Holm Forests

Outline for Today

- ***Recap from Last Time***
  - Reviewing Euler tour trees and their augmentations.
- ***Why Fully Dynamic Connectivity is Hard***
  - Seeing how this differs from the forest case.
- ***Localizing Edges***
  - And bringing in some CS161 topics in clever ways.
- ***"Blame It On The Little Guy"***
  - A very creative way to decrease runtime costs.
- ***The Holm Forest***
  - A brilliant way to solve dynamic connectivity.

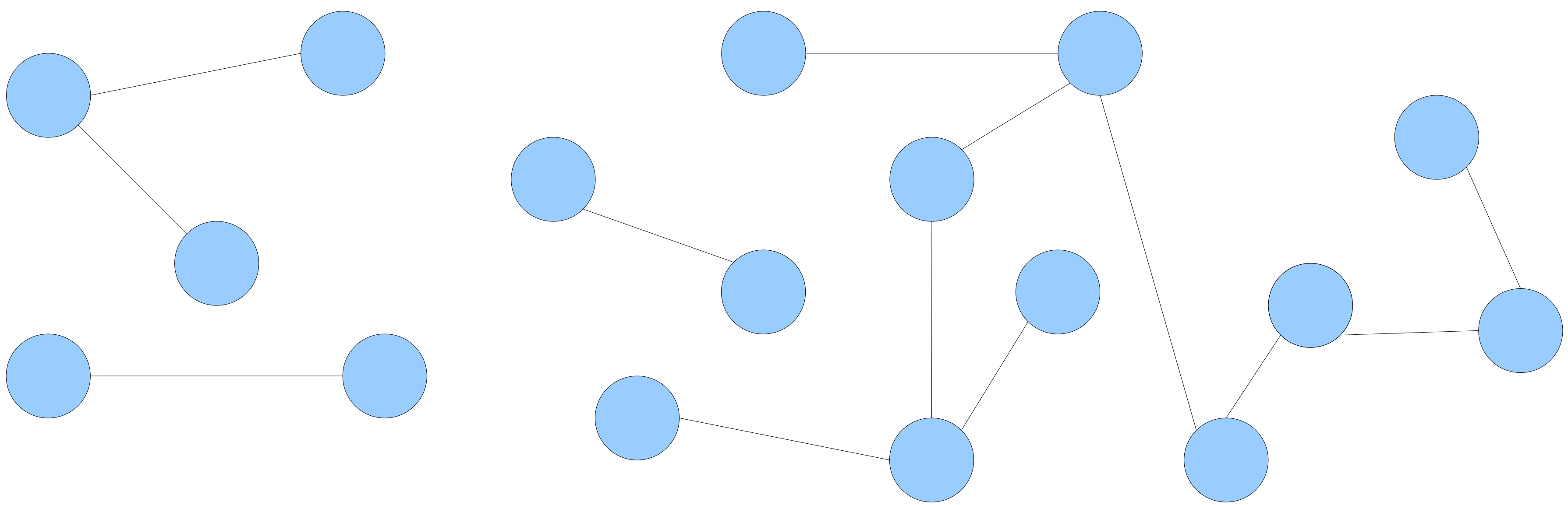
Recap from Last Time

Dynamic Connectivity in Forests

- Consider the following special-case of the dynamic connectivity problem:

Maintain an undirected ***forest*** *F* so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- Each deleted edge splits a tree in two; each added edge joins two trees and never closes a cycle.

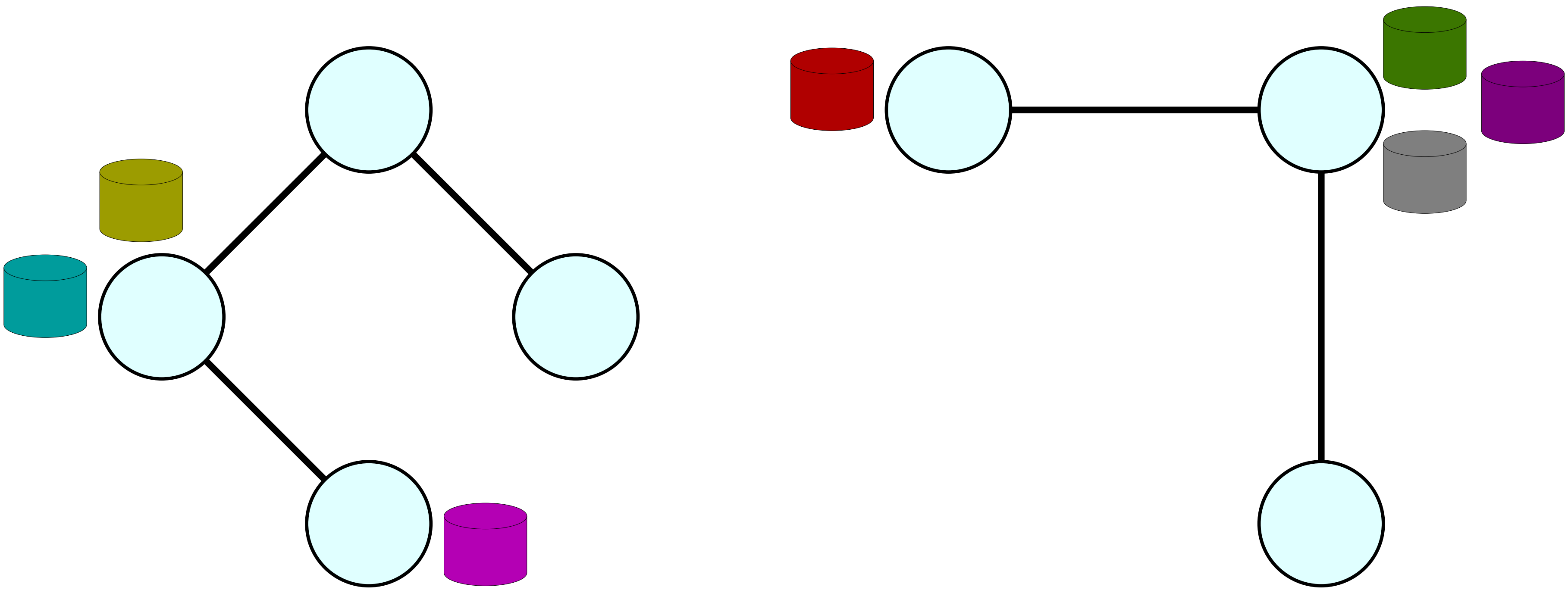


Euler Tour Trees

- The ***Euler tour tree*** data structure solves dynamic connectivity in forests with these (amortized) costs:
  - ***are-connected***: O(log n)
  - ***link***: O(log n)
  - ***cut***: O(log n)
- These bounds can be made worst-case efficient using different types of balanced BSTs instead of splay trees.

Extending Euler Tour Trees

- Euler tour trees can be augmented to aggregate information about the trees in the forest. With the right augmentations, we can support the following operations in (amortized) O(log n) time each.
  - ***size***(x), which returns the number of nodes in x's tree.
  - ***add-packet***(x, p), which attaches packet p to node x; and
  - ***remove-packet***(x), which removes and returns some packet reachable from x, chosen arbitrarily from all the options.



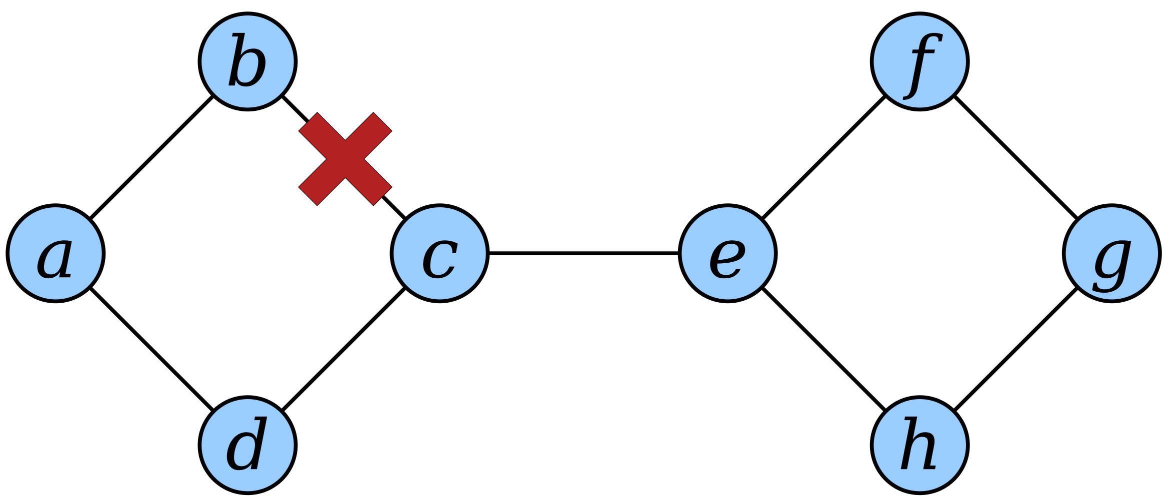
New Stuff!

Goal: Solve dynamic connectivity on arbitrary undirected graphs.

Why is Fully-Dynamic Connectivity Hard?

Can We Use Euler Tours?

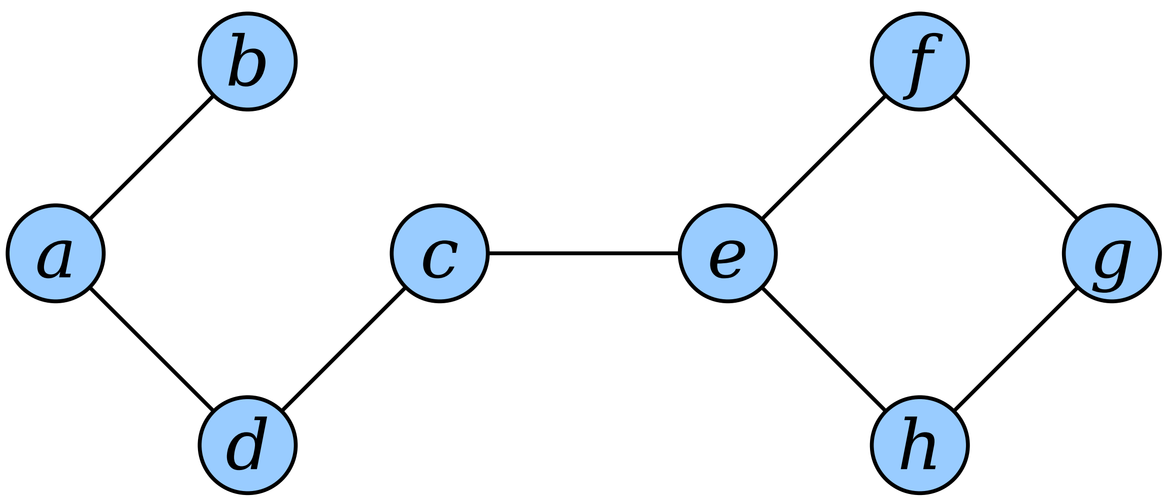
- In the case of maintaining a forest, we represented each tree as an Euler tour.
- Can we do something like that for general graphs?
- ***Problem:*** While we can form Euler tours in the general case, the behavior during a cut depends on whether the cut disconnects the graph.



ab bc cd da ad dc ce ef fg gh he eh hg gf fe ec cb ba

Can We Use Euler Tours?

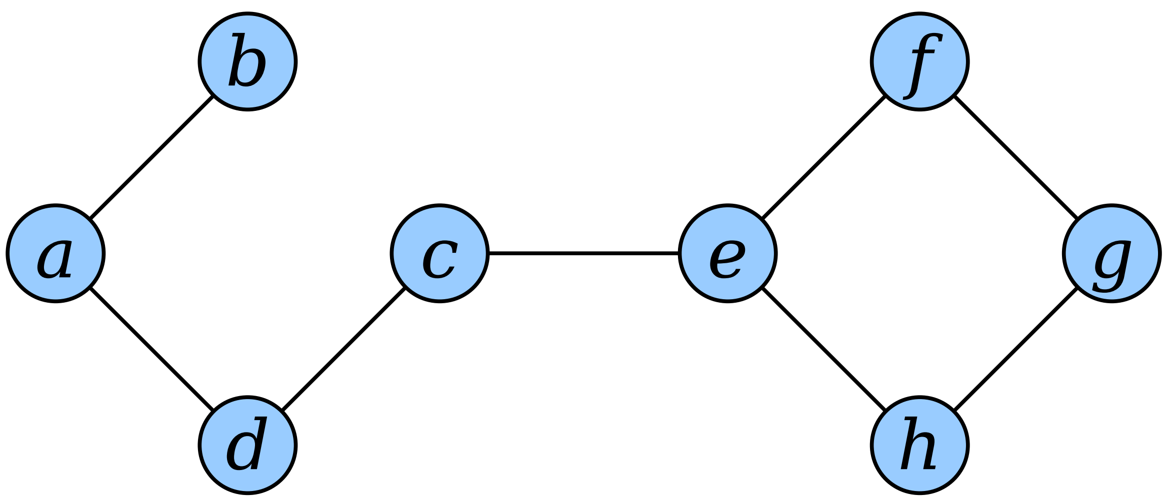
- In the case of maintaining a forest, we represented each tree as an Euler tour.
- Can we do something like that for general graphs?
- ***Problem:*** While we can form Euler tours in the general case, the behavior during a cut depends on whether the cut disconnects the graph.



ab cd da ad dc ce ef fg gh he eh hg gf fe ec ba

Can We Use Euler Tours?

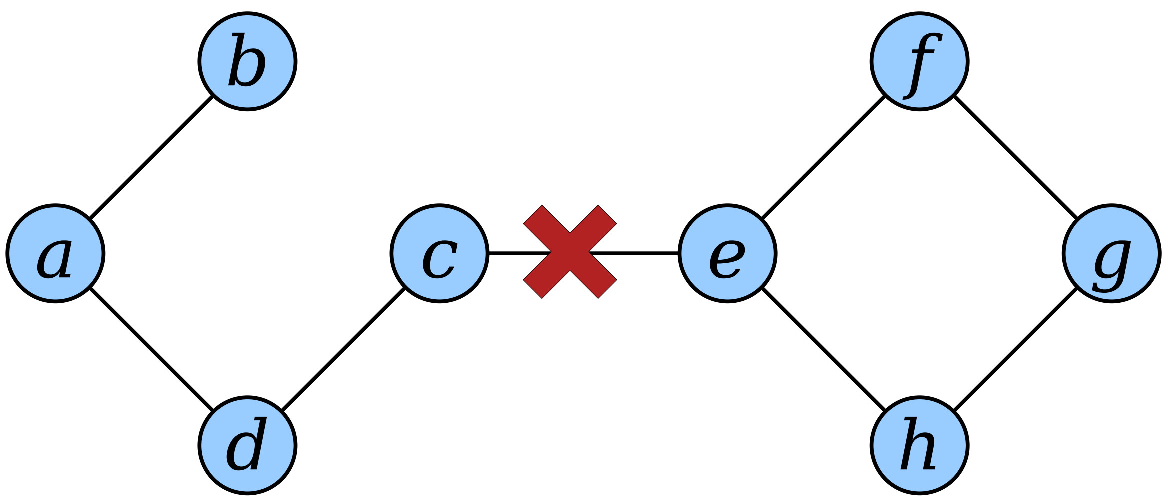
- In the case of maintaining a forest, we represented each tree as an Euler tour.
- Can we do something like that for general graphs?
- ***Problem:*** While we can form Euler tours in the general case, the behavior during a cut depends on whether the cut disconnects the graph.



ab ba ad dc ce ef fg gh he eh hg gf fe ec cd da

Can We Use Euler Tours?

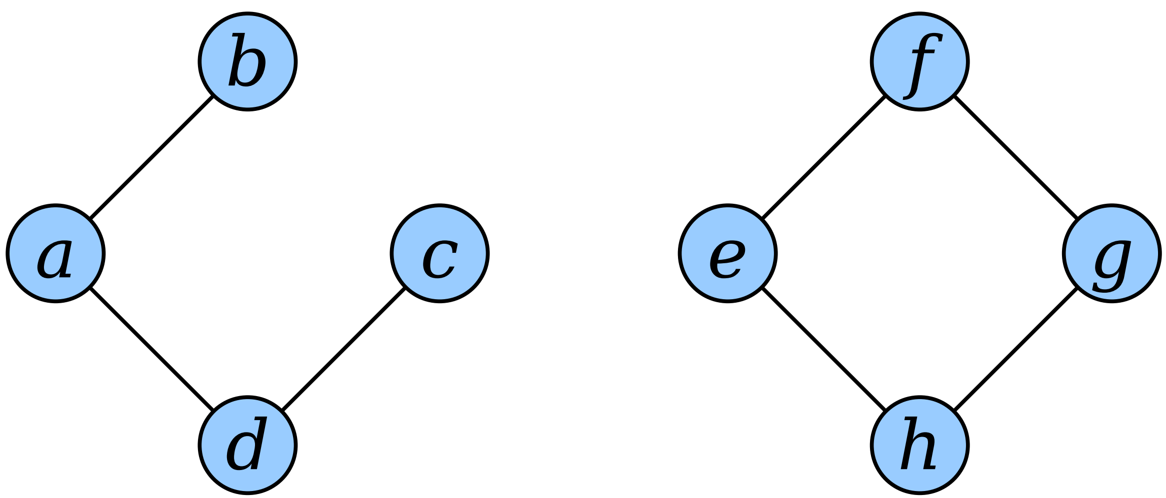
- In the case of maintaining a forest, we represented each tree as an Euler tour.
- Can we do something like that for general graphs?
- ***Problem:*** While we can form Euler tours in the general case, the behavior during a cut depends on whether the cut disconnects the graph.



ab ba ad dc ce ef fg gh he eh hg gf fe ec cd da

Can We Use Euler Tours?

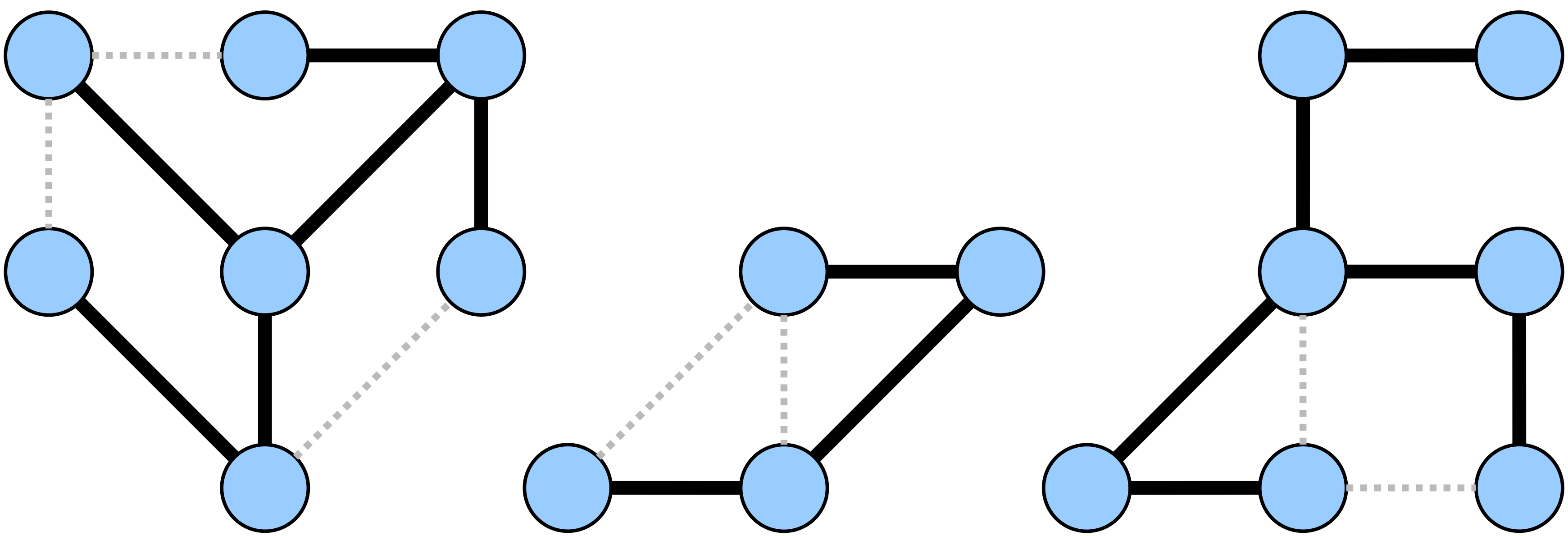
- In the case of maintaining a forest, we represented each tree as an Euler tour.
- Can we do something like that for general graphs?
- ***Problem:*** While we can form Euler tours in the general case, the behavior during a cut depends on whether the cut disconnects the graph.



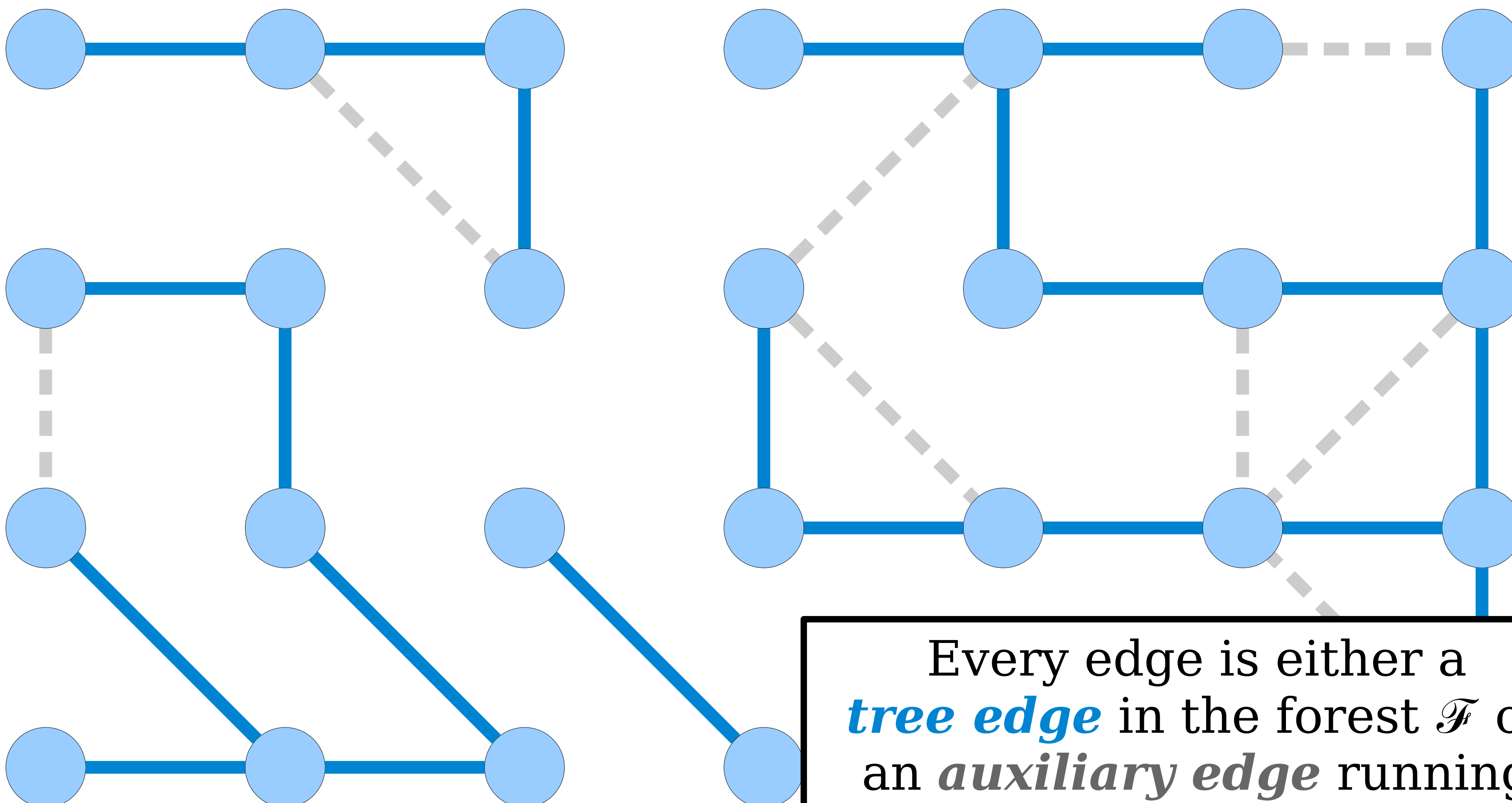
ab ba ad dc cd da ***ef fg gh he eh hg gf fe***

Can We Use Forests?

- We already have a solution for dynamic connectivity that works for forests. Can we adapt that to work for general graphs?
- ***Key Insight:*** If all we care about is connectivity, we just need to maintain a spanning forest of the graph.

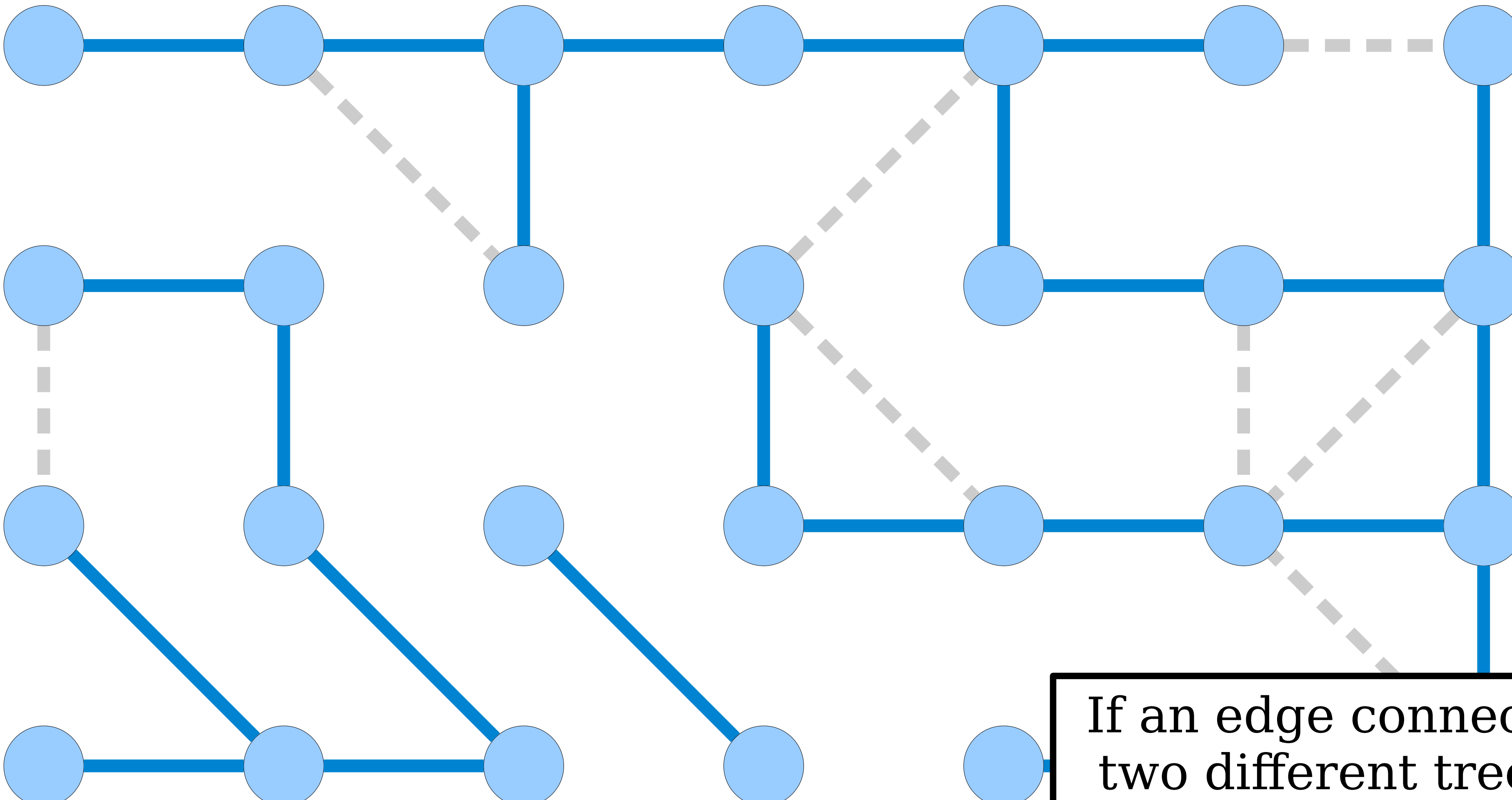


Maintaining a Forest



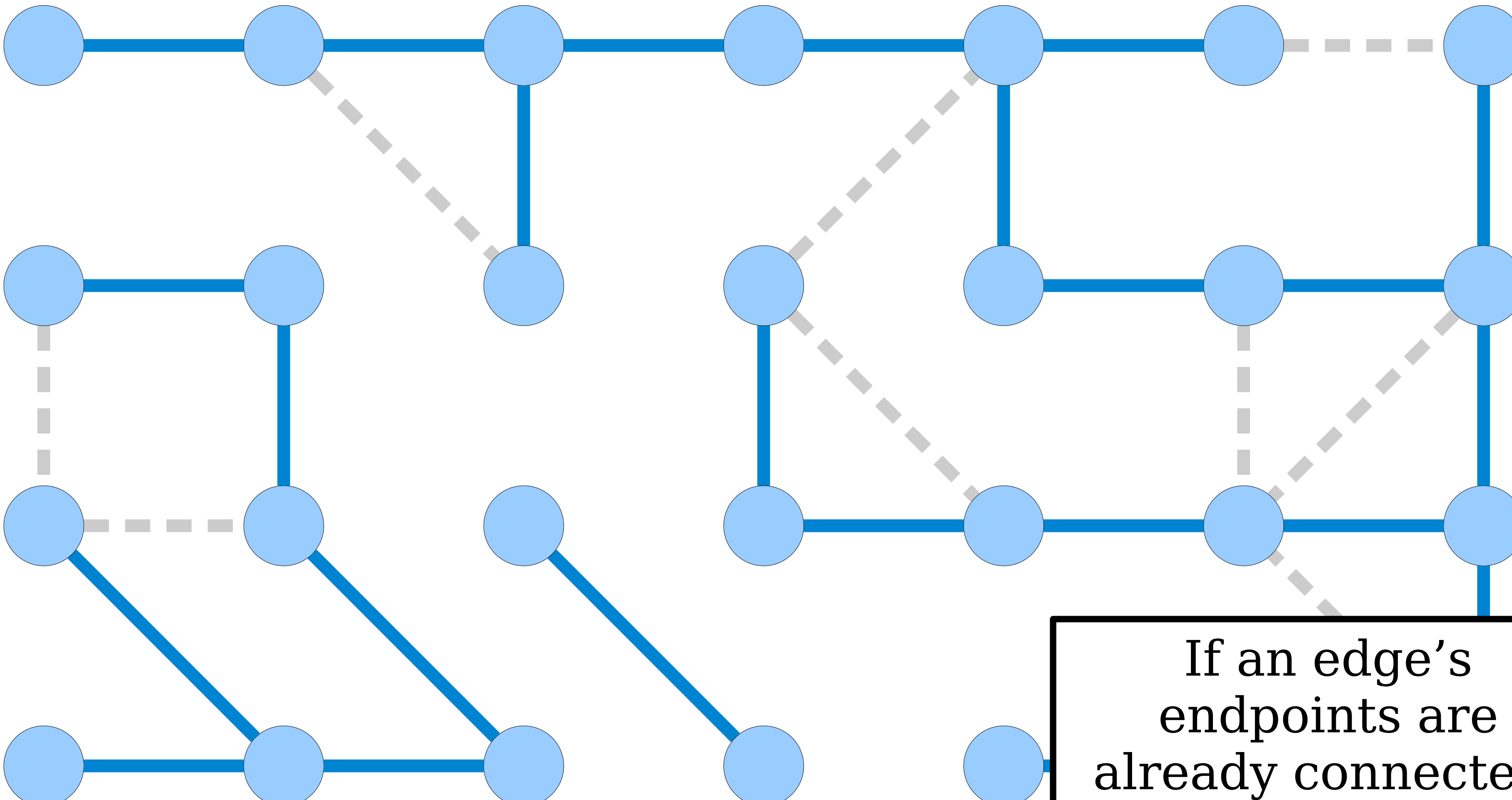
Every edge is either a ***tree edge*** in the forest $\mathscr{F}$ or an ***auxiliary edge*** running between two nodes in the same tree in $\mathscr{F}$.

