SAT-based Program Synthesis

Ruben Martins

SAT/SMT/AR Summer School 2019
July 6, 2019

Carnegie Mellon University

What is Program Synthesis?

Specifications ϕ **Program** P

$$\exists P. \forall x. \phi(x, P(x))$$

- Find a program P that for all inputs x meets the specification ϕ

Programming by Examples

Email		First Name
Nancy.Freehafer@fourthcoffee.com	→	Nancy
Andrew.Cencini@northwindtraders.com		Andrew
Jan.Kotas@itwareinc.com		Jan
Mariya.Sergienko@graphicdesigns.com		Mariya
Alexander.David@contoso.com		Alexander
Amr.Zaid@traders.com		Amr

Programming by Examples

Email	First Name
Nancy.Freehafer@fourthcoffee.com	Nancy
Andrew.Cencini@northwindtraders.com	Andrew
Jan.Kotas@itwareinc.com	Jan
Mariya.Sergienko@graphicdesigns.com	Mariya
Alexander.David@contoso.com	Alexander
Amr.Zaid@traders.com	Amr

- Flash Fill (Excel 2013 feature):
  - Automating **string processing** in spreadsheets using input-output examples. POPL 2011

Programming by Examples

Name	Month	Rate1	Rate2
Aira	1	12	23
Aira	2	18	73
Ben	1	53	19
Ben	2	22	87
Cat	1	22	87
Cat	2	67	43

Name	avg1	avg2
Aira	15.0	48
Ben	37.5	53
Cat	44.5	65

- Can we find a program that automatically **transforms tables** given input-output examples?

Programming by Examples

Name	Month	Rate1	Rate2
Aira	1	12	23
Aira	2	18	73
Ben	1	53	19
Ben	2	22	87
Cat	1	22	87
Cat	2	67	43

Name	avg1	avg2
Aira	15.0	48
Ben	37.5	53
Cat	44.5	65

R program:

```
TBL_15=group_by(p8_input1,`Name`)
TBL_7=summarise(TBL_15,avg2=mean(`Rate2`))
TBL_3=inner_join(TBL_7,p8_input1)
TBL_1=group_by(TBL_3,`Name`,`avg2`)
morpheus=summarise(TBL_1,avg1=mean(`Rate1`))
morpheus=select(morpheus,1,3,2)
```

- Component-based synthesis of **table** consolidation and **transformation** tasks from examples. PLDI 2017

Programming by Examples

- Can we find a **sequence of API calls** using **java.time** in Java 8 to get the day from a Date in string format?

```
public static int getDayFromString(String date, String pat) {


                    }

public static boolean test() {
   return (getDayFromString("2013/06/13", "yyyy/MM/dd") == 13); }
```

Programming by Examples

- Can we find a **sequence of API calls** using **java.time** in Java 8 to get the day from a Date in string format?

```
public static int getDayFromString(String date, String pat) {
    DateTimeFormatter dtf = DateTimeFormatter.ofPattern(pat);
    LocalDate localdate = LocalDate.parse(date, dtf);
    int day = localdate.getDayOfMonth();
    return day; }

public static boolean test() {
  return (getDayFromString("2013/06/13", "yyyy/MM/dd") == 13); }
```

- Component-based **synthesis for complex APIs**. POPL 2017

Who can Program Synthesis help?

Who can Program Synthesis help?

- Are we trying to replace programmers? **No!**
  - We want to make programmers life easier
  - Automating tedious and repetitive tasks

Who can Program Synthesis help?

- Are we trying to replace programmers? **No!**
  - We want to make programmers life easier
  - Automating tedious and repetitive tasks
- 99% of computer users **cannot program**!
  - They struggle with simple repetitive tasks
  - Help non-CS people to automate their daily tasks

How do Program Synthesizers Work?

Enumerative Search

Stochastic Search

Constraint Solving

How do Program Synthesizers Work?

Enumerative Search

Stochastic Search

Constraint Solving

- Combinatorial search for all possible programs

How do Program Synthesizers Work?

Enumerative Search

Stochastic Search

Constraint Solving

- Build a statistical models using large corpora
- Guide the search using statistical models

How do Program Synthesizers Work?

Enumerative Search

Stochastic Search

Constraint Solving

- Encode the synthesis problem to SAT/SMT
- Prune infeasible incomplete programs with logical deduction

Outline

- Examples of Synthesizers
- Synthesis of Java programs
- Conflict-driven Synthesis

Outline

- Examples of Synthesizers
- Synthesis of Java programs
- Conflict-driven Synthesis

Microsoft Program Synthesis using Examples SDK

A framework for automatic programming or data wrangling from input-output examples.

Latest version: 6.20.1 Release notes NuGet package Yeoman generator Samples

Program Synthesis Framework

Microsoft PROSE SDK is a framework of technologies for *programming by examples*: automatic generation of programs from input-output examples at runtime.

Given a domain-specific language (DSL) and some input-output examples for the desired program's behavior, PROSE synthesizes a ranked set of DSL programs that are consistent with the examples.

Data Wrangling DSLs

PROSE SDK includes a pre-defined suite of technologies for various kinds of *data wrangling* – cleaning and pre-processing raw semi-structure data into a form amenable to analysis:

- Flash Fill, a technology for *text transformation by examples*, available in Microsoft Excel and PowerShell.
- *Data extraction from text files* by examples, available in PowerShell and Azure Log Analytics.
- *Data extraction* and *transformation* of JSON by examples.
- *Predictive file splitting* technology, which splits a text file into the structured columns without any examples.

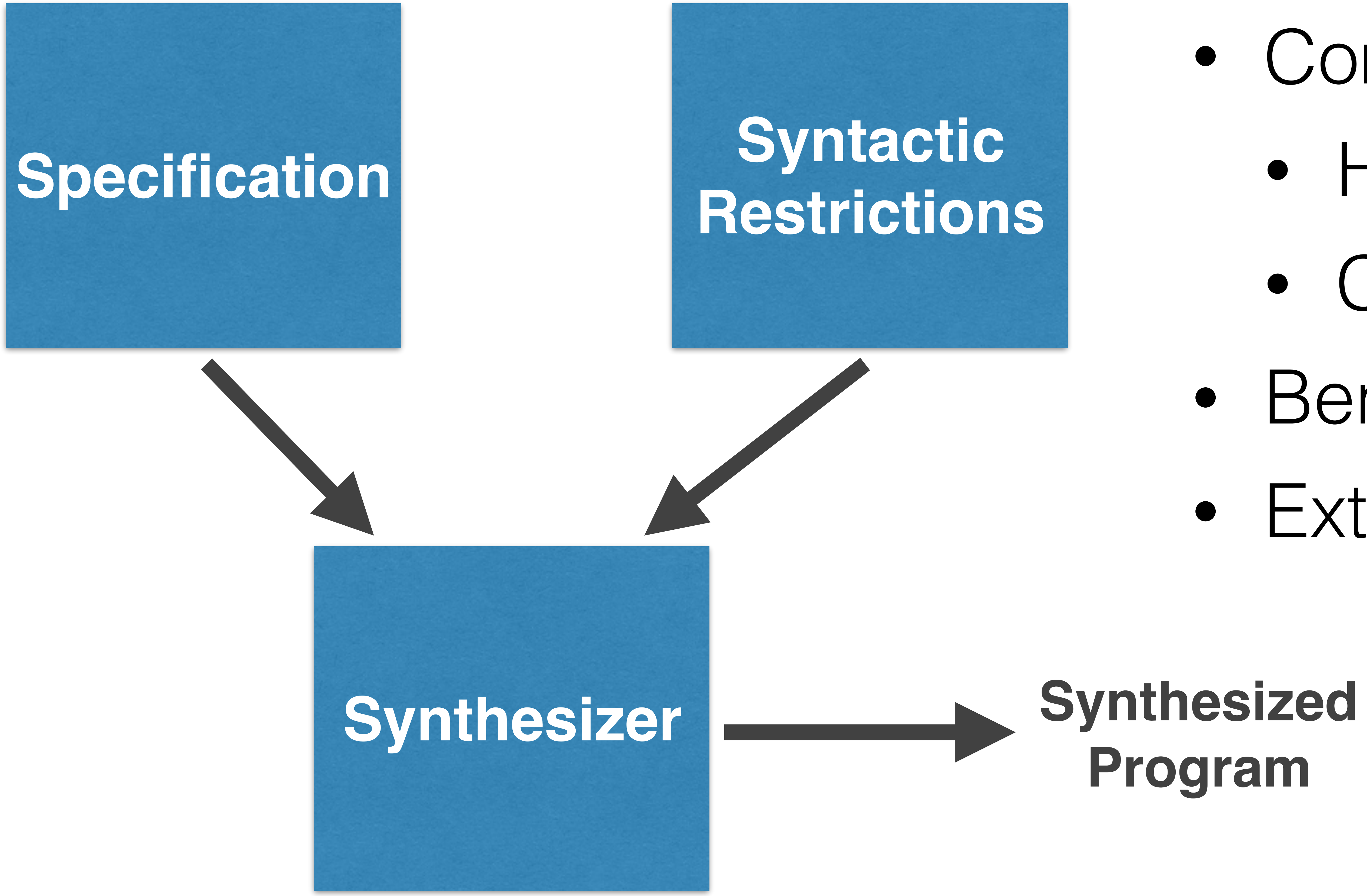
Demo PROSE

https://microsoft.github.io/prose/

Microsoft PROSE

- Program synthesis in a real-world scenario
- Very efficient for string manipulations
- Provides a ranking for the most likely solutions
- Requires witness functions for pruning the search space
- Hard to extend to other domains

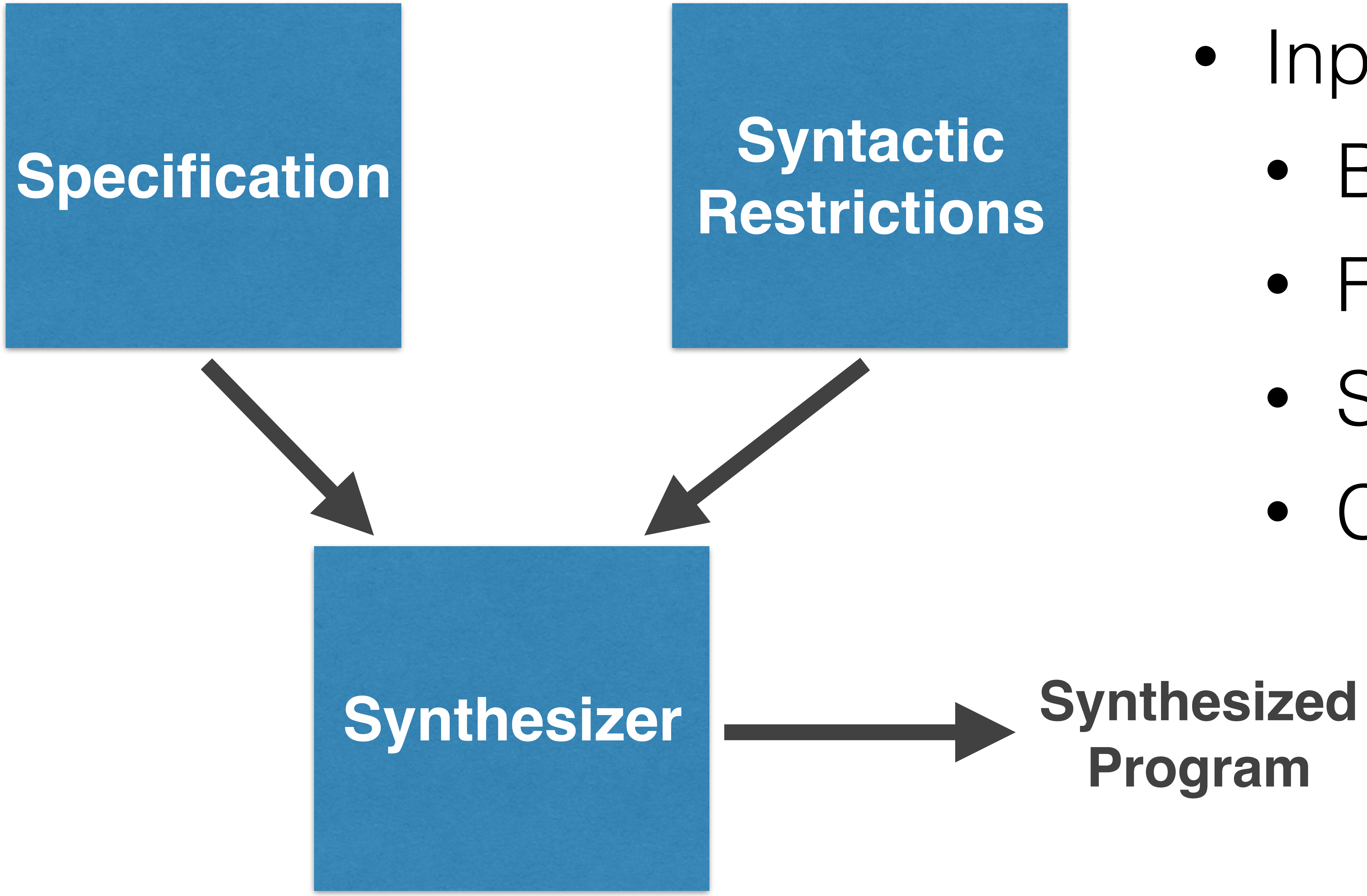
Syntax-Guided Synthesis (SyGuS)



- Combine:
  - Human expert insights
  - Constraint solving
- Benefit from progress in SAT / SMT
- Extends SMT-LIB to SYNTH-LIB

- Syntax-Guided Synthesis. FMCAD 2013

Syntax-Guided Synthesis (SyGuS)



- Inputs to SyGuS:
  - Background theory
  - Function to be synthesized
  - Specification
  - Context-free grammar

(set-logic LIA) $\longrightarrow$ Background theory (set-logic LIA) → Background theory

(synth-fun max2 ((x Int) (y Int)) Int → Function

```
(set-logic LIA)
(synth-fun max2 ((x Int) (y Int)) Int
    ((Start Int (x y 0 1
                 (+ Start Start)
                 (- Start Start)
                 (ite StartBool Start Start)))
     (StartBool Bool ((and StartBool StartBool)
                      (or StartBool StartBool)
                      (not StartBool)
                      (<= Start Start))))
(declare-var x Int)
(declare-var y Int)
```

Background theory

Function

Grammar

```
(set-logic LIA)
(synth-fun max2 ((x Int) (y Int)) Int
    ((Start Int (x y 0 1
                 (+ Start Start)
                 (- Start Start)
                 (ite StartBool Start Start)))
     (StartBool Bool ((and StartBool StartBool)
                      (or StartBool StartBool)
                      (not StartBool)
                      (<= Start Start))))
(declare-var x Int)
(declare-var y Int)
(constraint (>= (max2 x y) x))
(constraint (>= (max2 x y) y))
(constraint (or (= x (max2 x y)) (= y (max2 x y))))
```

Background theory

Function

Grammar

Specification

```
(set-logic LIA)
(synth-fun max2 ((x Int) (y Int)) Int
    ((Start Int (x y 0 1
                 (+ Start Start)
                 (- Start Start)
                 (ite StartBool Start Start)))
     (StartBool Bool ((and StartBool StartBool)
                      (or StartBool StartBool)
                      (not StartBool)
                      (<= Start Start))))
(declare-var x Int)
(declare-var y Int)
(constraint (>= (max2 x y) x))
(constraint (>= (max2 x y) y))
(constraint (or (= x (max2 x y)) (= y (max2 x y))))
(check-synth)
```

Background theory

Function

Grammar

Specification

Execute

Demo SyGus

http://sygus.seas.upenn.edu/

Syntax-Guided Synthesis

- Input is not specific to any synthesis problem
- Specifications allows for human insights
- Leverages the advances in SAT / SMT solving
- Theories are restricted to SMT-LIB
- Does not scale for large grammars

Program Synthesis as Sketching

```
harness void doubleSketch(int x){
    int t = x * ??;
    assert t == x + x;
}
```

- **Sketch** of the program:
  - Partial program with **holes** (“??”)
- Synthesizer finds values to the holes that satisfies the specifications
- Uses SAT / SMT technology to find the missing values

- Programming by sketching for bit-streaming programs. PLDI 2005

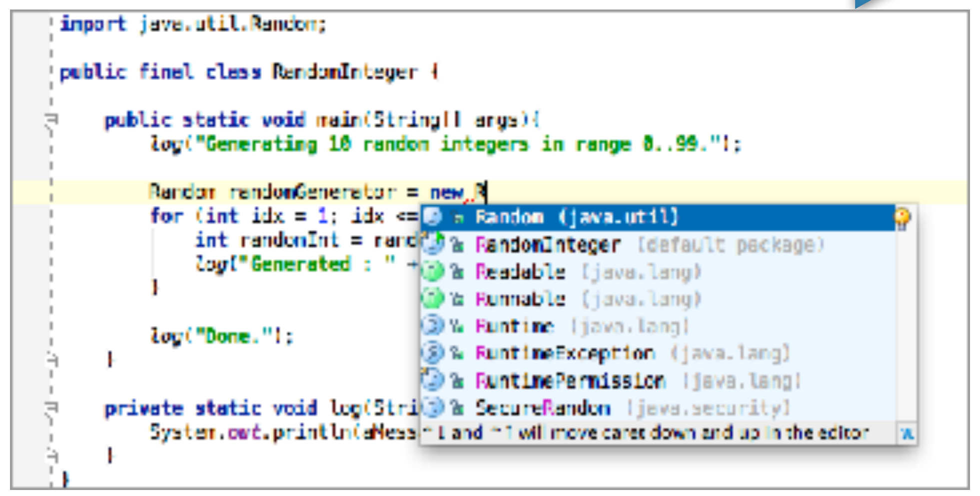
Demo Sketch

https://people.csail.mit.edu/asolar/

Sketching

- Synthesis real-world code (C, Java)
- Specifications allows for human insights
- Leverages the advances in SAT / SMT solving
- Holes must be expressed by SMT theories
- Writing the sketch may be as hard as writing the program

Synthesis of Java programs



Send HTTP request
Compute GCD
Rotate an image

Programmers spend a lot of effort
learning APIs!

- Component-based synthesis for complex APIs. POPL 2017

Synthesizing Programs with APIs

```java
public static int getDayFromString(String date, String pat) {



                    }


public static boolean test() {
  return (getDayFromString("2013/06/13", "yyyy/MM/dd") == 13); }
```

Synthesizing Programs with APIs

```
public static int getDayFromString(String date, String pat) {
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern(pat);
        LocalDate localdate = LocalDate.parse(date, dtf);
        int day = localdate.getDayOfMonth();
        return day; }

public static boolean test() {
    return (getDayFromString("2013/06/13", "yyyy/MM/dd") == 13); }
```

Click!

Demo SyPet

https://utopia-group.github.io/sypet/

SyPet

- Works for real-world code
- Can handle any Java library
- Scales for large libraries
- Does not work well for libraries with few types
- Does not support loops or conditionals

Outline

- Introduction to Syntax-Guided Synthesis (SyGus)
- Synthesis of Java code
- Conflict-driven Synthesis

Synthesizing Programs with APIs

```
public static int getDayFromString(String date, String pat) {
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern(pat);
        LocalDate localdate = LocalDate.parse(date, dtf);
        int day = localdate.getDayOfMonth();
        return day; }

public static boolean test() {
    return (getDayFromString("2013/06/13", "yyyy/MM/dd") == 13); }
```

How to Find the Correct Program?

How to Find the Correct Program?

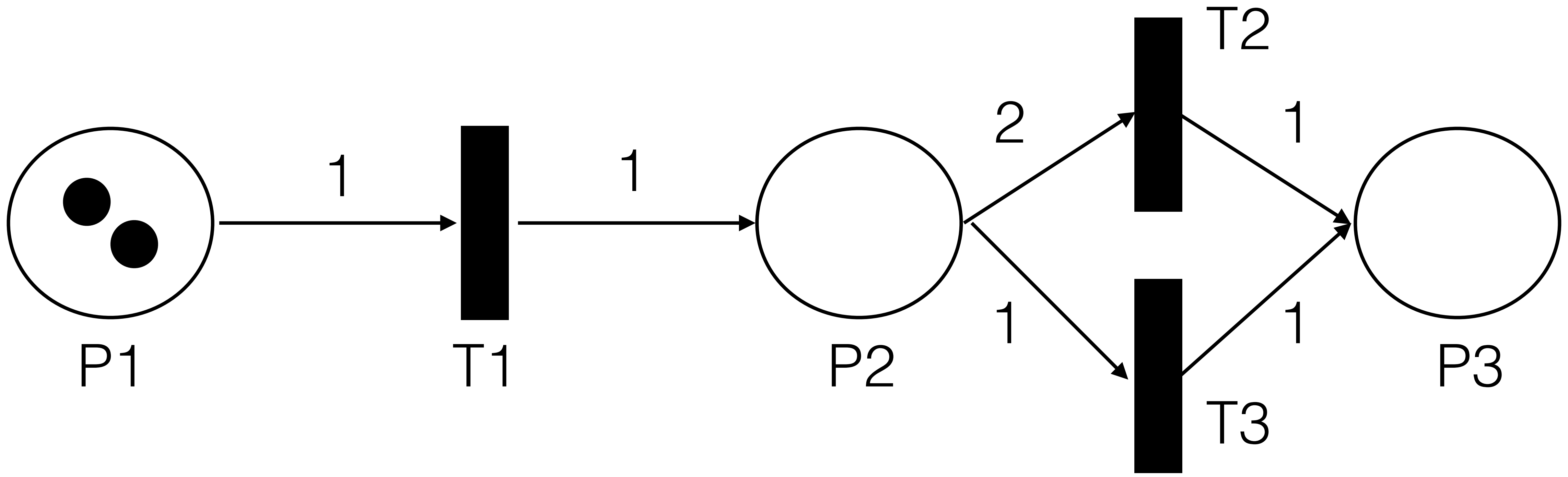
Use **Petri net reachability** analysis to look for well-typed programs of the desired type

How to Find the Correct Program?

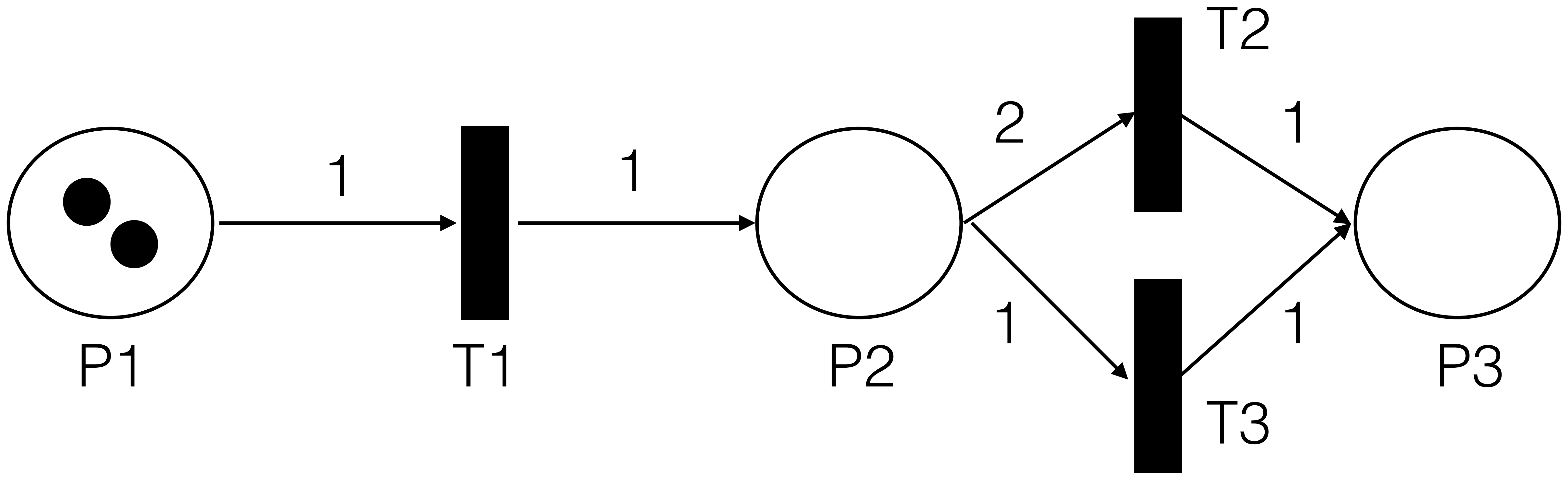
Use **Petri net reachability** analysis to look for well-typed programs of the desired type

- Model relationships between APIs using Petri nets
- Use type signature of desired method to mark **initial** and **target** configurations
- Perform **reachability analysis** to find valid sequences of method calls

Petri Nets in a Nutshell

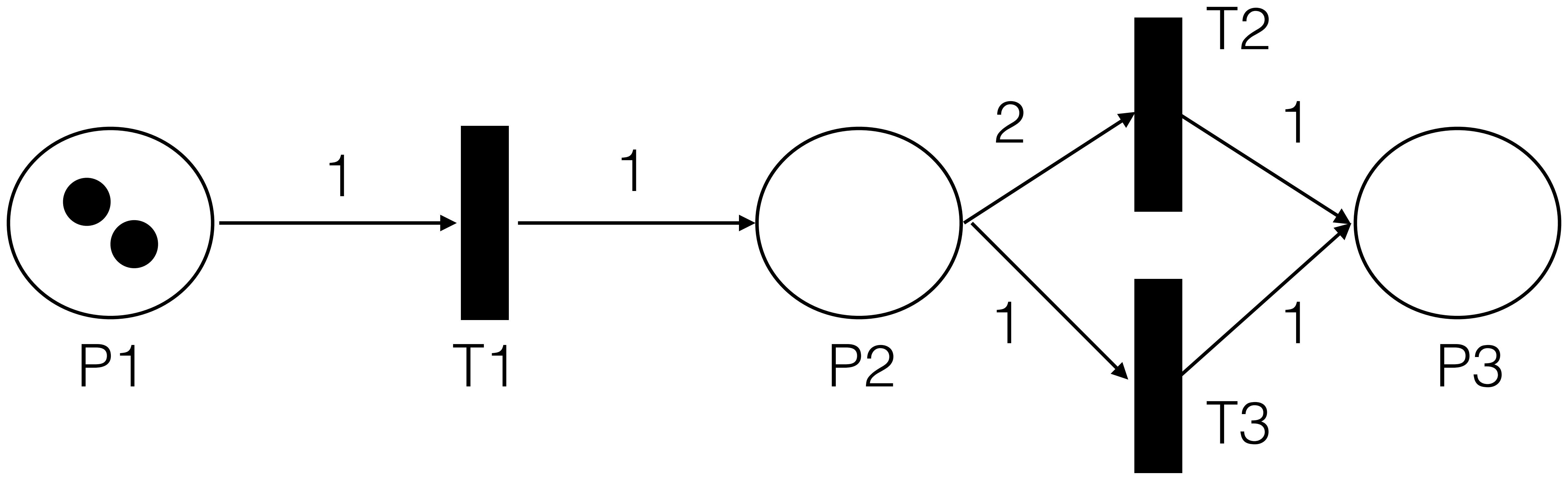


Petri Nets in a Nutshell



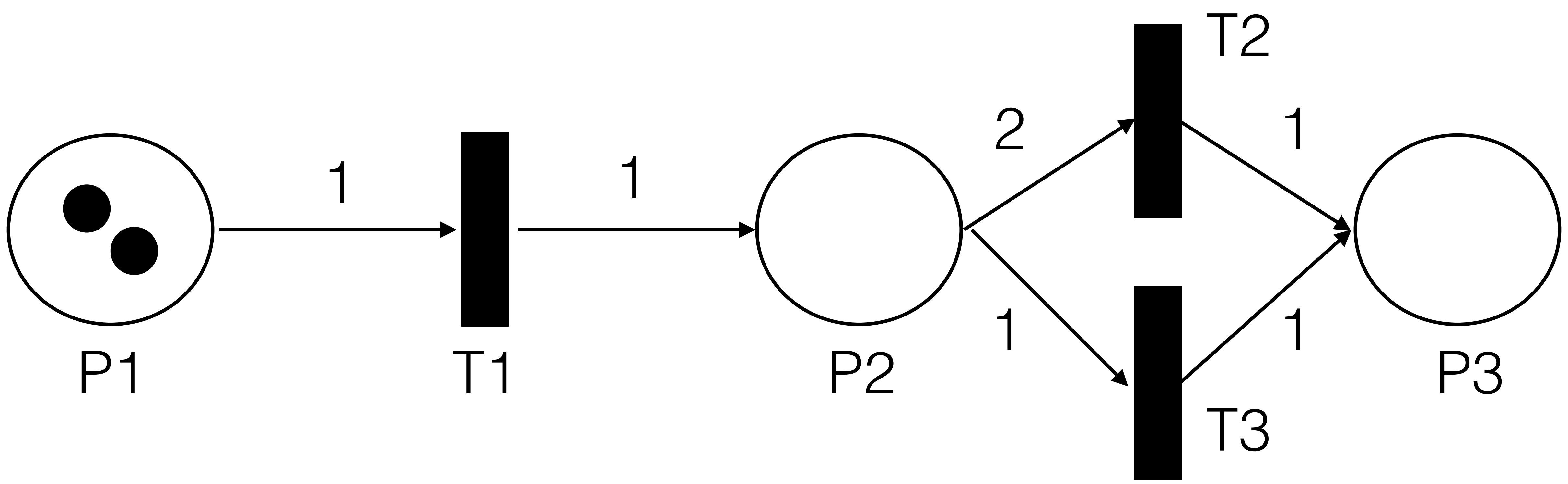
- Petri net is a generalized graph with two kinds of nodes: **places** and **transitions**

Petri Nets in a Nutshell



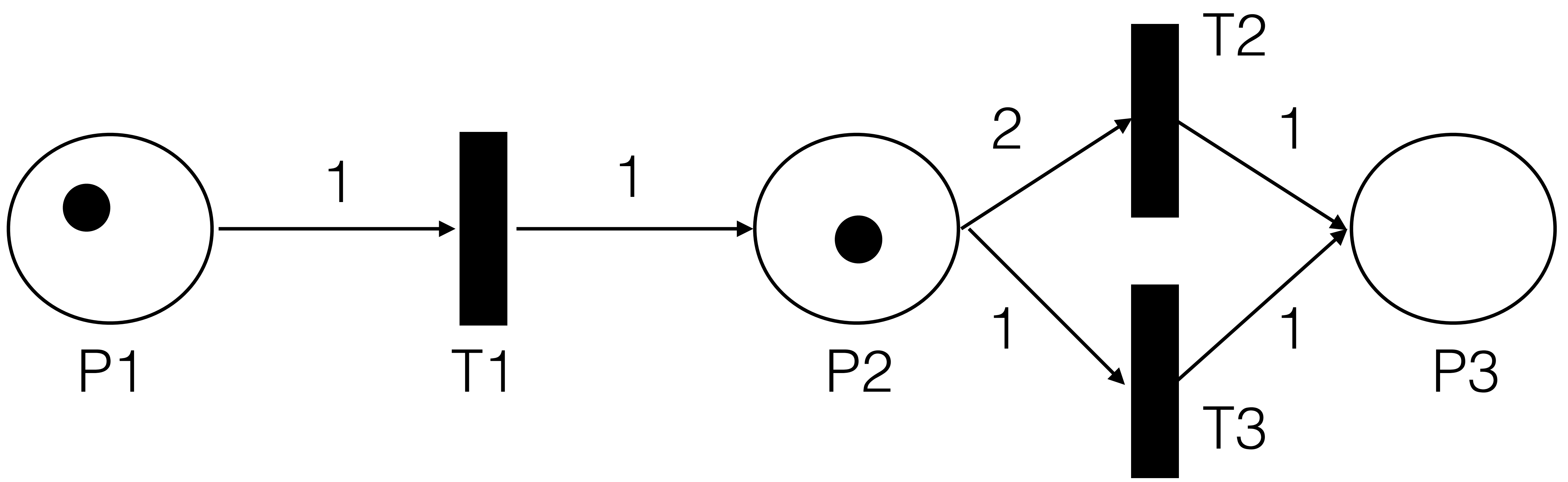
- Petri net is a generalized graph with two kinds of nodes: **places** and **transitions**
- Each place contains zero or more tokens; edges are labeled with a number of tokens

Petri Nets in a Nutshell



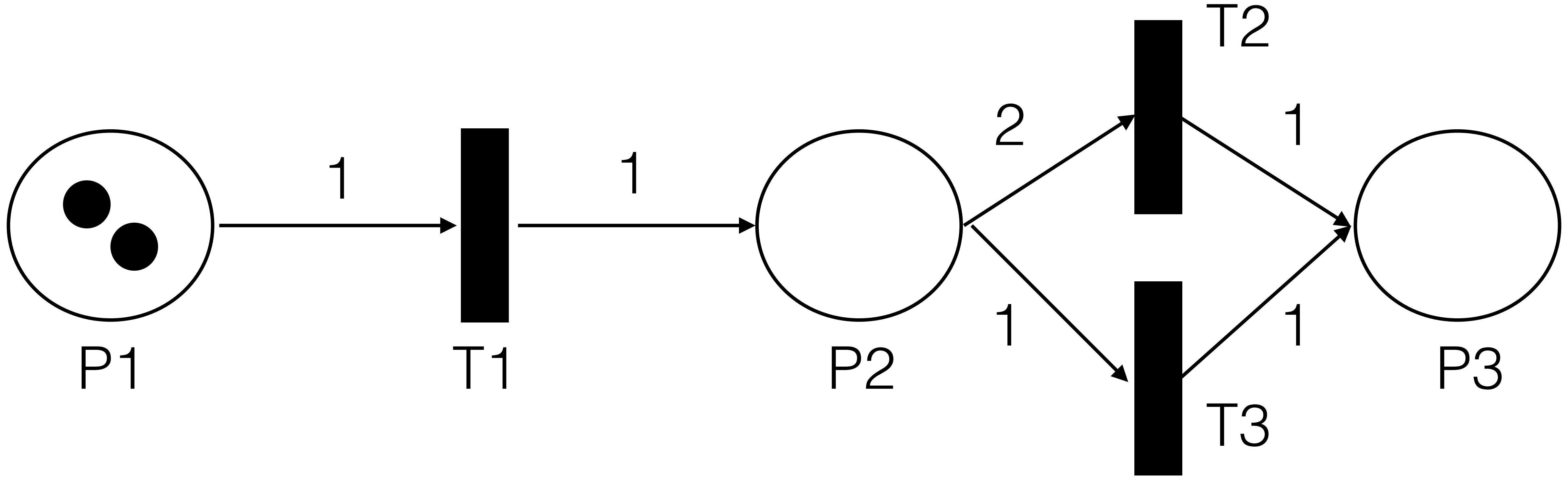
- A transition T **can fire if**, for each incoming edge (p,T) with label n, place p **contains at least** n tokens
- **Firing** a transition T **consumes** (resp. produces) the indicated number of tokens at the source (resp. target) nodes

Petri Nets in a Nutshell



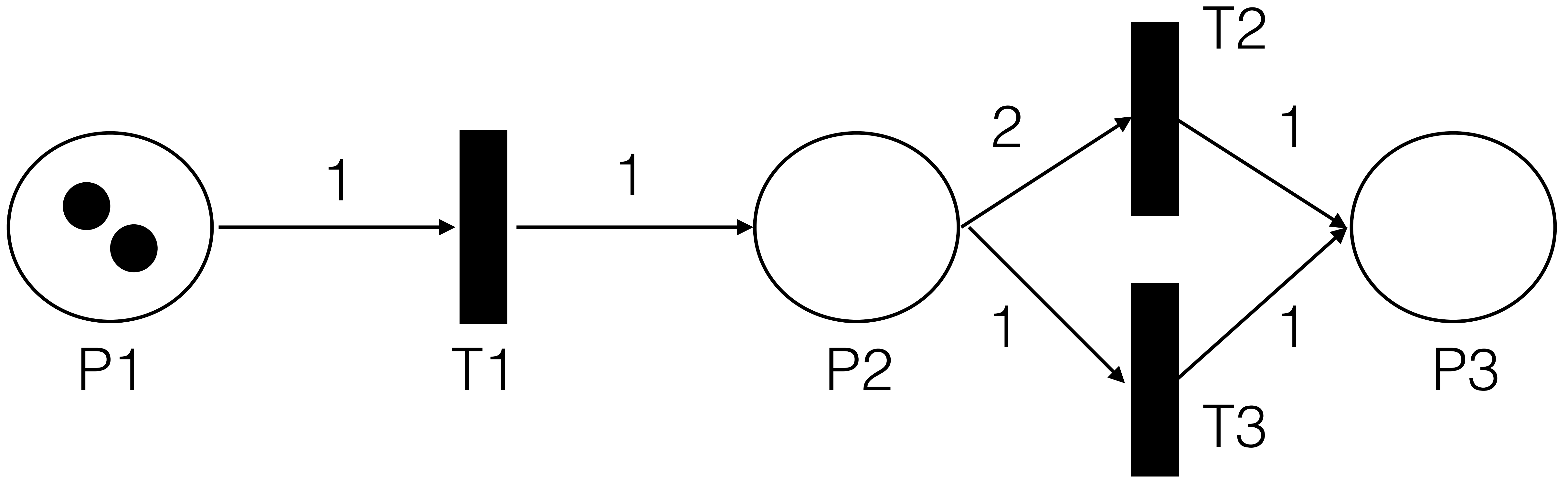
- A transition T **can fire if**, for each incoming edge (p,T) with label n, place p **contains at least** n tokens
- **Firing** a transition T **consumes** (resp. produces) the indicated number of tokens at the source (resp. target) nodes

Reachability Problem in Petri Nets



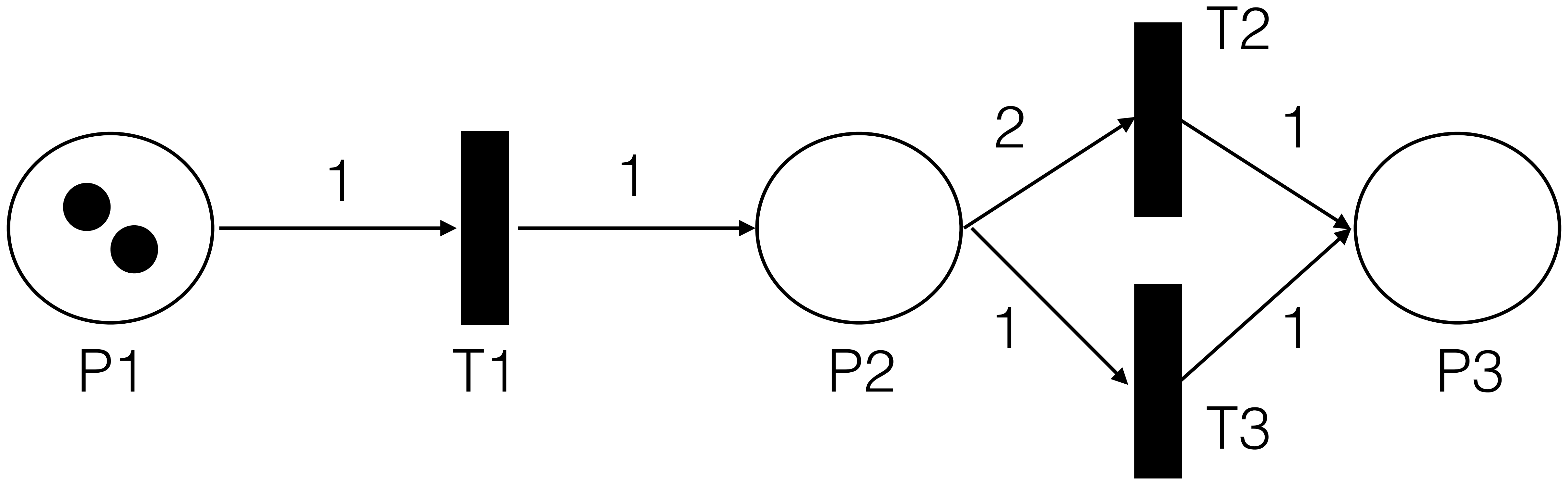
- **Reachability problem:** Given a Petri net with initial marking M and a target marking M', is it possible to obtain M' by firing a sequence of transitions?

