Near-linear Time Algorithm for Approximate Minimum Degree Spanning Trees

Ran Duan Haoqing He **Tianyi Zhang**

Tsinghua University

Min-deg spanning trees

- Given an undirected graph $G = (V, E)$
  Find a spanning tree T minimizing $\max_{u \in V} \deg_T(u)$
- Generalize Hamiltonian Path, thus NP-hard
- Look for approximations

History

Reference	Approximation	Time
[Fürer and Raghavachari, 1992]	$O(\Delta^* + \log n)$	Poly(n)
[Fürer and Raghavachari, 1994]	$\Delta^* + 1$	$O(mn)$
New	$(1+\epsilon)\Delta^* + O(\log n/\epsilon^2)$	$O(m \log^7 n/\epsilon^6)$

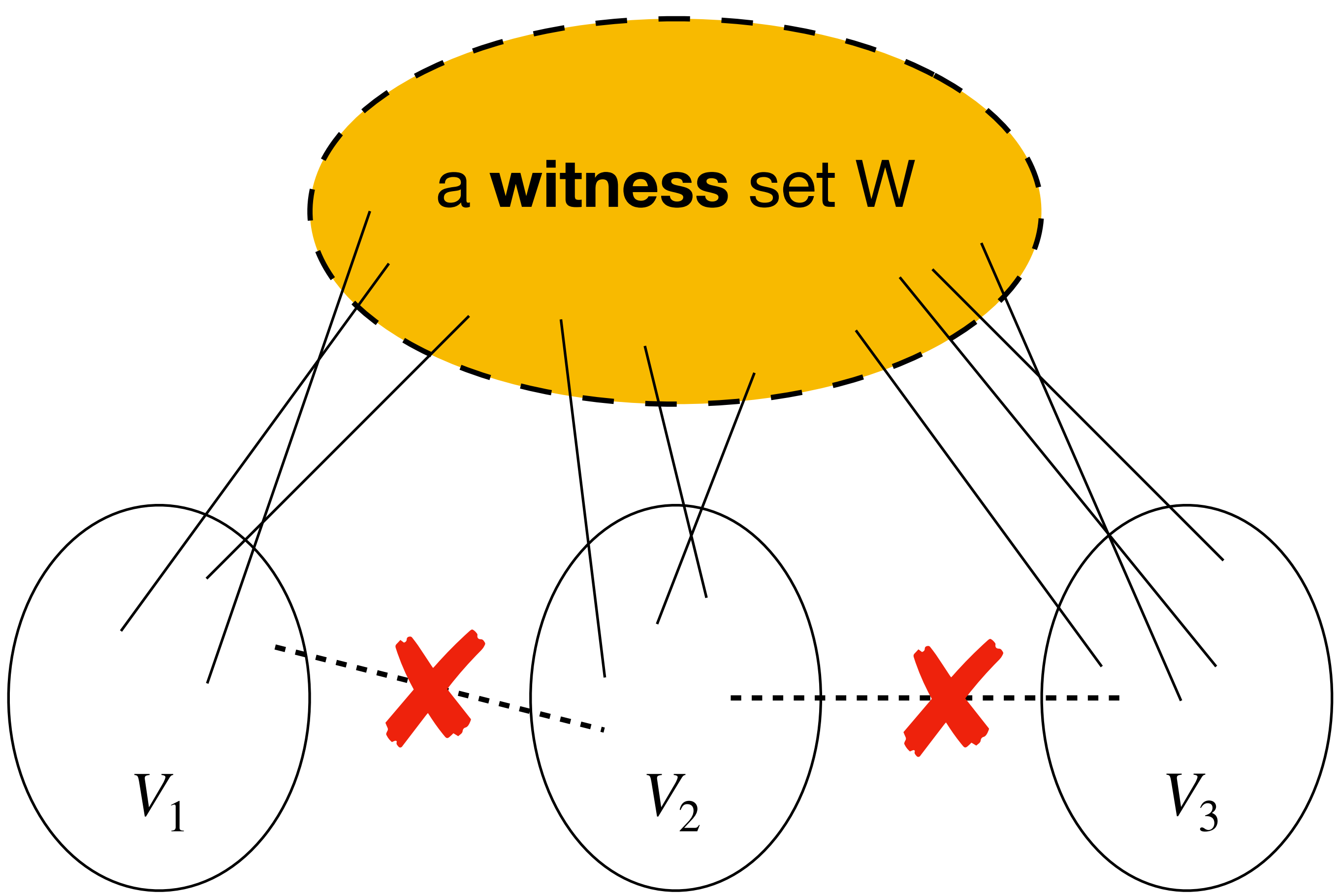
Δ^* denotes the minimum spanning tree degree
***m* and *n* denote #edges and #vertices**

$O(\Delta^* + \log n)$ in Poly(n) time

[Fürer and Raghavachari, 1992]

A witness lemma

<u>Lemma:</u> (witness)
If V is partitioned into $W, V_1, V_2, \cdots, V_l$ such that all inter-component edges touch the **witness set** W, then a lower bound holds $\Delta^* \geq (l-1)/|W|$



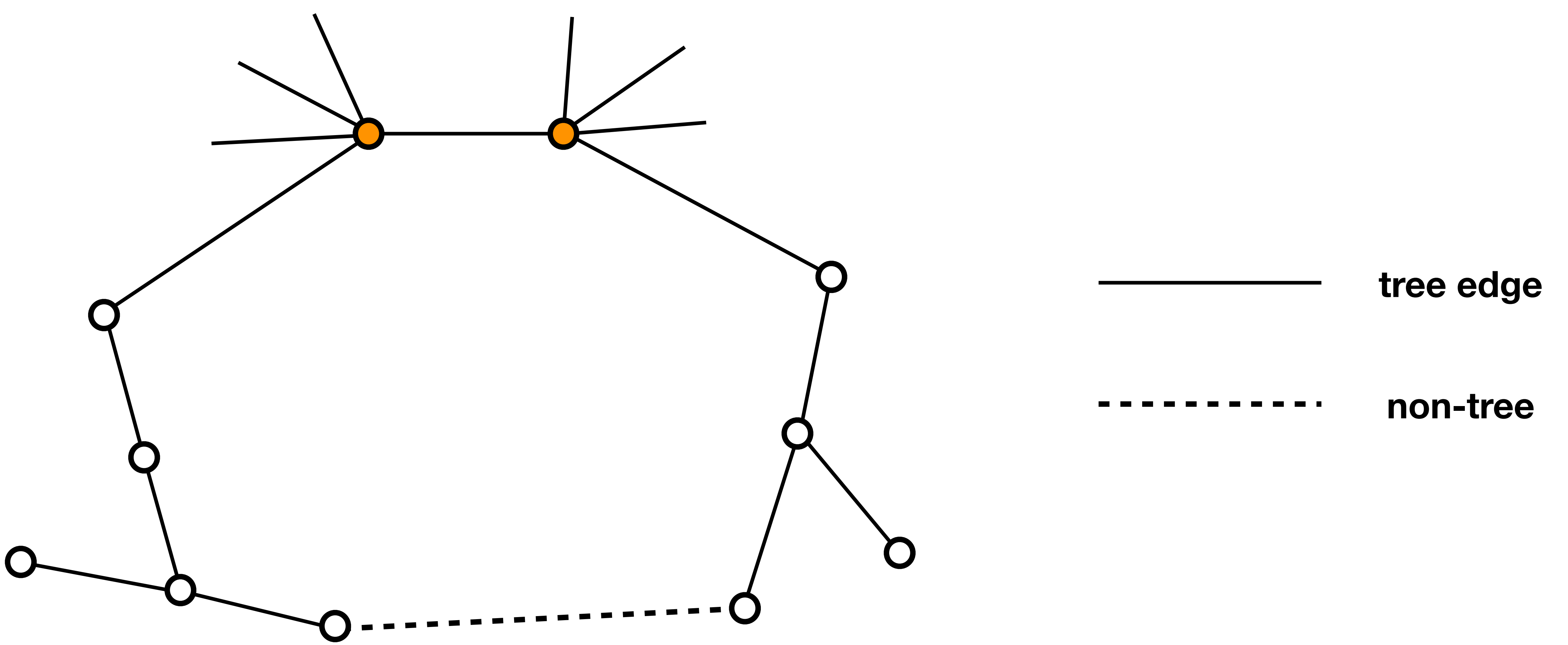
Any spanning tree has at least $l-1$ inter-component edges

All these edges are incident on the witness set W

So, at least one vertex in W has tree degree $\geq (l-1)/|W|$

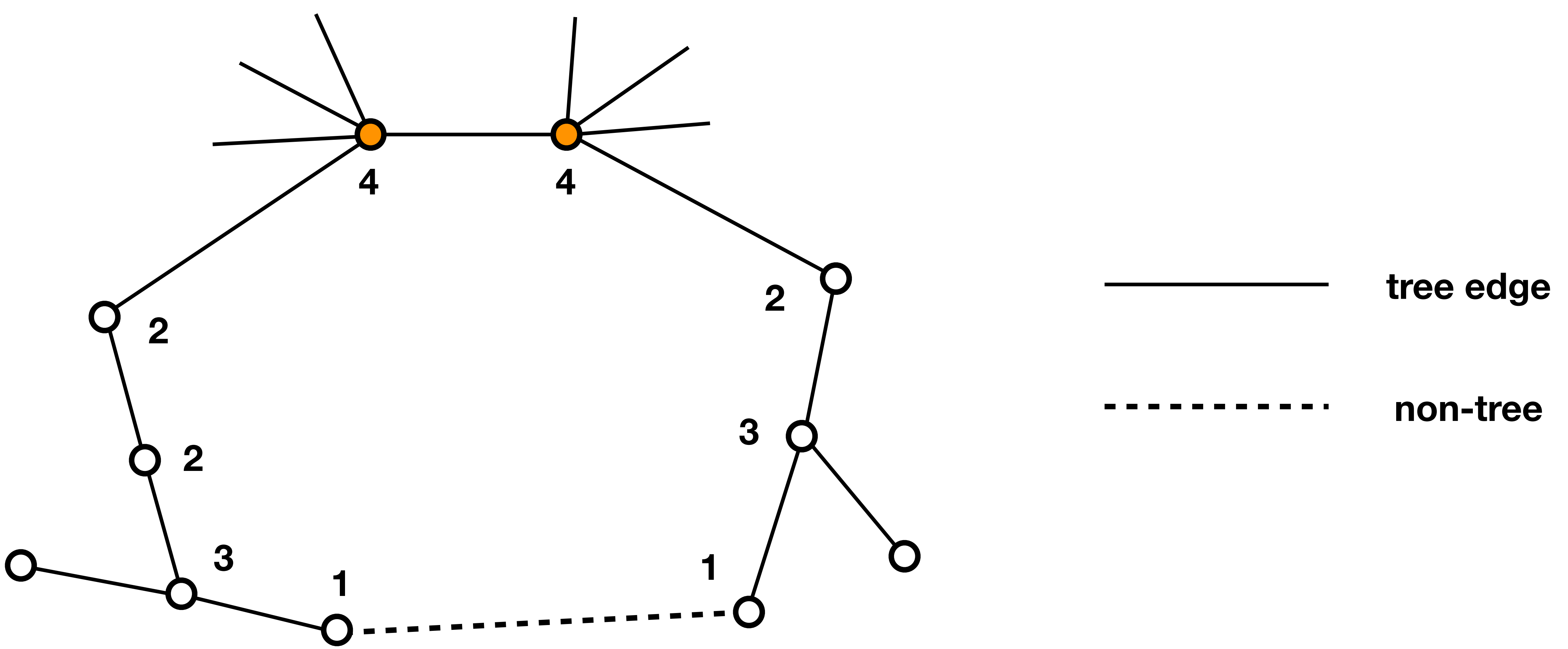
Local search

- Given a tree T, try to reduce its vertex degrees
- Find non-tree edge (u, v), $\deg_T(u), \deg_T(v) \leq d - 2$
  tree path contains a vertex *w* with $\deg_T(w) \geq d$
- Switch non-tree and tree edges



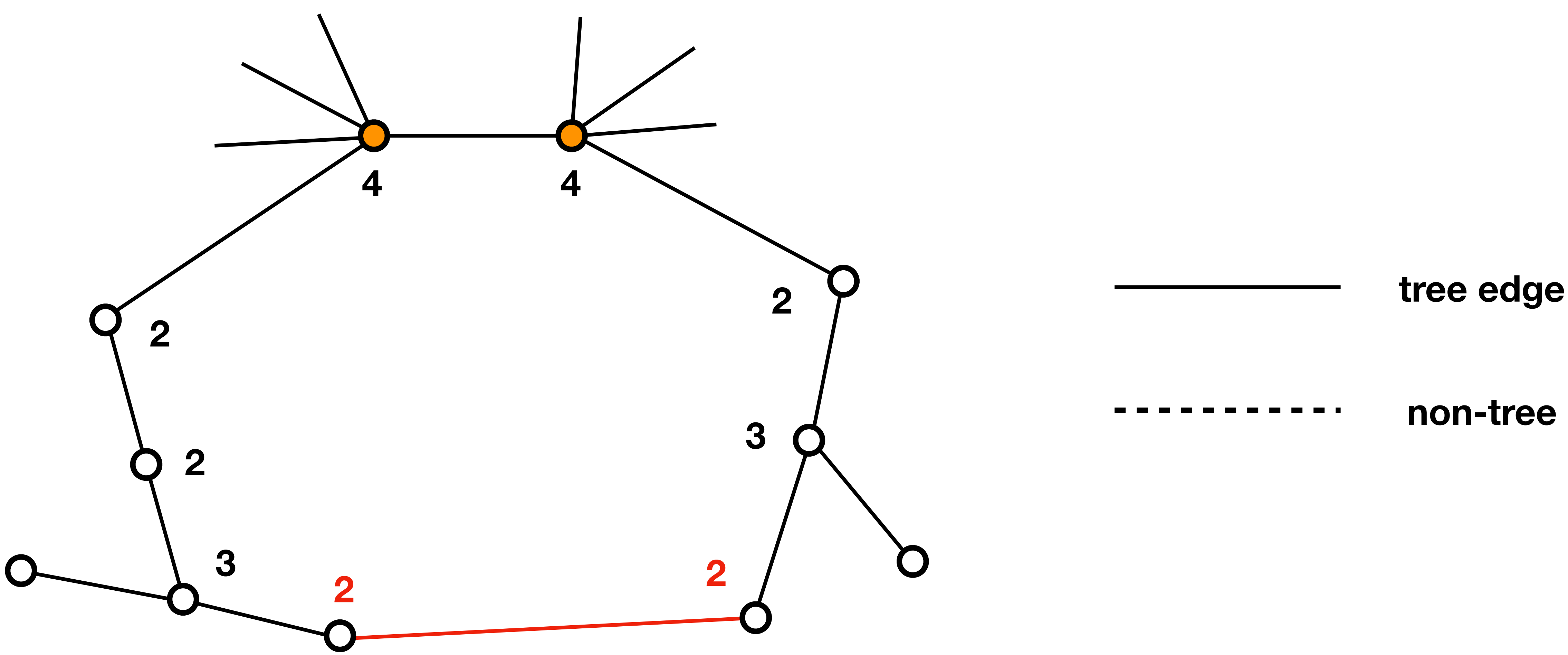
Local search

- Given a tree T, try to reduce its vertex degrees
- Find non-tree edge (u, v), $\deg_T(u), \deg_T(v) \leq d - 2$
  tree path contains a vertex *w* with $\deg_T(w) \geq d$
- Switch non-tree and tree edges



Local search

- Given a tree T, try to reduce its vertex degrees
- Find non-tree edge (u, v), $\deg_T(u), \deg_T(v) \leq d - 2$
  tree path contains a vertex *w* with $\deg_T(w) \geq d$
- Switch non-tree and tree edges

