CS 598CM: ML for Compilers and Architecture

Instructor: Charith Mendis

ILLINOIS

Brief Announcements

- **Reading List:** Live on the website!
- **Paper Selections:** Due on August 31st
- **Paper Reviews:** We will use hotCRP to facilitate review writing. We will change the fields of the reviews.
- **Resources and tutorials:** Towards the bottom of the website

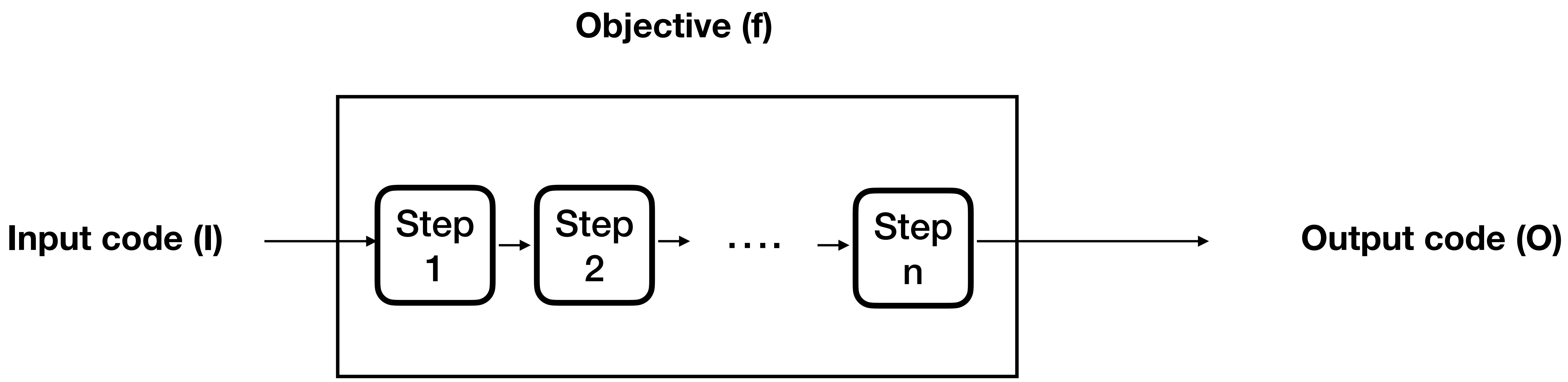
Recap

- Compiler Stages
  - Lexer => Parser => Sema => Optimization => Code Generation
- Two types of compiler optimizations
- Phase ordering problem

