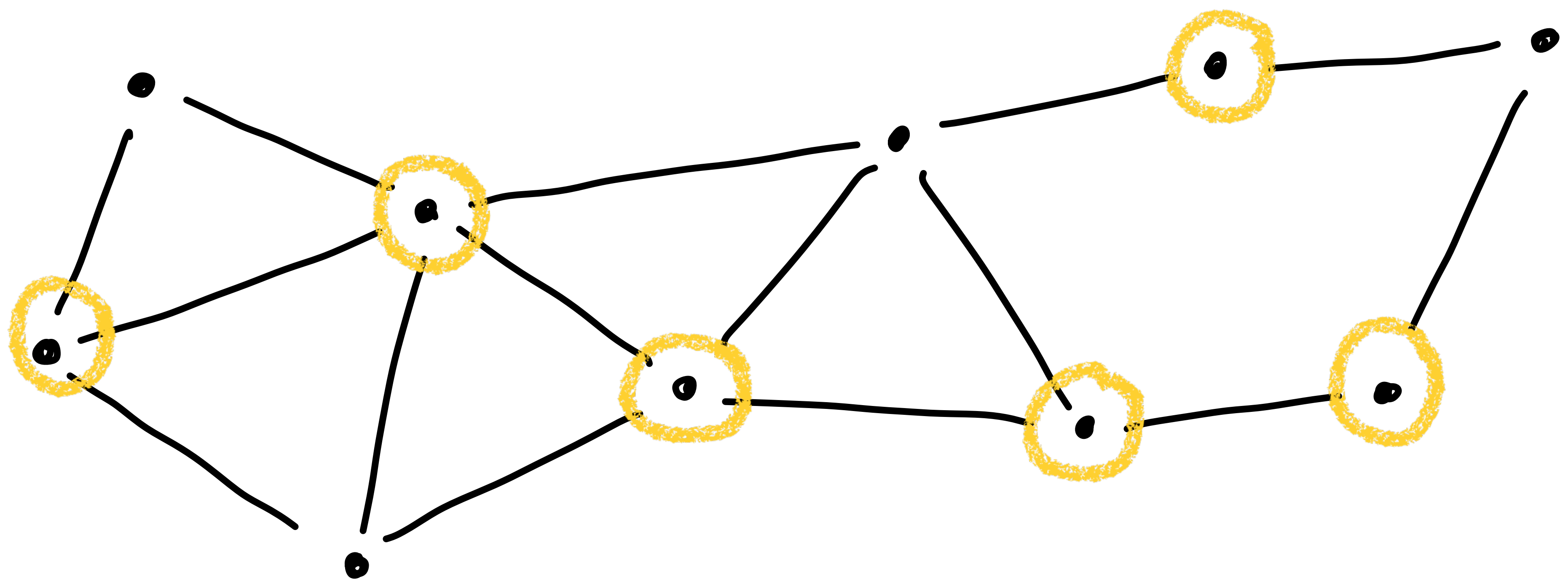
Lecture 22

Linear programming IV

Chinmay Nirkhe | CSE 421 Spring 2025

What's a problem LPs can't solve?

Vertex cover



- **Input**: an undirected graph $G = (V, E)$
- **Output**: a *minimal* set $S \subseteq V$ such that every edge contains at least one endpoint from S.
- There seems to be a pretty obvious LP for this problem. What goes wrong?

There is nothing ensuring that the optimal solution x will be integer.

One variable x_v for every vertex v.

$$\begin{cases} \min \sum_{v \in V} x_v \\ \text{s.t.} \quad x_v \leq 1 \quad \forall v \in V \\ \quad x_u + x_v \geq 1 \quad \forall e = (u,v) \in E \\ \quad x \geq 0 \end{cases}$$

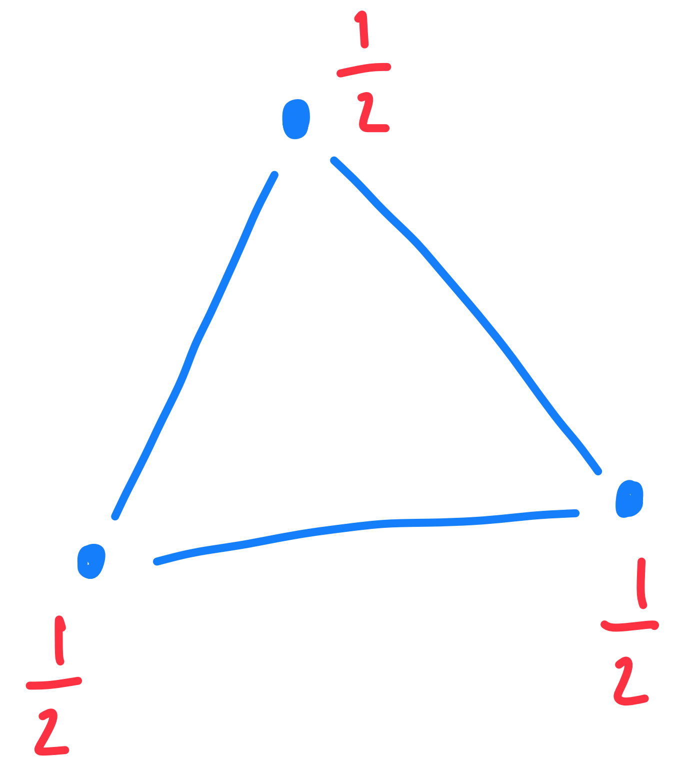
What's a problem LPs can't solve?

Vertex cover

- **Input**: an undirected graph $G = (V, E)$
- **Output**: a *minimal* set $S \subseteq V$ such that every edge contains at least one endpoint from S.
- There seems to be a pretty obvious LP for this problem. What goes wrong?

There is nothing ensuring that the optimal solution x will be integer.

Ex.



LP solution is $\frac{1}{2}$ on each vertex.

(a) LP min is $\frac{3}{2}$

(b) optimal sol has value 2.

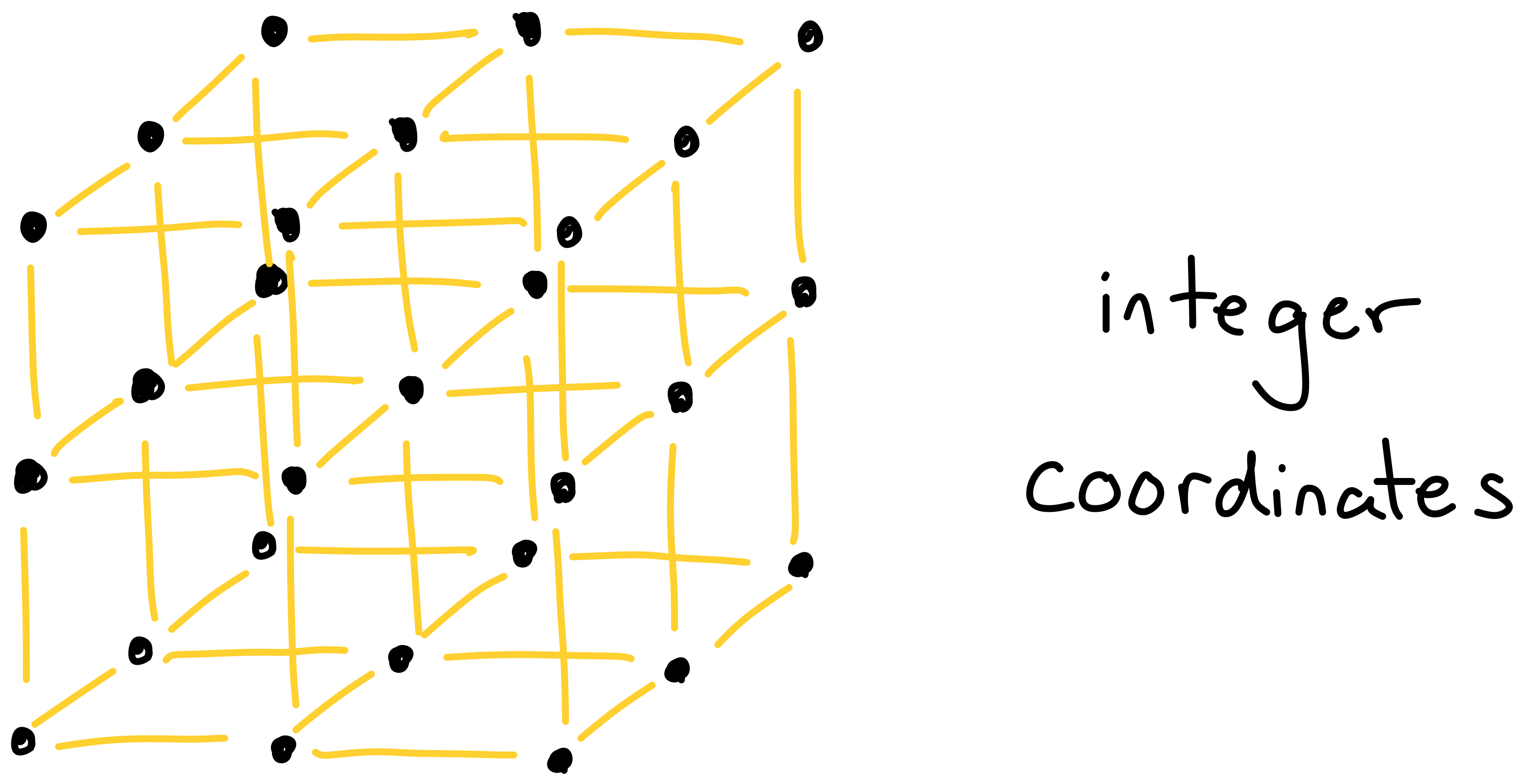
One variable x_v for every vertex v.

