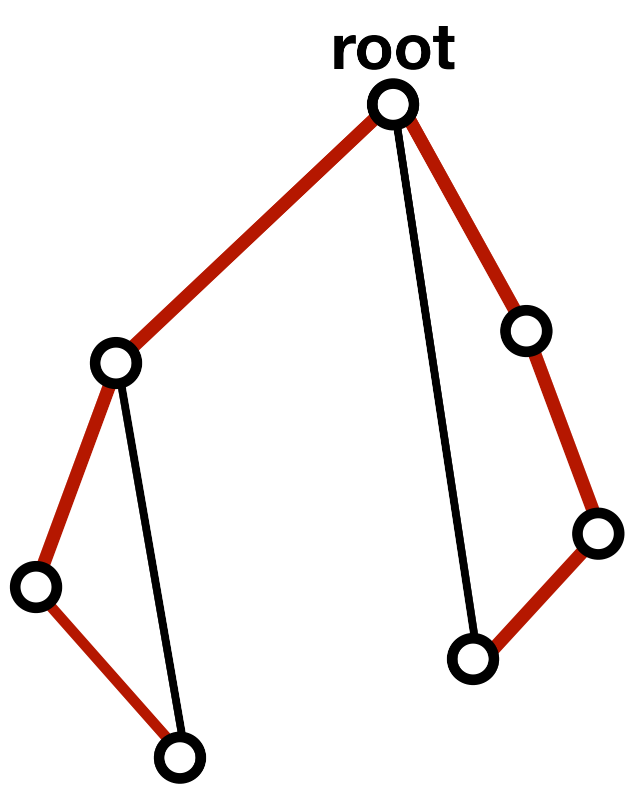
# An Improved Algorithm for Incremental DFS Tree in Undirected Graphs

Lijie Chen[1], Ran Duan[2], Ruosong Wang[3], Hanrui Zhang[4], **Tianyi Zhang[2]**

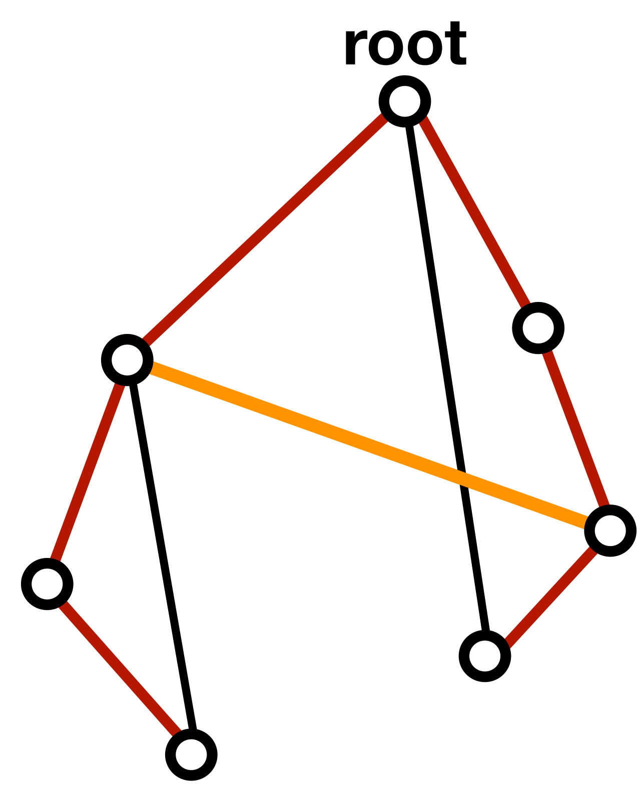
[1]MIT, [2]Tsinghua University, [3]CMU, [4]Duke University

# Definition: DFS tree

- Given an undirected graph $G = (V, E)$ with a designated root
- DFS tree: a maximal tree containing the root, where every non-tree edge connects an ancestor and a descendant




**A DFS tree**




**Not a DFS tree**

# Definition: Incremental DFS tree

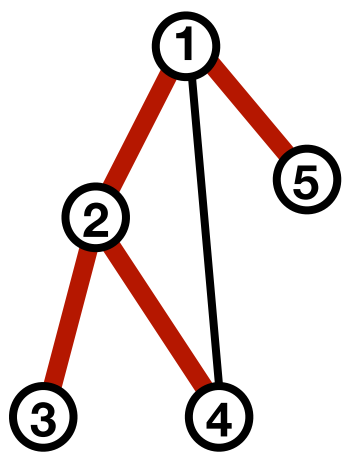
Data structure

- Maintain a DFS tree *T* in graph *G*

Update operations

- **Input:** insert an edge/vertex to *G*
- **Output:** print all edges of *T*

# Example: Incremental DFS tree

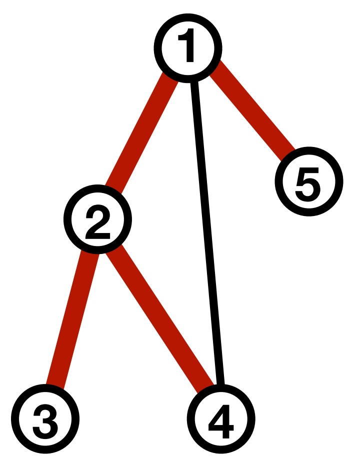
| | | | | |
|---|---|---|---|---|
| **Input:** Updates to *G* | | | | |
| **Picture** |   | | | |
| **Output:** Change in *T* | | | | |

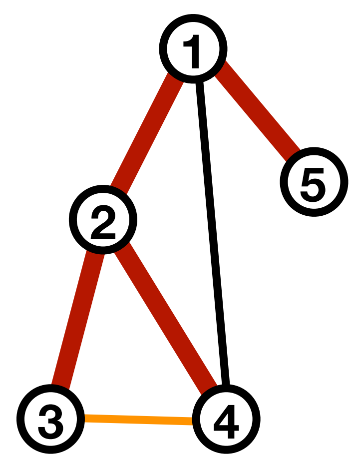
# Example: Incremental DFS tree

**Input:**
Updates to *G*

Insert(3,4)

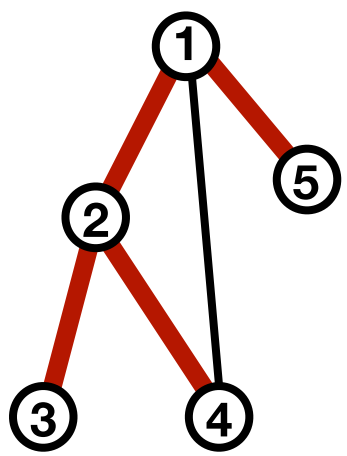
**Picture**

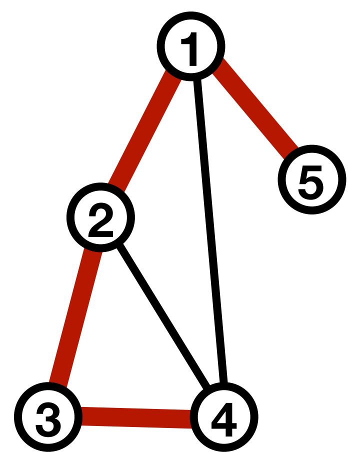



**Output:**
Change in *T*

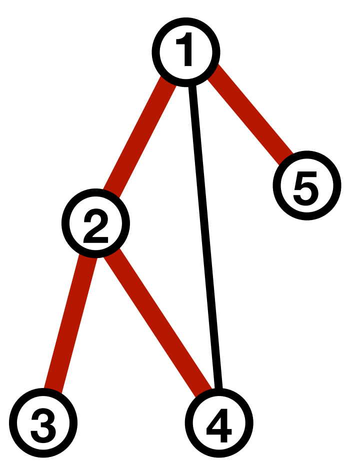
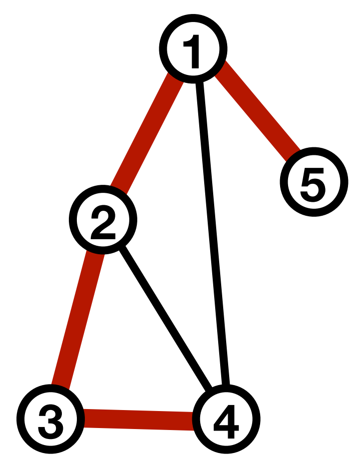
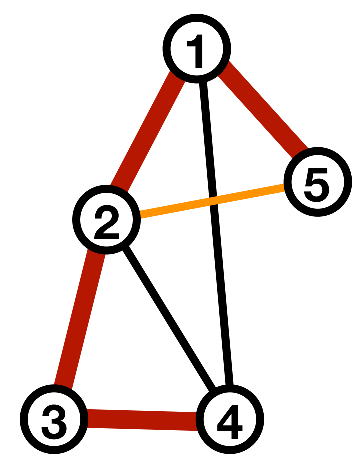
# Example: Incremental DFS tree

<table>
<tr><td>Input:<br>Updates to G</td><td></td><td>Insert(3,4)</td><td></td><td></td></tr>
<tr><td>Picture</td><td>


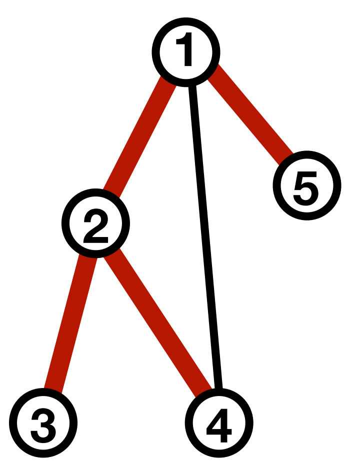
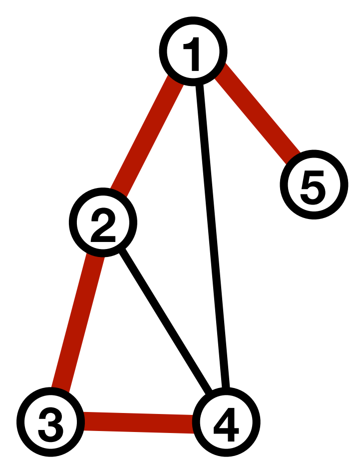
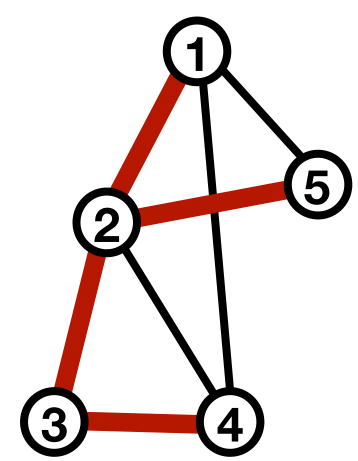
</td><td>


</td><td></td><td></td></tr>
<tr><td>Output:<br>Change in T</td><td></td><td>Delete(2,4)<br>Insert(3,4)</td><td></td><td></td></tr>
</table>

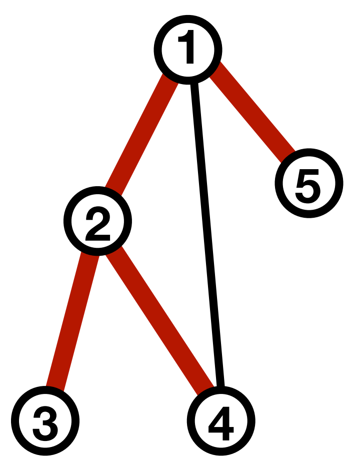
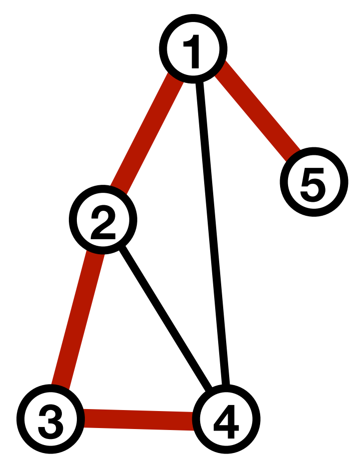
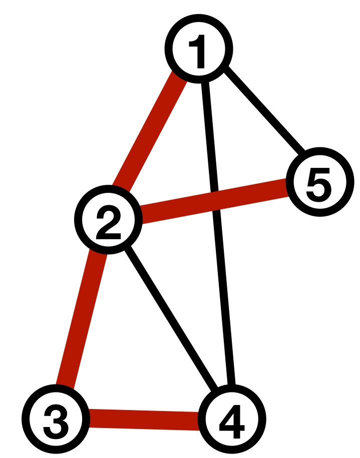
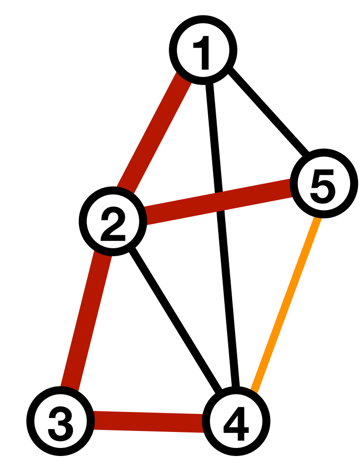
# Example: Incremental DFS tree

| | | | | |
|---|---|---|---|---|
| **Input:** Updates to *G* | | Insert(3,4) | Insert(2,5) | |
| **Picture** |   |   |   | |
| **Output:** Change in *T* | | Delete(2,4)<br>Insert(3,4) | | |

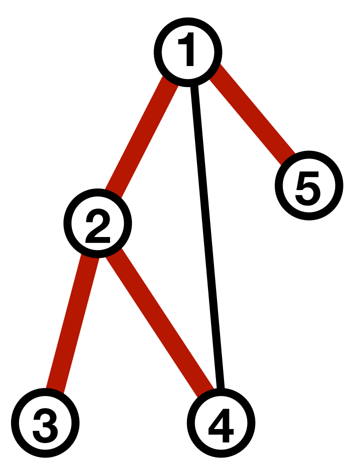
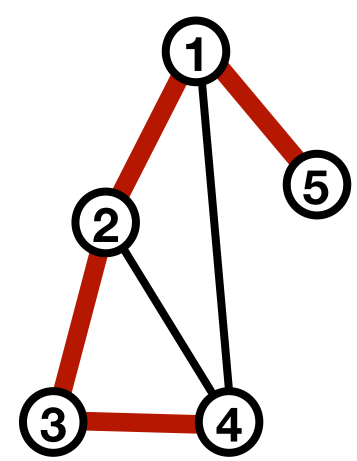
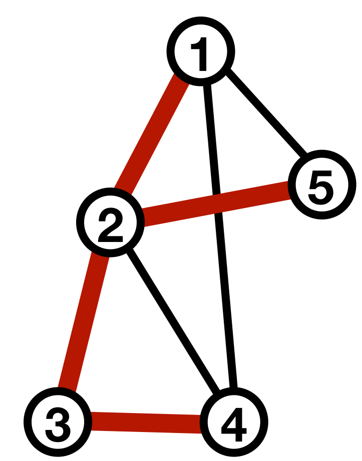
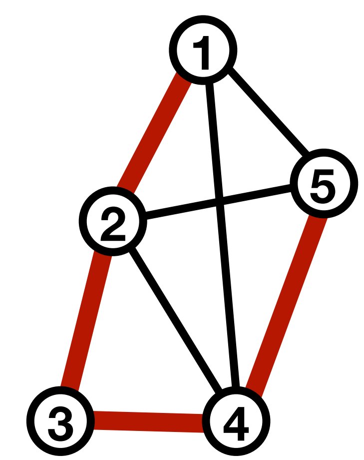
# Example: Incremental DFS tree

| | | | | |
|---|---|---|---|---|
| **Input:**<br>Updates to *G* | | Insert(3,4) | Insert(2,5) | |
| **Picture** |  |  |  | |
| **Output:**<br>Change in *T* | | Delete(2,4)<br>Insert(3,4) | Delete(1,5)<br>Insert(2,5) | |

# Example: Incremental DFS tree

| | | | | |
|---|---|---|---|---|
| **Input:** Updates to *G* | | Insert(3,4) | Insert(2,5) | Insert(4,5) |
| **Picture** |  |  |  |  |
| **Output:** Change in *T* | | Delete(2,4)<br>Insert(3,4) | Delete(1,5)<br>Insert(2,5) | |

# <u>Example</u>: Incremental DFS tree

| | | | | |
|---|---|---|---|---|
| **Input:**<br>Updates to $G$ | | Insert(3,4) | Insert(2,5) | Insert(4,5) |
| **Picture** |  |  |  |  |
| **Output:**<br>Change in $T$ | | Delete(2,4)<br>Insert(3,4) | Delete(1,5)<br>Insert(2,5) | Delete(2,5)<br>Insert(4,5) |

# Progress on incremental DFS

- *n* = # of vertices, *m* = # of edges

| Reference | Naïve | [BK'14] | [BCC+'16] | [NS'17] | New |
|---|---|---|---|---|---|
| Update time | $O(m+n)$ | $O(n)$ | $O(n \log^3 n)$ | $O(n \log n)$ | $O(n)$ |
| Space | $O(m+n)$ | $O(m+n)$ | $O(m \log n)$ | $O(m \log n)$ in bits | $O(m \log n)$ |
| Worst-case? | Yes | No | Yes | Yes | Yes |

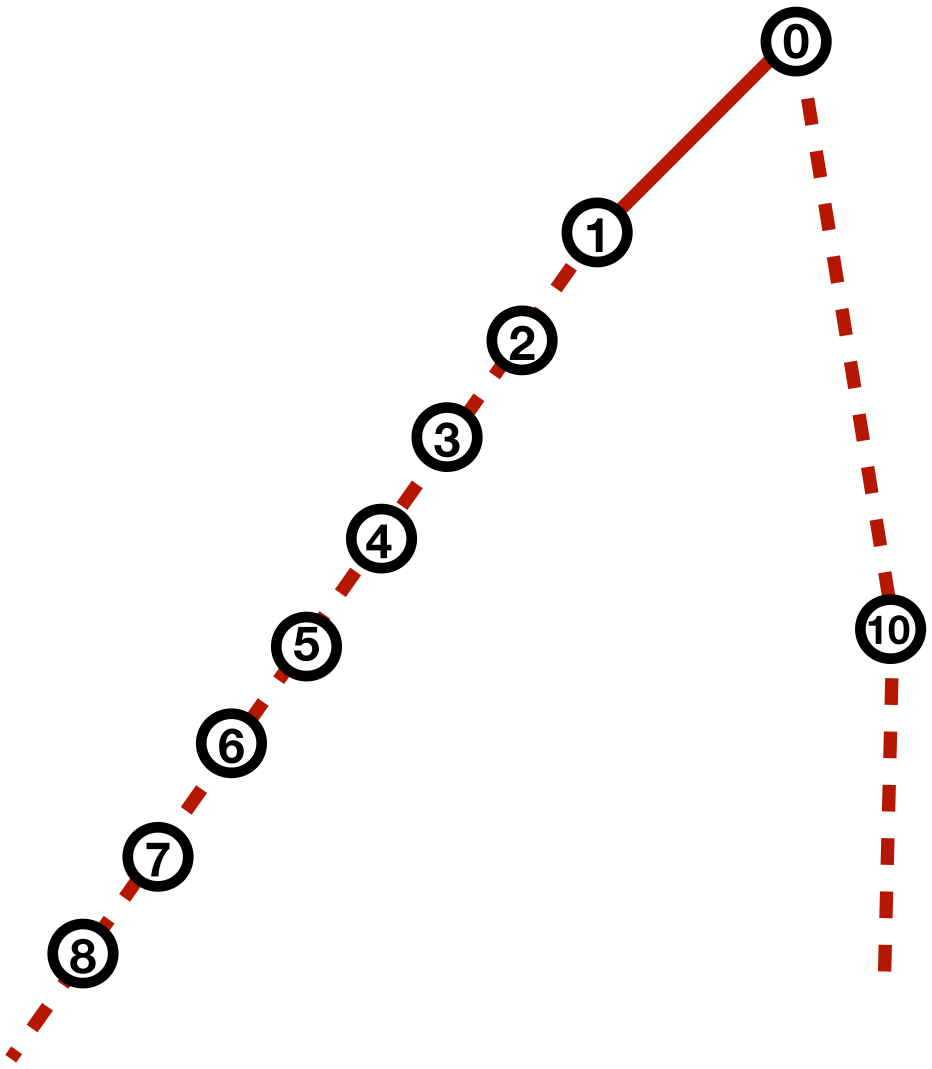
# Reduction to batch insertion

Main Theorem:

- Preprocess graph *G* in $O(\min\{m \log n, n^2\})$ time
- **Input:** a set *U* of *k* edge insertions
- **Output:** a DFS tree of *G*+*U* in $O(n+k)$

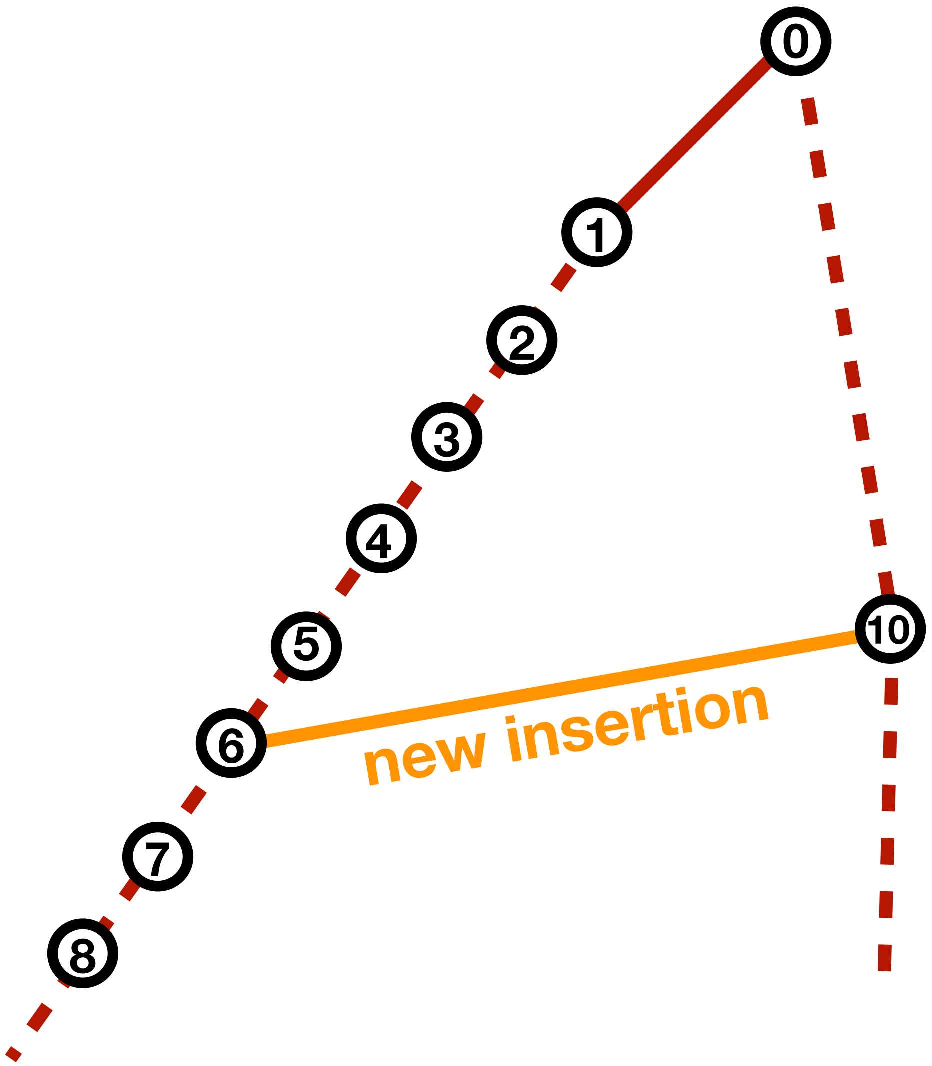
# Batch insertions [BCC+'16]

Revert & reroot



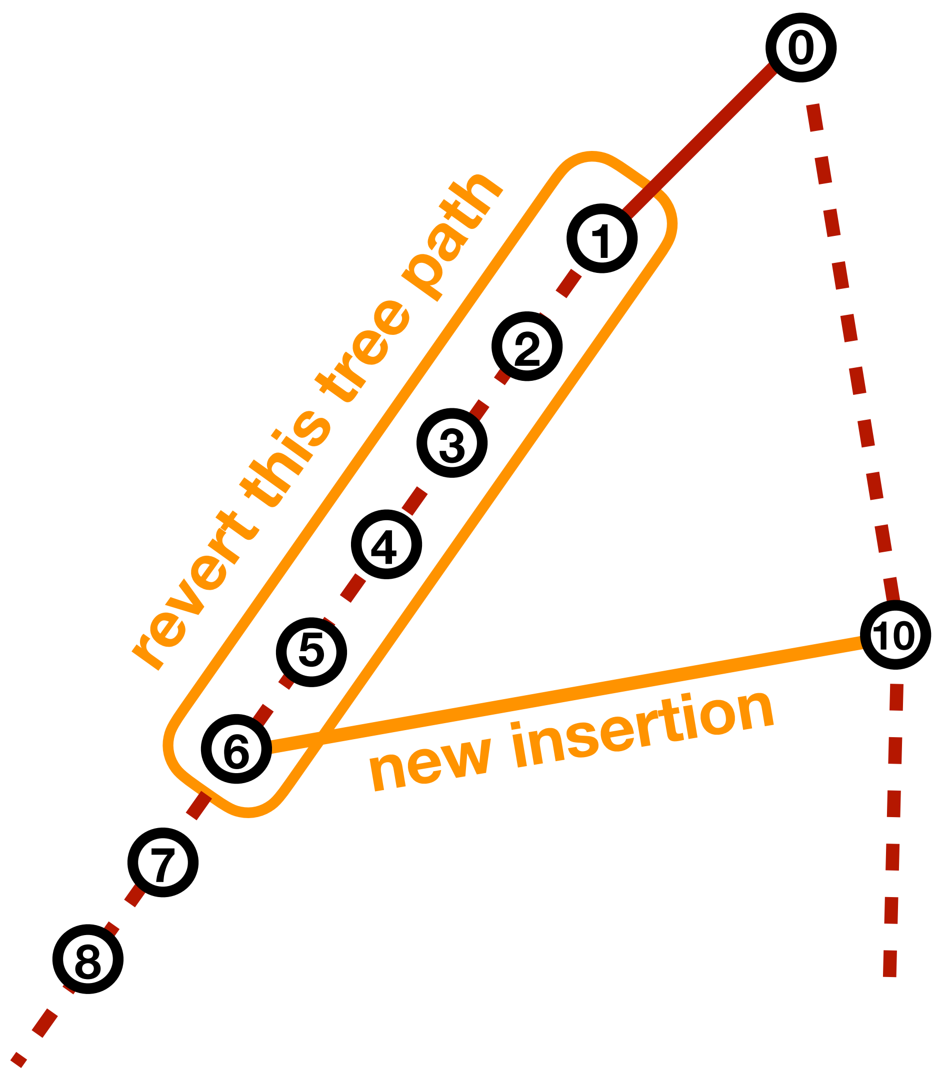
# Batch insertions [BCC+'16]

Revert & reroot



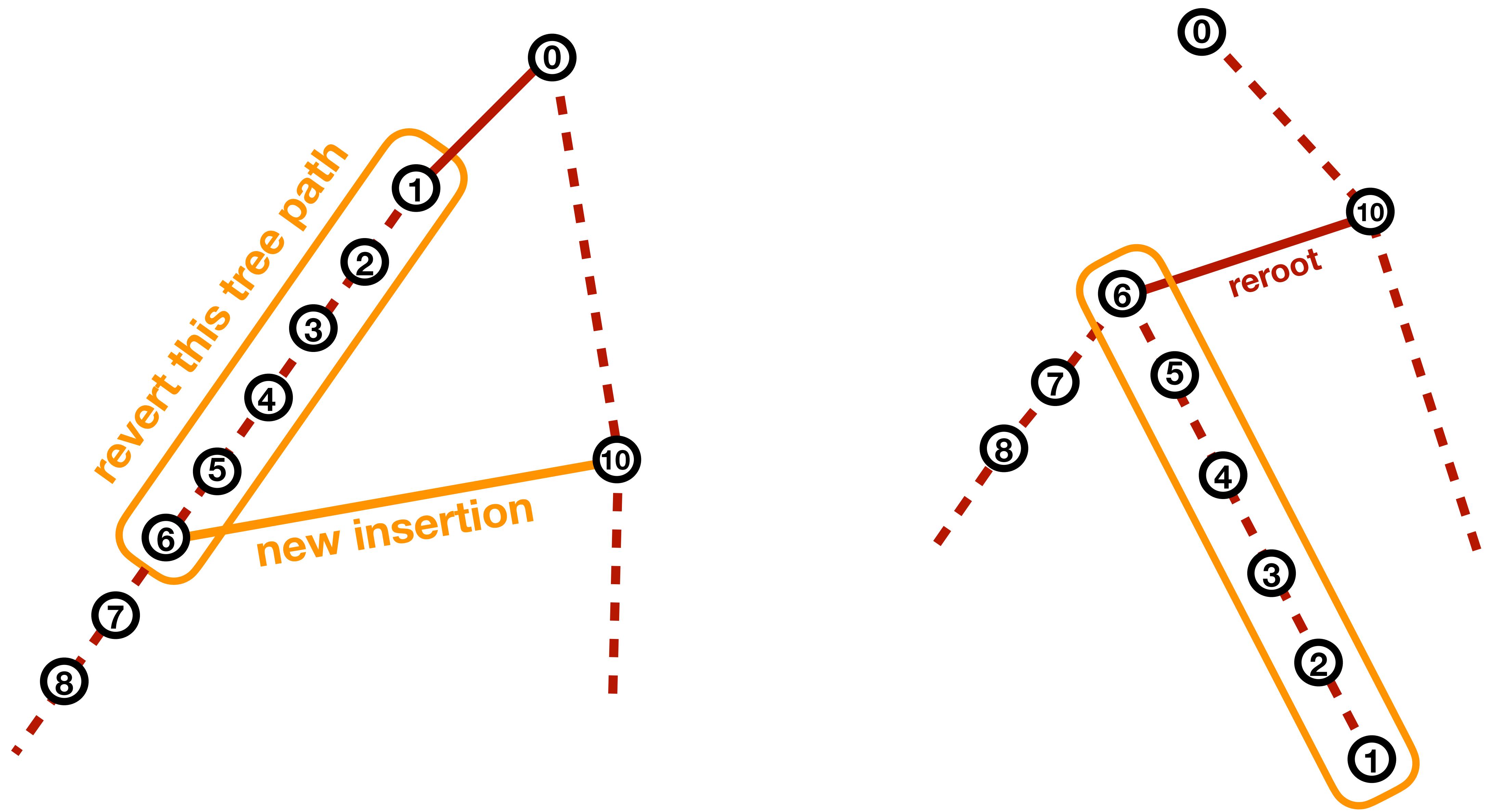
# Batch insertions [BCC+'16]

Revert & reroot



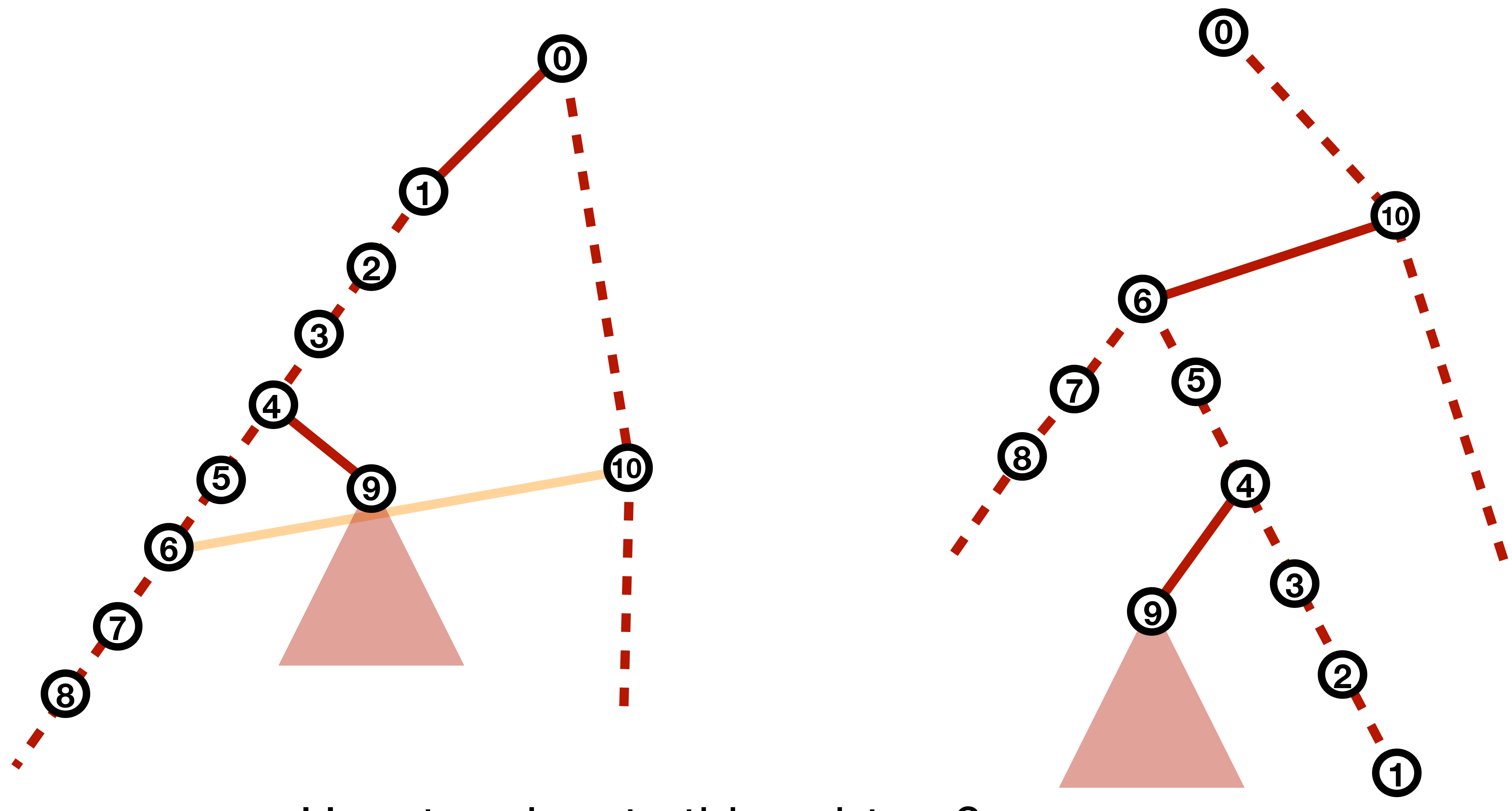
# Batch insertions [BCC+'16]

Revert & reroot



# Batch insertions [BCC+'16]

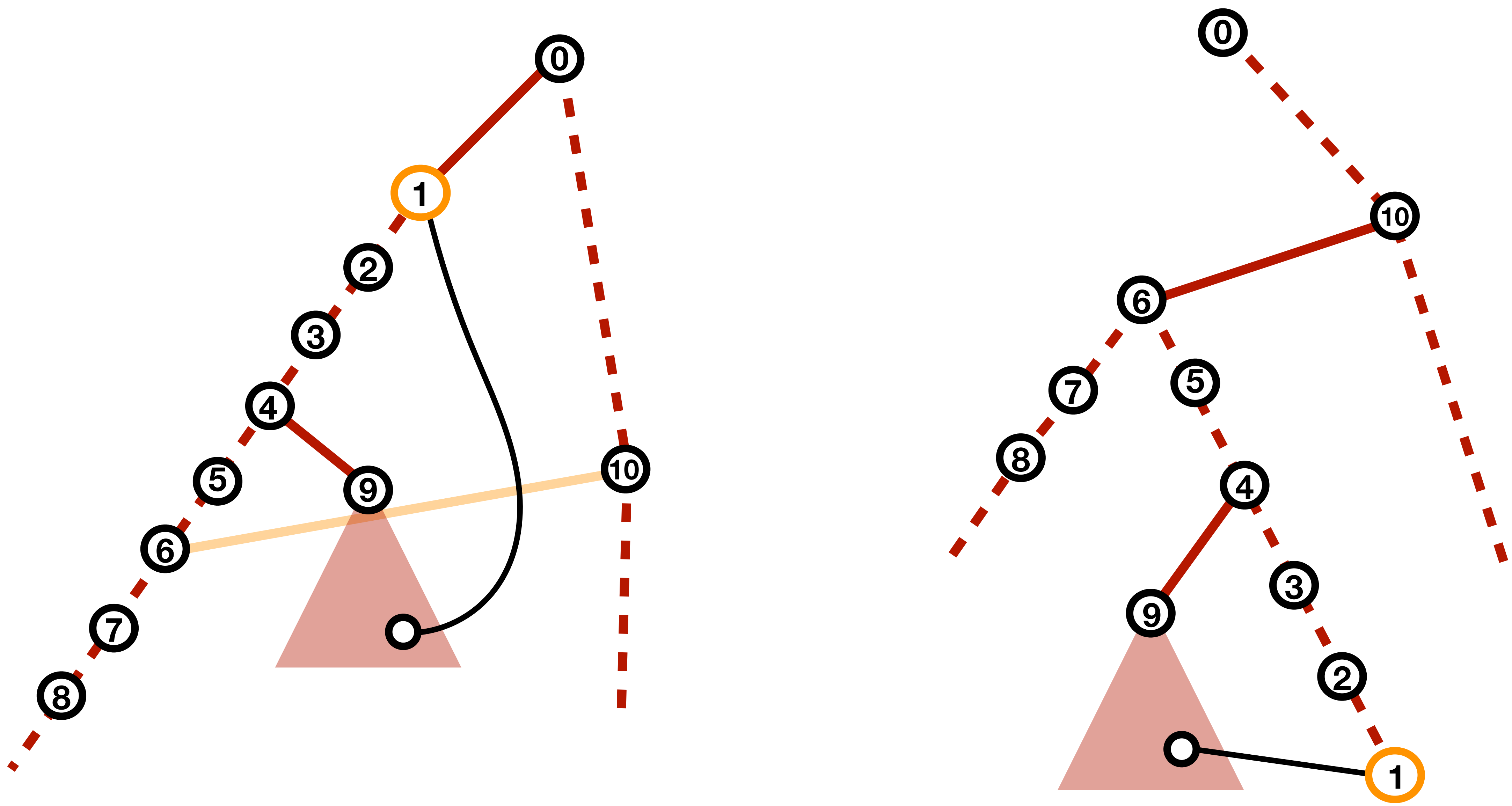
Revert & reroot




How to relocate this subtree?

