CS 381 – FALL 2019

Week 5.3, Monday, Sept 16

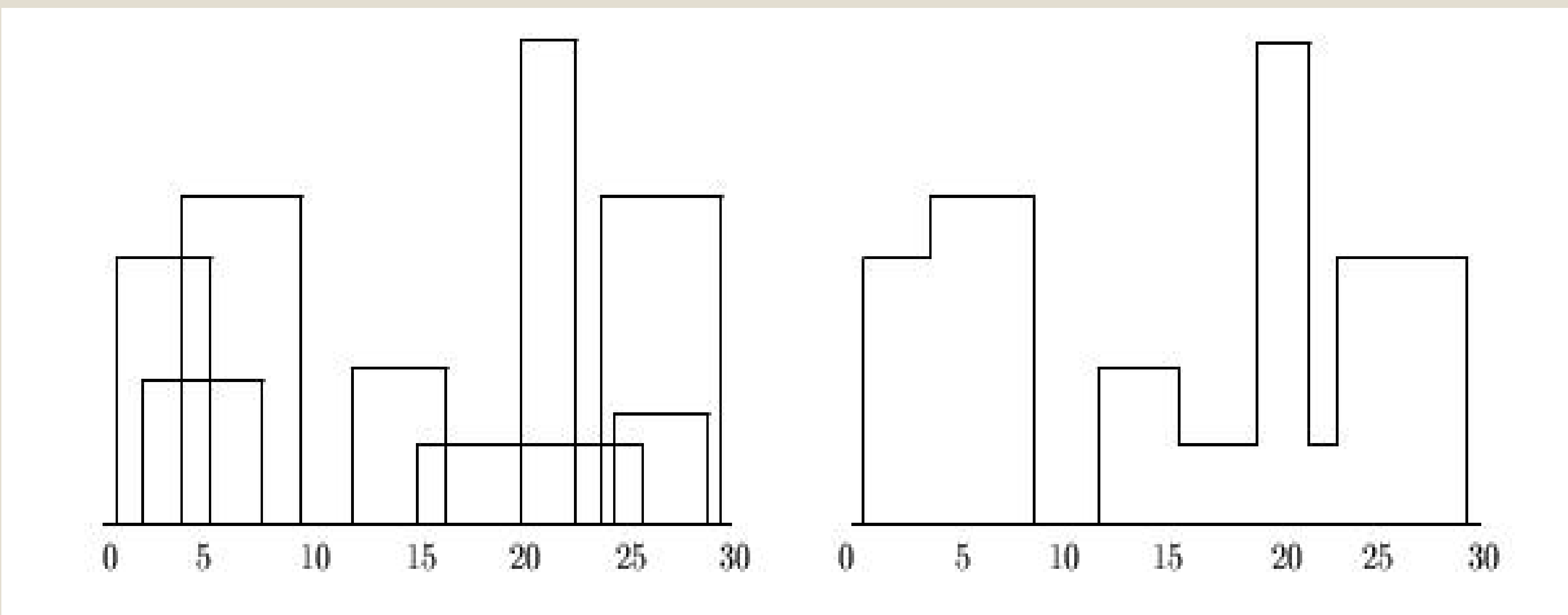
Guest Lecture (Prof. Hambrusch)
Homework 2 Due Tonight!
September 16th, 2019 @ 11:59PM on Gradescope

Homework 2 Reminders

- Must include collaborator/resource statement
  - No credit for solutions that don’t include CR statement
- Must type solutions
  - Only allowed to scan hand drawn graphs/diagrams
  - No credit if the entire solution is a scan of your handwritten homework
  - Expectation to use math notation $\sqrt{n}$ vs n^(1/2))
- Remember to select the appropriate pages for each problem on Gradescope

Skyline problem

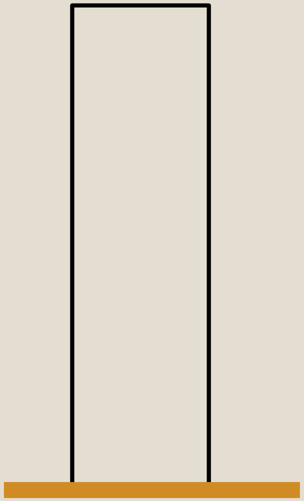
- Given are n "buildings" positioned on a common horizontal line. Building i is described by the triple (l_i, r_i, h_i), $1 \leq i \leq n$.
- The skyline is the union of the outline formed by the n buildings. Represented by coordinates forming the corners.

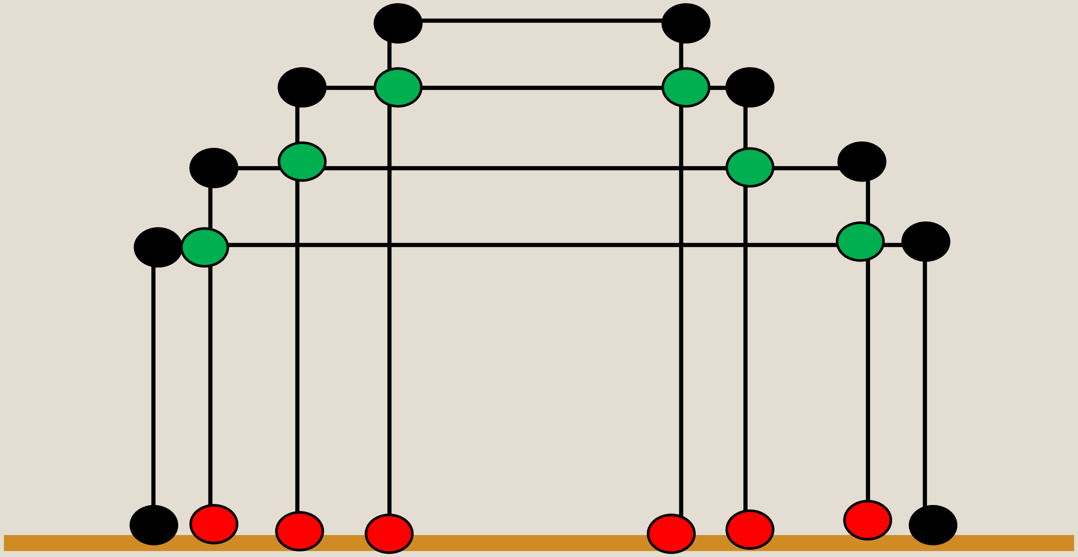


Towards a first solution …

How many corners can a skyline have?

- Minimum of 4 and maximum of 4n (**#new** ≤ **#hidden**)





Formalizing the Problem…

Building i is described by the triple (l_i, r_i, h_i), $1 \le i \le n$.

The skyline is represented by its corner coordinates

How many corners can a skyline have?

- Minimum of 4 and maximum of 4n

How should one maintain the skyline?

- Store corners by non-decreasing x-coordinates

Formalizing the Problem…

Input: List of n triples (l_i, r_i, h_i), $1 \leq i \leq n$ encoding the location of building i.

- Encodes:
  - Left corner $(l_i,0)$ of building i
  - Right corner $(r_i,0)$ of building i
  - Height h_i of building i

Output: List of (at most 4n) corner points (x_i, y_i) sorted by x-coordinate i.e., $x_1 < x_2 < \ldots$

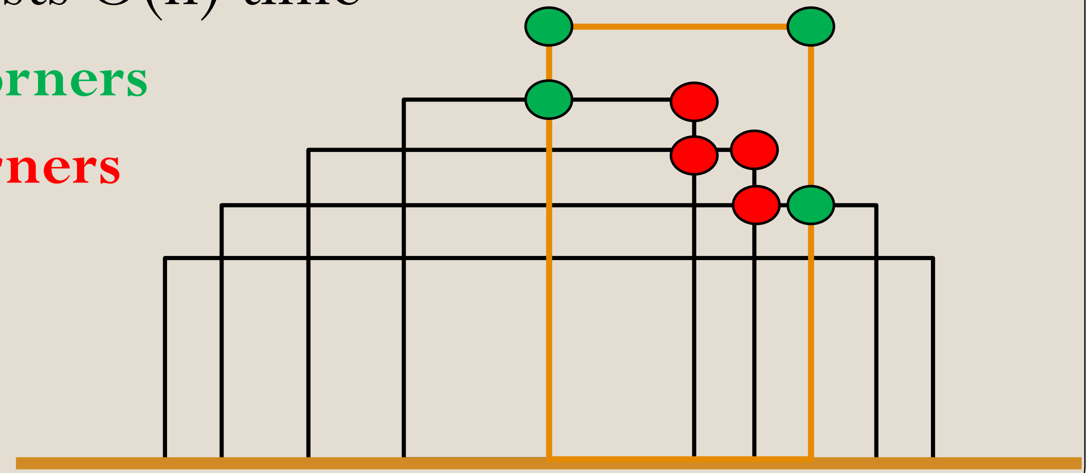
A first solution …

The skyline is represented by its corner coordinates.