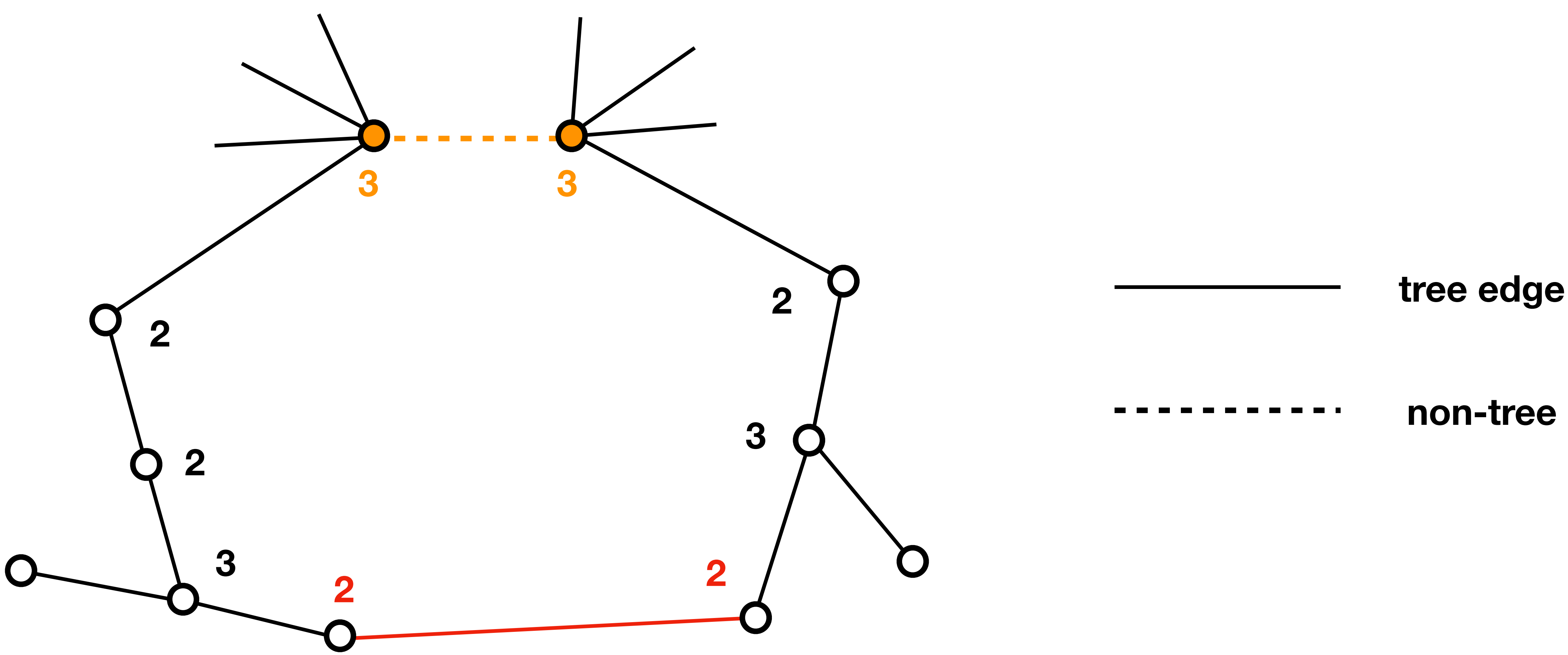


Local search

- Given a tree T, try to reduce its vertex degrees

- Find non-tree edge (u, v), $\deg_T(u), \deg_T(v) \leq d - 2$
  tree path contains a vertex *w* with $\deg_T(w) \geq d$

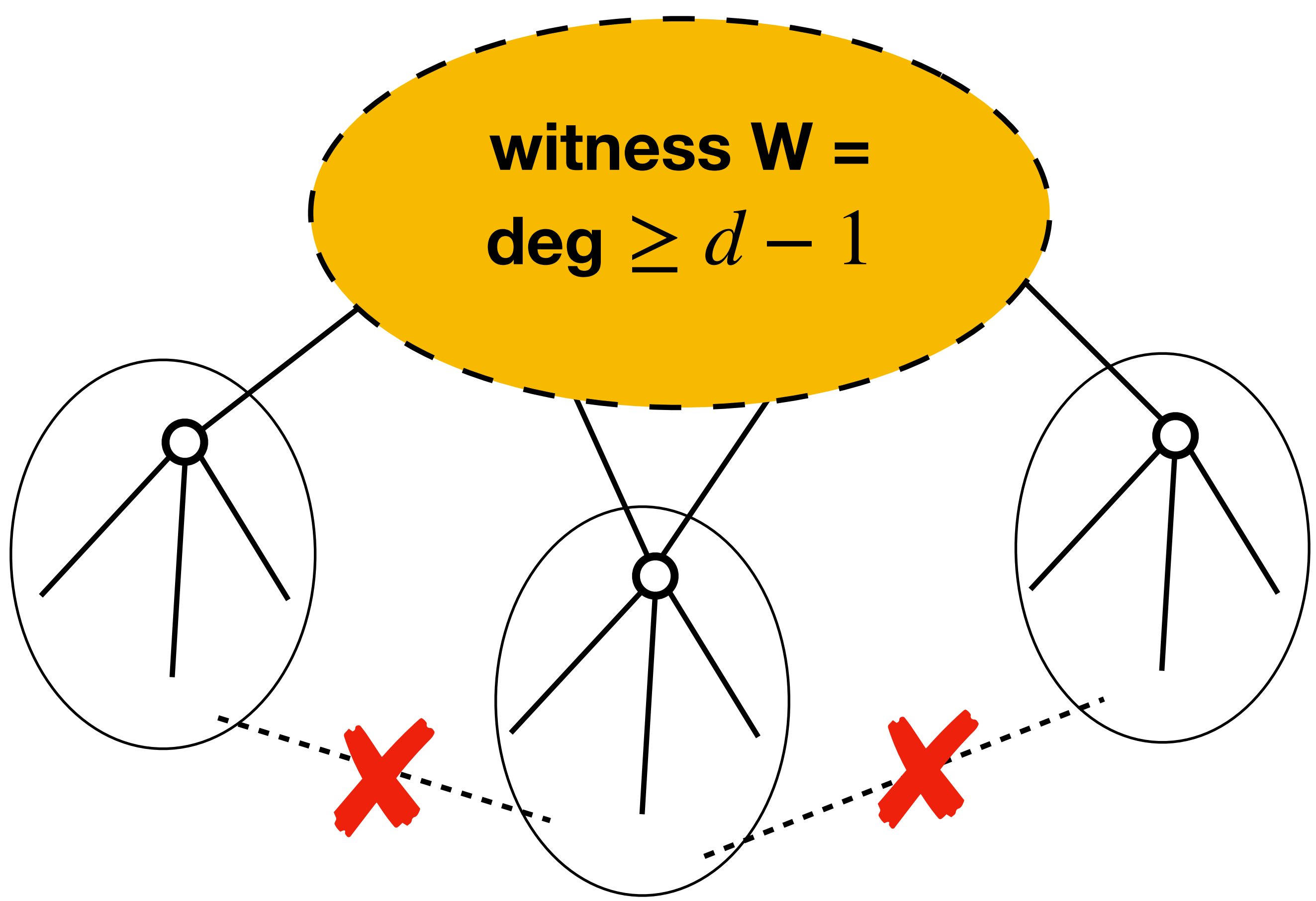
- Switch non-tree and tree edges



Local search

- Repeatedly find non-tree/tree edge switches

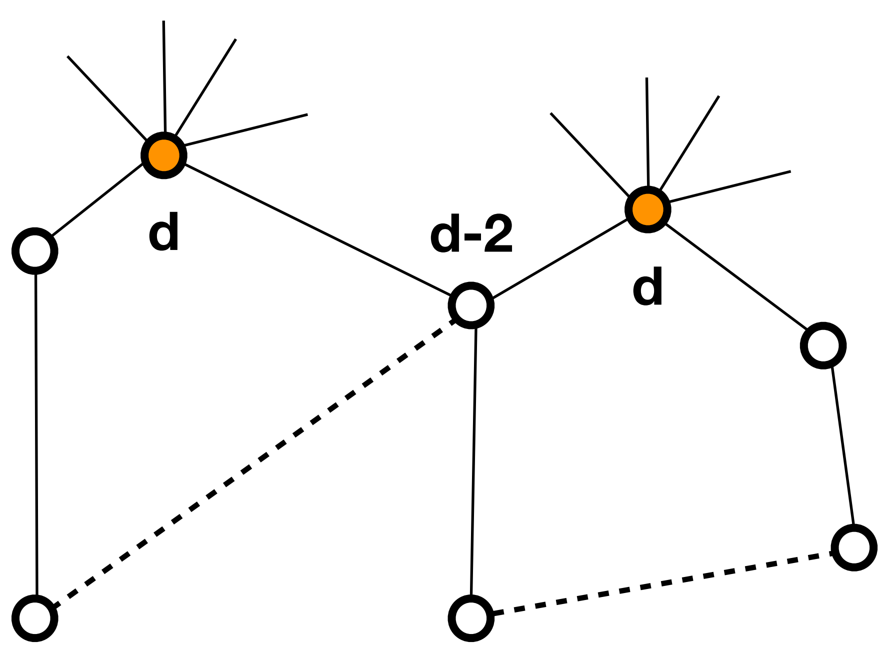
- **Stopping condition:**
  If no such switches, then **prove** $\max\{\deg_T(u)\} = O(\Delta^* + \log n)$



Running time

- Repeatedly go over all non-tree edges (u, v), find *w* on the tree path s.t. $\deg_T(u), \deg_T(v) \le d - 2$, $\deg_T(w) \ge d$

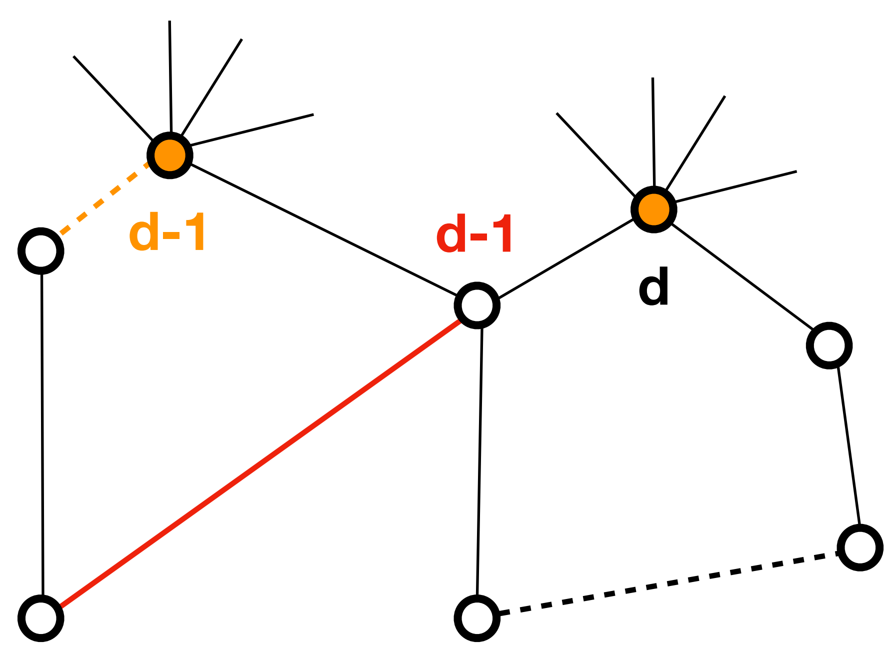
- $\deg_T(u)$ could switch between $d - 1$ and $\le d - 2$, so each edge may need to be **checked multiple times**



Running time

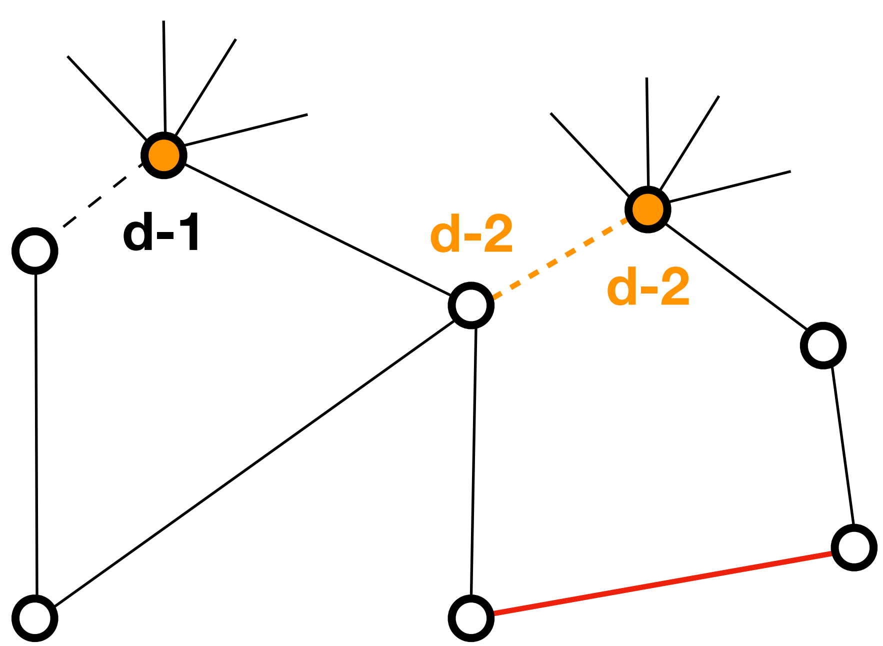
- Repeatedly go over all non-tree edges (u, v),
  find *w* on the tree path
  s.t. $\deg_T(u), \deg_T(v) \leq d - 2$, $\deg_T(w) \geq d$

- $\deg_T(u)$ could switch between $d - 1$ and $\leq d - 2$,
  so each edge may need to be **checked multiple times**



Running time

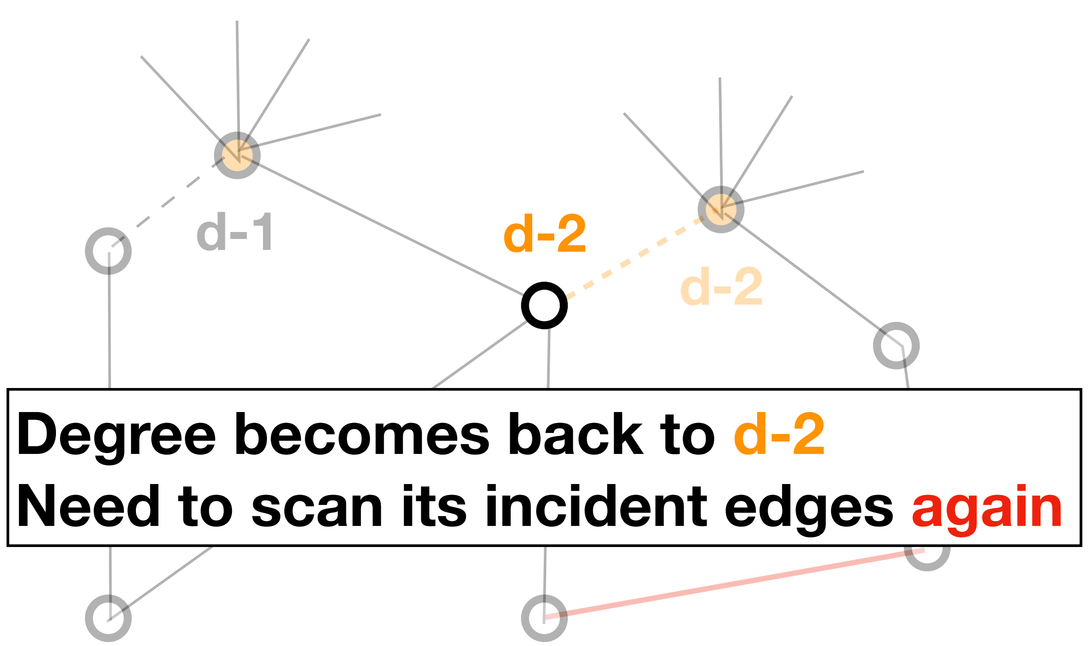
- Repeatedly go over all non-tree edges (u, v), find *w* on the tree path s.t. $\deg_T(u), \deg_T(v) \leq d - 2$, $\deg_T(w) \geq d$
- $\deg_T(u)$ could switch between $d - 1$ and $\leq d - 2$, so each edge may need to be **checked multiple times**



Running time

- Repeatedly go over all non-tree edges (u, v), find *w* on the tree path s.t. $\deg_T(u), \deg_T(v) \leq d - 2$, $\deg_T(w) \geq d$

