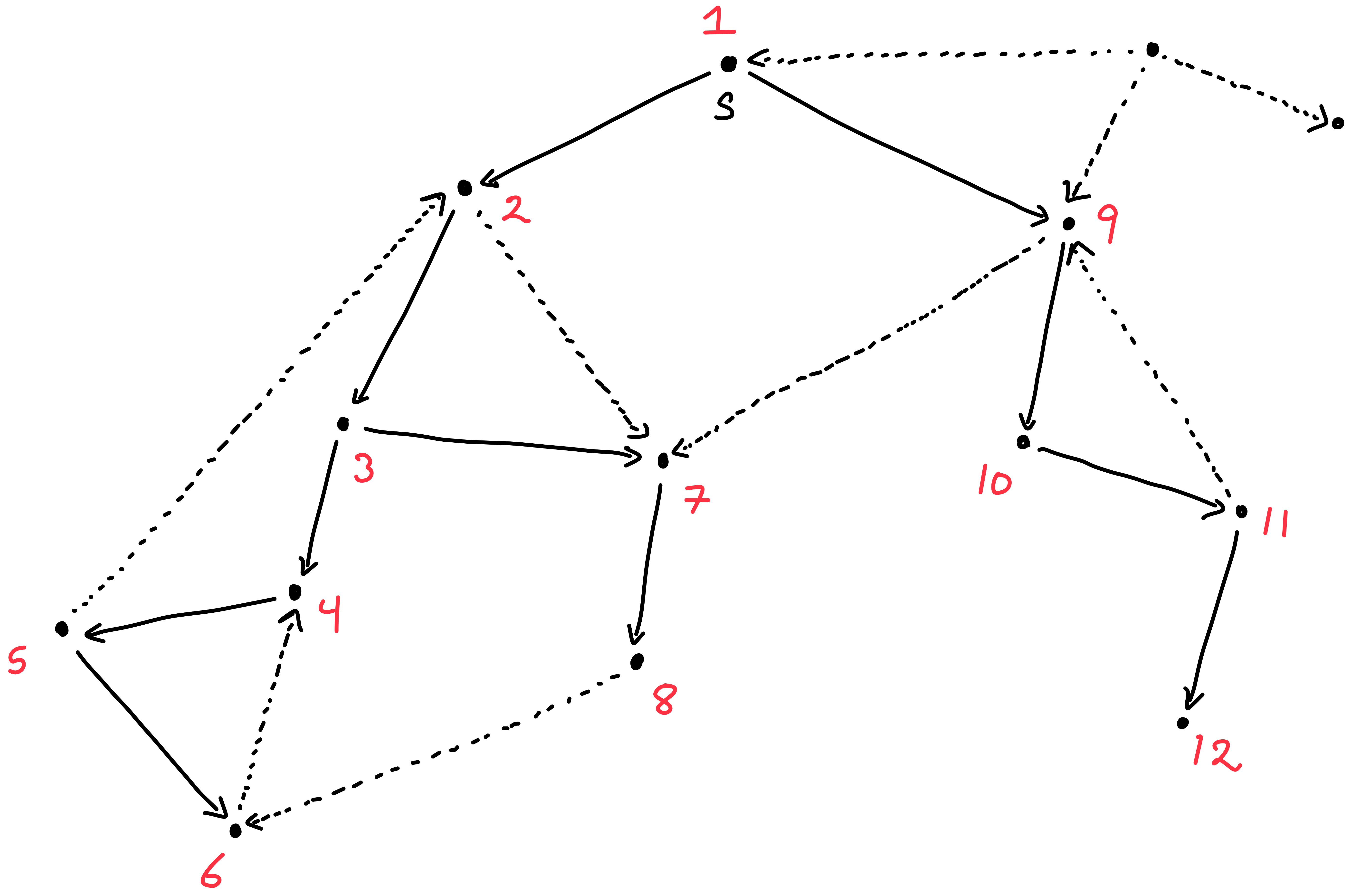
Lecture 4

Directed graphs and greedy algorithms

Chinmay Nirkhe | CSE 421 Winter 2026

Depth-first search on directed graphs

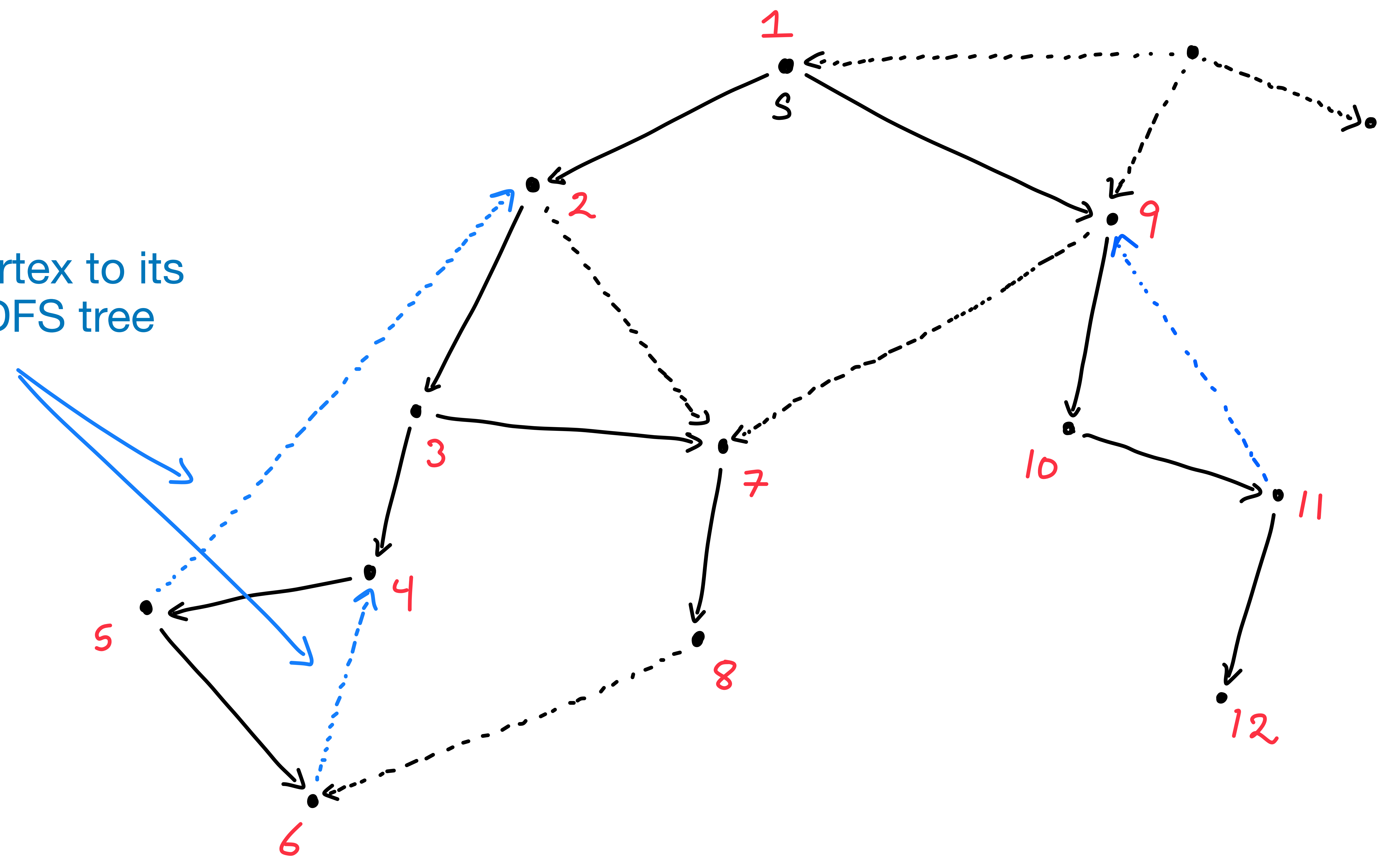
- Same as DFS on undirected graphs except we only add neighbor v if an edge points from $u \to v$.
- DFS starting from s will visit all vertices u reachable by a *directed* path $s \rightsquigarrow u$.