Reachability Problem in Petri Nets



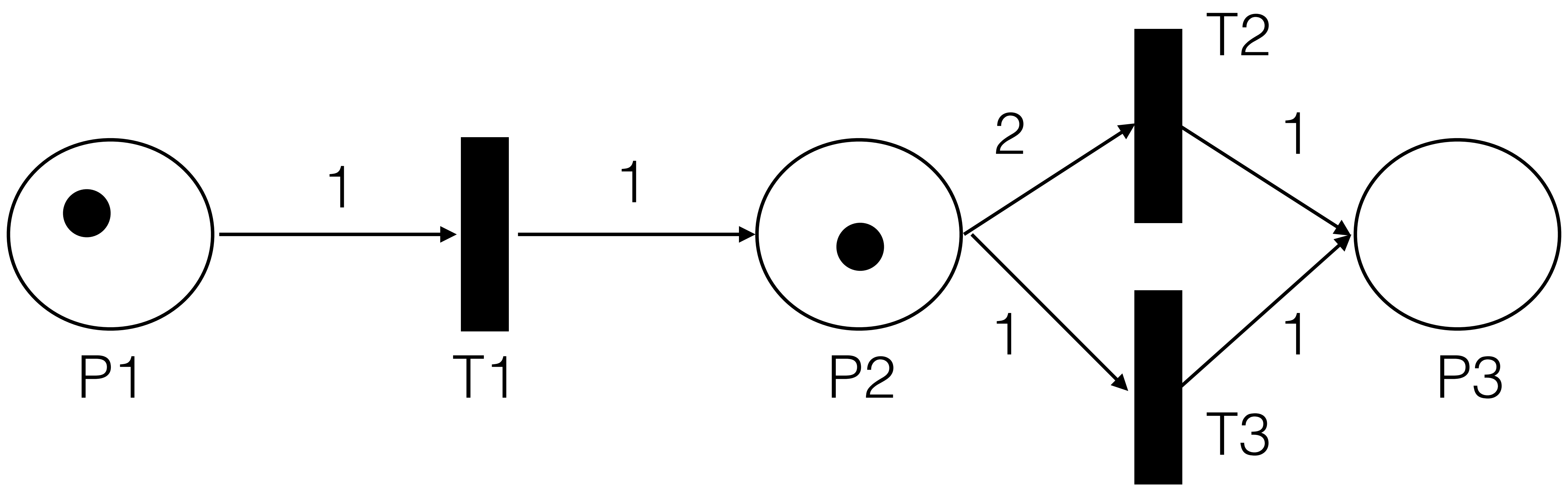
- **Example:** Consider marking M' : [P1→0,P2→0,P3→1].

Reachability Problem in Petri Nets



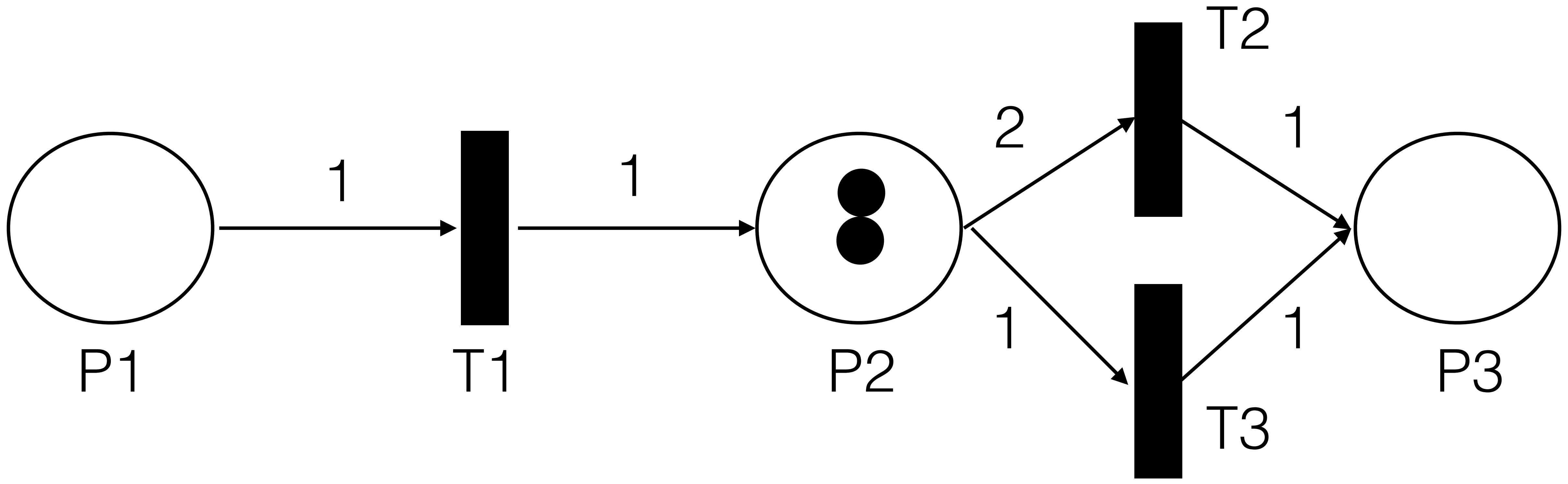
- **Example:** Consider marking M' : [P1→0,P2→0,P3→1].
- This marking is reachable, and accepting run is T1,T1,T2.

Reachability Problem in Petri Nets



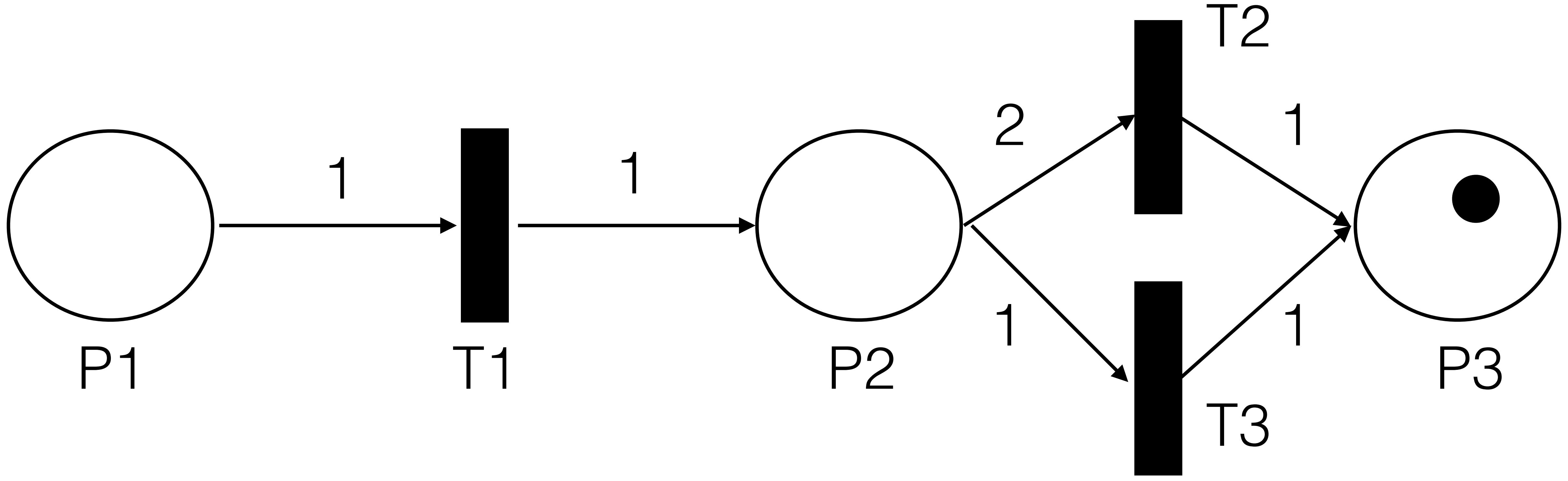
- **Example:** Consider marking M’ : [P1→0,P2→0,P3→1].
- This marking is reachable, and accepting run is T1,T1,T2.

Reachability Problem in Petri Nets



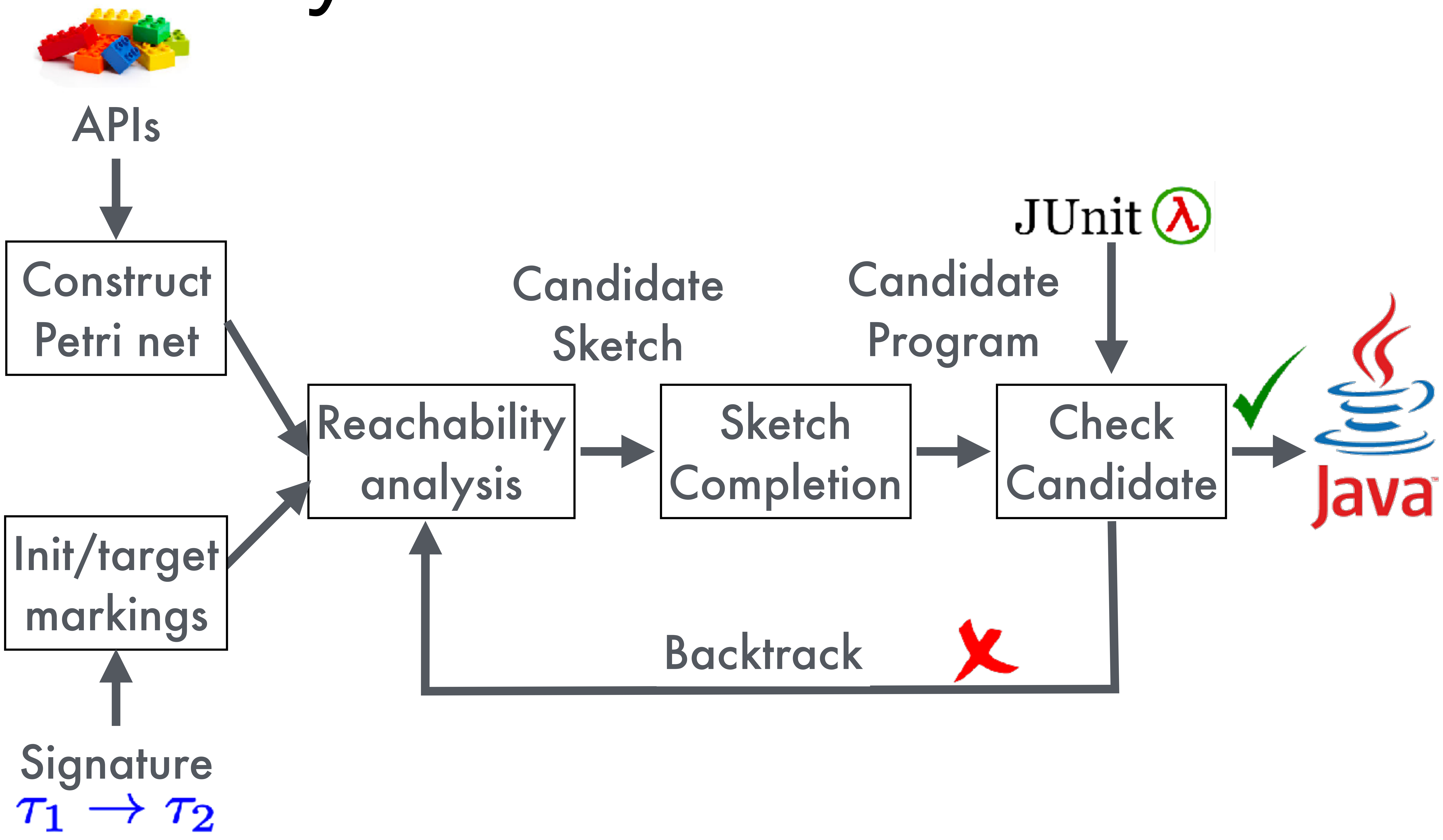
- **Example:** Consider marking M' : [P1→0,P2→0,P3→1].
- This marking is reachable, and accepting run is T1,T1,T2.

Reachability Problem in Petri Nets

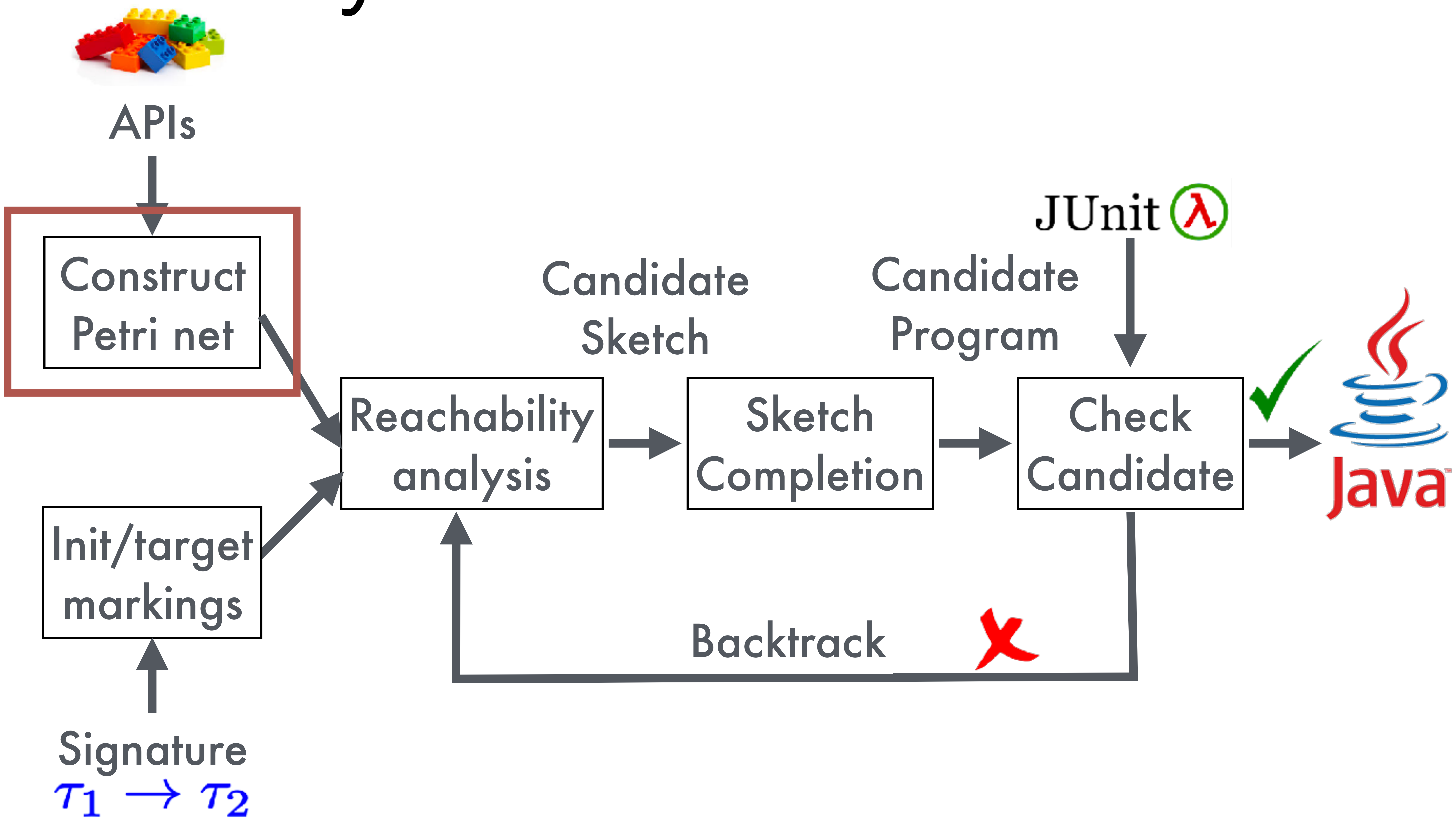


- **Example:** Consider marking M' : [P1→0,P2→0,P3→1].
- This marking is reachable, and accepting run is T1,T1,T2.

SyPet Architecture

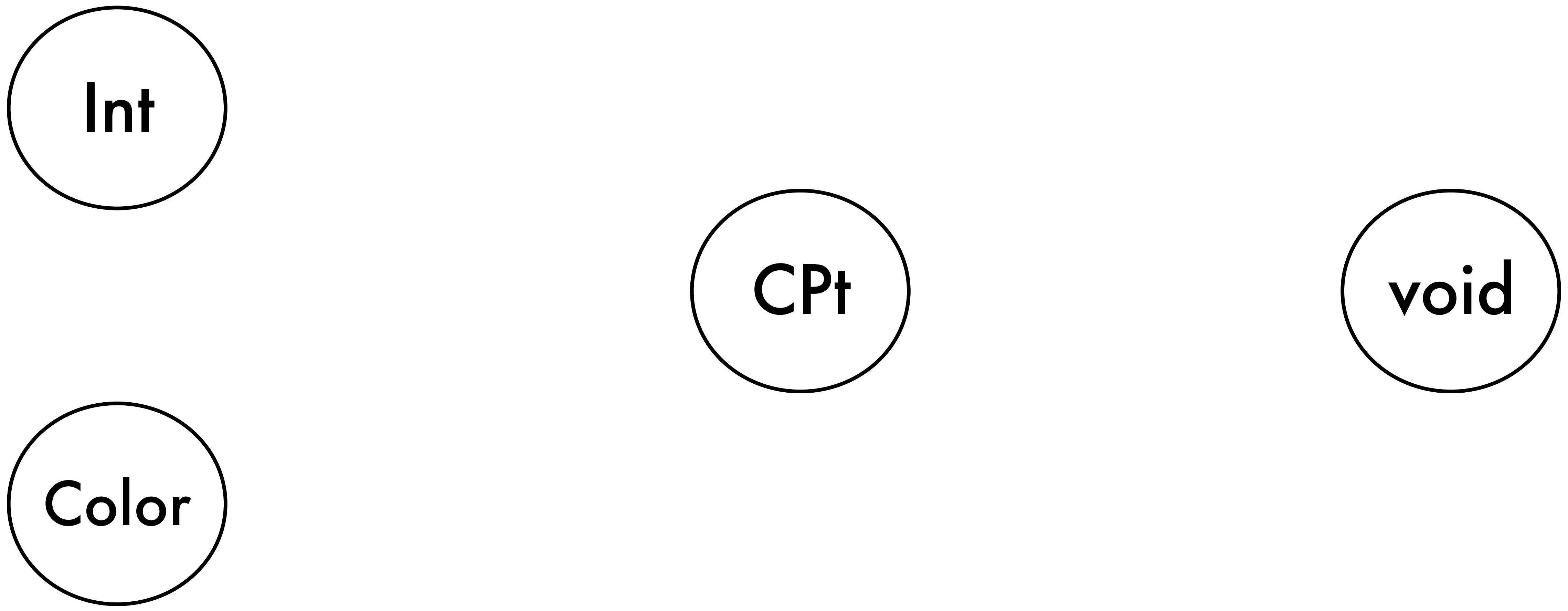


SyPet Architecture



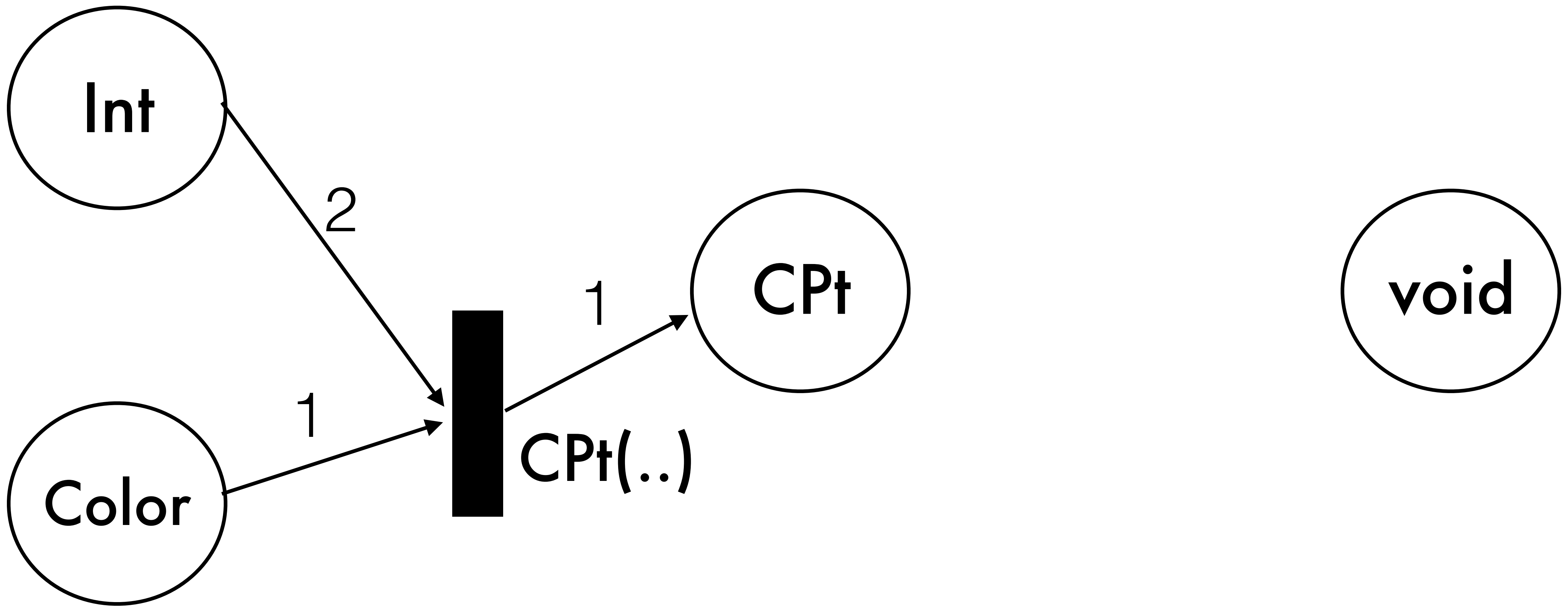
Petri Net Construction

```
class CPt {
  CPt(Int x, Int y, Color c);
  Int getX();
  void setColor(Color c);
  ...
}
```



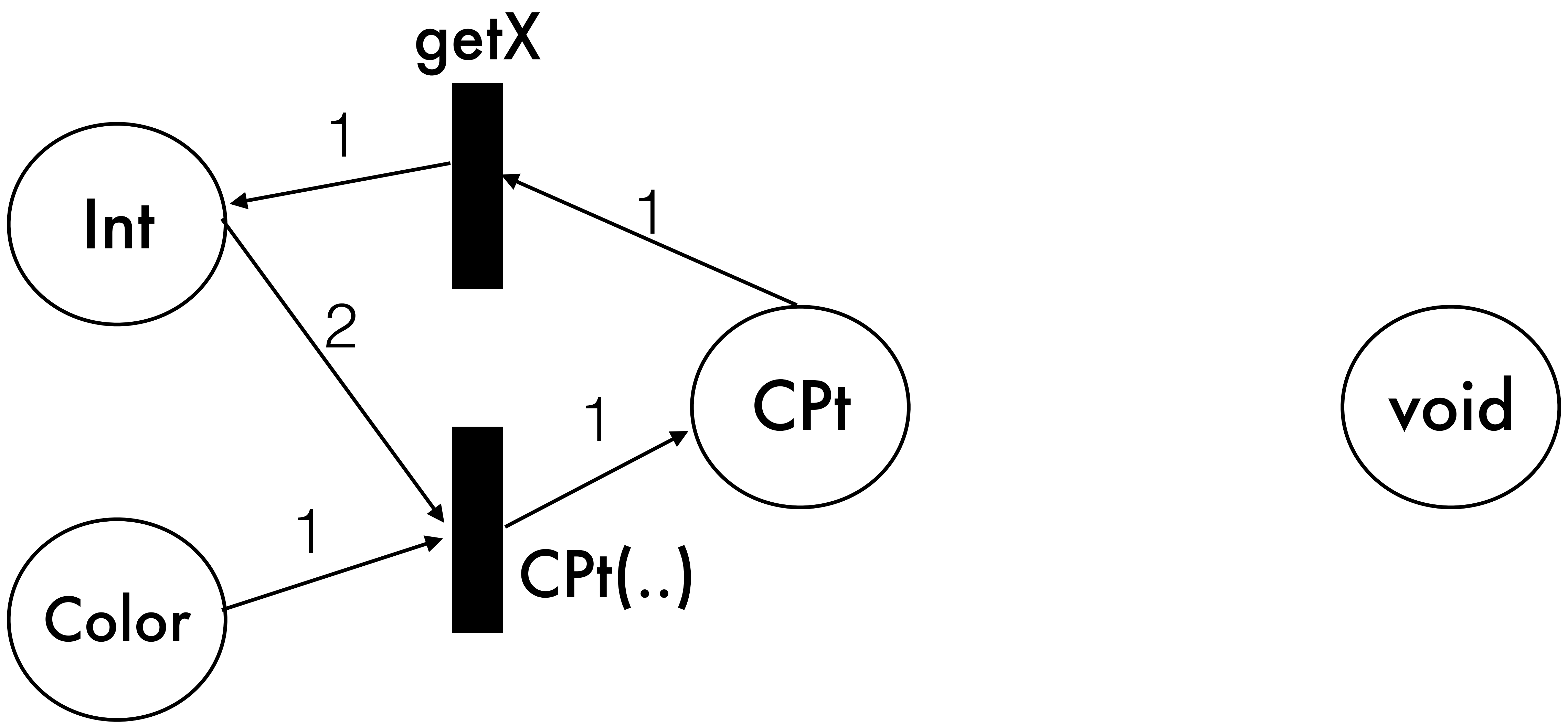
Petri Net Construction

```
class CPt {
  CPt(Int x, Int y, Color c);
  Int getX();
  void setColor(Color c);
  ...
}
```



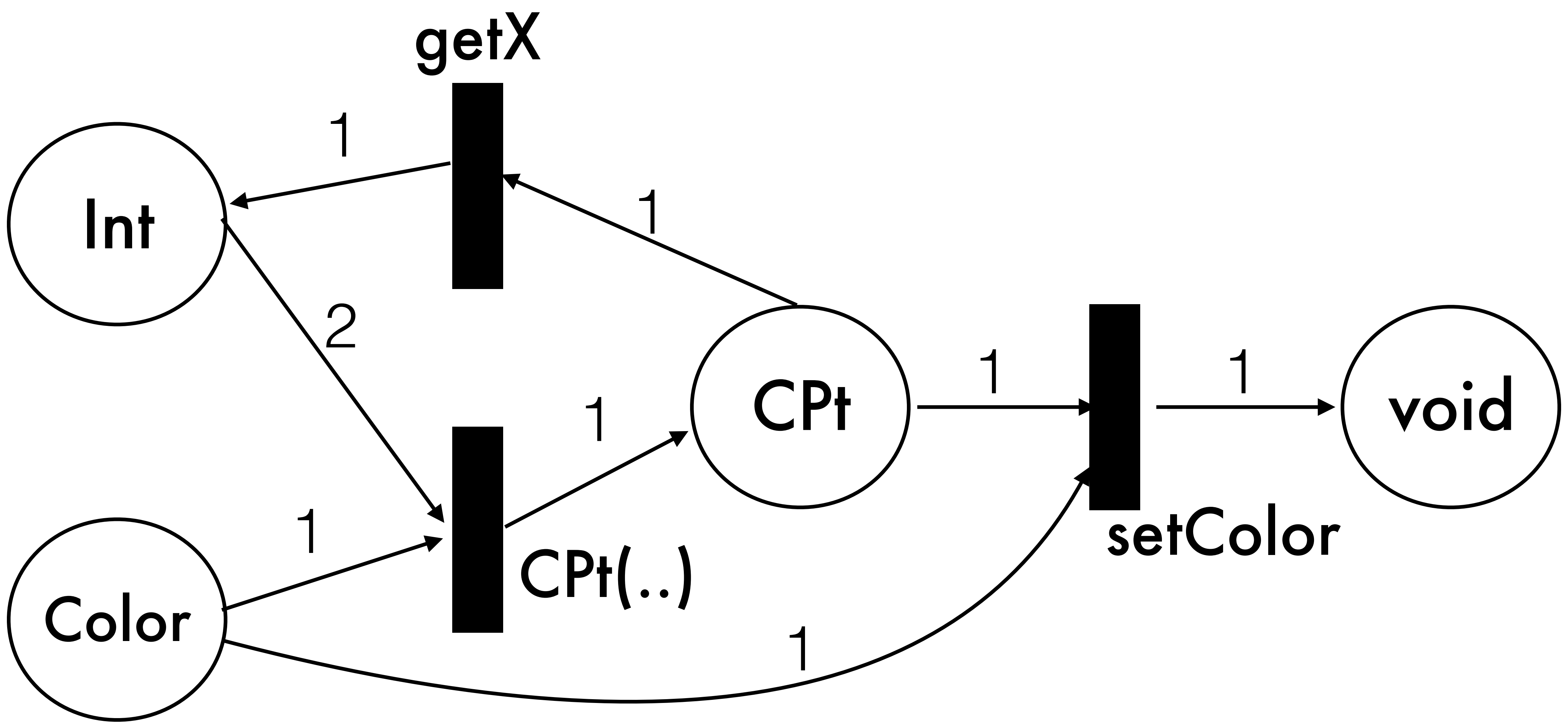
Petri Net Construction

```
class CPt {
  CPt(Int x, Int y, Color c);
  Int getX();
  void setColor(Color c);
  ...
}
```

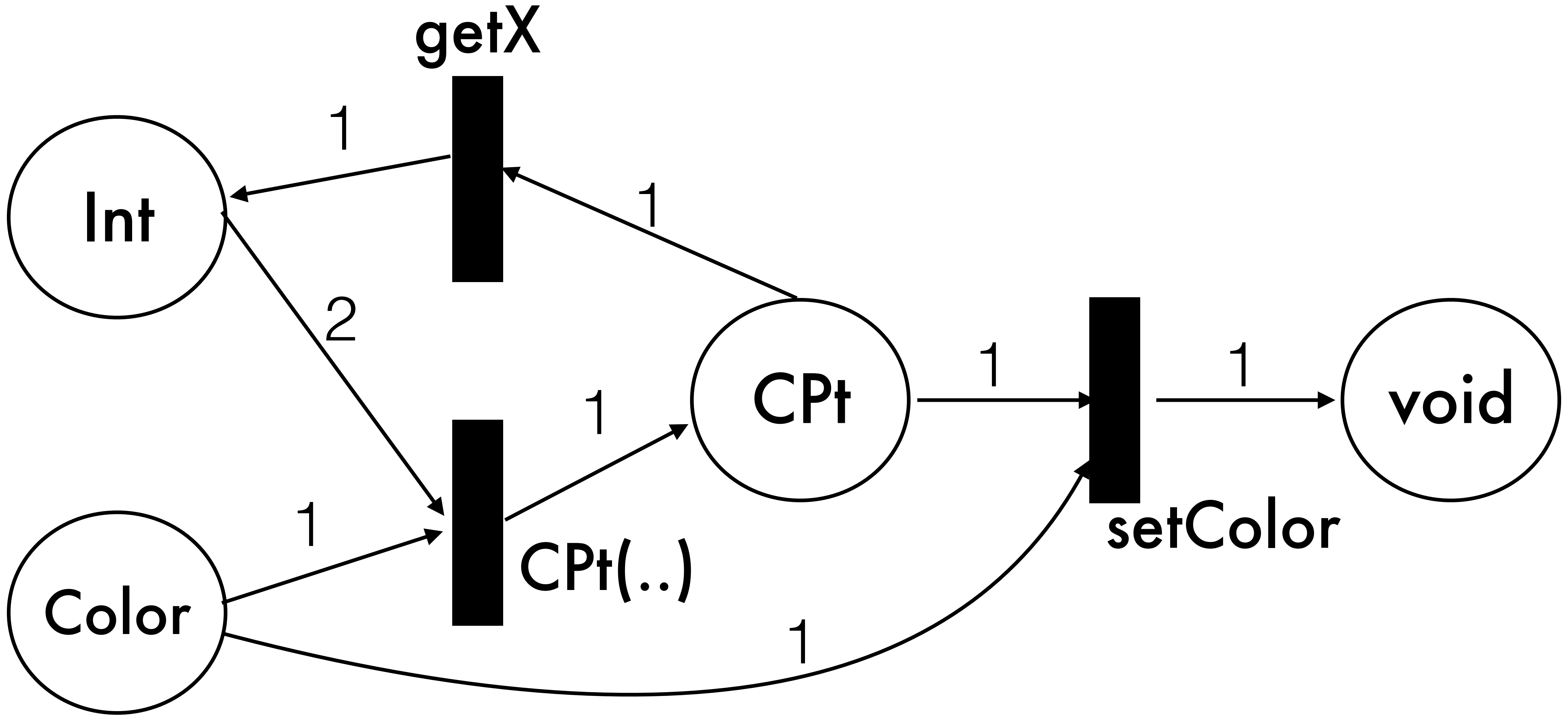


Petri Net Construction

```
class CPt {
  CPt(Int x, Int y, Color c);
  Int getX();
  void setColor(Color c);
  ...
}
```

