Constant Approximation of Min-Distances in Near-Linear Time

Shiri Chechik

Tianyi Zhang

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

Distance parameters in graphs

$G = (V, E, \omega)$ be a weighted directed graph

Distance parameters:

diameter, radius, eccentricity

- Eccentricity of any vertex
  = Maximum distance to other vertices
- Radius = Minimum eccentricity among all vertices
- Diameter = Maximum eccentricity among all vertices

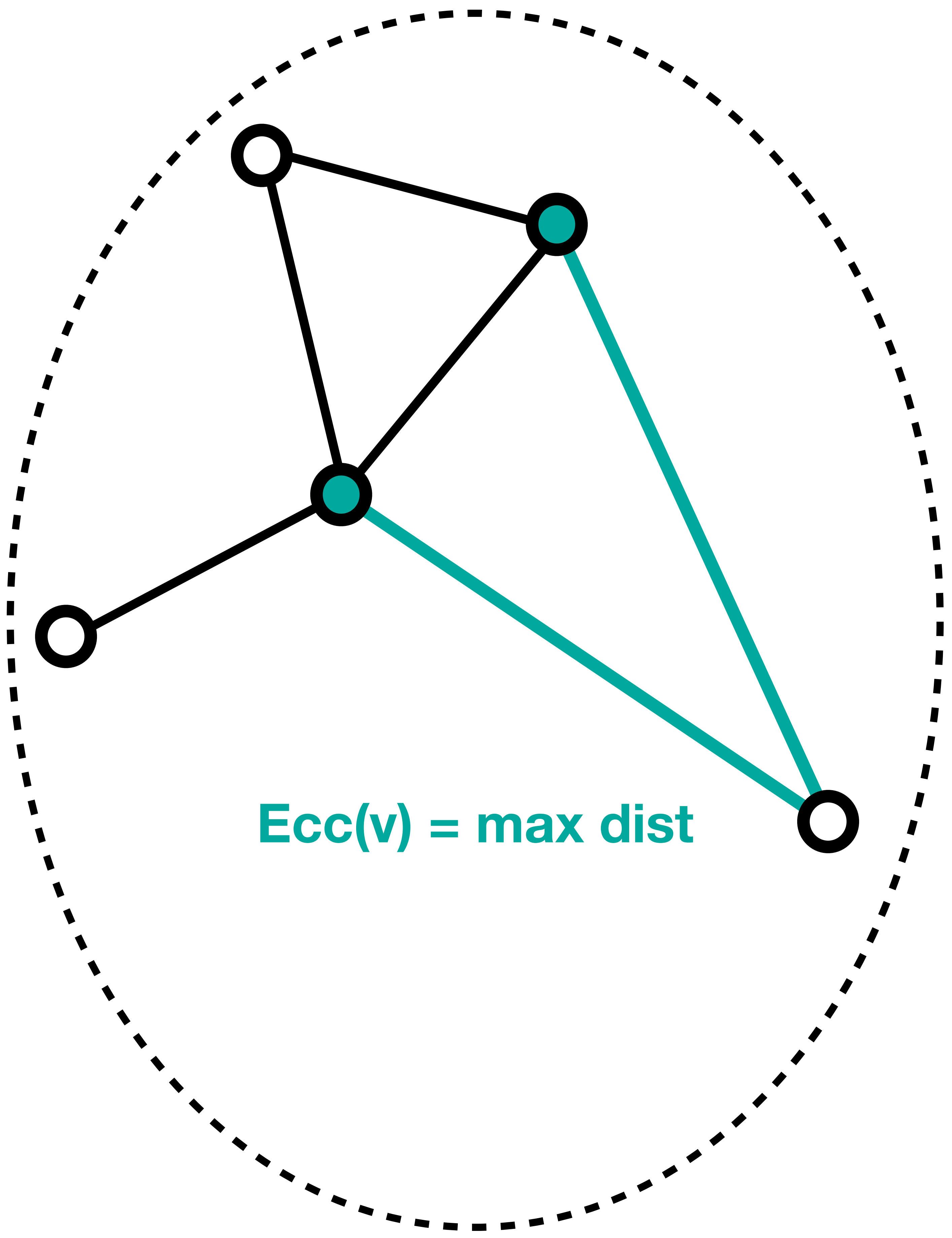
Distance parameters in graphs

$G = (V, E, \omega)$ be a weighted directed graph

Distance parameters:

diameter, radius, eccentricity

- **Eccentricity of any vertex = Maximum distance to other vertices**
- Radius = Minimum eccentricity among all vertices
- Diameter = Maximum eccentricity among all vertices



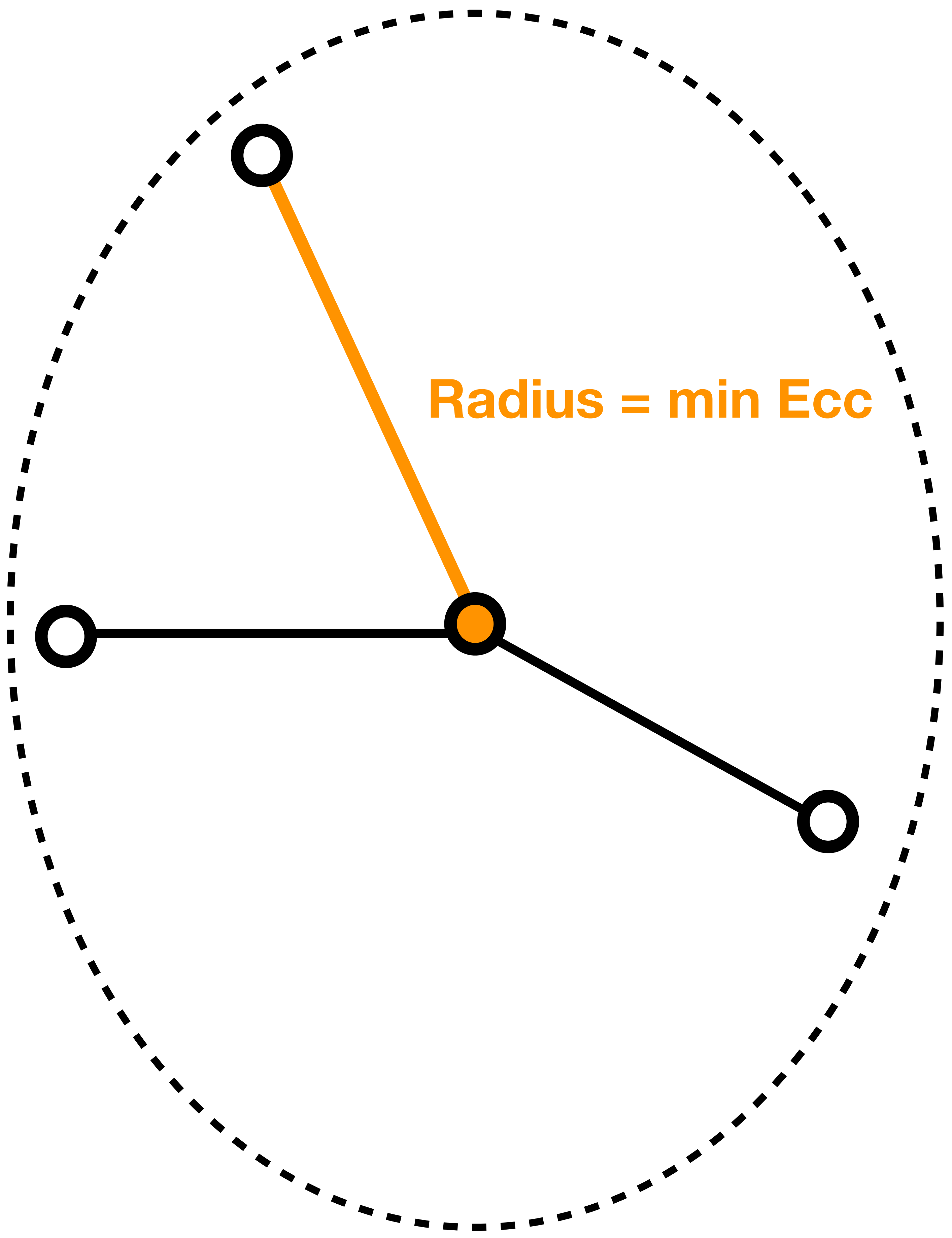
Distance parameters in graphs

$G = (V, E, \omega)$ be a weighted directed graph

<u>Distance parameters:</u>

diameter, radius, eccentricity

- Eccentricity of any vertex = Maximum distance to other vertices
- **Radius = Minimum eccentricity among all vertices**
- Diameter = Maximum eccentricity among all vertices



Distance parameters in graphs

$G = (V, E, \omega)$ be a weighted directed graph

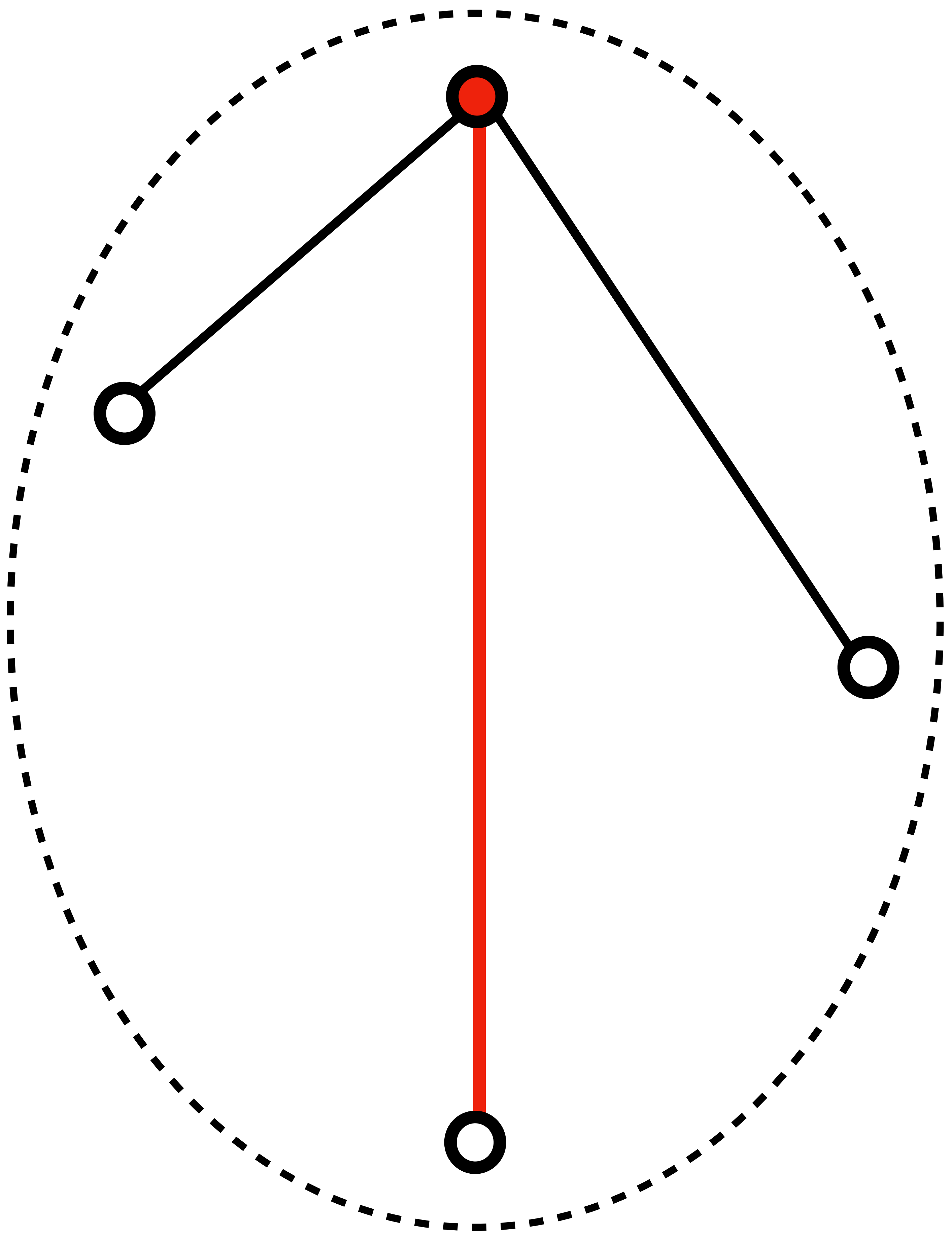
<u>Distance parameters:</u>

diameter, radius, eccentricity

- Eccentricity of any vertex
  = Maximum distance to other vertices

- Radius = Minimum eccentricity among all vertices

- **Diameter = Maximum eccentricity among all vertices**



Diameter = max Ecc

Variants of distances in digraphs

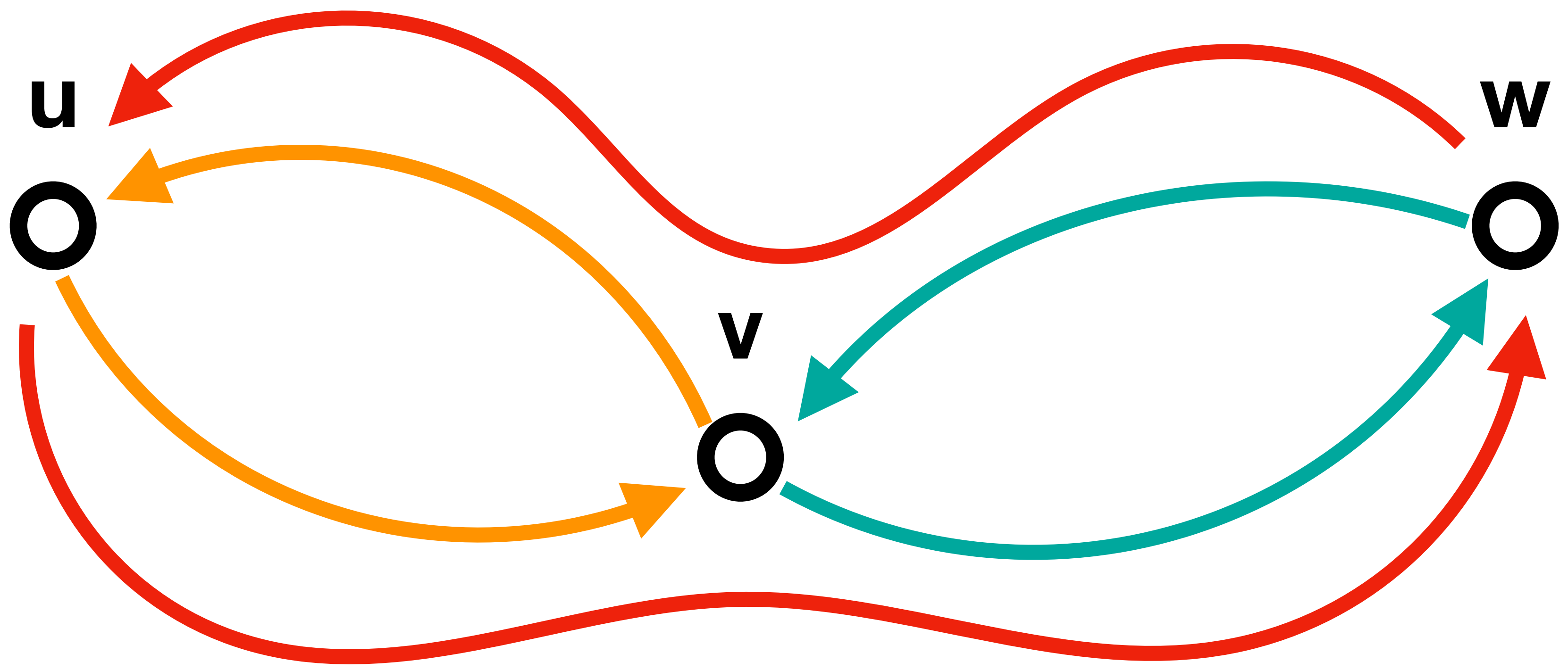
- In directed graphs, $\mathrm{dist}(u, v) \neq \mathrm{dist}(v, u)$ not symmetric
- **Round-trip distance** = $\mathrm{dist}(u, v) + \mathrm{dist}(v, u)$
- **Max-distance** = $\max\{\mathrm{dist}(u, v), \mathrm{dist}(v, u)\}$
- **Min-distance** = $\min\{\mathrm{dist}(u, v), \mathrm{dist}(v, u)\}$

Exactly computing distance parameters (**ecc**, **diam**, **rad**) requires $m^{2-\epsilon}$ time under **SETH** and **HSC**

Faster algorithms need to allow **approximations**

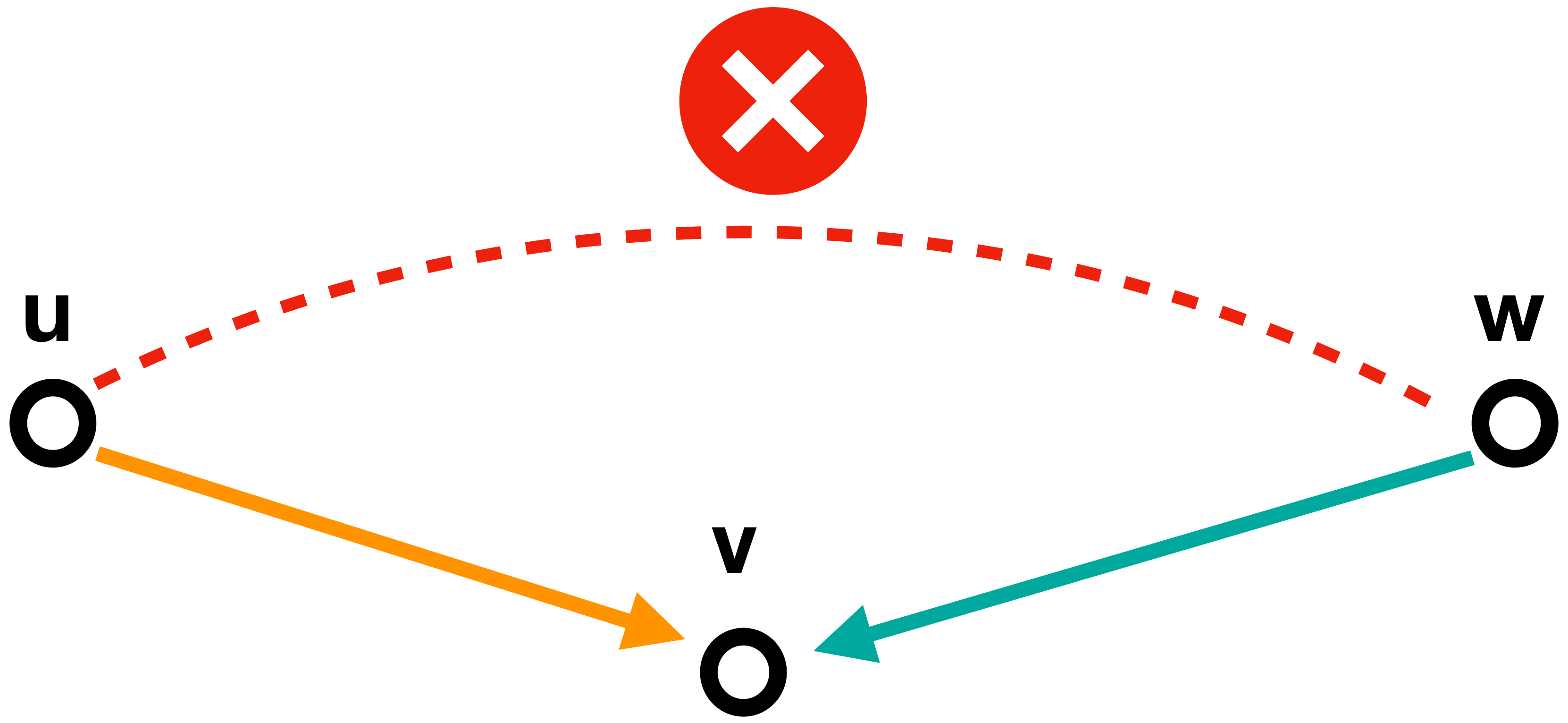
Variants of distances in digraphs

- Rndtrip-dist & max-dist satisfy **triangle inequalities**
- Min-dist violates **triangle inequalities**



round-trip distance

rnd(u, v) + rnd(v, w) $\geq$ rnd(u, w)

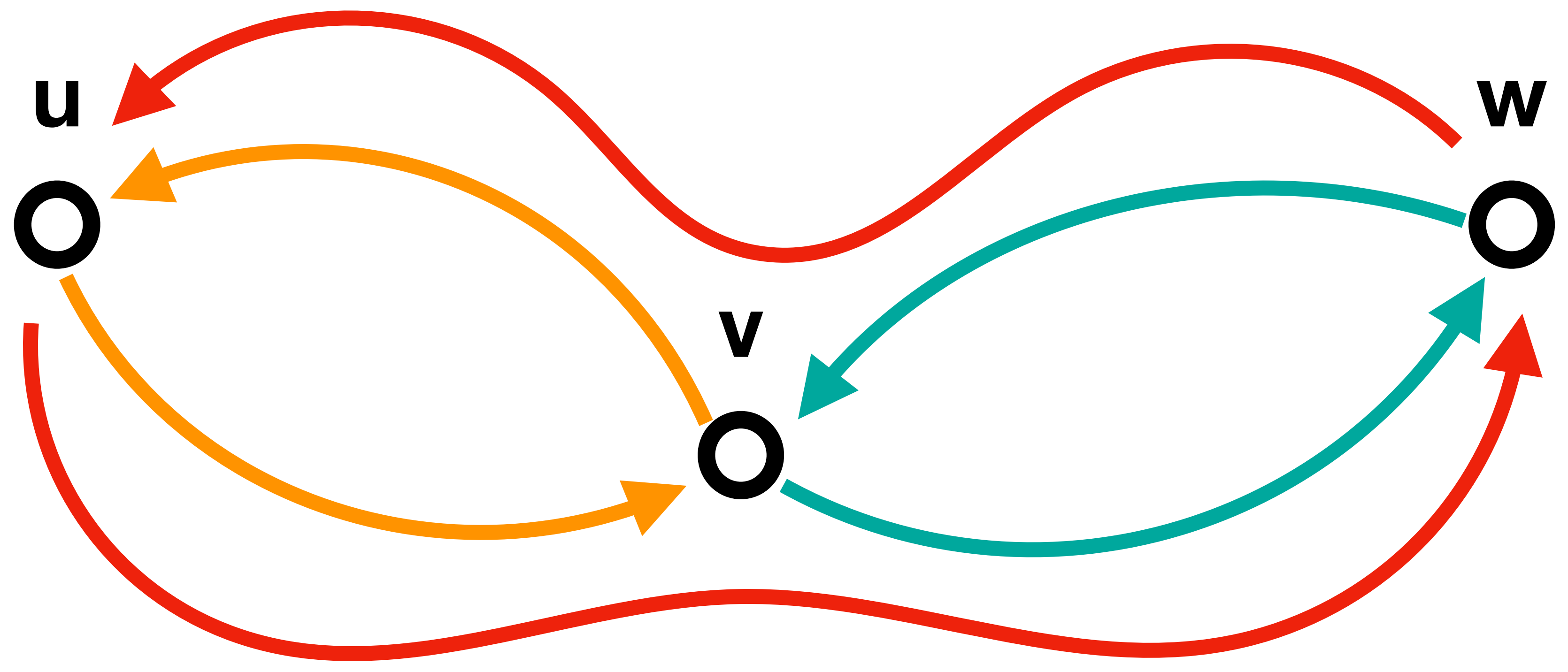


min-distance

min-dist(u, v) + min-dist(v, w)
< min-dist(u, w) = ∞

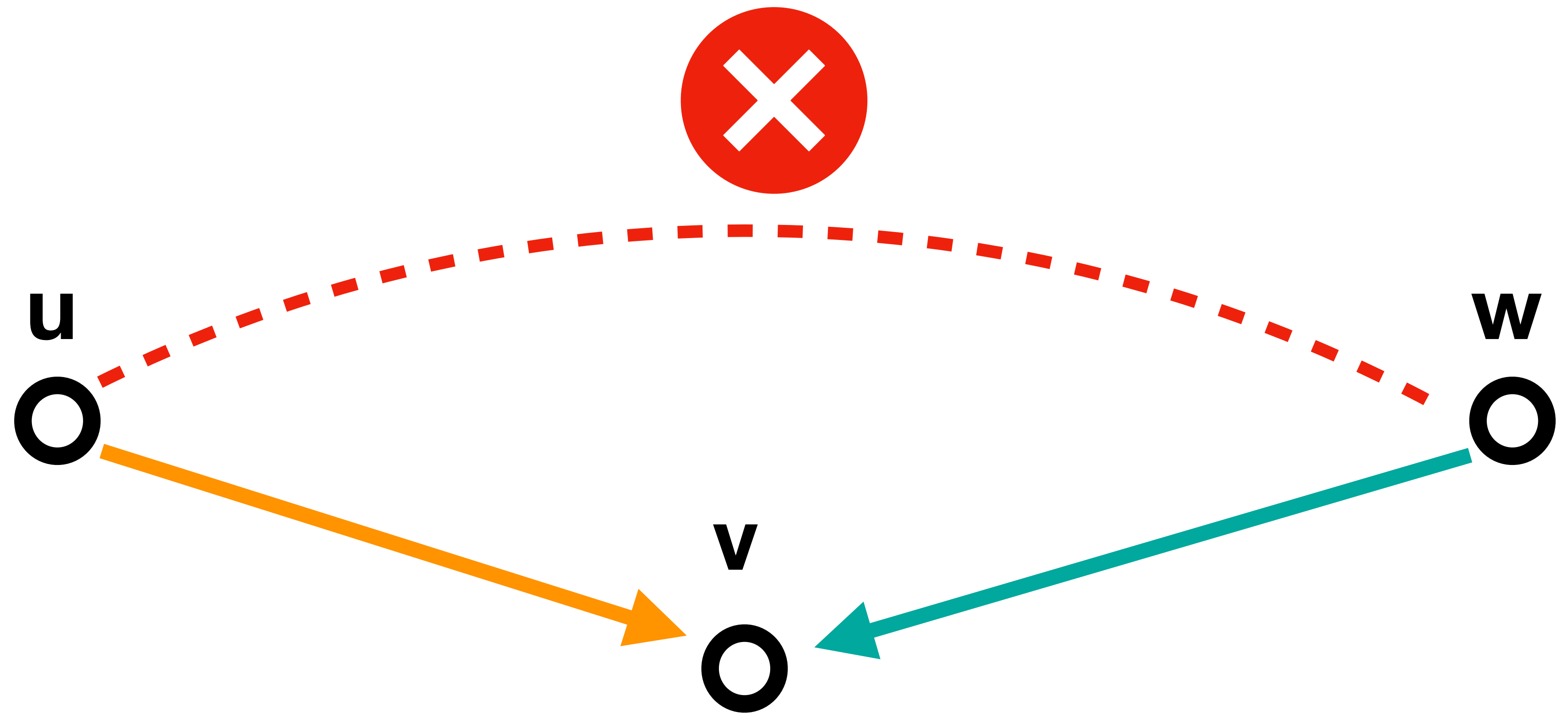
Variants of distances in digraphs

- Rndtrip-dist & max-dist satisfy **triangle inequalities**
- Min-dist violates **triangle inequalities**



round-trip distance

Ecc(v) is **2-approx** of round-trip diameter



min-distance

Ecc(v) is **∞-approx** of min-diameter

A short history

reference	which param?	approx	runtime	input type
AVW, 2016	min-diameter	2	m	acyclic
AVW, 2016	min-radius	3	$mn^{1/2}$	acyclic
DK, 2021	min-radius	k	$mn^{1/k}$	acyclic
DK, 2021	min-eccentricities	k + 0.001	$mn^{1/k}$	acyclic
open	**all param**	**O(1)**	**m**	**acyclic**

