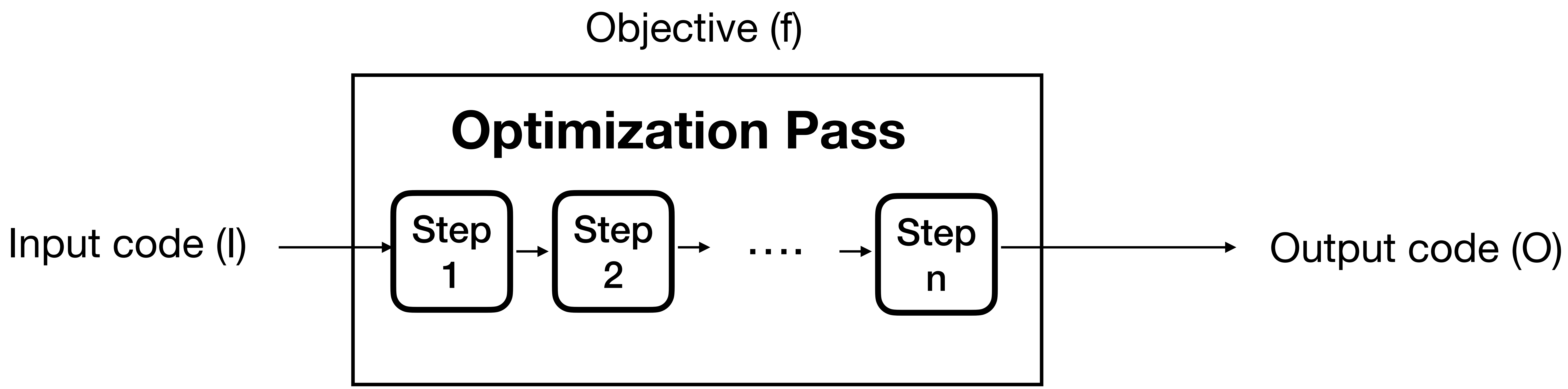
CS 526

Advanced Compiler Construction

Machine Learning in Compilers

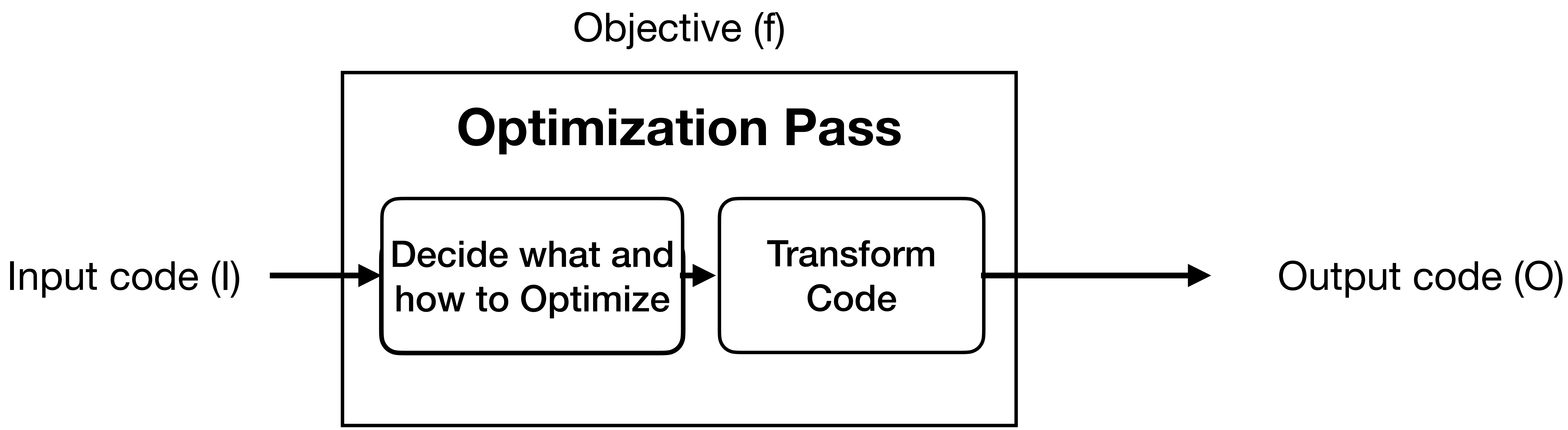
Anatomy of an Optimization Pass



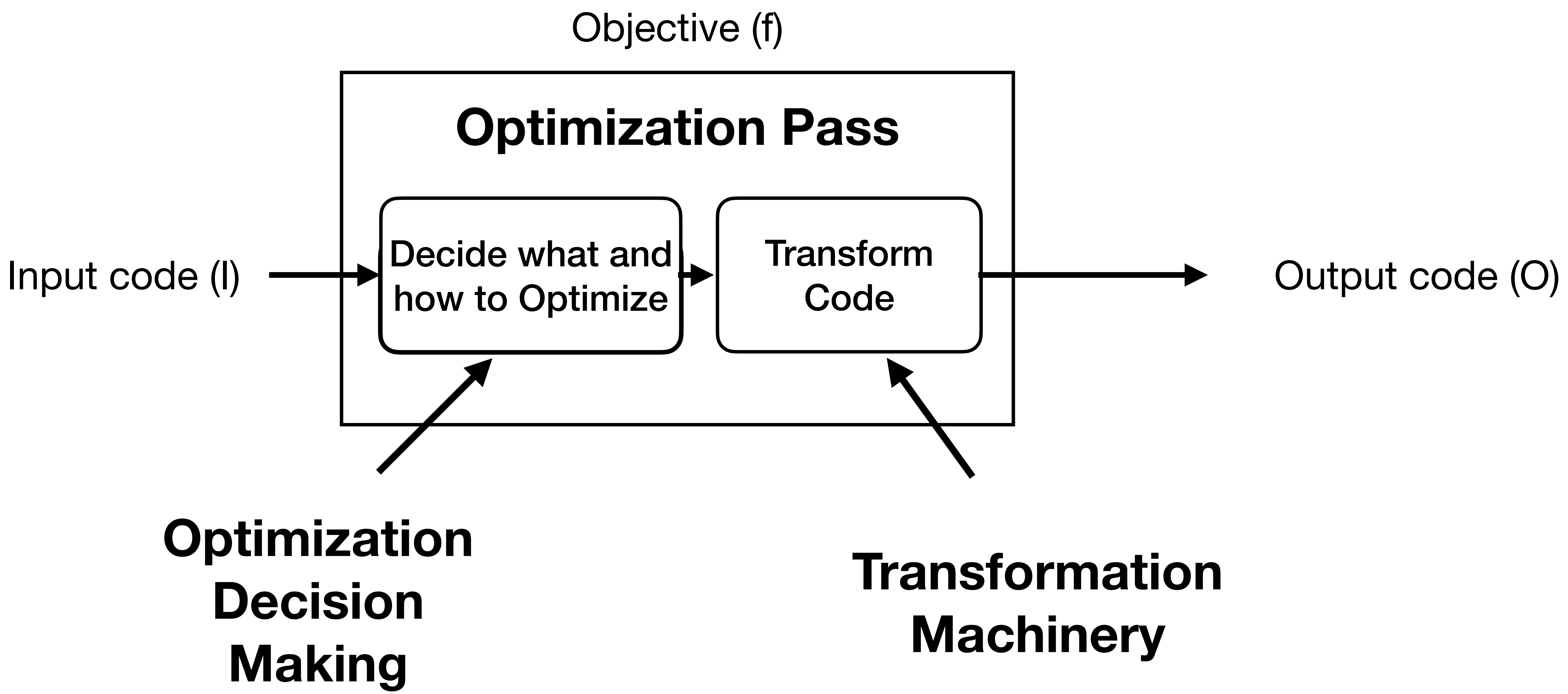
What are possible objectives?

- Produce Correct Code (semantic equivalence)
- Produce Fast Code
- Produce Energy-efficient Code
- Produce secure code

Anatomy of an Optimization Pass

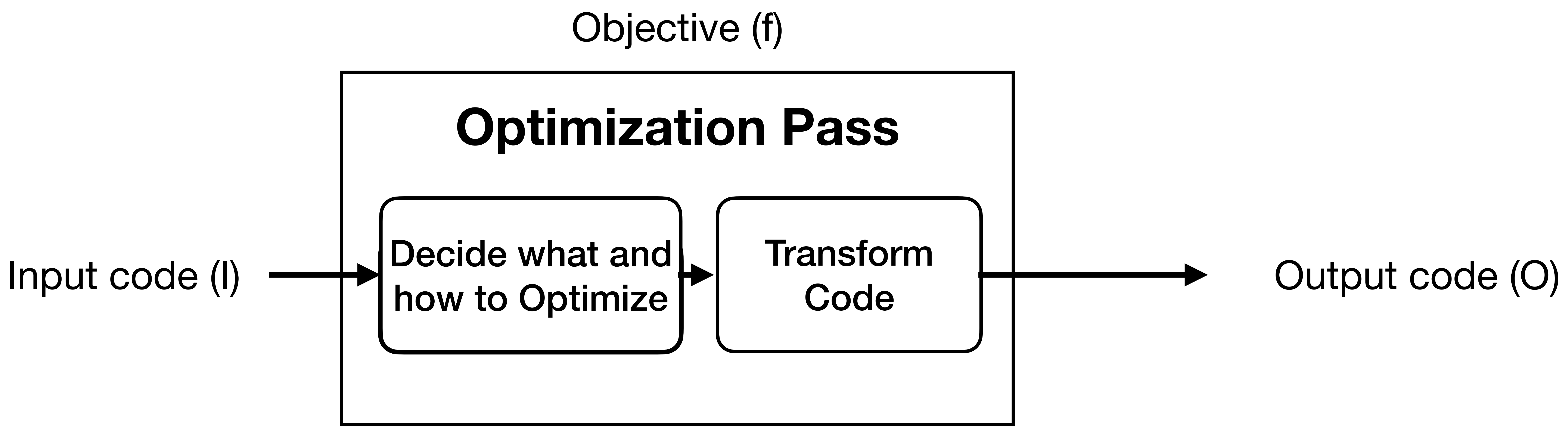


Anatomy of an Optimization Pass



Goal: `f(O) > f(I)`

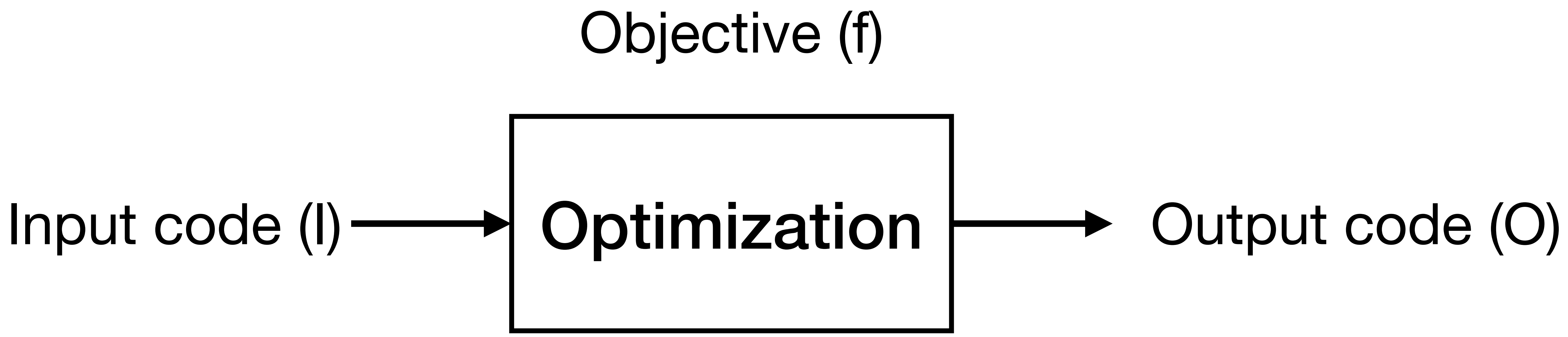
Anatomy of an Optimization Pass



- Find Dead Code
- Decide on a set of loop transformations
- Decide where to inline

Generate the Code!

Two types of Optimizations



`Type I`

- Steps are always Profitable
  `f(O) > f(I)`
- Mostly independent

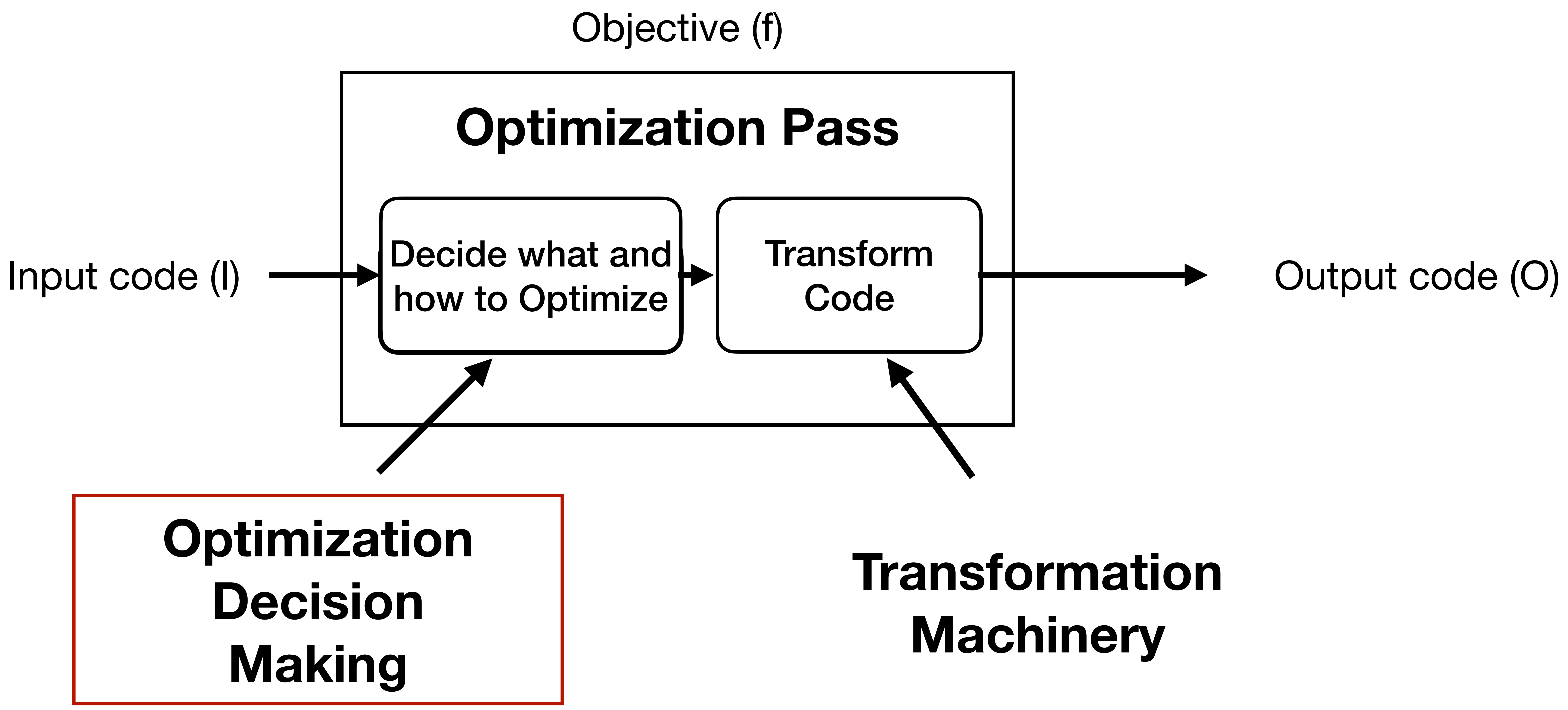
`Type II`

- Steps may not lead to global profitability
  `f(O) > f(I) ??`
- Mostly mutually-exclusive

Let's try to categorize

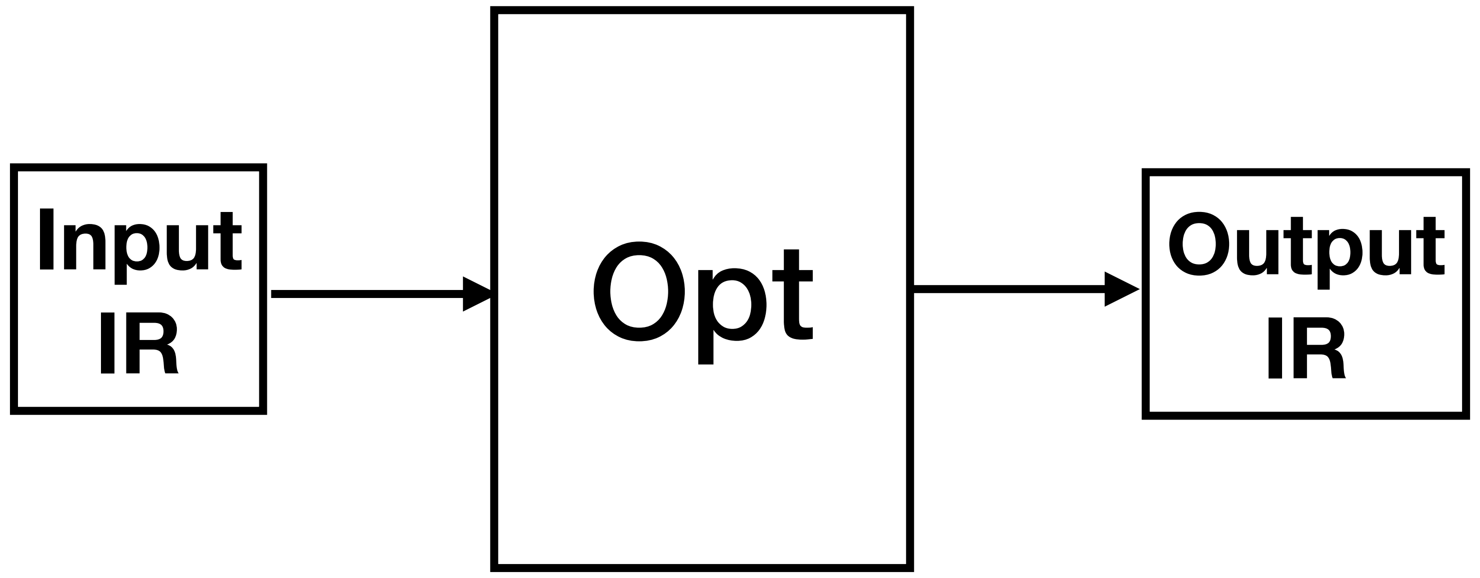
- Dead-Code elimination
- Sparse Conditional Constant Propagation
- Global Value Numbering
- Inlining
- Loop Transformations (interchange, tiling etc.)
- Vectorization
- Peephole Optimizations
- Automatic Parallelizations