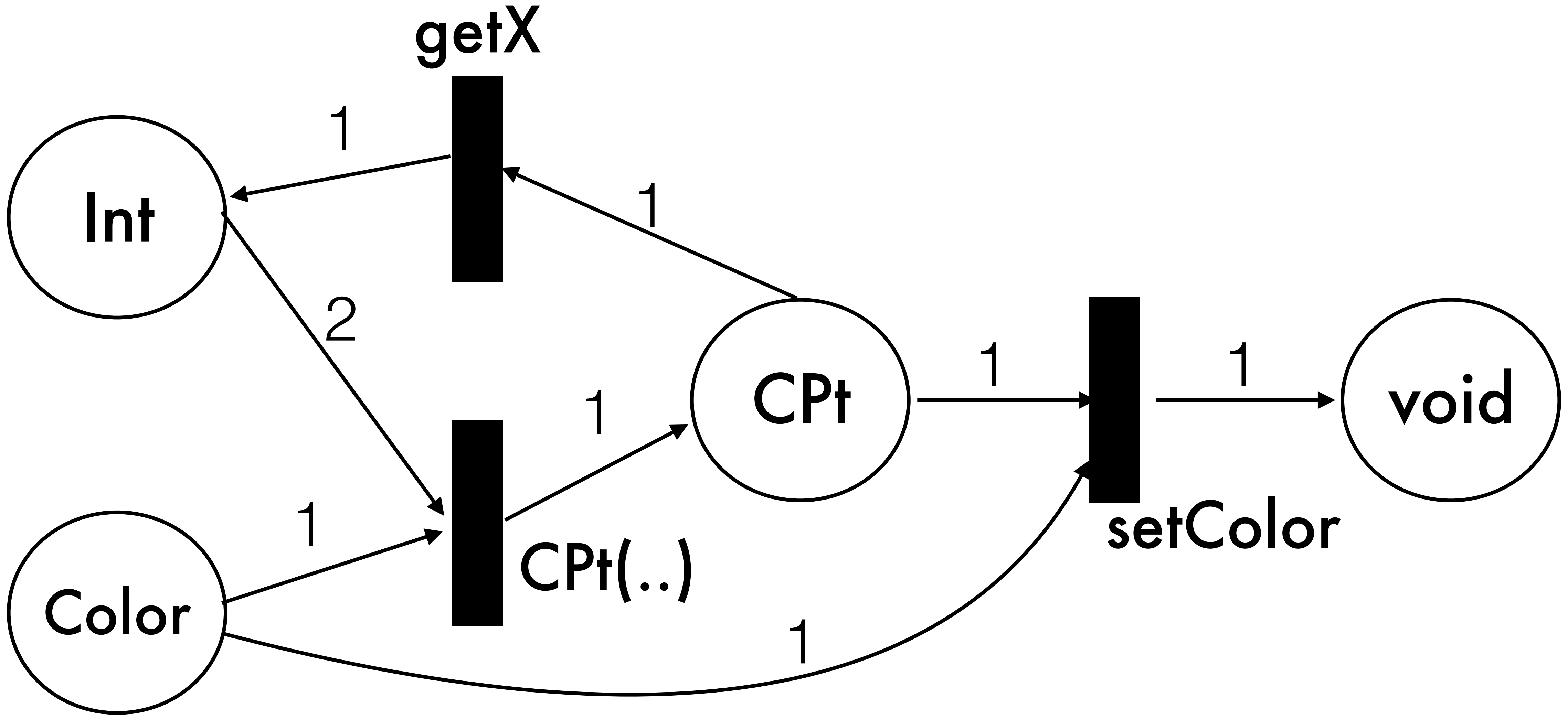


Clone Transitions



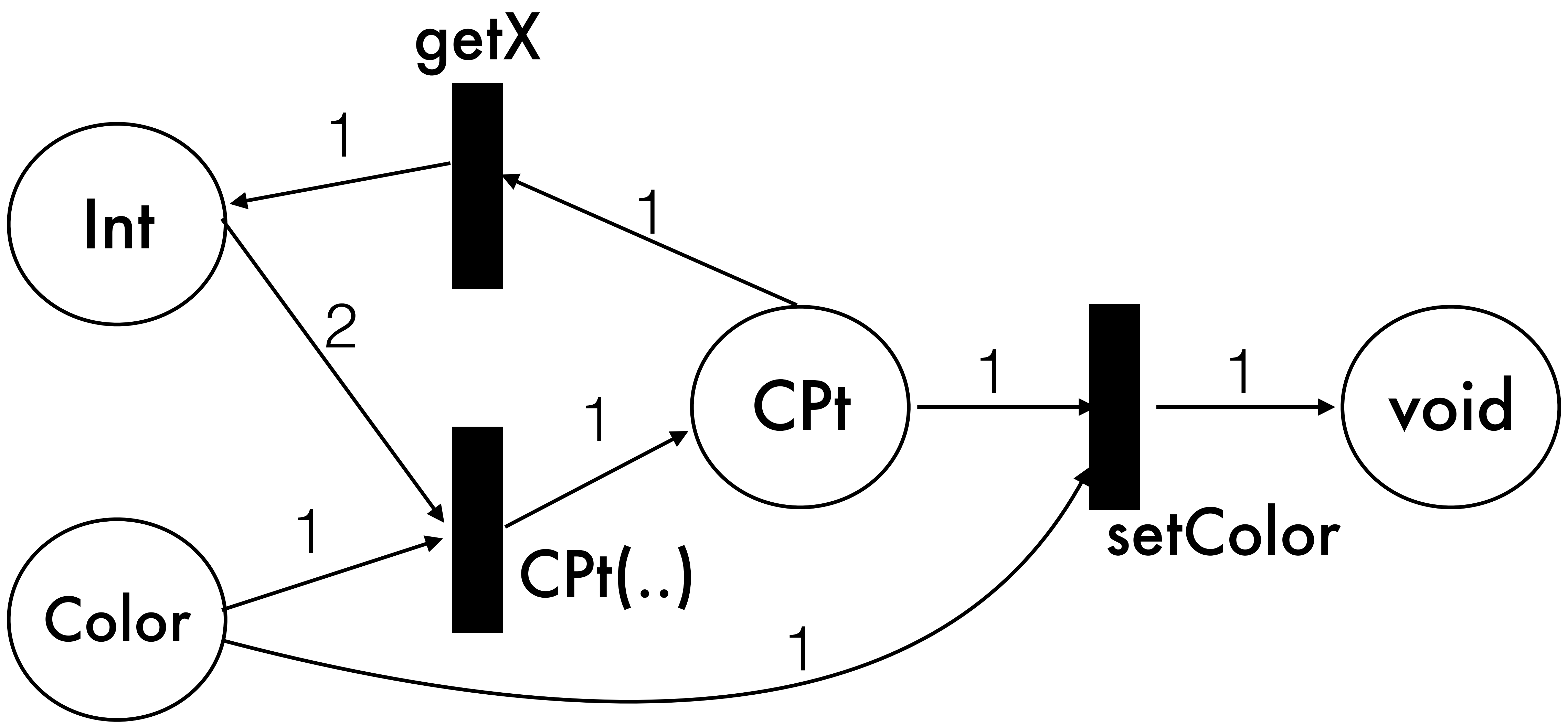
Clone Transitions

- Our construction so far views objects as“resources”– every method “consumes” and “produces” objects



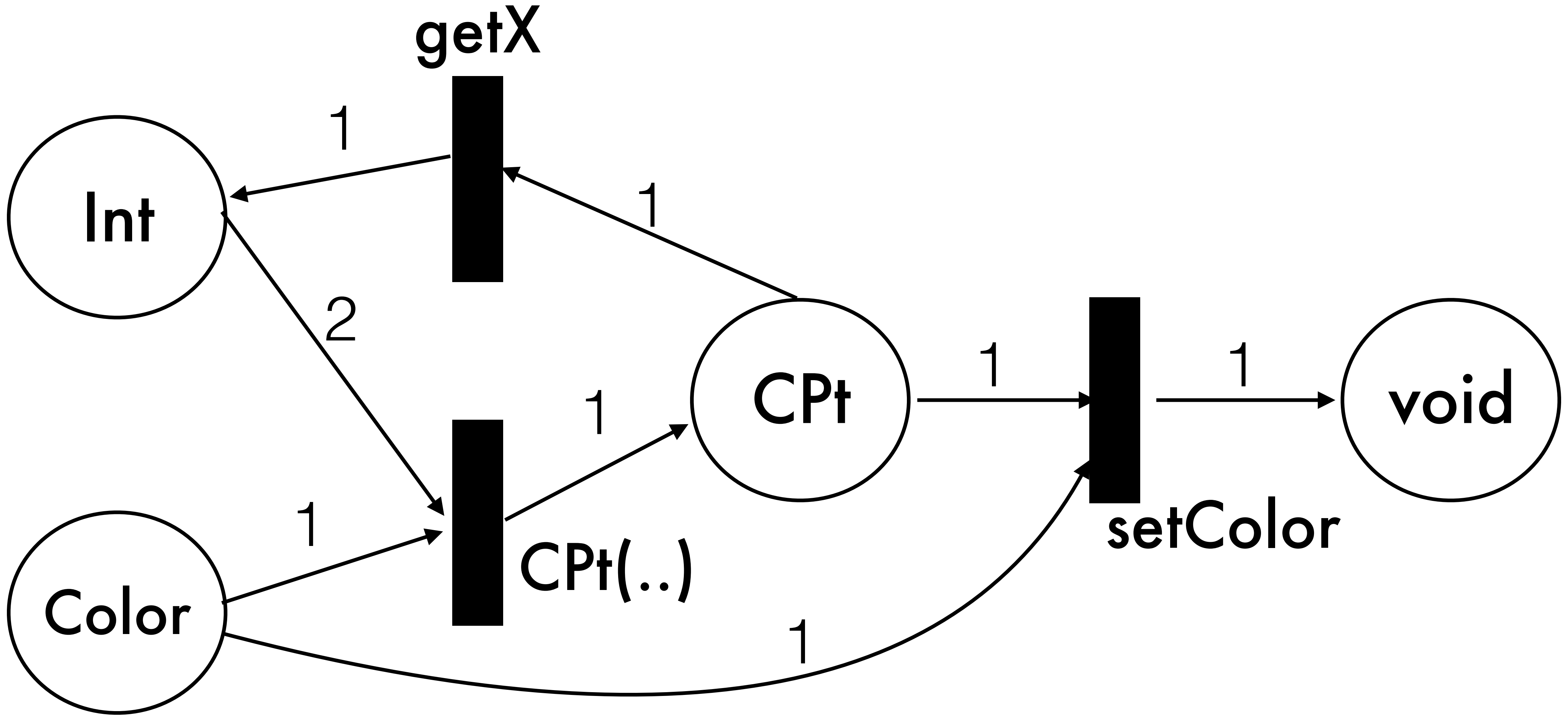
Clone Transitions

- Our construction so far views objects as“resources”– every method “consumes” and “produces” objects
- But in conventional languages, we can reuse objects!



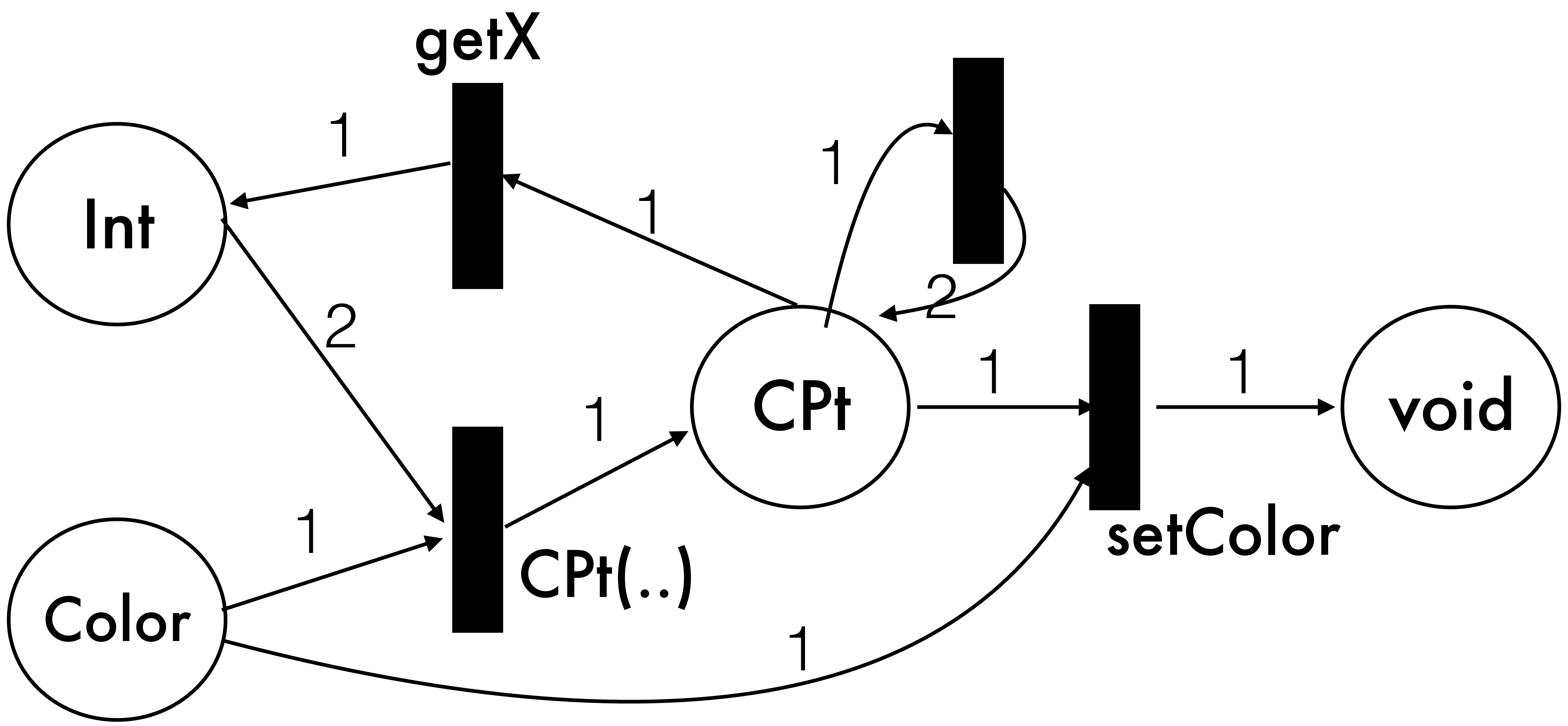
Clone Transitions

- Our construction so far views objects as“resources”– every method “consumes” and “produces” objects
- But in conventional languages, we can reuse objects!
- Therefore, augment Petri net model with **clone transitions**



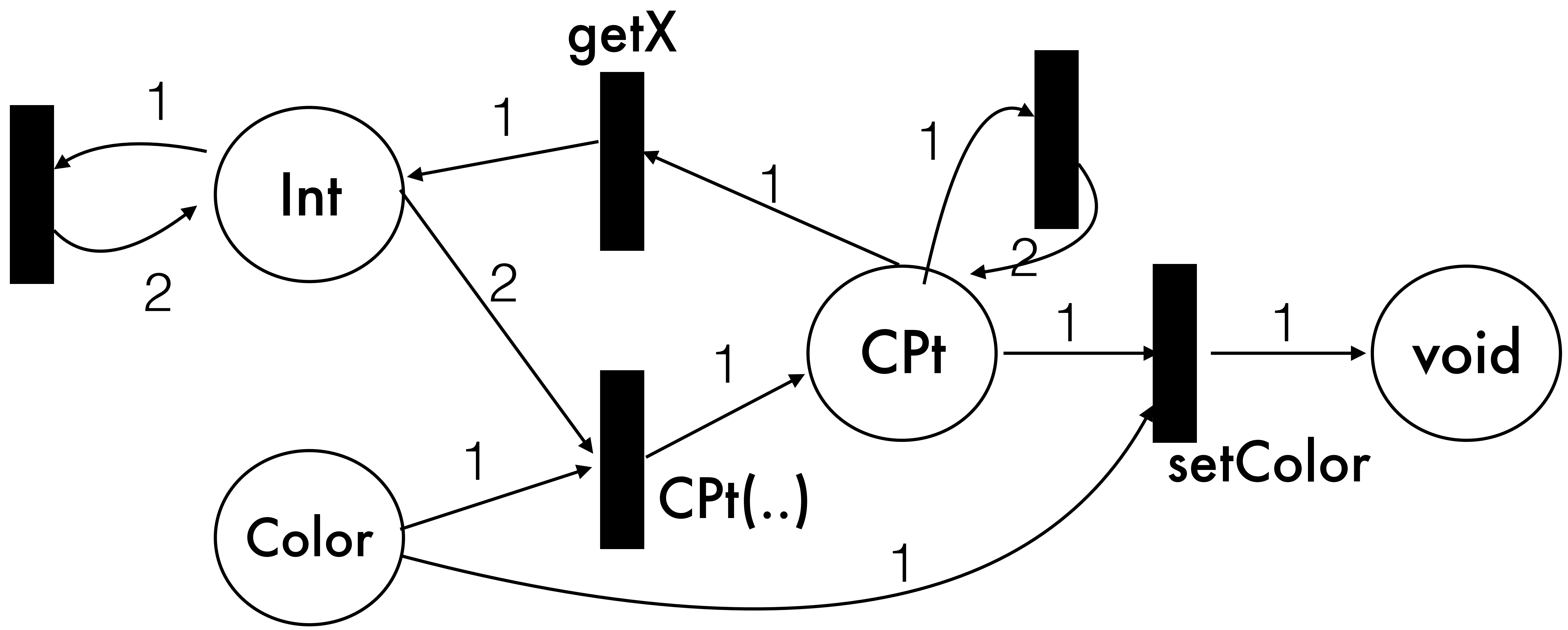
Clone Transitions

- Our construction so far views objects as"resources"– every method "consumes" and "produces" objects
- But in conventional languages, we can reuse objects!
- Therefore, augment Petri net model with **clone transitions**



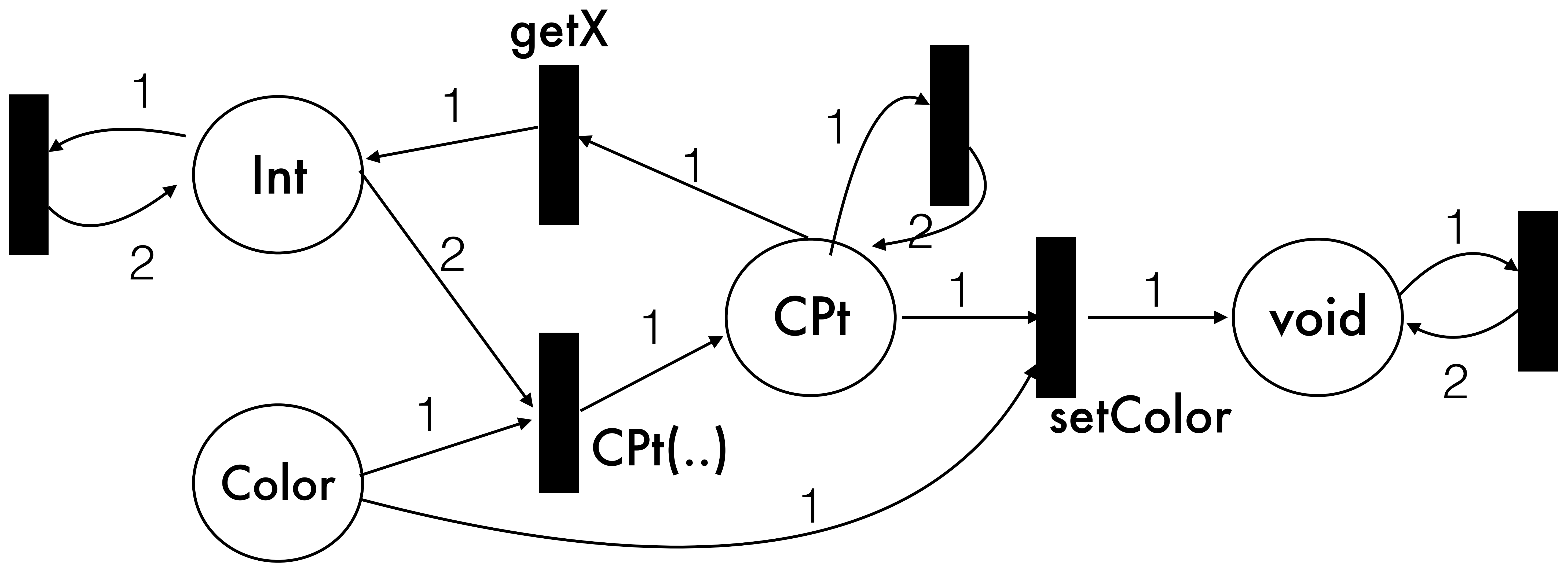
Clone Transitions

- Our construction so far views objects as“resources”– every method “consumes” and “produces” objects
- But in conventional languages, we can reuse objects!
- Therefore, augment Petri net model with **clone transitions**



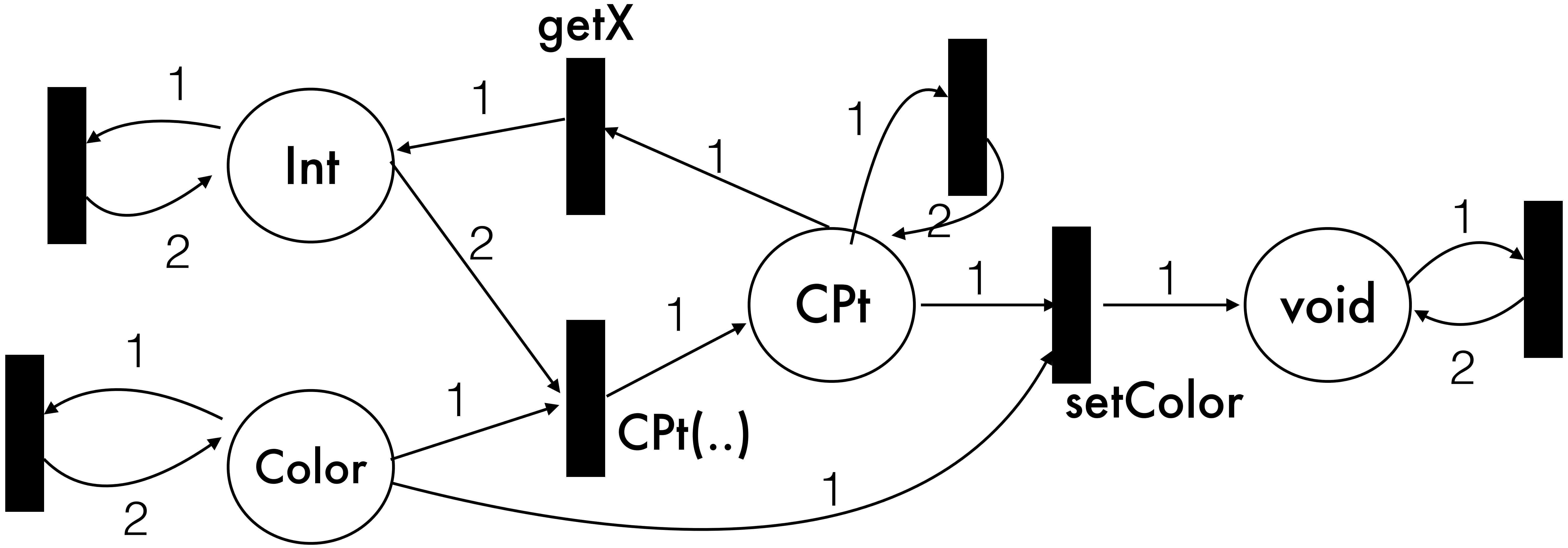
Clone Transitions

- Our construction so far views objects as“resources”– every method “consumes” and “produces” objects
- But in conventional languages, we can reuse objects!
- Therefore, augment Petri net model with **clone transitions**

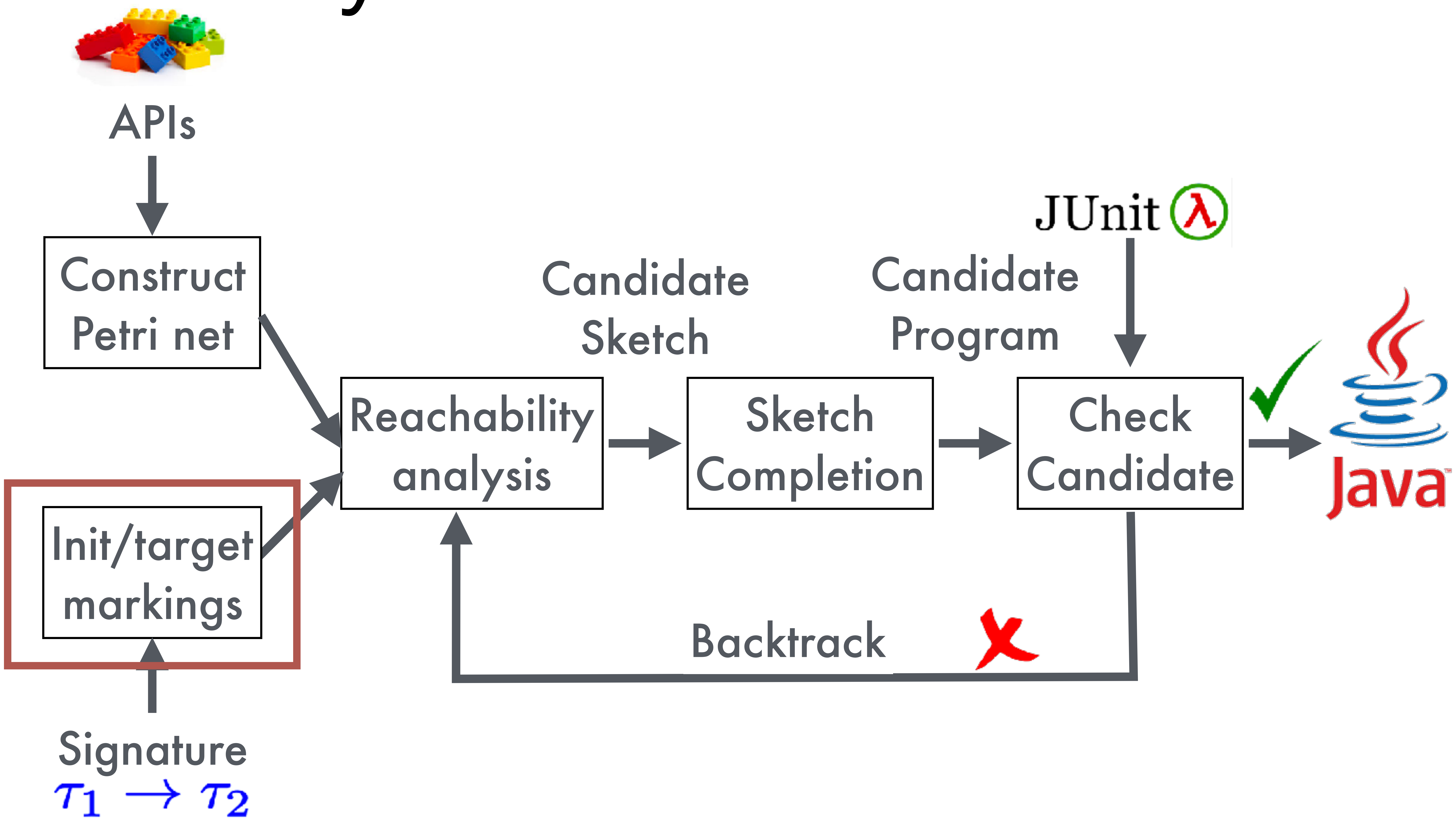


Clone Transitions

- Our construction so far views objects as“resources”– every method “consumes” and “produces” objects
- But in conventional languages, we can reuse objects!
- Therefore, augment Petri net model with **clone transitions**

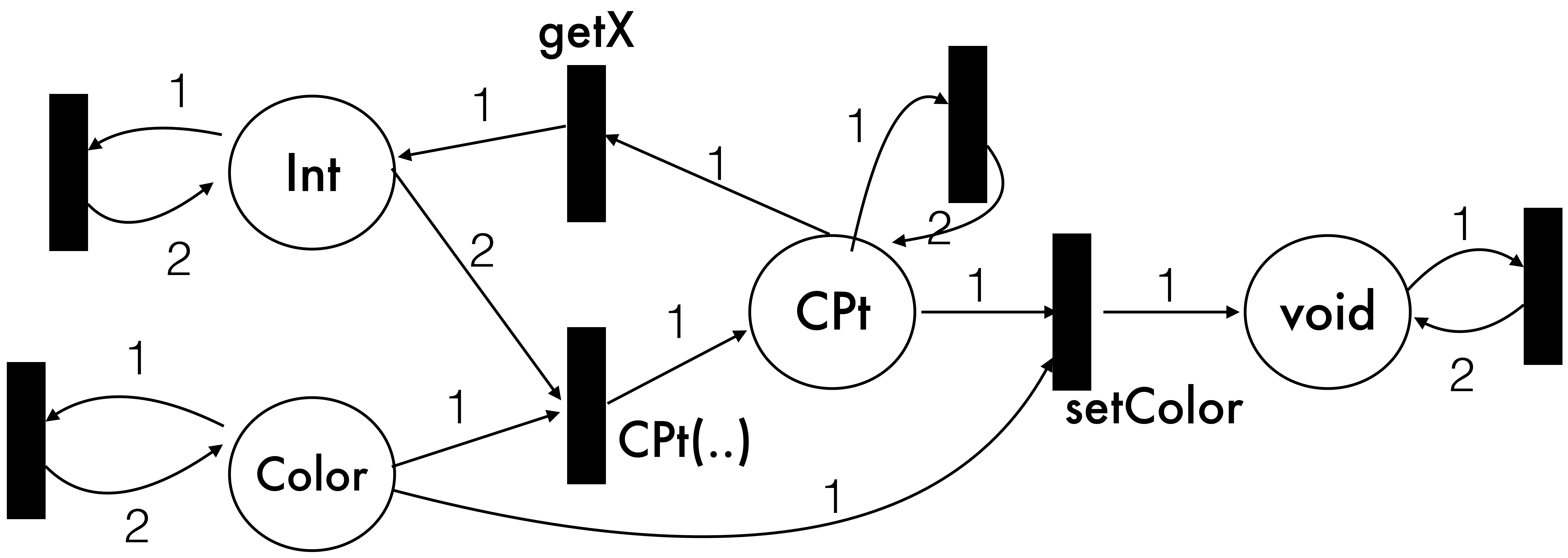


SyPet Architecture



Initial and Target Markings

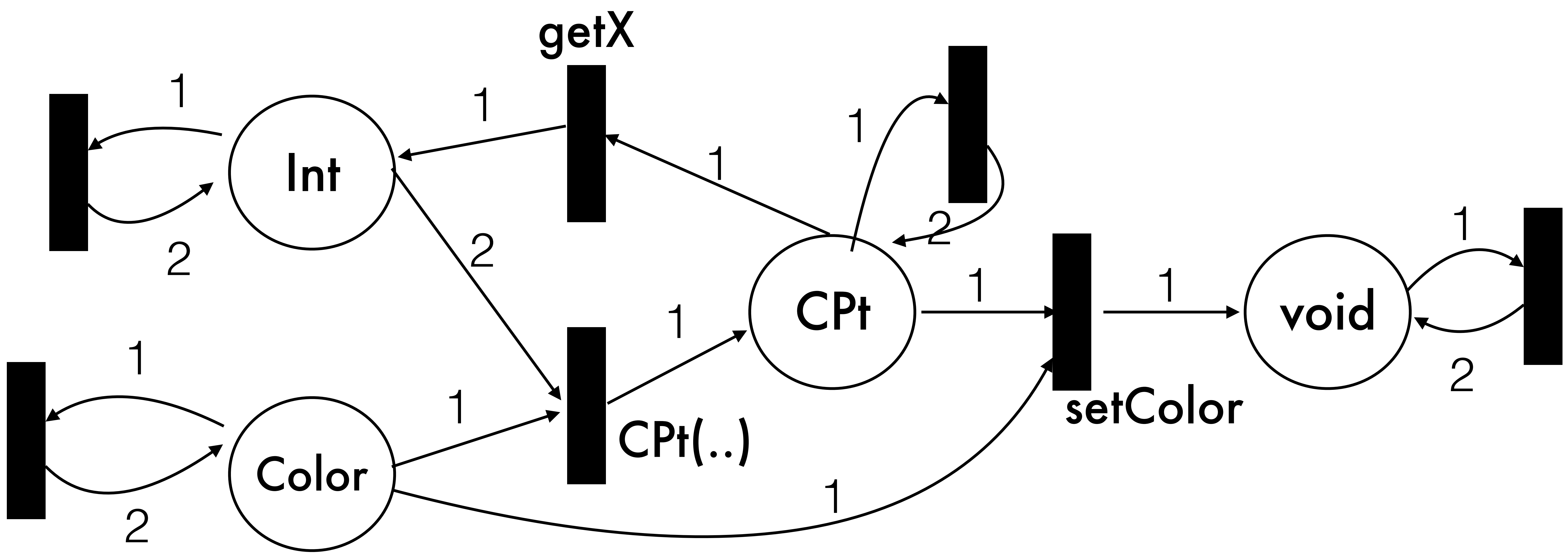
Use signature to determine initial and target markings of Petri net



Initial and Target Markings

Use signature to determine initial and target markings of Petri net

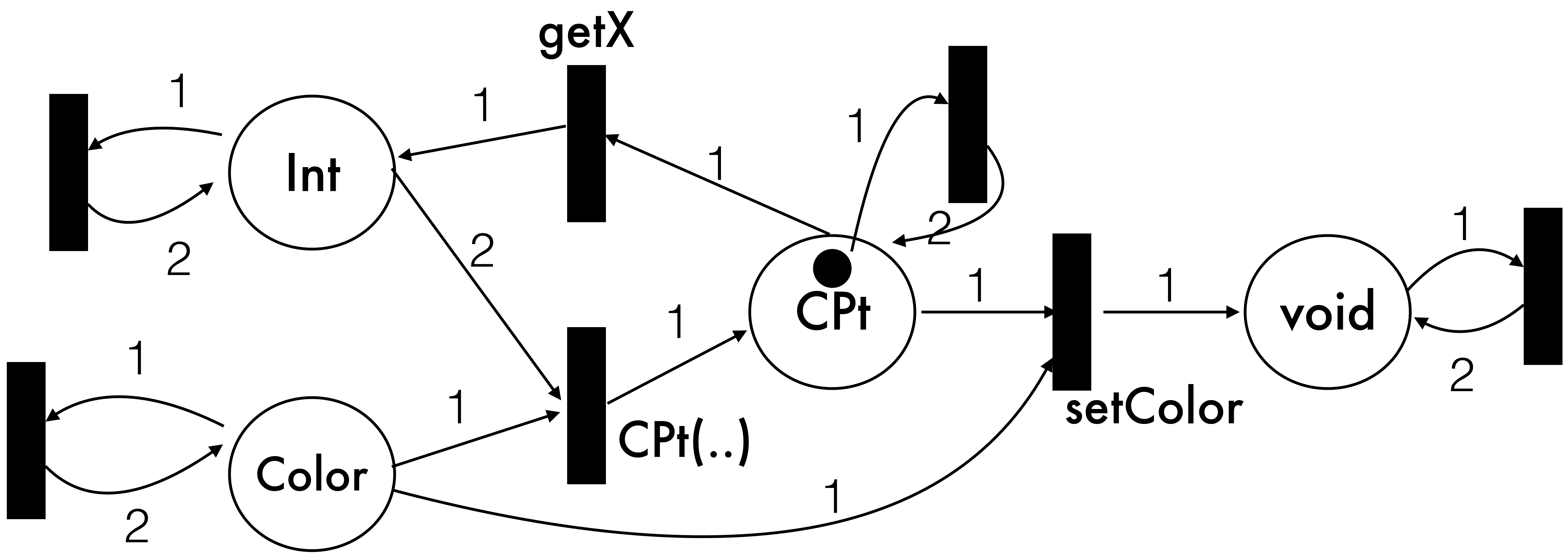
```
CPt shift (CPt p, Int shiftX, Int shiftY)
```



Initial and Target Markings

Use signature to determine initial and target markings of Petri net

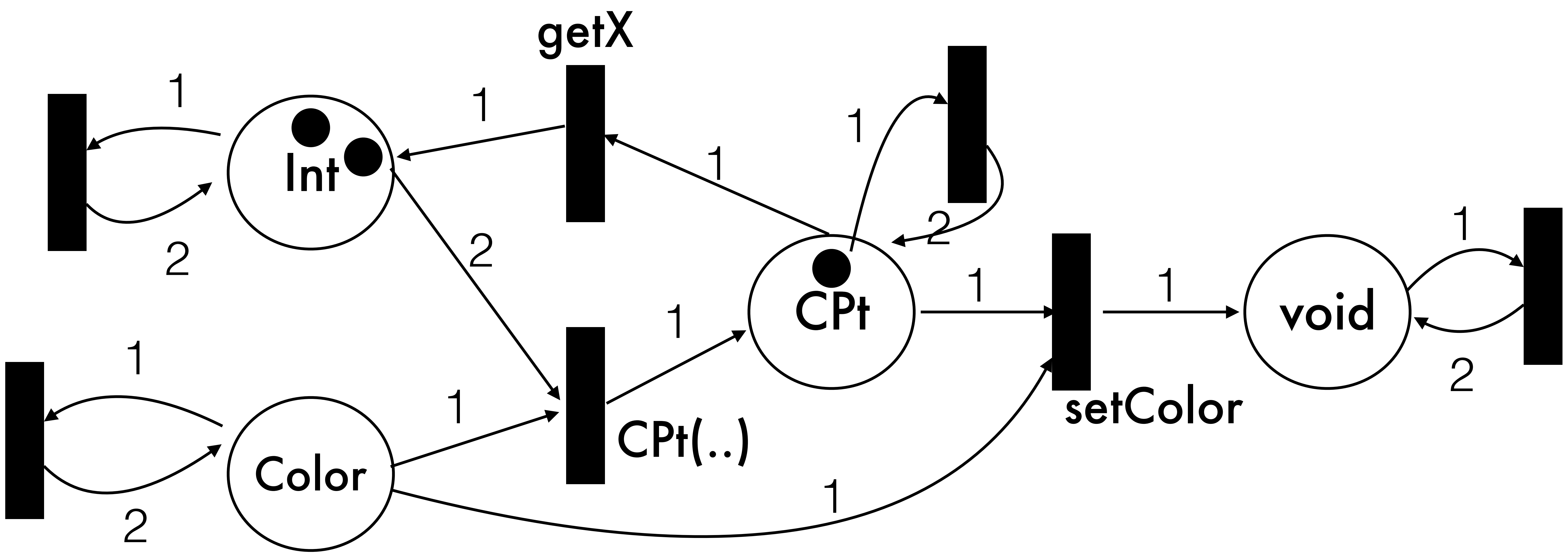
```
CPt shift (CPt p, Int shiftX, Int shiftY)
```