Maintaining a Forest



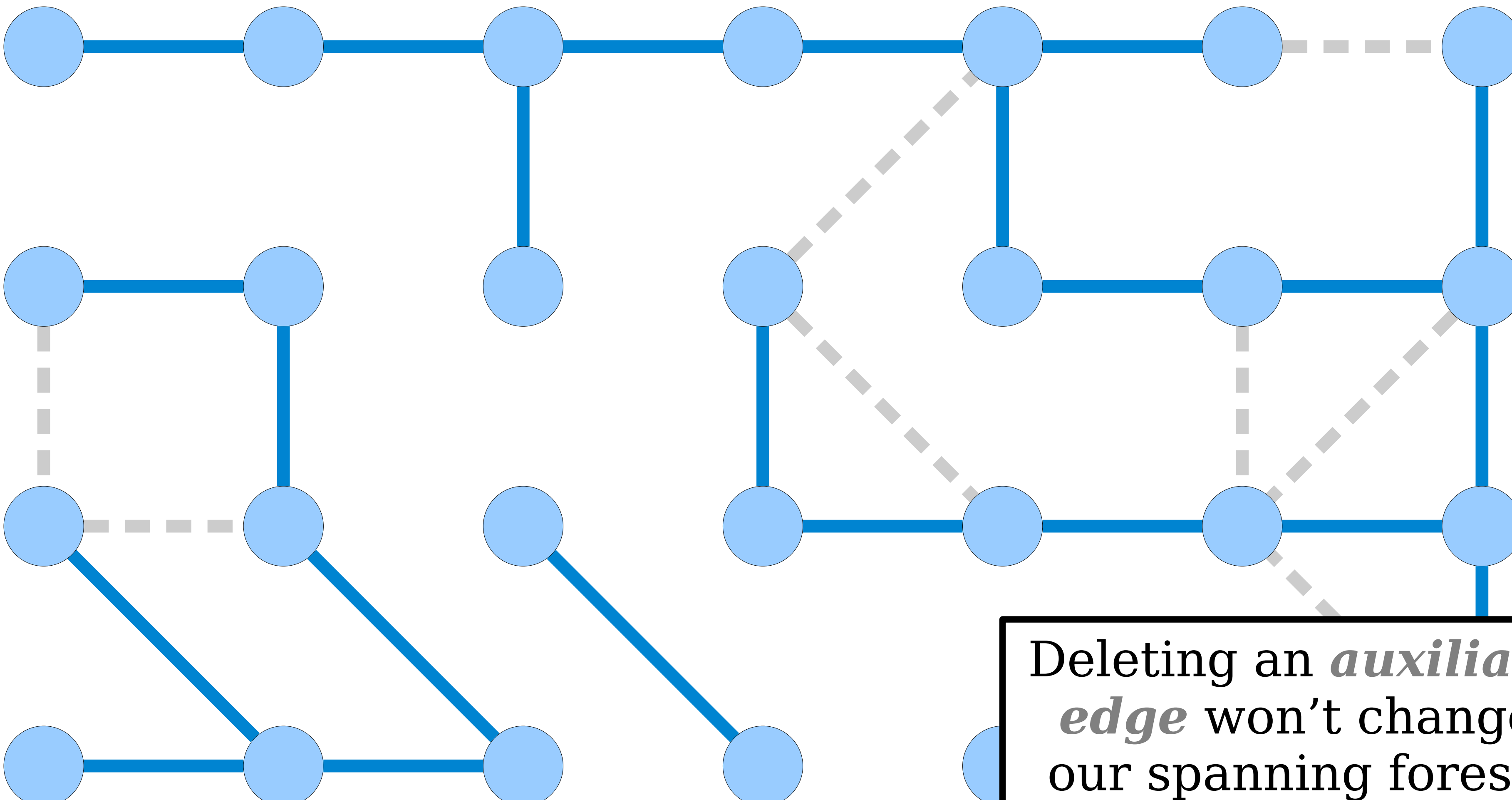
If an edge connects two different trees in $\mathscr{F}$, we add it as a ***tree edge***.

Maintaining a Forest



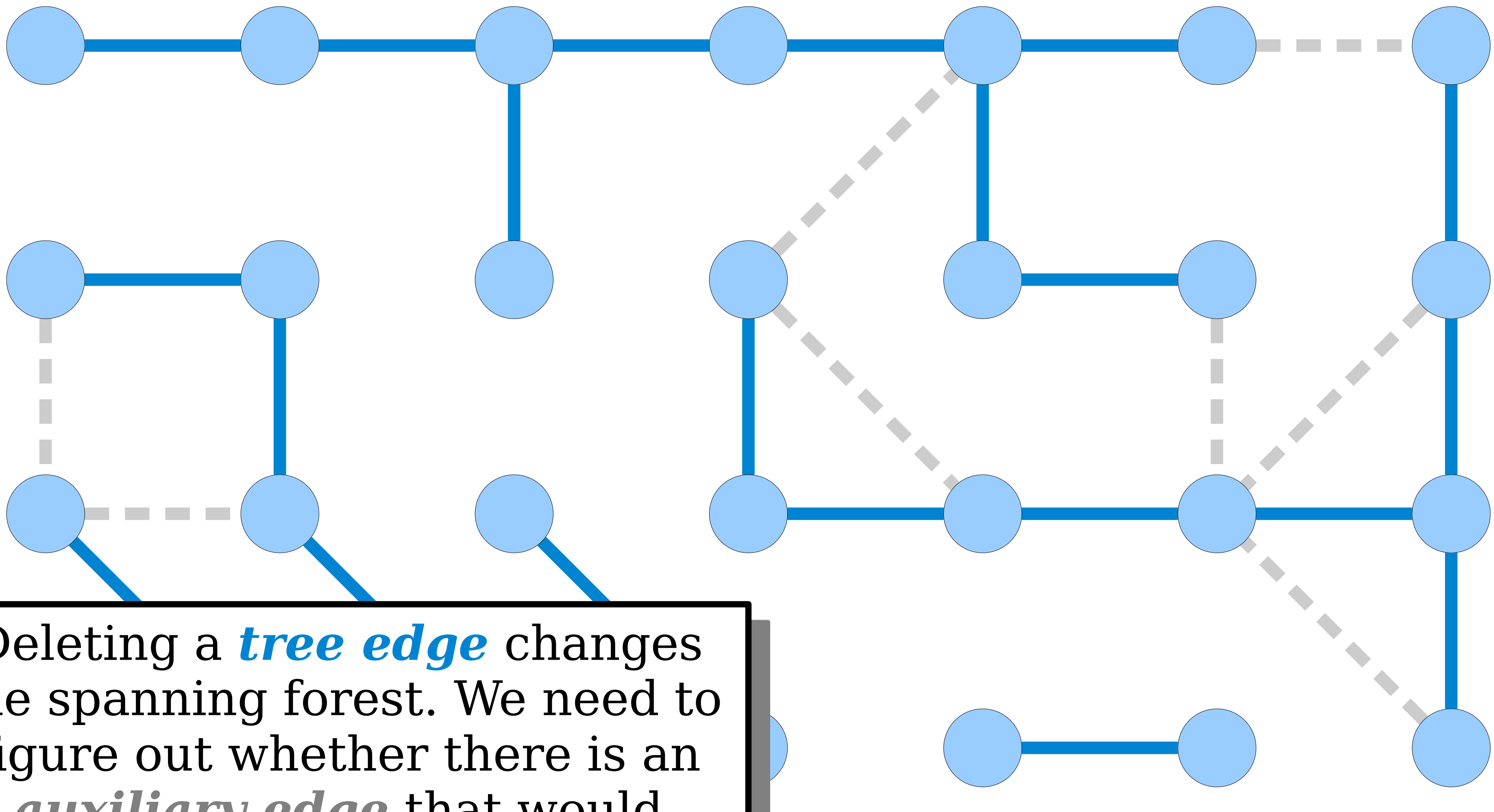
If an edge's endpoints are already connected, we add it as an ***auxiliary edge***.

Maintaining a Forest



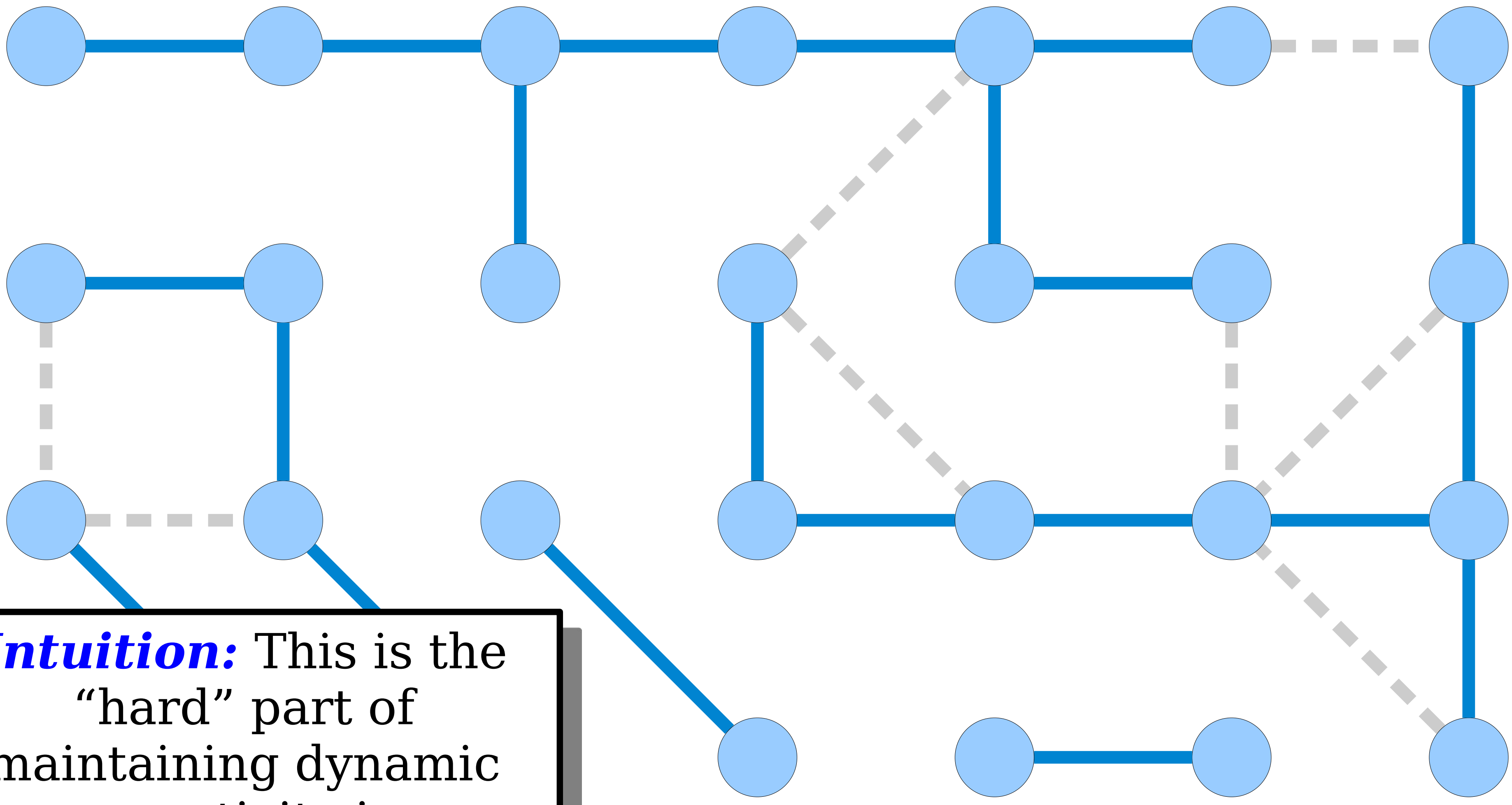
Deleting an ***auxiliary edge*** won't change our spanning forest, and so we just remove it.

Maintaining a Forest



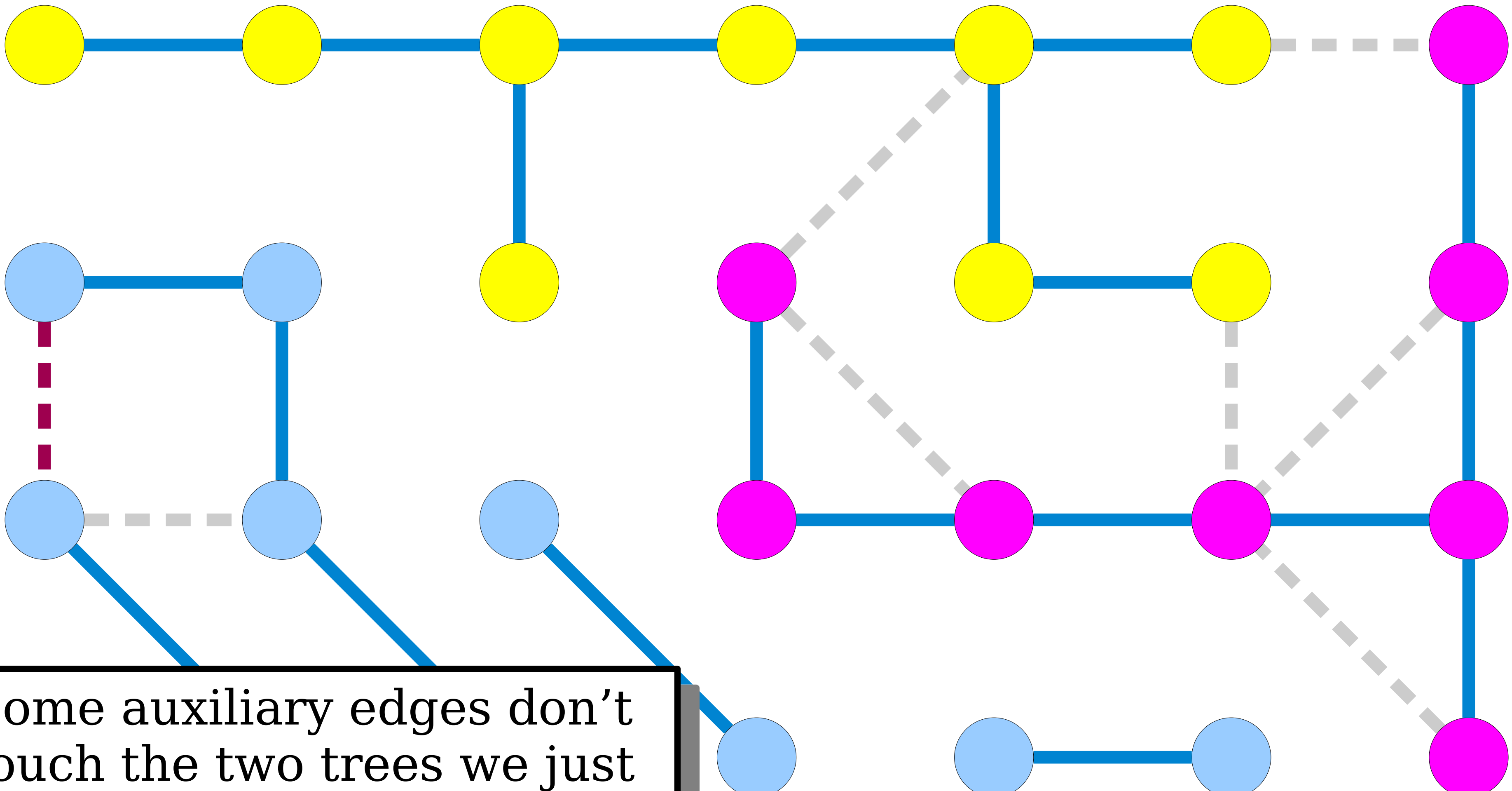
Deleting a ***tree edge*** changes the spanning forest. We need to figure out whether there is an ***auxiliary edge*** that would reconnect the tree we just split.

Maintaining a Forest



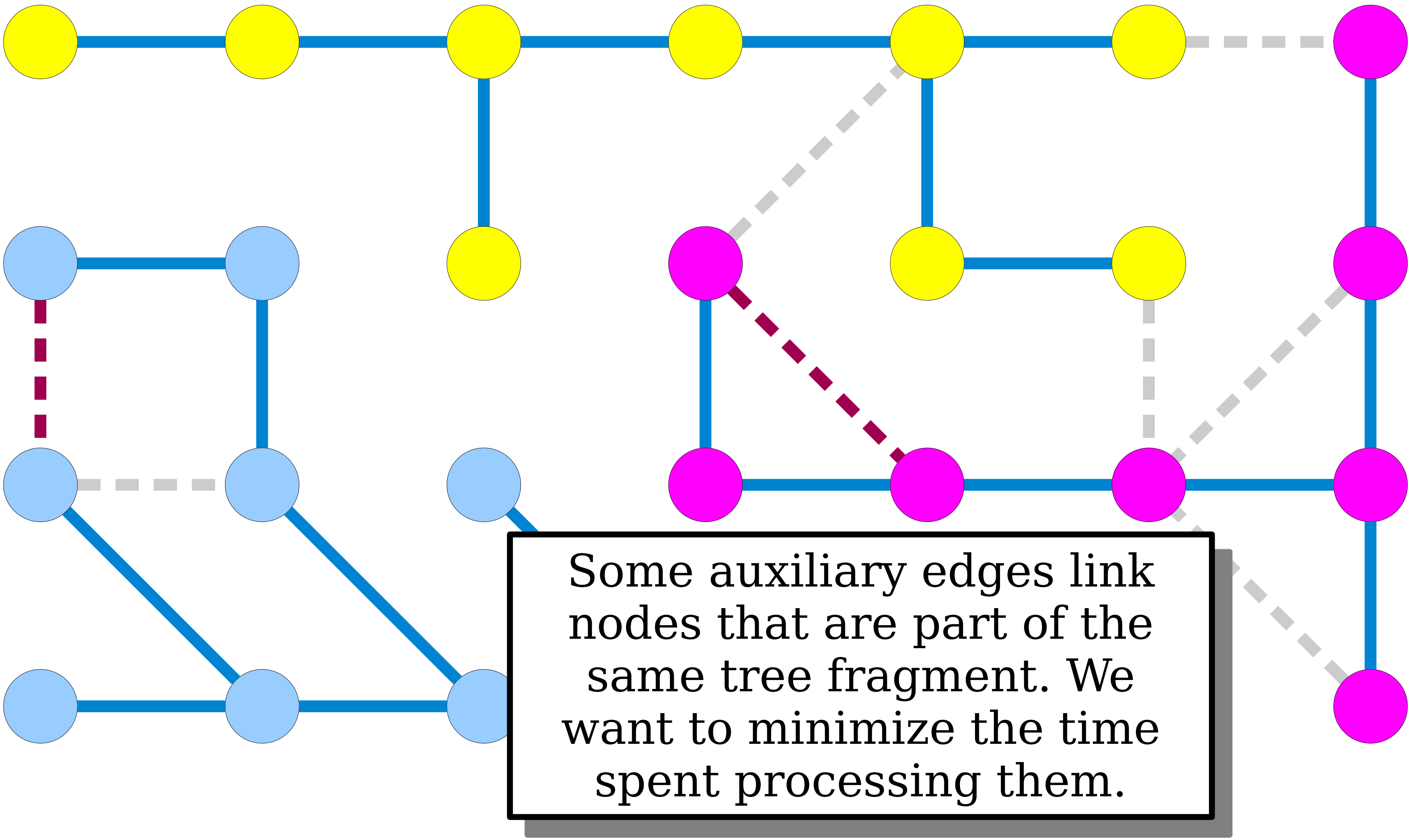
Intuition: This is the “hard” part of maintaining dynamic connectivity in a general graph.

Maintaining a Forest

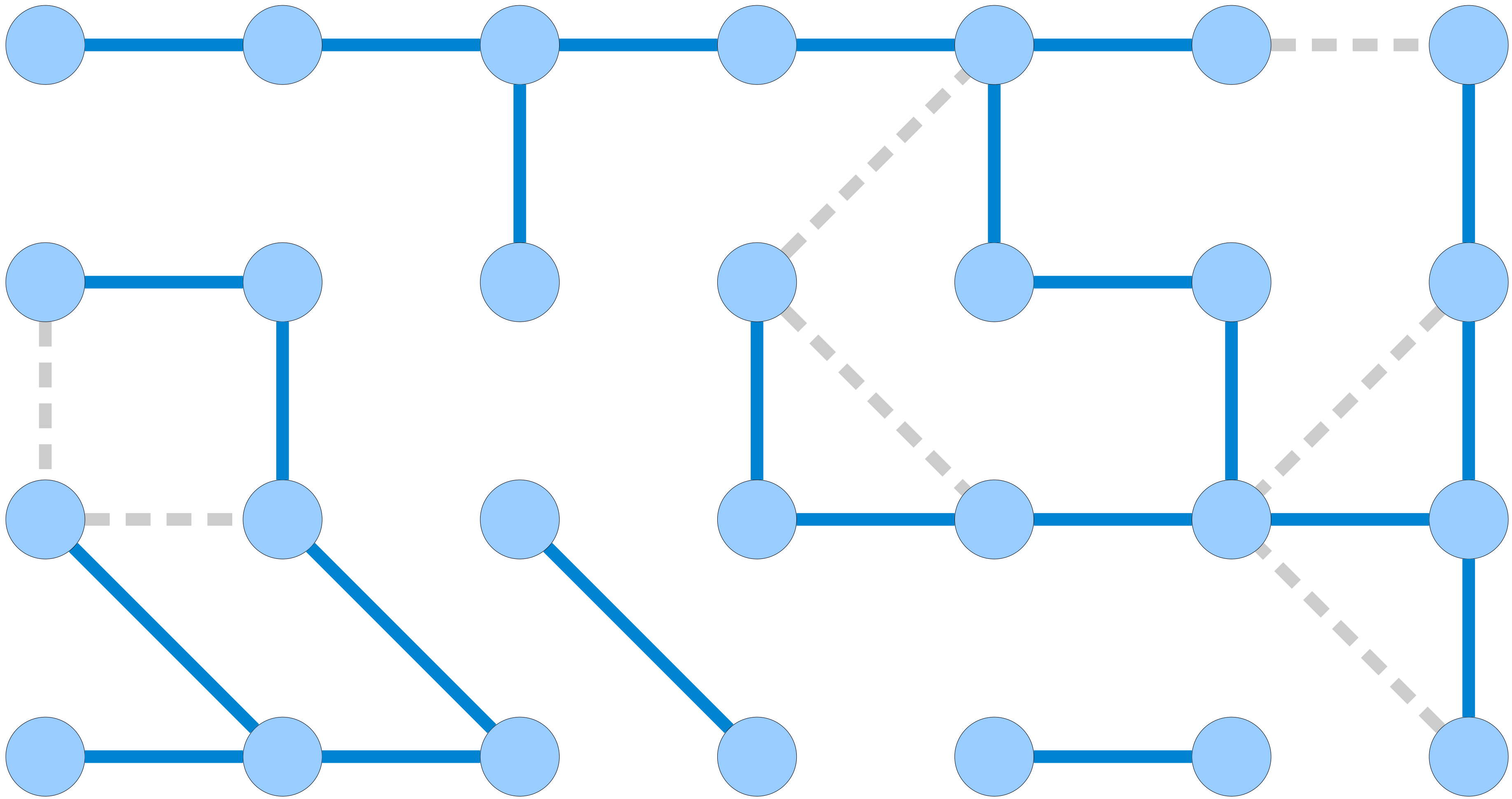


Some auxiliary edges don't touch the two trees we just disconnected. Processing them is a waste of time.

