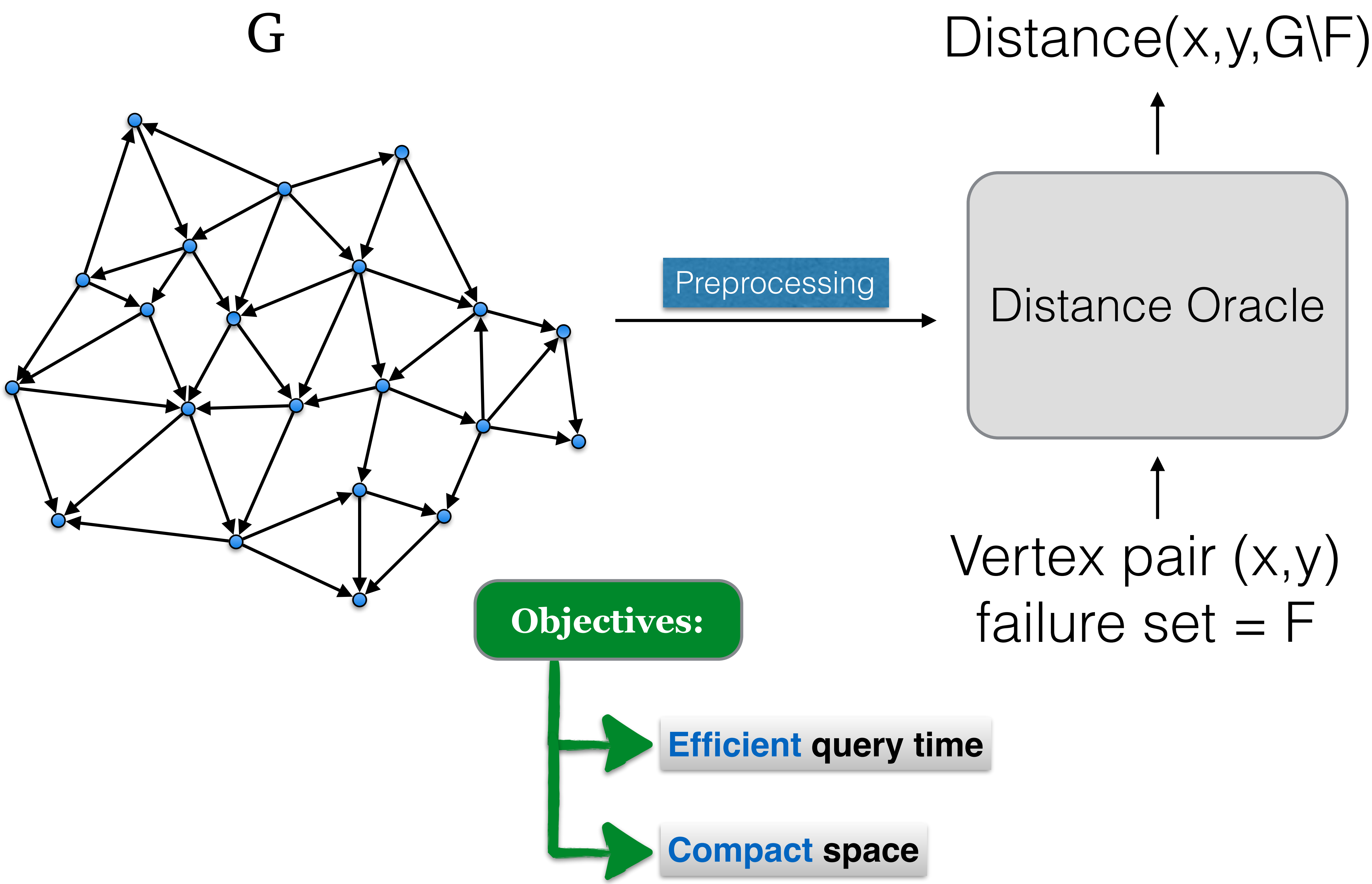


Objectives:

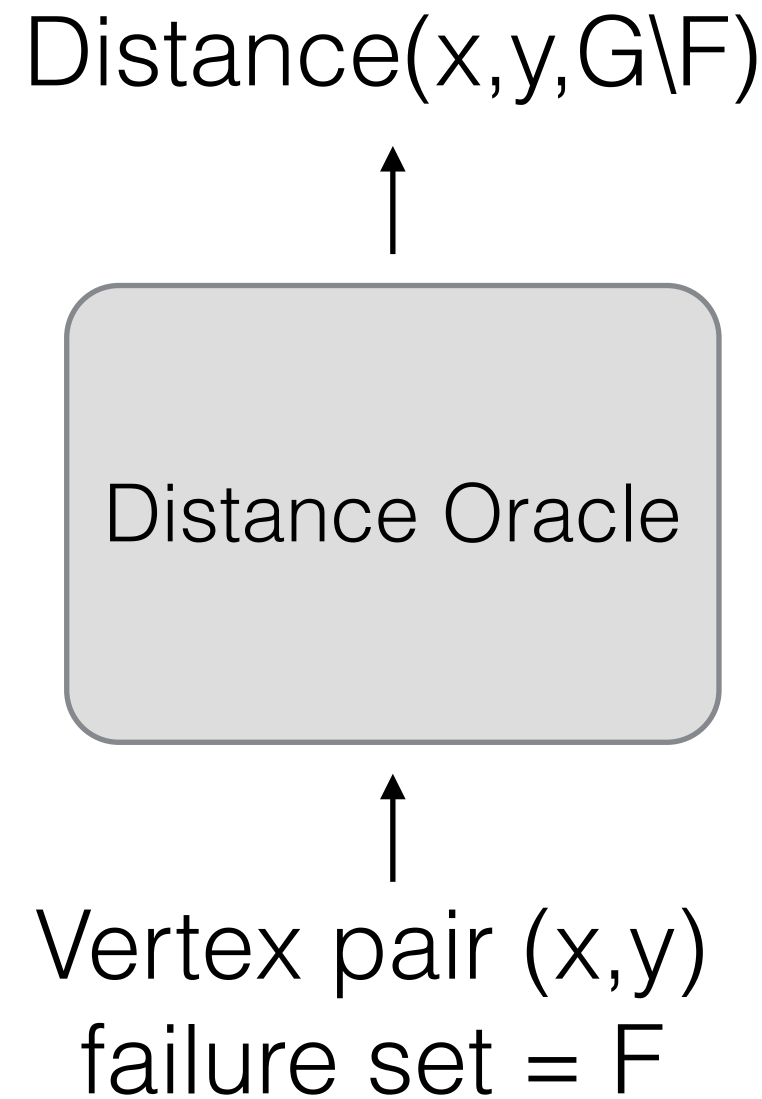
Fault Tolerant Oracle



Fault Tolerant Oracle

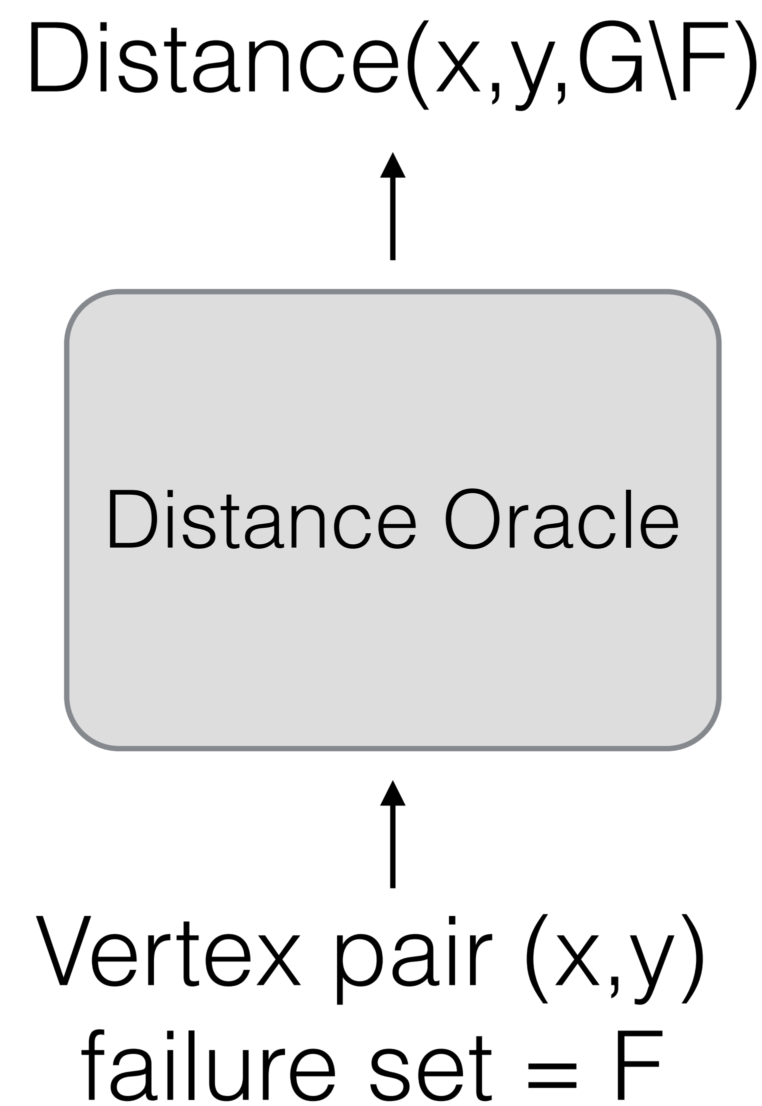


Fault Tolerant Oracle



Fault Tolerant Oracle

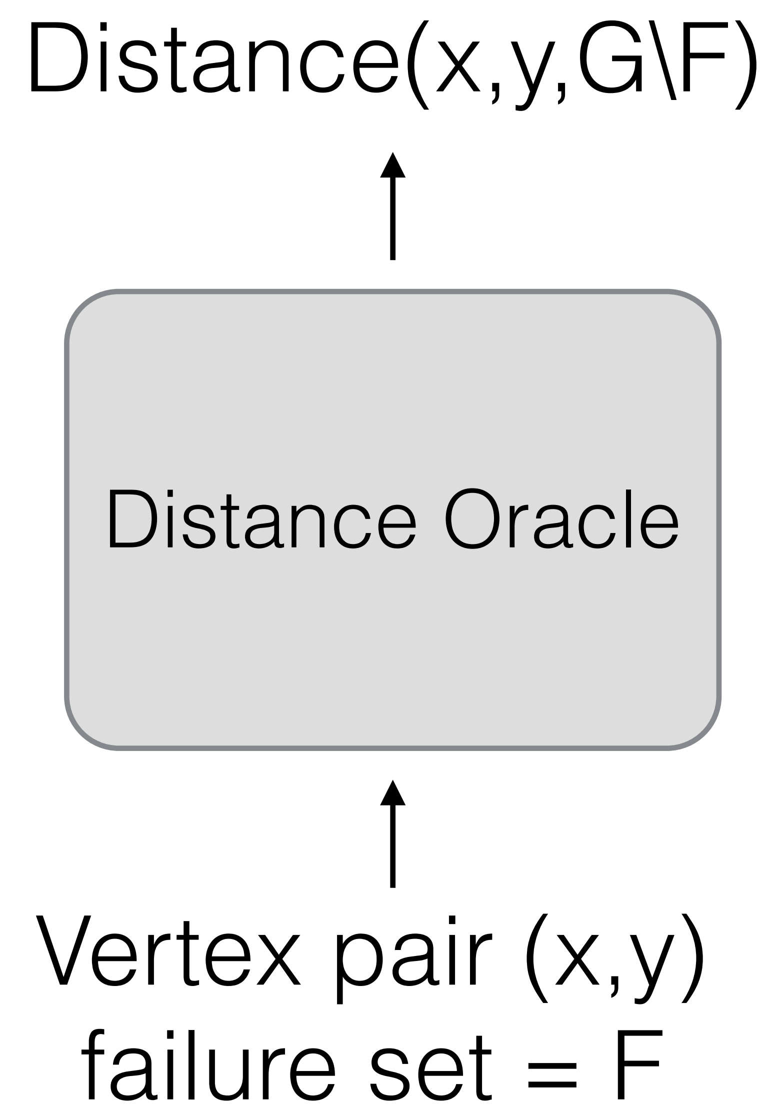
Trivial Solutions:



Fault Tolerant Oracle

Trivial Solutions:

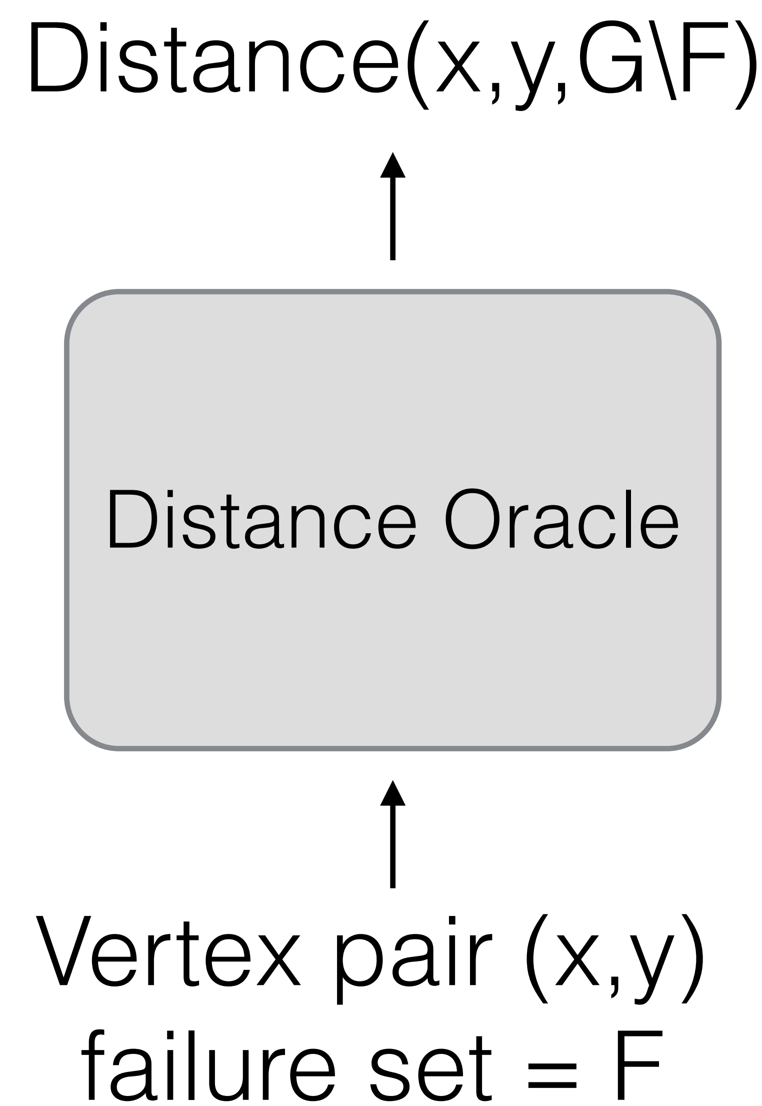
Compute & Store ALL solutions	*Store only graph G*



Fault Tolerant Oracle

Trivial Solutions:

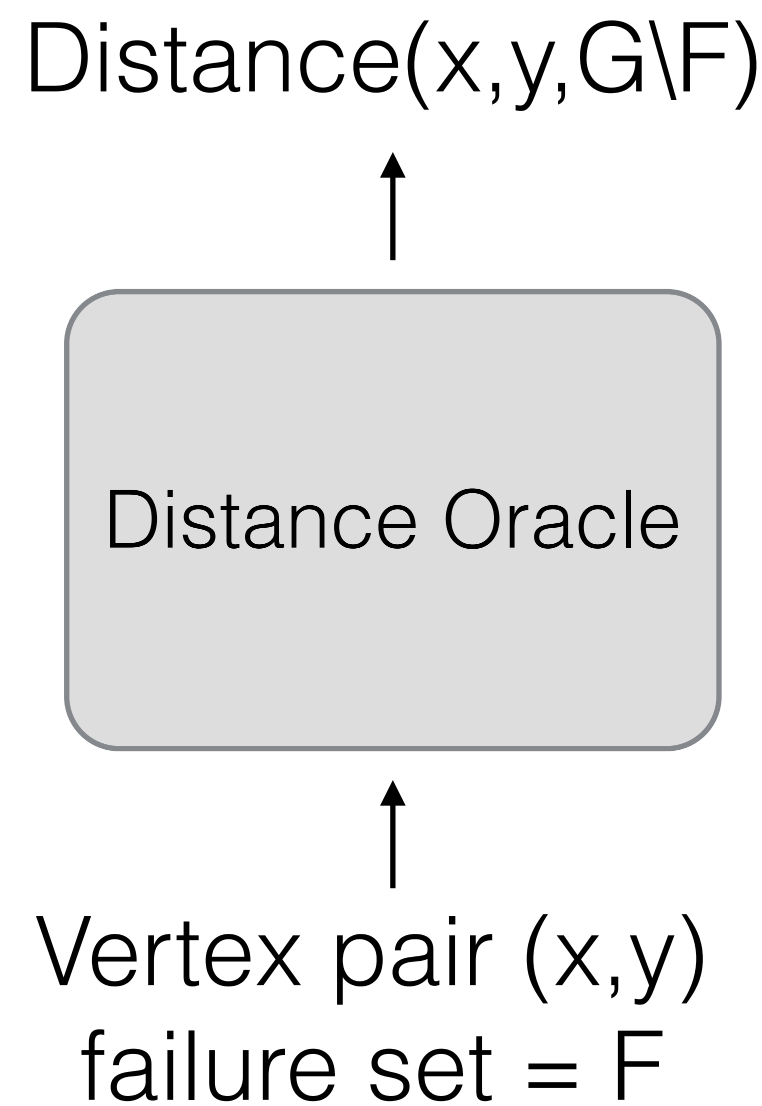
Compute & Store ALL solutions	*Store only graph G*
Space = $O({}^{n}C_{k} \cdot n^{2})$ Time = $O(1)$	



Fault Tolerant Oracle

Trivial Solutions:

