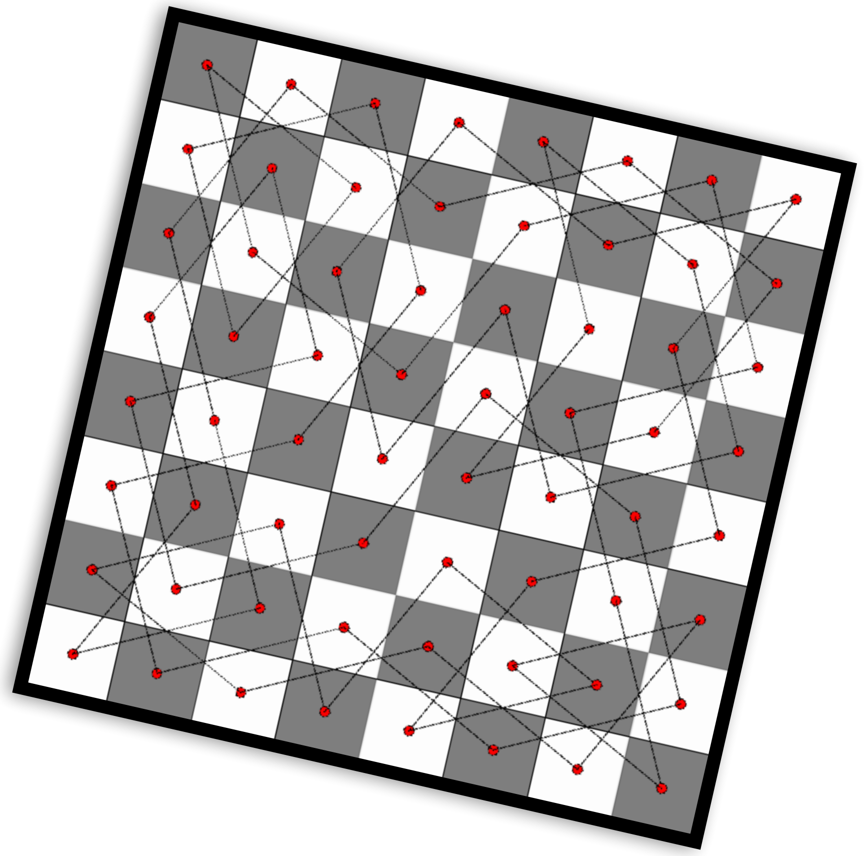
Advanced Algorithms for Optimization
Linear Programming

Pierre Schaus



*Many figures from Sedgewick and Wayne, Algorithms part 2, Coursera

A linear programme is

1) The maximization of a **linear** function

$$
\begin{aligned}
\text{maximize} \quad & x_1 + x_2 \\
\text{subject to} \quad & 4x_1 - x_2 \leq 8 \\
& 2x_1 + x_2 \leq 10 \\
& 5x_1 - 2x_2 \geq -2 \\
& x_1, x_2 \geq 0
\end{aligned}
$$

2) Over **linear** constraints

Matrix Notation

$$
\begin{aligned}
\text{maximize} \quad & cx \\
\text{subject to} \quad & Ax \leq b \\
& x \geq 0
\end{aligned}
$$

The Brewer Problem

- A small brewery produces ale and beer and wants to maximize profit

- Production limited by scarce resources (corn, hops, malt)

corn (480 lbs)   hops (160 oz)   malt (1190 lbs)

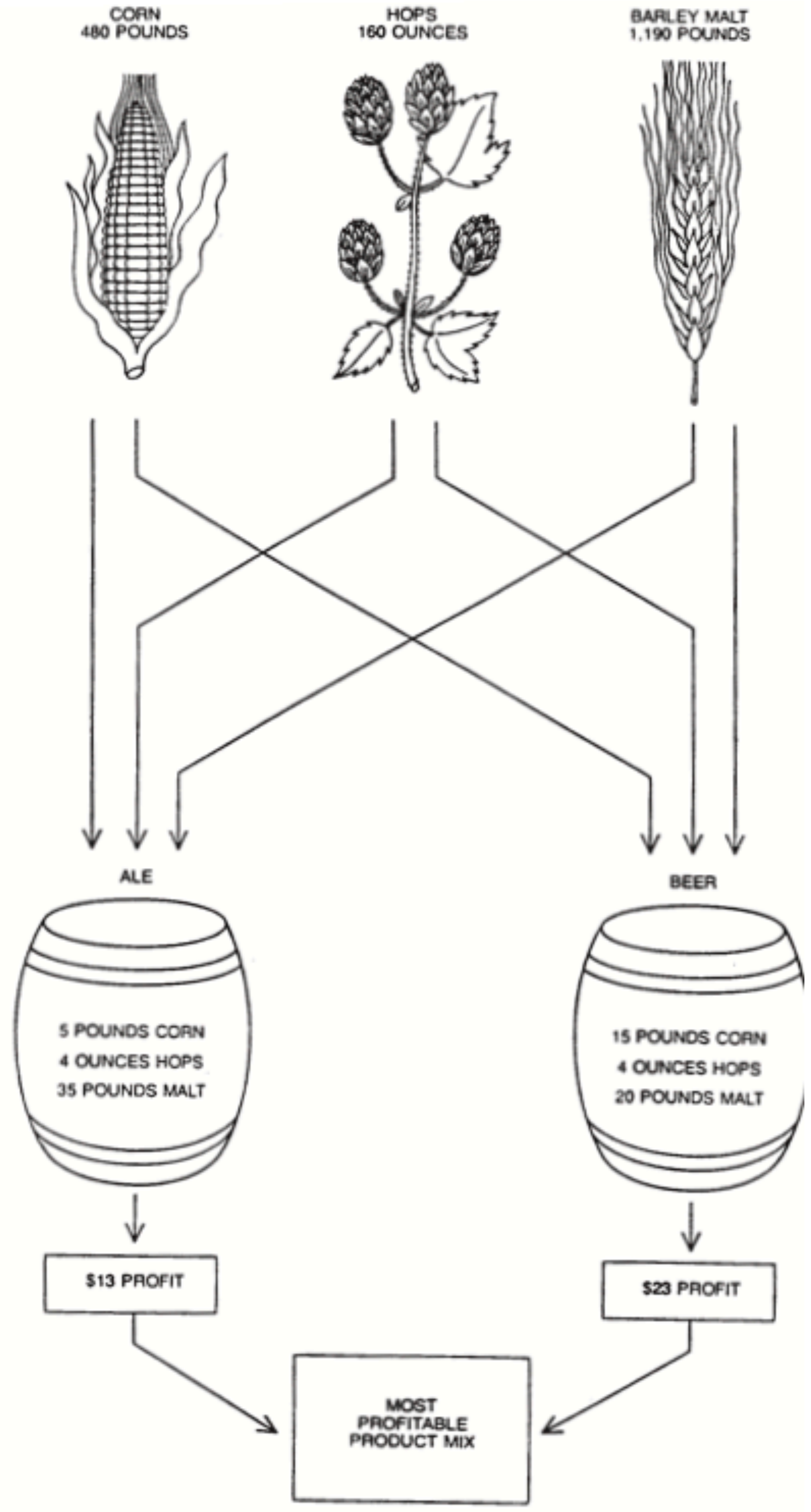
- The recipe for ale and beer require different proportions of resources

$13 profit per barrel $23 profit per barrel

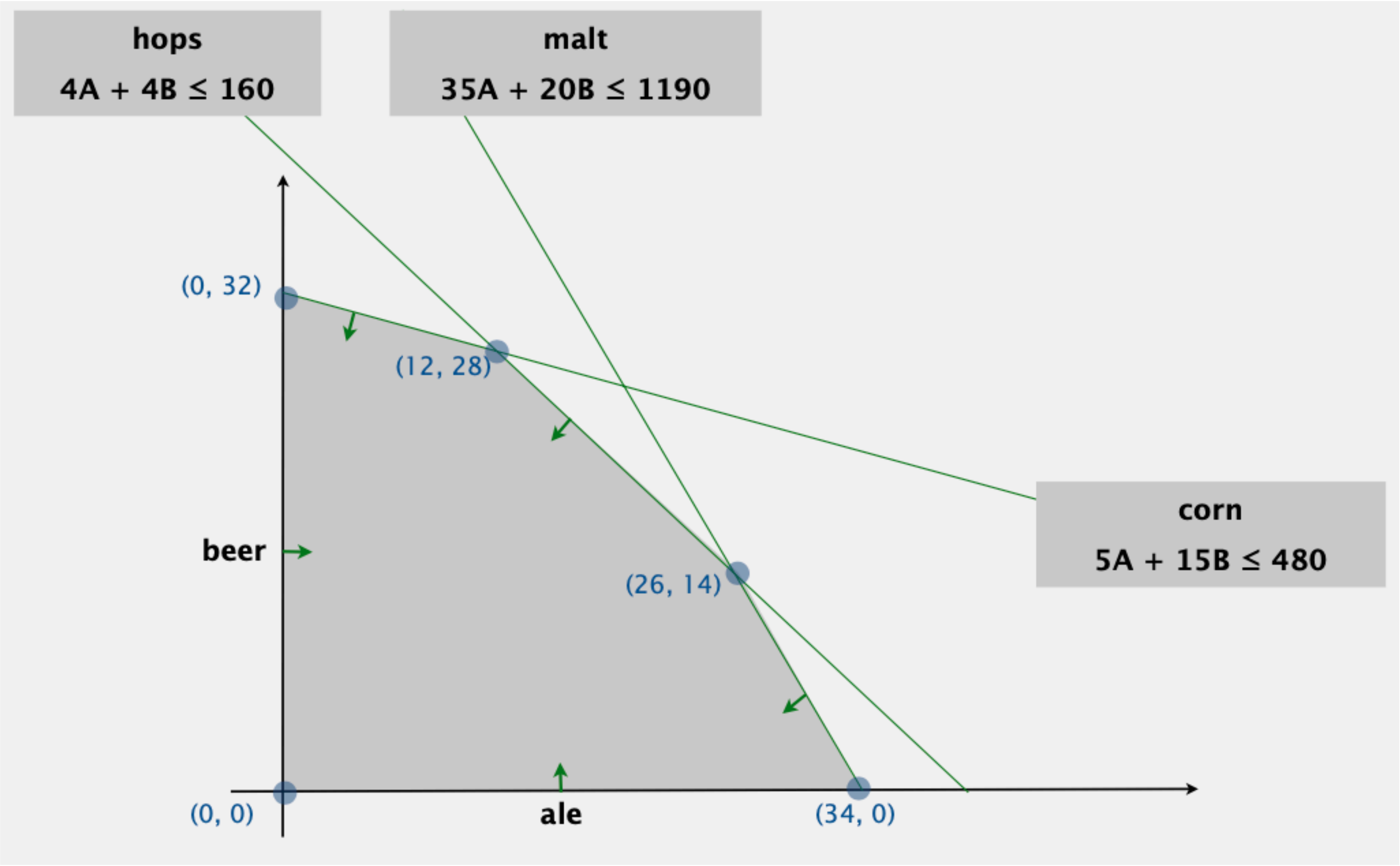
Brewer Problem: Linear Programming Formulation

- Variables: A = the number of barrels of ale, B = the number of barrels of beer.

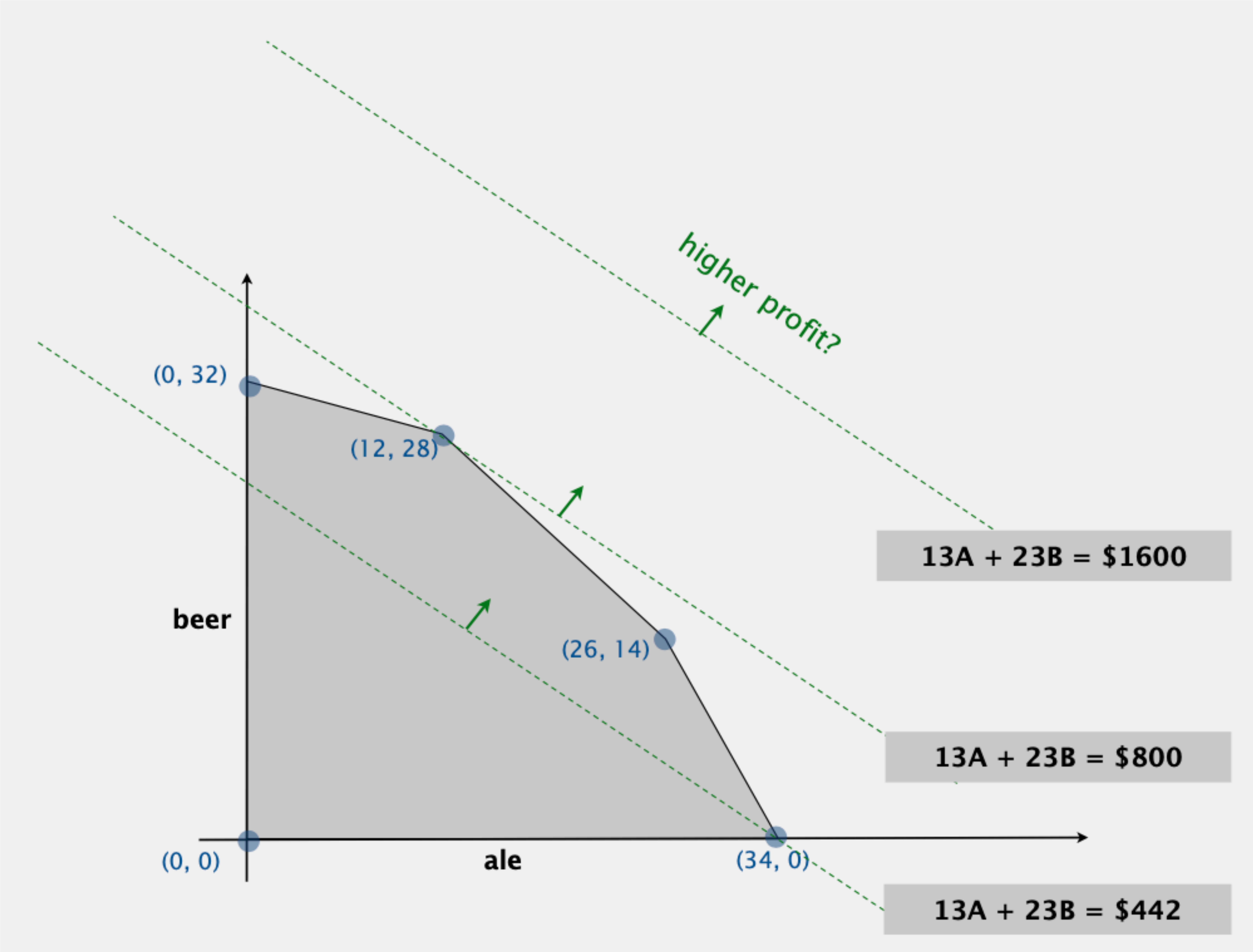
	ale		beer			
maximize	13A	+	23B			profits
subject to the constraints	5A	+	15B	≤	480	corn
	4A	+	4B	≤	160	hops
	35A	+	20B	≤	1190	malt
	A	,	B	≥	0	