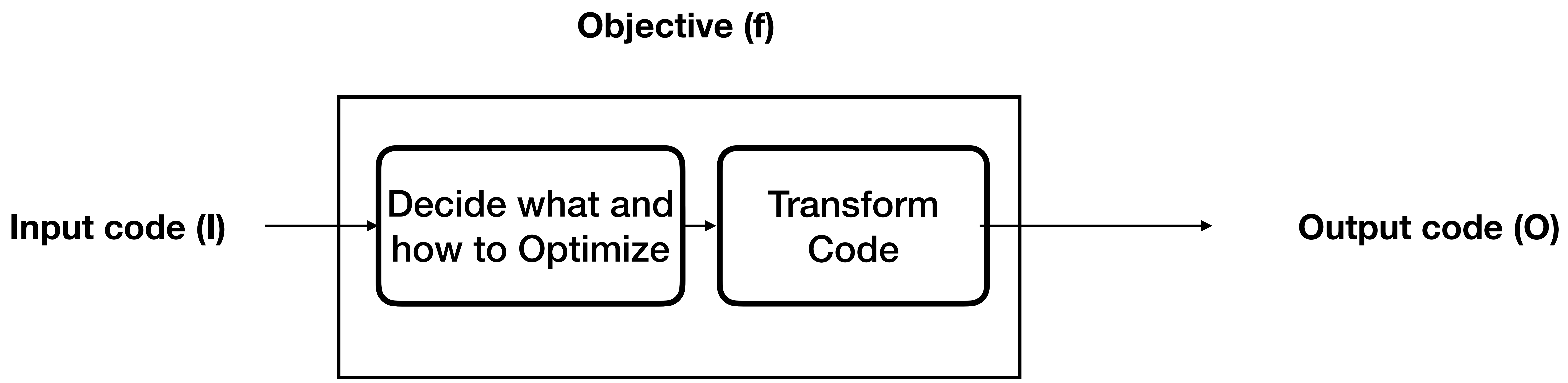
Lecture 3:
Compiler Optimizations

Optimizations + DSLs

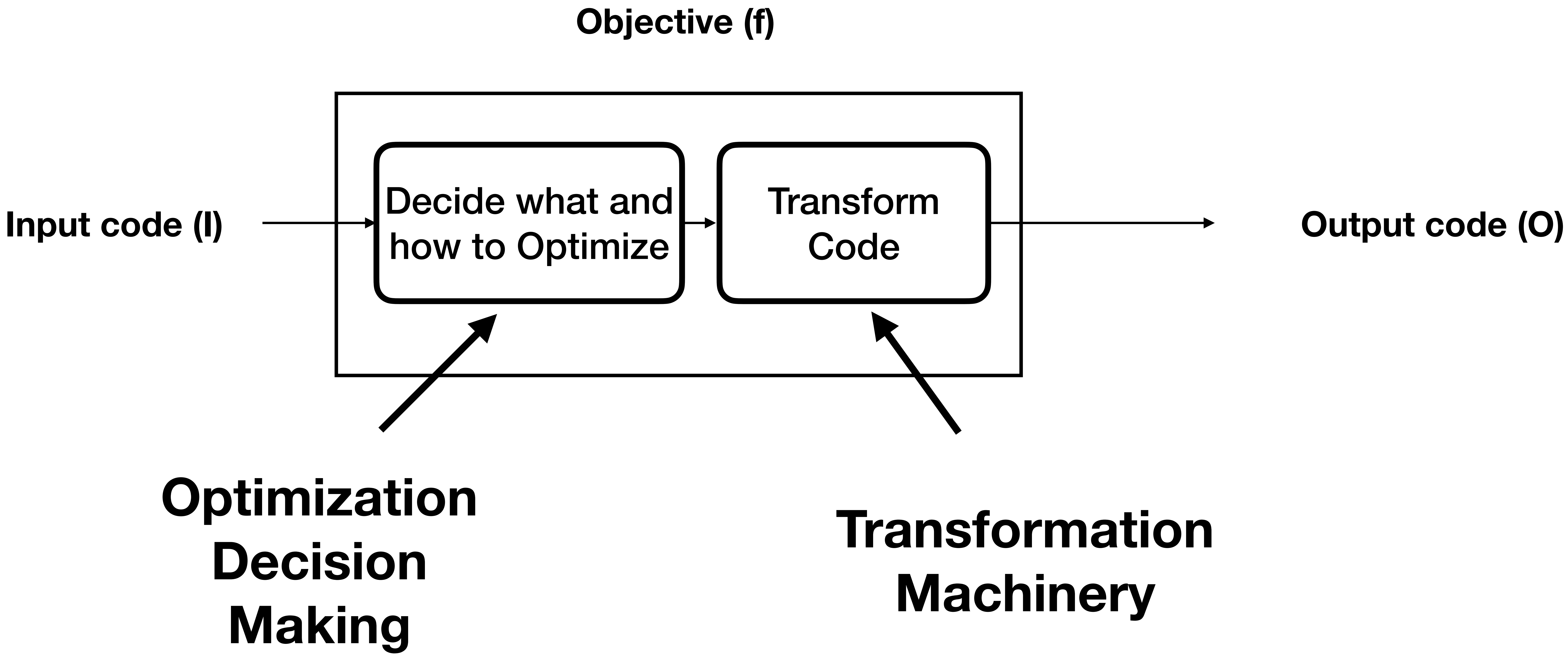
Anatomy of an Optimization Pass



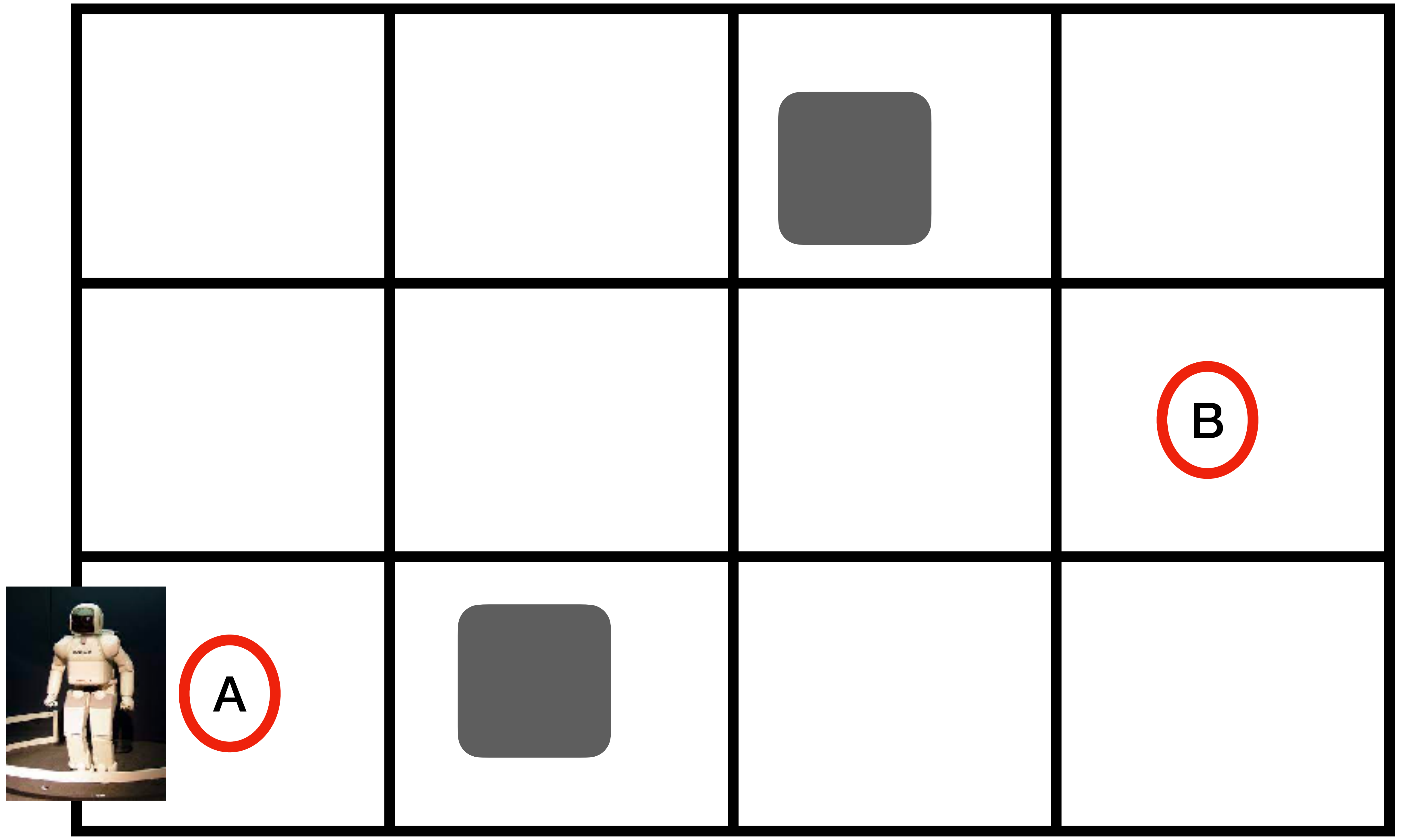
Anatomy of an Optimization Pass



Anatomy of an Optimization Pass

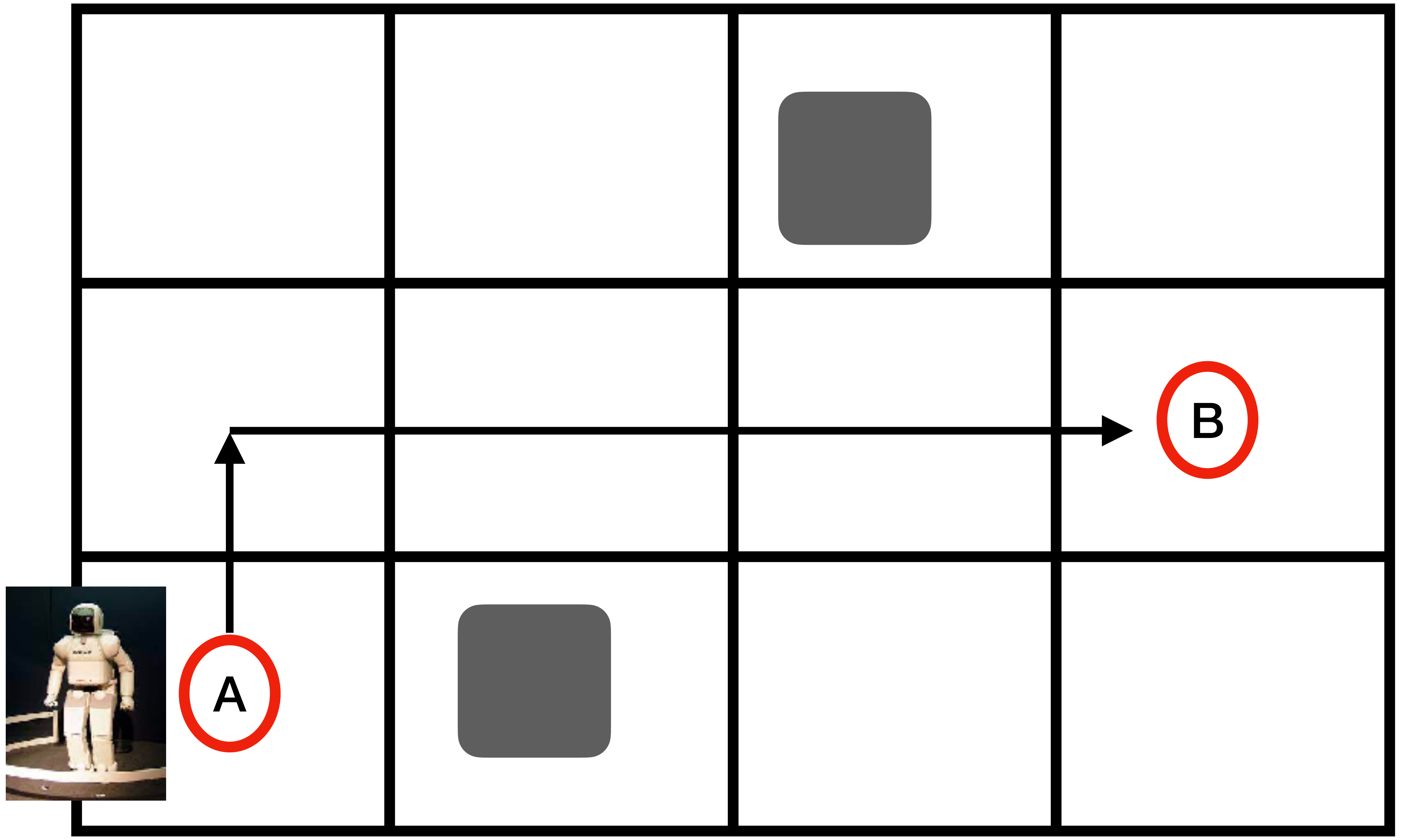


Robot Analogy



Task: Move from A to B cheaply

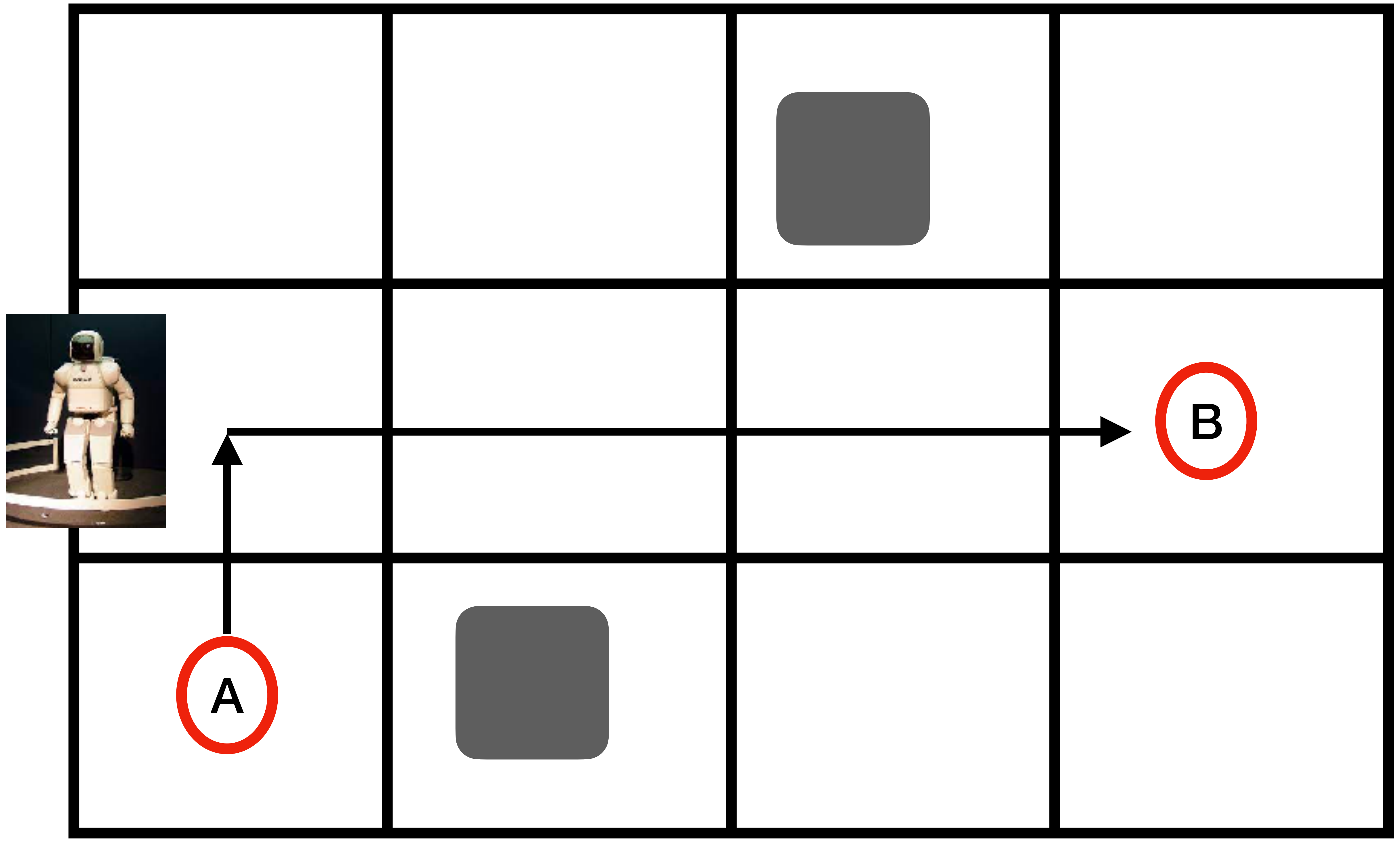
Robot Analogy



