# Incremental Single Source Shortest Paths in Sparse Digraphs

Shiri Chechik
Tel Aviv University

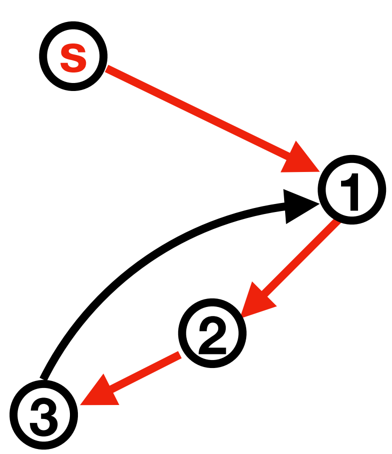
Tianyi Zhang
Tsinghua University

# Partially dynamic SSSP

A weighted digraph G = (V, E) undergoes edge updates

- Decremental: all updates are deletions
- Incremental: all updates are insertions

**Goal.** Answer queries of distances from a source vertex $s \in V$
**Cost.** Total update time

| | | | | |
|---|---|---|---|---|
| **Edge insertions** | | | | |
| **Picture** |  | | | |

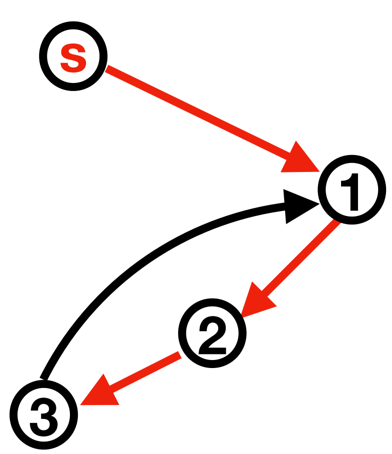
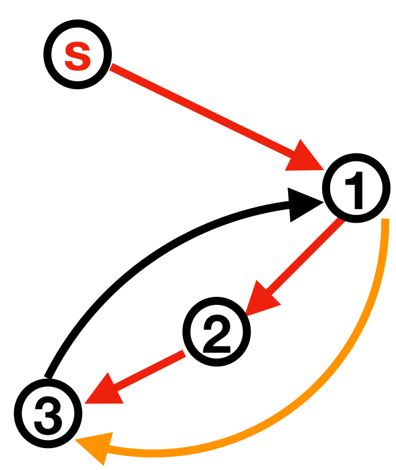
# Partially dynamic SSSP

A weighted digraph G = (V, E) undergoes edge updates

- Decremental: all updates are deletions
- Incremental: all updates are insertions

**Goal.** Answer queries of distances from a source vertex $s \in V$

**Cost.** Total update time

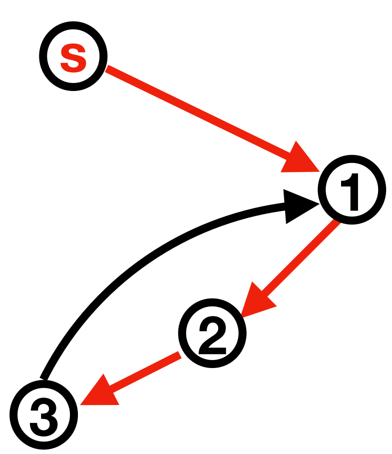
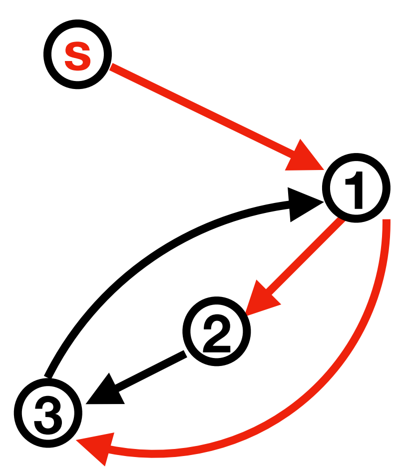
| | | | | |
|---|---|---|---|---|
| **Edge insertions** | | Insert (1, 3) | | |
| **Picture** |  |  | | |

# Partially dynamic SSSP

A weighted digraph G = (V, E) undergoes edge updates

- Decremental: all updates are deletions
- Incremental: all updates are insertions

**Goal.** Answer queries of distances from a source vertex $s \in V$
**Cost.** Total update time

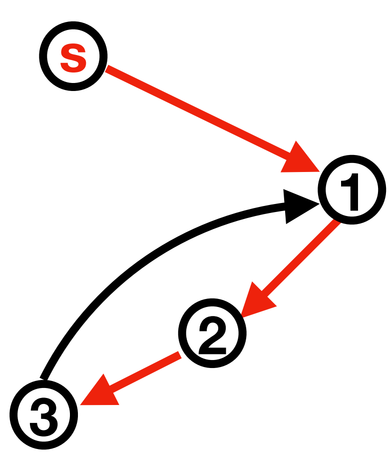
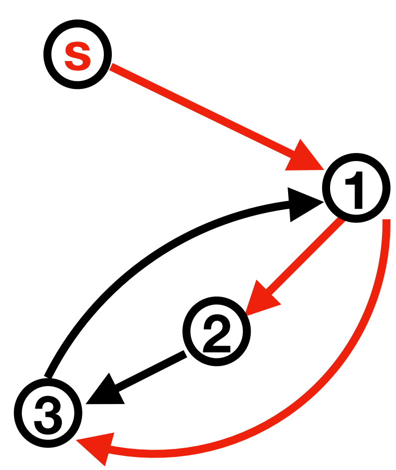
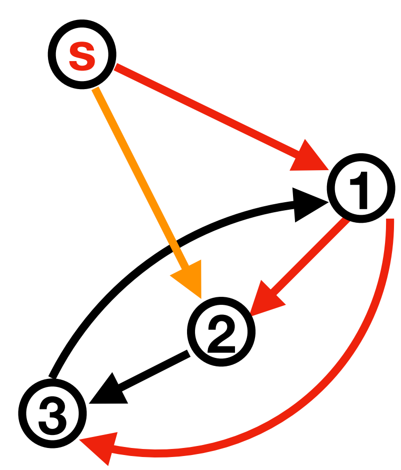
| | | | | |
|---|---|---|---|---|
| **Edge insertions** | | Insert (1, 3) | | |
| **Picture** |  |  | | |

# Partially dynamic SSSP

A weighted digraph G = (V, E) undergoes edge updates

- Decremental: all updates are deletions
- Incremental: all updates are insertions

**Goal.** Answer queries of distances from a source vertex $s \in V$
**Cost.** Total update time

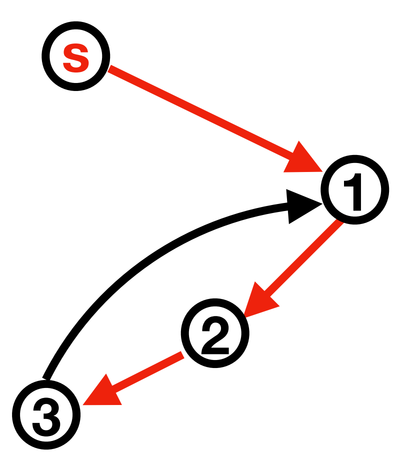
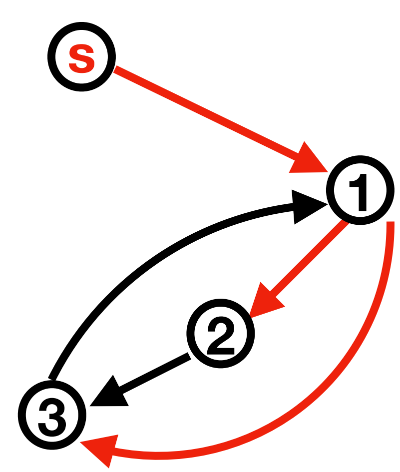
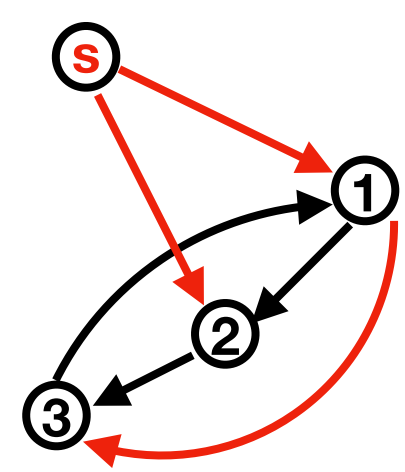
| **Edge insertions** | | Insert (1, 3) | Insert (s, 2) | |
|---|---|---|---|---|
| **Picture** |  |  |  | |

# Partially dynamic SSSP

A weighted digraph G = (V, E) undergoes edge updates

- Decremental: all updates are deletions
- Incremental: all updates are insertions

**Goal.** Answer queries of distances from a source vertex $s \in V$
**Cost.** Total update time

| **Edge insertions** | | Insert (1, 3) | Insert (s, 2) | |
|---|---|---|---|---|
| **Picture** |  |  |  | |

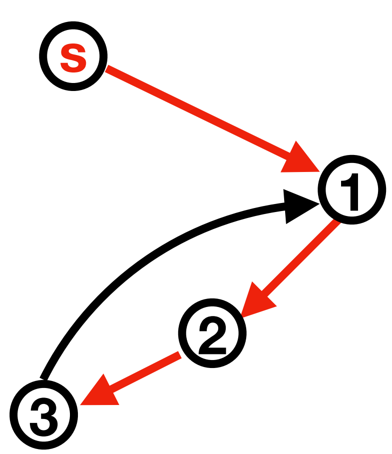
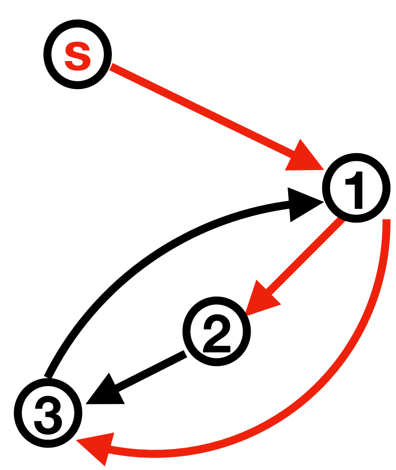
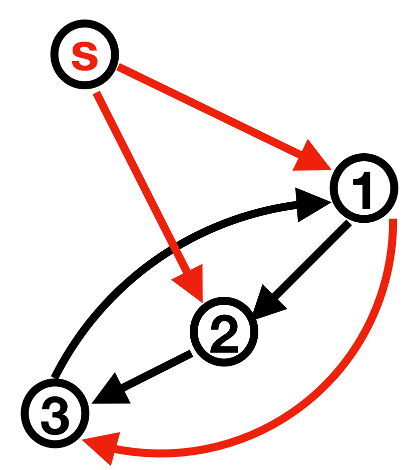
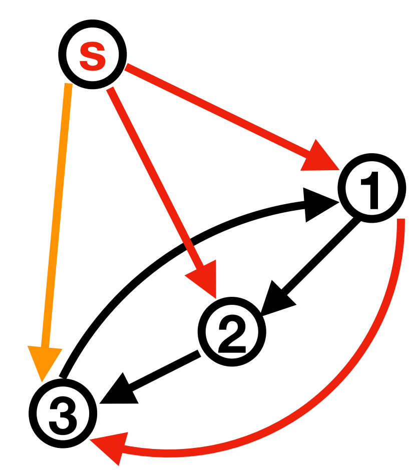
# Partially dynamic SSSP

A weighted digraph G = (V, E) undergoes edge updates

- Decremental: all updates are deletions
- Incremental: all updates are insertions

**Goal.** Answer queries of distances from a source vertex $s \in V$

**Cost.** Total update time

| **Edge insertions** | | Insert (1, 3) | Insert (s, 2) | Insert (s, 3) |
|---|---|---|---|---|
| **Picture** |  |  |  |  |

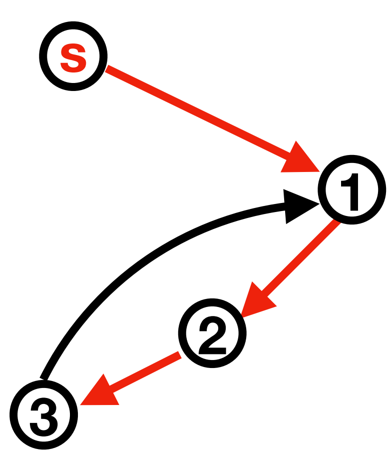
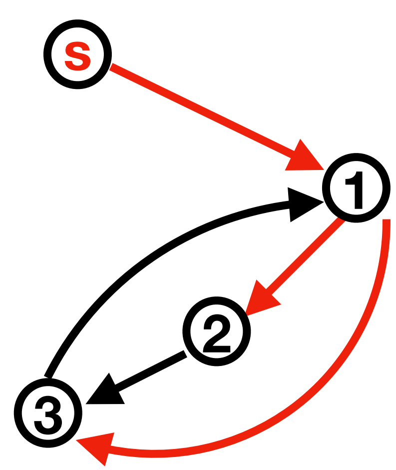
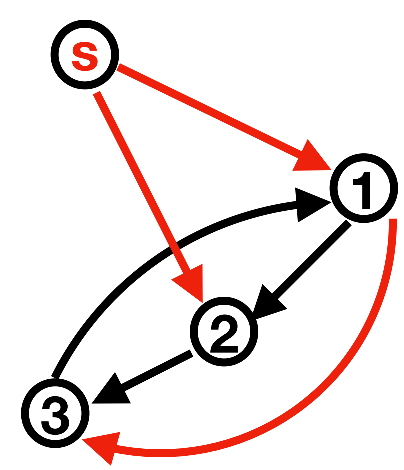
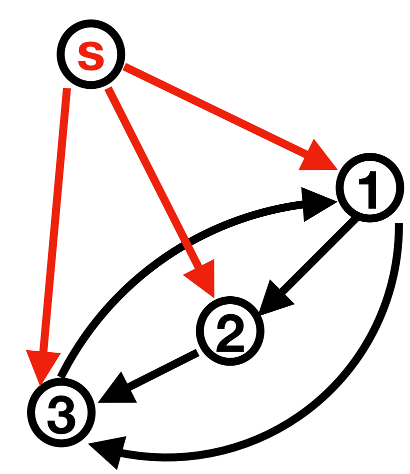
# Partially dynamic SSSP

A weighted digraph G = (V, E) undergoes edge updates

- Decremental: all updates are deletions
- Incremental: all updates are insertions

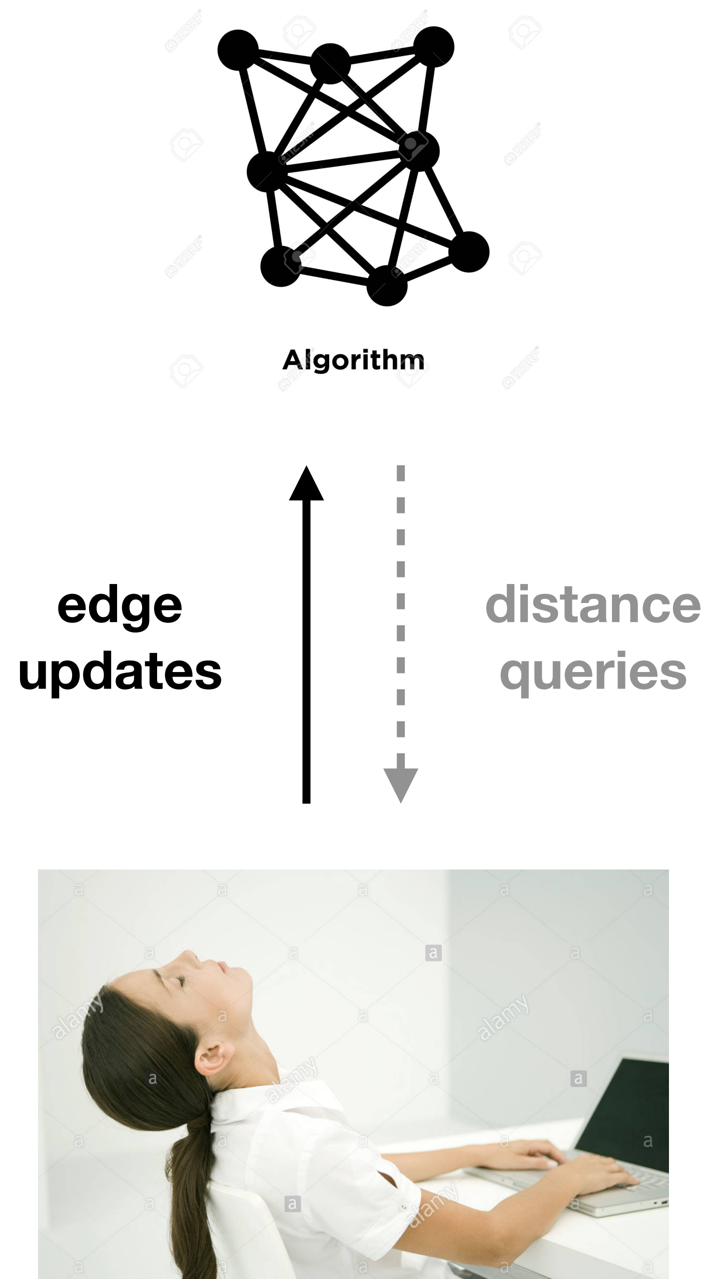
**Goal.** Answer queries of distances from a source vertex $s \in V$

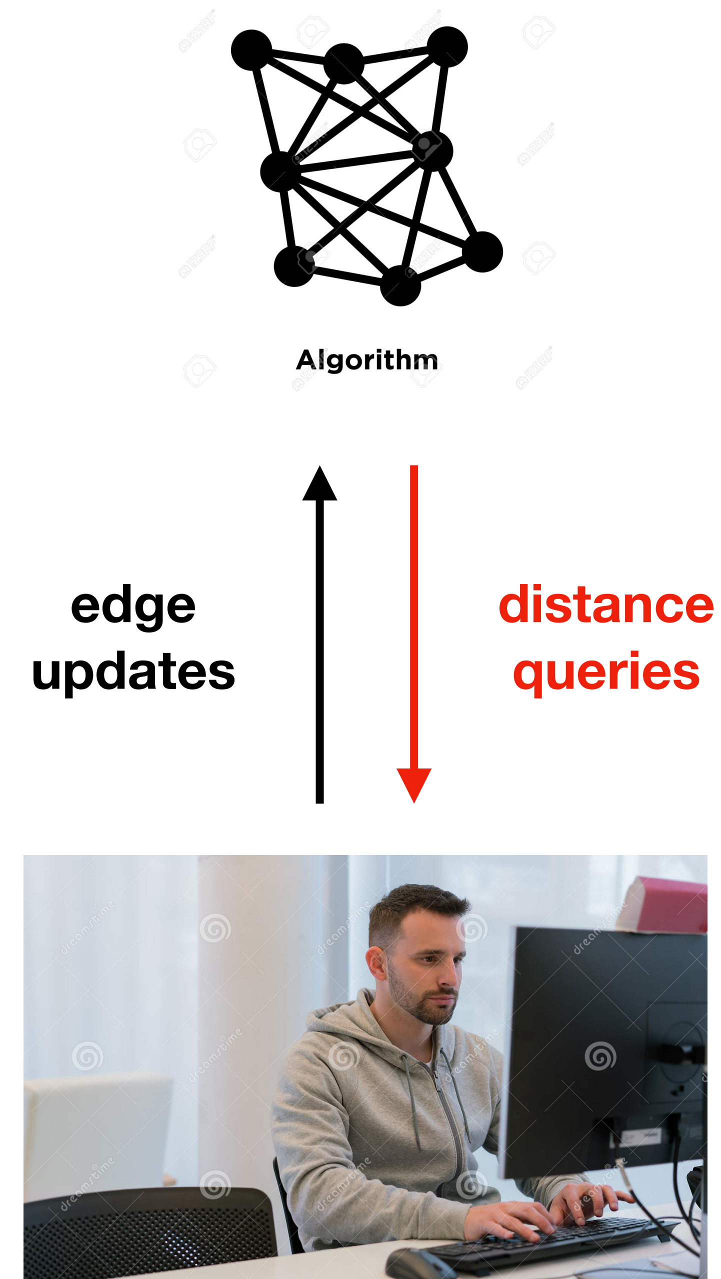
**Cost.** Total update time

| **Edge insertions** | | Insert (1, 3) | Insert (s, 2) | Insert (s, 3) |
|---|---|---|---|---|
| **Picture** |  |  |  |  |

# Oblivious vs Adaptive

- Oblivious: edge updates are fixed at the beginning
- Adaptive: future edge updates may depend on queries



# History

**Exact** distances in partially dynamic SSSP (either decr or incr)

| | | |
|---|---|---|
| Classic | $O(mn)$ (W=1) | [ES'81] |
| APSP-hard | $\tilde{\Omega}(mn)$ | [RZ'04] |
| k-cycle-hard | $\tilde{\Omega}(m^2)$ | [PWW'20] |
| OMv3-hard | $\tilde{\Omega}(m^{(\omega+1)/2})$ | [PWW'20] |

To break $O(mn)$, should consider $(1+\epsilon)$**-approximation**

Assume **digraph** G has **n** vertices and **m** edges ever appear in the graph

$\tilde{O}(\cdot)$ hides **poly-log(nW)** factors, where **W** is the largest integer weight

# History

To break $O(mn)$, should consider $(1+\epsilon)$**-approximation**

Decr-SSSP is a subroutine in many static algorithms,
e.g. max-flow, sparsest cut

| | | |
|---|---|---|
| Best oblivious | $\tilde{O}(n^2), \tilde{O}(mn^{2/3})$ | [BPW'20] |
| Best adaptive | $\tilde{O}(m^{3/4}n^{5/4})$ | [PW'20] |
| Best deterministic | $n^{8/3+o(1)}$ | [BPS'20] |

Incr-SSSP is a natural sister problem of Decr-SSSP

| | | |
|---|---|---|
| Best oblivious | $\tilde{O}(mn^{0.9})$ | [HKN'14] |
| Best deterministic | $\tilde{O}(n^2)$ | [PWW'20] |

Assume **digraph** G has **n** vertices and **m** edges ever appear in the graph

$\tilde{O}(\cdot)$ hides **poly-log(nW)** factors, where **W** is the largest integer weight

# Results

| Reference | Total update time | det / obl / ada |
|---|---|---|
| [HKN'14] | $\tilde{O}(mn^{0.9})$ | oblivious |
| [PWW'20] | $\tilde{O}(n^2)$ | deterministic |
| New | $\tilde{O}(m^{5/3})$ | deterministic |
| New | $\tilde{O}(mn^{1/2} + m^{1.4})$ | adaptive |

Our algorithm is the **sub-quadratic** when $m = o(n^{1.42})$

# A deterministic algorithm

# A basic procedure

- Similar to Dijkstra's algorithm, but in a local & lazy manner

maintain dist labels $d(\cdot)$ for each $v \in V$

**Propagate**( $Q$ ):

while($Q \neq \emptyset$)

$u \leftarrow$ dequeue $Q$

for each $(u, v) \in E$

if $d(v) - d(u) - \omega(u, v) \geq D$ **or** $v \in Q$

$d(v) \leftarrow \min\{d(u) + \omega(u, v), d(v)\}$

$Q \leftarrow Q \cup \{v\}$

**Dijkstra**:

initialize dist labels $d(\cdot)$ for each $v \in V$

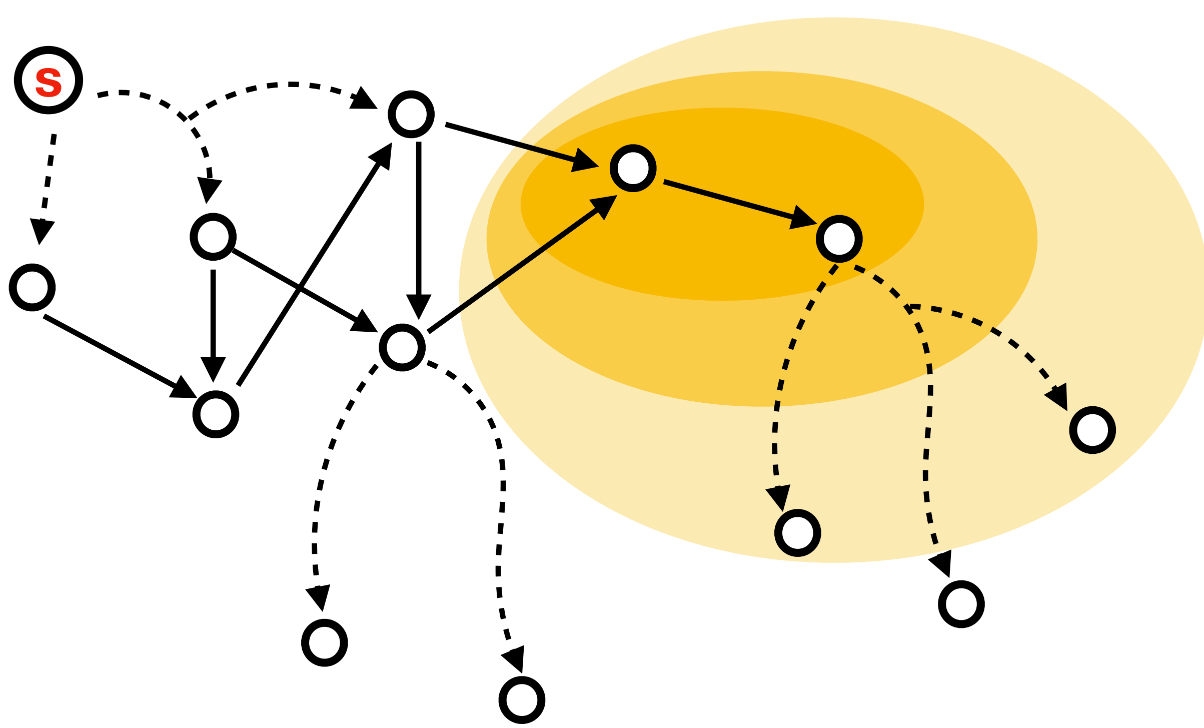
initialize a queue $Q \leftarrow V$

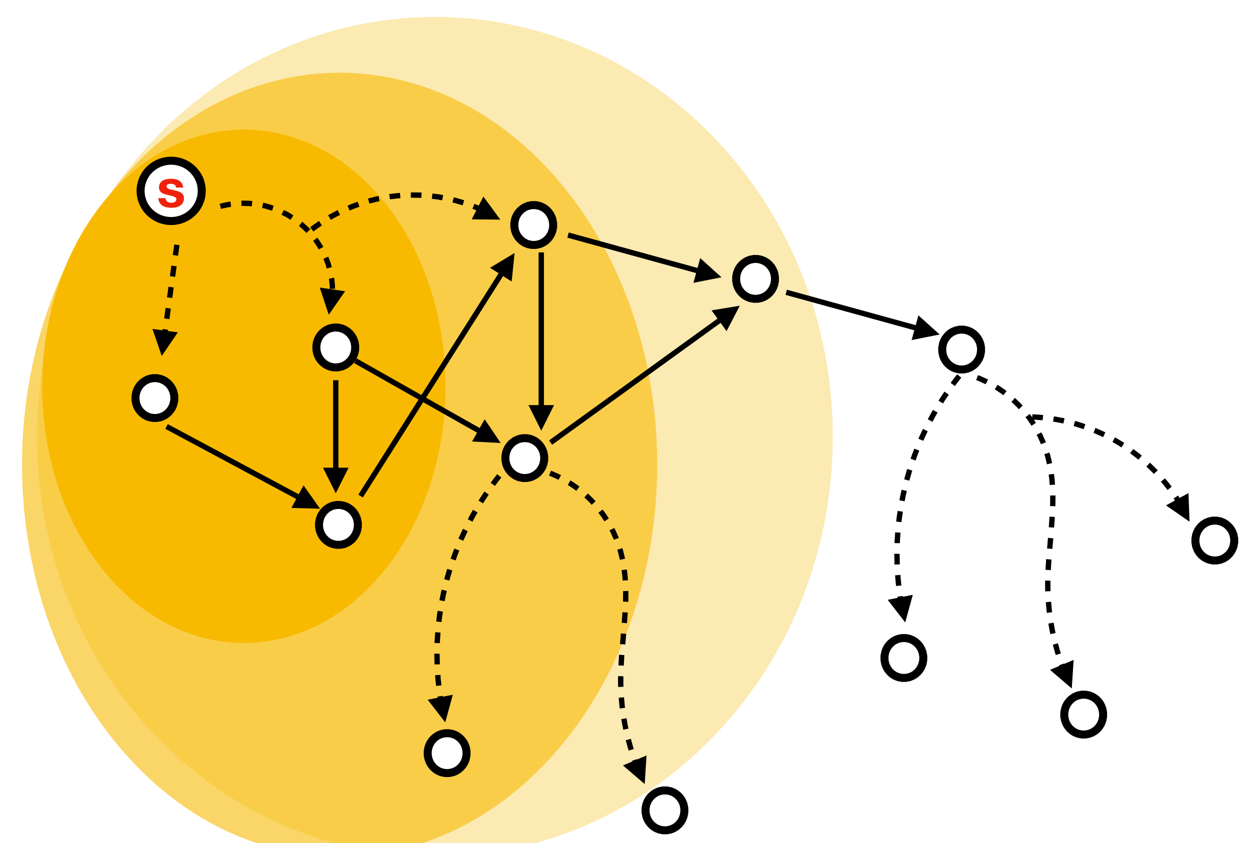
while($Q \neq \emptyset$)

$u \leftarrow$ dequeue $Q$

For each $(u, v) \in E$

$d(v) \leftarrow \min\{d(u) + \omega(u, v), d(v)\}$



# A basic procedure

- Similar to Dijkstra's algorithm, but in a local & lazy manner

maintain dist labels $d(\cdot)$ for each $v \in V$

**Propagate**( $Q$ ):

  while($Q \neq \emptyset$)

    $u \leftarrow$ dequeue $Q$

    for each $(u, v) \in E$

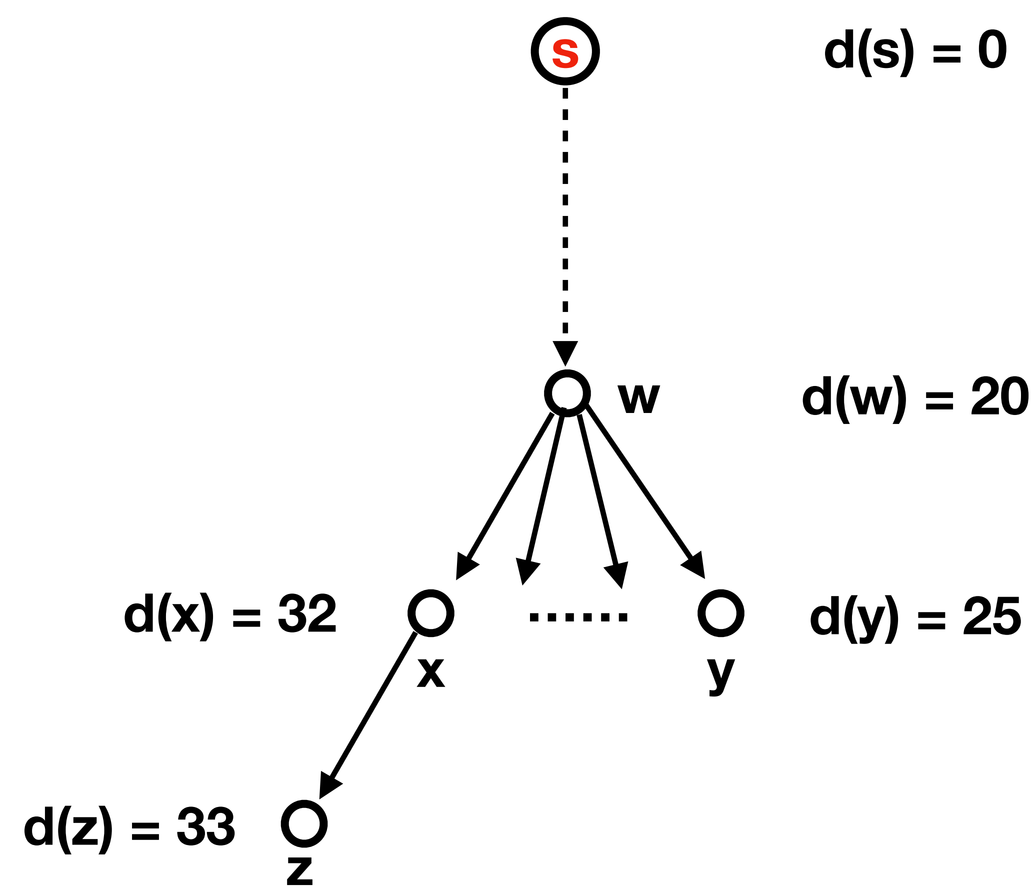
      if $d(v) - d(u) - \omega(u, v) \geq D$ **or** $v \in Q$

        $d(v) \leftarrow \min\{d(u) + \omega(u, v), d(v)\}$

        $Q \leftarrow Q \cup \{v\}$


An example of **Propagate**

Parameters: D = 10, Q = {w}



# A basic procedure

- Similar to Dijkstra's algorithm, but in a local & lazy manner

maintain dist labels $d(\cdot)$ for each $v \in V$

**Propagate**( $Q$ ):

while($Q \neq \emptyset$)