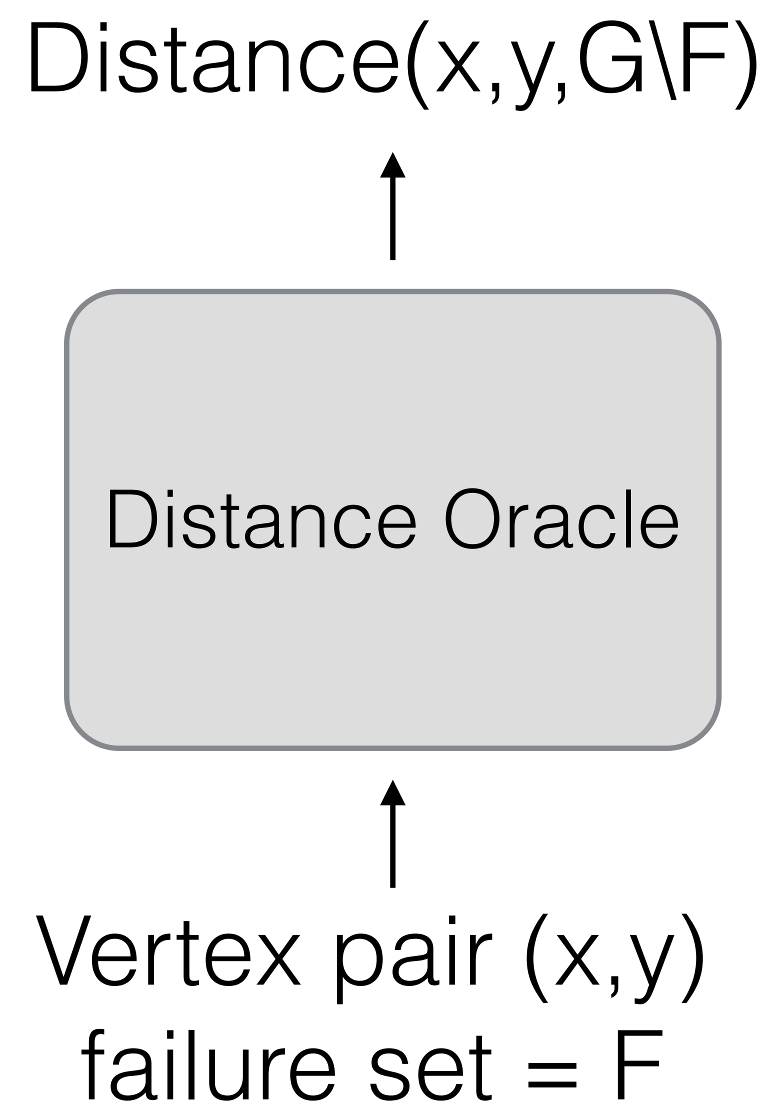
Compute & Store ALL solutions	*Store only graph G*
Space = $O({}^{n}C_{k} \cdot n^2)$	Space = $O(m+n)$
Time = $O(1)$	Time = $O(m+n)$



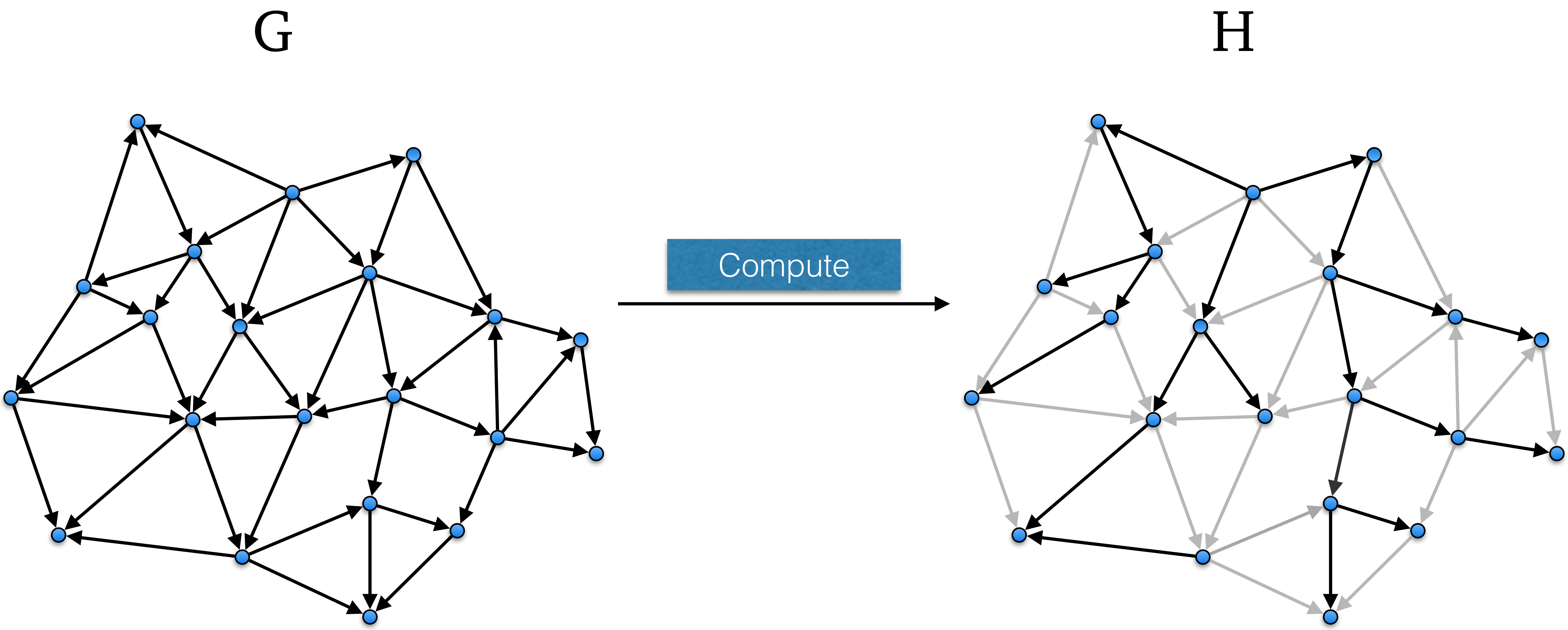
Fault Tolerant Oracle

Trivial Solutions:

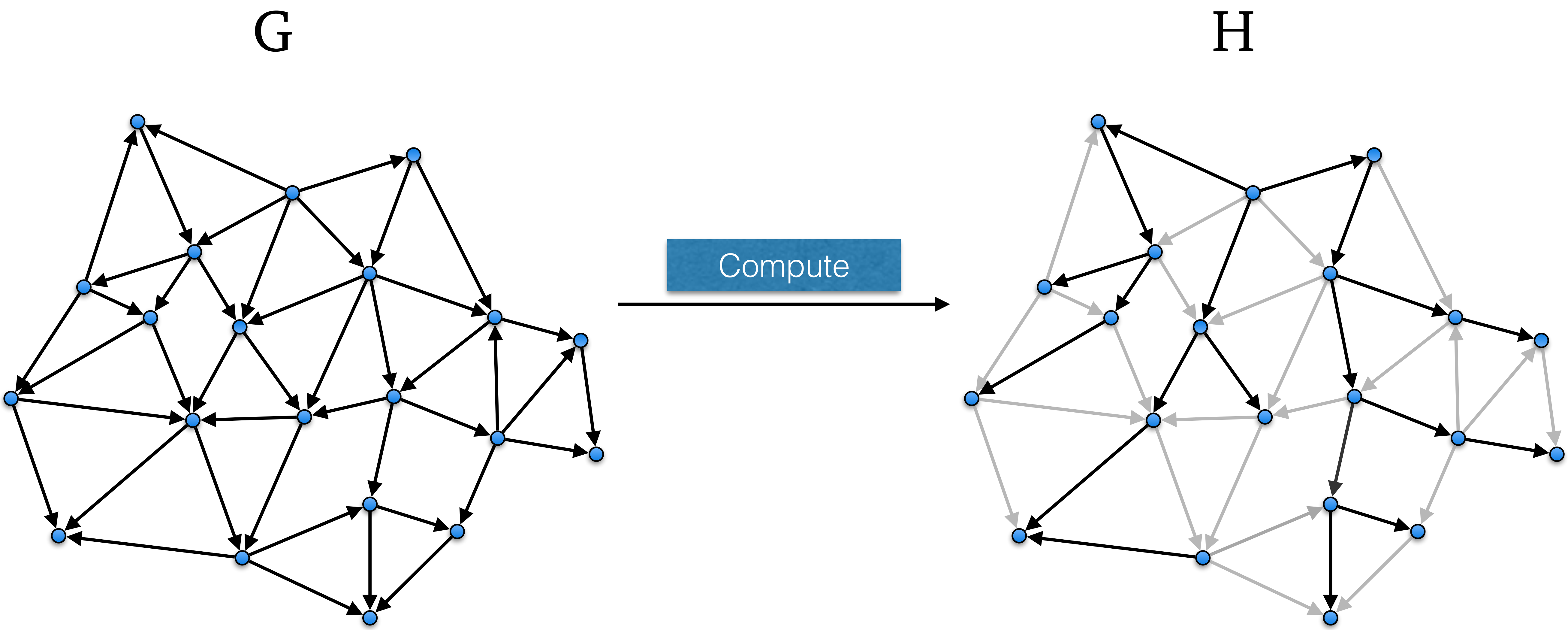
Compute & Store ALL solutions	***Store only graph G***
Space = $O({}^{n}C_{k} \cdot n^2)$	Space = $O(m+n)$
Time = $O(1)$	Time = $O(m+n)$



Fault Tolerant Preservers



Fault Tolerant Preservers



$H \setminus F$ **preserves** a "pre-specified property" of $G \setminus F$,

for all possible F, $|F| \leqslant k$

Previous Works

Many works in the recent decade (partial list):

Previous Works

Many works in the recent decade (partial list):

Demetrescu, Thorup, Chowdhury, Ramachandran (SICOMP 2008) APSP – one fault

Previous Works

Many works in the recent decade (partial list):

Demetrescu, Thorup, Chowdhury, Ramachandran (SICOMP 2008) APSP – one fault

Bernstein, Karger (SODA'08, STOC'09) – Improved running time

Previous Works

Many works in the recent decade (partial list):

- Demetrescu, Thorup, Chowdhury, Ramachandran (SICOMP 2008) APSP – one fault
- Bernstein, Karger (SODA’08, STOC’09) – Improved running time
- Patrascu, Thorup, (FOCS’07) – Connectivity

Previous Works

Many works in the recent decade (partial list):

Demetrescu, Thorup, Chowdhury, Ramachandran (SICOMP 2008) APSP – one fault

Bernstein, Karger (SODA'08, STOC'09) – Improved running time

Patrascu, Thorup, (FOCS'07) – Connectivity

Chechik (Inf. Comp 2013) – Compact routing schemes

Previous Works

Many works in the recent decade (partial list):

Demetrescu, Thorup, Chowdhury, Ramachandran (SICOMP 2008) APSP – one fault

Bernstein, Karger (SODA'08, STOC'09) – Improved running time

Patrascu, Thorup, (FOCS'07) – Connectivity

Chechik (Inf. Comp 2013) – Compact routing schemes

Dinitz, Krauthgamer: (PODC'11) – Spanners

Previous Works

Many works in the recent decade (partial list):

Demetrescu, Thorup, Chowdhury, Ramachandran (SICOMP 2008) APSP – one fault

Bernstein, Karger (SODA’08, STOC’09) – Improved running time

Patrascu, Thorup, (FOCS’07) – Connectivity

Chechik (Inf. Comp 2013) – Compact routing schemes

Dinitz, Krauthgamer: (PODC’11) – Spanners

Georgiadis, Italiano, Parotsidis: (SODA’17) – Strong connectivity – one fault

Previous Works

Many works in the recent decade (partial list):

Demetrescu, Thorup, Chowdhury, Ramachandran (SICOMP 2008) APSP – one fault

Bernstein, Karger (SODA'08, STOC'09) – Improved running time

Patrascu, Thorup, (FOCS'07) – Connectivity

Chechik (Inf. Comp 2013) – Compact routing schemes

Dinitz, Krauthgamer: (PODC'11) – Spanners

Georgiadis, Italiano, Parotsidis: (SODA'17) – Strong connectivity – one fault

Bodwin, Grandoni, Parter, V. William: (ICALP'17) – Distances

This Talk

Problems of Reachability and strong-connectivity:

This Talk

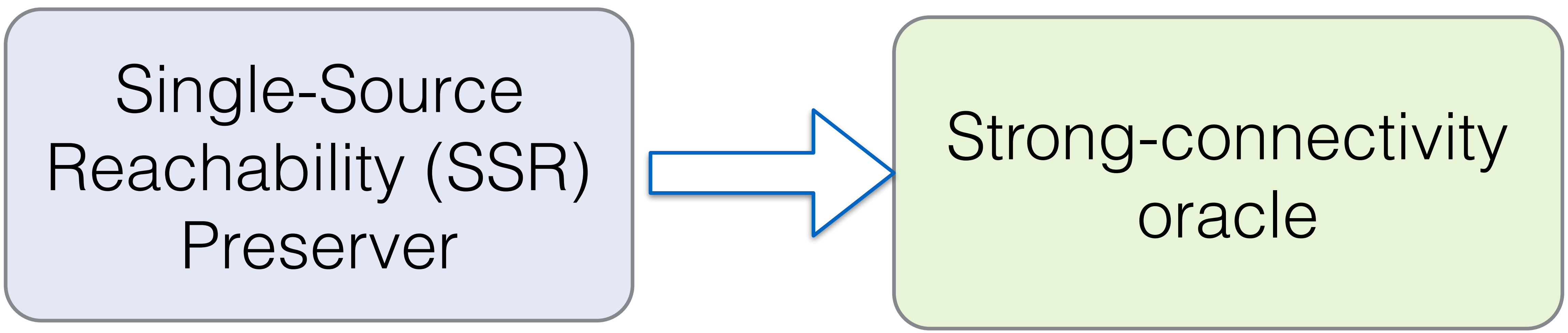
Problems of Reachability and strong-connectivity:

Single-Source Reachability (SSR) Preserver

Problem 1

This Talk

Problems of Reachability and strong-connectivity:



Problem 1

Problem 2

Our Contributions

Problem 1: Reachability Preserver

Problem 1: Reachability Preserver

Input: directed graph $G=(V,E)$, parameter k, and a source s.

Problem 1: Reachability Preserver

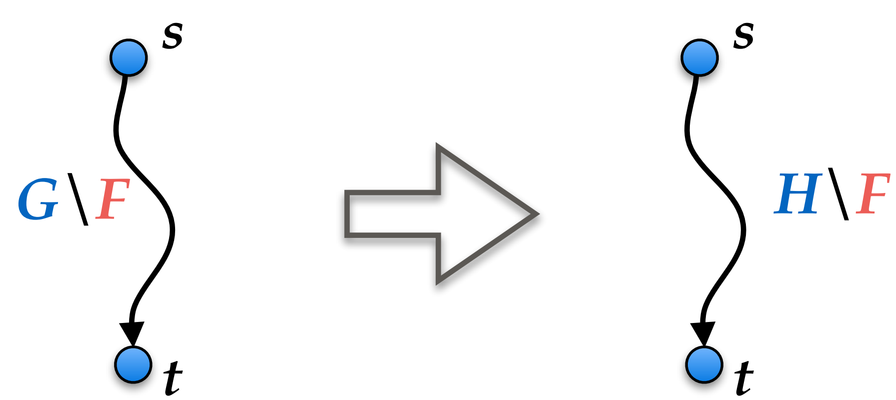
Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:

Problem 1: Reachability Preserver

Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:

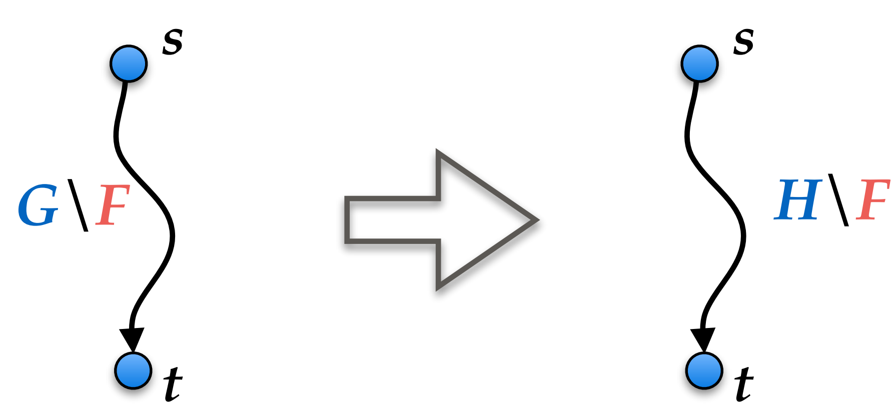


for every $t \in V$

Problem 1: Reachability Preserver

Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:



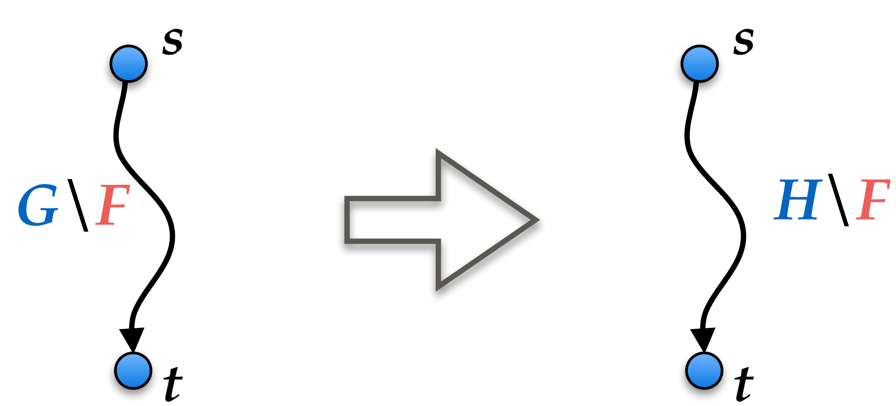
for every $t \in V$

Example:

Problem 1: Reachability Preserver

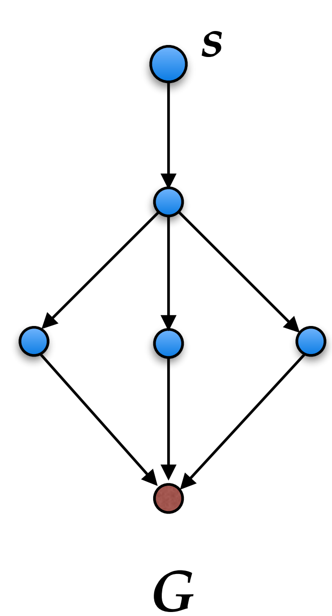
Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:



for every $t \in V$

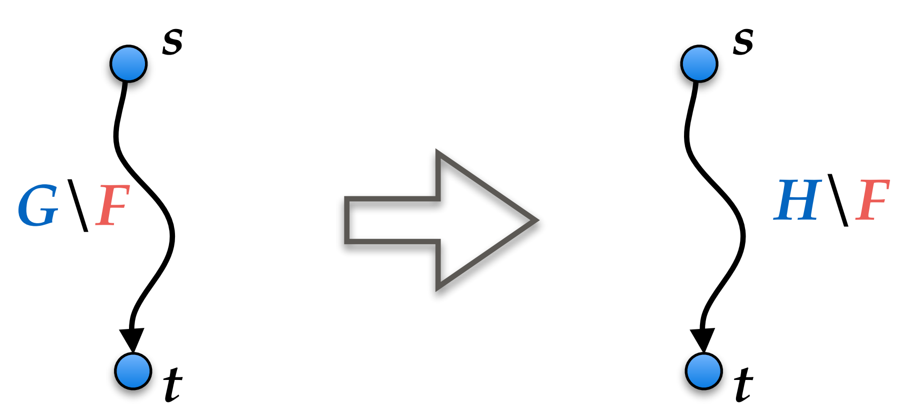
Example:



Problem 1: Reachability Preserver

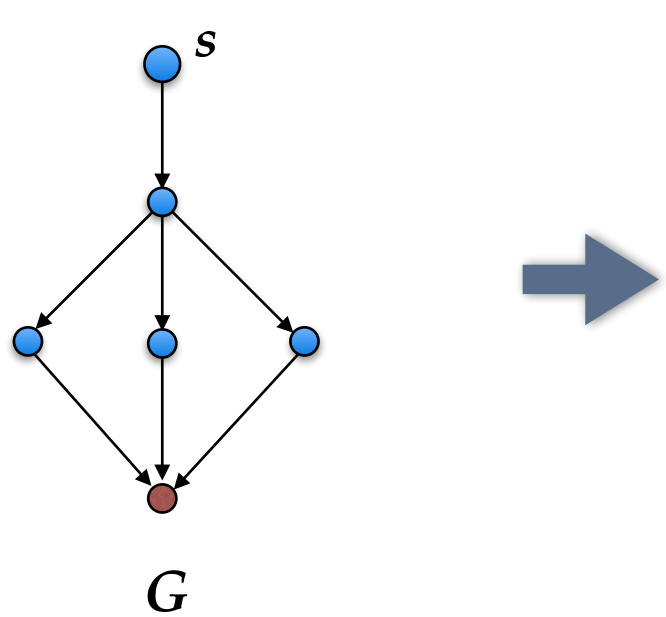
Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:



for every $t \in V$

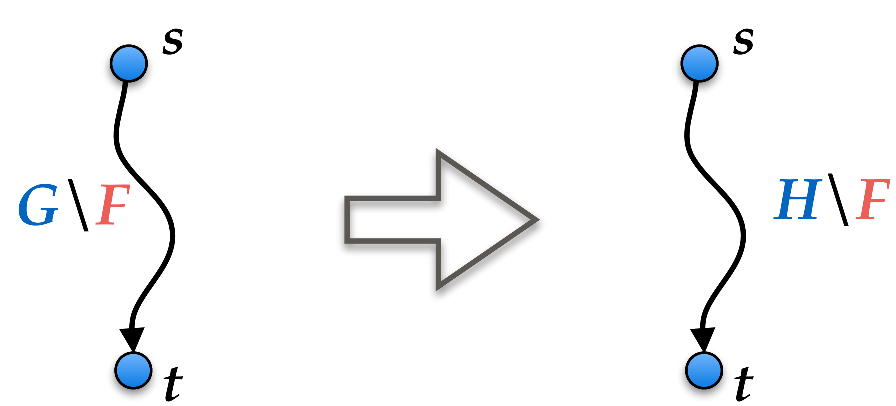
Example:



Problem 1: Reachability Preserver

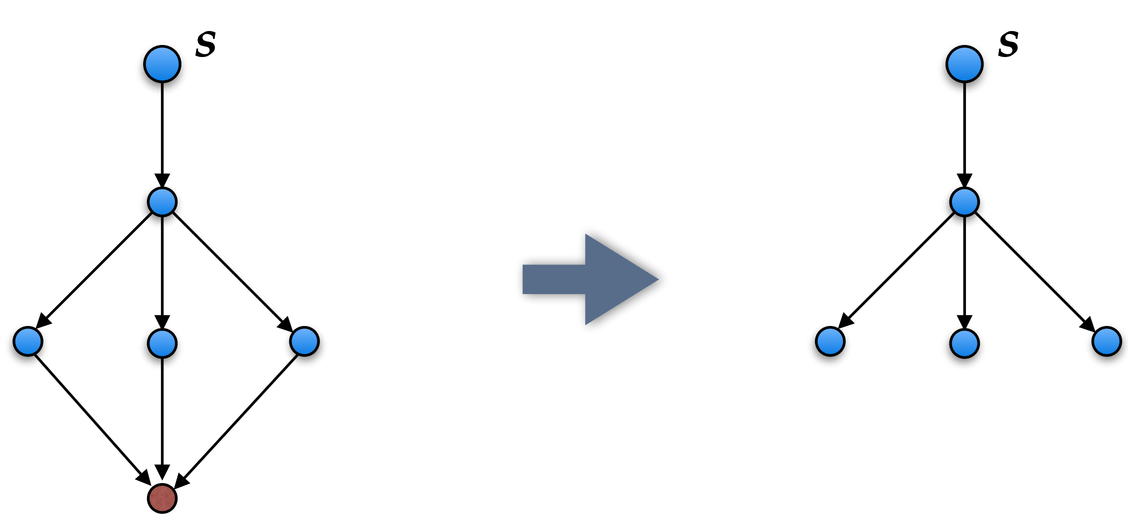
Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:



for every $t \in V$

Example:

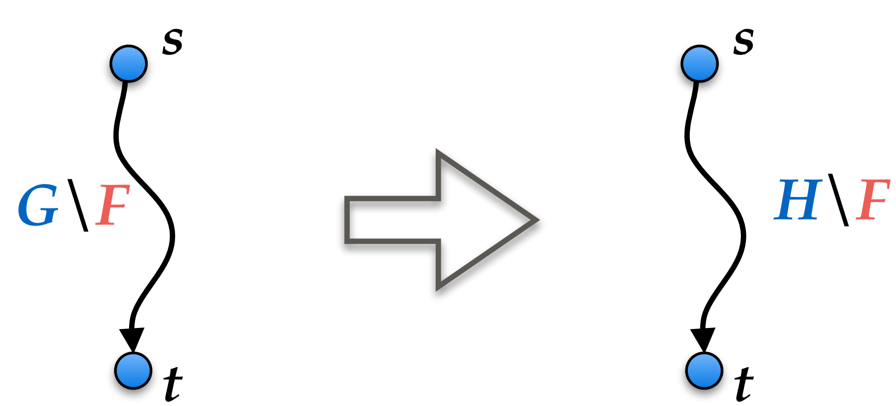


G 1-FT-Preserver

Problem 1: Reachability Preserver

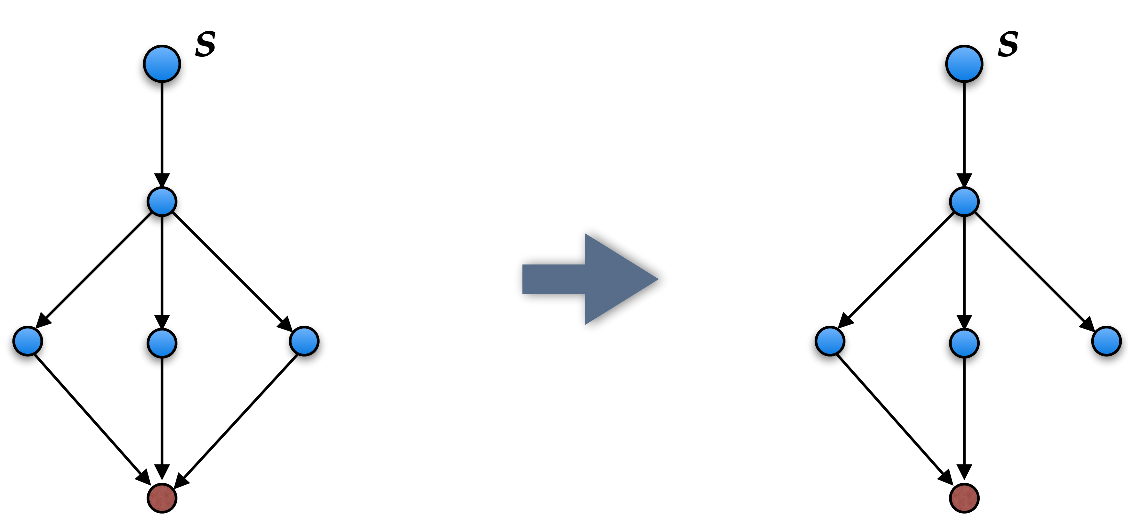
Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:



for every $t \in V$

Example:

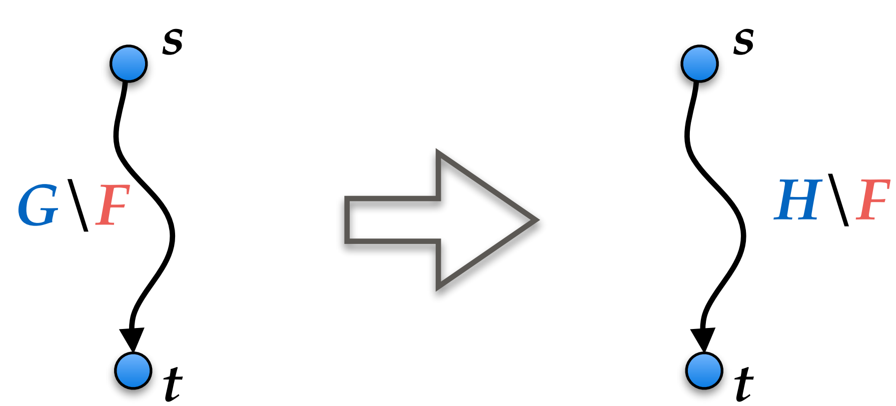


G **1**-FT-Preserver

Problem 1: Reachability Preserver

Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:

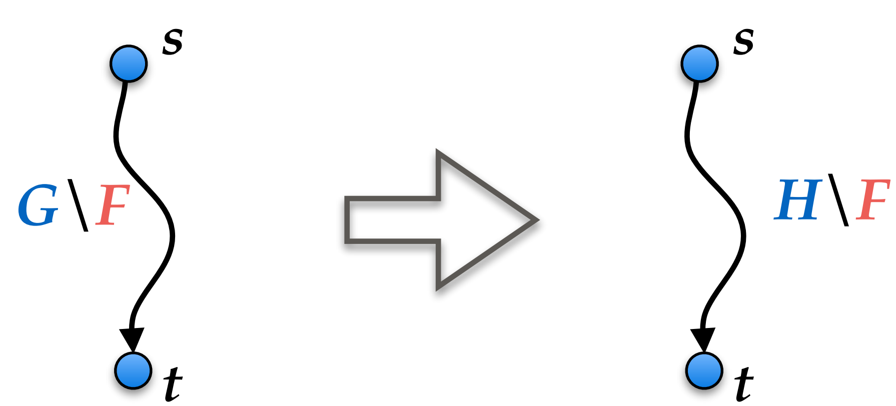


for every $t \in V$

Problem 1: Reachability Preserver

Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:



for every $t \in V$

Prior Work:

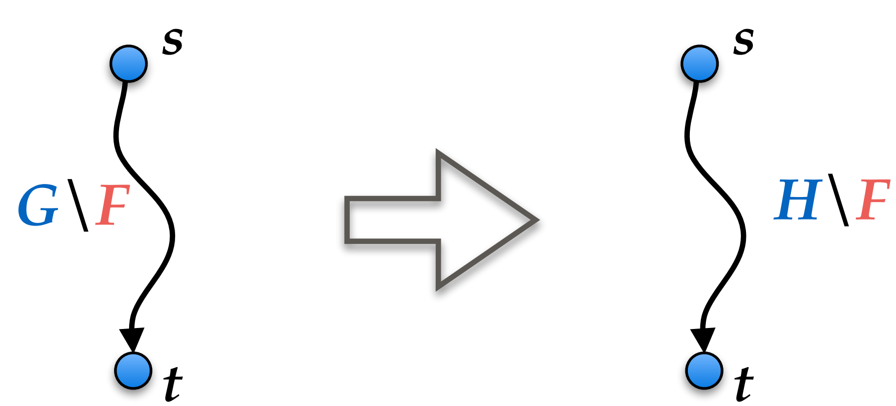
[Lengauer and Tarjan (1979)]:

- k=1 (single failure)
- An upper bound of $(2n)$

Problem 1: Reachability Preserver

Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:



for every $t \in V$

Prior Work:

[Lengauer and Tarjan (1979)]:

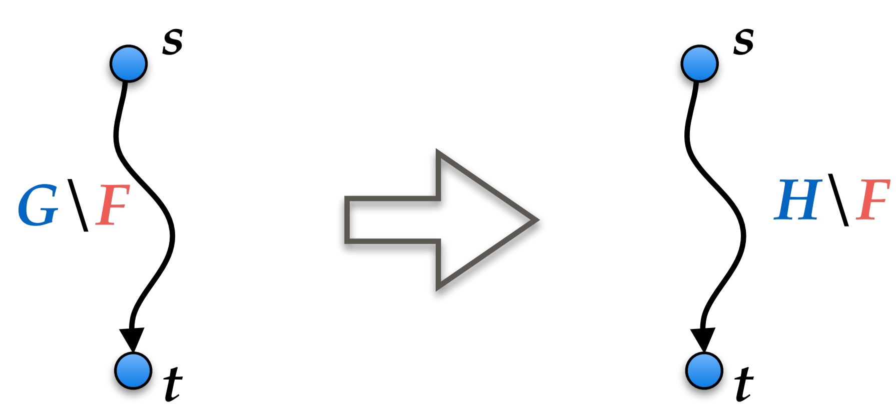
- k=1 (single failure)
- An upper bound of $(2n)$

Our Results for general k:

Problem 1: Reachability Preserver

Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:



for every $t \in V$

Prior Work:

[Lengauer and Tarjan (1979)]:

- k=1 (single failure)
- An upper bound of *(2n)*

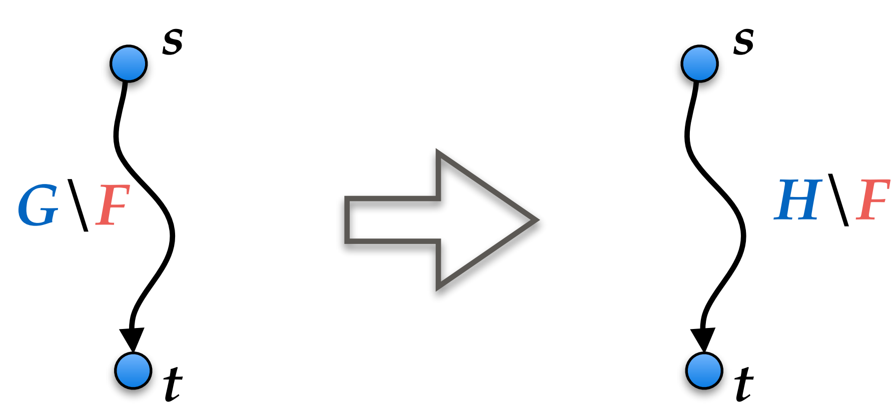
Our Results for general k:

Upper Bound:

Problem 1: Reachability Preserver

Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:



for every $t \in V$

Prior Work:

[Lengauer and Tarjan (1979)]:

- k=1 (single failure)
- An upper bound of $(2n)$

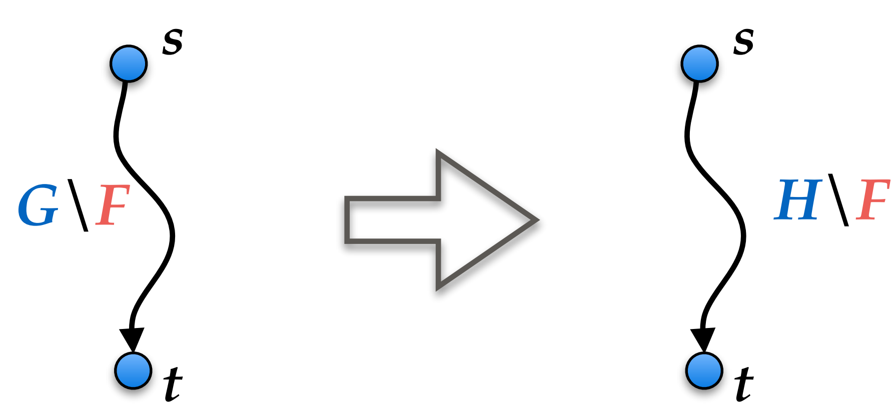
Our Results for general k:

Upper Bound: $O(2^k n)$ edges

Problem 1: Reachability Preserver

Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:



for every $t \in V$

Prior Work:

[Lengauer and Tarjan (1979)]:

- k=1 (single failure)
- An upper bound of $(2n)$

Our Results for general k:

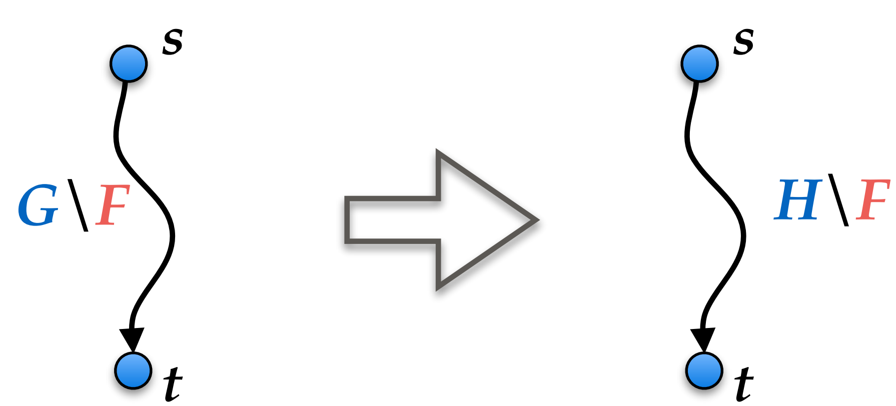
Upper Bound: $O(2^k n)$ edges

Lower Bound:

Problem 1: Reachability Preserver

Input: directed graph $G=(V,E)$, parameter k, and a source s.

Output: a sparse subgraph H of G that on any set F of k edges satisfies:



for every $t \in V$

Prior Work:

[Lengauer and Tarjan (1979)]:

- k=1 (single failure)
- An upper bound of $(2n)$

Our Results for general k:

Upper Bound: $O(2^k n)$ edges

Lower Bound: Existential bound of $\Omega(2^k n)$ edges

Problem 1: Reachability Preserver

Implication (i):

Problem 1: Reachability Preserver

Implication (i):

Reachability Oracle

Problem 1: Reachability Preserver

Implication (i):

Problem 1: Reachability Preserver

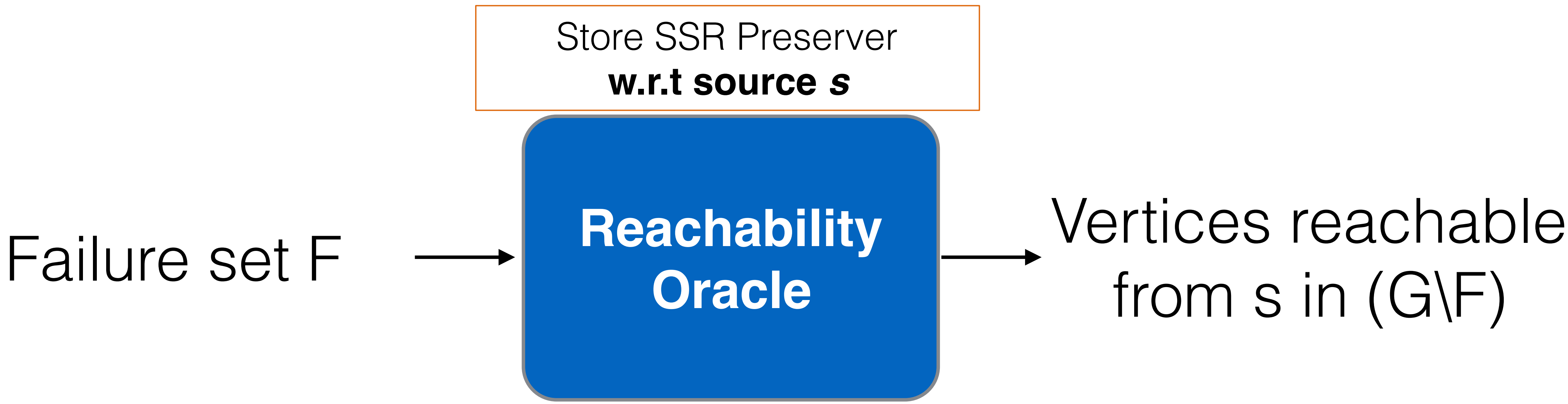
Implication (i):

Store SSR Preserver
w.r.t source *s*

Failure set F → **Reachability Oracle** → Vertices reachable from s in (G\F)

Problem 1: Reachability Preserver

Implication (i):



Time: $O(2^k n)$ time!