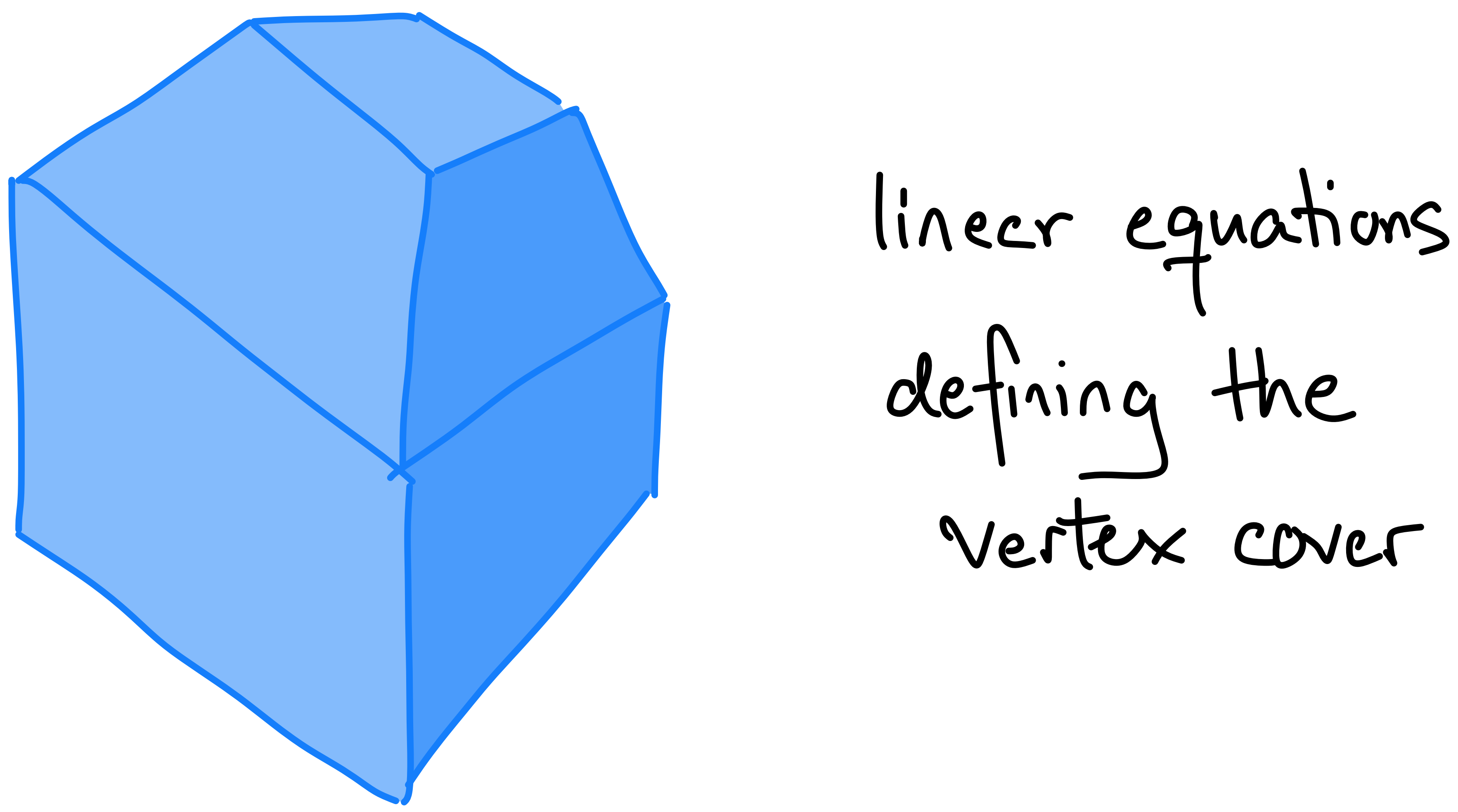
$$\begin{cases} \min \sum_{v \in V} x_v \\ \text{s.t.} \quad x_v \le 1 \quad \forall v \in V \\ \quad x_u + x_v \ge 1 \quad \forall e = (u,v) \in E \\ \quad x \ge 0 \end{cases}$$

LP relaxation

Vertex cover

- The LP we tried to write for vertex cover yields a fractional solution
- It is a “relaxation” of the vertex cover problem from integer to fractional solutions
    - In the relaxation we increase the feasible space from integer coordinates to include all solutions
    - Can be used to generate randomized approximation algorithms for vertex cover.





Max flow versus vertex cover

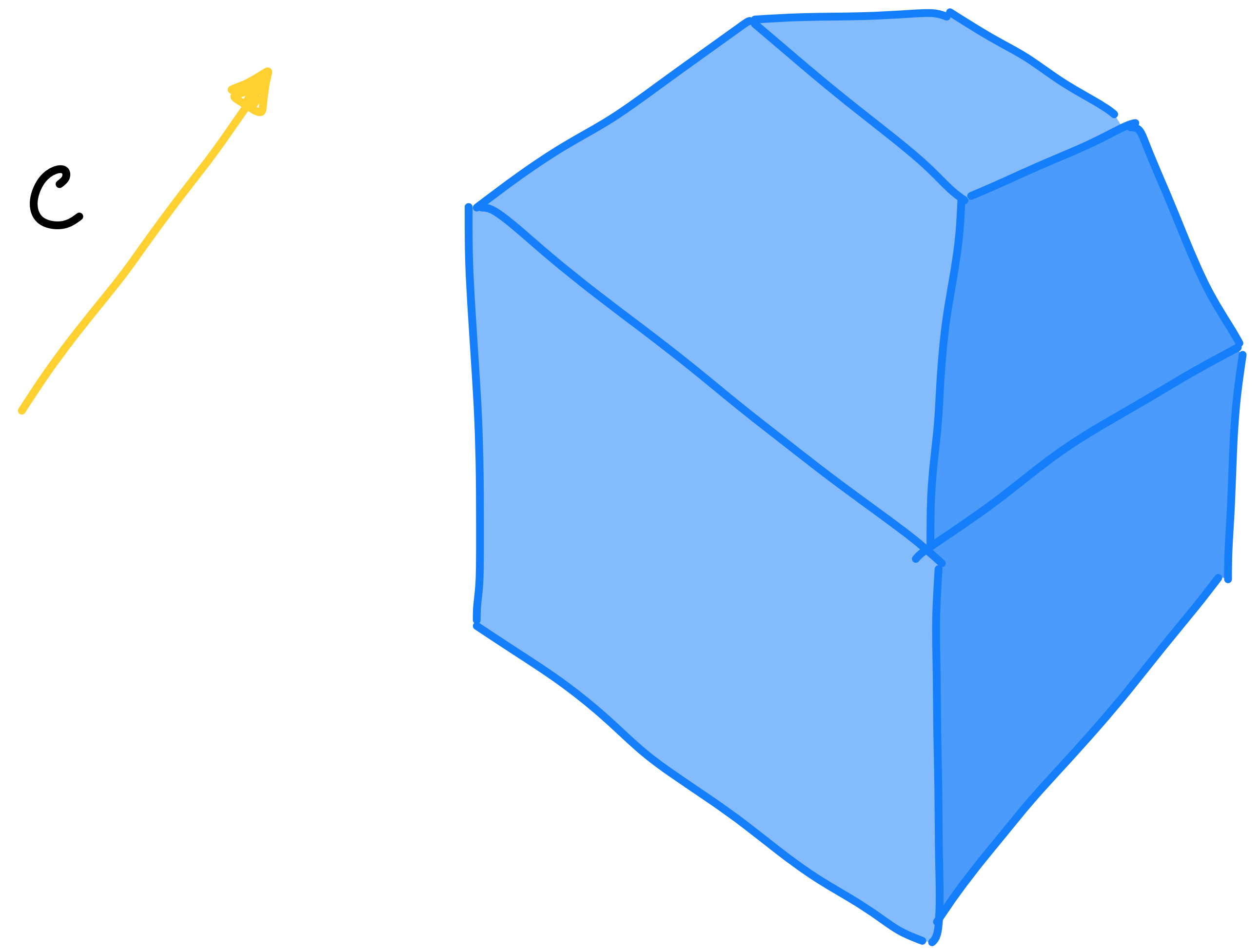
- Why can max flow natively guarantee integer solutions while vertex cover cannot?
- Recall, the optimum of an LP occurs at a vertex
  - The vertices of an LP correspond to points where linear equations are exactly satisfied
  - Turns out flow equations when exactly satisfied always have integer solutions
    - Quite a beautiful piece of mathematics
    - Too technical to warrant more time in this course

The simplex method

- Finally, we are going to cover an algorithm for solving LPs
- The algorithm is called **the simplex method** and it will be unique amongst the algorithms we study in this course
  - The simplex method runs incredibly fast in practice and is super useful
  - However, it can run in exponential time in the worst case even when there exist other polynomial time algorithms for the problem
- Later on, we will take a high-level glance at algorithms for solving LPs that are known to run in polynomial time

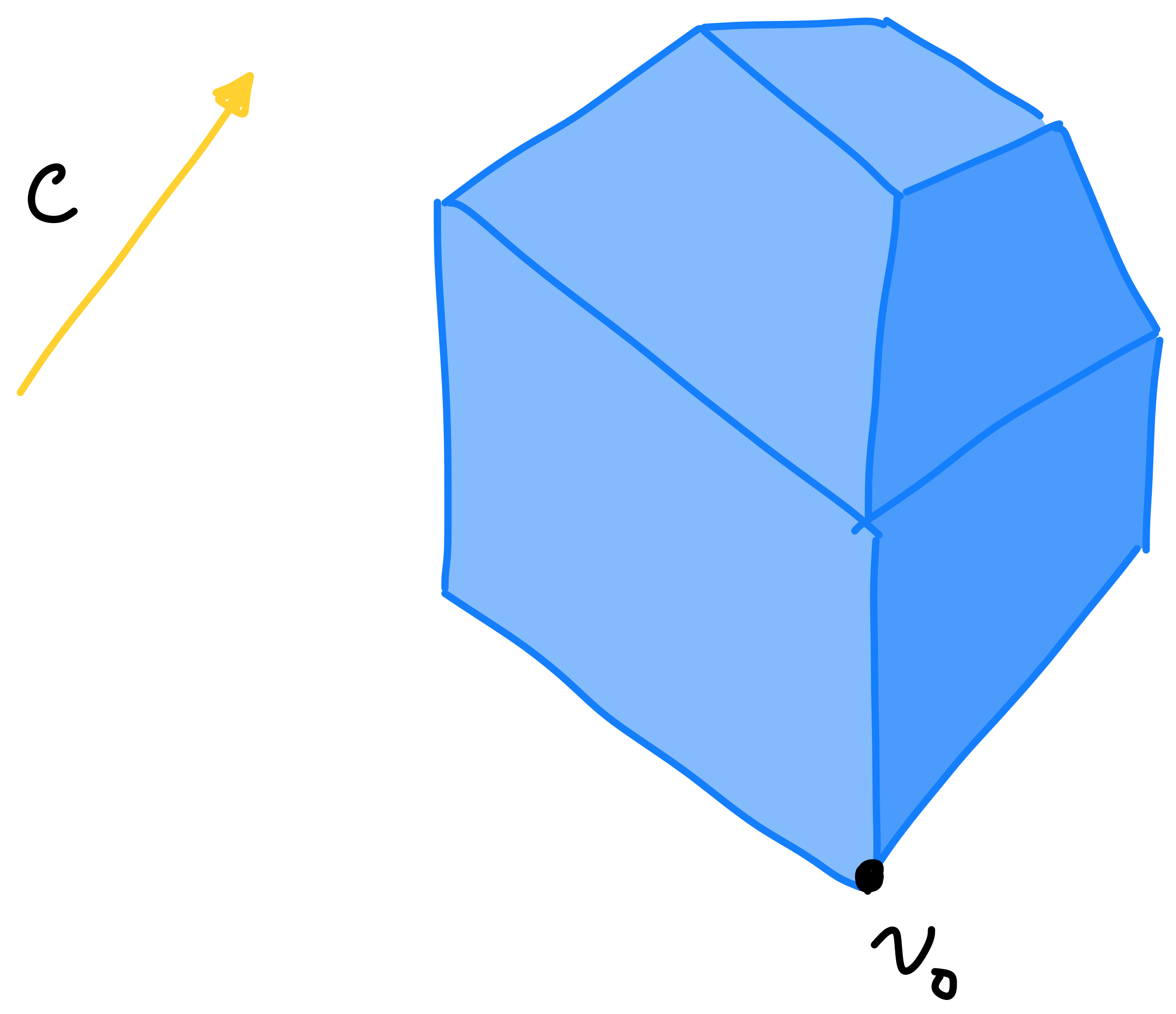
The simplex method

- Simplex is a greedy algorithm
- **High-level algorithm**:
  - Start from a vertex of the polytope
  - In each step, move to the neighboring vertex that optimizes $c^\top x$
  - Equivalently, move along the edge pointing the most in the c direction