A short history

reference	which param?	approx	runtime	input type
DWV+, 2019	min-diameter	4k-5	$mn^{1/k}$	general
DWV+, 2019	min-radius	3	$mn^{1/2}$	general
DWV+, 2019	min-eccentricities	5.001	$mn^{1/2}$	general
open	**any param**	**O(1)**	**m**	**general**

Our results

reference	which param?	approx	runtime	input type
open	**all/any param**	**O(1)**	**m**	**acyclic / general**
new	min-diameter	4k-5 vs **4**	$mn^{1/k}$ vs **m**	general
new	min-radius	3 vs **4**	$mn^{1/2}$ vs **m**	general
new	min-eccentricities	5.001	$mn^{1/2}$ vs **m**	general
new	min-radius	k vs **3**	$mn^{1/k}$ vs **m**	acyclic
new	min-eccentricities	k+0.01 vs **3.01**	$mn^{1/k}$ vs **m**	acyclic

Our results

reference	which param?	approx	runtime	input type
open	all/any param	O(1)	m	acyclic / general
new	min-diameter	4k-5 vs 4	$mn^{1/k}$ vs m	general
new	min-radius	3 vs 4	$mn^{1/2}$ vs m	general
new	min-eccentricities	5.001	$mn^{1/2}$ vs m	general
new	min-radius	k vs 3	$mn^{1/k}$ vs m	acyclic
new	min-eccentricities	k+0.01 vs 3.01	$mn^{1/k}$ vs m	acyclic

This talk

Min-diameter
in general digraphs

First attempt

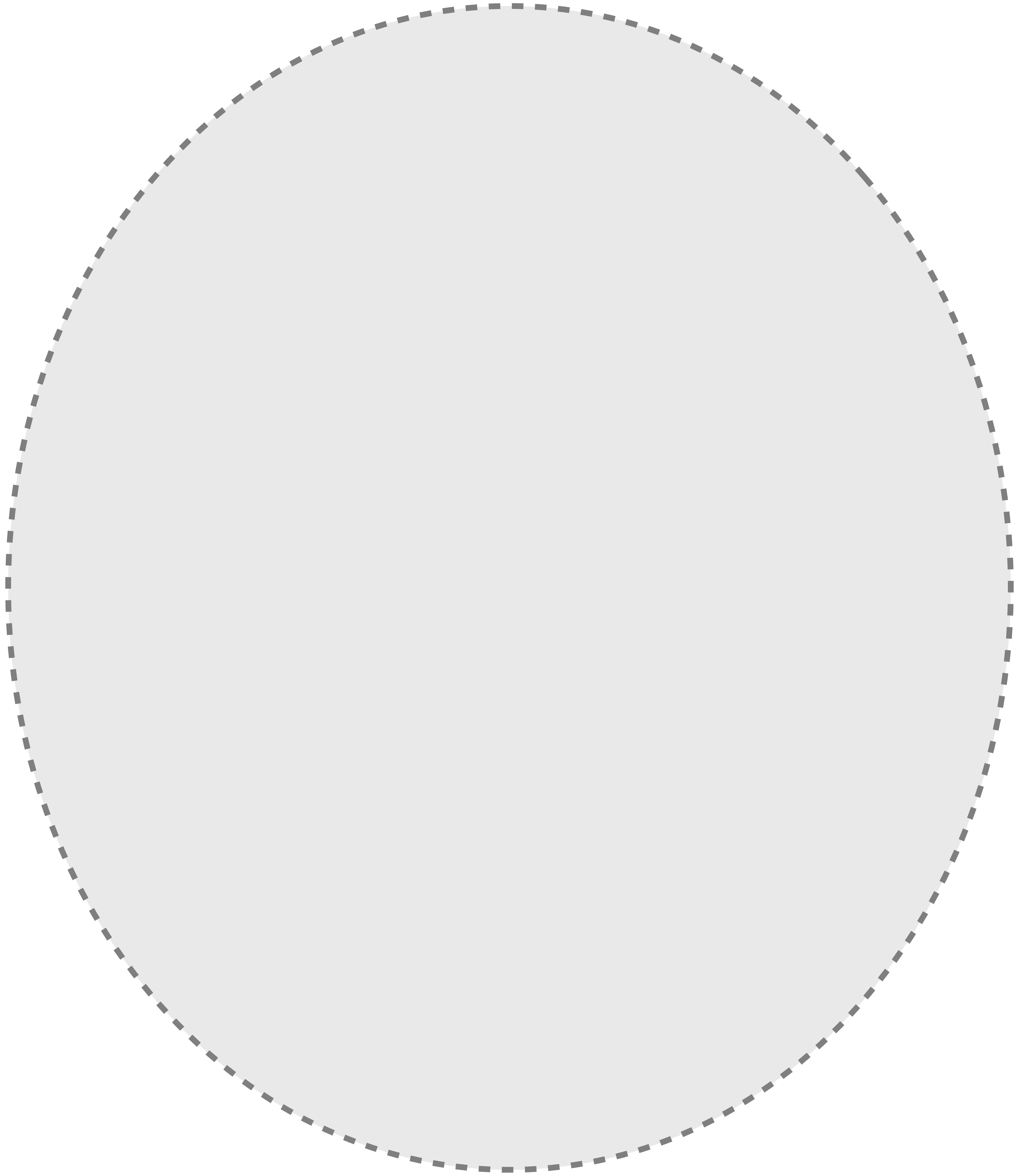
Goal: decide min-diam $<$D or $>$D/4

1. Pick a random vertex t

2. Compute SSSP in to and from t

3. Define two sets:
   $$U_+ = \{u \mid \mathrm{dist}(t, u) < D/4\}$$
   $$U_- = \{u \mid \mathrm{dist}(u, t) < D/4\}$$

4. Recurse on $G[U_+]$ and $G[U_-]$



First attempt

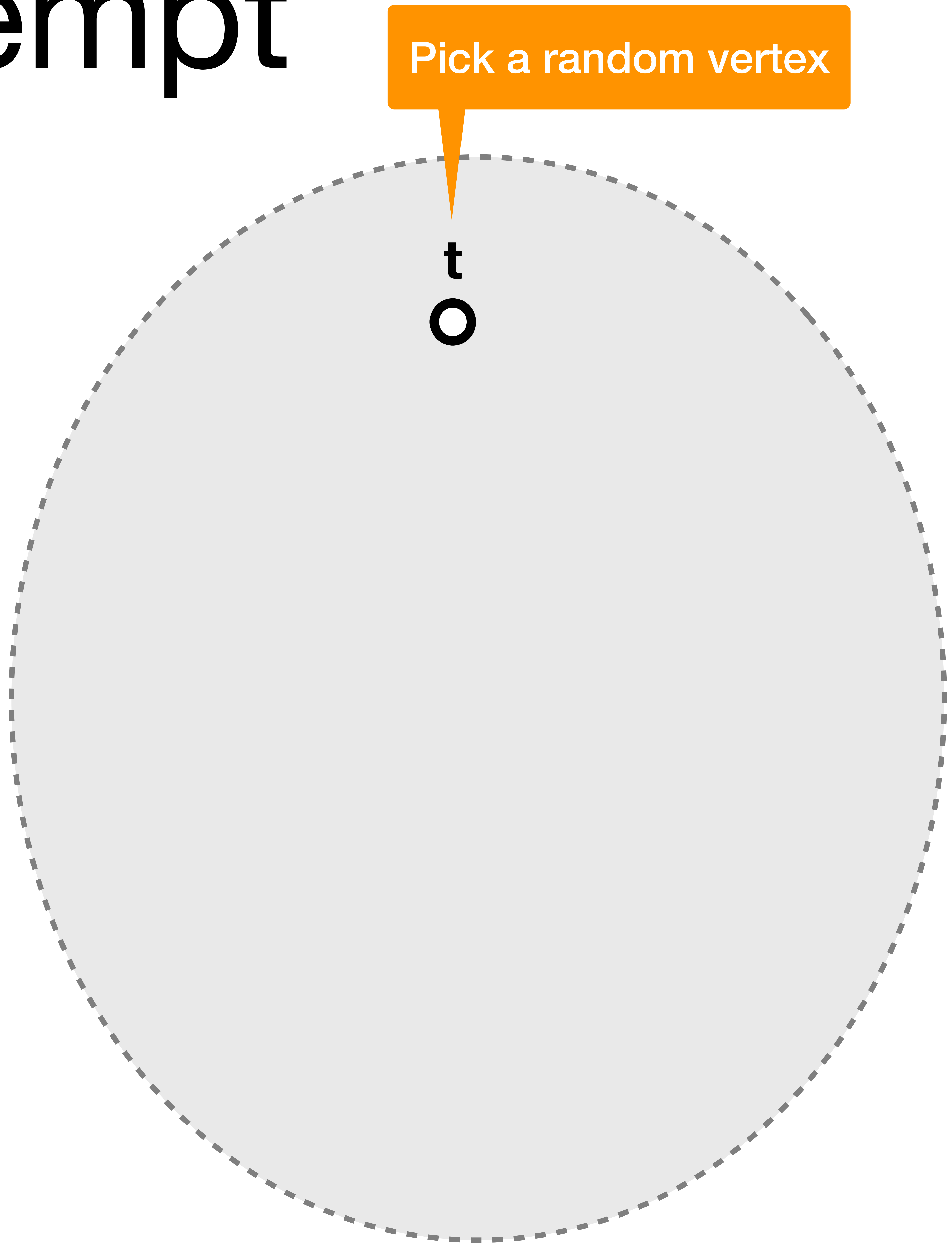
<u>Goal:</u> decide min-diam <D or >D/4

1. **Pick a random vertex t**

2. Compute SSSP in to and from t

3. Define two sets:
   $$U_+ = \{u \mid \mathrm{dist}(t, u) < D/4\}$$
   $$U_- = \{u \mid \mathrm{dist}(u, t) < D/4\}$$

4. Recurse on $G[U_+]$ and $G[U_-]$



First attempt

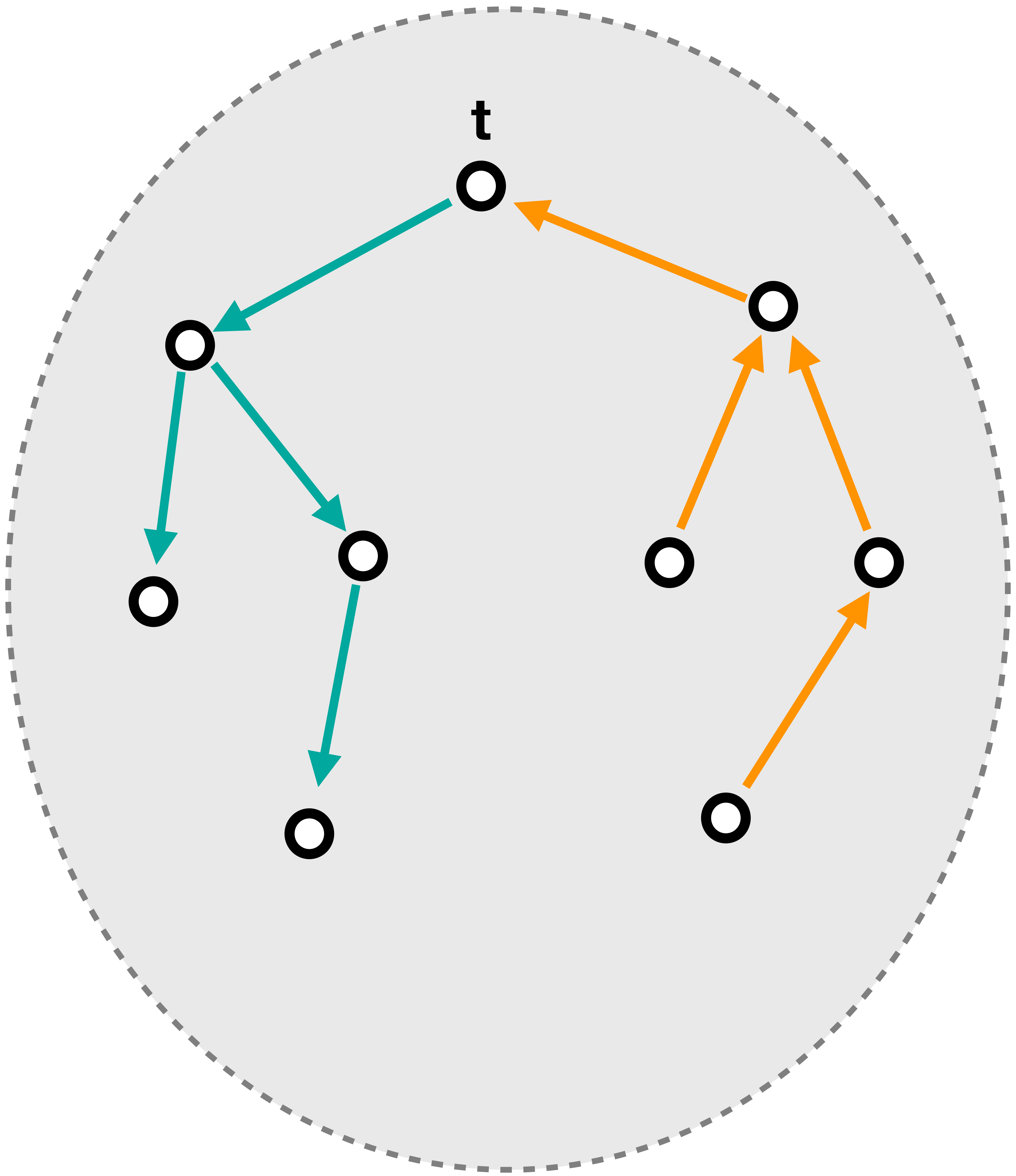
<u>Goal:</u> decide min-diam <D or >D/4

1. Pick a random vertex t

2. **Compute SSSP in to and from t**

3. Define two sets:
$$U_+ = \{u \mid \mathrm{dist}(t, u) < D/4\}$$
$$U_- = \{u \mid \mathrm{dist}(u, t) < D/4\}$$

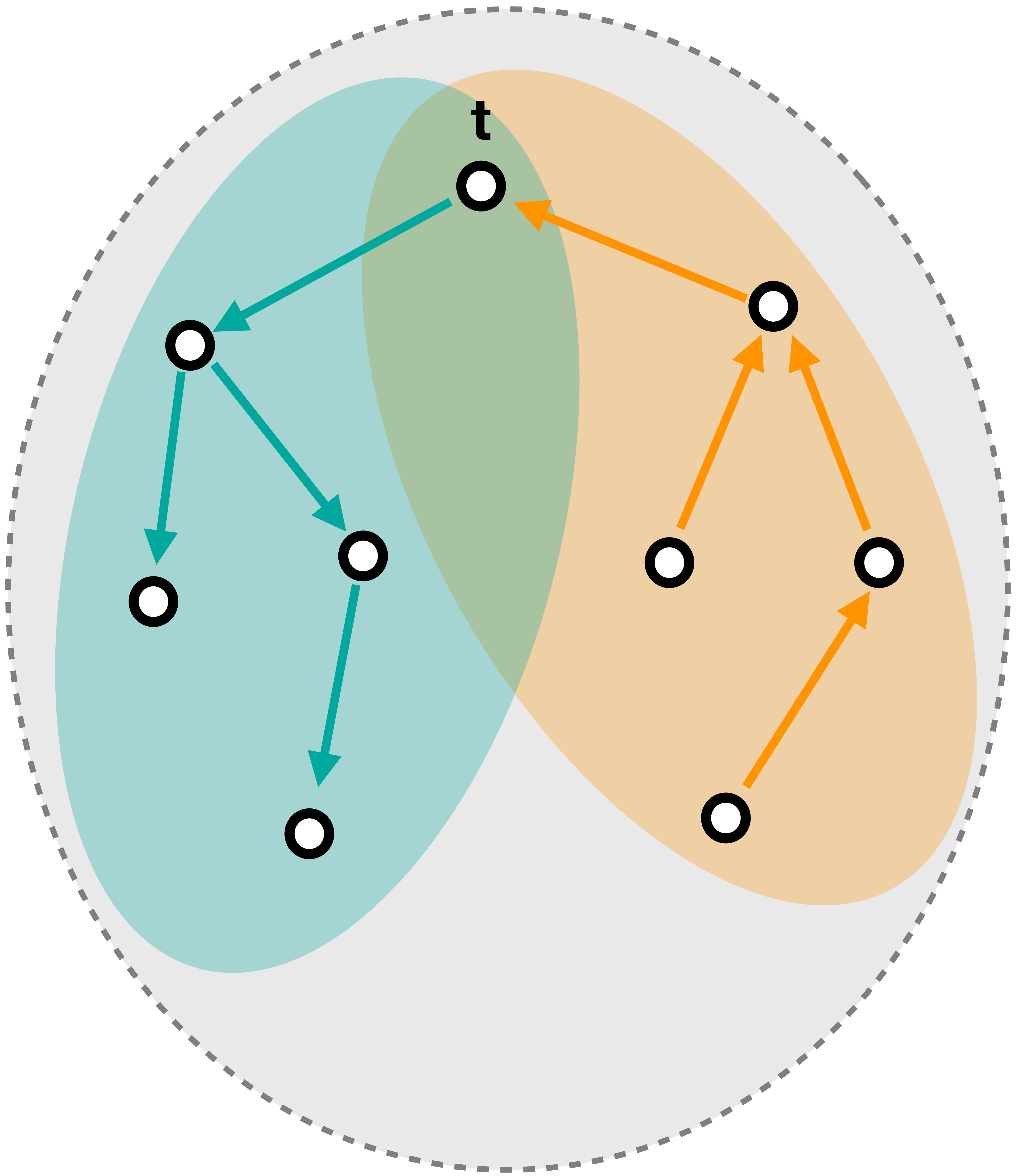
4. Recurse on $G[U_+]$ and $G[U_-]$



First attempt

<u>Goal:</u> decide min-diam <D or >D/4

1. Pick a random vertex t
2. Compute SSSP in to and from t
3. **Define two sets:**
   $U_+ = \{u \mid \text{dist}(t, u) < D/4\}$
   $U_- = \{u \mid \text{dist}(u, t) < D/4\}$
4. Recurse on $G[U_+]$ and $G[U_-]$



First attempt

Goal: decide min-diam <D or >D/4

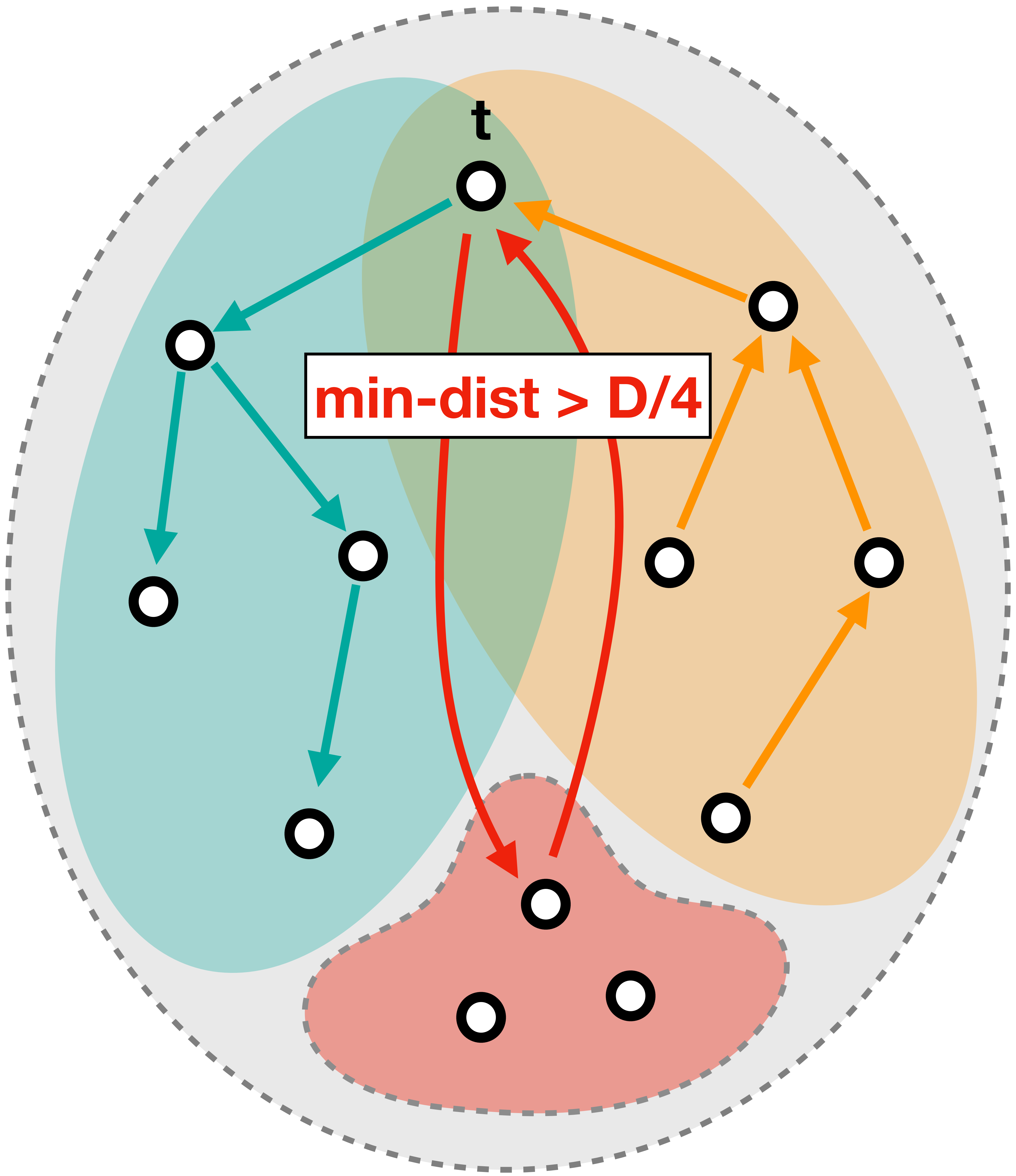
1. Pick a random vertex t
2. Compute SSSP in to and from t
3. **Define two sets:**