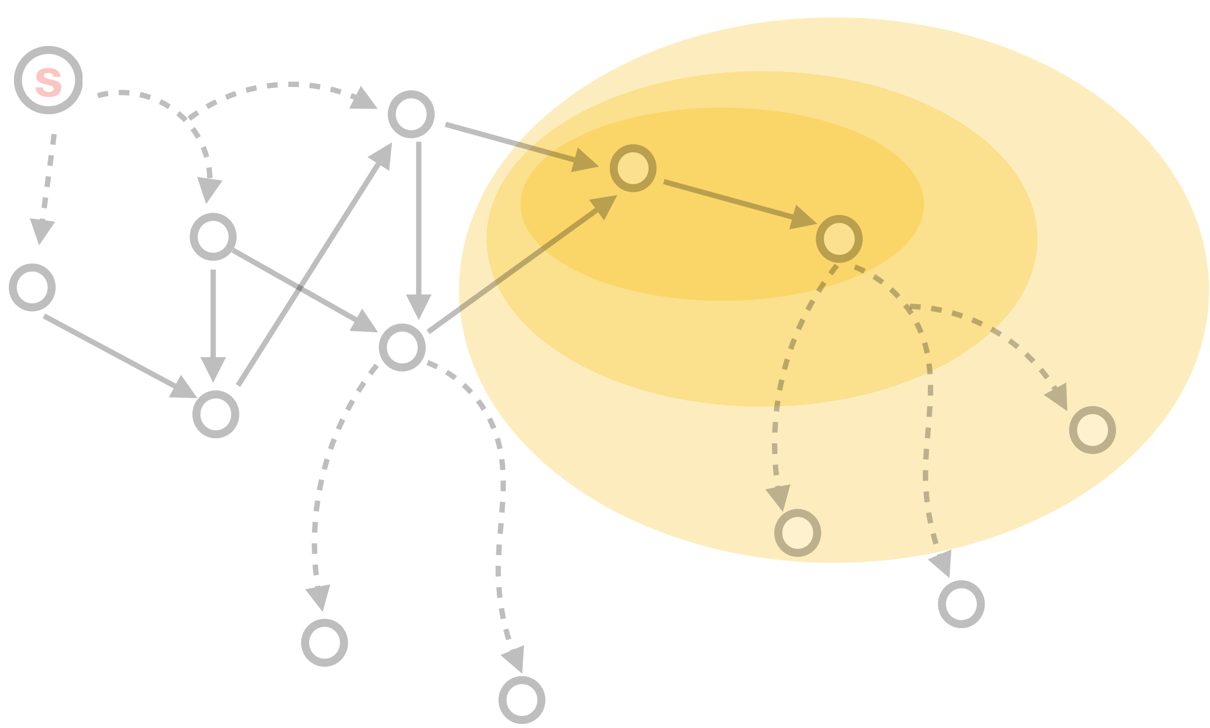
$u \leftarrow$ dequeue $Q$

for each $(u, v) \in E$

if $d(v) - d(u) - \omega(u, v) \geq D$ **or** $v \in Q$

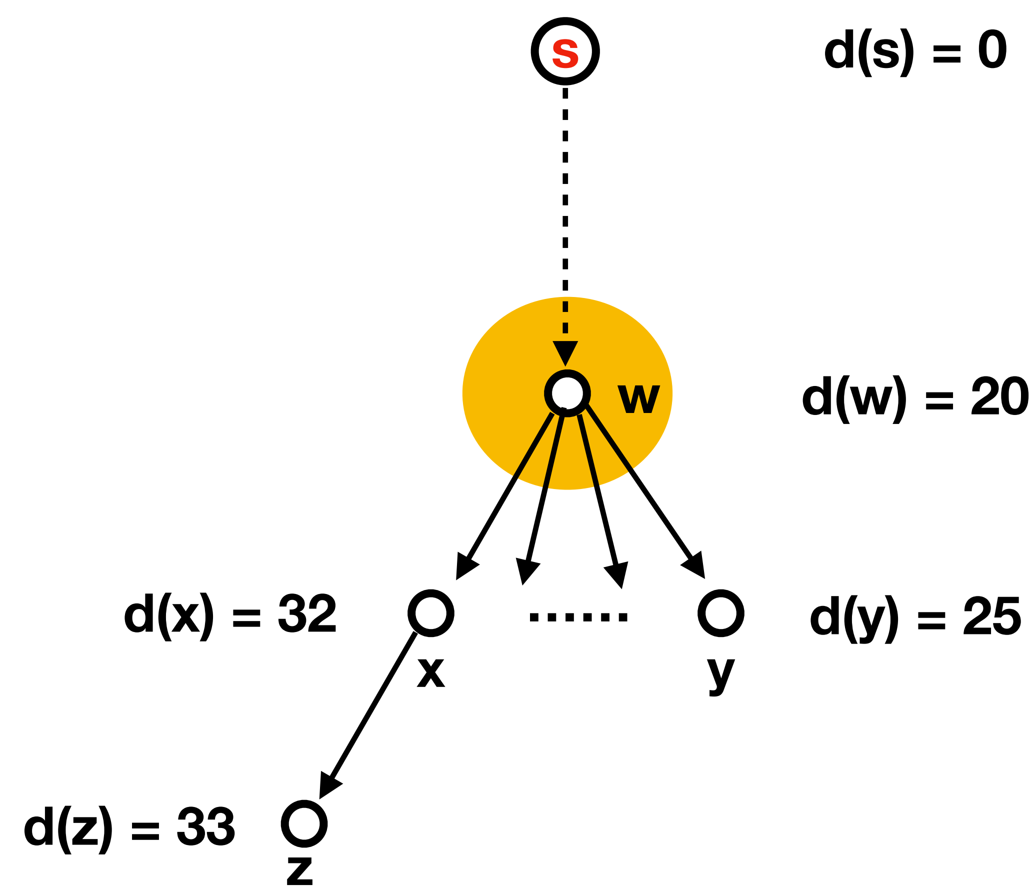
$d(v) \leftarrow \min\{d(u) + \omega(u, v), d(v)\}$

$Q \leftarrow Q \cup \{v\}$




An example of **Propagate**

Parameters: D = 10, Q = {w}



# A basic procedure

- Similar to Dijkstra's algorithm, but in a local & lazy manner

maintain dist labels $d(\cdot)$ for each $v \in V$

**Propagate**( $Q$ ):

  while($Q \neq \emptyset$)

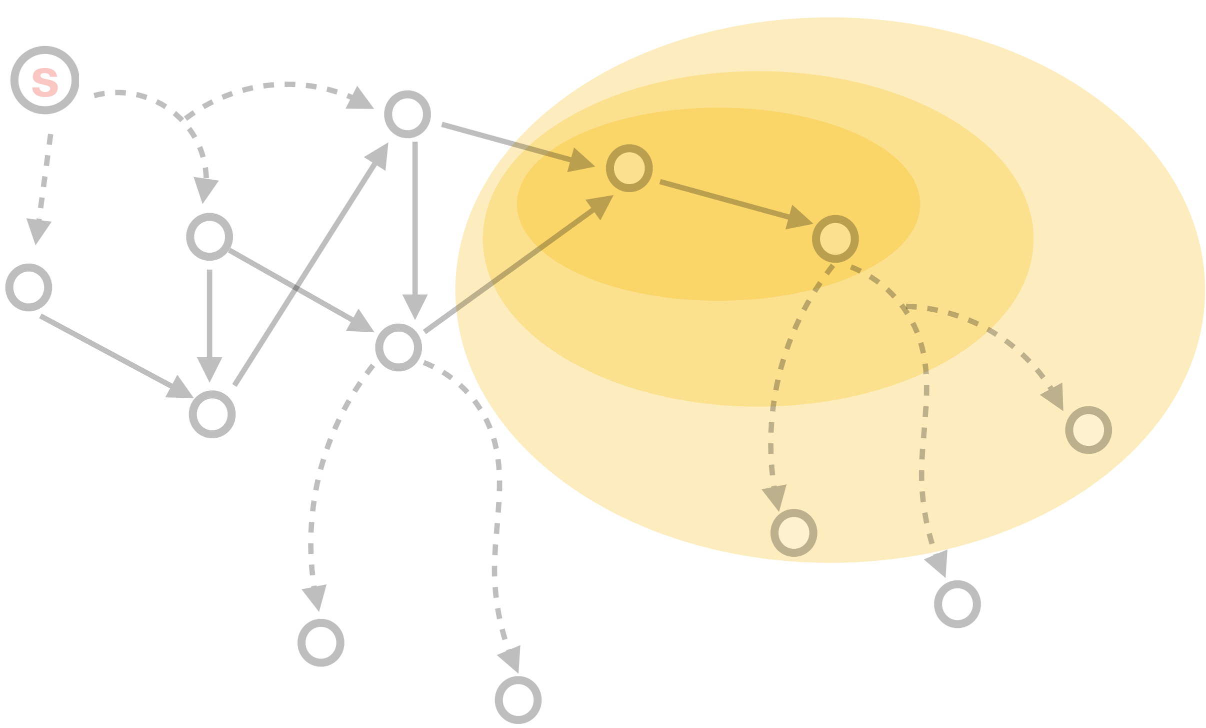
    $u \leftarrow$ dequeue $Q$

    for each $(u, v) \in E$

      if $d(v) - d(u) - \omega(u,v) \geq D$ **or** $v \in Q$

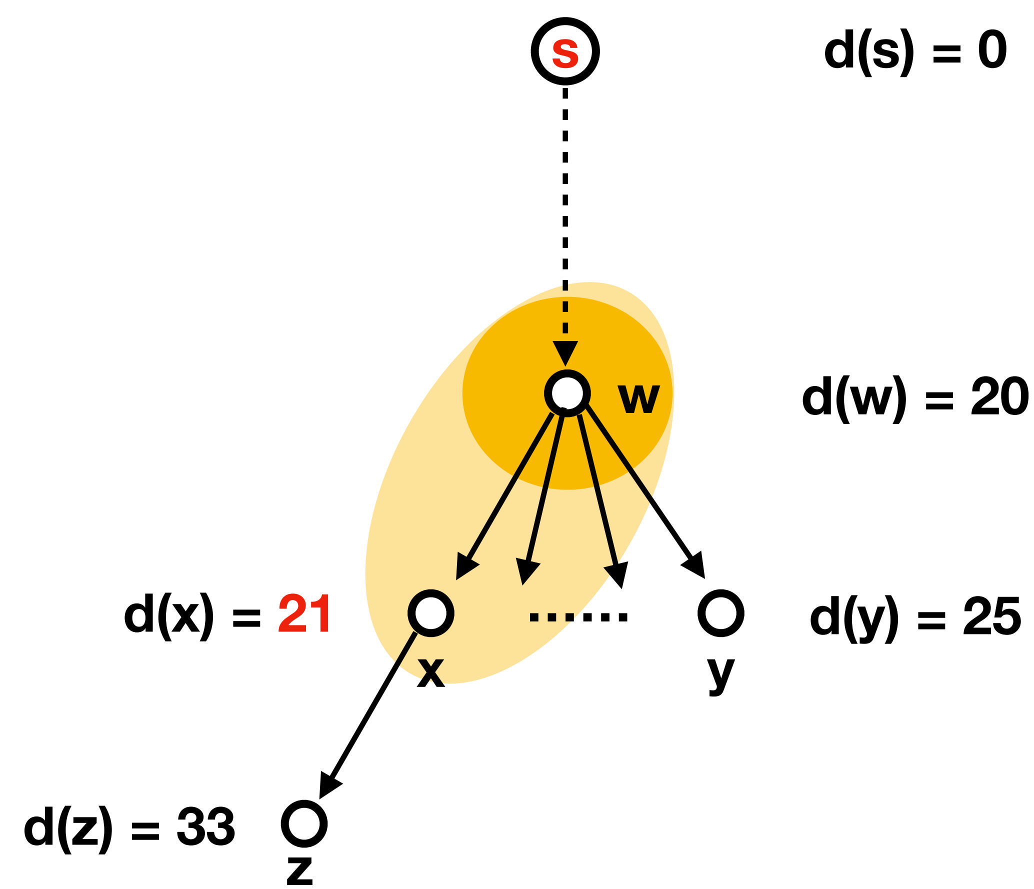
        $d(v) \leftarrow \min\{d(u) + \omega(u,v), d(v)\}$

        $Q \leftarrow Q \cup \{v\}$




An example of **Propagate**

Parameters: D = 10, Q = {w}



# A basic procedure

- Similar to Dijkstra's algorithm, but in a local & lazy manner

maintain dist labels $d(\cdot)$ for each $v \in V$

**Propagate**( $Q$ ):

while($Q \neq \emptyset$)

$u \leftarrow$ dequeue $Q$

for each $(u, v) \in E$

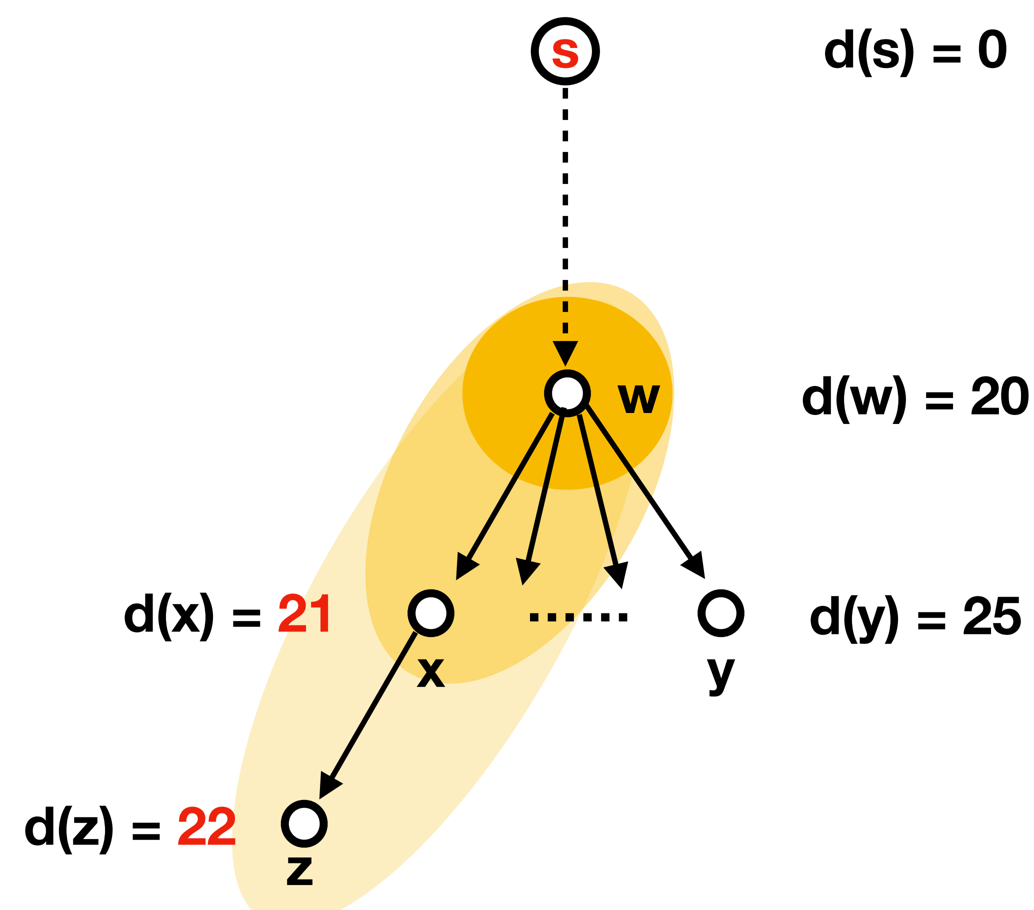
if $d(v) - d(u) - \omega(u, v) \geq D$ **or** $v \in Q$

$d(v) \leftarrow \min\{d(u) + \omega(u, v), d(v)\}$

$Q \leftarrow Q \cup \{v\}$


An example of **Propagate**

Parameters: D = 10, Q = {w}



# A basic procedure

- Similar to Dijkstra's algorithm, but in a local & lazy manner

maintain dist labels $d(\cdot)$ for each $v \in V$

**Propagate**( $Q$ ):

  while($Q \neq \emptyset$)

    $u \leftarrow$ dequeue $Q$

    for each $(u, v) \in E$

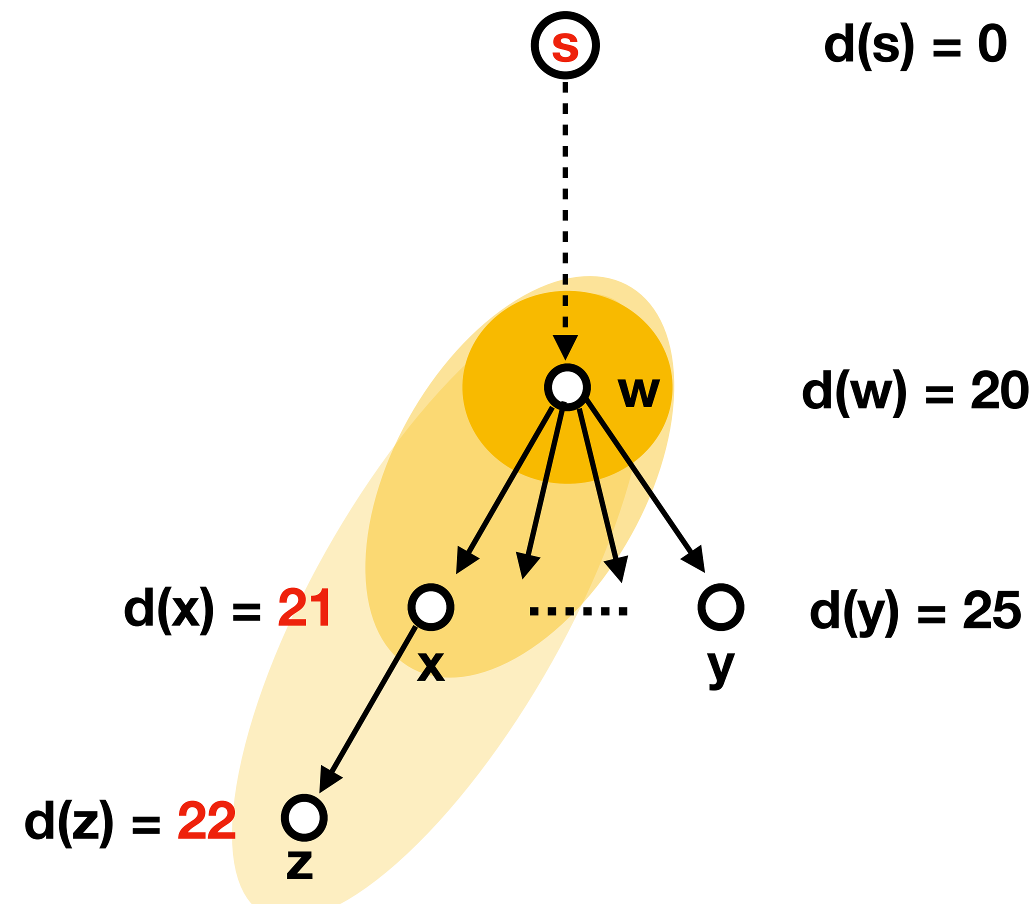
      if $d(v) - d(u) - \omega(u, v) \geq D$ **or** $v \in Q$

        $d(v) \leftarrow \min\{d(u) + \omega(u, v), d(v)\}$

        $Q \leftarrow Q \cup \{v\}$


An example of **Propagate**

Parameters: D = 10, Q = {w}




Running time = **sum of degrees in the queue**

Each **d(u) decreases by D** if u was added to queue

# A deterministic algorithm

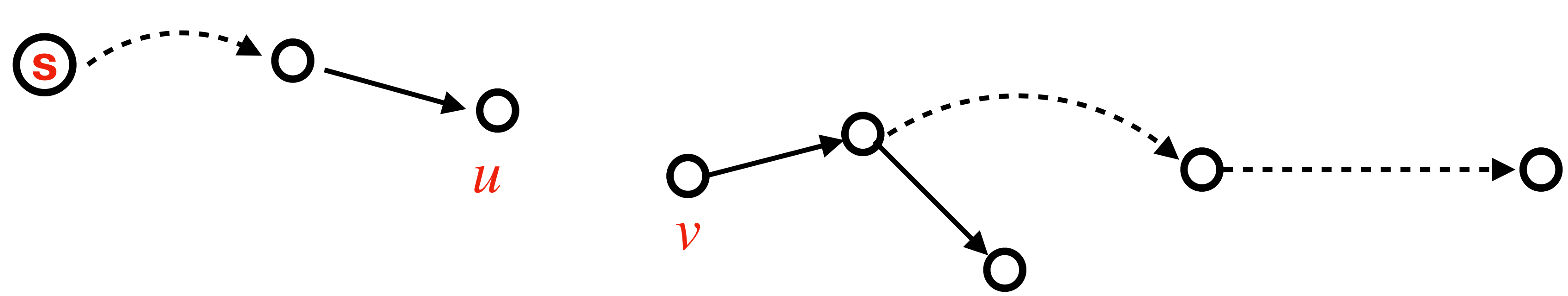
For every **B** insertions: $e_1, e_2, \cdots, e_B$

1. Run **Djikstra** to refresh all distance labels $d(\cdot)$ at the beginning

2. For each insertion $e_i = (u, v)$, if $d(v) - d(u) - \omega(u, v) \geq D$, then

   - update $d(v) \leftarrow d(u) + \omega(u, v)$

   - run **Propagate**( $\{ v \}$ )



# A deterministic algorithm

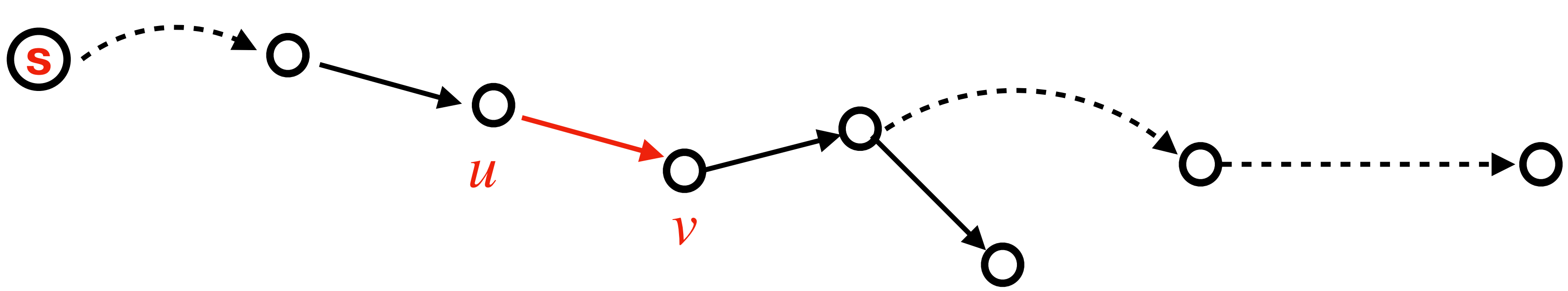
For every **B** insertions: $e_1, e_2, \cdots, e_B$

1. Run **Djikstra** to refresh all distance labels $d(\cdot)$ at the beginning

2. For each insertion $e_i = (u, v)$, if $d(v) - d(u) - \omega(u, v) \geq D$, then

   - update $d(v) \leftarrow d(u) + \omega(u, v)$

   - run **Propagate**( $\{ v \}$ )



# A deterministic algorithm

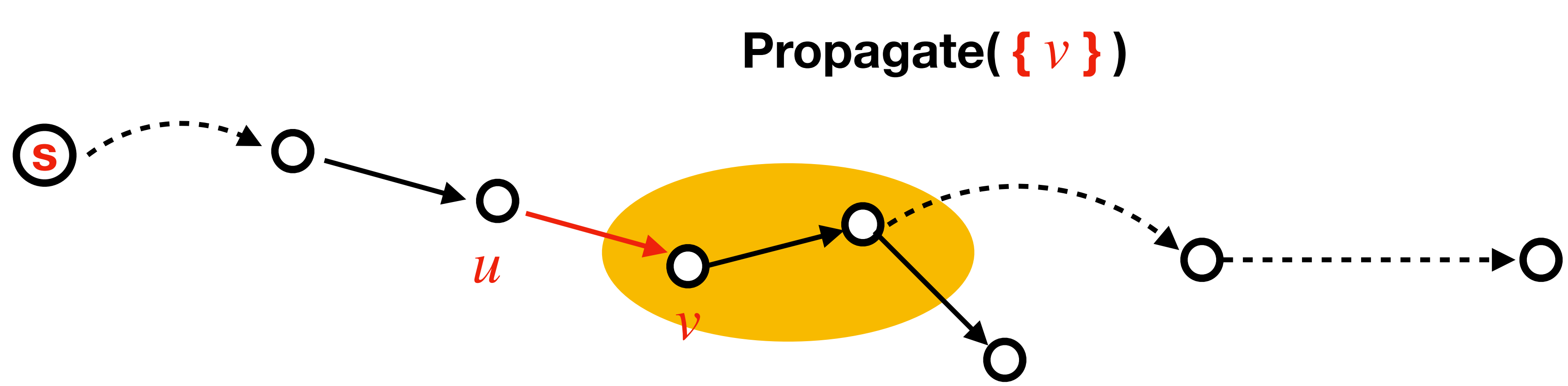
For every **B** insertions: $e_1, e_2, \cdots, e_B$

1. Run **Djikstra** to refresh all distance labels $d(\cdot)$ at the beginning

2. For each insertion $e_i = (u, v)$, if $d(v) - d(u) - \omega(u, v) \geq D$, then

   - update $d(v) \leftarrow d(u) + \omega(u, v)$

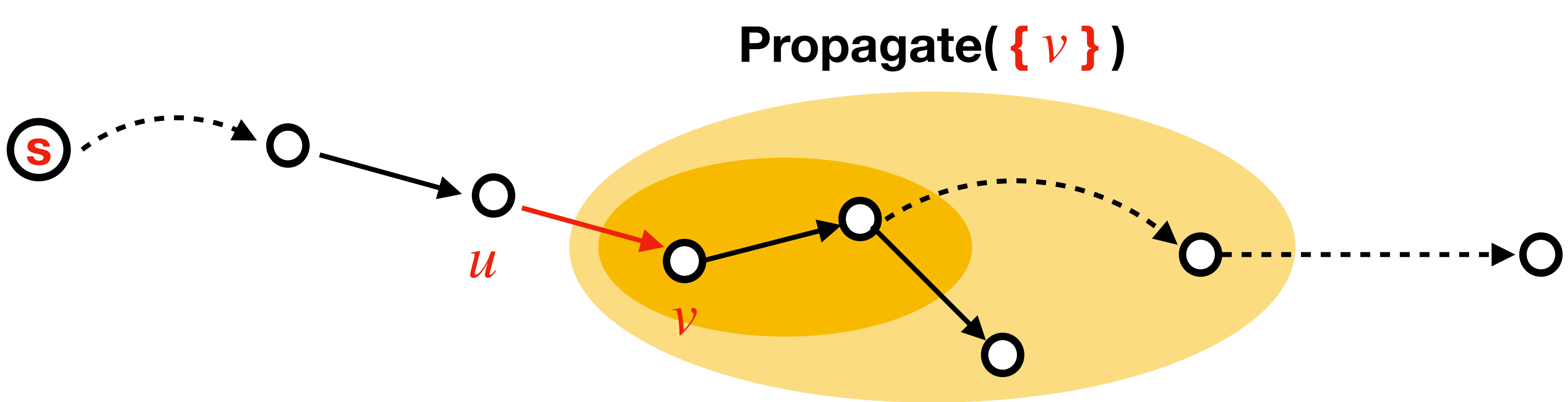
   - run **Propagate**( $\{ v \}$ )



# A deterministic algorithm

For every **B** insertions: $e_1, e_2, \cdots, e_B$

1. Run **Djikstra** to refresh all distance labels $d(\cdot)$ at the beginning
2. For each insertion $e_i = (u, v)$, if $d(v) - d(u) - \omega(u, v) \geq D$, then
   - update $d(v) \leftarrow d(u) + \omega(u, v)$
   - run **Propagate**( $\{ v \}$ )



# A deterministic algorithm

Running time analysis:

- Focus on dist(s, v) in [L, 2L], so there are only $\log(nW)$ scales
- Total number of **Djikstra** calls is $\leq m/B$
- d(v) drops by D each time we scan adj(v) during **Propagate**

  Total cost of **Propagate** is at most $\sum_v L/D \cdot \deg(v) = Lm/D$

- Total update time $\approx m^2/B + Lm/D$
- How to choose B?

# A wrong guess

- Total update time $\approx m^2/B + Lm/D$

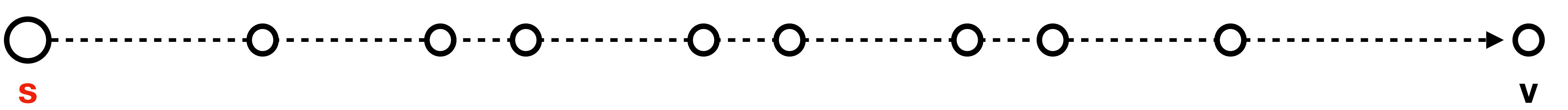
- How to choose B?

Example:

1. start with dist(s, v) = 2L = d(v)

2.

3.



# A wrong guess

- Total update time $\approx m^2/B + Lm/D$

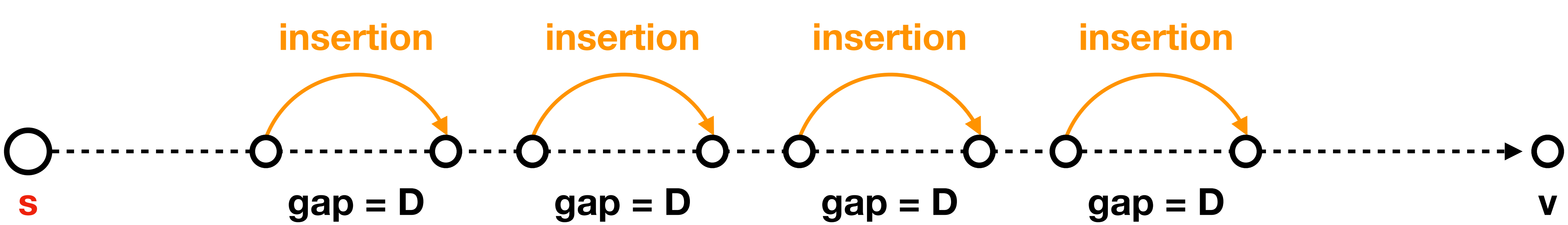
- How to choose B?

Example:

1. start with dist(s, v) = 2L = d(v)

2. insert $10\epsilon L/D$ shortcuts along the path

3.



# A wrong guess

- Total update time $\approx m^2/B + Lm/D$

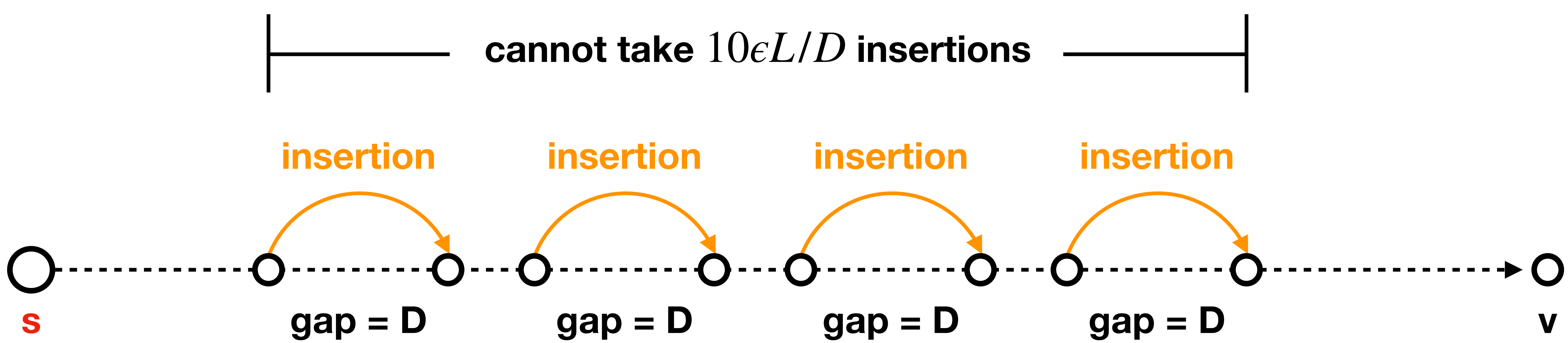
- How to choose B?