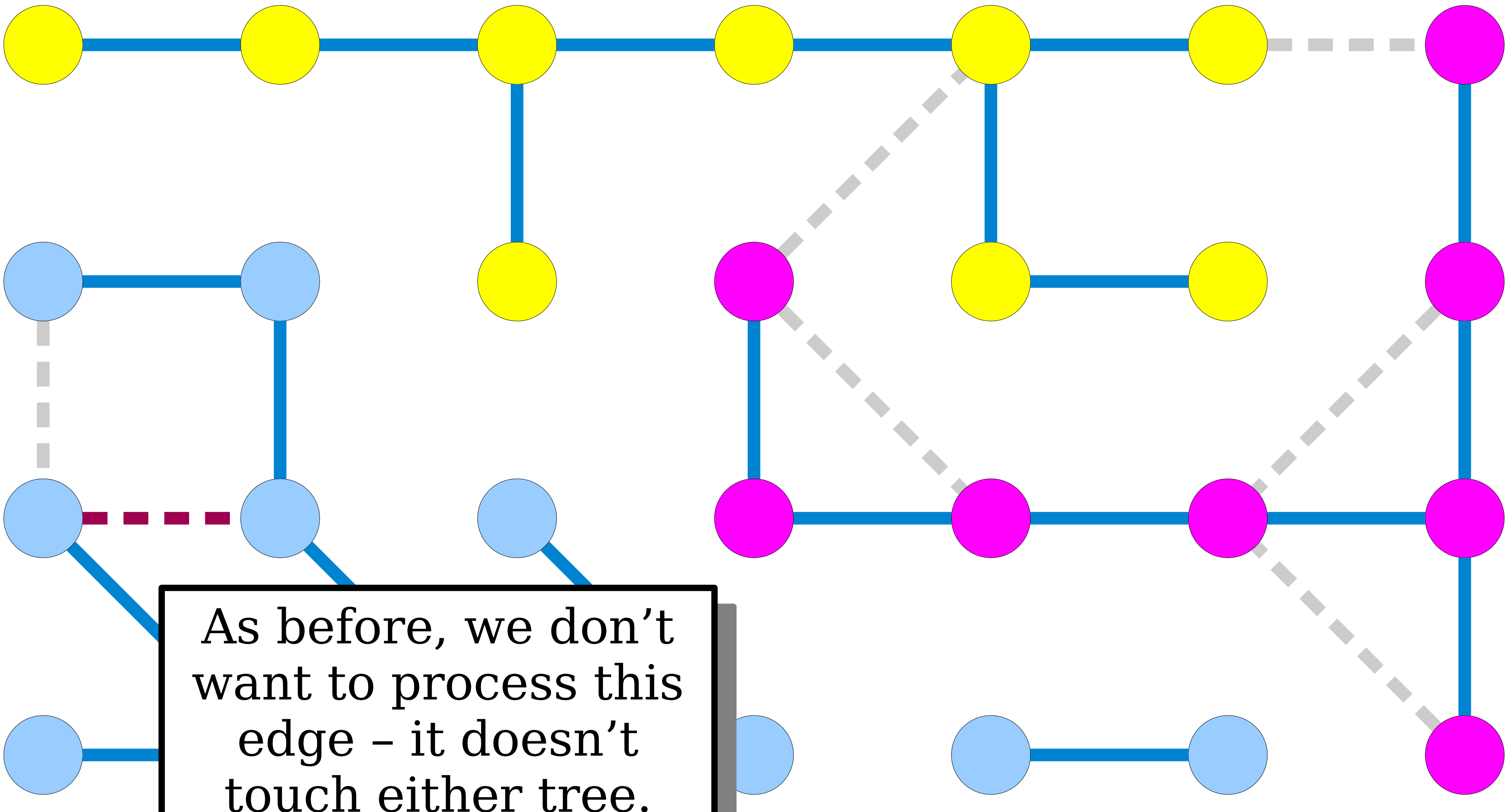
Maintaining a Forest



Maintaining a Forest

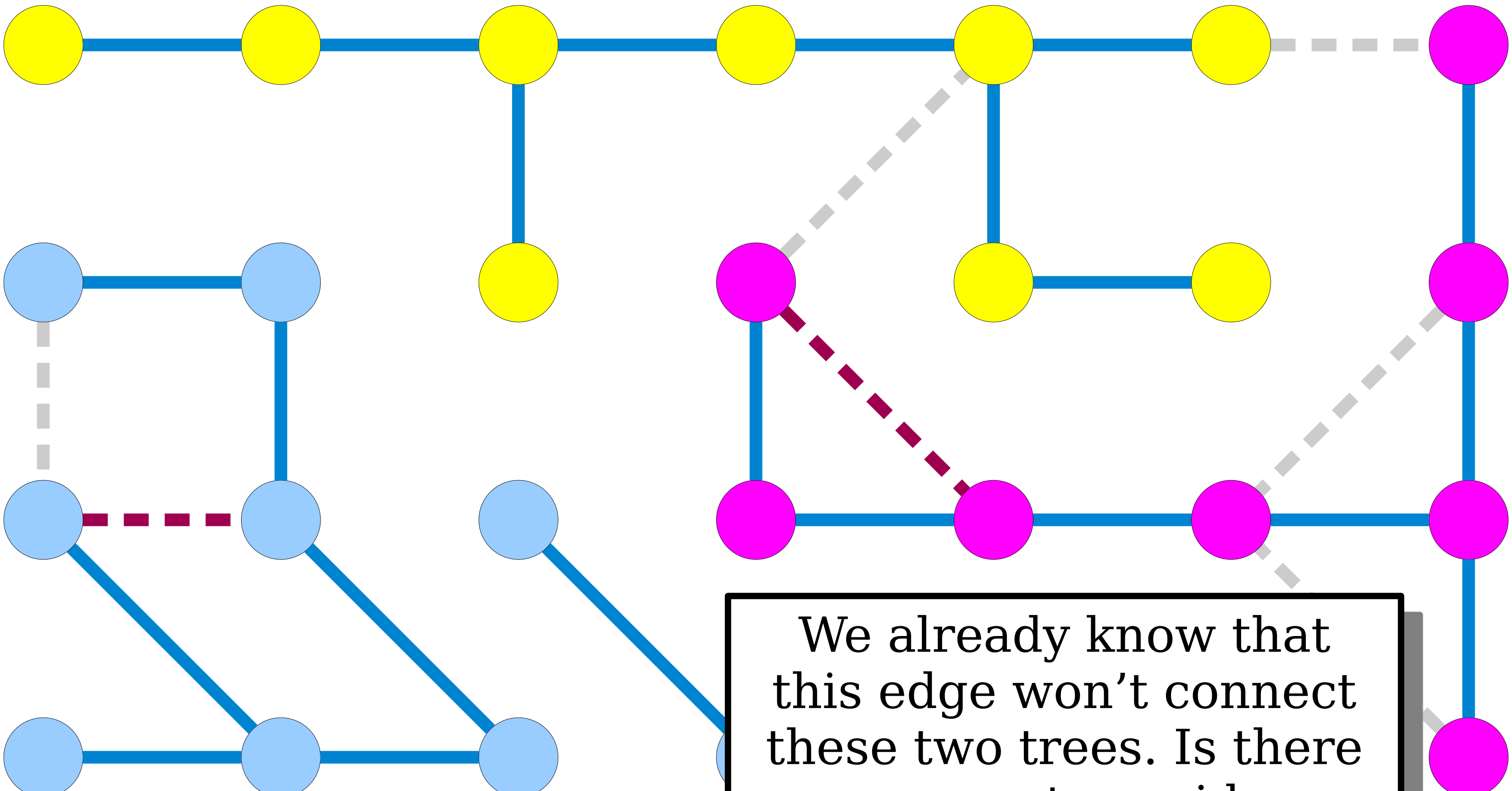


Maintaining a Forest



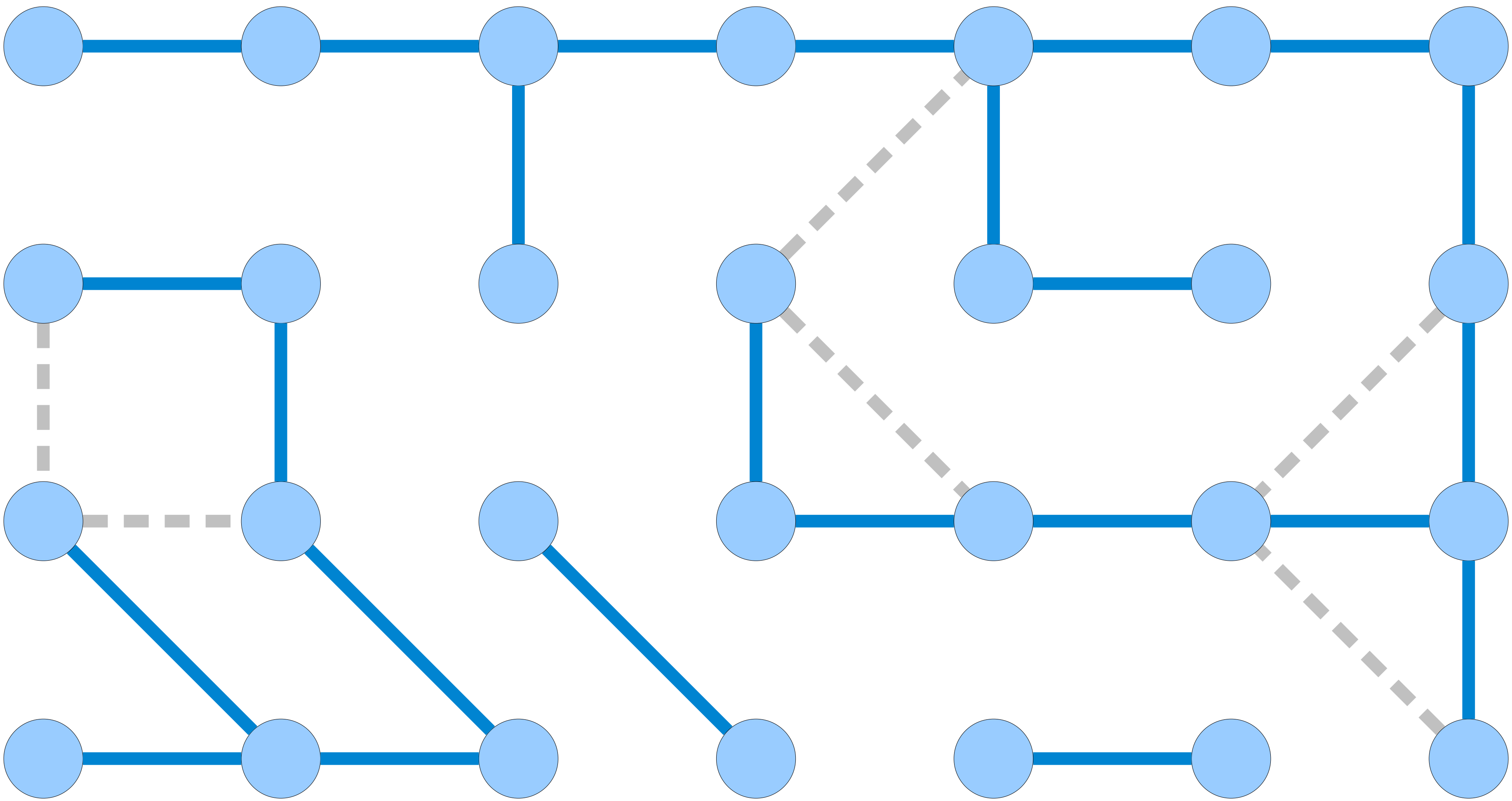
As before, we don't want to process this edge – it doesn't touch either tree.

Maintaining a Forest



We already know that this edge won't connect these two trees. Is there a way to avoid rescanning it?

Maintaining a Forest

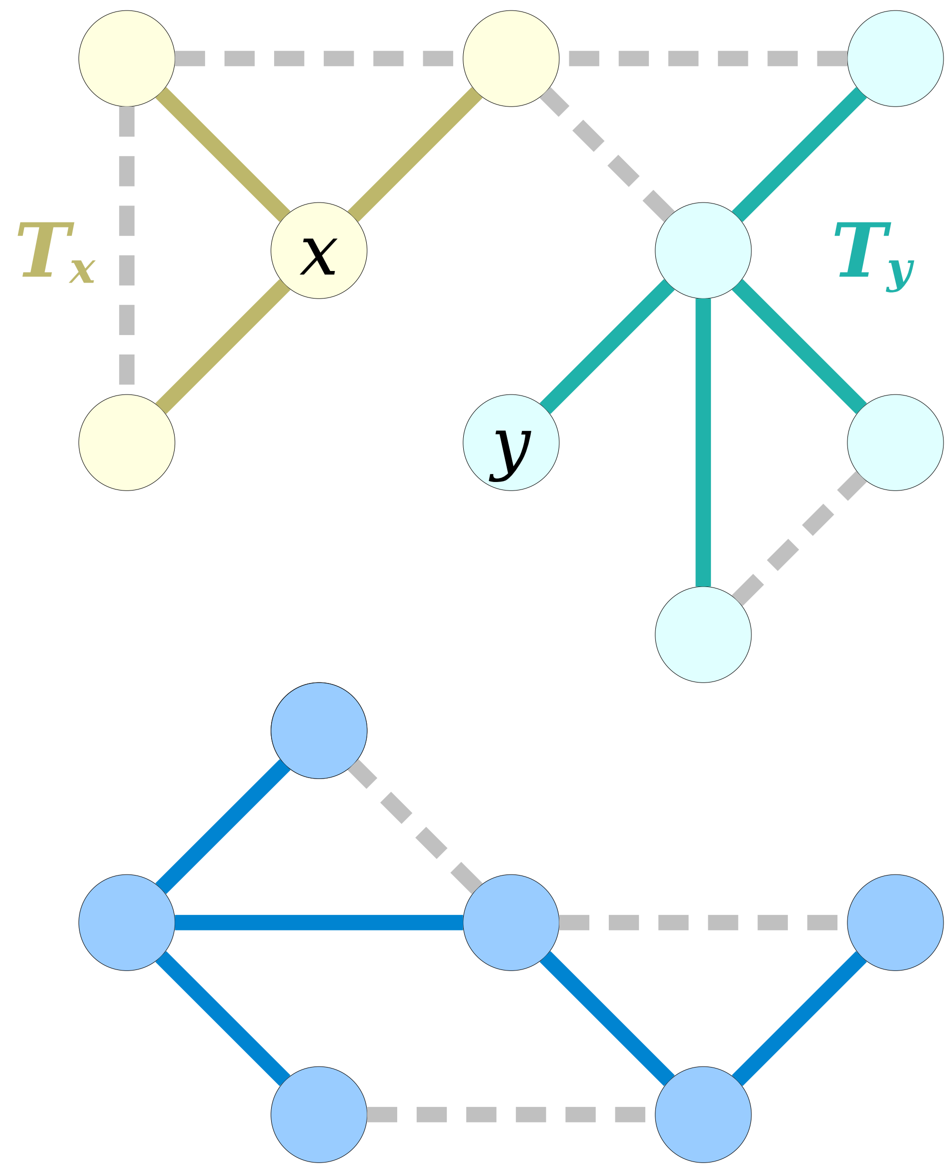


The Challenges

- ***Goal:*** After disconnecting a tree T into two trees T_1 and T_2, search for an edge that will reconnect it.
- ***Challenge 1:*** Avoid scanning edges that don't have endpoints in either T_1 or T_2.
- ***Challenge 2:*** Avoid rescanning edges that, based on past cuts, couldn't possibly work.

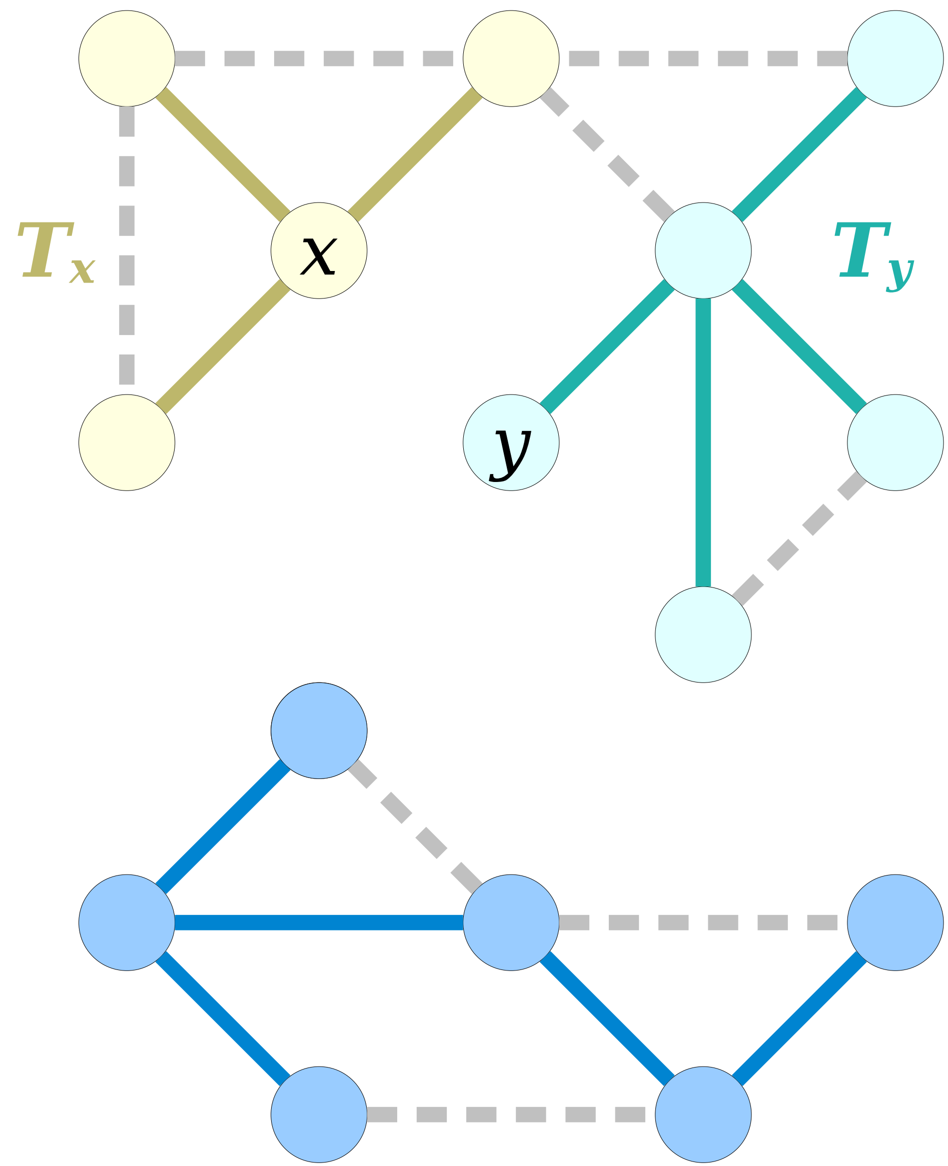
What We Need To Do

- Suppose we cut the tree edge xy, splitting a tree T into T_x and T_y.
- We need to search for an auxiliary edge that could reconnect T_x and T_y.
- ***Observation:*** Auxiliary edges with one endpoint in T_x either run between T_x and itself or between T_x and T_y.
- ***Goal:*** Organize auxiliary edges so we can find just those incident to T_x.



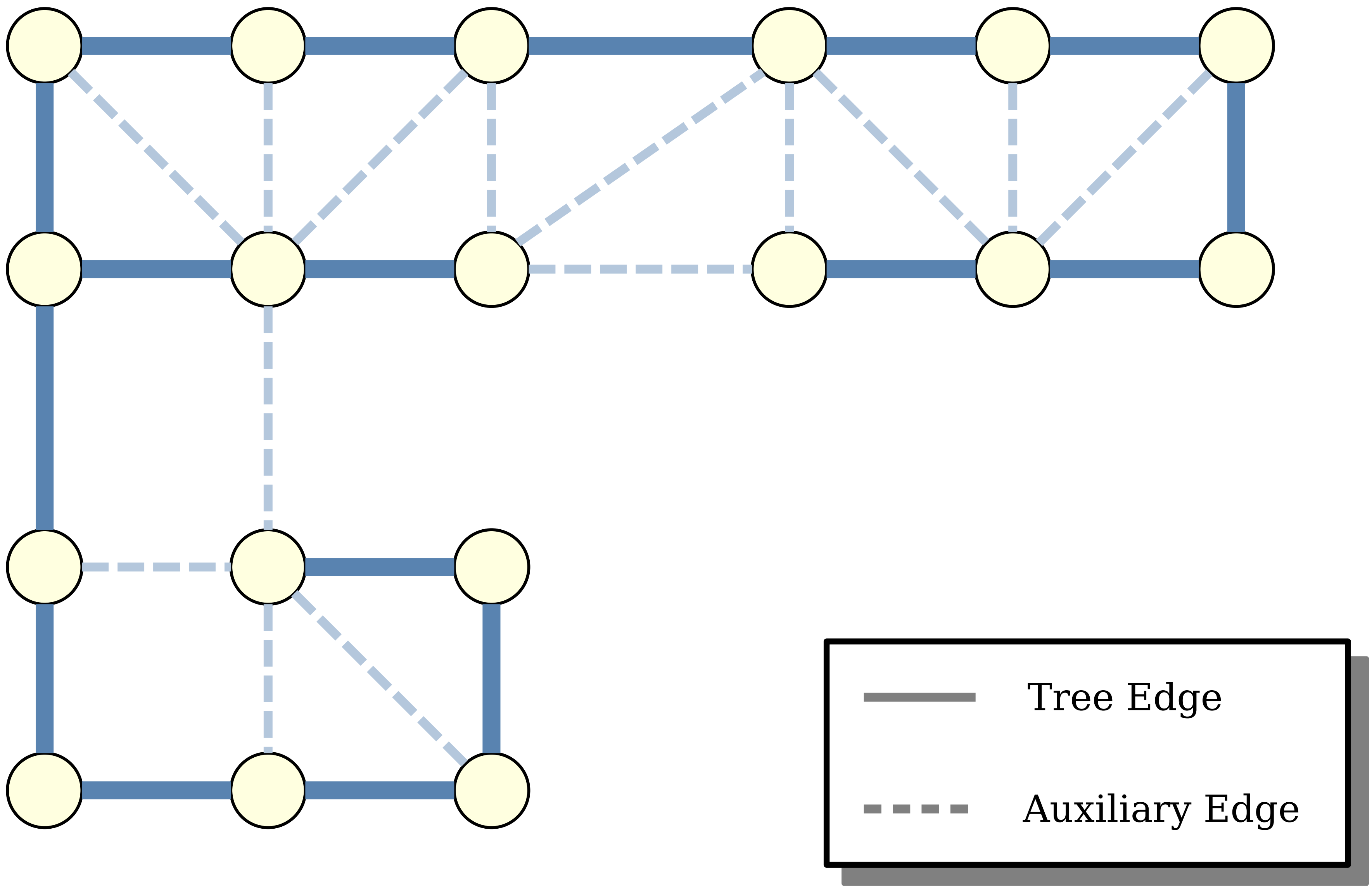
What We Need To Do

- Fortunately, we've already seen a way to do this!
- ***Recall:*** Euler tour trees can be augmented so that we can
  - attach packets to nodes, and
  - quickly execute queries of the form "find and remove some packet in this tree."
- Replace "packet" with "auxiliary edge" and we can find an auxiliary edge with one endpoint in T_x in amortized time **O(log *n*)**.
- ***Intuition:*** We can "quickly" find an edge touching T_x. This will not be our bottleneck.

