Lecture 6

More greedy algorithms and minimum spanning trees

Chinmay Nirkhe | CSE 421 Winter 2026

Previously in CSE 421…

Dijkstra's algorithm

- Initialize $d(v) \leftarrow \infty, p(v) \leftarrow \perp$ ("parent" of v is undefined) for all $v \neq s$.
- Set $d(s) \leftarrow 0,\ p(s) \leftarrow \text{root}$
- Create priority queue Q and $\text{insert}(Q, \text{key} = d(v), v)$ for each $v \in V$
- While Q isn't empty, pop minimum key-element u from queue
  - For each neighbor v of u, check if $d(u) + w(u, v) < d(v)$
  - If so, $d(v) \leftarrow d(u) + w(u, v),\ p(v) \leftarrow u$, and $\text{setkey}(Q, \text{key} = d(v), v)$
- Return d, p for distance and parent functions.

once popped off $d(u)$ is fixed forever

update parent of v to be u.

Today

Minimizing lateness

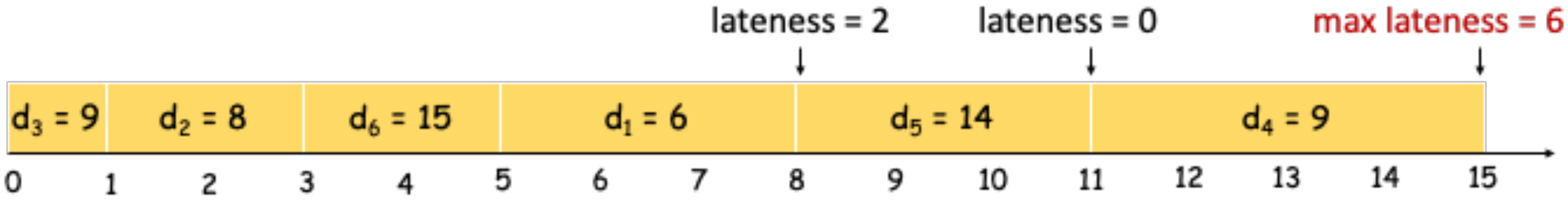
- A new scheduling problem. There is a single resource but instead of start and finish times, each job i has
  - A time requirement τ_i which must be scheduled contiguously
  - A target deadline d_i by which the request is ideally finished
- Minimum start time is 0.
- Each item is graded a lateness: $\ell_i := \max\{0, t_i - d_i\}$ where t_i is it's end time
- Total lateness is defined as the **max** lateness: $L = \max_{i=1,\dots,n} \ell_i$.

crucially different than $L = \sum_{i=1..n} \ell_i$.

- **Goal:** Find a scheduling that *minimizes* the *maximum lateness* L.

Example minimizing lateness problem

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Finding the right greedy strategy

- Greedy template suggests finding a strategy and seeing if there are any glaring counterexamples.
  - **Shortest processing time.** Sort the jobs according to τ_i and select in order.
  - **Earliest deadline first.** Sort according to d_i and select in order.
  - **Smallest slack.** Sort according to slack, $d_i - \tau_i$, and select in order.

Counterexamples

Shortest processing time

- Sort the jobs according to τ_i and select in order.

	1	2
τ_i	1	10
d_i	100	10

Job 1 is selected due to shorter duration.

But then Job 2 incurs a lateness of 1.

Opposite order has 0 lateness.

Counterexamples

Smallest slack

- Sort according to slack, $d_i - \tau_i$, and select in order.

	1	2
τ_i	1	10
d_i	2	10

Job 2 has smaller slack.

Causes a lateness of $11 - 2 = 9$

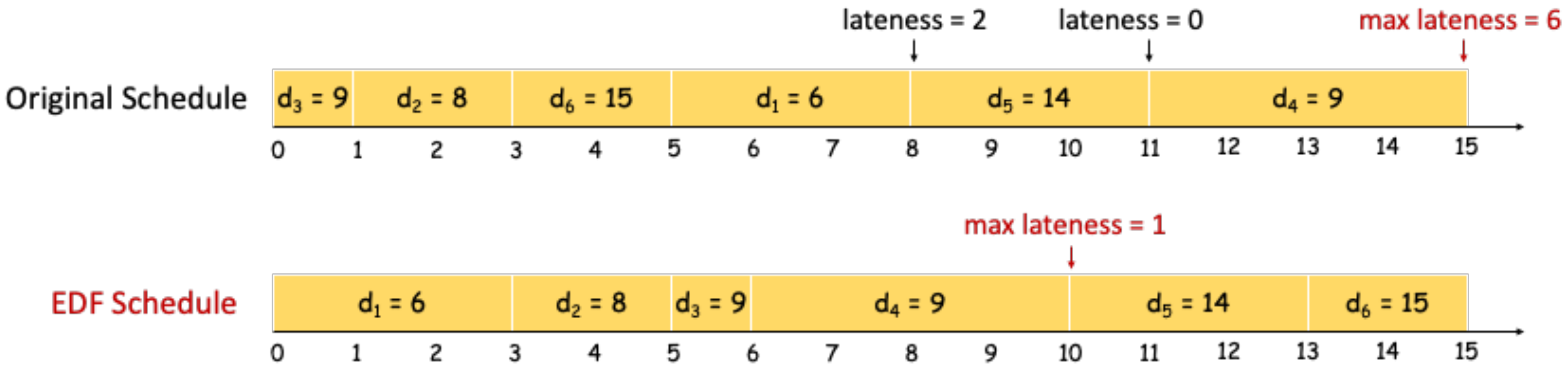
Other order has lateness of 1.

Earliest deadline first (EDF)

- **Algorithm:**
  - Sort deadlines in increasing order $d_1 \leq d_2 \leq \ldots \leq d_n$.
  - Set $T \leftarrow 0$.
  - For $i \leftarrow 1$ to n
    - Assign job i to run in interval. $(T, T + \tau_i)$. Increment $T \leftarrow T + \tau_i$.

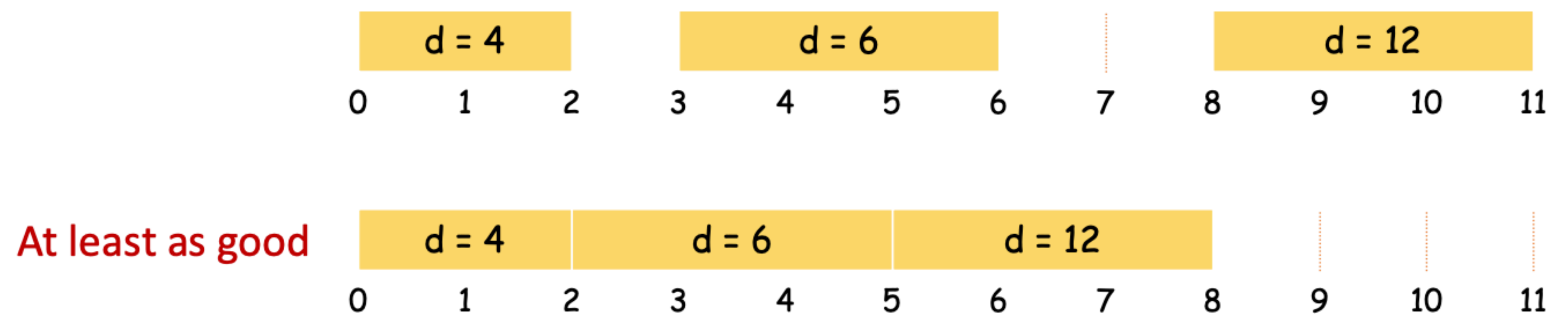
Example EDF schedule

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Exchange argument for optimality

- If for any solution there exists a modification that modifies solution but its value is at least as good as the original, then wlog optimal solution has modification
- Consider a solution with "gaps" between jobs



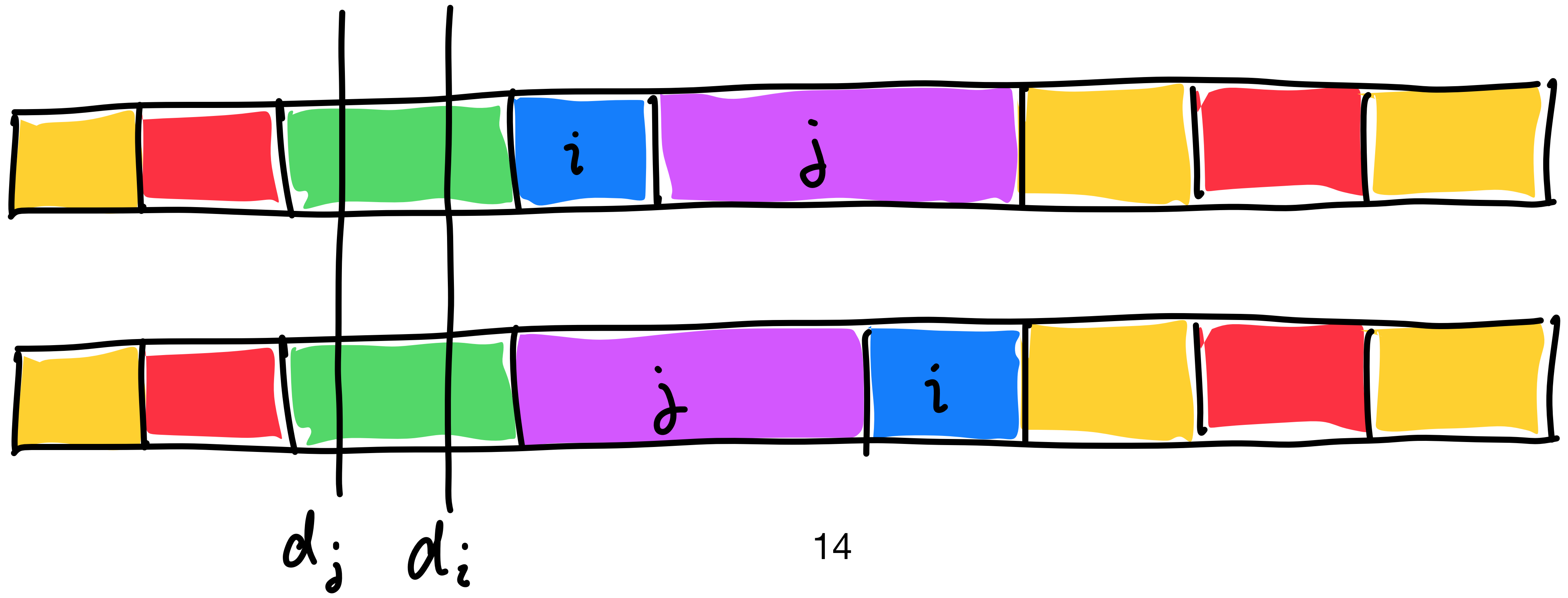
- Then a "gapless" solution by shifting every job earlier is just as good
  - Proof: The new t_i' for every job is at most t_i. And ℓ_i is monotonic in t_i. So, the new loss L' is at most L.

The EDF Schedule

- By construction, the EDF schedule is gapless
- This doesn't alone prove optimality
- **Property of EDF:** No inversions in EDF schedule.
  - An inversion is if job i is before job j but $d_i > d_j$.
  - An inversion is adjacent if it occurs between adjacent jobs.
- **Exchange principle:** If a schedule has an adjacent inversion, flipping the adjacent inversion yields a schedule of shorter lateness.

