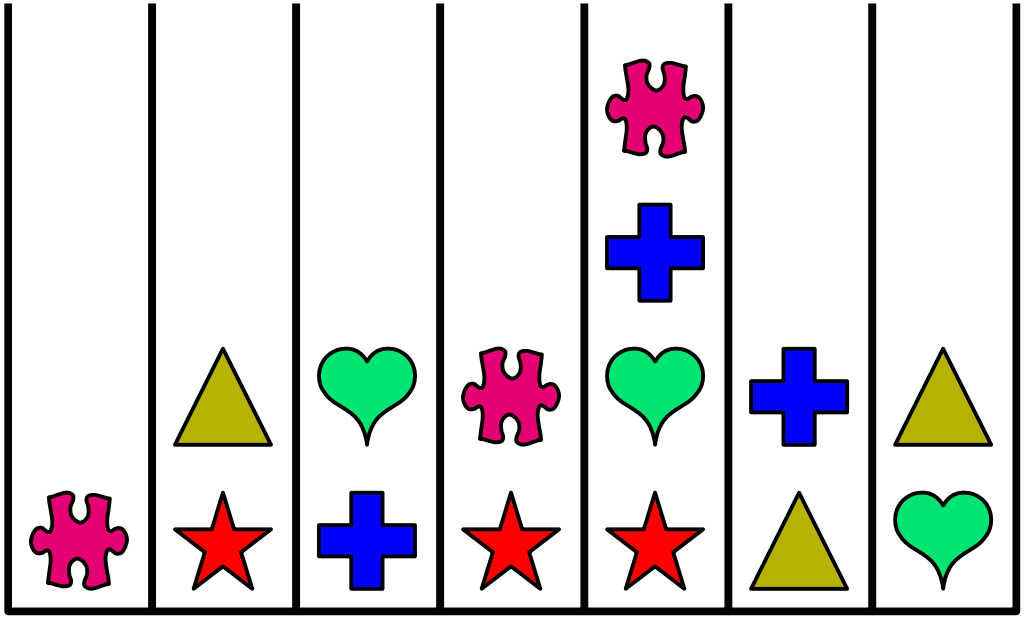
Peeling Algorithms

Part Two

Recap from Last Time

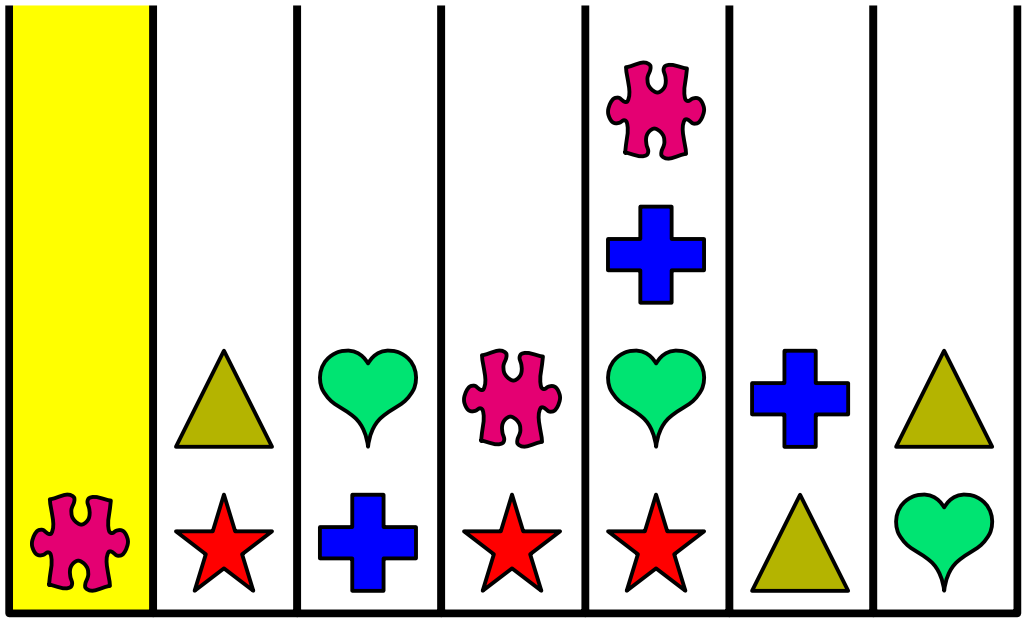
The Peeling Algorithm

- Assign a collection of items to a collection of slots.
- Repeat the following:
  - Find a slot with one item in it.
  - Remove that item from all slots it contains.
- This is called ***peeling***.



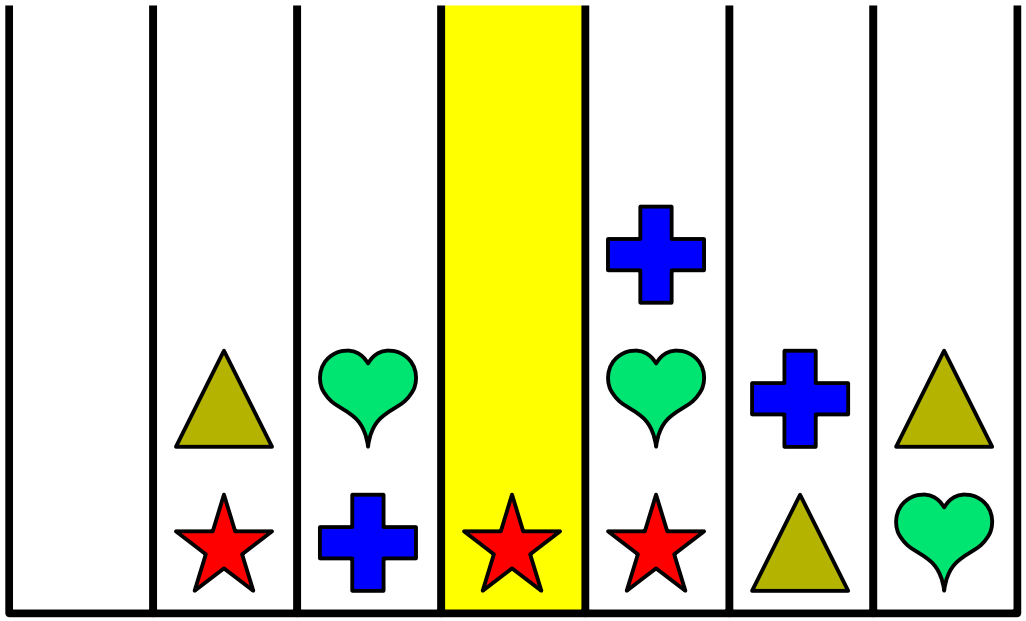
The Peeling Algorithm

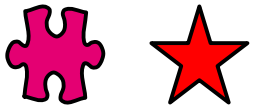
- Assign a collection of items to a collection of slots.
- Repeat the following:
  - Find a slot with one item in it.
  - Remove that item from all slots it contains.
- This is called ***peeling***.



The Peeling Algorithm

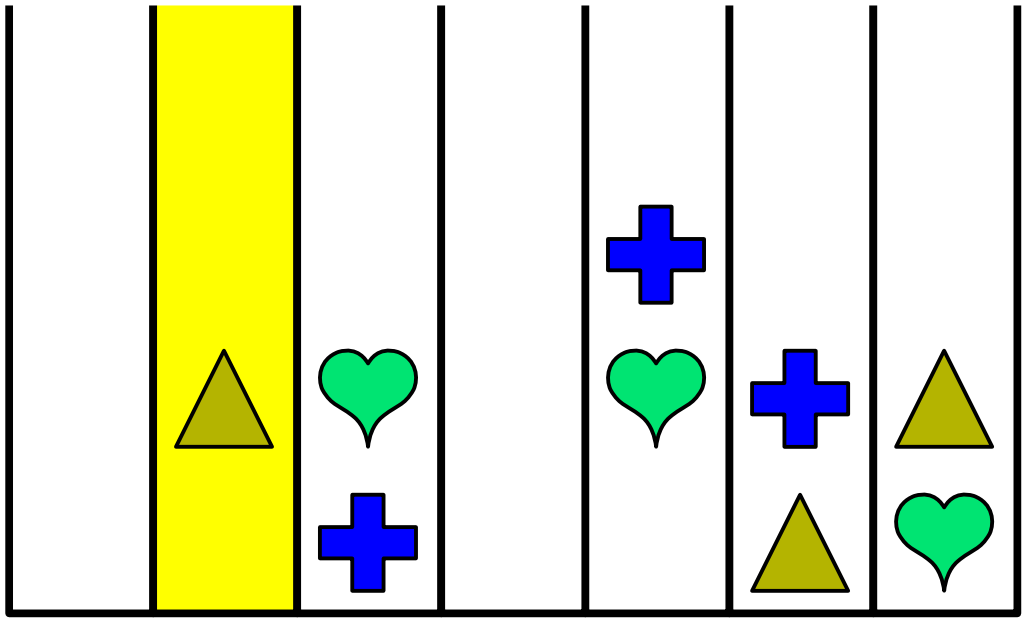
- Assign a collection of items to a collection of slots.
- Repeat the following:
  - Find a slot with one item in it.
  - Remove that item from all slots it contains.
- This is called ***peeling***.





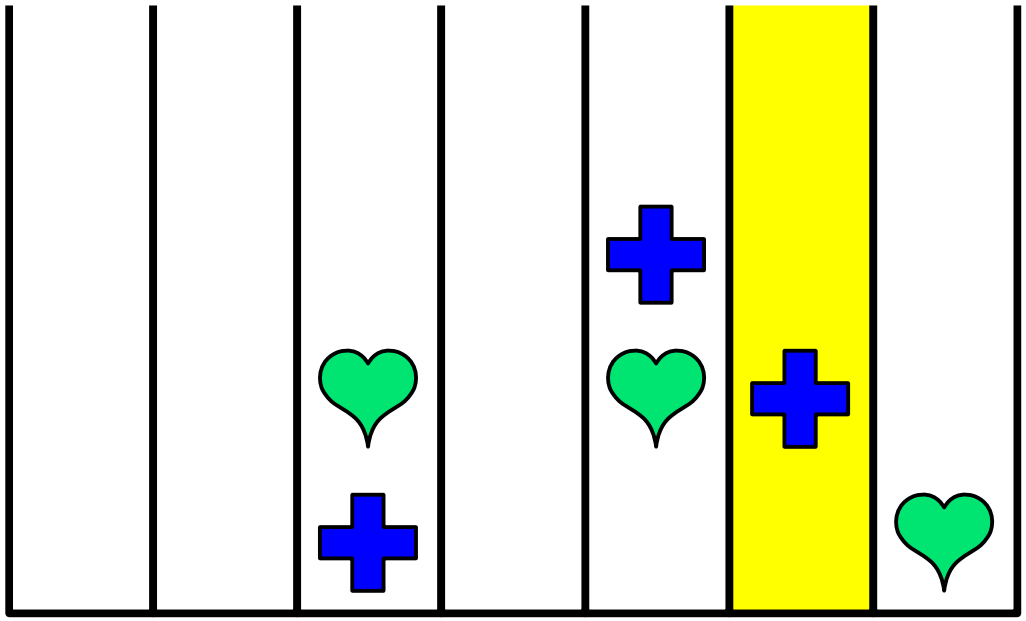
The Peeling Algorithm

- Assign a collection of items to a collection of slots.
- Repeat the following:
  - Find a slot with one item in it.
  - Remove that item from all slots it contains.
- This is called ***peeling***.



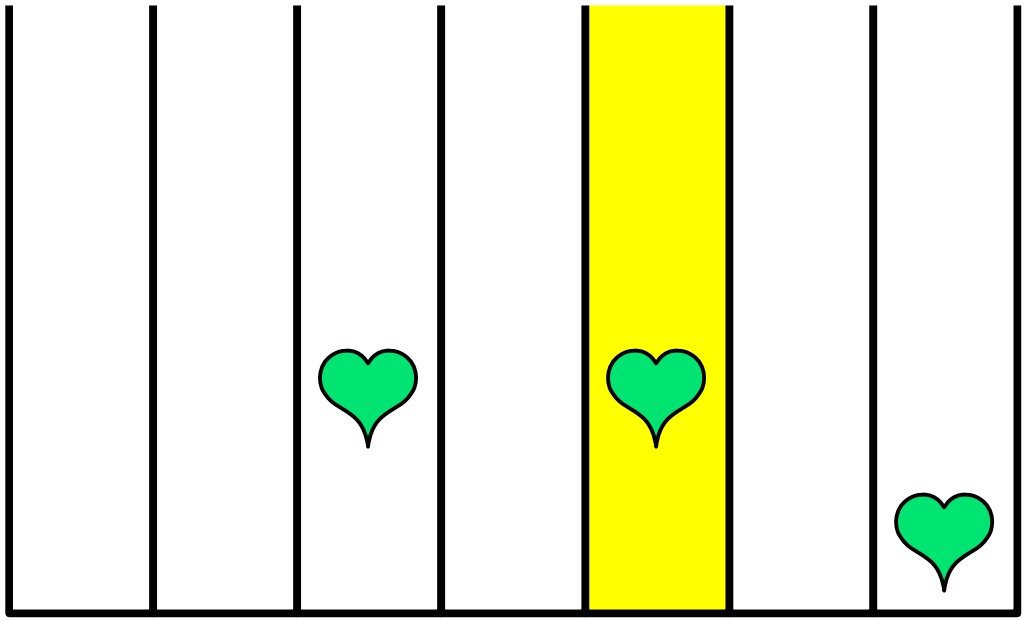
The Peeling Algorithm

- Assign a collection of items to a collection of slots.
- Repeat the following:
  - Find a slot with one item in it.
  - Remove that item from all slots it contains.
- This is called ***peeling***.



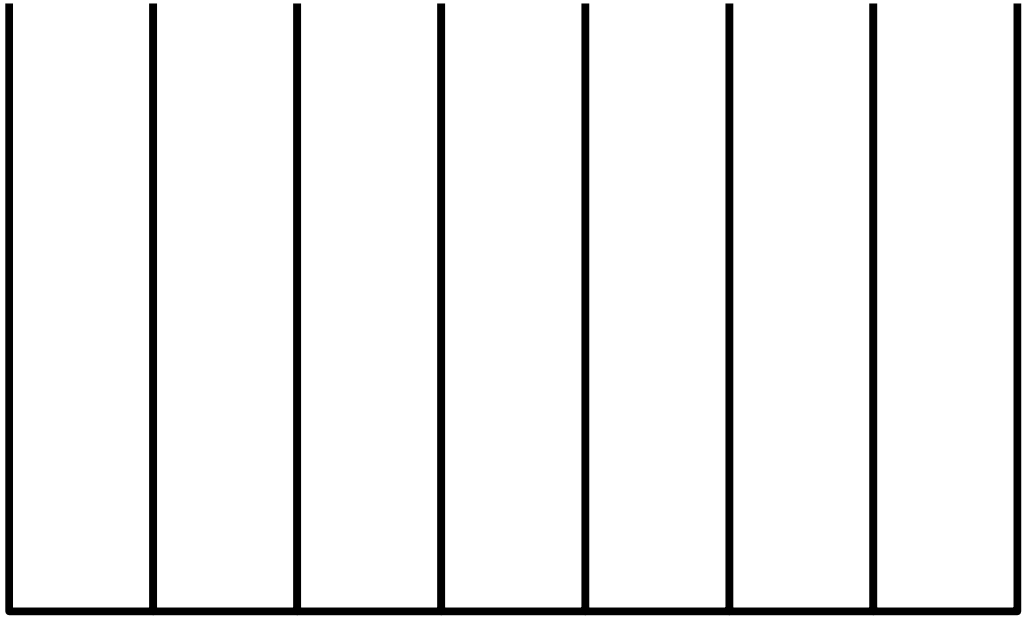
The Peeling Algorithm

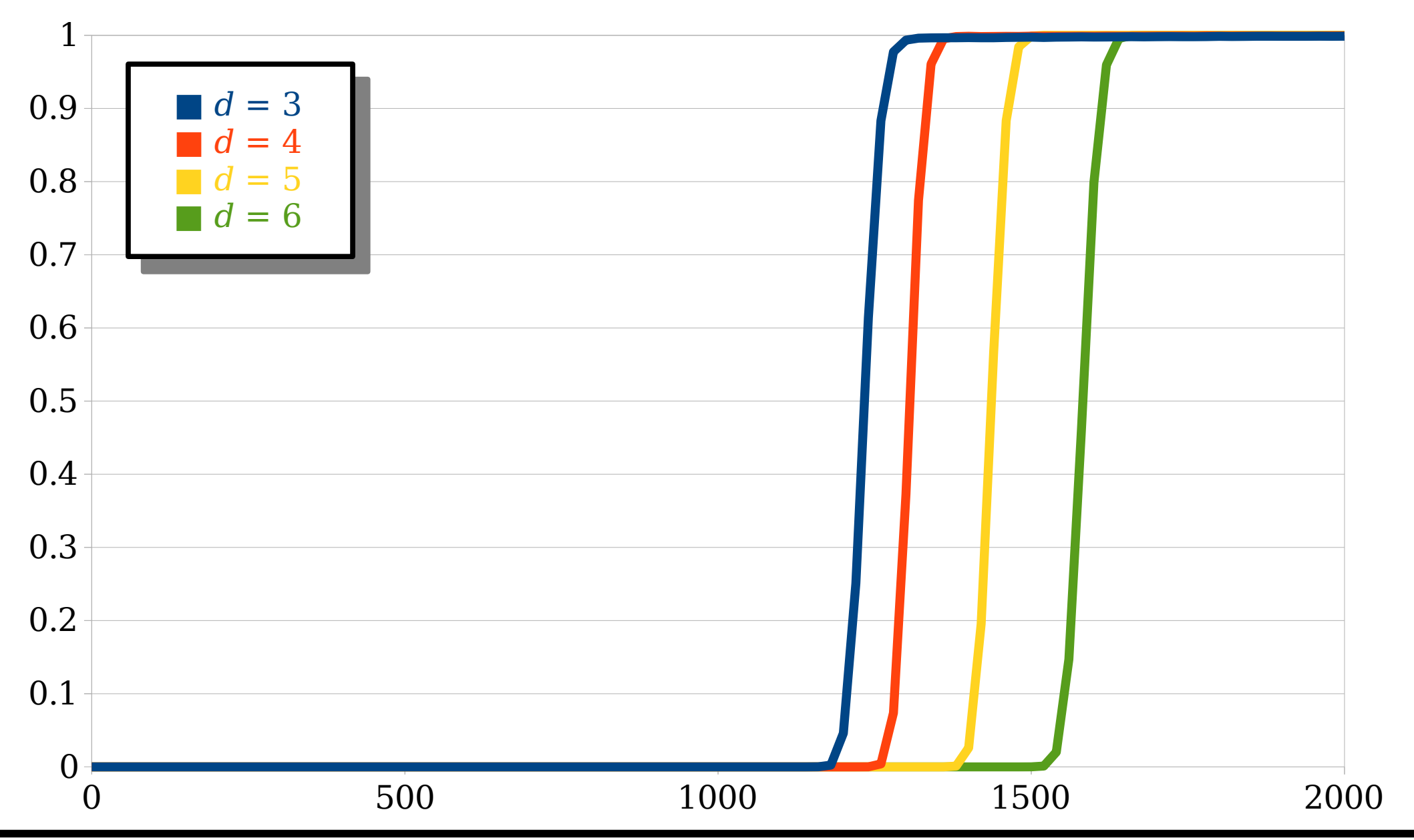
- Assign a collection of items to a collection of slots.
- Repeat the following:
  - Find a slot with one item in it.
  - Remove that item from all slots it contains.
- This is called ***peeling***.



The Peeling Algorithm

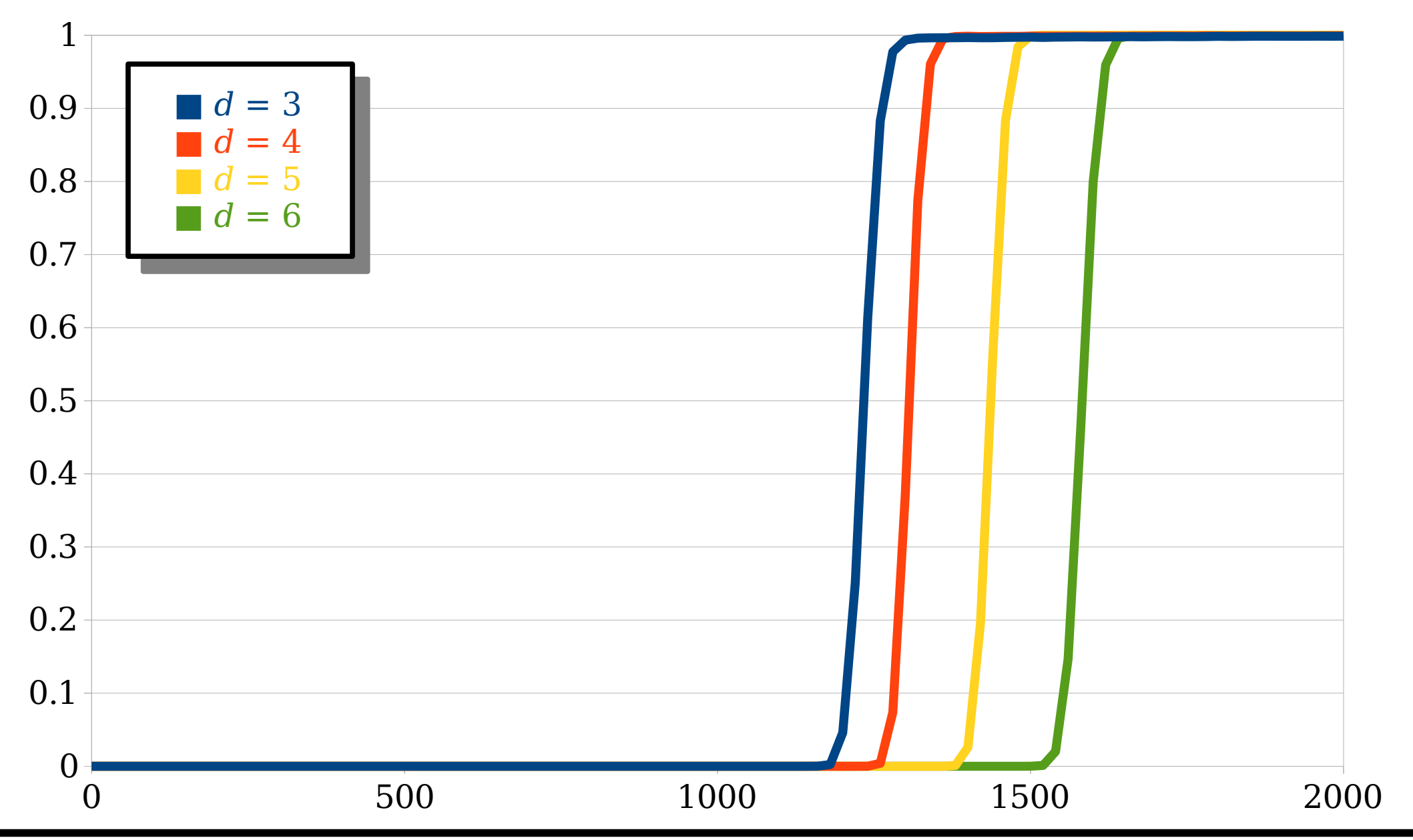
- Assign a collection of items to a collection of slots.
- Repeat the following:
  - Find a slot with one item in it.
  - Remove that item from all slots it contains.
- This is called ***peeling***.





What is the probability that 1000 items can be peeled from a collection of m buckets with d hash functions?

New Stuff!



What's going on here?

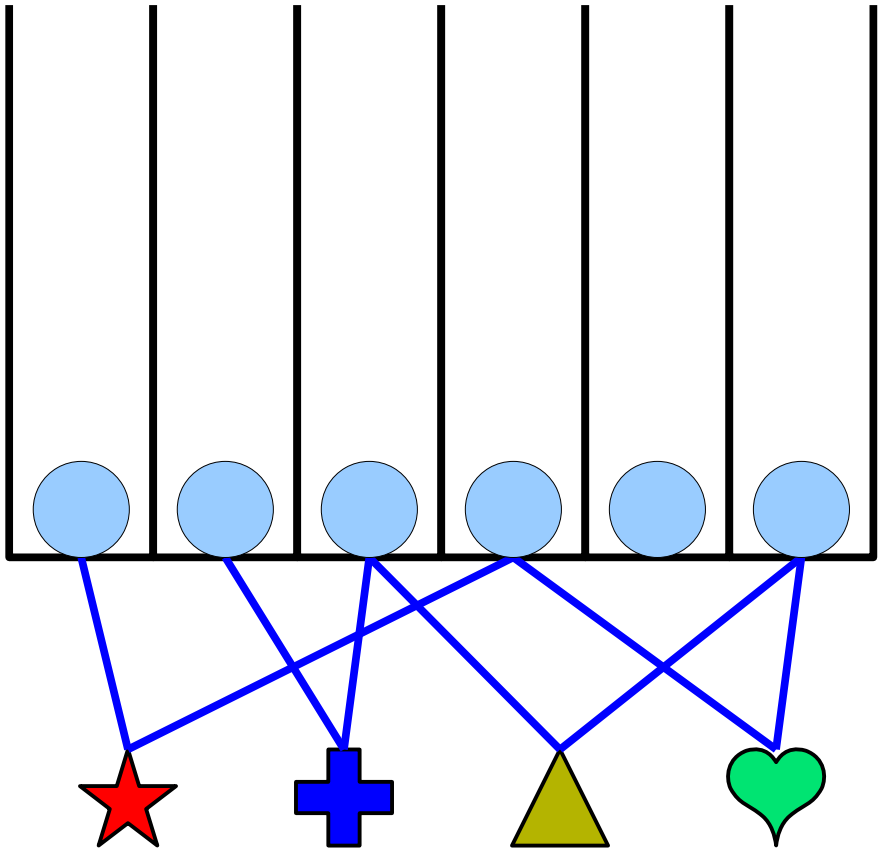
The Gameplan

- ***Part 1:*** Model peeling with random graphs.
- ***Part 2:*** Zoom into the graph to find trees.
- ***Part 3:*** Model trees with Poisson variables.
- ***Part 4:*** View peeling as a process on trees.
- ***Part 5:*** Get a recurrence from this process.
- ***Part 6:*** Solve the recurrence.

Part 1: Model peeling as a process on graphs.

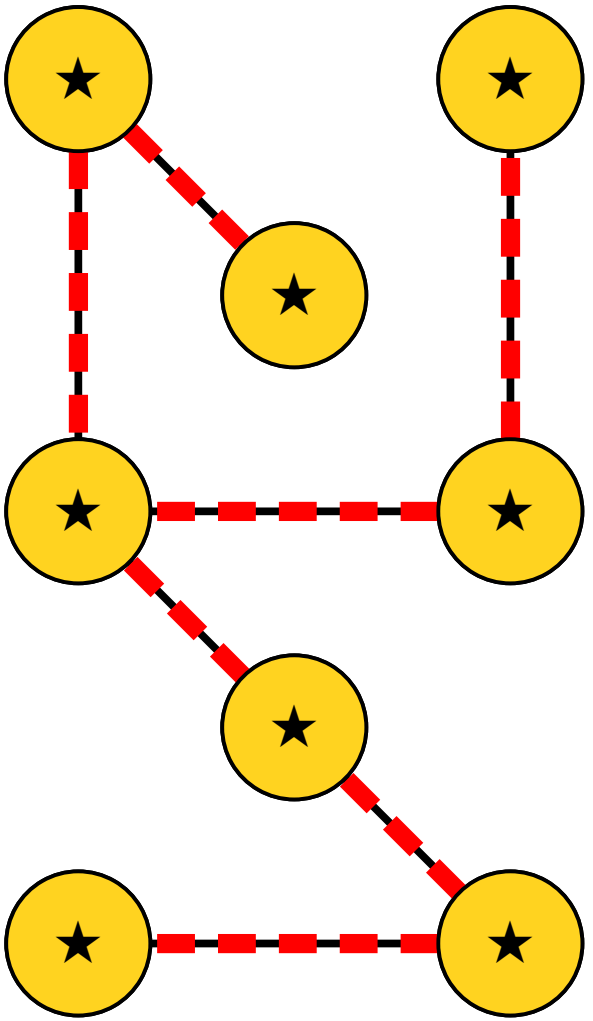
Peeling, $d = 2$

- Imagine we have $d = 2$ hash functions hashing items to slots.
- We can model peeling as a process on a graph.
- Each slot is a node.
- Each edge is an item linking $h_1(x)$ and $h_2(x)$.
- *(Where have you seen this before?)*



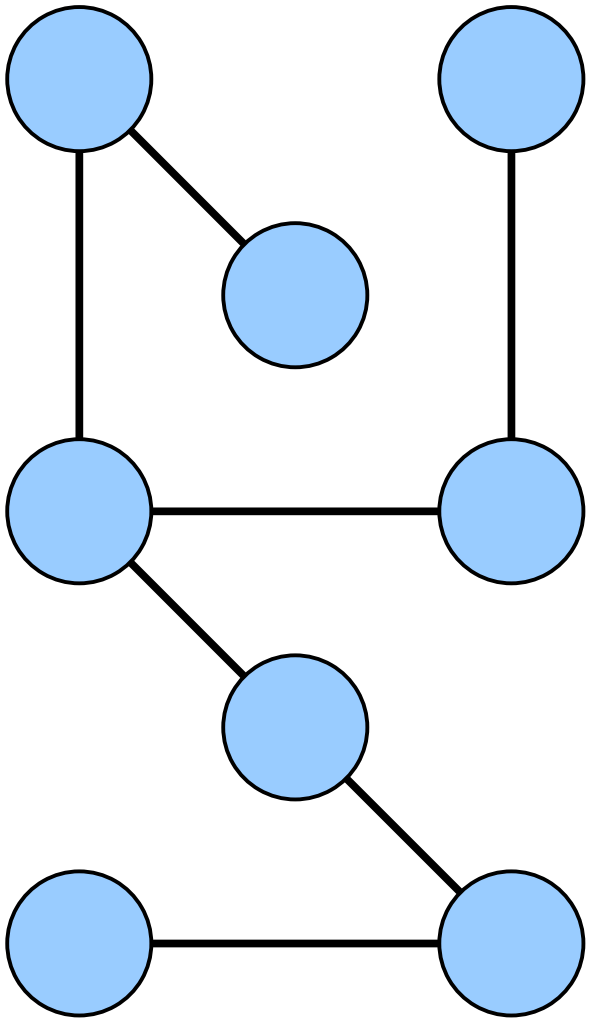
Peeling, $d = 2$

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Graphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident edges.
- *(When will this process succeed?)*



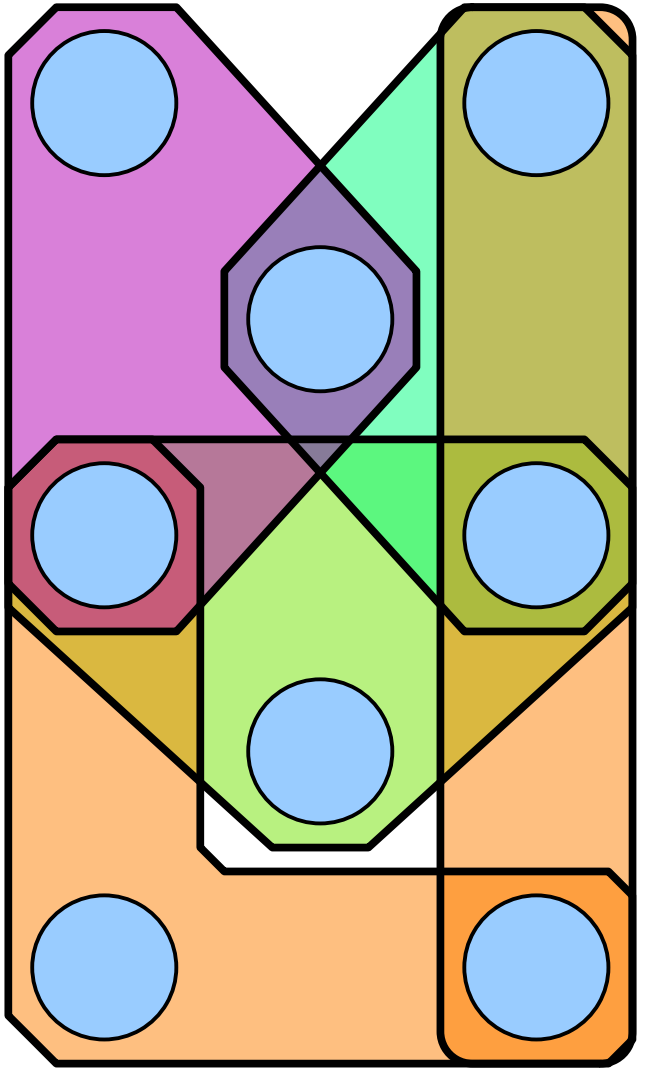
Peeling, $d > 2$

- The analogy to graphs works well when $d = 2$.
  - Each item links two slots.
- What do we do when $d > 2$?
- Now our edges need to link three or more slots. 🤔



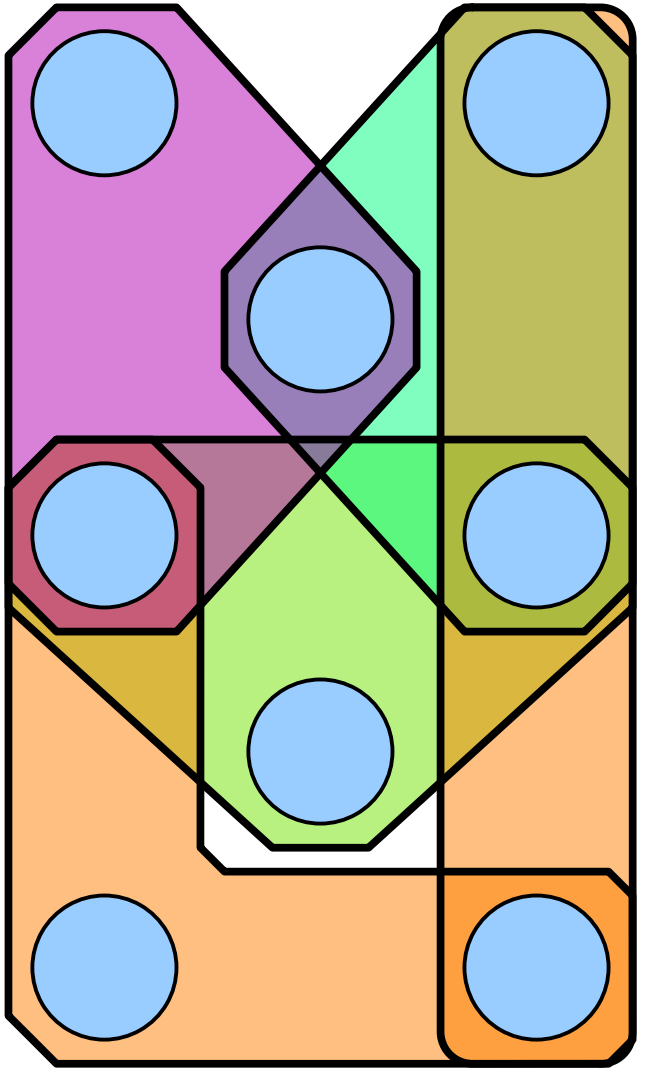
Introducing Hypergraphs

- A ***hypergraph*** is a generalization of a graph.
- As before, we have nodes.
- Instead of edges, we have ***hyperedges*** that link arbitrary number of nodes together.



