Waterloo ML + Security + Verification Workshop

Machine Learning for Integer Programming

Elias B. Khalil ekhalil.com

Postdoc

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

→

Assistant Professor of Industrial Engineering

UNIVERSITY OF TORONTO

starting July 2020

In Memoriam: Shabbir Ahmed

- Anderson-Interface Chair and professor in Georgia Tech's H. Milton Stewart School of Industrial and Systems Engineering (ISyE)

- Giant of Stochastic Optimization and Integer Optimization

Learning to Run Heuristics in Tree Search

Elias B. Khalil[1], Bistra Dilkina*[1], George L. Nemhauser[2], Shabbir Ahmed[2], Yufen Shao[3]

Learning to Solve Large-Scale Security-Constrained Unit Commitment Problems

Álinson S. Xavier[1], Feng Qiu[1], and Shabbir Ahmed[2]

Humans learn to **design** **algorithms**.

Humans learn to **design algorithms**.

Can **algorithms** "learn" to **design algorithms**?

Humans learn to **design algorithms**.

Machine Learning

Can **algorithms** “learn” to **design algorithms**?

Humans learn to **design algorithms**.

Machine Learning

Can **algorithms** "learn" to **design algorithms**?

Discrete Optimization

Data Center Resource Management

Photo from: https://www.reit.com/what-reit/reit-sectors/data-center-reits

Data Center Resource Management

 Photo from: https://www.reit.com/what-reit/reit-sectors/data-center-reits

Data Center Resource Management

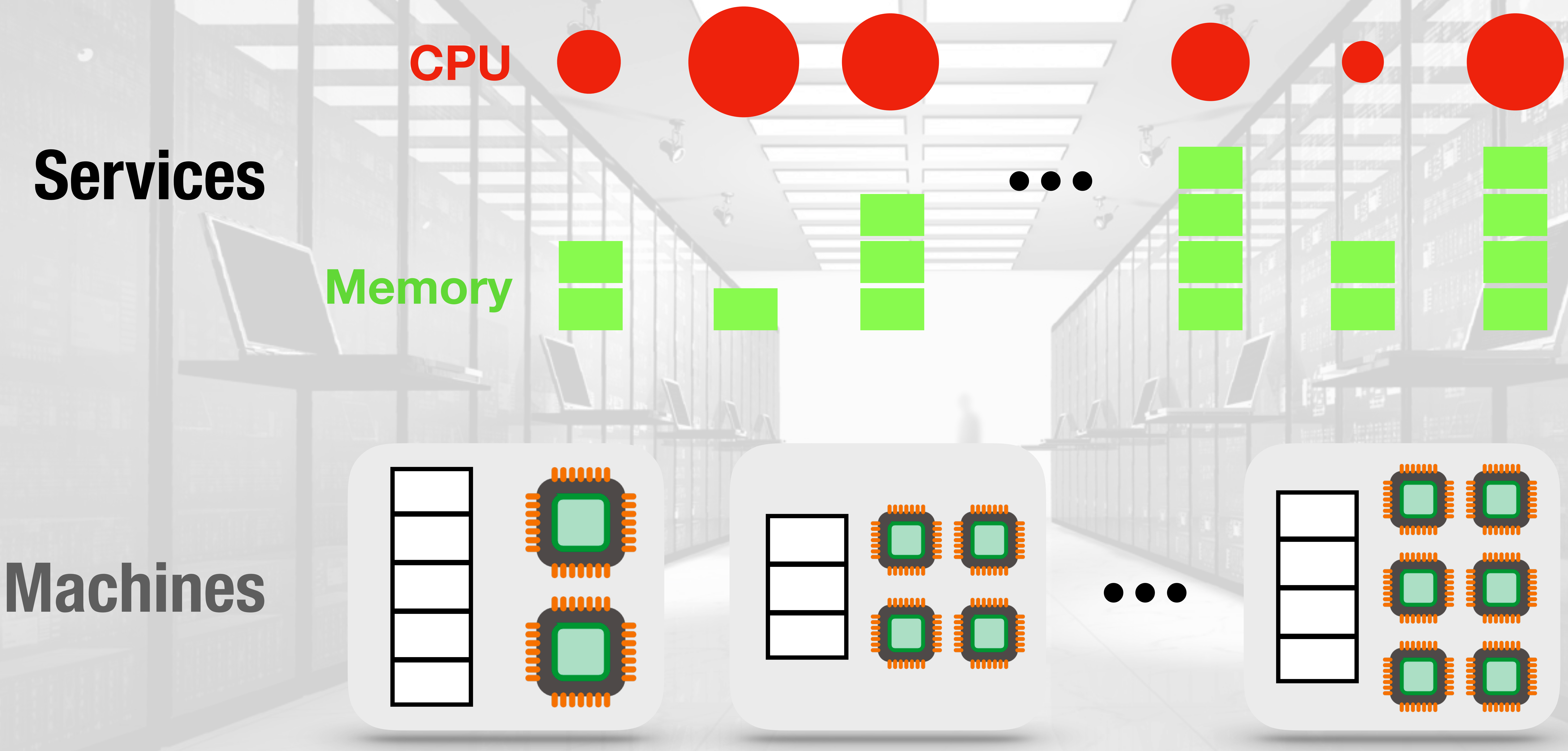


 Photo from: https://www.reit.com/what-reit/reit-sectors/data-center-reits

Data Center Resource Management

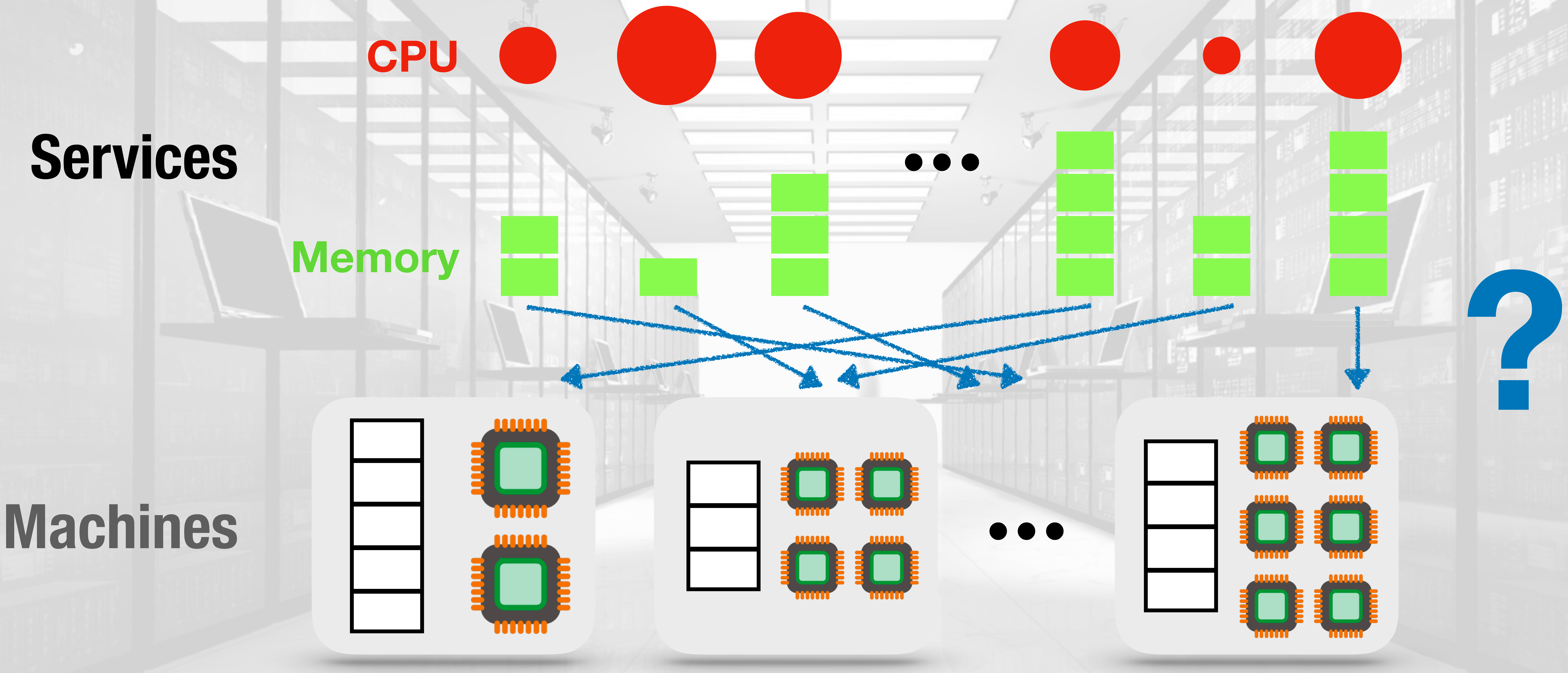
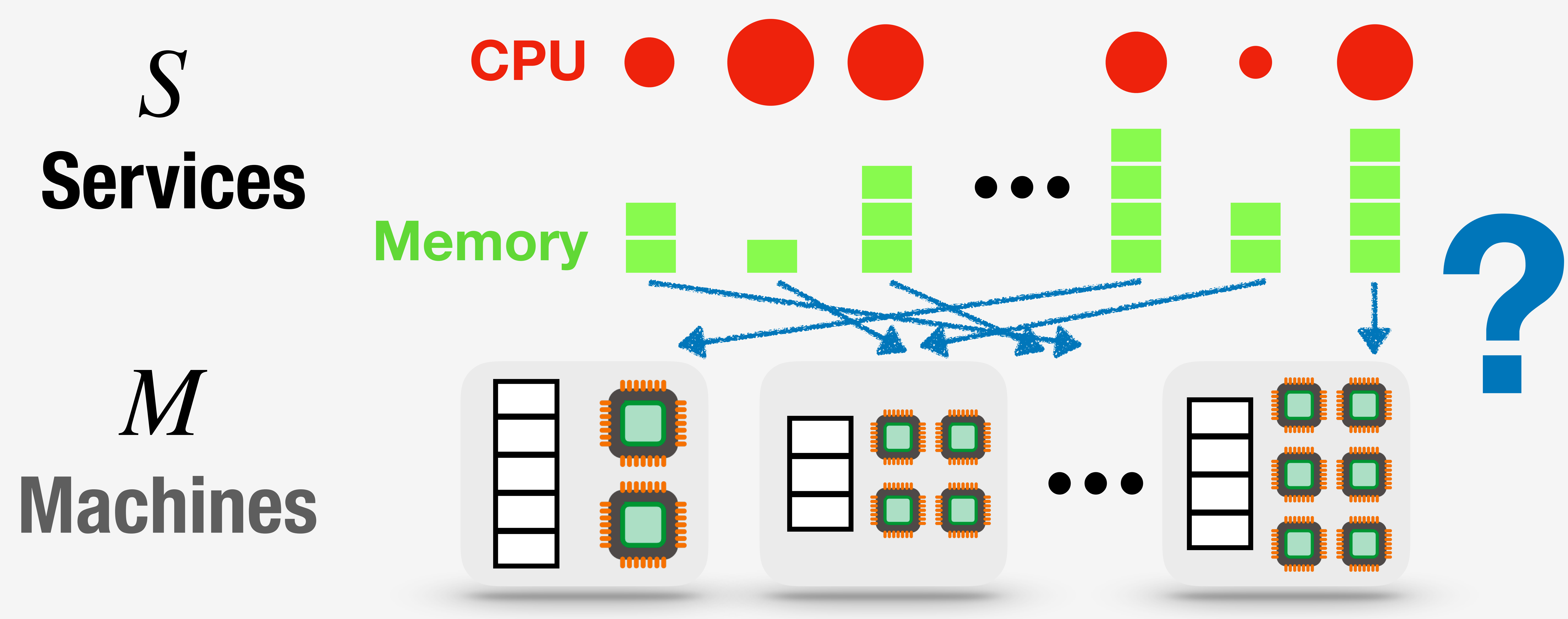
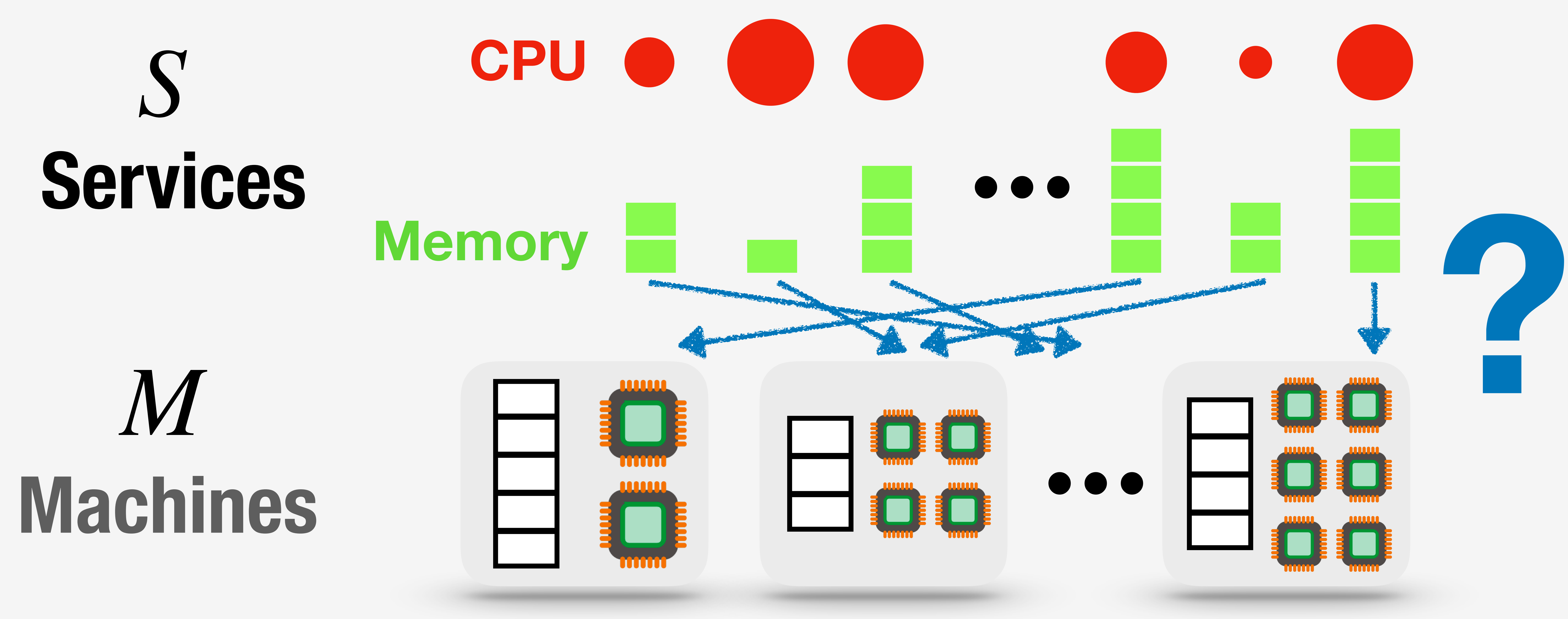


Photo from: https://www.reit.com/what-reit/reit-sectors/data-center-reits


S
Services
CPU
Memory
M
Machines
?

$y_m = 1$ if machine m is used

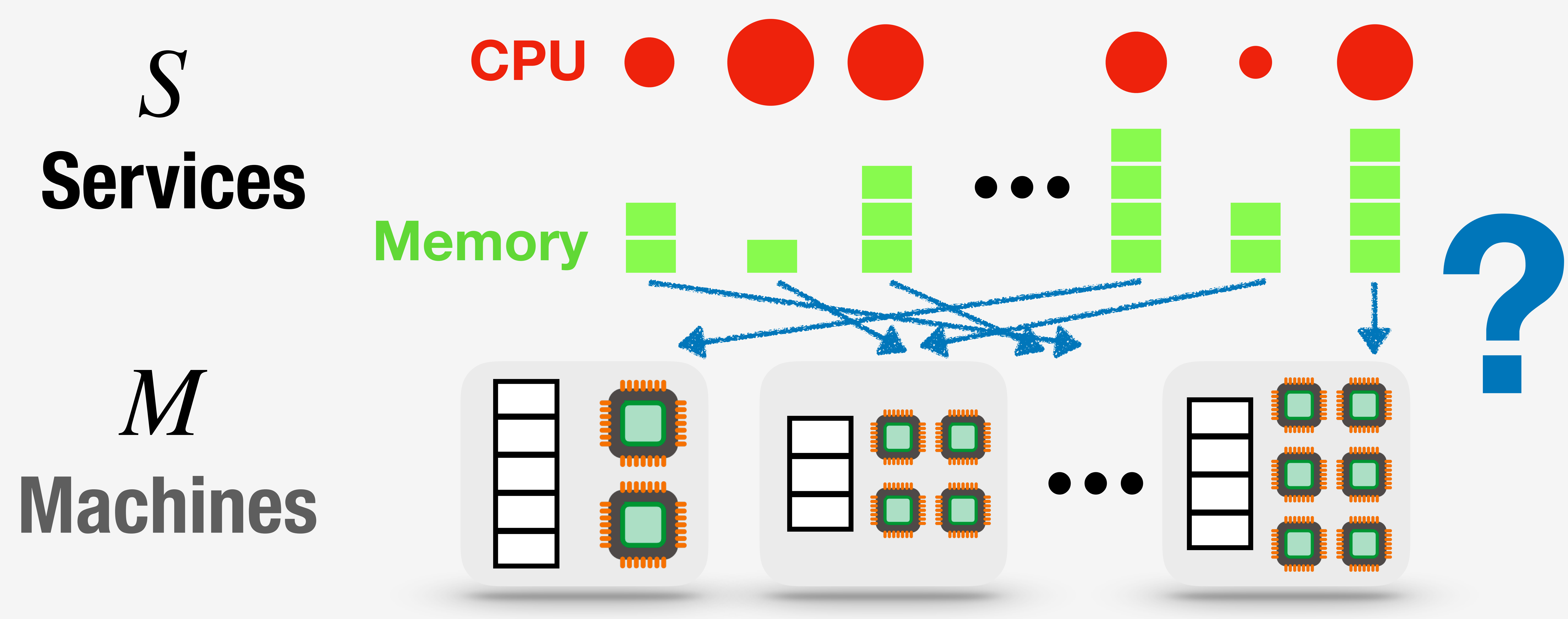
$x_{s,m} = 1$ if service s runs on m



$y_m = 1$ if machine m is used

$x_{s,m} = 1$ if service s runs on m

$x \in \{0,1\}^{S \times M}, y \in \{0,1\}^M$

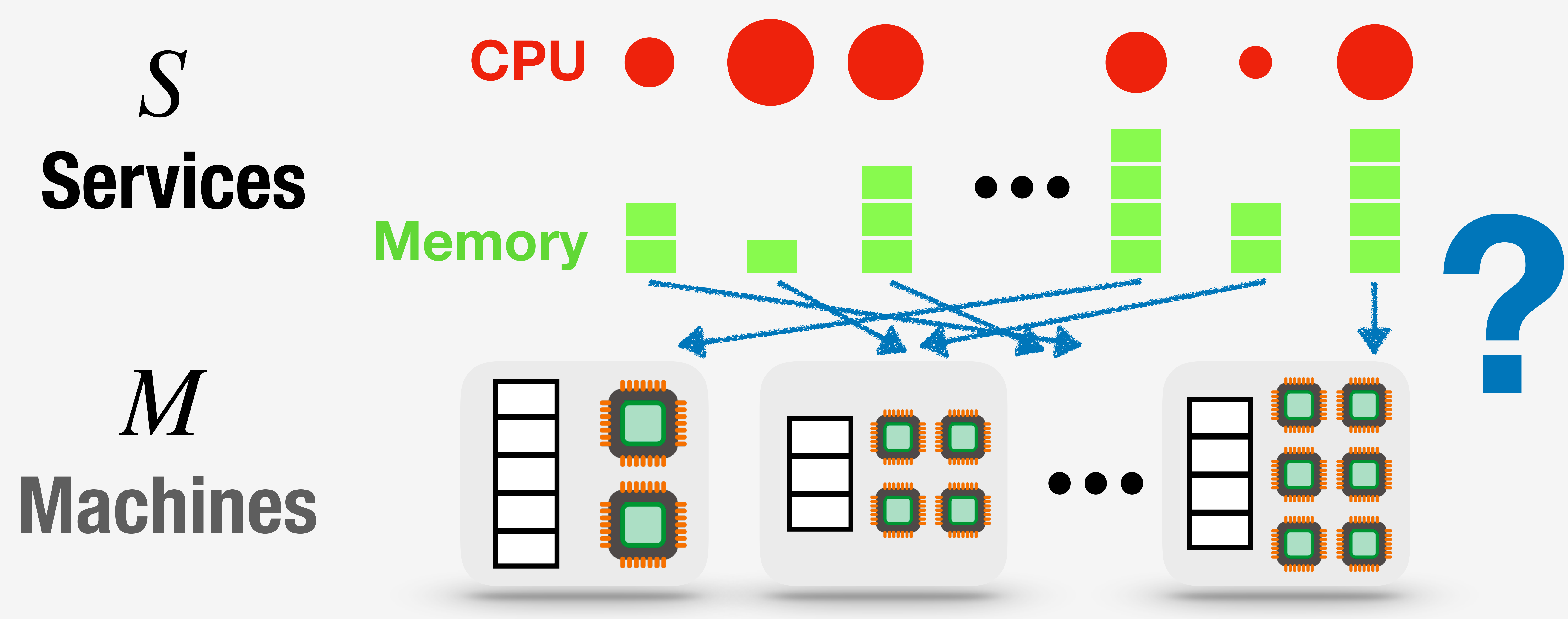


$y_m = 1$ if machine m is used

$x_{s,m} = 1$ if service s runs on m

$x \in \{0,1\}^{S \times M},\ y \in \{0,1\}^M$

minimize $\sum_{m=1}^{M} y_m$



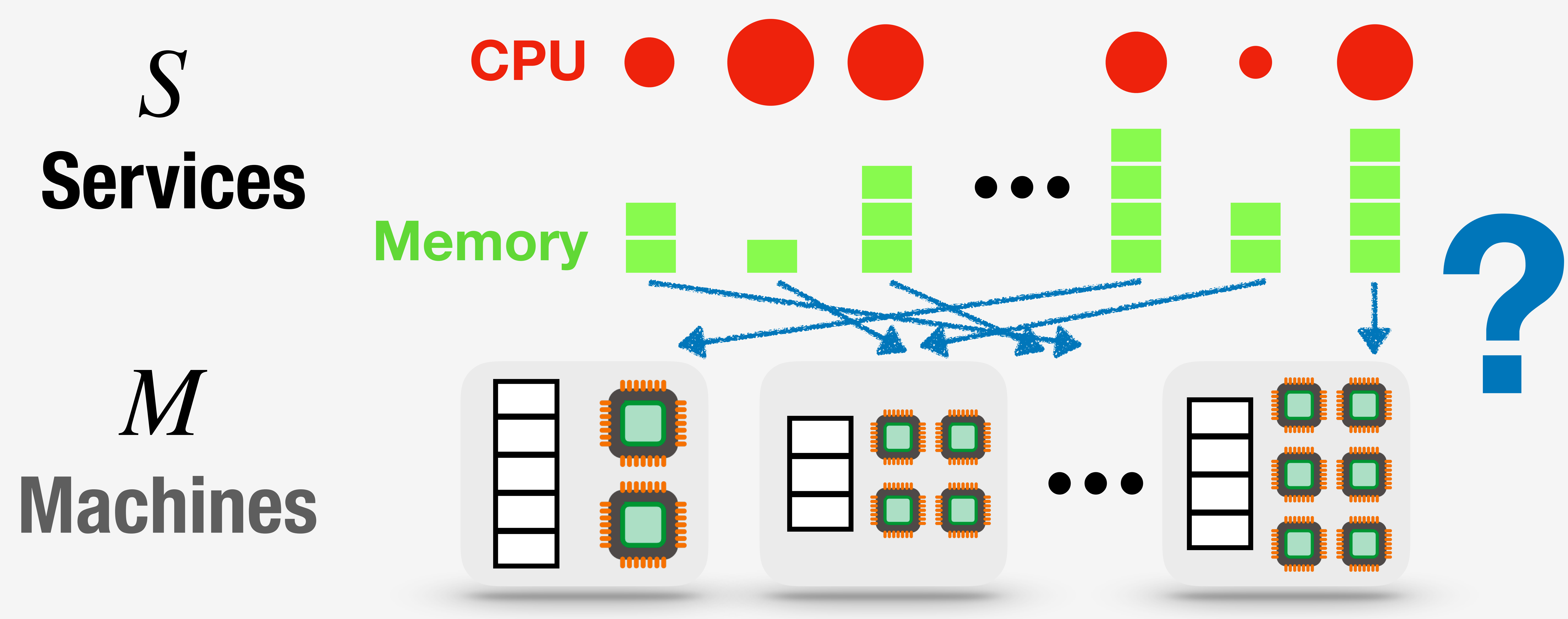
$y_m = 1$ if machine m is used

$x_{s,m} = 1$ if service s runs on m

$x \in \{0,1\}^{S \times M}$, $y \in \{0,1\}^M$

minimize $\sum_{m=1}^{M} y_m$

Constraints:

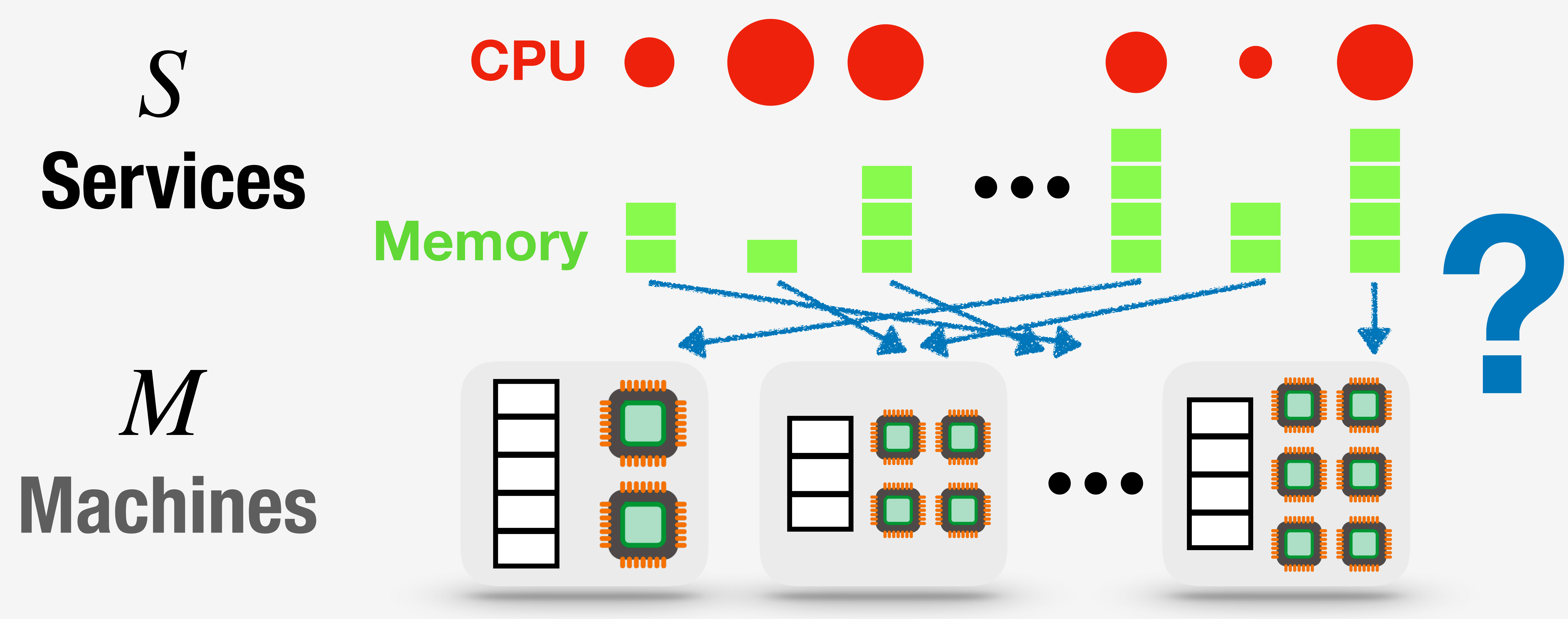


$y_m = 1$ if machine m is used

$x_{s,m} = 1$ if service s runs on m

$x \in \{0,1\}^{S \times M}, y \in \{0,1\}^M$

minimize $\sum_{m=1}^{M} y_m$



Constraints:

Each service on one machine only

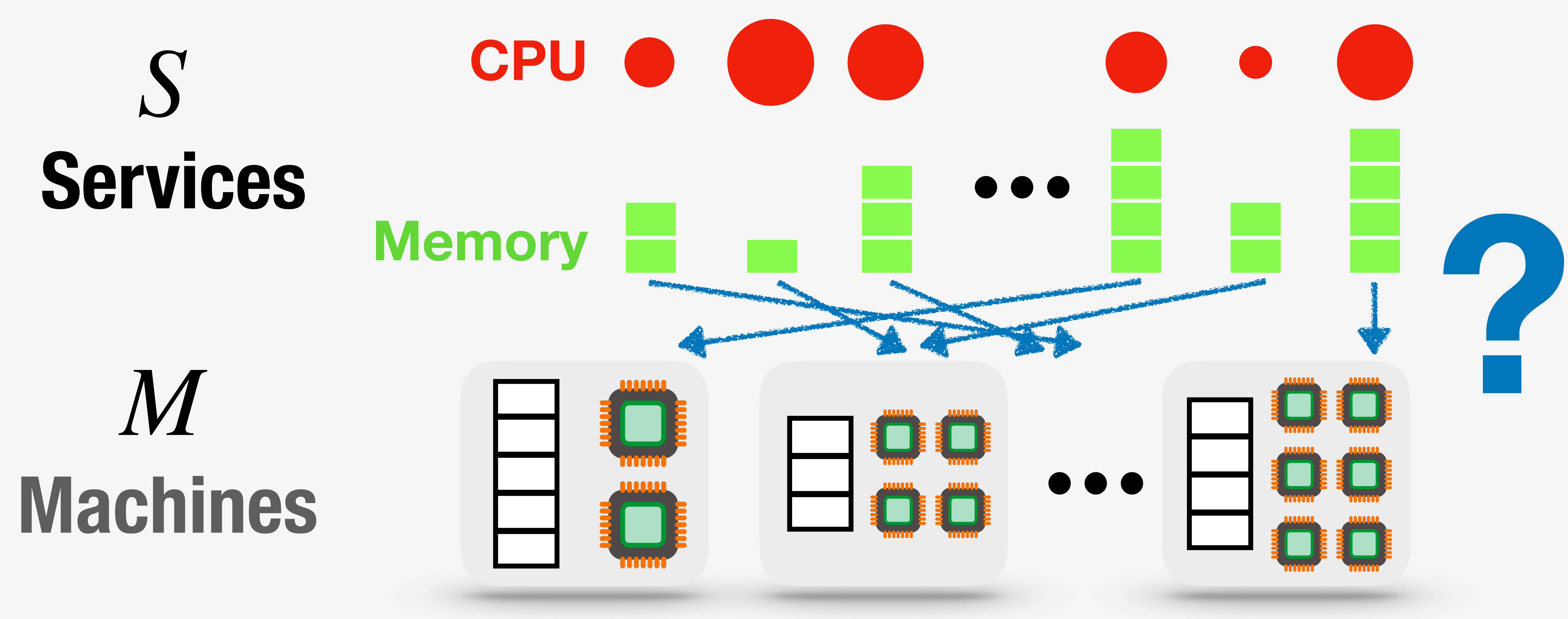
$$\sum_{m=1}^{M} x_{s,m} = 1 \quad \forall s$$

$y_m = 1$ if machine m is used

$x_{s,m} = 1$ if service s runs on m

$$x \in \{0,1\}^{S \times M},\ y \in \{0,1\}^M$$

$$\textbf{minimize} \sum_{m=1}^{M} y_m$$



Constraints:

Each service on one machine only

$$\sum_{m=1}^{M} x_{s,m} = 1 \quad \forall s$$

$$y_m \geq x_{s,m} \quad \forall s, m$$

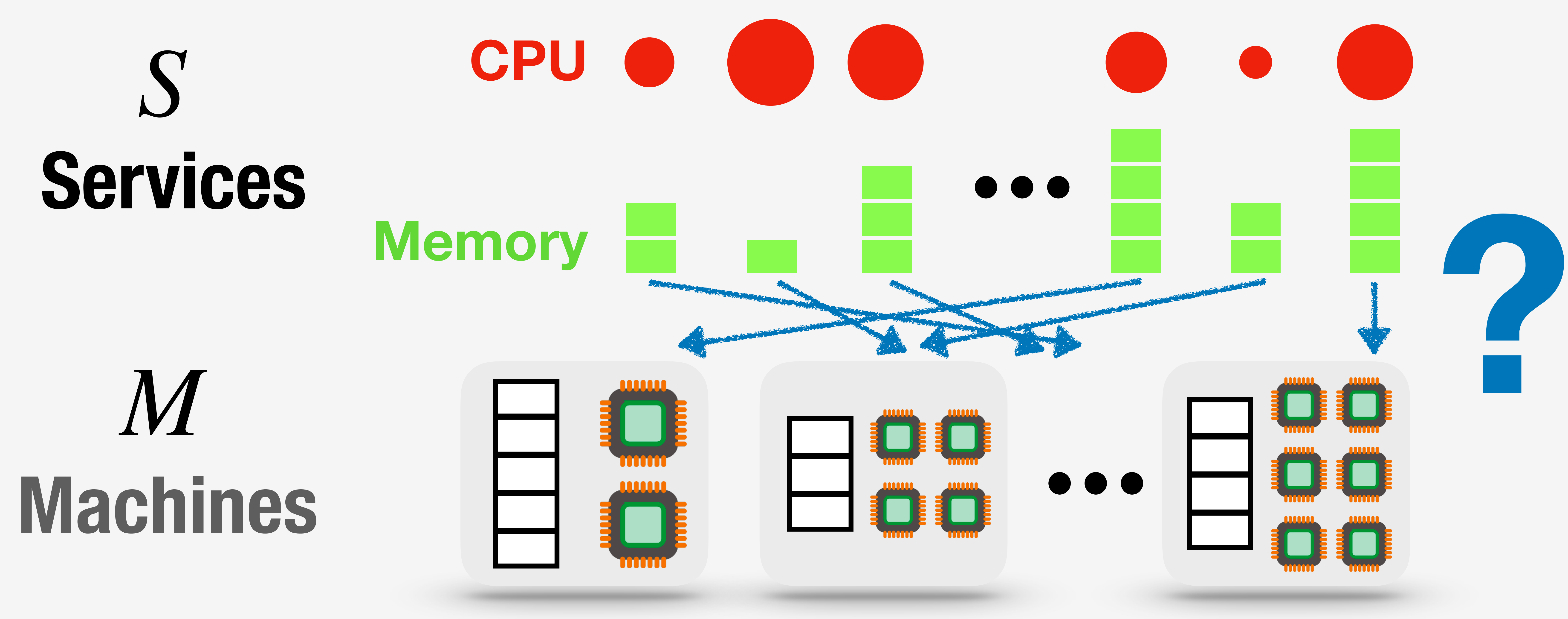
Machine is "ON" if a job is assigned to it

$y_m = 1$ if machine m is used

$x_{s,m} = 1$ if service s runs on m

$x \in \{0,1\}^{S \times M}$, $y \in \{0,1\}^M$

minimize $\sum_{m=1}^{M} y_m$



Constraints:

Each service on one machine only

$$\sum_{m=1}^{M} x_{s,m} = 1 \quad \forall s$$

Memory capacity

$$\sum_{s=1}^{S} \textbf{mem}(s) \cdot x_{s,m} \leq \textbf{cap-mem}(m) \quad \forall m$$

$$y_m \geq x_{s,m} \quad \forall s, m$$

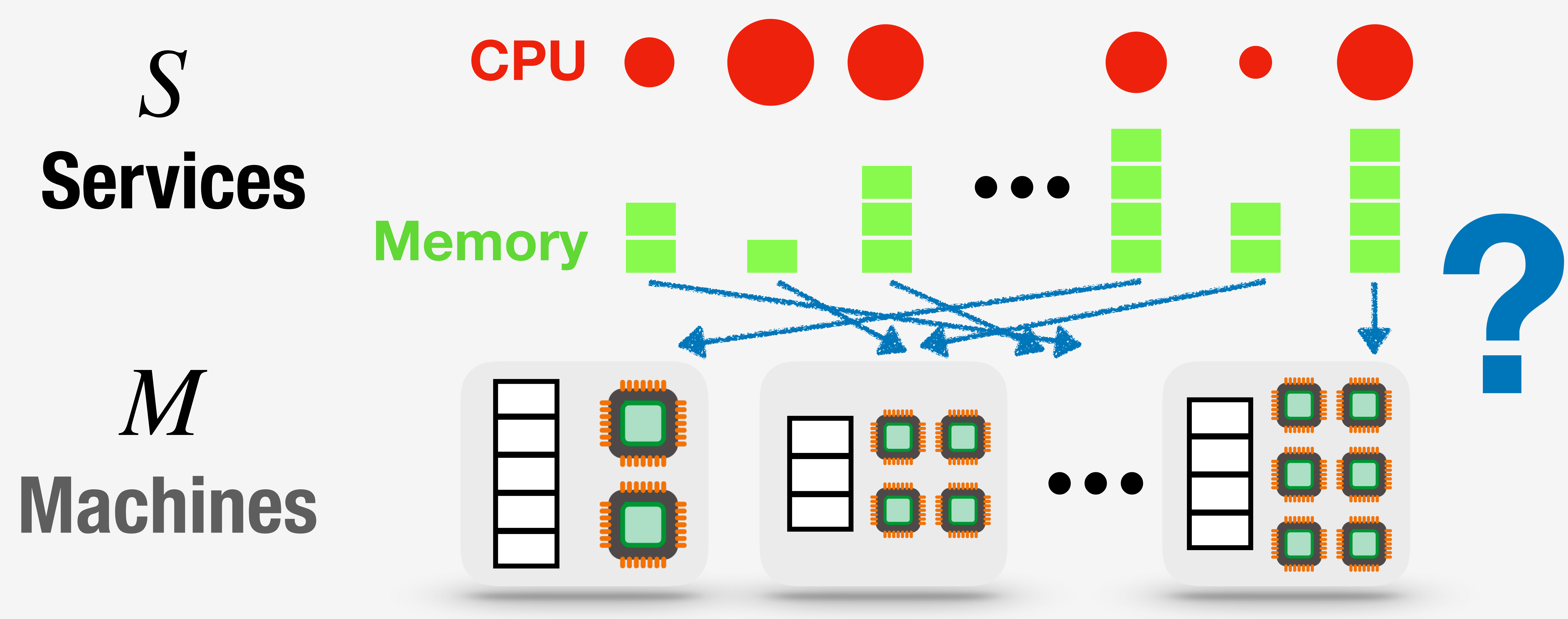
Machine is "ON" if a job is assigned to it

$y_m = 1$ if machine m is used

$x_{s,m} = 1$ if service s runs on m

$$x \in \{0,1\}^{S \times M},\ y \in \{0,1\}^M$$

minimize $\sum_{m=1}^{M} y_m$



Constraints:

Each service on one machine only

$$\sum_{m=1}^{M} x_{s,m} = 1 \quad \forall s$$

$$y_m \geq x_{s,m} \quad \forall s, m$$

Machine is "ON" if a job is assigned to it

Memory capacity

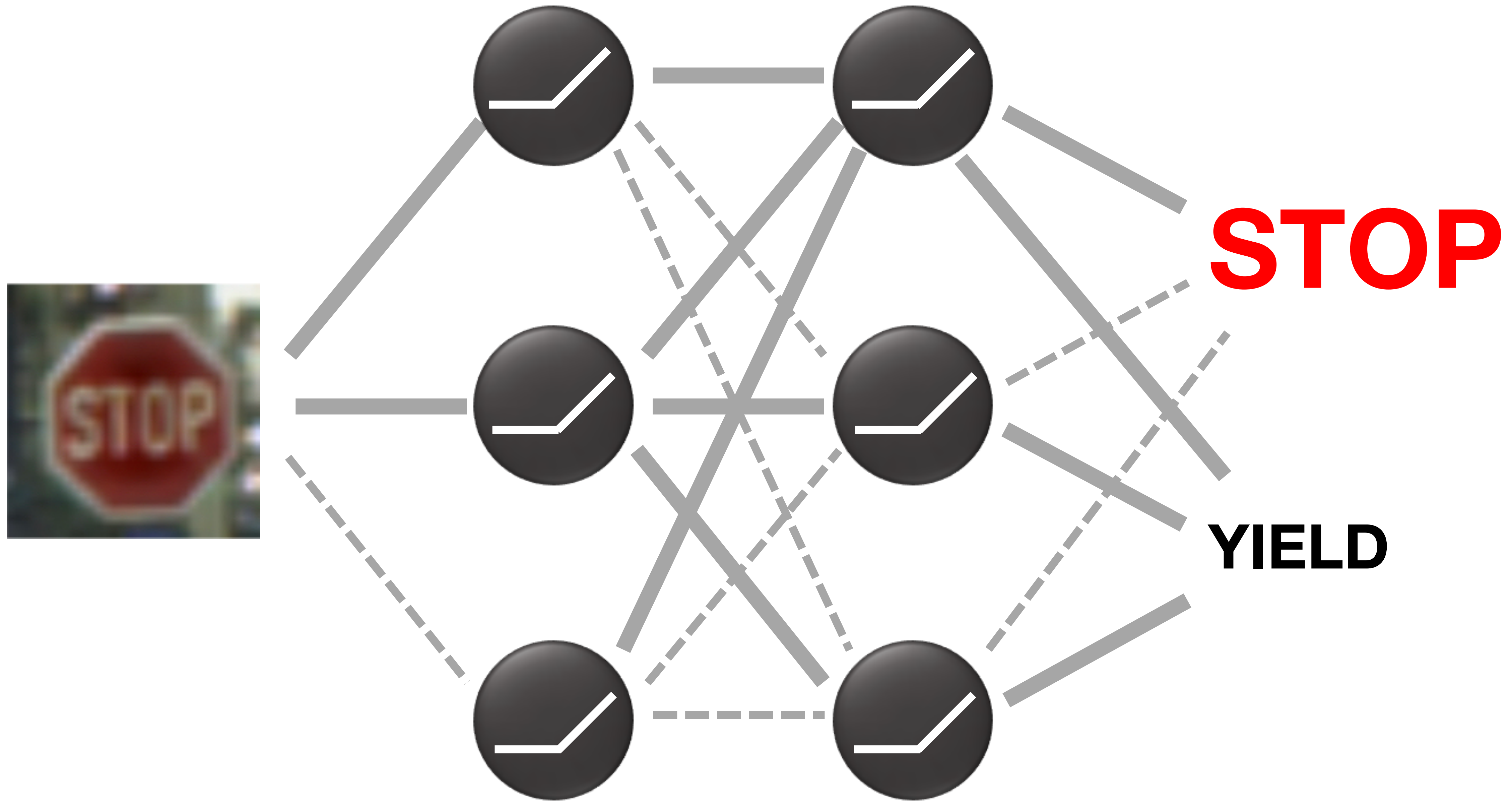
$$\sum_{s=1}^{S} \textbf{mem}(s) \cdot x_{s,m} \leq \textbf{cap-mem}(m) \quad \forall m$$

$$\sum_{s=1}^{S} \textbf{cpu}(s) \cdot x_{s,m} \leq \textbf{cap-cpu}(m) \quad \forall m$$

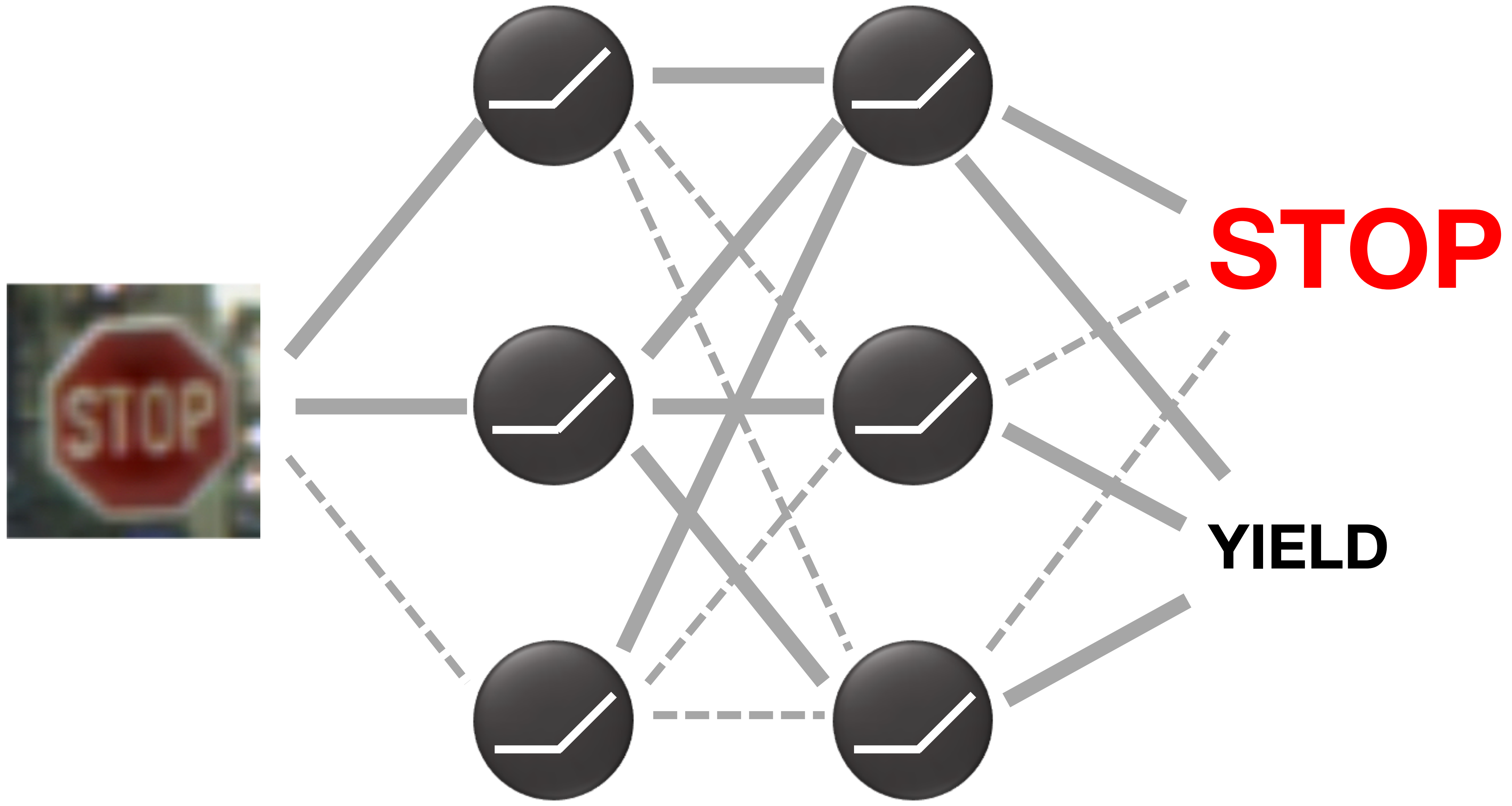
Processor capacity

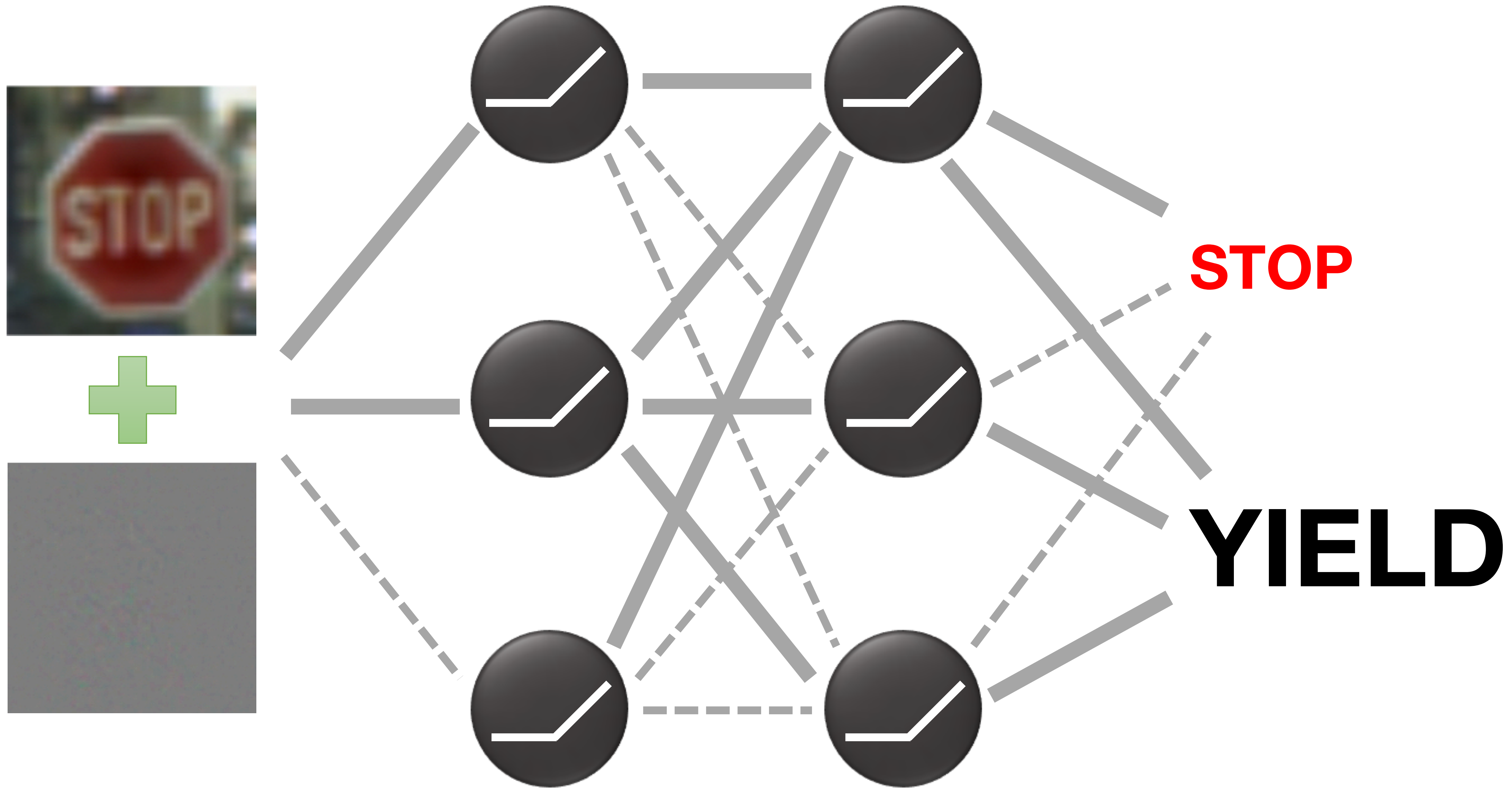
Safety-Critical Machine Learning

Safety-Critical Machine Learning



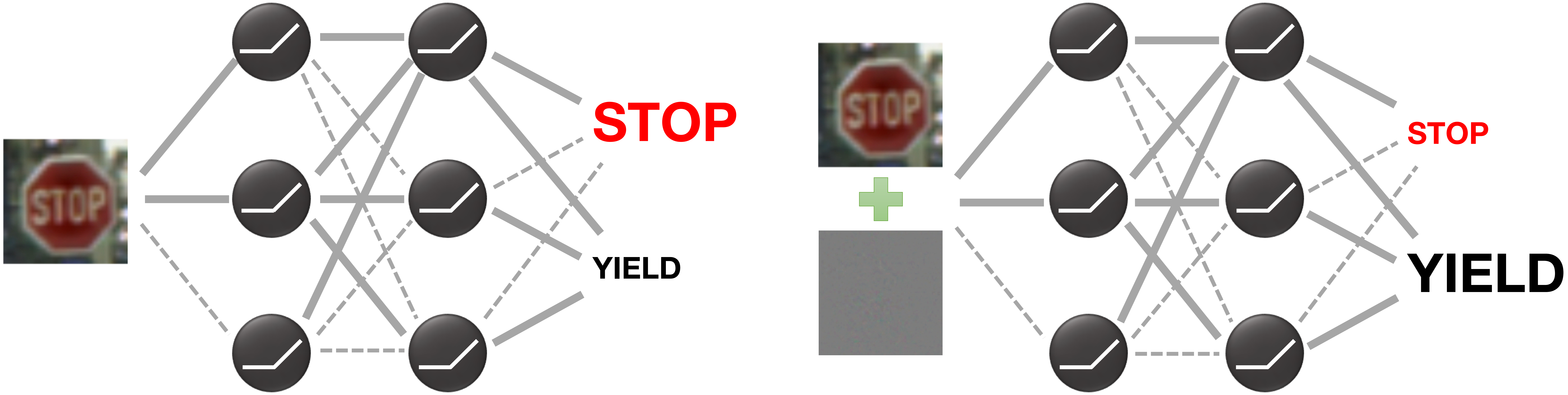
Safety-Critical Machine Learning





Safety-Critical Machine Learning

Goal: **Guarantee** that trained model has **desirable behavior**

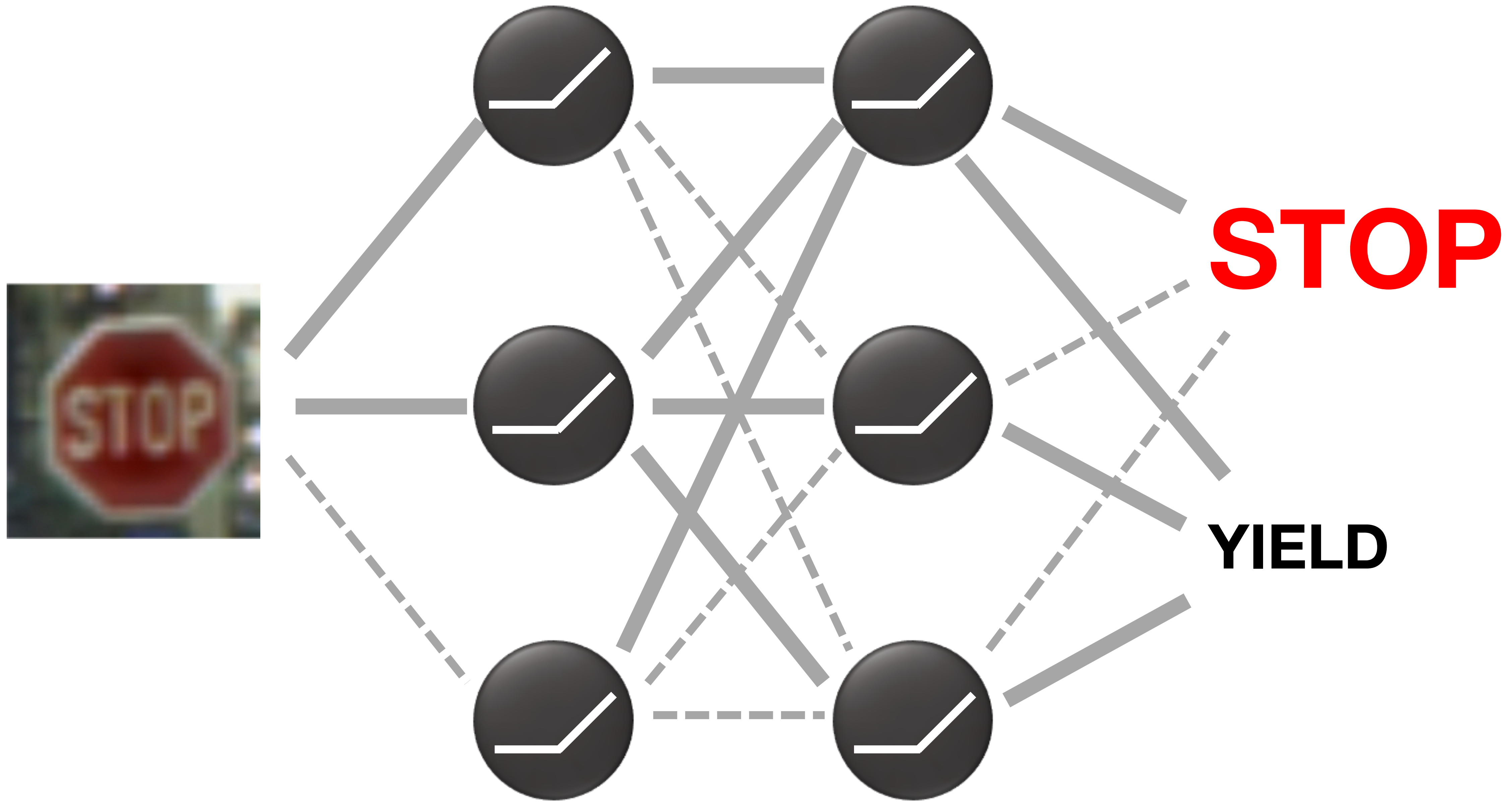


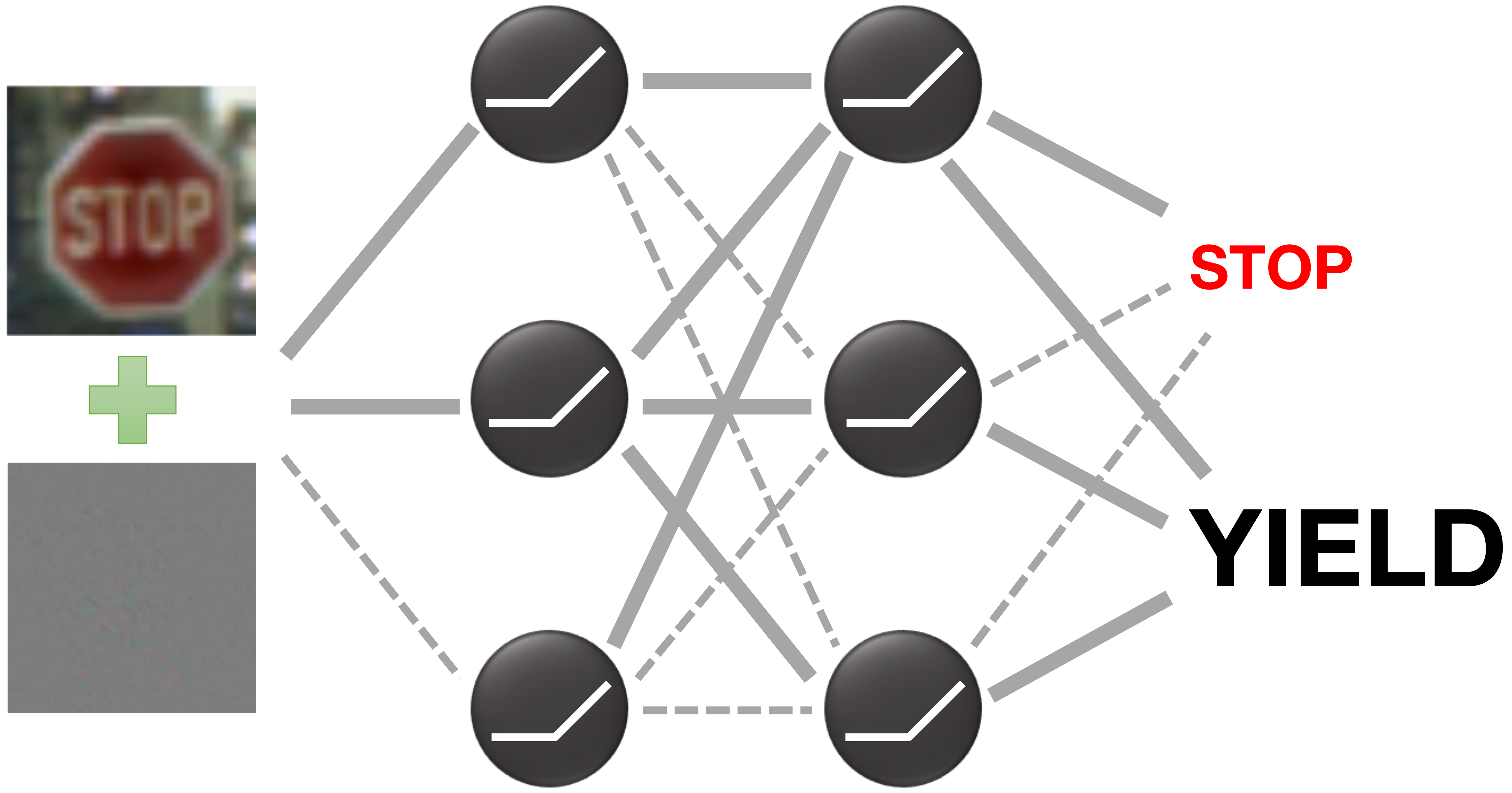
Safety-Critical Machine Learning

Goal: **Guarantee** that trained model has **desirable behavior**

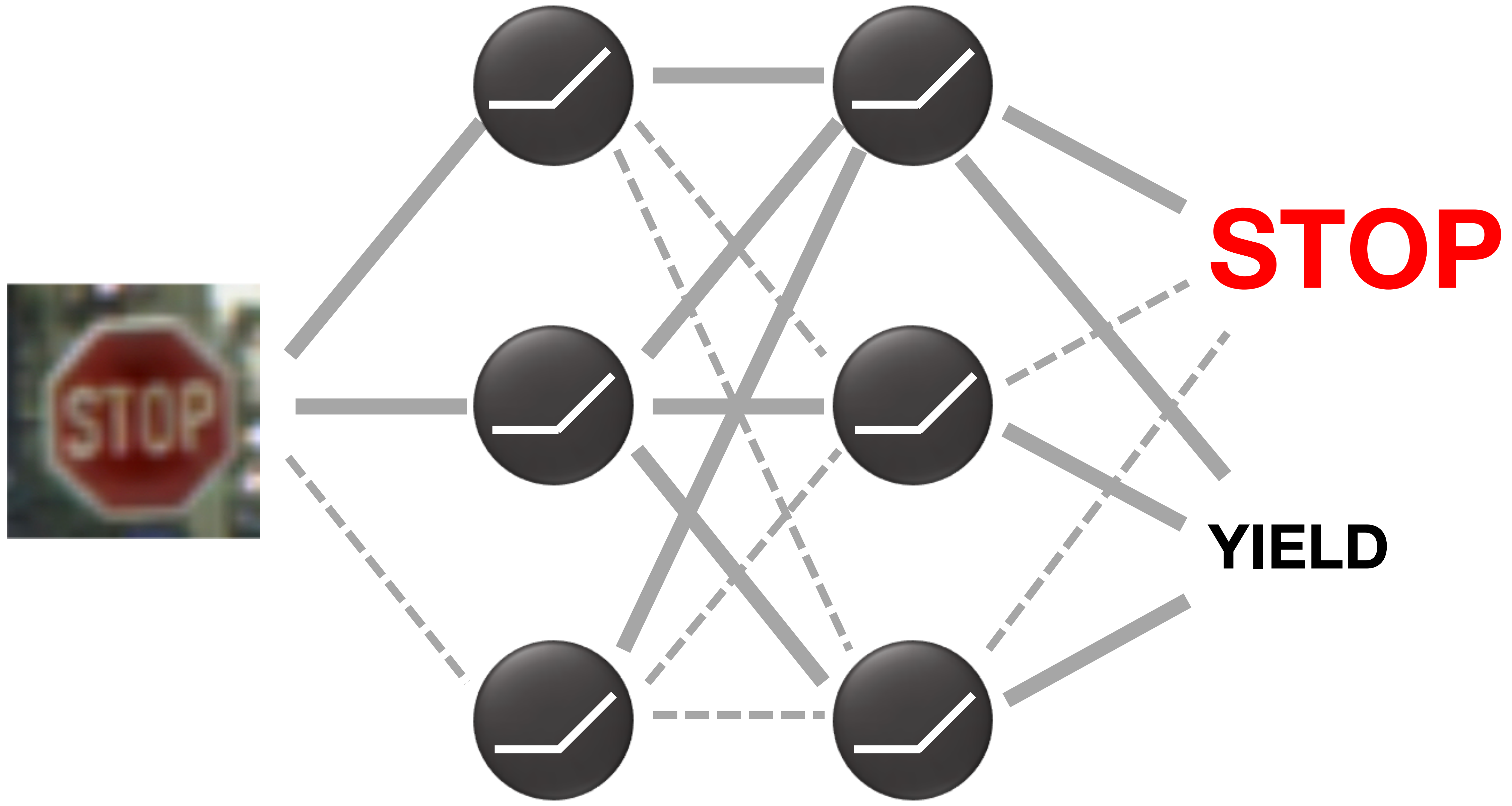
→

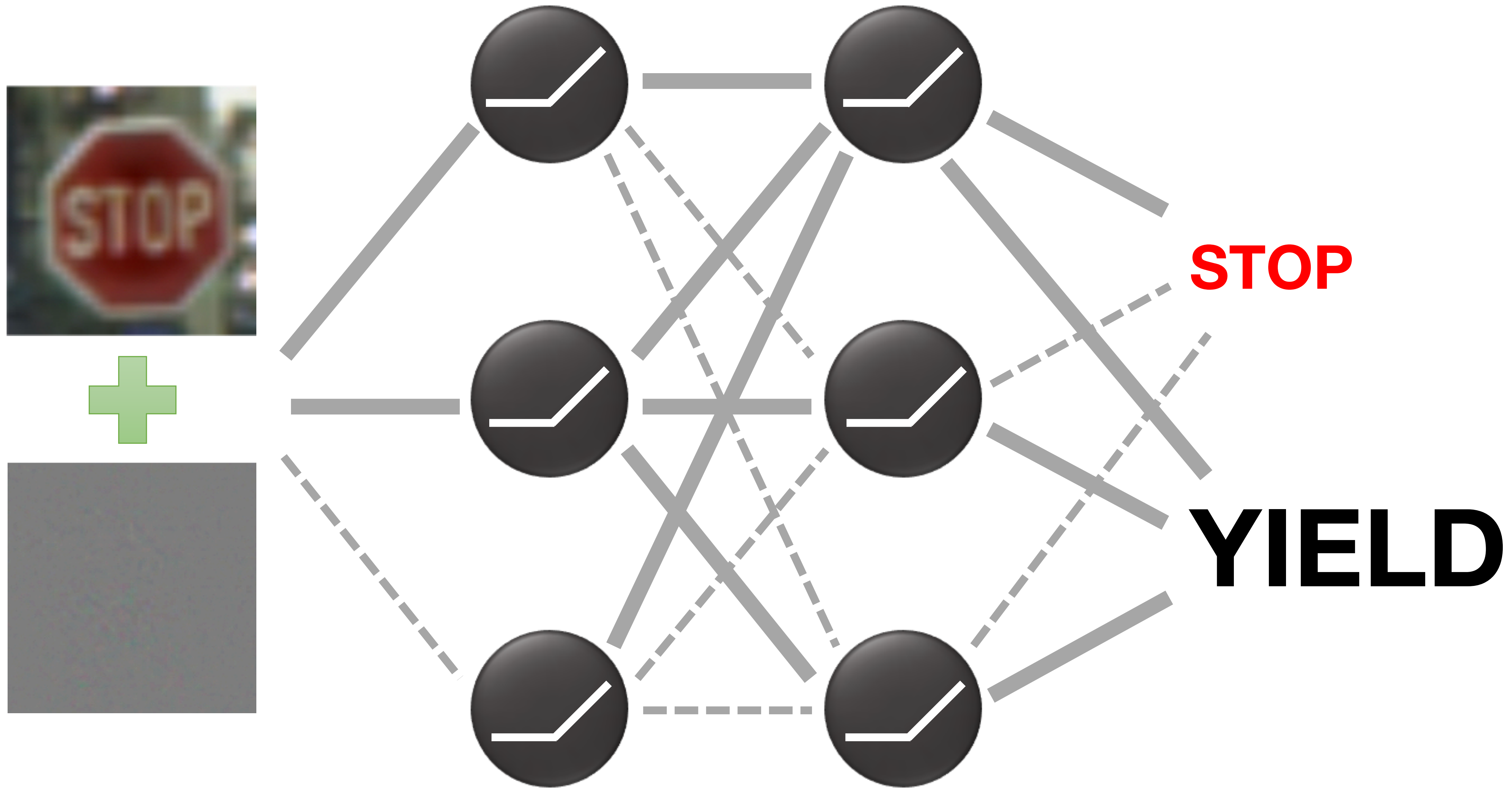
Model ReLU with **Binary** variables + Linear **Inequalities**