# Batch insertions [BCC+'16]

Revert & reroot

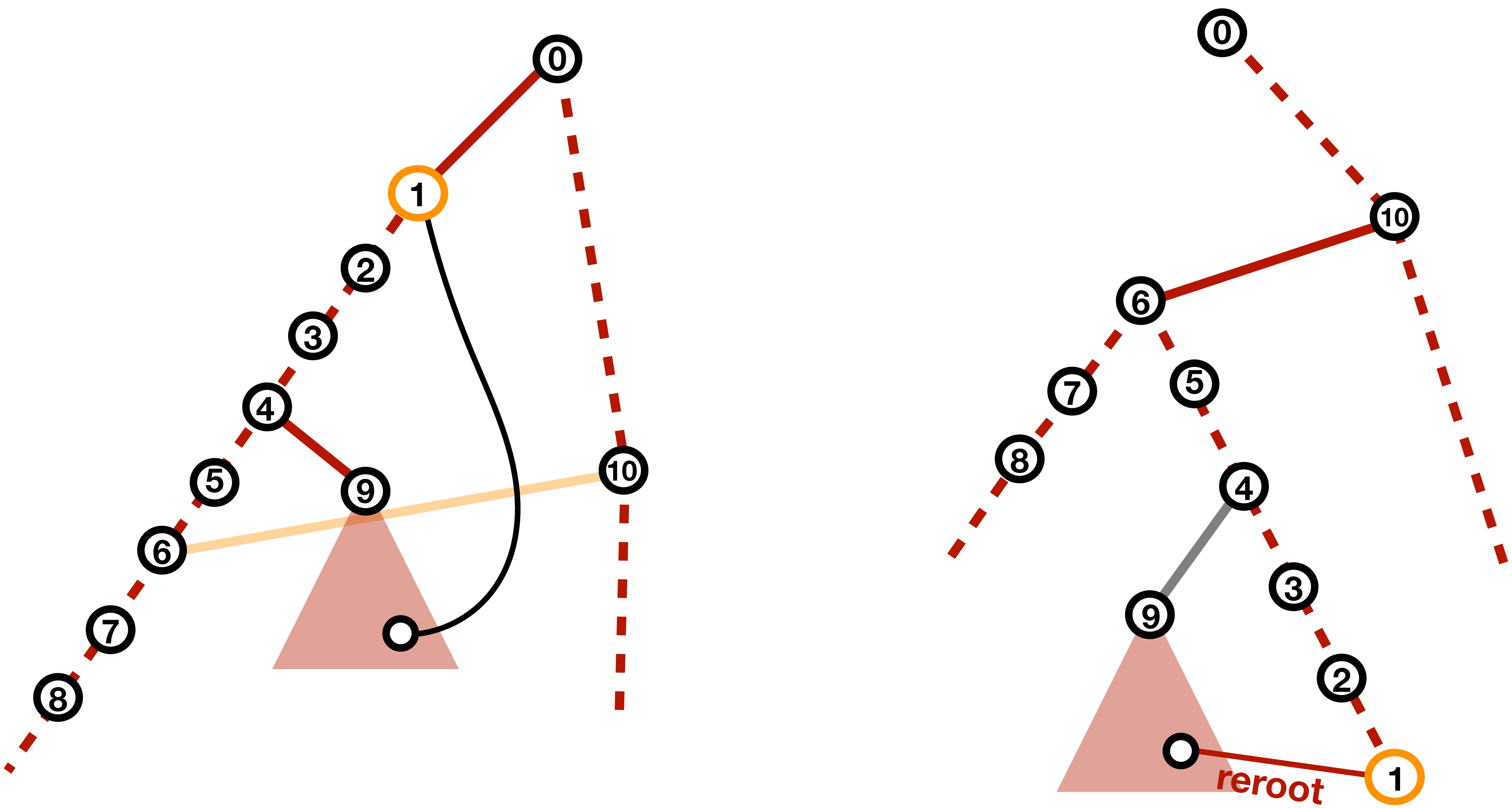



How to relocate this subtree?

Find the highest ancestor…

# Batch insertions [BCC+'16]

Revert & reroot

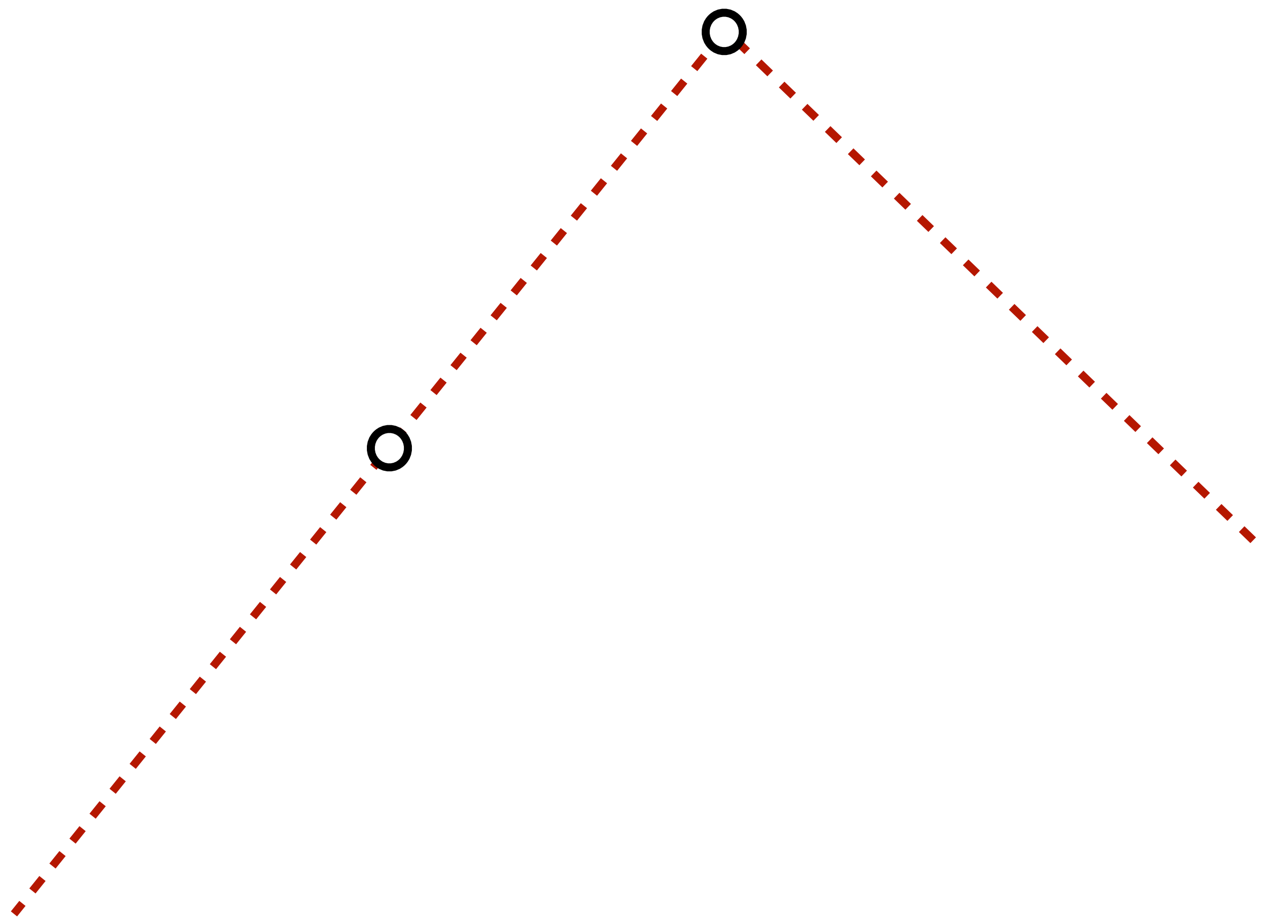



How to relocate this subtree?

Find the highest ancestor and reroot again.

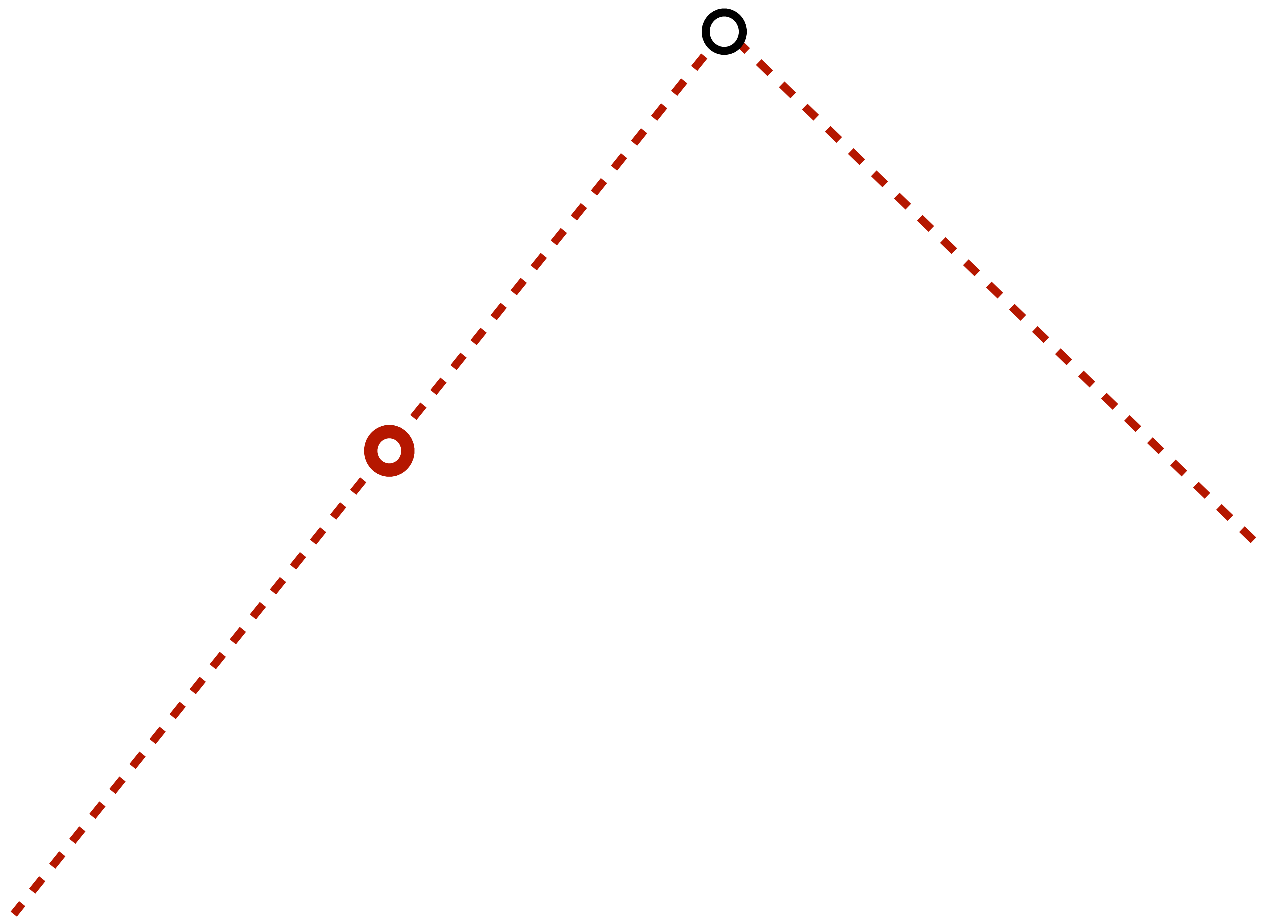
# Batch insertions [BCC+'16]

Recursively revert & reroot



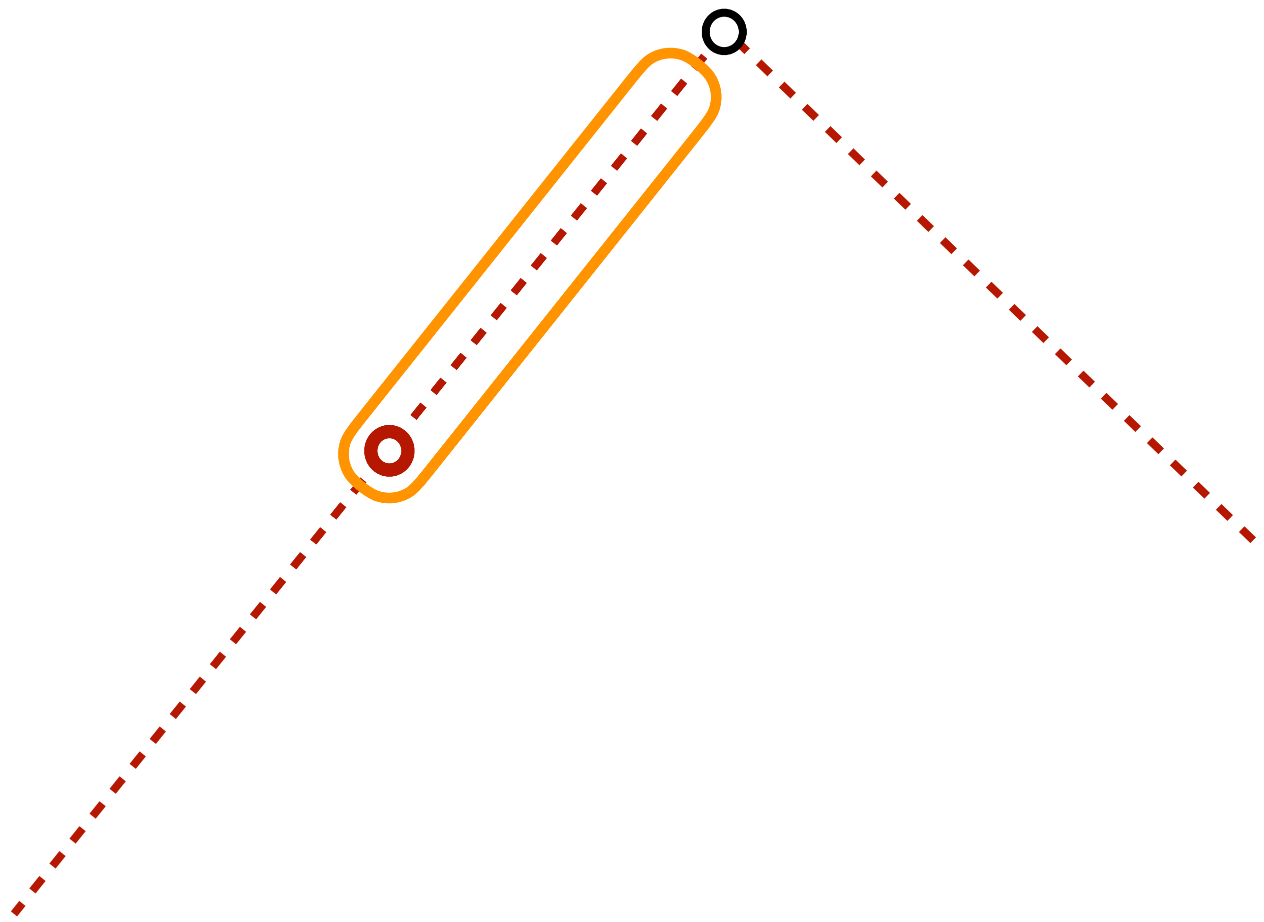
# Batch insertions [BCC+'16]

Recursively revert & reroot



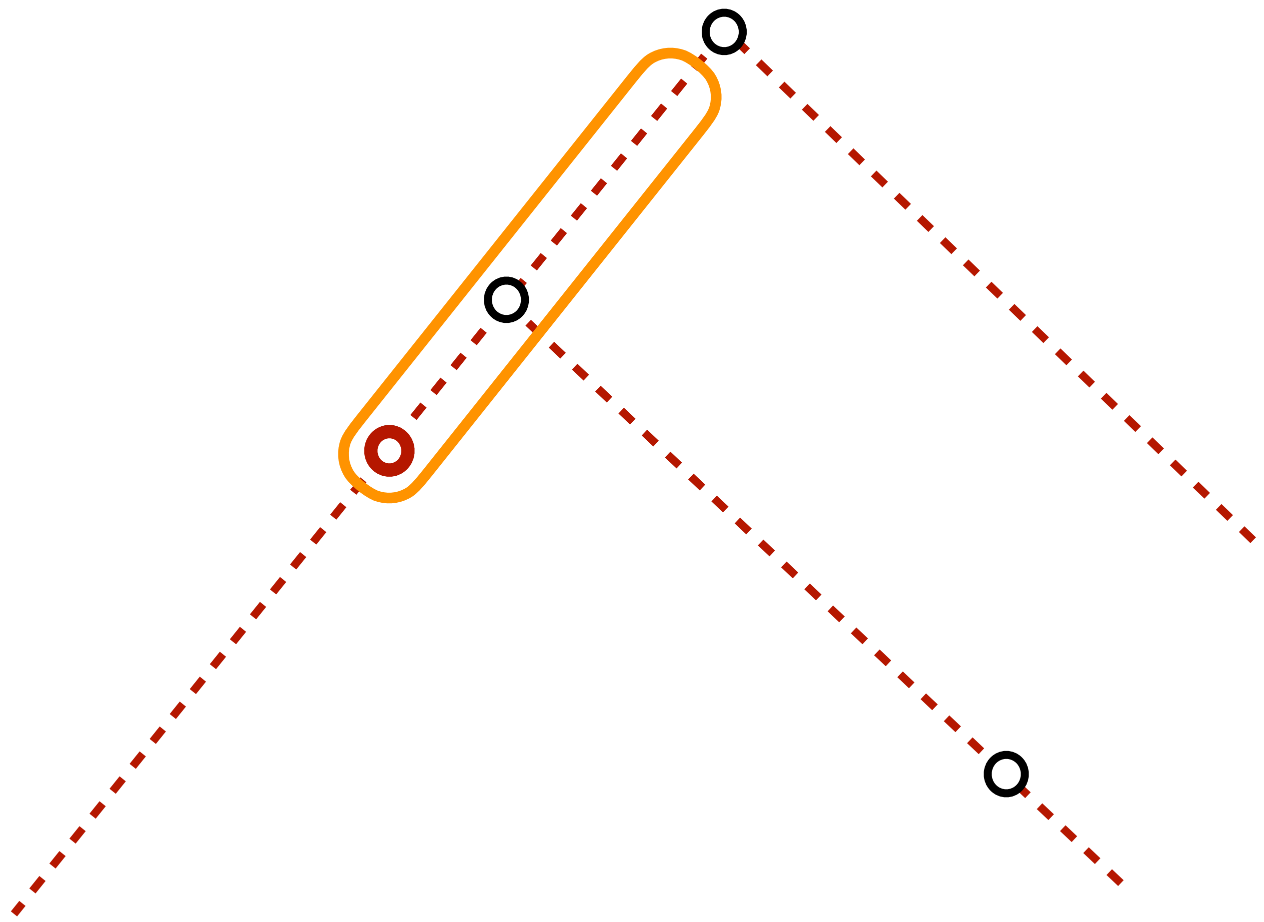
# Batch insertions [BCC+'16]

Recursively revert & reroot



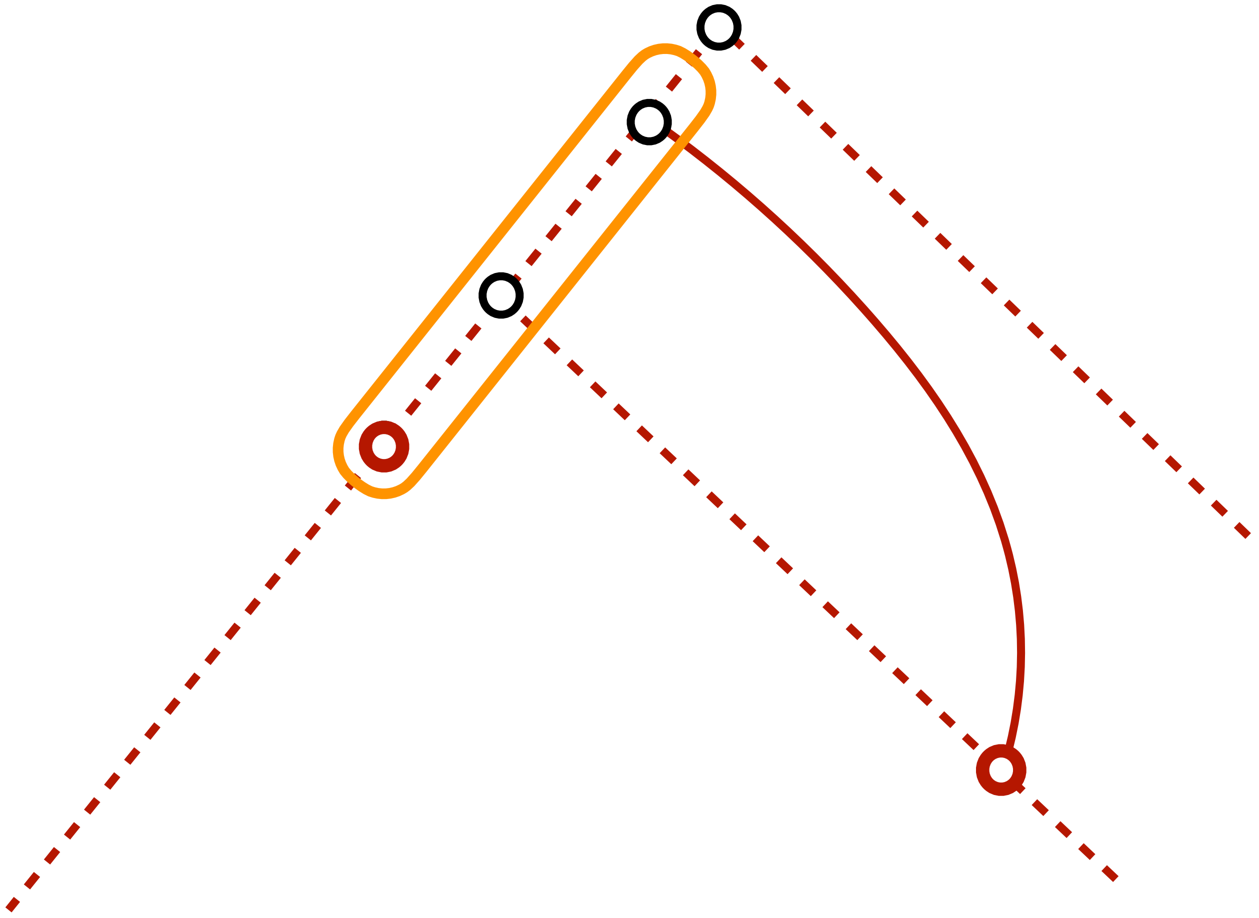
# Batch insertions [BCC+'16]

Recursively revert & reroot



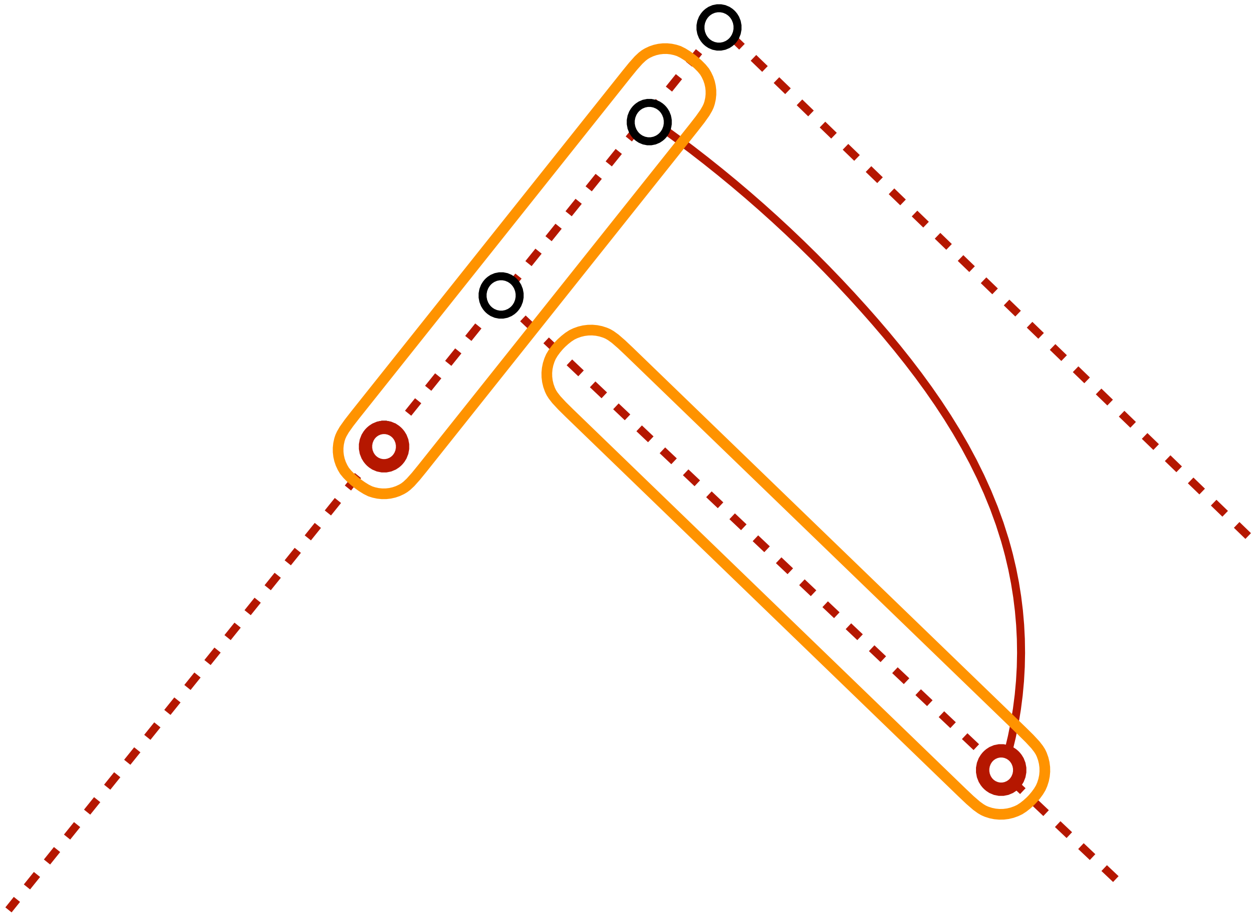
# Batch insertions [BCC+'16]

Recursively revert & reroot



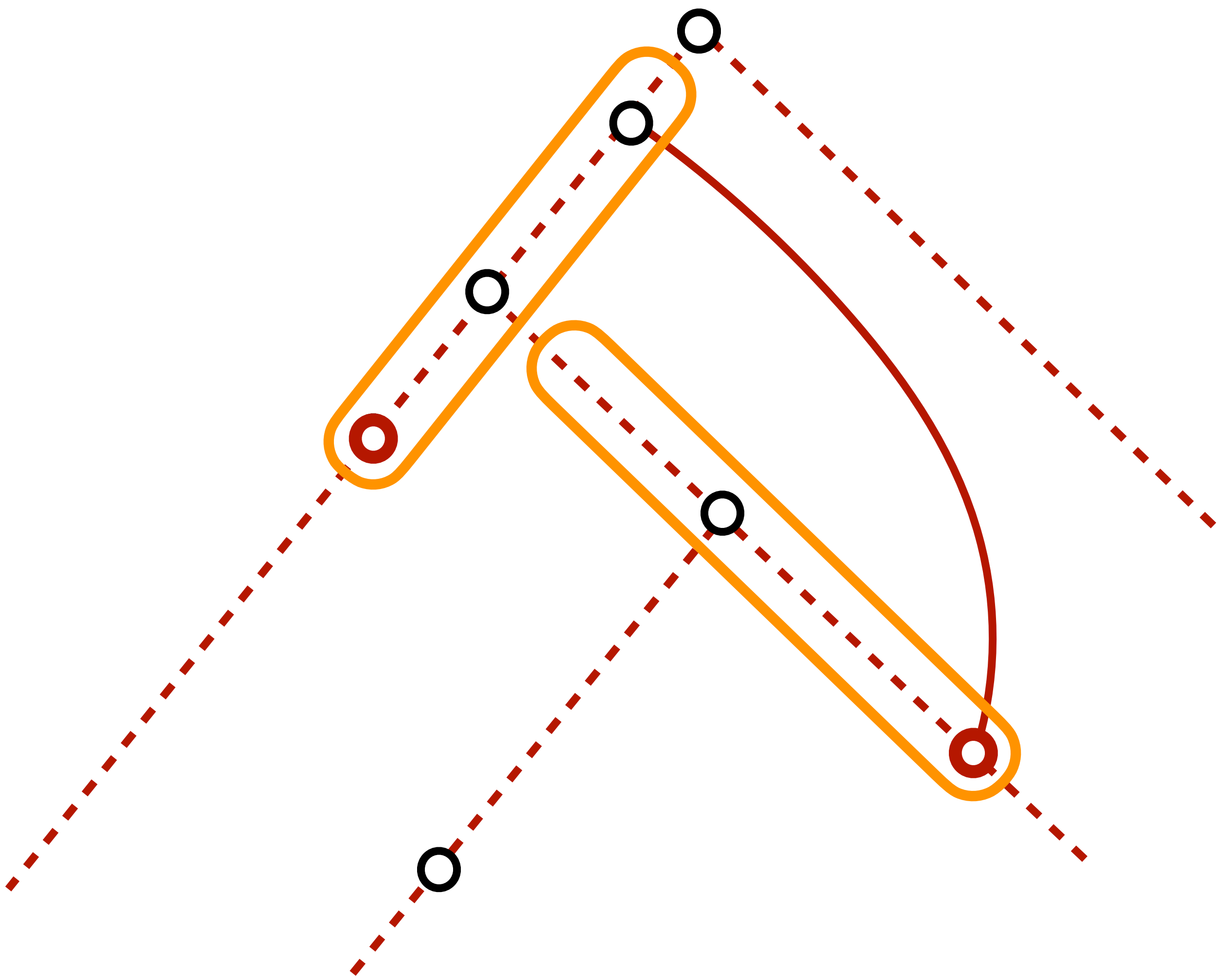
# Batch insertions [BCC+'16]

Recursively revert & reroot



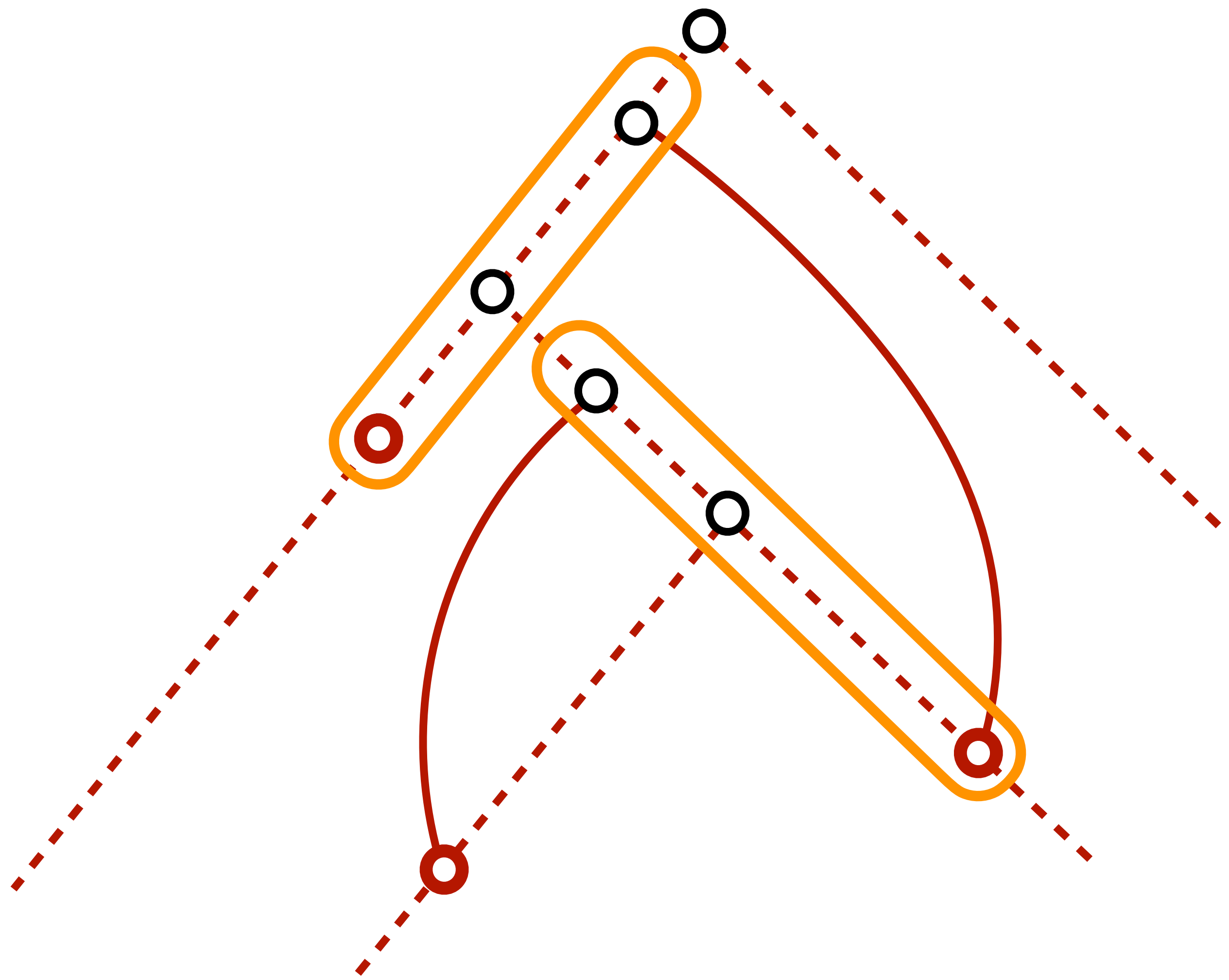
# Batch insertions [BCC+'16]

Recursively revert & reroot



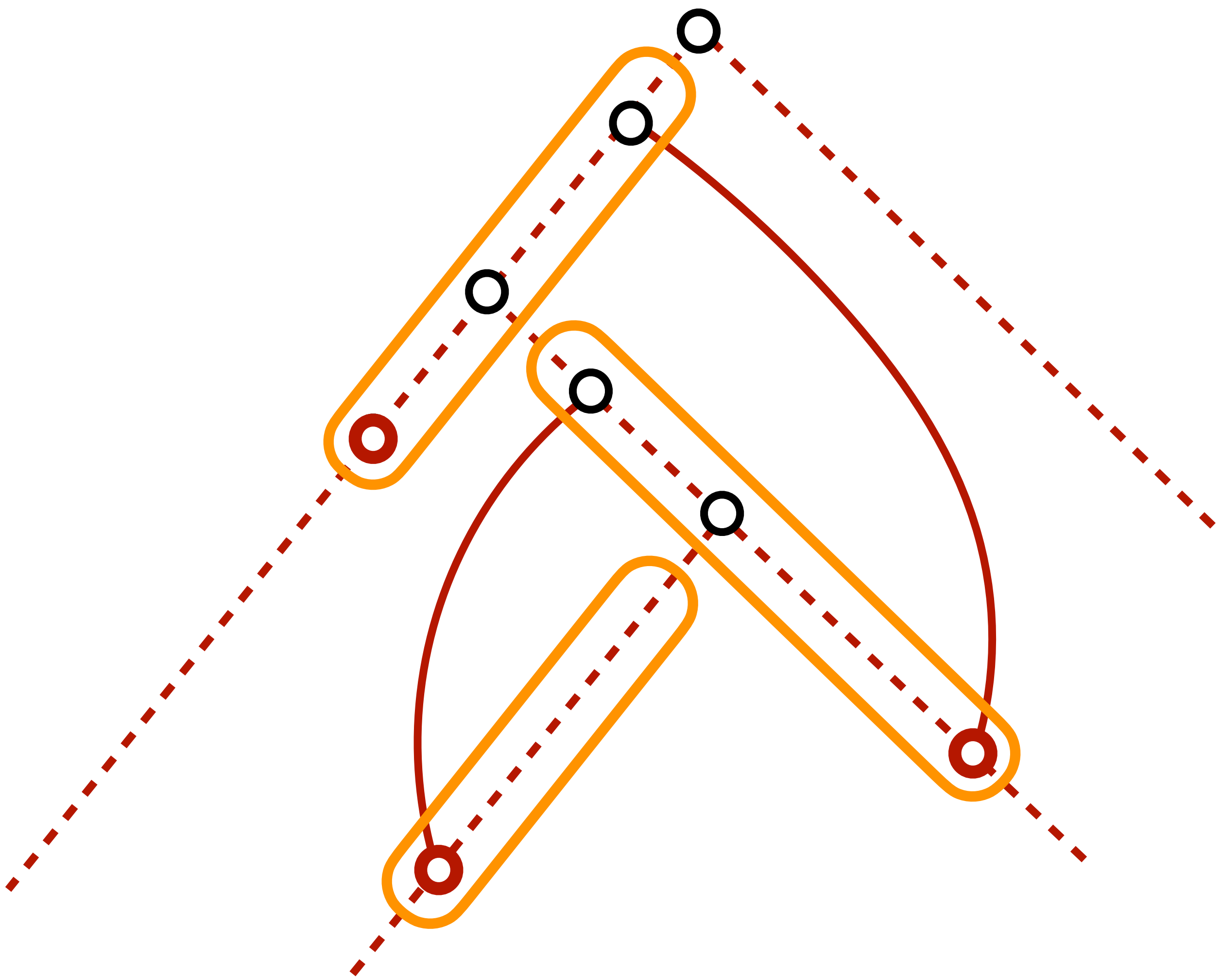
# Batch insertions [BCC+'16]