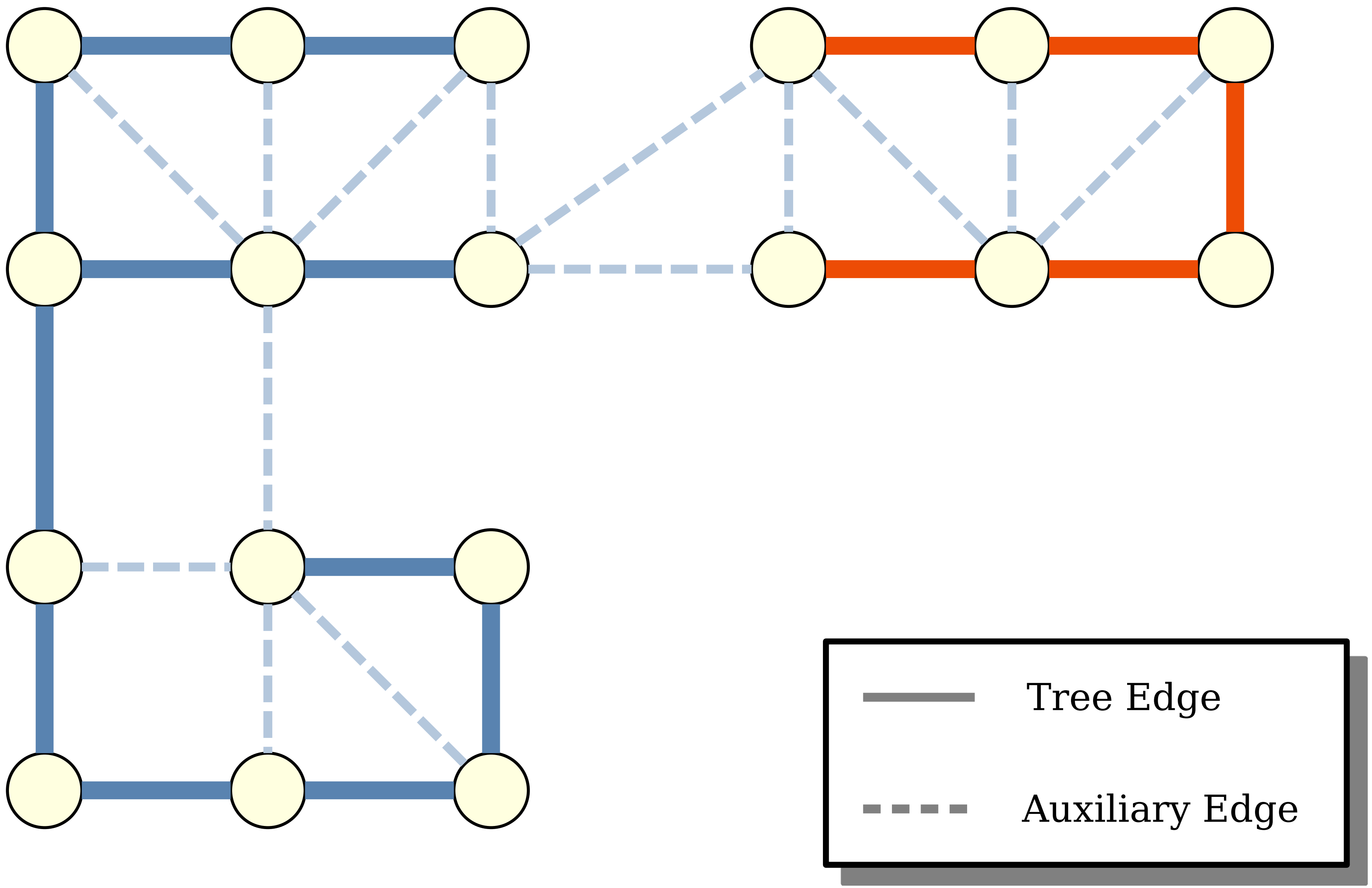


Avoiding Rescans

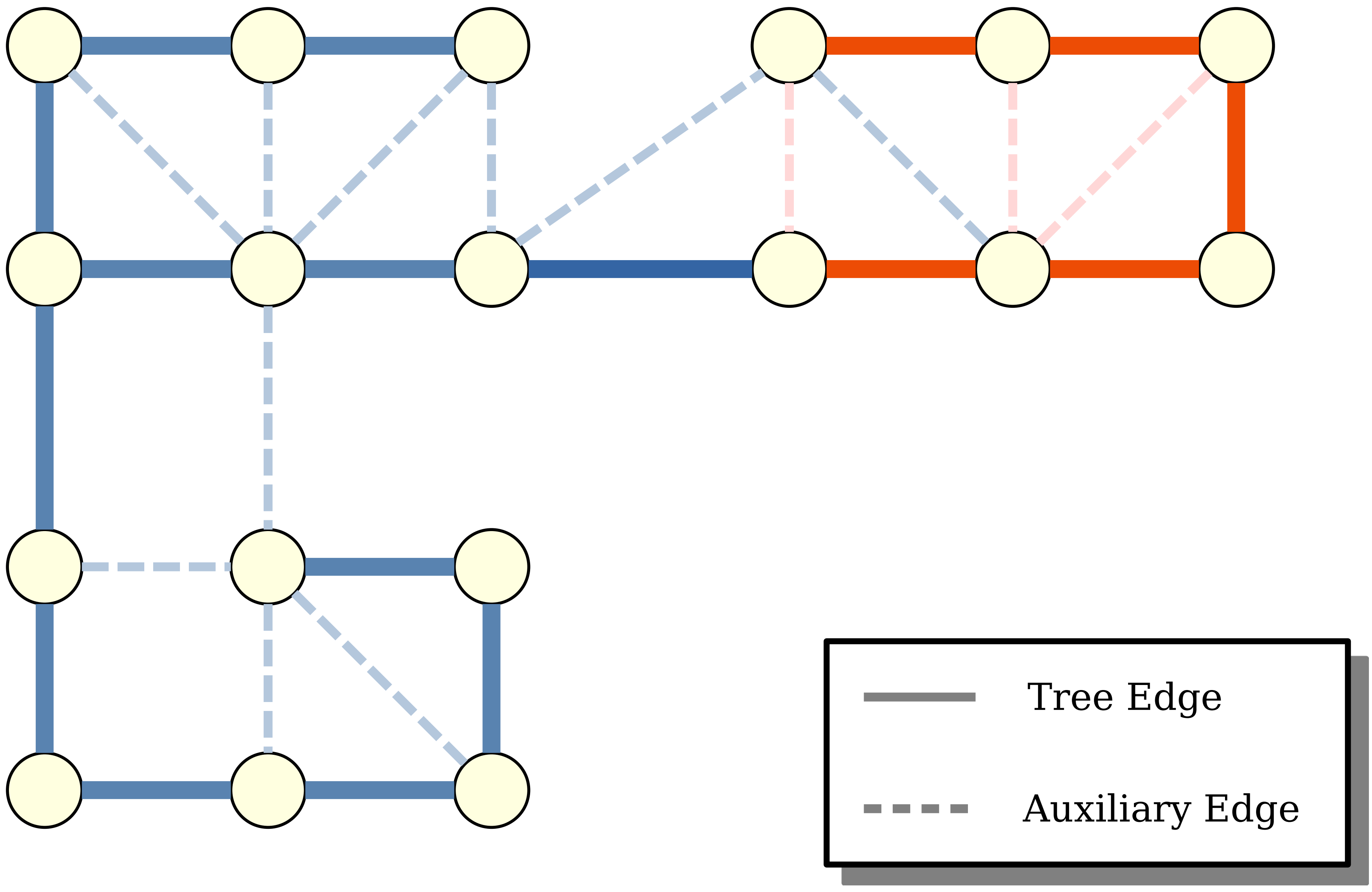
Avoiding Rescans



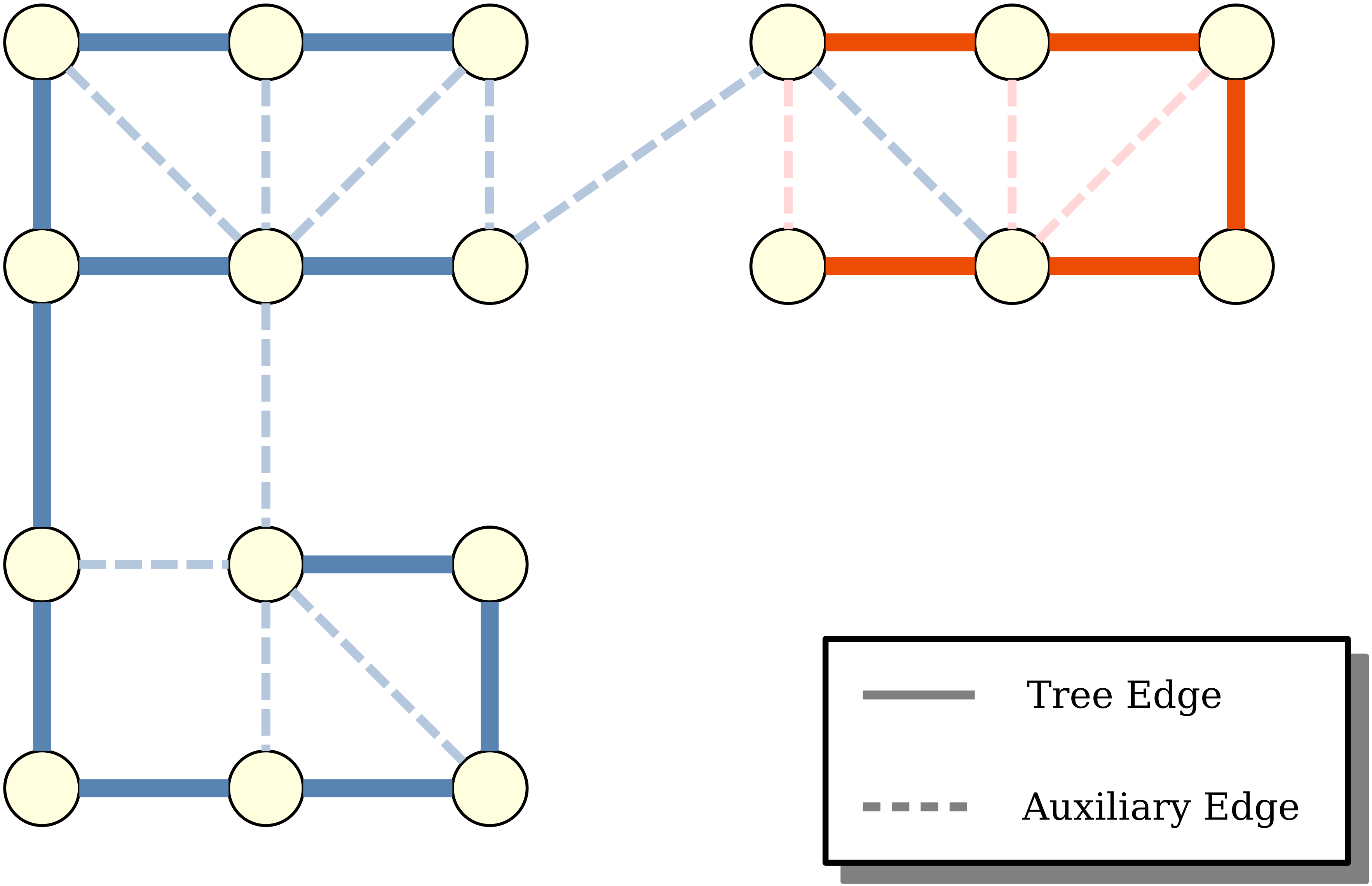
Avoiding Rescans



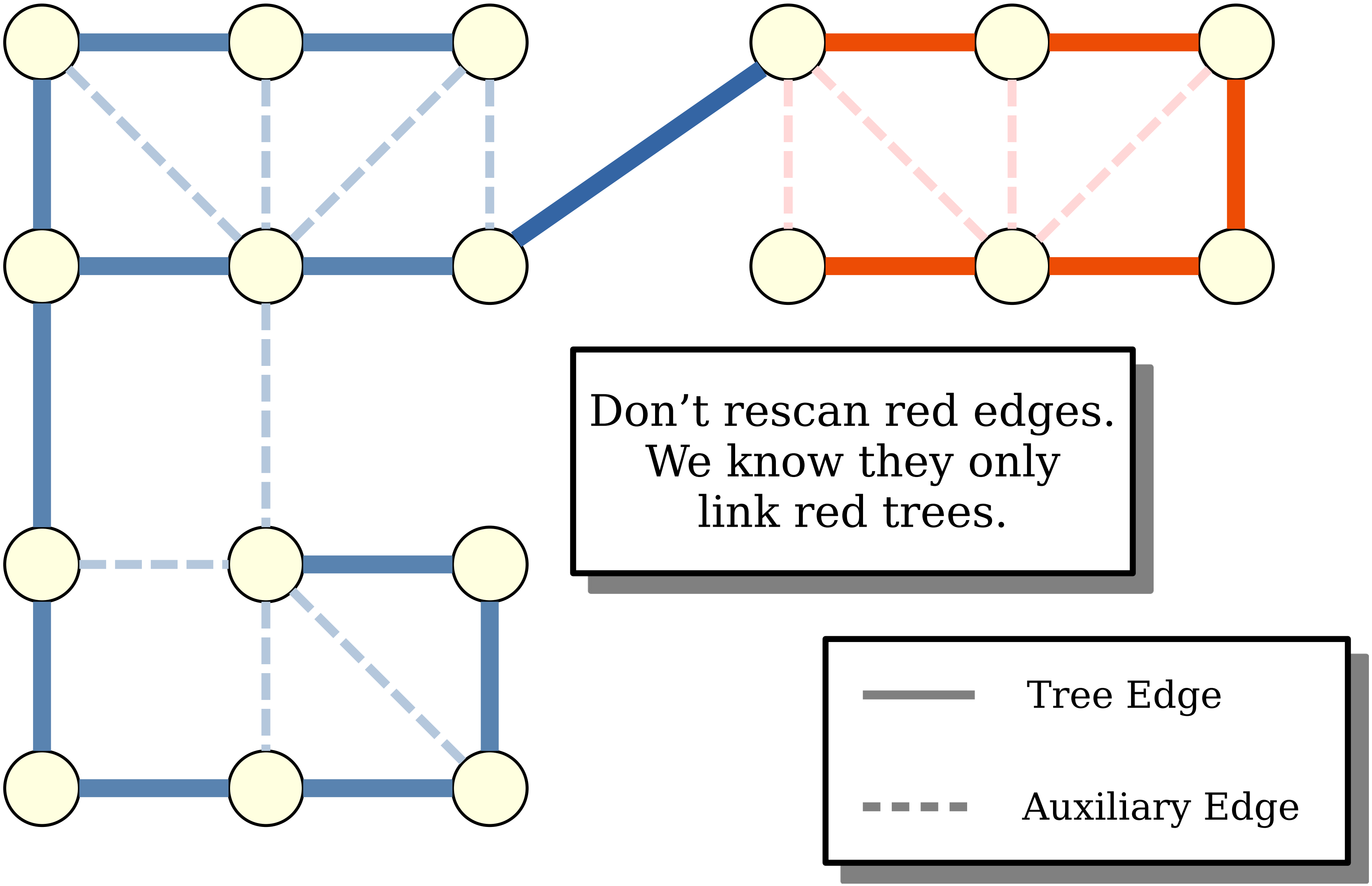
Avoiding Rescans



Avoiding Rescans

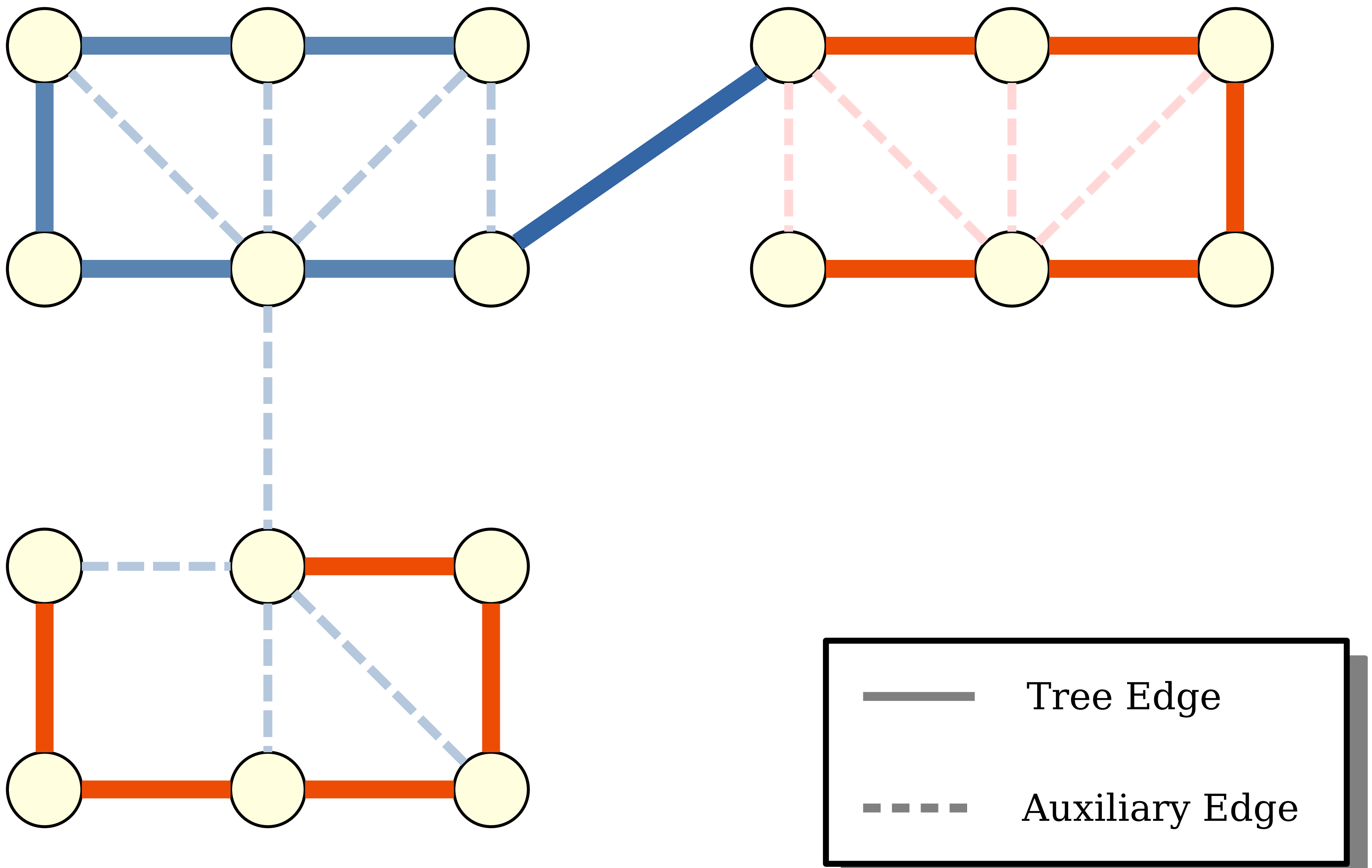


Avoiding Rescans

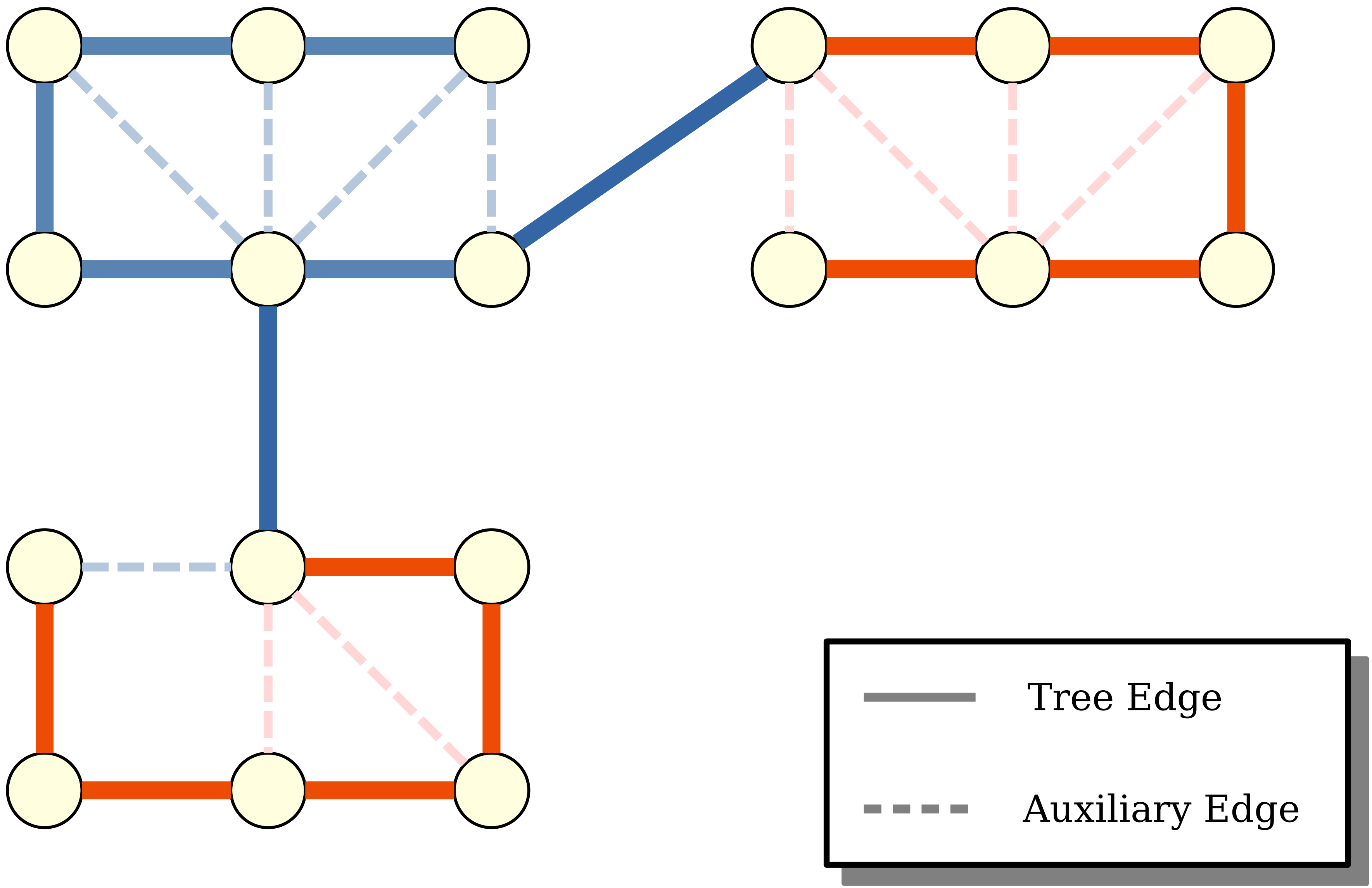


Don’t rescan red edges.
We know they only
link red trees.

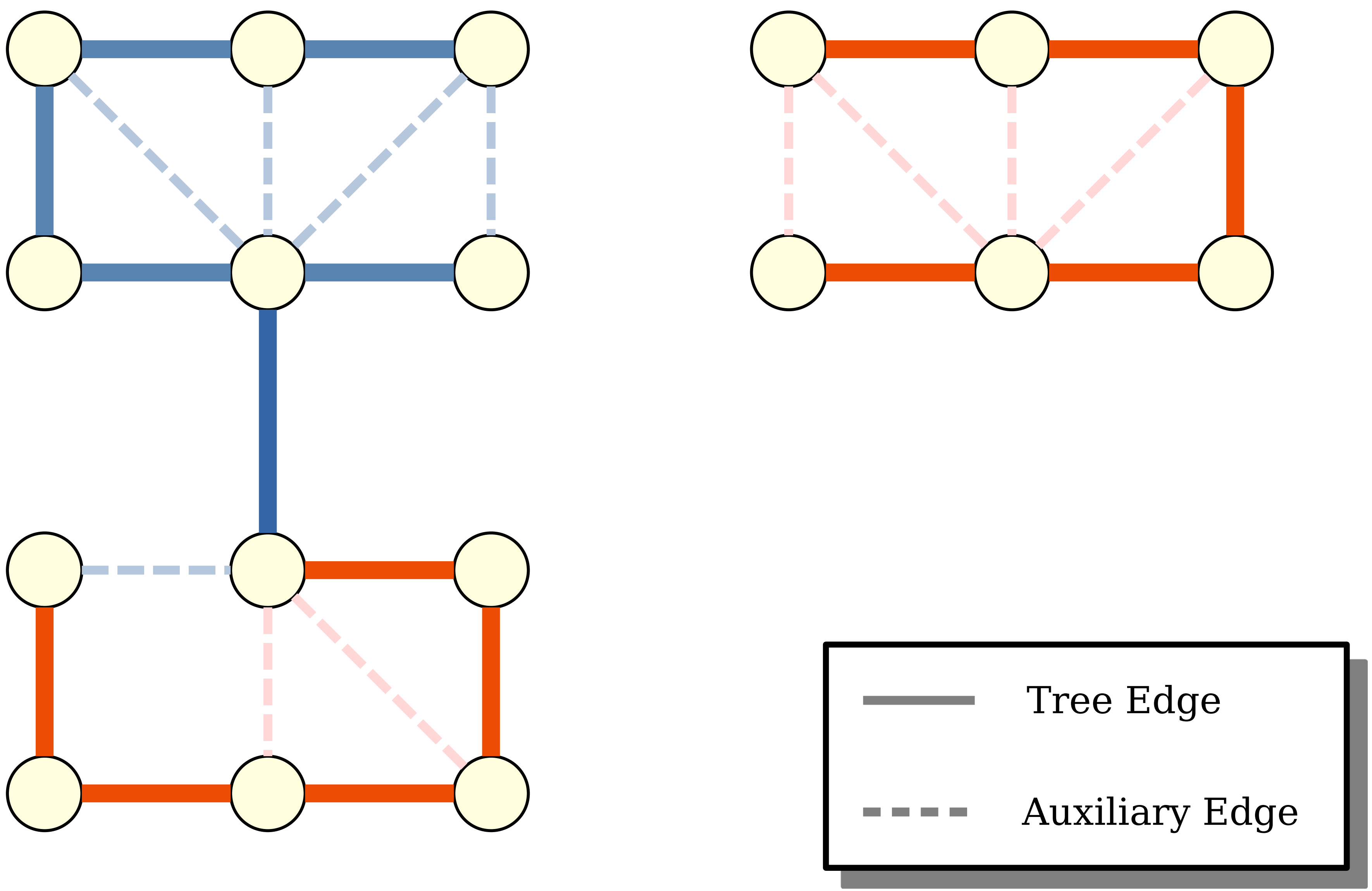
Avoiding Rescans



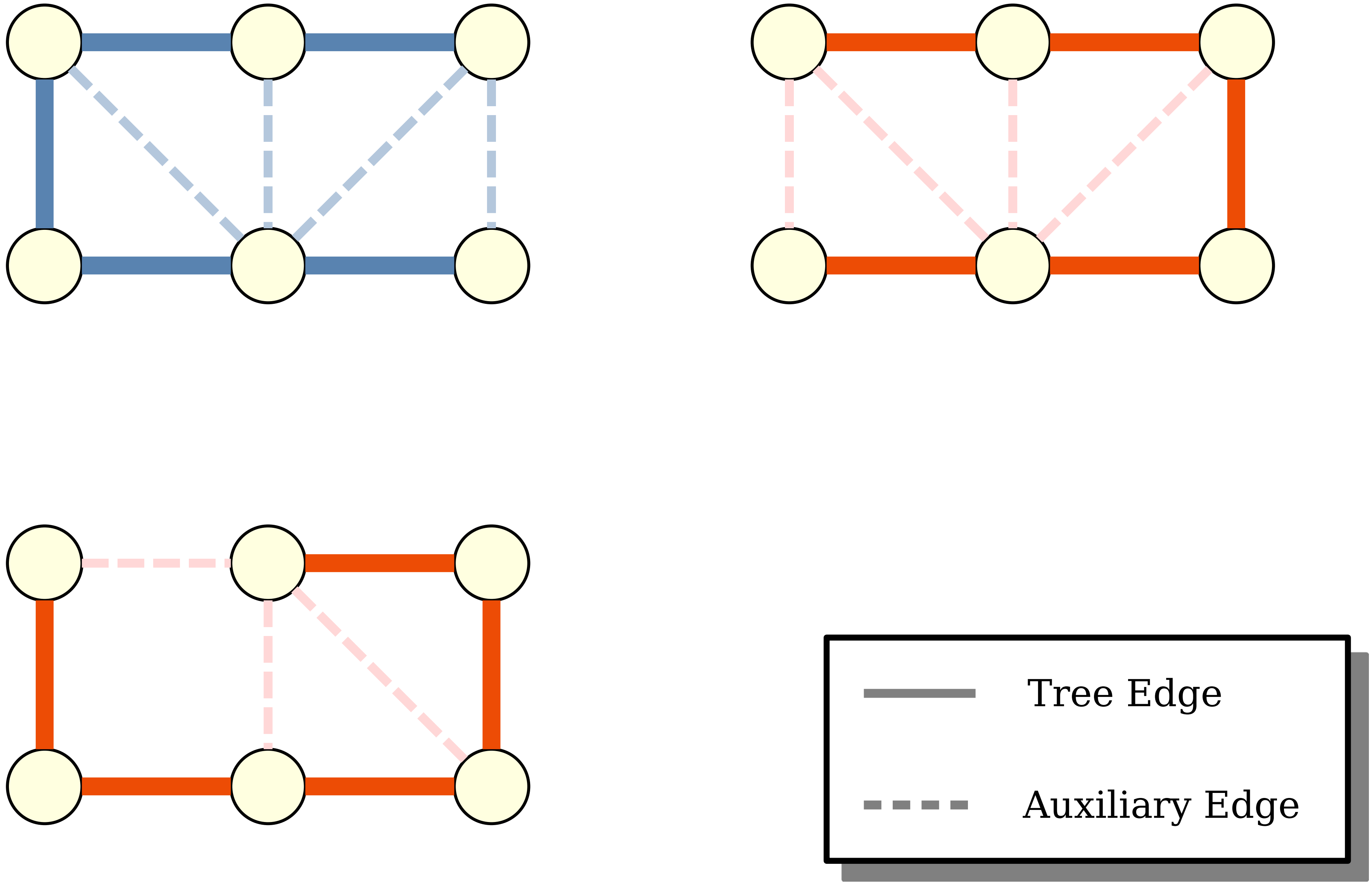
Avoiding Rescans



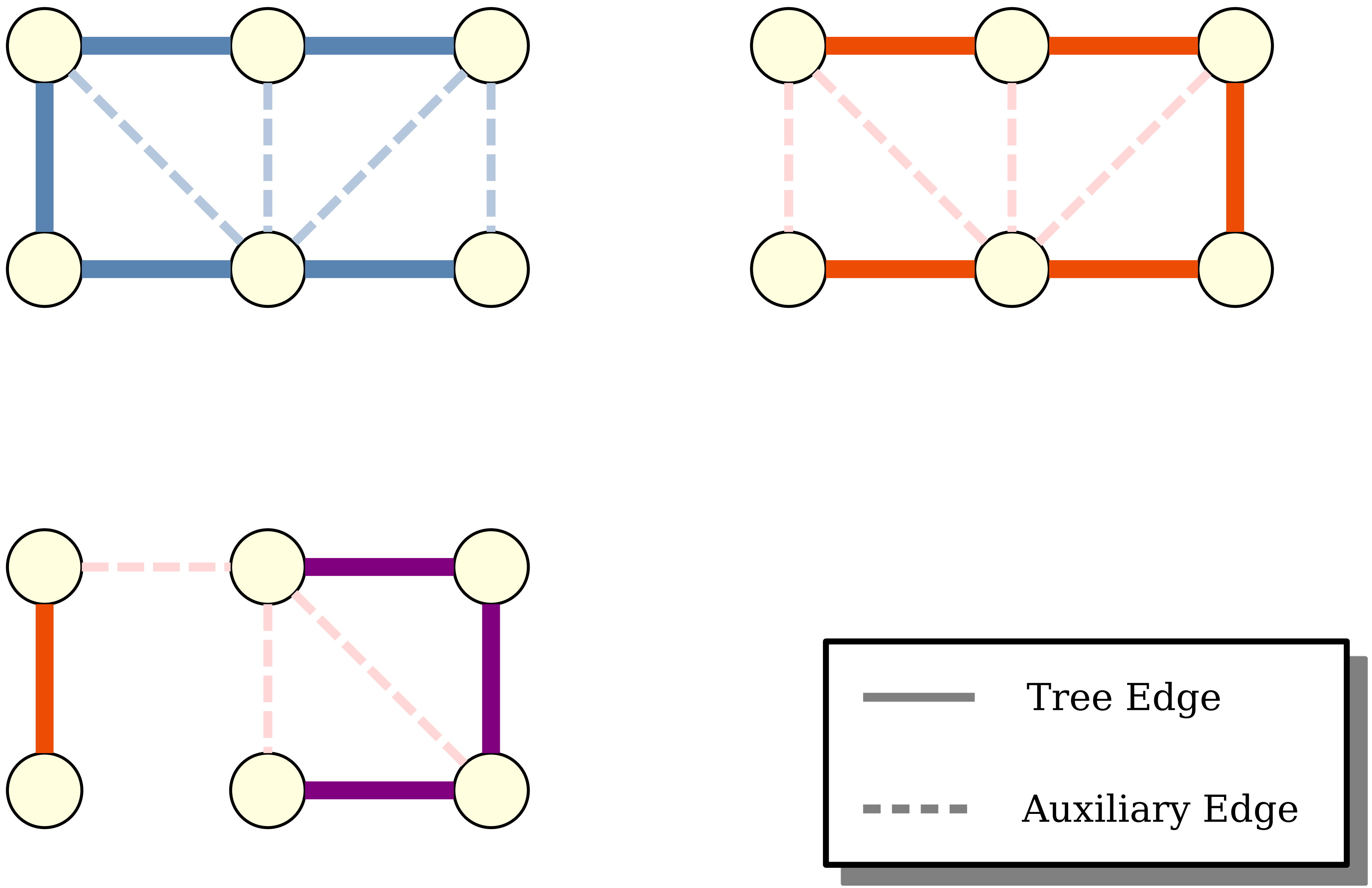
Avoiding Rescans



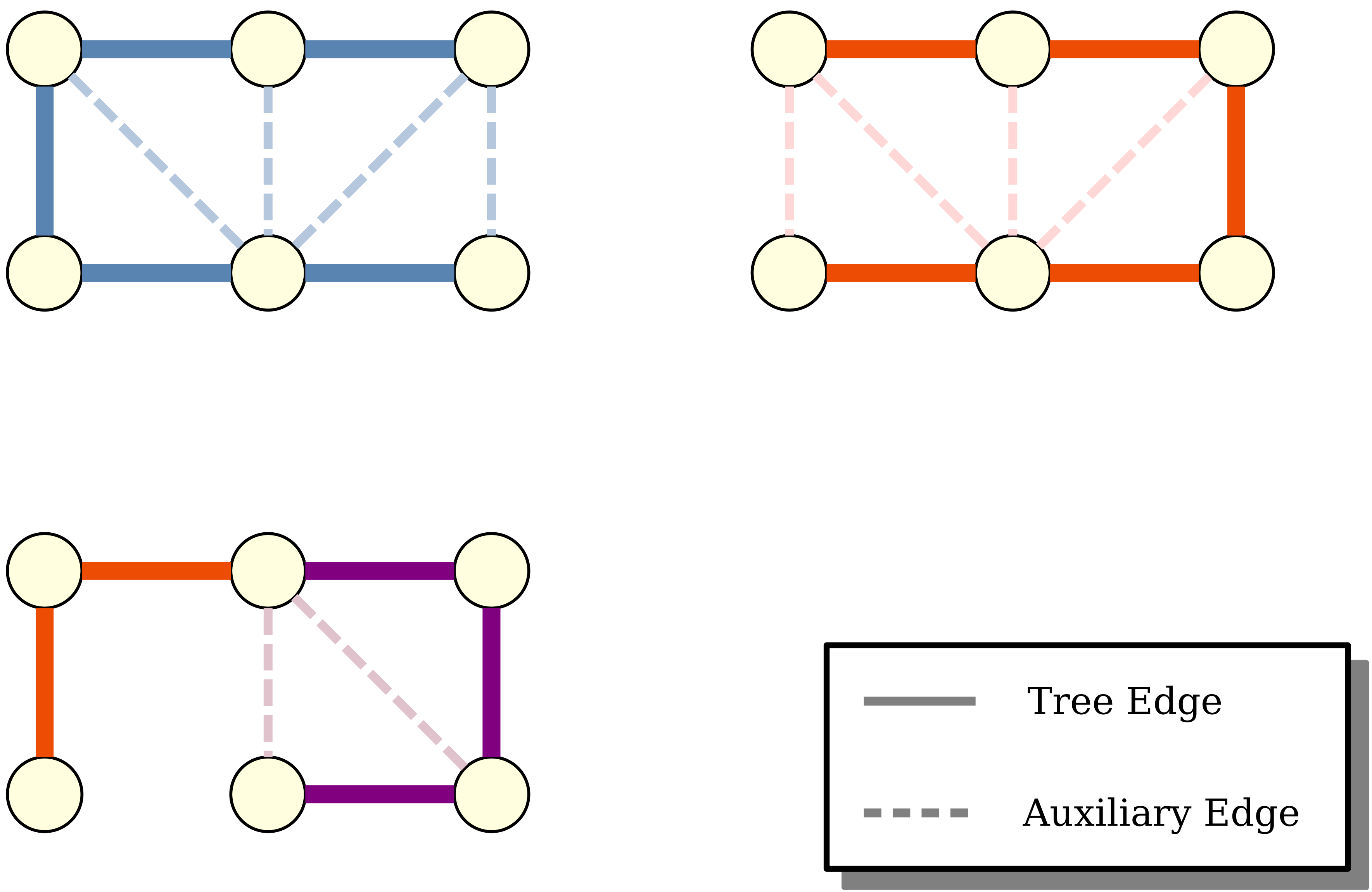
Avoiding Rescans



Avoiding Rescans



Avoiding Rescans



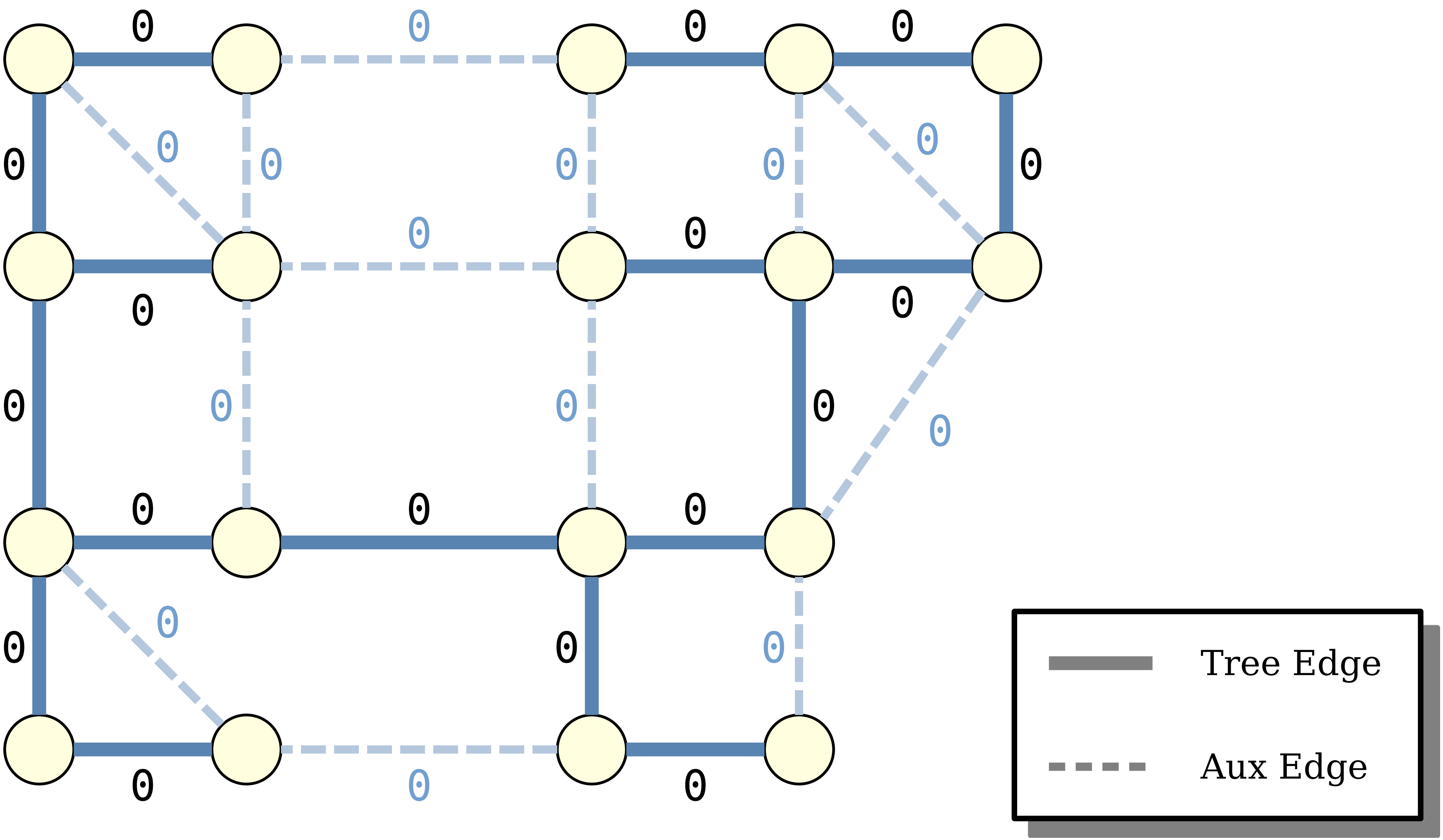
A Germ of an Idea

- Begin with all edges having the same color.
- When cutting a tree edge, assign edges in one of the two trees a new color.
- When an edge fails to reconnect a tree, give it the color of the tree it belongs to.
- When looking for a replacement edge, don’t use edges that are the same color as the tree itself, since those can’t work.