Problem 1: Reachability Preserver

Implication (i):

Store SSR Preserver
w.r.t source *s*

Failure set F → **Reachability Oracle** → Vertices reachable from s in (G\F)

Time: O(2^k n) time!

Space: O(2^k n)

Problem 1: Reachability Preserver

Implication (ii):

Problem 1: Reachability Preserver

Implication (ii):

SCC
Oracle

Problem 1: Reachability Preserver

Implication (ii):

Problem 1: Reachability Preserver

Implication (ii):

Store **ALL** SSR Preservers
w.r.t. G as well G-reverse

Vertex y, set F → **SCC Oracle** → SCC of y in (G\F)

Problem 1: Reachability Preserver

Implication (ii):

Store **ALL** SSR Preservers
w.r.t. G as well G-reverse

Vertex y, set F → **SCC Oracle** → SCC of y in (G\F)

Time: O($2^k n$) time!

Problem 1: Reachability Preserver

Implication (ii):

Store **ALL** SSR Preservers w.r.t. G as well G-reverse

Vertex y, set F → **SCC Oracle** → SCC of y in (G\F)

Time: $O(2^k n)$ time!

Space: $O(2^k n^2)$

Problem 1: Reachability Preserver

Proof Snippet

Problem 1: Reachability Preserver

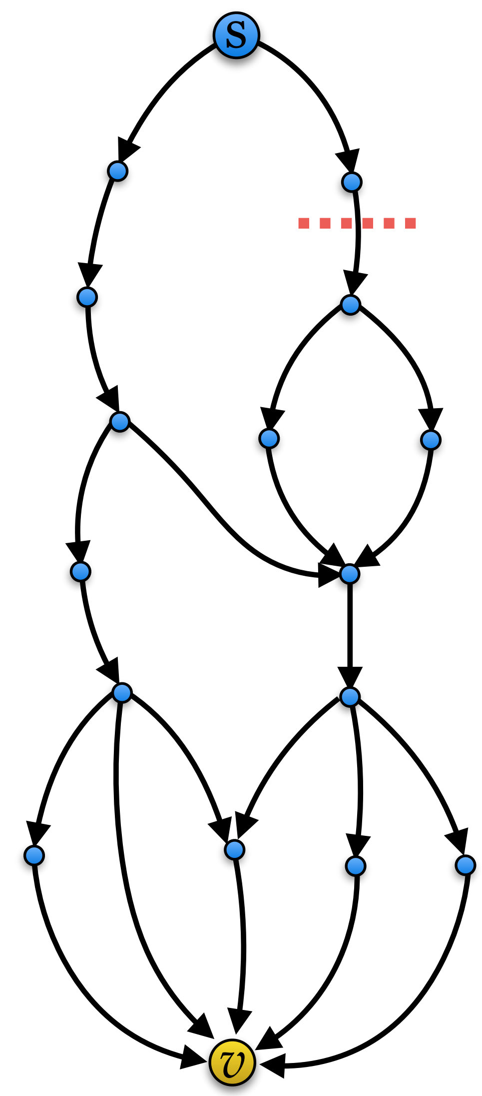
Proof Snippet

Farthest min-cut

Problem 1: Reachability Preserver

Proof Snippet

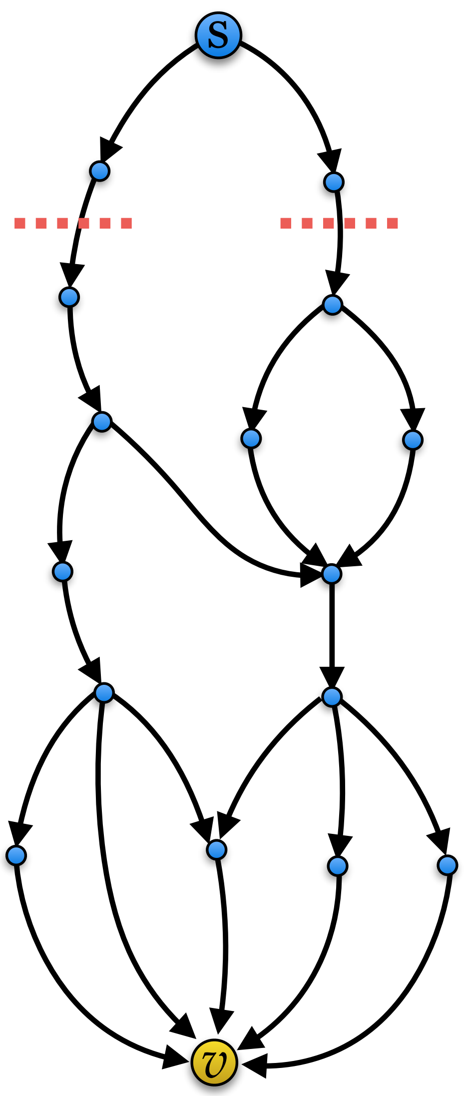
Farthest min-cut



Problem 1: Reachability Preserver

Proof Snippet

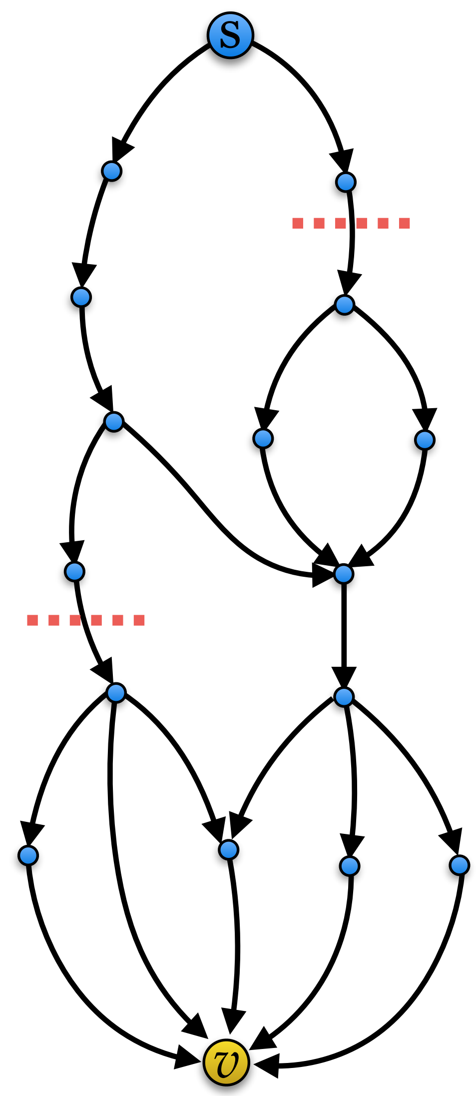
Farthest min-cut



Problem 1: Reachability Preserver

Proof Snippet

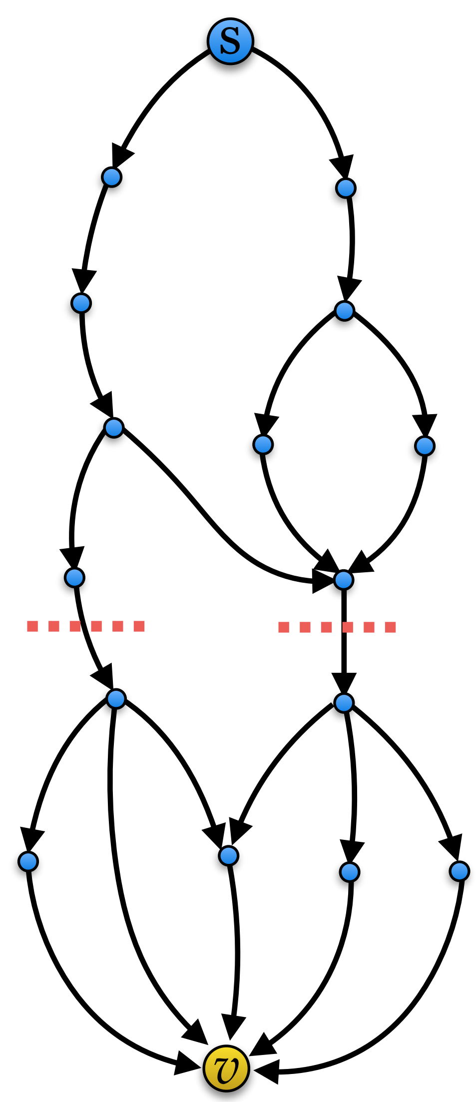
Farthest min-cut



Problem 1: Reachability Preserver

Proof Snippet

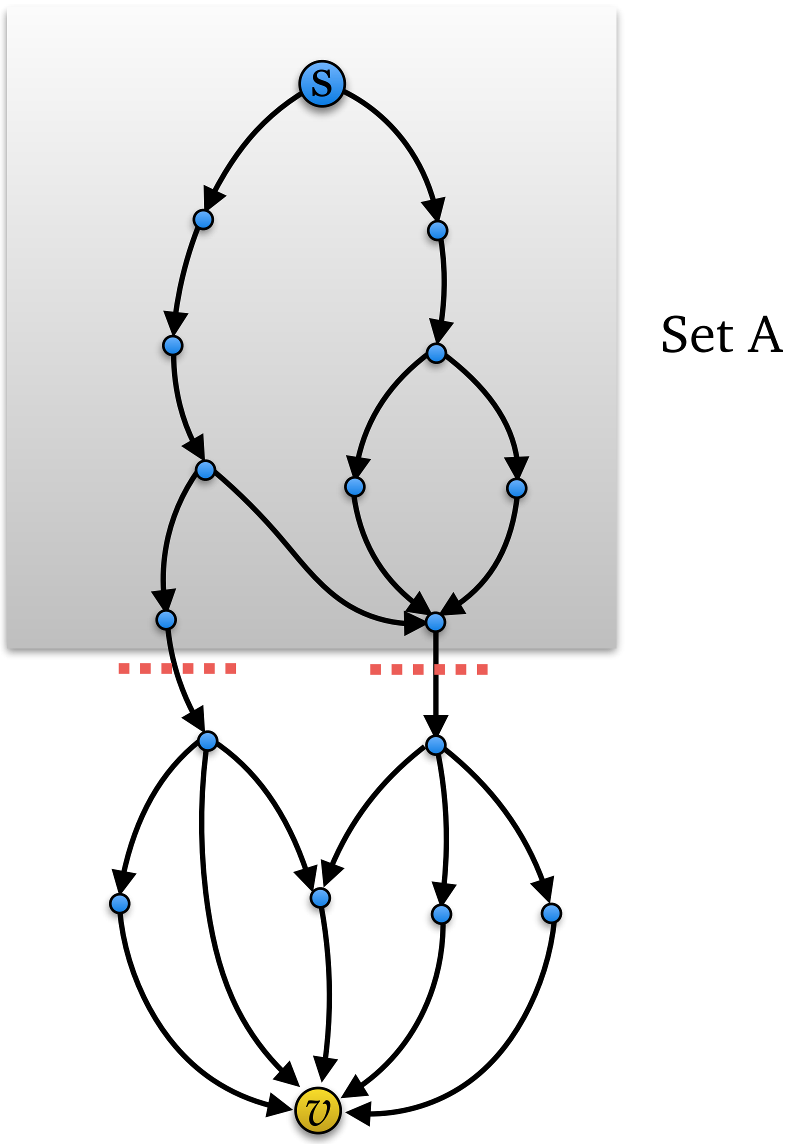
Farthest min-cut



Problem 1: Reachability Preserver

Proof Snippet

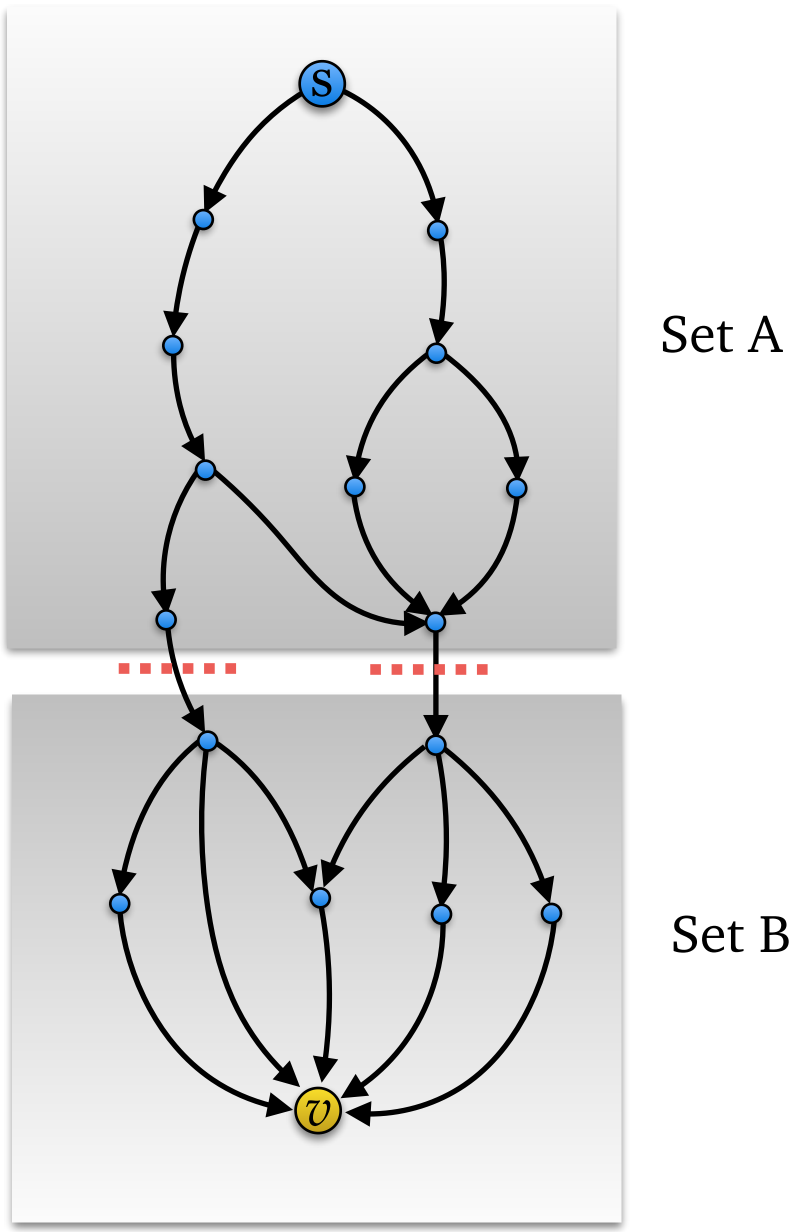
Farthest min-cut



Problem 1: Reachability Preserver

Proof Snippet

Farthest min-cut



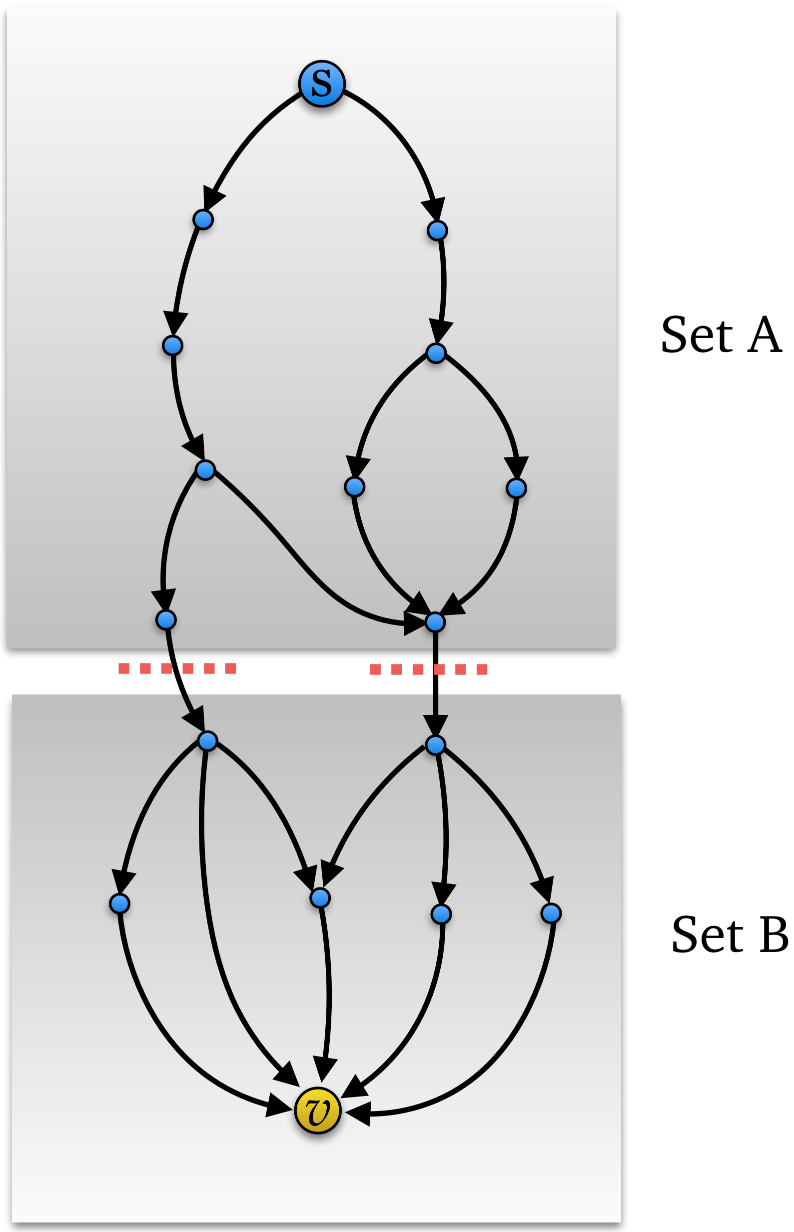
Problem 1: Reachability Preserver

Proof Snippet

Farthest min-cut

Definition

The min cut $\{A,B\}$ for which the set A **is of maximum size**.