Initial and Target Markings

Use signature to determine initial and target markings of Petri net

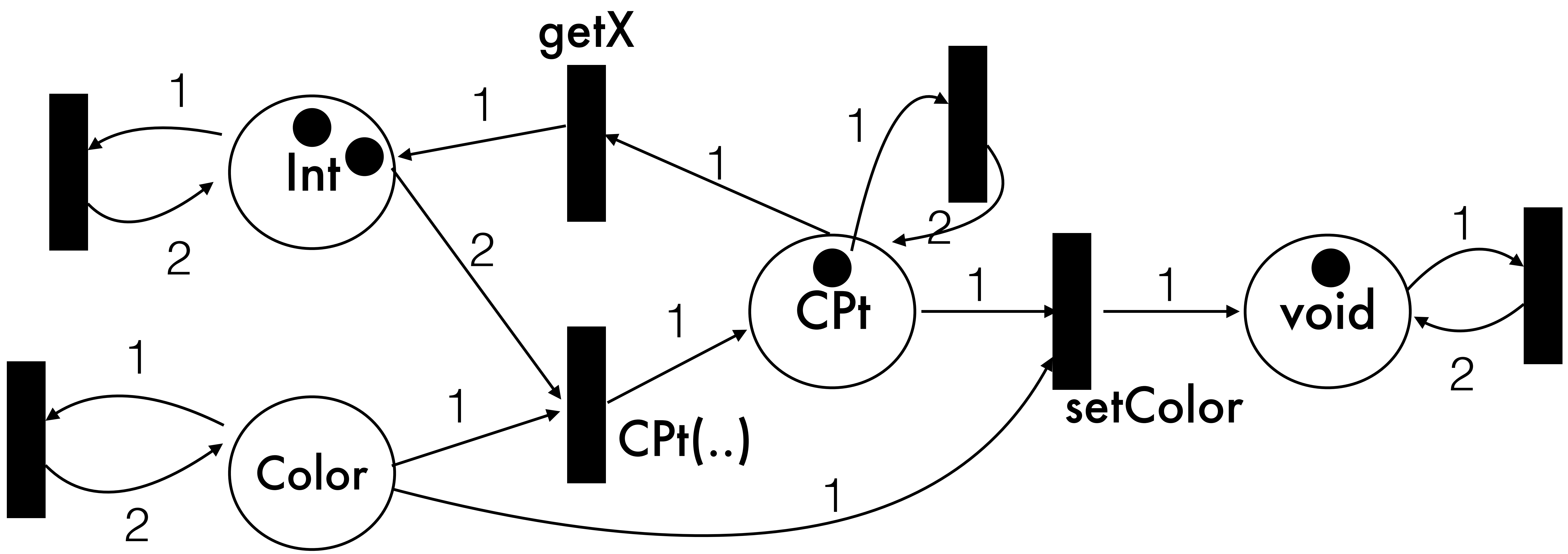
```
CPt shift (CPt p, Int shiftX, Int shiftY)
```



Initial and Target Markings

Use signature to determine initial and target markings of Petri net

```
CPt shift (CPt p, Int shiftX, Int shiftY)
```

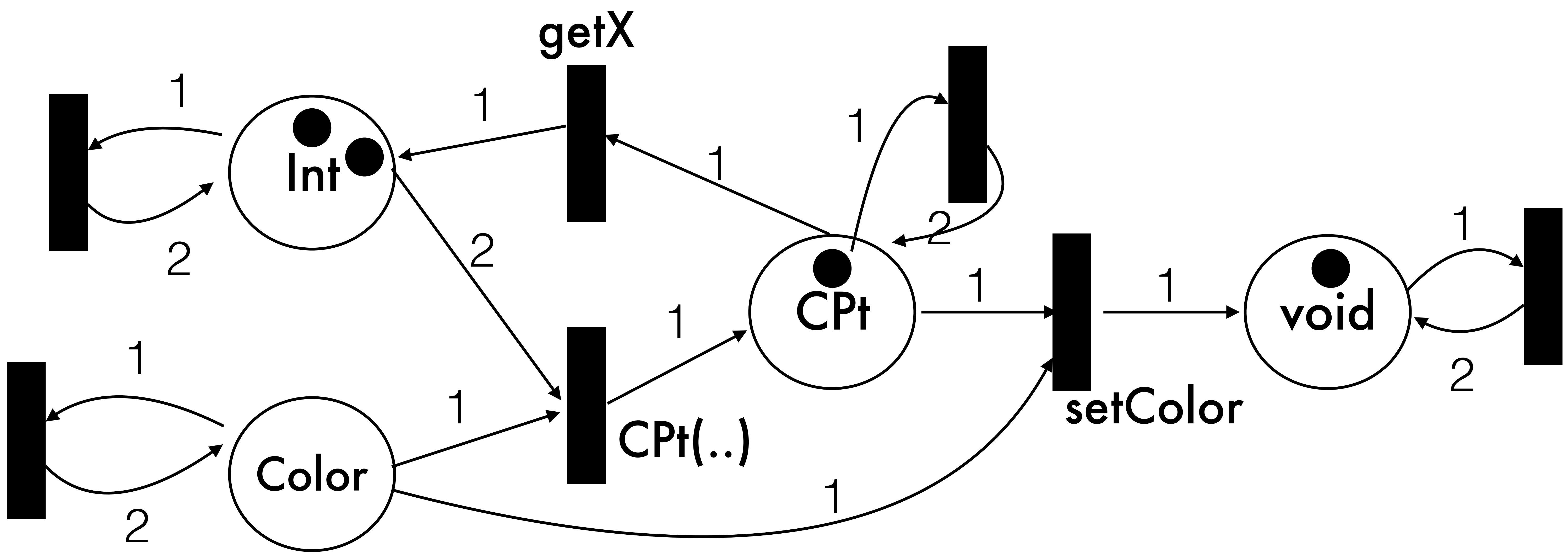


Initial and Target Markings

Use signature to determine initial and target markings of Petri net

```
CPt shift (CPt p, Int shiftX, Int shiftY)
```

Target marking:



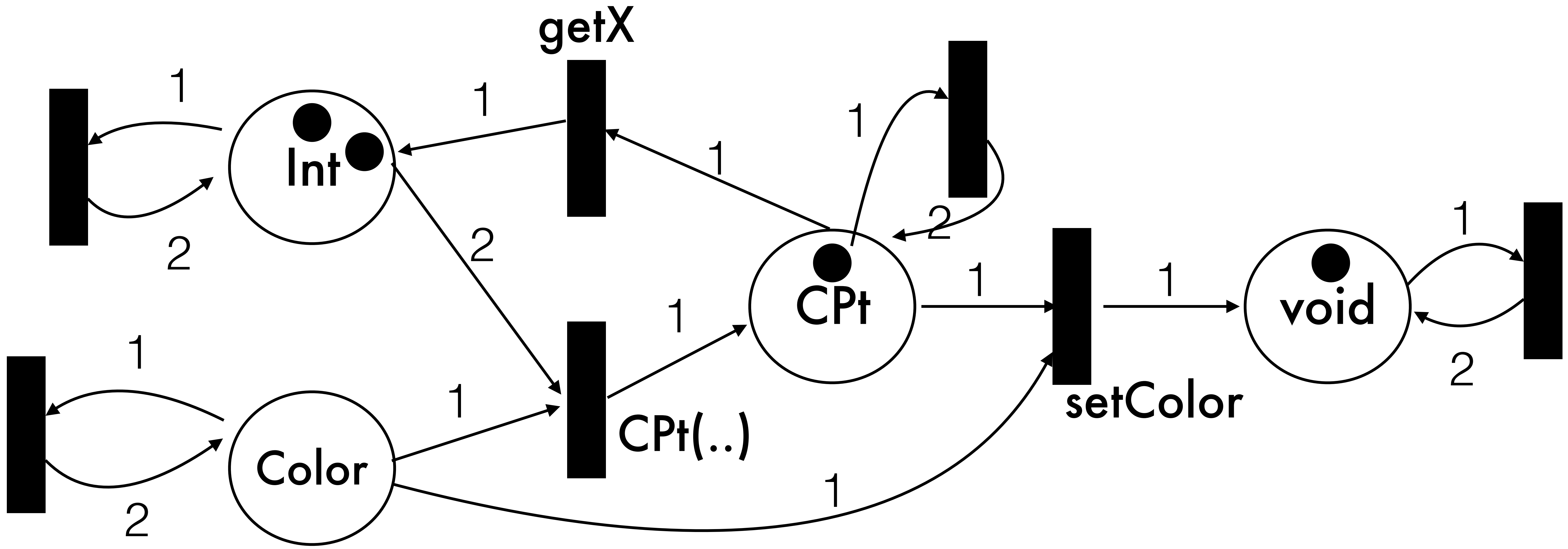
Initial and Target Markings

Use signature to determine initial and target markings of Petri net

```
CPt shift (CPt p, Int shiftX, Int shiftY)
```

Target marking:

Cpt = 1



Initial and Target Markings

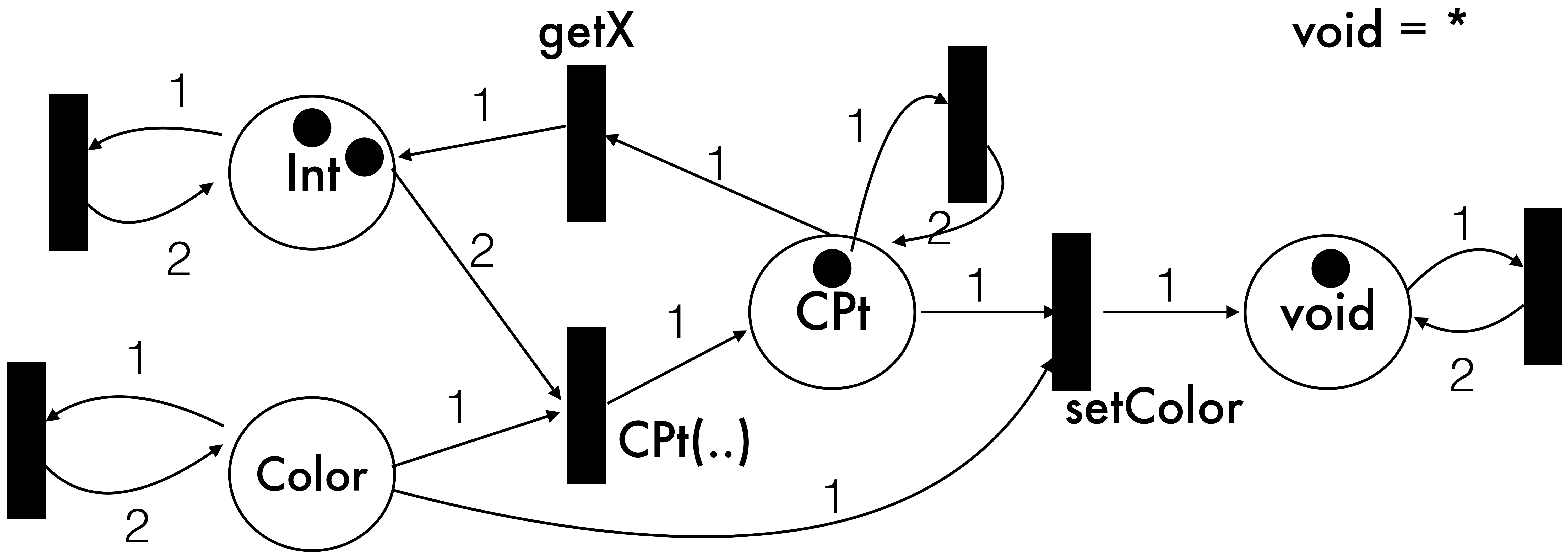
Use signature to determine initial and target markings of Petri net

```
CPt shift (CPt p, Int shiftX, Int shiftY)
```

Target marking:

Cpt = 1

void = *



Initial and Target Markings

Use signature to determine initial and target markings of Petri net

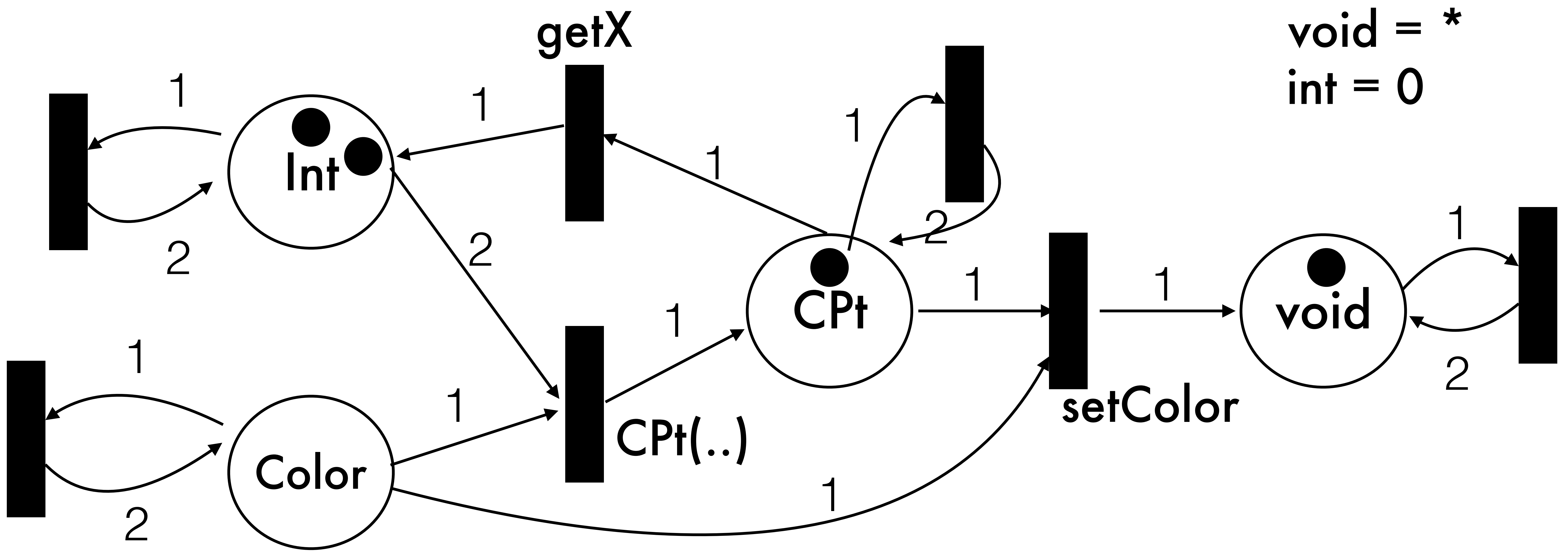
```
CPt shift (CPt p, Int shiftX, Int shiftY)
```

Target marking:

Cpt = 1

void = *

int = 0



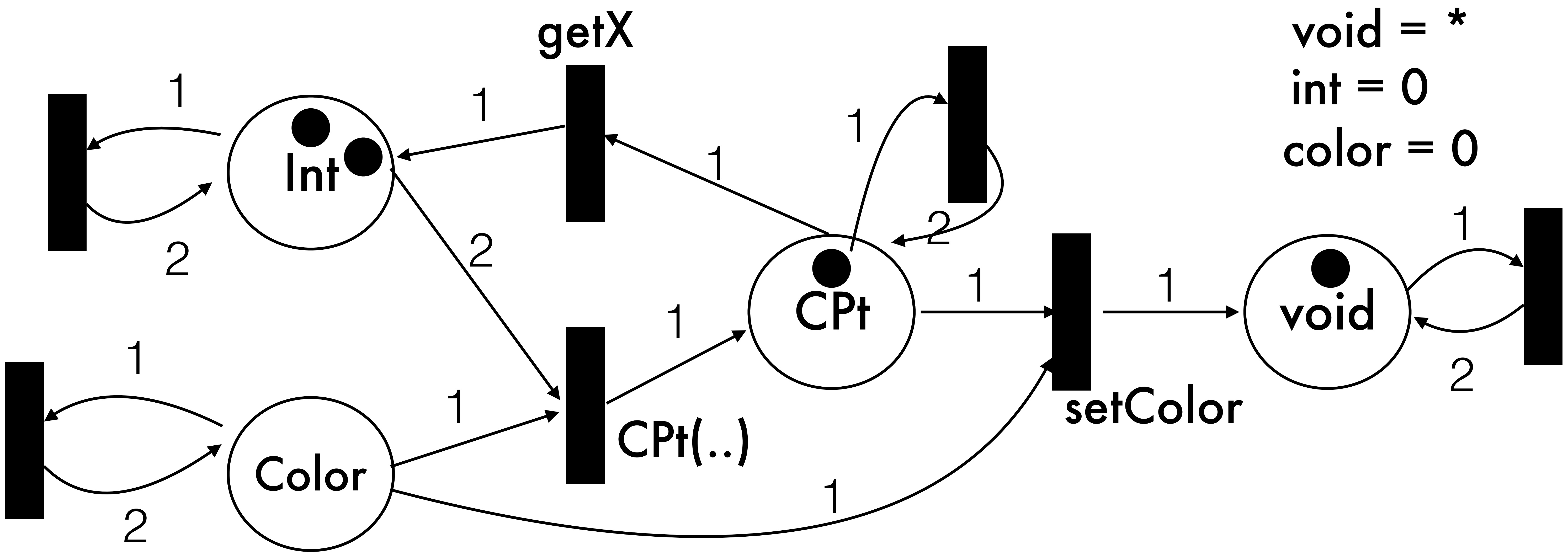
Initial and Target Markings

Use signature to determine initial and target markings of Petri net

```
CPt shift (CPt p, Int shiftX, Int shiftY)
```

Target marking:

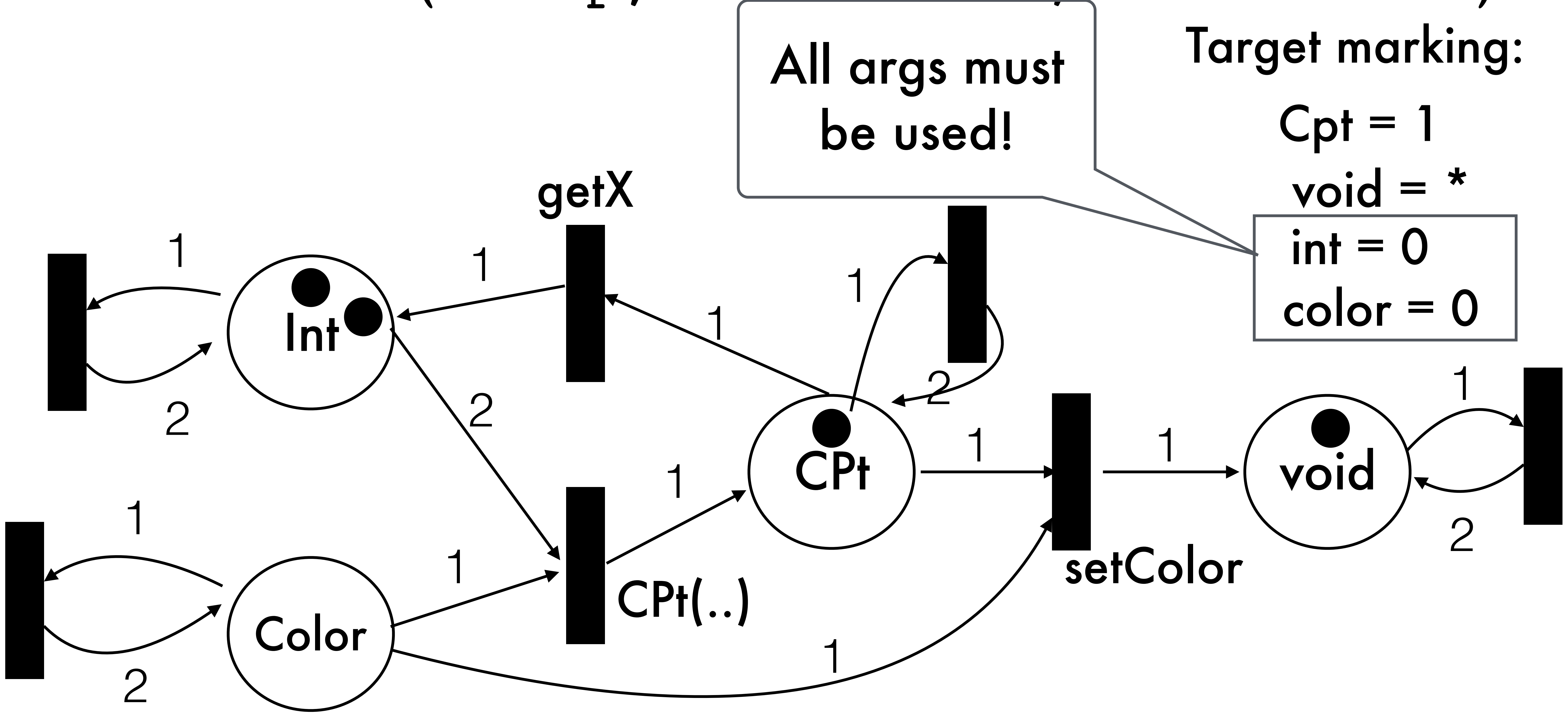
Cpt = 1
void = *
int = 0
color = 0



Initial and Target Markings

Use signature to determine initial and target markings of Petri net

```
CPt shift (CPt p, Int shiftX, Int shiftY)
```



Building a Petri Net

```
class Point {
  Point();
  int getX();
  int getY();
  void setX(int);
  void setY(int);
}
```

```
class MyPoint {
  MyPoint(int x, int y);
  int getX();
  int getY();
}
```

Building a Petri Net

```
class Point {
  Point();
  int getX();
  int getY();
  void setX(int);
  void setY(int);
}
```

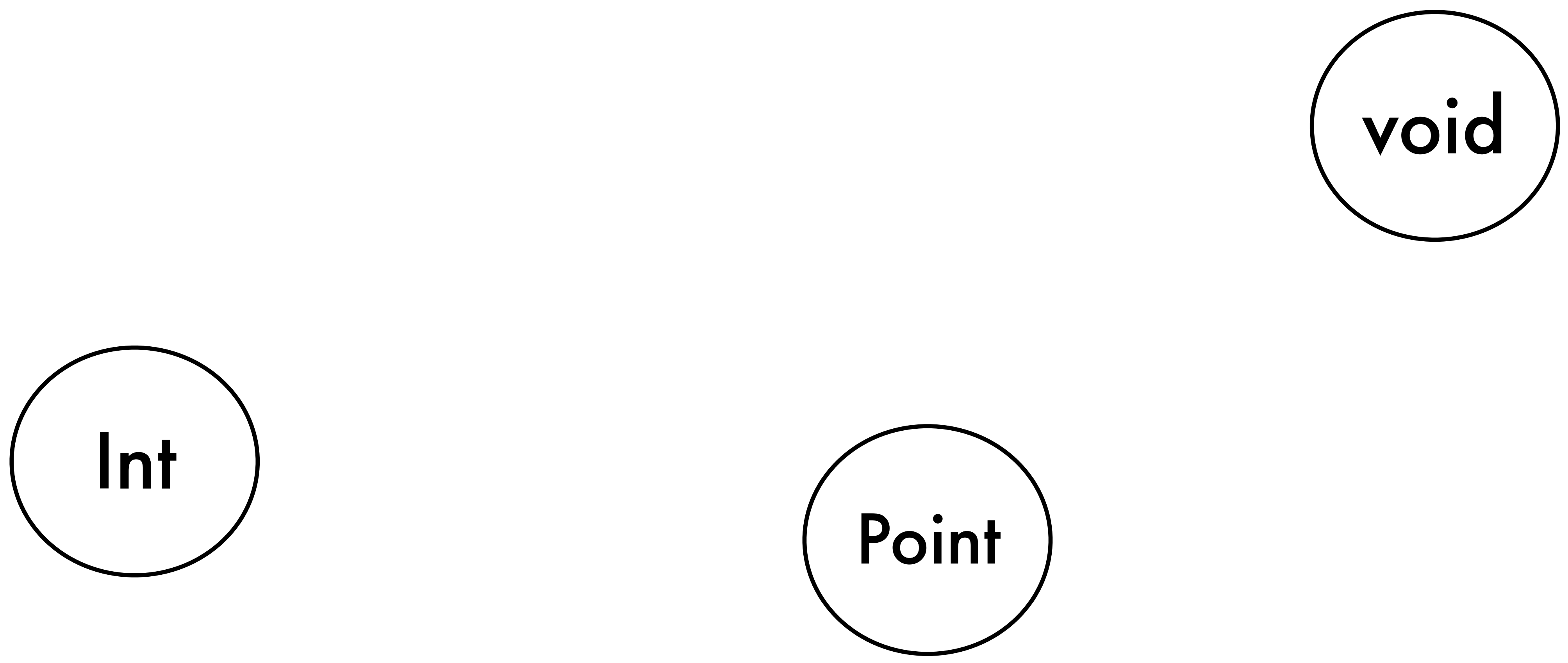
```
class MyPoint {
  MyPoint(int x, int y);
  int getX();
  int getY();
}
```

- What are the **places** (i.e., types)?
- What are the **transitions** (i.e., methods)?

Building a Petri Net

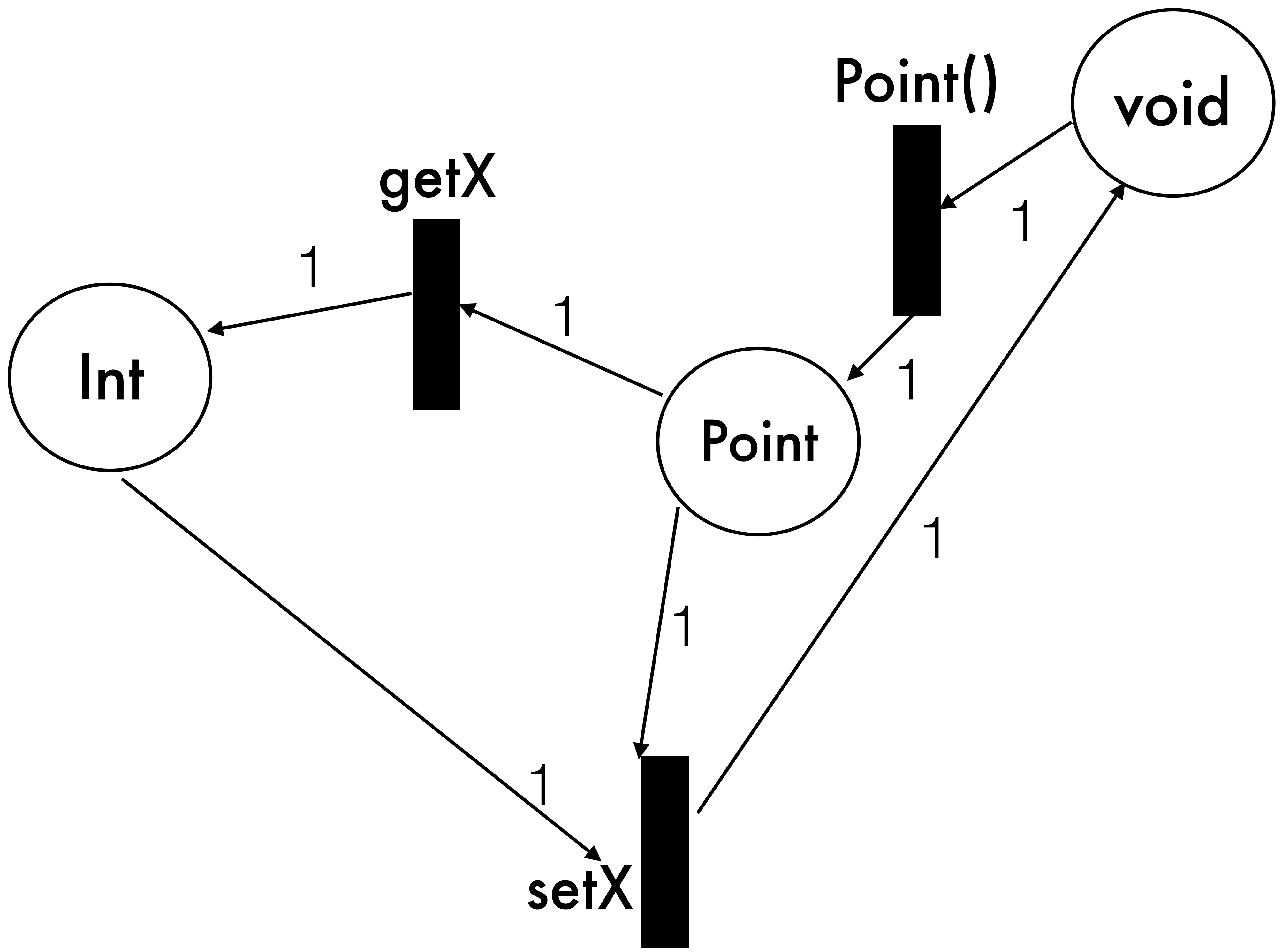
```
class Point {
  Point();
  int getX();
  int getY();
  void setX(int);
  void setY(int);
}
```

Building a Petri Net



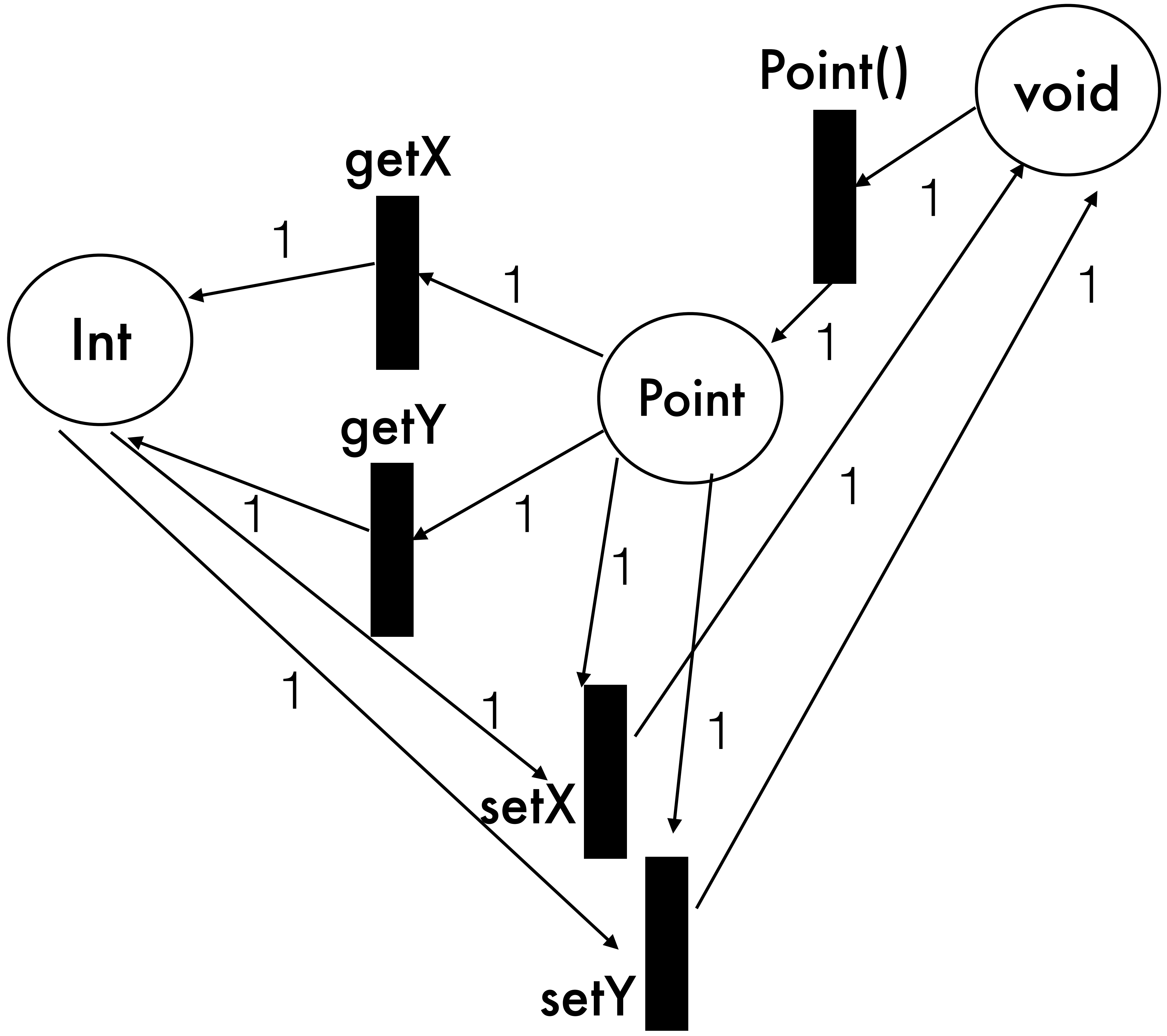

```
class Point {
  Point();
  int getX();
  int getY();
  void setX(int);
  void setY(int);
 }
```

Building a Petri Net



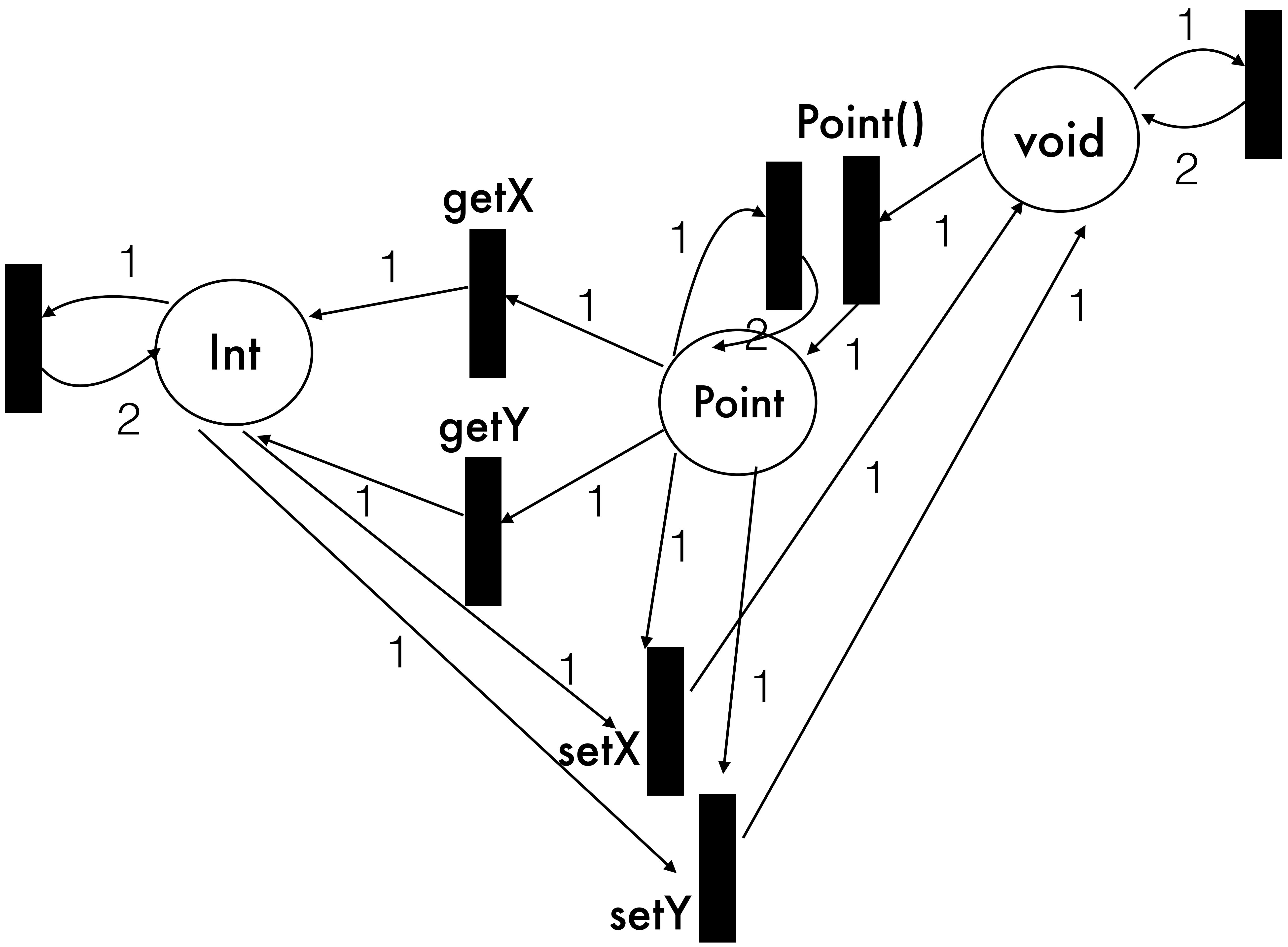

```
class Point {
  Point();
  int getX();
  int getY();
  void setX(int);
  void setY(int);
 }
```

Building a Petri Net




```
class Point {
  Point();
  int getX();
  int getY();
  void setX(int);
  void setY(int);
}
```