Recursively revert & reroot



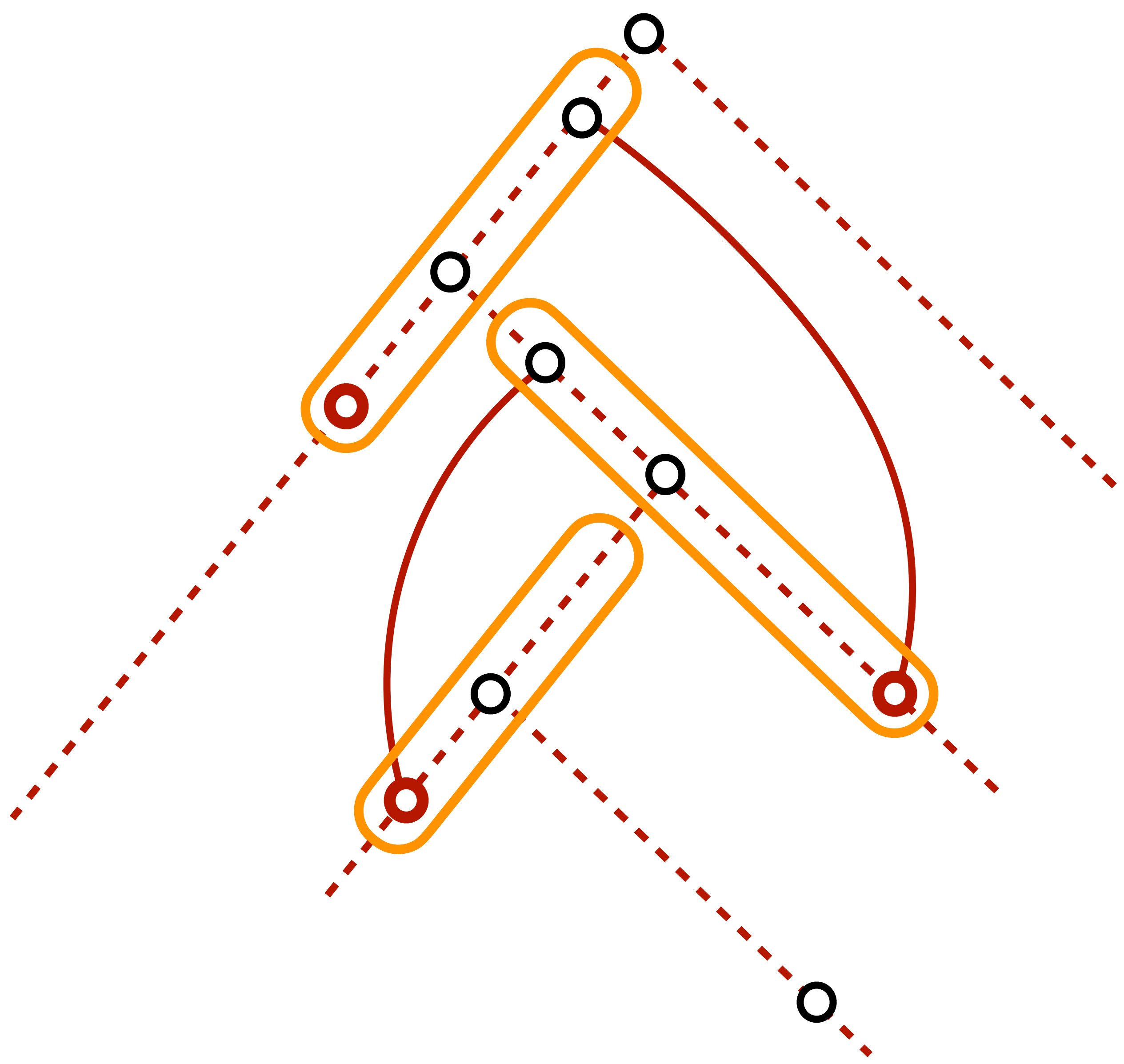
# Batch insertions [BCC+'16]

Recursively revert & reroot



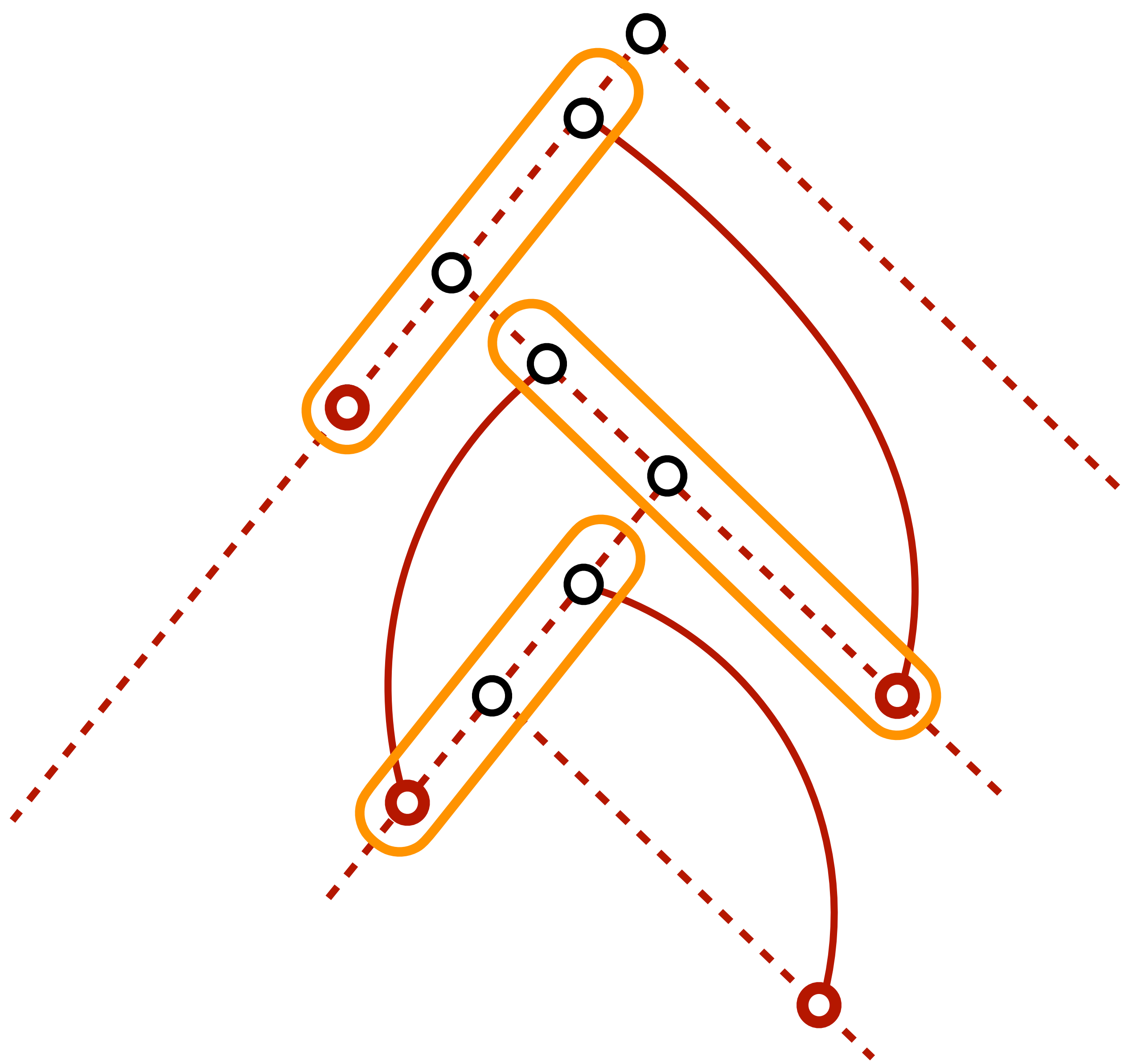
# Batch insertions [BCC+'16]

Recursively revert & reroot



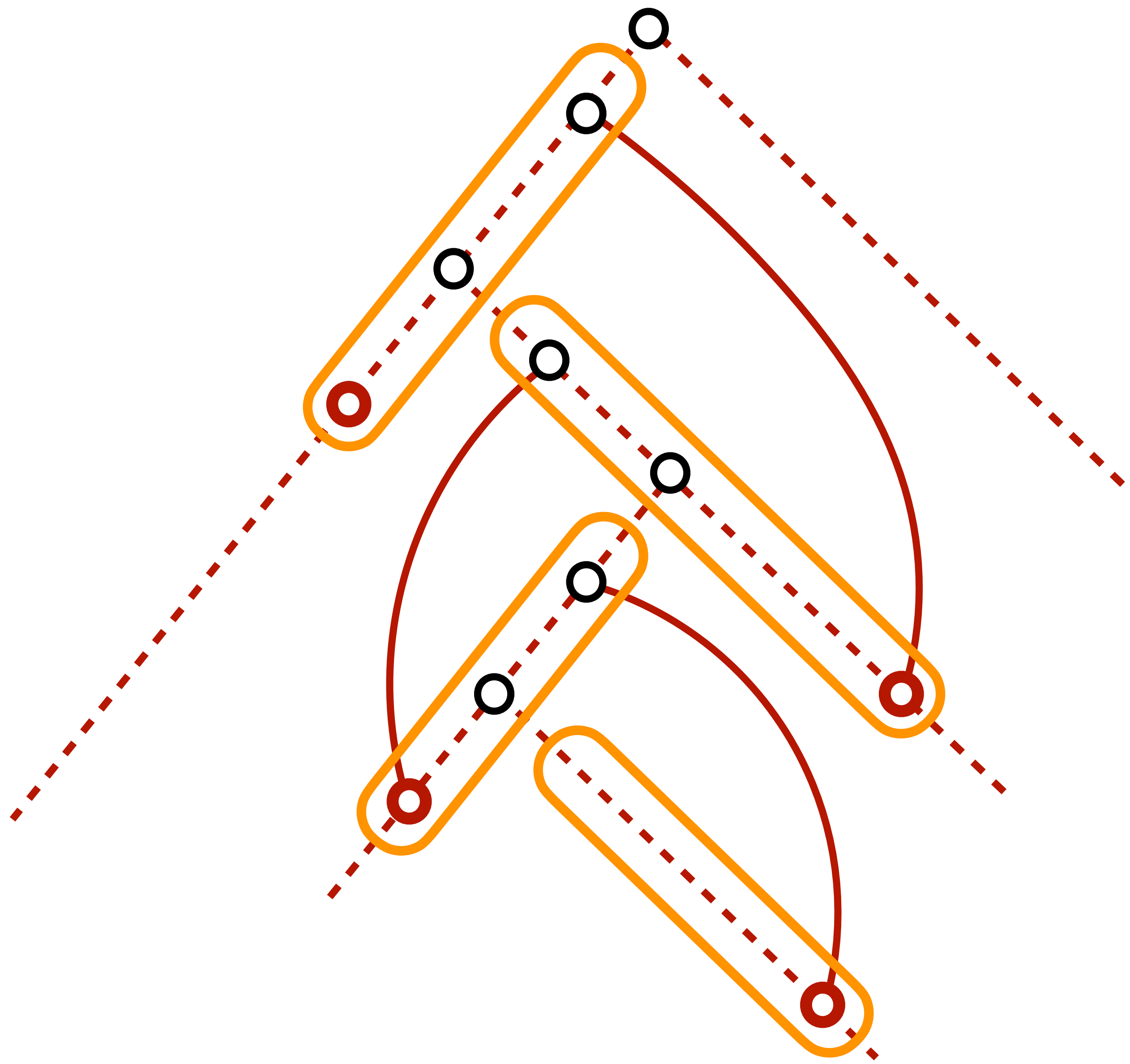
# Batch insertions [BCC+'16]

Recursively revert & reroot



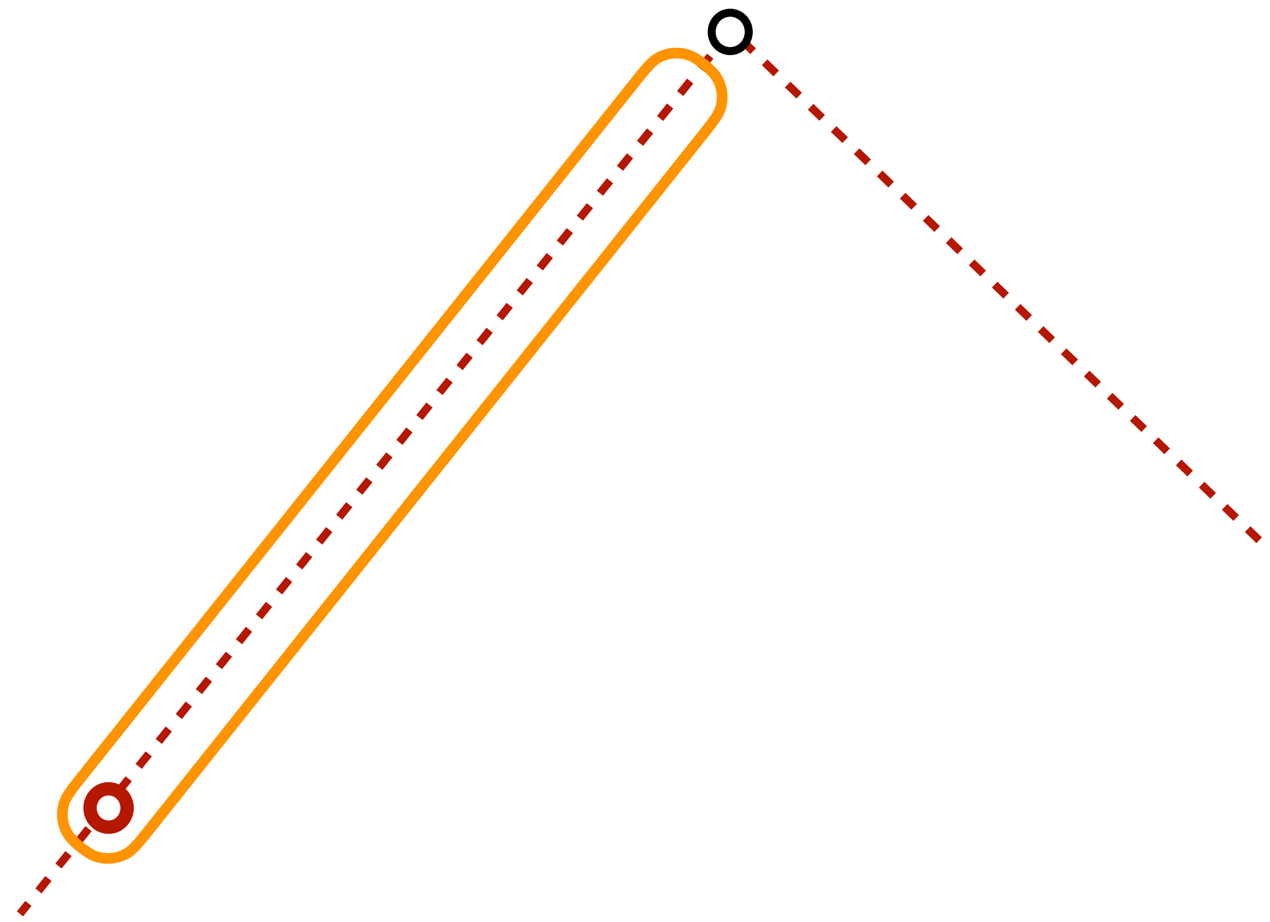
# Batch insertions [BCC+'16]

Recursively revert & reroot



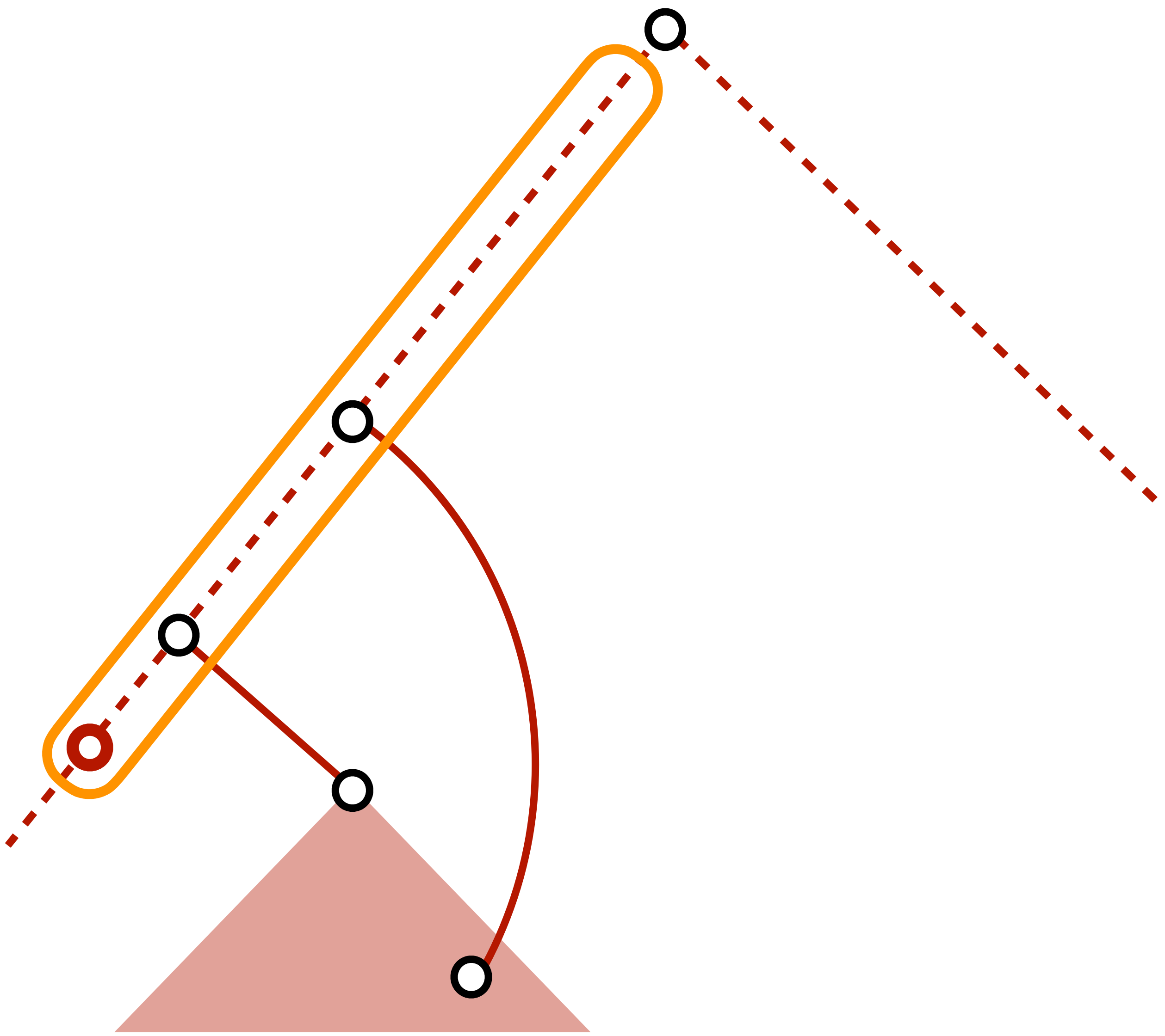
# Batch insertions [BCC+'16]

**Bottleneck:** finding highest ancestors on reverted paths



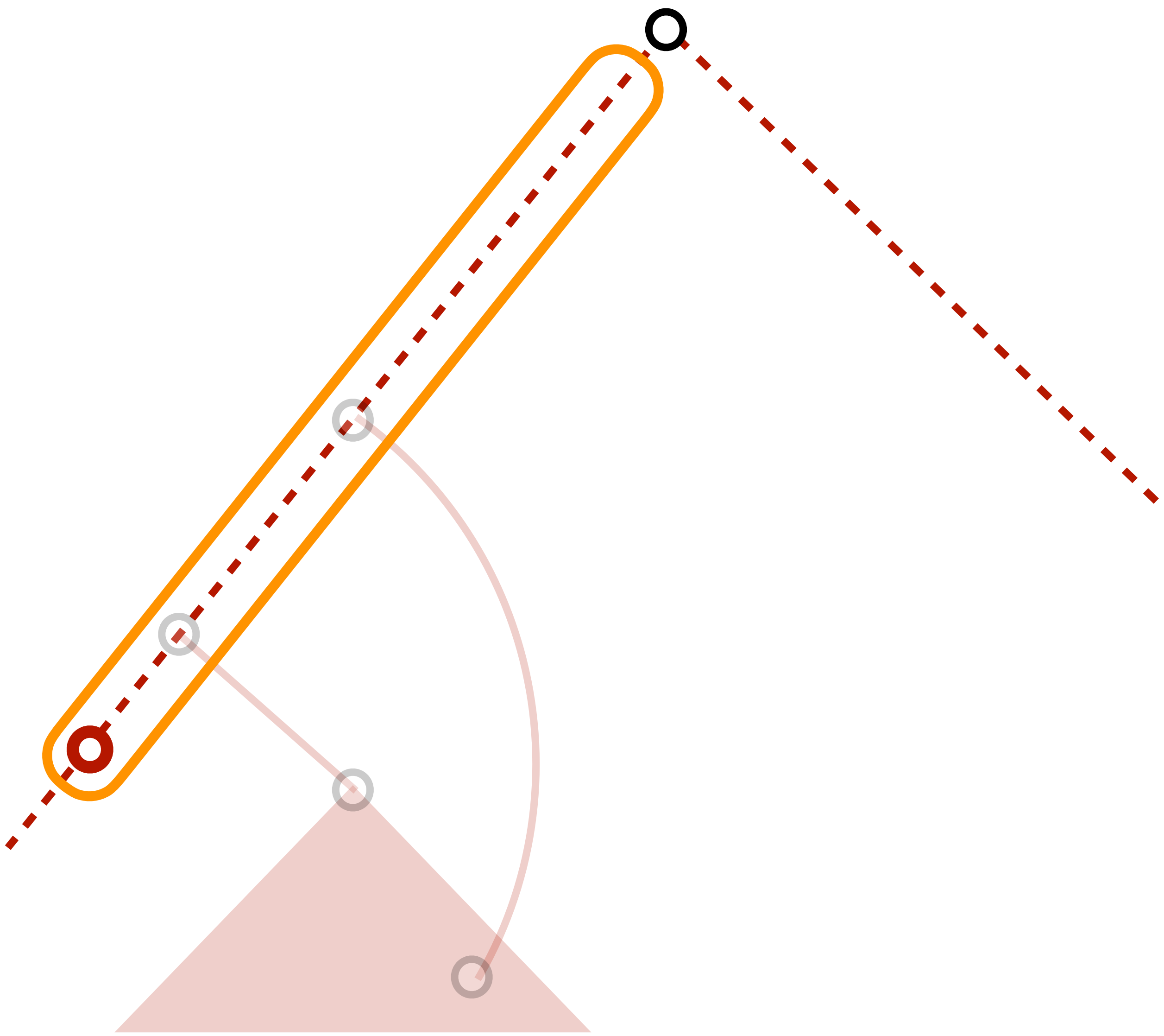
# Batch insertions [BCC+'16]

**Bottleneck:** finding highest ancestors on reverted paths



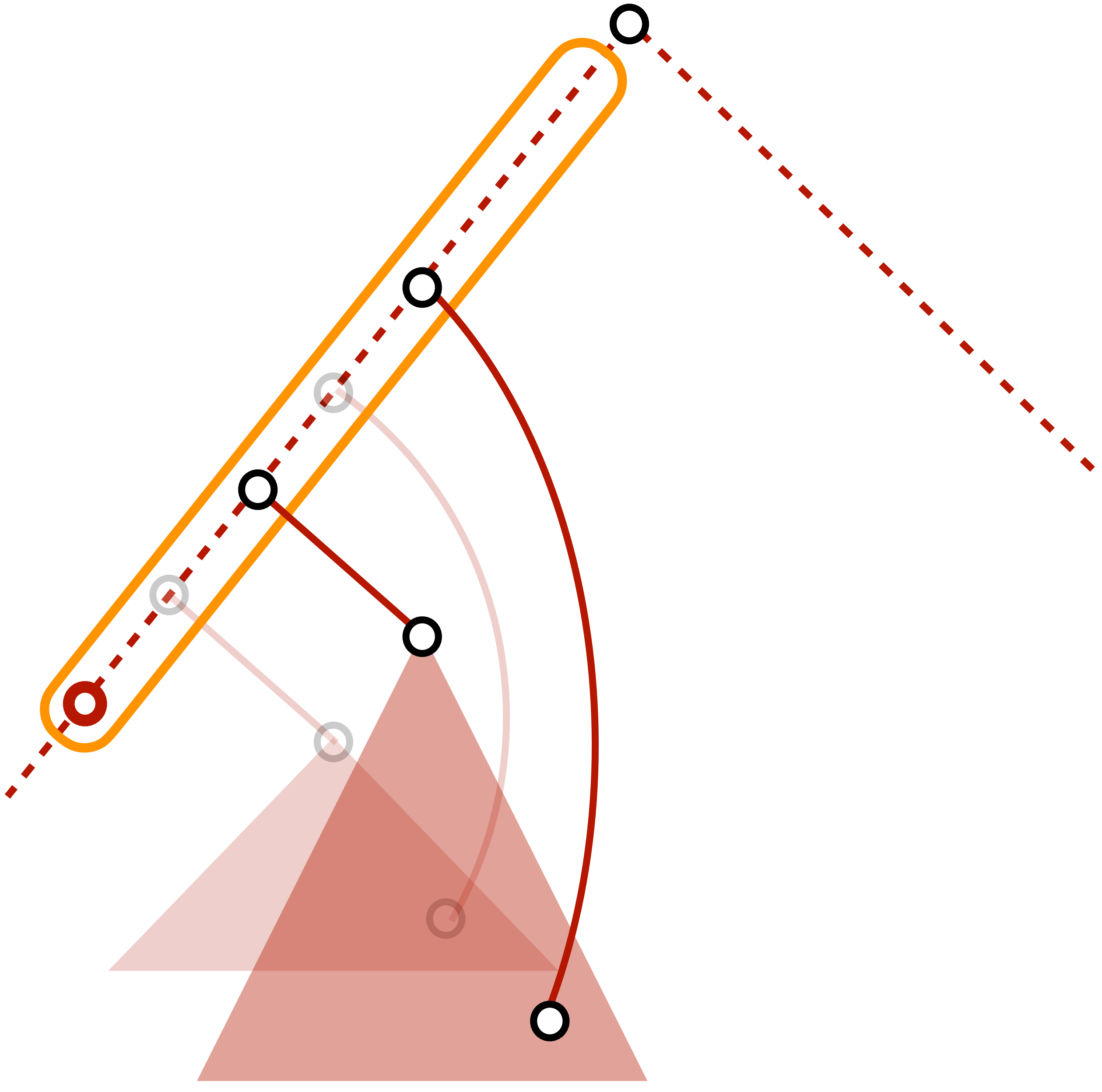
# Batch insertions [BCC+'16]

**Bottleneck:** finding highest ancestors on reverted paths



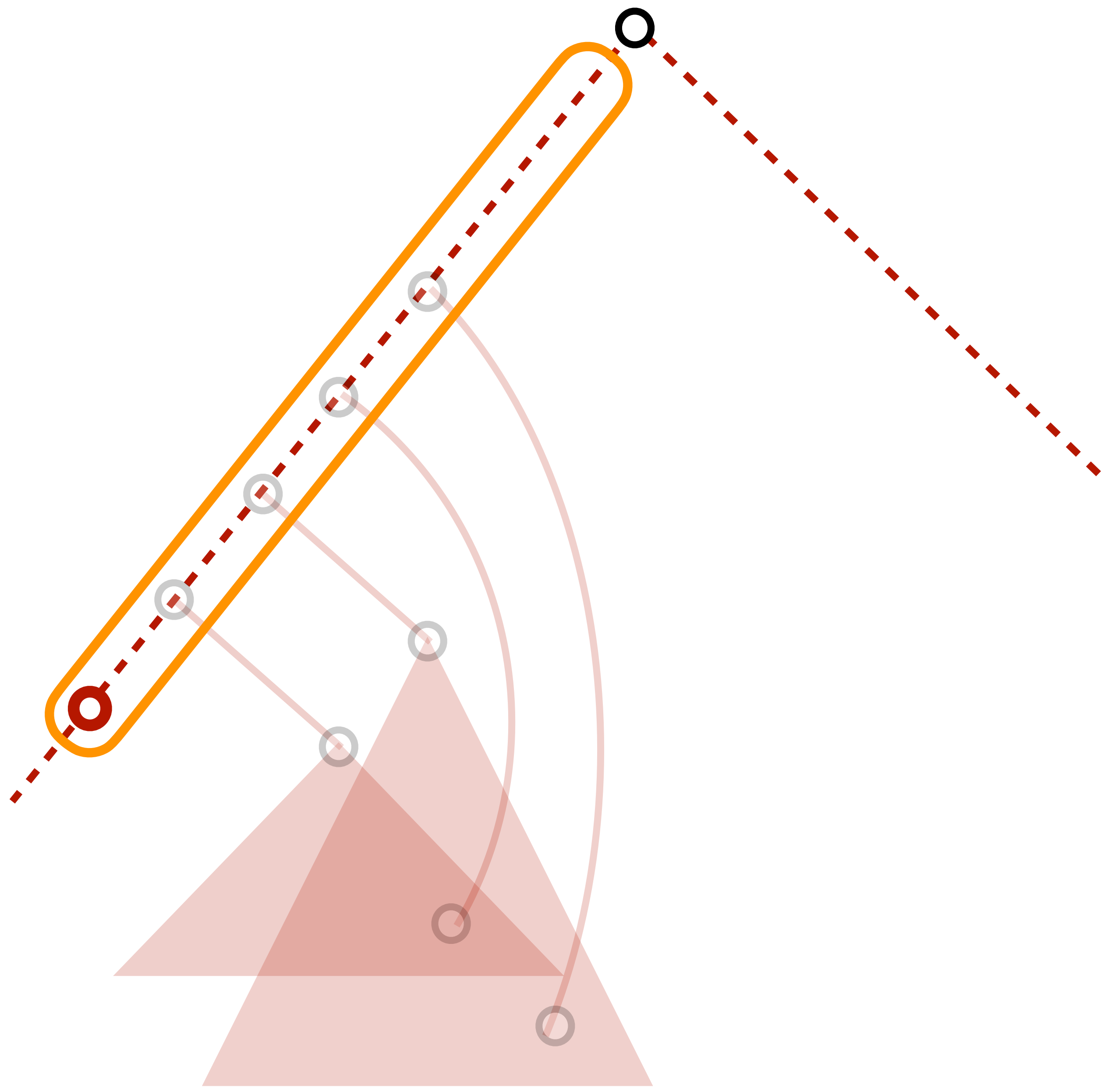
# Batch insertions [BCC+'16]

**Bottleneck:** finding highest ancestors on reverted paths



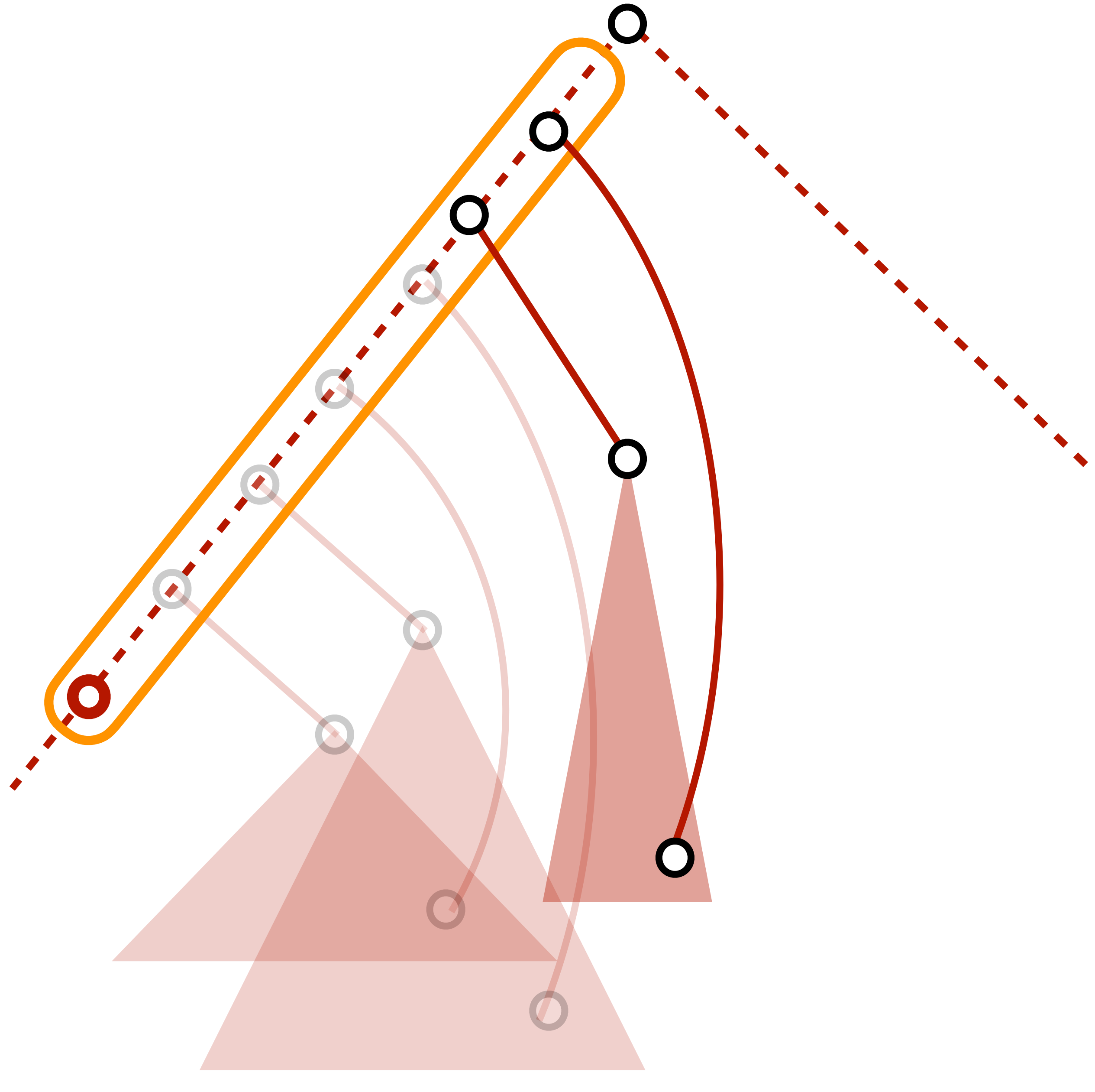
# Batch insertions [BCC+'16]

**Bottleneck:** finding highest ancestors on reverted paths



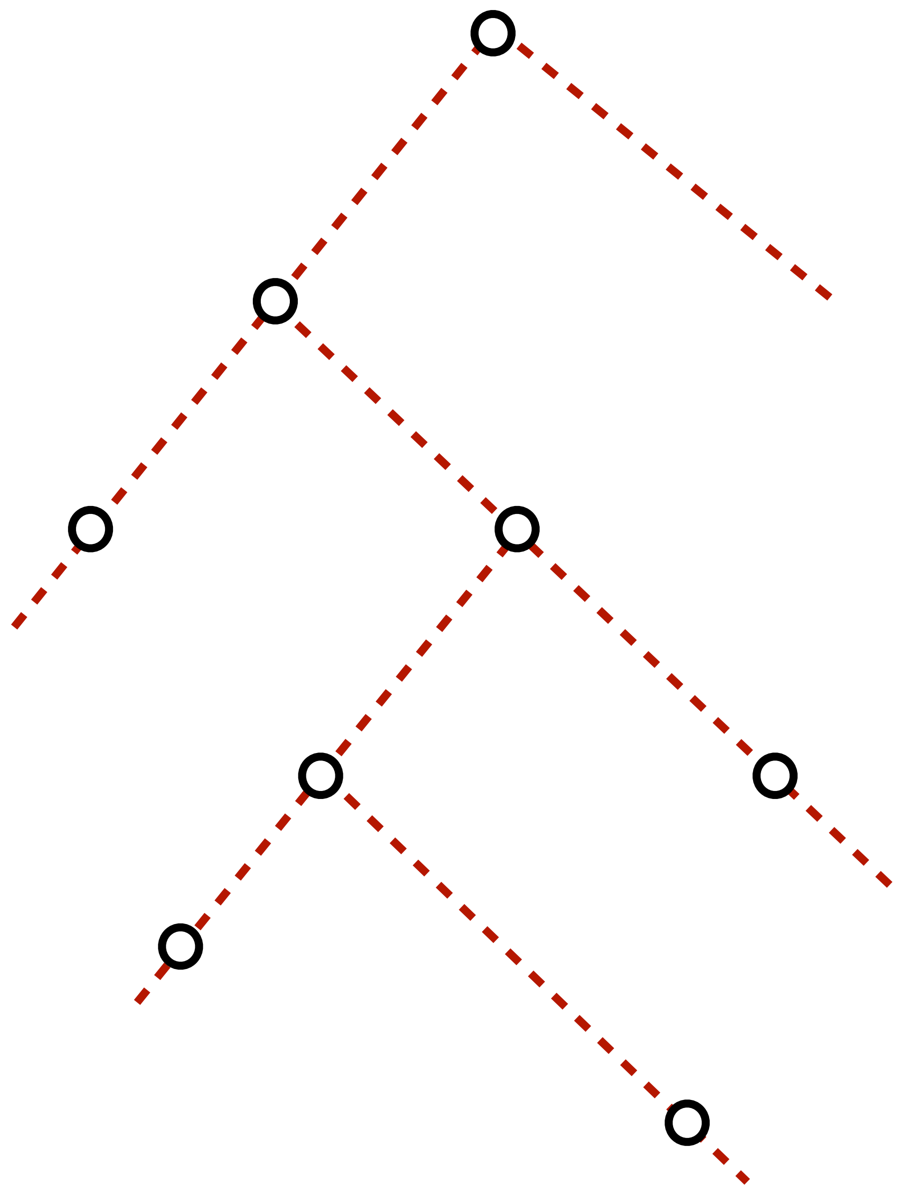
# Batch insertions [BCC+'16]

