Near-Optimal Approximate Dual-Failure Replacement Paths

Shiri Chechik Tianyi Zhang

Tel Aviv University

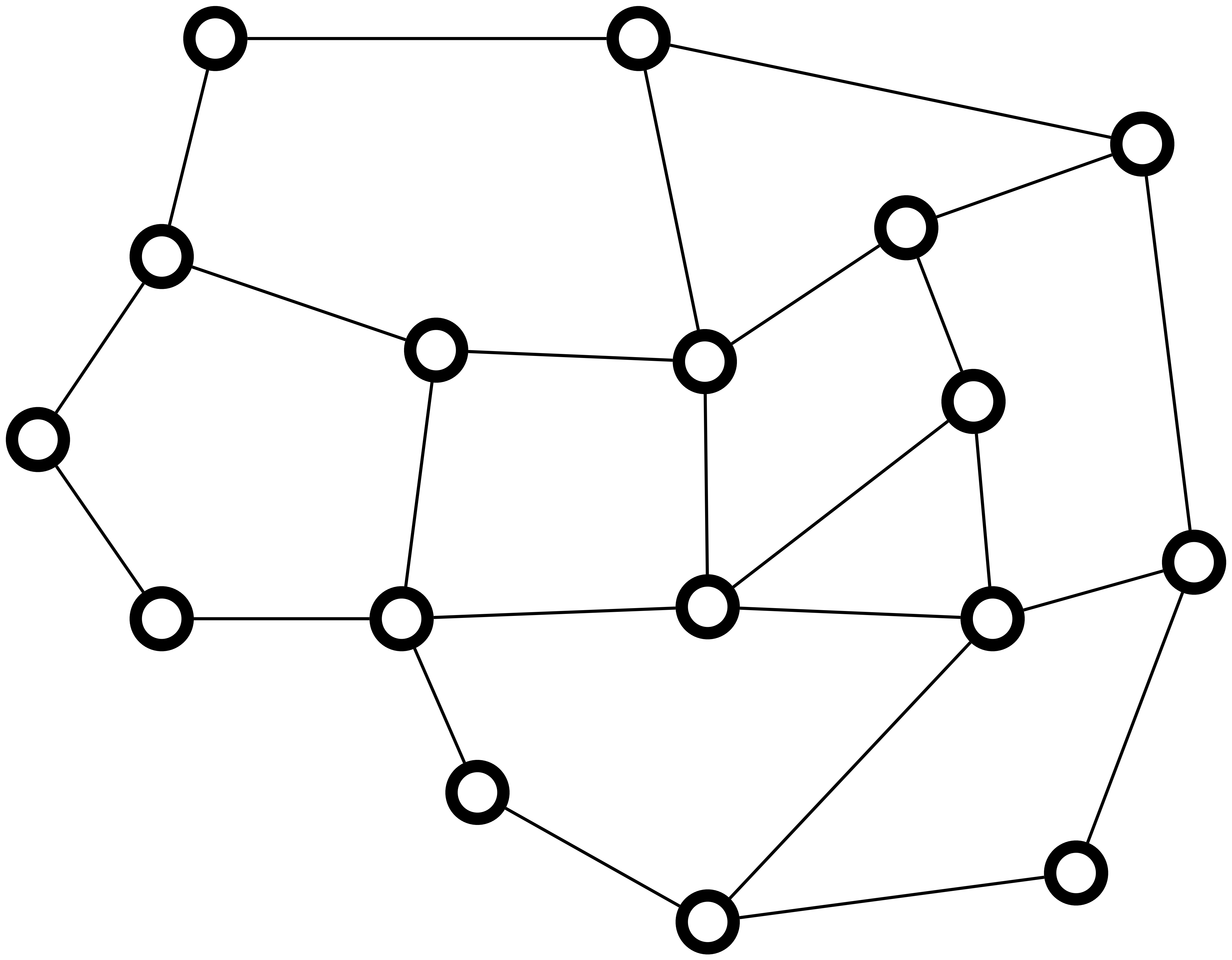
Emergency Planning

Two-phase problem:

1. Preprocess the input graph
2. One/multiple links break, and recover info in the new graph

Costs:

1. Preprocessing time
2. Recovery time



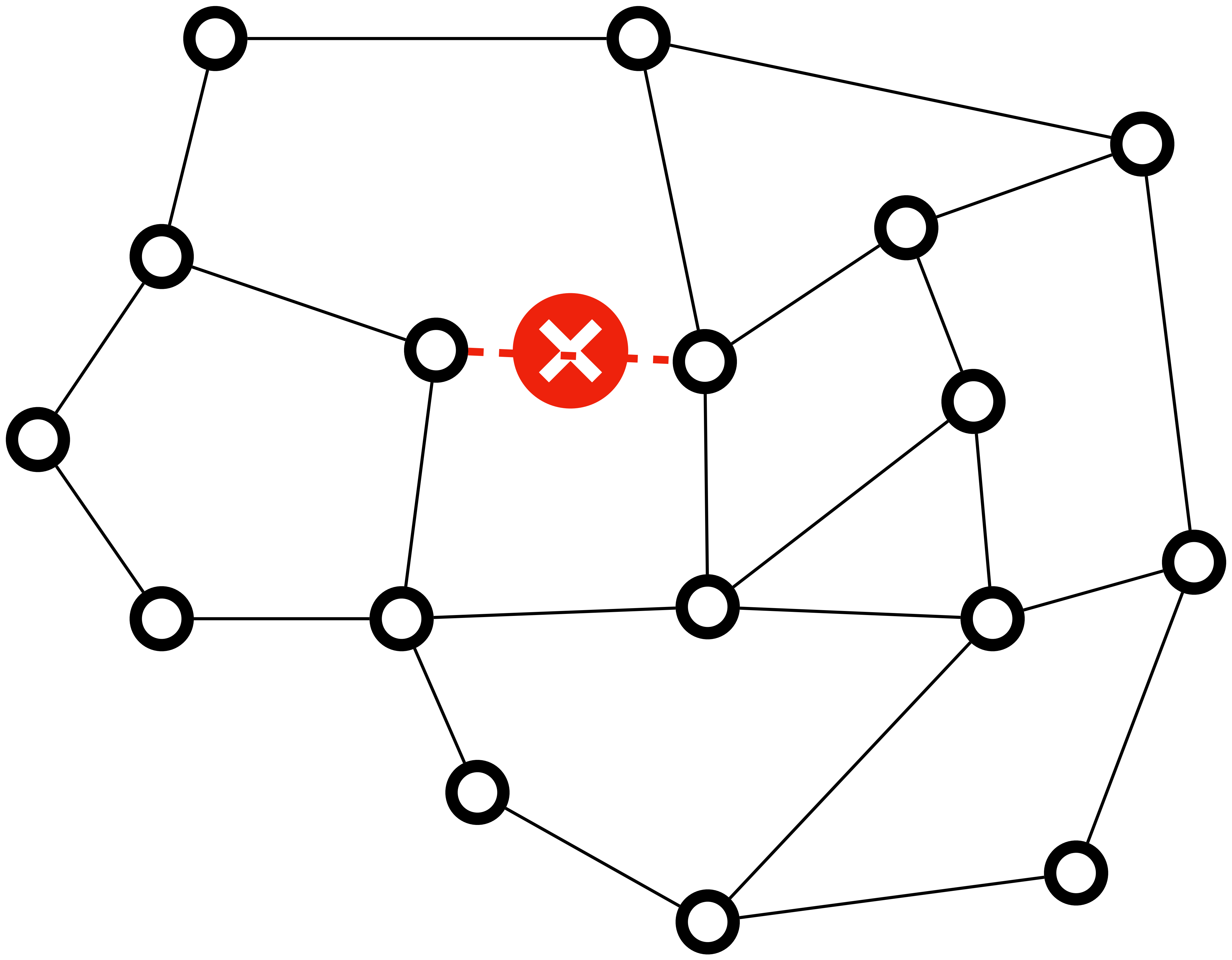
Emergency Planning

Two-phase problem:

1. Preprocess the input graph
2. **One/multiple links break**, and recover info in the new graph

Costs:

1. Preprocessing time
2. Recovery time



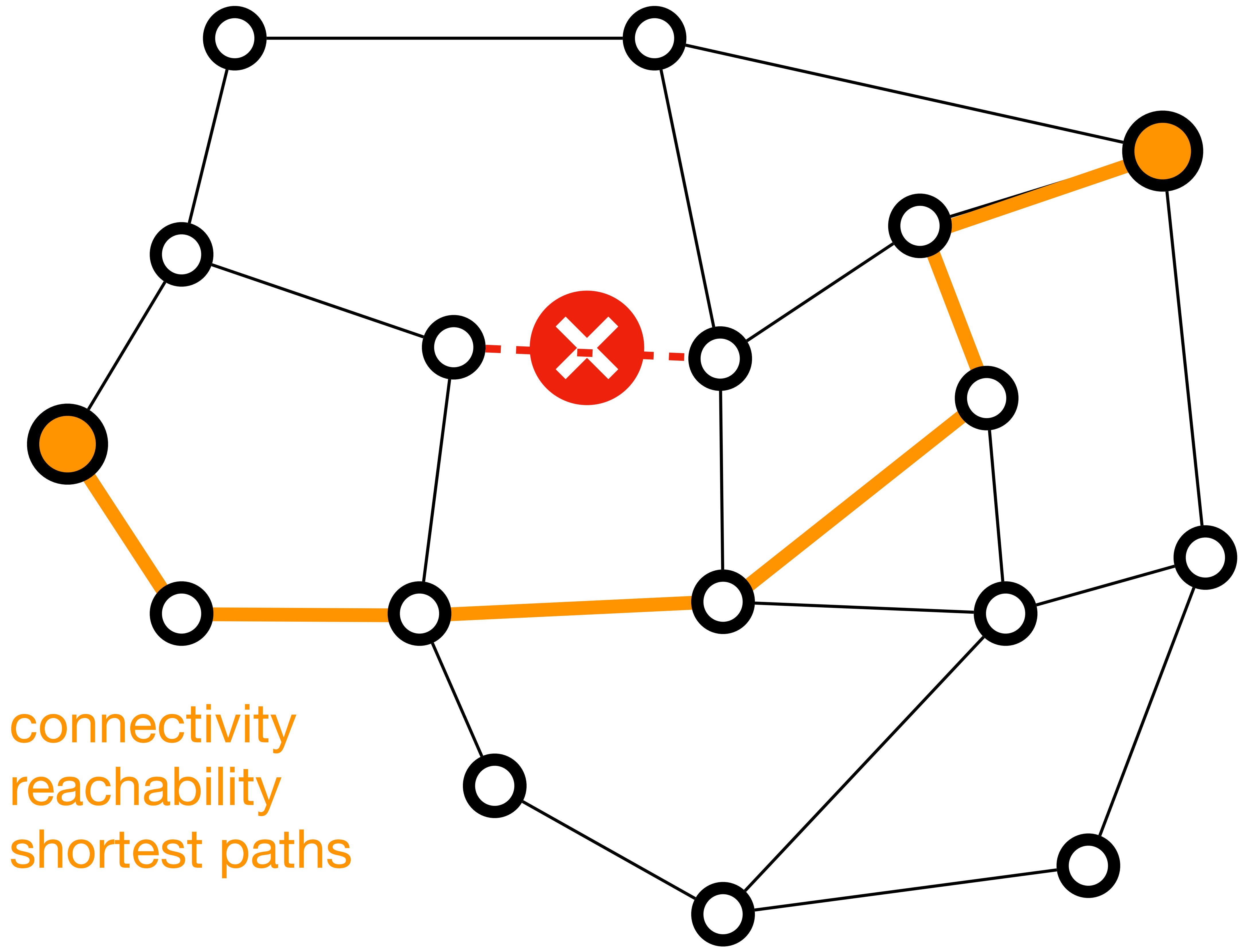
Emergency Planning

Two-phase problem:

1. Preprocess the input graph
2. **One/multiple links break**, and **recover info** in the new graph

Costs:

1. Preprocessing time
2. Recovery time



Emergency Planning

Two-phase problem:

1. Preprocess the input graph
2. **One/multiple links break**, and **recover info** in the new graph

Costs:

1. Preprocessing time
2. Recovery time

Today's focus:

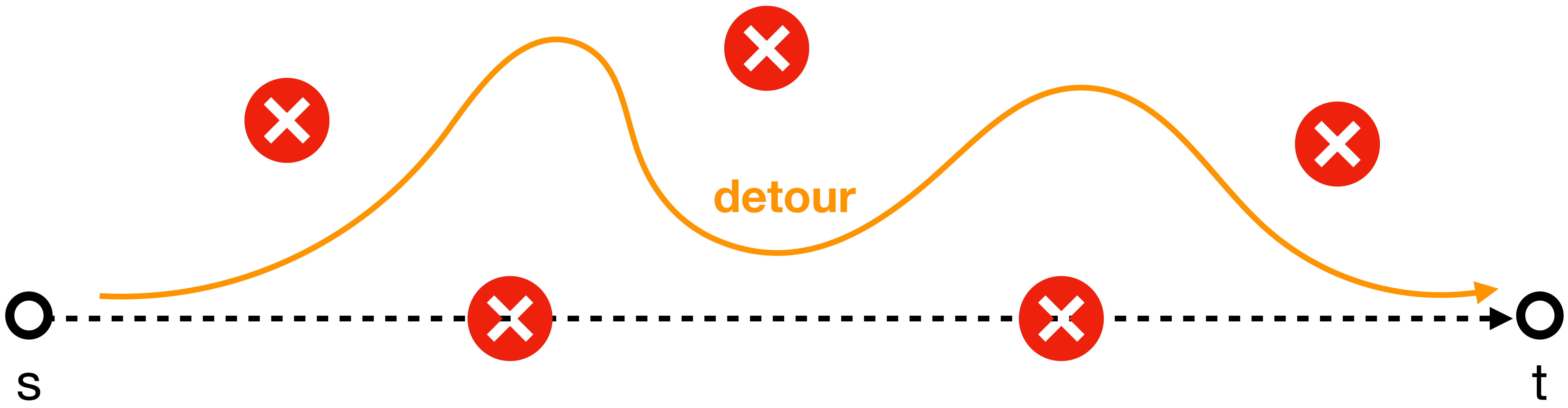
1. Recover **shortest paths info**
2. **Precompute** all answers, so recovery time is poly-log

Other settings:

1. Connectivity / reachability
2. Preprocessing vs. recovery

Replacement Path

- Weighted directed graph $G = (V, E, \omega)$, source & terminal vertices
- For all $\leq f$ edges $F \subseteq E$, compute dist$(s, t, G \backslash F)$



Replacement Path

- Weighted directed graph $G = (V, E, \omega)$, source & terminal vertices
- For all $\leq f$ edges $F \subseteq E$, compute dist$(s, t, G \backslash F)$
- Total size of output $\leq n^f$ (**exercise:** why not m^f)
- Trivial algorithm takes time $mn^f \leq n^{f+2}$
- **Main question:** How to save the **quadratic overhead**?

Single-failure replacement paths

- Trivial algorithm takes cubic runtime $mn \leq n^3$
- [VW, 2010] showed this is the best possible under a widely believed conjecture
- $(1+\epsilon)$-approximations in runtime $m \leq n^2$ [Bernstein, 2010]
- **Corollary:** $(1+\epsilon)$-approximations for f-failures in $mn^{f-1} \leq n^{f+1}$ time

Dual-failure replacement paths

- Optimal exact algorithm in runtime n^3 [VWX, 2022]
- **Corollary:** exact solutions for f-failures in n^{f+1} time when $f \geq 2$

Our result [CZ, 2024]

- $(1+\epsilon)$-approximations in runtime n^2, **optimal** runtime
- **Corollary:** $(1+\epsilon)$-approximations for f-failures in n^f time when $f \geq 2$
- **Open:** exact solutions for 3-failures in n^3 time?

Summary of results

	$f = 1$	$f = 2$	$f \geq 3$
Exact	n^3 [VW, 2010]	n^3 [VWX, 2022] [VW, 2010]	n^{f+1} [VWX, 2022]
Approximate	n^2 [Bernstein, 2010]	n^2 **New**	n^f **New**

Different variants of RP (exact)

Special cases of single-failure RP

- Undirected RP in linear time [NPW, 2001]
- Unweighted RP in $m\sqrt{n}$ time [RZ, 2012]
- Small edge weights RP in Wn^{ω} time [CN, 2020]

Single-failure single-source RP

- Unweighted single-source RP in $m\sqrt{n}$ time [CM, 2020]
- Small edge weights single-source RP in $W^{0.805}n^{2.496}$ time [GPWX, 2021]

Single-failure all-pairs RP

- All-pairs RP in $Wn^{2.58}$ time [GR, 2021]

Today’s plan

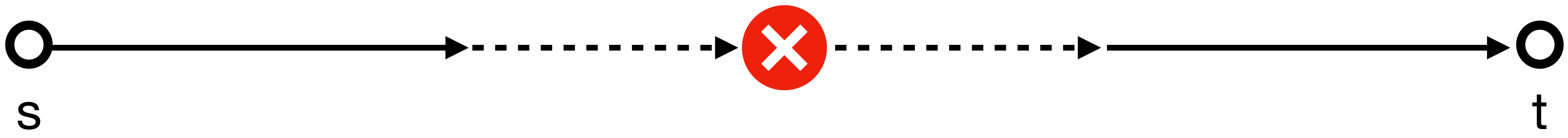
- Review of single-failure approximate st-RP [Bernstein, 2010]
- Two main cases for dual-failure approximate st-RP
  - Only one failure is on the st-path
  - Both failures are on the st-path

Single-Failure Approx-RP

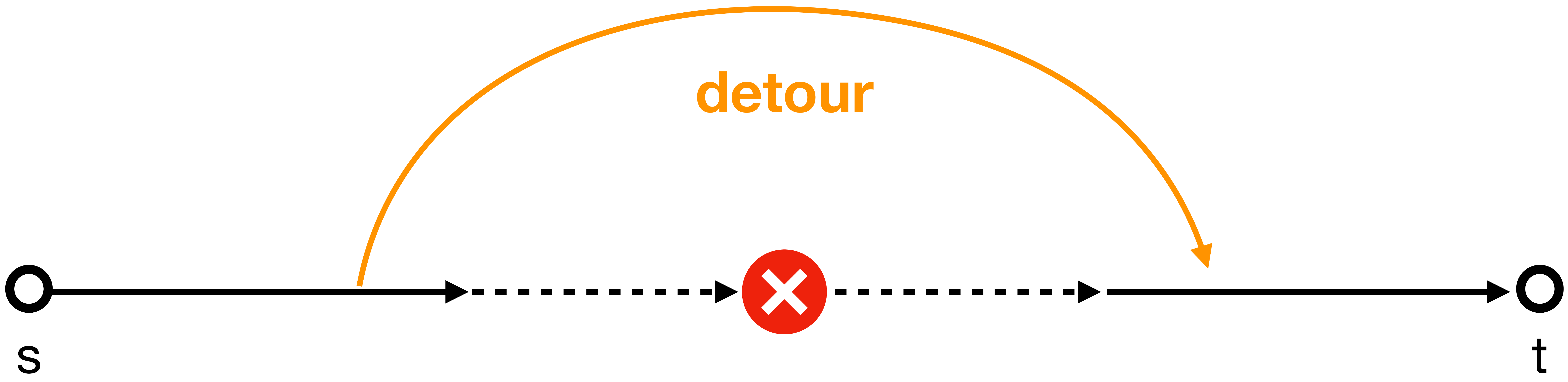
[Bernstein’10]

Single-Failure RP

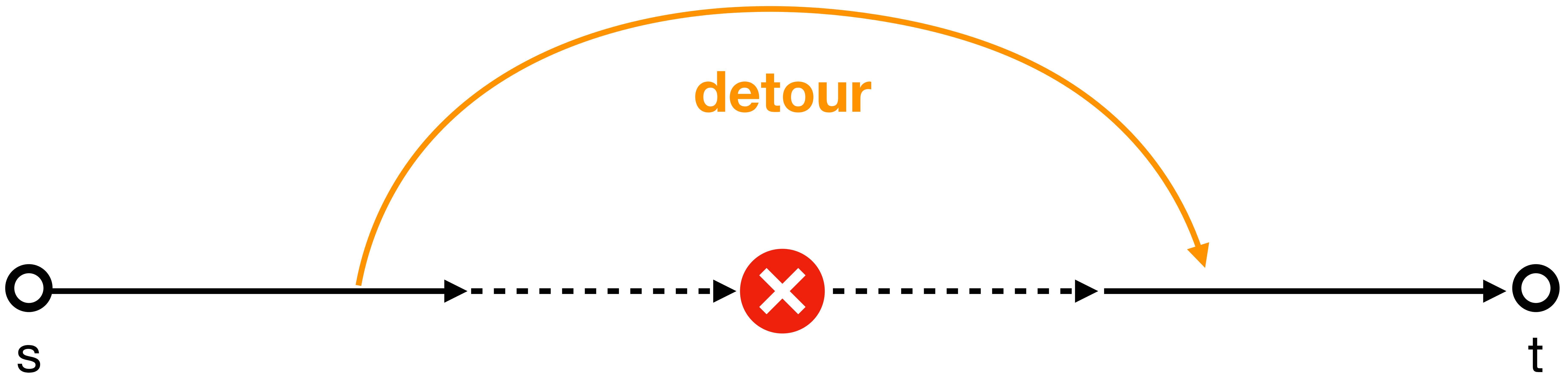
Single-Failure RP



Single-Failure RP

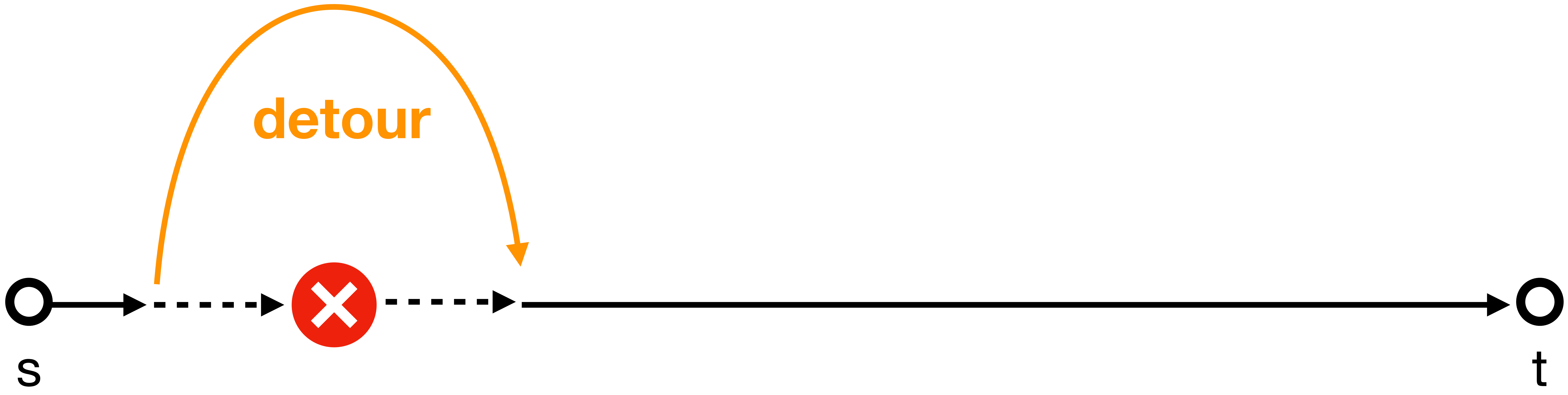


Single-Failure RP



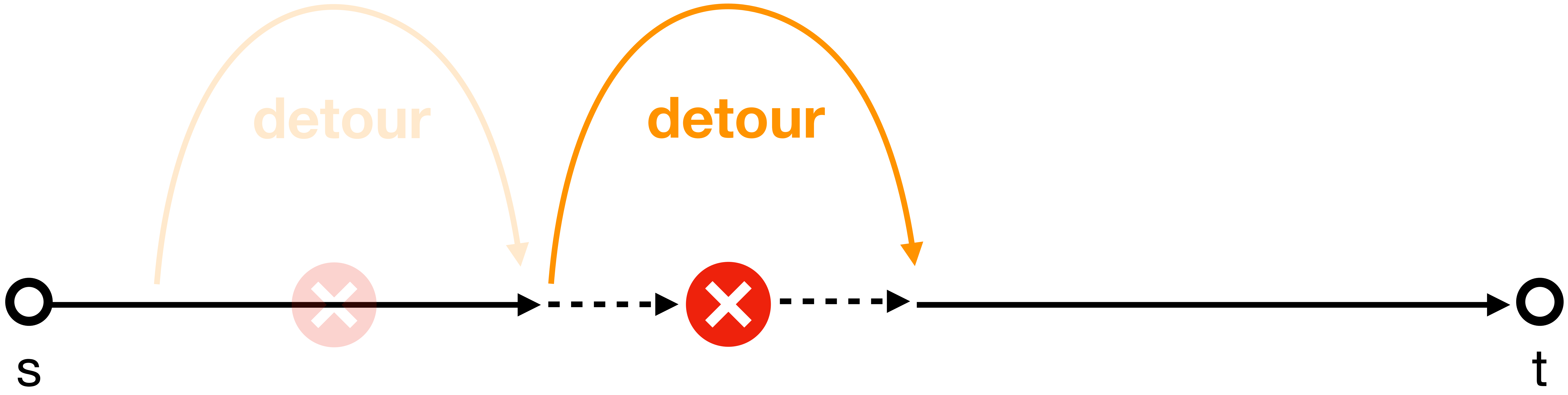
- Single-failure RP = best detour avoiding intervals
- **Compute the detours** for **all** possible failures

Single-Failure RP



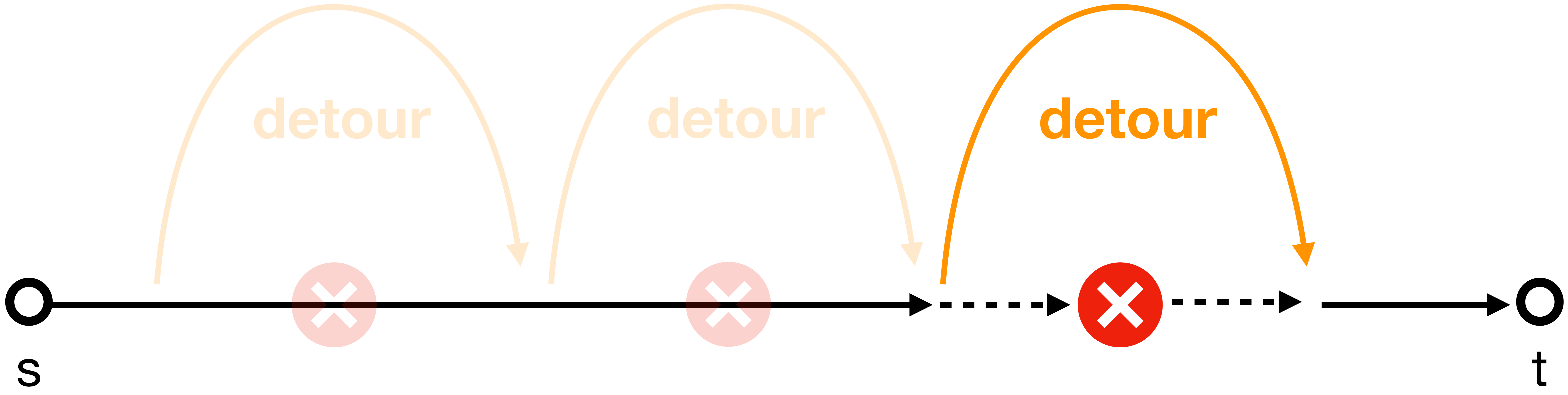
- Single-failure RP = best detour avoiding intervals
- **Compute the detours** for **all** possible failures

Single-Failure RP



- Single-failure RP = best detour avoiding intervals
- **Compute the detours** for **all** possible failures

Single-Failure RP



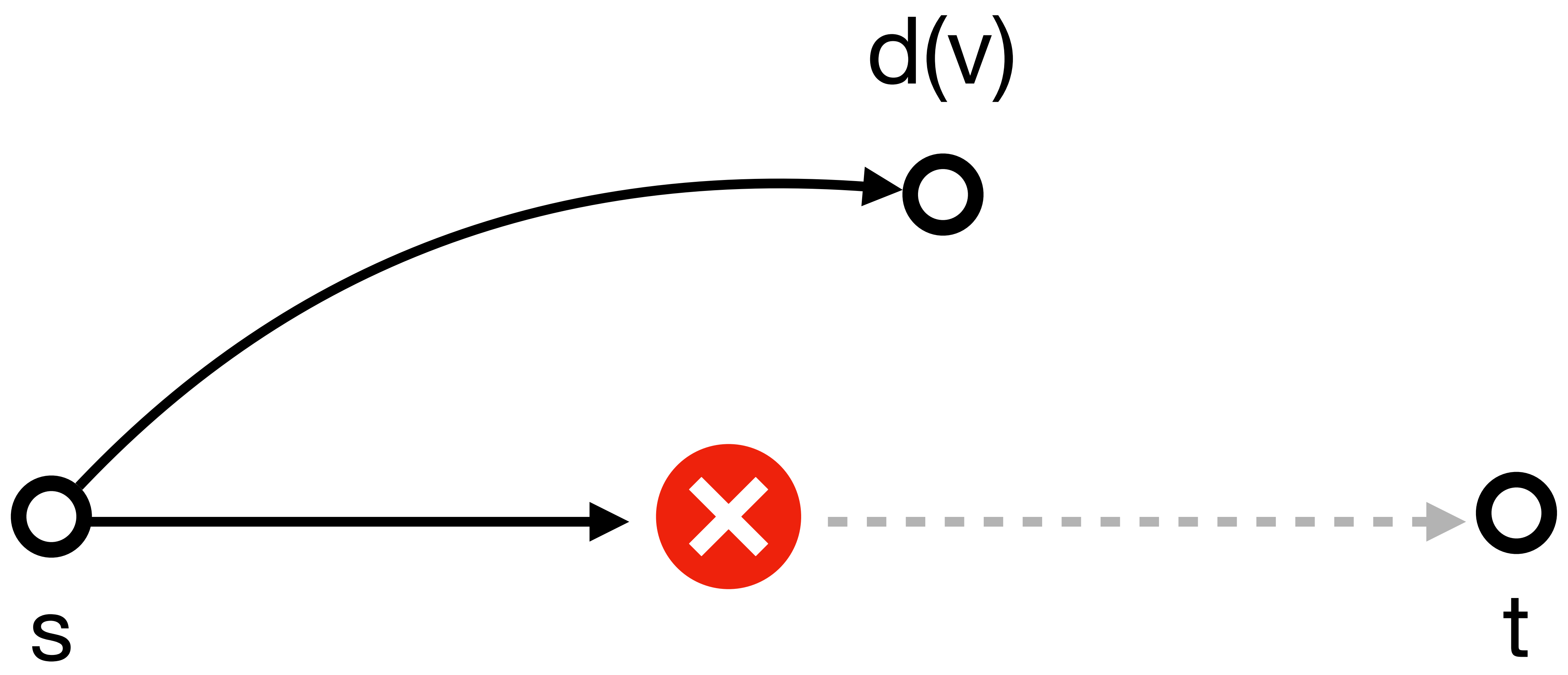
- Single-failure RP = best detour avoiding intervals
- **Compute the detours** for **all** possible failures

Progressive Dijkstra [Bern'10]

First idea:

Incre maintain all Dijkstra labels

1. Start with $G\backslash\pi$
2. add back π edge by edge
3. Update $d(v)$, but scan out-edges of v iff $d(v)$ has decreased by $1-\epsilon$

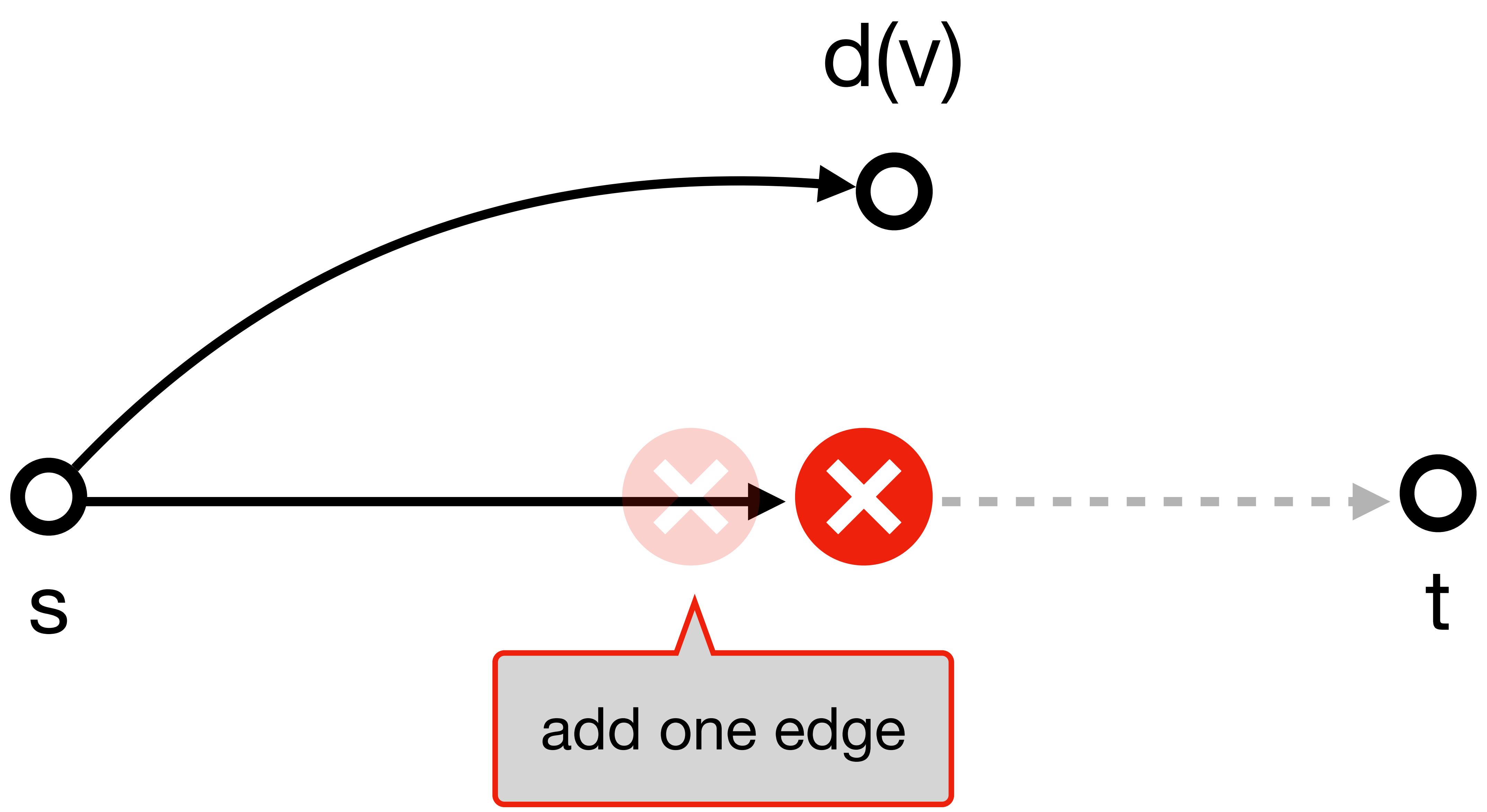


Progressive Dijkstra [Bern'10]