Example:

1. start with dist(s, v) = 2L = d(v)

2. insert $10\epsilon L/D$ shortcuts along the path

3. dist(s, v) gets below $(2-2\epsilon)L$, so d(v) becomes a bad approximation



# A wrong guess

- Total update time $\approx m^2/B + Lm/D$

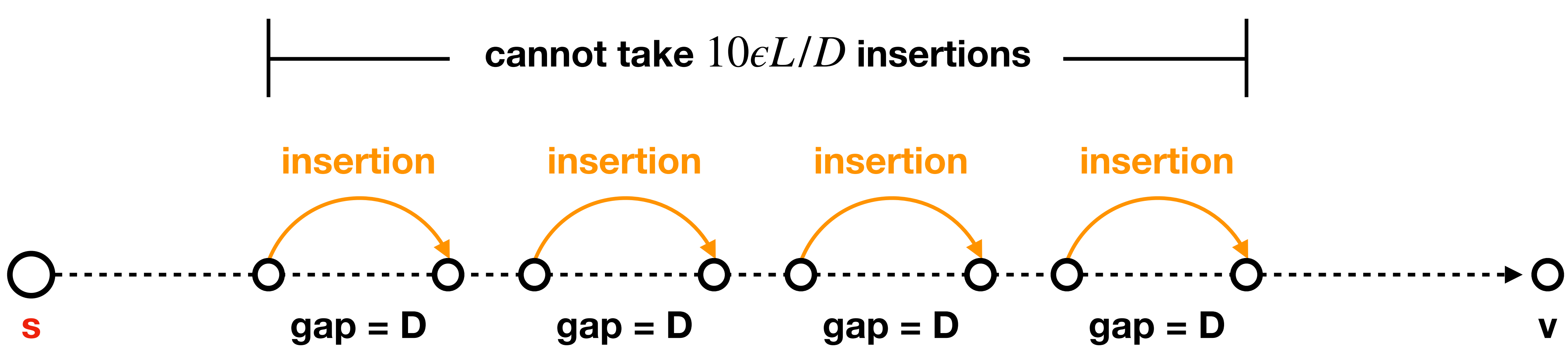
- How to choose B? **Choose B = $\epsilon L/D$ ?**

<u>Example:</u>

1. start with dist(s, v) = 2L = d(v)

2. insert $10\epsilon L/D$ shortcuts along the path

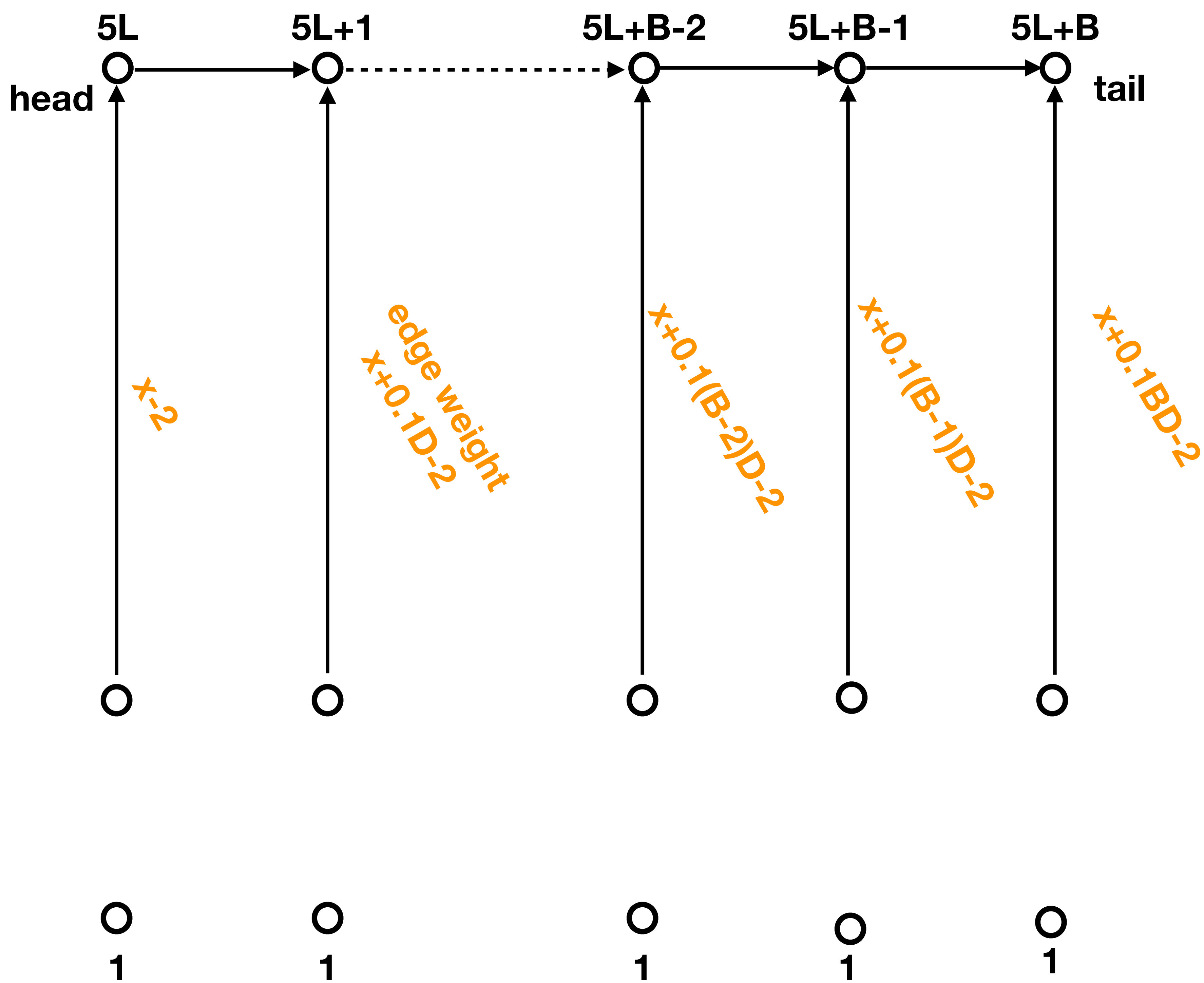
3. dist(s, v) gets below $(2-2\epsilon)L$, so d(v) becomes a bad approximation



# A counter example

- Construct the following gadget

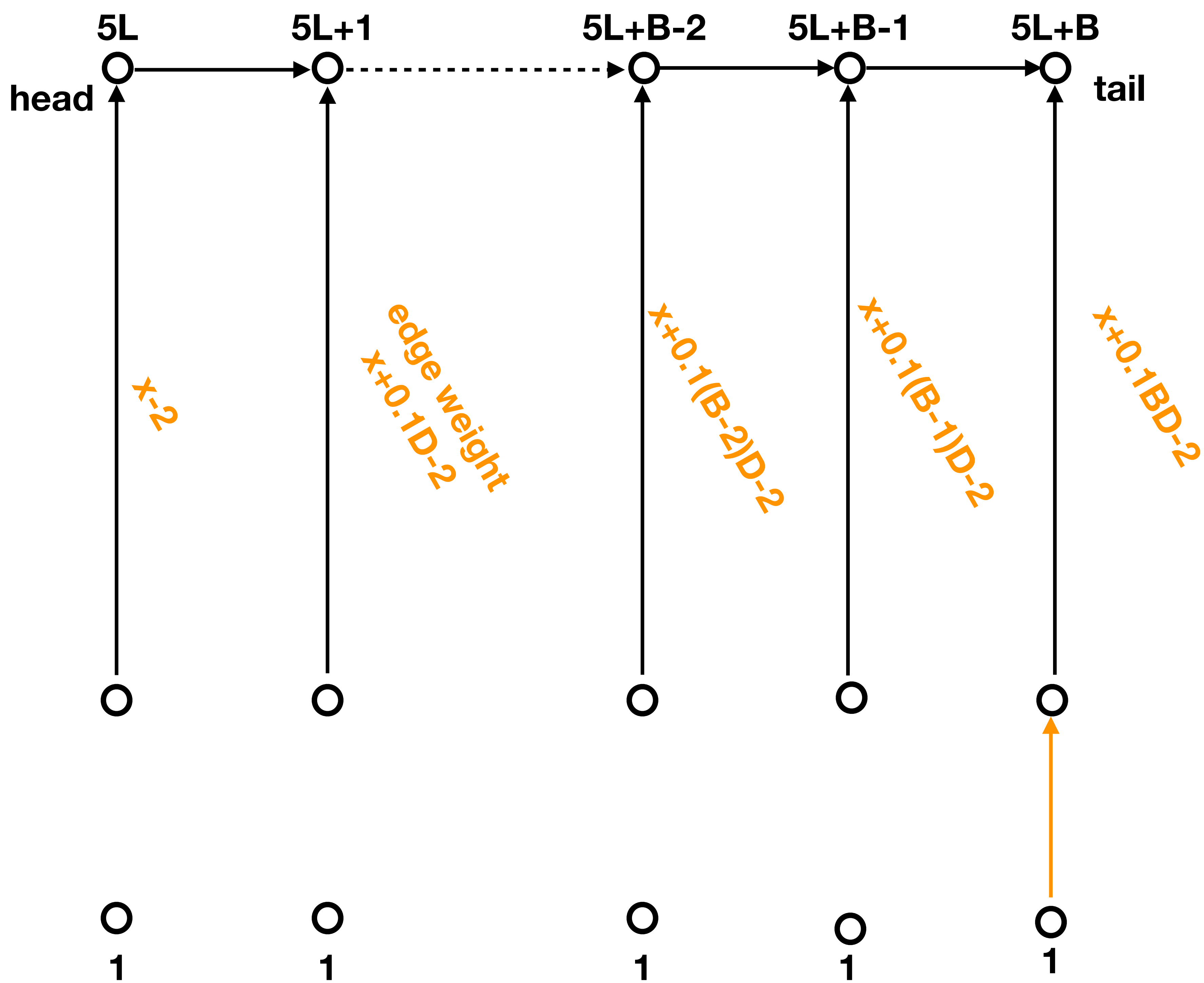
**Before insertions:**
$d(\textbf{tail}) - d(\textbf{head}) = B$



# A counter example

- Construct the following gadget

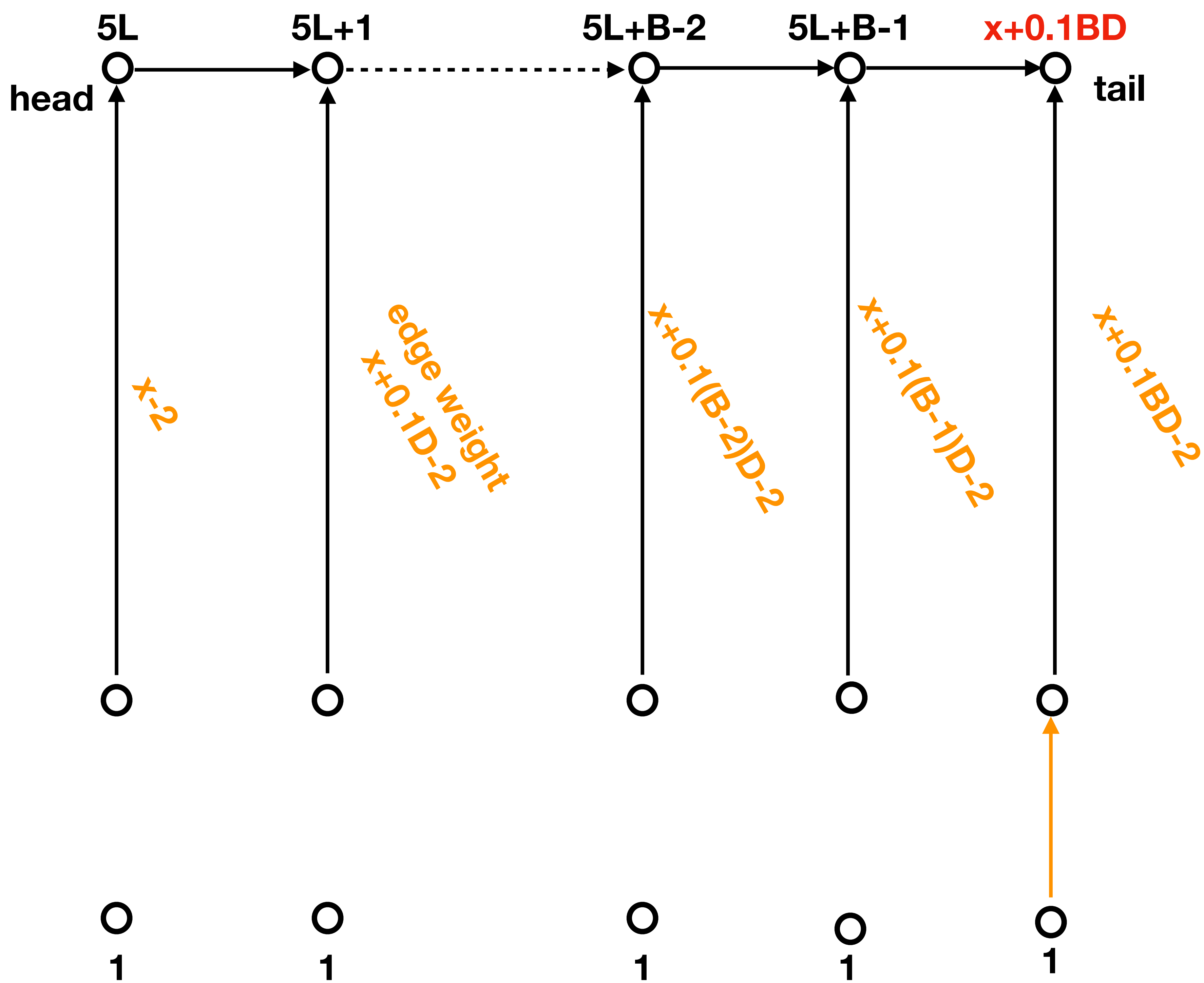
**Before insertions:**
$d(\textbf{tail}) - d(\textbf{head}) = B$



# A counter example

- Construct the following gadget

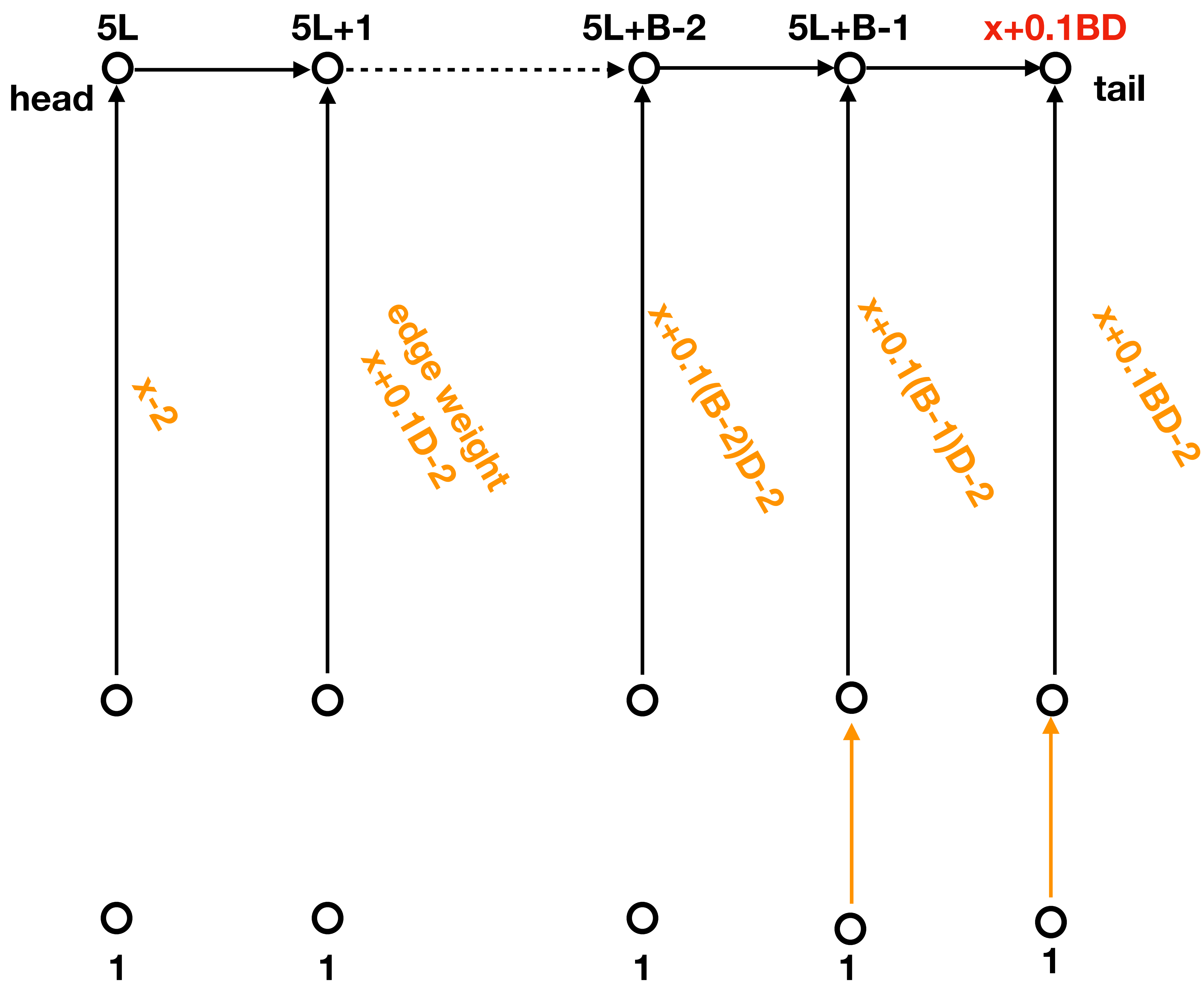
**Before insertions:**
$d(\textbf{tail}) - d(\textbf{head}) = B$



# A counter example

- Construct the following gadget

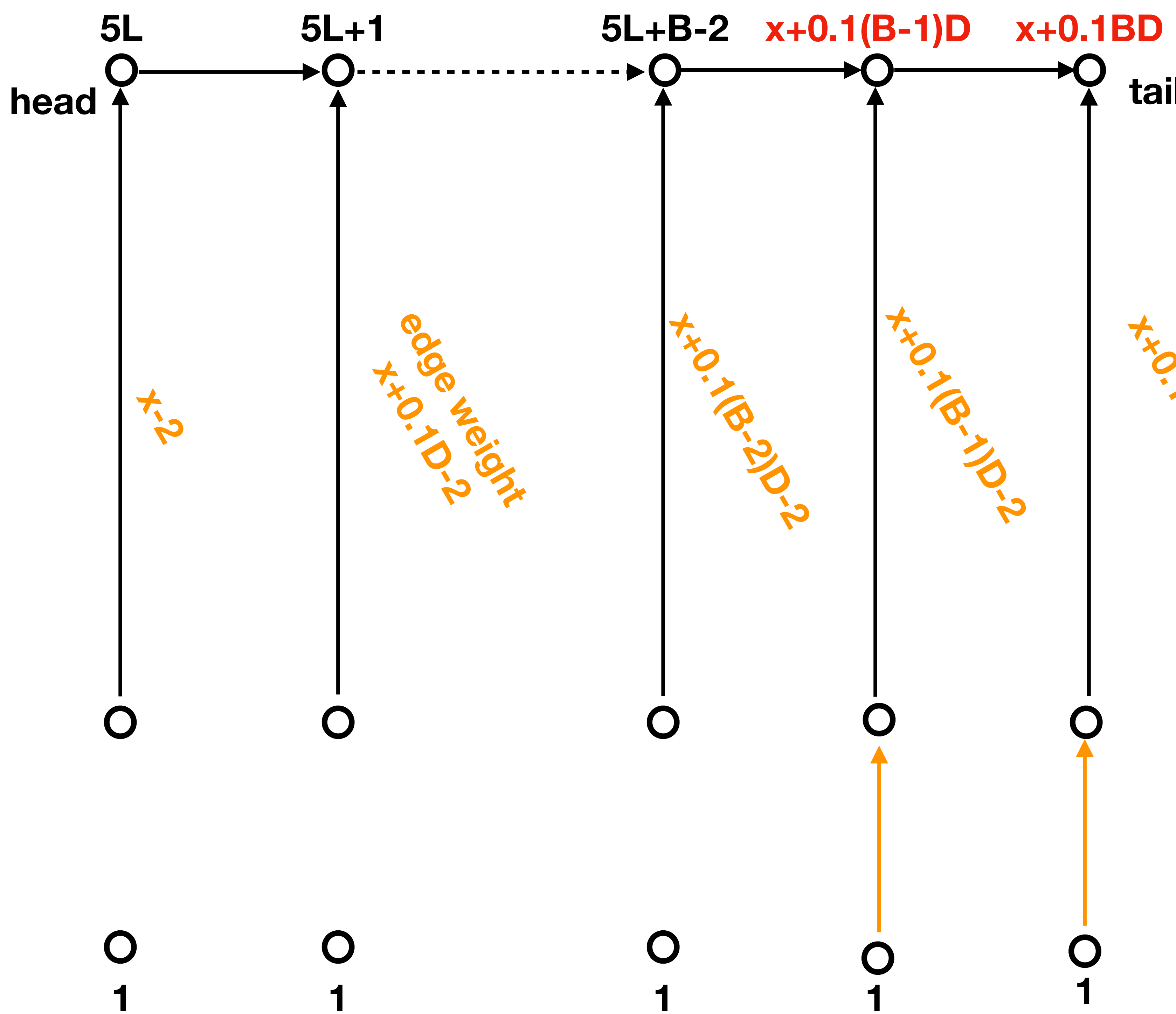
**Before insertions:**
$d(\textbf{tail}) - d(\textbf{head}) = B$



# A counter example

- Construct the following gadget

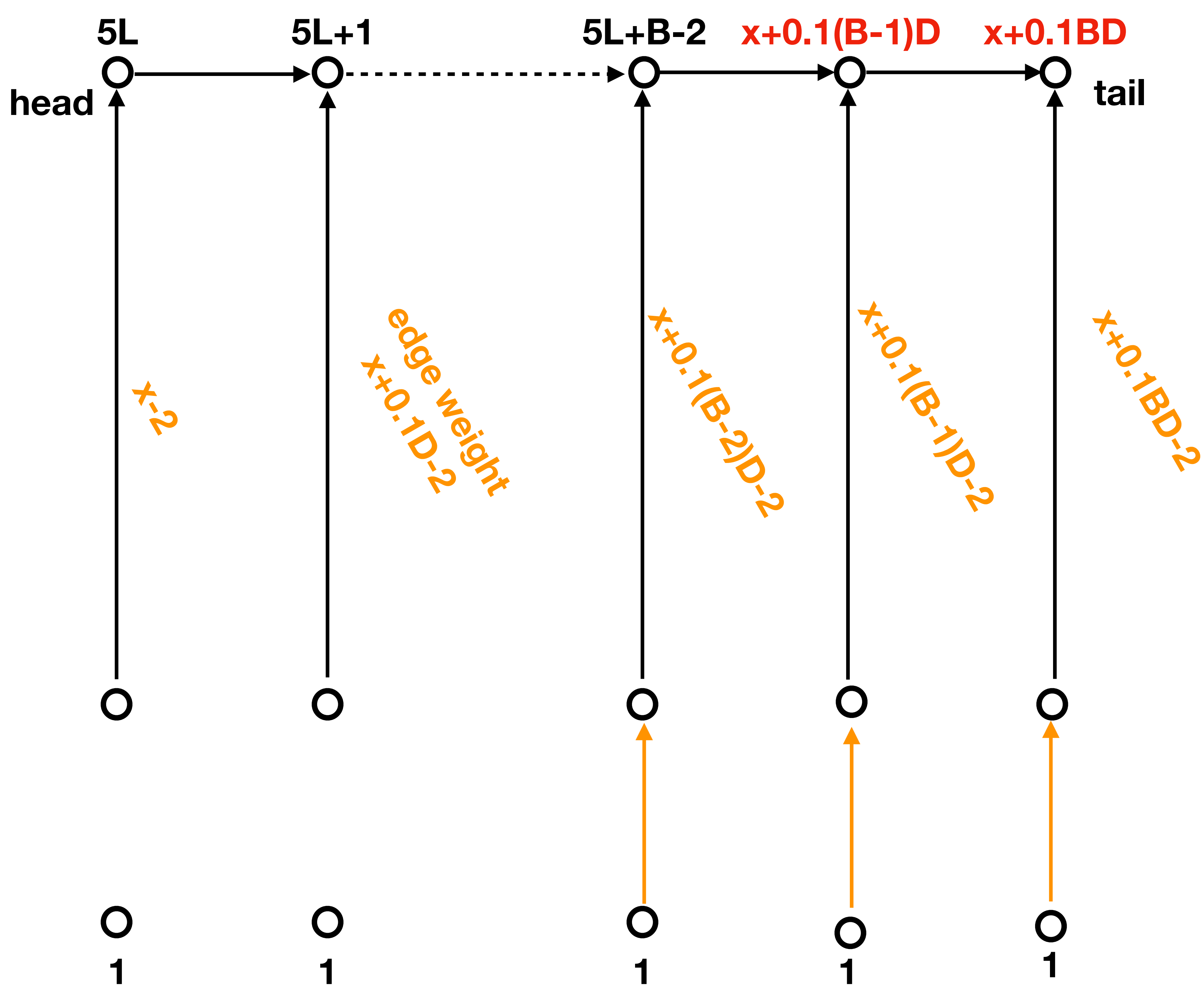
**Before insertions:**
$d(\textbf{tail}) - d(\textbf{head}) = B$



# A counter example

- Construct the following gadget

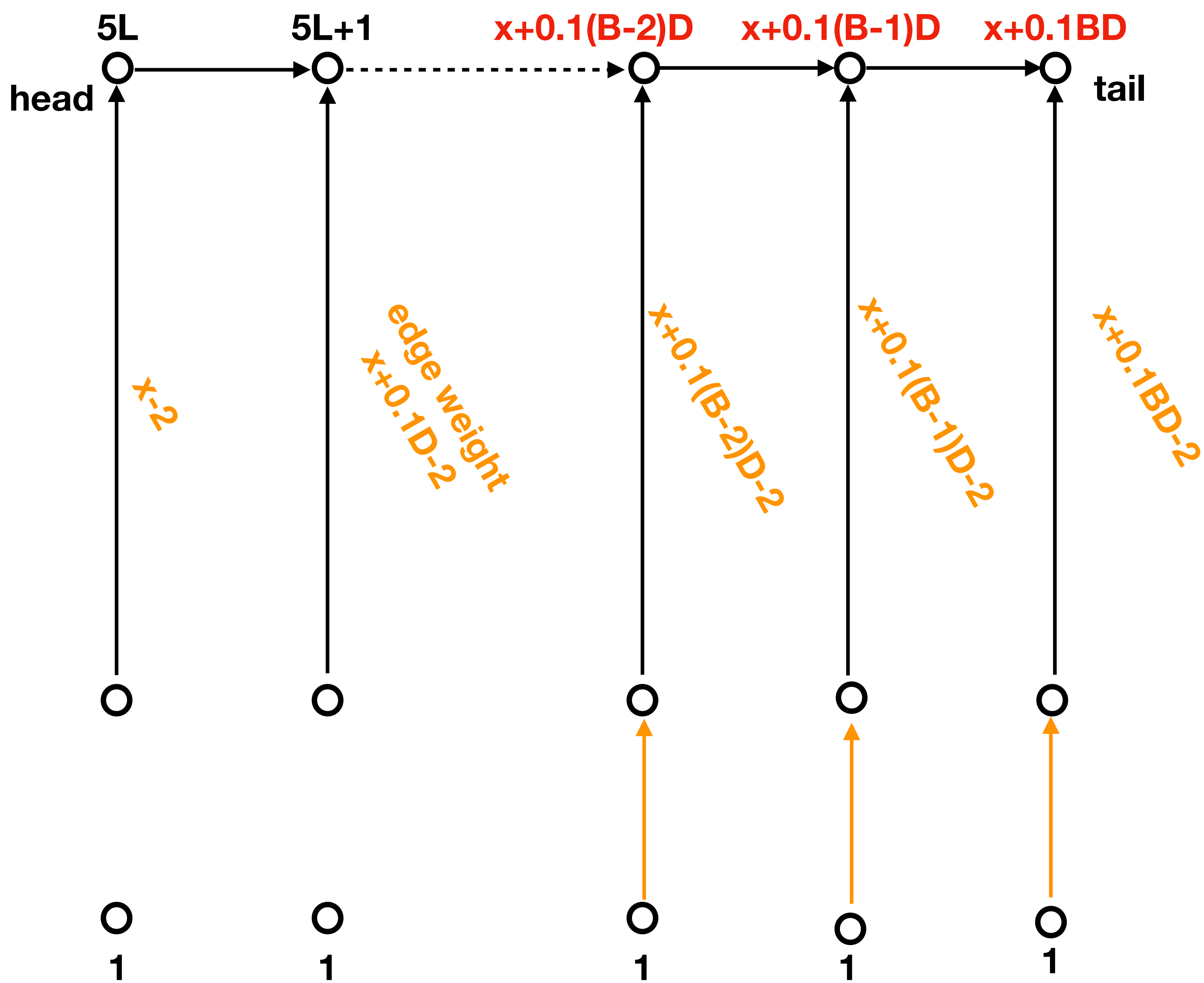
**Before insertions:**
$d(\textbf{tail}) - d(\textbf{head}) = B$



# A counter example

- Construct the following gadget

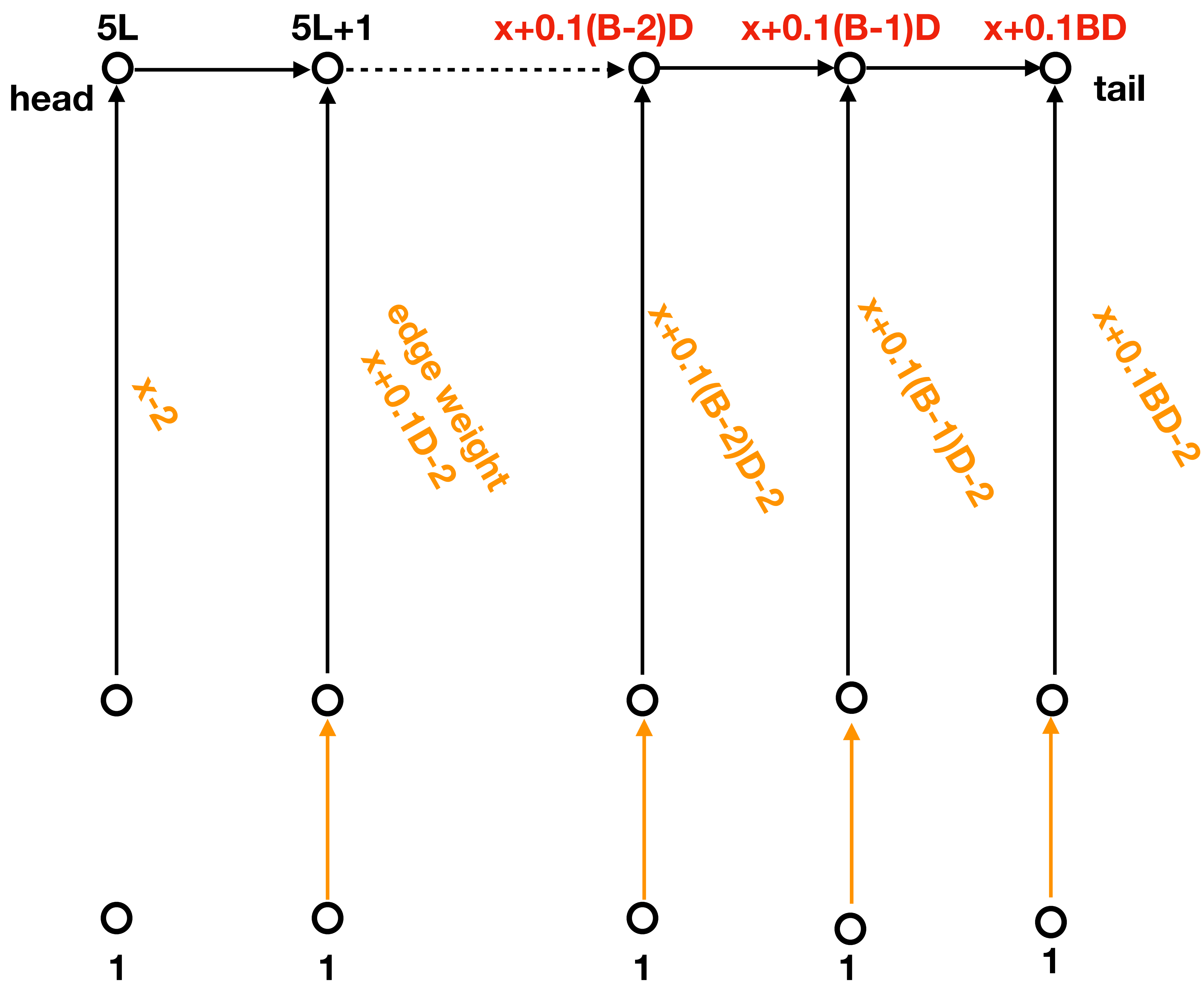
**Before insertions:**
$d(\textbf{tail}) - d(\textbf{head}) = B$



# A counter example

- Construct the following gadget

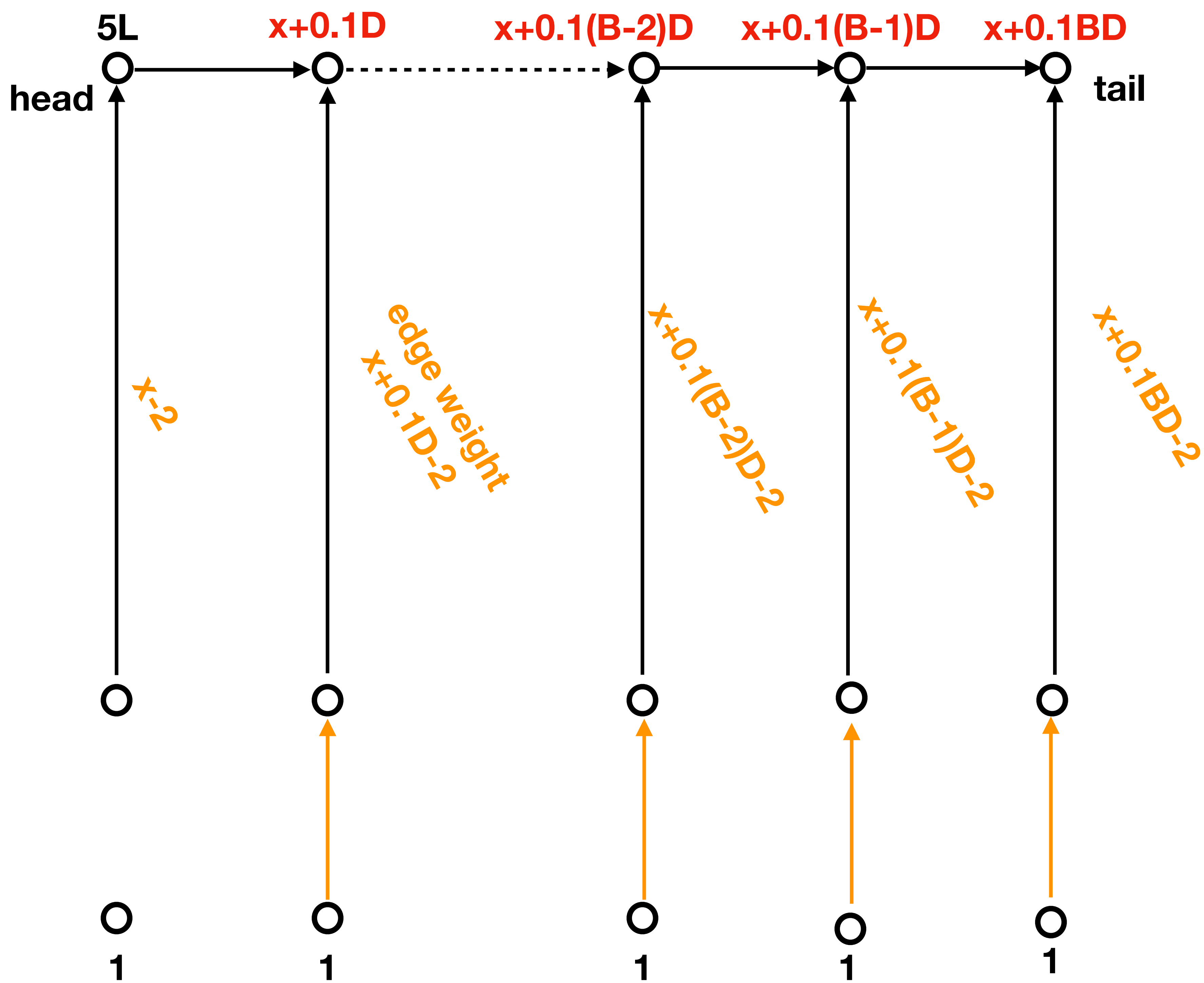
**Before insertions:**
$d(\textbf{tail}) - d(\textbf{head}) = B$