Store corners by non-decreasing x-coordinates

Assume we build the skyline by add adding one building at a time

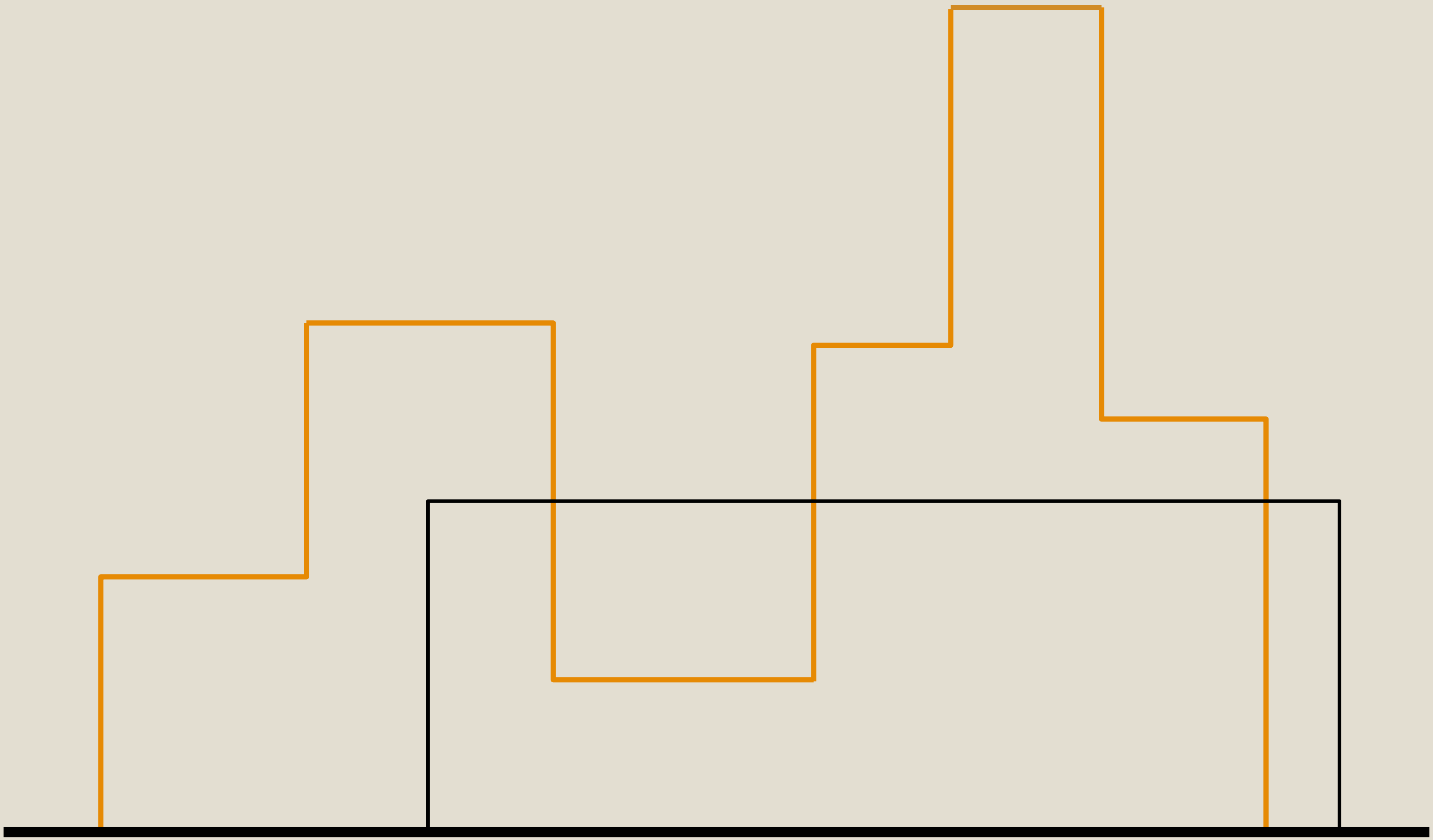
- Adding one building costs O(n) time
  - **Add (up to) 4 new corners**
  - **Remove (up to) 4 corners**
- Yields $O(n^2)$ algorithm



Can we do better than $O(n^2)$?

Can we work with the sorted skyline more effectively?

- If we use binary search, we need to store the skyline in an array
- If we use an array, we need to move elements in the array after inserting into the current skyline
- Worst case seems to still be n^2
- Consider other data structures?
  - Balanced search trees?
  - Structures for searching in 2 dimensions?



A divide and conquer solution

1. Split the n buildings into two sets B1 and B2 of n/2 building each (no need to sort the buildings).
2. Find the skyline S1 for set B1.
3. Find the skyline S2 for set B2.
4. Merge the two skylines S1 and S2.
   - Traverse S1 and S2 and pick up the piece belonging to the new skyline from one of them.