Anatomy of an Optimization Pass



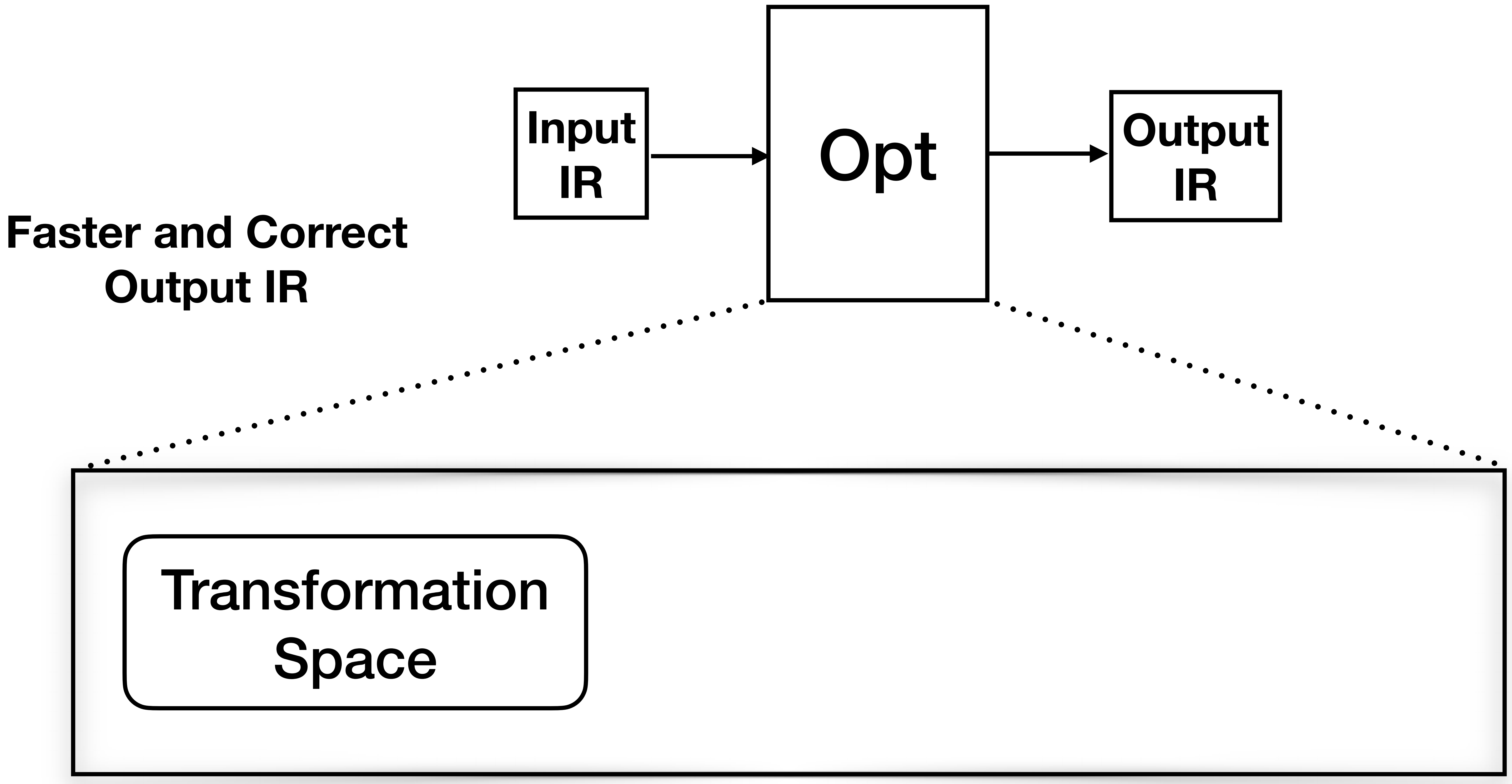
Goal: `f(O) > f(I)`

Optimization Decision Making



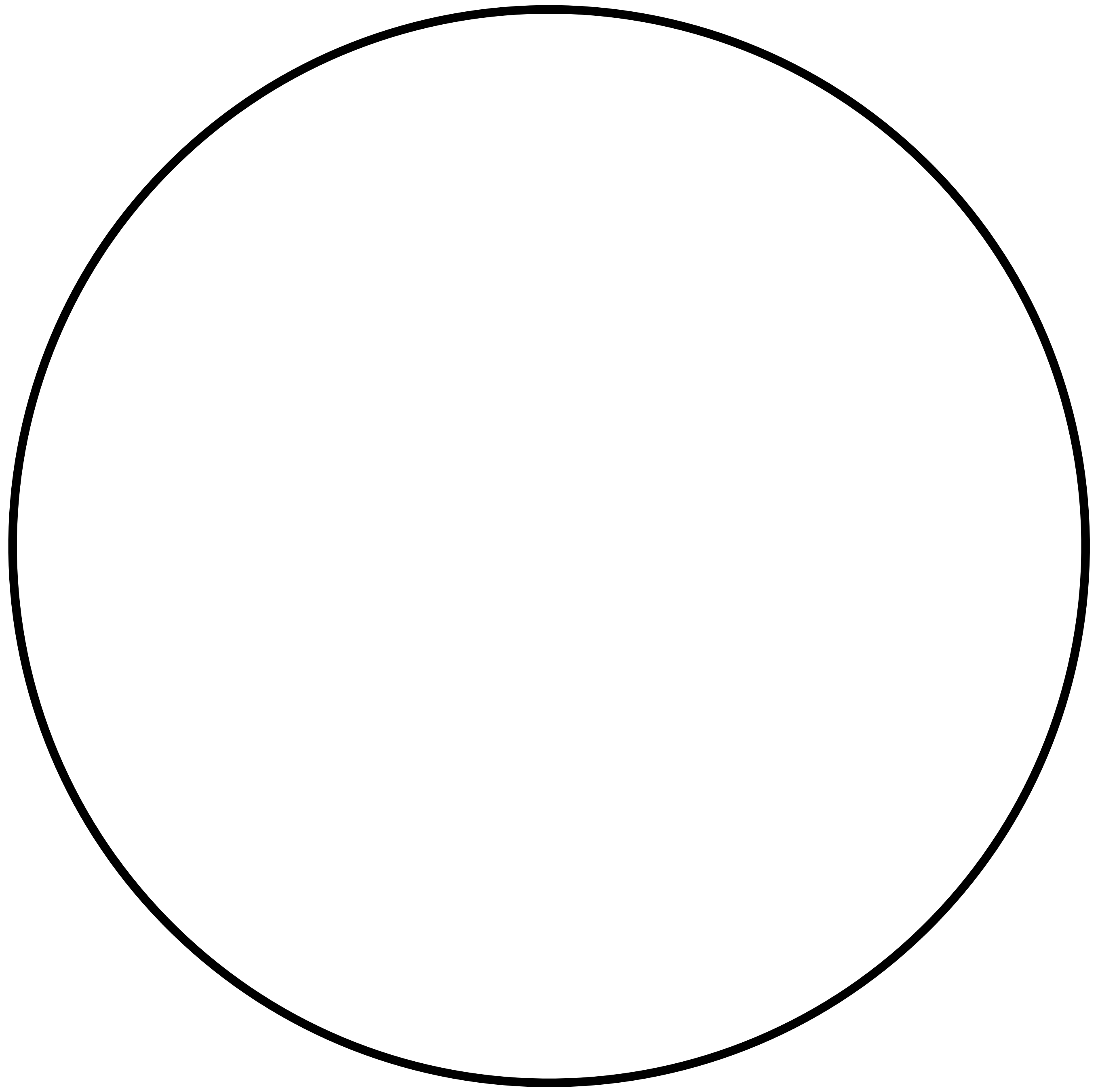
Faster and Correct Output IR

Optimization Decision Making

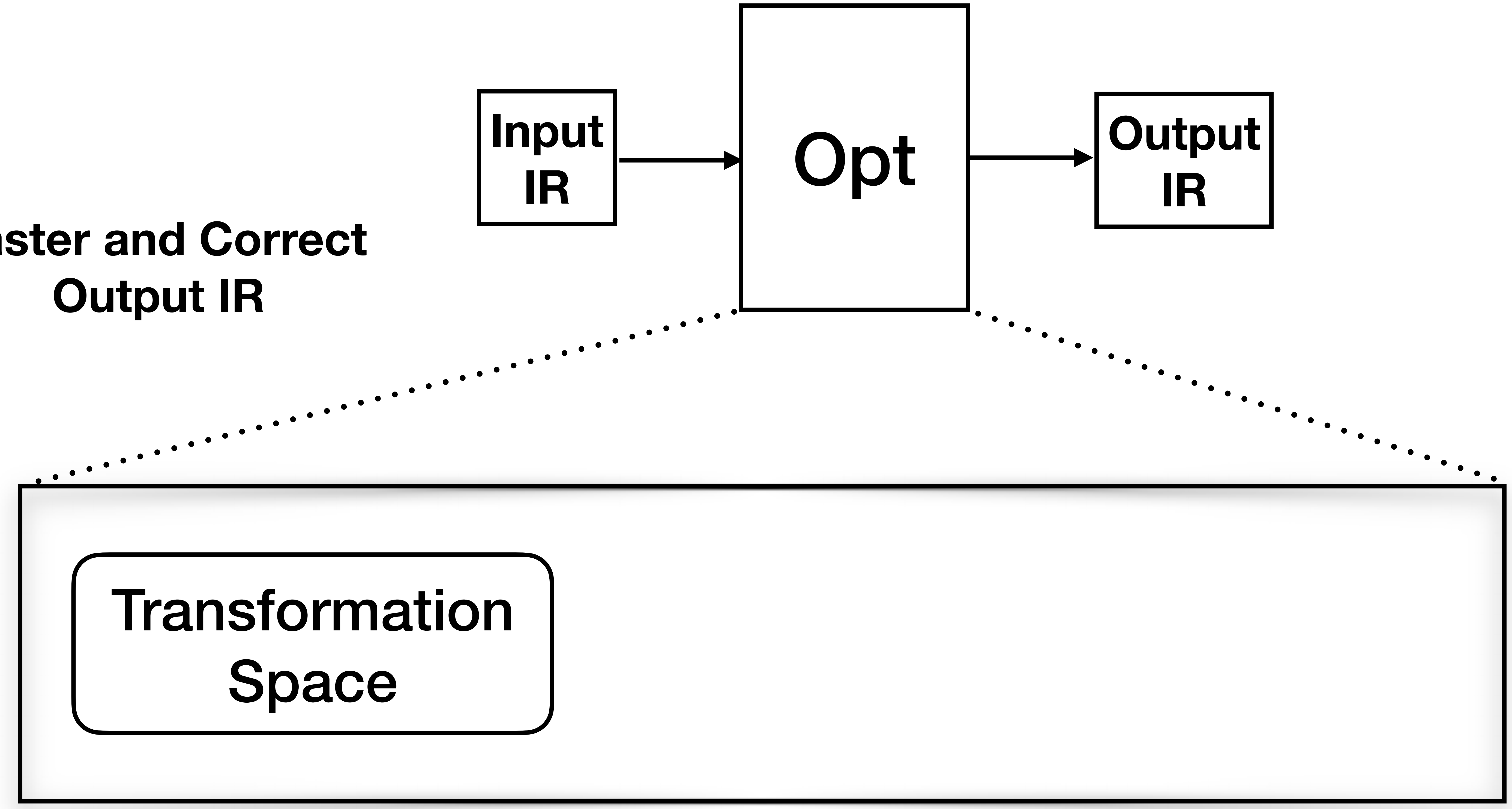


Optimization Decision Making

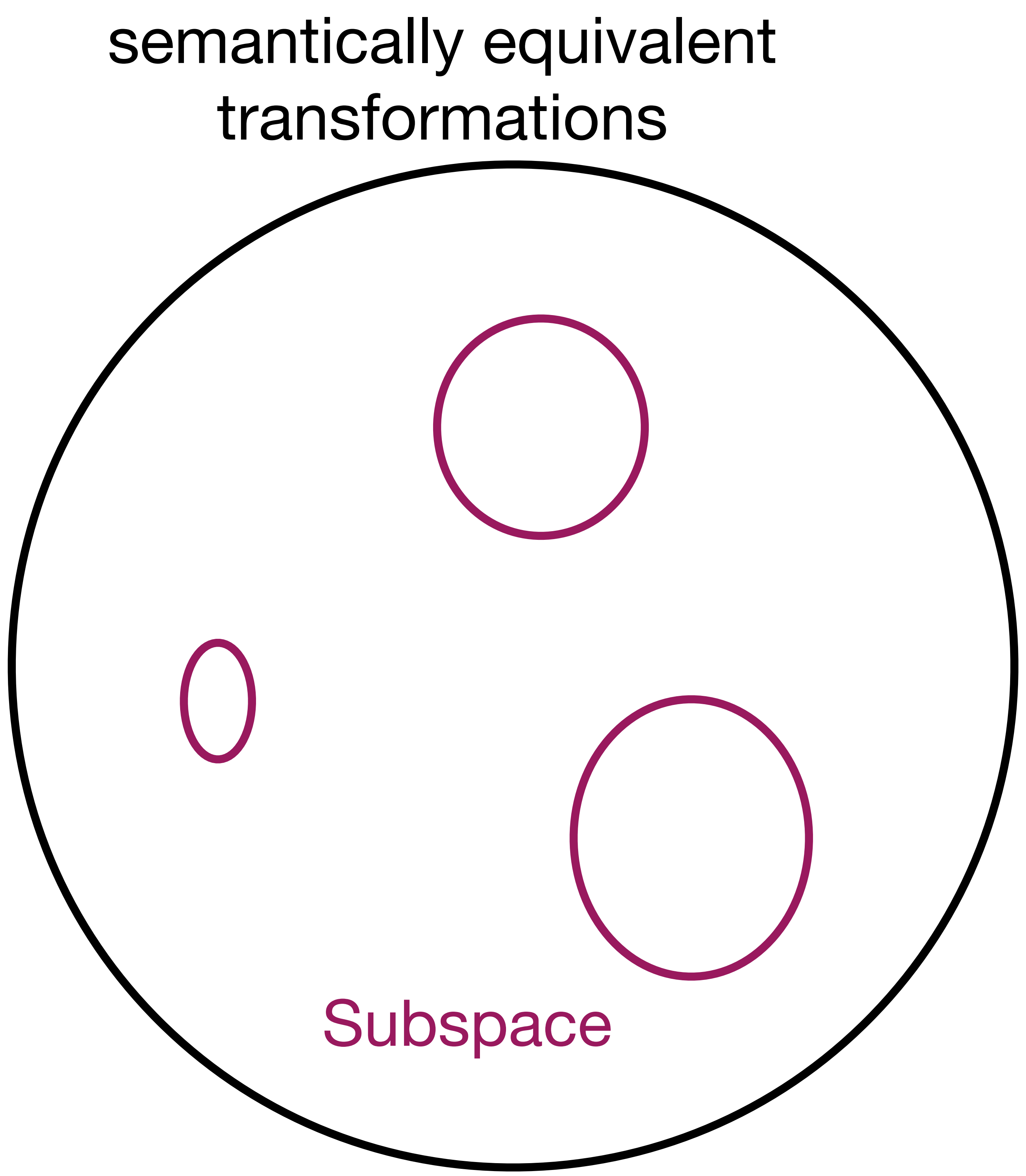
semantically equivalent transformations

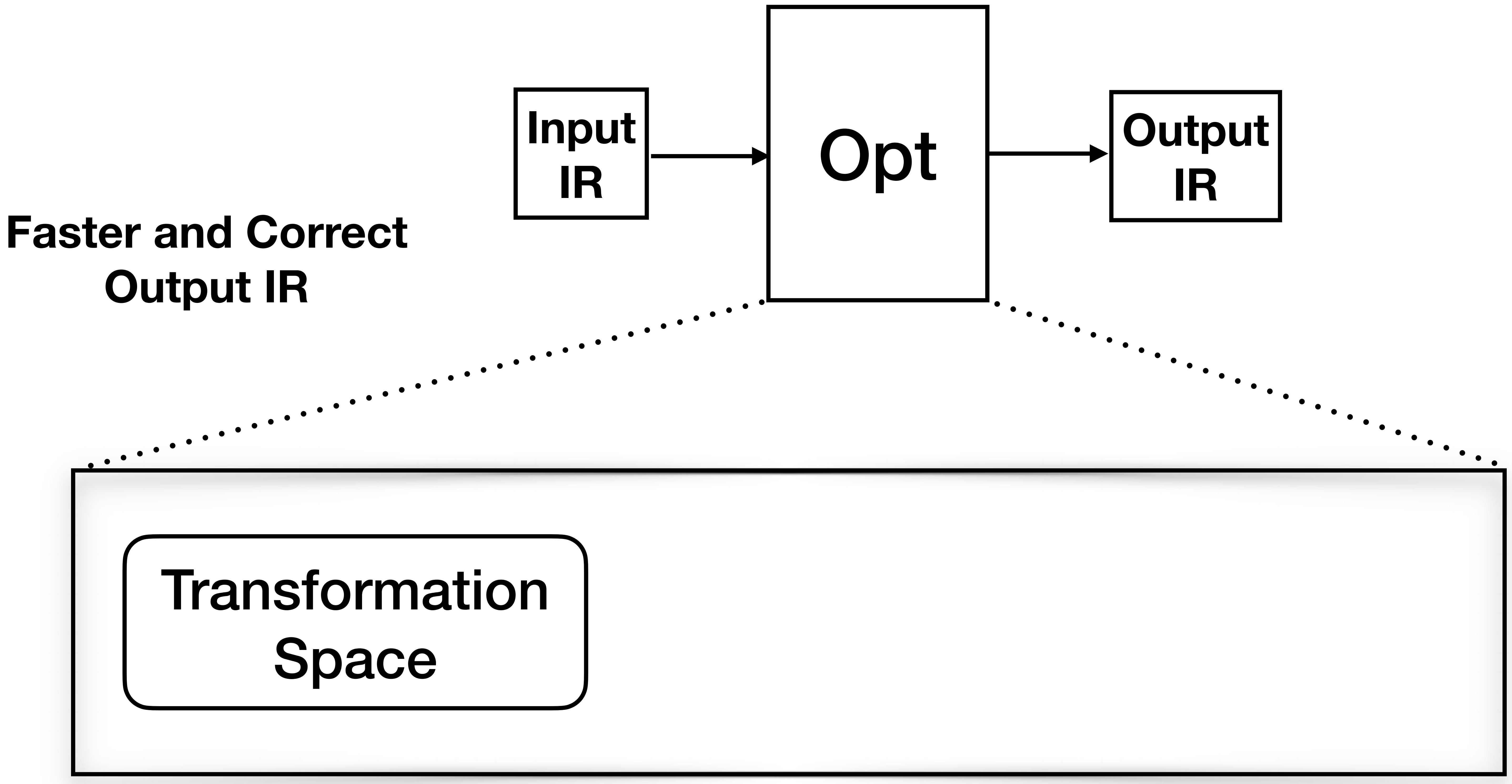


Faster and Correct Output IR

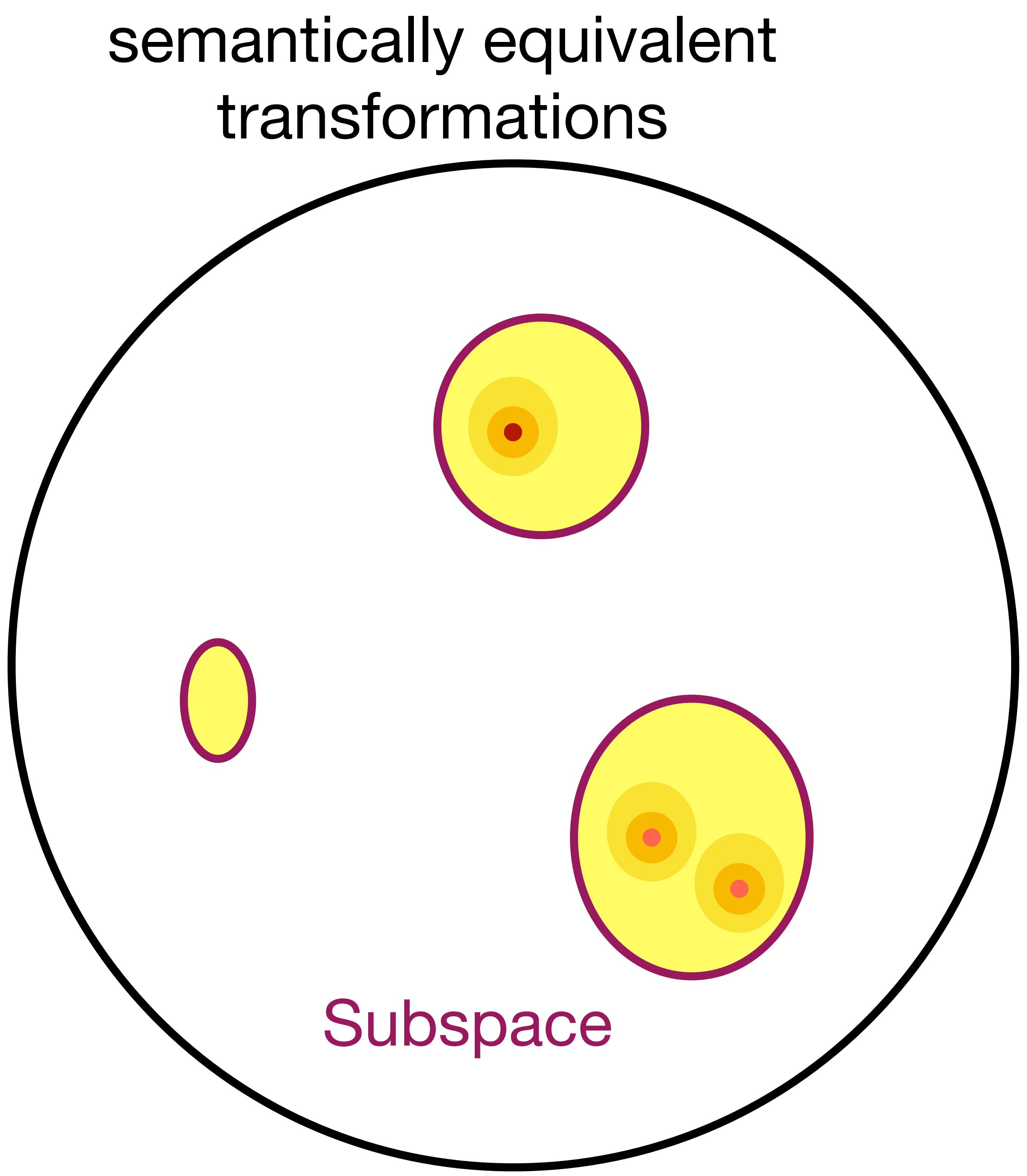


Optimization Decision Making

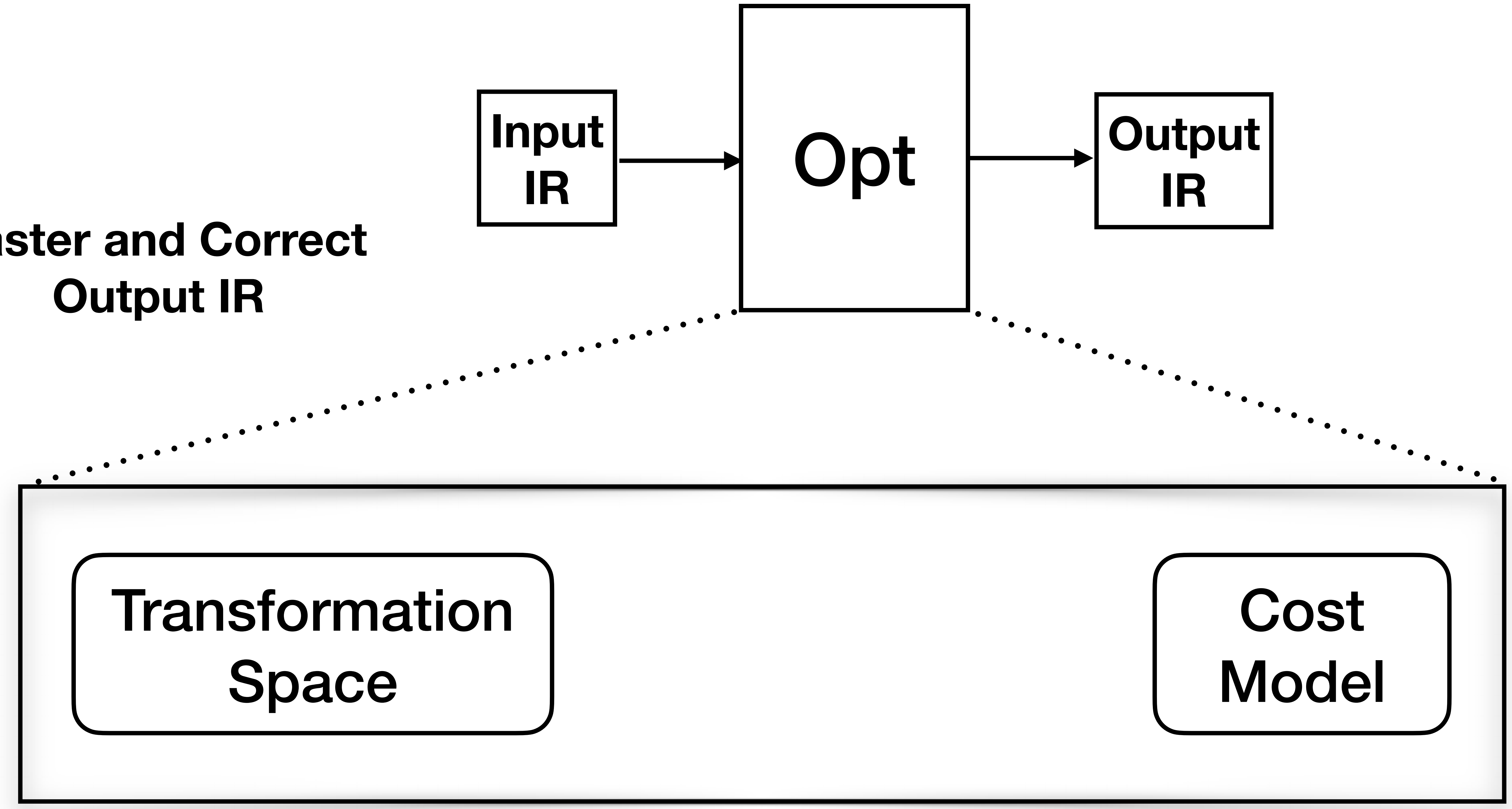




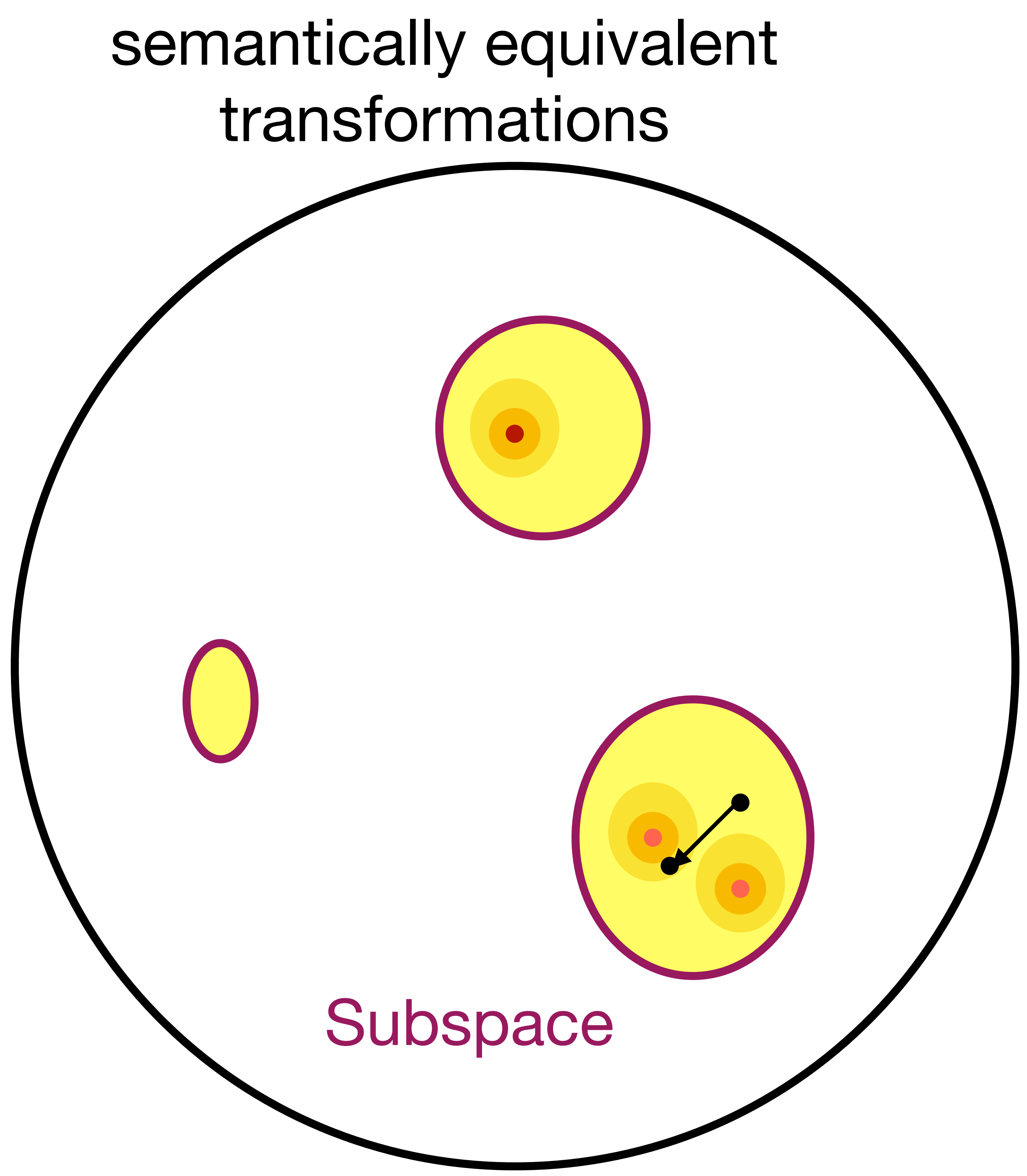
Optimization Decision Making

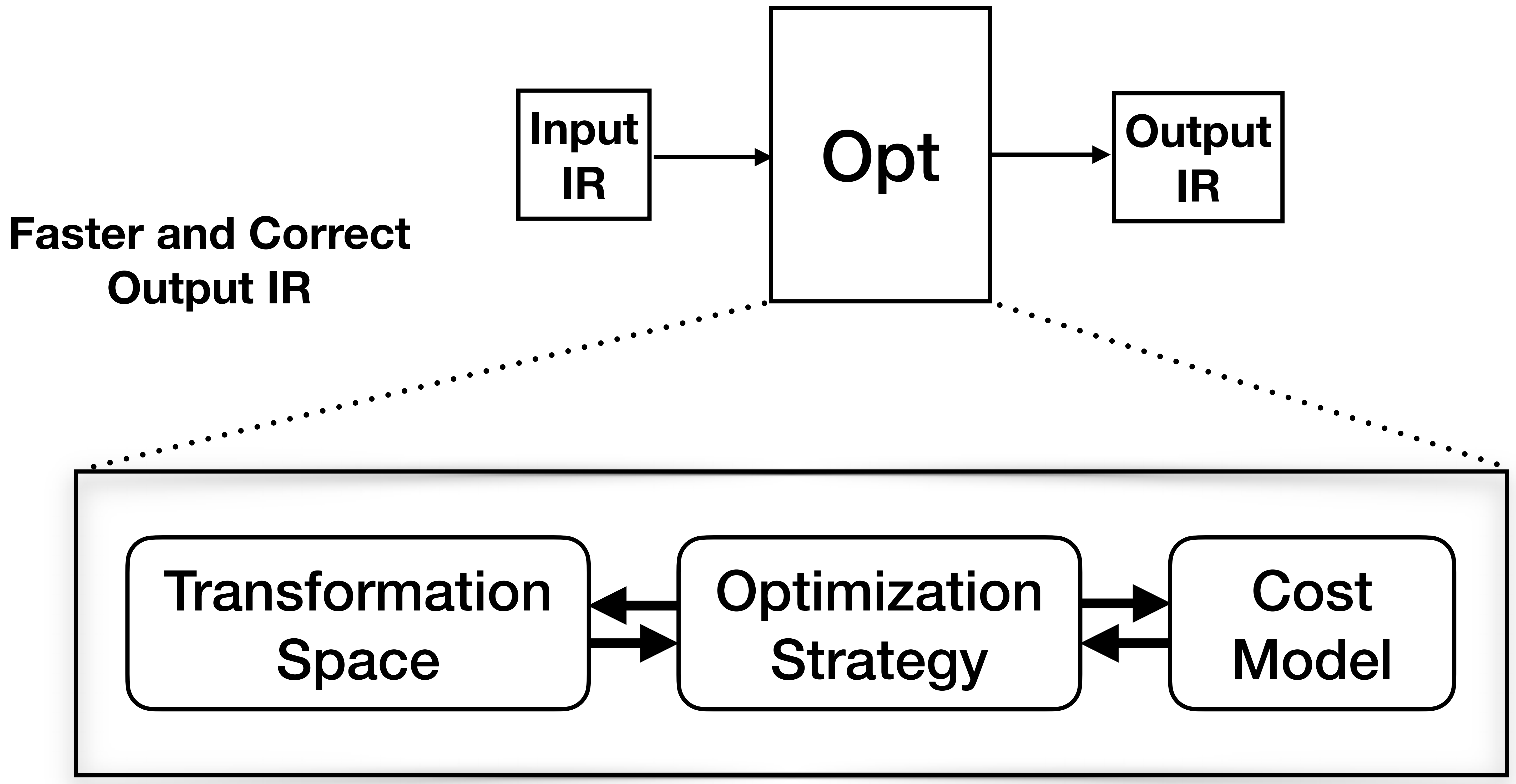


Faster and Correct Output IR



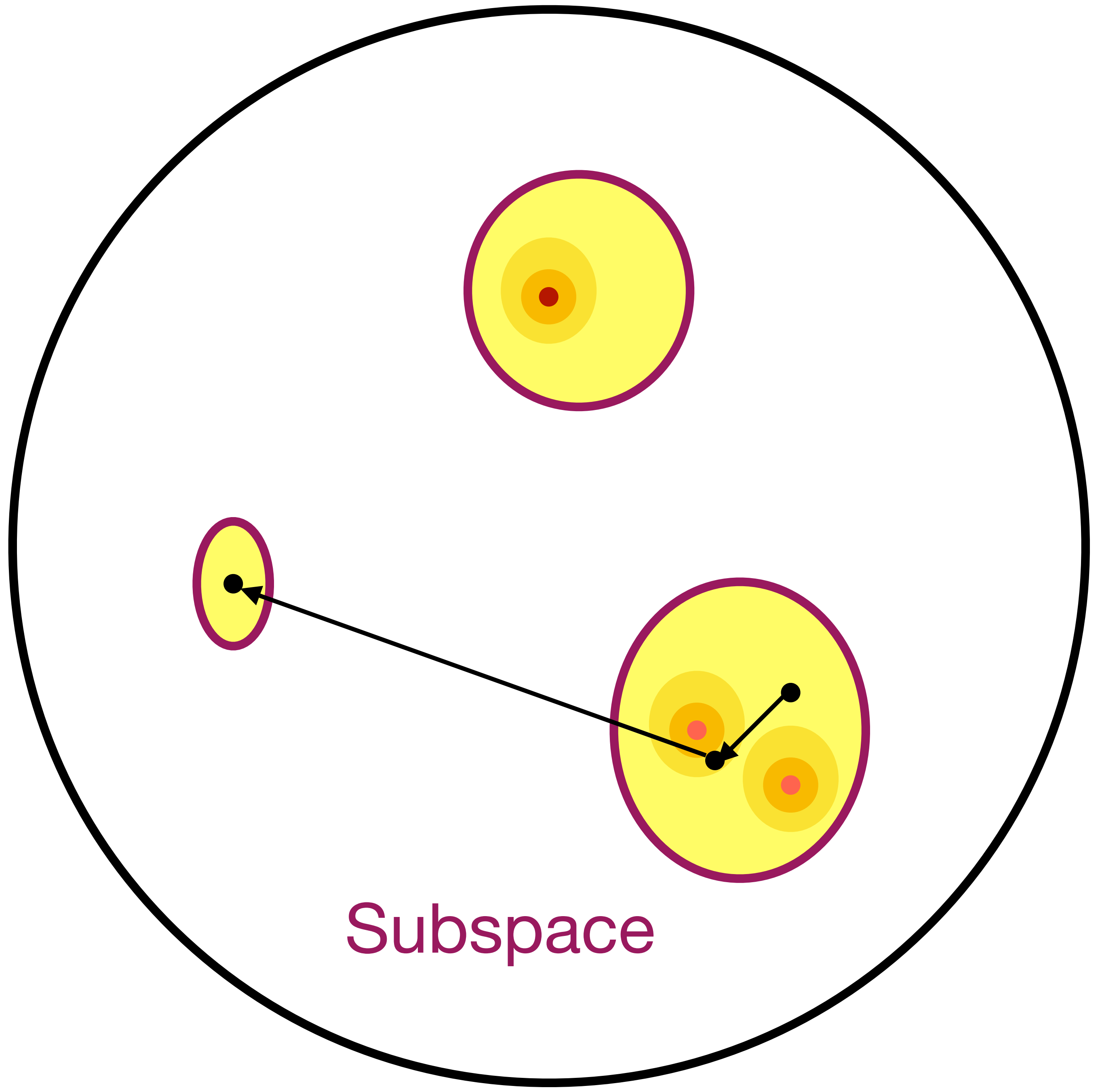
Optimization Decision Making



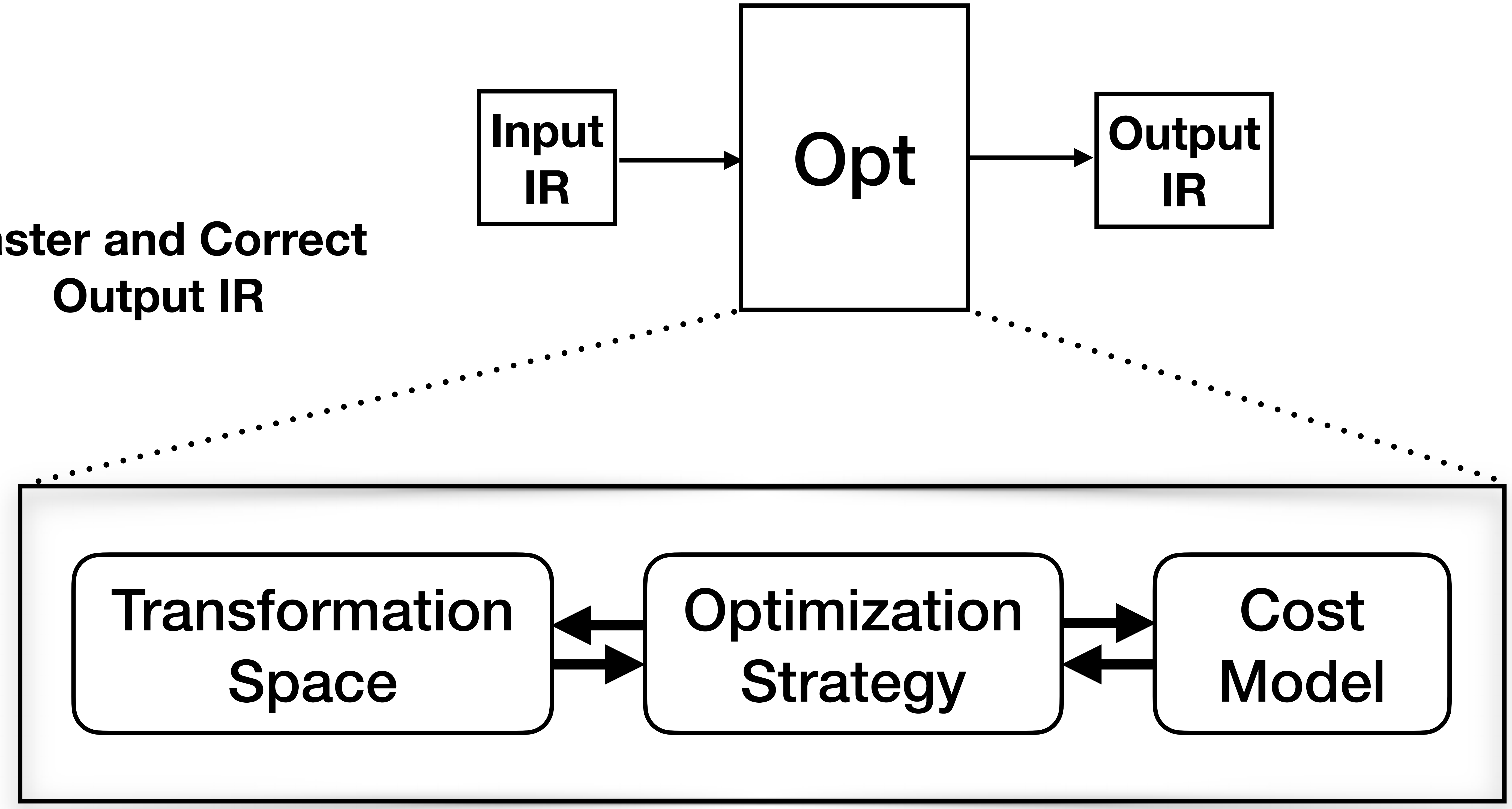


Optimization Decision Making

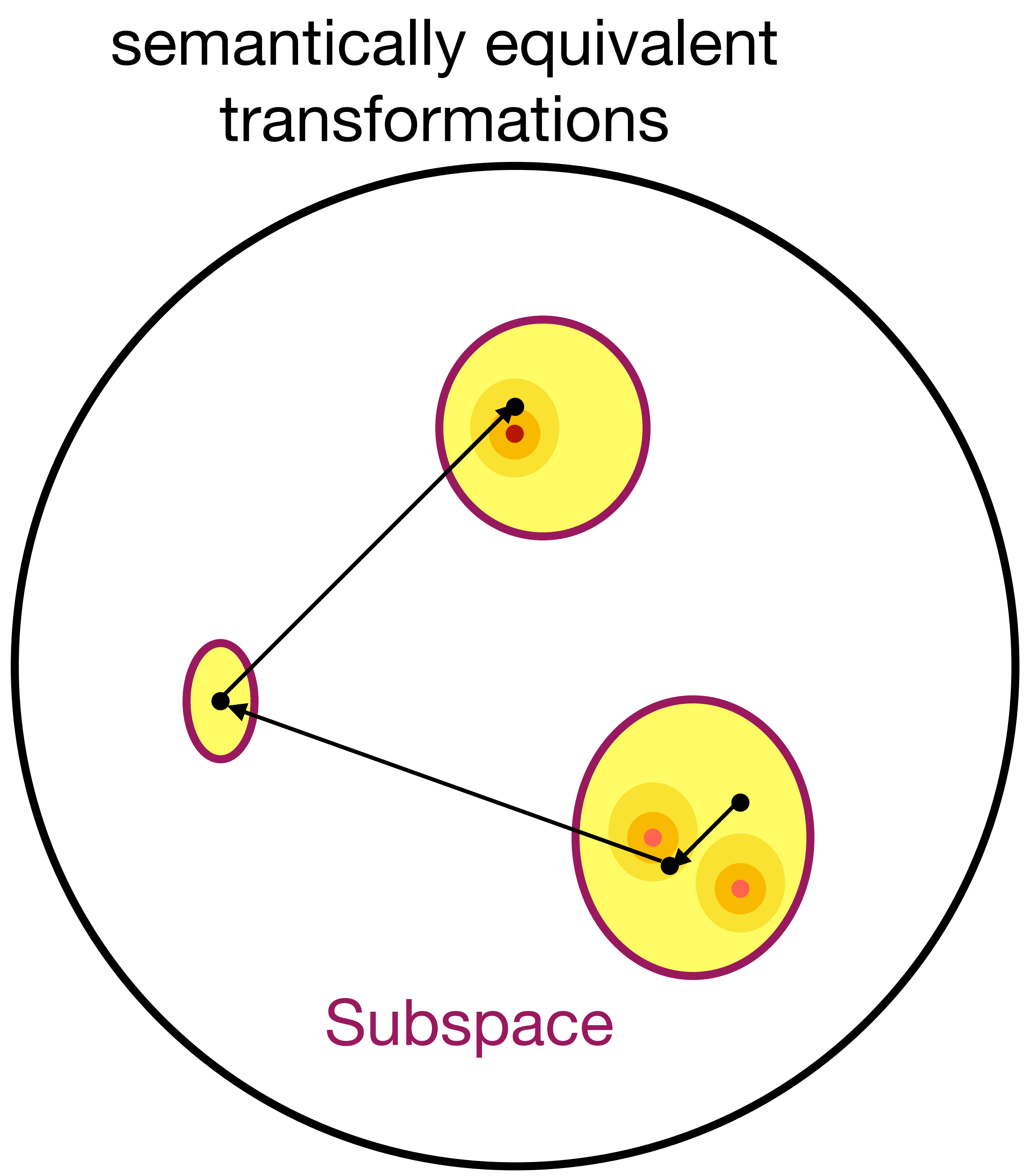
semantically equivalent
transformations



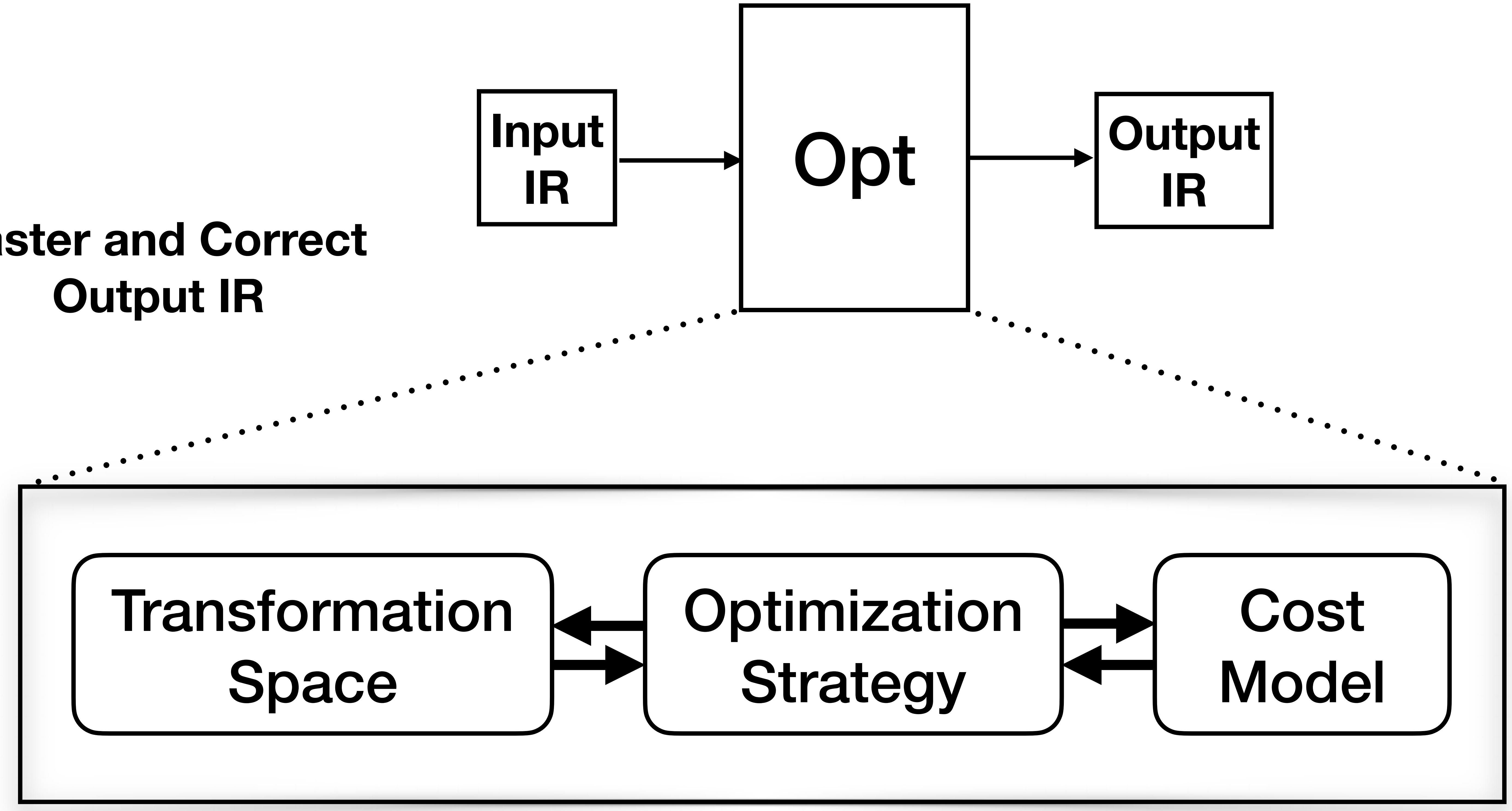
Faster and Correct
Output IR