Task: Move from A to B cheaply

1. Plan

Robot Analogy

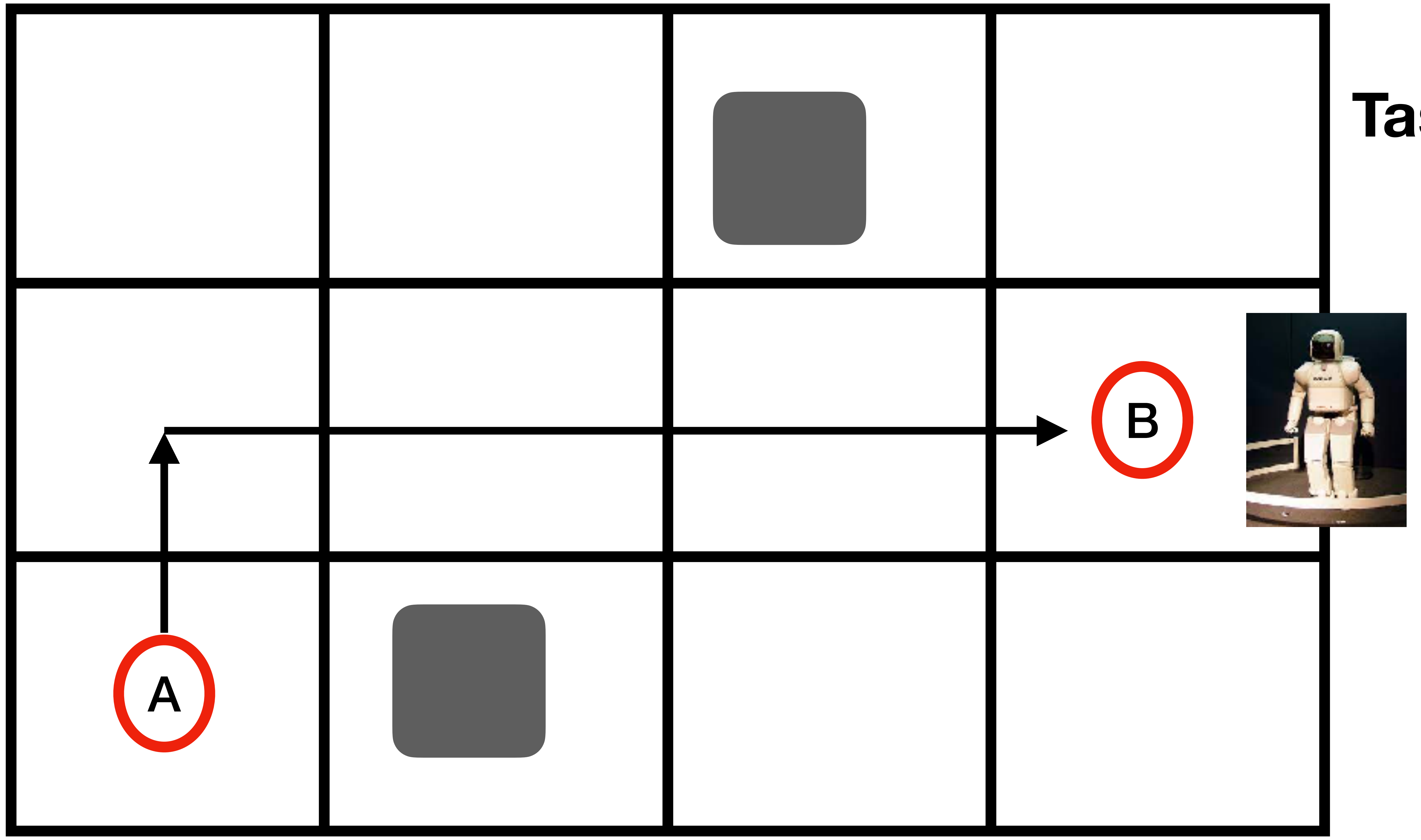


Task: Move from A to B cheaply

1. **Plan**

2. **Execute**

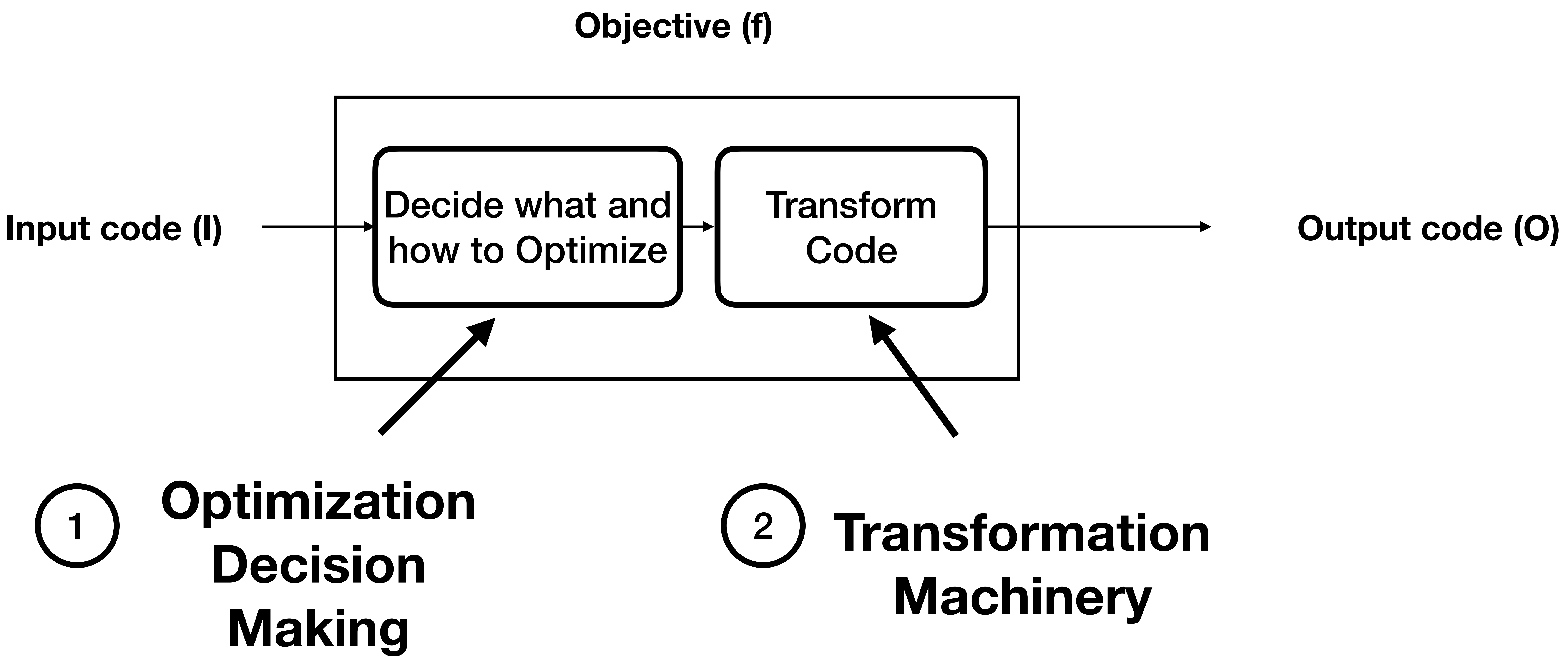
Robot Analogy



Task: Move from A to B cheaply

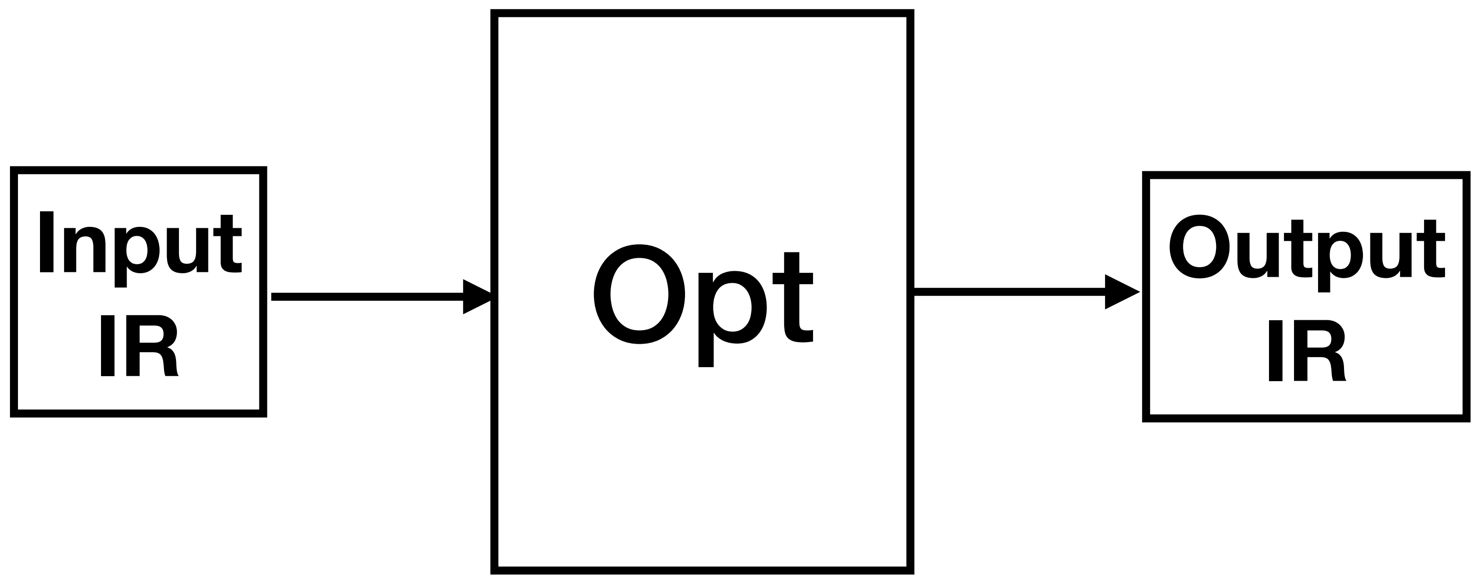
1. **Plan**
2. **Execute**

Anatomy of an Optimization Pass

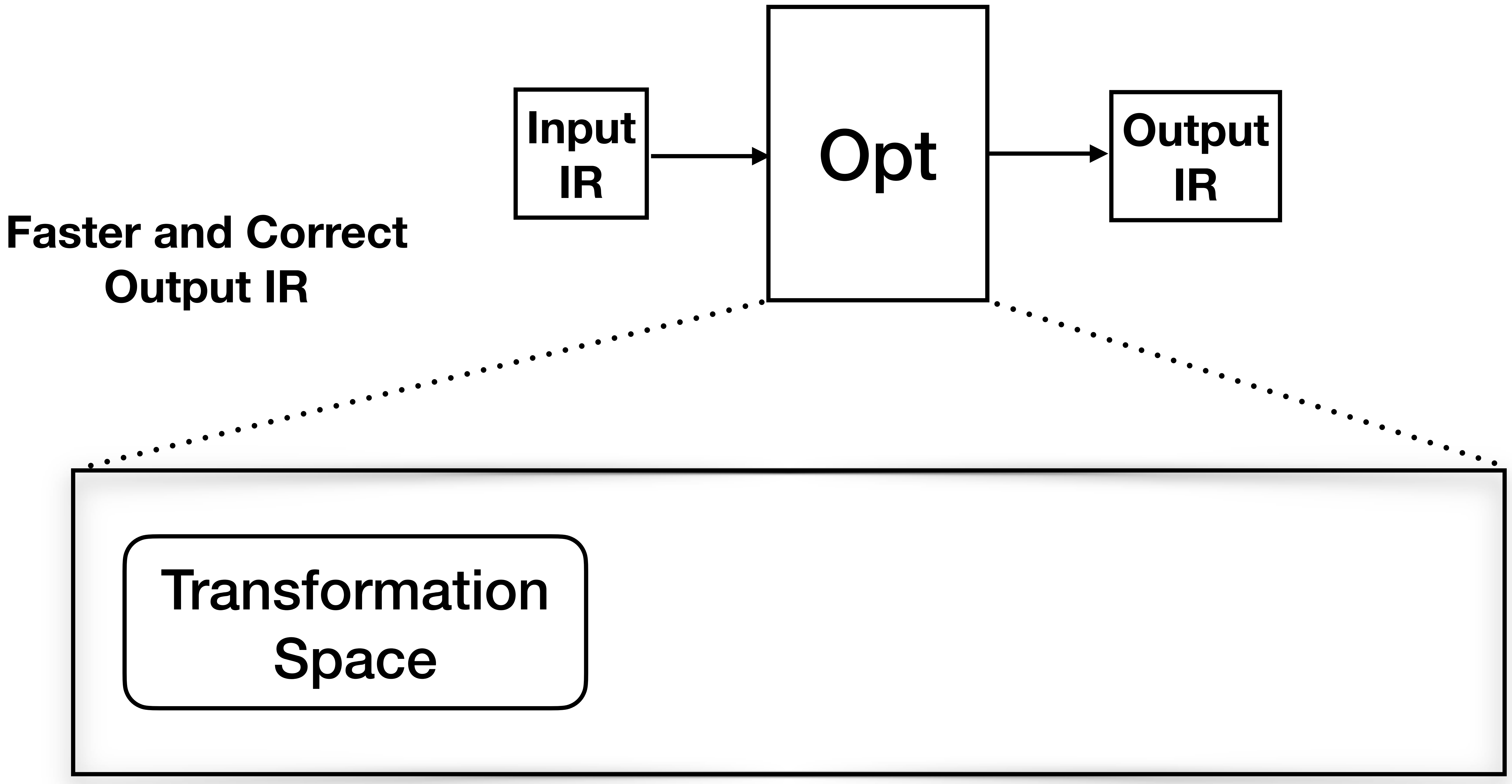


Optimization Decision Making

Faster and Correct