DFS edge nomenclature

Back edge

Connects vertex to its ancestor in DFS tree



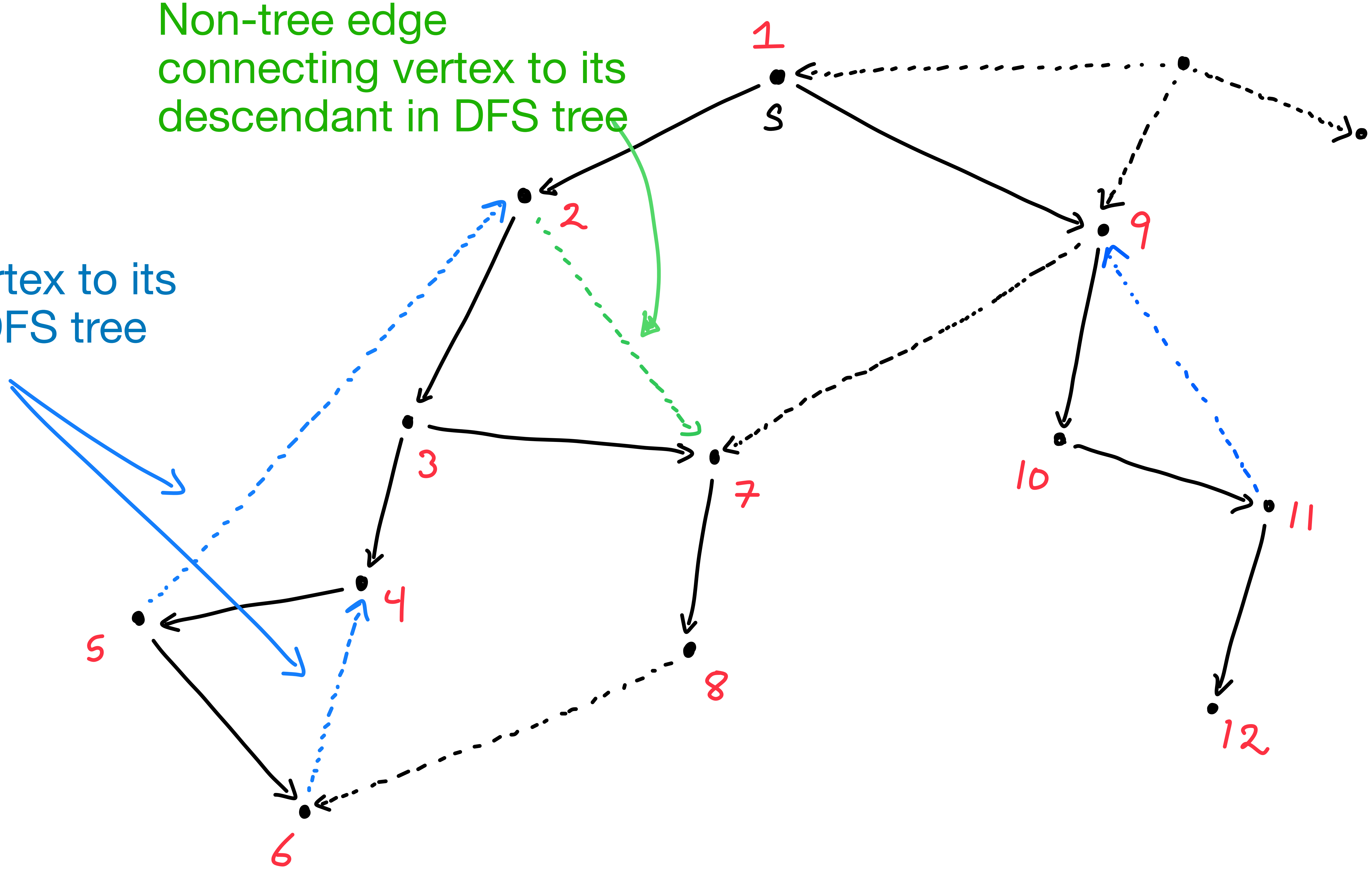
DFS edge nomenclature

Forward edge

Non-tree edge
connecting vertex to its
descendant in DFS tree

Back edge

Connects vertex to its
ancestor in DFS tree



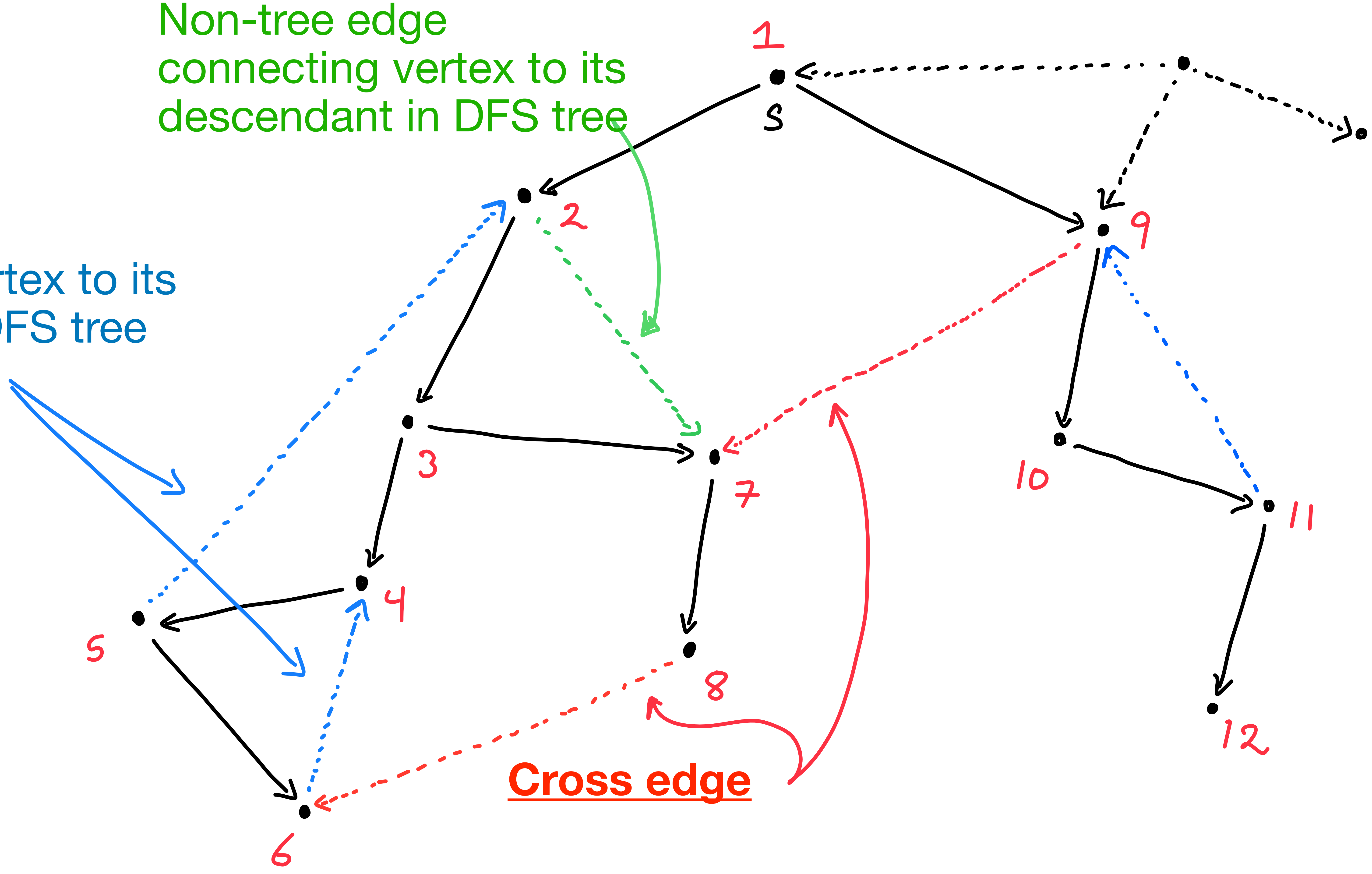
DFS edge nomenclature

Forward edge

Non-tree edge connecting vertex to its descendant in DFS tree

Back edge

Connects vertex to its ancestor in DFS tree



Cross edge

Connects vertices across branches. Always high → low in DFS tree

DFS edge nomenclature

Forward edge

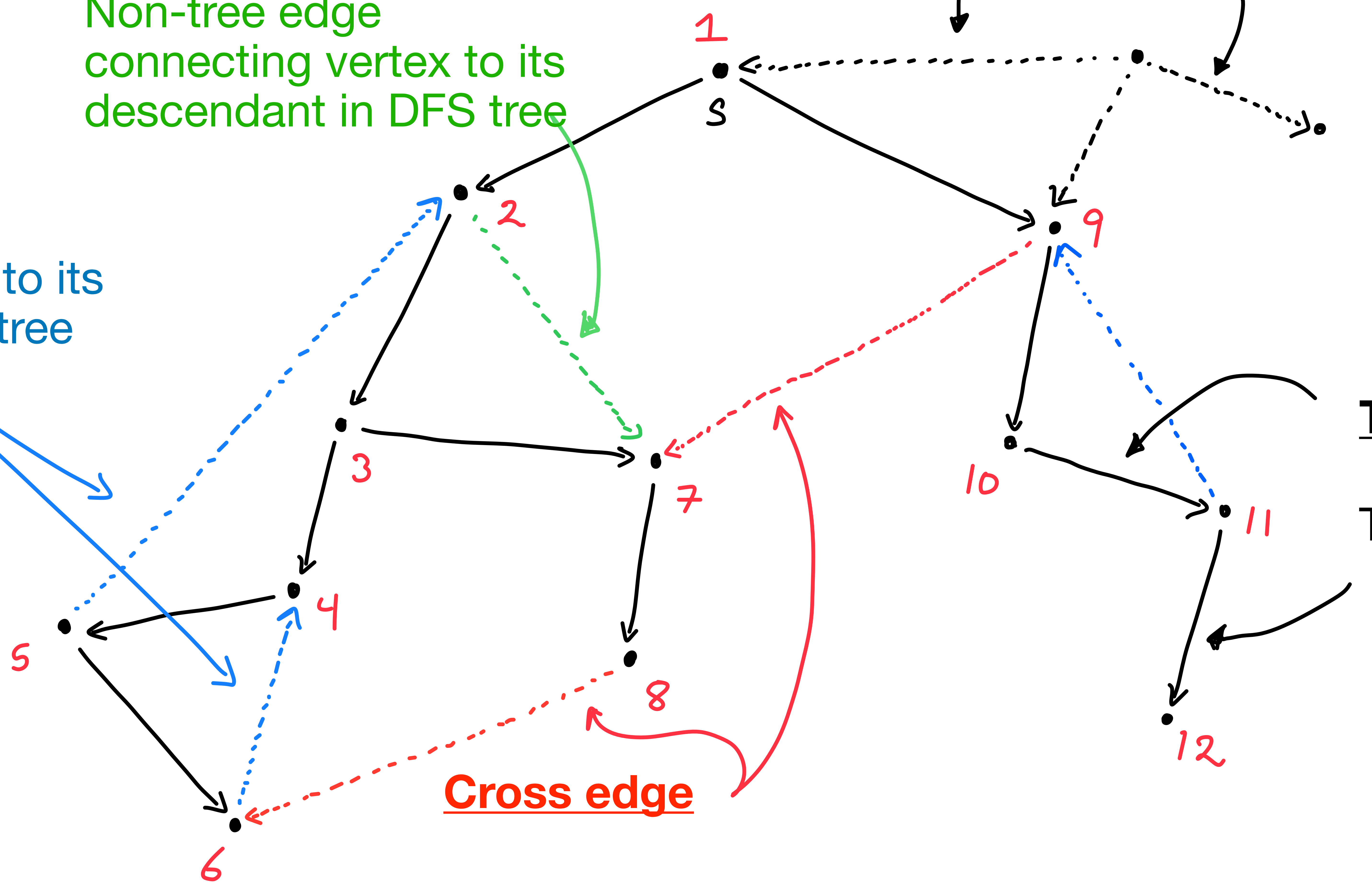
Non-tree edge connecting vertex to its descendant in DFS tree