Brewer Problem: Feasible Region

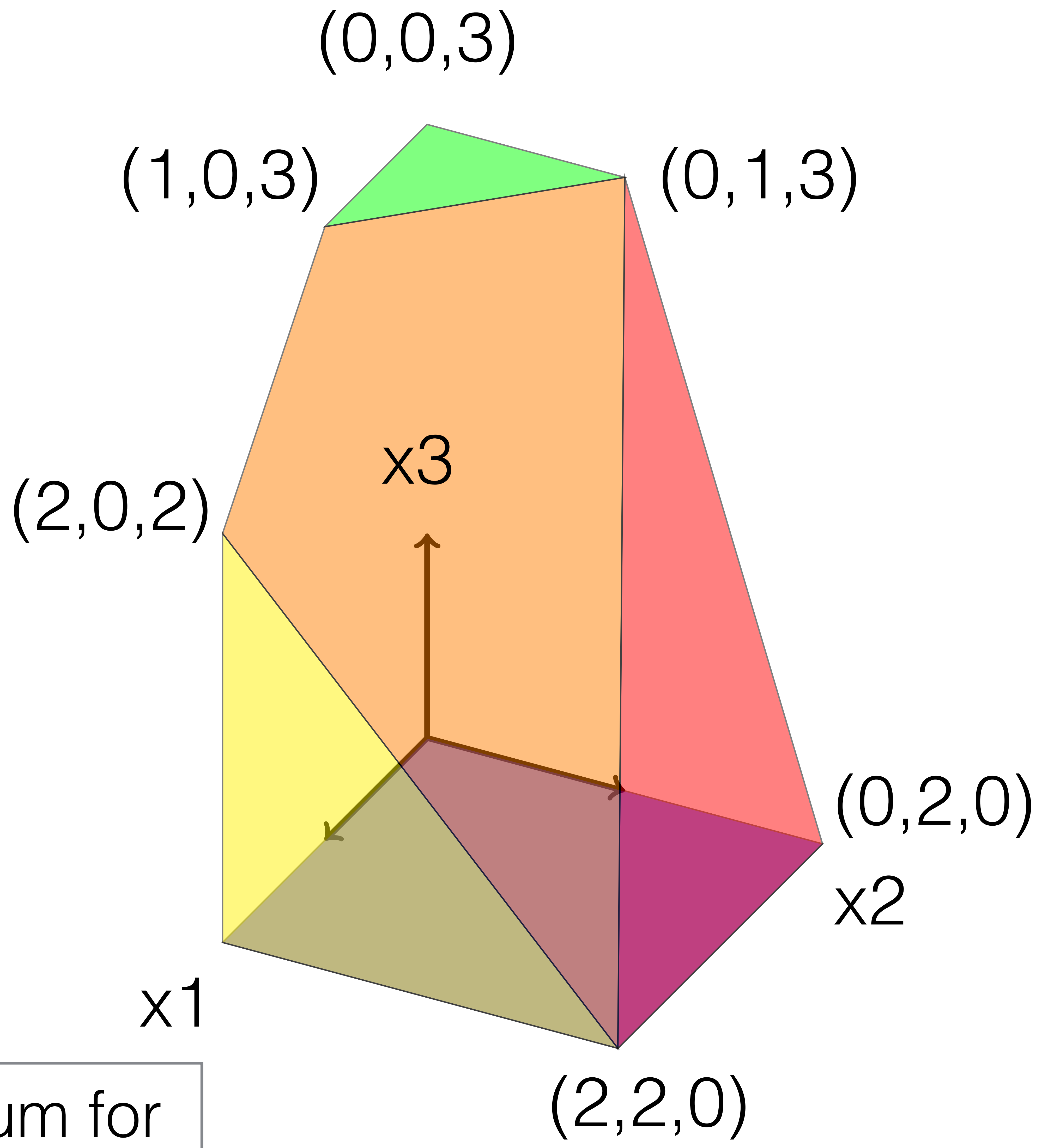


Brewer Problem: Objective Function



3D example, feasible region

$$x_1 + x_2 + x_3 \leq 4$$
$$x_1 \leq 2$$
$$x_3 \leq 3$$
$$3x_2 + x_3 \leq 6$$
$$x_1, x_2, x_3 \geq 0$$



Exercise: What would be optimum for

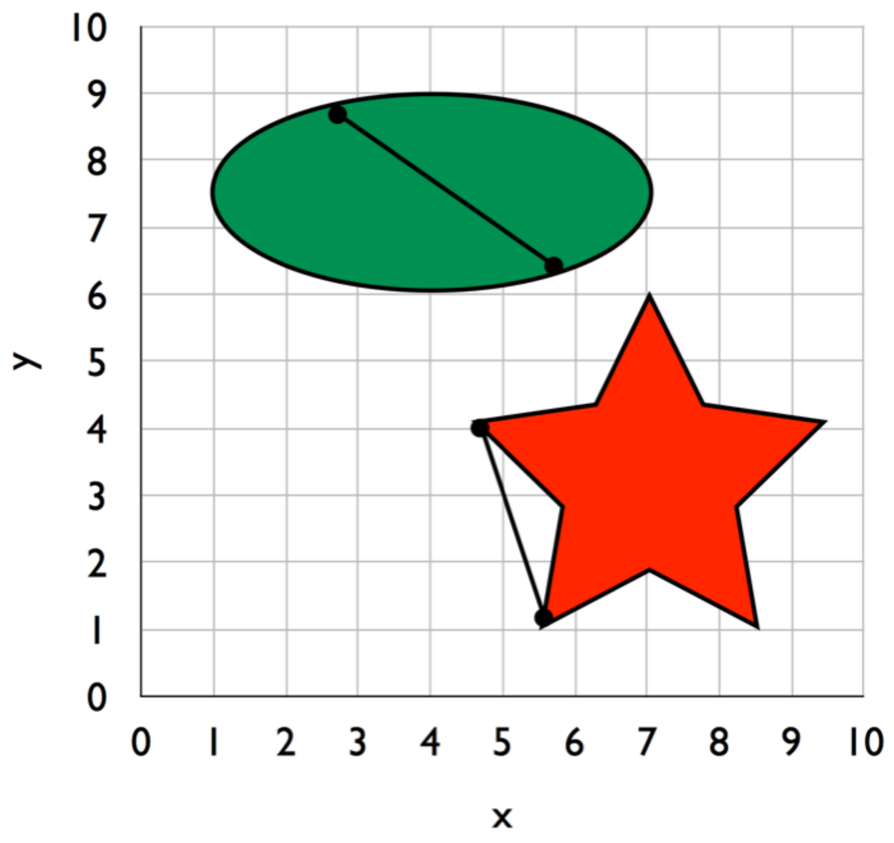
- maximize $x_1+x_2+x_3$?
- maximize x_1+x_2 ?

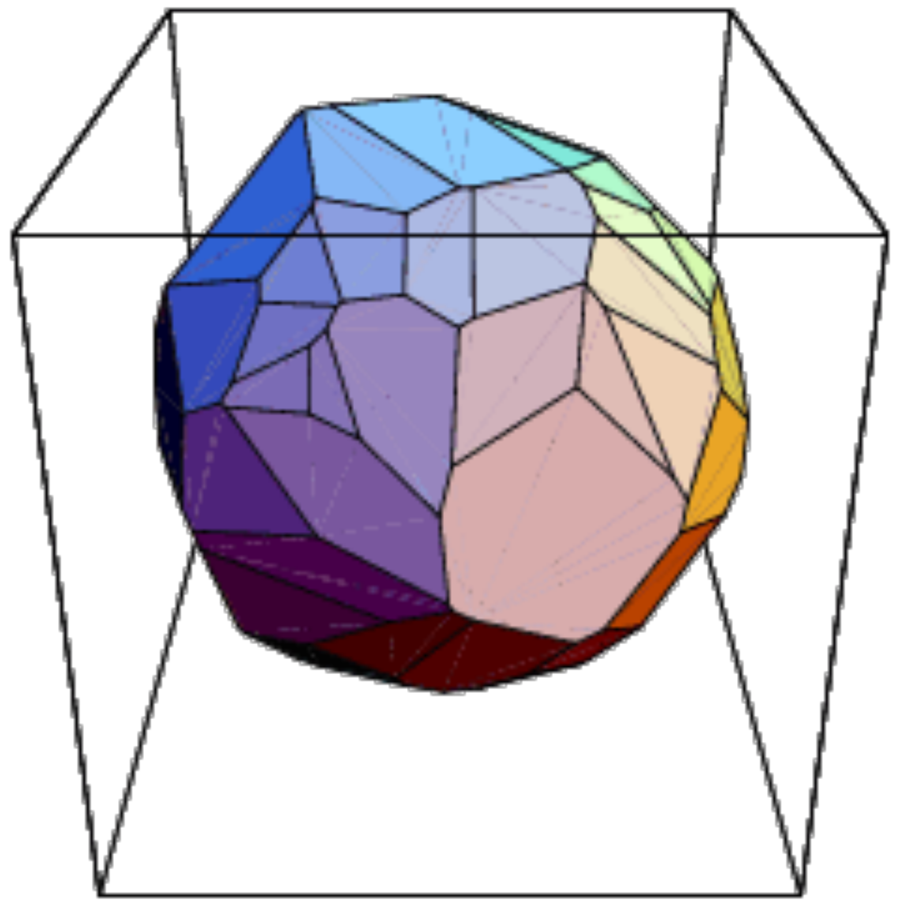
Convex Polytope

- The solution space of a linear system of linear equalities is a convex polytope.

- S is a set of points. S is convex iff for any point x and y in S, any convex combination is in S

$$(\alpha x + (1-\alpha)y) \in S \ with \ \alpha \in [0, 1]$$





Theorem: Every point in a polytope is a convex combination of its vertices

Optimality is at vertices

maximize $c_1 x_1 + \ldots + c_n x_n$

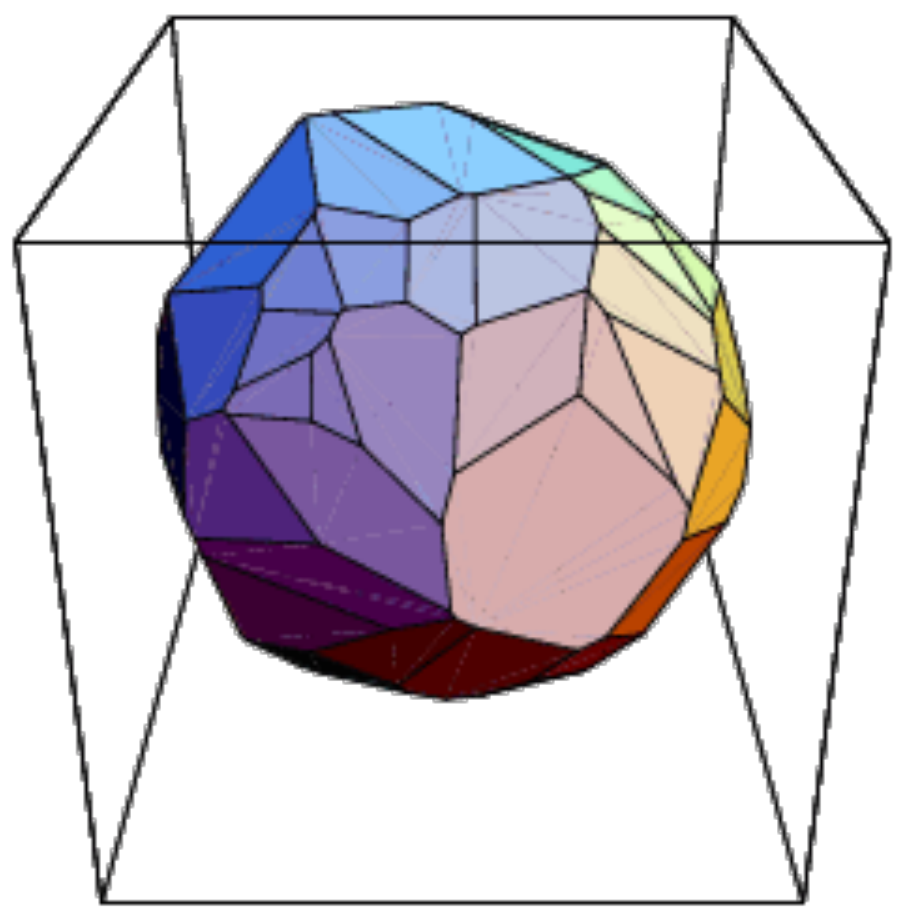
subject to

$$a_{11} x_1 + \ldots + a_{1n} x_n \leq b_1$$

$$\ldots$$

$$a_{m1} x_1 + \ldots + a_{mn} x_n \leq b_m$$

$$x_i \geq 0 \quad (1 \leq i \leq n)$$



Theorem: At least one of the points where the objective value is maximal is a vertex.