# A counter example

- Construct the following gadget

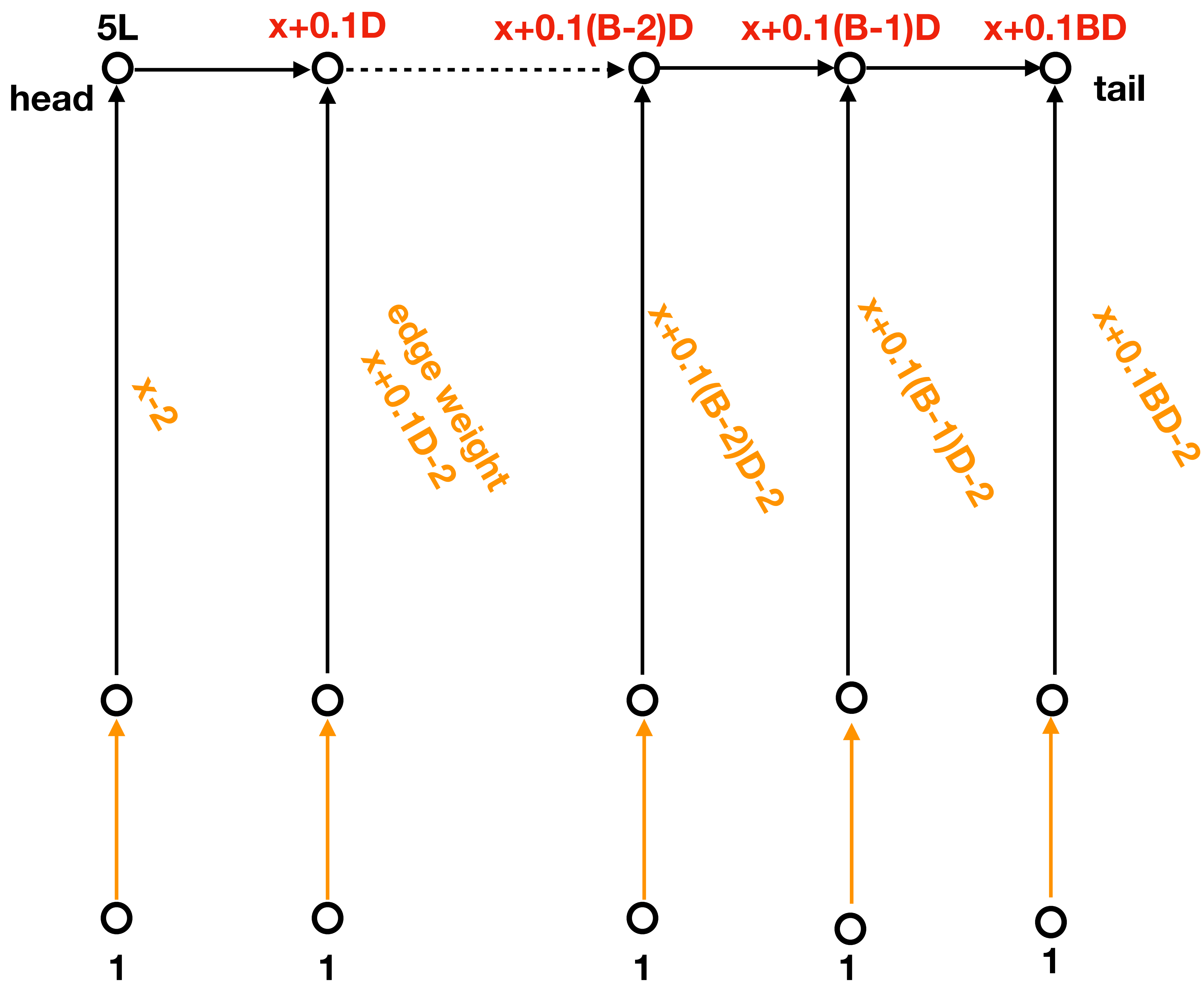
**Before insertions:**
$d(\textbf{tail}) - d(\textbf{head}) = B$



# A counter example

- Construct the following gadget

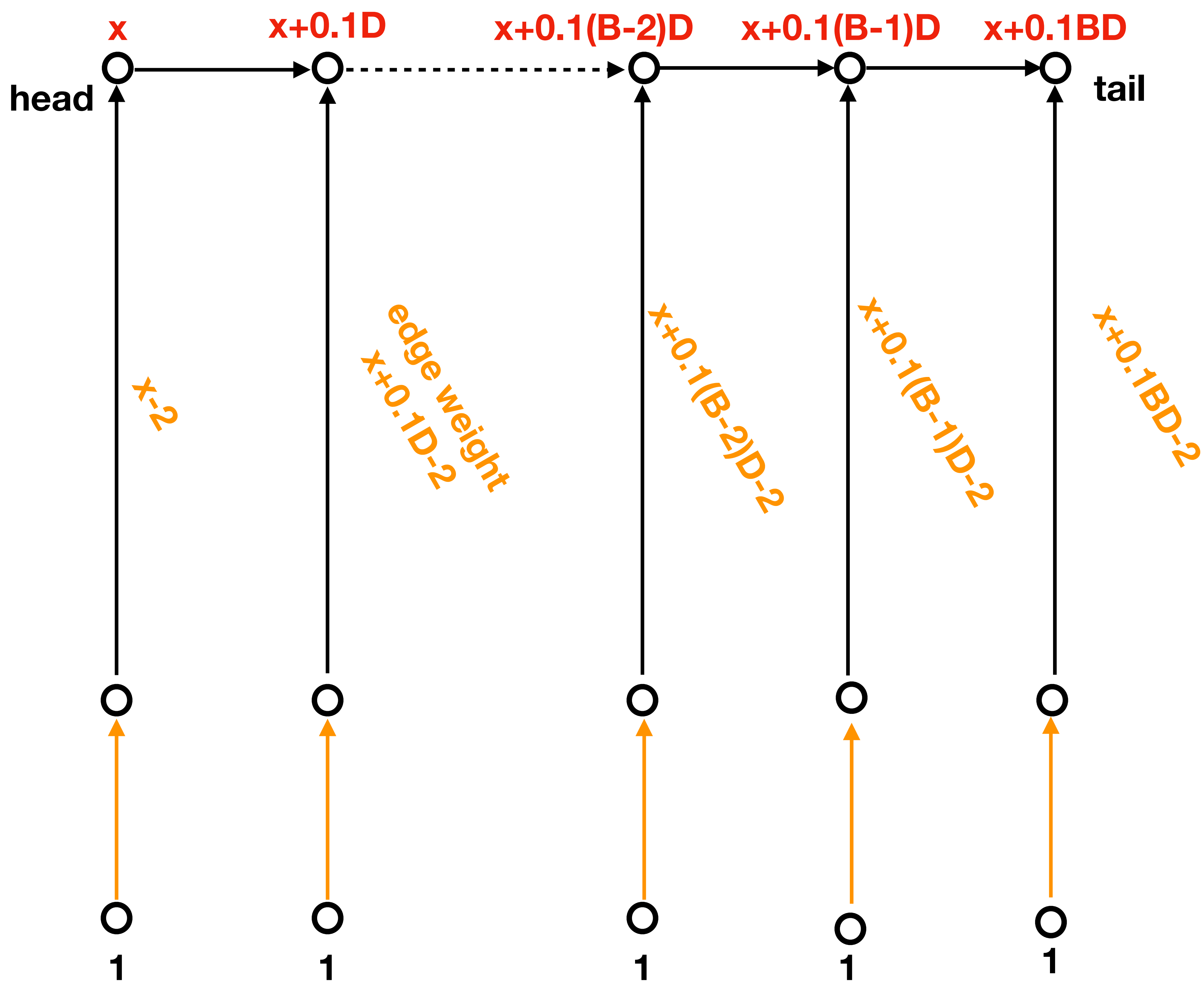
**Before insertions:**
$d(\textbf{tail}) - d(\textbf{head}) = B$



# A counter example

- Construct the following gadget

**Before insertions:**
$d(\textbf{tail}) - d(\textbf{head}) = B$

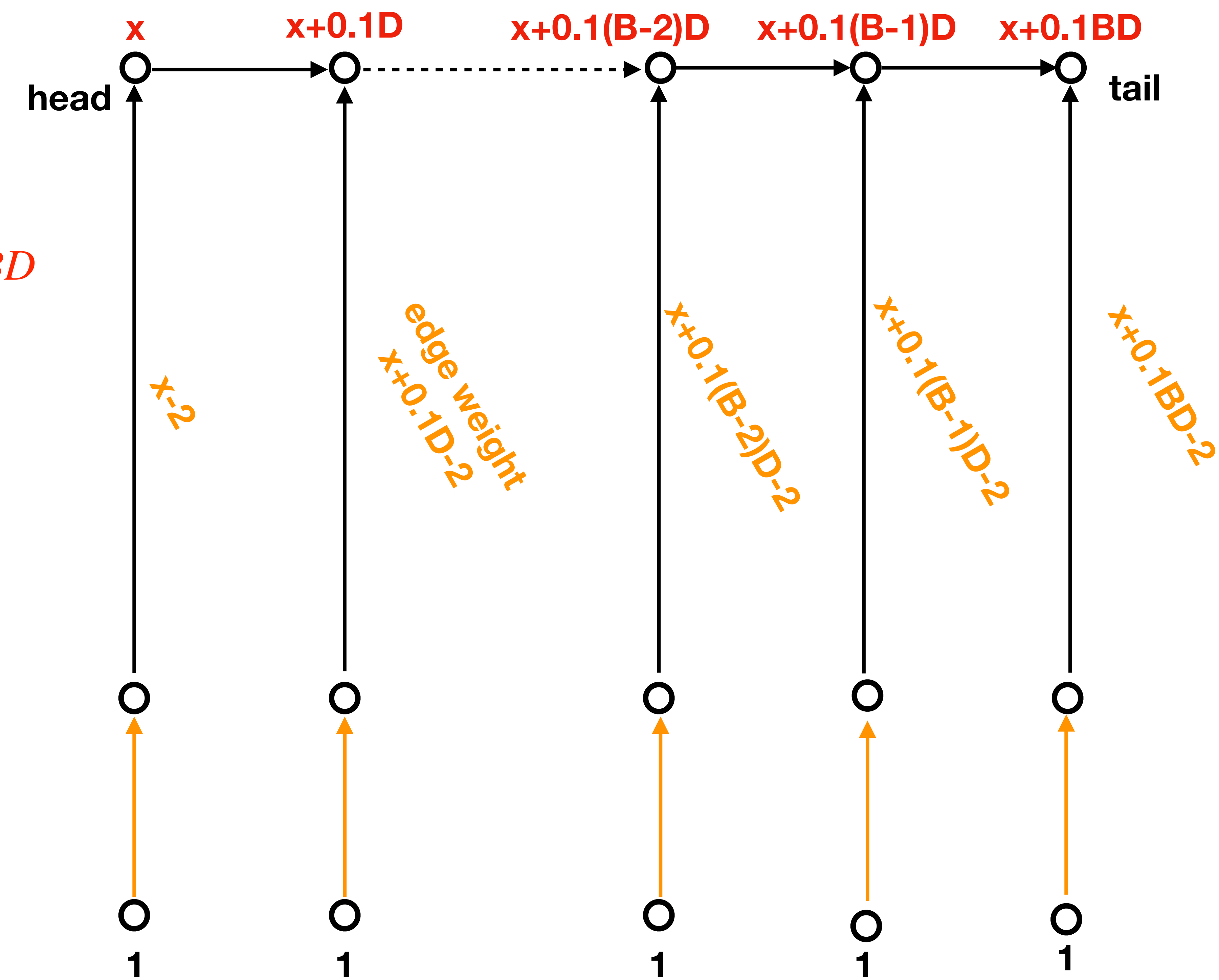


# A counter example

- Construct the following gadget

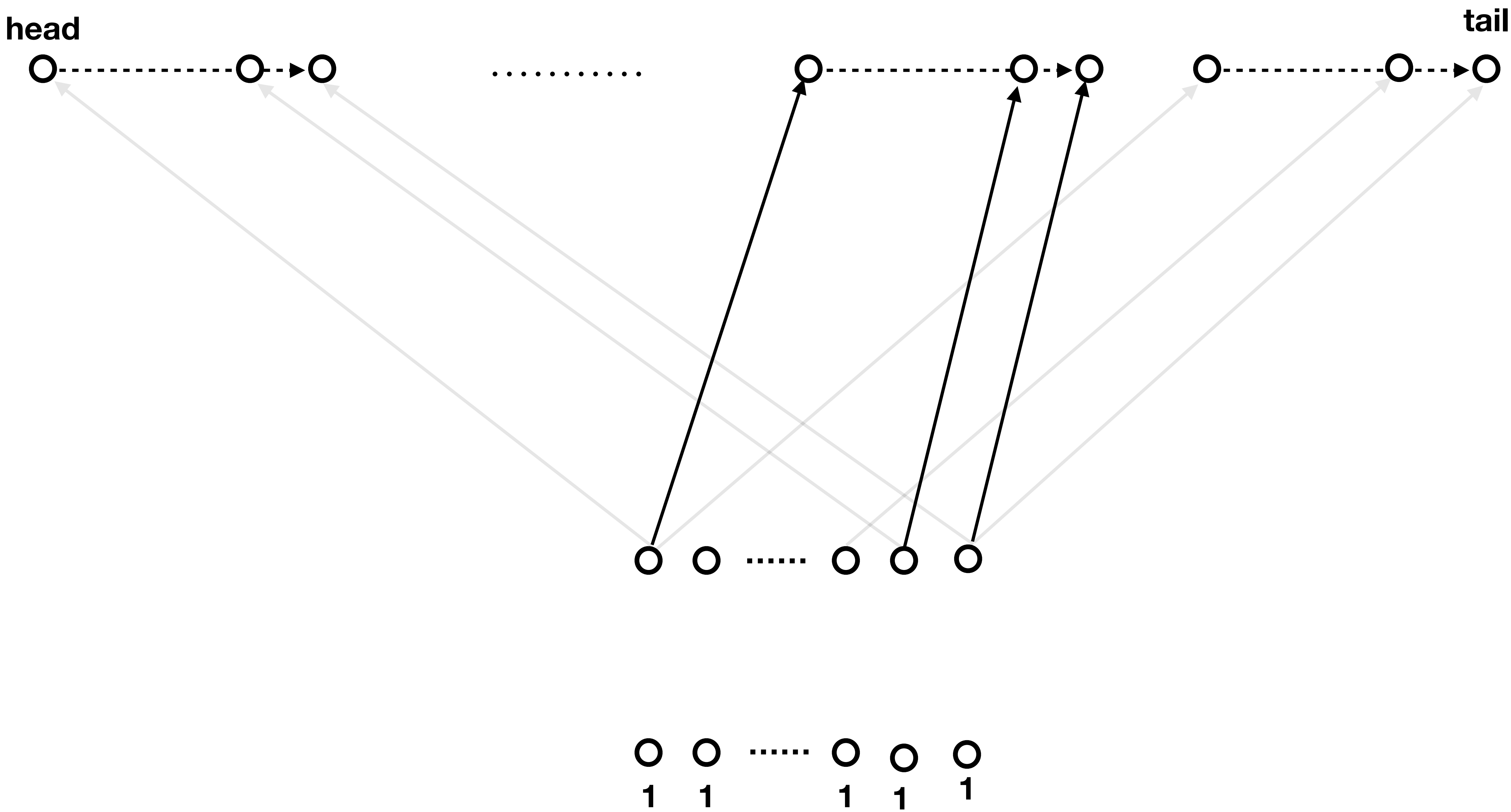
**Before insertions:**
$d(\textbf{tail}) - d(\textbf{head}) = B$