First idea:

Incre maintain all Dijkstra labels

1. Start with $G \backslash \pi$
2. add back π edge by edge
3. Update $d(v)$, but scan out-edges of v iff $d(v)$ has decreased by $1 - \epsilon$

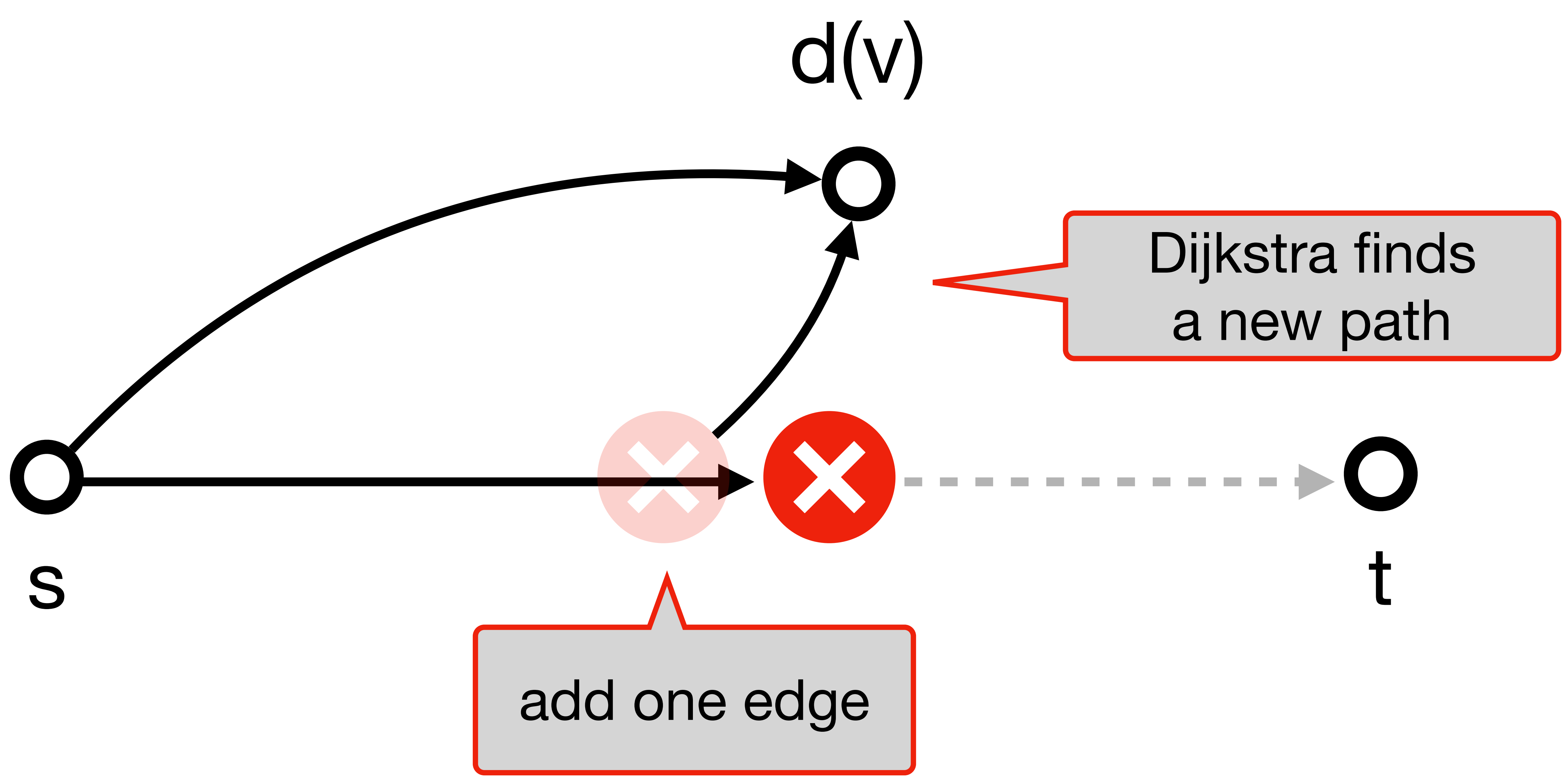


Progressive Dijkstra [Bern'10]

First idea:

Incre maintain all Dijkstra labels

1. Start with $G \backslash \pi$
2. add back π edge by edge
3. Update $d(v)$, but scan out-edges of v iff $d(v)$ has decreased by $1 - \epsilon$

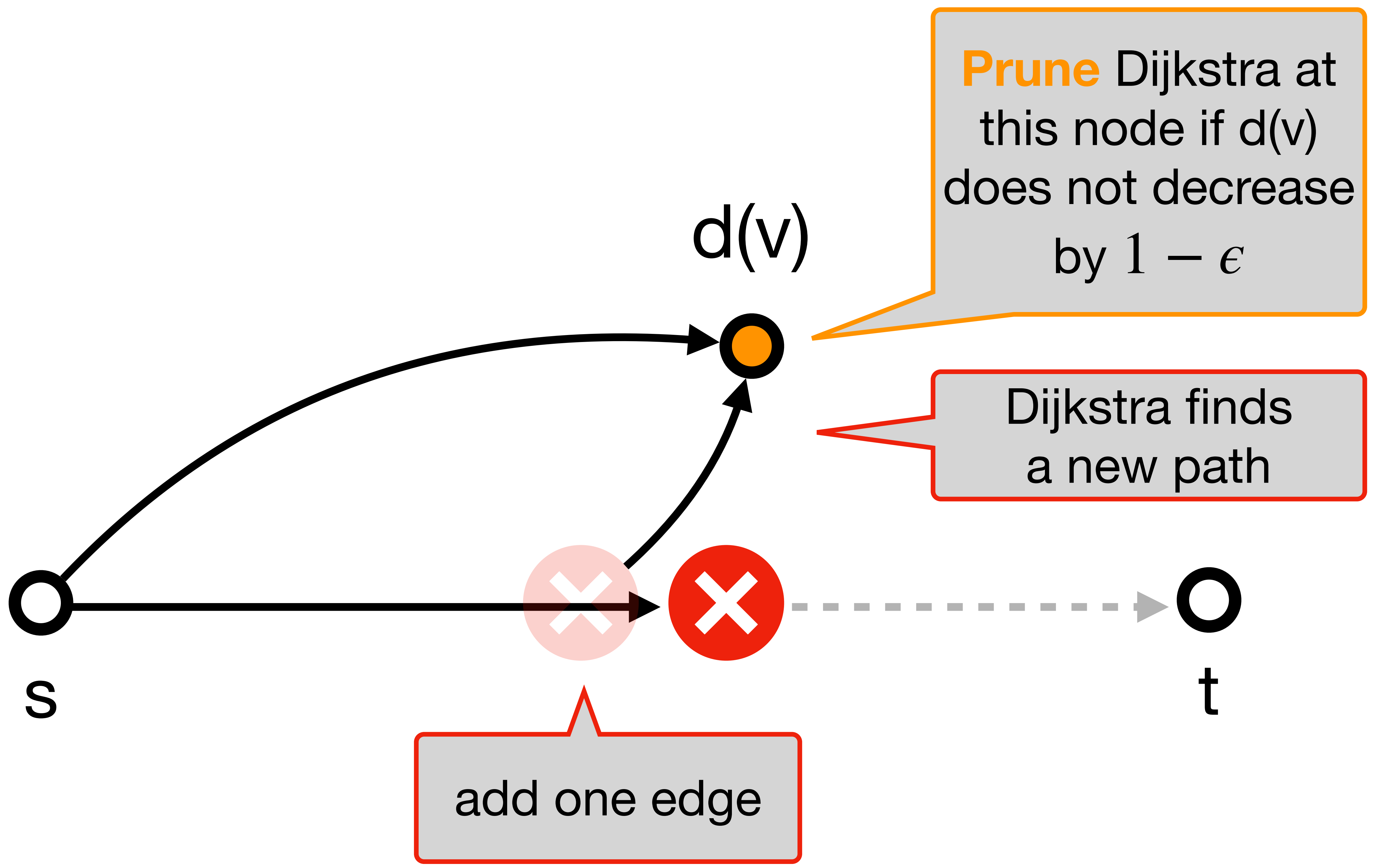


Progressive Dijkstra [Bern'10]

First idea:

Incre maintain all Dijkstra labels

1. Start with $G \backslash \pi$
2. add back π edge by edge
3. Update $d(v)$, but scan out-edges of v iff $d(v)$ has decreased by $1-\epsilon$



Progressive Dijkstra [Bern'10]

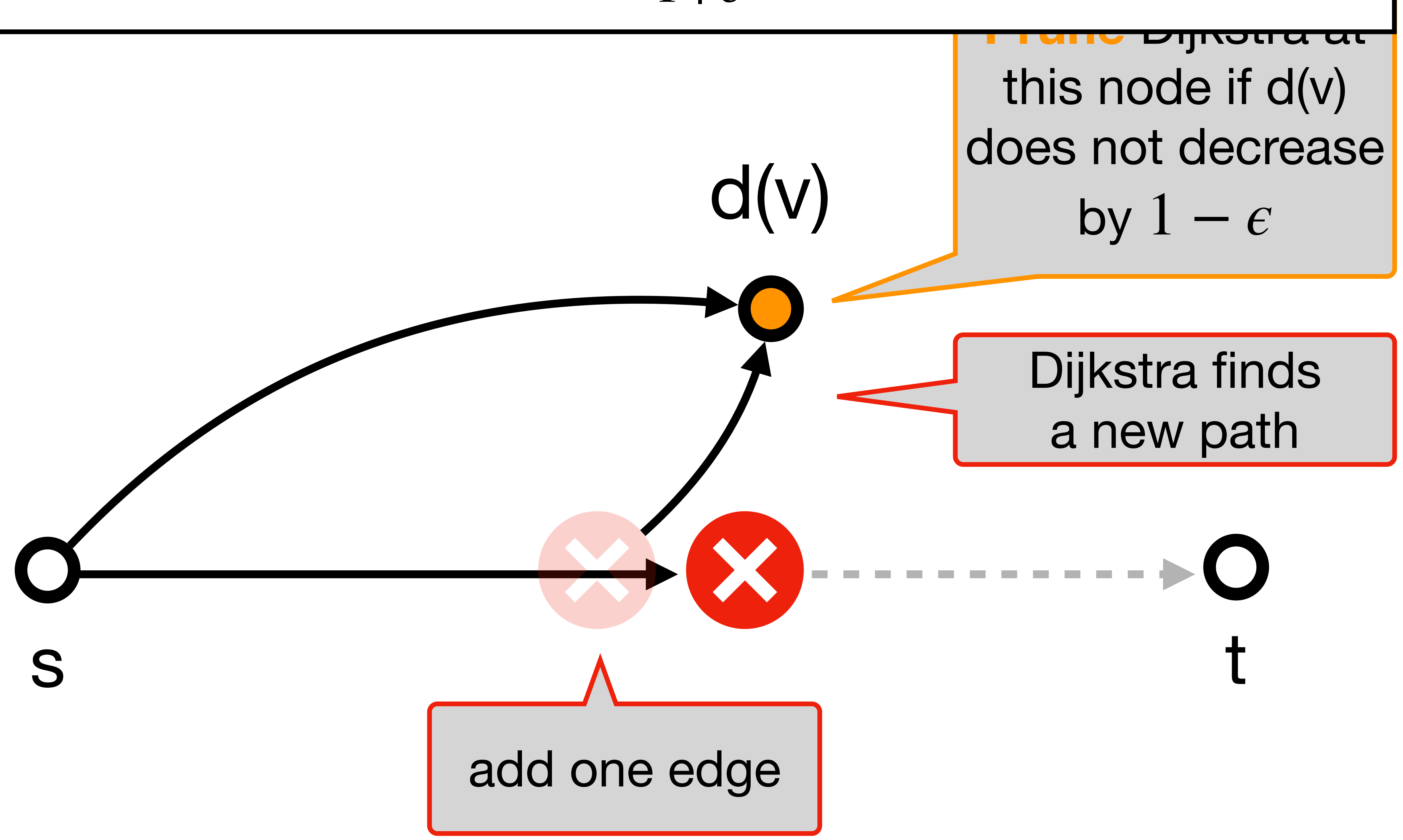
First idea:

Incre maintain all Dijkstra labels

1. Start with $G \backslash \pi$
2. add back π edge by edge
3. Update $d(v)$, but scan out-edges of v iff $d(v)$ has decreased by $1-\epsilon$

Runtime:

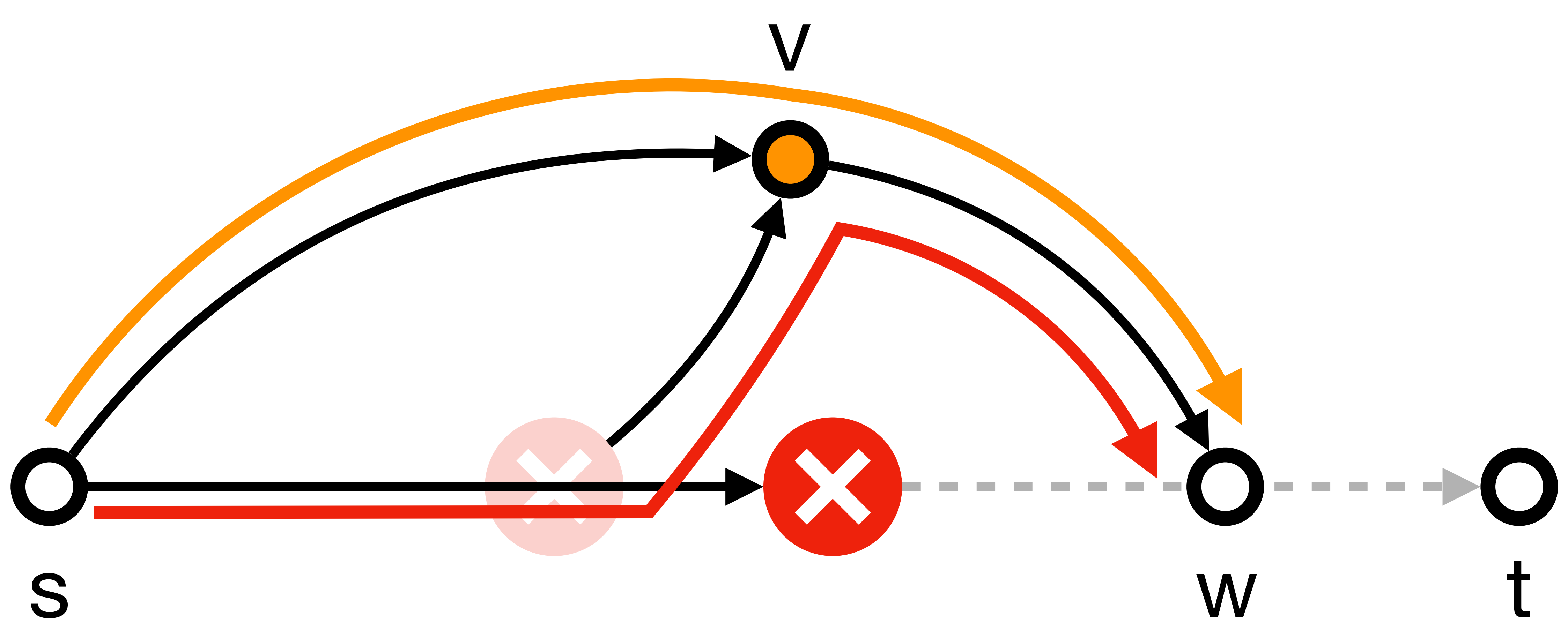
- Each out-neighbor scanned $\log_{1+\epsilon}(nW)$ times
- Total runtime = $n^2 \log_{1+\epsilon}(nW)$



Progressive Dijkstra [Bern'10]

Approximation error:

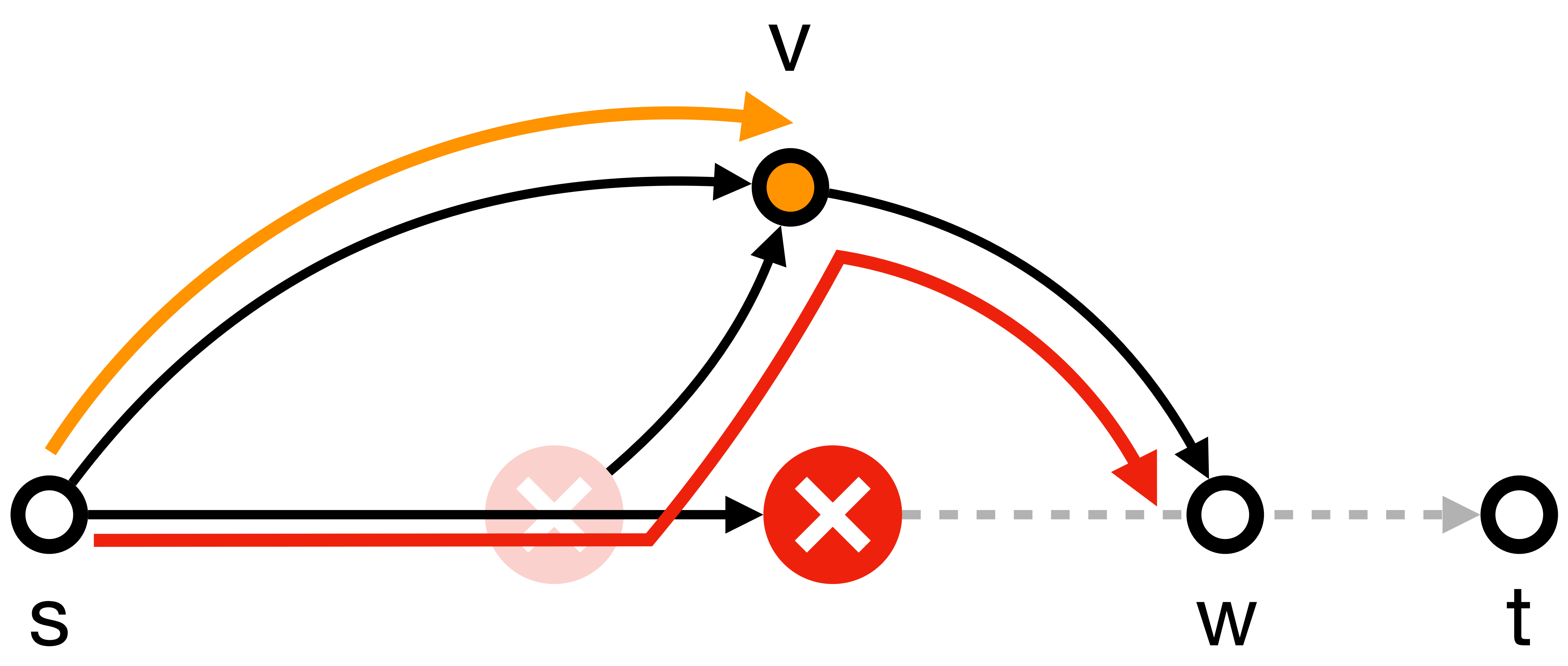
- If $d(v)$ doesn't decrease by $1-\epsilon$, then yellow $< (1+\epsilon)\times$ red



Progressive Dijkstra [Bern'10]

Approximation error:

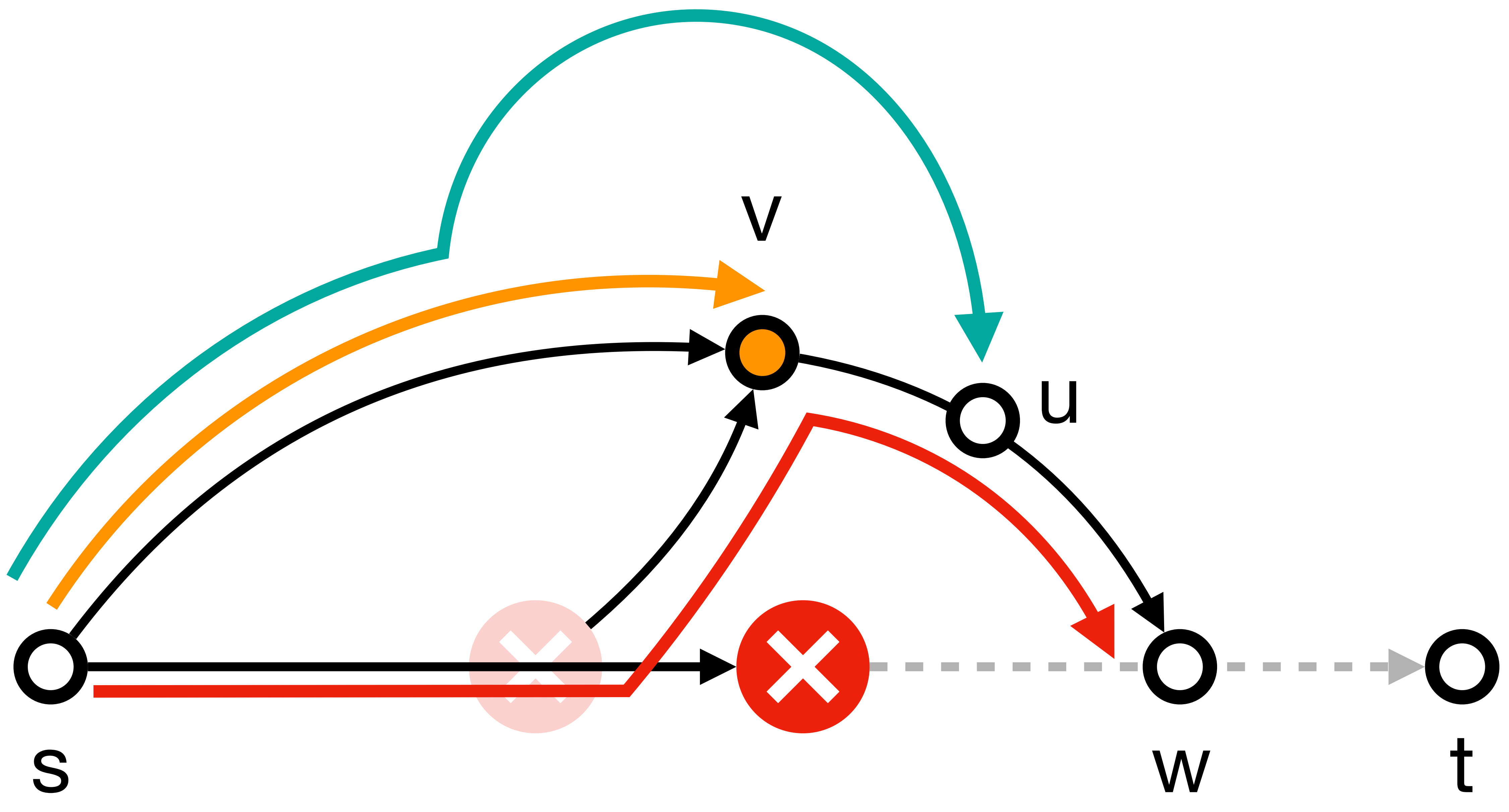
- If $d(v)$ doesn't decrease by $1 - \epsilon$, then yellow $< (1 + \epsilon) \times$ red
- However, previous iterations only know sv-path



Progressive Dijkstra [Bern'10]

Approximation error:

- If $d(v)$ doesn't decrease by $1 - \epsilon$, then **yellow** $< (1 + \epsilon) \times$ **red**
- However, previous iterations only know **sv-path**
- **vw-path** could be intercepted by even **earlier iterations**; that is, **blue** $< (1 + \epsilon) \times$ **yellow**

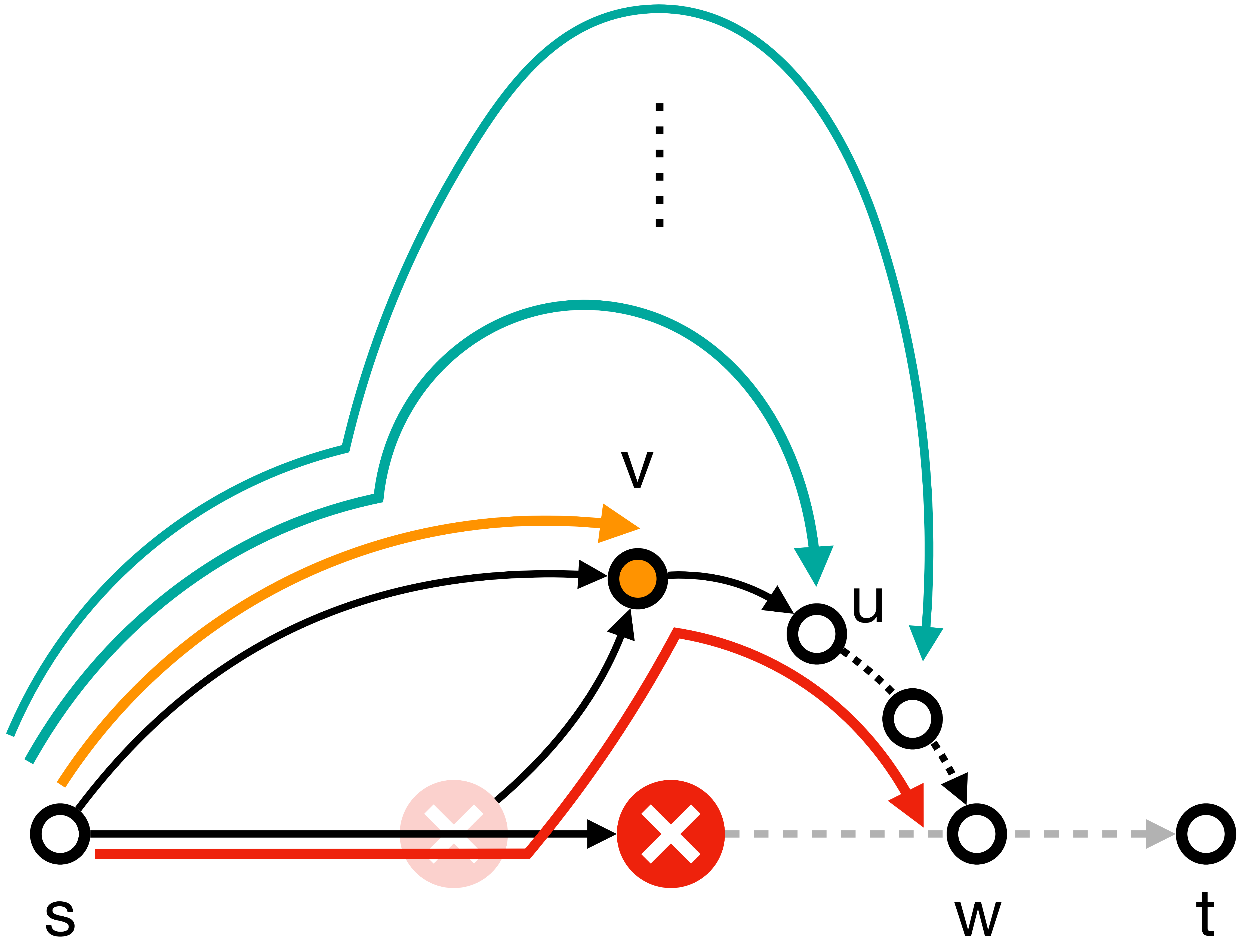


Progressive Dijkstra [Bern'10]

Approximation error:

- If $d(v)$ doesn't decrease by $1 - \epsilon$, then yellow $< (1 + \epsilon) \times$ red
- However, previous iterations only know **sv-path**
- **vw-path** could be intercepted by even **earlier iterations**; that is, **blue** $< (1 + \epsilon) \times$ **yellow**
- $(1 + \epsilon)$ factors could accumulate

i-th earlier **blue** $< (1 + \epsilon)^i \times$ **red**



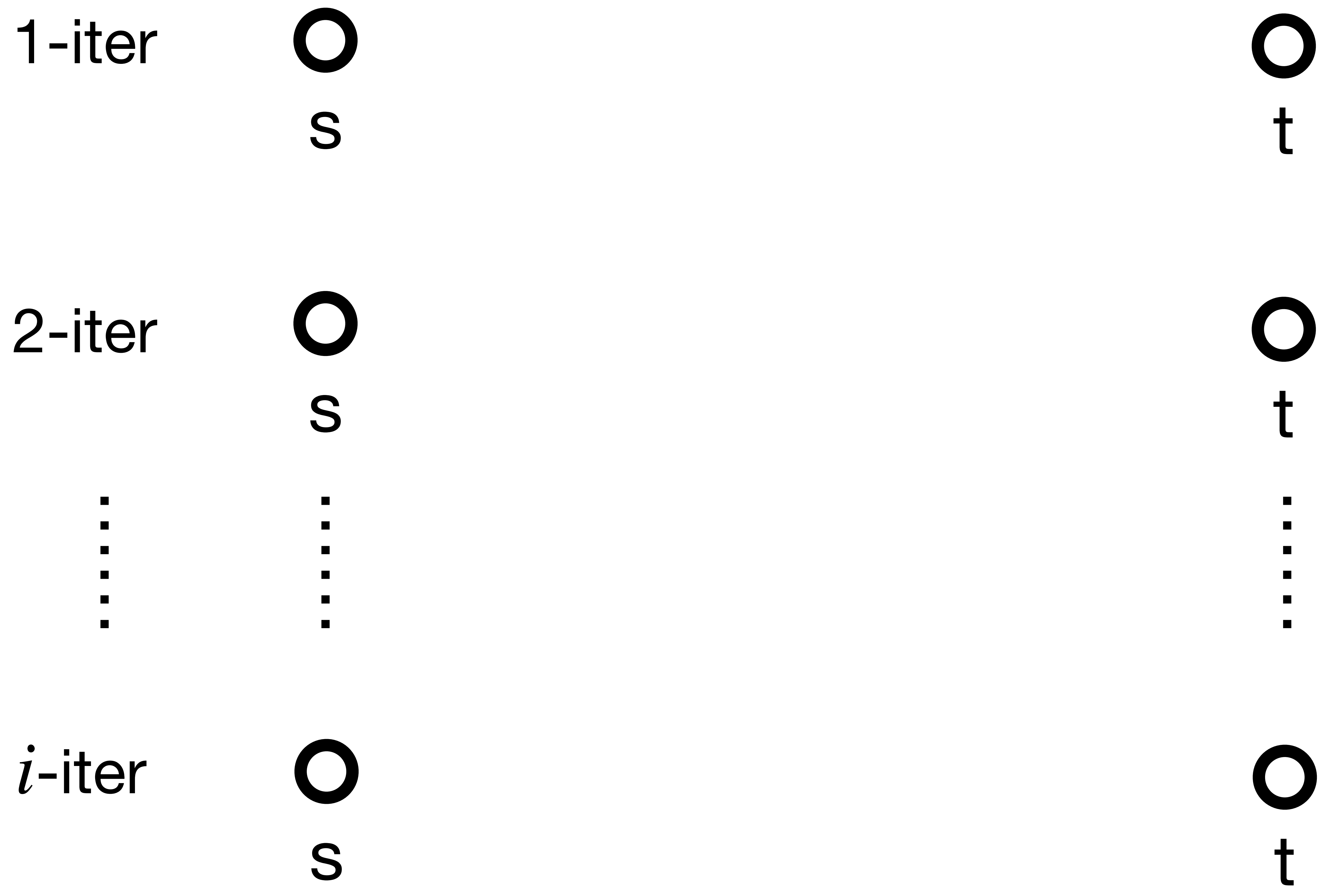
Progressive Dijkstra [Bern’10]

Main issue:

- $(1+\epsilon)$ factors could accumulate

Second idea:

- Run $\log n$ iterations of Dijkstra
- In the **i-th iter**ation, begin with graph $G \backslash \pi$, and **add $n/2^i$ edges** each time
- Update Dijkstra labels lazily