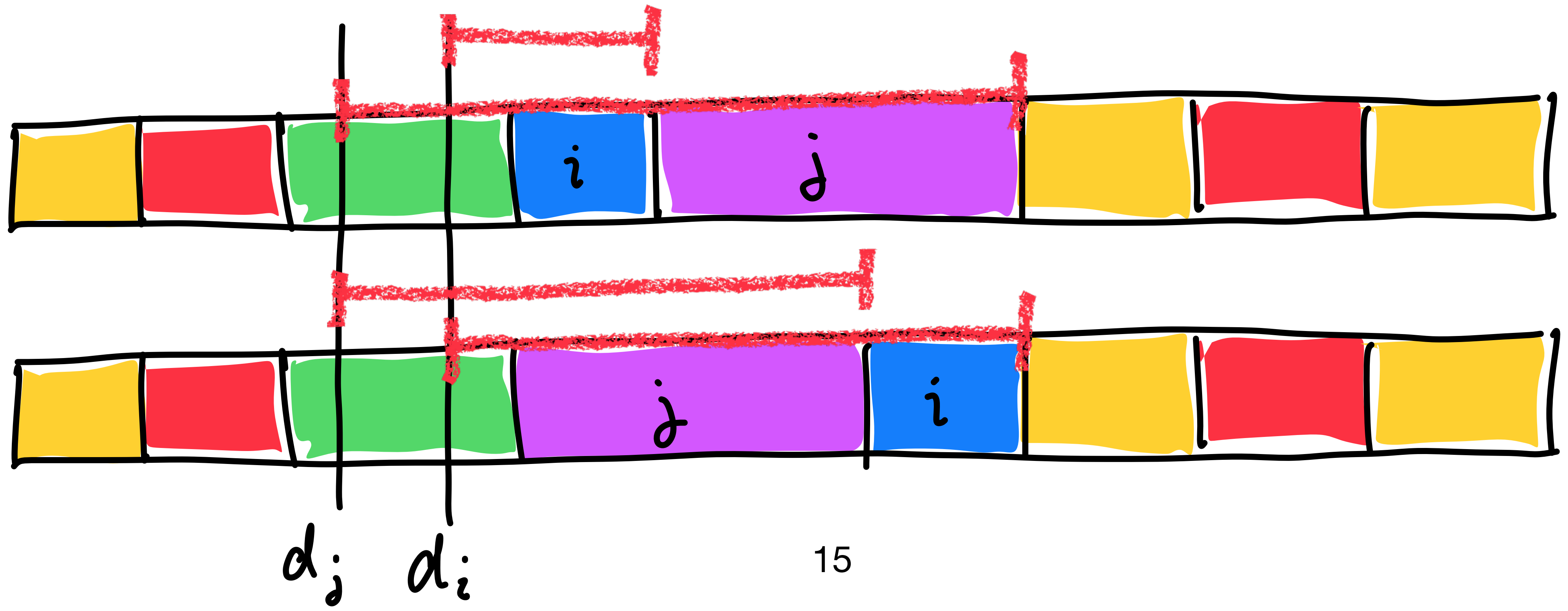
The EDF Schedule

- **Property of EDF:** No inversions in EDF schedule.
  - An inversion is if job i is before job j but $d_i > d_j$.
  - An inversion is adjacent if it occurs between adjacent jobs.
- **Exchange principle:** If a schedule has an adjacent inversion, flipping the adjacent inversion yields a schedule of shorter max lateness.
- **Proof:**



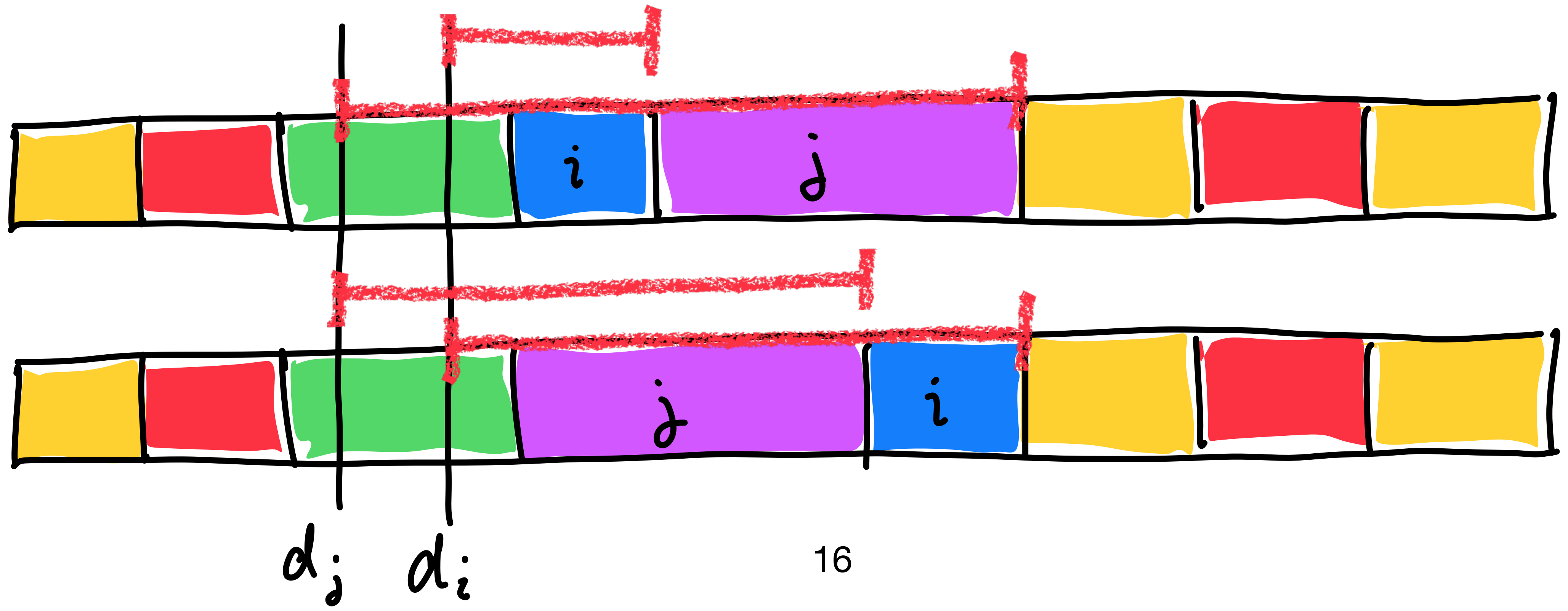
The EDF Schedule

- **Property of EDF:** No inversions in EDF schedule.
    - An inversion is if job i is before job j but $d_i > d_j$.
    - An inversion is adjacent if it occurs between adjacent jobs.
- **Exchange principle:** If a schedule has an adjacent inversion, flipping the adjacent inversion yields a schedule of shorter max lateness.
- **Proof:**



The EDF Schedule

- **Property of EDF:** No inversions in EDF schedule.
    - An inversion is if job i is before job j but $d_i > d_j$.
    - An inversion is adjacent if it occurs between adjacent jobs.
- **Exchange principle:** If a schedule has an adjacent inversion, flipping the adjacent inversion yields a schedule of shorter max lateness.
- **Proof:**



Notice: max lateness decreases by fixing inversion.

Sum of lateness may increase.

Inversion removal

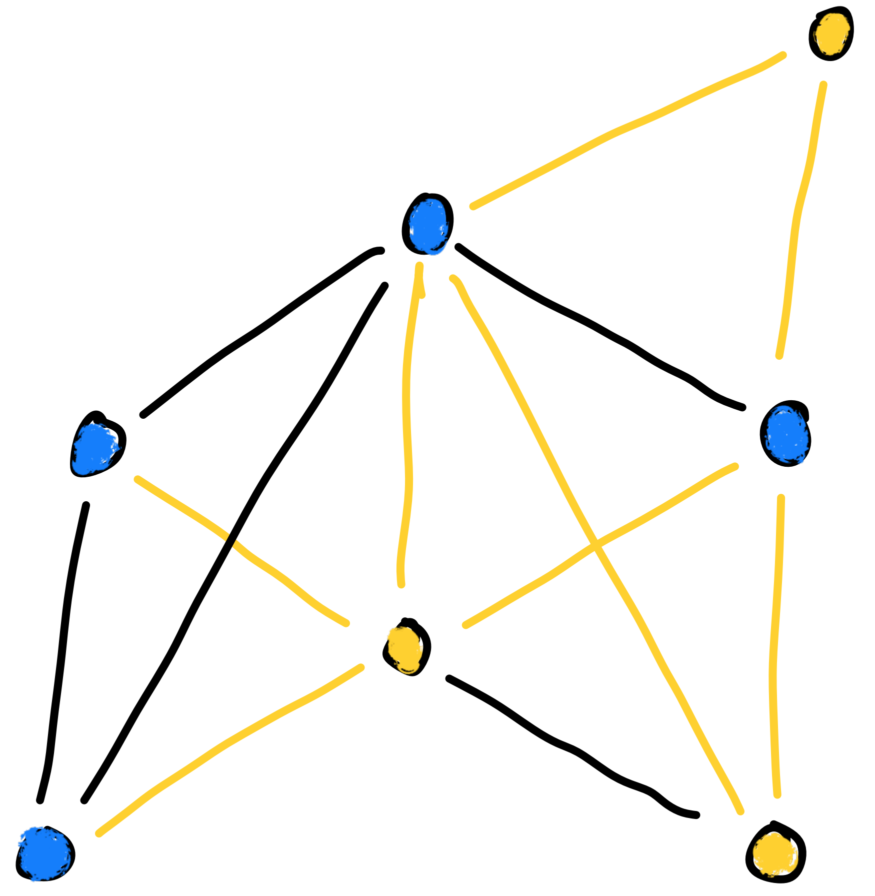
- **Lemma (exercise):** If a schedule has an inversion, then it must also have an *adjacent inversion*
  - Hint: Prove by induction, that if (i, j) is an inversion for $i < j$ but (i, j') is not an inversion for $i < j' < j$, then $(j - 1, j)$ is an *adjacent inversion*
- Exchange principle lets us clean up all the adjacent inversions
- "Gapless" and "inversion" exchange principles yield a gapless schedule with no inversions
- This is precisely, the earliest deadline first (EDF) schedule up to events of equal deadline. All such schedules have same lateness. Thus, it is optimal

Maximizing bipartiteness

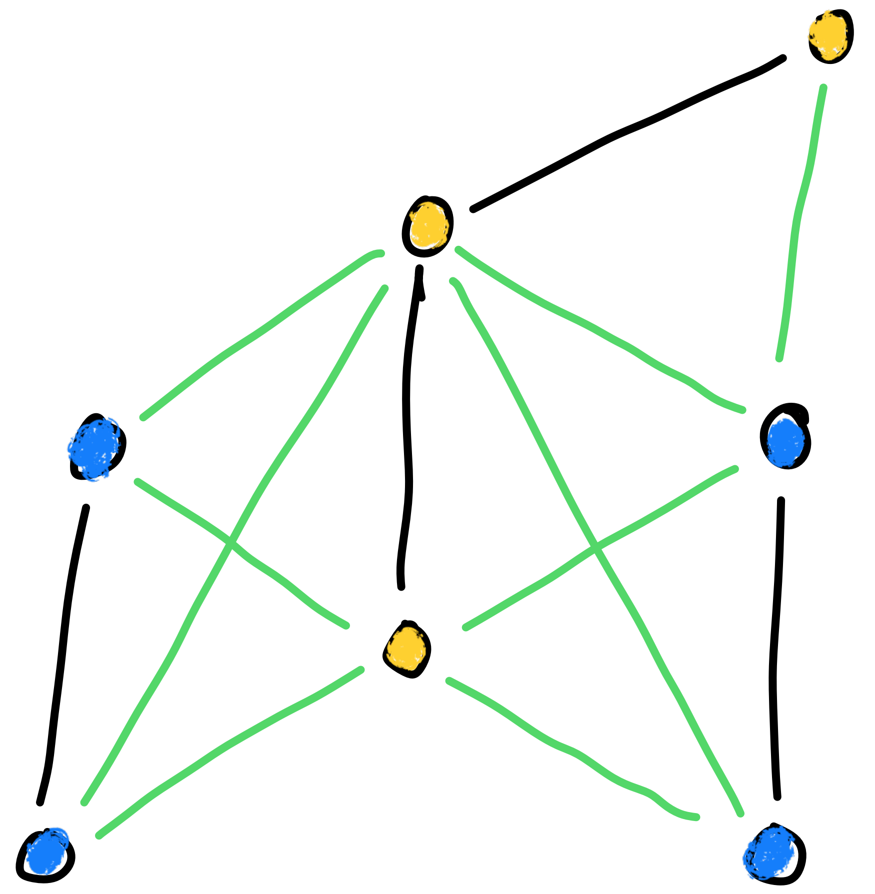
- We saw how to verify if a graph is bipartite or not using a BFS algorithm
- We could also come up with a “measure of bipartiteness”
- $$\text{maxcut}(G) = \max_{C:V\to\{0,1\}} \underbrace{\sum_{(u,v)\in E} \mathbf{1}_{\{C(u)\neq C(v)\}}}$$

  number of correctly colored edges

  - For each possible coloring C, measure how many edges are colored “correctly”
  - The $\text{maxcut}(G)$ is the max number of edges colored correctly over all colorings
- Deciding if $\text{maxcut}(G) = m$ or $\neq m$ can be done by the BFS algorithm
- Is there an algorithm for computing $\text{maxcut}(G)$ in general?