- $\deg_T(u)$ could switch between $d - 1$ and $\leq d - 2$, so each edge may need to be **checked multiple times**



$O(\Delta^* \log n)$ in $\tilde{O}(m)$ time

A lazy approach

- Previous issue:
  $\deg_T(u)$ could switch between $d-1$ and $\leq d-2$,
  so each edge may need to be **checked multiple times**
- Idea:
  Scan each adjacency list **only once**,
  even if $\deg_T(u) = d-1$ drops again

A lazy approach

A linear time algorithm

1. Define $S = \{u \mid \deg_T(u) \geq d\}$

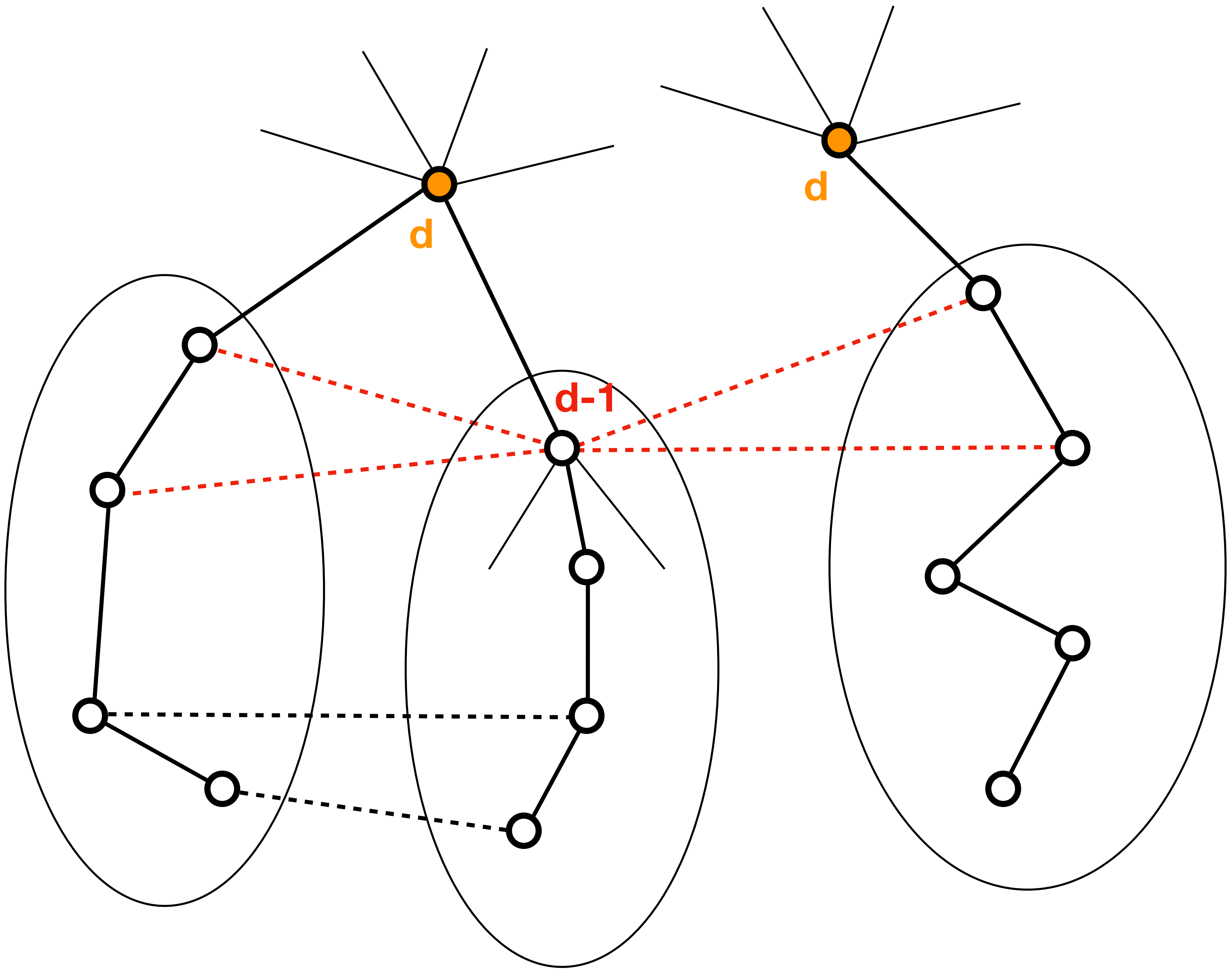
2. Go over all edges (u, v)
   If u, v are in different component in $T \backslash S$,
   and $\deg_T(u), \deg_T(v)$ **have never been $d - 1$**,
   then switch (u, v) with an edge on S

3. Each edge is **visited only once**, thus linear time

A lazy approach

Scan all dotted edges to find non-tree/tree edge switches

Red dotted edges are forbidden at the beginning

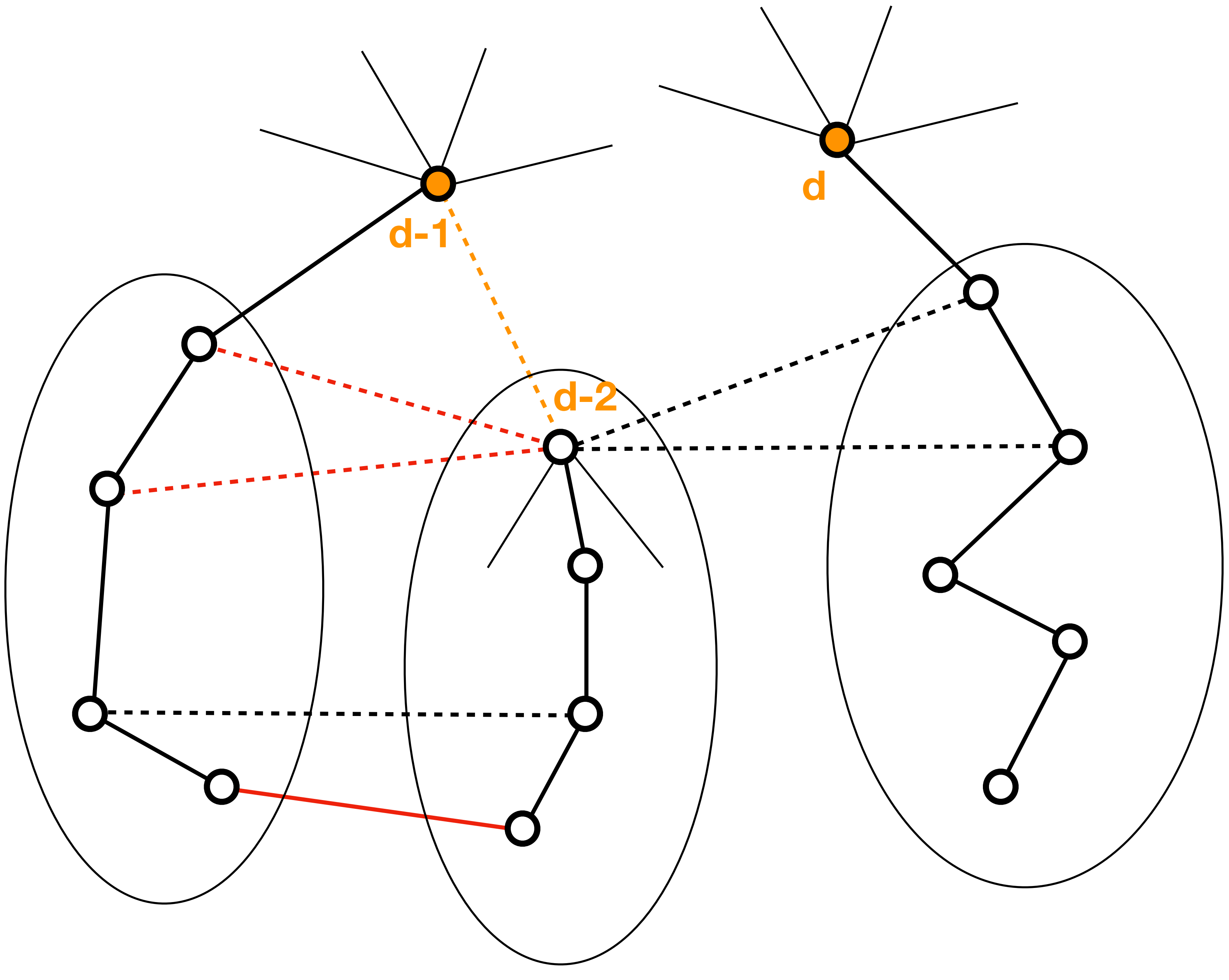


A lazy approach

Scan all dotted edges to find non-tree/tree edge switches

Red dotted edges are forbidden at the beginning

Find a switch which reduces degrees



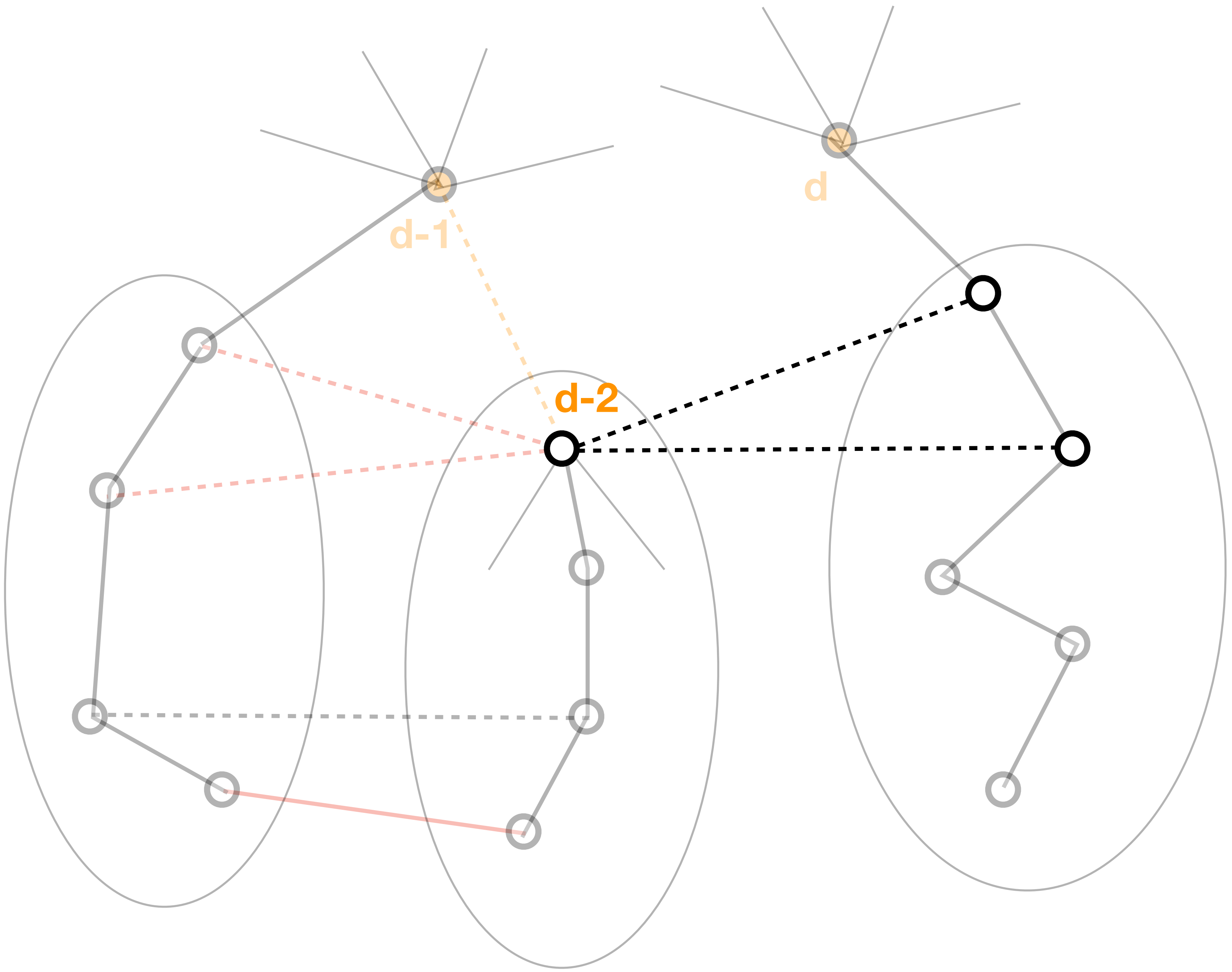
A lazy approach

Scan all dotted edges
to find non-tree/tree
edge switches

Red dotted edges are
forbidden at the beginning

Find a switch which
reduces degrees

After a non-tree/tree
edge switch, a degree
drops below d-1,
introducing more
possible edge switches



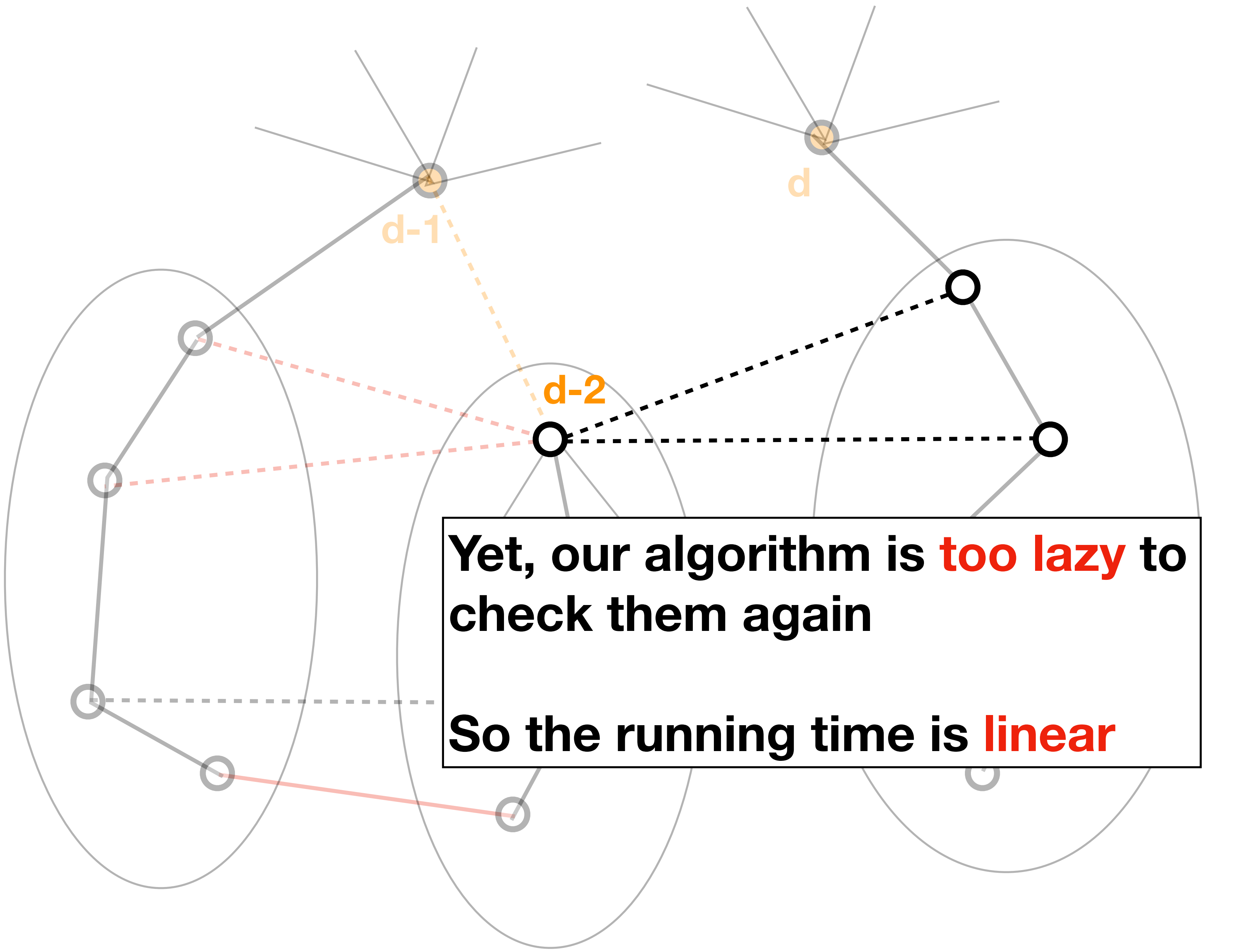
A lazy approach

Scan all dotted edges to find non-tree/tree edge switches

Red dotted edges are forbidden at the beginning

Find a switch which reduces degrees

After a non-tree/tree edge switch, a degree drops below d-1, introducing more possible edge switches