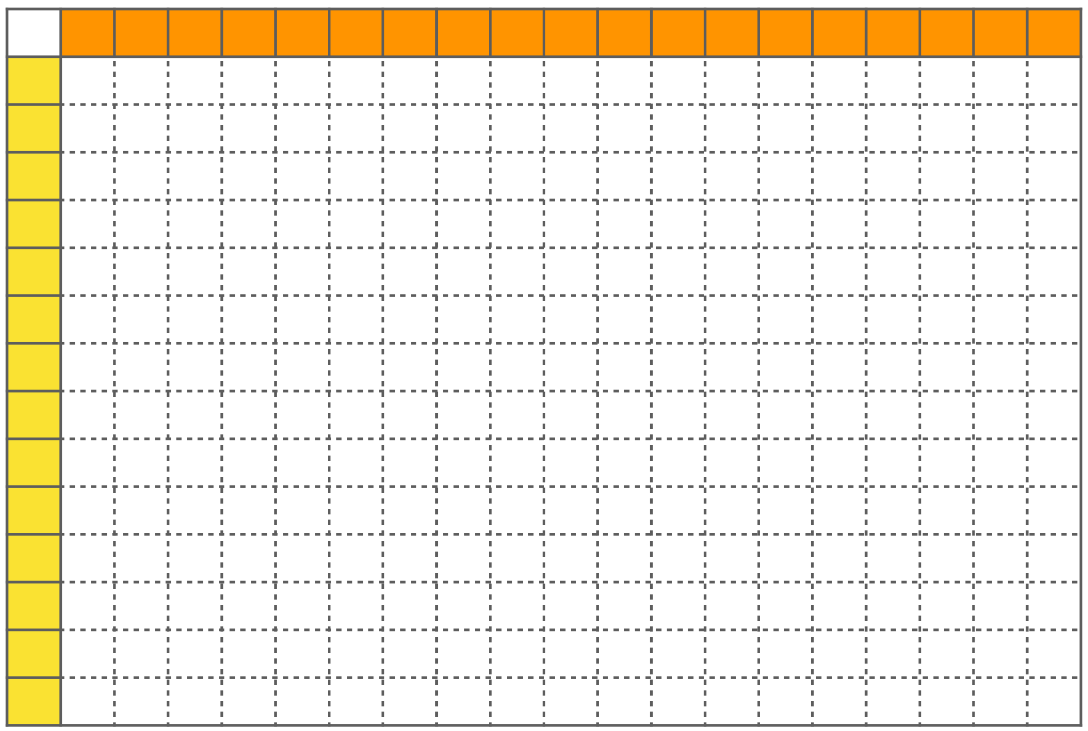
**Bottleneck:** finding highest ancestors on reverted paths