Optimality is at vertices (proof)

Theorem: At least one of the points where the objective value is maximal is a vertex.

Let x^* be the maximum. Since each point in a polytope is a convex combination of the vertices $v_1, \ldots, v_t$, we have

$$x^* = \lambda_1 v_1 + \ldots + \lambda_t v_t$$

and the objective value at optimality can be expressed as

scalar product (c is a vector)

$$cx^* = \lambda_1 * (cv_1) + \ldots + \lambda_t (cv_t).$$

Assume that the maximum is not at a vertex, i.e.,

$$cx^* > cv_i \quad \forall i : 1 \leq i \leq t.$$

each vertex is less good than x*

It follows that

$$\begin{aligned} cx^* &= \lambda_1 * (cv_1) + \ldots + \lambda_t (cv_t) \\ &< \lambda_1 * (cx^*) + \ldots + \lambda_t (cx^*) \\ &< (\lambda_1 + \ldots + \lambda_t)(cx^*) \\ &< cx^*. \end{aligned}$$

contradiction

Hence, it must be the case that $x^* = v_i$ for some $1 \leq i \leq t$.

A first Algorithm

- Enumerate all the vertices

- Select the one with the largest objective value

- What do you think about this ? 🧐

Number of of vertices

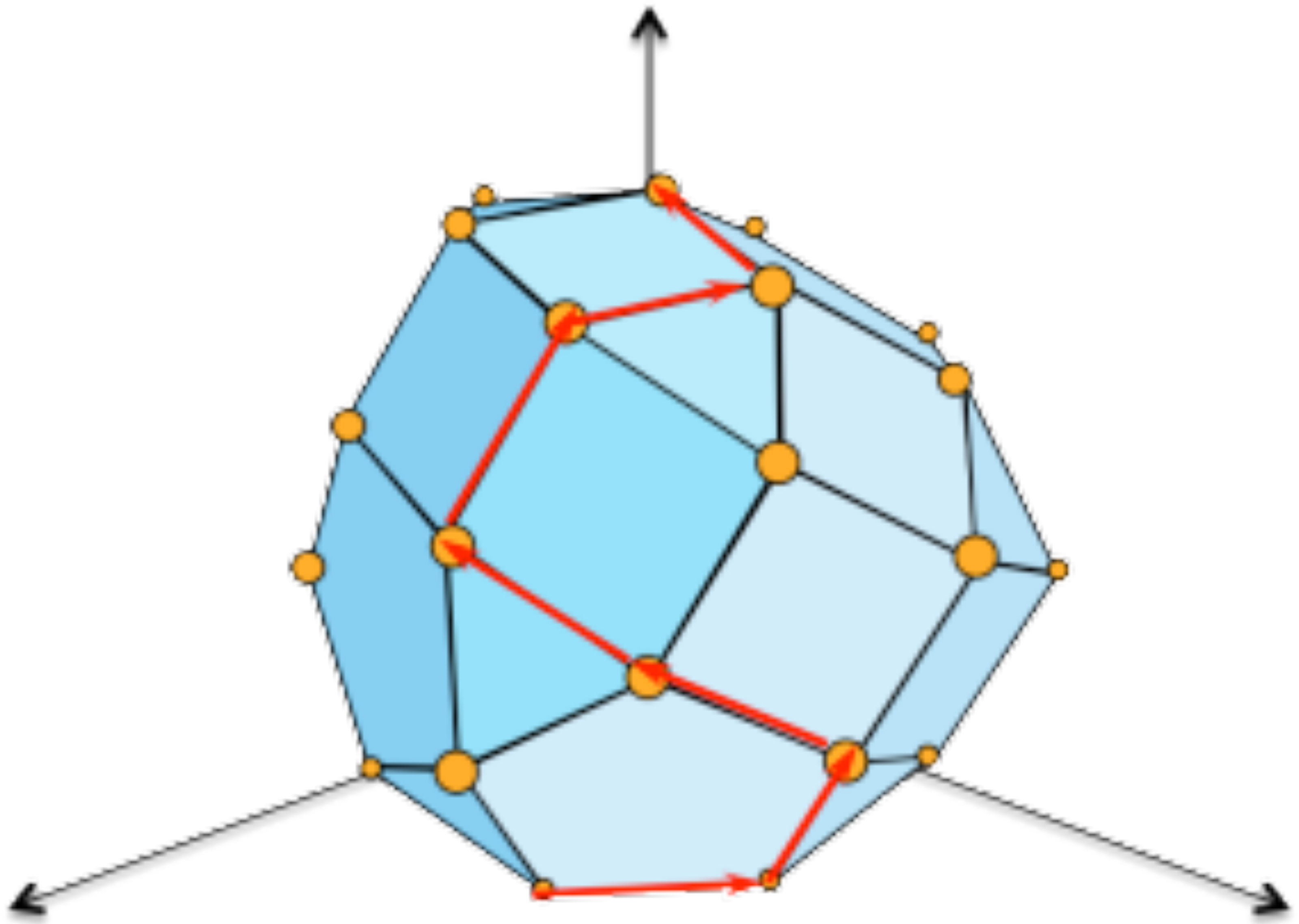
- In 1D, the unit interval [0, 1] has 2 endpoints. Intersection of 2 half-spaces.
- In 2D, the unit square has 4 vertices. Intersection of 4 half-spaces
- In 3D, the unit cube has 8 vertices, defined by the intersections of the 6 half-spaces
  ...
- In nD, the unit hypercube has 2^n vertices defined by the intersection of 2n half spaces.

The pattern is clear: The number of vertices can grow exponentially with the number of inequalities.

Conclusion: enumerating all the vertices is impractical 💣.

Solution: Simple Algorithm 🥳

Simplex Algorithm



- Move from one vertex, to a neighboring vertex with an improving objective function.
- Move until no more improving neighbor vertex
- An optimal vertex is always reached because of convexity of polytope

The different forms of Linear Programme

$$
\begin{array}{ll}
\text{maximize} & 2x_1 - 3x_2 \\
\text{subject to} & x_1 + x_2 = 7 \\
& x_1 - 2x_2 \leq 4 \\
& x_1 \geq 0
\end{array}
$$

equality constraint

inequality constraint

Non negativity of some variables but not all

- This is a bit messy, let's make a unique form, called ***canonical form***

Canonical Form (all constraints are <= and all vars non negative)

n variables, m constraints

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \text{ for } i = 1, 2, \ldots, m$$

$$x_j \geq 0 \text{ for } i = 1, 2, \ldots, n$$

non negativity constraints for all variables

💡Always possible to convert a linear program into canonical form

Converting into canonical form is an easy process

- If variable x_j has no non-negativity constraint replace each occurrence by

$$x'_j - x''_j$$

- Convert equality constraints (=) into two ($\geq$, $\leq$)

- Exercise: Convert this into standard form

$$\begin{aligned} \text{maximize} \quad & 2x_1 - 3x_2 \\ \text{subject to} \quad & x_1 + \ \ x_2 = 7 \\ & x_1 - 2x_2 \leq 4 \\ & x_1 \qquad\ \ \geq 0 \end{aligned}$$

Not yet the panacea

- For the solving, it is easier to deal with equality constraint.
- After all, we know how to solve system of linear equations (Gauss-Jordan)
- Let's create another form, called the ***standard form*** with
  - equality constraints only, and
  - non negativity constraints on all the variables

Standard form (only non-negative constraints are inequalities)

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \text{ for } i = 1, 2, \ldots, m$$

$$x_j \geq 0 \text{ for } i = 1, 2, \ldots, n$$

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j$$

$$x_{n+i} \geq 0$$

Introduce one slack variable/inequality
If slack = 0, we say the constraint is tight

n+m variables, m constraints

From Canonical Form to Standard Form

$$\begin{array}{ll} \text{maximize} & 2x_1 - 3x_2 \\ \text{subject to} & x_1 + x_2 = 7 \\ & x_1 - 2x_2 \leq 4 \\ & x_1 \geq 0 \end{array}$$

Canonical form

$$\begin{array}{ll} \text{maximize} & 2x_1 - 3x_2 + 3x_3 \\ \text{subject to} & x_1 + x_2 - x_3 \leq 7 \\ & -x_1 - x_2 + x_3 \leq -7 \\ & x_1 - 2x_2 + 2x_3 \leq 4 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

Standard form

$$\begin{array}{lrcrcrcrcr} \text{maximize} & & & & & 2x_1 & - & 3x_2 & + & 3x_3 \\ \text{subject to} & x_4 & = & 7 & - & x_1 & - & x_2 & + & x_3 \\ & x_5 & = & -7 & + & x_1 & + & x_2 & - & x_3 \\ & x_6 & = & 4 & - & x_1 & + & 2x_2 & - & 2x_3 \\ & & & & & & & & & x_1, x_2, x_3, x_4, x_5, x_6 >= 0 \end{array}$$

The Brewer Problem*

- A small brewery produces ale and beer and wants to maximize profit
- Production limited by scarce resources (corn, hops, malt)

corn (480 lbs)   hops (160 oz)   malt (1190 lbs)

- The recipe for ale and beer require different proportions of resources

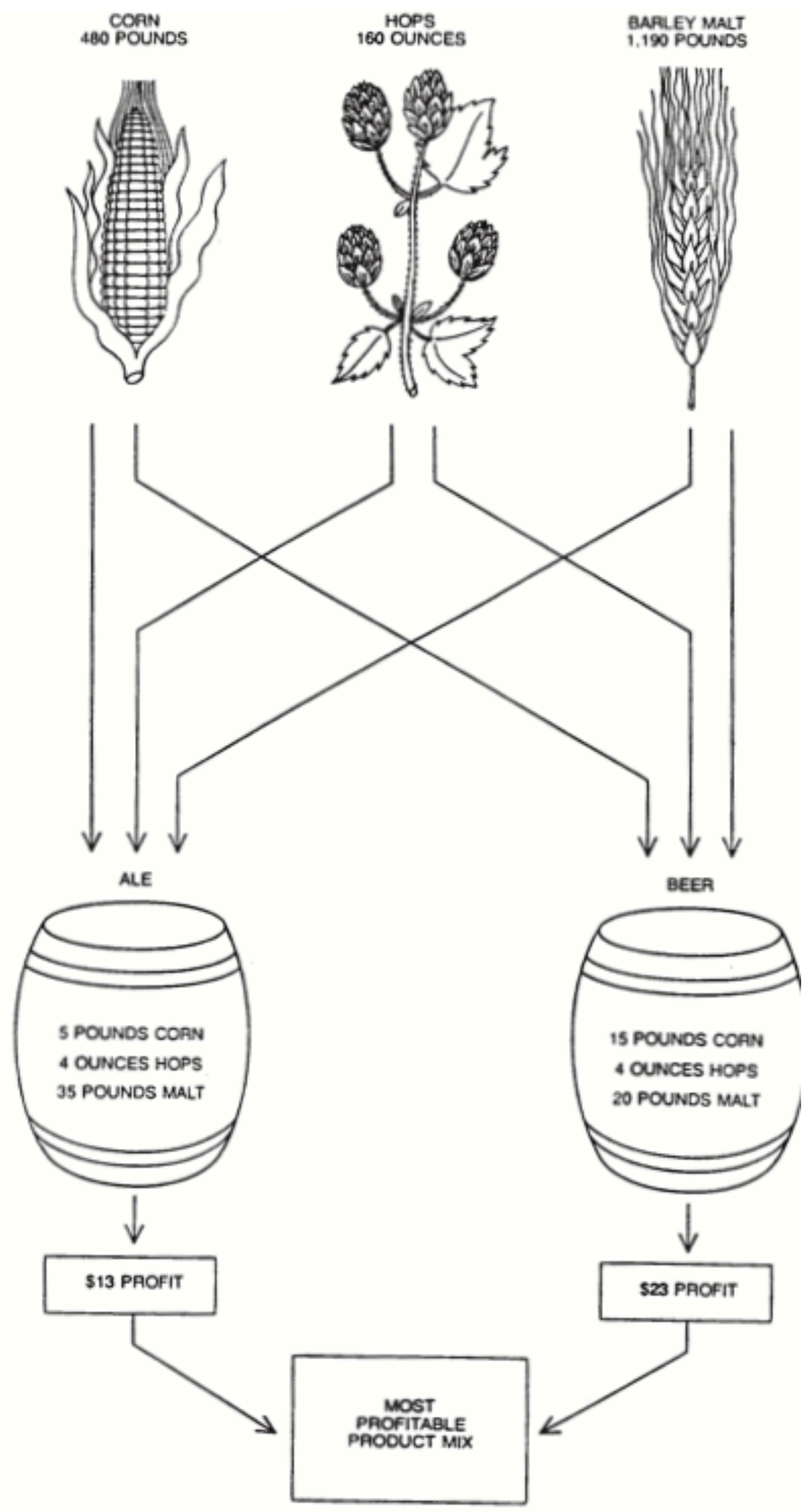
$13 profit per barrel   **$23 profit per barrel**