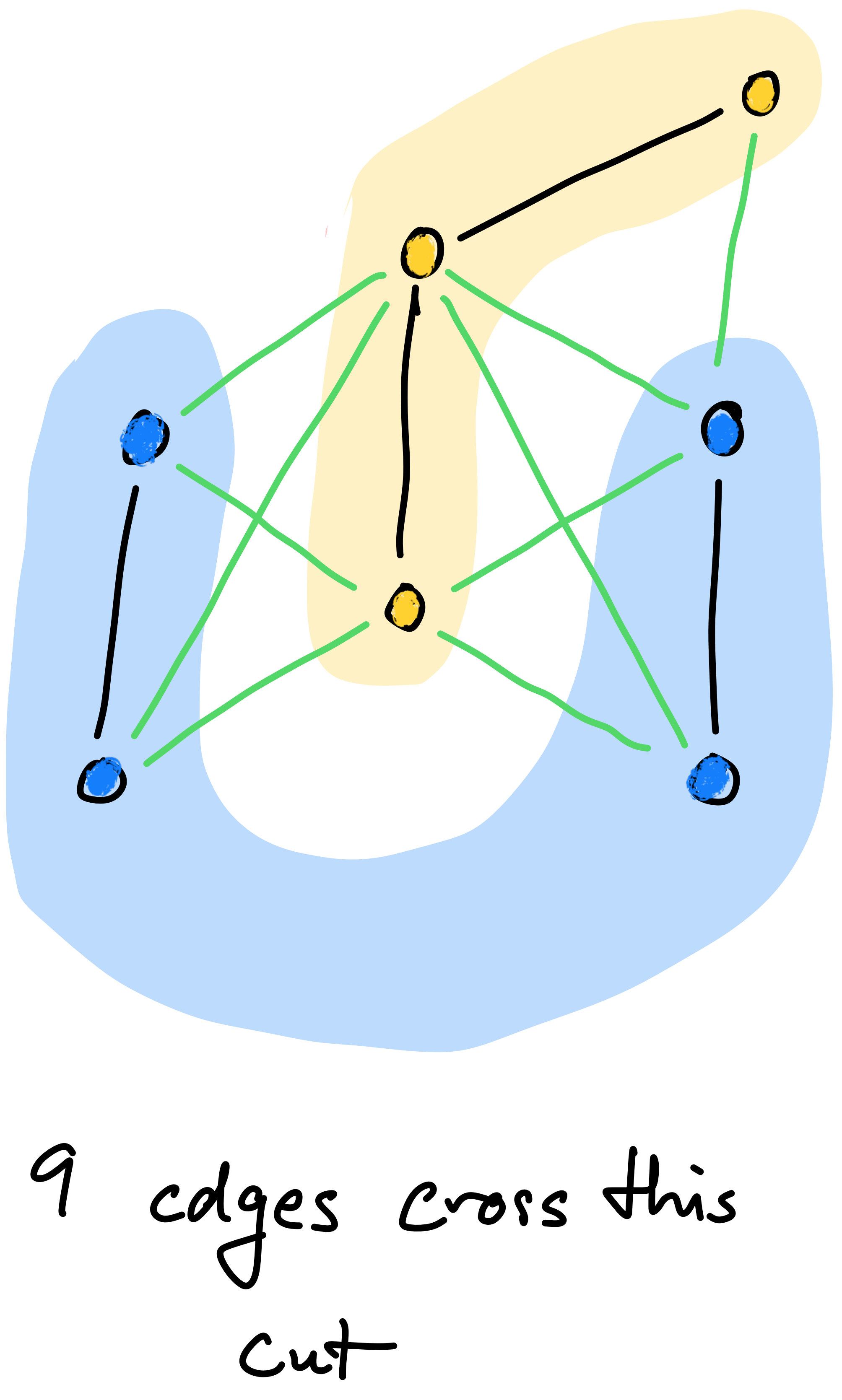
8 edges correctly colored



9 edges correctly colored

Why is it called MaxCut?

- The fn. $C : V \to \{0,1\}$ partitions the vertices in two sets (yellow and blue).
  - A partition of the vertices into two sets (S, T) is also called a **cut**.
  - We say that an edge (u, v) **crosses** the cut if $u \in S$ and $v \in T$.
  - $\text{maxcut}(G) = \max_{C:V\to\{0,1\}} \sum_{(u,v)\in E} \mathbf{1}_{\{C(u)\neq C(v)\}}$ counts the maximum number of edges that cross any cut.
  - Computing "bipartiteness" is equivalent to computing the max cut.

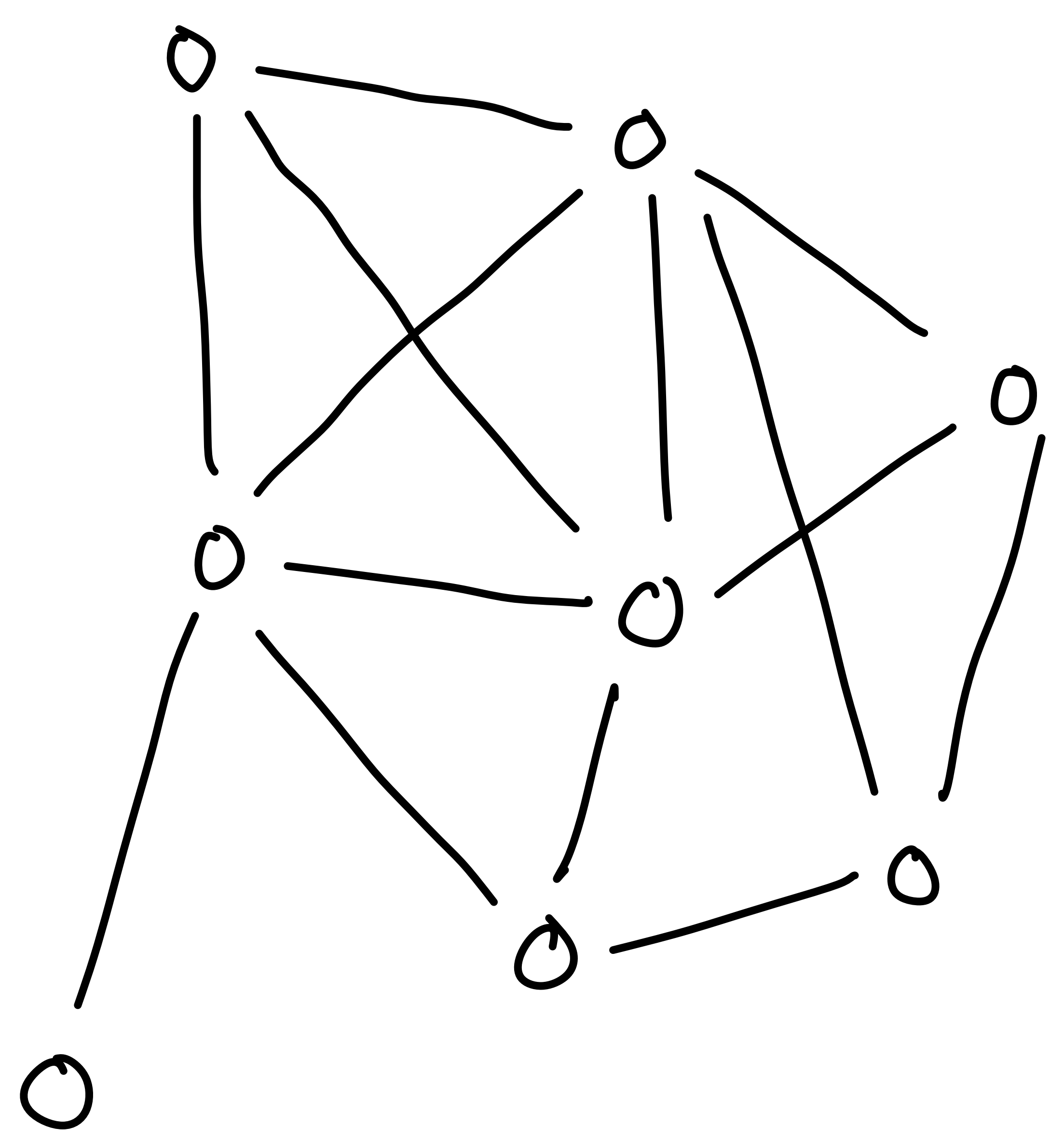


A proof that Max Cut is always $\geq m/2$

- Choose $C : V \to \{0,1\}$ uniformly randomly and independently.
- Then for any edge $(u, v) = e \in E$, let X_e be the event that e crosses the cut.
- Since $C(u)$ and $C(v)$ are chosen uniformly randomly, $\mathbb{E}X_e = 1/2$.
- By linearity of expectation, $\mathbb{E}\sum_{e \in E} X_e = \sum_{e \in E} \mathbb{E}X_e = \frac{m}{2}$.
- A random cut C crosses $m/2$ edges. Therefore, there exists a cut that crosses $\geq m/2$ edges and $m/2 \leq \mathrm{maxcut}(G) \leq m$.