Output IR

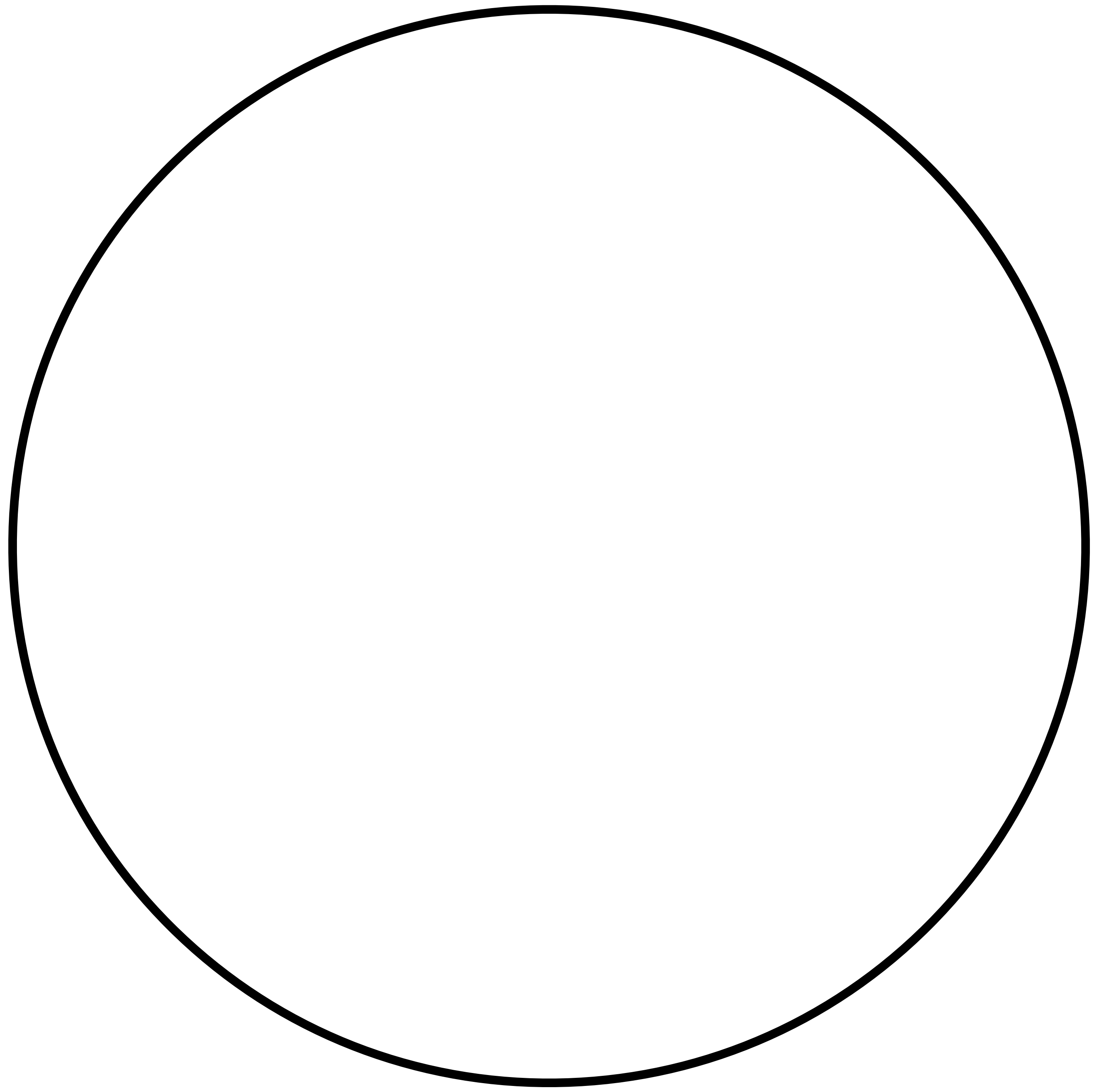


Optimization Decision Making

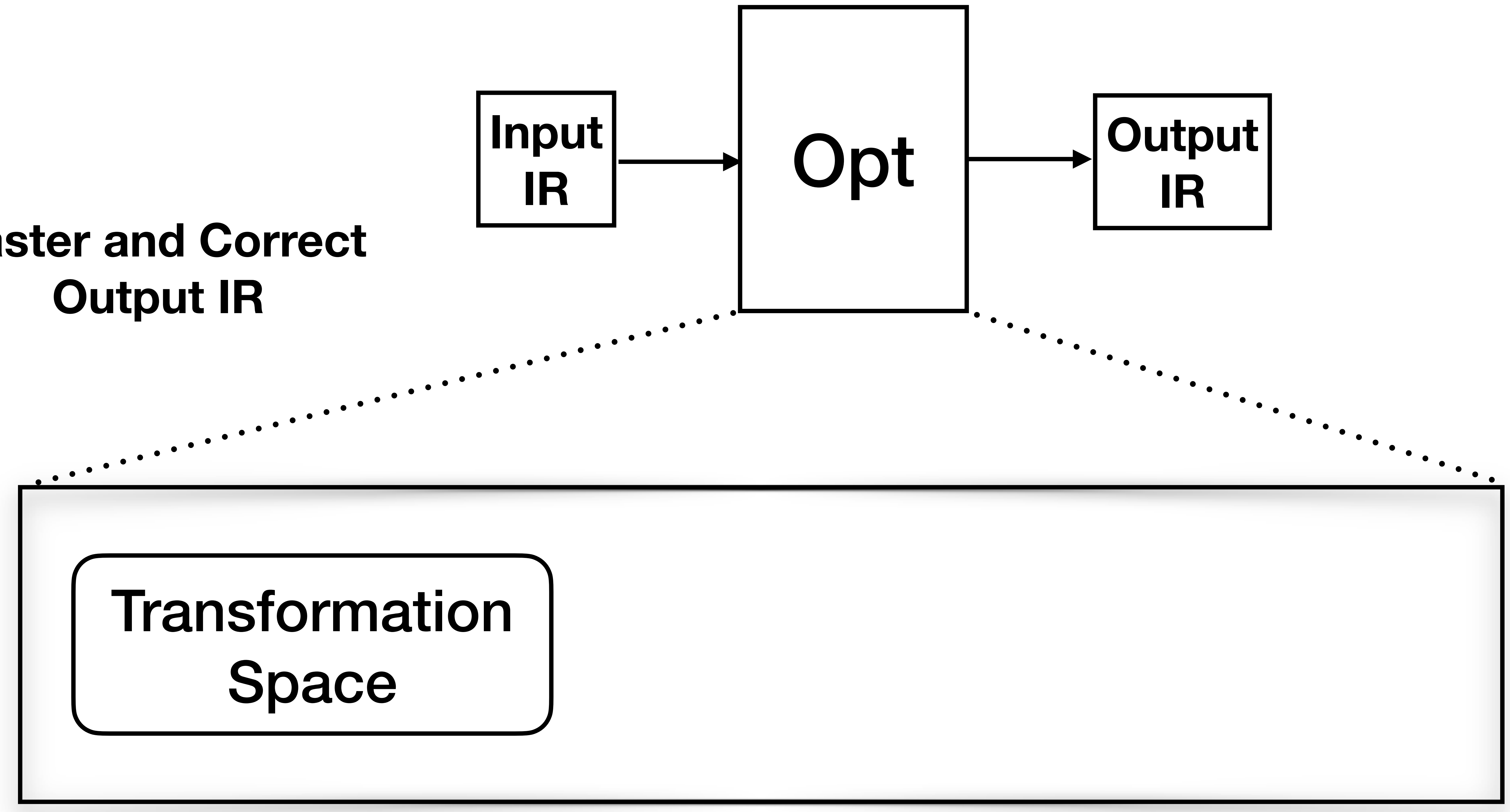


Optimization Decision Making

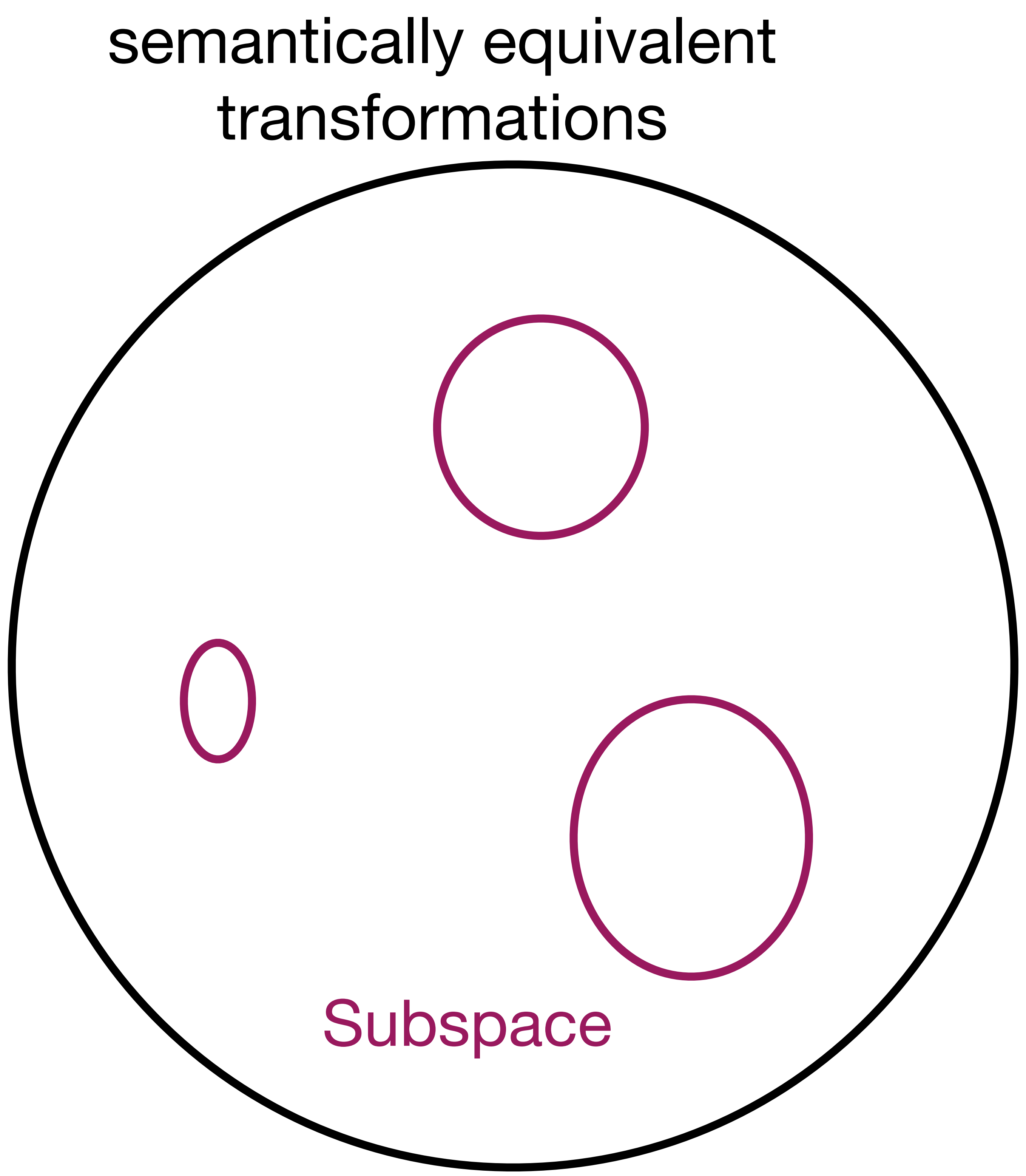
semantically equivalent
transformations

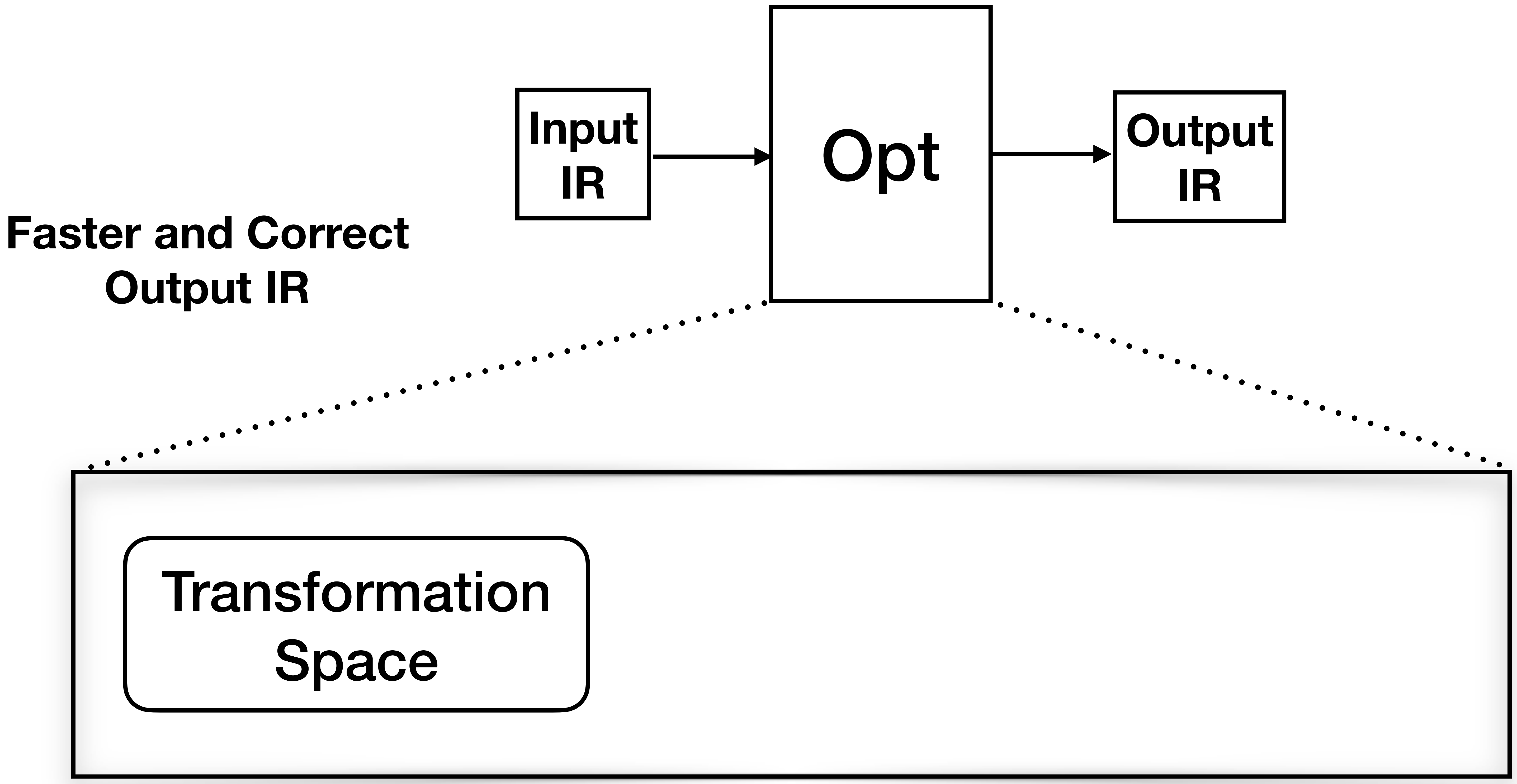


**Faster and Correct
Output IR**

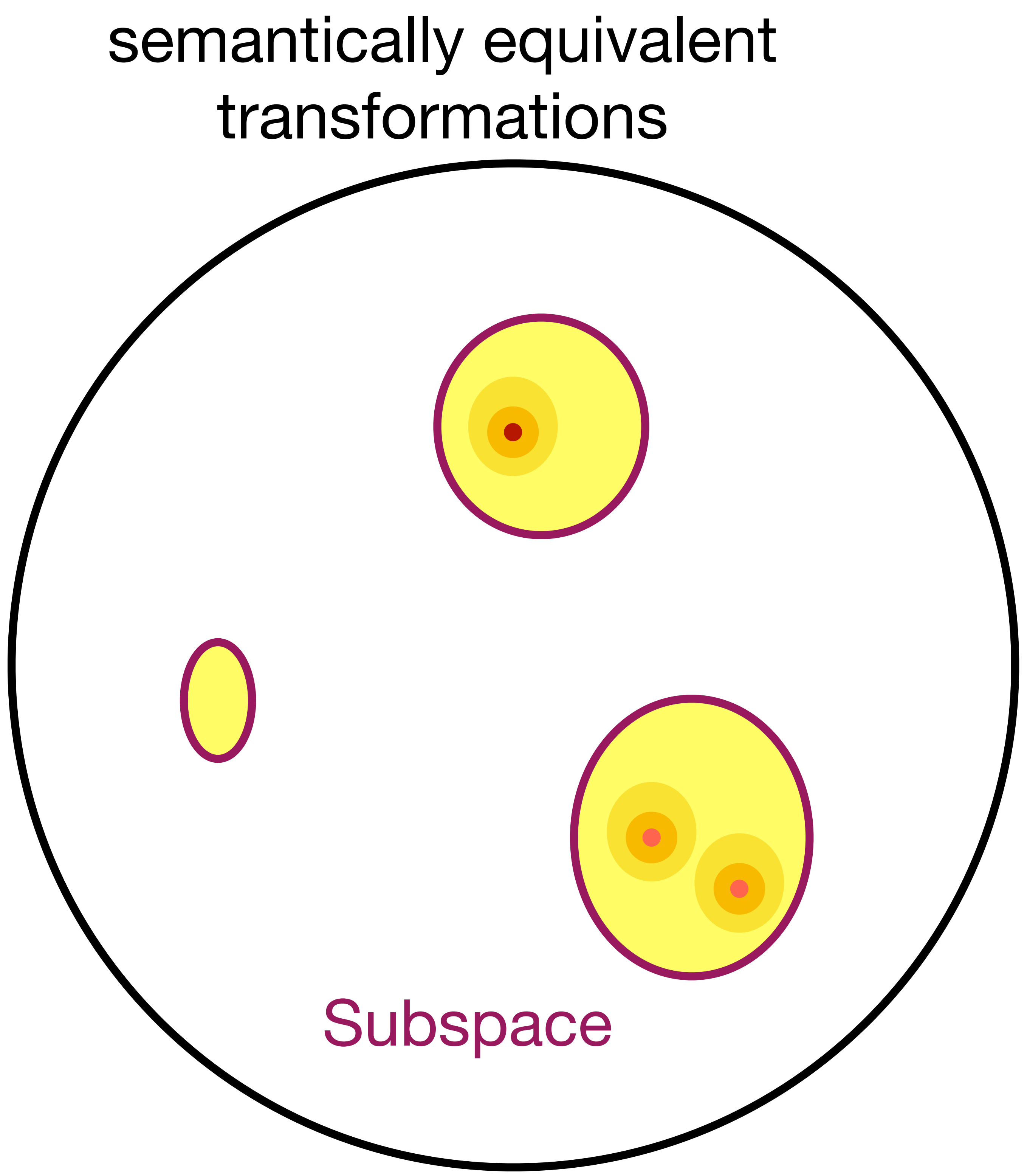


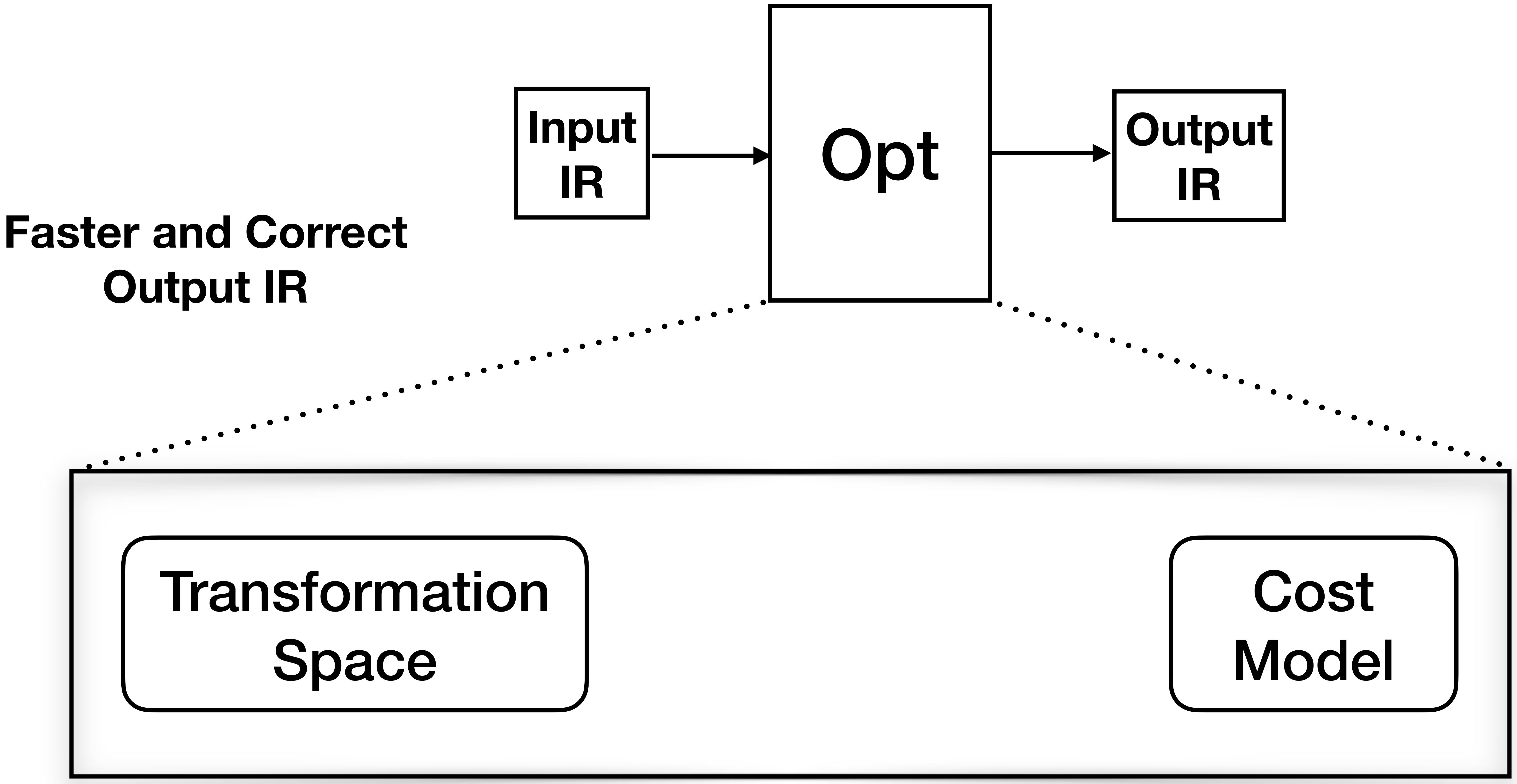
Optimization Decision Making