   $U_+ = \{u \mid \text{dist}(t,u) < D/4\}$

   $U_- = \{u \mid \text{dist}(u,t) < D/4\}$

   If $U_+ \cup U_- \neq V$
   then claim min-diam >D/4

4. Recurse on $G[U_+]$ and $G[U\_]$



First attempt

<u>Goal:</u> decide min-diam <D or >D/4

1. Pick a random vertex t

2. Compute SSSP in to and from t

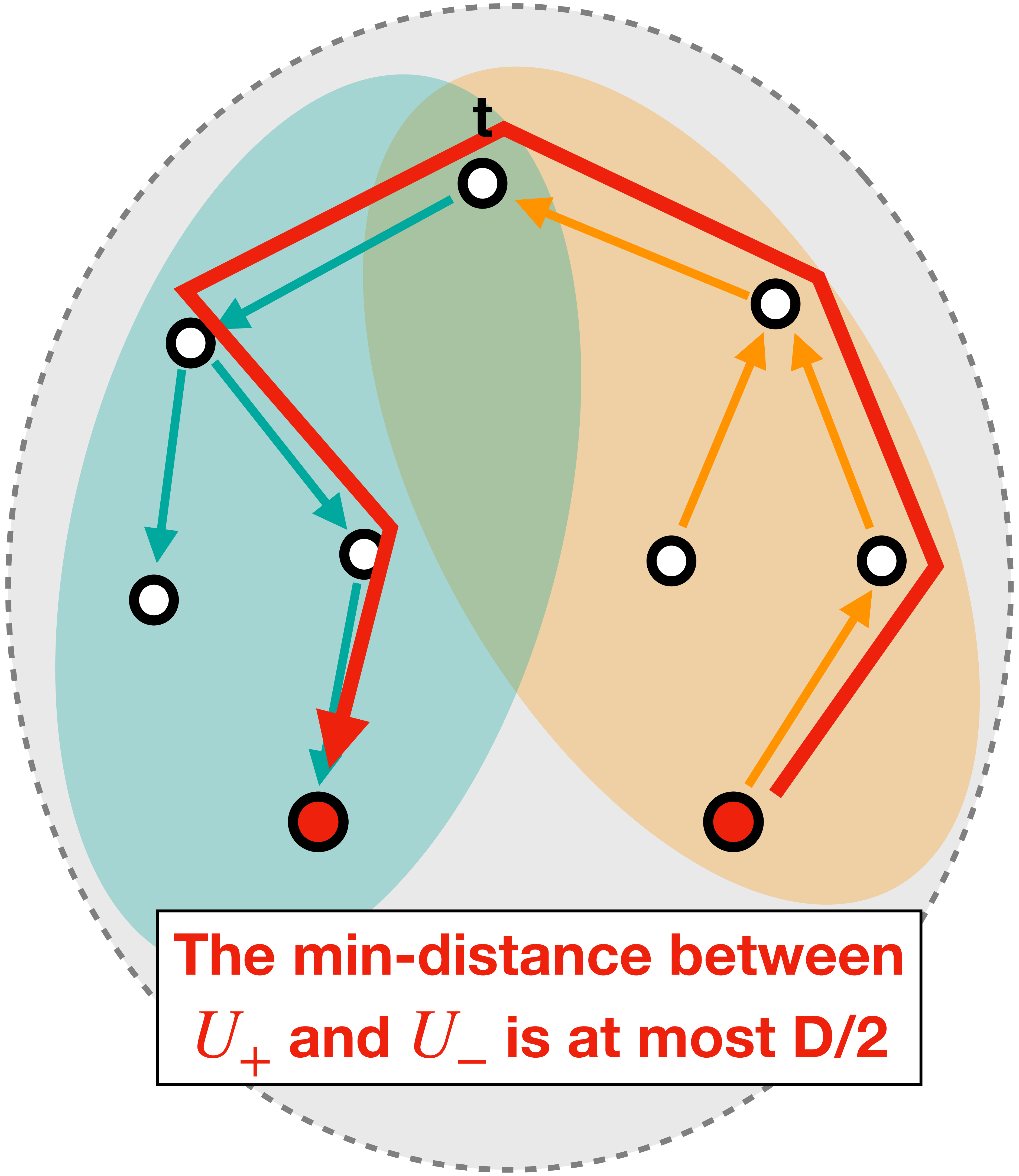
3. **Define two sets:**

If $U_+ \cup U_- = V$

Endpoints of **min-diam** should

belong to the same side

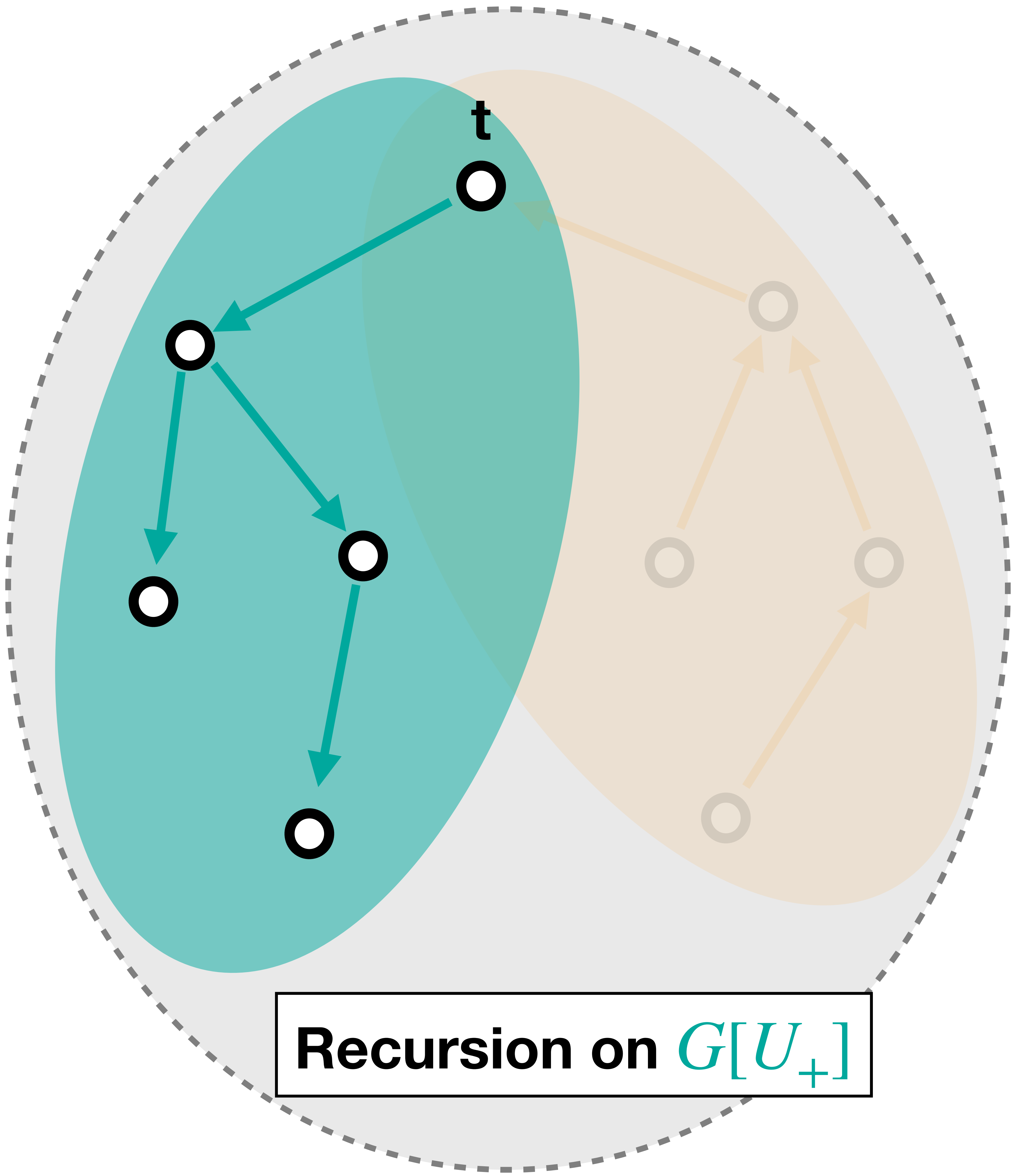
4. Recurse on $G[U_+]$ and $G[U_-]$



First attempt

Goal: decide min-diam <D or >D/4

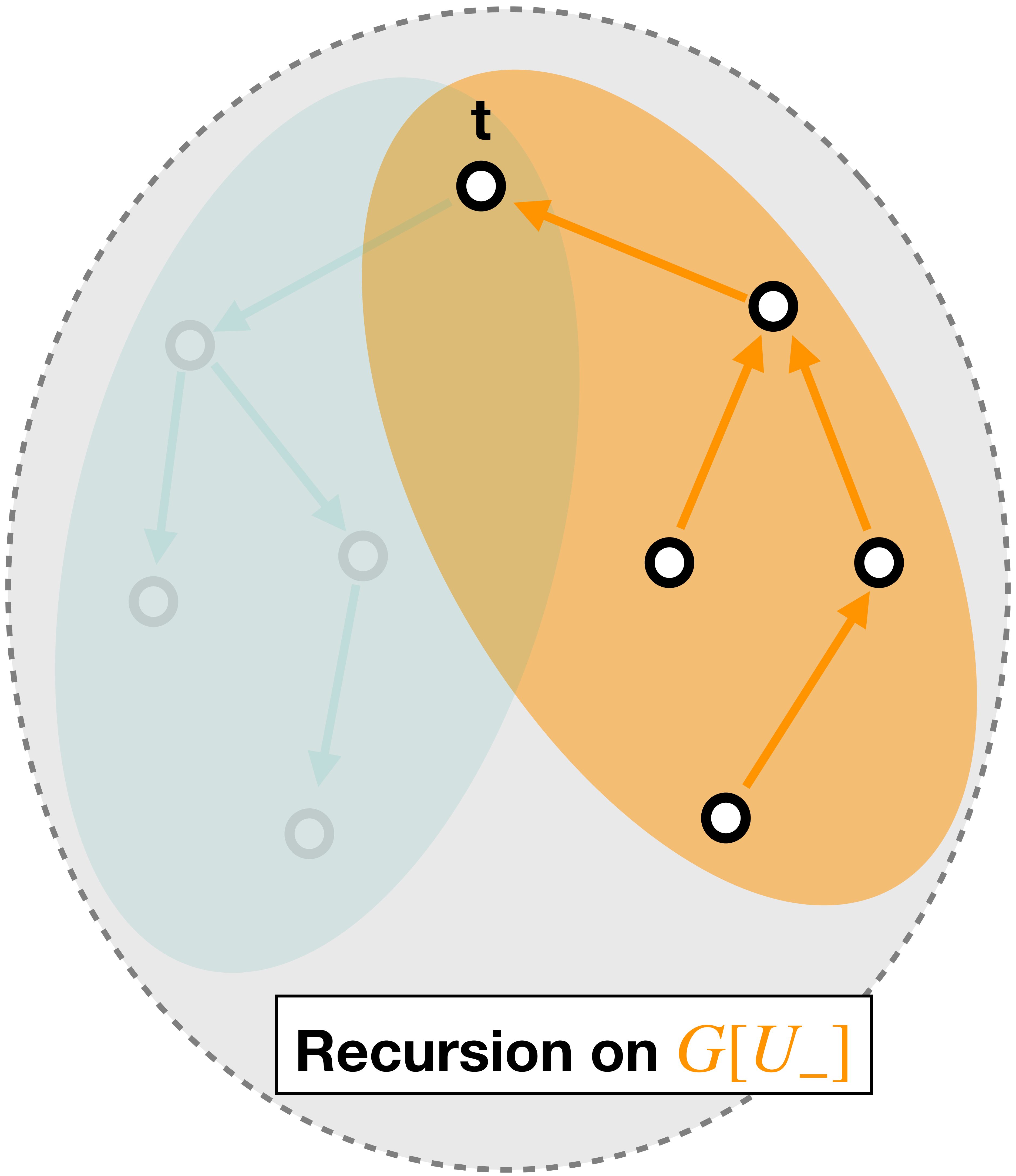
1. Pick a random vertex t
2. Compute SSSP in to and from t
3. Define two sets:
   $U_+ = \{u \mid \mathrm{dist}(t, u) < D/4\}$
   $U_- = \{u \mid \mathrm{dist}(u, t) < D/4\}$
4. **Recurse on** $G[U_+]$ **and** $G[U_-]$



First attempt

<u>Goal:</u> decide min-diam <D or >D/4

1. Pick a random vertex t
2. Compute SSSP in to and from t
3. Define two sets:
   $U_+ = \{u \mid \mathrm{dist}(t, u) < D/4\}$
   $U\_ = \{u \mid \mathrm{dist}(u, t) < D/4\}$
4. **Recurse on** $G[U_+]$ **and** $G[U\_]$

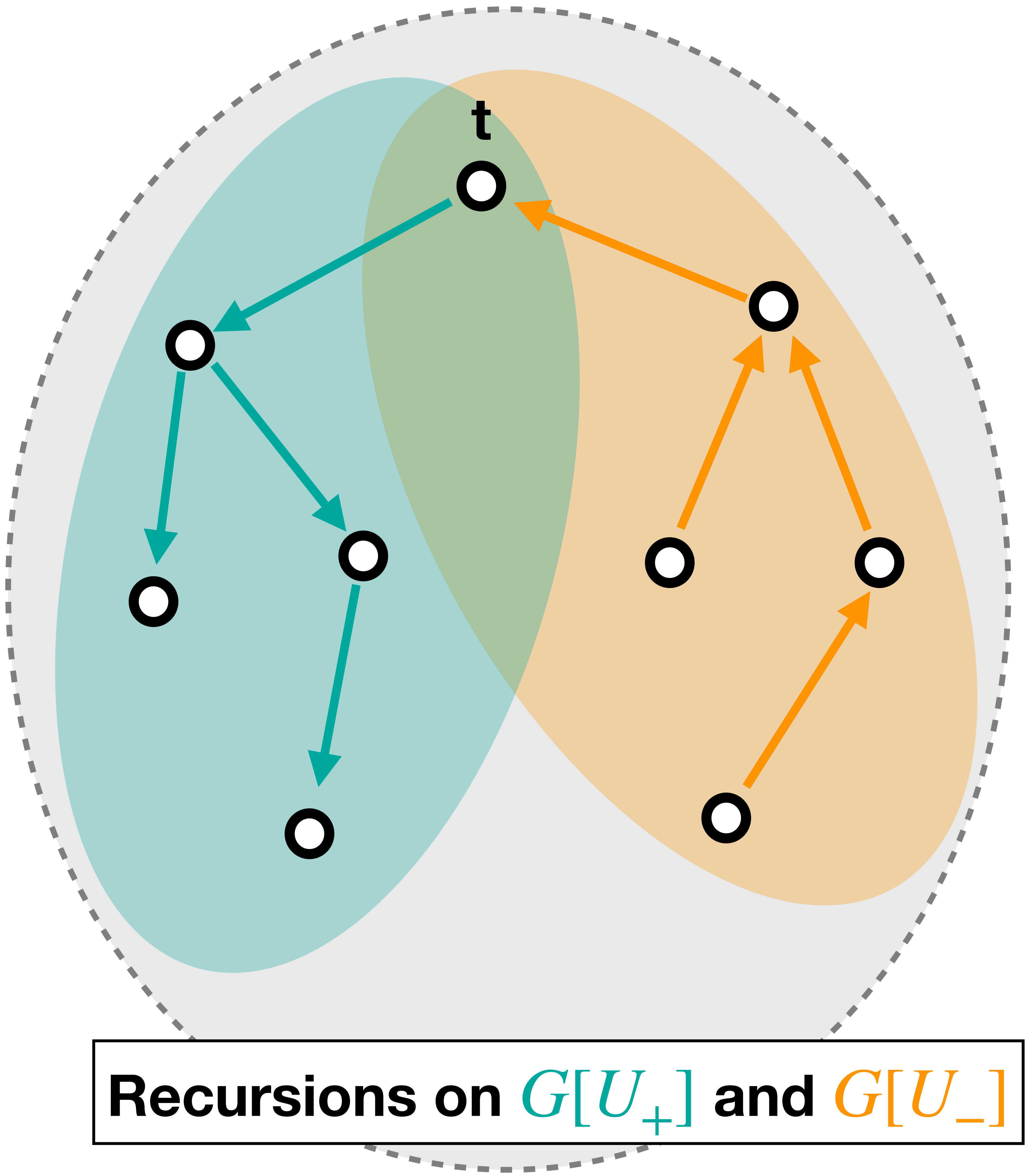


First attempt

<u>Some issues:</u>

- **Soundness**
  A large min-distance within $G[U_+]$ does not imply a large min-distance in $G[V]$

- **Runtime**
  $G[U_+]$ and $G[U_-]$ are usually intersecting, runtime can be high

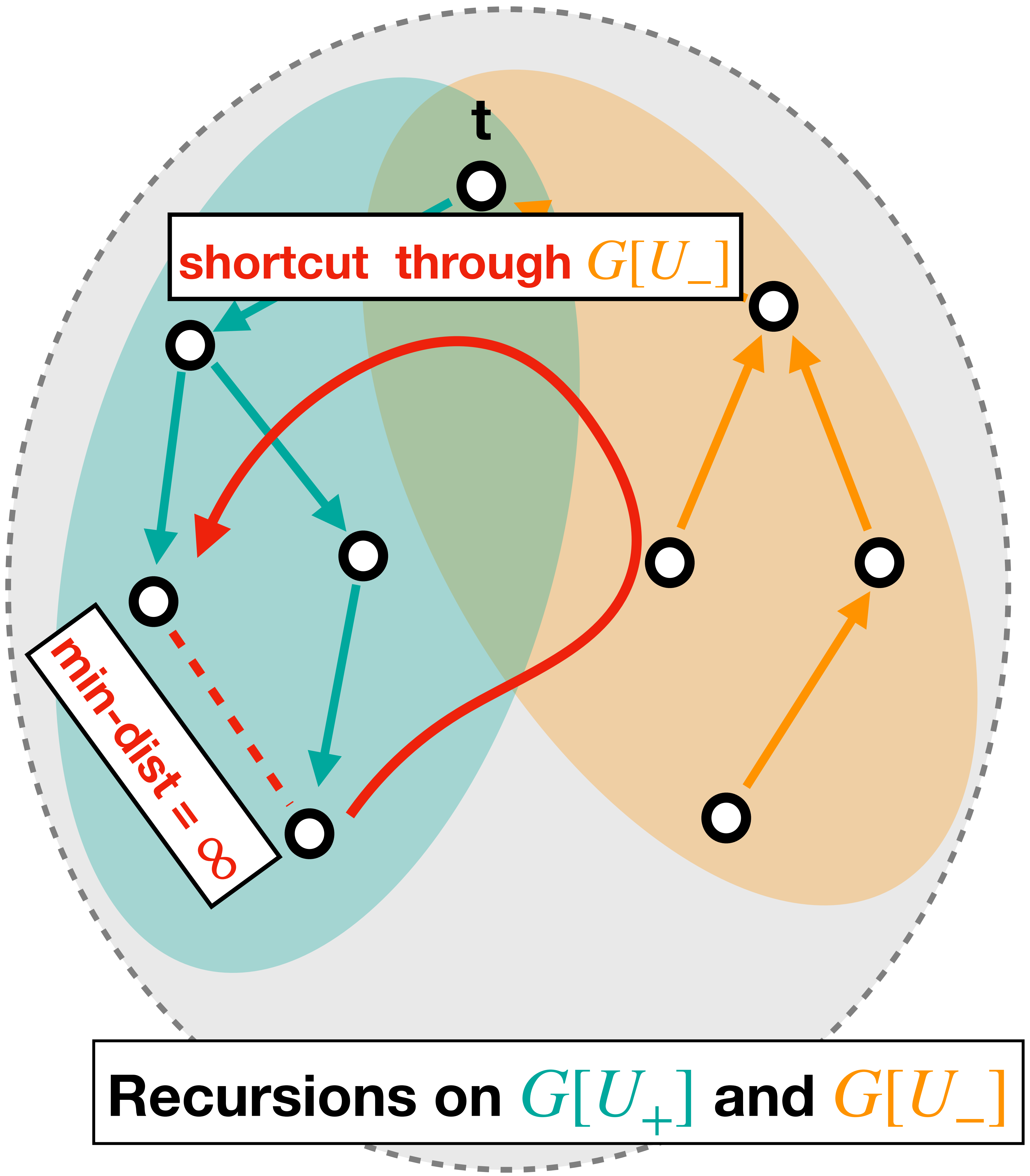


First attempt

<u>Some issues:</u>

- **Soundness**
  A large min-distance within $G[U_+]$ does not imply a large min-distance in $G[V]$

- **Runtime**
  $G[U_+]$ and $G[U\_]$ are usually intersecting, runtime can be high

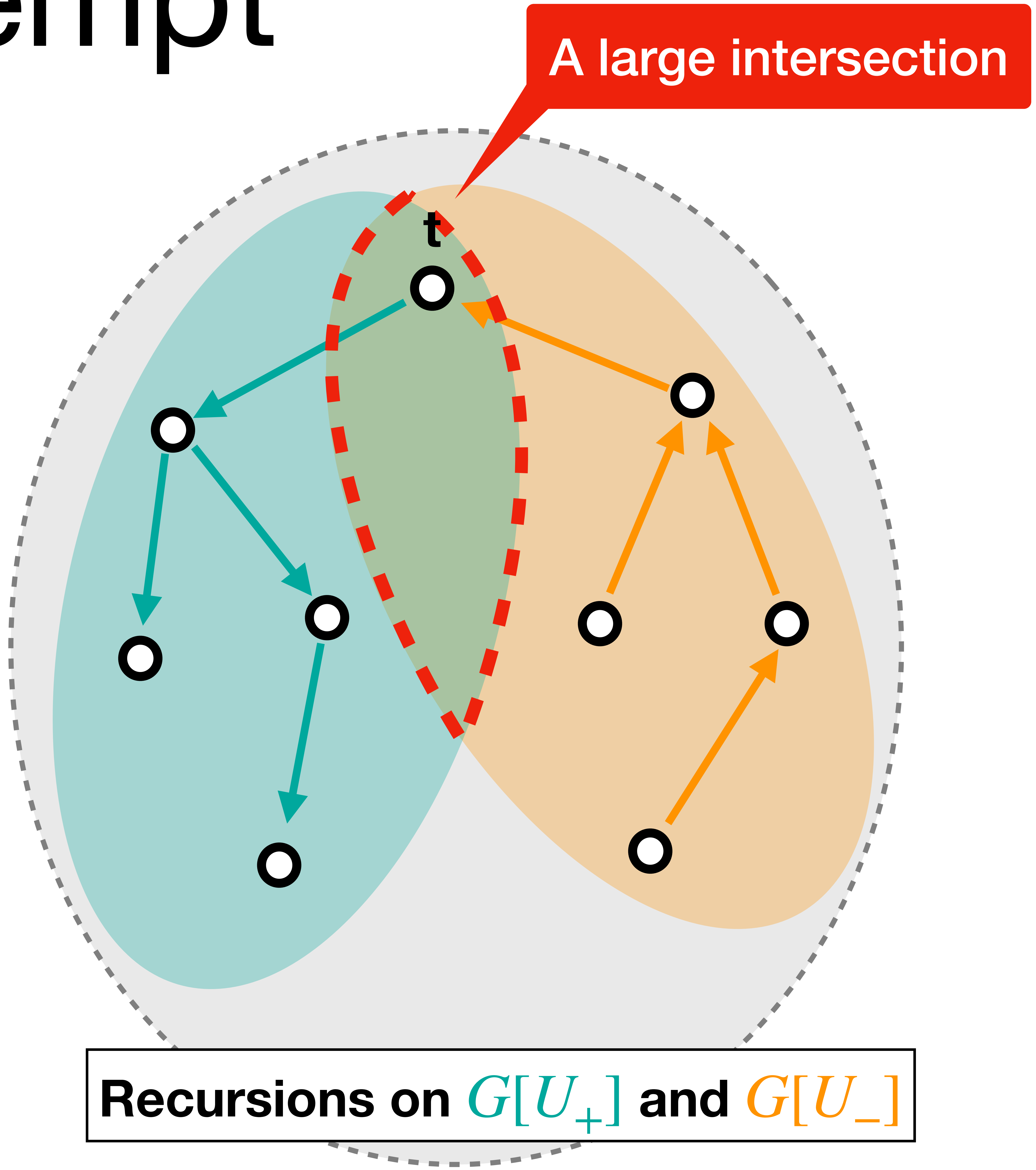


First attempt

<u>Some issues:</u>

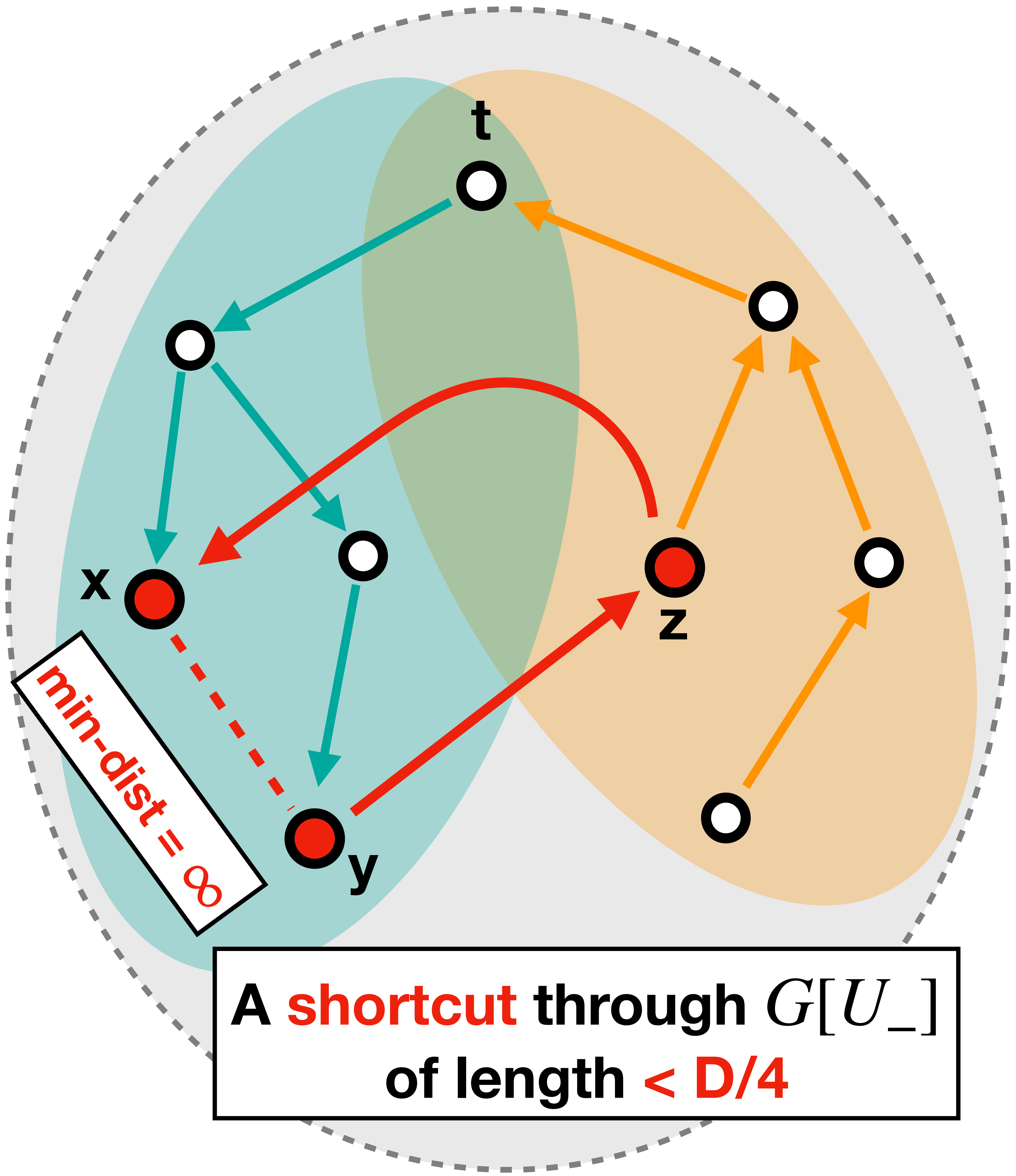
- **Soundness**
  A large min-distance within $G[U_+]$ does not imply a large min-distance in $G[V]$

- **Runtime**
  $G[U_+]$ and $G[U\_]$ are usually intersecting, runtime can be high



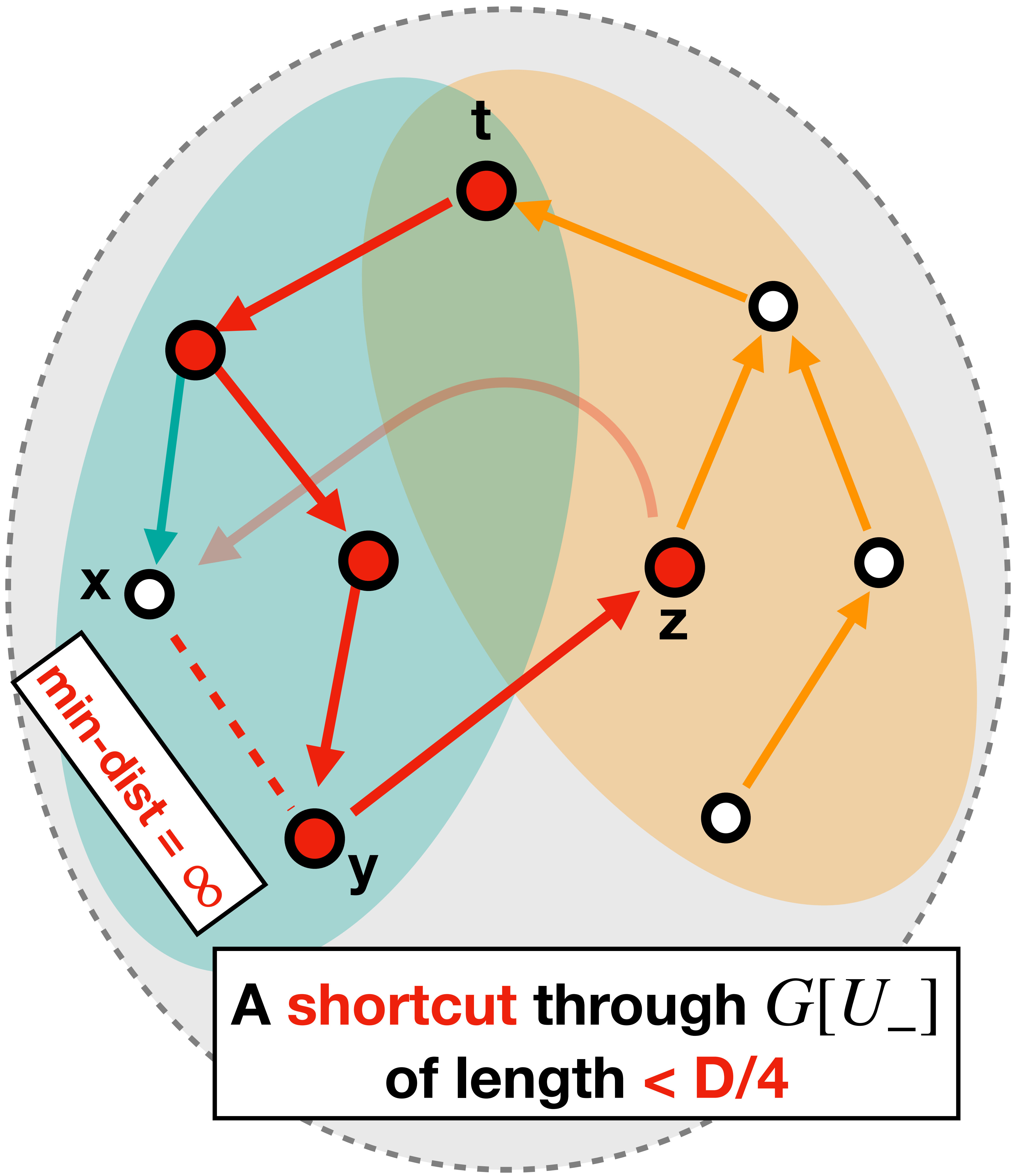
Soundness issue

- A large min-distance within $G[U_+]$ does not imply a large min-distance in $G[V]$



Soundness issue

- A large min-distance within $G[U_+]$ does not imply a large min-distance in $G[V]$
- **Observation**
  Distance from **t to z is < D/4 + D/4**