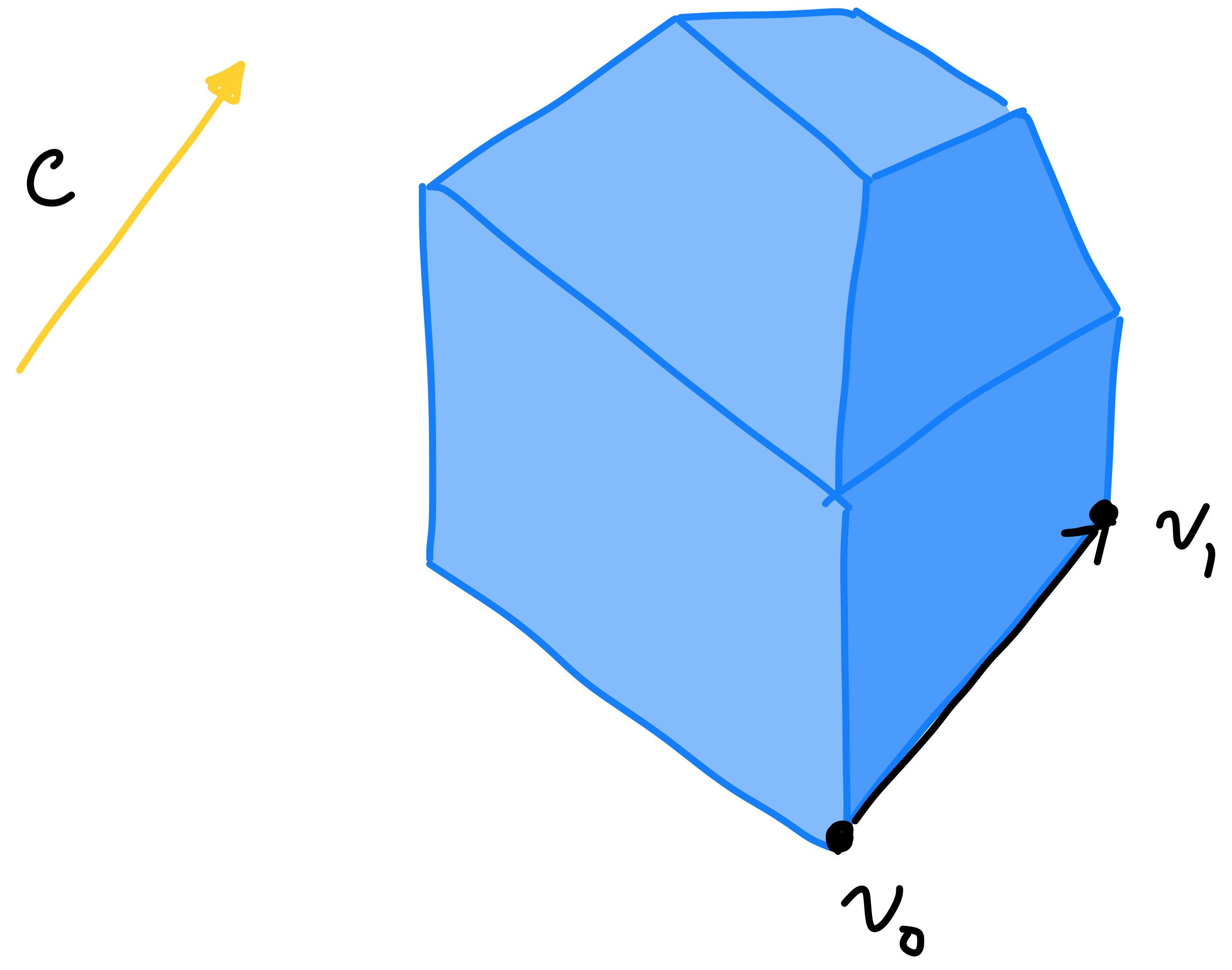
The simplex method

- Simplex is a greedy algorithm
- **High-level algorithm**:
  - Start from a vertex of the polytope
  - In each step, move to the neighboring vertex that optimizes $c^\top x$
  - Equivalently, move along the edge pointing the most in the c direction



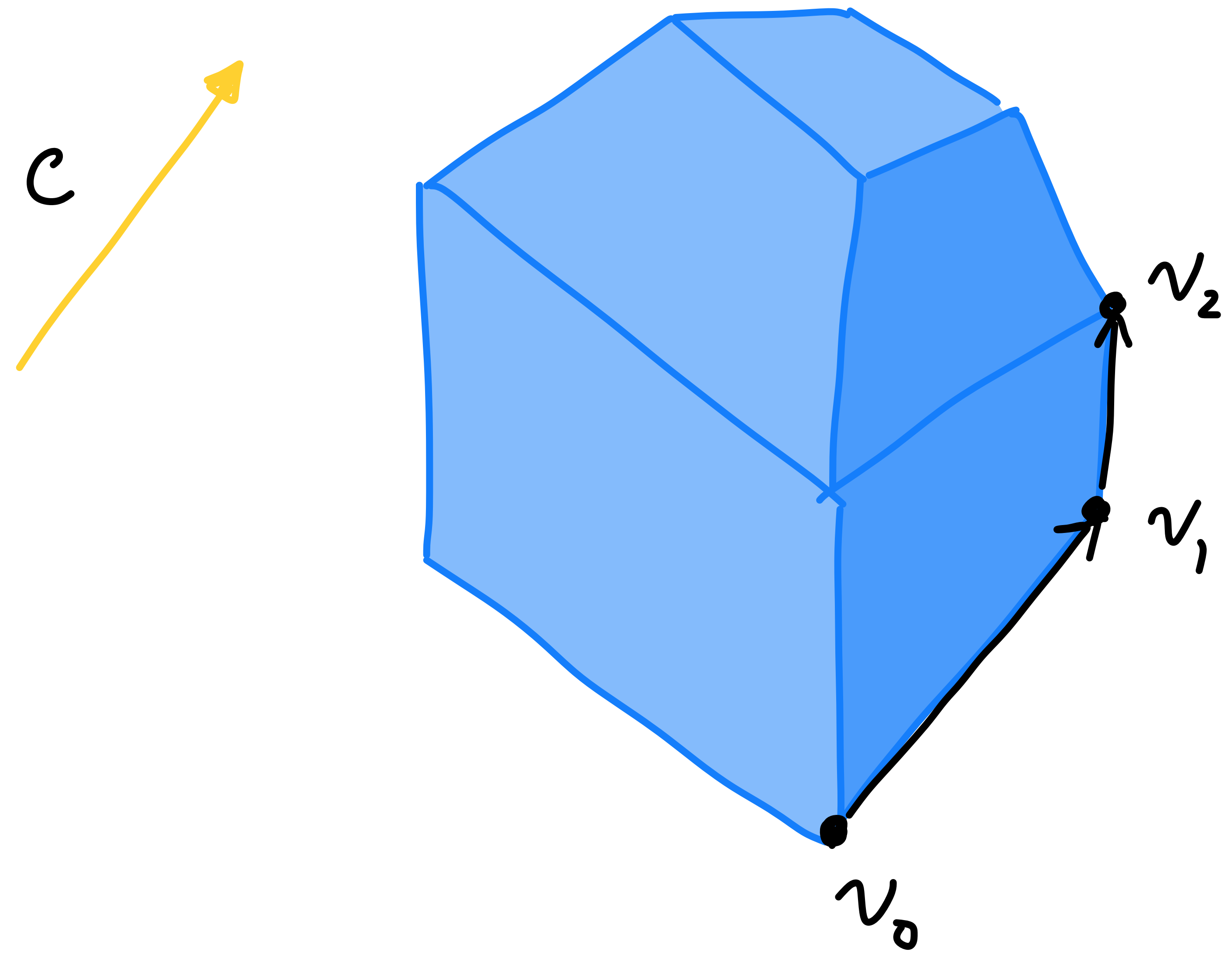
The simplex method

- Simplex is a greedy algorithm
- **High-level algorithm**:
  - Start from a vertex of the polytope
  - In each step, move to the neighboring vertex that optimizes $c^\top x$
  - Equivalently, move along the edge pointing the most in the c direction



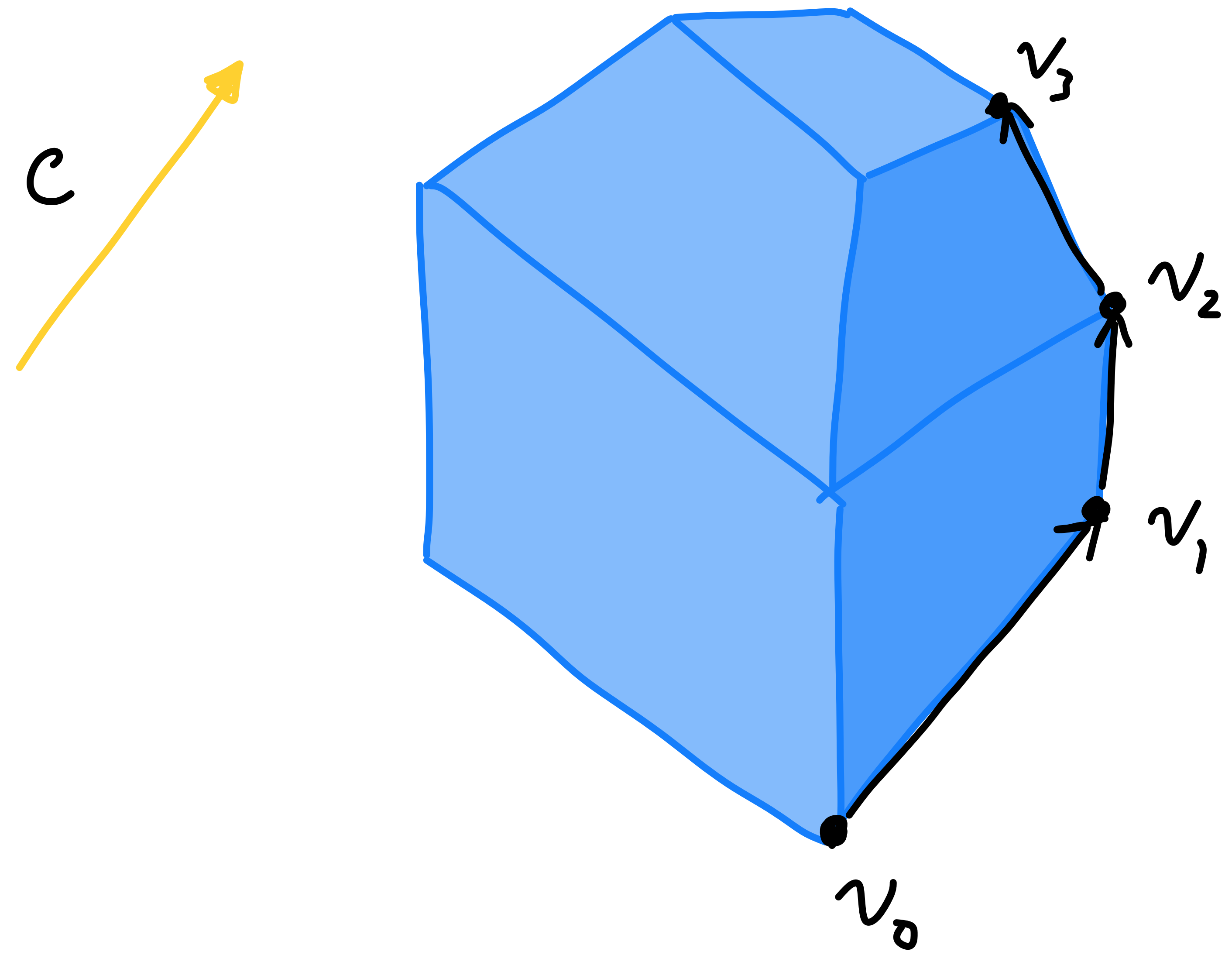
The simplex method

- Simplex is a greedy algorithm
- **High-level algorithm**:
  - Start from a vertex of the polytope
  - In each step, move to the neighboring vertex that optimizes $c^\top x$
  - Equivalently, move along the edge pointing the most in the c direction