Optimization Decision Making

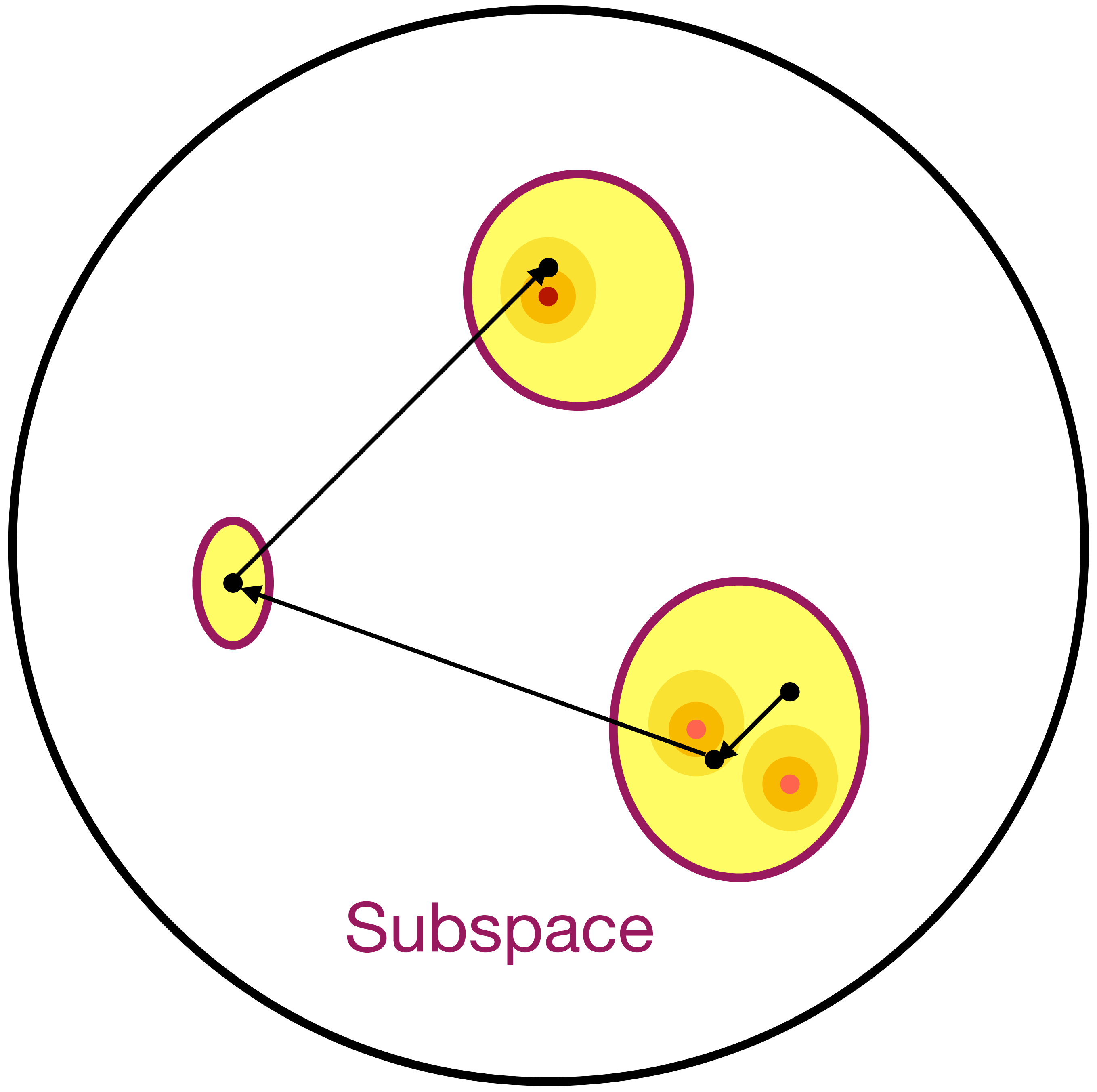


Faster and Correct Output IR

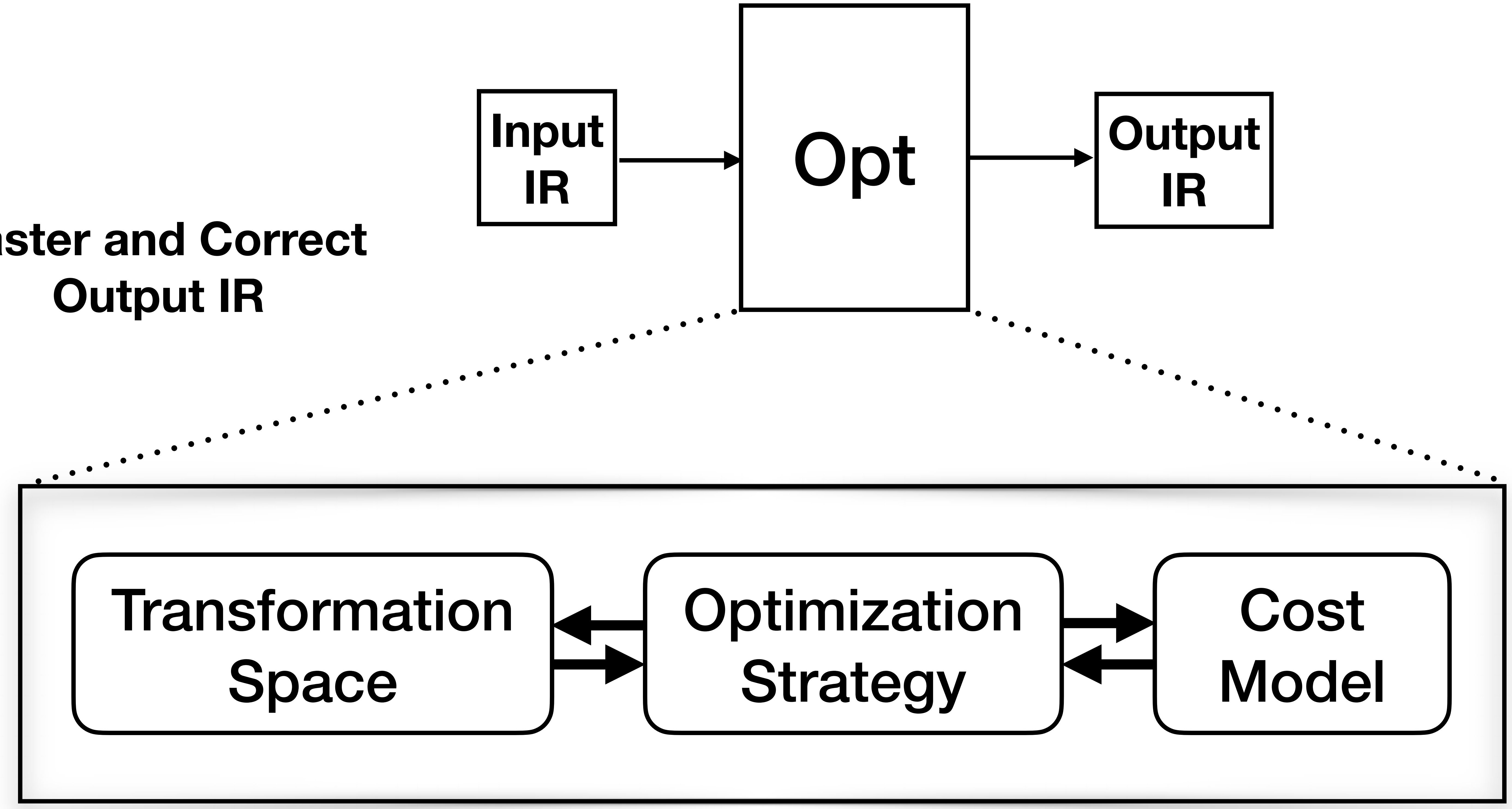


Optimization Decision Making

semantically equivalent
transformations



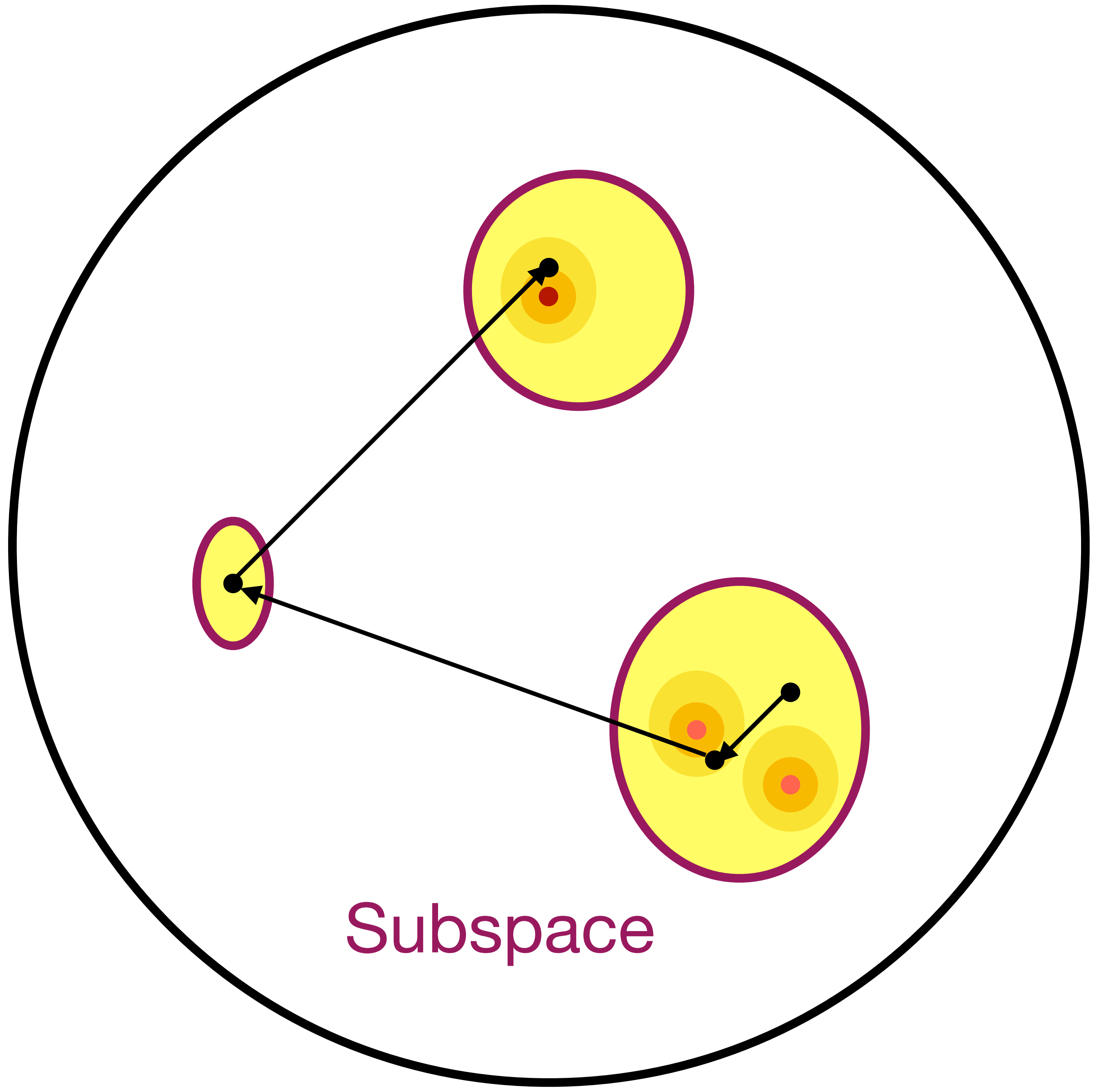
**Faster and Correct
Output IR**



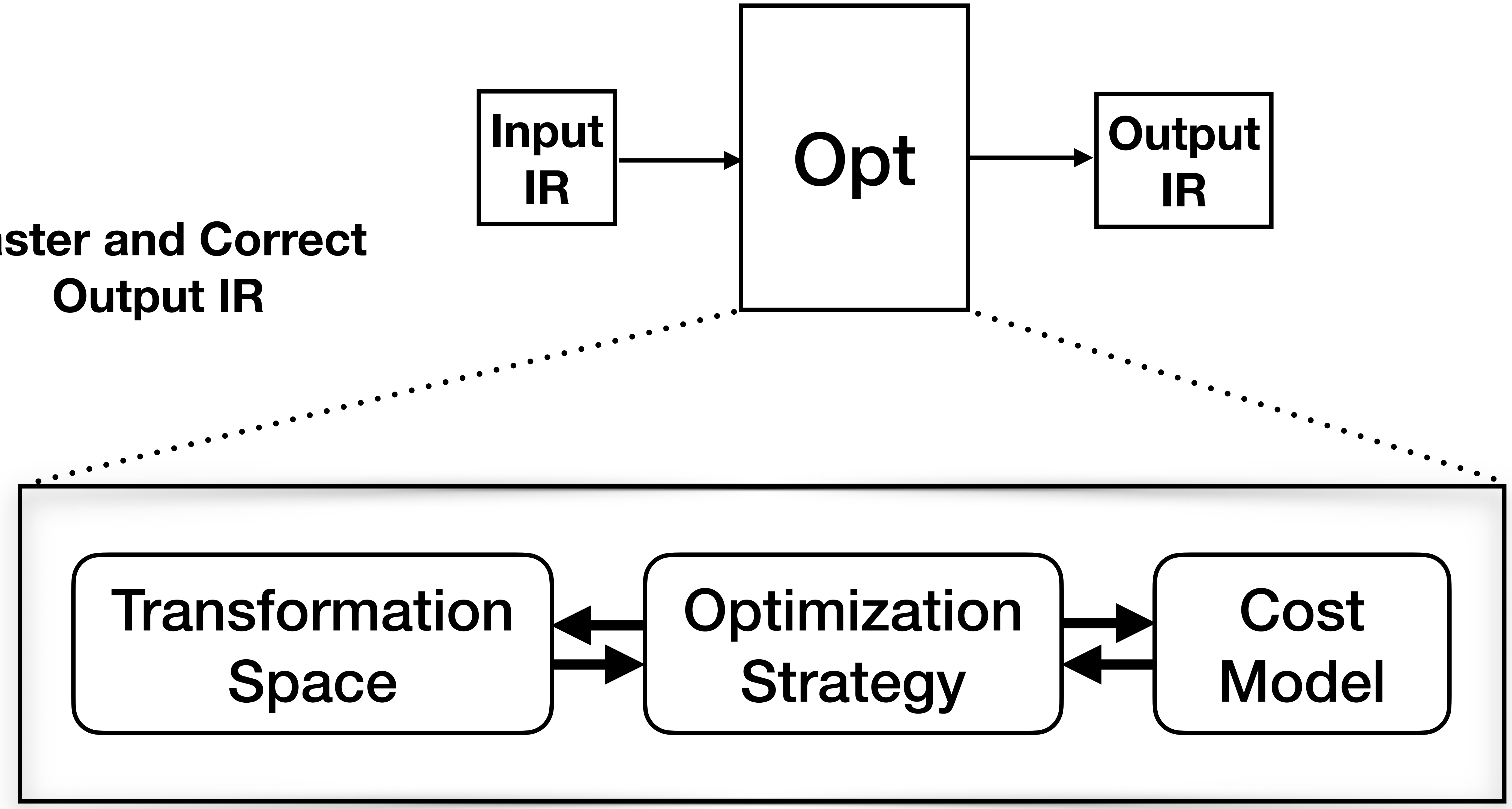
Transformation Space	Optimization Strategy	Cost Model
Ideal	**Ideal**	**Ideal**
All Legal Transformations	**Optimal**	**Ground Truth Runtime**

Optimization Decision Making

semantically equivalent
transformations



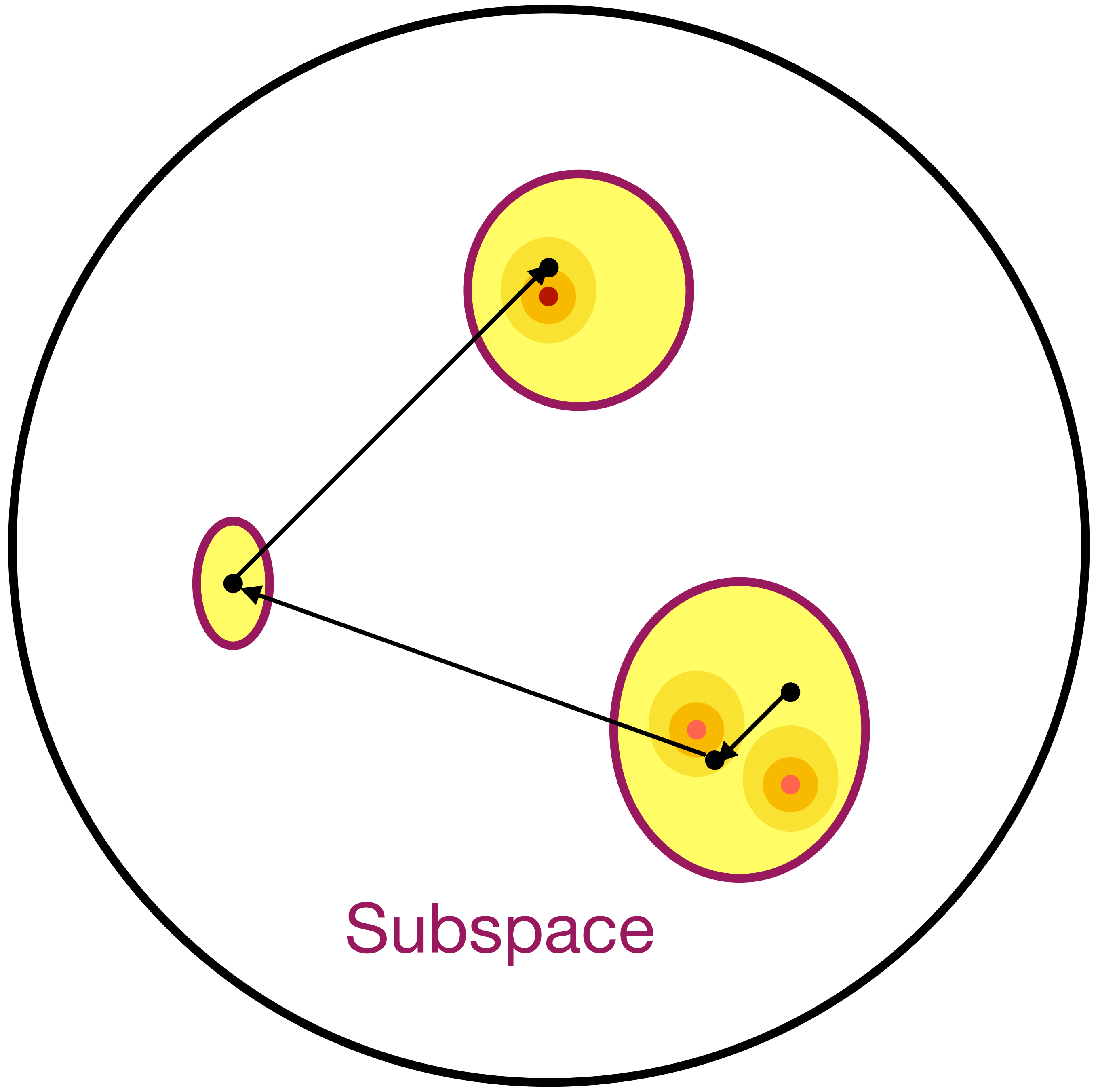
Faster and Correct
Output IR



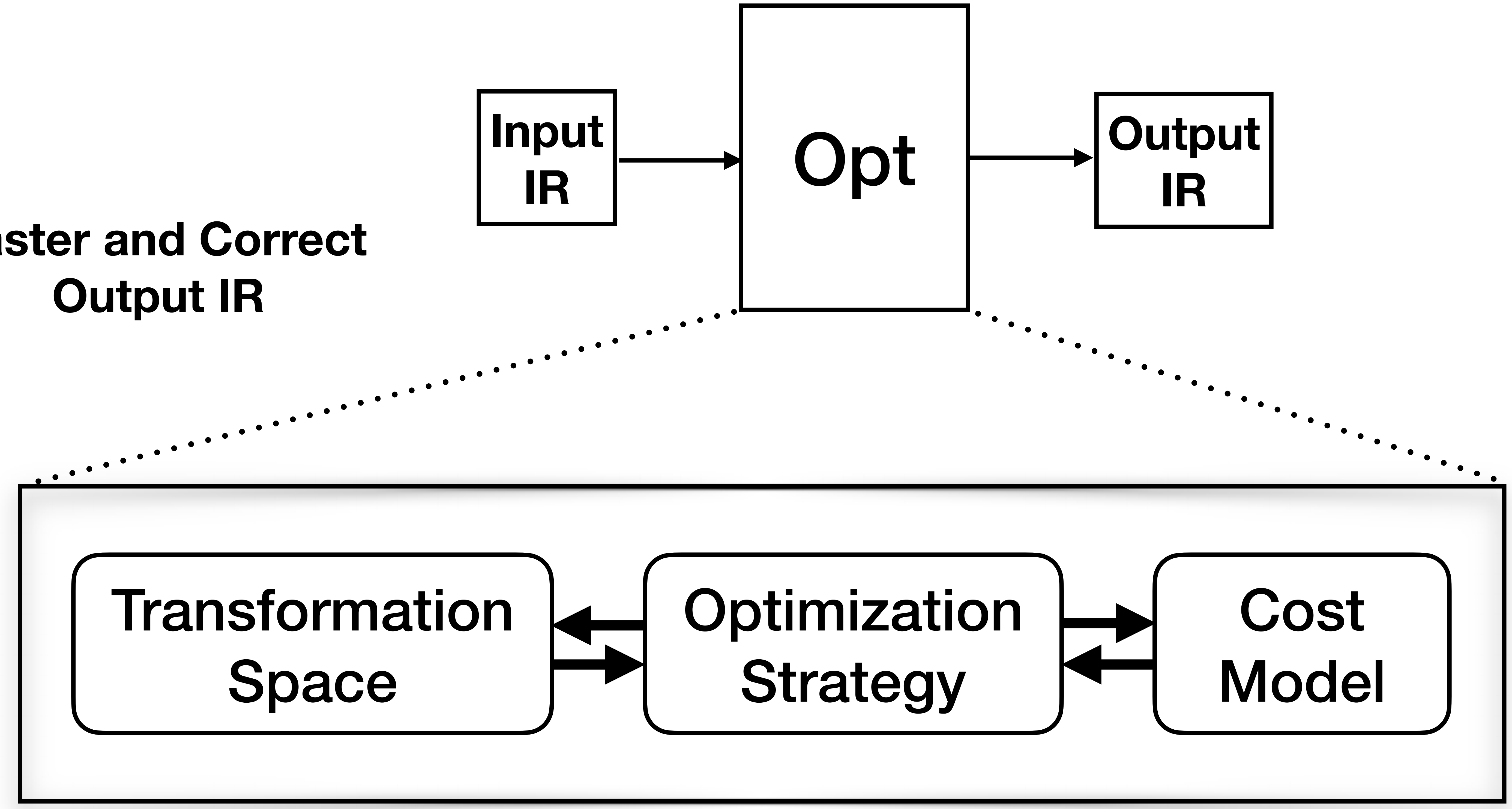
	Approximated	Approximated	Approximated
Manually Constructed :	Limited Subspace	Hand-crafted Heuristics	Hand-crafted simple Cost Models

Optimization Decision Making

semantically equivalent transformations



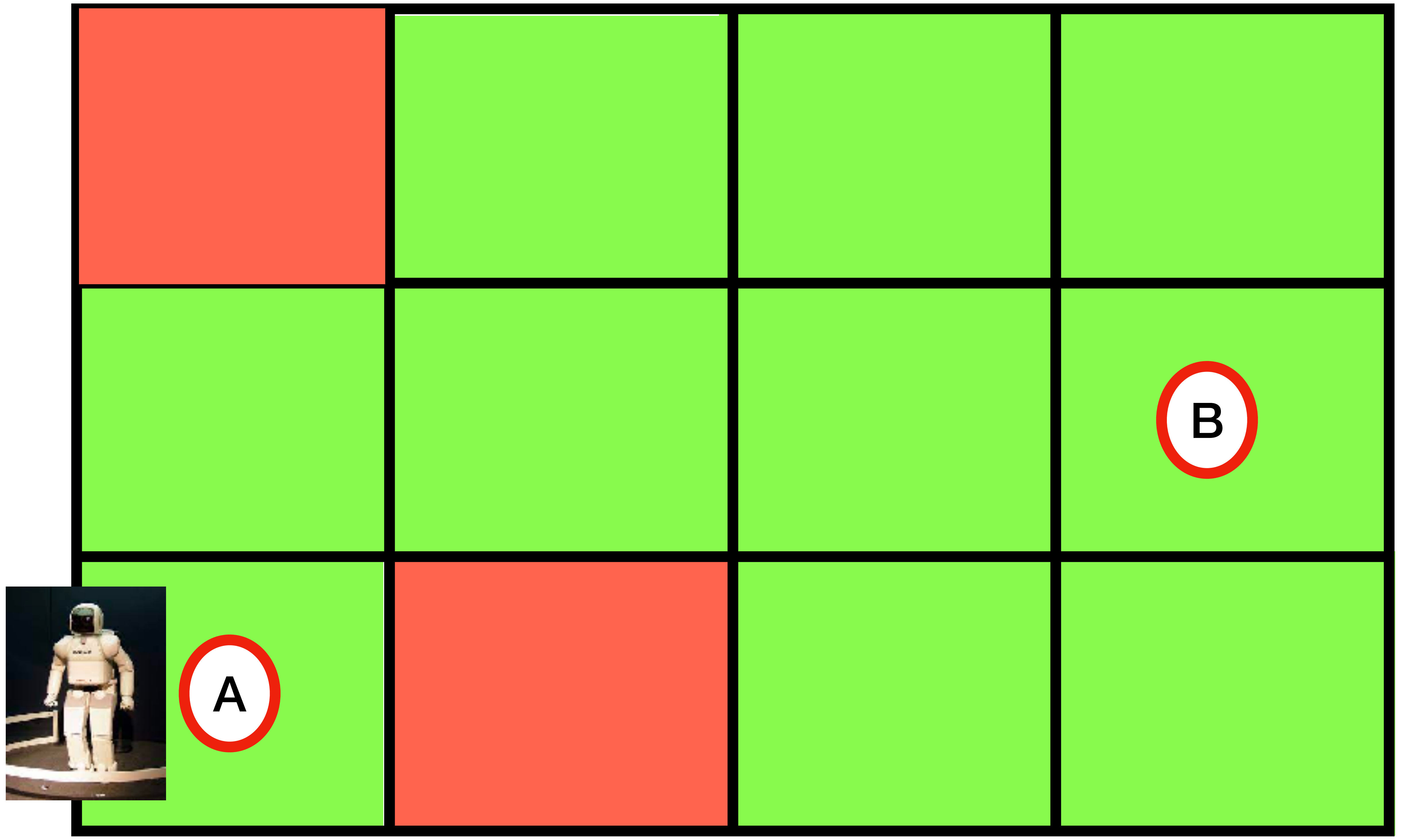
Faster and Correct Output IR



Goal: Automate Construction of these components

Machine Learning is going to help!