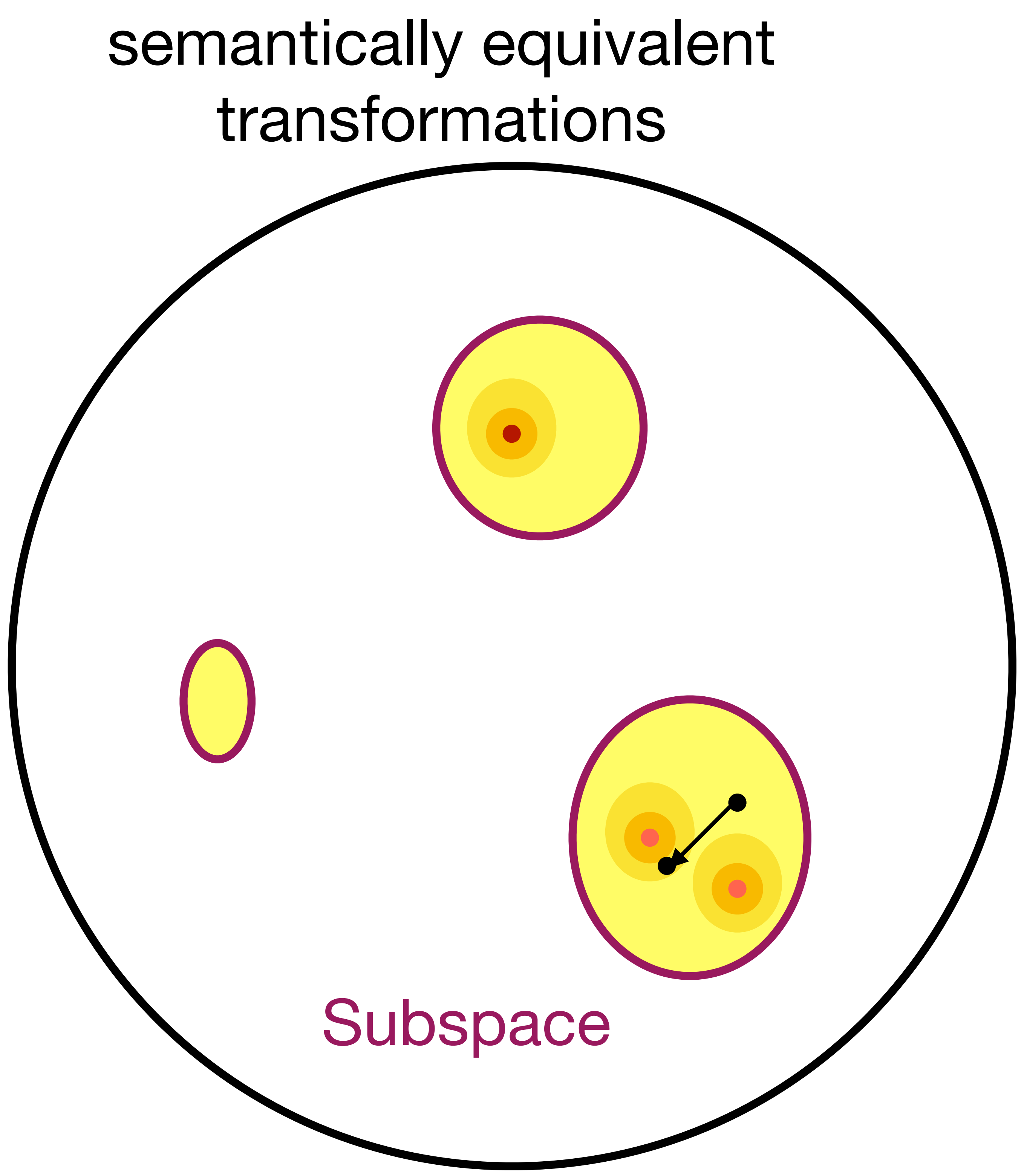


Optimization Decision Making

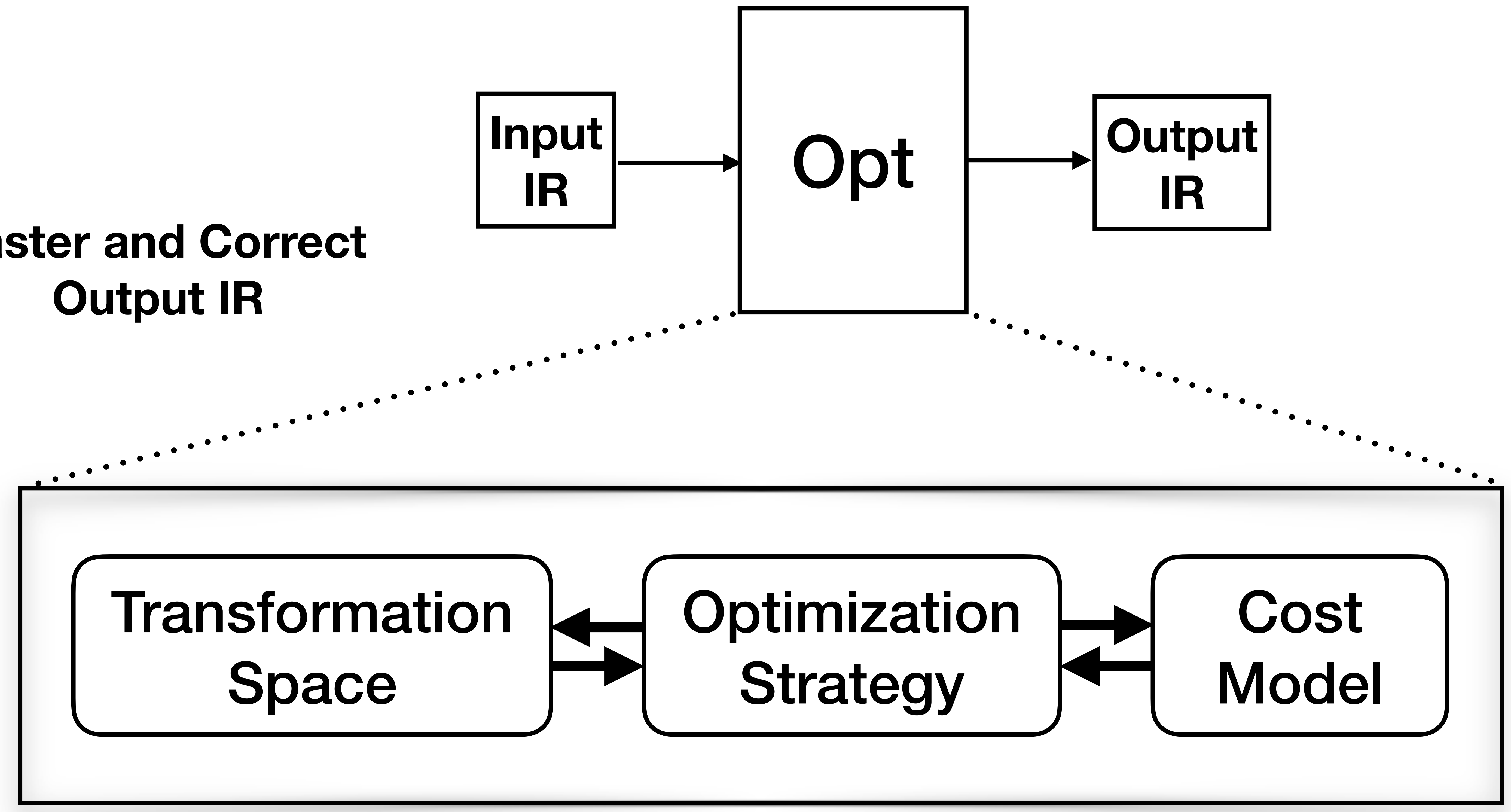




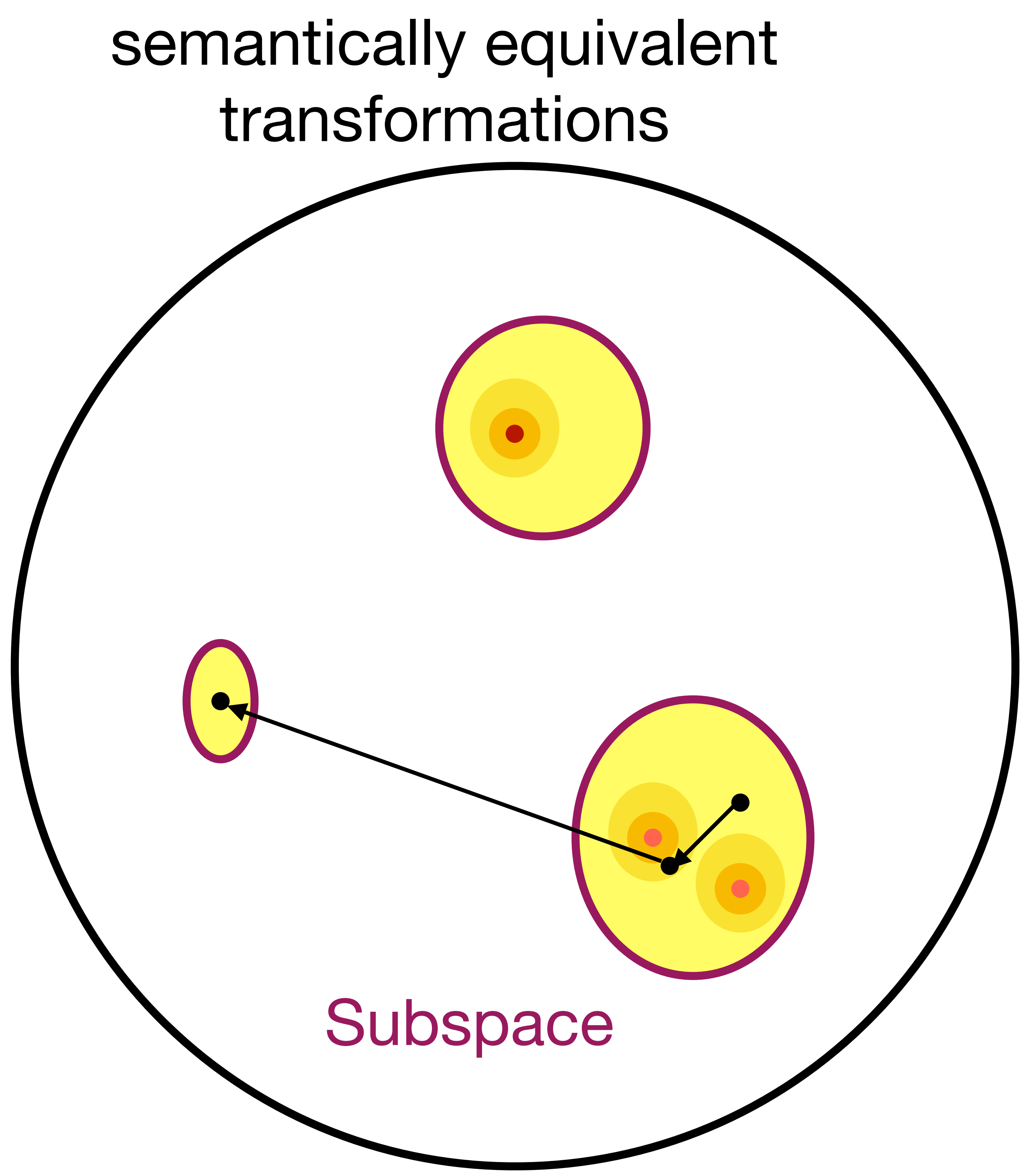
Optimization Decision Making

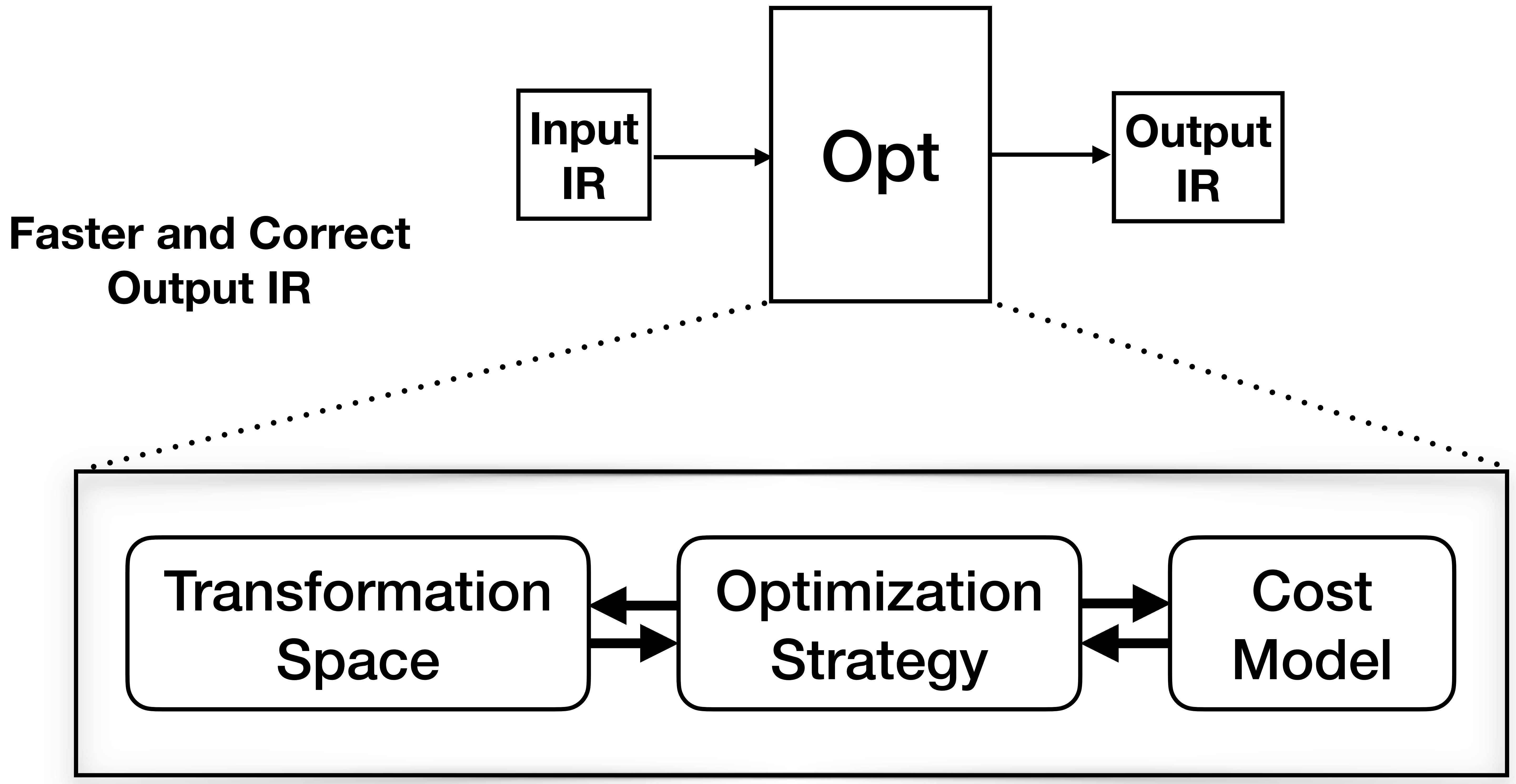


Faster and Correct Output IR



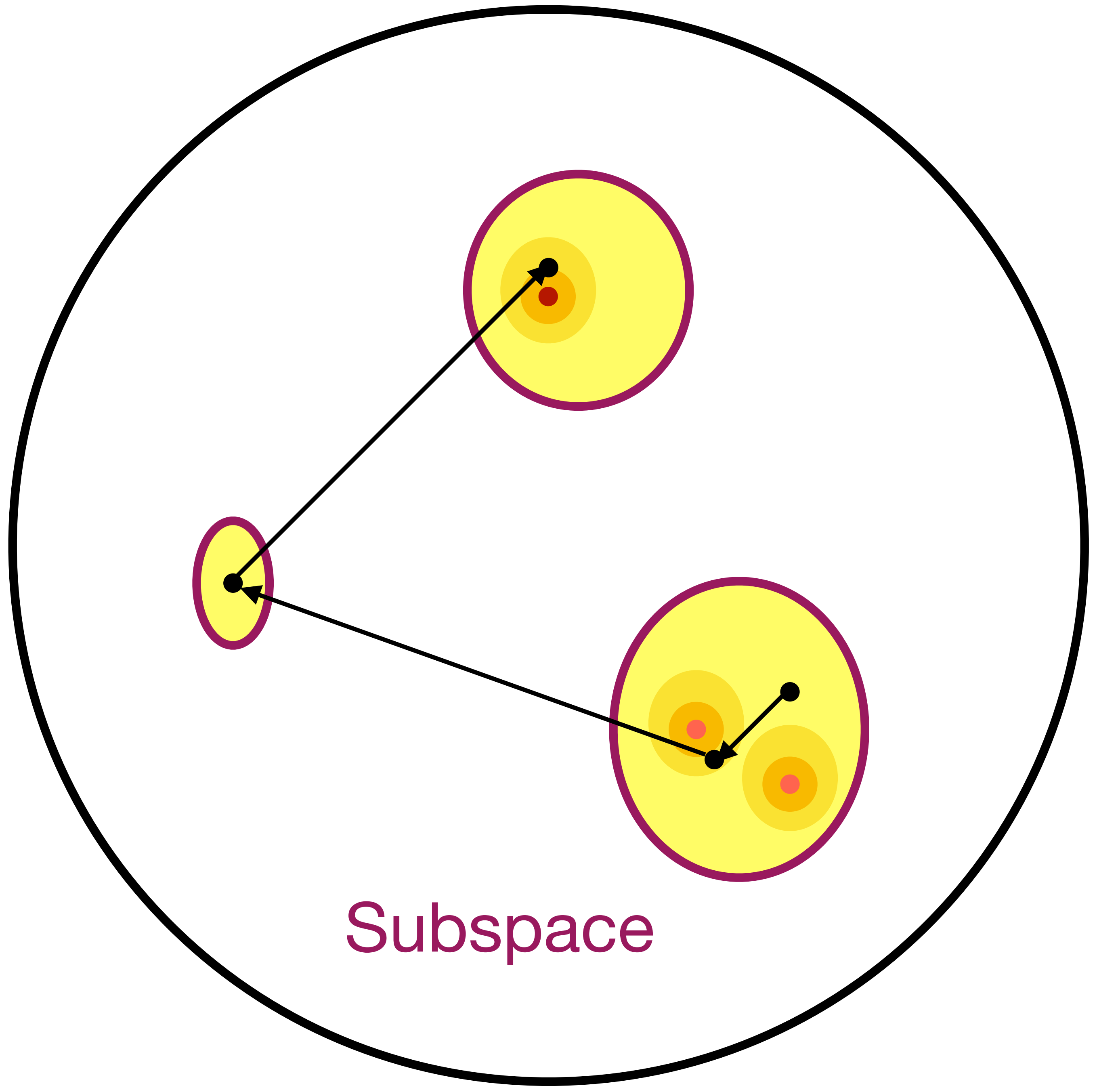
Optimization Decision Making



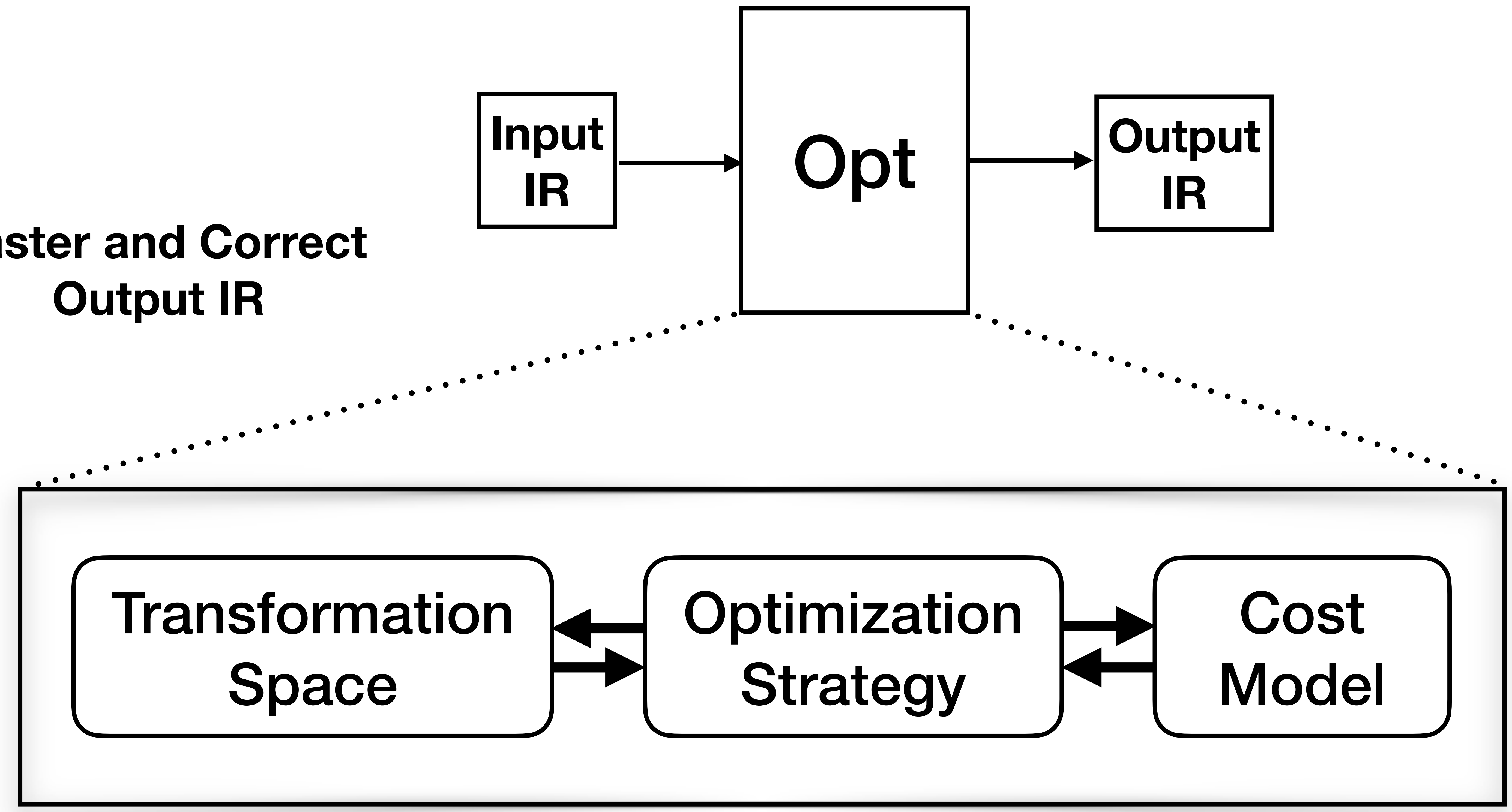


Optimization Decision Making

semantically equivalent transformations