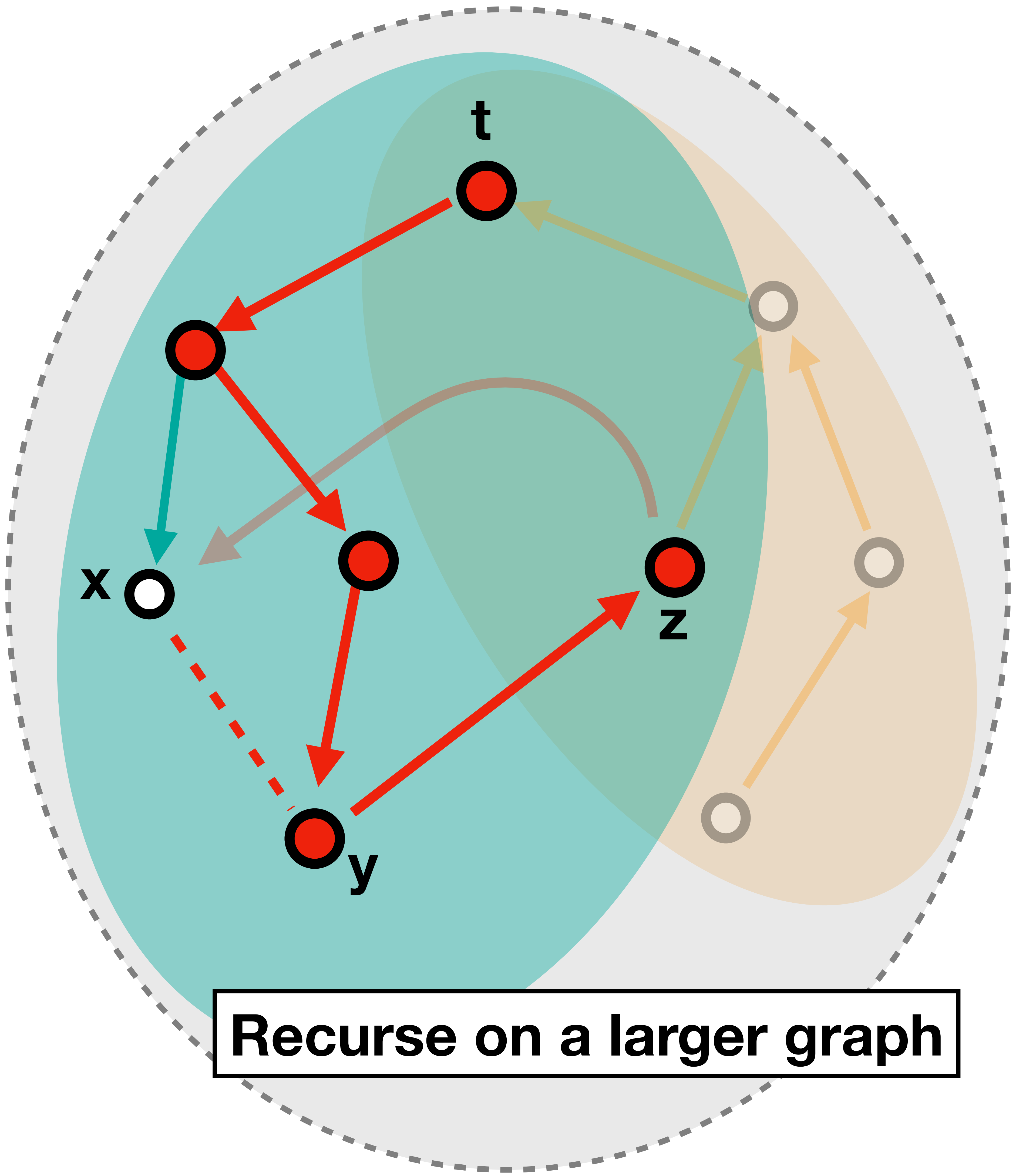


Soundness issue

- A large min-distance within $G[U_+]$ does not imply a large min-distance in $G[V]$

- **Observation**
  Distance from **t to z is < D/4 + D/4**

- **Idea**
  Add more vertices in the recursion
  $S_+ = \{u \mid \mathrm{dist}(t, u) < D/2\}$
  This **preserves pairwise distances** for
  $U_+ = \{u \mid \mathrm{dist}(t, u) < D/4\}$

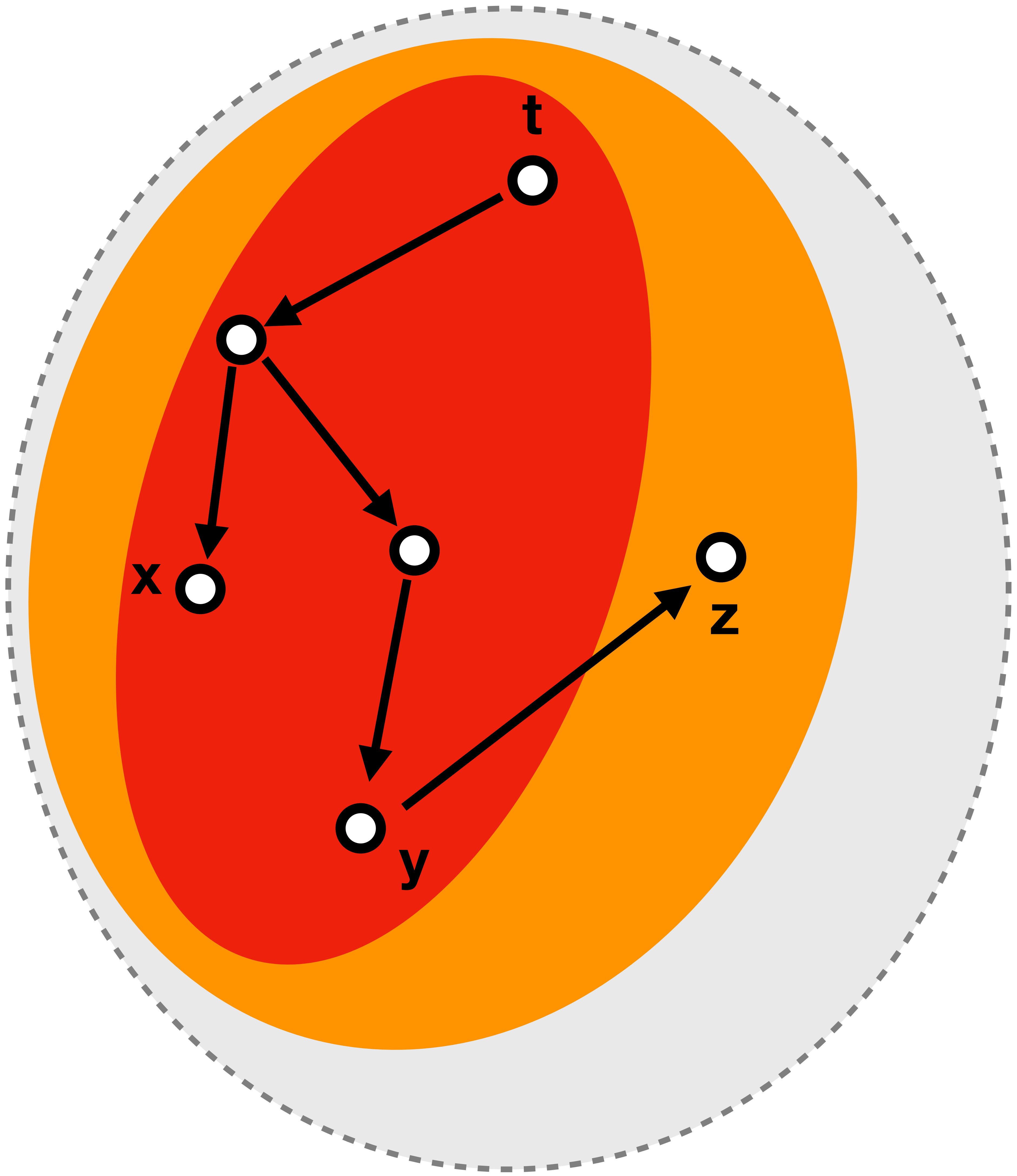


Soundness issue

- Two different vertex sets:
  $S_+ = \{u \mid \mathrm{dist}(t, u) < D/2\}$
  $U_+ = \{u \mid \mathrm{dist}(t, u) < D/4\}$

- $G[S_+]$ only preserves pairwise distances among U_+, not the entire S_+

- In general, the recursive algorithm needs to take **two parameters** (S, U) such that $U \subseteq S \subseteq V$

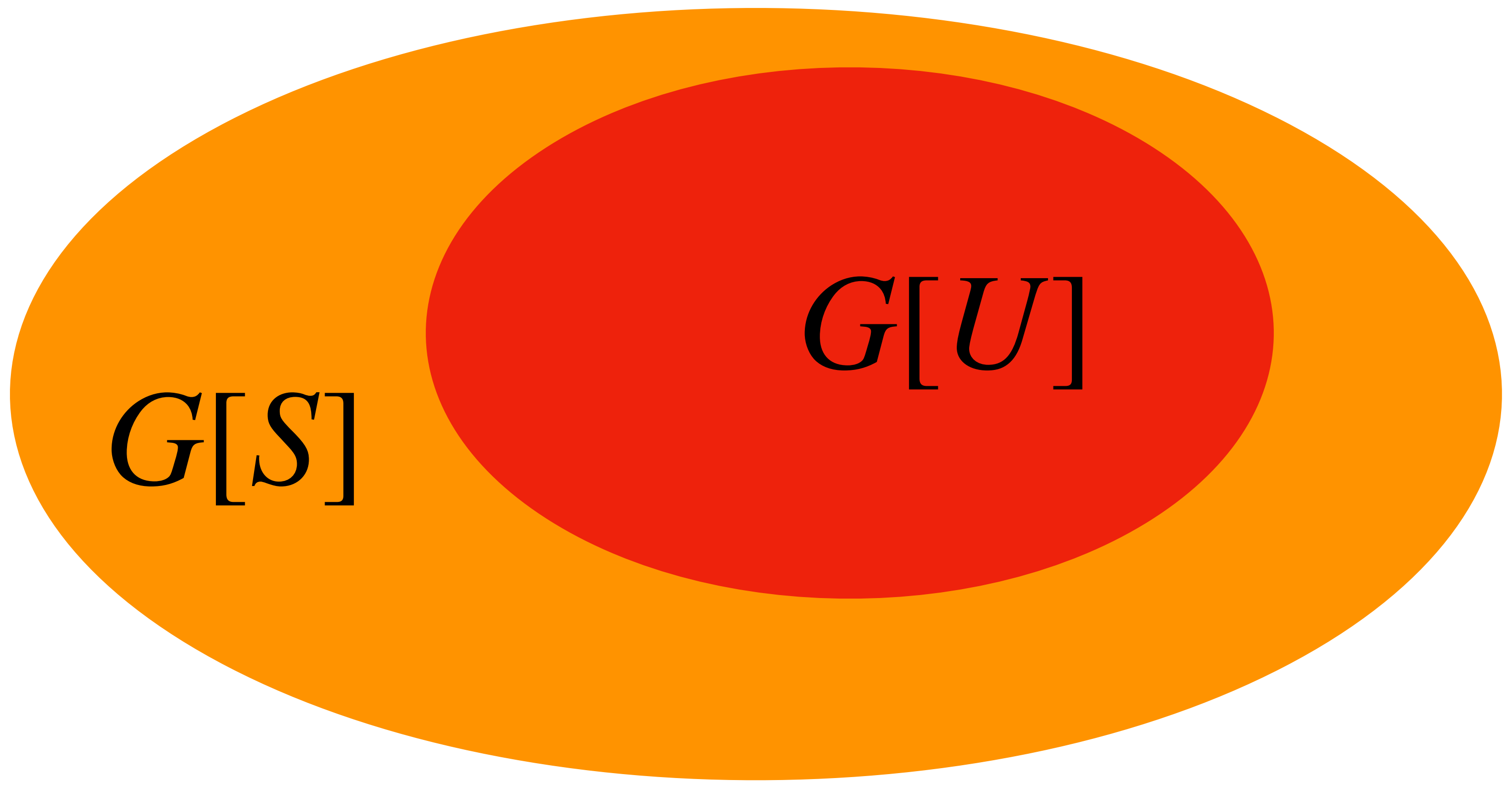


Soundness issue

- Two different vertex sets:
  $S_+ = \{u \mid \mathrm{dist}(t, u) < D/2\}$
  $U_+ = \{u \mid \mathrm{dist}(t, u) < D/4\}$

- $G[S_+]$ only preserves pairwise distances among U_+, not the entire S_+

- **In general, the recursive algorithm needs to take two parameters (S, U) such that $U \subseteq S \subseteq V$**



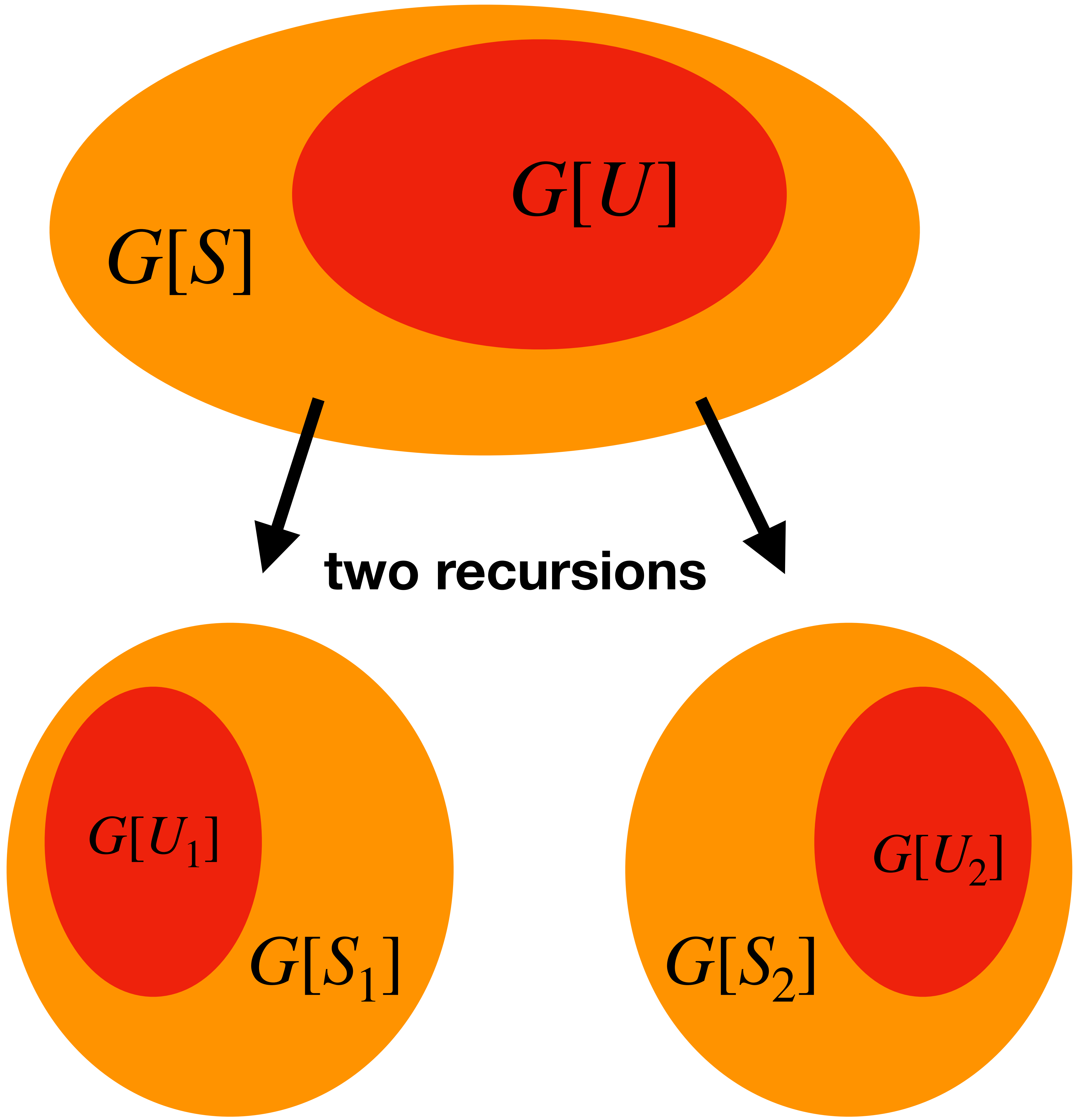
$G[S]$ preserves pairwise distances among U

Soundness issue

- Two different vertex sets:
  $S_+ = \{u \mid \text{dist}(t, u) < D/2\}$
  $U_+ = \{u \mid \text{dist}(t, u) < D/4\}$

- $G[S_+]$ only preserves pairwise distances among U_+, not the entire S_+

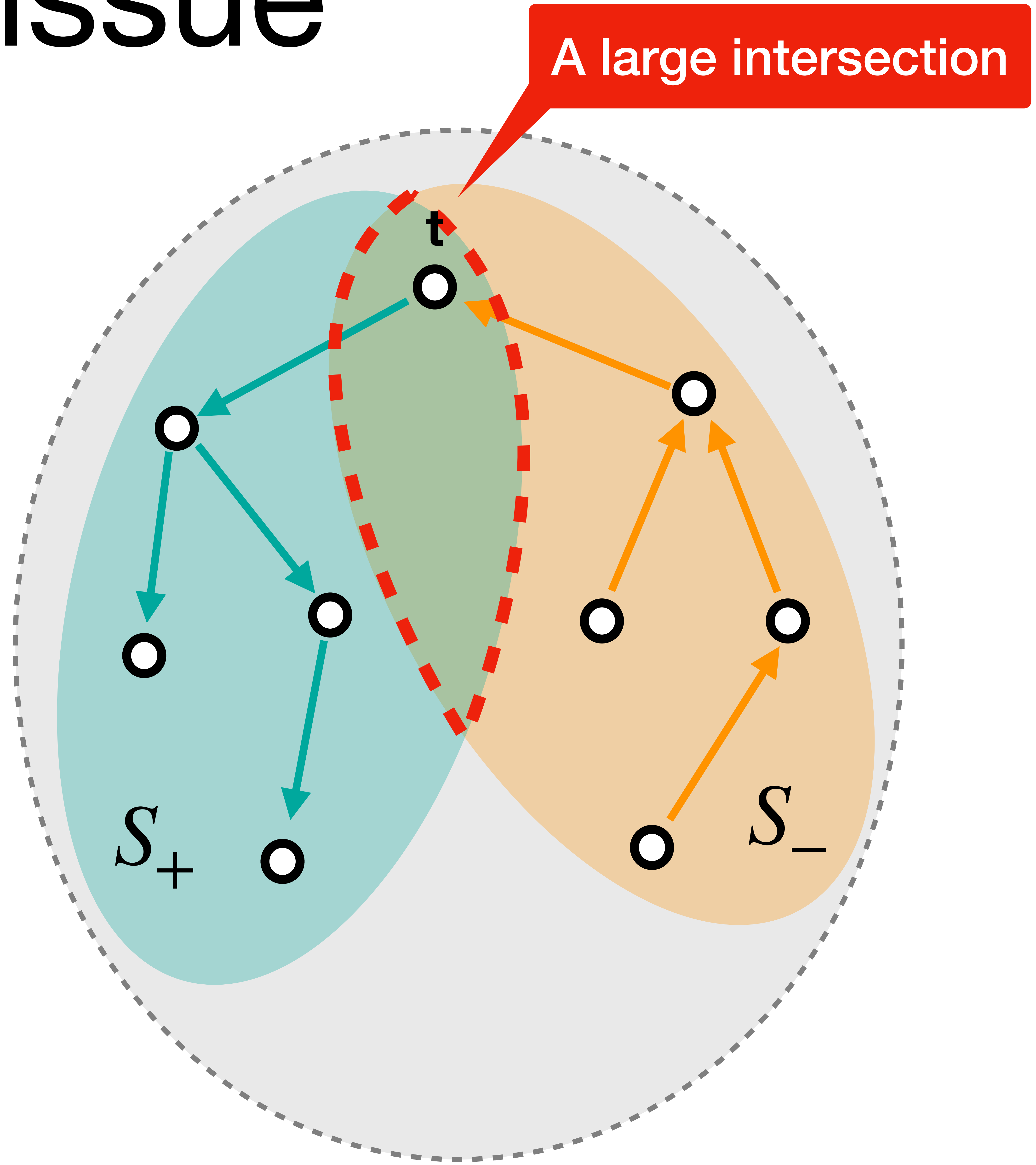
- **In general, the recursive algorithm needs to take two parameters (S, U) such that $U \subseteq S \subseteq V$**



$G[S_i]$ **preserves pairwise distances among** U_i

Runtime issue

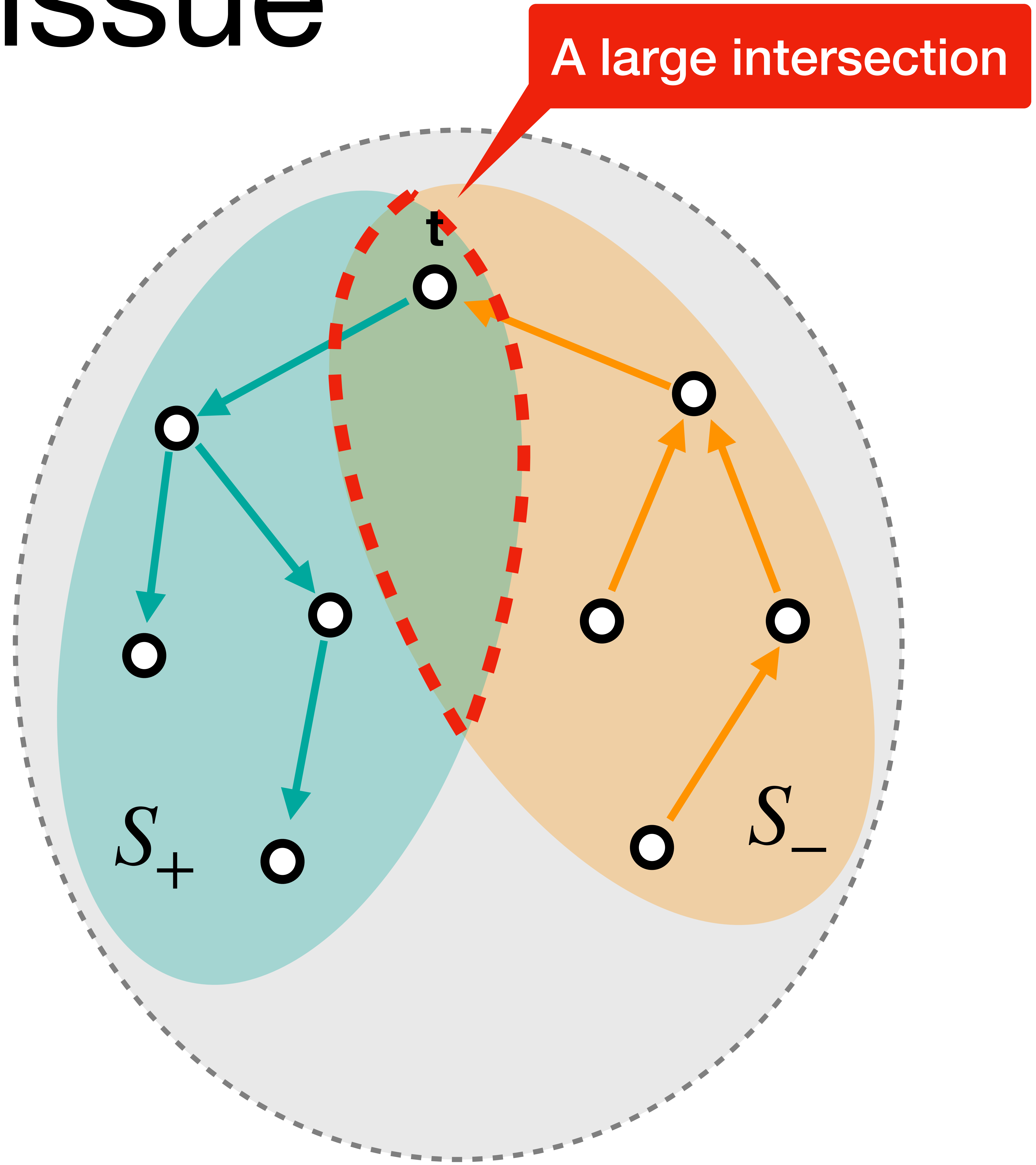
- S_+ and S_- are usually intersecting, so the runtime can be high
  $S_+ = \{u \mid \mathrm{dist}(t, u) < D/2\}$
  $S_- = \{u \mid \mathrm{dist}(u, t) < D/2\}$



Runtime issue

- S_+ and S_- are usually intersecting, so the runtime can be high
  $S_+ = \{u \mid \mathrm{dist}(t, u) < D/2\}$
  $S_- = \{u \mid \mathrm{dist}(u, t) < D/2\}$

- **Enforce disjointness** by recursing on instances $(S_+ \backslash S_-, U_+)$ & $(S_- \backslash S_+, U_-)$

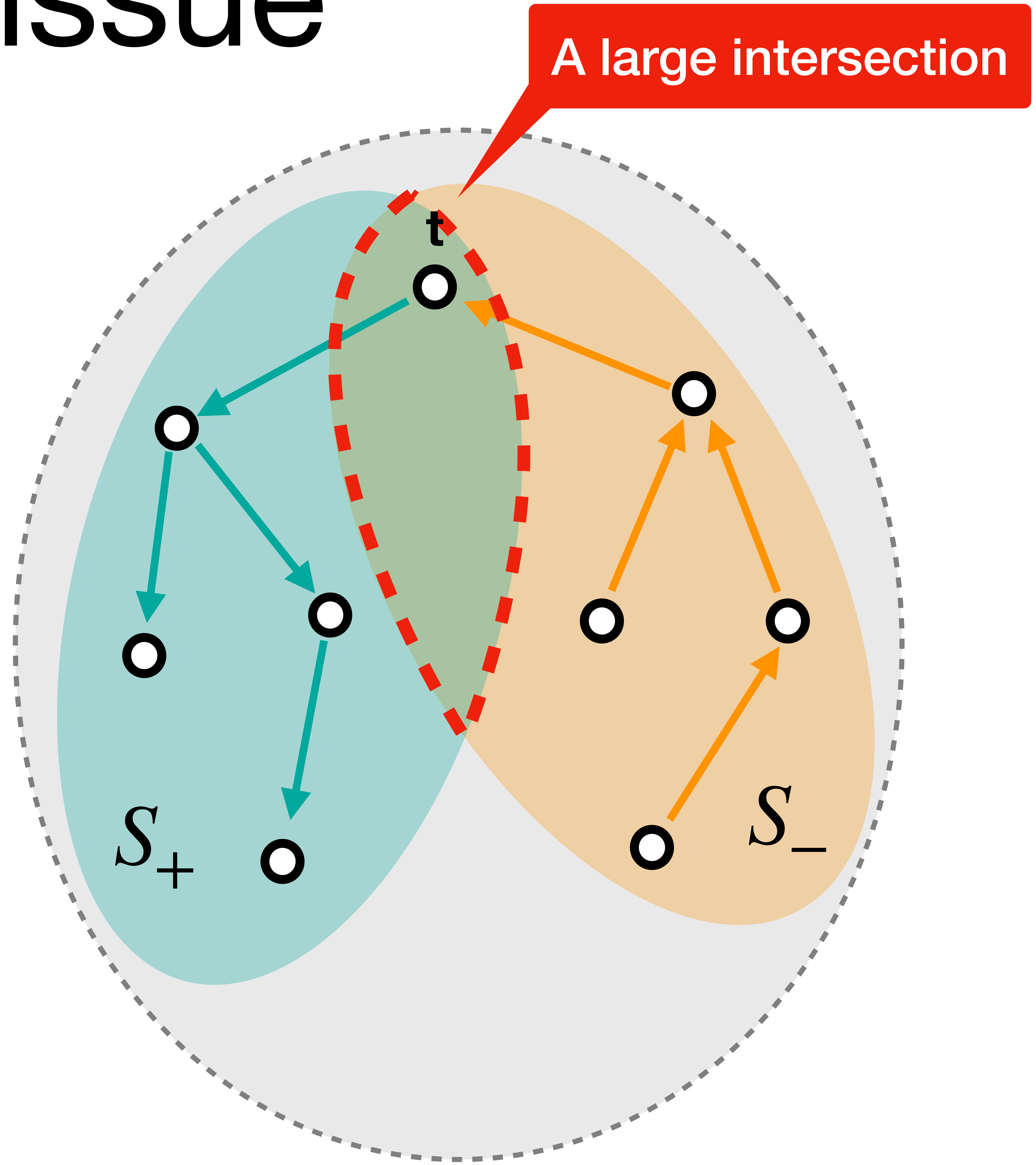


Runtime issue

- S_+ and S_- are usually intersecting, so the runtime can be high
  $S_+ = \{u \mid \mathrm{dist}(t, u) < D/2\}$
  $S_- = \{u \mid \mathrm{dist}(u, t) < D/2\}$

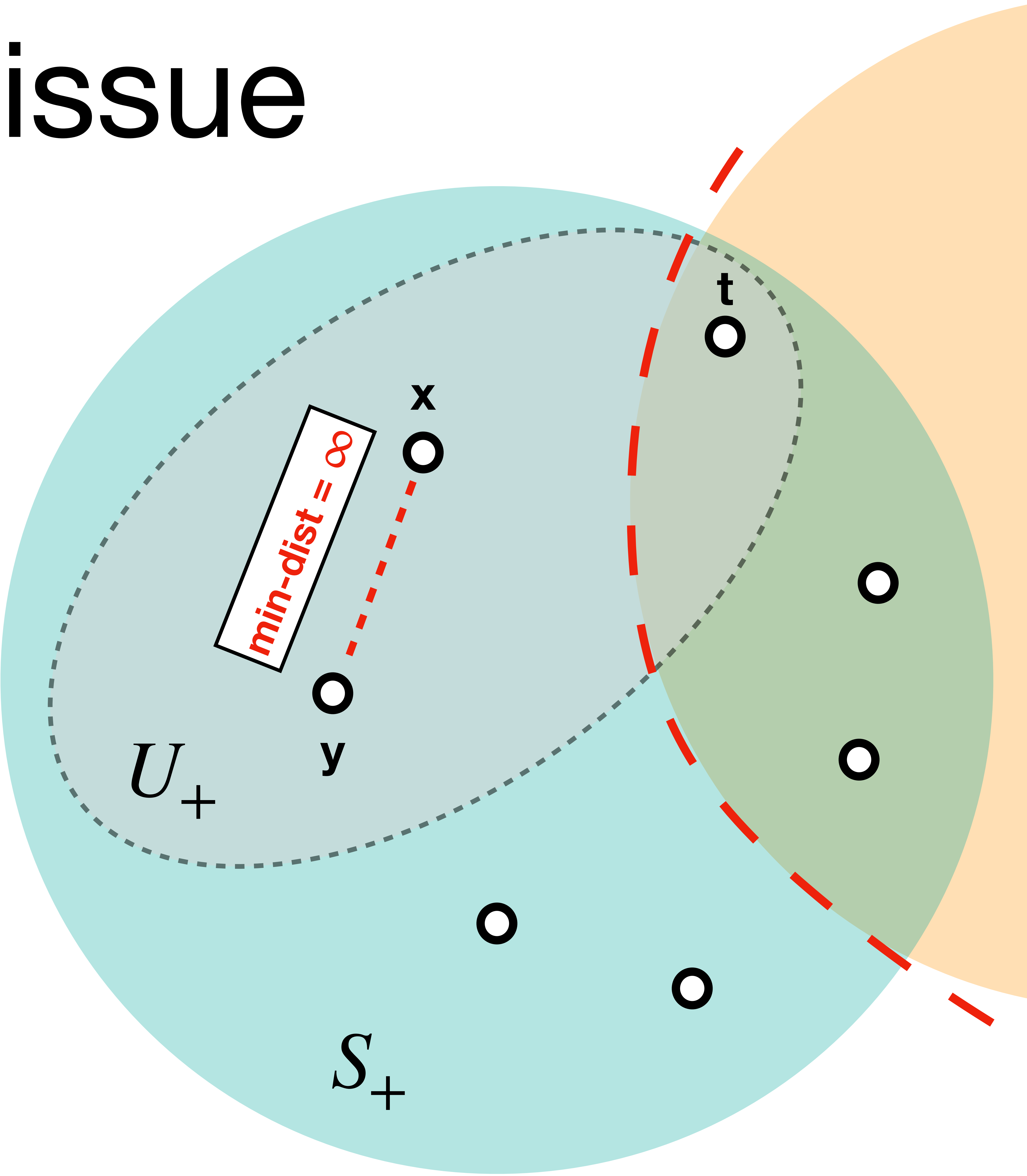
- **Enforce disjointness** by recursing on instances $(S_+ \backslash S_-, U_+)$ & $(S_- \backslash S_+, U_-)$

- How about soundness again?