*Slides from Sedgewick and Wayne, Algorithms part 2, Coursera

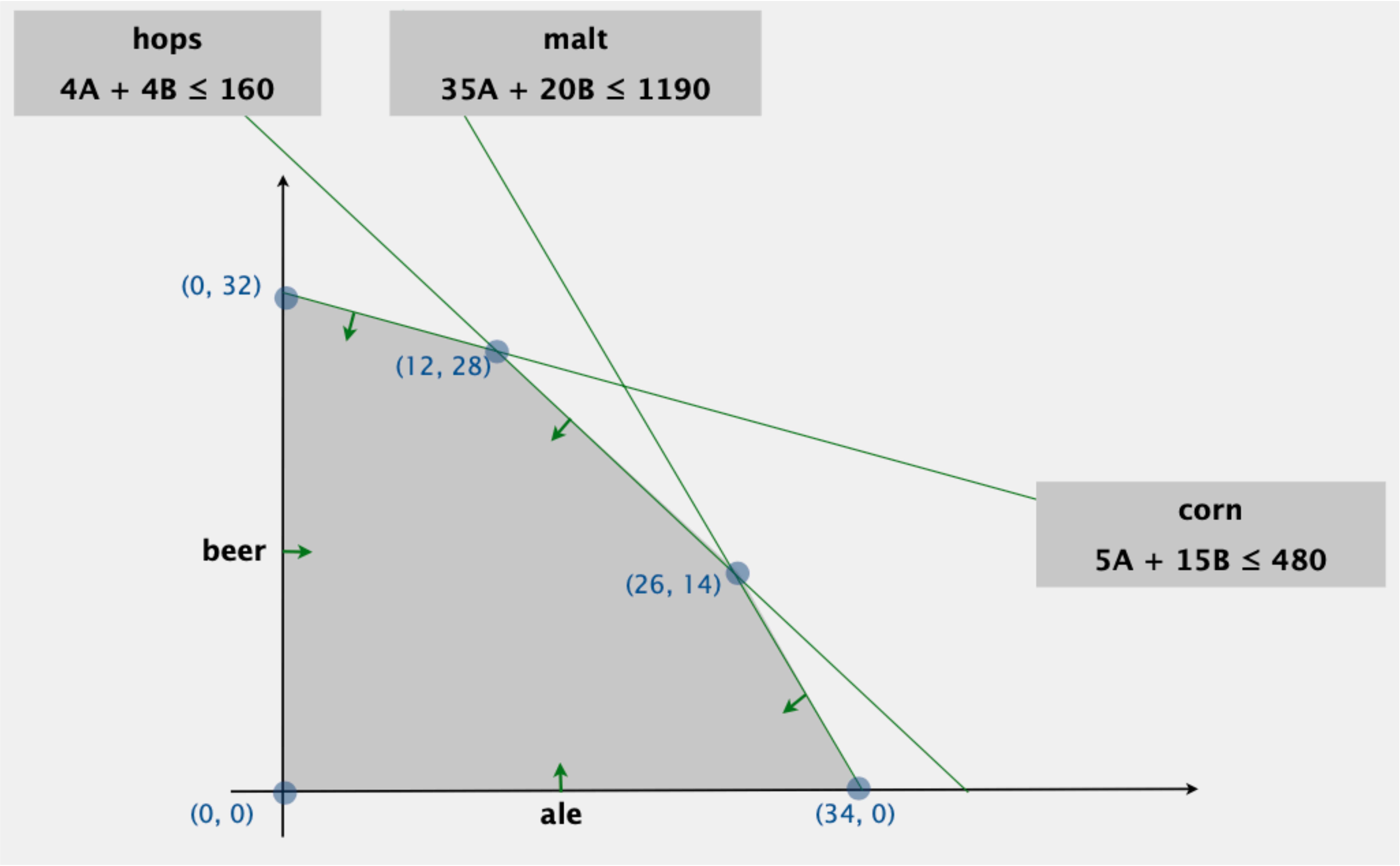
Brewer Problem: Linear Programming Formulation

- Variables: A = the number of barrels of ale, B = the number of barrels of beer.

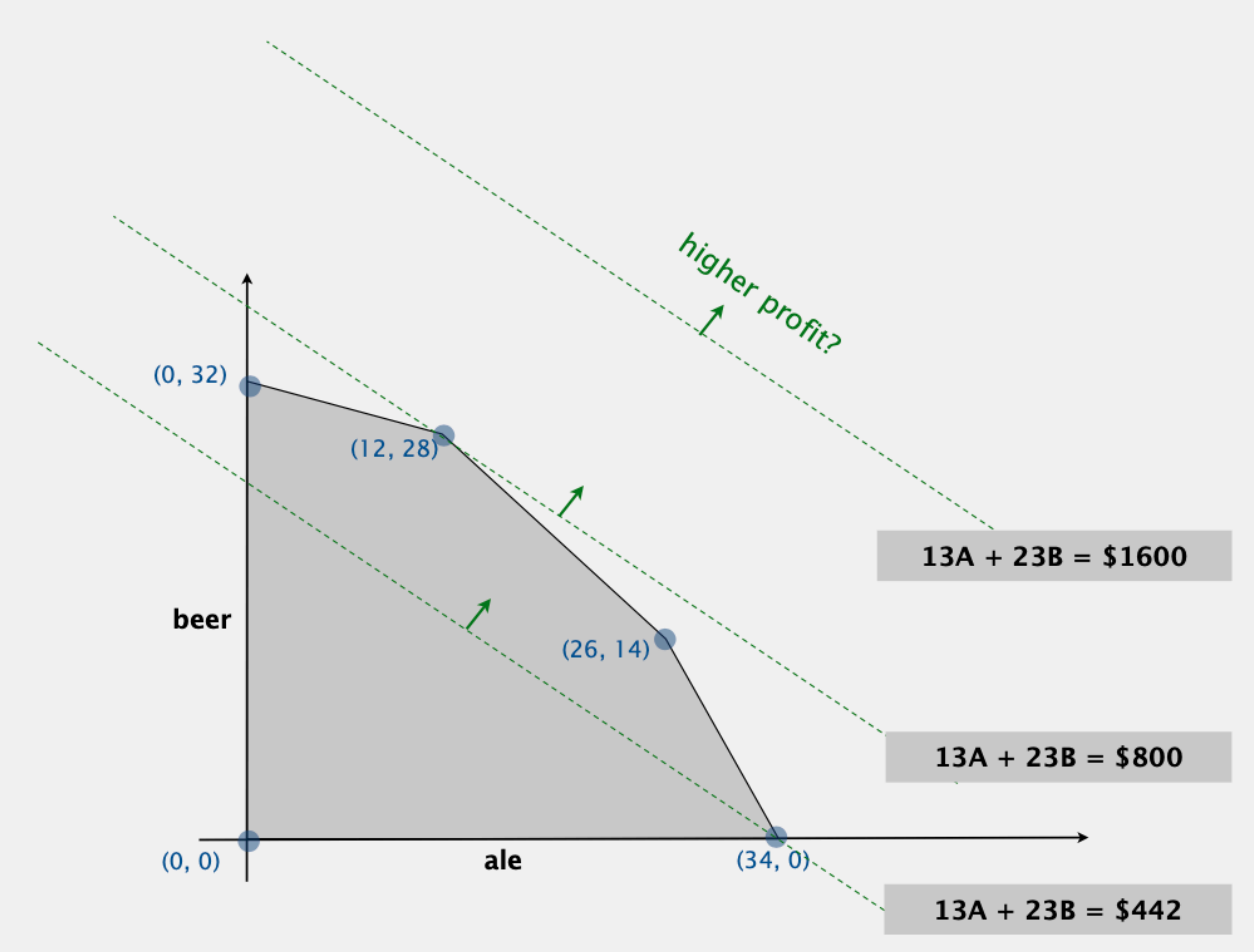
	ale		beer			
maximize	13A	+	23B			profits
subject to the constraints	5A	+	15B	≤	480	corn
	4A	+	4B	≤	160	hops
	35A	+	20B	≤	1190	malt
	A	,	B	≥	0	



Brewer Problem: Feasible Region



Brewer Problem: Objective Function



Brewer Problem: Standard Form

- Add three slack variables, S_C, S_H, S_M for the crop, hop and malt constraints

- Introduce variable Z for the objective function

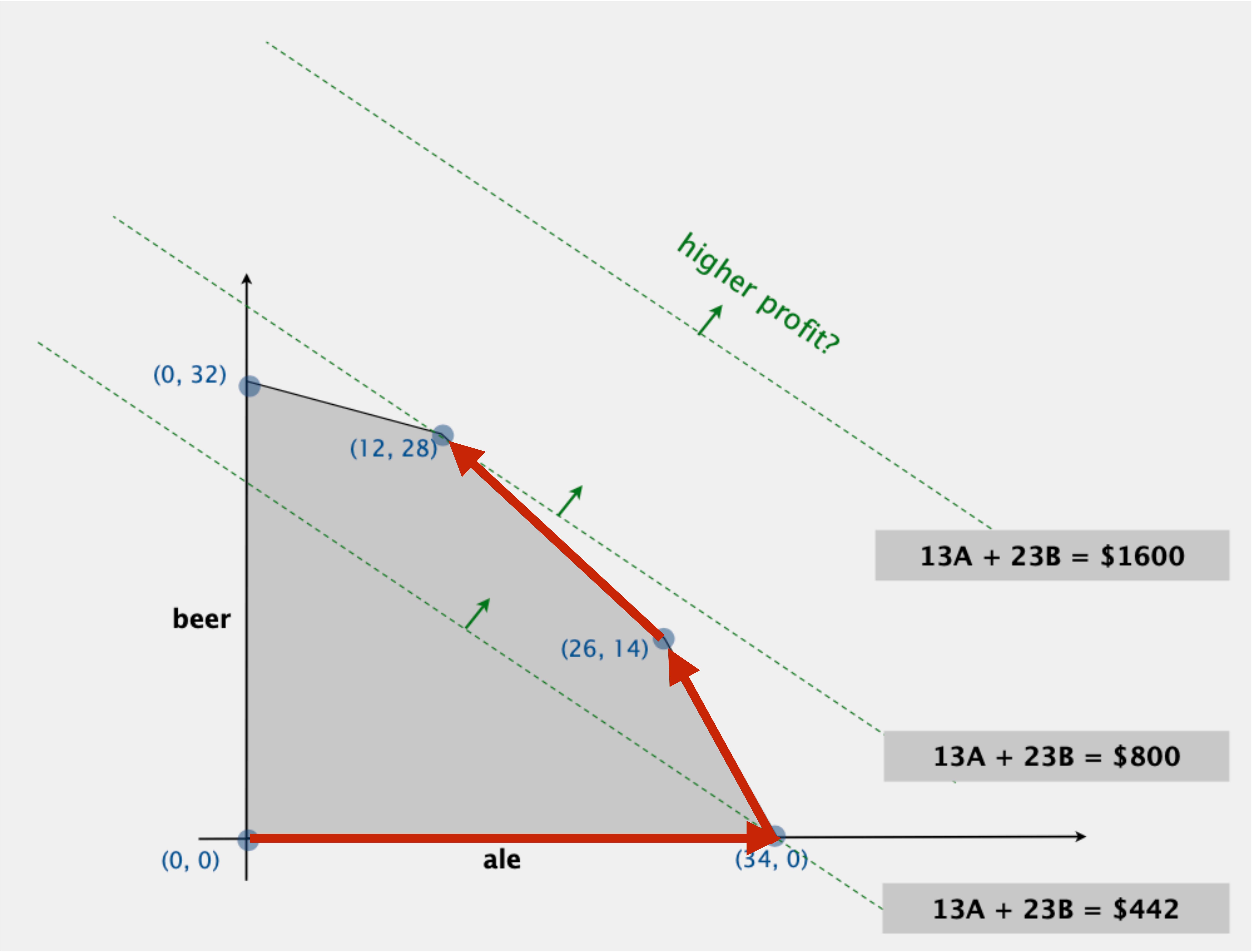
Canonical Form

$$
\begin{array}{llcclcl}
\text{maximize} & 13A & + & 23B & & \\
\text{subject to the constraints} & 5A & + & 15B & \le & 480 \\
 & 4A & + & 4B & \le & 160 \\
 & 35A & + & 20B & \le & 1190 \\
 & A & , & B & \ge & 0
\end{array}
$$