$T(n) = 2T(n/2) + cn$ and $T(1)=1$

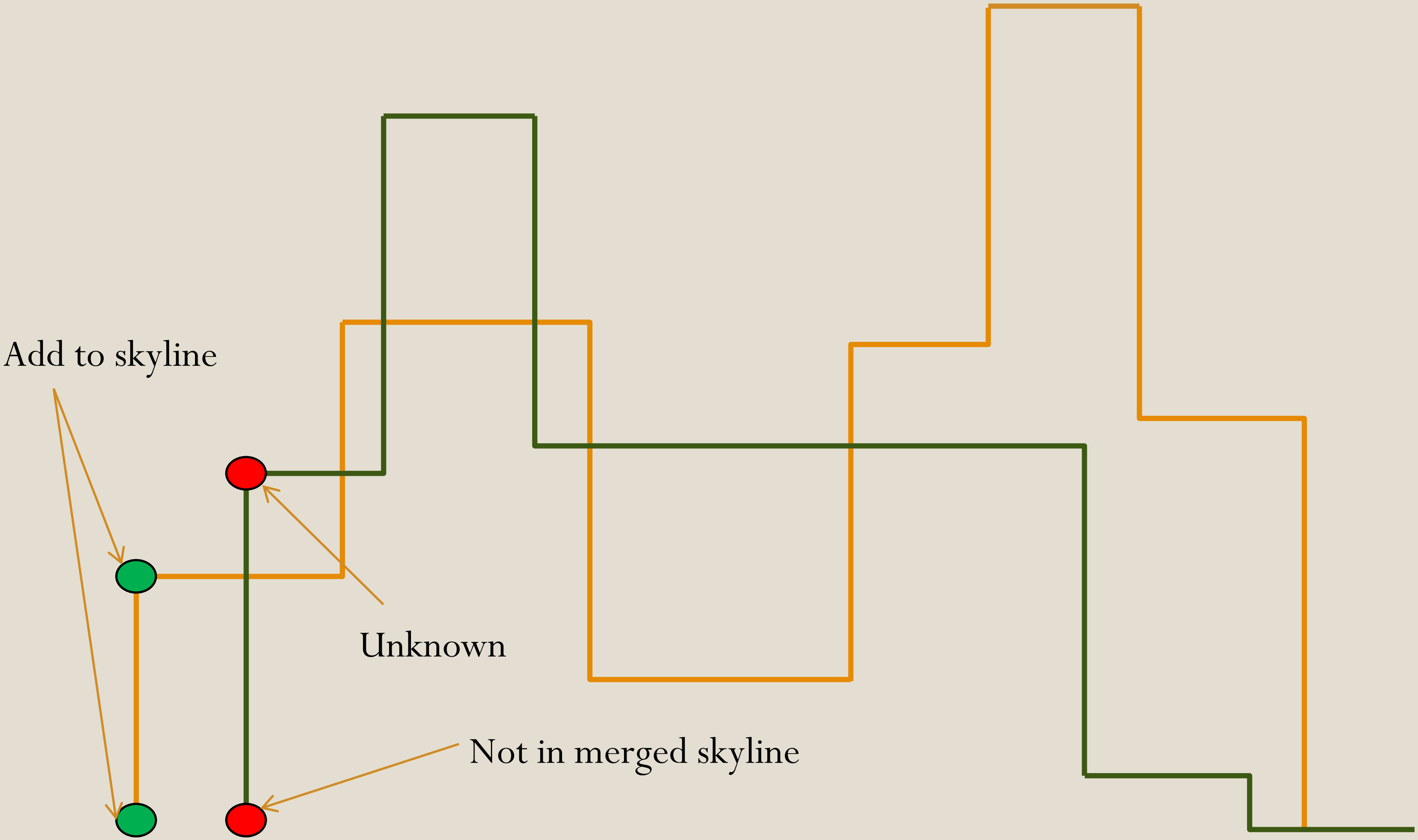
$T(n) = O(n \log n)$

Merge Skylines in Linear Time

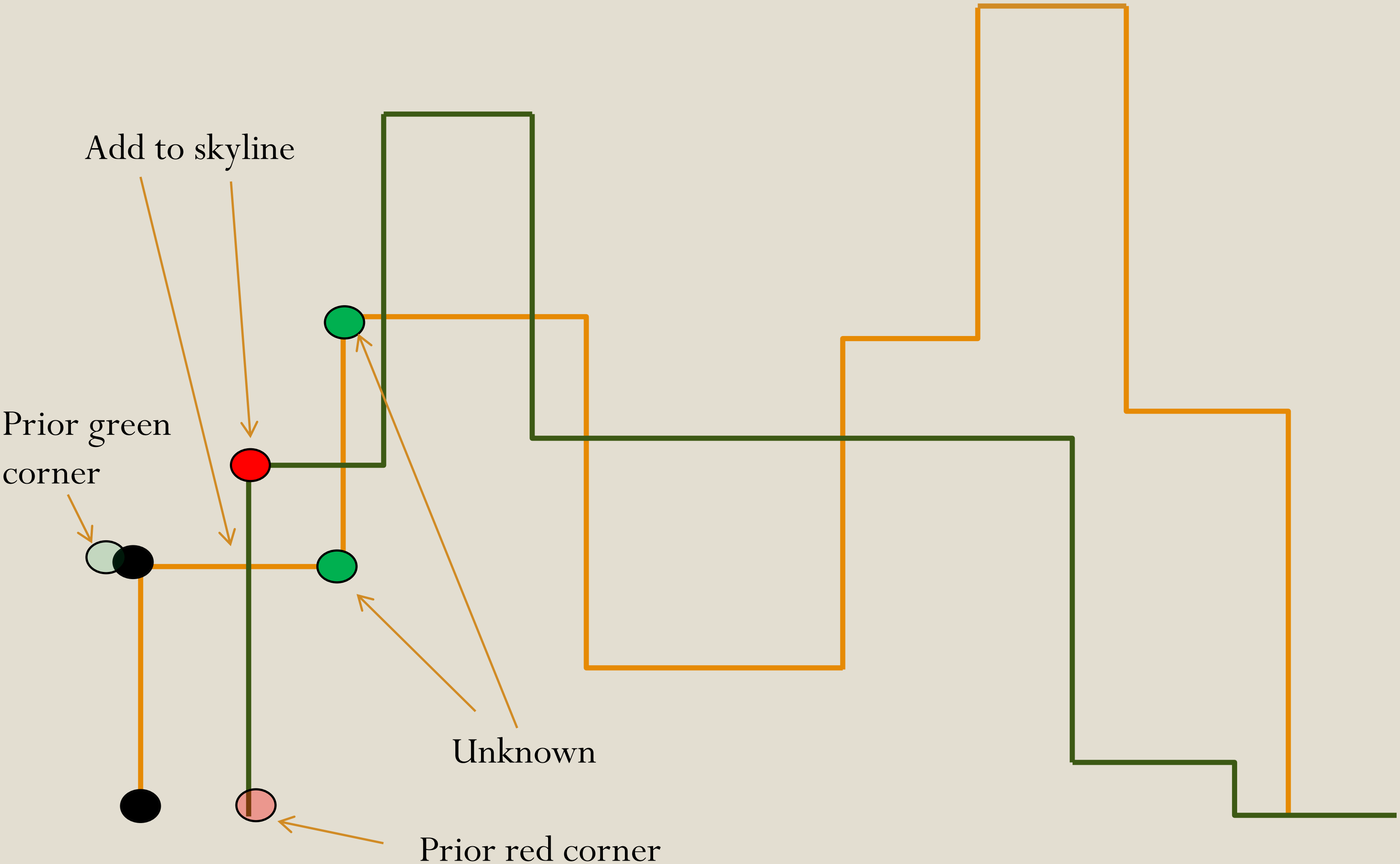
Skyline A: $(x_{1,A}, y_{1,A}), (x_{2,A}, y_{2,A}) \ldots$

Skyline B: $(x_{1,B}, y_{1,B}), (x_{2,A}, y_{2,A}) \ldots$

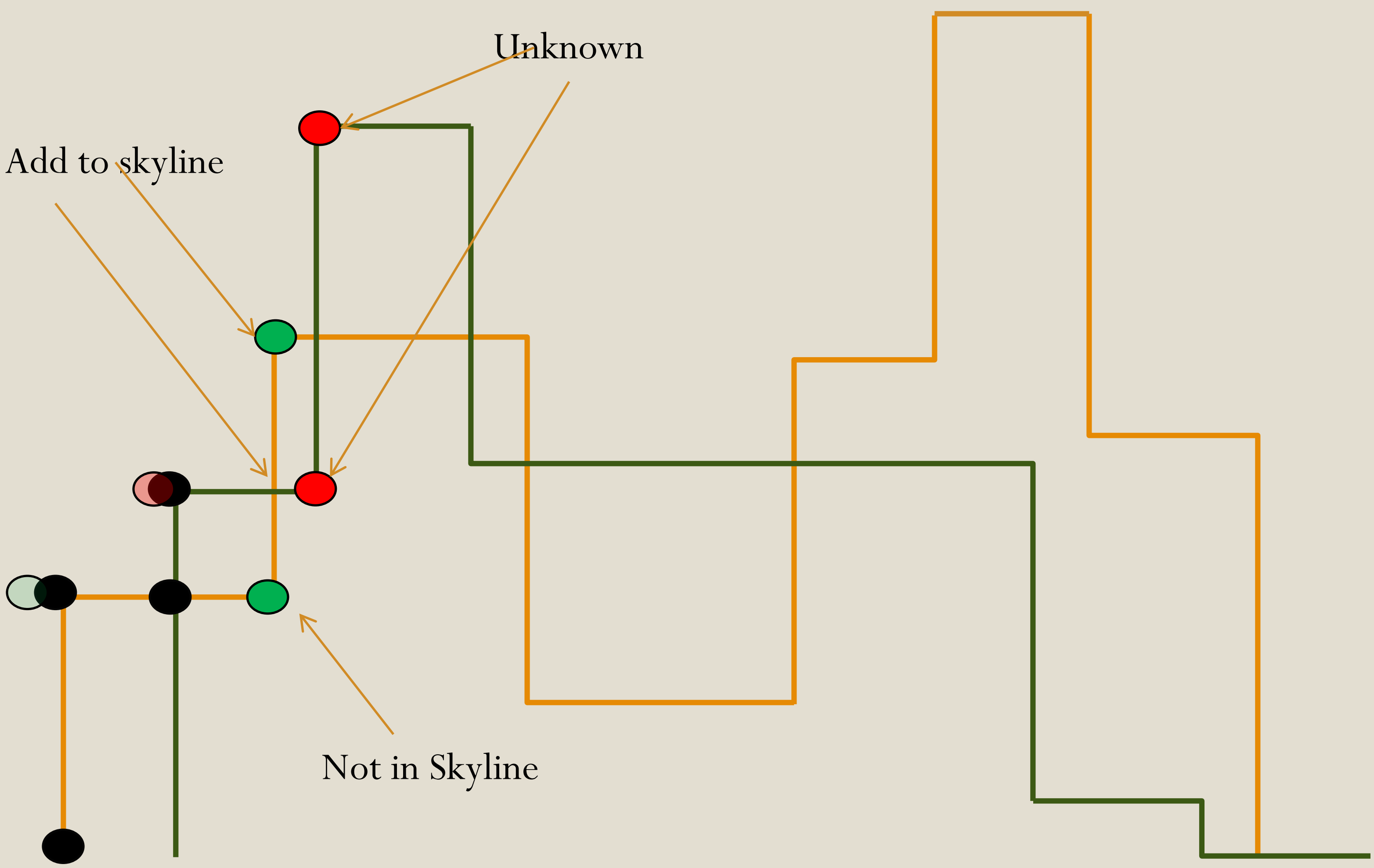
Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