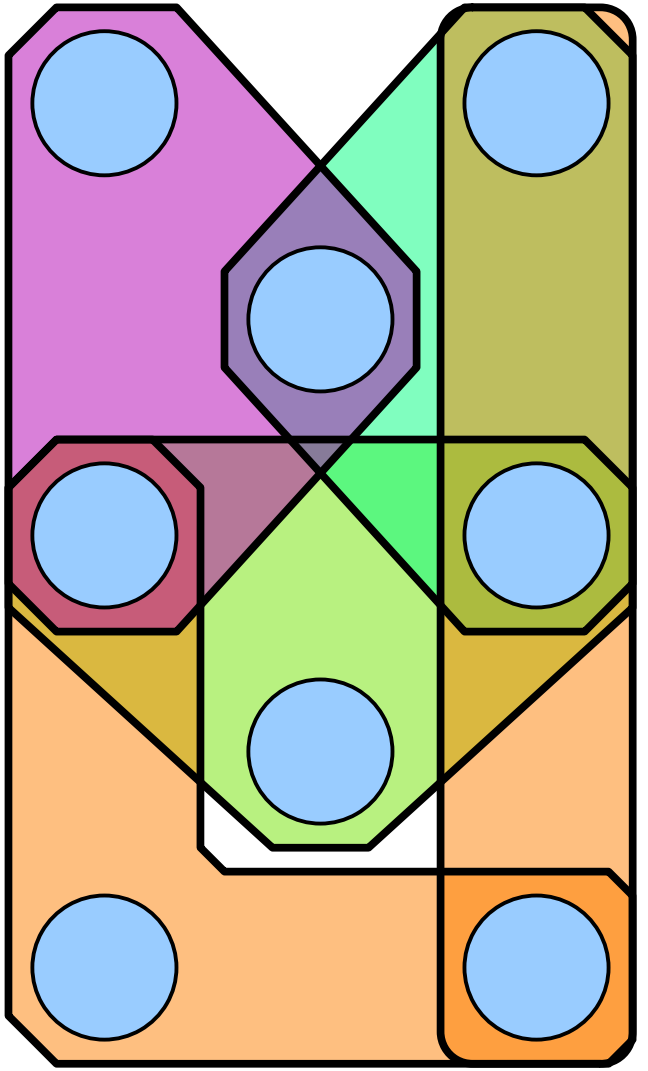
Hypergraph Peeling

- Imagine we have d hash functions hashing items to slots.
- We can model peeling as a process on a hypergraph.
- Each slot is a node.
- Each hyperedge is an item linking $h_1(x)$, …, $h_d(x)$.
- A hypergraph where each edge touches d nodes is ***d-uniform***.



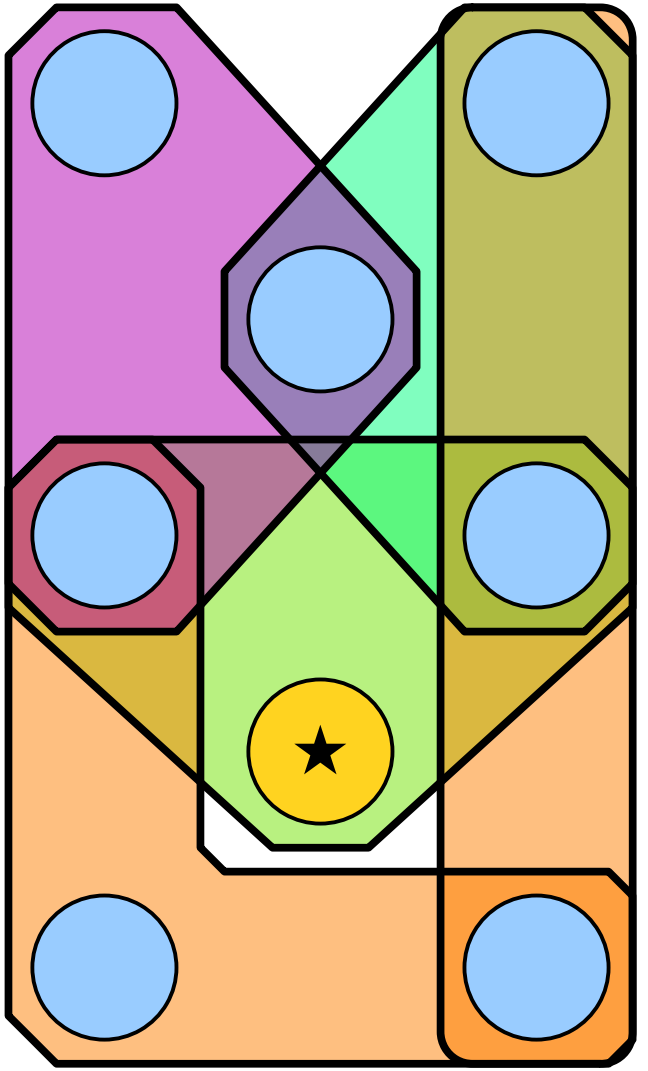
Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!



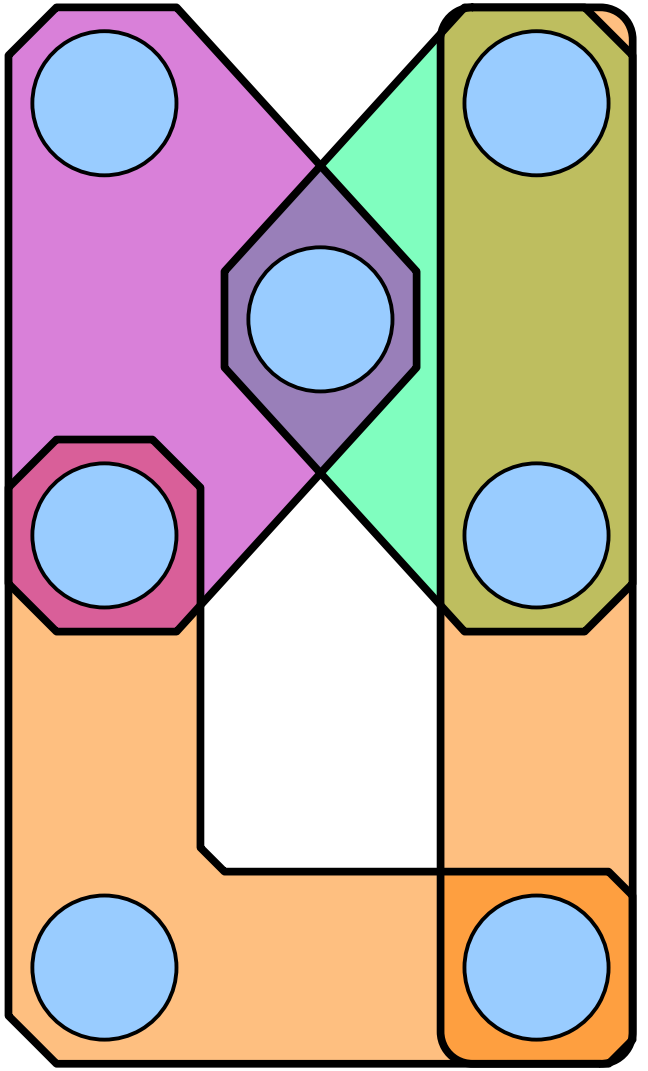
Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!



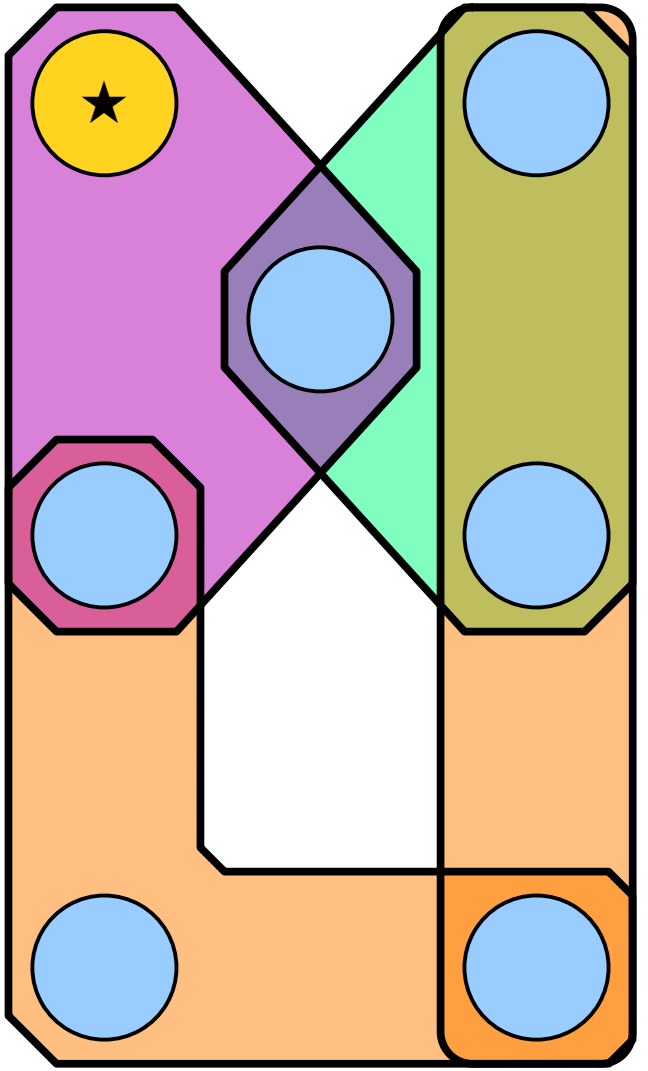
Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!



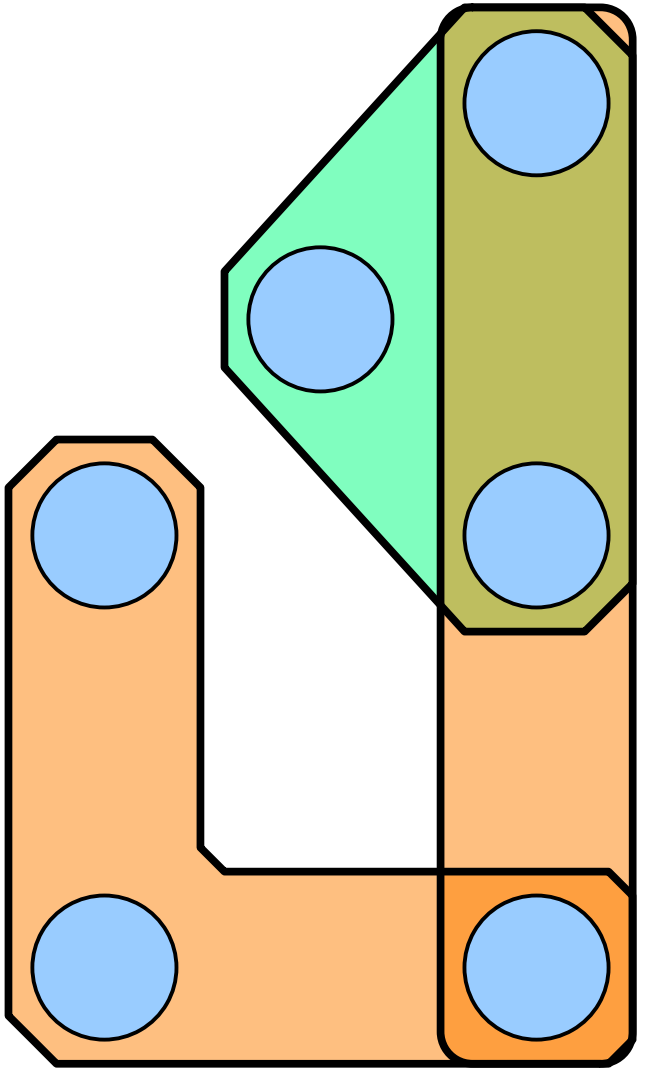
Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!



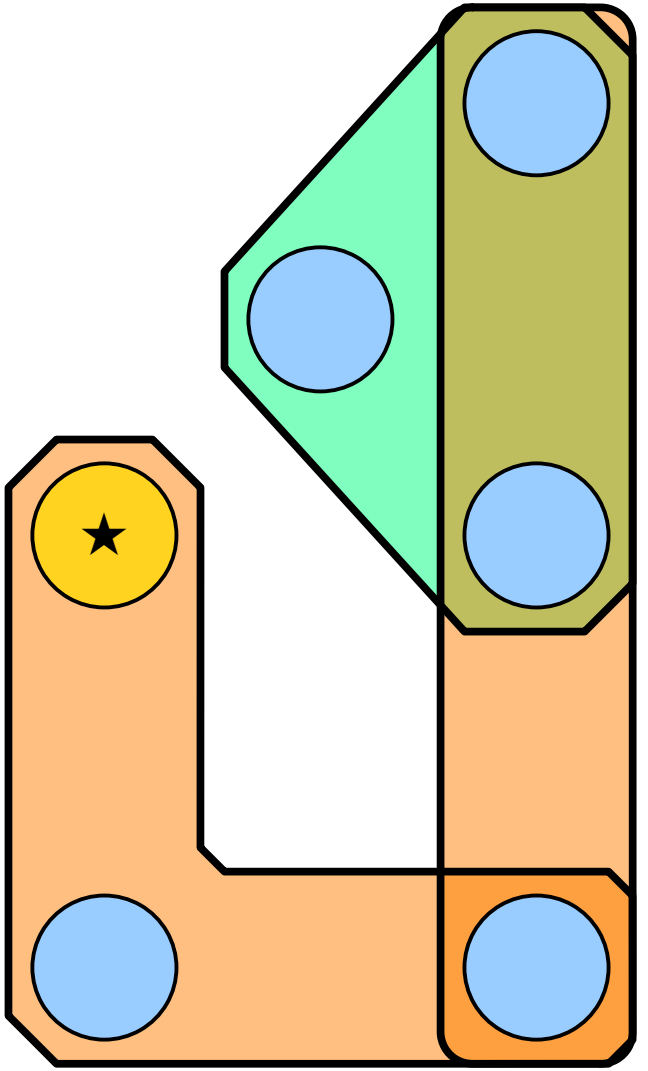
Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!



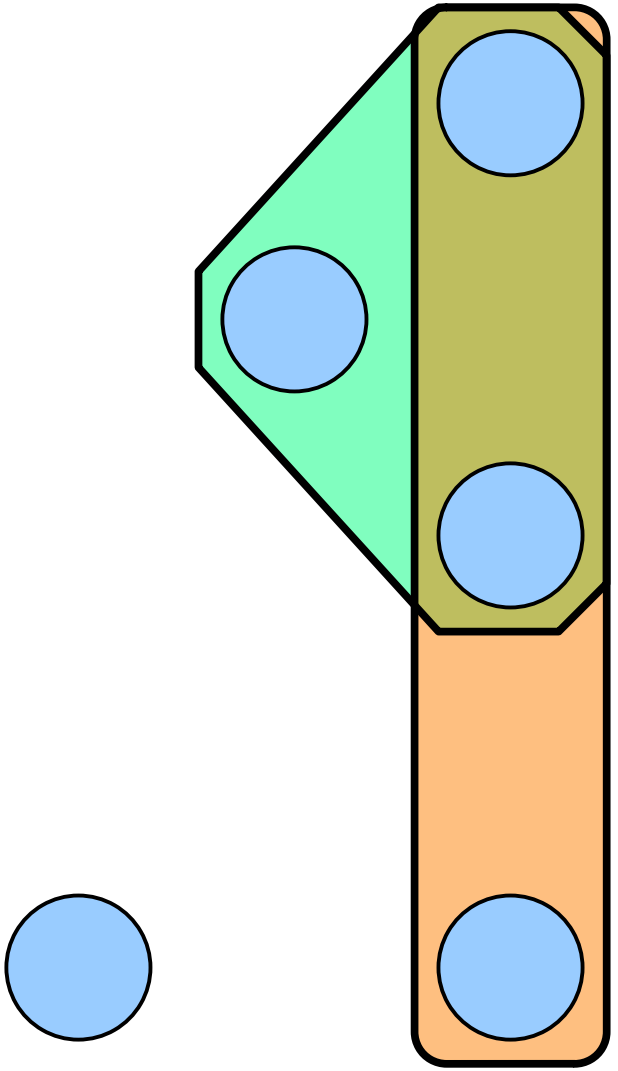
Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!



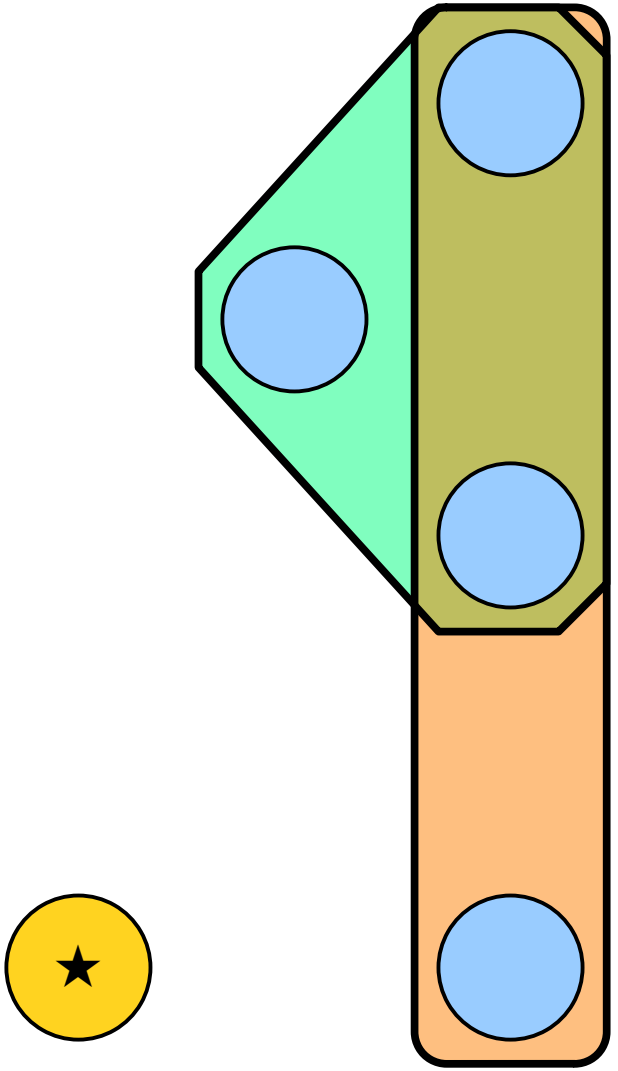
Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!



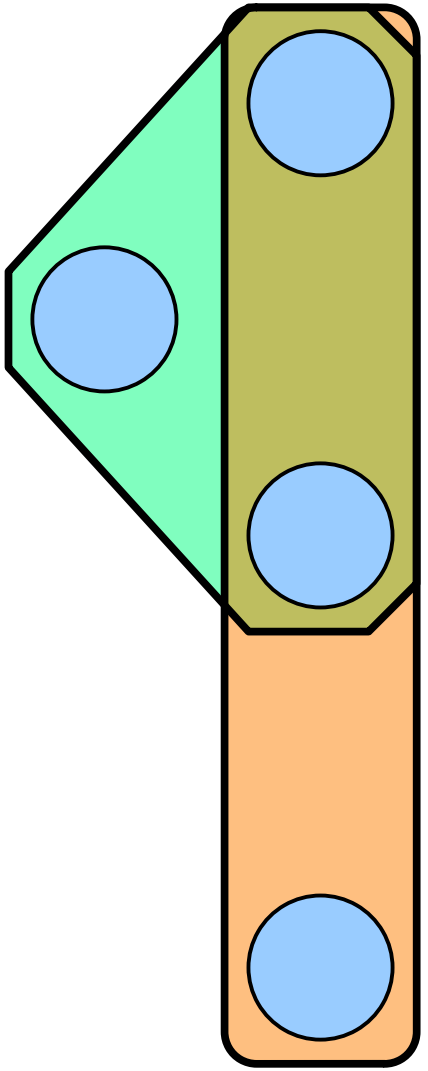
Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!



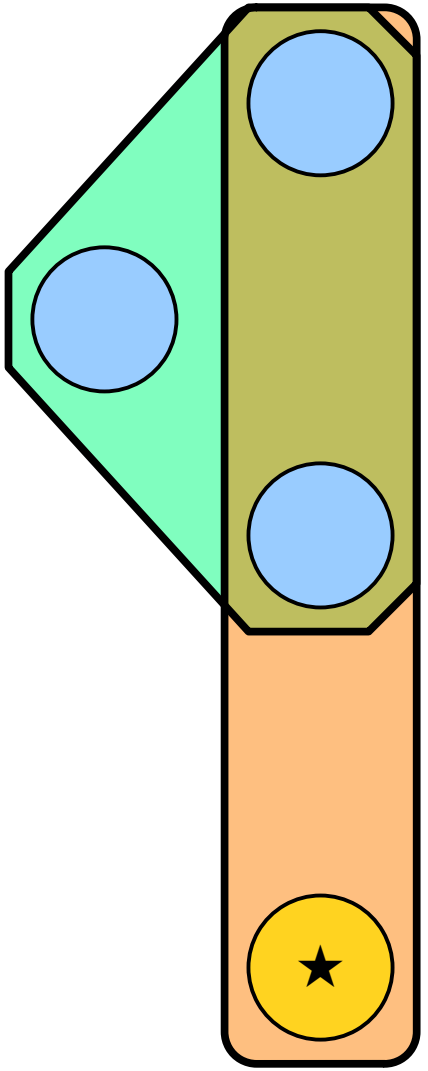
Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!



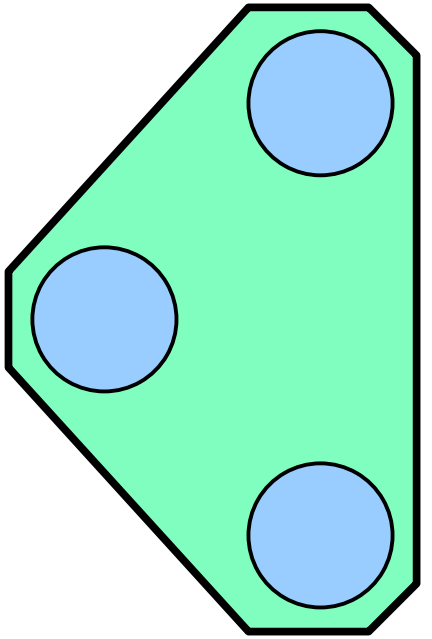
Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!



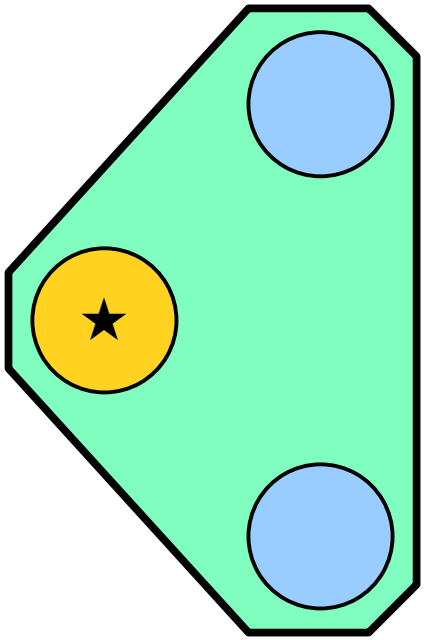
Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!



Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!

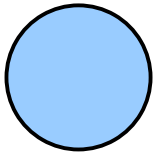


Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!

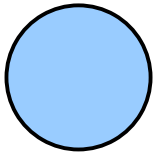
Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!



Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!

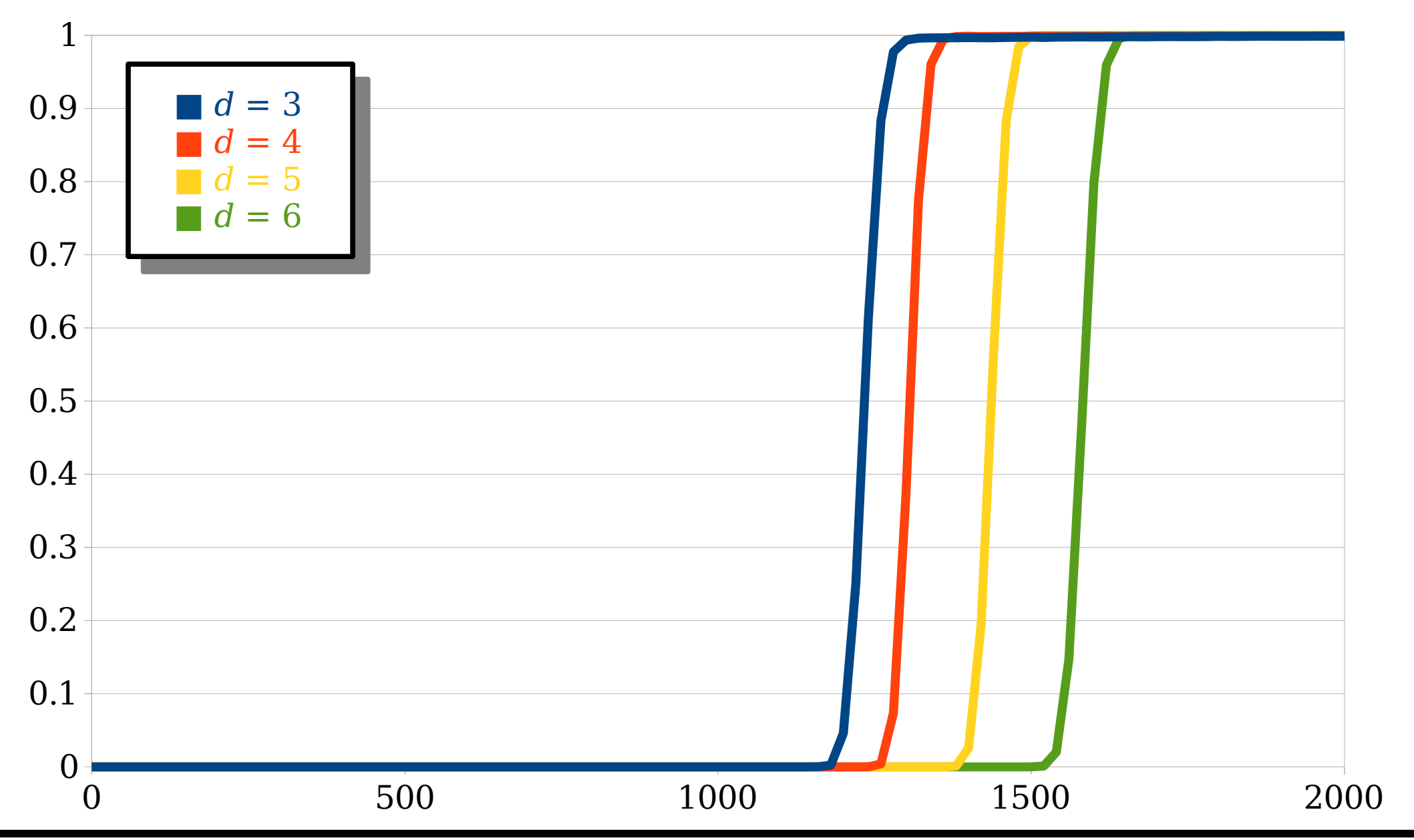


Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!

Hypergraph Peeling

- Our peeling algorithm finds and removes slots with at most one item in them.
- In Hypergraphland, we do the following:
  - Find a node of degree at most 1.
  - Delete it and all incident hyperedges.
- The hypergraph does *not* have to be a tree!

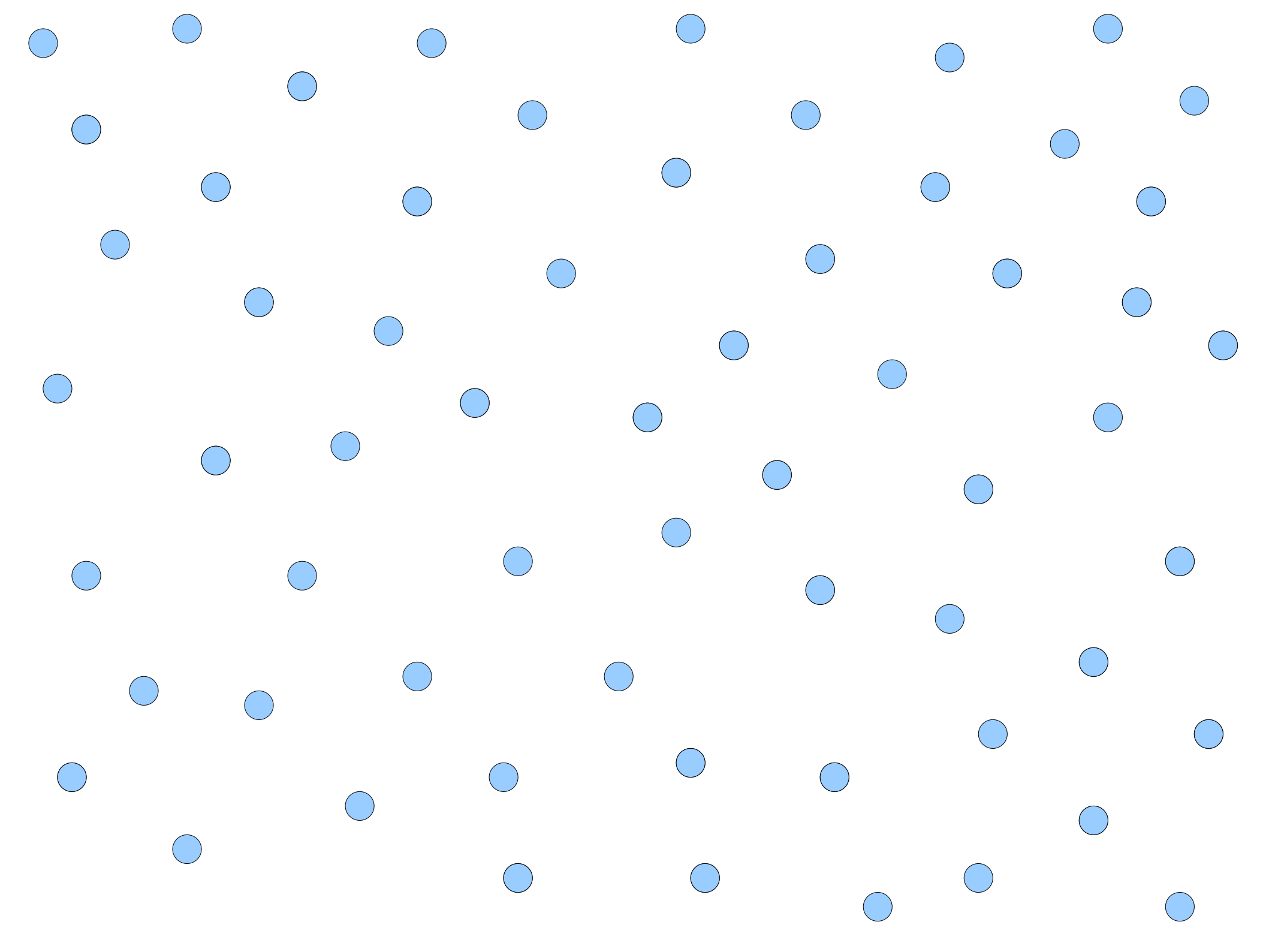


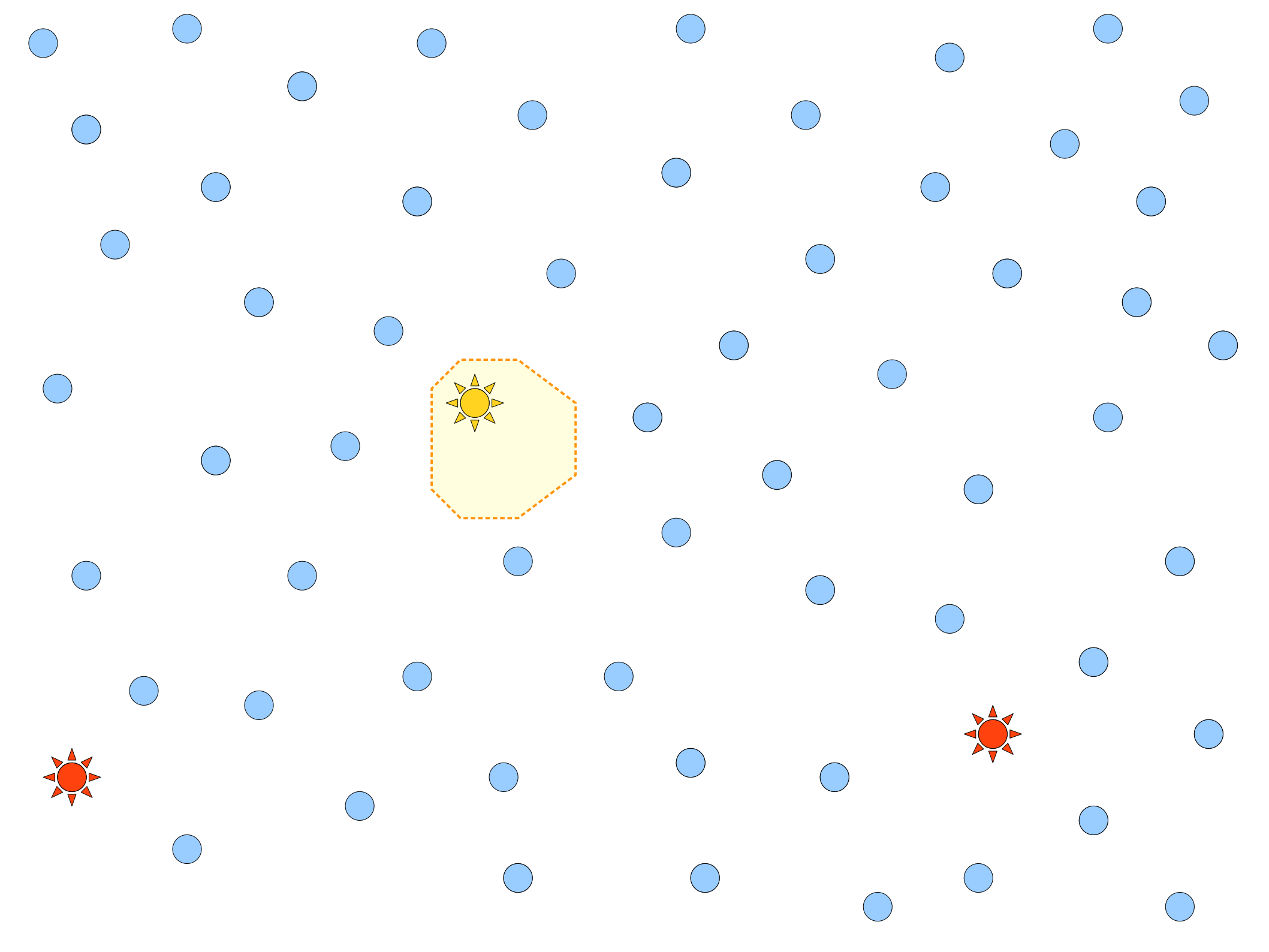
Empirically, our hypergraphs will have V nodes and $E = \alpha V$ edges for some choice of α.