Building a Petri Net

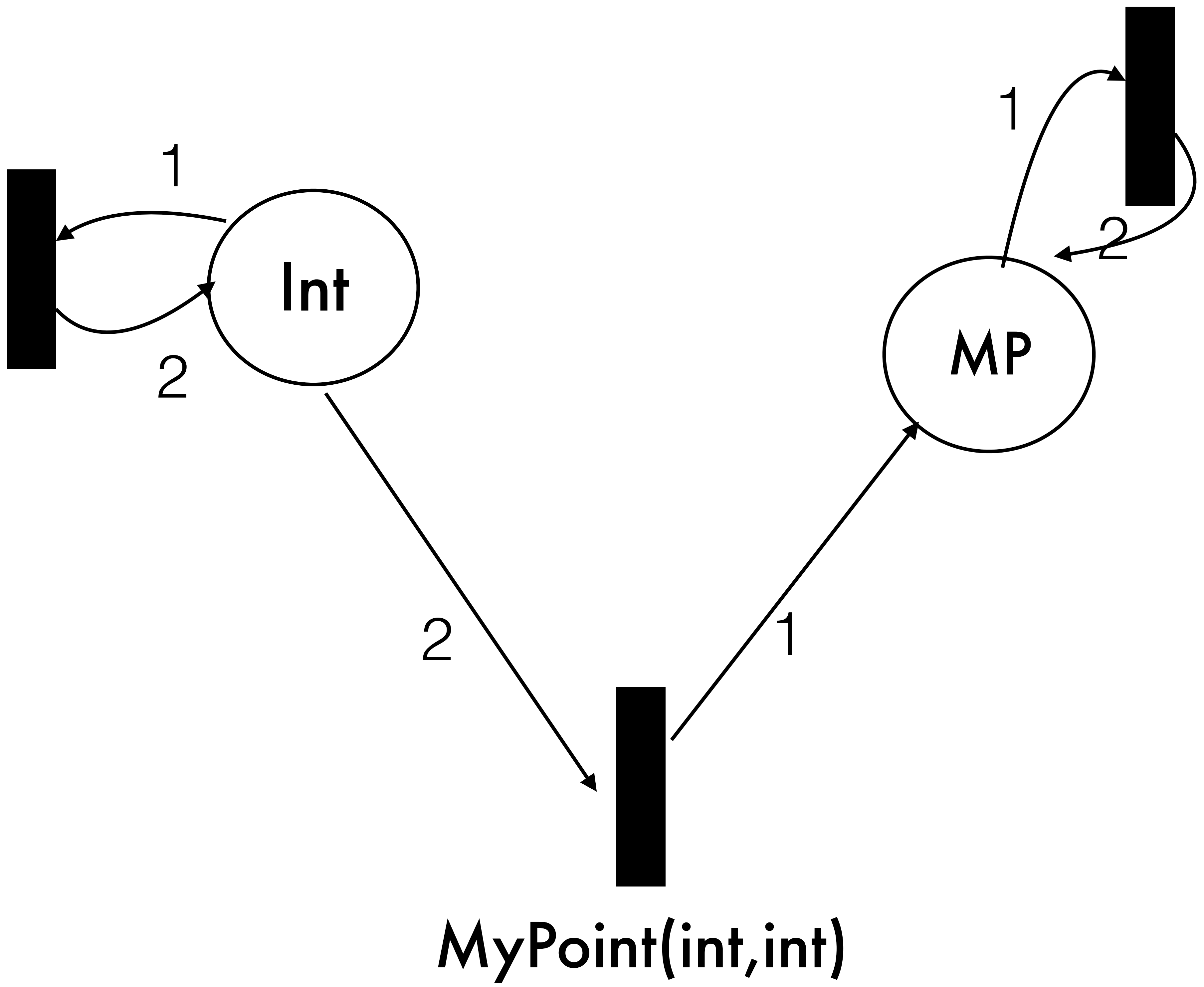



```
class Point {
  Point();
  int getX();
  int getY();
  void setX(int);
  void setY(int);
}
```

Building a Petri Net

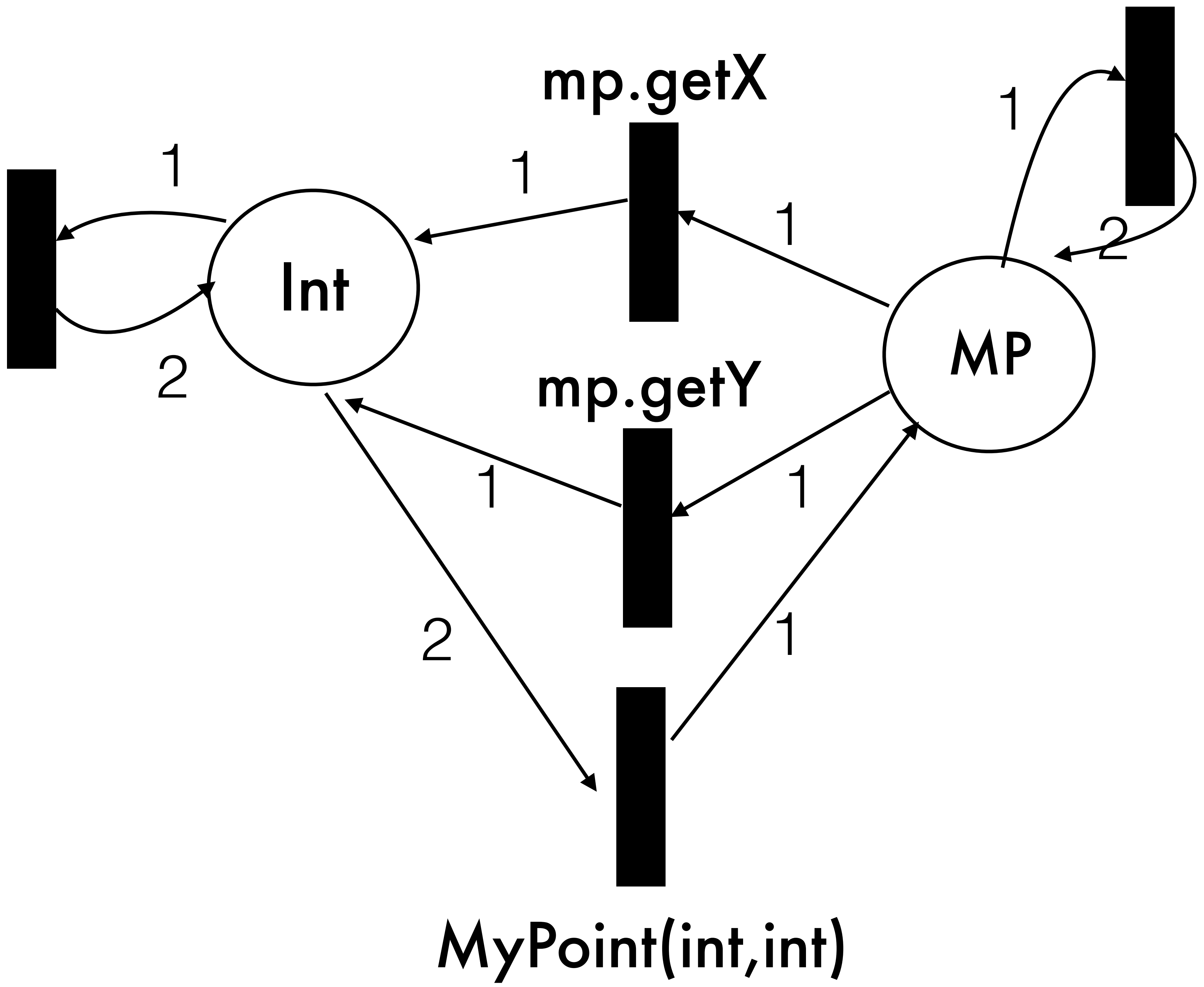
```
class MyPoint {
  MyPoint(int x, int y);
  int getX();
  int getY();
}
```

Building a Petri Net



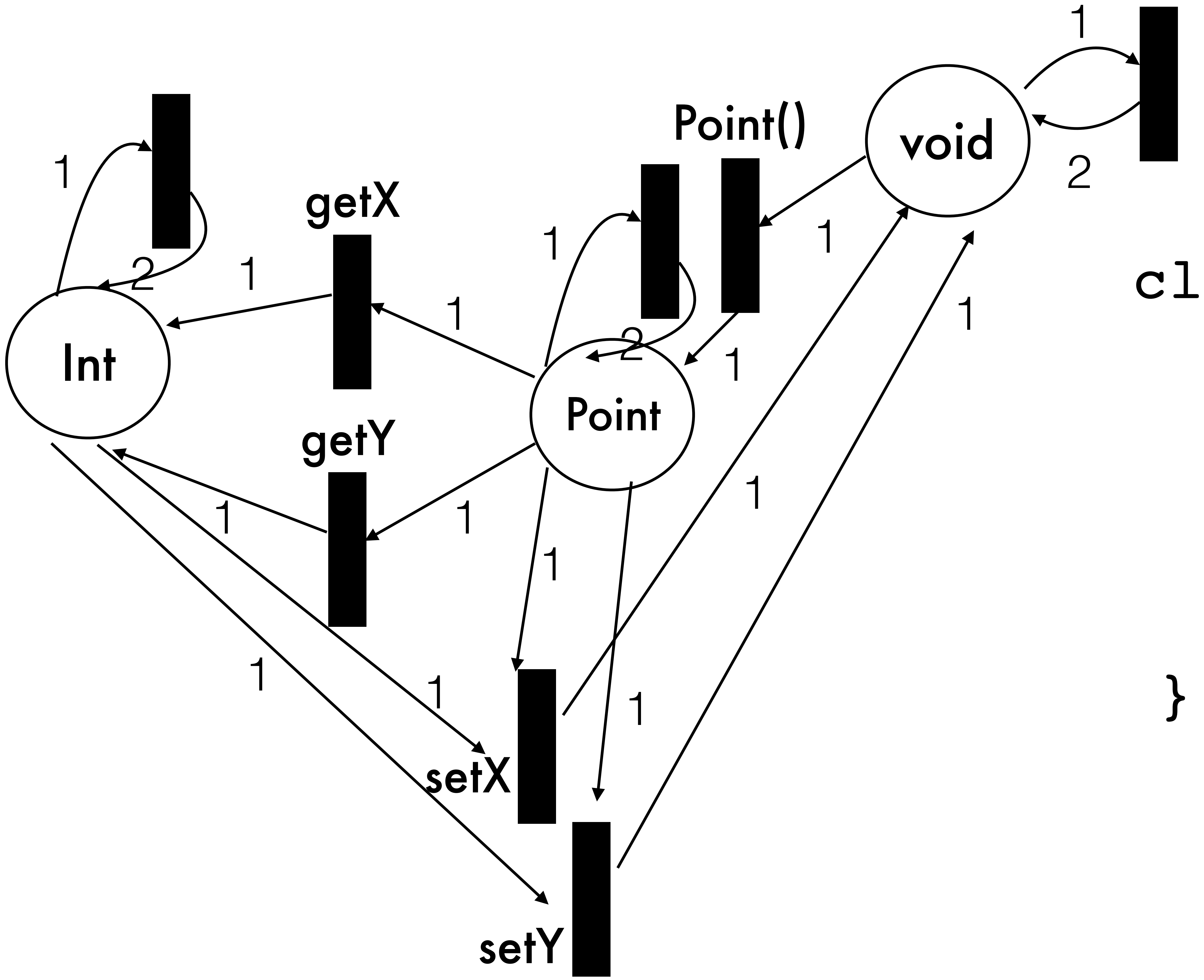

```
class MyPoint {
  MyPoint(int x, int y);
  int getX();
  int getY();
}
```

Building a Petri Net



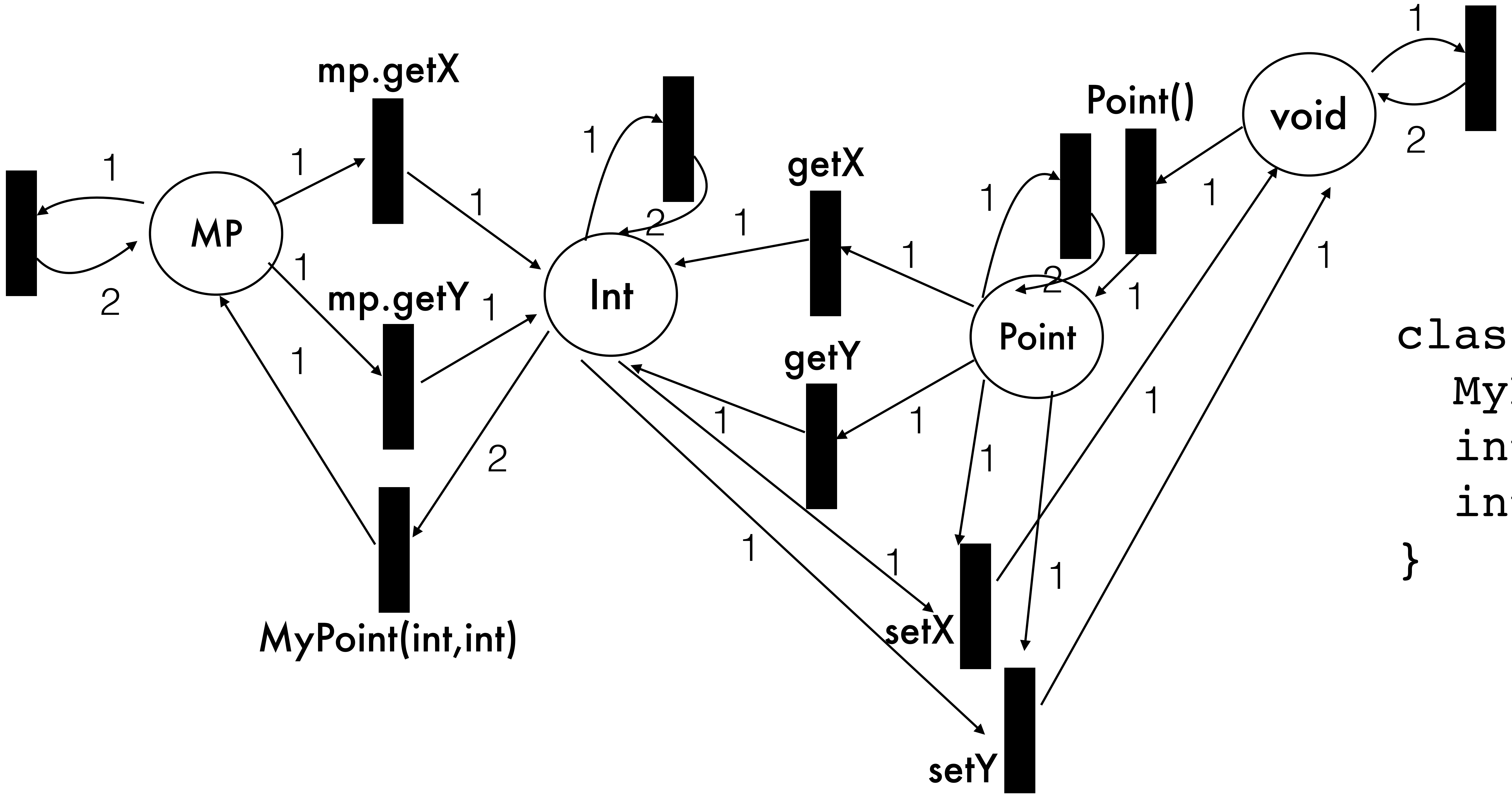

```
class MyPoint {
  MyPoint(int x, int y);
  int getX();
  int getY();
}
```

Building a Petri Net



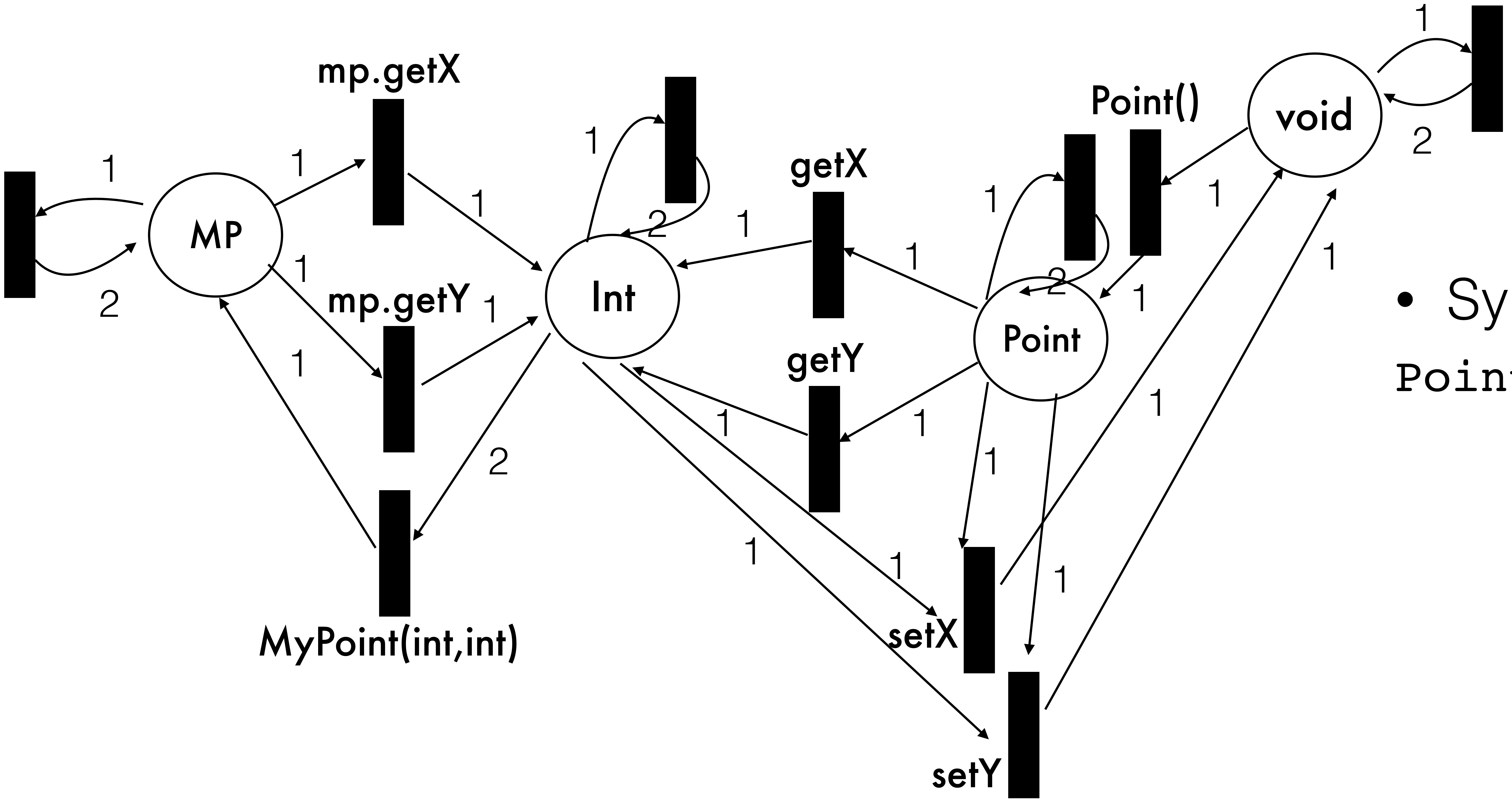

```
class Point {
  Point();
  int getX();
  int getY();
  void setX(int);
  void setY(int);
}
```

Building a Petri Net




```
class MyPoint {
  MyPoint(int x, int y);
  int getX();
  int getY();
}
```

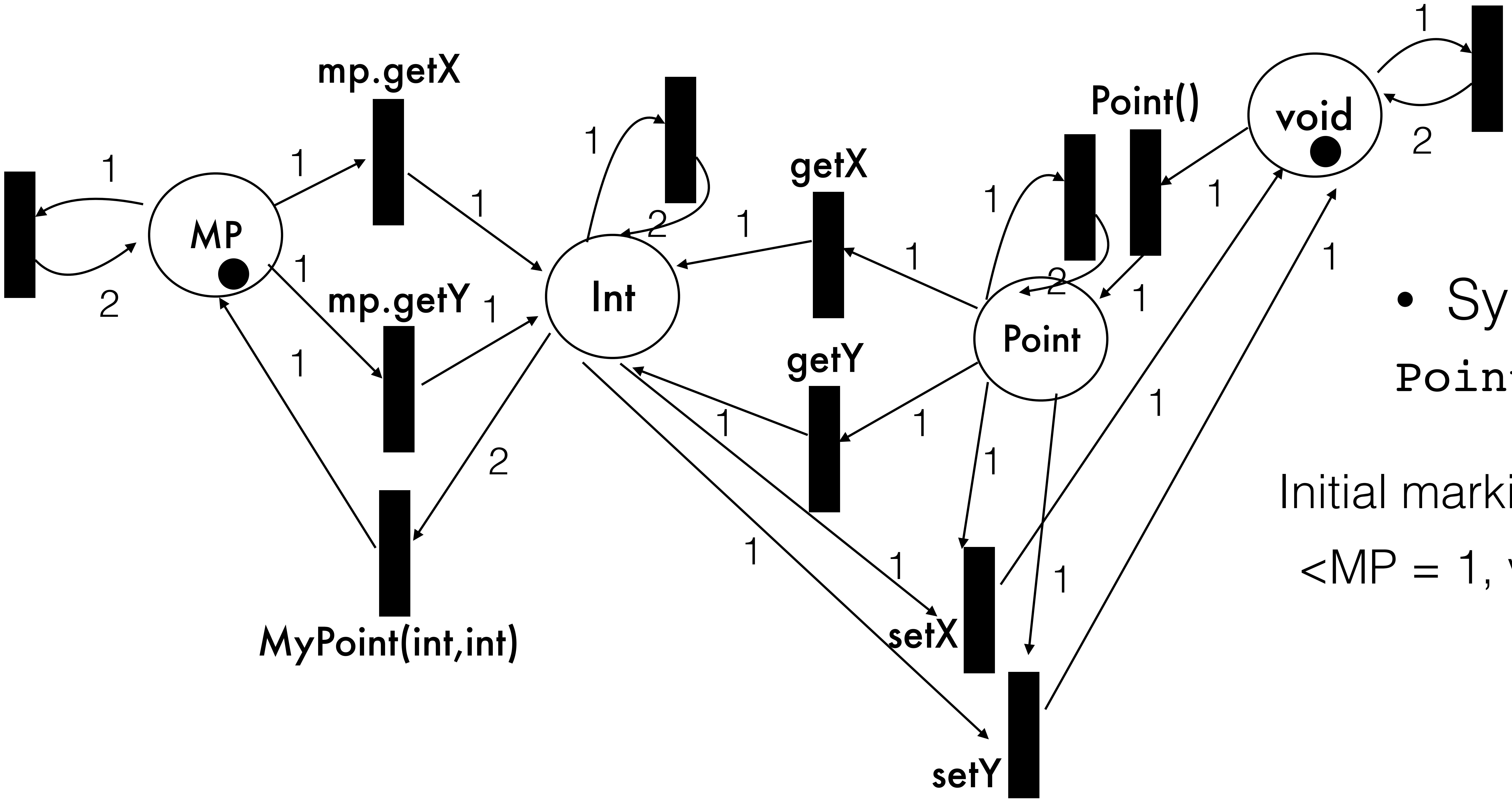
What is the Initial Marking?



- Synthesize this function:

```
Point convert(Mypoint pt)
```

What is the Initial Marking?



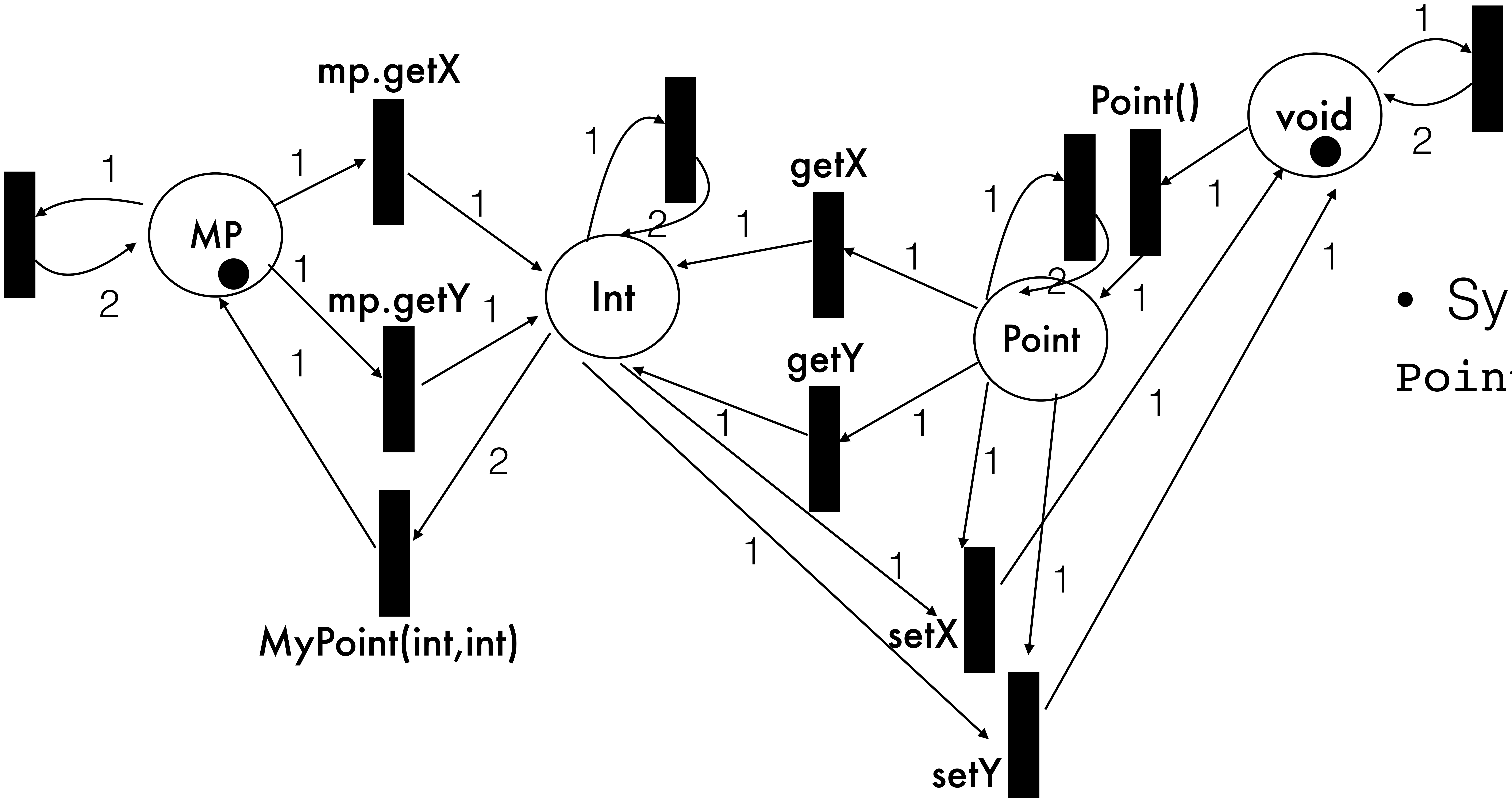
- Synthesize this function:

```
Point convert(Mypoint pt)
```

Initial marking:

<MP = 1, void = 1, Int = 0, Point = 0>

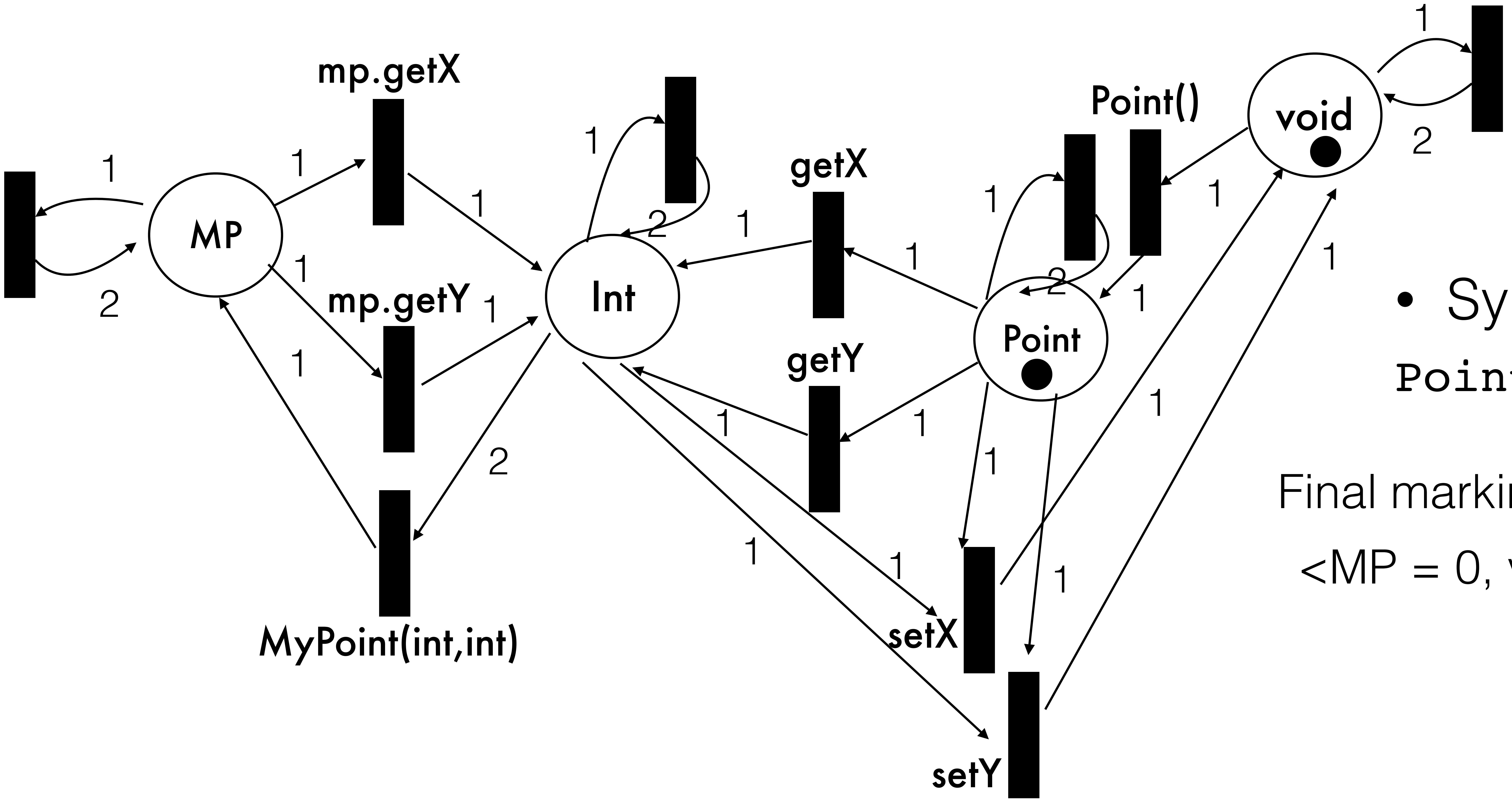
What is the Final Marking?



- Synthesize this function:

```
Point convert(Mypoint pt)
```

What is the Final Marking?



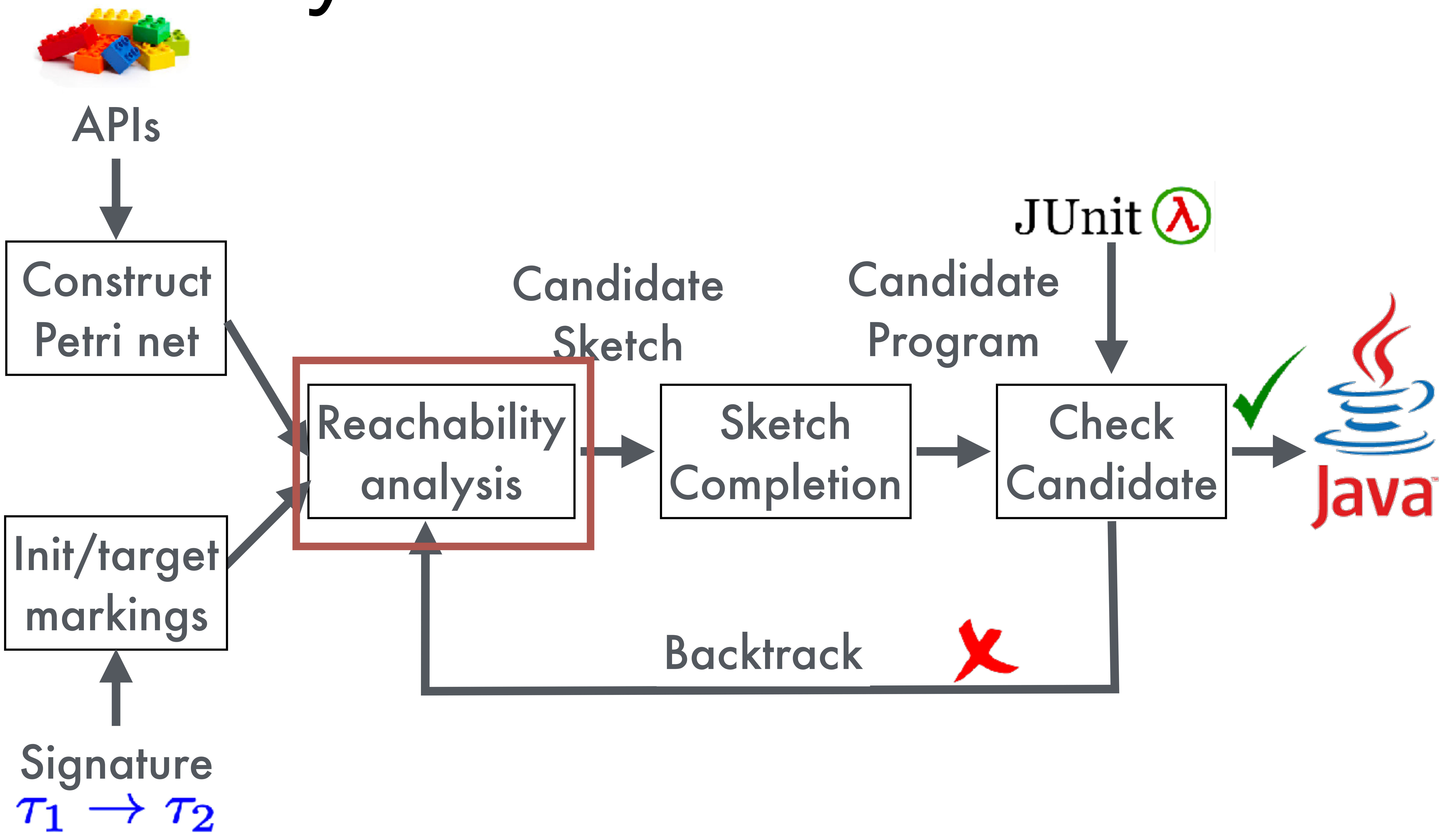
- Synthesize this function:

```
Point convert(Mypoint pt)
```

Final marking:

<MP = 0, void = *, Int = 0, Point = 1>

SyPet Architecture

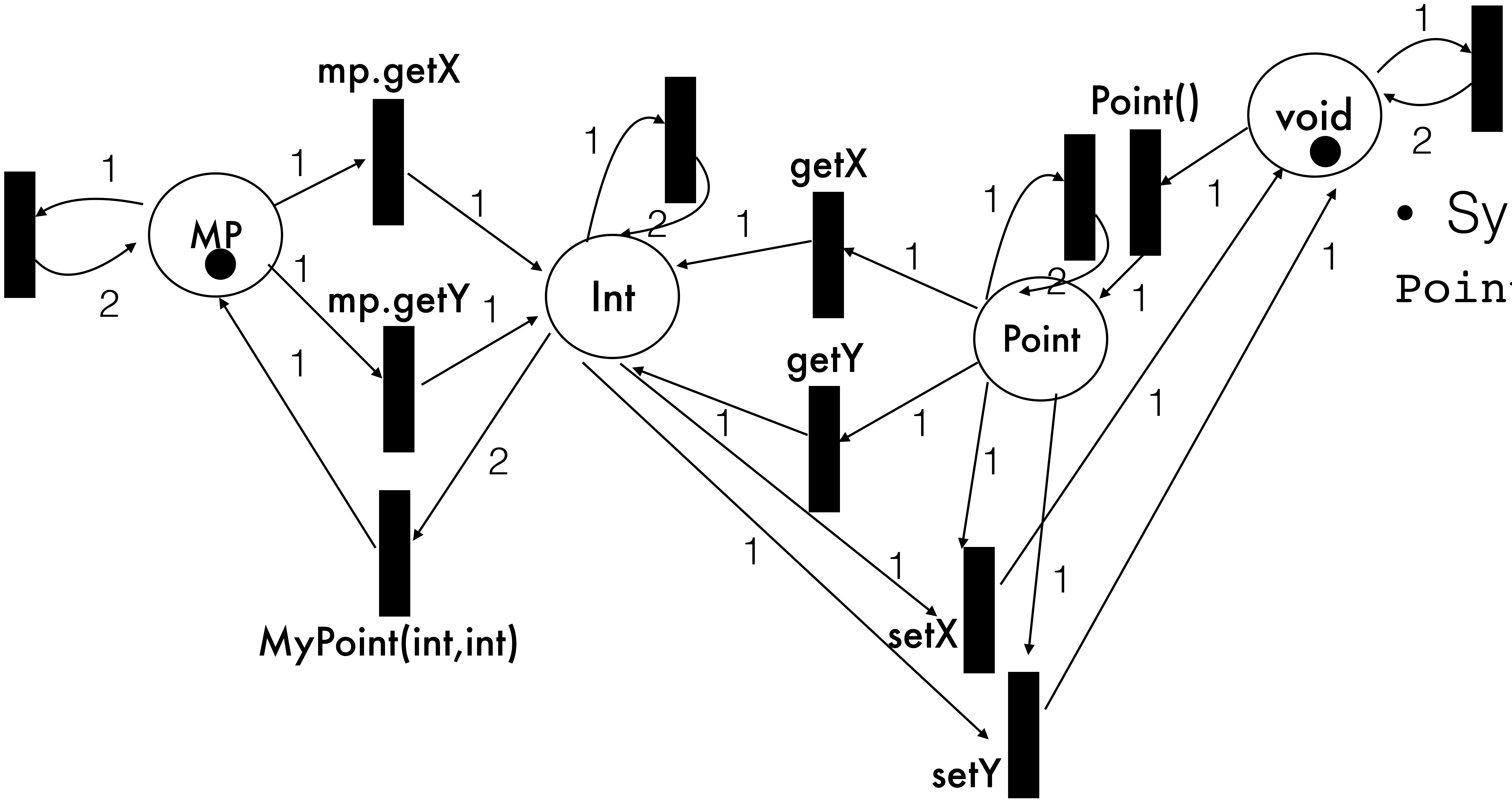


Reachability Analysis

All accepting runs of Petri net correspond to method call sequences with desired type signature!