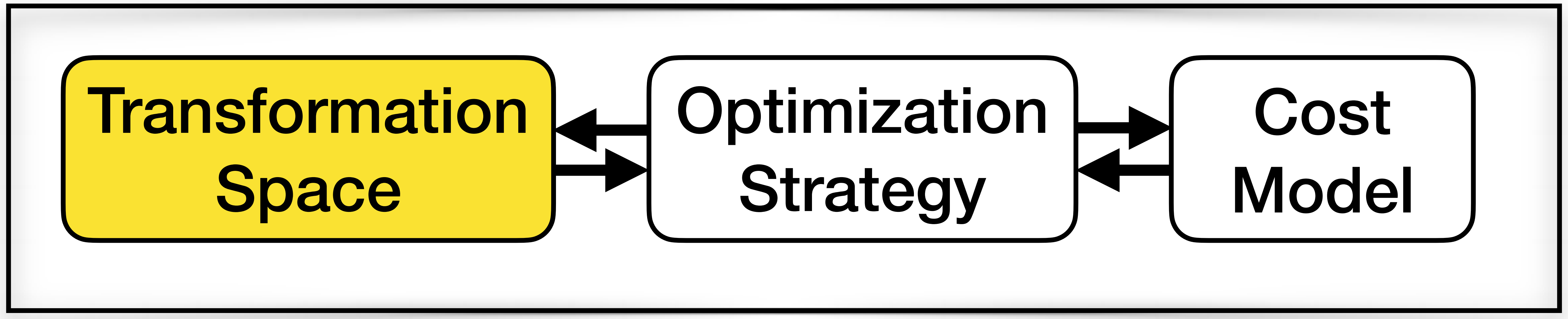
Robot Analogy



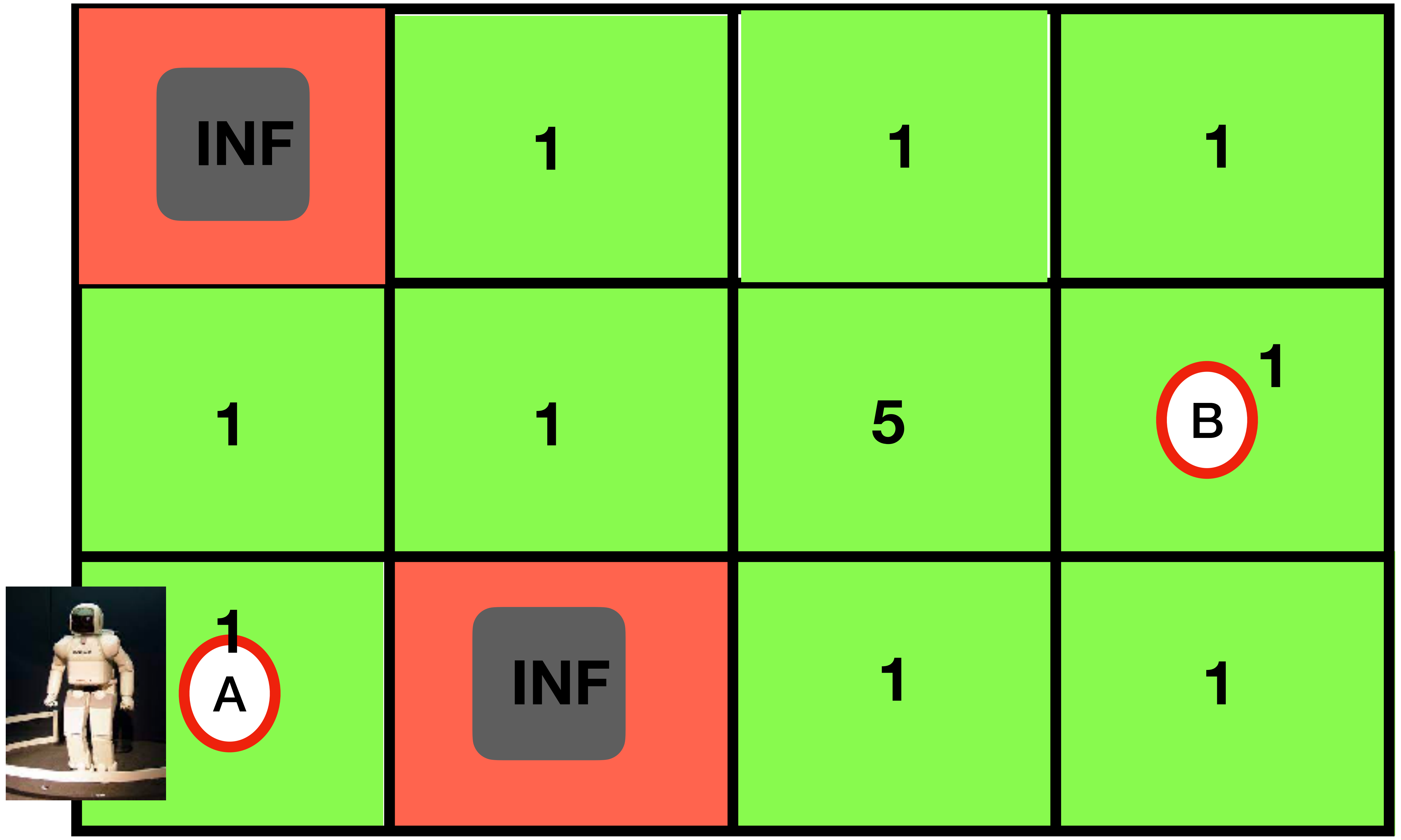
Task: Move from A to B cheaply

1. Plan

2. Execute



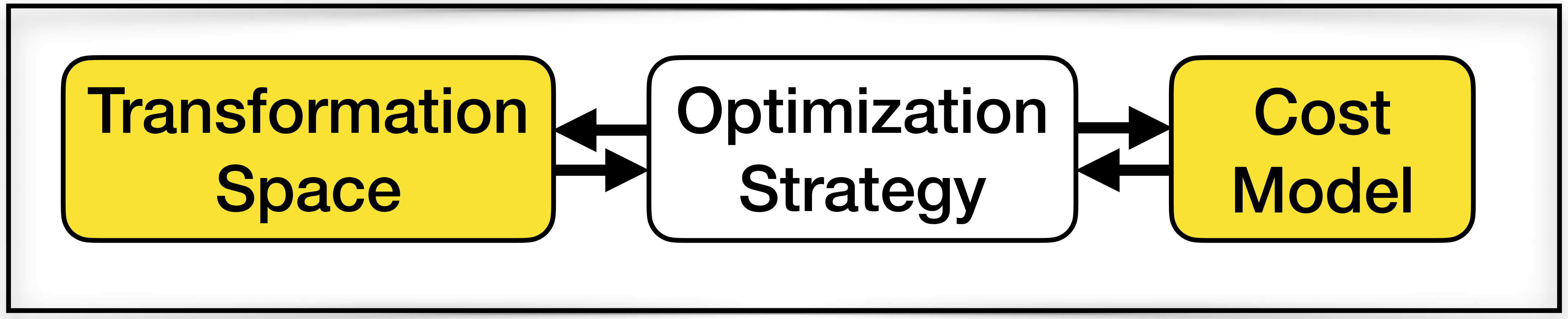
Robot Analogy



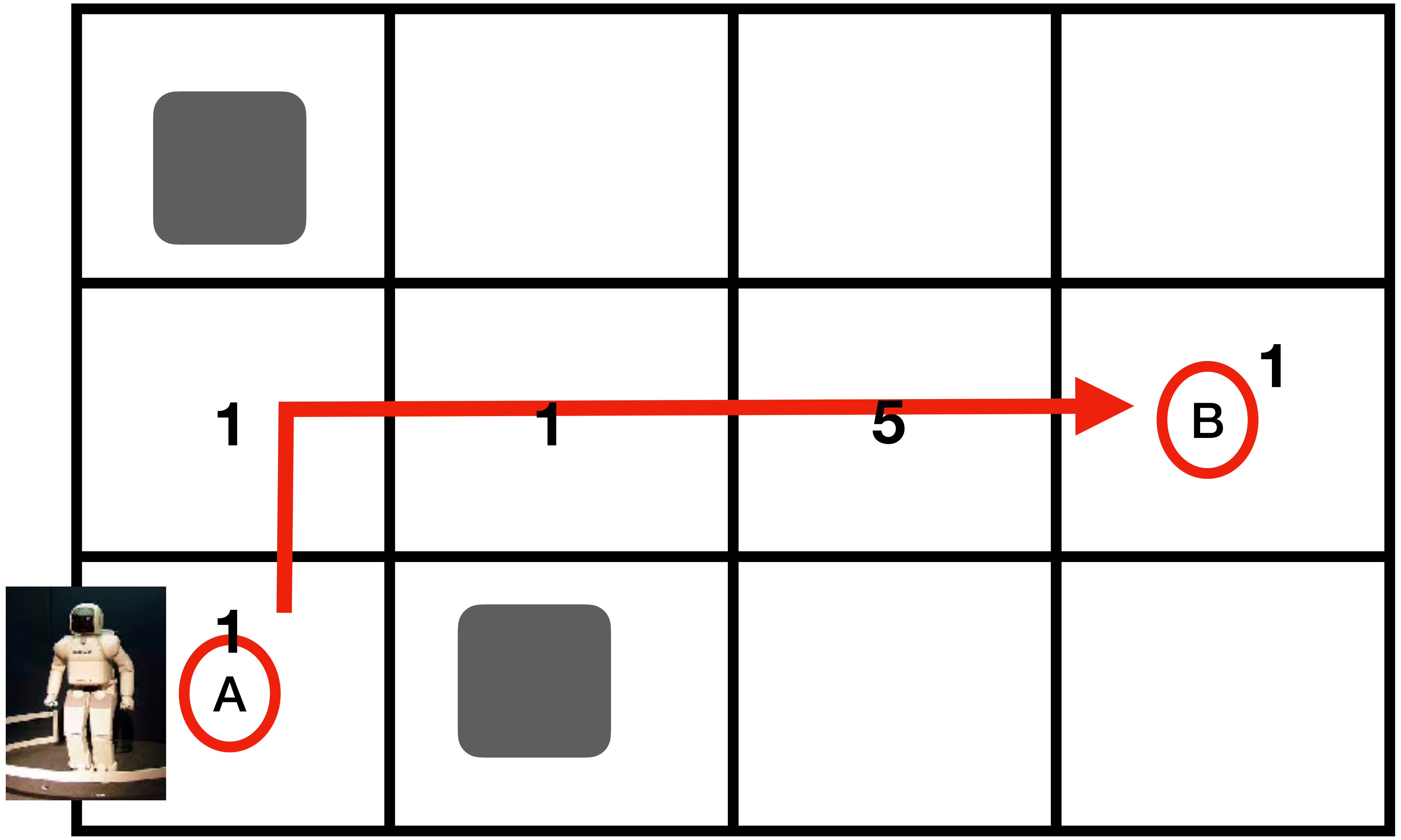
Task: Move from A to B cheaply

1. Plan

2. Execute



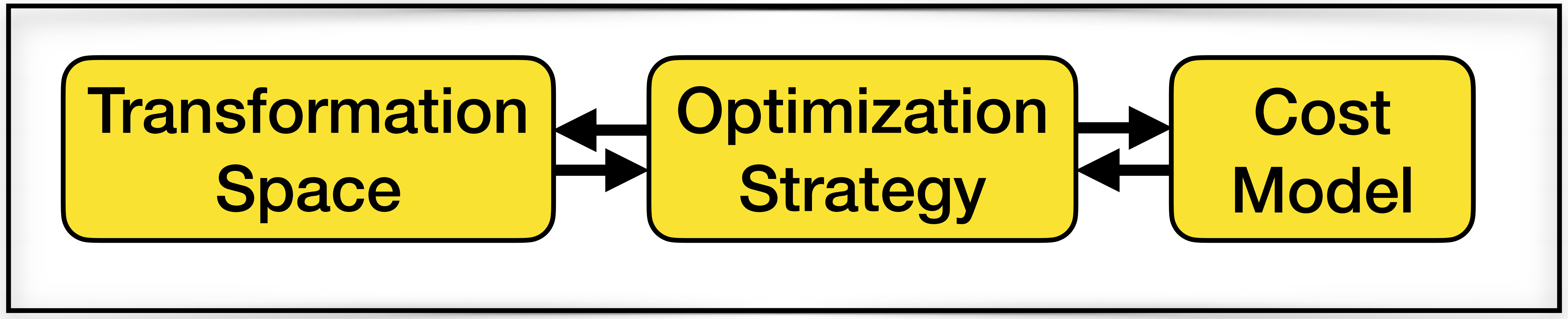
Robot Analogy



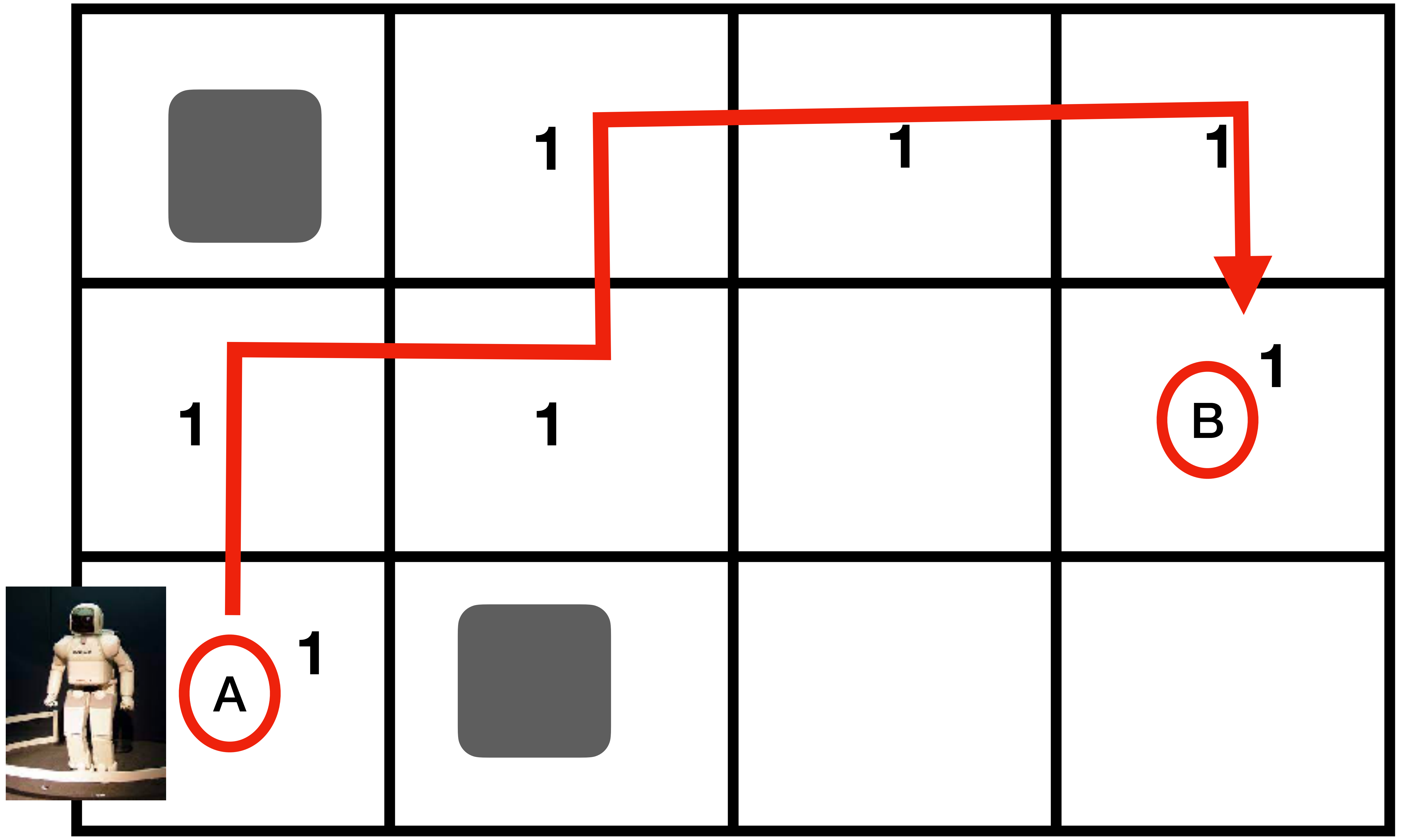
Task: Move from A to B cheaply

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

Cost: 9



Robot Analogy

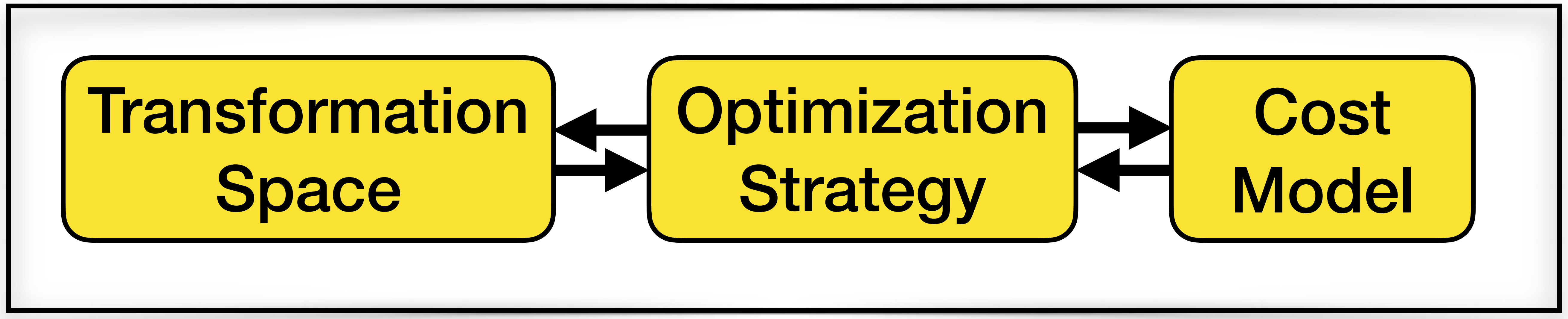


Task: Move from A to B cheaply

1. Plan

2. Execute

Cost: 7



Transformation Spaces

- Loop Transformations
- We will use a combination of horizontal and vertical blurs

```
L1:for(int x = 0; x < width - 2; x++)
    for(int y = 0; y < height; y++)
      blur_x[x][y] = (input[x][y] + input[x+1][y] +
                      input[x+2][y])/3;

L2:for(int x = 0; x < width; x++)
    for(int y = 0; y < height - 2; y++)
      blur_y[x][y] = (blur_x[x][y] + blur_x[x][y+1] +
                      blur_x[x+2][y])/3;
```

- Loop Stripmine
- Loop peeling
- Loop fusion
- Loop unrolling
- Vectorization
- Parallelization
- compute_at

- Transformations are **dependent** on past transformations. Examples?
- **Order** of transformations?
- **Profitability?**

Transformation Spaces

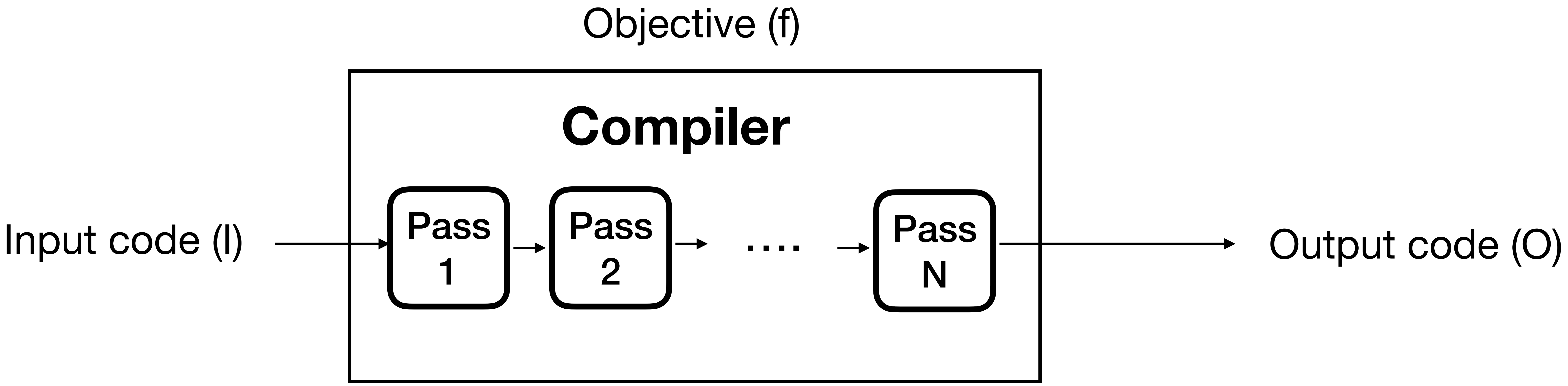
- SLP Vectorization

```
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}

- **Mutually exclusive** options
- **Profitability**

Transformation Spaces



Phase Ordering Problem

{Pass 1, Pass 2, …., Pass N}

N! Options

Where can ML fit in?

- Can ML design transformation spaces?

- Machine Learning is a good fit for
  - Cost Models
  - Optimization Strategies

- Benefits
  - Adaptive and responsive to workload changes
  - Automated; less human burden in the design process
  - Can achieve state-of-the art results

- Drawbacks
  - May be less interpretable than manually written approaches

Types of Learning

- Supervised Learning (labelled data)
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning

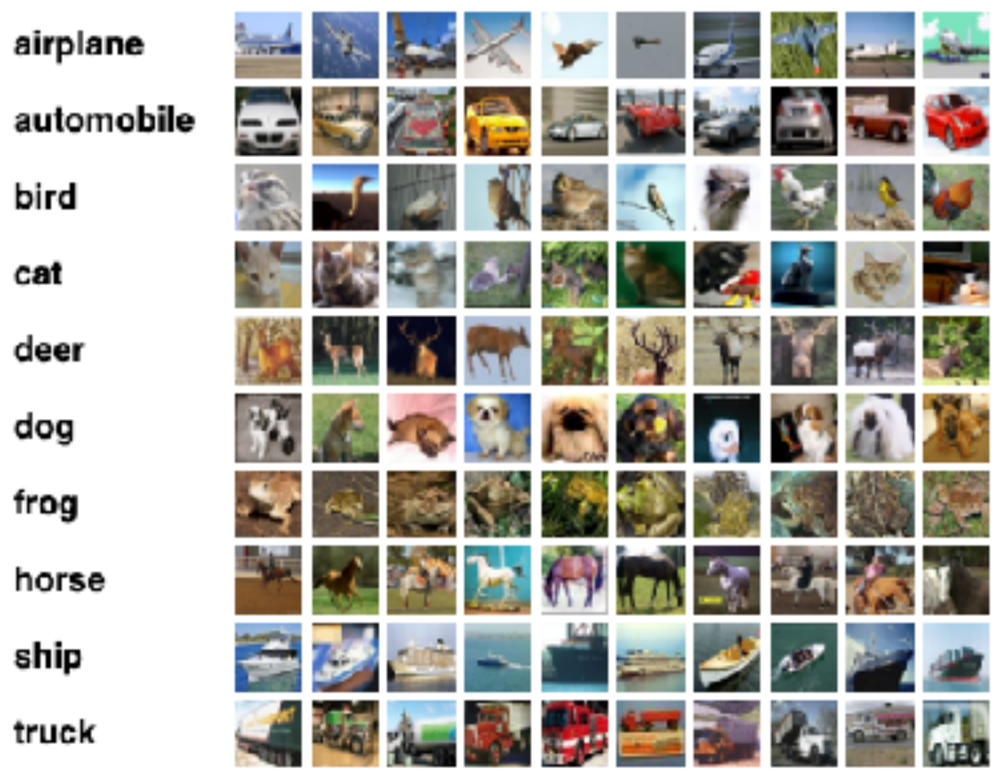
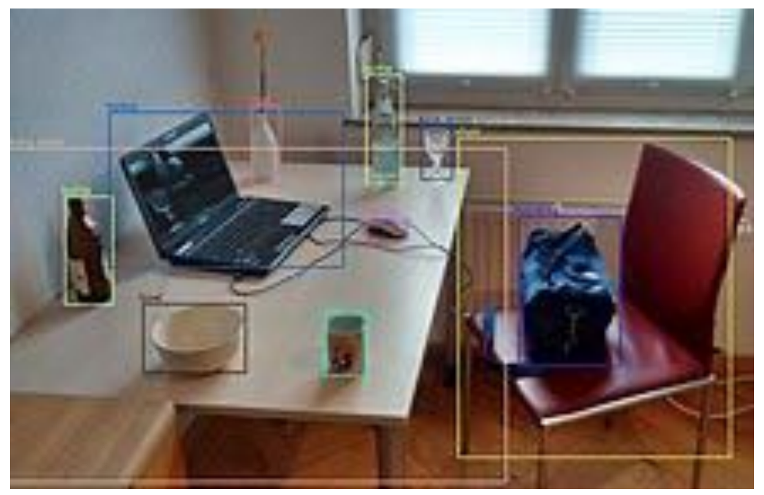
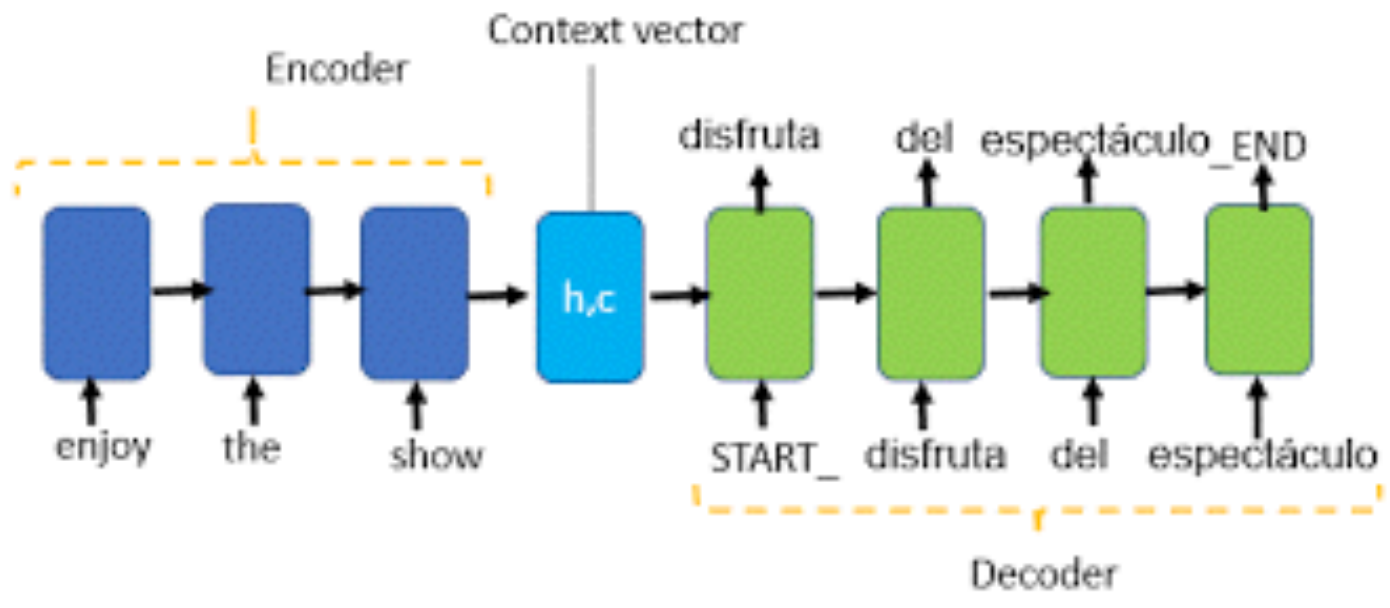


Image Classification



Object Detection

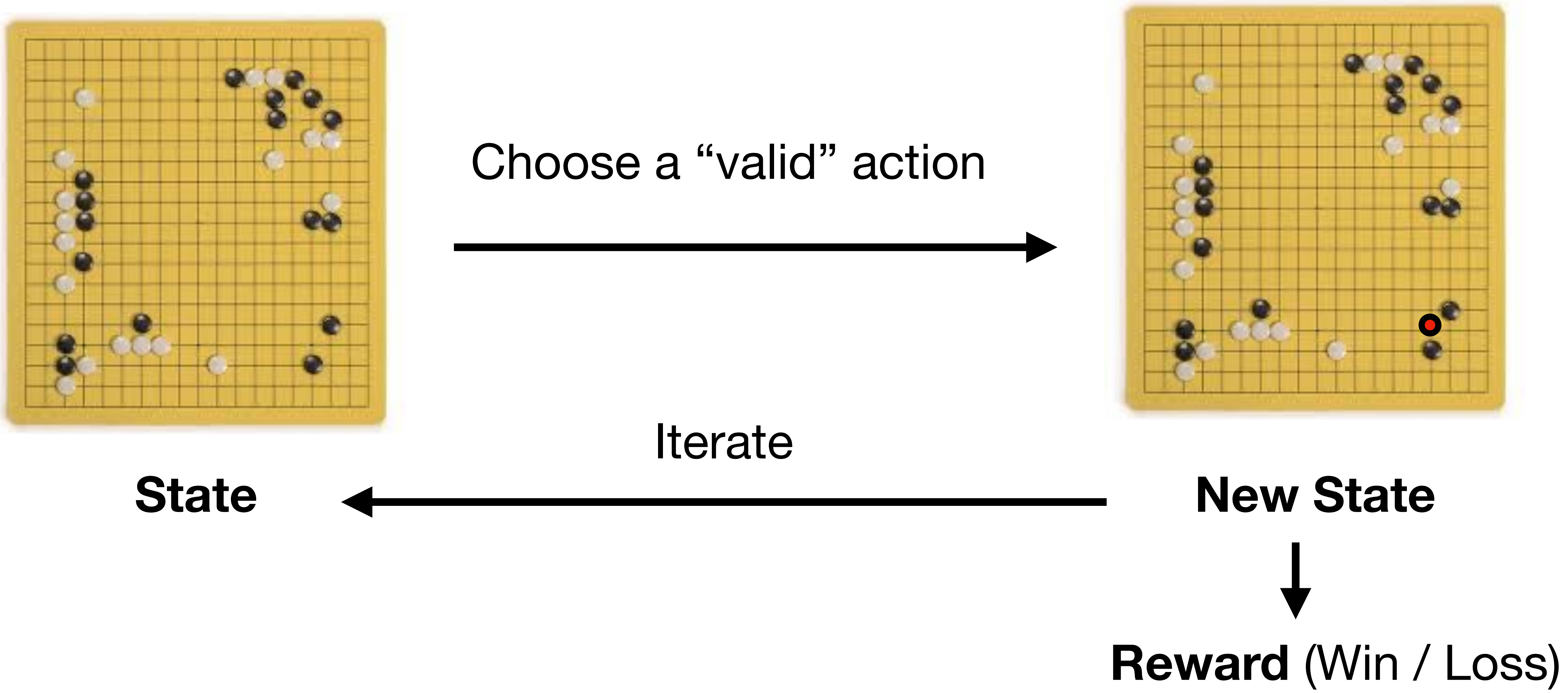


Machine Translation

Types of Learning

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning

No labelled data; learn from experience



Cost Models

- Analytical Models
- e.g., Basic block cost estimation: LLVM-MCA
- Hand-written and cumbersome to maintain
- Usually built with many assumptions baked in
  - Costs are additive
  - Costs are linear
  - Hardware manuals are the ground truth

```
~2000 lines

// BMI1 BEXTR/BLS, BMI2 BZHI
defm : HWWriteResPair<WriteBEXTR, [HWPort06,HWPort15], 2, [1,1], 2>;
defm : HWWriteResPair<WriteBLS,   [HWPort15], 1>;
defm : HWWriteResPair<WriteBZHI,  [HWPort15], 1>;

// TODO: Why isn't the HWDivider used?
defm : X86WriteRes<WriteDiv8,     [HWPort0,HWPort1,HWPort5,HWPort6], 22, [], 9>;
defm : X86WriteRes<WriteDiv16,
[HWPort0,HWPort1,HWPort5,HWPort6,HWPort01,HWPort0156], 98, [7,7,3,3,1,11], 32>;
defm : X86WriteRes<WriteDiv32,
[HWPort0,HWPort1,HWPort5,HWPort6,HWPort01,HWPort0156], 98, [7,7,3,3,1,11], 32>;
defm : X86WriteRes<WriteDiv64,
[HWPort0,HWPort1,HWPort5,HWPort6,HWPort01,HWPort0156], 98, [7,7,3,3,1,11], 32>;
defm : X86WriteRes<WriteDiv8Ld,   [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteDiv16Ld,  [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteDiv32Ld,  [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteDiv64Ld,  [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;

defm : X86WriteRes<WriteIDiv8,    [HWPort0,HWPort1,HWPort5,HWPort6], 23, [], 9>;
defm : X86WriteRes<WriteIDiv16,
[HWPort0,HWPort1,HWPort5,HWPort6,HWPort06,HWPort0156], 112, [4,2,4,8,14,34], 66>;
defm : X86WriteRes<WriteIDiv32,
[HWPort0,HWPort1,HWPort5,HWPort6,HWPort06,HWPort0156], 112, [4,2,4,8,14,34], 66>;
defm : X86WriteRes<WriteIDiv64,
[HWPort0,HWPort1,HWPort5,HWPort6,HWPort06,HWPort0156], 112, [4,2,4,8,14,34], 66>;
defm : X86WriteRes<WriteIDiv8Ld,  [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteIDiv16Ld, [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteIDiv32Ld, [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteIDiv64Ld, [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;