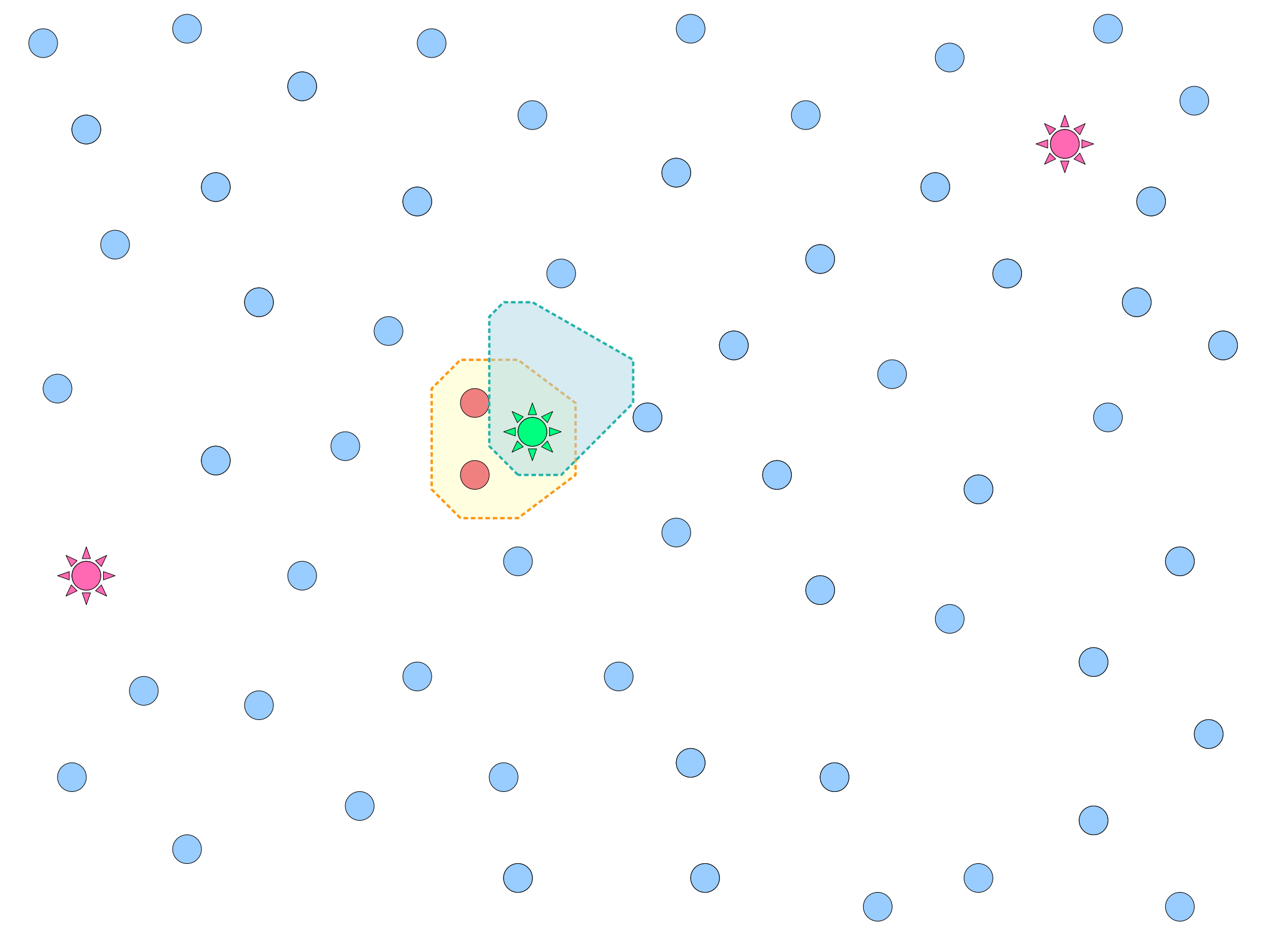
Question: What is the probability that peeling succeeds with d hash functions, m slots, and $n = \alpha m$ items?

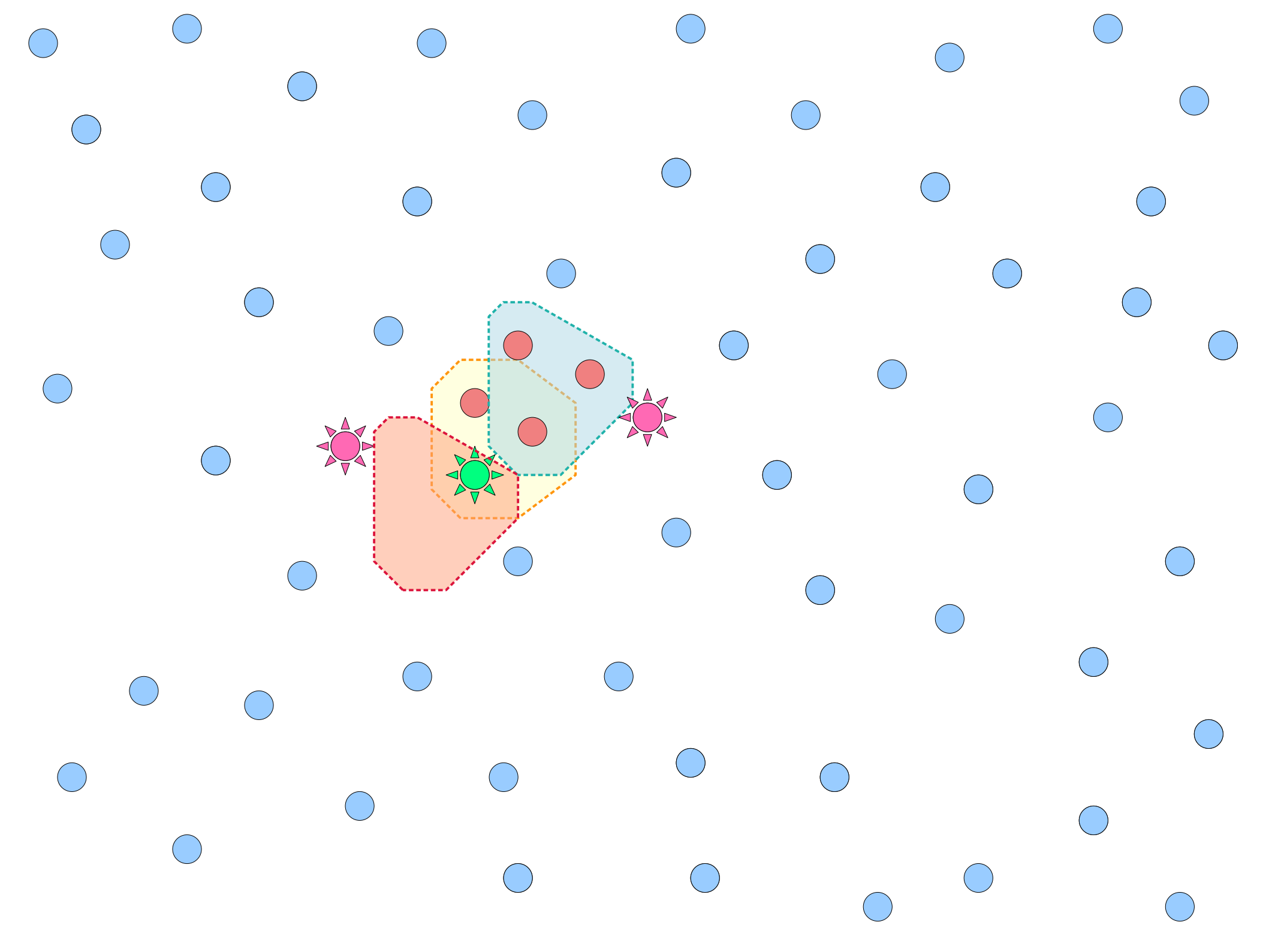
New Question: What is the probability that a random d-uniform hypergraph with V nodes and $E = \alpha V$ edges is peelable?

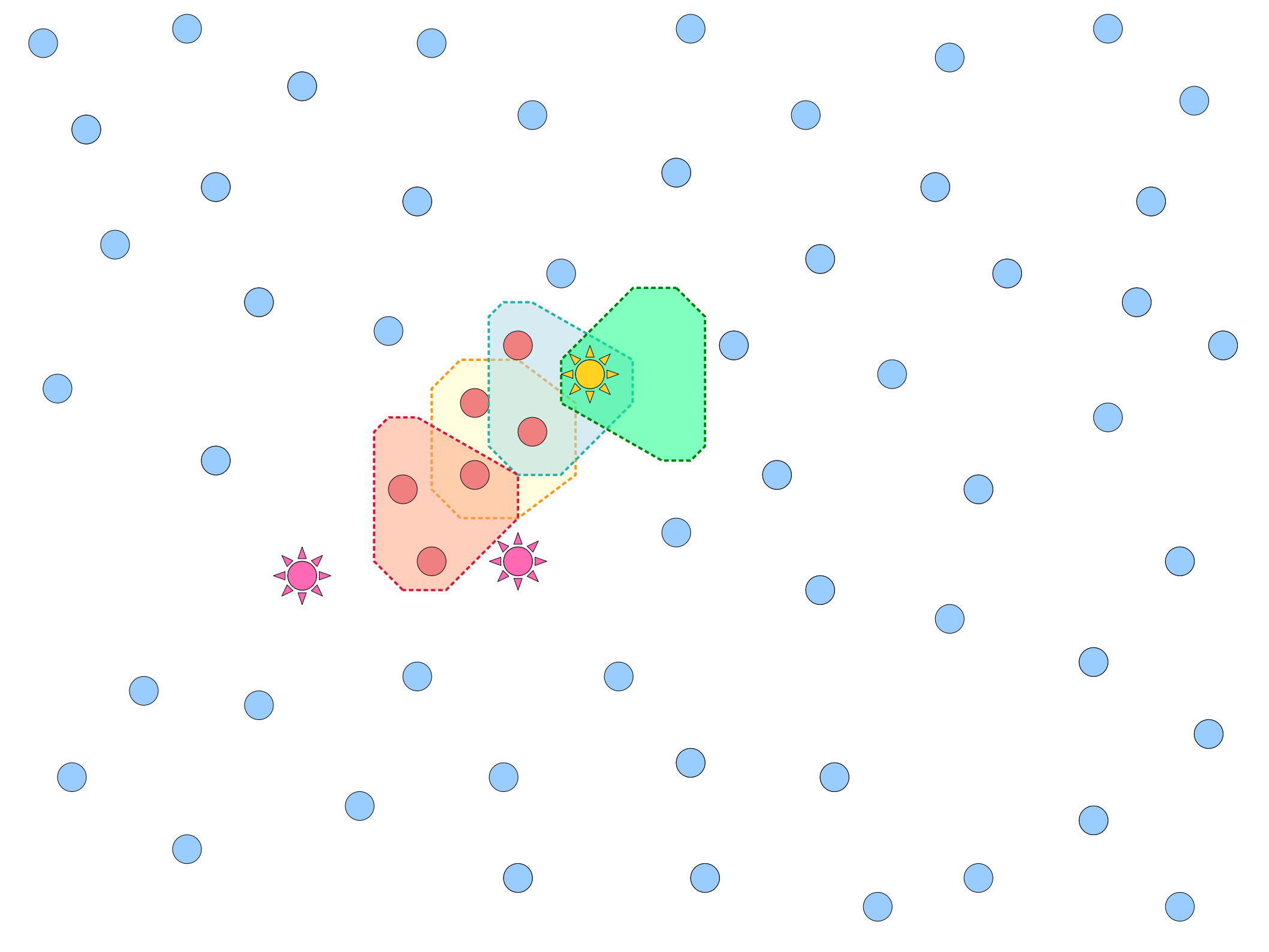
Part 2: What do random hypergraphs look like?

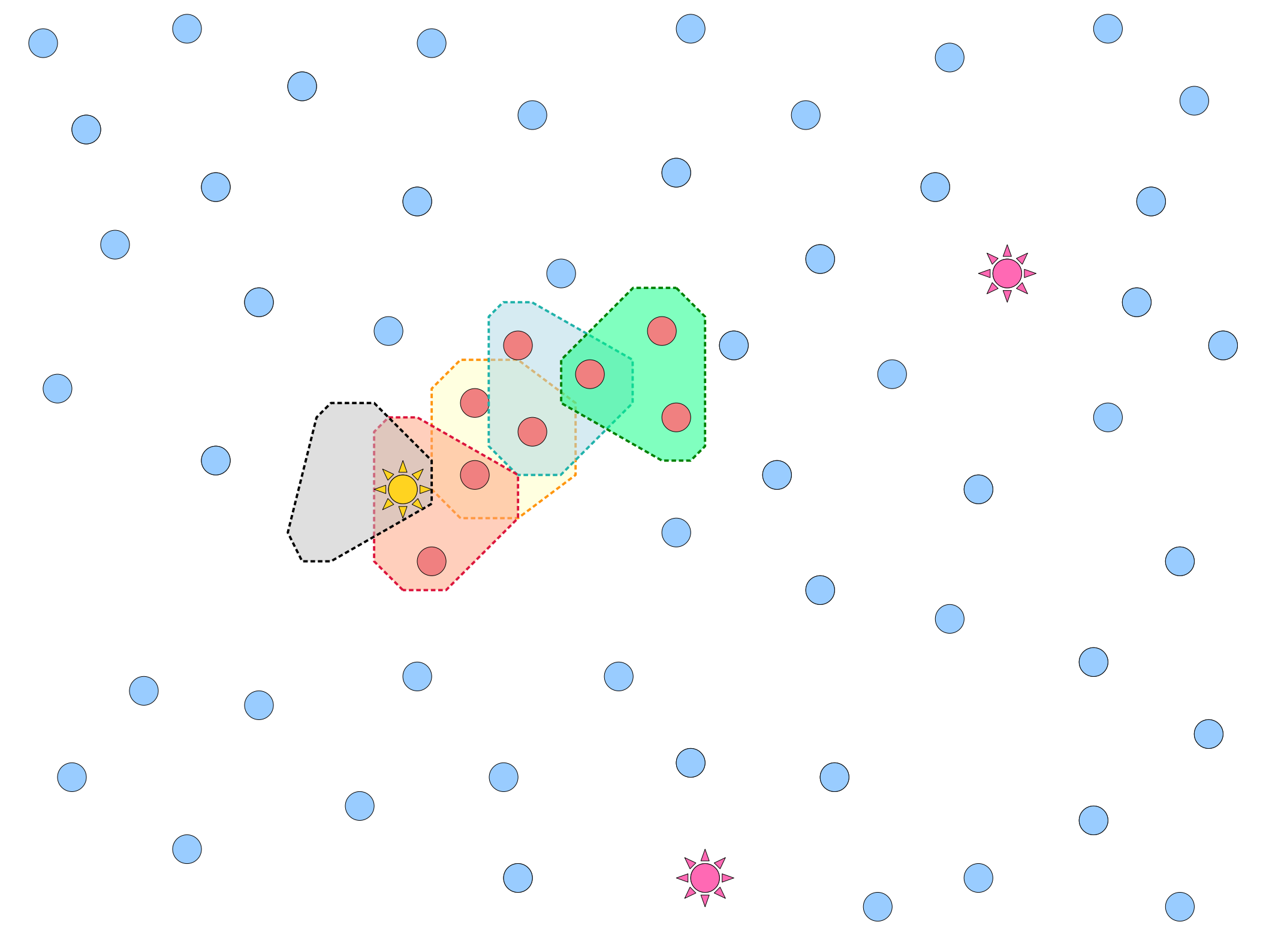


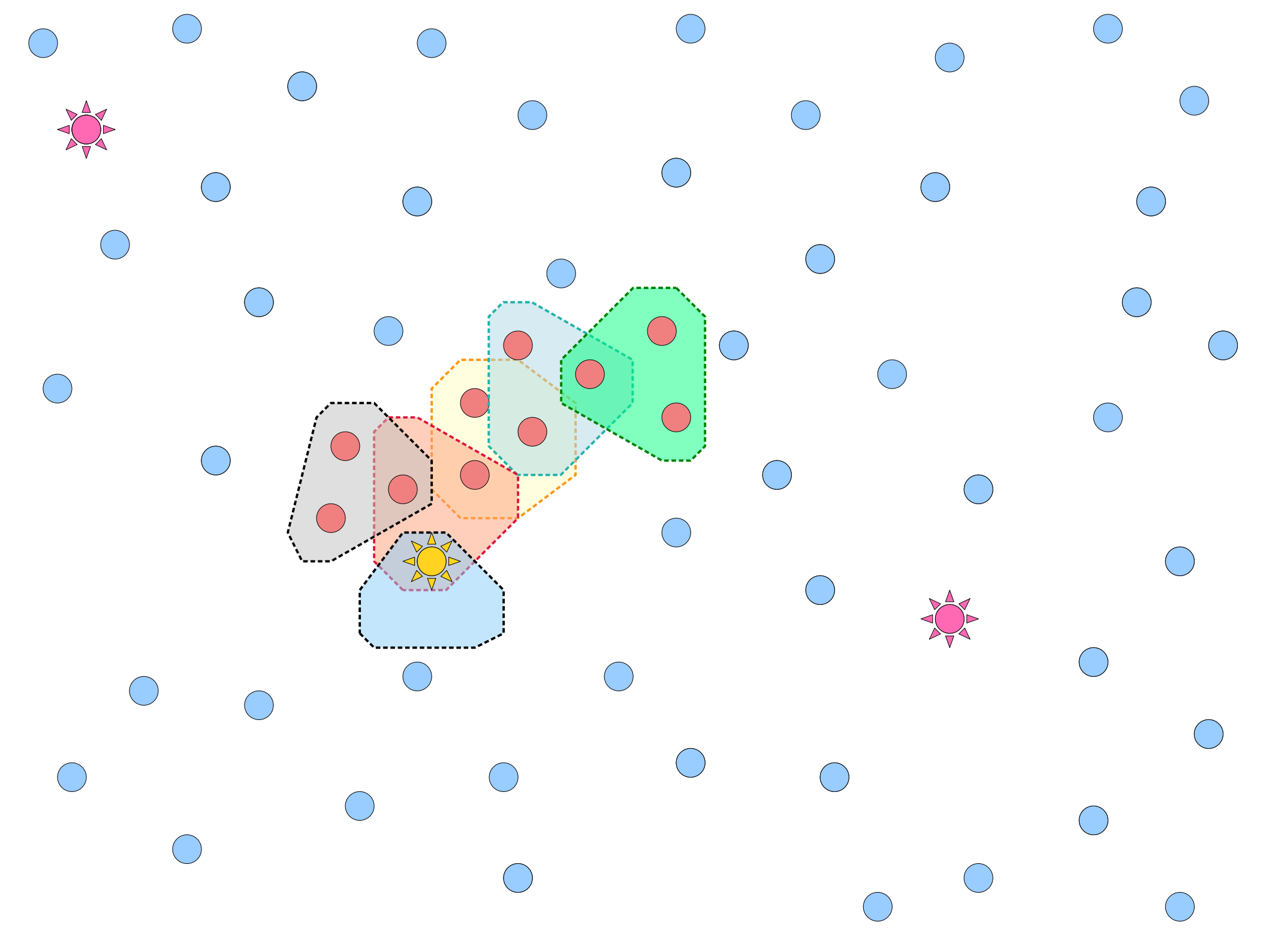


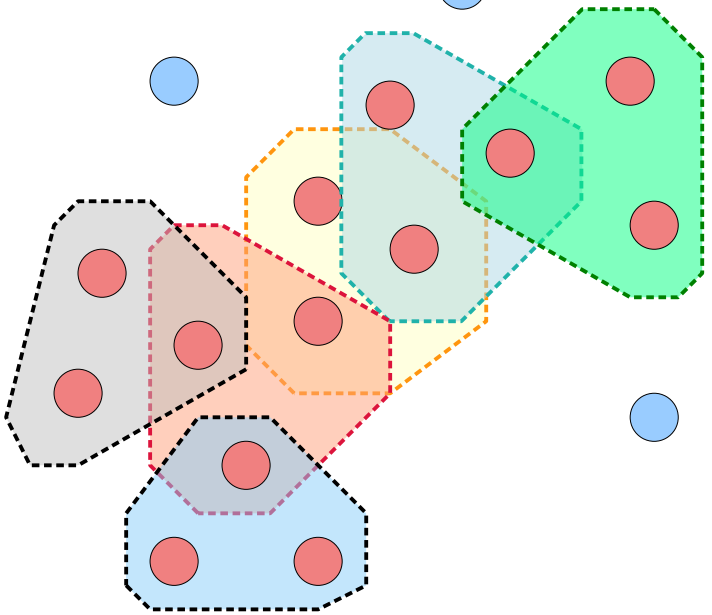












Every time we add a hyperedge touching a ***red*** node, the other endpoints are (probably) ***blue***.

Theorem: If $E = \Theta(V)$, then the nodes at distance $r = O(1)$ from any node v in the hypergraph form a tree with probability $1 - o(1)$.

Part 3: Explore the degrees of nodes in these trees.

Degree Distributions

- Pick any node $x \in V$. Recall that $E = \alpha V$ and that the hypergraph is d-uniform.
- The degree of x (the number of hyperedges touching it) is a random variable. What kind of random variable is it, and what are its parameters?

Answer at

https://cs166.stanford.edu/pollev

Degree Distributions

- Pick any node $x \in V$. Recall that $E = \alpha V$ and that the hypergraph is d-uniform.
- The degree of x (the number of hyperedges touching it) is a random variable. What kind of random variable is it, and what are its parameters?
- The probability that any one hyperedge touches x is d/V.
- Thus the degree of x is a Binom(E, d/V) variable.
- We'll approximate it as a **Poisson(αd)** variable.

Degree Distributions

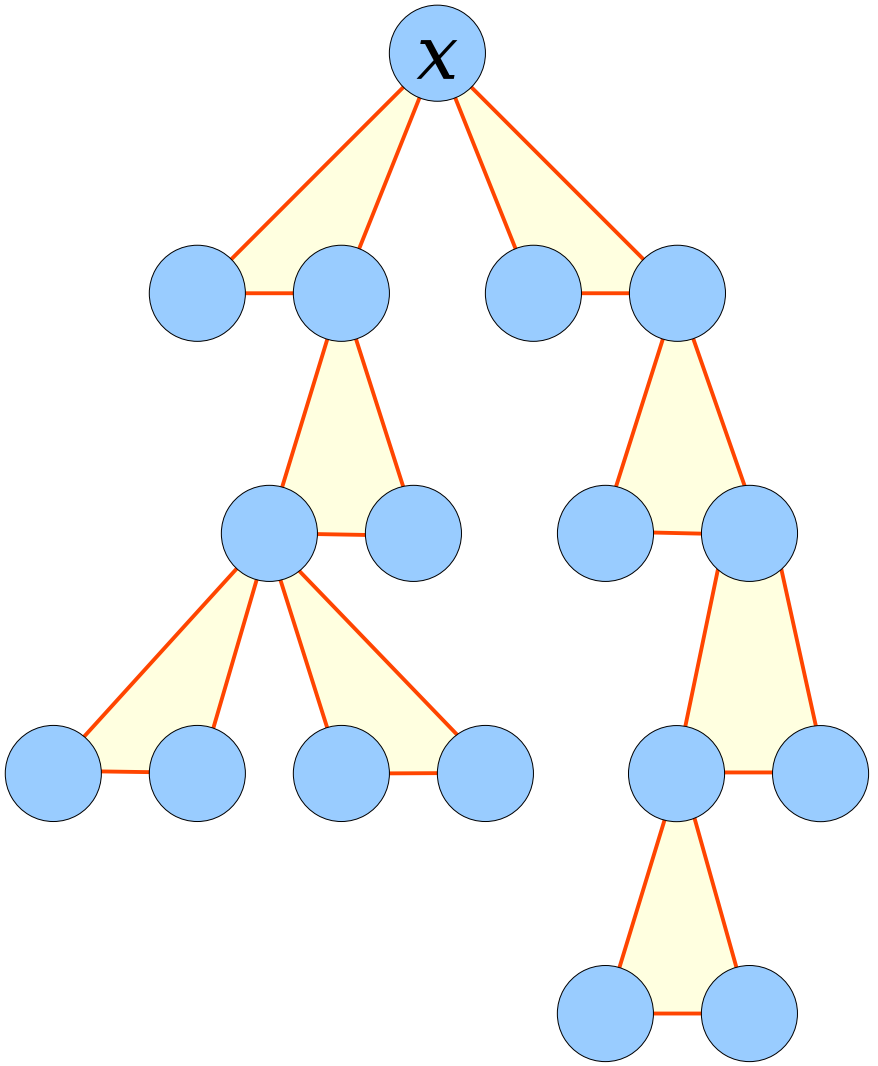
- Empirically, with $d = 3$, we need to have $\alpha < 0.81$. Thus when $d = 3$ we have $d\alpha \approx 2.43$.
- This means that a Poisson($d\alpha$) variable on expectation is greater than 1.

The phase transition is not due to exponential growth/decay!

- What's going on?

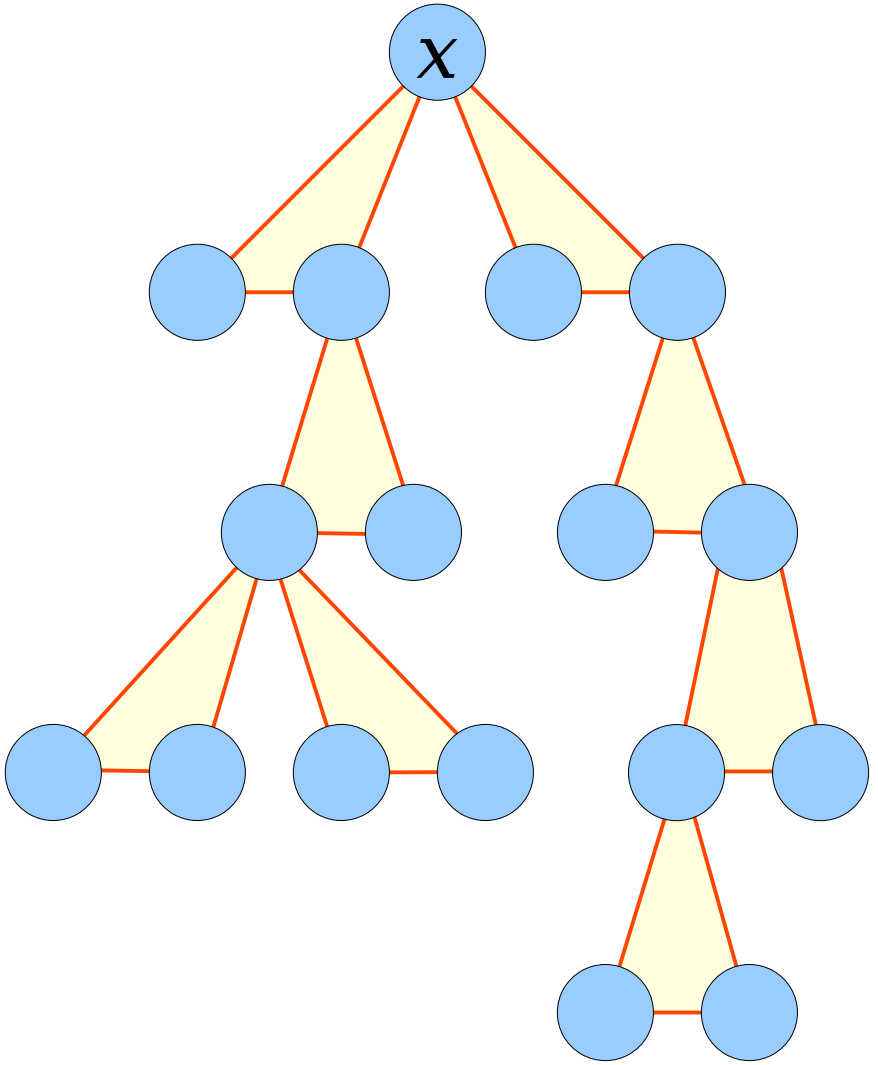
Exploring the Neighborhood

- The neighborhood of a node x in our hypergraph is (almost certainly) tree-shaped.
- The root node has **Poisson($d\alpha$)** child edges, each of which touch d – 1 nodes.
- (We need to distinguish "child edges" and "child nodes" because we're in a hypergraph. Yikes!)



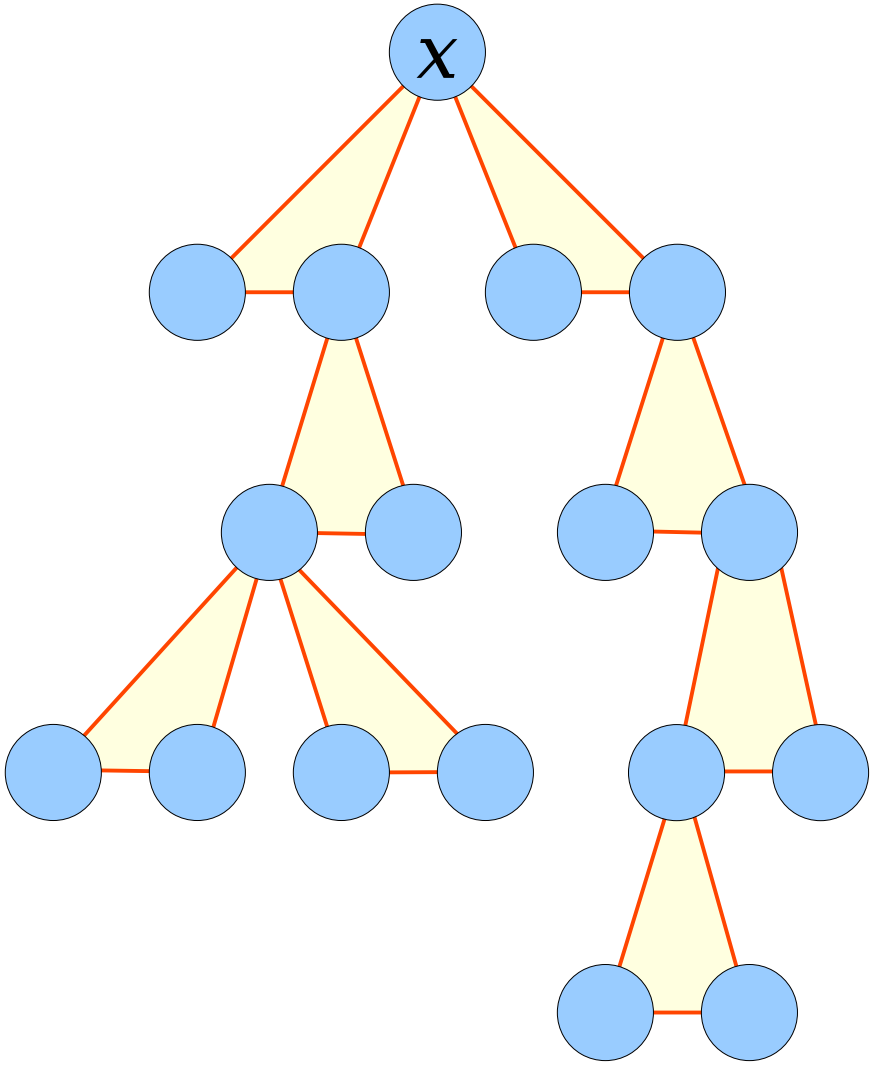
Exploring the Neighborhood

- ***Claim:*** More generally, *every* node in the tree has a child edge count that is well-modeled by a Poisson(αd) variable.
- Why?
  - With constant size, only a few other edges out of the total are used up.
  - We already are assuming the hypergraph is tree-shaped.