Standard Form

$$
\begin{array}{lcccccccccccccc}
\text{maximize} & Z & & & & & & & & & & & & \\
 & 13A & + & 23B & & & & & & & - & Z & = & 0 \\
\hline
\text{subject to the constraints} & 5A & + & 15B & + & S_C & & & & & & & = & 480 \\
 & 4A & + & 4B & & & + & S_H & & & & & = & 160 \\
 & 35A & + & 20B & & & & & + & S_M & & & = & 1190 \\
 & A & , & B & , & S_C & , & S_C & , & S_M & & & \ge & 0
\end{array}
$$

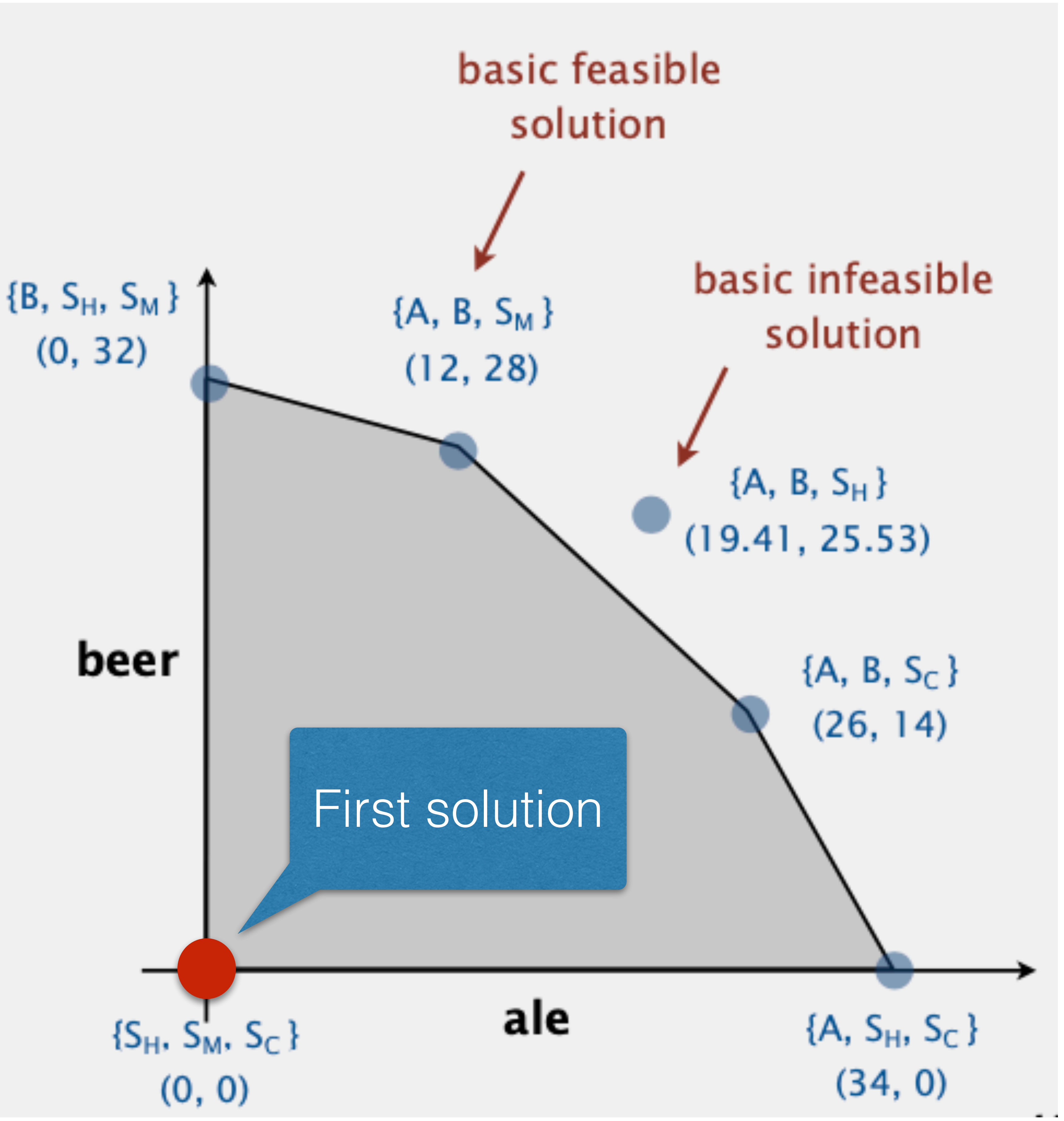
Vertices explored by the simplex algorithm



First Basic Feasible Solution

- A subset m=3 of the n=6 variables
- Only those (basic) variables are non zero
- It correspond to feasible solution

maximize	Z											
	13A	+	23B						– Z	=	0
subject to the constraints	5A	+	15B	+	S_C					=	480
	4A	+	4B			+	S_H			=	160
	35A	+	20B					+	S_M	=	1190
	A	,	B	,	S_C	,	S_H	,	S_M	≥	0



- First basis, start with slack variables {S_C, S_H, S_M} as the basis
- A and B = 0 {S_C=480, S_H,=160, S_M=119} can be read from the tableau, no algebra needed

Pivot 1

- Take one non basic variable and turn it into a basic variable to improve the solution
- Let us choose B to enter into the basis. If B increases, Z must increase.
- What variable does B replace ? Answer: S_C because if B increases, S_C, S_H and S_M must decrease but S_C will hit zero first.
- Min ratio rule (480/15=32, 160/4=40, 1190/20=59.5)

pivot

$$
\begin{array}{lrcrcccccrcr}
\text{maximize} & Z & & & & & & & & & & \\
 & 13A & + & 23B & & & & & & -\ Z & = & 0 \\
\hline
\text{subject to the constraints} & 5A & + & 15B & + & S_C & & & & & = & 480 \\
 & 4A & + & 4B & & & + & S_H & & & = & 160 \\
 & 35A & + & 20B & & & & & + \ S_M & & = & 1190 \\
 & A & , & B & , & S_C & , & S_H & , \ S_M & & \geq & 0
\end{array}
$$

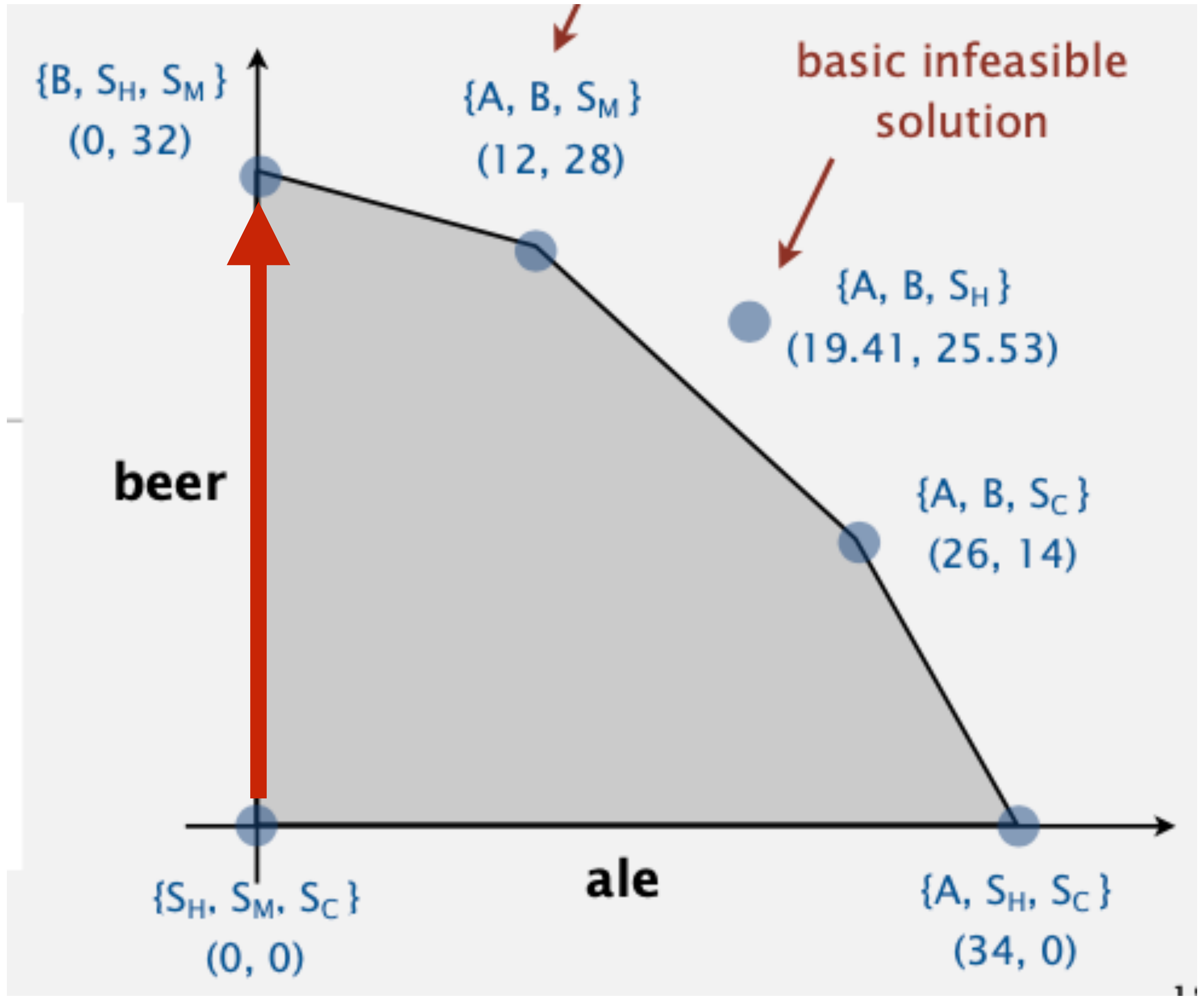
Pivot 1 (cont)

$$
\begin{array}{lrcrcccccrcr}
\text{maximize} & Z \\
& 13A & + & 23B & & & & & - & Z & = & 0 \\
\hline
\text{subject} & 5A & + & 15B & + & S_C & & & & & = & 480 \\
\text{to the} & 4A & + & 4B & & & + & S_H & & & = & 160 \\
\text{constraints} & 35A & + & 20B & & & & & + & S_M & = & 1190 \\
& A & , & B & , & S_C & , & S_H & , & S_M & \geq & 0
\end{array}
$$

pivot

Substitute $B = (1/15)(480 - 5A - S_C)$ and add B into the basis (rewrite 2nd equation, eliminate B in 1st, 3rd, and 4th equations)

$$
\begin{array}{lrcccccccrcr}
\text{maximize} & Z \\
& (16/3)A & & & - & (23/15)S_C & & & - & Z & = & -736 \\
\hline
\text{subject} & (1/3)A & + & B & + & (1/15)S_C & & & & & = & 32 \\
\text{to the} & (8/3)A & & & - & (4/15)S_C & + & S_H & & & = & 32 \\
\text{constraints} & (85/3)A & & & - & (4/3)S_C & & & + & S_M & = & 550 \\
& A & , & B & , & S_C & , & S_H & , & S_M & \geq & 0
\end{array}
$$