Runtime issue

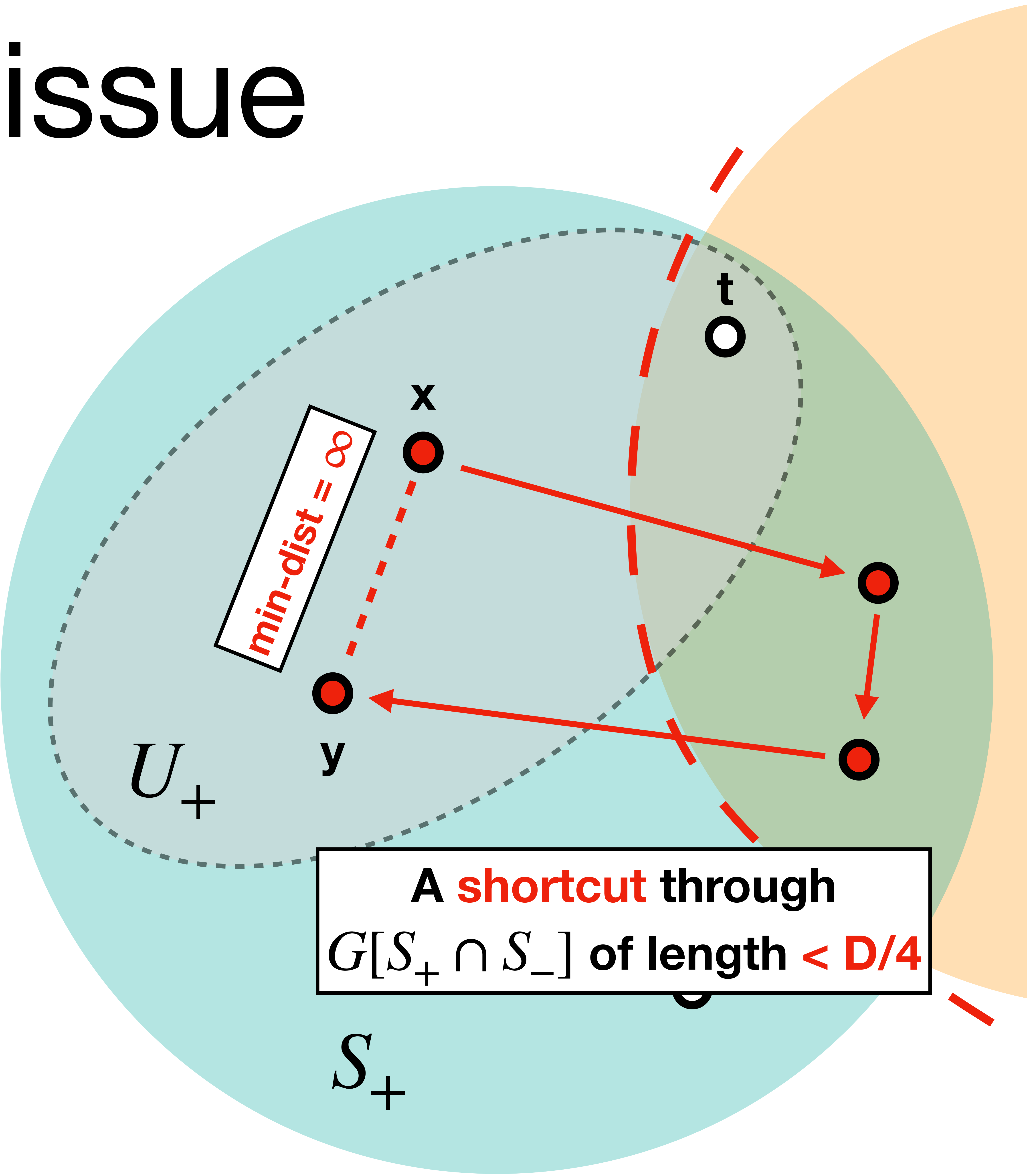
- **Enforce disjointness** by recursing on instances $(S_+ \backslash S_-, U_+)$ & $(S_- \backslash S_+, U_-)$
- How about soundness again?
  x, y such that $\text{dist}^{\min}_{G[S_+ \backslash S_-]}(x, y) = \infty$



Runtime issue

- **Enforce disjointness** by recursing on instances $(S_+ \setminus S_-, U_+)$ & $(S_- \setminus S_+, U_-)$

- How about soundness again?
  x, y such that $\mathrm{dist}^{\min}_{G[S_+ \setminus S_-]}(x, y) = \infty$
  but $\mathrm{dist}_{G[S_+]}(x, y) < D/4$

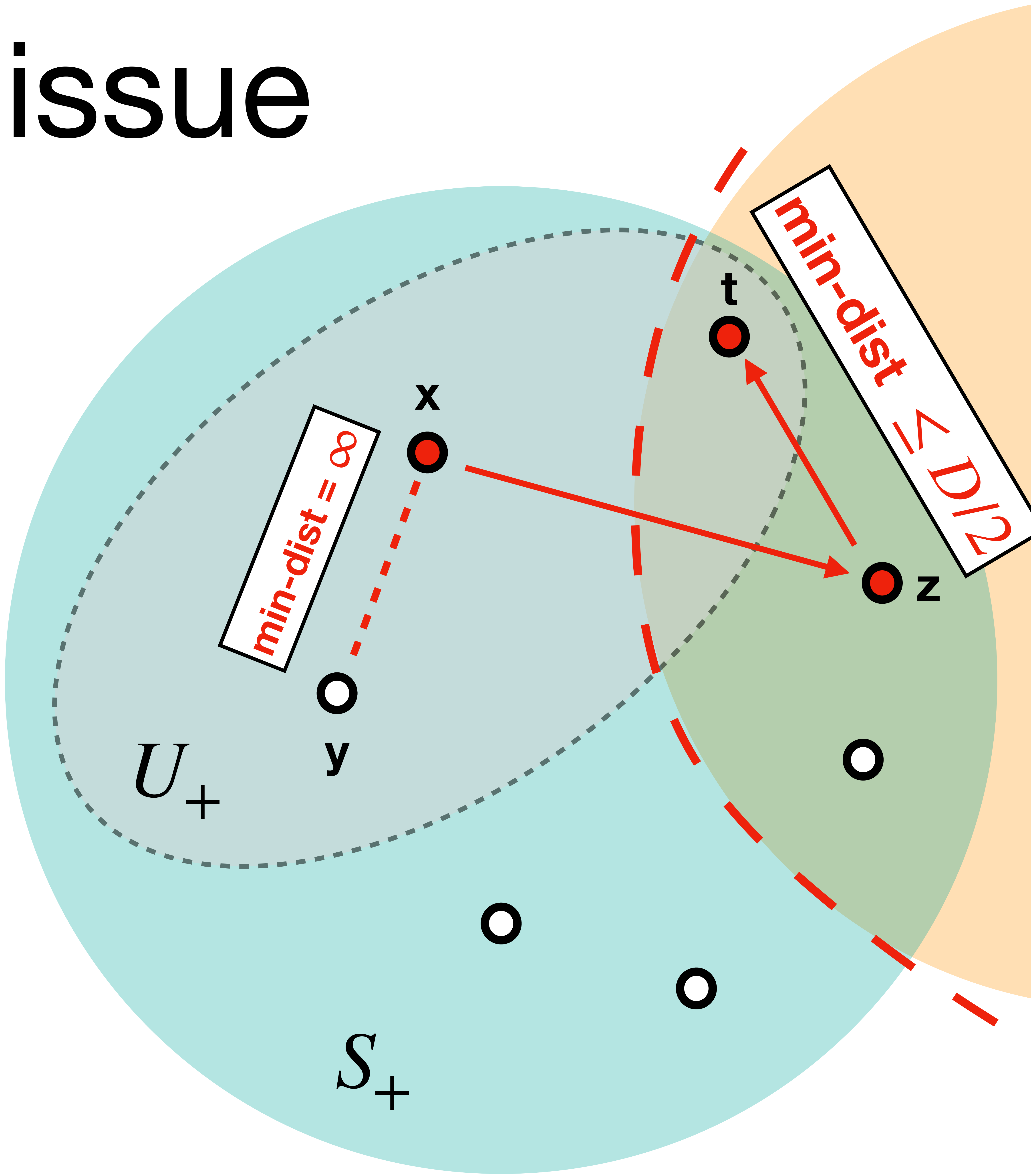


Runtime issue

- **Enforce disjointness** by recursing on instances $(S_+ \backslash S_-, U_+)$ & $(S_- \backslash S_+, U_-)$

- How about soundness again?
  x, y such that $\mathrm{dist}^{\min}_{G[S_+ \backslash S_-]}(x, y) = \infty$
  but $\mathrm{dist}_{G[S_+]}(x, y) < D/4$

- $\mathrm{dist}(x, t) \leq \mathrm{dist}(x, z) + \mathrm{dist}(z, t) < 3D/4$



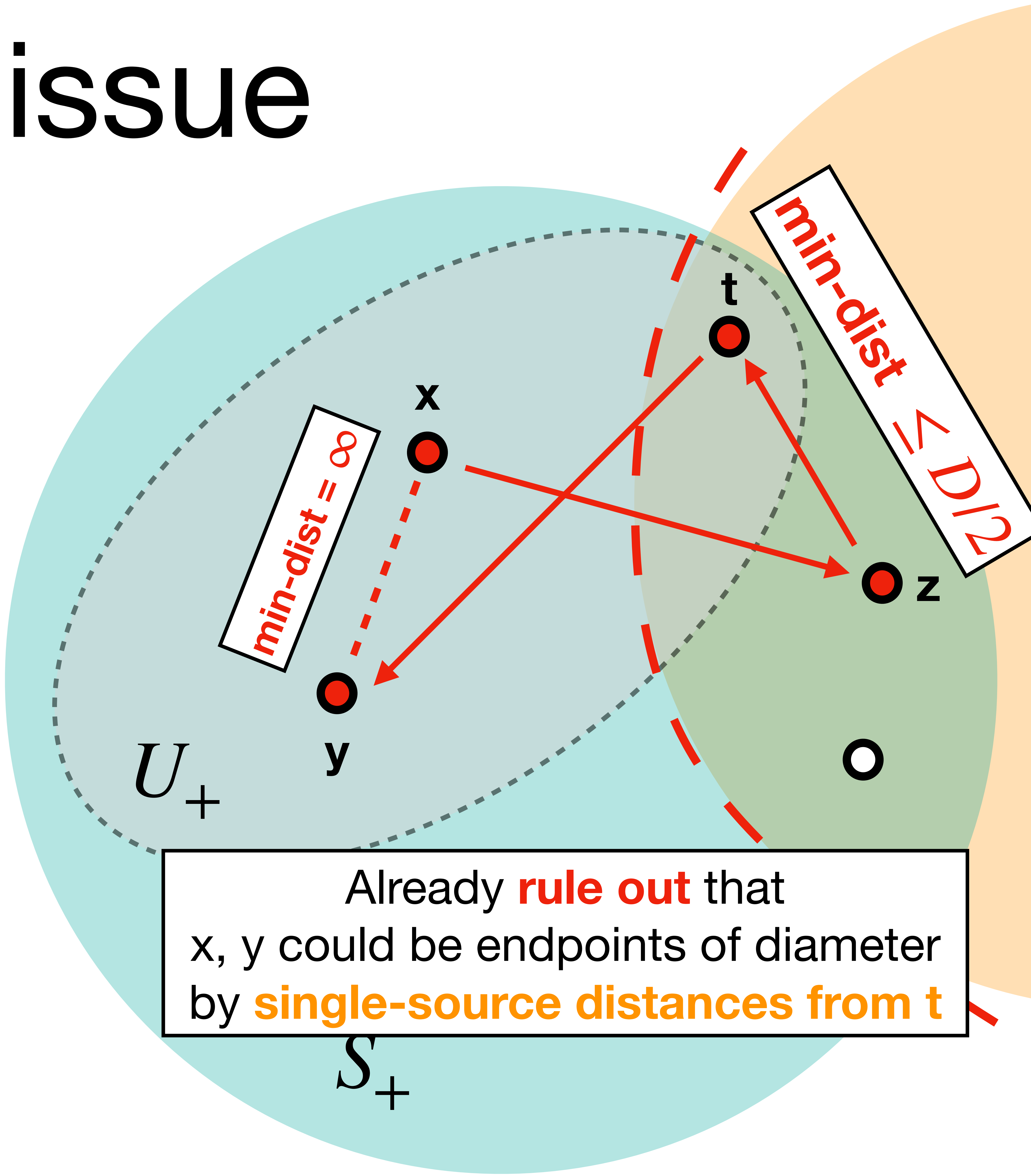
Runtime issue

- **Enforce disjointness** by recursing on instances $(S_+ \backslash S_-, U_+)$ & $(S_- \backslash S_+, U_-)$

- How about soundness again?
  x, y such that $\mathrm{dist}^{\min}_{G[S_+ \backslash S_-]}(x, y) = \infty$
  but $\mathrm{dist}_{G[S_+]}(x, y) < D/4$

- $\mathrm{dist}(x, t) \leq \mathrm{dist}(x, z) + \mathrm{dist}(z, t) < 3D/4$
  $\mathrm{dist}(x, t) + \mathrm{dist}(t, y) < D$

- Recurse on $(S_+ \backslash S_-, C_+)$
  $C_+ = U_+ \cap \{x \mid \mathrm{dist}(x, t) \geq 3D/4\}$



The recursive algorithm

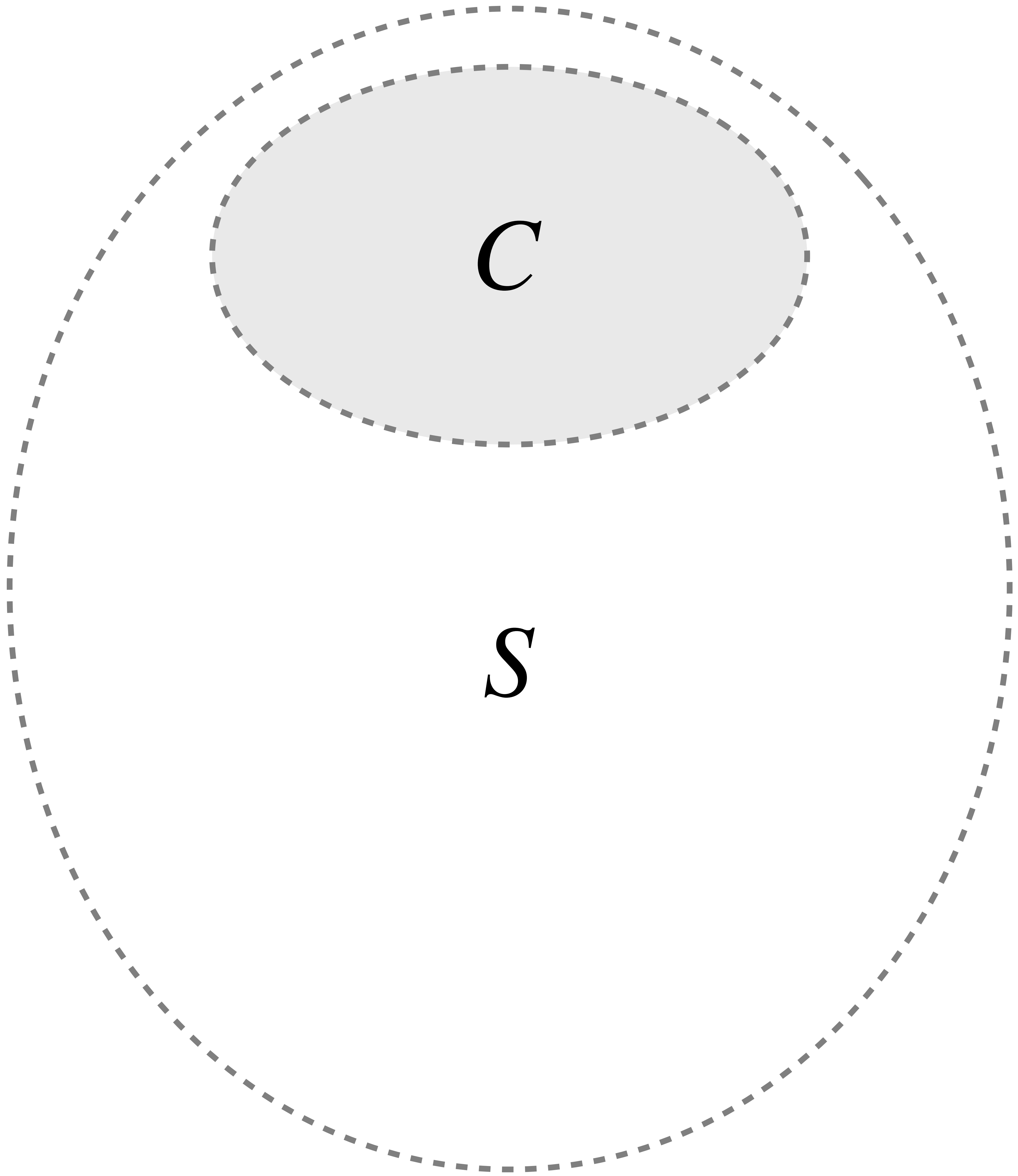
On input pair (S, C)

1. Pick a random vertex $t \in C$

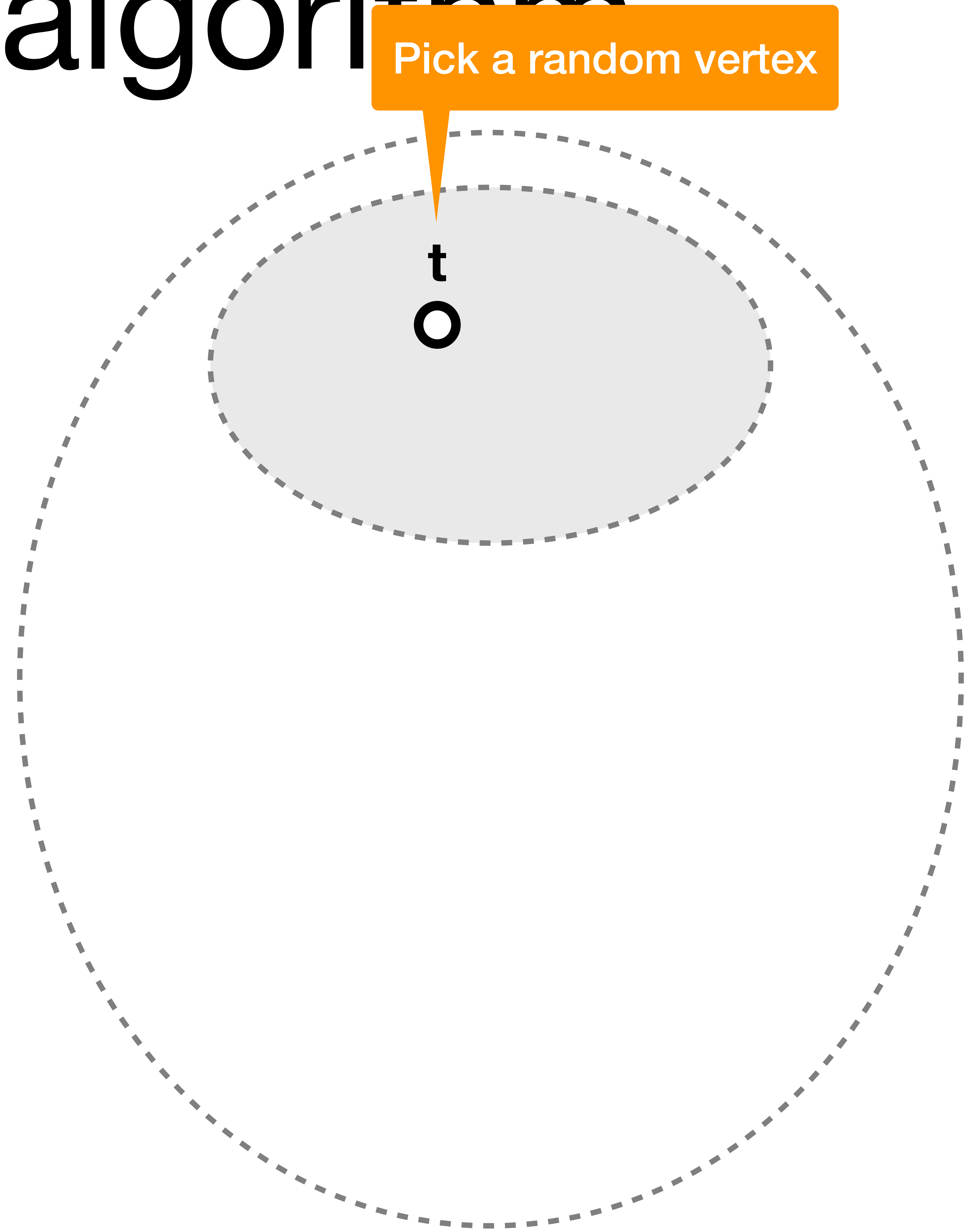
2. Compute SSSP in to and from t

3. Define sets:
   $S_+ = \{x \mid \mathrm{dist}(t, x) < D/2\}$
   $C_+ = C \cap \{x \mid \mathrm{dist}(t, x) < D/4,\ \mathrm{dist}(x, t) \geq 3D/4\}$

4. Recurse on $(S_+ \backslash S_-, C_+ \backslash S_-)$
   and symmetrically $(S_- \backslash S_+, C_- \backslash S_+)$



The recursive algorithm



On input pair (S, C)

1. **Pick a random vertex** $t \in C$

2. Compute SSSP in to and from t

3. Define sets:
   $S_+ = \{x \mid \mathrm{dist}(t, x) < D/2\}$
   $C_+ = C \cap \{x \mid \mathrm{dist}(t, x) < D/4,\ \mathrm{dist}(x, t) \geq 3D/4\}$

4. Recurse on $(S_+ \setminus S_-, C_+ \setminus S_-)$
   and symmetrically $(S_- \setminus S_+, C_- \setminus S_+)$

The recursive algorithm

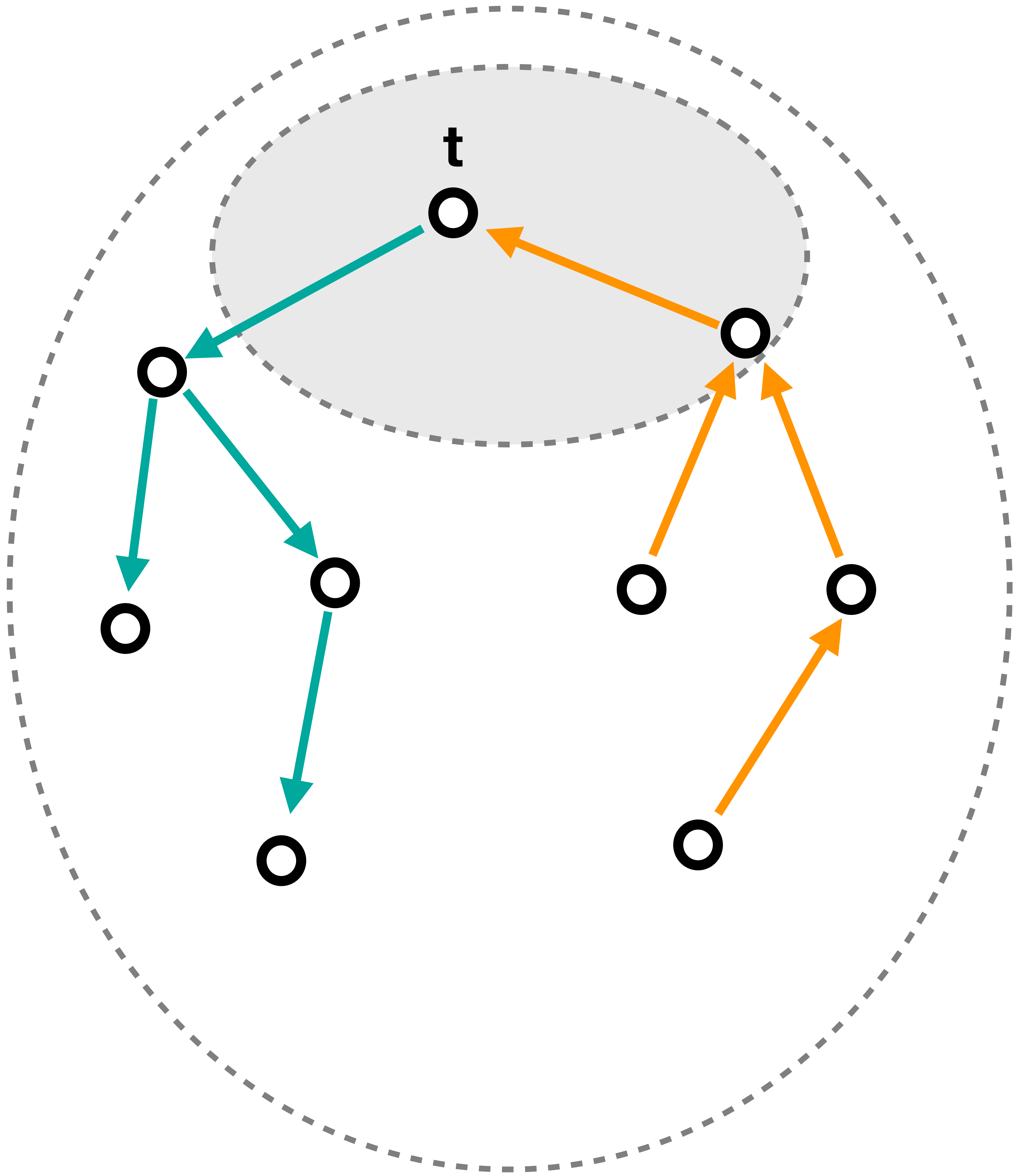
On input pair (S, C)