Safety-Critical Machine Learning



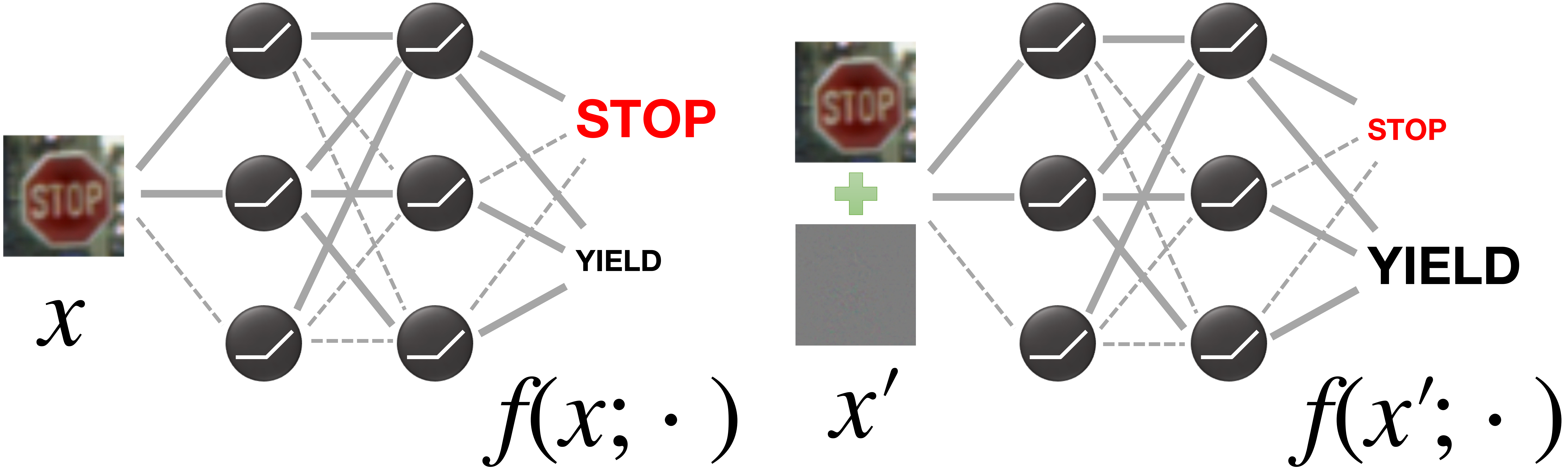


Safety-Critical Machine Learning

Verification Problem

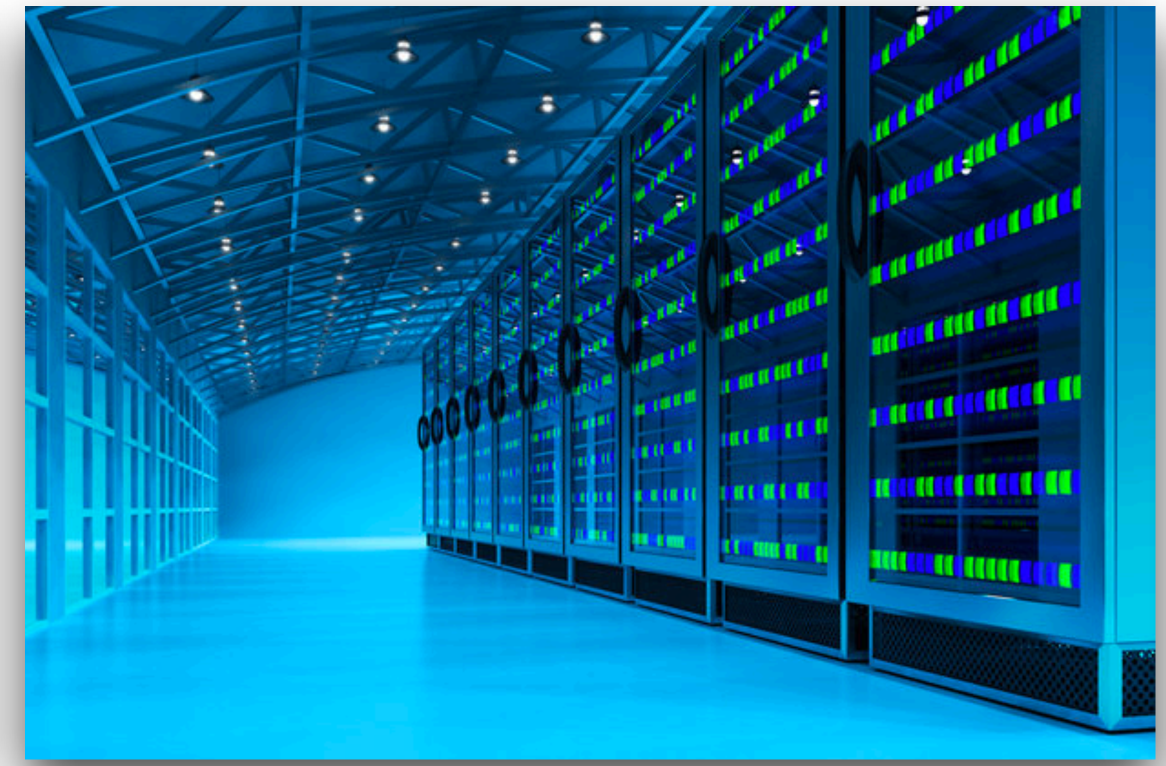
prove $\nexists x'$ **close to** x

such that $f(x';\ $**STOP**$) < f(x';\ $**YIELD**$)$

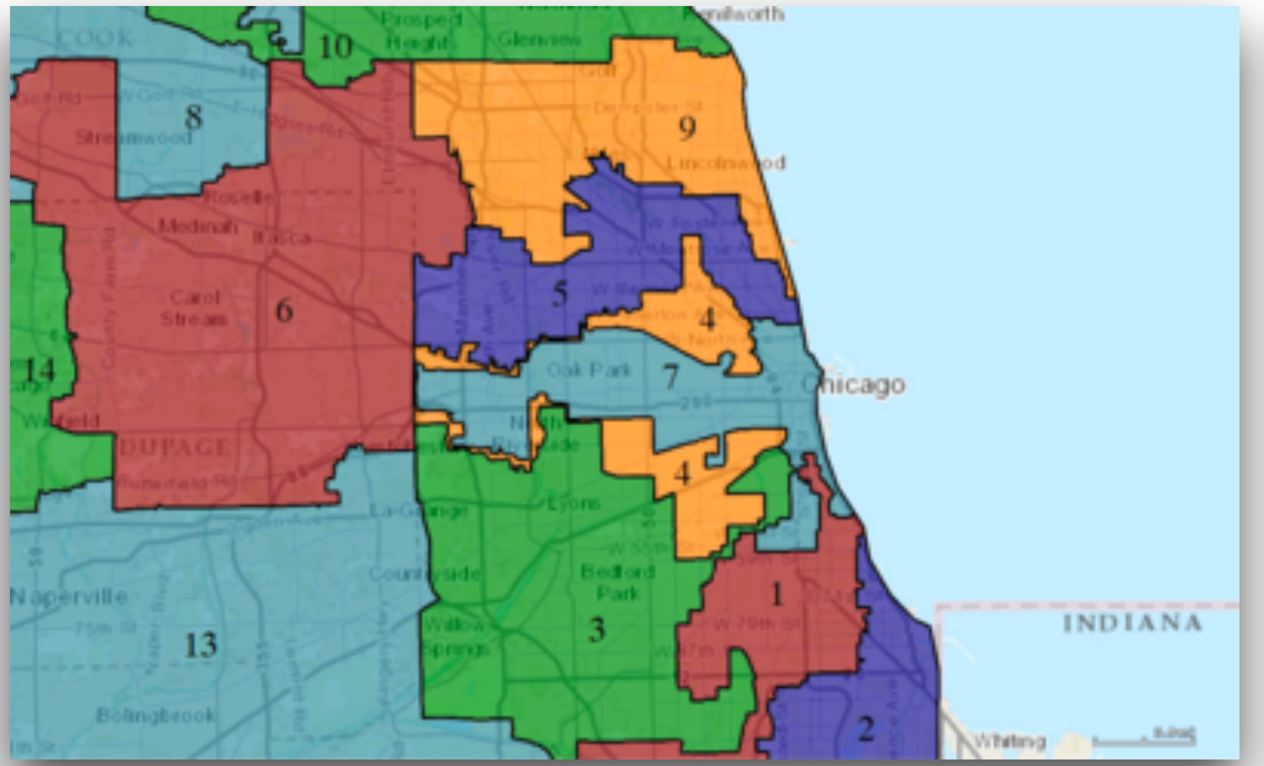


Auction Design

Data Center Management



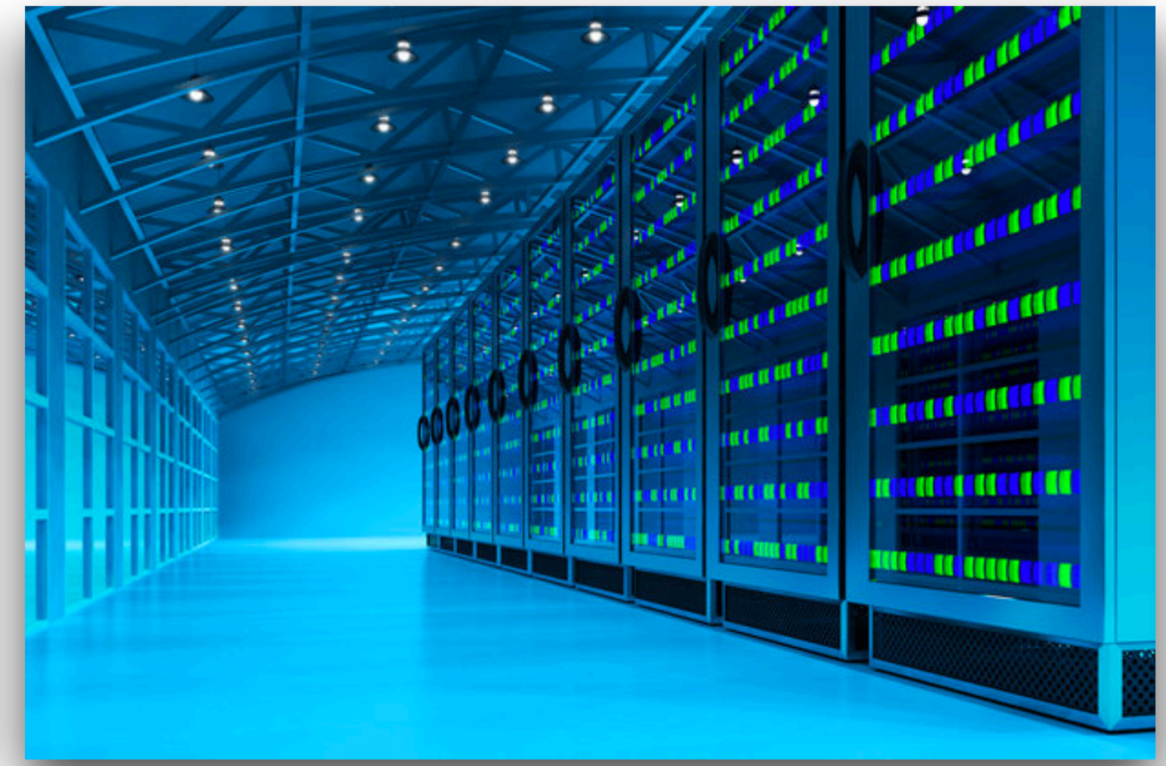
Political Districting



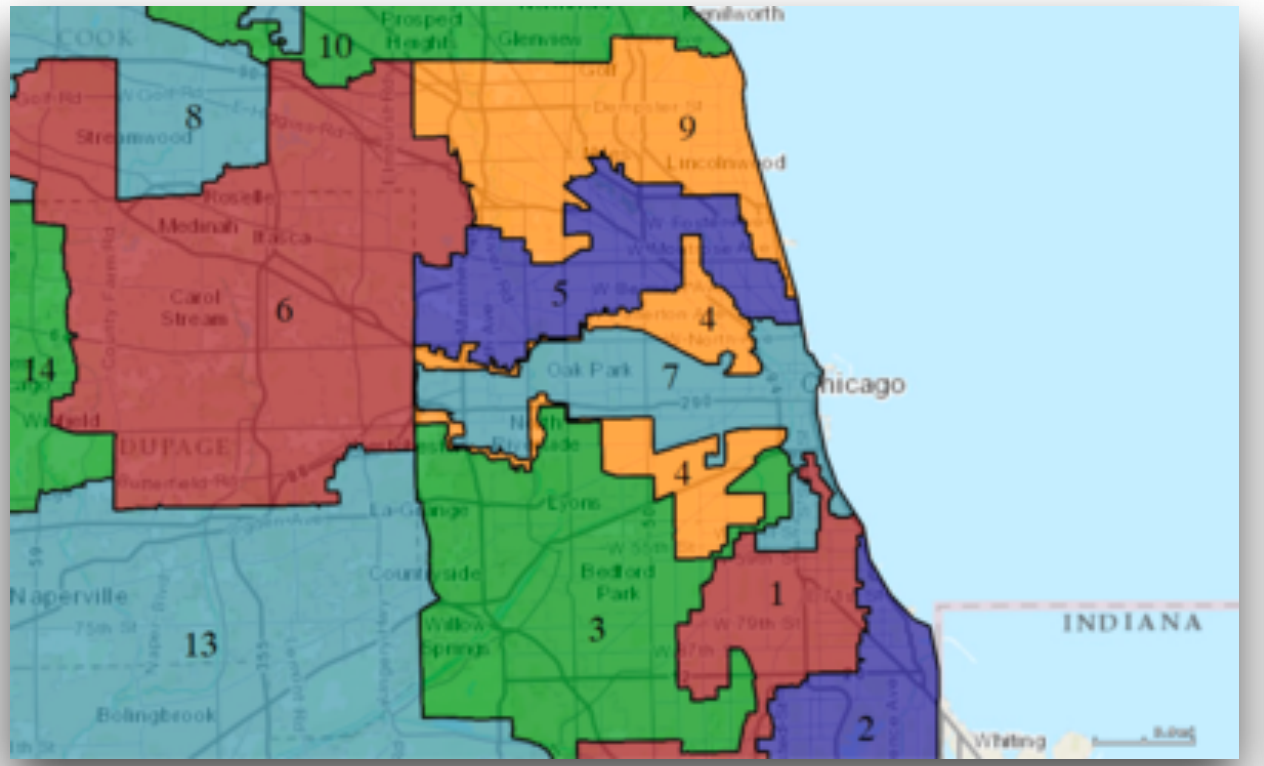
Kidney Exchange

Auction Design

Data Center Management

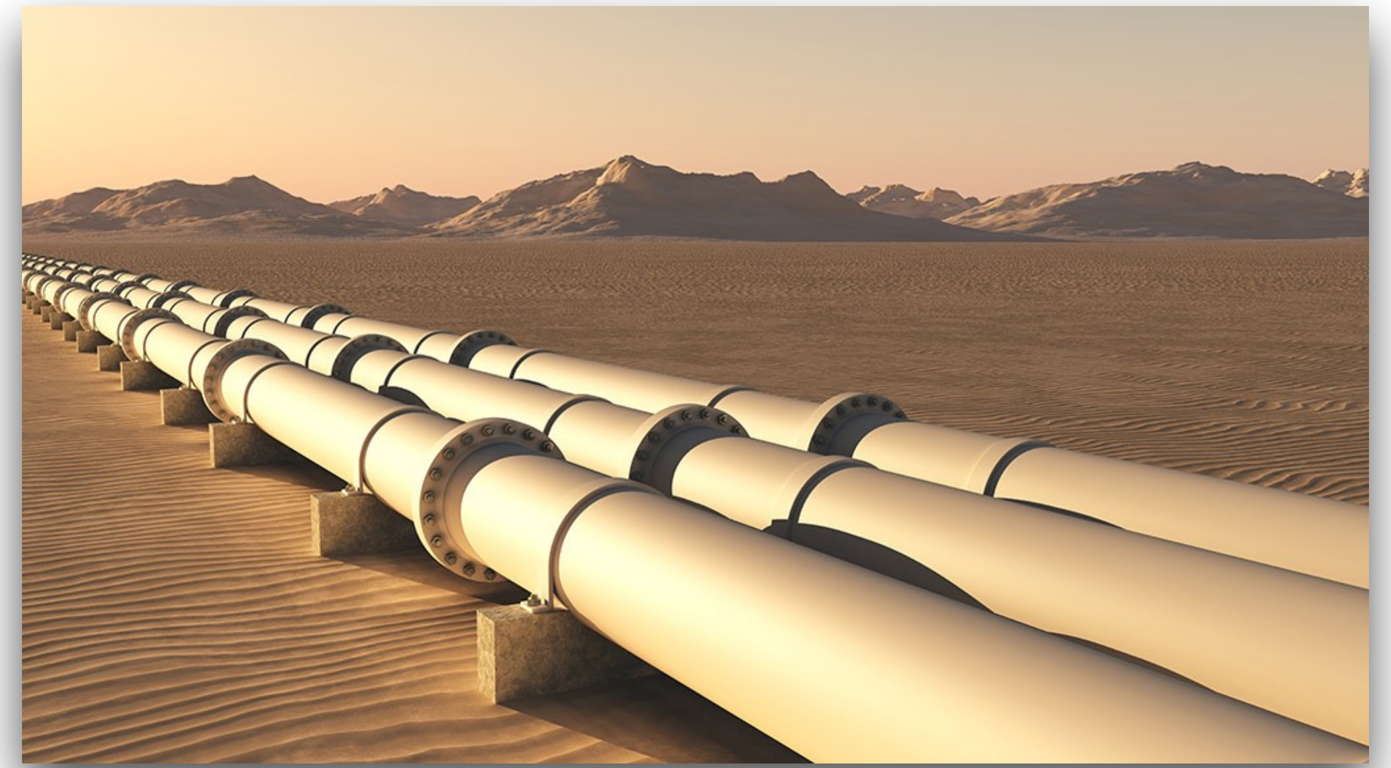


Political Districting

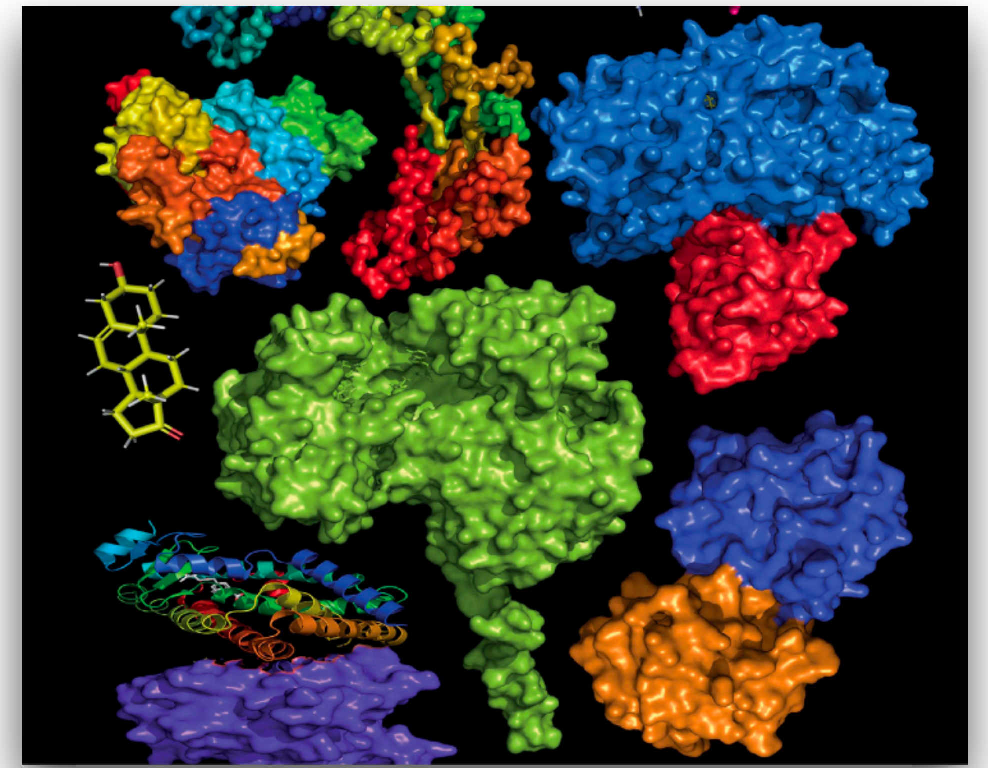


Kidney Exchange

Energy Systems



Scientific Discovery

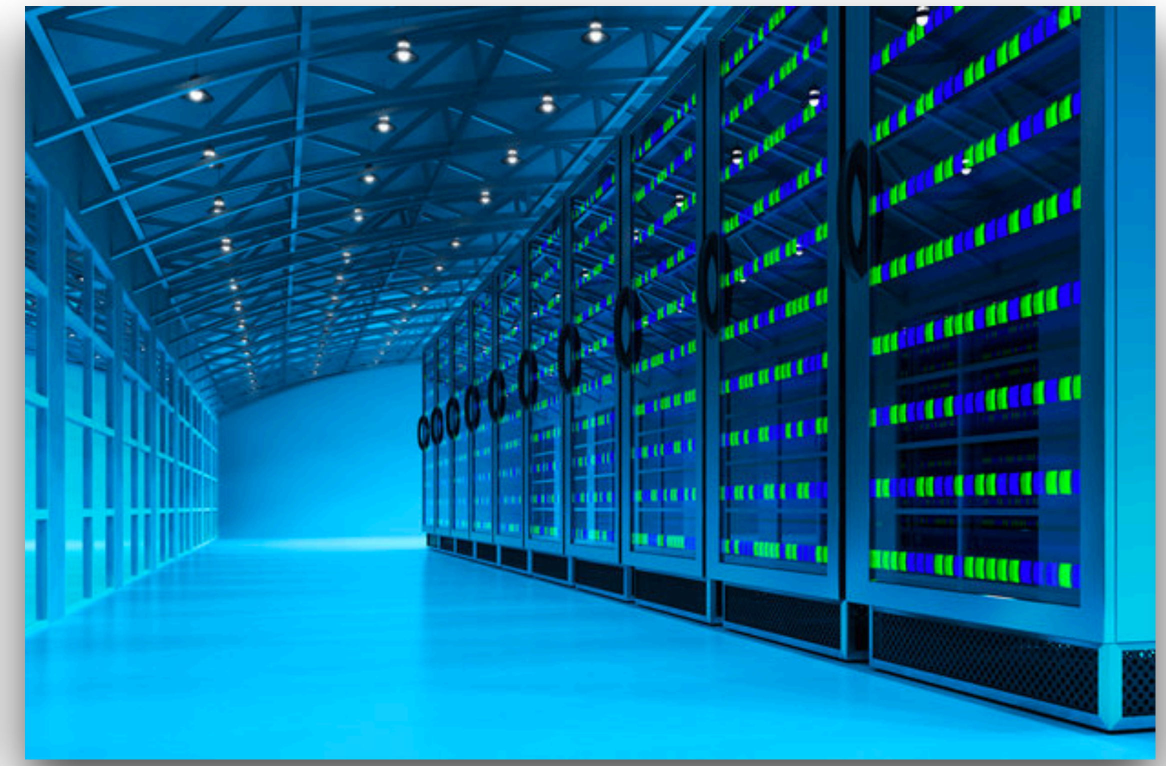


Ridesharing

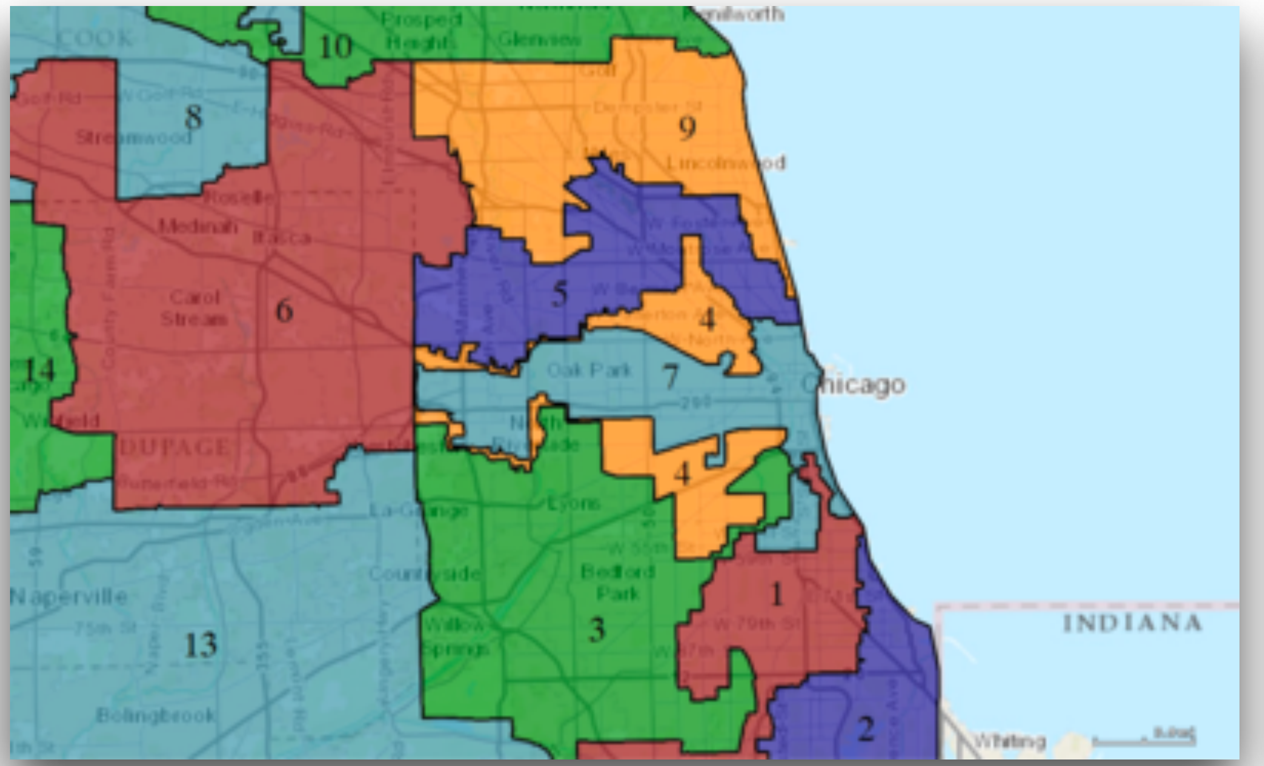
Cancer Therapeutics

Auction Design

Data Center Management

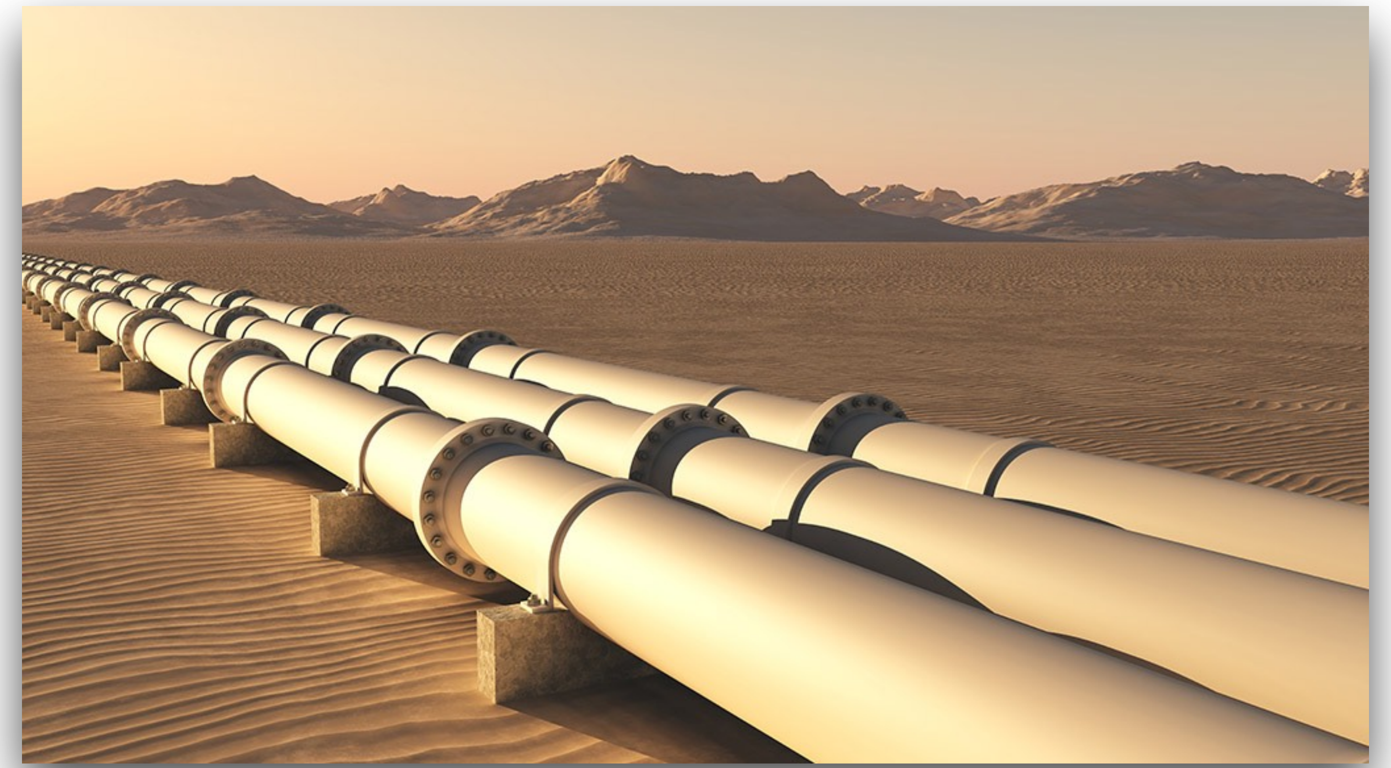


Political Districting

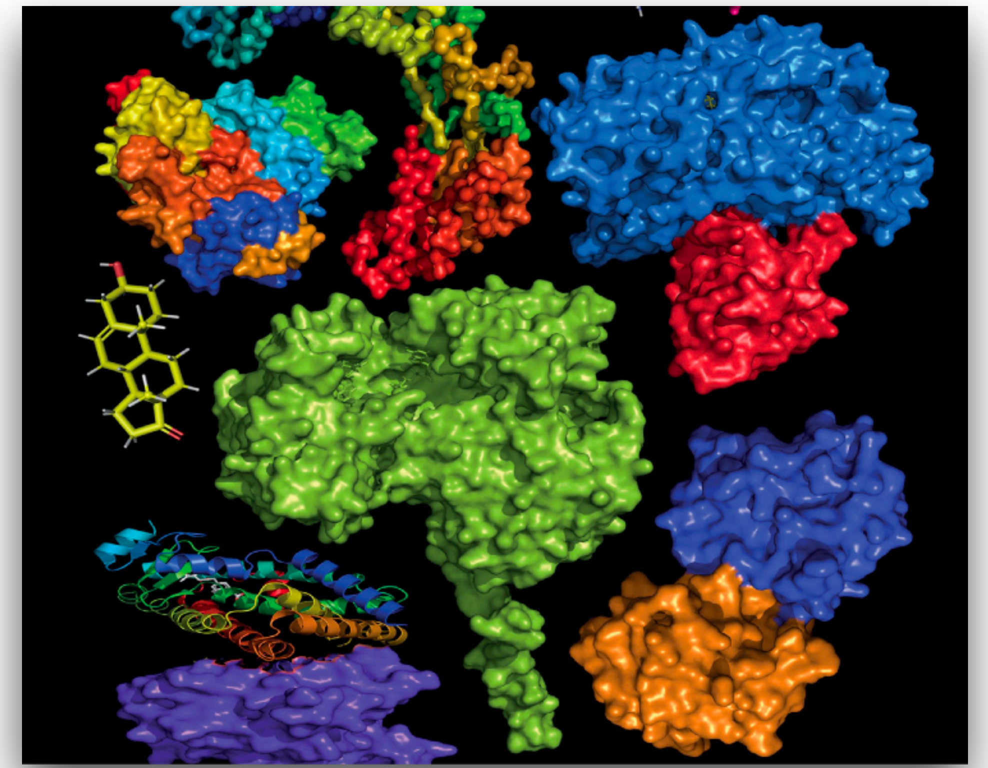


Kidney Exchange

Energy Systems



Scientific Discovery



Ridesharing

Cancer Therapeutics

Airline Scheduling

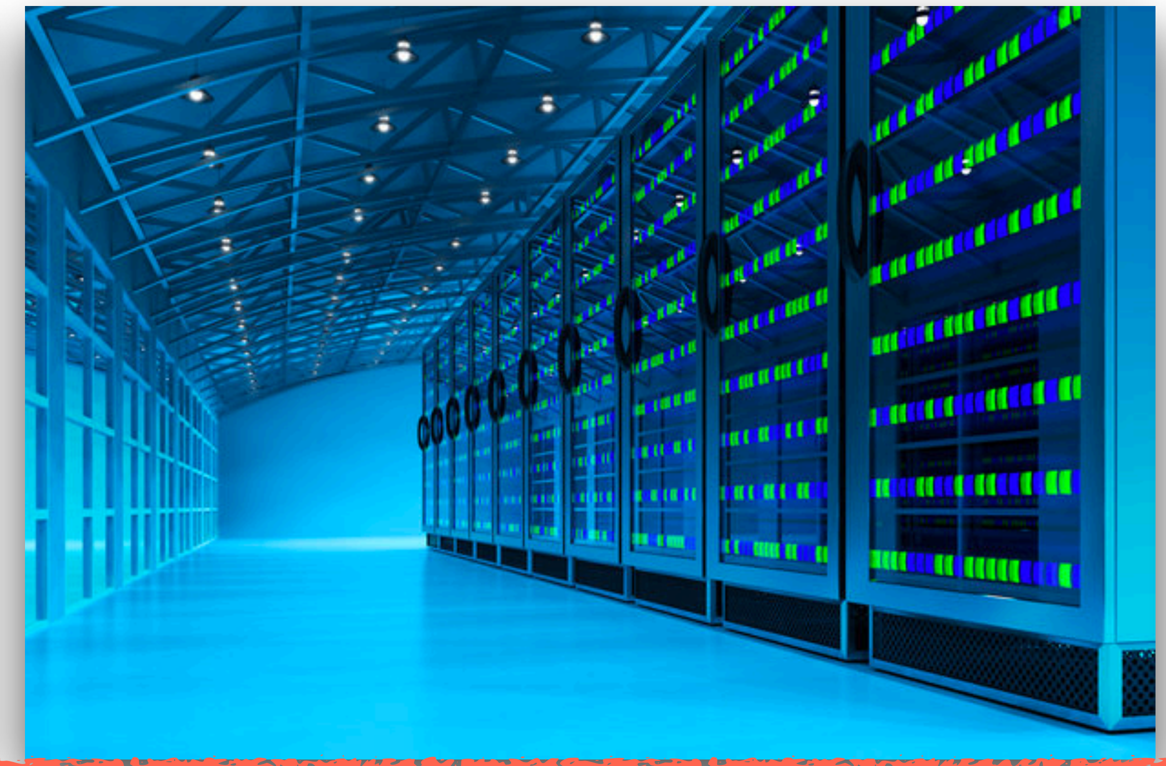
Conservation Planning

Disaster Response

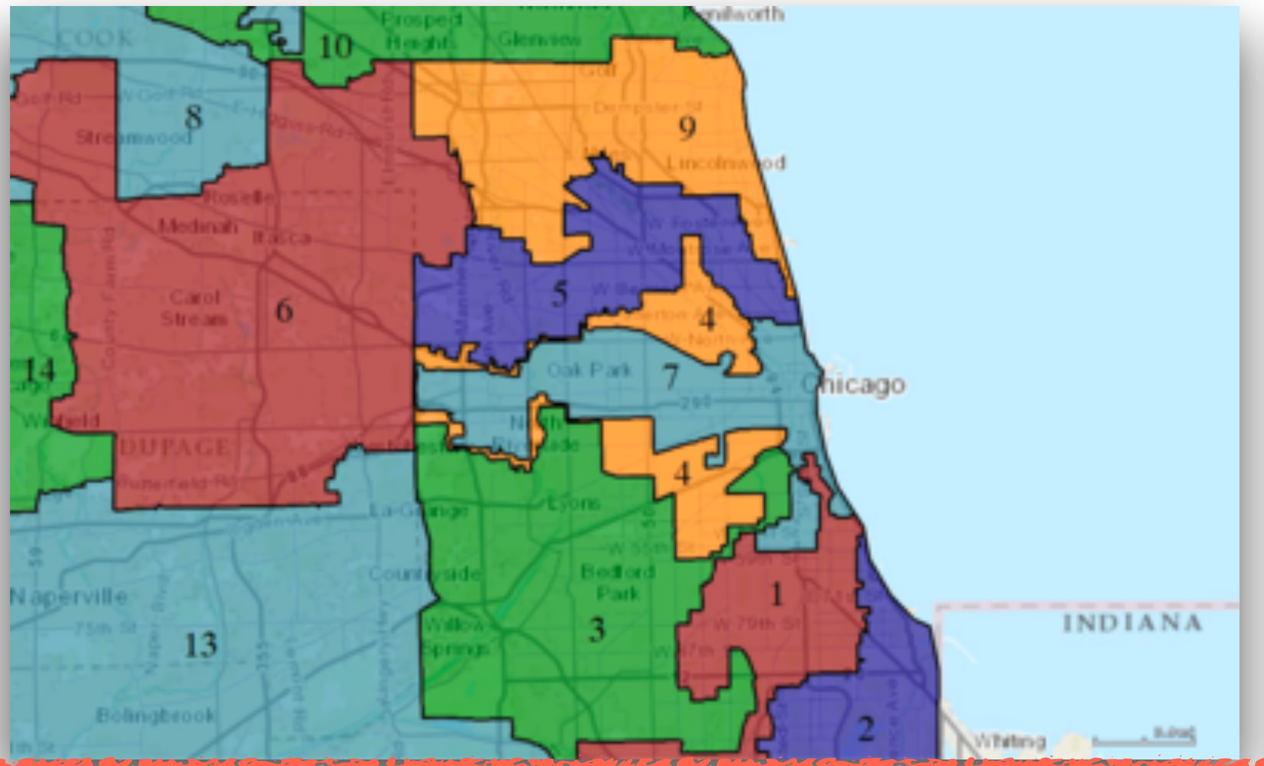
College Admissions

Auction Design

Data Center Management



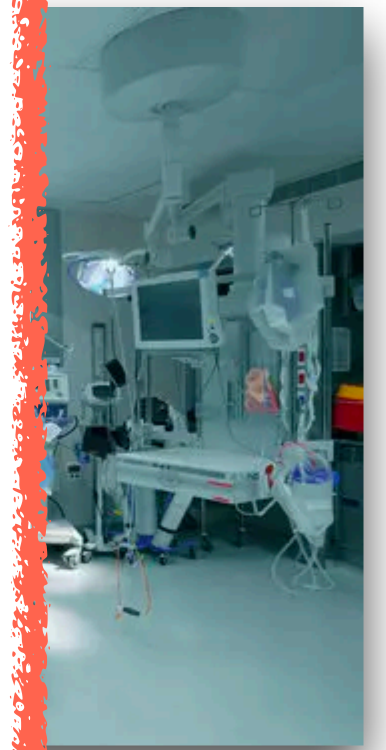
Political Districting



Kidney Exchange

E

utics



> 50% of INFORMS Edelman Award winners use Discrete Optimization

→ **Billions** ($) in savings/profit

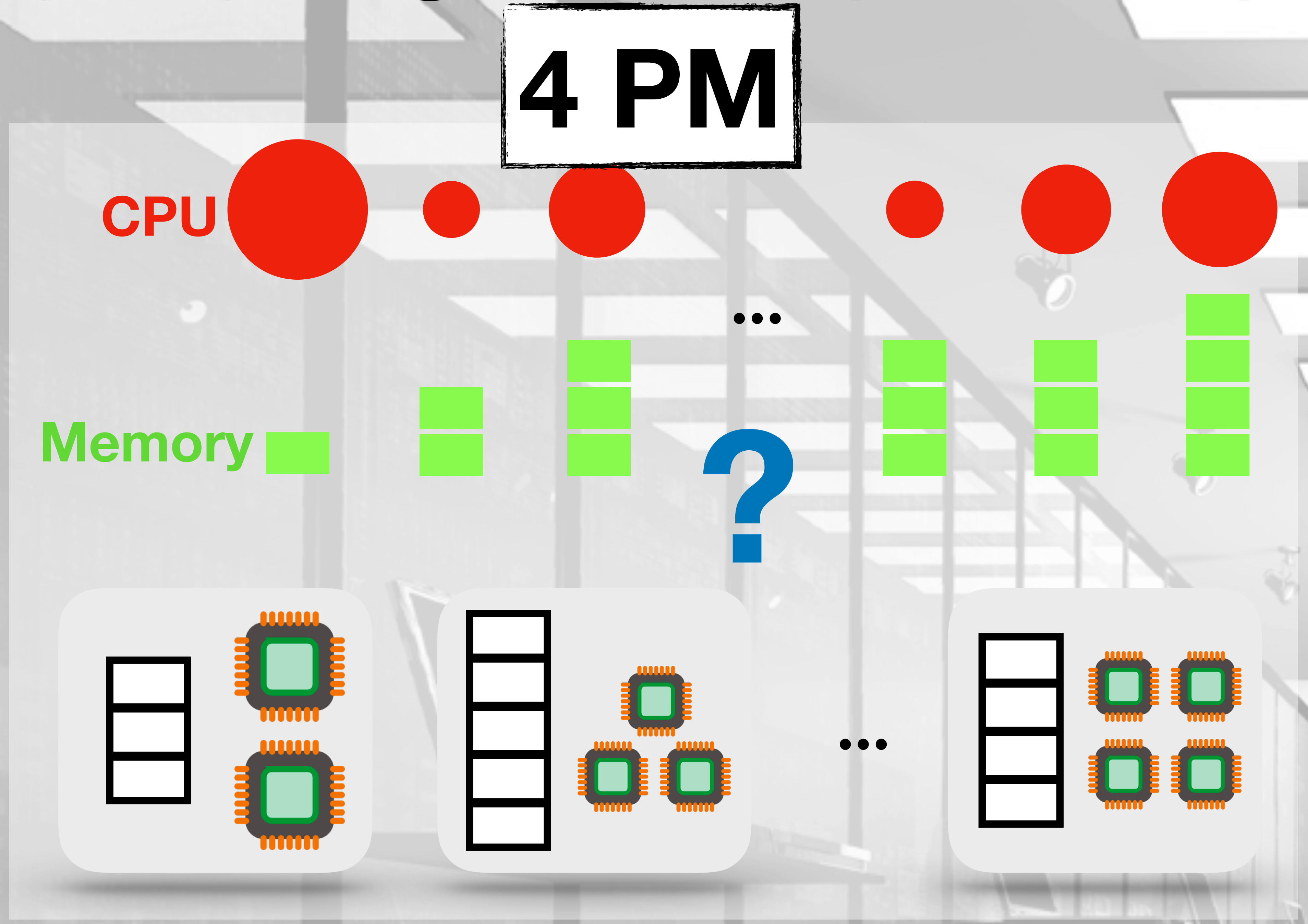
George Nemhauser, Plenary at EURO INFORMS, 2013

Airlin

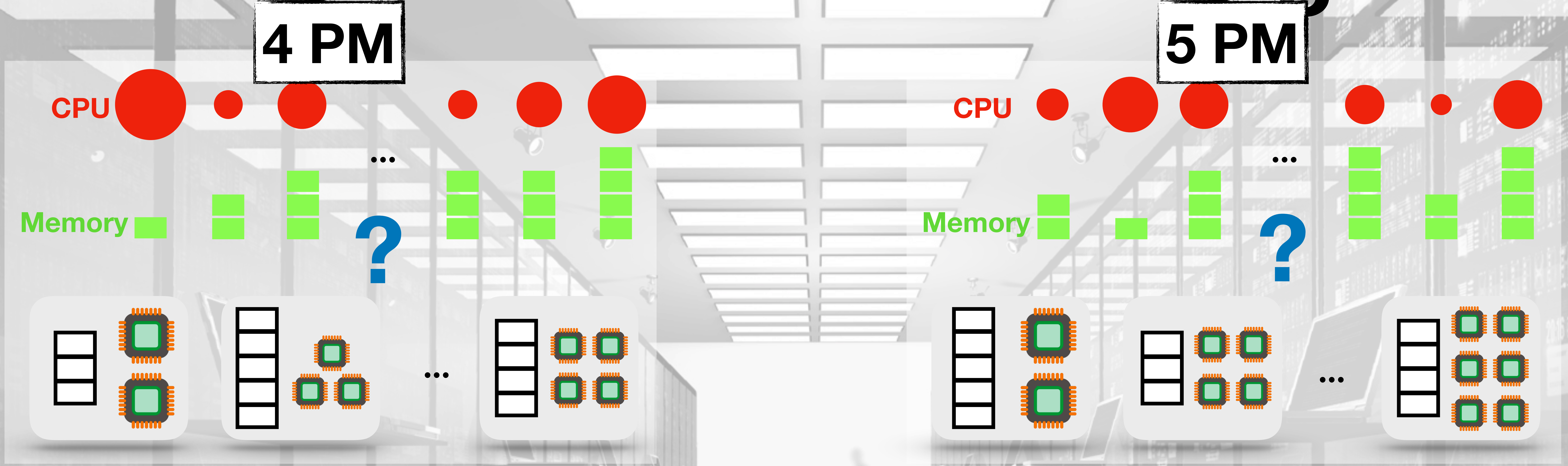
ions

Data Center Resource Management

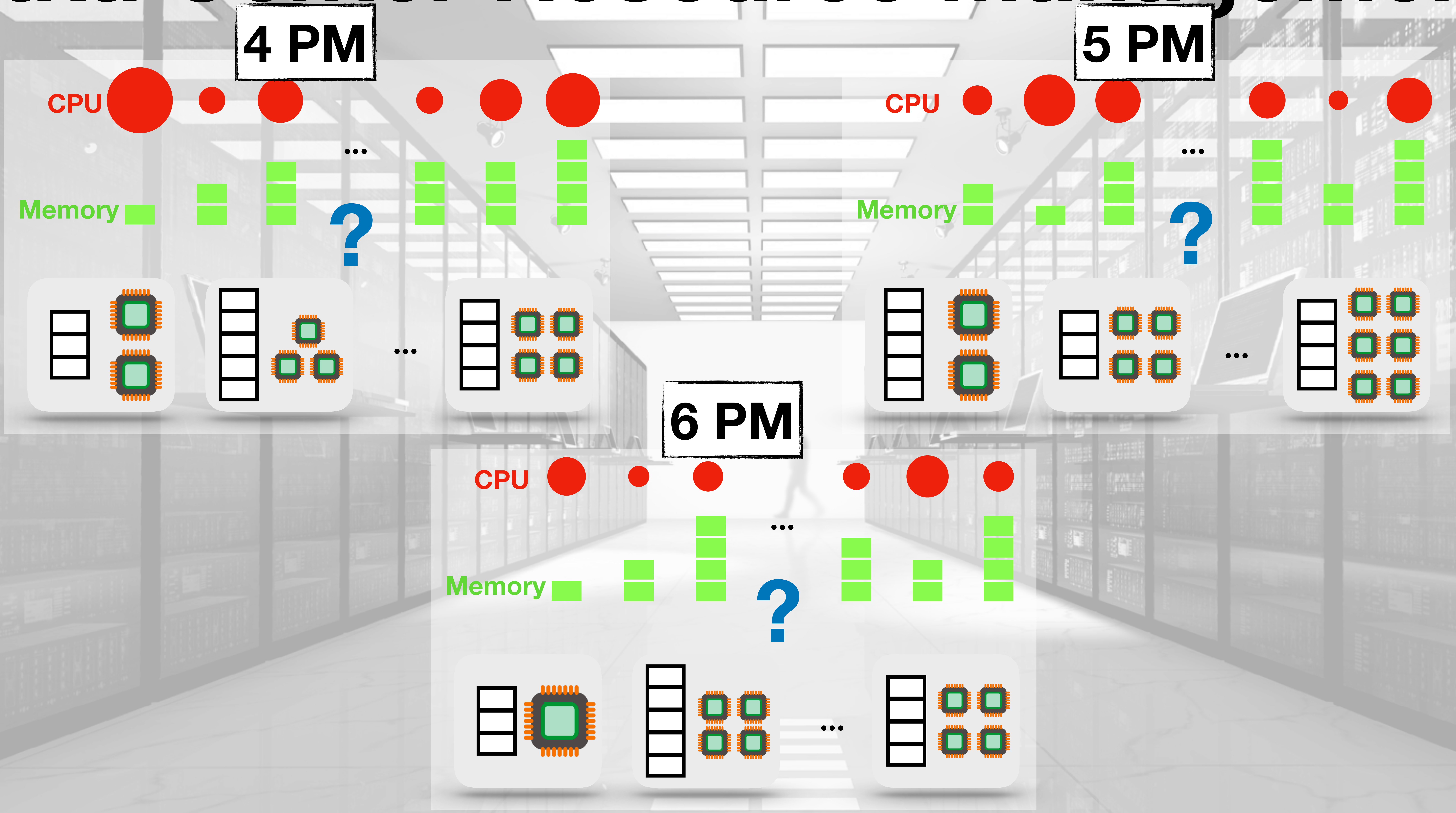
Data Center Resource Management



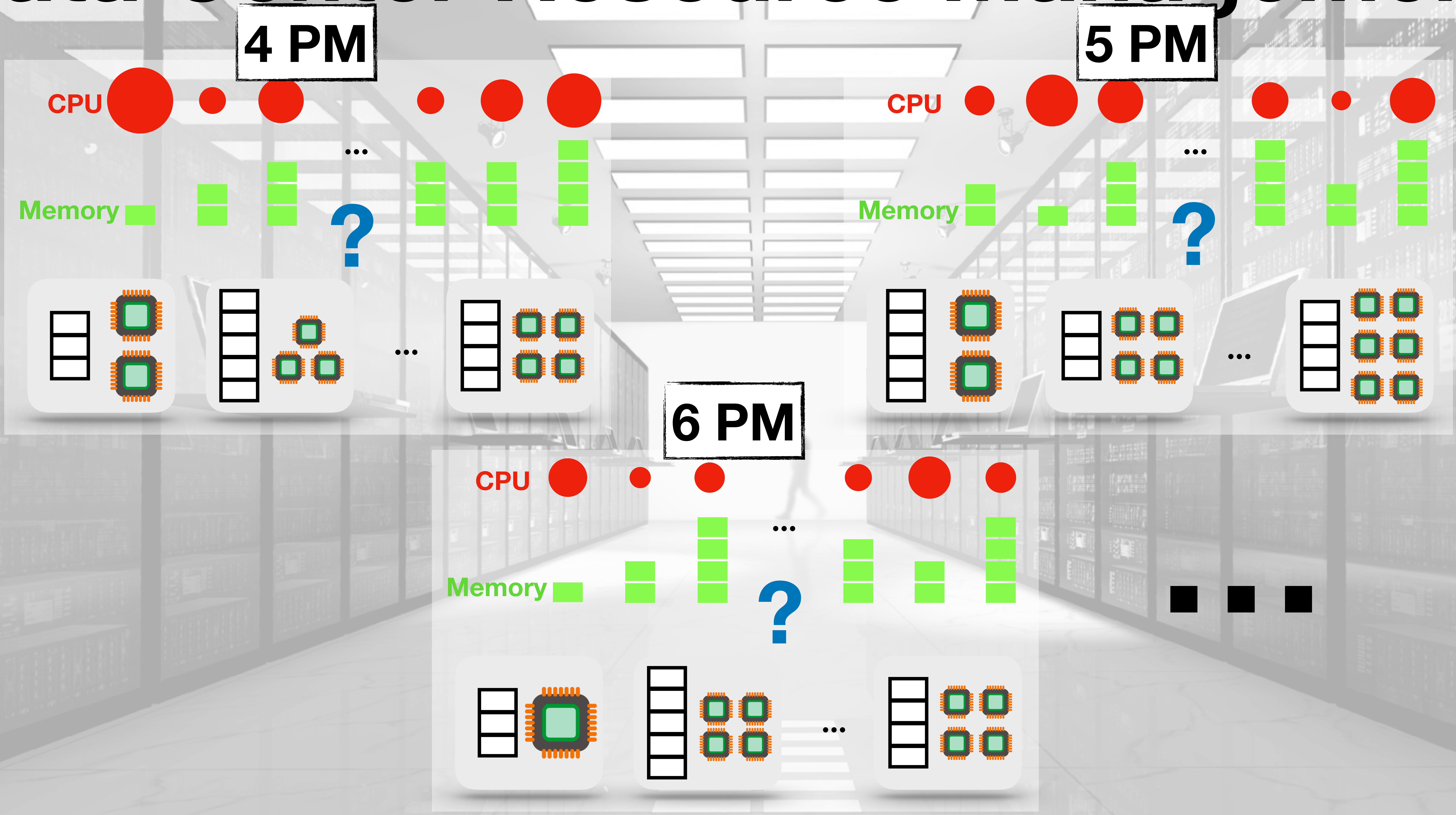
Data Center Resource Management



Data Center Resource Management



Data Center Resource Management



Tackling NP-Hard Problems

Paradigm	Design Rationale

Tackling NP-Hard Problems

Paradigm	Design Rationale
Exhaustive Search	**Tight formulations** Powerful **Branch-and-Bound** solvers

Tackling NP-Hard Problems

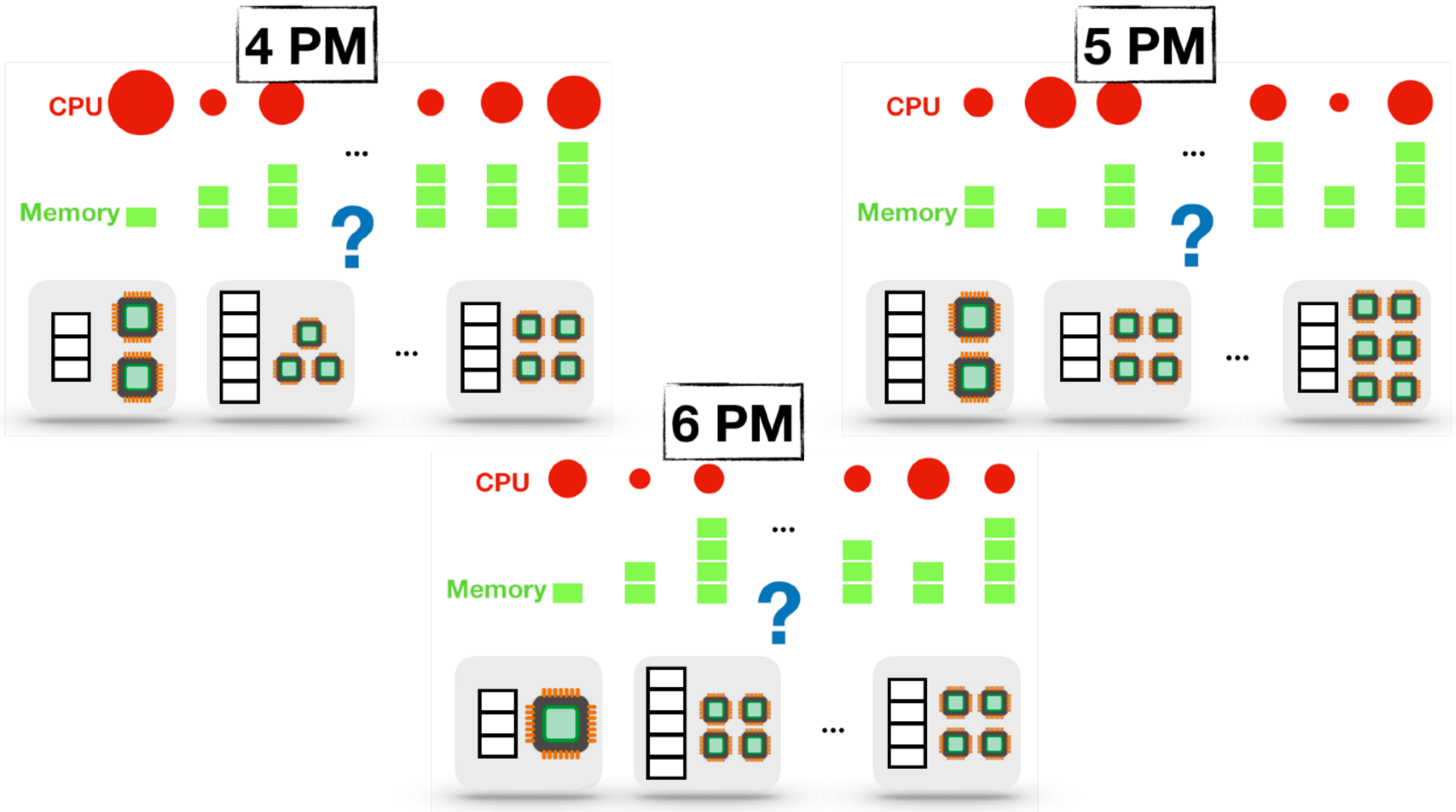
Paradigm	Design Rationale
Exhaustive Search	**Tight formulations** Powerful **Branch-and-Bound** solvers
Approximation Algorithms	**Good worst-case** guarantees

Tackling NP-Hard Problems

Paradigm	Design Rationale
Exhaustive Search	**Tight formulations** Powerful **Branch-and-Bound** solvers
Approximation Algorithms	**Good worst-case** guarantees
Heuristics	**Intuition** exploiting problem structure Empirical **trial-and-error**

Tackling NP-Hard Problems

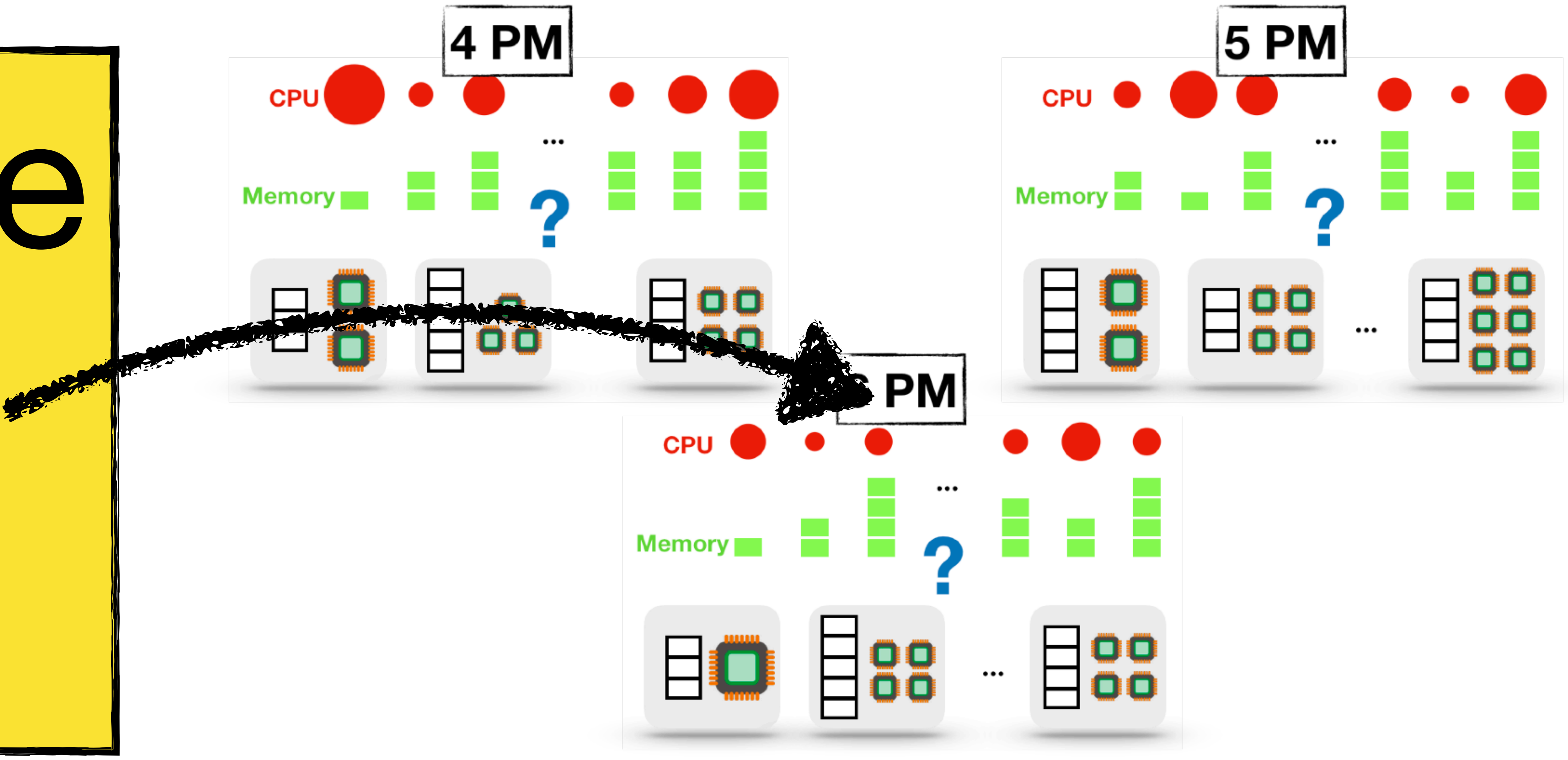
Paradigm	Design Rationale
Exhaustive Search	**Tight formulations** Powerful **Branch-and-Bound** solvers
Approximation Algorithms	**Good worst-case** guarantees
Heuristics	**Intuition** exploiting problem structure Empirical **trial-and-error**



Tackling NP-Hard Problems

Paradigm	Design Rationale
Exhaustive Search	**Tight formulations** Powerful **Branch-and-Bound** solvers
Approximation Algorithms	**Good worst-case** guarantees
Heuristics	**Intuition** exploiting problem structure Empirical **trial-and-error**

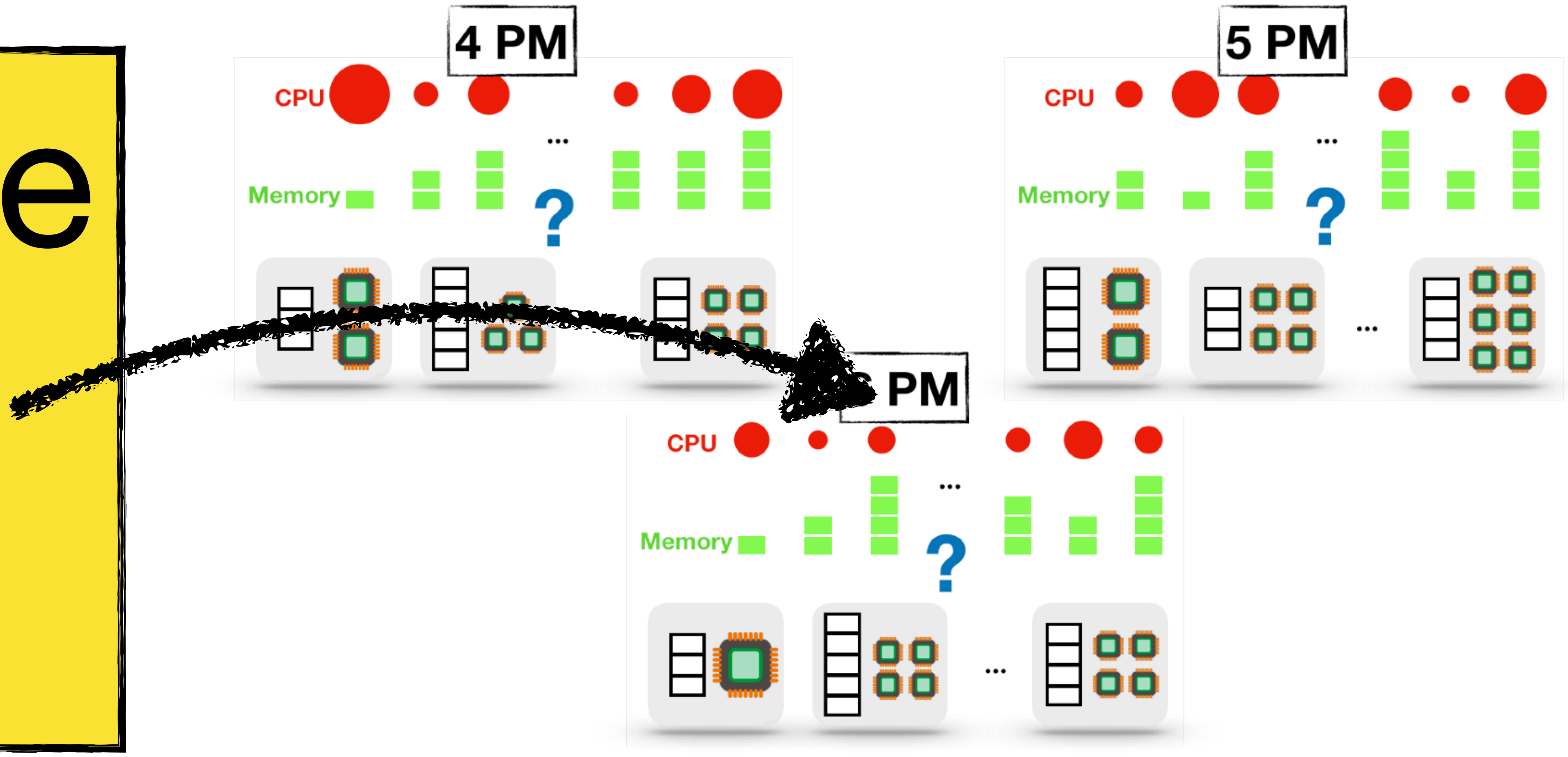
How do you **tailor** the algorithm to **YOUR** instances?