- Need to perform reachability analysis to identify accepting runs of the Petri net
- Reachability analysis of Petri nets can be encoded to SAT:
  - Find a reachable path of size *k*
  - Enumerate all reachable paths

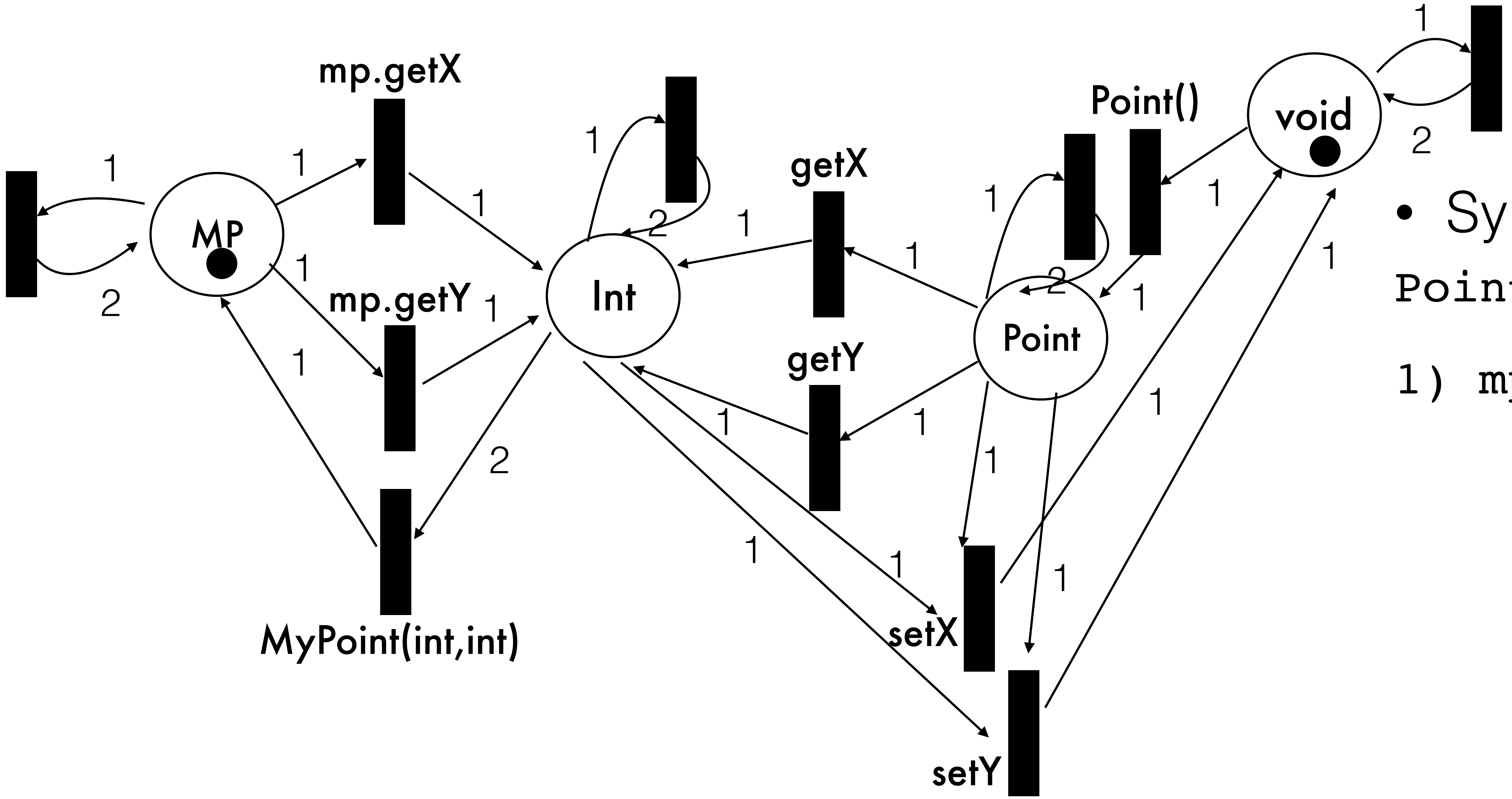
Reachable Paths



- Synthesize this function:

```
Point convert(Mypoint pt)
```

Not all Reachable Paths are a Solution!

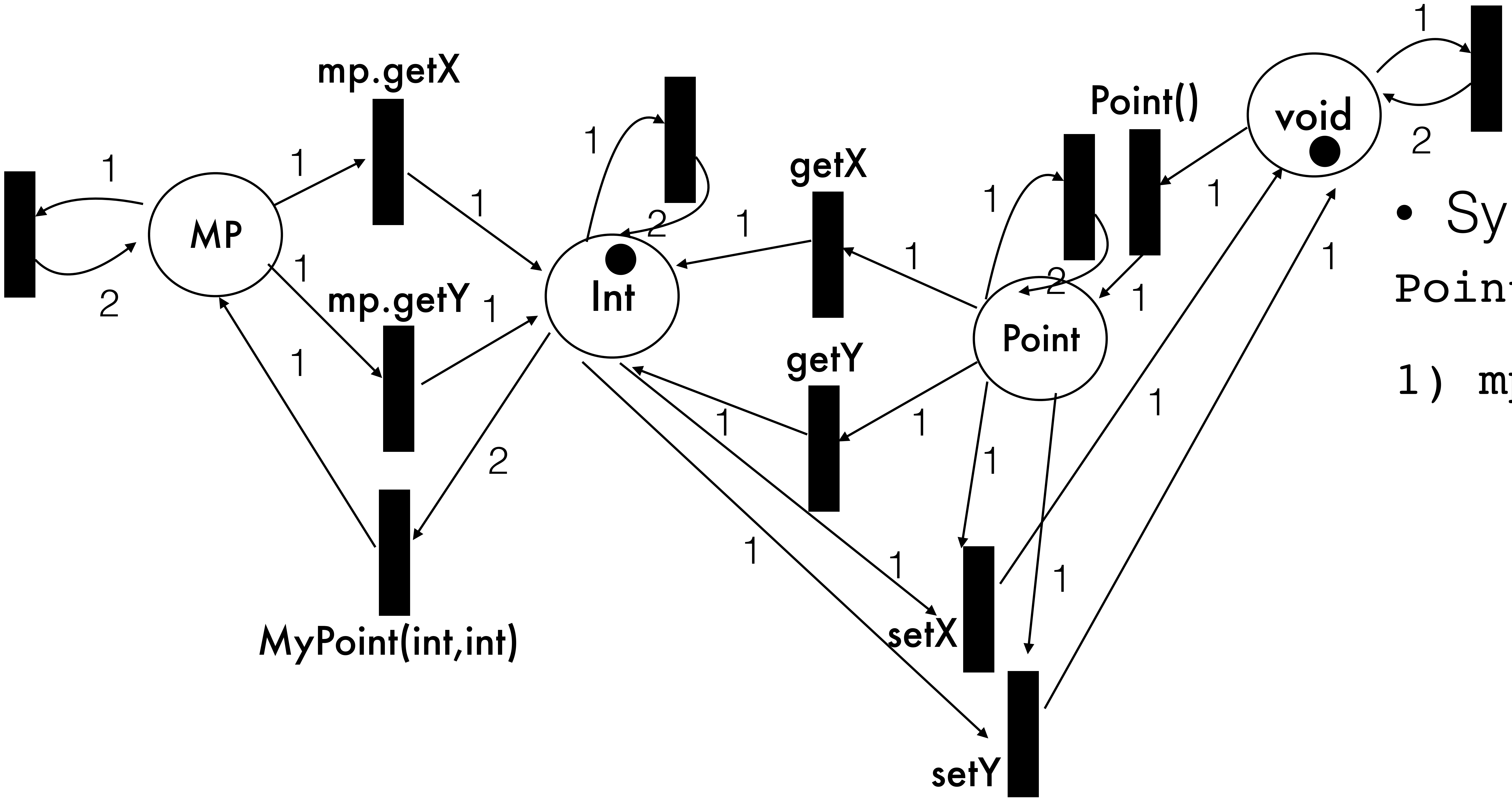


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) mp.getX
```

Not all Reachable Paths are a Solution!

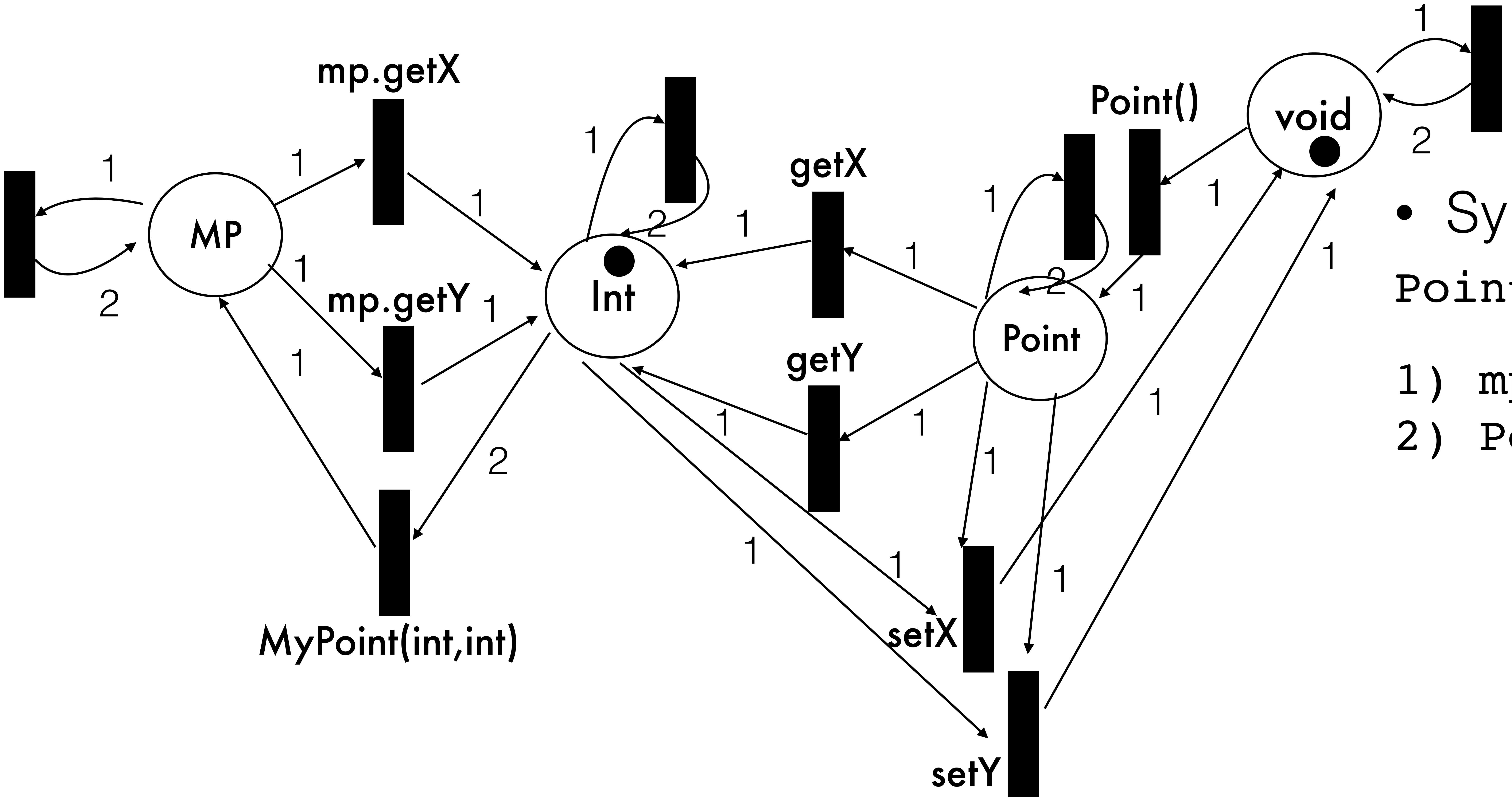


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) mp.getX
```

Not all Reachable Paths are a Solution!

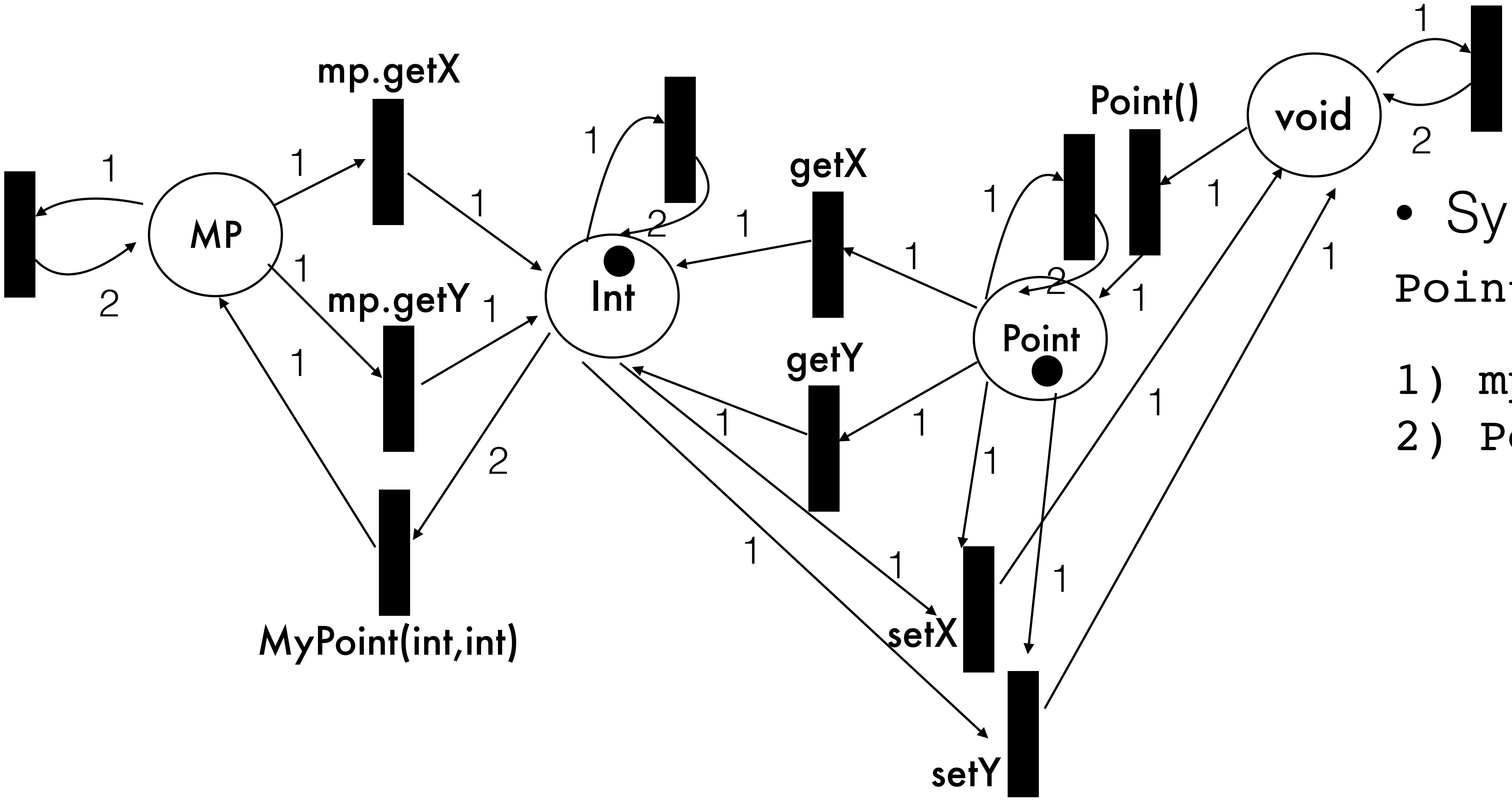


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) mp.getX
2) Point()
```

Not all Reachable Paths are a Solution!

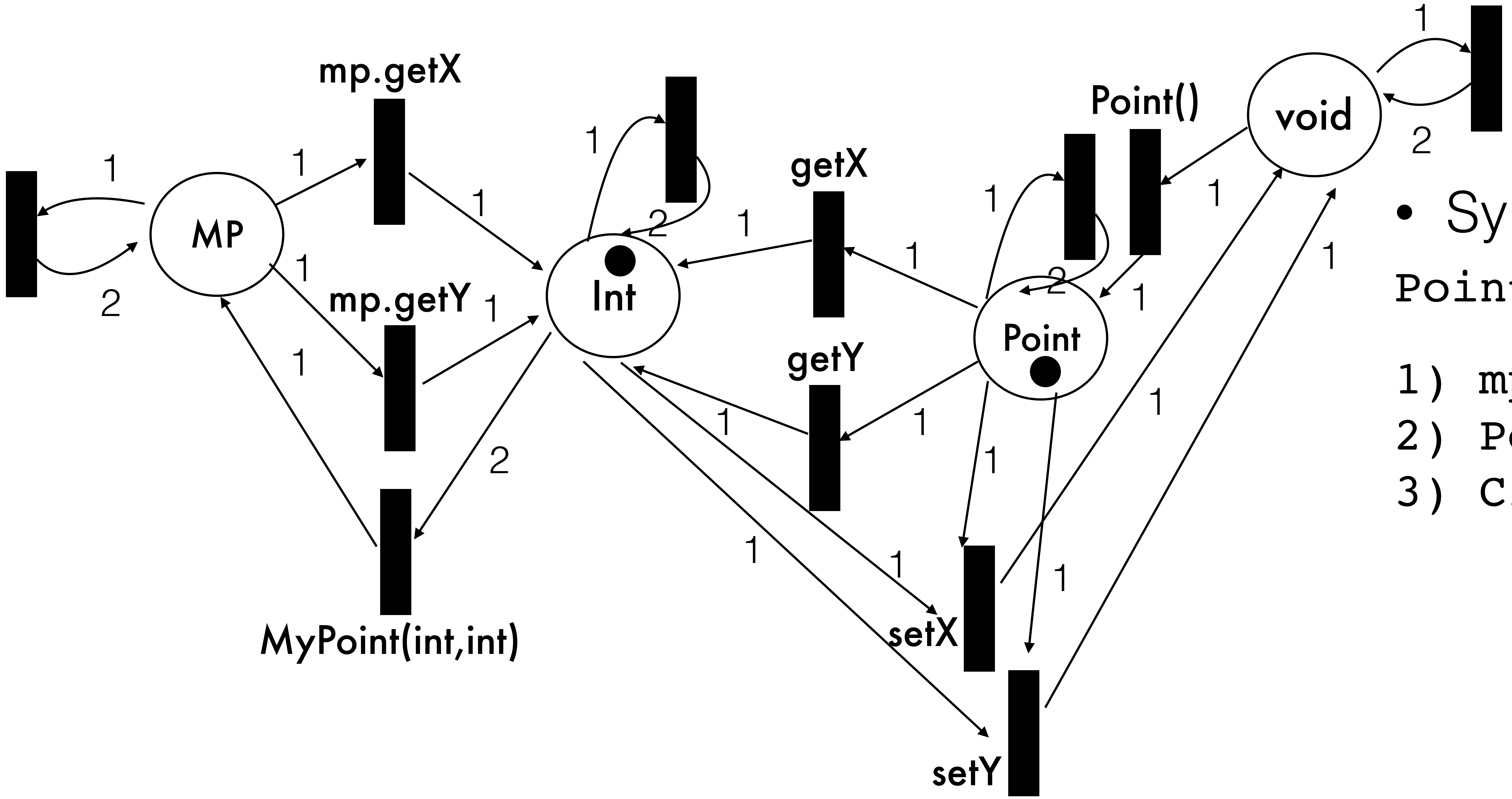


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) mp.getX
2) Point()
```

Not all Reachable Paths are a Solution!

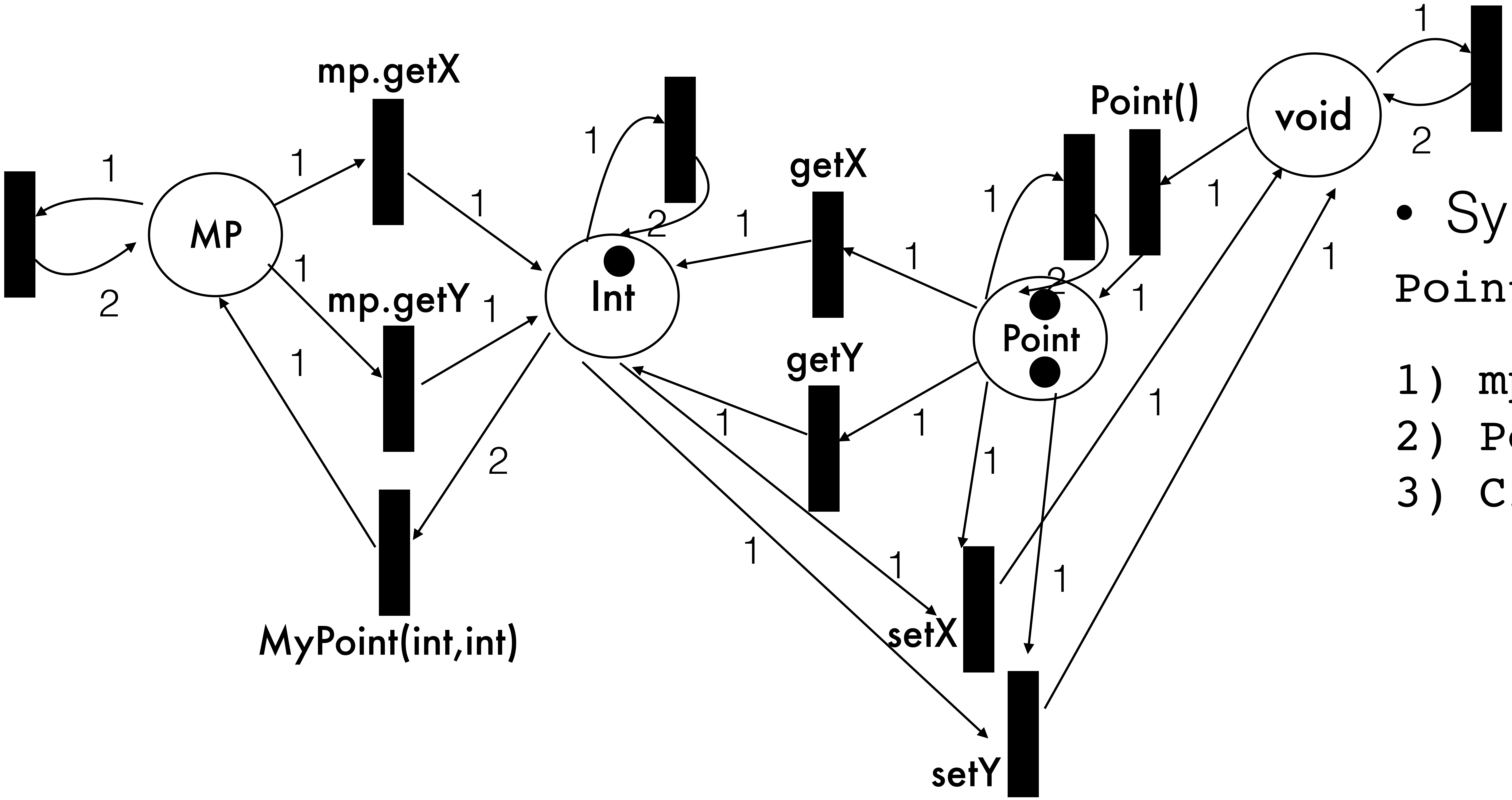


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) mp.getX
2) Point()
3) Clone-Point
```

Not all Reachable Paths are a Solution!

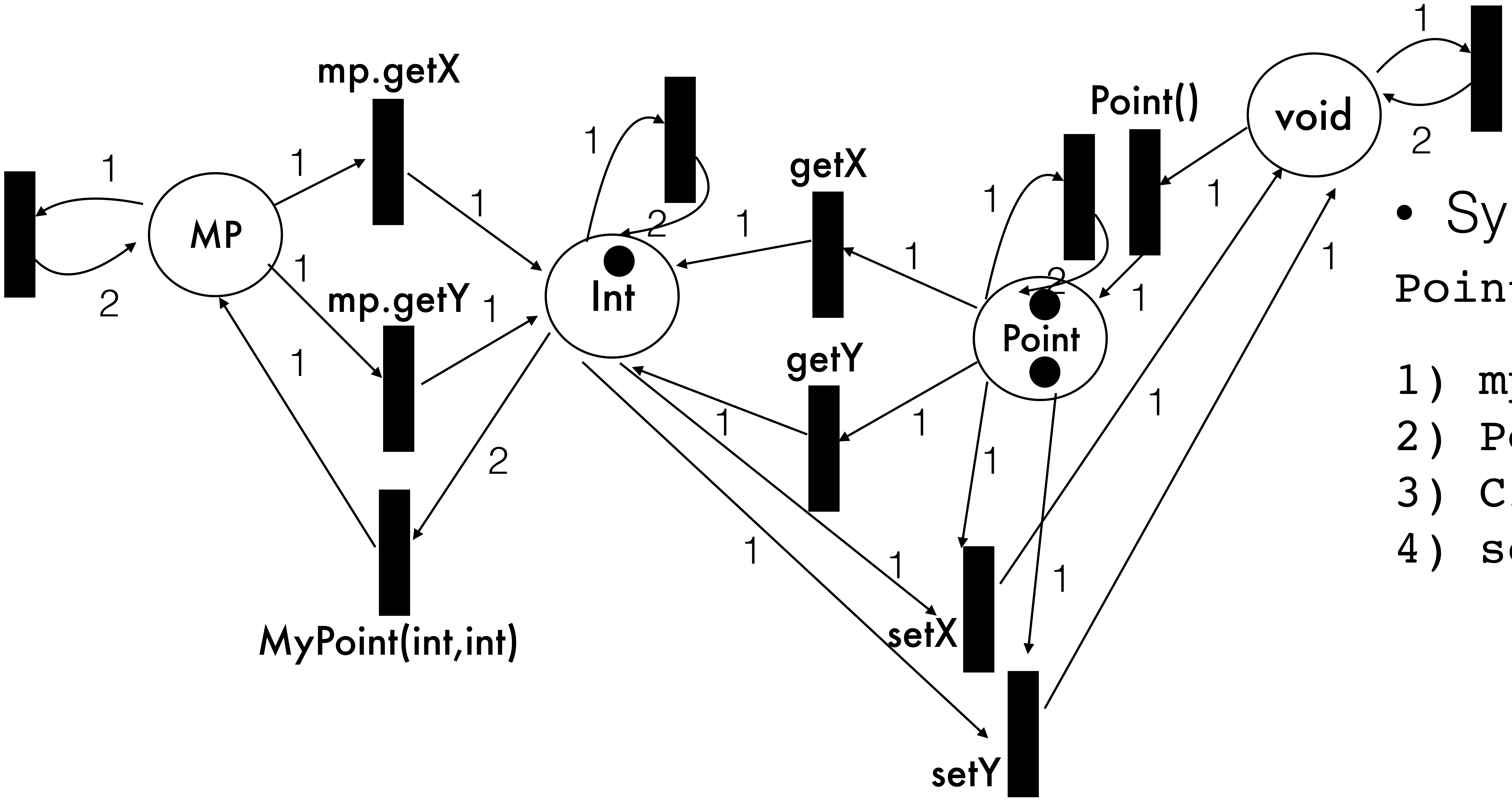


- Synthesize this function:

```
Point convert(Mypoint pt)

1) mp.getX
2) Point()
3) Clone-Point
```

Not all Reachable Paths are a Solution!

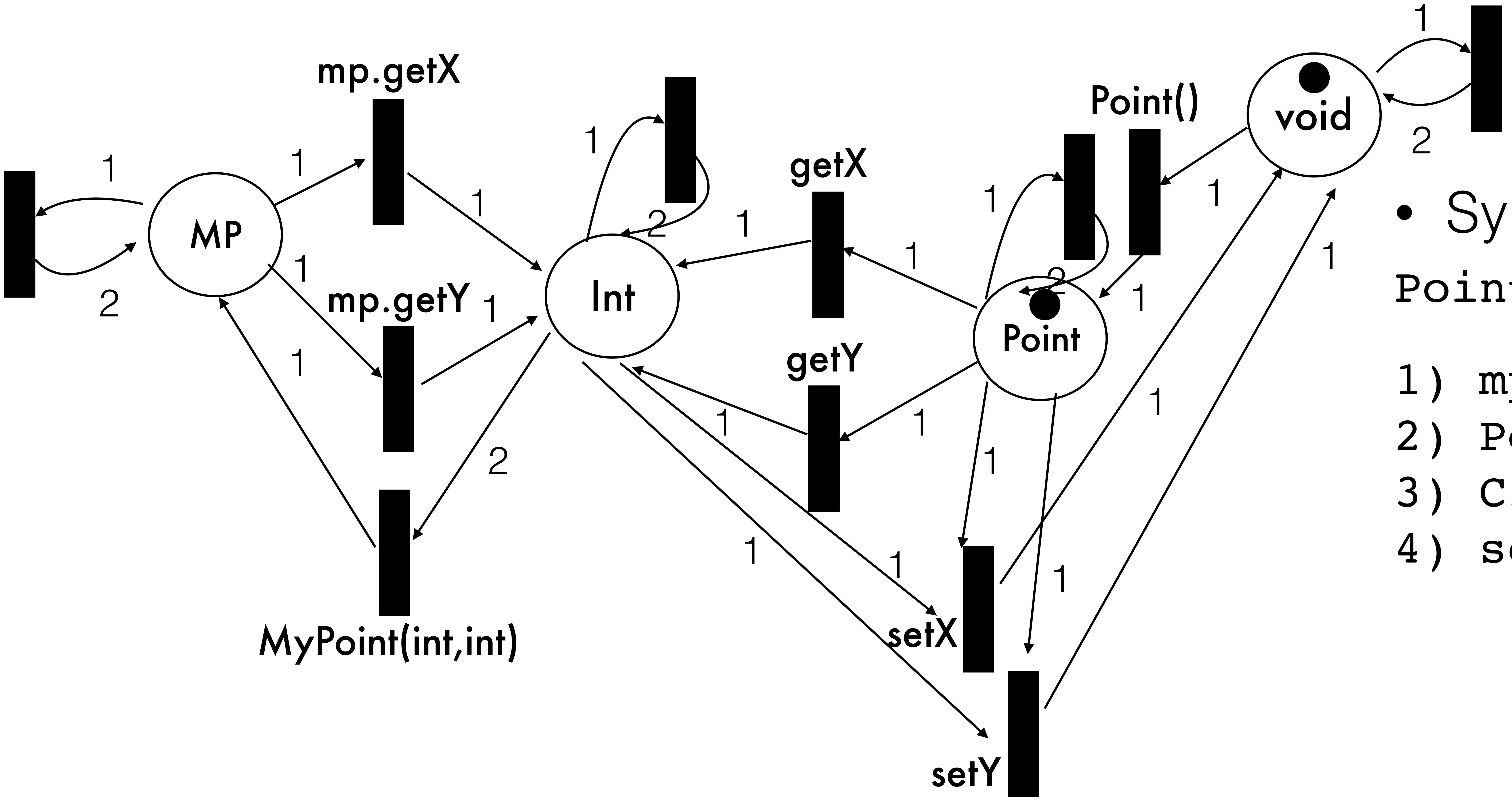


- Synthesize this function:

```
Point convert(Mypoint pt)

1) mp.getX
2) Point()
3) Clone-Point
4) setX
```

Not all Reachable Paths are a Solution!

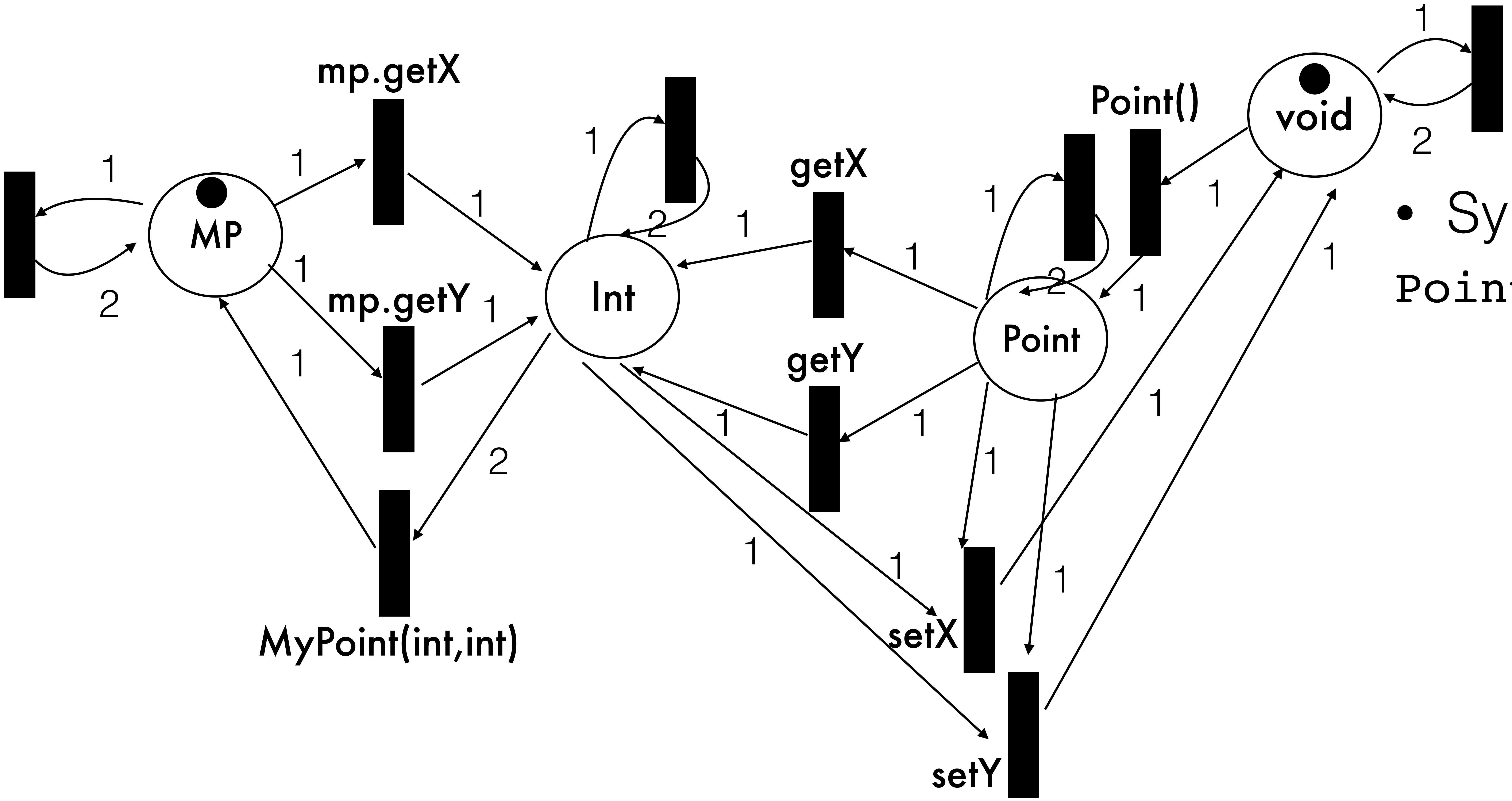


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) mp.getX
2) Point()
3) Clone-Point
4) setX
```