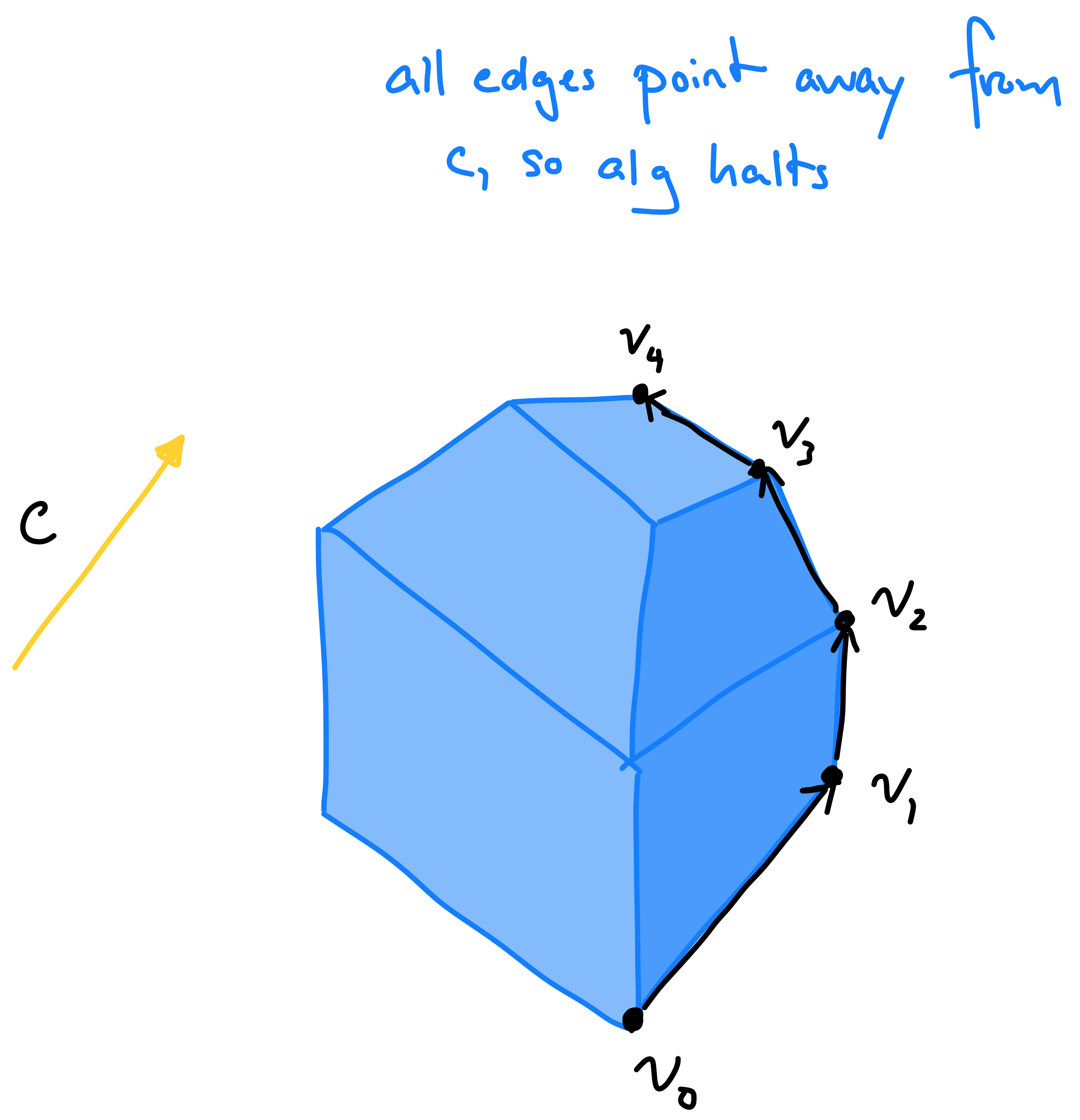
The simplex method

- Simplex is a greedy algorithm
- **High-level algorithm**:
  - Start from a vertex of the polytope
  - In each step, move to the neighboring vertex that optimizes $c^\top x$
  - Equivalently, move along the edge pointing the most in the c direction



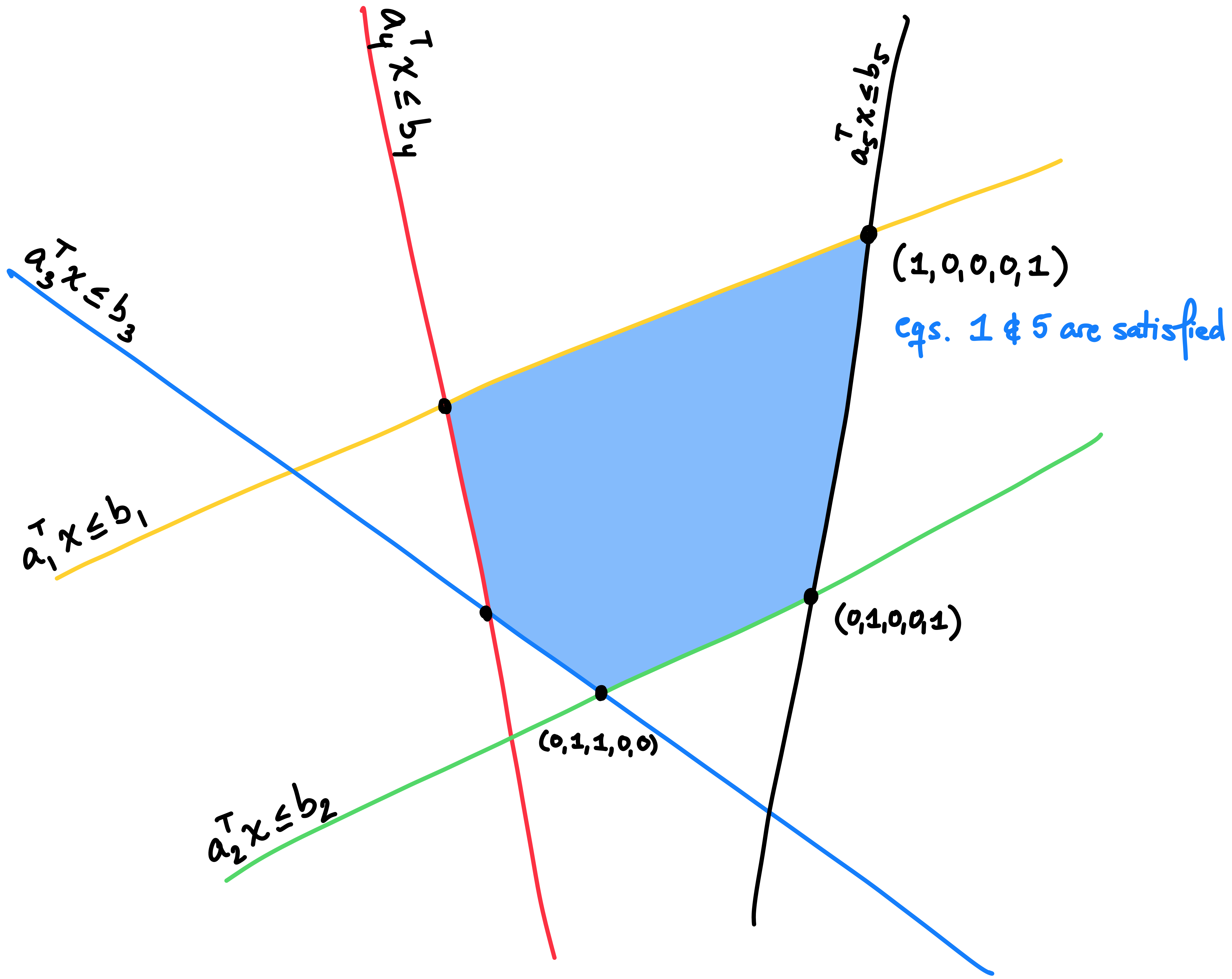
The simplex method

- Simplex is a greedy algorithm
- **High-level algorithm**:
  - Start from a vertex of the polytope
  - In each step, move to the neighboring vertex that optimizes $c^\top x$
  - Equivalently, move along the edge pointing the most in the c direction



The simplex method

- We are effectively consider a graph $G = (V, E)$ whose interior is the feasible region Γ.
- If we consider a feasible region defined by $\Gamma = \{Ax \leq b\}$ for $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$
  - Then, each vertex can be described by which n of the m equations are exactly satisfied
  - Describe vertices by points in $\{0,1\}^m$ of Hamming weight n
  - Two vertices are neighbors if they share all but 1 equation or equiv. the descriptions differ in two bits



The simplex method

Digging deeper into the algorithm

- Algorithm has two major steps:
  - Finding the first vertex (if one even exists as Γ could be infeasible)
  - Moving along an edge
- Moving along an edge:
  - Currently at a vertex described by n out of m equations
  - Can consider all possible vertices that share all but one equation
  - At most $n \cdot (m - n)$ neighbors
  - Gives a polynomial time algorithm for moving along an edge

The simplex method

Digging deeper into the algorithm

- **Finding the first vertex**

Input: $\max \; c^T x$

s.t. $Ax \le b$

$x \ge 0$

The feasible region is

$\Gamma \; \{Ax \le b, \; x \ge 0\}$

Goal: Output a vertex of Γ.

Consider a second LP

known as slack variables

$\min \; z_1 + \dots + z_m$

s.t. $b_i - a_i^T x \le z_i \quad \forall \, i = 1, \dots, m$

$x \ge 0$

$z \ge 0$.

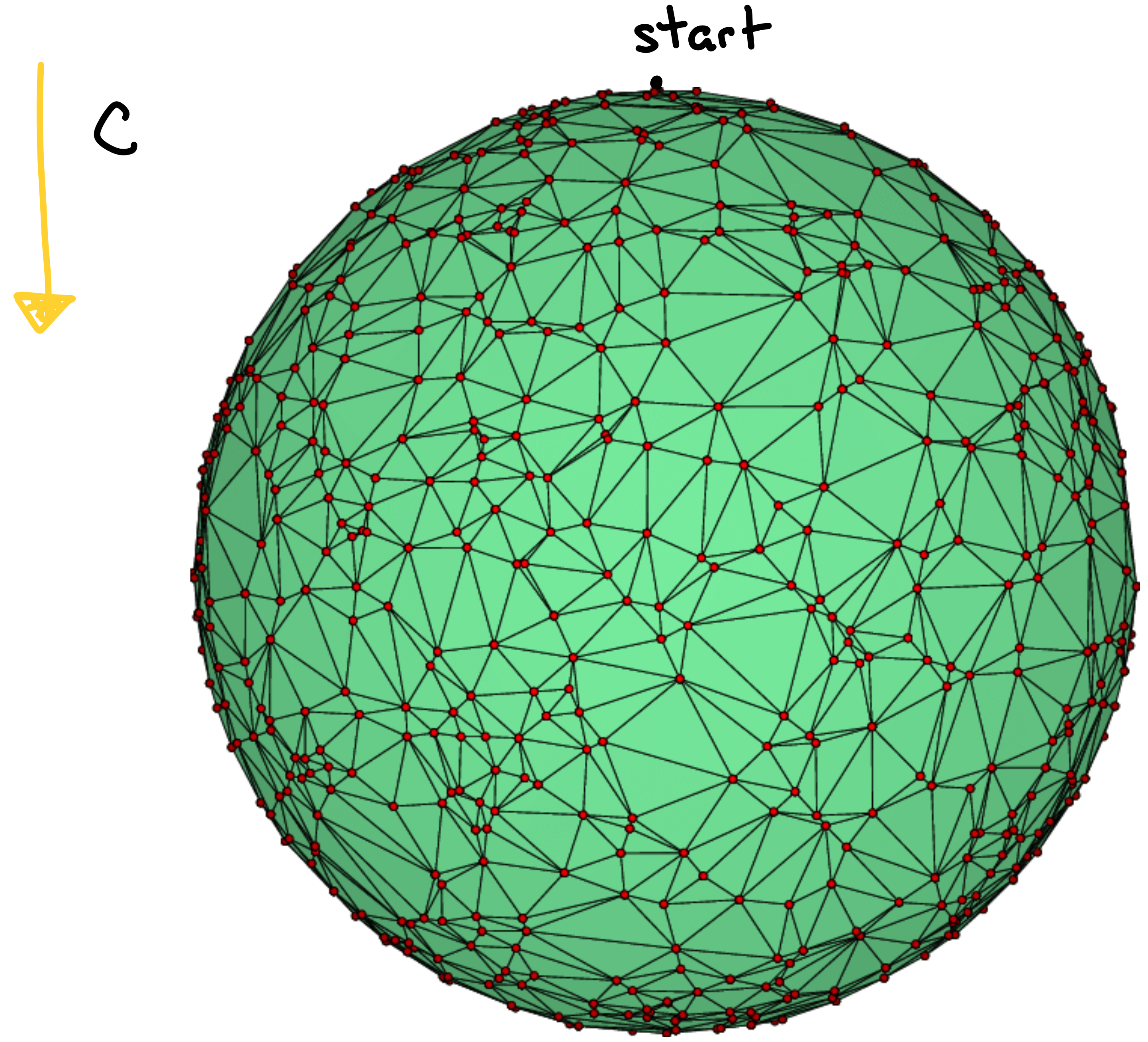
Notice that $(x = 0, \; z = b^{(+)})$ is a vertex of 2^{nd} LP.