Unvisited edge

The edges not visited during DFS

Back edge

Connects vertex to its ancestor in DFS tree



Tree edge

The edges in the DFS tree

Cross edge

Connects vertices across branches. Always high → low in DFS tree

DFS edge nomenclature

Unvisited edge

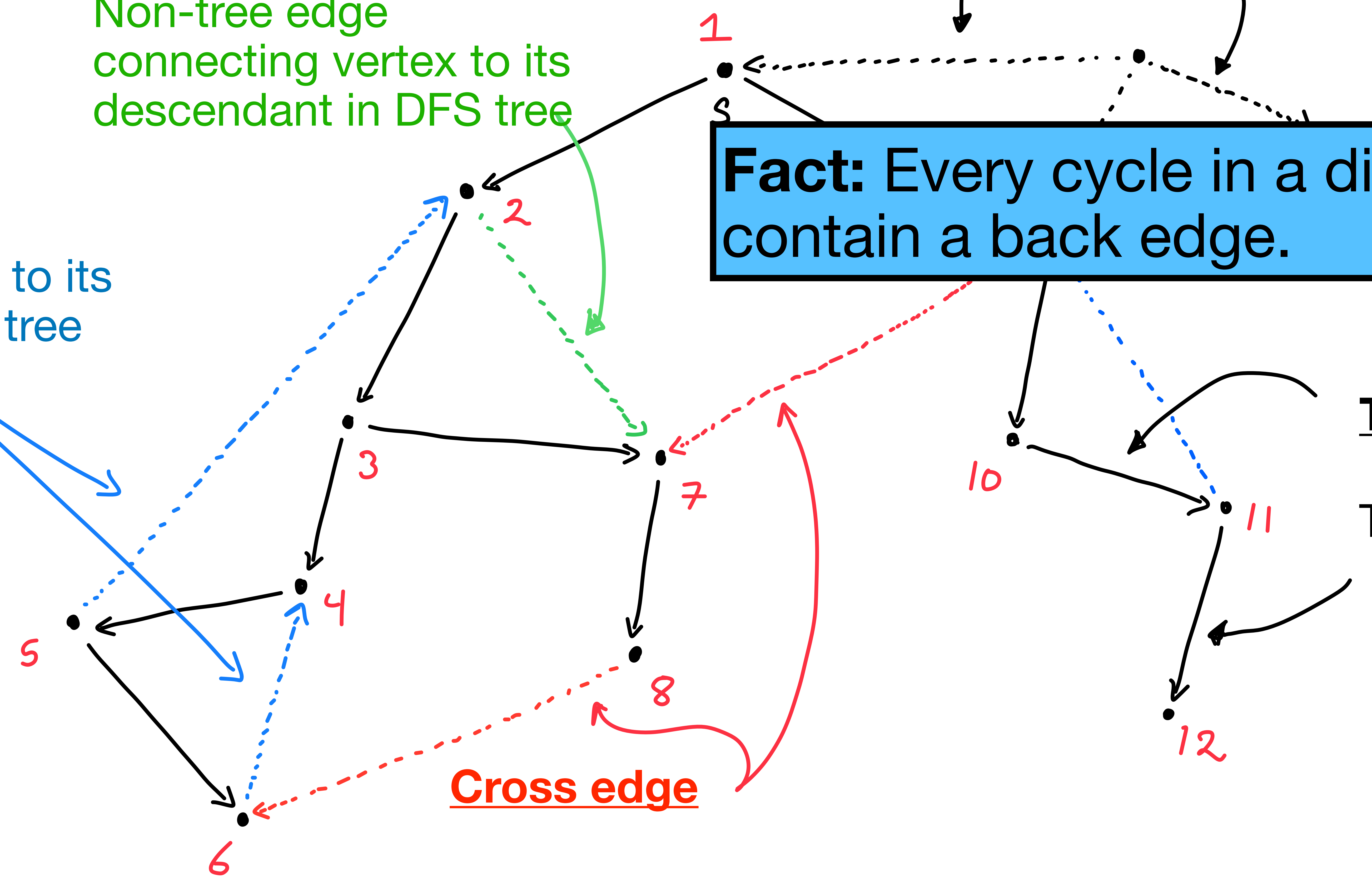
The edges not visited during DFS

Forward edge

Non-tree edge connecting vertex to its descendant in DFS tree

Back edge

Connects vertex to its ancestor in DFS tree



Fact: Every cycle in a directed graph must contain a back edge.

Tree edge

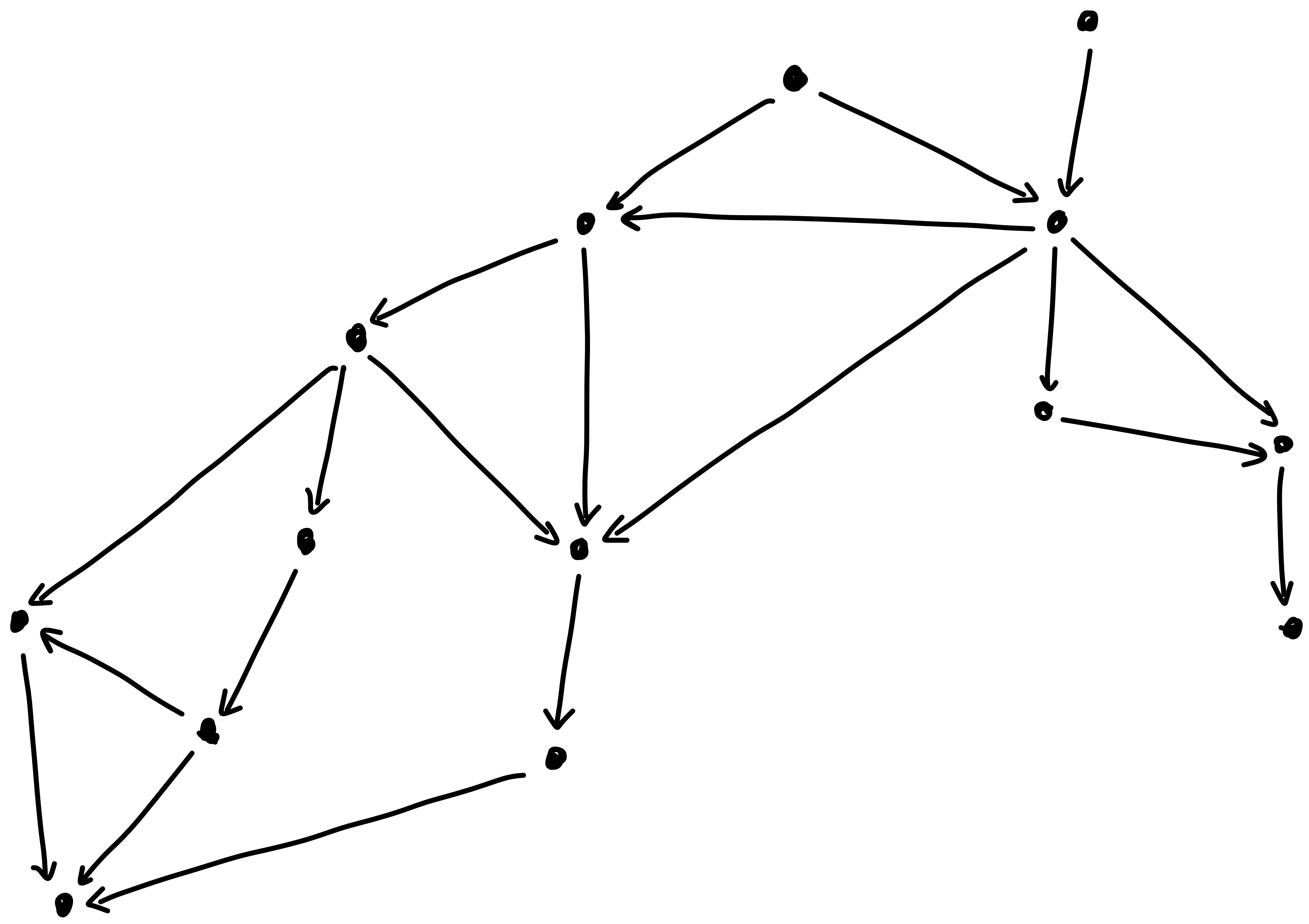
The edges in the DFS tree

Cross edge

Connects vertices across branches. Always high → low in DFS tree

Directed acyclic graphs

- A directed graph G is *acyclic* iff it has no directed cycles
- Also referred to as a “DAG”
- Advanced: There is a $O(n + m)$ algorithm (Kosaraju’s or Tarjan’s) for shrinking the “strongly connected components” of a general graph to convert it into a DAG

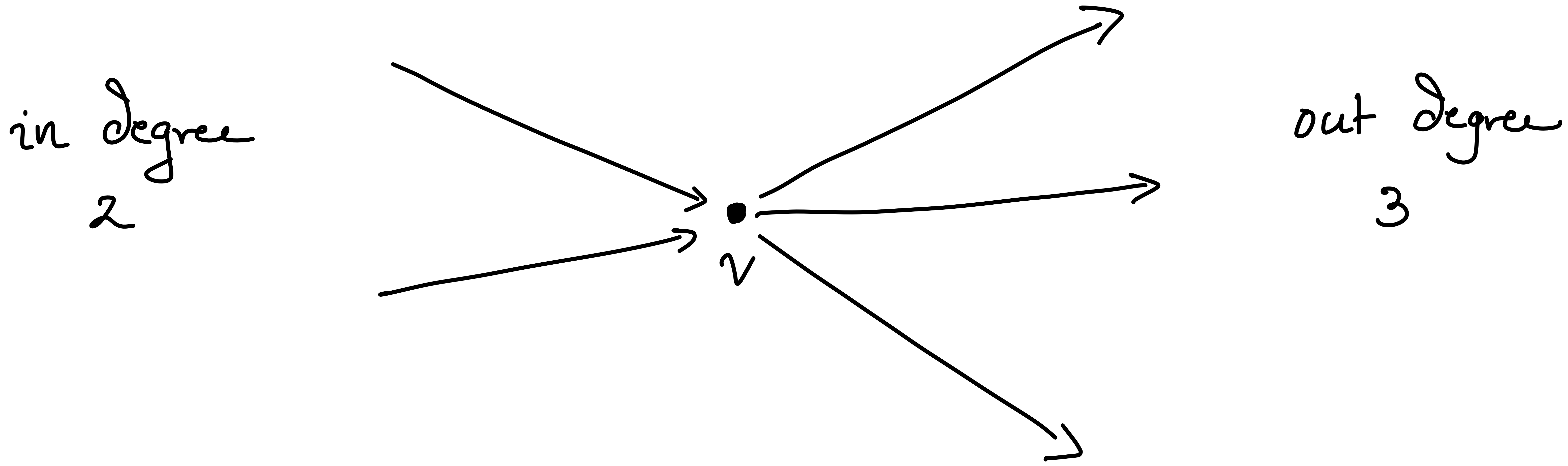


Topological sorting of graphs

- **Input:** a directed acyclic graph DAG $G = (V, E)$
- **Output:** An injective numbering $N : V \hookrightarrow \{1, \ldots, n\}$ such that edges only go from lower numbered to higher numbered vertices.

  i.e. for $u \rightarrow v$, we must have $N(u) < N(v)$.
- **Applications**
  - Vertices represents tasks and edges represent prerequisites
  - Topological sorts gives a sequential ordering for how to solve the system
- For general graphs, generate DAG by shrinking SCCs and then process SCCs in the order given by topological sort.