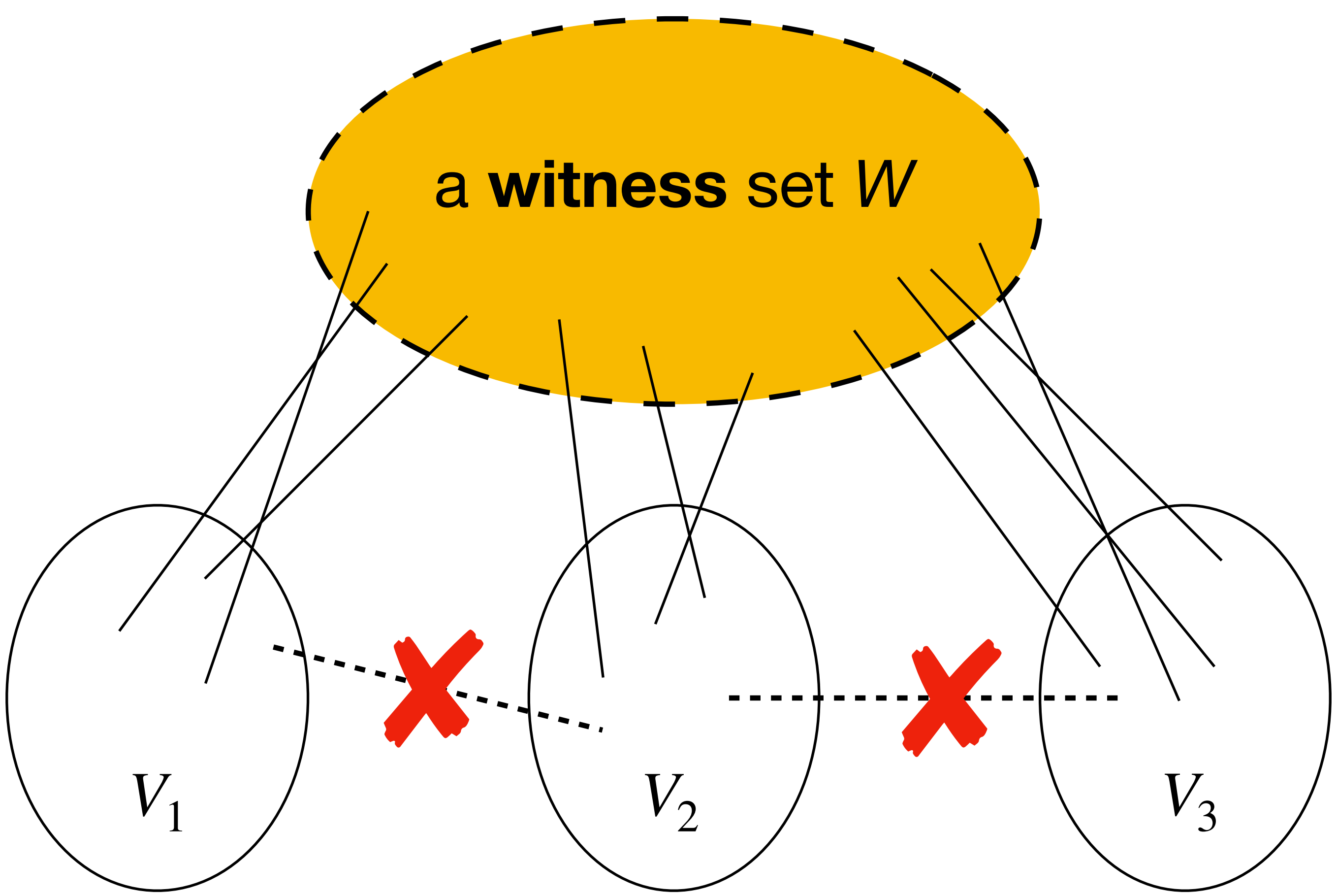
Yet, our algorithm is too lazy to check them again

So the running time is linear

A lazy approach

Problems with applying **the witness lemma**

- Take witness set W = { u | $\deg_T(u)$ was once $\geq d - 1$ }



By the witness lemma,
$\Delta^* \geq (l-1)/|W|$

W could contain too many vertices with low tree degrees, which leads to $l \ll d|W|,\ \Delta^* \ll d$

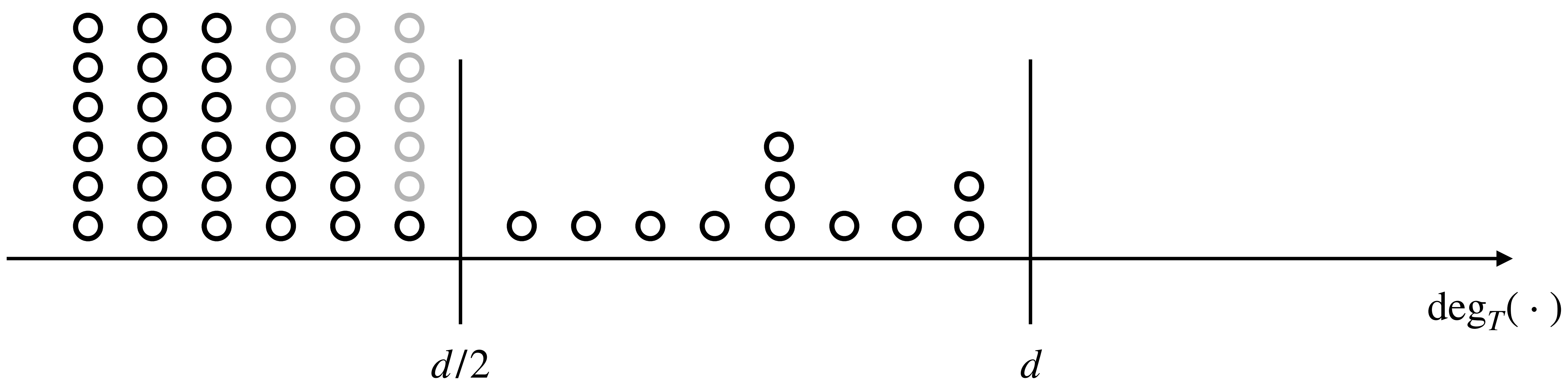
Not a good stopping condition

A key observation

- Ideally, for vertices with a low tree degree ($< d/2$) at the beginning, most of them may never reach $d-1$
- Hopefully, *W* mostly consists of high-deg vertices

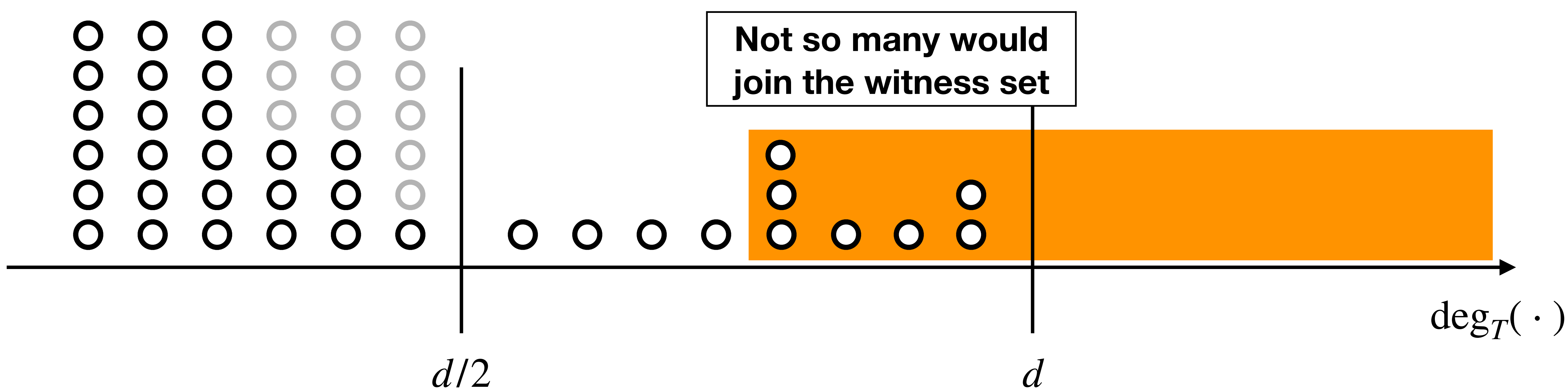
A key observation

- Ideally, for vertices with a low tree degree ($< d/2$) at the beginning, most of them may never reach $d-1$
- Hopefully, *W* mostly consists of high-deg vertices



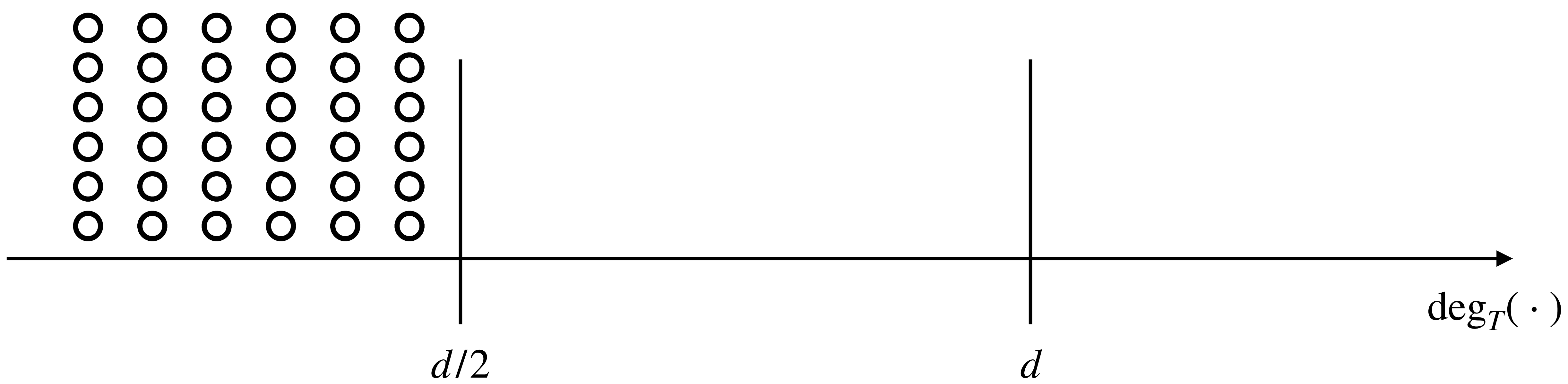
A key observation

- Ideally, for vertices with a low tree degree ($< d/2$) at the beginning, most of them may never reach $d-1$
- Hopefully, W mostly consists of high-deg vertices



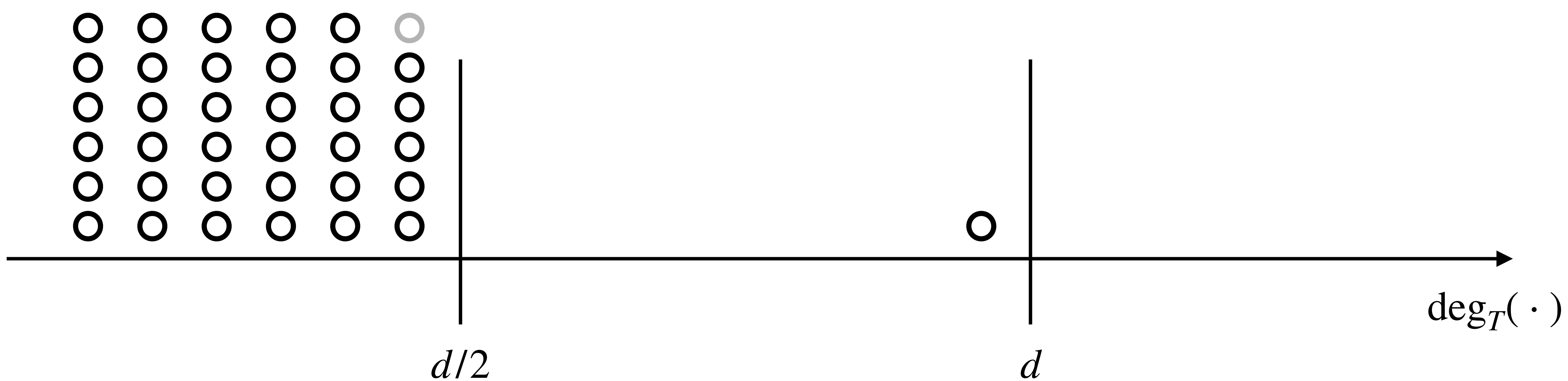
A key observation

- Ideally, for vertices with a low tree degree ($< d/2$) at the beginning, most of them may never reach $d-1$
- What if most of them have reached $d-1$



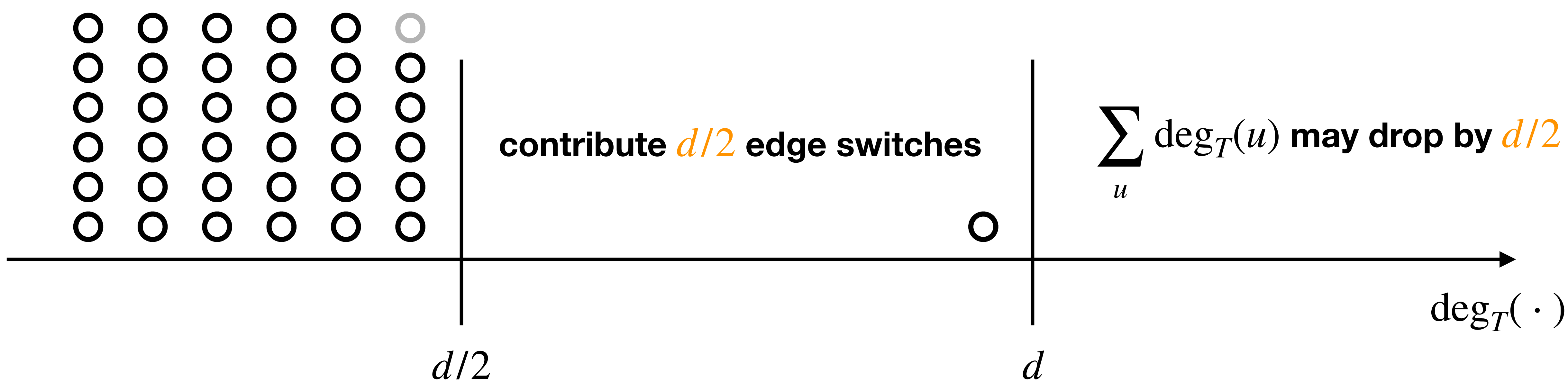
A key observation

- Ideally, for vertices with a low tree degree ($< d/2$) at the beginning, most of them may never reach $d-1$
- What if most of them have reached $d-1$