Since we know a vertex of 2^{nd} LP, we can find it's OPT with simplex.

Claim: If (x, z) is OPT of 2^{nd} LP, then x is a vertex of Γ. Proof: exercise.

We have found a vertex of original polytope.

The simplex method

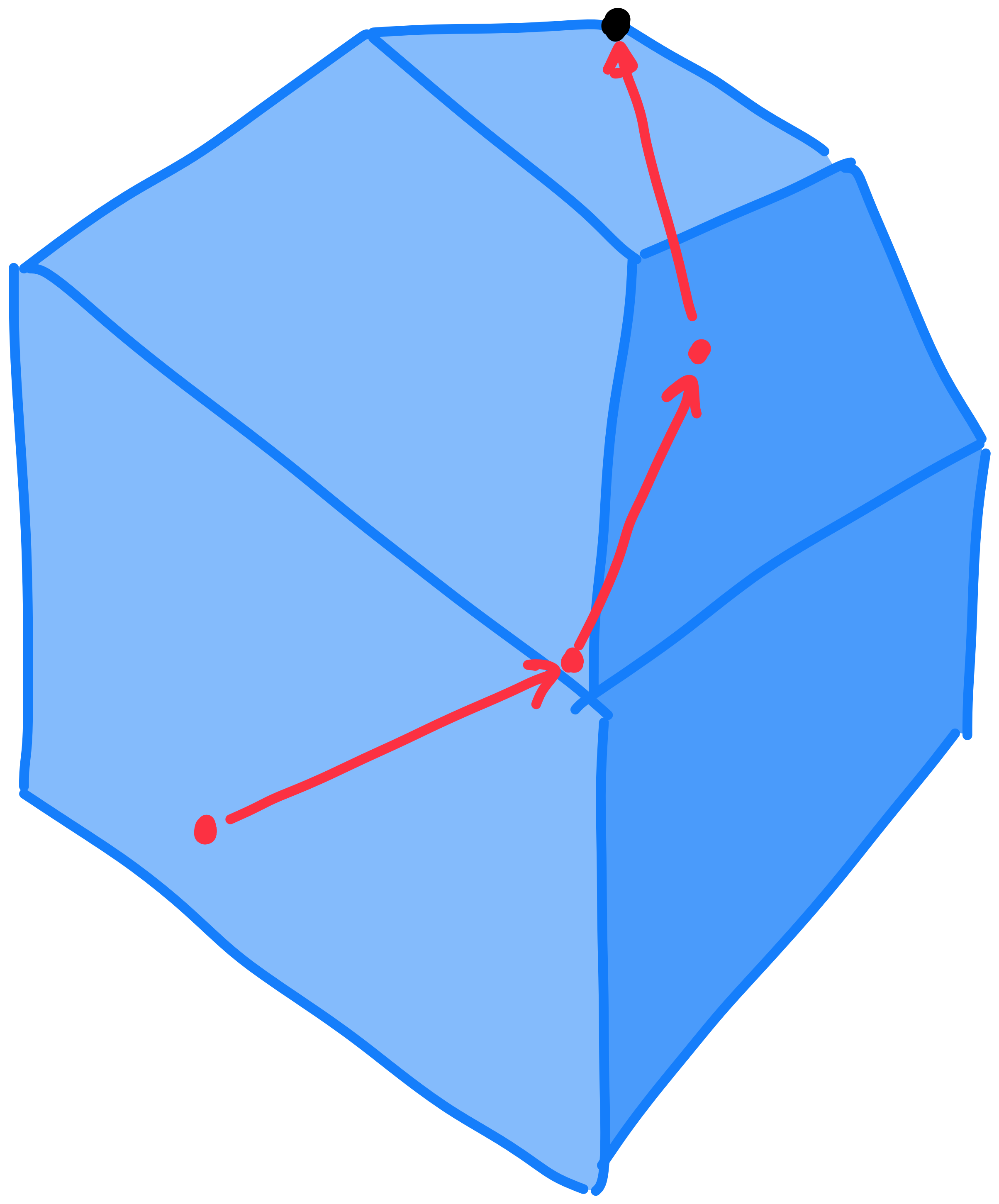


- We have not given runtimes for the simplex method on purpose
  - The runtime can be exponential because the algorithm goes on the *outside* of the polytope which could have lots of vertices, edges, and facets
  - However, simplex runs remarkably well in practice
  - Is there a reconciliation? An algorithm that may do okay in practice but has guaranteed worst case runtime that is polynomial?

Interior point and ellipsoid methods

Interior point

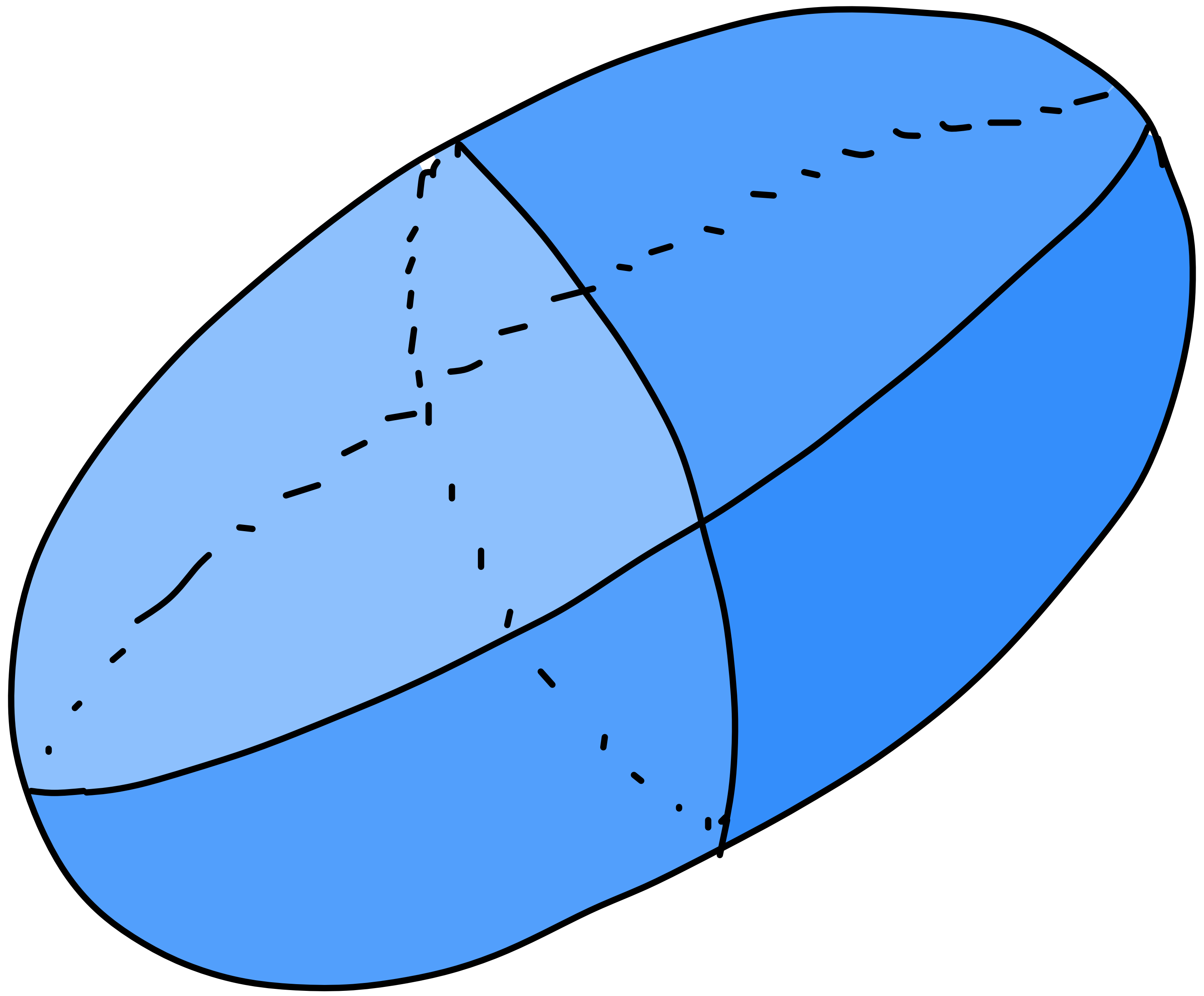
- Keep track of a point *inside* the polytope
- Follow a trajectory through the interior to optimal solution
- Solve a sequence of easier problems to approximate original LP, gradually becoming more accurate
- Runs about as fast as simplex in practice and has guarantees on runtime
- The “state-of-the-art” algorithm and a key step in optimal algorithms for problems like max flow