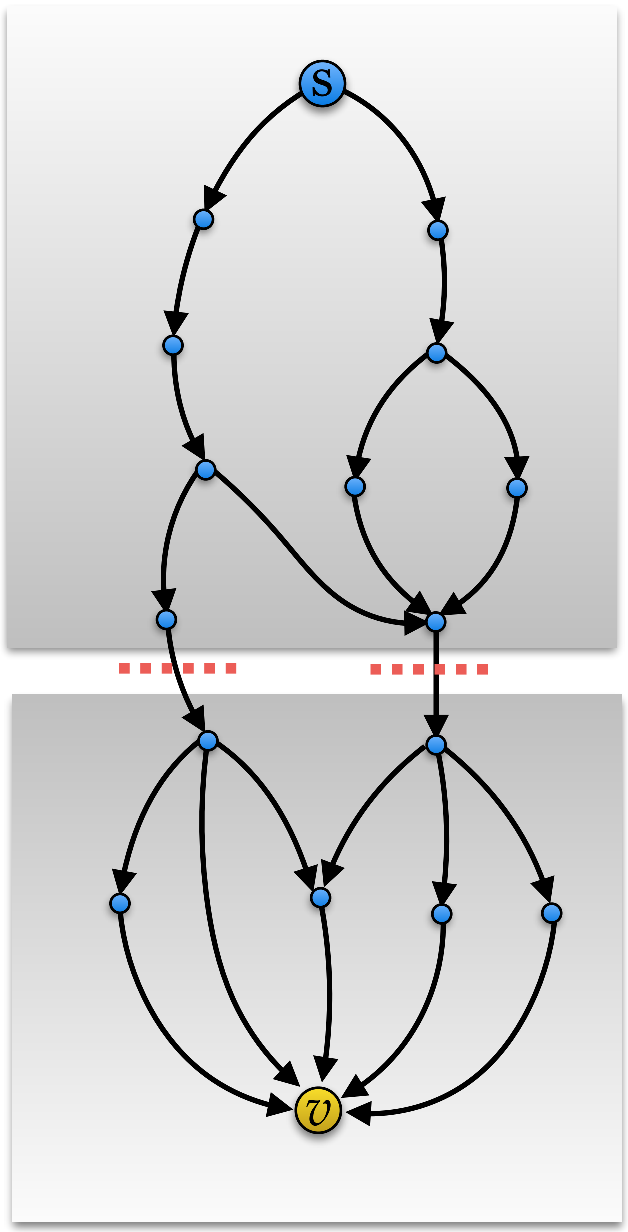
Problem 1: Reachability Preserver

Proof Snippet

Farthest min-cut

Definition

The min cut $\{A,B\}$ for which the set A **is of maximum size**.



Set A

Farthest (s,v)-min-cut

Set B

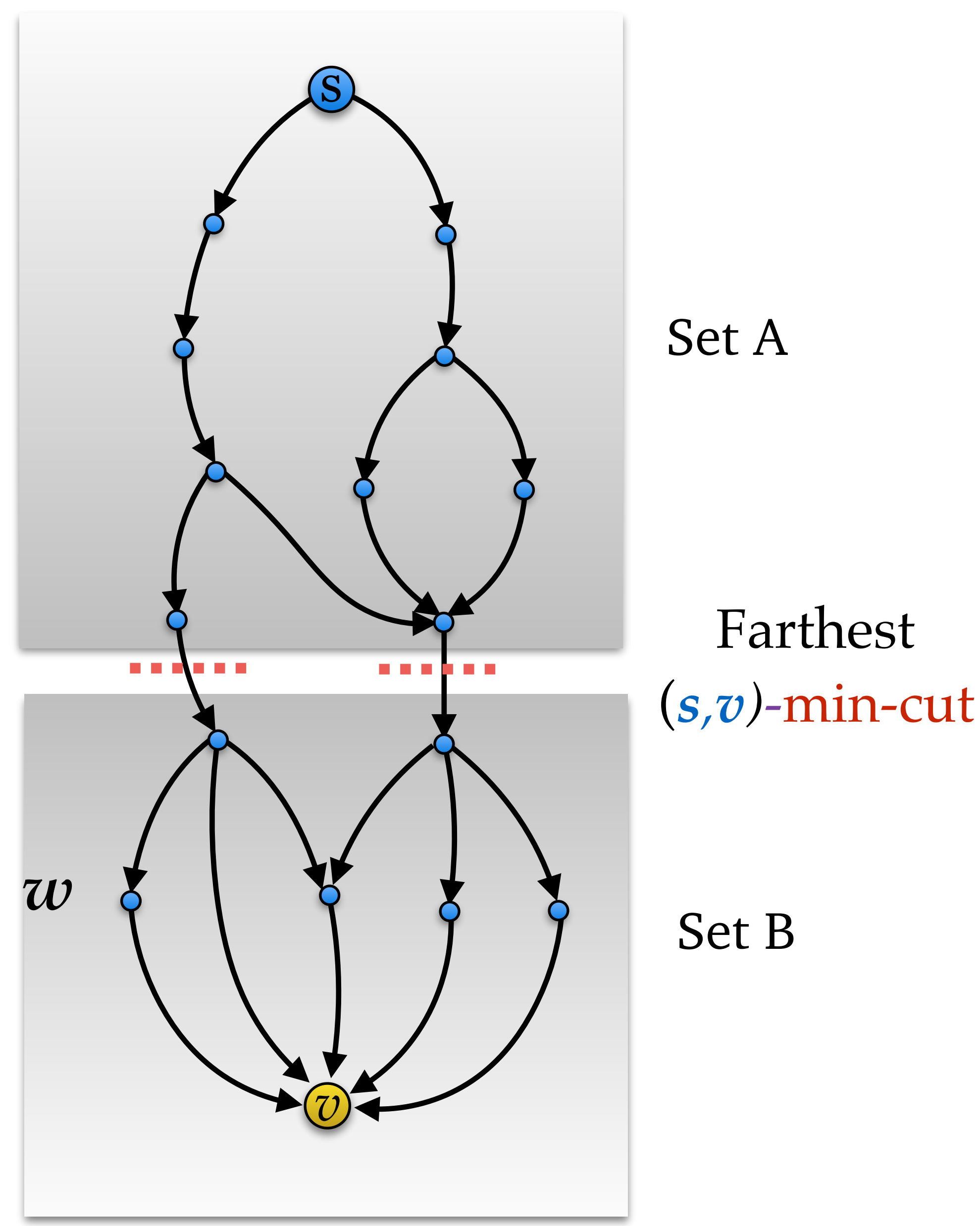
Problem 1: Reachability Preserver

Proof Snippet

Farthest min-cut

Definition

The min cut $\{A,B\}$ for which the set A **is of maximum size**.



Characterisation

Vertex w lies in B, iff

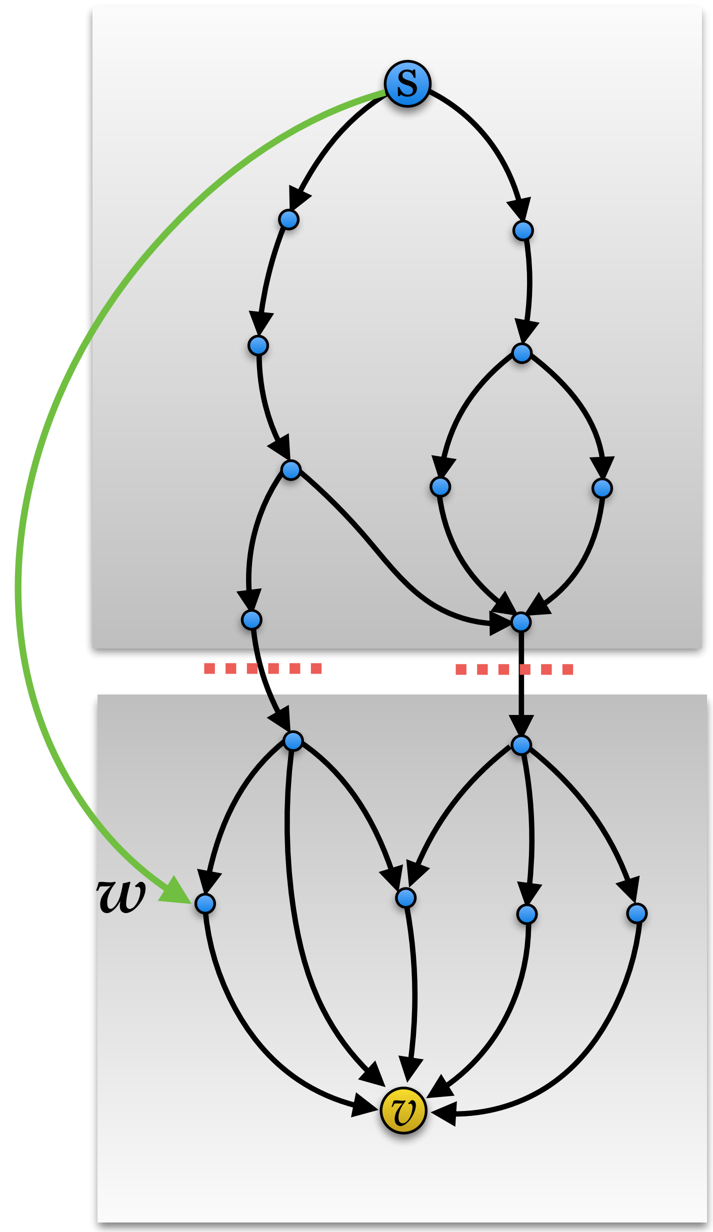
Problem 1: Reachability Preserver

Proof Snippet

Farthest min-cut

Definition

The min cut $\{A,B\}$ for which the set A **is of maximum size**.



Set A

Farthest (s,v)-min-cut

Set B

Characterisation

Vertex w lies in B, iff

Problem 1: Reachability Preserver

Proof Snippet

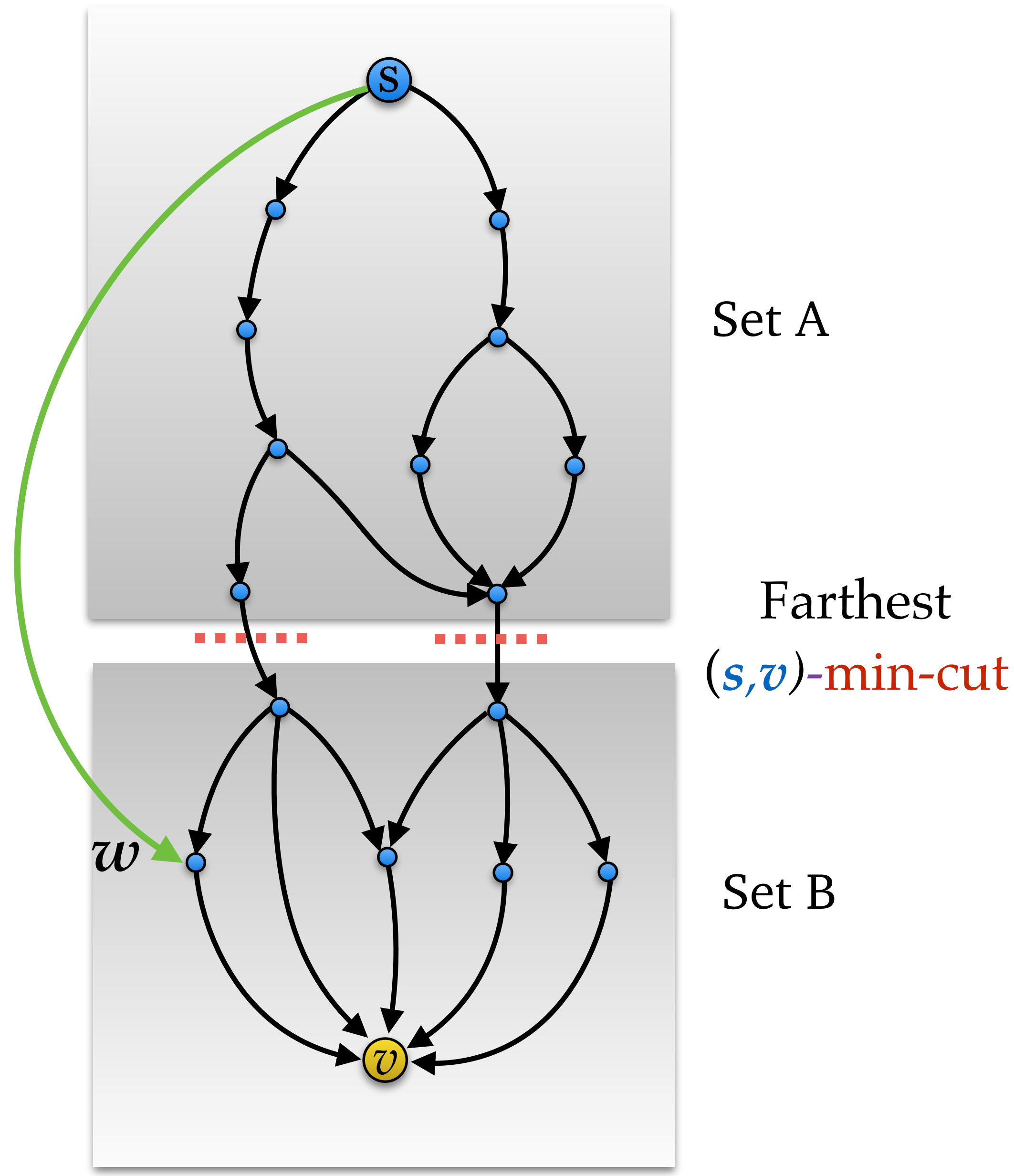
Farthest min-cut

Definition

The min cut $\{A,B\}$ for which the set A **is of maximum size**.

Characterisation

Vertex w lies in B, iff $\textit{max-flow}(G+(s,w)) > \textit{max-flow}(G)$



Problem 1: Reachability Preserver

Proof Snippet

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: $(k+1)!$

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: $s \times v$ - preserver , with bounded in-degree(v)

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: s **x** v - preserver , with bounded in-degree(v)

*s **x** $V(G)$ - preserver*

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: s **x** v - preserver , with bounded in-degree(v)

s* x *V(G)* - *preserver

Preserves reachability from s to **each** vertex of G

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: $(k+1)$!

Bottleneck: s x v - preserver , with bounded in-degree(v)

s* x *V(G)* - *preserver

Preserves reachability from s to **each** vertex of G

upon failure of at most k edges

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: $(k+1)$!

Bottleneck: s x v - preserver , with bounded in-degree(v)

s* x *V(G)* - *preserver

Preserves reachability from s to **each** vertex of G

upon failure of at most k edges

Main Goal

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: s x v - preserver , with bounded in-degree(v)

s **x** *V(G)* - *preserver*

Preserves reachability from s to **each** vertex of G

upon failure of at most k edges

Main Goal

s **x** *V(G)* - *preserver* in which in-degree of **each vertex** is bounded

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: $(k+1)$!

Bottleneck: s x v - preserver , with bounded in-degree(v)

s* x *V(G)* - *preserver

Preserves reachability from s to **each** vertex of G upon failure of at most k edges

Main Goal

s **x** *V(G)* - *preserver* in which in-degree of **each vertex** is bounded

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: $(k+1)$!

Bottleneck: s x v - preserver , with bounded in-degree(v)

s* x *V(G)* - *preserver

Preserves reachability from s to **each** vertex of G upon failure of at most k edges

Main Goal

s **x** *V(G)* - *preserver* in which in-degree of **each vertex** is bounded

s* x *v* - *preserver

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: s **x** v - preserver , with bounded in-degree(v)

<u>*s* **x** *V(G) - preserver*</u>

Preserves reachability from s to **<u>each</u>** vertex of G

upon failure of at most k edges

Main Goal

s **x** *V(G) - preserver* in which in-degree of **<u>each vertex</u>** is bounded

<u>*s* **x** *v - preserver*</u>

Preserves reachability from s to **<u>only</u>** v

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: $s \times v$ - preserver , with bounded in-degree(v)

s* x *V(G)* - *preserver

Preserves reachability from s to **each** vertex of G

upon failure of at most k edges

Main Goal

$s \times V(G)$ - *preserver* in which in-degree of **each vertex** is bounded

s* x *v* - *preserver

Preserves reachability from s to **only** v

upon failure of at most k edges

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: s x v - preserver , with bounded in-degree(v)

s* x *V(G)* - *preserver

Preserves reachability from s to **each** vertex of G upon failure of at most k edges

s* x *v* - *preserver

Preserves reachability from s to **only** v upon failure of at most k edges

Main Goal

s **x** *V(G)* - *preserver* in which in-degree of **each vertex** is bounded

A Simpler Problem

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: s x v - preserver , with bounded in-degree(v)

s **x** *V(G) - preserver*

Preserves reachability from s to **each** vertex of G upon failure of at most k edges

Main Goal

s **x** *V(G) - preserver* in which in-degree of **each vertex** is bounded

s **x** *v - preserver*

Preserves reachability from s to **only** v upon failure of at most k edges

A Simpler Problem

s **x** *v - preserver* in which in-degree of **only** v is bounded

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: s x v - preserver , with bounded in-degree(v)

***s* x *V(G)* - preserver**

Preserves reachability from s to **each** vertex of G upon failure of at most k edges

***s* x *v* - preserver**

Preserves reachability from s to **only** v upon failure of at most k edges

Main Goal

s x $V(G)$ - *preserver* in which in-degree of **each vertex** is bounded

A Simpler Problem

s x v - *preserver* in which in-degree of **only** v is bounded

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: s x v - preserver , with bounded in-degree(v)

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: s x v - preserver , with bounded in-degree(v)

Case 1: Max-Flow(s,v) ≥ k+1

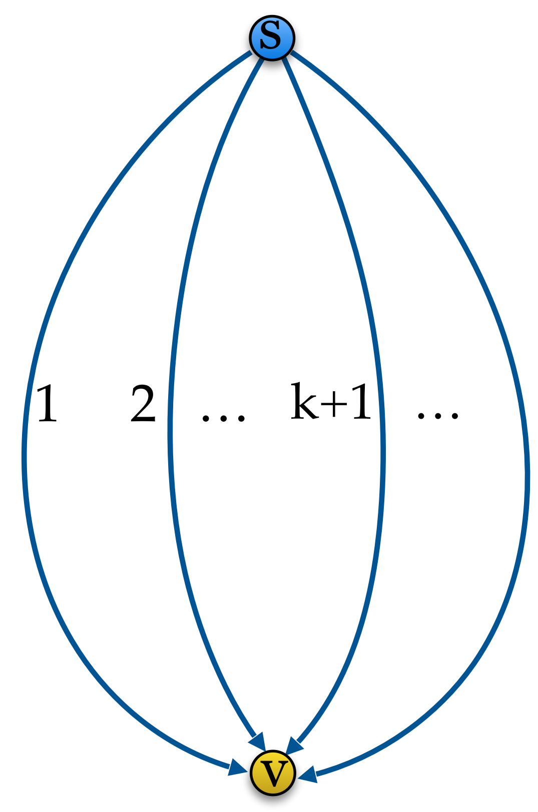
Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: $s \times v$ - preserver , with bounded in-degree(v)

Case 1: Max-Flow(s,v) ≥ k+1



Problem 1: Reachability Preserver

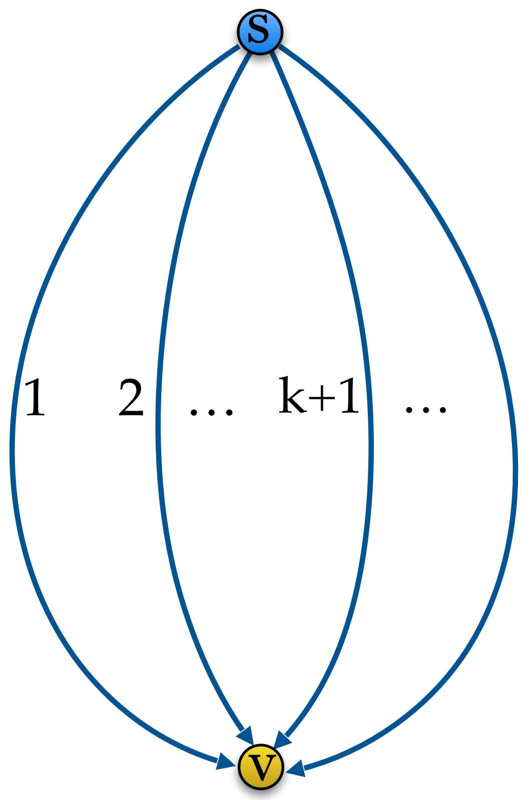
Proof Snippet

In-degree at most: ***(k+1)***!