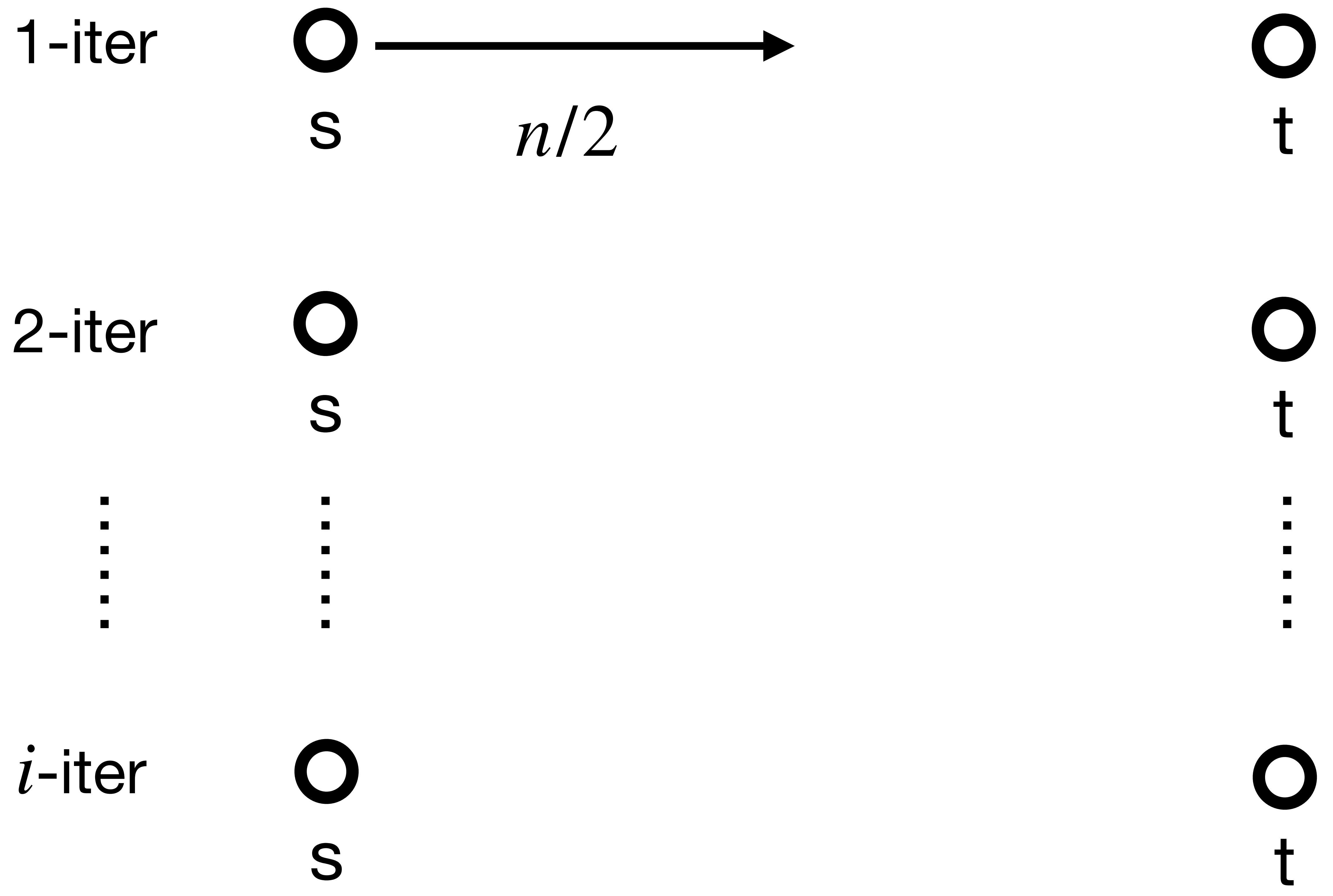
Progressive Dijkstra [Bern'10]

Main issue:

- $(1+\epsilon)$ factors could accumulate

Second idea:

- Run $\log n$ iterations of Dijkstra
- In the **i-th iter**ation, begin with graph $G \backslash \pi$, and **add $n/2^i$ edges** each time
- Update Dijkstra labels lazily



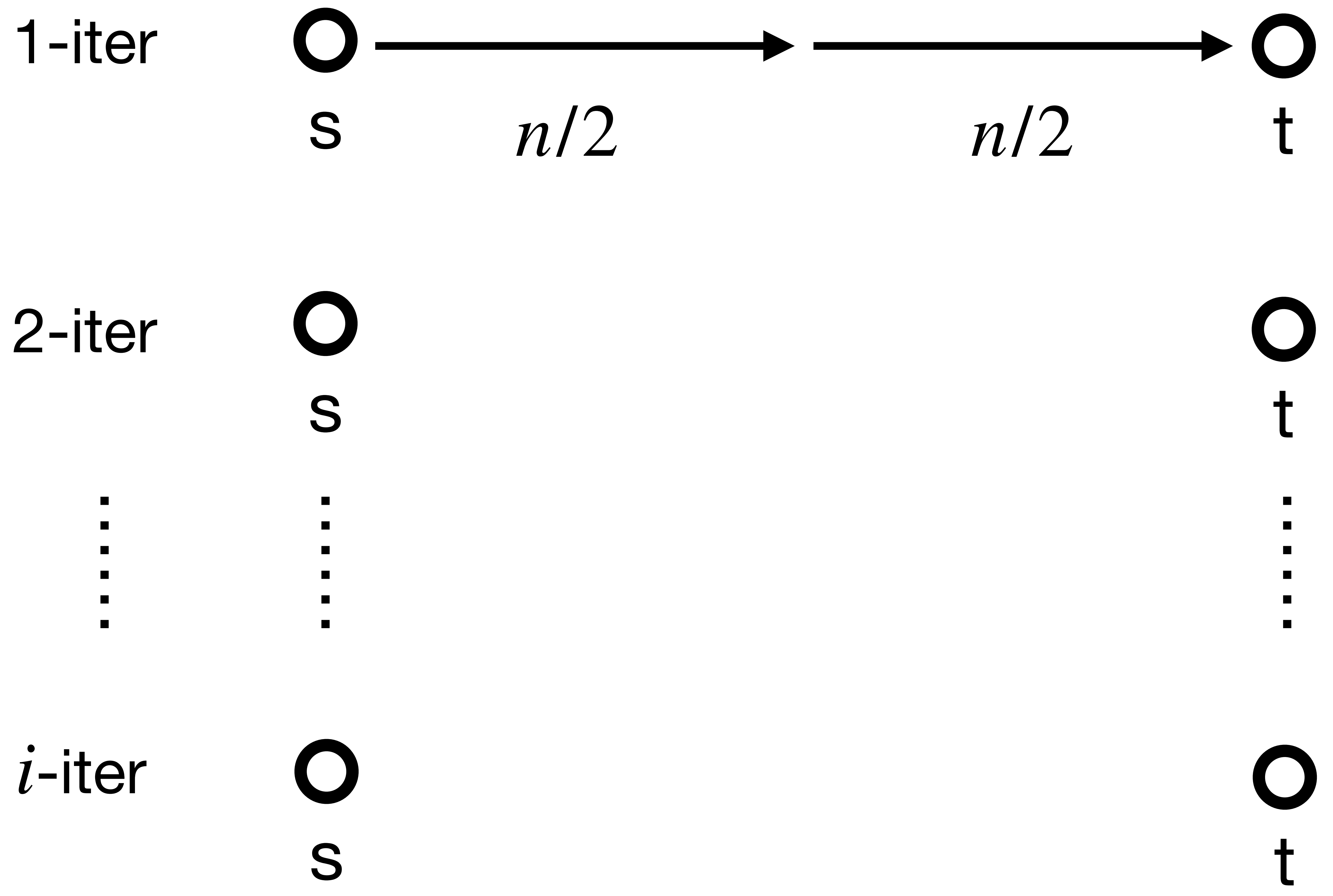
Progressive Dijkstra [Bern'10]

Main issue:

- $(1+\epsilon)$ factors could accumulate

Second idea:

- Run $\log n$ iterations of Dijkstra
- In the **i-th iter**ation, begin with graph $G \backslash \pi$, and **add $n/2^i$ edges** each time
- Update Dijkstra labels lazily



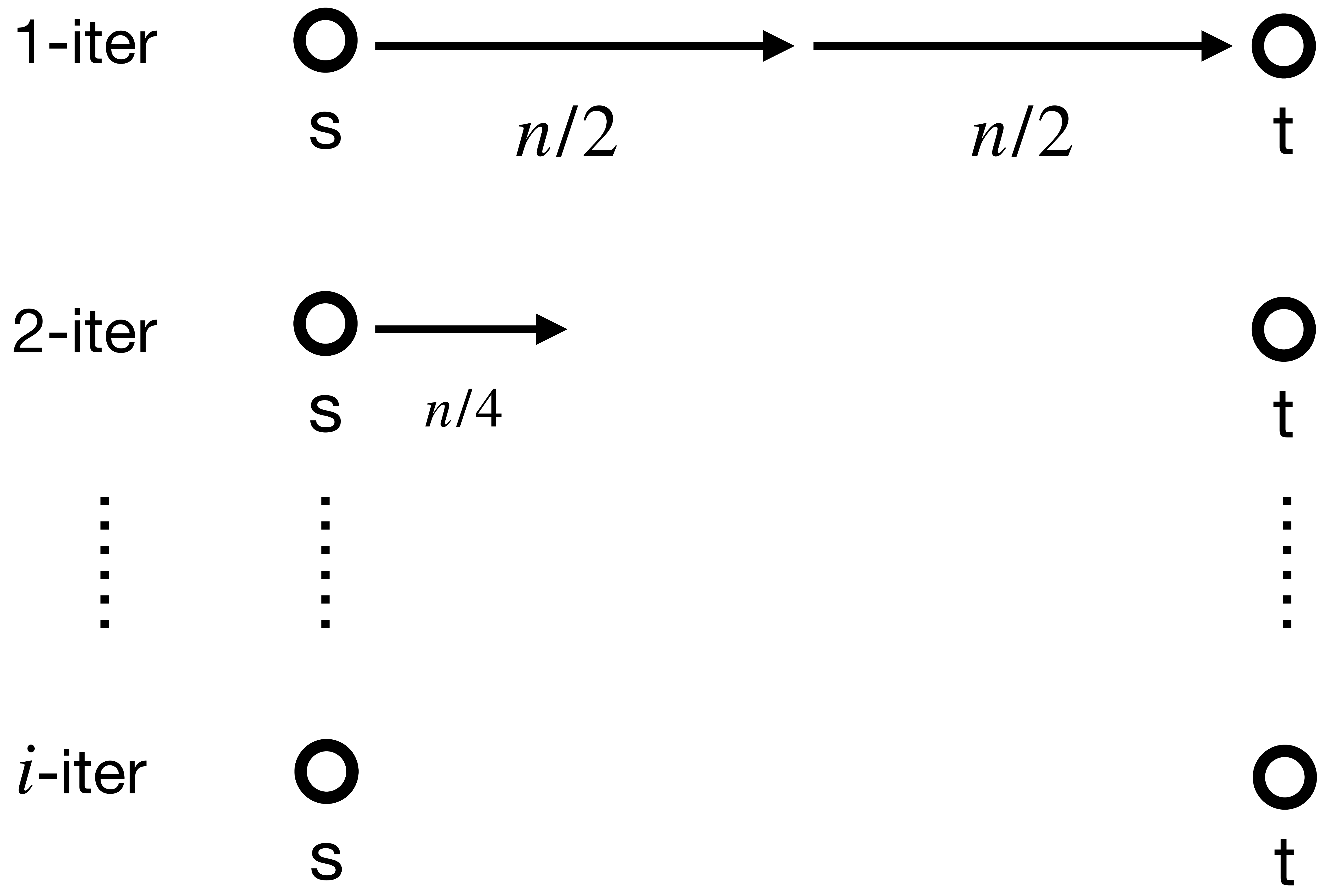
Progressive Dijkstra [Bern'10]

Main issue:

- $(1+\epsilon)$ factors could accumulate

Second idea:

- Run $\log n$ iterations of Dijkstra
- In the **i-th iter**ation, begin with graph $G \backslash \pi$, and **add $n/2^i$ edges** each time
- Update Dijkstra labels lazily



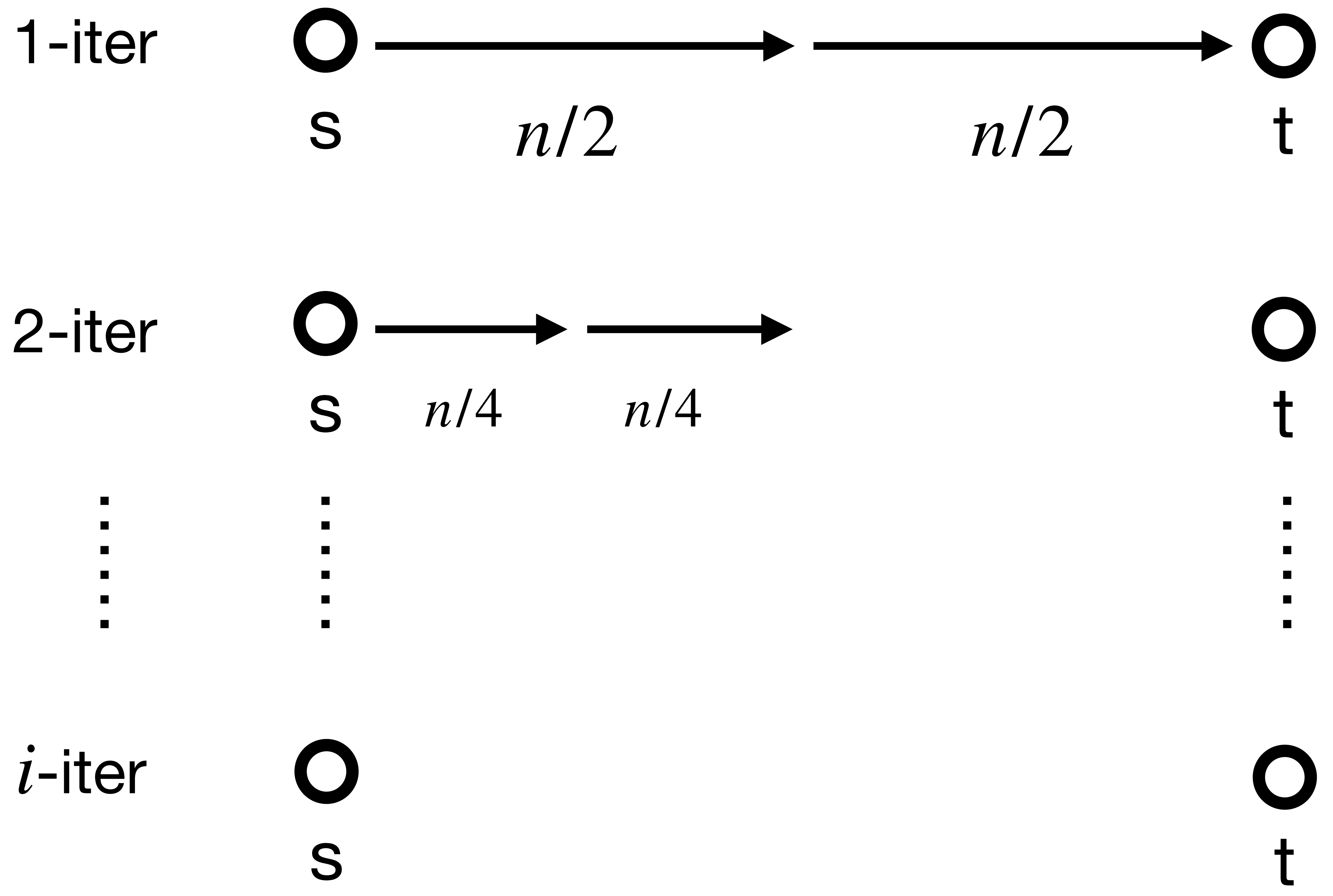
Progressive Dijkstra [Bern'10]

Main issue:

- $(1+\epsilon)$ factors could accumulate

Second idea:

- Run $\log n$ iterations of Dijkstra
- In the **i-th iter**ation, begin with graph $G\backslash\pi$, and **add $n/2^i$ edges** each time
- Update Dijkstra labels lazily



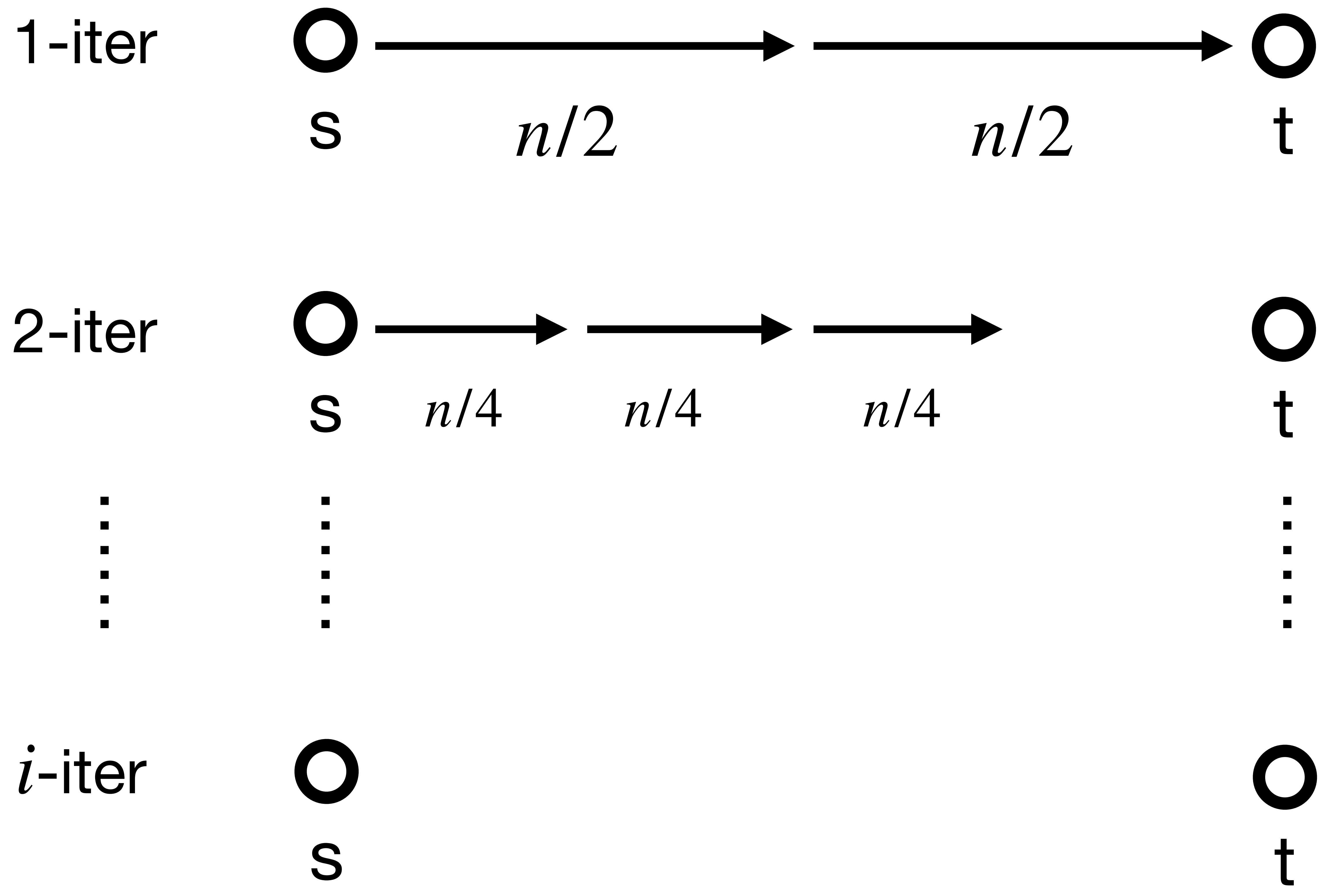
Progressive Dijkstra [Bern'10]

Main issue:

- $(1+\epsilon)$ factors could accumulate

Second idea:

- Run $\log n$ iterations of Dijkstra
- In the **i-th iter**ation, begin with graph $G \backslash \pi$, and **add $n/2^i$ edges** each time
- Update Dijkstra labels lazily



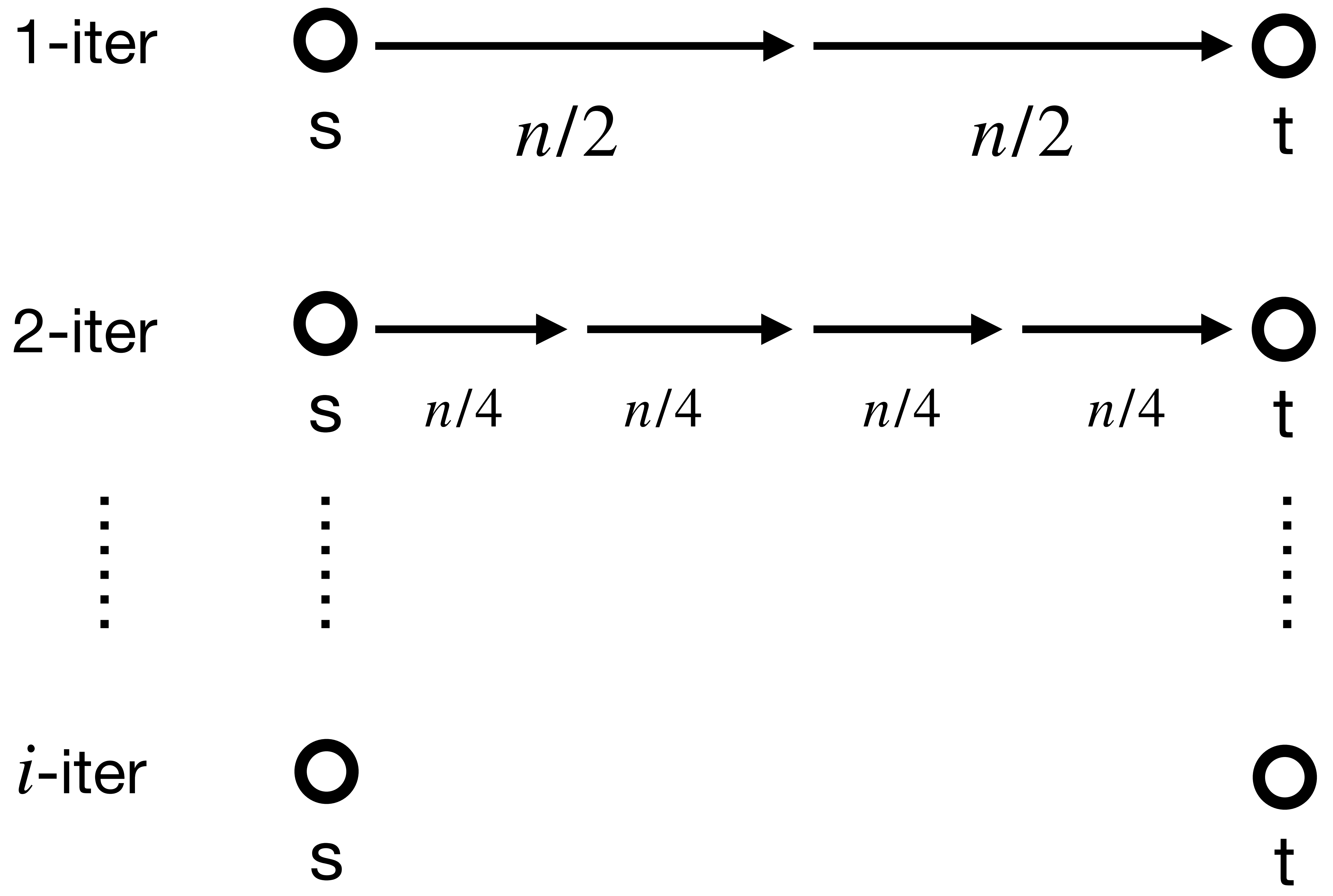
Progressive Dijkstra [Bern'10]

Main issue:

- $(1+\epsilon)$ factors could accumulate

Second idea:

- Run $\log n$ iterations of Dijkstra
- In the **i-th iter**ation, begin with graph $G \backslash \pi$, and **add $n/2^i$ edges** each time
- Update Dijkstra labels lazily



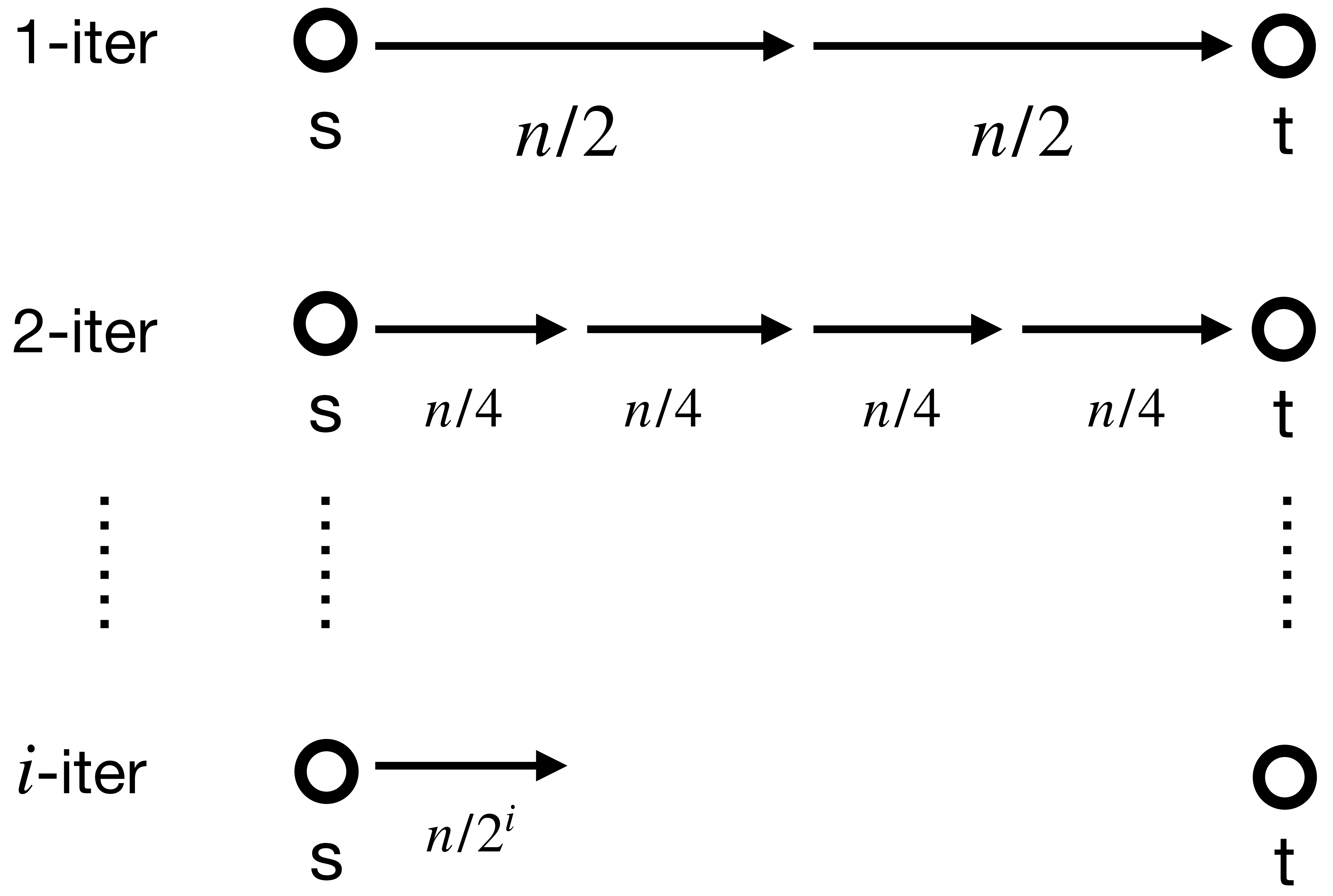
Progressive Dijkstra [Bern'10]

Main issue:

- $(1+\epsilon)$ factors could accumulate

Second idea:

- Run $\log n$ iterations of Dijkstra
- In the **i-th iter**ation, begin with graph $G \backslash \pi$, and **add $n/2^i$ edges** each time
- Update Dijkstra labels lazily



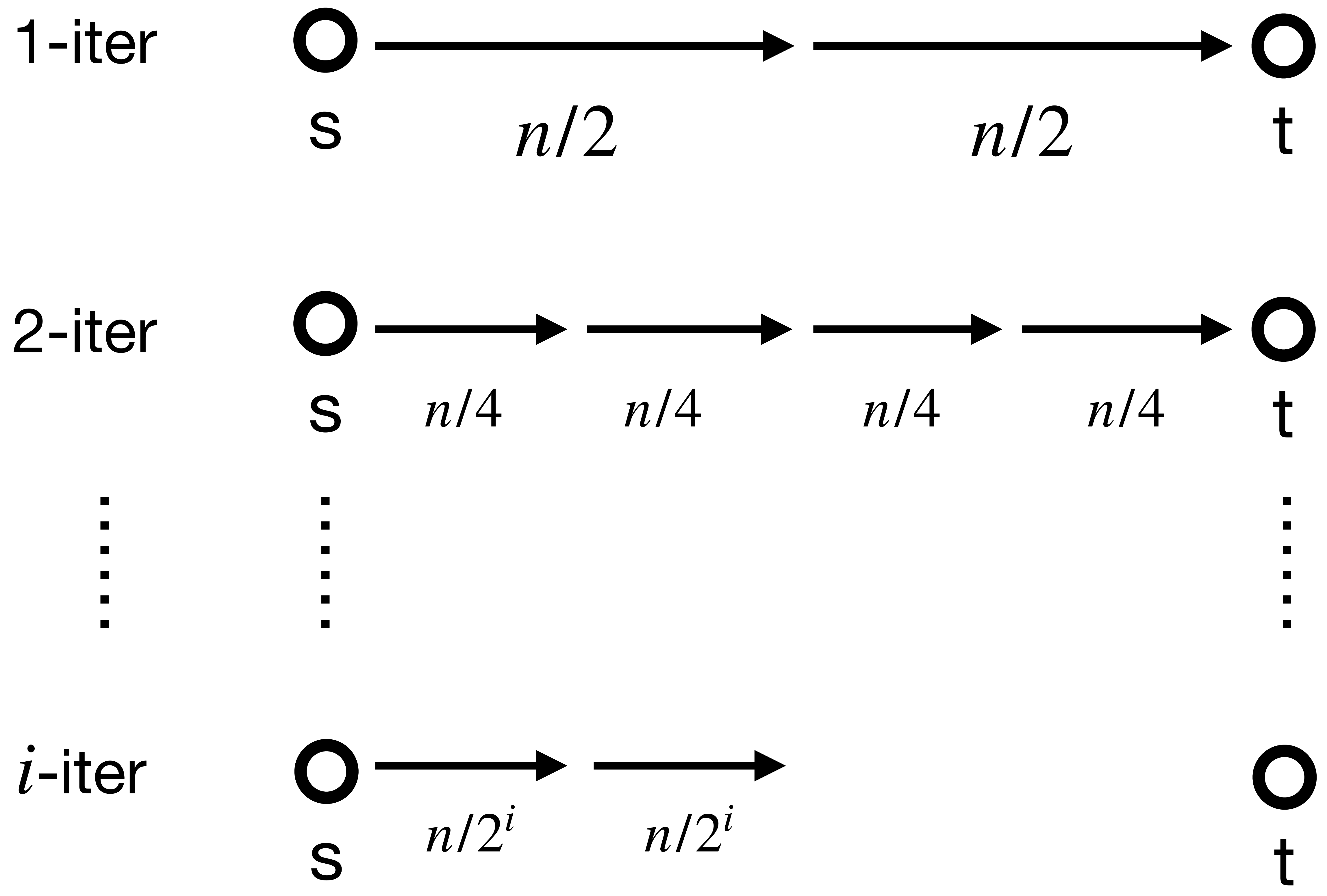
Progressive Dijkstra [Bern'10]

Main issue:

- $(1+\epsilon)$ factors could accumulate

Second idea:

- Run $\log n$ iterations of Dijkstra
- In the **i-th iter**ation, begin with graph $G \backslash \pi$, and **add $n/2^i$ edges** each time
- Update Dijkstra labels lazily



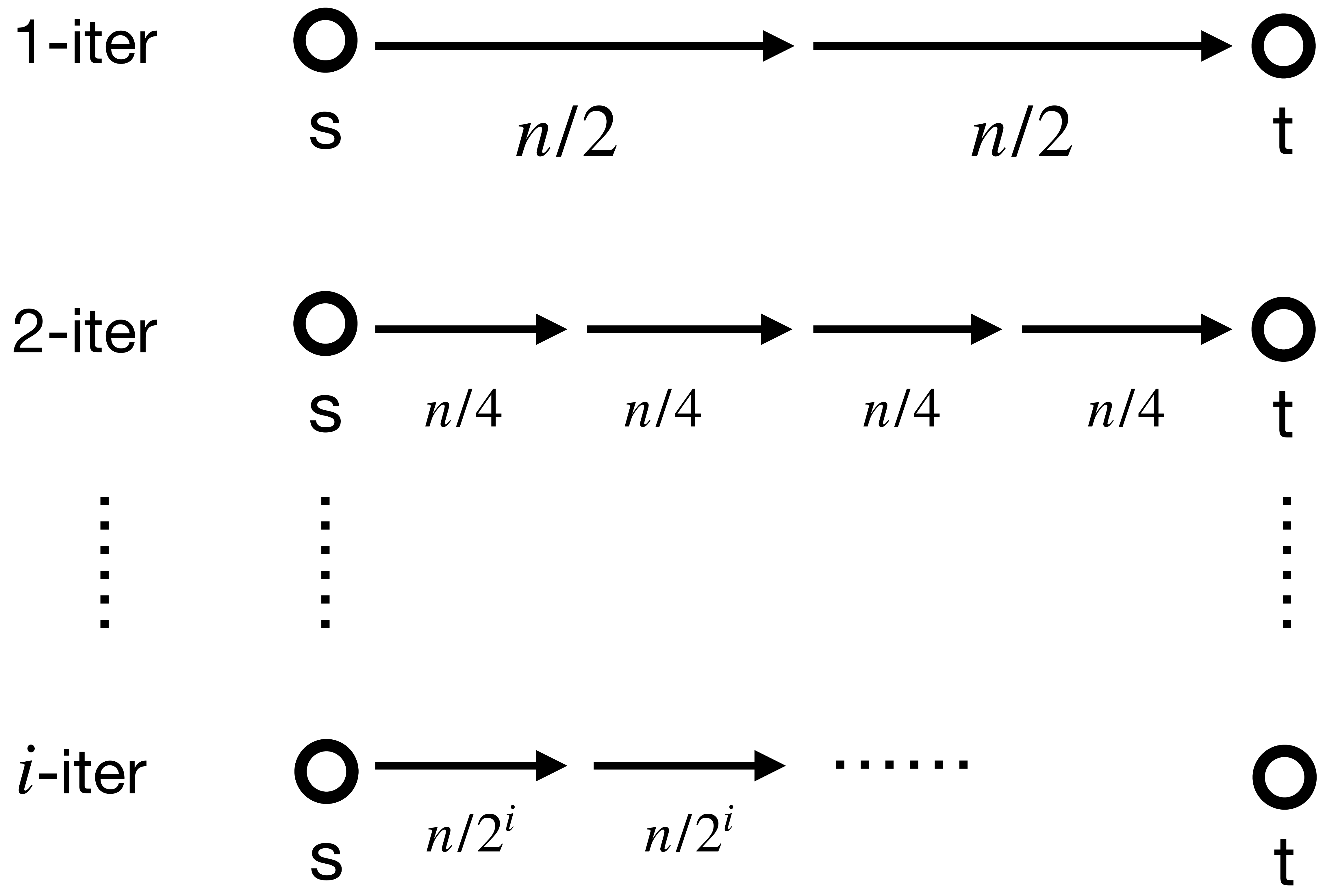
Progressive Dijkstra [Bern'10]