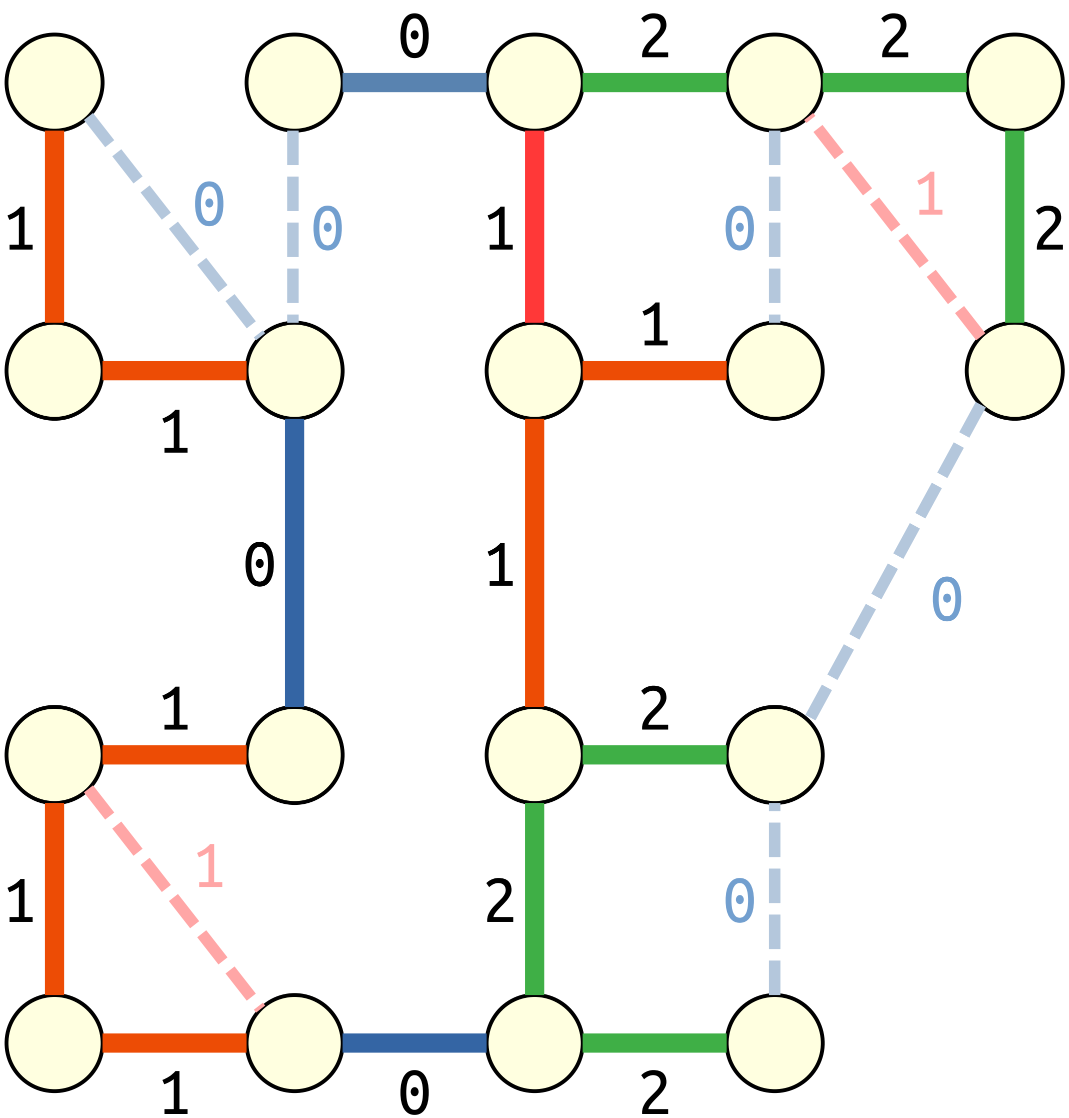
Refining This Idea

- ***Idea:*** Assign each edge a ***level***, initially 0.
- ***Initial Proposal:***
  - When cutting an edge at level l, pick one of the two resulting trees and raise all its level-l edges to level (l+1).
  - When looking for an edge to reconnect the tree, if an edge at level l fails to reconnect, raise it to level l+1.
- There are a lot of details we still need to work out, but this is a reasonable guess for a starting point.

Refining This Idea



Refining This Idea

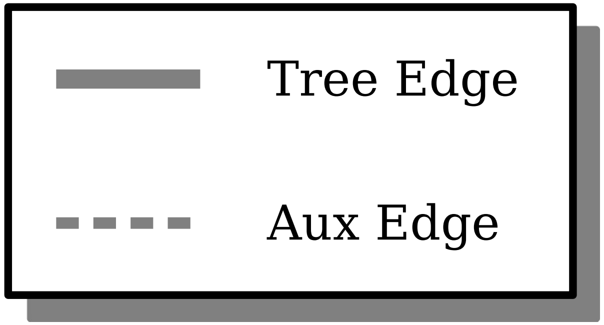


We have a spanning tree for our graph. Each edge has an associated level.

Notice anything about which spanning tree we picked?

Answer at

https://pollev.com/cs166spr23



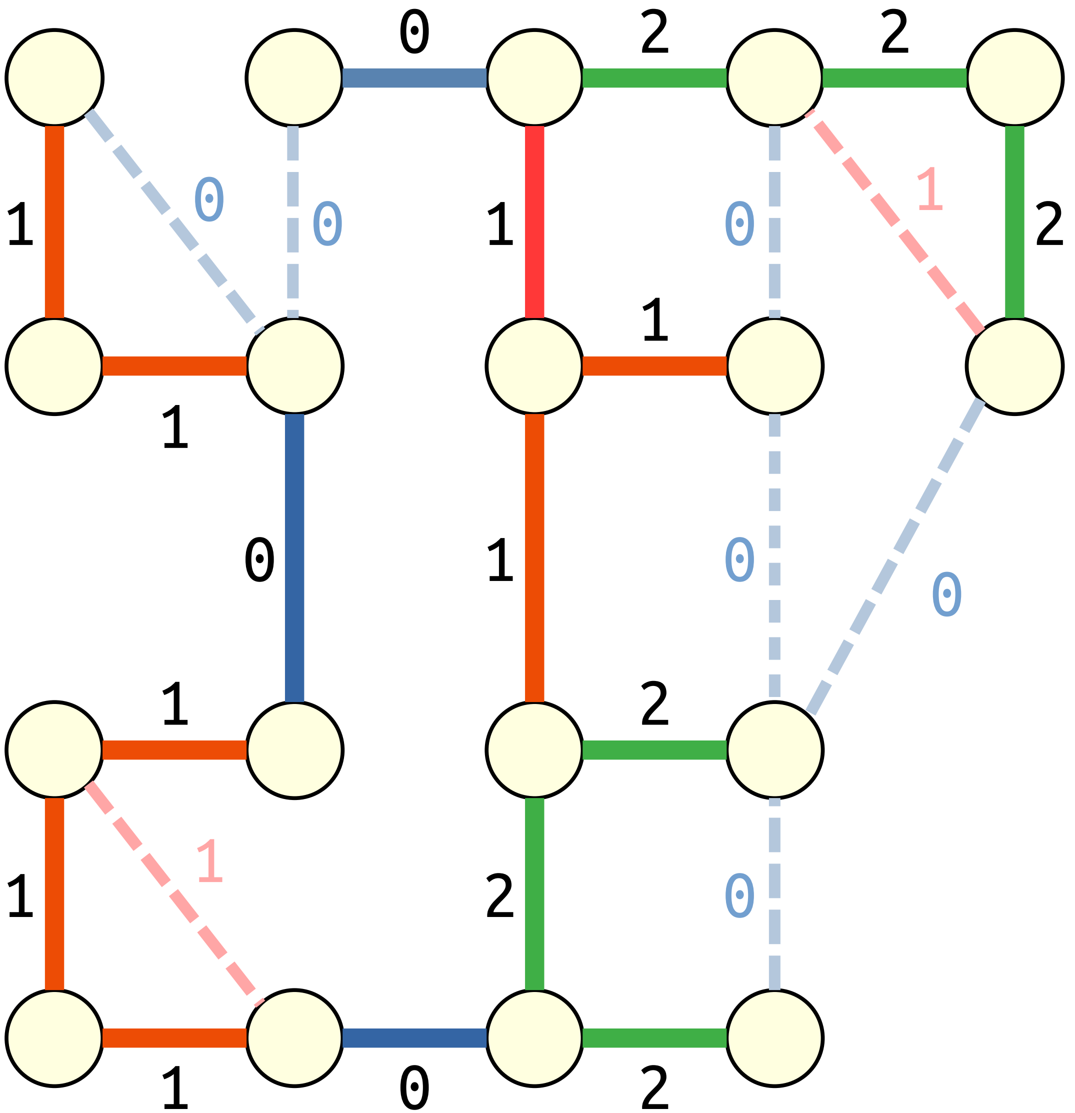
Maximum Spanning Forests

- ***Key Idea:*** Maintain the following invariant throughout all operations:

  The forest $\mathscr{F}$ is a maximum spanning forest with respect to levels.

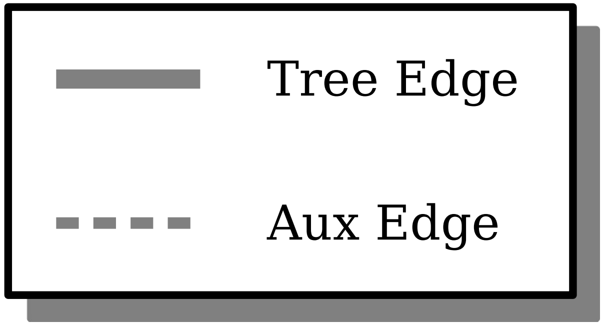
- We'll use this both to formalize the details of all the operations and to ensure correctness.
- Plus, this will help us in some tricky corner cases!

MSF Implications

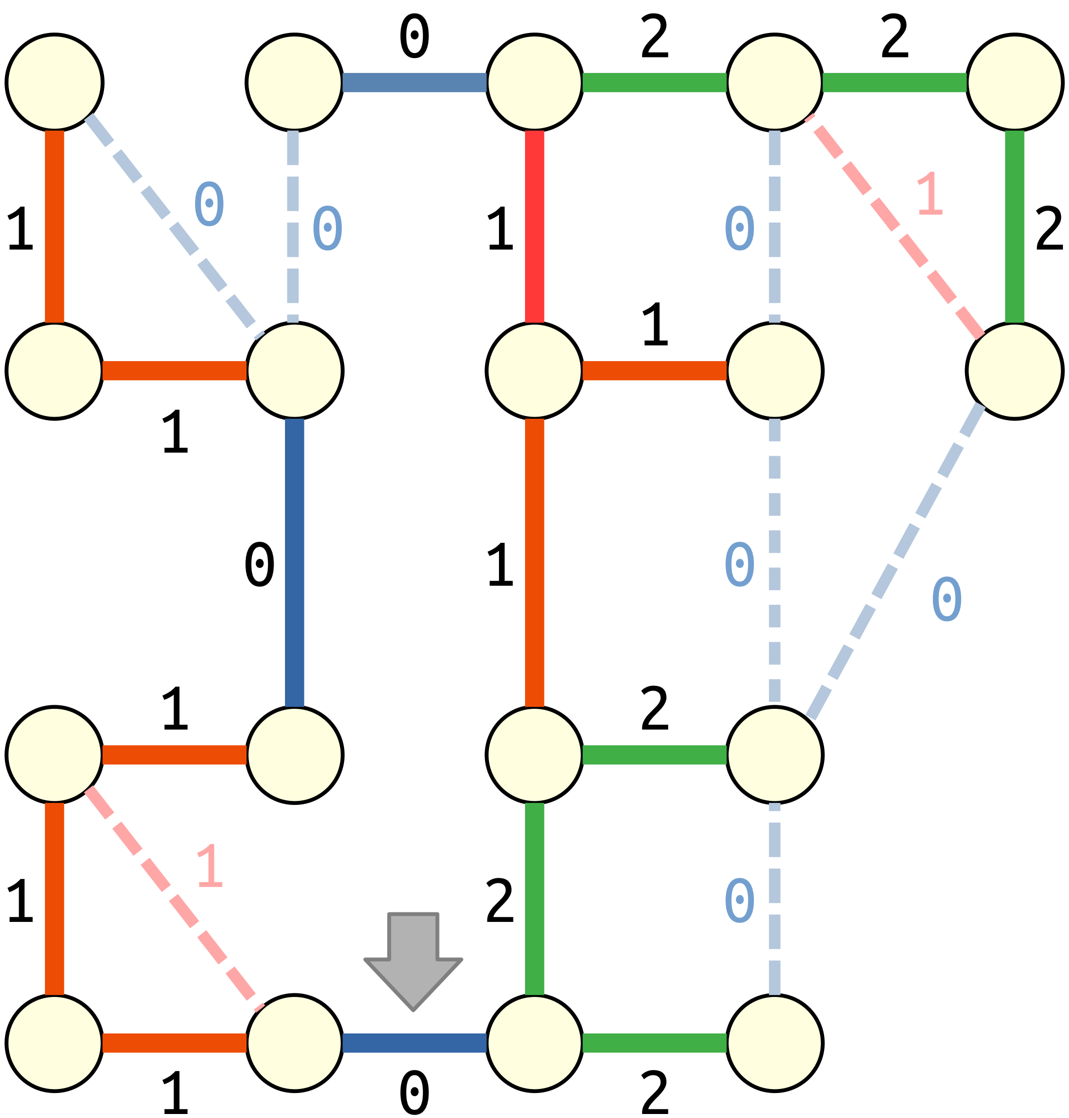


Suppose we add a new edge into the graph. What level should it get?

Answer: Give it level 0, which ensures we still have a MSF.



MSF Implications



Suppose we delete this edge of level 0.

Claim: No auxiliary edge of level 1 or higher can reconnect the tree.

Proof Idea: If such an edge existed, we would have used it in the MSF.

Tree Edge

Aux Edge

An Initial Idea

Invariant: $\mathscr{F}$ is a maximum spanning forest.

Does this work?
How fast is it?

To check ***are-connected***(x, y):

Return whether x and y are connected in $\mathscr{F}$.

To ***link***(x, y):

If ***are-connected***(x, y), add xy as an auxiliary edge to $\mathscr{F}$.

Otherwise add xy as a tree edge to $\mathscr{F}$.

To ***cut***(x, y), where xy is a tree edge of level l:

Delete xy from $\mathscr{F}$.

Let T_x and T_y be the trees in $\mathscr{F}$ containing x and y.

Select one of T_x and T_y arbitrarily; WLOG assume it's T_x.

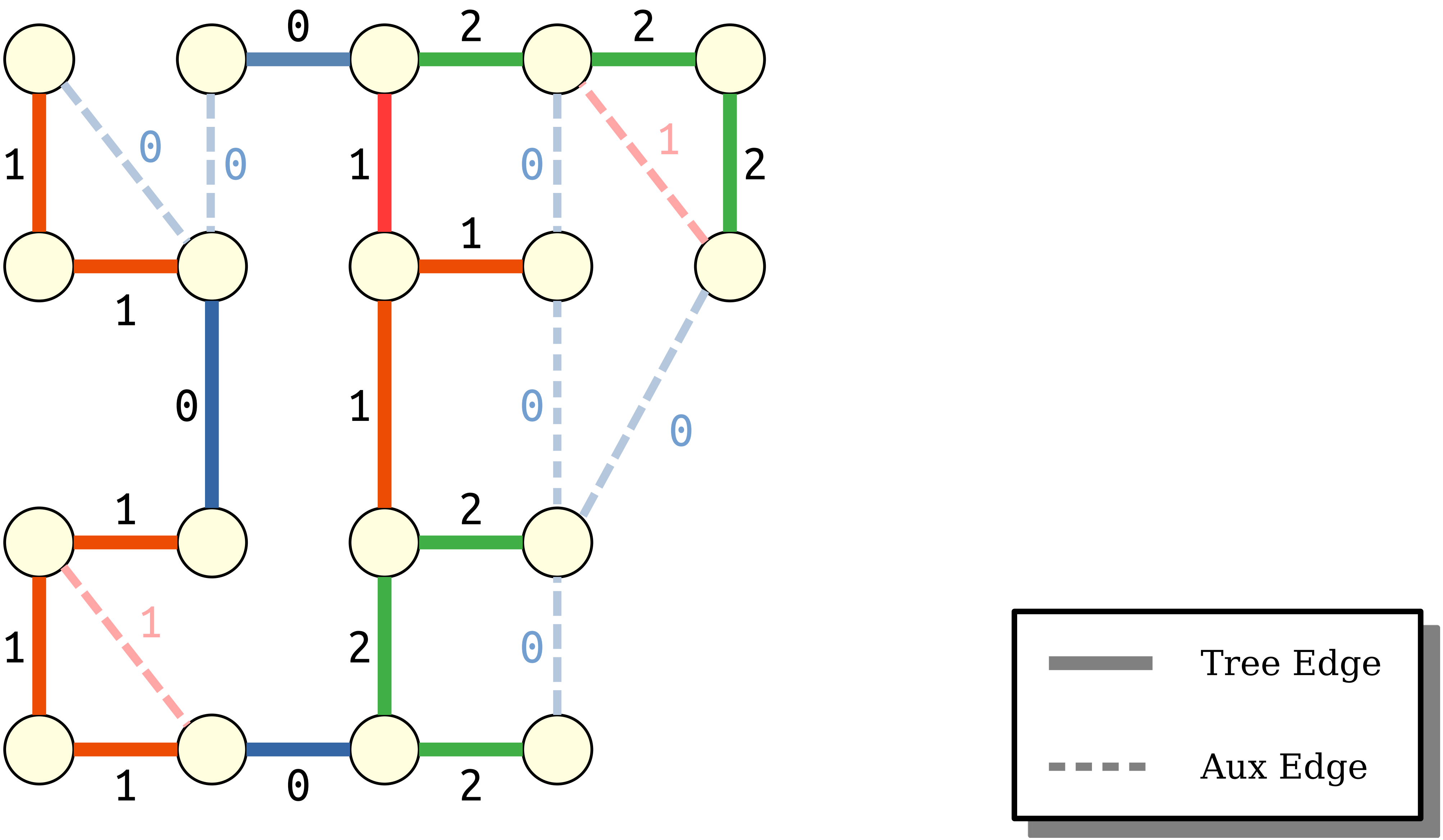
Increment the level of each tree edge of level l in T_x.

For each auxiliary edge uv in $\mathscr{F}$ touching T_x:

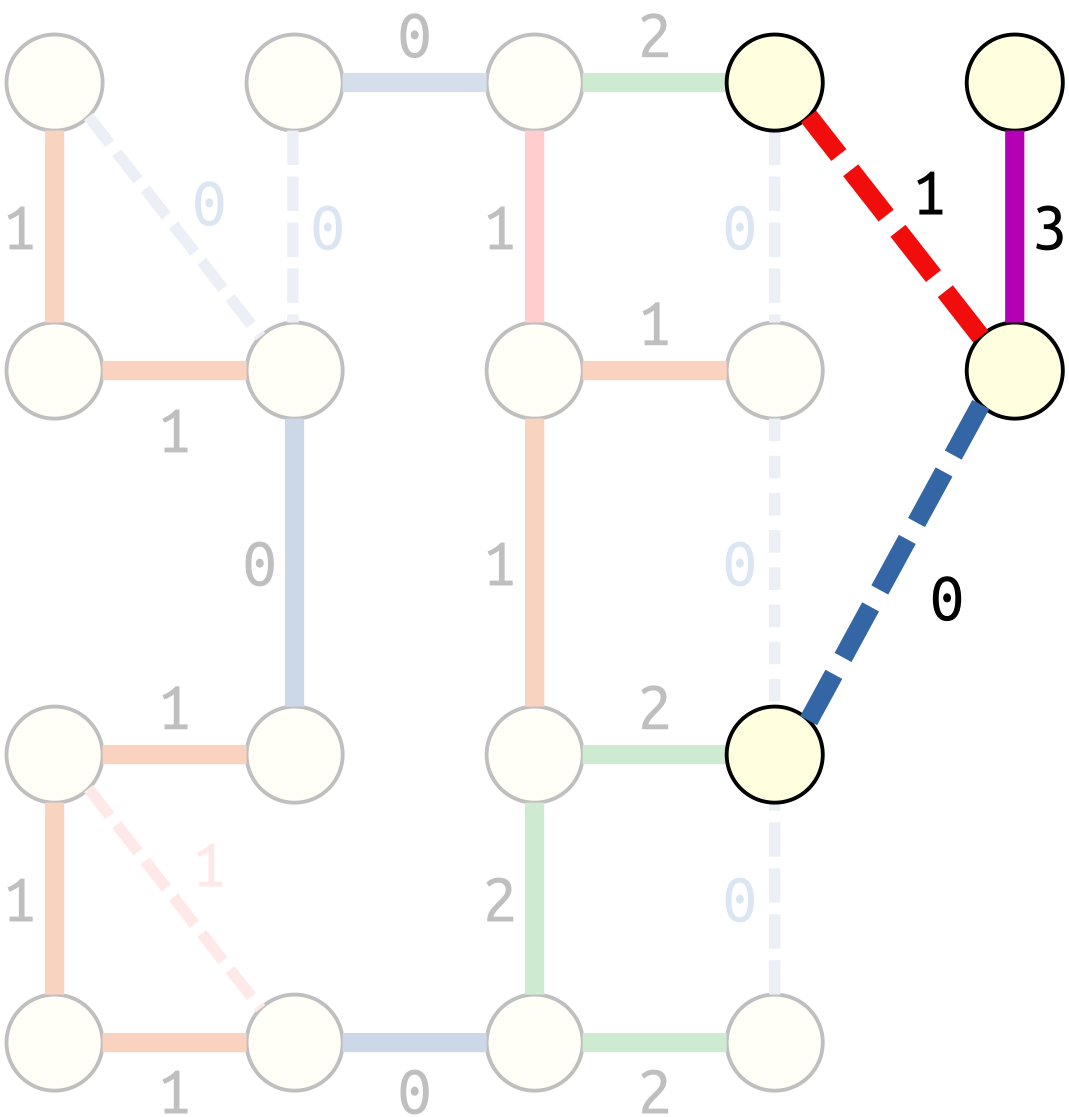
If uv connects T_x and T_y, add uv as a tree edge to $\mathscr{F}$. Stop.

Else increment its level.

An Important Detail

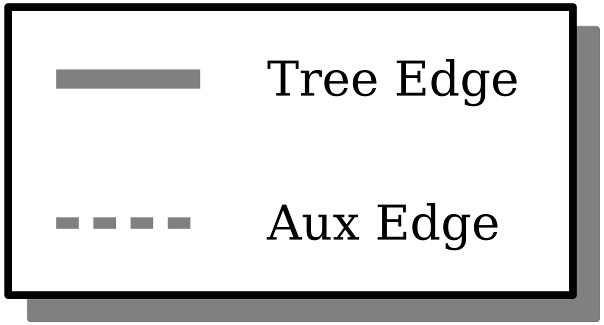


An Important Detail

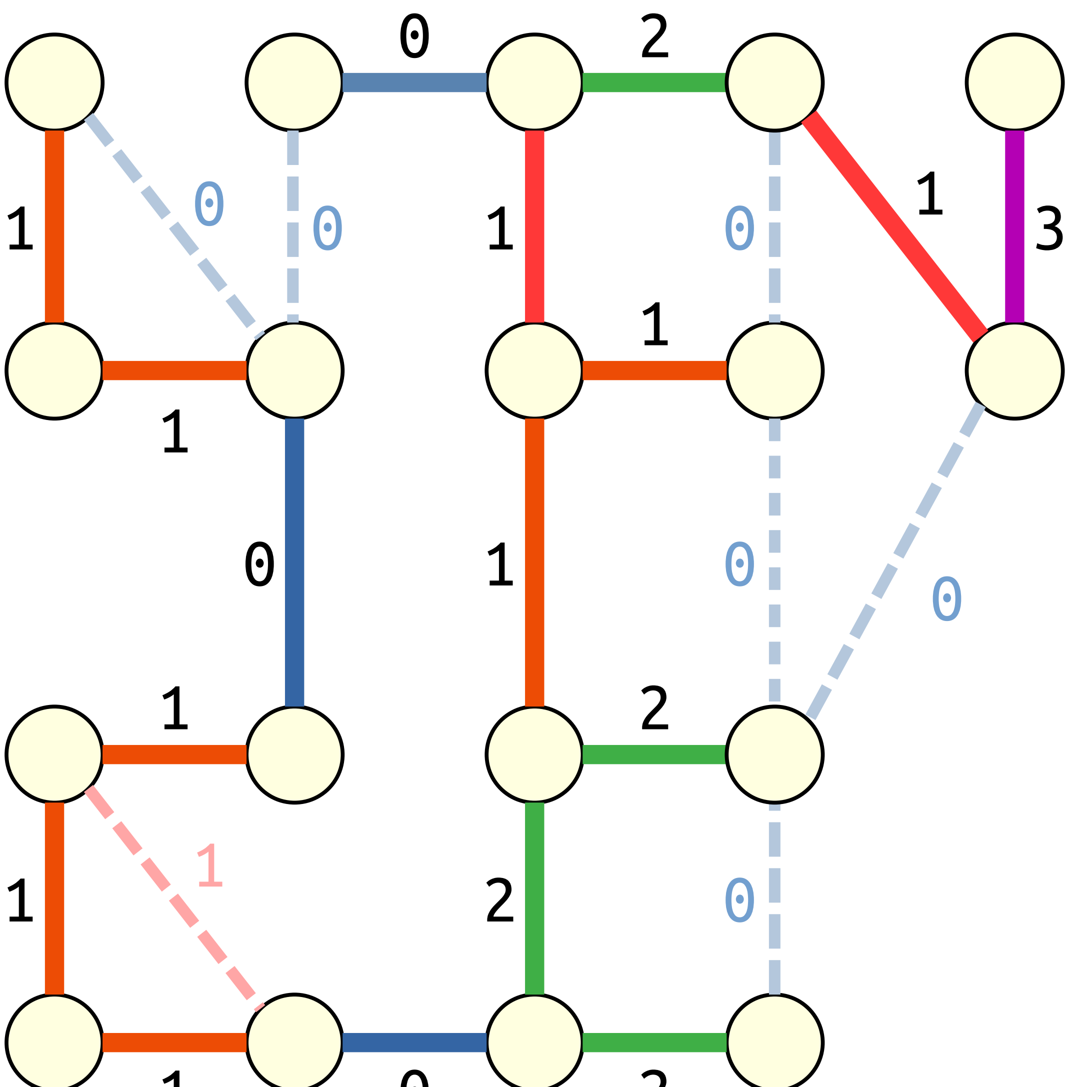


Both of these edges would reconnect. Which should we pick?