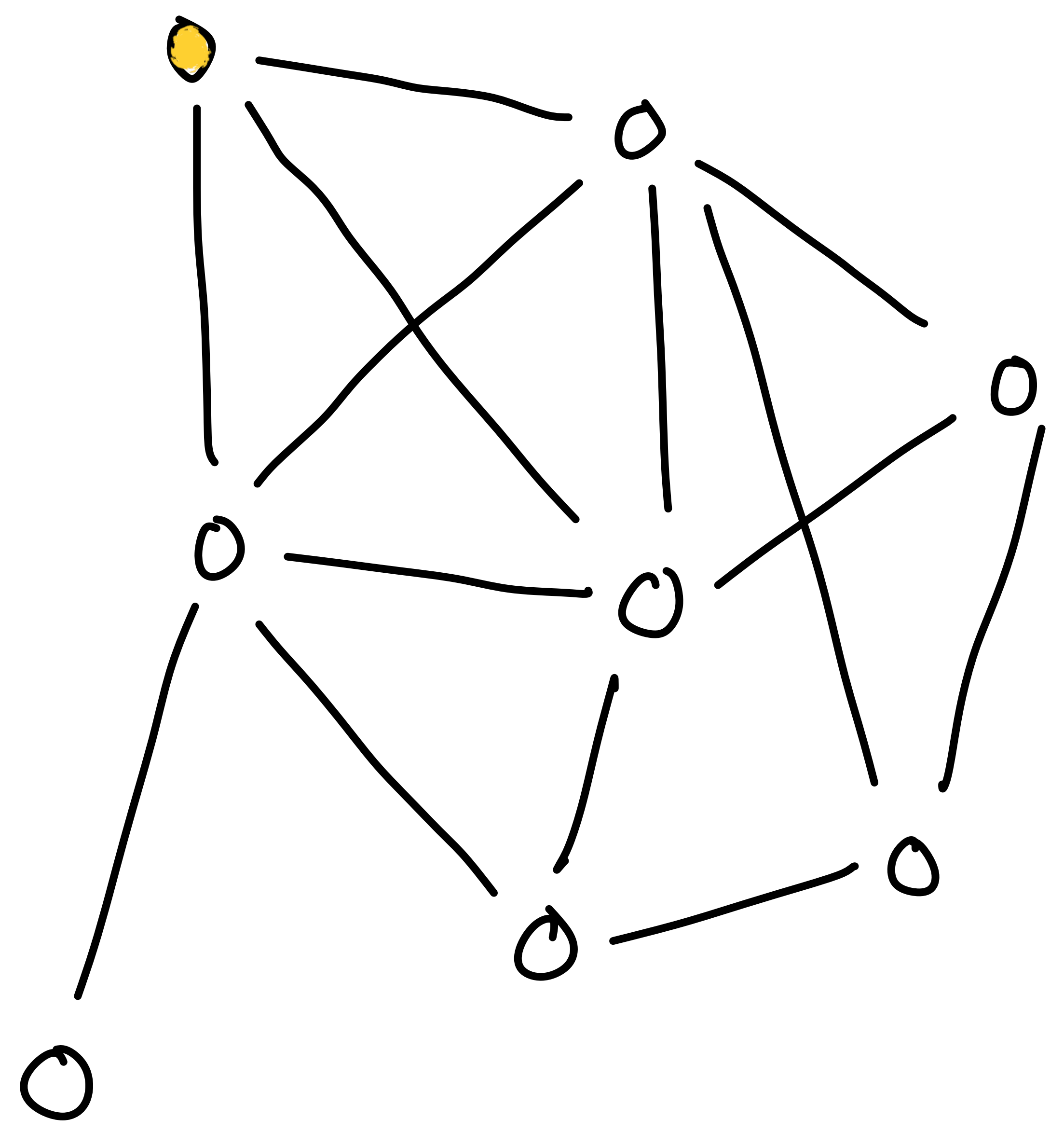
Finding a cut crossing $\geq m/2$ edges

- We know a cut exists crossing $\geq m/2$ edges. Can we find it efficiently?
- Let's use a greedy algorithm.
- **Algorithm overview:** Color the first vertex as 0 (yellow). Then, for every future vertex v, if v has more 0 (yellow) neighbors than 1's (blues), assign it the color 1 (blue), otherwise assign it 0 (yellow).



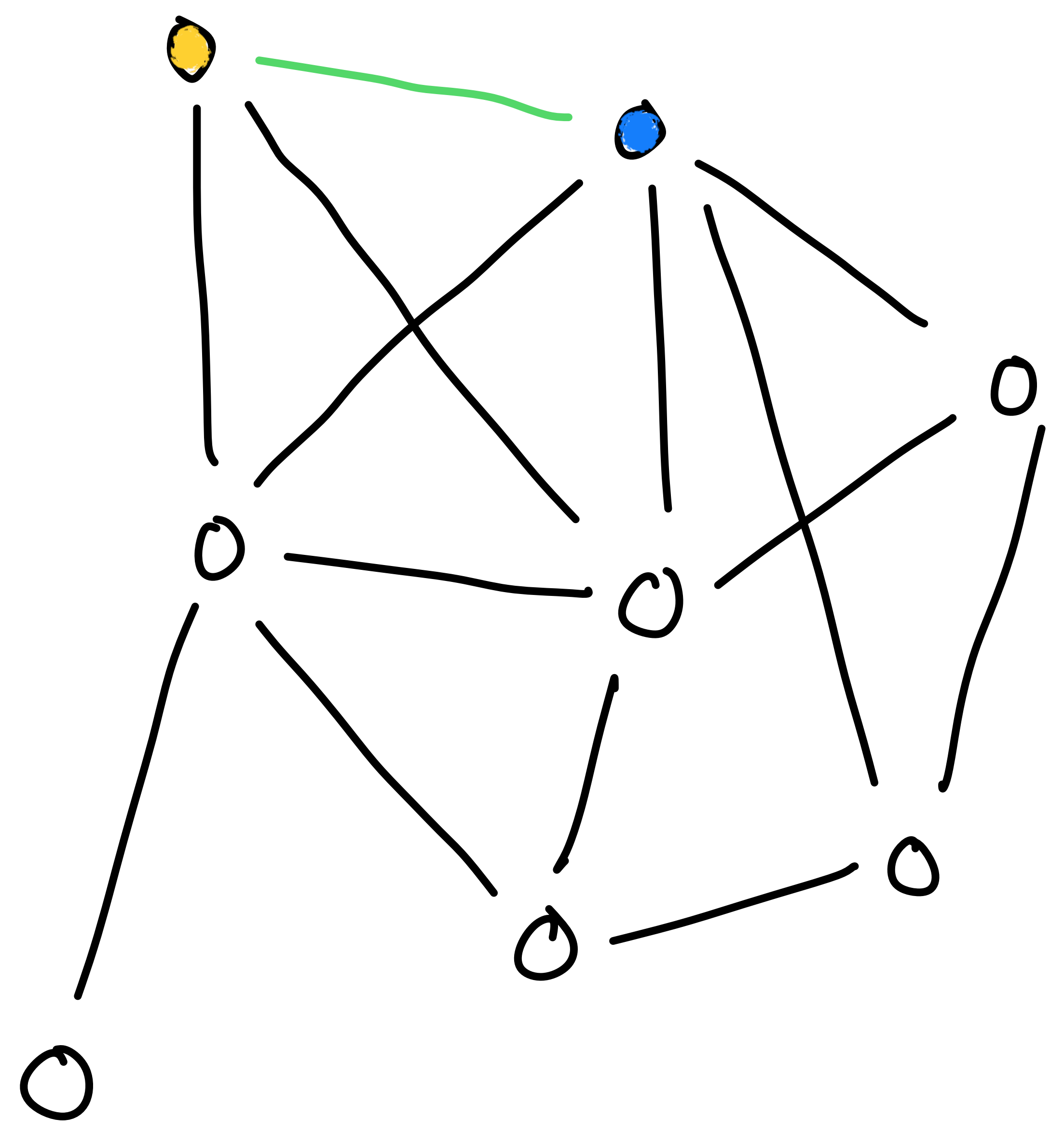
Finding a cut crossing $\geq m/2$ edges

- We know a cut exists crossing $\geq m/2$ edges. Can we find it efficiently?
- Let's use a greedy algorithm.
- **Algorithm overview:** Color the first vertex as 0 (yellow). Then, for every future vertex v, if v has more 0 (yellow) neighbors than 1's (blues), assign it the color 1 (blue), otherwise assign it 0 (yellow).



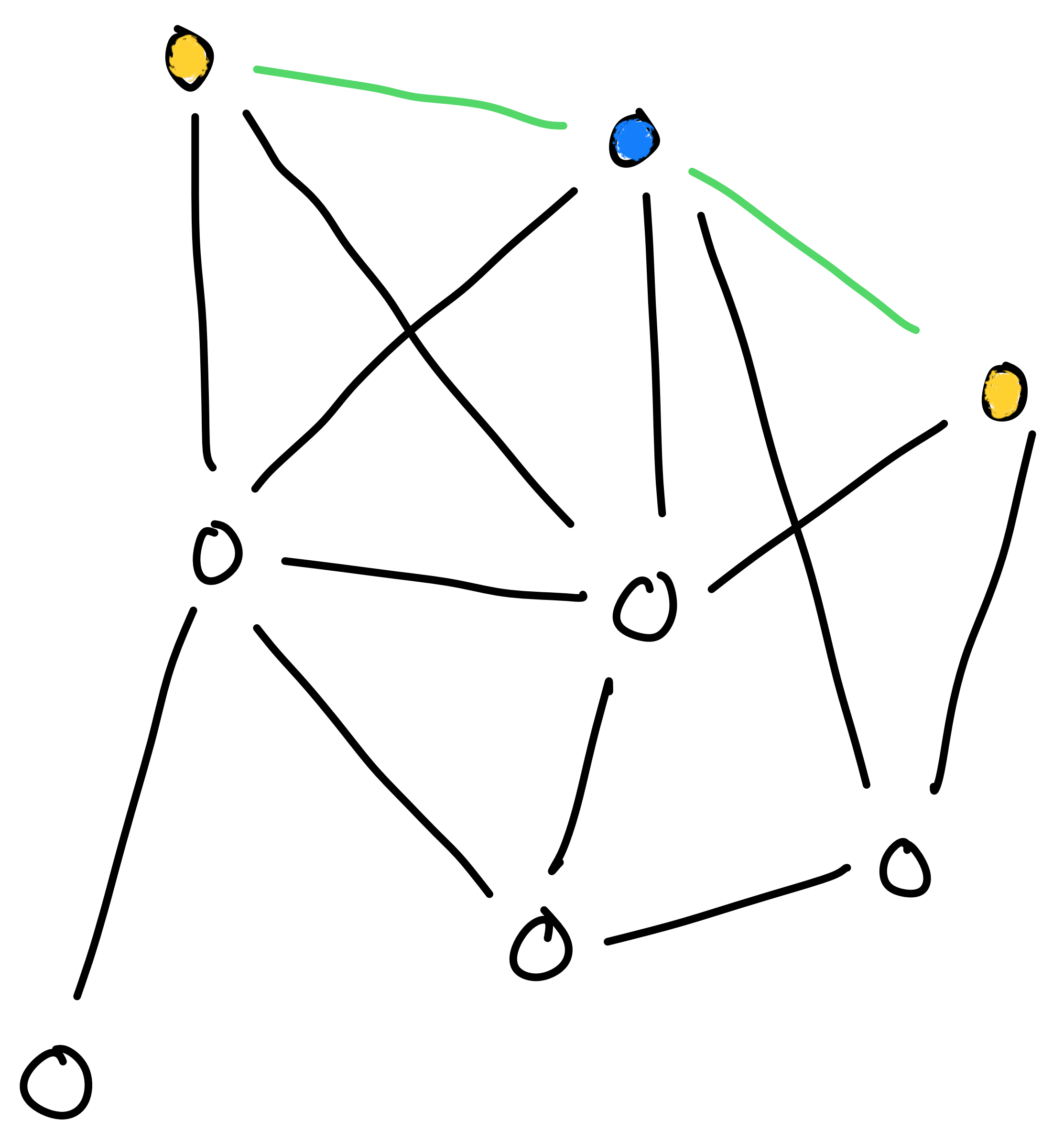
Finding a cut crossing $\geq m/2$ edges

- We know a cut exists crossing $\geq m/2$ edges. Can we find it efficiently?
- Let's use a greedy algorithm.
- **Algorithm overview:** Color the first vertex as 0 (yellow). Then, for every future vertex v, if v has more 0 (yellow) neighbors than 1's (blues), assign it the color 1 (blue), otherwise assign it 0 (yellow).