# Finding highest ancestors

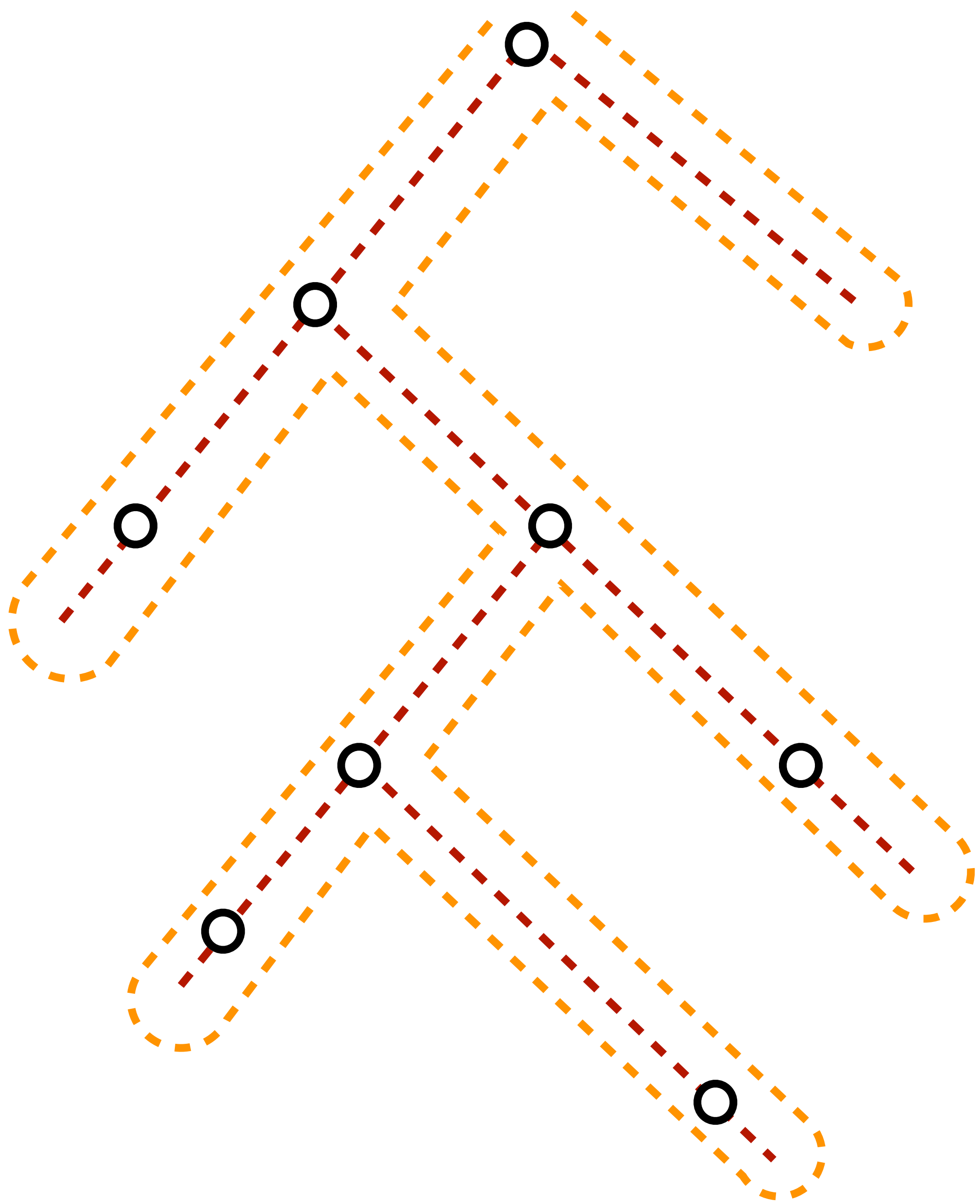
**Tool I:** 2D-range query

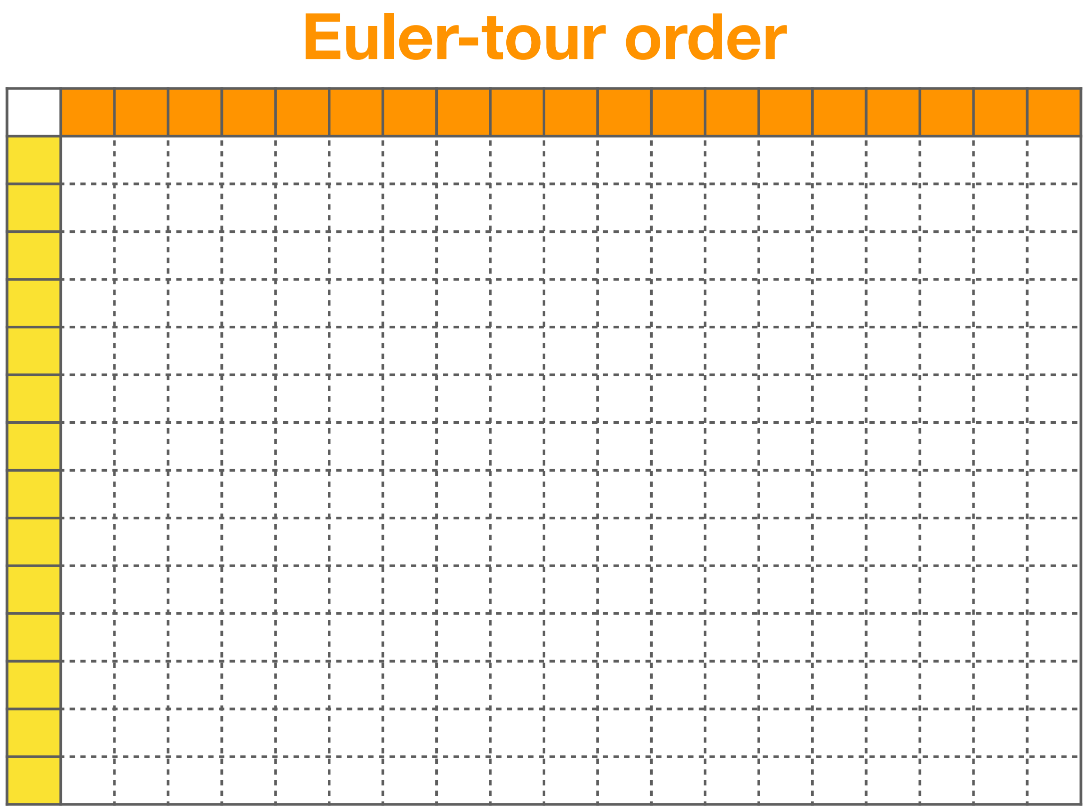




# Finding highest ancestors

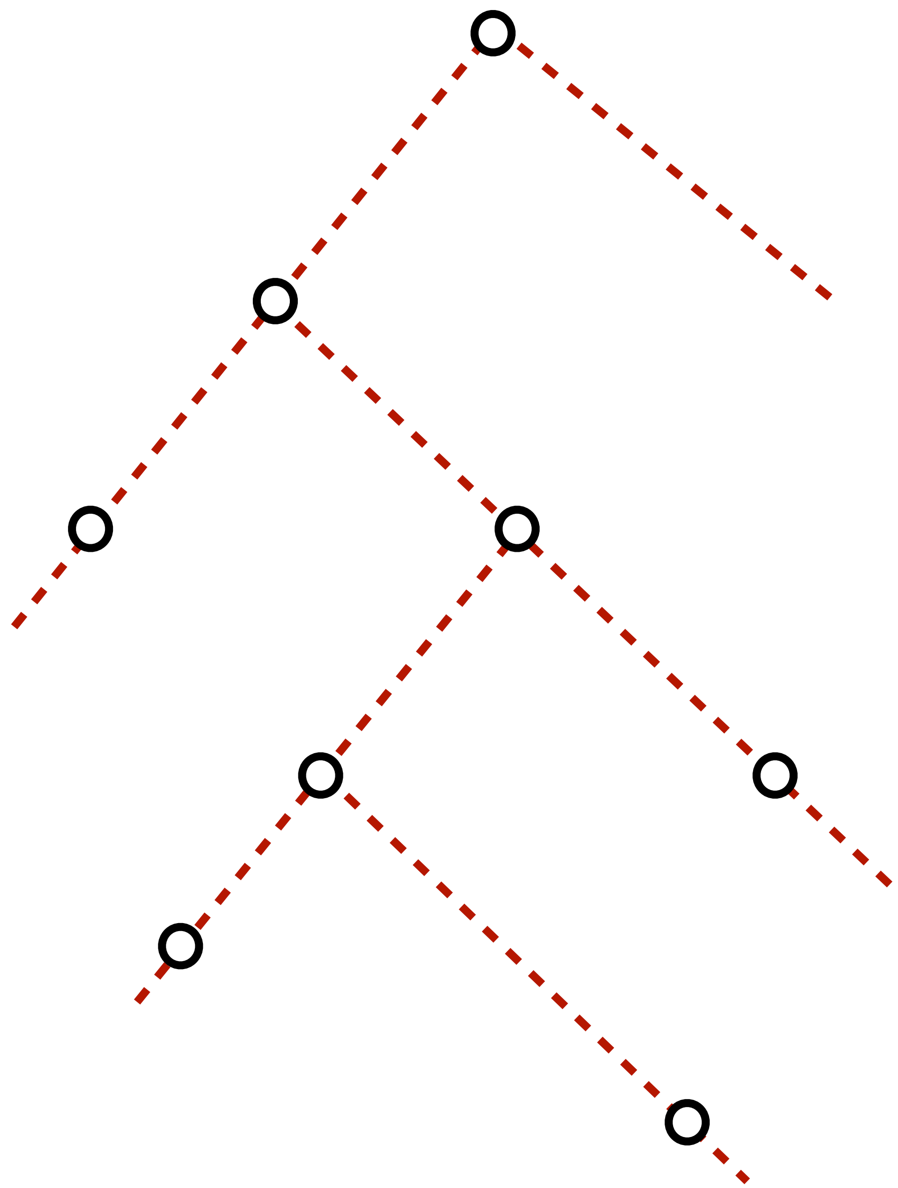
**Tool I:** 2D-range query

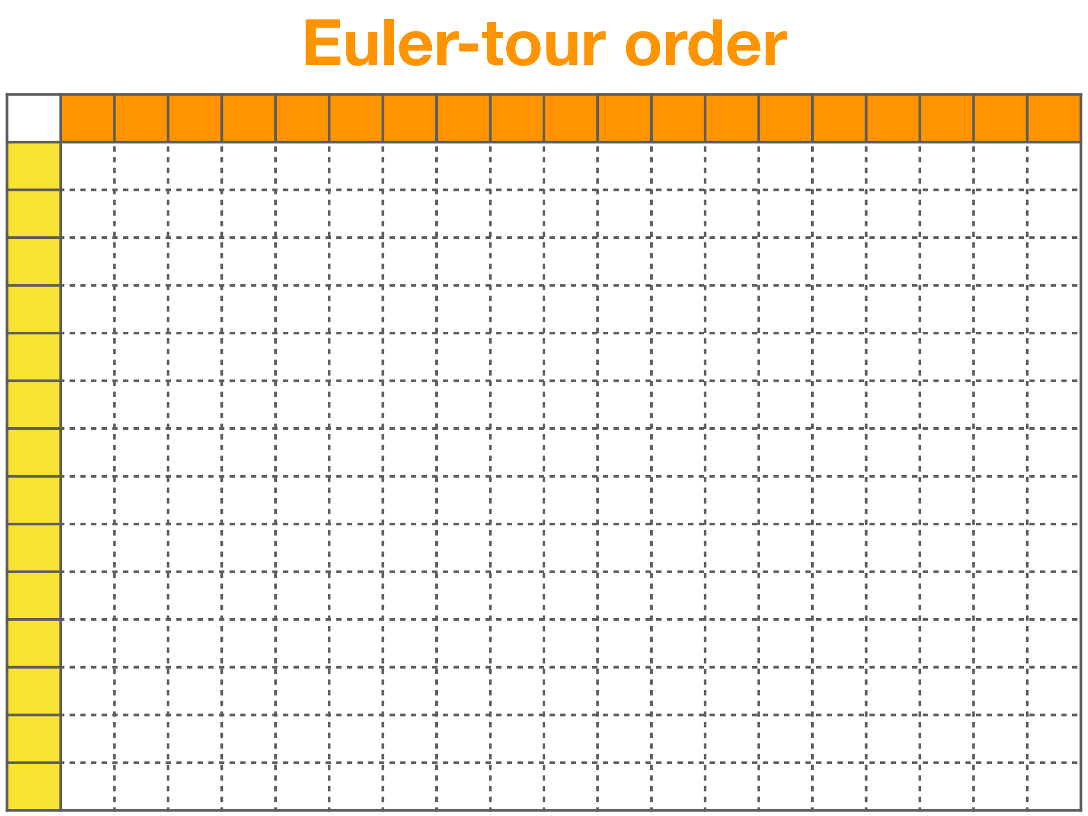




# Finding highest ancestors

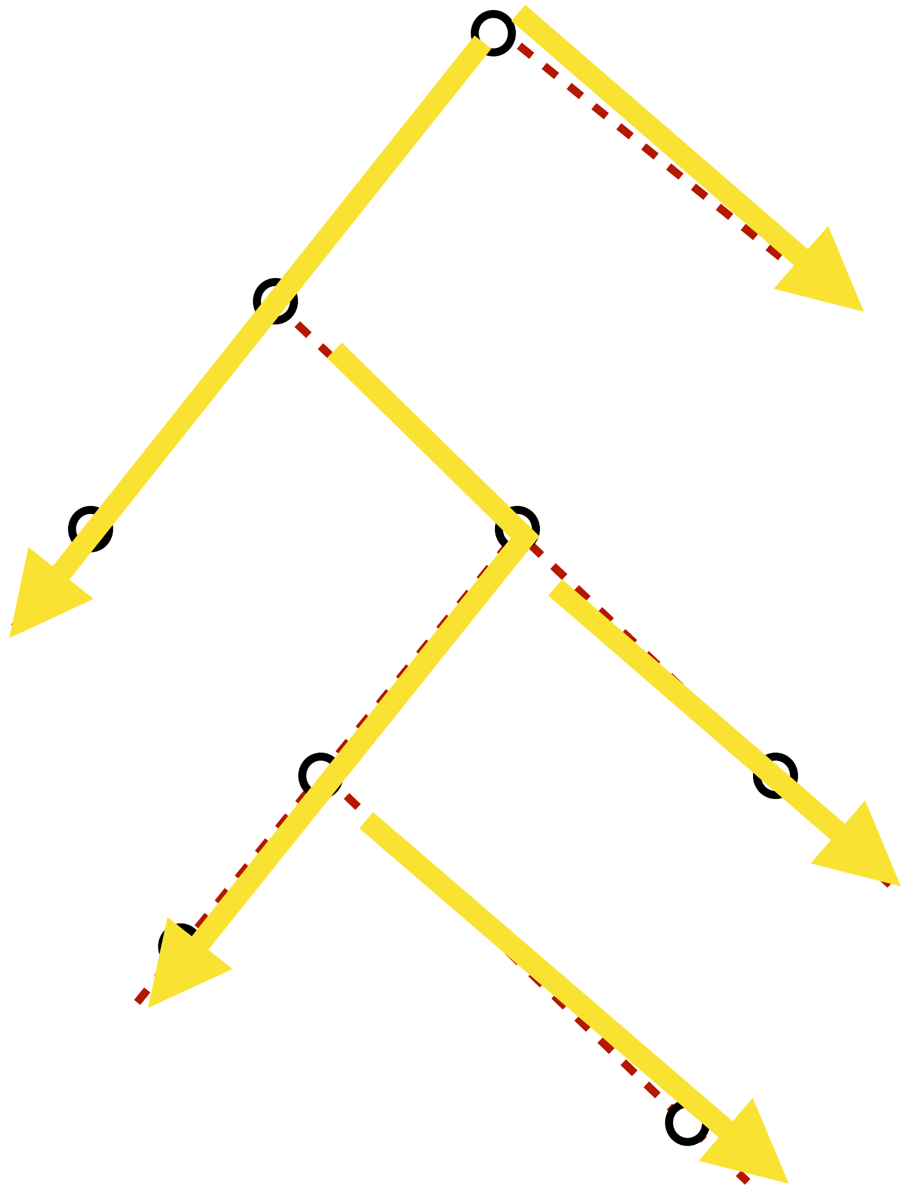
**Tool I:** 2D-range query

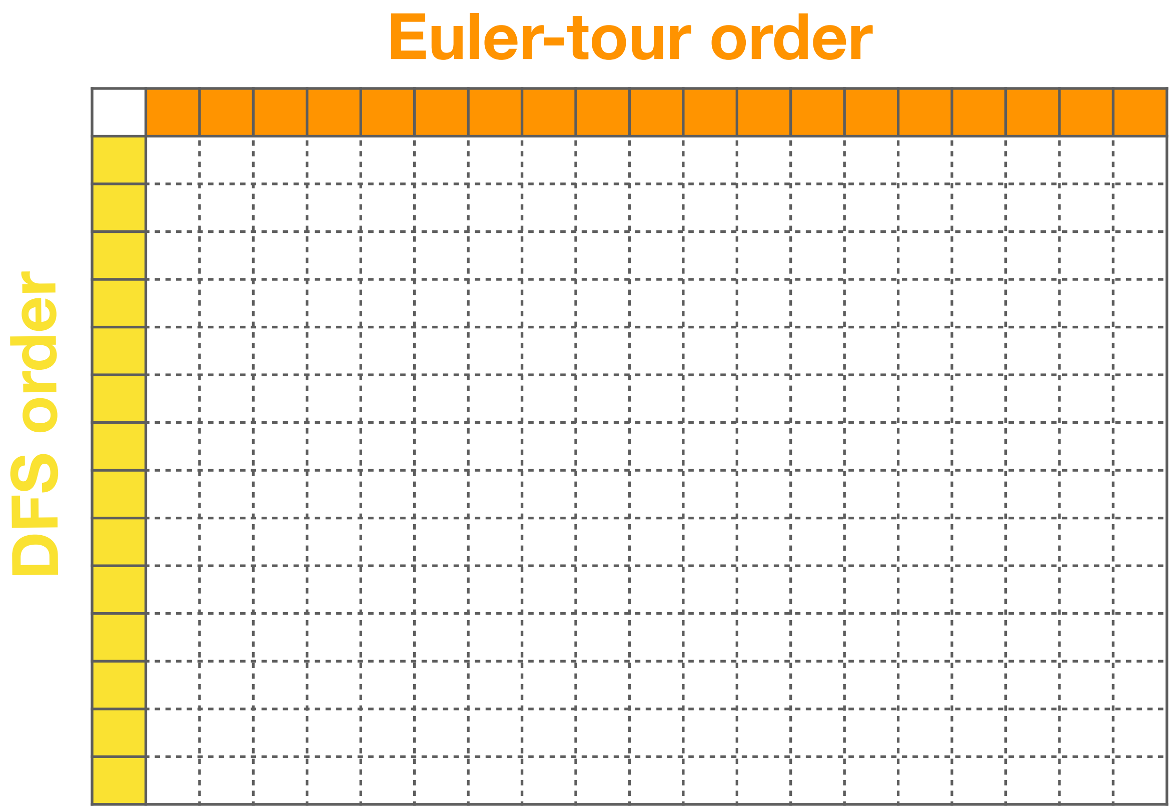




# Finding highest ancestors

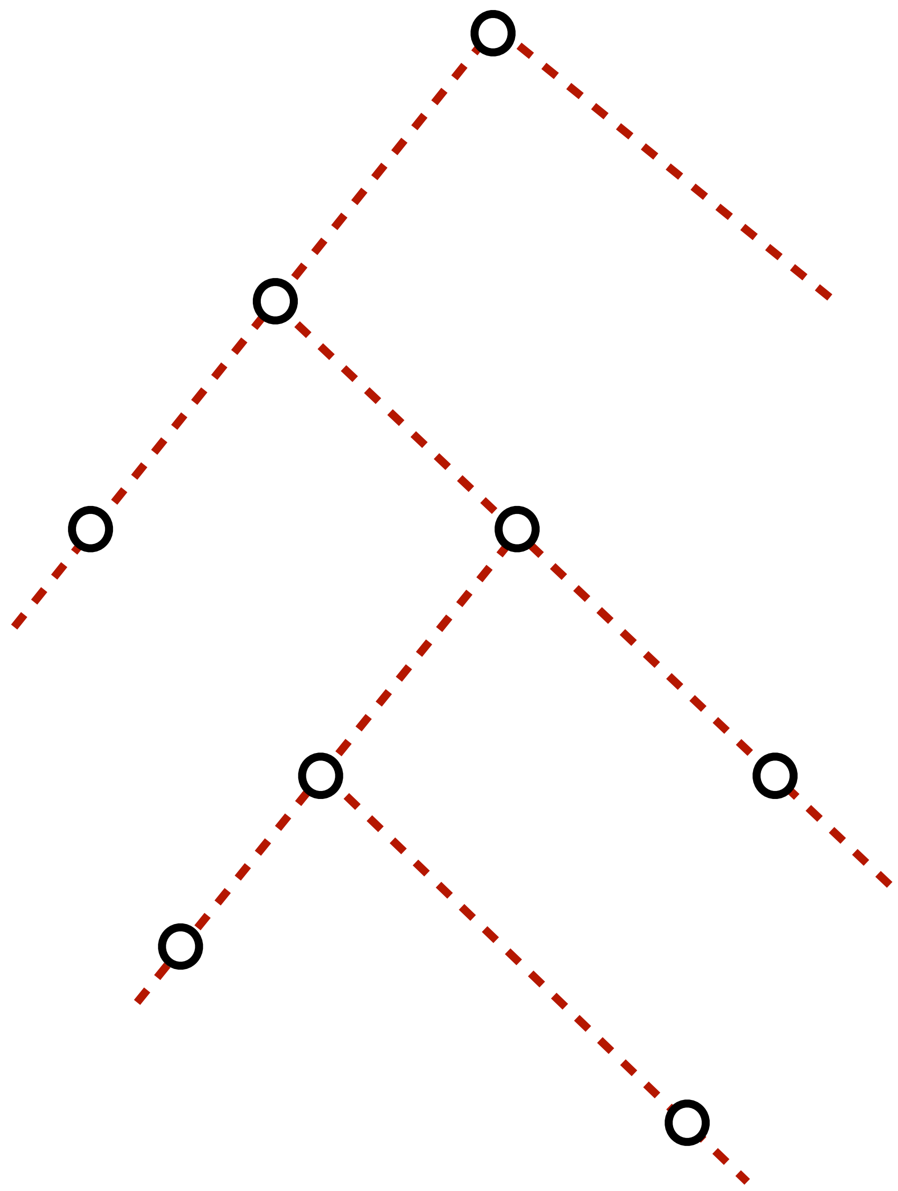
**Tool I:** 2D-range query

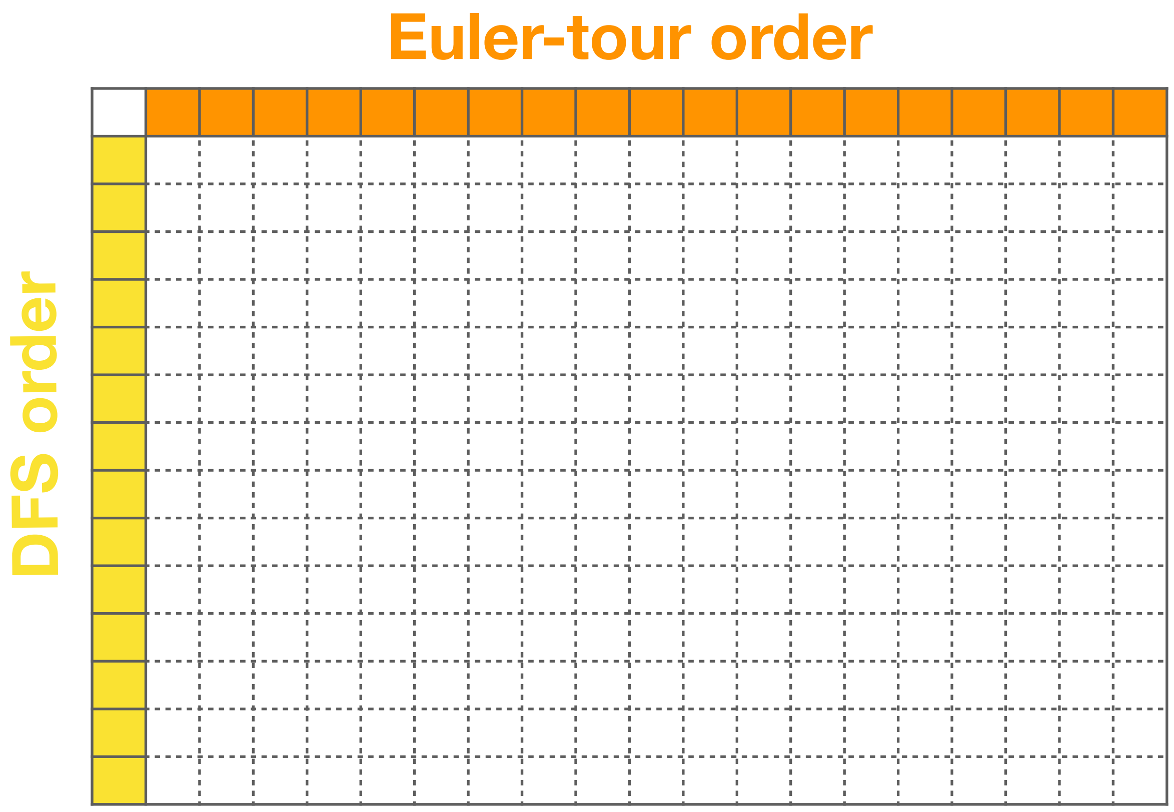




# Finding highest ancestors

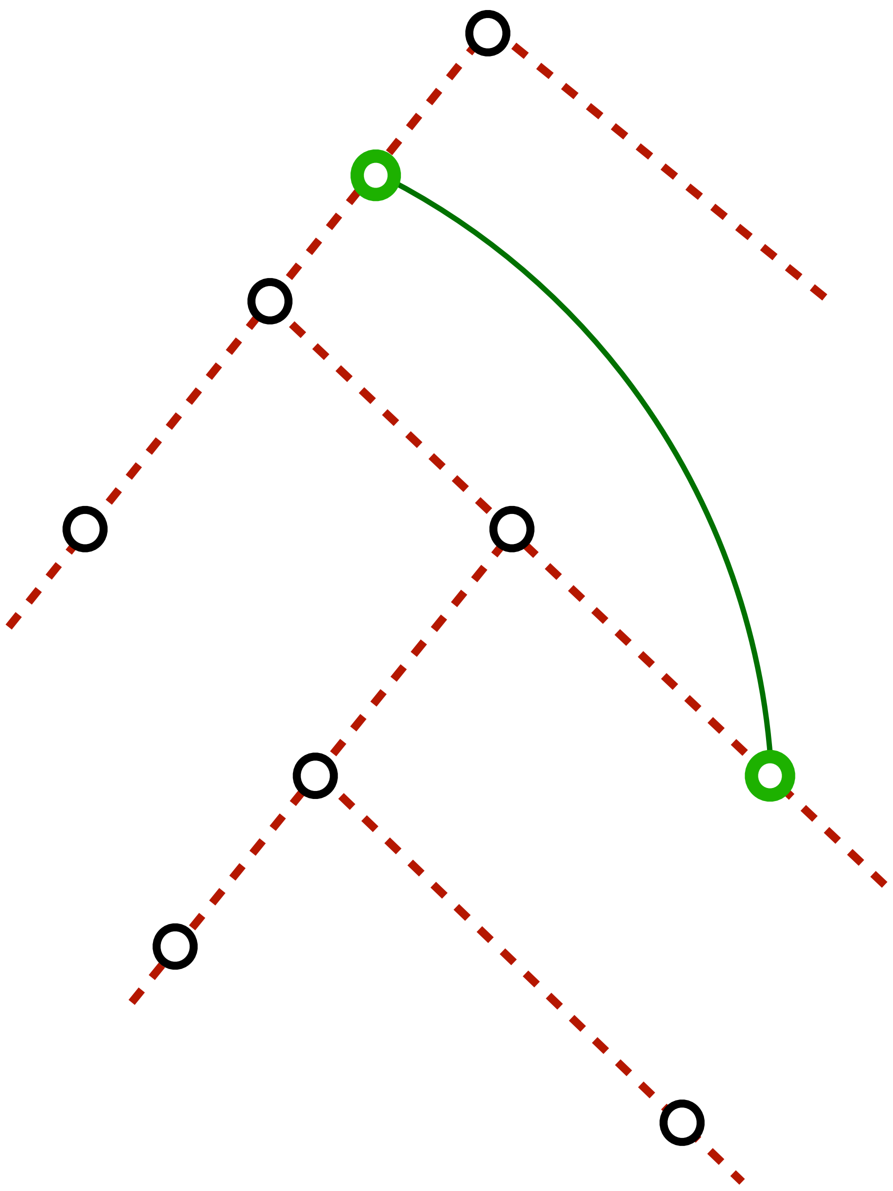
**Tool I:** 2D-range query

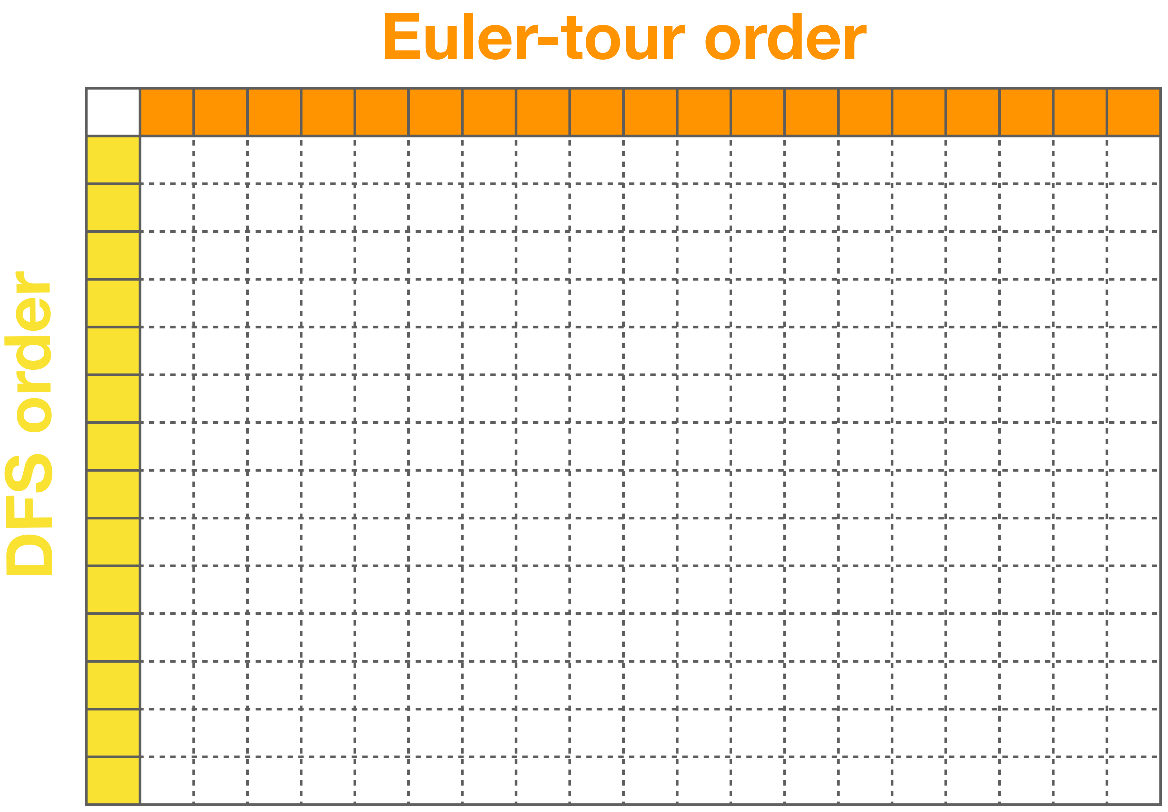




# Finding highest ancestors

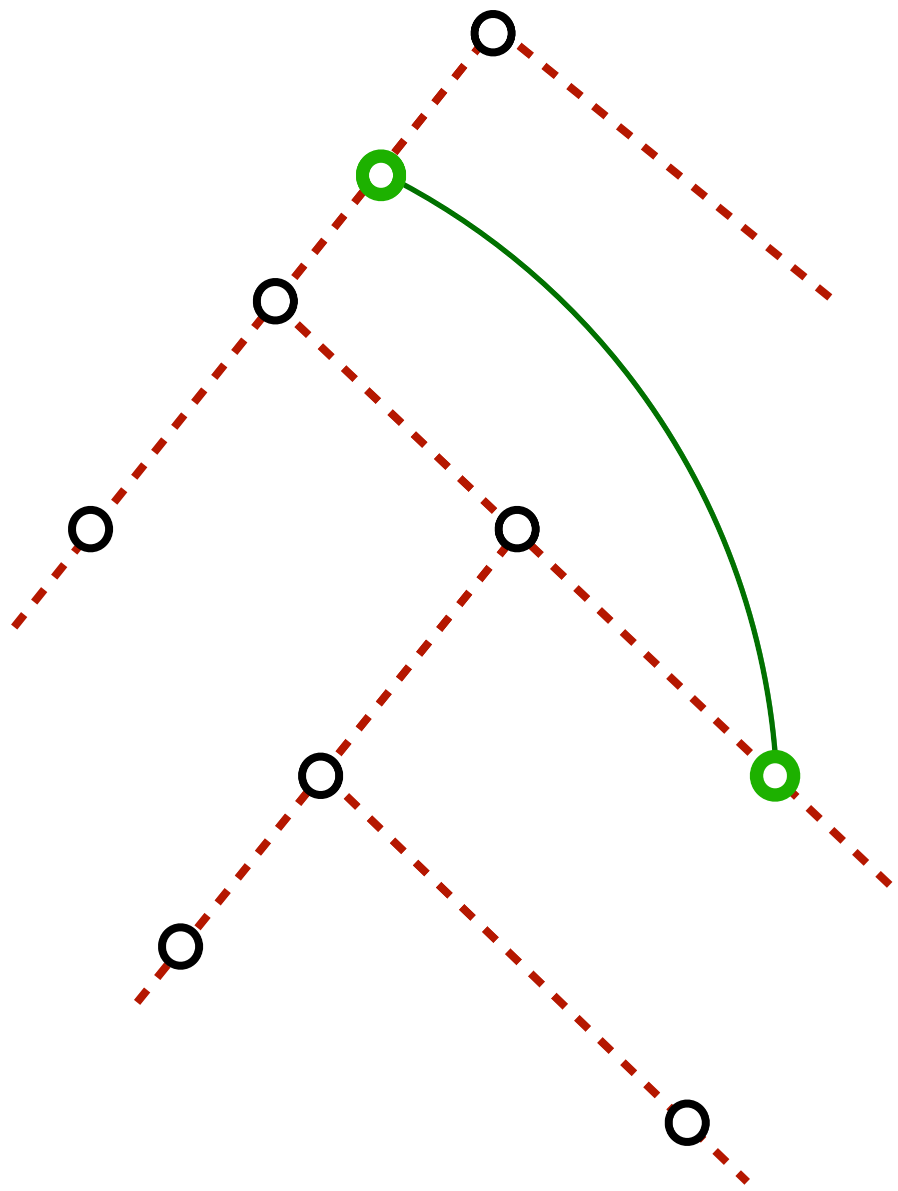
**Tool I:** 2D-range query

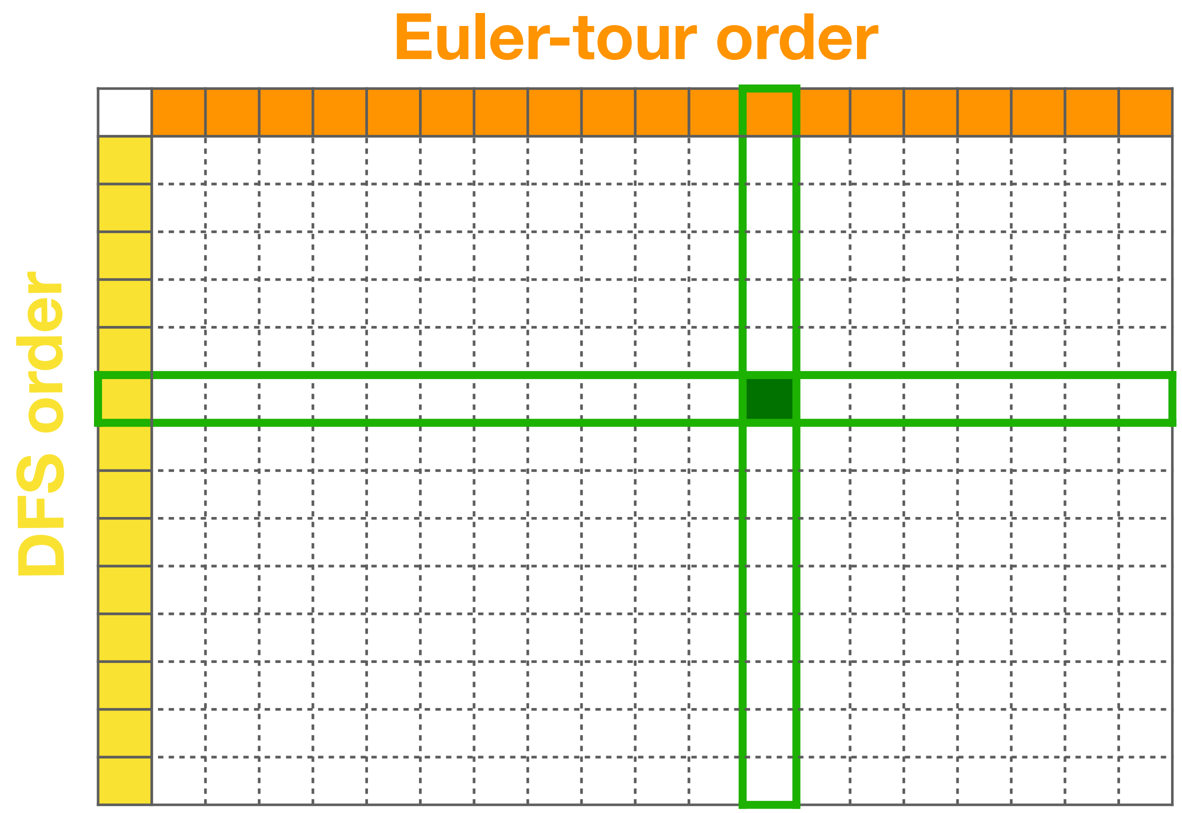




# Finding highest ancestors

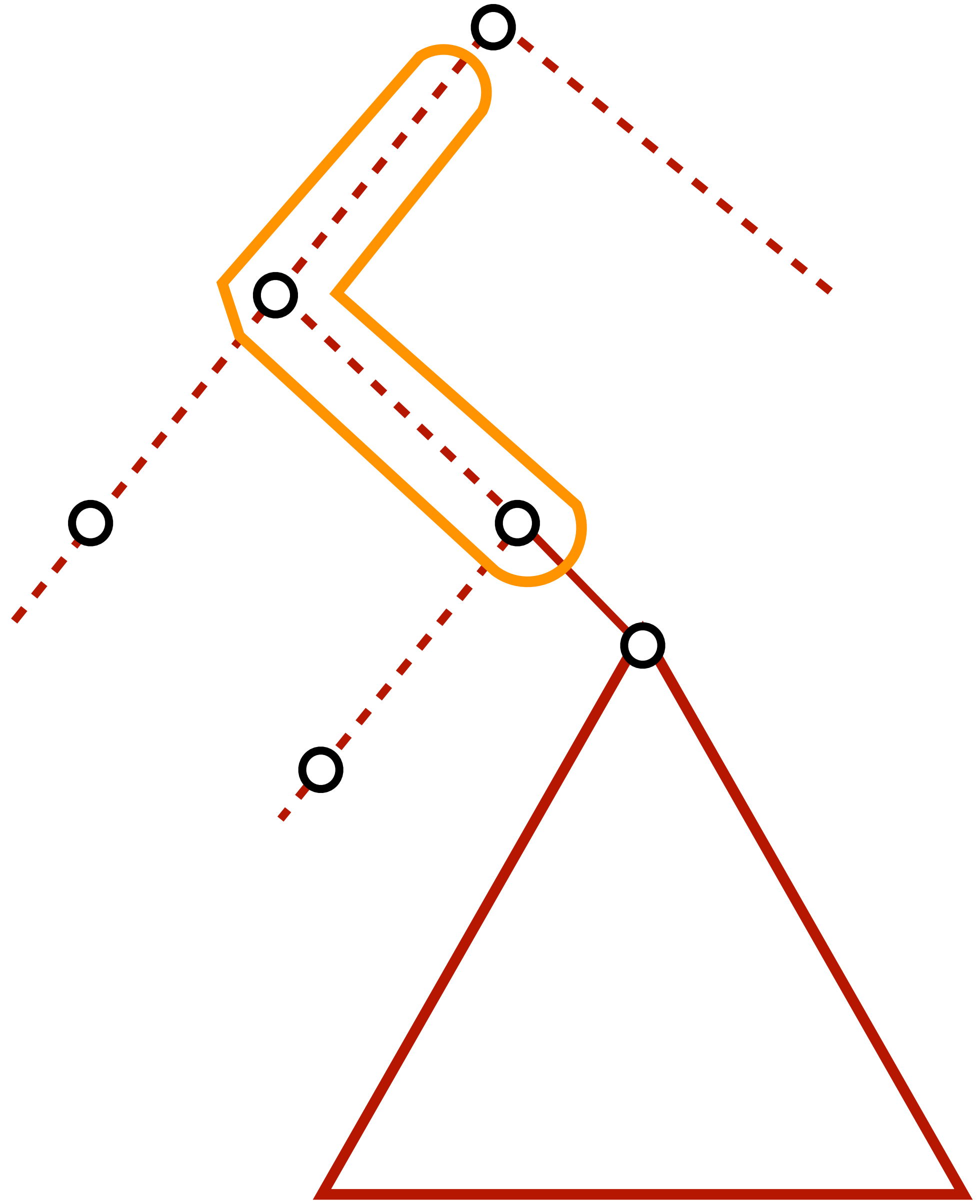
**Tool I:** 2D-range query

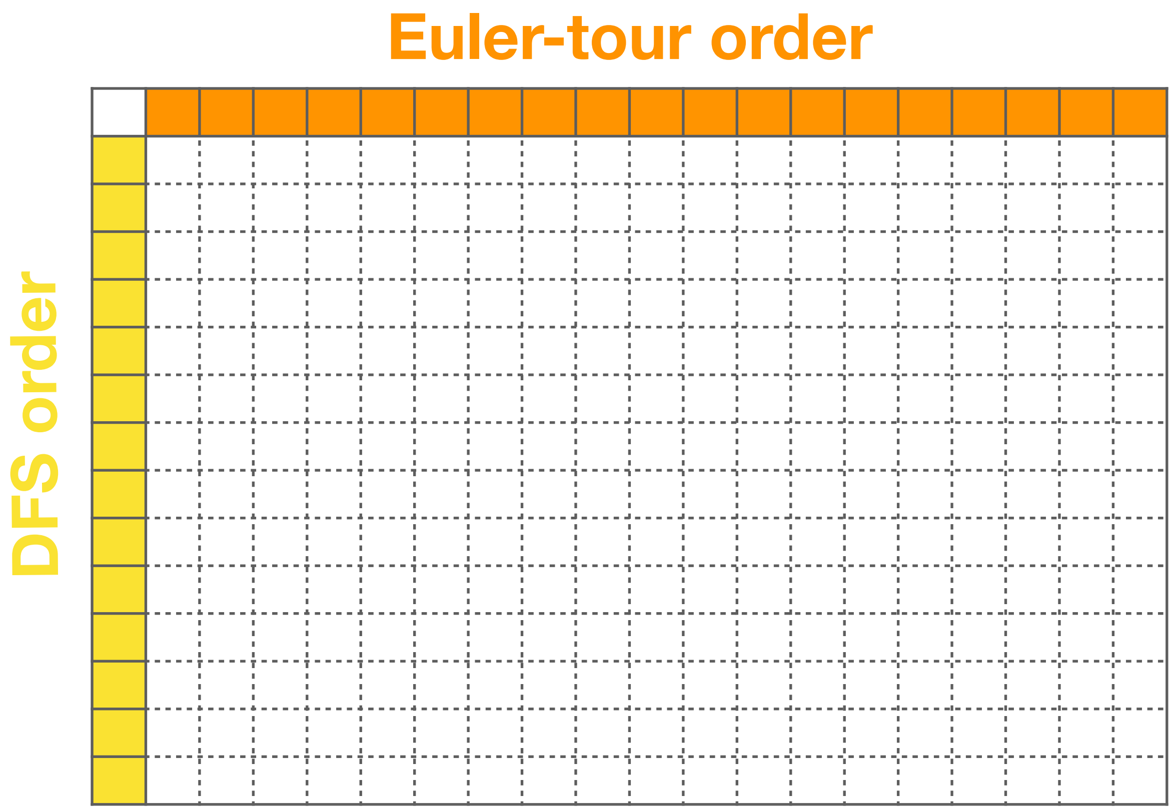




# Finding highest ancestors

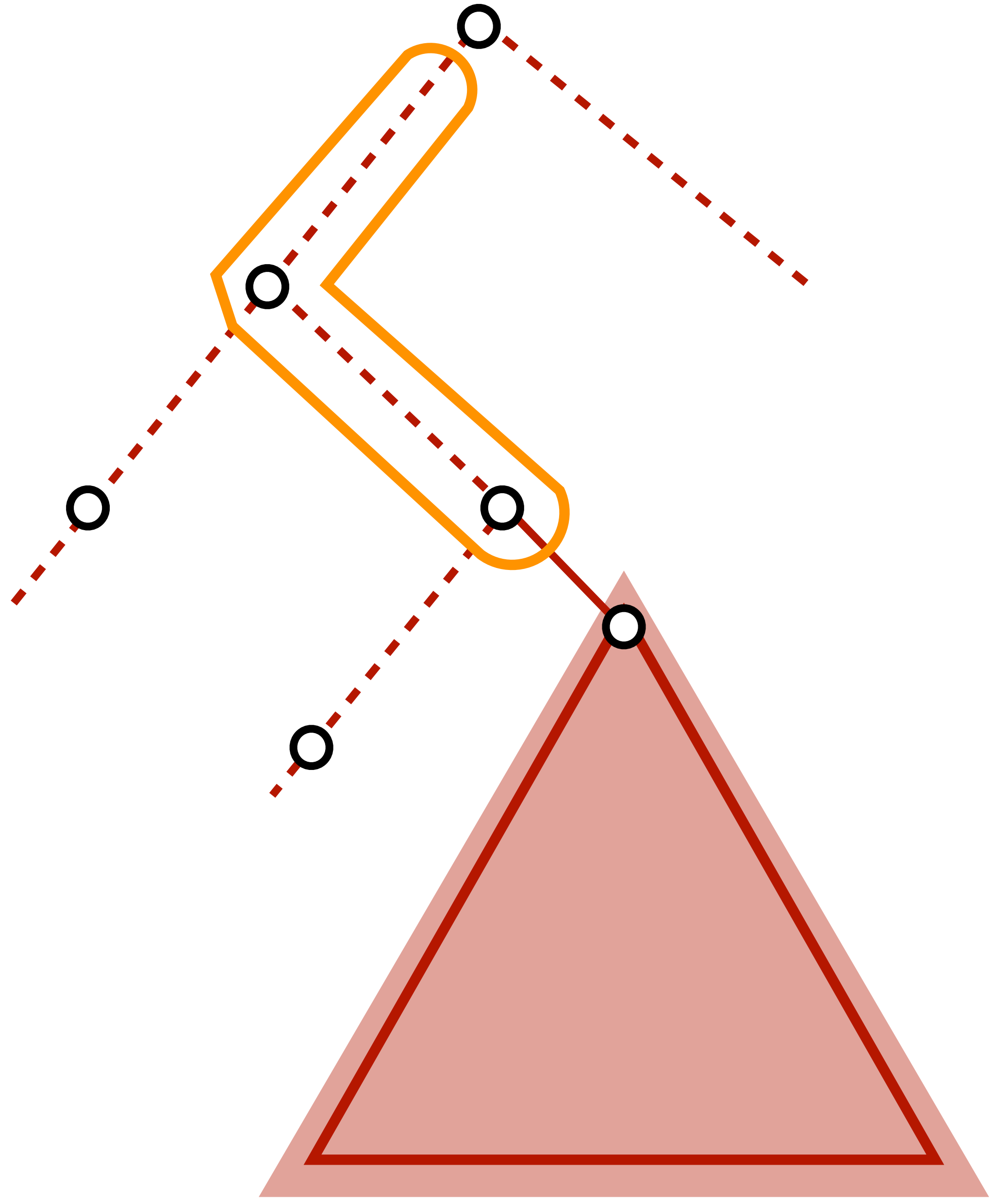
**Tool I:** 2D-range query

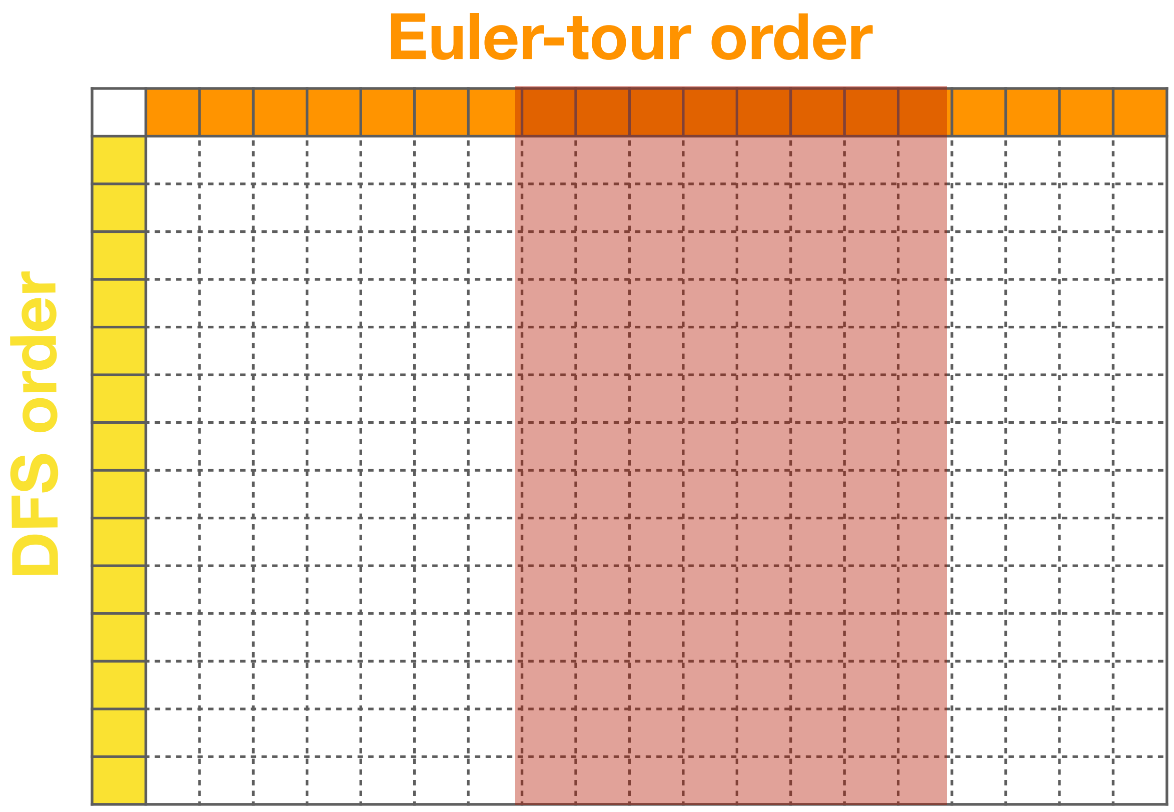




# Finding highest ancestors

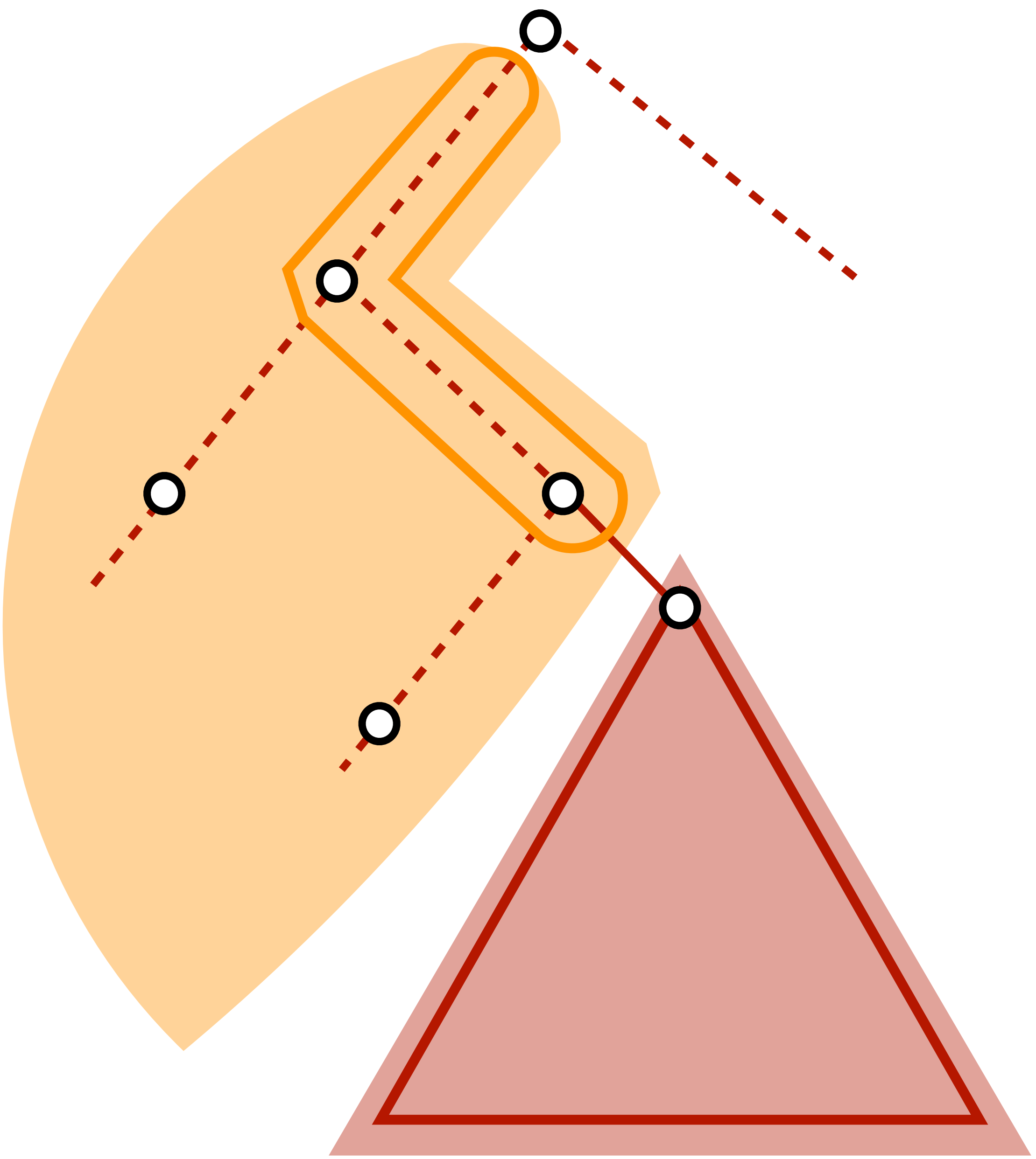
**Tool I:** 2D-range query

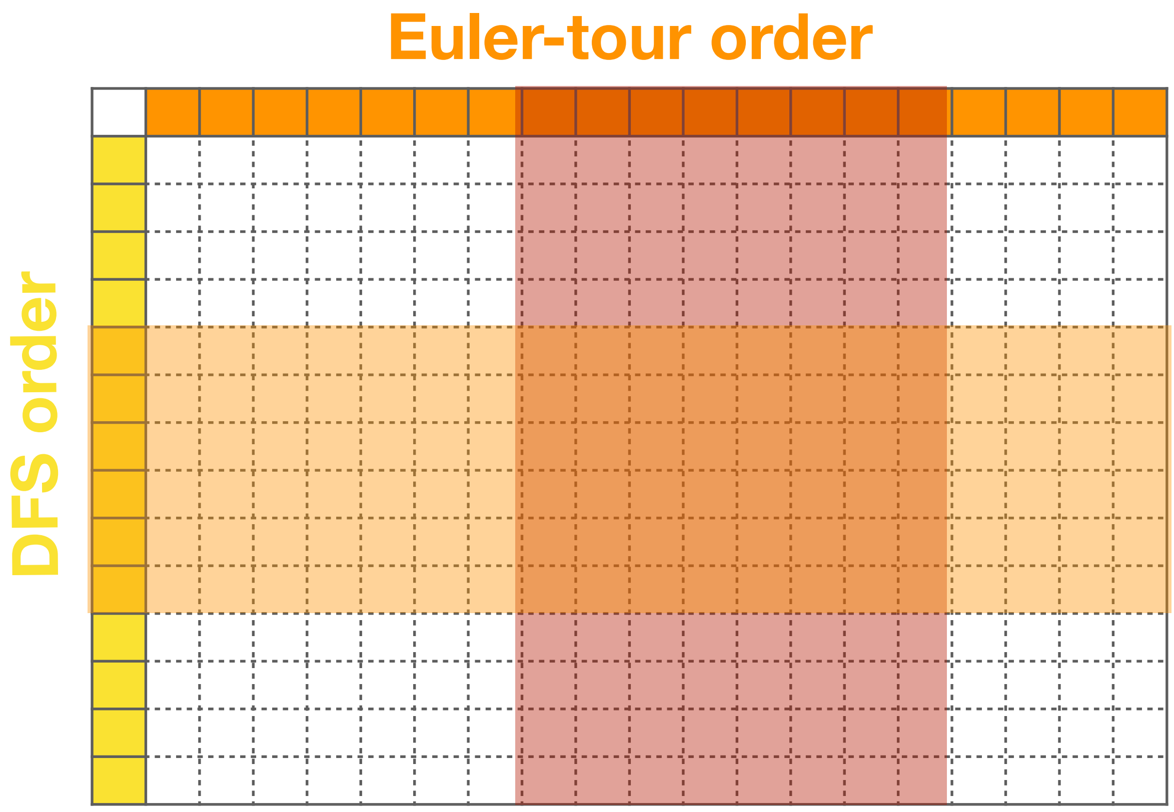




# Finding highest ancestors

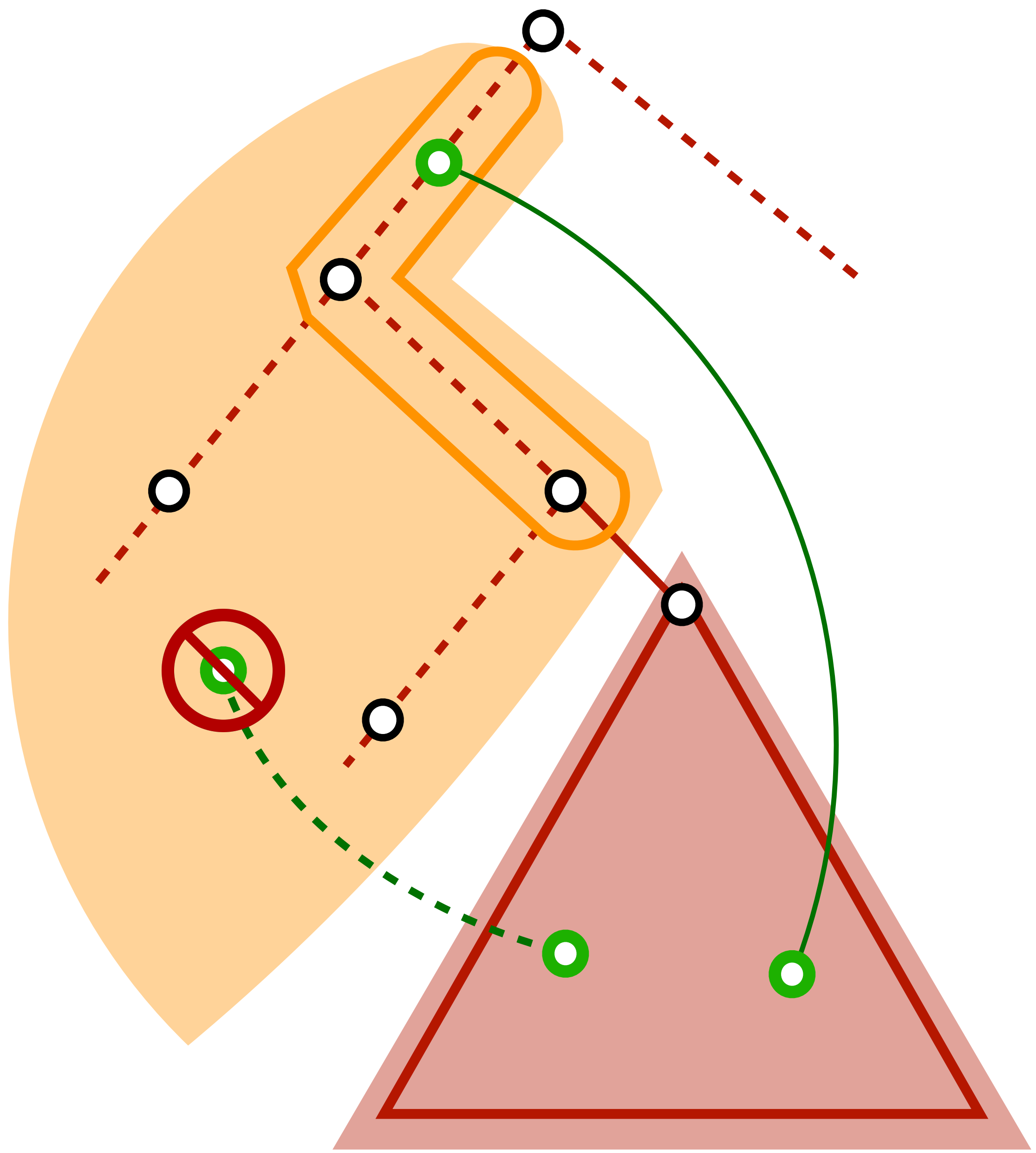
**Tool I:** 2D-range query

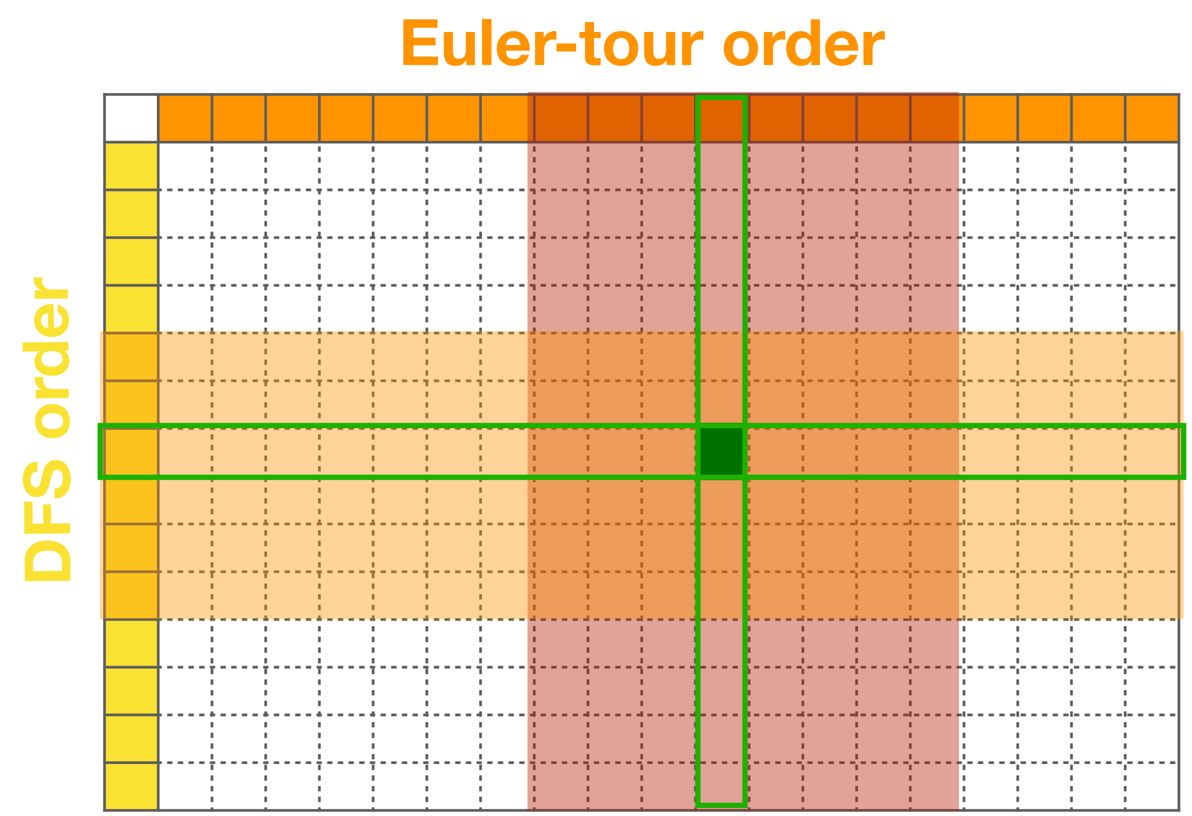




# Finding highest ancestors

**Tool I:** 2D-range query

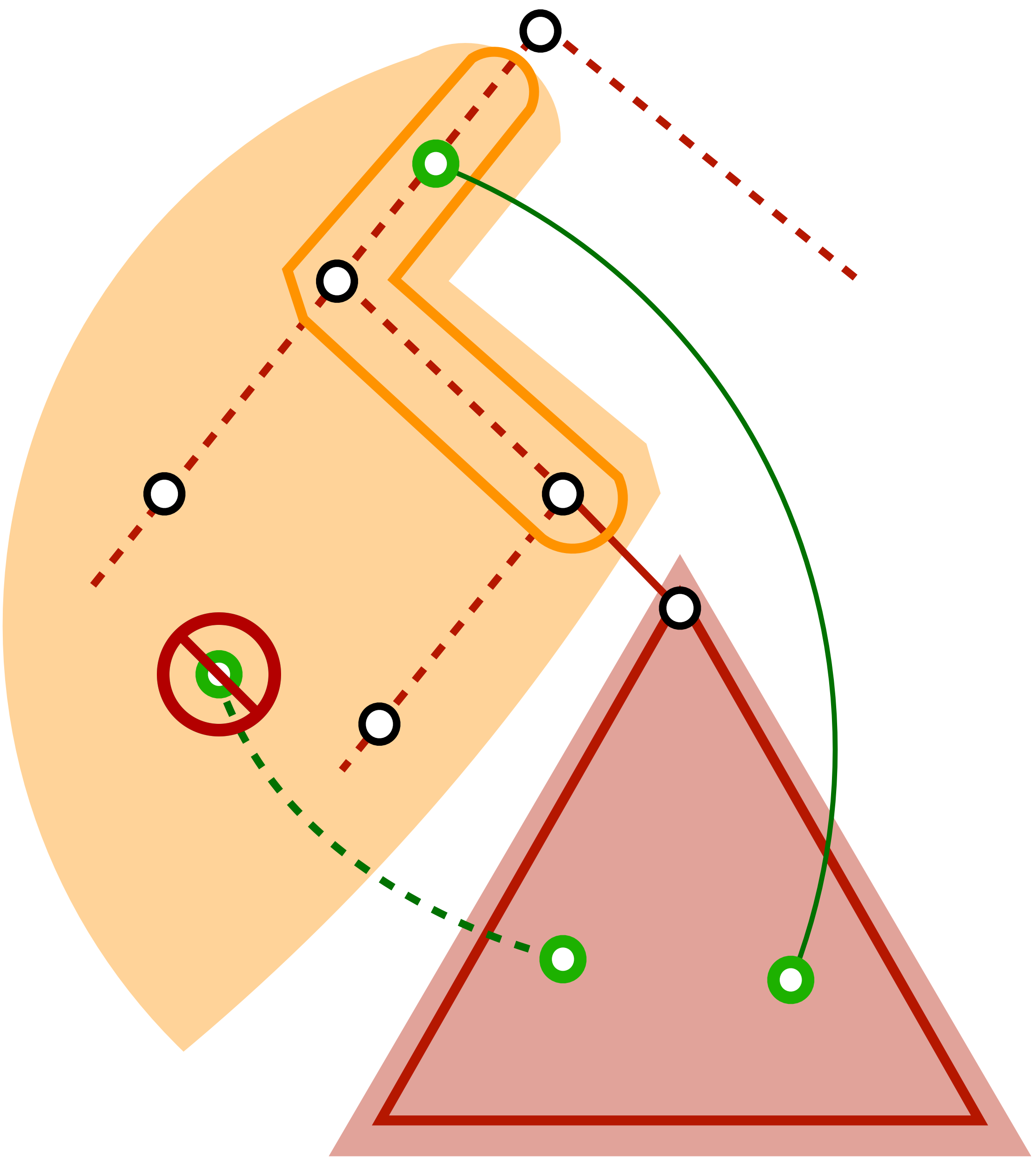


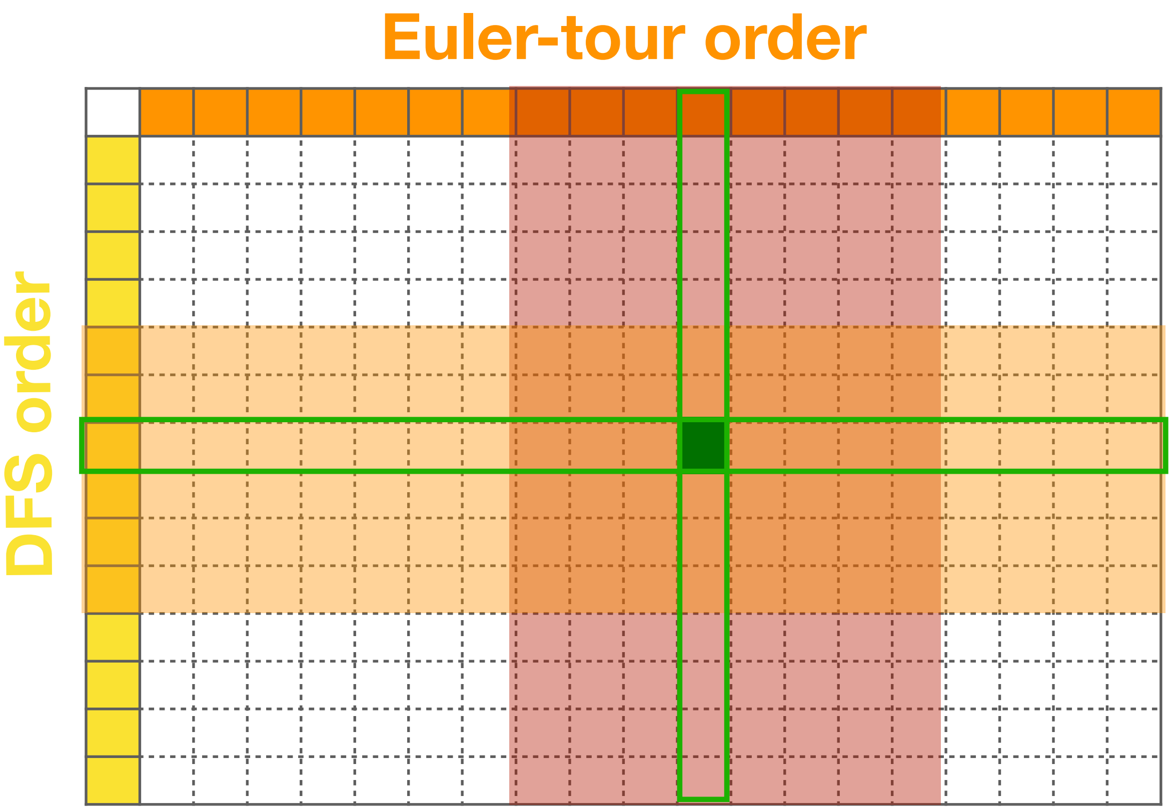



**2D-range minimum takes $O(\log n)$ time**

# Finding highest ancestors

**Tool I:** 2D-range query






**2D-range minimum takes $O(\log n)$ time**

**Total time 'd be** $O(k + n \log n)$

# Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

# Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

**Lemma:** [CG'86]

Given *k* sorted integer arrays of total size *m*

- **Input:** integer *x*
- **Output:** successors of *x* in each array, in time $O(k + \log m)$