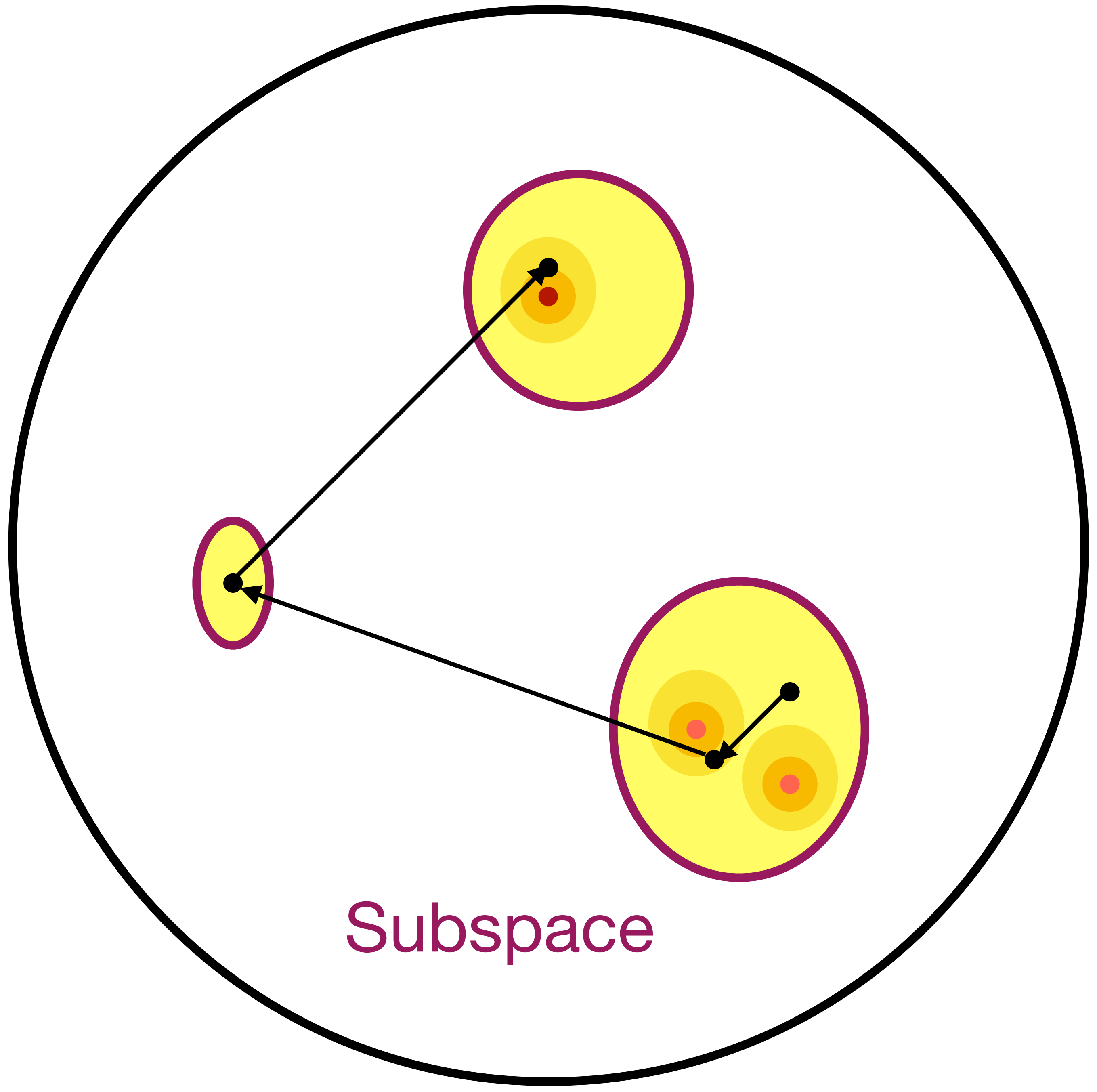


Faster and Correct Output IR

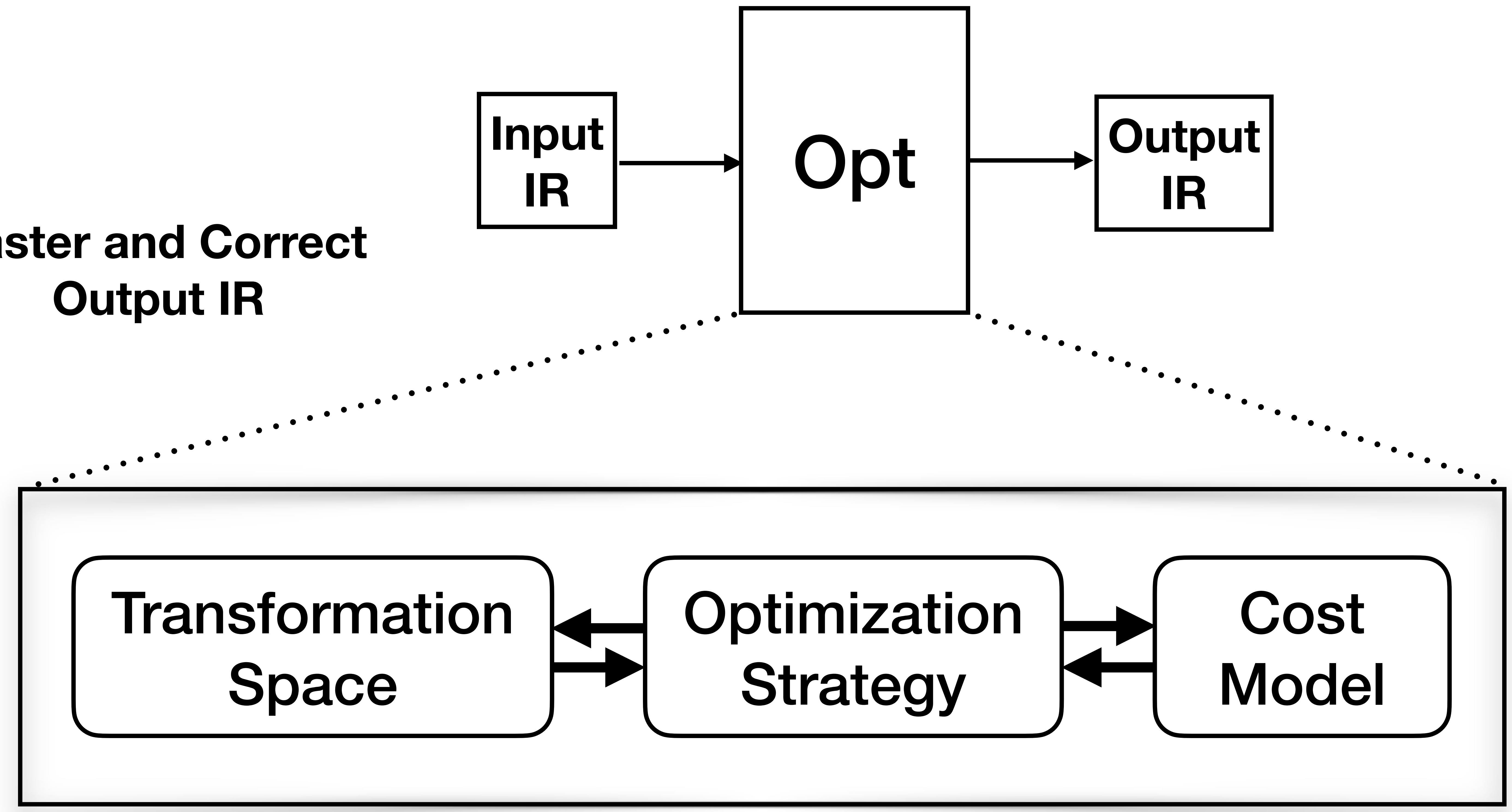


Optimization Decision Making

semantically equivalent transformations

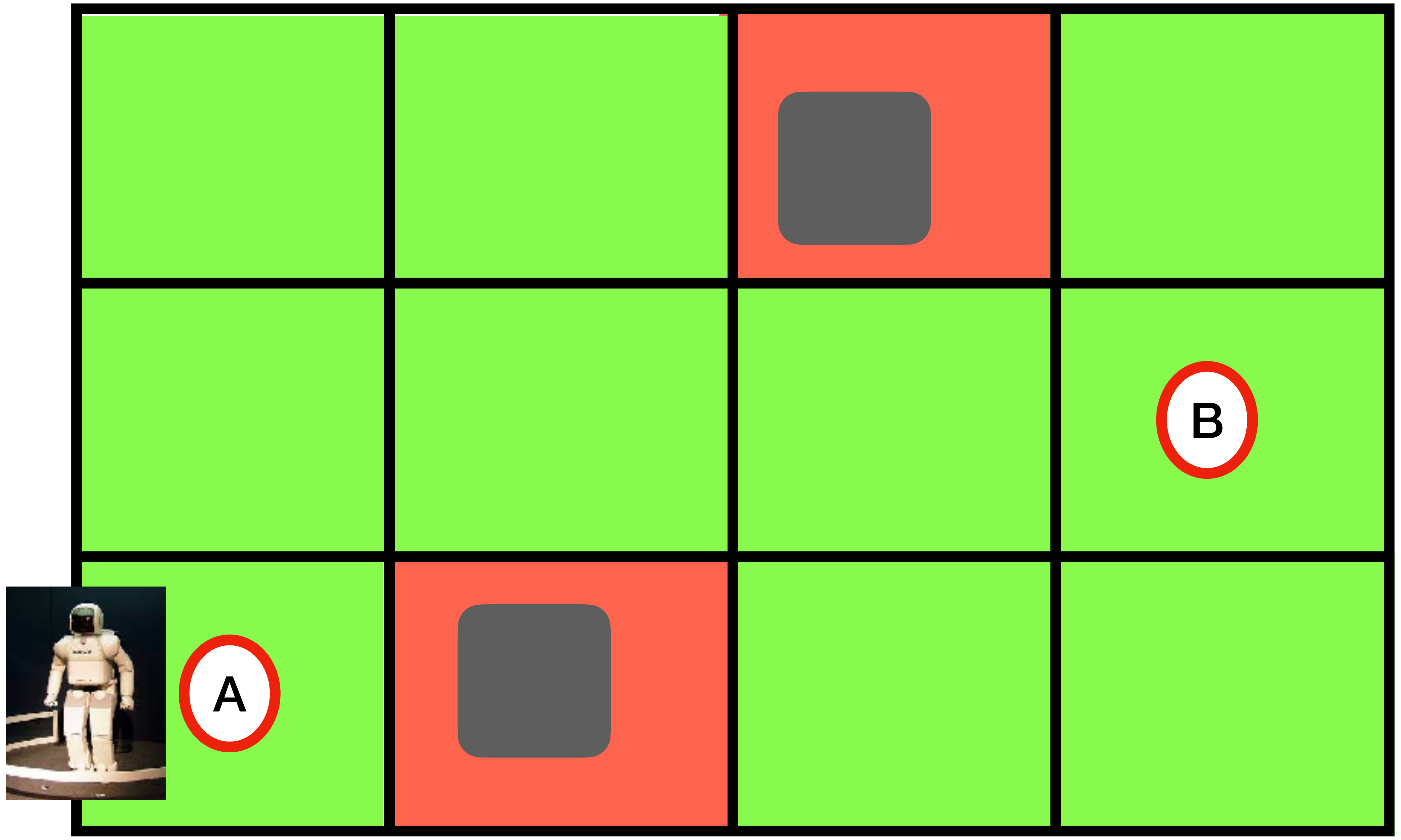


Faster and Correct Output IR



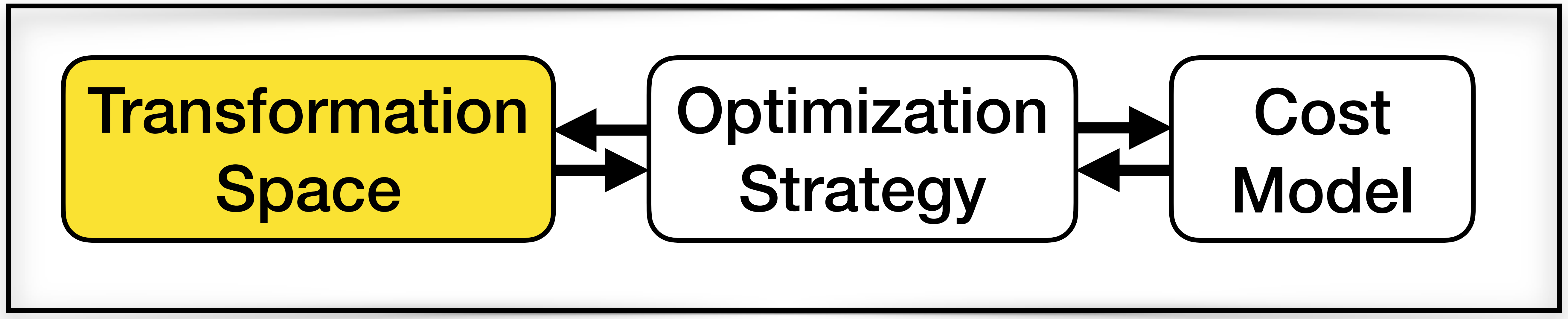
Transformation Space	Optimization Strategy	Cost Model
	Ideal	Ideal
Legal Transformations	**Optimal**	**Ground Truth Runtime**