1. Pick a random vertex $t \in C$

2. **Compute SSSP in to and from t**

3. Define sets:
   $S_+ = \{x \mid \text{dist}(t, x) < D/2\}$
   $C_+ = C \cap \{x \mid \text{dist}(t, x) < D/4,\ \text{dist}(x, t) \geq 3D/4\}$

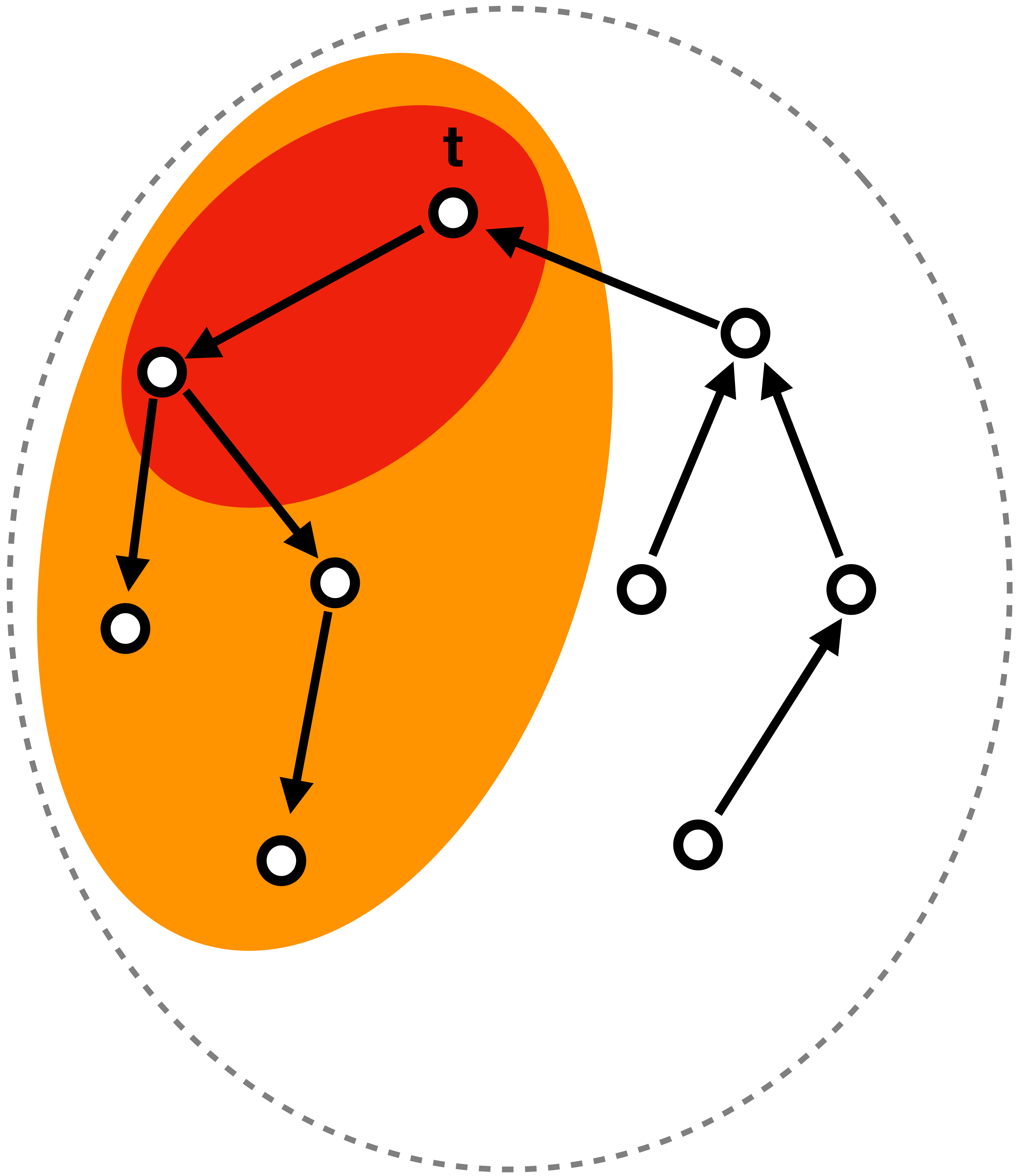
4. Recurse on $(S_+ \backslash S_-, C_+ \backslash S_-)$
   and symmetrically $(S_- \backslash S_+, C_- \backslash S_+)$



The recursive algorithm

On input pair (S, C)

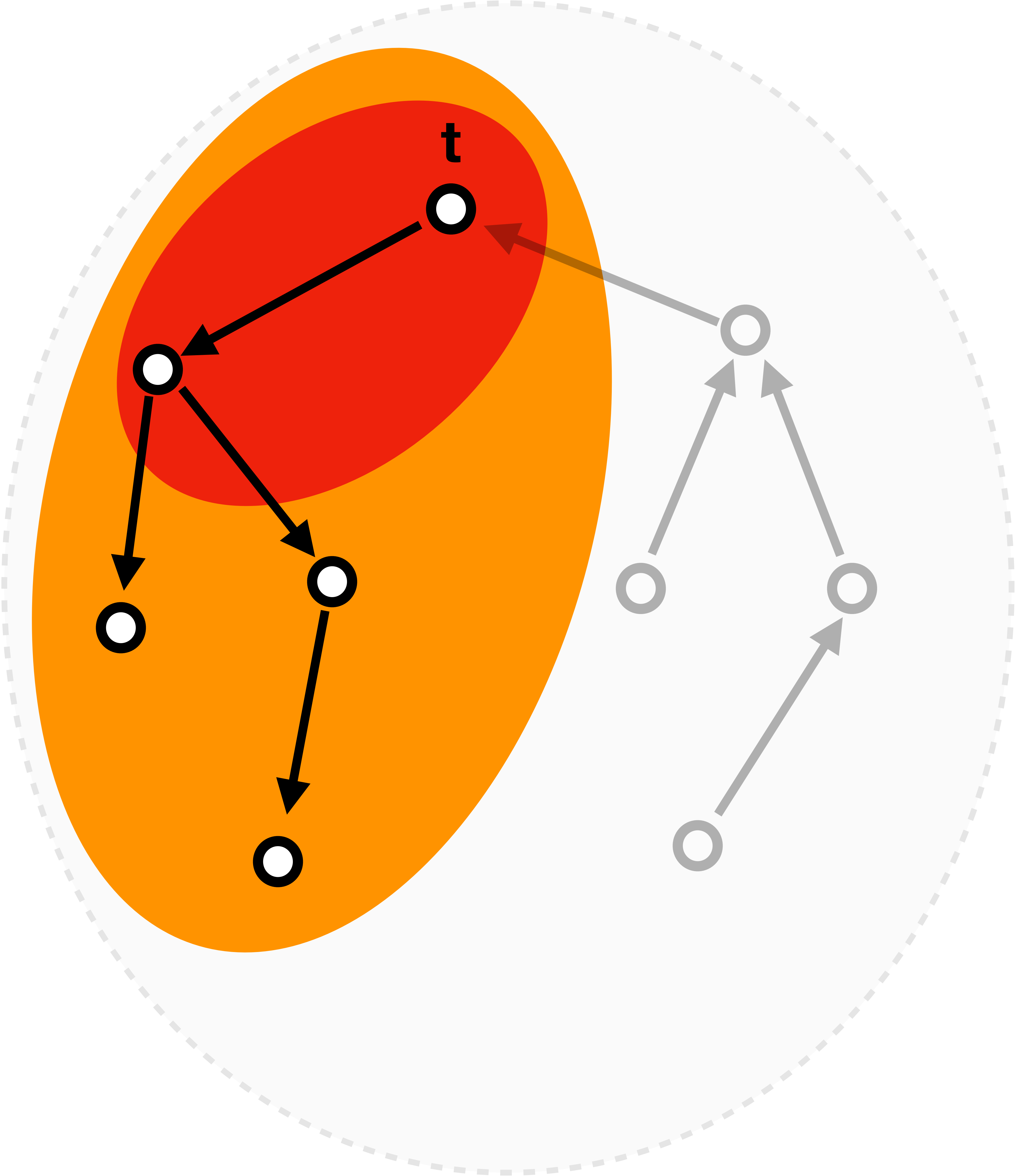
1. Pick a random vertex $t \in C$
2. Compute SSSP in to and from t
3. **Define sets:**
   $S_+ = \{x \mid \mathrm{dist}(t, x) < D/2\}$
   $C_+ = C \cap \{x \mid \mathrm{dist}(t, x) < D/4,\ \mathrm{dist}(x, t) \geq 3D/4\}$
4. Recurse on $(S_+ \backslash S_-, C_+ \backslash S_-)$
   and symmetrically $(S_- \backslash S_+, C_- \backslash S_+)$



The recursive algorithm

On input pair (S, C)

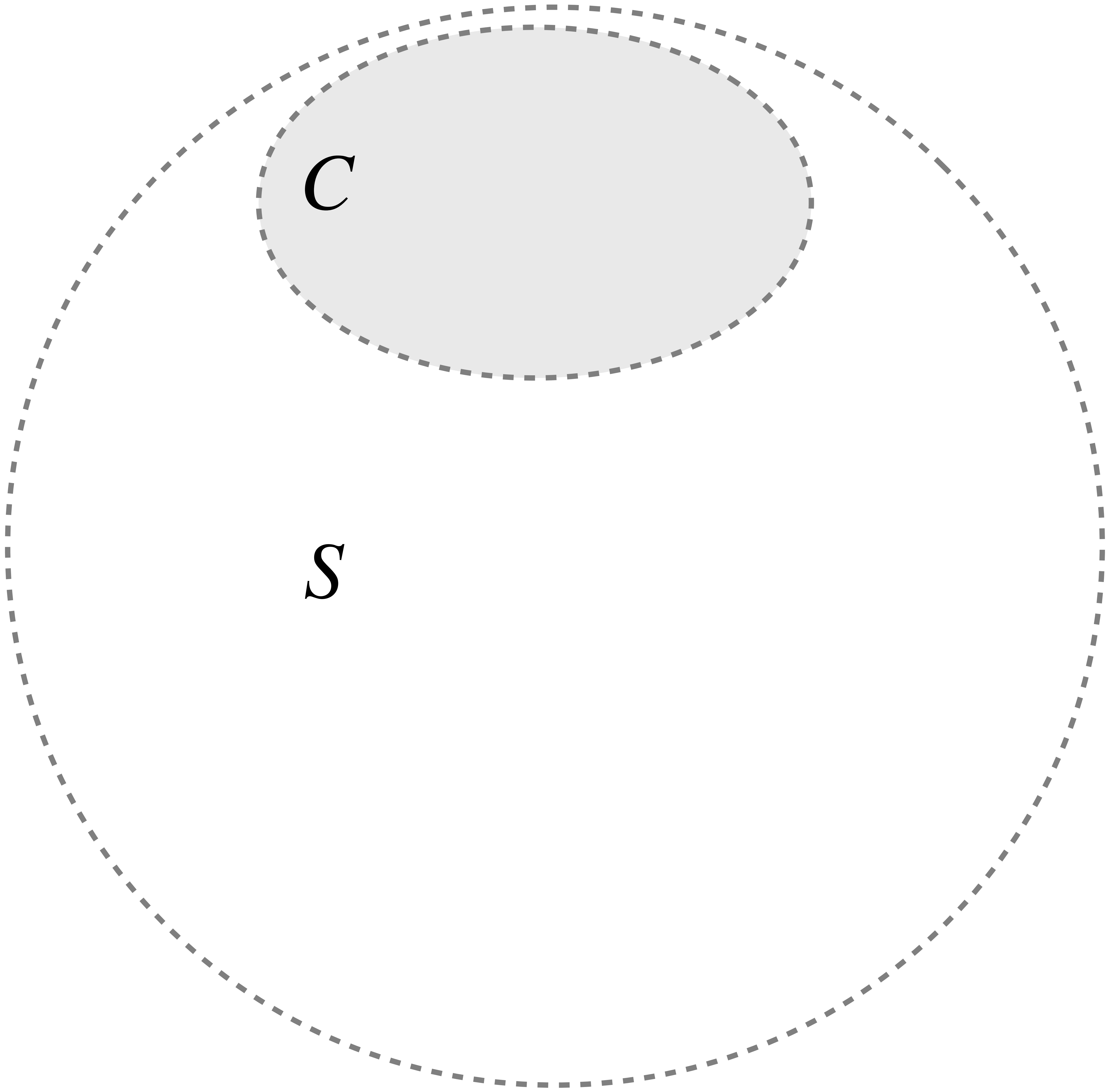
1. Pick a random vertex $t \in C$
2. Compute SSSP in to and from t
3. Define sets:
   $S_+ = \{x \mid \mathrm{dist}(t, x) < D/2\}$
   $C_+ = C \cap \{x \mid \mathrm{dist}(t, x) < D/4,\ \mathrm{dist}(x, t) \geq 3D/4\}$
4. **Recurse on** $(S_+ \backslash S_-, C_+ \backslash S_-)$
   and symmetrically $(S_- \backslash S_+, C_- \backslash S_+)$



$O(\log n)$-approx min-radius
in general digraphs

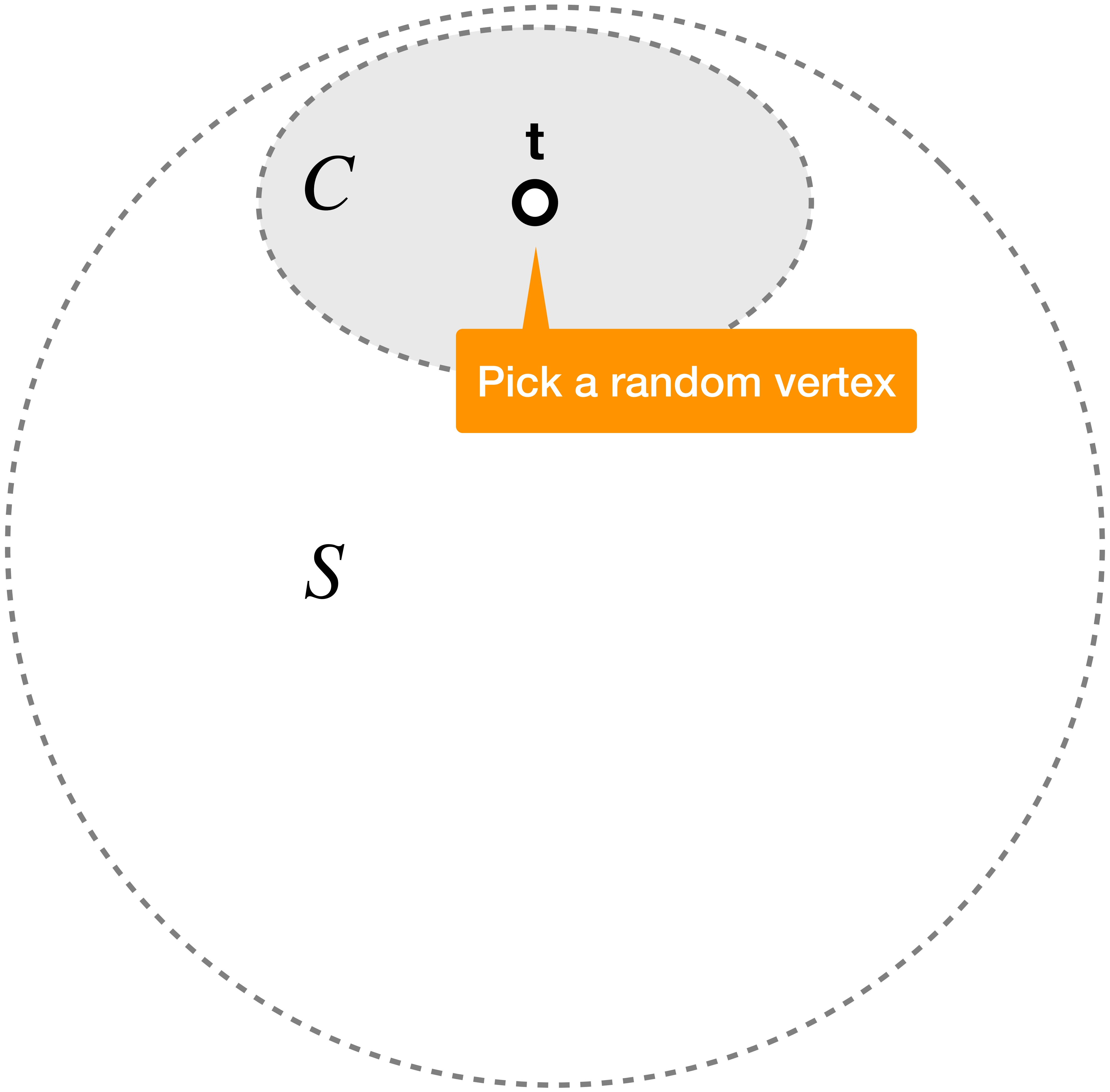
Contraction & Recurse

Goal: decide if the min-radius is
$< O(R \log n)$ or >R



Contraction & Recurse

Goal: decide if the min-radius is
$< O(R \log n)$ or $>R$



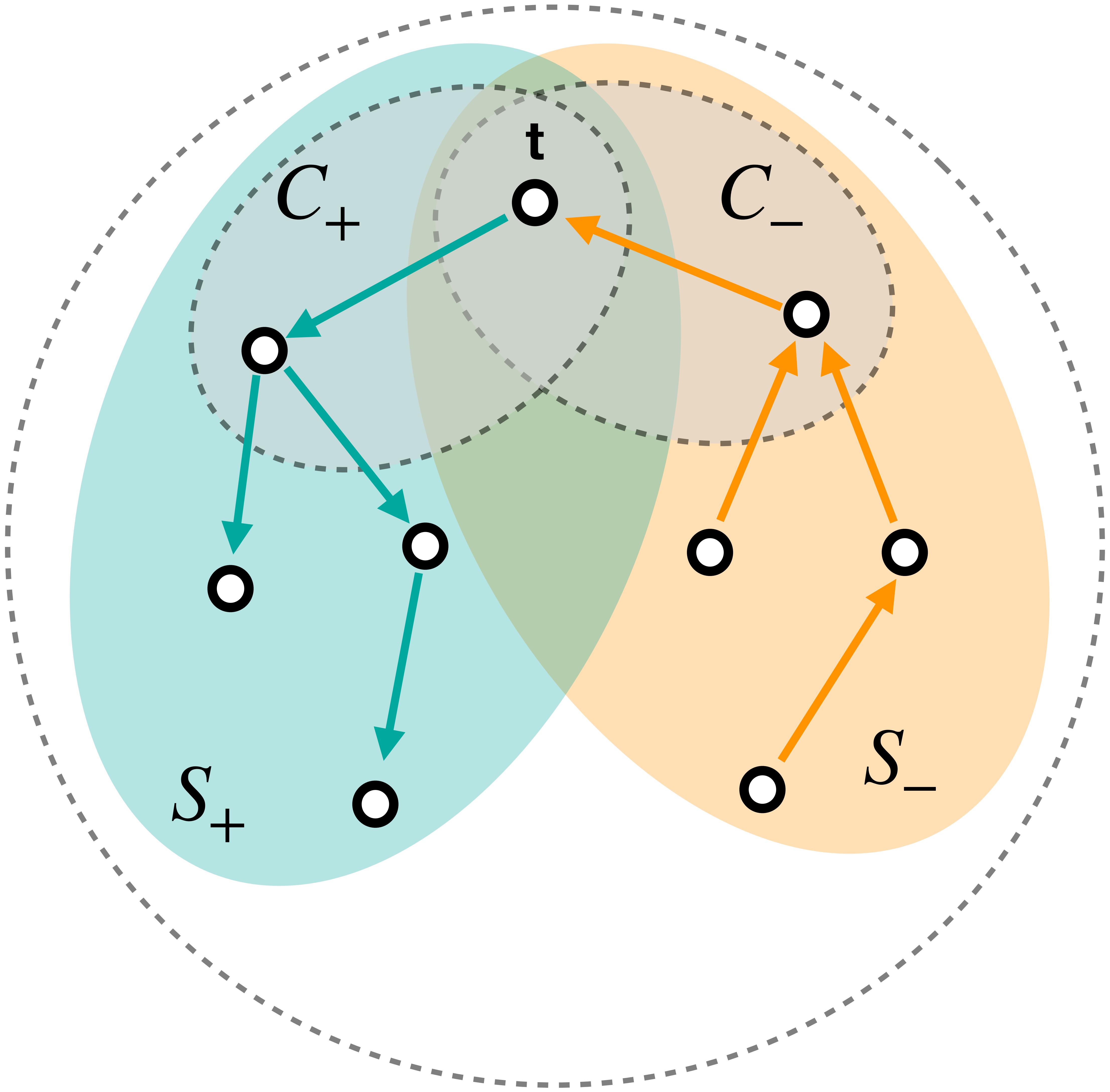
Contraction & Recurse

<u>Goal:</u> decide if the min-radius is $< O(R \log n)$ or >R

1. Define four sets $C_{+,-}, S_{+,-}$

   $C_+ = \{u \mid \mathrm{dist}(t, u) < R\}$

   $S_+ = \{u \mid \mathrm{dist}(t, u) < 2R\}$

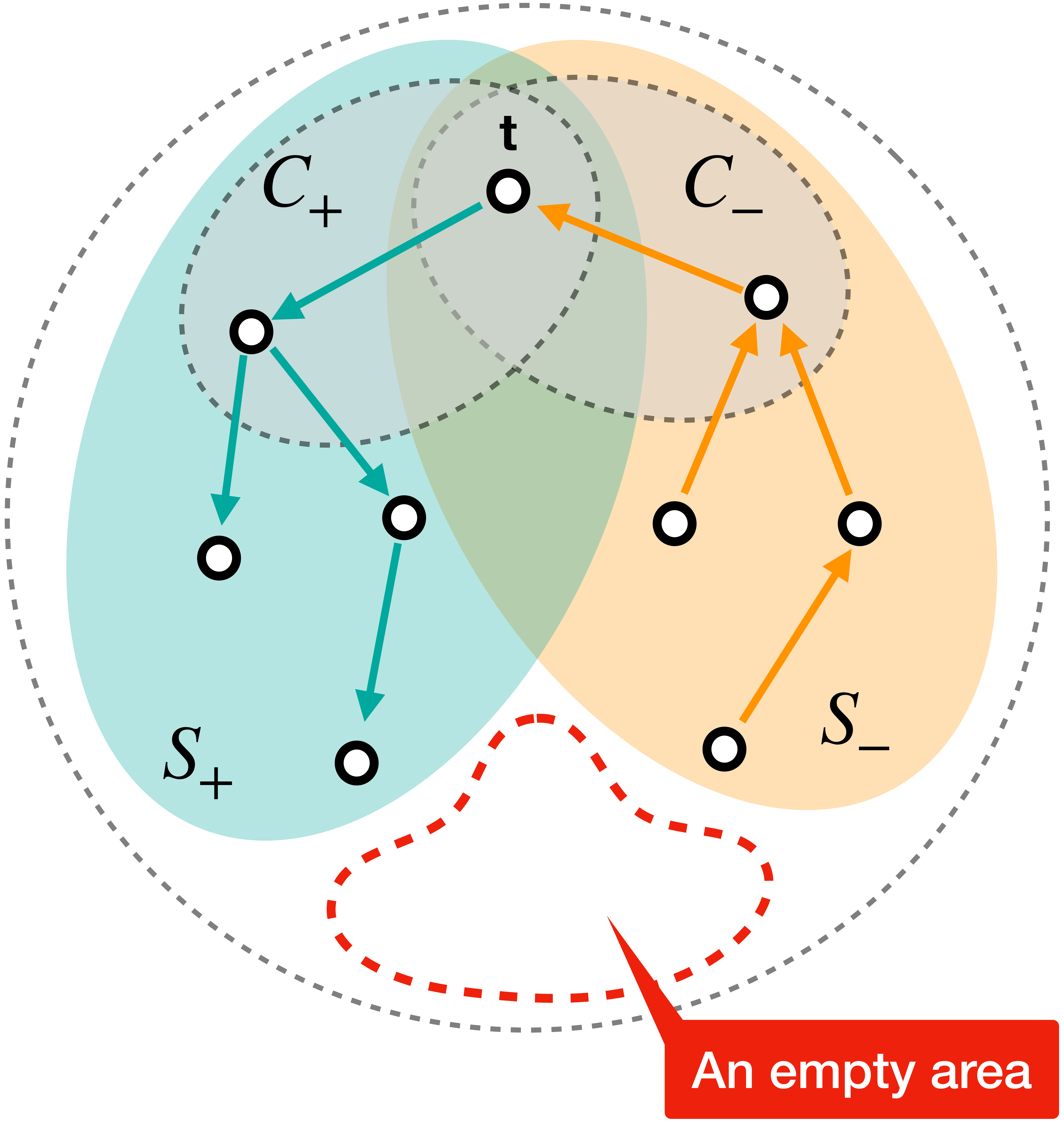


Contraction & Recurse

Goal: decide if the min-radius is $< O(R \log n)$ or >R

1. Define four sets $C_{+,-}, S_{+,-}$
   $C_+ = \{u \mid \text{dist}(t,u) < R\}$
   $S_+ = \{u \mid \text{dist}(t,u) < 2R\}$