// Scalar and vector floating point.
defm : X86WriteRes<WriteFLD0,          [HWPort01], 1, [1], 1>;
defm : X86WriteRes<WriteFLD1,          [HWPort01], 1, [2], 2>;
defm : X86WriteRes<WriteFLDC,          [HWPort01], 1, [2], 2>;
defm : X86WriteRes<WriteFLoad,         [HWPort23], 5, [1], 1>;
defm : X86WriteRes<WriteFLoadX,        [HWPort23], 6, [1], 1>;
defm : X86WriteRes<WriteFLoadY,        [HWPort23], 7, [1], 1>;
defm : X86WriteRes<WriteFMaskedLoad,   [HWPort23,HWPort5], 8, [1,2], 3>;
defm : X86WriteRes<WriteFMaskedLoadY,  [HWPort23,HWPort5], 9, [1,2], 3>;
defm : X86WriteRes<WriteFStore,        [HWPort237,HWPort4], 1, [1,1], 2>;
defm : X86WriteRes<WriteFStoreX,       [HWPort237,HWPort4], 1, [1,1], 2>;
defm : X86WriteRes<WriteFStoreY,       [HWPort237,HWPort4], 1, [1,1], 2>;
defm : X86WriteRes<WriteFStoreNT,      [HWPort237,HWPort4], 1, [1,1], 2>;
defm : X86WriteRes<WriteFStoreNTX,     [HWPort237,HWPort4], 1, [1,1], 2>;
defm : X86WriteRes<WriteFStoreNTY,     [HWPort237,HWPort4], 1, [1,1], 2>;