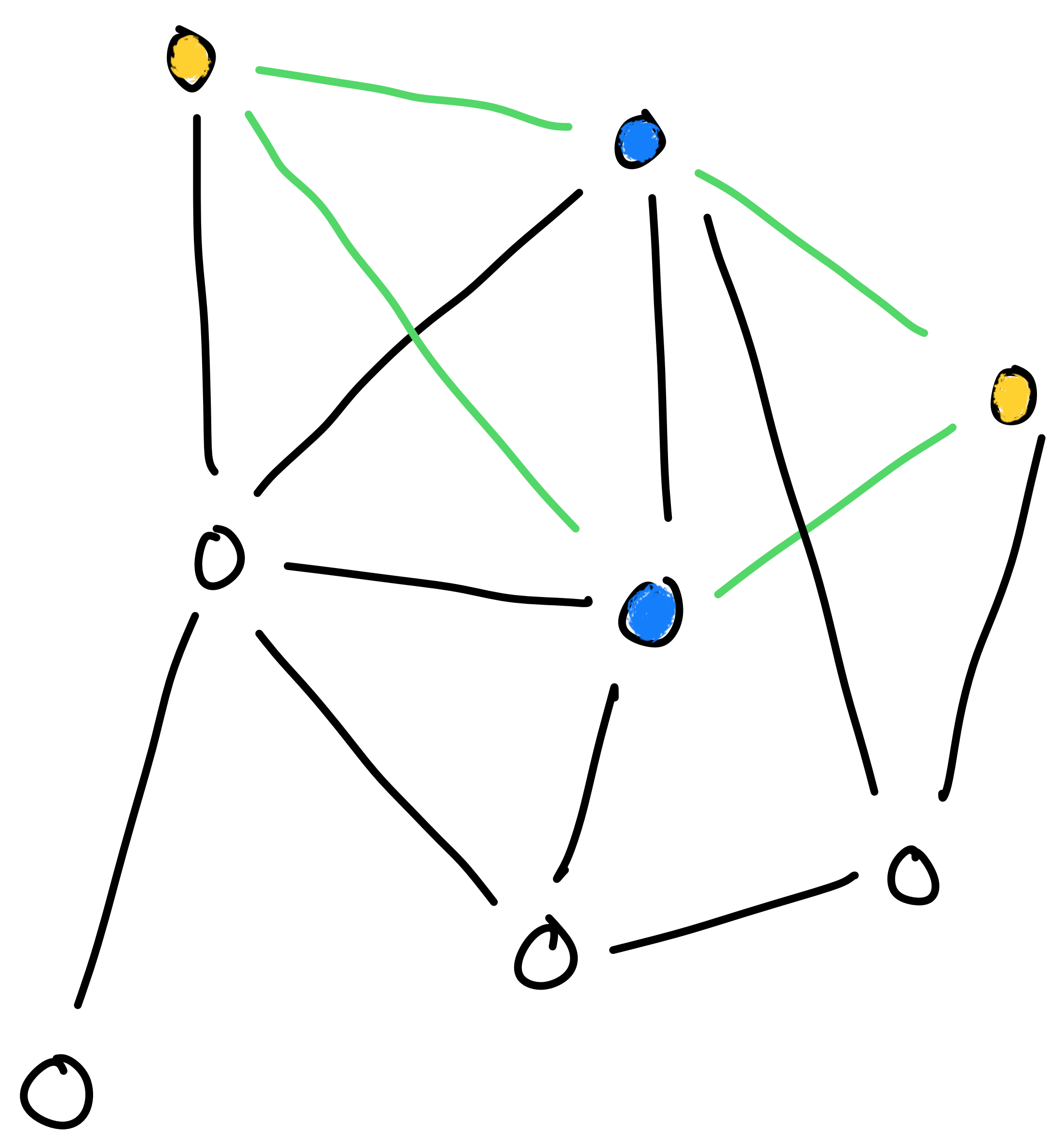
Finding a cut crossing $\geq m/2$ edges

- We know a cut exists crossing $\geq m/2$ edges. Can we find it efficiently?
- Let's use a greedy algorithm.
- **Algorithm overview:** Color the first vertex as 0 (yellow). Then, for every future vertex v, if v has more 0 (yellow) neighbors than 1's (blues), assign it the color 1 (blue), otherwise assign it 0 (yellow).



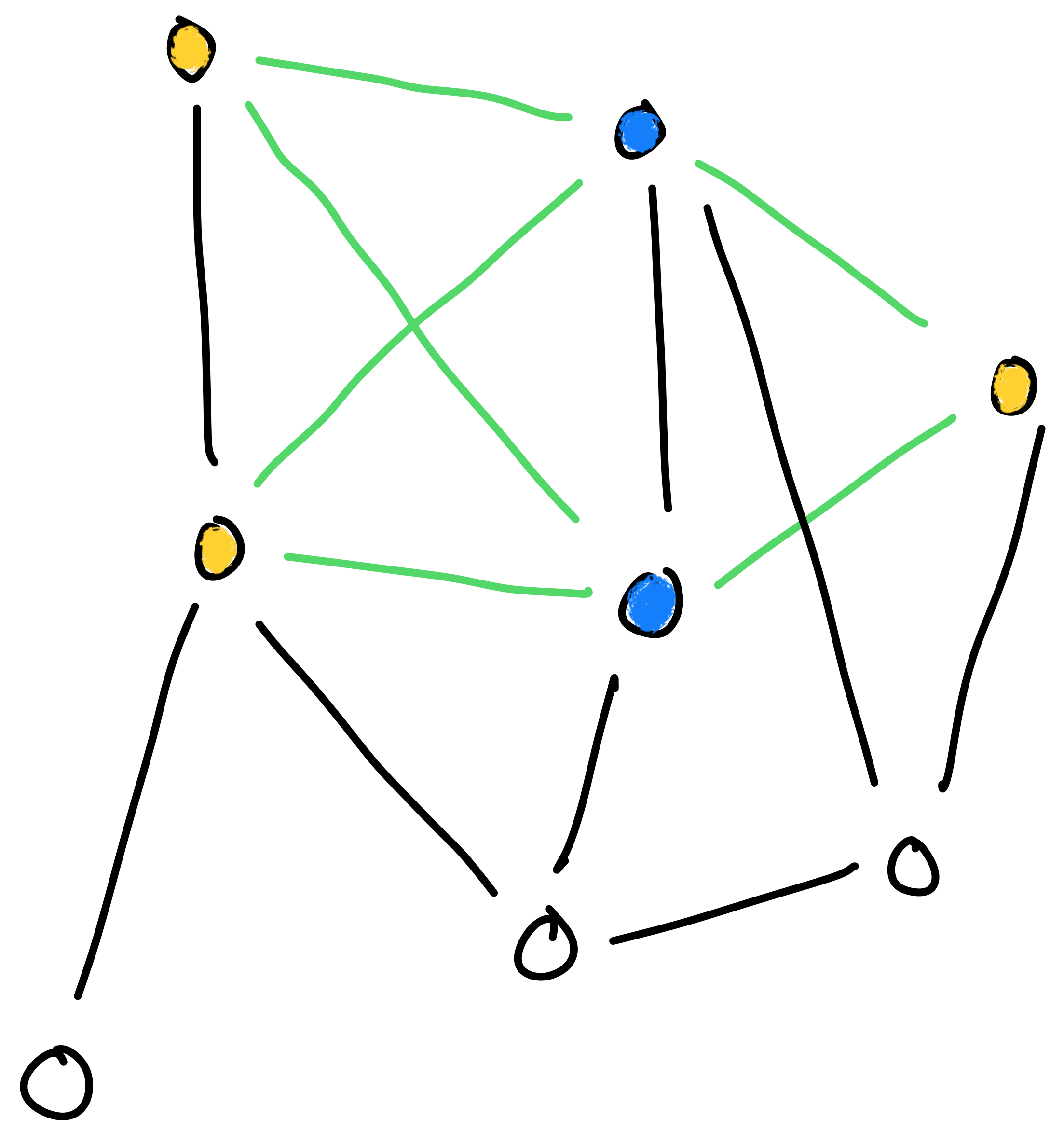
Finding a cut crossing $\geq m/2$ edges

- We know a cut exists crossing $\geq m/2$ edges. Can we find it efficiently?
- Let's use a greedy algorithm.
- **Algorithm overview:** Color the first vertex as 0 (yellow). Then, for every future vertex v, if v has more 0 (yellow) neighbors than 1's (blues), assign it the color 1 (blue), otherwise assign it 0 (yellow).



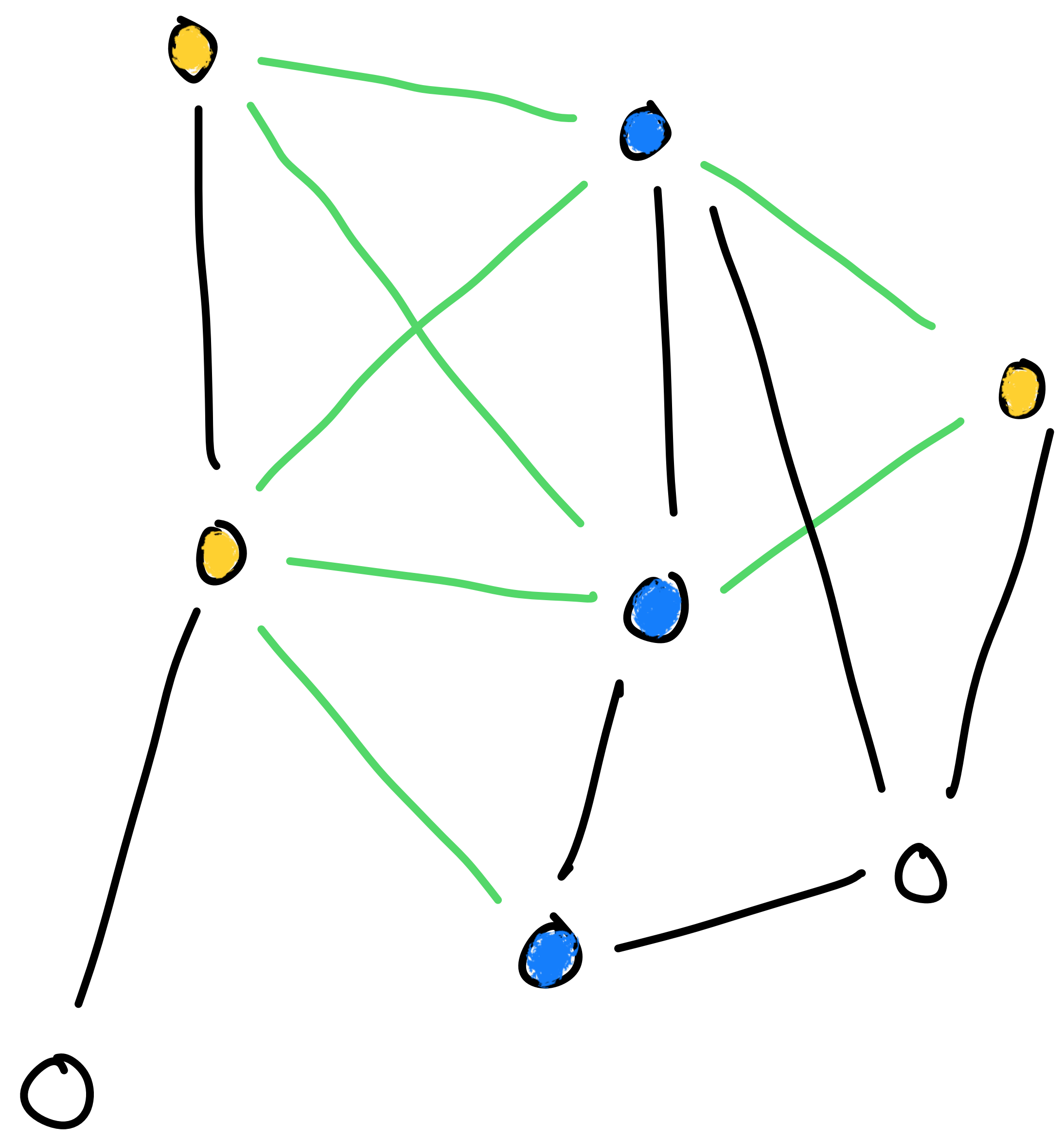
Finding a cut crossing $\geq m/2$ edges

- We know a cut exists crossing $\geq m/2$ edges. Can we find it efficiently?
- Let's use a greedy algorithm.
- **Algorithm overview:** Color the first vertex as 0 (yellow). Then, for every future vertex v, if v has more 0 (yellow) neighbors than 1's (blues), assign it the color 1 (blue), otherwise assign it 0 (yellow).