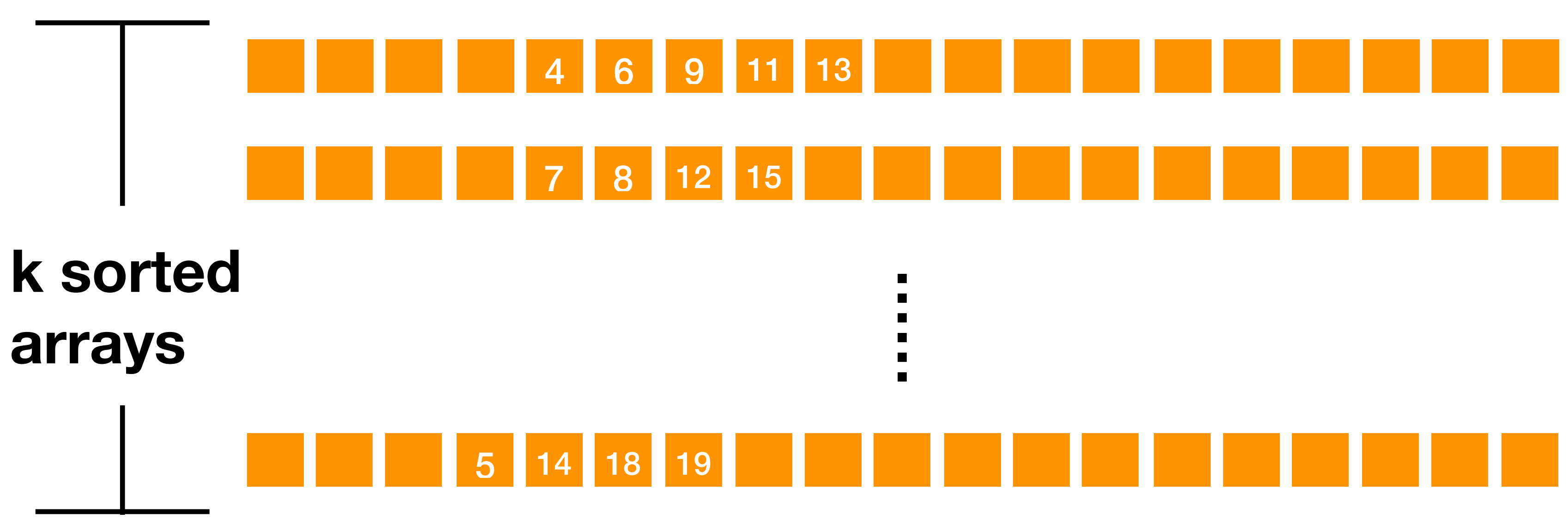
# Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

<u>**Lemma:**</u> [CG'86]

Given *k* sorted integer arrays of total size *m*

- **Input:** integer *x*
- **Output:** successors of *x* in each array, in time $O(k + \log m)$



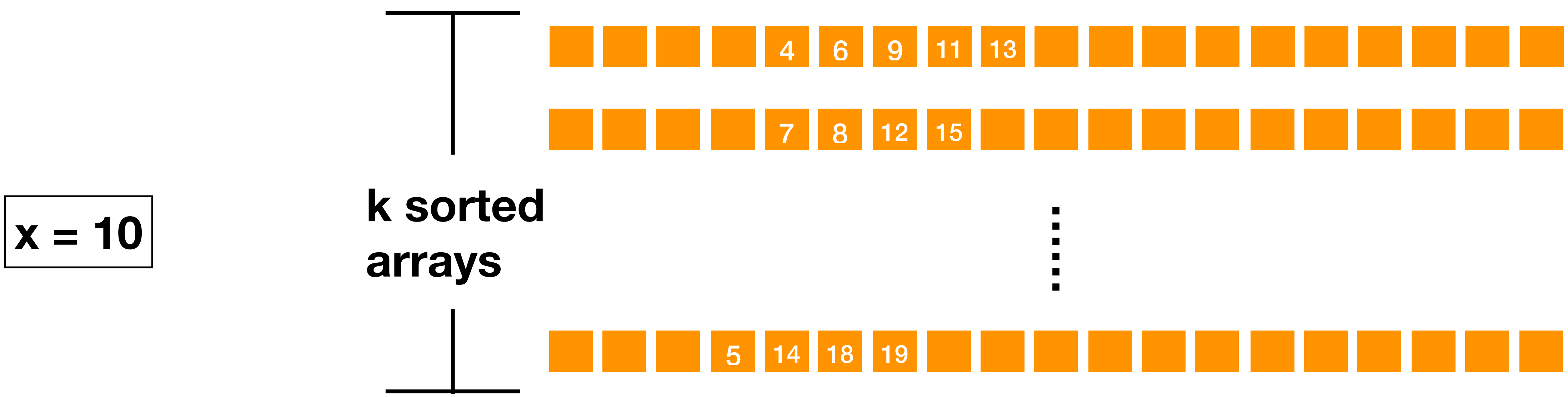
# Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

<u>**Lemma:**</u> [CG'86]

Given *k* sorted integer arrays of total size *m*

- **Input:** integer *x*
- **Output:** successors of *x* in each array, in time $O(k + \log m)$



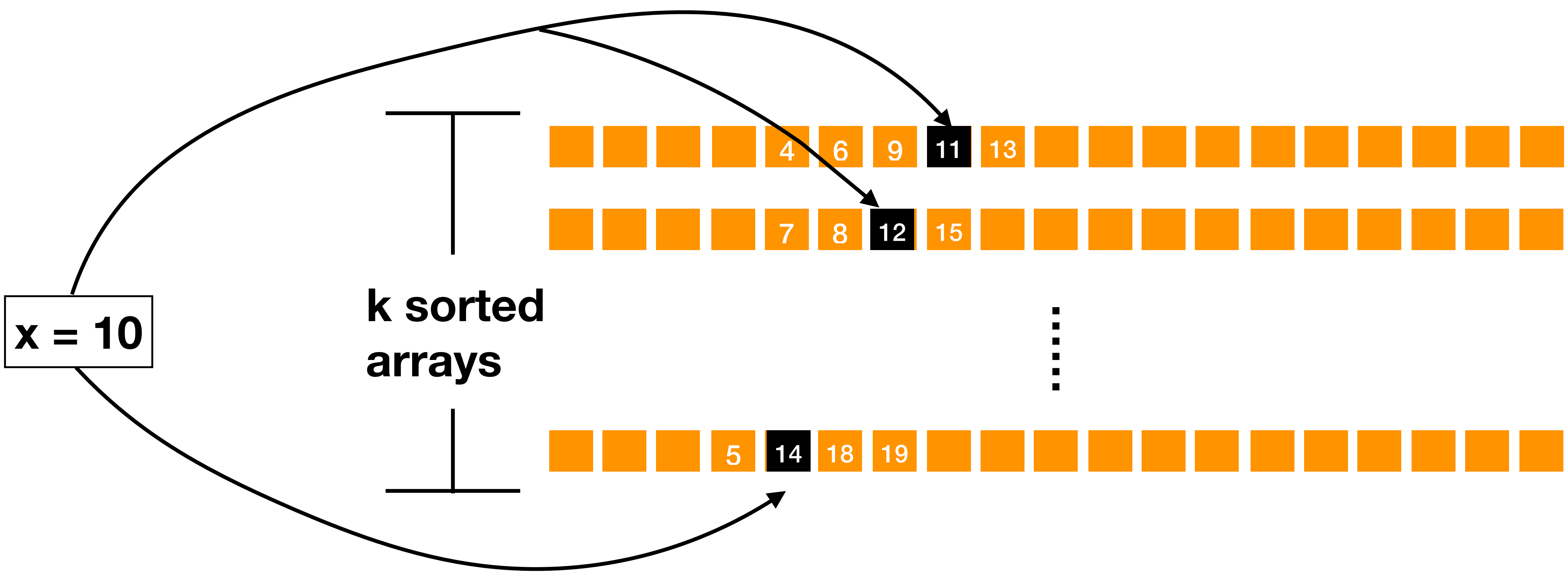
# Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

**Lemma:** [CG'86]

Given *k* sorted integer arrays of total size *m*

- **Input:** integer *x*
- **Output:** successors of *x* in each array, in time $O(k + \log m)$



# Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

<u>**Lemma:**</u> [DZ'17]

Given a tree *T* of size *n*,

- **Input:** integer *k*
- **Output:** remove $O(n/k)$ special vertices to partition *T* into subtrees of size at most *k*

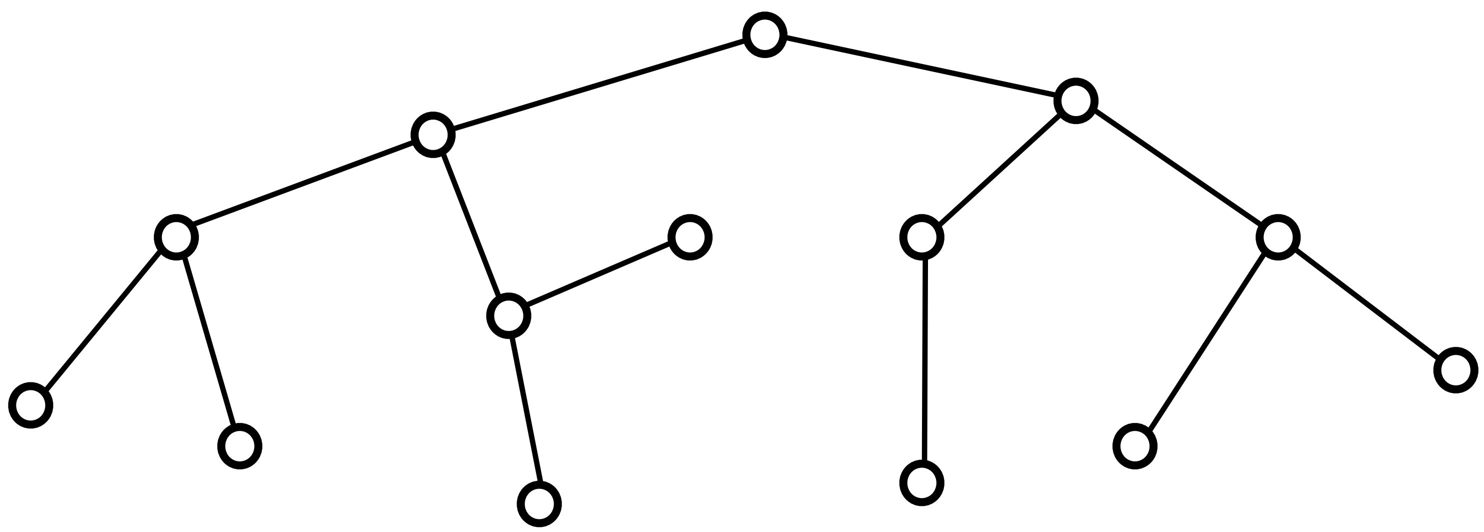
# Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

<u>**Lemma:**</u> [DZ'17]

Given a tree *T* of size *n,*

- **Input:** integer *k*
- **Output:** remove $O(n/k)$ special vertices to partition *T* into subtrees of size at most *k*



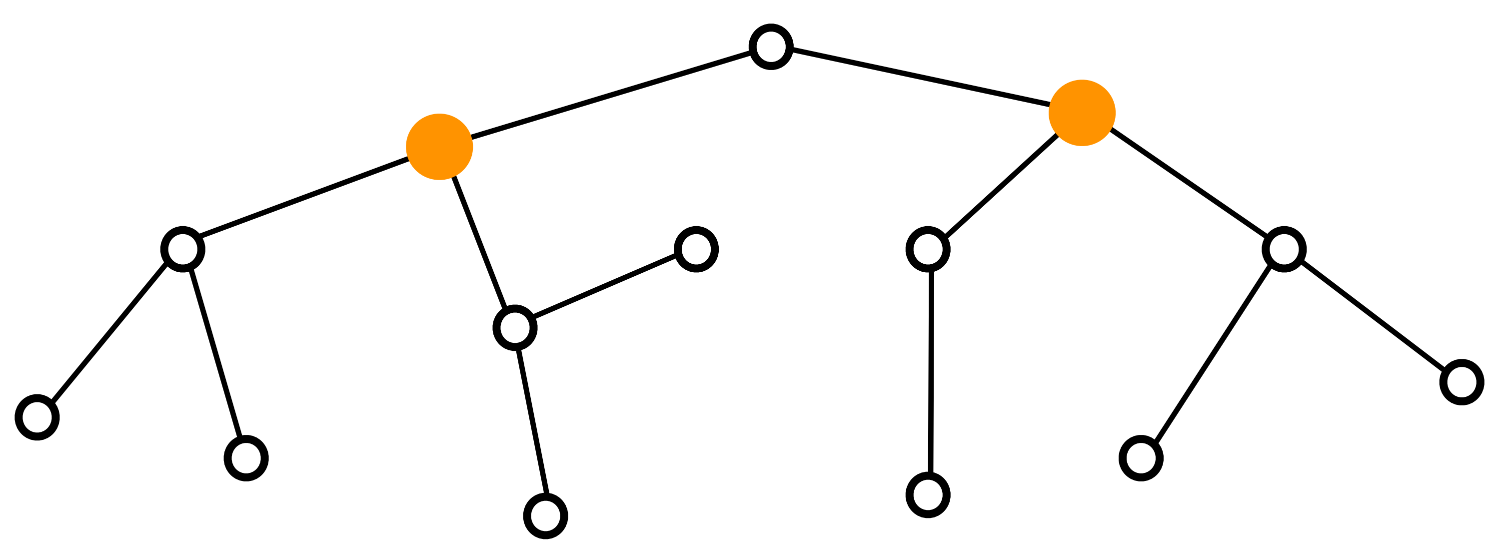
# Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

<u>**Lemma:**</u> [DZ'17]

Given a tree *T* of size *n*,

- **Input:** integer *k*
- **Output:** remove $O(n/k)$ special vertices to partition *T* into subtrees of size at most *k*



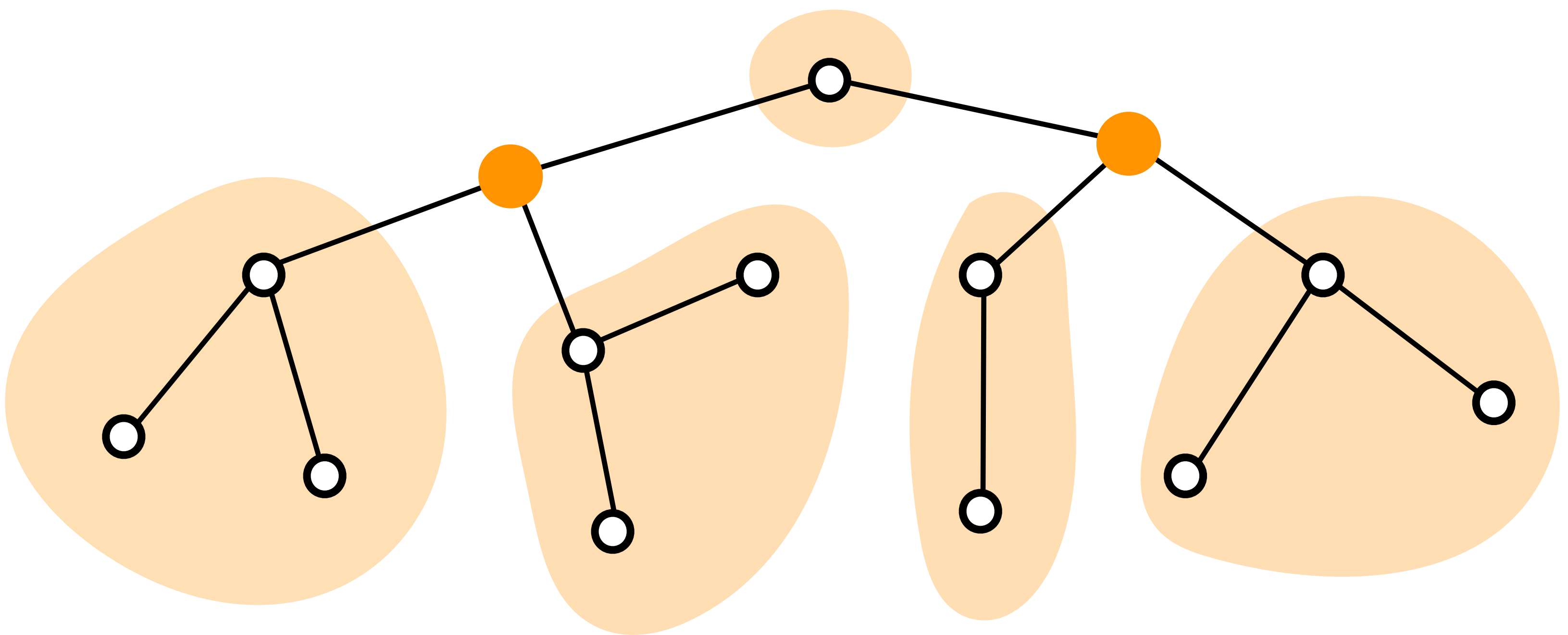
# Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

<u>**Lemma:**</u> [DZ'17]

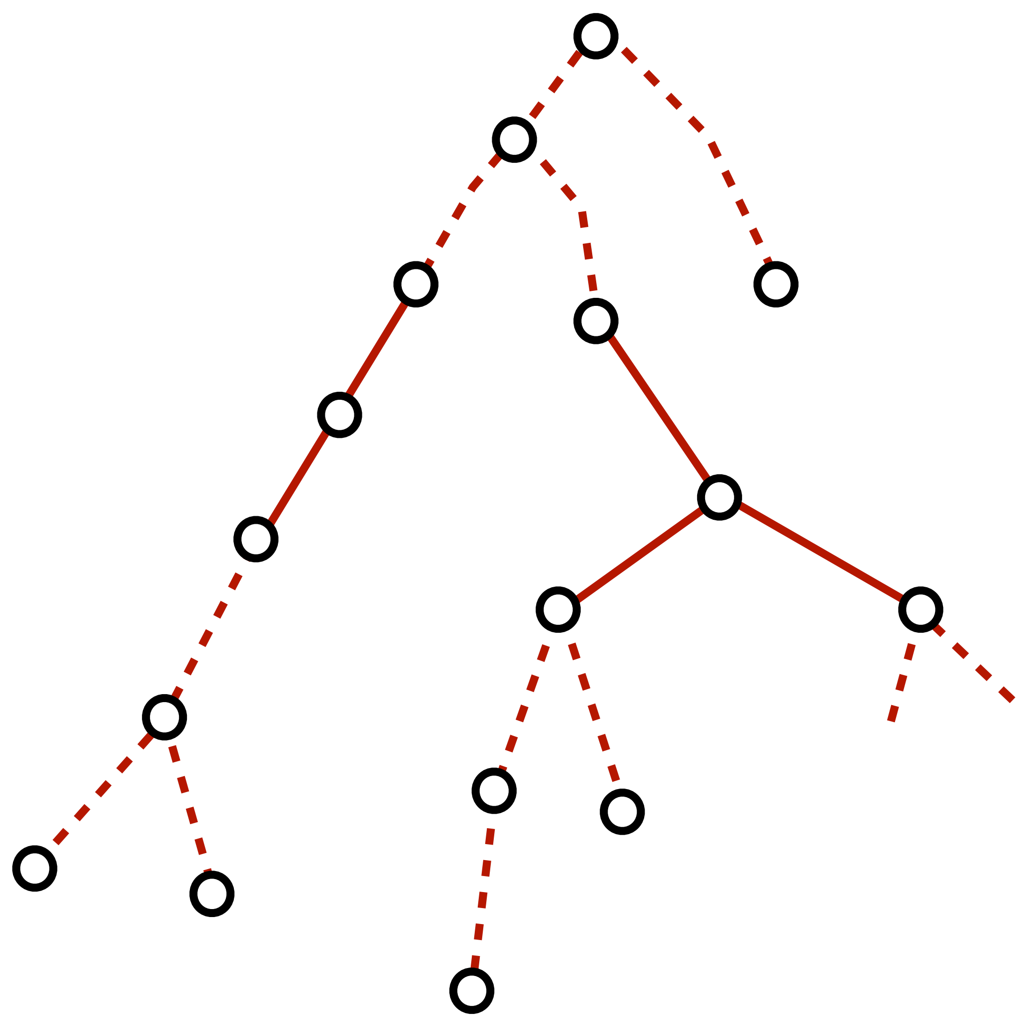
Given a tree *T* of size *n*,

- **Input:** integer *k*
- **Output:** remove $O(n/k)$ special vertices to partition *T* into subtrees of size at most *k*



# Finding highest ancestors

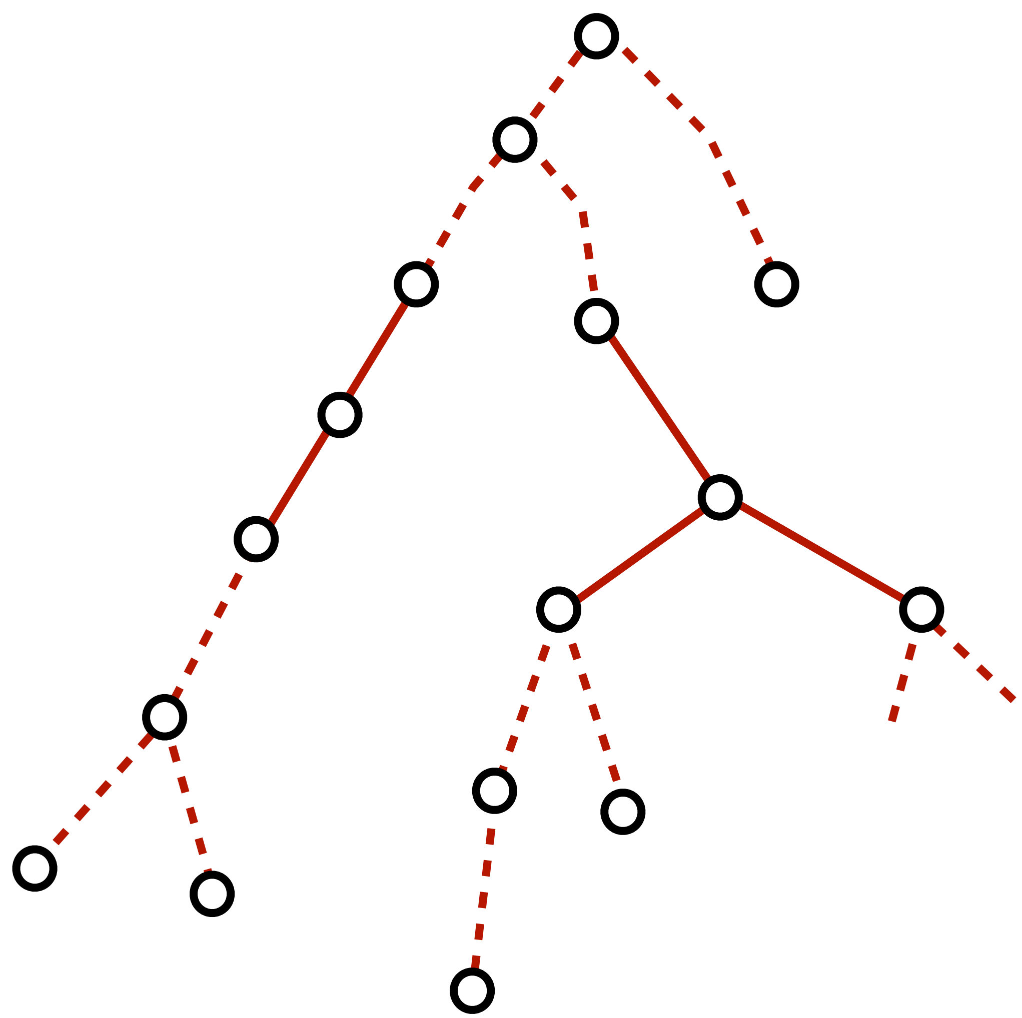
New data structure for finding highest ancestors



# Finding highest ancestors

New data structure for finding highest ancestors

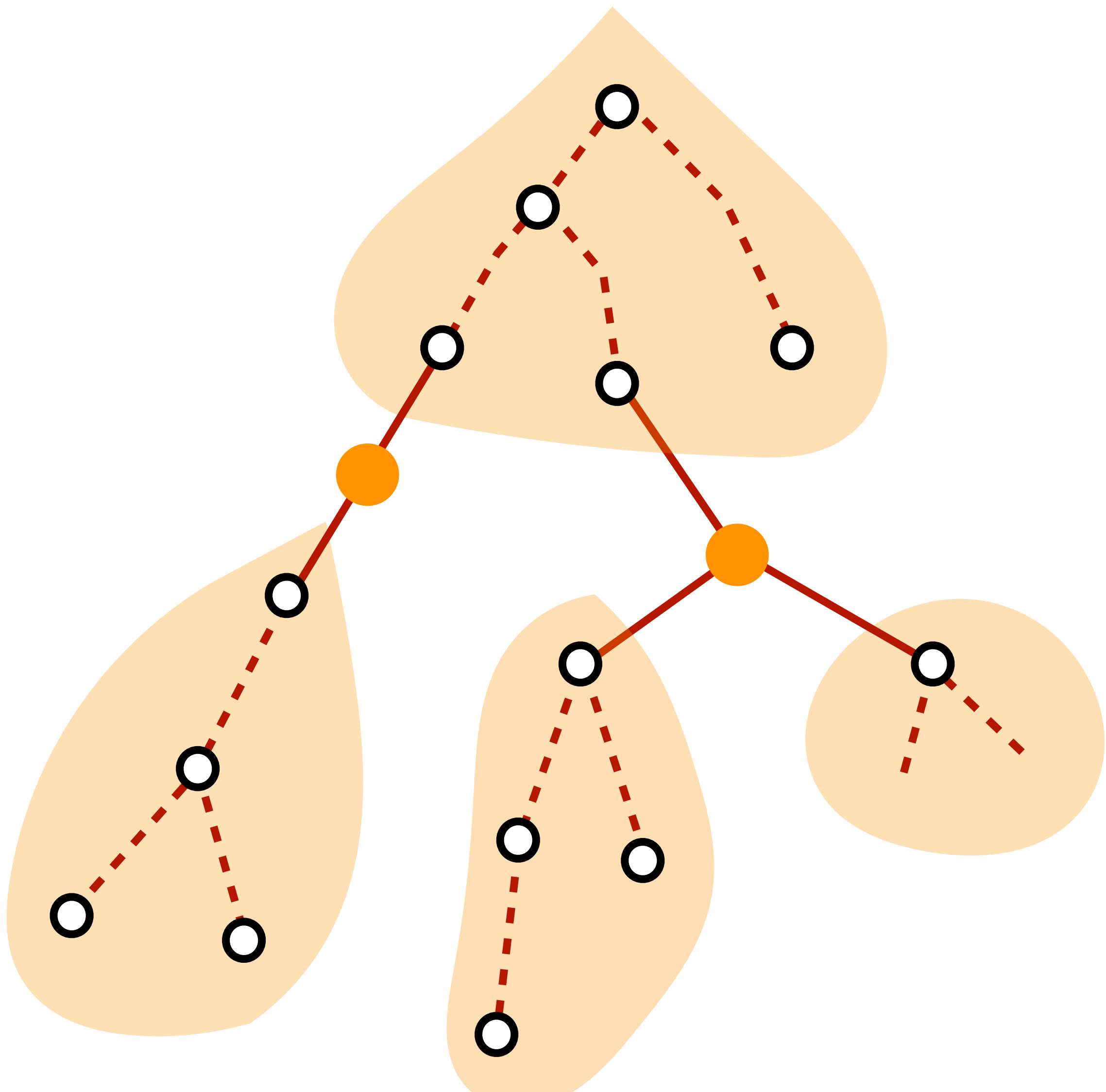
**Apply tree partition with** $k = \log n$
**So every subtree has size** $O(\log n)$



# Finding highest ancestors

New data structure for finding highest ancestors

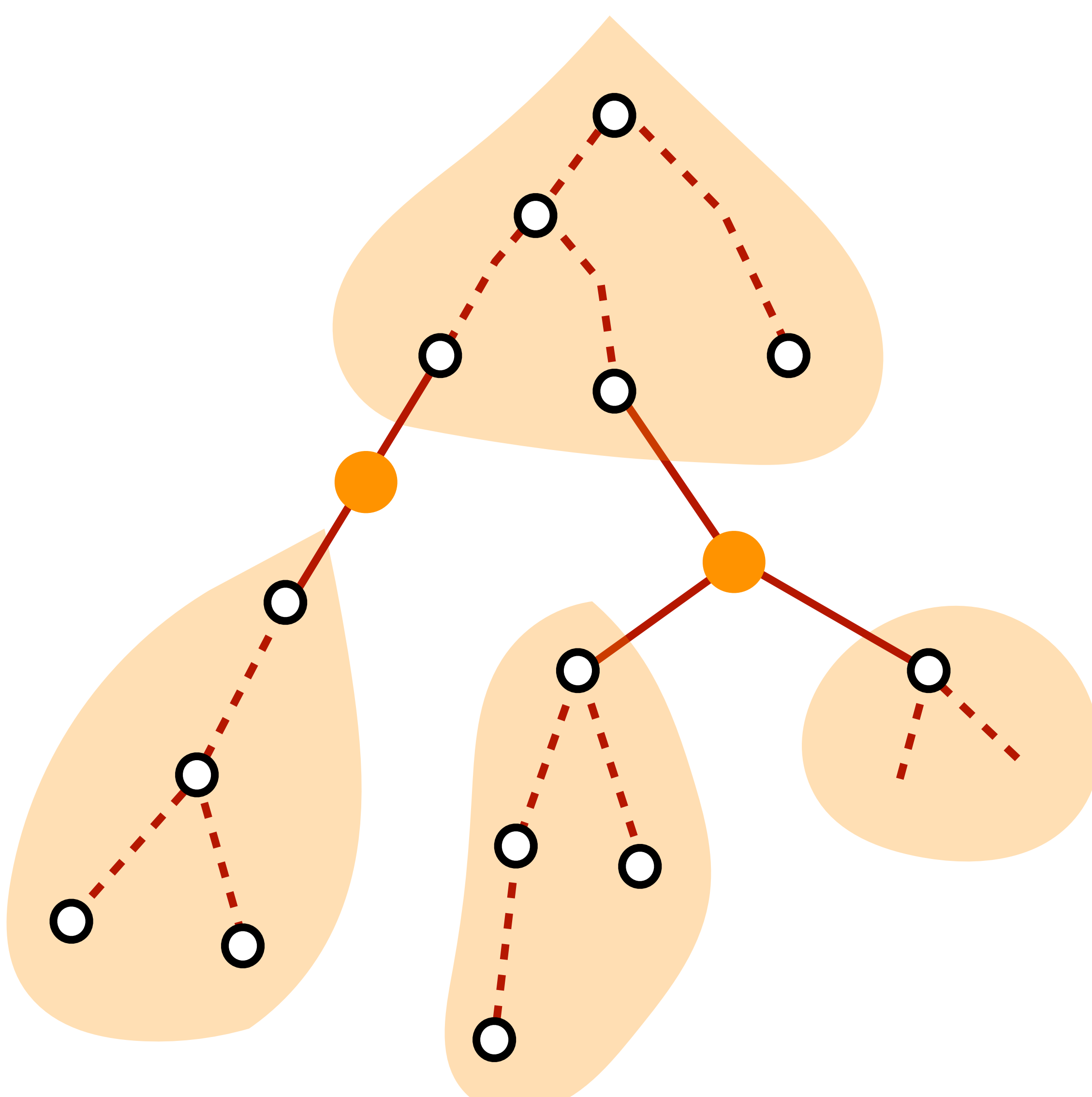
**Apply tree partition with** $k = \log n$
**So every subtree has size** $O(\log n)$



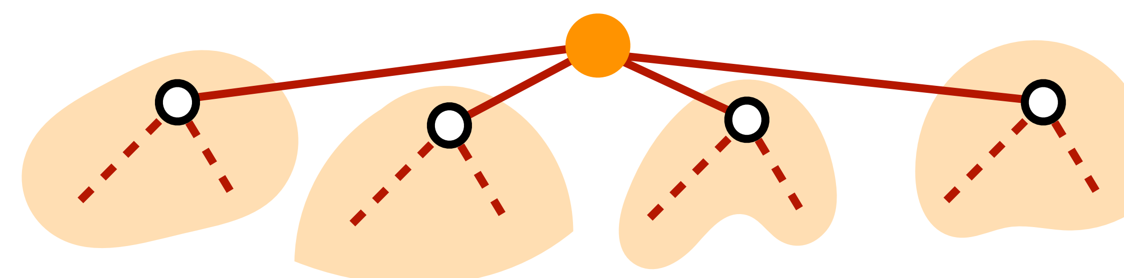
# Finding highest ancestors

New data structure for finding highest ancestors

**Apply** **tree partition** **with** $k = \log n$
**So every** **subtree** **has size** $O(\log n)$



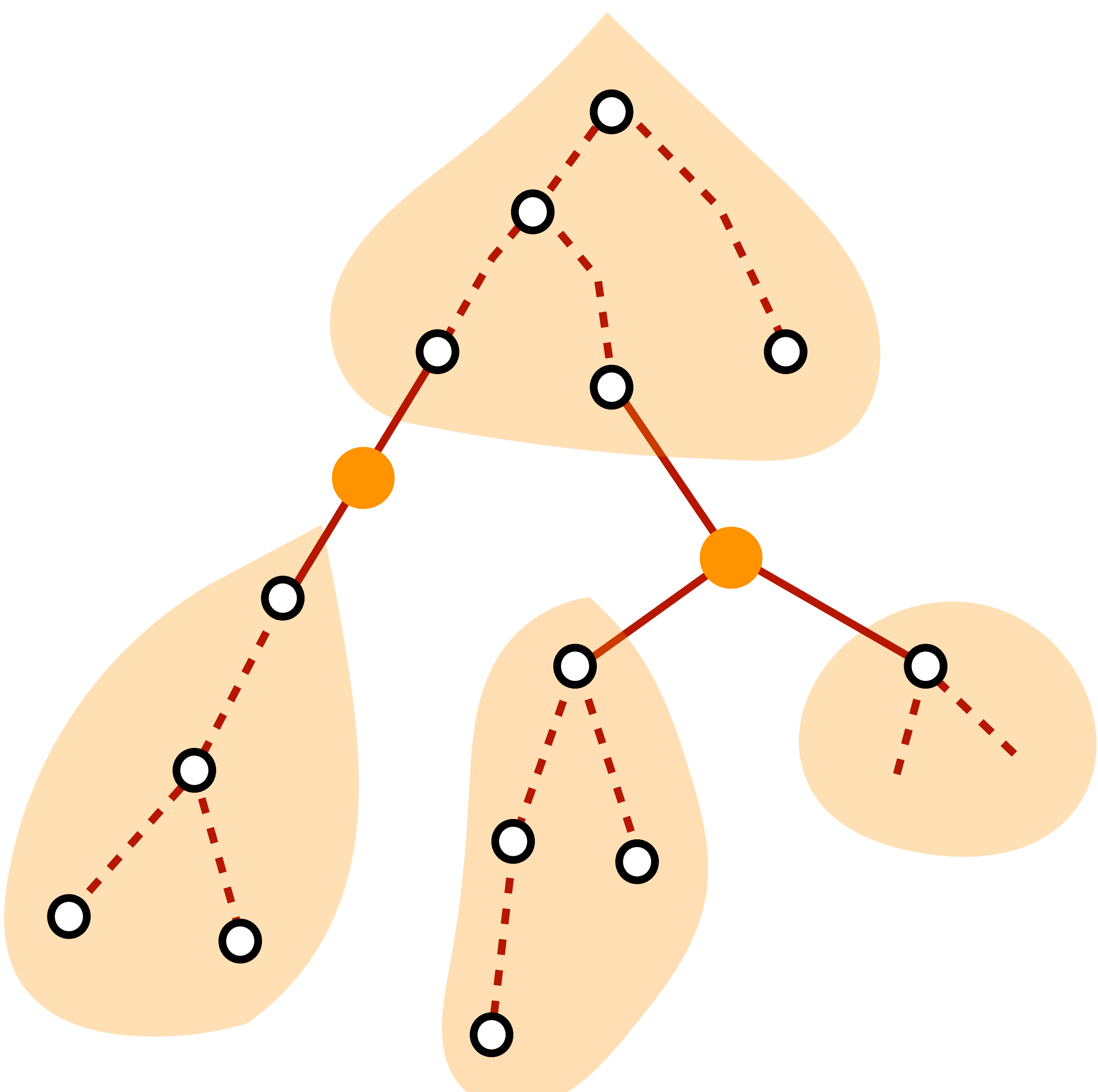
**Build** **fractional cascading** **on a bunch of** **subtrees**