defm : X86WriteRes<WriteFMaskedStore32,  [HWPort0,HWPort4,HWPort237,HWPort15], 5,
[1,1,1,1], 4>;
defm : X86WriteRes<WriteFMaskedStore32Y, [HWPort0,HWPort4,HWPort237,HWPort15], 5,
```

Data-driven Cost Models

Approach 1: Specify structure and then learn the coefficients

$$\tilde{y}(t,x) = C_{flop} \times \boxed{t_{flop}} + C_{msg} \times \boxed{t_{msg}} + C_{vol} \times \boxed{t_{vol}}$$

$$C_{flop} = \frac{2n^2(3m-n)}{2p} + \frac{b_r n^2}{2p_c} + \frac{3b_r n(2m-n)}{2p_r} + \frac{b_r{}^2 n}{3p_r}$$

$$C_{msg} = 3n \log p_r + \frac{2n}{b_r} \log p_c$$

$$C_{vol} = \left(\frac{n^2}{p_c} + b_r n\right) \log p_r + \left(\frac{mn - n^2/2}{p_r} + \frac{b_r n}{2}\right) \log p_c$$

t_{flop}, t_{msg}, t_{vol} are learned

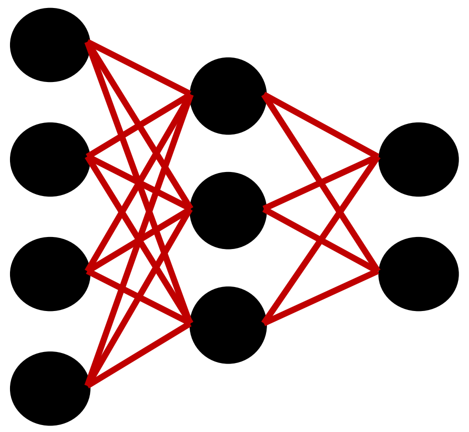
Liu et. al, "GPTune: multitask learning for autotuning exascale applications", PPoPP 2021

Data-driven Cost Models

Approach 2: Model parameterized with features

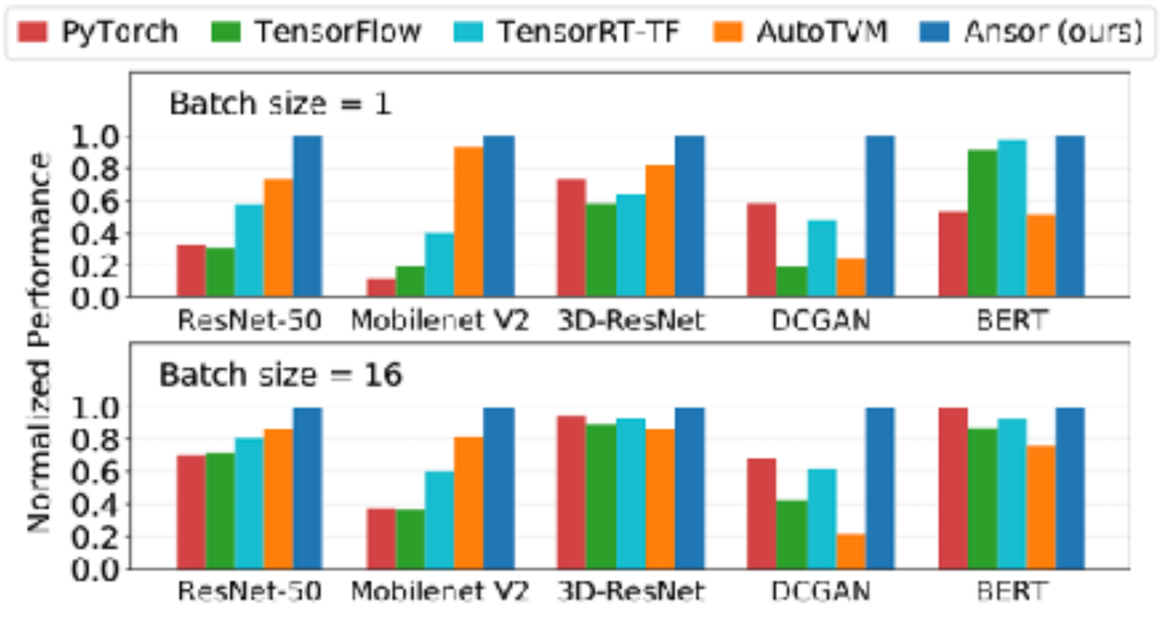
Manually extracted

$y = f(s)$

Code → Features (s) →  → Runtime (y)

Ansor: Generating High-Performance Tensor Programs for Deep Learning

Lianmin Zheng [1], Chengfan Jia [2], Minmin Sun [2], Zhao Wu [2], Cody Hao Yu [3], Ameer Haj-Ali [1], Yida Wang [3], Jun Yang [2], Danyang Zhuo [1,4], Koushik Sen [1], Joseph E. Gonzalez [1], Ion Stoica [1]



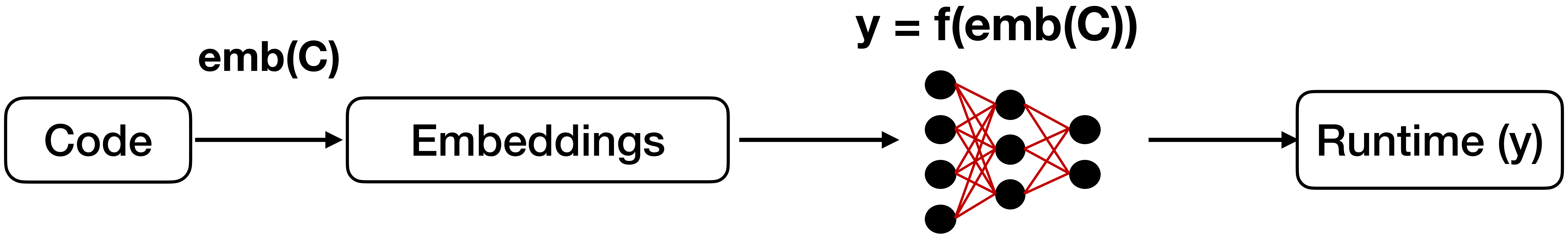
(b) NVIDIA GPU

Learning to Optimize Halide with Tree Search and Random Programs

ANDREW ADAMS, Facebook AI Research
KARIMA MA, UC Berkeley
LUKE ANDERSON, MIT CSAIL
RIYADH BAGHDADI, MIT CSAIL
TZU-MAO LI, MIT CSAIL
MICHAËL GHARBI, Adobe
BENOIT STEINER, Facebook AI Research
STEVEN JOHNSON, Google
KAYVON FATAHALIAN, Stanford University
FRÉDO DURAND, MIT CSAIL
JONATHAN RAGAN-KELLEY, UC Berkeley

Data-driven Cost Models

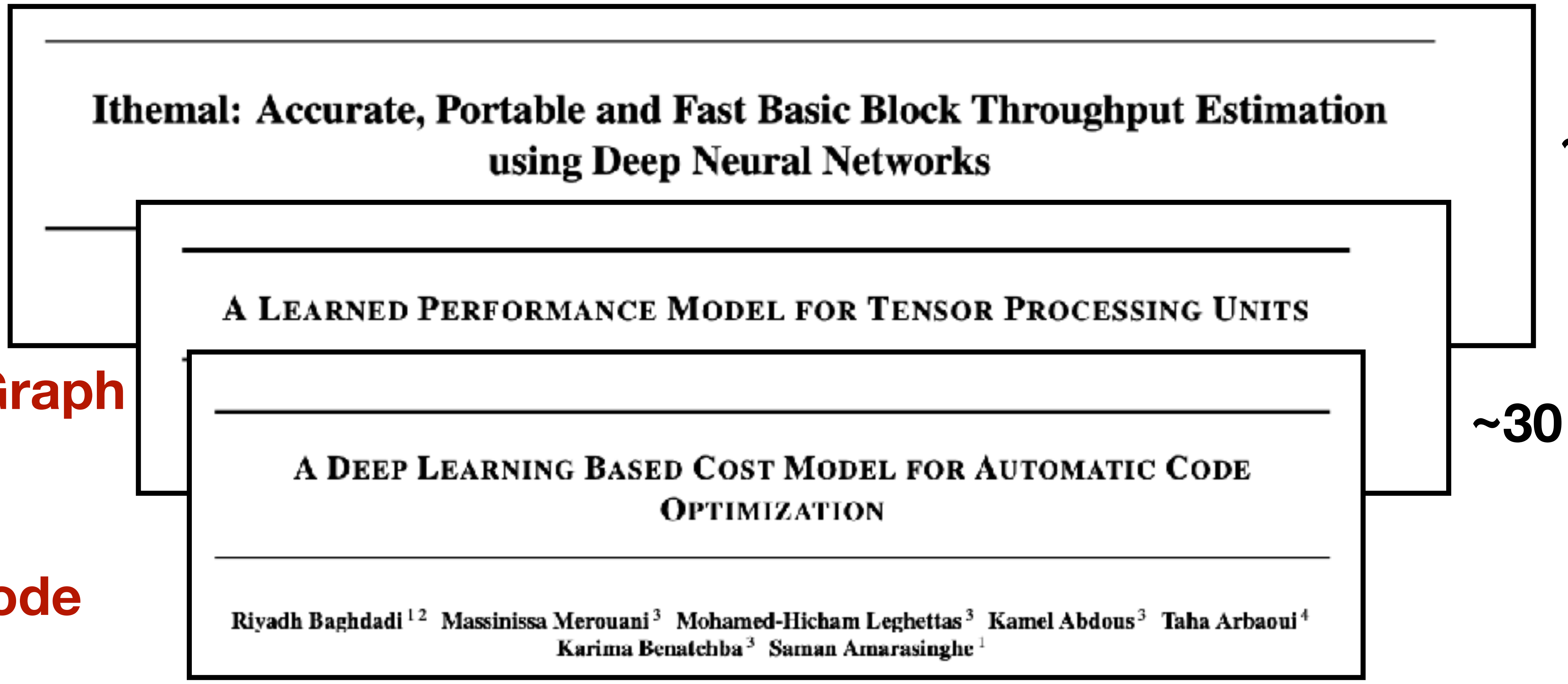
Approach 3: black box models that are completely learned



Machine Code

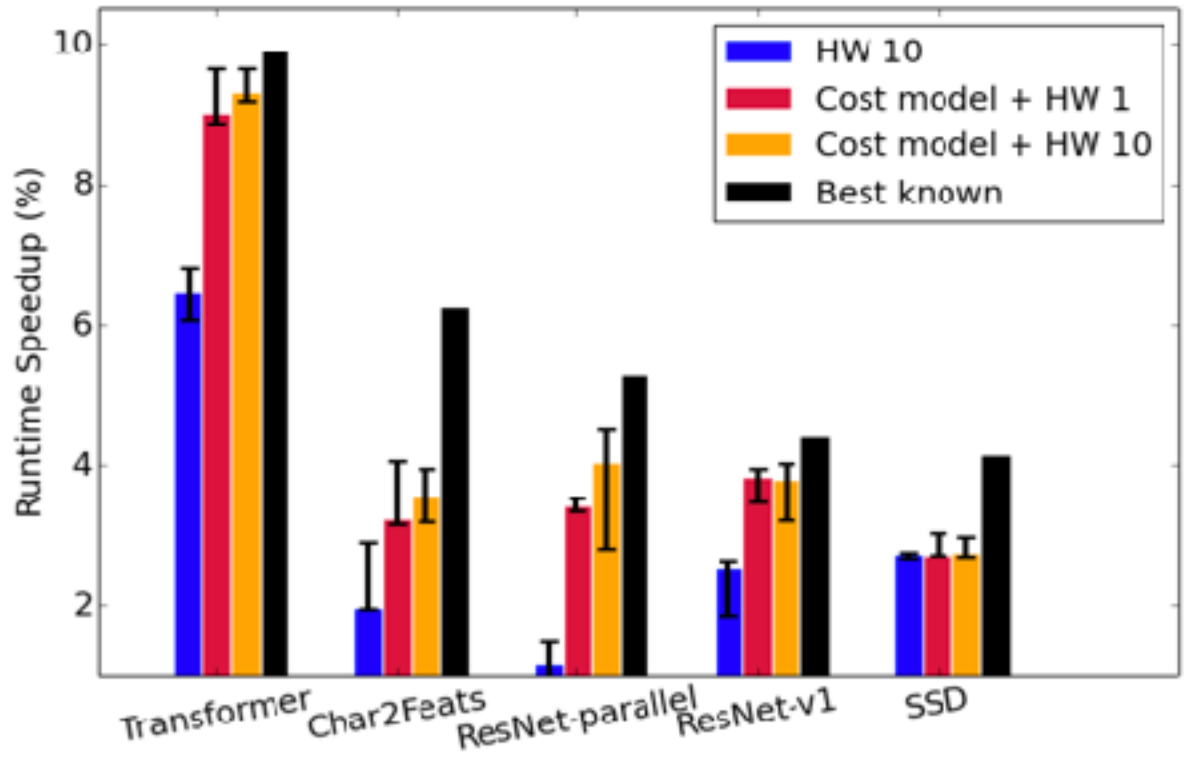
Computational Graph (XLA IR)

Source Code



~20 -> 8 MAPE

~30 -> 4.5 MAPE

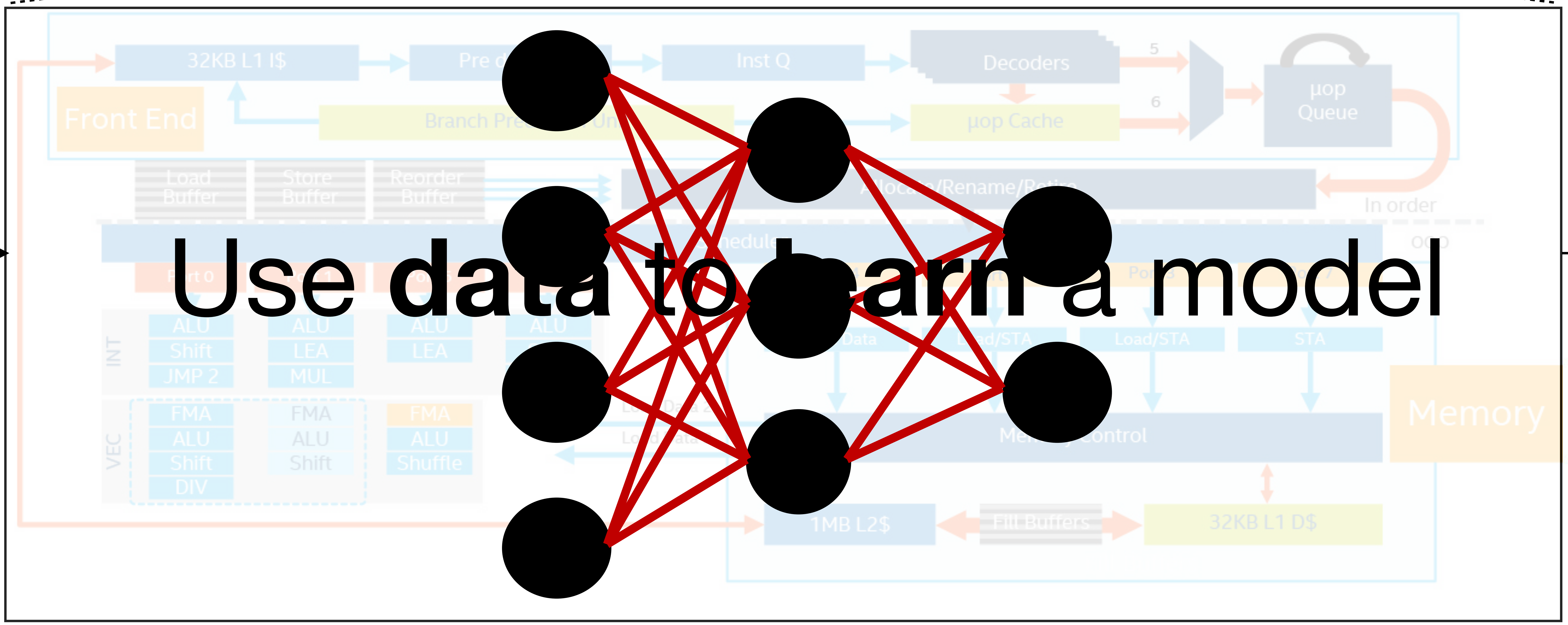


Basic Block Throughput Estimation

Mendis et. al "Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks" [ICML'19]

Ithemal

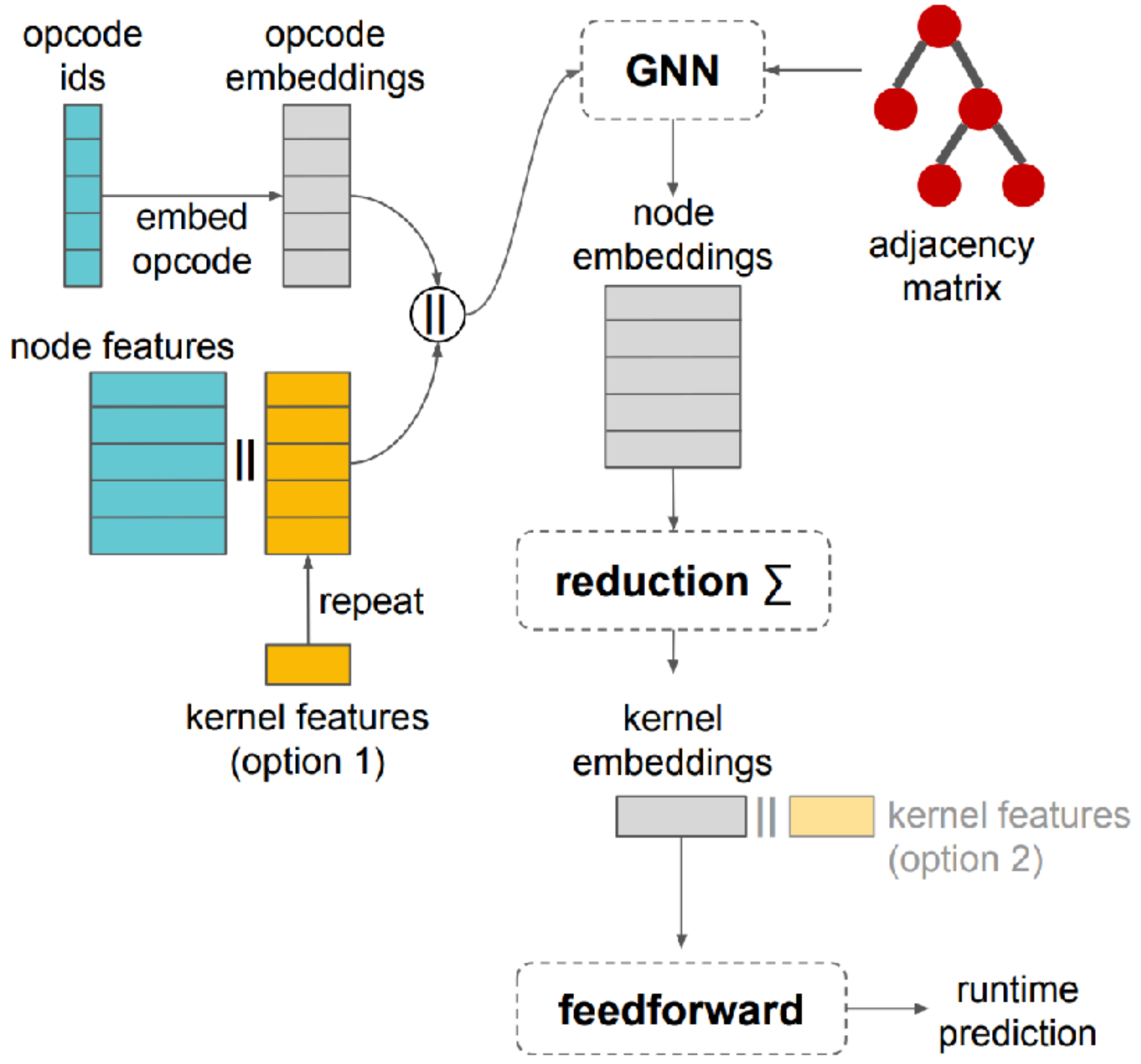
```
lea    r14, [rbx-0x40]
       ........
       ........
lea    rdx, [rbp+0x38]
cmp    rdi, rax
```



44 cycles

Basic Block Throughput Estimation

Learned TPU Cost Model

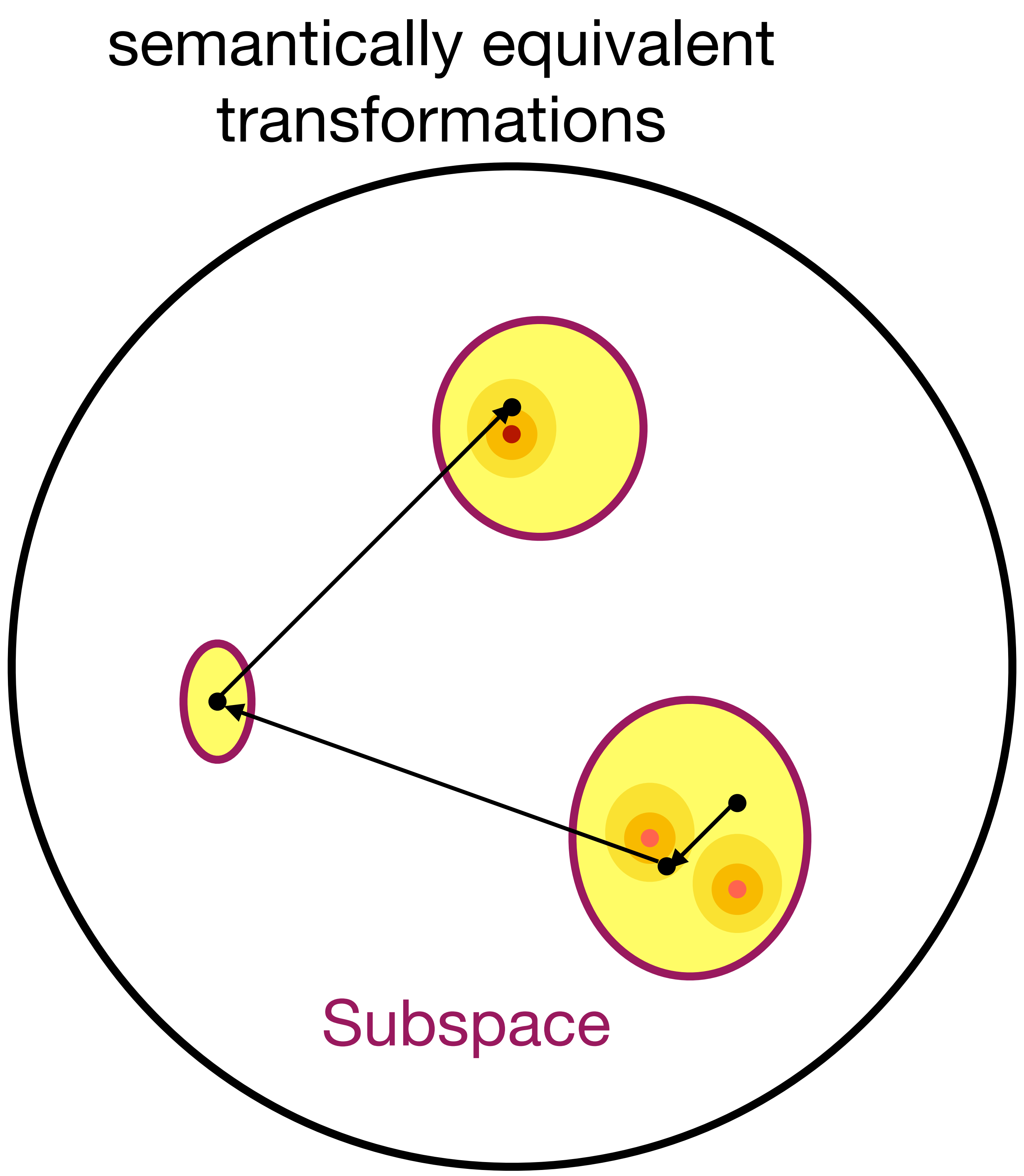


Program Embeddings

- In NLP, they use continuous representations of words that can be fed into a NN. These are known as word embeddings.
- They pre-train these embeddings (e.g., word2vec, GloVe embeddings)
- Similarly, programs can be embedded in continuous space.
- Challenges
  - Programs have strict semantics.
  - Programs have graph structure.
- Some efforts
  - Inst2vec
  - Blended Semantic Embeddings
  - PrograML
  - CuBERT
  - Contextual Flow Graphs and so on.

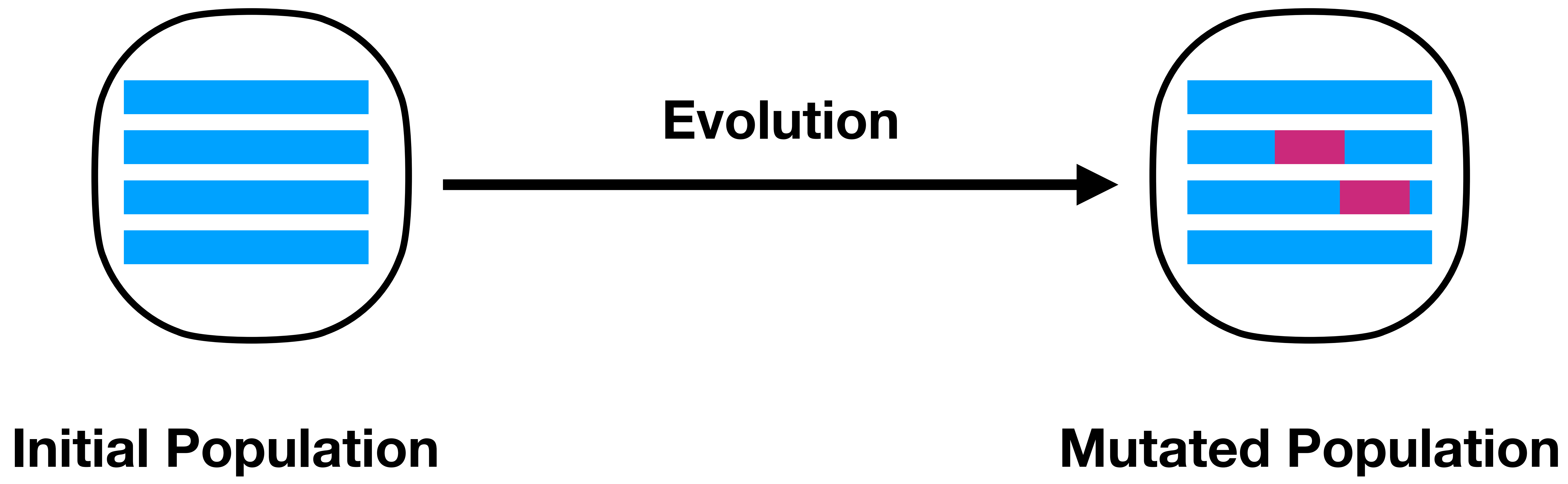
Optimization Strategies

- Two main ML options
  - **Search**
    - Genetic Algorithms
    - Beam Search
    - Monte Carlo Tree Search
  - **Learned**
    - Supervised Learning
    - Sequential Decision Making
    - Bayesian Optimization



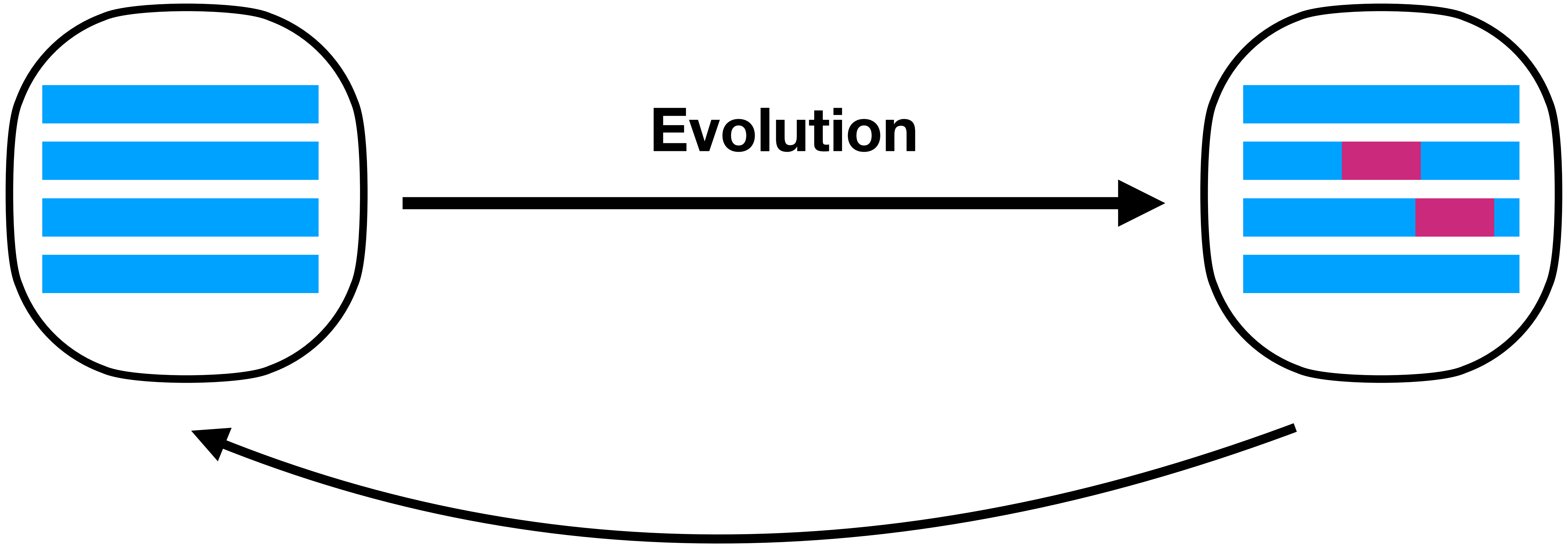
Genetic Algorithms

- Find the set of genes (parameters settings) that are the fittest (optimizes an objective) using genetic evolution.



Genetic Algorithms

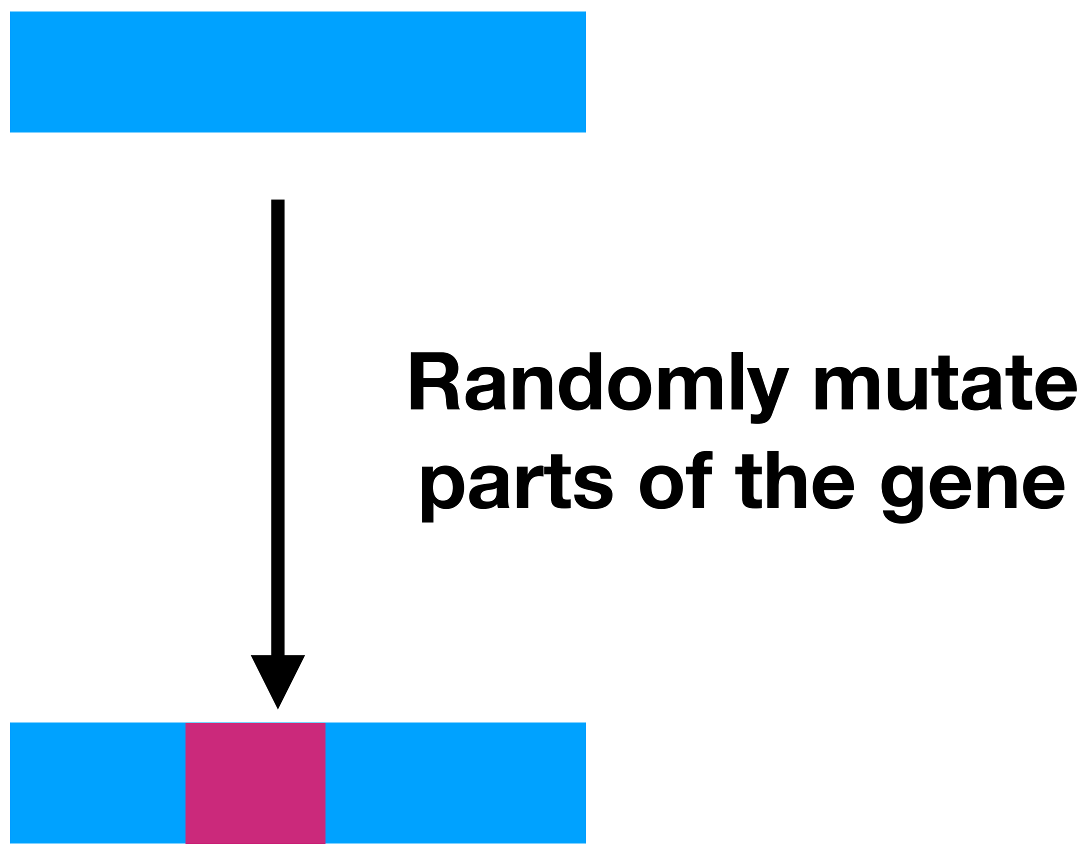
- Find the set of genes (parameters settings) that are the fittest (optimizes an objective) using genetic evolution.



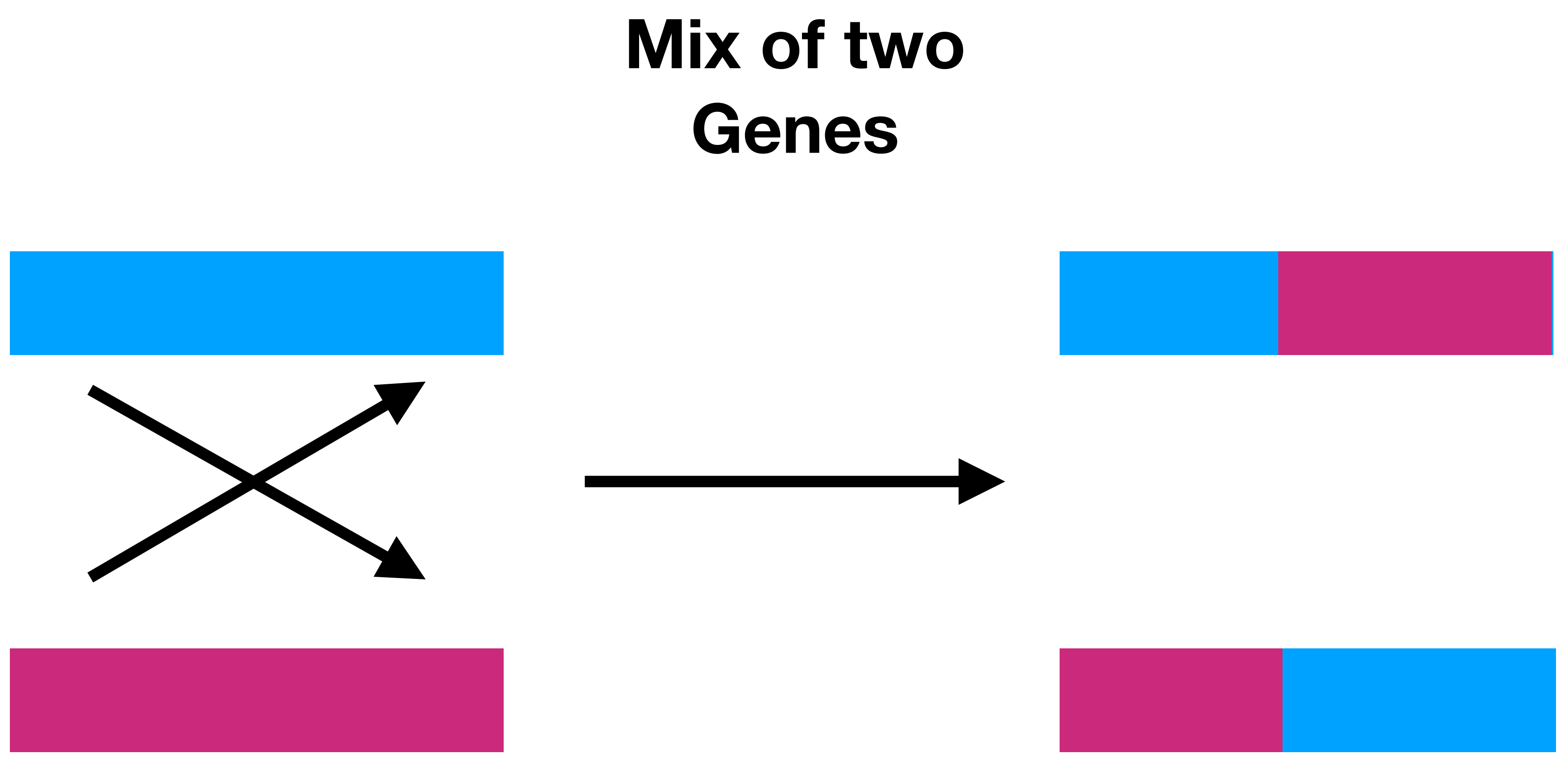
Repeat until budget exhausted or population meets convergence criteria

Evolution