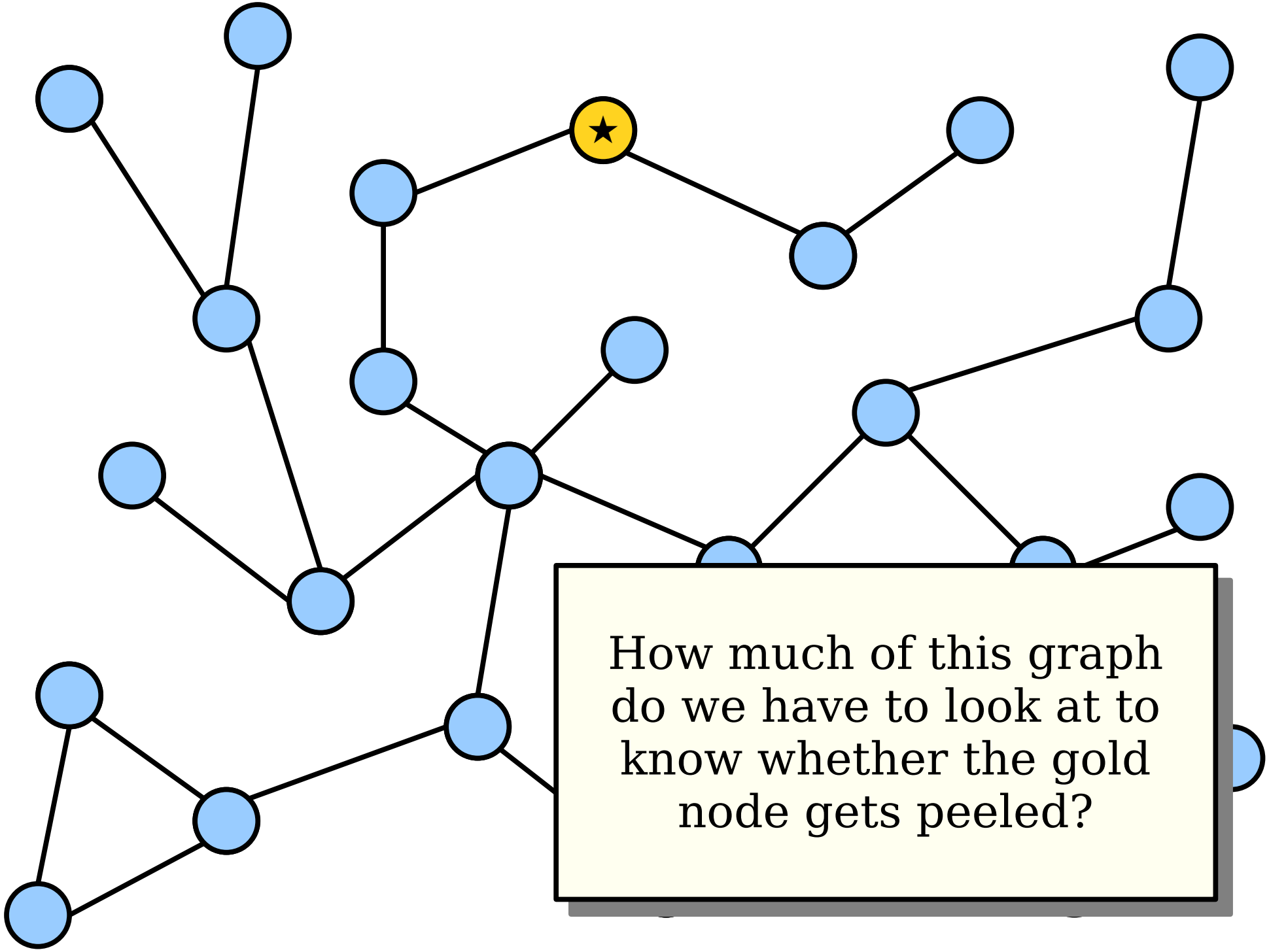


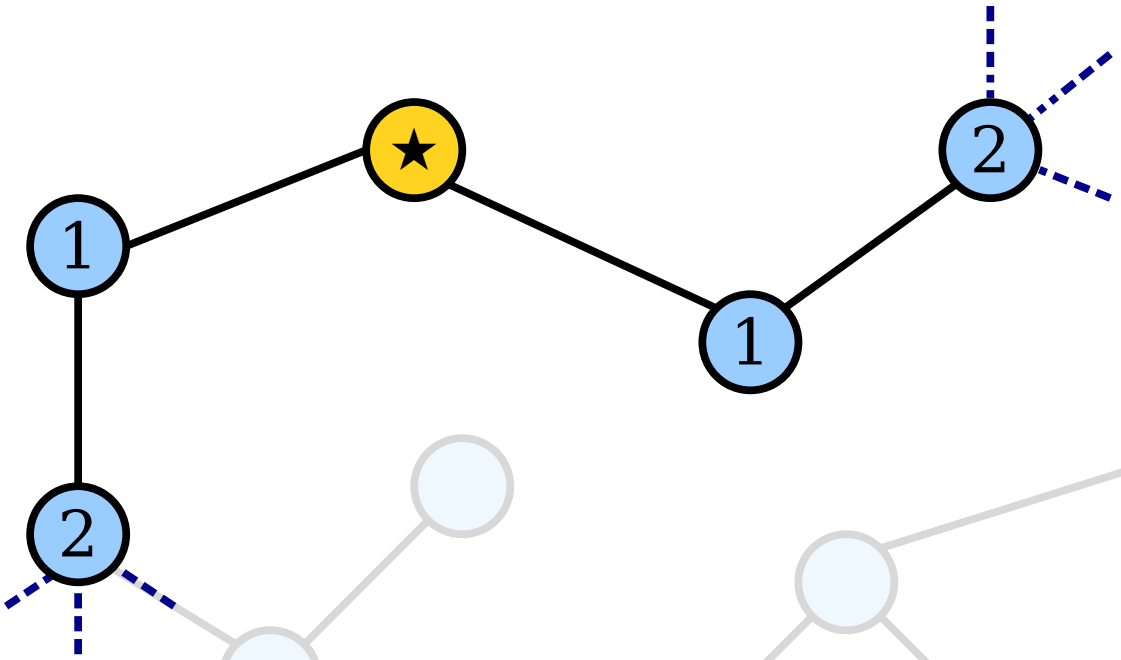
The Story So Far

- Pick a node x and look at the neighborhood of nodes at distance $r = O(1)$ from x.
- It's (almost certainly) a tree.
- Each node in that tree has (roughly) **Poisson(αd)** child edges.
- Great! Now what?



Part 4: Investigate peeling on these trees.

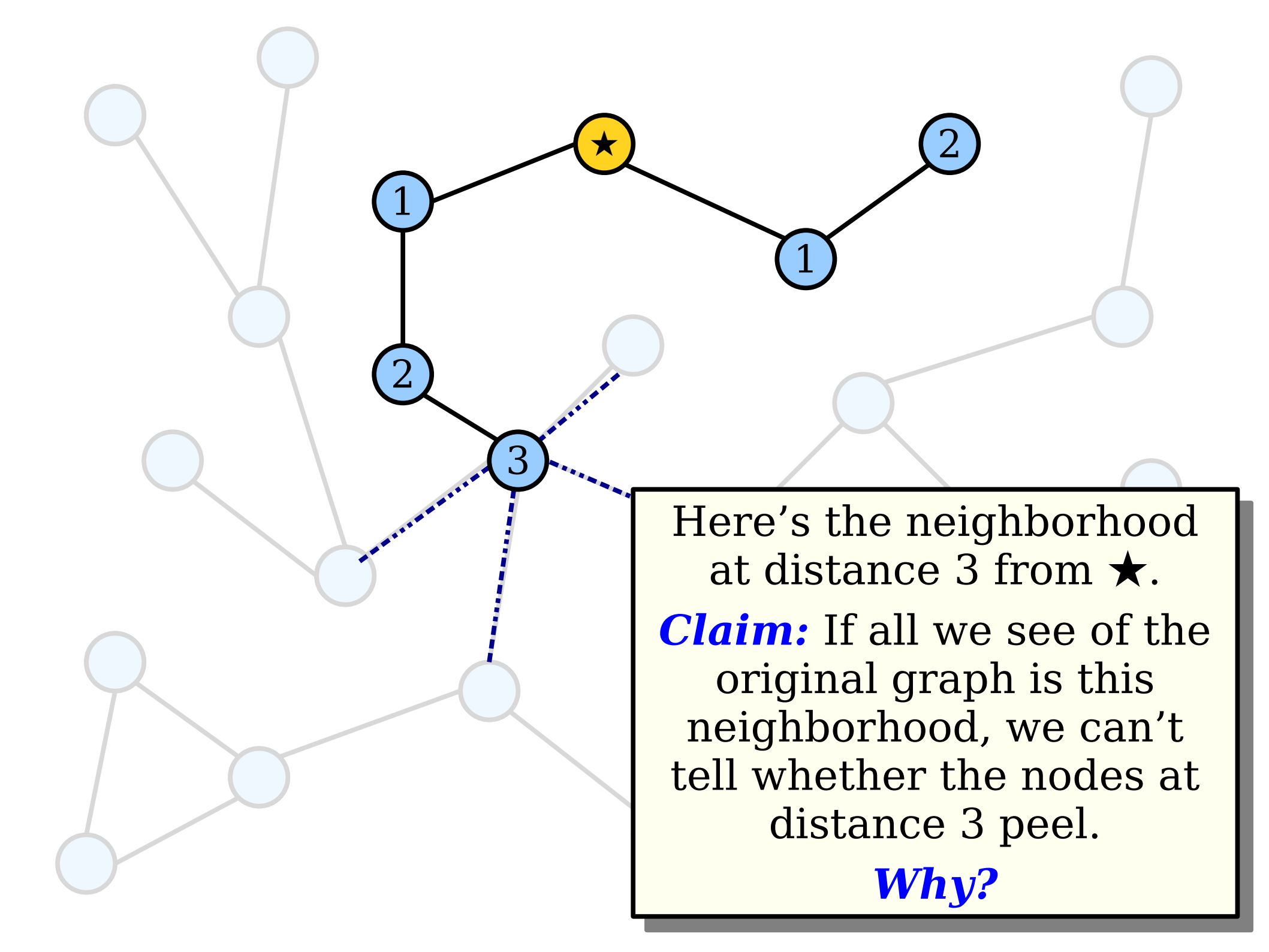

How much of this graph do we have to look at to know whether the gold node gets peeled?



Here's the neighborhood at distance 2 from ★.

Claim: If all we see of the original graph is this neighborhood, we can't tell whether the nodes at distance 2 peel.

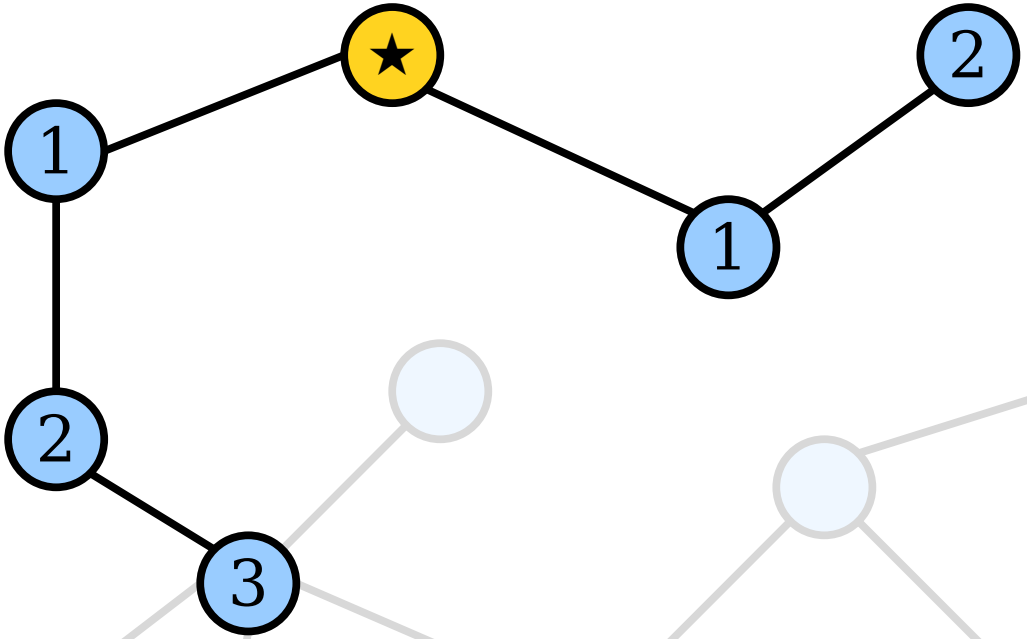
Why?



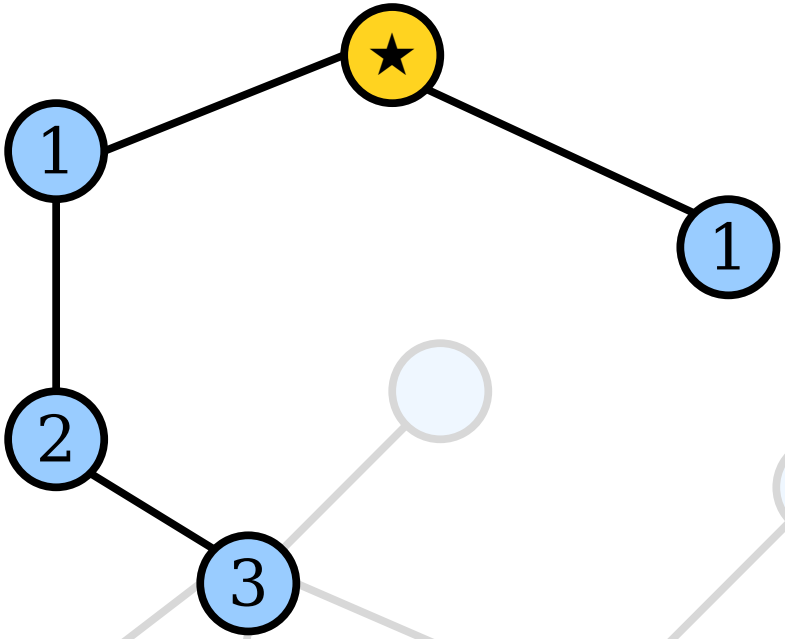
Here's the neighborhood at distance 3 from ★.

Claim: If all we see of the original graph is this neighborhood, we can't tell whether the nodes at distance 3 peel.

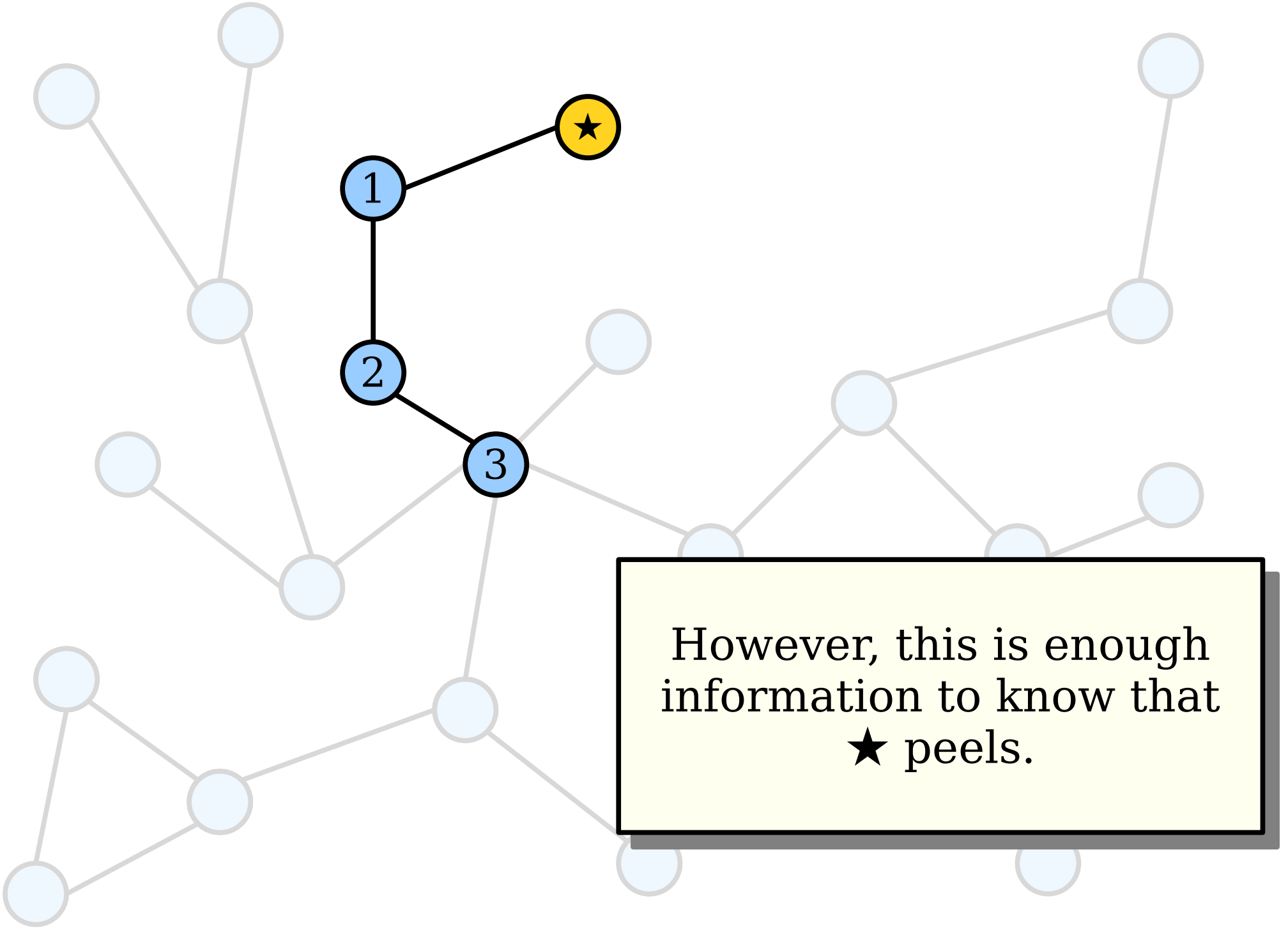
Why?



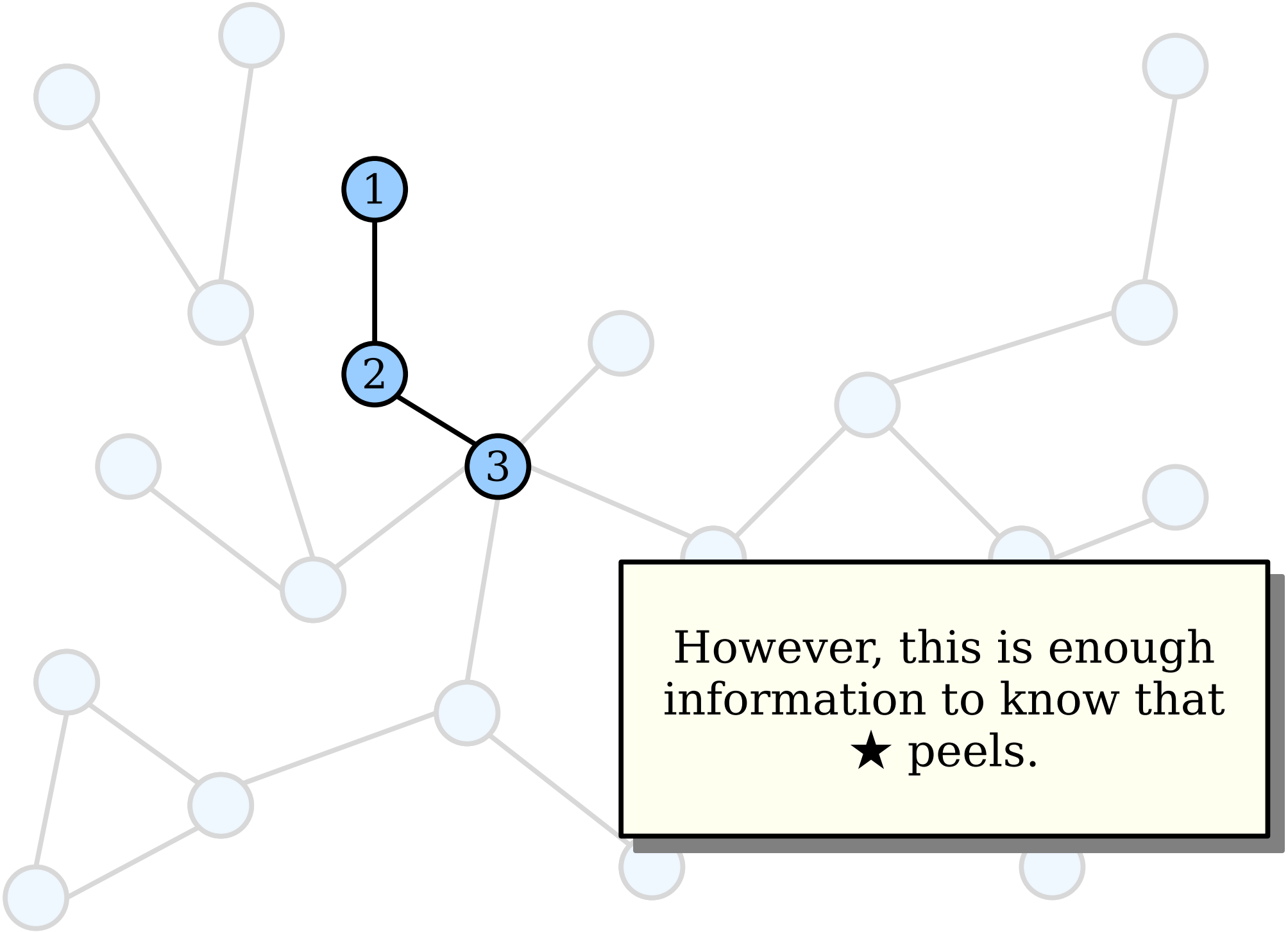
However, this is enough information to know that ★ peels.



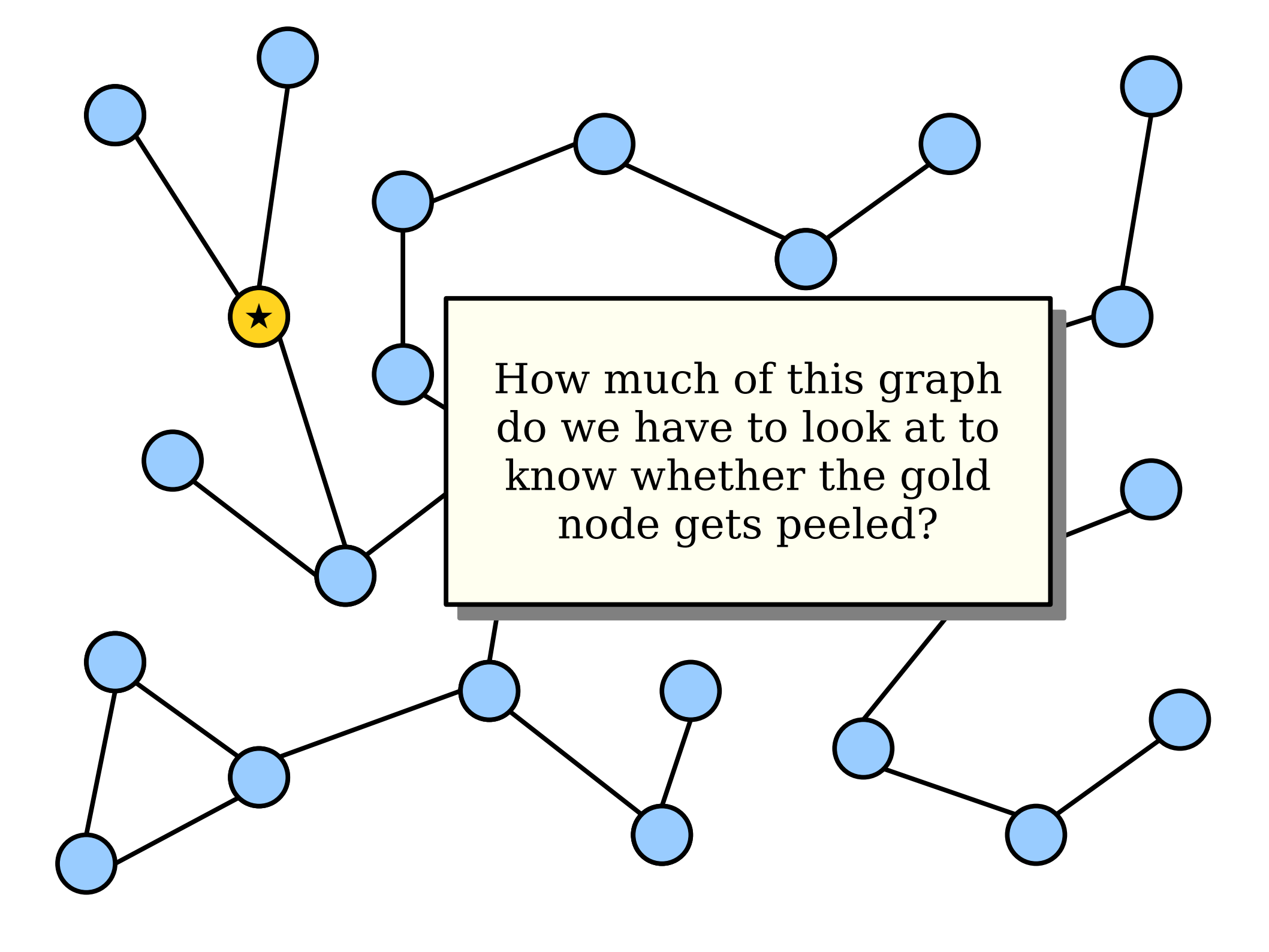
However, this is enough information to know that ★ peels.

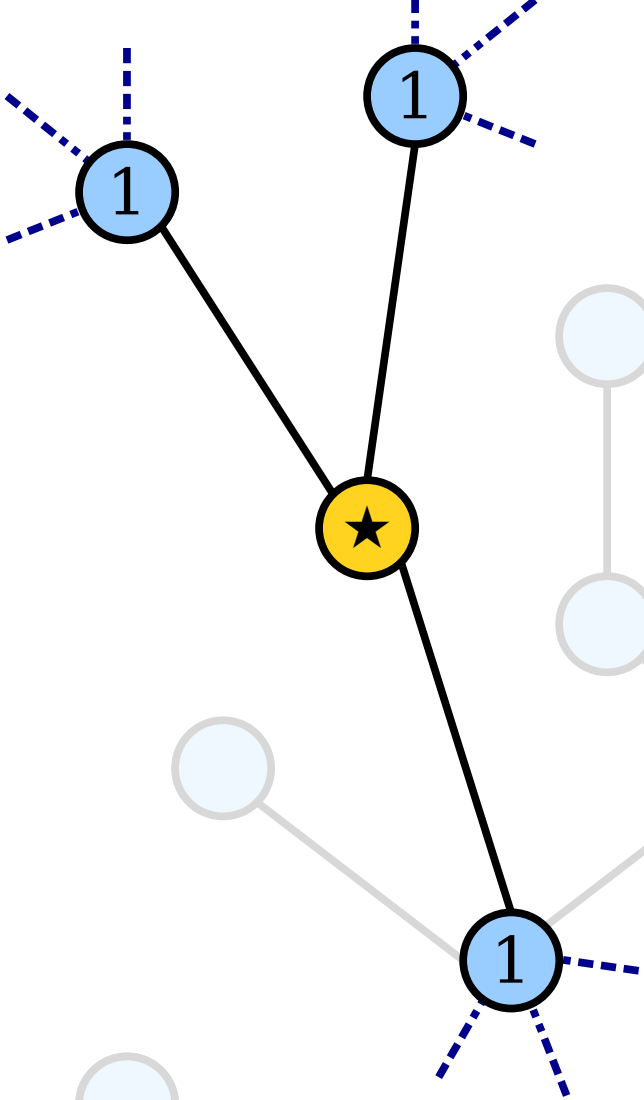


However, this is enough information to know that ★ peels.



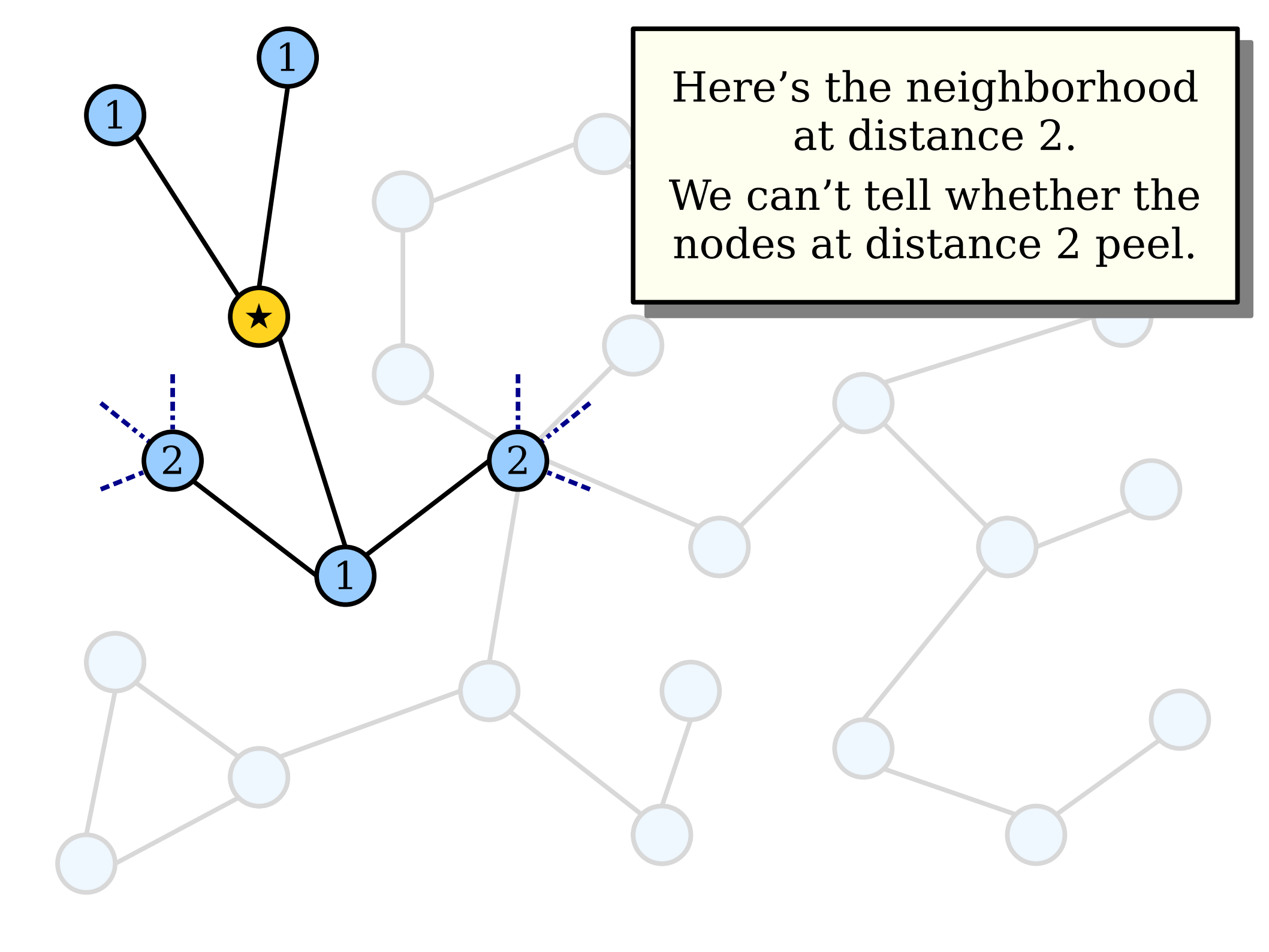
However, this is enough information to know that ★ peels.


How much of this graph do we have to look at to know whether the gold node gets peeled?



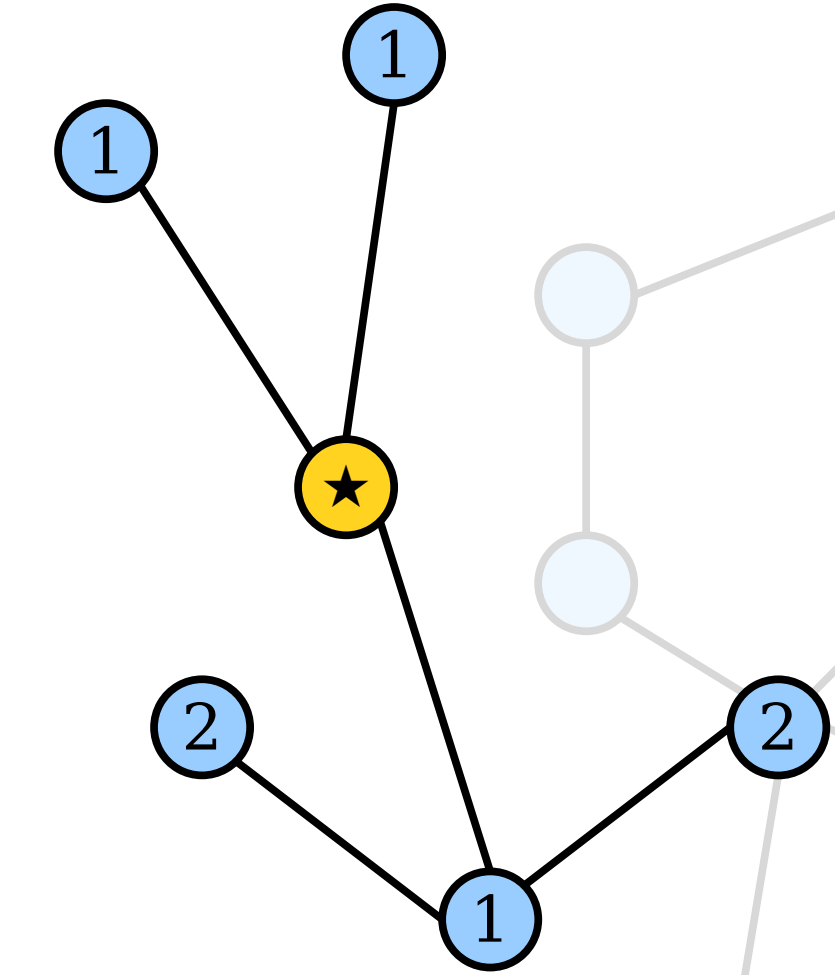
Here’s the neighborhood at distance 1.