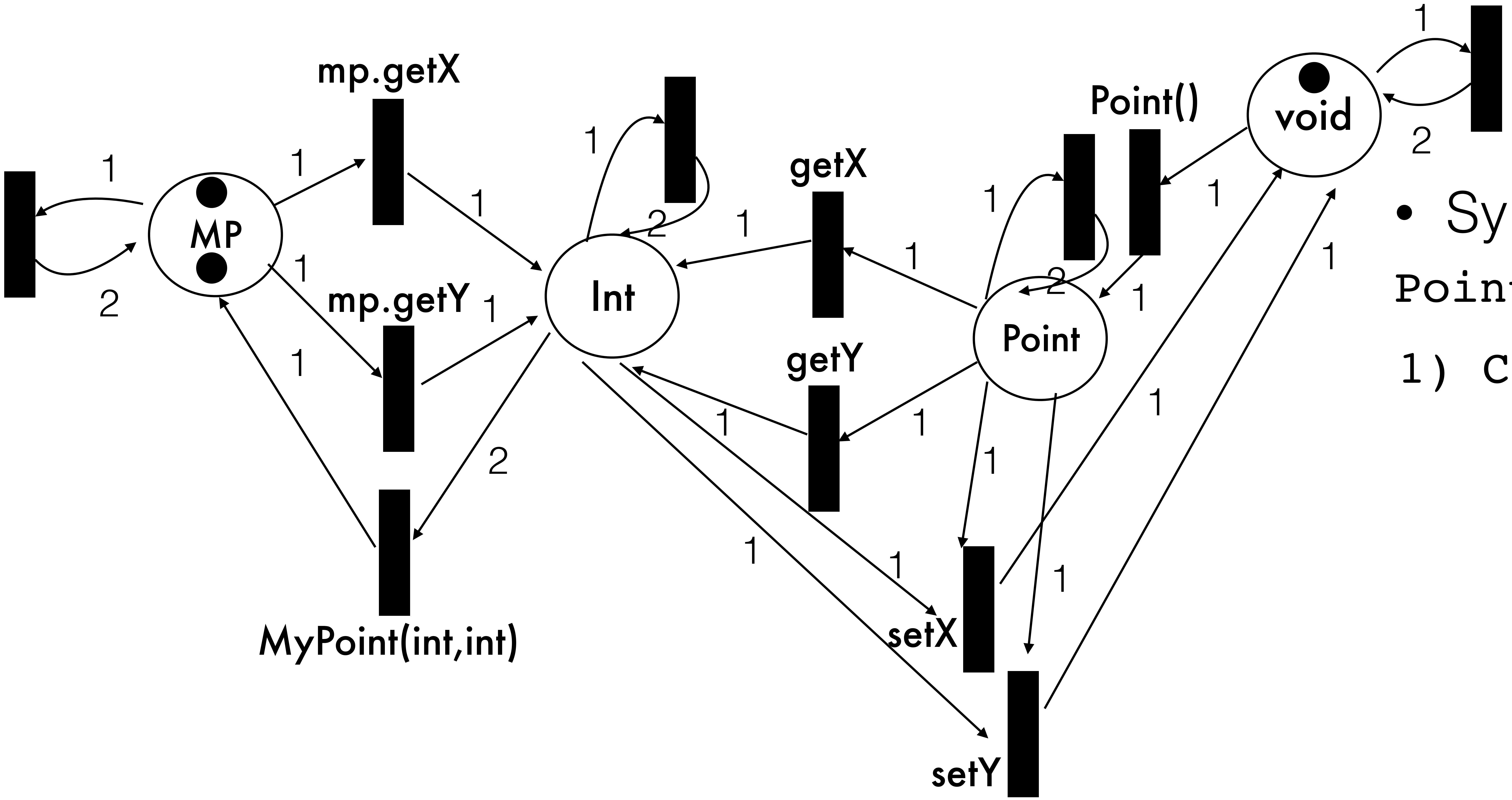
Reachable Path that Corresponds to a Solution



- Synthesize this function:

```
Point convert(Mypoint pt)
```

Reachable Path that Corresponds to a Solution

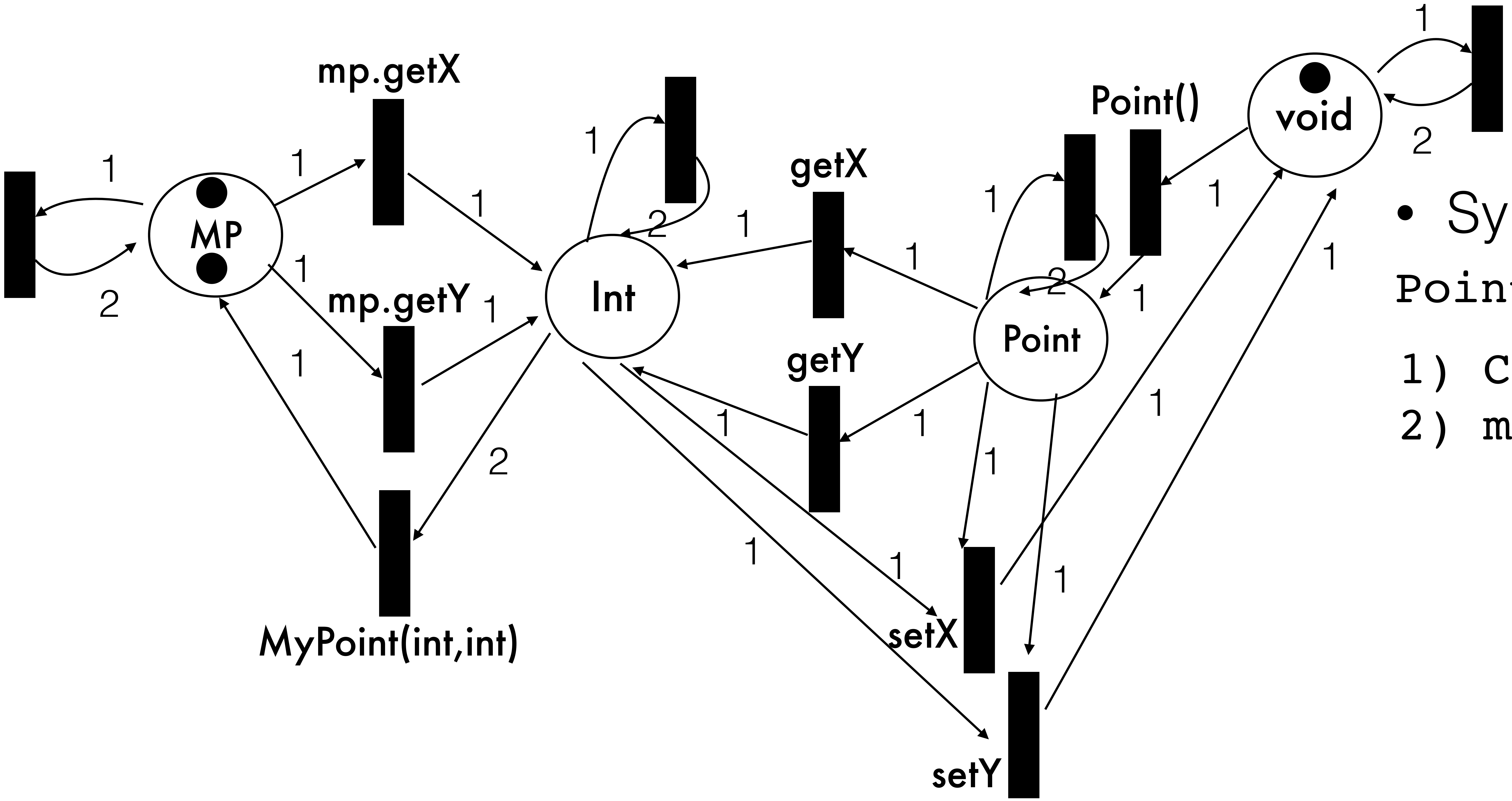


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) Clone-MP
```

Reachable Path that Corresponds to a Solution

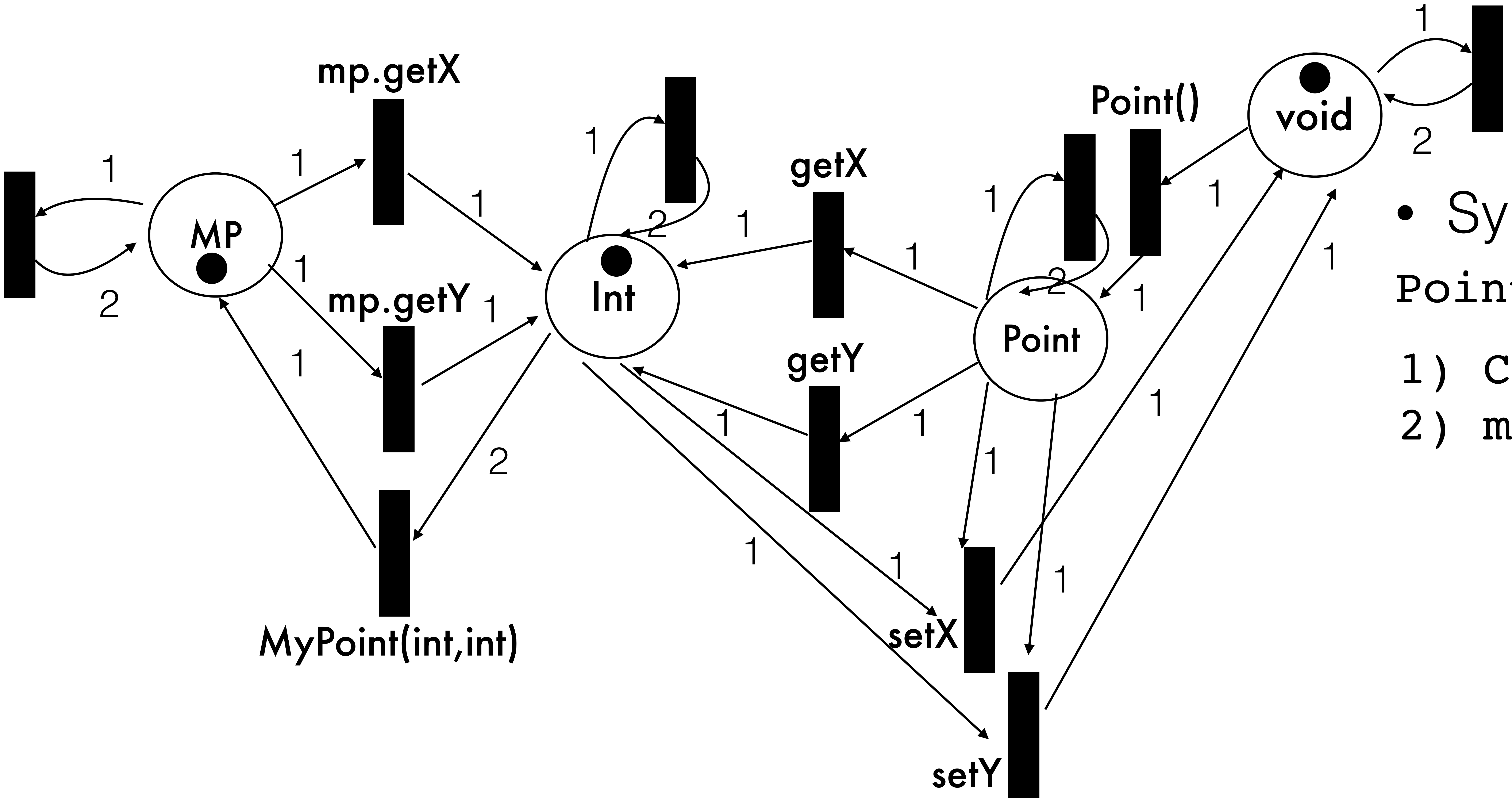


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) Clone-MP
2) mp.getX
```

Reachable Path that Corresponds to a Solution

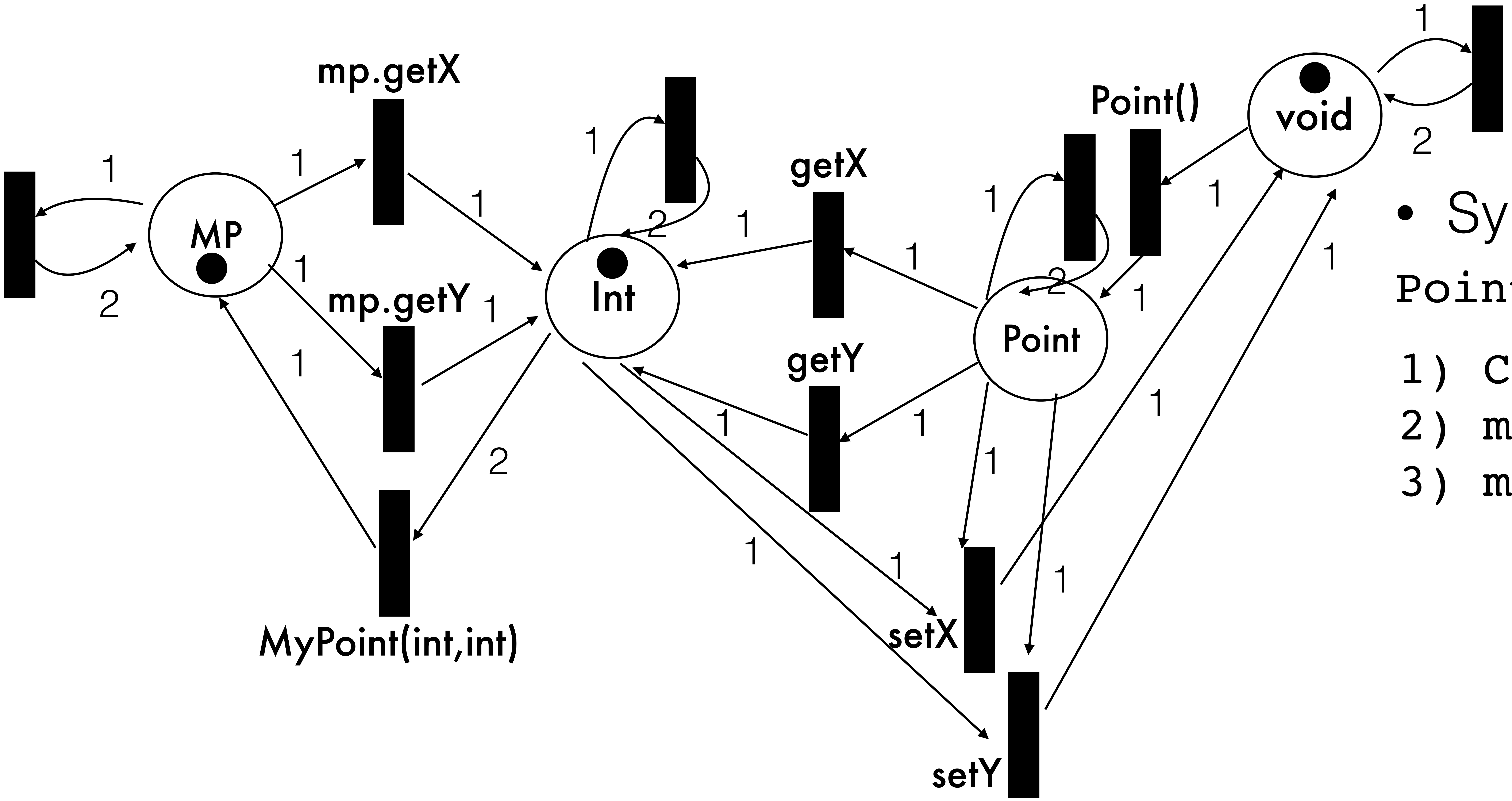


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) Clone-MP
2) mp.getX
```

Reachable Path that Corresponds to a Solution

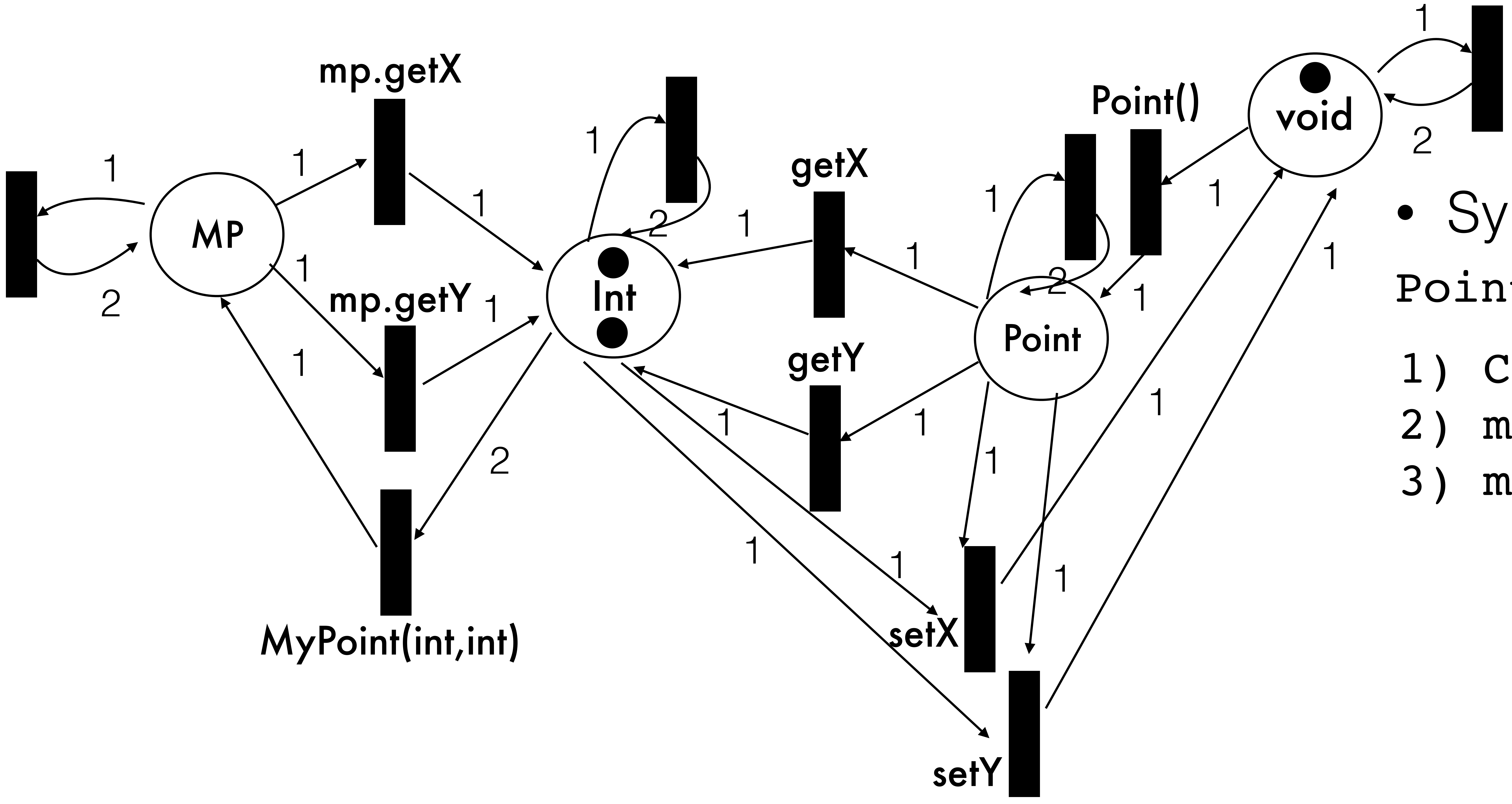


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) Clone-MP
2) mp.getX
3) mp.getY
```

Reachable Path that Corresponds to a Solution

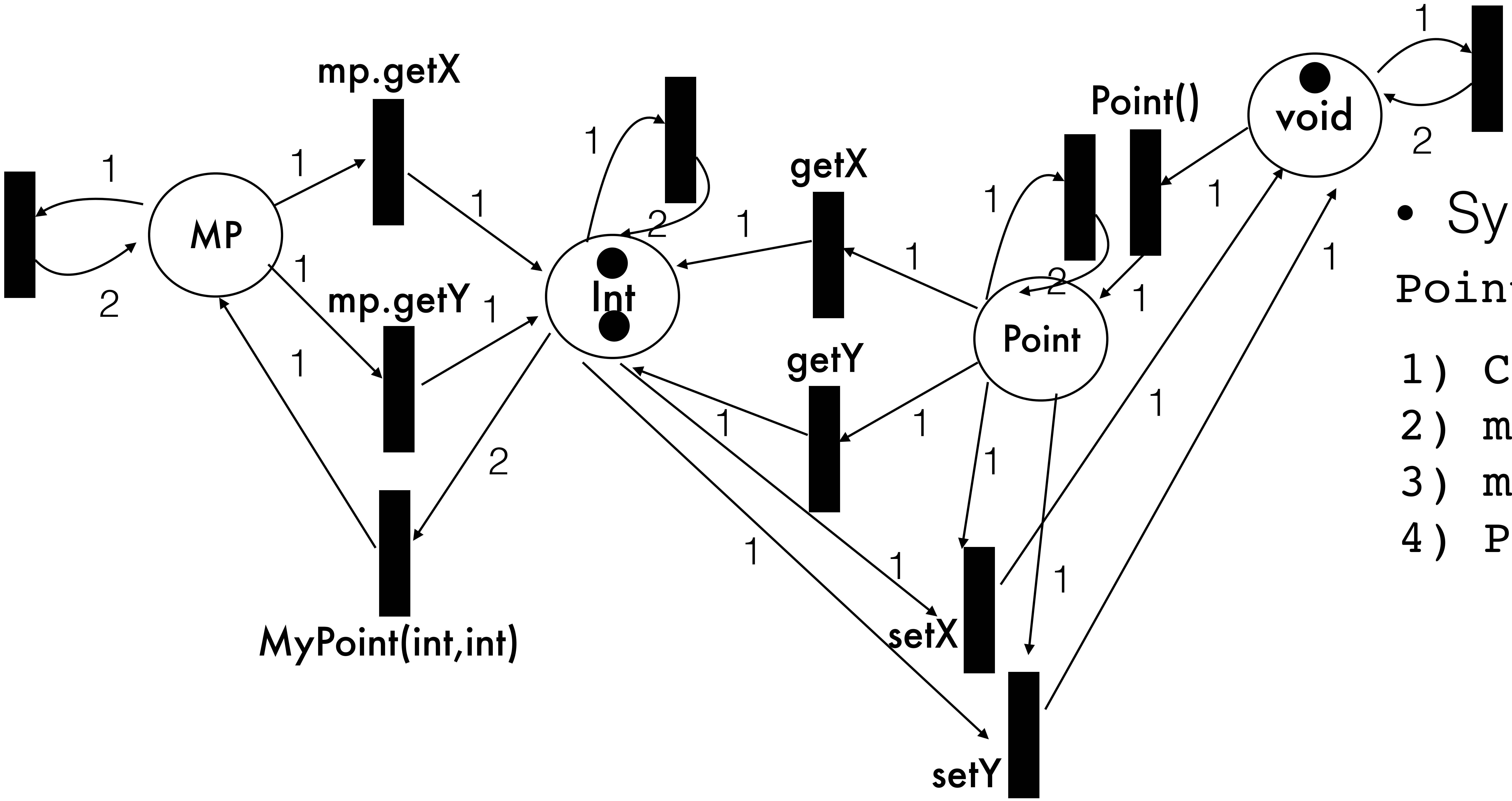


- Synthesize this function:

```
Point convert(Mypoint pt)

1) Clone-MP
2) mp.getX
3) mp.getY
```

Reachable Path that Corresponds to a Solution

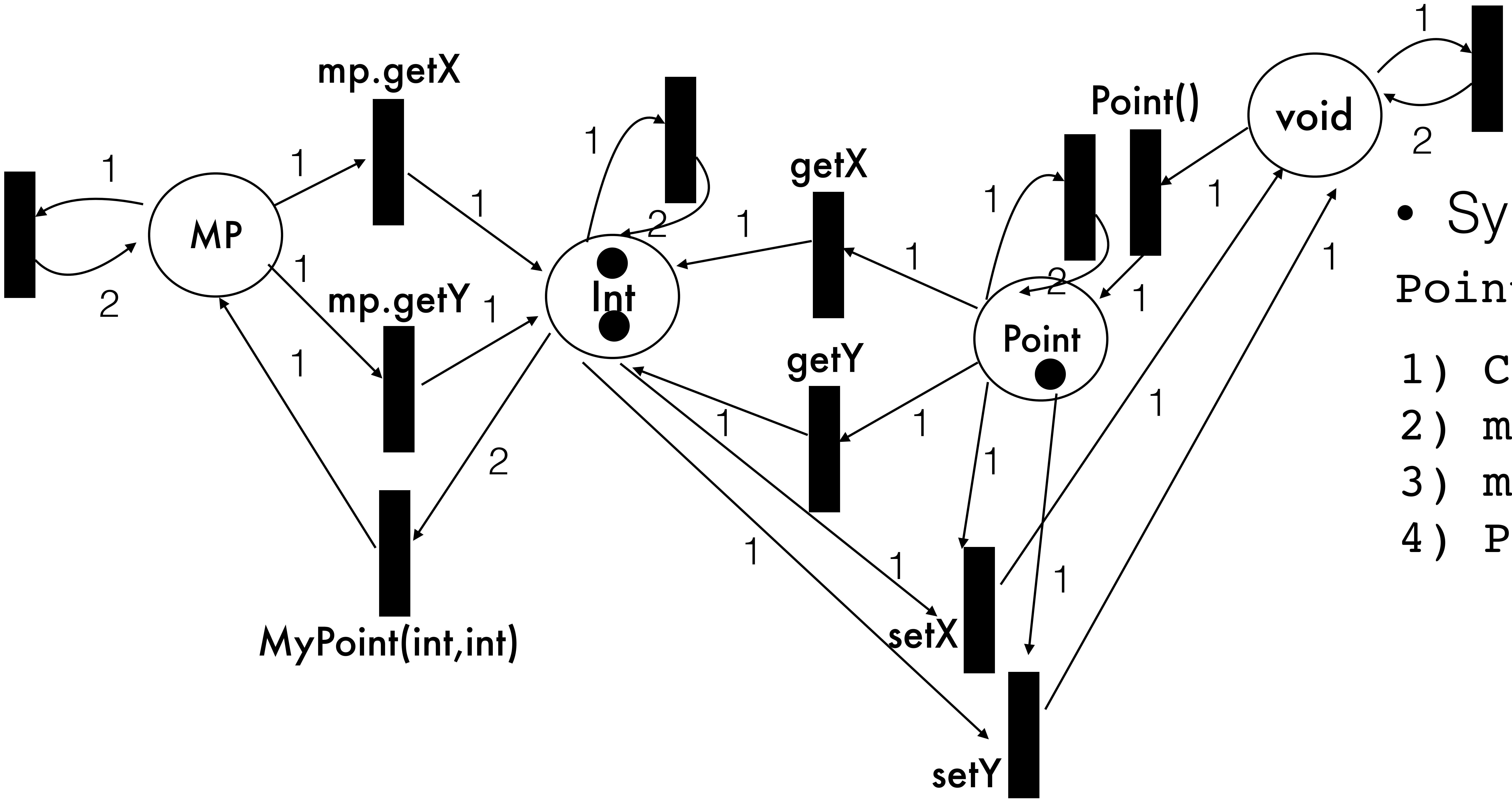


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) Clone-MP
2) mp.getX
3) mp.getY
4) Point()
```

Reachable Path that Corresponds to a Solution

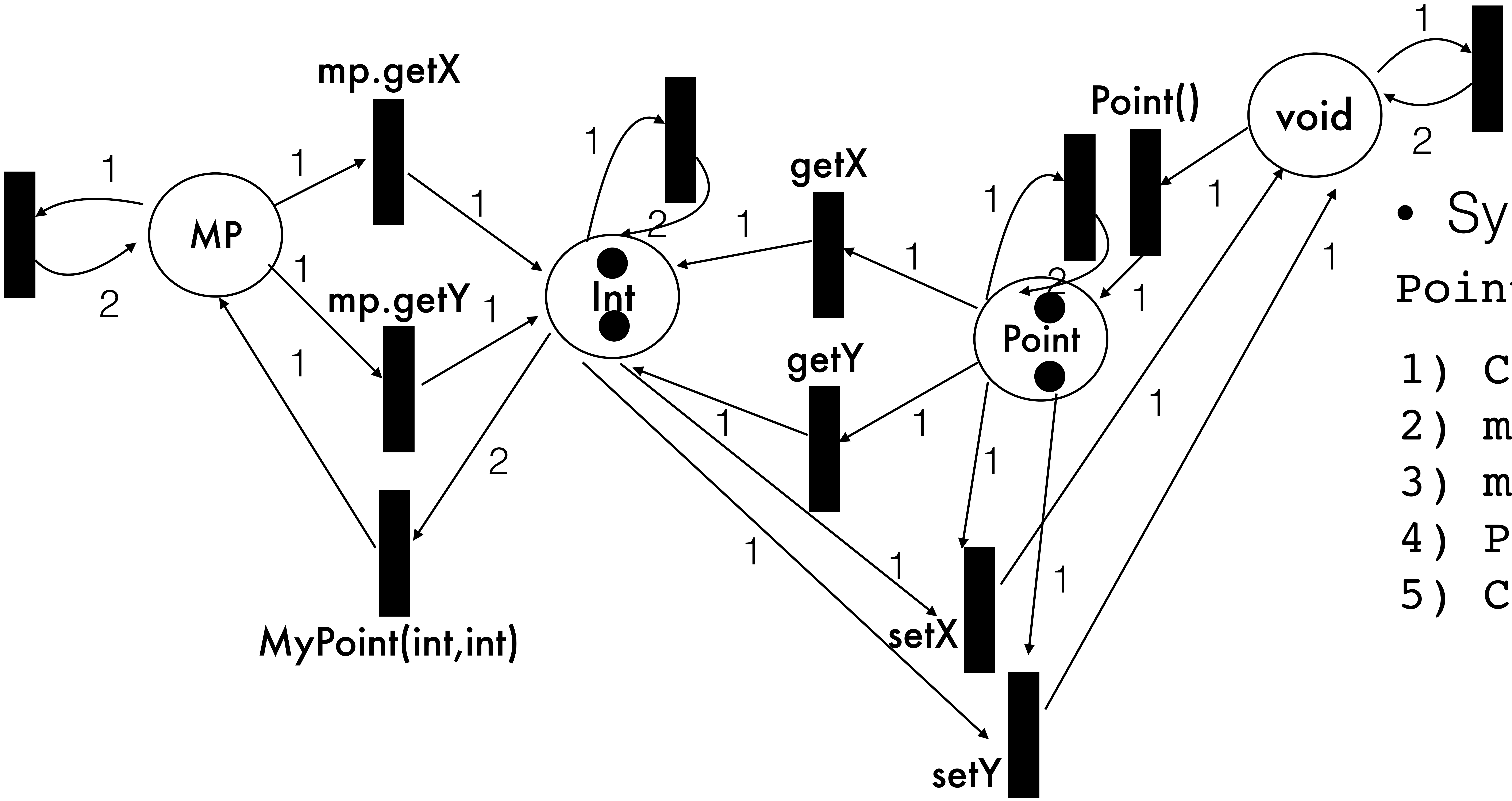


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) Clone-MP
2) mp.getX
3) mp.getY
4) Point()
```

Reachable Path that Corresponds to a Solution

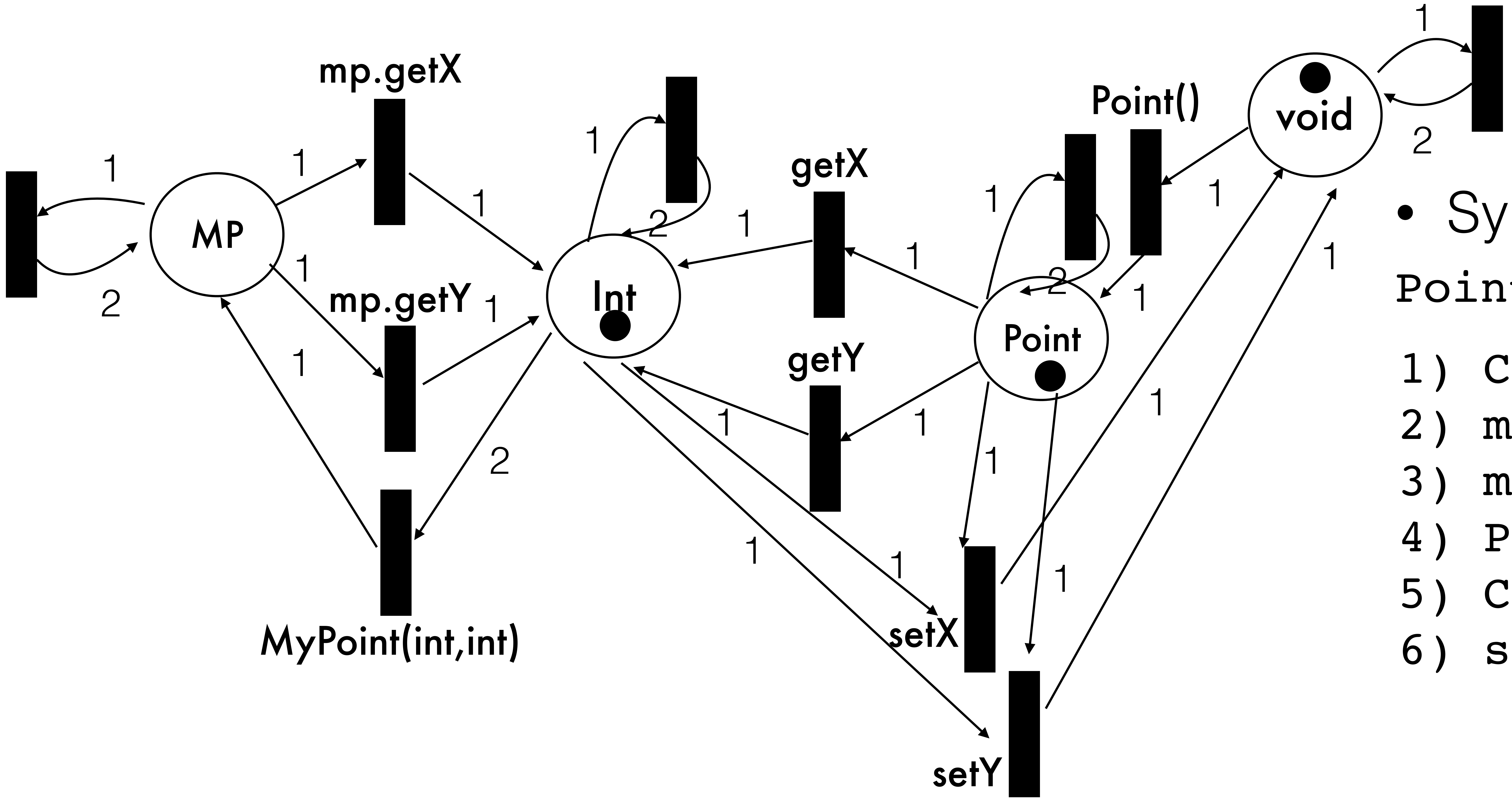


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) Clone-MP
2) mp.getX
3) mp.getY
4) Point()
5) Clone-Point
```

Reachable Path that Corresponds to a Solution

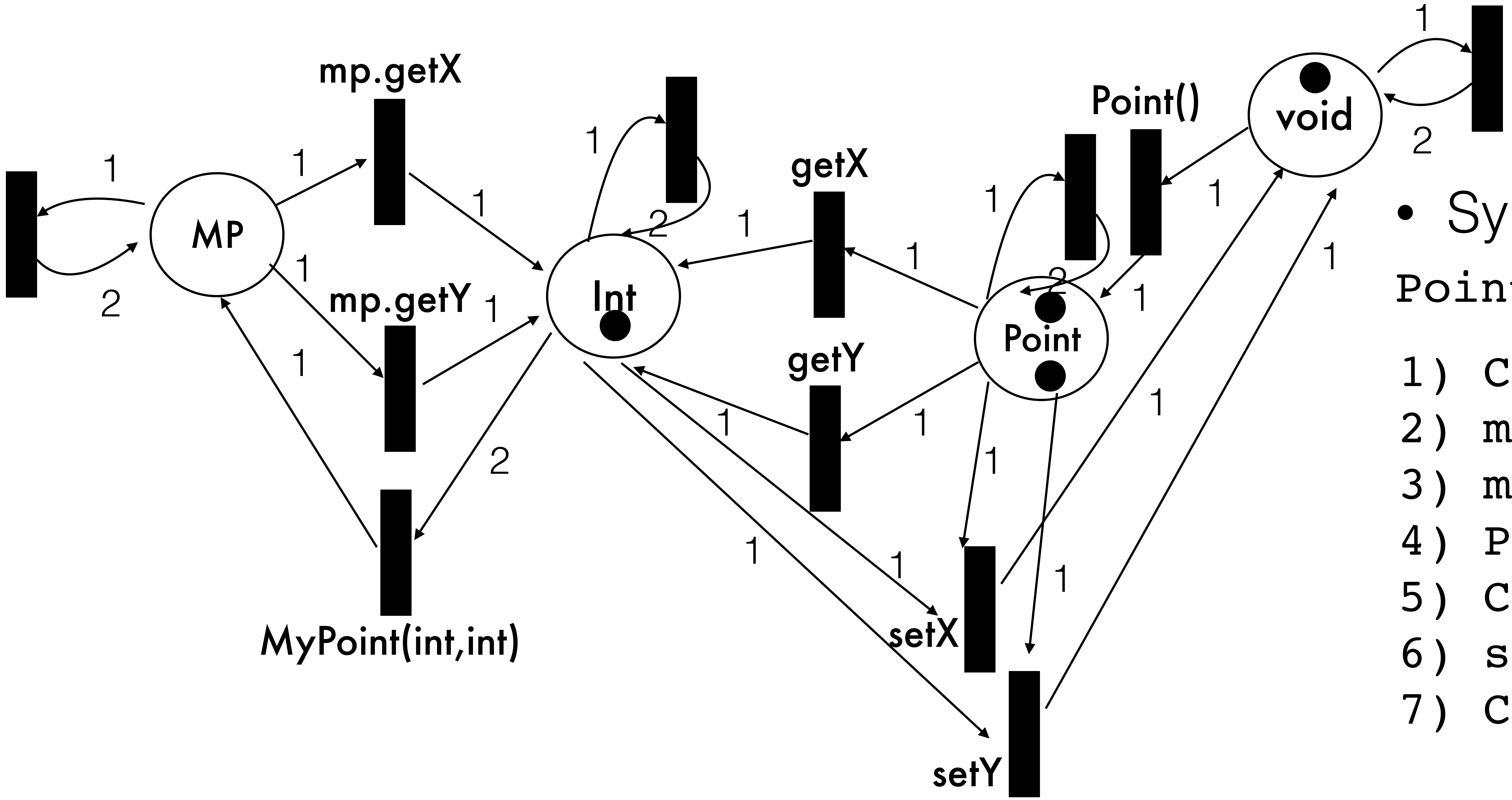


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) Clone-MP
2) mp.getX
3) mp.getY
4) Point()
5) Clone-Point
6) setX
```