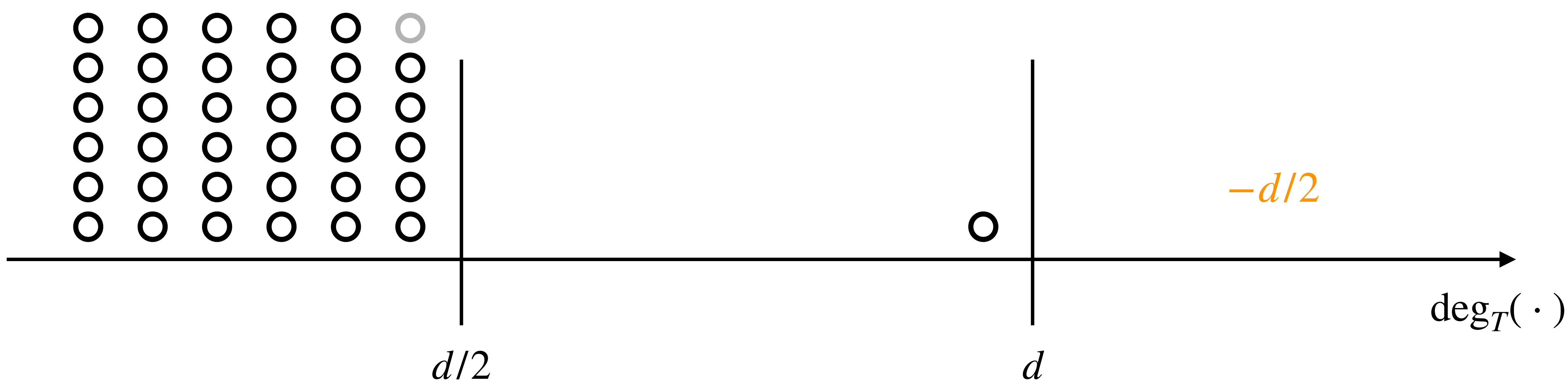
A key observation

- Ideally, for vertices with a low tree degree ($< d/2$) at the beginning, most of them may never reach $d-1$
- What if most of them have reached $d-1$



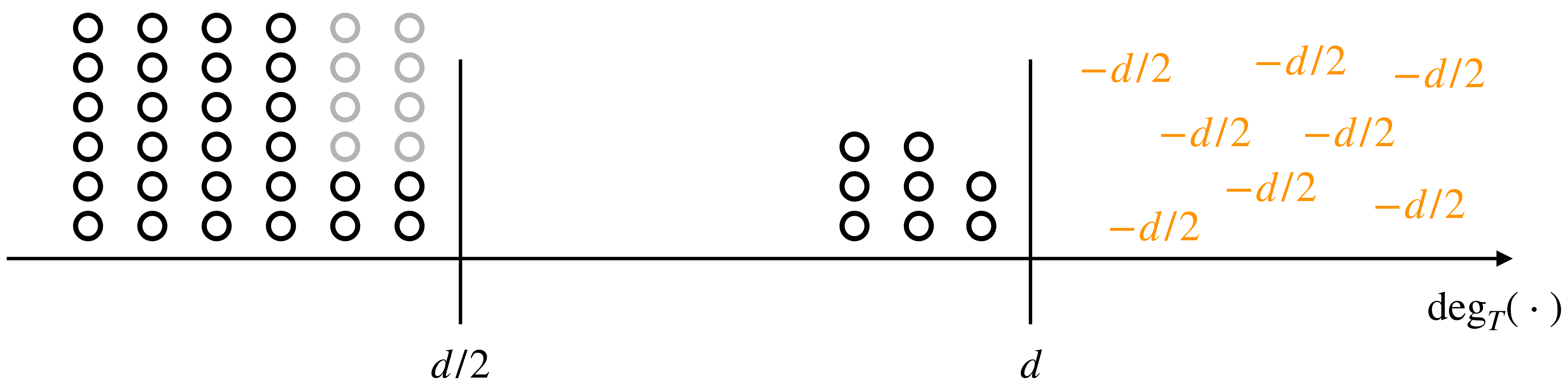
A key observation

- Ideally, for vertices with a low tree degree ($< d/2$) at the beginning, most of them may never reach $d-1$
- What if most of them have reached $d-1$



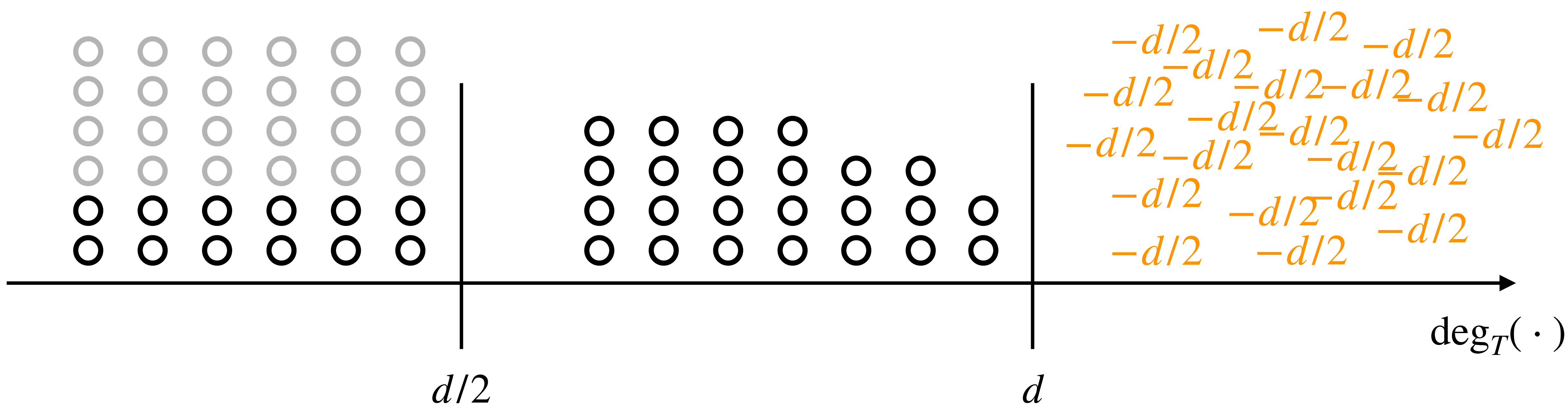
A key observation

- Ideally, for vertices with a low tree degree ($< d/2$) at the beginning, most of them may never reach $d-1$
- What if most of them have reached $d-1$



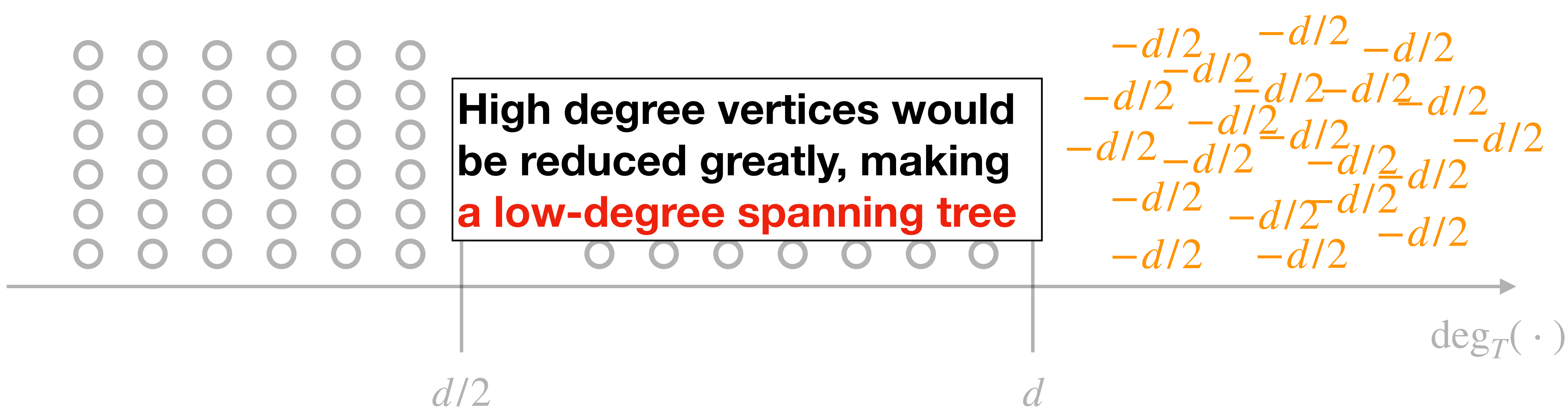
A key observation

- Ideally, for vertices with a low tree degree ($< d/2$) at the beginning, most of them may never reach $d-1$
- What if most of them have reached $d-1$



A key observation

- Ideally, for vertices with a low tree degree ($< d/2$) at the beginning, most of them may never reach $d-1$
- What if most of them have reached $d-1$



A remaining issue

- Ideally, for vertices with a low tree degree ($< d/2$) at the beginning, most of them may never reach $d-1$

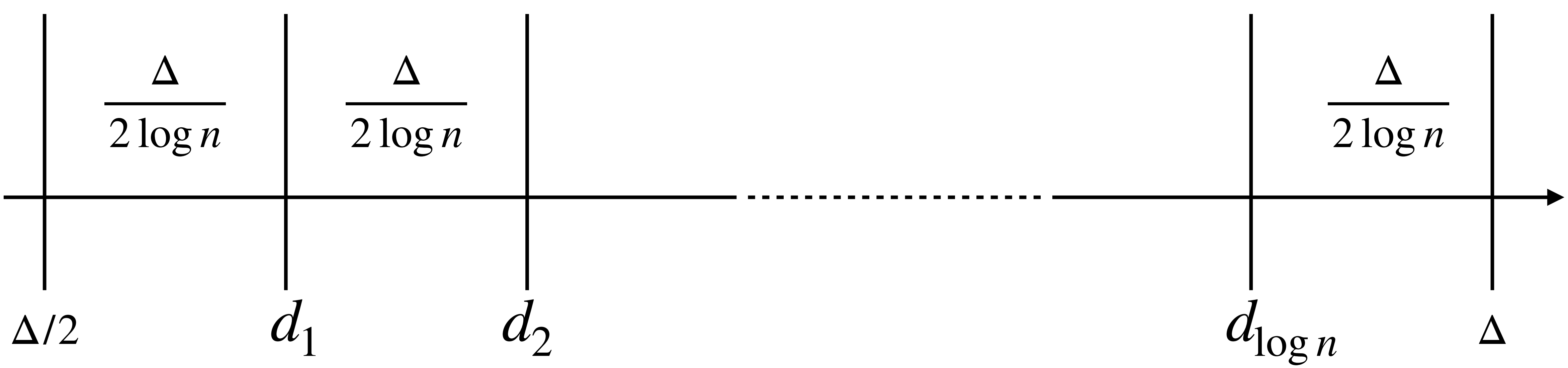
- What happens to vertices with medium degree $\in [d/2, d-1)$

- Try $O(\log n)$ **different choices** of d, make sure that #{ u | $deg_T(u) \geq d/2$ } $\leq$ **2 *** #{ u | $deg_T(u) \geq d-1$ }

- Overall, the witness set won't be blown up too much

A remaining issue

- Try $O(\log n)$ **different choices** of d; make sure that #{ u | $deg_T(u) \geq d/2$ } ≤ **2** * #{ u | $deg_T(u) \geq d - 1$ }
- Overall, the witness set won't be blown up too much
- Apply our lazy local-search on d_i for $i = 1,2,\cdots$ Can prove $\Delta = \max \deg_T(\cdot)$ will be reduced multiplicatively

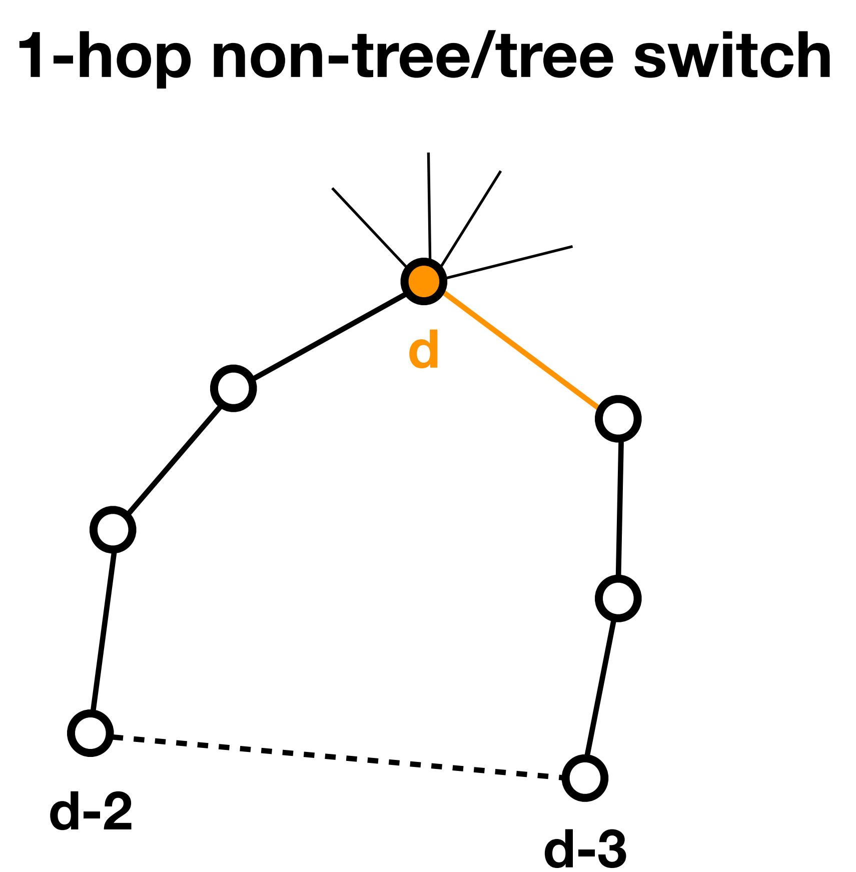


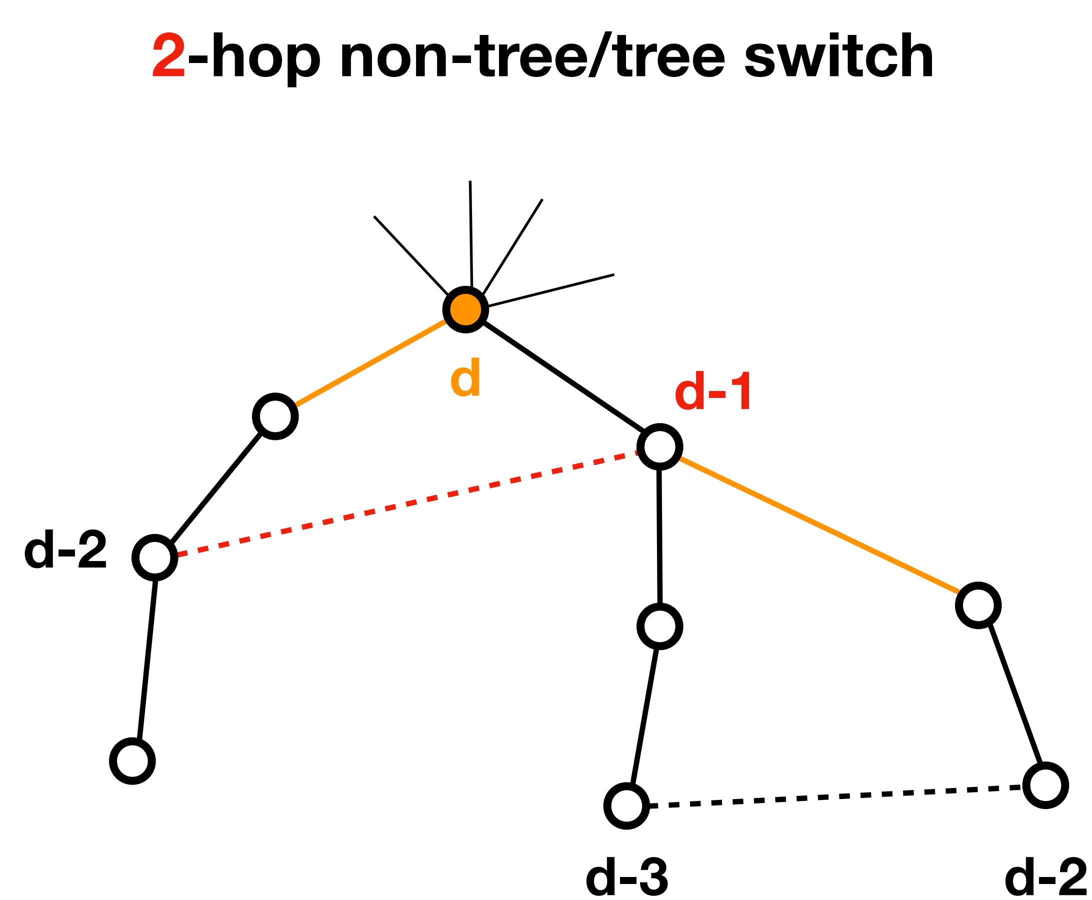
$(1+\epsilon)\Delta^*$ in $\tilde{O}(m)$ time

high-level overview

Multi-hop switches

- Generalize the concept of non-tree/tree edge switches
- Reduce $O(\Delta^* \log n)$ to $(1+\epsilon)\Delta^* + O(\log n/\epsilon)$
- Run $\tilde{O}(m)$ time using the lazy approach as well