We can’t tell whether the nodes at distance 1 peel.



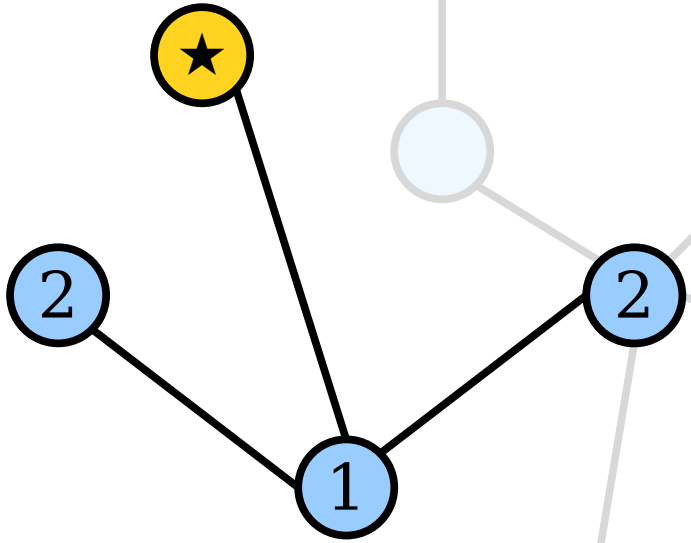
Here's the neighborhood at distance 2.

We can't tell whether the nodes at distance 2 peel.

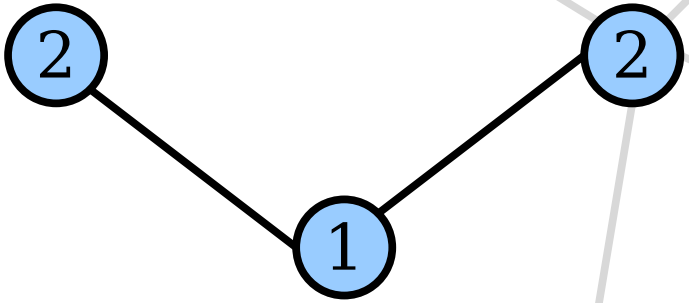


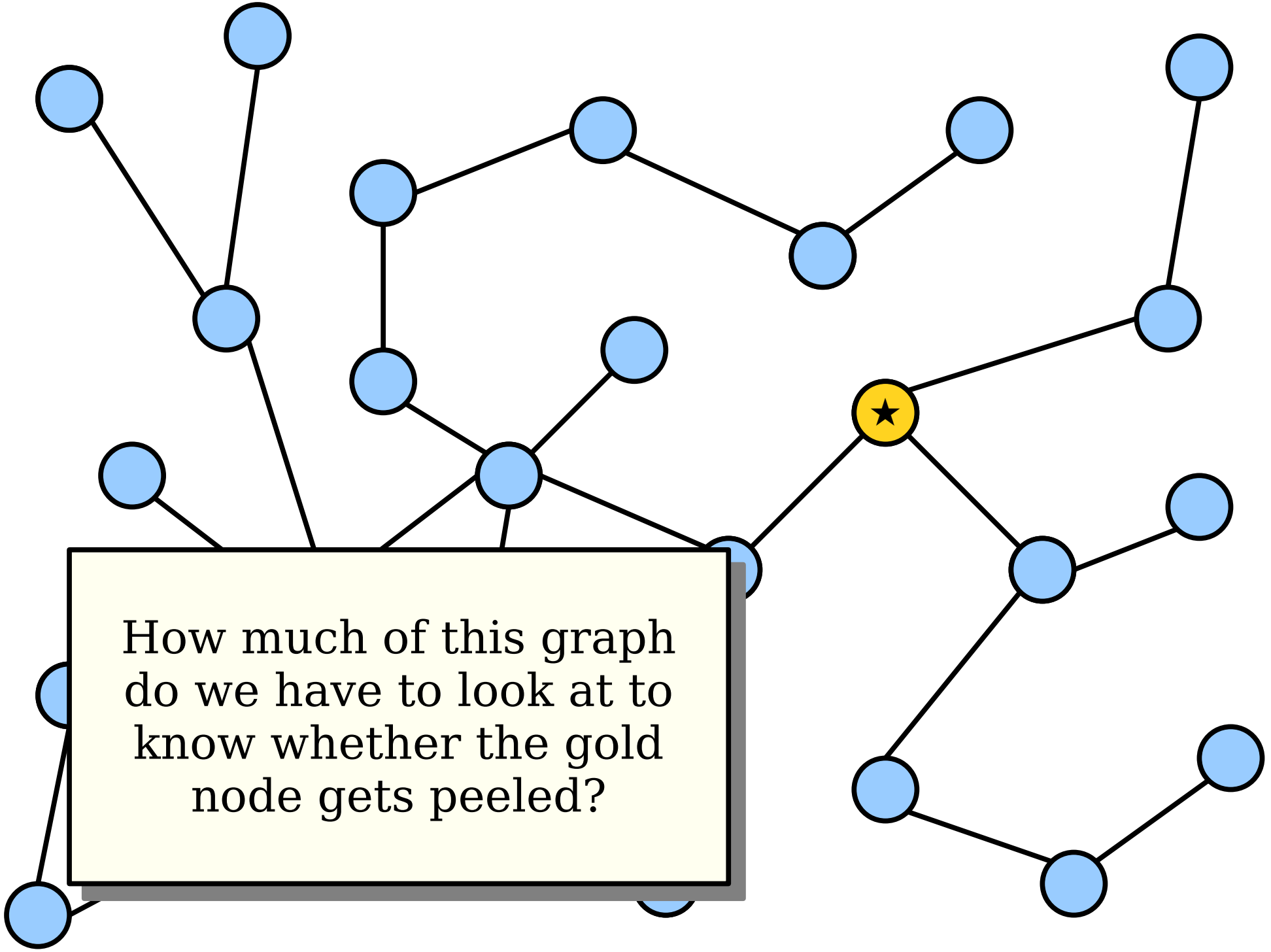
But we do know enough to know that ★ peels.

But we do know enough to know that ★ peels.

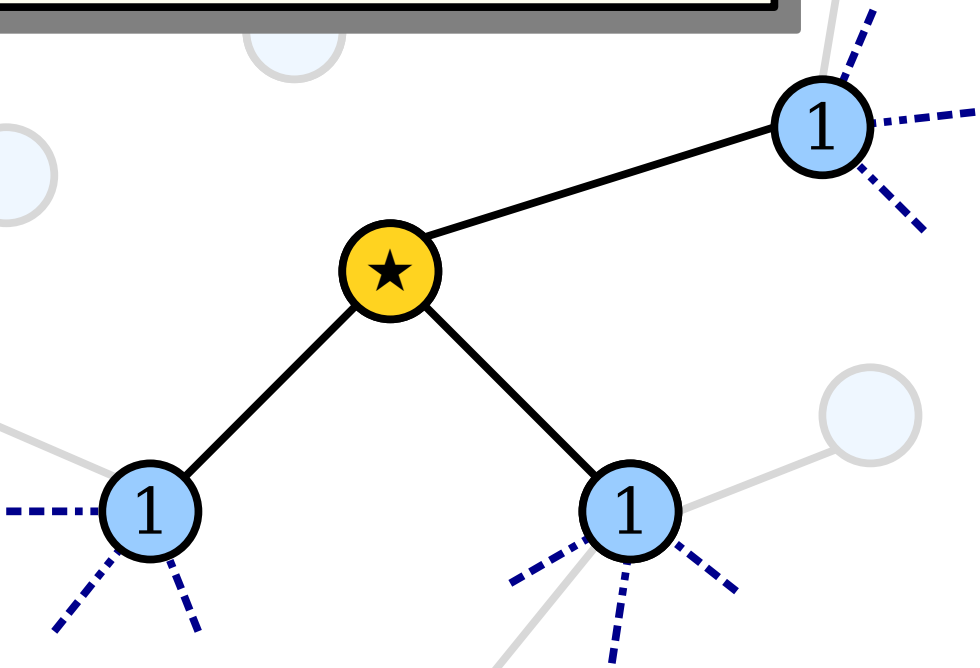


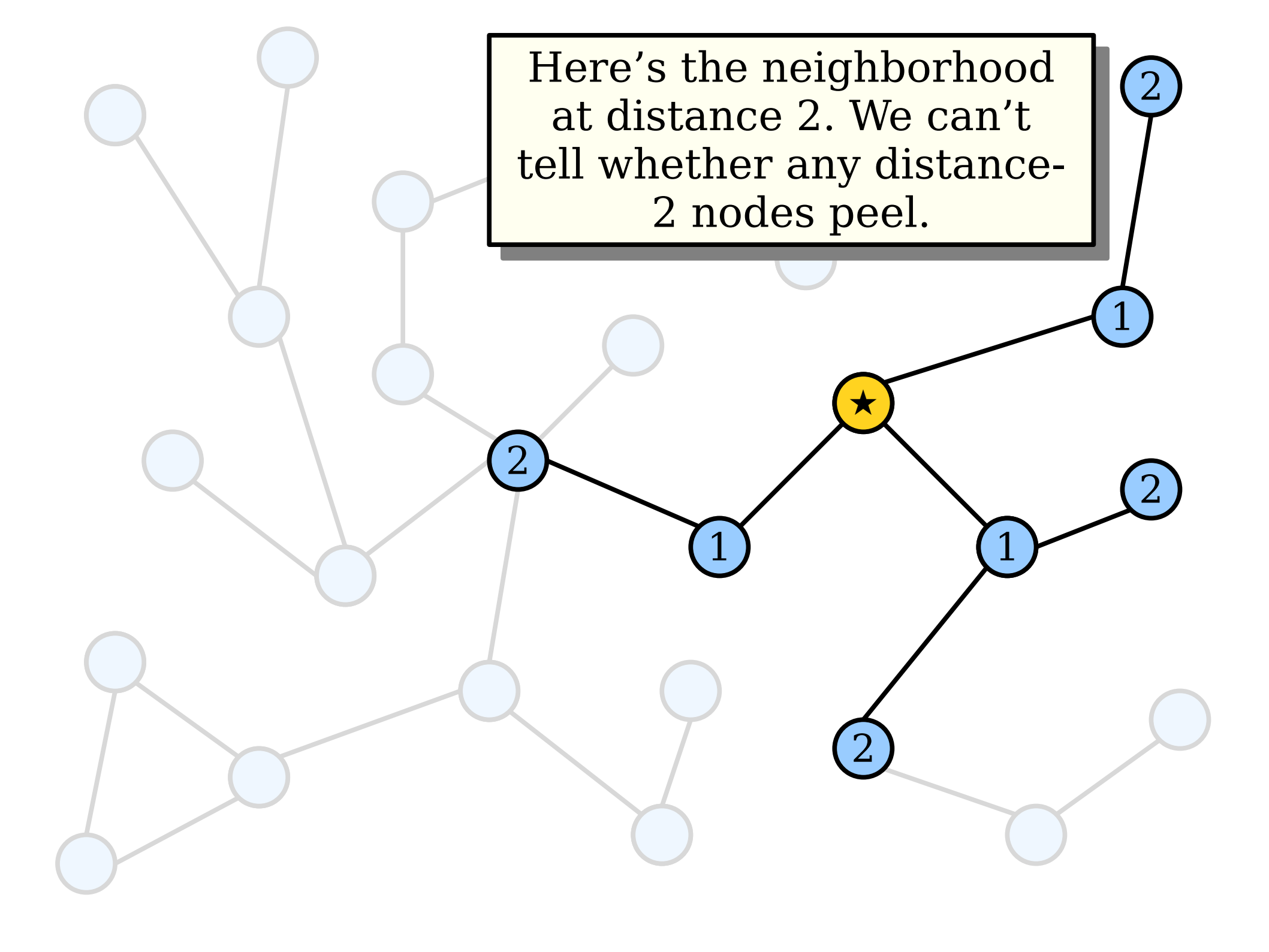
But we do know enough to know that ★ peels.

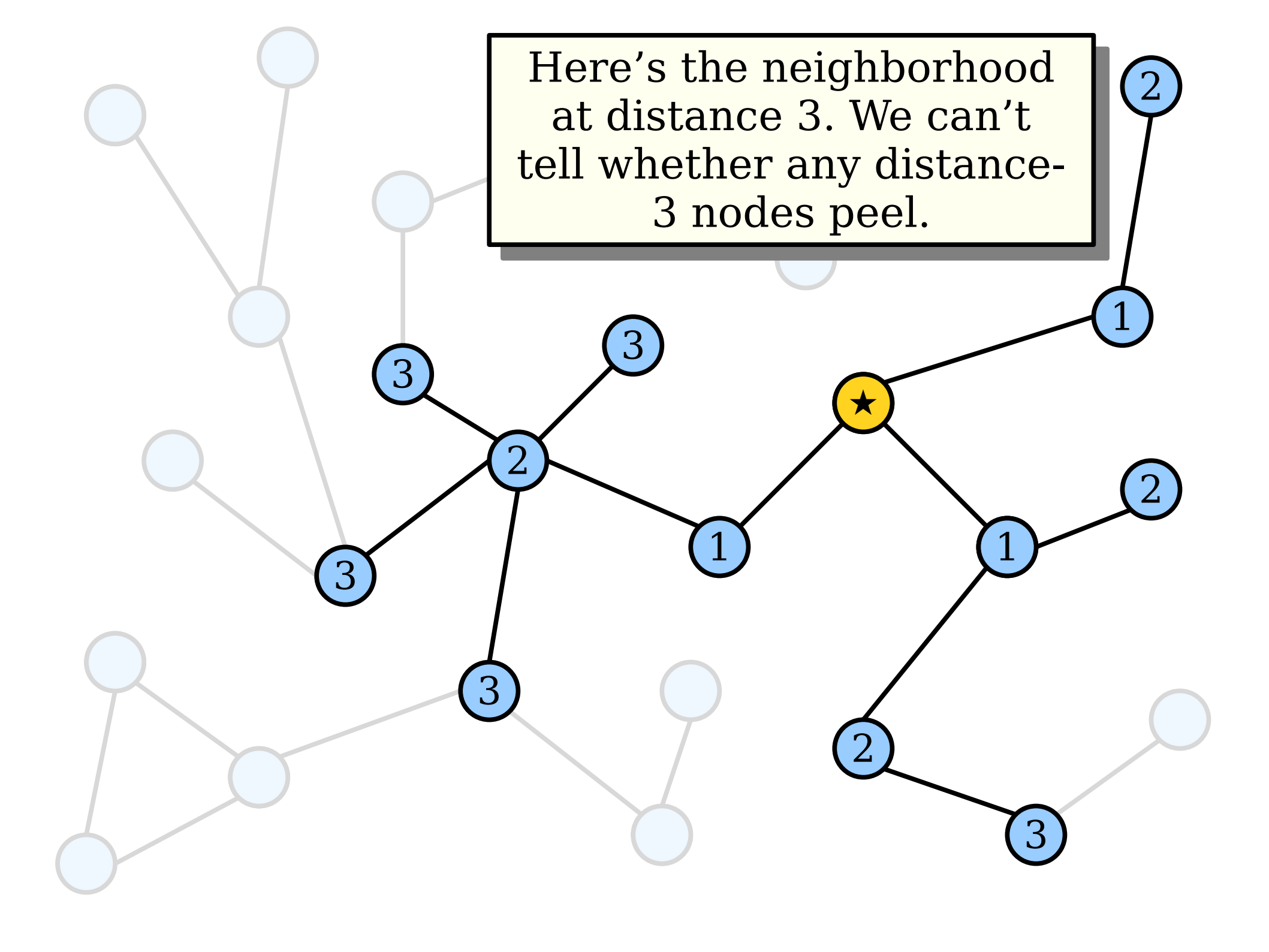


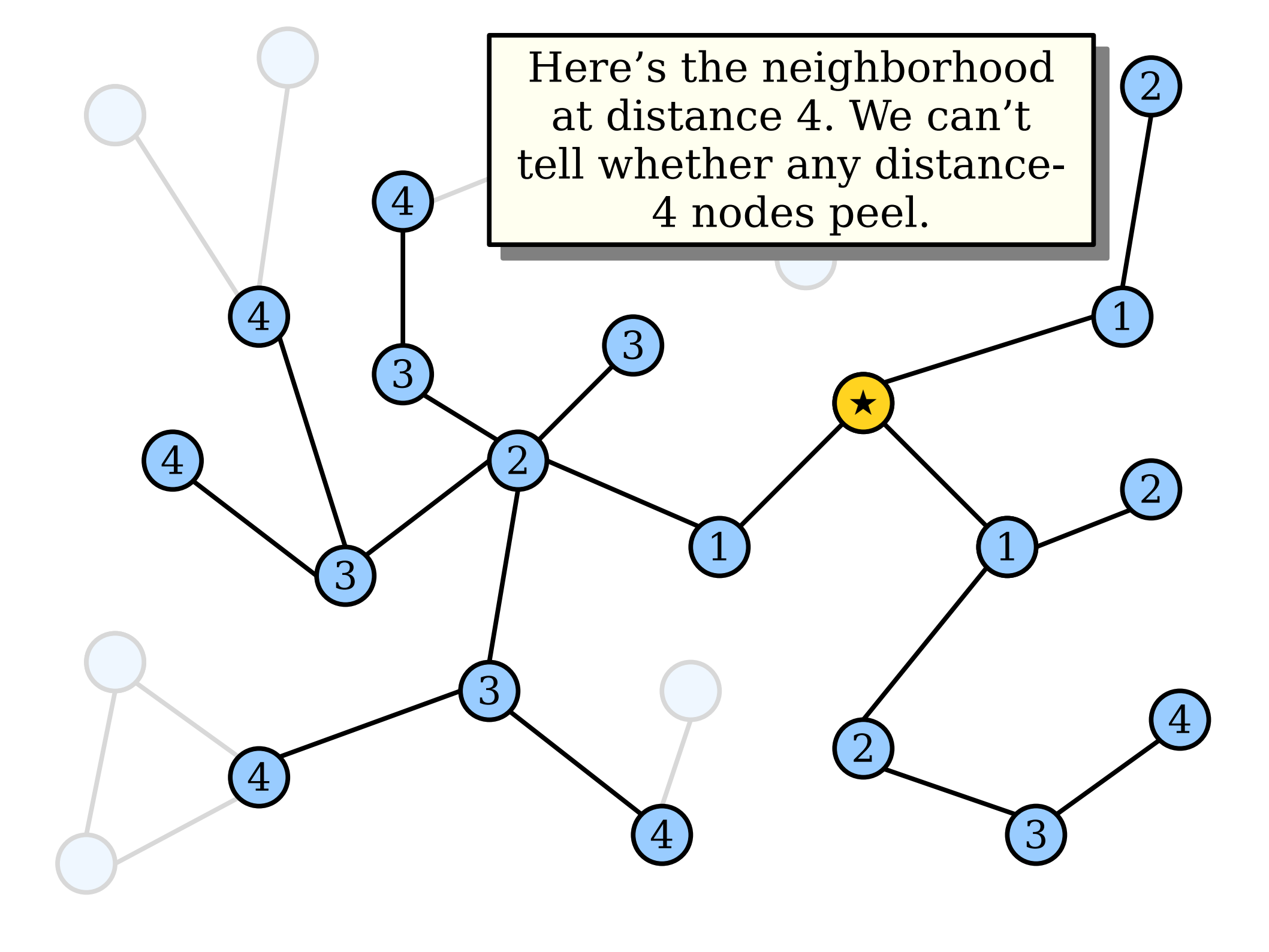

How much of this graph do we have to look at to know whether the gold node gets peeled?

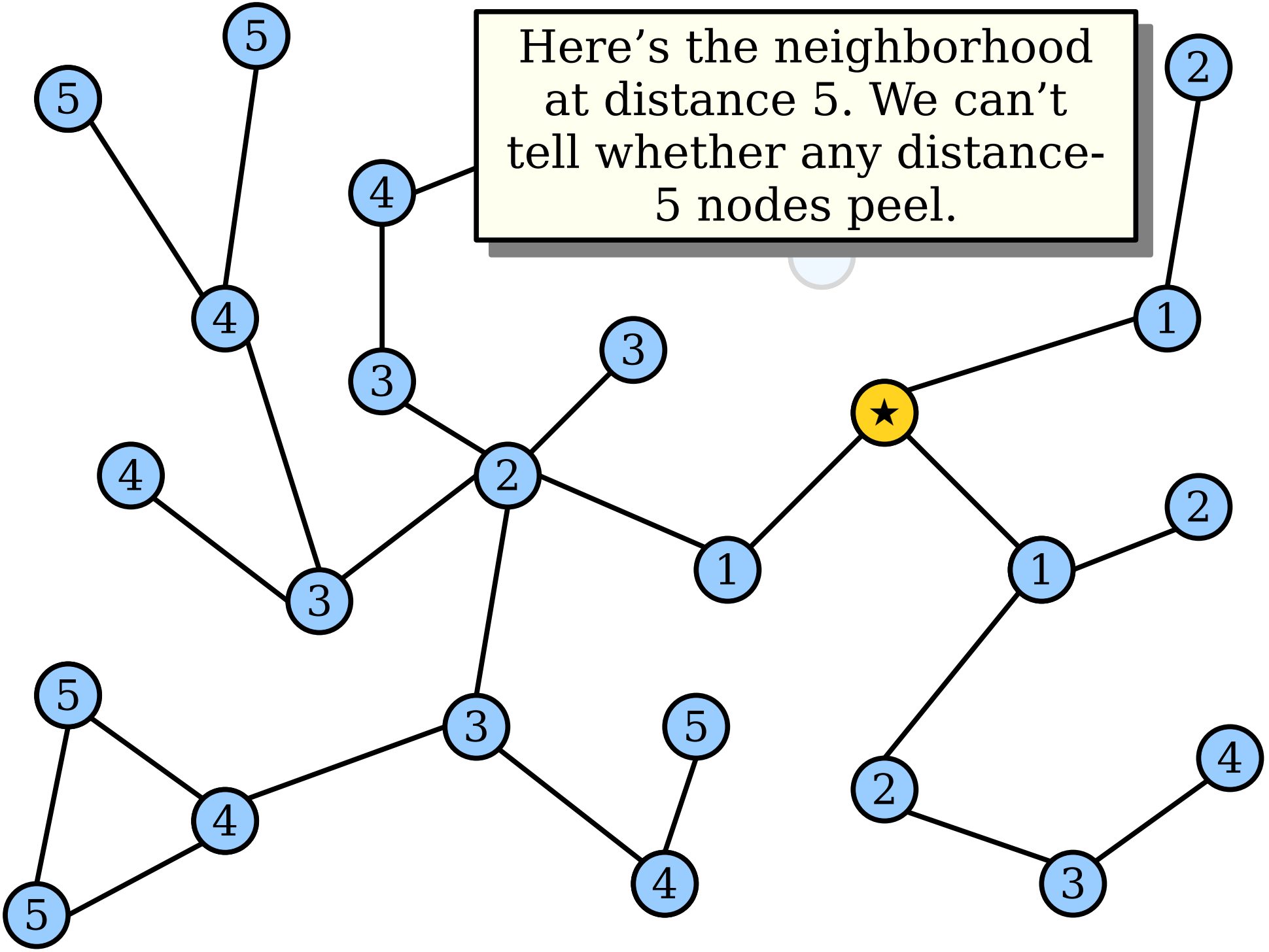
Here's the neighborhood at distance 1. We can't tell whether any distance-1 nodes peel.

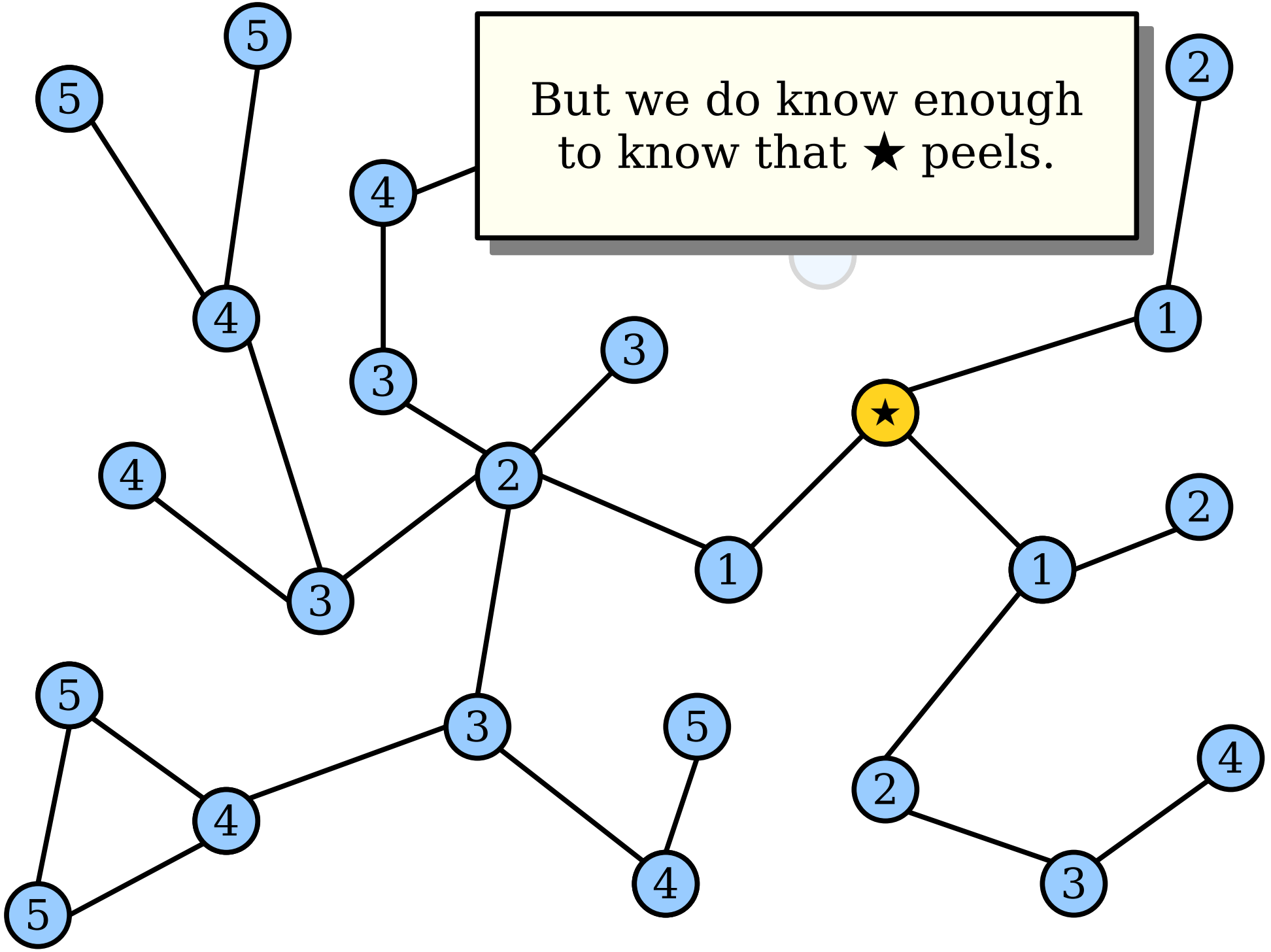


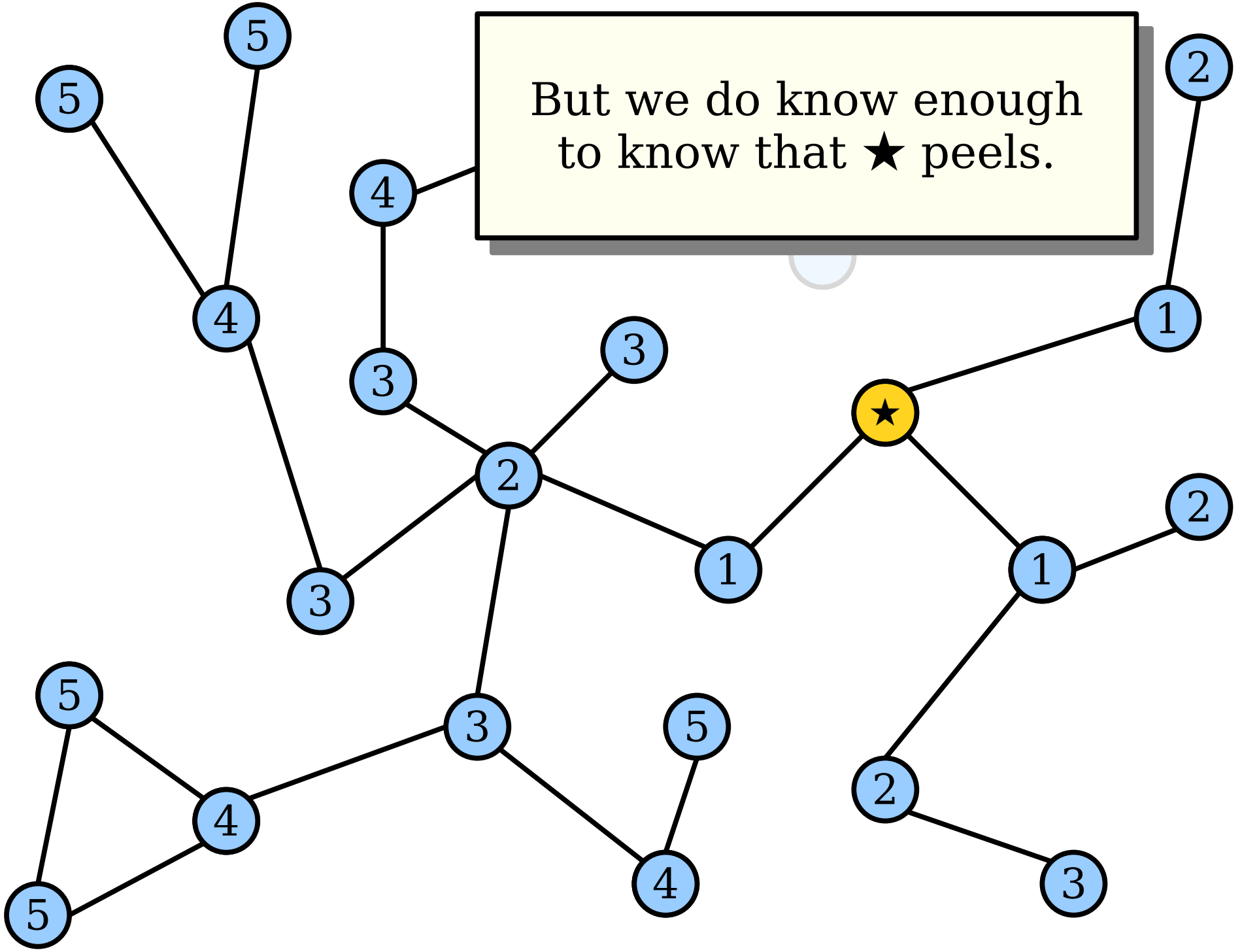

Here’s the neighborhood at distance 2. We can’t tell whether any distance-2 nodes peel.
2
1
★
2
1
1
2
2

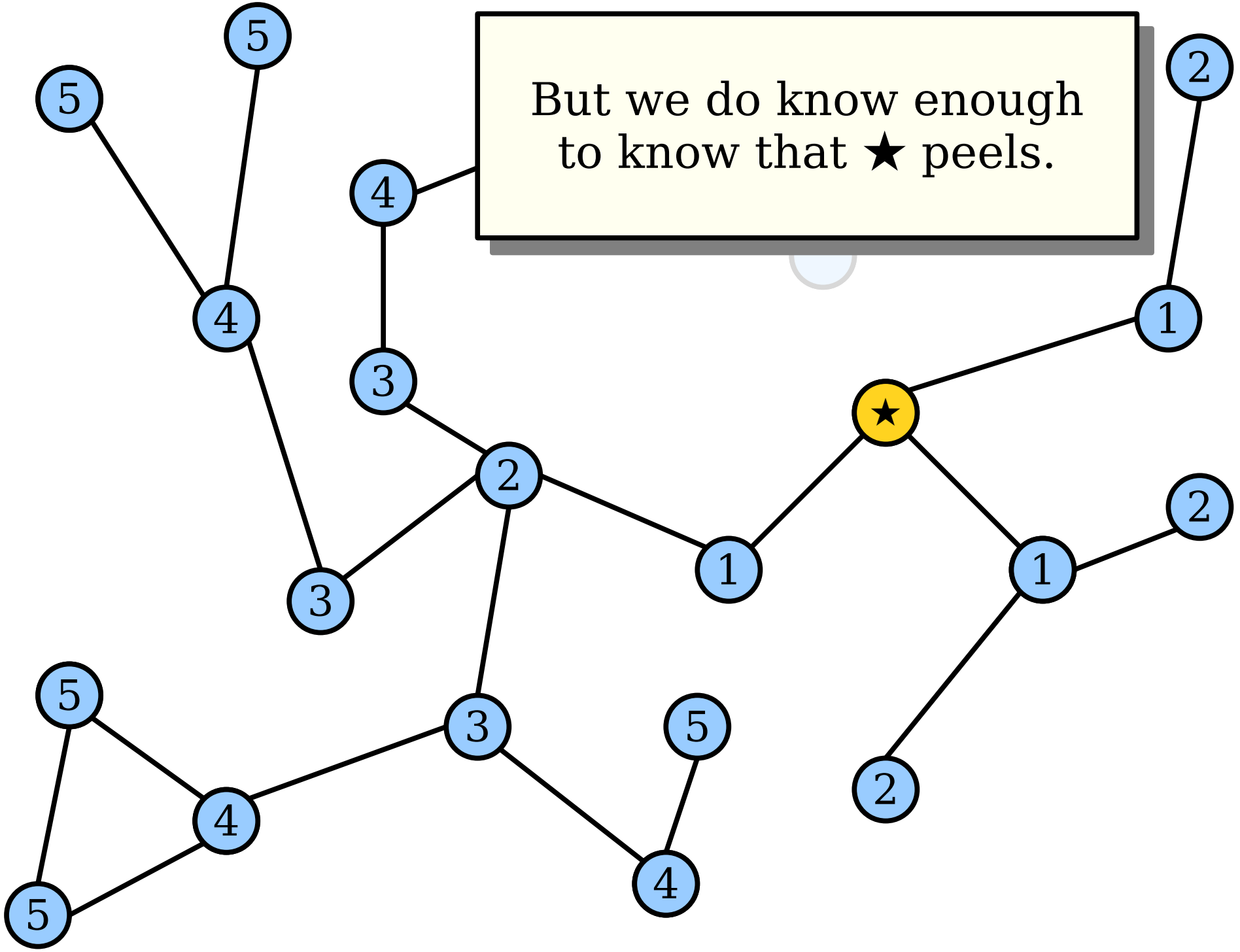

Here’s the neighborhood at distance 3. We can’t tell whether any distance-3 nodes peel.
2
1
3
3
★
2
3
1
1
2
3
2
3