Tackling NP-Hard Problems

Paradigm	Customization via...
Exhaustive Search	**Tight formulations** Powerful **Branch-and-Bound** solvers
Approximation Algorithms	**Good worst-case** guarantees
Heuristics	**Intuition** exploiting problem structure Empirical **trial-and-error**

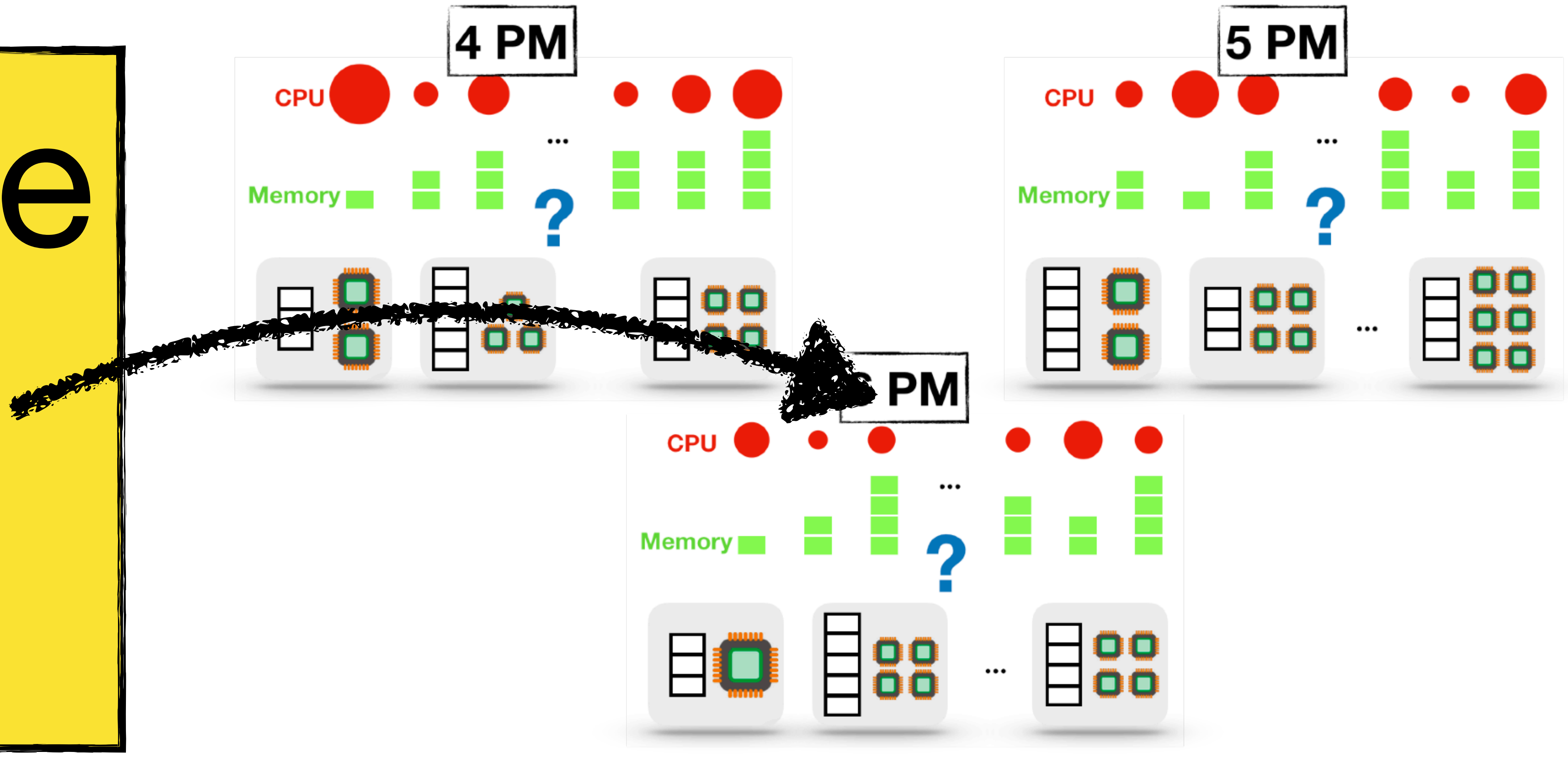
How do you **tailor** the algorithm to **YOUR** instances?



Tackling NP-Hard Problems

Paradigm	Customization via...
Exhaustive Search	**Problem-Specific Bounding** functions or **search rules**
Approximation Algorithms	**Good worst-case** guarantees
Heuristics	**Intuition** exploiting problem structure Empirical **trial-and-error**

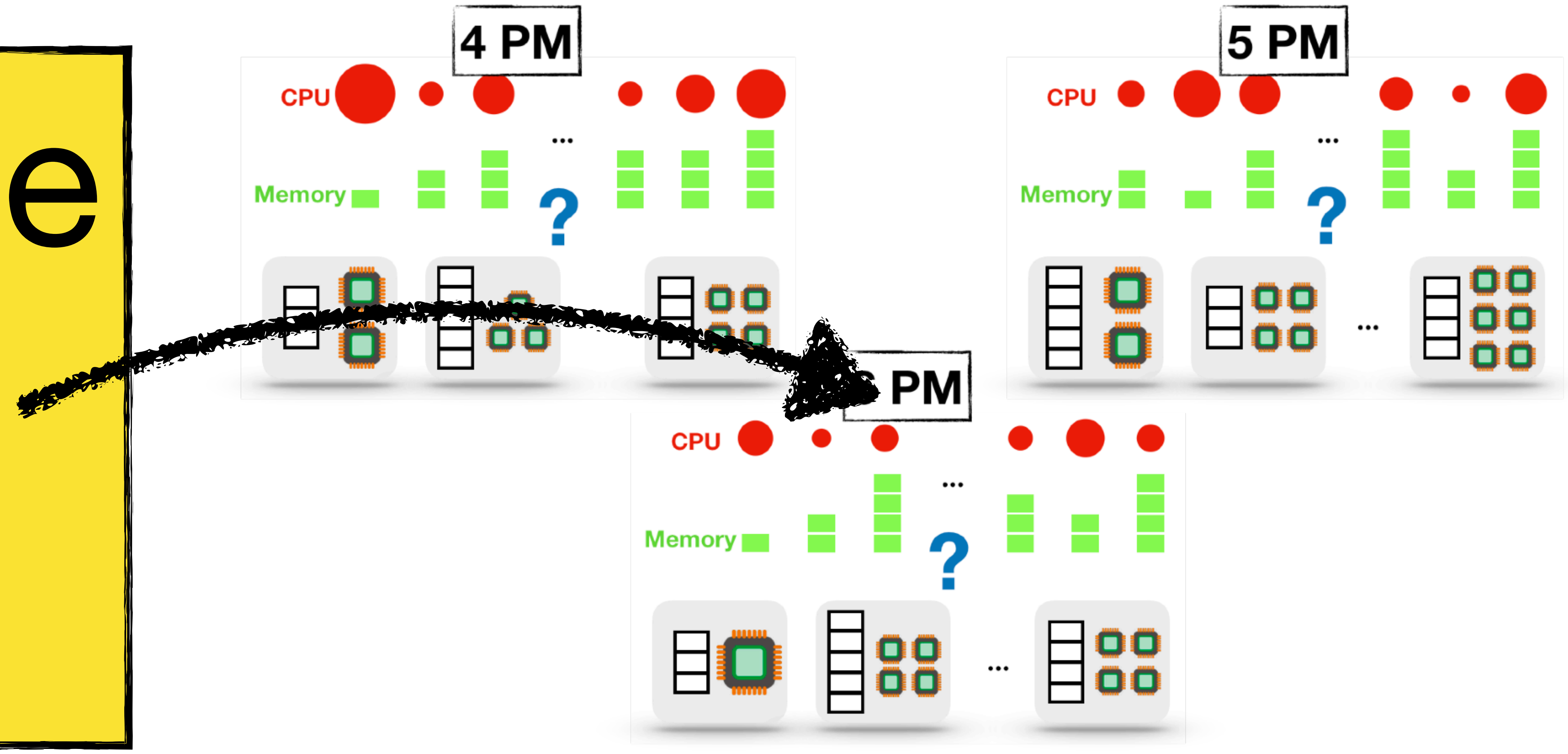
How do you **tailor** the algorithm to **YOUR** instances?



Tackling NP-Hard Problems

Paradigm	Customization via...
Exhaustive Search	**Problem-Specific Bounding** functions or **search rules**
Approximation Algorithms	Make explicit **assumptions** on input distribution and **redesign algo.**
Heuristics	**Intuition** exploiting problem structure Empirical **trial-and-error**

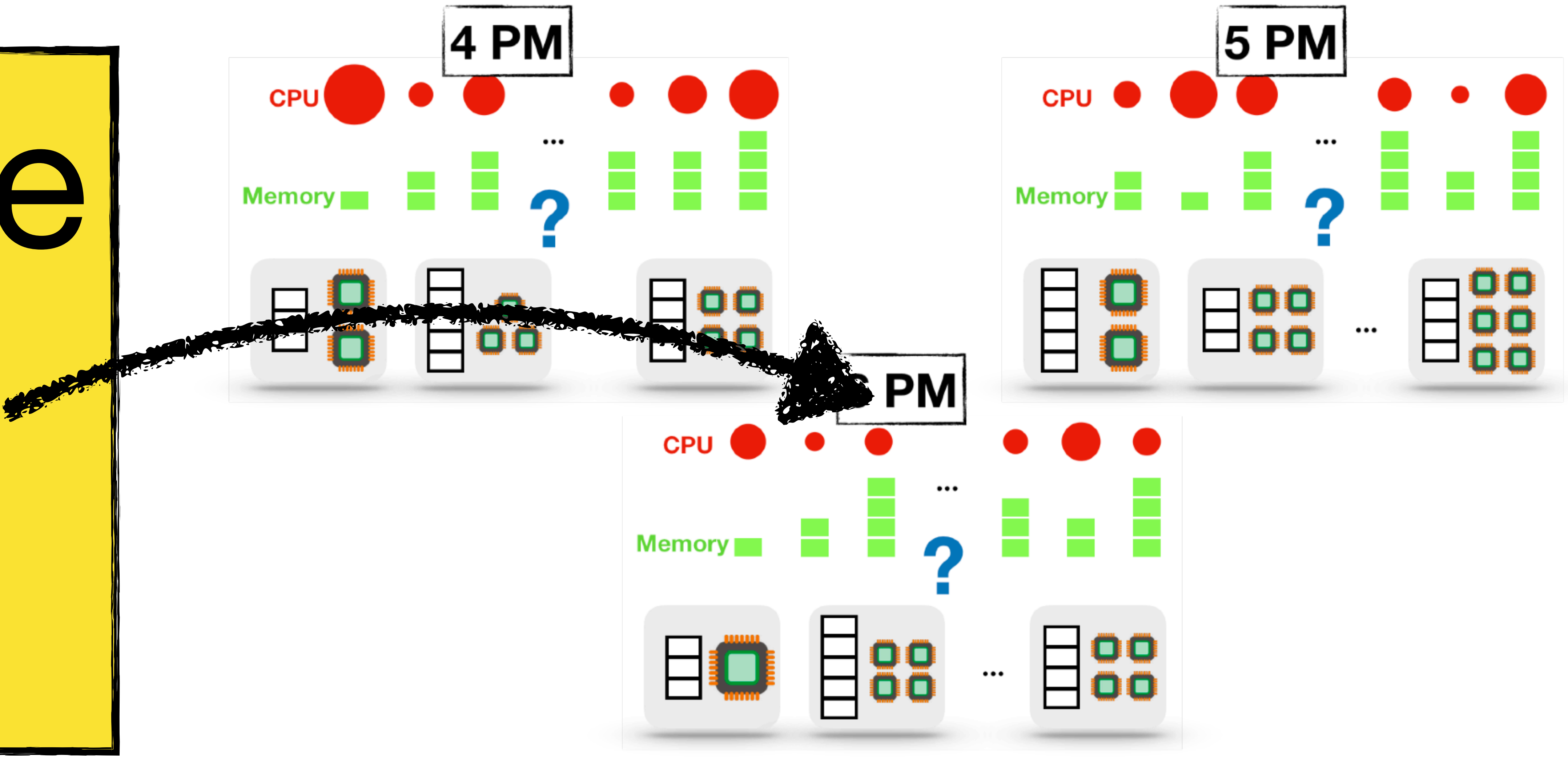
How do you **tailor** the algorithm to **YOUR** instances?



Tackling NP-Hard Problems

Paradigm	Customization via...
Exhaustive Search	**Problem-Specific Bounding** functions or **search rules**
Approximation Algorithms	Make explicit **assumptions** on input distribution and **redesign algo.**
Heuristics	**Analyze** algorithm **behavior** on your inputs; look for **patterns** to exploit

How do you **tailor** the algorithm to **YOUR** instances?

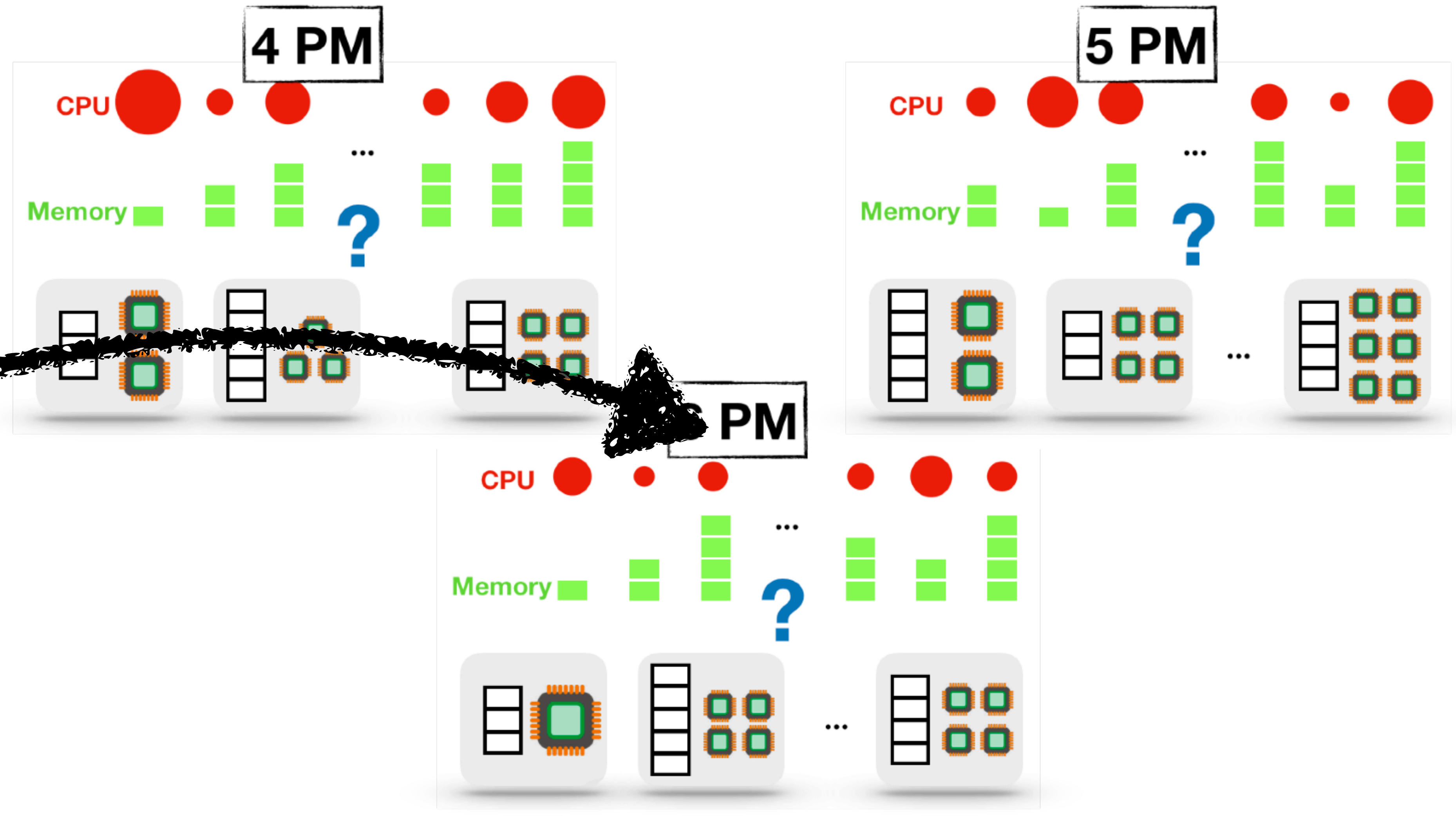


Tackling NP-Hard Problems

Paradigm	Customization via...
	Problem-Specific Bounding functions or **search rules**
	Make explicit **assumptions** on input distribution and **redesign algo.**
	Analyze algorithm **behavior** on your inputs; look for **patterns** to exploit

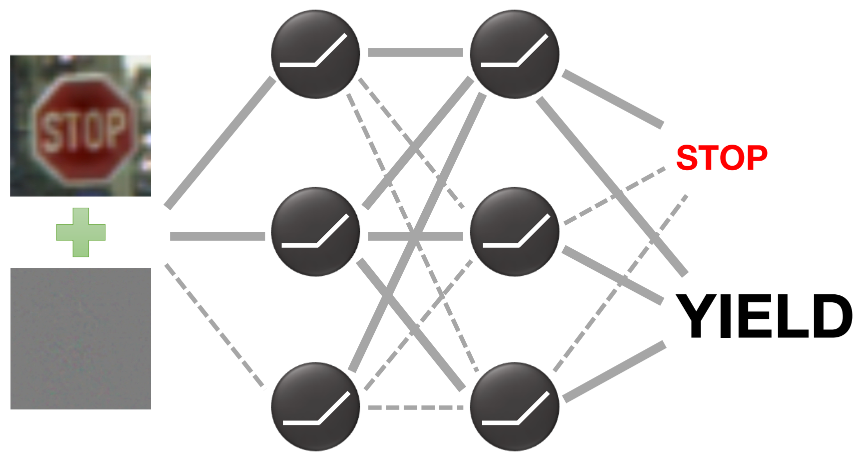
ANSWER:
Manual intellectual/ experimental **effort** required

How do you **tailor** the algorithm to **YOUR** instances?



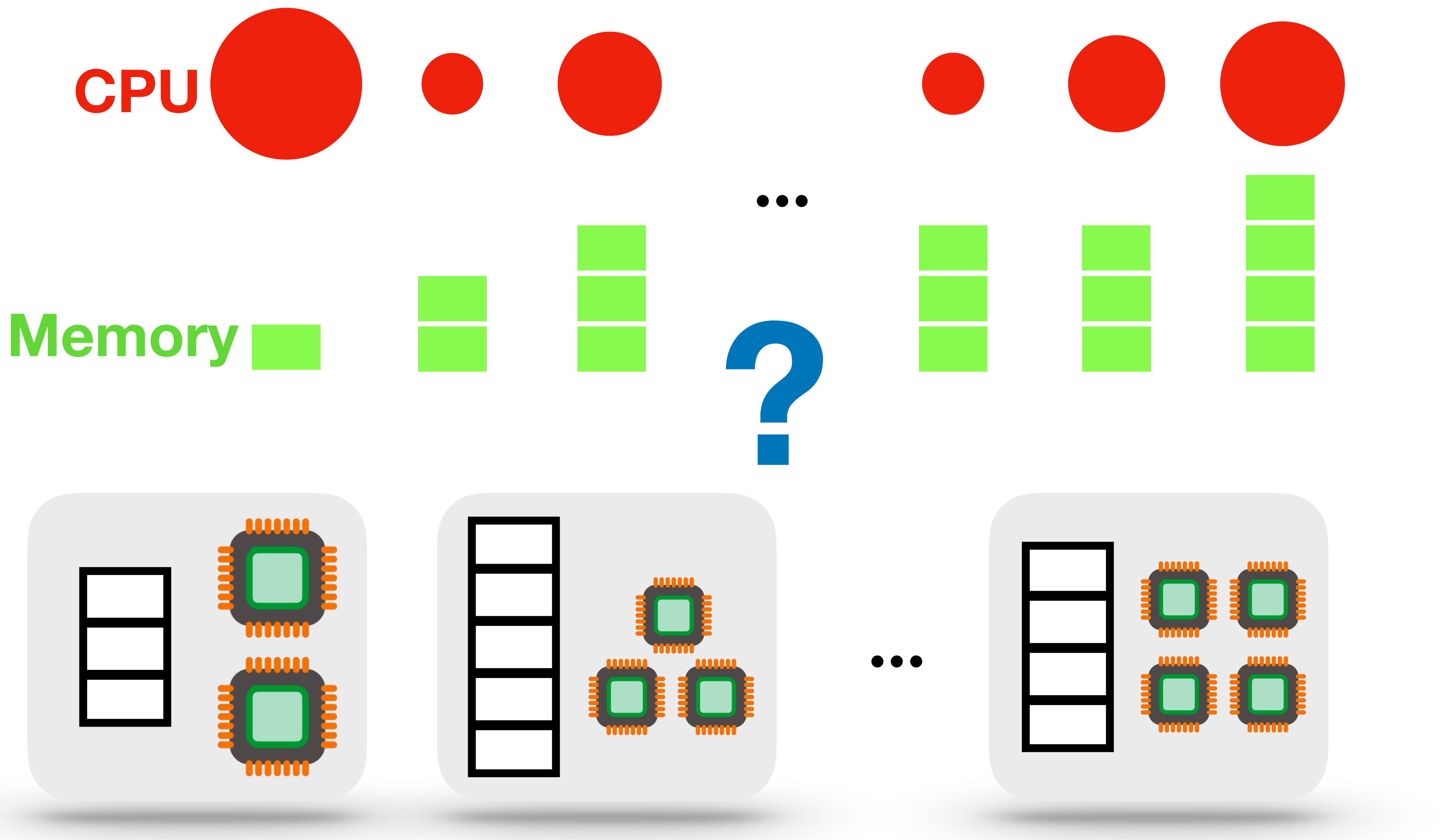
Opportunity

Adversarial ML

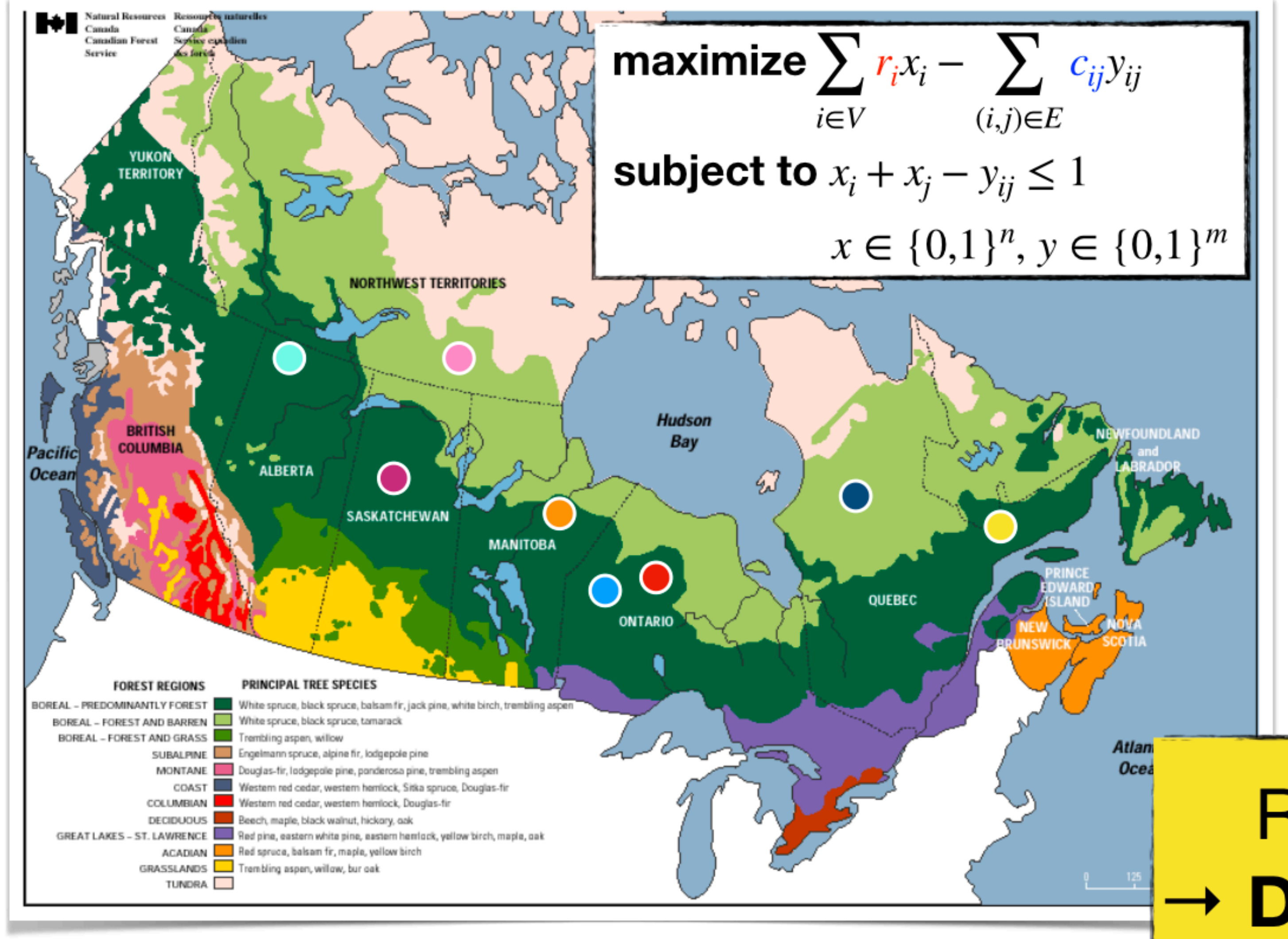


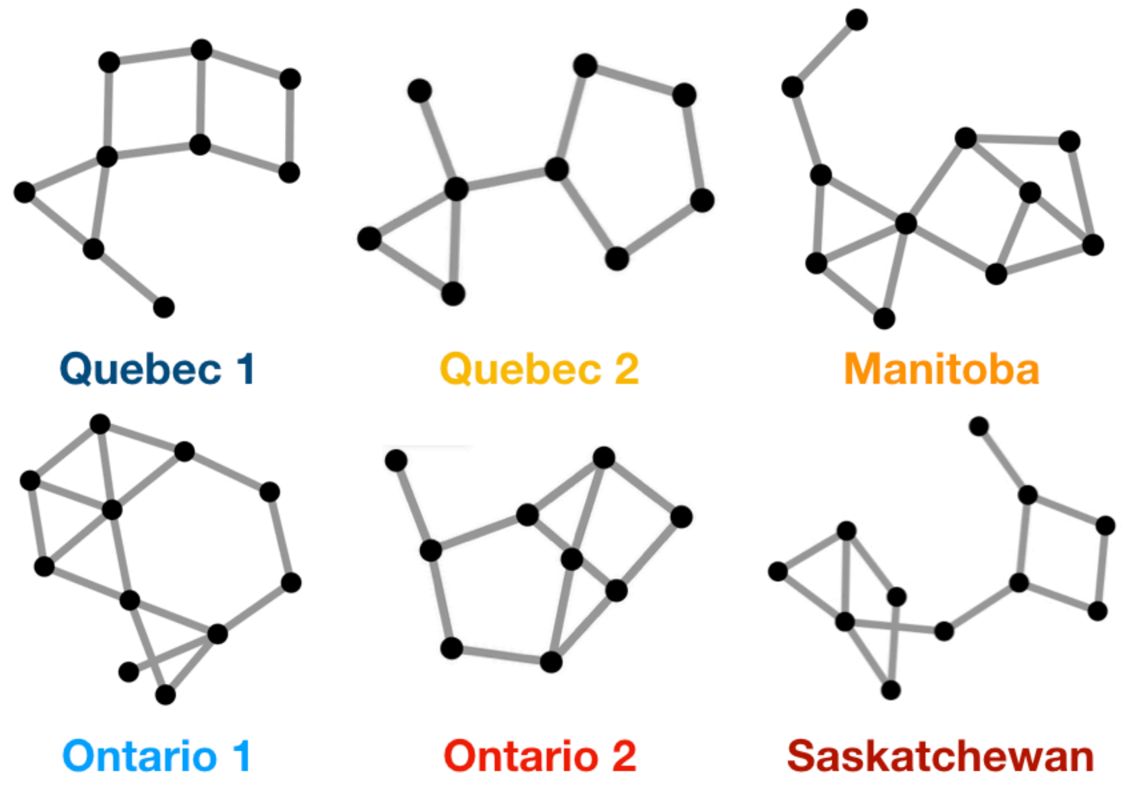
Automatically tailor algorithms to a **family** of **instances**

Data Center Resource Management



Forest Harvesting



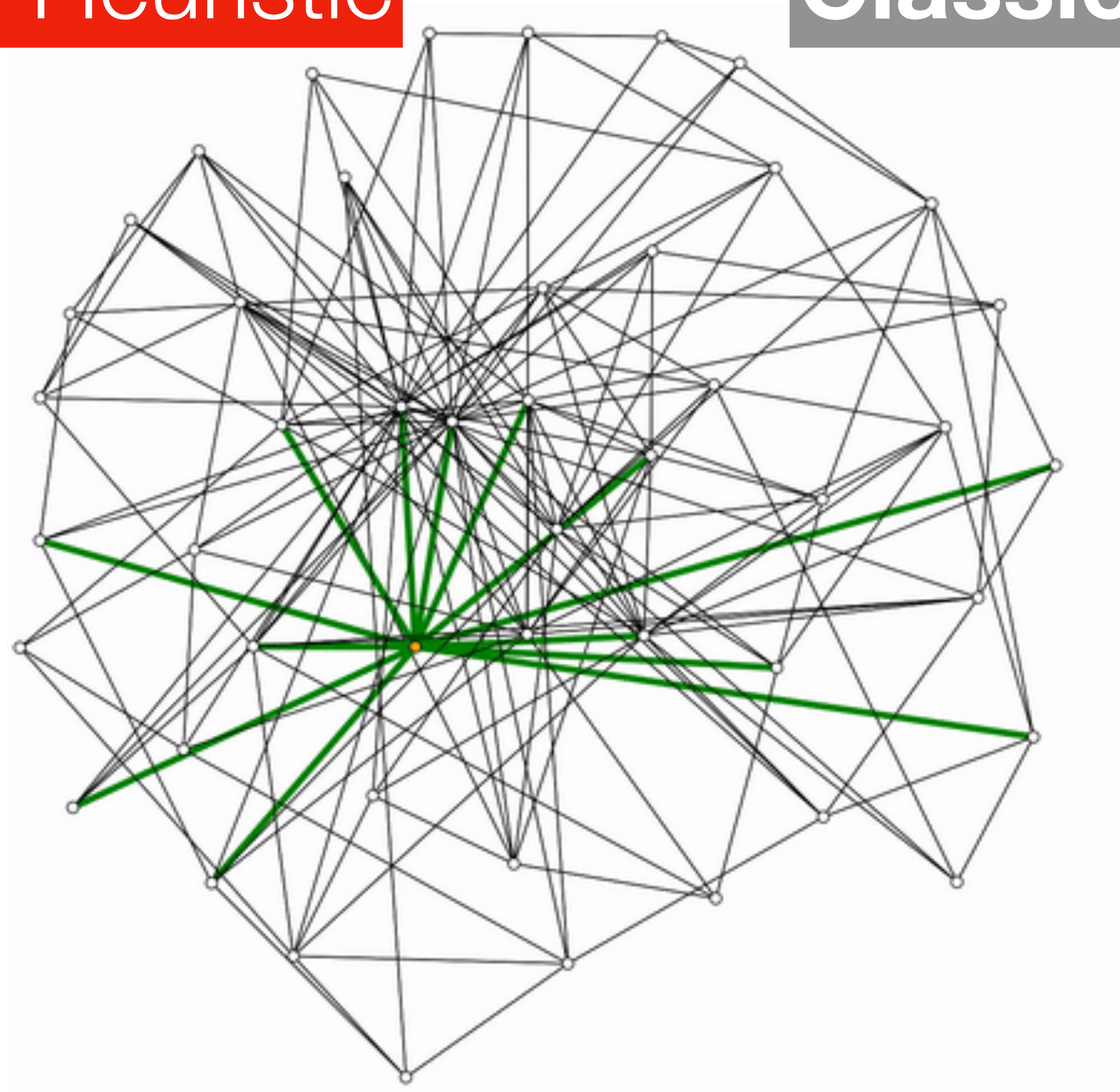


Revenue varies **over time**
→ **Different** instances, **same** problem

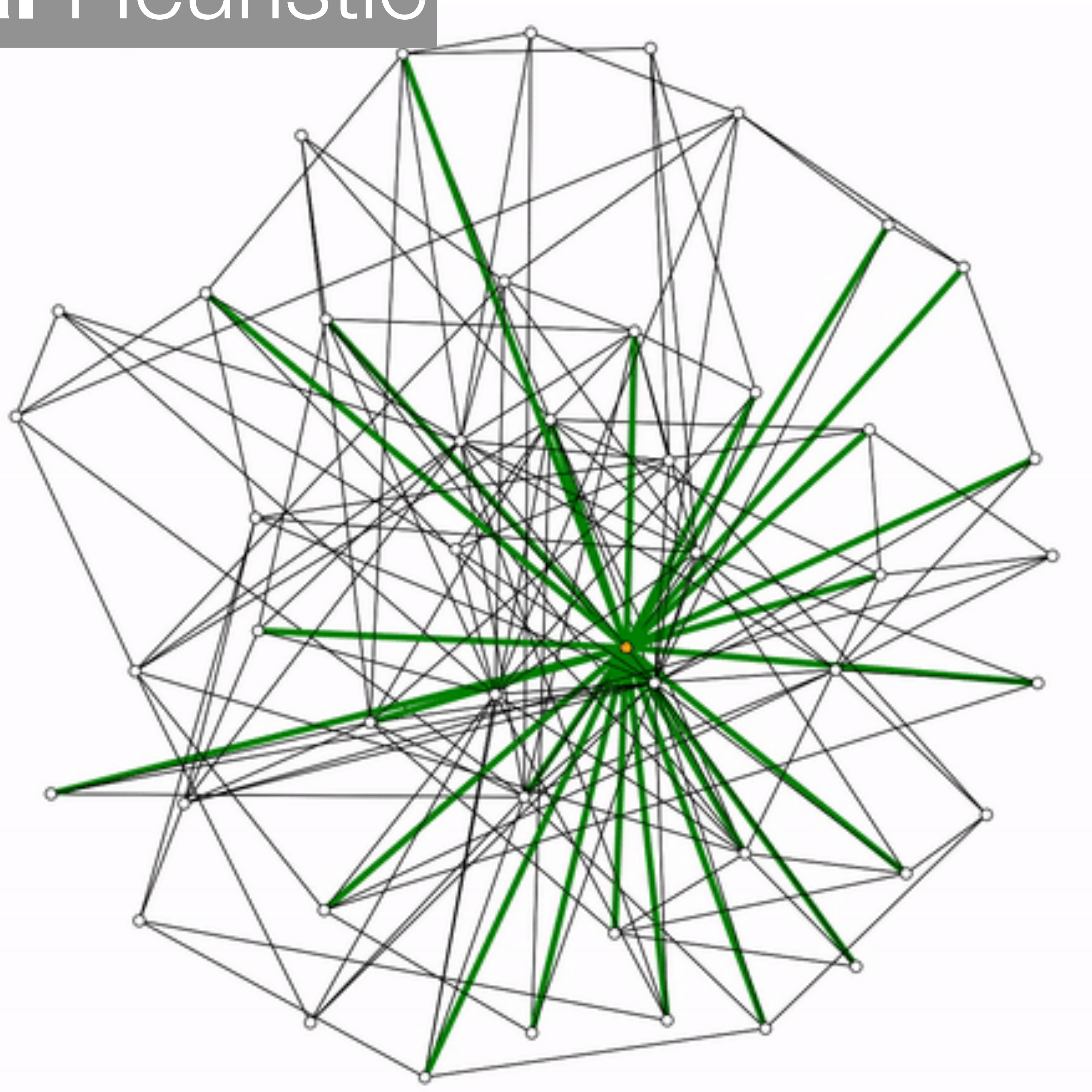
Data-Driven Algorithm Design

automatically **discovers** **novel** search **strategies**

Learned Heuristic



Classical Heuristic



Minimum Vertex Cover

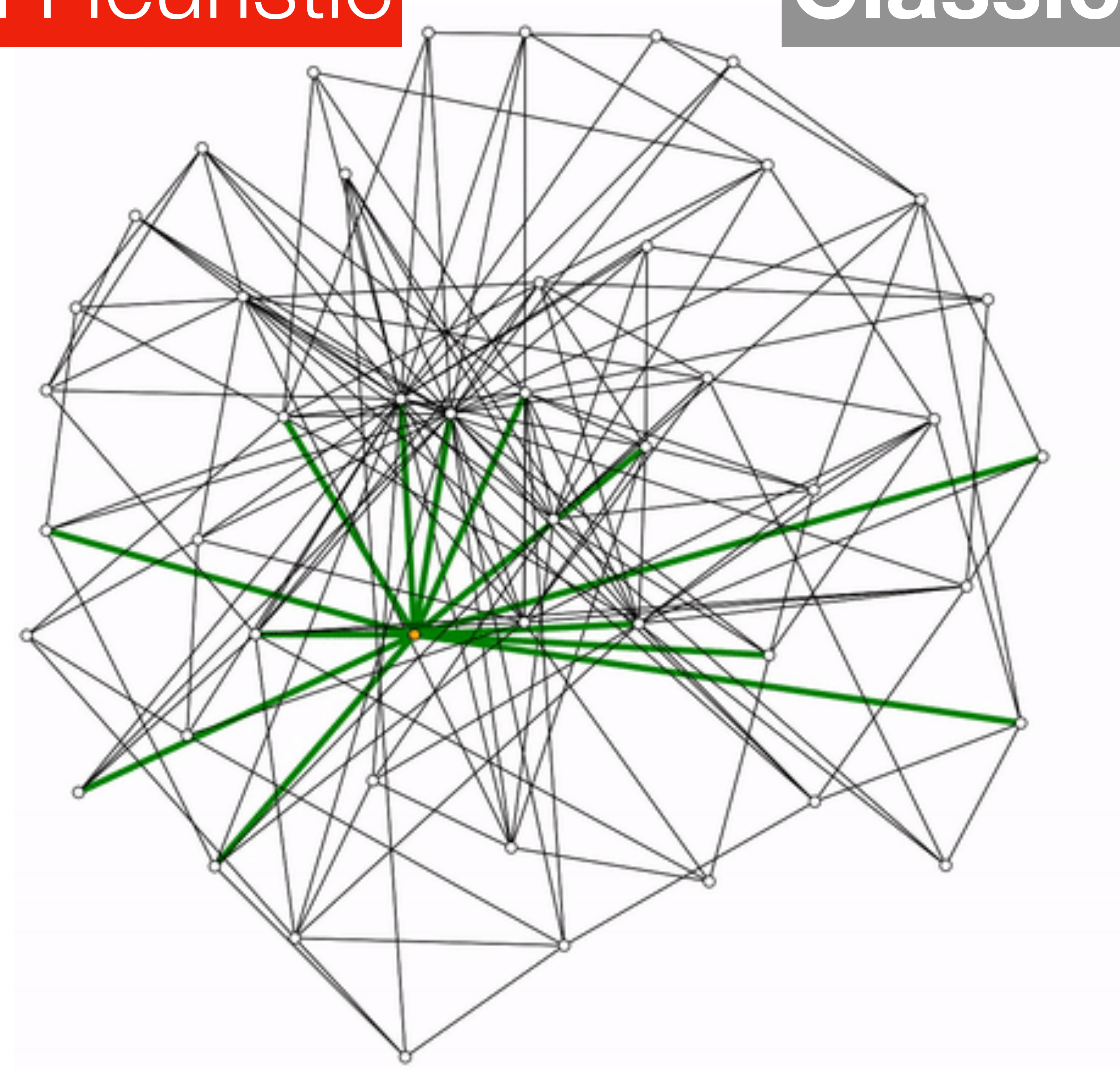
Find **smallest vertex subset** such that each **edge is covered**

Data-Driven Algorithm Design

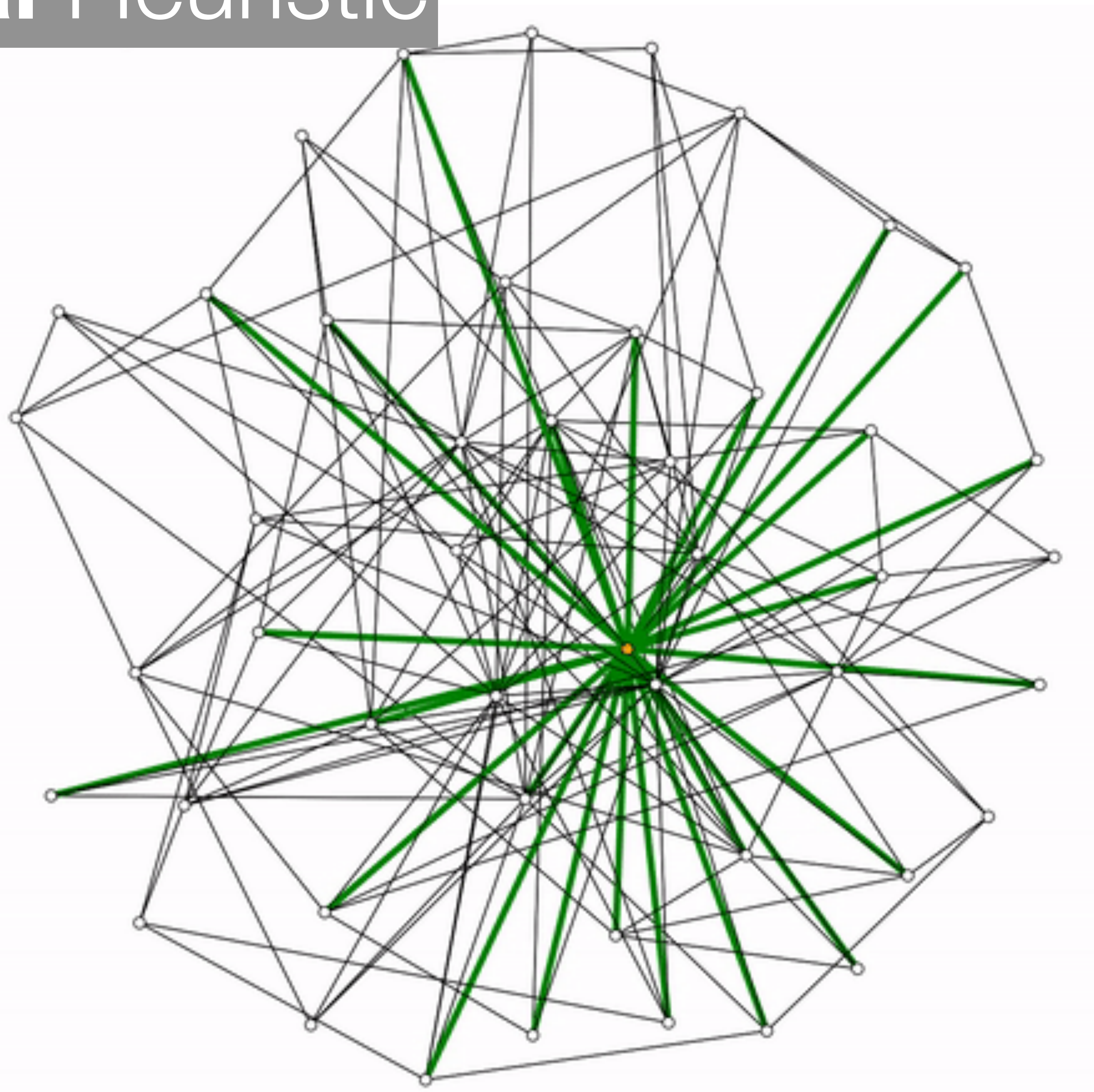
automatically **discovers**

novel search **strategies**

Learned Heuristic



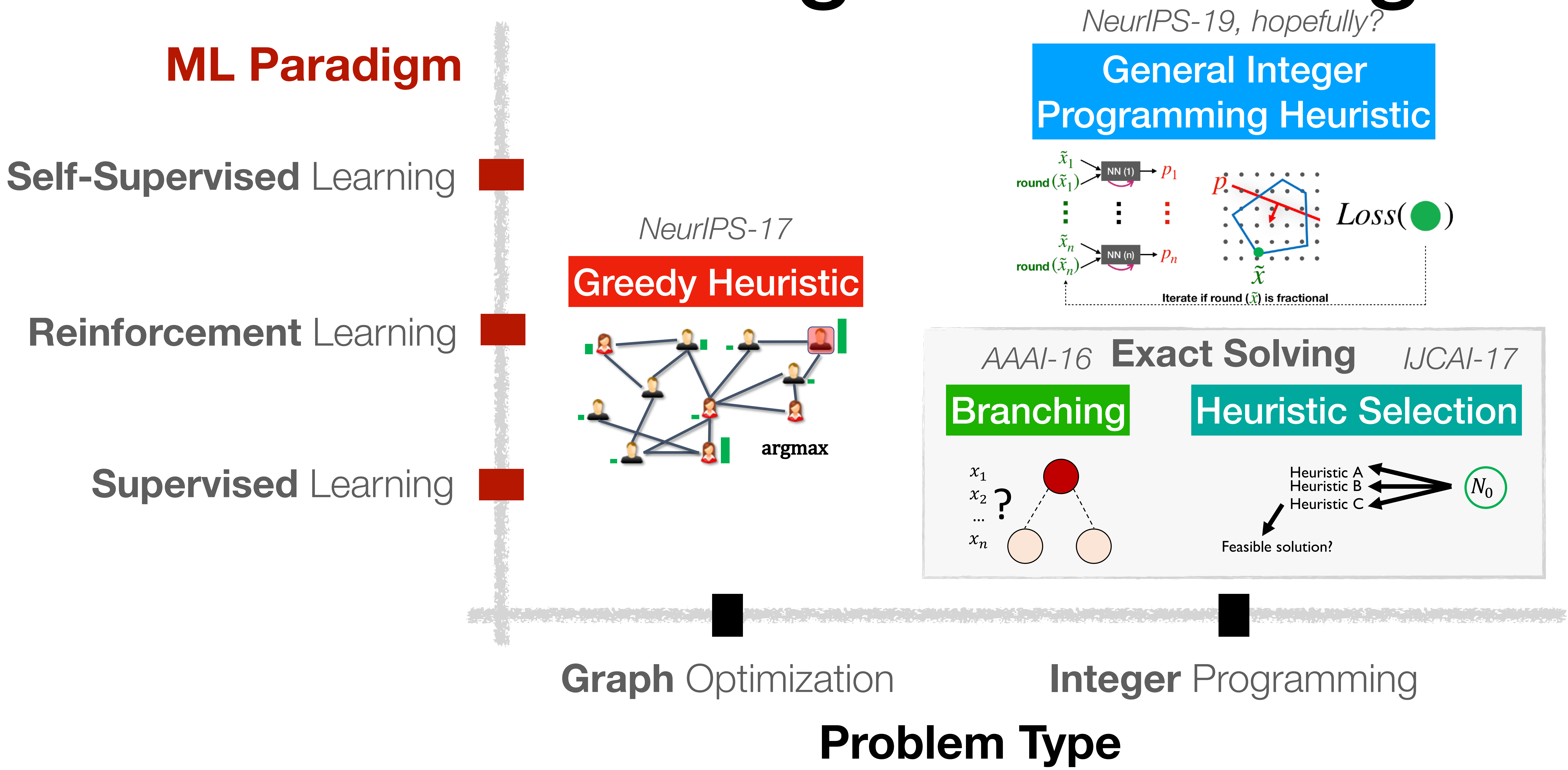
Classical Heuristic



Minimum Vertex Cover

Find **smallest vertex subset** such that each **edge is covered**

Data-Driven Algorithm Design



Branch & Bound for Integer Optimization

LP-based $\min_x c^T x$ **s.t.** $Ax \leq b, x \in \{0,1\}^n$

Land & Doig, 1960

Repeat:

1 **Select Node**

2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**

Branch & Bound for Integer Optimization

LP-based

$$\min_x c^T x \text{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1 Select Node
2 Solve LP Relaxation
3 Prune?
4 Add Cuts
5 Run Heuristics
6 Branch

N_0

Branch & Bound for Integer Optimization

Land & Doig, 1960

LP-based

$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

$[0,1]^n$

Repeat:

1 **Select Node**

2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**

N_0 Solve LP Relaxation

→ Lower Bound on OPT

Branch & Bound for Integer Optimization

LP-based

$$\min_x c^T x \text{ s.t. } Ax \le b, x \in \{0,1\}^n$$

$[0,1]^n$

Land & Doig, 1960

Repeat:

1 **Select Node**

2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**

N_0

Solve LP Relaxation
→ Lower Bound on OPT

worse than best solution?

Prune!

Branch & Bound for Integer Optimization

LP-based

$$\min_x c^T x \text{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1 **Select Node**

2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**

N_0

Add Cuts:
Tightening Constraints

Branch & Bound for Integer Optimization

LP-based

$$\min_{x} c^T x \textbf{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1 **Select Node**

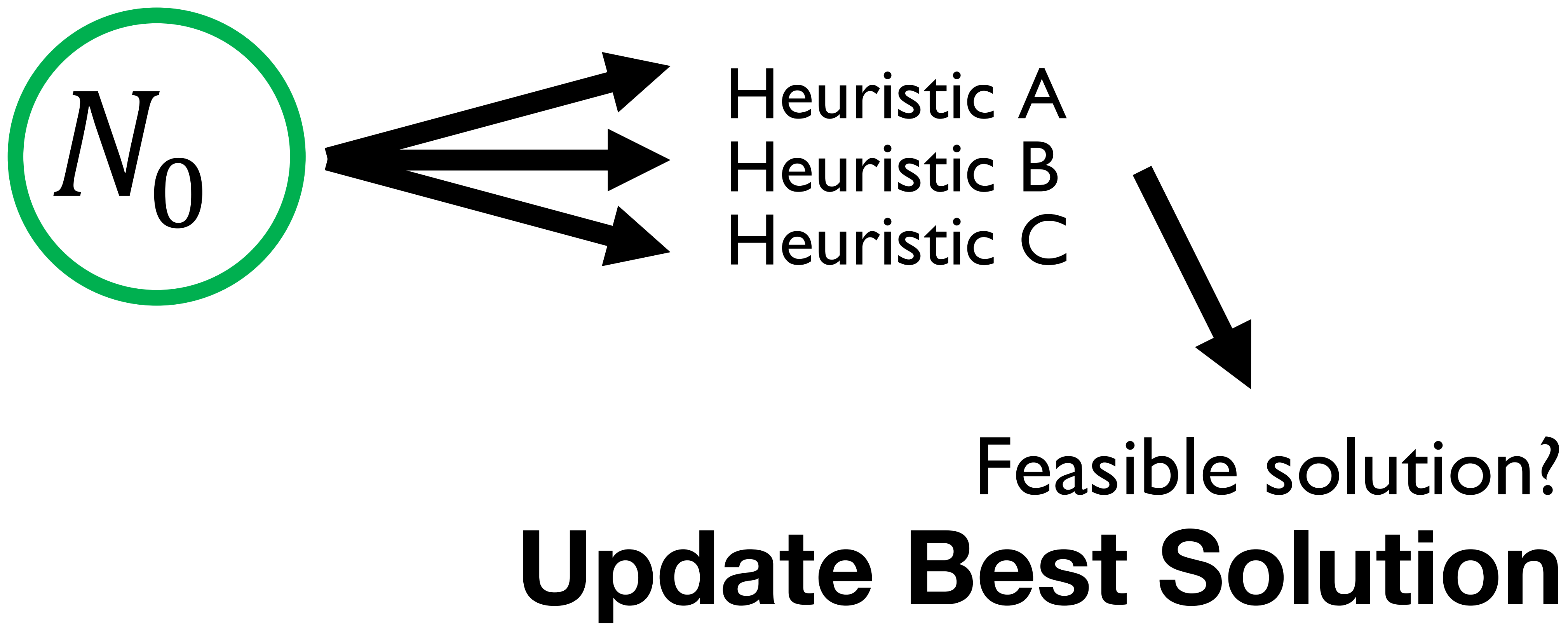
2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 Run Heuristics

6 **Branch**



Branch & Bound for Integer Optimization

LP-based

$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1 Select Node

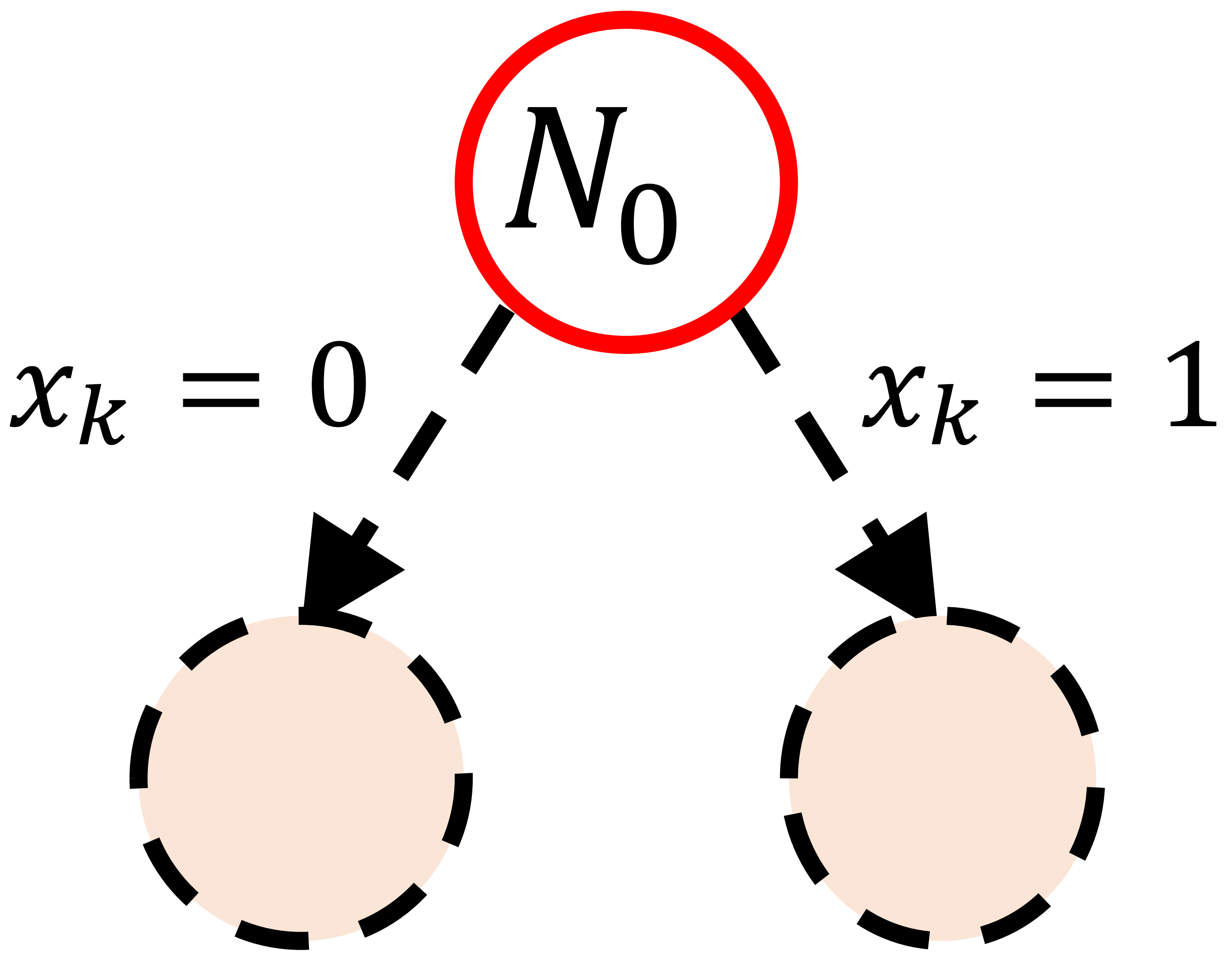
2 Solve LP Relaxation

3 Prune?

4 Add Cuts

5 Run Heuristics

6 Branch



x_1?
x_2?
...
x_n?

Branch & Bound for Integer Optimization

LP-based

$$\min_x c^T x \text{ s.t. } Ax \le b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1 Select Node

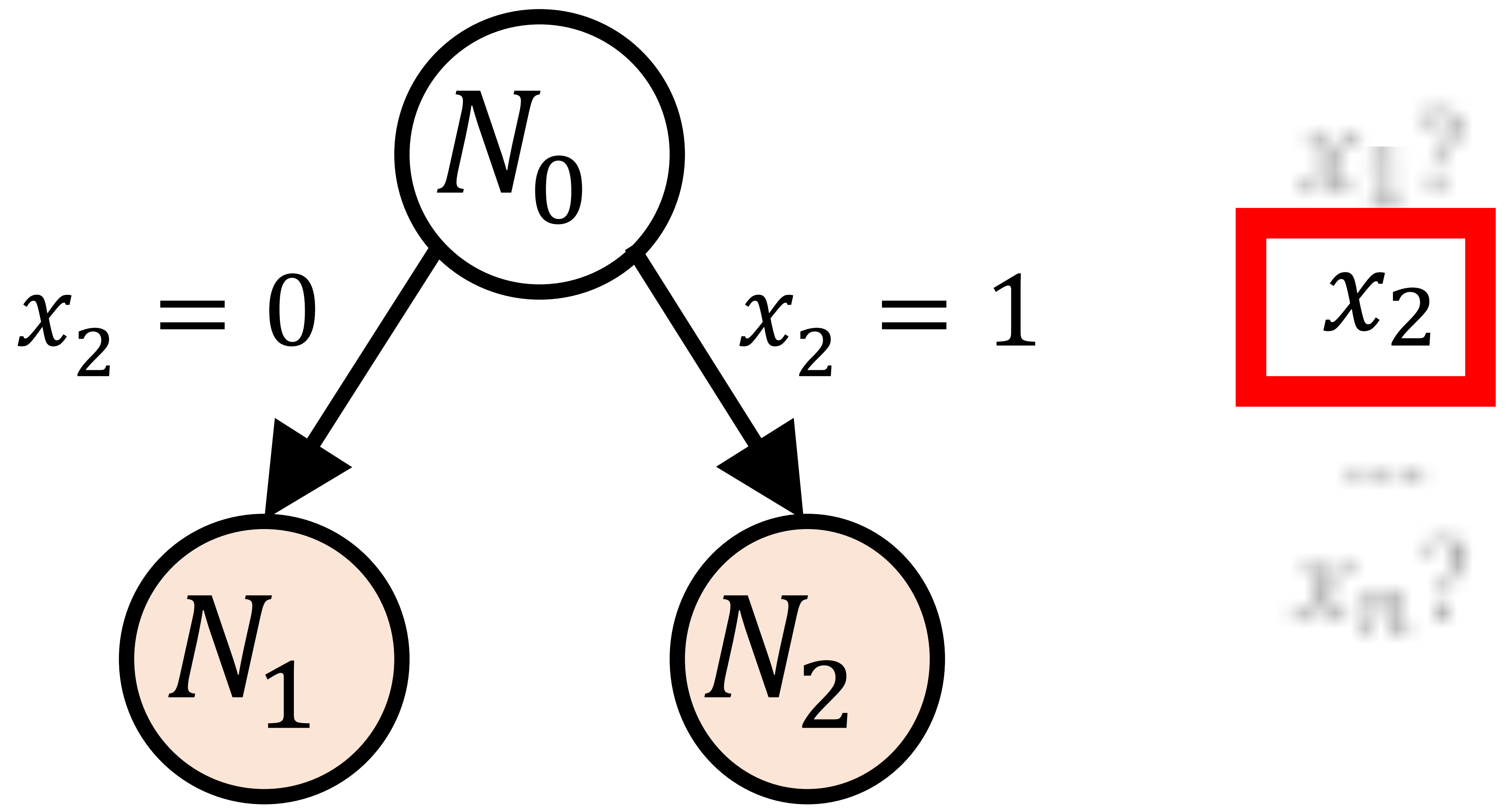
2 Solve LP Relaxation

3 Prune?

4 Add Cuts

5 Run Heuristics

6 Branch



Branch & Bound for Integer Optimization

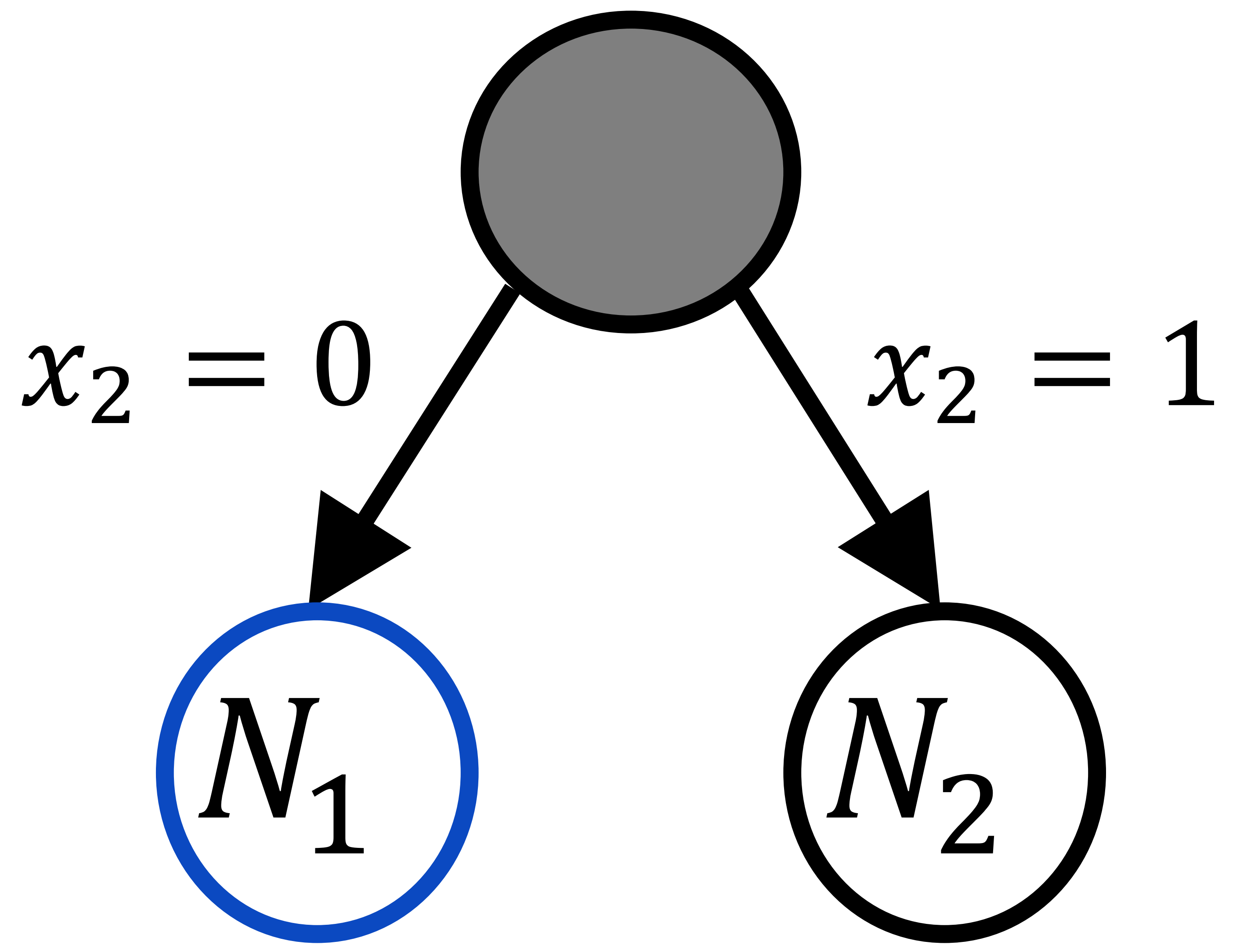
LP-based

$$\min_x c^T x \text{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1. Select Node
2. Solve LP Relaxation
3. Prune?
4. Add Cuts
5. Run Heuristics
6. Branch