**After insertions:**
$d(\textbf{tail}) - d(\textbf{head}) = 0.1BD$



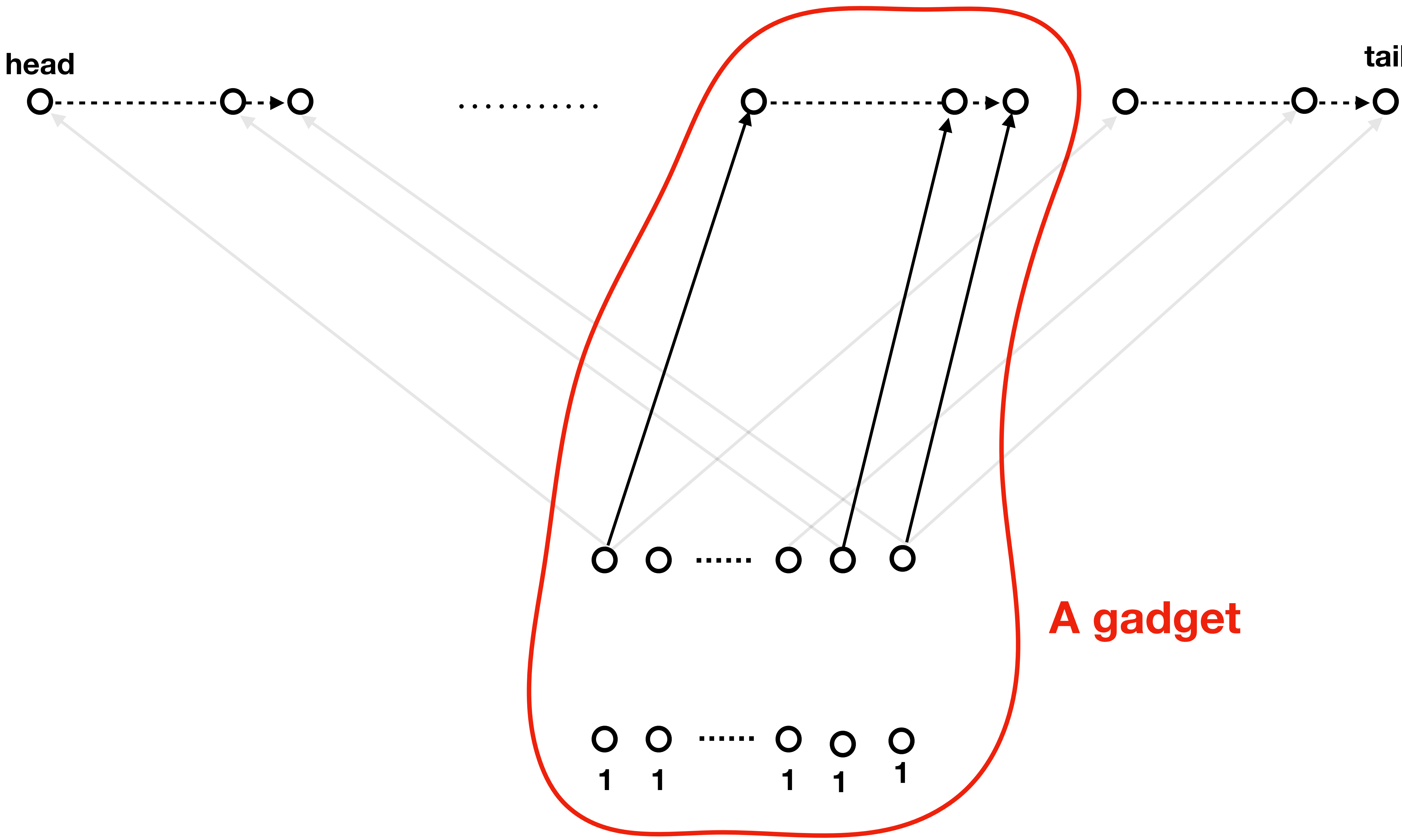
# A counter example

- Use gadgets to construct a counter example



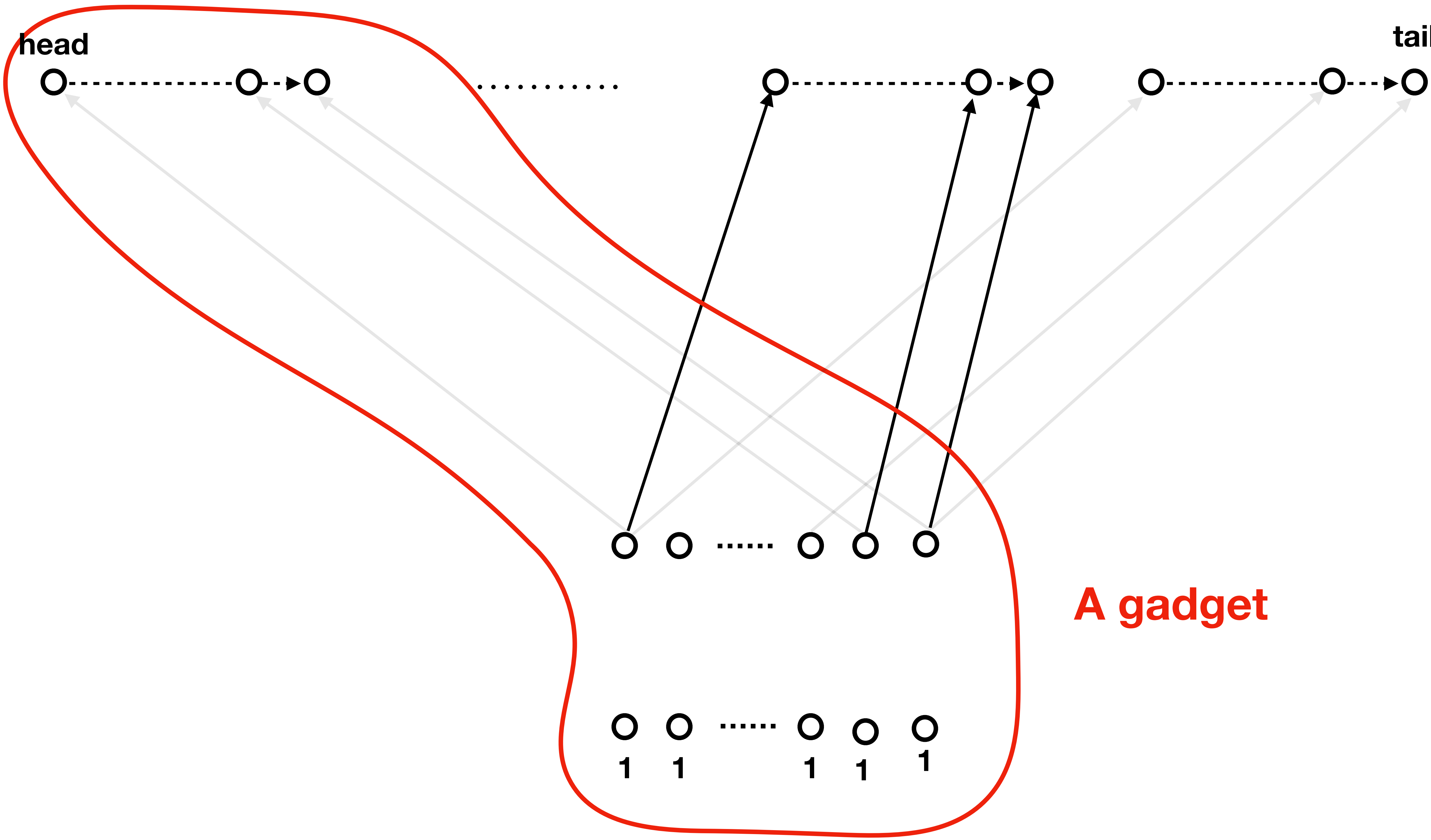
# A counter example

- Use gadgets to construct a counter example



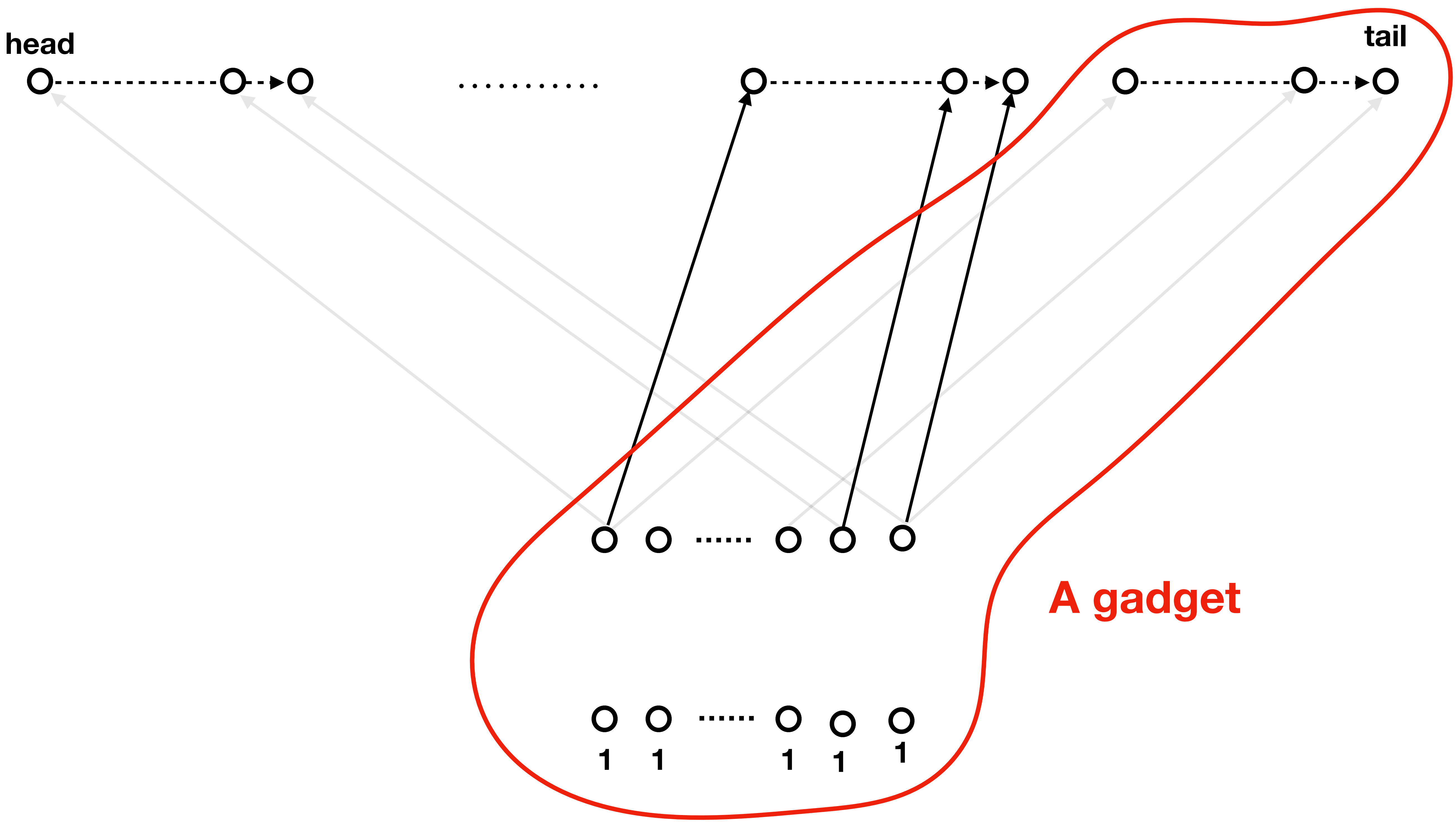
# A counter example

- Use gadgets to construct a counter example



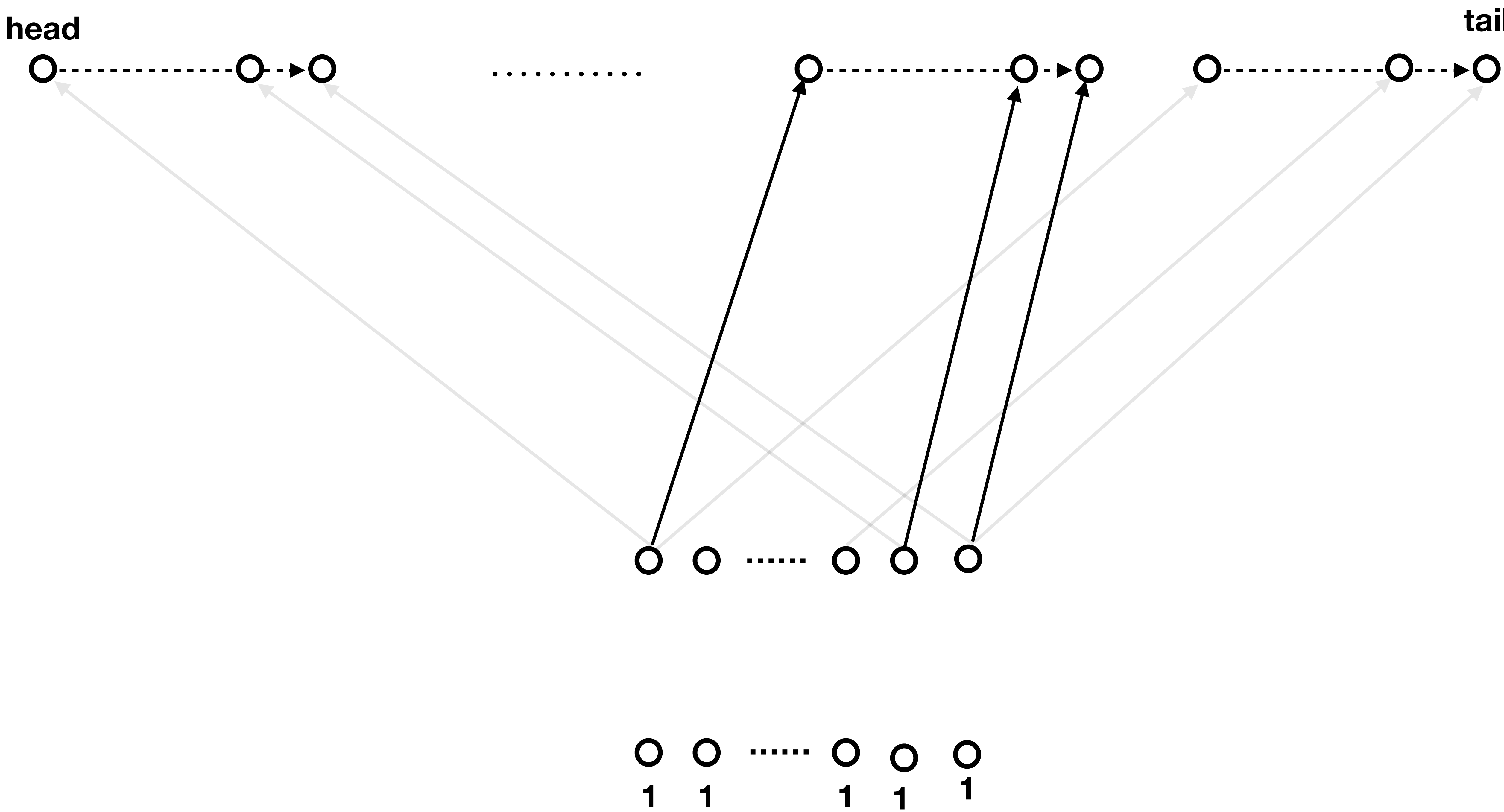
# A counter example

- Use gadgets to construct a counter example



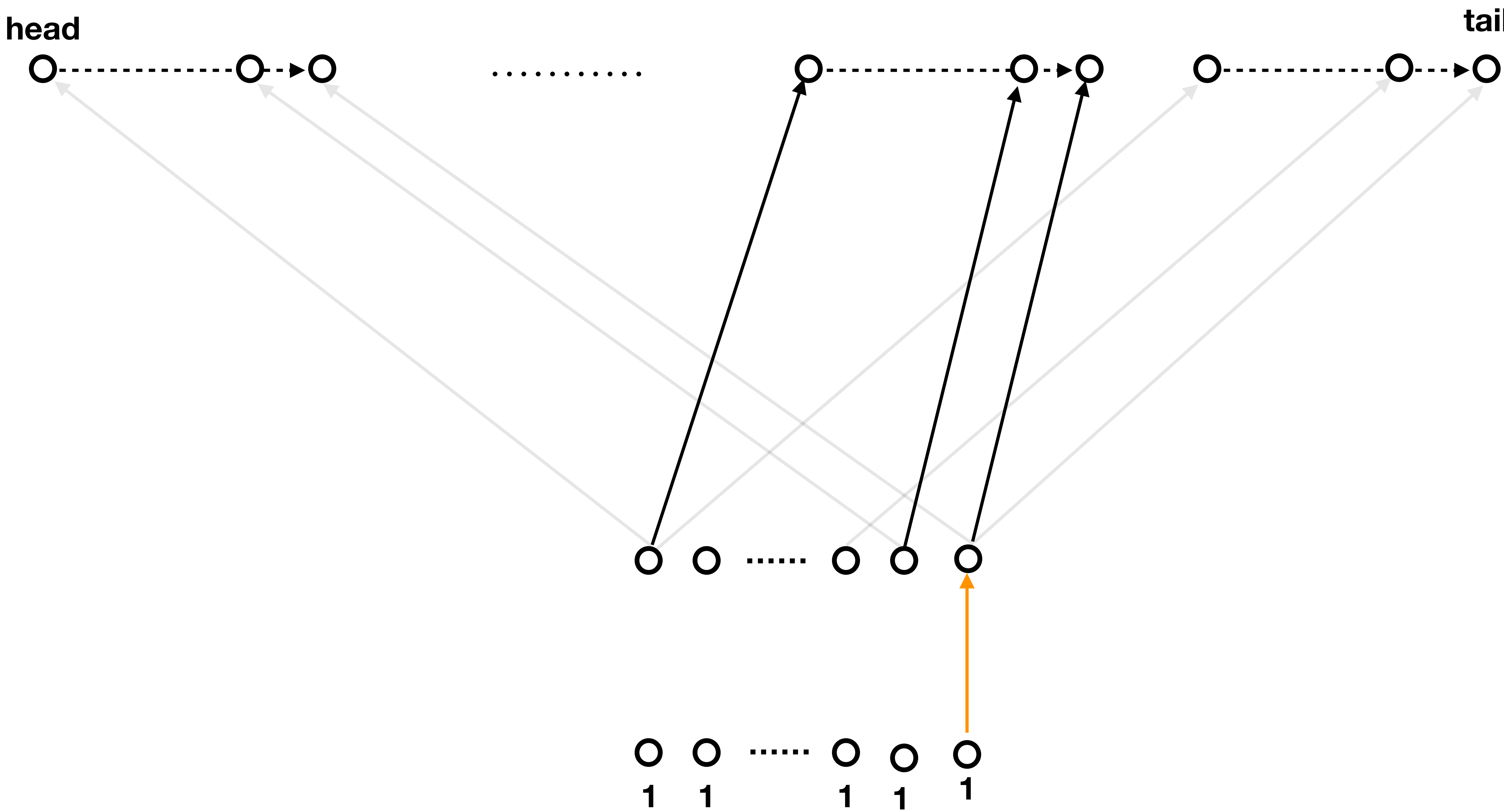
# A counter example

- Use gadgets to construct a counter example



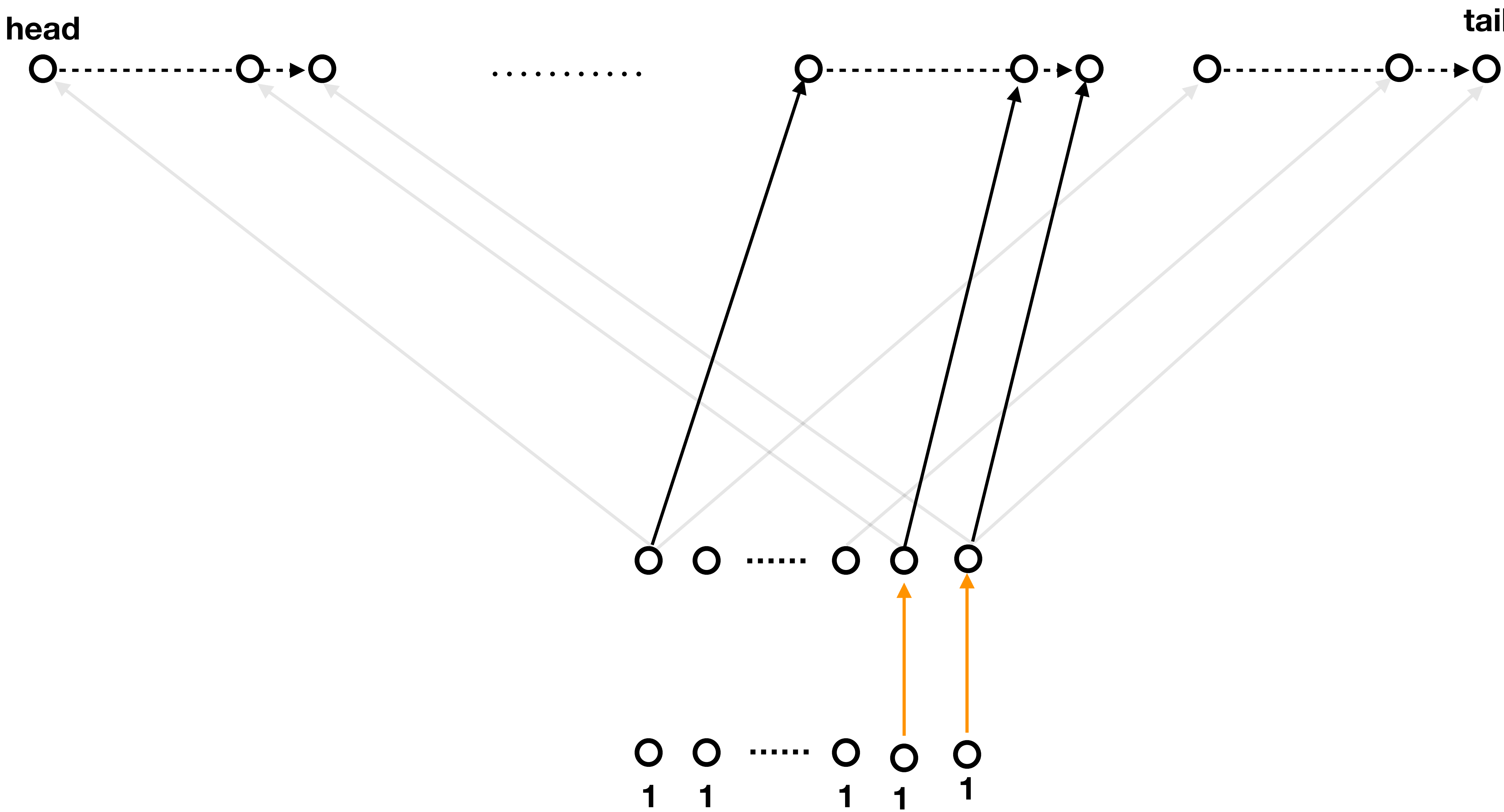
# A counter example

- Use gadgets to construct a counter example



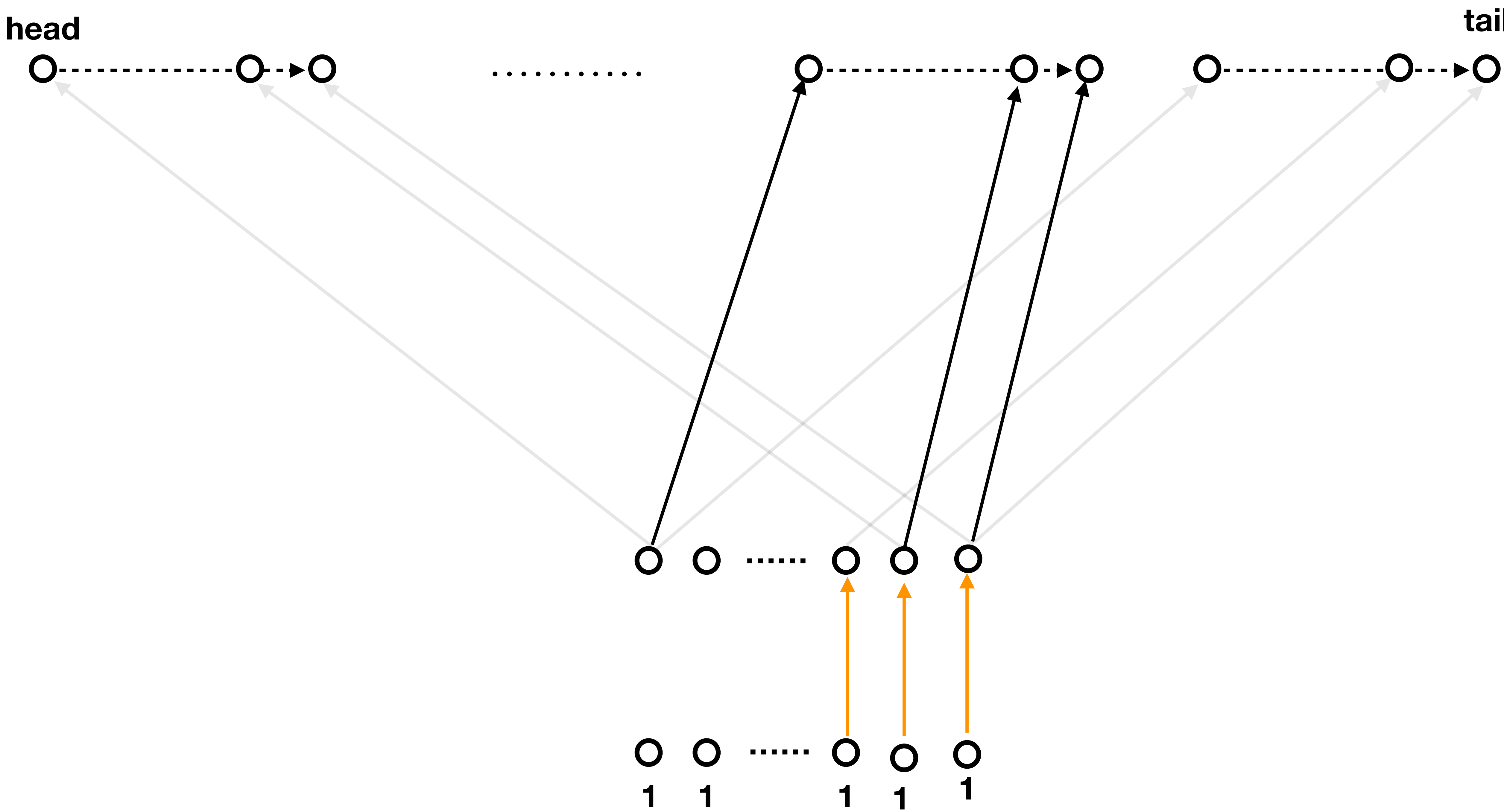
# A counter example

- Use gadgets to construct a counter example



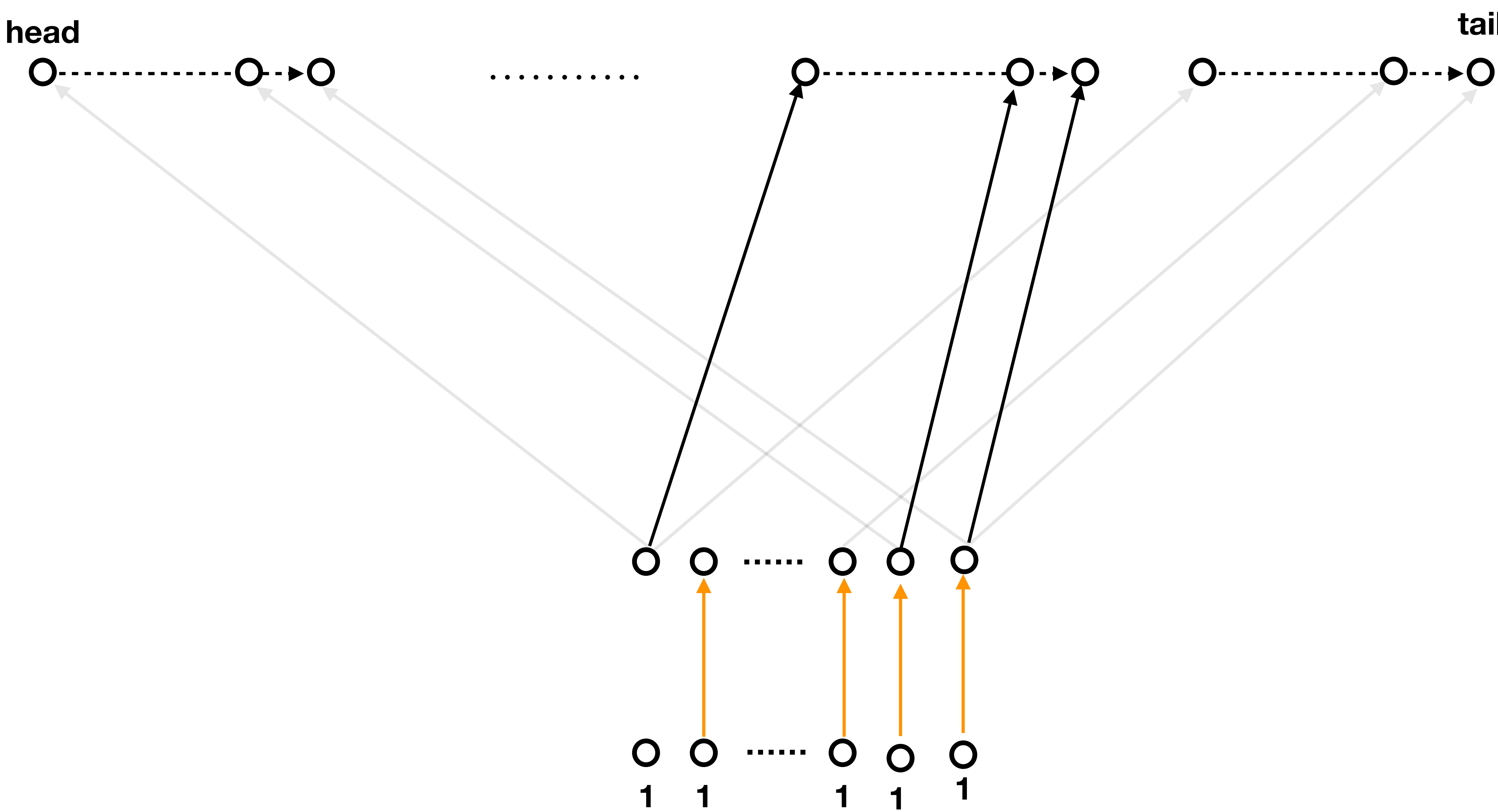
# A counter example

- Use gadgets to construct a counter example



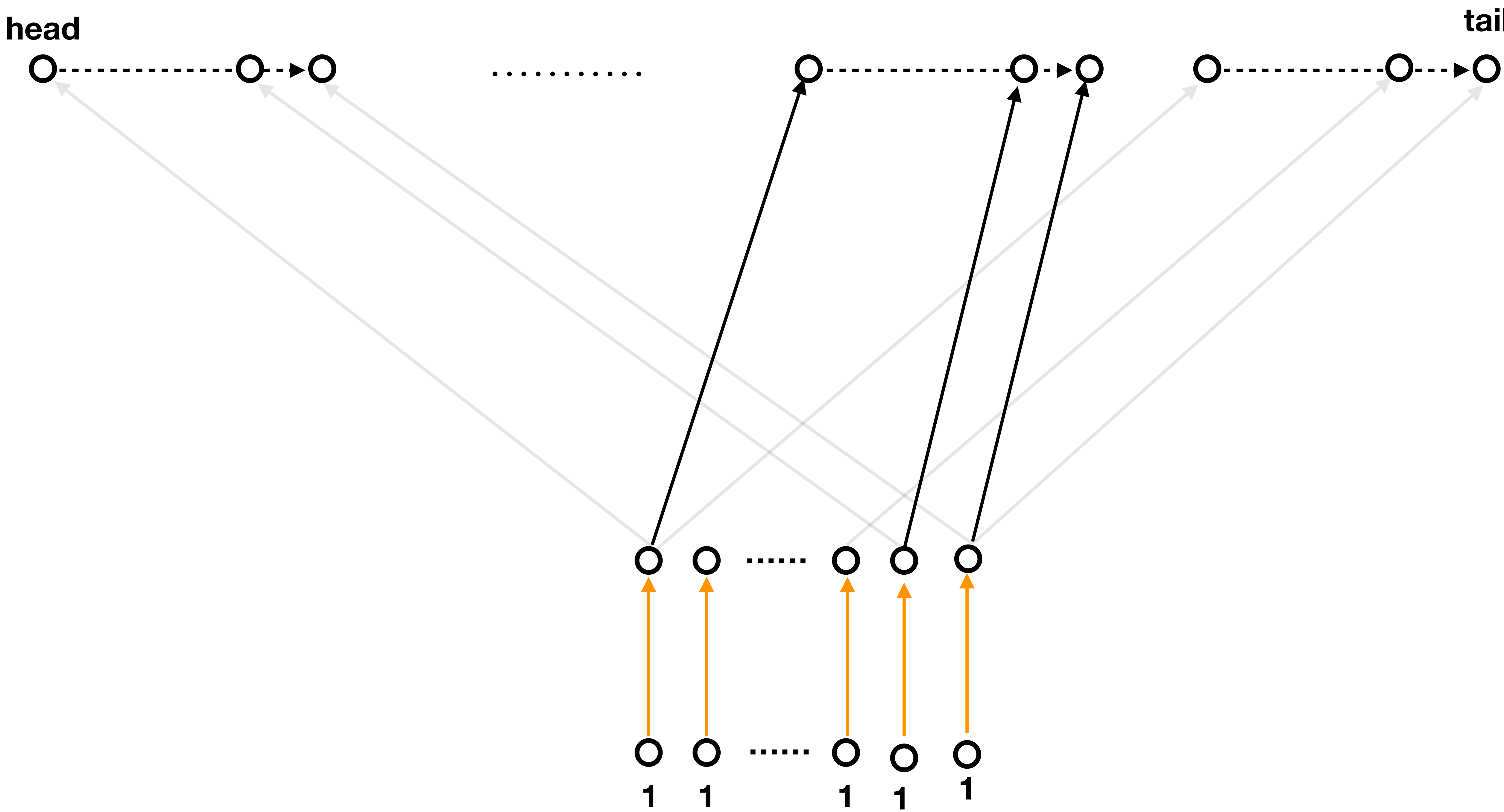
# A counter example

- Use gadgets to construct a counter example



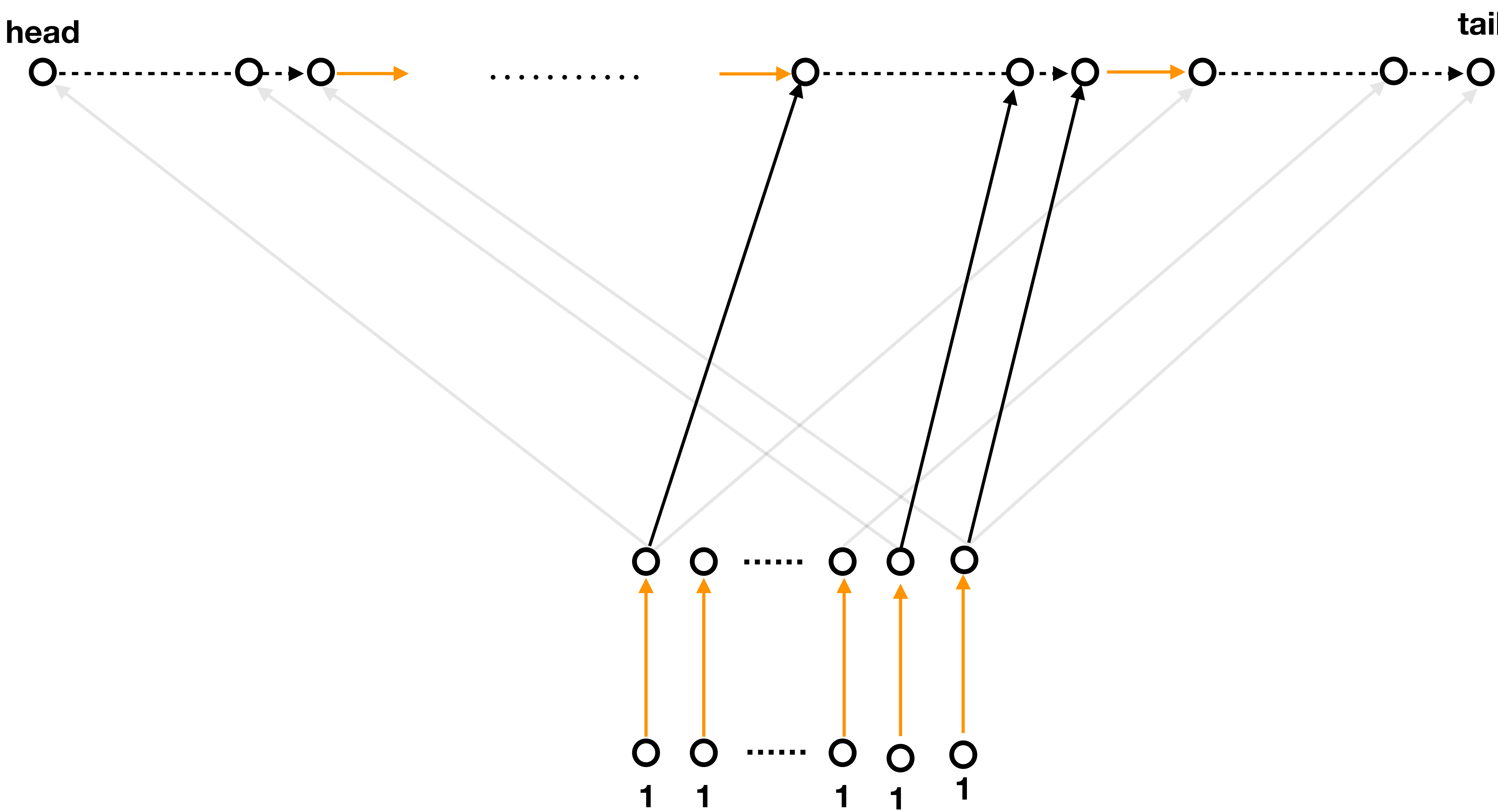
# A counter example

- Use gadgets to construct a counter example



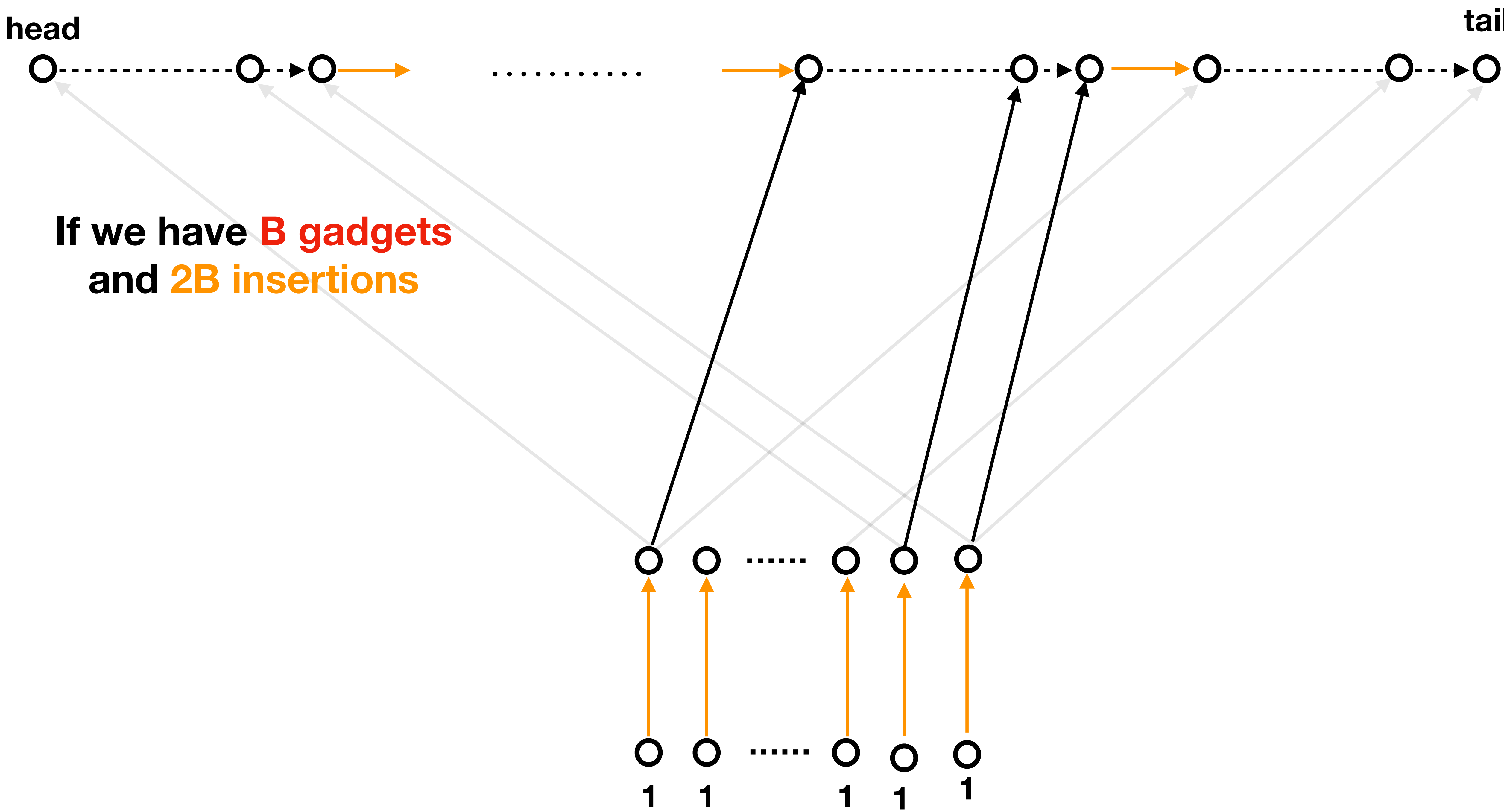
# A counter example

- Use gadgets to construct a counter example



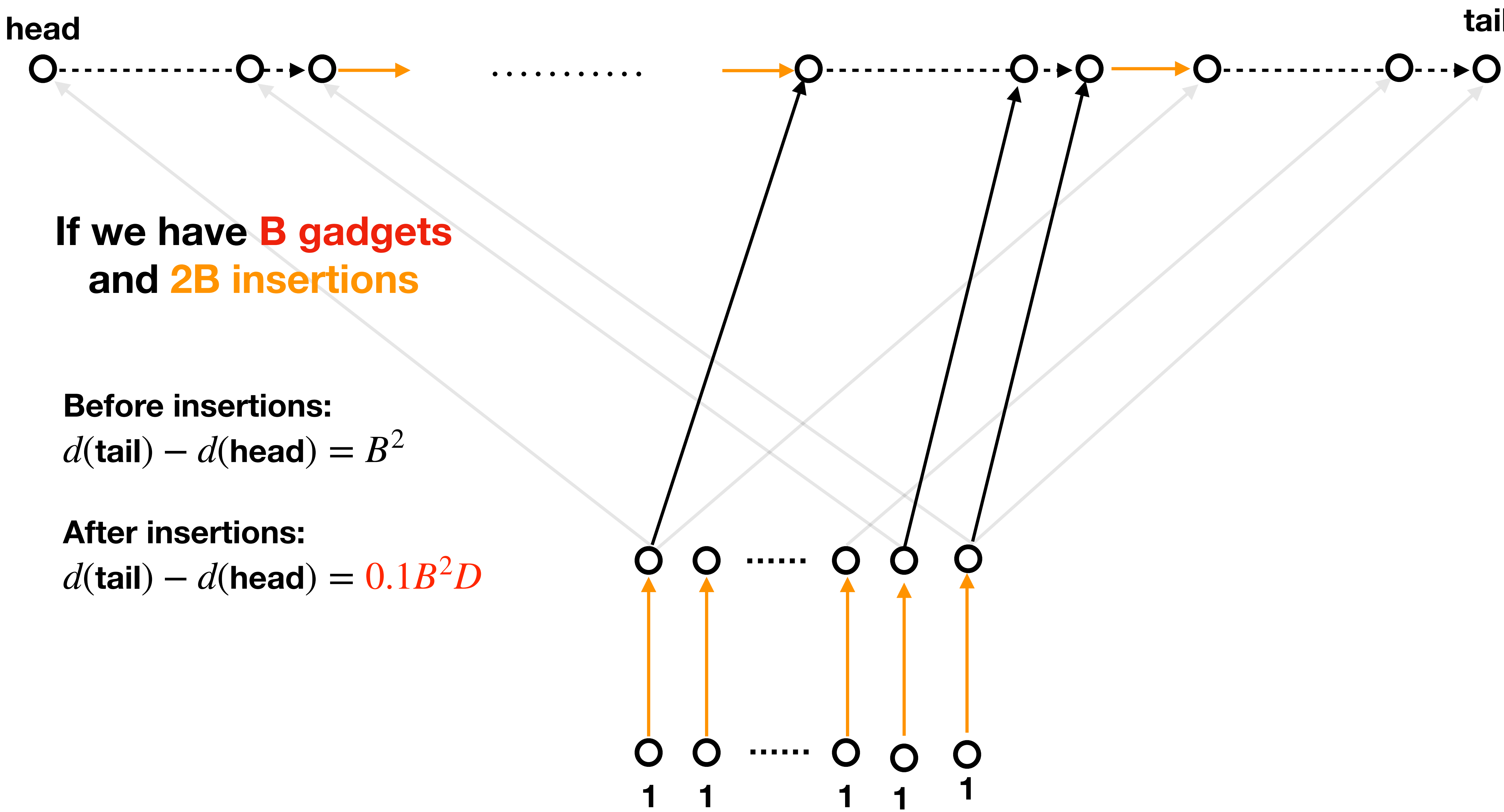
# A counter example

- Use gadgets to construct a counter example



# A counter example

- Use gadgets to construct a counter example




**If we have B gadgets and 2B insertions**

**Before insertions:**

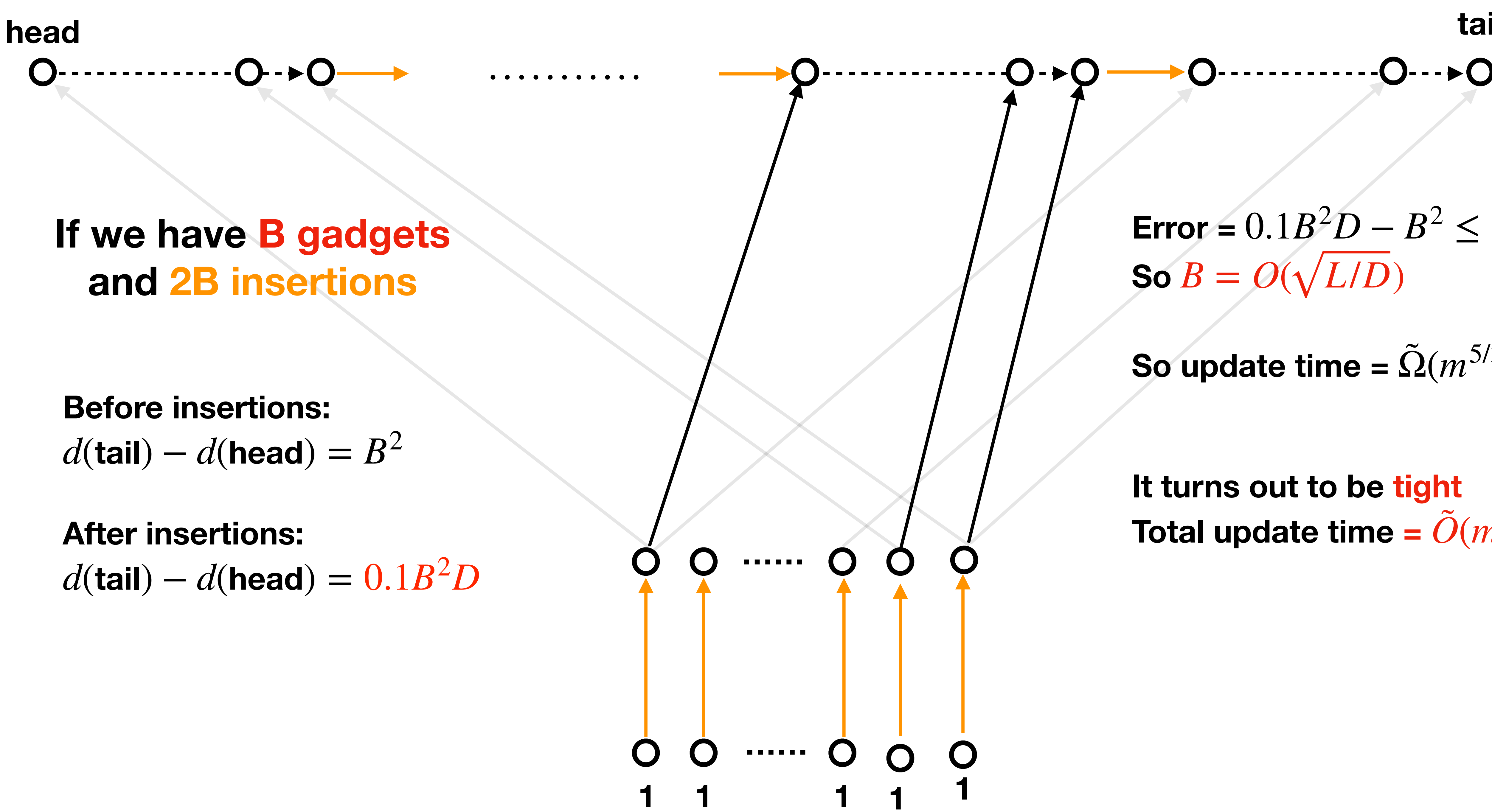
$d(\textbf{tail}) - d(\textbf{head}) = B^2$

**After insertions:**

$d(\textbf{tail}) - d(\textbf{head}) = 0.1B^2D$

# A counter example

- Use gadgets to construct a counter example




**If we have B gadgets and 2B insertions**

**Before insertions:**

$d(\textbf{tail}) - d(\textbf{head}) = B^2$

**After insertions:**

$d(\textbf{tail}) - d(\textbf{head}) = 0.1B^2D$

**Error =** $0.1B^2D - B^2 \leq \epsilon L$

**So** $B = O(\sqrt{L/D})$

**So update time =** $\tilde{\Omega}(m^{5/3})$

**It turns out to be tight**

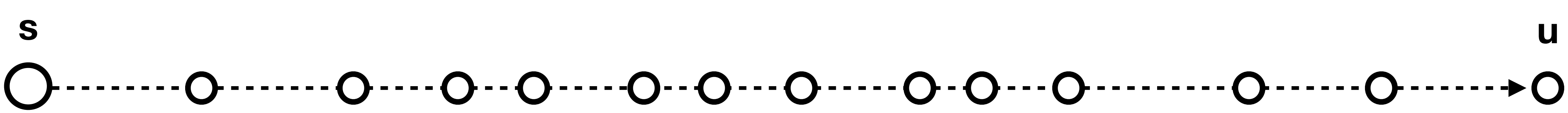
**Total update time =** $\tilde{O}(m^{5/3})$

# A randomized algorithm

with $\tilde{O}(m^{1.5})$ total update time

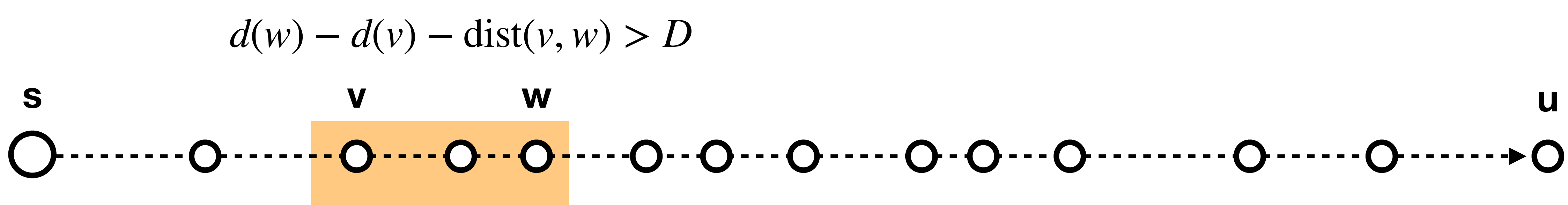
# Key idea

- Assume stretch $d(u) - \text{dist}(s, u) > 10\epsilon L$ at some point
- Then, stretch $d(w) - d(v) - \text{dist}(v, w) > D$
  for many subpaths from v to w



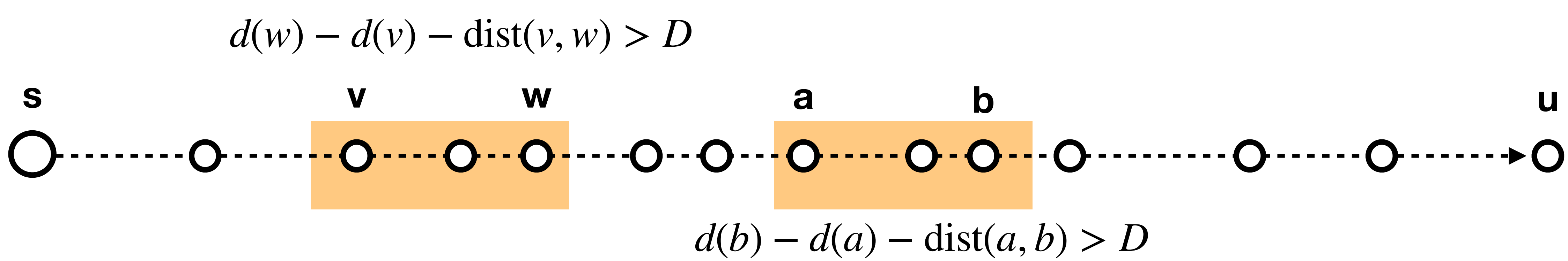
# Key idea

- Assume stretch $d(u) - \text{dist}(s, u) > 10\epsilon L$ at some point
- Then, stretch $d(w) - d(v) - \text{dist}(v, w) > D$ for many subpaths from v to w



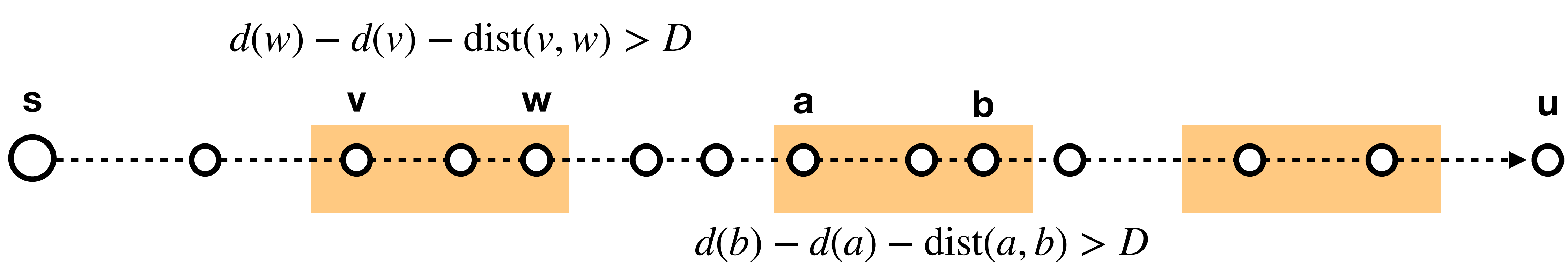
# Key idea

- Assume stretch $d(u) - \text{dist}(s, u) > 10\epsilon L$ at some point
- Then, stretch $d(w) - d(v) - \text{dist}(v, w) > D$ for many subpaths from v to w