2. If $S_+ \cup S_- = V$
   t is a center with small radius



Contraction & Recurse

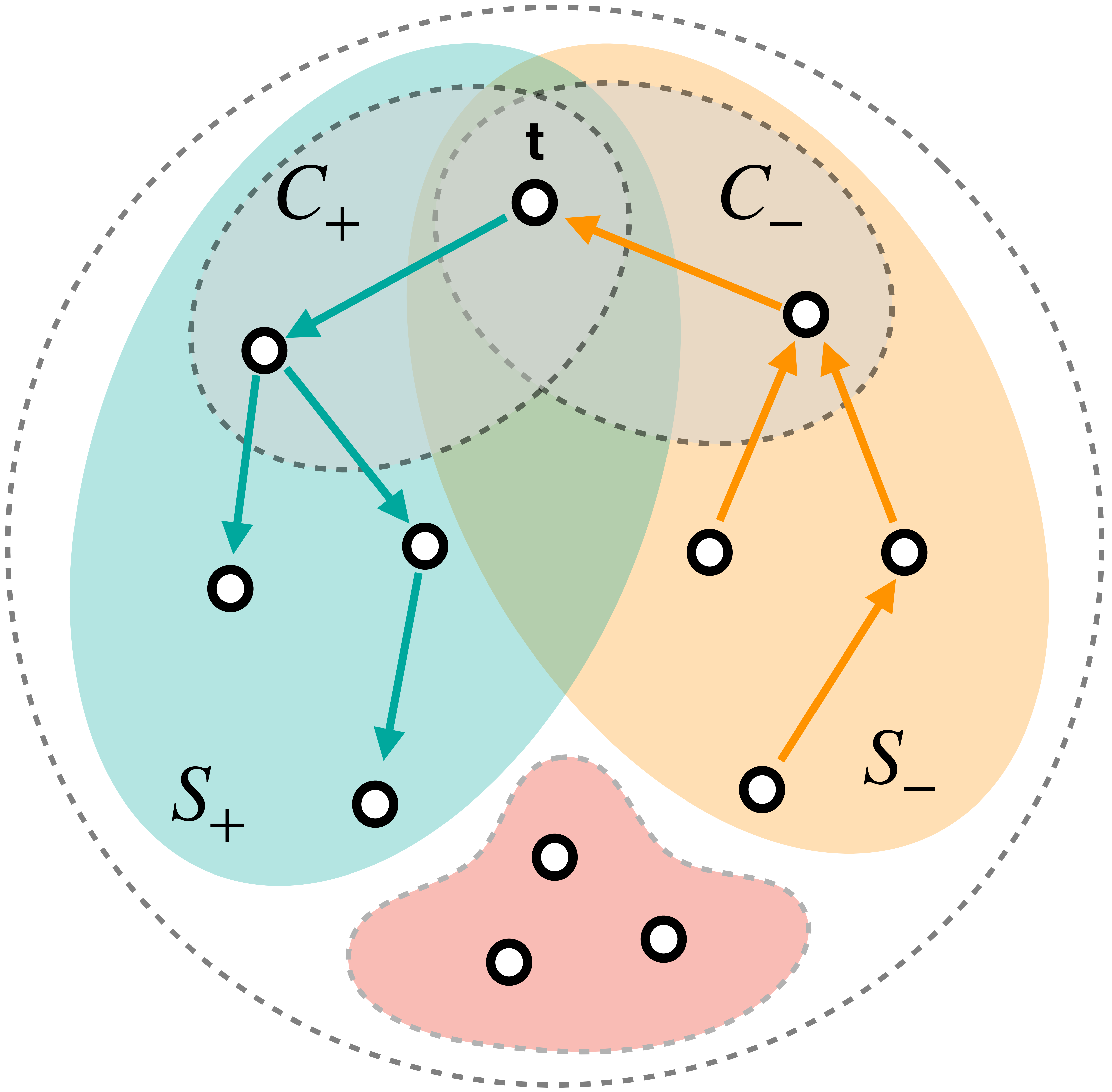
<u>Goal:</u> decide if the min-radius is $< O(R \log n)$ or >R

1. Define four sets $C_{+,-}, S_{+,-}$

   $C_+ = \{u \mid \mathrm{dist}(t,u) < R\}$

   $S_+ = \{u \mid \mathrm{dist}(t,u) < 2R\}$

2. **Define** $W = V \backslash (S_+ \cup S_-) \neq \emptyset$



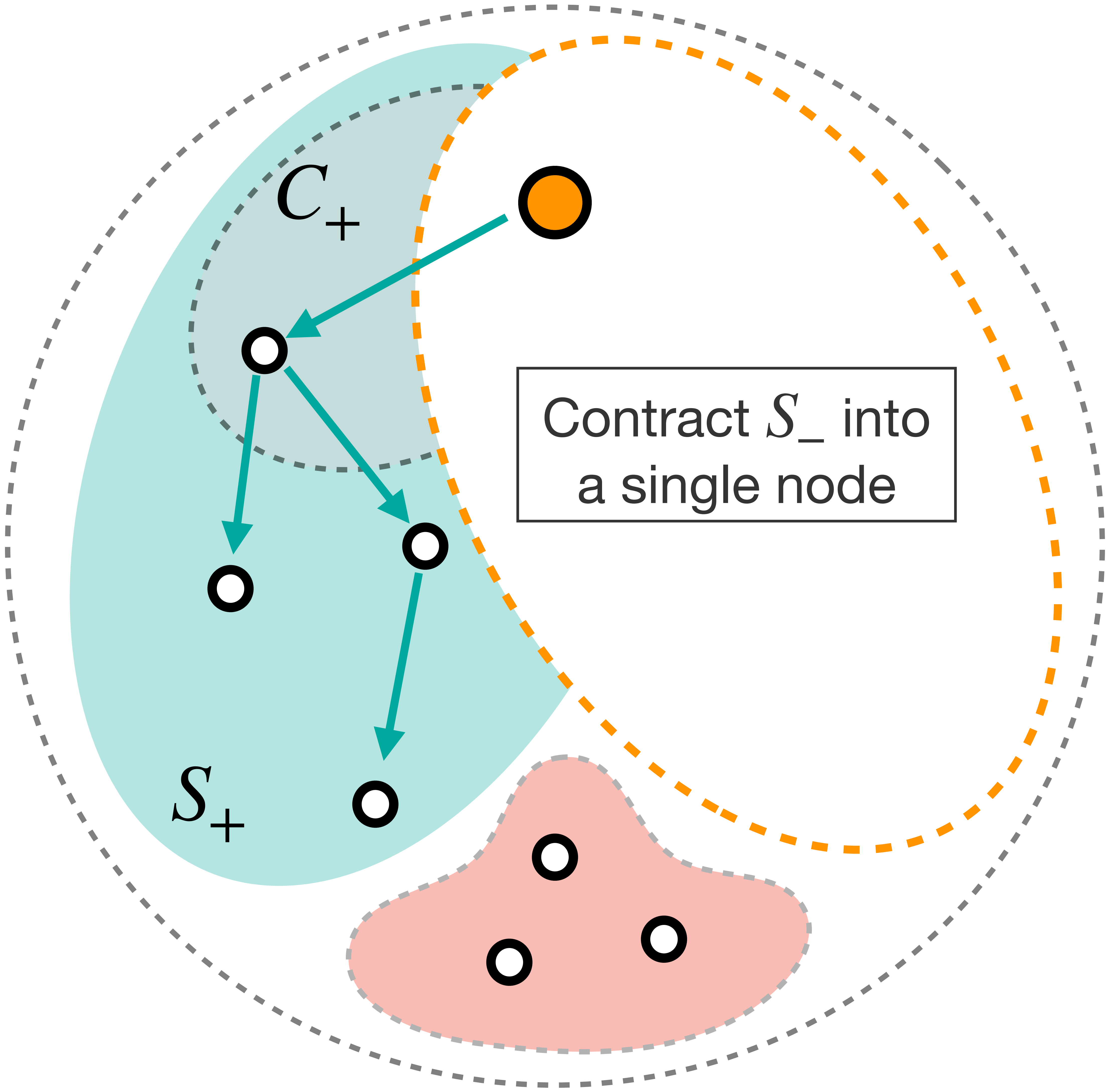
Contraction & Recurse

Goal: decide if the min-radius is $< O(R \log n)$ or >R

1. Define four sets $C_{+,-}, S_{+,-}$
   $C_+ = \{u \mid \mathrm{dist}(t,u) < R\}$
   $S_+ = \{u \mid \mathrm{dist}(t,u) < 2R\}$

2. Define $W = V \backslash (S_+ \cup S_-) \neq \varnothing$

3. **Contract S_-** into a single node, and **recurse** on the contracted graph (symmetrically for S_+)



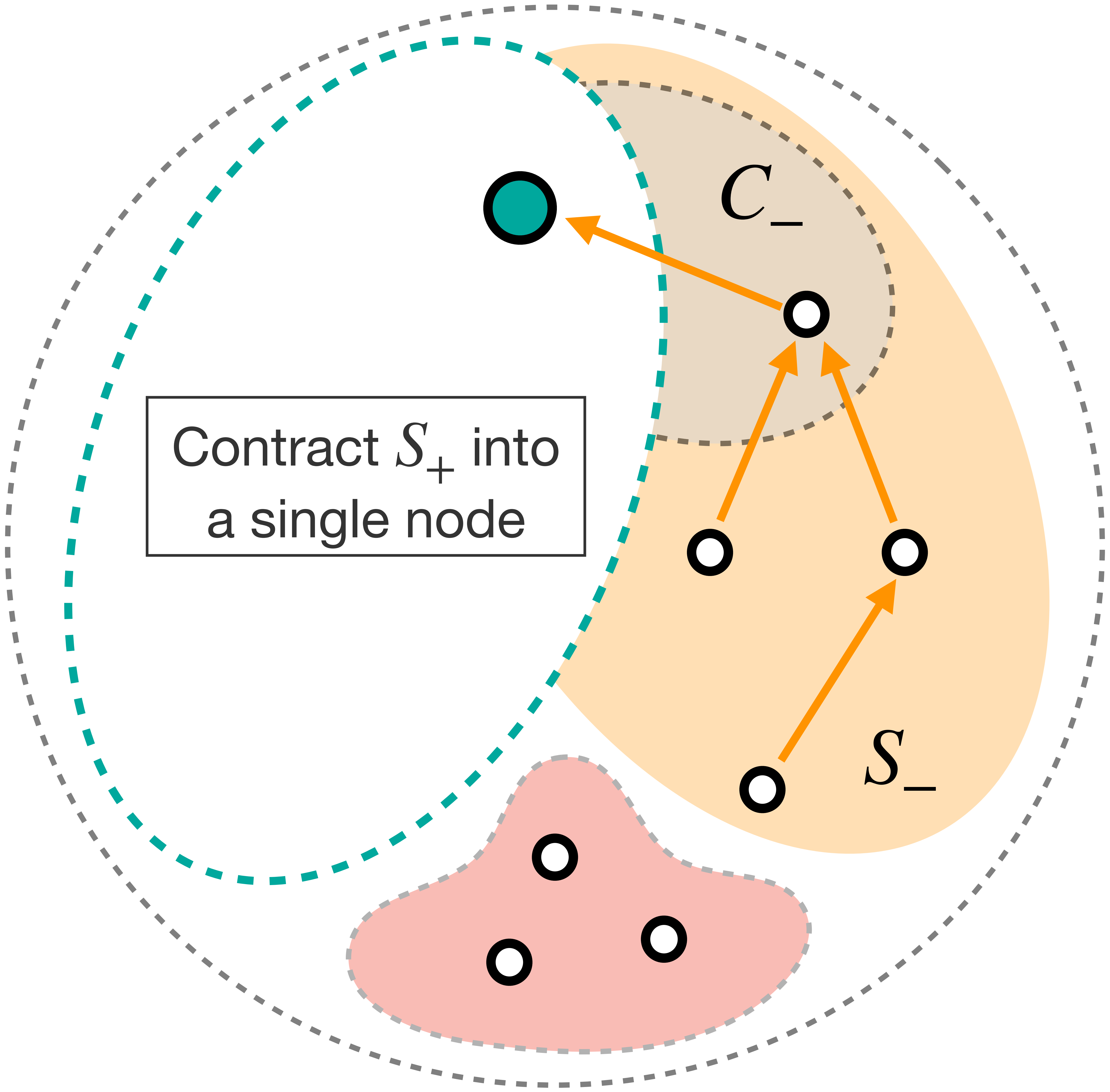
Contraction & Recurse

Goal: decide if the min-radius is
$< O(R \log n)$ or >R

1. Define four sets $C_{+,-}, S_{+,-}$
   $C_+ = \{u \mid \mathrm{dist}(t,u) < R\}$
   $S_+ = \{u \mid \mathrm{dist}(t,u) < 2R\}$

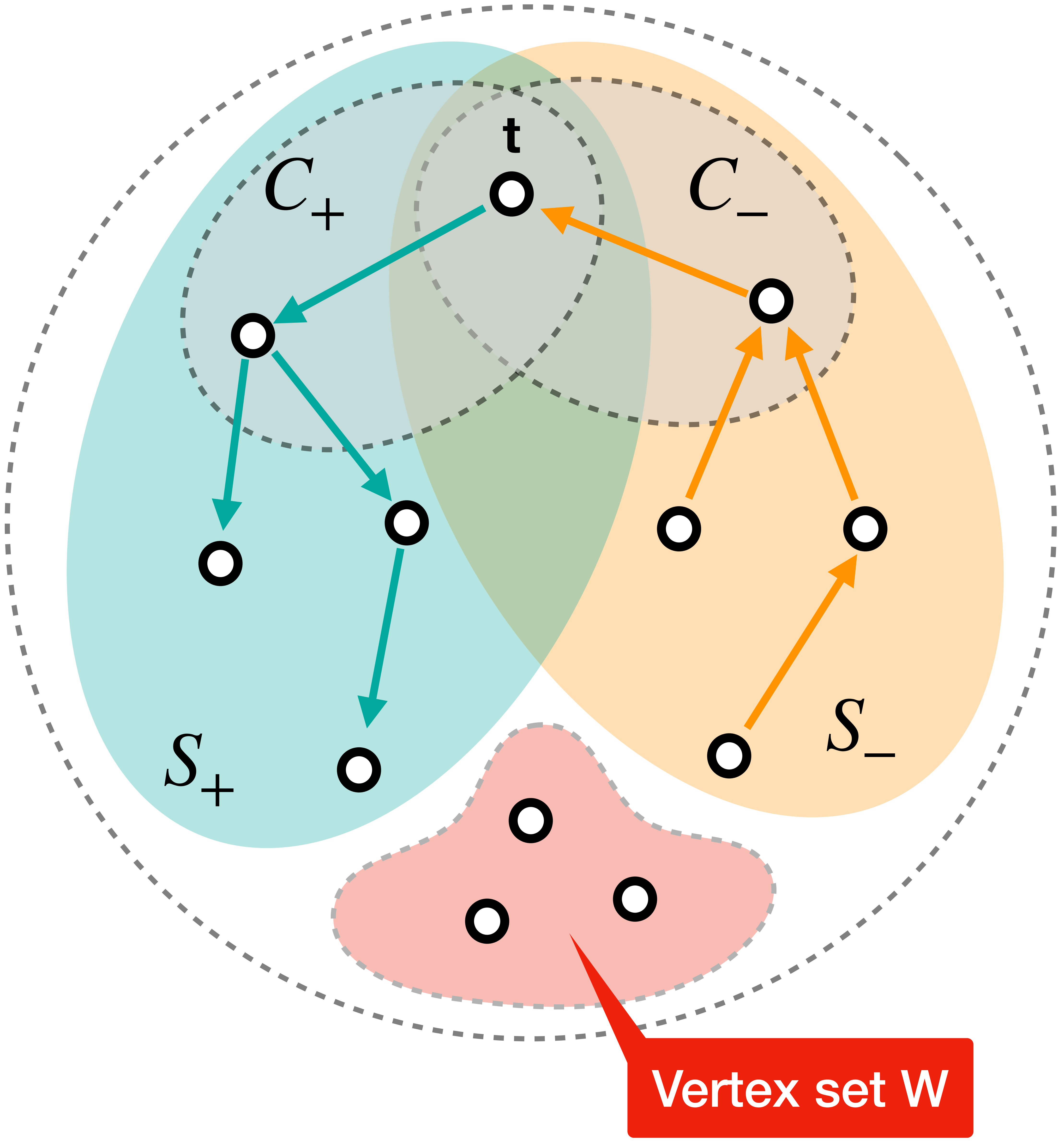
2. Define $W = V \backslash (S_+ \cup S_-) \neq \varnothing$

3. **Contract S_-** into a single node, and **recurse** on the contracted graph (symmetrically for S_+)



Contraction & Recurse

<u>Runtime issue:</u> $W = V \backslash (S_+ \cup S_-)$
appears in **both** recursion branches

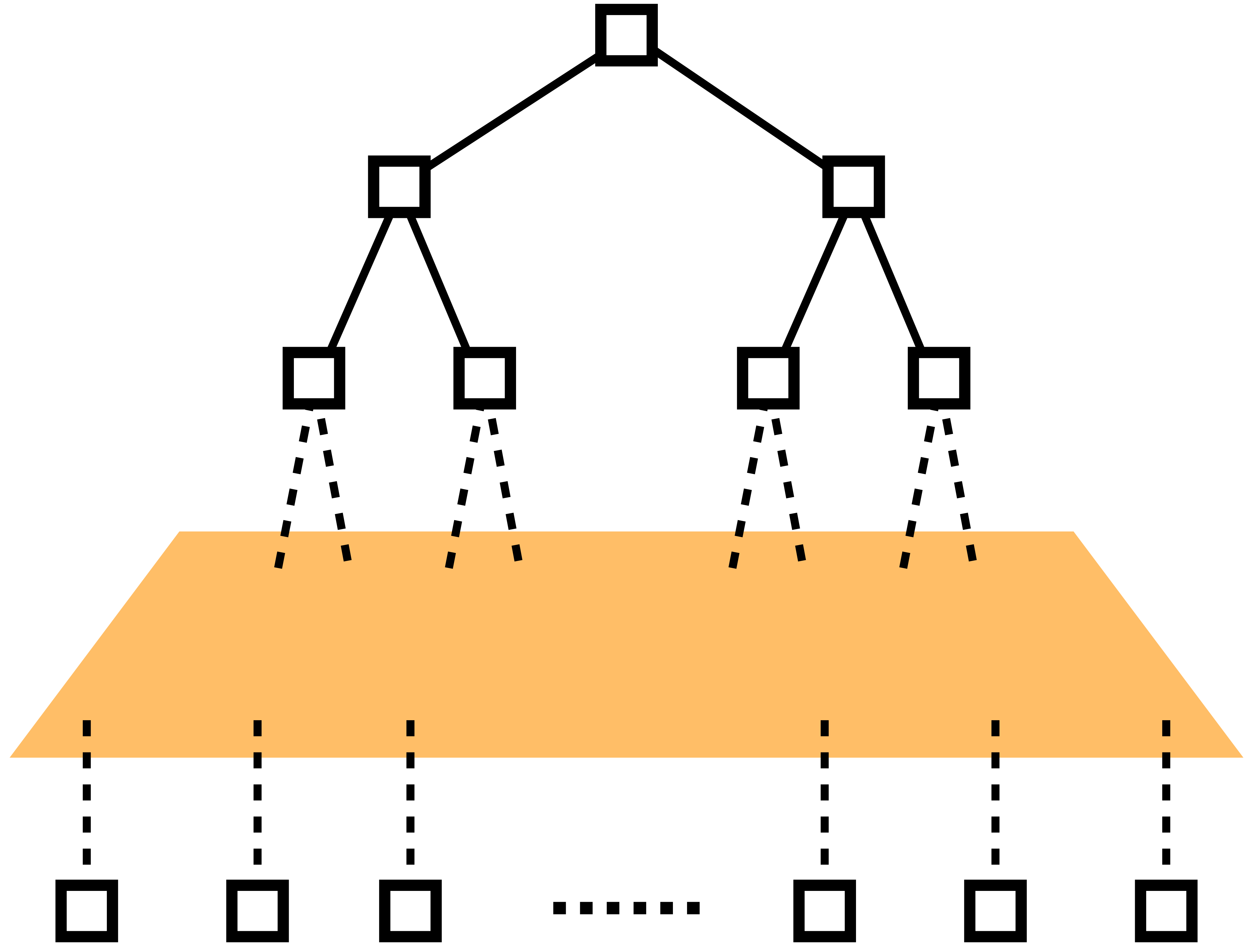


Contraction & Recurse

Runtime issue: $W = V \backslash (S_+ \cup S_-)$ appears in **both** recursion branches

- A single vertex could appear **everywhere** on the recursion tree

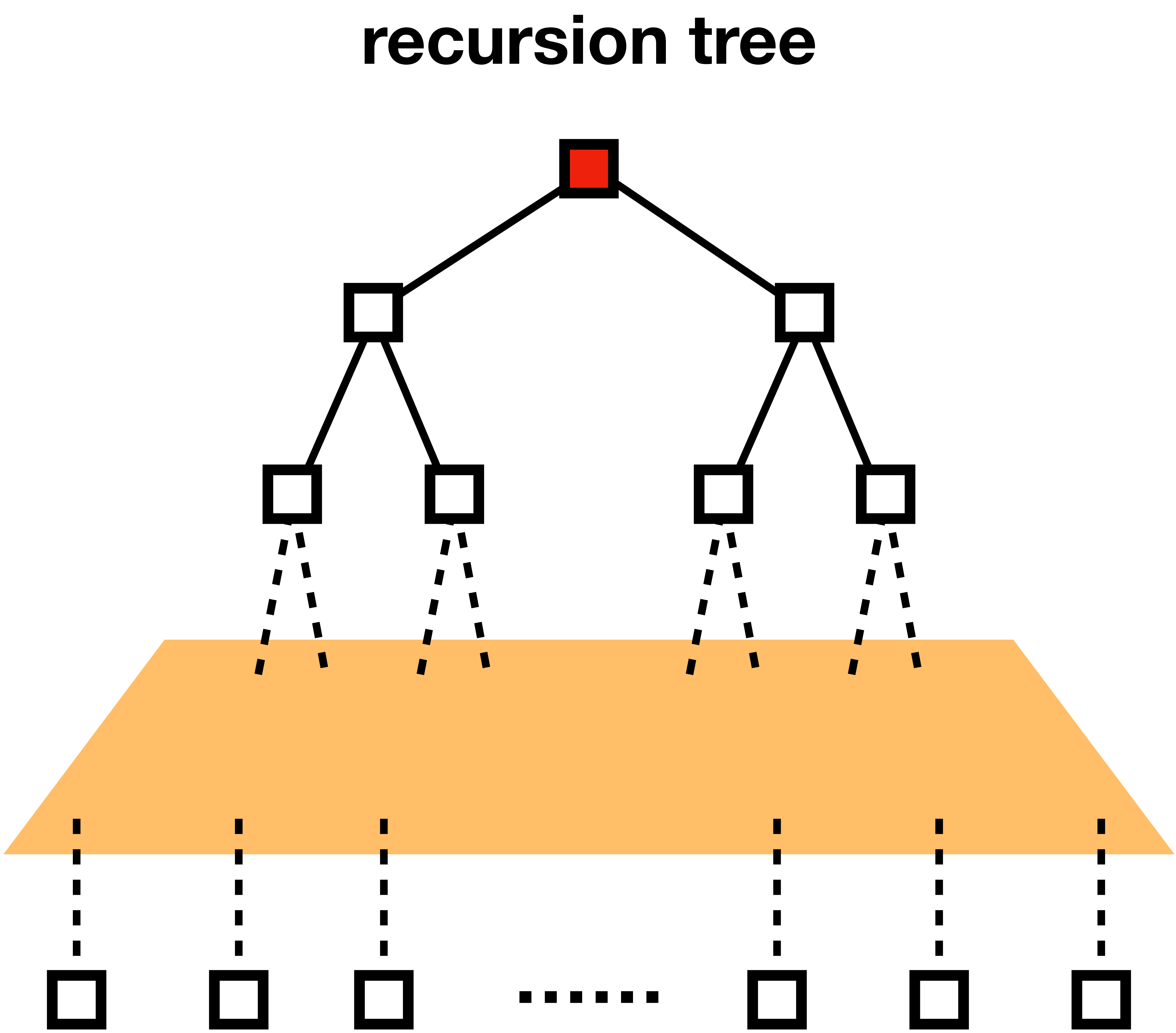
recursion tree



Contraction & Recurse

Runtime issue: $W = V \backslash (S_+ \cup S_-)$ appears in **both** recursion branches

- A single vertex could appear **everywhere** on the recursion tree

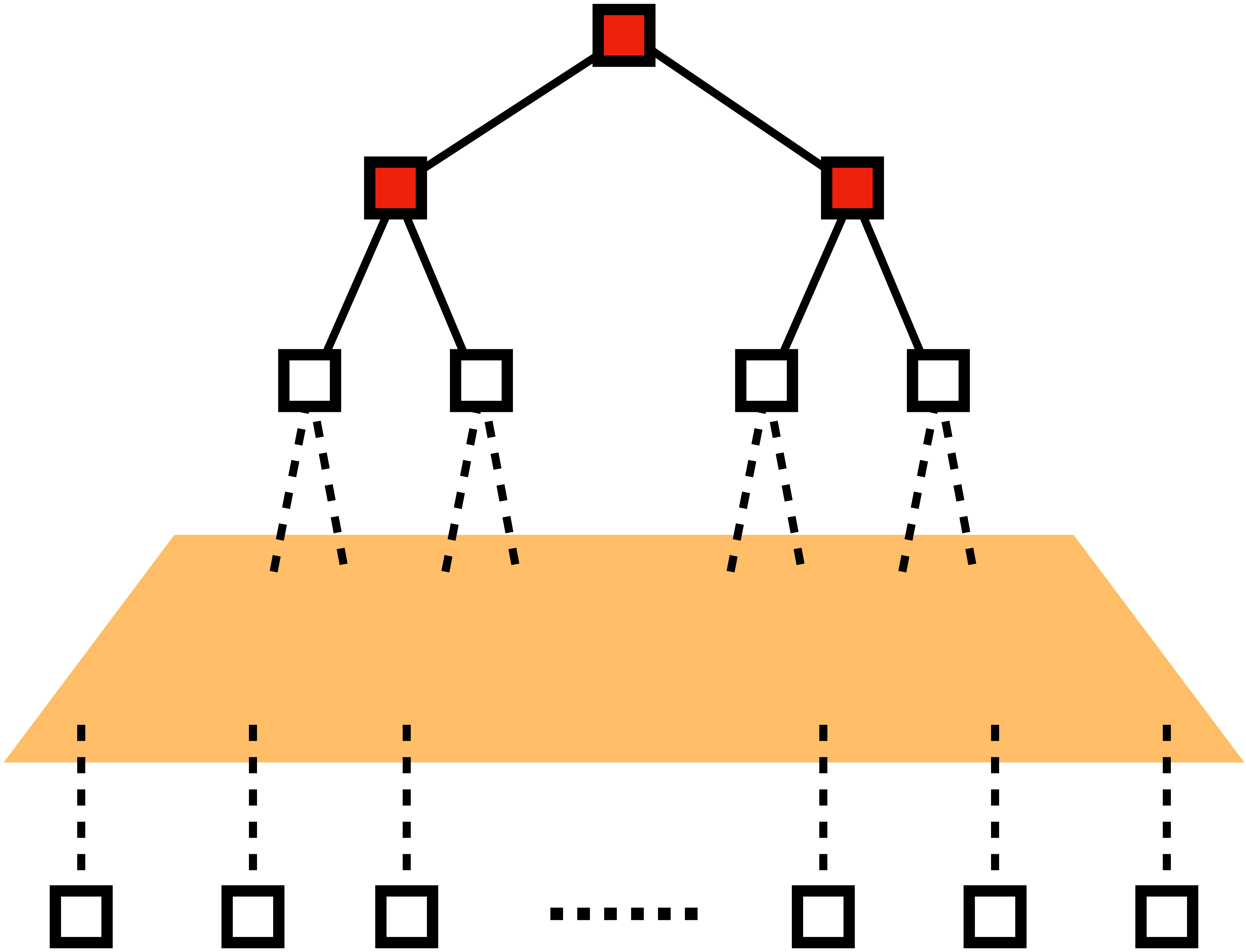


Contraction & Recurse

Runtime issue: $W = V \backslash (S_+ \cup S_-)$ appears in **both** recursion branches