Multi-hop switches

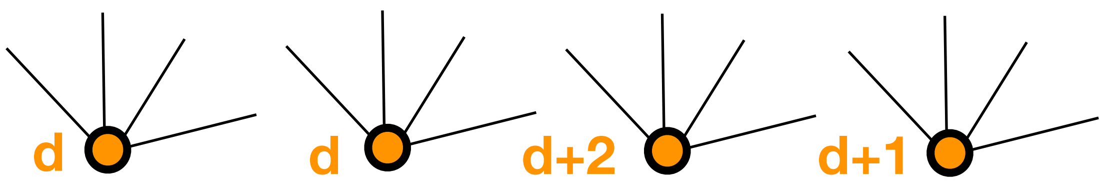
- Similar to **the blocking-flow** algorithm for max-flow
- Search for **longer & longer** hop non-tree/tree switches

- Assume we do not have switches with <k hops
- To find k-hop switches, partition into *k* layers
- Use a depth-first search to find *k*-hop switches

Multi-hop switches

- **To find k-hop switches, partition into *k* layers**

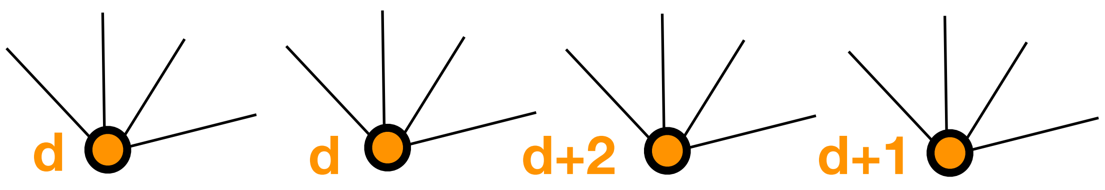
- Use a depth-first search to find *k*-hop switches



Multi-hop switches

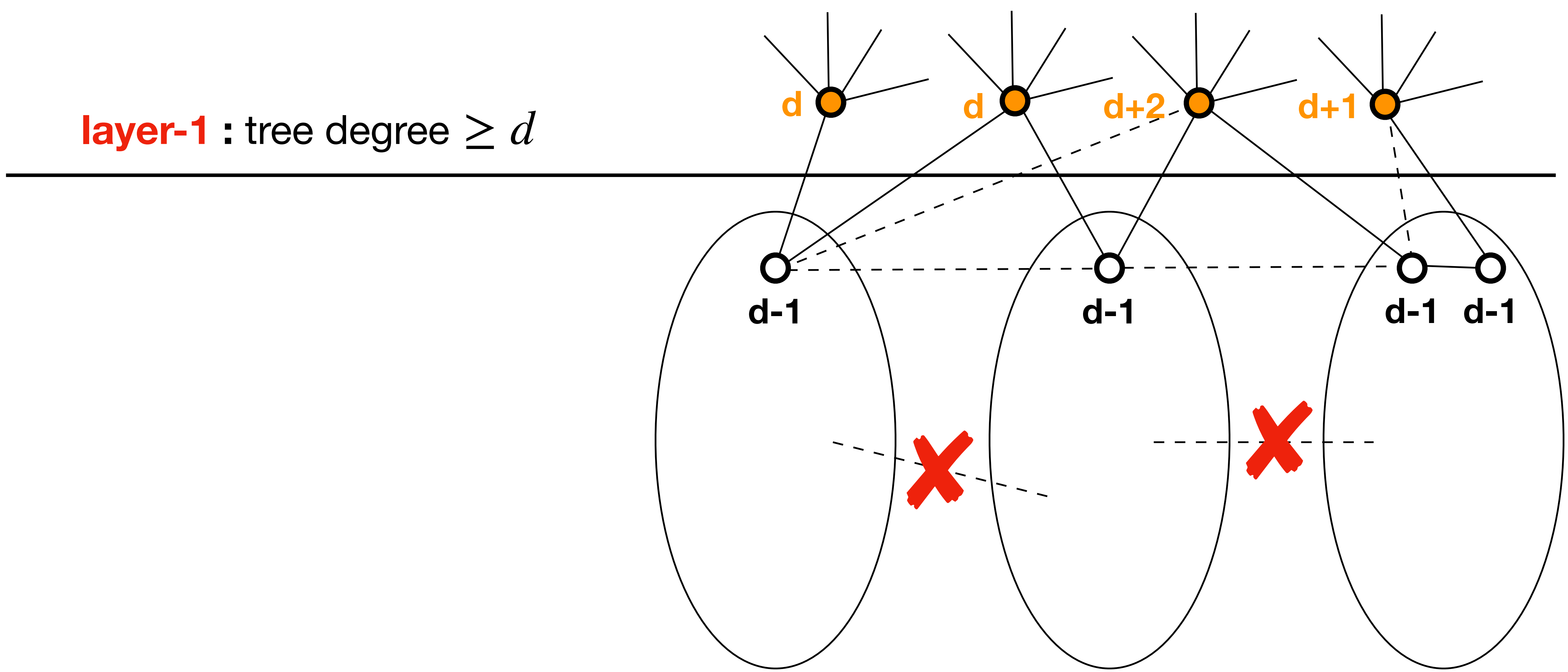
- **To find k-hop switches, partition into *k* layers**
- Use a depth-first search to find *k*-hop switches

layer-1 : tree degree $\geq d$



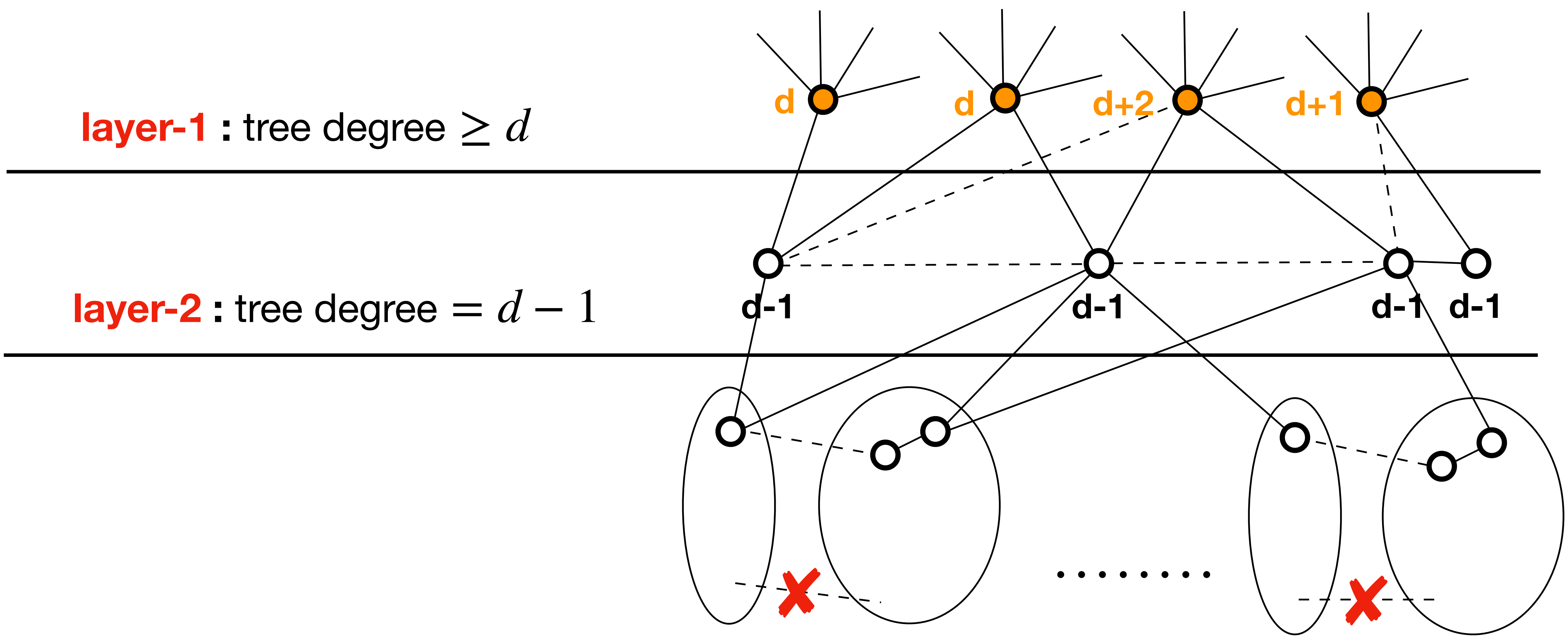
Multi-hop switches

- **To find k-hop switches, partition into *k* layers**
- Use a depth-first search to find *k*-hop switches



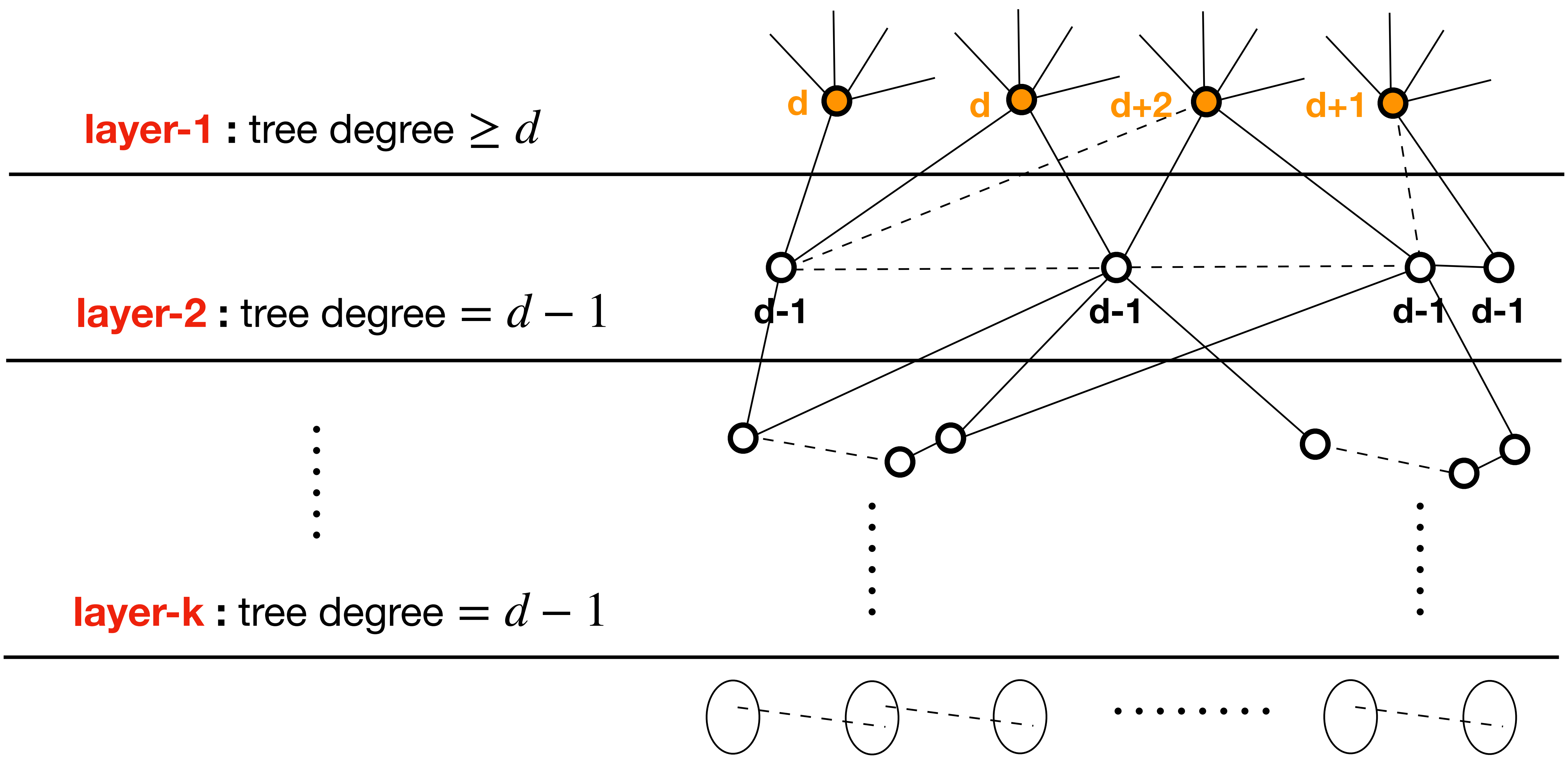
Multi-hop switches

- **To find k-hop switches, partition into *k* layers**
- Use a depth-first search to find *k*-hop switches



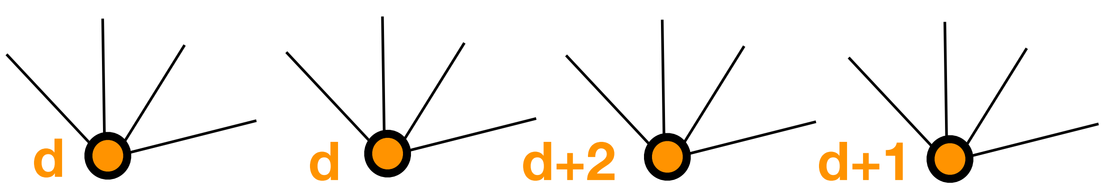
Multi-hop switches

- **To find k-hop switches, partition into *k* layers**
- Use a depth-first search to find *k*-hop switches



Multi-hop switches

- To find k-hop switches, partition into *k* layers
- **Use a depth-first search to find *k*-hop switches**