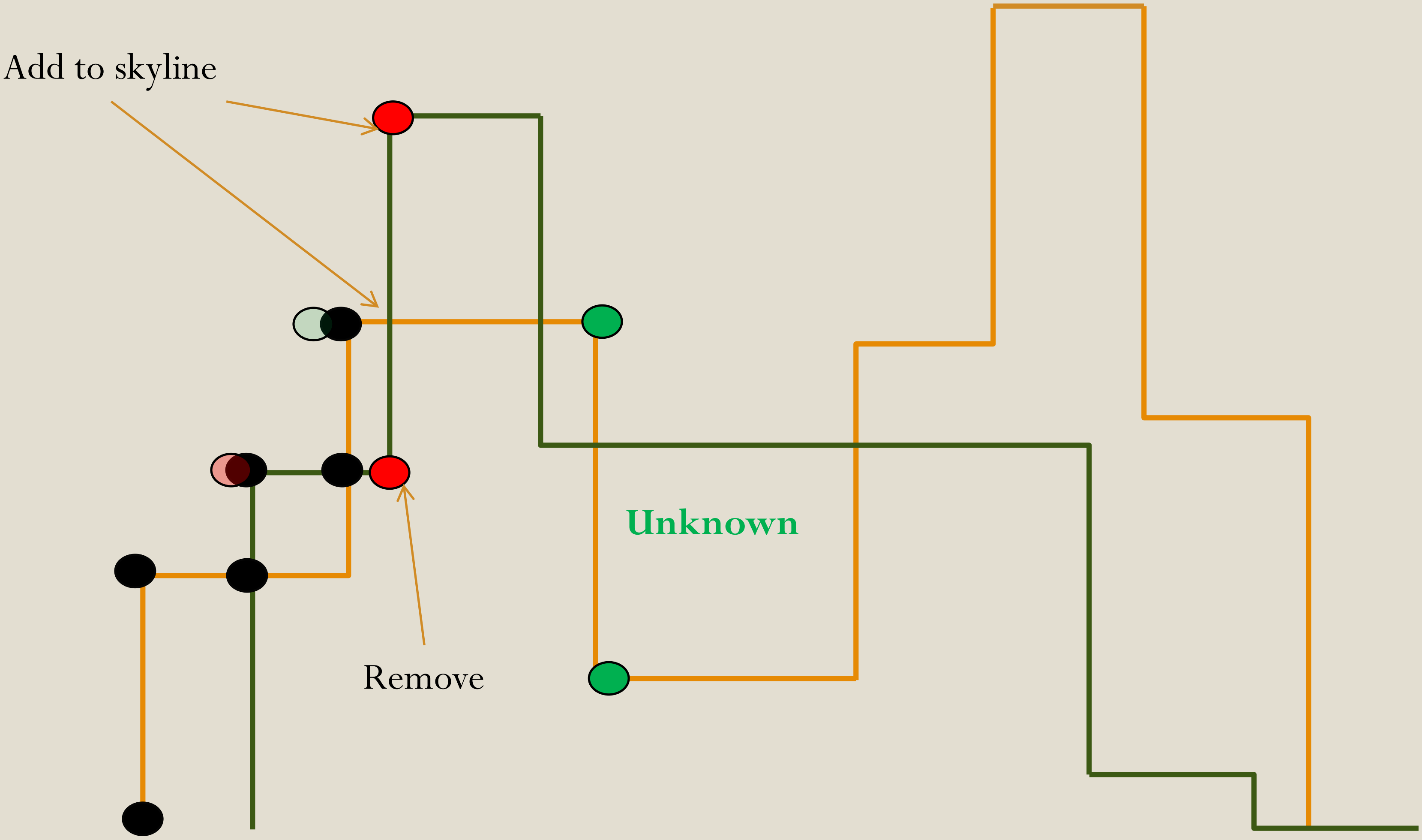
Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



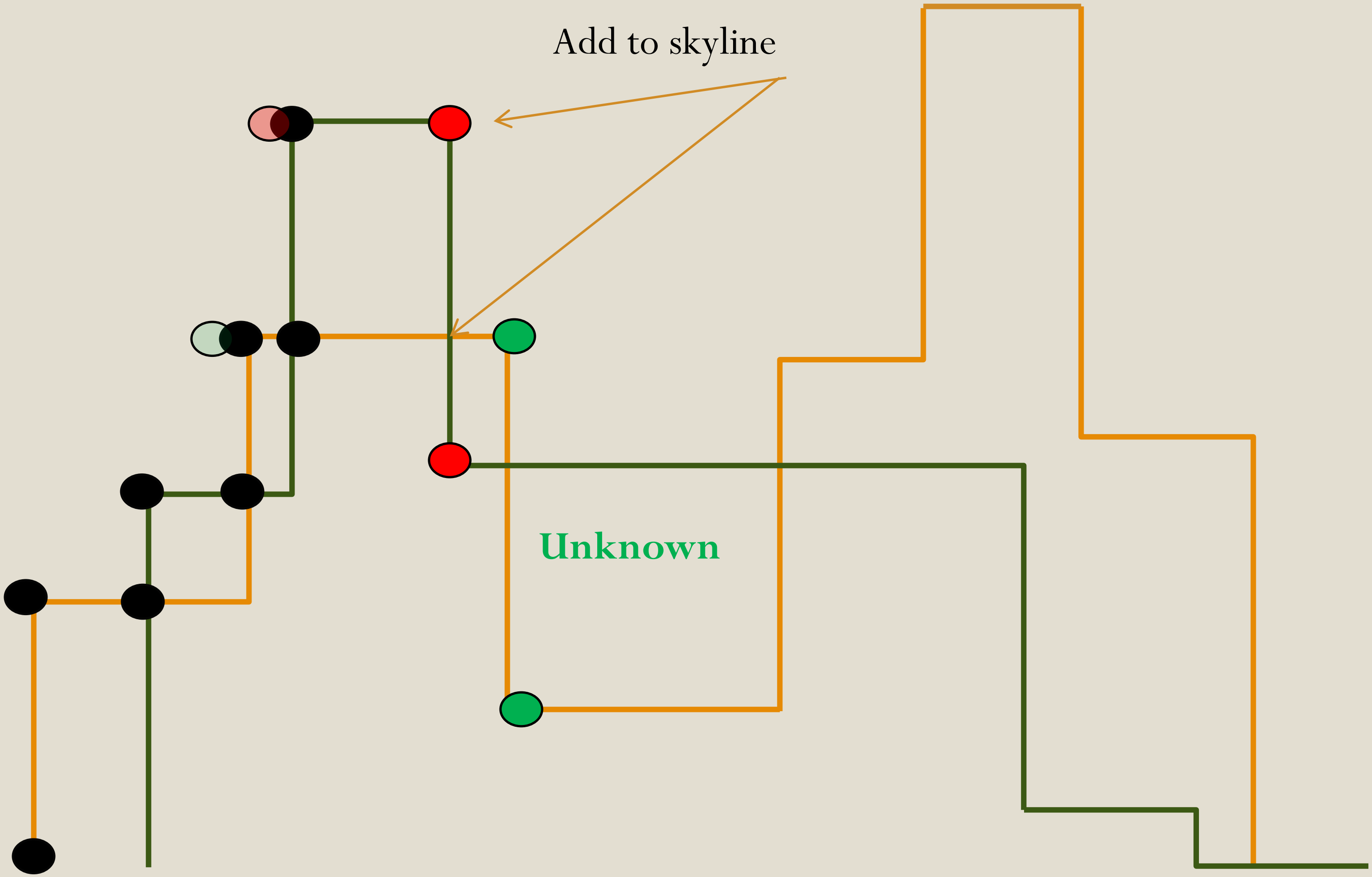
Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