Branch & Bound for Integer Optimization

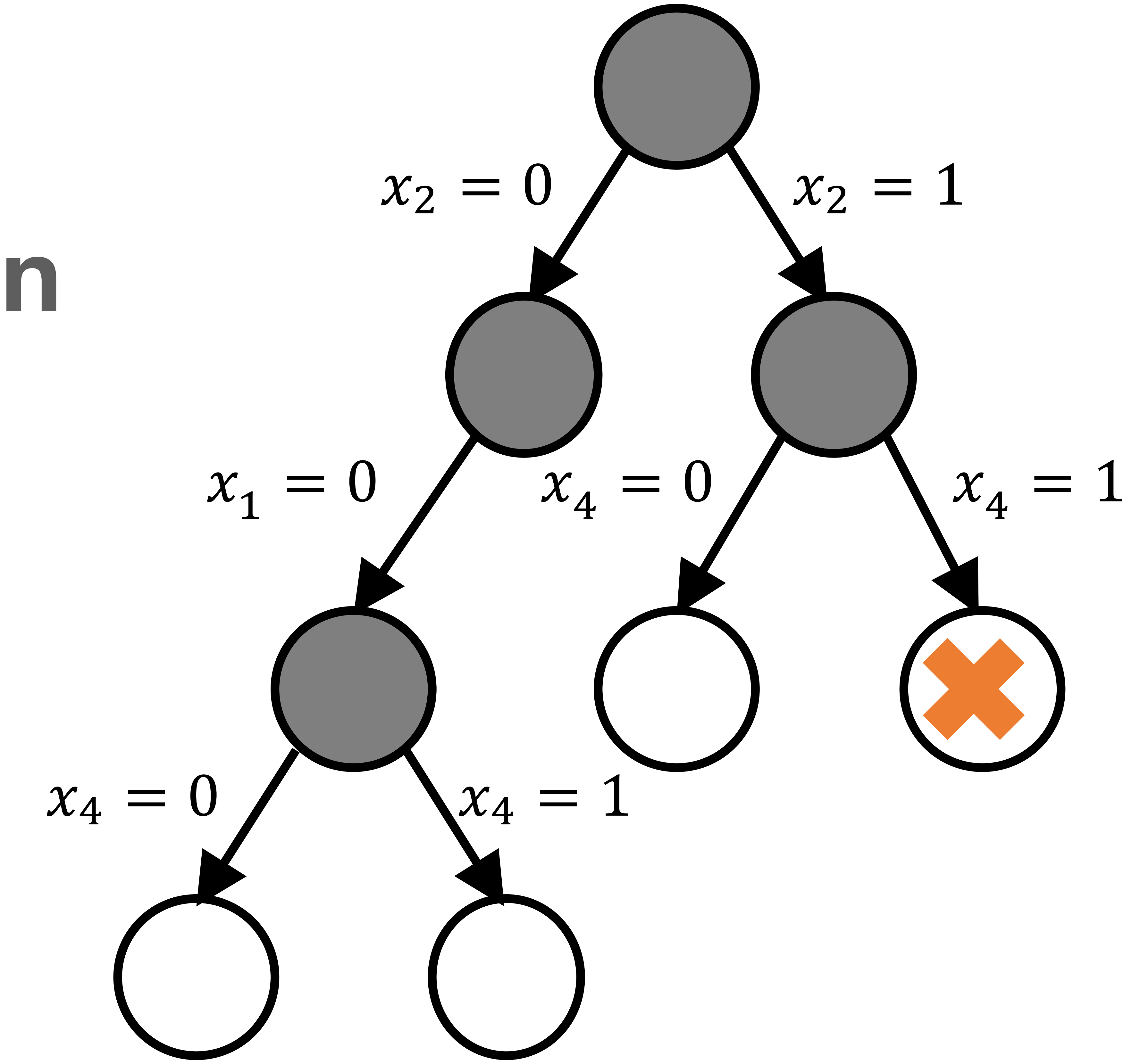
LP-based

$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

Land & Doig, 1960

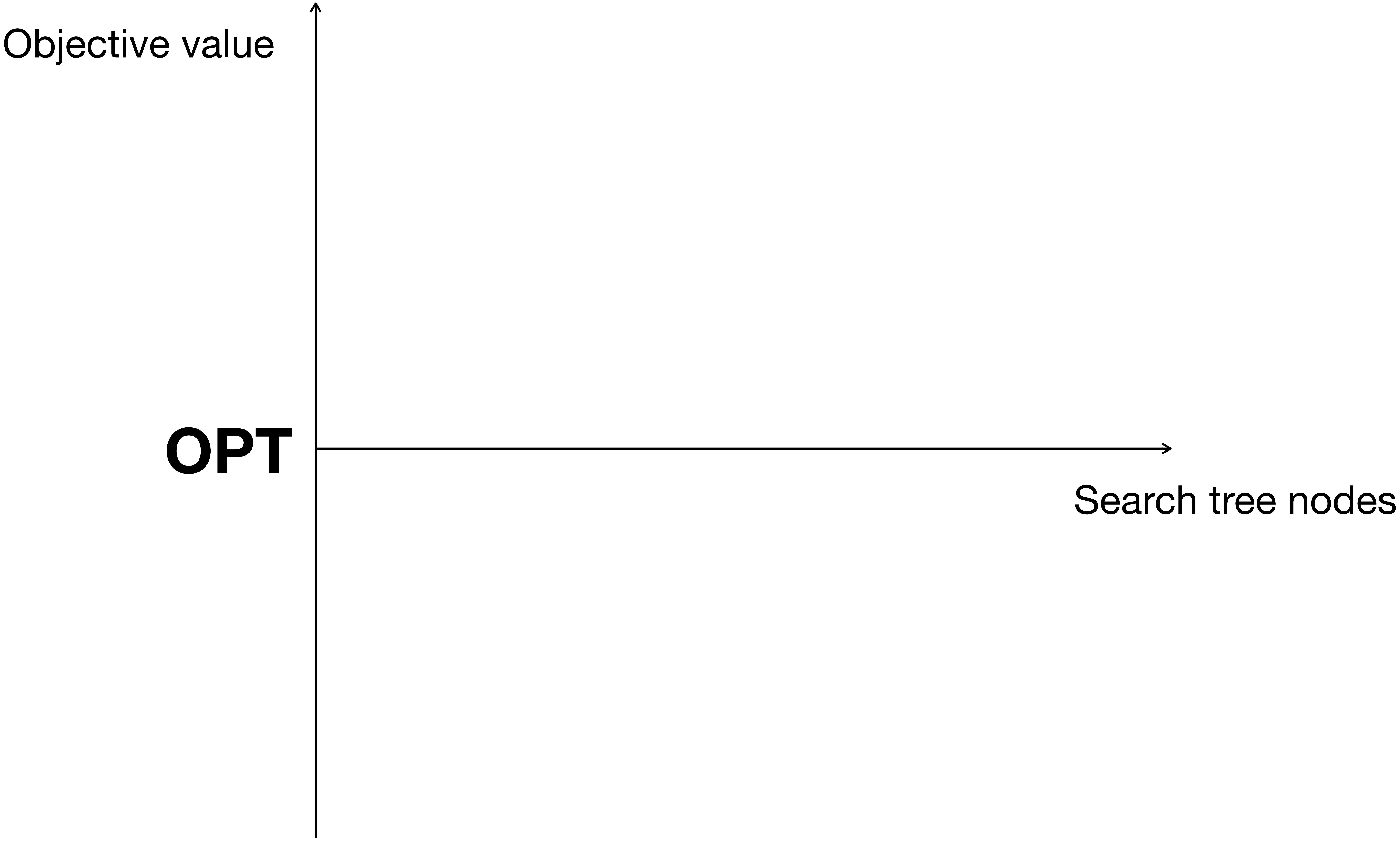
Repeat:

1 Select Node
2 Solve LP Relaxation
3 Prune?
4 Add Cuts
5 Run Heuristics
6 Branch

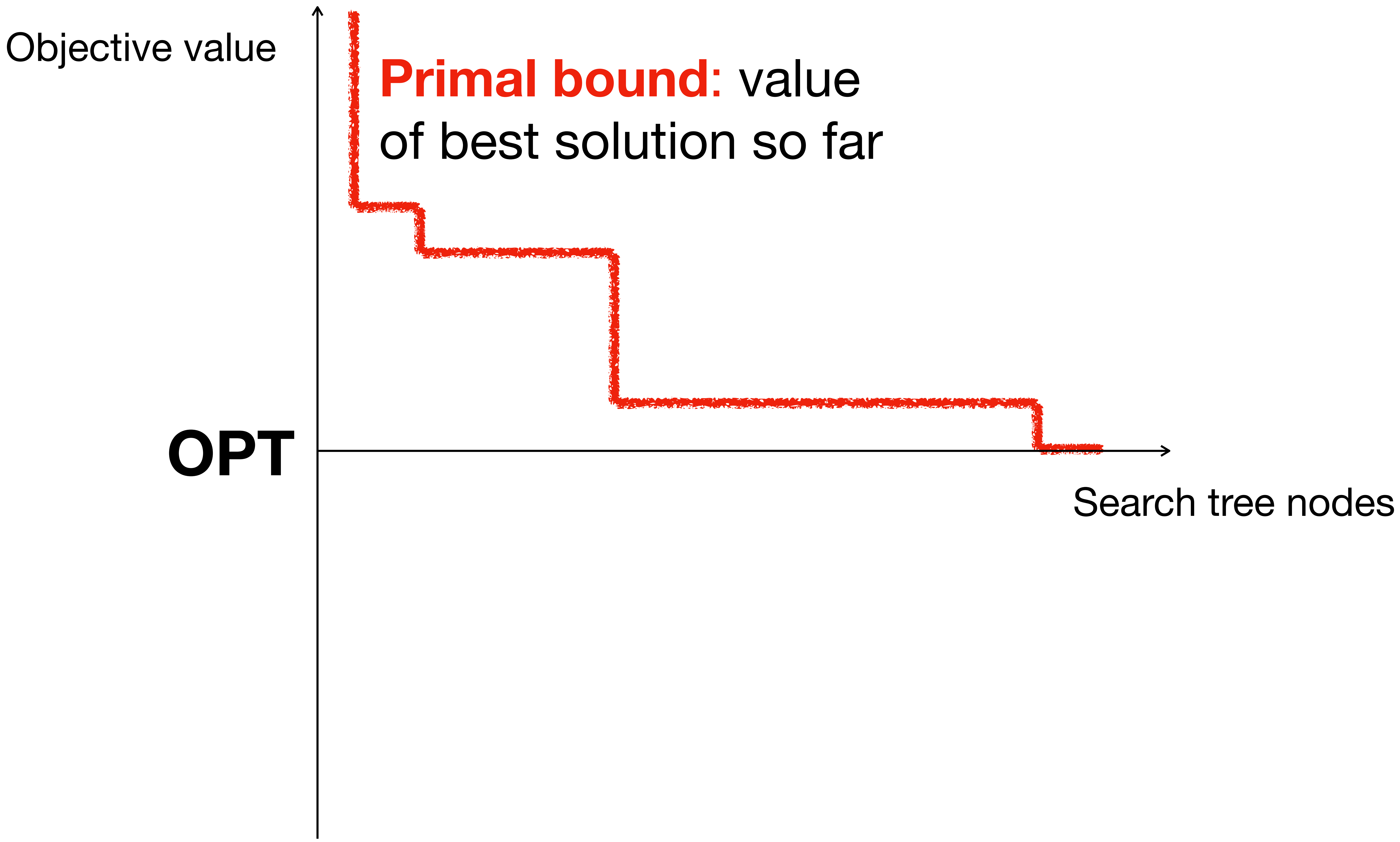


$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

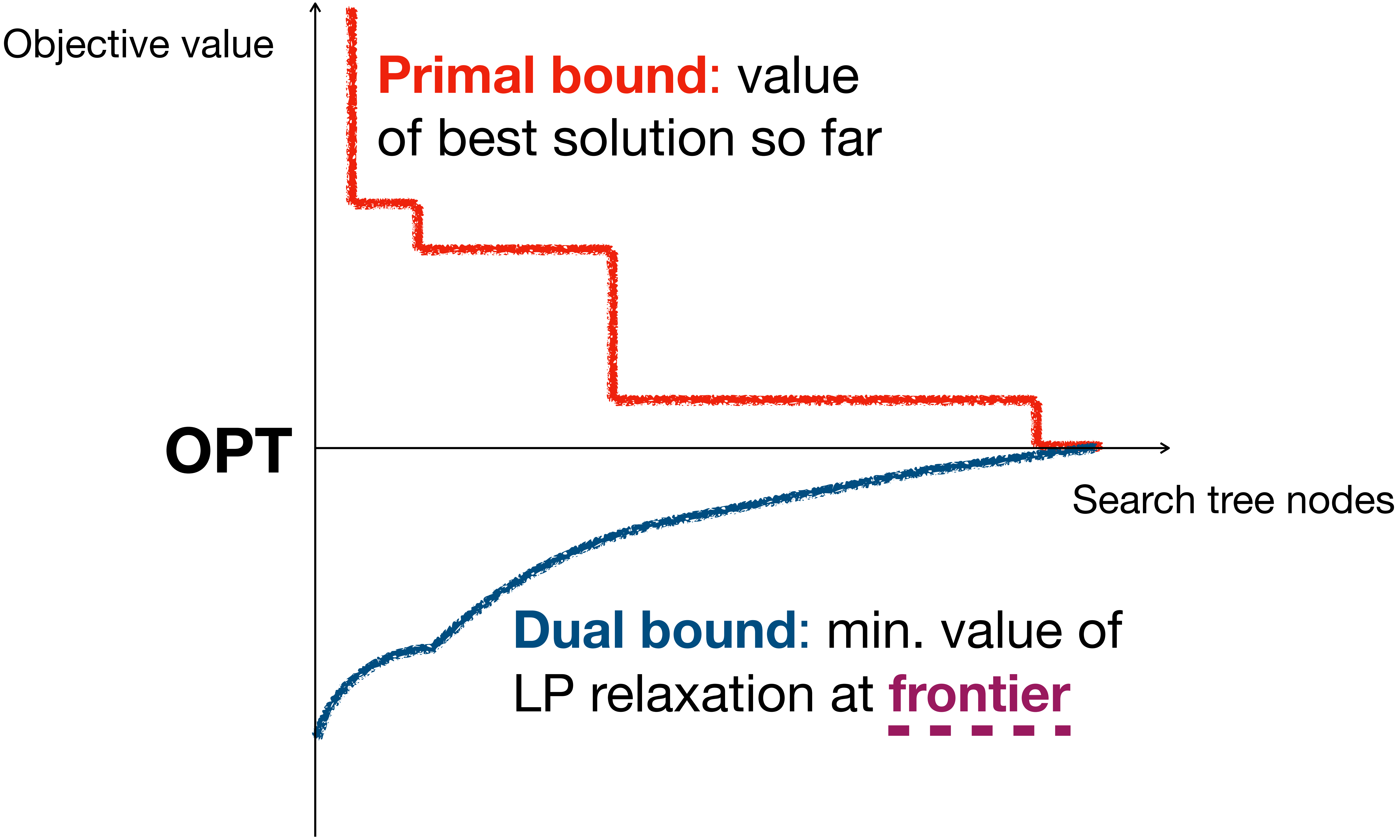
$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \{0,1\}^n$$



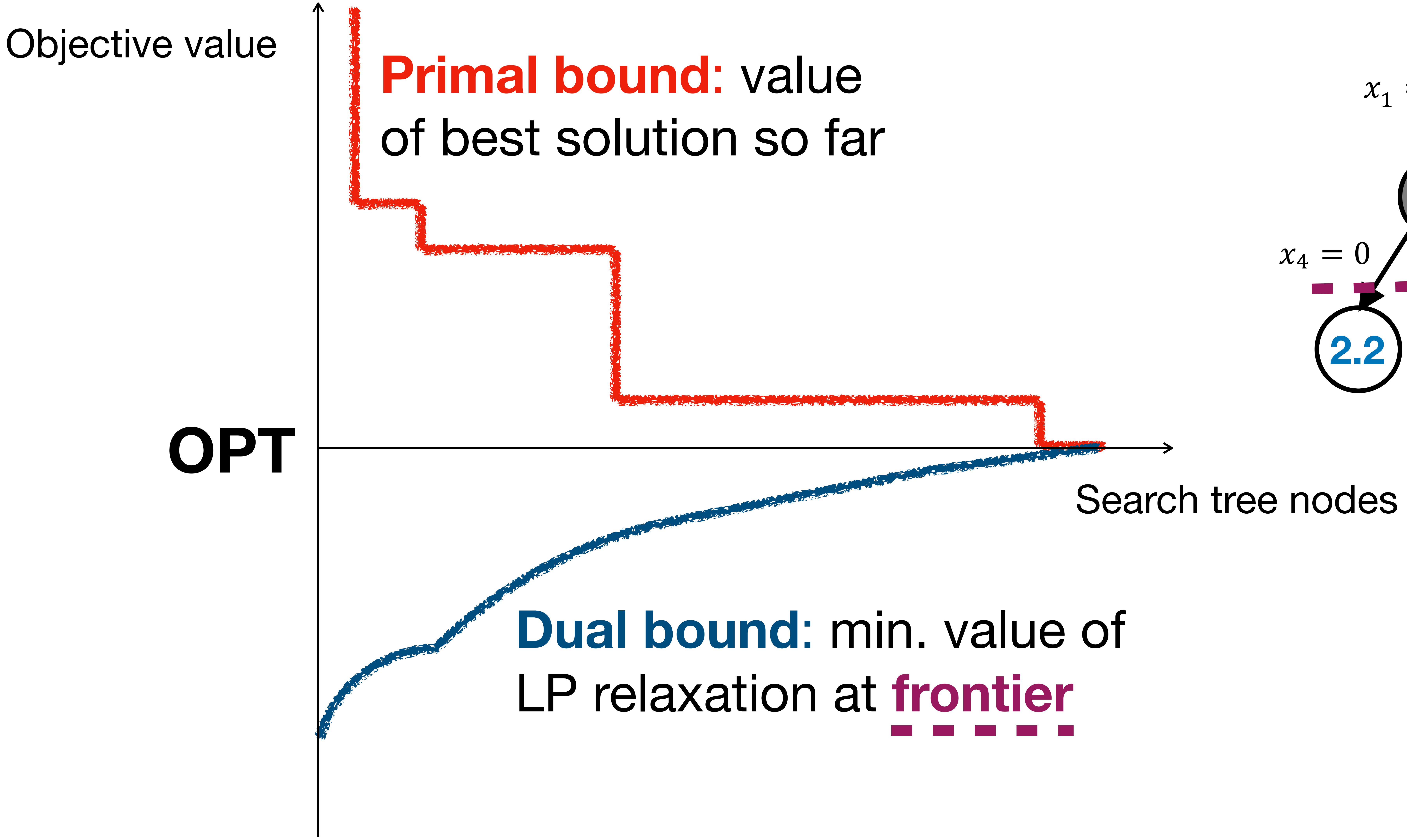
$$\min_x c^T x \textbf{ s.t. } Ax \le b, x \in \{0,1\}^n$$

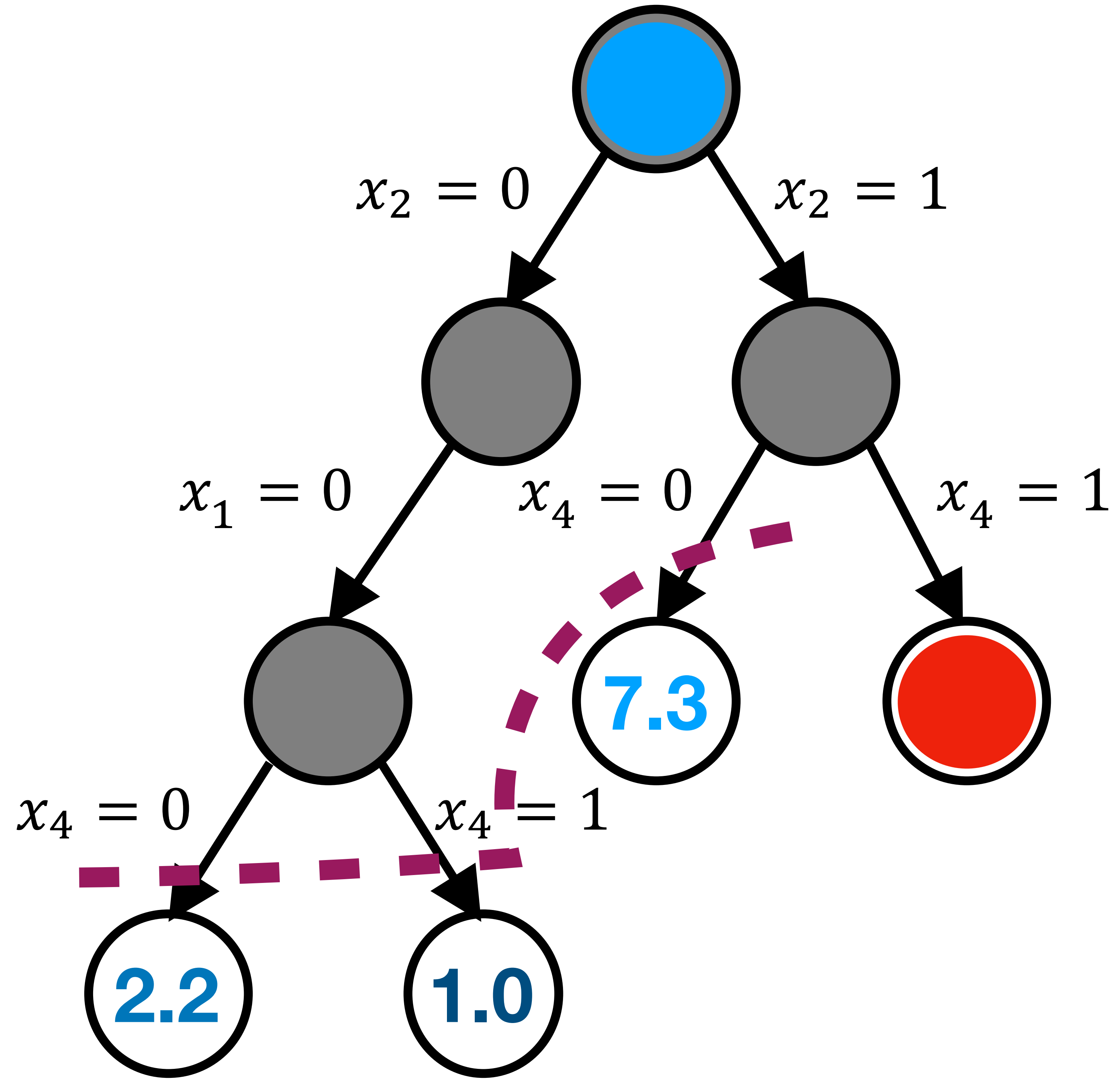


$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

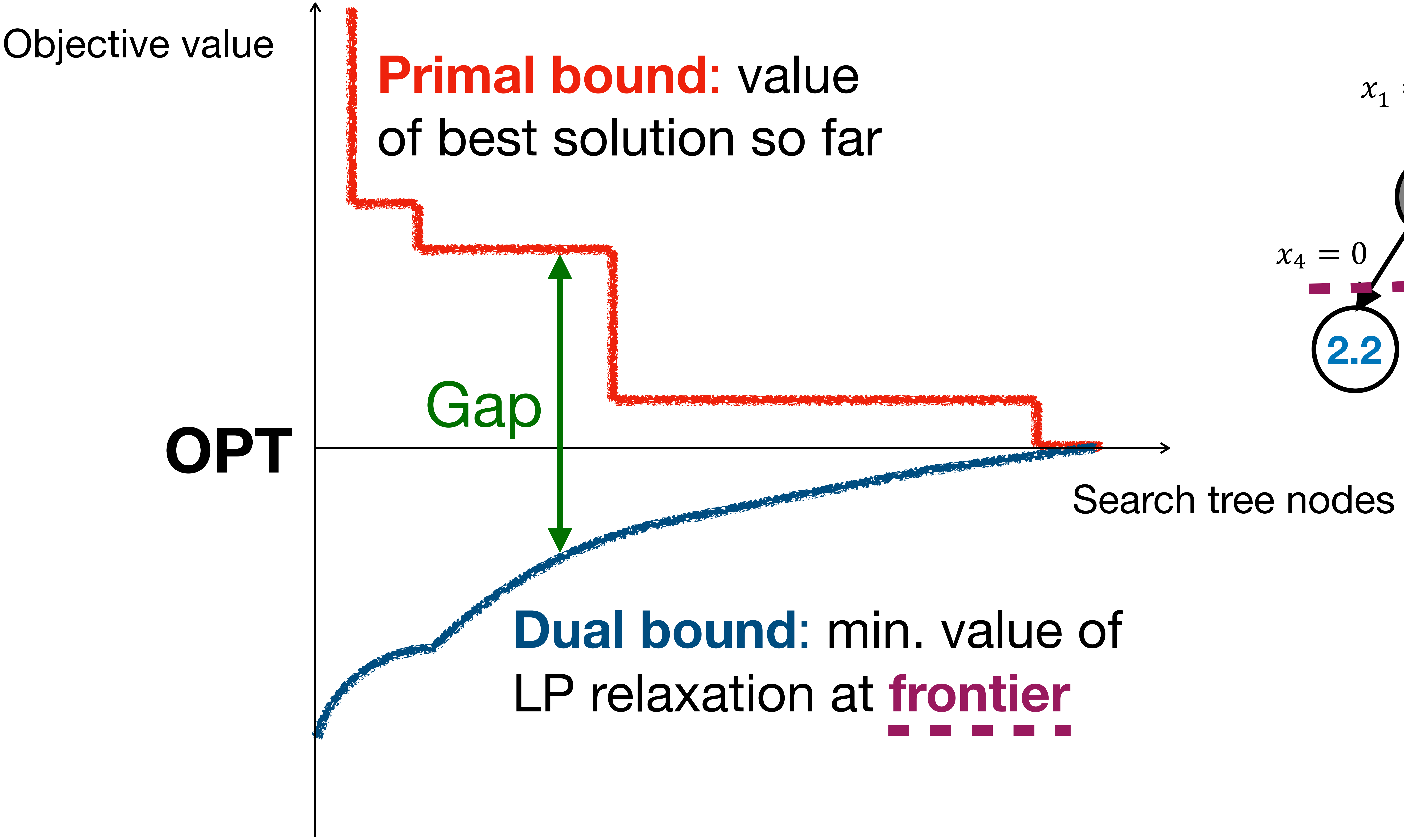


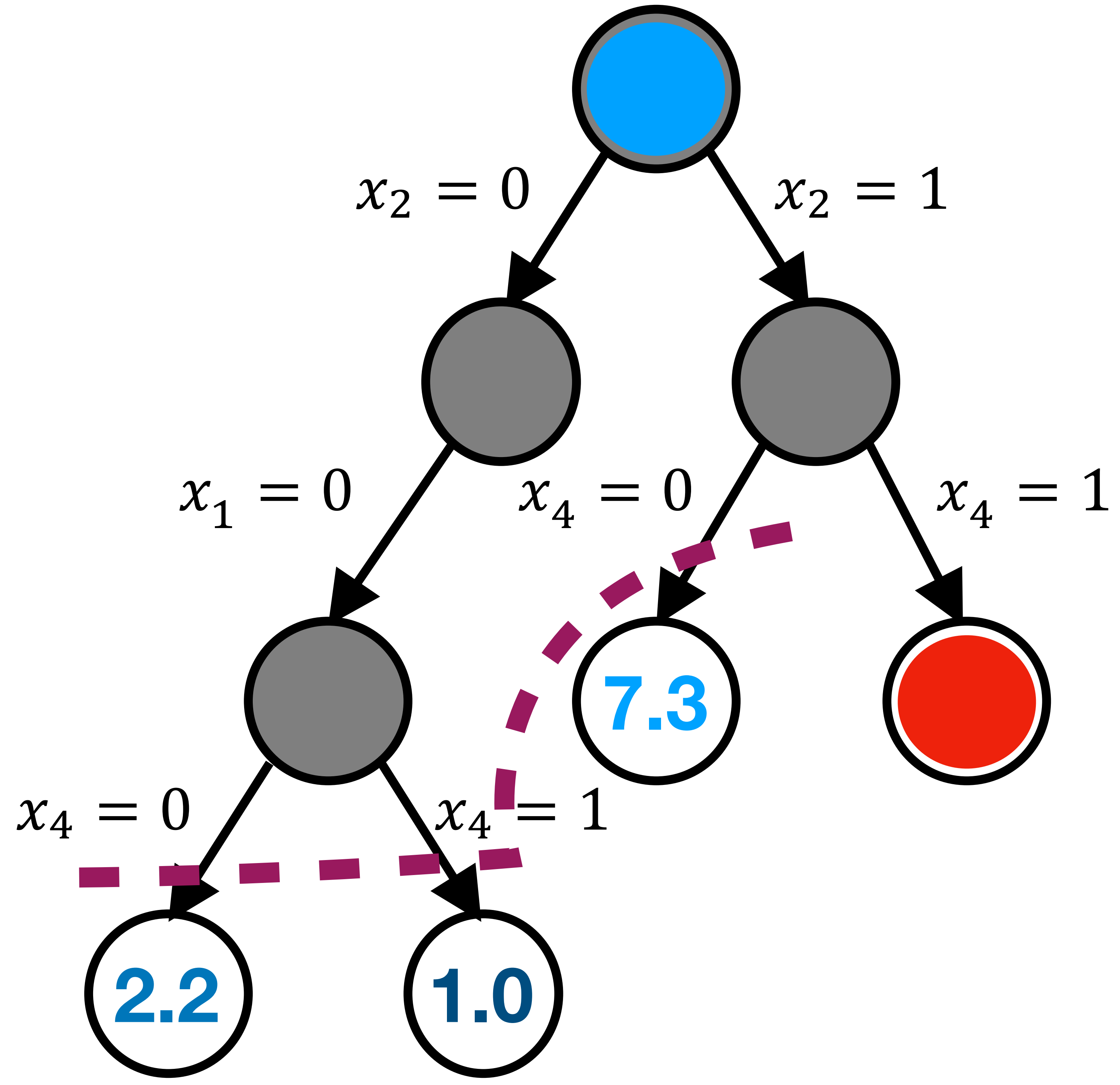
$$\min_x c^T x \text{ s.t. } Ax \le b, x \in \{0,1\}^n$$



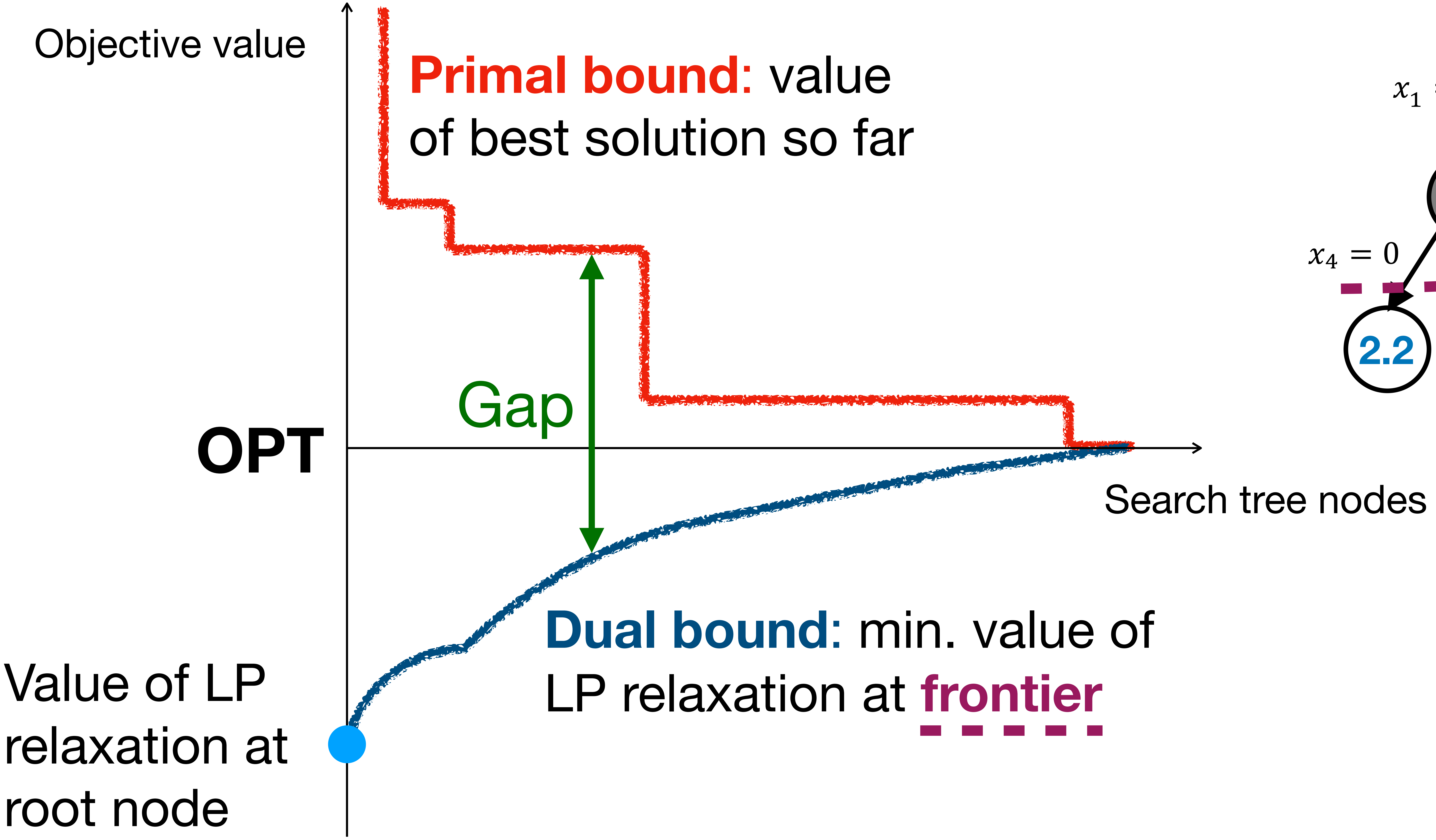


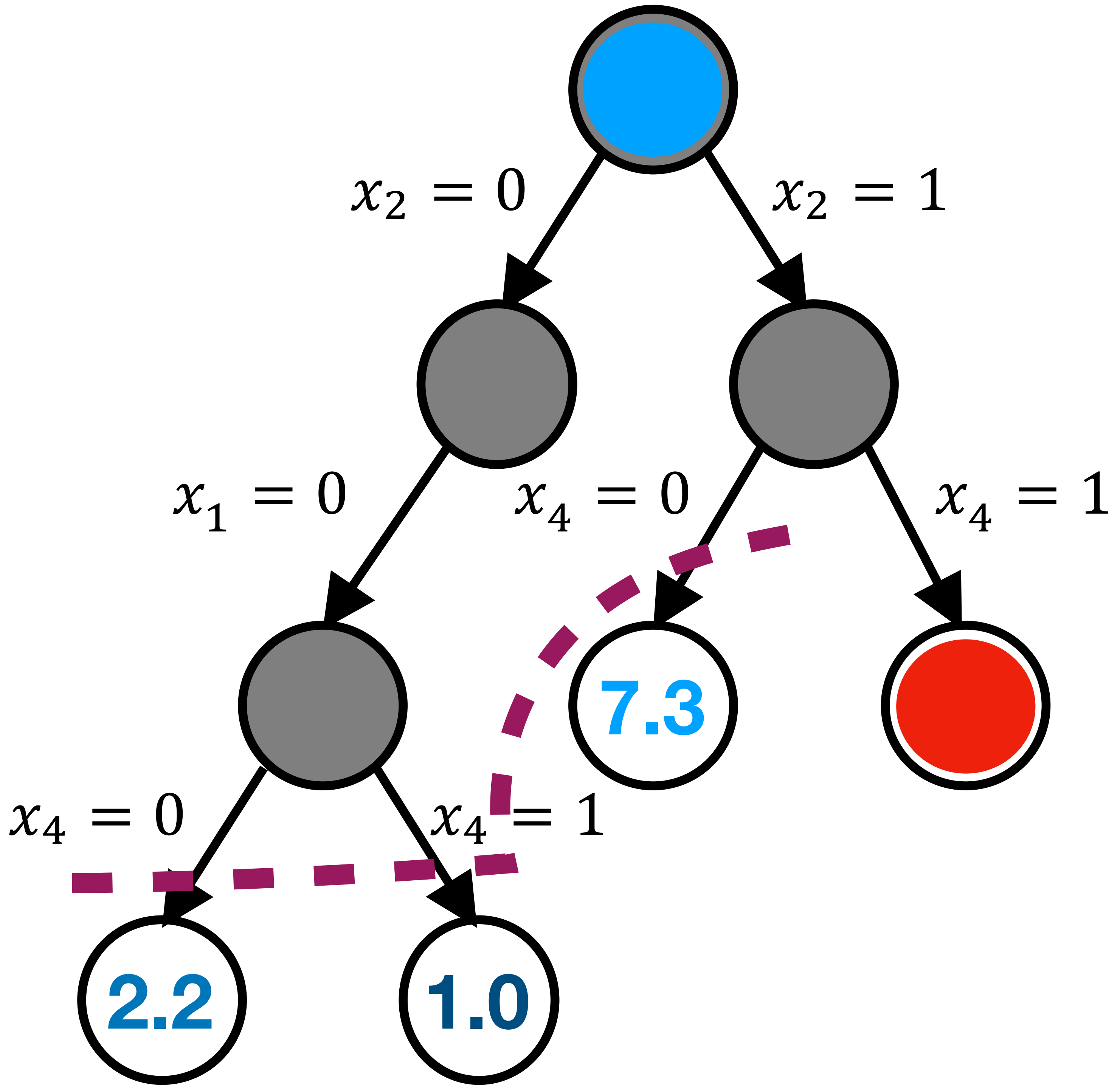
$$\min_x c^T x \textbf{ s.t. } Ax \le b, x \in \{0,1\}^n$$



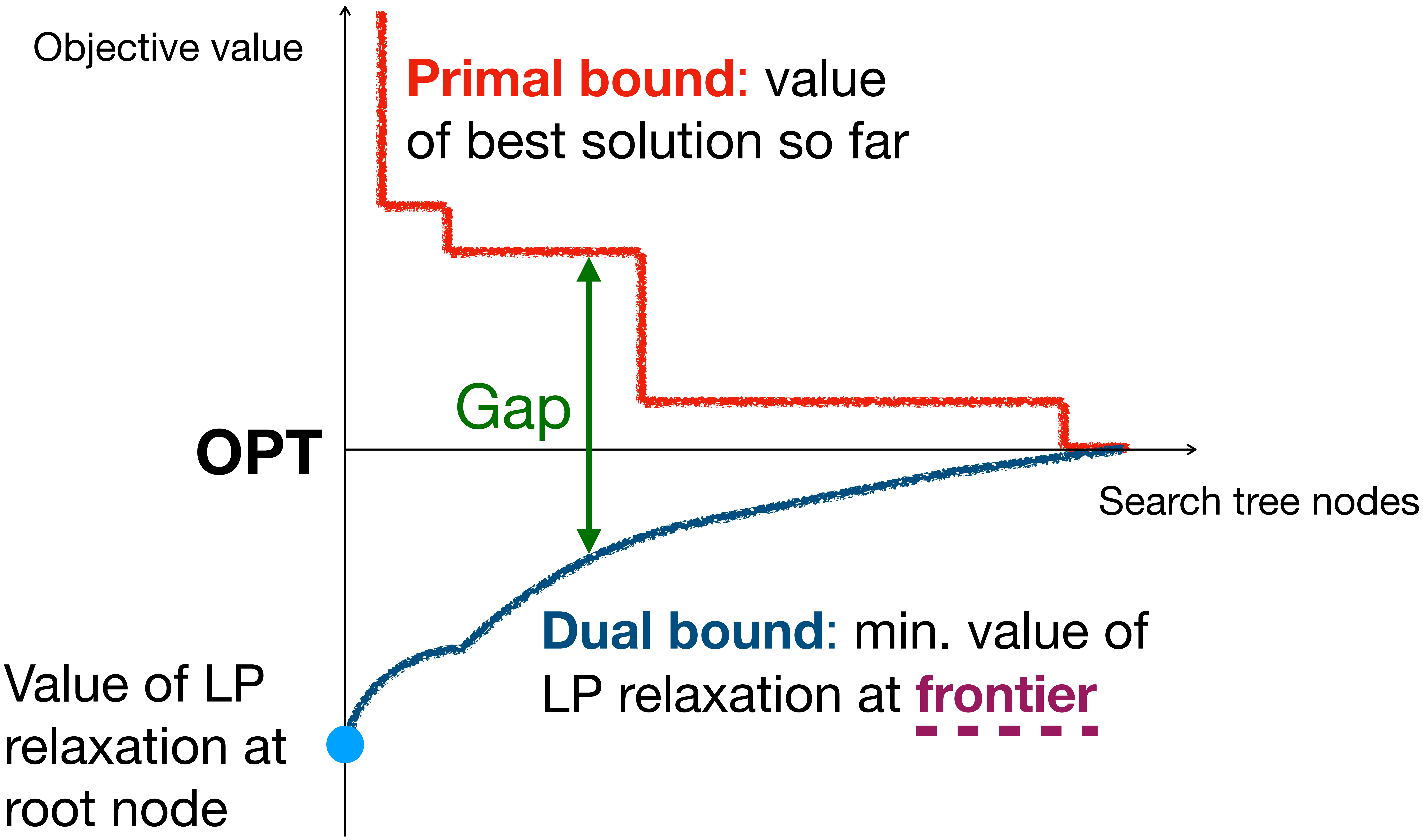


$$\min_x c^T x \textbf{ s.t. } Ax \le b, x \in \{0,1\}^n$$



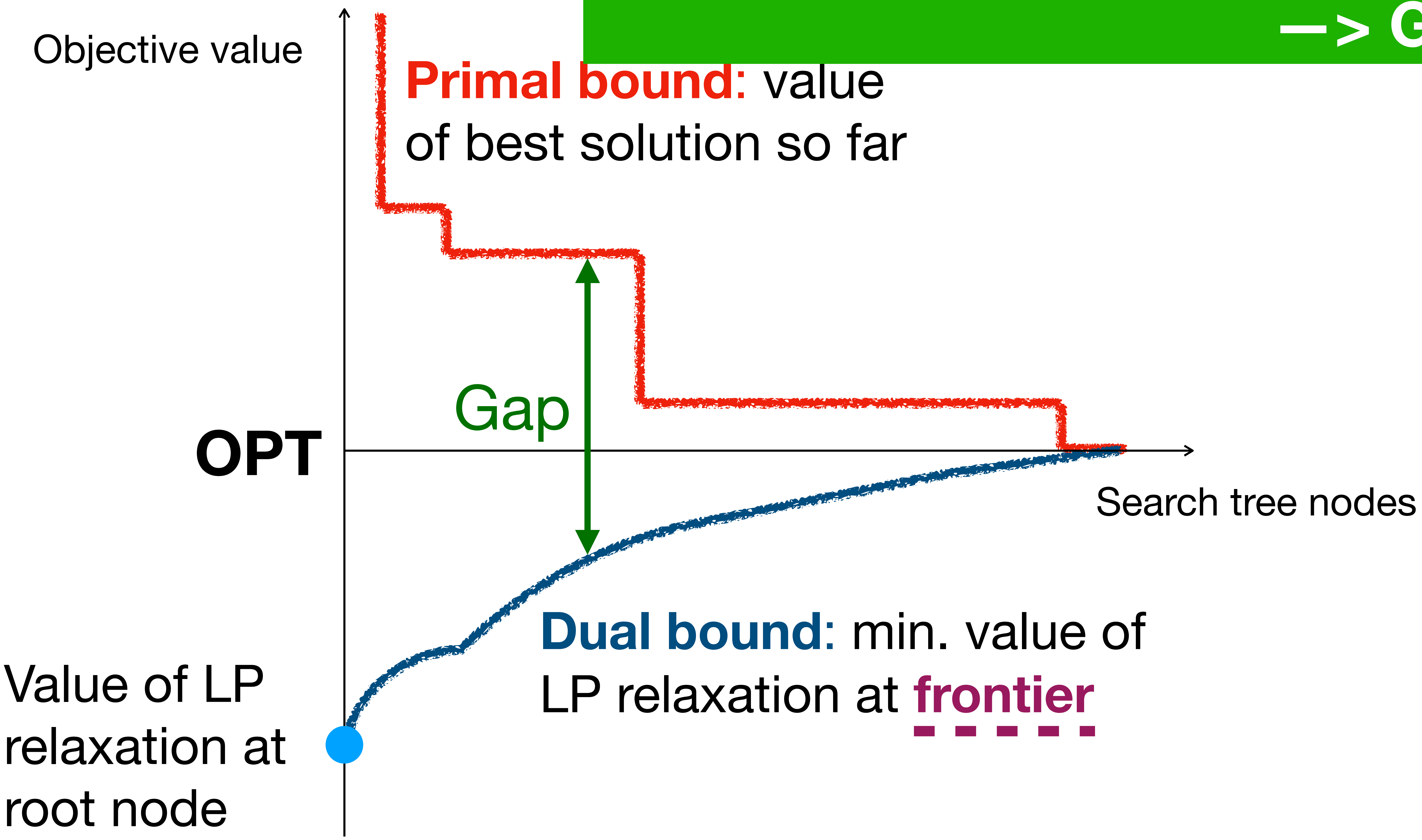


Heuristics matter!



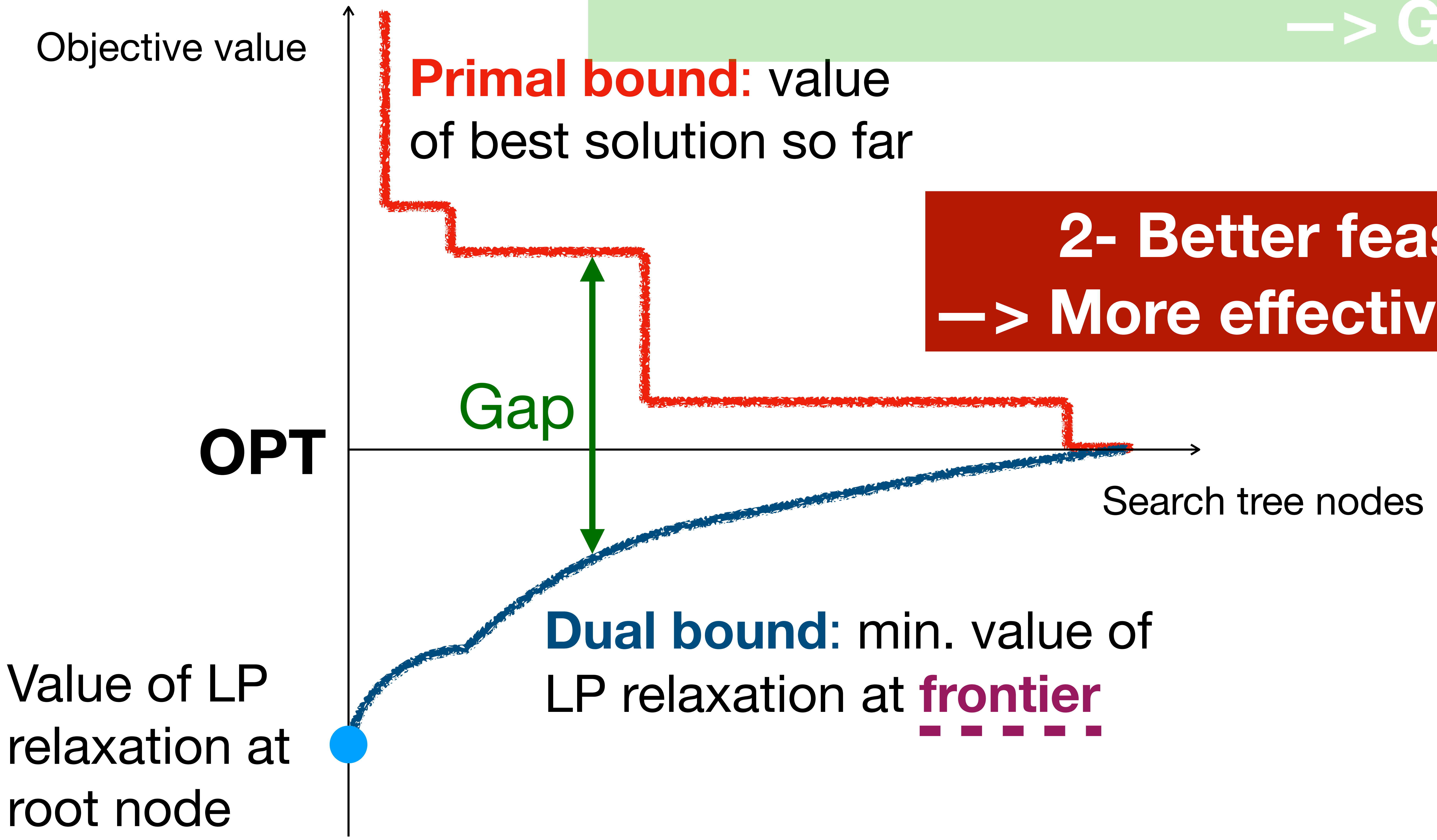
Heuristics matter!

1- Better primal bound —> More nodes pruned
—> Gap closed faster!



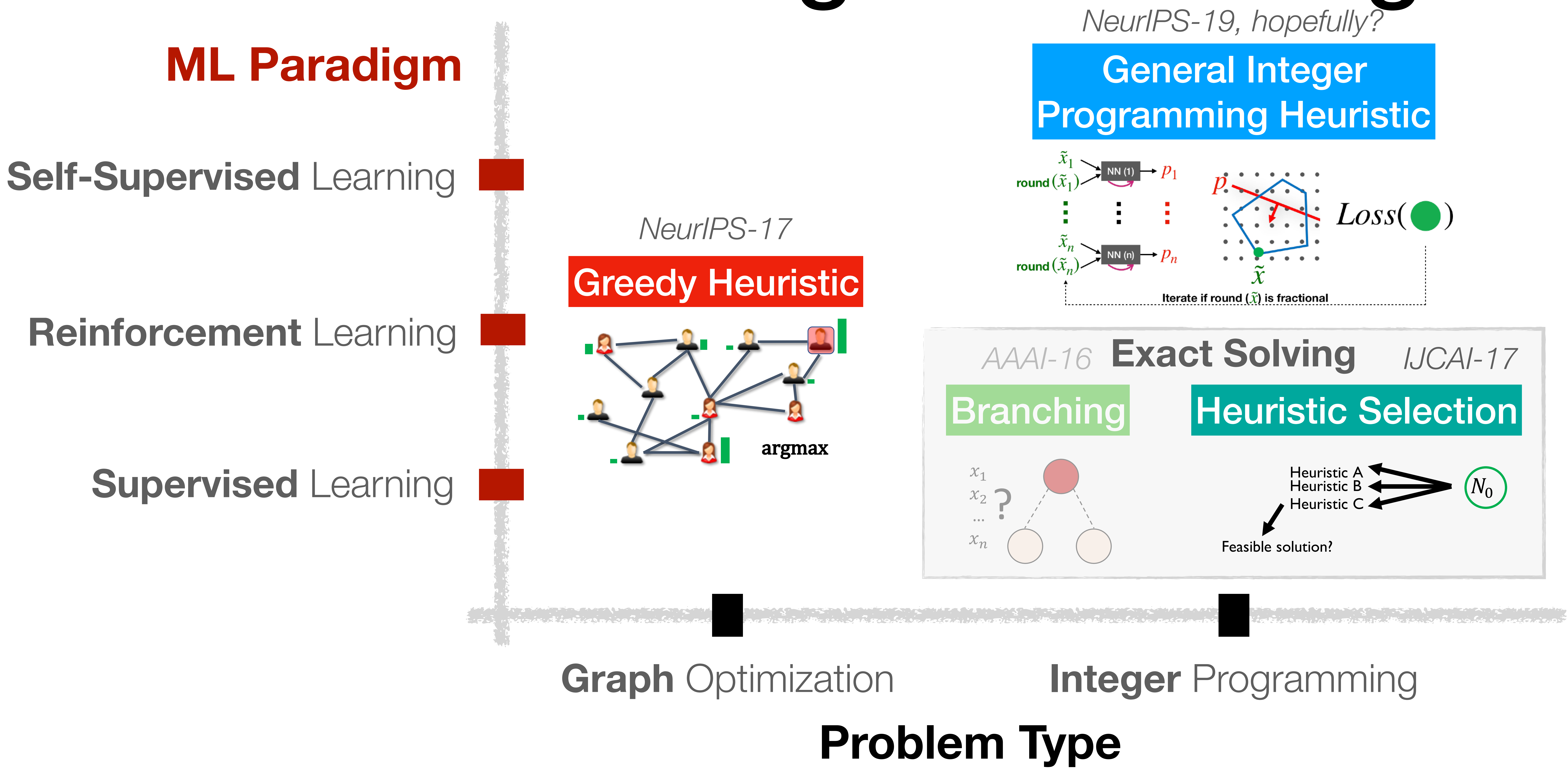
Heuristics matter!

1- Better primal bound —> More nodes pruned —> Gap closed faster!



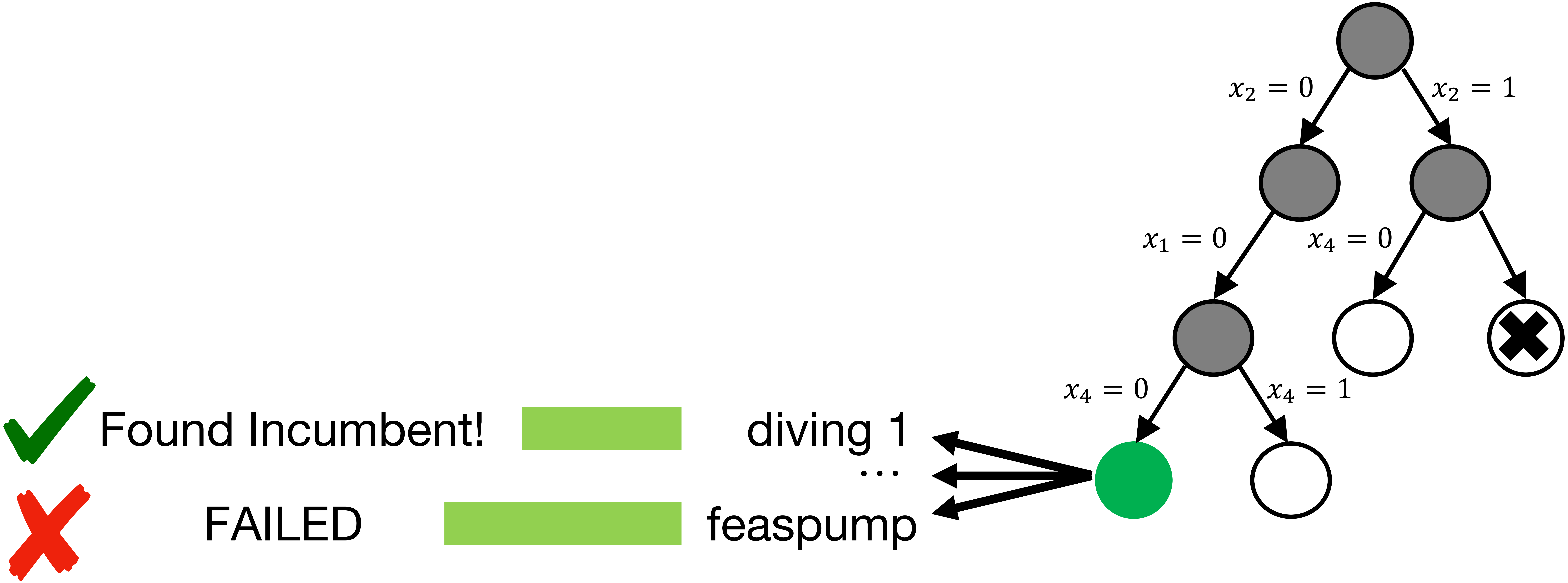
2- Better feasible solutions —> More effective decision-making

Data-Driven Algorithm Design



The Heuristic Selection Problem

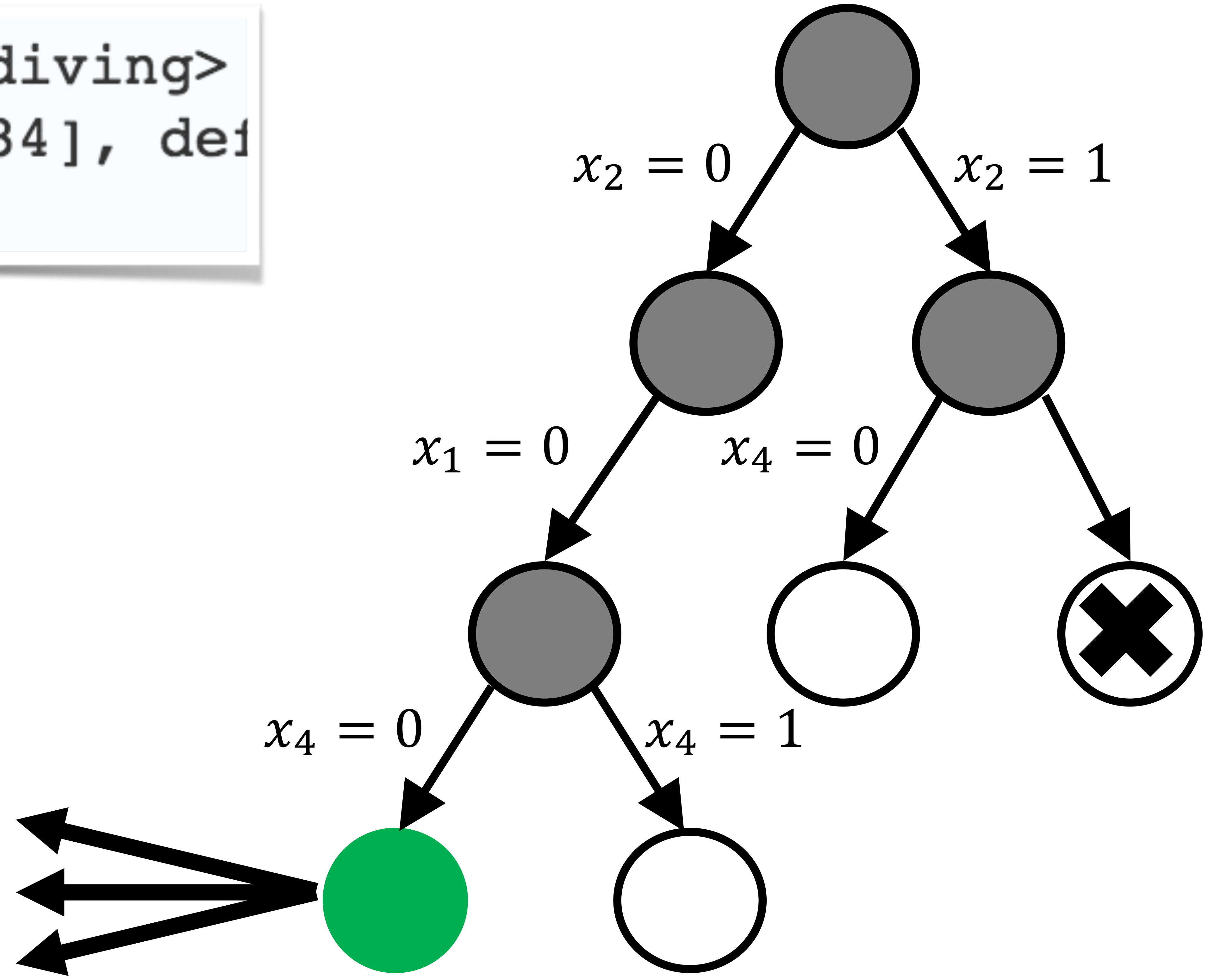
MIP solvers implement **many** primal **heuristics:** 54 in SCIP (2019)

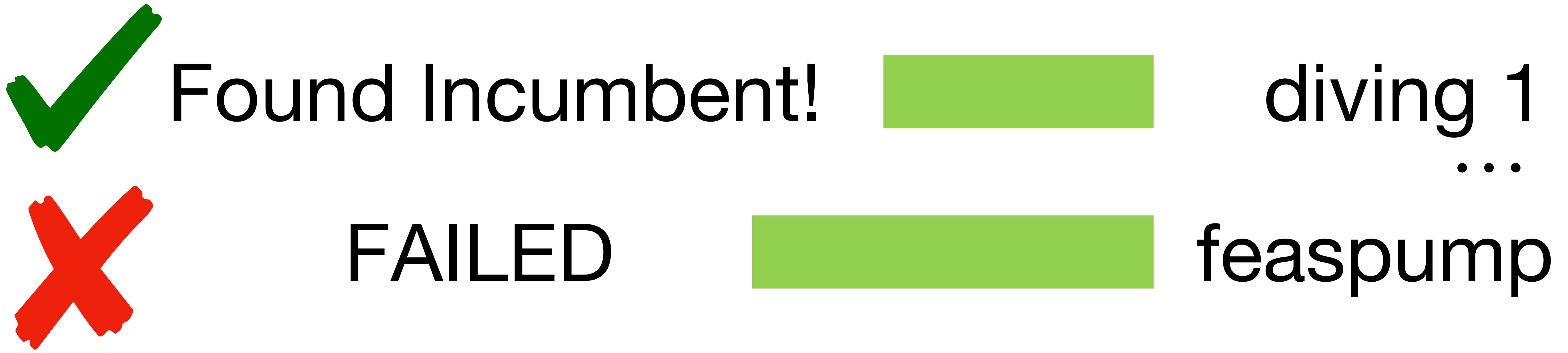


The Heuristic Selection Problem

MIP solvers implement **many** primal **heuristics**: 54 in SCIP (2019)

```
# frequency for calling primal heuristic <nlpdiving>
# [type: int, advanced: FALSE, range: [-1,65534], def
heuristics/nlpdiving/freq = 10
```



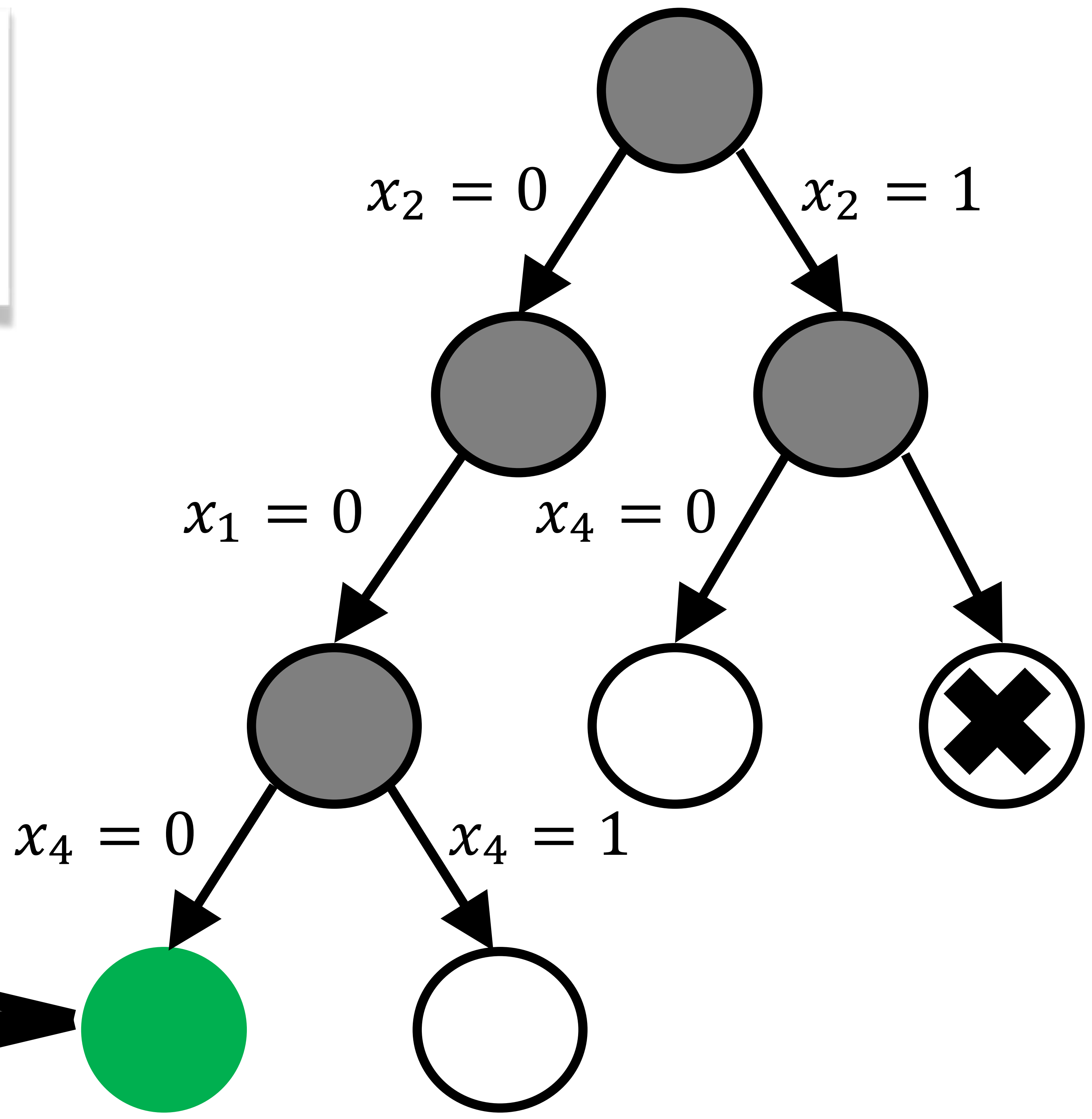


The Heuristic Selection Problem

MIP solvers implement **many** primal **heuristics**: 54 in SCIP (2019)

```
# frequency for calling primal heuristic <nlpdiving>
# [type: int, advanced: FALSE, range: [-1,65534], def
heuristics/nlpdiving/freq = 10
```

```
# frequency for calling primal heuristic <feaspump>
# [type: int, advanced: FALSE, range: [-1,65534], de
heuristics/feaspump/freq = 20
```



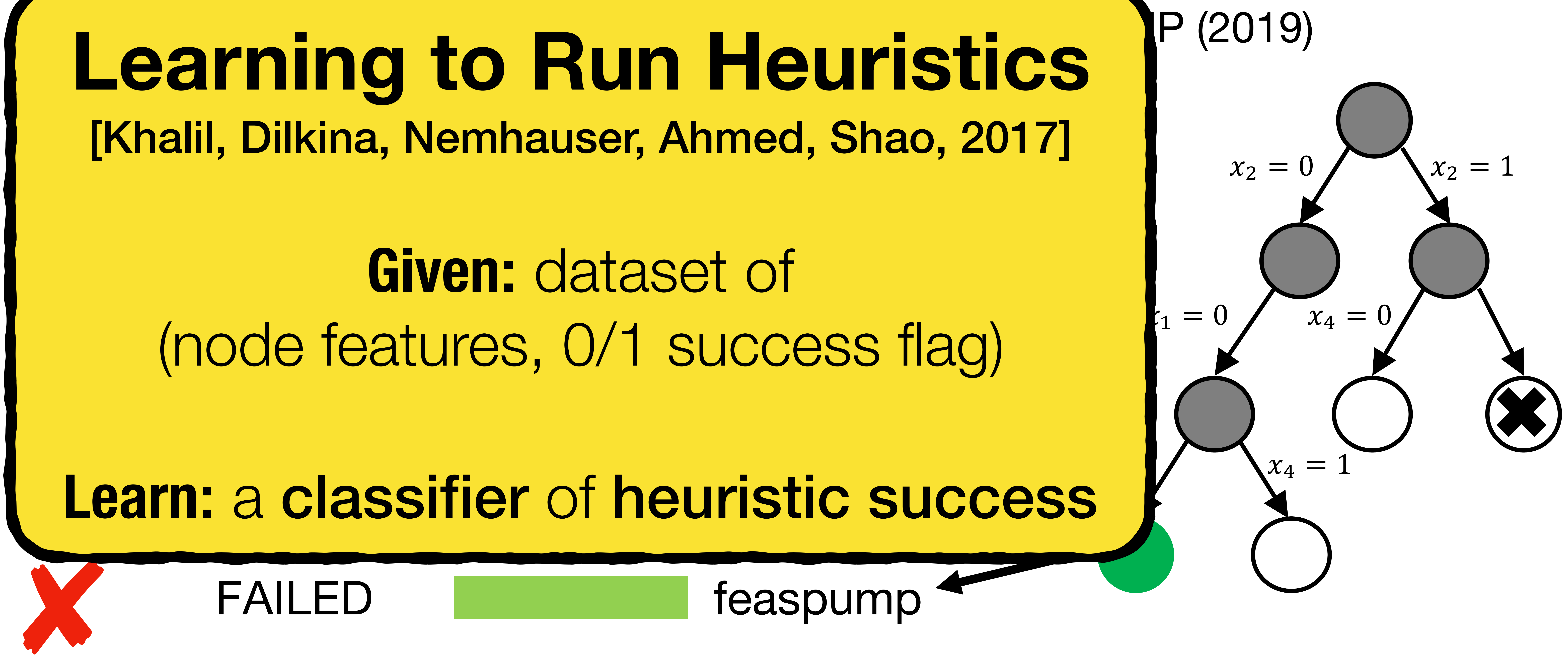
The Heuristic Selection Problem

Learning to Run Heuristics

[Khalil, Dilkina, Nemhauser, Ahmed, Shao, 2017]