In-degree and out-degree



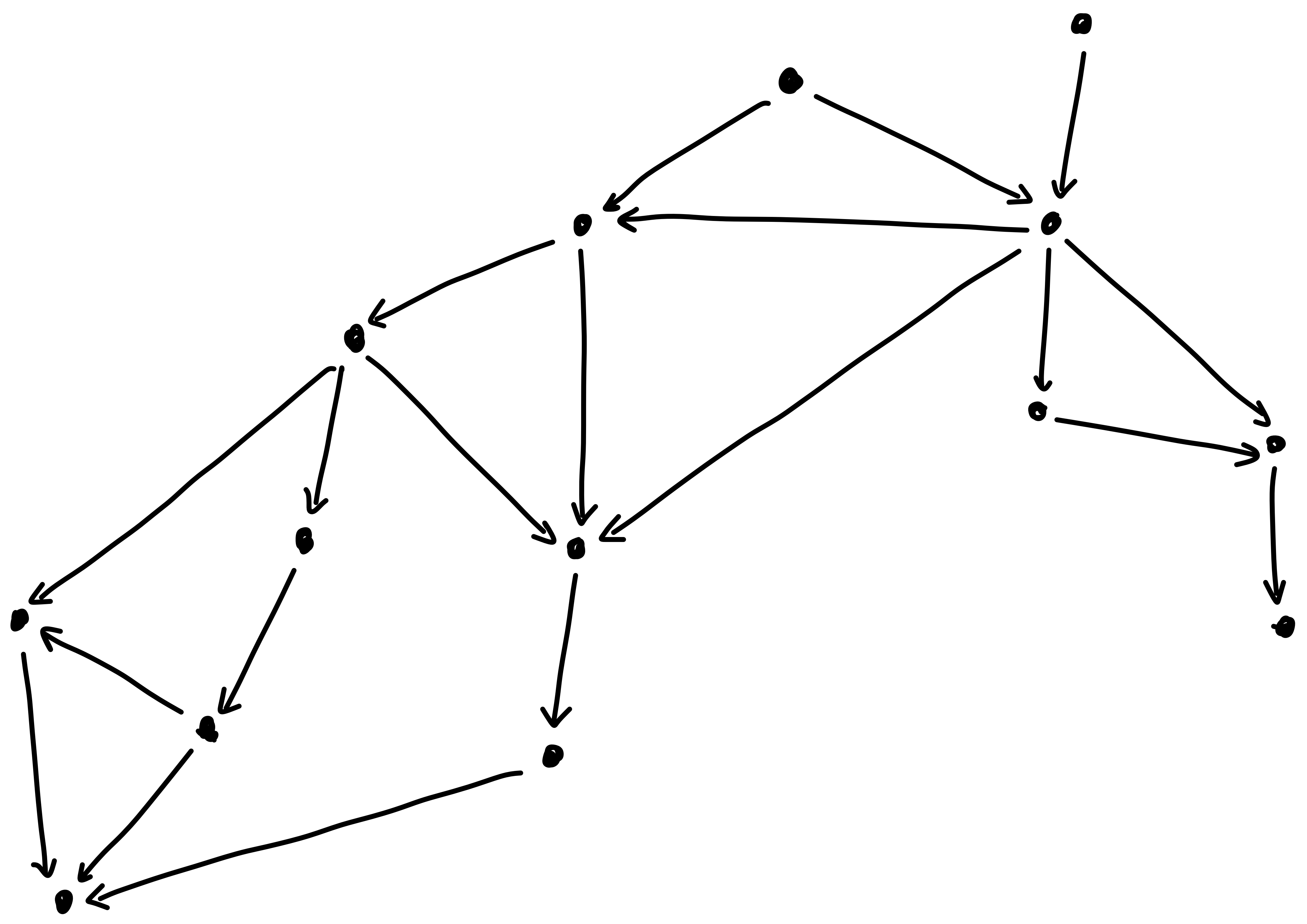
In-degree zero vertices

- **Claim:** Every DAG has at least one vertex of in-degree 0.
- **Proof:**
  - Assume every vertex has in-degree ≥ 1.
  - Starting with any vertex v pick an in-edge $u \to v$ and go in reverse to u. Repeat.
  - Since there are only n vertices, eventually a vertex will be repeated. This means there is a cycle, a contradiction.

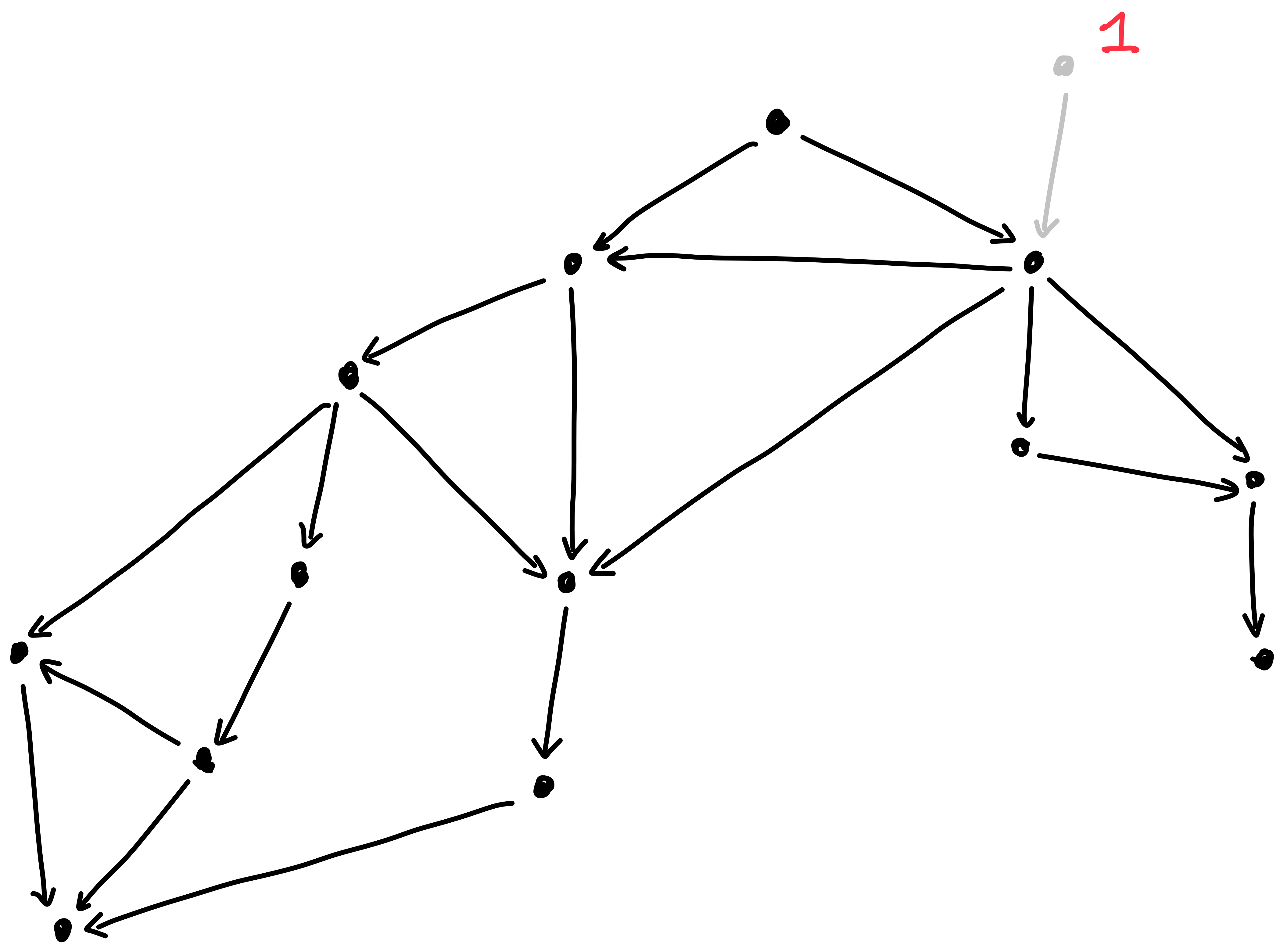
Algorithm for topological sort

- Any vertex v_1 of in-degree 0 can be numbered as 1
- Can run DFS starting from v_1
- Alternative simpler idea:
  - If we remove v_1 and assign $N(v_1) = 1$, then the rest is still a DAG
  - Then, there is a new vertex v_2 of in-degree 0
  - Repeat, until all vertices are exhausted