Main issue:

- $(1+\epsilon)$ factors could accumulate

Second idea:

- Run $\log n$ iterations of Dijkstra
- In the **i-th iter**ation, begin with graph $G\backslash\pi$, and **add $n/2^i$ edges** each time
- Update Dijkstra labels lazily



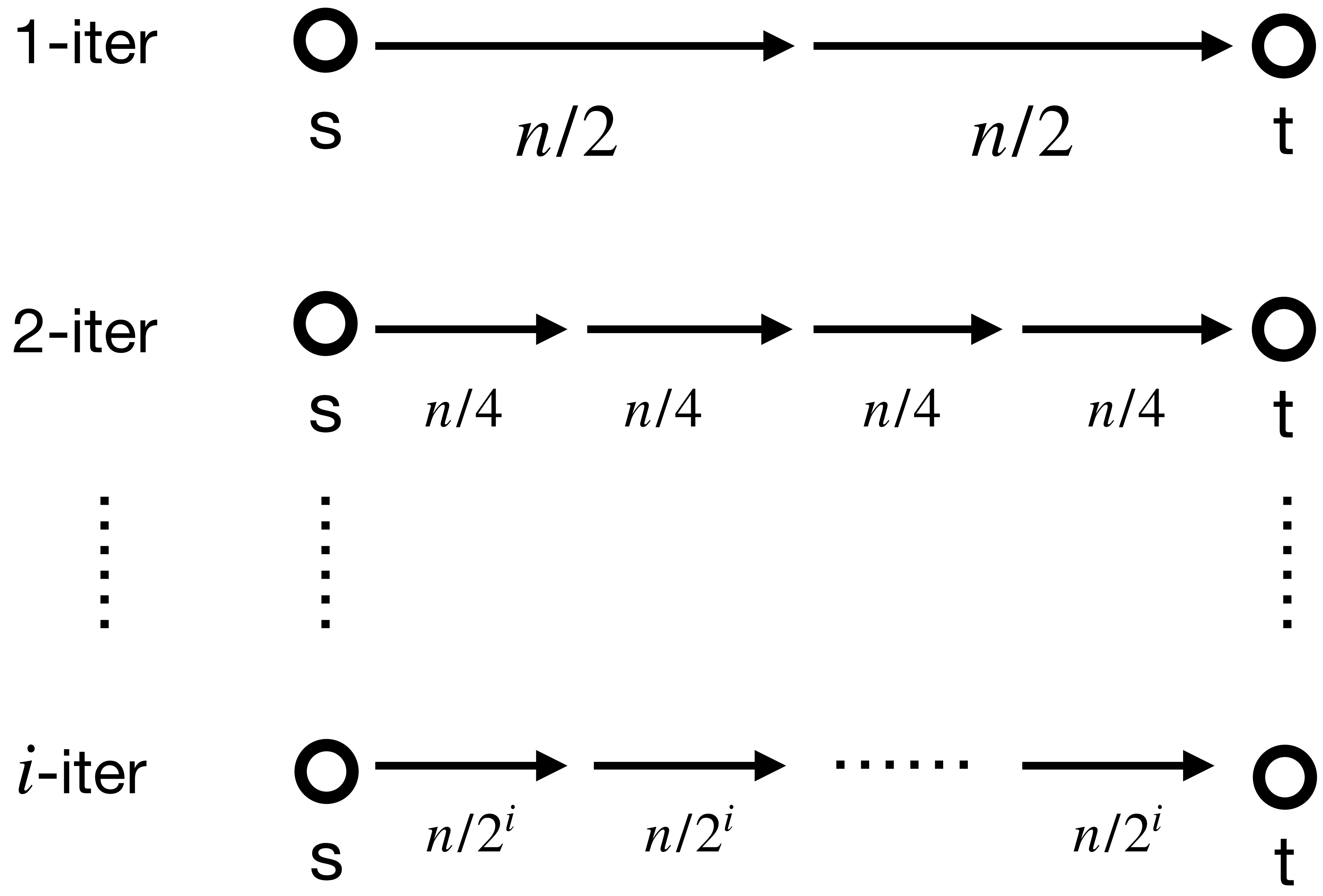
Progressive Dijkstra [Bern'10]

Main issue:

- $(1+\epsilon)$ factors could accumulate

Second idea:

- Run $\log n$ iterations of Dijkstra
- In the **i-th iter**ation, begin with graph $G \backslash \pi$, and **add $n/2^i$ edges** each time
- Update Dijkstra labels lazily



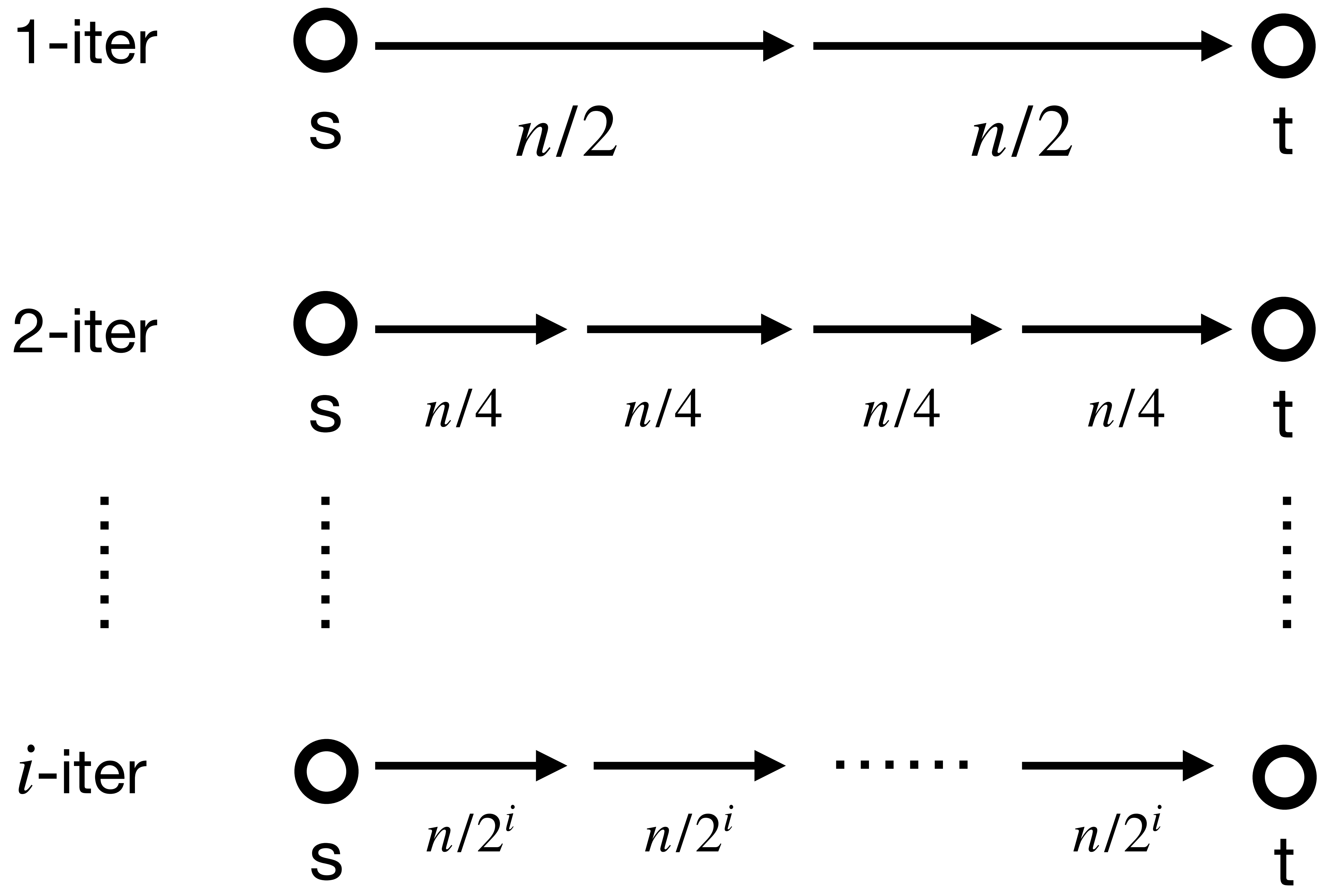
Progressive Dijkstra [Bern'10]

<u>Main issue:</u>

- $(1+\epsilon)$ factors could accumulate

<u>Second idea:</u>

- Run $\log n$ iterations of Dijkstra
- In the **i-th iter**ation, begin with graph $G \backslash \pi$, and **add $n/2^i$ edges** each time
- Update Dijkstra labels lazily

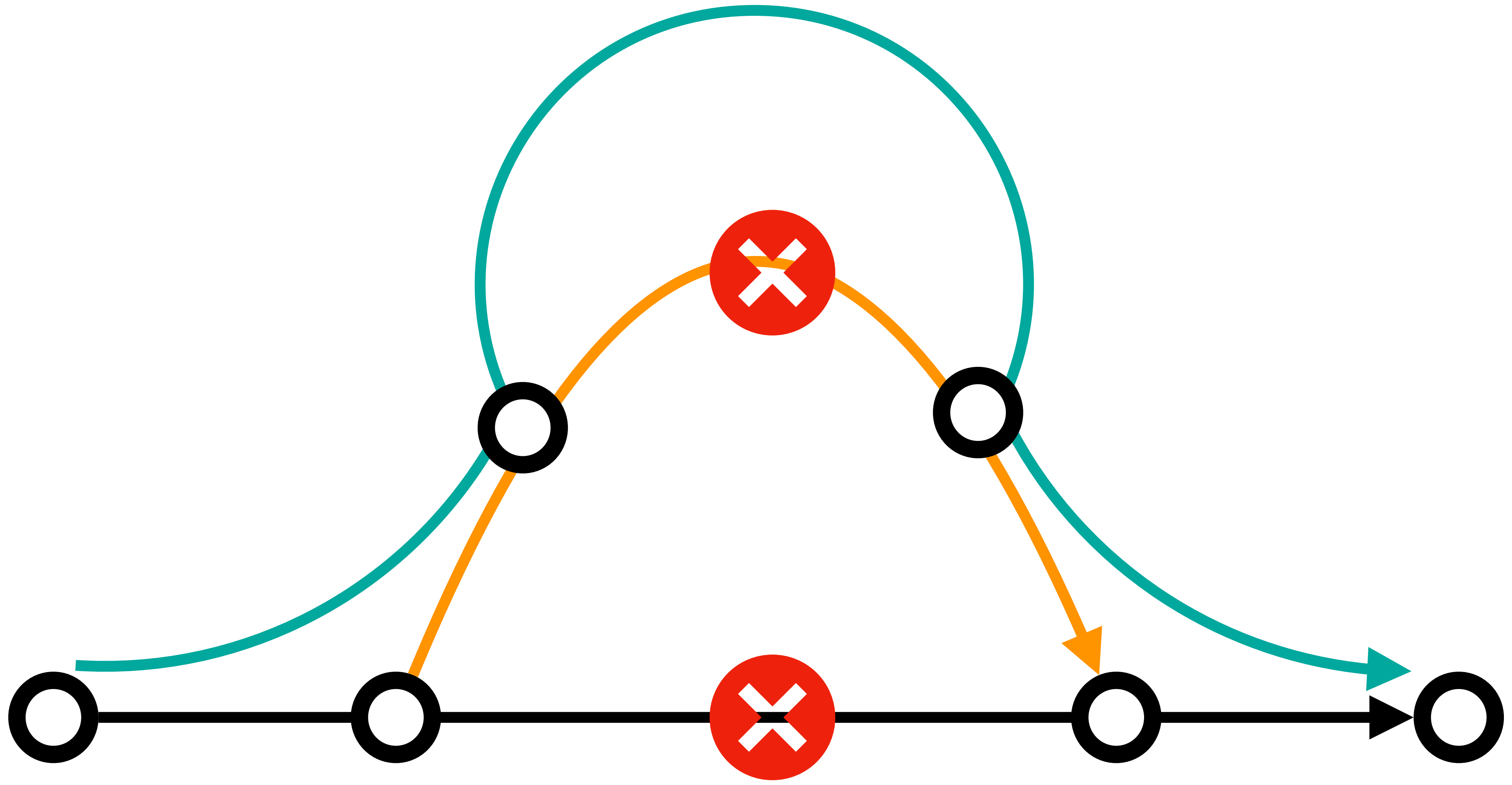


$(1+\epsilon)$ factors **accumulate $\log n$ times** [Bern'10]

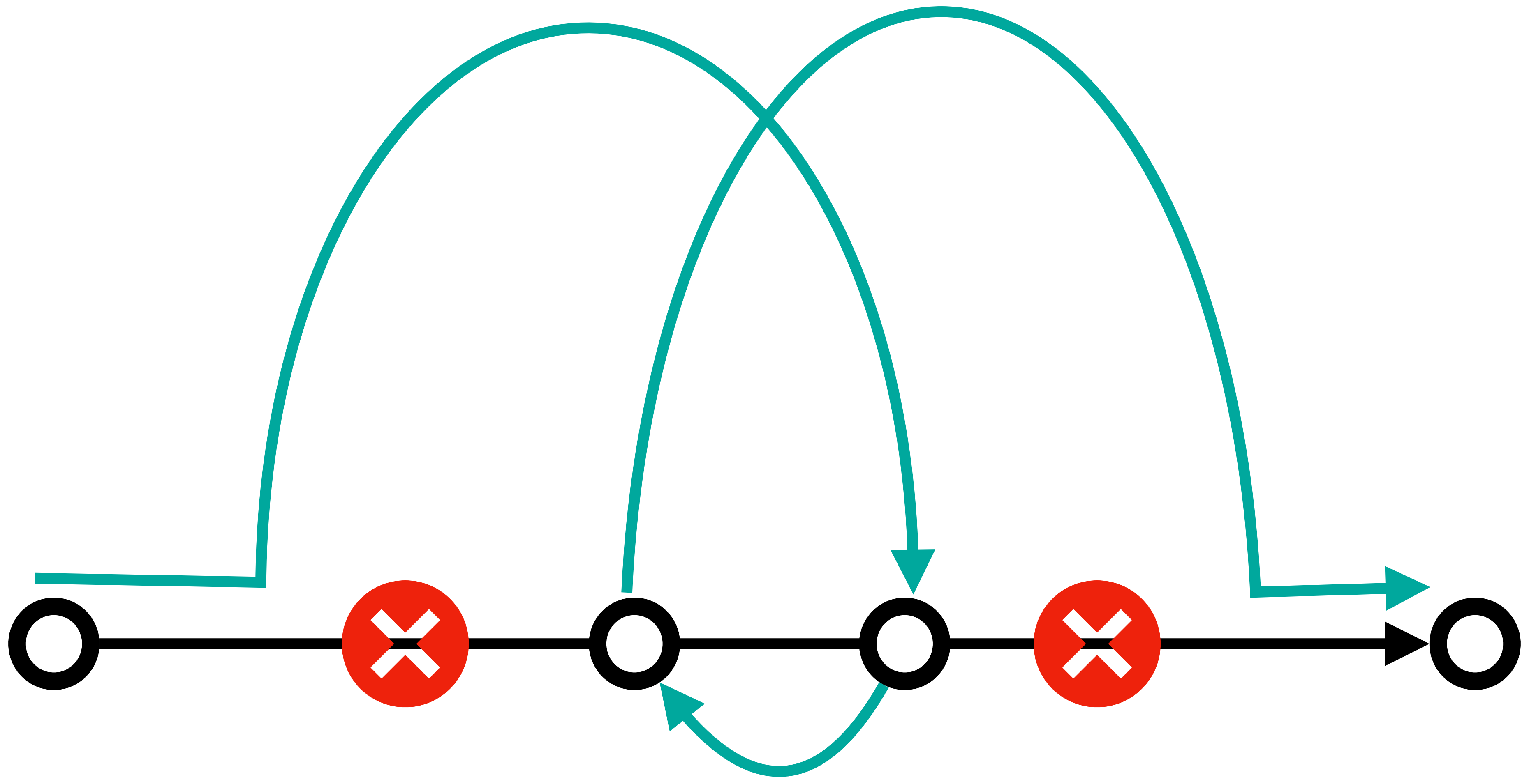
Dual-Failure Approx-RP

Two main cases

One failure on st-path

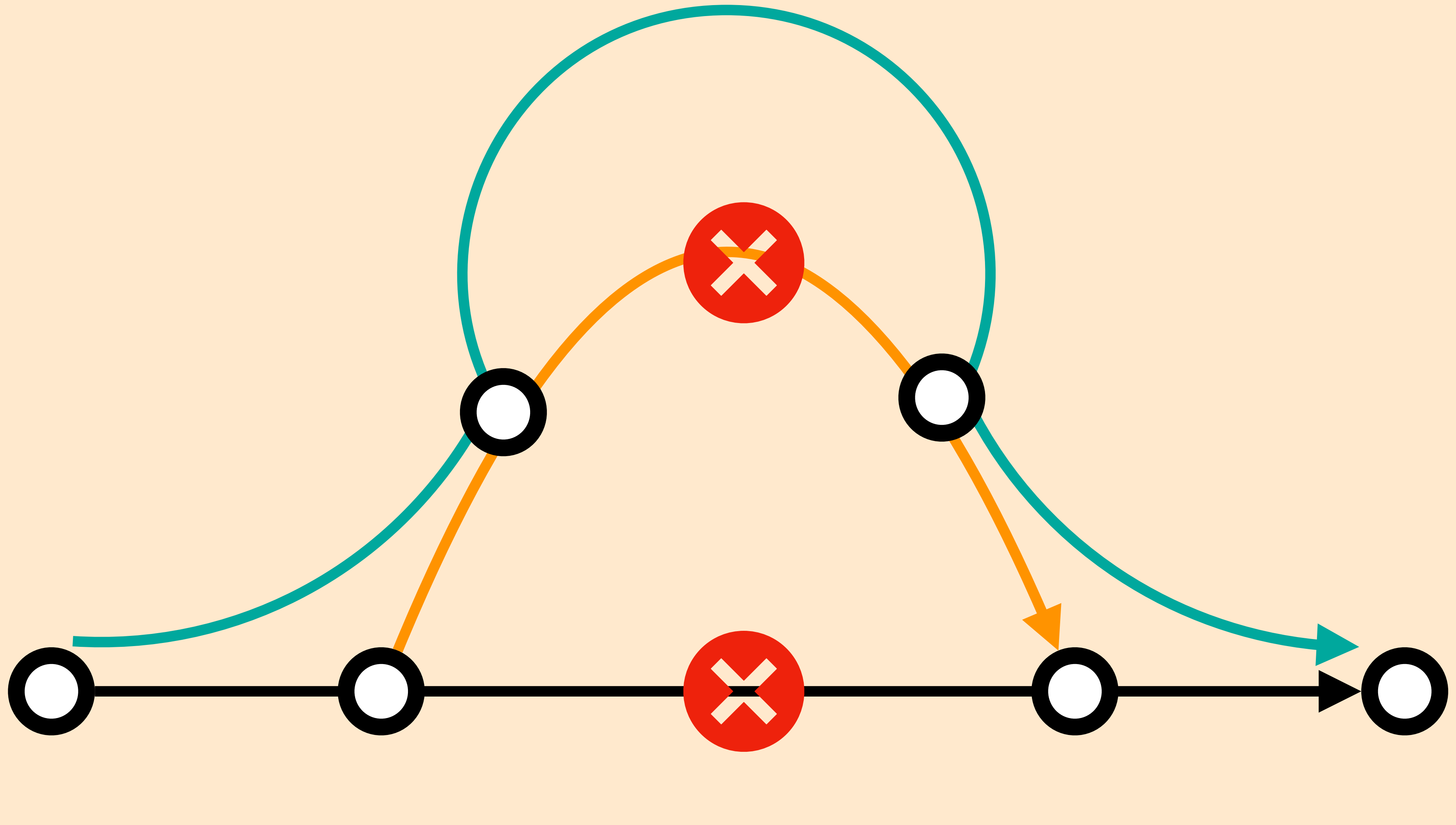


Both failures on st-path

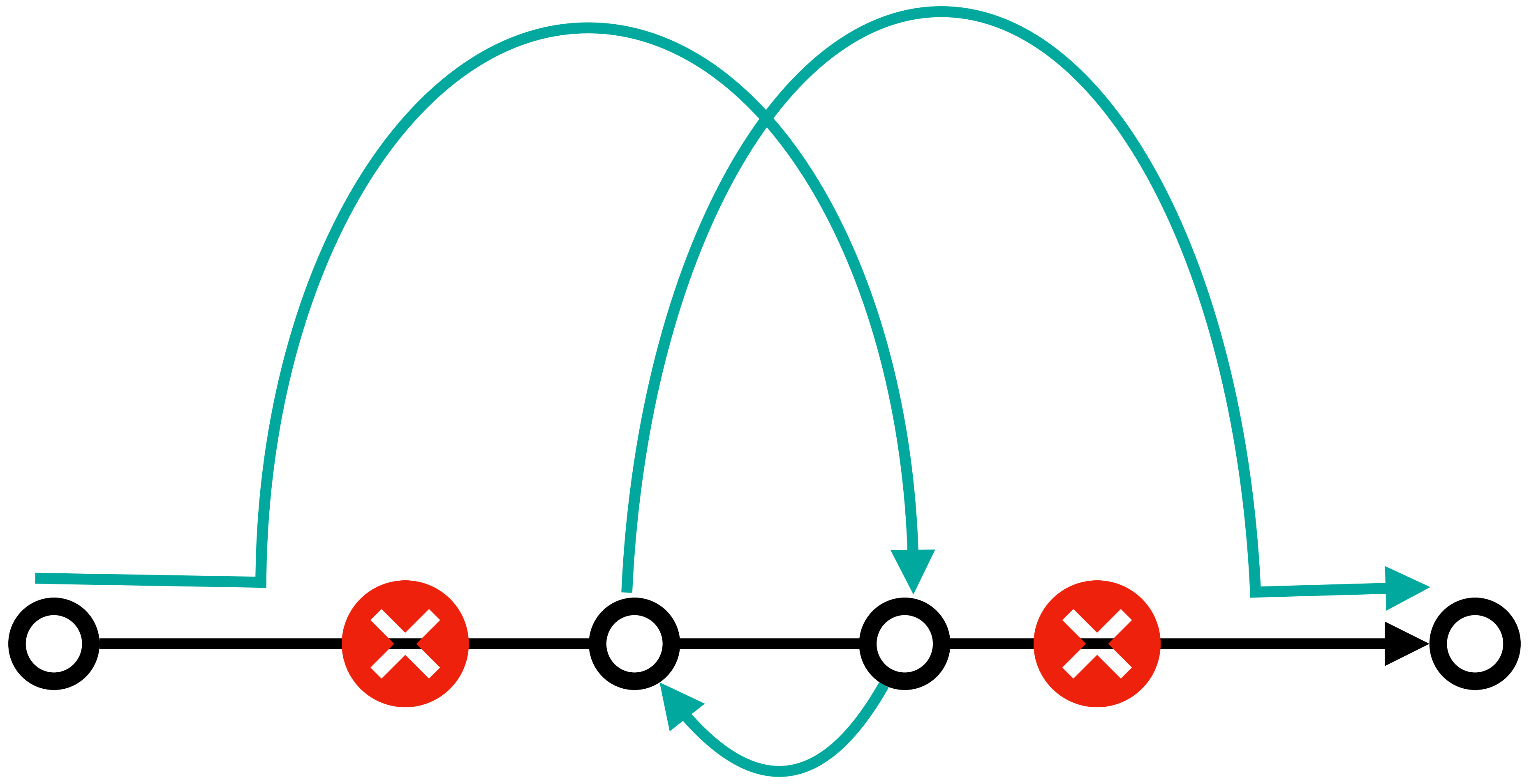


Two main cases

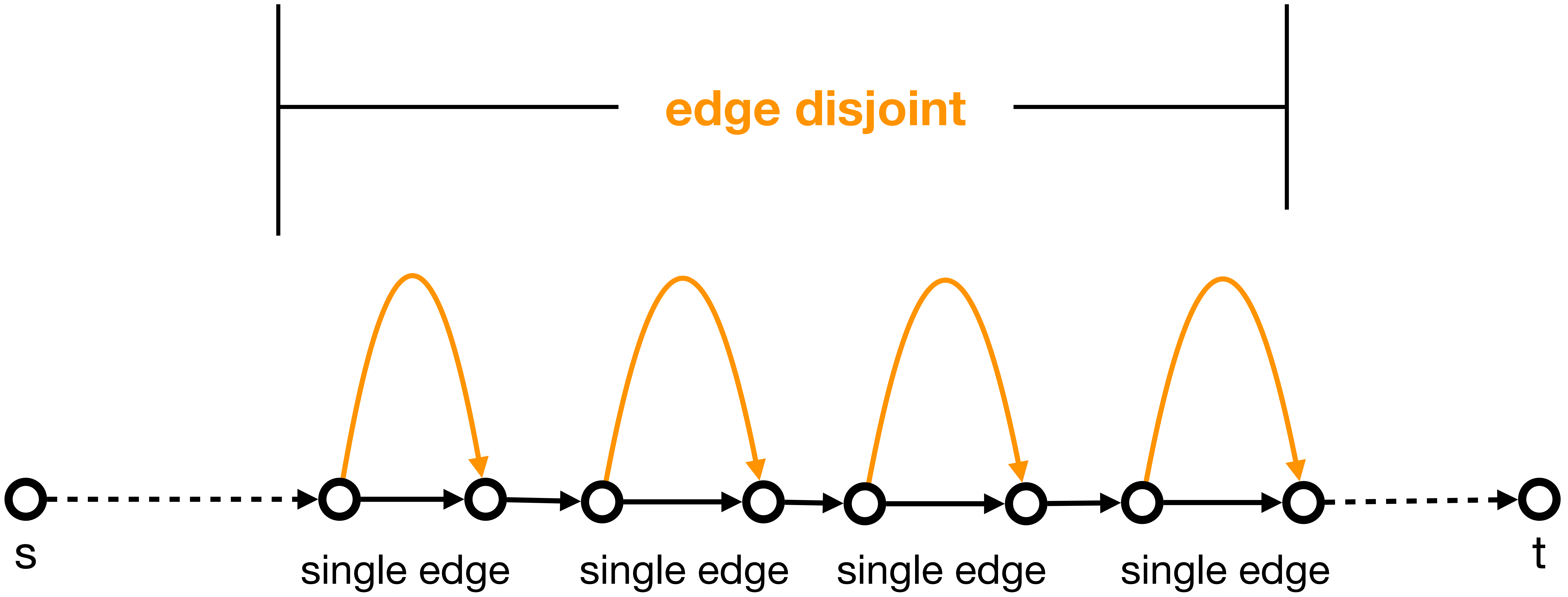
One failure on st-path



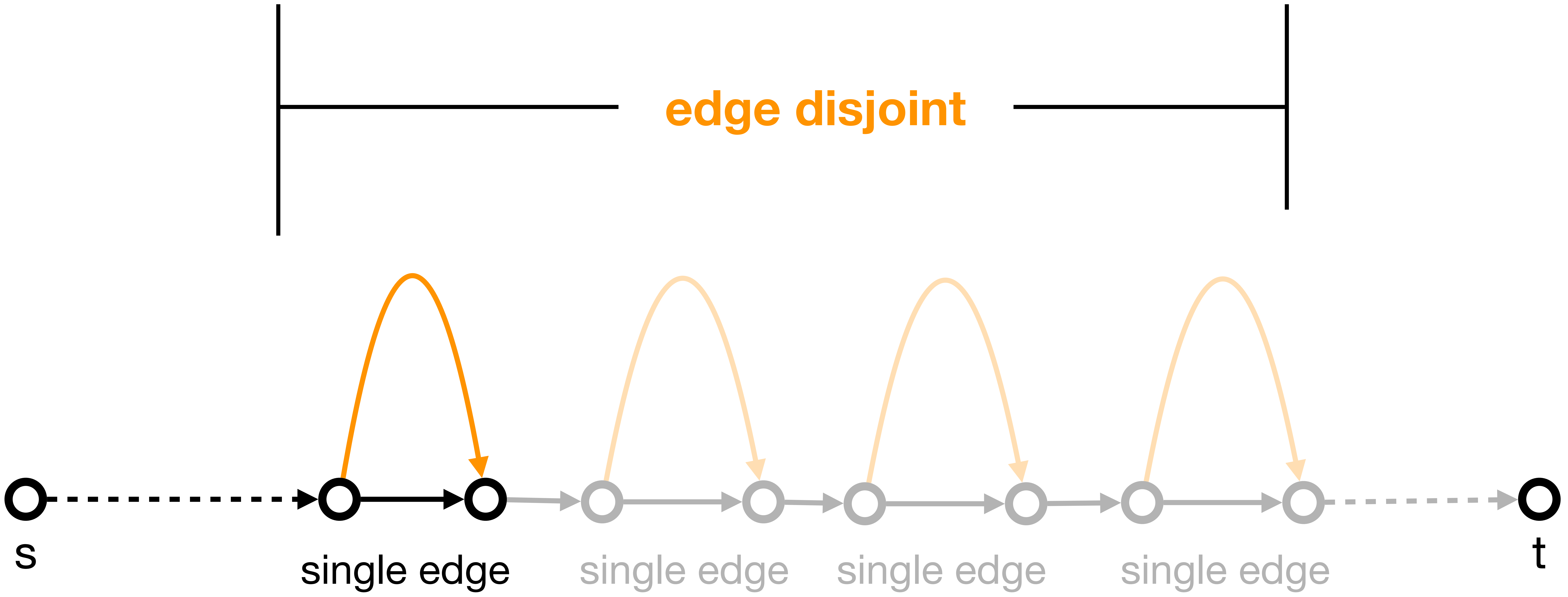
Both failures on st-path



Wishful thinking

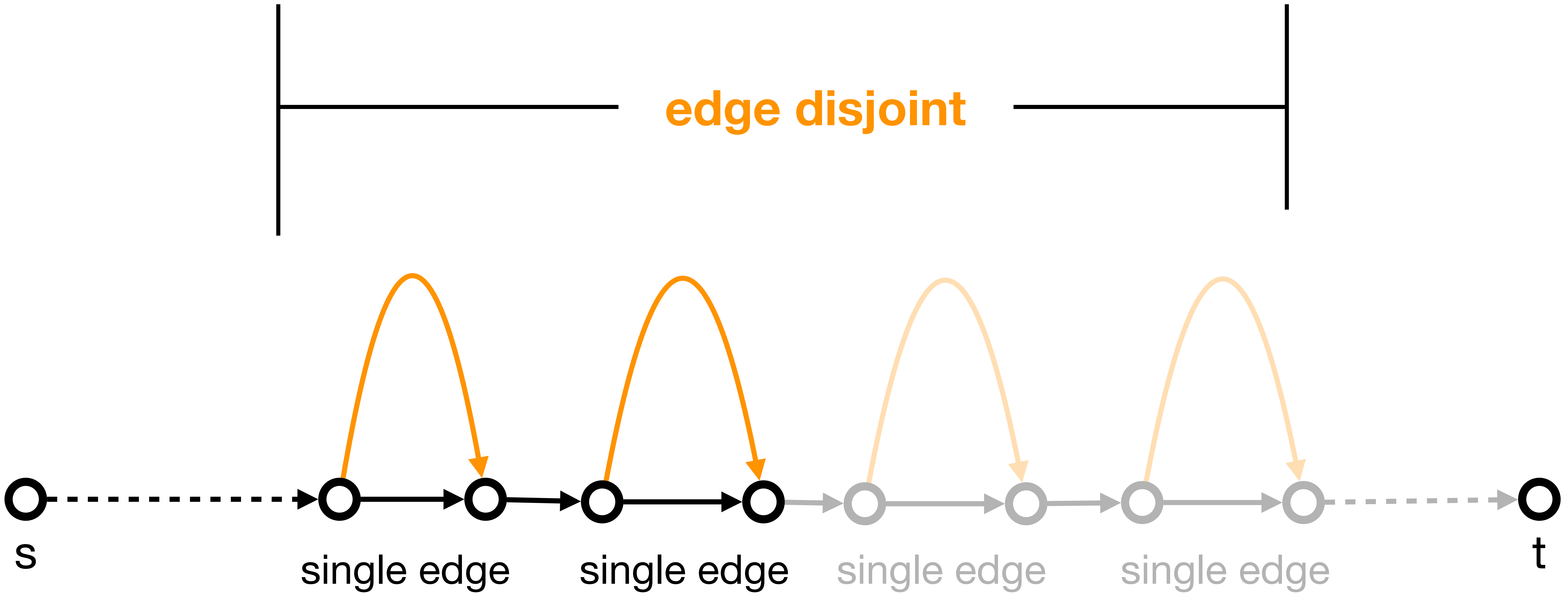


Wishful thinking



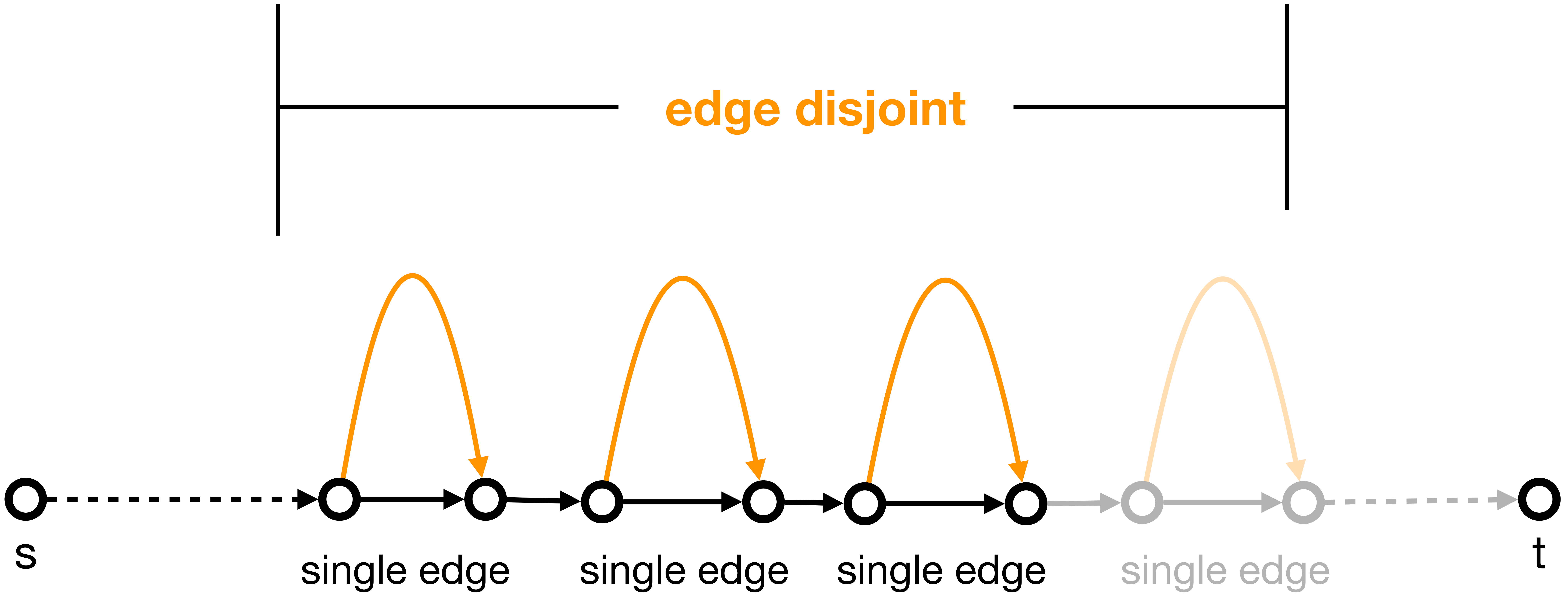
Add edges & detours one by one using progressive Dijkstra