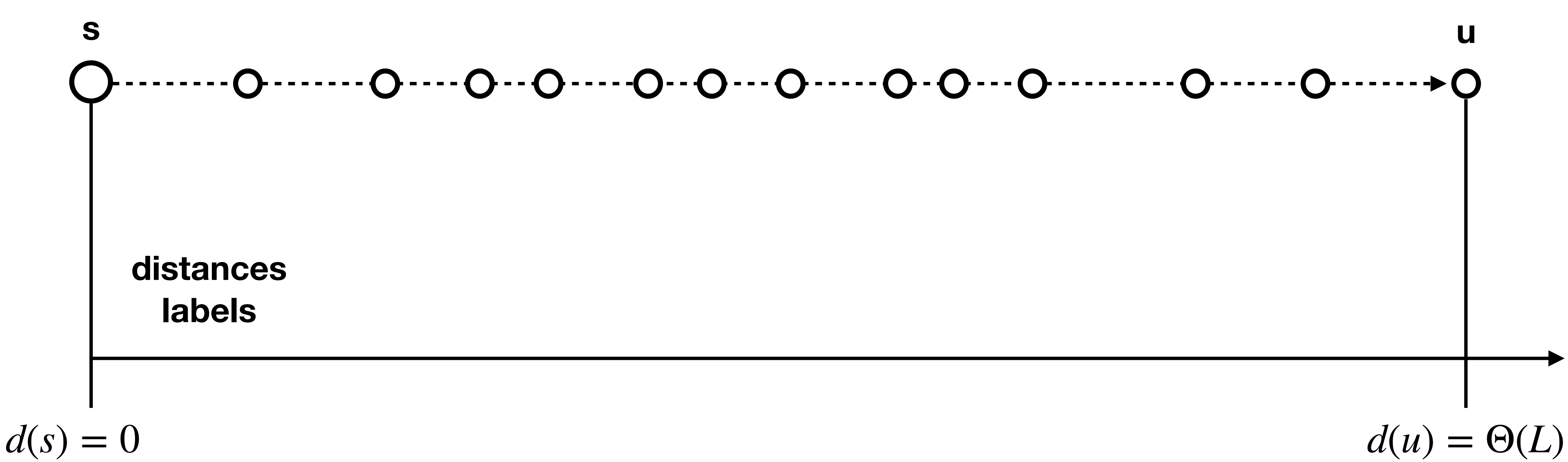
# Key idea

- Assume stretch $d(u) - \text{dist}(s,u) > 10\epsilon L$ at some point
- Then, stretch $d(w) - d(v) - \text{dist}(v,w) > D$
  for many subpaths from v to w



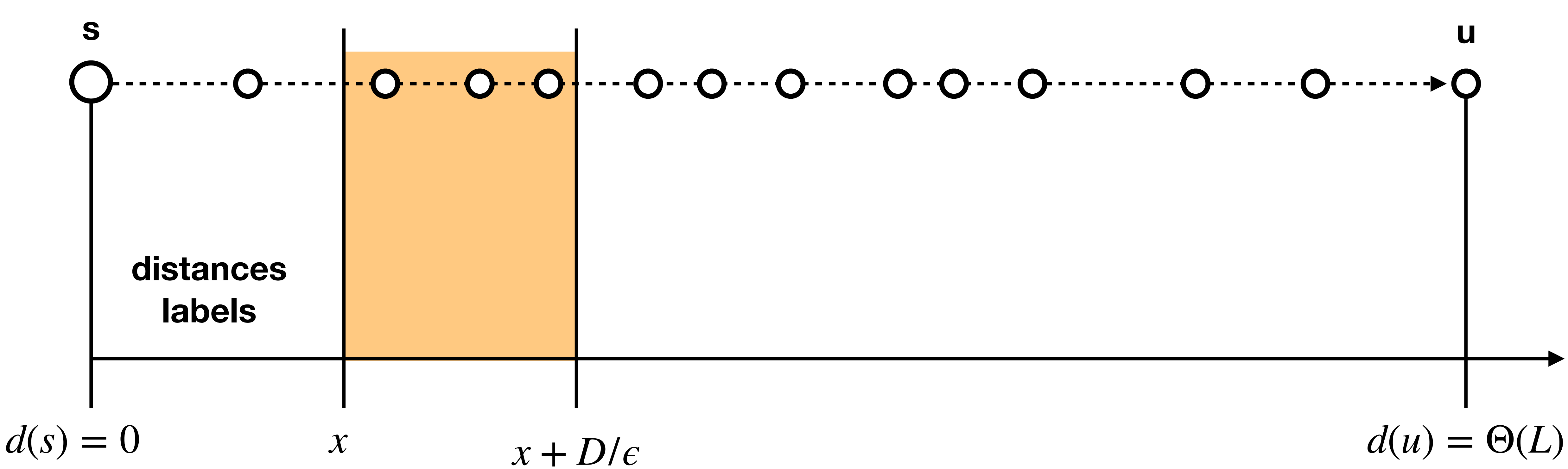
# Key idea

- Look at the interval [0, 3L]



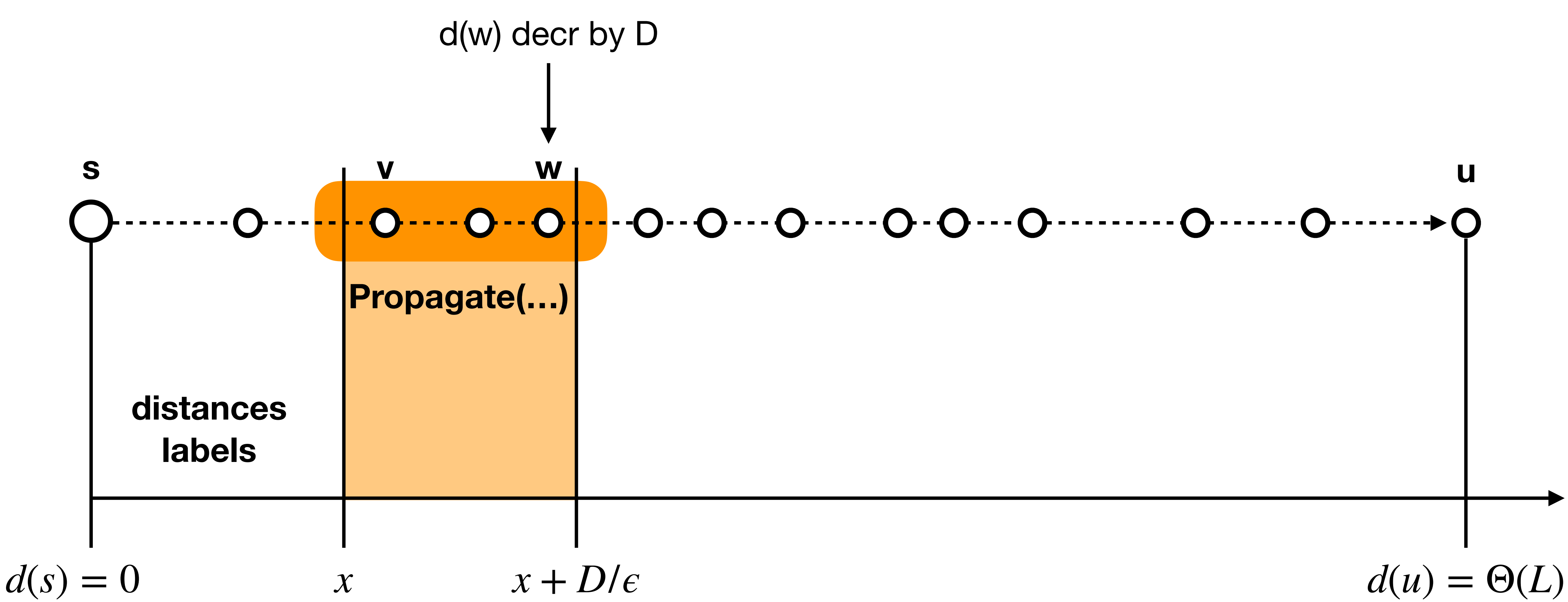
# Key idea

- Look at the interval [0, 3L]
- Randomly sample an interval $[x, x + D/\epsilon]$



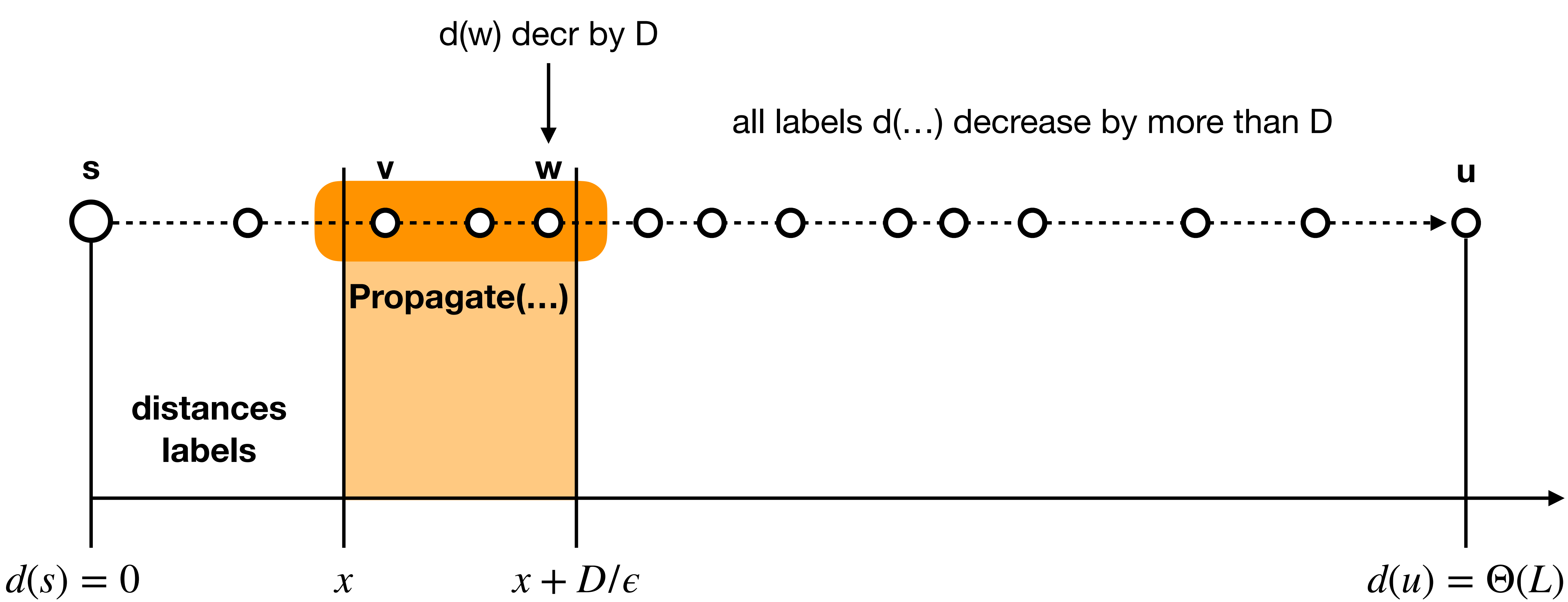
# Key idea

- Look at the interval [0, 3L]
- Randomly sample an interval $[x, x + D/\epsilon]$
- Call **Propagate**$(\{w \mid d(w) \in [x, x + D/\epsilon]\})$



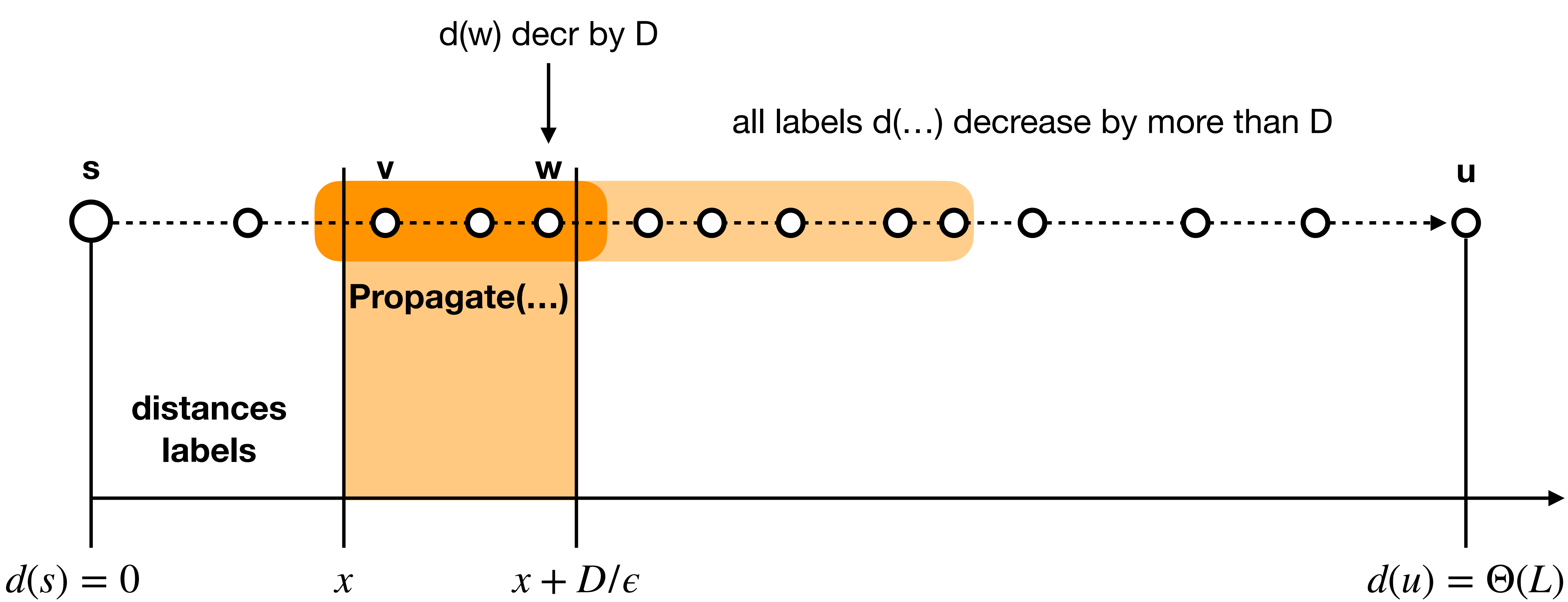
# Key idea

- Look at the interval [0, 3L]
- Randomly sample an interval $[x, x + D/\epsilon]$
- Call **Propagate**($\{w \mid d(w) \in [x, x + D/\epsilon]\}$)



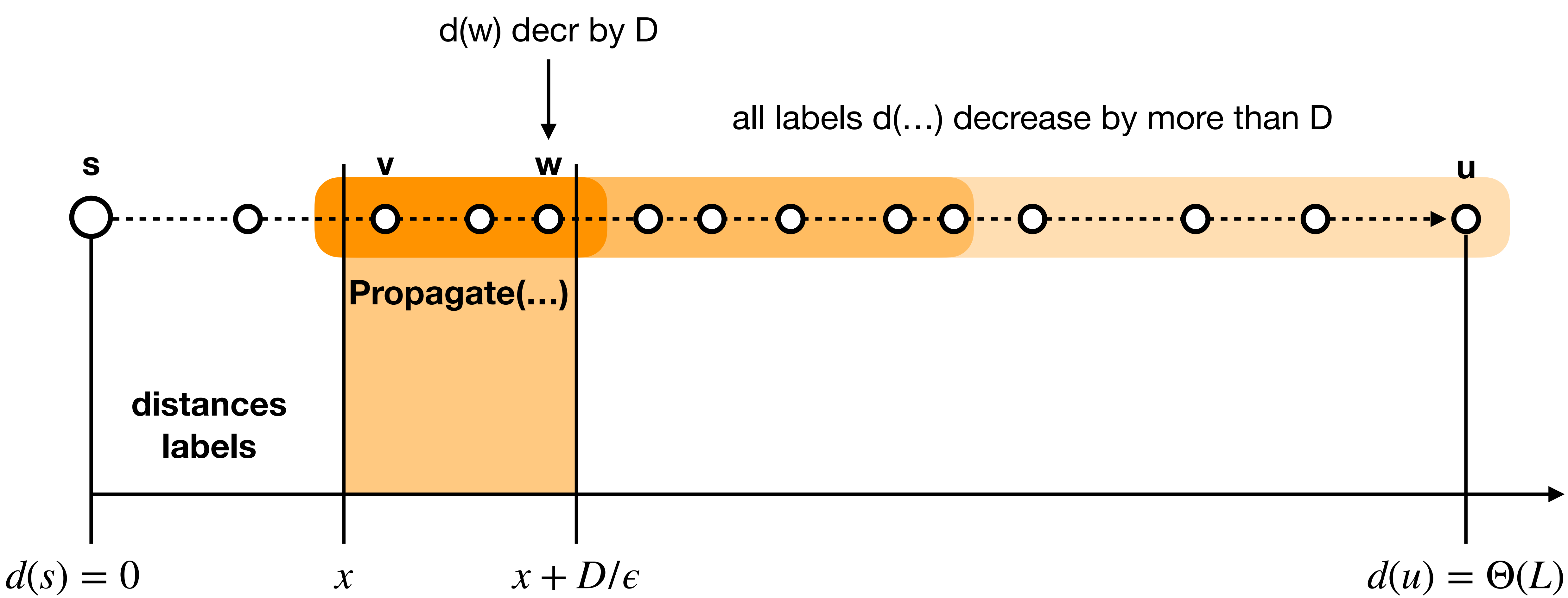
# Key idea

- Look at the interval [0, 3L]
- Randomly sample an interval $[x, x + D/\epsilon]$
- Call **Propagate**($\{w \mid d(w) \in [x, x + D/\epsilon]\}$)



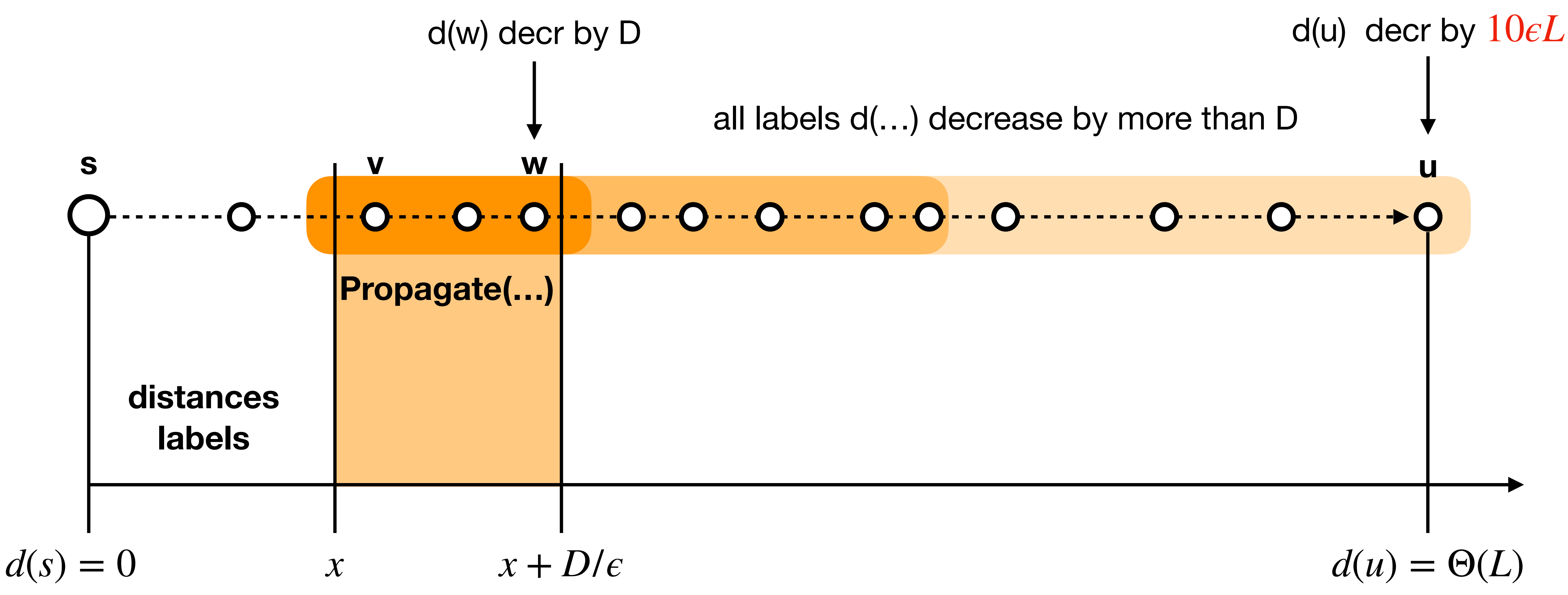
# Key idea

- Look at the interval [0, 3L]
- Randomly sample an interval $[x, x + D/\epsilon]$
- Call **Propagate**($\{w \mid d(w) \in [x, x + D/\epsilon]\}$)



# Key idea

- Look at the interval [0, 3L]
- Randomly sample an interval $[x, x + D/\epsilon]$
- Call **Propagate**$(\{w \mid d(w) \in [x, x + D/\epsilon]\})$



# Main algorithm

Pseudo-code

maintain dist labels $d(\cdot)$ for each $v \in V$

**Insert**(u, v):

If $d(v) - d(u) - \omega(u, v) \geq D$

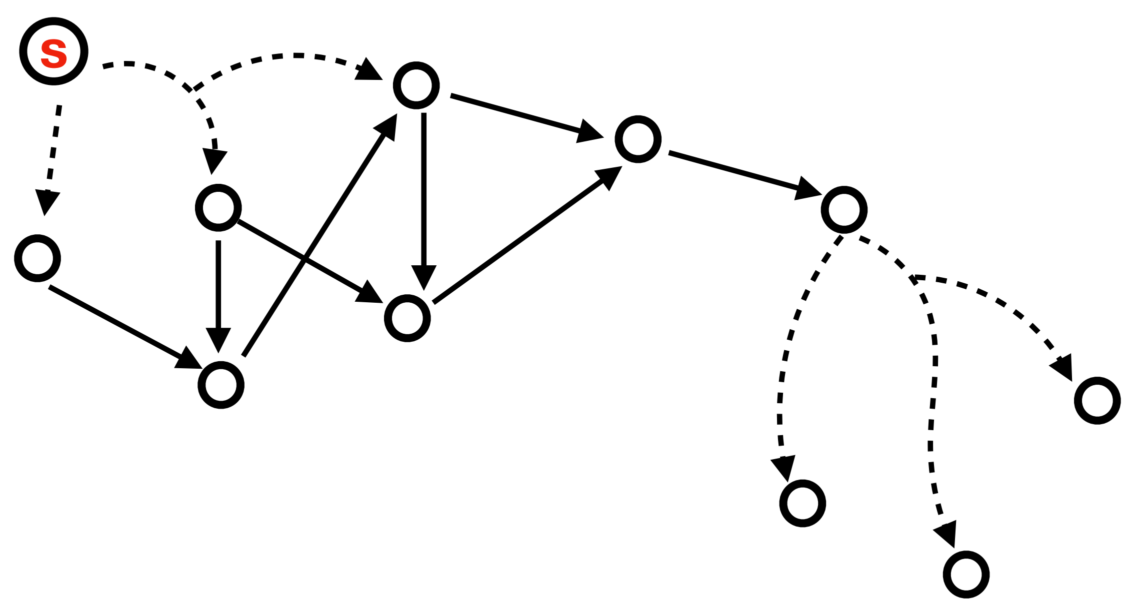
$d(v) \leftarrow \min\{d(u) + \omega(u, v), d(v)\}$

call **Propagate**( {v} )

uniformly sample $x \in [0, 2L]$

call **Propagate**( $\{w \mid d(w) \in [x, x + D/\epsilon]\}$ )

Running time of **Propagate**( $\{w \mid d(w) \in [x, x + D/\epsilon]\}$ )



# Main algorithm

Pseudo-code

maintain dist labels $d(\cdot)$ for each $v \in V$

**Insert**(u, v):

If $d(v) - d(u) - \omega(u, v) \geq D$

$d(v) \leftarrow \min\{d(u) + \omega(u, v), d(v)\}$

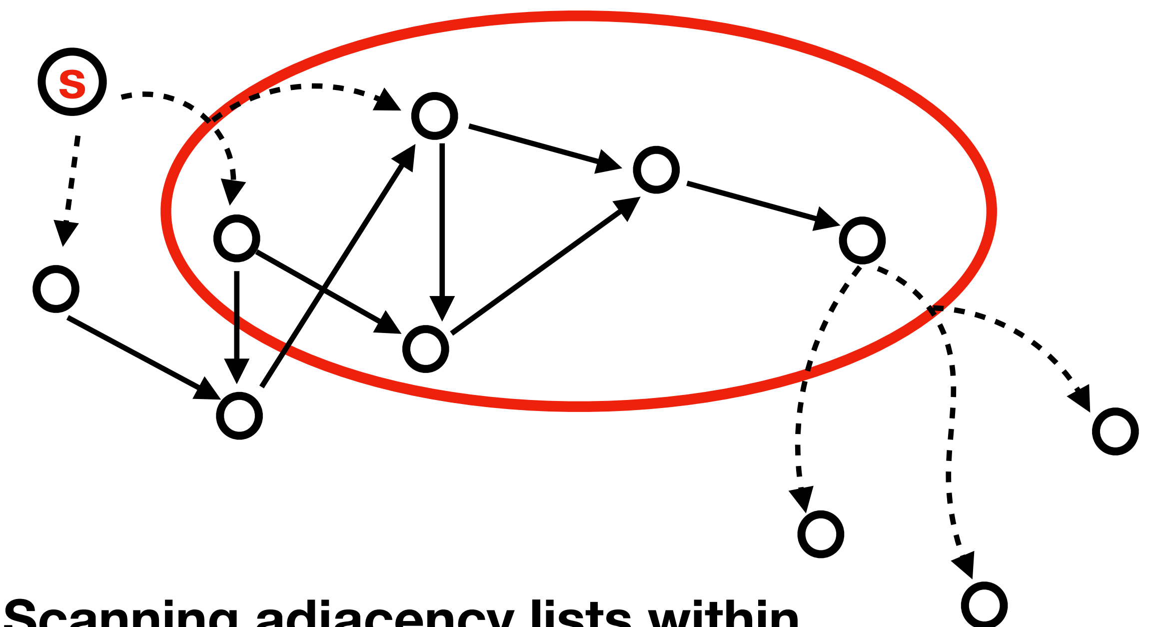
call **Propagate**( {v} )

uniformly sample $x \in [0,2L]$

call **Propagate**( $\{w \mid d(w) \in [x, x + D/\epsilon]\}$ )

Running time of

**Propagate**( $\{w \mid d(w) \in [x, x + D/\epsilon]\}$ )




**Scanning adjacency lists within**

$\{w \mid d(w) \in [x, x + D/\epsilon]\}$

**Time cost =** $mD/\epsilon L$ **each call**

# Main algorithm

Pseudo-code

maintain dist labels $d(\cdot)$ for each $v \in V$

**Insert**(u, v):

  If $d(v) - d(u) - \omega(u, v) \geq D$

    $d(v) \leftarrow \min\{d(u) + \omega(u, v), d(v)\}$

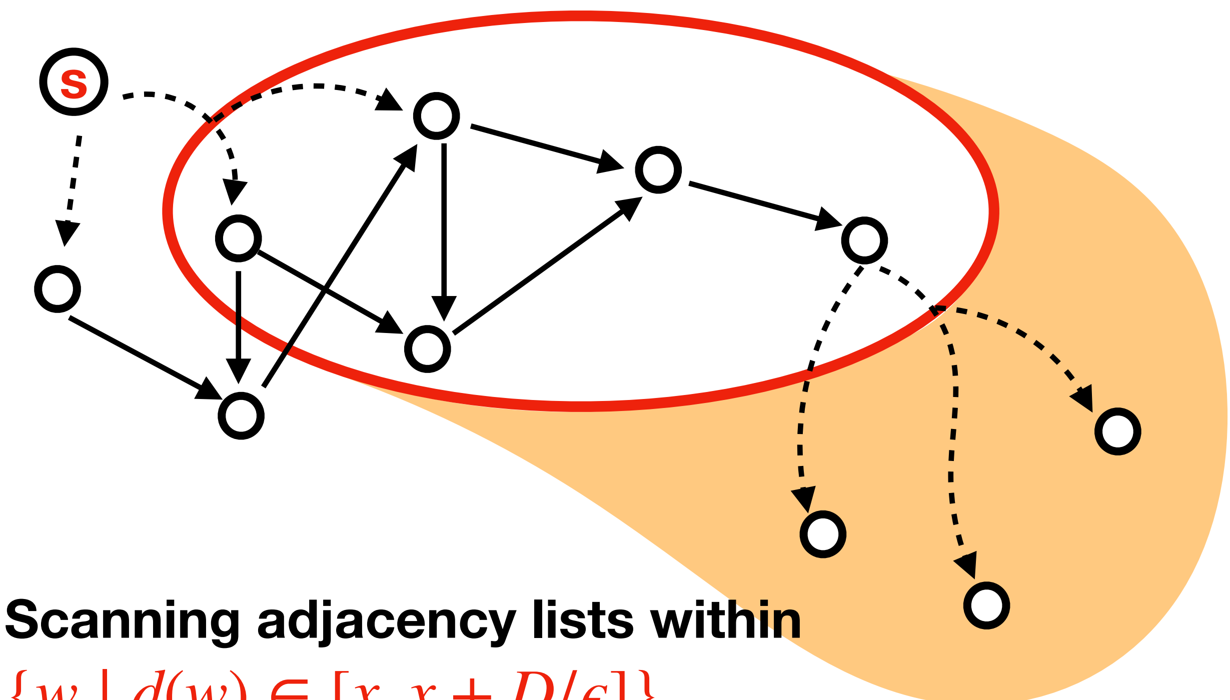
    call **Propagate**( {v} )

  uniformly sample $x \in [0, 2L]$

  call **Propagate**( $\{w \mid d(w) \in [x, x + D/\epsilon]\}$ )

Running time of

**Propagate**( $\{w \mid d(w) \in [x, x + D/\epsilon]\}$ )




**Scanning adjacency lists within**

$\{w \mid d(w) \in [x, x + D/\epsilon]\}$

**Time cost =** $mD/\epsilon L$ **each call**

**Propagation for decr-by-D vertices**

**Total Time cost =** $mL/D$

# Main algorithm

Pseudo-code

maintain dist labels $d(\cdot)$ for each $v \in V$

**Insert**(u, v):

If $d(v) - d(u) - \omega(u, v) \geq D$

$d(v) \leftarrow \min\{d(u) + \omega(u, v), d(v)\}$

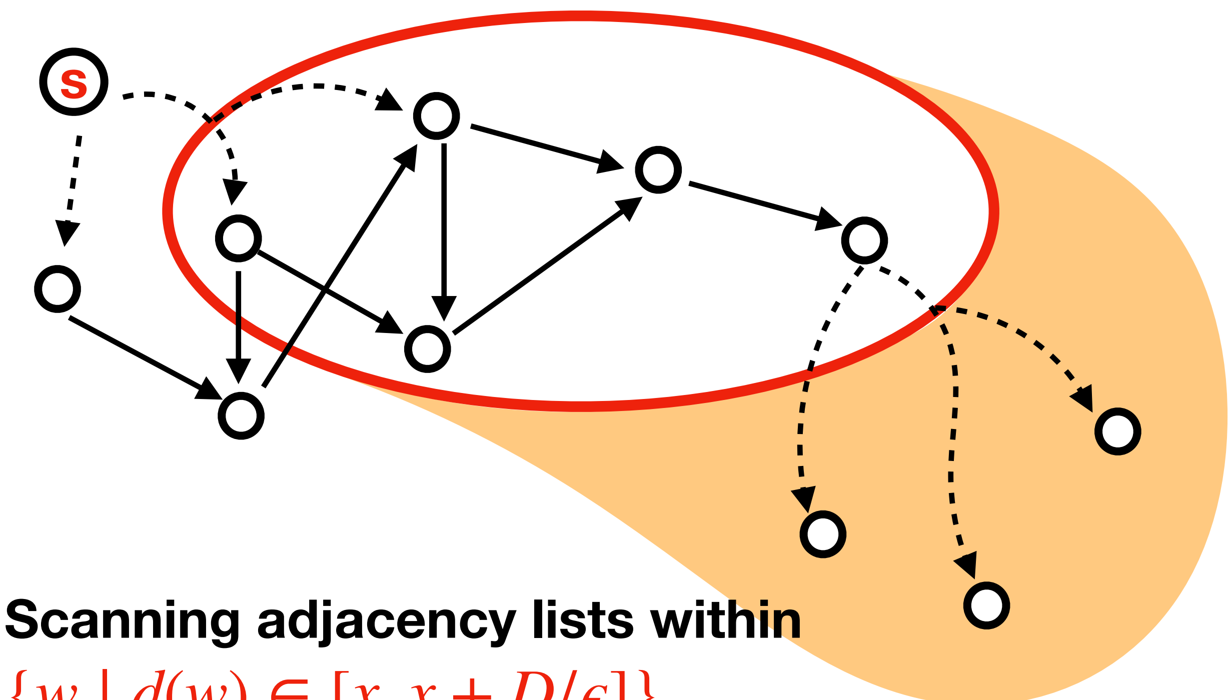
call **Propagate**( {v} )

uniformly sample $x \in [0,2L]$

call **Propagate**( $\{w \mid d(w) \in [x, x + D/\epsilon]\}$ )