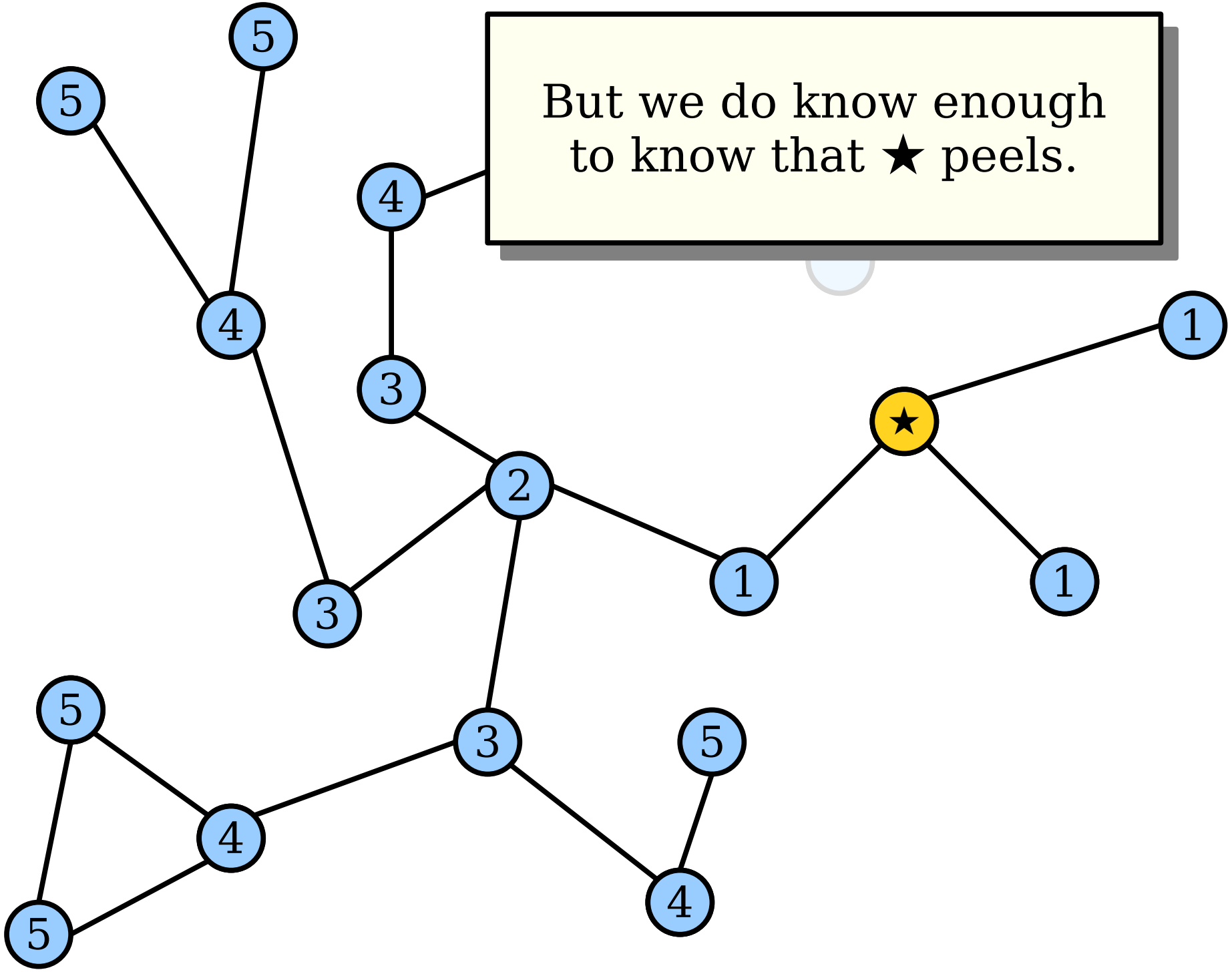

Here’s the neighborhood at distance 4. We can’t tell whether any distance-4 nodes peel.
★

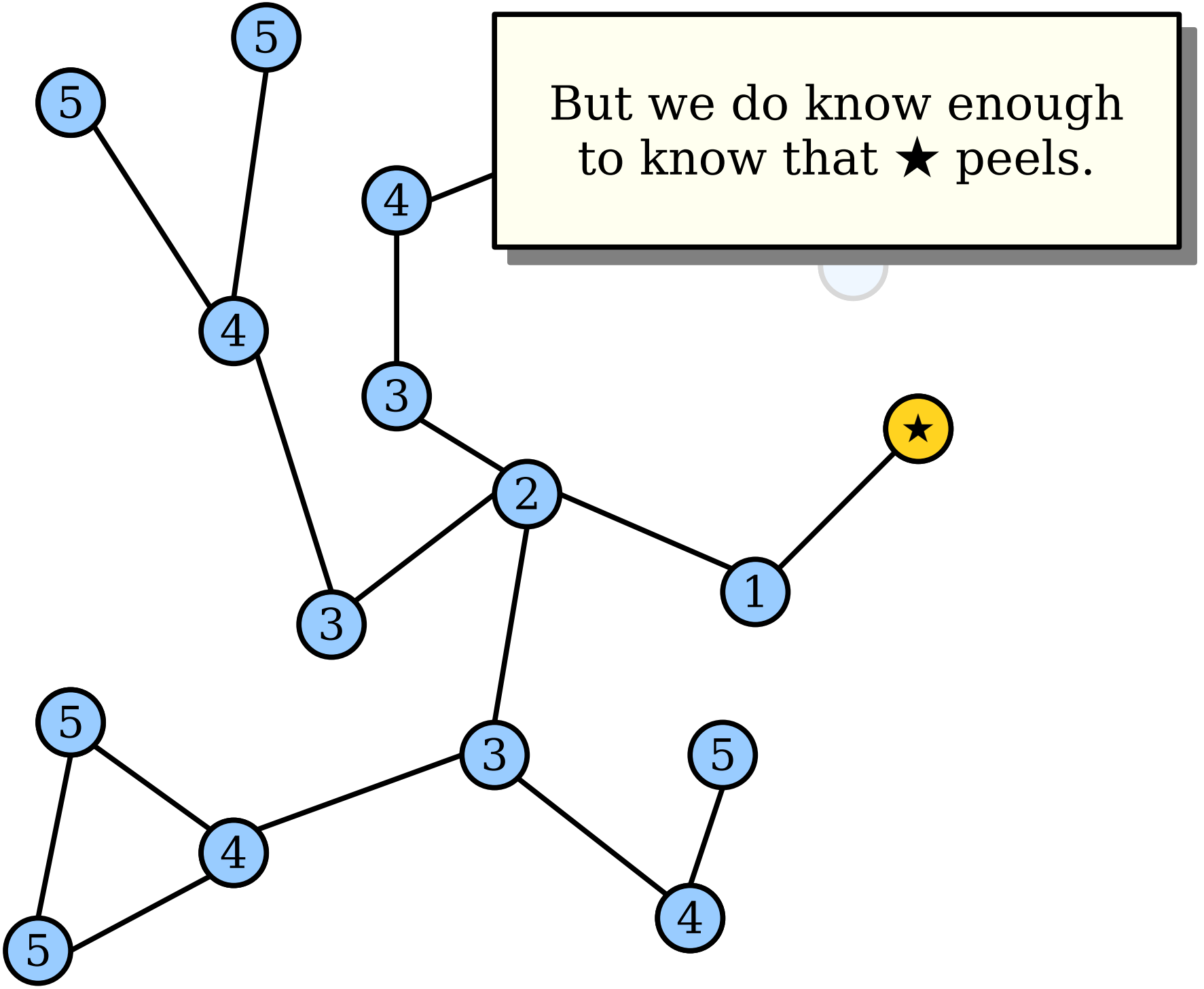

Here’s the neighborhood at distance 5. We can’t tell whether any distance-5 nodes peel.
★

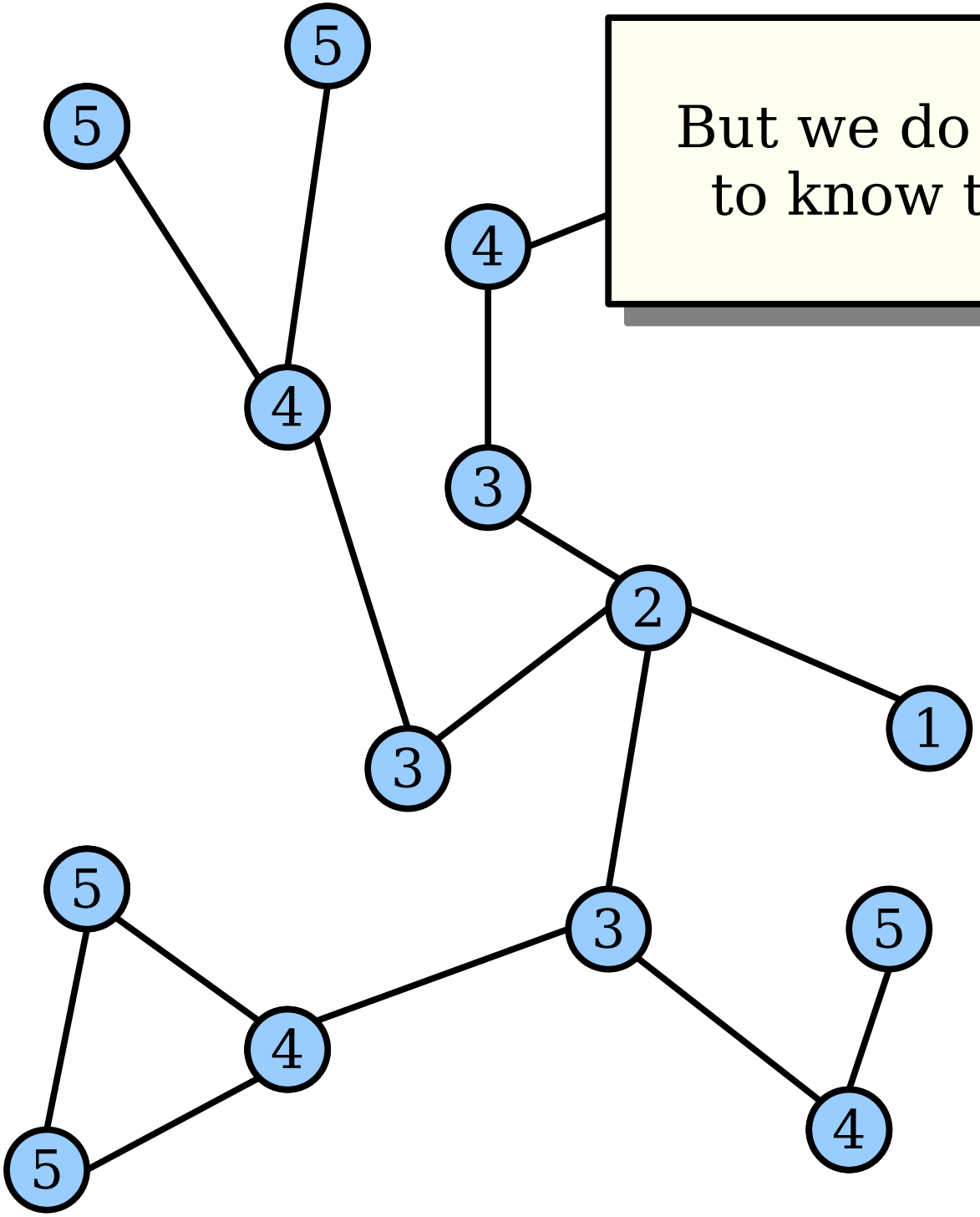

But we do know enough
to know that ★ peels.
5
5
5
4
4
3
3
4
2
3
1
★
1
2
2
1
5
3
5
4
4
2
3
4
5


But we do know enough
to know that ★ peels.
5
5
5
4
4
3
3
3
2
1
★
1
2
2
1
3
5
3
5
4
4
2
3


But we do know enough
to know that ★ peels.
5
5
5
4
4
3
3
2
2
1
★
1
2
1
2
3
5
4
5
4


But we do know enough
to know that ★ peels.
5
5
5
4
4
4
3
3
3
2
1
1
1
★
5
5
4
3
4


But we do know enough
to know that ★ peels.
5
5
5
4
4
3
★
2
3
1
5
3
5
4
4
5



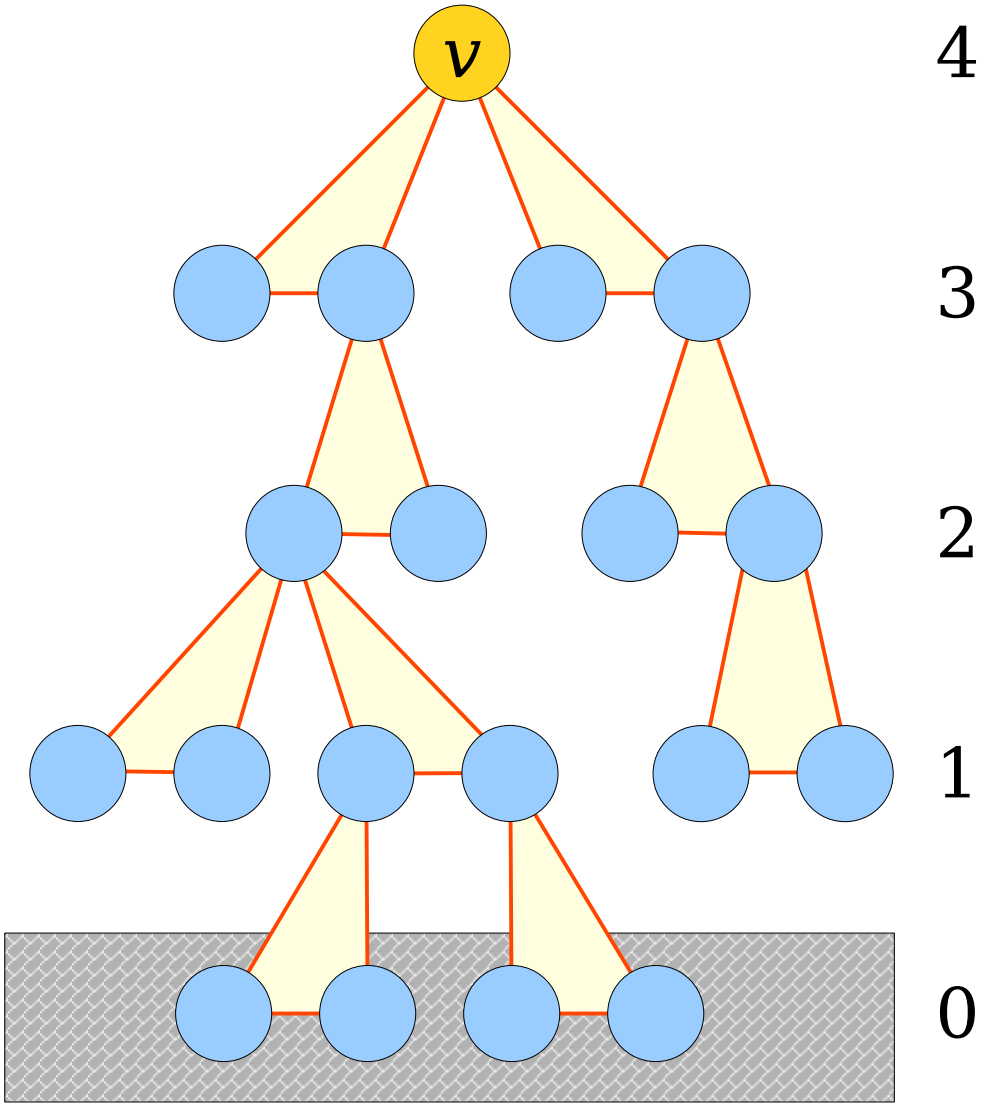
But we do know enough to know that ★ peels.

Summarizing our Observations

- When looking at the neighborhood at distance *r* from the start node:
  - We cannot tell if nodes at distance exactly *r* peel because we can’t tell what their degree is in the original graph.
  - We *can* start to see if nodes at distance less than *r* peel, because we have full knowledge of their degrees and neighbors.
- What can we do with this?

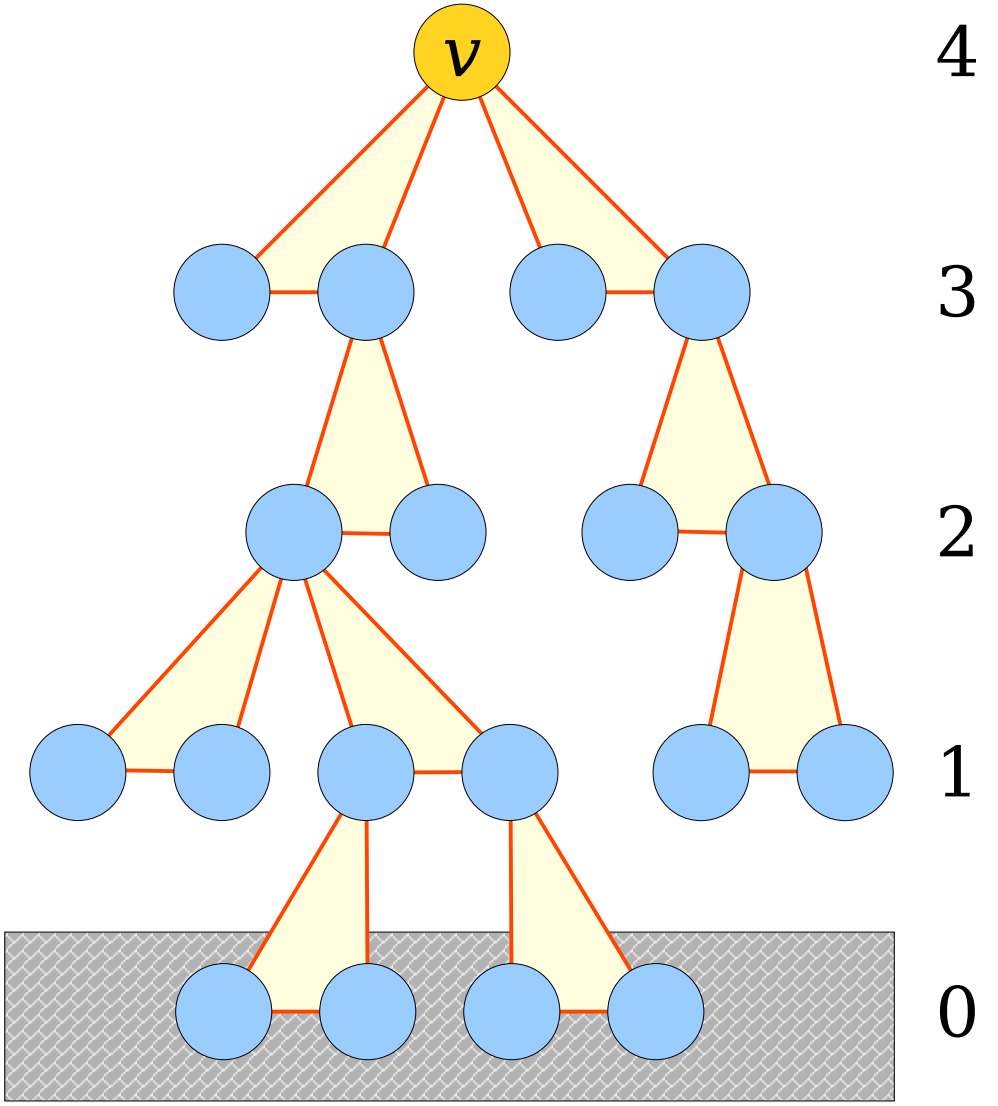
r-Peeling

- Begin with some node *v*. Look at the neighborhood of *v* at distance *r*.
  - As before, we never peel nodes at distance *r*.
  - All other nodes are fair game for peeling.
- Number the layers in the tree based on their distance to the bottom.



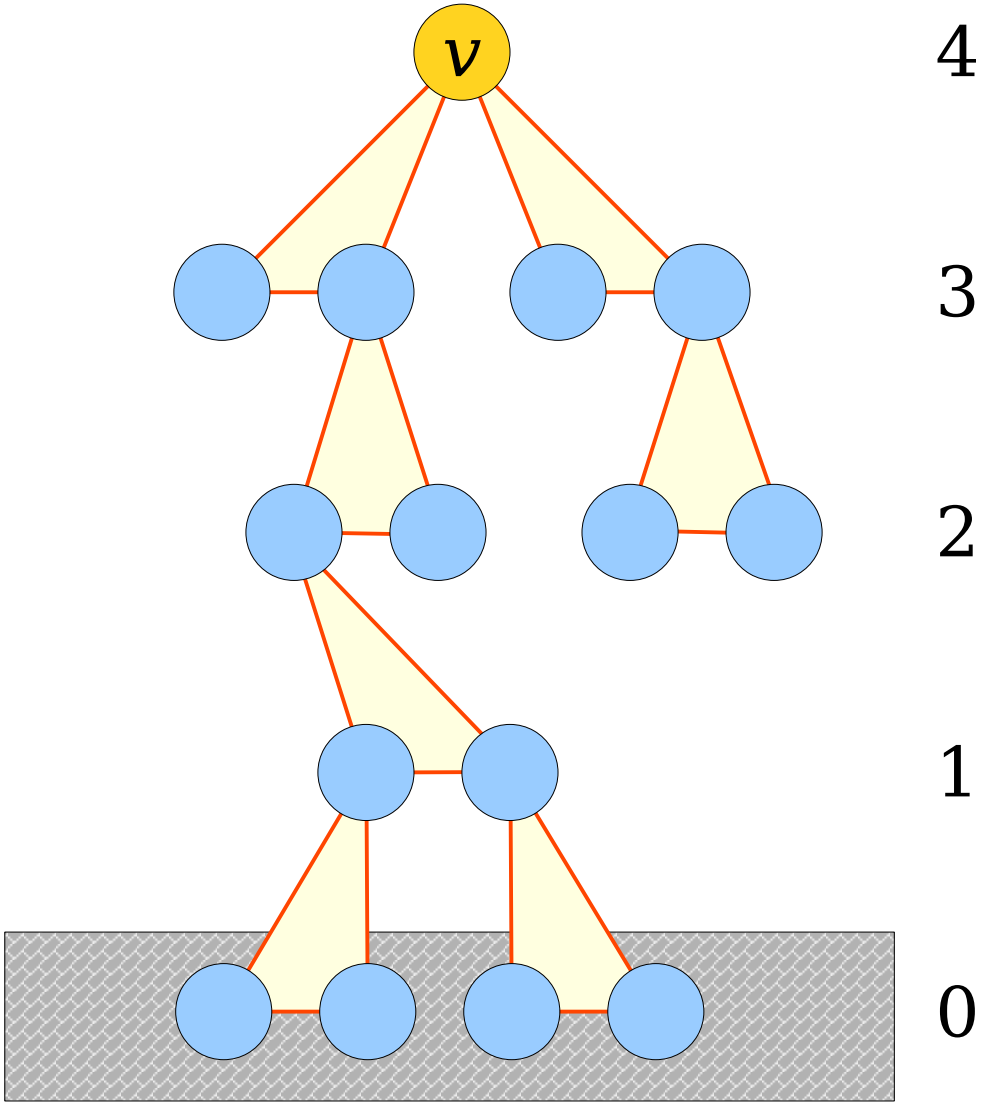
r-Peeling

- Nodes at level 0 never peel.
- Peel every node in level 1 with no children.



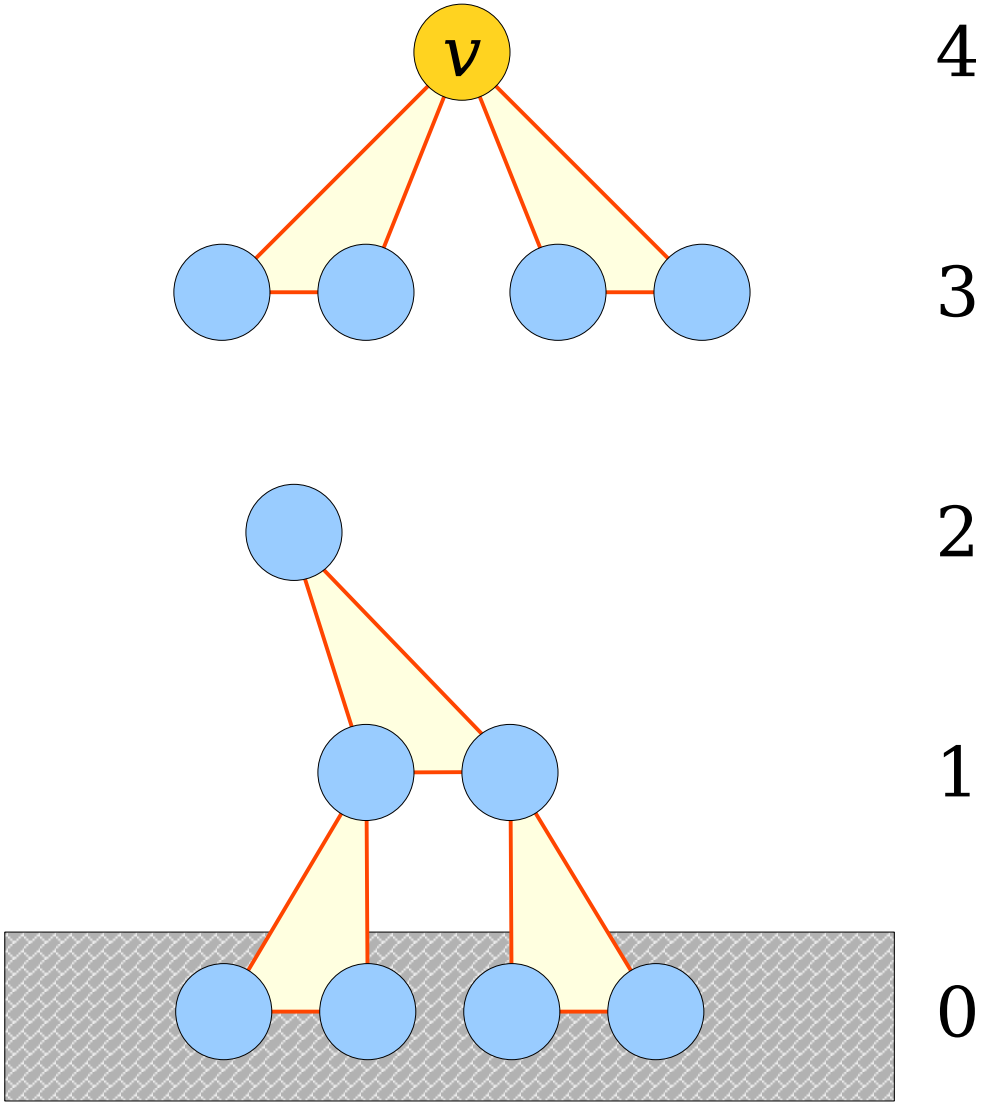
r-Peeling

- Nodes at level 0 never peel.
- Peel every node in level 1 with no children.
- Peel every node in level 2 with no children.



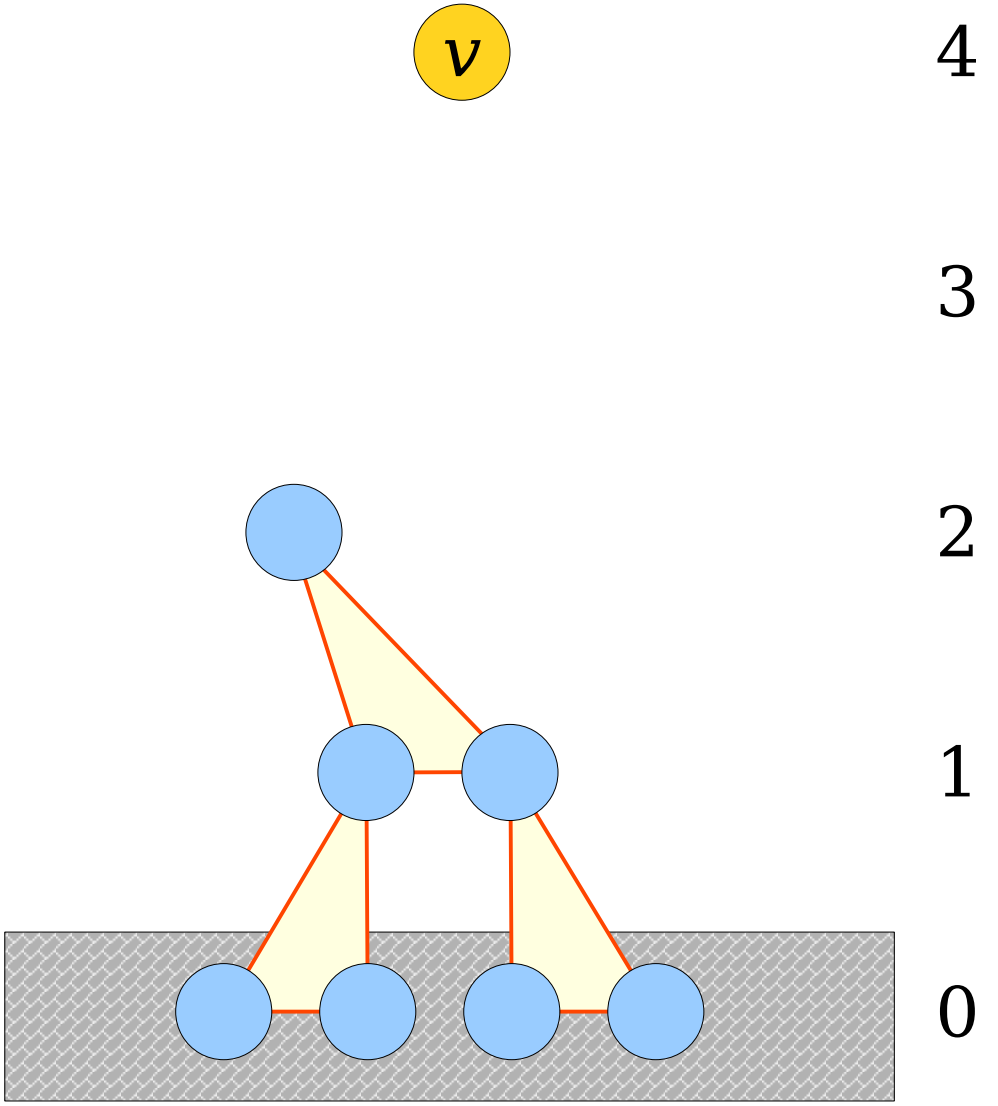
r-Peeling

- Nodes at level 0 never peel.
- Peel every node in level 1 with no children.
- Peel every node in level 2 with no children.
- Repeat all the way up the tree.



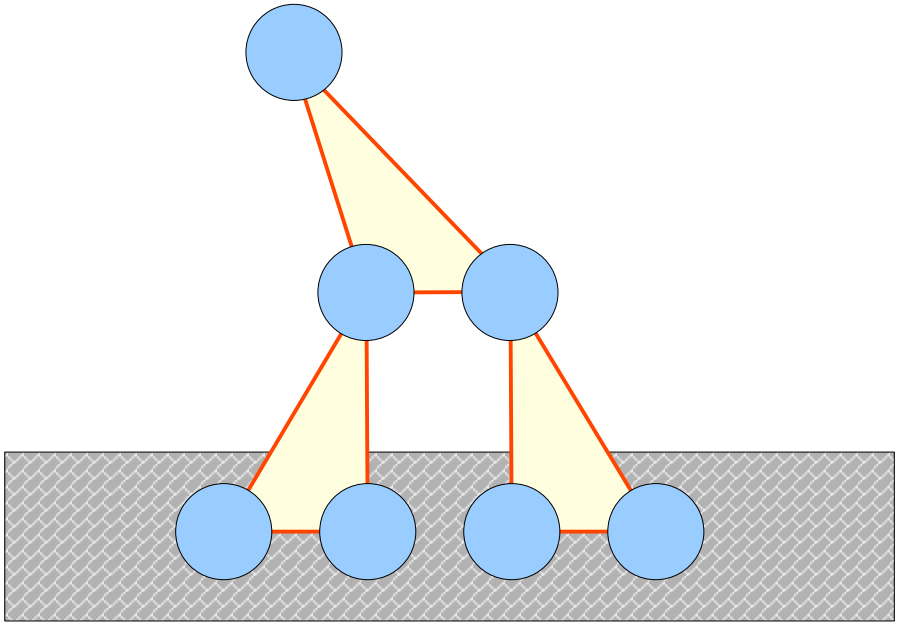
r-Peeling

- Nodes at level 0 never peel.
- Peel every node in level 1 with no children.
- Peel every node in level 2 with no children.
- Repeat all the way up the tree.