Bottleneck: *s* x *v* - preserver , with bounded in-degree(v)

Case 1: Max-Flow(s,v) ≥ k+1

H = Union of any ***k+1*** edge-disjoint paths



Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: s x v - preserver , with bounded in-degree(v)

Problem 1: Reachability Preserver

Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: $s \times v$ - preserver , with bounded in-degree(v)

Case 2: Max-Flow(s,v) = r < k+1

Problem 1: Reachability Preserver

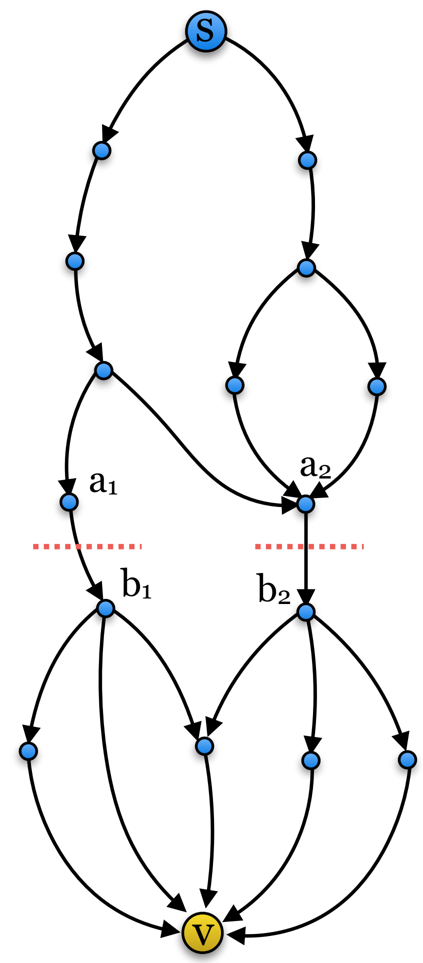
Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: $s \times v$ - preserver , with bounded in-degree(v)

Case 2: Max-Flow(s,v) = r < k+1

- Let farthest Min-cut = $\{(a_1,b_1), \ldots, (a_r,b_r)\}$



Problem 1: Reachability Preserver

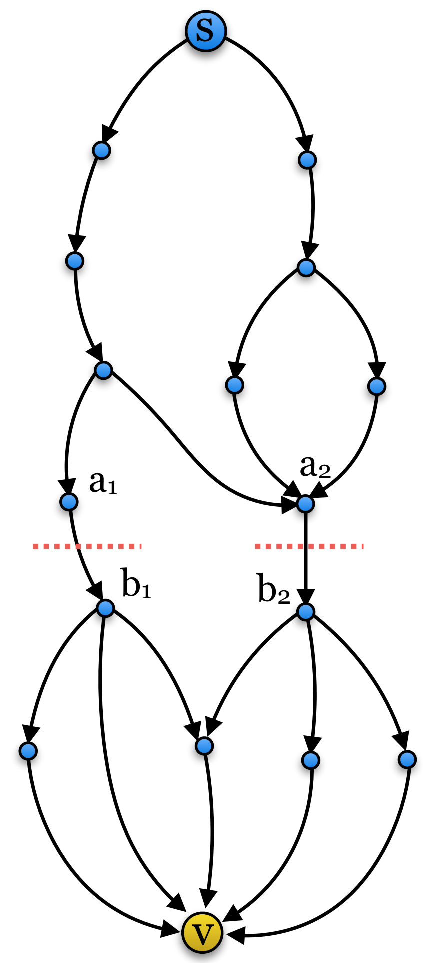
Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: $s \times v$ - preserver , with bounded in-degree(v)

Case 2: Max-Flow(s,v) = r < k+1

- Let farthest Min-cut = $\{(a_1,b_1), \ldots, (a_r,b_r)\}$



$G_1 := G+(s,b_1)$

Problem 1: Reachability Preserver

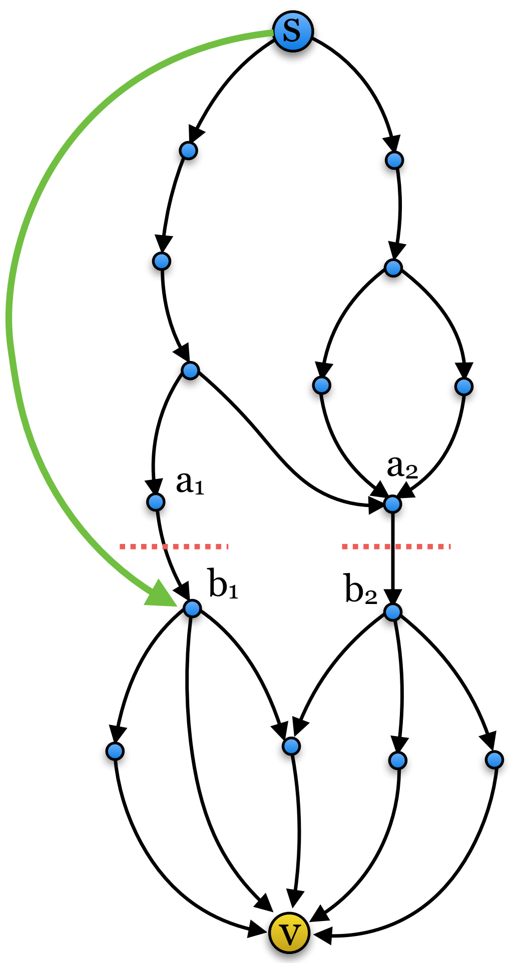
Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: $s \times v$ - preserver , with bounded in-degree(v)

Case 2: Max-Flow(s,v) = r < k+1

- Let farthest Min-cut = $\{(a_1,b_1), \ldots, (a_r,b_r)\}$



$G_1 := G+(s,b_1)$

Problem 1: Reachability Preserver

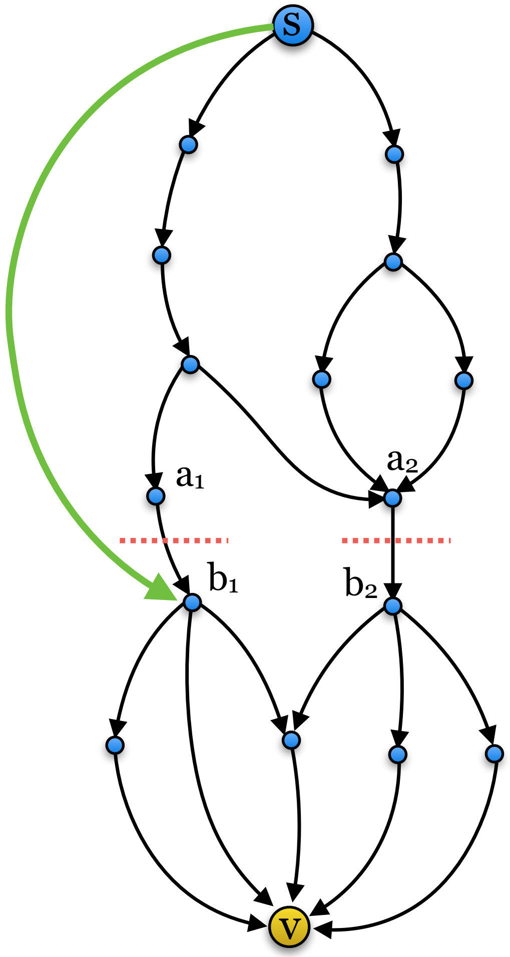
Proof Snippet

In-degree at most: *(k+1)*!

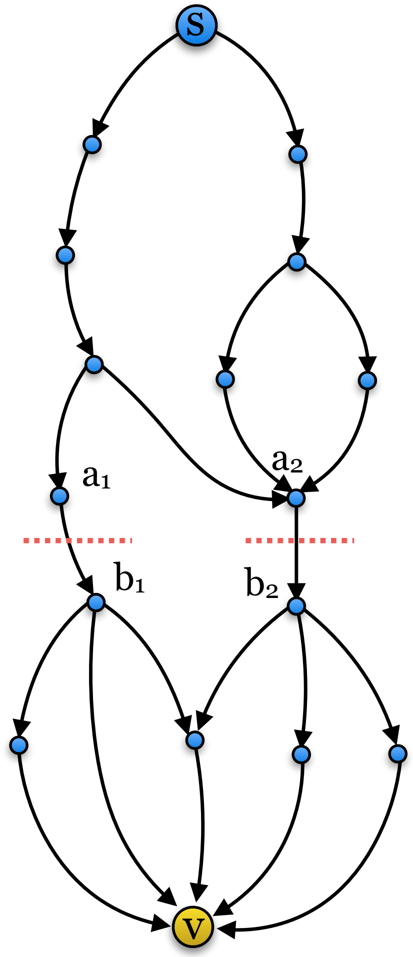
Bottleneck: $s \times v$ - preserver , with bounded in-degree(v)

Case 2: Max-Flow(s,v) = r < k+1

- Let farthest Min-cut = $\{(a_1,b_1), \ldots, (a_r,b_r)\}$



$G_1 := G+(s,b_1)$



Problem 1: Reachability Preserver

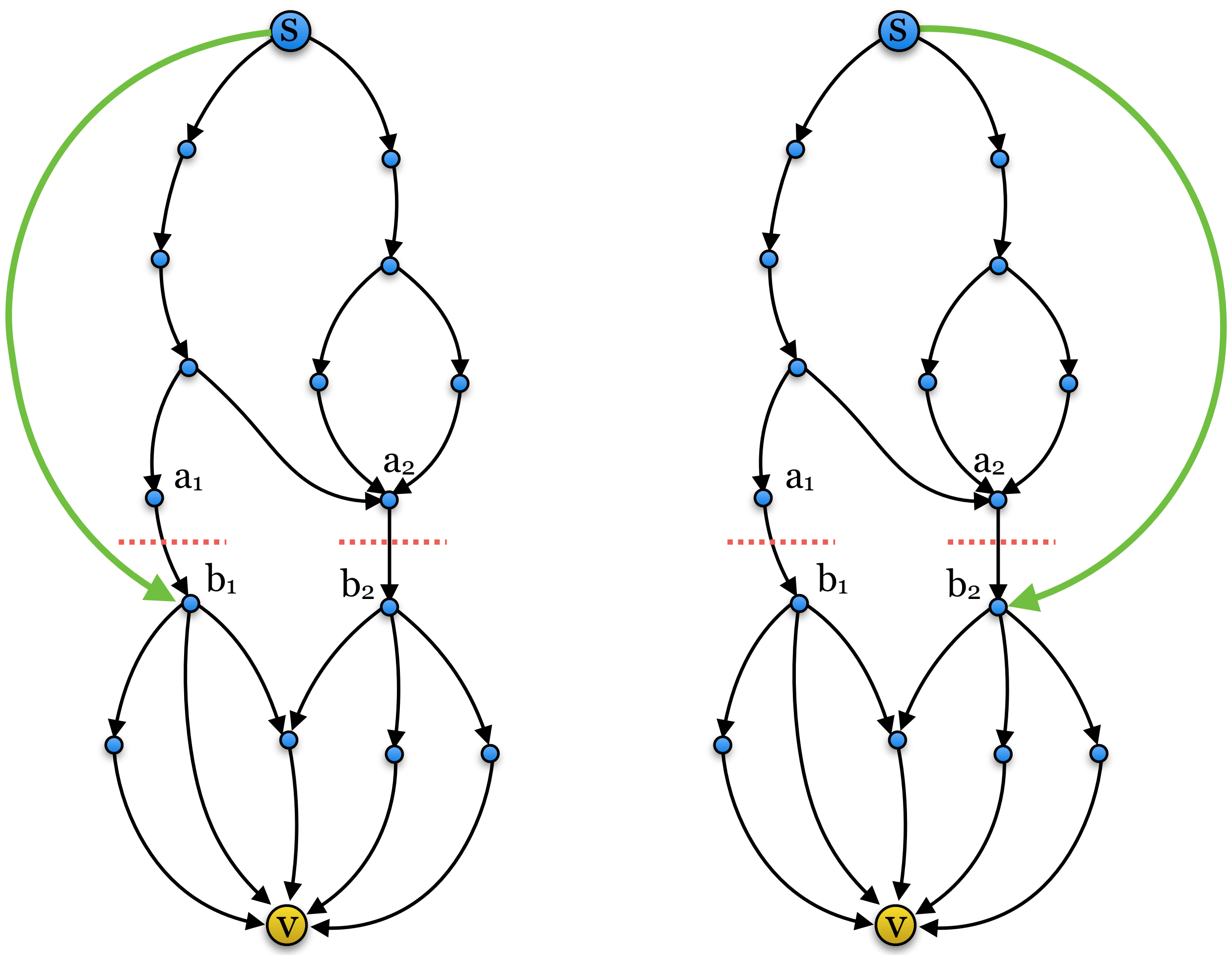
Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: $s \times v$ - preserver , with bounded in-degree(v)

Case 2: Max-Flow(s,v) = r < k+1

- Let farthest Min-cut = $\{(a_1,b_1), \ldots, (a_r,b_r)\}$



$G_1 := G+(s,b_1)$

Problem 1: Reachability Preserver

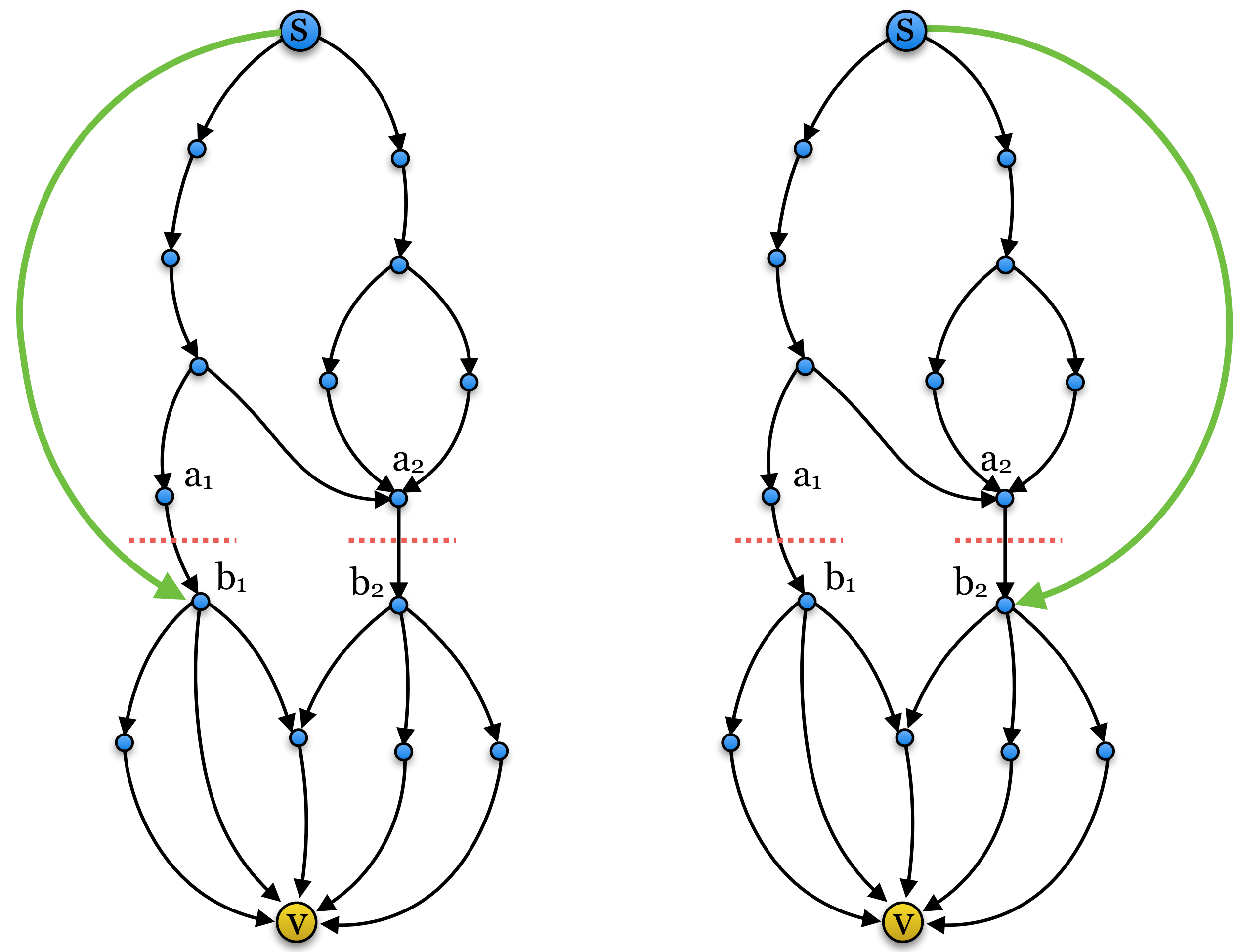
Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: $s \times v$ - preserver, with bounded in-degree(v)