Running time of

**Propagate**( $\{w \mid d(w) \in [x, x + D/\epsilon]\}$ )




**Scanning adjacency lists within**

$\{w \mid d(w) \in [x, x + D/\epsilon]\}$

**Time cost =** $mD/\epsilon L$ **each call**

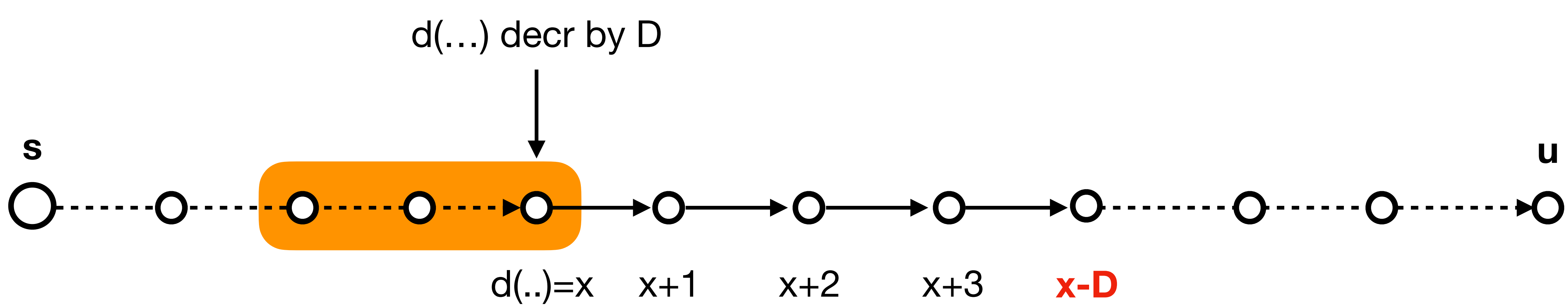
**Propagation for decr-by-D vertices**

**Total Time cost =** $mL/D$

**Total update time =** $m^2 D/\epsilon L + mL/D = m^{1.5}$

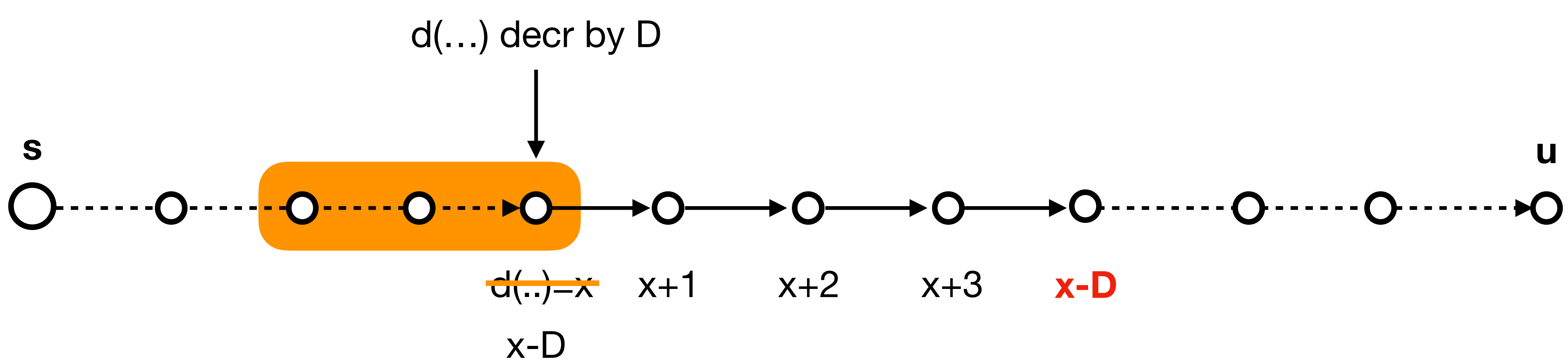
# Proof of correctness

- Main difficulty: propagation might stop early



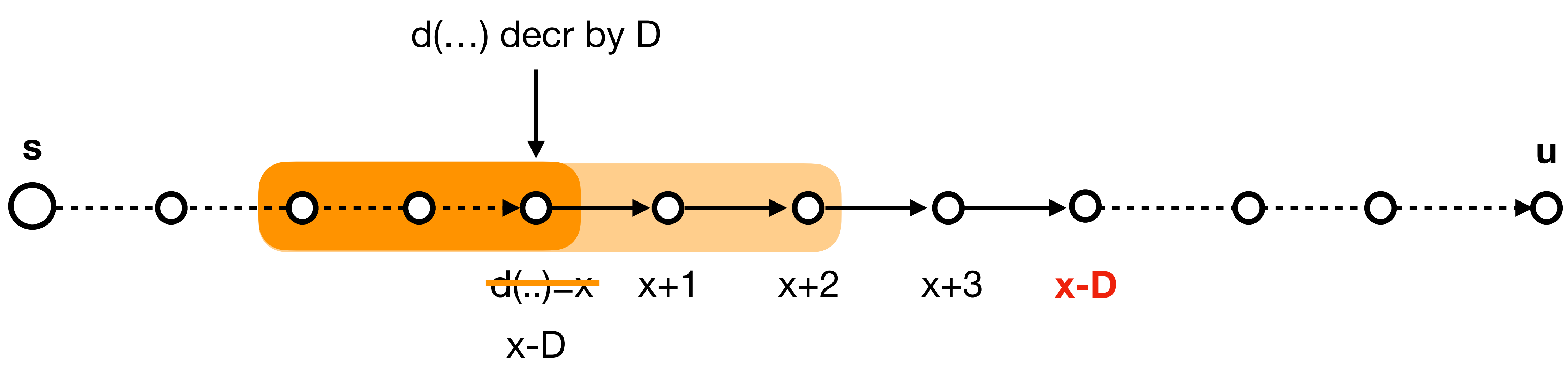
# Proof of correctness

- Main difficulty: propagation might stop early



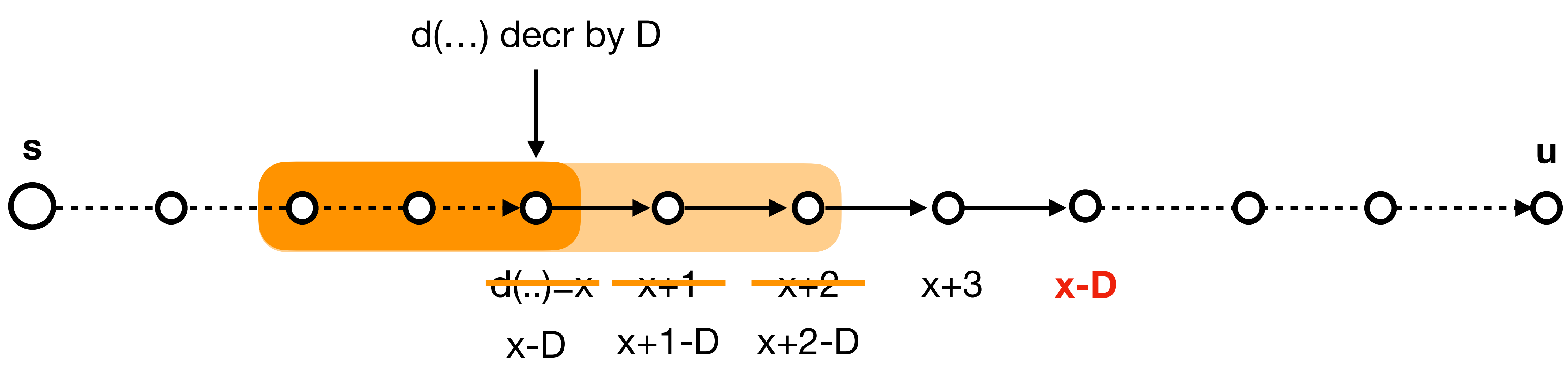
# Proof of correctness

- Main difficulty: propagation might stop early



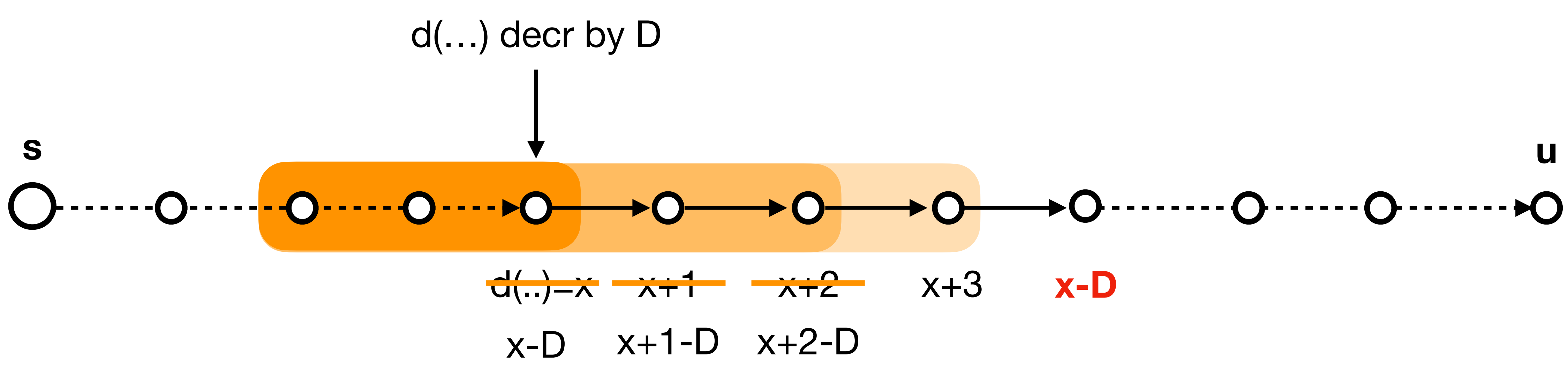
# Proof of correctness

- Main difficulty: propagation might stop early



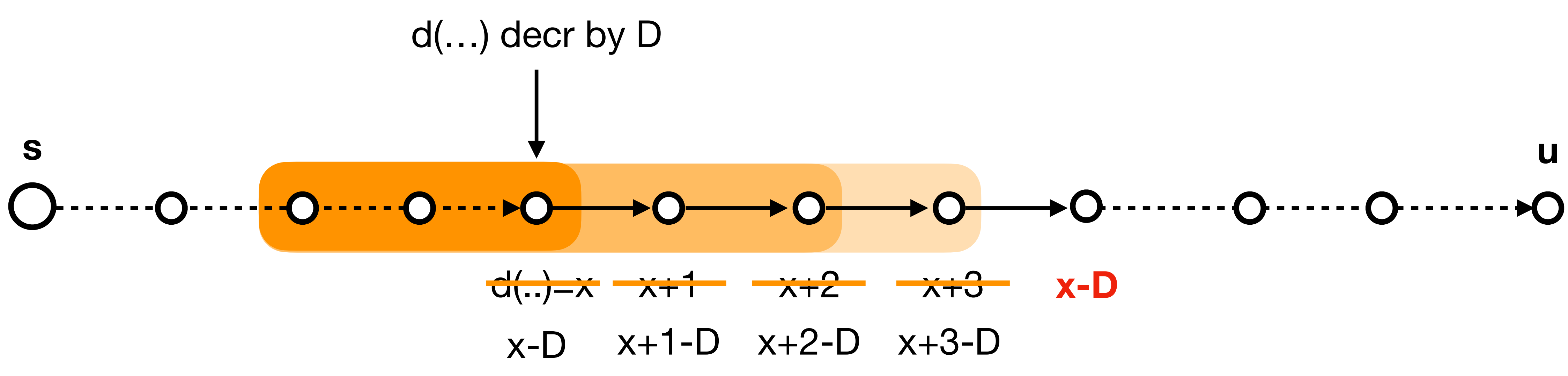
# Proof of correctness

- Main difficulty: propagation might stop early



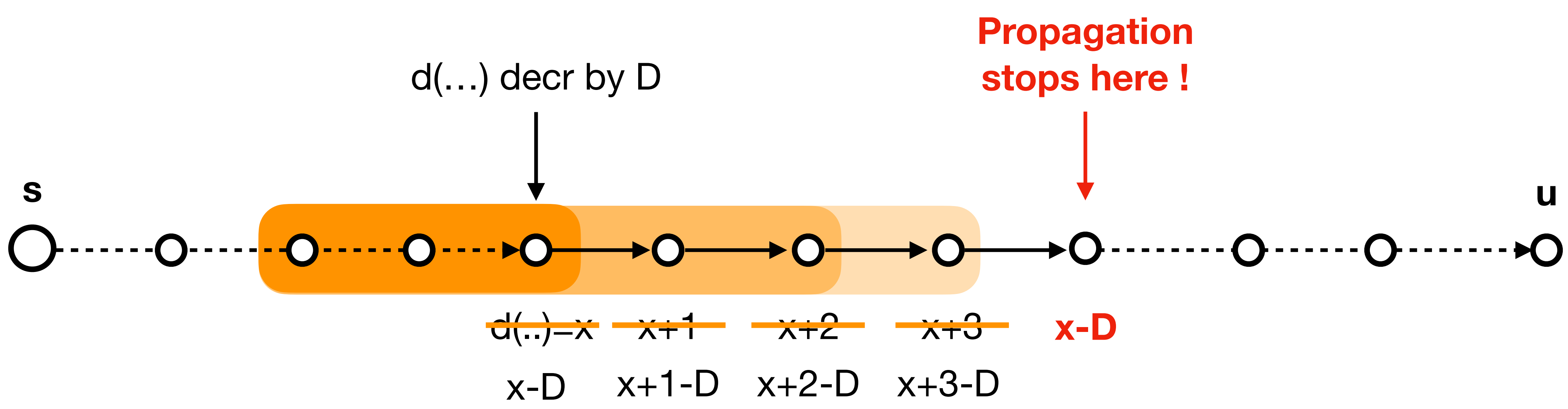
# Proof of correctness

- Main difficulty: propagation might stop early



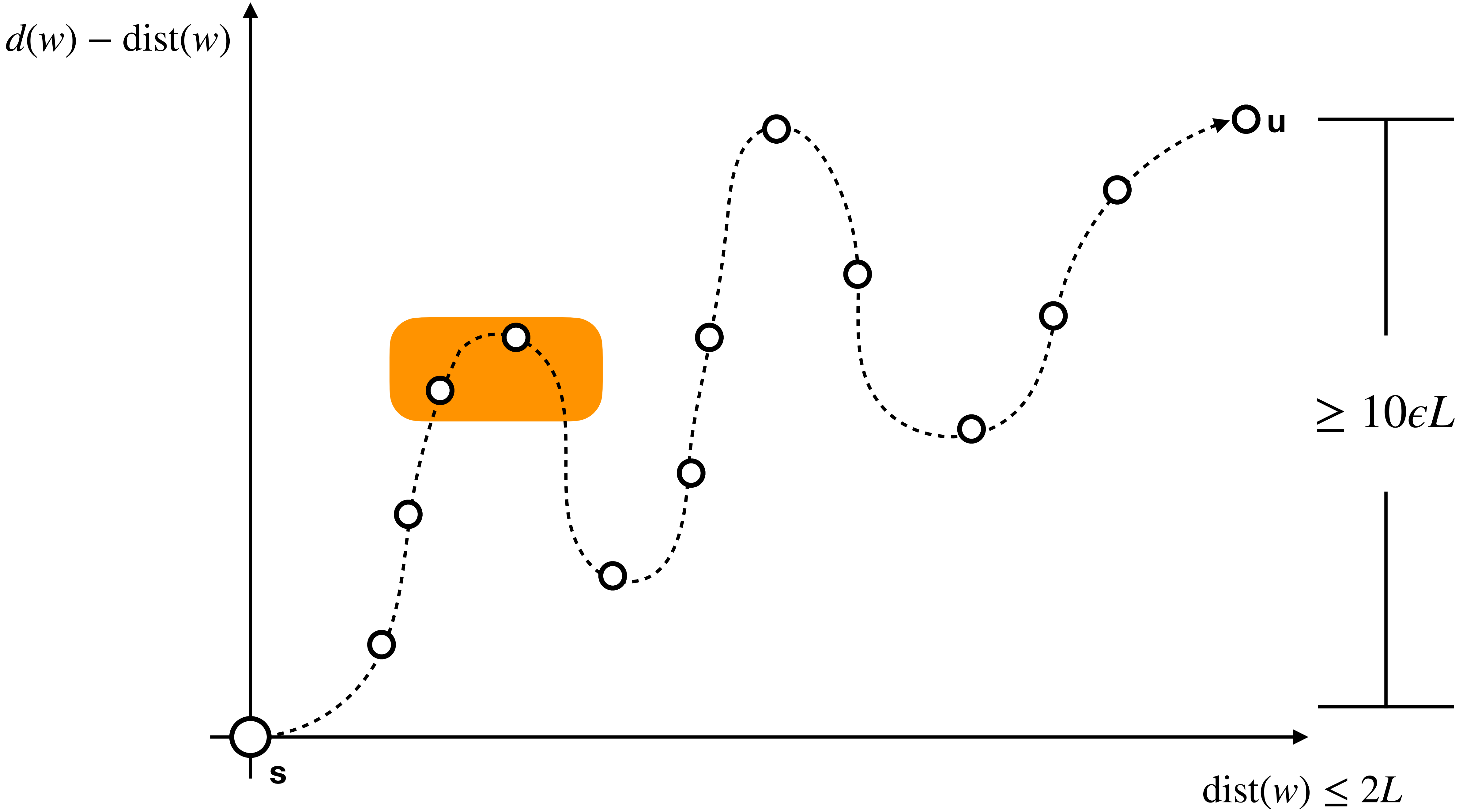
# Proof of correctness

- Main difficulty: propagation might stop early



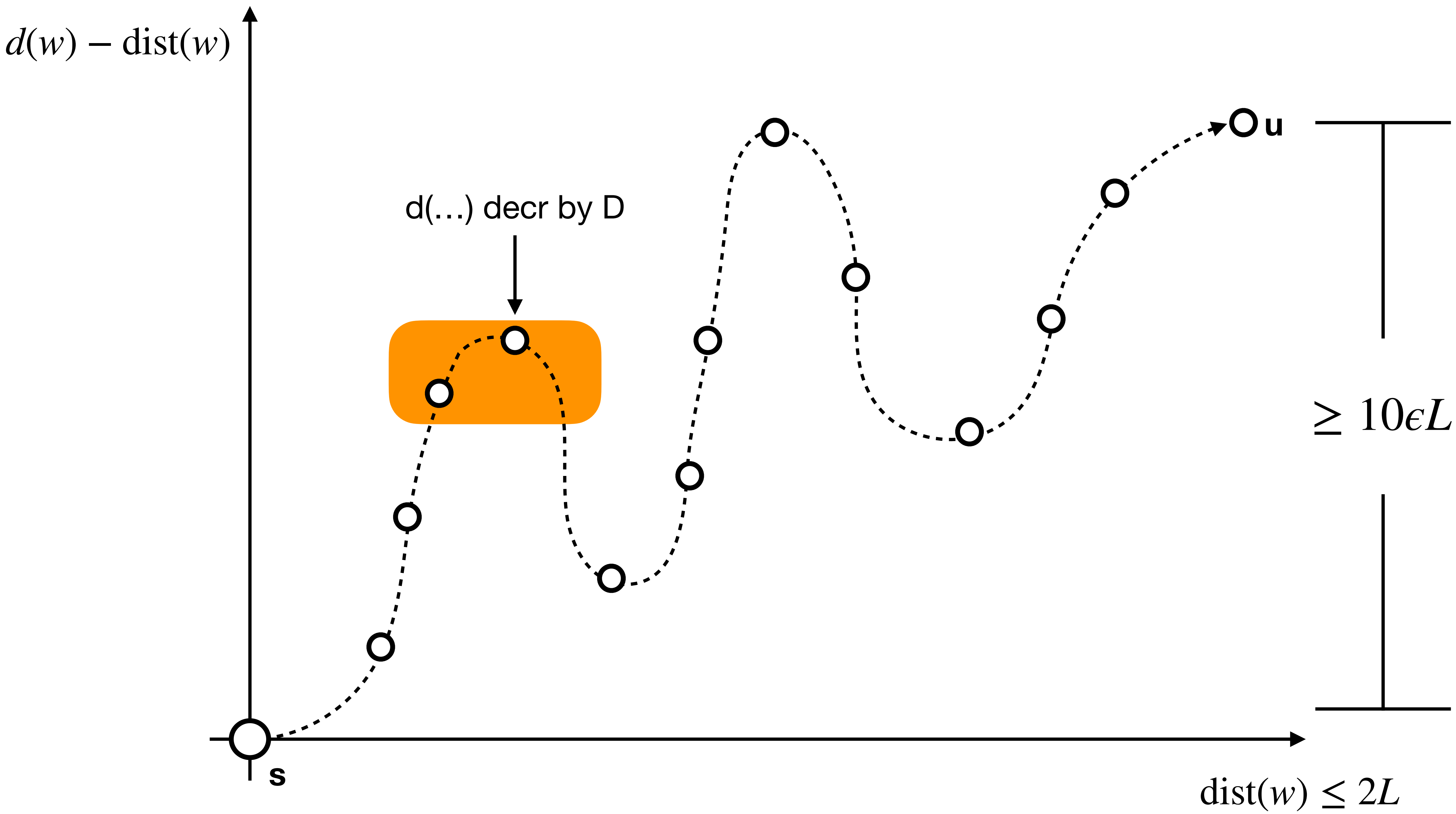
# Proof of correctness

- Main difficulty: propagation might stop early



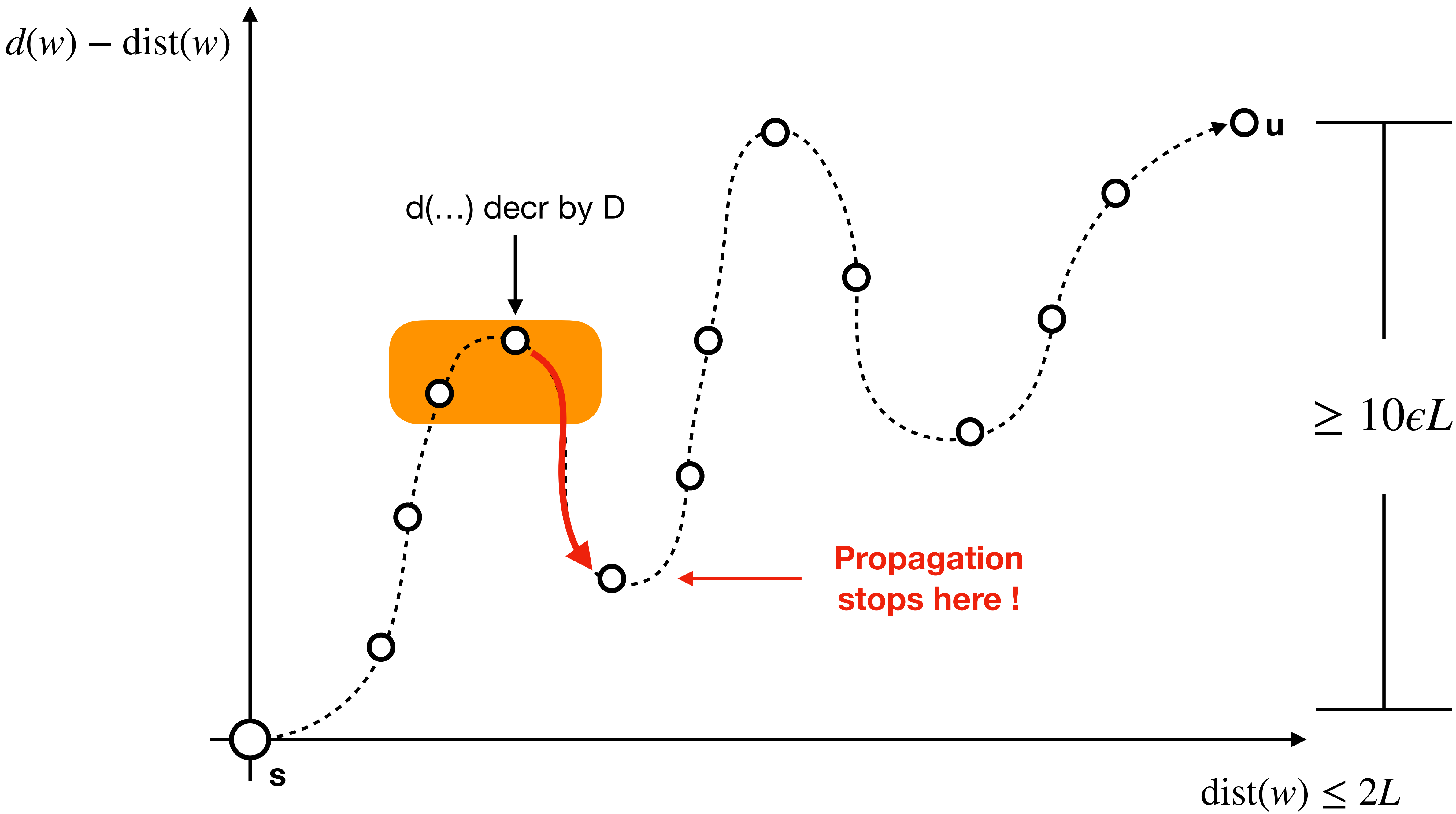
# Proof of correctness

- Main difficulty: propagation might stop early



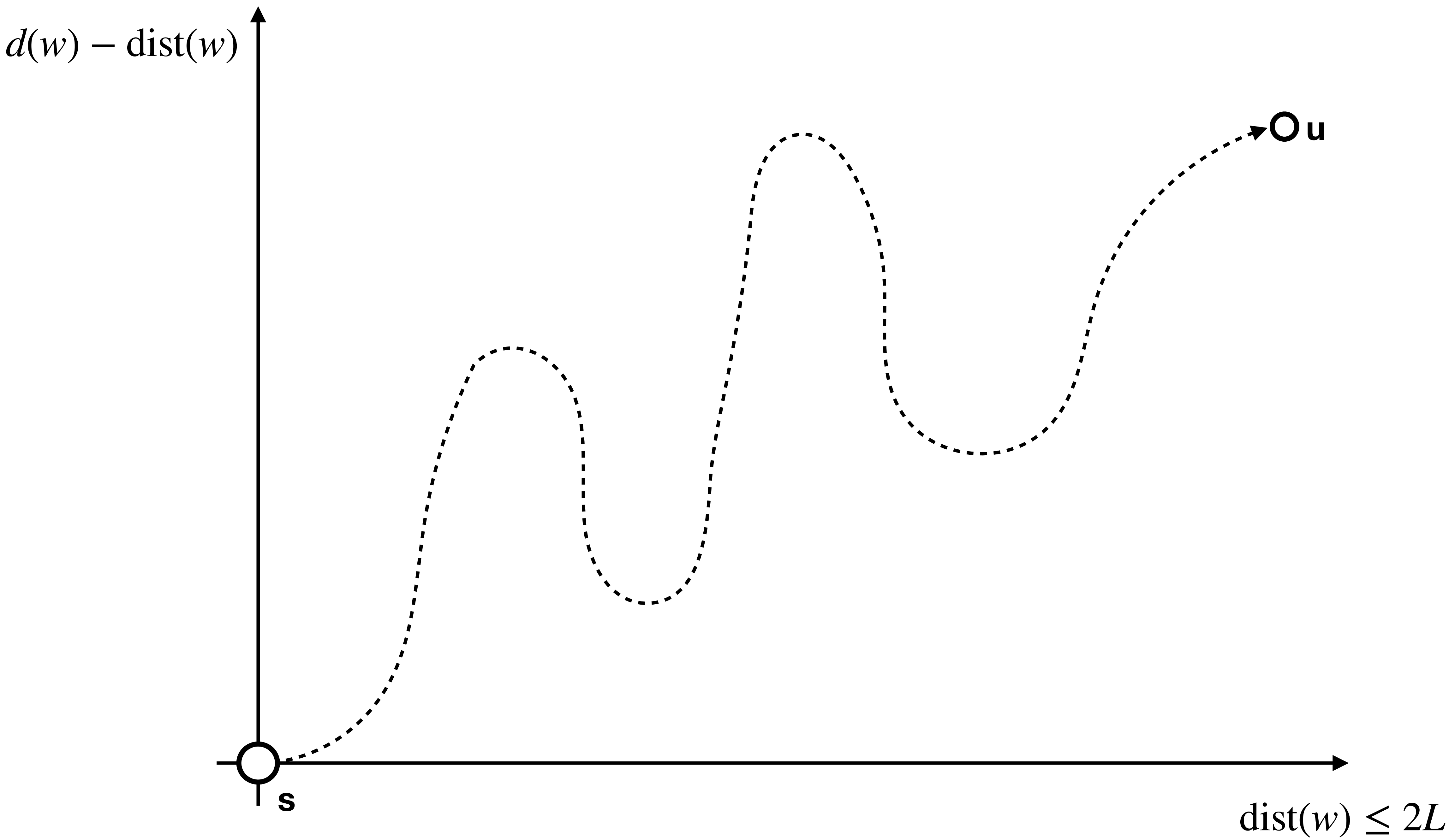
# Proof of correctness

- Main difficulty: propagation might stop early



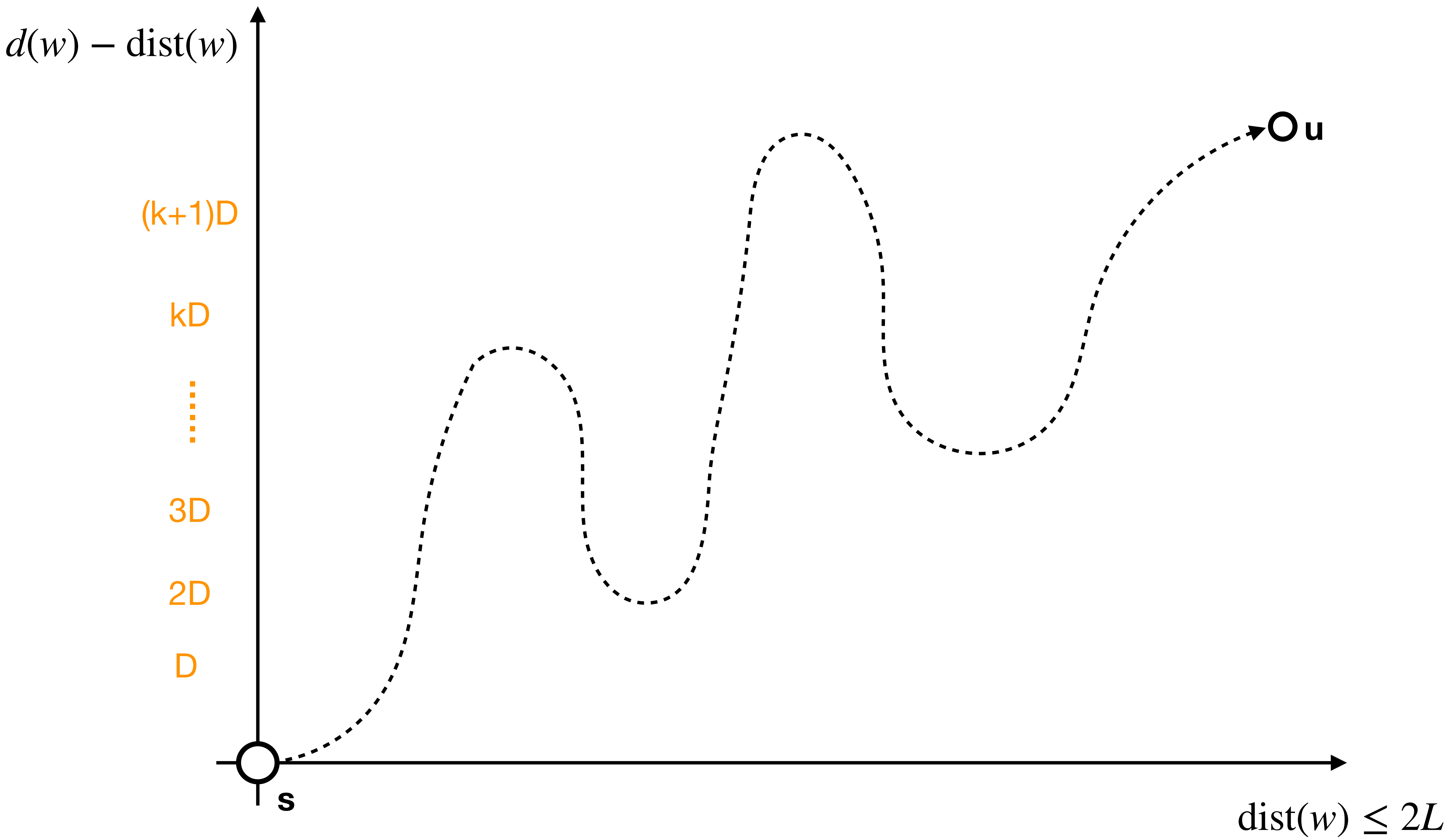
# Proof of correctness

- Where does **Propagation** succeed?



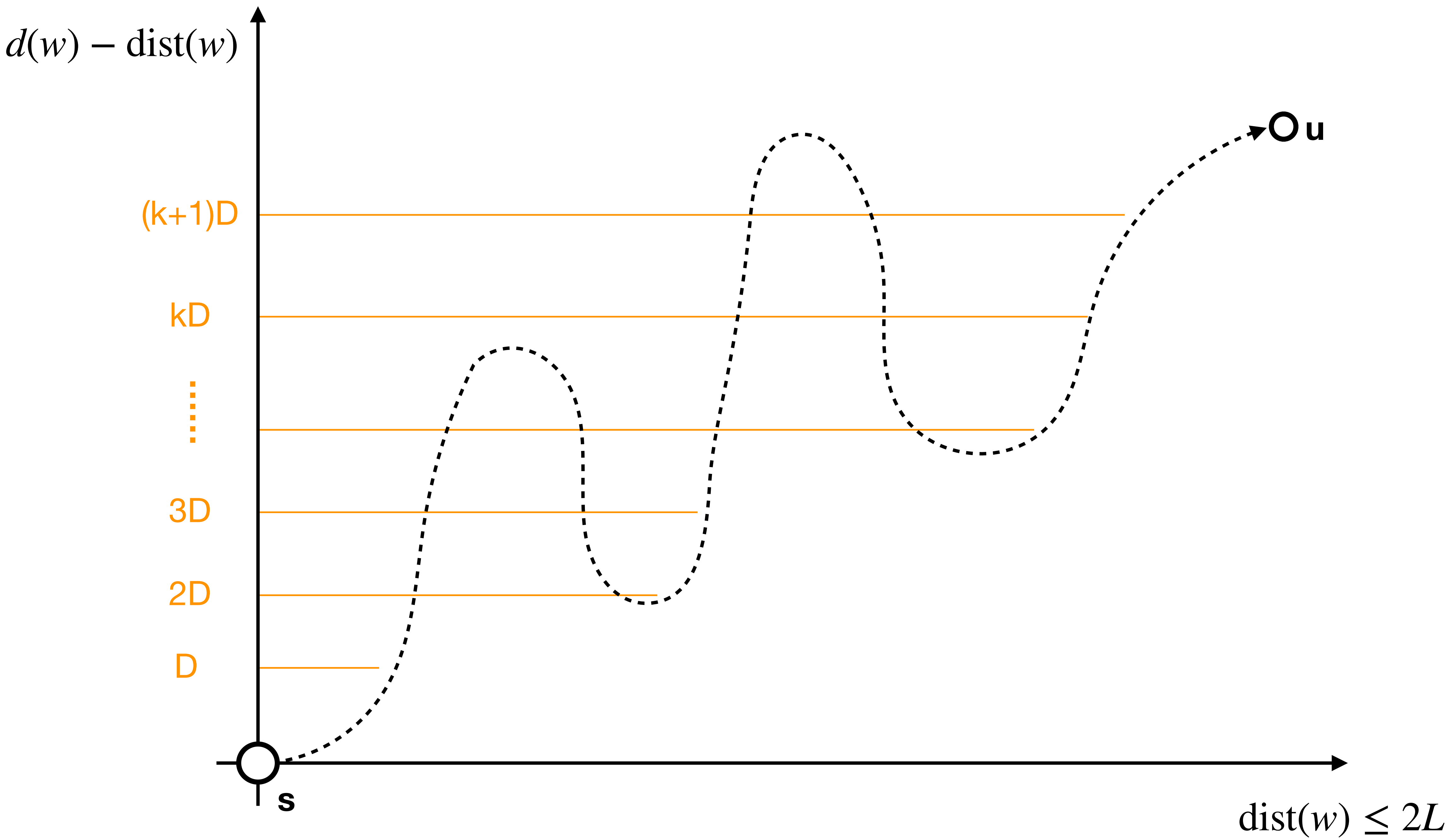
# Proof of correctness

- Where does **Propagation** succeed?