Interior point and ellipsoid methods

Ellipsoid method

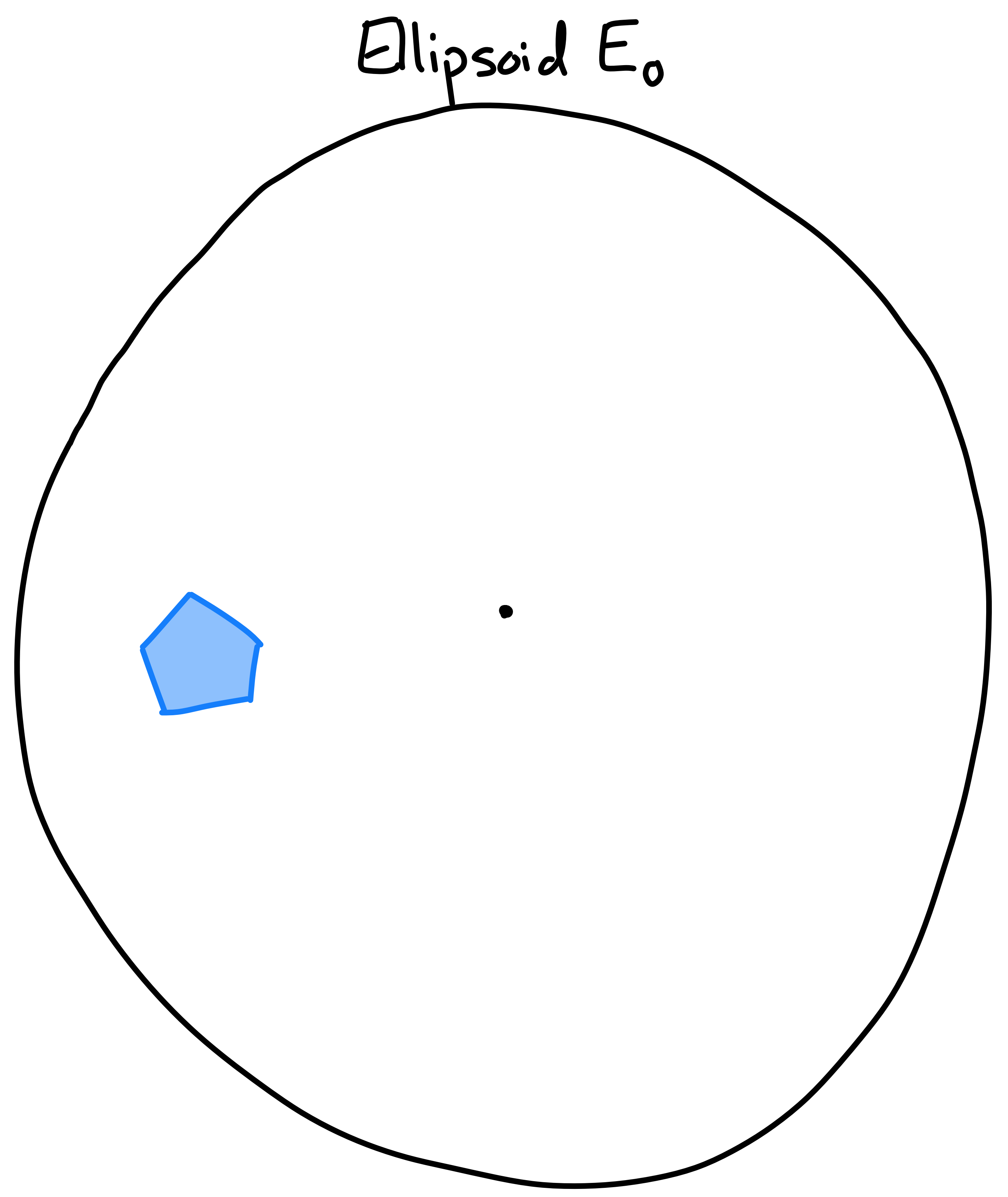
- What is an ellipsoid?
- An ellipsoid is a stretched sphere (in any direction)
- Can be defined by a quadratic equation



Interior point and ellipsoid methods

Ellipsoid method

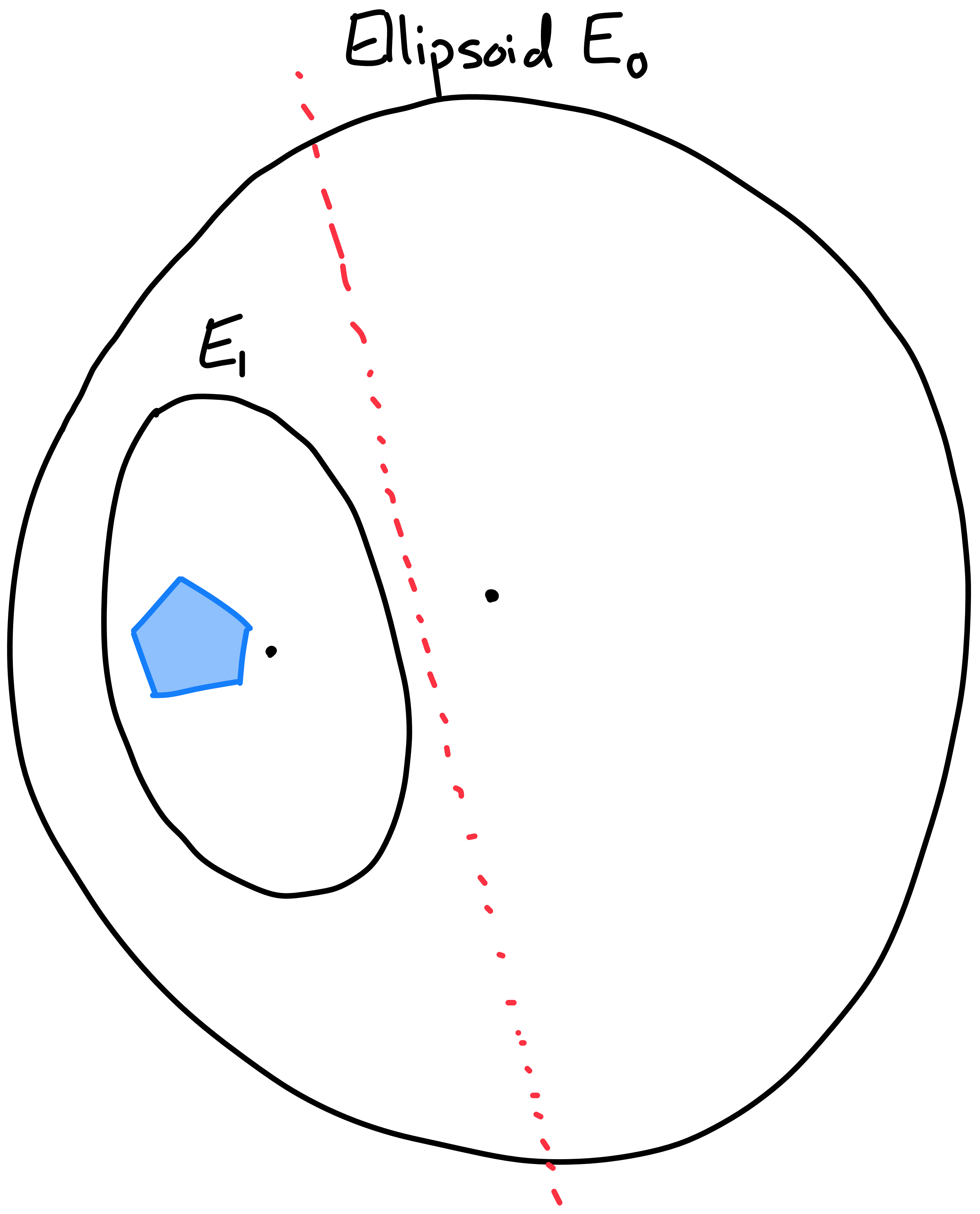
- Using LP duality, convert problem from optimizing a linear polytope to finding a feasible point in a different polytope Γ
- Generate a sequence of ellipsoids that always contain Γ
- Each time find a smaller ellipsoid (by guaranteed ratio) until the center of the ellipsoid must be in Γ
- Very slow in practice but first guaranteed algorithm for solving LPs



Interior point and ellipsoid methods

Ellipsoid method

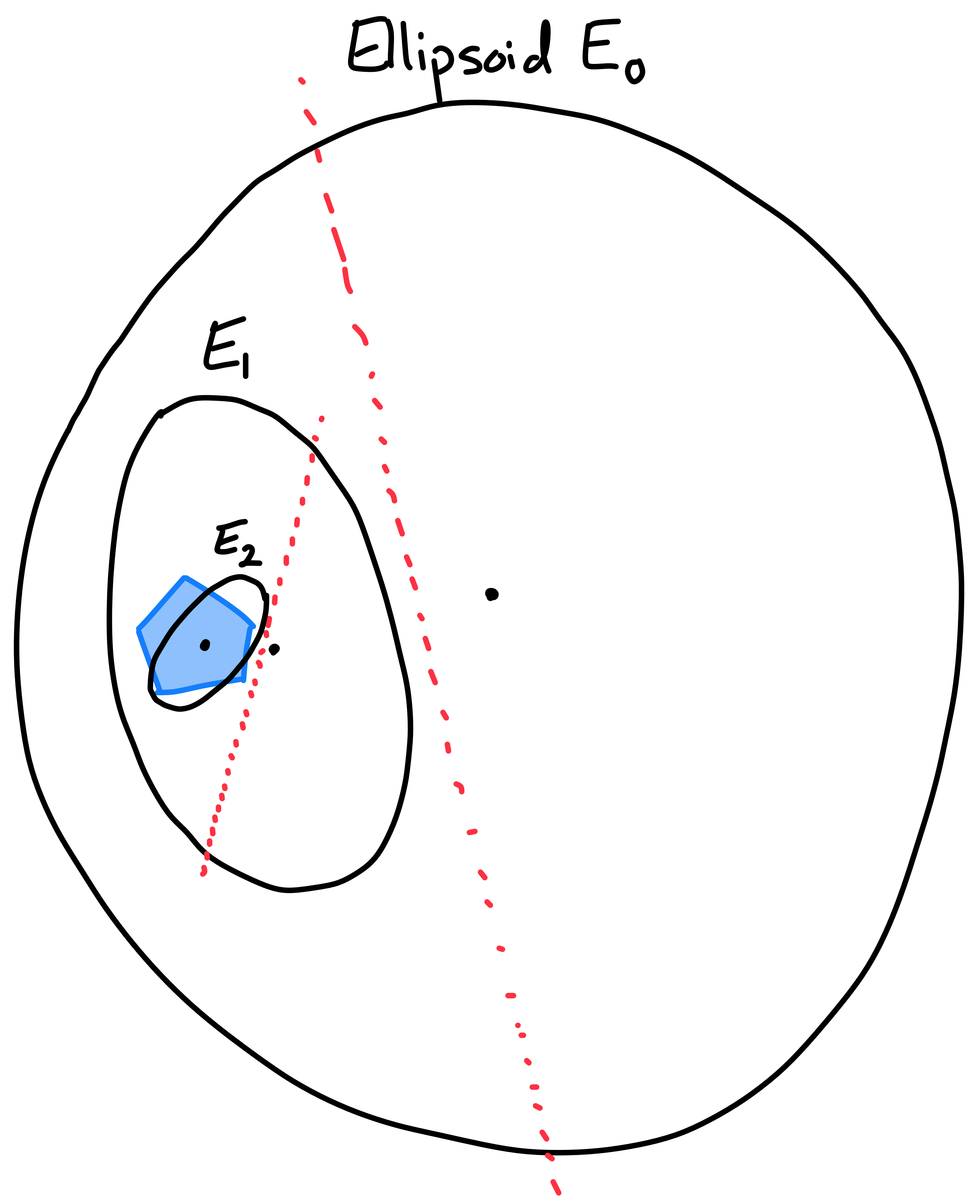
- Using LP duality, convert problem from optimizing a linear polytope to finding a feasible point in a different polytope Γ
- Generate a sequence of ellipsoids that always contain Γ
- Each time find a smaller ellipsoid (by guaranteed ratio) until the center of the ellipsoid must be in Γ
- Very slow in practice but first guaranteed algorithm for solving LPs