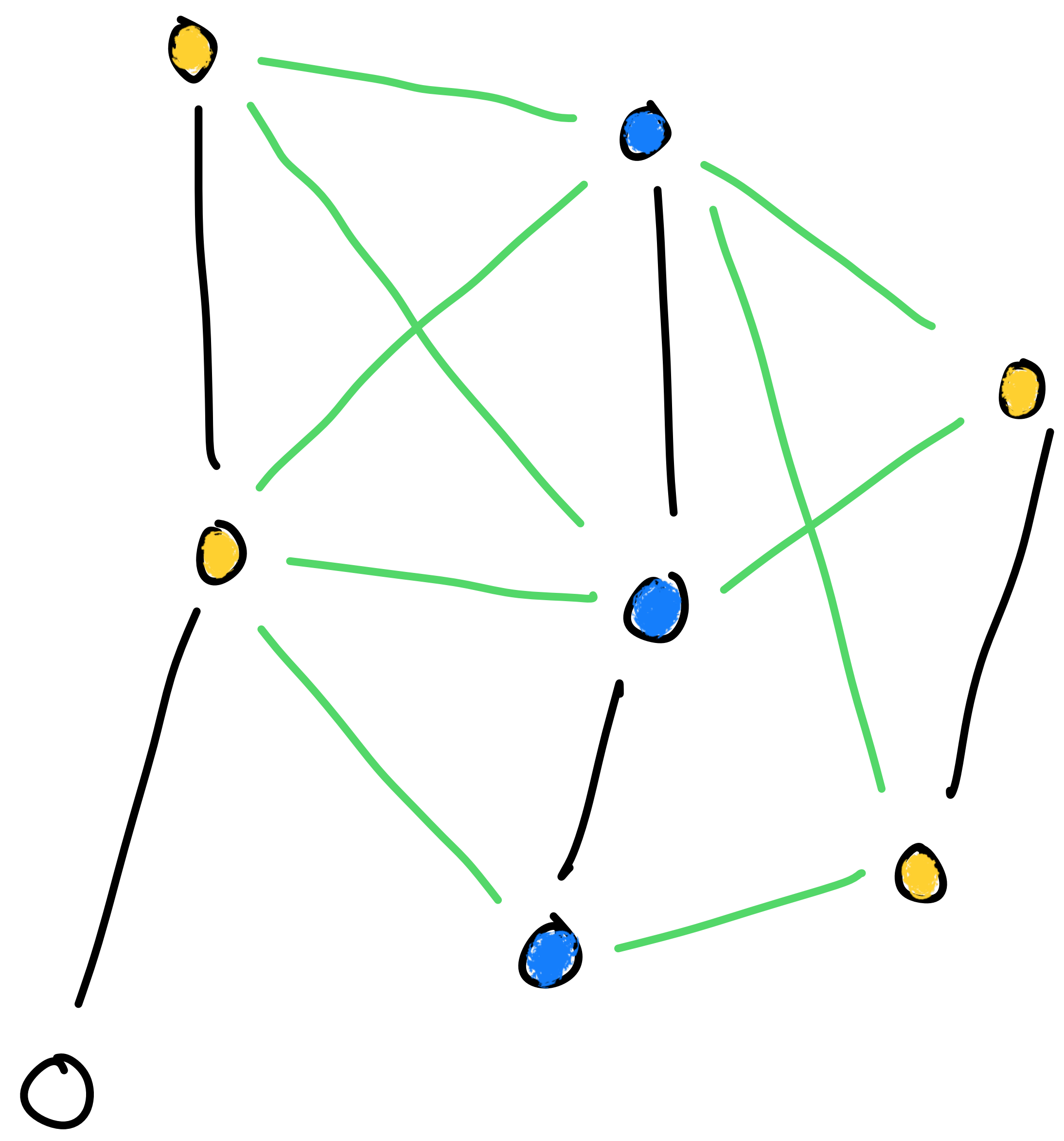
Finding a cut crossing $\geq m/2$ edges

- We know a cut exists crossing $\geq m/2$ edges. Can we find it efficiently?
- Let's use a greedy algorithm.
- **Algorithm overview:** Color the first vertex as 0 (yellow). Then, for every future vertex v, if v has more 0 (yellow) neighbors than 1's (blues), assign it the color 1 (blue), otherwise assign it 0 (yellow).



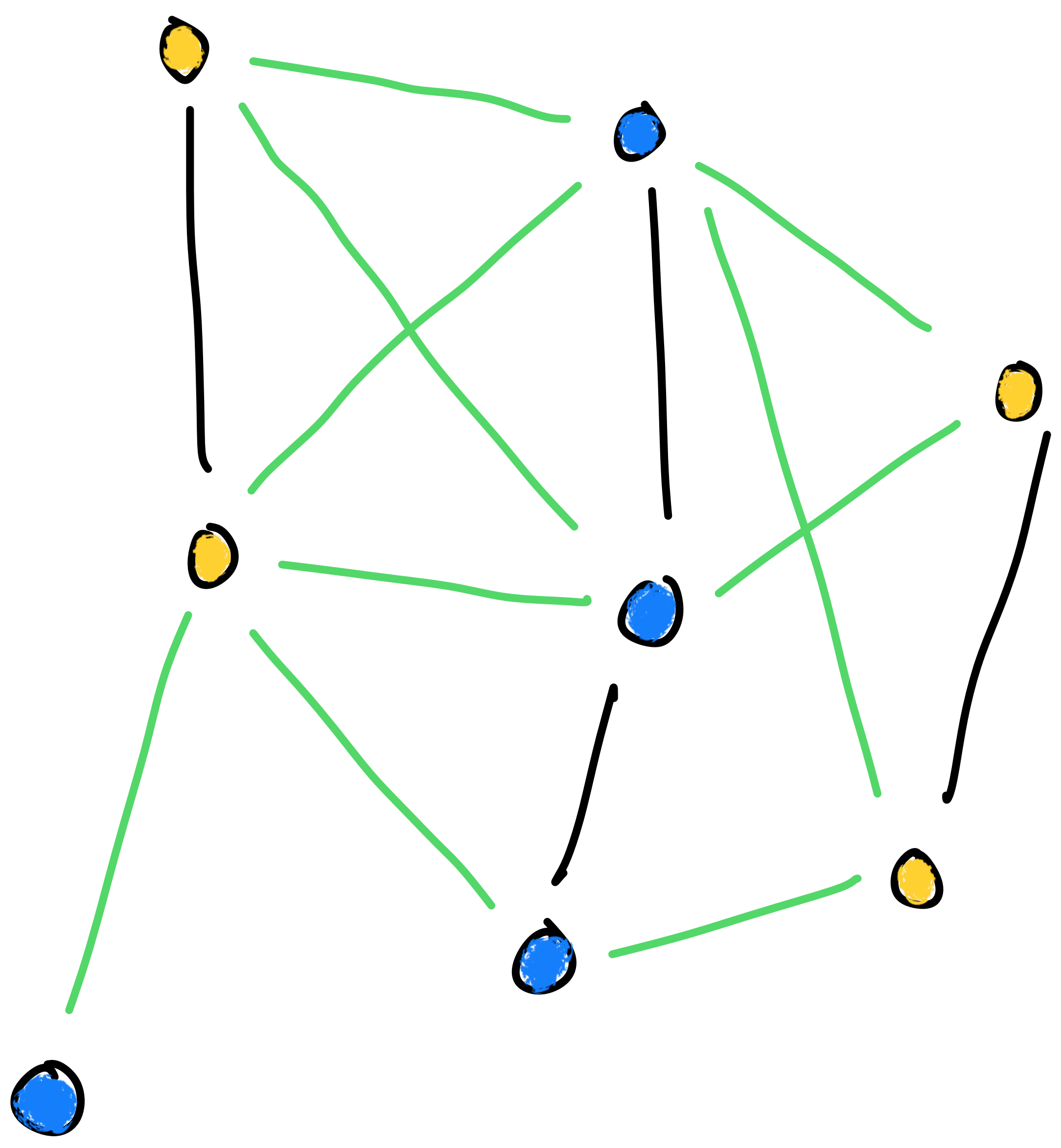
Finding a cut crossing $\geq m/2$ edges

- We know a cut exists crossing $\geq m/2$ edges. Can we find it efficiently?
- Let's use a greedy algorithm.
- **Algorithm overview:** Color the first vertex as 0 (yellow). Then, for every future vertex v, if v has more 0 (yellow) neighbors than 1's (blues), assign it the color 1 (blue), otherwise assign it 0 (yellow).



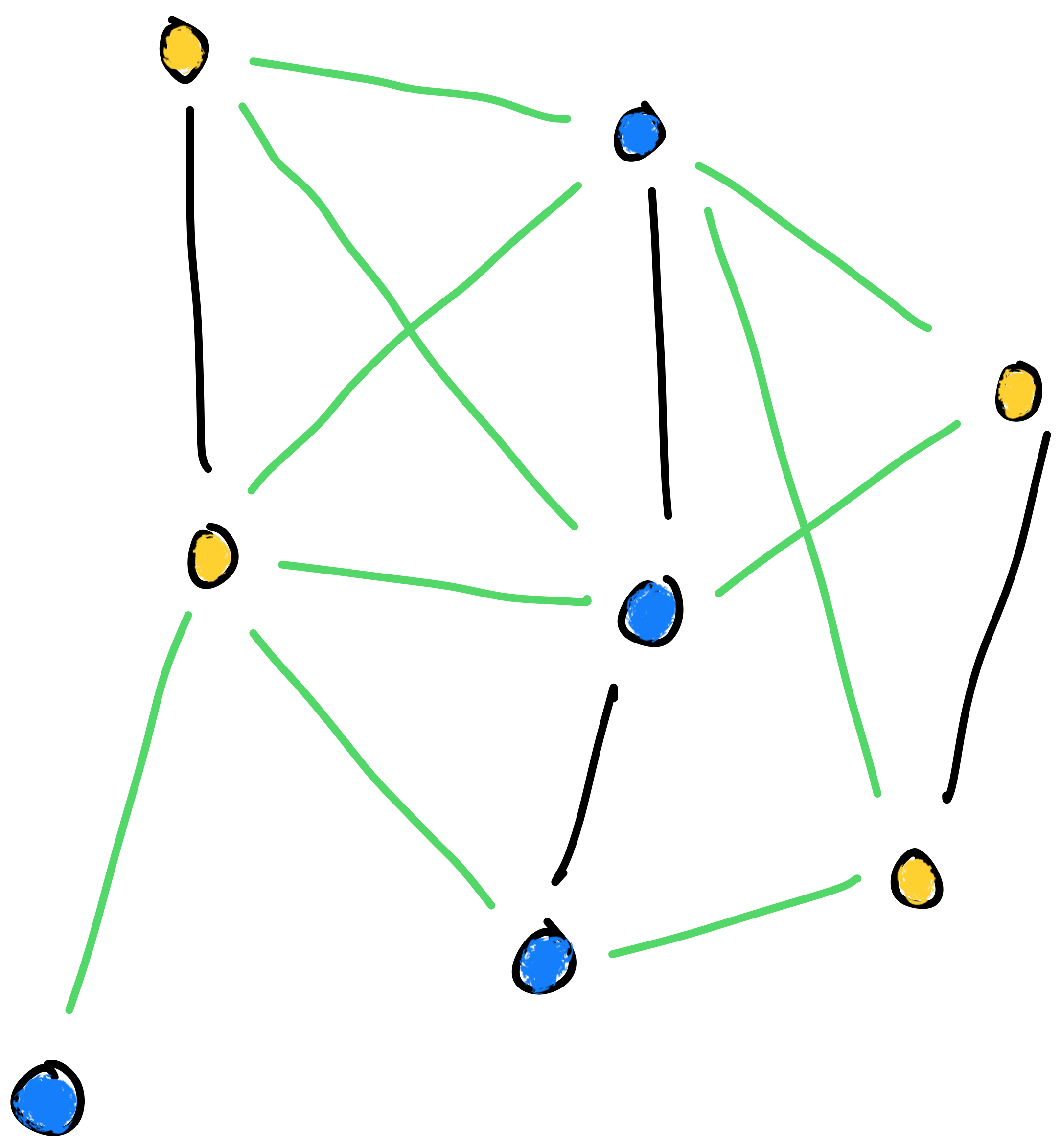
Finding a cut crossing $\geq m/2$ edges

- We know a cut exists crossing $\geq m/2$ edges. Can we find it efficiently?
- Let's use a greedy algorithm.
- **Algorithm overview:** Color the first vertex as 0 (yellow). Then, for every future vertex v, if v has more 0 (yellow) neighbors than 1's (blues), assign it the color 1 (blue), otherwise assign it 0 (yellow).