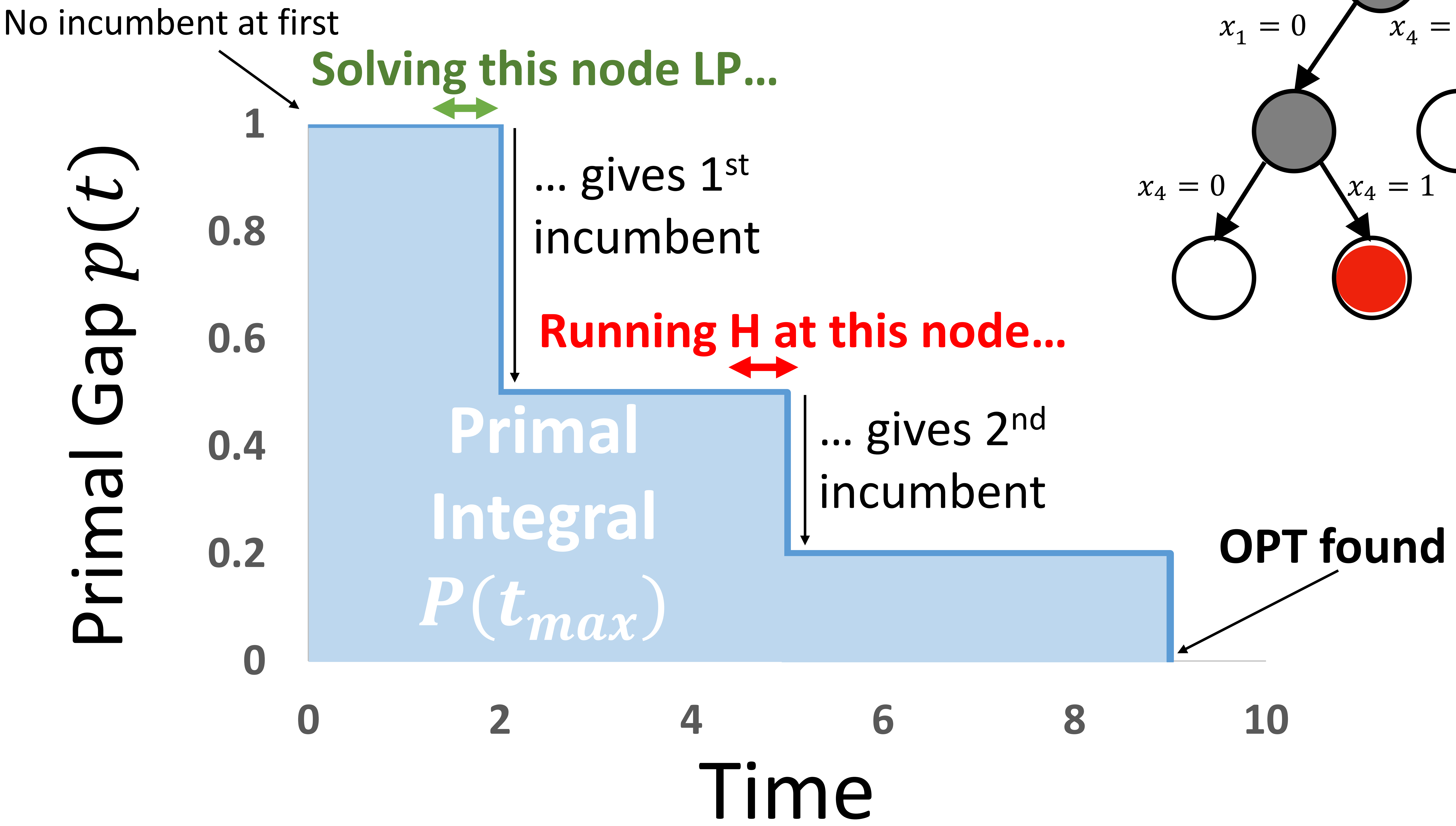
Given: dataset of (node features, 0/1 success flag)

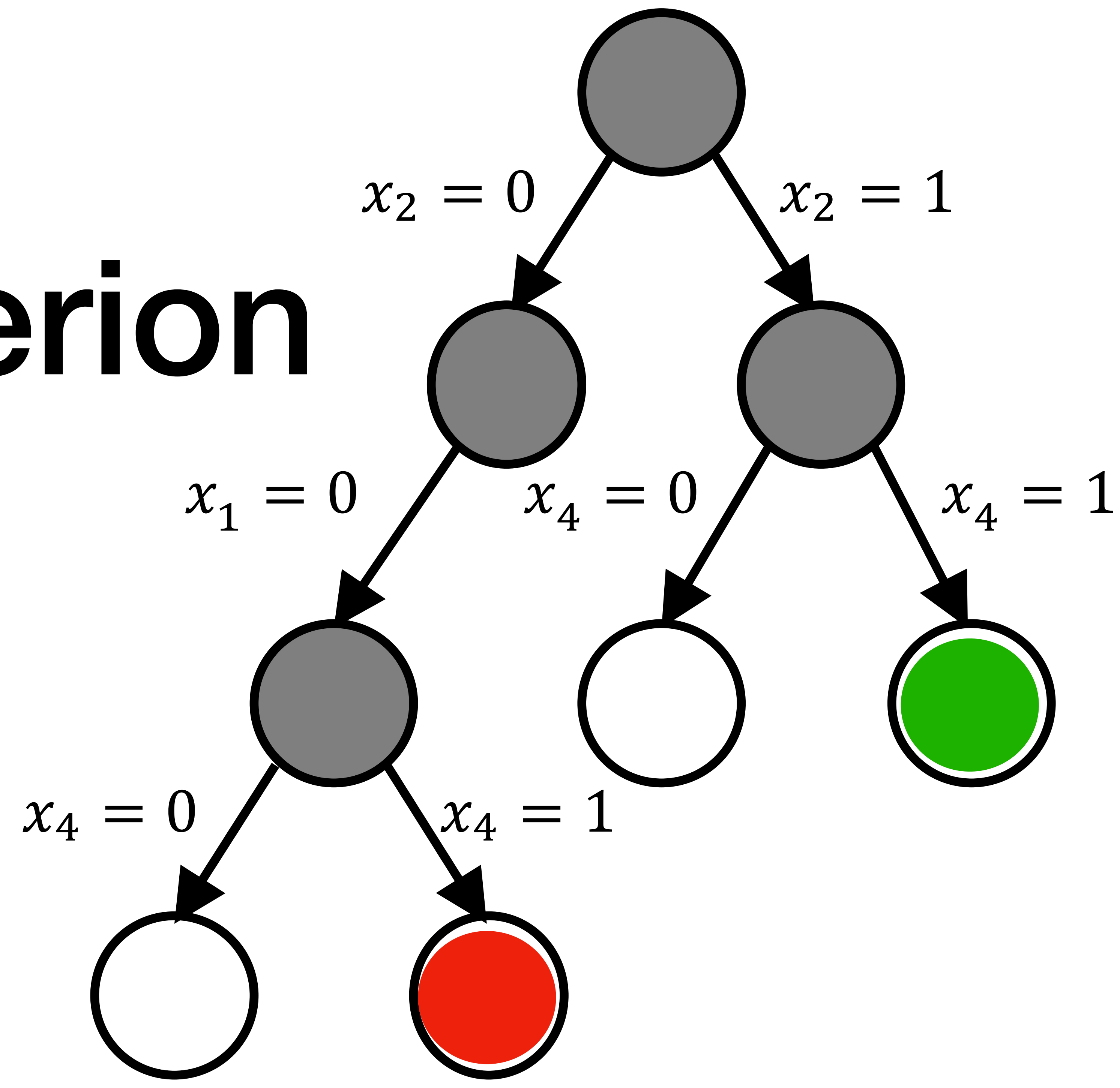
Learn: a **classifier** of **heuristic success**



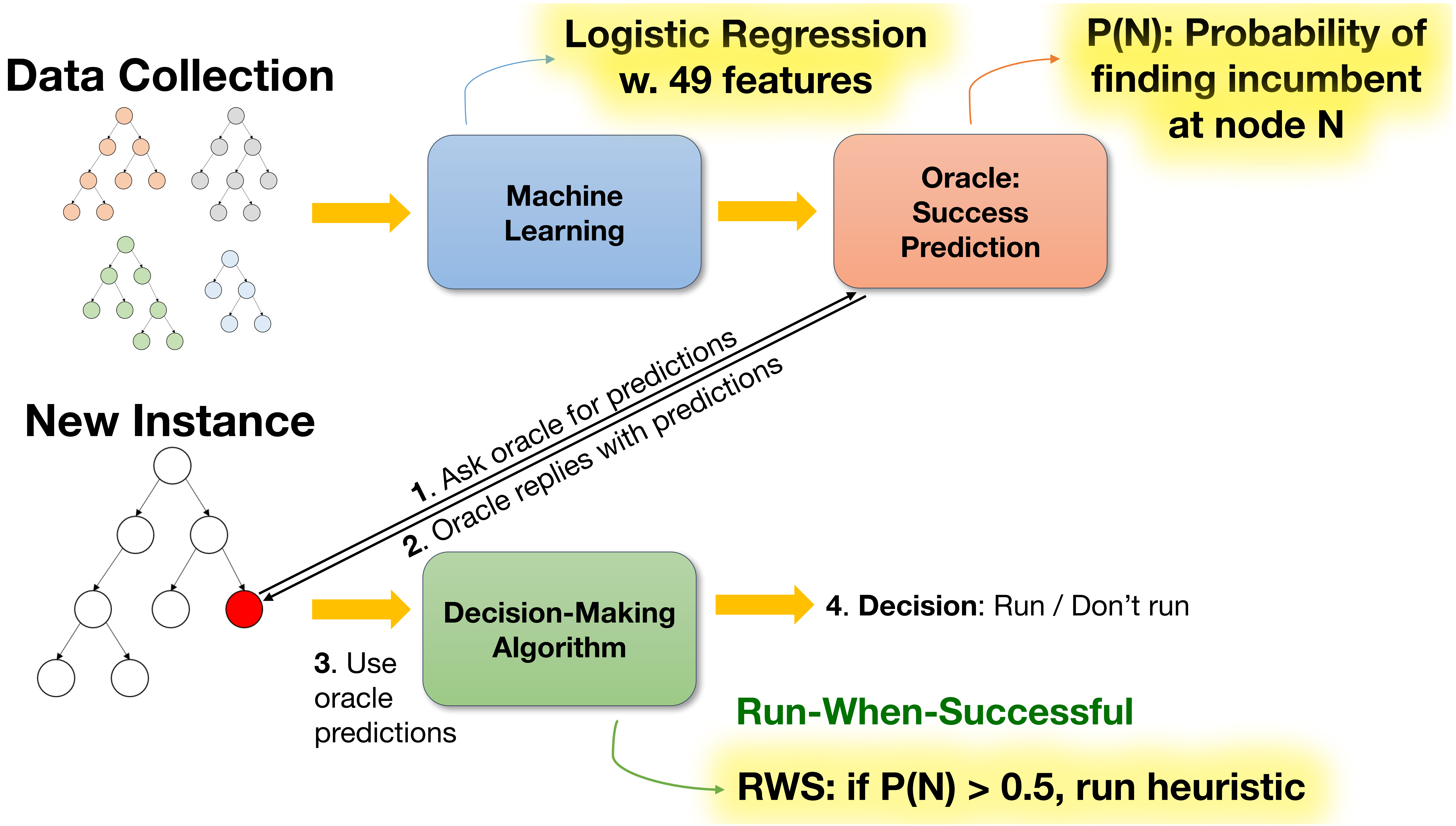
Primal Integral

A Good Performance Criterion

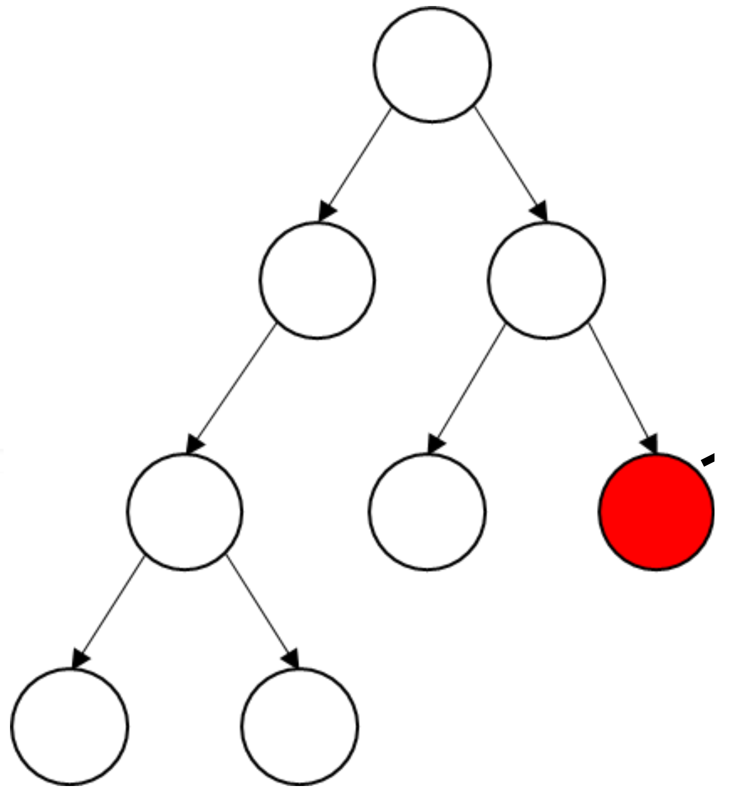




Learning to Run Heuristics



Feature Engineering

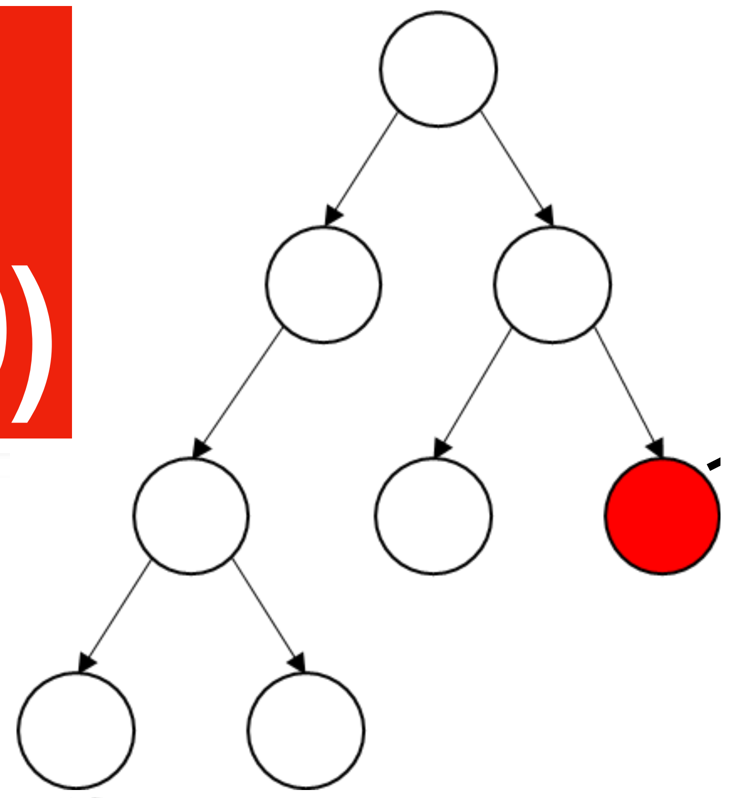


- **Global Features** (4):
  - optimality gap, root LP value / global lower (upper) bound
- **Depth Features** (2):
  - node depth / max. depth in tree (max. possible depth)
- **Node LP Features** (8):
  - sum of variables' LP sol. fractionalities / #fractional variables
  - num. of fractional variables / #integer variables
  - num. variables roundable up (down) / #integer variables
- **Scoring Features for Fractional Variables** (35):
  - number of up (down) locks
  - normalized objective coefficient
  - pseudocost score

  Five statistics (mean, min., max., median, standard deviation) for each metric over fractional variables in LP solution.

Feature Engi

Binary Label
found incumbent (1), o.w. (0)



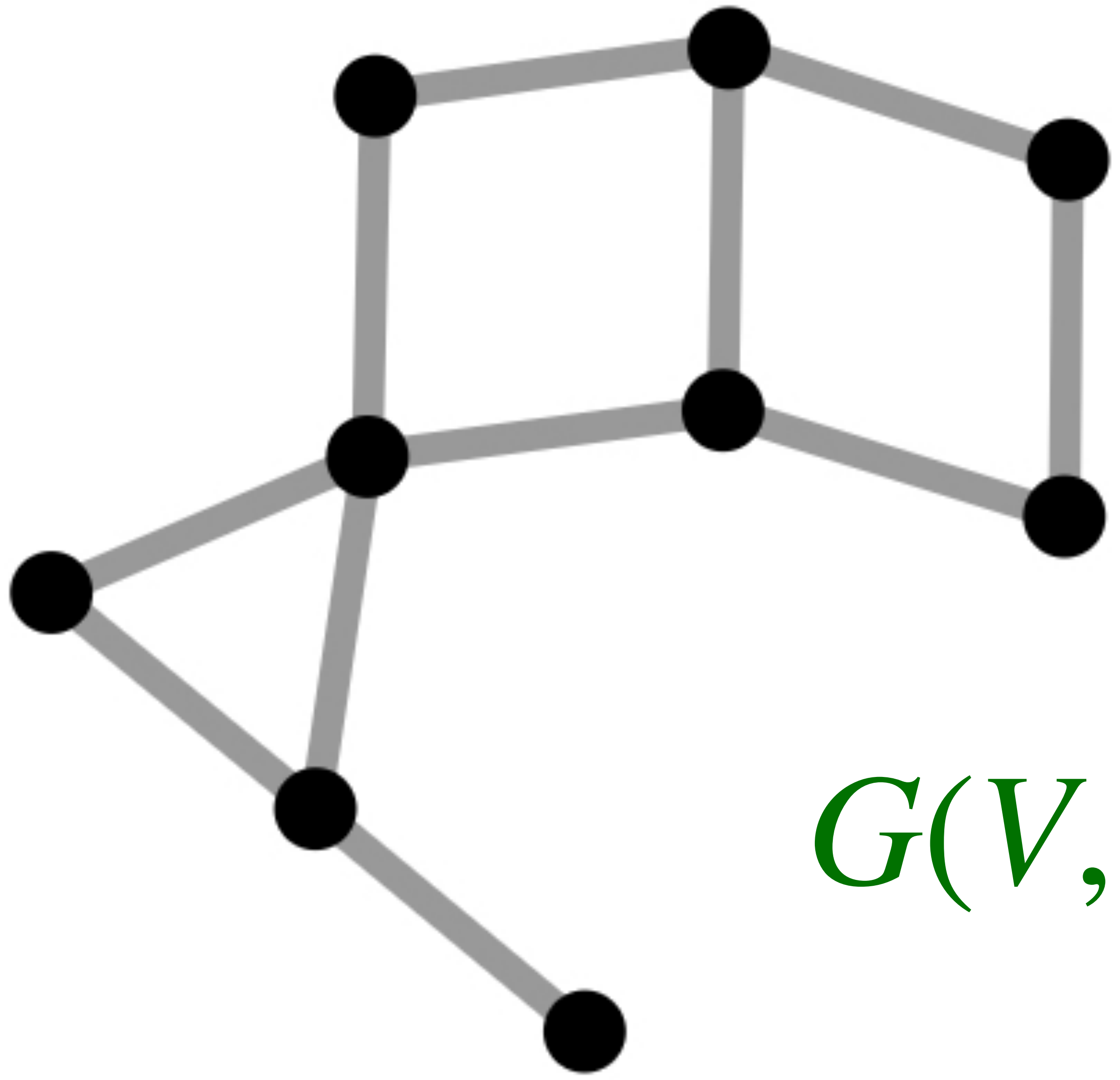
- **Global Features** (4):
  - optimality gap, root LP value / global lower (upper) bound
- **Depth Features** (2):
  - node depth / max. depth in tree (max. possible depth)
- **Node LP Features** (8):
  - sum of variables' LP sol. fractionalities / #fractional variables
  - num. of fractional variables / #integer variables
  - num. variables roundable up (down) / #integer variables
- **Scoring Features for Fractional Variables** (35):
  - number of up (down) locks
  - normalized objective coefficient
  - pseudocost score

  Five statistics (mean, min., max., median, standard deviation) for each metric over fractional variables in LP solution.

Forest Harvesting

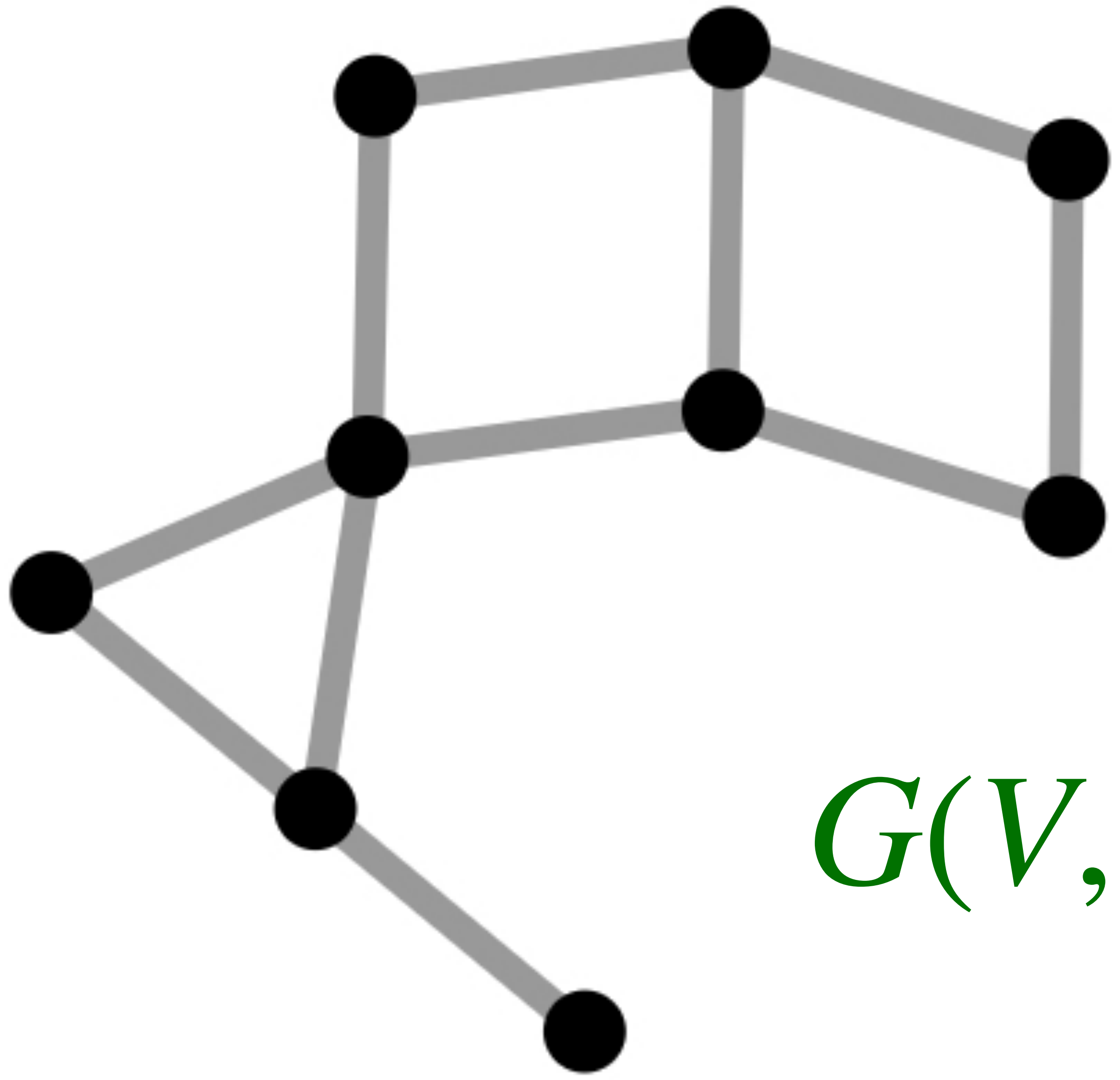
Forest Harvesting

Forest Harvesting



$G(V, E)$

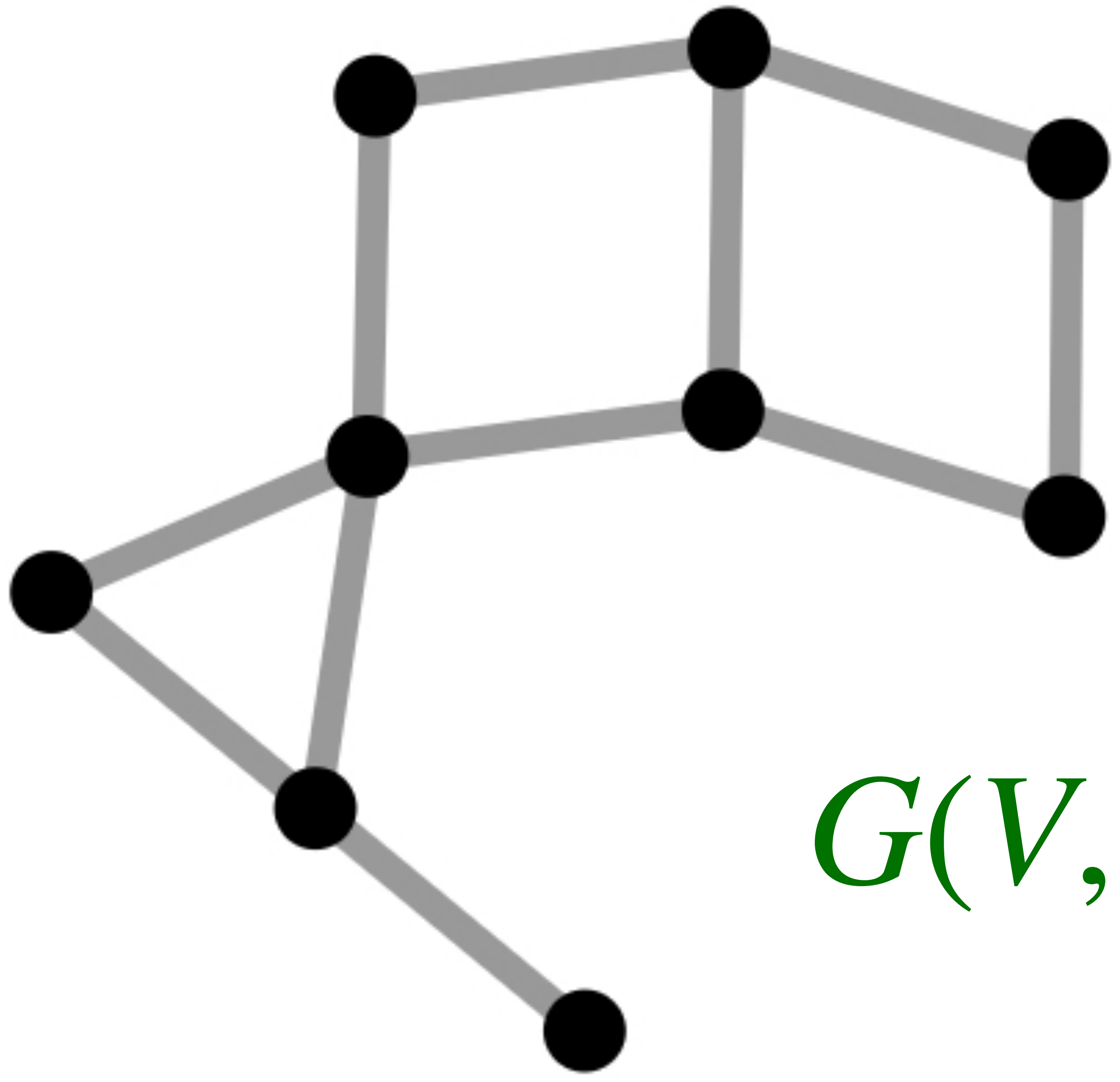
Forest Harvesting



$G(V, E)$

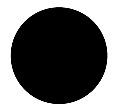
Forest Harvesting

Goal: Harvest subset of parcels to maximize **revenue**; pay **cost** for harvesting adjacent parcels



$G(V, E)$

Forest Harvesting

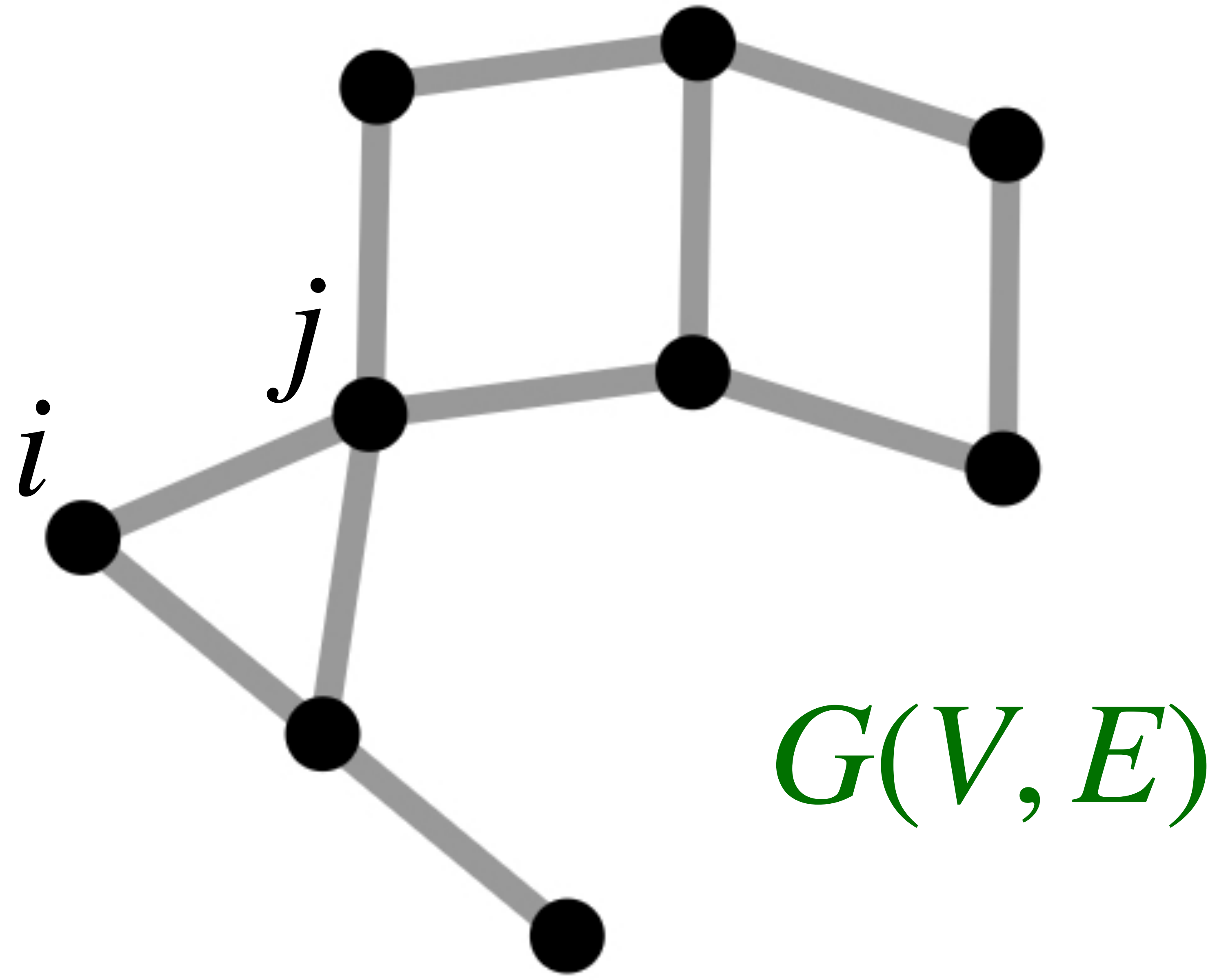


Goal: Harvest subset of parcels to maximize **revenue**; pay **cost** for harvesting adjacent parcels

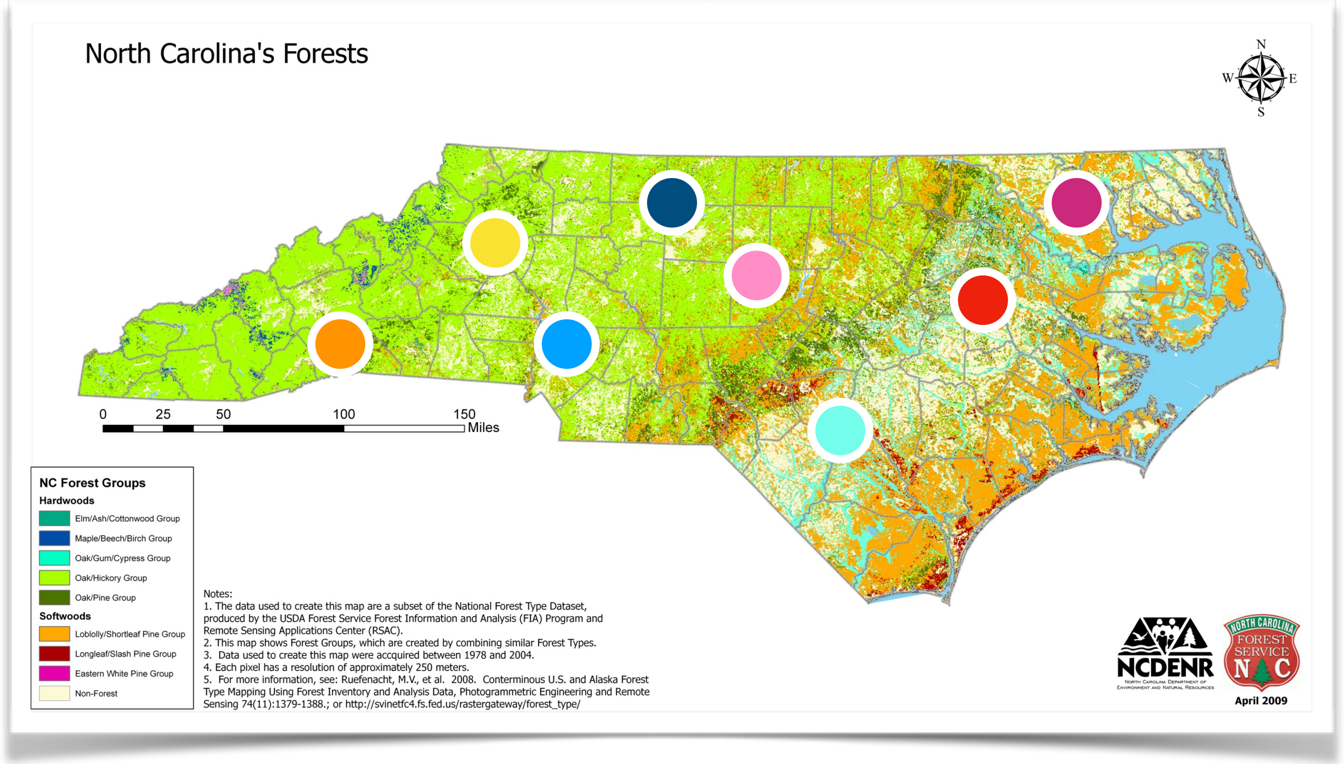
maximize $\sum_{i \in V} r_i x_i - \sum_{(i,j) \in E} c_{ij} y_{ij}$

subject to $x_i + x_j - y_{ij} \leq 1$

$$x \in \{0,1\}^n,\ y \in \{0,1\}^m$$



Heuristic Selection in Practice

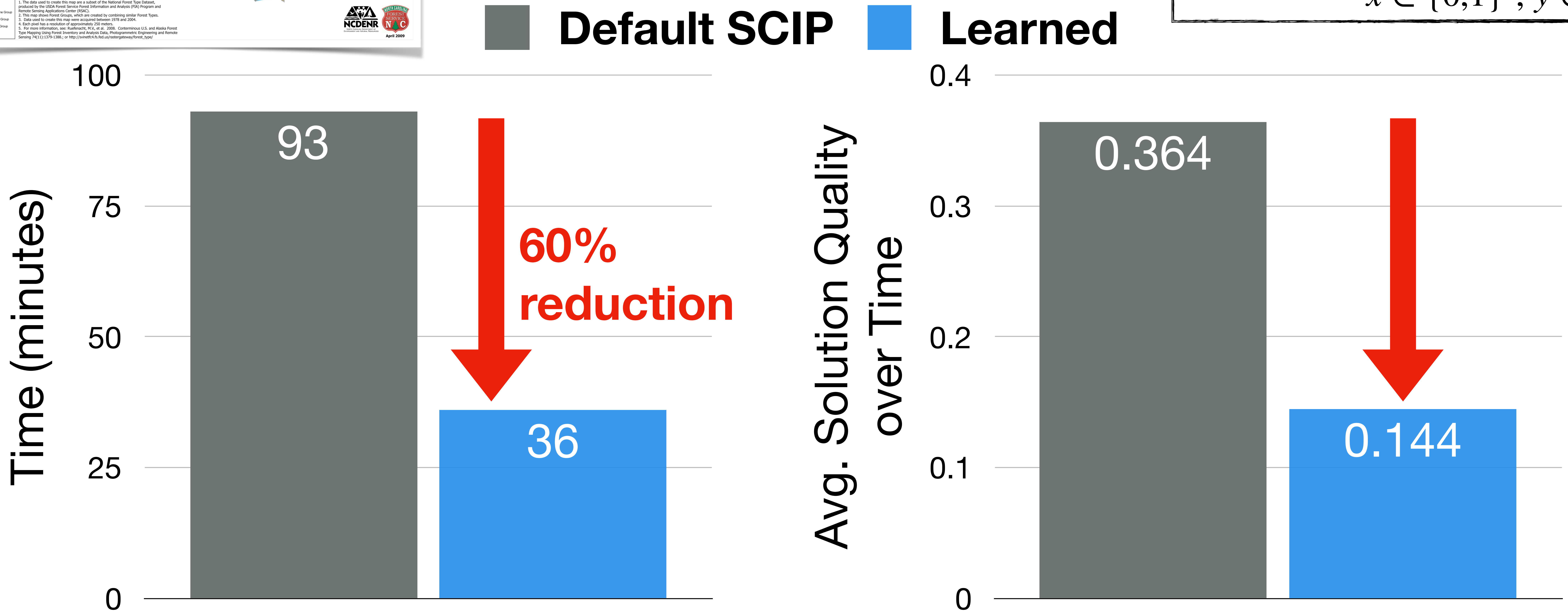


Generalized Independent Set

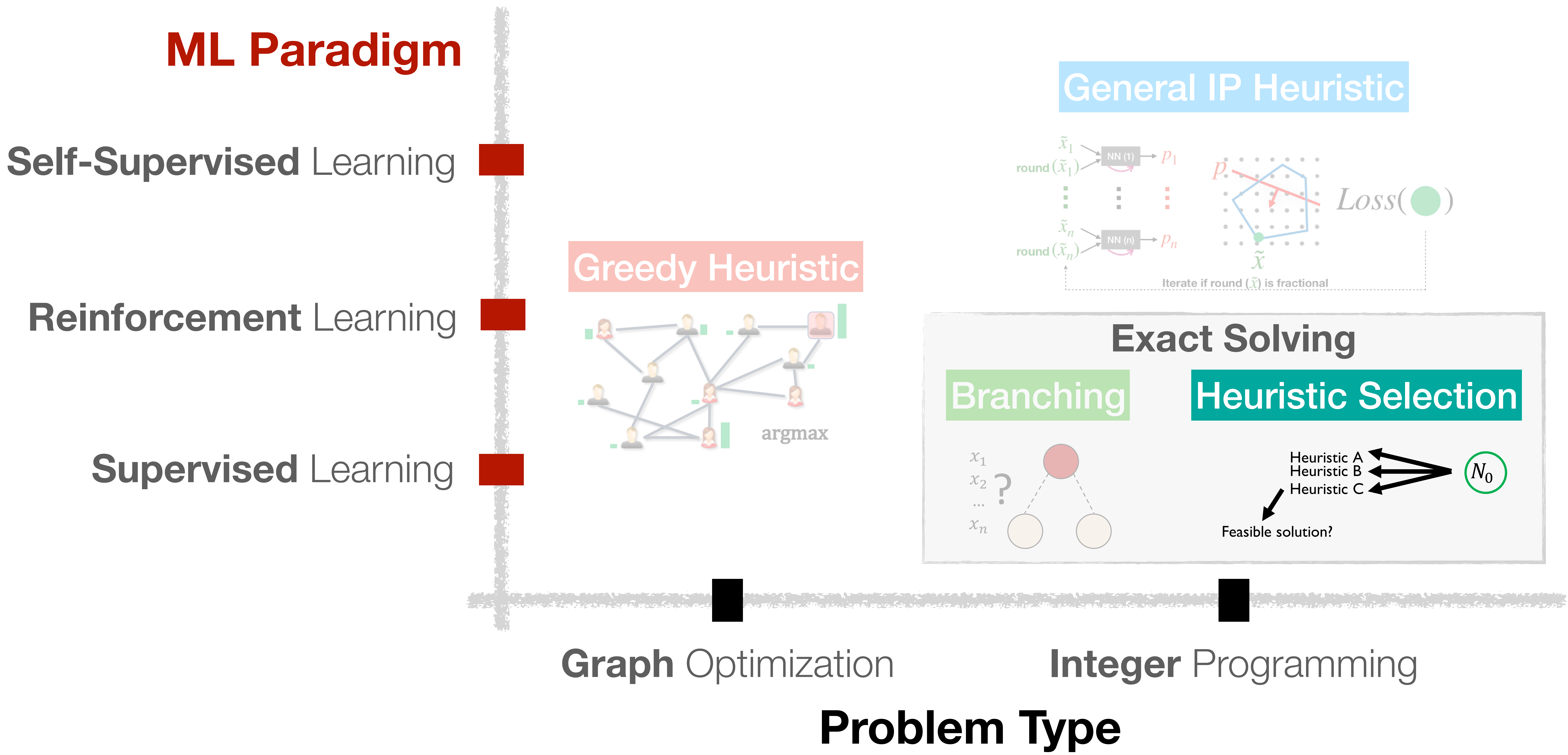
$$\textbf{maximize} \sum_{i \in V} r_i x_i - \sum_{(i,j) \in E} c_{ij} y_{ij}$$

$$\textbf{subject to } x_i + x_j - y_{ij} \leq 1$$

$$x \in \{0,1\}^n,\ y \in \{0,1\}^m$$



Time to Best Solution Primal Integral


ML Paradigm
Self-Supervised Learning
Reinforcement Learning
Supervised Learning
General IP Heuristic
Loss
Iterate if round ($\tilde{x}$) is fractional
Greedy Heuristic
argmax
Exact Solving
Branching
Heuristic Selection
Heuristic A
Heuristic B
Heuristic C
N_0
Feasible solution?
Graph Optimization
Integer Programming
Problem Type

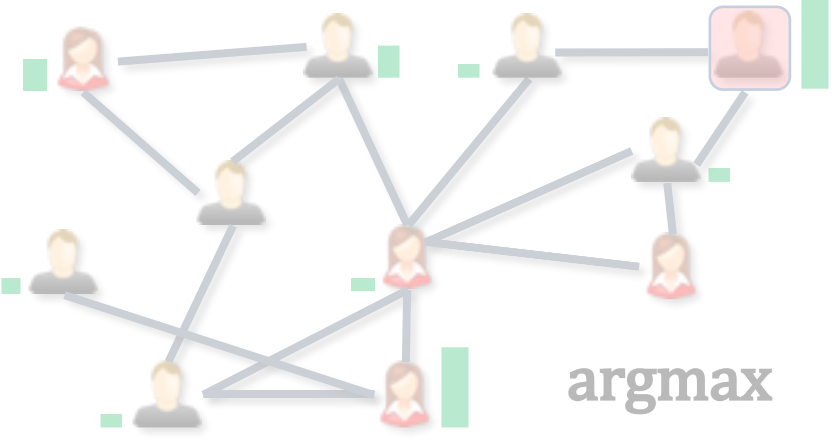
Takeaways

- **First ML framework for heuristic selection in B&B**
- **Dynamic,** node-dependent **decision-making**
- **Forest Harvesting: 60% reduction** in Primal Integral
- Even on the heterogeneous **MIPLIB2010** Benchmark**:**
  6% reduction in Primal Integral

Self-S

Reinforcement Learning

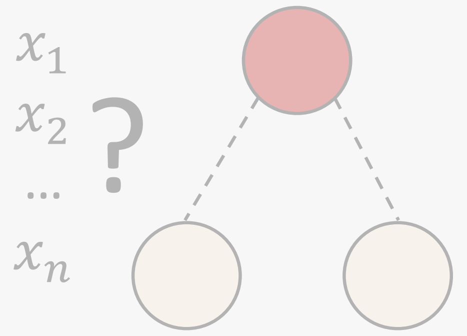
Supervised Learning

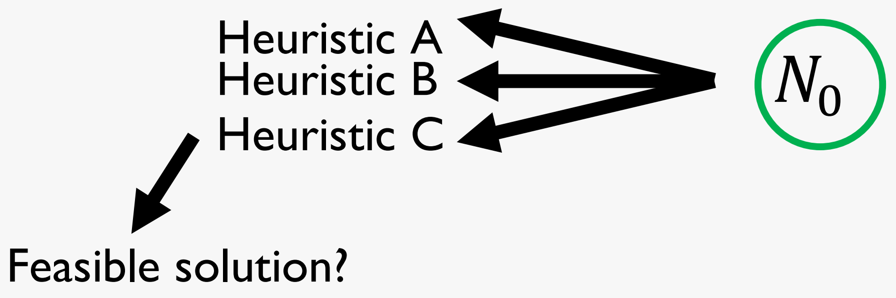


Exact Solving

Branching

Heuristic Selection





Graph Optimization

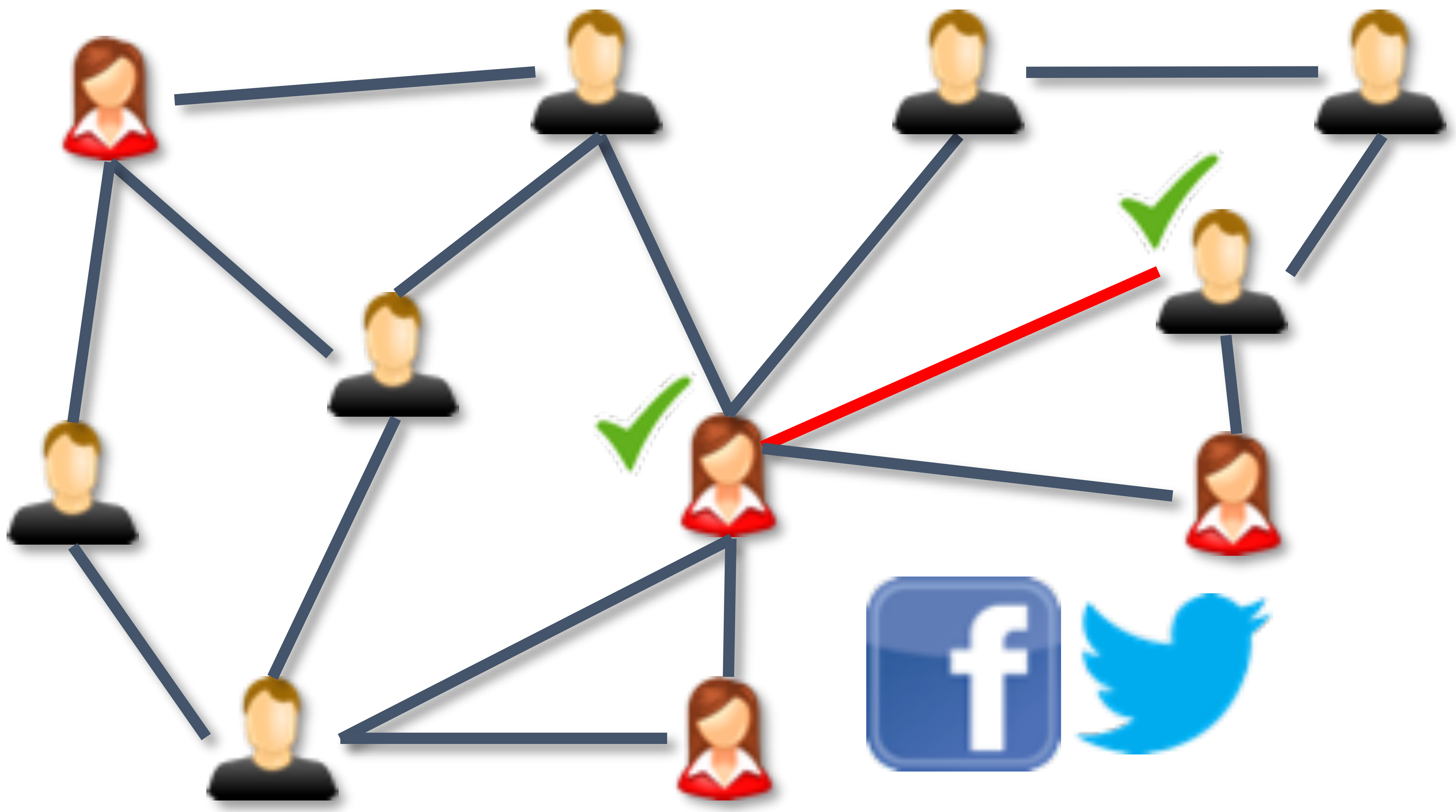
Integer Programming

Problem Type

Greedy Graph Optimization

Minimum Vertex Cover
Find smallest vertex subset such that each edge is covered

2-Approximation:
Greedily add vertices of edge with **max degree sum**



Greedy Graph Optimization

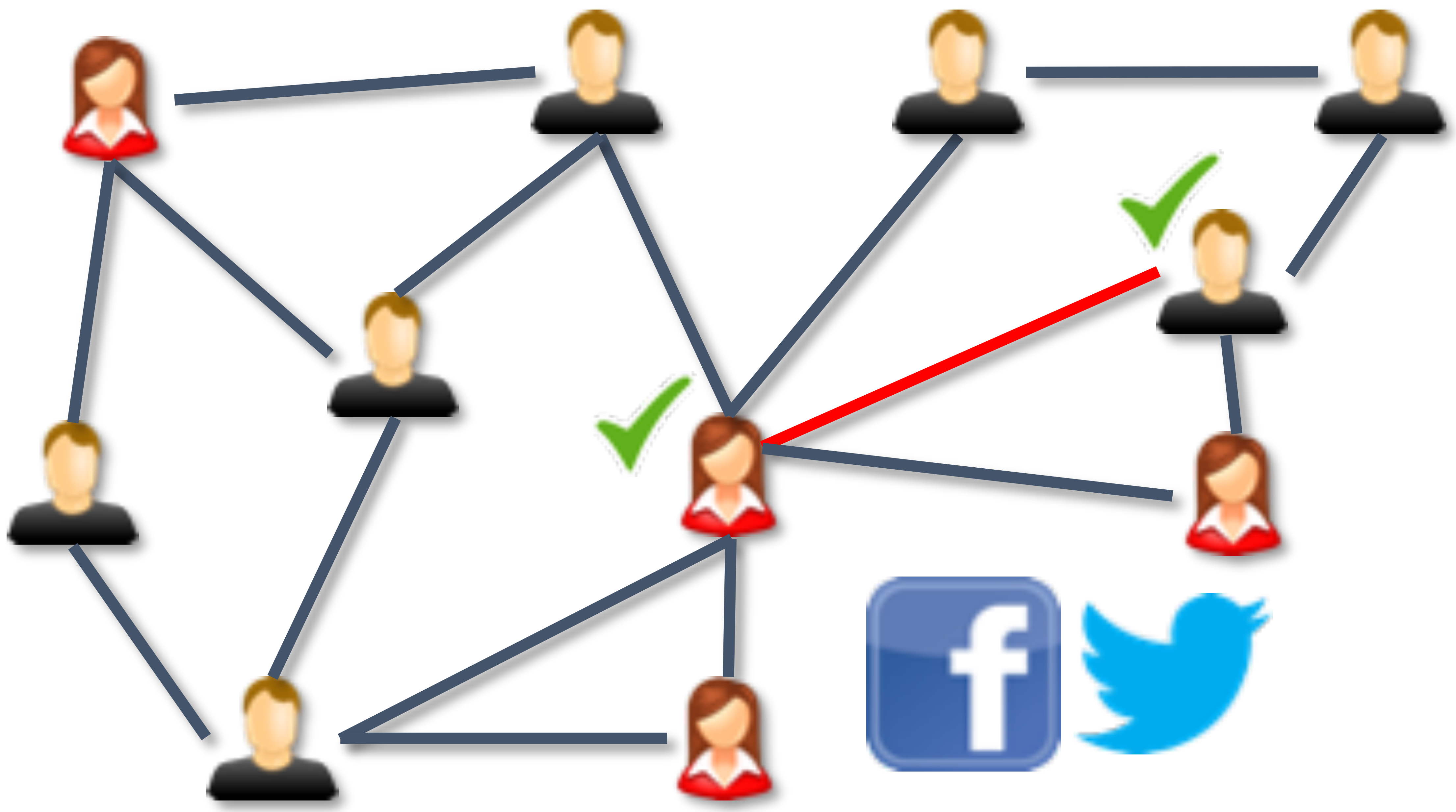
Minimum Vertex Cover

Find smallest vertex subset such that each edge is covered

Learning Greedy Graph Heuristics
[Dai*, Khalil*, Zhang, Dilkina, Song, 2017]

Given: graph problem, family of graphs

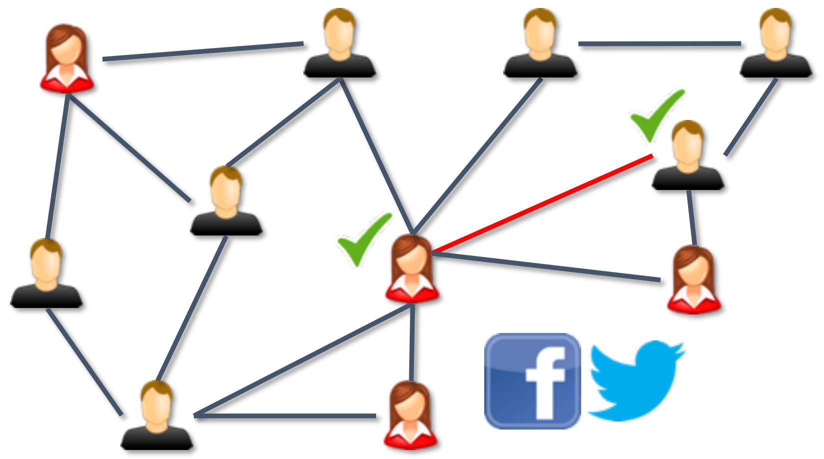
Learn: a **scoring function** to **guide** a **greedy** algorithm

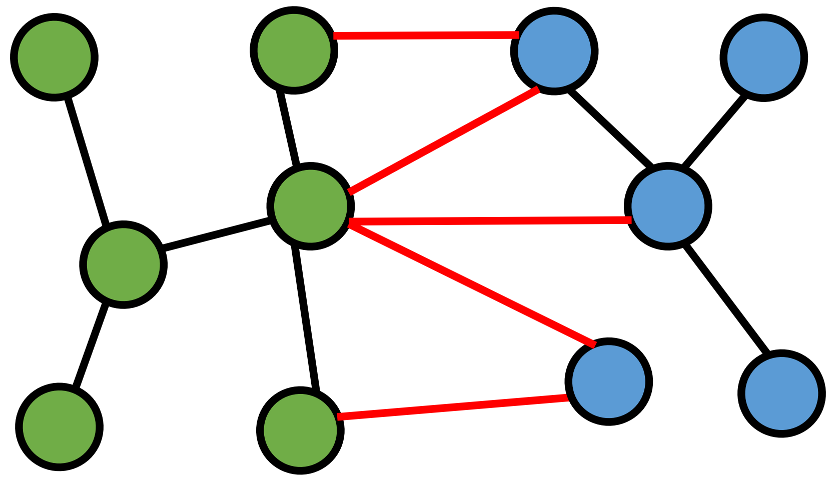


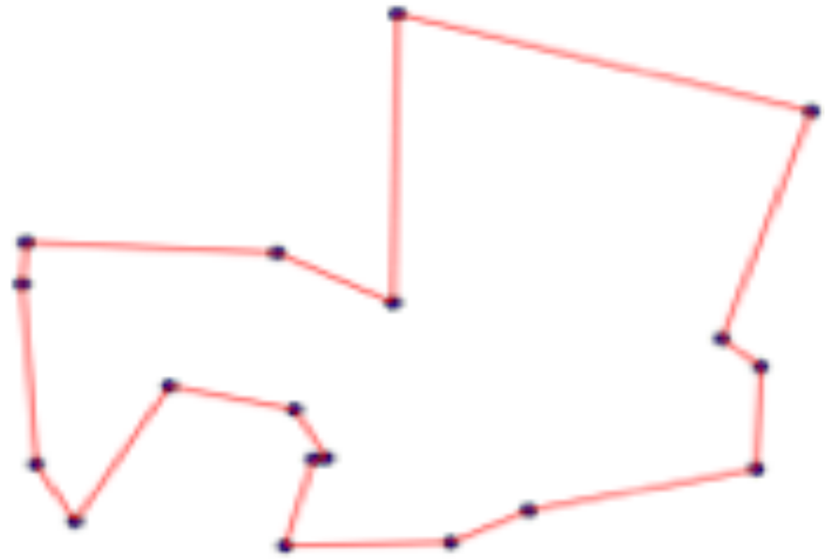
Learning Greedy Heuristics

Given: graph problem, family of graphs
Learn: a **scoring function** to **guide** a **greedy** algorithm

Problem	Minimum Vertex Cover	Maximum Cut	Traveling Salesman Problem
Domain	Social network snapshots	Spin glass models	Package delivery
Greedy operation	Insert nodes into cover	Insert nodes into subset	Insert nodes into sub-tour





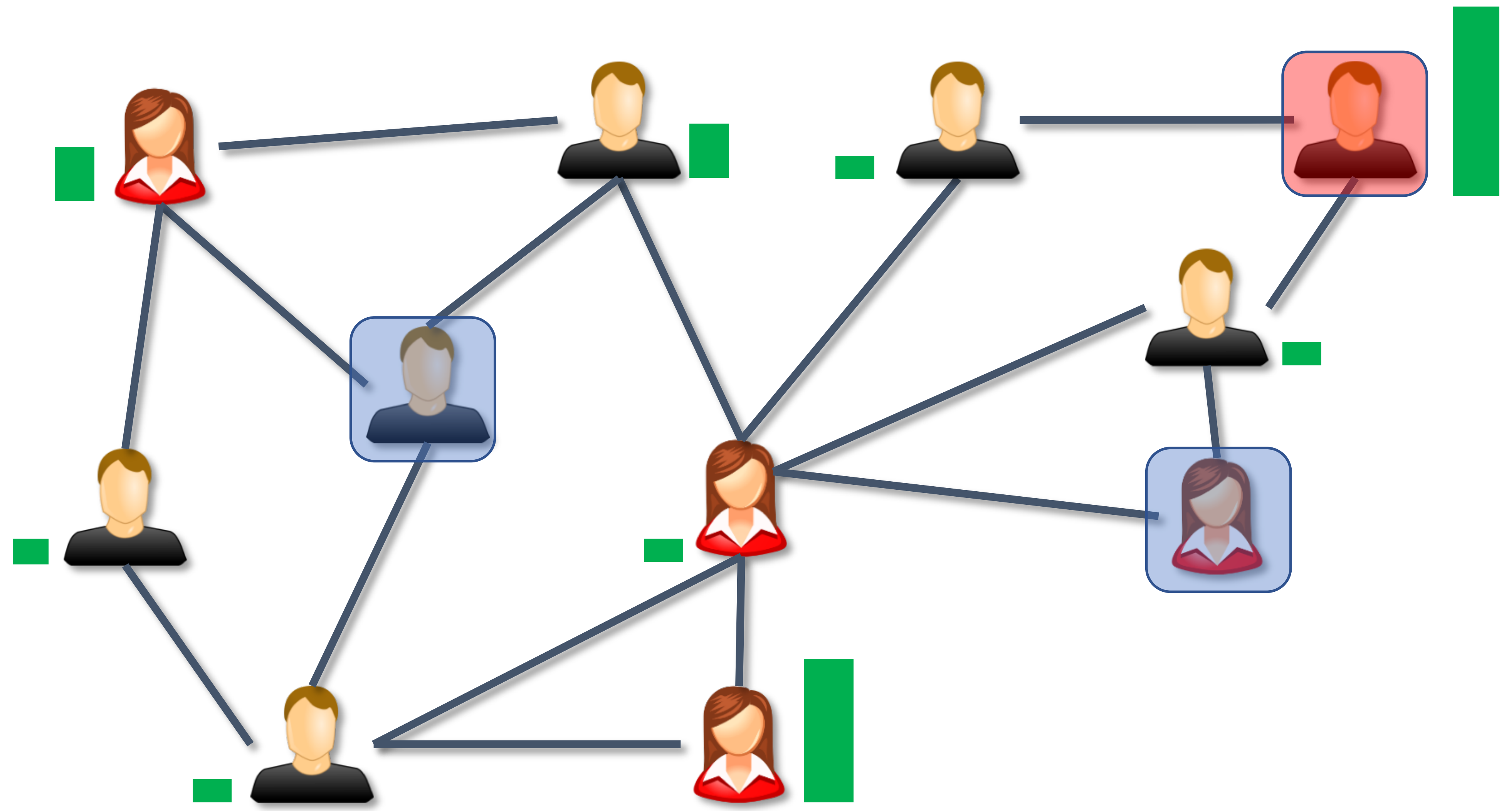


Reinforcement Learning

Greedy Algorithm		Reinforcement Learning
Partial solution	≡	State
Scoring function	≡	Q-function
Select best node	≡	Greedy Policy

Repeat until all edges are covered:

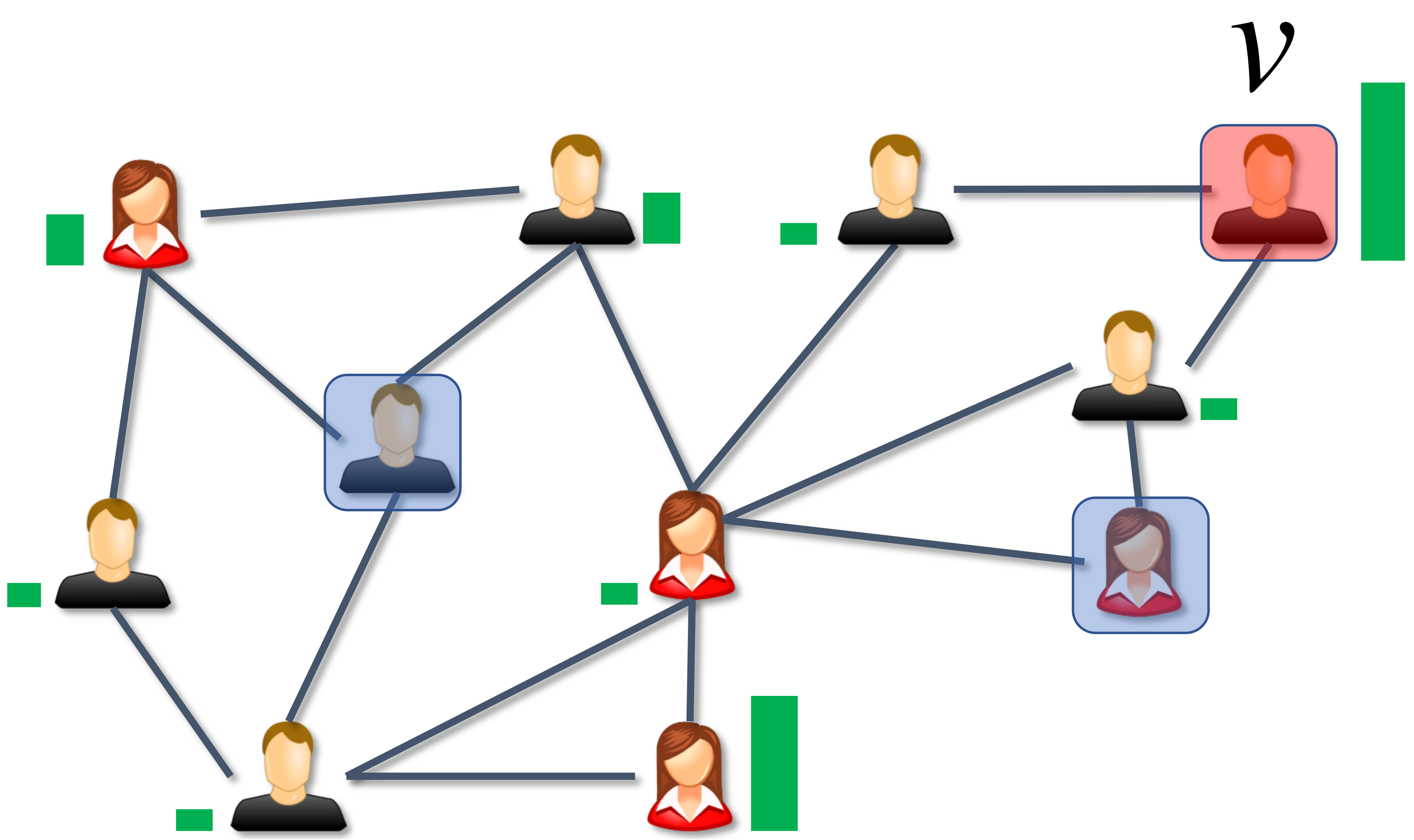
1. Compute node scores
2. Select best node w.r.t. score
3. Add best node to partial sol.