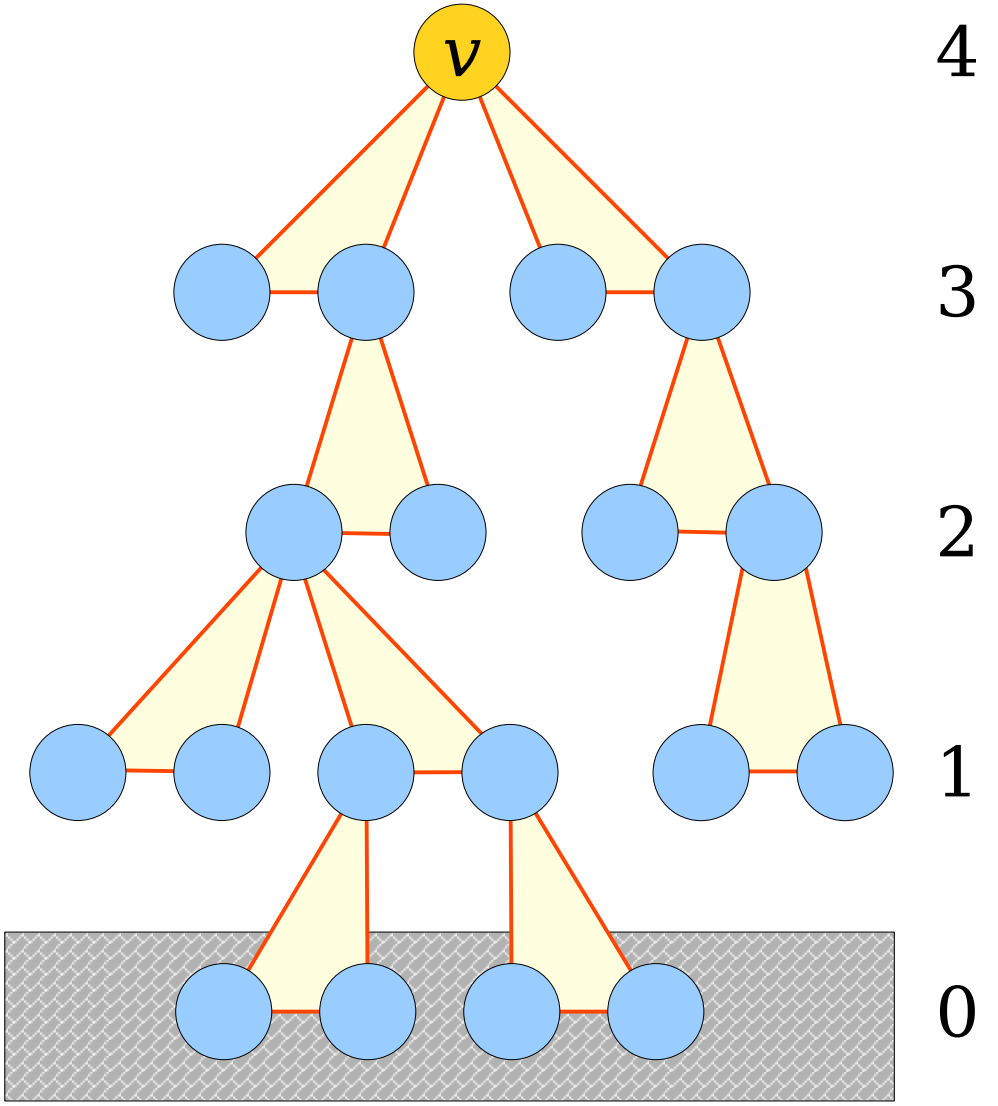
r-Peeling

- Nodes at level 0 never peel.
- Peel every node in level 1 with no children.
- Peel every node in level 2 with no children.
- Repeat all the way up the tree.



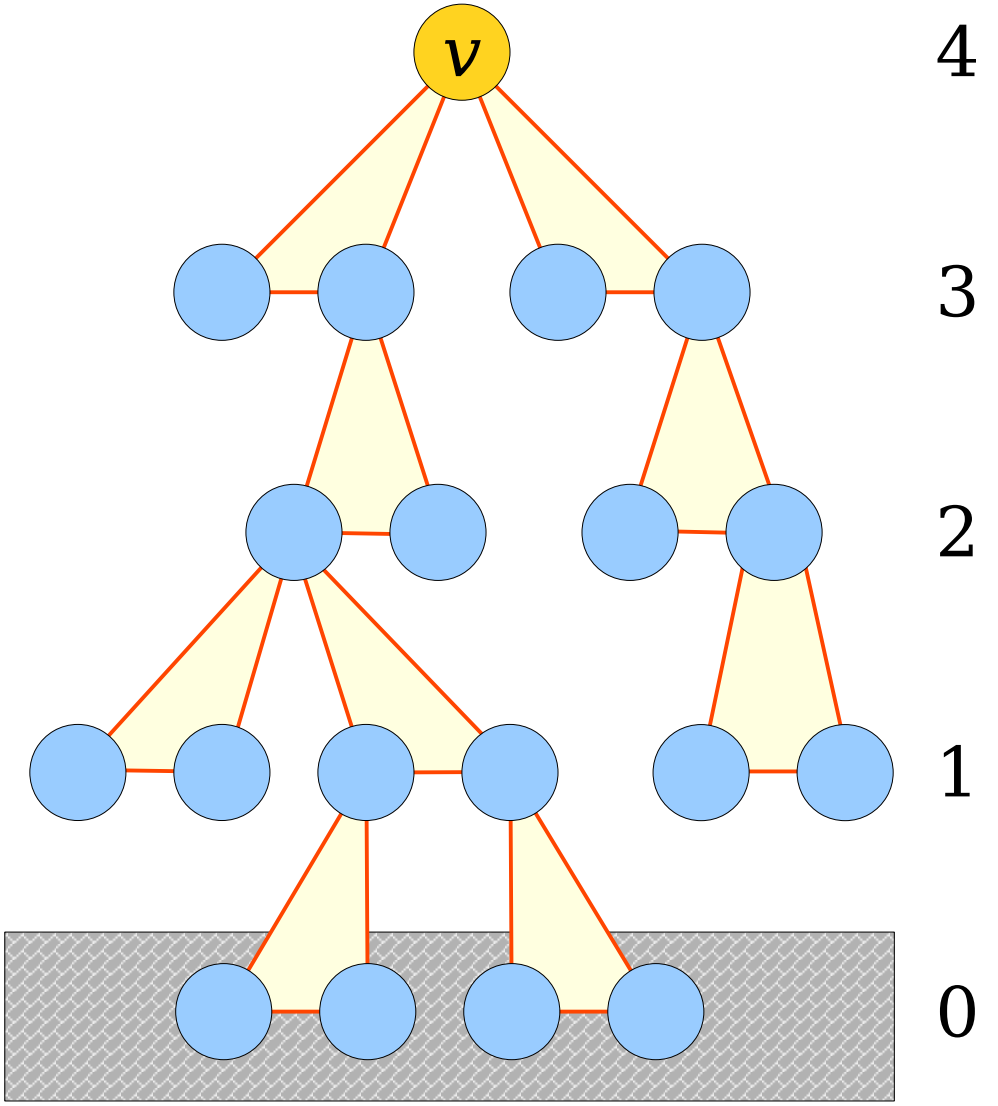
r-Peeling

- Nodes at level 0 never peel.
- Peel every node in level 1 with no children.
- Peel every node in level 2 with no children.
- Repeat all the way up the tree.



r-Peeling

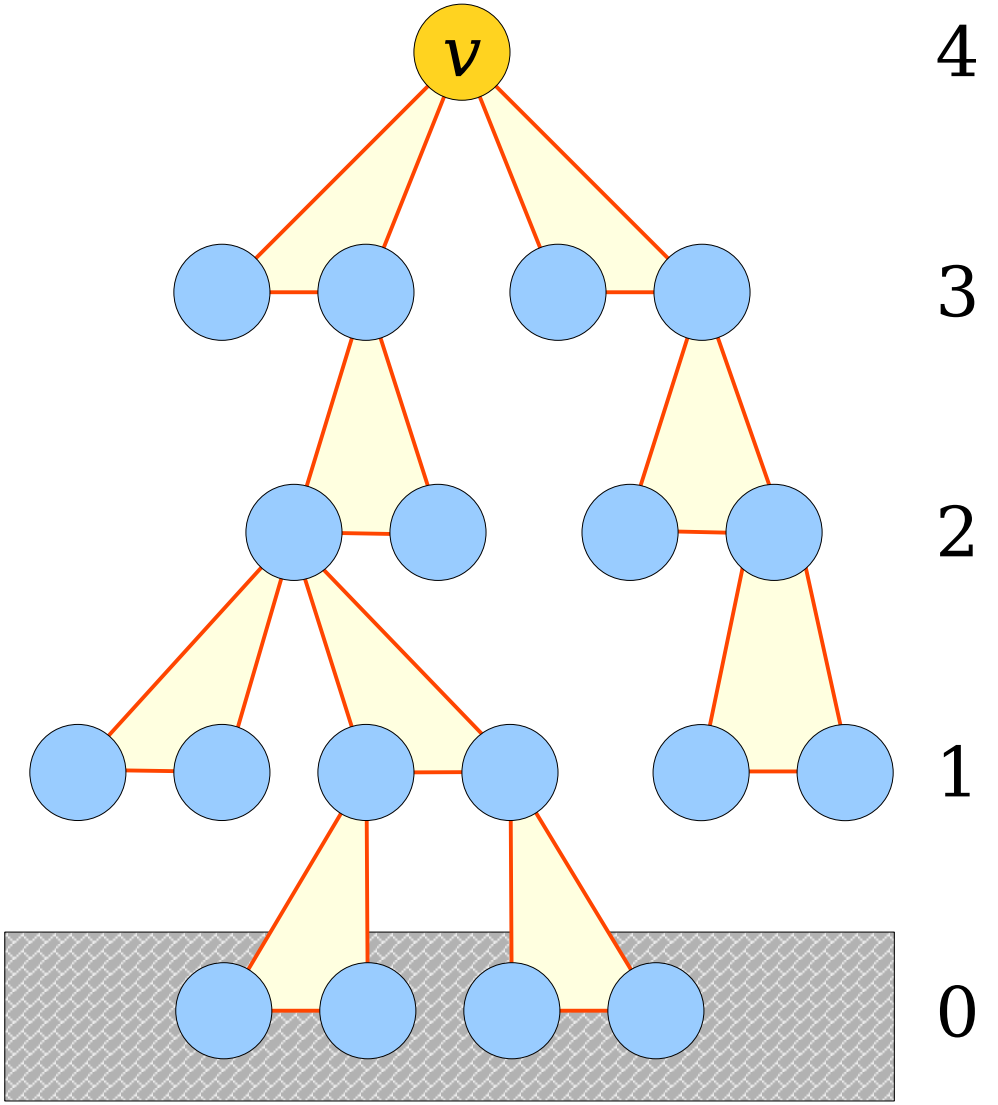
- ***Key Claim:*** If v is peelable in the original graph, then v will peel with this modified algorithm if there are enough layers. *(Why?)*
- ***Goal:*** Analyze this modified peeling process; it's much easier to reason about.



Part 5: Analyze peeling to get a recurrence.

r-Peeling

- Let p_k denote the probability that any individual node in layer k will be peeled.
- ***Goal:*** Determine bounds on p_k, and use that to determine the probability that v gets peeled.

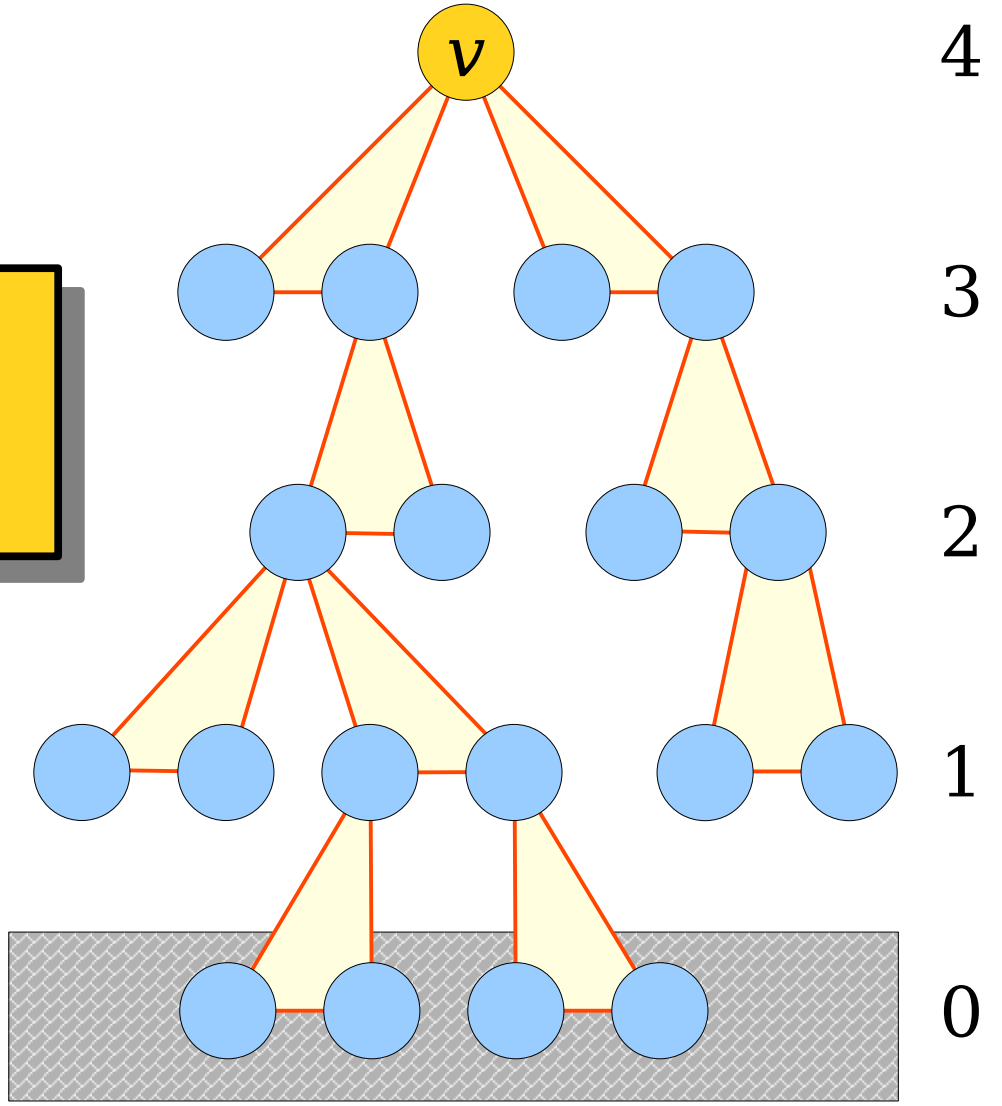


r-Peeling

- What is p_0?

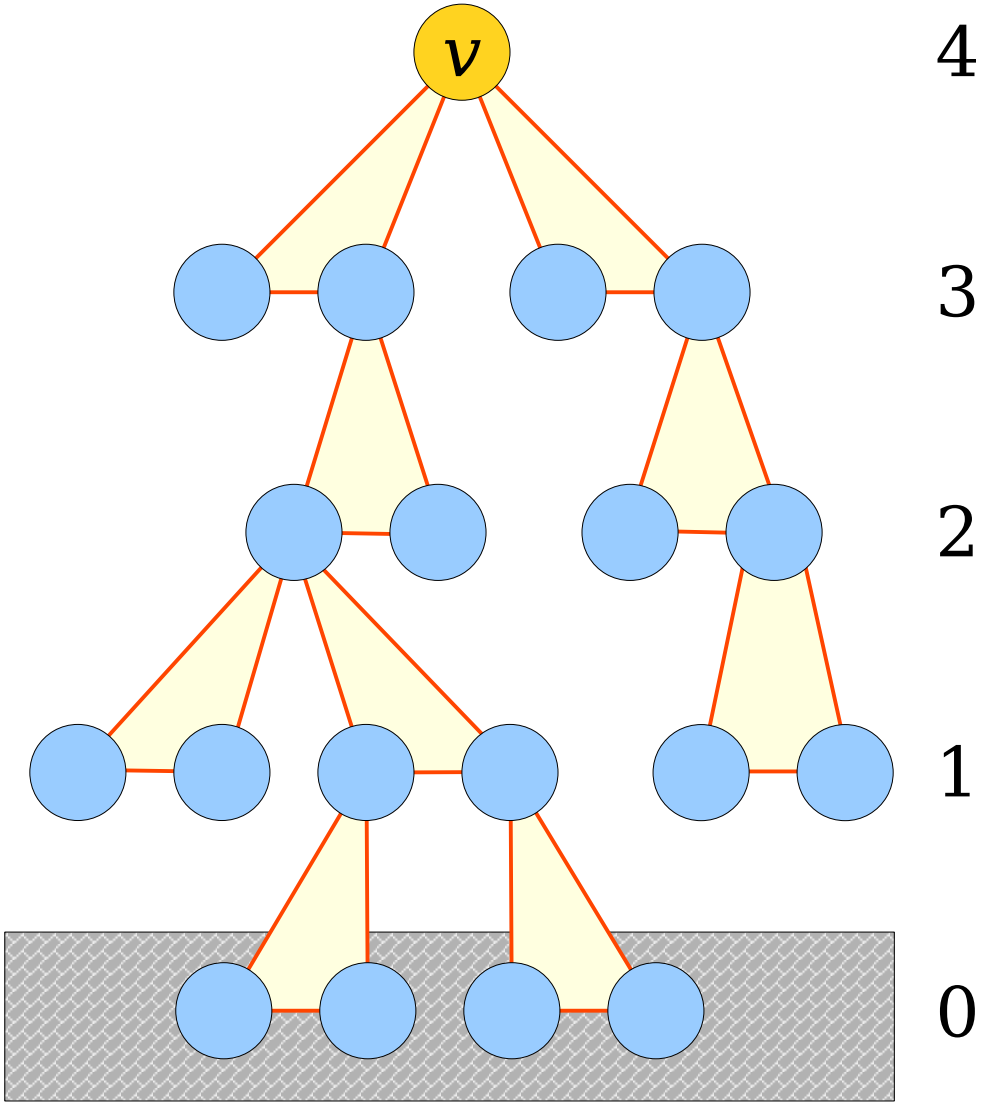
Answer at

https://cs166.stanford.edu/pollev



r-Peeling

- What is p_0?
- By definition, no nodes in layer 0 can be peeled.
  - *(Why?)*
- Thus $p_0 = 0$.

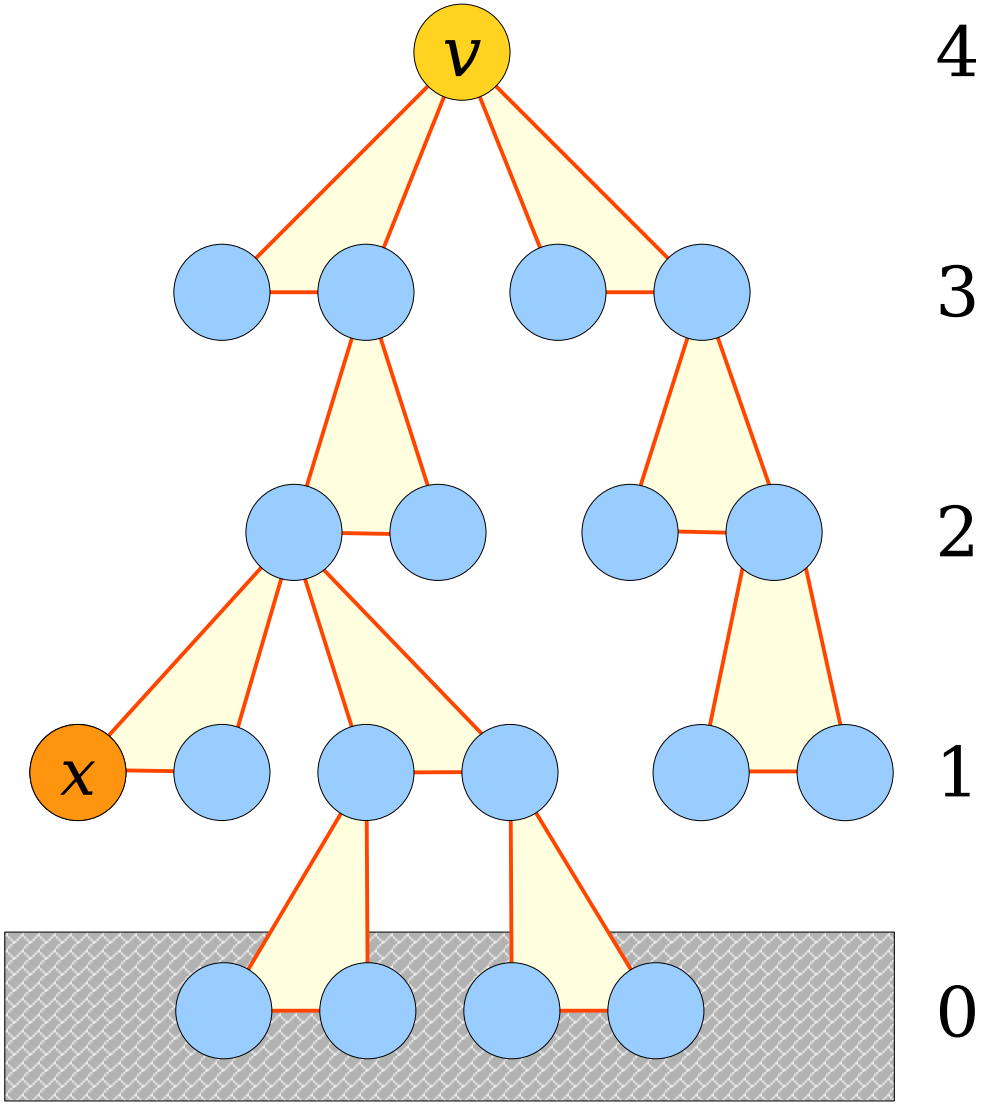


r-Peeling

- To determine p_1, pick a node x in layer 1. What's the probability it gets peeled?
- x is peeled iff it has degree 1.
- x has degree *at least* 1 because of its parent in the tree.
- Thus x is peeled iff x has no children.
- x's child count is a Poisson($d\alpha$) variable.

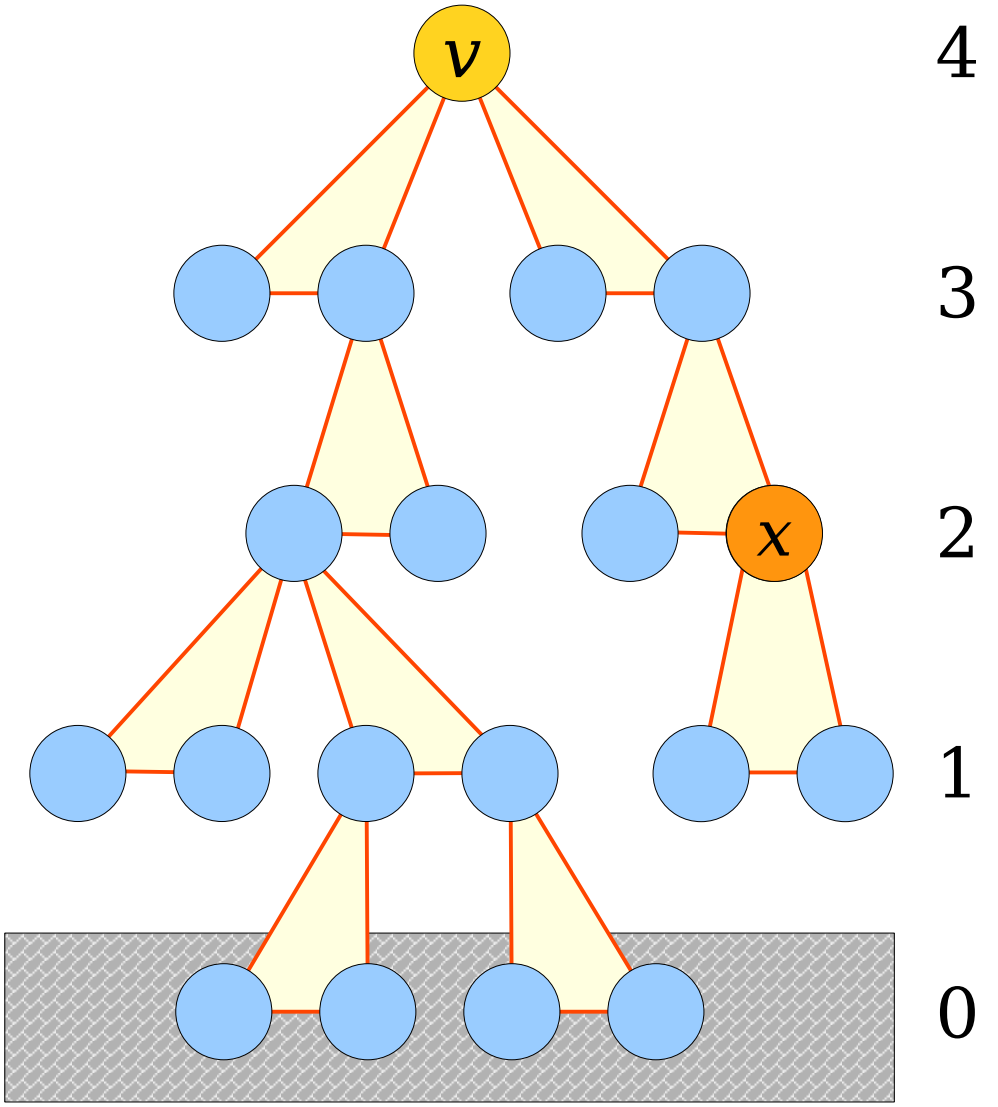
$$p_1 = \Pr[\text{Poisson}(d\alpha) = 0]$$

$$= \boldsymbol{\exp(-d\alpha)}$$



r-Peeling

- To determine p_2, pick a node x in layer 2. What's the probability it gets peeled?
- This is more complex than the previous case.
- ***Observation:*** x will be peeled, but it has children.
- ***Claim:*** x will be peeled iff it has no *unpeeled* child edges.

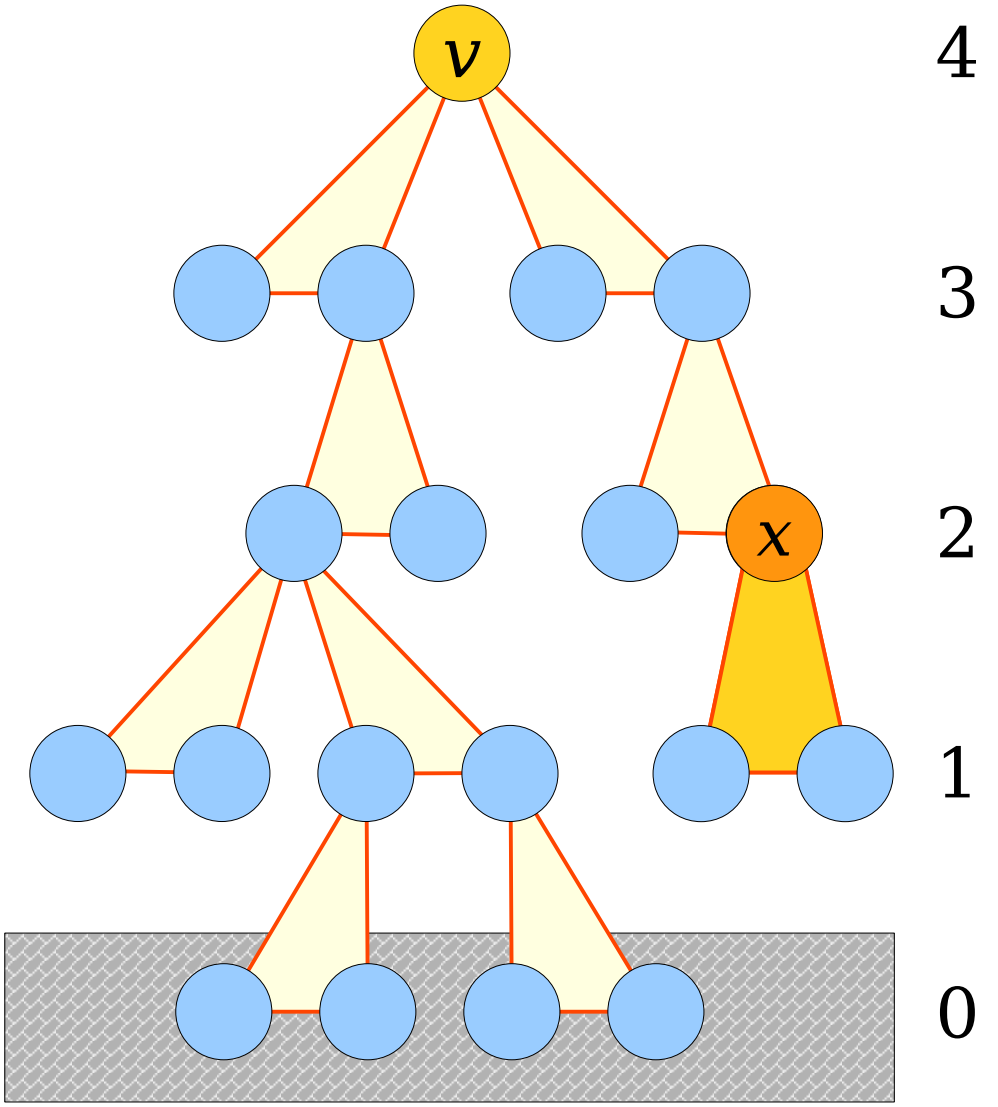


r-Peeling

- Focus on a child edge.
- It's peeled *unless* all of its child nodes are unpeeled.

What is the probability that this edge is unpeeled? Answer at

https://cs166.stanford.edu/pollev

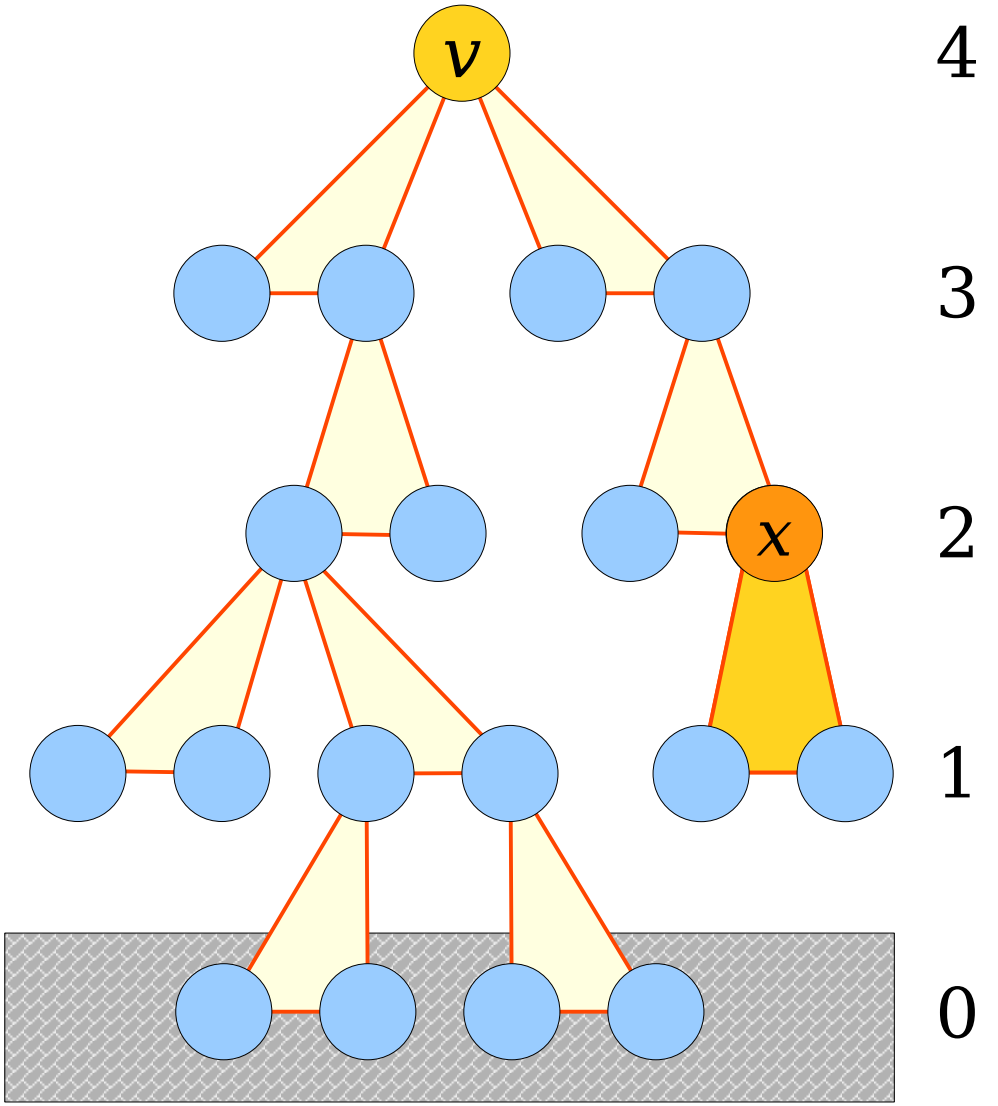


r-Peeling

- Focus on a child edge.
- It's peeled *unless* all of its child nodes are unpeeled.
- This happens with probability $\mathbf{(1 - p_1)^{d-1}}$.
  - There are $d - 1$ children.
  - Each has probability $1 - p_1$ of being unpeeled.
- x's unpeeled child edge count is a **Poisson($d\alpha \cdot (1 - p_1)^{d-1}$)** variable.