Case 2: Max-Flow(s,v) = r < k+1

- Let farthest Min-cut = $\{(a_1,b_1), \ldots, (a_r,b_r)\}$



$G_1 := G+(s,b_1)$ $\quad$ $G_2 := G+(s,b_2)$

Problem 1: Reachability Preserver

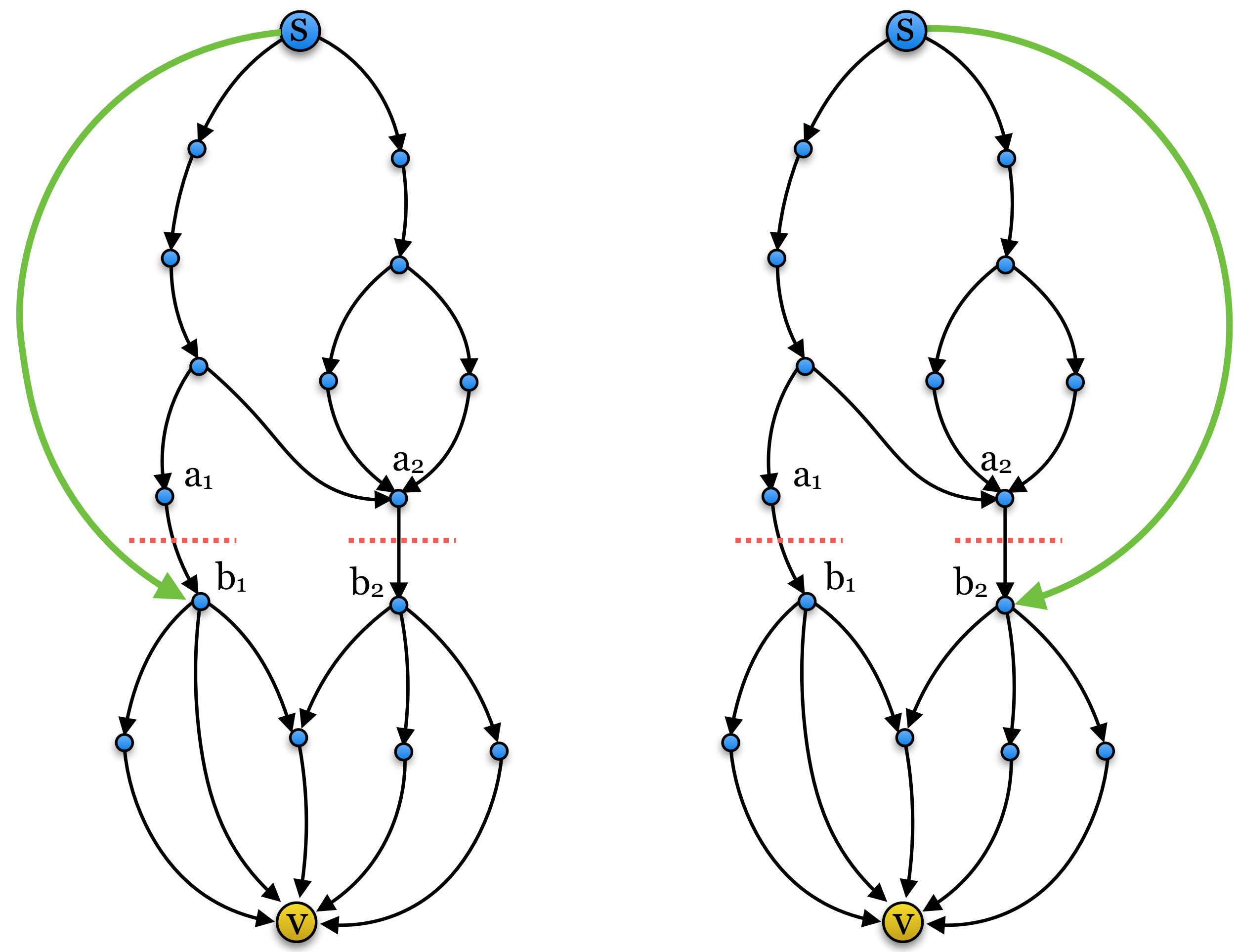
Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: $s \times v$ - preserver , with bounded in-degree(v)

Case 2: Max-Flow(s,v) = r < k+1

- Let farthest Min-cut = $\{(a_1,b_1), \ldots, (a_r,b_r)\}$
- Find Preserver (say H_i) w.r.t. $G_i = G+(s,b_i)$



$G_1 := G+(s,b_1)$ $G_2 := G+(s,b_2)$

Problem 1: Reachability Preserver

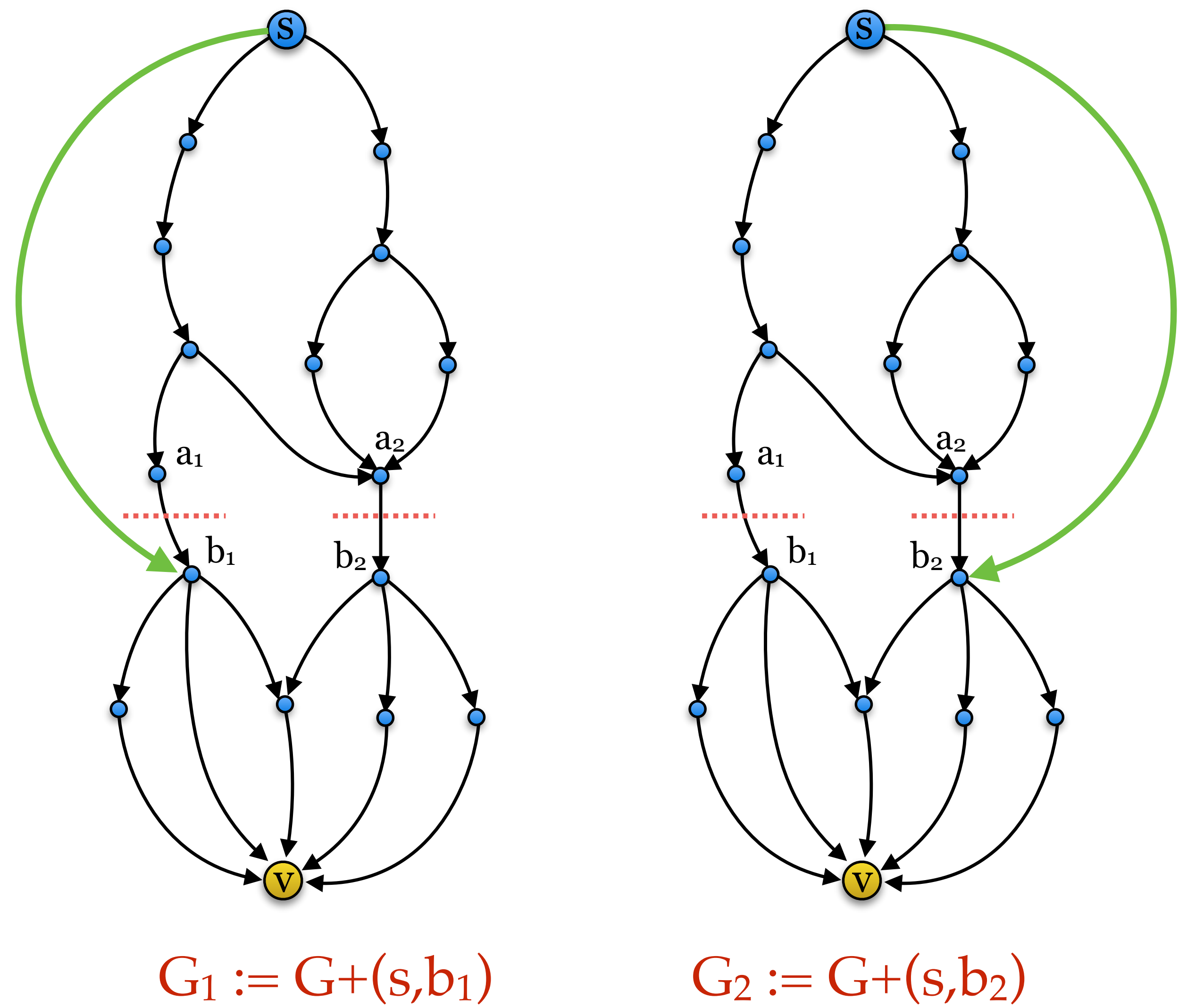
Proof Snippet

In-degree at most: *(k+1)*!

Bottleneck: $s \times v$ - preserver , with bounded in-degree(v)

Case 2: Max-Flow(s,v) = r < k+1

- Let farthest Min-cut = $\{(a_1,b_1), \ldots, (a_r,b_r)\}$
- Find Preserver (say H_i) w.r.t. $G_i = G+(s,b_i)$
- SET: $E(v,H) = \bigcup_{i=1 \text{ to } r} E(v,H_i)$



Problem 2: SCC Oracle

Input: directed graph $G=(V,E)$, parameter k.

Output: a data-structure that on failure of any set F of k edges outputs:

Problem 2: SCC Oracle

Input: directed graph $G=(V,E)$, parameter k.

Output: a data-structure that on failure of any set F of k edges outputs:

Strongly-connected-components **(SCCs)** of $G \setminus F$

Problem 2: SCC Oracle

Input: directed graph $G=(V,E)$, parameter k.

Output: a data-structure that on failure of any set F of k edges outputs:

Strongly-connected-components **(SCCs)** of $G \setminus F$

Example (k=1)

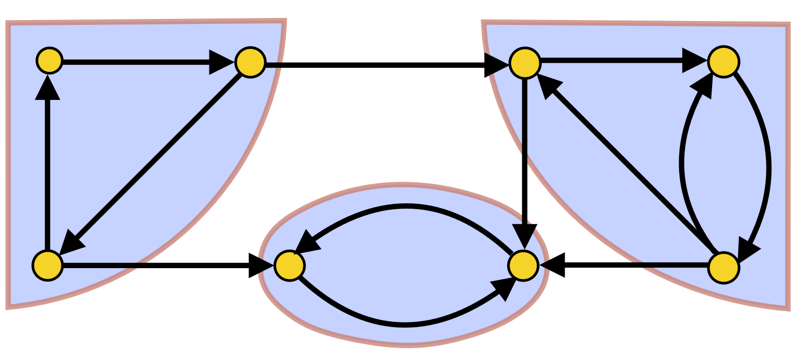
Problem 2: SCC Oracle

Input: directed graph ***G=(V,E)***, parameter ***k***.

Output: a data-structure that on failure of any set ***F*** of ***k*** edges outputs:

Strongly-connected-components **(SCCs)** of ***G\F***

Example (k=1)



G

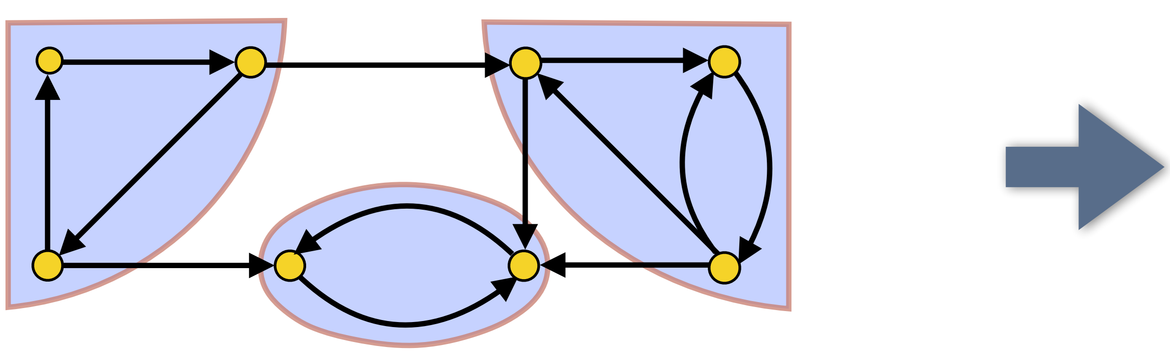
Problem 2: SCC Oracle

Input: directed graph ***G=(V,E)***, parameter ***k***.

Output: a data-structure that on failure of any set ***F*** of ***k*** edges outputs:

Strongly-connected-components **(SCCs)** of ***G\F***

Example (k=1)



G

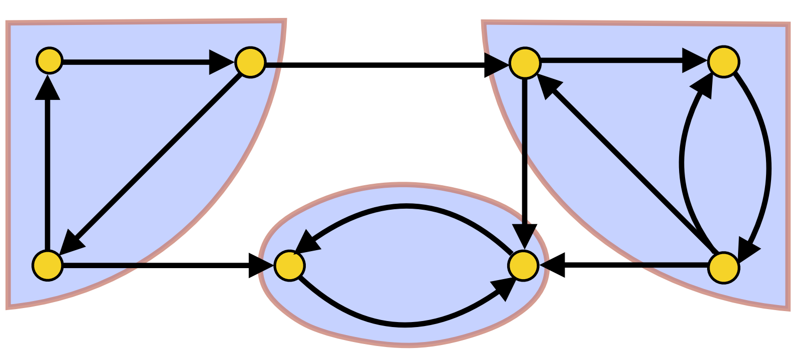
Problem 2: SCC Oracle

Input: directed graph $G=(V,E)$, parameter k.

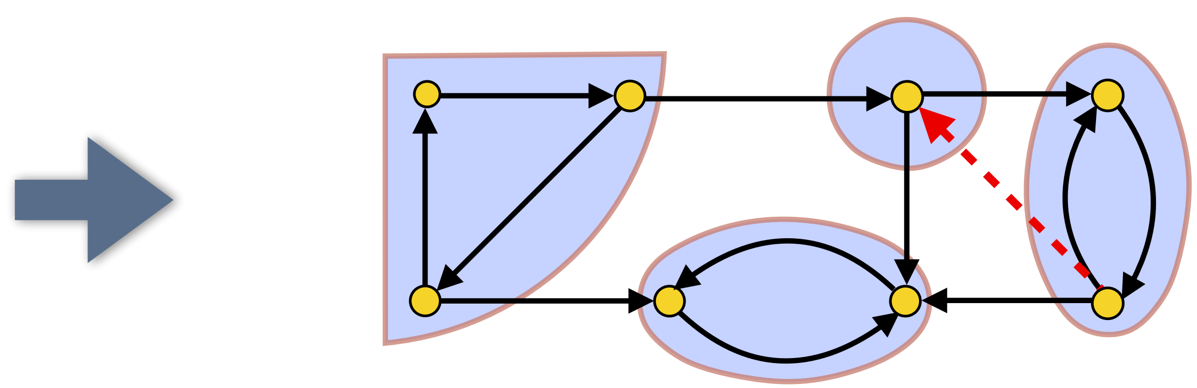
Output: a data-structure that on failure of any set F of k edges outputs:

Strongly-connected-components **(SCCs)** of $G\backslash F$

Example (k=1)



G



SCCs in G\F

Problem 2: SCC Oracle

Input: directed graph $G=(V,E)$, parameter k.

Output: a data-structure that on failure of any set F of k edges outputs:

Strongly-connected-components **(SCCs)** of $G \setminus F$

Problem 2: SCC Oracle

Input: directed graph $G=(V,E)$, parameter k.

Output: a data-structure that on failure of any set F of k edges outputs:

Strongly-connected-components **(SCCs)** of $G\setminus F$

Prior Work:

[Italiano et al. (2017)]:

- k=1 (single failure)
- An oracle of $O(n)$ *size*
- Reporting time is $O(n)$

Problem 2: SCC Oracle

Input: directed graph $G=(V,E)$, parameter k.

Output: a data-structure that on failure of any set F of k edges outputs:

Strongly-connected-components **(SCCs)** of $G \setminus F$

Prior Work:

[Italiano et al. (2017)]:

- k=1 (single failure)
- An oracle of $O(n)$ *size*
- Reporting time is $O(n)$

Our Results for general k:

Problem 2: SCC Oracle

Input: directed graph $G=(V,E)$, parameter k.

Output: a data-structure that on failure of any set F of k edges outputs:

Strongly-connected-components **(SCCs)** of $G \setminus F$

Prior Work:

[Italiano et al. (2017)]:

- k=1 (single failure)
- An oracle of $O(n)$ *size*
- Reporting time is $O(n)$

Our Results for general k:

Oracle:

Problem 2: SCC Oracle

Input: directed graph $G=(V,E)$, parameter k.

Output: a data-structure that on failure of any set F of k edges outputs:

Strongly-connected-components **(SCCs)** of $G \setminus F$

Prior Work:

[Italiano et al. (2017)]:

- k=1 (single failure)
- An oracle of $O(n)$ *size*
- Reporting time is $O(n)$

Our Results for general k:

Oracle: $O_k(n^2)$ size

Problem 2: SCC Oracle

Input: directed graph $G=(V,E)$, parameter k.

Output: a data-structure that on failure of any set F of k edges outputs:

Strongly-connected-components **(SCCs)** of $G \setminus F$

Prior Work:

[Italiano et al. (2017)]:

- k=1 (single failure)
- An oracle of $O(n)$ *size*
- Reporting time is $O(n)$

Our Results for general k:

Oracle: $O_k(n^2)$ size

Reporting time:

Problem 2: SCC Oracle

Input: directed graph $G=(V,E)$, parameter k.

Output: a data-structure that on failure of any set F of k edges outputs:

Strongly-connected-components **(SCCs)** of $G \setminus F$

Prior Work:

[Italiano et al. (2017)]:

- k=1 (single failure)
- An oracle of $O(n)$ *size*
- Reporting time is $O(n)$

Our Results for general k:

Oracle: $O_k(n^2)$ size

Reporting time: $O_k(n)$

Problem 2: SCC Oracle

Proof Snippet

Bottleneck: SCCs intersecting fixed path P

Problem 2: SCC Oracle

Proof Snippet

Bottleneck: SCCs intersecting fixed path P

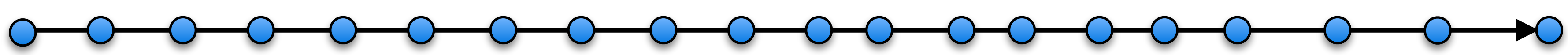
Lemma:

If we can compute **SCCs in G\F intersecting a path "P" in $F(n,k)$ time**, then, we can compute ***ALL* the SCCs of G\F in $O(F(n,k) \log n)$** time.