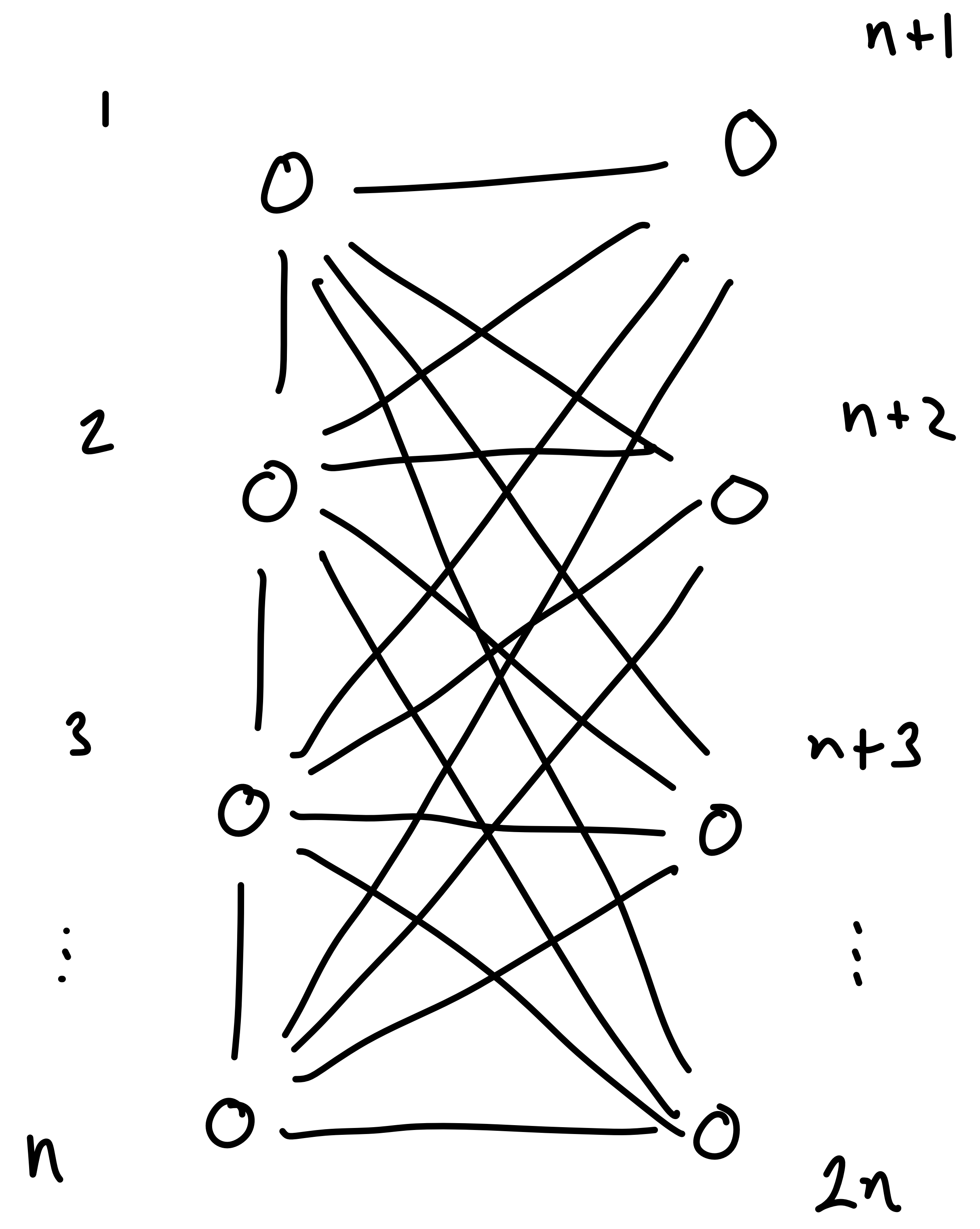
Finding a cut crossing $\geq m/2$ edges

- We know a cut exists crossing $\geq m/2$ edges. Can we find it efficiently?
- Let's use a greedy algorithm.
- **Algorithm overview:** Color the first vertex as 0 (yellow). Then, for every future vertex v, if v has more 0 (yellow) neighbors than 1's (blues), assign it the color 1 (blue), otherwise assign it 0 (yellow).



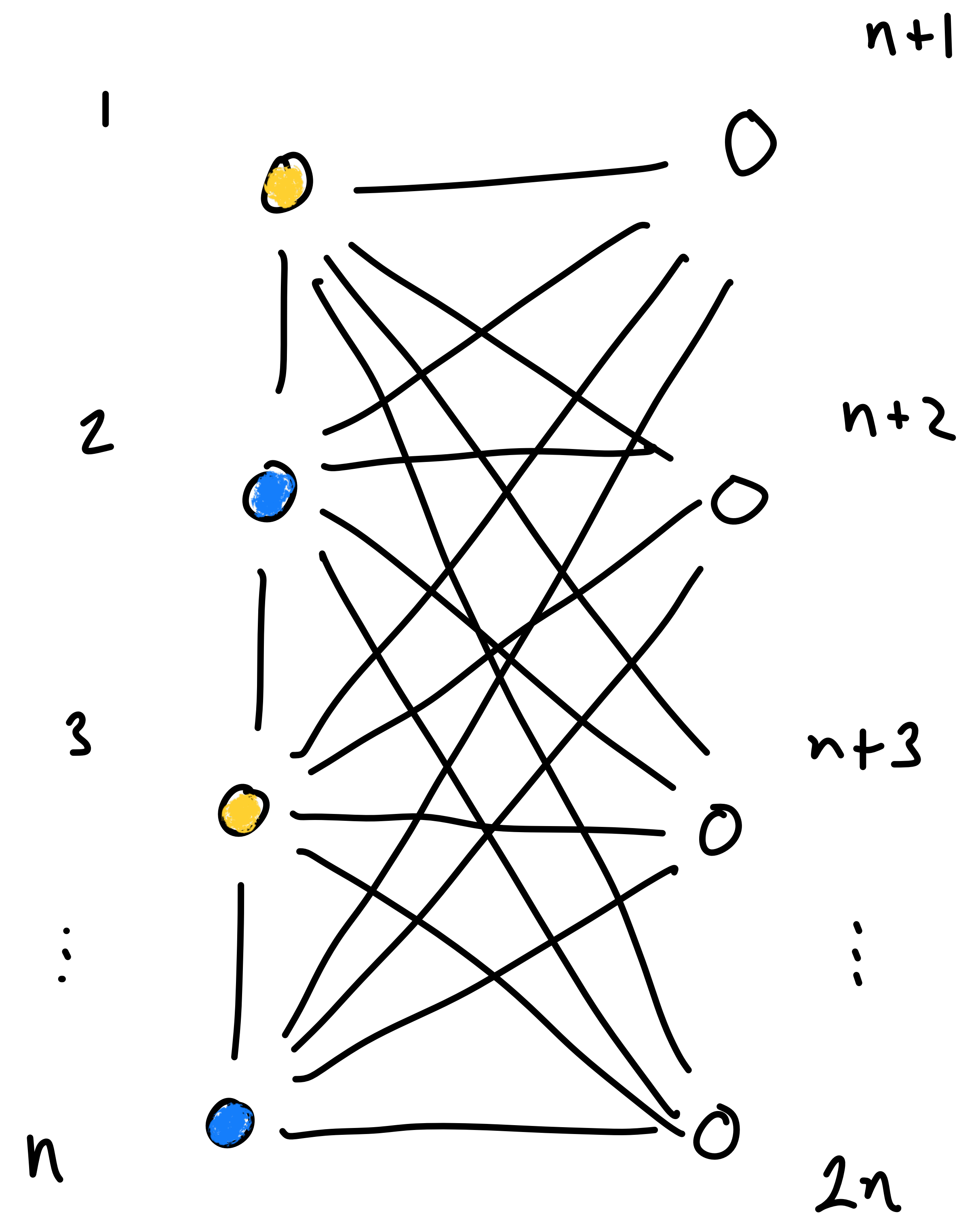
Greedy algorithm can be suboptimal

- The greedy algorithm can be suboptimal and fail to find the max cut.
- One can design examples where it fails.
- Consider the following graph on $2n$ vertices explored in the order that the vertices are numbered.