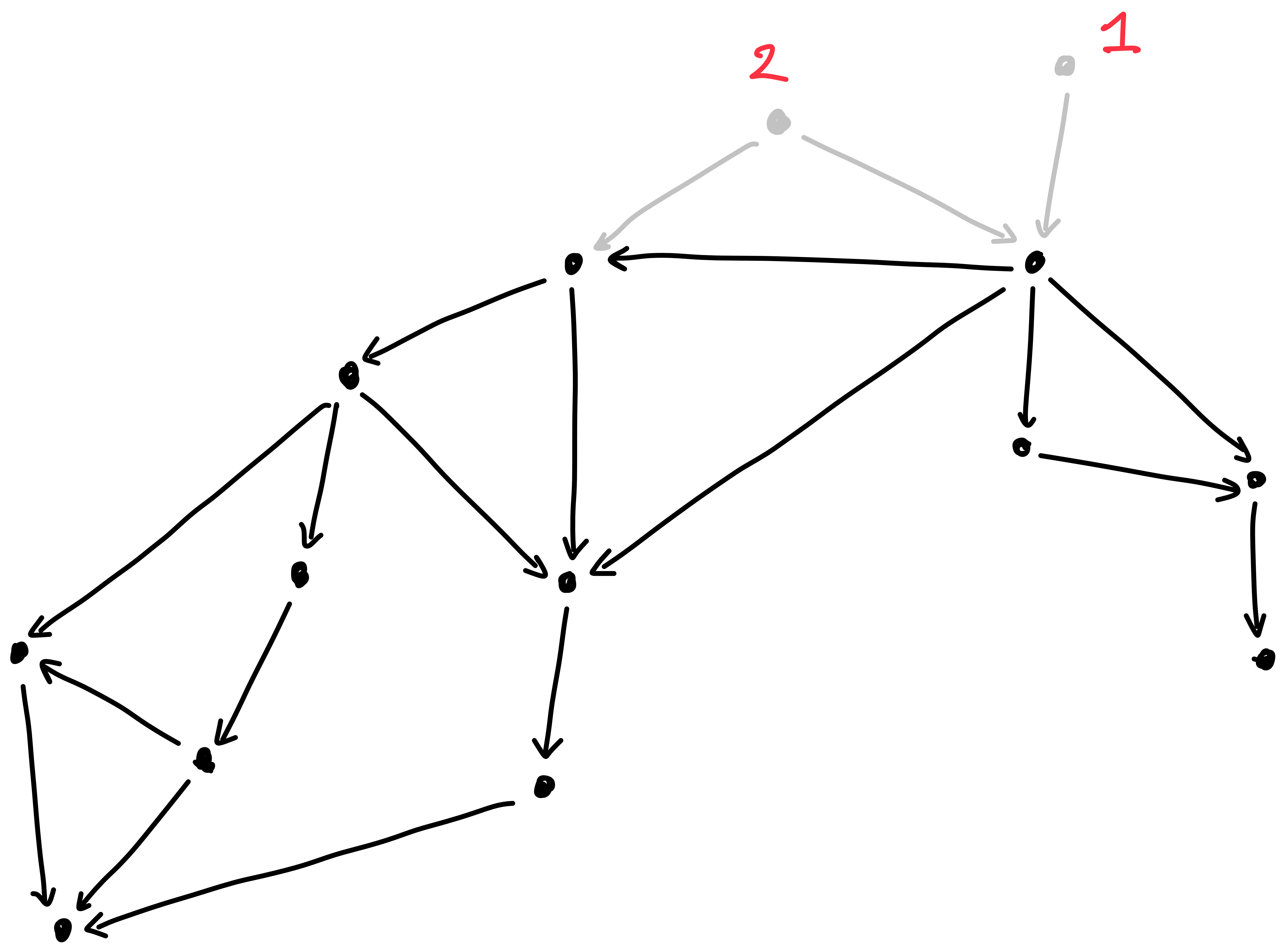
Implementing topological sort



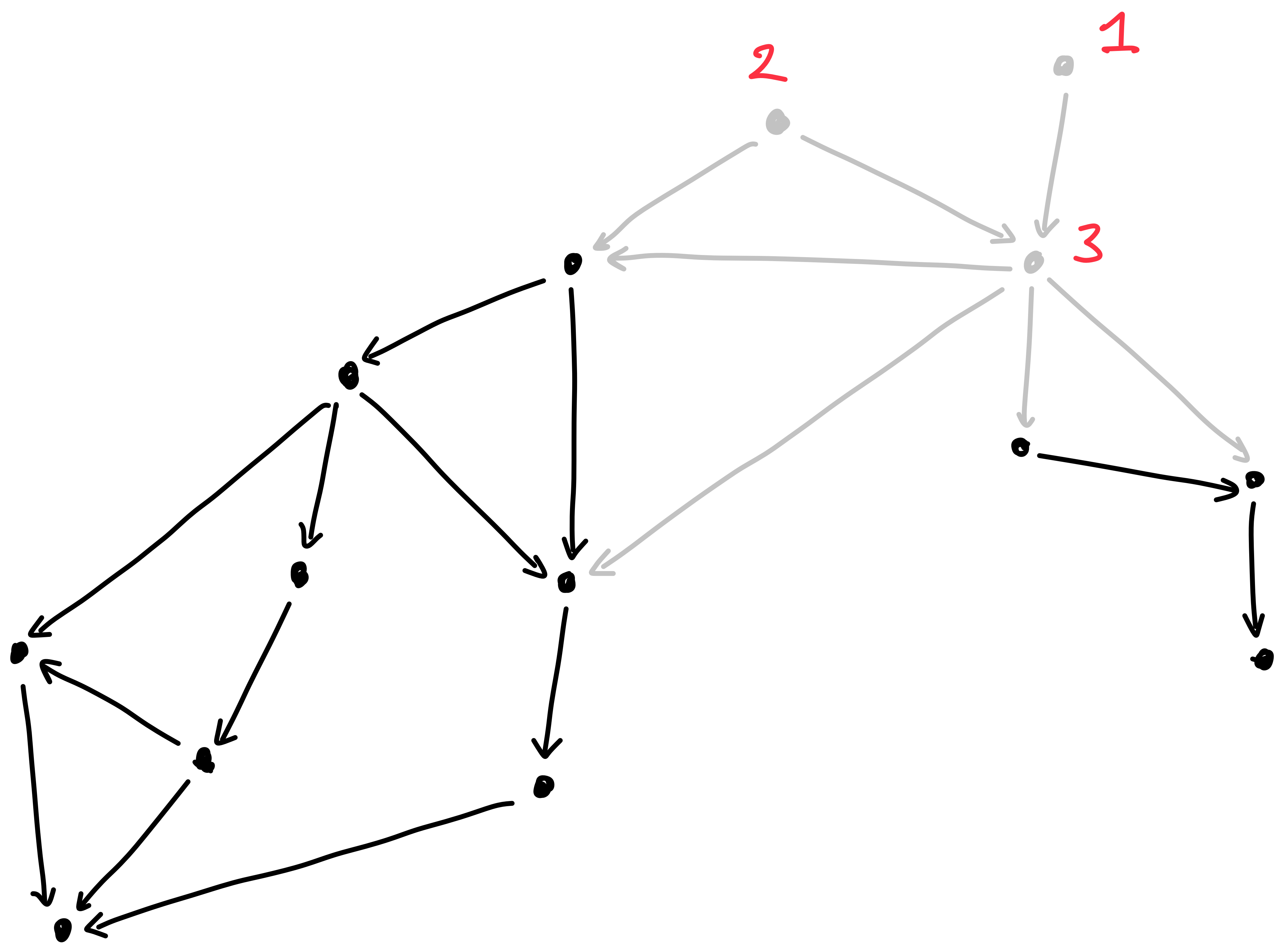
Implementing topological sort



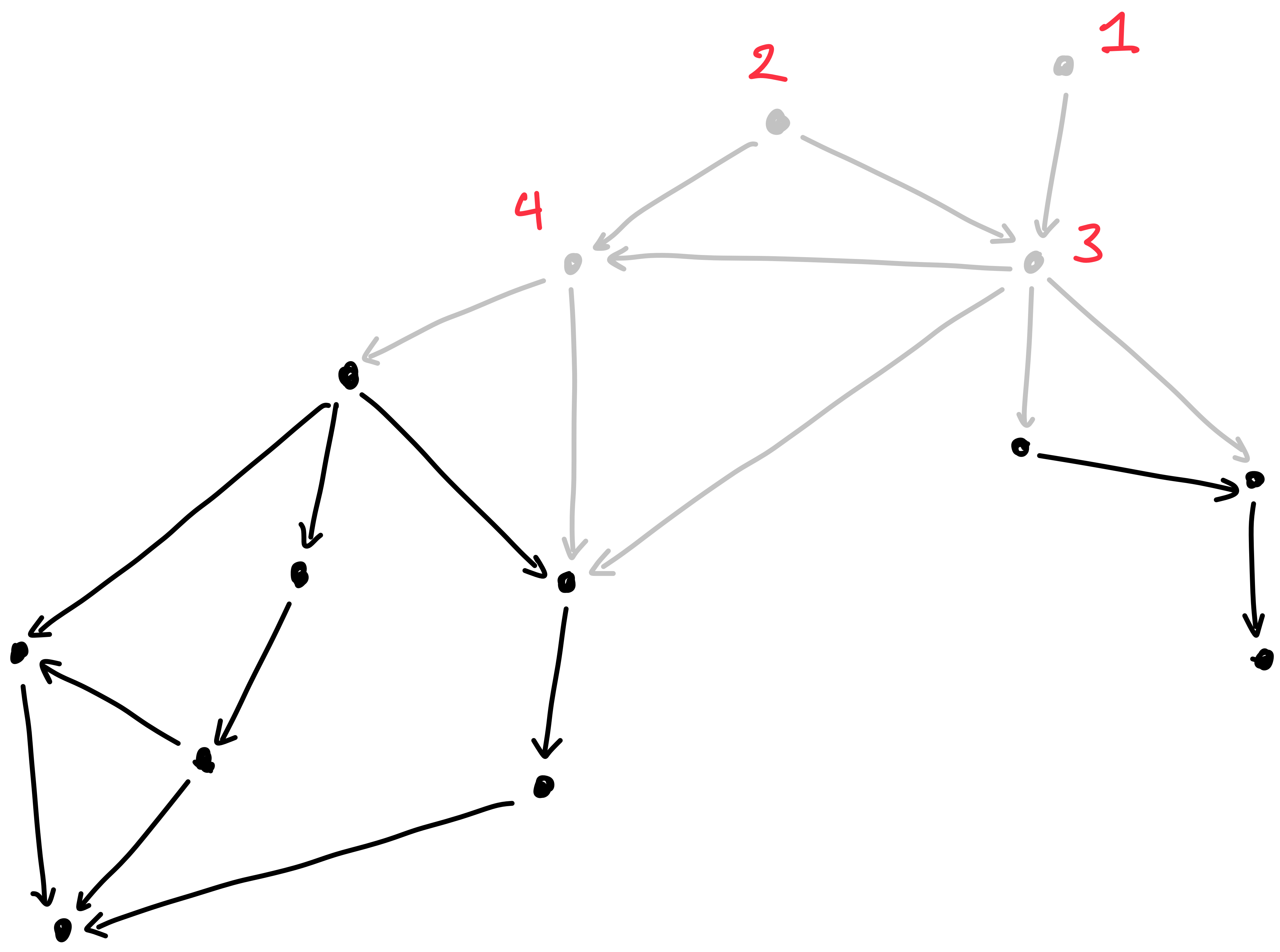
Implementing topological sort



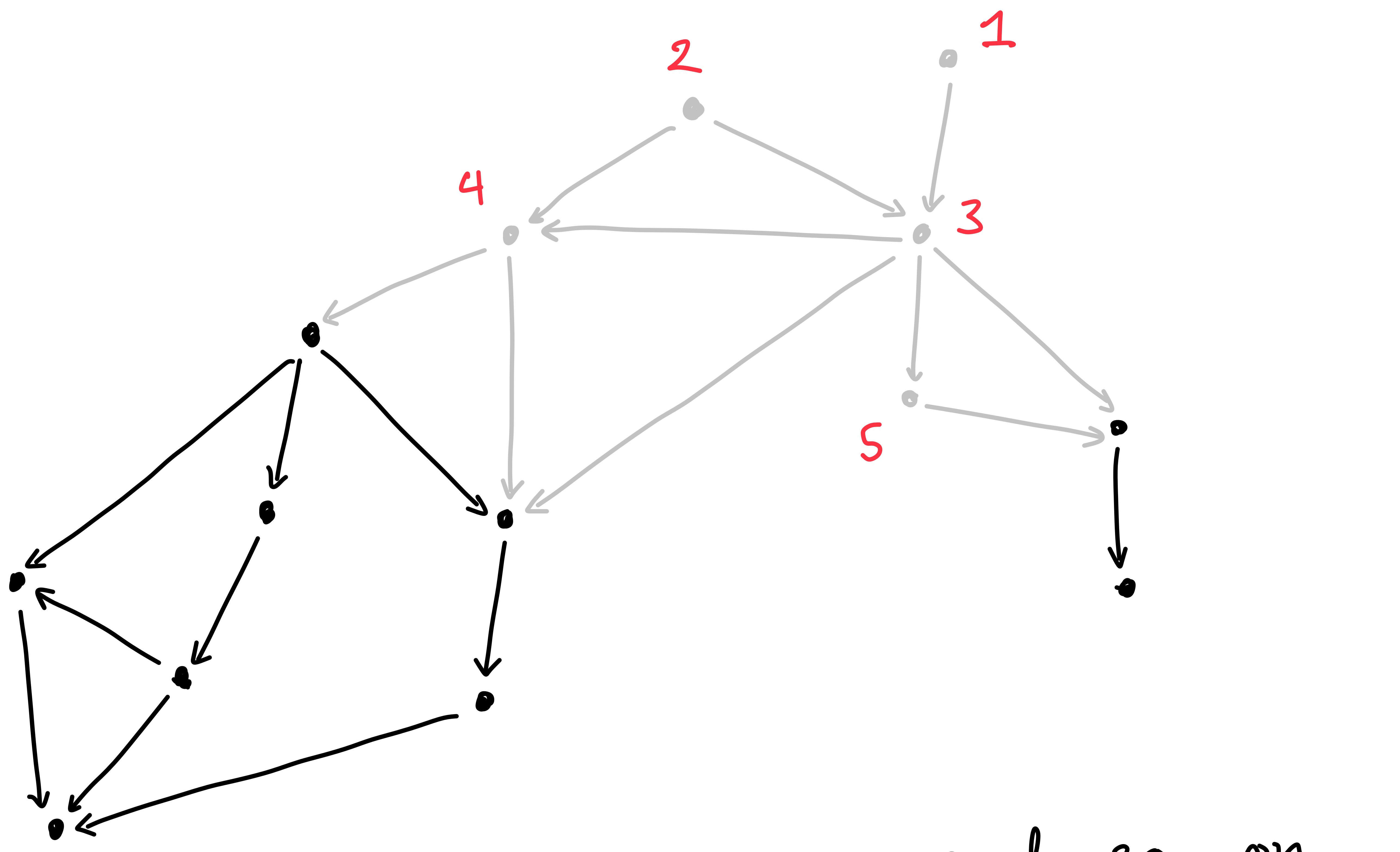
Implementing topological sort



Implementing topological sort



Implementing topological sort



Implementing topological sort

- Issue is finding the next vertex that has in-degree 0. Can be algorithmically slow.

- Observe that when we remove the vertex v_j, the in-degree of only the out-neighbors of v_j will decrease.

Implementing topological sort

- **Algorithm:**
  - Iterate through all vertices and set $d(v) =$ in-degree of each vertex. Initialize queue Q with vertices such that $d(v) = 0$. Set $j \leftarrow 1$.
  - While Q is non-empty, pop vertex u off queue
    - Set $N(u) \leftarrow j$. Increment $j \leftarrow j + 1$.
    - Decrease $d(v) \leftarrow d(v) - 1$ for every nbhr. v s.t. $u \rightarrow v$. If $d(v) = 0$, add v to Q.
  - **Runtime:** Each edge is visited only once. So $O(n + m)$ time.