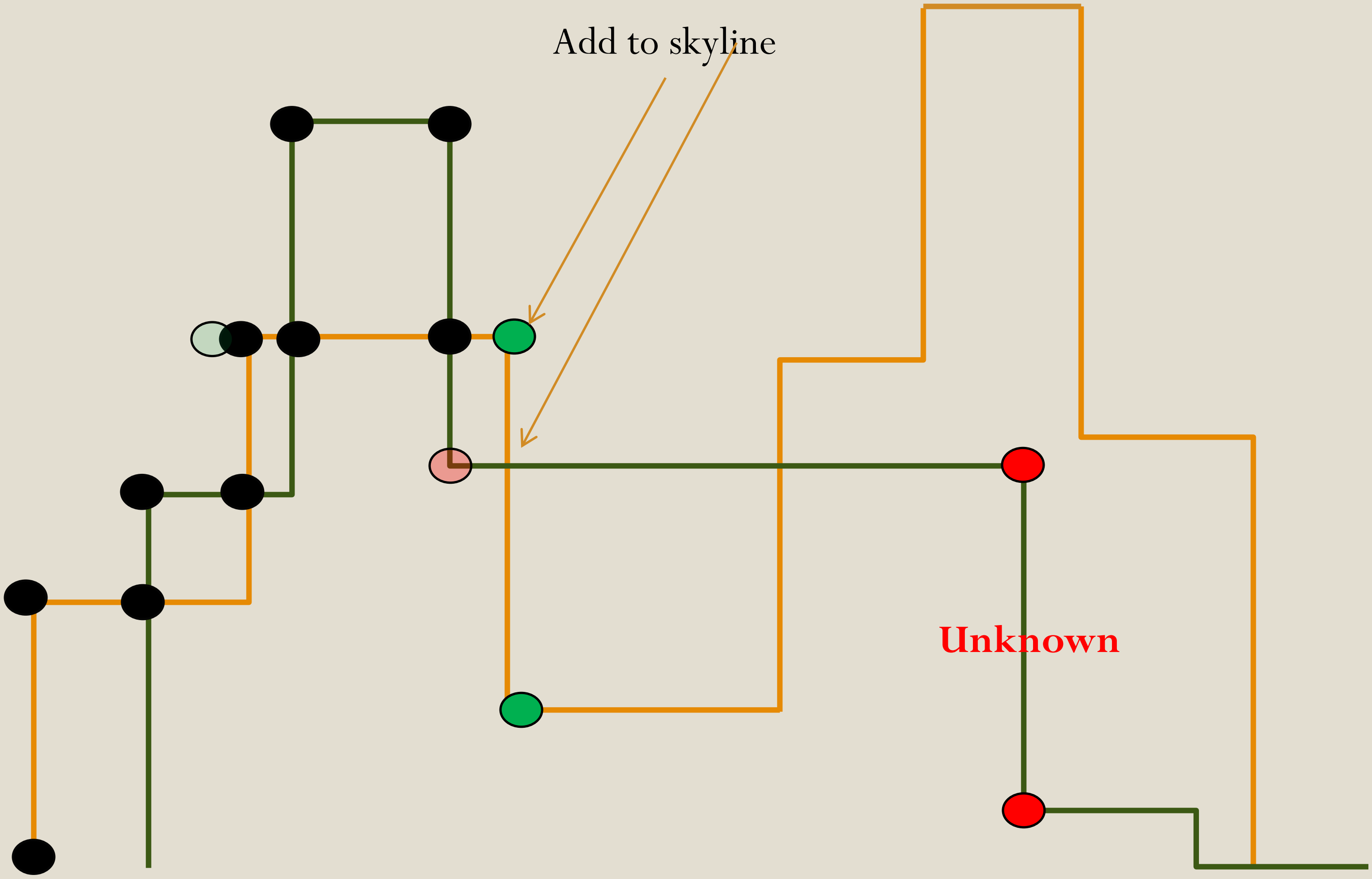
Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



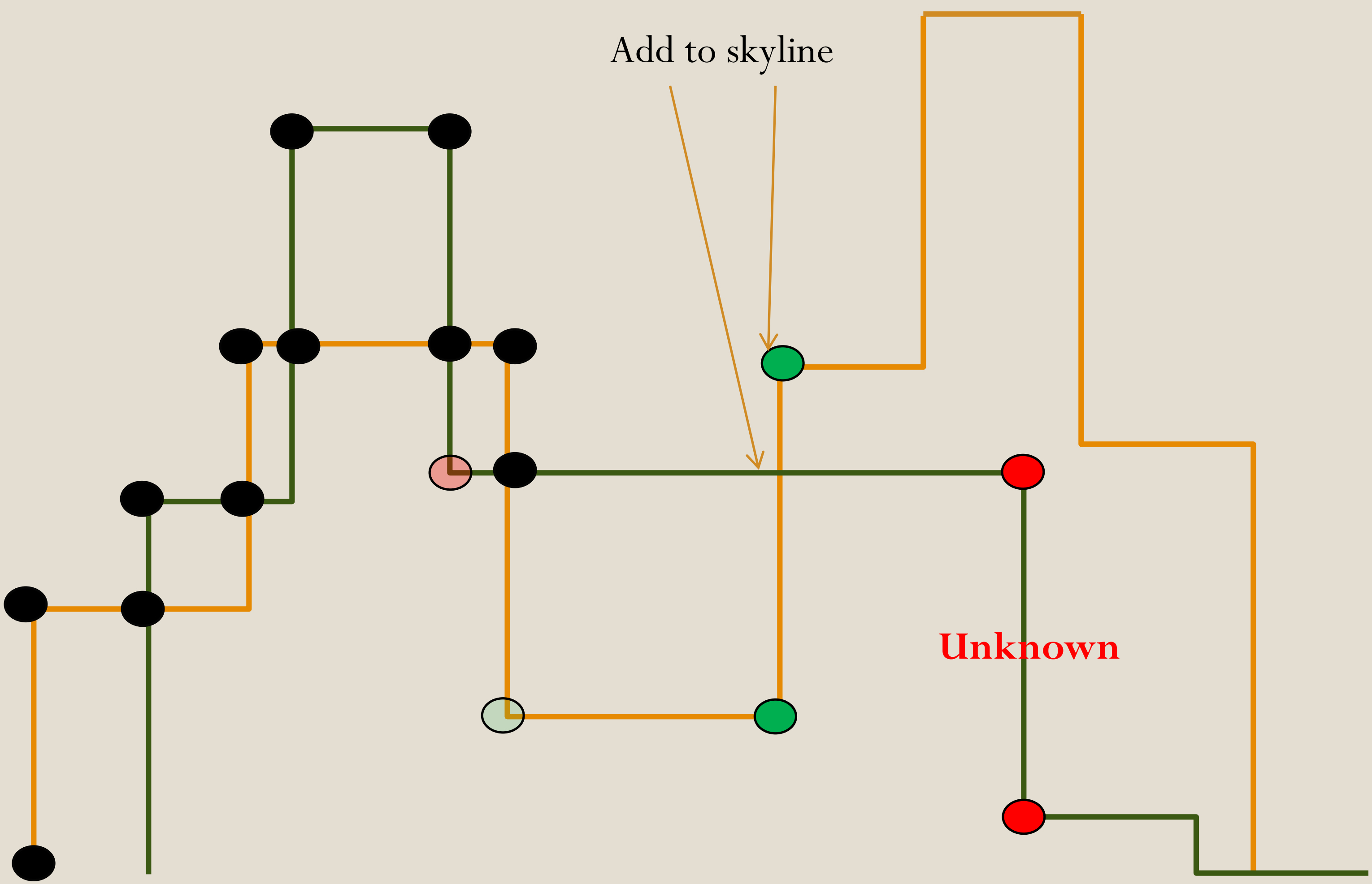
Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