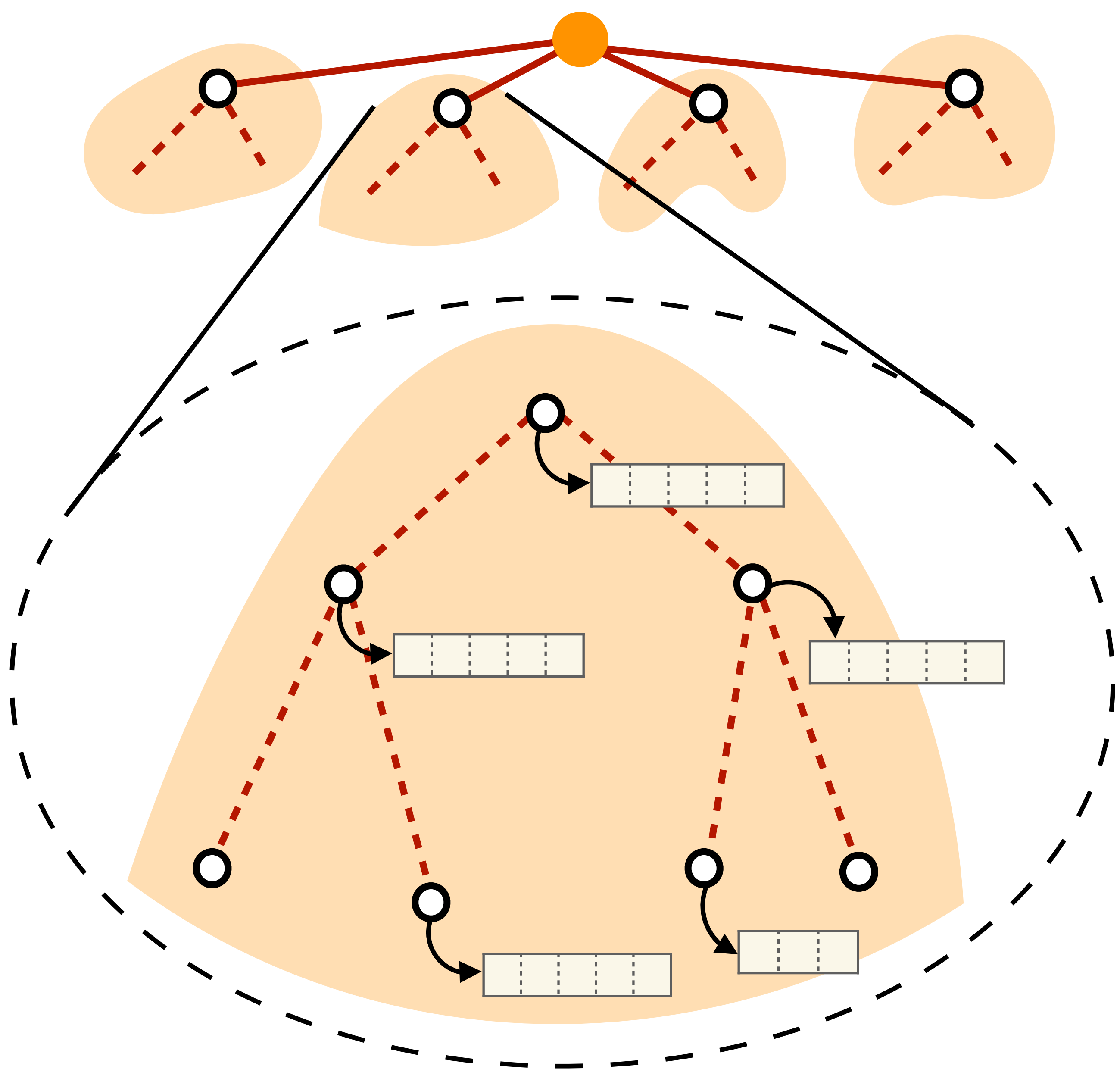
# Finding highest ancestors

New data structure for finding highest ancestors

**Apply tree partition with** $k = \log n$
**So every subtree has size** $O(\log n)$



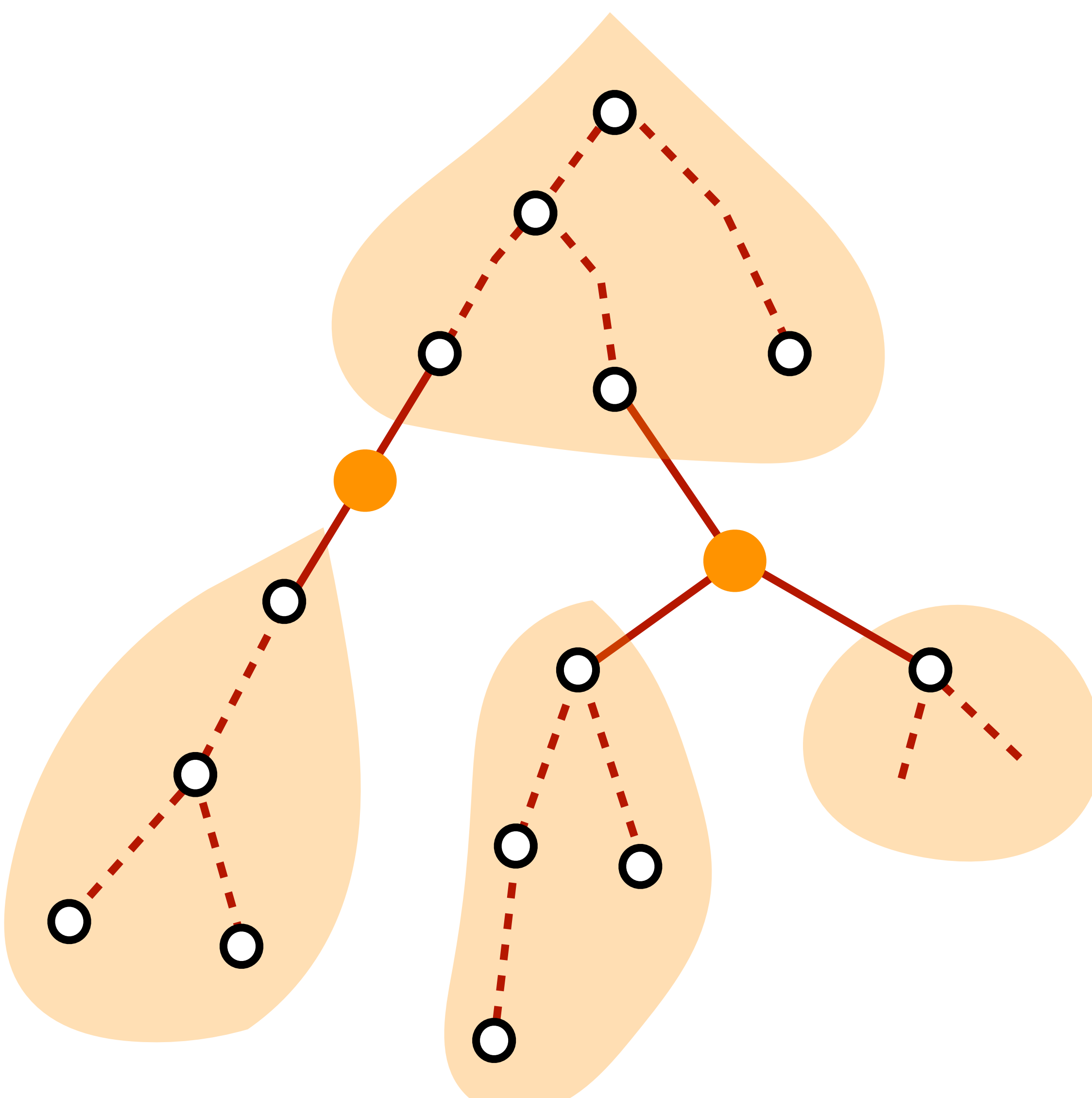
**Build fractional cascading on a bunch of subtrees**



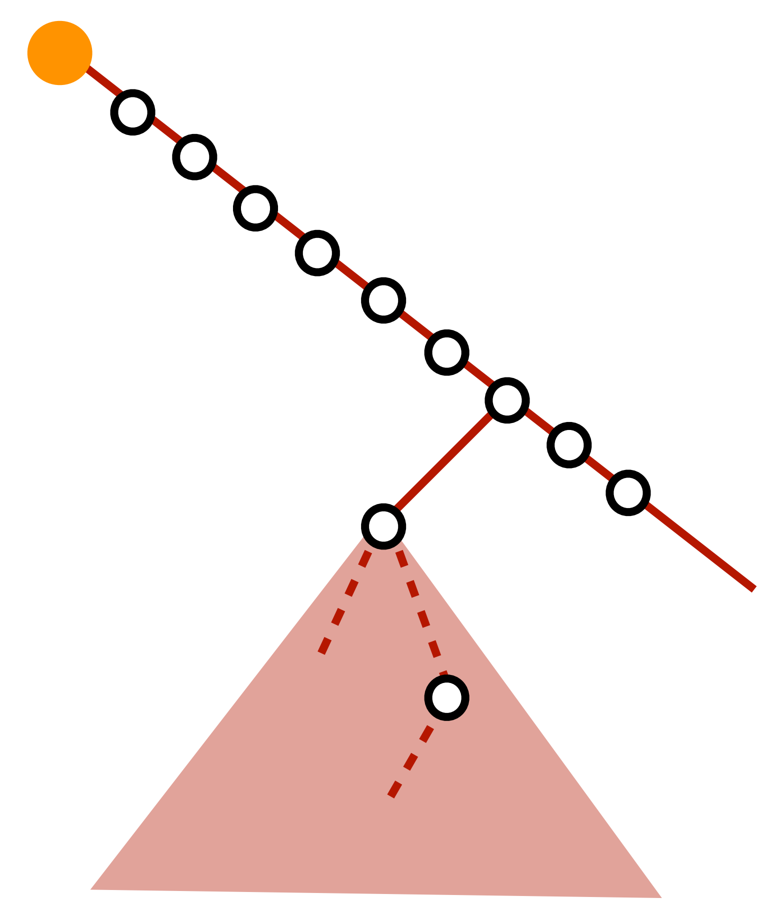
# Finding highest ancestors

New data structure for finding highest ancestors

**Apply tree partition with** $k = \log n$
**So every subtree has size** $O(\log n)$



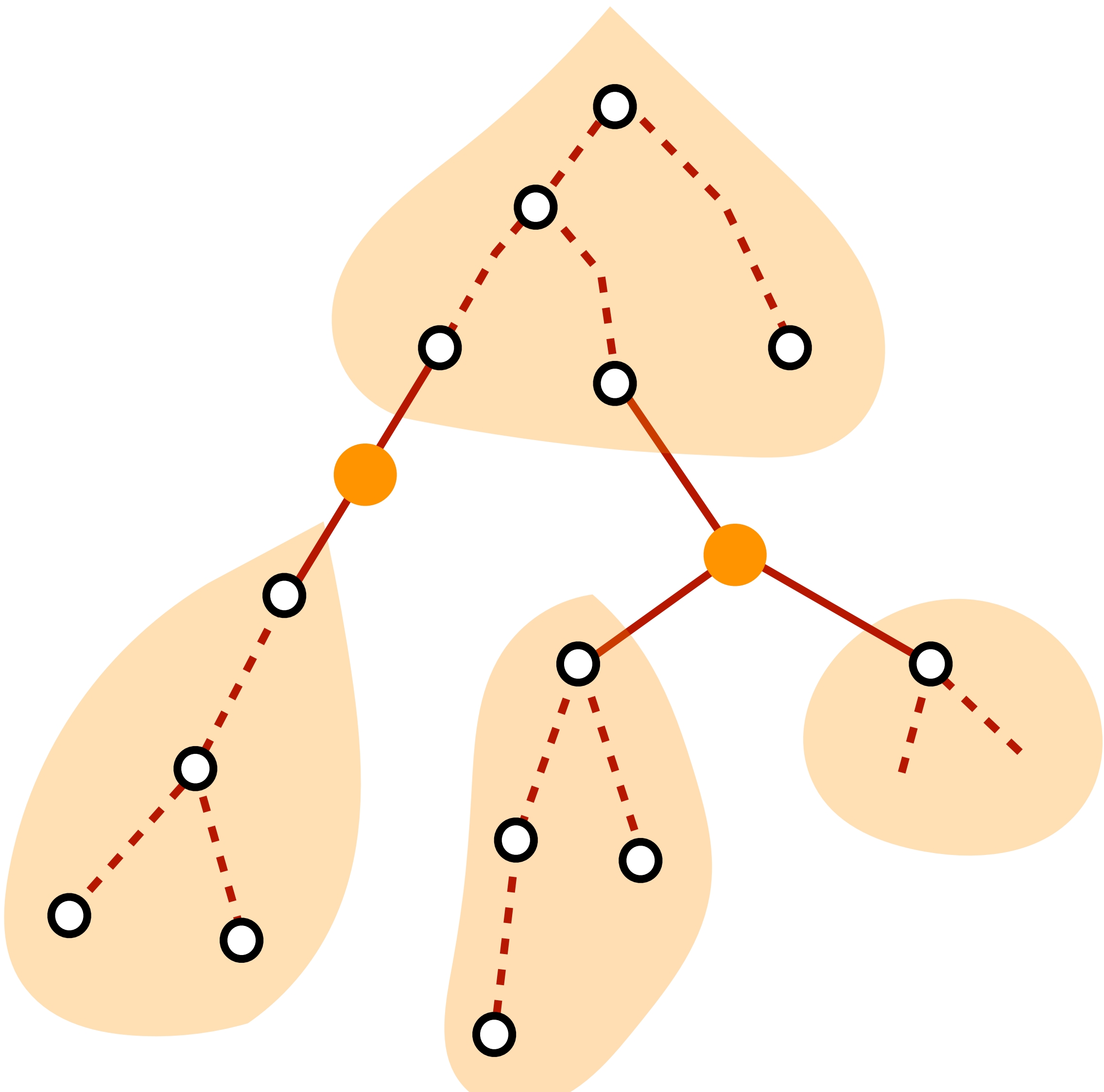
**Precompute for every choice of reverted tree path below the nearest special ancestor**



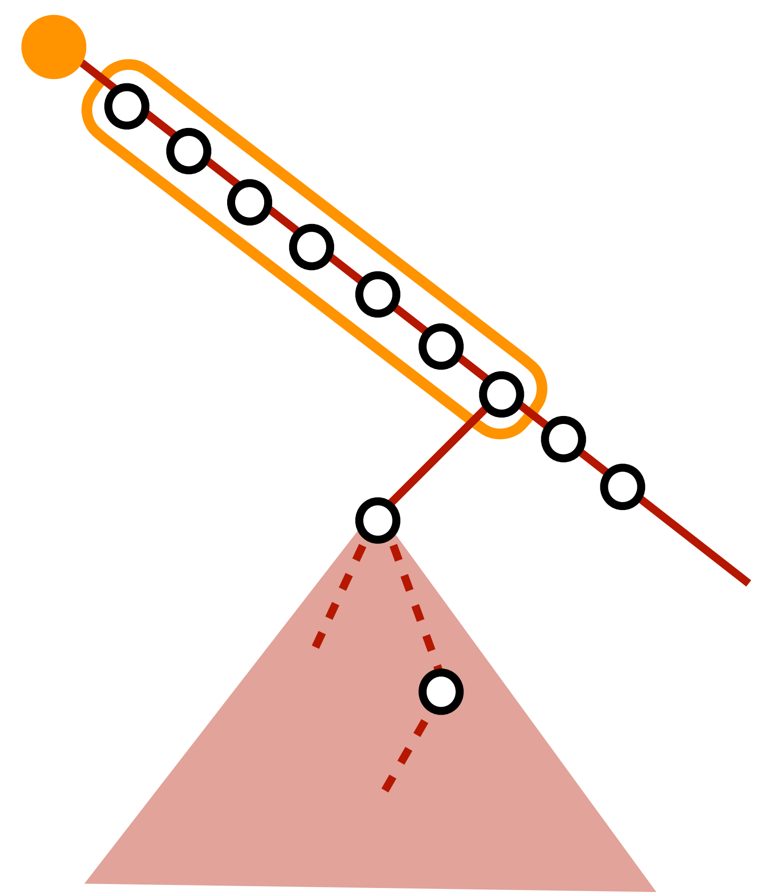
# Finding highest ancestors

New data structure for finding highest ancestors

**Apply tree partition with** $k = \log n$
**So every subtree has size** $O(\log n)$



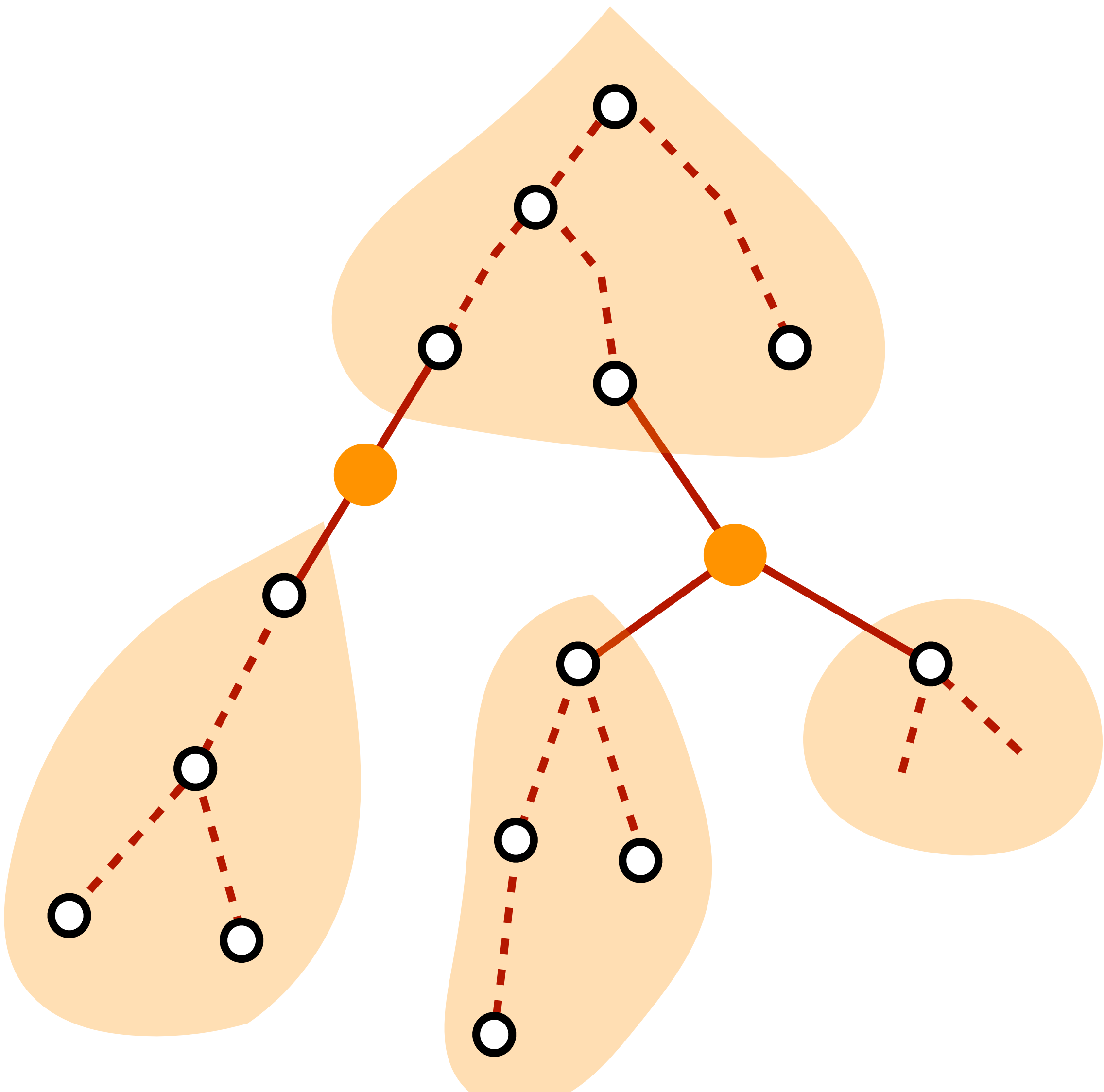
**Precompute for every choice of reverted tree path below the nearest special ancestor**



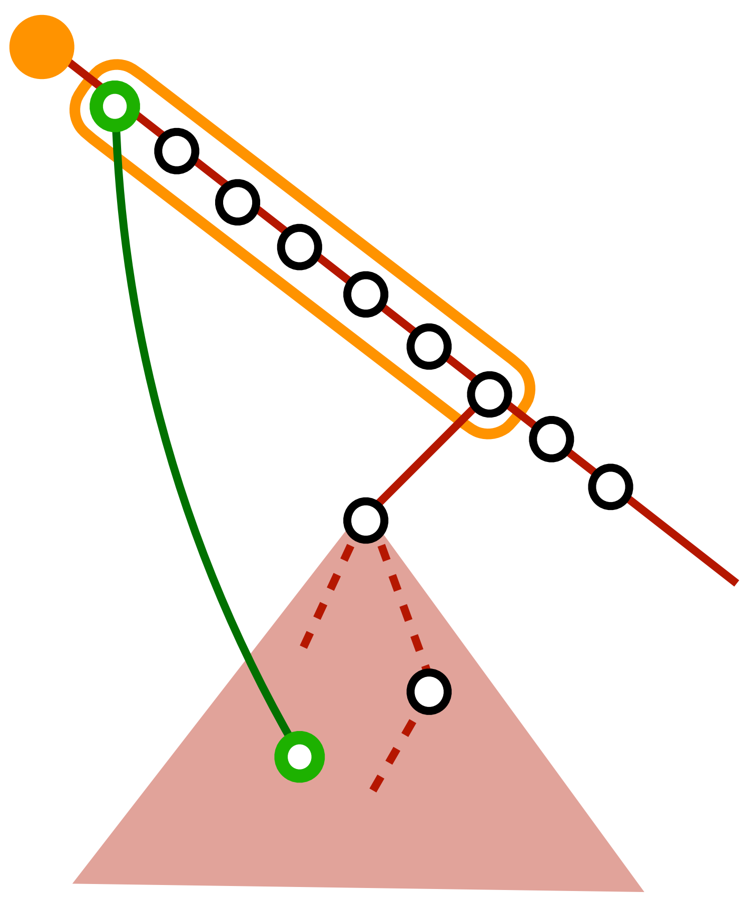
# Finding highest ancestors

New data structure for finding highest ancestors

**Apply tree partition with** $k = \log n$
**So every subtree has size** $O(\log n)$



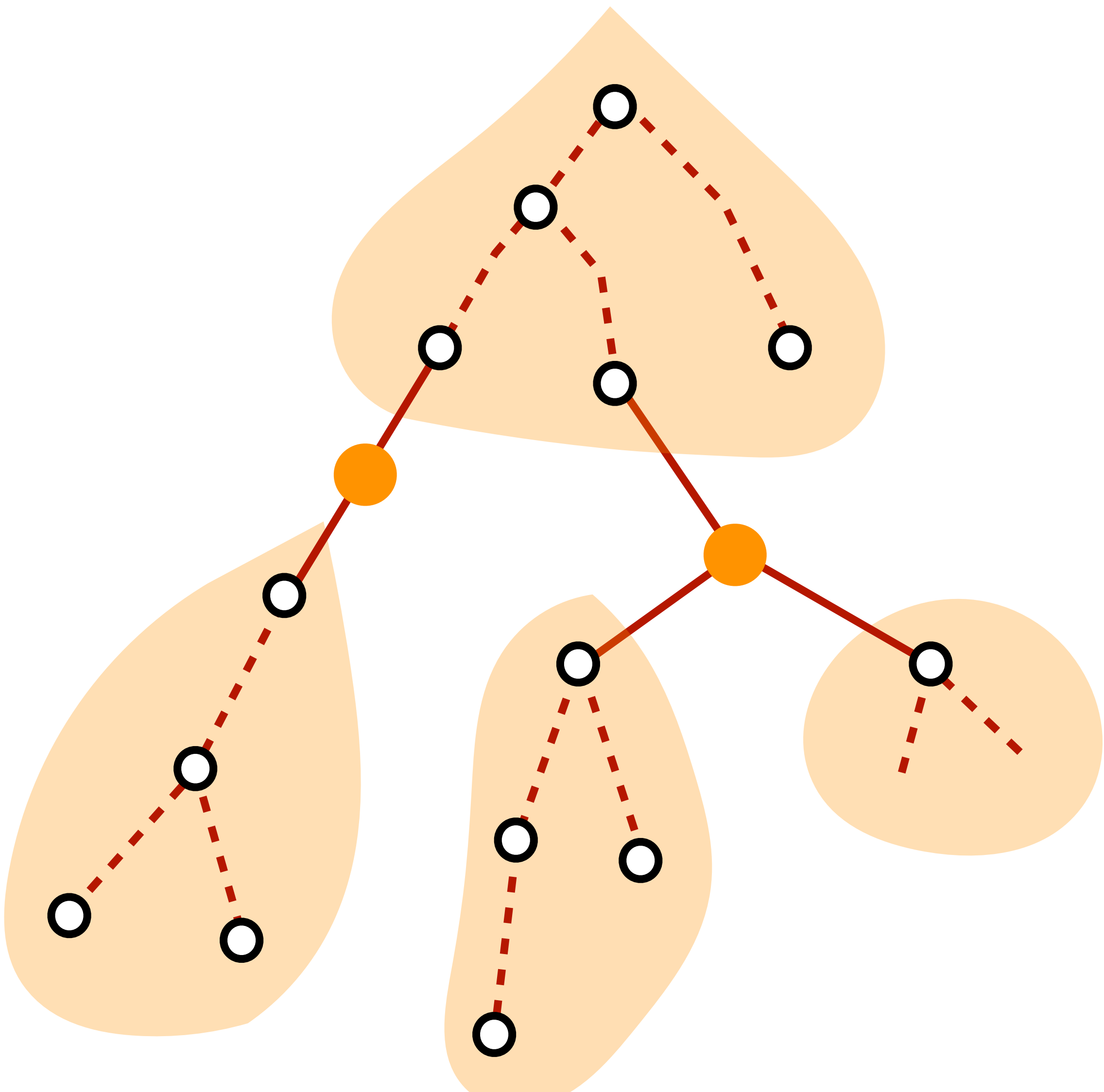
**Precompute for every choice of reverted tree path below the nearest special ancestor**



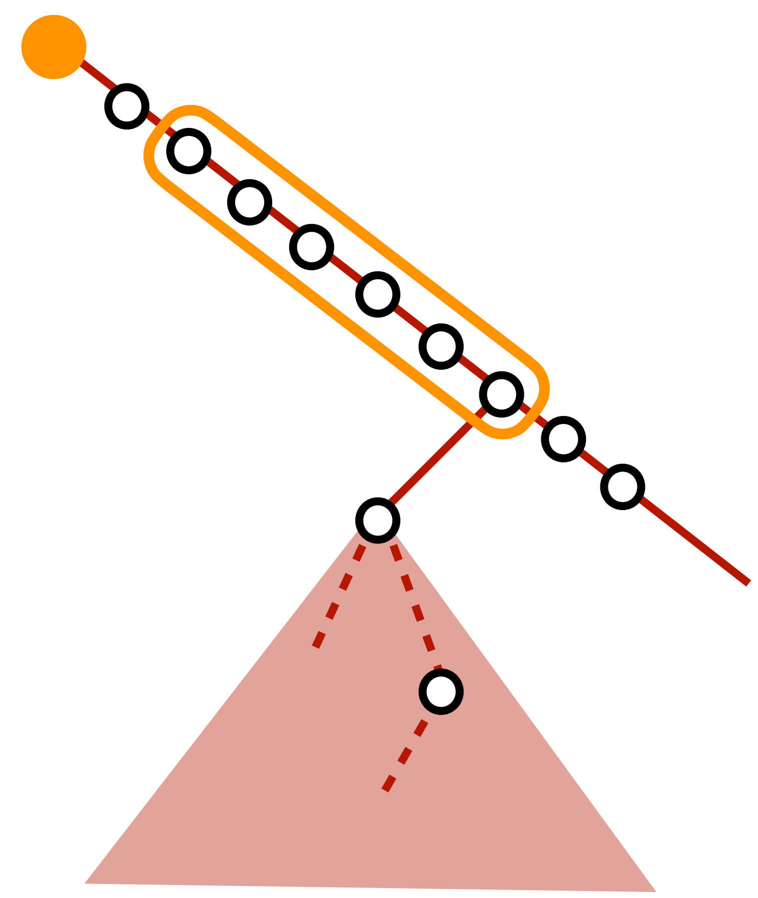
# Finding highest ancestors

New data structure for finding highest ancestors

**Apply tree partition with** $k = \log n$
**So every subtree has size** $O(\log n)$



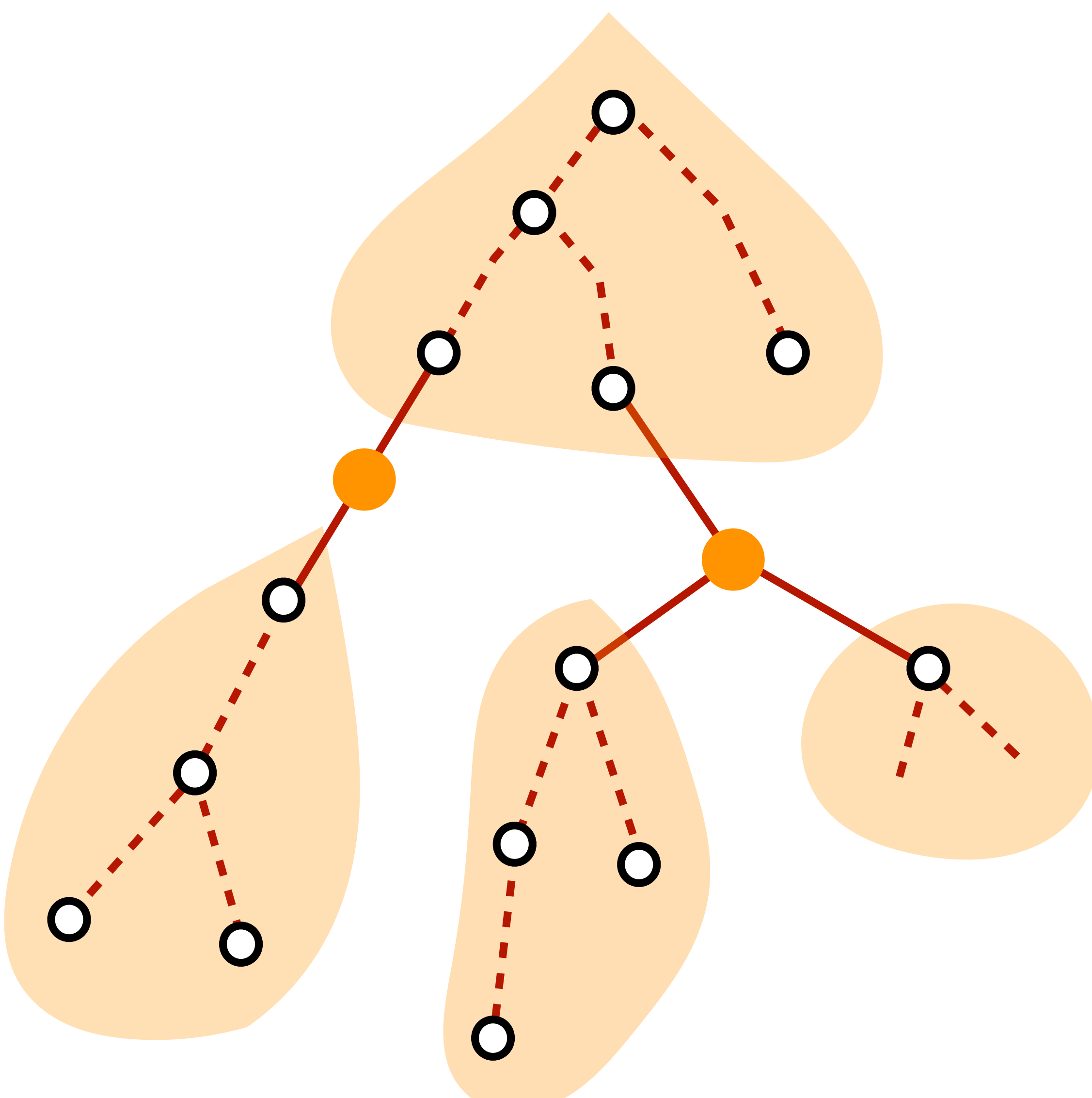
**Precompute for every choice of reverted tree path below the nearest special ancestor**



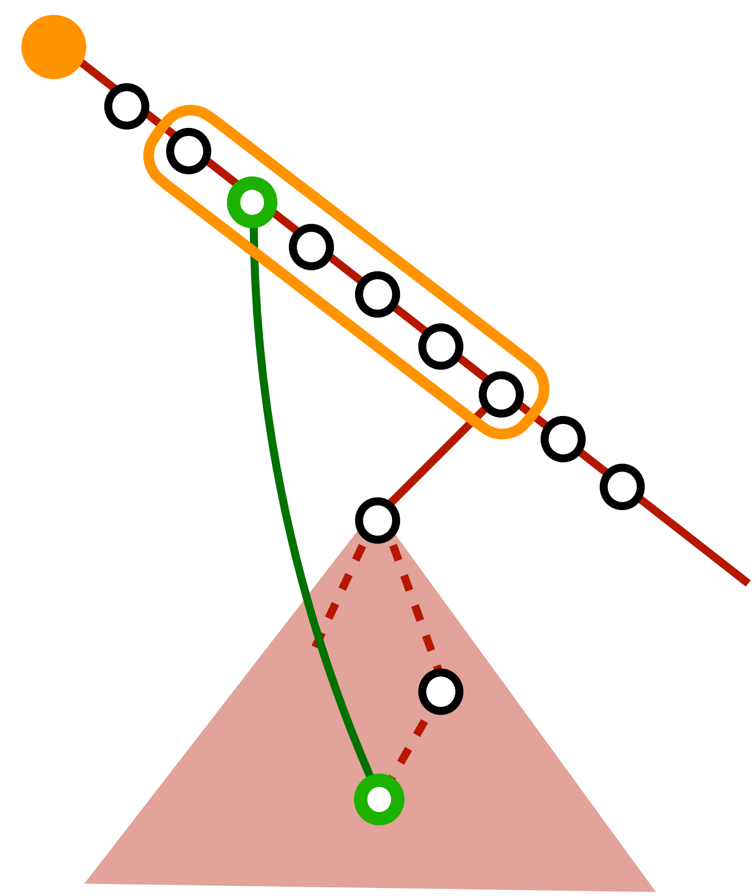
# Finding highest ancestors

New data structure for finding highest ancestors

**Apply tree partition with** $k = \log n$
**So every subtree has size** $O(\log n)$



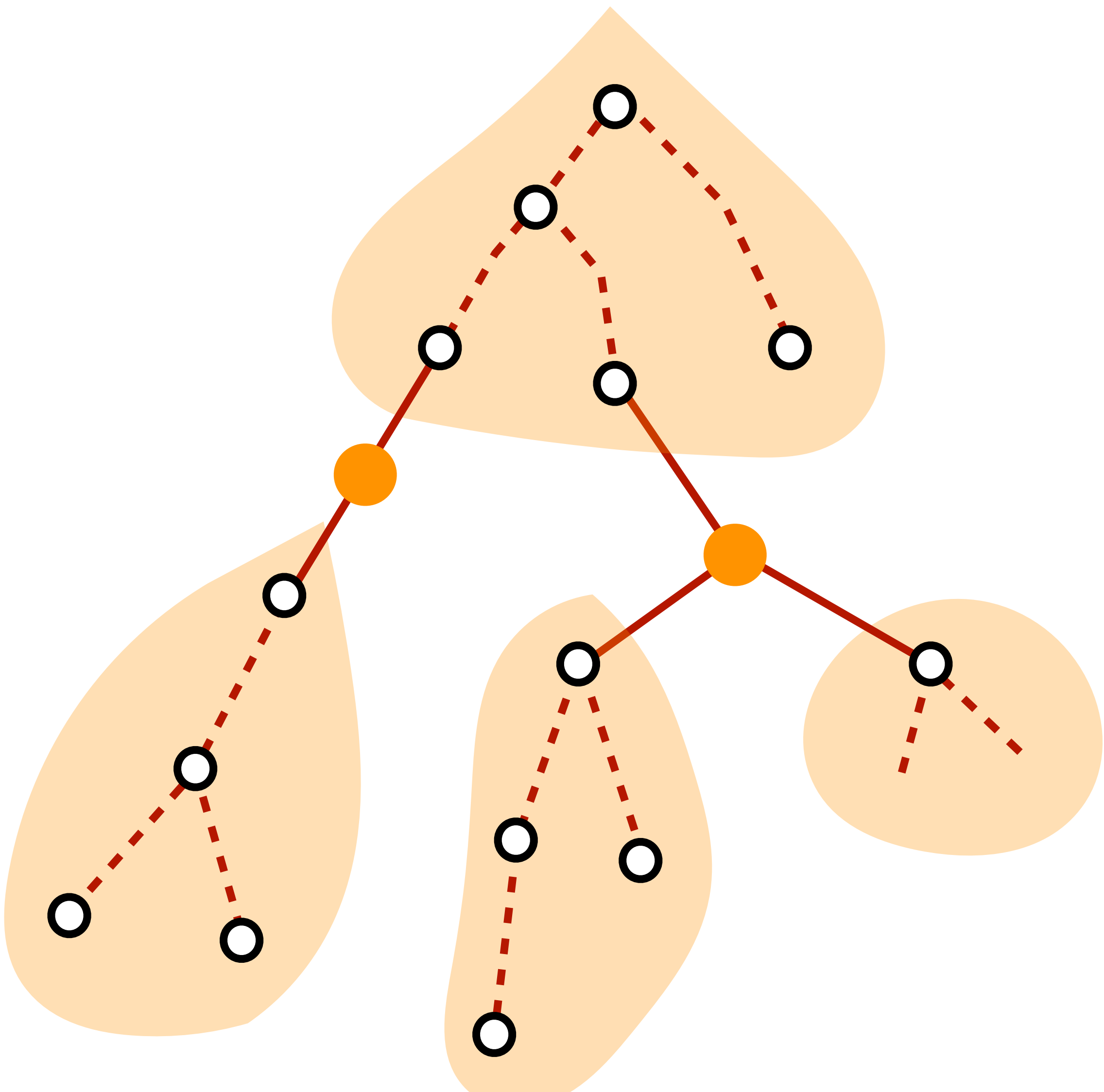
**Precompute for every choice of reverted tree path below the nearest special ancestor**