Interior point and ellipsoid methods

Ellipsoid method

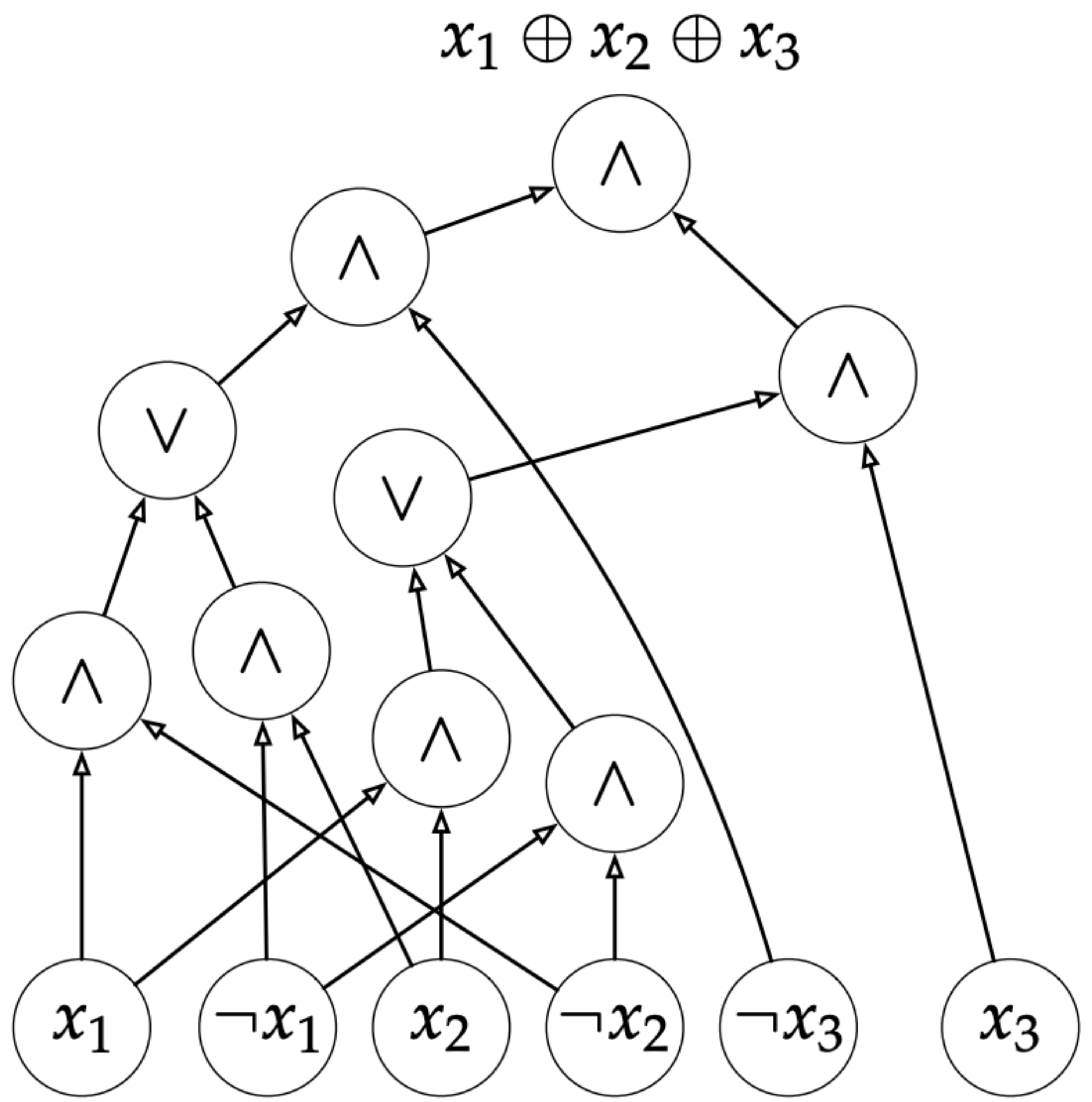
- Using LP duality, convert problem from optimizing a linear polytope to finding a feasible point in a different polytope Γ
- Generate a sequence of ellipsoids that always contain Γ
- Each time find a smaller ellipsoid (by guaranteed ratio) until the center of the ellipsoid must be in Γ
- Very slow in practice but first guaranteed algorithm for solving LPs



Why is linear programming so important?

- **Fact:** Every boolean function $f : \{0,1\}^n \to \{0,1\}^n$ that can be computed in time T can be computed by a boolean circuit with $O(T \log T)$ gates.

- **Theorem:** Every boolean function can be expressible as a linear program with $O(T \log T)$ variables and constraints.



Boolean circuits are built from elementary gates:

$\wedge$, $\vee$,

and $\neg$

Why is linear programming so important?

- **Fact:** Every boolean function $f : \{0,1\}^n \to \{0,1\}^n$ that can be computed in time T can be computed by a boolean circuit with $O(T \log T)$ gates.

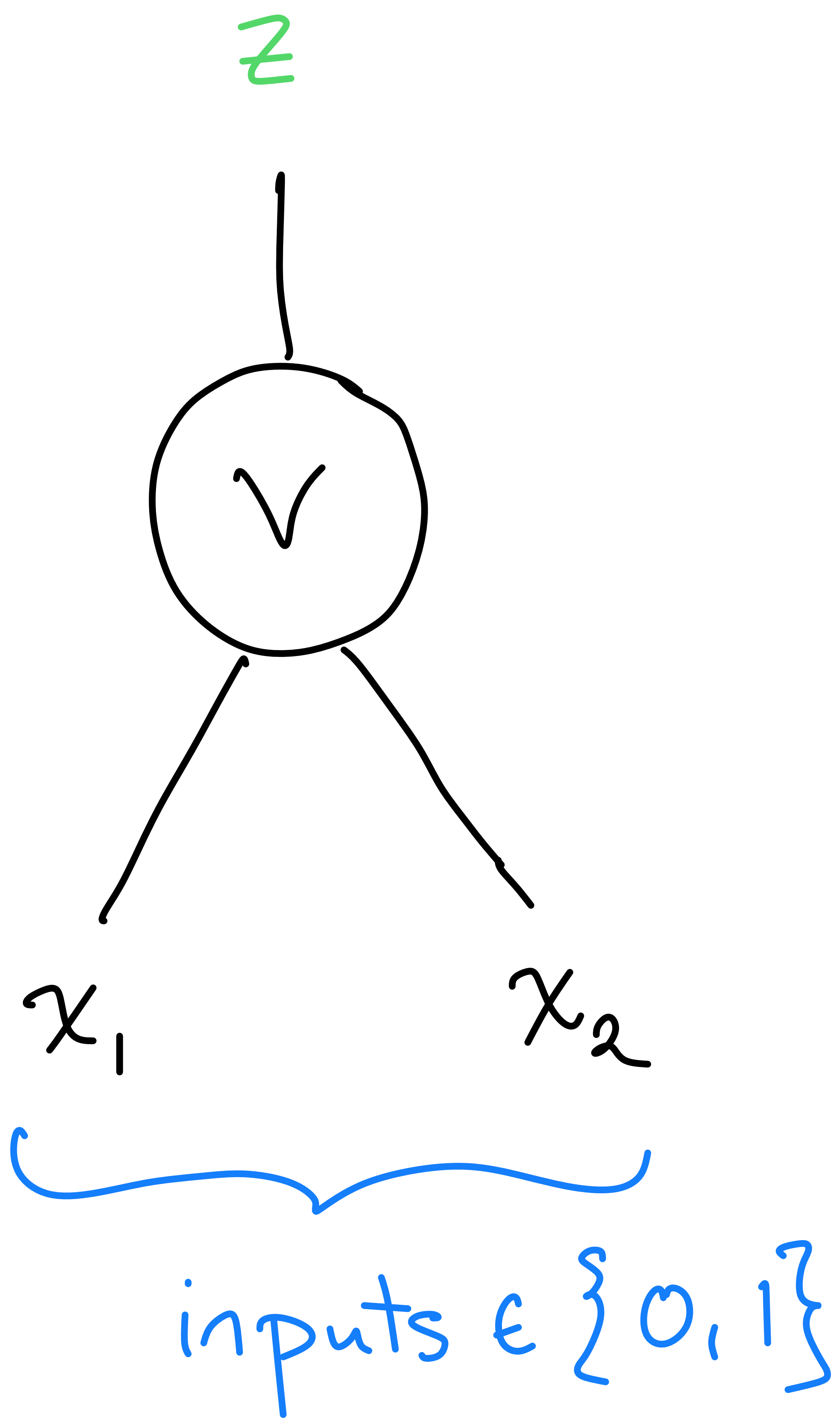
- **Theorem:** Every boolean function can be expressible as a linear program with $O(T \log T)$ variables and constraints.

Boolean circuits are built from elementary gates:

We create linear programming "gadgets" to handle each of the possible gates.

Converting Boolean circuits to LPs

OR gate



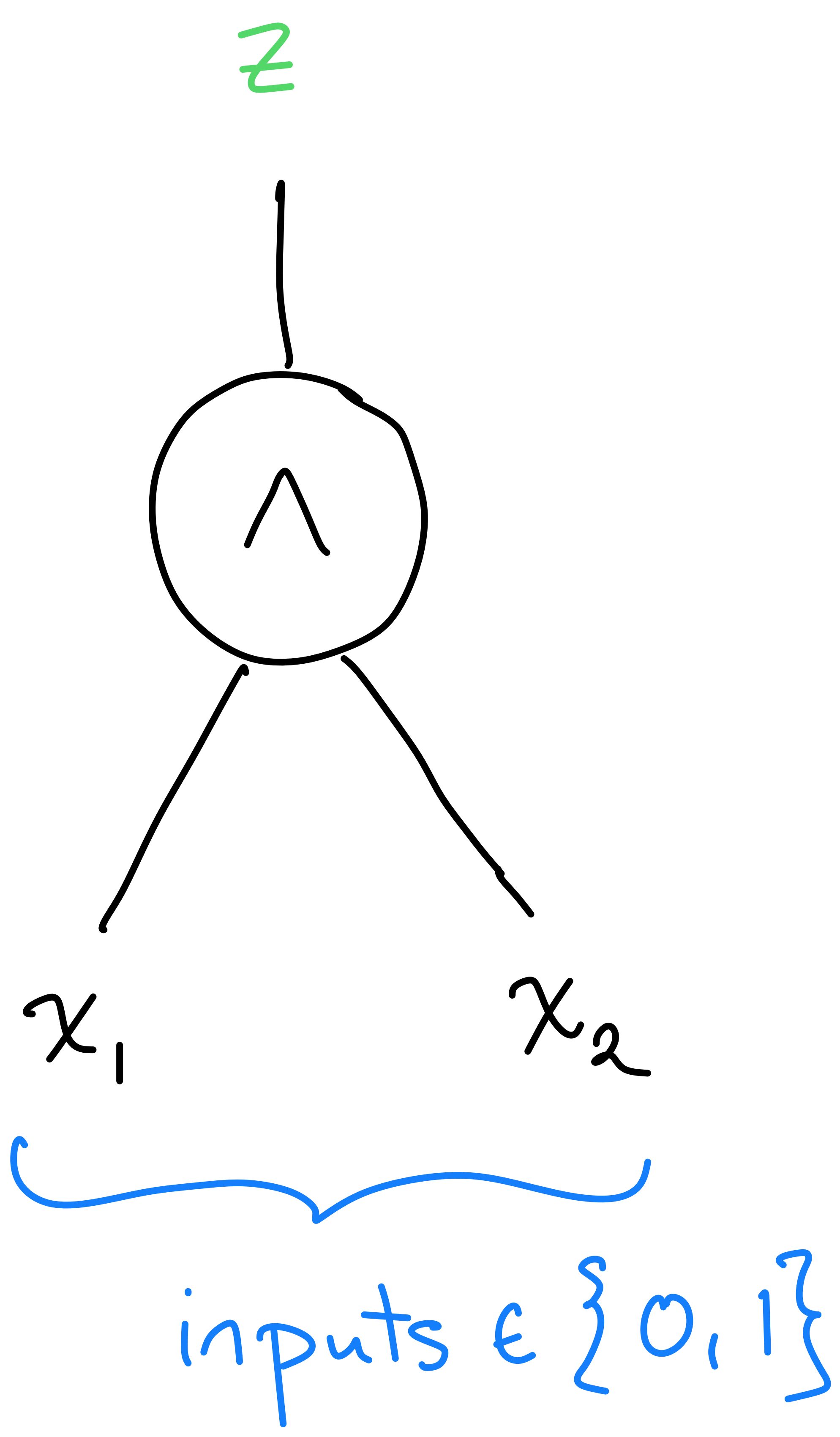
x_1	x_2	z
0	0	0
0	1	1
1	0	1
1	1	1

Observe: $z = \max(x_1, x_2)$

$$= \begin{cases} z \geq x_1 \\ z \geq x_2 \\ z \geq x_1 + x_2 \\ 0 \leq z \leq 1 \end{cases}$$