Problem 2: SCC Oracle

Proof Snippet

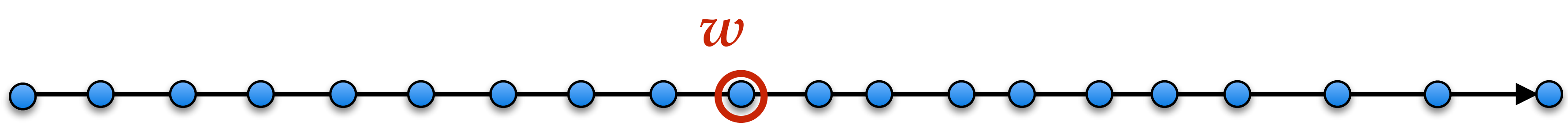
Bottleneck: SCCs intersecting fixed path P



Problem 2: SCC Oracle

Proof Snippet

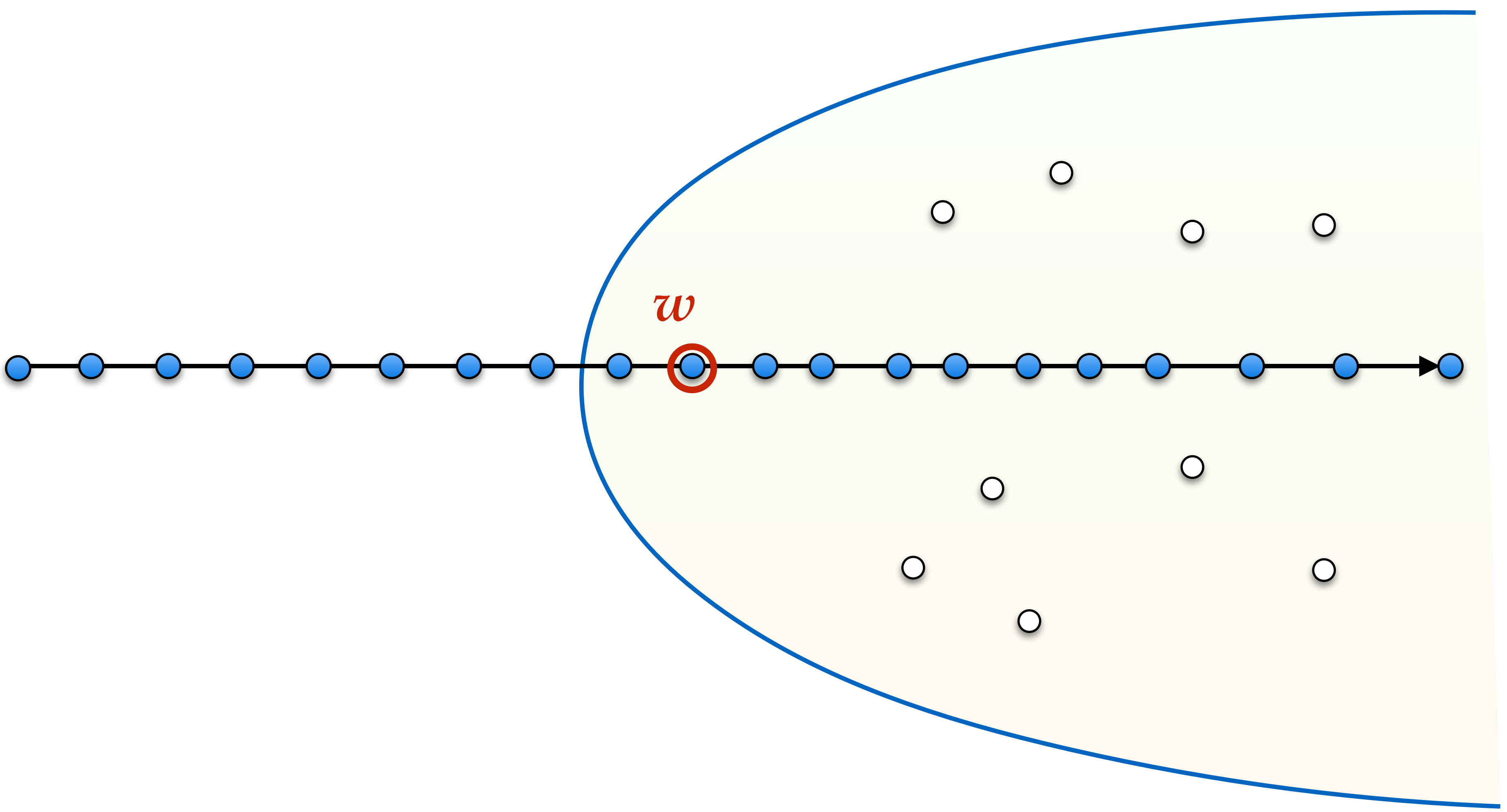
Bottleneck: SCCs intersecting fixed path P



Problem 2: SCC Oracle

Proof Snippet

Bottleneck: SCCs intersecting fixed path P

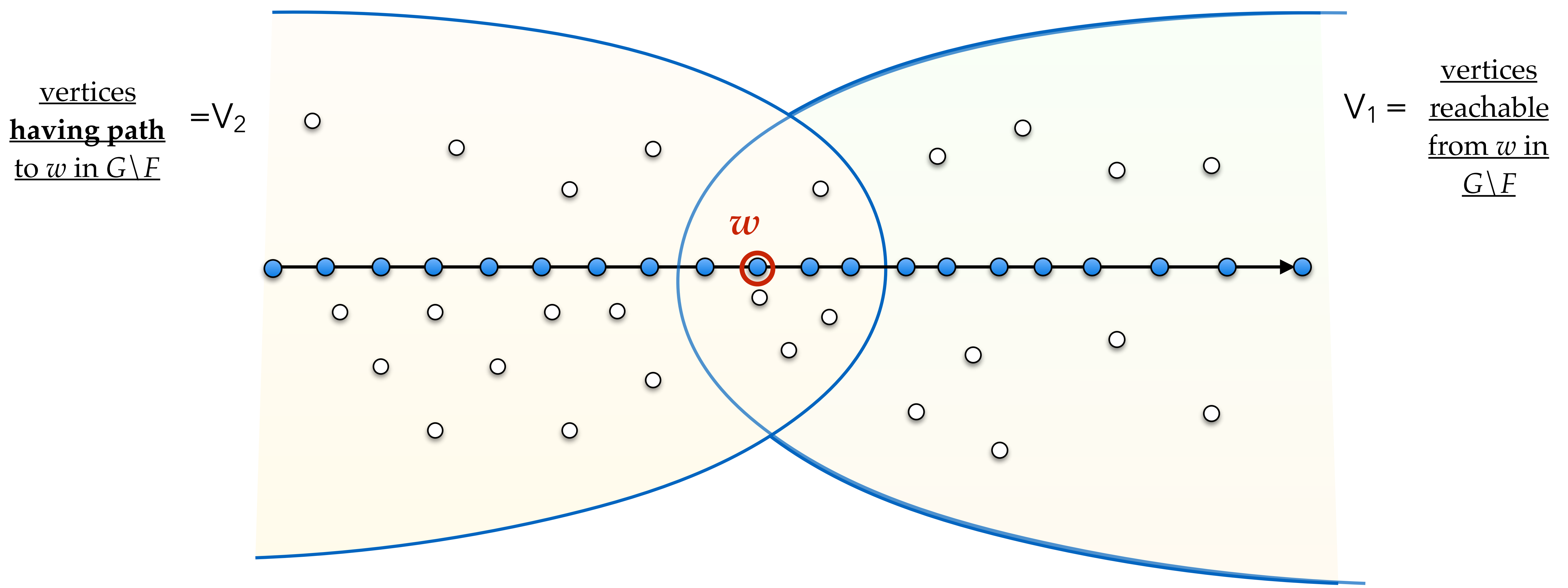


$V_1 =$ vertices reachable from w in $G \setminus F$

Problem 2: SCC Oracle

Proof Snippet

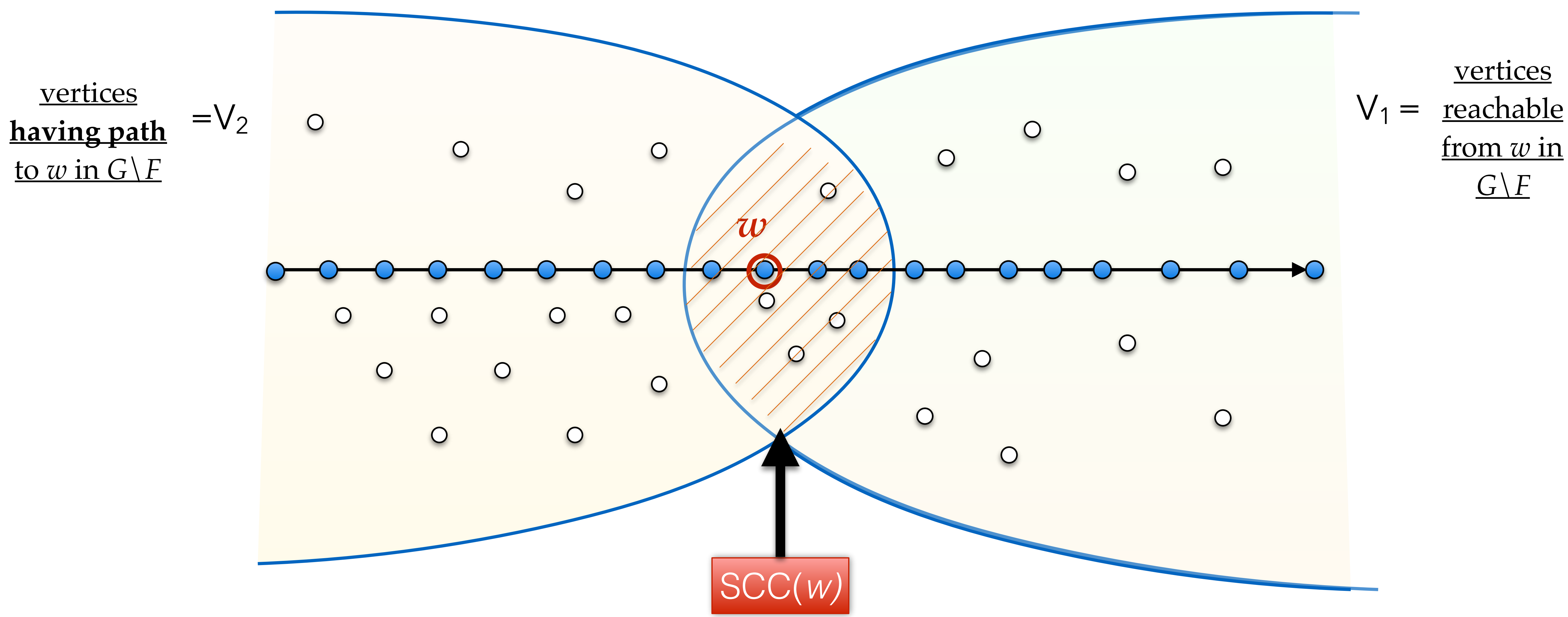
Bottleneck: SCCs intersecting fixed path P



Problem 2: SCC Oracle

Proof Snippet

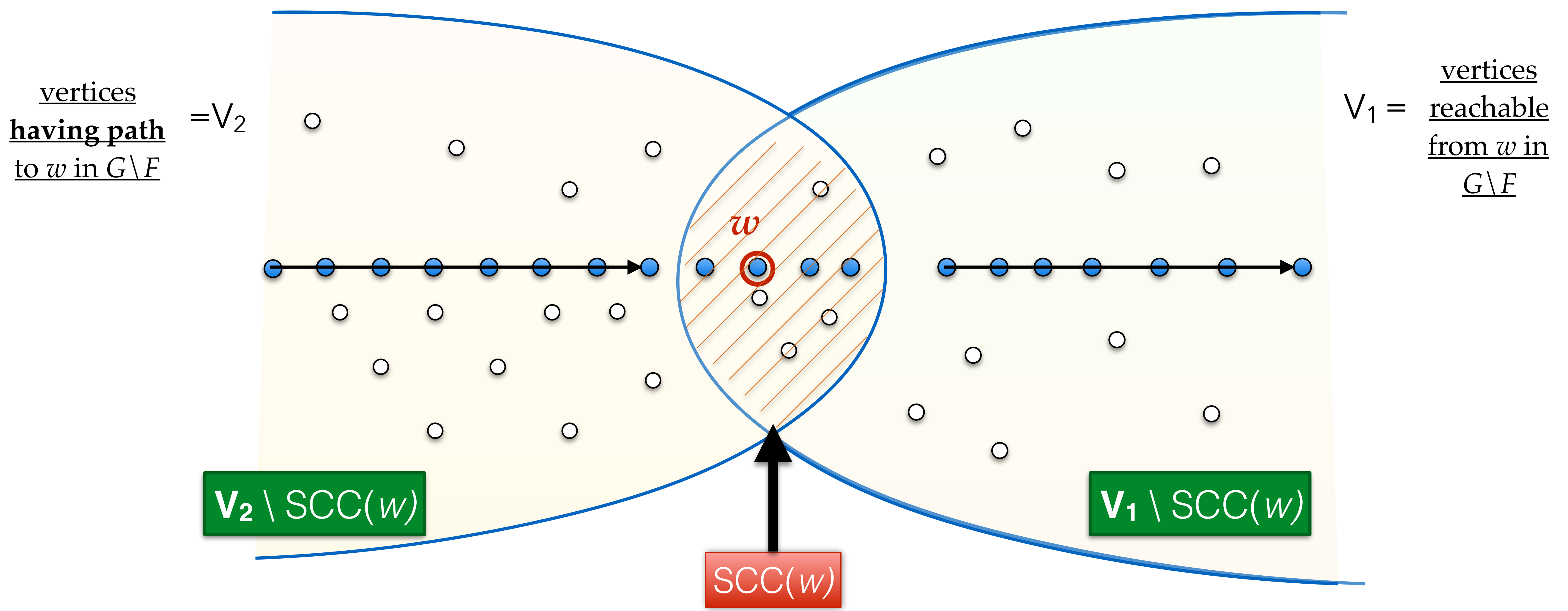
Bottleneck: SCCs intersecting fixed path P



Problem 2: SCC Oracle

Proof Snippet

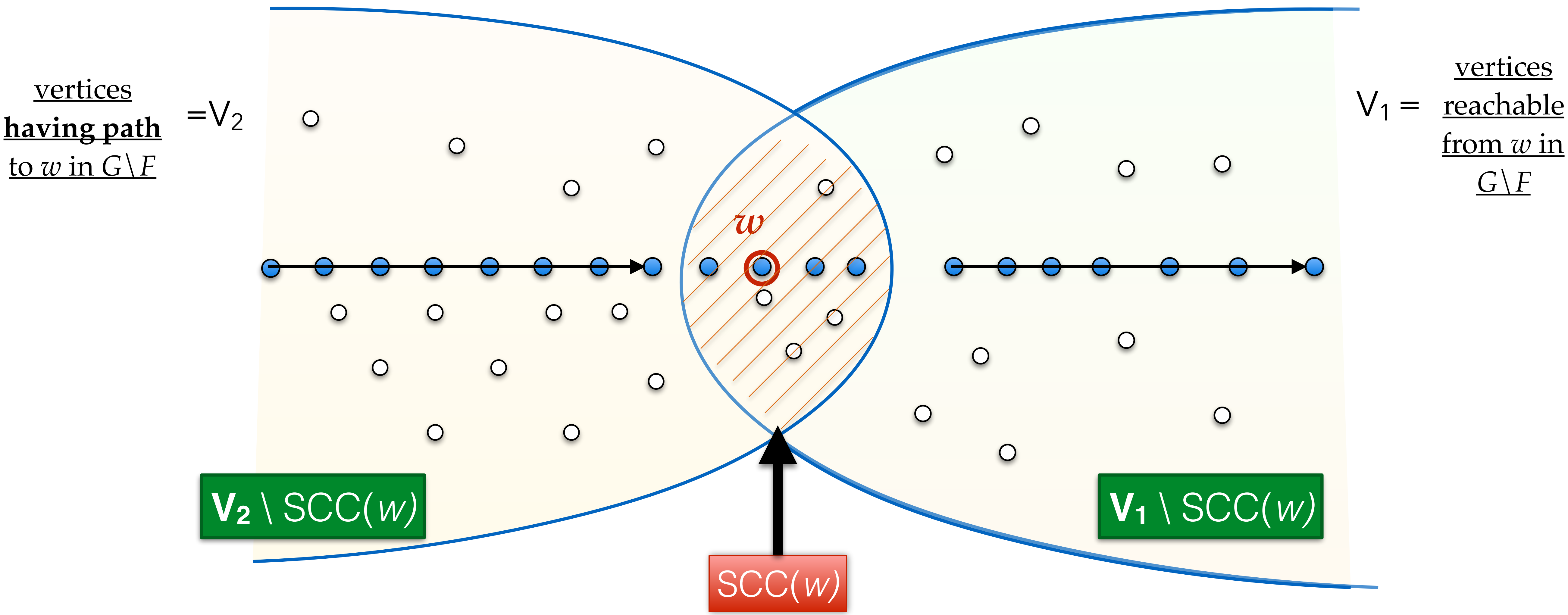
Bottleneck: SCCs intersecting fixed path P



Problem 2: SCC Oracle

Proof Snippet

Bottleneck: SCCs intersecting fixed path P

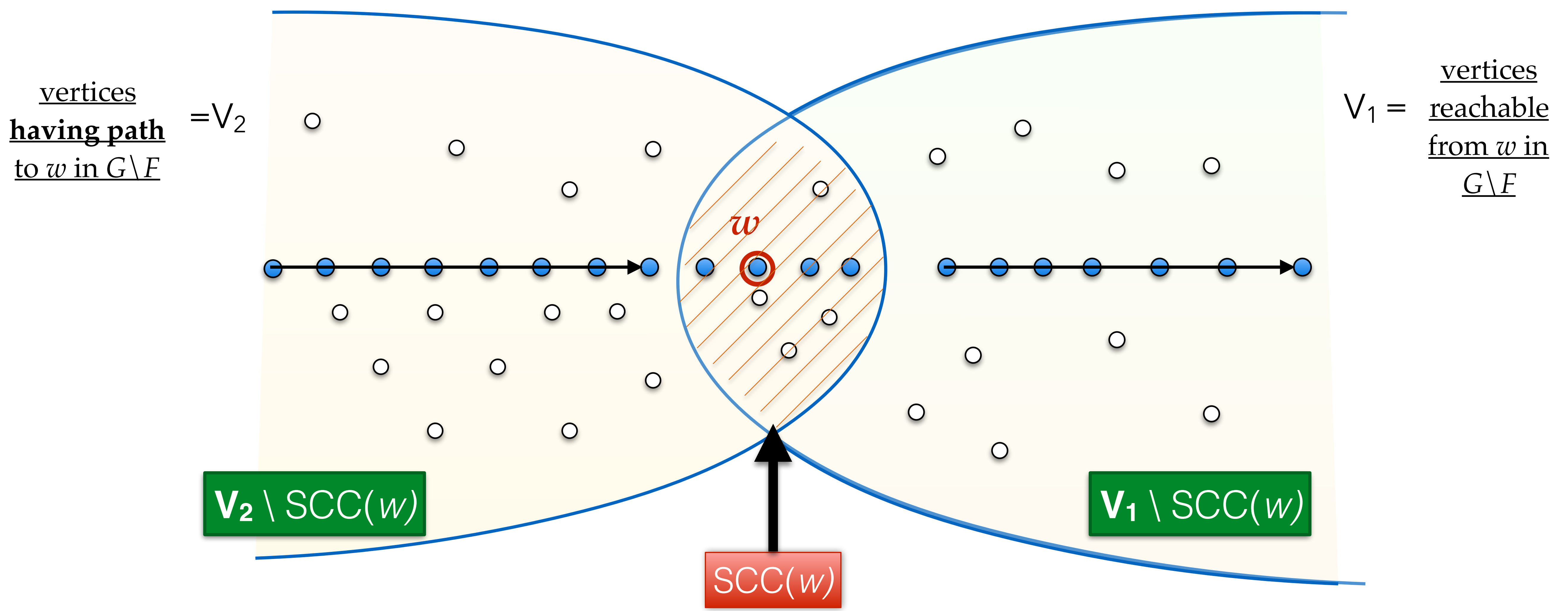


In $O(2^k n)$ time — divide problem into two sub-problems

Problem 2: SCC Oracle

Proof Snippet

Bottleneck: SCCs intersecting fixed path P



In $O(2^k n)$ time — divide problem into two sub-problems

Recursively solve in $O(2^k n \log |P|)$ time

Problem 2: SCC Oracle

Proof Snippet

Computing all SCCs

Lemma:

If we can compute **SCCs in G\F intersecting a path "P" in $F(n,k)$ time**, then, we can compute ***ALL* the SCCs of G\F in $O(F(n,k)\log n)$** time.

Problem 2: SCC Oracle

Proof Snippet

Computing all SCCs

<u>Lemma:</u>

If we can compute **SCCs in G\F intersecting a path "P" in $F(n,k)$ time**, then, we can compute ***ALL* the SCCs of G\F in O($F(n,k)$) log n)** time.

<u>**Main Result:**</u>

For any set F of k failures, we can compute SCCs of graph $G\backslash F$ in
<u>$O(2^k n \log^2 n)$</u> time.

Problem 2: SCC Oracle

Proof Snippet

Computing all SCCs

Lemma:

If we can compute **SCCs in G\F intersecting a path "P" in $F(n,k)$ time**, then, we can compute ***ALL*** **the SCCs of G\F in $O(F(n,k) \log n)$** time.

Main Result:

For any set F of k failures, we can compute SCCs of graph $G\backslash F$ in $\underline{O(2^k n \log^2 n)}$ time.

Size of the oracle is $O(2^k n^2)$.

Thank You