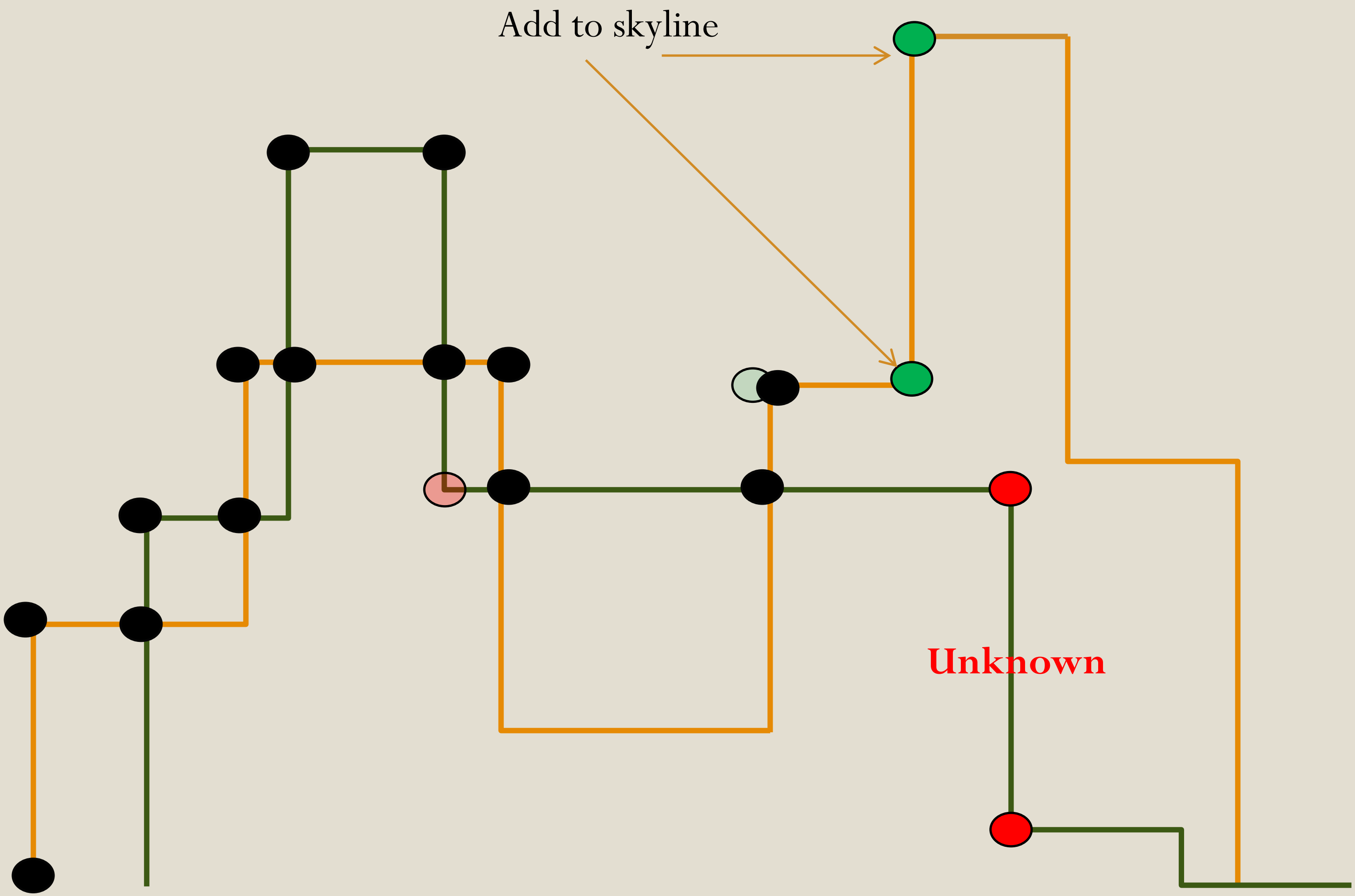
Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



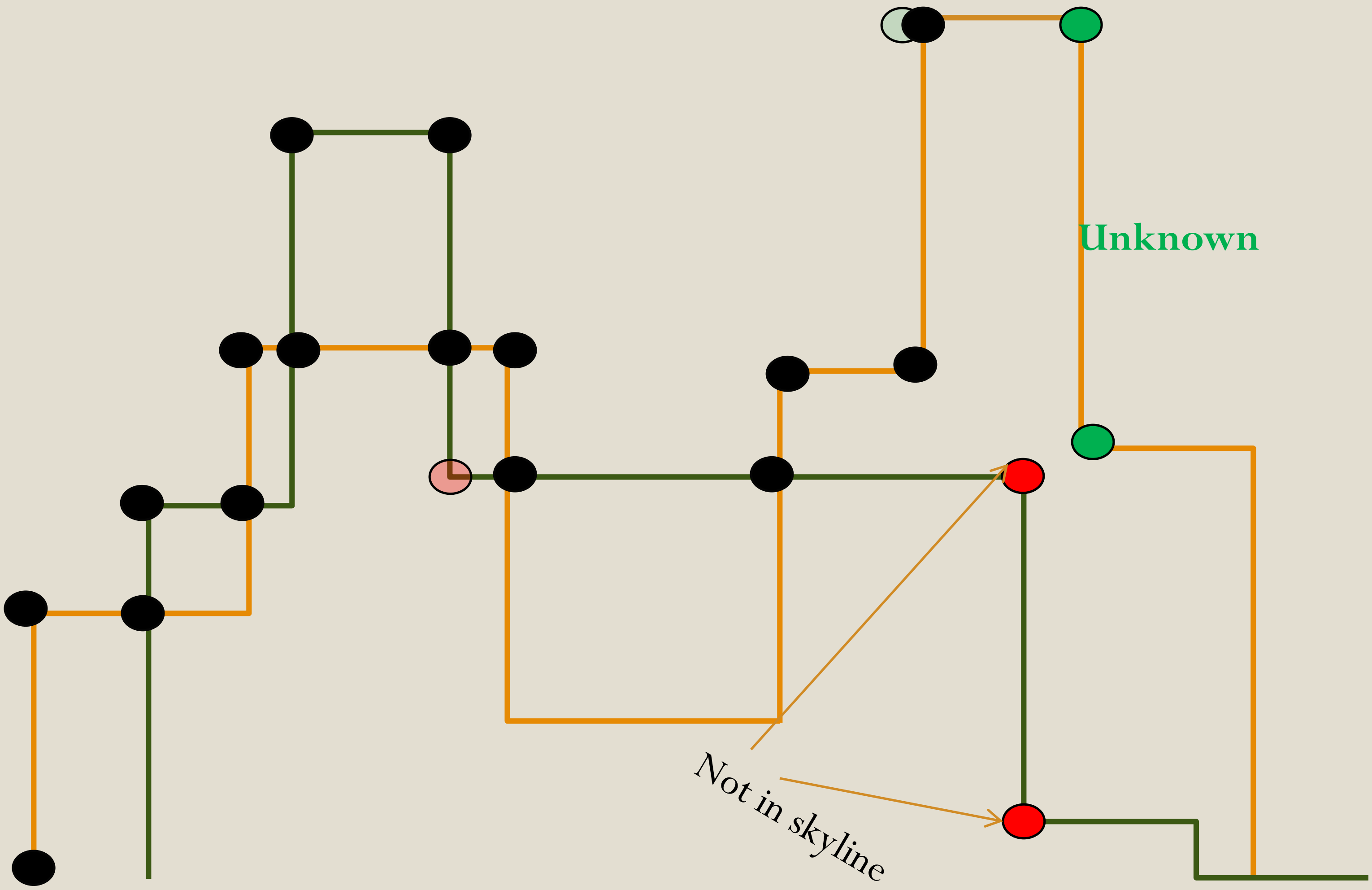
Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



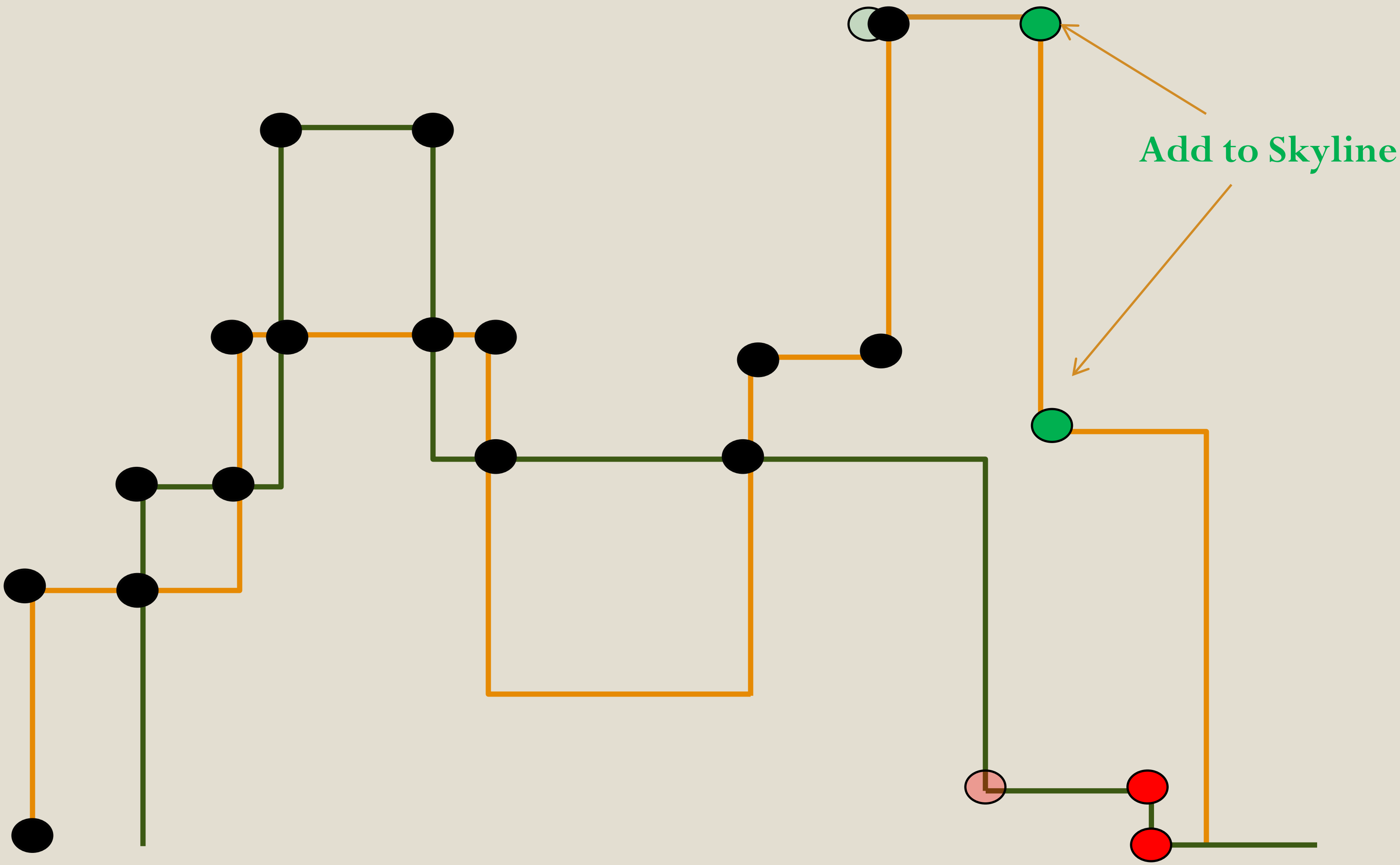
Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