Answer: The edge of level 1, since we want to maintain an MSF.

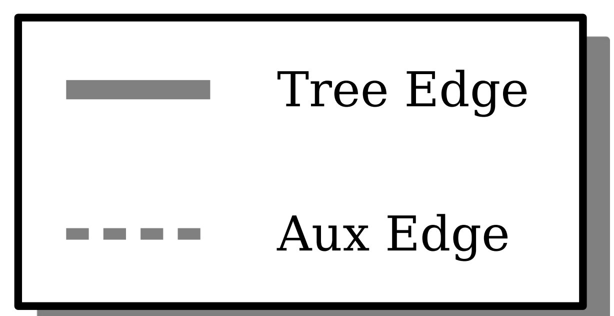


An Important Detail



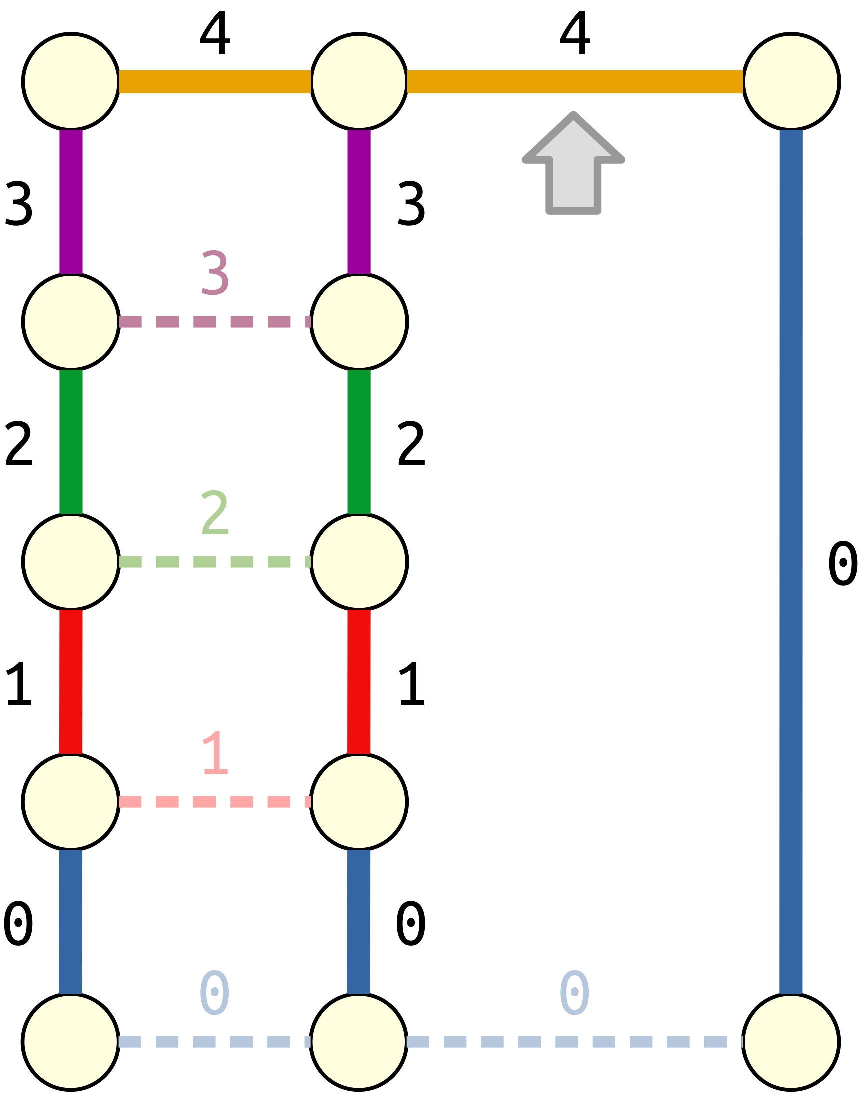
Both of these edges would reconnect. Which should we pick?

Answer: The edge of level 1, since we want to maintain an MSF.



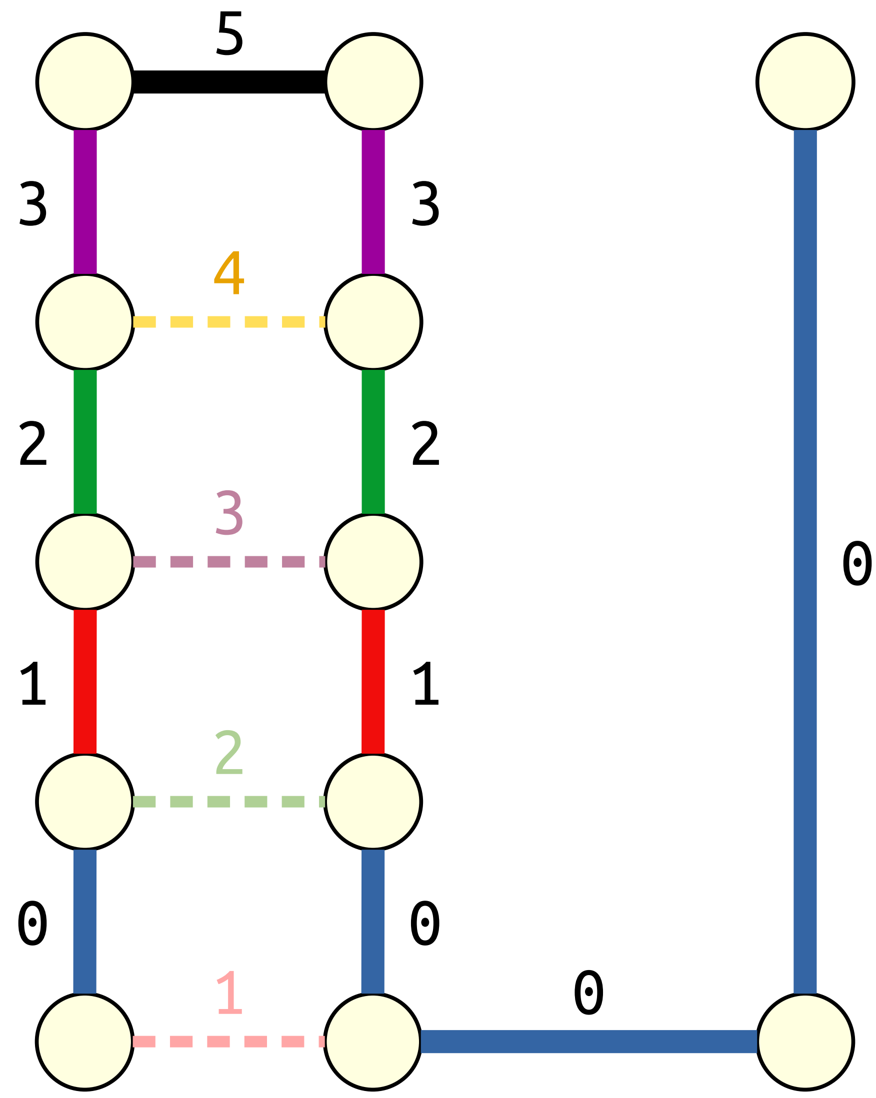
A Tricky Case

- Suppose we remove the indicated edge.



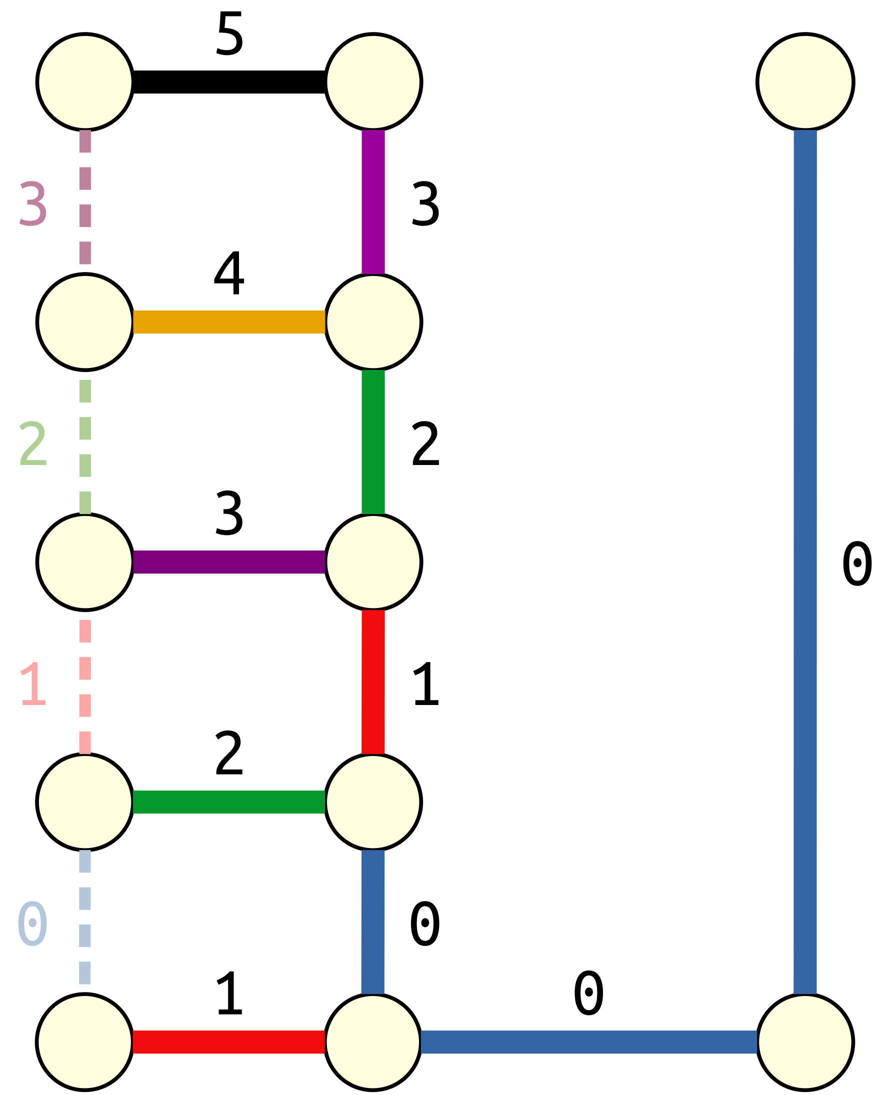
A Tricky Case

- Suppose we remove the indicated edge.
- Let's follow our standard procedure:
  - Increment all edges of level 4 in the left subtree.
  - Try reconnecting using auxiliary edges in decreasing level order.



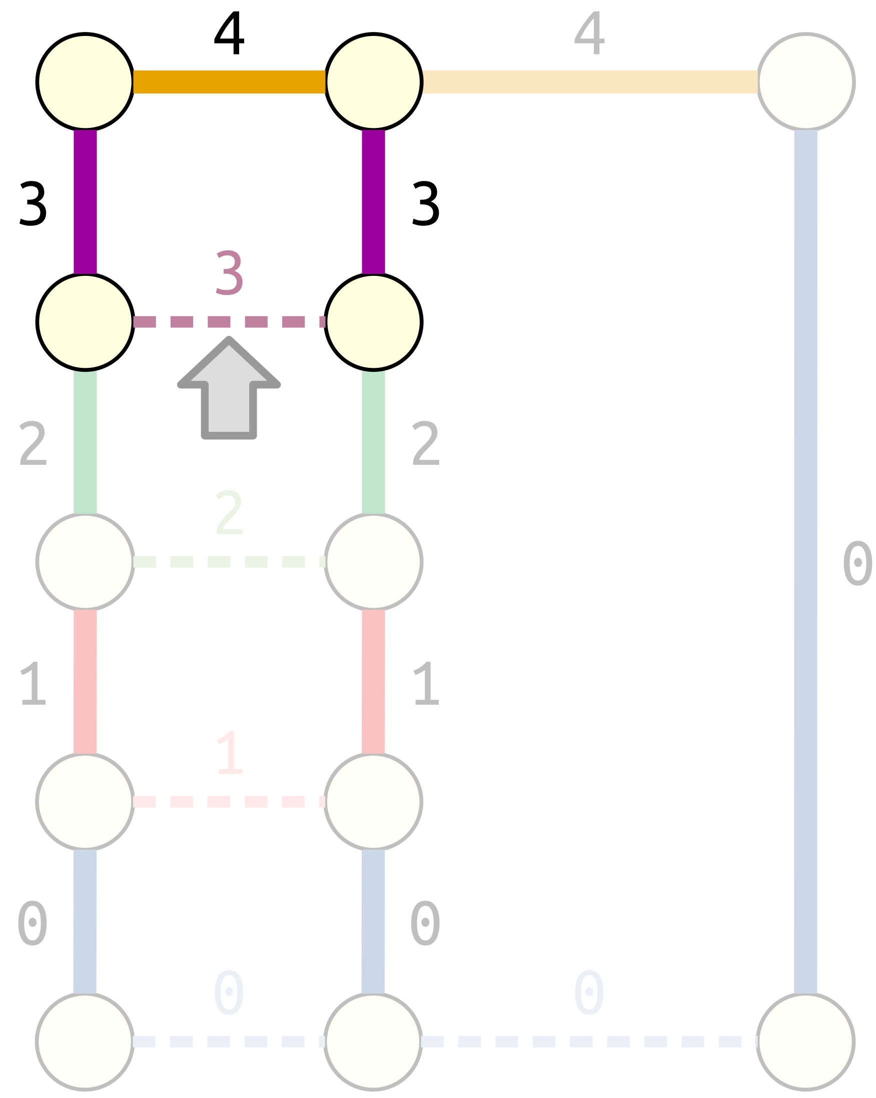
A Tricky Case

- Suppose we remove the indicated edge.
- Let's follow our standard procedure:
  - Increment all edges of level 4 in the left subtree.
  - Try reconnecting using auxiliary edges in decreasing level order.
- ***Problem:*** The resulting tree is not a maximum spanning forest.
- What went wrong?



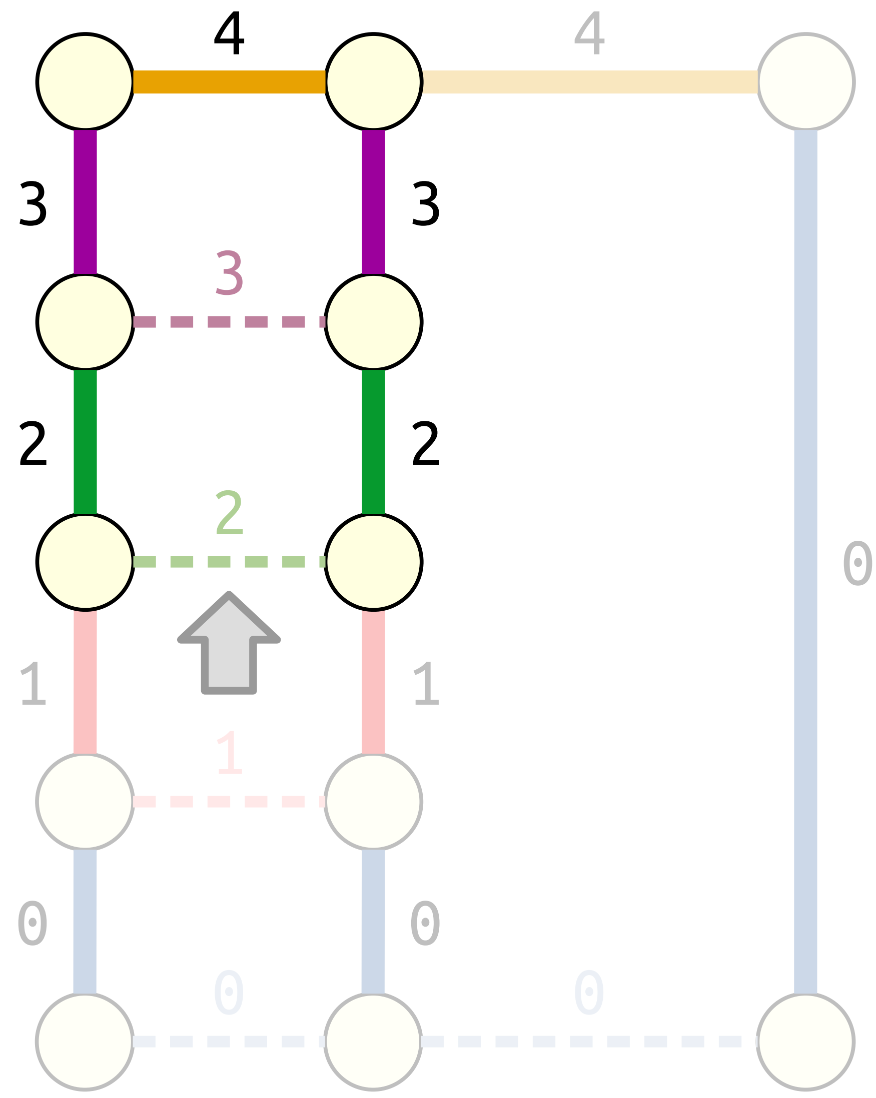
A Tricky Case

- Pick an auxiliary edge of level *l*. Why wasn't it in the MSF?
- Adding it must close a cycle where it's (tied for) the cheapest edge.



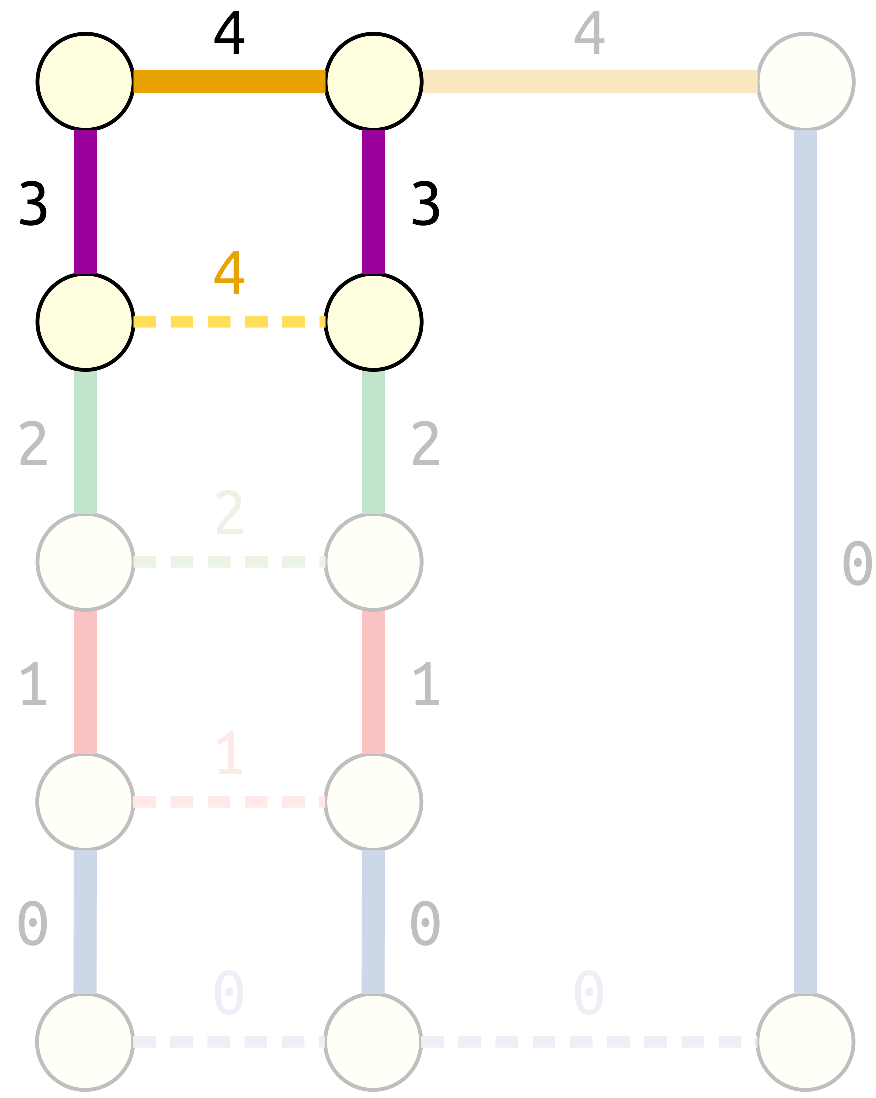
A Tricky Case

- Pick an auxiliary edge of level *l*. Why wasn't it in the MSF?
- Adding it must close a cycle where it's (tied for) the cheapest edge.



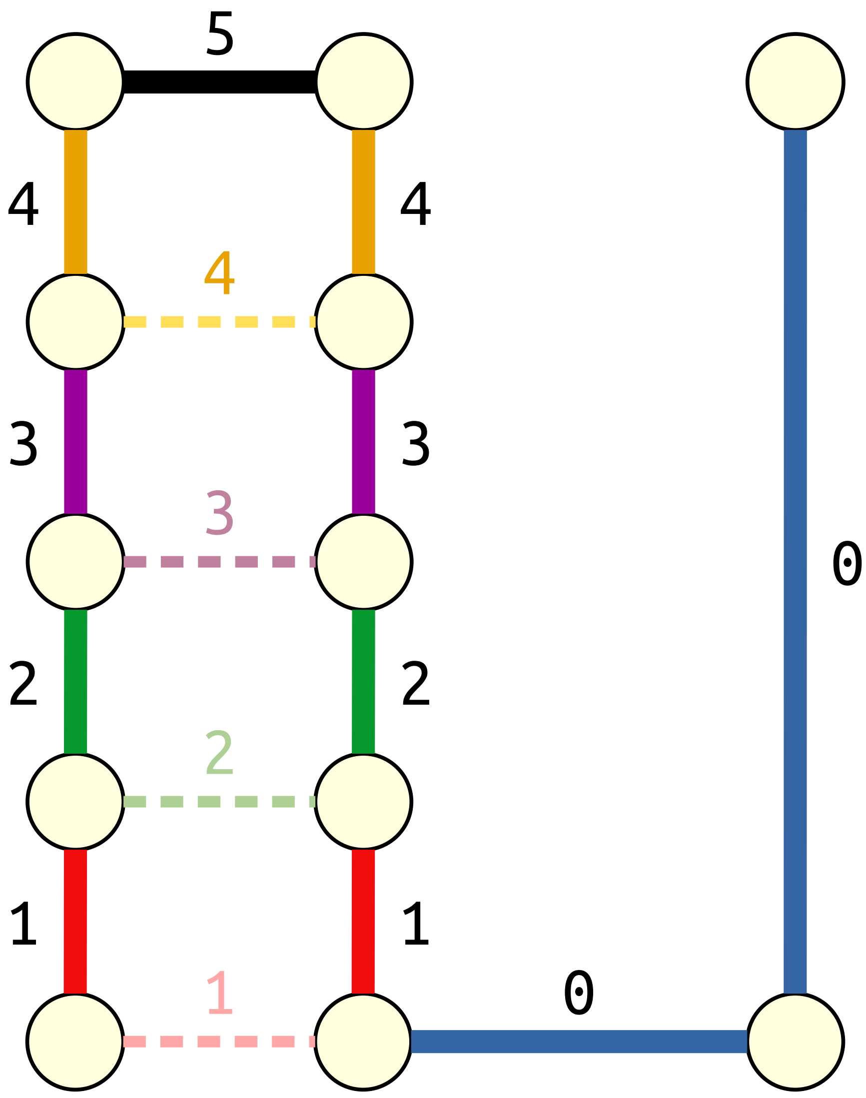
A Tricky Case

- Pick an auxiliary edge of level l. Why wasn't it in the MSF?
- Adding it must close a cycle where it's (tied for) the cheapest edge.
- Thus the MSF cycle containing the edge must be made of edges of levels l or higher.
- If we increment the level of an auxiliary edge from level l to $l+1$, this is no longer guaranteed.



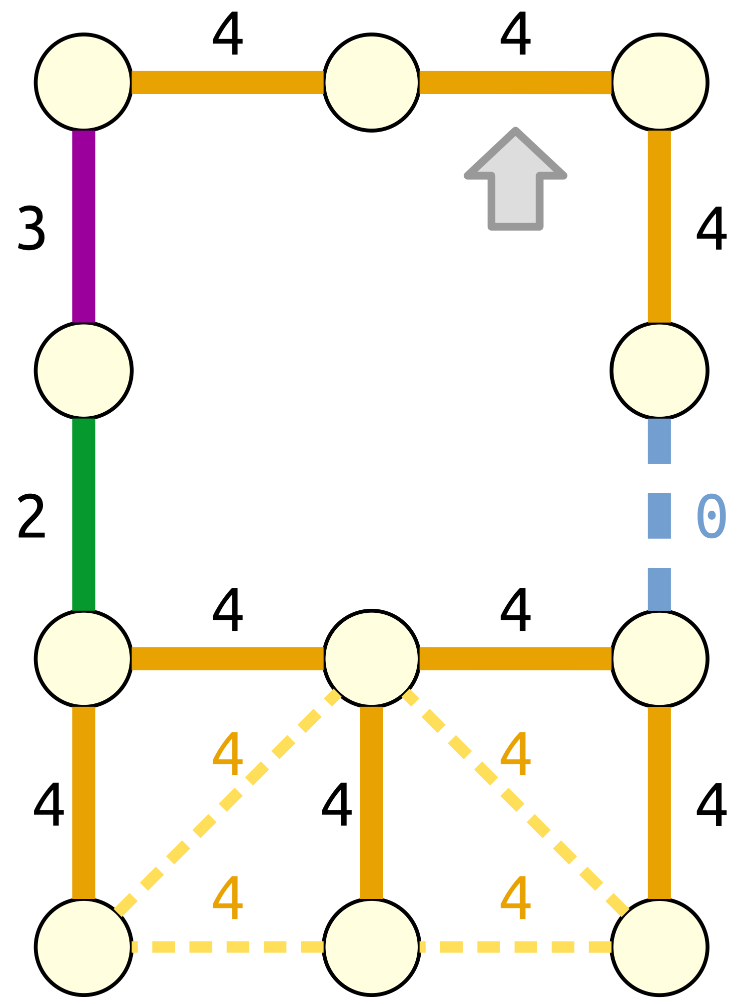
Resolving the Issue

- When we cut an edge of level l, we already increment the level of all edges of level l in one of the new trees.
- ***Revised Rule:*** If we fail to reconnect at level l, when proceeding to level l – 1, increment all edges of level l – 1 before trying to reconnect.
  - This preserves the MSF property. *(Why?)*
  - This ensures that auxiliary edges, when incremented, preserve the MSF property.



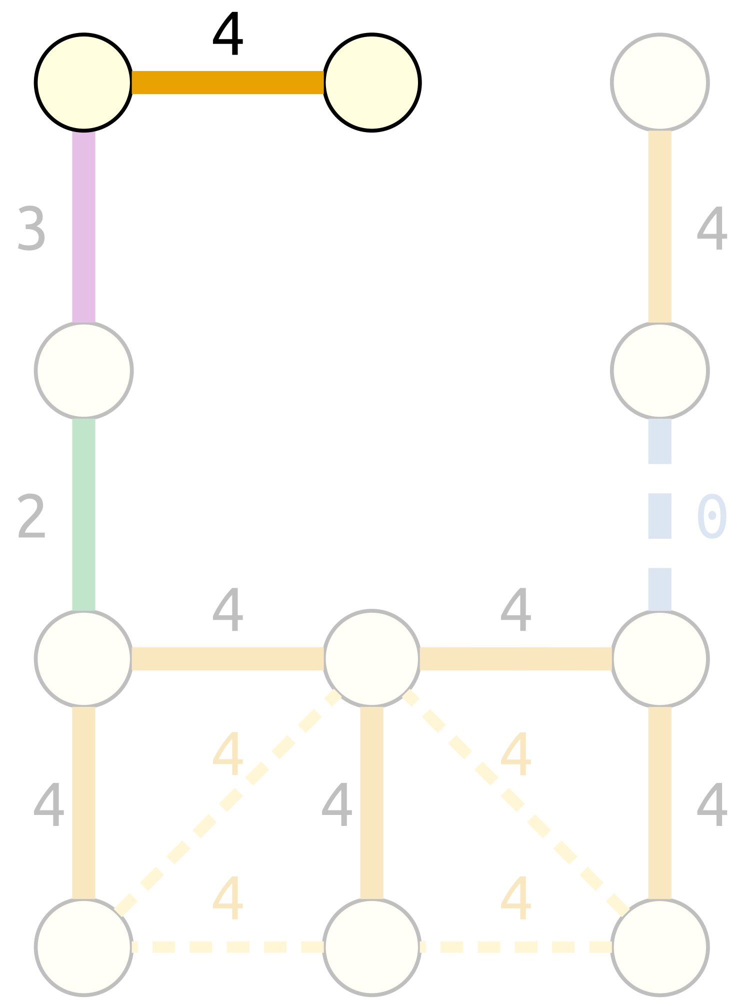
An Interesting Case

- Suppose we delete the indicated edge.
- ***Claim:*** None of the edges of level 4 in the bottom region can reconnect the trees.
- ***Why?***
  - If any of them reconnected, they would close a cycle in the original tree using an edge of weight 2.
  - But then we didn't have an MSF – we should have used the edge of level 4.
- What can we do with this insight?