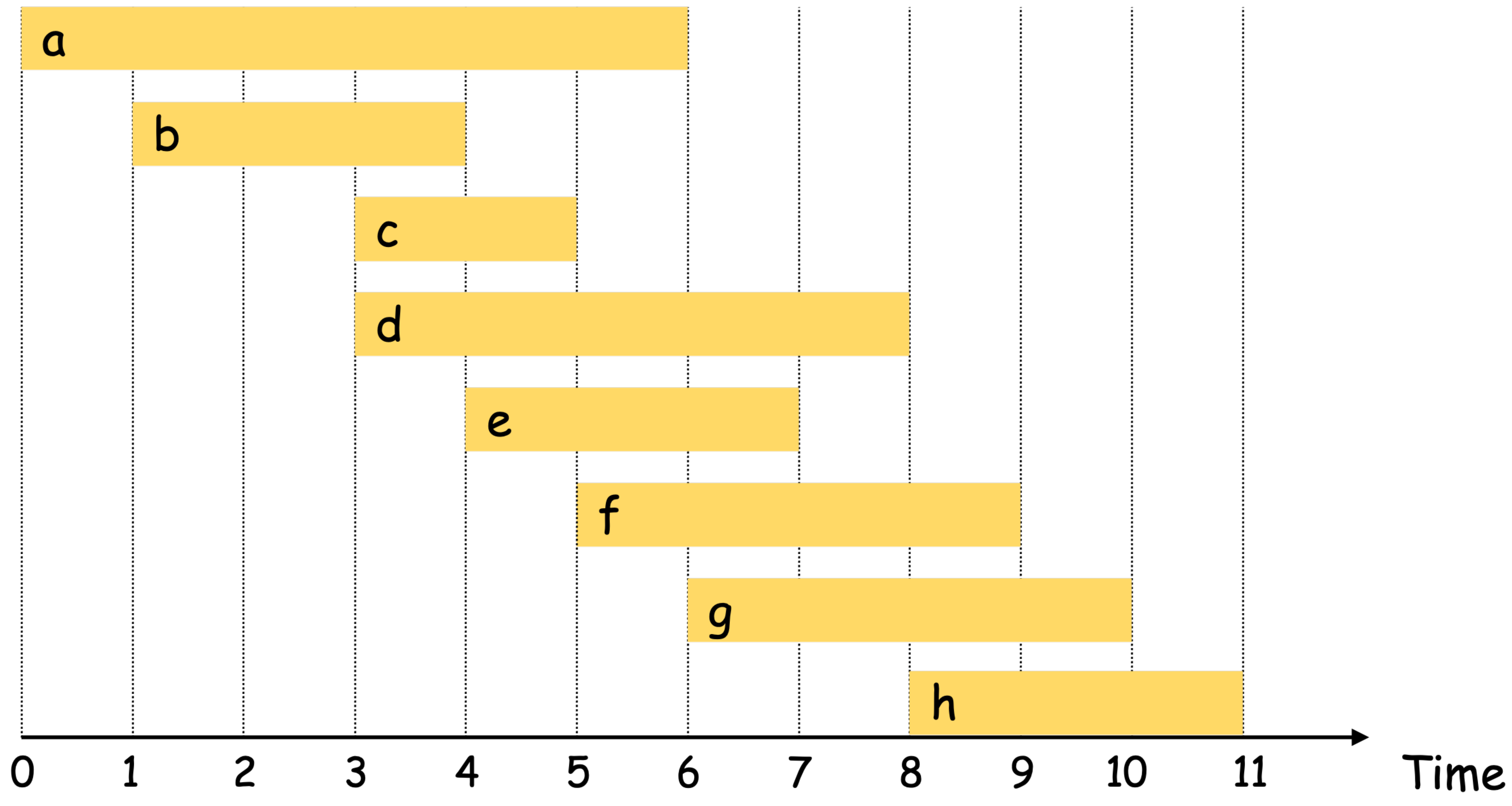
Greedy algorithms

An introduction to algorithms

- Goal is to understand *how* to analyze and *design* algorithms
    - To understand how small changes have big effects on outcomes
    - Build a repertoire of techniques for designing algorithms
    - Identifying when to use which family of algorithms
- Course is structured by teaching various families of algorithms
    - Section and problem sets will cover example instantiations pertinent to that week
    - Midterms and finals will have problems but won't say which family of algorithms to use

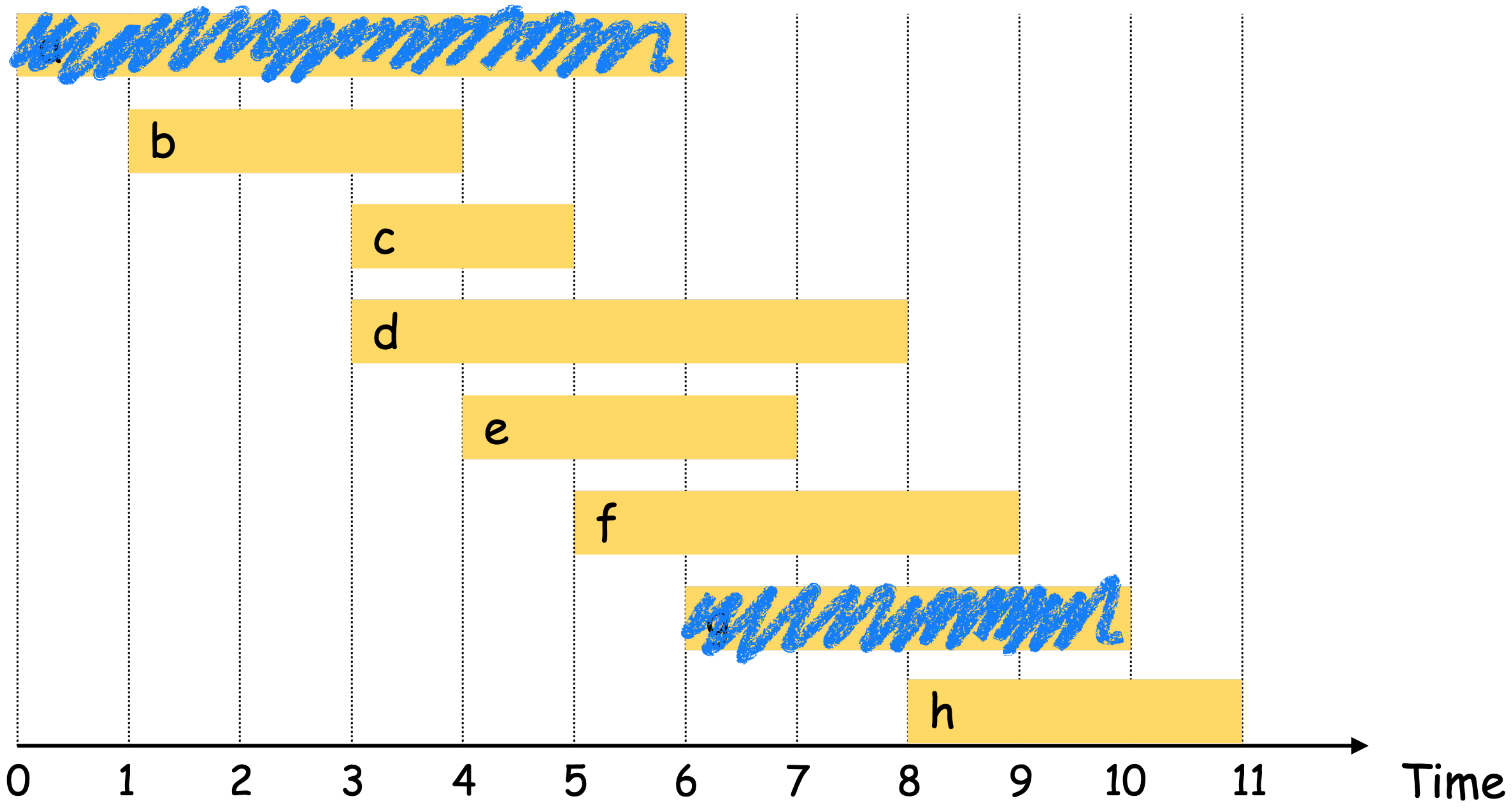
Interval scheduling

- **Input:** start and end times (s_i, t_i) for $i = 1,\ldots,n$ for n "jobs"
- **Output:** A maximal set of mutually compatible jobs



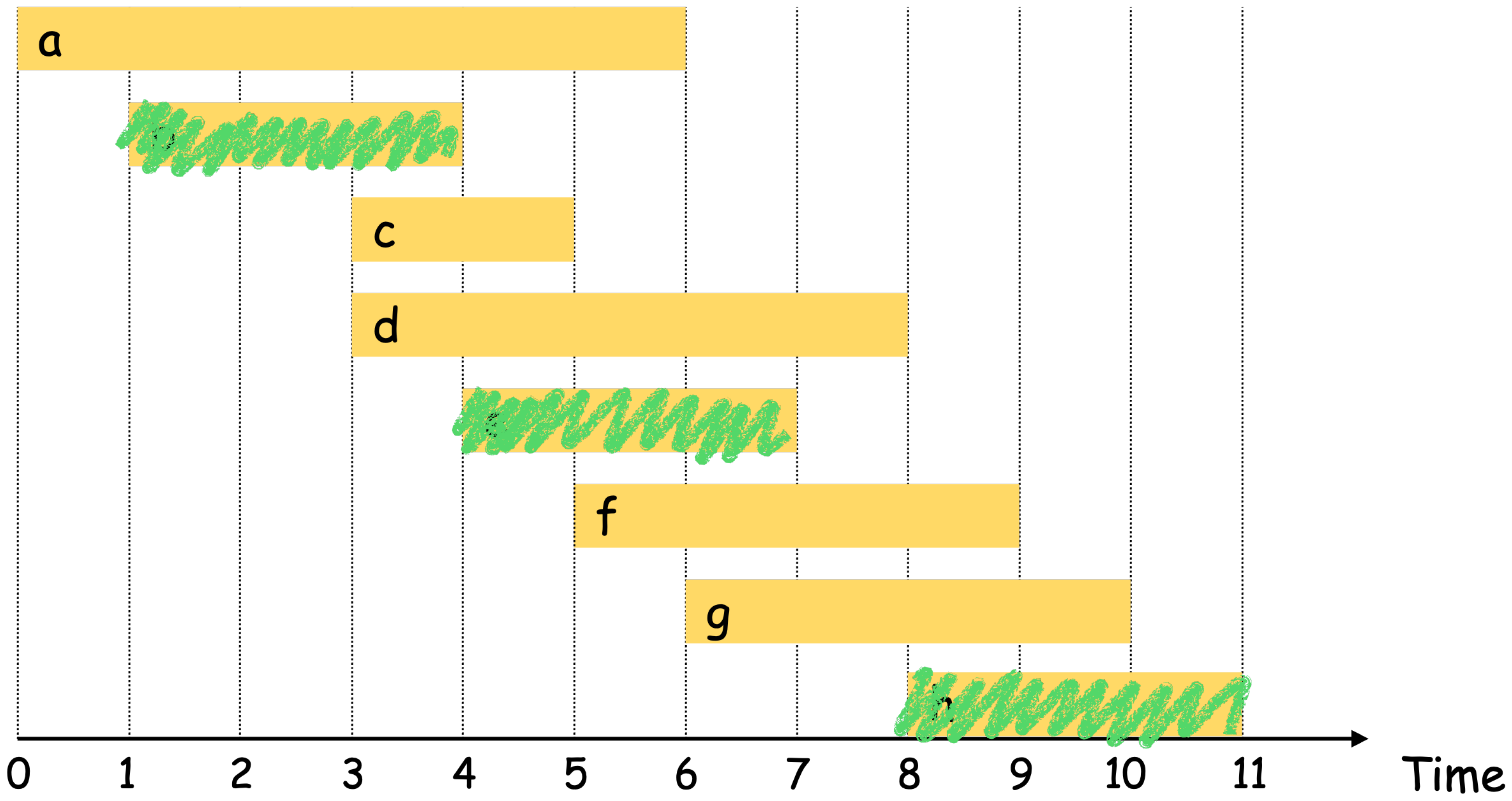
Interval scheduling

- **Input:** start and end times (s_i, t_i) for $i = 1, \ldots, n$ for n "jobs"
- **Output:** A maximal set of mutually compatible jobs