Pivot 2

maximize Z

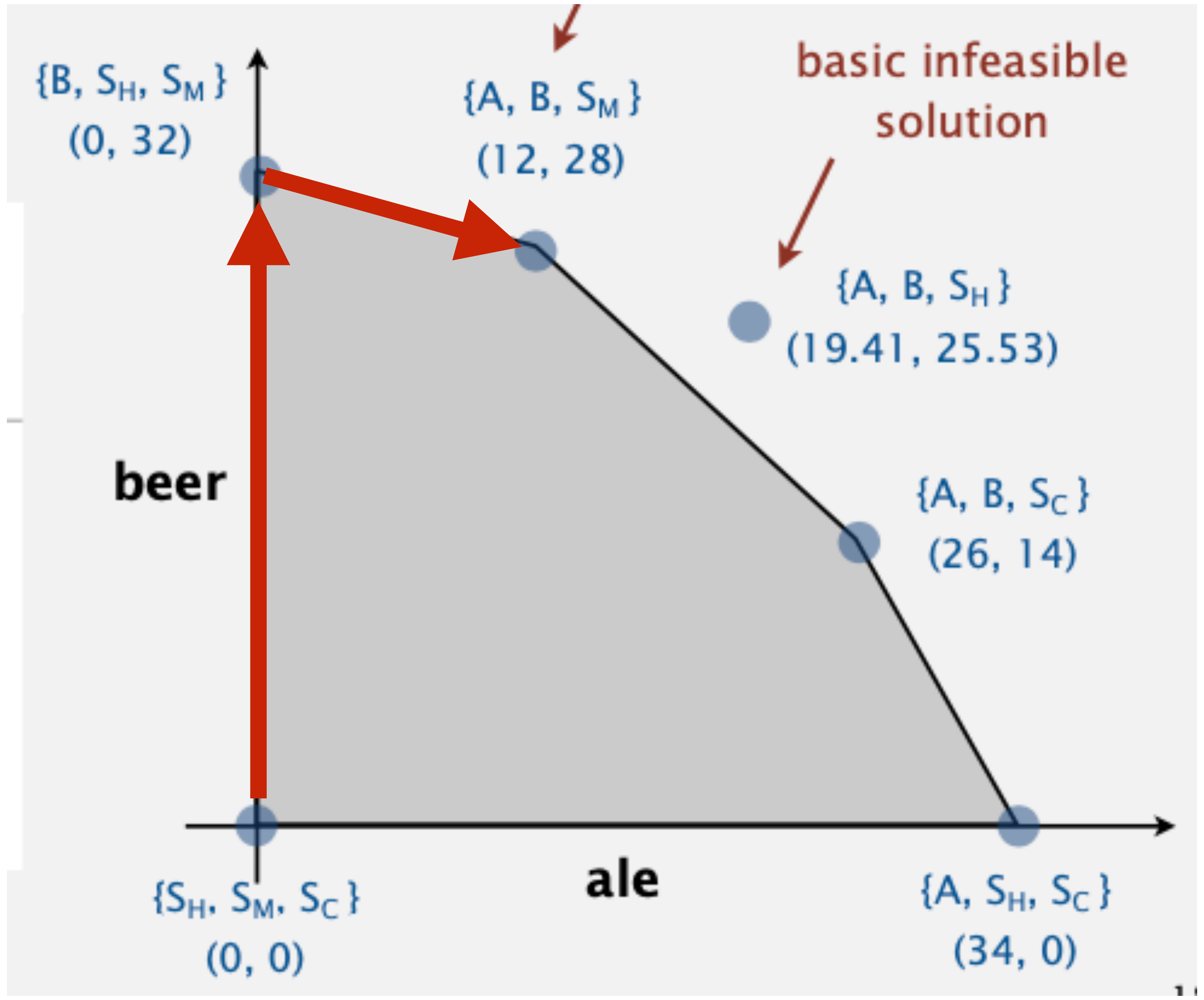
pivot

$$
\begin{array}{llllllllll}
\text{subject to the constraints} & (16/3)\,A & & - (23/15)\,S_C & & & -\,Z & = & -736 \\
\hline
 & (1/3)\,A & +\ B & + (1/15)\,S_C & & & & = & 32 \\
 & (8/3)\,A & & - (4/15)\,S_C & +\ S_H & & & = & 32 \\
 & (85/3)\,A & & - (4/3)\,S_C & & +\ S_M & & = & 550 \\
 & A\ , & B\ , & S_C\ , & S_H\ , & S_M & & \geq & 0
\end{array}
$$

Substitute $A = (3/8)\,(32 + (4/15)\,S_C - S_H)$ and add A into the basis (rewrite 3rd equation, eliminate A in 1st, 2nd, and 4th equations)

maximize Z

$$
\begin{array}{lllllll}
\text{subject to the constraints} & - S_C & - 2\,S_H & & -\,Z & = & -800 \\
\hline
 & B + (1/10)\,S_C & + (1/8)\,S_H & & & = & 28 \\
 & A - (1/10)\,S_C & + (3/8)\,S_H & & & = & 12 \\
 & - (25/6)\,S_C & - (85/8)\,S_H & +\ S_M & & = & 110 \\
 & A\ ,\ B\ ,\ S_C\ , & S_H\ , & S_M & & \geq & 0
\end{array}
$$



Optimality

- Basis = {A,B,S_M}, Z = 800, A = 12, B = 28
- Optimal solution. Stop pivoting when no objective coefficient is positive. Why

$$
\begin{array}{llcl}
\text{maximize} & Z & & \\
 & -\ S_C - 2\,S_H - Z & = & -800 \\
\text{subject to the constraints} & B + (1/10)\,S_C + (1/8)\,S_H & = & 28 \\
 & A - (1/10)\,S_C + (3/8)\,S_H & = & 12 \\
 & -\ (25/6)\,S_C - (85/8)\,S_H + S_M & = & 110 \\
 & A,\ B,\ S_C,\ S_H,\ S_M & \geq & 0
\end{array}
$$

- Any feasible solution satisfies the system of equations.
- In particular: $Z = 800 - S_C - 2S_H$. Since $S_C, S_H \geq 0$, $Z^* \leq 800$
- Current BFS has Z=800, thus it is optimal

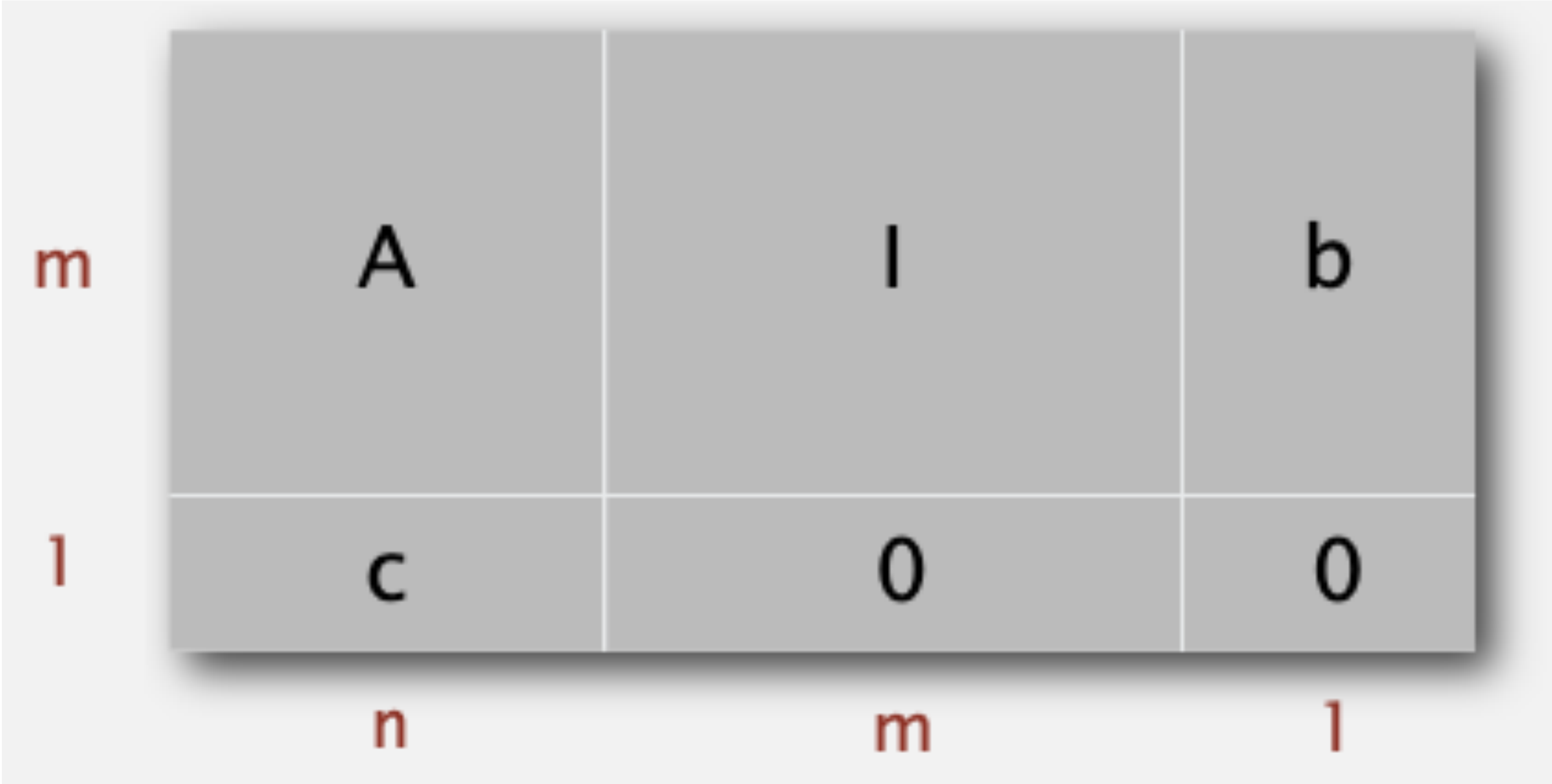
Let’s translate this into Java (using 2D arrays)

$$
\begin{array}{llllllllll}
\text{maximize} & Z & & & & & & & & \\
 & 13A & + & 23B & & & & -\ Z & = & 0 \\
\text{subject} & 5A & + & 15B & +\ S_C & & & & = & 480 \\
\text{to the} & 4A & + & 4B & & +\ S_H & & & = & 160 \\
\text{constraints} & 35A & + & 20B & & & +\ S_M & & = & 1190 \\
 & A & , & B & ,\ S_C & ,\ S_H & ,\ S_M & & \geq & 0
\end{array}
$$

m+1

5	15	1	0	0	480
4	4	0	1	0	160
35	20	0	0	1	1190
13	23	0	0	0	0

n+m+1



Initialization of 2D Array

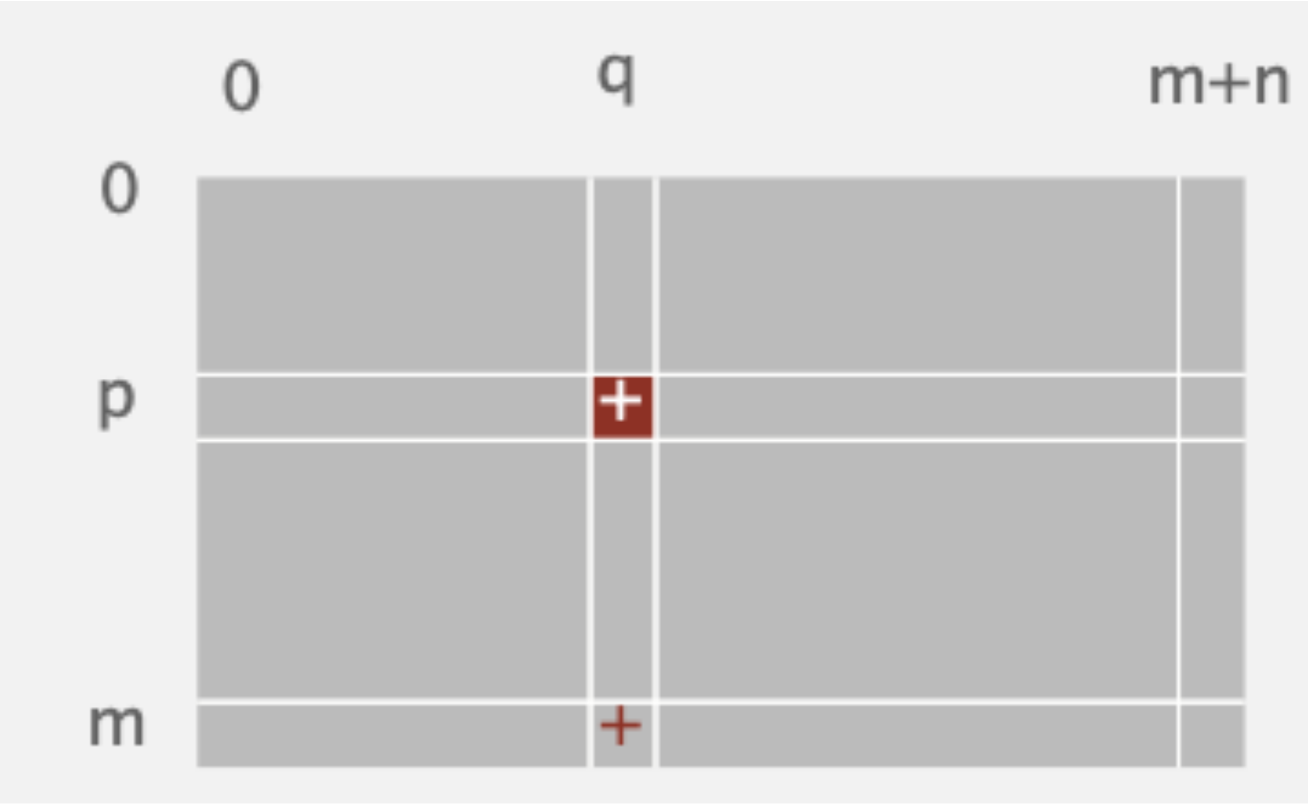
```
public class Simplex {

    private double[][] a;   // tableaux
    private int m;          // number of constraints
    private int n;          // number of original variables


    public Simplex(double[][] A, double[] b, double[] c) {

        m = b.length;
        n = c.length;

        for (int i = 0; i < m; i++)
            if (!(b[i] >= 0)) throw new IllegalArgumentException("RHS must be nonnegative");

        a = new double[m+1][n+m+1];
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                a[i][j] = A[i][j];
        for (int i = 0; i < m; i++)
            a[i][n+i] = 1.0;
        for (int j = 0; j < n; j++)
            a[m][j] = c[j];
        for (int i = 0; i < m; i++)
            a[i][m+n] = b[i];


    }
}
```

Cannonical Form: max cx st: Ax <= b, x>=0
With b >= 0

Bland Rule: What variable to enter the basis for next pivot ?