An Interesting Case

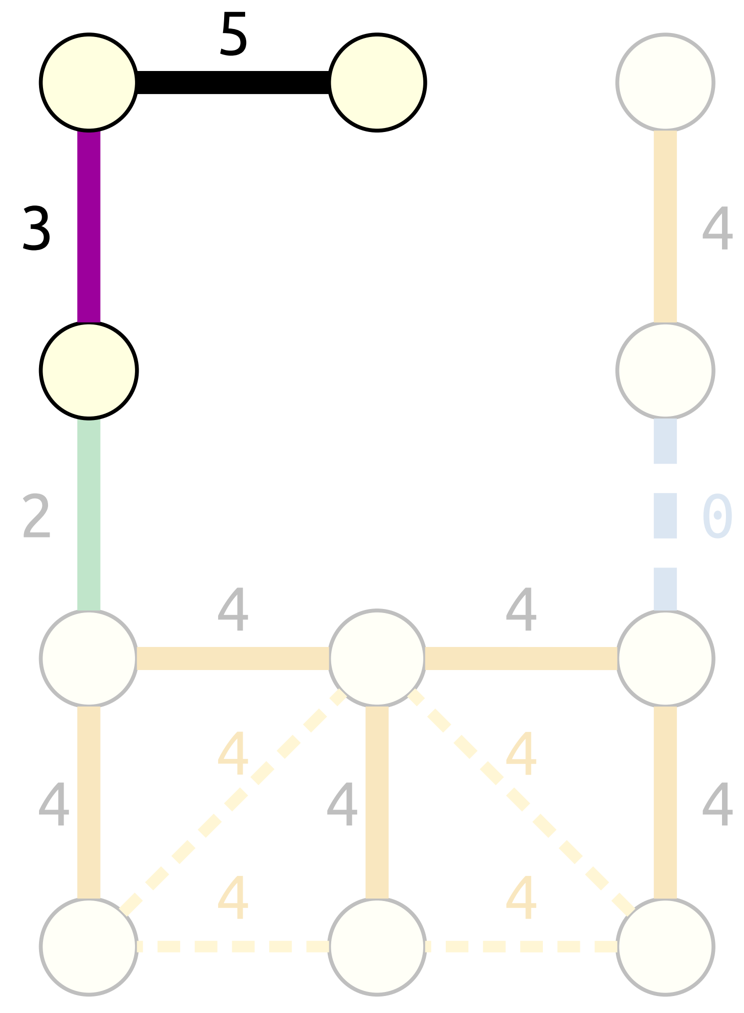
- ***Idea:*** After cutting an edge of level *l*, don't look at the full subtrees formed. Instead just look at subtrees using edges of levels *l* and above.
- Use the same algorithms as before, except restricted to those trees.



Level 4

An Interesting Case

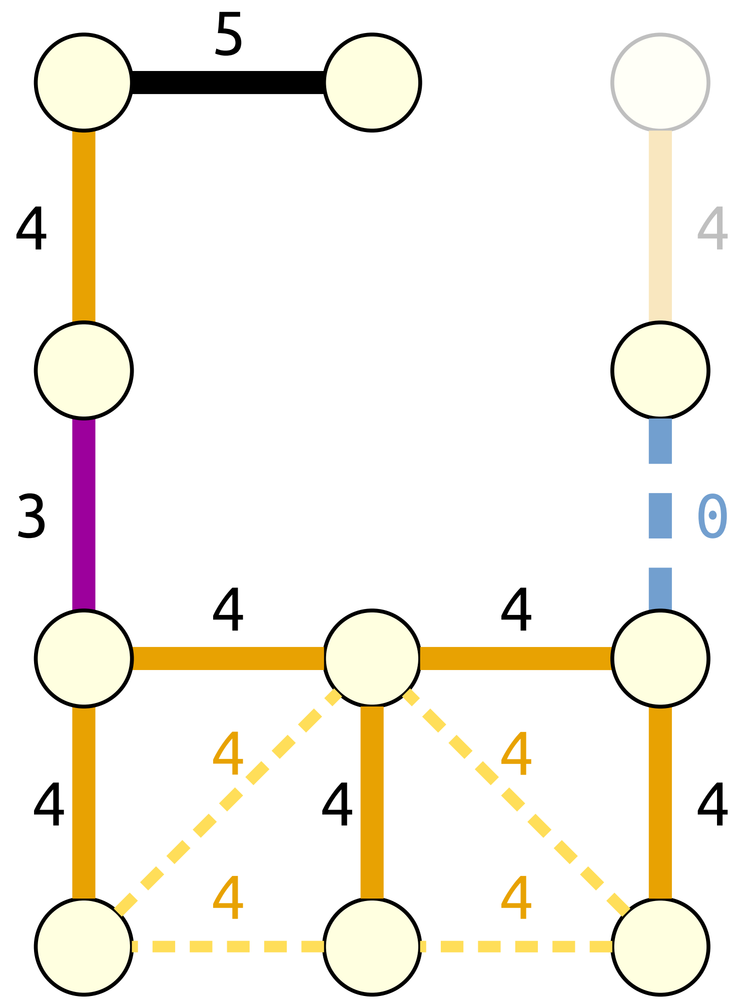
- ***Idea:*** After cutting an edge of level *l*, don't look at the full subtrees formed. Instead just look at subtrees using edges of levels *l* and above.
- Use the same algorithms as before, except restricted to those trees.



Level 3

An Interesting Case

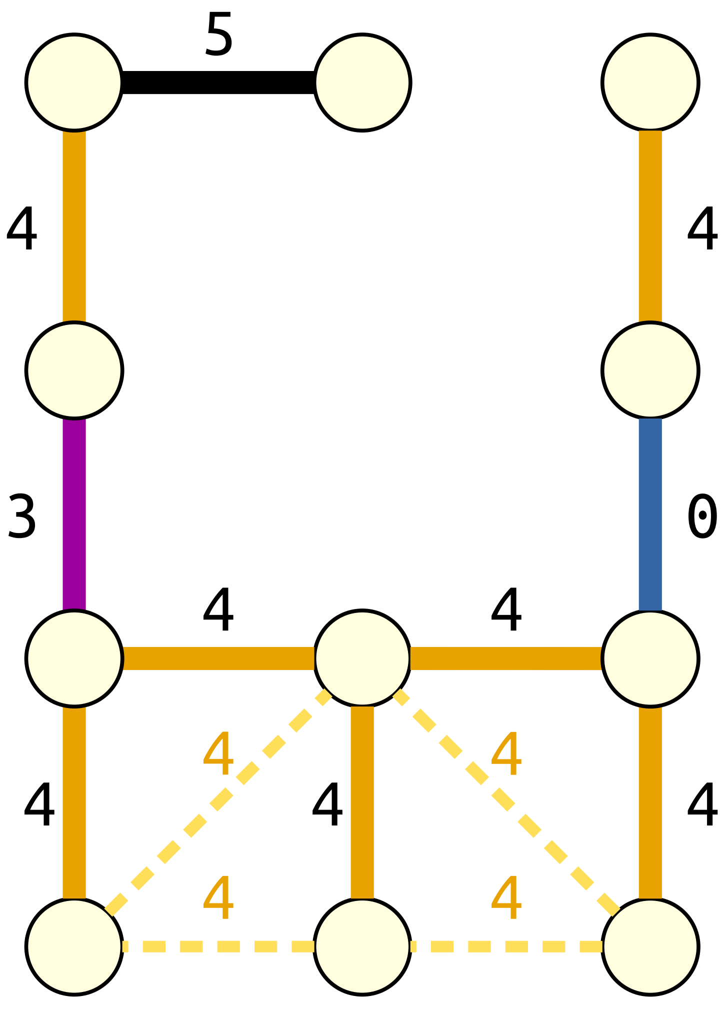
- ***Idea:*** After cutting an edge of level *l*, don't look at the full subtrees formed. Instead just look at subtrees using edges of levels *l* and above.
- Use the same algorithms as before, except restricted to those trees.



Level 2

An Interesting Case

- ***Idea:*** After cutting an edge of level *l*, don't look at the full subtrees formed. Instead just look at subtrees using edges of levels *l* and above.
- Use the same algorithms as before, except restricted to those trees.

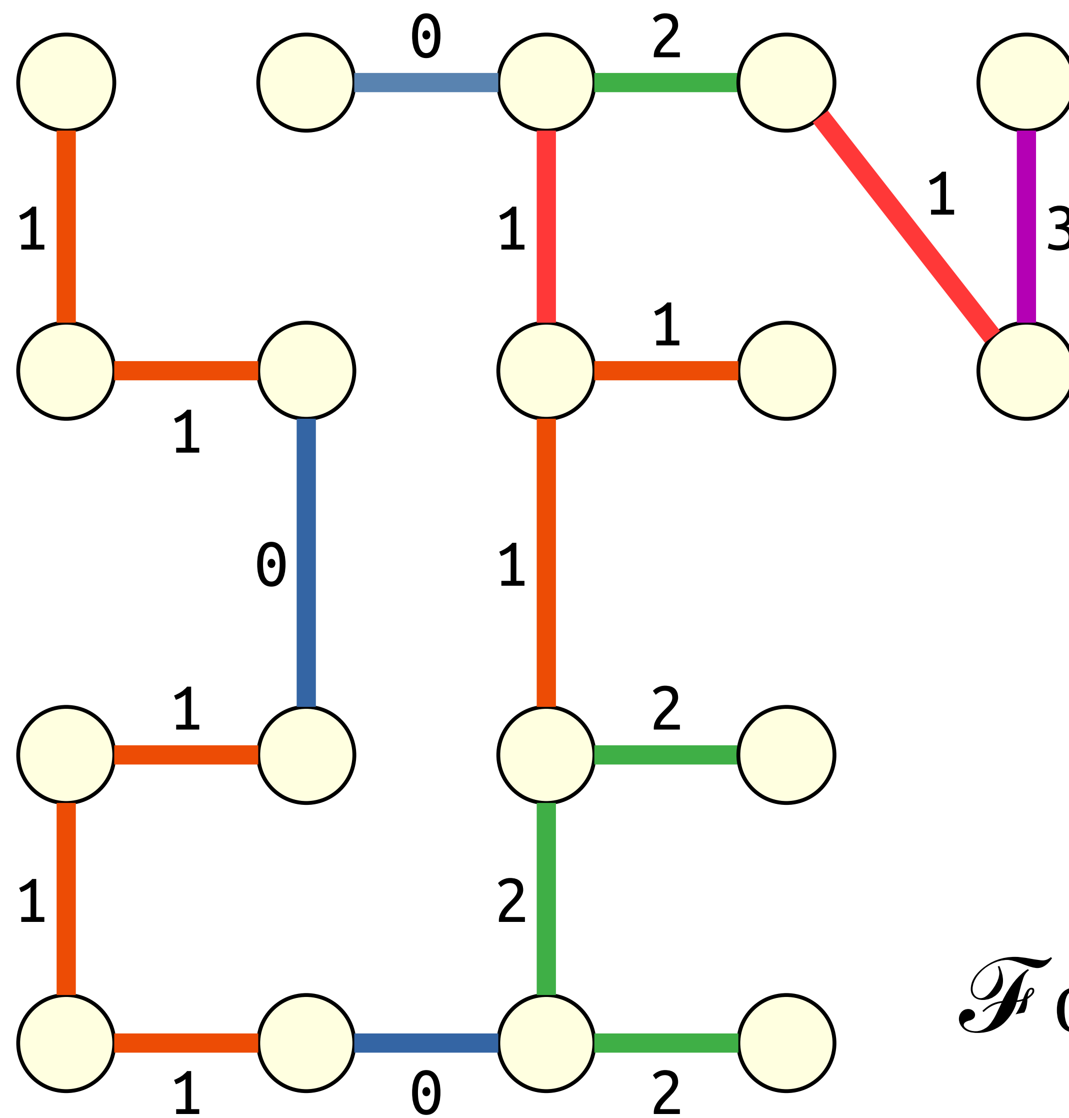


To Recap

- To maintain our MSF, we need to do the following:
  - If we cut an edge at level l, we need to search for auxiliary edges first at level l, then l – 1, then l – 2, etc. Otherwise the edge we add might not result in an MSF.
  - When searching for auxiliary edges, we only need to search the part of the tree reachable by edges of level i or higher. Any other auxiliary edges in the tree can't possibly be part of the MSF.
  - For each level i, before we search for auxiliary edges, we need to increase the level of all tree edges at level i to level i+1. Otherwise when an edge fails to reconnect and we boost its level, we might not get an MSF.
- Putting this all together:
  - We need a mechanism to quickly find all auxiliary edges of a given level, in a subtree reachable using only edges of a given level or higher, while being able to find all tree edges of a given level quickly.

Layered Forests

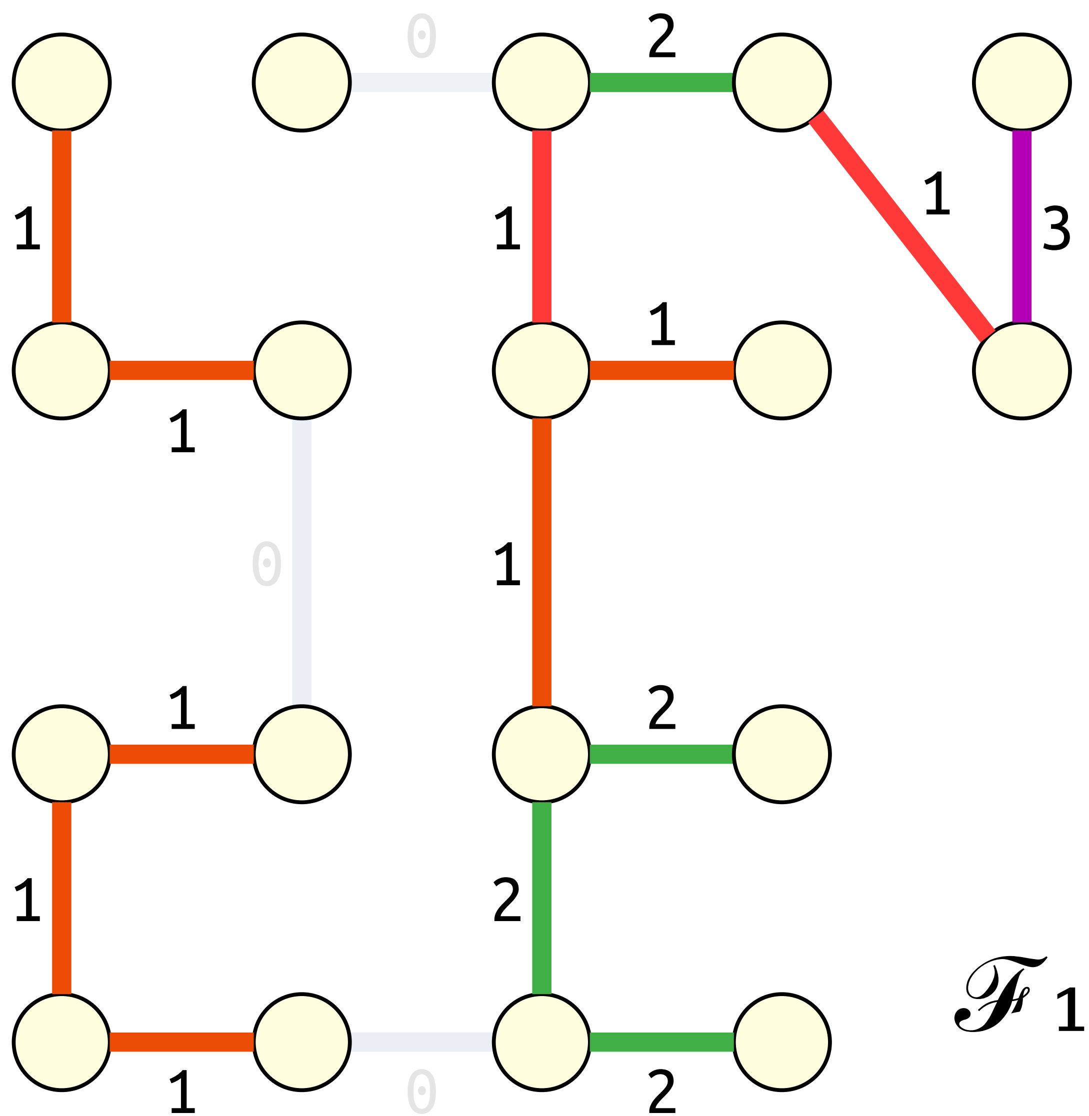
- ***Idea:*** Store multiple versions of the forest, each focusing on edges of some level or above.
- Let $\mathscr{F}_l$ to be the forest of all edges of level l or higher.
- We maintain a series of forests $\mathscr{F}_0 \supseteq \mathscr{F}_1 \supseteq \mathscr{F}_2 \supseteq \ldots$, with one forest per level.
- Each edge of level l then appears in all forests of level l and below.



$\mathscr{F}_0$

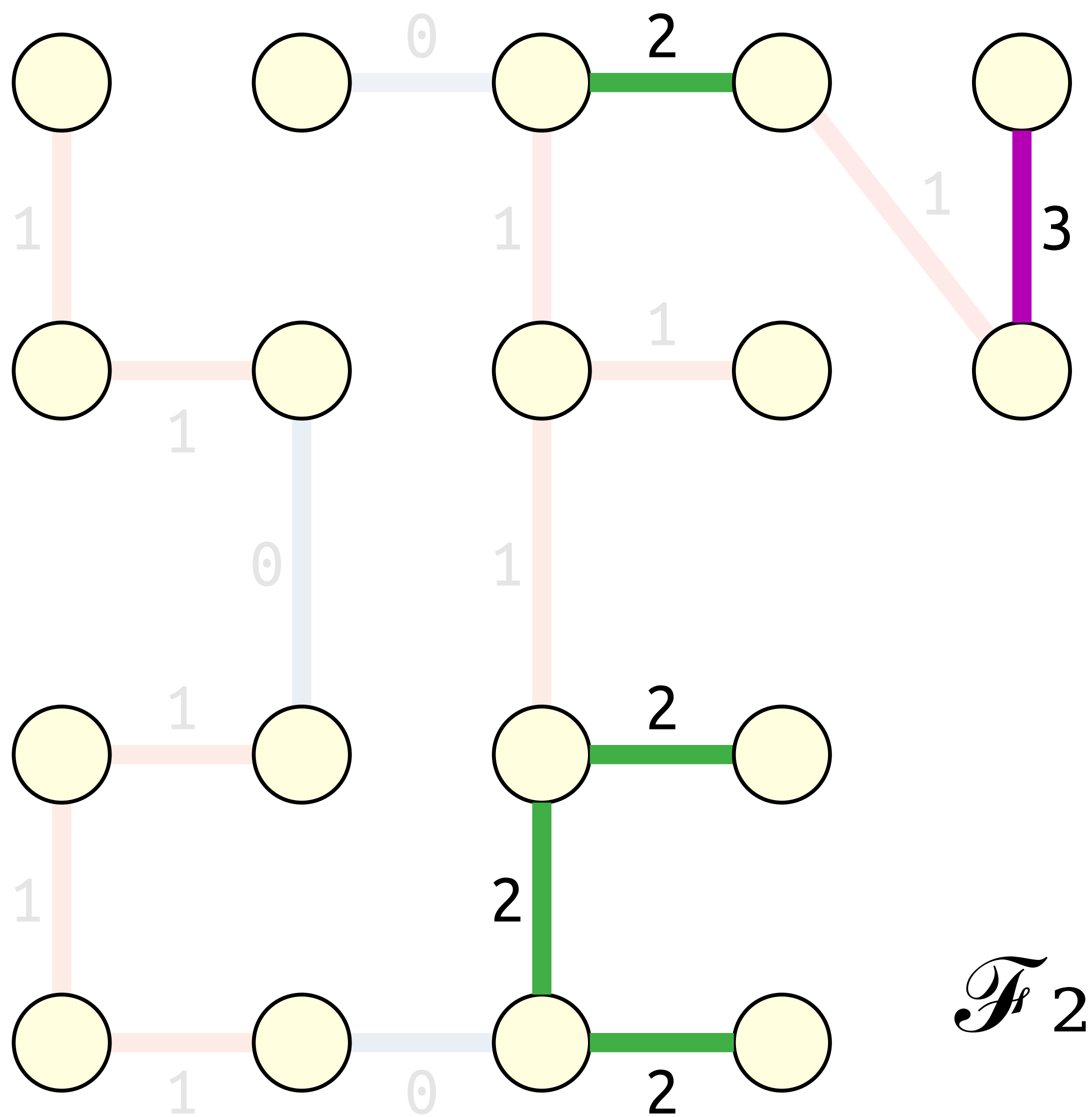
Layered Forests

- ***Idea:*** Store multiple versions of the forest, each focusing on edges of some level or above.
- Let $\mathscr{F}_l$ to be the forest of all edges of level l or higher.
- We maintain a series of forests $\mathscr{F}_0 \supseteq \mathscr{F}_1 \supseteq \mathscr{F}_2 \supseteq \ldots$, with one forest per level.
- Each edge of level l then appears in all forests of level l and below.



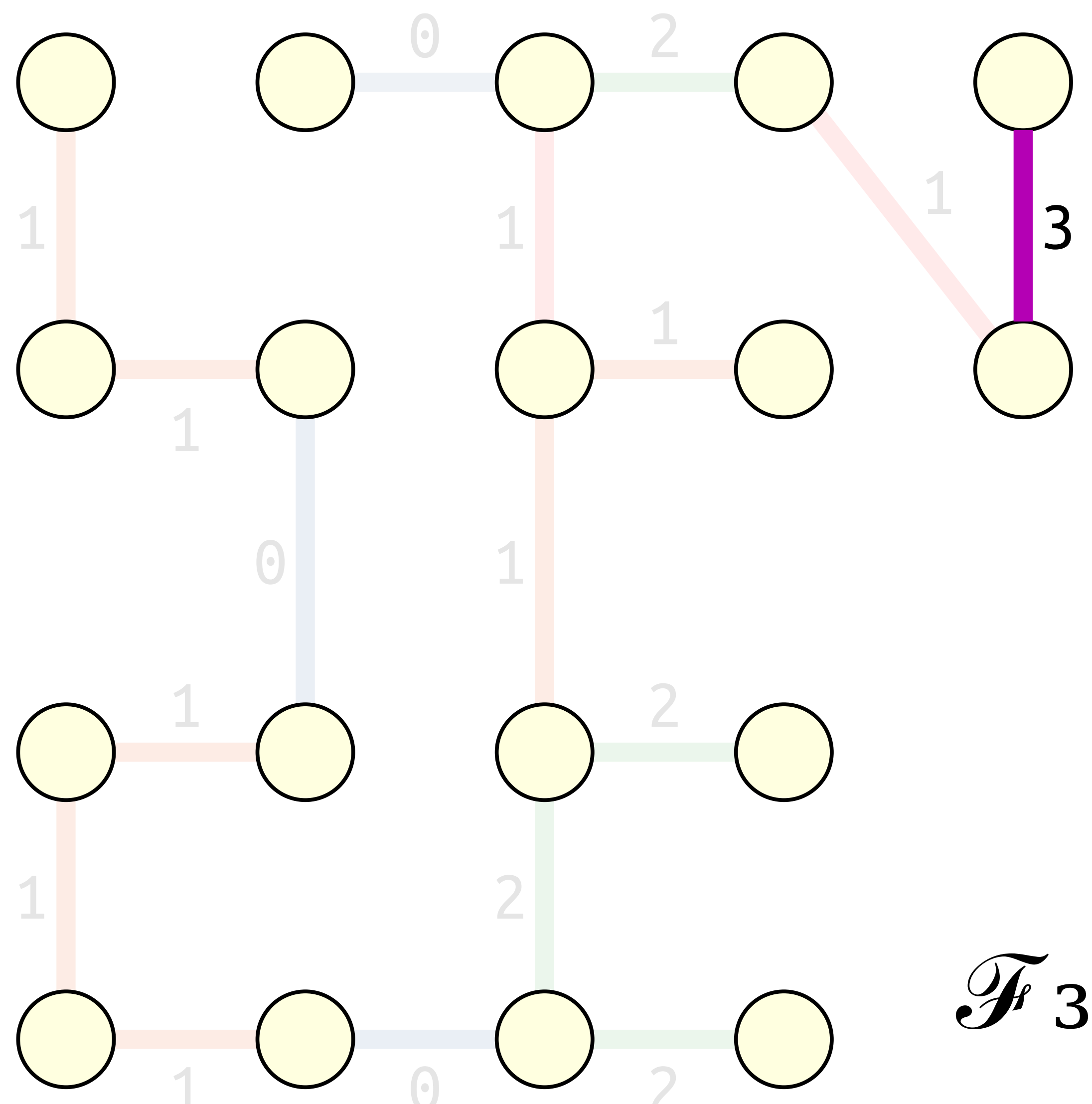
Layered Forests

- ***Idea:*** Store multiple versions of the forest, each focusing on edges of some level or above.
- Let $\mathscr{F}_l$ to be the forest of all edges of level l or higher.
- We maintain a series of forests $\mathscr{F}_0 \supseteq \mathscr{F}_1 \supseteq \mathscr{F}_2 \supseteq \ldots$, with one forest per level.
- Each edge of level l then appears in all forests of level l and below.



Layered Forests

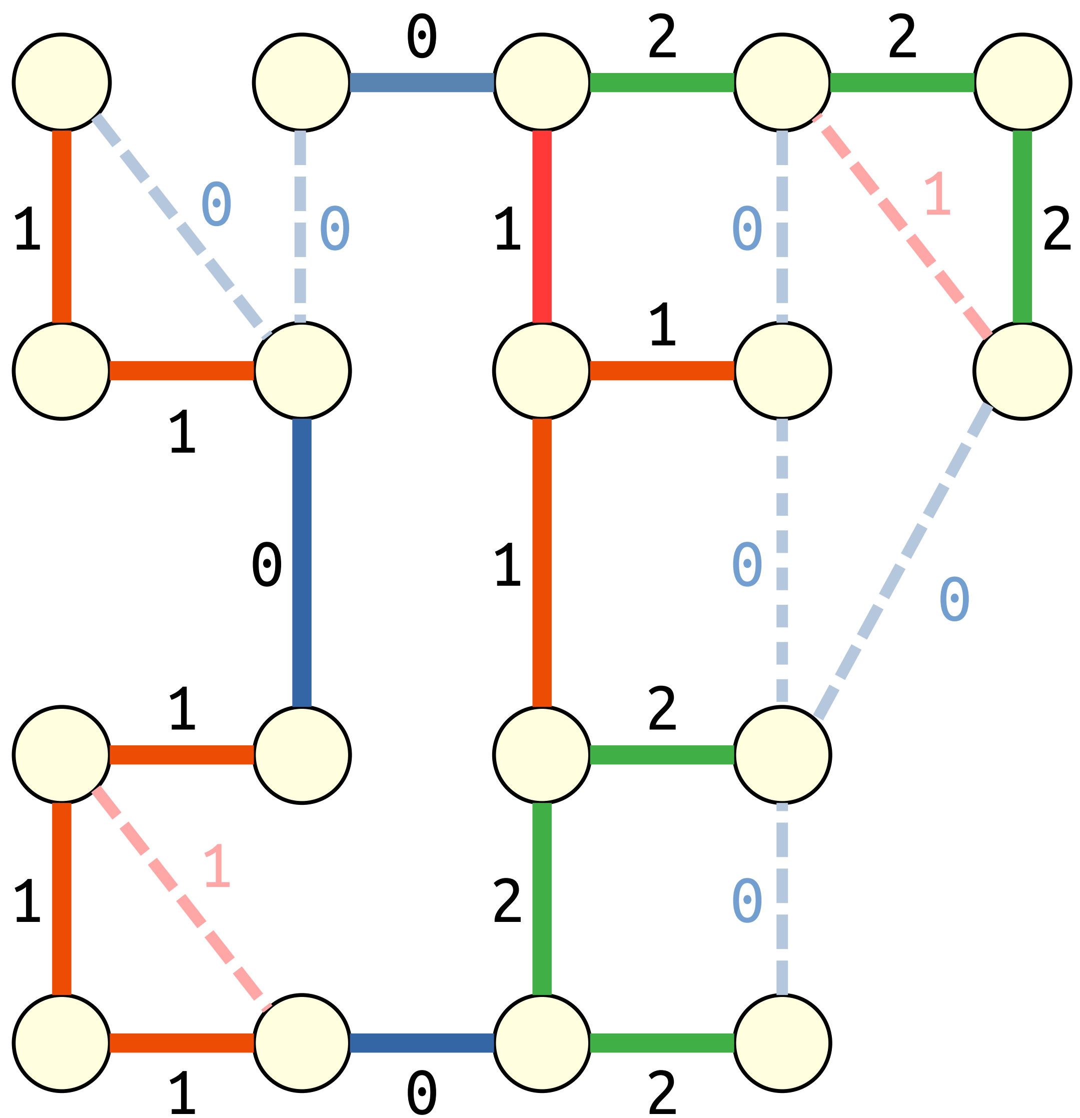
- ***Idea:*** Store multiple versions of the forest, each focusing on edges of some level or above.
- Let $\mathscr{F}_l$ to be the forest of all edges of level l or higher.
- We maintain a series of forests $\mathscr{F}_0 \supseteq \mathscr{F}_1 \supseteq \mathscr{F}_2 \supseteq \ldots$, with one forest per level.
- Each edge of level l then appears in all forests of level l and below.