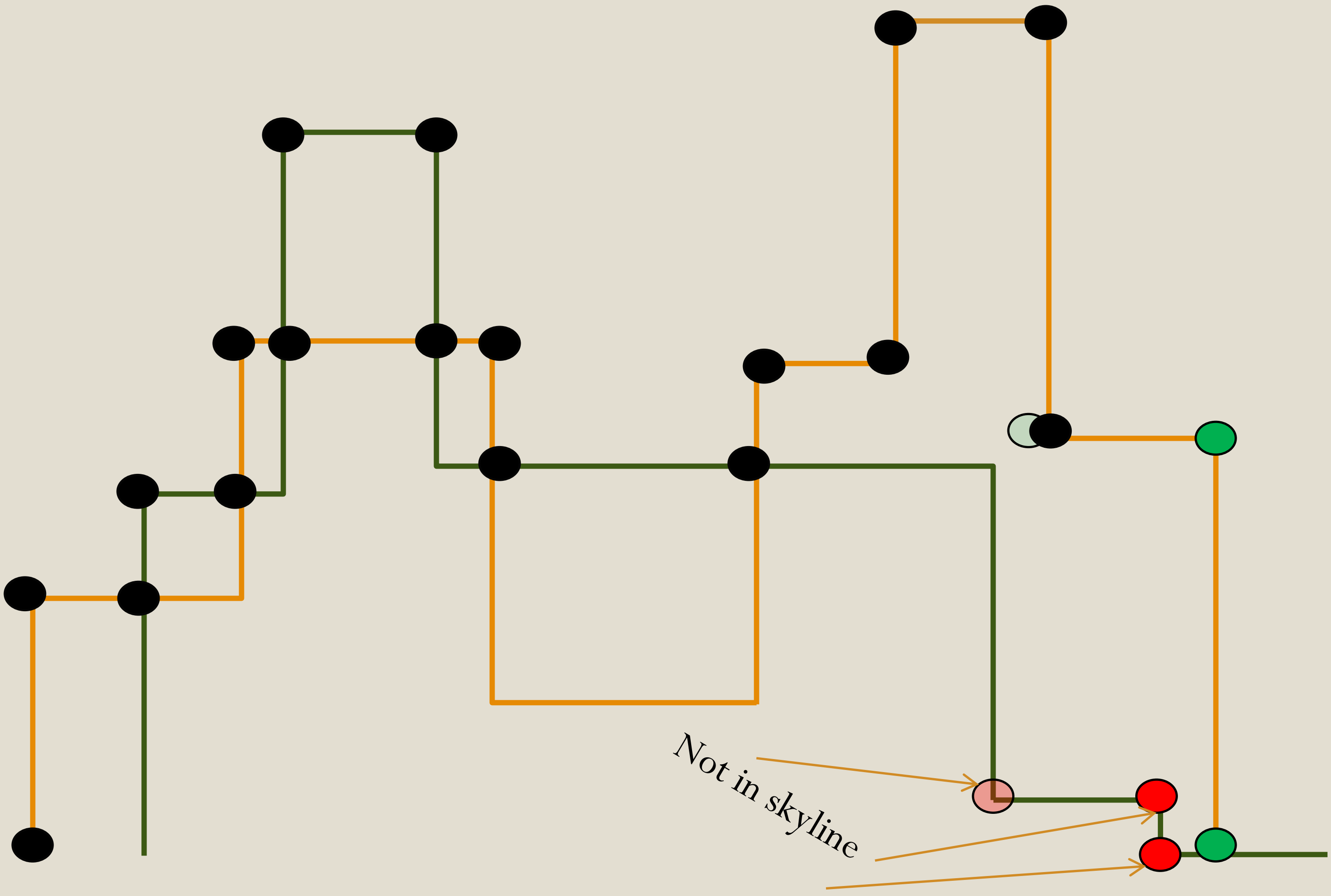
Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



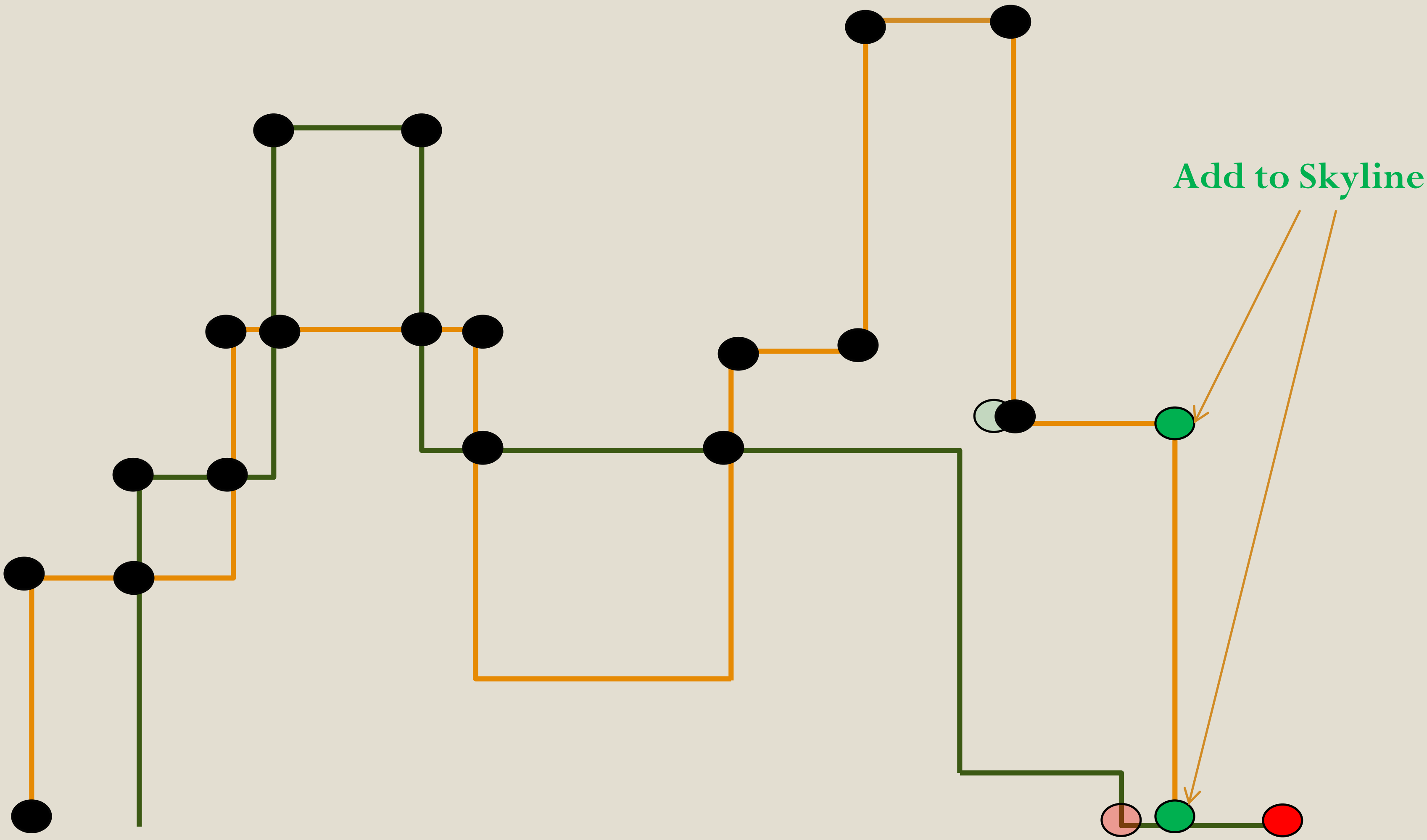
Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