Robot Analogy

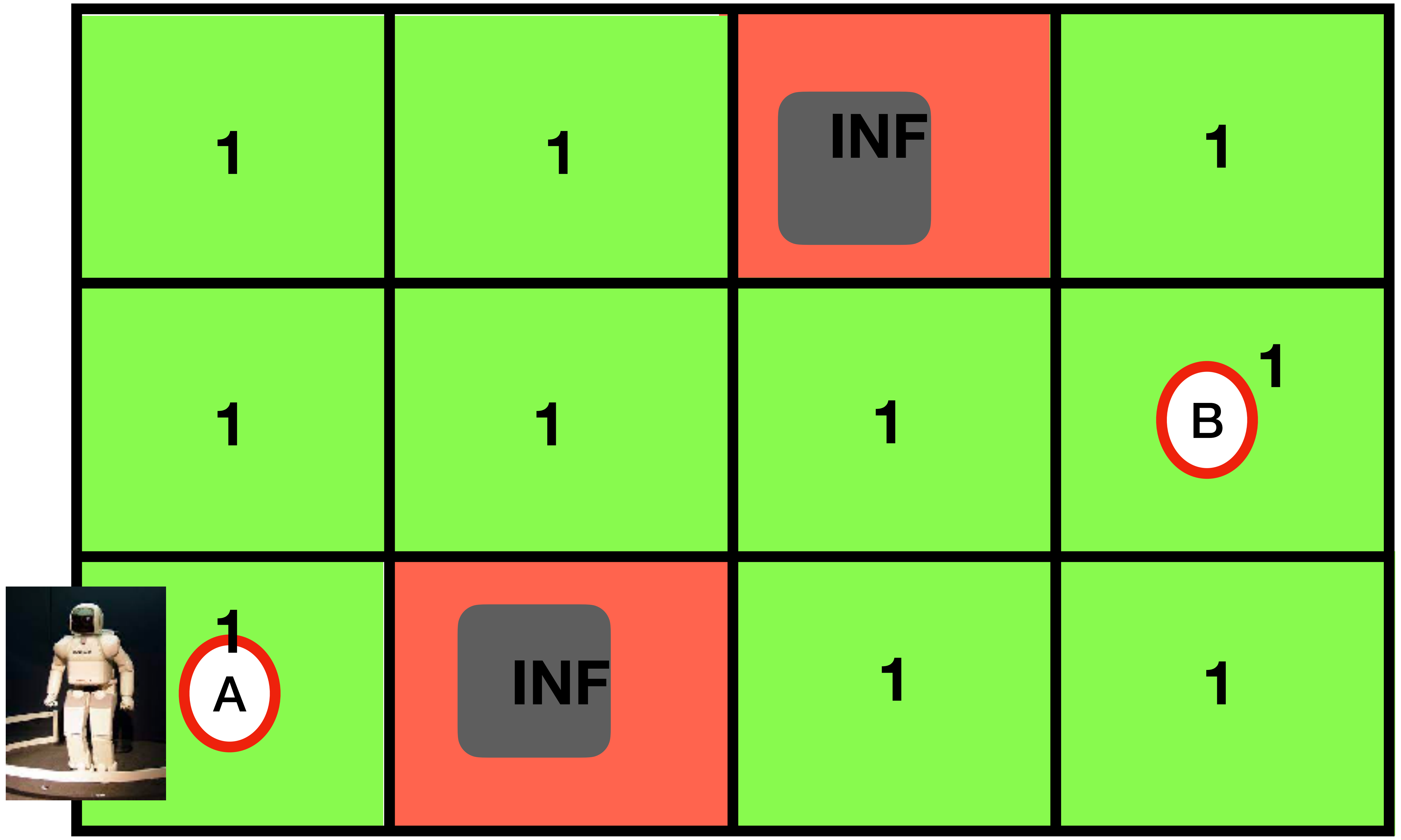


Task: Move from A to B cheaply

1. Plan
2. Execute



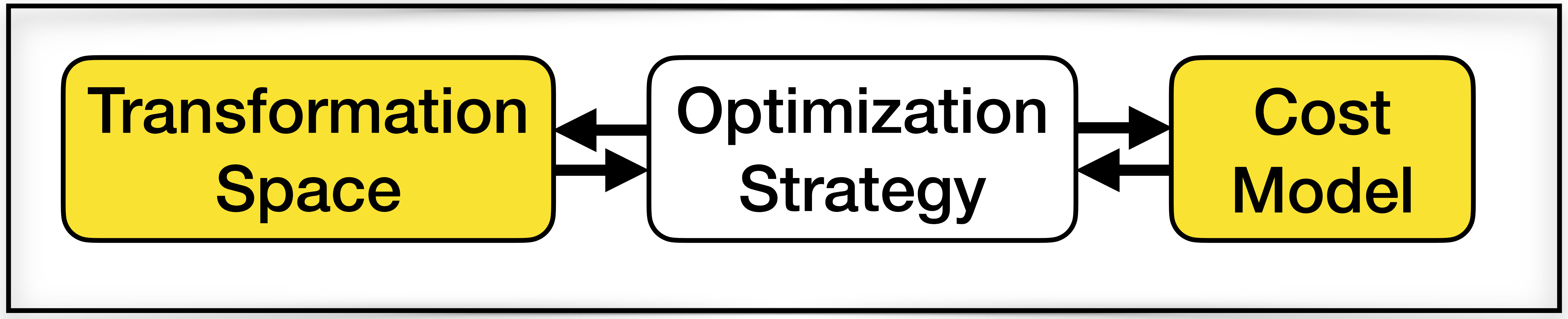
Robot Analogy



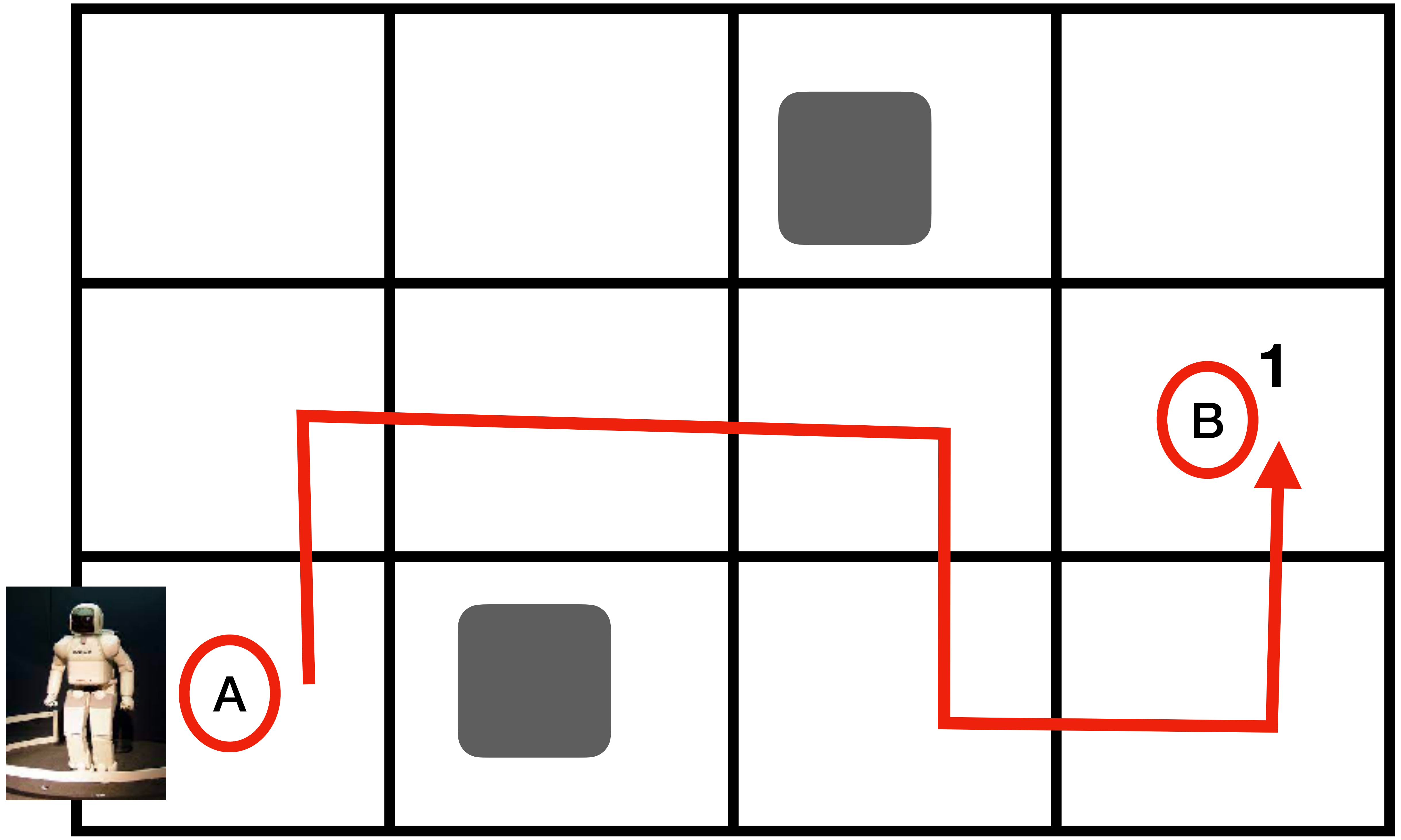
Task: Move from A to B cheaply

1. Plan

2. Execute



Robot Analogy

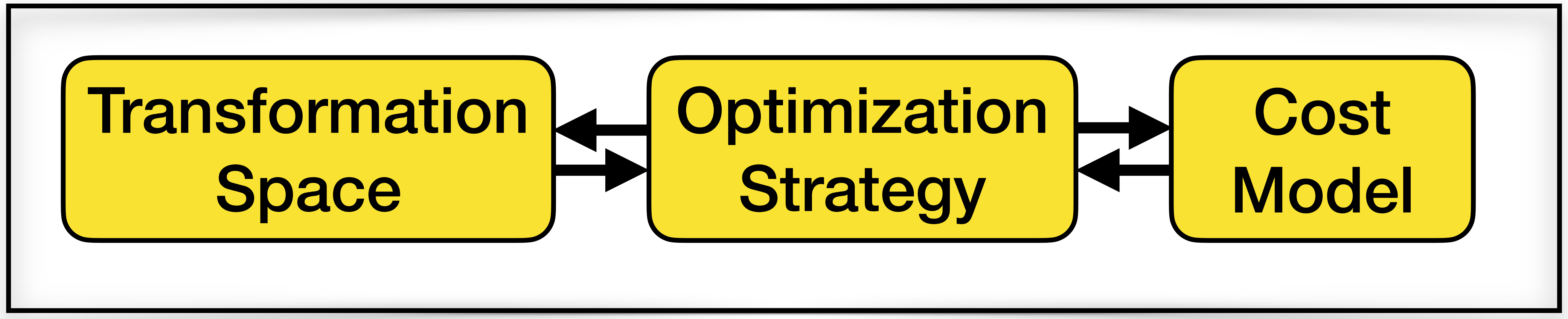


Task: Move from A to B cheaply

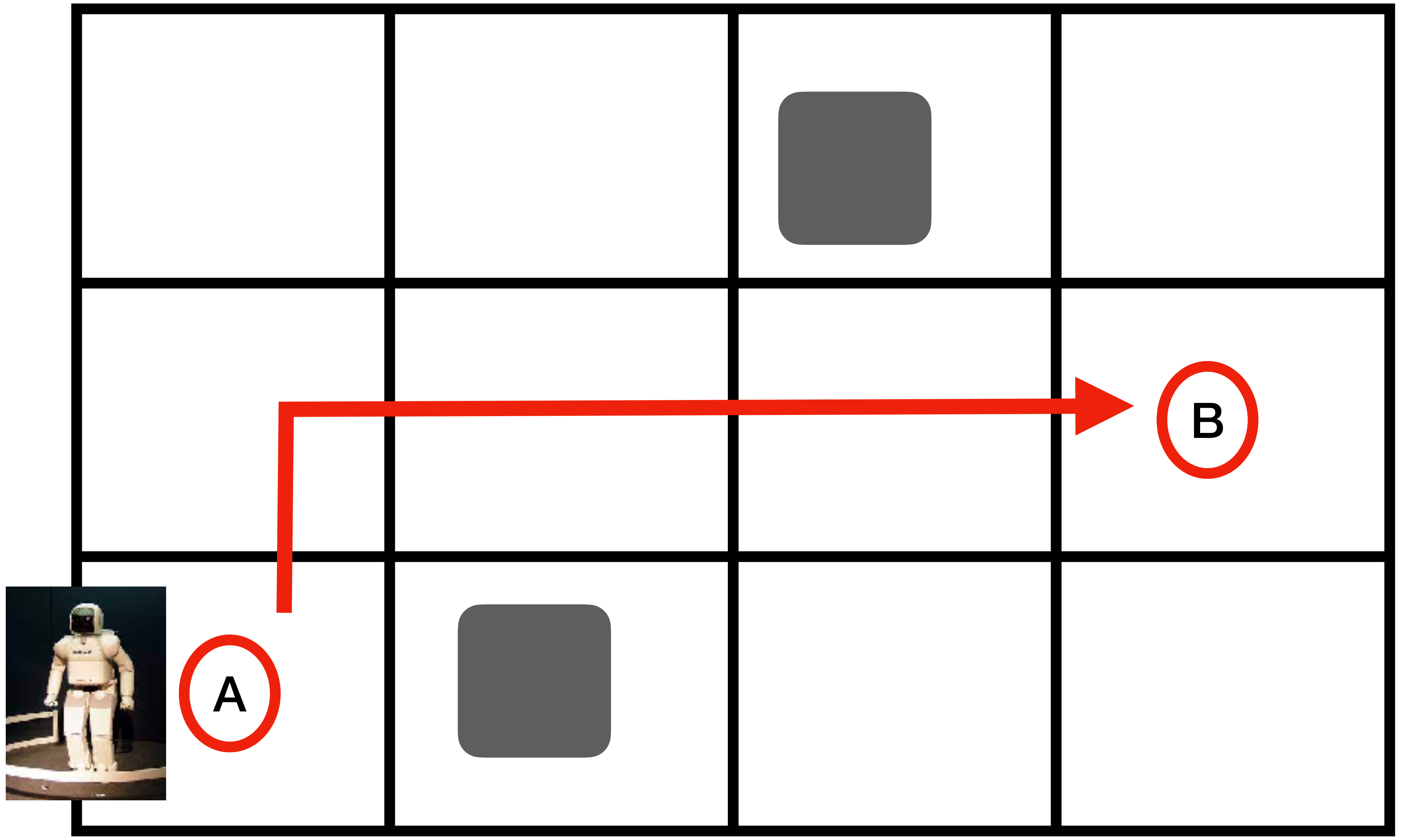
1. Plan

2. Execute

Cost: 7



Robot Analogy

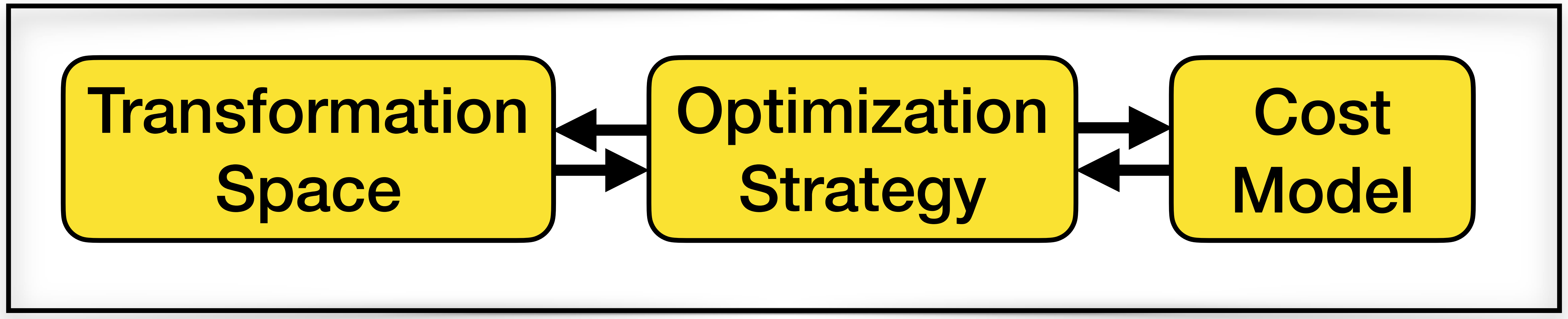


Task: Move from A to B cheaply

1. Plan

2. Execute

Cost: 5



Vectorization

Independent and **Isomorphic** statements can be vectorized

Scalar Code

```
a[0] = b[0] + c[0]
a[1] = b[1] + c[1]
```

Vector Packs

Vector Code

Single **I**nstruction **M**ultiple **D**ata (SIMD)

```
{a[0],a[1]} = {b[0],b[1]} + {c[0],c[1]}
```

Larsen & Amarasinghe "Exploiting Superword Level Parallelism with Multimedia Instruction Sets" [PLDI'00]

Statement Packing Problem

- Find **independent** and **isomorphic** statements
- Not all vector packs can exist with each other
- Need to select the most profitable packing strategy

```
S1 : A1 = L[5] / L[2]
S2 : A2 = L[6] / L[3]
S3 : A3 = L[7] / L[4]
S4 : A4 = L[1] - A2
S5 : A5 = L[2] - A3
S6 : A6 = L[3] - A1
```

```
{S1,S2}    {S4,S5}
{S2,S3}    {S5,S6}
{S1,S3}    {S4,S6}
```

Transformation Space

Statement packing strategy 1

Scalar code

```
S1 : A1 = L[5] / L[2]
S2 : A2 = L[6] / L[3]
S3 : A3 = L[7] / L[4]
S4 : A4 = L[1] - A2
S5 : A5 = L[2] - A3
S6 : A6 = L[3] - A1
```

Vector code

```
S1  : A1 = L[5] / L[2]
S2  : A2 = L[6] / L[3]

S3  : A3 = L[7] / L[4]

S4  : A4 = L[1] - A2
S5  : A5 = L[2] - A3
S6  : A6 = L[3] - A1
```

Instruction Breakdown

0 vector
0 packing
0 unpacking

There are costs associated with vectorization

Scalar code

```
S1 : A1 = L[5] / L[2]
S2 : A2 = L[6] / L[3]
S3 : A3 = L[7] / L[4]
S4 : A4 = L[1] - A2
S5 : A5 = L[2] - A3
S6 : A6 = L[3] - A1
```

Vector code

```
SV1 : {A1,A2} = {L[5],L[6]} / {L[2],L[3]}

S3  : A3 = L[7] / L[4]

S4  : A4 = L[1] - A2
SV2 : {A5,A6} = {L[2],L[3]} - {A3,A1}
```

Non-isomorphic

Instruction Breakdown

4 vector
0 packing
0 unpacking

There are costs associated with vectorization

Scalar code

```
S1 : A1 = L[5] / L[2]
S2 : A2 = L[6] / L[3]
S3 : A3 = L[7] / L[4]
S4 : A4 = L[1] - A2
S5 : A5 = L[2] - A3
S6 : A6 = L[3] - A1
```

Vector code

```
SV1 : {A1,A2} = {L[5],L[6]} / {L[2],L[3]}
SU1 : A1 = unpack(SV1,1)

S3  : A3 = L[7] / L[4]

S4  : A4 = L[1] - A2
SV2 : {A5,A6} = {L[2],L[3]} - {A3,A1}
```

Instruction Breakdown

4 vector
0 packing
1 unpacking

There are costs associated with vectorization

Scalar code

```
S1 : A1 = L[5] / L[2]
S2 : A2 = L[6] / L[3]
S3 : A3 = L[7] / L[4]
S4 : A4 = L[1] - A2
S5 : A5 = L[2] - A3
S6 : A6 = L[3] - A1
```

Vector code

```
SV1 : {A1,A2} = {L[5],L[6]} / {L[2],L[3]}
SU1 : A1 = unpack(SV1,1)

S3  : A3 = L[7] / L[4]
SP1 : {A3,A1} = pack(A3,A1)
S4  : A4 = L[1] - A2
SV2 : {A5,A6} = {L[2],L[3]} - {A3,A1}
```

Instruction Breakdown

4 vector
1 packing
1 unpacking

There are costs associated with vectorization

Scalar code

```
S1 : A1 = L[5] / L[2]
S2 : A2 = L[6] / L[3]
S3 : A3 = L[7] / L[4]
S4 : A4 = L[1] - A2
S5 : A5 = L[2] - A3
S6 : A6 = L[3] - A1
```

Vector code

```
SV1 : {A1,A2} = {L[5],L[6]} / {L[2],L[3]}
SU1 : A1 = unpack(SV1,1)
SU2 : A2 = unpack(SV1,2)
S3  : A3 = L[7] / L[4]
SP1 : {A3,A1} = pack(A3,A1)
S4  : A4 = L[1] - A2
SV2 : {A5,A6} = {L[2],L[3]} - {A3,A1}
```

Instruction Breakdown

- 4 vector
- 1 packing
- 2 unpacking

Statement packing strategy 2

Scalar code

```
S1  : A1 = L[5] / L[2]
S2  : A2 = L[6] / L[3]
S3  : A3 = L[7] / L[4]
S4  : A4 = L[1] - A2
S5  : A5 = L[2] - A3
S6  : A6 = L[3] - A1
```

Vector code

```
SV1 : {A2,A3} = {L[6],L[7]} / {L[3],L[4]}
SU1 : L[2] = unpack(SLV1,2)
S1  : A1 = L[5] / L[2]
SU2 : L[3] = unpack(SLV2,1)
SV2 : {A4,A5} = {L[1],L[2]} - {A2,A3}
S6  : A6 = L[3] - A1
```

Instruction Breakdown

5 vector
0 packing
2 unpacking

Different vectorization schemes have different profitability

Strategy 1

Liu et. al [PLDI'12]

```
SV1 : {A1,A2} = {L[5],L[6]} / {L[2],L[3]}
SU1 : A1 = unpack(SV1,1)
SU2 : A2 = unpack(SV1,2)
S3  : A3 = L[7] / L[4]
SP1 : {A3,A1} = pack(A3,A1)
S4  : A4 = L[1] - A2
SV2 : {A5,A6} = {L[2],L[3]} - {A3,A1}
```

4 vector
1 packing
2 unpacking

Strategy 2

Optimal

```
SV1 : {A2,A3} = {L[6],L[7]} / {L[3],L[4]}
SU1 : L[2] = unpack(SLV1,2)
S1  : A1 = L[5] / L[2]
SU2 : L[3] = unpack(SLV2,1)
SV2 : {A4,A5} = {L[1],L[2]} - {A2,A3}
S6  : A6 = L[3] - A1
```

5 vector
0 packing
2 unpacking

Machine Learning Influence

	Transformation Space	Optimization Strategy	Cost Model
Traditional solutions ↓	Hand-written	• Greedy / Heuristic • Integer Linear Programming • Dynamic Programing	• Analytical Linear Non-linear
Automated solutions	**Program Logics**	**Data-driven** **Imitation Learning** **10/19**	**Data-driven** **LSTM based Cost Model** **10/05** **(related reading)**

Domain Specific Languages

- Programming model specific to one domain
  - Image / Array Processing - Halide, MATLAB
  - Sparse Tensor Computations - TACO
  - Tensor Algebra - Tensorflow, Pytorch (frameworks)
  - Graphs - GraphIt, Gunrock
  - Genomic Computations - Seq

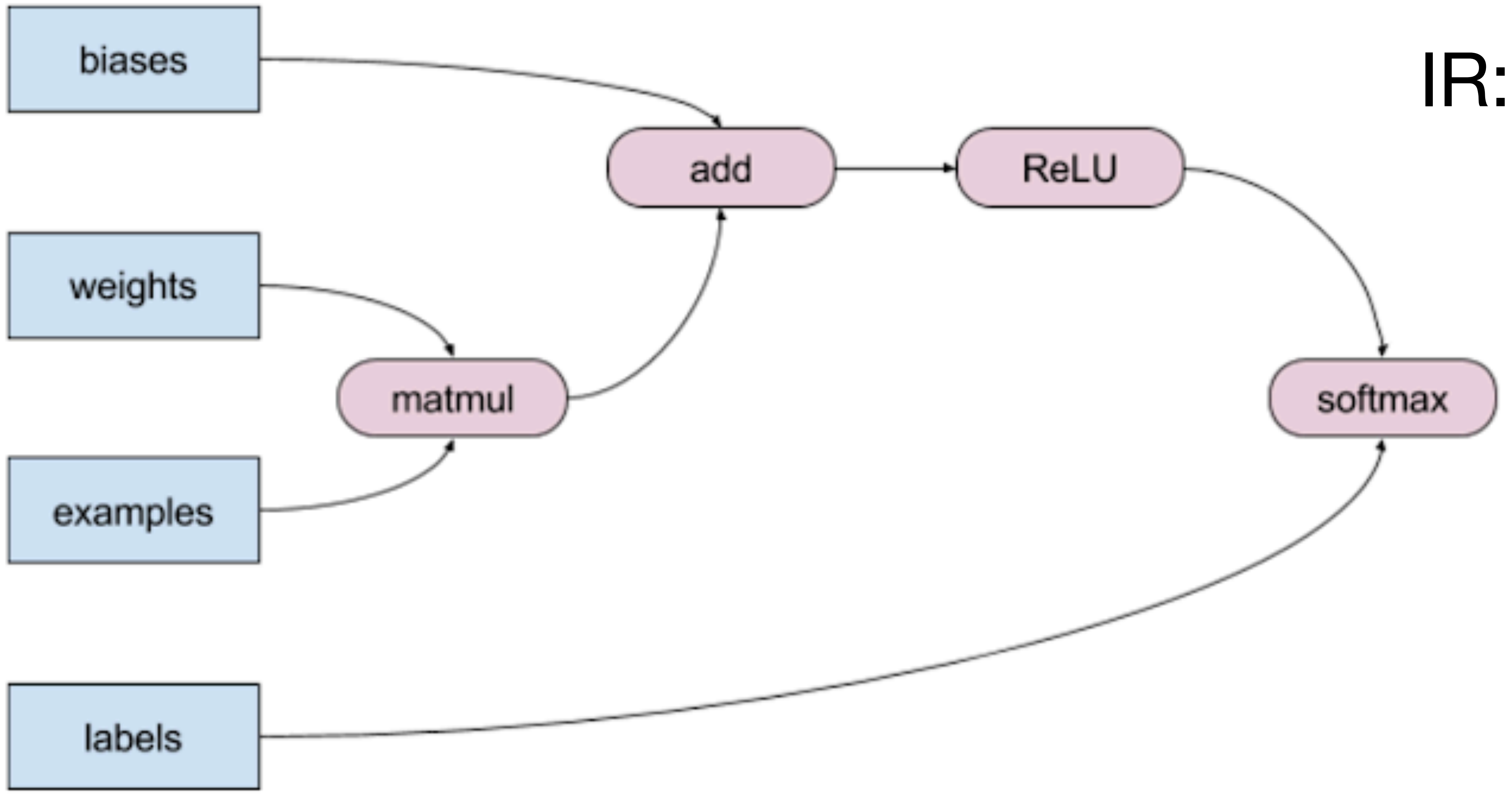
- Usually comes with a set of domain specific optimizations