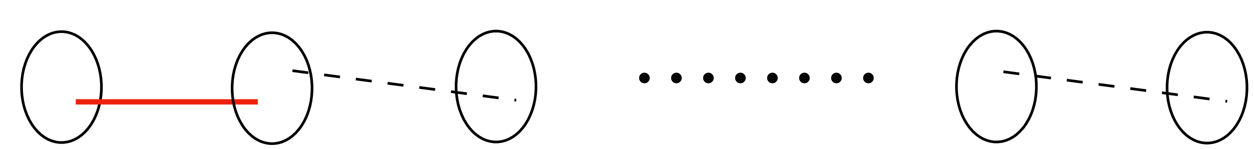


layer-1 : tree degree $\geq d$

layer-2 : tree degree $= d - 1$

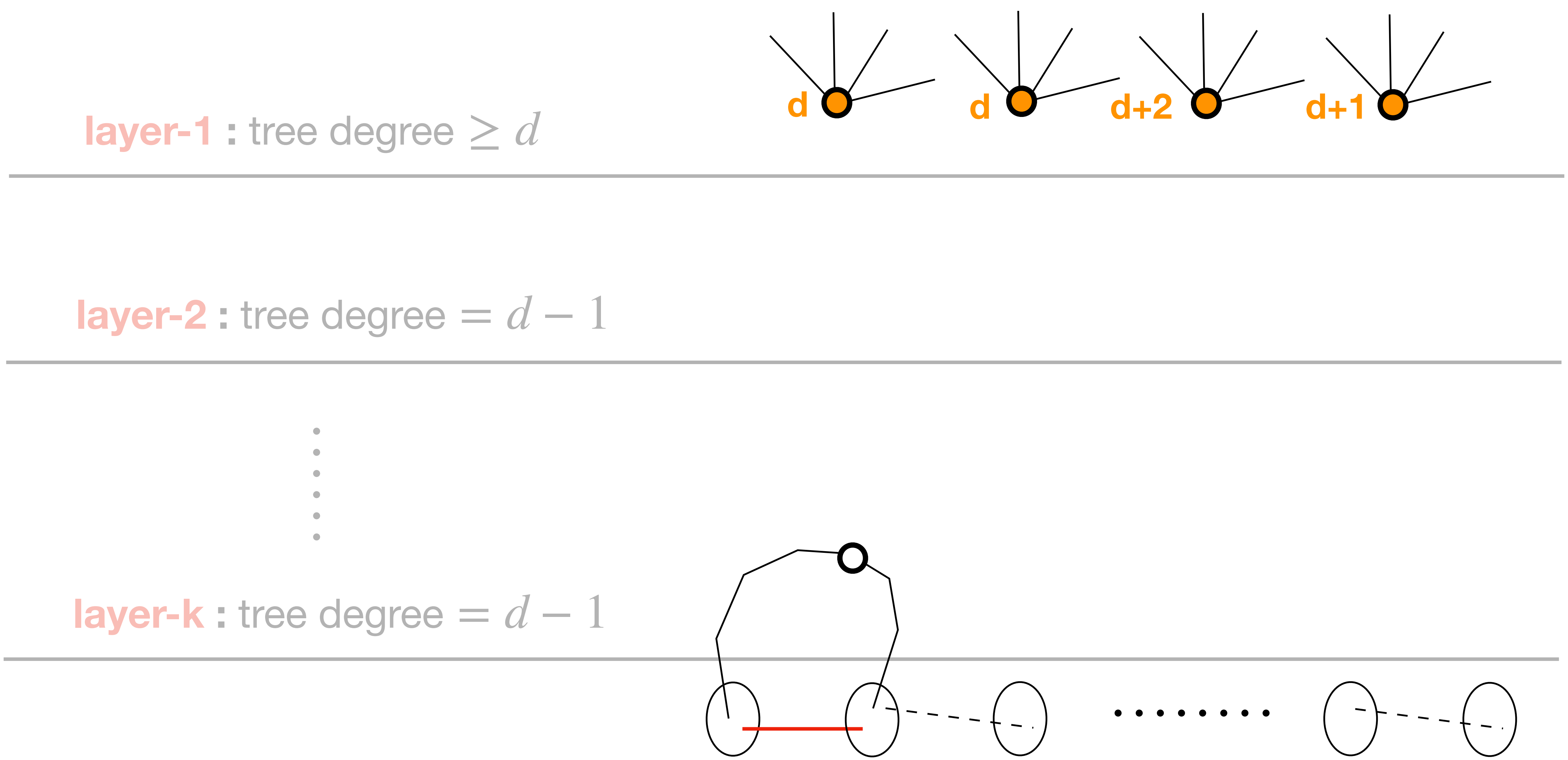
⋮

layer-k : tree degree $= d - 1$



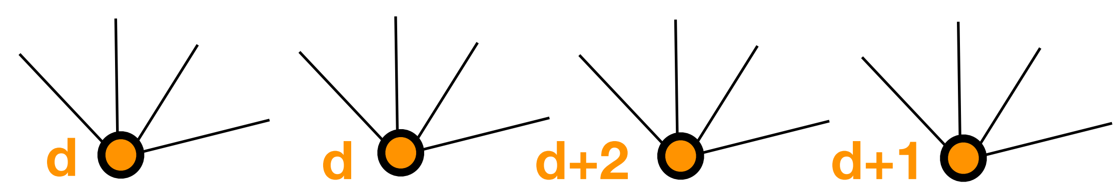
Multi-hop switches

- To find k-hop switches, partition into *k* layers
- **Use a depth-first search to find *k*-hop switches**



Multi-hop switches

- To find k-hop switches, partition into *k* layers
- **Use a depth-first search to find *k*-hop switches**

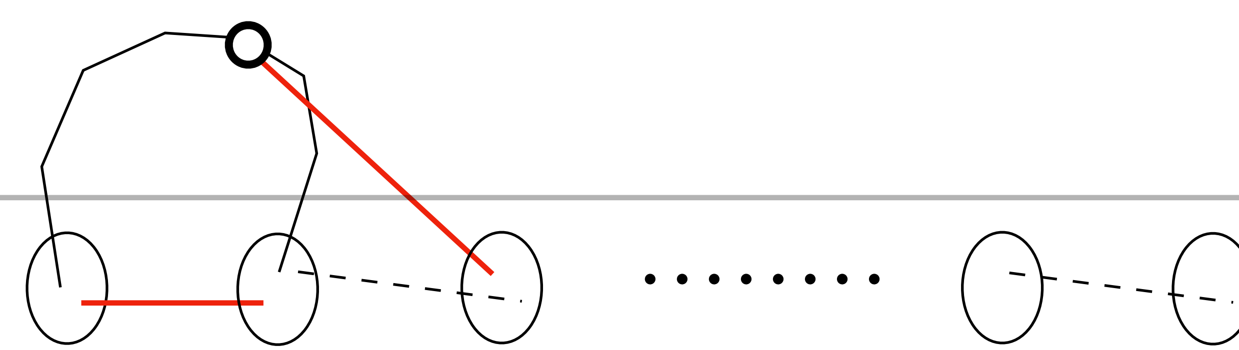


layer-1 : tree degree $\geq d$

layer-2 : tree degree $= d - 1$

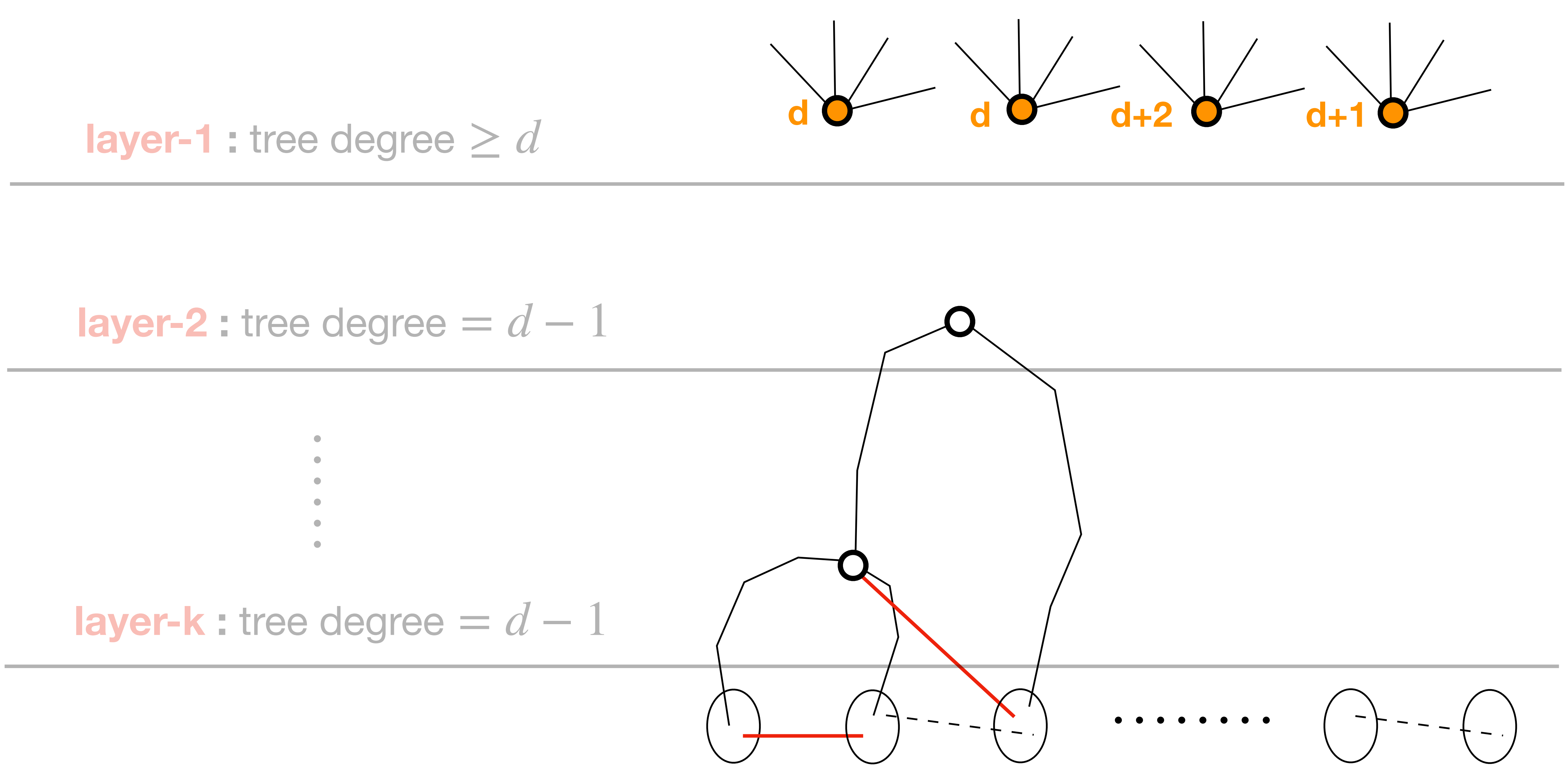
⋮

layer-k : tree degree $= d - 1$



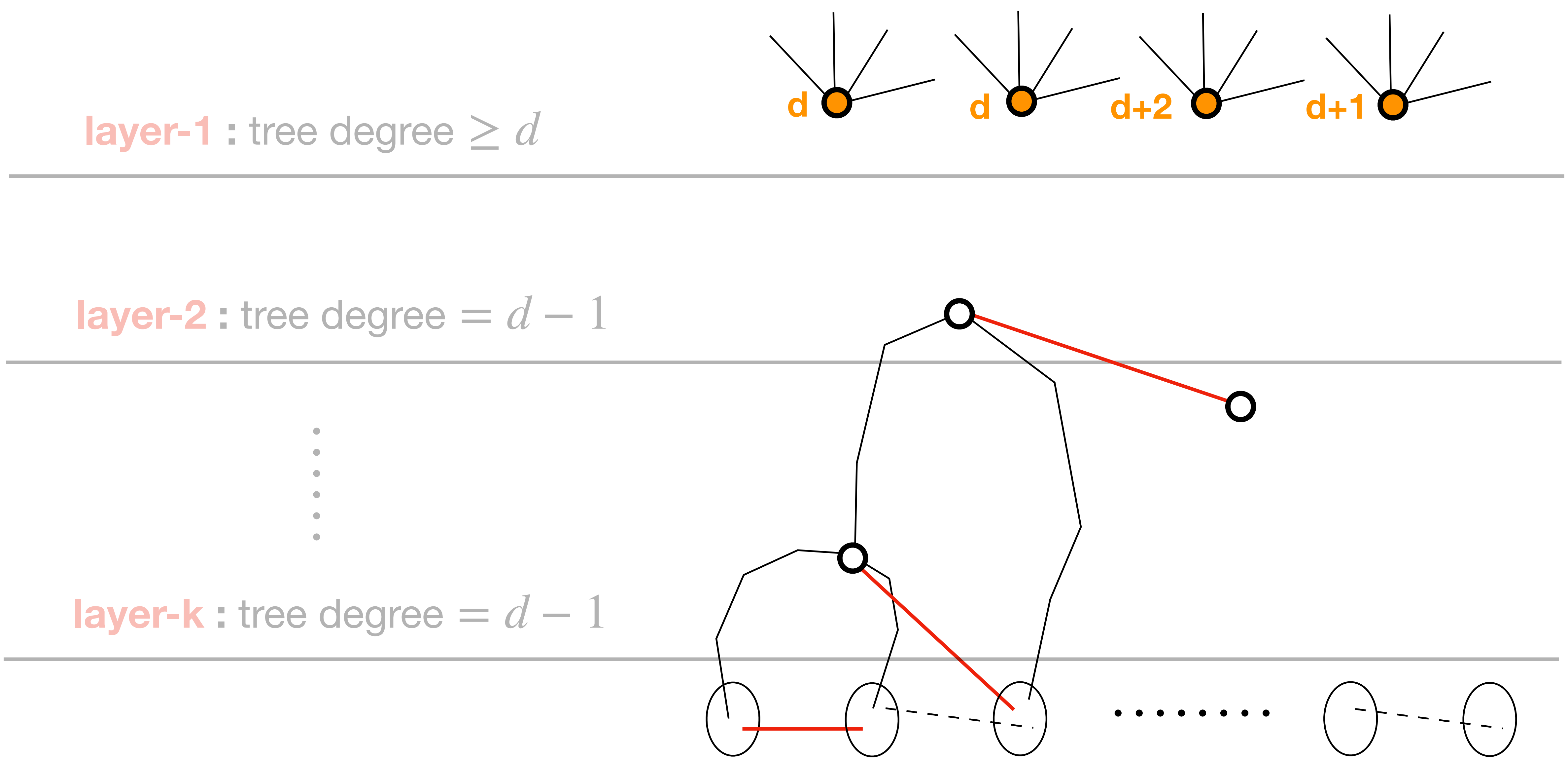
Multi-hop switches

- To find k-hop switches, partition into *k* layers
- **Use a depth-first search to find *k*-hop switches**



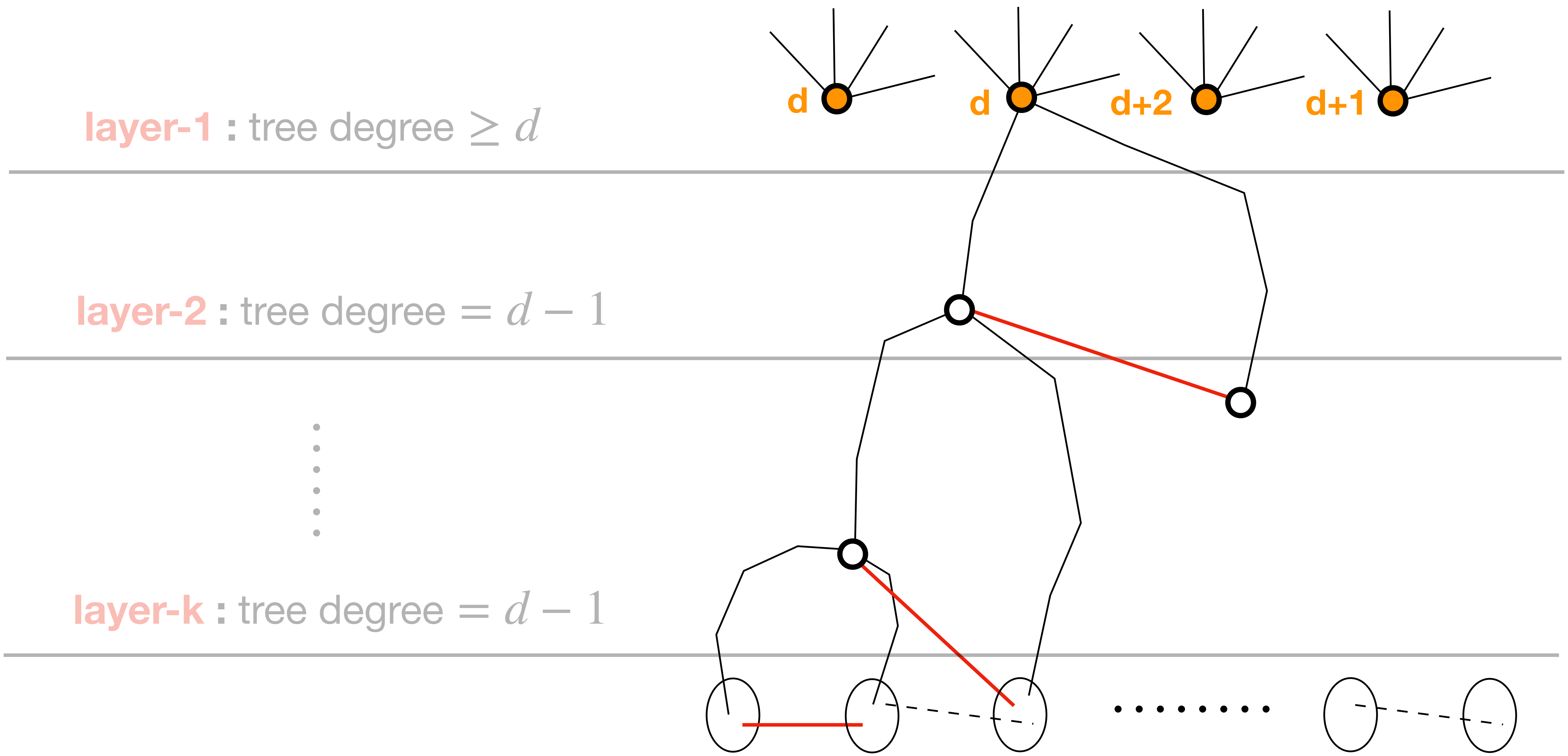
Multi-hop switches

- To find k-hop switches, partition into *k* layers
- **Use a depth-first search to find *k*-hop switches**