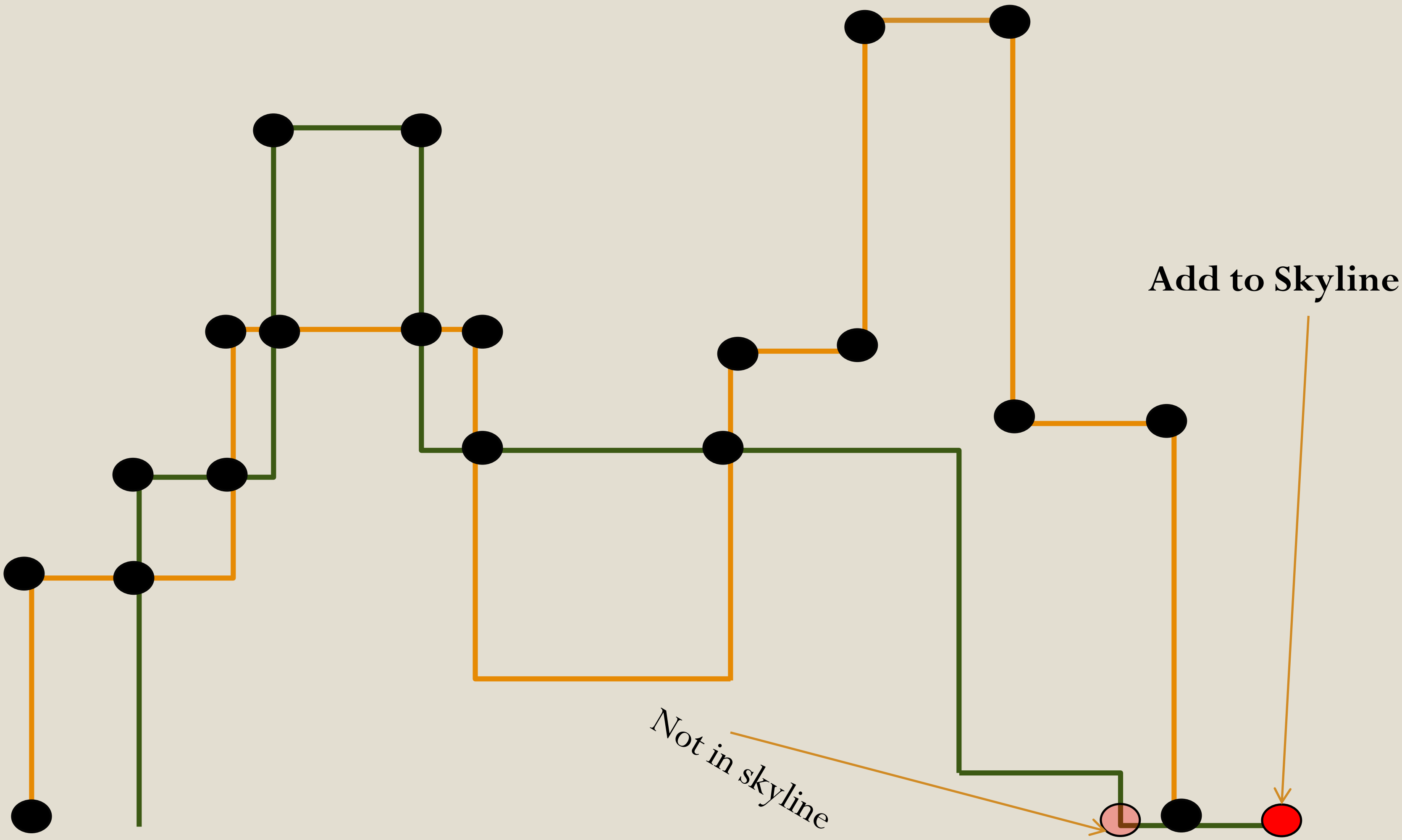
Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



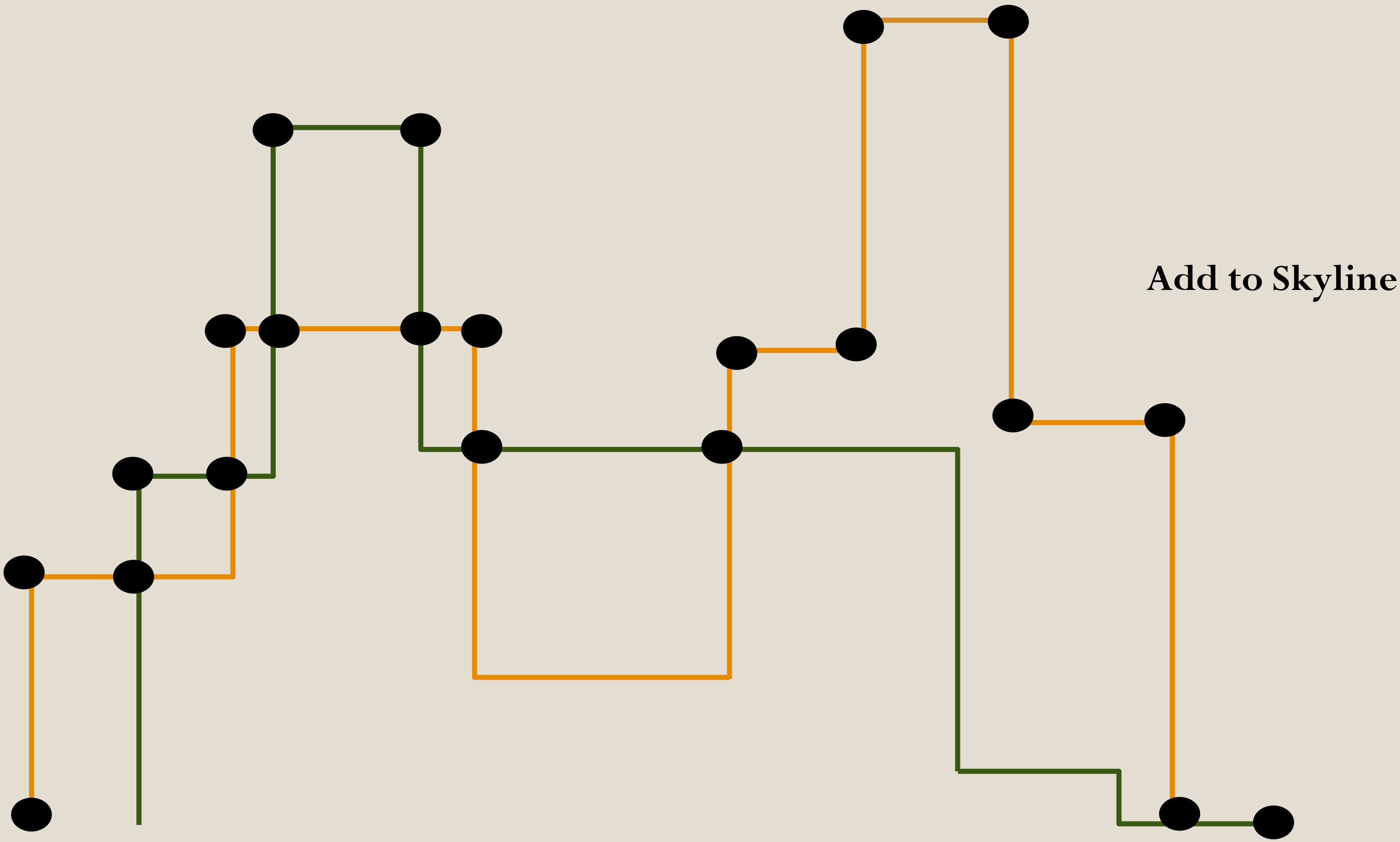
Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.



Traverse skyline A and B (left to right) and pick up the piece belonging to the new skyline from one of them.

Merge Skylines in Linear Time

- Iterate through skyline A and B in order
- Keep track of current/previous corners from skyline A and B
- Compute new corners (intersections), and decide which corners to include/exclude
- Requires O(1) work in each iteration