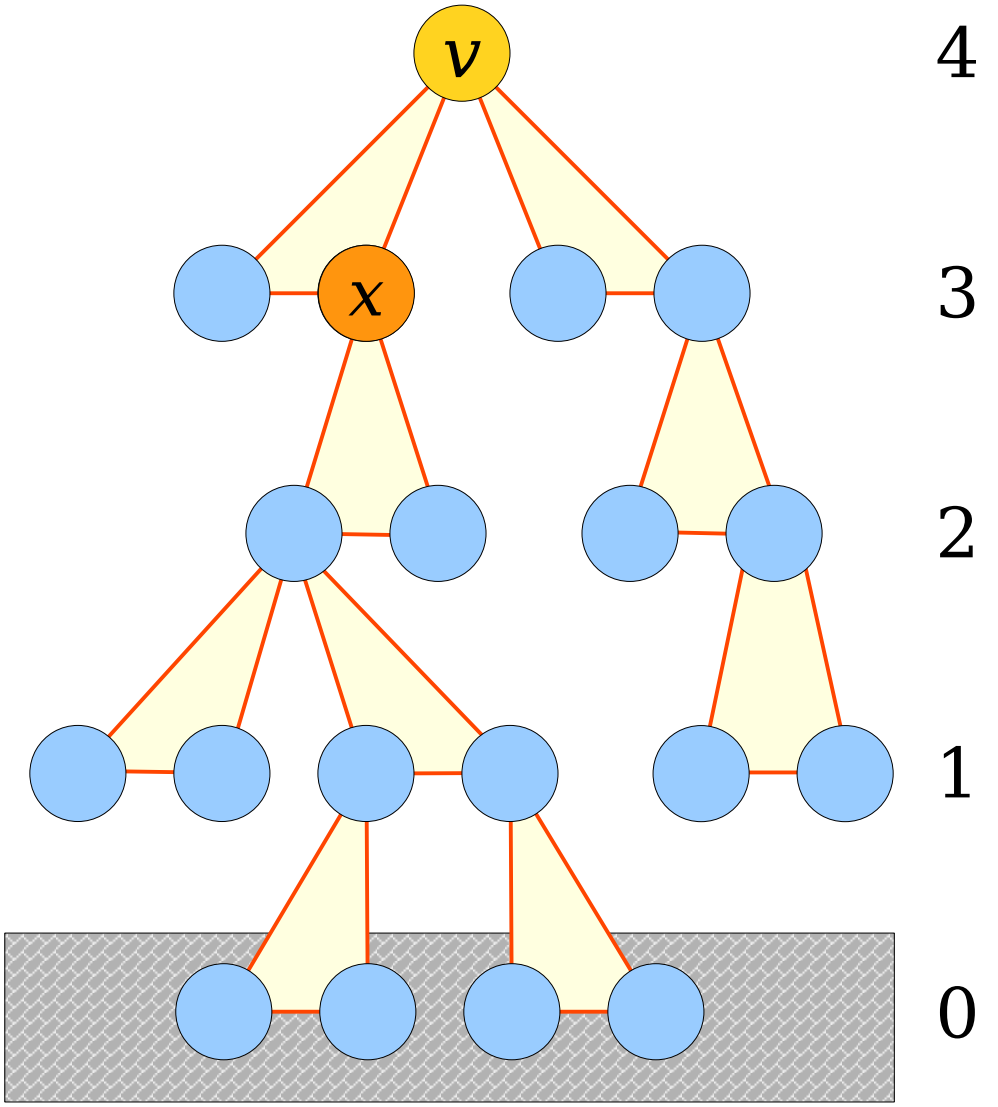
$p_2 = \Pr[\text{Poisson}(d\alpha \cdot (1 - p_1)^{d-1}) = 0]$

$= \mathbf{exp(-d\alpha \cdot (1 - p_1)^{d-1})}$



r-Peeling

- To determine p_3, pick a node x in layer 3. What's the probability it gets peeled?
- ***Claim:*** x will be peeled iff it has no *unpeeled* child edges.

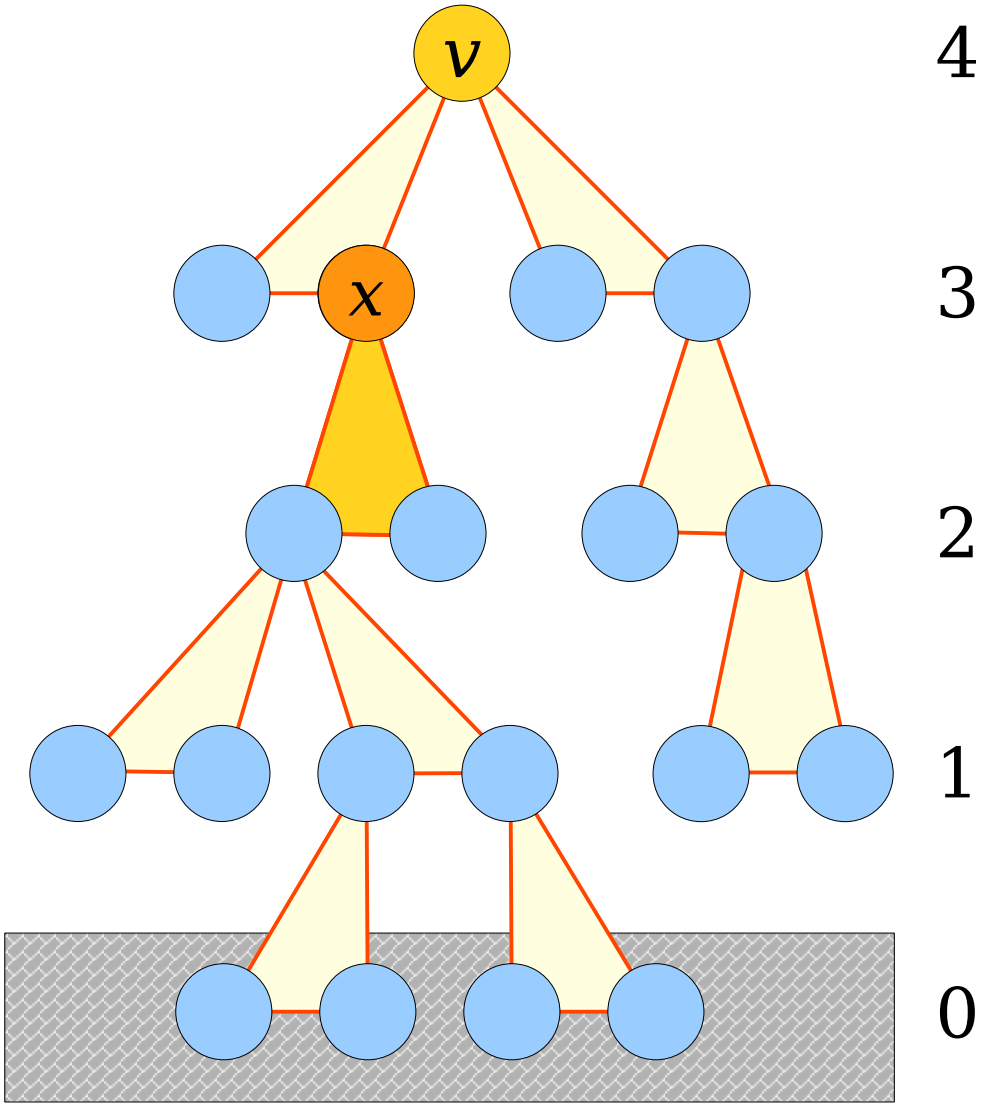


r-Peeling

- Focus on one child edge.
- It's peeled *unless* all of its child nodes are unpeeled.
- This happens with probability $(1 - p_2)^{d-1}$.
  - There are $d - 1$ children.
  - Each has probability $1 - p_2$ of being unpeeled.
- x's unpeeled child edge count is a **Poisson($d\alpha \cdot (1 - p_2)^{d-1}$)** variable.

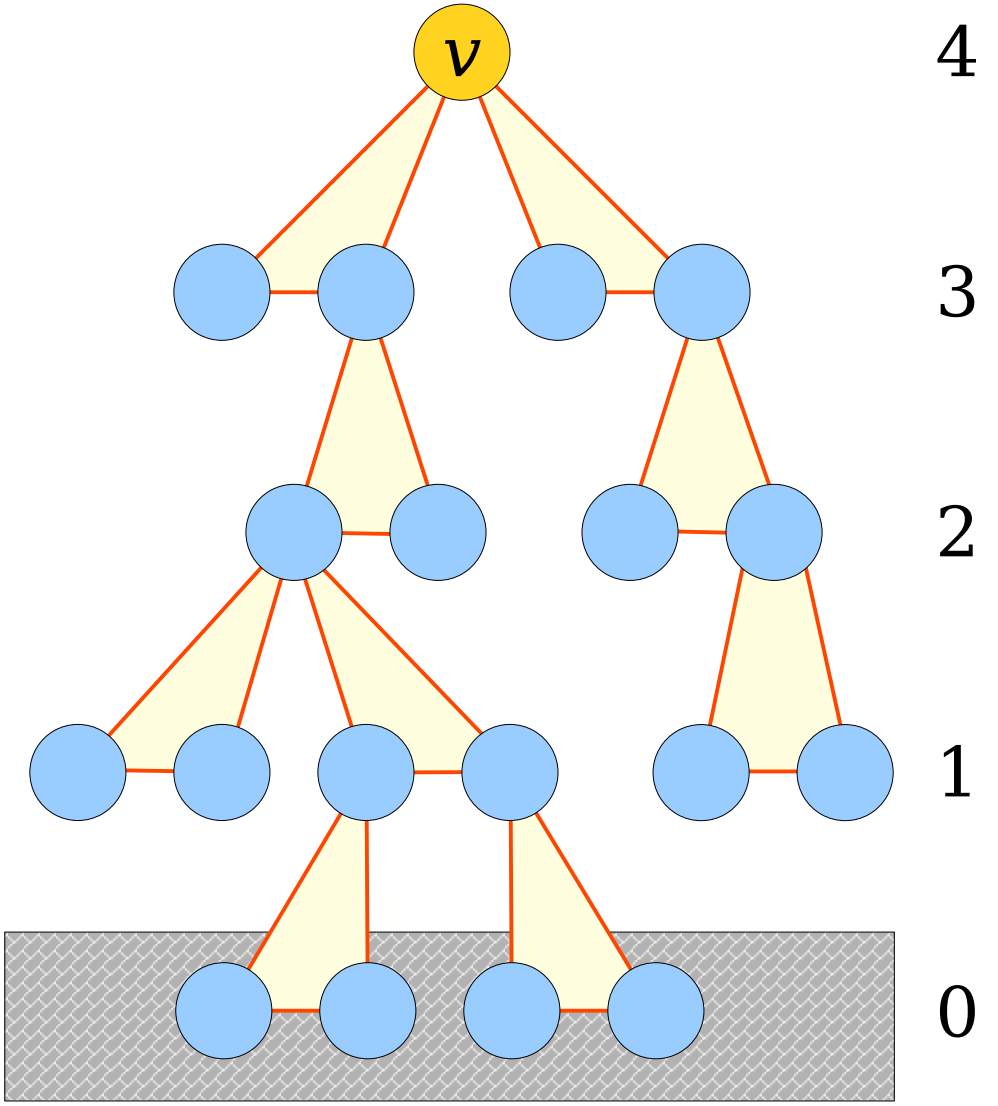
$p_3 = \Pr[\text{Poisson}(d\alpha \cdot (1 - p_2)^{d-1}) = 0]$

$= \boldsymbol{exp(-d\alpha \cdot (1 - p_2)^{d-1})}$



r-Peeling

- To determine p_4, look at v in level 4. What's the probability it gets peeled?
- ***Claim:*** v will be peeled iff it has *at most one* unpeeled child edge.



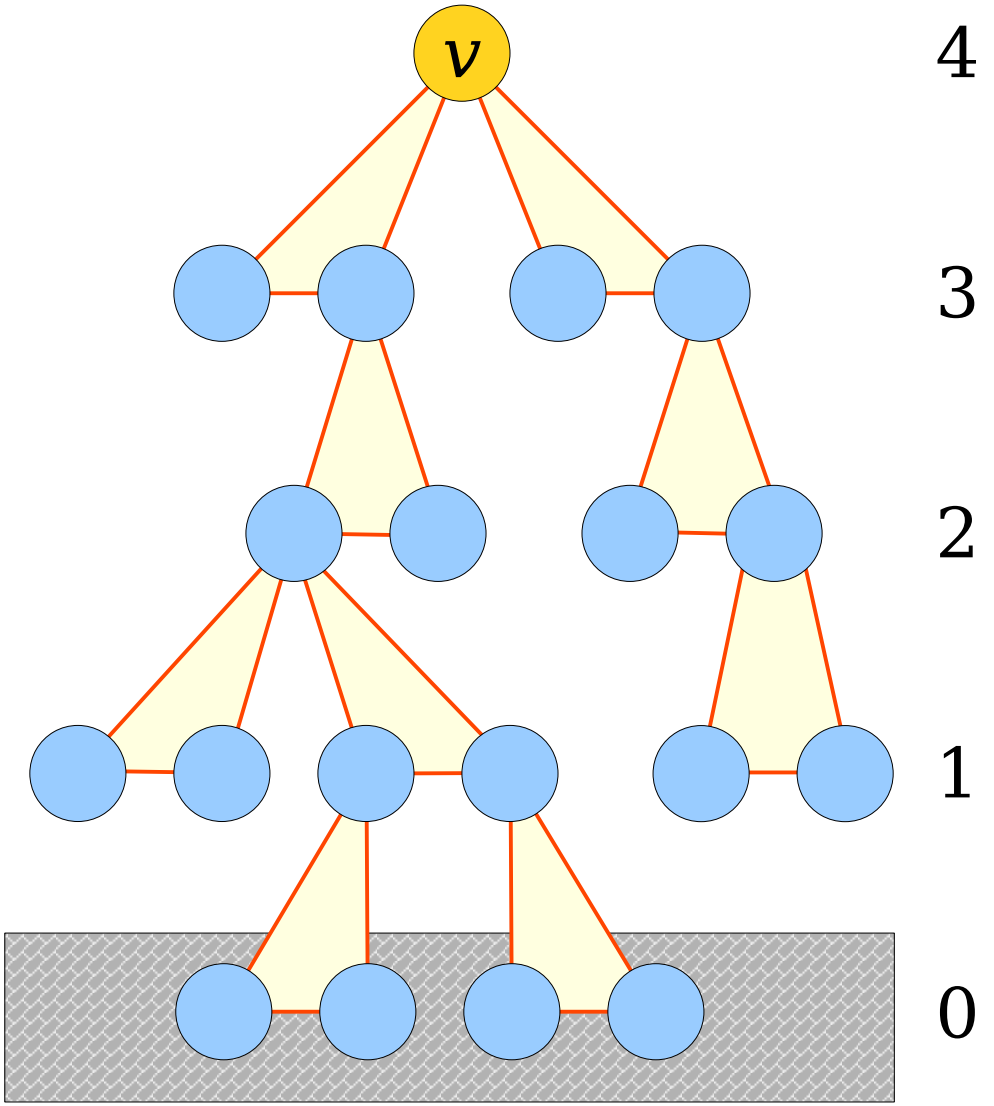
r-Peeling

- x's unpeeled child edge count is a **Poisson($d\alpha \cdot (1 - p_3)^{d-1}$)** variable.
  - Same reasoning as before.
- Thus we have

$$p_4 = \Pr[\text{Poisson}(d\alpha \cdot (1 - p_3)^{d-1}) \leq 1]$$

$$\approx \Pr[\text{Poisson}(d\alpha \cdot (1 - p_3)^{d-1}) = 0]$$

$$= \boldsymbol{exp(-d\alpha \cdot (1 - p_3)^{d-1})}$$



To Summarize

$$p_0 = 0$$

$$p_1 = exp(-\alpha d)$$

$$p_2 = exp(-\alpha d \cdot (1 - p_1)^{d-1})$$

$$p_3 = exp(-\alpha d \cdot (1 - p_2)^{d-1})$$

$$p_4 = exp(-\alpha d \cdot (1 - p_3)^{d-1})$$

...

To Summarize

$$p_0 = 0$$

$$p_1 = exp(-\alpha d \cdot \mathbf{(1 - p_0)^{d-1}})$$

$$p_2 = exp(-\alpha d \cdot (1 - p_1)^{d-1})$$

$$p_3 = exp(-\alpha d \cdot (1 - p_2)^{d-1})$$

$$p_4 = exp(-\alpha d \cdot (1 - p_3)^{d-1})$$

…

To Summarize

$$p_0 = 0$$

$$p_1 = exp(-\alpha d \cdot (1 - p_0)^{d-1})$$

$$p_2 = exp(-\alpha d \cdot (1 - p_1)^{d-1})$$

$$p_3 = exp(-\alpha d \cdot (1 - p_2)^{d-1})$$

$$p_4 = exp(-\alpha d \cdot (1 - p_3)^{d-1})$$

…

$$\boldsymbol{p_0 = 0} \qquad \boldsymbol{p_{k+1} = exp(-\alpha d \cdot (1 - p_k)^{d-1})}$$

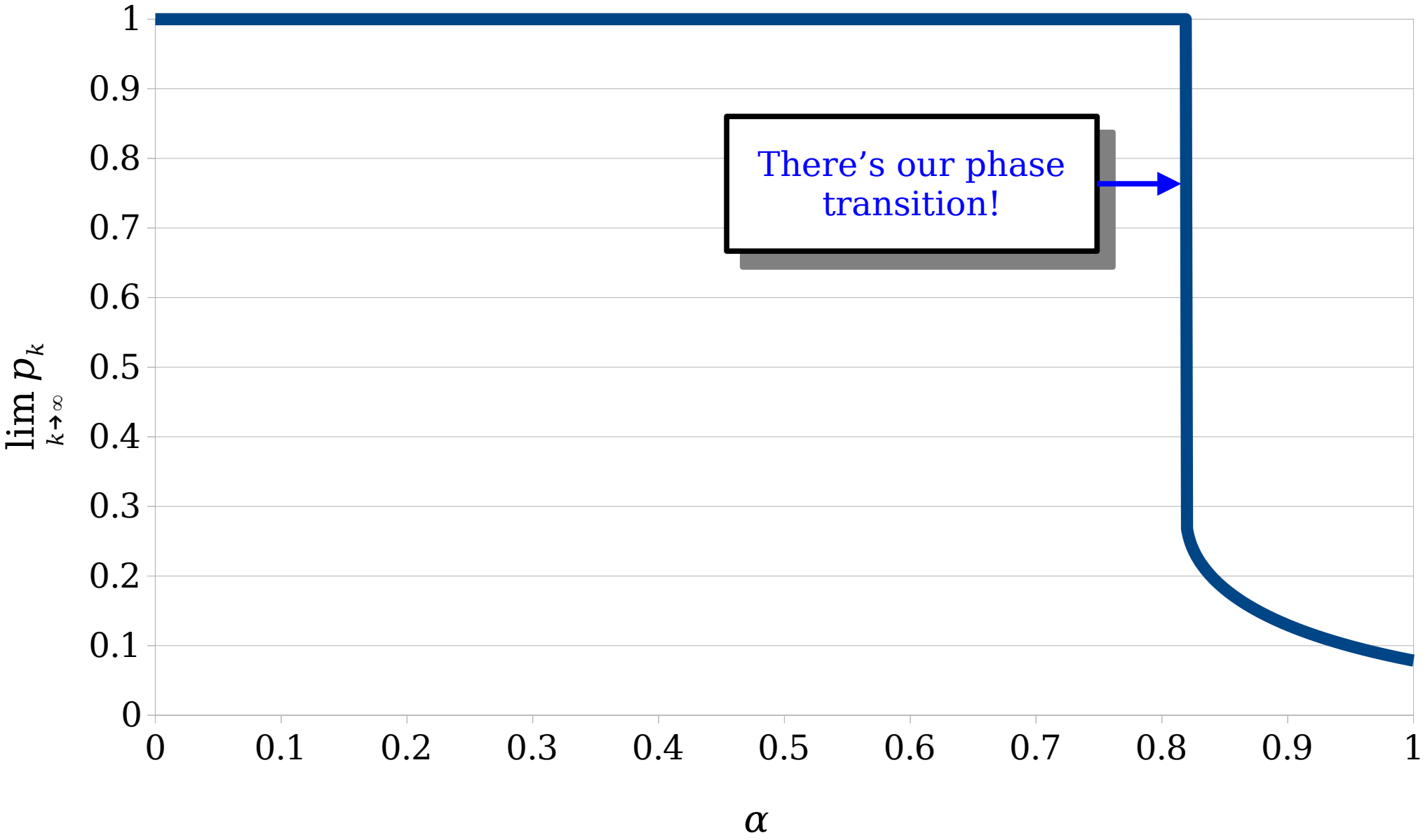
Part 6: Solve the Recurrence

The Recurrence

- The probability that a node at level k in the tree is peeled is given by

$$p_0 = 0 \qquad p_{k+1} = exp(-\alpha d \cdot (1 - p_k)^{d-1})$$

- If we want to know the probability that a given node will peel, we see if it peels when considering larger and larger distances.
- What is the limiting behavior of this recurrence?



How this recurrence work in the limit with $d = 3$?

(i.e. What is $\lim_{k\to\infty} p_k$?)

Exploring the Recurrence

- Using a binary search, we can find the limiting values α_d^* where peeling stops working.

$$\alpha_3^* \approx 0.8184689470824...$$

$$\alpha_4^* \approx 0.7722797165477...$$

$$\alpha_5^* \approx 0.7017801791593...$$

$$\alpha_6^* \approx 0.6370810596662...$$

$$\alpha_7^* \approx 0.5817751219116...$$

- ***Bigger Question:*** Why does this recurrence have a phase transition at all?

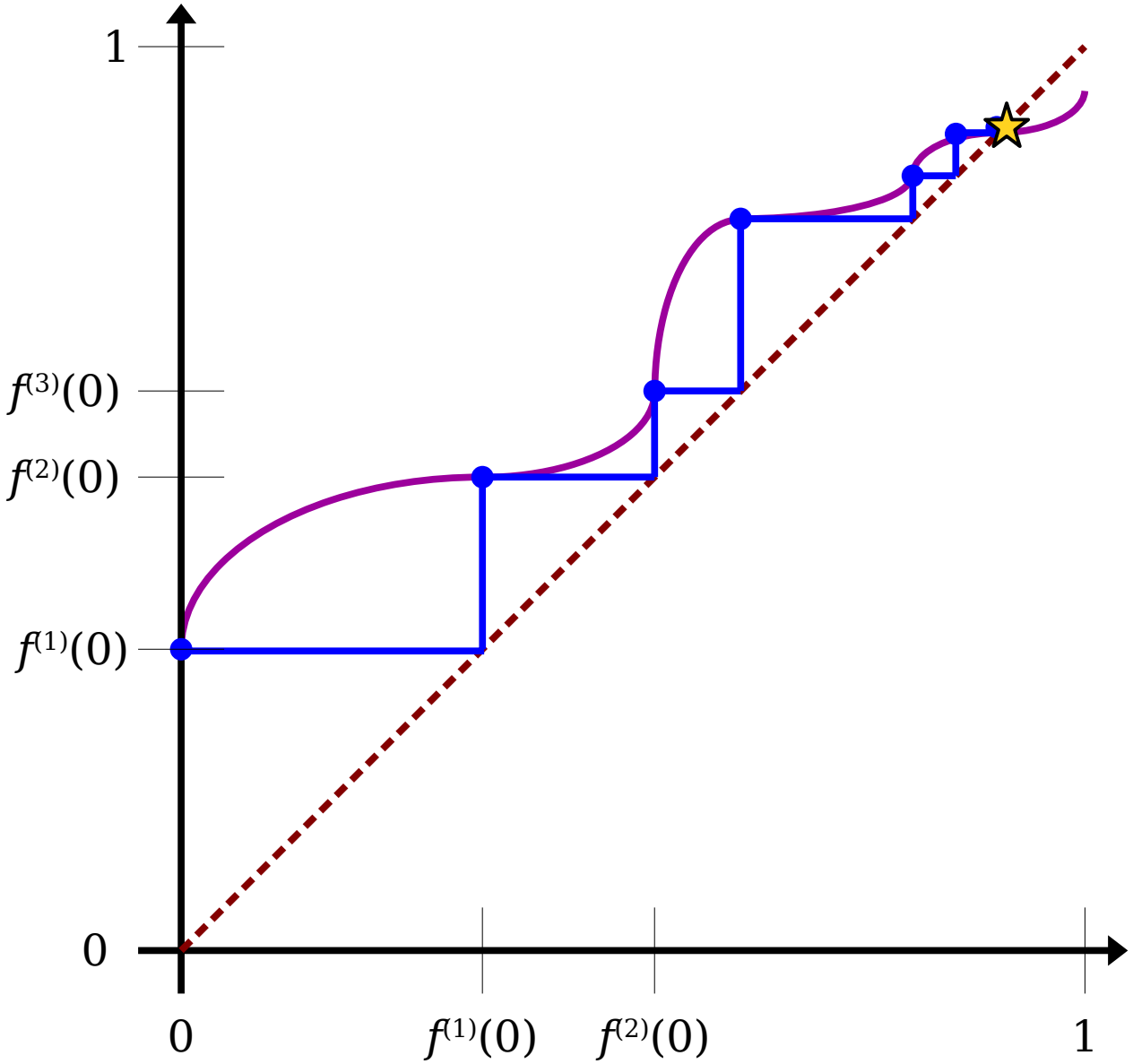
Iterated Functions

- Let $f(p) = exp(-\alpha d(1 - p)^{d-1})$.
- The probability of peeling a node approaches the limit of this sequence:

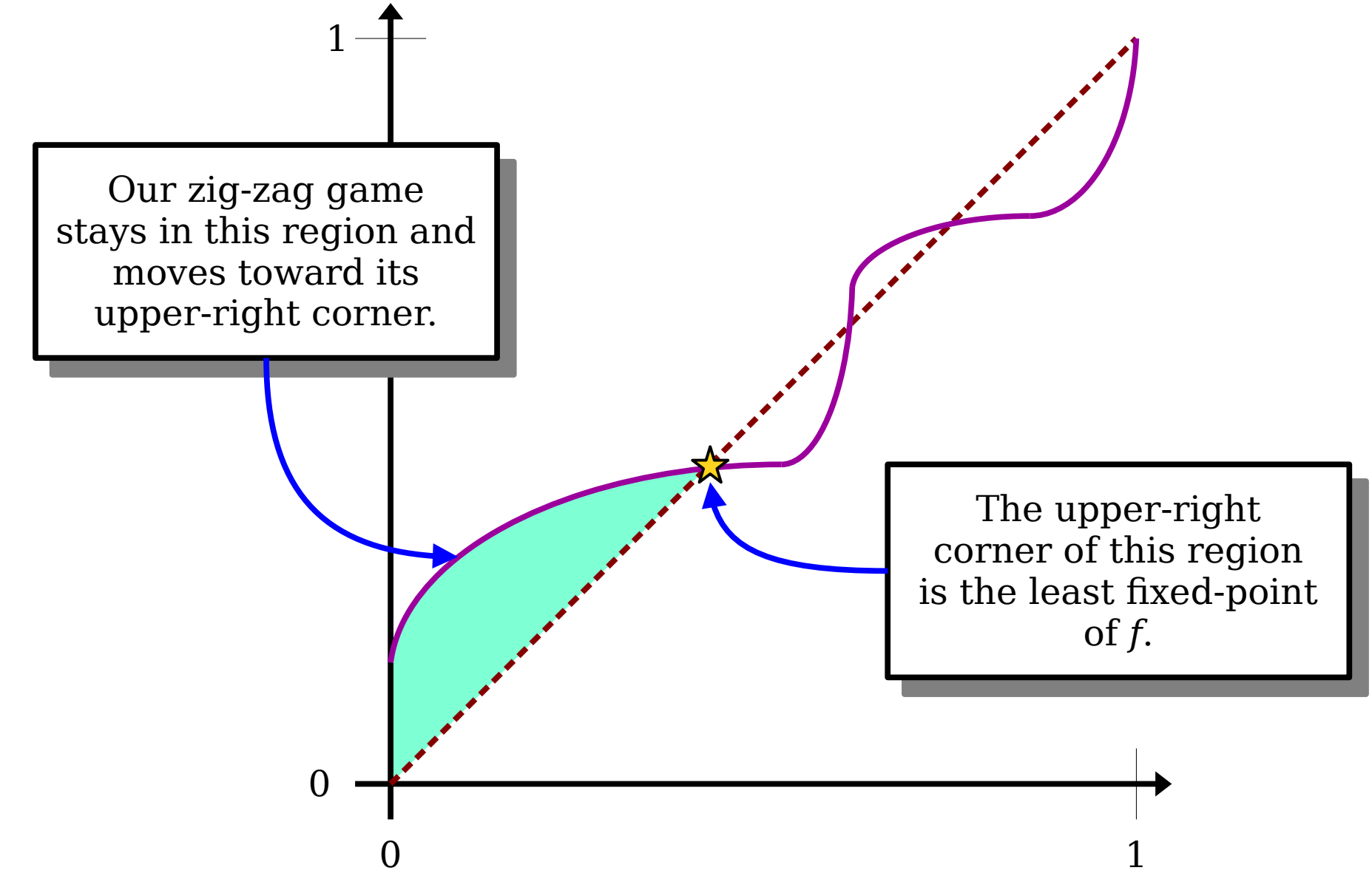
$$f^{(0)}(0), \quad f^{(1)}(0), \quad f^{(2)}(0), \quad f^{(3)}(0), \quad \ldots$$

- What does this converge to?

$$p_0 = 0$$
$$p_{k+1} = \boldsymbol{f(p_k)}$$



Kleene's Fixed-Point Theorem: Let $f : [0, 1] \to [0, 1]$ be continuous and monotone. Then $f^{(0)}(0)$, $f^{(1)}(0)$, $f^{(2)}(0)$, ... converges to the smallest $x \in [0, 1]$ for which $f(x) = x$.



Kleene's Fixed-Point Theorem: Let $f : [0, 1] \to [0, 1]$ be continuous and monotone. Then $f^{(0)}(0)$, $f^{(1)}(0)$, $f^{(2)}(0)$, ... converges to the smallest $x \in [0, 1]$ for which $f(x) = x$.

Something ***very interesting*** happens
if we apply Kleene's Fixed-Point Theorem
to our particular recurrence...

Putting it All Together

- The probability that a node gets peeled increases the more layers we peel back, according to the transform

$$p \mapsto exp(-\alpha d(1 - p)^{d-1}).$$

- This converges to the least fixed point of this map.
- This curve always has a fixed point of $p = 1$, but for each d there's an α_d^* where the curve gains a second fixed point below 1 as it intersects the line $y = x$.
- When $\alpha < \alpha_d^*$, we converge to the fixed point of 1 and peeling almost always succeeds.
- When $\alpha > \alpha_d^*$, we converge to the lower fixed point and peeling has only a constant (non-1) probability of success.

More to Explore

- Recent (~2020) work shows that if you don't choose uniformly-random hypergraphs and instead choose "fuse" graphs, you can make adding additional hash functions *increase* the value of α.
  - Check out the ***binary fuse filter*** as an improved XOR filter that does this.
- The same techniques here can be applied to solving random linear systems of equations, to randomized XOR sat instances, to representing perfect hash functions, etc.
- Oh, and one more application…

A Mathematical Foundation for Foundation Paper Pieceable Quilts

MACKENZIE LEAKE, Stanford University, USA
GILBERT BERNSTEIN, UC Berkeley, USA
ABE DAVIS, Cornell University, USA
MANEESH AGRAWALA, Stanford University, USA

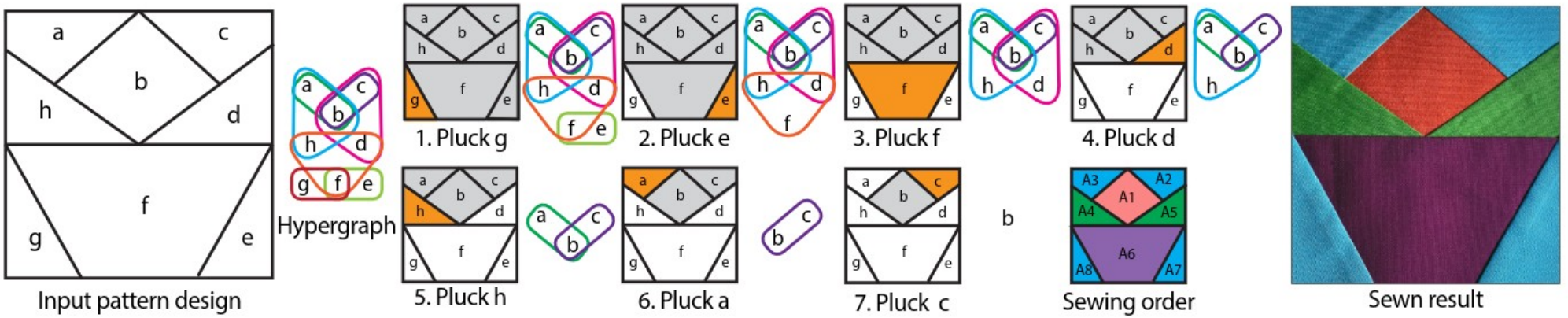


Fig. 1. Given an input pattern design we encode the geometry as a *dual hypergraph*, where nodes represent faces and hyperedges represent *seams* connecting two or more faces. We visualize the hyperedges with colored boundaries (left). In this work we prove that if this hypergraph is acyclic, the pattern design is foundation paper pieceable, and we present a *leaf-plucking* algorithm that iteratively removes leaf hyperedges, where a node is only contained in that hyperedge, to generate a sewing order for the design, which is the reverse of the order in which we plucked the nodes (center). Our quilt design tool shows the resulting sewing order by numbering the faces (center, Sewing order) and lets users color the faces to visualize the design. Quilters can use foundation paper piecing to sew the quilt by attaching fabric pieces one at a time in the sewing order and precisely construct the quilt top (right).

https://doi.org/10.1145/3450626.3459853

https://www.youtube.com/watch?v=g04VgzzRhlQ

Next Time

- ***Disjoint-Set Forests***
  - Making Kruskal's algorithm fast.
- ***Analyzing Disjoint-Set Forests***
  - With a very surprising conclusion…