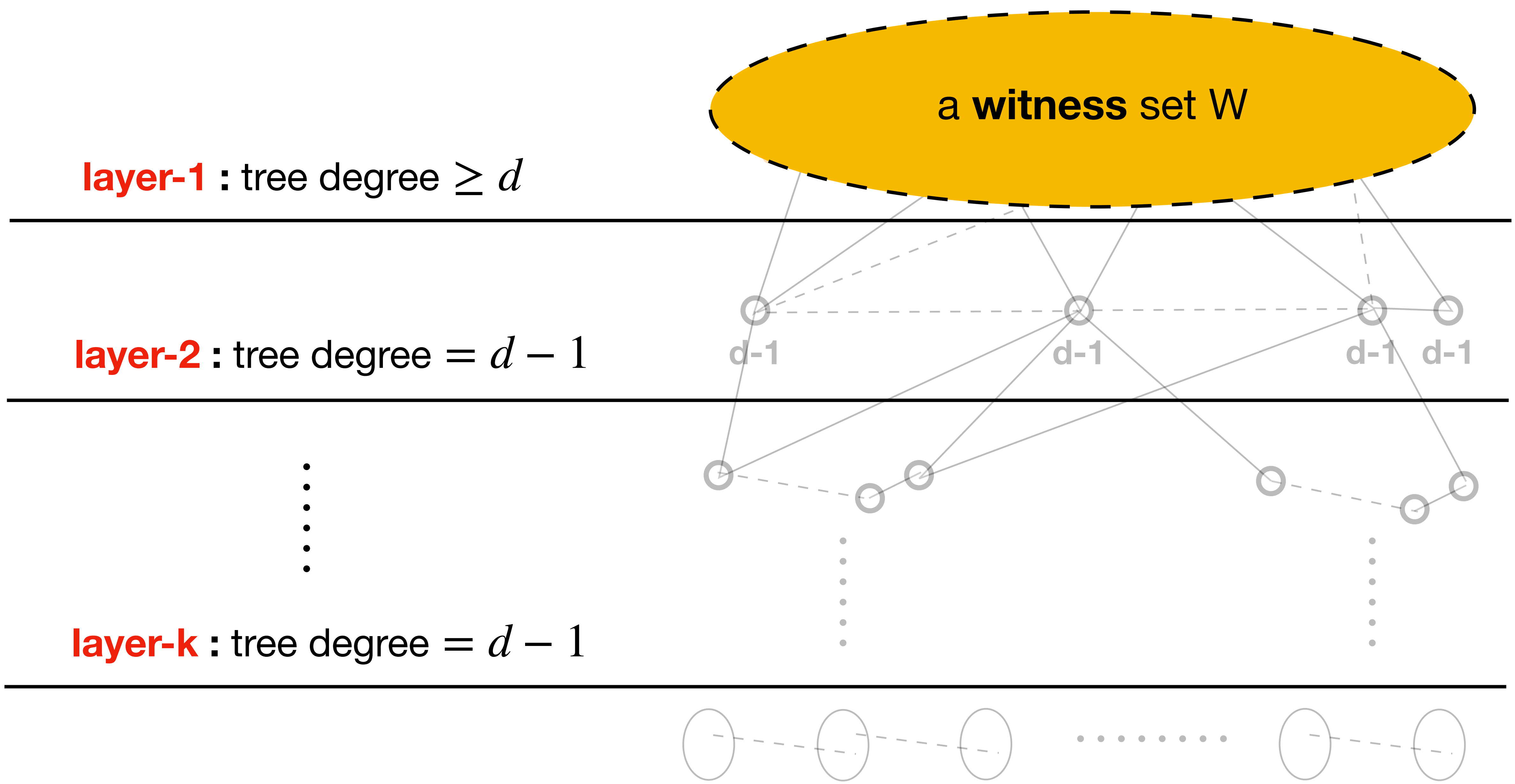
Multi-hop switches

- To find k-hop switches, partition into *k* layers
- **Use a depth-first search to find *k*-hop switches**



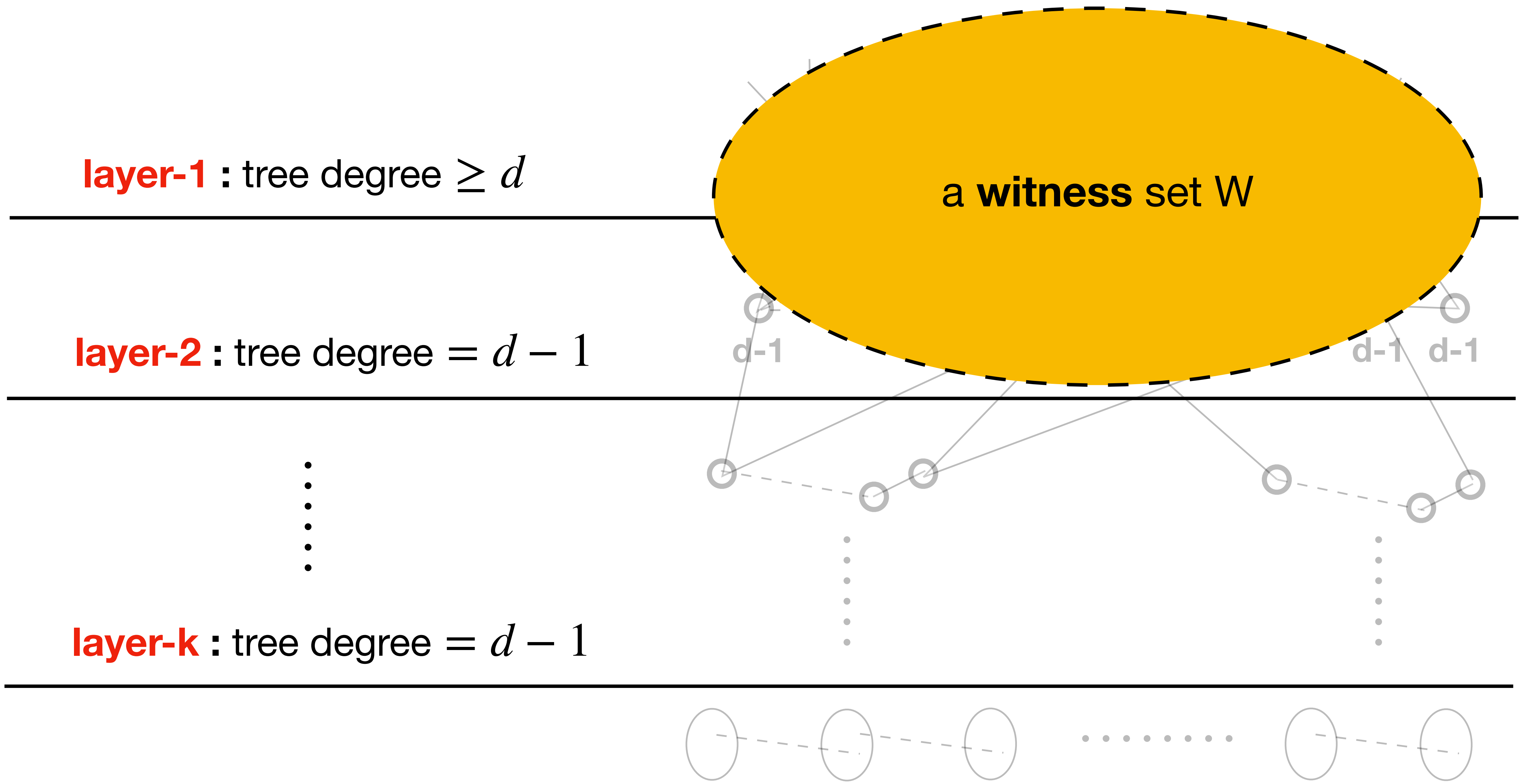
Multi-hop switches

- If $k > \log_{1+\epsilon} n$, either tree degree is reduced multiplicatively,
- or try the witness lemma **at each layer**, proving $\Delta^* \geq (1 - \epsilon)d - O(\log n)$



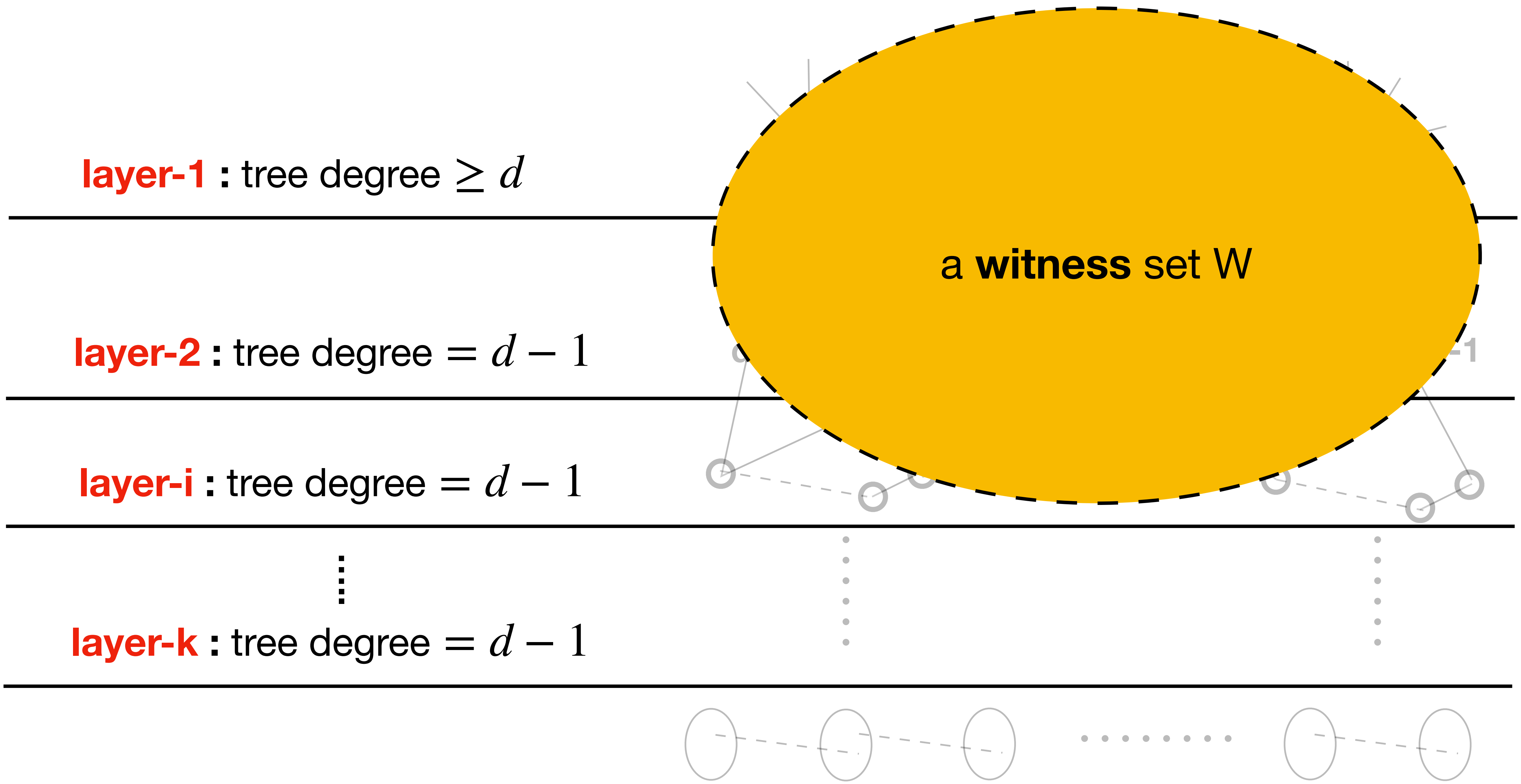
Multi-hop switches

- If $k > \log_{1+\epsilon} n$, either tree degree is reduced multiplicatively,
- or try the witness lemma **at each layer**, proving $\Delta^* \geq (1-\epsilon)d - O(\log n)$



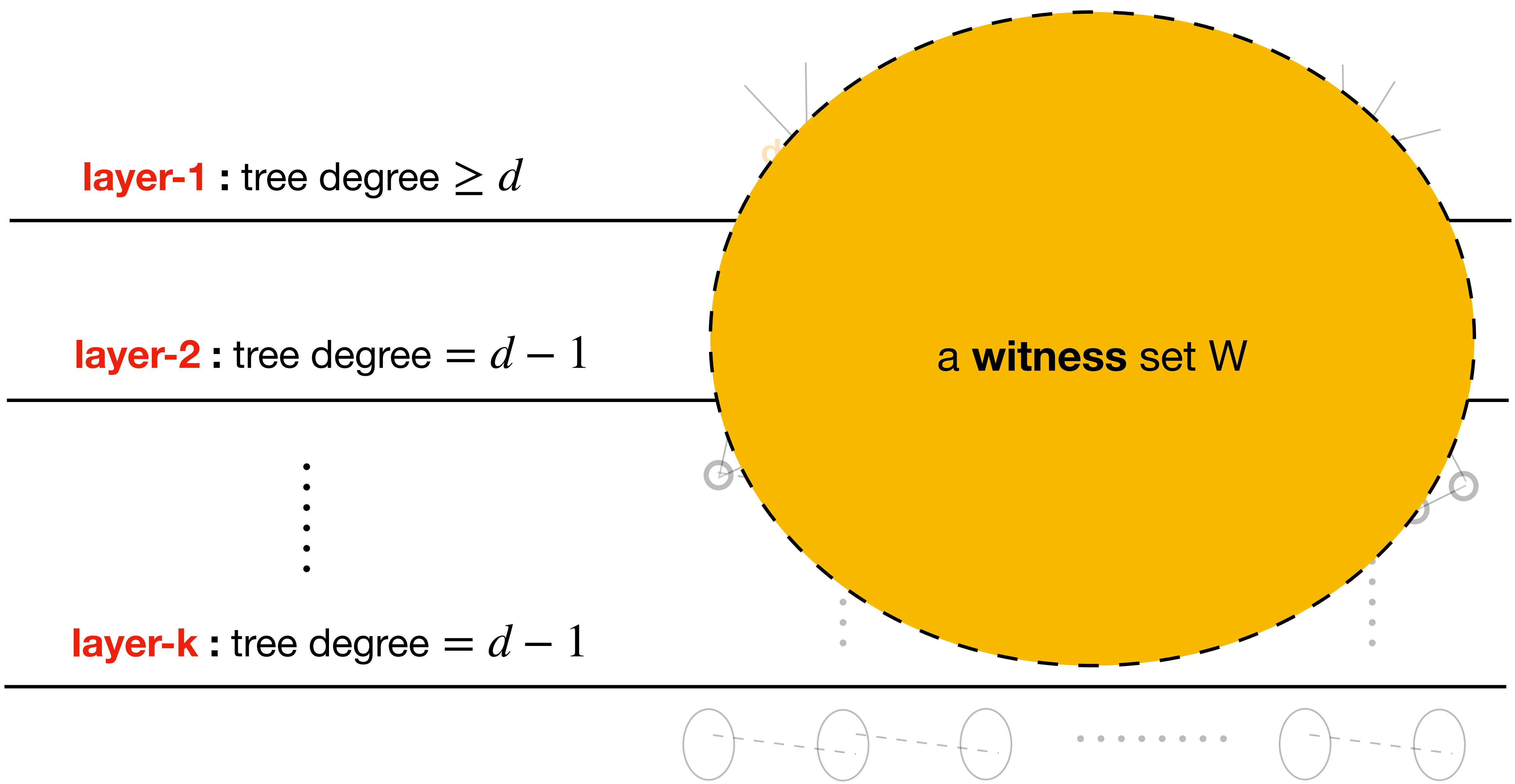
Multi-hop switches

- If $k > \log_{1+\epsilon} n$, either tree degree is reduced multiplicatively,
- or try the witness lemma **at each layer**, proving $\Delta^* \geq (1-\epsilon)d - O(\log n)$



Multi-hop switches

- If $k > \log_{1+\epsilon} n$, either tree degree is reduced multiplicatively,
- or try the witness lemma **at each layer**, proving $\Delta^* \geq (1-\epsilon)d - O(\log n)$



Thank you!