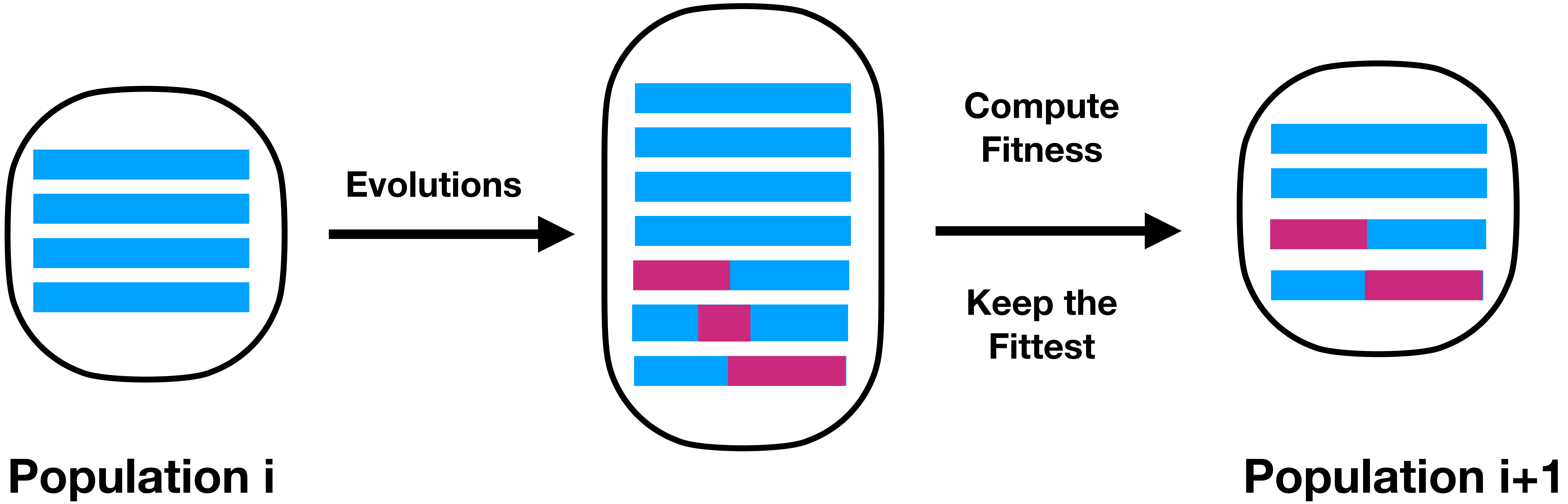
Mutations



Crossovers



Evolution

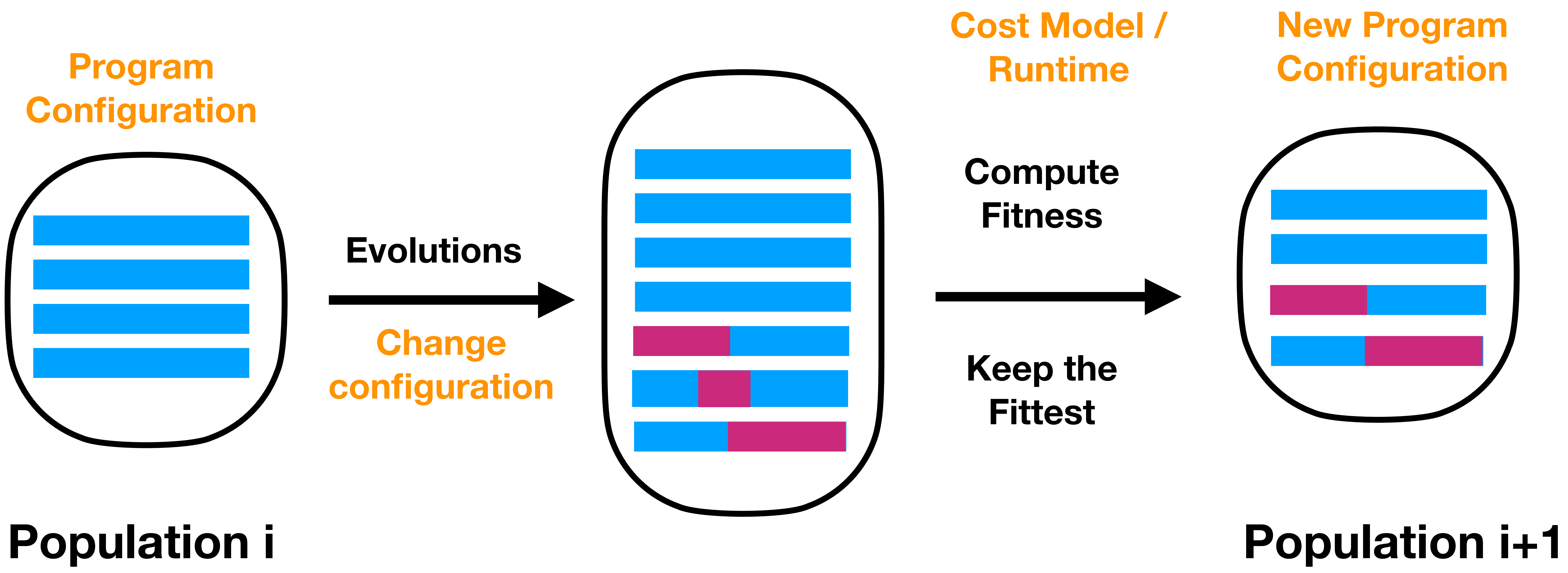


Evolutions
Compute
Fitness
Keep the
Fittest
Population i
Population i+1

Auto-tuning

- Generally, tuning parameters of a fixed set of transformations.
    - e.g. deciding on the unroll factor, tiling factor, vectorization factor
- Also extends to deciding the transformations themselves
    - e.g. Deciding when to unroll or not
- In either case, auto-tuning **searches** for the best performing code transformations.

Auto-tuning



Program
Configuration
Evolutions
Change
configuration
Cost Model /
Runtime
Compute
Fitness
Keep the
Fittest
New Program
Configuration
Population i
Population i+1

Auto-tuning use cases

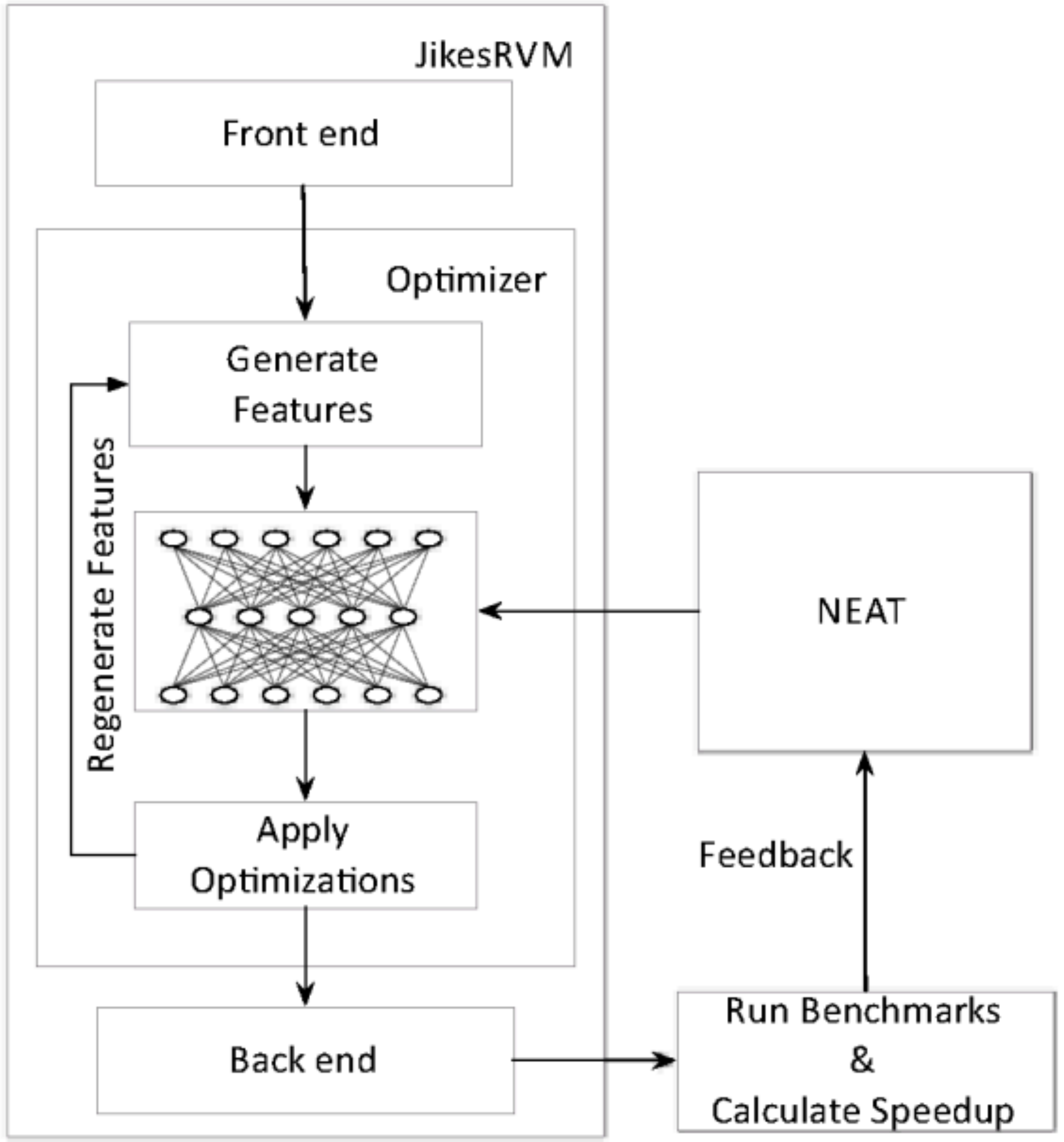
Mitigating the Compiler Optimization Phase-Ordering Problem using Machine Learning

Sameer Kulkarni John Cavazos
University of Delaware
{skulkarn,cavazos}@cis.udel.edu

Meta Optimization: Improving Compiler Heuristics with Machine Learning

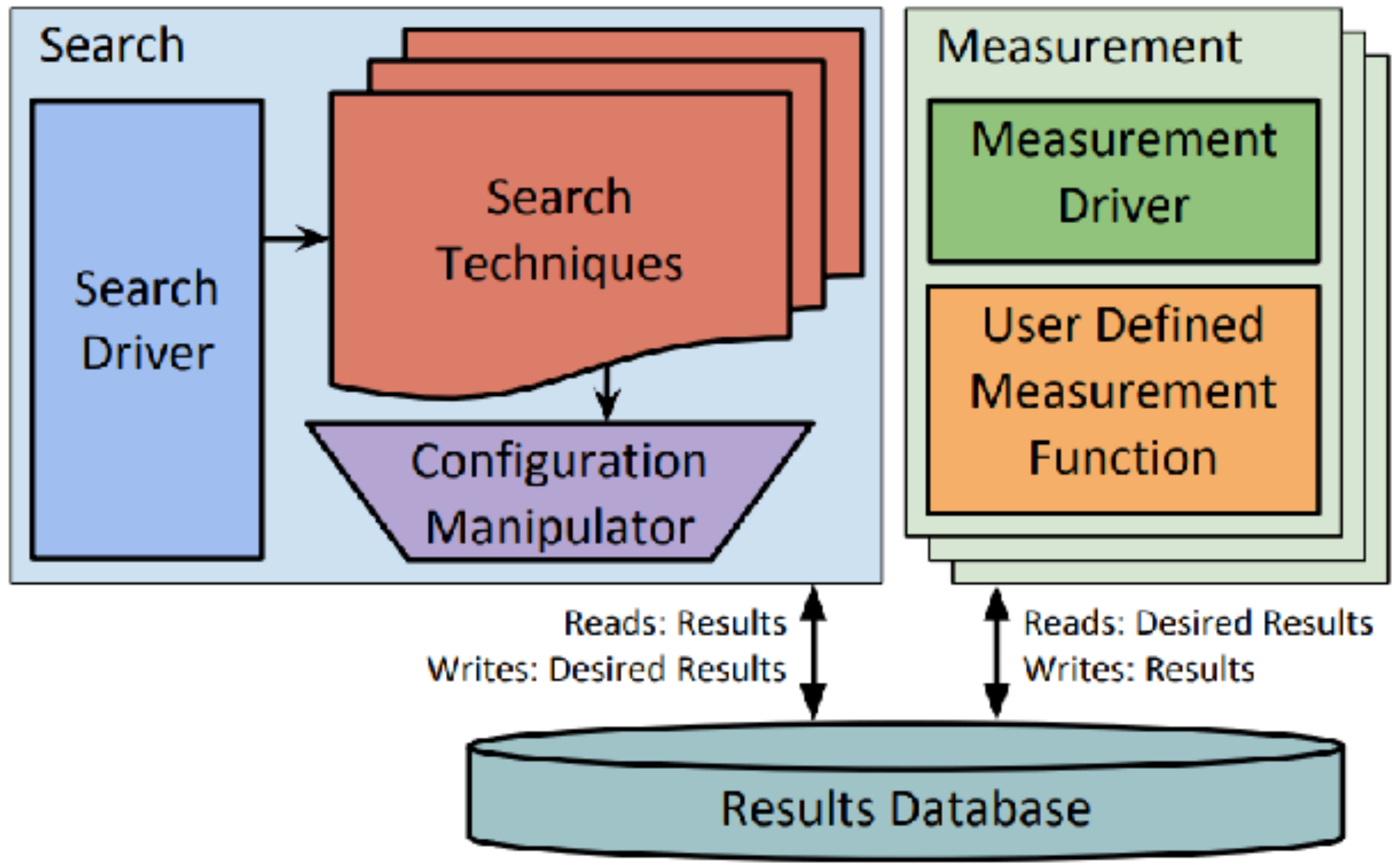
Mark Stephenson and Saman Amarasinghe
Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, MA 02139
{mstephen, saman}@cag.lcs.mit.edu

Martin Martin and Una-May O'Reilly
Massachusetts Institute of Technology
Artificial Intelligence Laboratory
Cambridge, MA 02139
{mcm, unamay}@ai.mit.edu



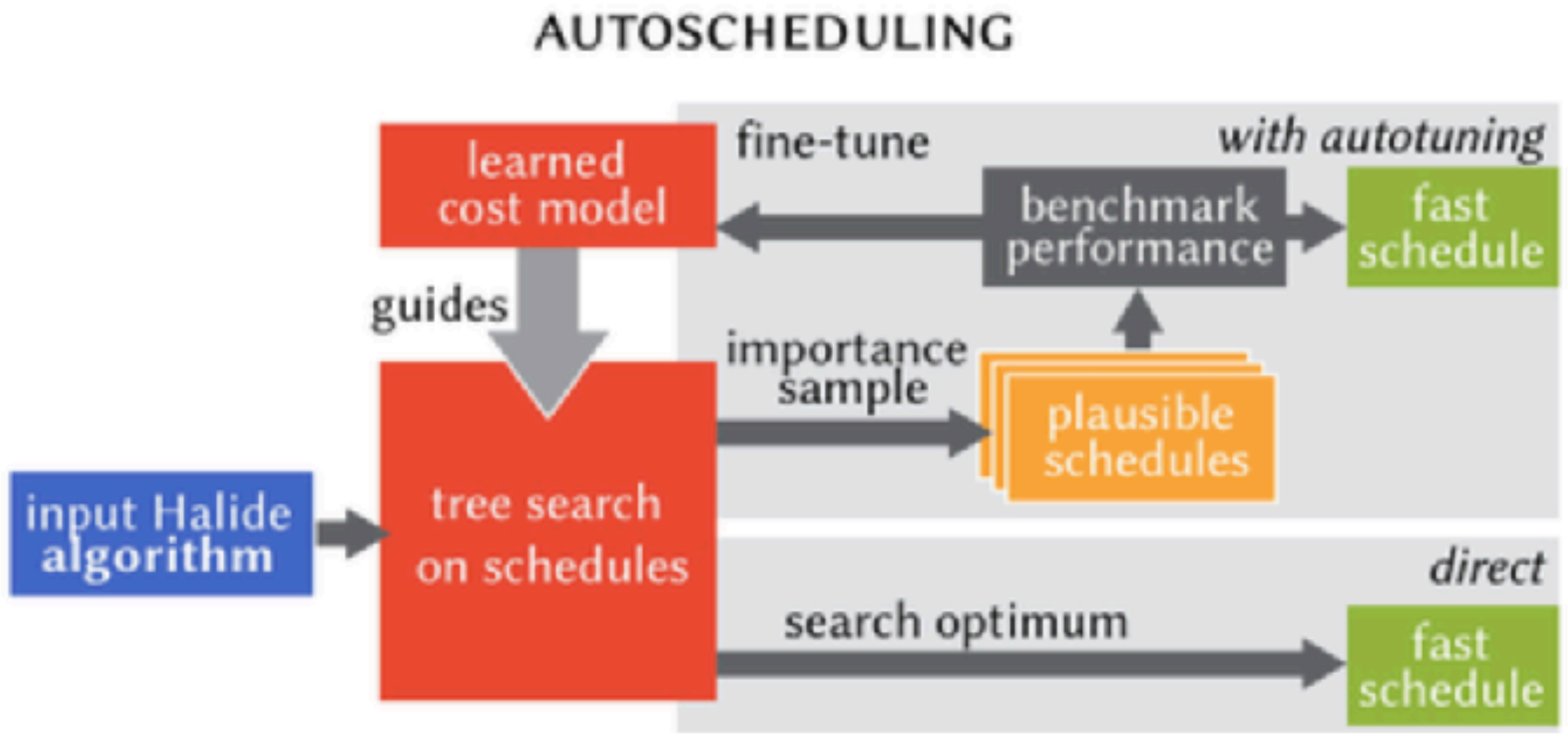
Auto-tuning using OpenTuner

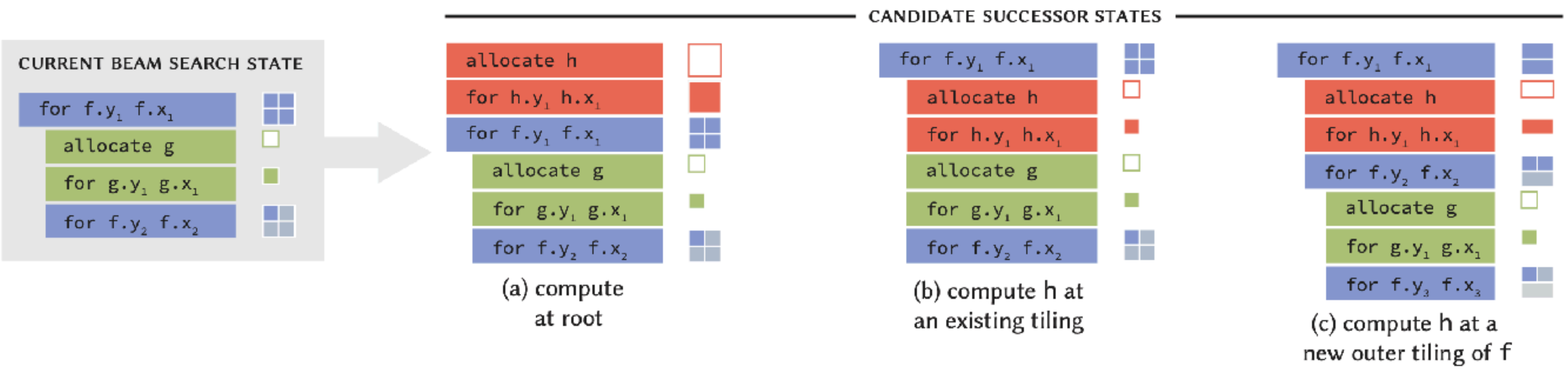
- A general framework for program auto-tuning
- Provides an interface
  - To specify parameter spaces
  - To specify search strategies
  - To specify multi-objective tuning
- Provides a meta-optimization heuristic
  - Multi-arm bandit technique



Ansel et. Al, “OpenTuner: An Extensible Framework for Program Auto-tuning”, PACT 2014

Auto-tuning DSLs

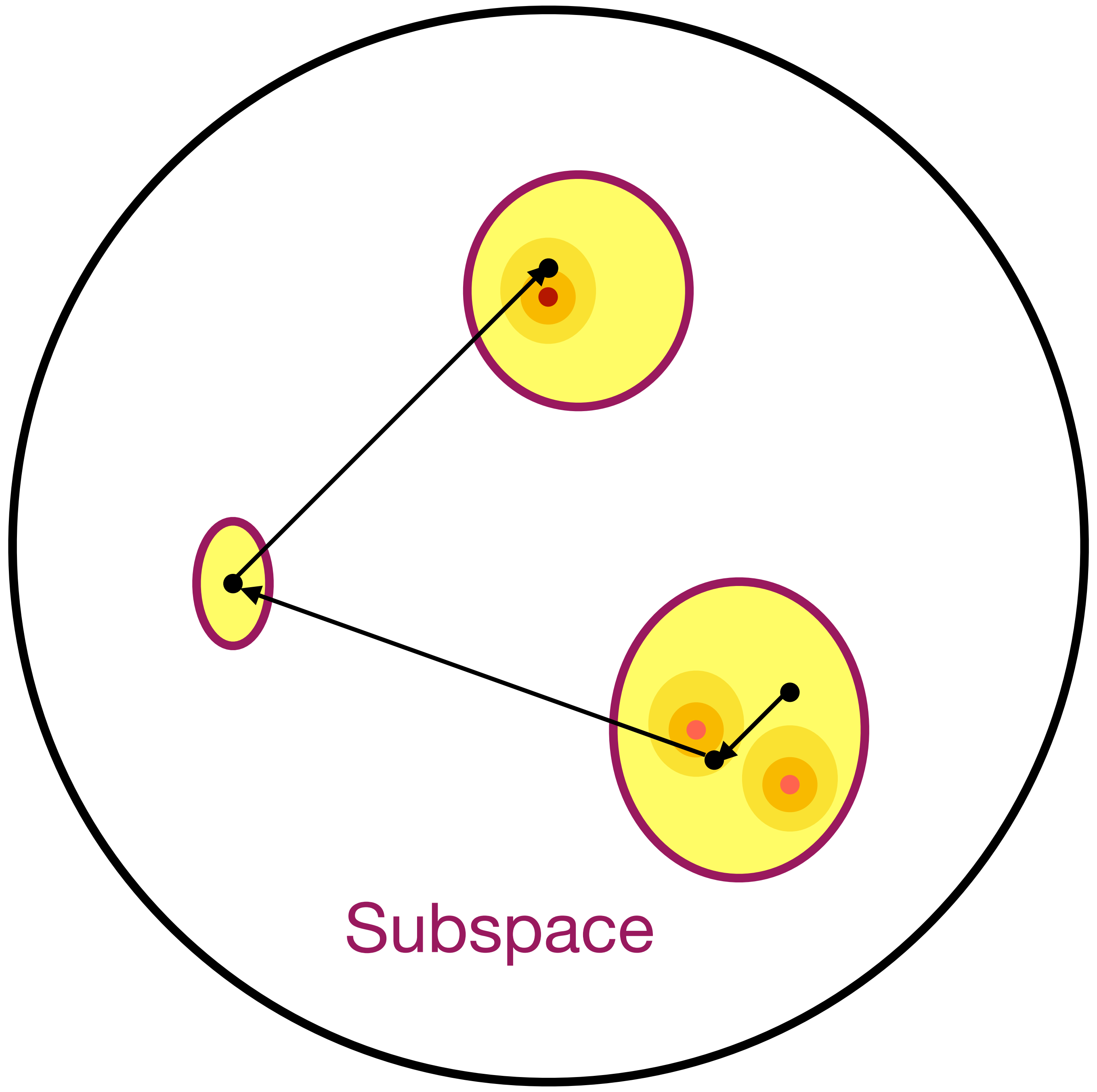




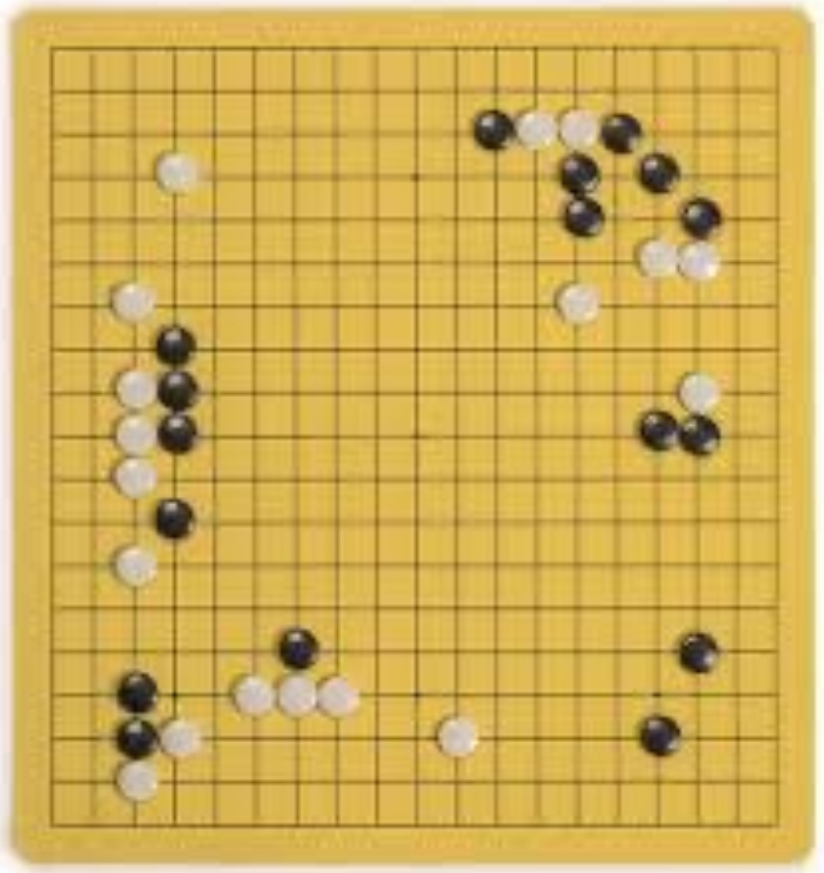
Andrew et. al “Learning to Optimize Halide with Tree Search and Random Programs” SIGGRAPH 2019

Optimization Strategies

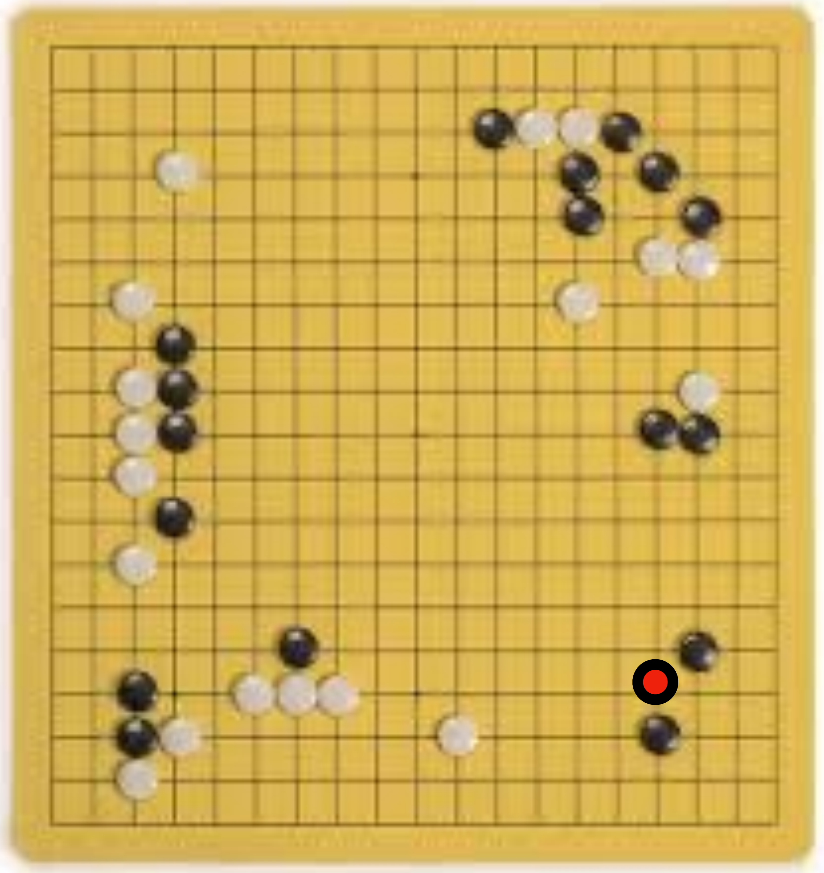
- Two main ML options
  - **Search**
    - Genetic Algorithms
    - Beam Search
    - Monte Carlo Tree Search
  - **Learned**
    - Supervised Learning
    - Sequential Decision Making
    - Bayesian Optimization



Sequential Decision Making



Choose a “valid” action



Iterate

State ← **New State**

↓

Reward (Win / Loss)

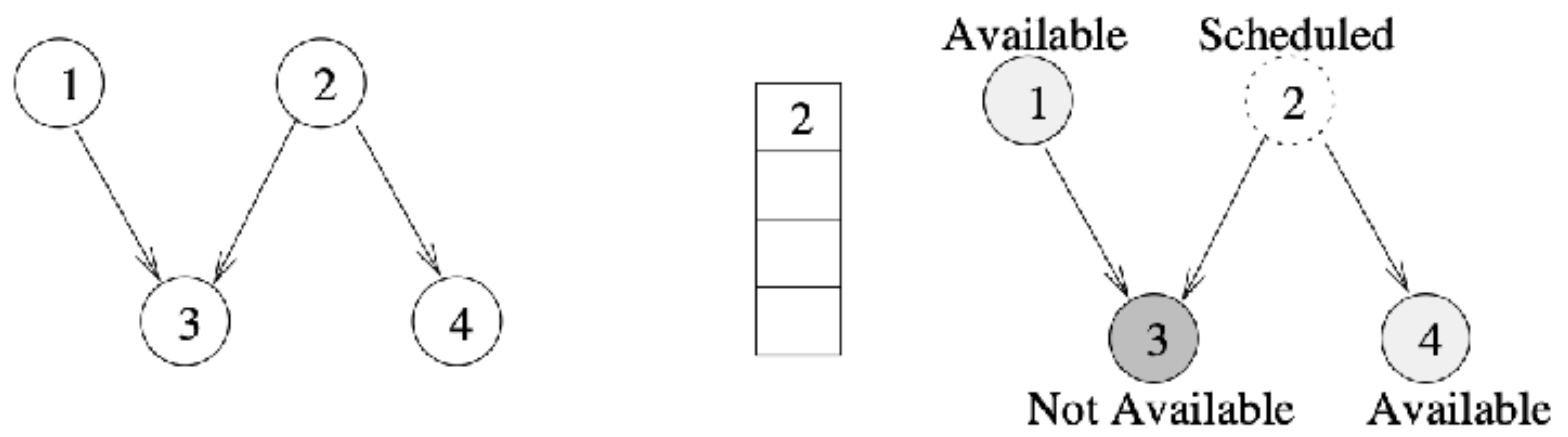
Markov Decision Process (MDP)

Sequential Decision Making

Building a Basic Block Instruction Scheduler with Reinforcement Learning and Rollouts

AMY MCGOVERN amy@cs.umass.edu
ELIOT MOSS moss@cs.umass.edu
ANDREW G. BARTO barto@cs.umass.edu
Department of Computer Science, University of Massachusetts, Amherst, Amherst, MA 01003, USA

Instruction Scheduling

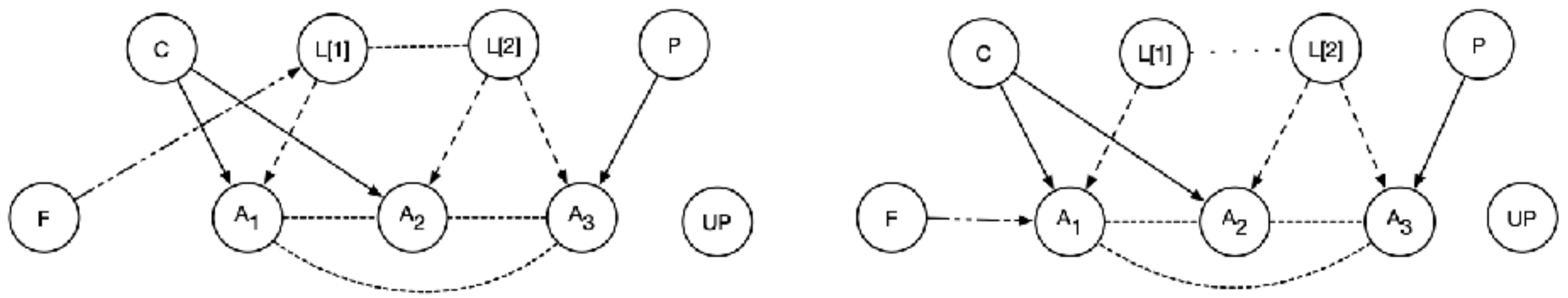


(c) Dependence Dag of Instructions (d) Partial Schedule

Compiler Auto-Vectorization with Imitation Learning

Charith Mendis, MIT CSAIL, charithm@mit.edu
Cambridge Yang, MIT CSAIL, camyang@csail.mit.edu
Yewen Pu, MIT CSAIL, yewenpu@mit.edu
Saman Amarasinghe, MIT CSAIL, saman@csail.mit.edu
Michael Carbin, MIT CSAIL, mcarbin@csail.mit.edu

Auto-vectorization



(b) (c)

Optimization decisions trigger state transitions

Vectorization as a Markov Decision Process

a[1] = b[1] + c[1]
a[2] = b[2] + c[2]
a[3] = a[1] + c[3]
a[4] = a[2] + c[4]
a[5] = b[5] * c[5]

State

Choose a “valid” action

{a[1],a[2]}
{a[3],a[4]}

{a[1],a[4]}
{a[2],a[3]}
{a[3],a[4]}

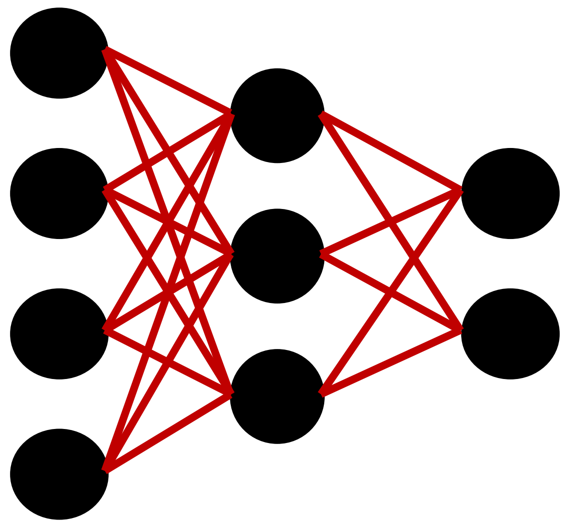
a[1:2] = b[1:2] + c[1:2]
a[3:4] = a[1:2] + c[3:4]
a[5] = b[5] * c[5]

New State

Iterate

Reward (Speed of execution)

What we do to solve this MDP



≈ Oracle goSLP