Converting Boolean circuits to LPs

AND gate



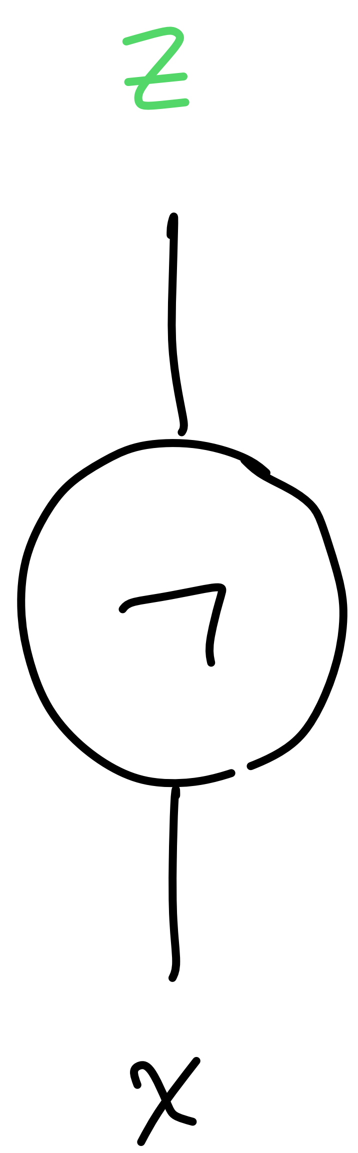
x_1	x_2	z
0	0	0
0	1	0
1	0	0
1	1	1

Observe: $z = \min(x_1, x_2)$

$$= \begin{cases} z \le x_1 \\ z \le x_2 \\ z \ge x_1 + x_2 - 1 \\ 0 \le z \le 1 \end{cases}$$

Converting Boolean circuits to LPs

NOT gate



input $\in \{0,1\}$

x	z
0	1
1	0

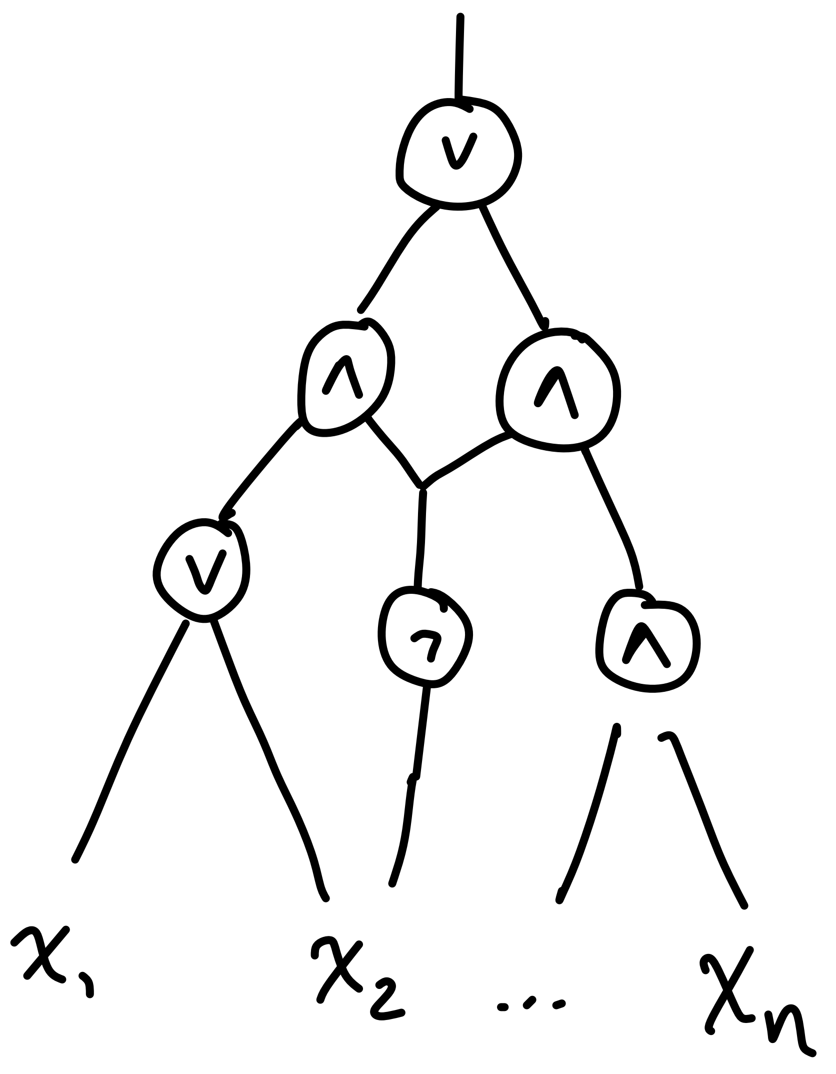
Observe: $z = 1 - x$

$$= \begin{cases} z \geq 1 - x \\ z \geq x - 1 \end{cases}$$

Converting Boolean circuits to LPs

Given the ability to convert an elementary gate to a system of lin. eqs.,

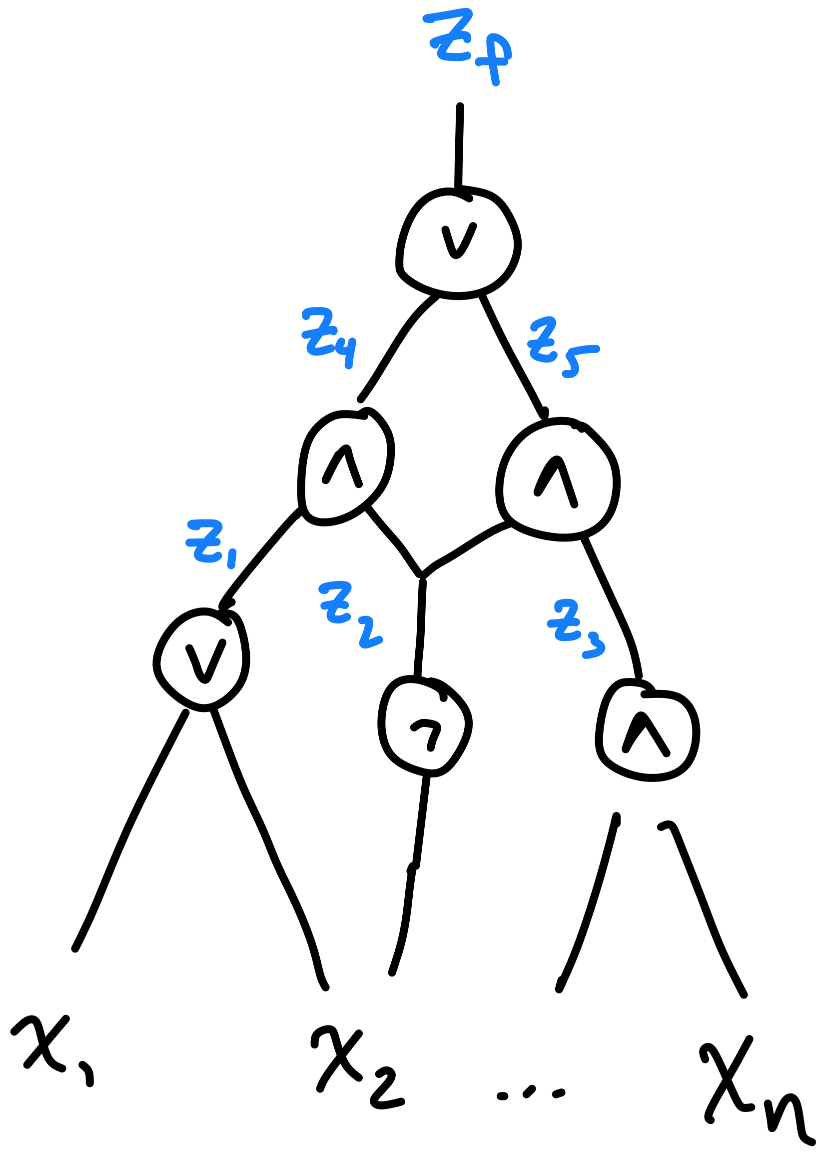
we take the full circuit and create a system of lin. eqs.



Converting Boolean circuits to LPs

Given the ability to convert an elementary gate to a system of lin. eqs.,
we take the full circuit and create a system of lin. eqs.

Assign a variable for the intermediate "wires".



$$
\begin{cases}
\max & z_f \\
\text{s.t.} & \forall \text{ gates } g. \quad \boxed{\text{eqs about } g} \\
 & z \geq 0
\end{cases}
$$

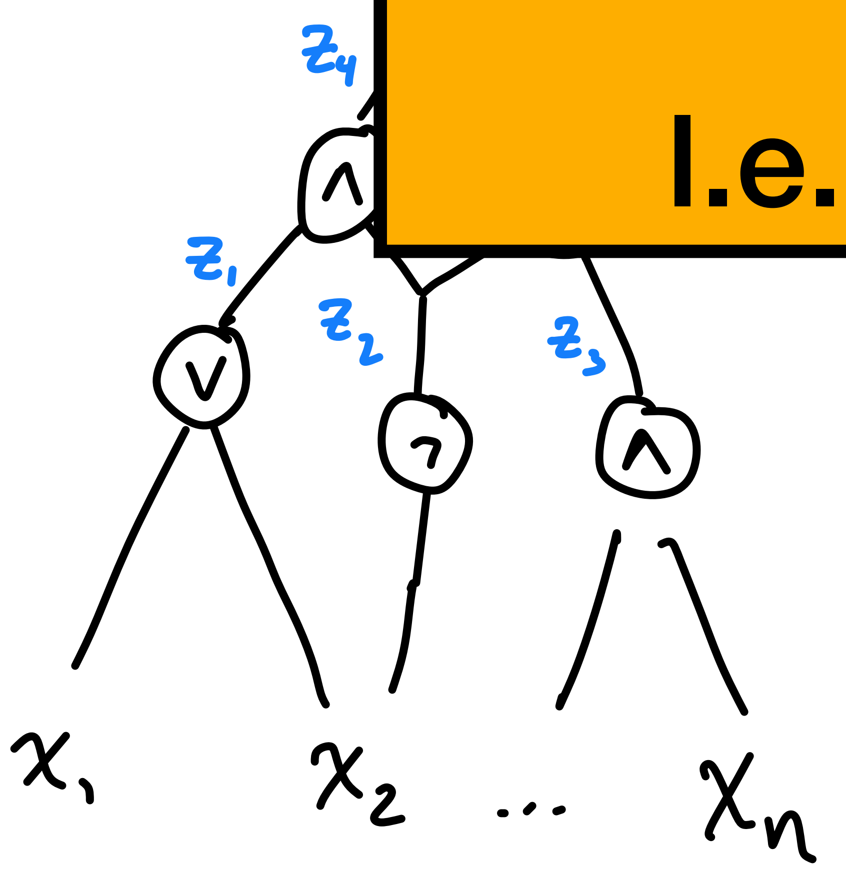
Converting Boolean circuits to LPs

Given the ability to convert an elementary gate to a system of lin. eqs.,

we to

Therefore, every computational problem computable by a boolean circuit of size T can be expressed as a linear program of size $O(T \log T)$.

I.e. linear programming is *universal* for computation



s.t. $\forall$ gates g. eqs about g

$z \geq 0$