Wishful thinking



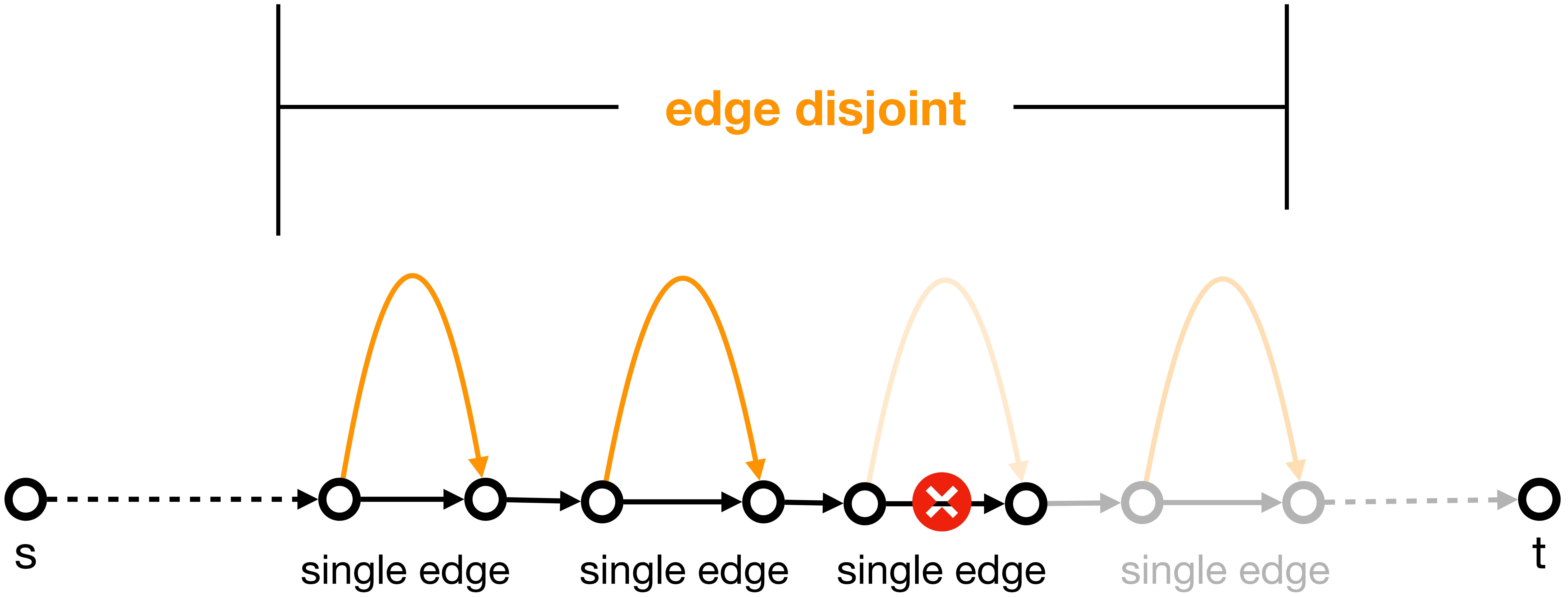
Add edges & detours one by one using progressive Dijkstra

Wishful thinking



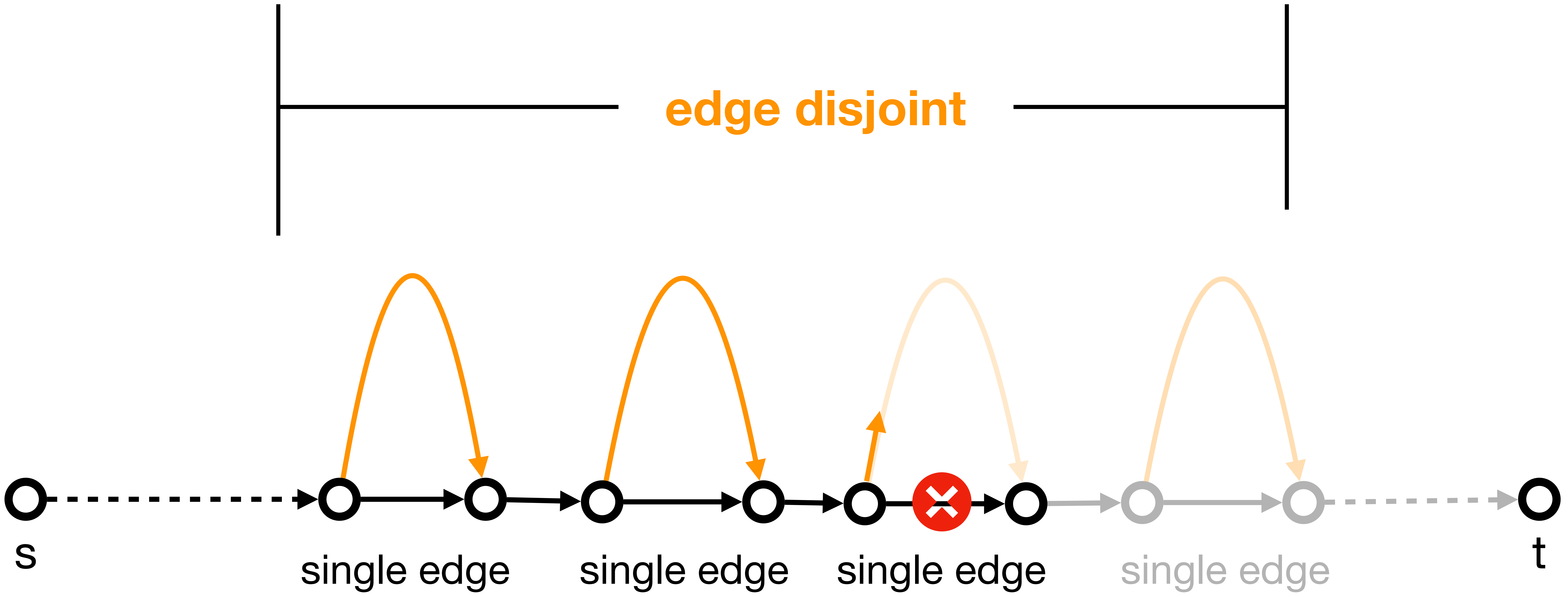
Add edges & detours one by one using progressive Dijkstra

Wishful thinking



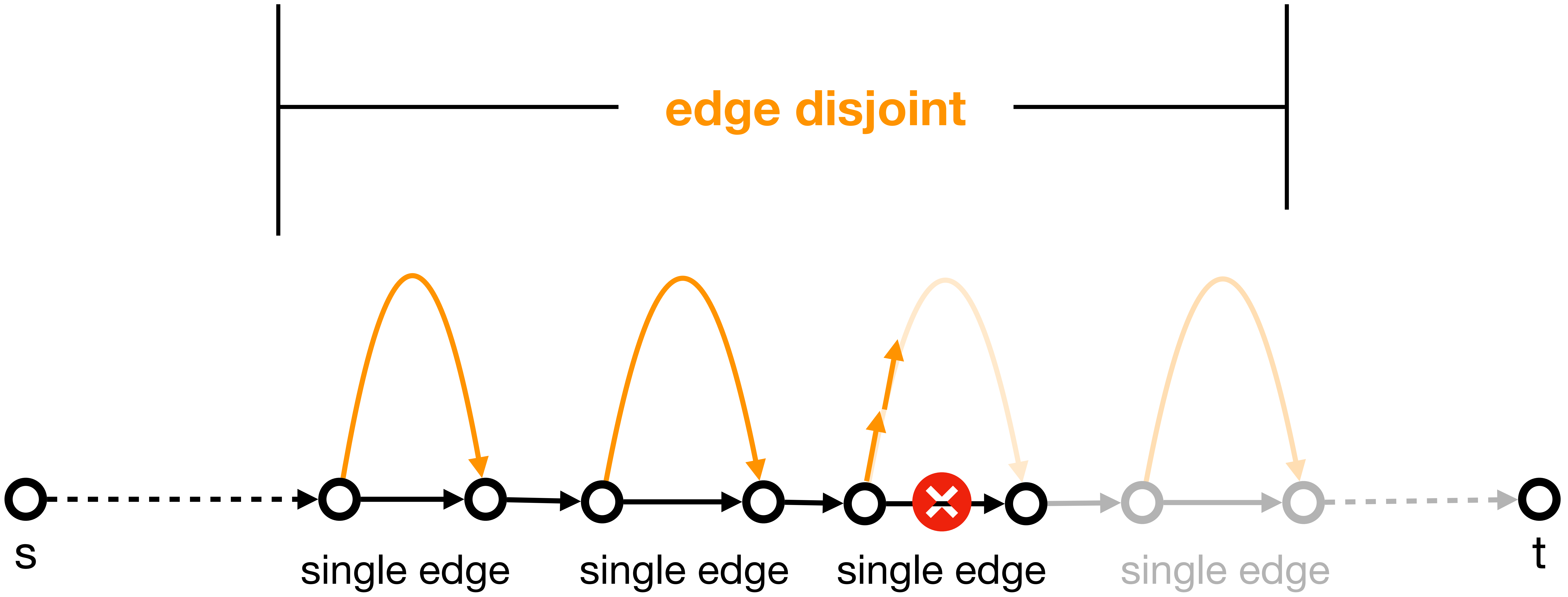
Add edges on detours one by one, also using progressive Dijkstra

Wishful thinking



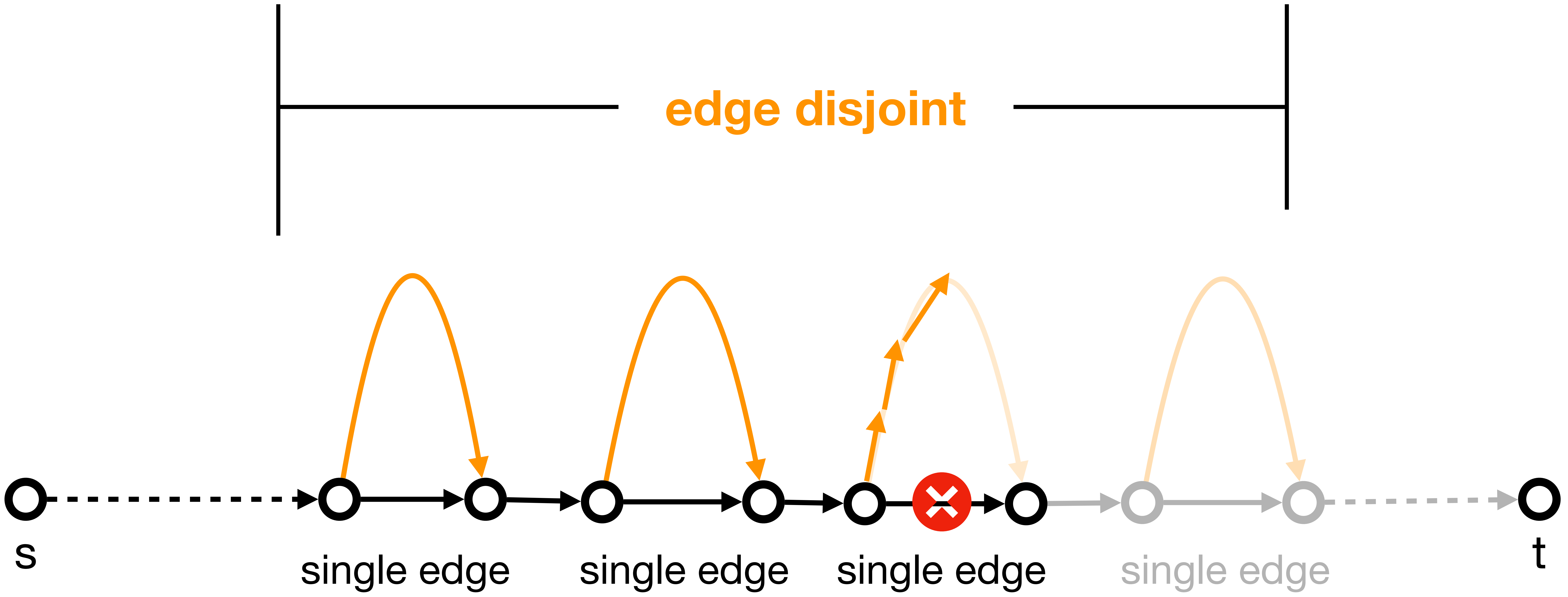
Add edges on detours one by one, also using progressive Dijkstra

Wishful thinking



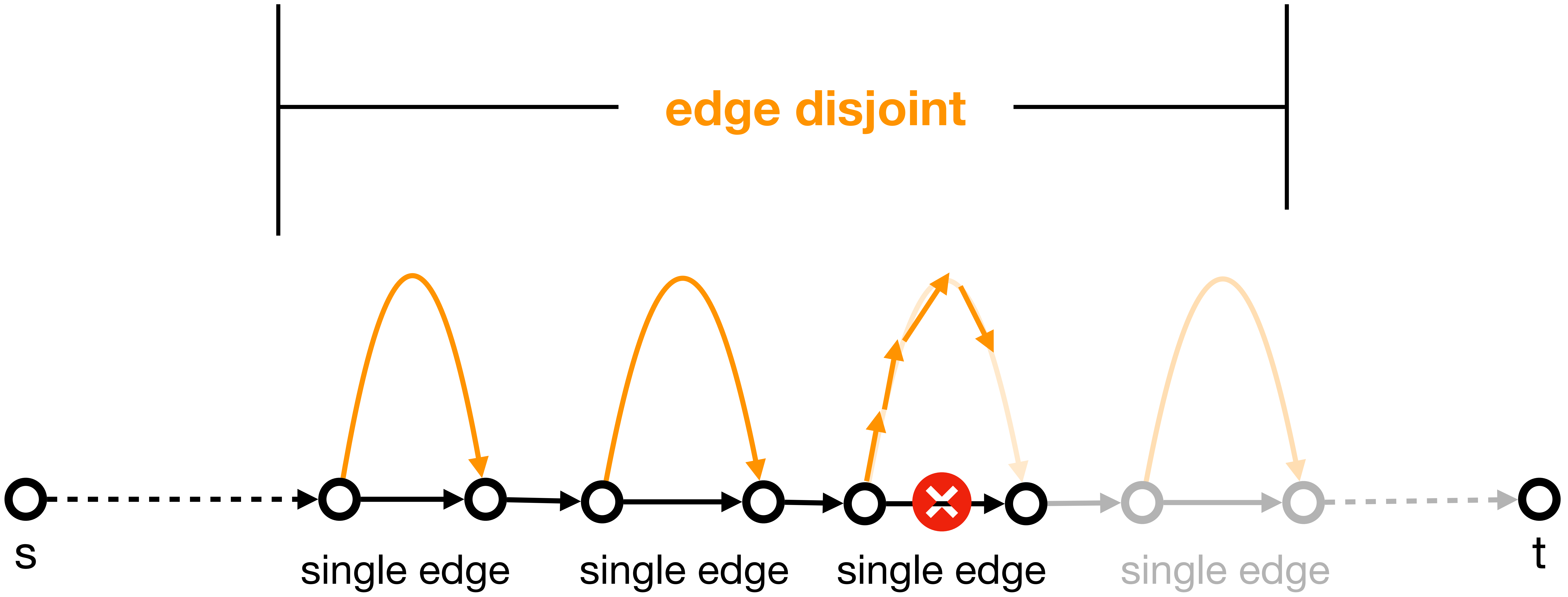
Add edges on detours one by one, also using progressive Dijkstra

Wishful thinking



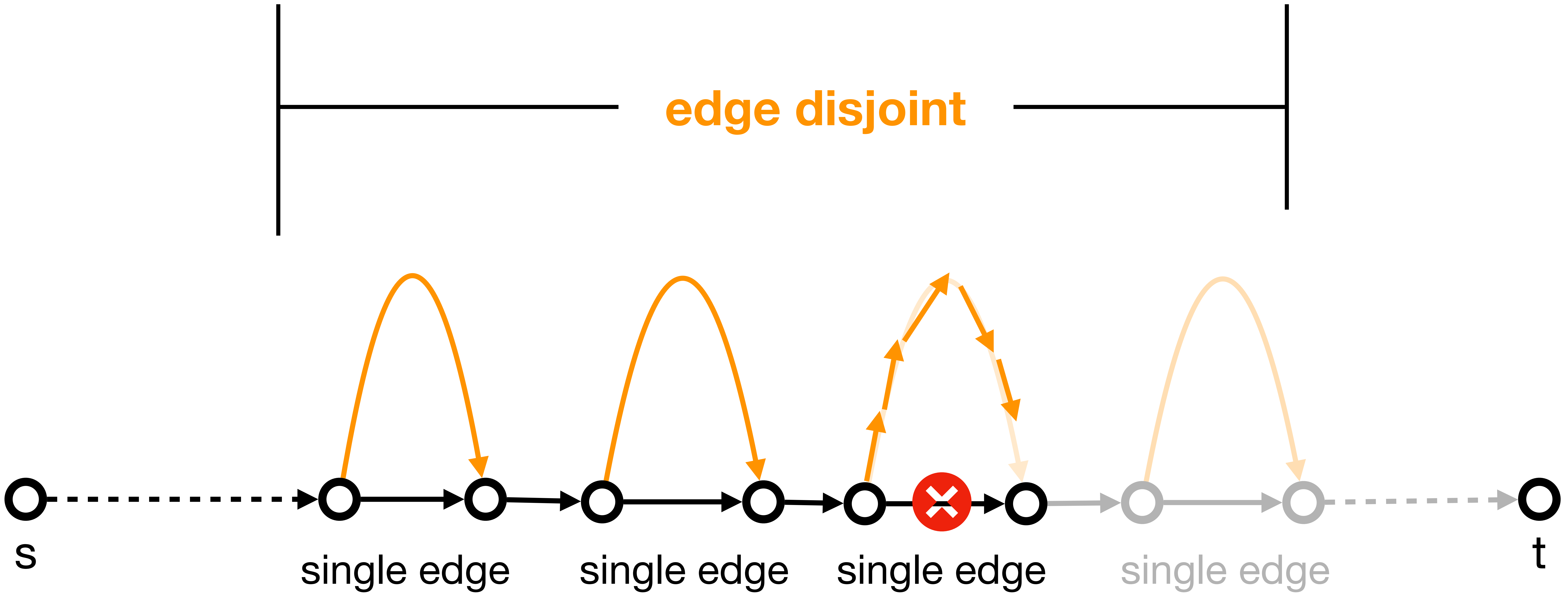
Add edges on detours one by one, also using progressive Dijkstra

Wishful thinking



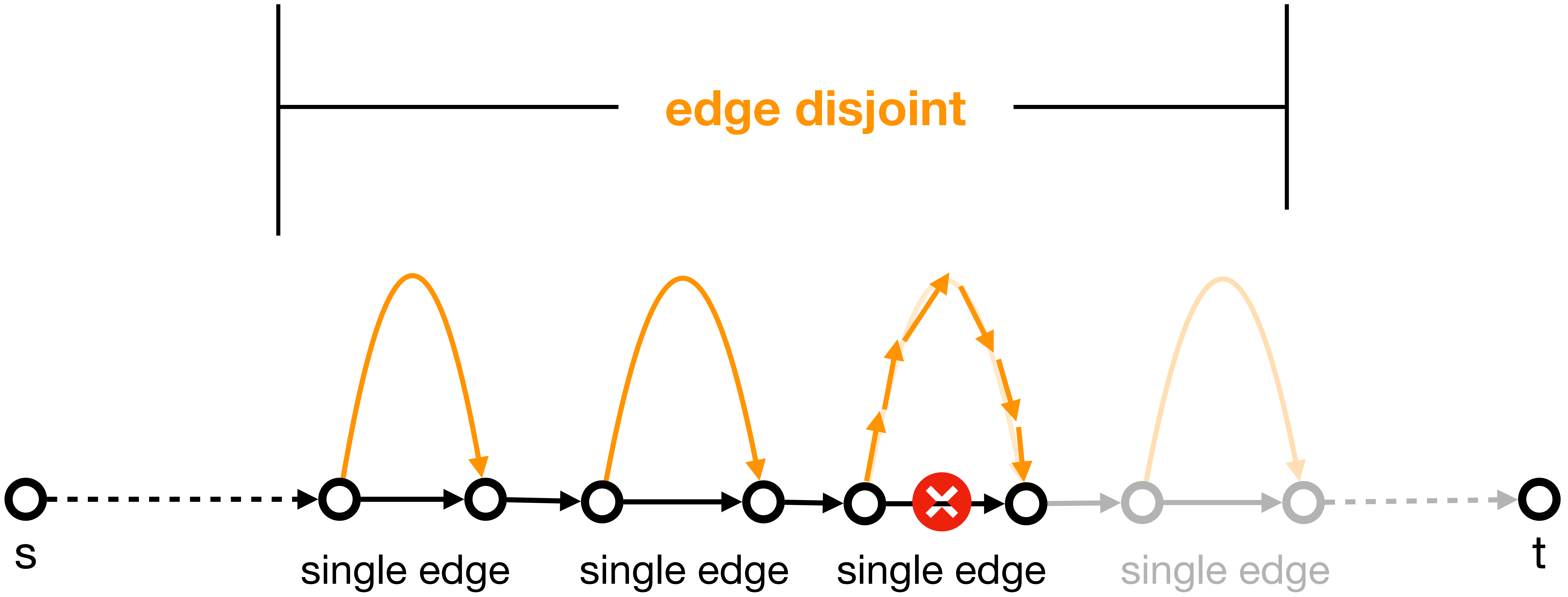
Add edges on detours one by one, also using progressive Dijkstra

Wishful thinking



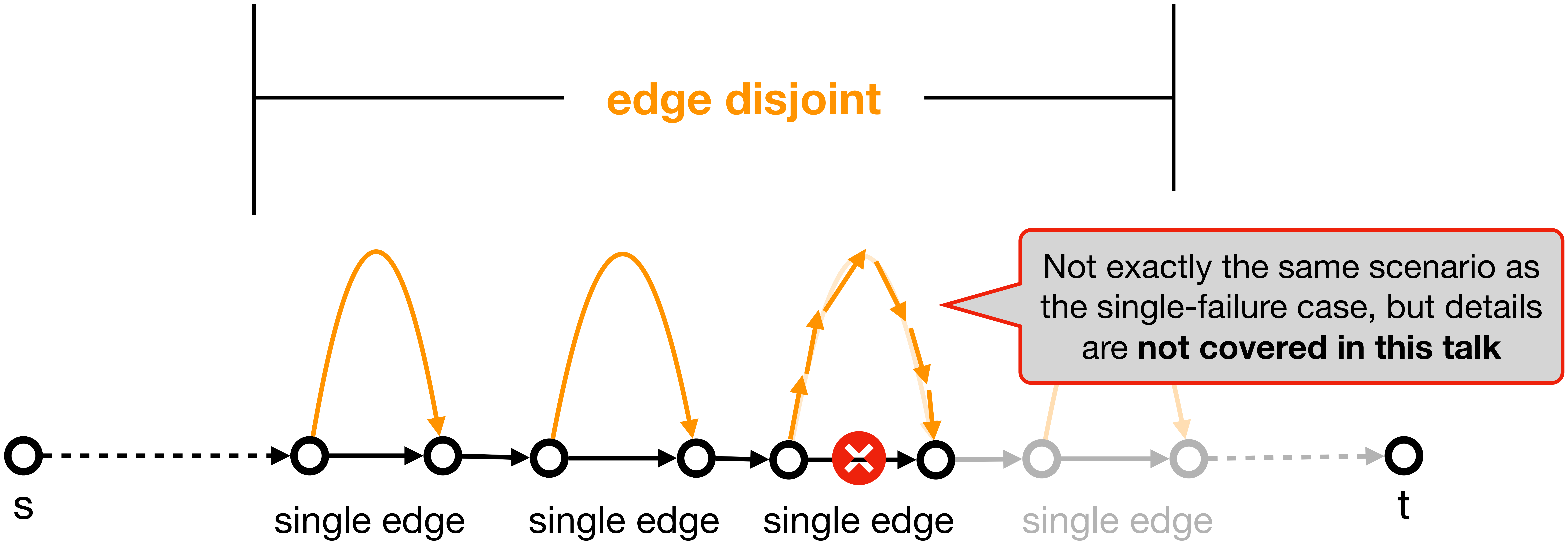
Add edges on detours one by one, also using progressive Dijkstra