Halide

- **Main idea:** Separate algorithm specification from optimizations (schedules)
- Halide Video
  - https://www.youtube.com/watch?v=3uiEyEKji0M&t=3s
- **Optimization objective:** find the best schedule or optimization sequence for a given Halide algorithm

Tensorflow

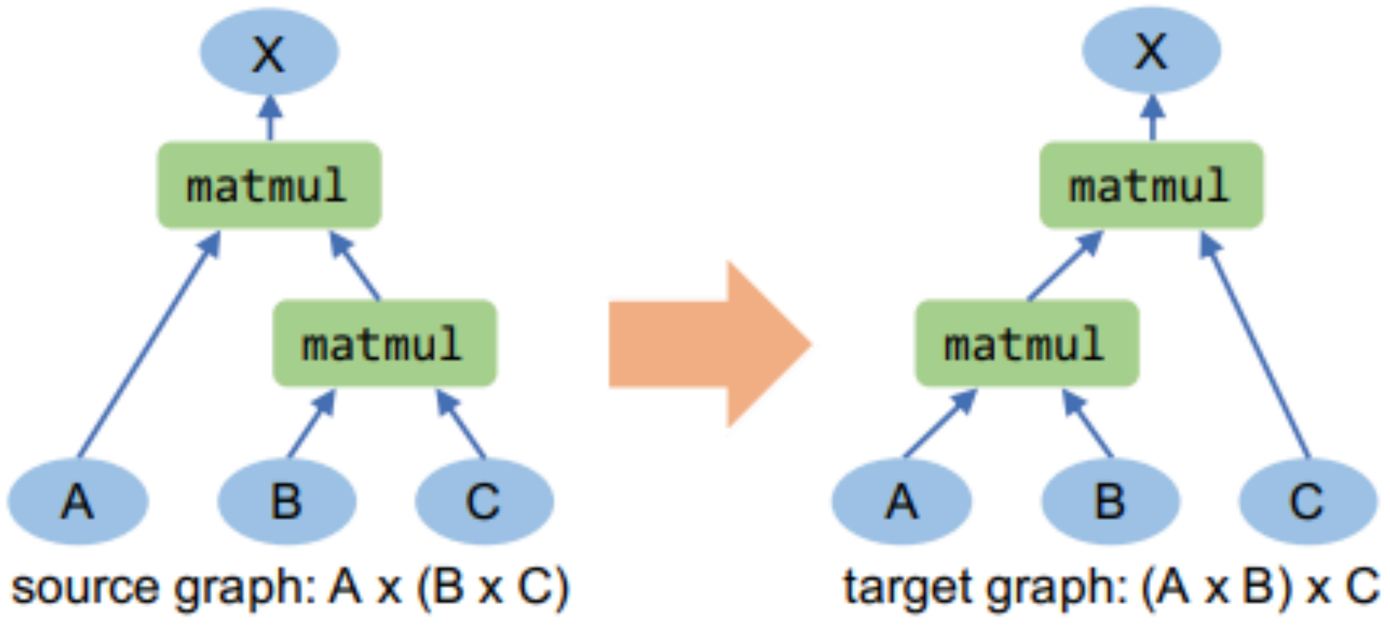
- Model tensor manipulating programs
- Uses the XLA compiler to target GPUs, TPUs and CPUs
- Main abstraction: Computational Graphs



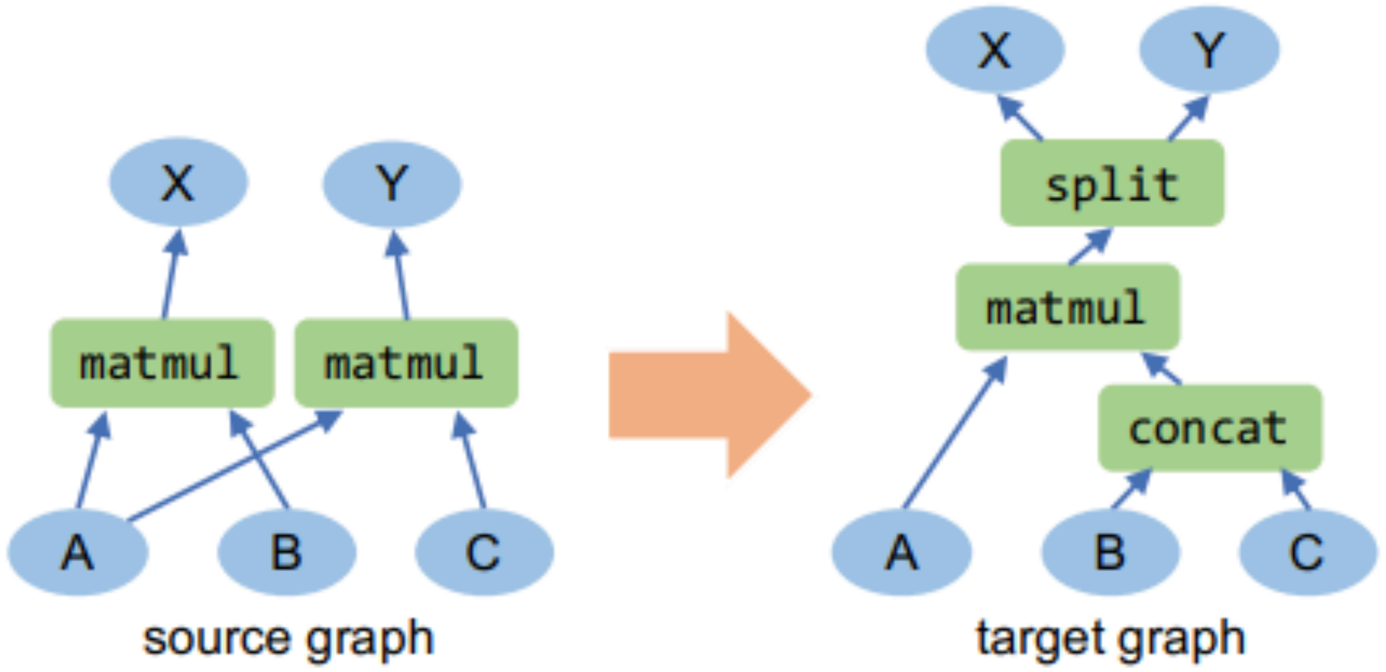
IR: High Level Operations (HLO)

XLA Compiler

- (Most) optimizations can be expressed as computational graph rewrites



(a) Associativity of matrix multiplication.



(b) Fusing two matrix multiplications using concatenation and split.

TASO [SOSP'19]

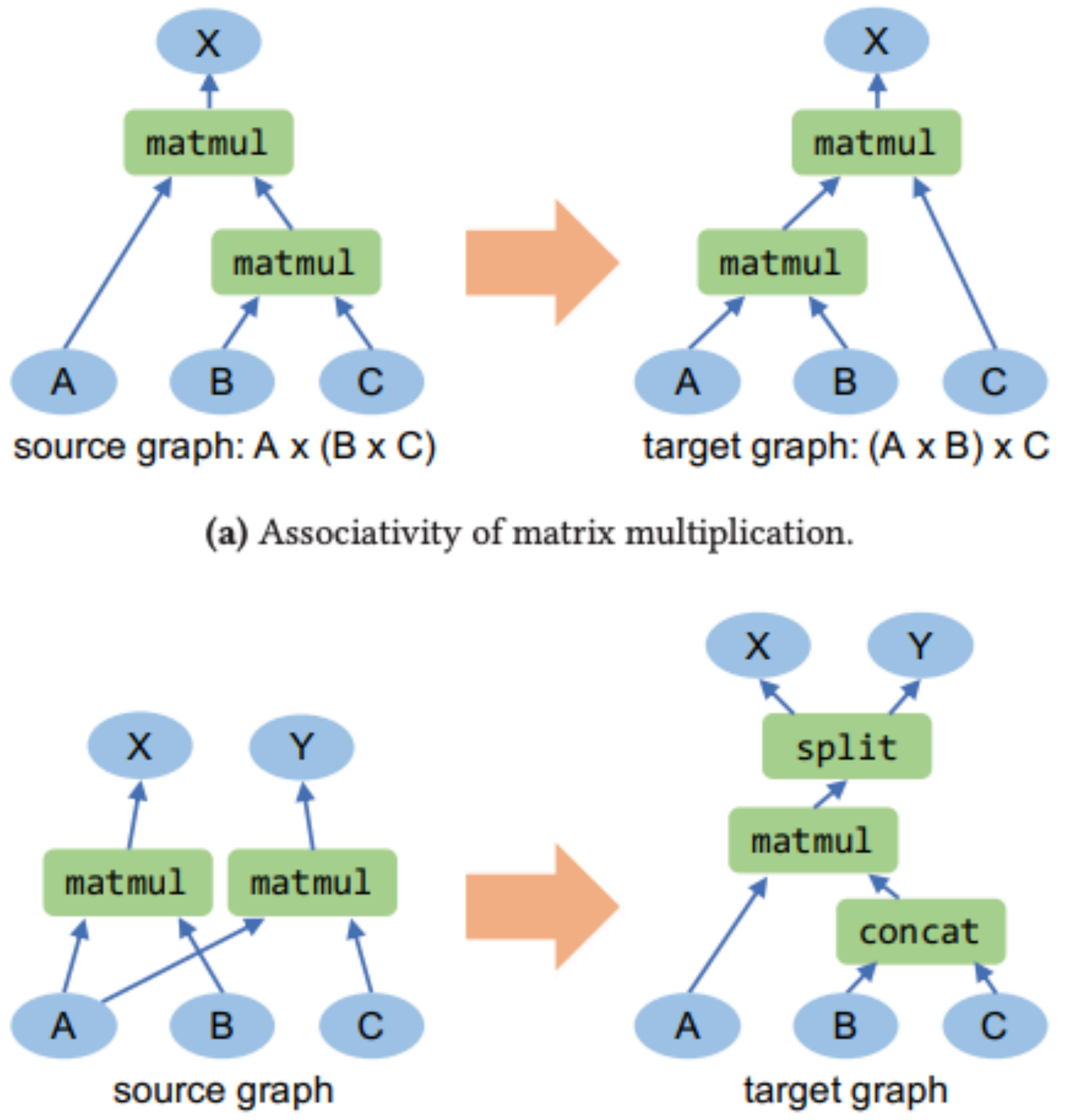
https://cs.stanford.edu/~padon/taso-sosp19.pdf

Machine Learning Influence

	Transformation Space	Optimization Strategy	Cost Model
Automated solutions	**Program Logics**	**Data-driven**	**Data-driven**
		10/26: Tree Search (Halide) **10/31: Gradient-based Methods (TVM)**	**10/10: GNN based Cost Model (XLA)**

XLA Compiler

- (Most) optimizations can be expressed as computational graph rewrites



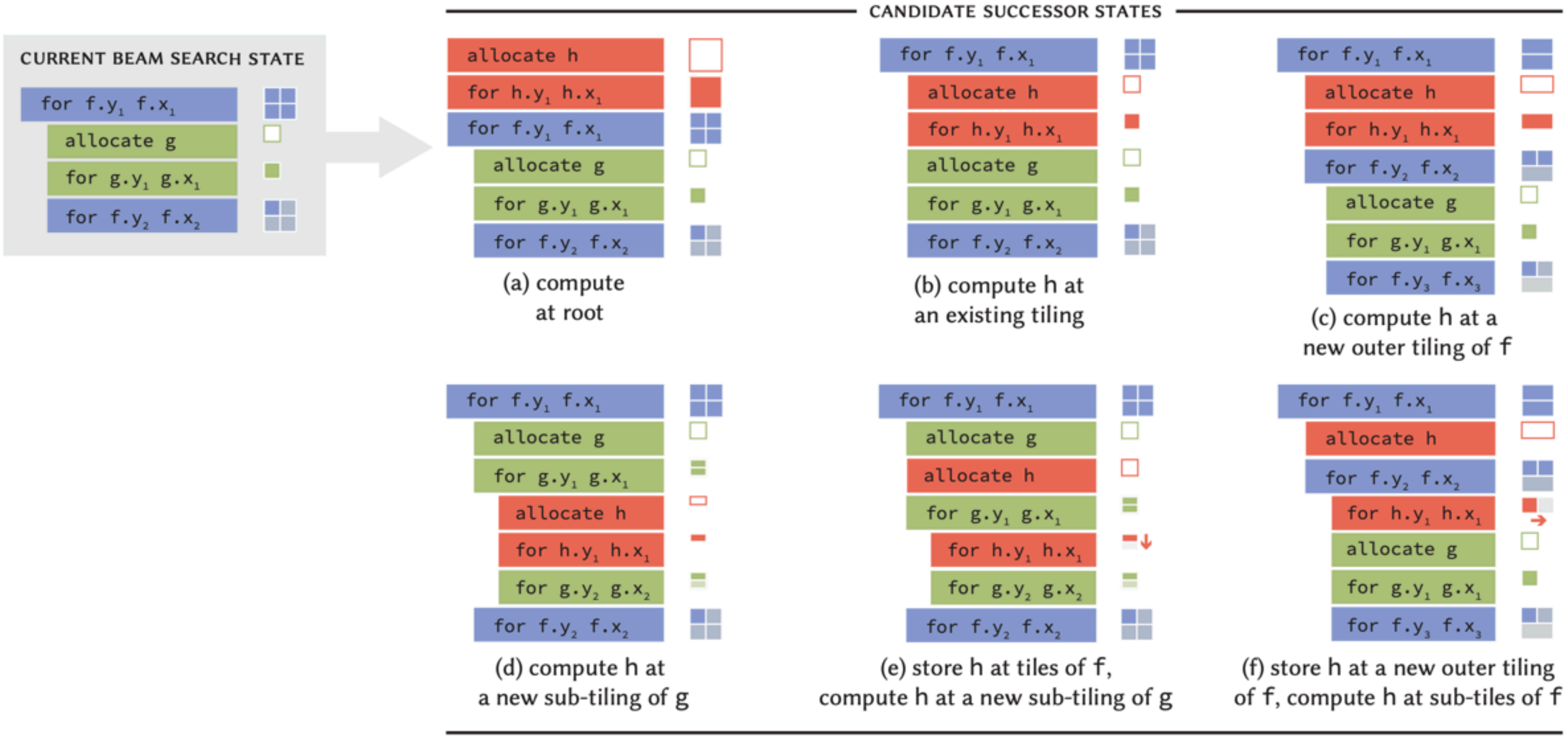
(a) Associativity of matrix multiplication.

(b) Fusing two matrix multiplications using concatenation and split.

- What if there are 2 or rewrites that can be performed at the same time?

Halide

- **Optimization objective:** find the best schedule or optimization sequence for a given Halide algorithm



Paper Presentation

- Paper presentations assigned on September 4th
- Week before: Meet instructor to discuss the presentation plan (compulsory!)
  - Use this time to ask questions and discuss the outline
  - Presentation slides are due when reviews are due for that class
  - Submit using the hotCRP system
- During the class: Be present in class (compulsory!)
  - Deliver a 30 min presentation on the paper
  - Answer questions for the following 20 min
  - Final 25 min for open discussion on the paper (lead by the instructor)

Paper Presentation

- After class: Summarize the discussion of the paper
  - Submit the summary by the start of the next class
- First presentation on September 12th
  - Whaley and Dongarra, "Automatically Tuned Linear Algebra Software" (SC 1998)
  - 30 min presentation
  - https://amturing.acm.org/award_winners/dongarra_3406337.cfm

Any Questions?