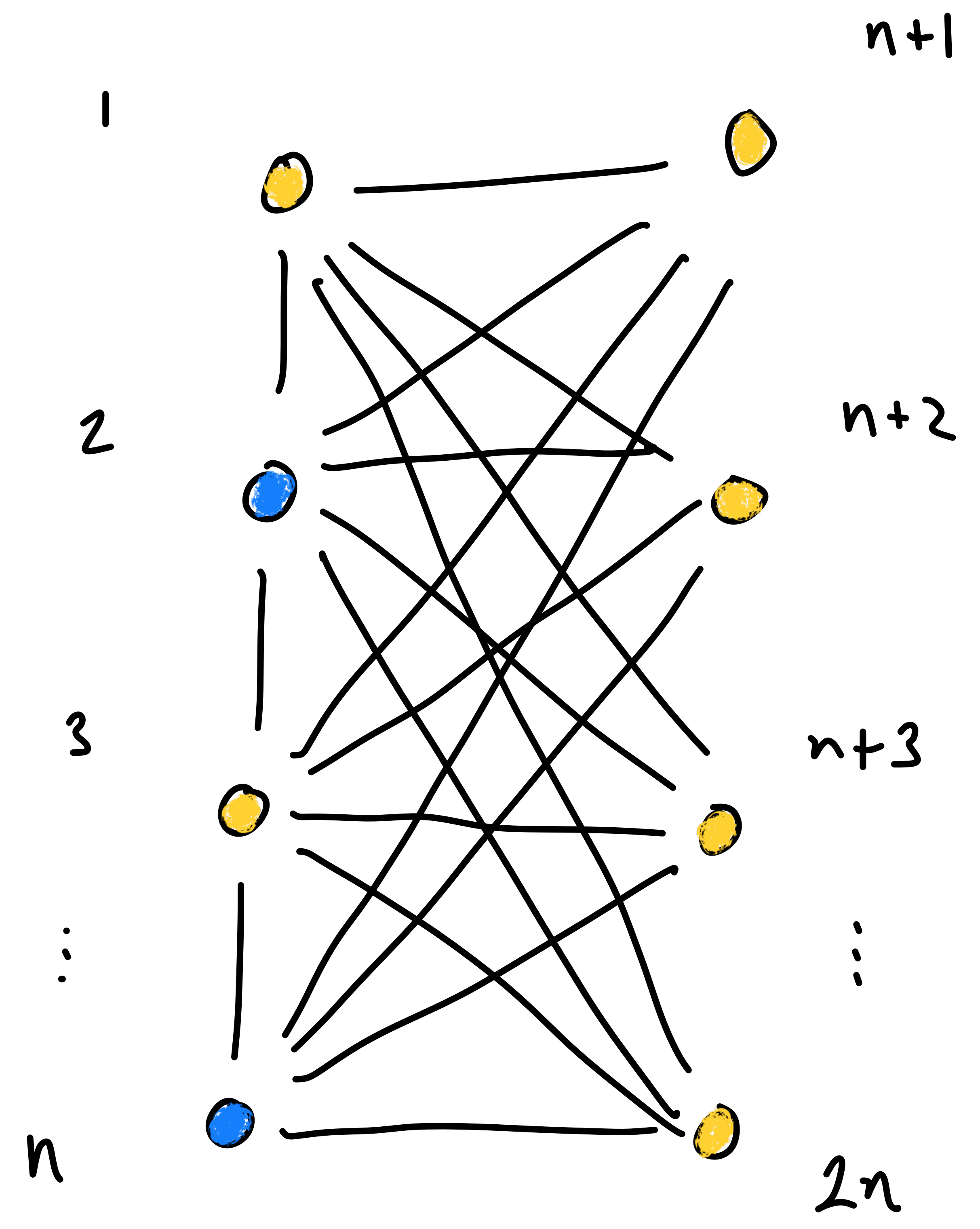
Greedy algorithm can be suboptimal

- The greedy algorithm can be suboptimal and fail to find the max cut.
- One can design examples where it fails.
- Consider the following graph on $2n$ vertices explored in the order that the vertices are numbered.
    - The left vertices have alternating colors.



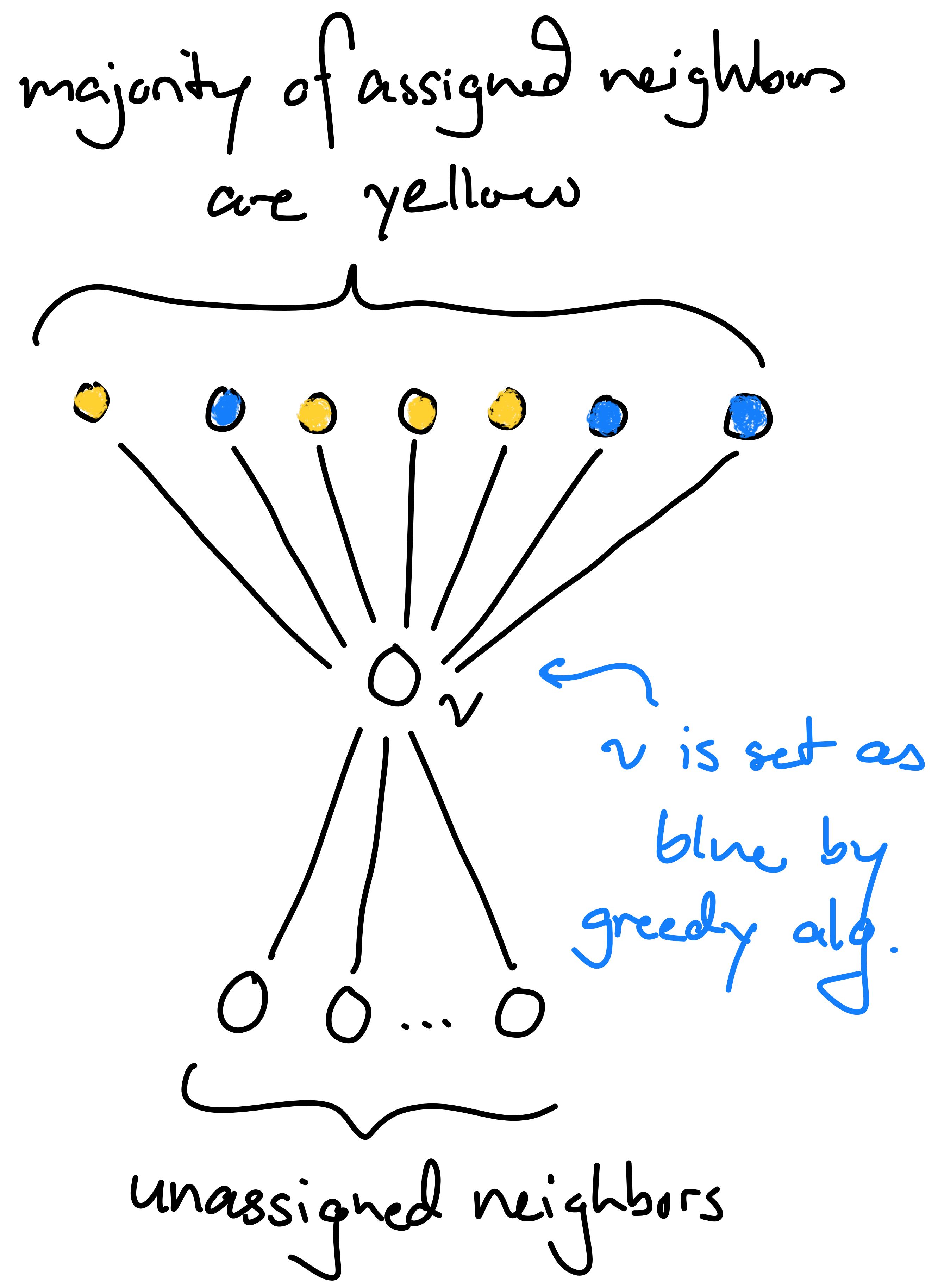
Greedy algorithm can be suboptimal

- The greedy algorithm can be suboptimal and fail to find the max cut.
- One can design examples where it fails.
- Consider the following graph on $2n$ vertices explored in the order that the vertices are numbered.
  - The left vertices have alternating colors.
  - Right vertices are all yellow.
- Greedy cut of the graph as $n^2/2 + (n-1)$ edges crossing cut. Optimal cut has n^2 edges crossing cut.
- So $\dfrac{|\text{greedy cut}|}{\text{maxcut}(G)} = \dfrac{\frac{n^2}{2} + (n-1)}{n^2} \to \dfrac{1}{2}$ as $n \to \infty$.



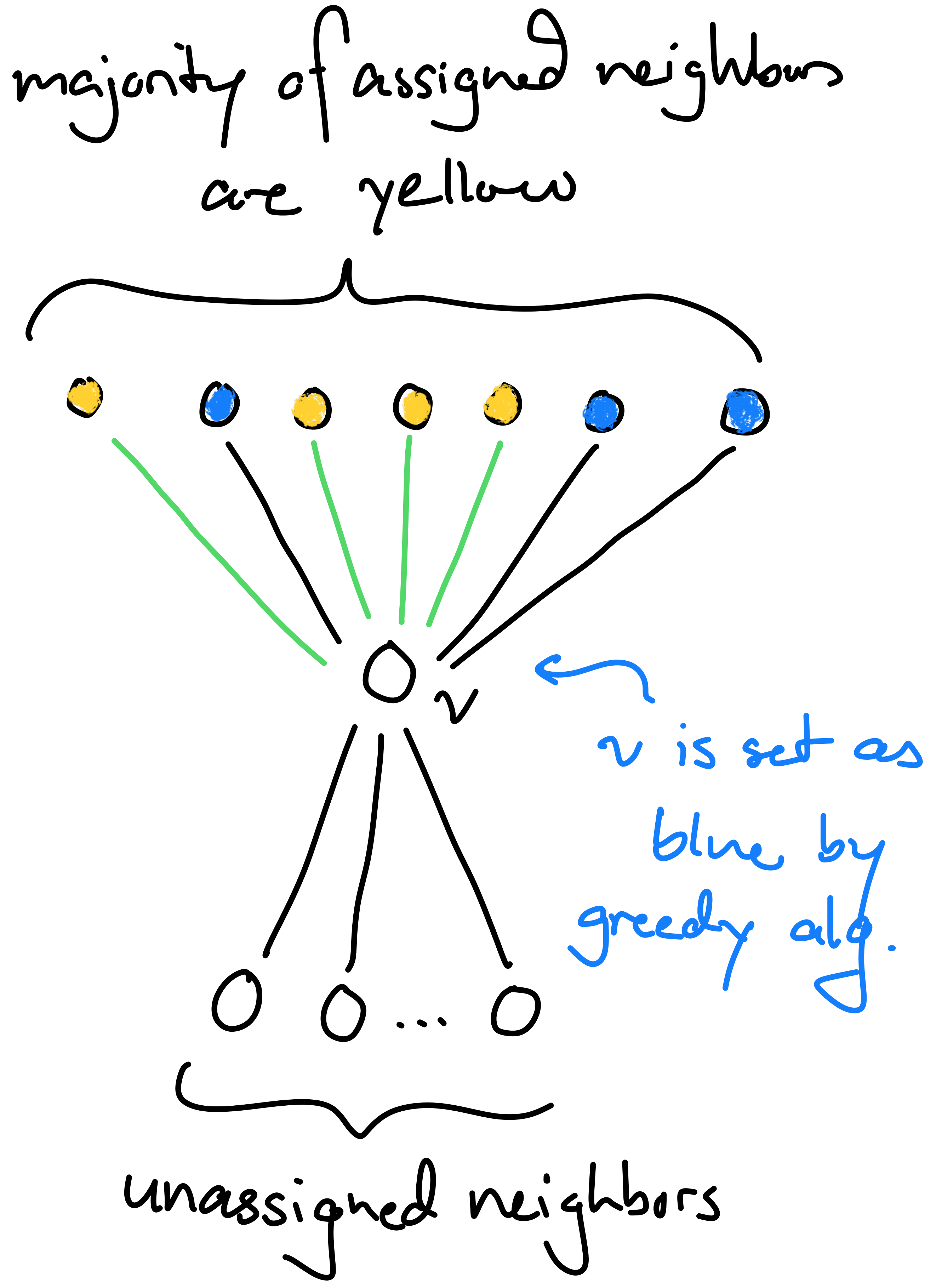
Proof of greedy algorithm optimality

- **Lemma:** The greedy algorithm always produces a cut crossing $\geq m/2$ edges.
- **Proof:**
  - Consider the set of edges E_v used to determine the color of vertex v. By choosing the color of v to be the opposite of the majority of neighbors, at least half of the edges of E_v cross the greedy cut.
  - Every edge is in exactly one set E_v where v is the later of its two vertices to be assigned a color.
  - Since $E = \bigsqcup_{v \in V} E_v$ and at least half of the edges of E_v cross the greedy cut, then at least half the edges of the E cross the greedy cut.



Proof of greedy algorithm optimality

- **Lemma:** The greedy algorithm always produces a cut crossing $\geq m/2$ edges.
- **Proof:**
  - Consider the set of edges E_v used to determine the color of vertex v. By choosing the color of v to be the opposite of the majority of neighbors, at least half of the edges of E_v cross the greedy cut.
  - Every edge is in exactly one set E_v where v is the later of its two vertices to be assigned a color.
  - Since $E = \bigsqcup_{v \in V} E_v$ and at least half of the edges of E_v cross the greedy cut, then at least half the edges of the E cross the greedy cut.



NP-completeness

- Max Cut is also a NP-complete problem.
- We strongly do not believe there is an efficient algorithm for Max Cut.
- The greedy algorithm always produces a $\geq 1/2$ factor of the optimal sized cut but cannot do better than this (due to our example).
    - Constitutes an approximation algorithm for the Max Cut problem.
    - [Goemans-Williamson]: Best known efficient approximation algorithm achieves a ~ 0.878 factor.
    - [UGC Conjecture]: Believed to be inefficient to approximate past this barrier