Wishful thinking



Add edges on detours one by one, also using progressive Dijkstra

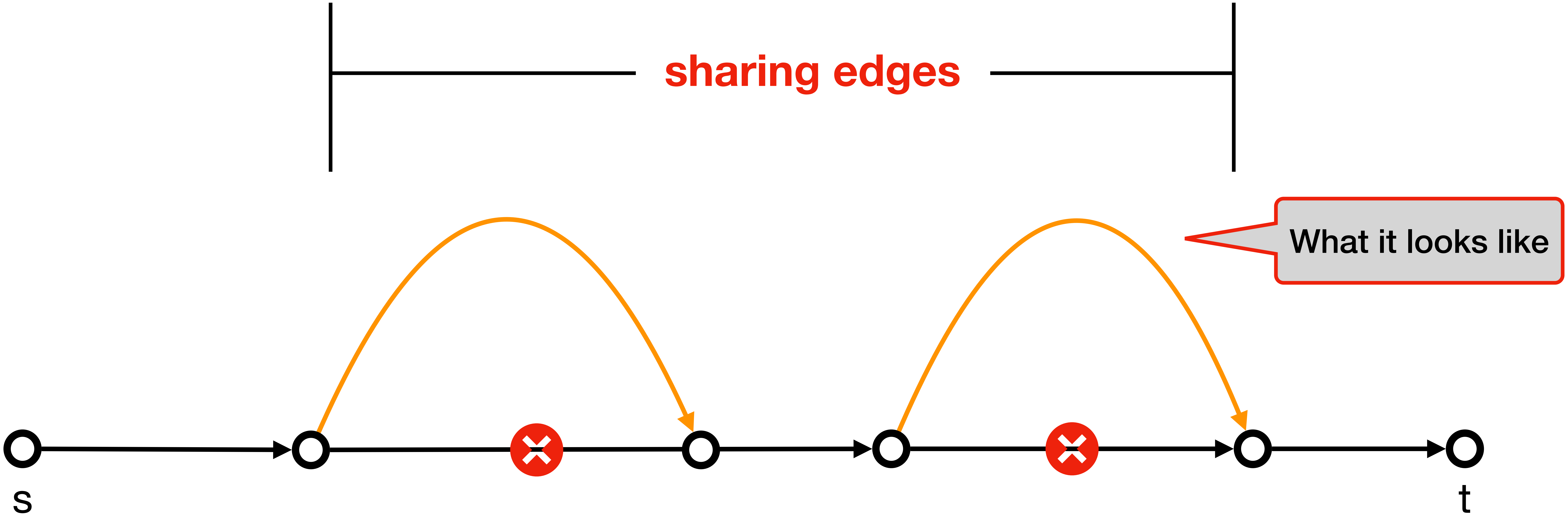
Wishful thinking



Add edges on detours one by one, also using progressive Dijkstra

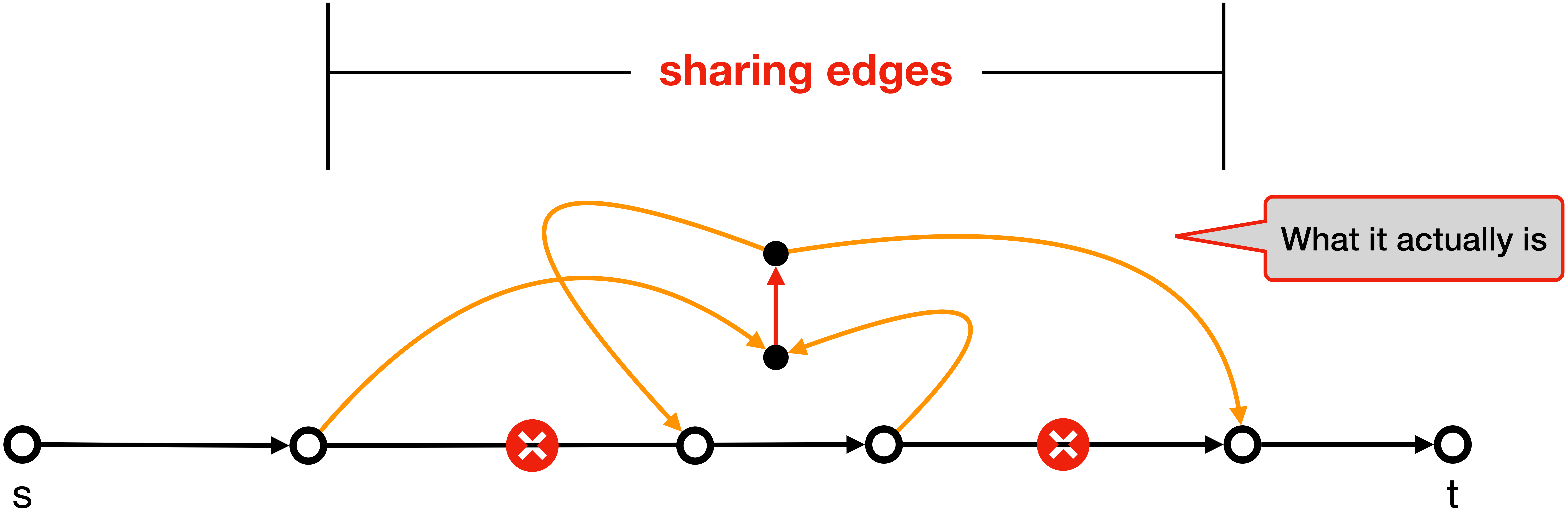
Wishful thinking: technical issues

<u>Main issue:</u> Detours may intersect



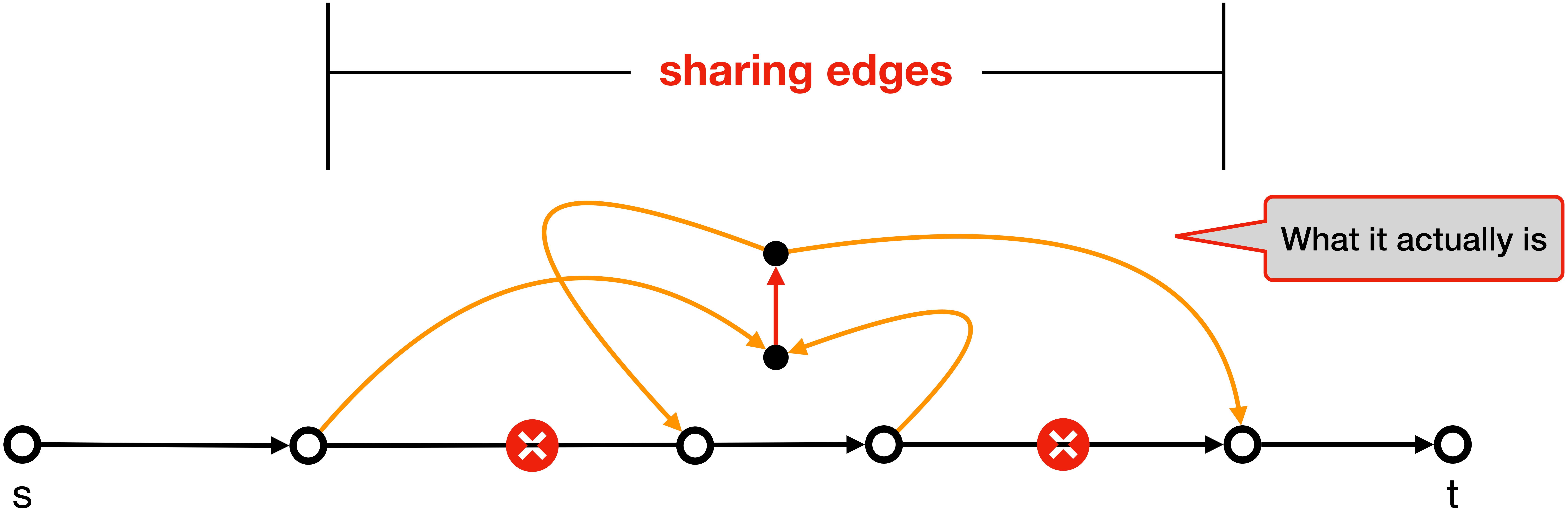
Wishful thinking: technical issues

Main issue: Detours may intersect



Wishful thinking: technical issues

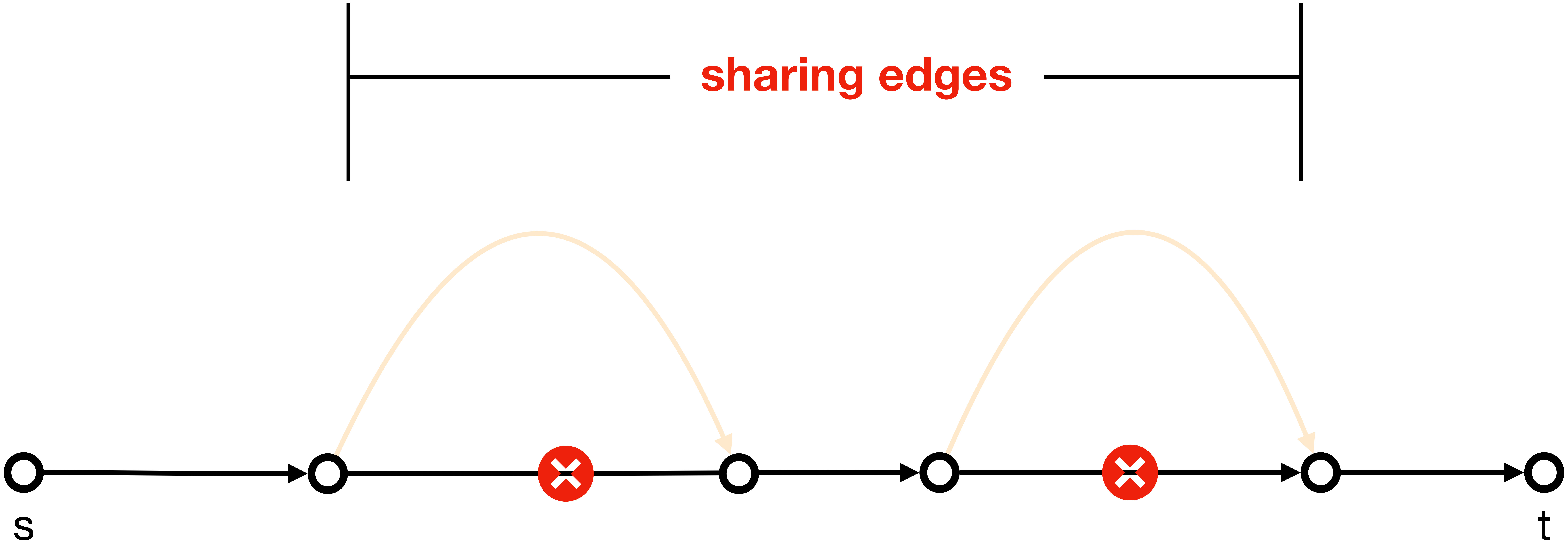
Main issue: Detours may intersect



Consequence: cannot use progressive Dijkstra

Wishful thinking: technical issues

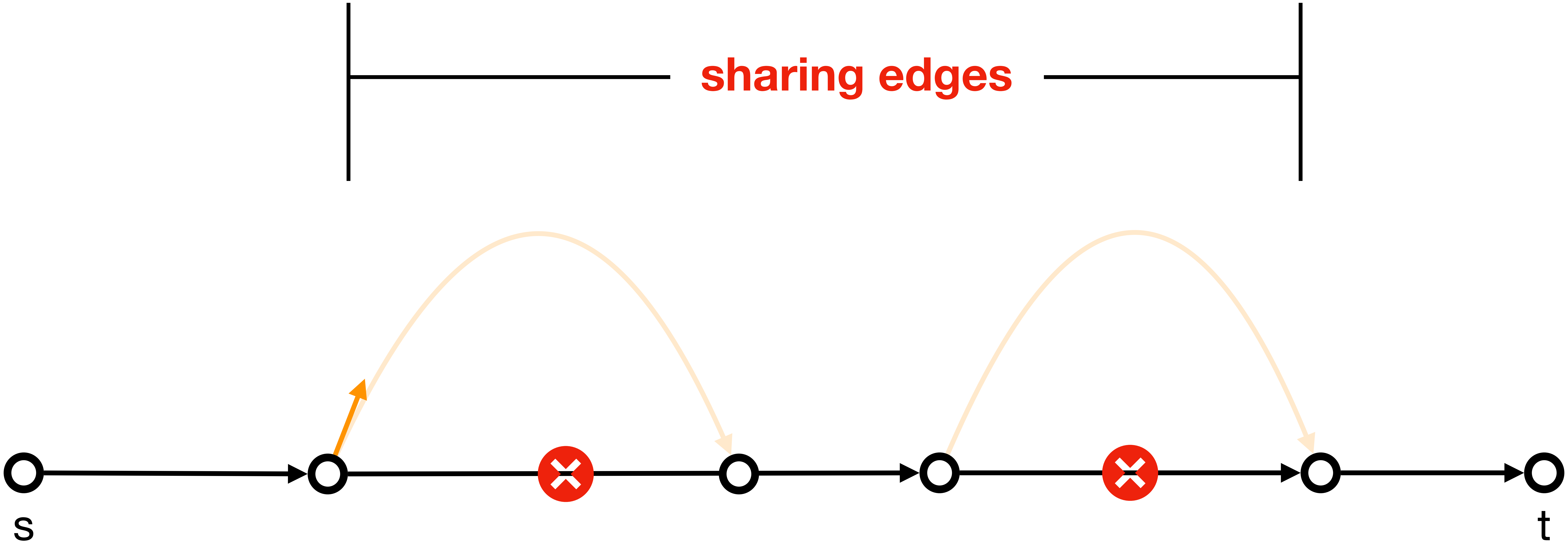
Main issue: Detours may intersect



Consequence: cannot use progressive Dijkstra

Wishful thinking: technical issues

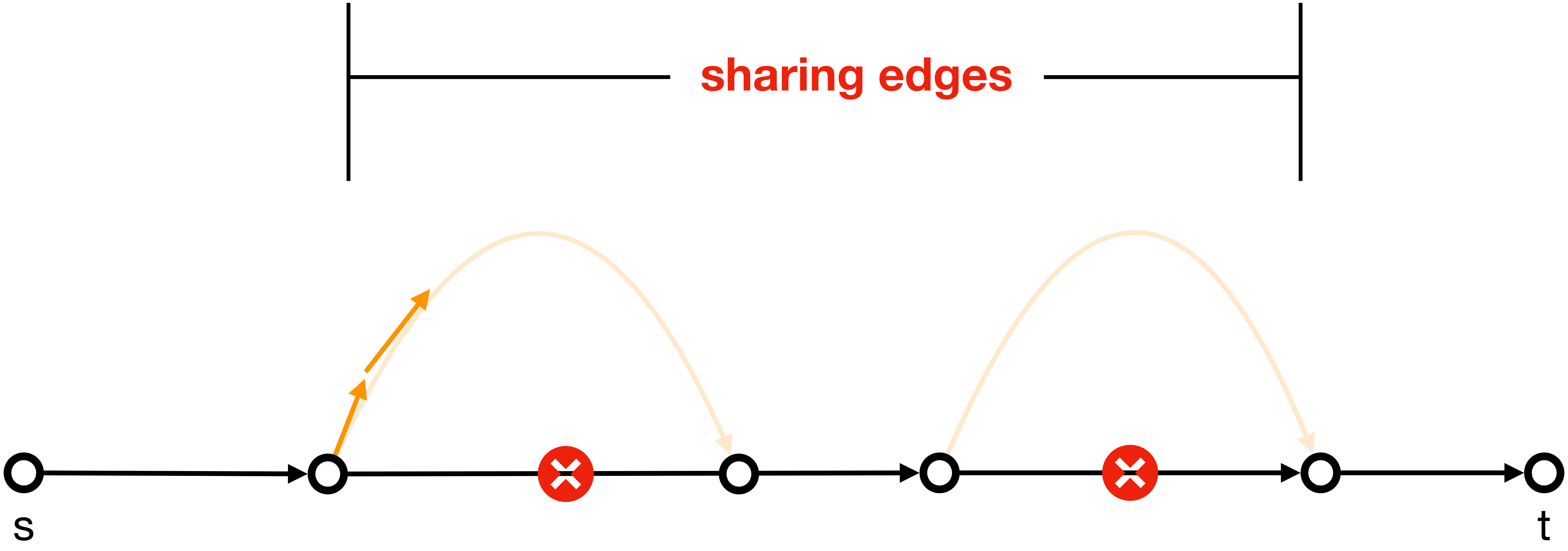
Main issue: Detours may intersect



Consequence: cannot use progressive Dijkstra

Wishful thinking: technical issues

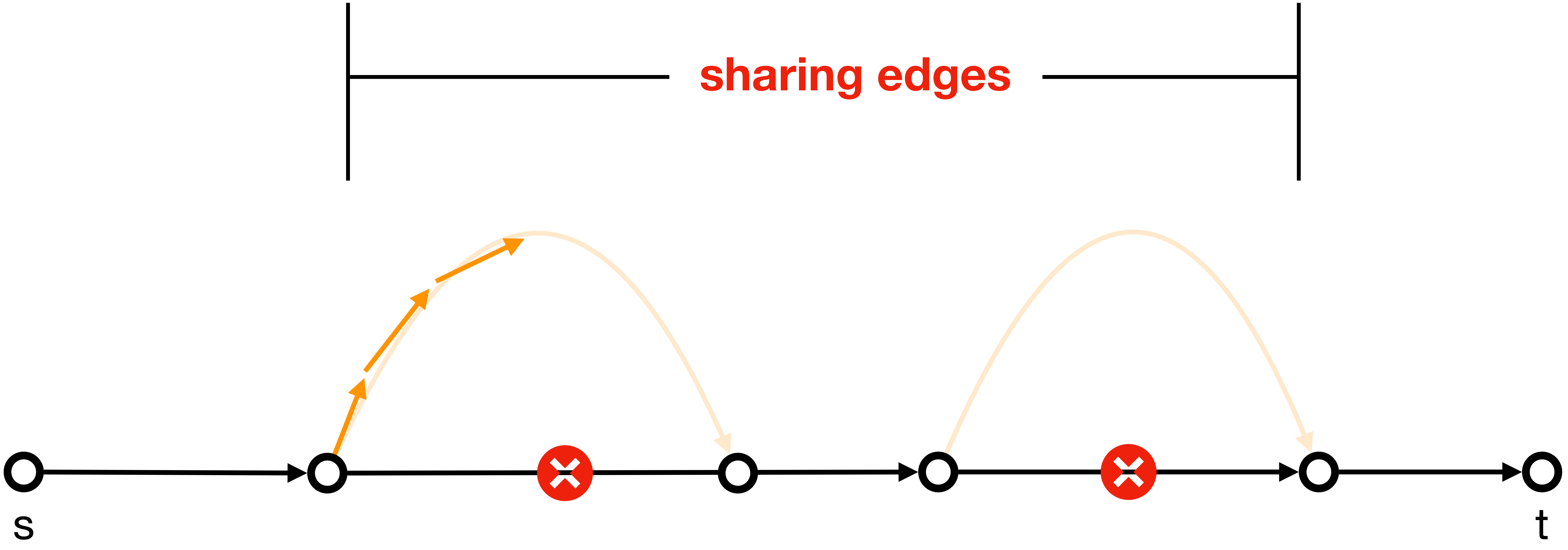
Main issue: Detours may intersect



Consequence: cannot use progressive Dijkstra

Wishful thinking: technical issues

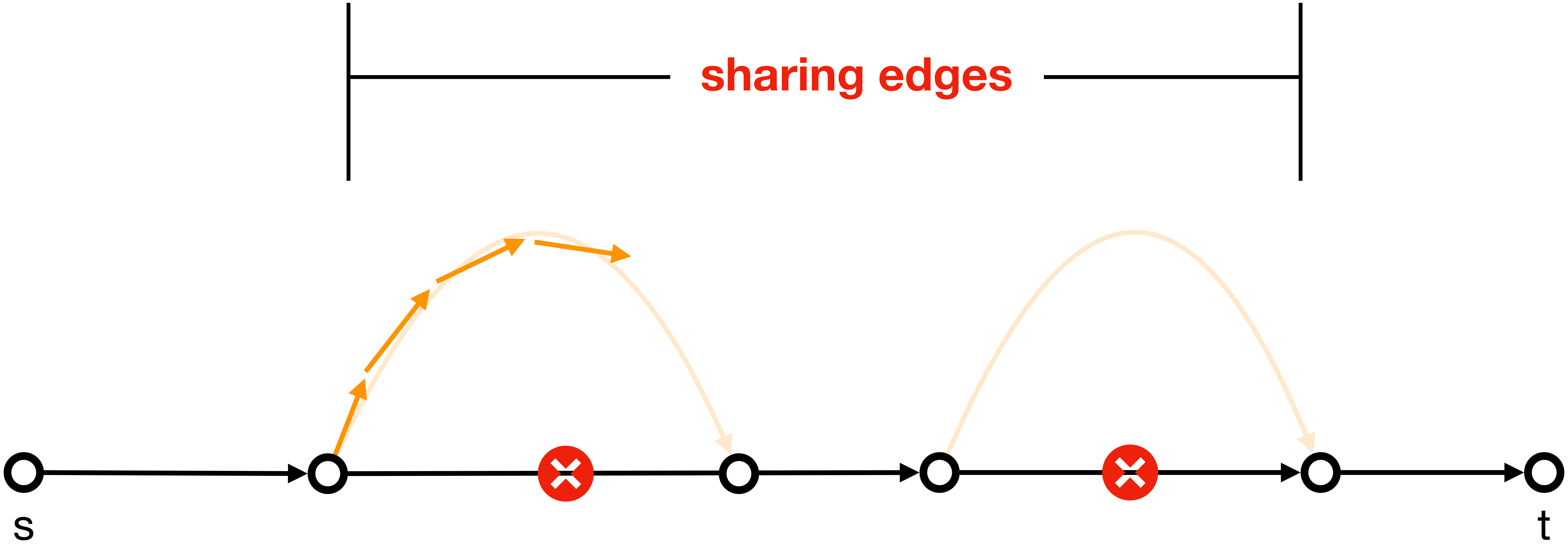
Main issue: Detours may intersect



Consequence: cannot use progressive Dijkstra

Wishful thinking: technical issues

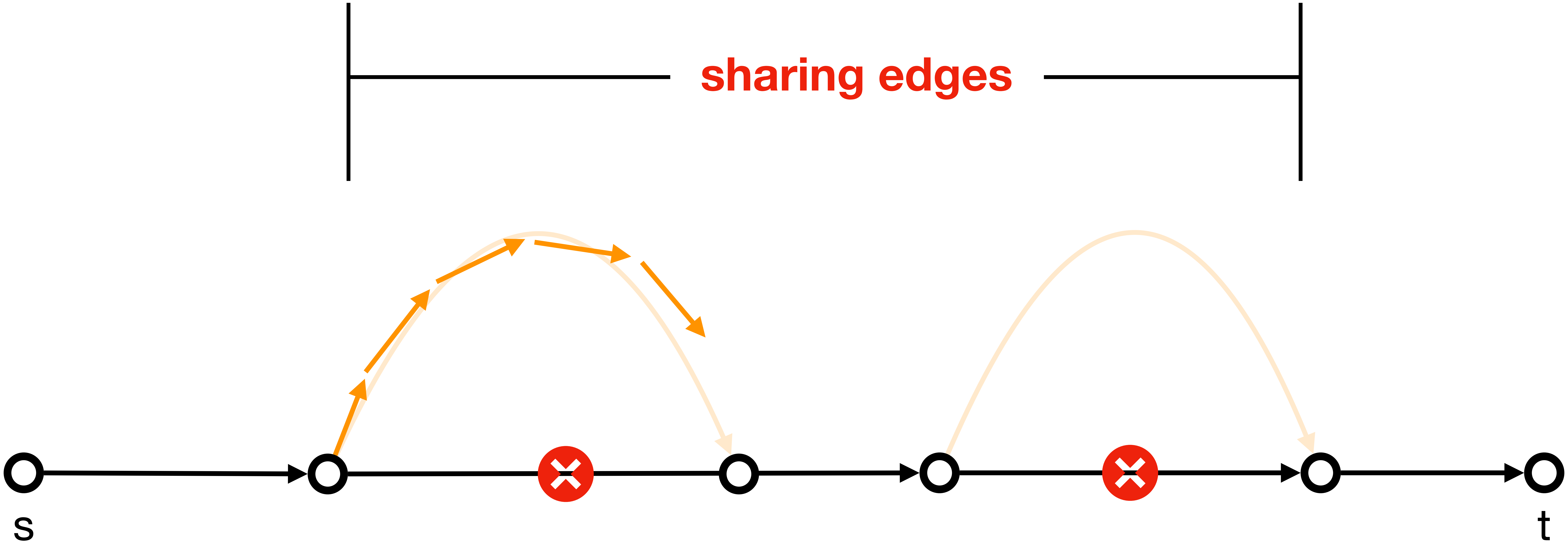
<u>Main issue:</u> Detours may intersect



<u>Consequence:</u> cannot use progressive Dijkstra

Wishful thinking: technical issues

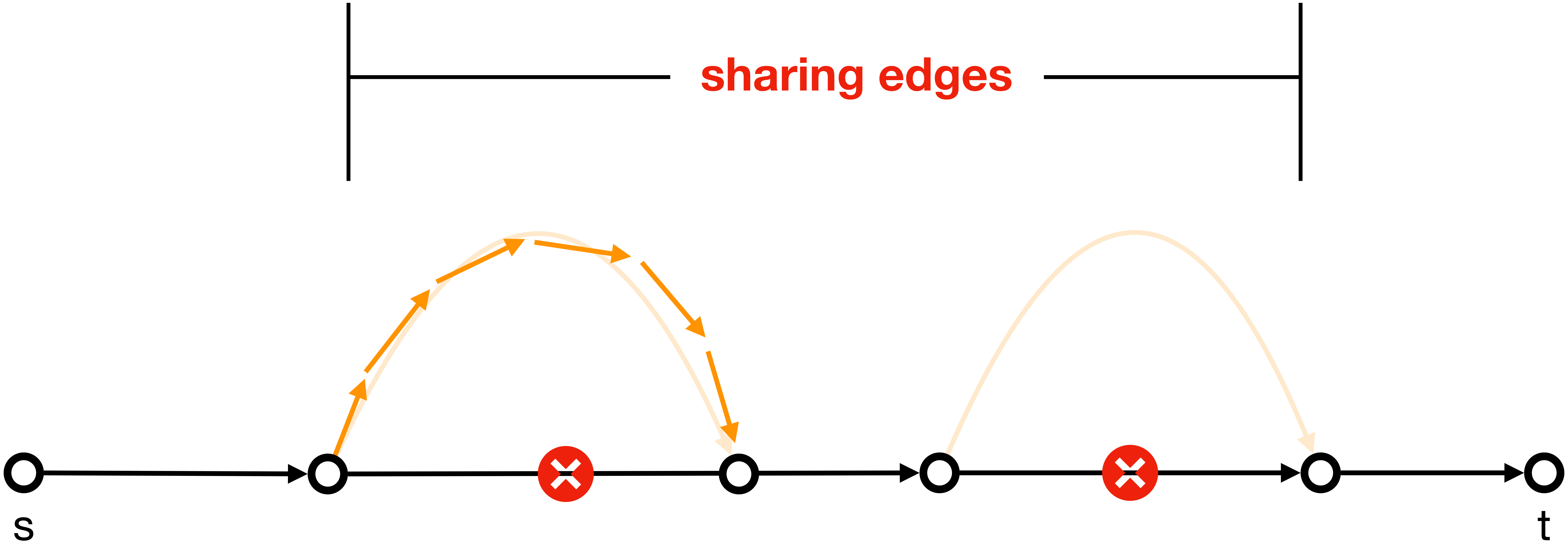
Main issue: Detours may intersect



Consequence: cannot use progressive Dijkstra

Wishful thinking: technical issues

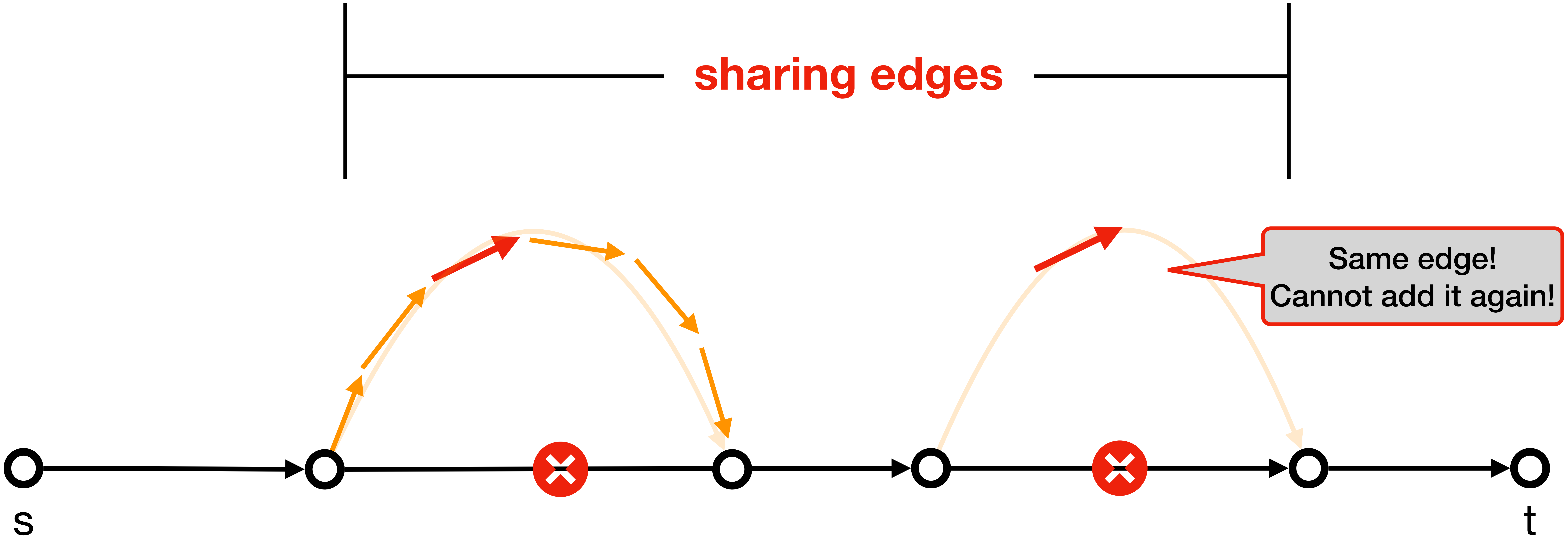
Main issue: Detours may intersect



Consequence: cannot use progressive Dijkstra

Wishful thinking: technical issues

Main issue: Detours may intersect

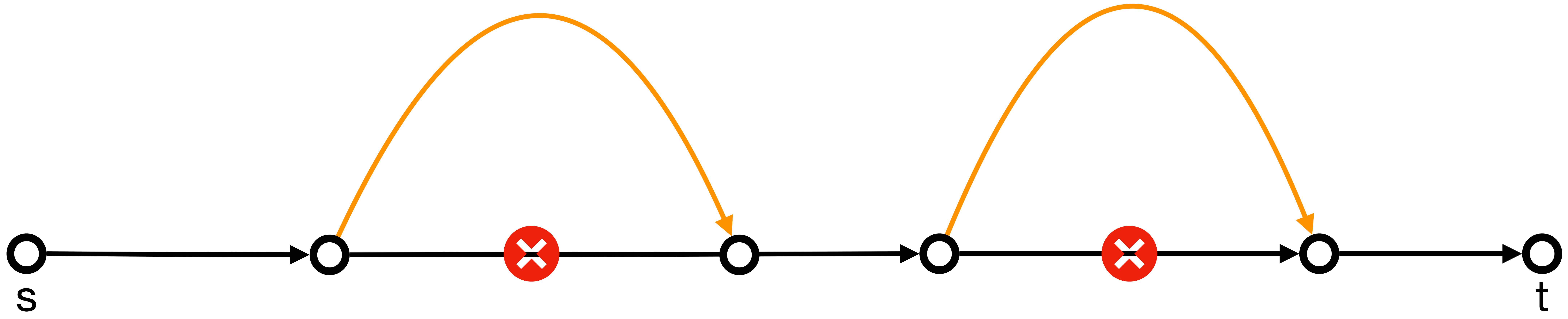


Consequence: cannot use progressive Dijkstra

Dealing with detour intersections

Main issue: Detours may intersect

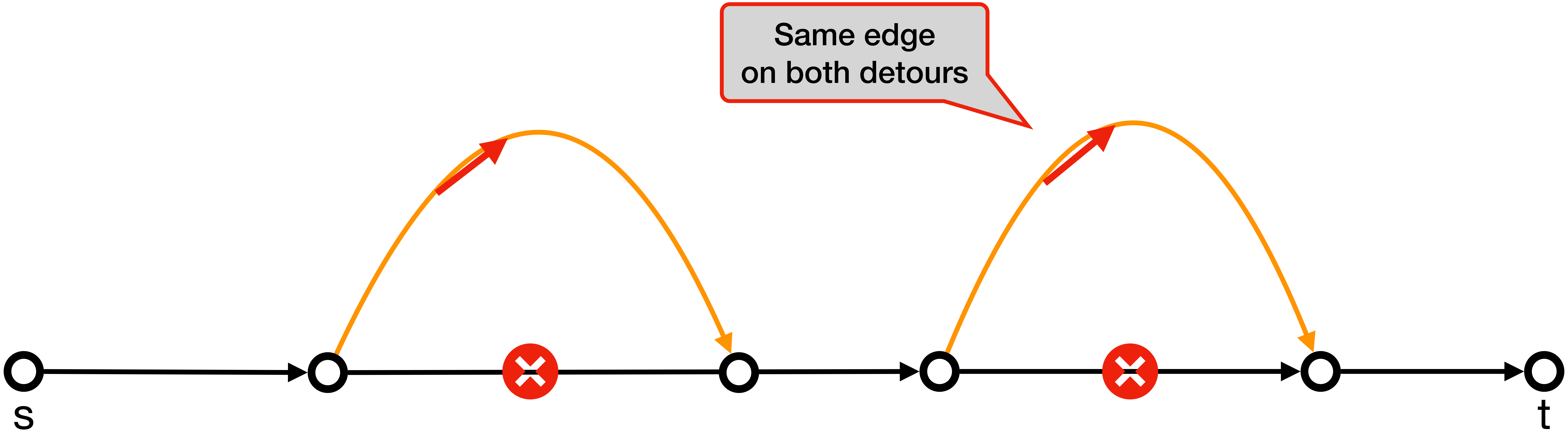
Solution: **Concat**enate detour while **doubling** the span on st-path



Dealing with detour intersections

Main issue: Detours may intersect

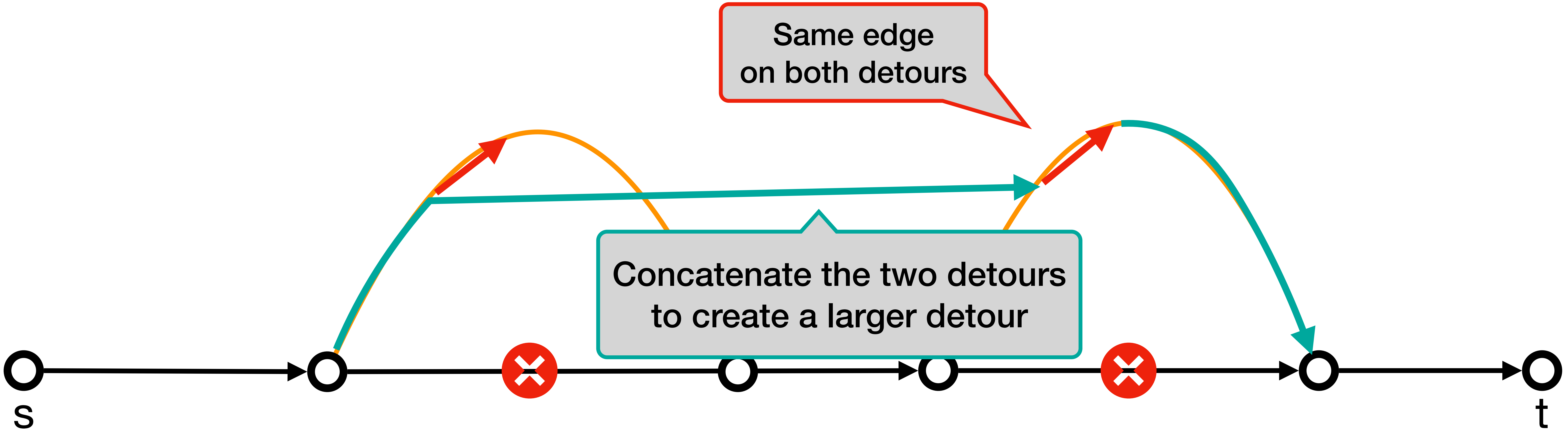
Solution: Concatenate detour while doubling the span on st-path



Dealing with detour intersections

Main issue: Detours may intersect

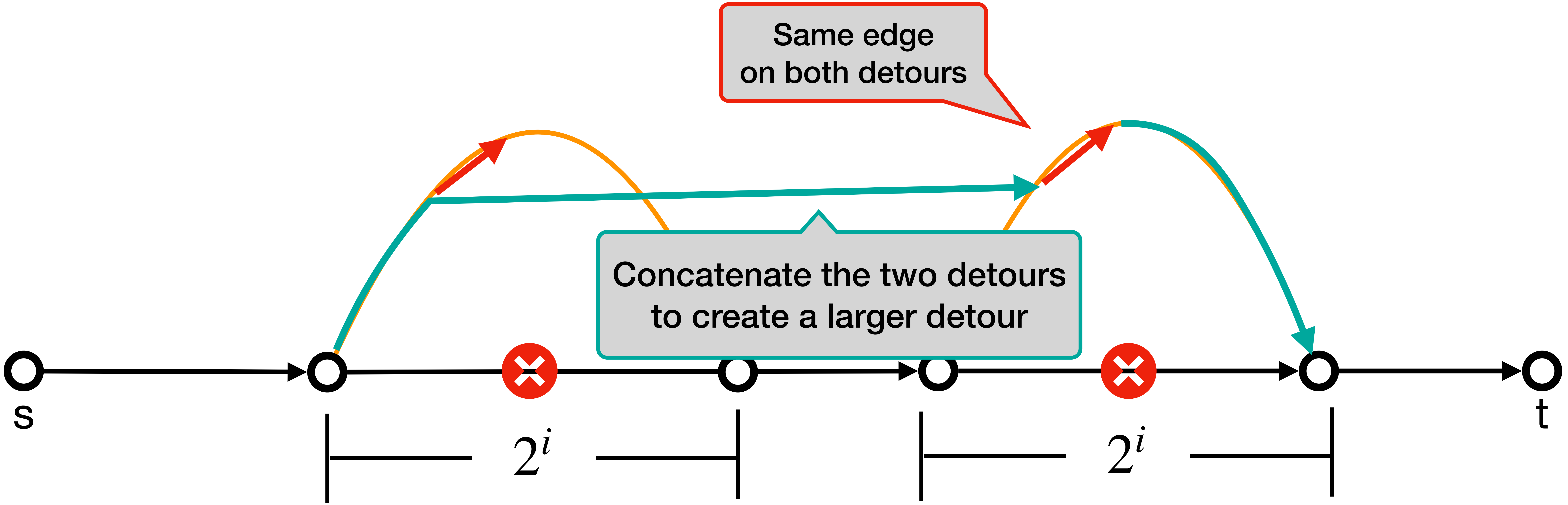
Solution: **Concat**enate detour while **doubling** the span on st-path



Dealing with detour intersections

Main issue: Detours may intersect

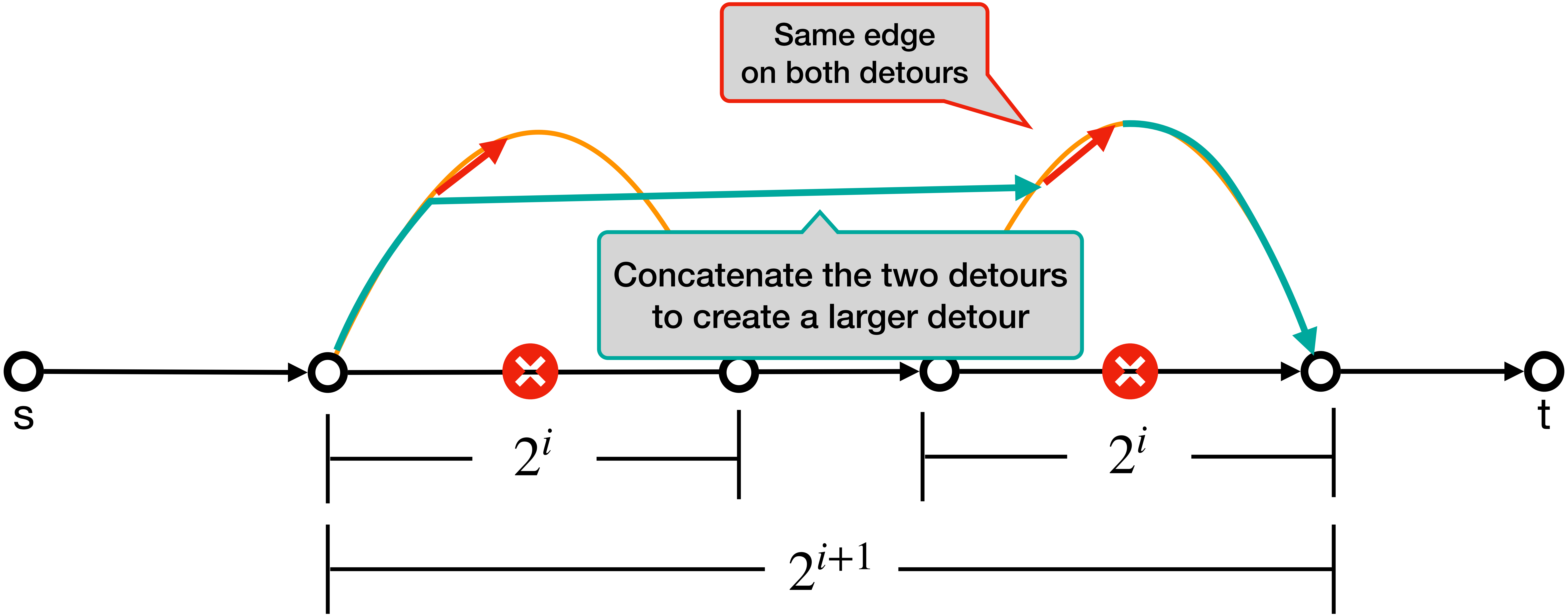
Solution: Concatenate detour while **doubling** the span on st-path