```
a[1] = b[1] + c[1]
a[2] = b[2] + c[2]
a[3] = a[1] + c[3]
a[4] = a[2] + c[4]
a[5] = b[5] * c[5]
```

State

Choose a “valid” action →

```
a[1:2] = b[1:2] + c[1:2]
a[3]   = b[3]   + c[3]
a[4]   = b[4]   + c[4]
a[5]   = b[5]   * c[5]
```

New State

← Iterate

Use Imitation Learning

Learnt Vectorization - Vemal

Mendis et. al “Compiler Auto-Vectorization with Imitation Learning” [NeurIPS’19]

Collect Demonstrations

Oracle
goSLP

```
a[1] = b[1] + c[1]
a[2] = b[2] + c[2]
a[3] = a[1] + c[3]
a[4] = a[2] + c[4]
```

{a[1],a[2]}

```
a[1:2] = b[1:2] + c[1:2]
a[3]   = a[1] + c[3]
a[4]   = a[2] + c[4]
```

State-Action Pairs

```
a[1] = b[1] + c[1]
a[2] = b[2] + c[2]
a[3] = a[1] + c[3]      ,   {a[1],a[2]}
a[4] = a[2] + c[4]
```

Learnt Vectorization - Vemal

Mendis et. al “Compiler Auto-Vectorization with Imitation Learning” [NeurIPS’19]

Collect Demonstrations

Oracle
goSLP

{a[3],a[4]}

```
a[1:2] = b[1:2] + c[1:2]
a[3]   = a[1] + c[3]
a[4]   = a[2] + c[4]
```

State-Action Pairs

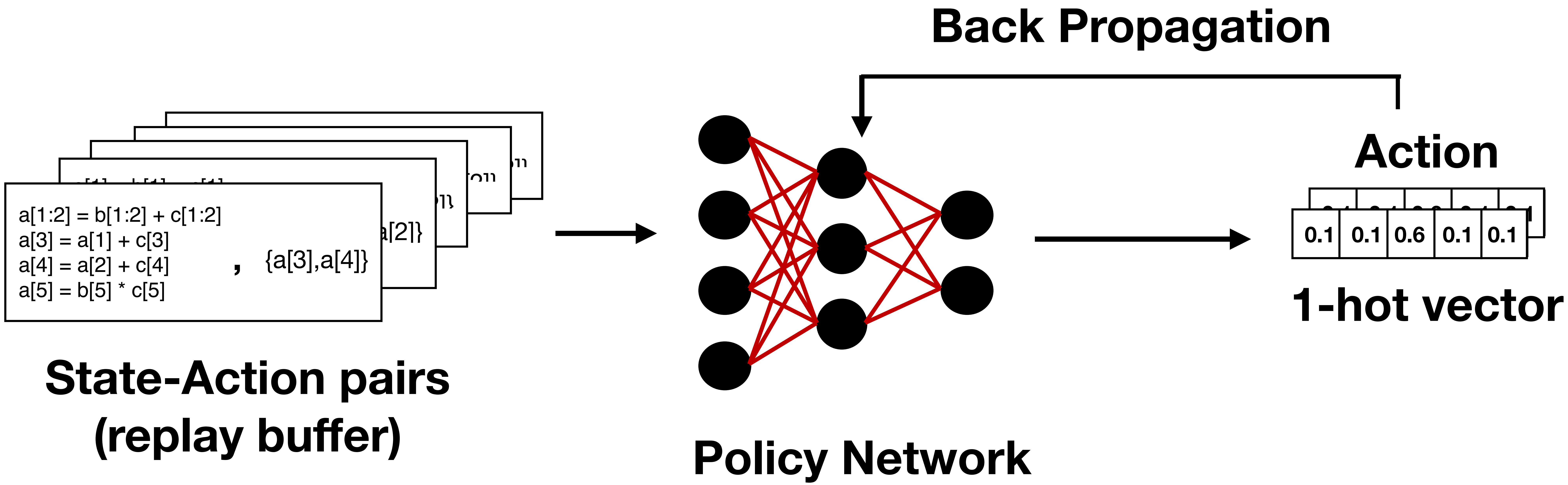
```
a[1:2] = b[1:2] + c[1:2]
a[3] = a[1] + c[3]
a[4] = a[2] + c[4]
a[5] = b[5] * c[5]
```

, {a[3],a[4]}

Learnt Vectorization - Vemal

Mendis et. al “Compiler Auto-Vectorization with Imitation Learning” [NeurIPS’19]

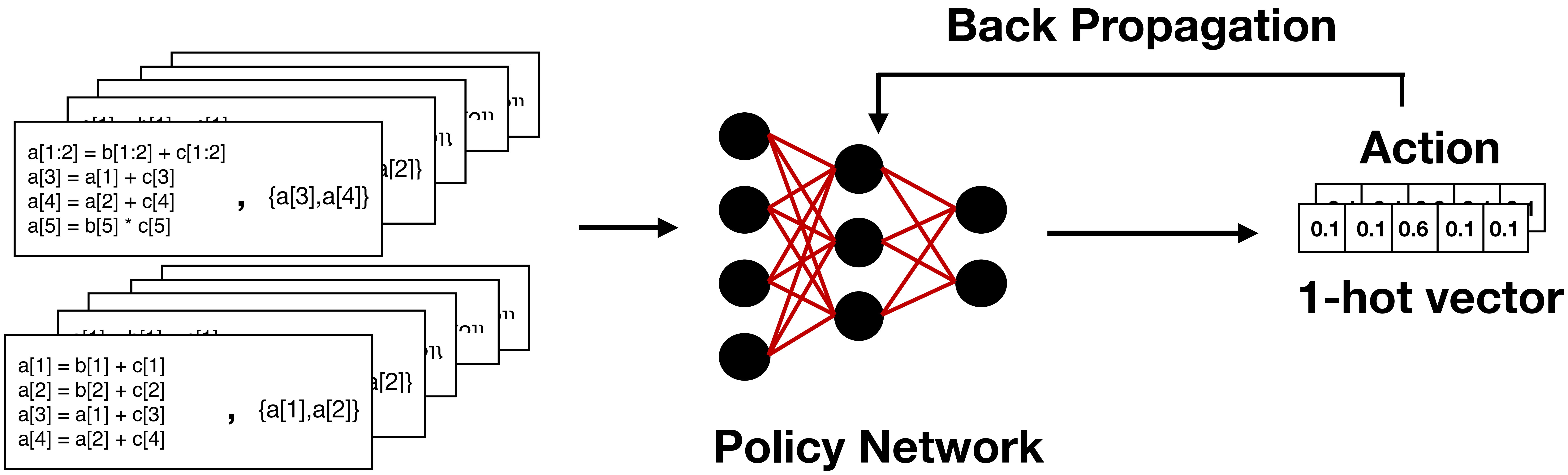
Training



Learnt Vectorization - Vemal

Mendis et. al “Compiler Auto-Vectorization with Imitation Learning” [NeurIPS’19]

Training



Augmented Replay Buffer

DAGGER to augment dataset at each epoch

Ross et. al [AISTATS’11]

Thank You!

- We are almost there! Presentations, report and the final to go.
- I enjoyed teaching the class. I hope you all know more about compilers before we started the semester.
- Please give class feedback at,

https://ices.citl.illinois.edu