- A single vertex could appear **everywhere** on the recursion tree

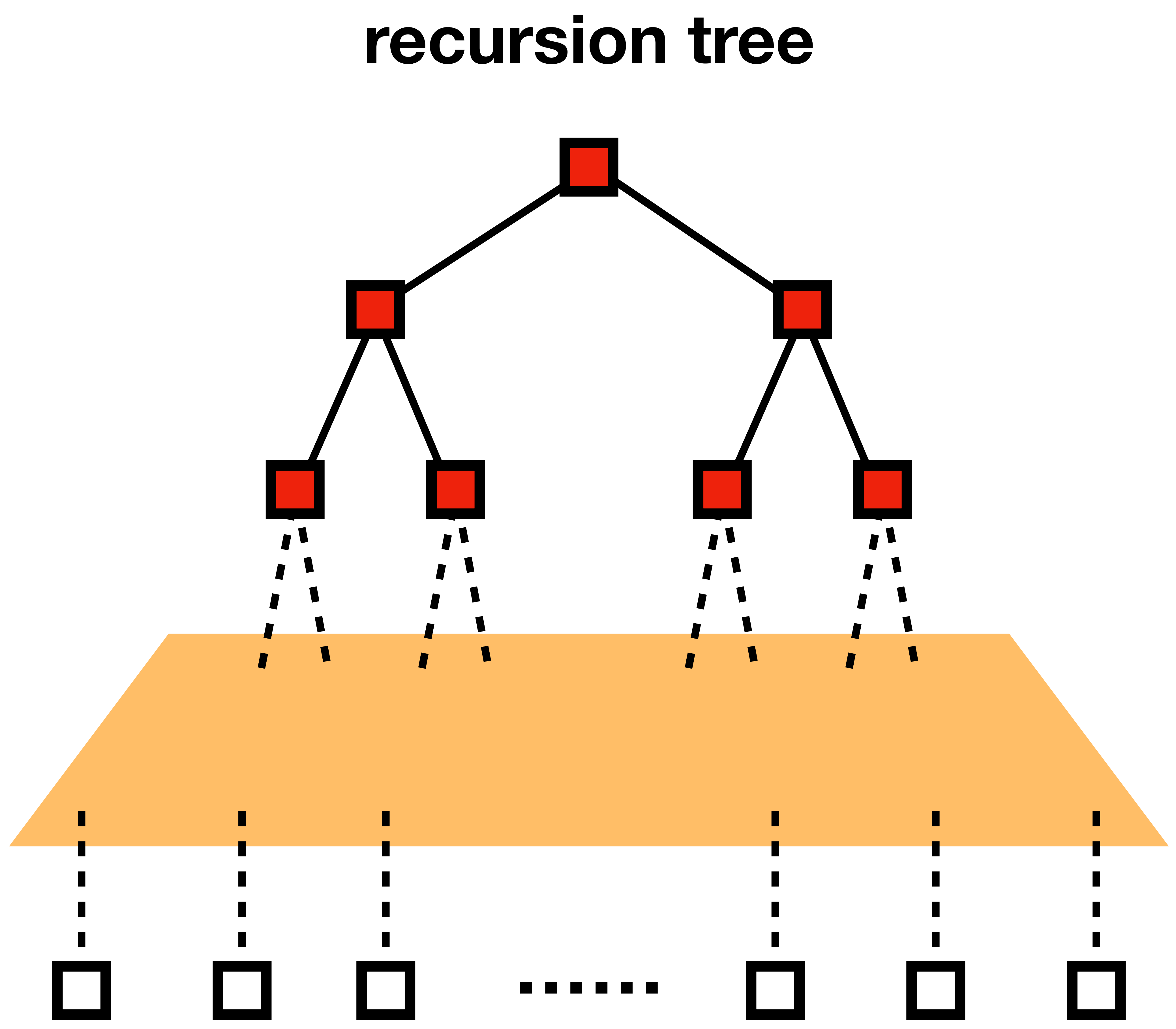
recursion tree



Contraction & Recurse

<u>Runtime issue:</u> $W = V \backslash (S_+ \cup S_-)$ appears in **both** recursion branches

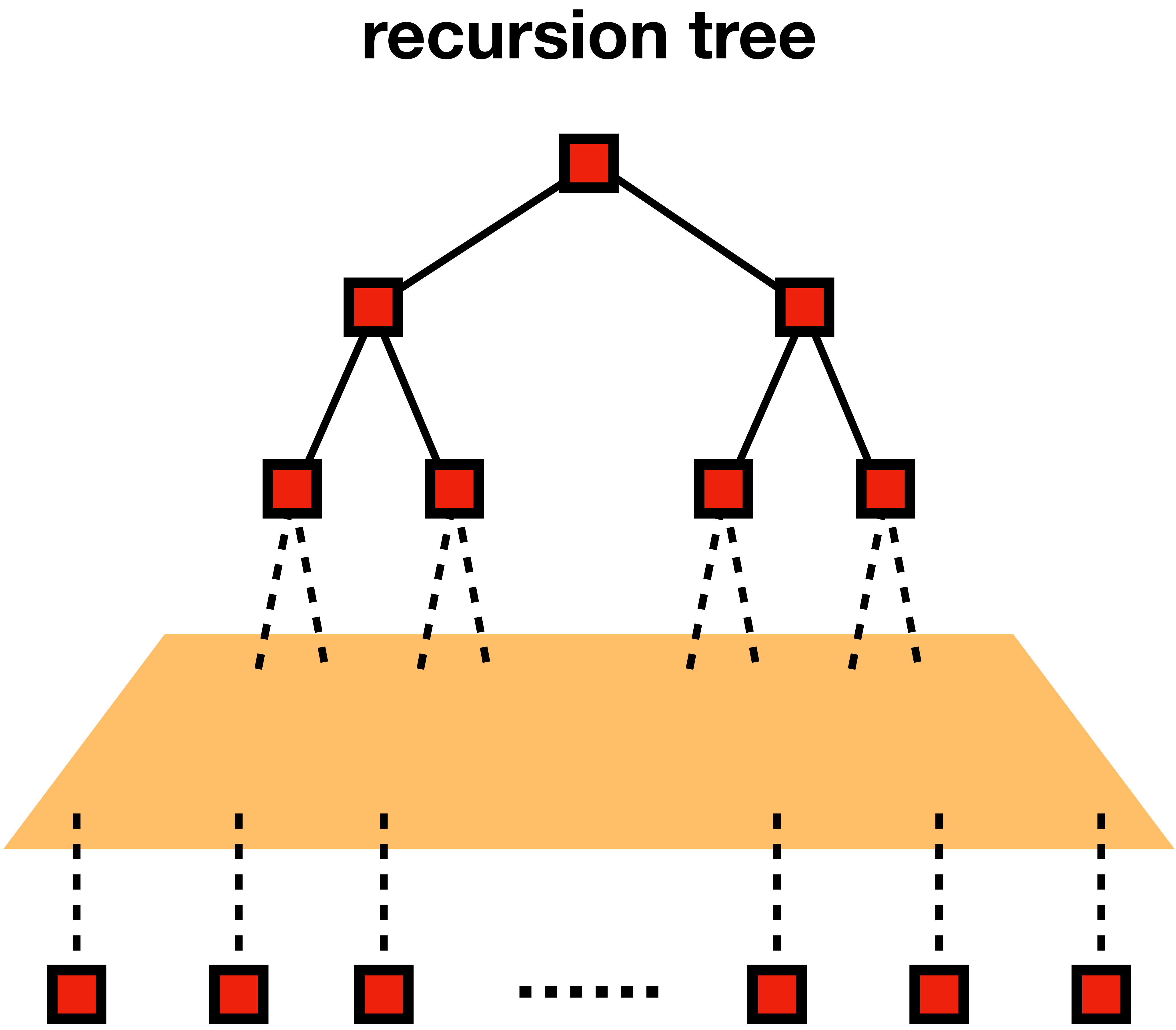
- A single vertex could appear **everywhere** on the recursion tree



Contraction & Recurse

<u>Runtime issue:</u> $W = V \backslash (S_+ \cup S_-)$ appears in **both** recursion branches

- A single vertex could appear **everywhere** on the recursion tree



Contraction & Recurse

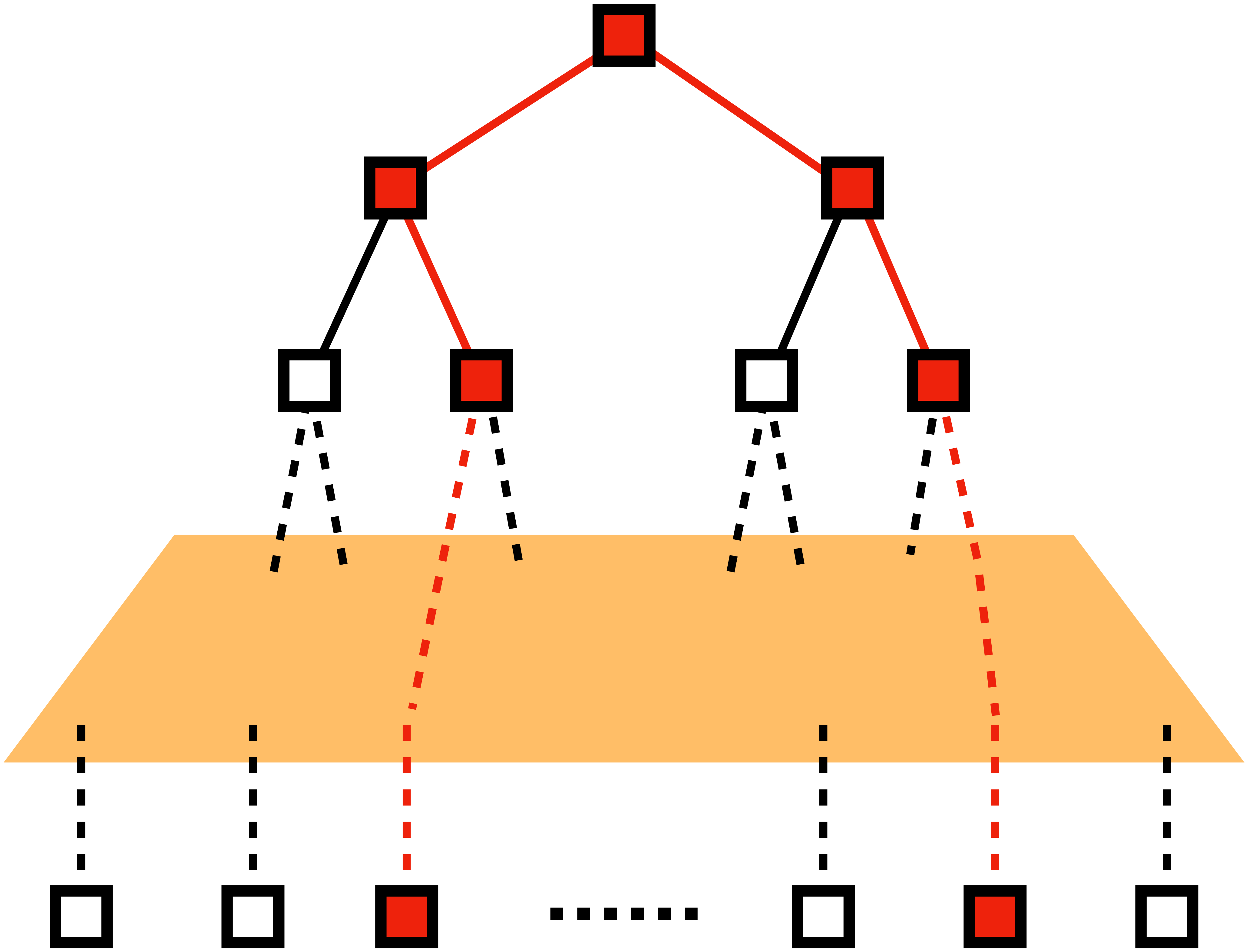
Runtime issue: $W = V \backslash (S_+ \cup S_-)$
appears in **both** recursion branches

- A single vertex could appear **everywhere** on the recursion tree

- **Pruning while recursing:**
  Every vertex appears in at most two branches on the recursion tree

  (Details omitted in this presentation)

recursion tree



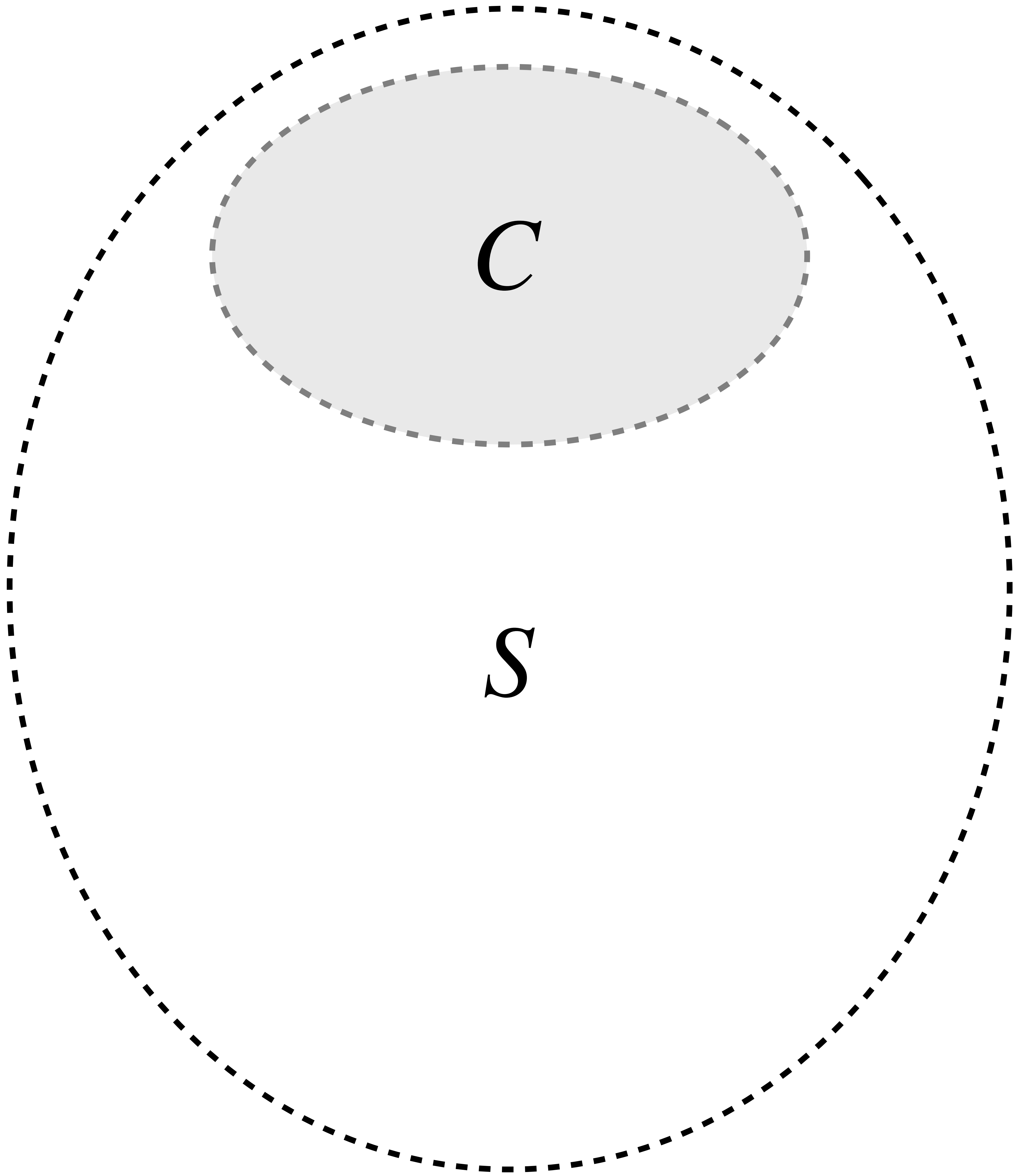
Each vertex only appears in at most **two branches**

Brief sketch of
4-approx of min-radius

Achieving 4-approx

Goal: decide if the min-radius is <4R or >R

- Cannot use contractions

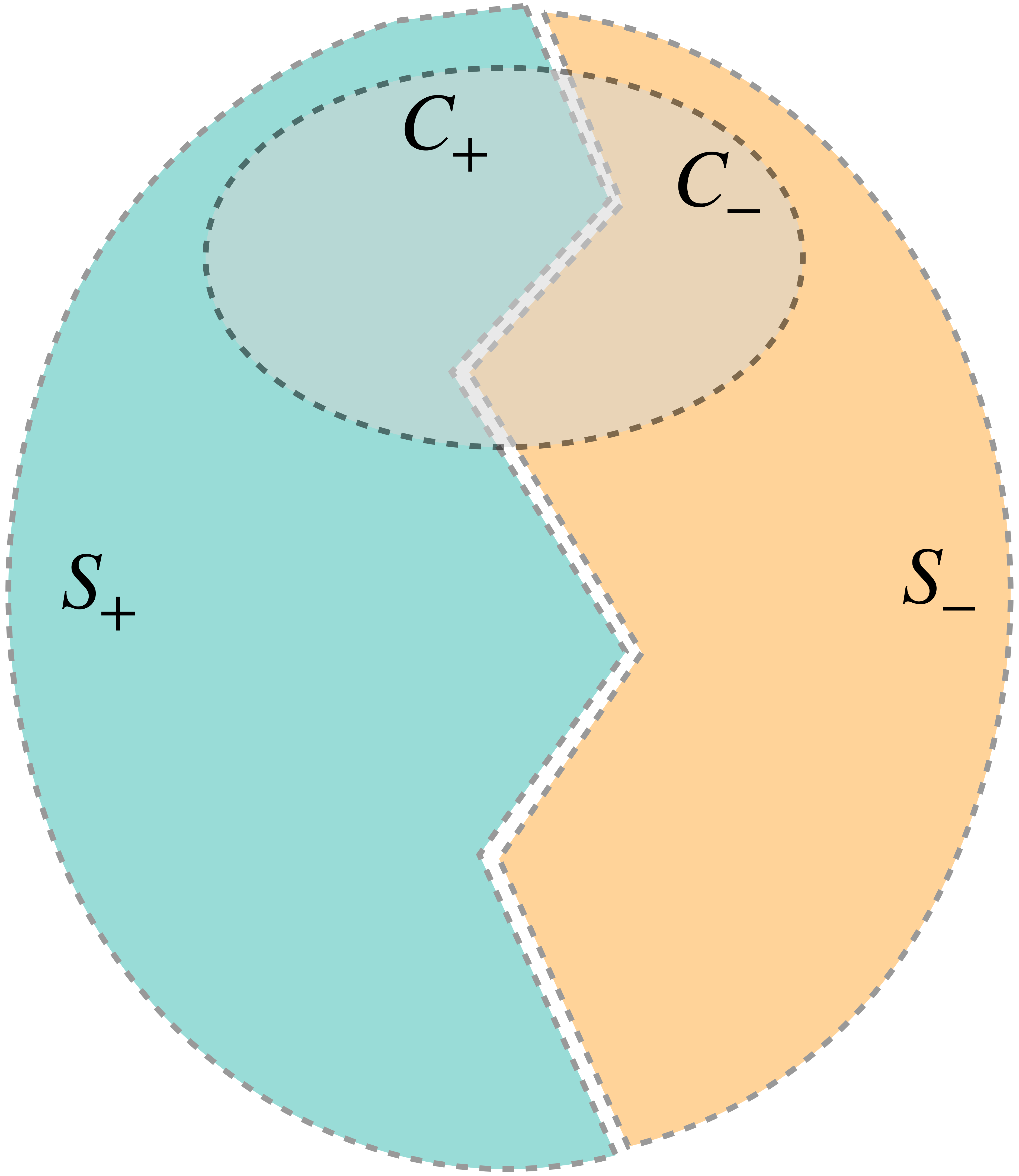


Achieving 4-approx

<u>Goal:</u> decide if the min-radius is
<4R or >R

- Cannot use contractions

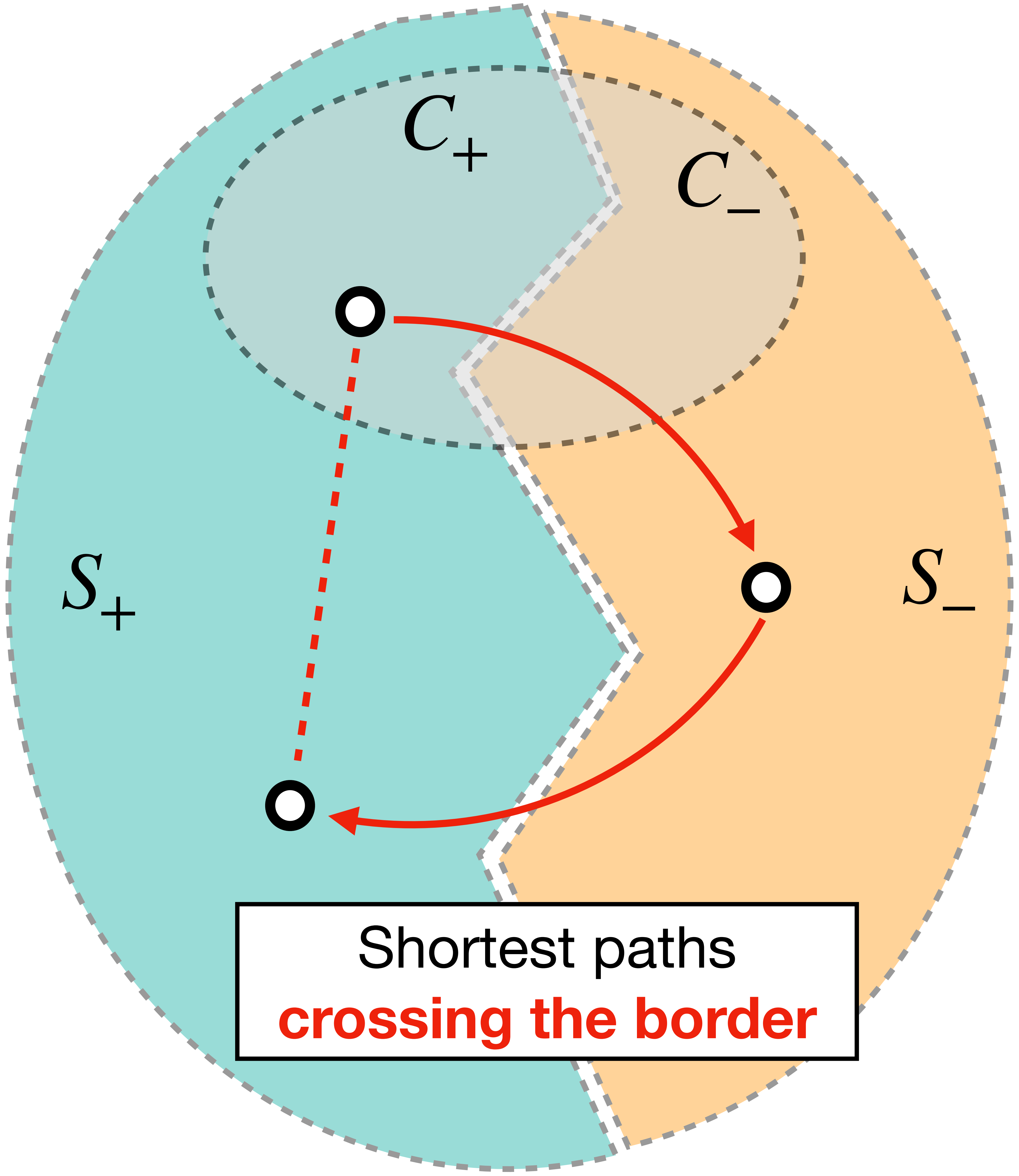
- Partition $S = S_+ \cup S_-$
  Recurse on $(S_+, C_+), (S_-, C_-)$



Achieving 4-approx

<u>Goal:</u> decide if the min-radius is <4R or >R

- Cannot use contractions
- Partition $S = S_+ \cup S_-$
  Recurse on $(S_+, C_+), (S_-, C_-)$
- **Impossible** to preserve pairwise distances in subgraphs



Achieving 4-approx

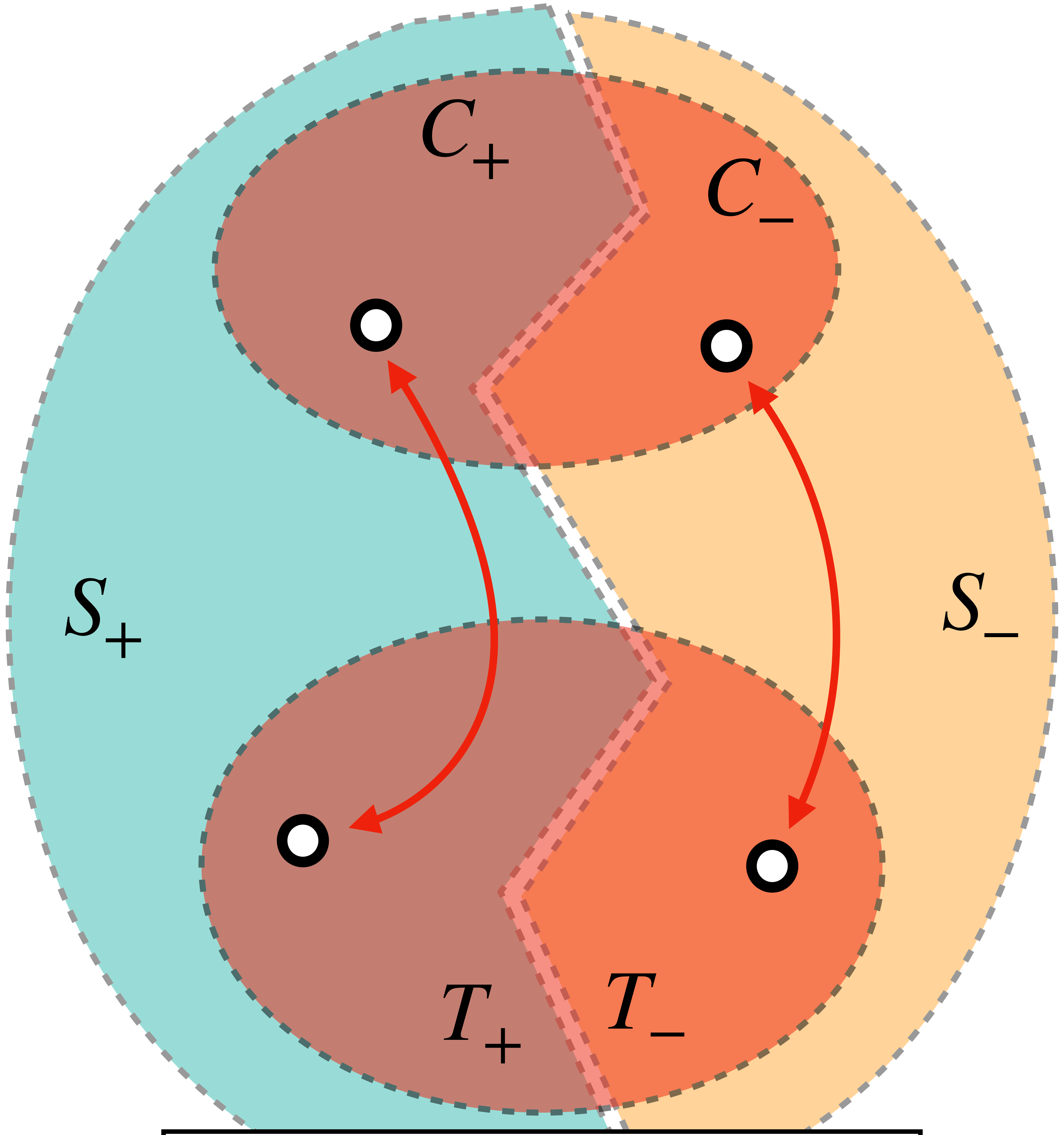
<u>Goal:</u> decide if the min-radius is
<4R or >R

- Cannot use contractions

- Partition $S = S_+ \cup S_-$
  Recurse on (S_+, C_+), (S_-, C_-)

- **Impossible** to preserve pairwise distances in subgraphs

- **<u>Solution:</u>** Only preserve some pairs of distances, i.e. $C \times (C \cup T)$



Only preserve distances between $C \times (C \cup T)$

Further questions

- Better approximation ratio in near-linear time?

  Min-diam/radius in general digraphs:

  3 in $\tilde{O}(mn^{1/2})$ vs **4** in $\tilde{O}(m)$

Thanks for listening