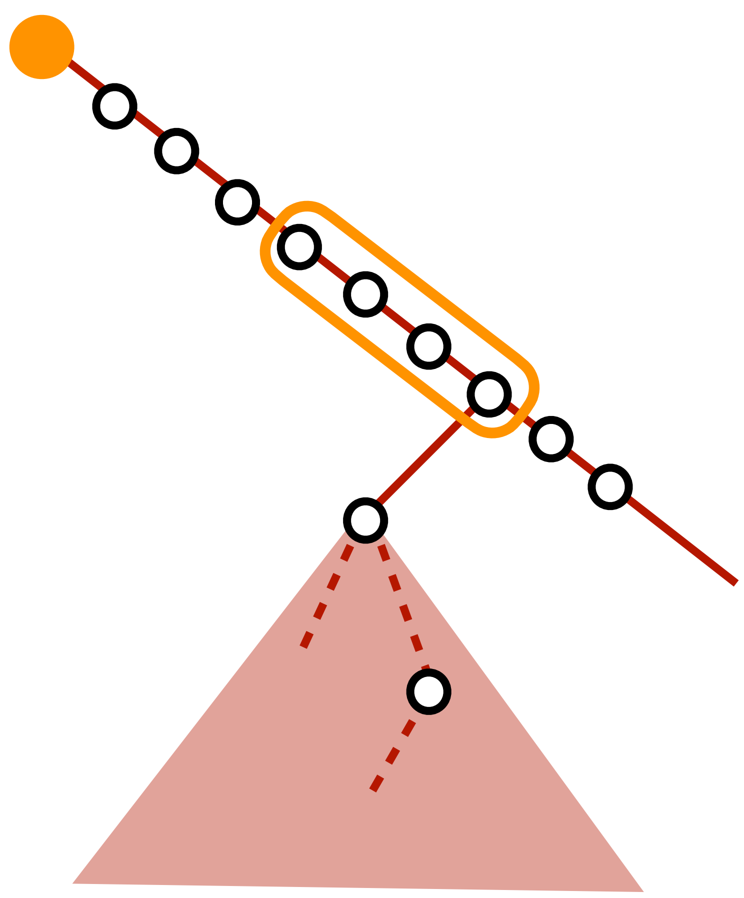
# Finding highest ancestors

New data structure for finding highest ancestors

**Apply tree partition with** $k = \log n$
**So every subtree has size** $O(\log n)$



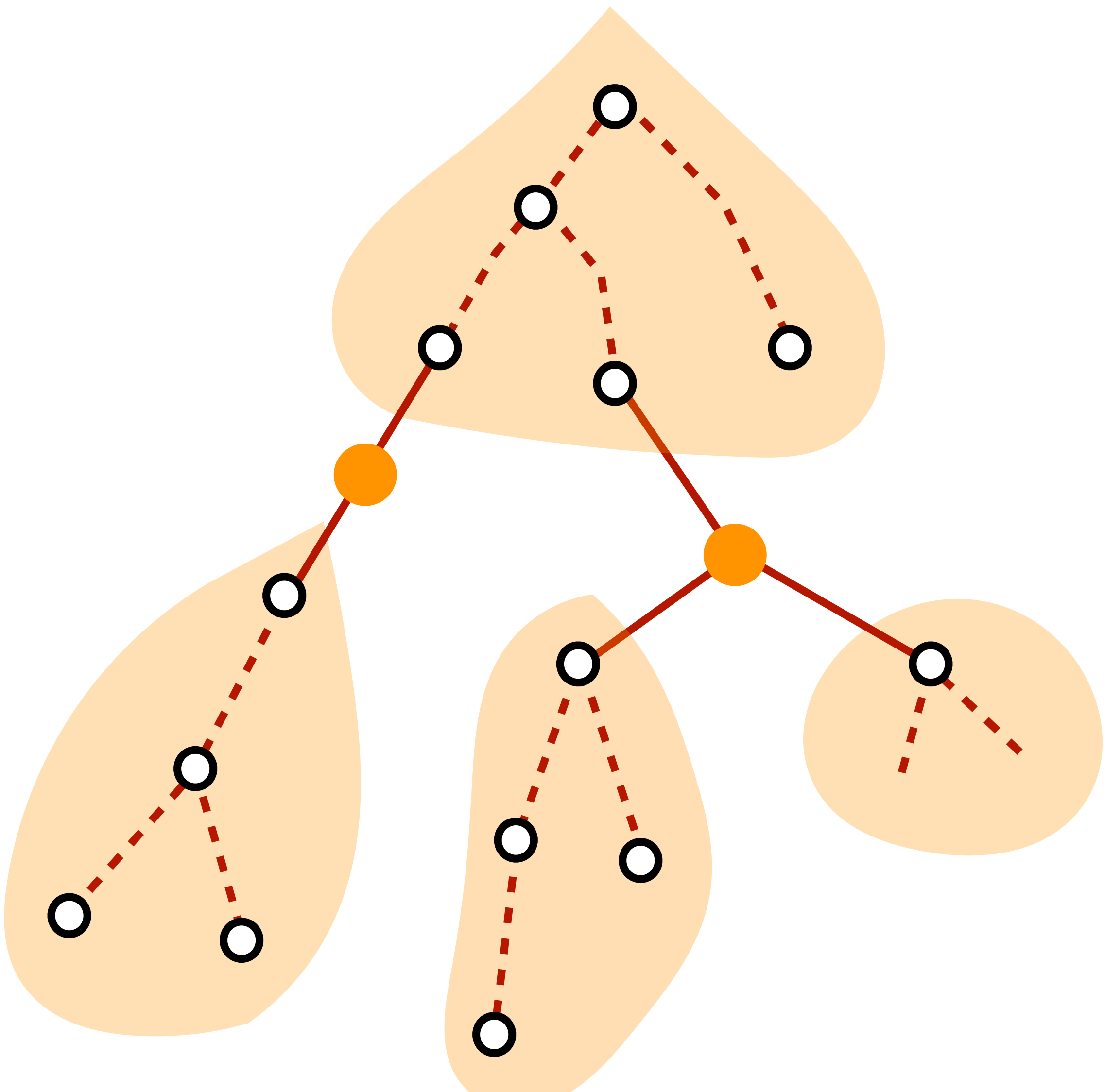
**Precompute for every choice of reverted tree path below the nearest special ancestor**



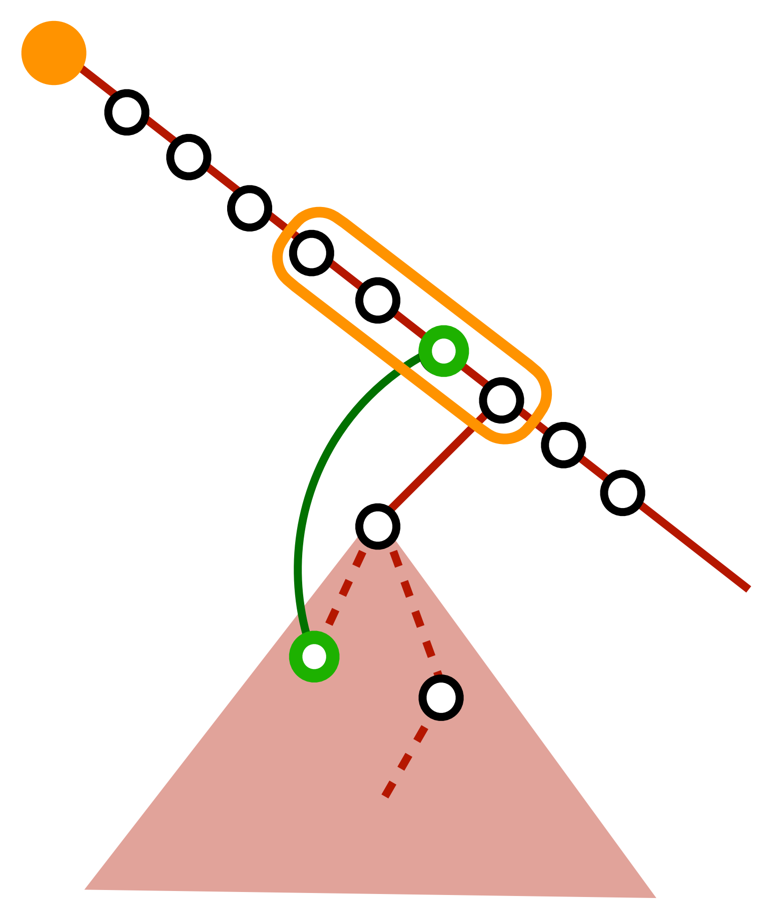
# Finding highest ancestors

New data structure for finding highest ancestors

**Apply tree partition with** $k = \log n$
**So every subtree has size** $O(\log n)$



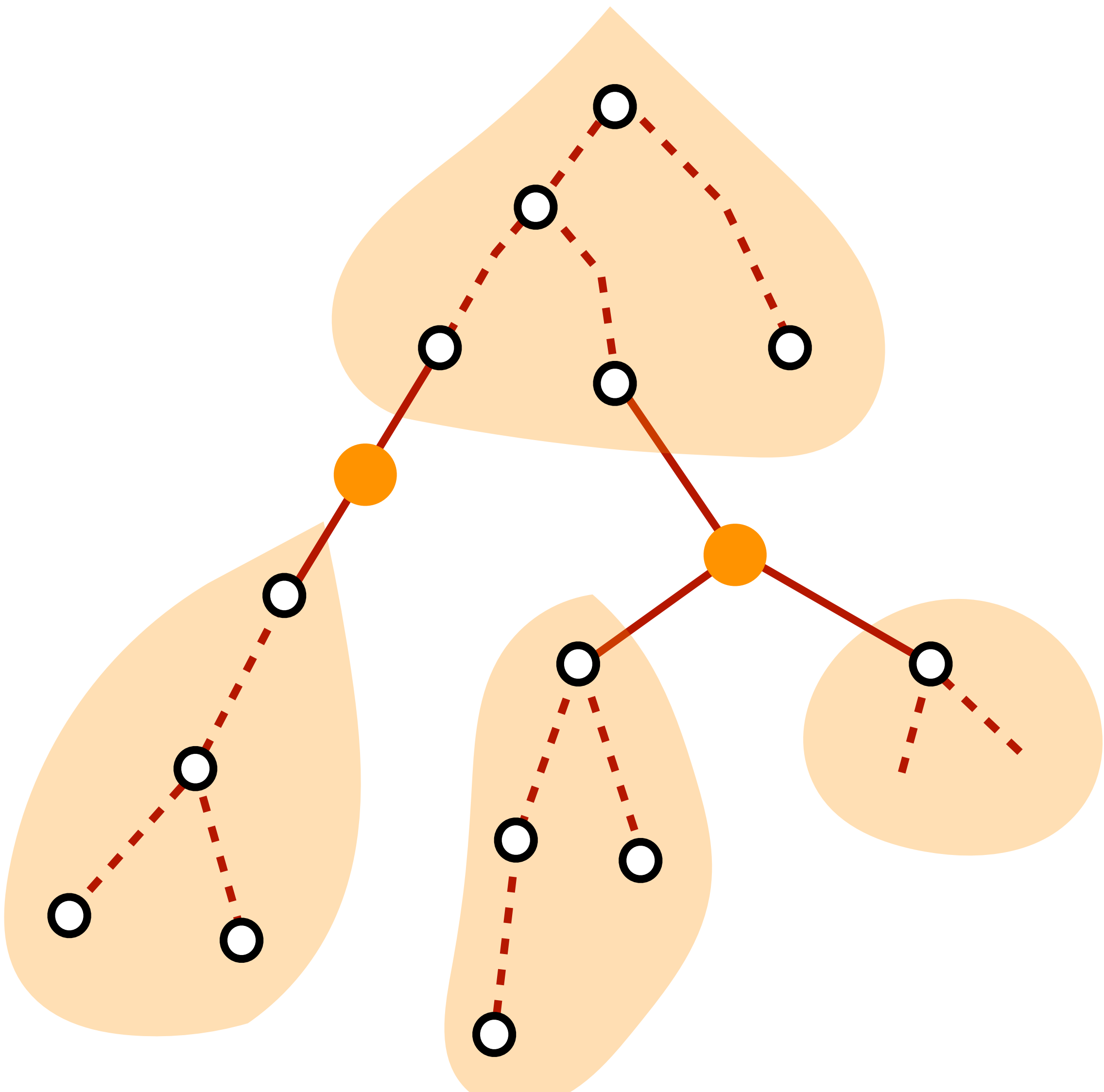
**Precompute for every choice of reverted tree path below the nearest special ancestor**



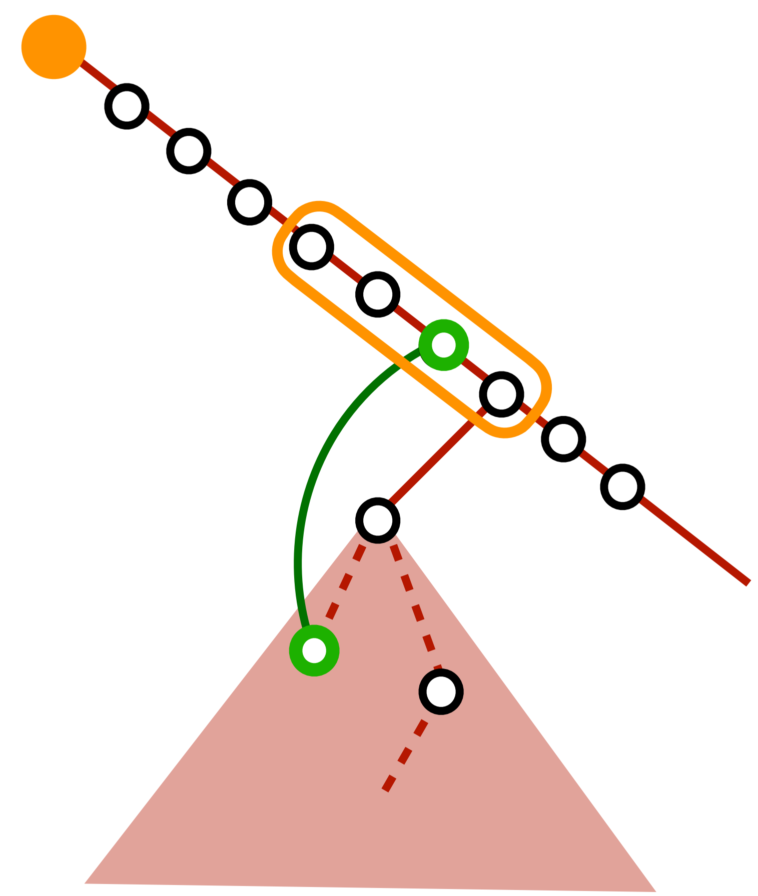
# Finding highest ancestors

New data structure for finding highest ancestors

**Apply tree partition with $k = \log n$**
**So every subtree has size $O(\log n)$**



**Precompute for every choice of reverted tree path below the nearest special ancestor**



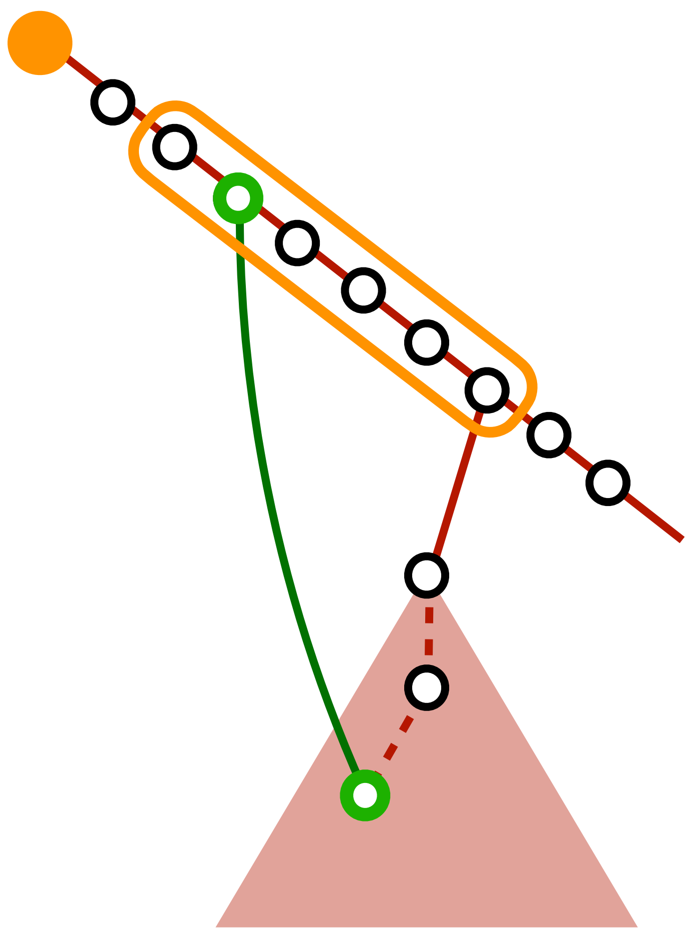
$O(n \log n)$ **entries in total**

# Highest ancestors in $O(n)$ total time

Consider three cases below

# Highest ancestors in *O(n)* total time

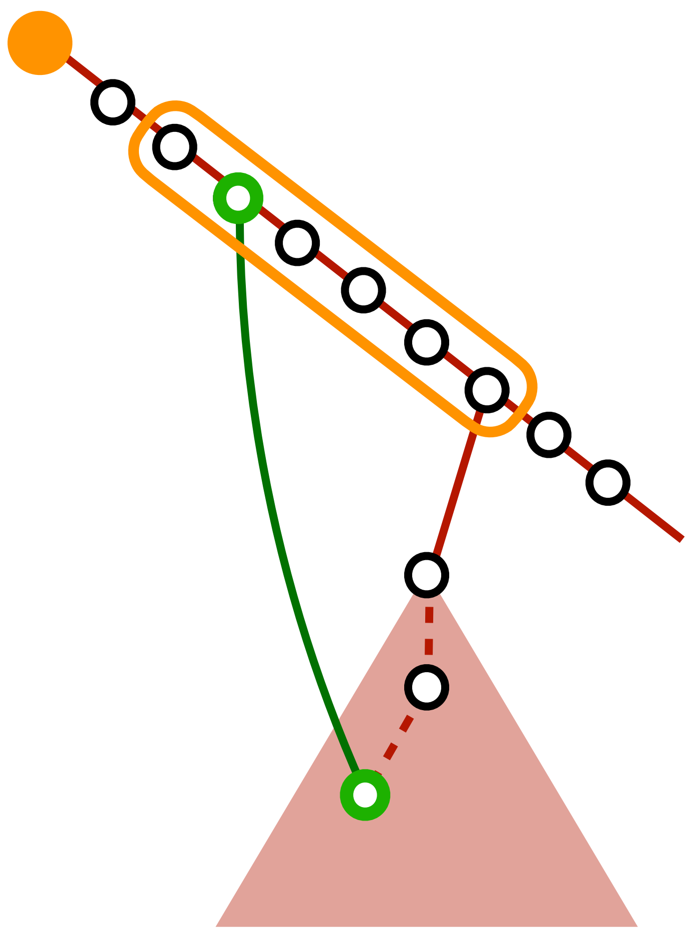
Consider three cases below



**The reverted tree path contains no special vertices**

# Highest ancestors in *O(n)* total time

Consider three cases below

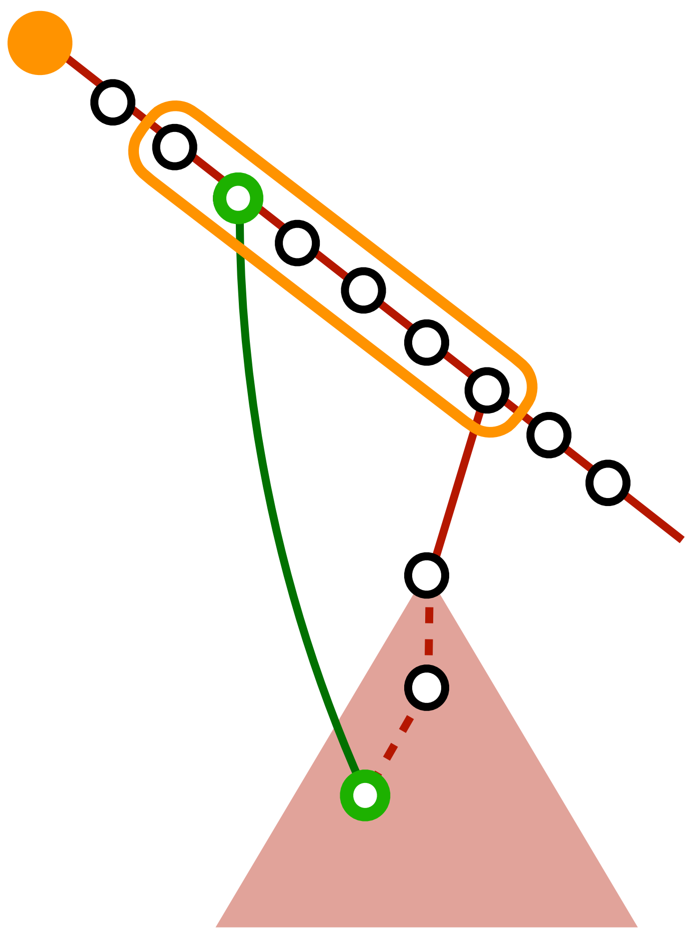


**The reverted tree path contains no special vertices**

**Use precomputed entries**
***O(1)* time**

# Highest ancestors in *O(n)* total time

Consider three cases below



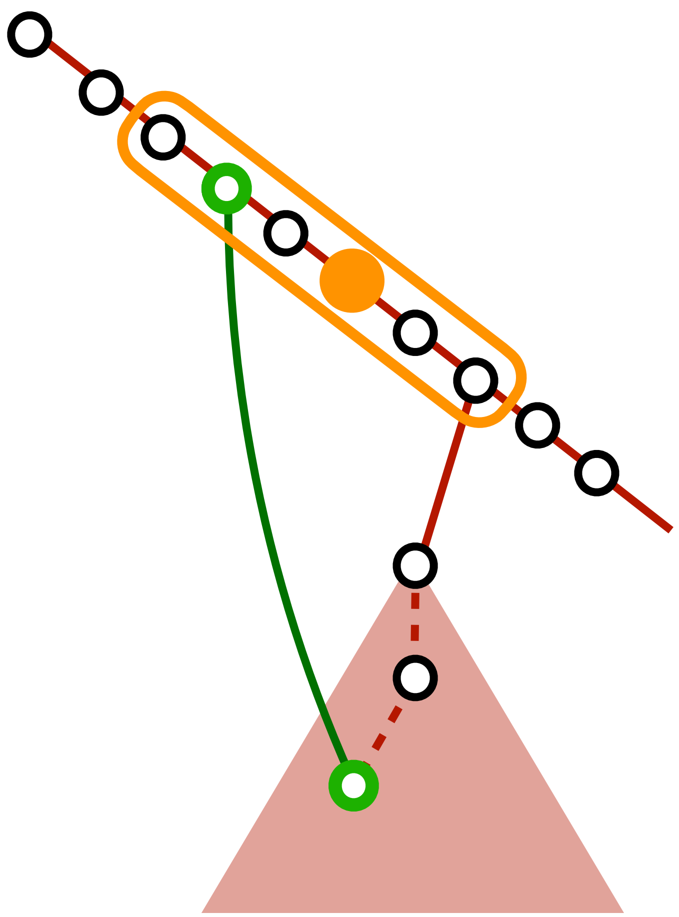
**The reverted tree path contains no special vertices**

**Use precomputed entries *O(1)* time**

**Total time = *O(n)***

# Highest ancestors in *O(n)* total time

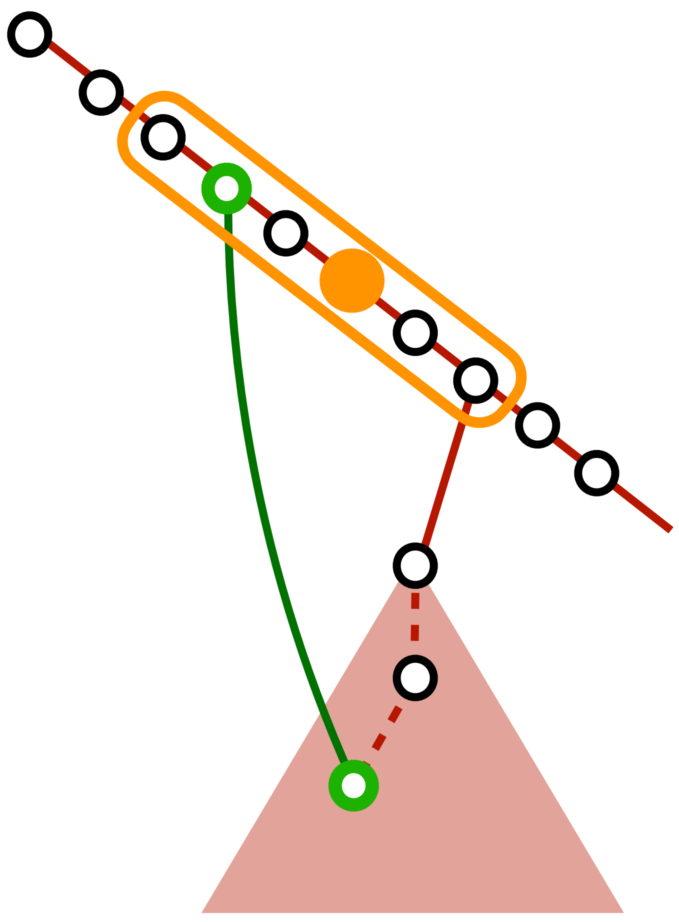
Consider three cases below



**The reverted tree path contains a special vertex, and subtree-size > *log n***

# Highest ancestors in *O(n)* total time

Consider three cases below

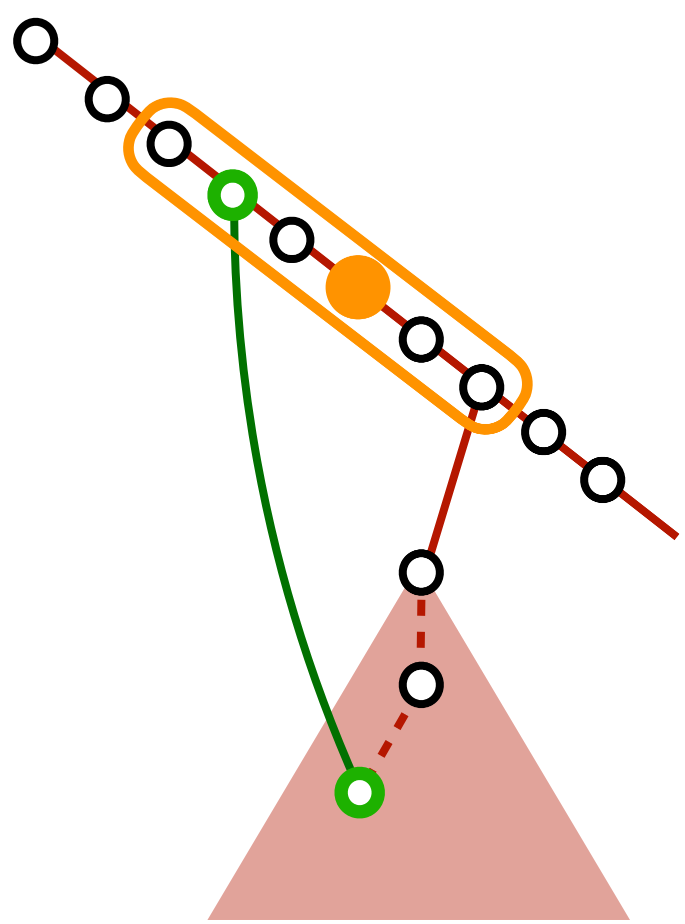


**The reverted tree path contains a special vertex, and subtree-size > *log n***

**Apply 2D-range minimum *O(log n)* time**

# Highest ancestors in *O(n)* total time

Consider three cases below



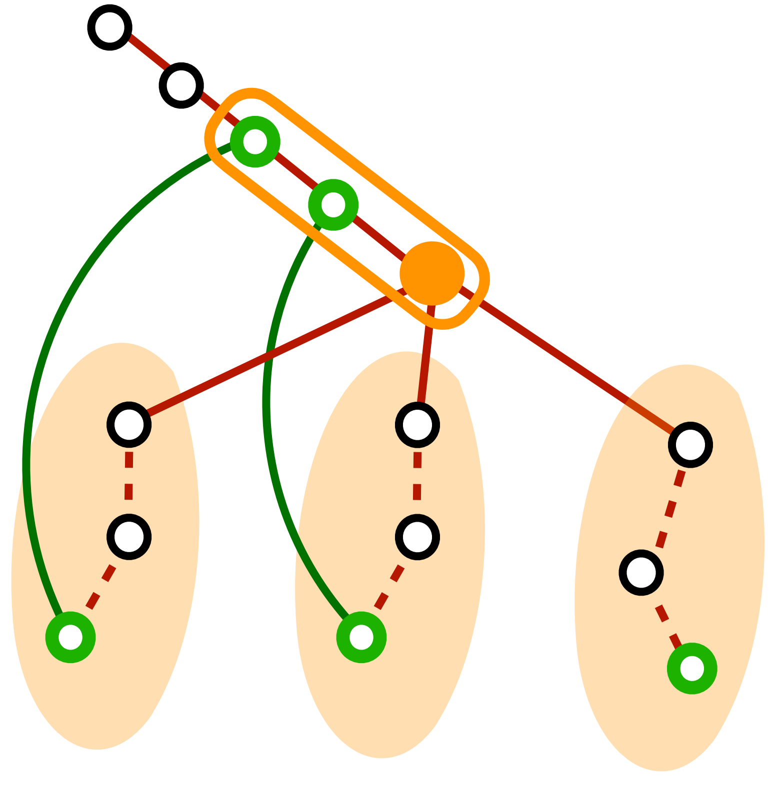
**The reverted tree path contains a special vertex, and subtree-size > *log n***

**Apply 2D-range minimum *O(log n)* time**

**One can prove this happens at most *O(n / log n)* times, so total time becomes *O(n)***

# Highest ancestors in *O(n)* total time

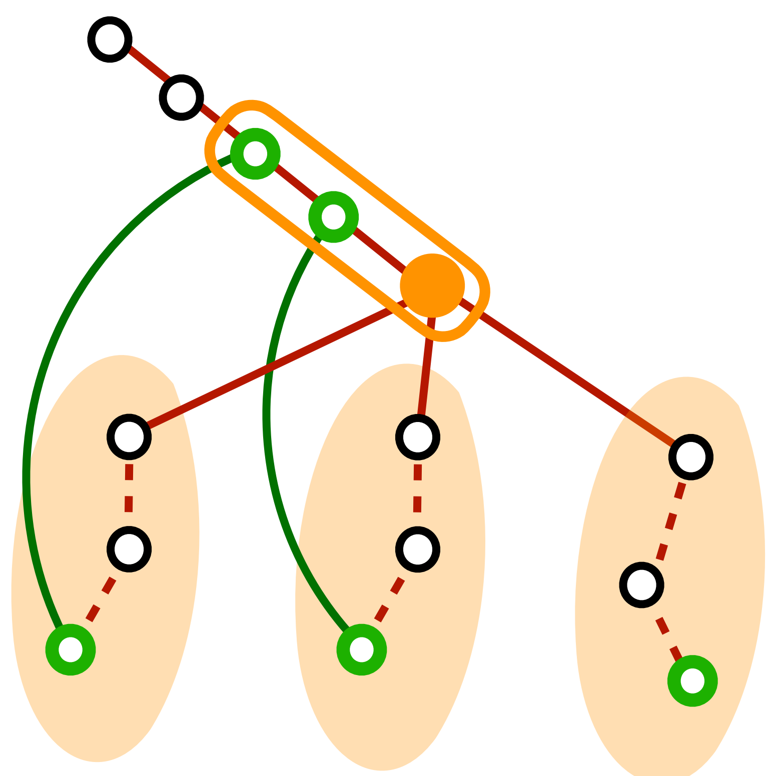
Consider three cases below



**A bunch of subtrees containing no special vertices**

# Highest ancestors in $O(n)$ total time

Consider three cases below

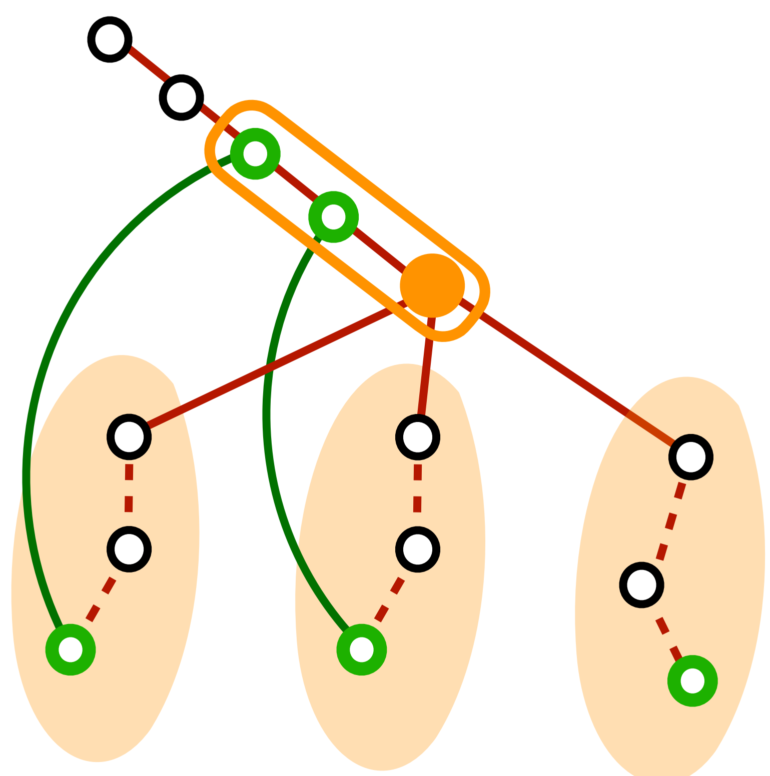


**A bunch of subtrees containing no special vertices**

**Apply fractional-cascade $O$(subtree-sizes + $\log n$) time**

# Highest ancestors in *O(n)* total time

Consider three cases below



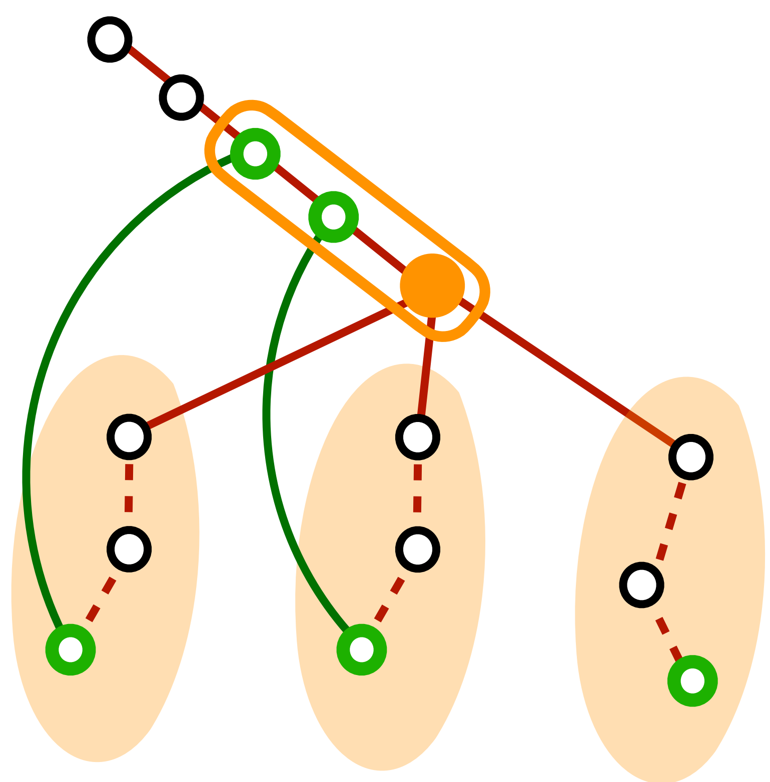
**A bunch of subtrees containing no special vertices**

**Apply fractional-cascade**
***O*(subtree-sizes + *log n*) time**

**Sum of subtree-sizes = *O(n)***

# Highest ancestors in *O(n)* total time

Consider three cases below



**A bunch of subtrees containing no special vertices**

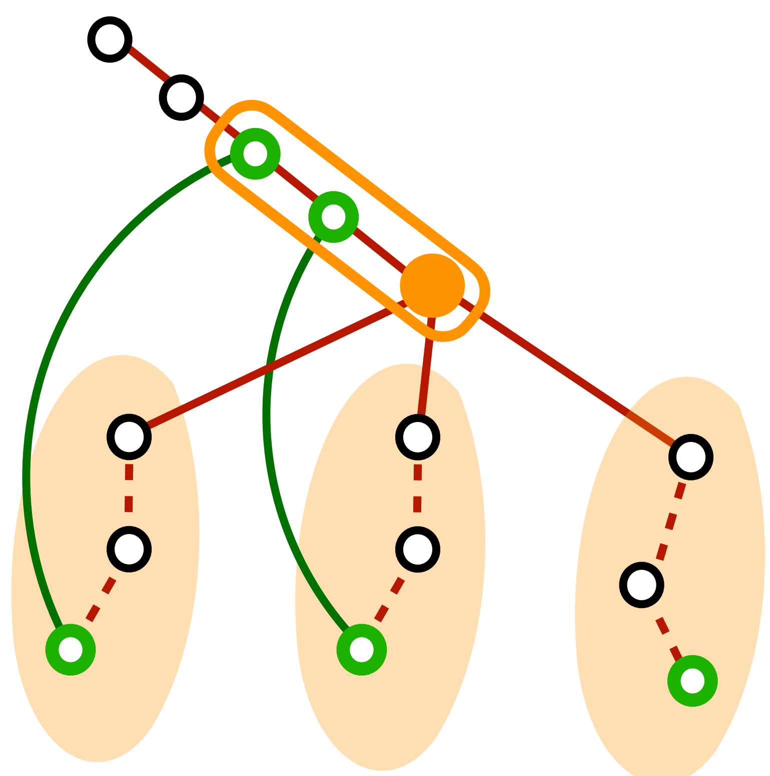
**Apply fractional-cascade**
***O*(subtree-sizes + *log n*) time**

**Sum of subtree-sizes = *O(n)***

**One can prove this happens at most *O(n / log n)* times, so total time becomes *O(n)***

# Highest ancestors in *O(n)* total time

Consider three cases below



**A bunch of subtrees containing no special vertices**

**Apply fractional-cascade *O*(subtree-sizes + *log n*) time**

**Sum of subtree-sizes = *O(n)***

**One can prove this happens at most *O(n / log n)* times, so total time becomes *O(n)***

**Summary: total time is *O(n)***

# Thanks