Find entering column *q* using Bland's rule: index of first column whose objective function coefficient is positive.

```
// lowest index of a non-basic column with a positive cost
private int bland() {
    for (int j = 0; j < m+n; j++)
        if (a[m][j] > 0) return j;
    return -1;  // optimal
}
```



Min Ration rule: What variable should exist the BFS

```
// find row p using min ratio rule (-1 if no such row)
// (smallest such index if there is a tie)
private int minRatioRule(int q) {
    int p = -1; // leaving row
    for (int i = 0; i < m; i++) {
        if (a[i][q] <= 0) continue; // only positive entries
        else if (p == -1) p = i;
        else if ((a[i][m+n] / a[i][q]) < (a[p][m+n] / a[p][q]))
            p = i; // row p has min ration so far
    }
    return p;
}
```

Pivoting on row p, column q

```
private void pivot(int p, int q) {

    // everything but row p and column q
    for (int i = 0; i <= m; i++)
        for (int j = 0; j <= m+n; j++)
            if (i != p && j != q) a[i][j] -= a[p][j] * (a[i][q] / a[p][q]);

    // zero out column q
    for (int i = 0; i <= m; i++)
        if (i != p) a[i][q] = 0.0;

    // scale row p
    for (int j = 0; j <= m+n; j++)
        if (j != q) a[p][j] /= a[p][q];
    a[p][q] = 1.0;
}
```

Min loop of the simplex

```java
private void solve() {
    while (true) {

        // find entering column q
        int q = bland();
        if (q == -1) break;  // optimal

        // find leaving row p
        int p = minRatioRule(q);
        if (p == -1) throw new ArithmeticException("Linear program is unbounded");

        // pivot
        pivot(p, q);
    }
}
```

But we were very lucky here 😮‍💨

- Because it was very easy to find a first BFC

$$
\begin{array}{lrcrcccccl}
\text{maximize} & Z & & & & & & & & \\
 & 13A & + & 23B & & & & -Z & = & 0 \\
\text{subject to the constraints} & 5A & + & 15B & + S_C & & & & = & 480 \\
 & 4A & + & 4B & & + S_H & & & = & 160 \\
 & 35A & + & 20B & & & + S_M & & = & 1190 \\
 & A & , & B & , S_C & , S_H & , S_M & & \geq & 0
\end{array}
$$

If any of those coefficient was negative, we wouldn't have been able to start with this BFS

Two Phase-Simplex

- Phase 1: find a BFS (using pivoting, with modified objective)

- Phase 2: optimize original objective starting with BFS of phase 1

Finding an initial BFS

$$
\begin{array}{lrcrcr}
\text{maximize} & 2x_1 & - & x_2 & & \\
\text{subject to} & & & & & \\
& 2x_1 & - & x_2 & \leq & 2 \\
& x_1 & - & 5x_2 & \leq & -4 \\
& & x_1, x_2 & & \geq & 0
\end{array}
$$

Standard form is (unfortunately not a BFS):

$$
\begin{array}{rclcrcr}
z & = & & & 2x_1 & - & x_2 \\
x_3 & = & 2 & - & 2x_1 & + & x_2 \\
x_4 & = & -4 & - & x_1 & + & 5x_2
\end{array}
$$

$$x_1, x_2, x_3, x_4 >= 0$$

Finding an initial BFS

- Put all the variables to the right such that you have

- constraints of type [0= bi+…variables…] with bi >=0

$$
\begin{array}{rcrcrcr}
z & = & & & 2x_1 & - & x_2 \\
x_3 & = & 2 & - & 2x_1 & + & x_2 \\
x_4 & = & -4 & - & x_1 & + & 5x_2
\end{array}
$$

$$x_1, x_2, x_3, x_4 >= 0$$

$$\Downarrow$$

$$
\begin{array}{rcrcrcrcrcr}
z & = & & & 2x_1 & - & x_2 & & & & \\
0 & = & 2 & - & 2x_1 & + & x_2 & - & x_3 & & \\
0 & = & 4 & + & x_1 & - & 5x_2 & & & + & x_4
\end{array}
$$

$$x_1, x_2, x_3, x_4 >= 0$$

Finding an initial BFS

$$
\begin{array}{rcrcrcrcrcr}
z & = & & & 2x_1 & - & x_2 & & & & \\
0 & = & 2 & - & 2x_1 & + & x_2 & - & x_3 & & \\
0 & = & 4 & + & x_1 & - & 5x_2 & & & + & x_4
\end{array}
$$

$$x_1, x_2, x_3, x_4 >= 0$$

- Replace 0 in each constraint with a new fresh variable and minimize their sum. By construction you have a BFS for this modified problem so you can use simplex Algo to start optimizing it.
- If you arrive at z=0 you have a BFS to the initial problem, otherwise initial problem is unfeasible

$$
\begin{array}{rcrcrcrcrcrcc}
z & = & & & & & & & & & & -x_5 & -x_6 \\
x_5 & = & 2 & - & 2x_1 & + & x_2 & - & x_3 & & & & \\
x_6 & = & 4 & + & x_1 & - & 5x_2 & & & + & x_4 & &
\end{array}
$$

$$x_1, x_2, x_3, x_4, x_5, x_6 >= 0$$

Finding an initial BFS

$$
\begin{aligned}
z &= && && && -x_5 - x_6 \\
x_5 &= 2 - 2x_1 + x_2 - x_3 \\
x_6 &= 4 + x_1 - 5x_2 + x_4 \\
& x_1, x_2, x_3, x_4, x_5, x_6 >= 0
\end{aligned}
$$

initial objective

$- 2x_1 - 1x_2$

$$
\begin{aligned}
z &= -6 + 1x_1 + 4x_2 + 1x_3 - x_4 \\
x_5 &= 2 - 2x_1 + x_2 - x_3 \\
x_6 &= 4 + x_1 - 5x_2 + x_4
\end{aligned}
$$

Summary: Simplex is a two step method

1. Find a BFS
   - by solving a modified problem with Simplex for which it is easy to find a BFS

2. Optimize starting from the BFS

Two phase simplex

```
public class TwoPhaseSimplex {

    private double[][] a;   // tableaux
    private int m;          // number of constraints
    private int n;          // number of original variables

    public TwoPhaseSimplex(double[][] A, double[] b, double[] c) {
        m = b.length;
        n = c.length;
        a = new double[m+2][n+m+m+1];

        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                a[i][j] = A[i][j];
        for (int i = 0; i < m; I++) a[i][n+i] = 1.0;
        for (int i = 0; i < m; I++) a[i][n+m+m] = b[i];
        for (int j = 0; j < n; j++) a[m][j] = c[j];
        // if negative RHS, multiply by -1
        for (int i = 0; i < m; i++) {
            if (b[i] < 0) {
                a[i][n+m+m] = -b[i];
                for (int j = 0; j <= n; j++)
                    a[i][j] = -a[i][j];
                a[i][n+i] = -1.0;
            }
        }
        // artificial variables form initial basis
        for (int i = 0; i < m; i++)
            a[i][n+m+i] = 1.0;
        for (int i = 0; i < m; i++)
            a[m+1][n+m+i] = -1.0;
        for (int i = 0; i < m; i++)
            pivot(i, n+m+i);

        basis = new int[m];
        for (int i = 0; i < m; i++)
            basis[i] = n + m + i;
    }
}
```

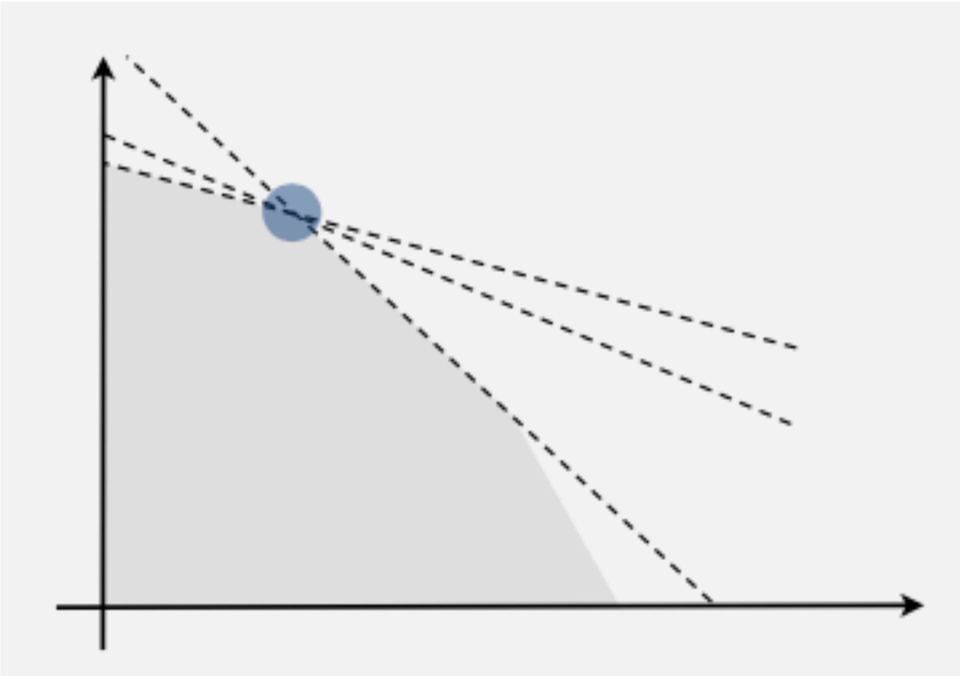
The two phases

```
private void phase1() {
    while (true) {
        // find entering column q
        int q = bland1();
        if (q == -1) break;  // optimal
        // find leaving row p
        int p = minRatioRule(q);
        assert p != -1 : "Entering column = " + q;

        // pivot
        pivot(p, q);
    }
    if (a[m+1][n+m+m] > 0)
     throw new ArithmeticException("Linear program is infeasible");
}

private void phase2() {
    while (true) {

        // find entering column q
        int q = bland2();
        if (q == -1) break;  // optimal

        // find leaving row p
        int p = minRatioRule(q);
        if (p == -1)
         throw new ArithmeticException("Linear program is unbounded");

        // pivot
        pivot(p, q);
    }
}
```

```
// lowest index of a non-basic column with a positive cost - using
artificial objective function
private int bland1() {
    for (int j = 0; j < n+m; j++)
        if (a[m+1][j] > EPSILON) return j;
    return -1;  // optimal
}

// lowest index of a non-basic column with a positive cost
private int bland2() {
    for (int j = 0; j < n+m; j++)
        if (a[m][j] > EPSILON) return j;
    return -1;  // optimal
}
```

Important remarks on computation

- Degeneracy: new basis, same extreme point (stalling quite frequent, don’t worry too much)



- Cycling: get stuck by cycling through different bases that all correspond to same extreme point. If you use Bland rule, you are guaranteed to terminate 🥳

Simplex Time Complexity (wikipedia)

The simplex method is remarkably efficient in practice and was a great improvement over earlier methods such as Fourier–Motzkin elimination. However, in 1972, Klee and Minty gave an example showing that the **worst-case complexity of simplex method as formulated by Dantzig is exponential time.** Since then, for almost every variation on the method, it has been shown that there is a family of linear programs for which it performs badly. It is an open question if there is a variation with polynomial time, or even sub-exponential worst-case complexity.

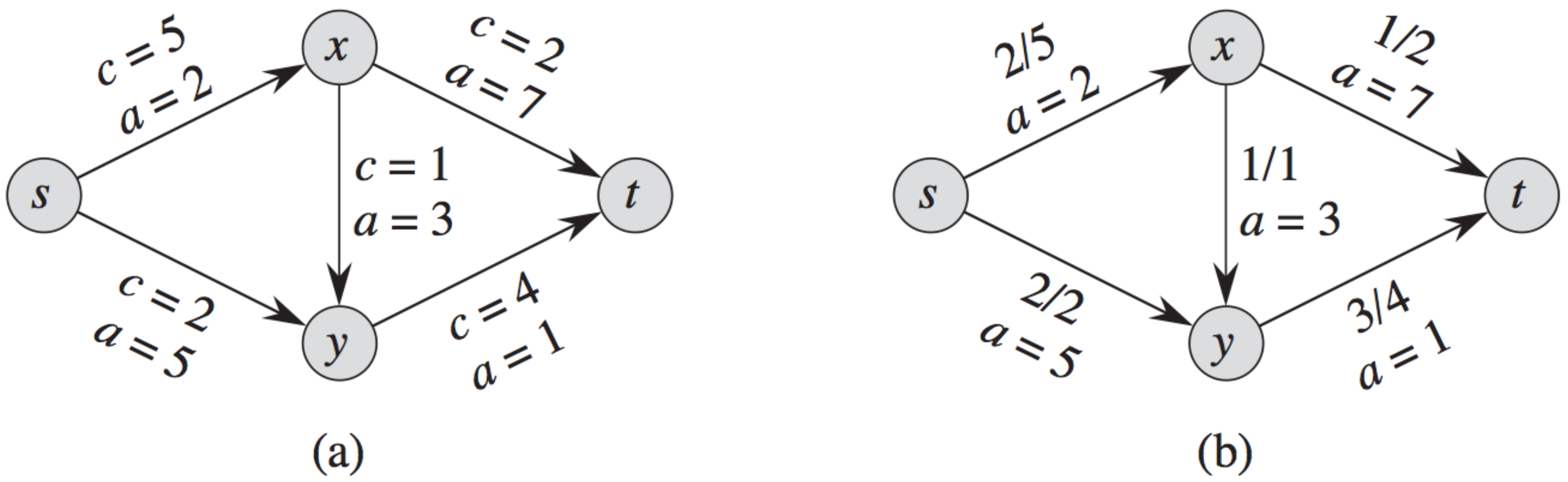
The simplex algorithm has **polynomial-time average-case complexity under various probability distributions**, with the precise average-case performance of the simplex algorithm depending on the choice of a probability distribution for the random matrices.