Partial Solution

Learning Node Features

Scoring Function: Need to represent node with a **feature vector** first



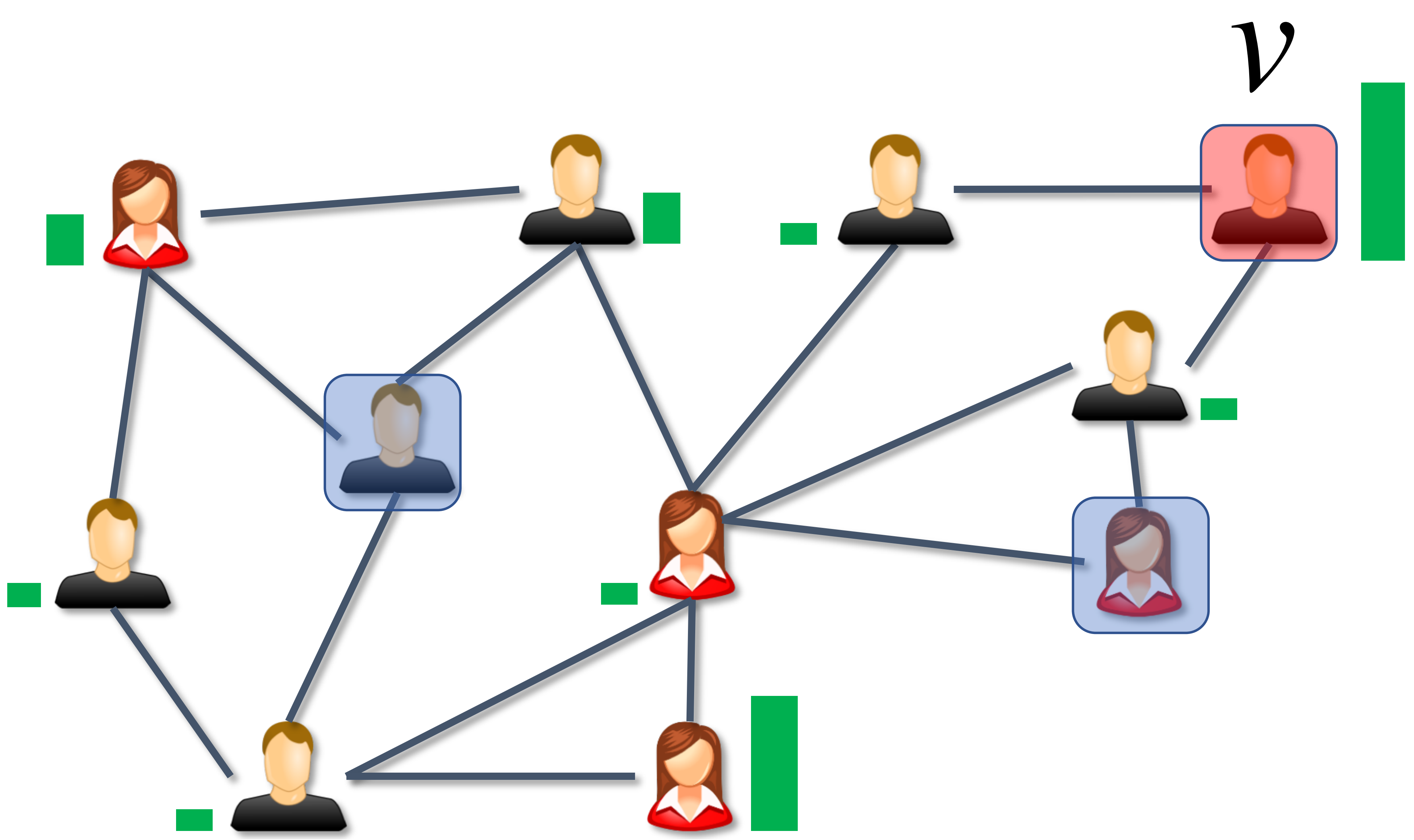
$$= Q(S_t, v; \Theta)$$

$S_t = \{$  $\}$

Learning Node Features

Scoring Function: Need to represent node with a **feature vector** first

Problem: Not clear what good node features are!



$$= Q(S_t, v; \Theta)$$

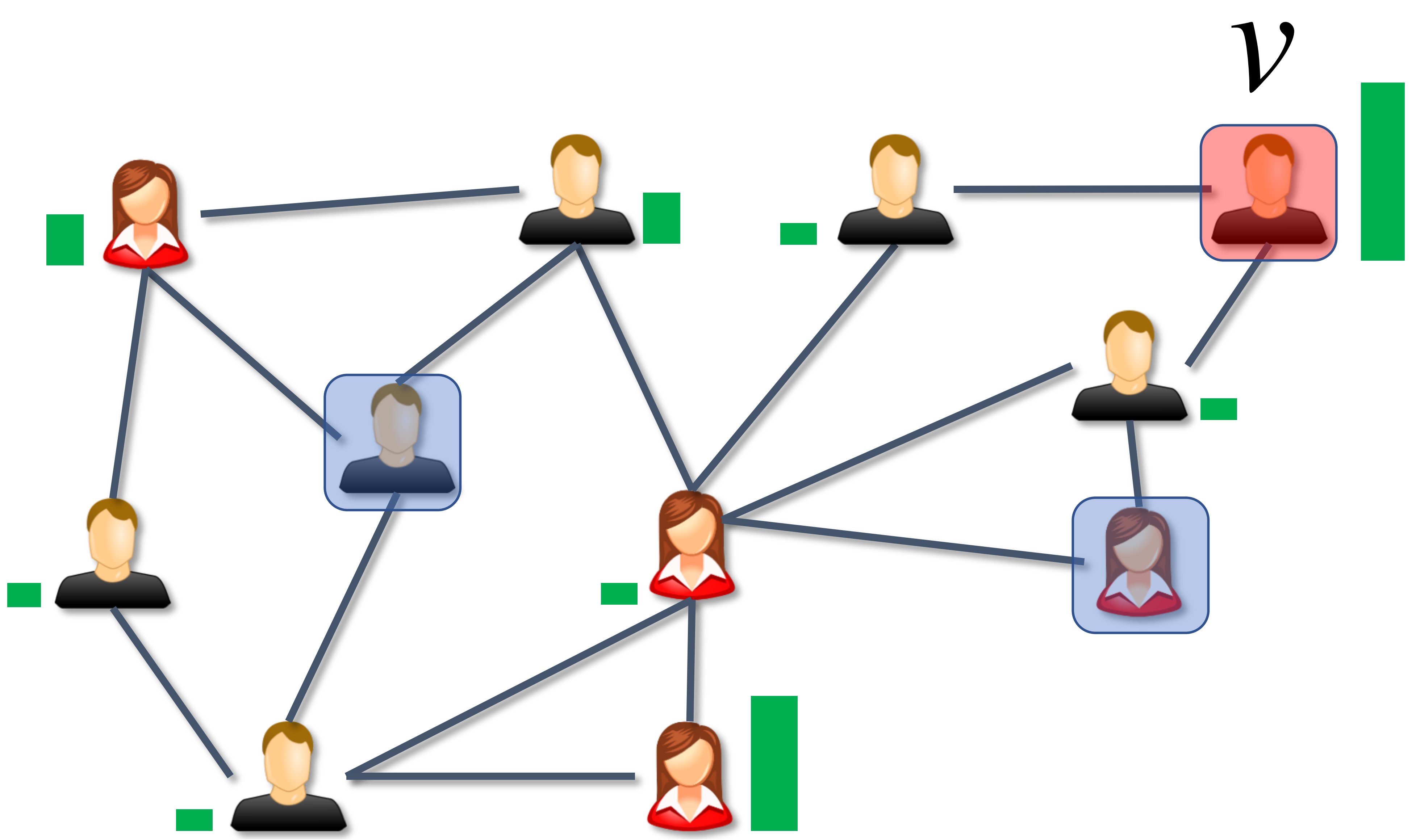
$$S_t = \{\ \}$$

Learning Node Features

Scoring Function: Need to represent node with a **feature vector** first

Problem: Not clear what good node features are!

Solution: Parametrize a **Graph Neural Network** with parameters Θ



$$= Q(S_t, v; \Theta)$$

$S_t = \{$  $\}$

Learning Node Features

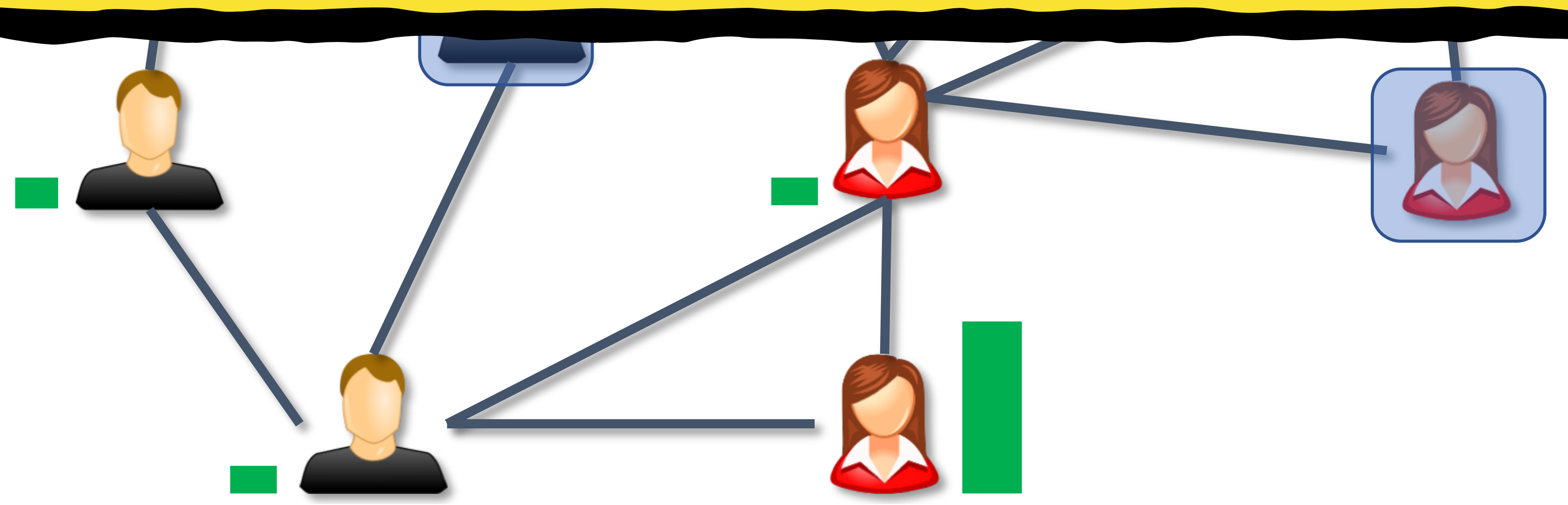
Scoring Function: Need to represent node with a **feature vector** first

Problem: Not clear what good node features are!

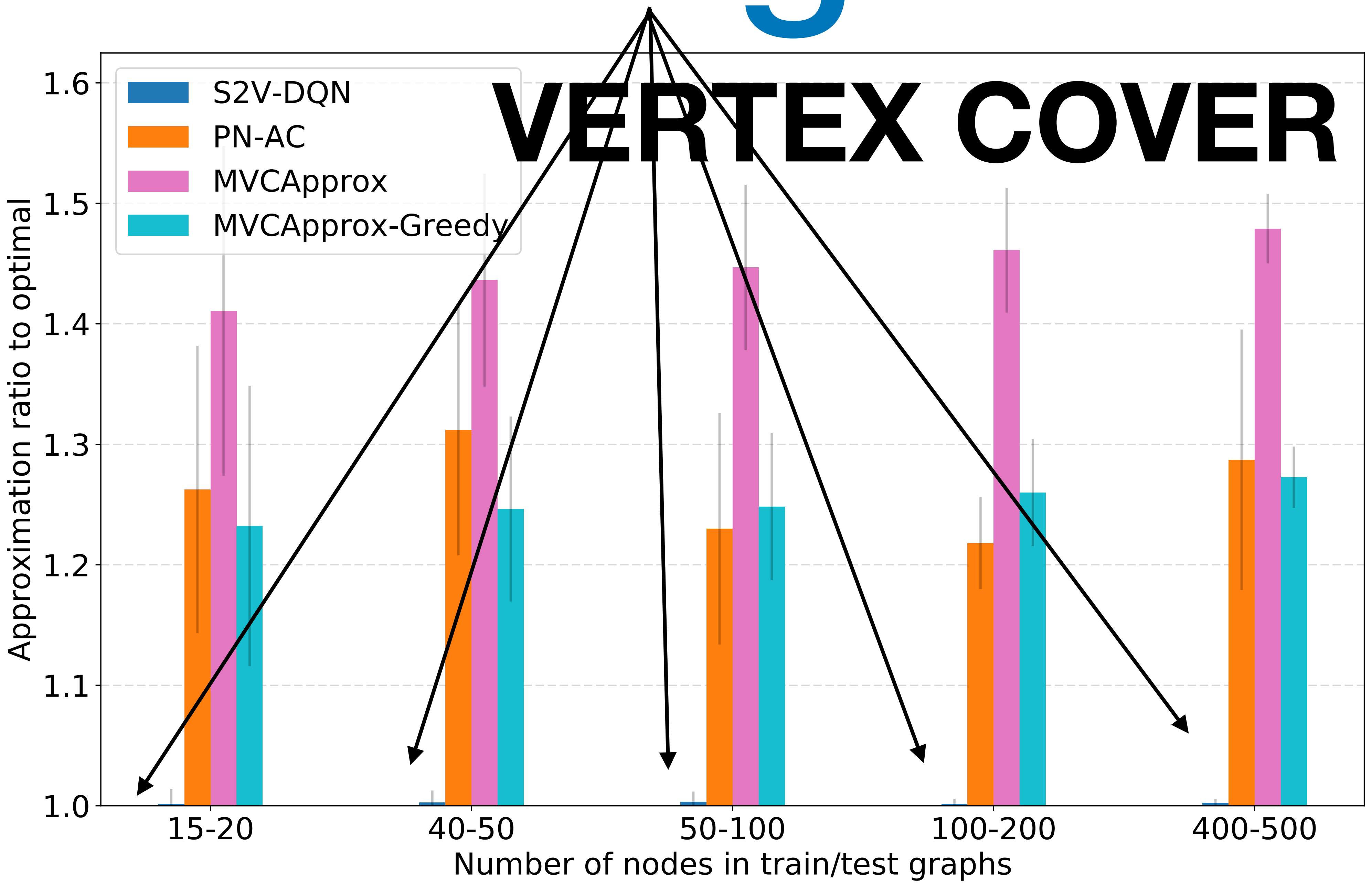
Solution: Parametrize a **Graph Neural Network** with parameters Θ

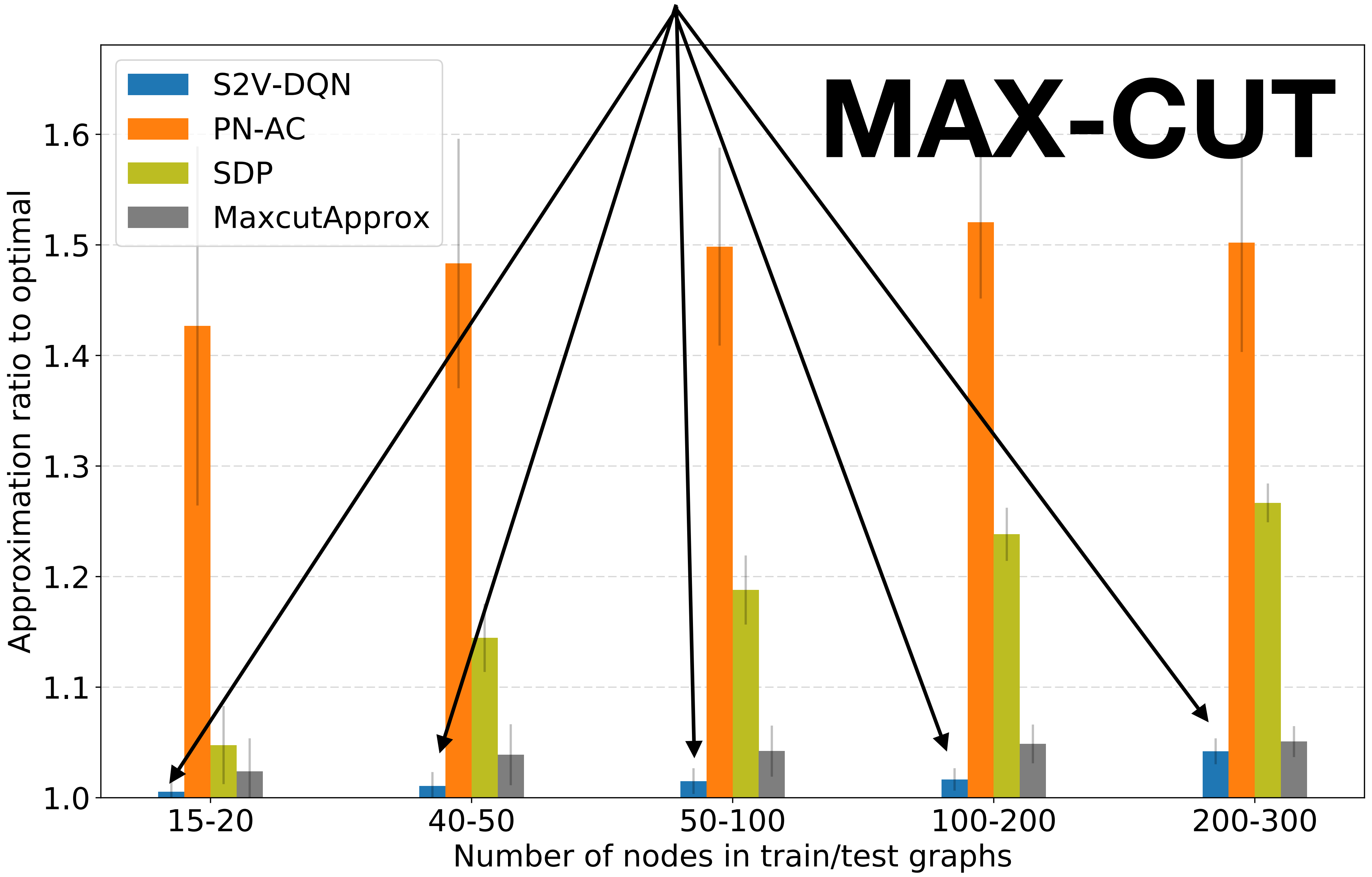
Run **RL** algorithm (e.g. Q-Learning)

Use **gradient of solution cost to update** Θ

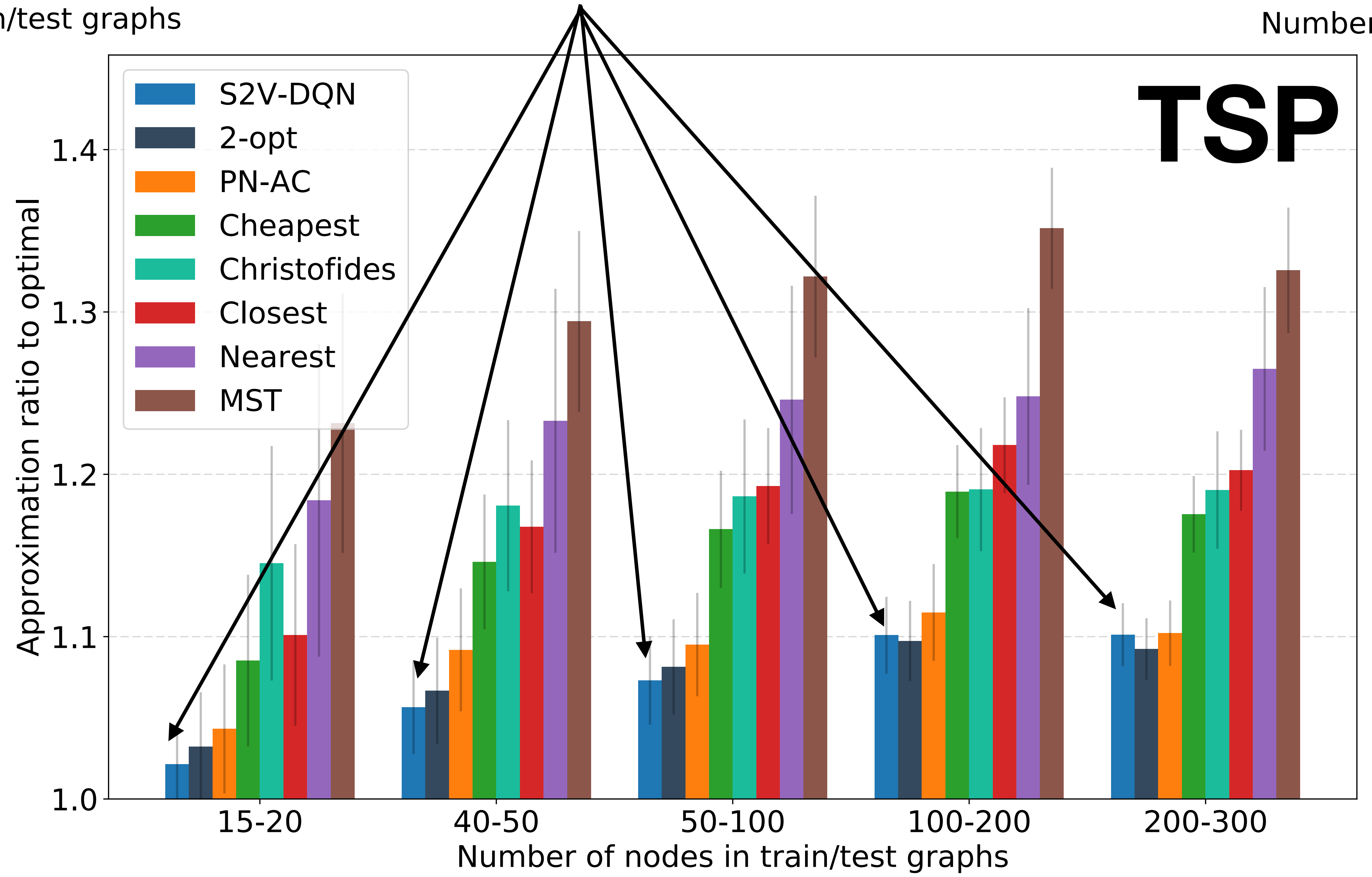


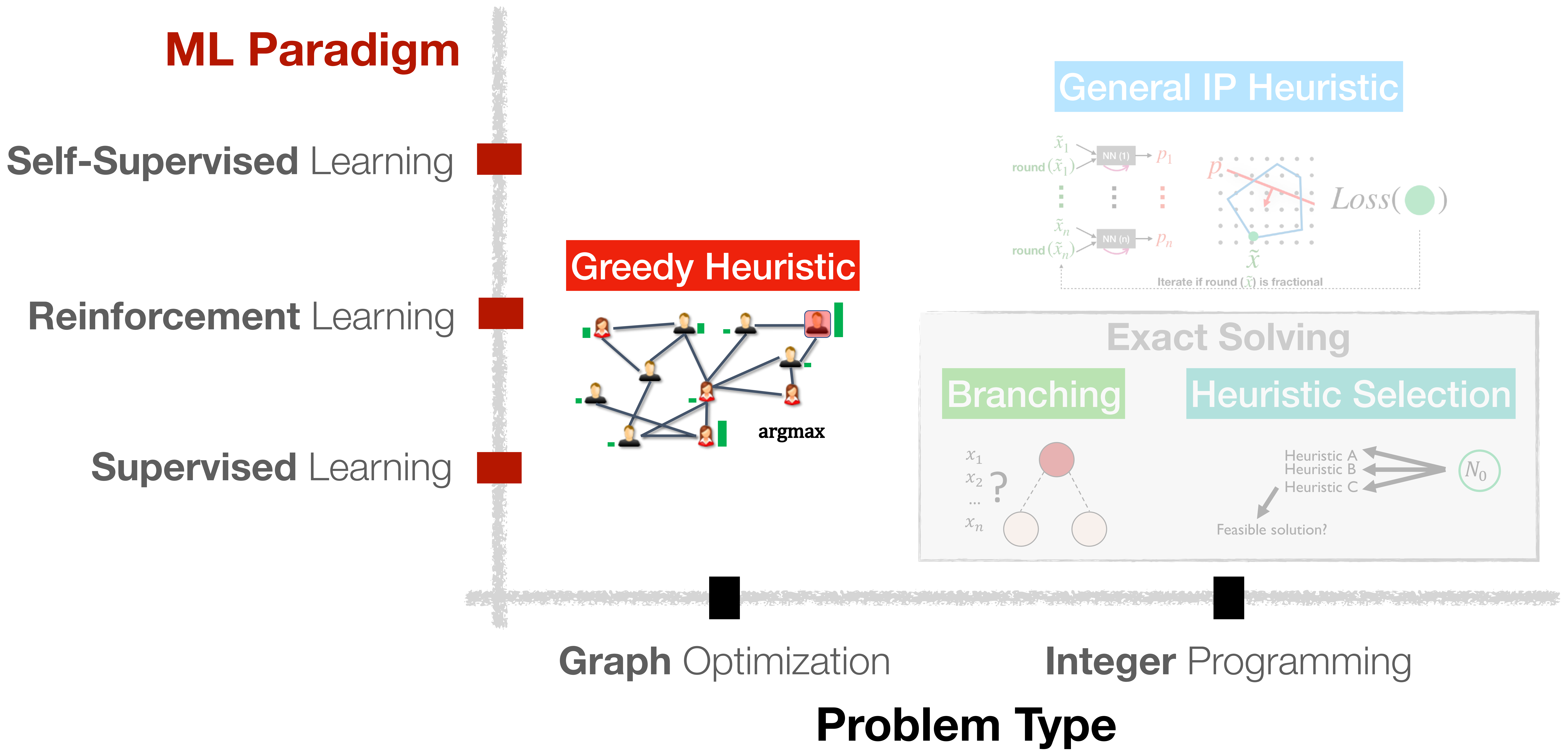
Learning Greedy in Practice





Approximation Ratio



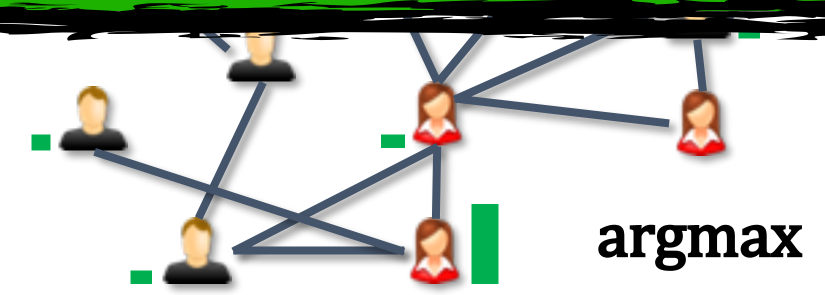

ML Paradigm
Self-Supervised Learning
Reinforcement Learning
Supervised Learning
General IP Heuristic
Loss
Iterate if round is fractional
Greedy Heuristic
argmax
Exact Solving
Branching
Heuristic Selection
Heuristic A
Heuristic B
Heuristic C
Feasible solution?
Graph Optimization
Integer Programming
Problem Type

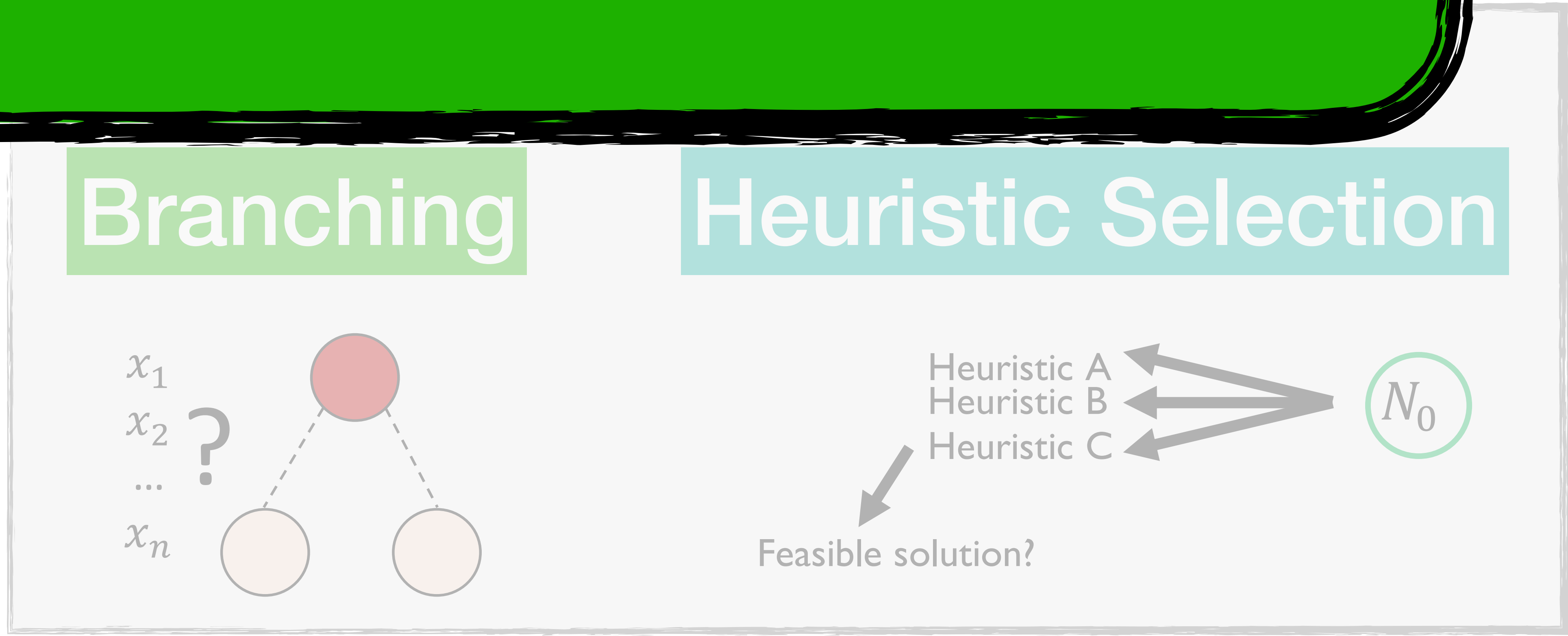
Takeaways

- **Reinforcement Learning tailors greedy** search to your instances
- Learn **features jointly with** greedy **policy**
- **Human priors** encoded via (greedy) **meta-algorithm**
- **Interesting, novel strategies emerge**

Self-S

Reinf





Supervised Learning

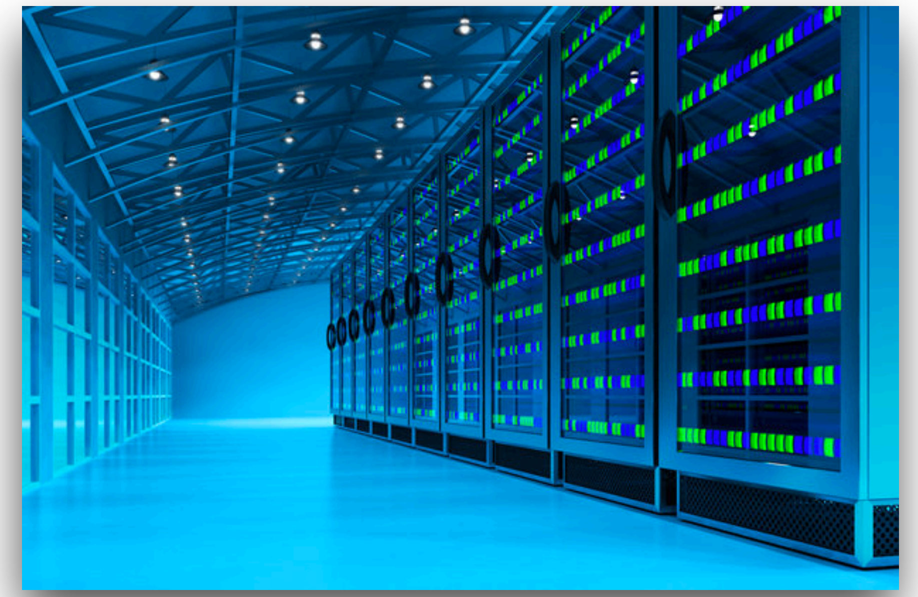
Graph Optimization

Integer Programming

Problem Type

Power Systems

...



Data Center
Resource Management

...

Airline Scheduling

$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

General Heuristic

Feasible Solution

General IP Heuristics

Power Systems

...

Data Center Resource Management

...

Airline Scheduling

$$\min_x c^T x \textbf{ s.t. } Ax \le b, x \in \{0,1\}^n$$

↓

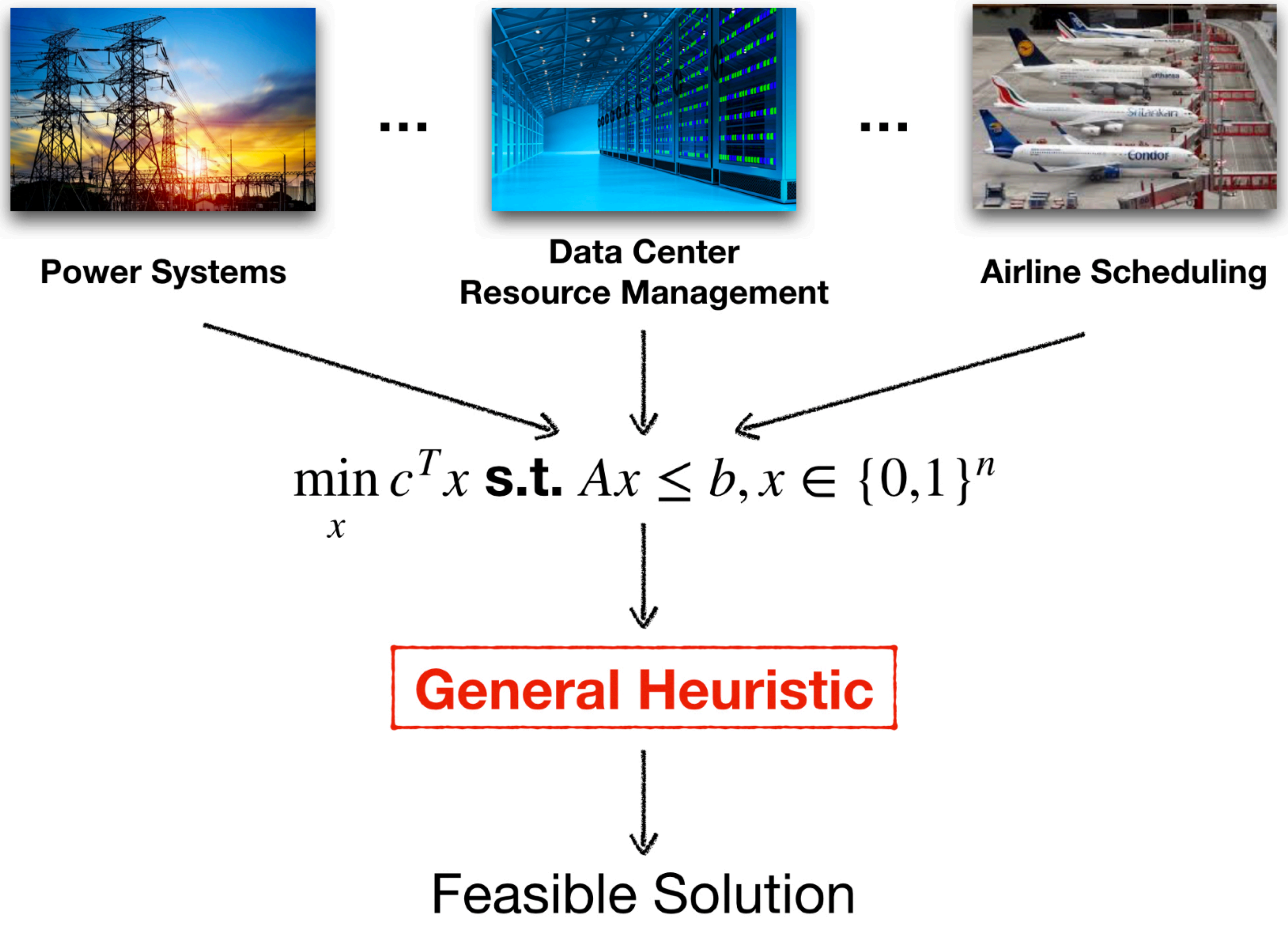
General Heuristic

↓

Feasible Solution

General IP Heuristics

Strengths



General IP Heuristics

Strengths

- Applicable to many problems
- Usable inside Branch-and-Bound

$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

General Heuristic

Feasible Solution

Feasibility Pump

0 Start with LP-feasible (fractional) solution

1 **Round** to nearest integer, **return** if LP-feasible

2 **Project** integer point to **nearest** LP-feasible point

3 Go back to step 1

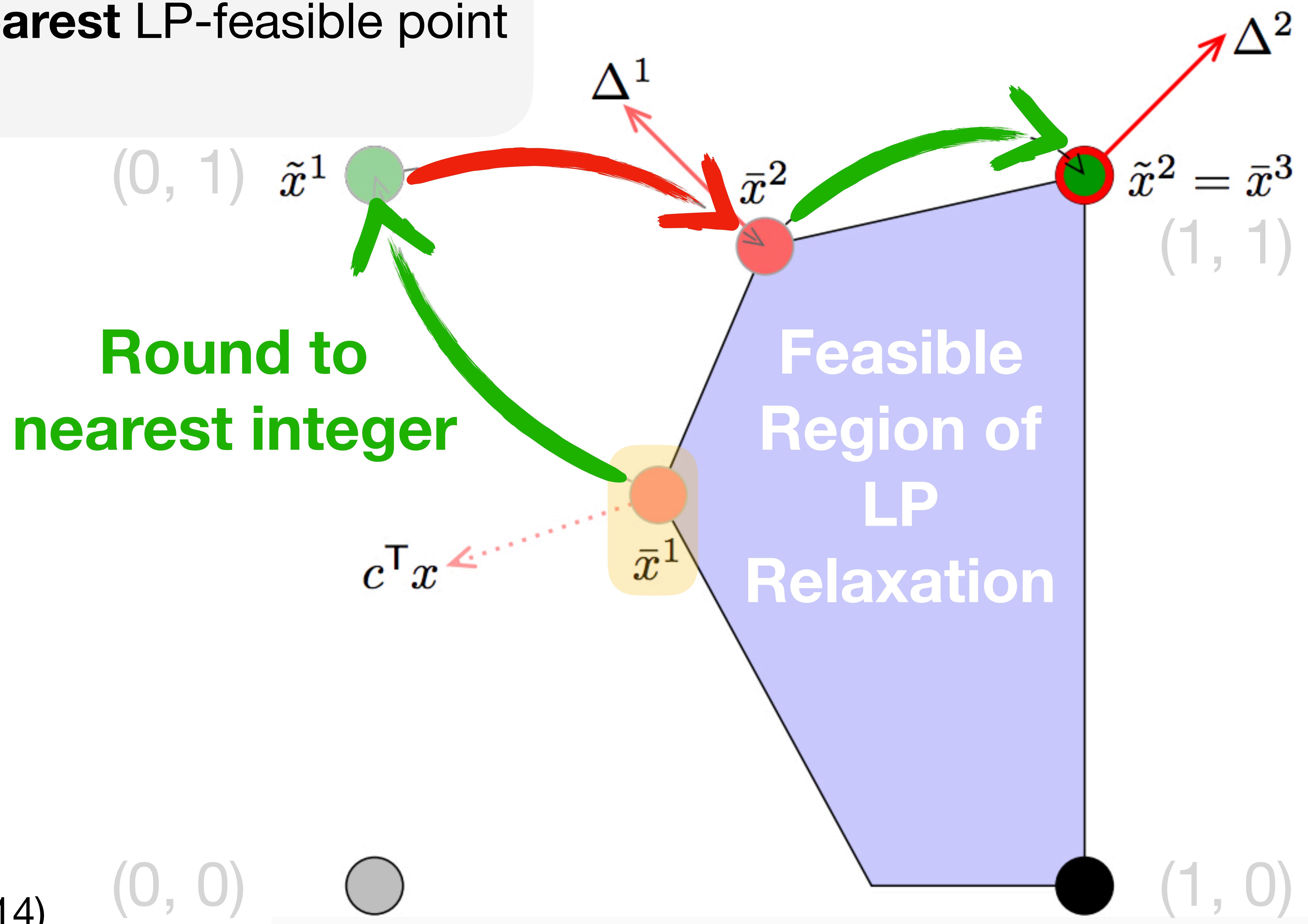
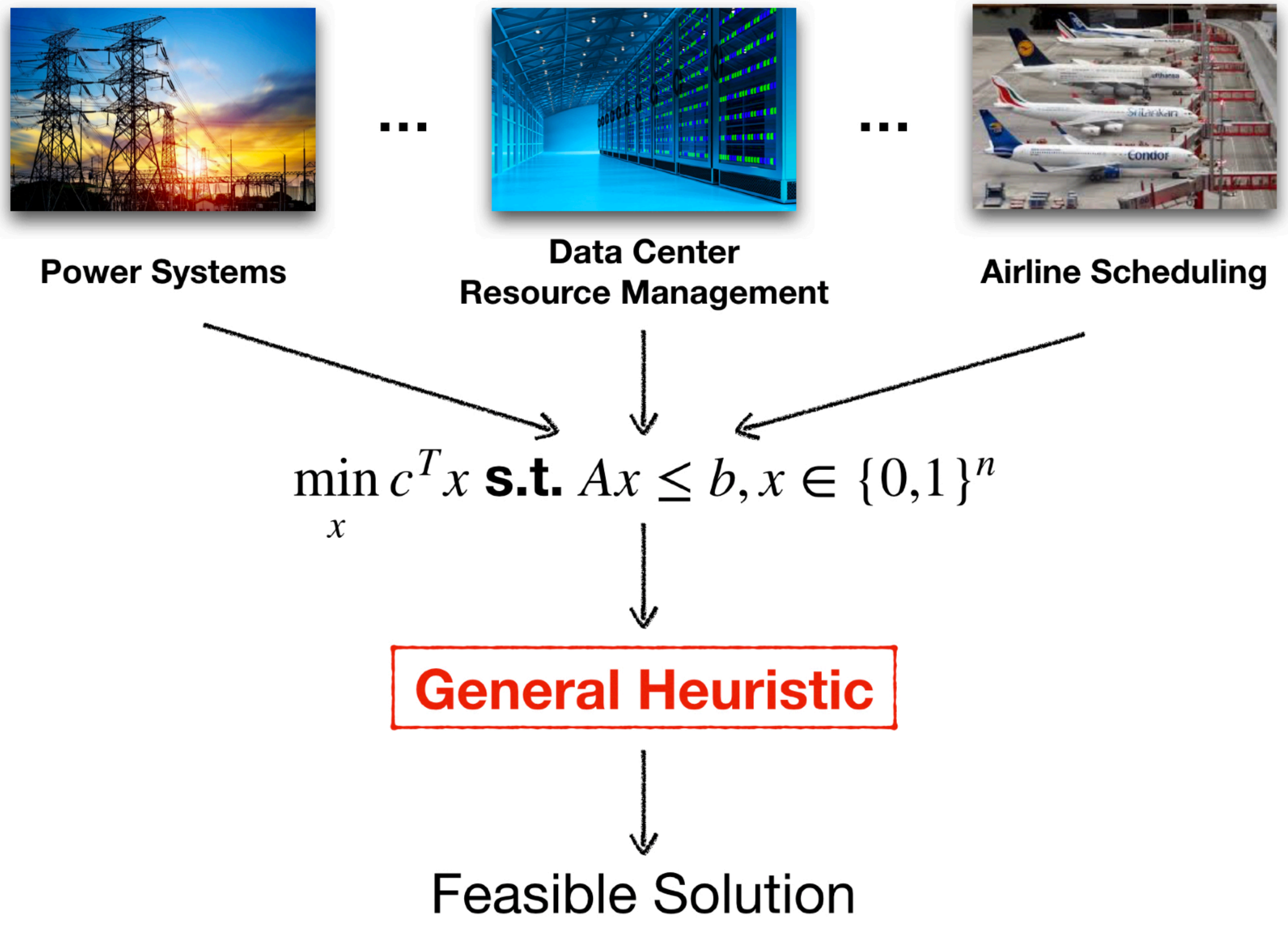


 Figure in part from Berthold (2014)

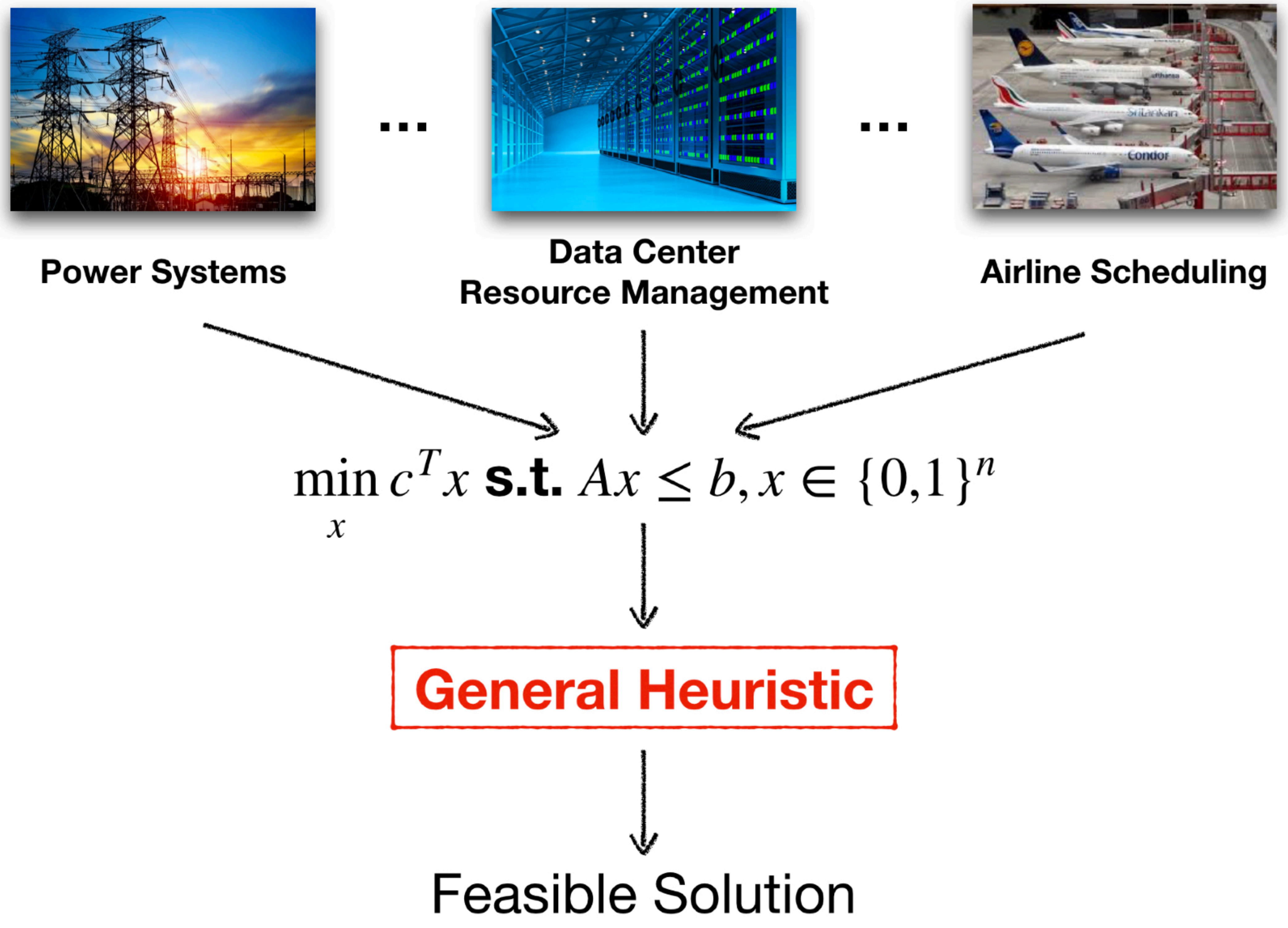
General IP Heuristics

Strengths

- Applicable to many problems
- Usable inside Branch-and-Bound



General IP Heuristics

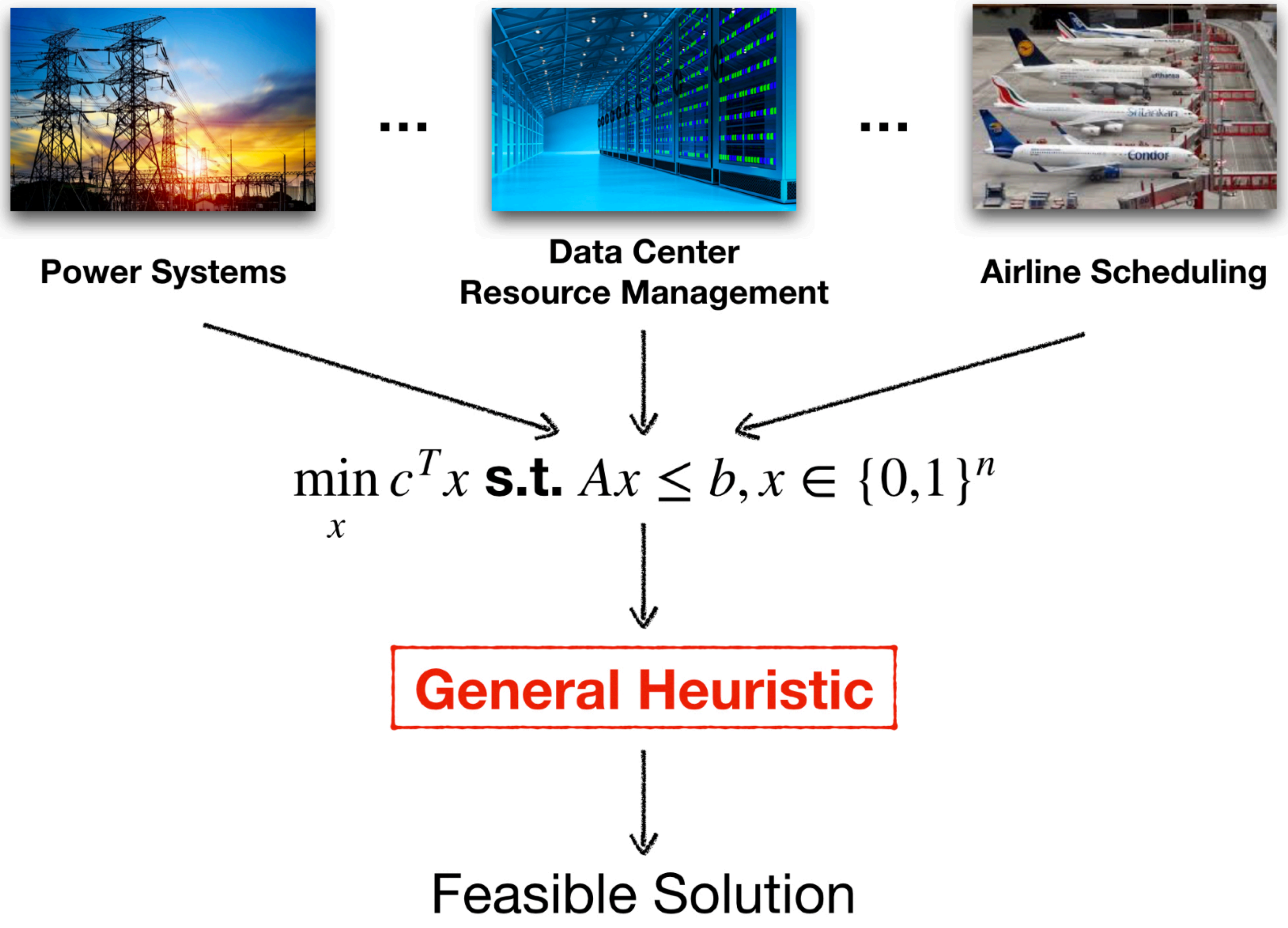


Strengths

- Applicable to many problems
- Usable inside Branch-and-Bound

Weaknesses

General IP Heuristics

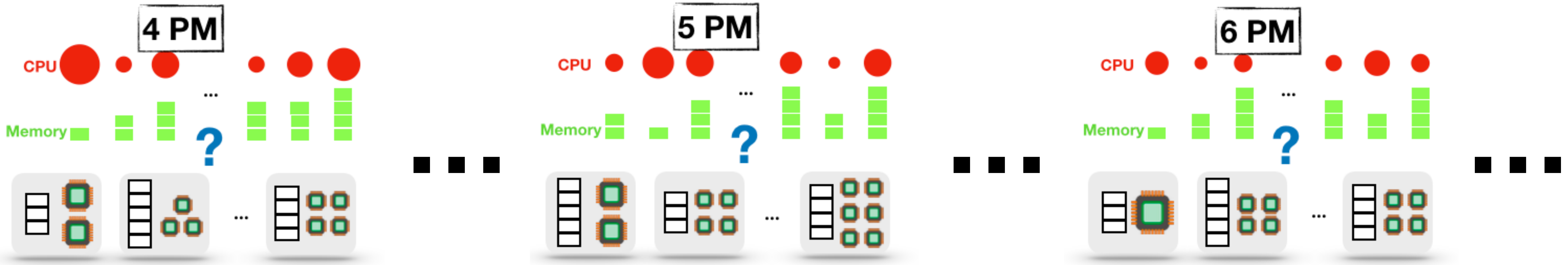


Strengths

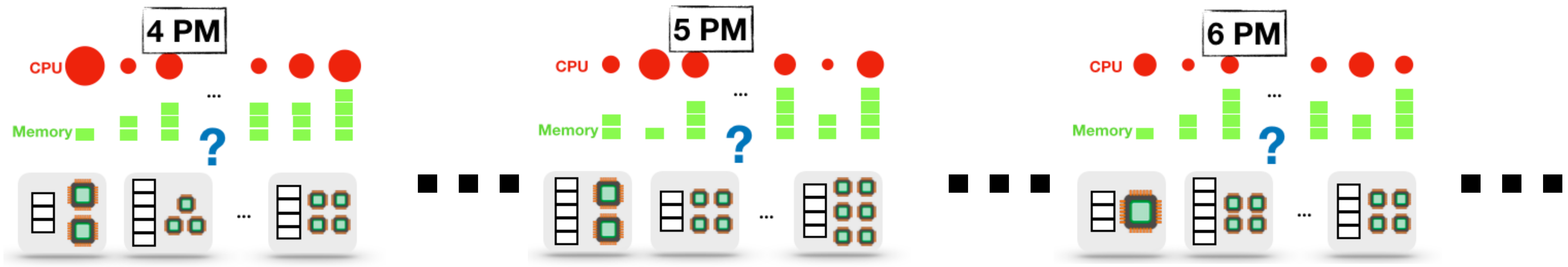
- Applicable to many problems
- Usable inside Branch-and-Bound

Weaknesses

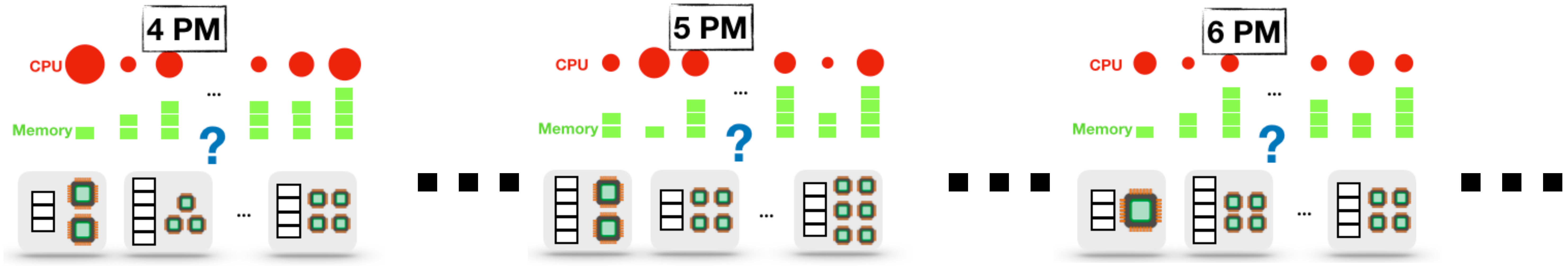
- May not work well for **your** problem
- Cannot exploit **distribution of instances**



Problem Statement

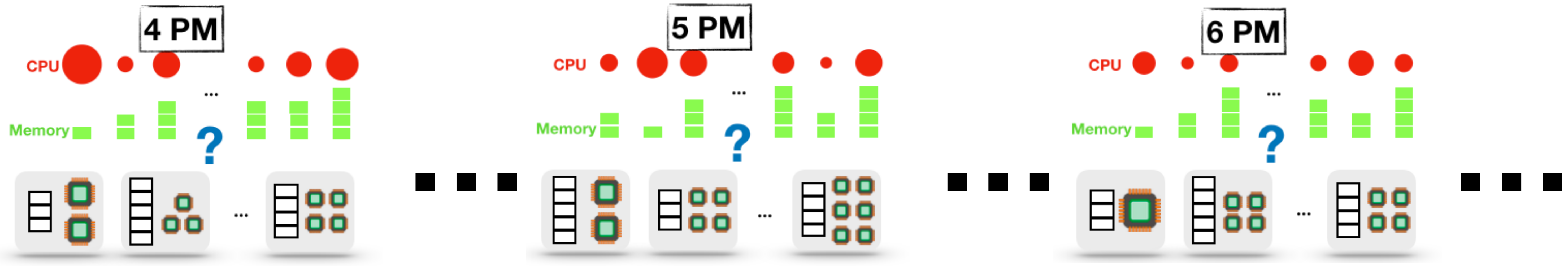


Problem Statement



$$
\begin{aligned}
\underset{x}{\text{maximize}} \quad & \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{n} w_{ij} x_{ij} \leq c_i, \; i = 1, \ldots, m, \\
& \sum_{i=1}^{m} x_{ij} = 1, \; j = 1, \ldots, n, \\
& x_{ij} \in \{0, 1\}, \; i = 1, \ldots, m, j = 1, \ldots, n.
\end{aligned}
$$

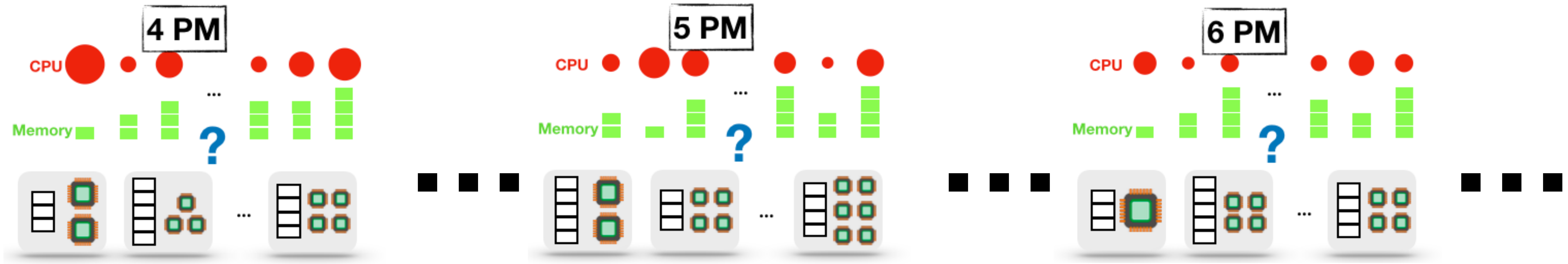
Problem Statement



$\mathcal{I}$: set of **training IP instances**

$$
\begin{aligned}
\underset{x}{\text{maximize}} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{n} w_{ij}x_{ij} \leq c_i,\ i = 1,\ldots,m, \\
& \sum_{i=1}^{m} x_{ij} = 1,\ j = 1,\ldots,n, \\
& x_{ij} \in \{0,1\},\ i = 1,\ldots,m, j = 1,\ldots,n.
\end{aligned}
$$

Problem Statement



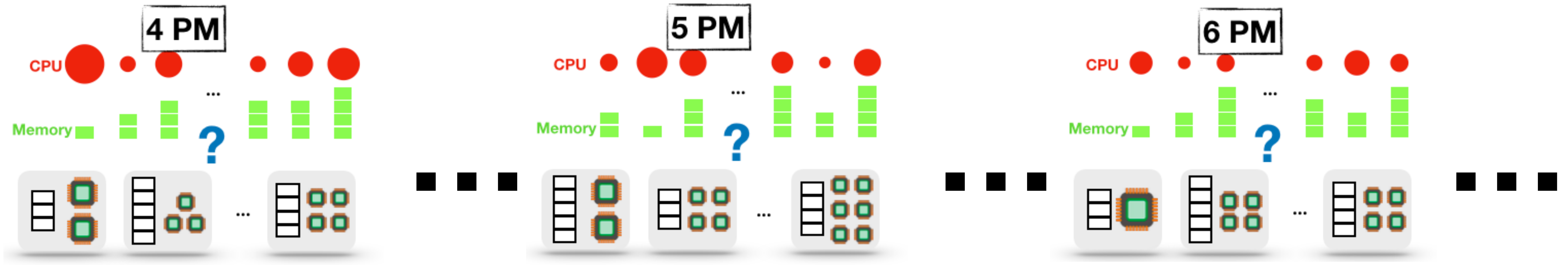
$\mathcal{I}$: set of **training IP instances**

$$
\begin{aligned}
\underset{x}{\text{maximize}} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{n} w_{ij}x_{ij} \le c_i,\ i = 1,\ldots,m, \\
& \sum_{i=1}^{m} x_{ij} = 1,\ j = 1,\ldots,n, \\
& x_{ij} \in \{0,1\},\ i = 1,\ldots,m, j = 1,\ldots,n.
\end{aligned}
$$

$\mathcal{A}$: a **parametric algorithm**; outputs $\begin{cases} 1 \text{ if feasible solution is found} \\ 0 \text{ otherwise} \end{cases}$

Θ

Problem Statement



$\mathcal{I}$: set of **training IP instances**

$$
\begin{aligned}
\underset{x}{\text{maximize}} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{n} w_{ij}x_{ij} \leq c_i, \; i = 1, \ldots, m, \\
& \sum_{i=1}^{m} x_{ij} = 1, \; j = 1, \ldots, n, \\
& x_{ij} \in \{0,1\}, \; i = 1, \ldots, m, j = 1, \ldots, n.
\end{aligned}
$$

$$\text{Find } \Theta^* = \underset{\Theta \in \mathbb{R}^p}{\arg\max} \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} \mathcal{A}(I; \Theta)$$

$\mathcal{A}$: a **parametric algorithm** (Θ); outputs $\begin{cases} 1 \text{ if feasible solution is found} \\ 0 \text{ otherwise} \end{cases}$

Towards Learning General Heuristics

$$\text{Find } \Theta^* = \underset{\Theta \in \mathbb{R}^p}{\arg\max} \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} \mathcal{A}(I; \Theta)$$

Towards Learning General Heuristics

$$\text{Find } \Theta^* = \arg\max_{\Theta \in \mathbb{R}^p} \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} \mathcal{A}(I; \Theta)$$

1 *What **type of algorithm** is* $\mathcal{A}$ *?*

Towards Learning General Heuristics

$$\text{Find } \Theta^* = \arg\max_{\Theta \in \mathbb{R}^p} \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} \mathcal{A}(I; \Theta)$$

1. *What **type of algorithm** is* $\mathcal{A}$ *?*

2. *What is the **role of the ML model**, parameterized by* Θ*, in* $\mathcal{A}$*?*

Towards Learning General Heuristics

$$\text{Find } \Theta^* = \underset{\Theta \in \mathbb{R}^p}{\arg\max} \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} \mathcal{A}(I; \Theta)$$

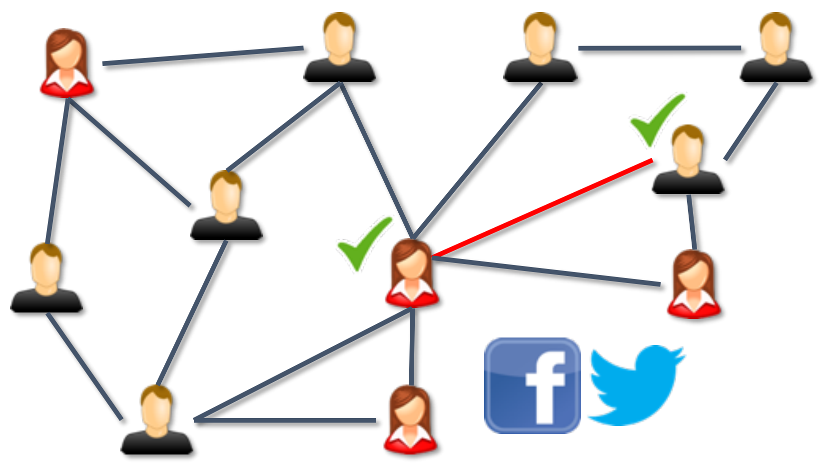
1. *What* ***type of algorithm*** *is* $\mathcal{A}$ *?*
2. *What is the* ***role of the ML model****, parameterized by* Θ*, in* $\mathcal{A}$*?*
3. *How can we* ***train*** *the algorithm?*

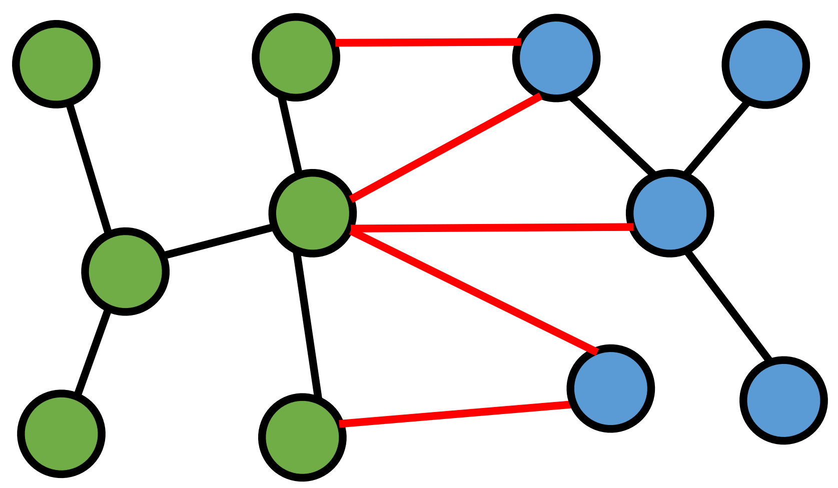
1 *What* ***type of algorithm*** *is* $\mathcal{A}$ *?*

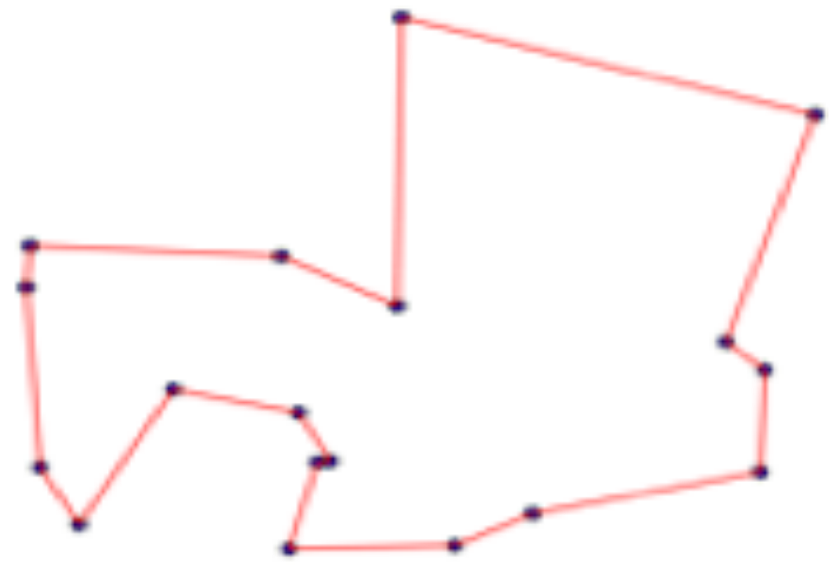
[Dai & Khalil, et al. (2017)]

Given: graph problem, family of graphs
Learn: a **scoring function** to **guide** a **greedy** algorithm

Problem	Minimum Vertex Cover	Maximum Cut	Traveling Salesman Problem
Domain	Social network snapshots	Spin glass models	Package delivery
Greedy operation	Insert nodes into cover	Insert nodes into subset	Insert nodes into sub-tour





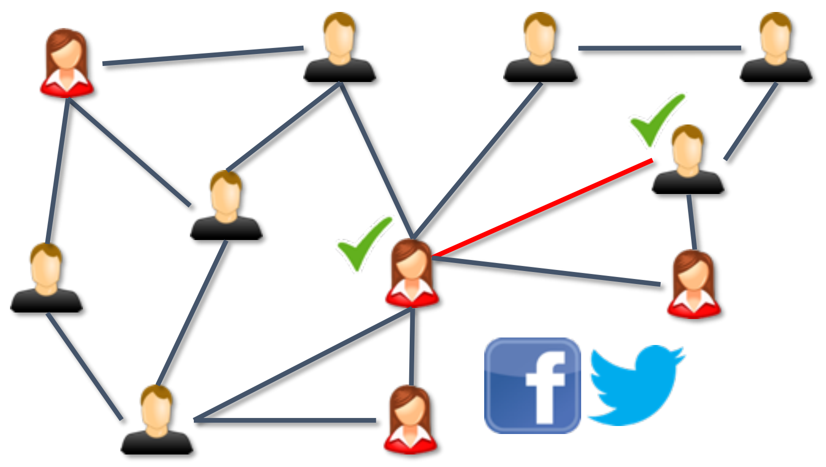
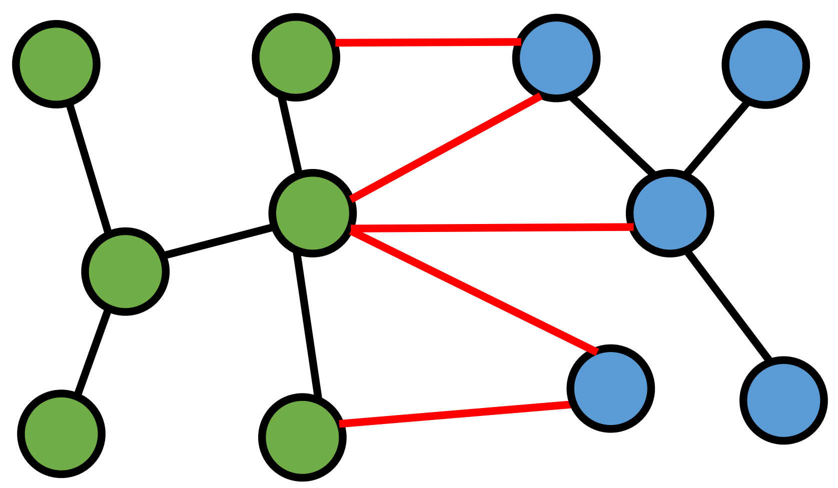
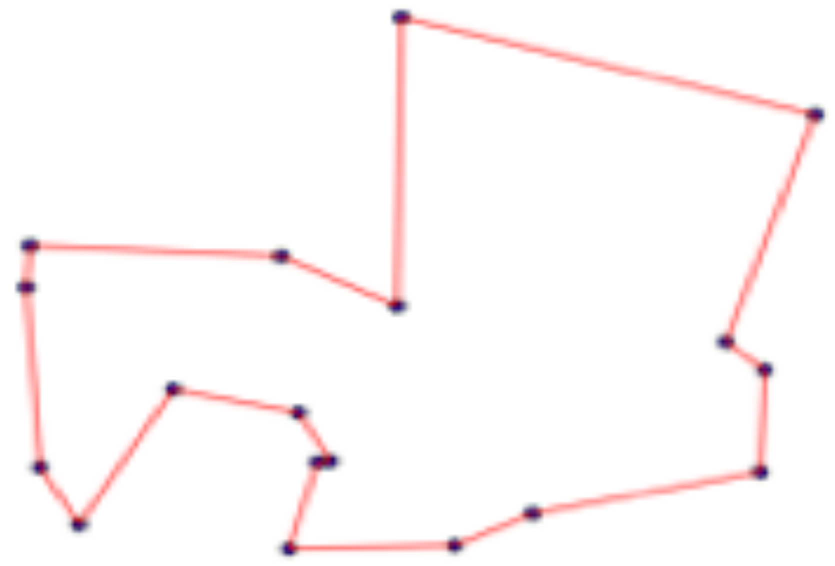