$\mathscr{F}_3$

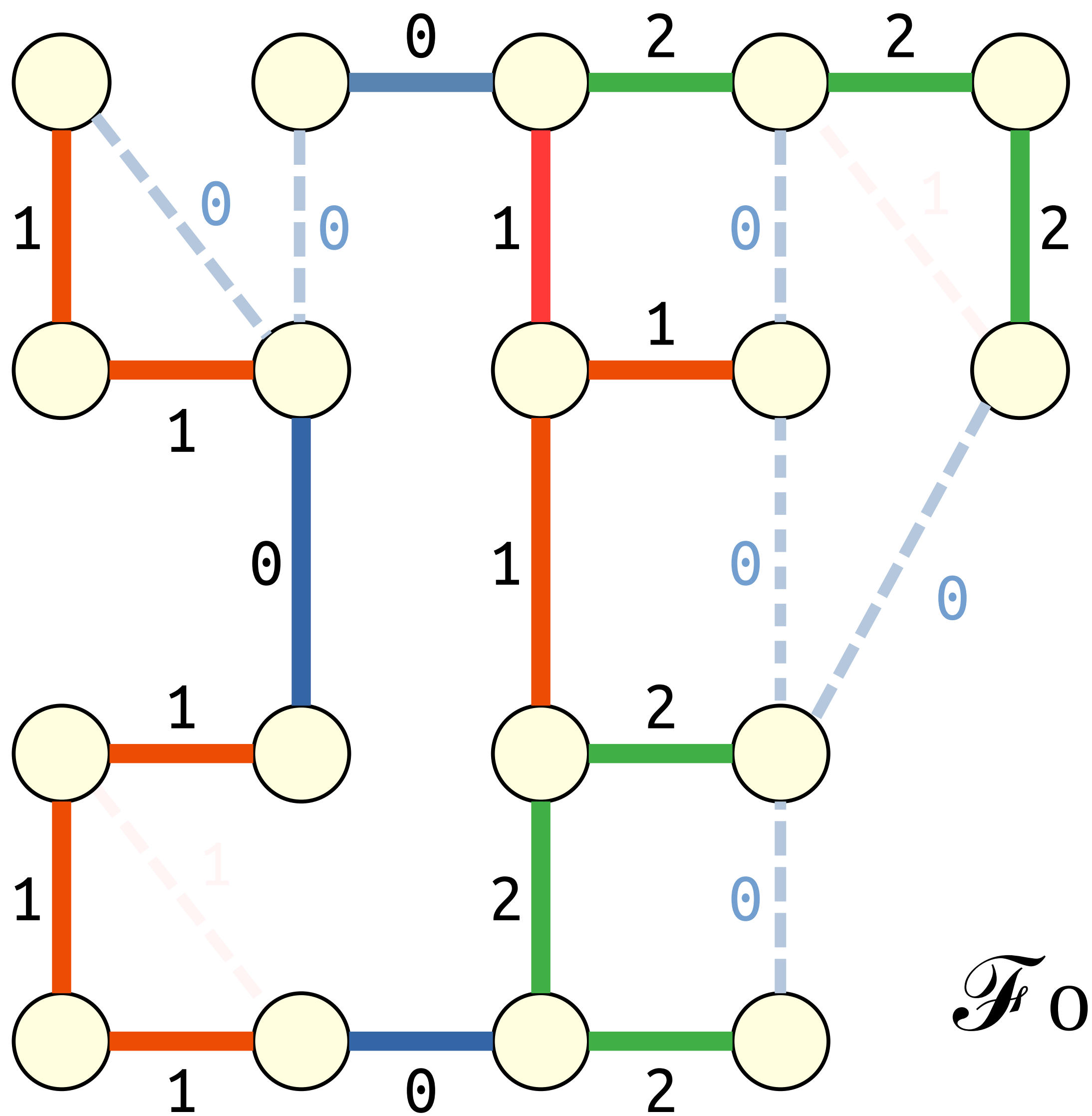
Layered Forests

- To make it faster to find auxiliary edges, each auxiliary edge of level l will be stored attached only to $\mathscr{F}_l$.
- After all, we only need to look for auxiliary edges of level l when we're focusing on trees made of edges of level l or above.



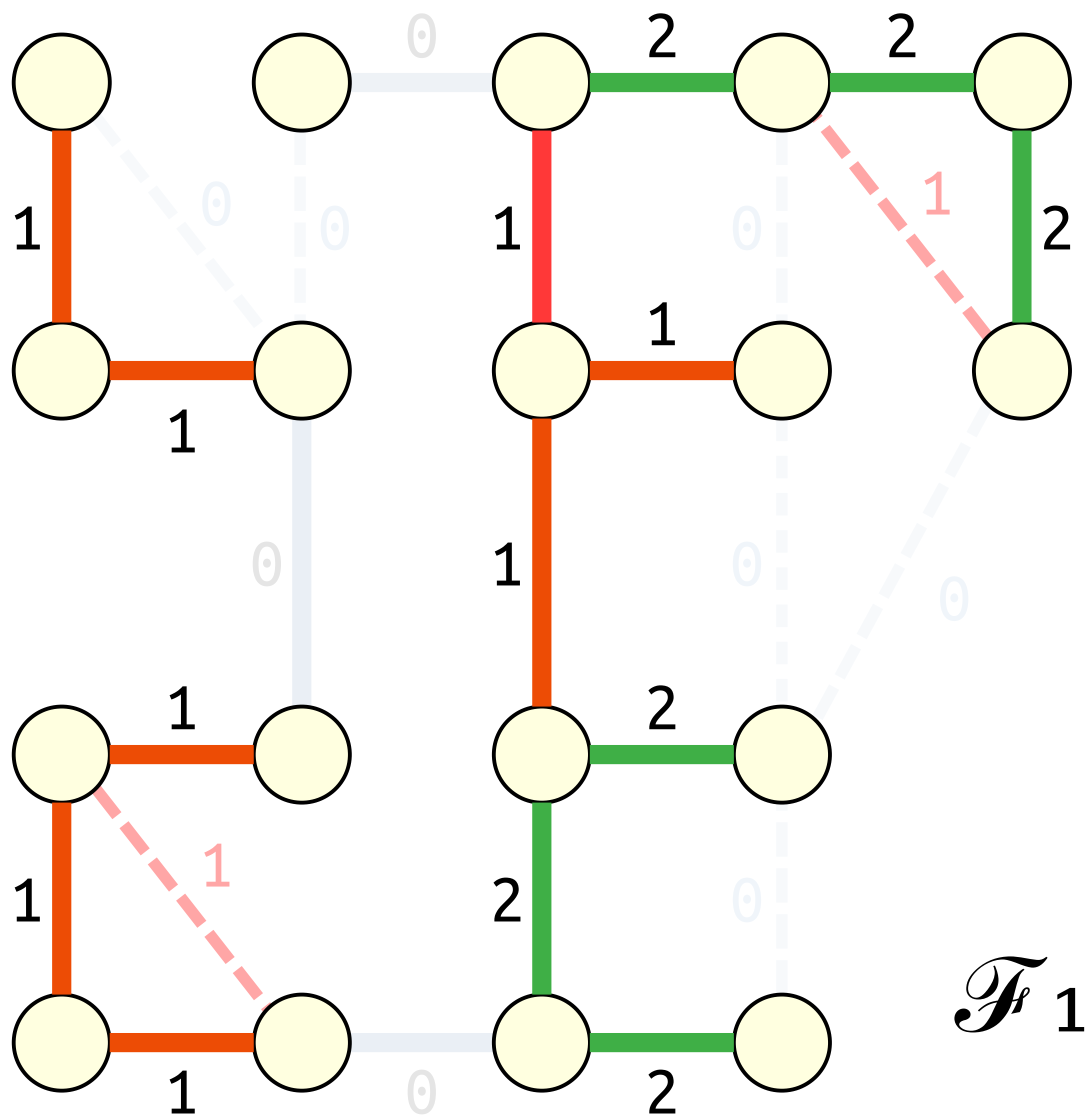
Layered Forests

- To make it faster to find auxiliary edges, each auxiliary edge of level l will be stored attached only to $\mathscr{F}_l$.
- After all, we only need to look for auxiliary edges of level l when we're focusing on trees made of edges of level l or above.



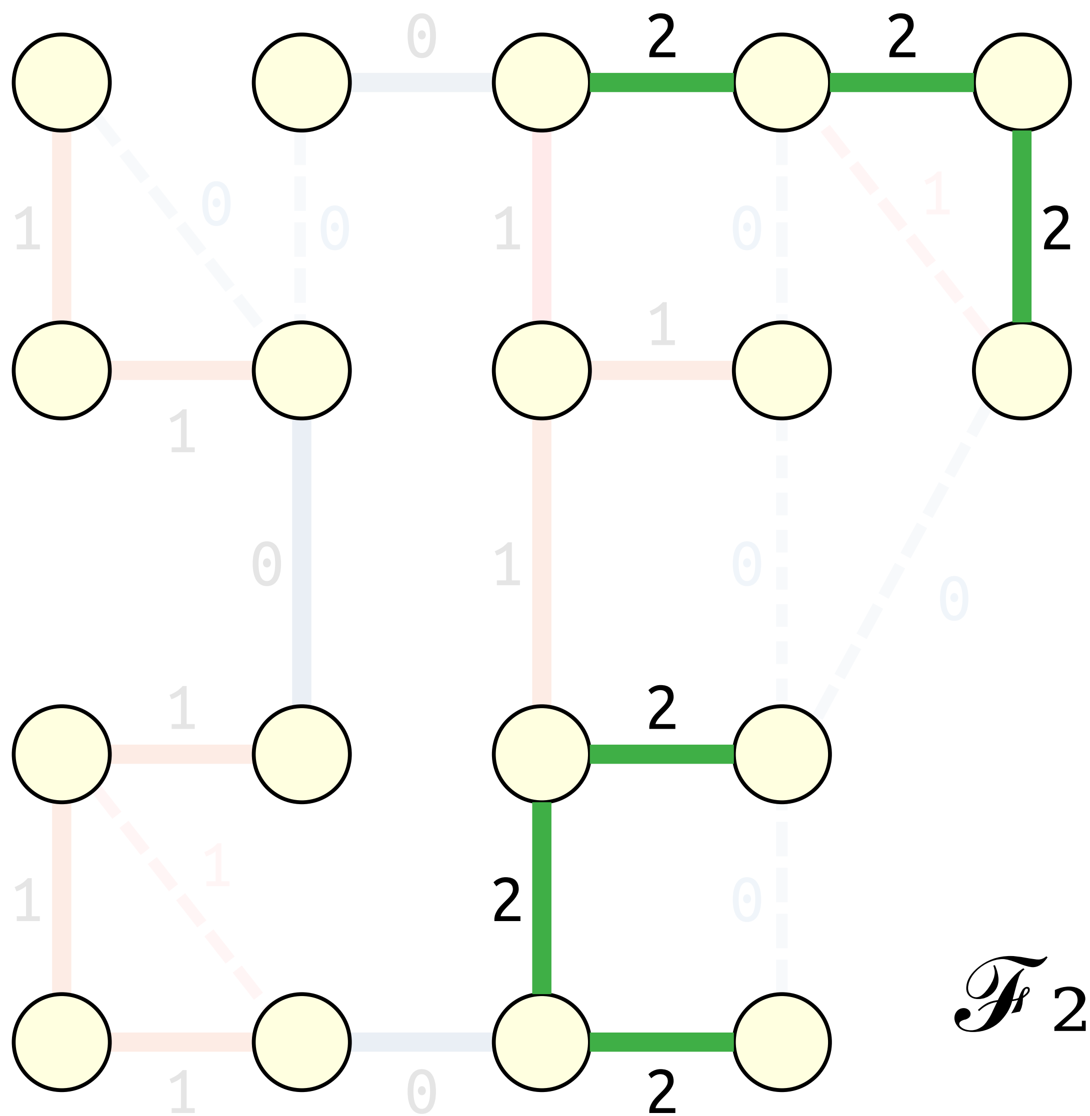
Layered Forests

- To make it faster to find auxiliary edges, each auxiliary edge of level l will be stored attached only to $\mathscr{F}_l$.
- After all, we only need to look for auxiliary edges of level l when we're focusing on trees made of edges of level l or above.



Layered Forests

- To make it faster to find auxiliary edges, each auxiliary edge of level l will be stored attached only to $\mathscr{F}_l$.
- After all, we only need to look for auxiliary edges of level l when we're focusing on trees made of edges of level l or above.



Invariant 1: $\mathscr{F}_0$ is a maximum spanning forest.
Invariant 2: $\mathscr{F}_0 \supseteq \mathscr{F}_1 \supseteq \mathscr{F}_2 \supseteq \ldots$

How fast is this?

To check ***are-connected***(x, y):

Return whether x and y are connected in $\mathscr{F}_0$.

To ***link***(x, y):

If ***are-connected***(x, y), add xy as an auxiliary edge to $\mathscr{F}_0$.

Otherwise add xy as a tree edge to $\mathscr{F}_0$.

To ***cut***(x, y), where xy is a tree edge of level l:

Delete xy from $\mathscr{F}_0$, $\mathscr{F}_1$, …, and $\mathscr{F}_l$.

For each level i from l down to 0:

Let T_x and T_y be the trees in $\mathscr{F}_i$ containing x and y.

Select one of T_x and T_y arbitrarily; WLOG assume it's T_x.

Increment the level of each tree edge of level i in T_x, adding each as tree edges to $\mathscr{F}_{i+1}$.

For each auxiliary edge uv in $\mathscr{F}_i$ touching T_x:

If uv connects T_x and T_y, add uv as a tree edge to $\mathscr{F}_r$ for $r \leq i$. Stop.

Else remove uv as an aux edge from $\mathscr{F}_i$, increment its level, and add it as an aux edge to $\mathscr{F}_{i+1}$.

Invariant 1: $\mathscr{F}_0$ is a maximum spanning forest.

Invariant 2: $\mathscr{F}_0 \supseteq \mathscr{F}_1 \supseteq \mathscr{F}_2 \supseteq \ldots$

How fast is this?

To check ***are-connected***(x, y):

Return whether x and y are connected in $\mathscr{F}_0$.

To ***link***(x, y):

If ***are-connected***(x, y), add xy as

Otherwise add xy as a tree edge t

To ***cut***(x, y), where xy is a tree edge

Delete xy from $\mathscr{F}_0$, $\mathscr{F}_1$, …, and $\mathscr{F}$

For each level i from l down to 0:

Let T_x and T_y be the trees in $\mathscr{F}_i$ containing x and y.

Select one of T_x and T_y arbitrarily; WLOG assume it's T_x.

Increment the level of each tree edge of level i in T_x, adding each as tree edges to $\mathscr{F}_{i+1}$.

For each auxiliary edge uv in $\mathscr{F}_i$ touching T_x:

If uv connects T_x and T_y, add uv as a tree edge to $\mathscr{F}_r$ for $r \leq i$. Stop.

Else remove uv as an aux edge from $\mathscr{F}_i$, increment its level, and add it as an aux edge to $\mathscr{F}_{i+1}$.

This is an ***are-connected*** query in $\mathscr{F}_0$. It takes amortized time $\mathbf{O(\log \mathit{n})}$.

How fast is this?

Invariant 1: $\mathscr{F}_0$ is a maximum spanning forest.

Invariant 2: $\mathscr{F}_0 \supseteq \mathscr{F}_1 \supseteq \mathscr{F}_2 \supseteq \dots$

To check ***are-connected***(x, y):

Return whether x and y are connected in $\mathscr{F}_0$.

To ***link***(x, y):

If ***are-connected***(x, y), add xy as an auxiliary edge to $\mathscr{F}_0$.

Otherwise add xy as a tree edge to $\mathscr{F}_0$.

To ***cut***(x, y), where xy is a tree edge of level l:

Delete xy from $\mathscr{F}_0$, $\mathscr{F}_1$, …

For each level i from l do

Let T_x and T_y be the tr

Select one of T_x and T

Increment the level o
adding each as tree e

For each auxiliary edge uv in $\mathscr{F}_i$ touching T_x:

If uv connects T_x and T_y, add uv as a tree edge to $\mathscr{F}_r$ for $r \le i$. Stop.

Else remove uv as an aux edge from $\mathscr{F}_i$, increment its level, and add it as an aux edge to $\mathscr{F}_{i+1}$.

This is an ***are-connected*** query in $\mathscr{F}_0$, plus either a ***link*** in $\mathscr{F}_0$ or an augmentation update to $\mathscr{F}_0$ to add xy as a tree edge.

Cost: amortized **O(log *n*)**.

Invariant 1: $\mathscr{F}_0$ is a maximum spanning forest.
Invariant 2: $\mathscr{F}_0 \supseteq \mathscr{F}_1 \supseteq \mathscr{F}_2 \supseteq \ldots$

How fast is this?

To check ***are-connected***(x, y):

Return whether x and y are connected in $\mathscr{F}_0$.

To ***link***(x, y):

If ***are-connected***(x, y), add xy as an auxiliary edge to $\mathscr{F}_0$.

Otherwise add xy as a tree edge to $\mathscr{F}_0$.

To ***cut***(x, y), where xy is a tree edge of level l:

Delete xy from $\mathscr{F}_0$, $\mathscr{F}_1$, …, and $\mathscr{F}_l$.

For each level i from l down to 0:

Let T_x and T_y be the trees in $\mathscr{F}_i$

Select one of T_x and T_y arbitrari

Increment the level of each tree
adding each as tree edges to $\mathscr{F}$

For each auxiliary edge uv in $\mathscr{F}$

If uv connects T_x and T_y, add

Else remove uv as an aux edge from $\mathscr{F}_i$, increment its level, and add it as an aux edge to $\mathscr{F}_{i+1}$.

This is a ***cut*** operation across all the levels.

Suppose there are L levels in the forest. Then this takes (amortized) time $\mathbf{O}(L \log n)$.

Invariant 1: $\mathscr{F}_0$ is a maximum spanning forest.

Invariant 2: $\mathscr{F}_0 \supseteq \mathscr{F}_1 \supseteq \mathscr{F}_2 \supseteq \ldots$

How fast is this?

To check ***are-connected***(x, y):

Return whether x and y are connected in $\mathscr{F}_0$.

To ***link***(x, y):

If ***are-connected***(x, y), add xy as an auxiliary edge to $\mathscr{F}_0$.

Otherwise add xy as a tree edge to $\mathscr{F}_0$.

To ***cut***(x, y), where xy is a tree edge of level l:

Delete xy from $\mathscr{F}_0$, $\mathscr{F}_1$, …, and $\mathscr{F}_l$.

For each level i from l down to 0:

Let T_x and T_y be the trees in $\mathscr{F}_i$ containing

Select one of T_x and T_y arbitrarily; WLOG

Increment the level of each tree edge of l

adding each as tree edges to $\mathscr{F}_{i+1}$.

For each auxiliary edge uv in $\mathscr{F}_i$ touching T_x:

If uv connects T_x and T_y, add uv as a tree edge to $\mathscr{F}_r$ for $r \leq i$. Stop.

Else remove uv as an aux edge from $\mathscr{F}_i$, increment its level, and add it as an aux edge to $\mathscr{F}_{i+1}$.

This is a ***link*** operation across all the levels. It takes (amortized) time **O(*L* log *n*)**.

Invariant 1: $\mathscr{F}_0$ is a maximum spanning forest.
Invariant 2: $\mathscr{F}_0 \supseteq \mathscr{F}_1 \supseteq \mathscr{F}_2 \supseteq \dots$

How fast is this?

To check ***are-connected***(x, y):

Return whether x and y are connected in $\mathscr{F}_0$.

To ***link***(x, y):

If ***are-conne***

Otherwise ad

To ***cut***(x, y), whe

Delete xy fro

For each leve

Let T_x an

These steps might take a long time if there are a lot of edges to move.

However, ***each individual edge's level can only be incremented L times***.

Idea: Amortize away this cost by "blaming" it on the cost of adding the edge.

Select one of T_x and T_y arbitrarily; WLOG assume it's T_x.

Increment the level of each tree edge of level i in T_x, adding each as tree edges to $\mathscr{F}_{i+1}$.

For each auxiliary edge uv in $\mathscr{F}_i$ touching T_x:

If uv connects T_x and T_y, add uv as a tree edge to $\mathscr{F}_r$ for $r \leq i$. Stop.

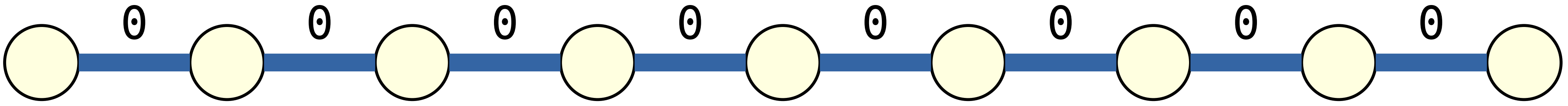
Else remove uv as an aux edge from $\mathscr{F}_i$, increment its level, and add it as an aux edge to $\mathscr{F}_{i+1}$.

The Amortized Analysis

- Suppose the forest has a maximum of L levels.
- Raising the level of an individual edge takes time $O(\log n)$, so across all ***cut*** operations, we spend at most $O(L \log n)$ work on any one edge.
- With the right choice of Φ, we can get these amortized costs on the operations:
  - ***link***: $\mathbf{O(L \log n)}$, paying the full cost of raising the edge's level up front.
  - ***cut***: $\mathbf{O(L \log n)}$, accounting for the costs of all operations not attributable to edge raising.
- ***Question:*** How big can L get?

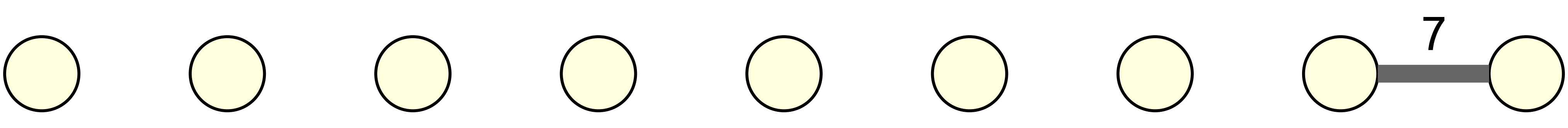
Bounding the Max Level

- ***Problem:*** Without further restrictions, the maximum level in a forest of n nodes is $L = n - 2$.
- How can that happen? How can we prevent it?



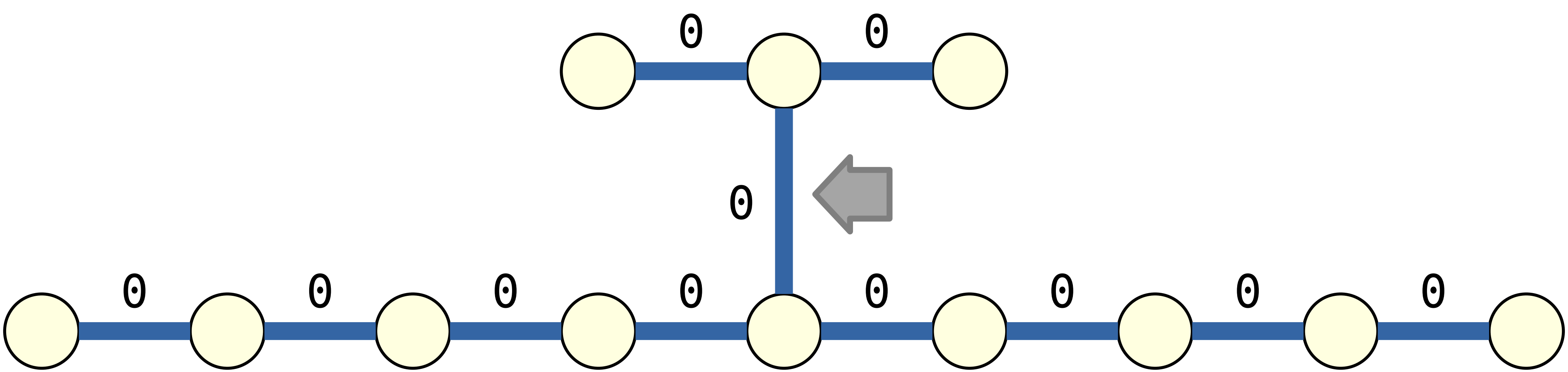
Bounding the Max Level

- ***Problem:*** Without further restrictions, the maximum level in a forest of n nodes is $L = n - 2$.
- How can that happen? How can we prevent it?



Bounding the Max Level

- Suppose we cut the indicated edge.



Blame It On The Little Guy

- ***Claim:*** If we always pick the smaller tree when boosting levels, the maximum level will be $L = O(\log n)$.
- ***Why?***
  - The maximum tree size in $\mathscr{F}_0$ is n nodes.
  - The maximum tree size in $\mathscr{F}_1$ is $n/2$ nodes.
  - The maximum tree size in $\mathscr{F}_2$ is $n/4$ nodes.
  - …
  - The maximum tree size in $\mathscr{F}_{\lg n}$ is 1 node.
- ***General Technique:*** "Blame it on the little guy" by repeatedly updating the smaller of two quantities, or accounting for work done on the smaller of two quantities, etc. This often converts linear bounds to logarithmic ones.

Invariant 1: $\mathscr{F}_0$ is a maximum spanning forest.

Invariant 2: $\mathscr{F}_0 \supseteq \mathscr{F}_1 \supseteq \mathscr{F}_2 \supseteq \ldots$

Invariant 3: Each tree in $\mathscr{F}_i$ has at most $^n/_{2^i}$ nodes.

To check ***are-connected***(x, y):

Return whether x and y are connected in $\mathscr{F}_0$.

To ***link***(x, y):

If ***are-connected***(x, y), add xy as an auxiliary edge to $\mathscr{F}_0$.

Otherwise add xy as a tree edge to $\mathscr{F}_0$.

To ***cut***(x, y), where xy is a tree edge of level l:

Delete xy from $\mathscr{F}_0$, $\mathscr{F}_1$, …, and $\mathscr{F}_l$.

For each level i from l down to 0:

Let T_x and T_y be the trees in $\mathscr{F}_i$ containing x and y.

Select the smaller of T_x and T_y; WLOG assume it's T_x.

Increment the level of each tree edge of level i in T_x, adding each as tree edges to $\mathscr{F}_{i+1}$.

For each auxiliary edge uv in $\mathscr{F}_i$ touching T_x:

If uv connects T_x and T_y, add uv as a tree edge to $\mathscr{F}_r$ for $r \leq i$. Stop.

Else remove uv as an aux edge from $\mathscr{F}_i$, increment its level, and add it as an aux edge to $\mathscr{F}_{i+1}$.

This is our final structure!

The Final Scorecard

- This final data structure is called the ***Holm forest*** or ***layered forest***. It maintains an MSF using our earlier approach while ensuring that $L = O(\log n)$.
- It supports
  - ***link***(u, v) in amortized time $\mathbf{O(\log^2 n)}$,
  - ***cut***(u, v) in amortized time $\mathbf{O(\log^2 n)}$, and
  - ***are-connected***(u, v) in amortized time $\mathbf{O(\log n)}$.
- These bounds are *substantially* better than the naive approach – isn't that amazing?

The Final-er Scorecard

- There is one further improvement we can make to the structure, and it's clever.
  - All connectivity queries are done in $\mathscr{F}_0$.
  - Instead of representing the Euler tour tree for $\mathscr{F}_0$ using splay trees, represent them with B-trees of order log n.
  - This makes ***are-connected*** take worst-case time $O\left(\frac{\log n}{\log \log n}\right)$.
  - Updating $\mathscr{F}_0$ now takes time $O\left(\frac{\log^2 n}{\log \log n}\right)$, but we don't notice this because the amortized cost of ***link*** and ***cut*** is still $O(\log^2 n)$.
- This structure then supports
  - ***link***(u, v) in amortized time $\mathbf{O(\log^2 n)}$,
  - ***cut***(u, v) in amortized time $\mathbf{O(\log^2 n)}$, and
  - ***are-connected***(u, v) in worst-case time $\mathbf{O\left(\frac{\log n}{\log \log n}\right)}$.

Going Forward

- Here's some other amazing work folks have done in this space:
  - In 2000, Thorup introduced randomization into the Holm forest to get expected amortized $O(\log n (\log \log n)^3)$ costs for ***link*** and ***cut***, with $O(\log n / \log \log \log n)$ for ***are-connected*** queries.
  - In 2013, Kapron et al used randomization without amortization to get $O(\log^5 n)$ worst-case costs per ***link*** or ***cut*** and $O(\log n / \log \log n)$ ***are-connected*** query times, with a high chance of success.
- Every data structure for dynamic connectivity must have ***link*** and ***cut*** run in $\Omega(\log n)$ or ***are-connected*** run in time $\Omega(\log n / \log \log n)$. Is there still a ways to go, or are these lower bounds too loose? ***We don't know!***

More Dynamic Problems

- Many other dynamic graph problems exist:
  - Maintaining an MST; can do in $O(\log^4 n)$ time per insertion or deletion.
  - Maintaining single-source or all-pairs shortest paths.
  - Maintaining reachability in a *directed* graph.
- All of these problems were solved in the static case 50+ years ago.
- We have somewhat decent solutions to the dynamic cases.
- ***This is an active area of research!***

Next Time

- ***Word-Level Parallelism***
  - Harnessing a degree of parallelism we've overlooked thus far.
- ***Sardine Trees***
  - Outperforming BSTs for small integers.
- ***MSB in O(1)***
  - A seemingly impossible bitwise operation.