But LP solving is not NP hard, polynomial algorithms exist (Ellipsoid, Interior points). Those polynomial time algorithms are not necessarily better than simplex (and also less incremental).

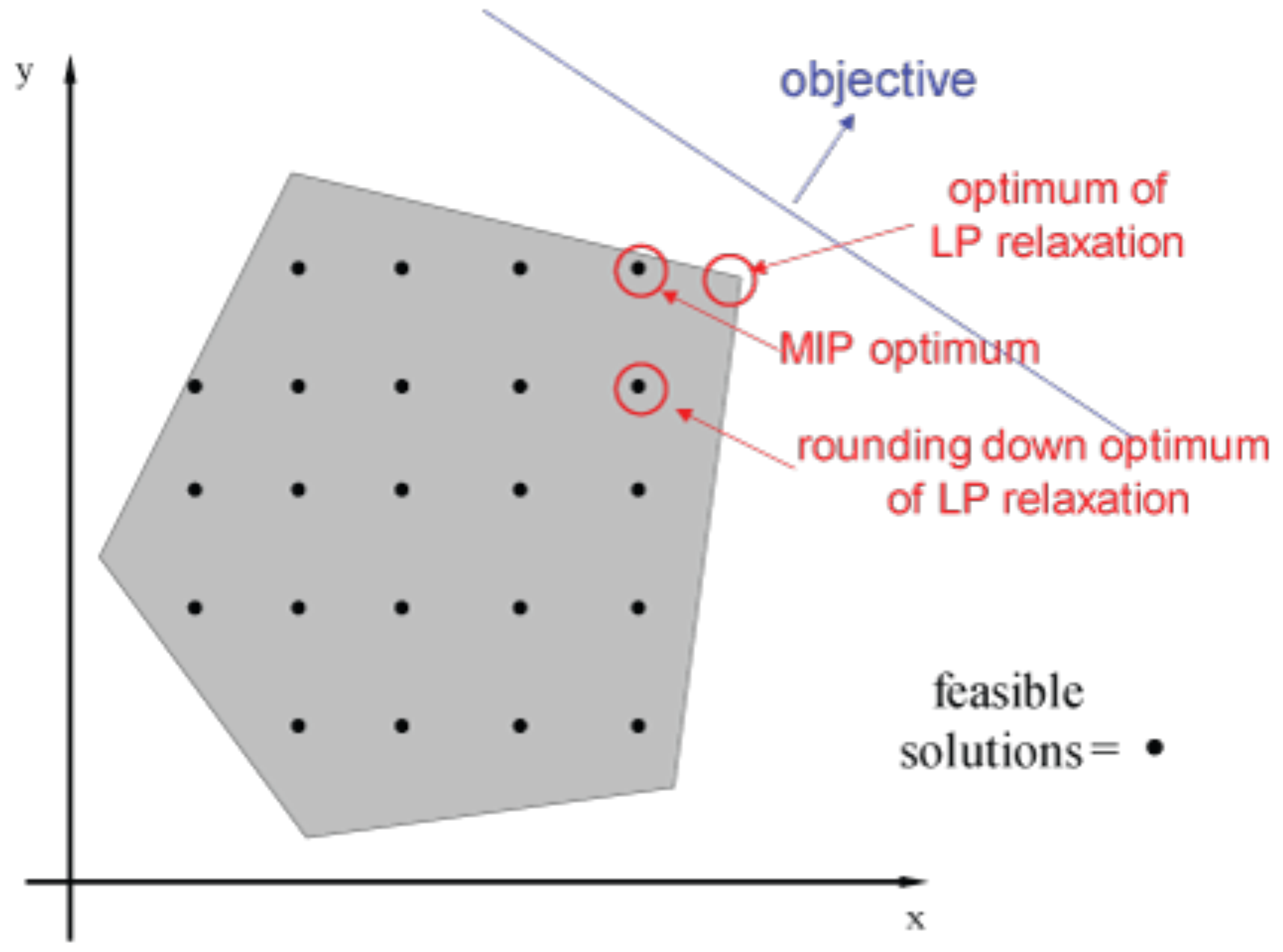
Thus anytime you can reduce your problem to an LP, you know it can be solved in polynomial time. Example: Network Flow Problems

Example of problem well solved by LP

- Minimum Cost Flow
  - accommodate d unit of flow from s to t at minimum cost without exceeding the capacity.



Integer Linear Programming (NP-Hard)



Integer Program

maximize $\sum_{j=1}^{n} c_j x_j$

subject to $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ for $i = 1, 2, \ldots, m$

$$x_j \in \cancel{\mathbb{N}} \text{ for } i = 1, 2, \ldots, n$$

$\mathbb{R}$

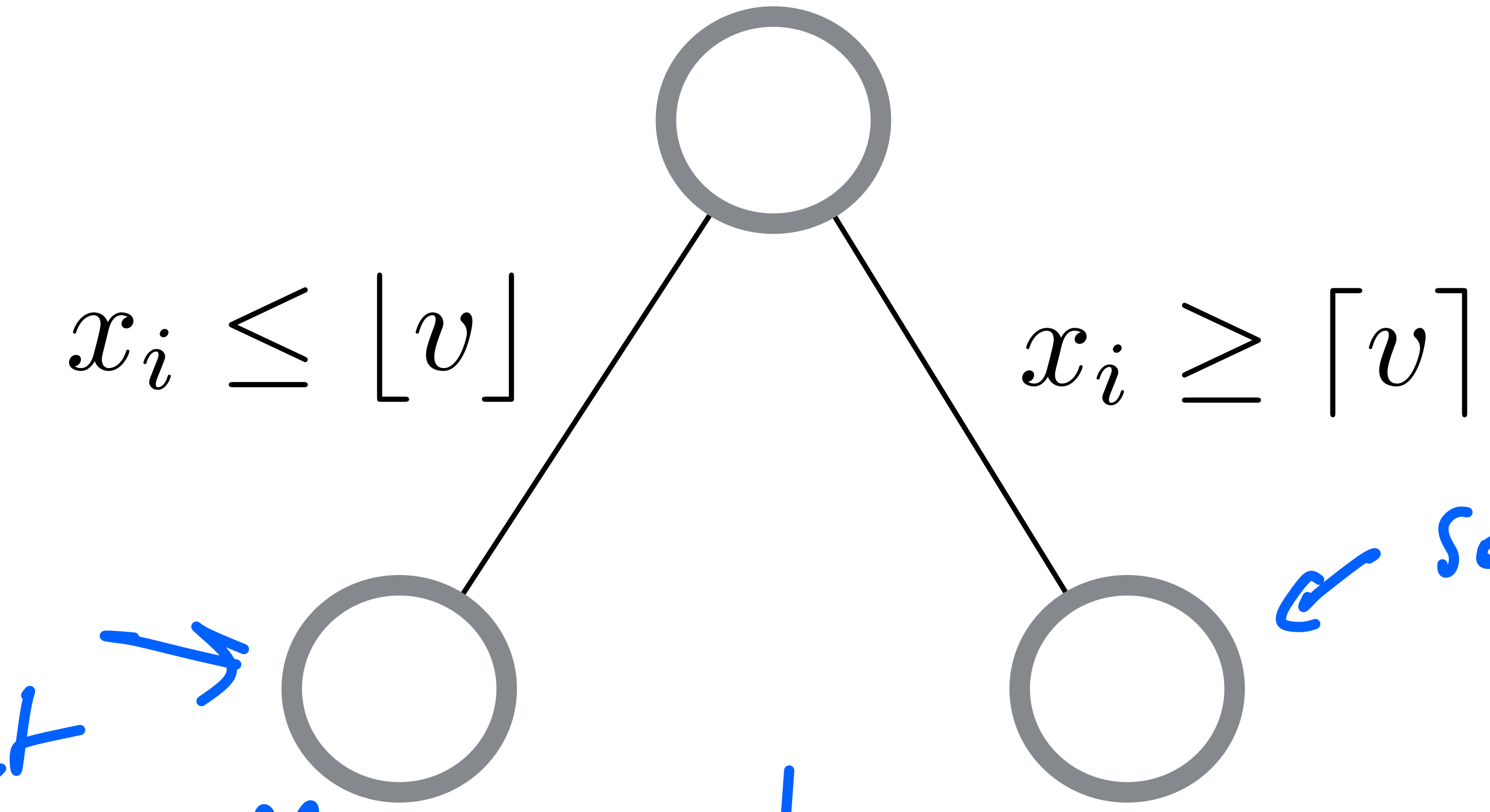
How to solve using Branch & Bound

What do you suggest as relaxation/upper-bound?

Branch and Bound with LP relaxation

- If at the optimal solution of the linear programming relaxation, one variable is not an integer $x_i = v$

- Create two branches



And solve the LP with the new constraint

same here

Lower bound will increase!

- Adding those constraints can only decrease the upper bound (pruning of upper bound < best-so-far feasible solution)

ILP Branch and Bound DFS

```
function ILP_Solver(A, b, c):
    return BranchAndBoundDFS(A, b, c, -inf, [])

function BranchAndBoundDFS(A, b, c, bestValue, bestSolution):
    // Solve the LP relaxation
    solution, value = Solve_Simplex(A, b, c)
    // If no feasible solution, return
    if solution == null:
        return bestValue, bestSolution
    // If solution is integer, update bestValue and bestSolution if necessary
    if IsInteger(solution):
        if value > bestValue:
            bestValue = value
            bestSolution = solution
        return bestValue, bestSolution
    // Otherwise, branch on a non-integer variable
    variableToBranch = FindNonIntegerVariable(solution)
    floorValue = floor(solution[variableToBranch])
    ceilValue = ceil(solution[variableToBranch])
    // Add constraints to fix the variable at its floor value and solve the resulting problem
    A1, b1 = AddConstraint(A, b, variableToBranch, "<=", floorValue)
    bestValue, bestSolution = BranchAndBoundDFS(A1, b1, c, bestValue, bestSolution)

    // Add constraints to fix the variable at its ceiling value and solve the resulting problem
    A2, b2 = AddConstraint(A, b, variableToBranch, ">=", ceilValue)
    bestValue, bestSolution = BranchAndBoundDFS(A2, b2, c, bestValue, bestSolution)
    return bestValue, bestSolution
```

Simplex Inventor

George Dantzig

November 8, 1914 – May 13, 2005

Dantzig's roles in the discovery of LP and the simplex method are intimately linked with the historical circumstances, notably the Cold War and the early days of the Computer Age.
Richard Cottle, Ellis Johnson, and Roger Wets

Exercise

- Optimize this current LP to optimality
- How do you know it is optimal?

$$
\begin{array}{rcrcrcrcr}
z & = & 20 & + & 2x_1 & - & x_2 & - & x_3 \\
x_4 & = & 25 & + & x_1 & - & x_2 & + & 3x_3 \\
x_5 & = & 12 & - & 2x_1 & - & 3x_2 & - & 4x_3 \\
x_6 & = & 15 & - & 3x_1 & + & x_2 & - & 4x_3 \\
\end{array}
$$

$$x_1, x_2, x_3, x_4, x_5, x_6 >= 0$$

Exercise

- Transform this into slack form and find (and) explain how to find an initial BFS.

$$\begin{aligned} \max\ & 4x_1 + x_2 - x_3 \\ & x_1 \qquad\quad + 3x_3 \le 6 \\ & 3x_1 + x_2 + 3x_3 \le 9 \\ & x_1 \ge 0,\ x_2 \ge 0,\ x_3 \ge 0. \end{aligned}$$

Best software for MIP and LP are commercial

- In practice, people rarely implement the simplex themself because to make it efficient and robust you need to take advantage of scarcity, you need to deal with mathematical error and stability (artful engineering)

- Best tools are commercial softwares, free for Universities and students

Example Gurobi Model in Python

```
# Create optimization model
m = Model('netflow')

# Create variables
flow = m.addVars(commodities, arcs, obj=cost, name="flow")

# Arc capacity constraints
m.addConstrs(
    (flow.sum('*',i,j) <= capacity[i,j] for i,j in arcs), "cap")



# Flow conservation constraints
m.addConstrs(
    (flow.sum(h,'*',j) + inflow[h,j] == flow.sum(h,j,'*')
    for h in commodities for j in nodes), « node")

# Compute optimal solution
m.optimize()

# Print solution
if m.status == GRB.Status.OPTIMAL:
    solution = m.getAttr('x', flow)
```

For the project you will

- Implement a new form of initialization
- Model the network flow problem with LP and compare it with a dedicated algorithm (next week)