Dealing with detour intersections

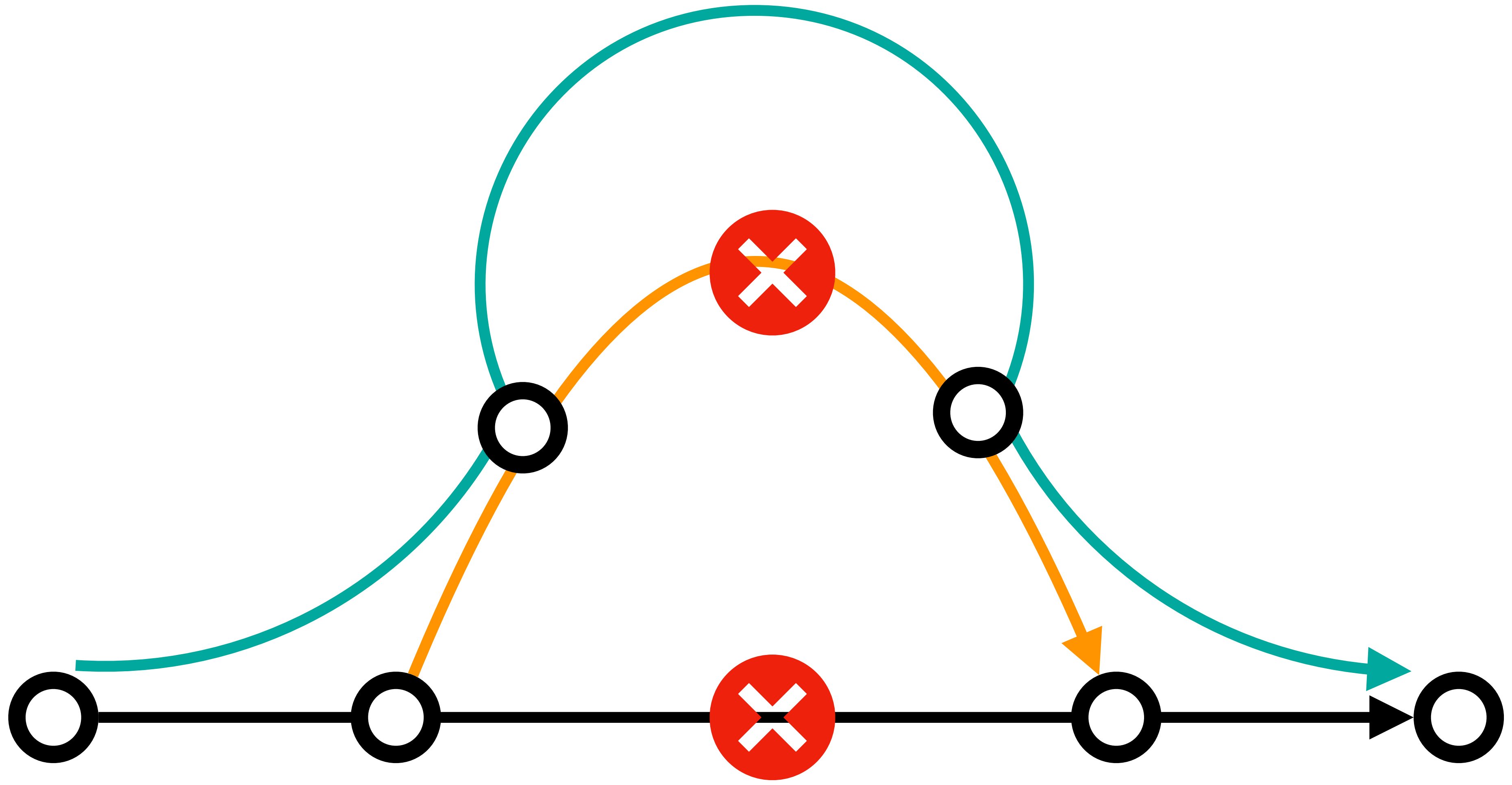
Main issue: Detours may intersect

Solution: **Concat**enate detour while **doubling** the span on st-path

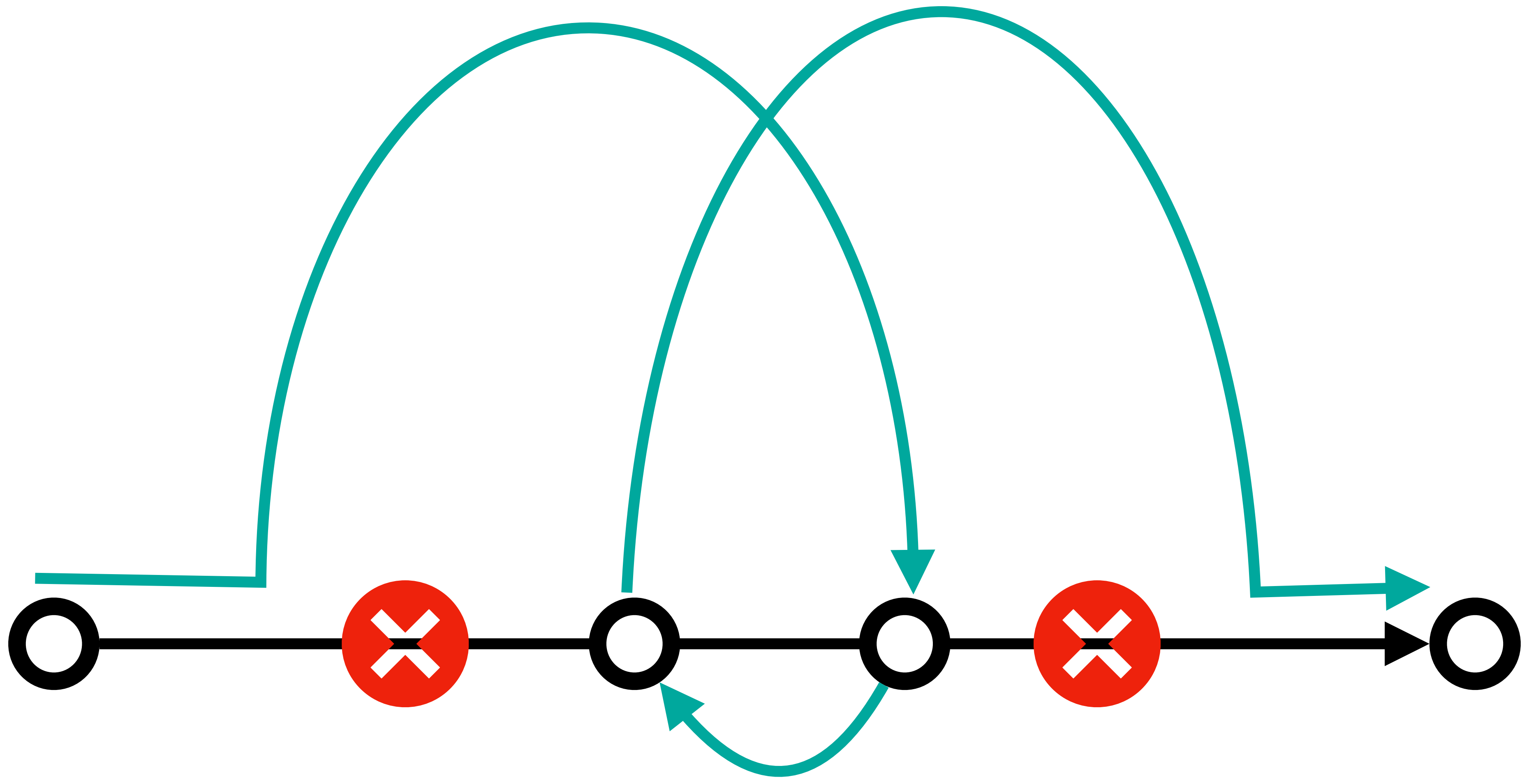


Two main cases

One failure on st-path

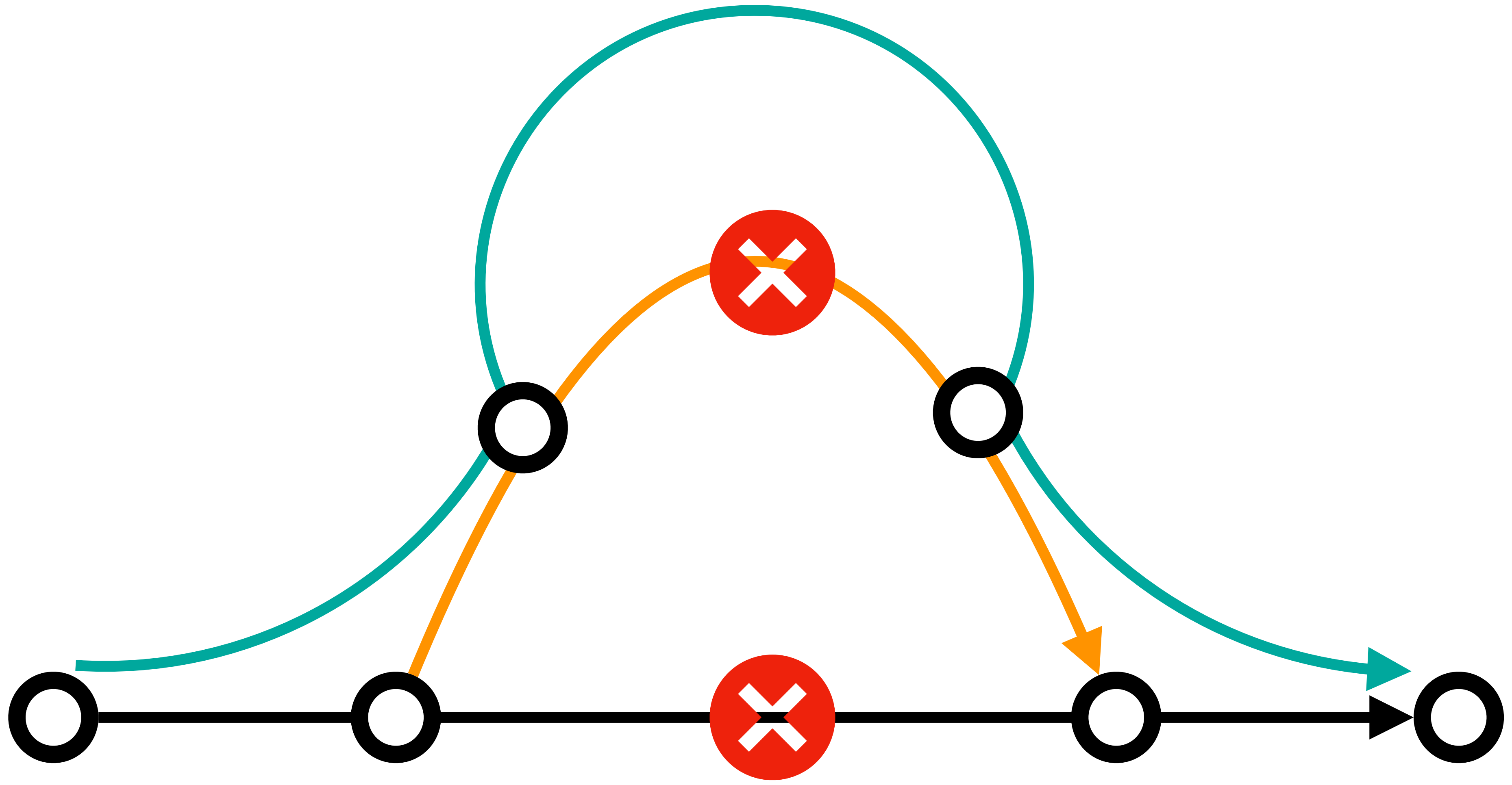


Both failures on st-path

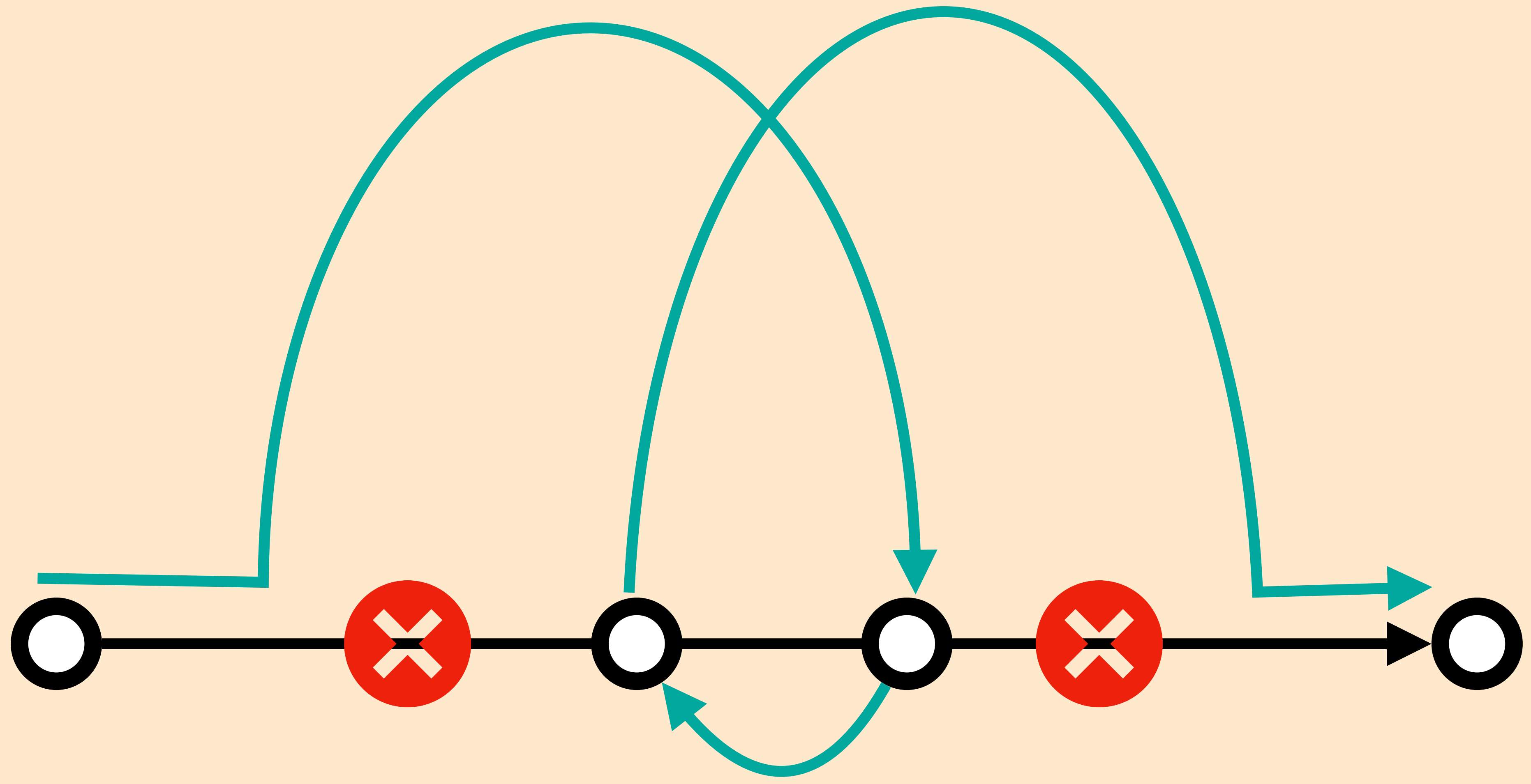


Two main cases

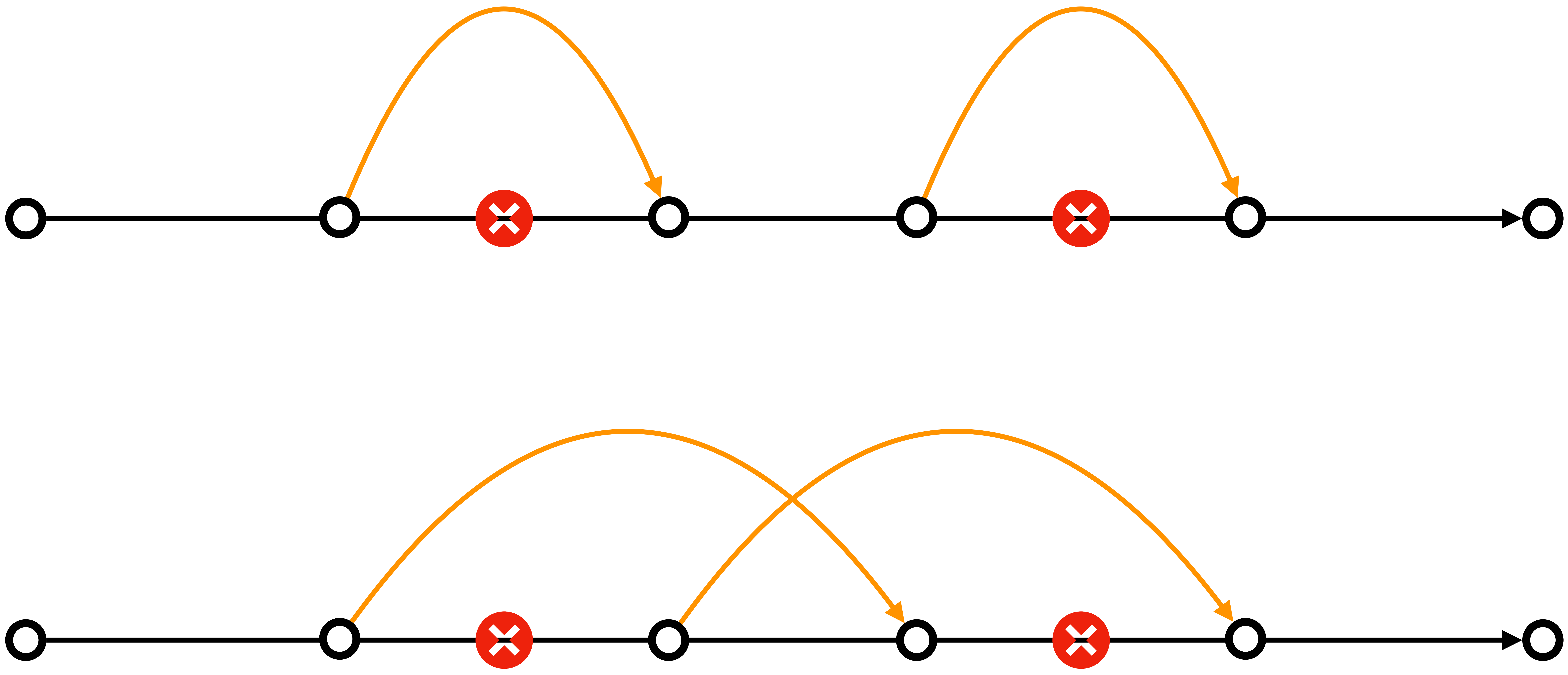
One failure on st-path



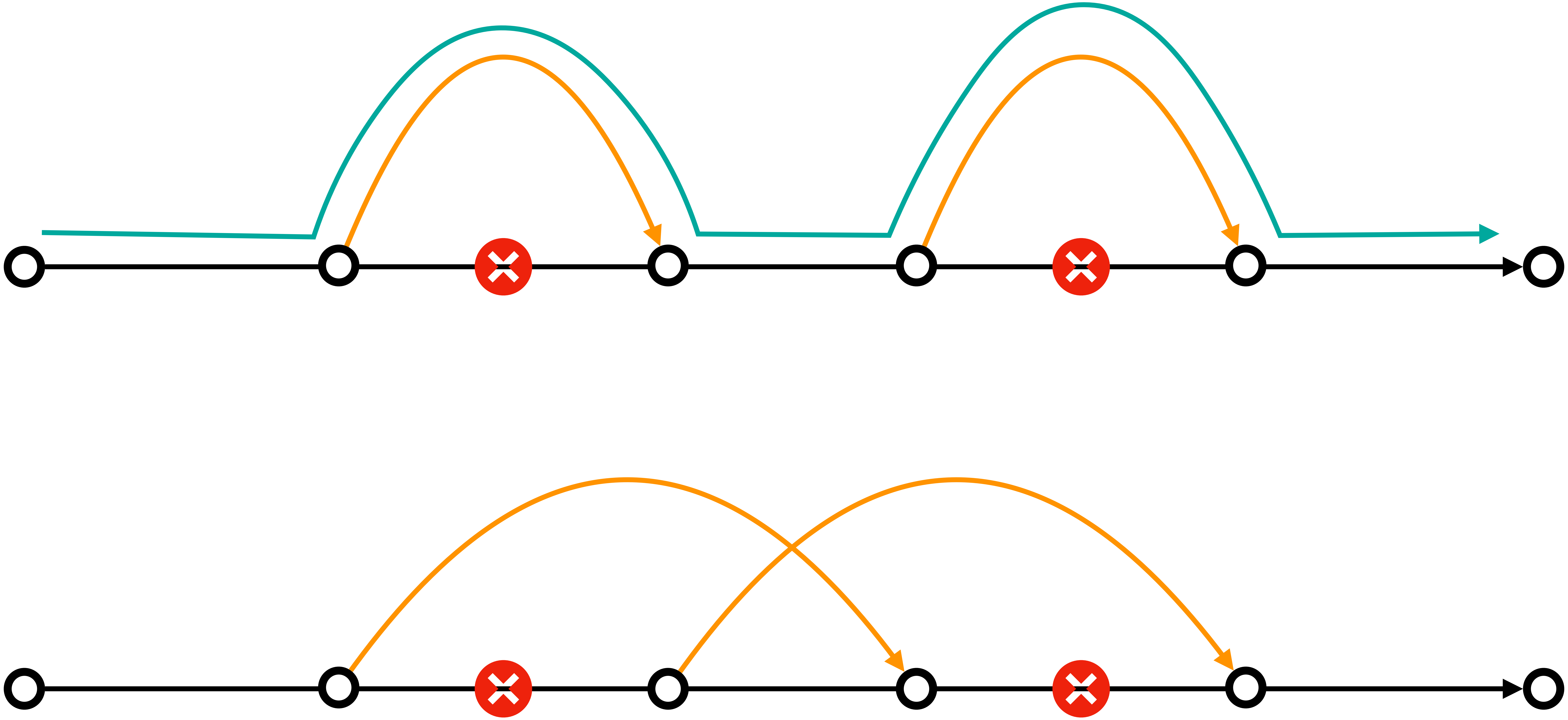
Both failures on st-path



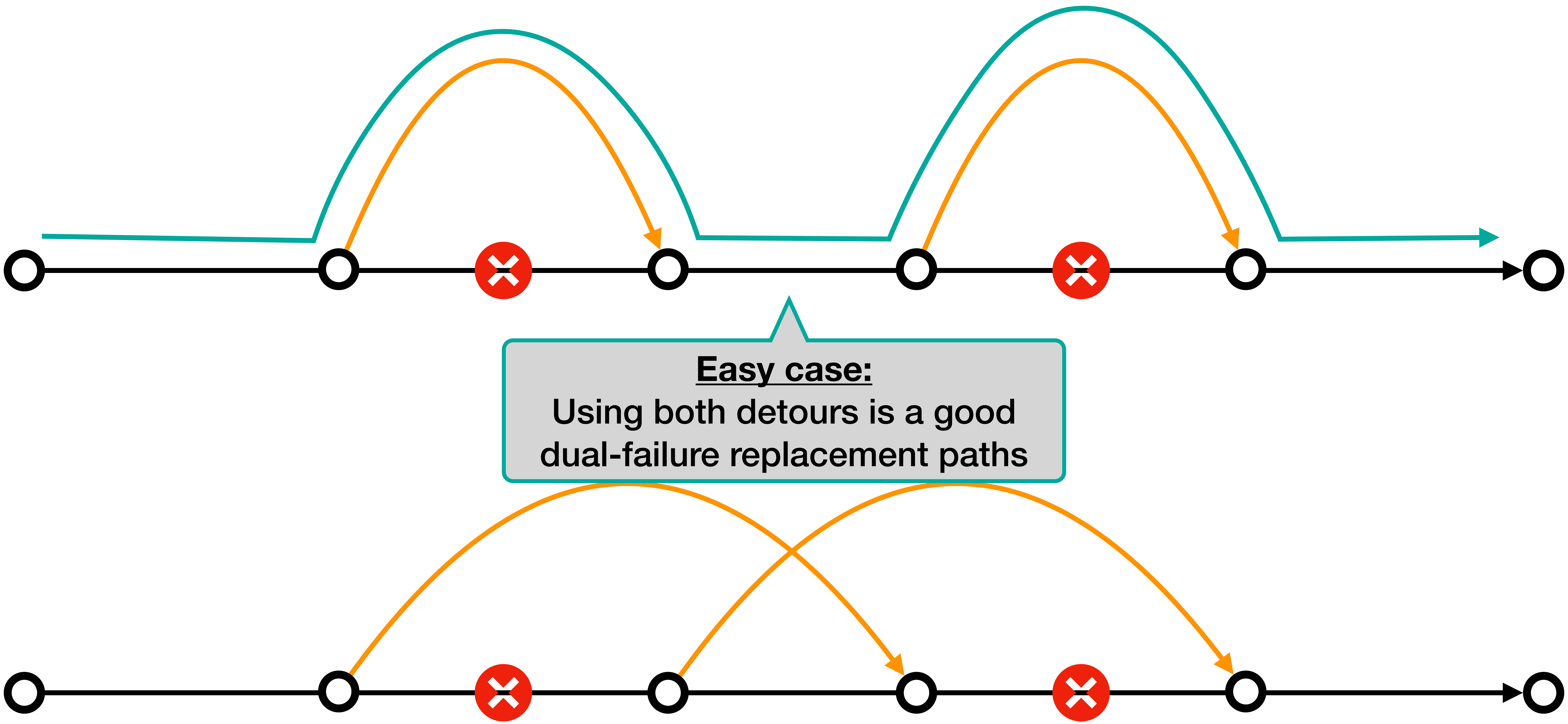
Easy & Hard Cases



Easy & Hard Cases

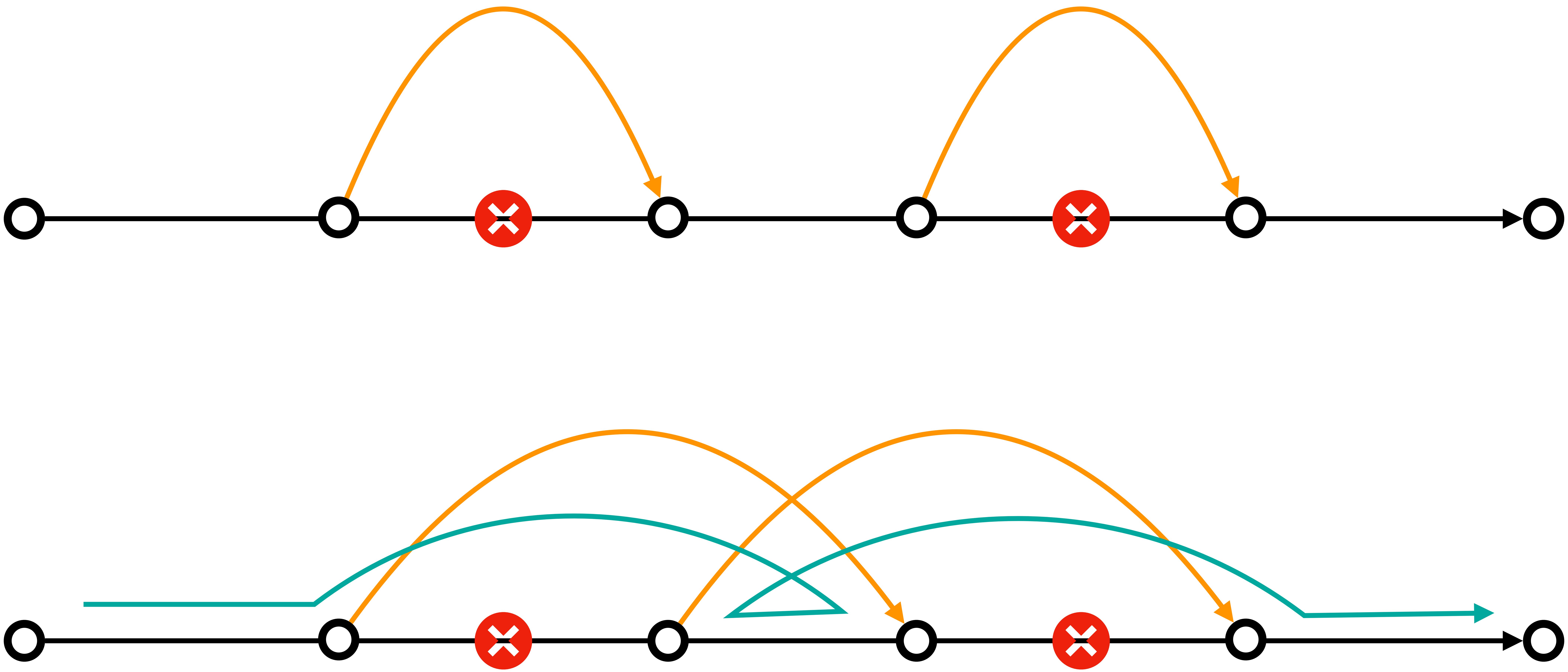


Easy & Hard Cases

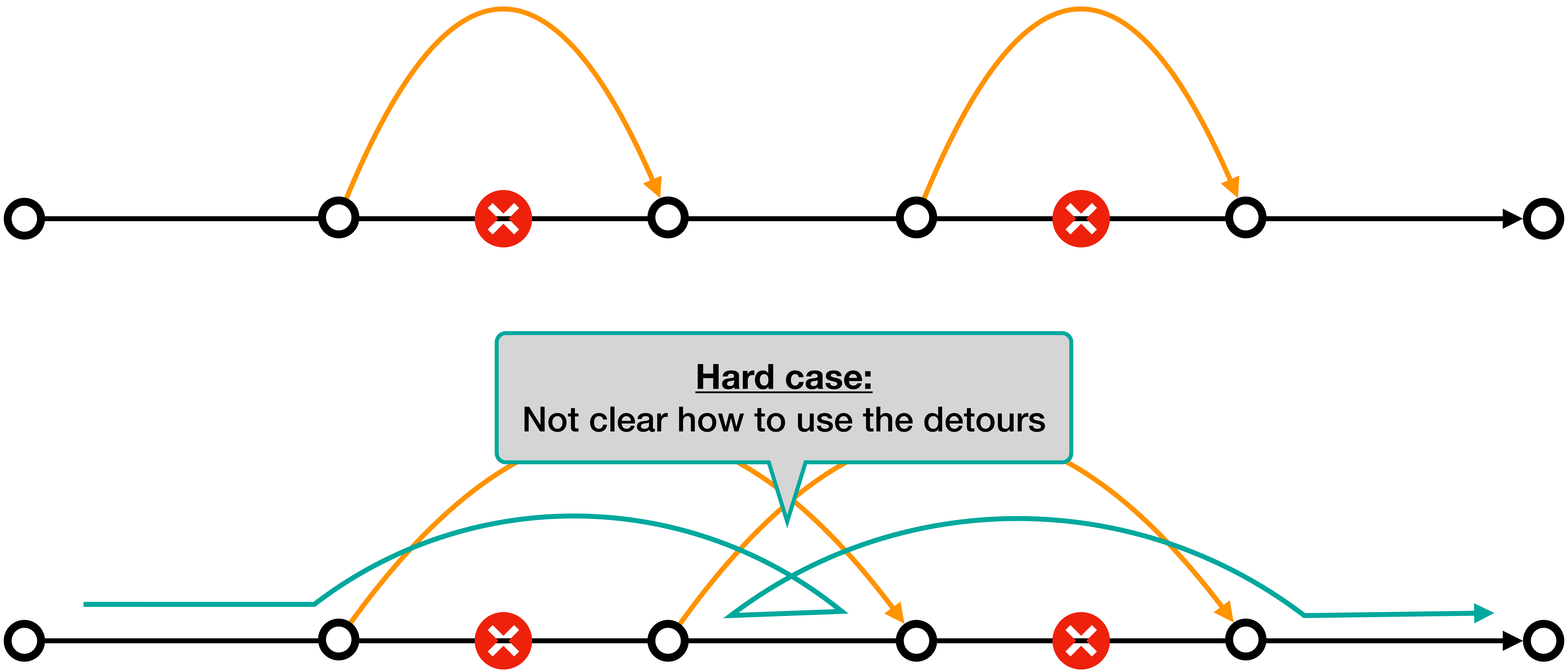


<u>Easy case:</u>
Using both detours is a good dual-failure replacement paths

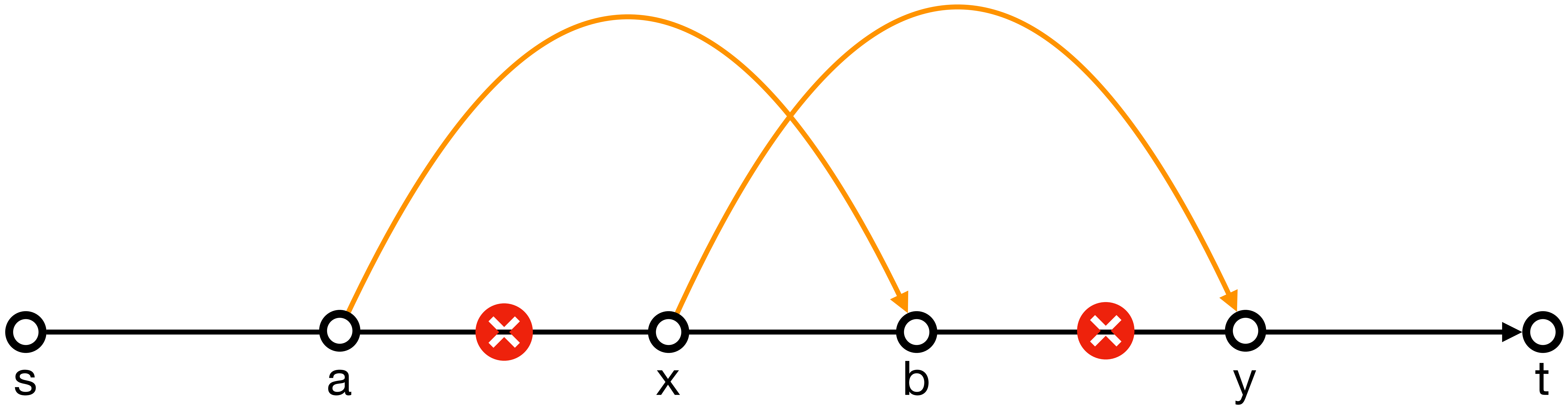
Easy & Hard Cases



Easy & Hard Cases

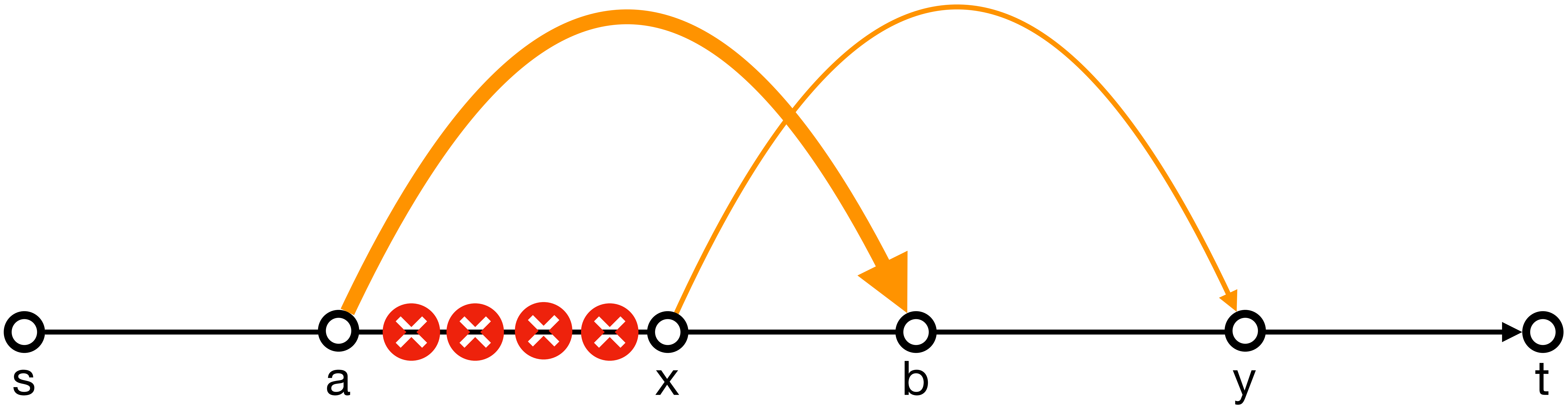


A Simplified Setting



Two overlapping detours, two failures on two intervals

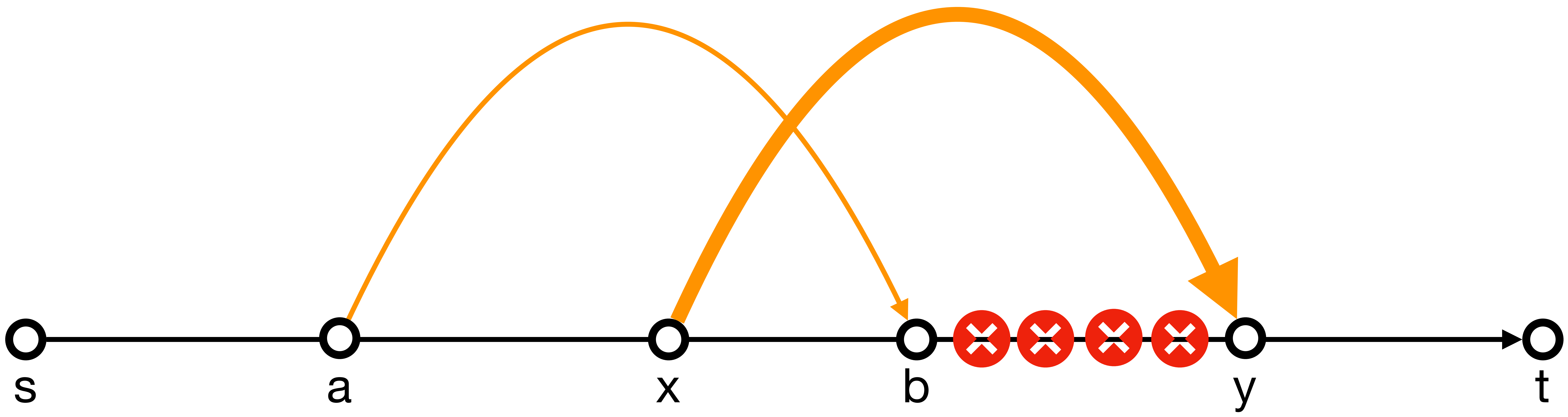
A Simplified Setting



Simplifying assumption:

All failures in the interval share the same single-failure replacement path

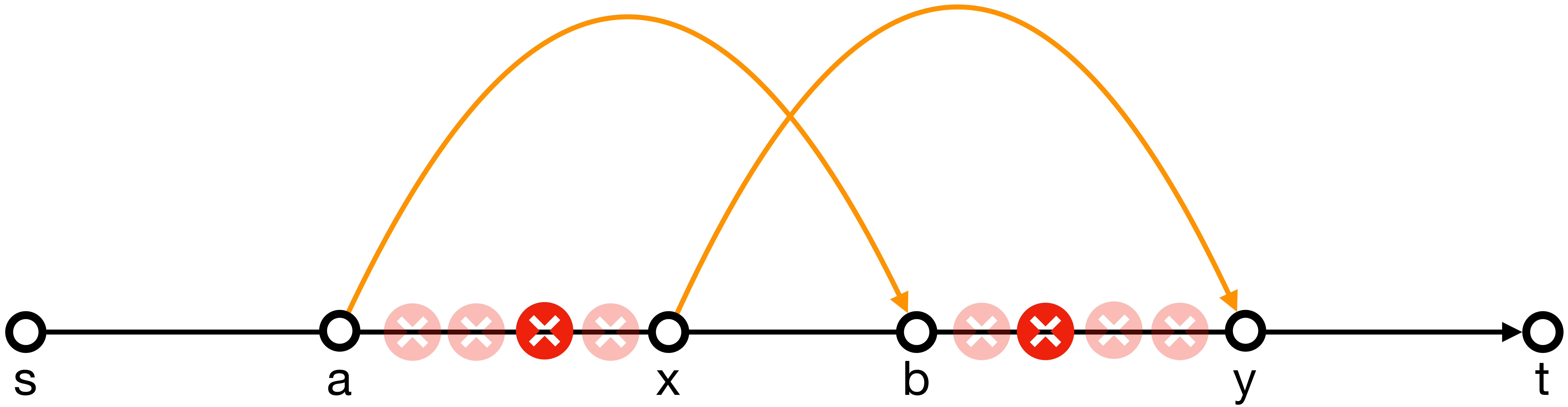
A Simplified Setting



<u>Simplifying assumption:</u>

All failures in the interval share the same single-failure replacement path

A Simplified Setting



<u>Simplified goal:</u>

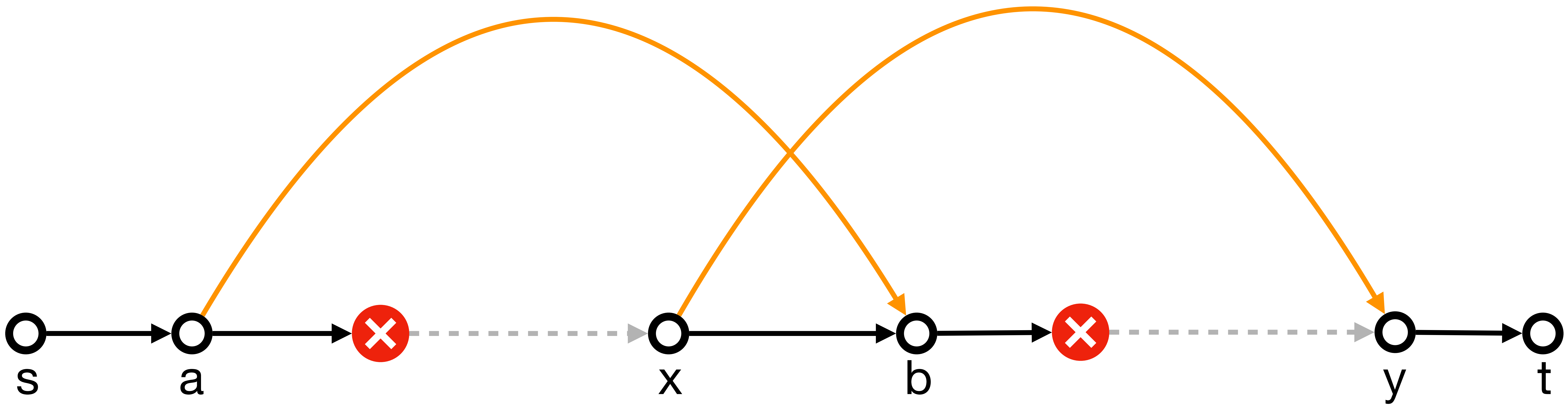
Enumerate **pairs** of failures in both intervals, and compute dual-failure **RP**

Technical difficulties

Main issue with progressive Dijkstra:

Impossible to **order all the pairs** so that the graph is **monotonically growing**

For example, alphabetic order does not work



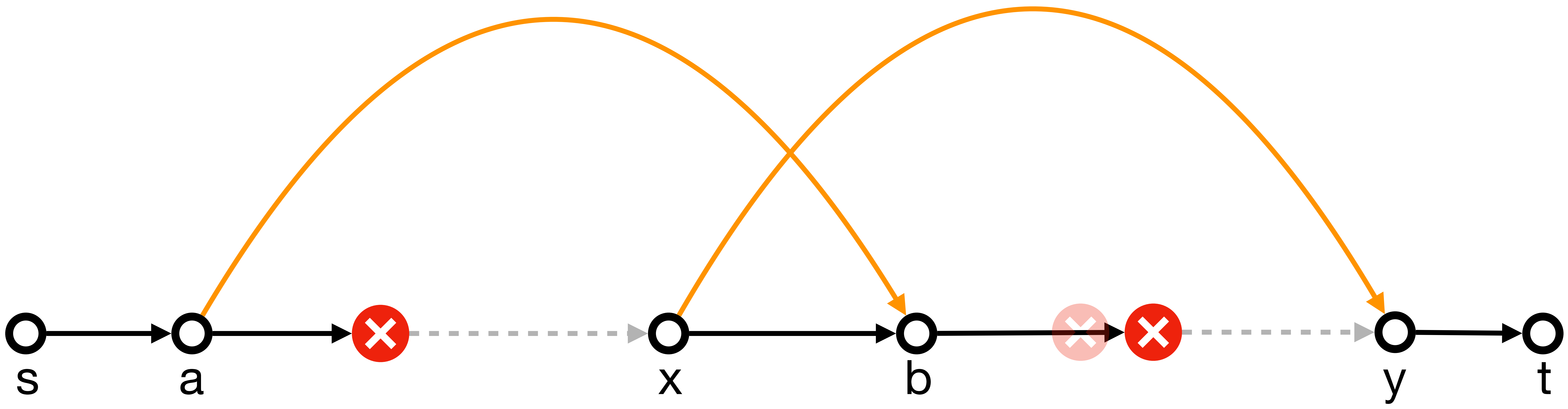
Using an alphabetic order of the failure pairs, the graph is **not monotone**

Technical difficulties

Main issue with progressive Dijkstra:

Impossible to **order all the pairs** so that the graph is **monotonically growing**

For example, alphabetic order does not work



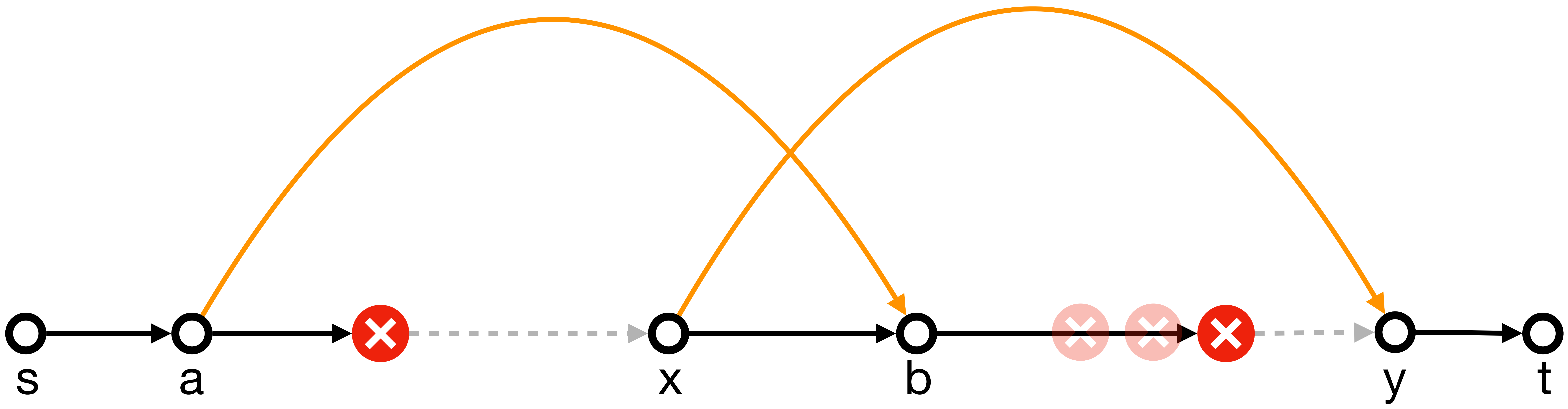
Using an alphabetic order of the failure pairs, the graph is **not monotone**

Technical difficulties

Main issue with progressive Dijkstra:

Impossible to **order all the pairs** so that the graph is **monotonically growing**

For example, alphabetic order does not work



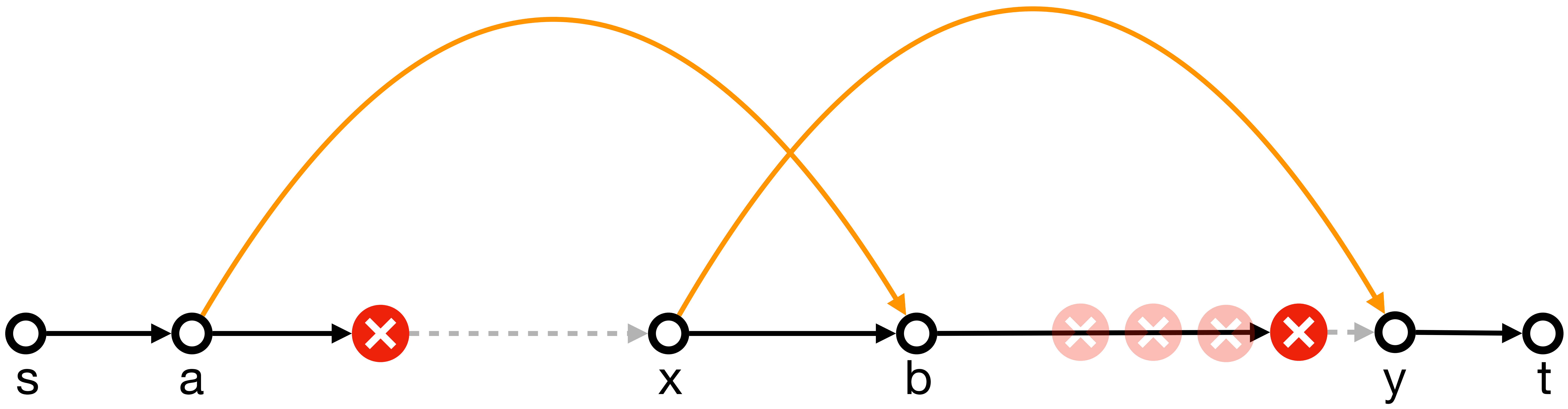
Using an alphabetic order of the failure pairs, the graph is **not monotone**