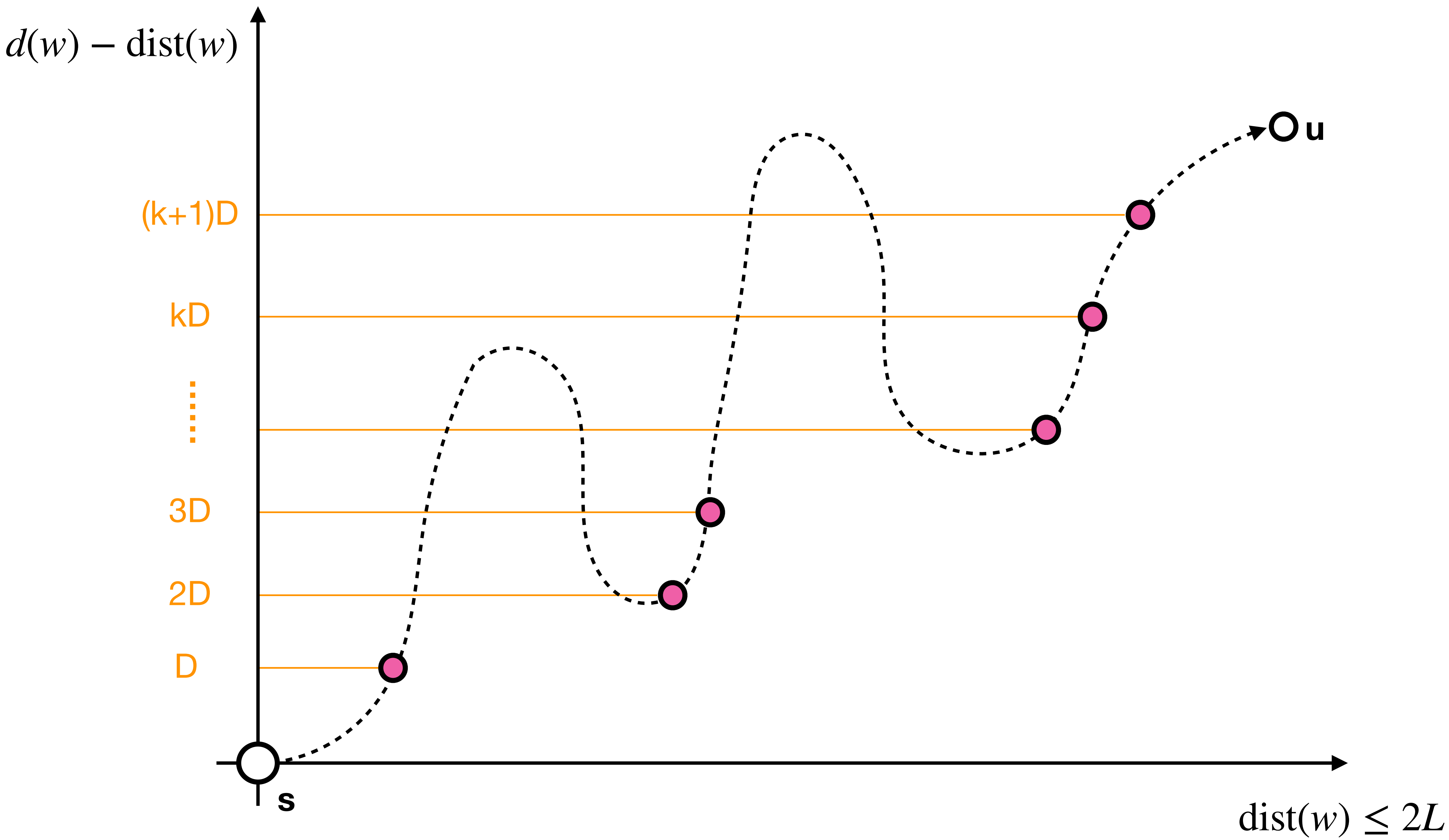
# Proof of correctness

- Where does **Propagation** succeed?



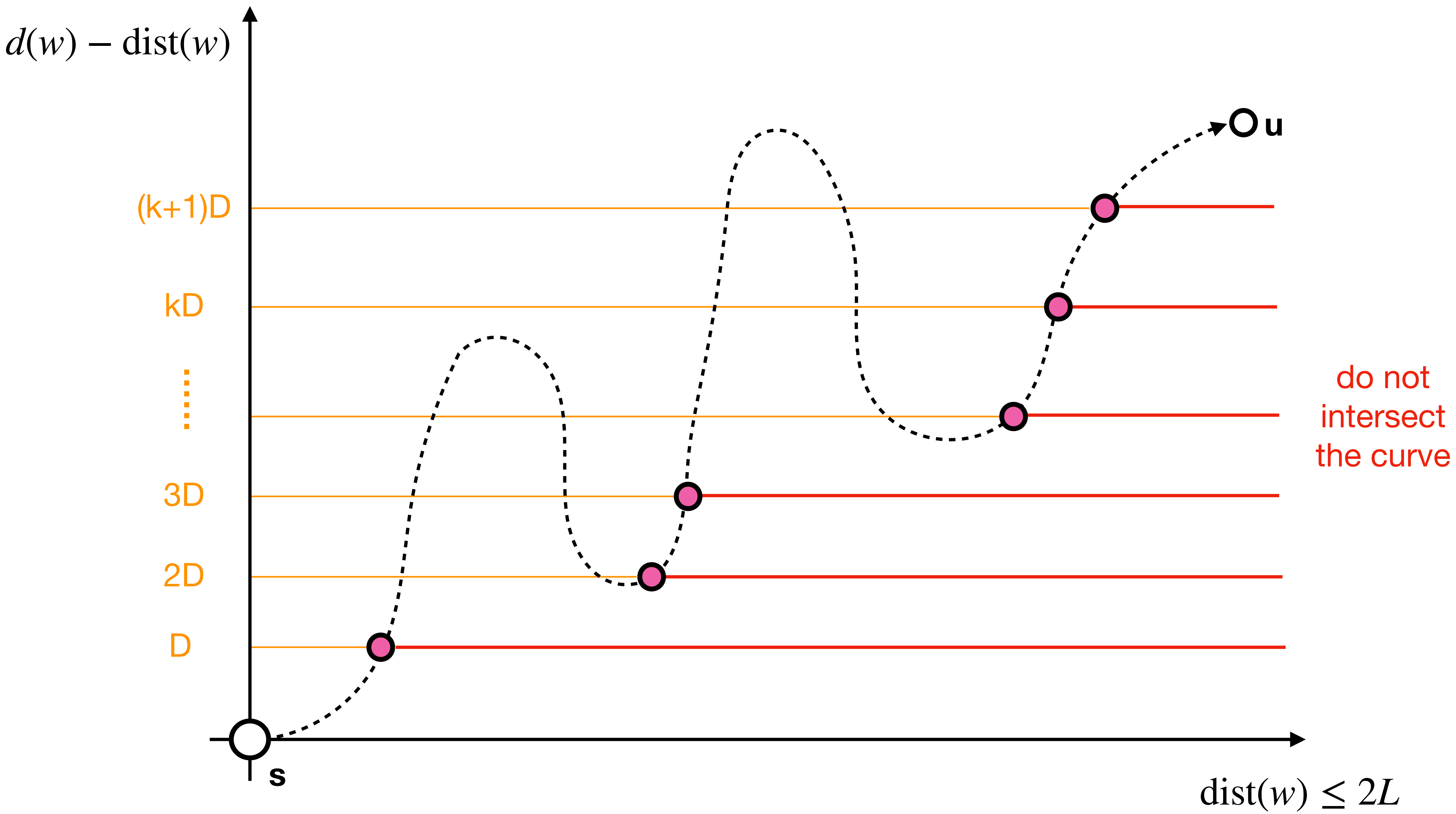
# Proof of correctness

- Where does **Propagation** succeed?



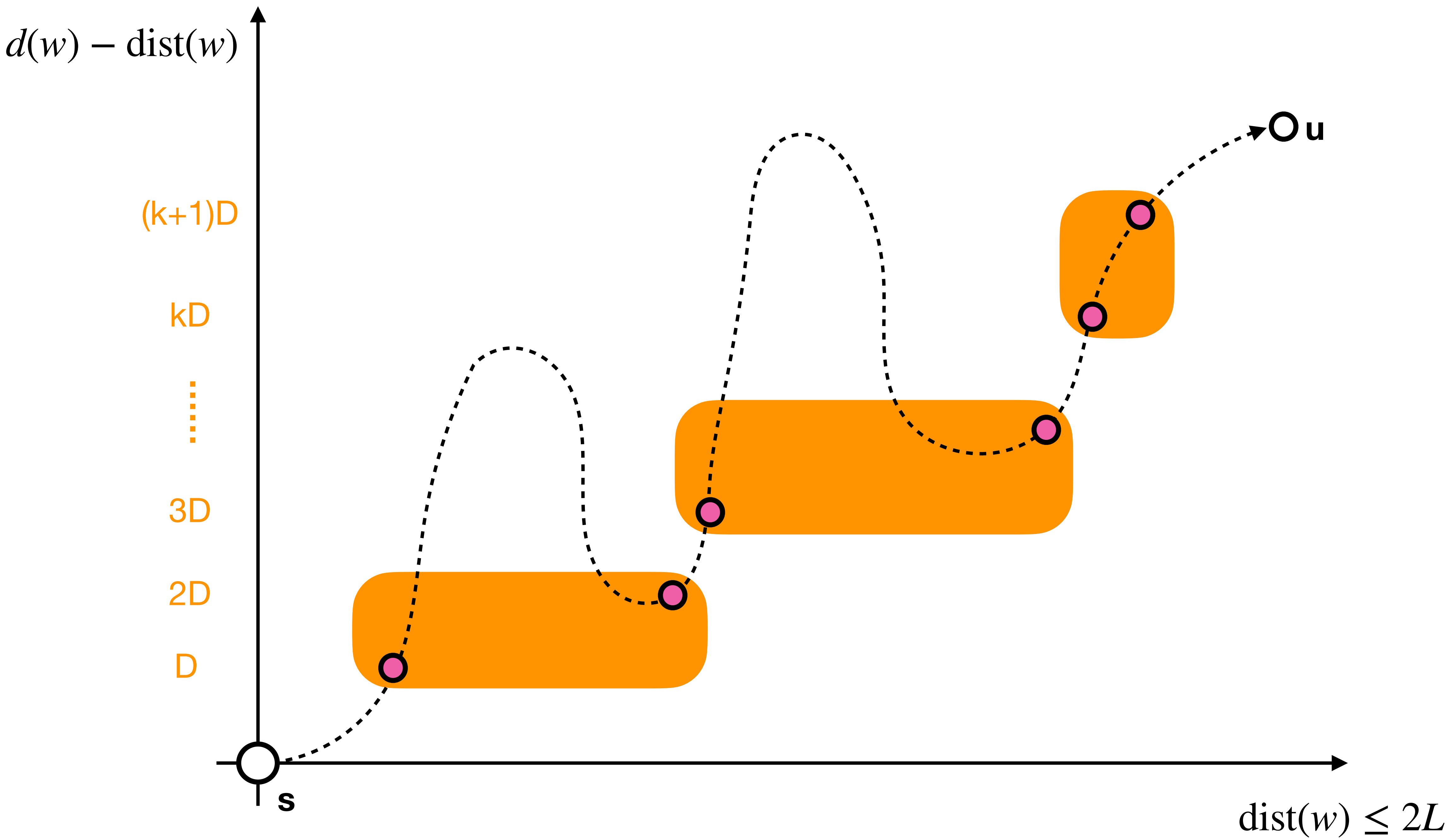
# Proof of correctness

- Where does **Propagation** succeed?



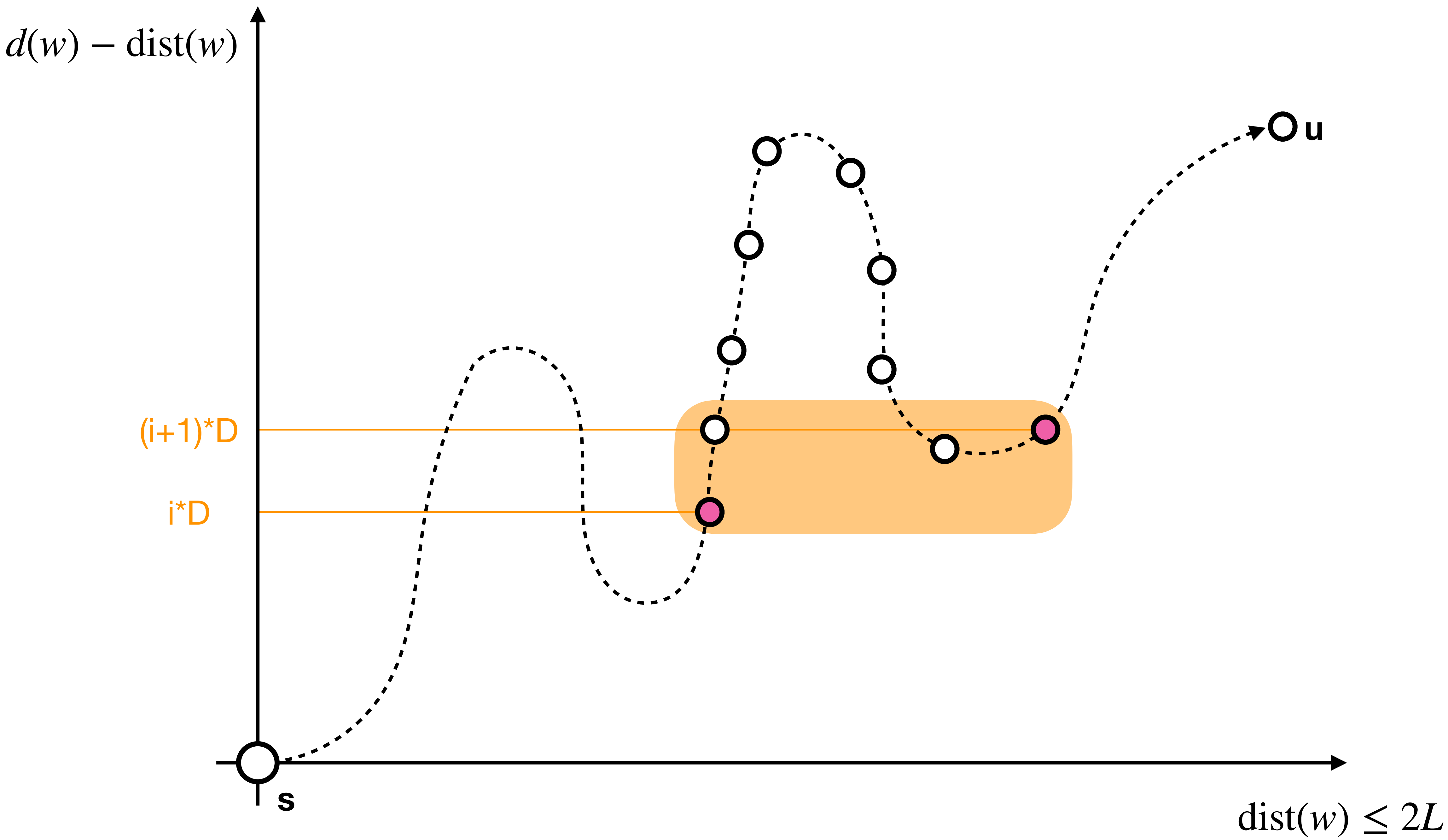
# Proof of correctness

- **Propagation** could succeed at these places



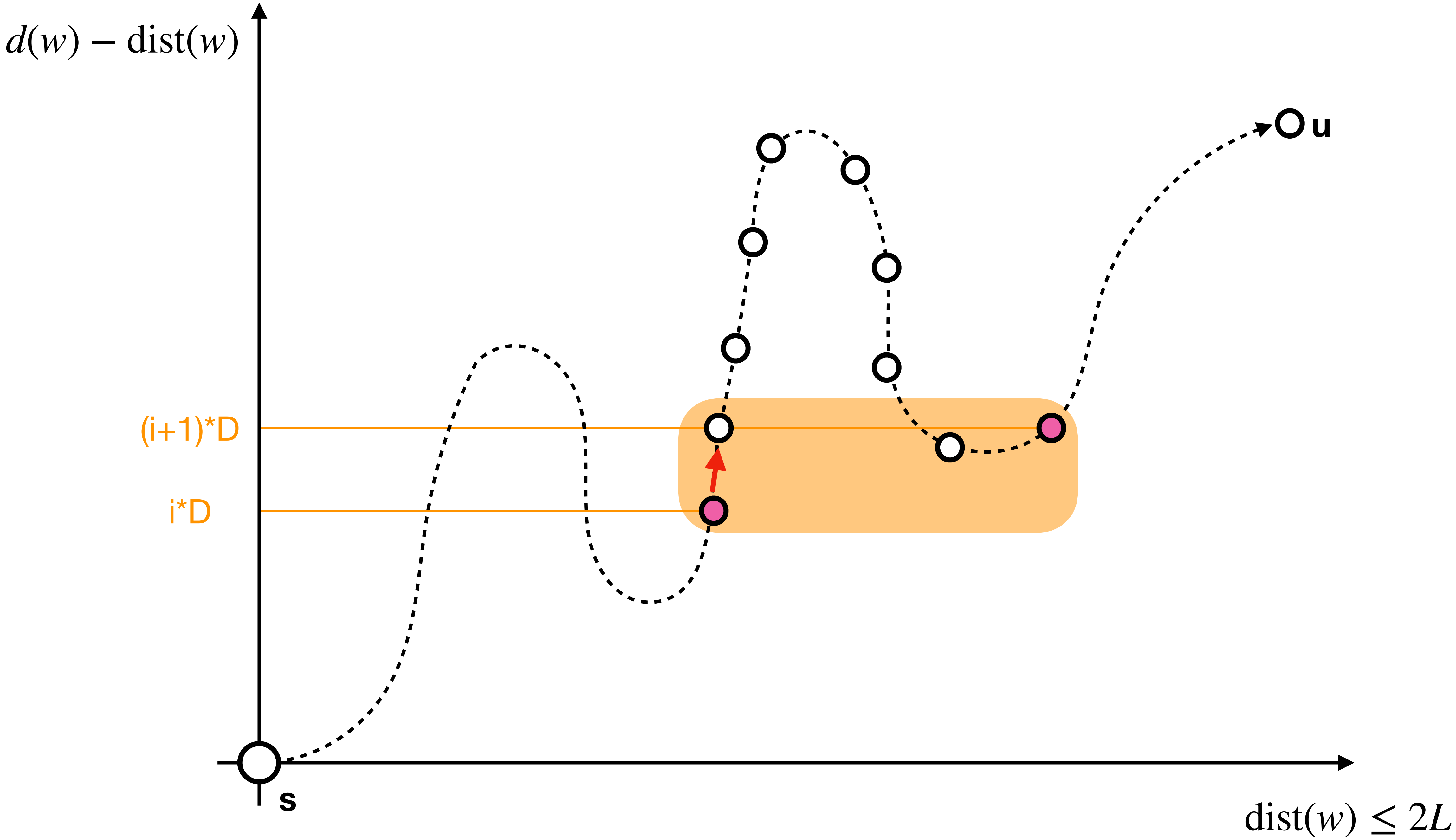
# Proof of correctness

- **Propagation** could succeed at these places



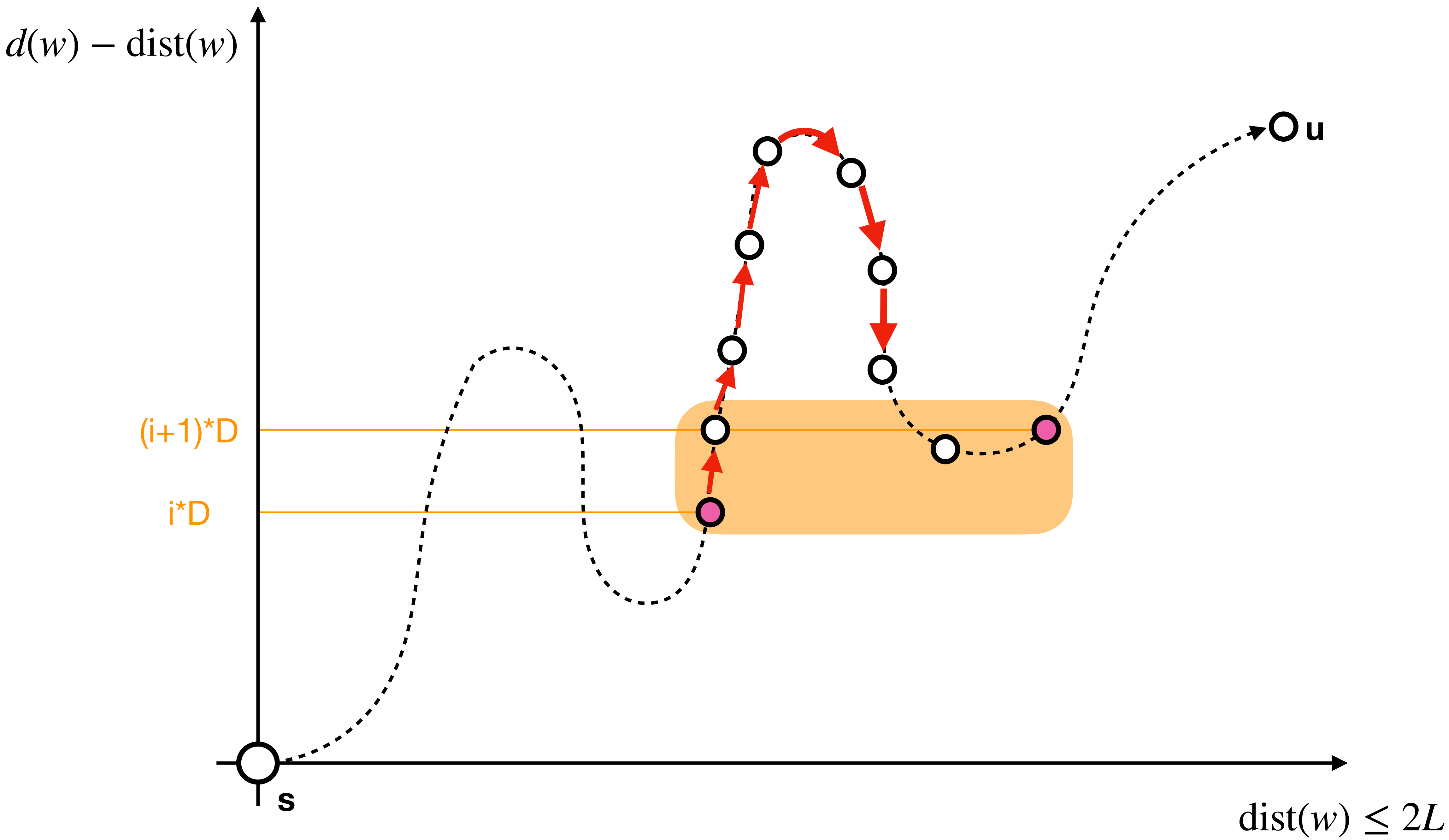
# Proof of correctness

- **Propagation** could succeed at these places



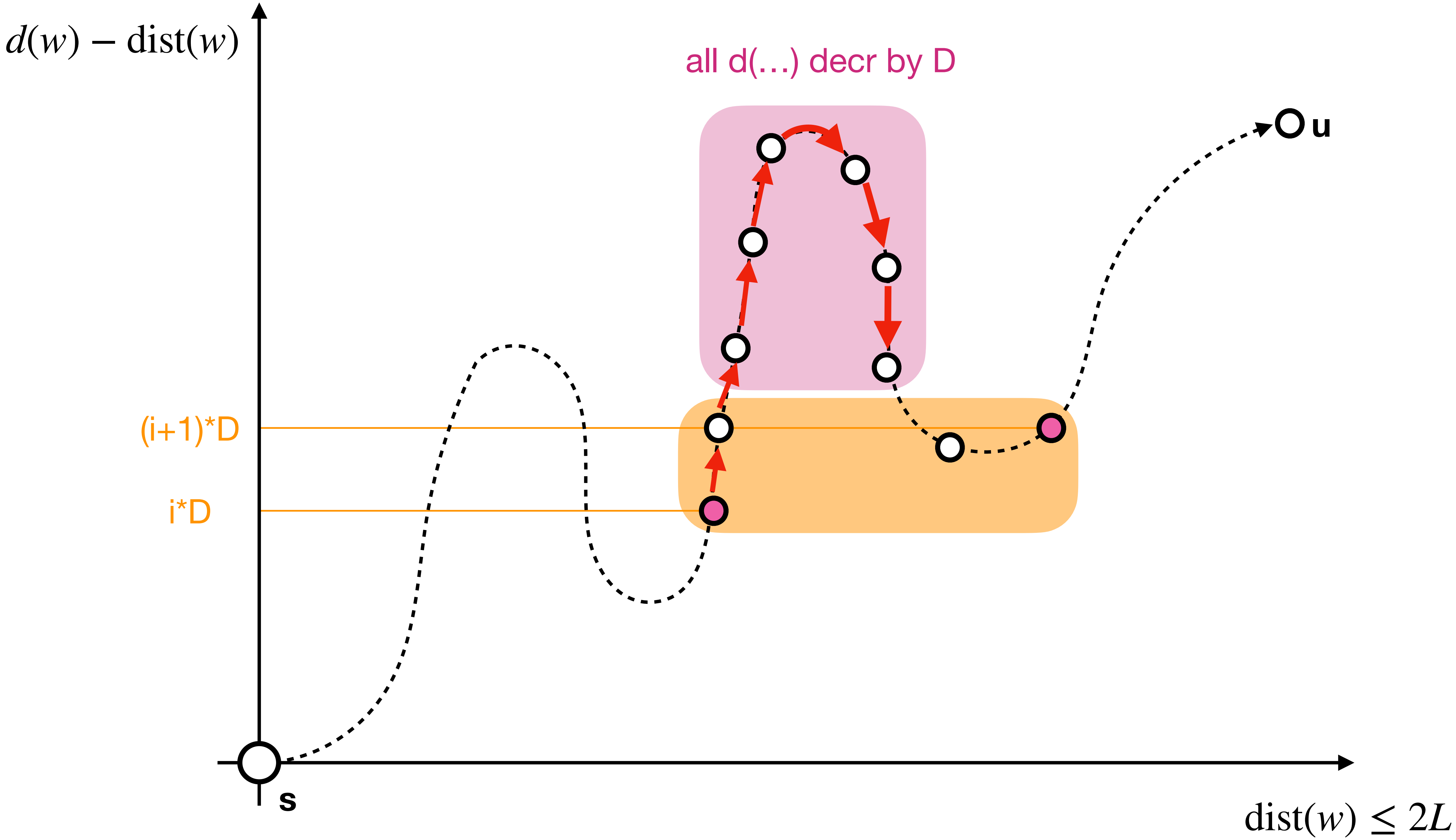
# Proof of correctness

- **Propagation** could succeed at these places



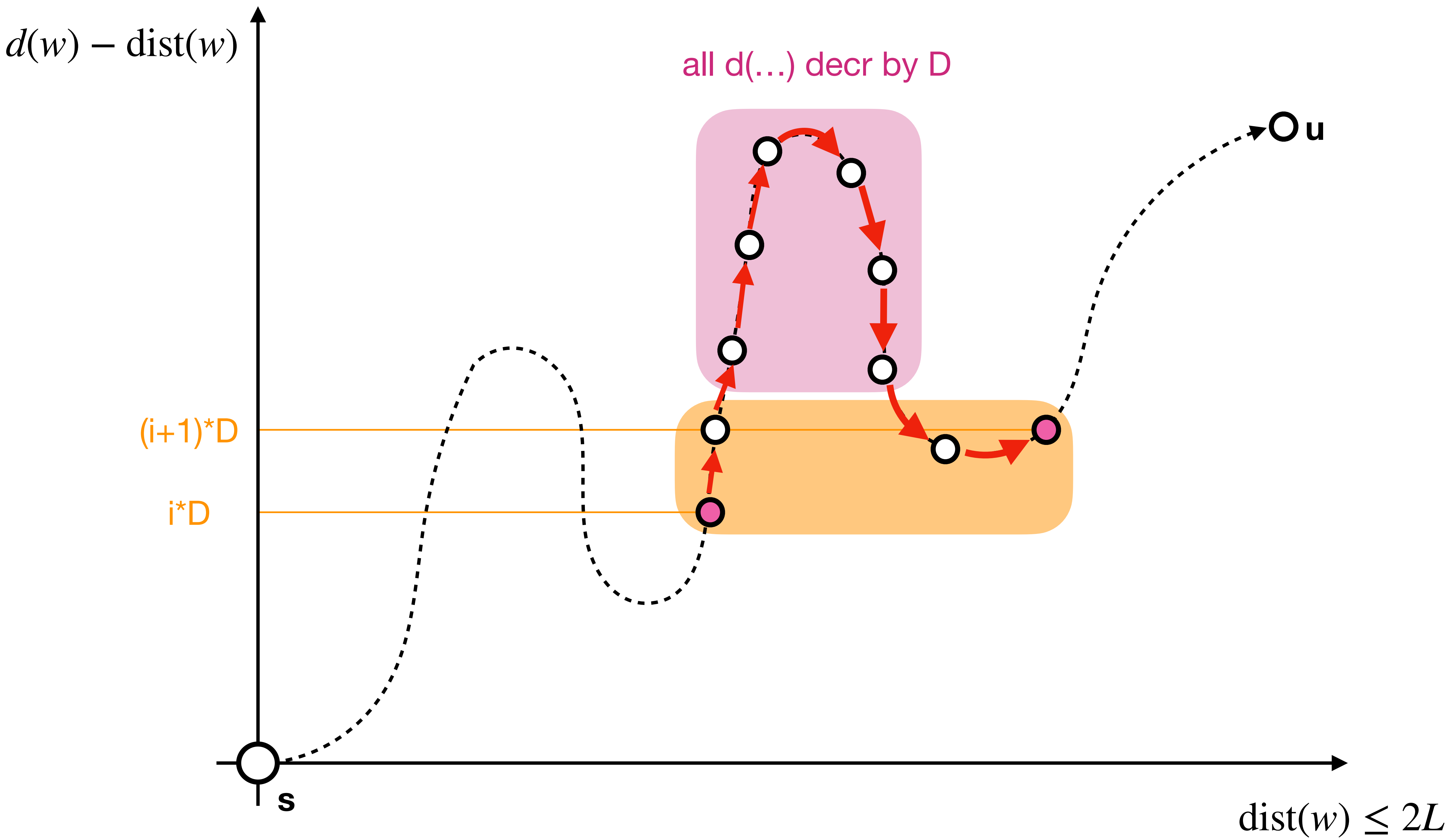
# Proof of correctness

- **Propagation** could succeed at these places



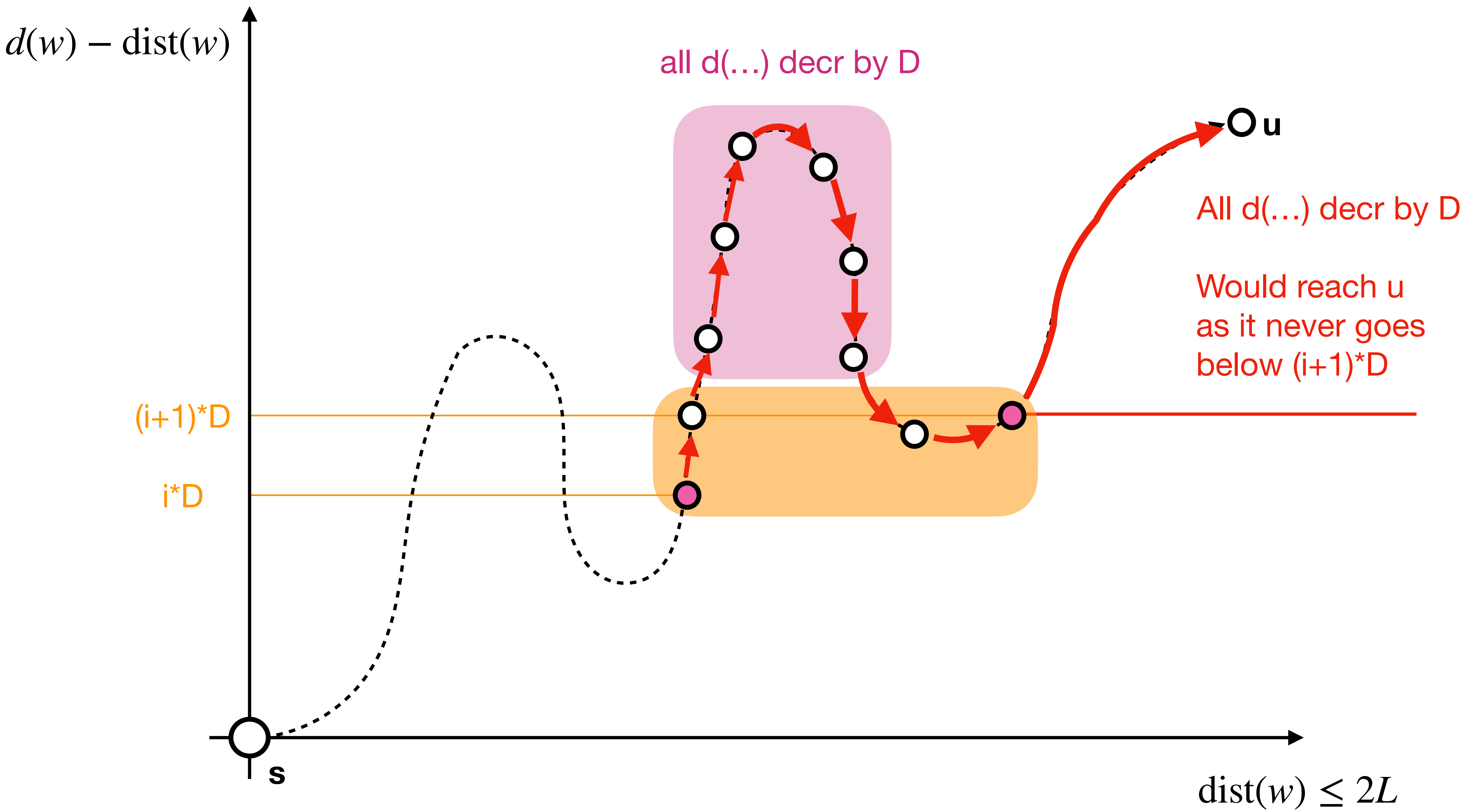
# Proof of correctness

- **Propagation** could succeed at these places



# Proof of correctness

- **Propagation** could succeed at these places



# Thank you!