1 What **type of algorithm** is $\mathcal{A}$?

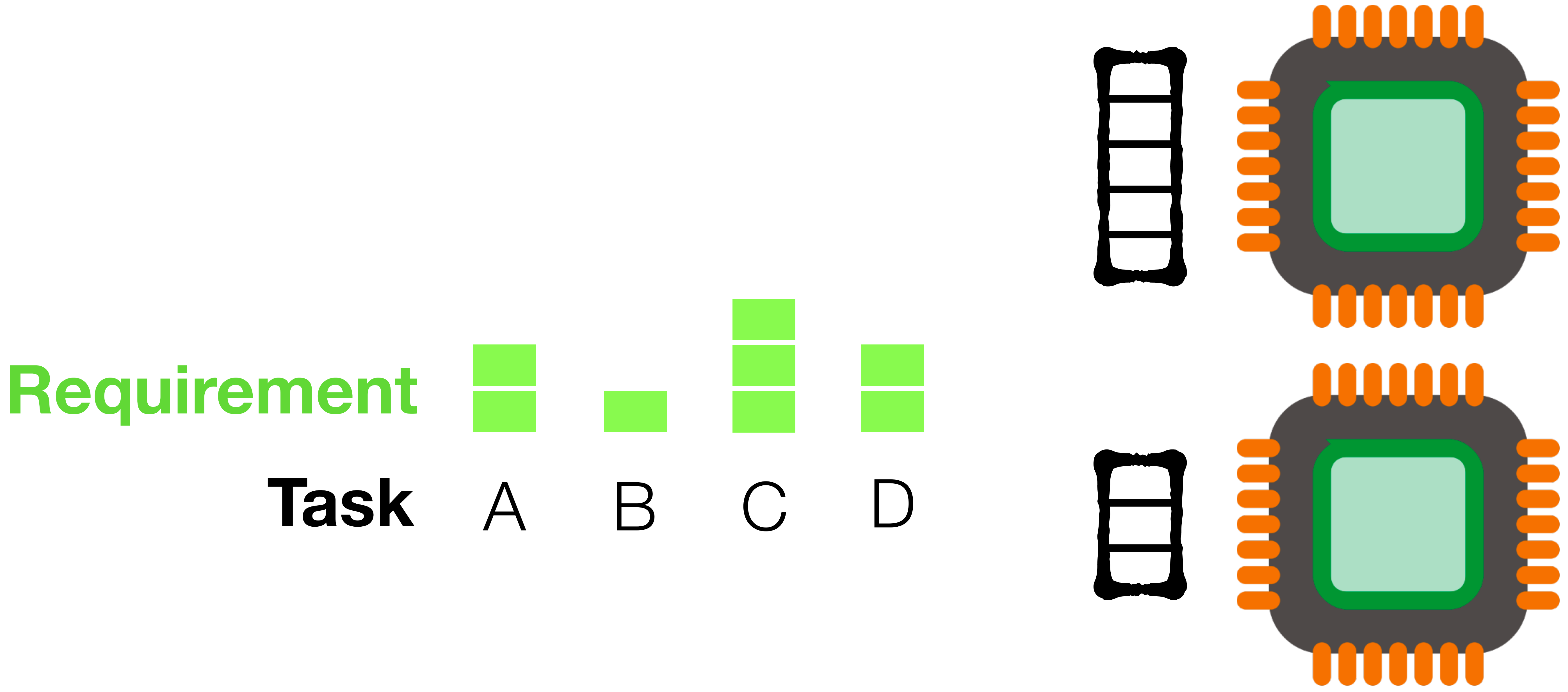
[Dai & Khalil, et al. (2017)]

Given: graph problem, family of graphs
Learn: a **scoring function** to **guide** a **greedy** algorithm

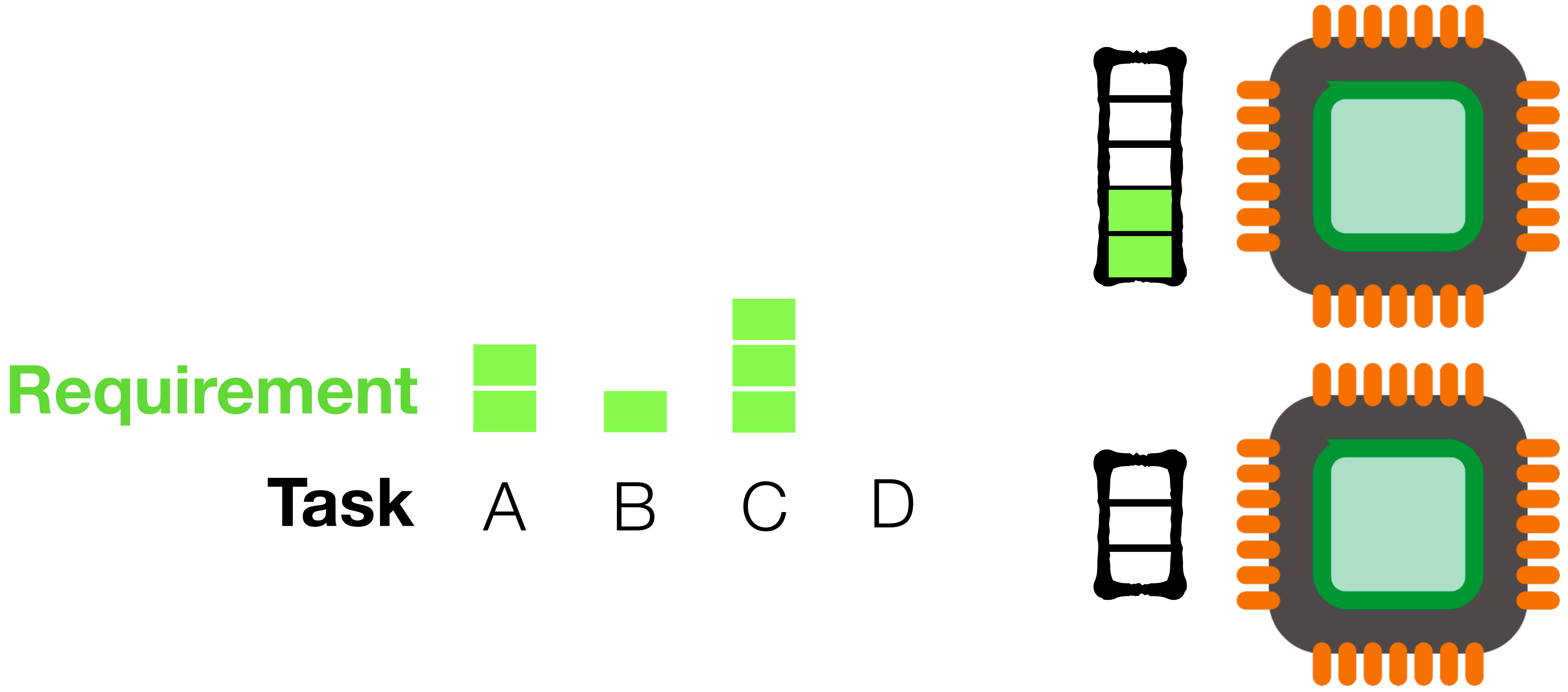
Problem	Minimum Vertex Cover	Maximum Cut	Traveling Salesman Problem
Domain	Social network snapshots	Spin glass models	Package delivery
Greedy operation	Insert nodes into cover	Insert nodes into subset	Insert nodes into sub-tour

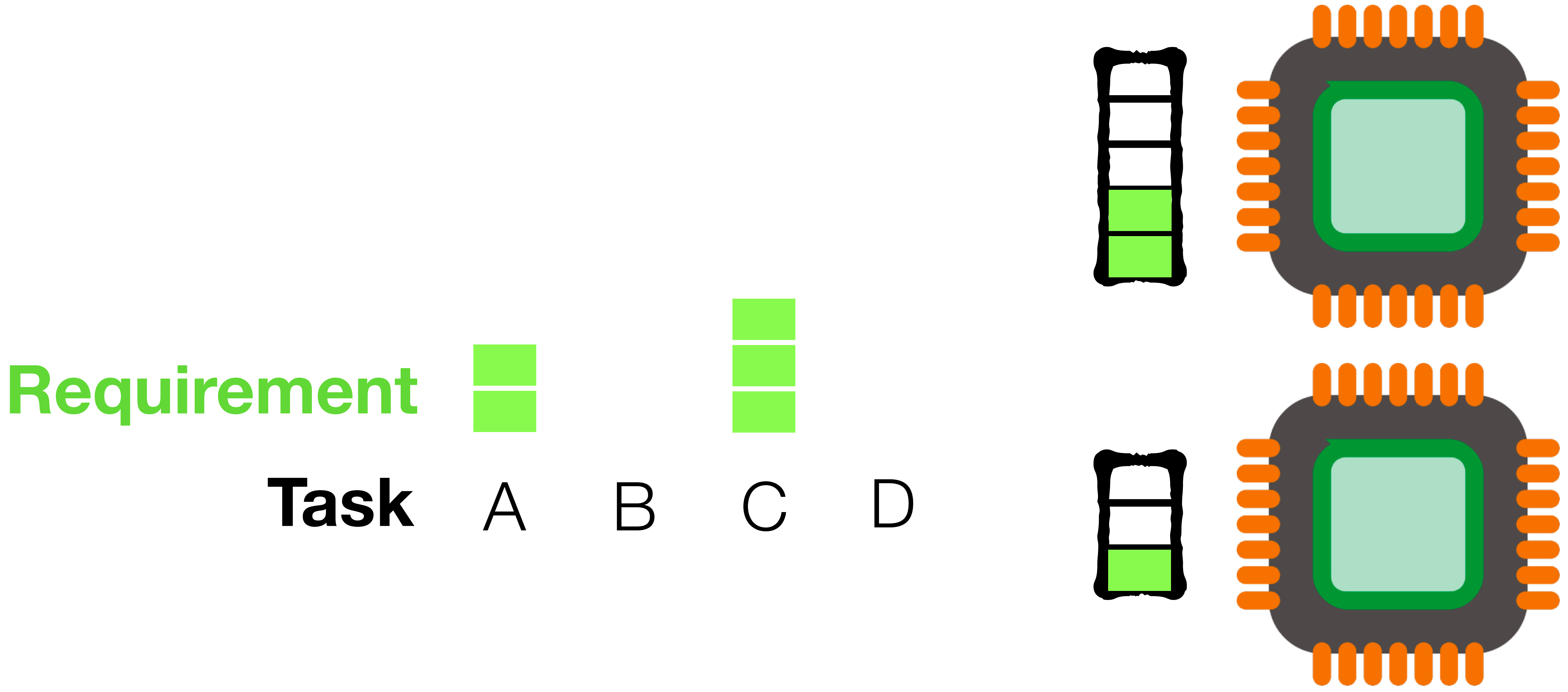
1 *What **type of algorithm** is $\mathcal{A}$?*



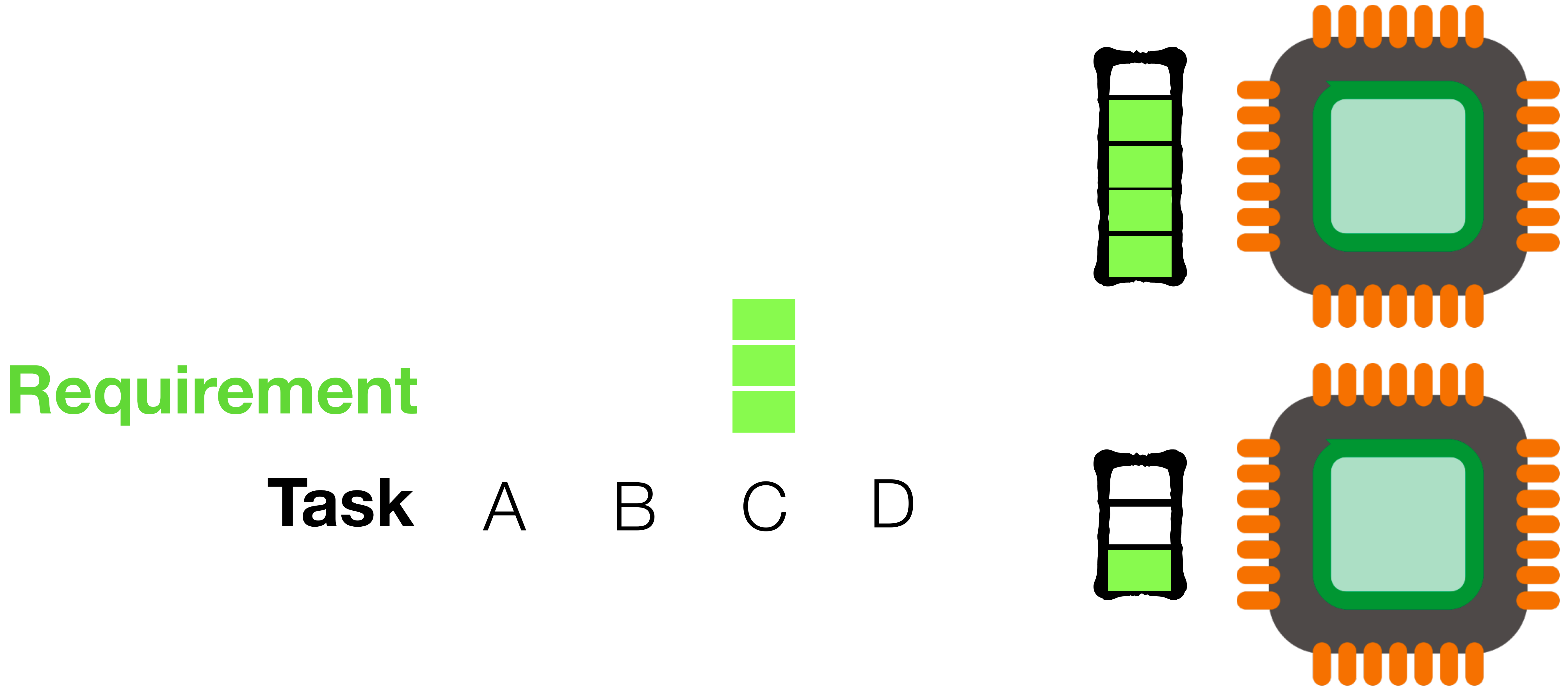
1 *What **type of algorithm** is $\mathcal{A}$?*



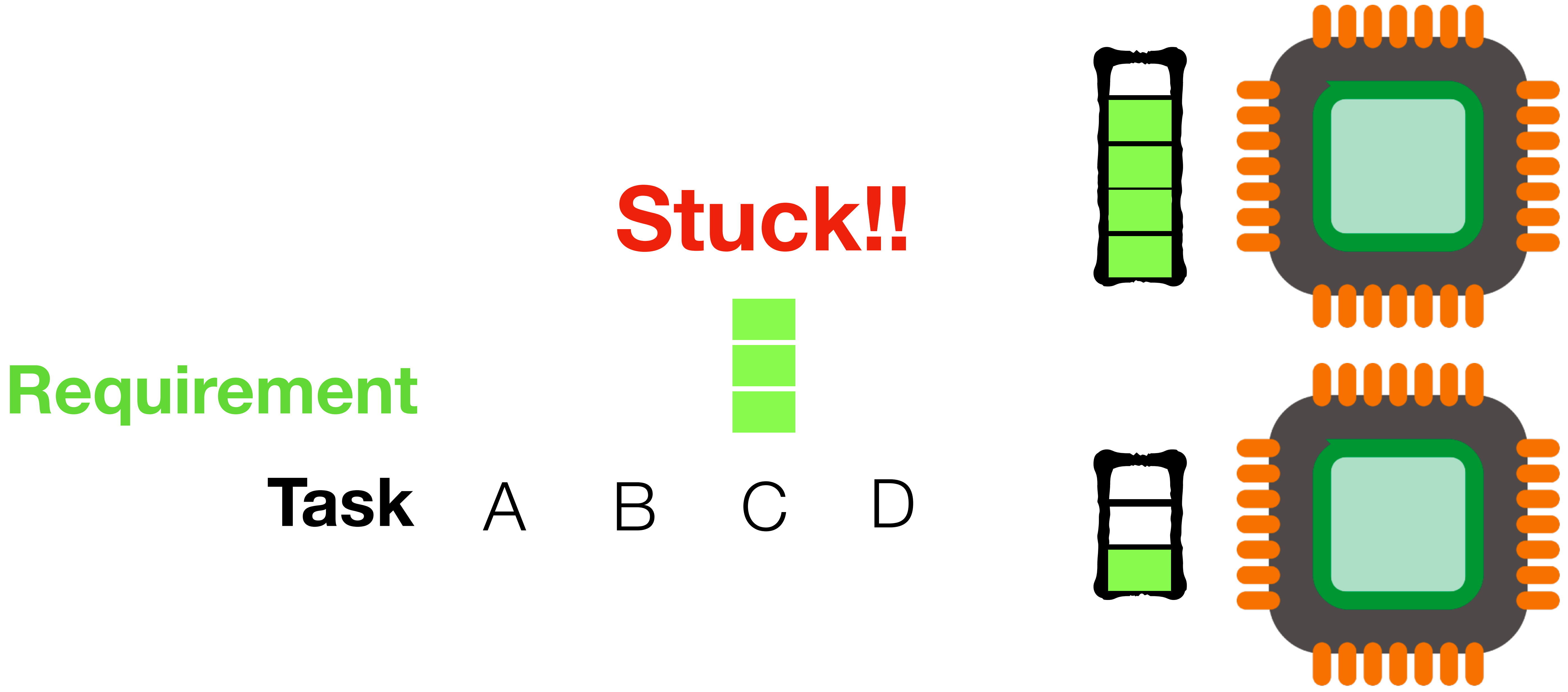
1 *What **type of algorithm** is $\mathcal{A}$?*



1 What **type of algorithm** is $\mathcal{A}$?

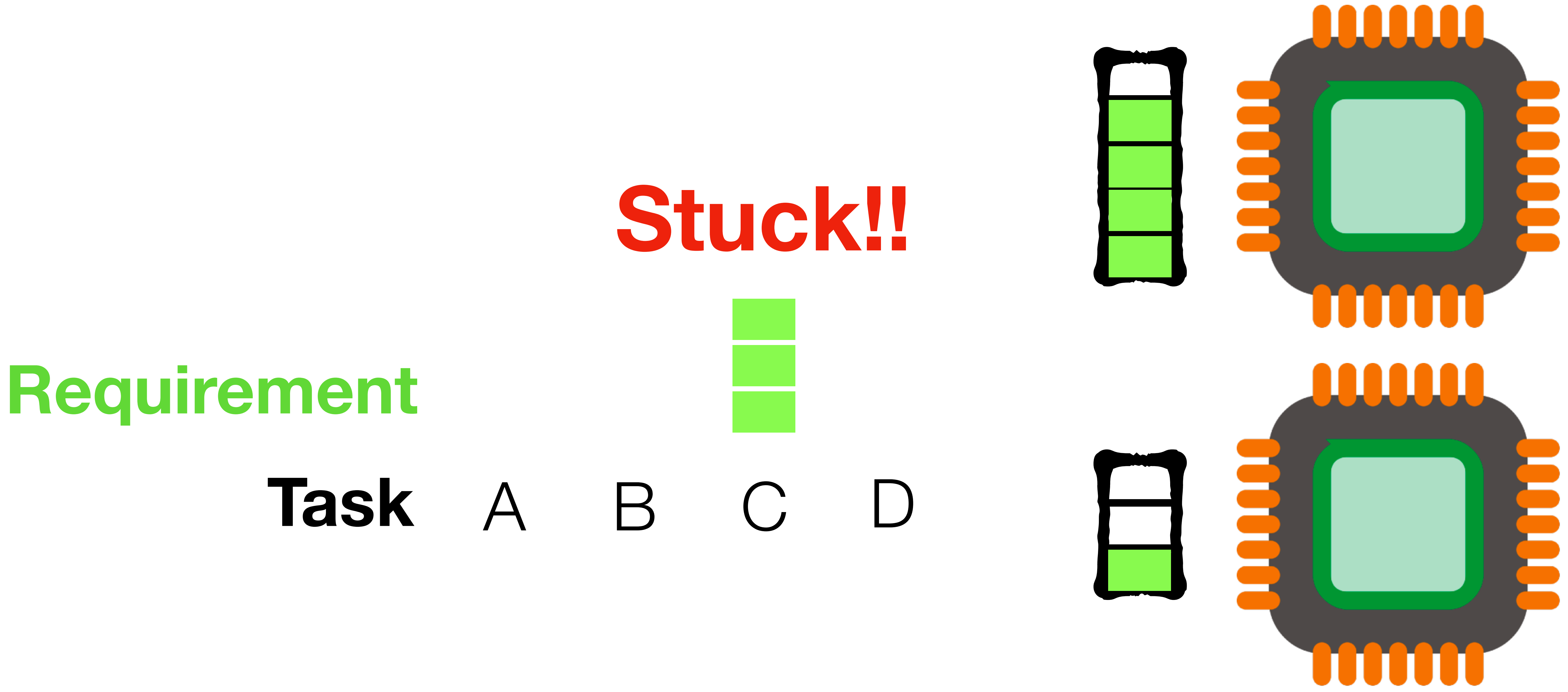


1 *What* ***type of algorithm*** *is* $\mathcal{A}$ *?*



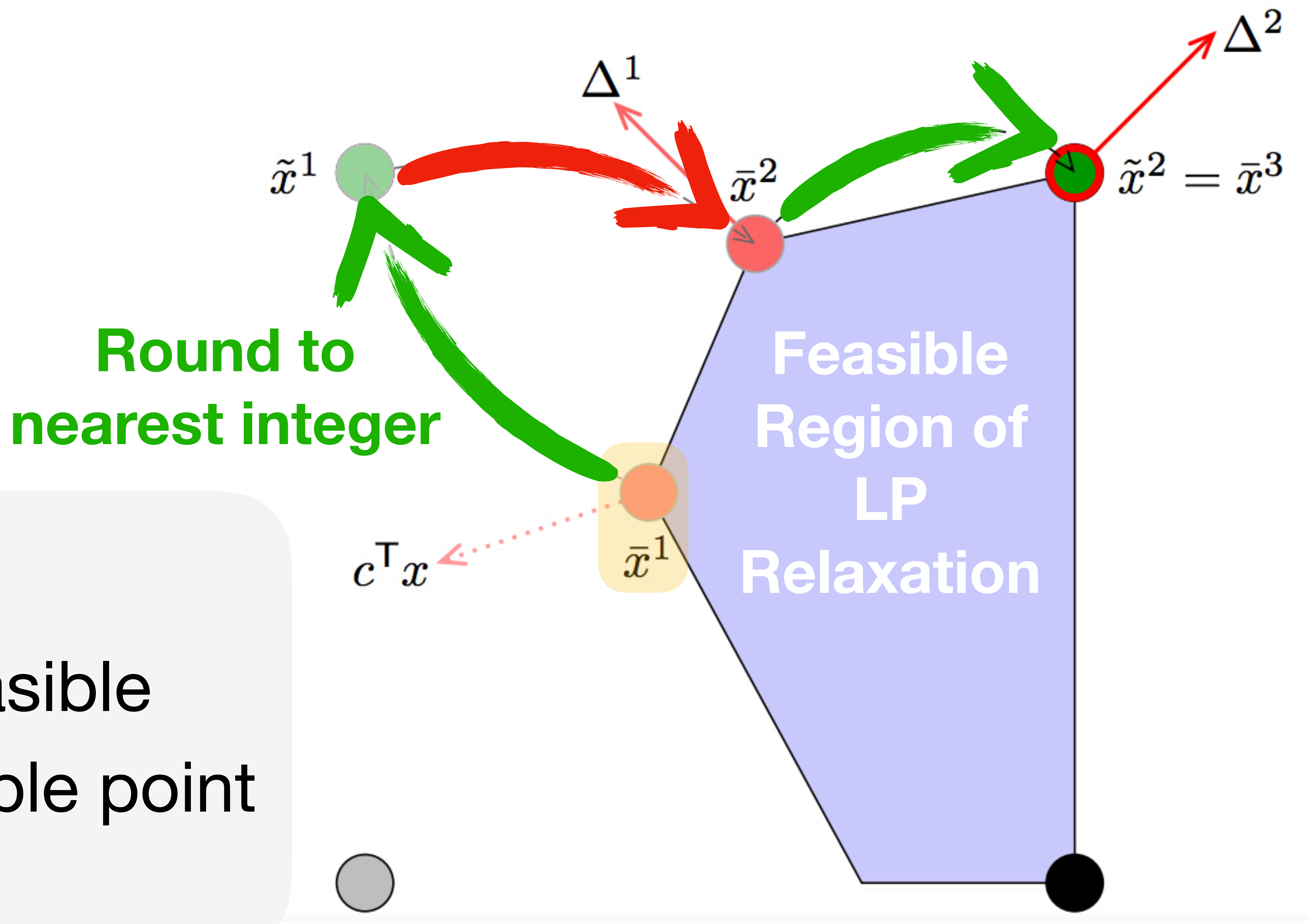
1 *What **type of algorithm** is $\mathcal{A}$?*

Local algorithms may **fail** in the presence of **hard constraints**



1 *What **type of algorithm** is $\mathcal{A}$?*

Repeated Projections
maintain constraint feasibility via LP solving



0 Start with LP-feasible (fractional) solution

1 Round to nearest integer, **return** if LP-feasible

2 Project integer point to **nearest** LP-feasible point

3 Go back to step 1

Figure in part from Berthold (2014)

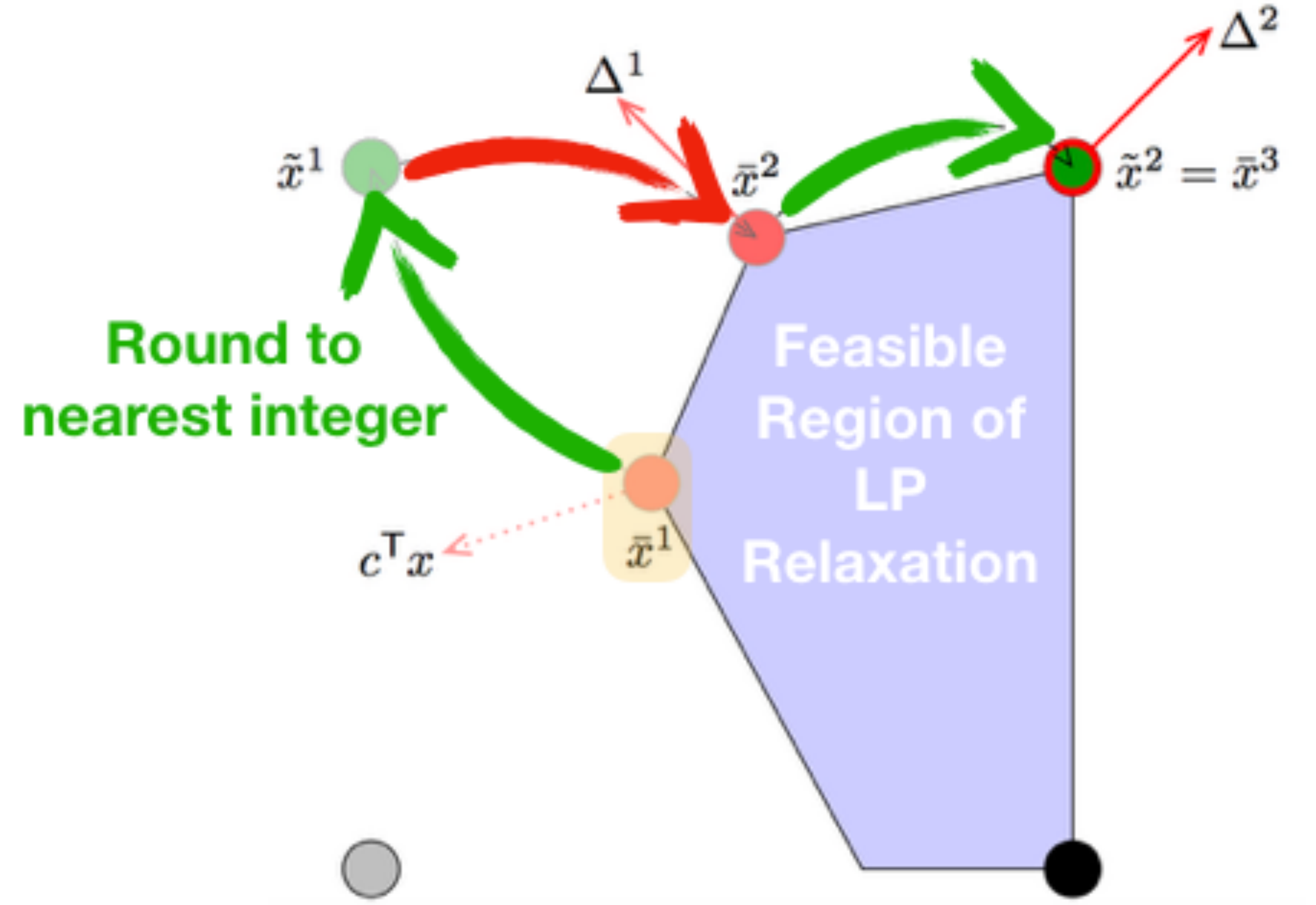
2 *What is the **role of ML** in the algorithm?*

0 Start with LP-feasible (fractional) solution

1 **Round** to nearest integer, **return** if LP-feasible

iterate **2** **Project** integer point to **nearest** LP-feasible point

3 Go back to step 1

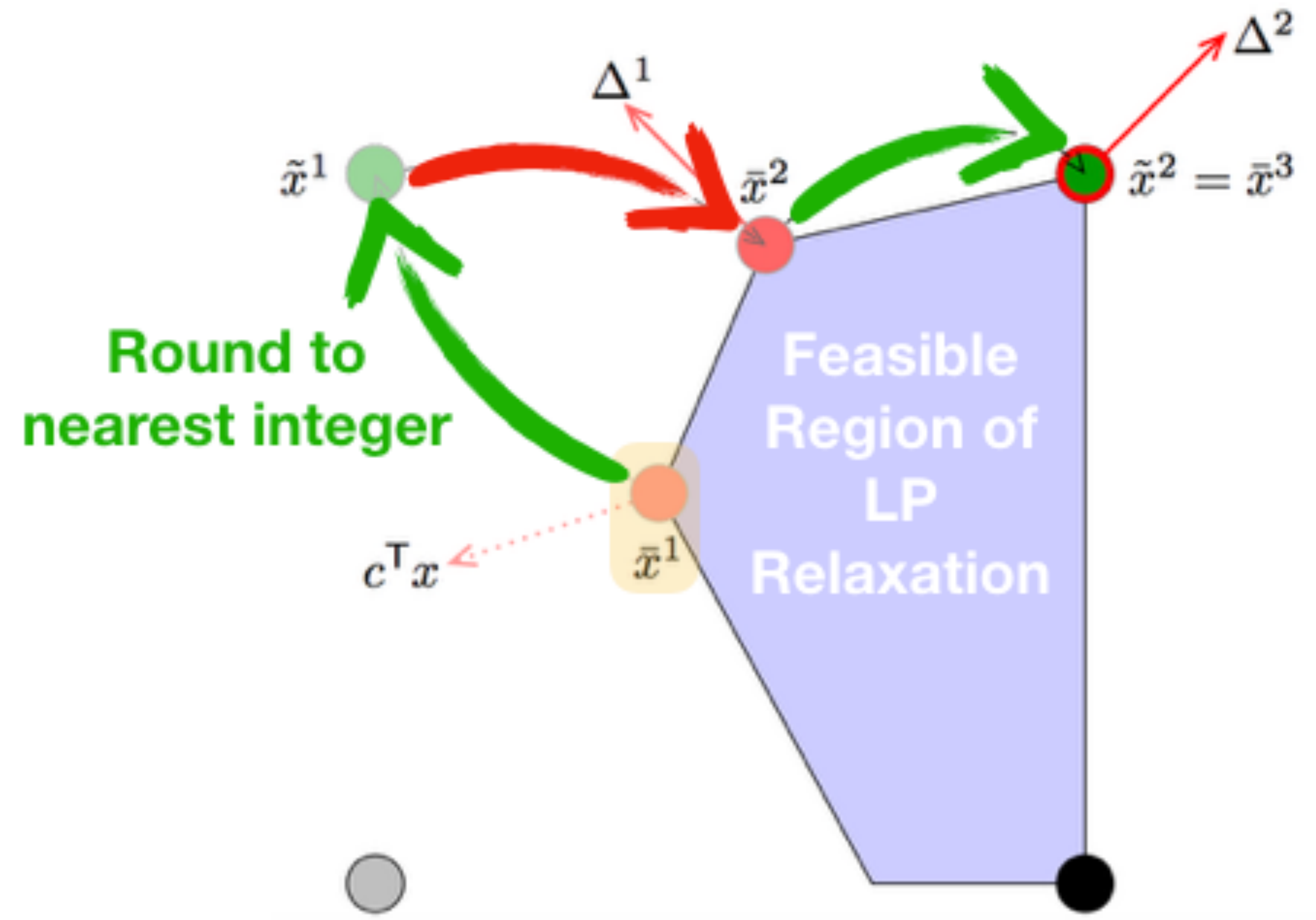


2 *What is the **role of ML** in the algorithm?*

$\bar{x}^1$

- **0** Start with LP-feasible (fractional) solution
- **1** **Round** to nearest integer, **return** if LP-feasible
- **2** **Project** integer point to **nearest** LP-feasible point
- **3** Go back to step 1

iterate



2 What is the ***role of ML*** in the algorithm?

0 Start with LP-feasible (fractional) solution

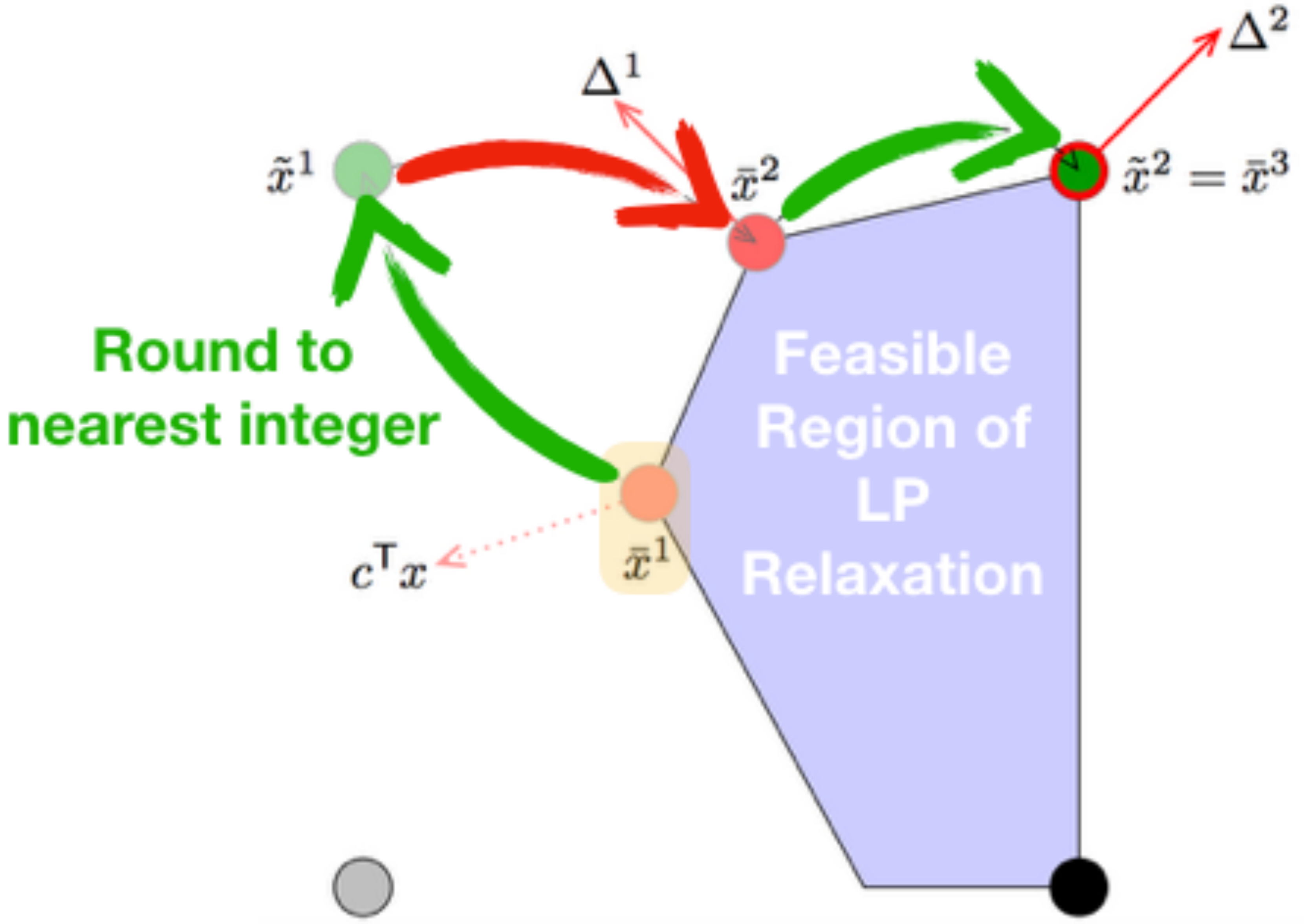
1 **Round** to nearest integer, **return** if LP-feasible

2 **Project** integer point to **nearest** LP-feasible point

3 Go back to step 1

iterate

$\bar{x}^1$ $[\bar{x}^1]$



2 *What is the **role of ML** in the algorithm?*

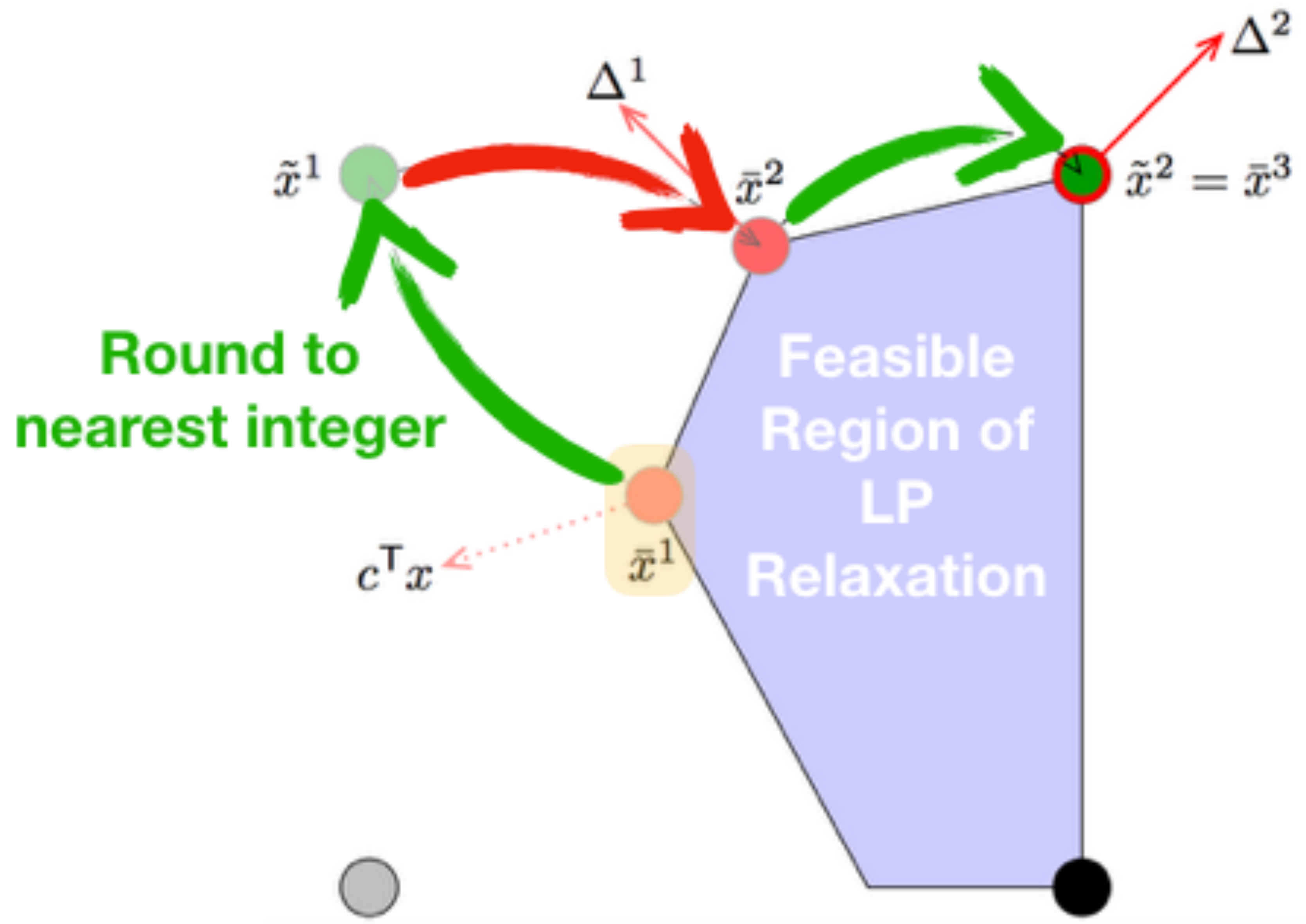
0 Start with LP-feasible (fractional) solution $\bar{x}^1$

1 **Round** to nearest integer, **return** if LP-feasible $[\bar{x}^1]$

iterate

2 **Project** integer point to **nearest** LP-feasible point $\bar{x}^2$

3 Go back to step 1



2 *What is the **role of ML** in the algorithm?*

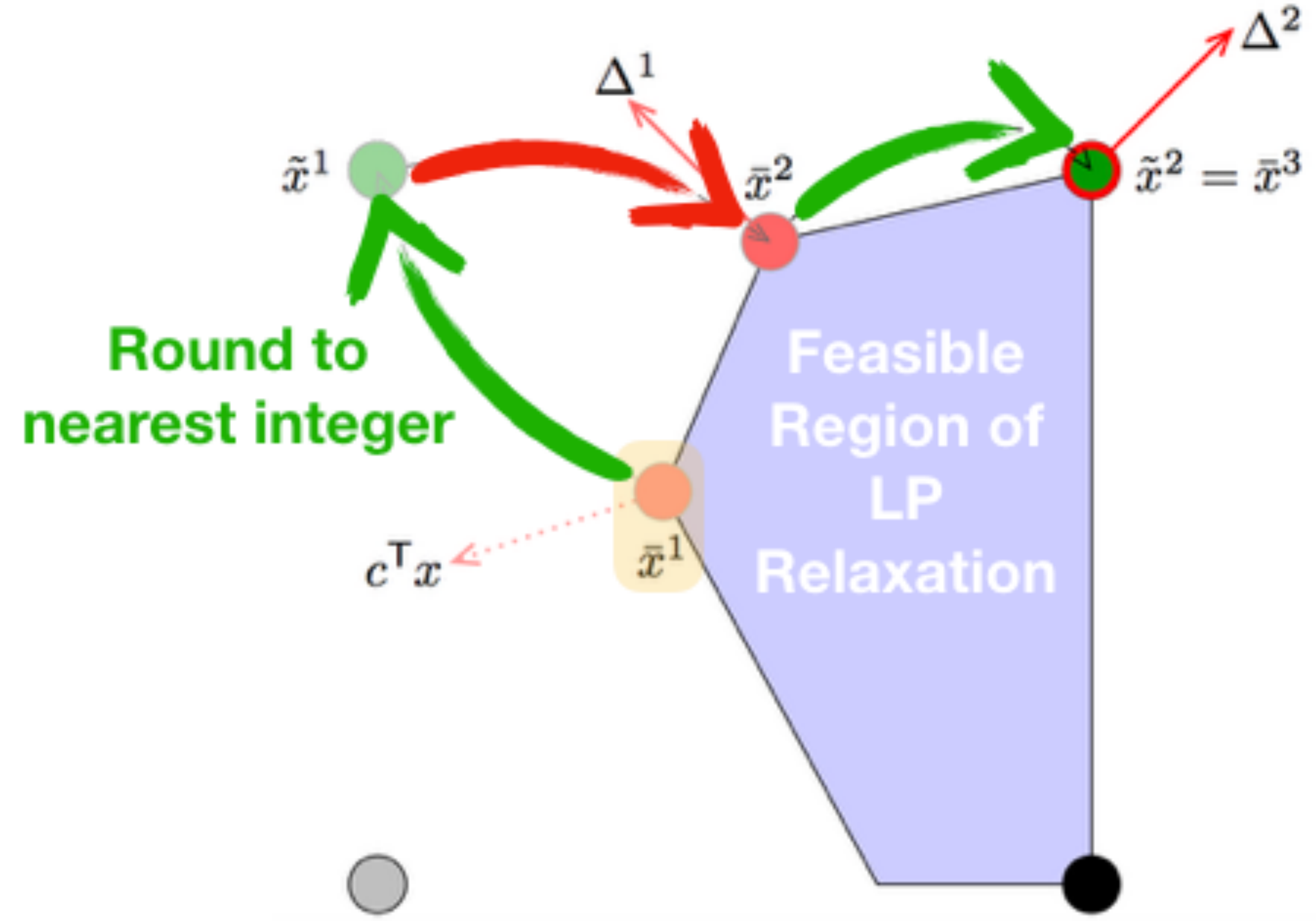
0 Start with LP-feasible (fractional) solution $\bar{x}^1$

1 **Round** to nearest integer, **return** if LP-feasible $[\bar{x}^1]$

2 **Project** integer point to **nearest** LP-feasible point $\bar{x}^2$

3 Go back to step 1

iterate



2 What is the ***role of ML*** in the algorithm?

iterate

0 Start with LP-feasible (fractional) solution $\bar{x}^1$

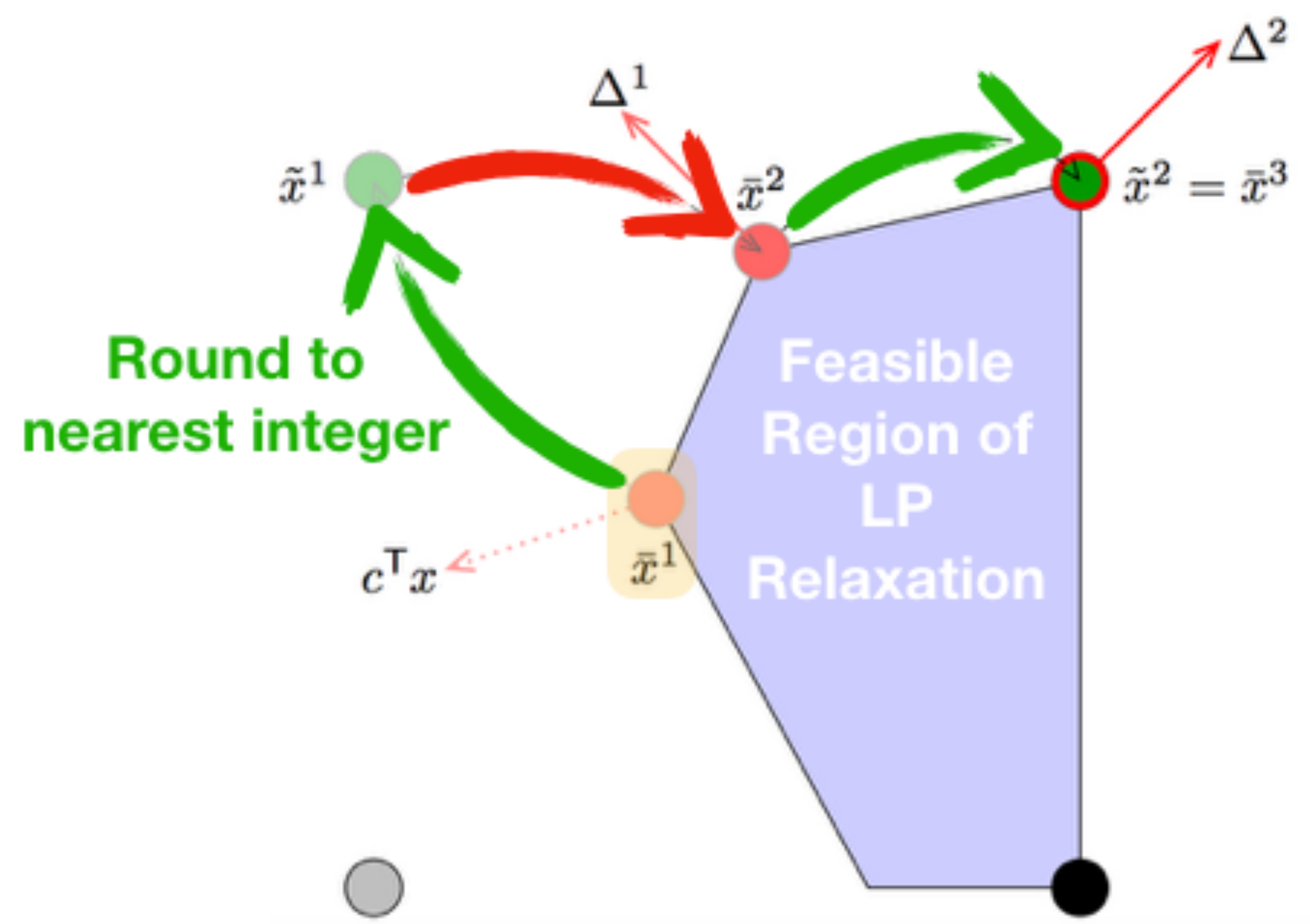
1 **Round** to nearest integer, **return** if LP-feasible $[\bar{x}^1]$

2 **Project** integer point to **nearest** LP-feasible point $\bar{x}^2$

3 Go back to step 1

Key Step:

$$\min_x \Delta(x, [\bar{x}^t])$$
$$\text{s.t. } Ax \leq b,$$
$$x \in [0,1]^n$$



2 *What is the **role of ML** in the algorithm?*

0 Start with LP-feasible (fractional) solution $\bar{x}^1$

1 **Round** to nearest integer, **return** if LP-feasible $[\bar{x}^1]$

2 **Project** integer point to **nearest** LP-feasible point $\bar{x}^2$

3 Go back to step 1

iterate

$$\min_x \Delta(x, [\bar{x}^t])$$

$$\text{s.t. } Ax \leq b,$$

$$x \in [0,1]^n$$

2 What is the ***role of ML*** in the algorithm?

0 Start with LP-feasible (fractional) solution $\bar{x}^1$

1 **Round** to nearest integer, **return** if LP-feasible $[\bar{x}^1]$

2 **Project** integer point to **nearest** LP-feasible point $\bar{x}^2$

3 Go back to step 1

iterate

L1-distance

$$\min_x \Delta(x, [\bar{x}^t])$$
$$\text{s.t. } Ax \leq b,$$
$$x \in [0,1]^n$$

2 *What is the **role of ML** in the algorithm?*

0 Start with LP-feasible (fractional) solution $\bar{x}^1$

iterate

1 **Round** to nearest integer, **return** if LP-feasible $[\bar{x}^1]$

2 **Project** integer point to **nearest** LP-feasible point $\bar{x}^2$

3 Go back to step 1

$$\min_x \Delta(x, [\bar{x}^t])$$
$$\text{s.t. } Ax \leq b,$$
$$x \in [0,1]^n$$

L1-distance

$$\sum_j \left| x_j - [\bar{x}^t]_j \right|$$
$$= \sum_{j:[\bar{x}^t]=0} x_j + \sum_{j:[\bar{x}^t]=1} (1 - x_j)$$

2 *What is the **role of ML** in the algorithm?*

0 Start with LP-feasible (fractional) solution

1 Round to nearest integer, **return** if LP-feasible

iterate

2 Project integer point to **nearest** LP-feasible point

3 Go back to step 1

$\bar{x}^1$ $[\bar{x}^1]$

$\bar{x}^2$

L1-distance $\sum_{j:[\bar{x}^t]=0} x_j + \sum_{j:[\bar{x}^t]=1} (1 - x_j)$

$$\min_x \Delta(x, [\bar{x}^t])$$
$$\text{s.t. } Ax \leq b,$$
$$x \in [0,1]^n$$

2 *What is the **role of ML** in the algorithm?*

iterate

0 Start with LP-feasible (fractional) solution $\bar{x}^1$

1 **Round** to nearest integer, **return** if LP-feasible $[\bar{x}^1]$

2 **Project** integer point to **nearest** LP-feasible point $\bar{x}^2$

3 Go back to step 1

~~L1-distance $\sum_{j:[\bar{x}^t]=0} x_j + \sum_{j:[\bar{x}^t]=1} (1-x_j)$~~

$$\min_x \Delta(x, [\bar{x}^t])$$
$$\text{s.t. } Ax \le b,$$
$$x \in [0,1]^n$$

2 *What is the **role of ML** in the algorithm?*

iterate

0 Start with LP-feasible (fractional) solution

1 **Round** to nearest integer, **return** if LP-feasible

2 **Project** integer point to **nearest** LP-feasible point

3 Go back to step 1

$\bar{x}^1$ $\bar{x}^2$ $[\bar{x}^1]$

~~**L1-distance** $\sum_{j:[\bar{x}^t]=0} x_j + \sum_{j:[\bar{x}^t]=1} (1 - x_j)$~~

$$\min_x \Delta(x, [\bar{x}^t])$$
$$\text{s.t. } Ax \leq b,$$
$$x \in [0,1]^n$$

Learn the projection coefficients!!

2 *What is the **role of ML** in the algorithm?*

$$\min_x \ \ell_1(x, [\bar{x}^t])$$
$$\text{s.t. } Ax \leq b,$$
$$x \in [0, 1]^n$$

2 *What is the **role of ML** in the algorithm?*

$$\min_x \ \ell_1(x, [\bar{x}^t])$$
$$\text{s.t. } Ax \le b,$$
$$x \in [0,1]^n$$

2 *What is the **role of ML** in the algorithm?*

$$\min_x \sout{\ell_1(x, [\bar{x}^t])}$$

$$\text{s.t. } Ax \leq b,$$

$$x \in [0,1]^n$$

2 *What is the* ***role of ML*** *in the algorithm?*

$$\mathbf{p}^\intercal x$$

$$\min_x \sout{\ell_1(x, [\bar{x}^t])}$$

$$\text{s.t. } Ax \leq b,$$

$$x \in [0,1]^n$$

2 *What is the* ***role of ML*** *in the algorithm?*

$$\mathbf{p}^\intercal x$$

$$\min_x \; \sout{\ell_\intercal(x, [\bar{x}^t])}$$

$$\text{s.t. } Ax \leq b,$$

$$x \in [0,1]^n$$

$$\mathbf{p}_i = \texttt{model}\left(\bar{x}_i{}^t, [\bar{x}_i{}^t]; \Theta\right)$$

2 *What is the **role of ML** in the algorithm?*

$\mathbf{p}^\intercal x$

$$\min_x \ \ell_1(x, [\bar{x}^t])$$

$$\text{s.t. } Ax \leq b,$$

$$x \in [0,1]^n$$

$$\mathbf{p}_i = \texttt{model}\left(\bar{x}_i{}^t, [\bar{x}_i{}^t]; \Theta\right)$$

Properties of `model`

2 *What is the **role of ML** in the algorithm?*

$\mathbf{p}^\intercal x$

$$\min_x \; \ell_1(x, [\bar{x}^t])$$

$$\text{s.t. } Ax \le b,$$

$$x \in [0,1]^n$$

$$\mathbf{p}_i = \texttt{model}\left(\bar{x}_i{}^t, [\bar{x}_i{}^t]; \Theta\right)$$

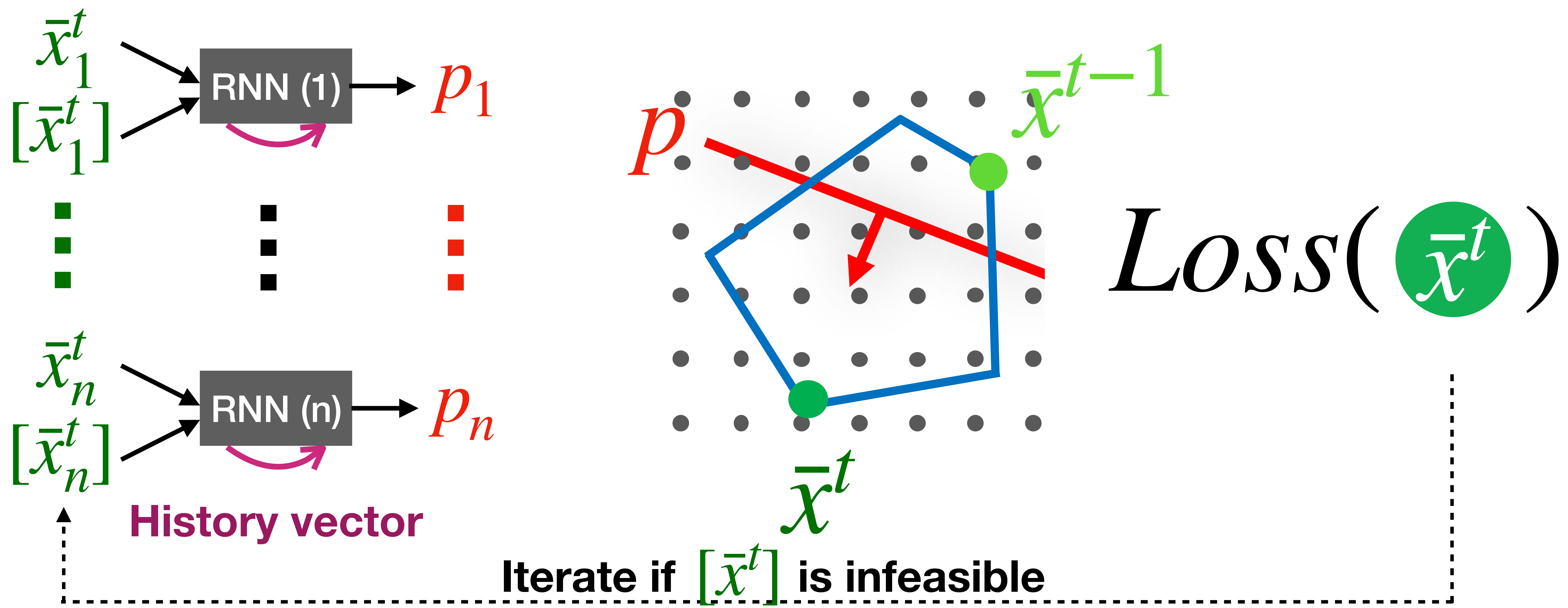
Properties of `model`

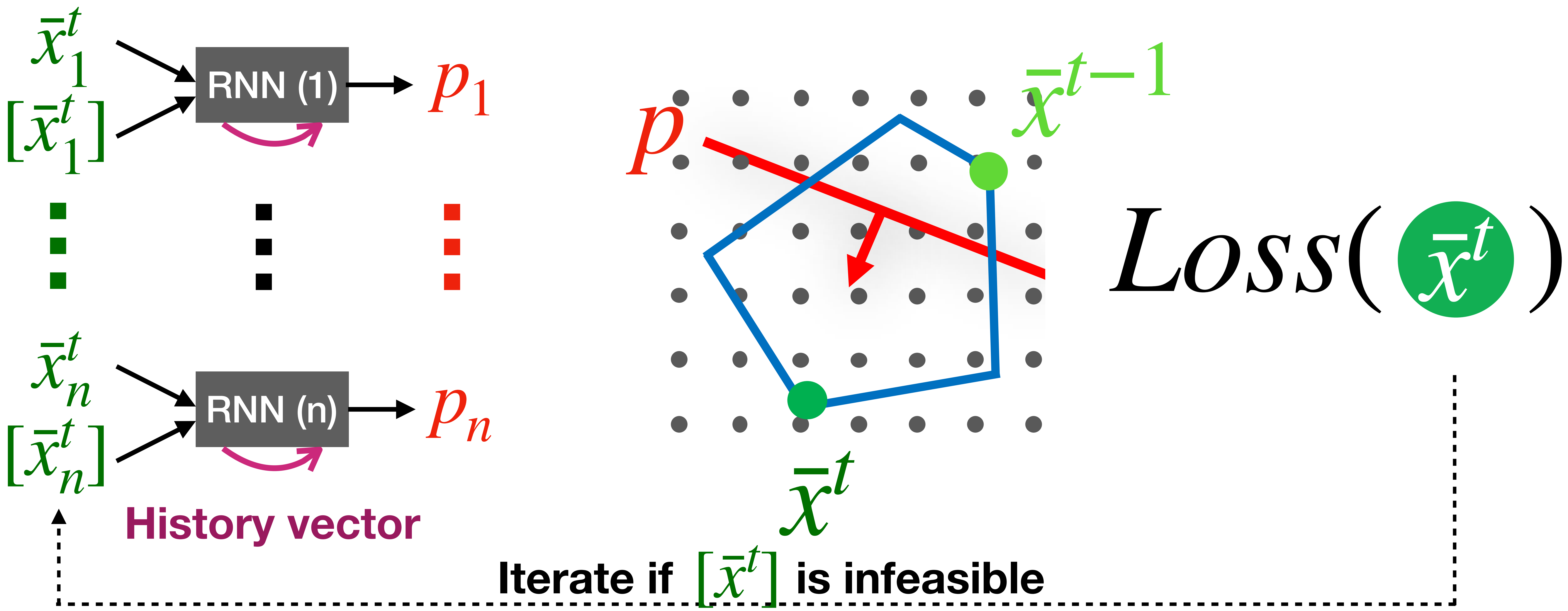
- **Parameters shared** across variables
- **Recurrent** across iterations

3 How can we ***train*** the algorithm?

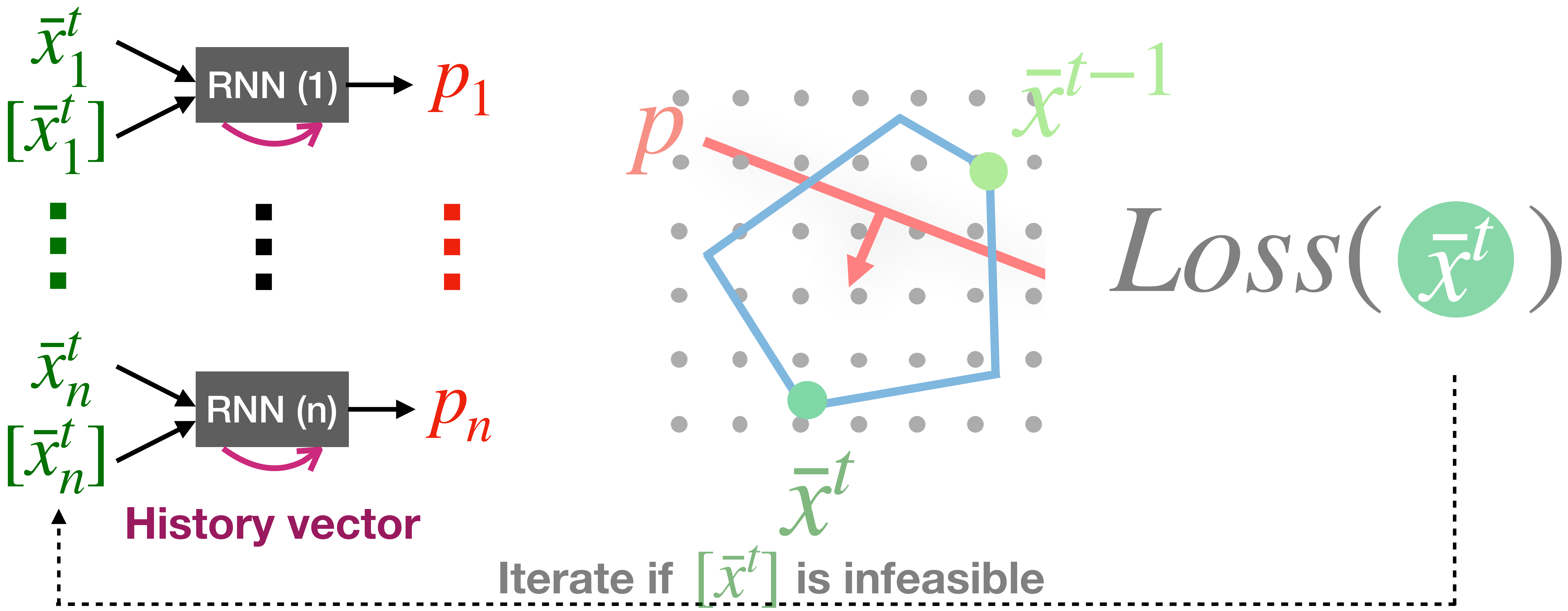
$$\min_x c^T x \text{ s.t. } Ax \le b, x \in \{0,1\}^n$$

input to Recurrent Neural Network → **predict** projection coefficients → **solve LP** projection → **penalize** fractional variables





input to Recurrent Neural Network → **predict** projection coefficients → **solve LP** projection → **penalize** fractional variables