Interval scheduling

- **Input:** start and end times (s_i, t_i) for $i = 1, \ldots, n$ for n "jobs"
- **Output:** A maximal set of mutually compatible jobs



Interval scheduling

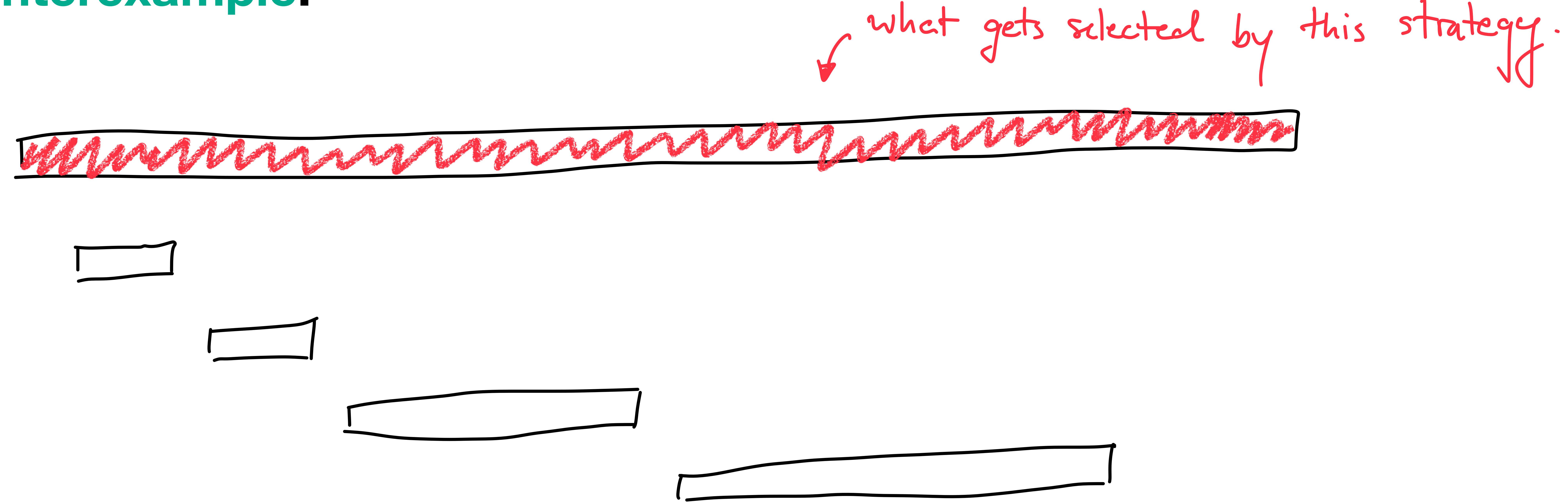
- **Input**: start and end times (s_i, t_i) for $i = 1, \ldots, n$ for n “jobs”
- **Output**: A maximal set of mutually compatible jobs
- **Algorithm**:
  - **Brute-force**: Iterate through all 2^n possible selections. Check in $O(n)$ time if selection is (a) feasible and (b) maximal.
  - **Greedy**: Decide a selection criteria and select jobs accordingly.

The principle of greedy algorithms

- Solving the *optimization problem* will require making many decisions (such as whether to include or not a job in the schedule)
- In a greedy algorithm, we make each decision locally without looking as to how it will effect future decisions
- Not every greedy criteria for making decisions works
  - It's not obvious which criteria will work
  - We will focus on methods for proving that greedy algorithms do work
- When a greedy decision is made, it will be *provably* optimal

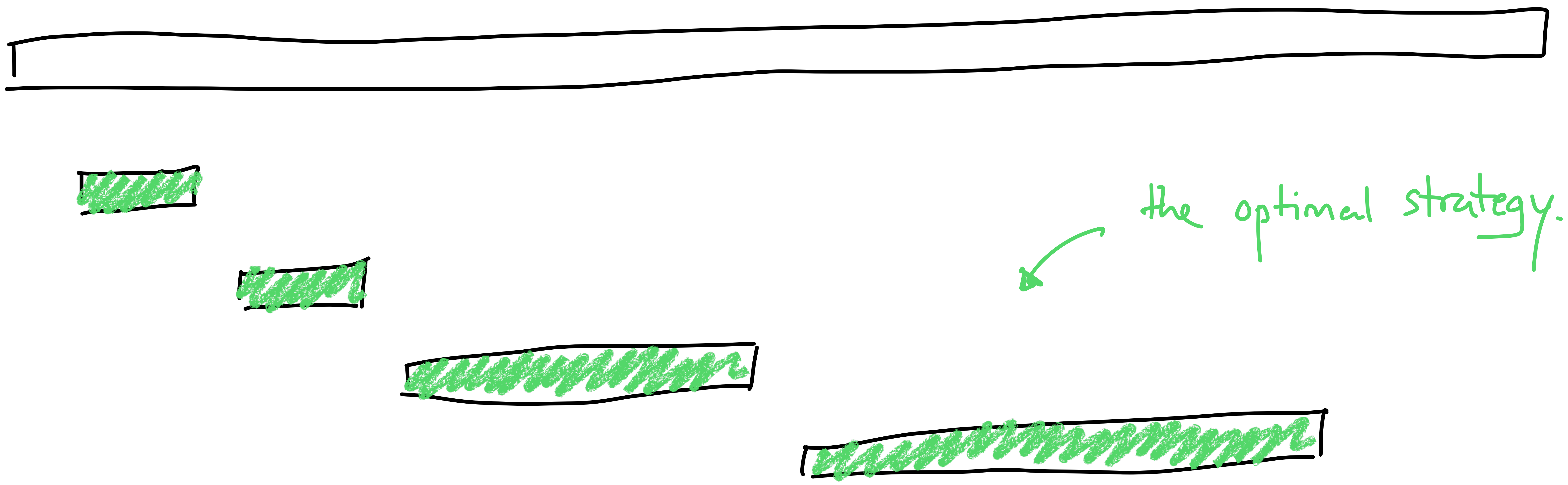
Greedy algorithms for interval scheduling

- **Algorithm:** Select the job with earliest start time s_i of jobs not selected.
- **Counterexample:**



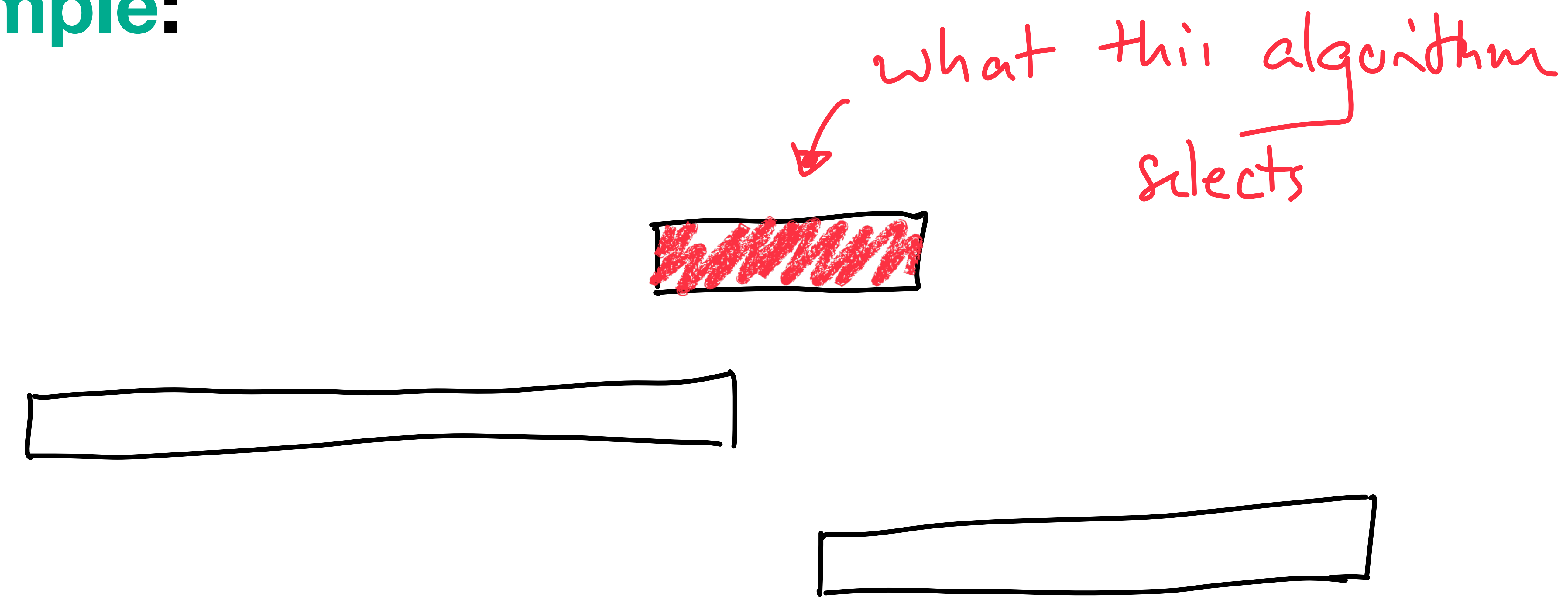
Greedy algorithms for interval scheduling

- **Algorithm:** Select the job with earliest start time s_i of jobs not selected.
- **Counterexample:**



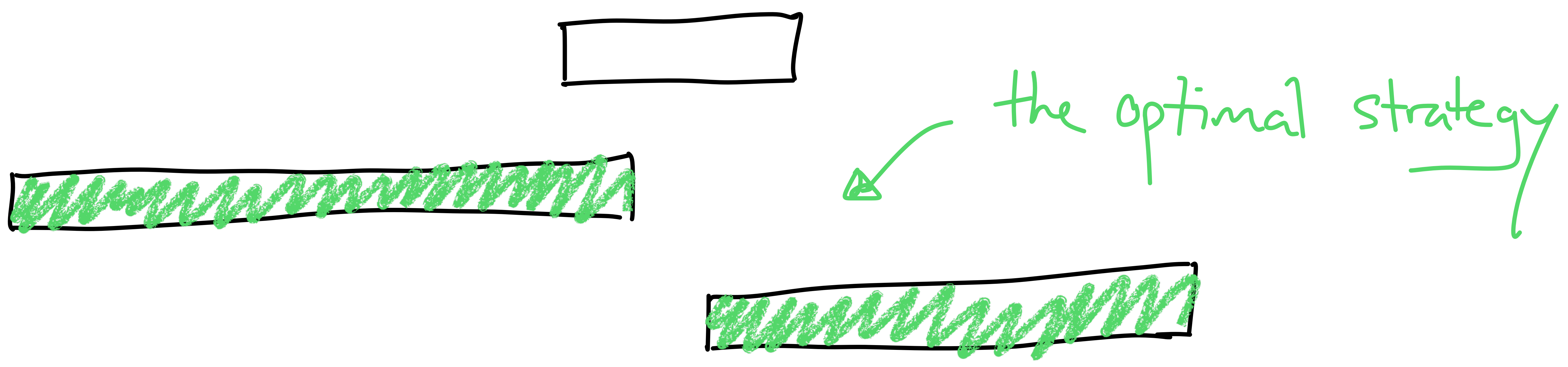
Greedy algorithms for interval scheduling