Technical difficulties

Main issue with progressive Dijkstra:

Impossible to **order all the pairs** so that the graph is **monotonically growing**

For example, alphabetic order does not work



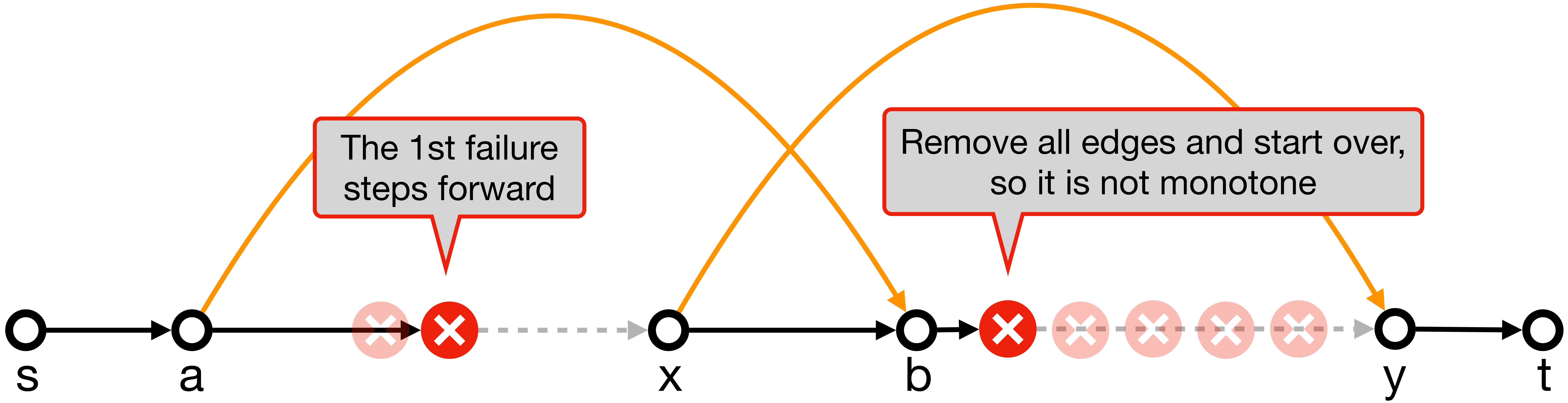
Using an alphabetic order of the failure pairs, the graph is **not monotone**

Technical difficulties

<u>Main issue with progressive Dijkstra:</u>

Impossible to **order all the pairs** so that the graph is **monotonically growing**

For example, alphabetic order does not work

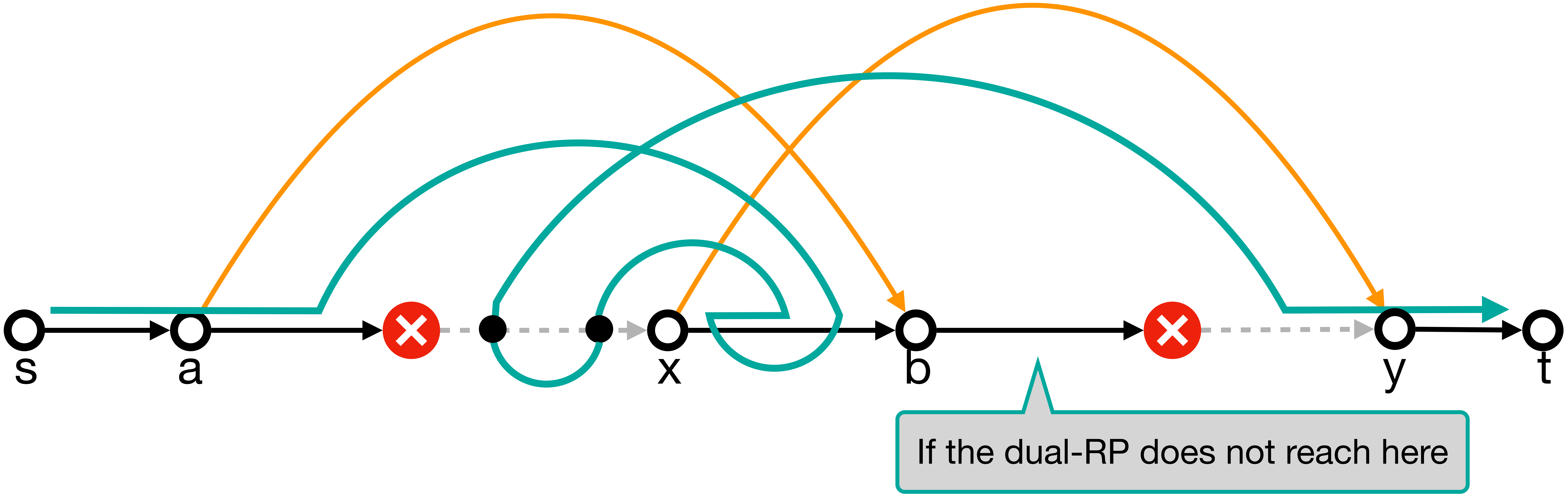


Using an alphabetic order of the failure pairs, the graph is **not monotone**

Decoupling dual-failures

Main issue with progressive Dijkstra:
Impossible to **order all the pairs** so that the graph is **monotonically growing**

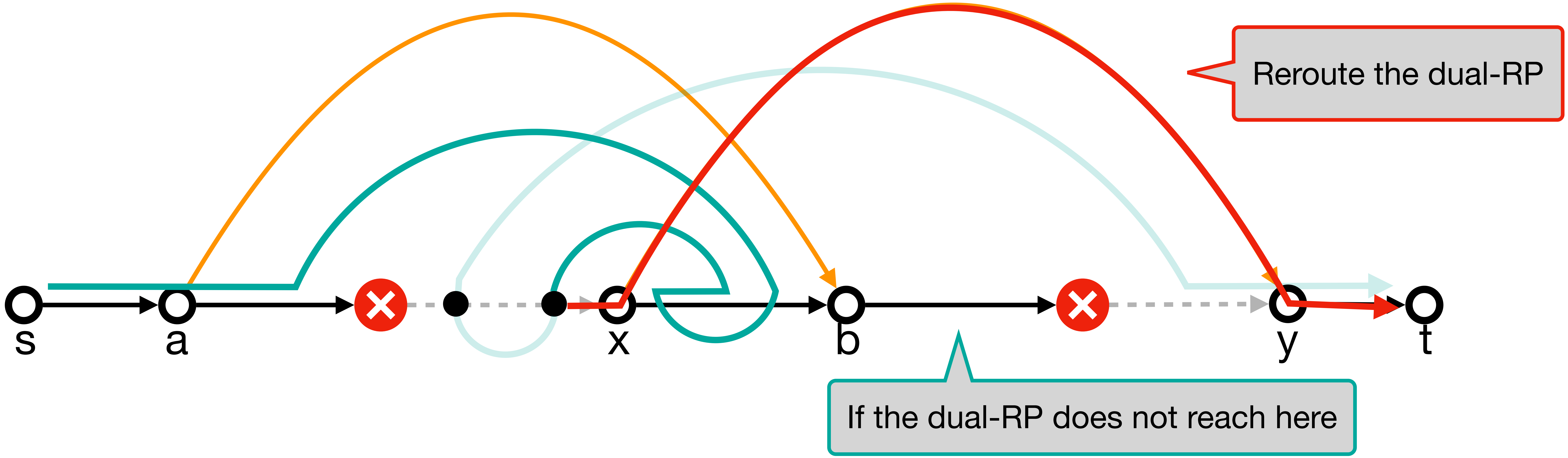
Solution: **Decouple** the two failures and use progressive Dijkstra separately



Decoupling dual-failures

Main issue with progressive Dijkstra:
Impossible to **order all the pairs** so that the graph is **monotonically growing**

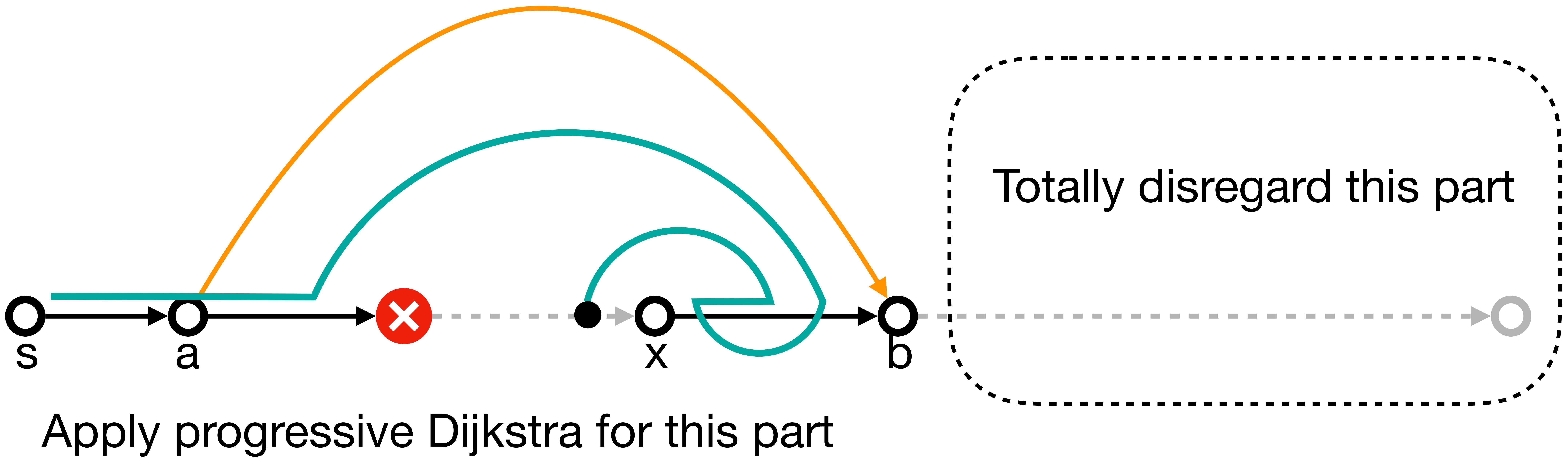
Solution: **Decouple** the two failures and use progressive Dijkstra separately



Decoupling dual-failures

Main issue with progressive Dijkstra:
Impossible to **order all the pairs** so that the graph is **monotonically growing**

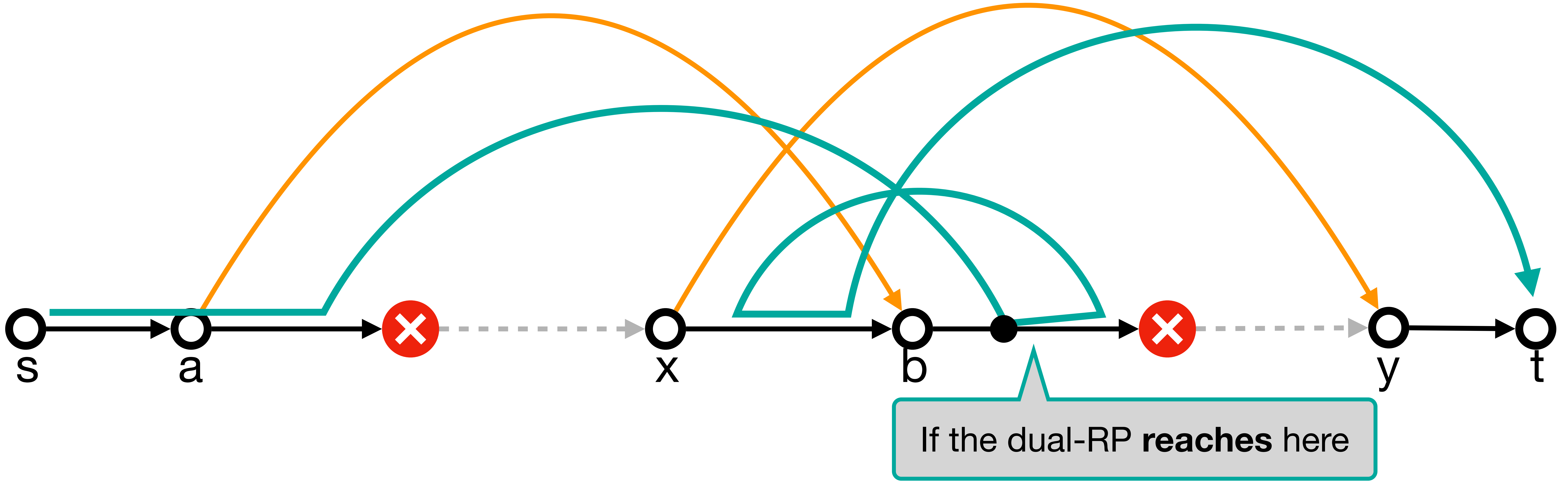
Solution: **Decouple** the two failures and use progressive Dijkstra separately



Decoupling dual-failures

<u>Main issue with progressive Dijkstra:</u>
Impossible to **order all the pairs** so that the graph is **monotonically growing**

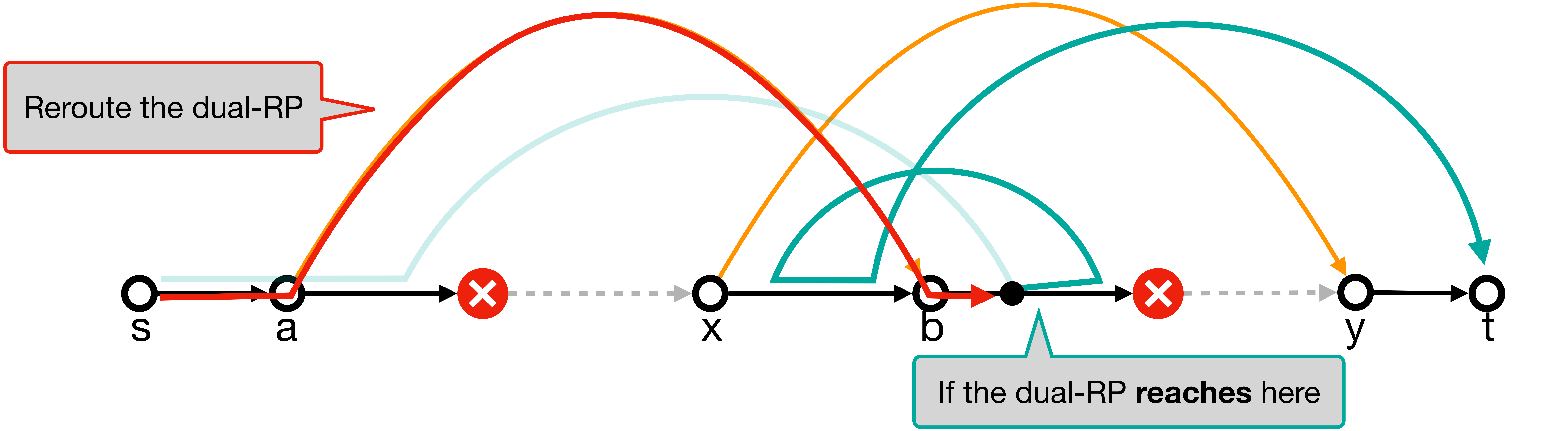
<u>Solution:</u> **Decouple** the two failures and use progressive Dijkstra separately



Decoupling dual-failures

Main issue with progressive Dijkstra:
Impossible to **order all the pairs** so that the graph is **monotonically growing**

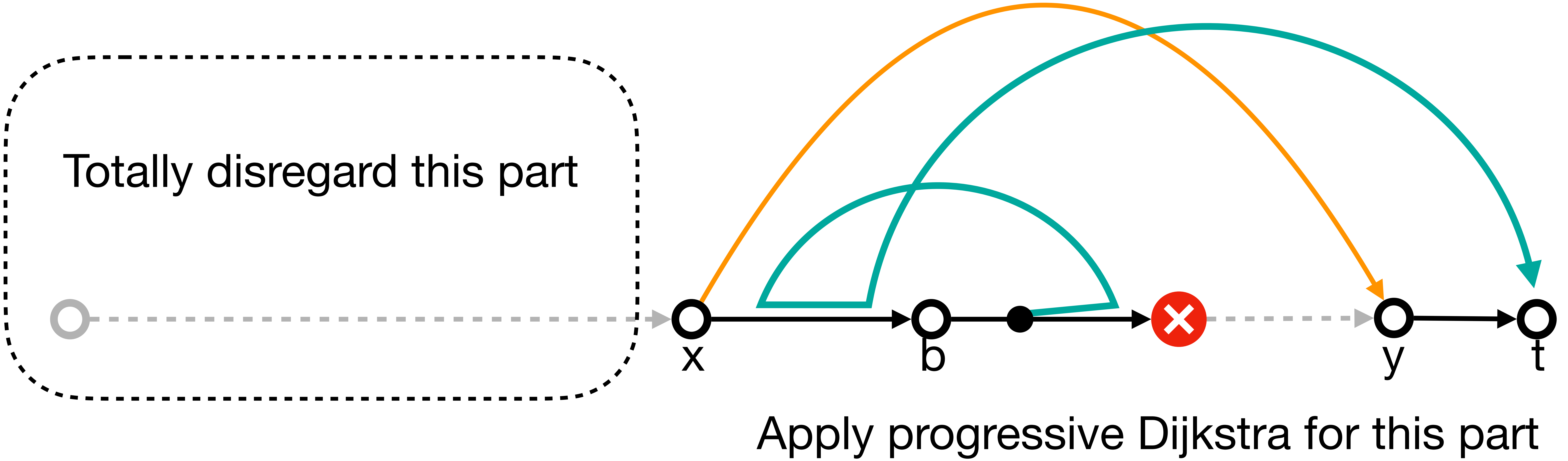
Solution: Decouple the two failures and use progressive Dijkstra separately



Decoupling dual-failures

Main issue with progressive Dijkstra:
Impossible to **order all the pairs** so that the graph is **monotonically growing**

Solution: **Decouple** the two failures and use progressive Dijkstra separately



Conclusion

Conclusion

- Quadratic time for approximate dual-failure st-shortest paths
- How about approximate **single-source** RP?
- Approximate single-failure single-source RP in linear time?