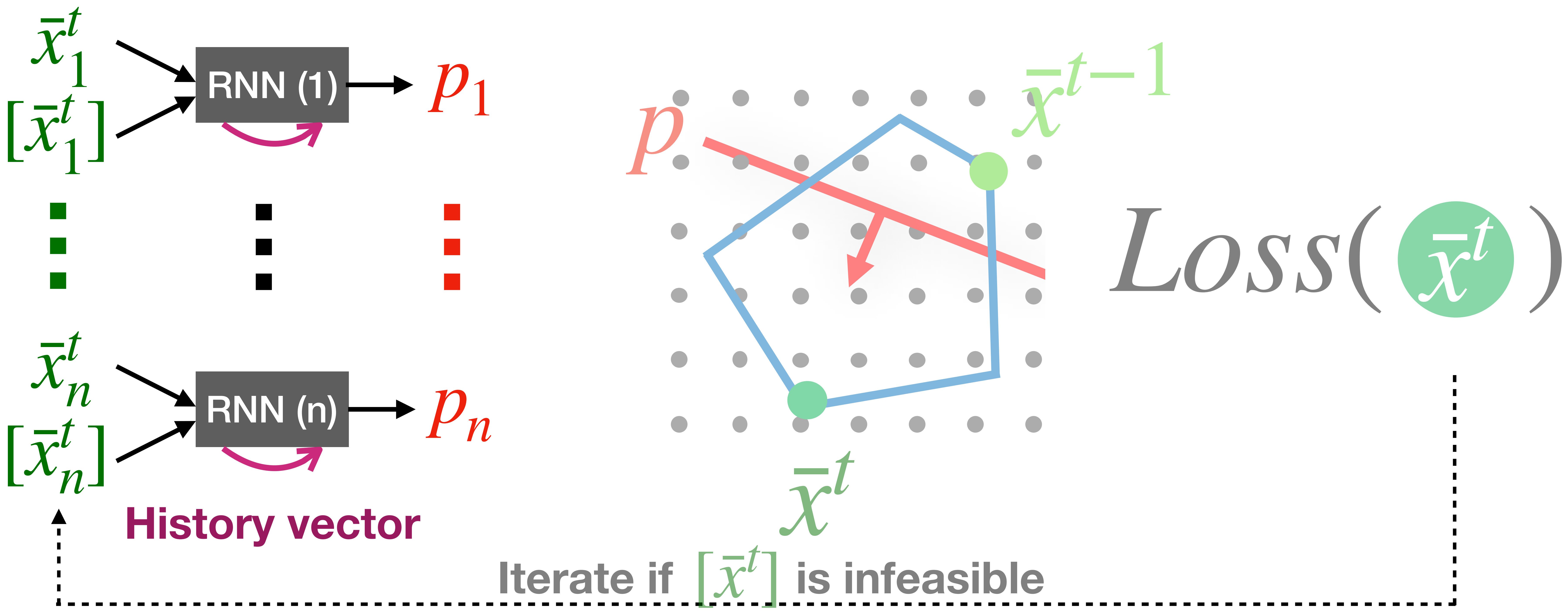
input to Recurrent Neural Network → **predict** projection coefficients → **solve LP** projection → **penalize** fractional variables

RNN

A neural network with parameters Θ

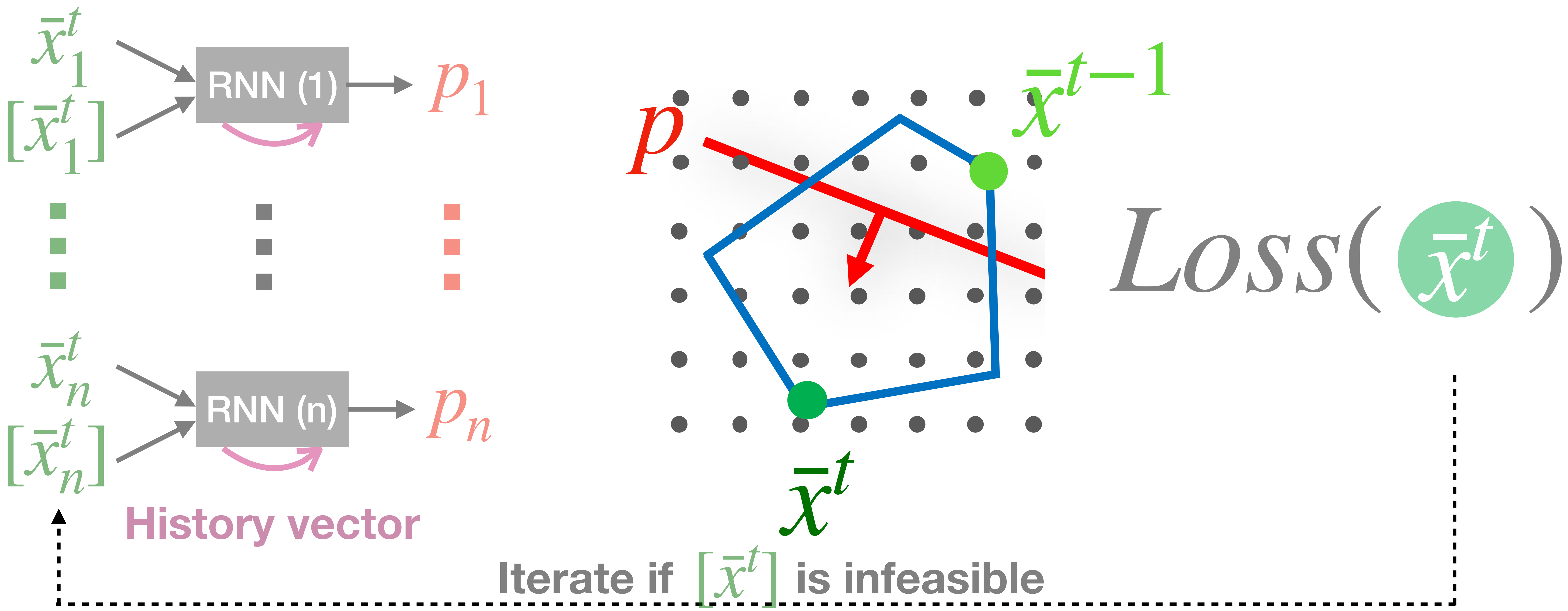
Same network used for all fractional variables

History vector is variable-specific



input to Recurrent Neural Network → **predict** projection coefficients → **solve LP** projection → **penalize** fractional variables

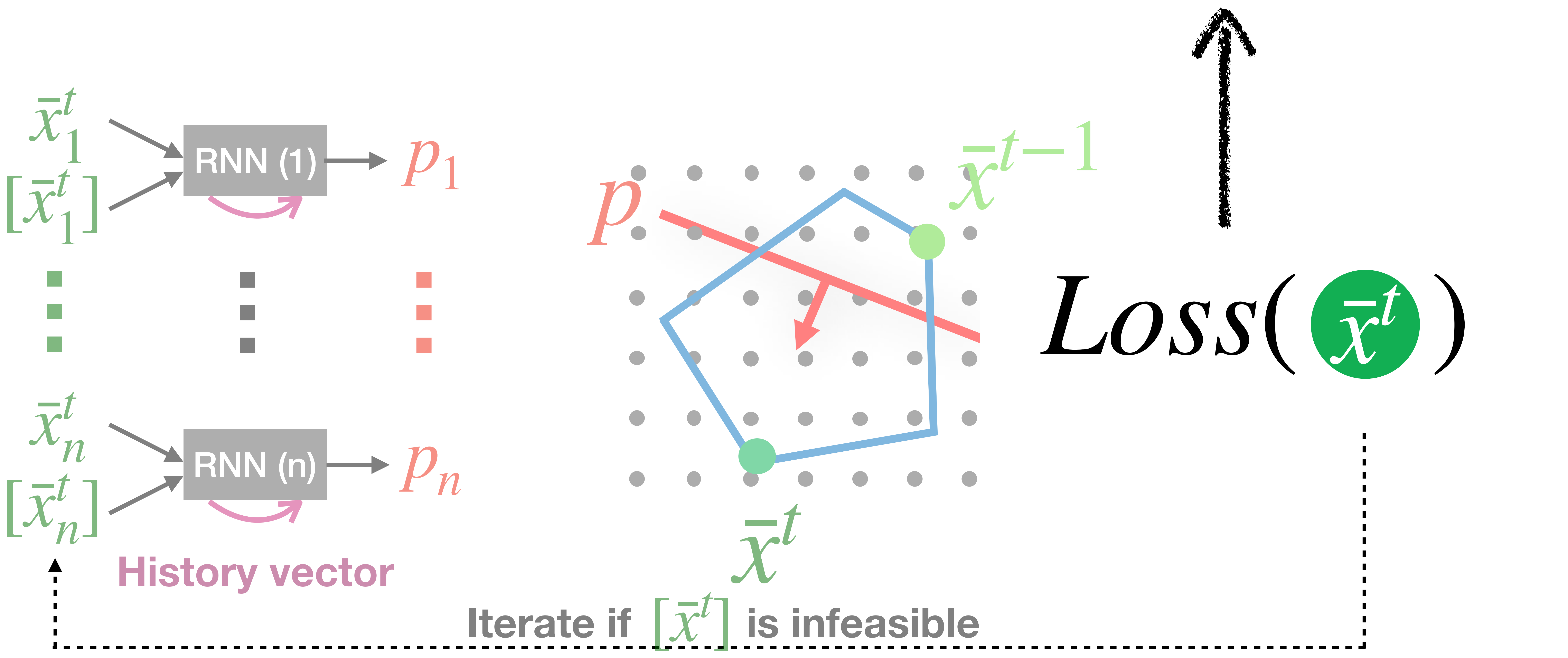
To make LP solution differentiable,
add small constant quadratic term
See OptNet by Amos & Kolter, 2017



input to Recurrent Neural Network → **predict** projection coefficients → **solve LP** projection → **penalize** fractional variables

Binary Cross-Entropy Loss

$$-[\bar{x}_j^t] \cdot \log \bar{x}_j^t + (1 - [\bar{x}_j^t]) \cdot \log (1 - \bar{x}_j^t)$$



input to Recurrent Neural Network → **predict** projection coefficients → **solve LP** projection → **penalize** fractional variables

Experimental Setup

- Generate Training / Validation / Testing instances
  - **No need to solve Training instances!**

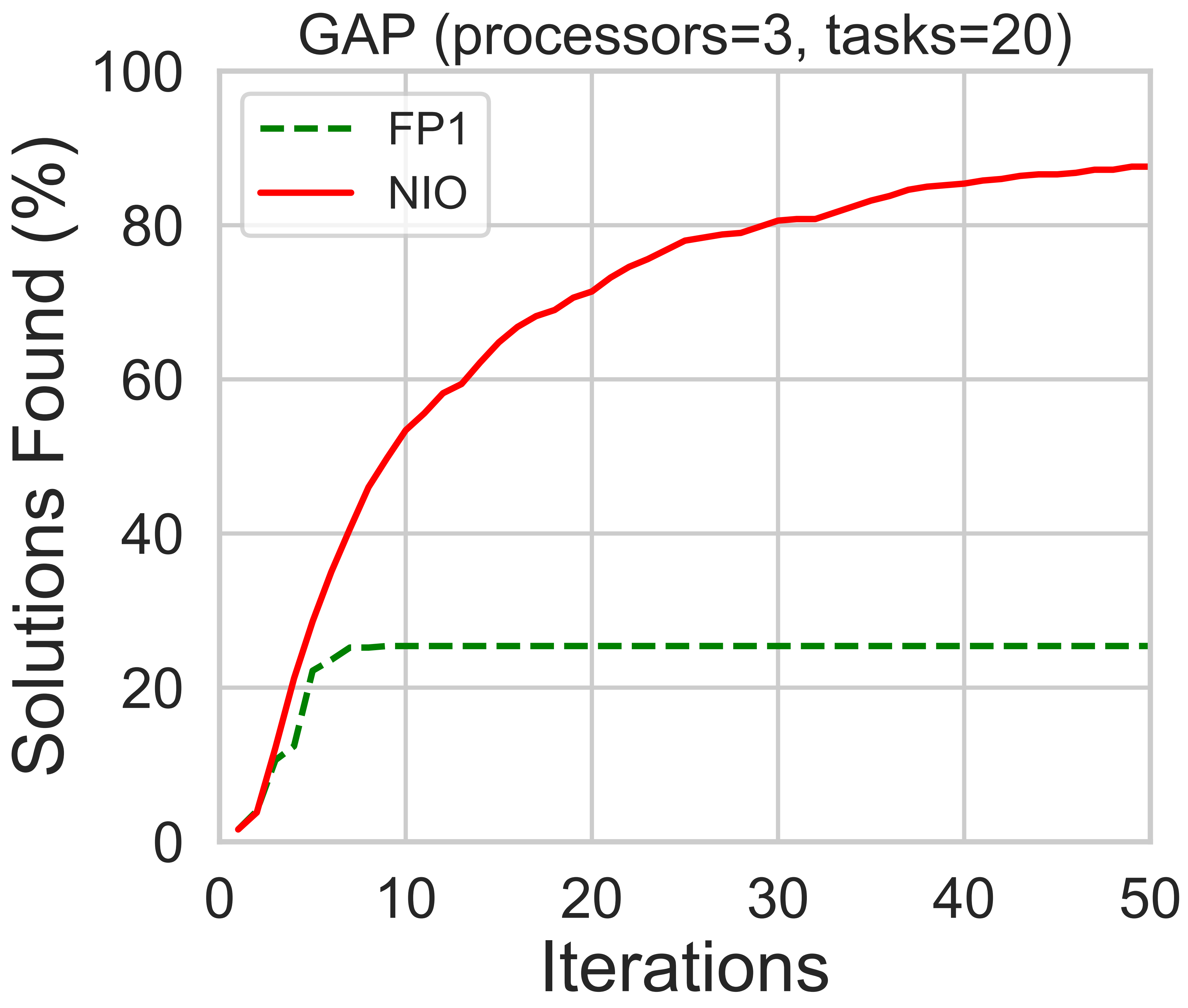
- NIO is **fully differentiable**
  - Train with gradient descent

$$
\begin{aligned}
\underset{x}{\text{maximize}} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{n} w_{ij} x_{ij} \leq c_i, \quad i = 1, \ldots, m, \\
& \sum_{i=1}^{m} x_{ij} = 1, \quad j = 1, \ldots, n, \\
& x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, m, j = 1, \ldots, n.
\end{aligned}
$$

Learning IP Heuristics in Practice

Generalized Assignment Problem (GAP)

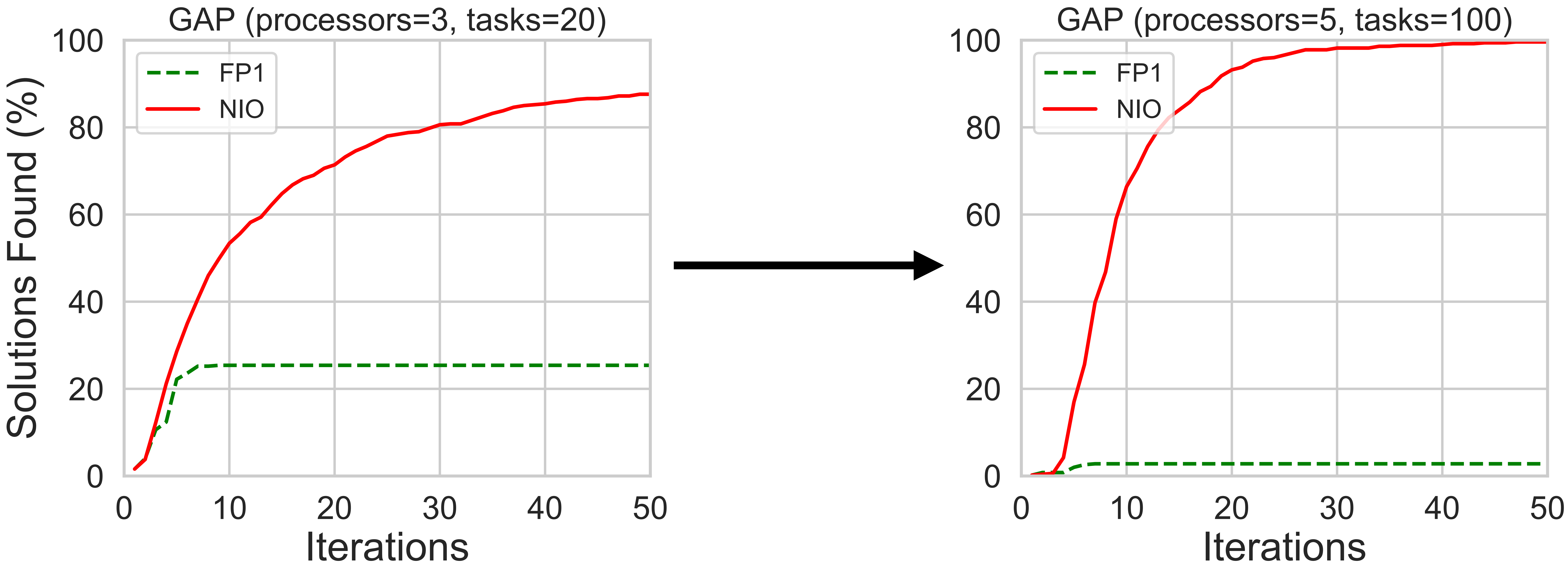
Train on 500 **small** instances, **Test on** 500 **larger** instances



Learning IP Heuristics in Practice

Generalized Assignment Problem (GAP)

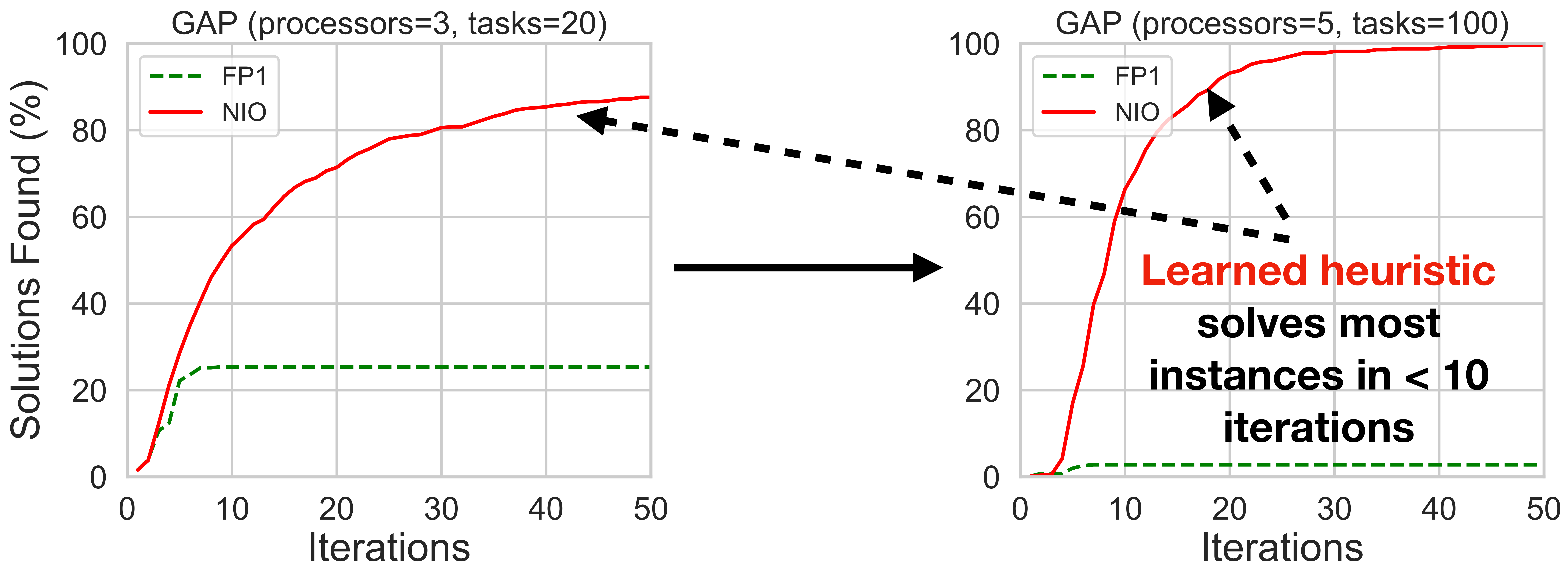
Train on 500 **small** instances, **Test on** 500 **larger** instances



Learning IP Heuristics in Practice

Generalized Assignment Problem (GAP)

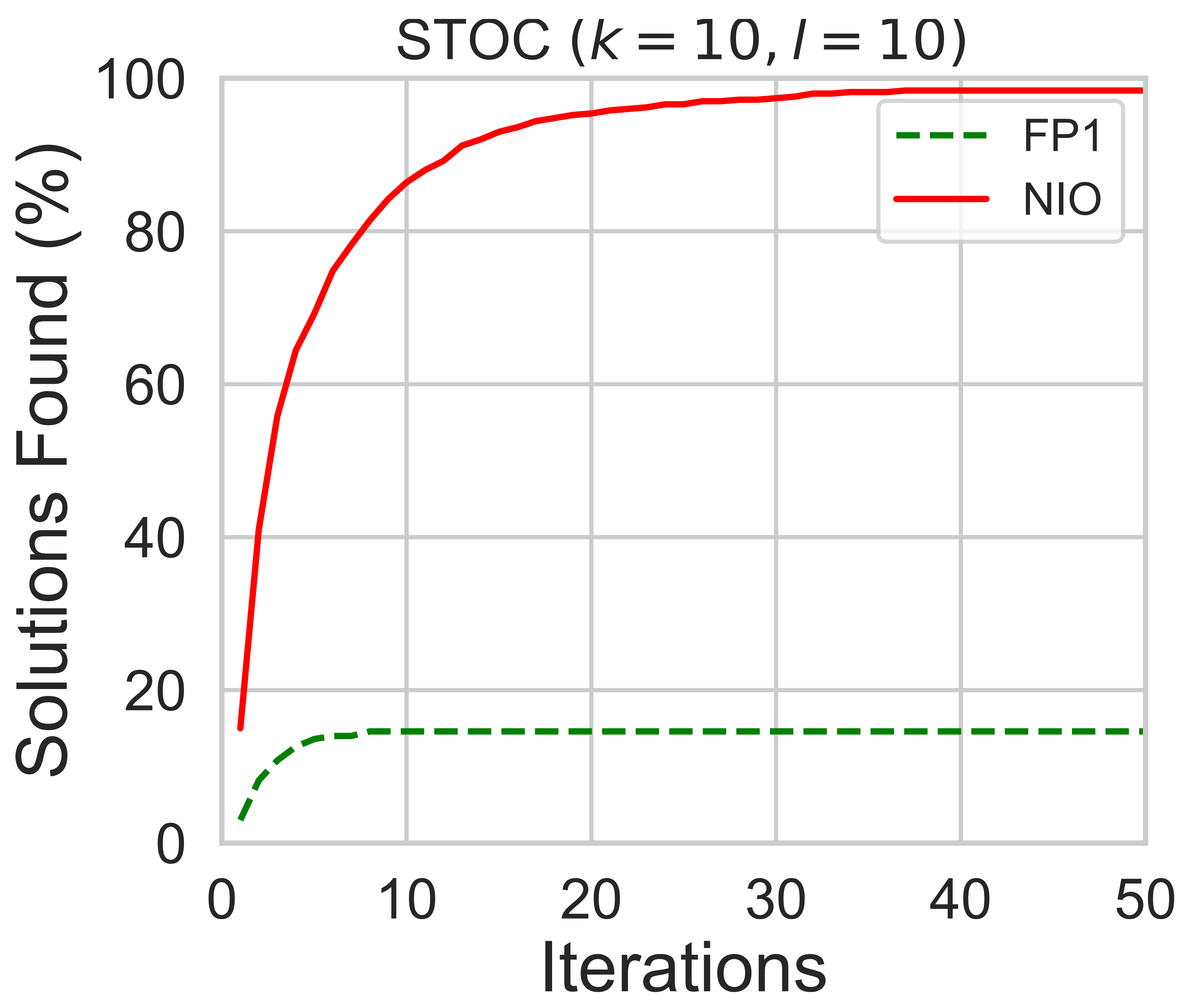
Train on 500 **small** instances, **Test on** 500 **larger** instances

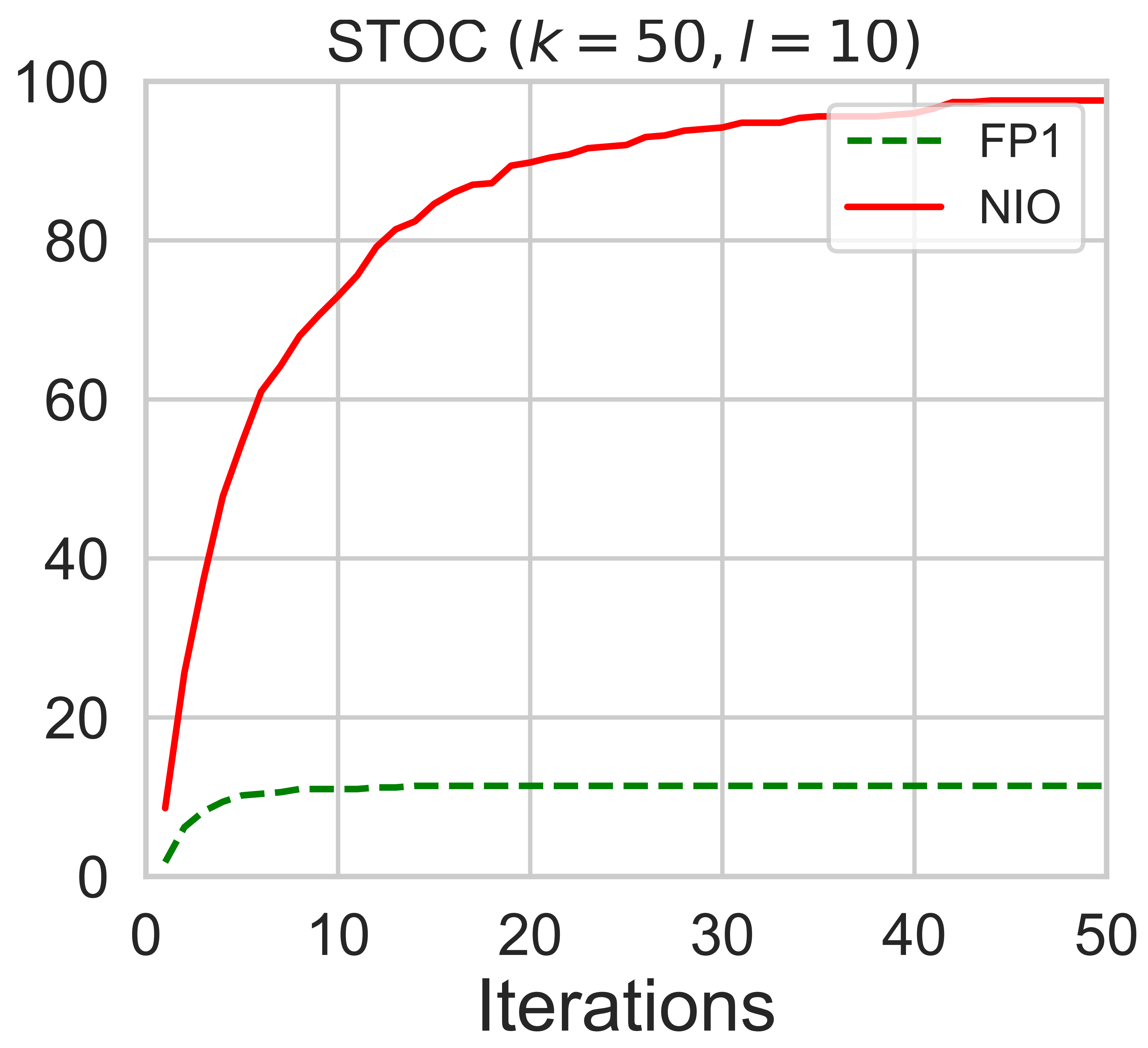


Learning IP Heuristics in Practice

Two-Stage Stochastic Integer Programs (STOC)

Train on 500 **small** instances, **Test on** 500 **larger** instances

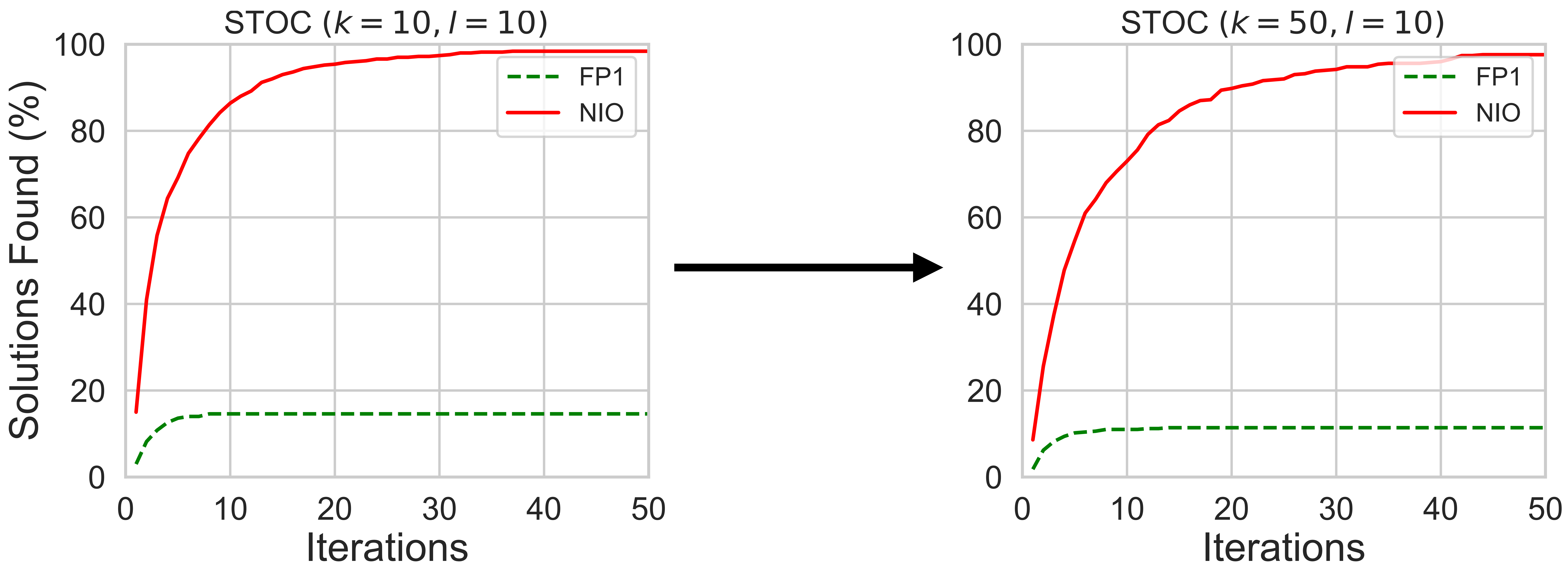




Learning IP Heuristics in Practice

Two-Stage Stochastic Integer Programs (STOC)

Train on 500 **small** instances, **Test on** 500 **larger** instances



What about advanced codes?

Two-Stage Stochastic Integer Programs (STOC)

FP1 + **presolve** + **propagation**

	NIO	FP2	FP1
STOC (10,10)	**99.2**	95.4	14.6
STOC (20,20)	**22.6**	0.6	0
STOC (30,20)	**7.6**	0	0

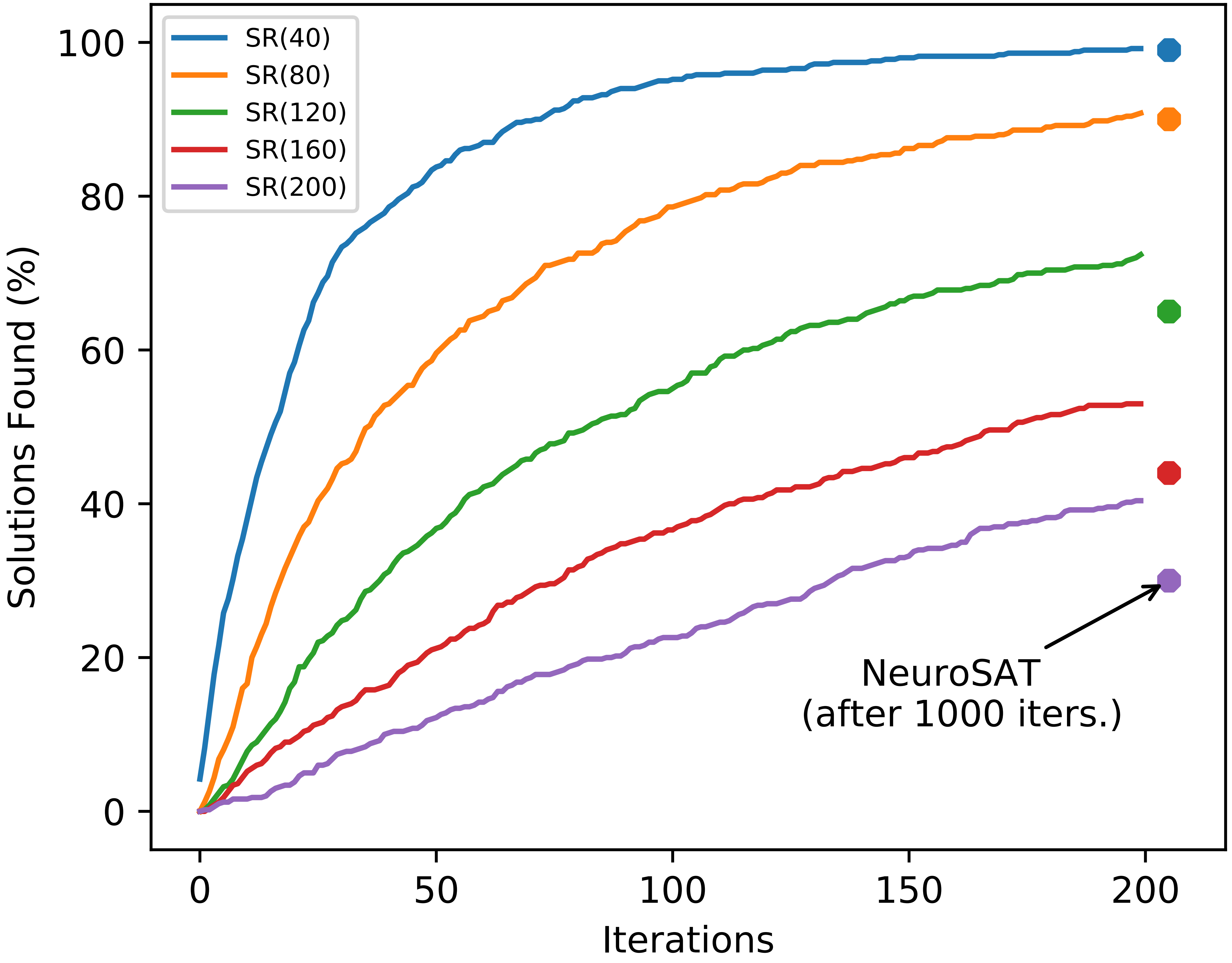
Solutions Found (%) in 100 iterations

Compared to Pure Neural Net

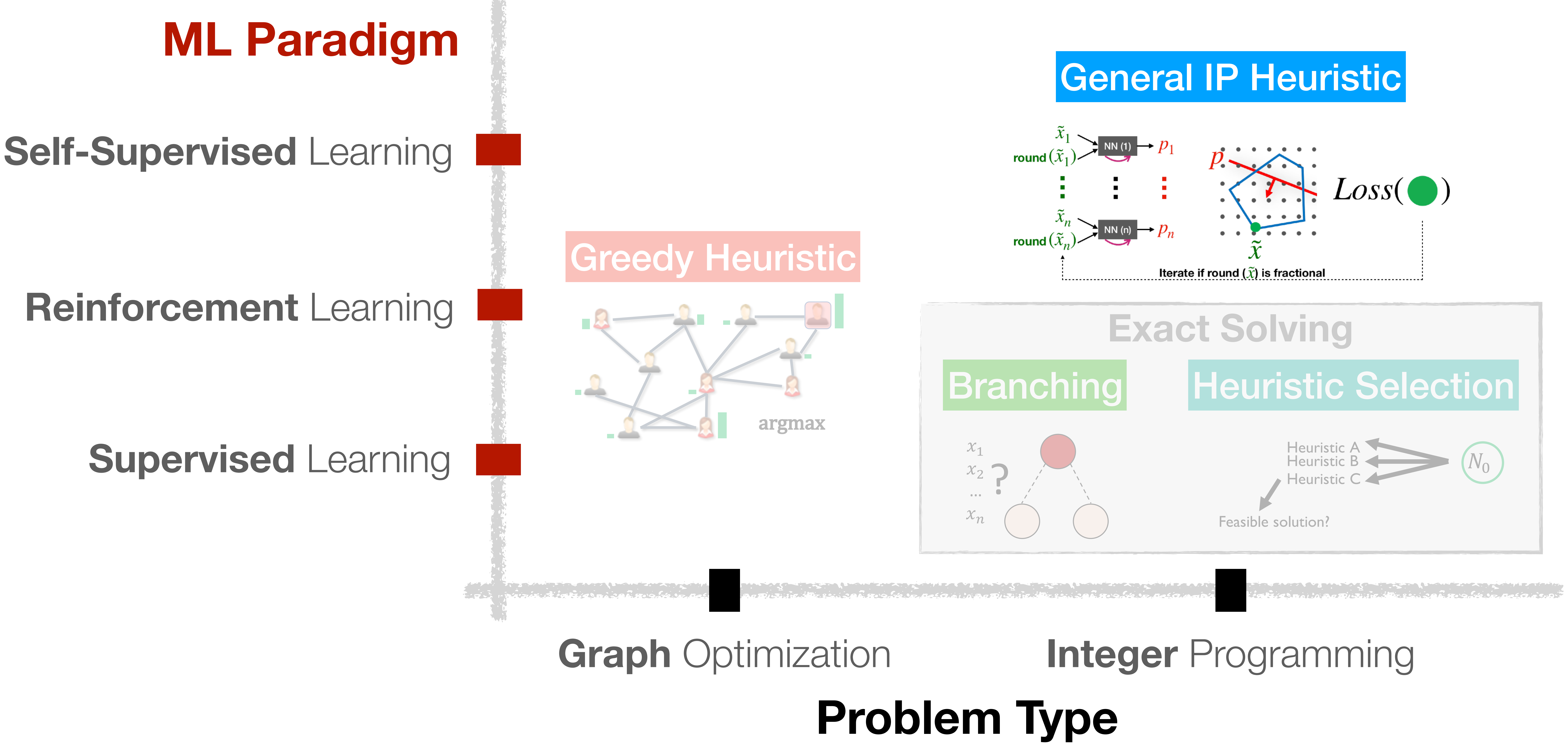
Easy for
SAT solvers

- **SAT**isfiability problem
- **NIO**: use model from GAP on SAT
- **NeuroSAT***: Deep Learning model for SAT solving
  - Trained with **supervised** learning
  - **Millions of training instances**

Higher is Better



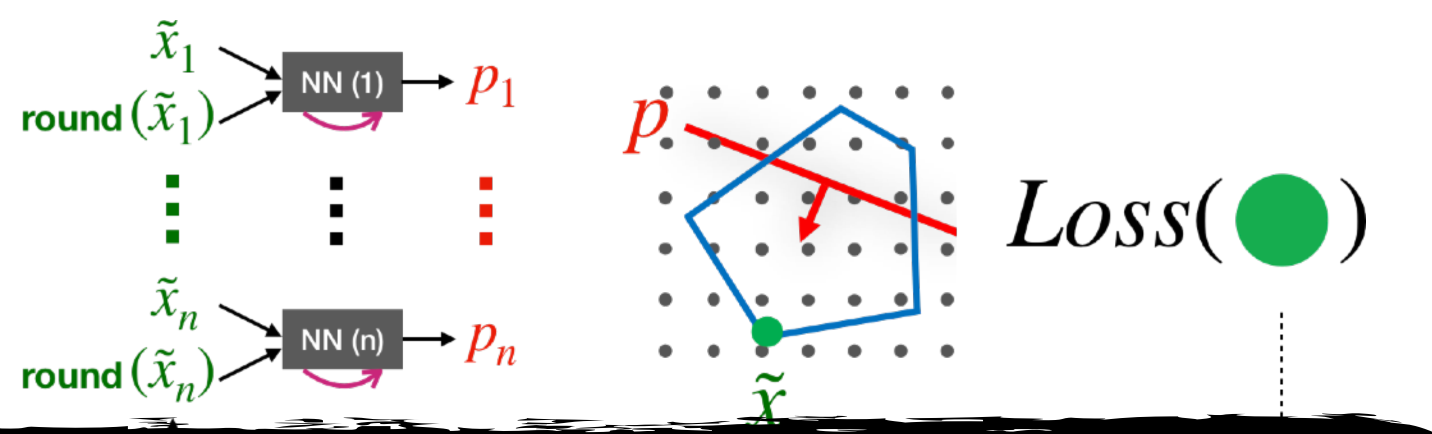
* Learning a SAT Solver from Single-Bit Supervision. ICLR 2019


ML Paradigm
Self-Supervised Learning
Reinforcement Learning
Supervised Learning
General IP Heuristic
NN (1)
NN (n)
$Loss$(
Iterate if round ($\tilde{x}$) is fractional
Greedy Heuristic
argmax
Exact Solving
Branching
Heuristic Selection
Heuristic A
Heuristic B
Heuristic C
Feasible solution?
Graph Optimization
Integer Programming
Problem Type

ML Paradigm

Self-Supervised Learning

General IP Heuristic



Takeaways

- Incorporate **LP-projections** into **neural network** model
- Can **learn** heuristics for **arbitrary Integer Programs**
- **No supervised or reinforcement learning required!**
- **Outperforms** the **Feasibility Pump** on various problems

Problem Type

Humans learn to **design** **algorithms**.

Machine Learning

Can **algorithms** “learn” to **design** **algorithms**?

Discrete Optimization

Machine Learning

Can **algorithms** “learn” to **design algorithms**?

Discrete Optimization

Machine Learning

Can **algorithms** "learn" to ~~**design**~~ **tailor** **algorithms**?

Discrete Optimization

Machine Learning

Can **algorithms** "learn" to ~~**design**~~ **tailor** **algorithms**?

Discrete Optimization

Yes!

Machine Learning

Can **algorithms** “learn” to ~~**design**~~ **tailor** **algorithms**?

Discrete Optimization

Yes!

ML complements human algorithms

ML fills in algorithm **details using data**

Data-Driven Algorithm Design

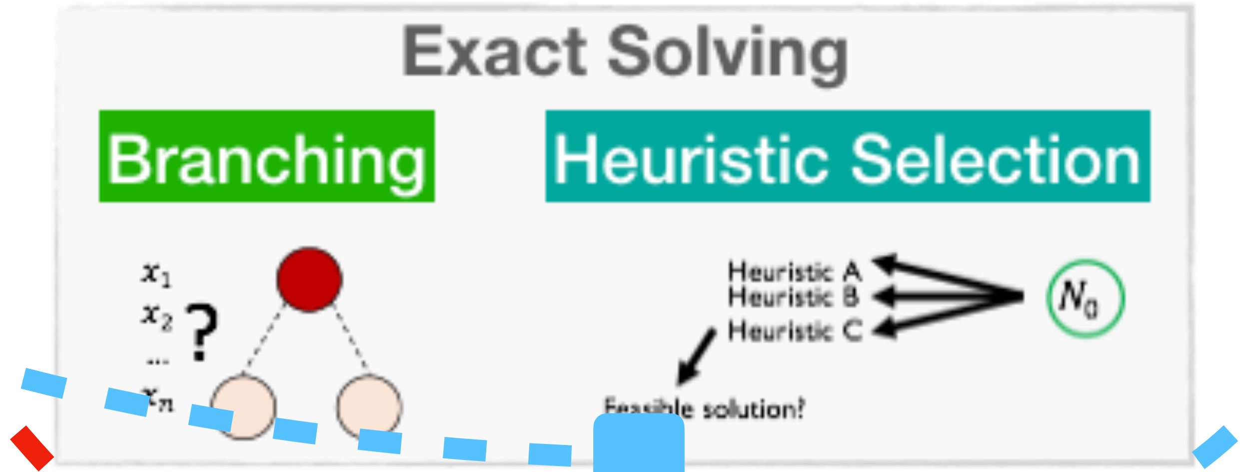
Impact in ML and OPT

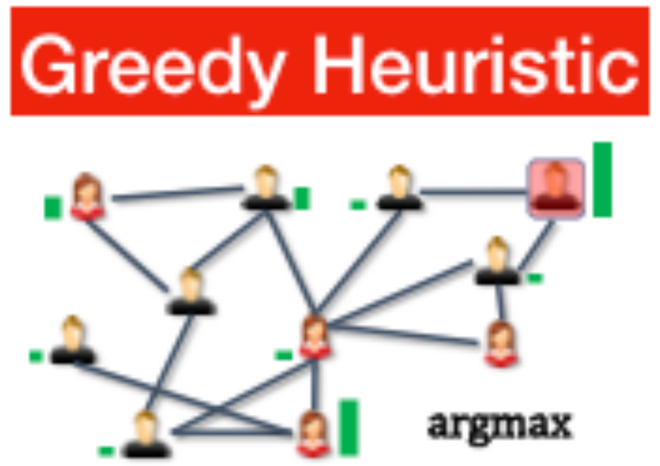
ML models for DiscOpt

- Attention for TSP [Kool+, 2019]
- Graph Convolutions [Li+, 2018]
- Imitation learning [Song+, 2018]

Combinatorial problems

- SAT [Selsam+, 2019]
- SMT [Balunovic+, 2018]
- k-Coverage [Li+, 2019]
- Scheduling [Mao+, 2019]
- Assignment [Emami+, 2018]
- VRP [Nazari+, 2018]
- Multiple-TSP [Kaempfer+, 2018]
- Stochastic Opt. [Nair+, 2018]





Branch-and-Bound

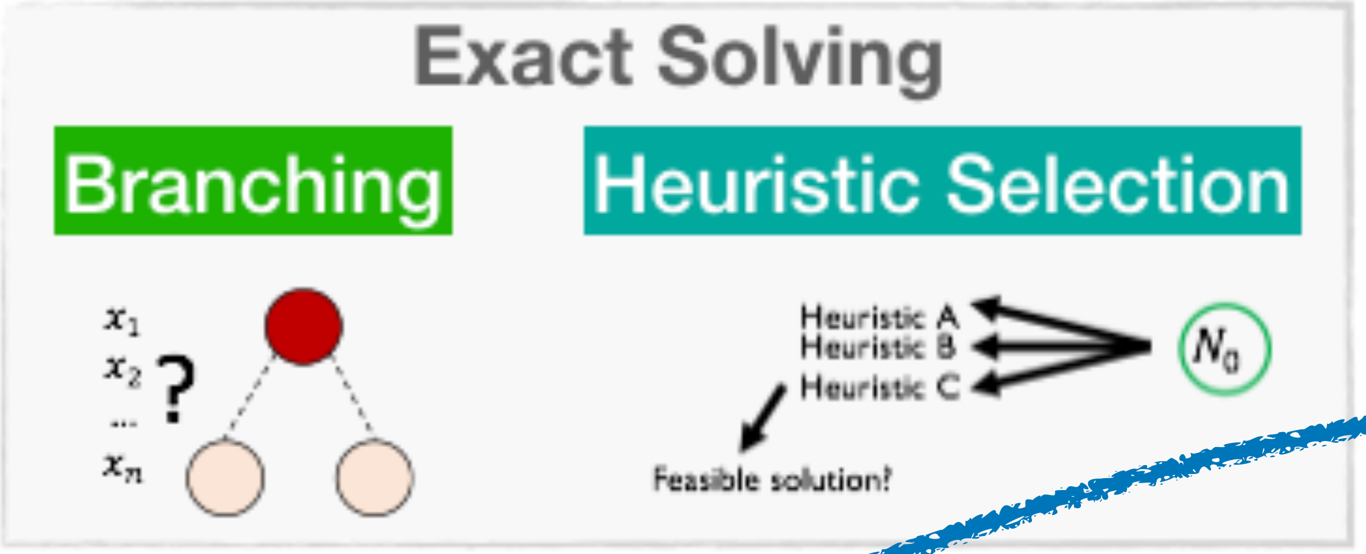
- Multi-objective IP [Sierra-Altamiranda+, 2019]
- Outcome prediction [Fischetti+, 2019]
- Cut selection [Baltean-Lugojan+, 2018]
- Formulation selection [Bonami+, 2018]
- Solution prediction [Larsen+, 2018]
- Decompositions [Kruber+, 2017]

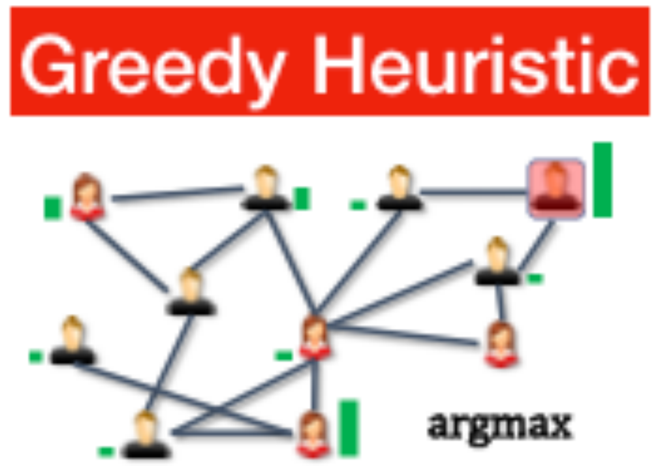
Applications

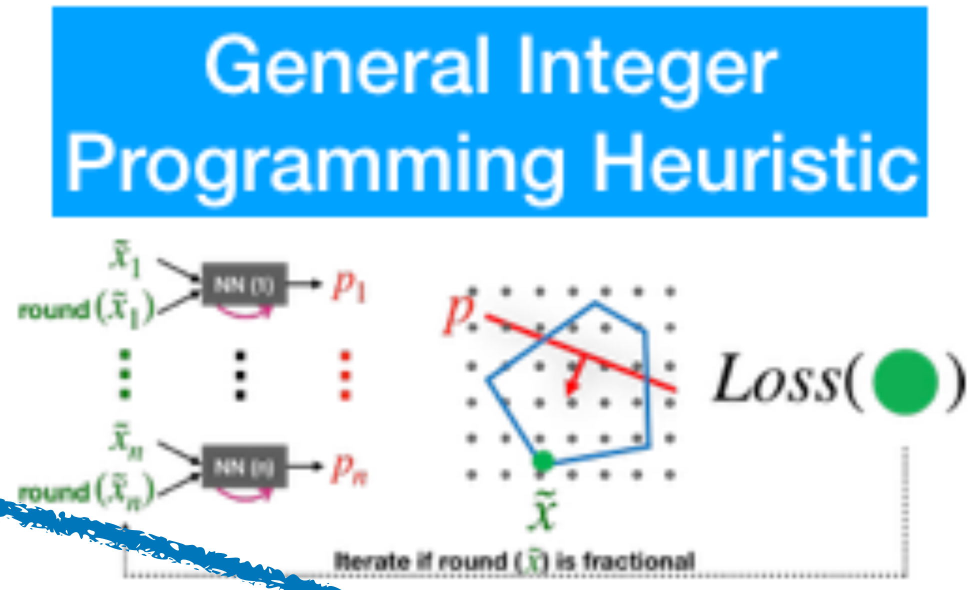
- Unit commitment [Xavier+, 2019]
- Sensor placement [Shen+, 2019]
- Recommender systems [Fu+, 2017]

Theory

- Learning to Branch [Balcan+, 2018]

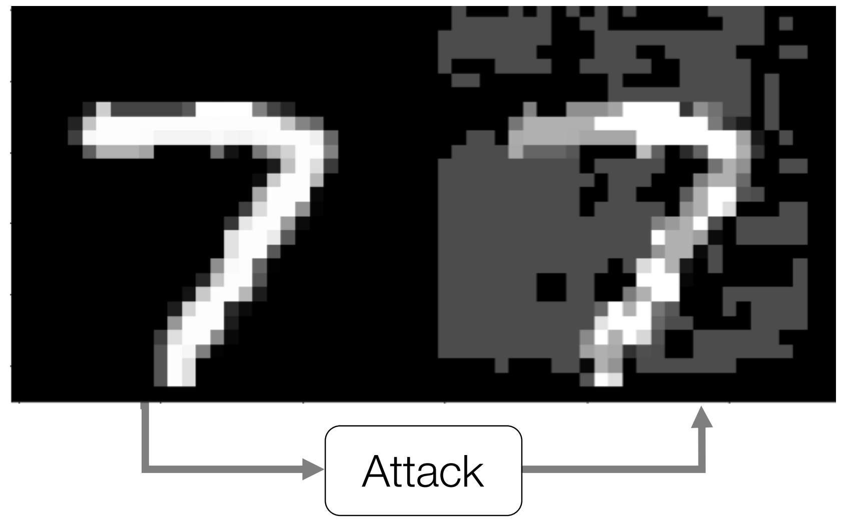


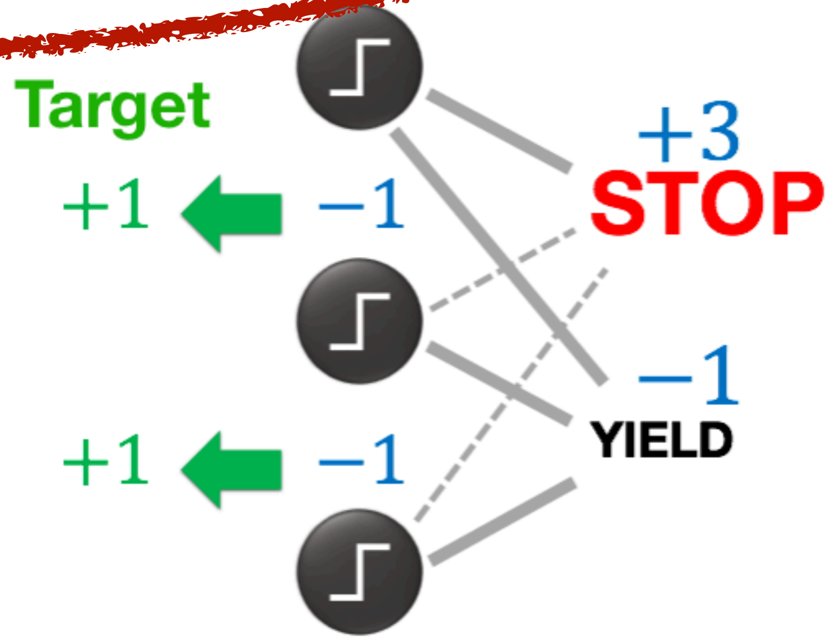


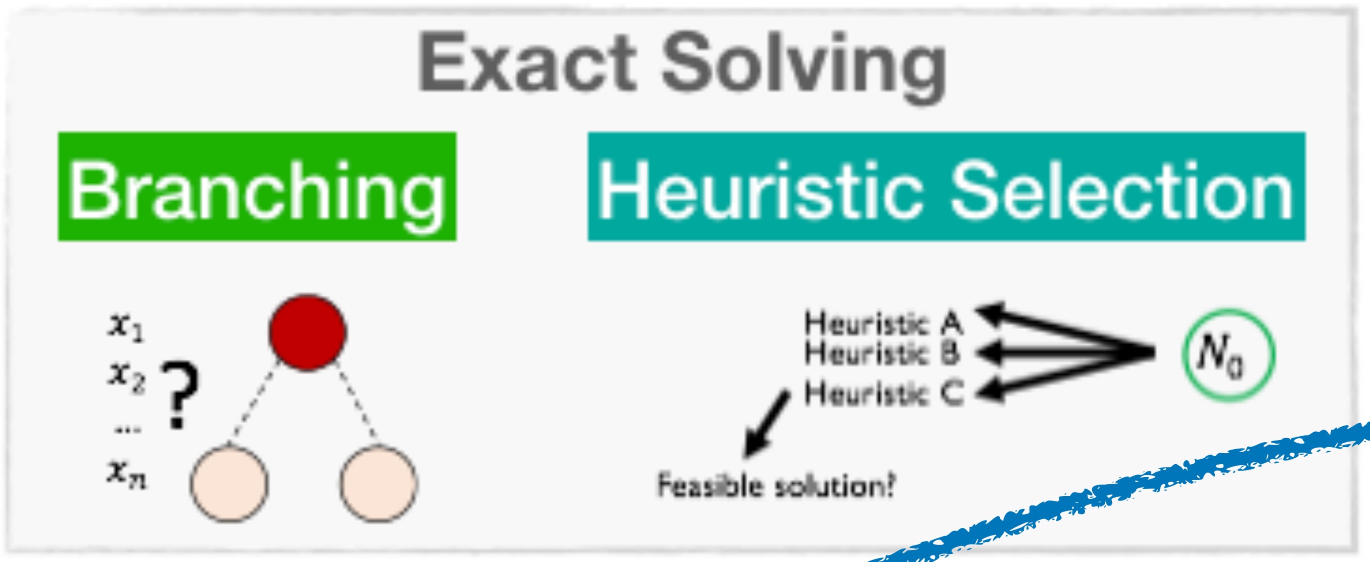


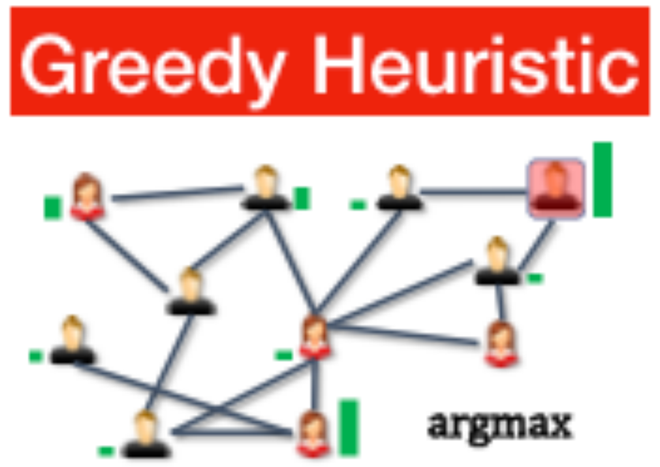
Machine Learning

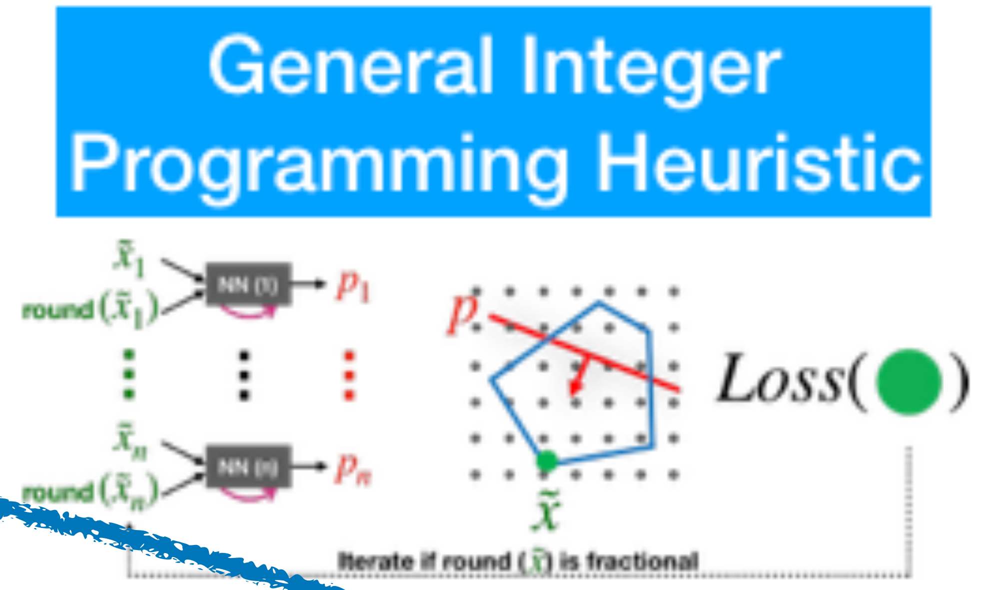
Discrete Optimization









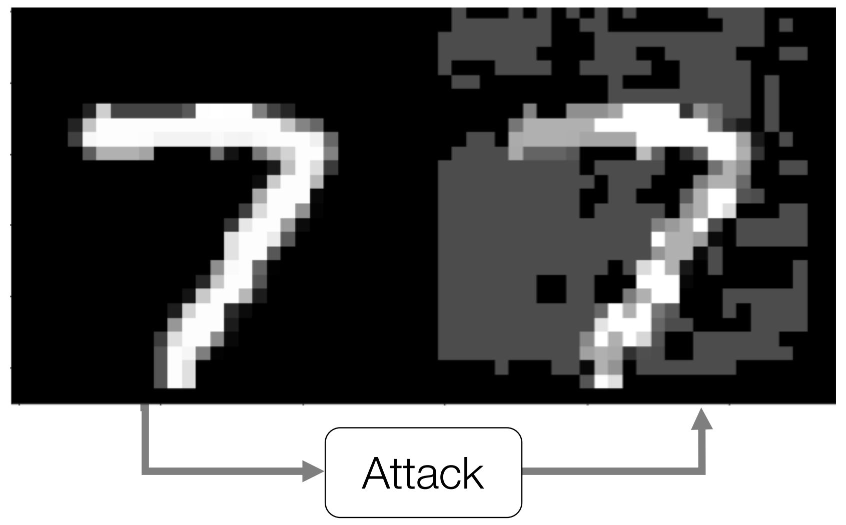


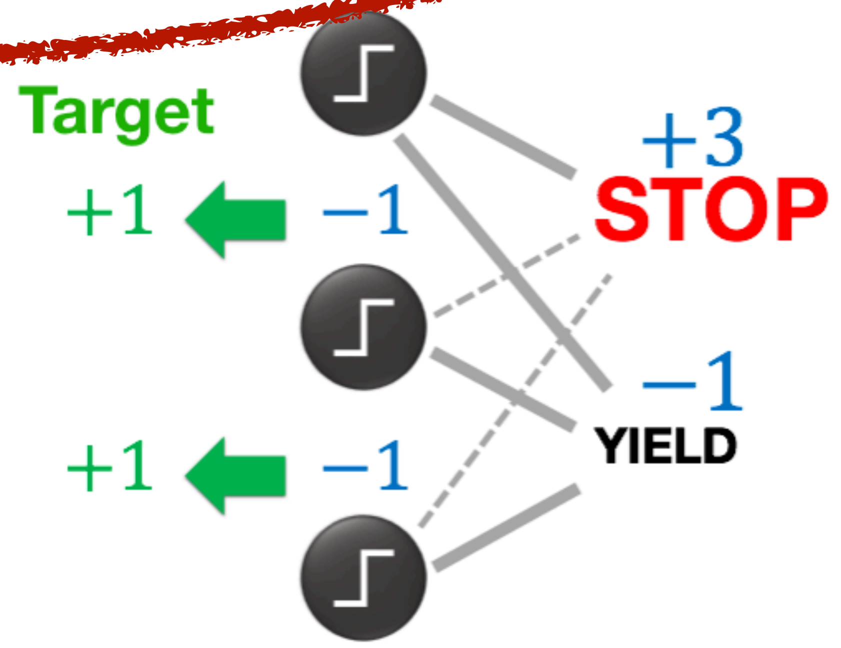
Machine Learning

ML x OPT

Exciting synergies and challenges in both directions

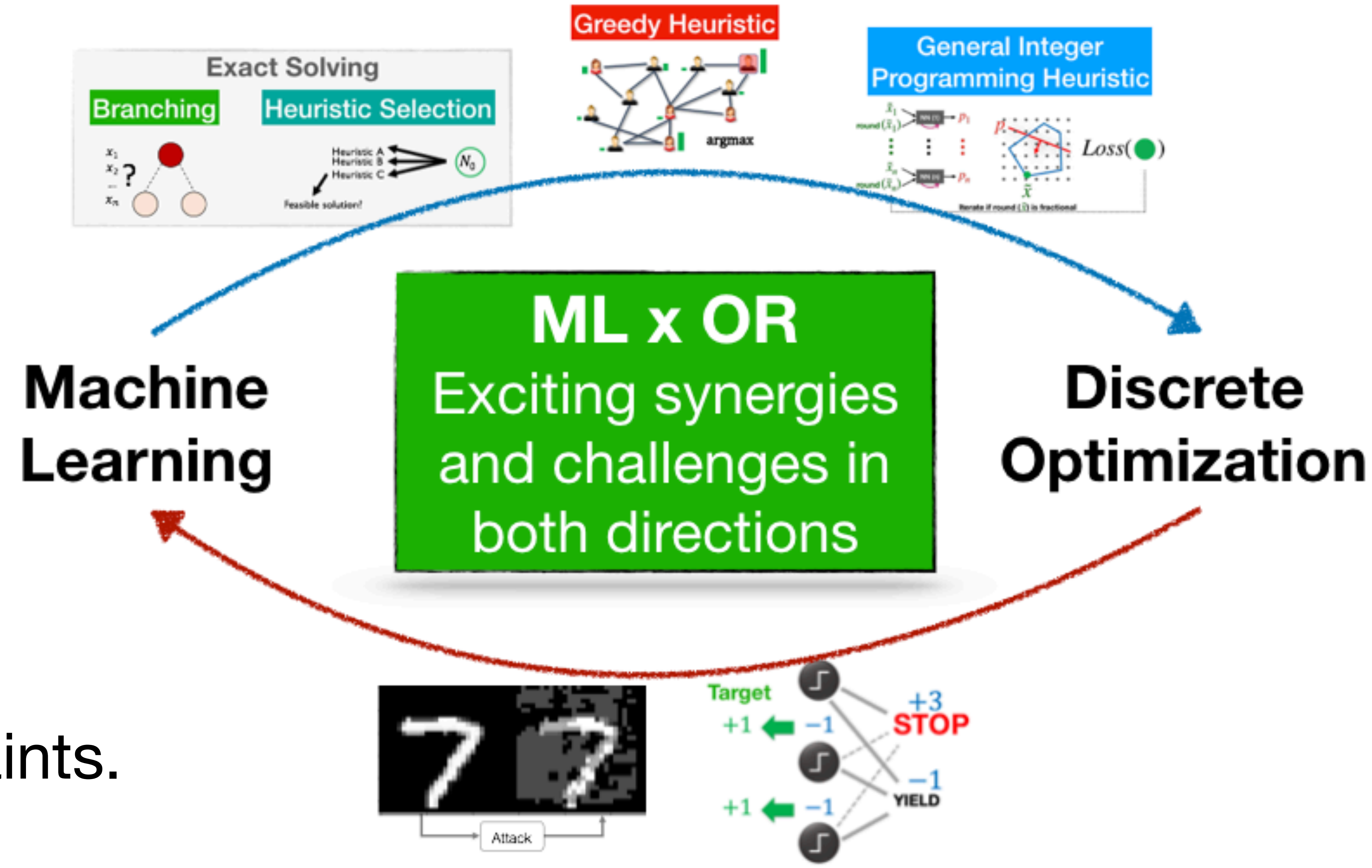
Discrete Optimization





Questions?

www.ekhalil.com



Relevant papers

Neural Integer Optimization: Learning to Satisfy Generic Constraints.
w/ R. Trivedi, B. Dilkina. **Submitted to NeurIPS 2019**.

Learning Combinatorial Optimization Algorithms over Graphs.
w/ H. Dai (co first auth.), Y. Zhang, B. Dilkina, L. Song. **NeurIPS 2017**.

Learning To Run Heuristics in Tree Search.
w/ B. Dilkina, G. Nemhauser, S. Ahmed, Y. Shao. **IJCAI 2017**.

Learning to Branch in Mixed Integer Programming.
w/ P. Le Bodic, L. Song, G. Nemhauser, B. Dilkina. **AAAI 2016**.

Combinatorial Attacks on Binarized Neural Networks.
w/ A. Gupta, B. Dilkina. **ICLR 2019**.