- **Algorithm:** Select the job with shortest duration $t_i - s_i$ of jobs not selected.
- **Counterexample:**



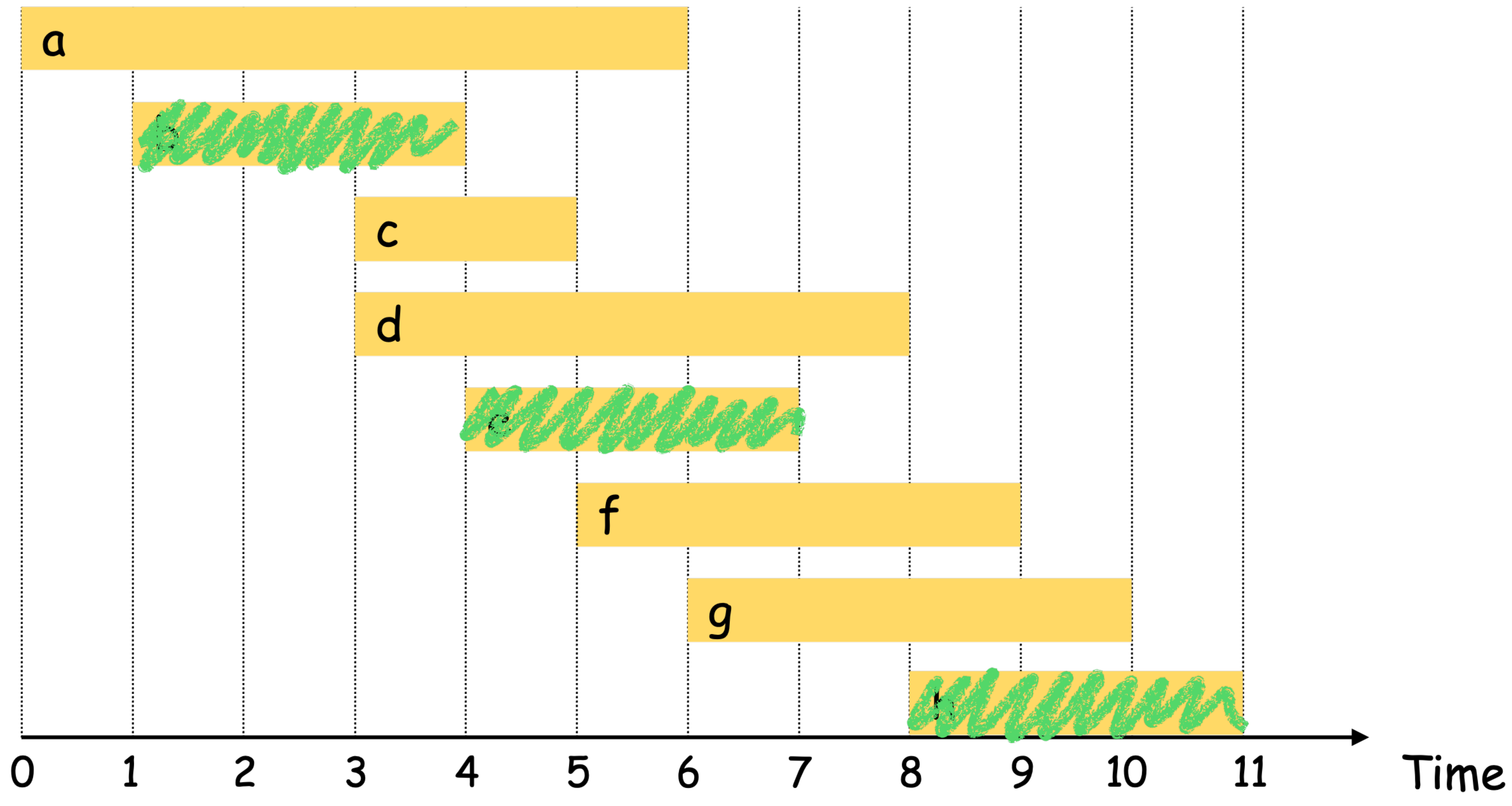
Greedy algorithms for interval scheduling

- **Algorithm:** Select the job with shortest duration $t_i - s_i$ of jobs not selected.
- **Counterexample:**



Greedy algorithms for interval scheduling

- **Algorithm:** Select the job with earliest ending t_i of jobs not selected and feasible.
- **Proof of**
- **Example:**



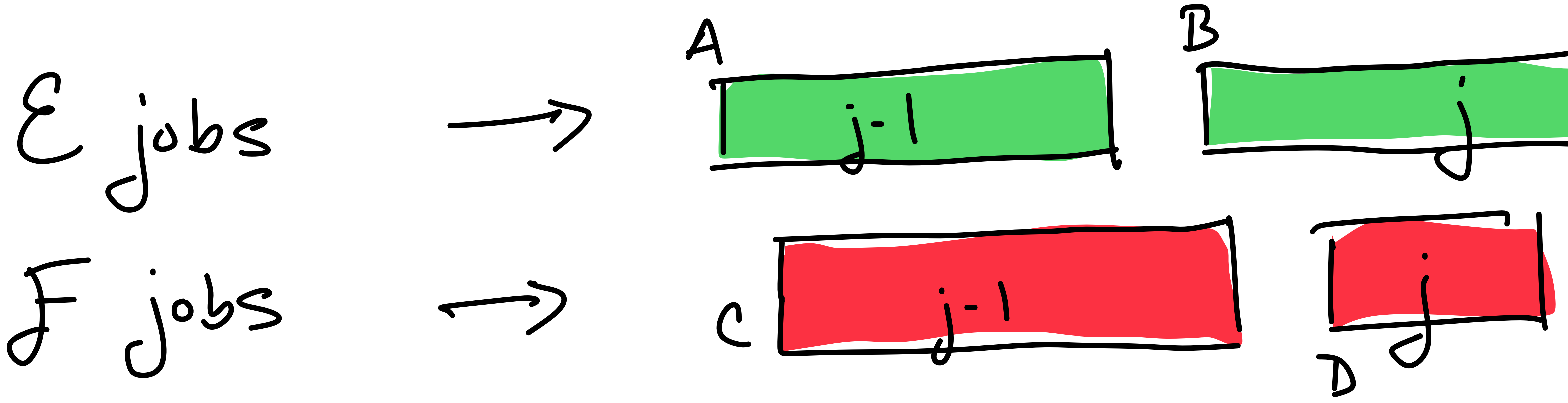
Greedy algorithms for interval scheduling

- **Algorithm:** Select the job with earliest ending t_i of jobs not selected and feasible.
- **Proof of correctness:**
  - Let $\mathscr{E} \subseteq [n]$ be the set of jobs selected by algorithm and $\mathscr{F} \subseteq [n]$ be any other *feasible* set of jobs.
  - **Claim:** The j-th job in $\mathscr{E}$ ends at least before the j-th job in $\mathscr{F}$ ends.

Greedy algorithms for interval scheduling

- **Claim:** The j-th job in $\mathscr{E}$ ends at least before the j-th job in $\mathscr{F}$ ends.
- **Proof:**

Assume (for contradiction) that this is false and let j be the smallest counterexample. Picture:



Contradicts the def. of $\mathscr{E}$ as job D isn't selected but ends before job B.

Greedy algorithms for interval scheduling

- **Algorithm:** Select the job with earliest ending t_i of jobs not selected.
- **Proof of correctness:**
  - Let $\mathscr{E} \subseteq [n]$ be the set of jobs selected by algorithm and $\mathscr{F} \subseteq [n]$ be any other *feasible* set of jobs.
  - **Claim:** The j-th job in $\mathscr{E}$ ends at least before the j-th job in $\mathscr{F}$ ends.
  - If $\mathscr{F}$ had more jobs than $\mathscr{E}$, we could have added the final job of $\mathscr{F}$ to $\mathscr{E}$, a contradiction to the def. of $\mathscr{E}$.
  - So, $\mathscr{E}$ has at least as many jobs as $\mathscr{F}$. True for all feasible $\mathscr{F}$, proving optimality.

Greedy algorithms for interval scheduling

- **Input**: start and end times (s_i, t_i) for $i = 1, \ldots, n$ for n "jobs"
- **Output**: A maximal set of mutually compatible jobs
- **Algorithm:** Select the job with earliest ending t_i of jobs not selected.
  - **Details:** Sort the jobs by earliest end time t_i. Keep track of T the current end time over all selected jobs. Add new job (s_i, t_i) if $s_i \geq T$ and update $T \leftarrow t_i$.
  - **Runtime:** Sorting + linear time to create list of jobs. $O(n \log n) + O(n) = O(n \log n)$.

The principle of greedy algorithms

- Solving the *optimization problem* will require making many decisions (such as whether to include or not a job in the schedule)
- In a greedy algorithm, we make each decision locally without looking as to how it will effect future decisions
- Not every greedy criteria for making decisions works
  - It's not obvious which criteria will work
  - We will focus on methods for proving that greedy algorithms do work
- When a greedy decision is made, it will be *provably* optimal

A writeup for Interval Scheduling

- **Input**: start and end times (s_i, t_i) for $i = 1, \ldots, n$ for n “jobs”
- **Output**: A maximal set of mutually compatible jobs
- **Algorithm:** Select the job with earliest ending t_i of jobs not selected.
  - **Details:** Sort the jobs by earliest end time t_i. Keep track of T the current end time over all selected jobs. Add new job (s_i, t_i) if $s_i \geq T$ and update $T \leftarrow t_i$.
  - **Runtime:** Sorting + linear time to create list of jobs. $O(n \log n) + O(n) = O(n \log n)$.

A writeup for Interval Scheduling

Correctness argument

- **Feasibility:** When a new job (s_i, t_i) is added by our algorithm, we require that $s_i \geq T$ where T is the latest end-time over all previously selected jobs. Therefore, the new job doesn't overlap with any previously selected jobs. By induction, the solution is feasible.
- **Remarks:**
    - We use the phrase 'by induction' liberally. It's implicit that the property being preserved is 'feasibility'. The induction is over the jobs selected.
    - This is more relaxed than your previous algorithm writing tasks in 300-level courses!

A writeup for Interval Scheduling

Correctness argument

- **Optimality:**
  - Let $\mathscr{E}$ be the jobs selected by our greedy algorithm. Consider any other choice of jobs $\mathscr{F}$ that has **more** jobs than $\mathscr{E}$. We claim that the j-th job in $\mathscr{E}$ ends at least before the j-th job in $\mathscr{F}$ ends.
  - To prove this, assume that the claim is false and let j be the smallest counterexample. The the end-time of the first $j-1$ jobs of $\mathscr{E}$ is $\leq$ than the end-time of the first $j-1$ jobs of $\mathscr{F}$. So, the job selected by $\mathscr{E}$ will end before that of $\mathscr{F}$ as our greedy choice is earliest selection.
  - Then, if $\mathscr{F}$ has more jobs than $\mathscr{E}$, our greedy algorithm would have added the final job of $\mathscr{F}$ to $\mathscr{E}$, a contradiction to the definition of $\mathscr{E}$.
- **Remarks:** The assumption that j is the smallest counterexample is a type of induction argument!