Reachable Path that Corresponds to a Solution

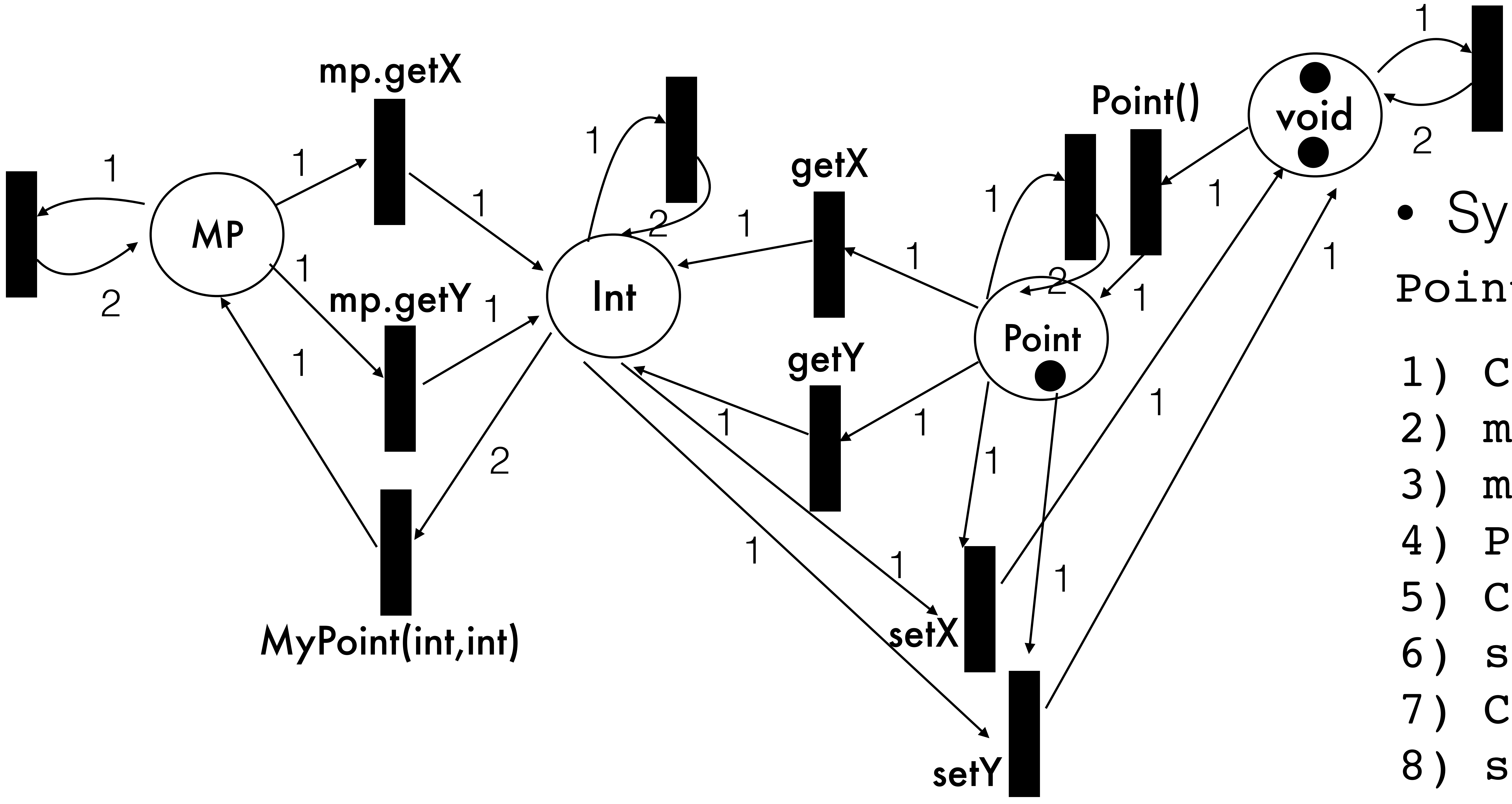


- Synthesize this function:

```
Point convert(Mypoint pt)
```

```
1) Clone-MP
2) mp.getX
3) mp.getY
4) Point()
5) Clone-Point
6) setX
7) Clone-Point
```

Reachable Path that Corresponds to a Solution

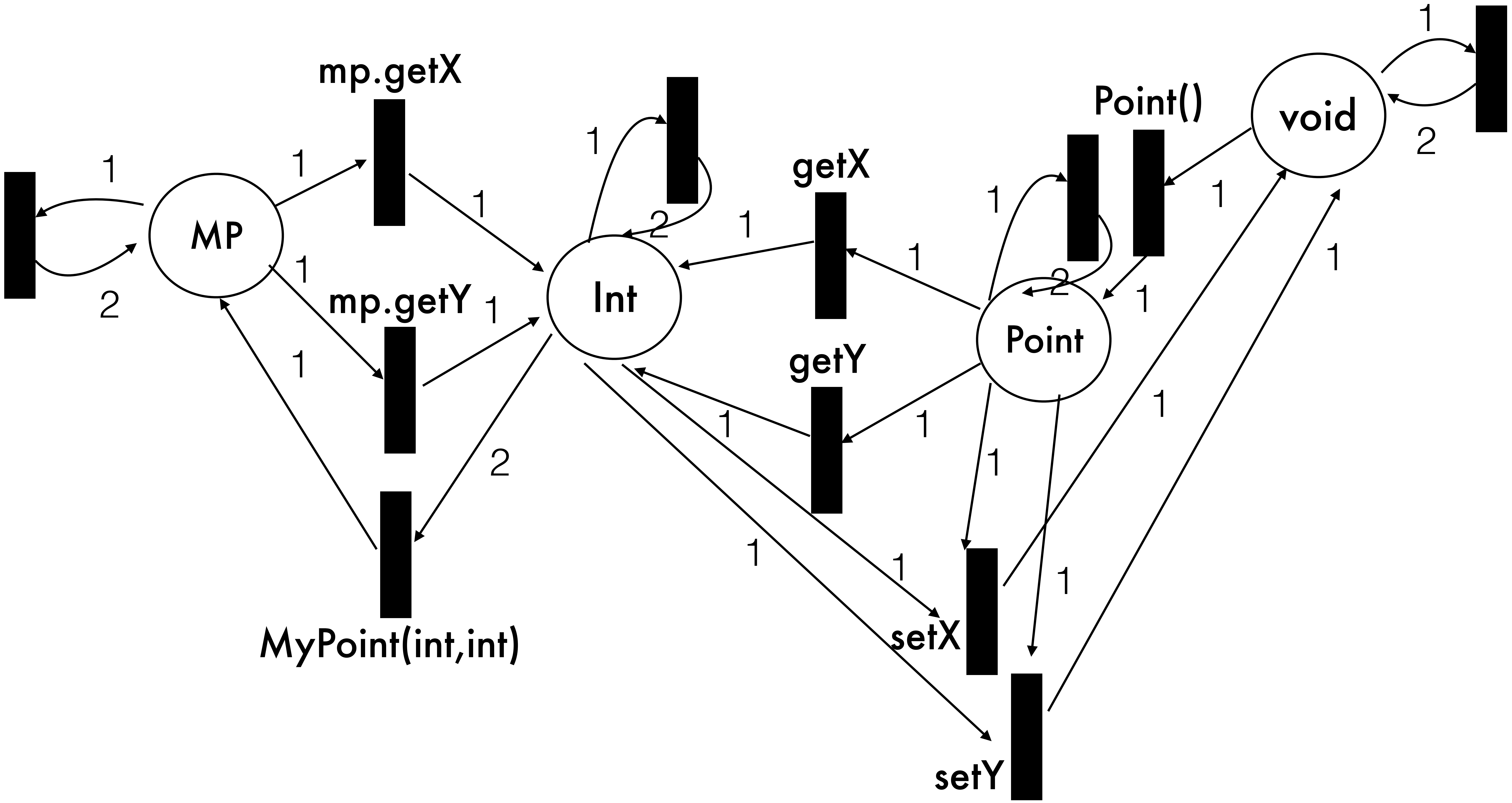


- Synthesize this function:

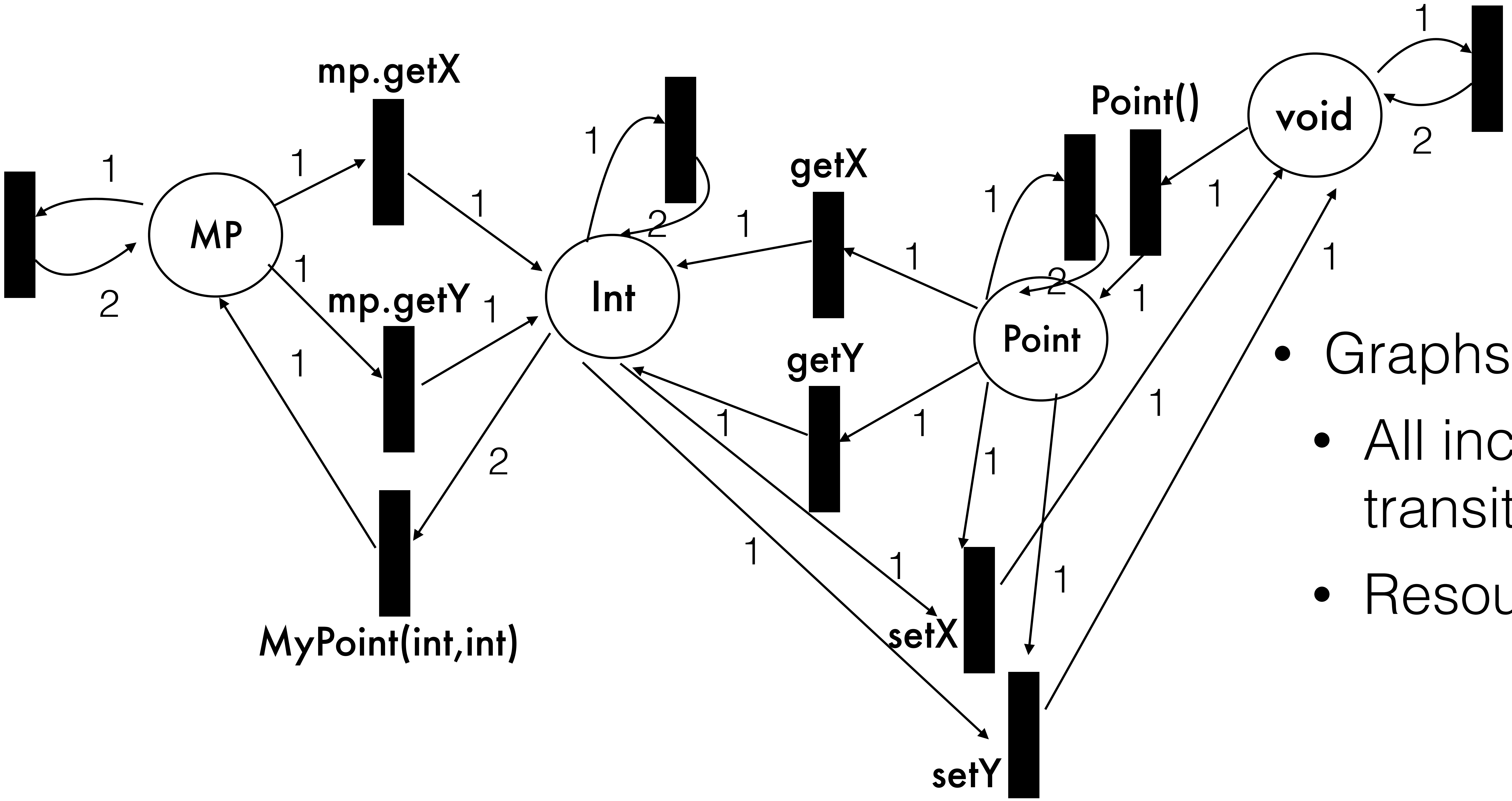
```
Point convert(Mypoint pt)
```

```
1) Clone-MP
2) mp.getX
3) mp.getY
4) Point()
5) Clone-Point
6) setX
7) Clone-Point
8) setY
```

Why a Petri Net and not a Graph?

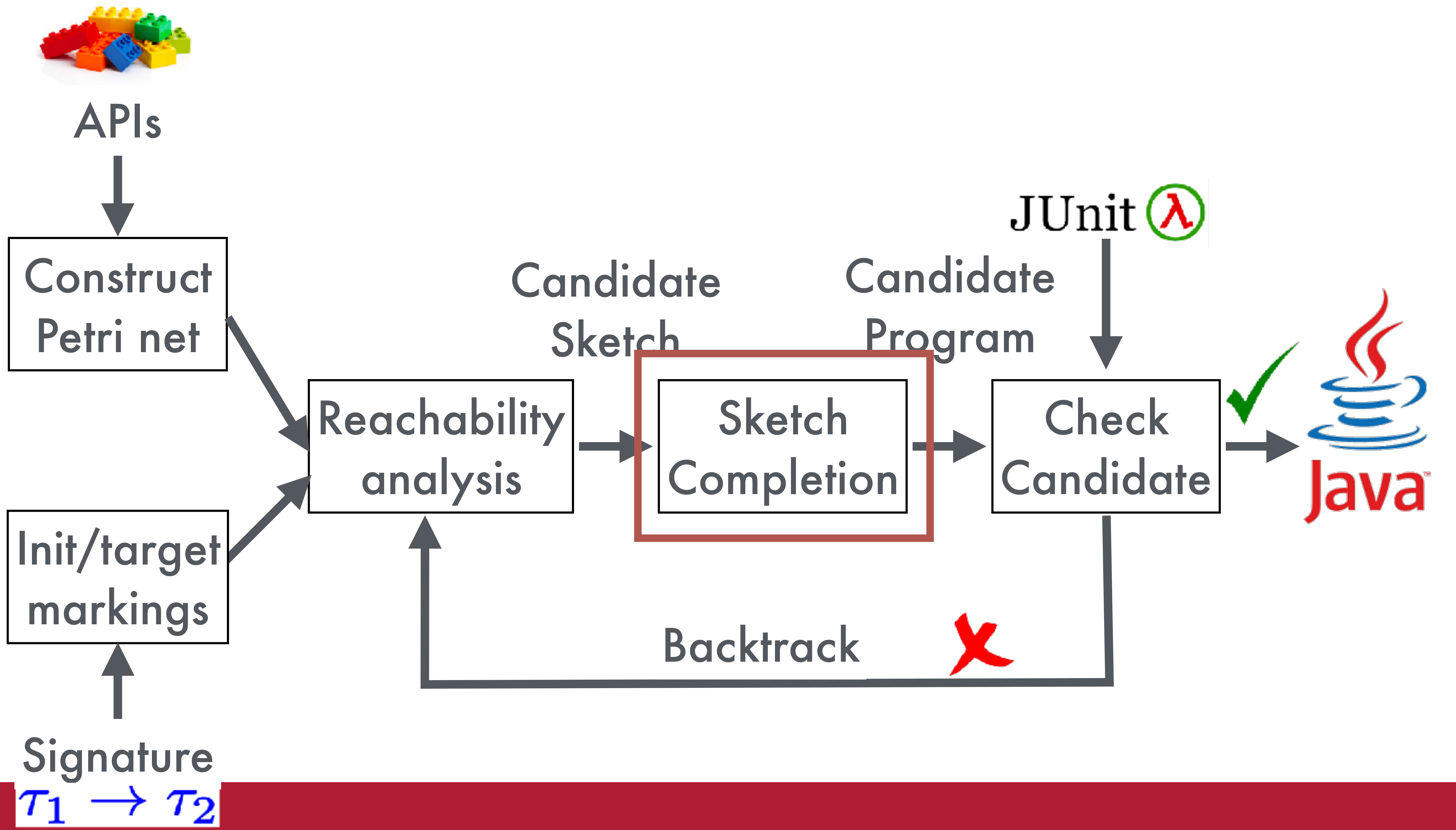


Why a Petri Net and not a Graph?



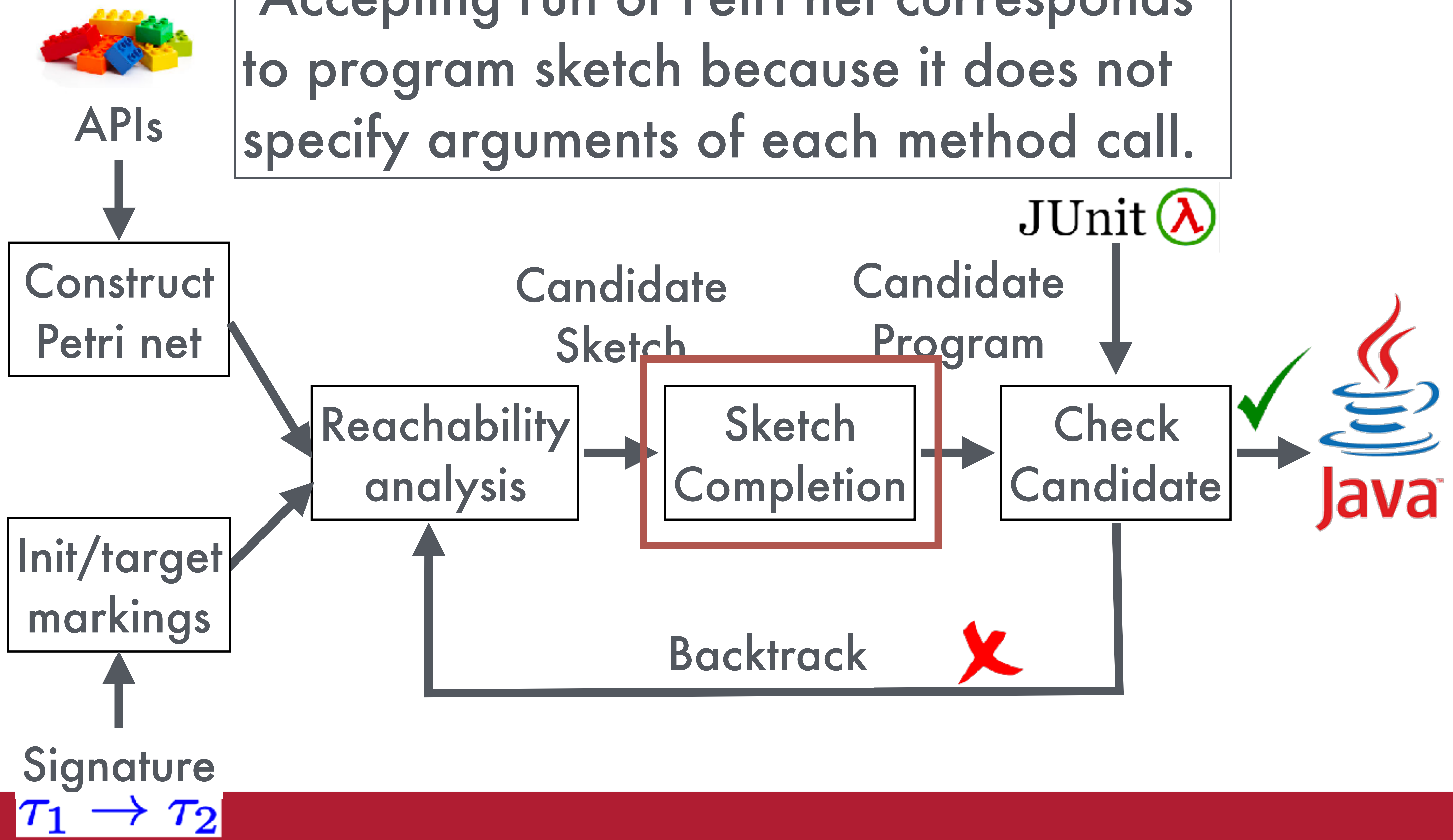
- Graphs do not support:
  - All incoming edges to a transition are part of the path
  - Resource consumption

Accepting Run as Program Sketch



Accepting Run as Program Sketch

Accepting run of Petri net corresponds to program sketch because it does not specify arguments of each method call.



Sketch Completion

1) Clone-MP
2) mp.getX
3) mp.getY
4) Point()
5) Clone-Point
6) setX
7) Clone-Point
8) setY

- Remove the Clone transitions

```
Point convert(Mypoint pt){



}
```

Sketch Completion

```
2) mp.getX
3) mp.getY
4) Point()
6) setX
8) setY
```

```
Point convert(Mypoint pt){



}
```

- What is the code with holes?

Sketch Completion

```
2) mp.getX
3) mp.getY
4) Point()
6) setX
8) setY
```

```
Point convert(Mypoint pt){

  int x = #1.getX();
  int y = #2.getY();
  Point p = new Point();
  p.setX(#3);
  p.setY(#4);
  return #5;

}
```

- What is the code with holes?
- Find the arguments that should be used in each hole such that the program type checks

Sketch Completion

```
2) mp.getX
3) mp.getY
4) Point()
6) setX
8) setY
```

```
Point convert(Mypoint pt){

  int x = pt.getX();
  int y = pt.getY();
  Point p = new Point();
  p.setX(#3);
  p.setY(#4);
  return p;

}
```

- What is the code with holes?
- Find the arguments that should be used in each hole such that the program type checks

Sketch Completion

```
2) mp.getX
3) mp.getY
4) Point()
6) setX
8) setY
```

```
Point convert(Mypoint pt){

  int x = pt.getX();
  int y = pt.getY();
  Point p = new Point();
  p.setX(y);
  p.setY(x);
  return p;

}
```

- What is the code with holes?
- Find the arguments that should be used in each hole such that the program type checks

Sketch Completion

```
2) mp.getX
3) mp.getY
4) Point()
6) setX
8) setY
```

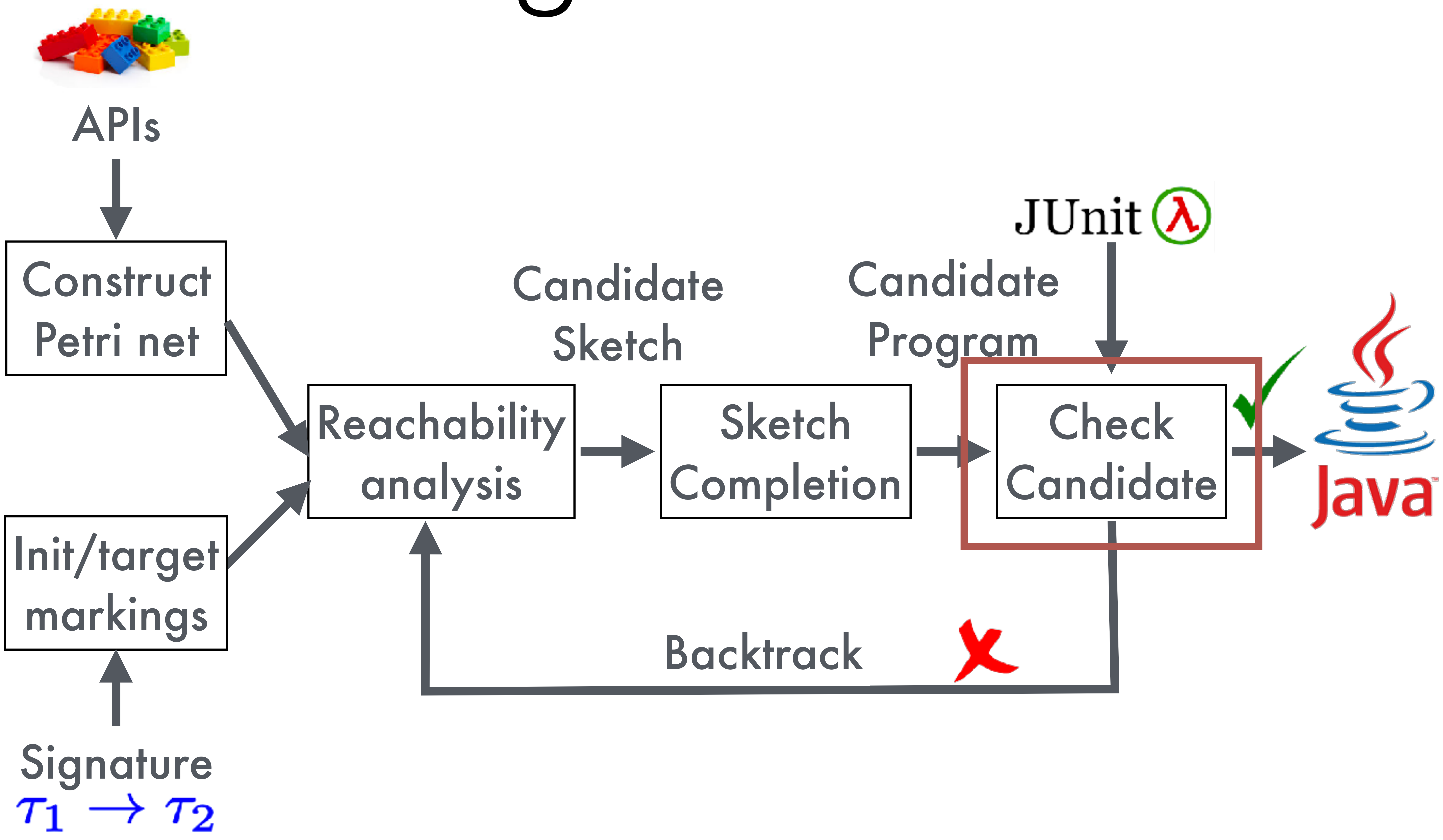
```
Point convert(Mypoint pt){

  int x = pt.getX();
  int y = pt.getY();
  Point p = new Point();
  p.setX(x);
  p.setY(y);
  return p;

}
```

- What is the code with holes?
- Find the arguments that should be used in each hole such that the program type checks

Checking the Candidate



Testing

```
Point convert(Mypoint pt){

  int x = pt.getX();
  int y = pt.getY();
  Point p = new Point();
  p.setX(x);
  p.setY(y);
  return p;

}
```

- Test case to check the conversion:

Testing

```
Point convert(Mypoint pt){

  int x = pt.getX();
  int y = pt.getY();
  Point p = new Point();
  p.setX(x);
  p.setY(y);
  return p;

}
```

- Test case to check the conversion:

```
bool test(){

  MyPoint mp = new MyPoint(1,2);
  Point p = convert(mp);
  return (p.getX() == 1 &&
          p.getY() == 2);

}
```

For more information

SyPet

Program synthesis tool for Java libraries that automatically constructs programs by composing APIs.

GITHUB DOWNLOAD

https://utopia-group.github.io/sypet/

Outline

- Introduction to Syntax-Guided Synthesis (SyGus)
- Synthesis of Java code
- Conflict-driven Synthesis

How do Program Synthesizers Work?

Enumerative Search

Stochastic Search

Constraint Solving

How do Program Synthesizers Work?

Enumerative Search

Stochastic Search

Constraint Solving

Can we **learn** from **past mistakes**?

Learning from Mistakes

• Input: x= [1,2,3] • Output: y= [1,2]

- λx.map(x,…)

This program will not satisfy the input-output specification since map **preserves** the size of the list!

Learning from Mistakes

• Input: x= [1,2,3] • Output: y= [1,2]

•λx.map(x,…) $\xrightarrow{\text{Learning}}$ •λx.**f**(x,…)

This program will not satisfy the input-output specification since map **preserves** the size of the list!

If *f* **preserves** or **increases** the size of the list then *f* will also be infeasible!

Learning from Mistakes

• Input: x= [1,2,3] • Output: y= [1,2]

$\bullet \lambda x.\text{map}(x,\ldots) \xrightarrow{\text{Learning}} \bullet \lambda x.\mathbf{f}(x,\ldots)$

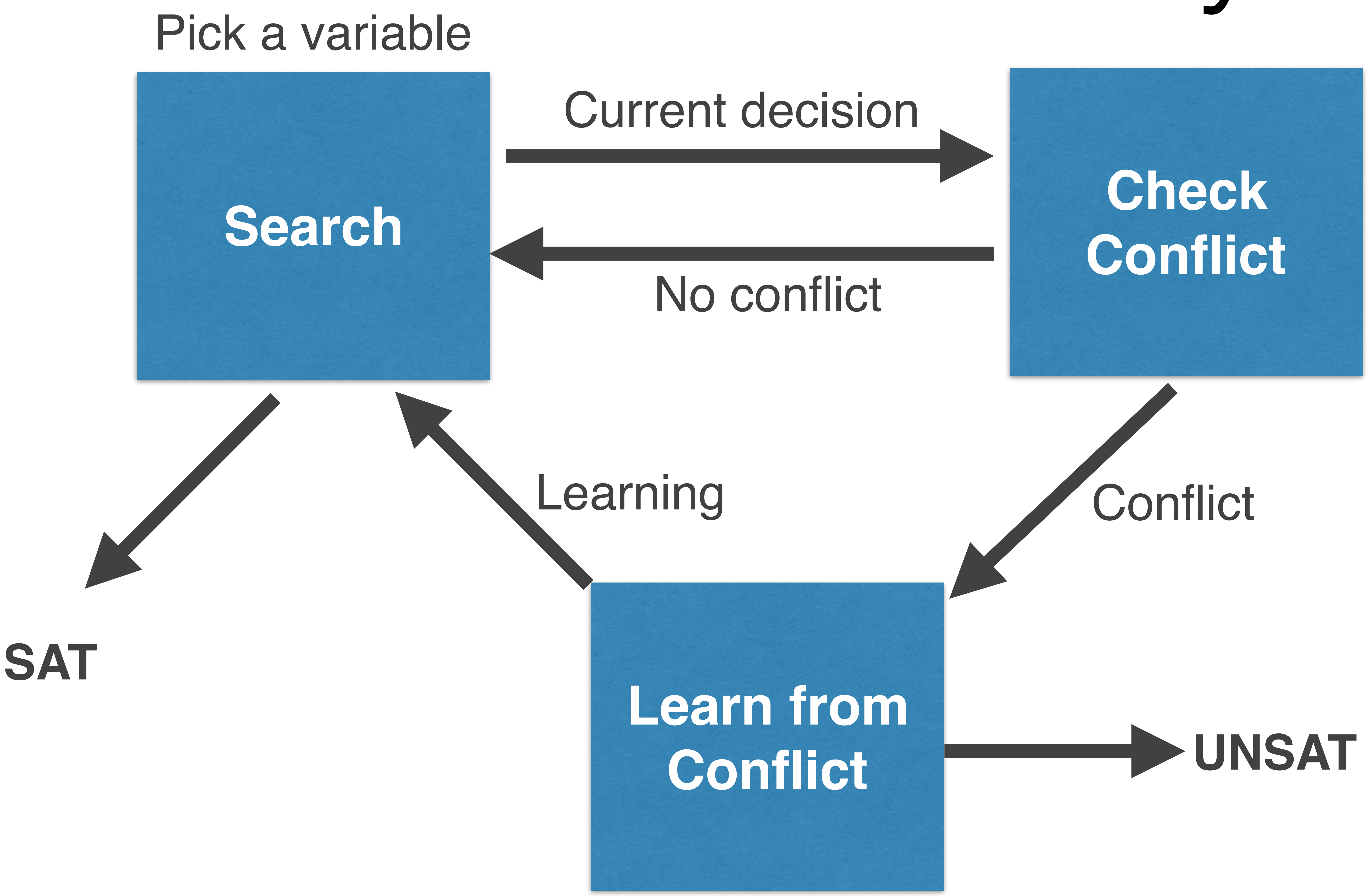
This program will not satisfy the input-output specification since map **preserves** the size of the list!

If *f* **preserves** or **increases** the size of the list then *f* will also be infeasible!

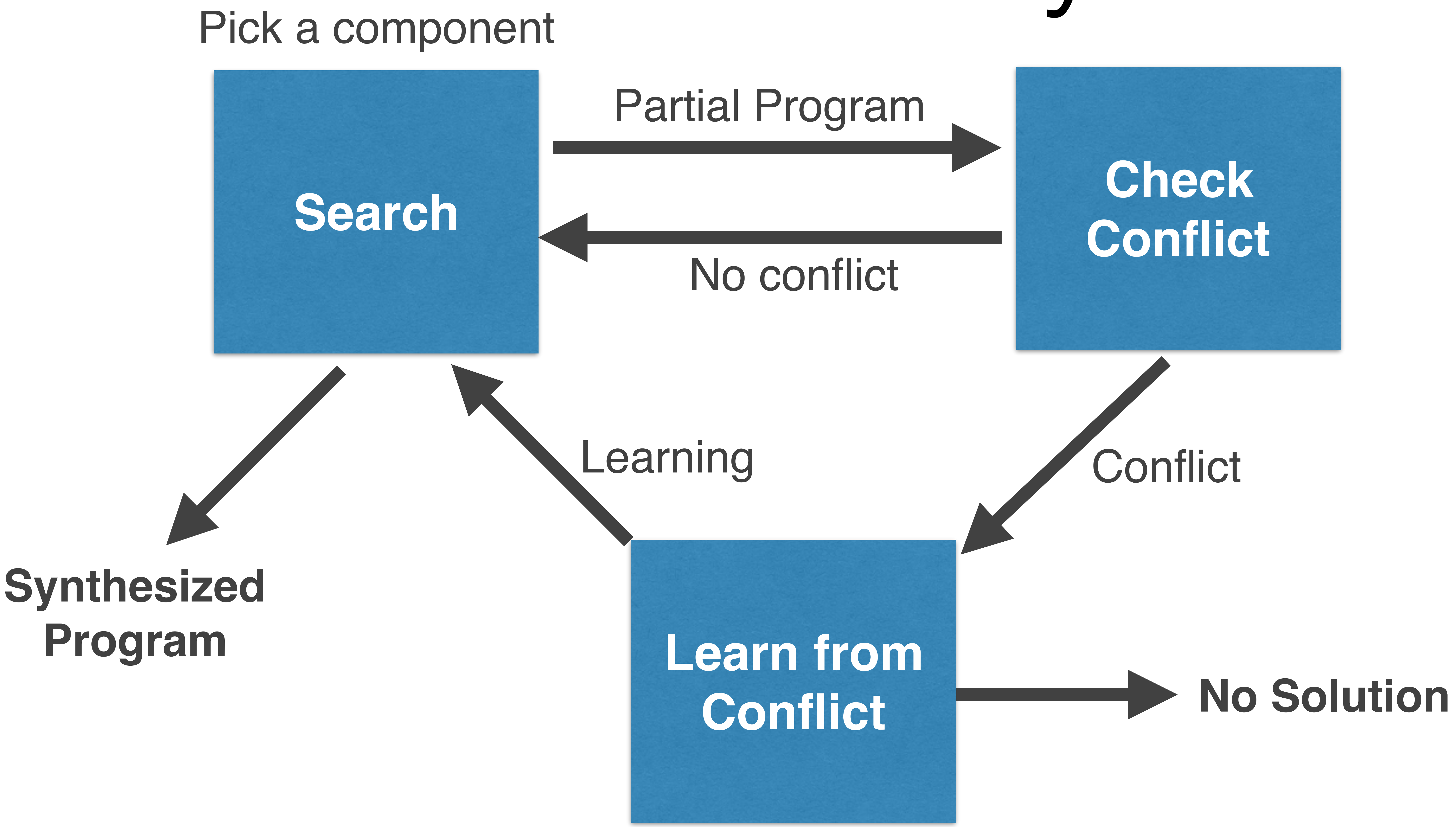
Can we **learn** from **past mistakes**?

If λx.map(x,…) in an infeasible program then we can rule out many other erroneous programs such as λx.reverse(x) or λx.sort(x)

From SAT solvers to Synthesis



Conflict-Driven Synthesis



- Program Synthesis using **Conflict-Driven Learning**. PLDI'18

Running Example

- Compute the **scores** of the **best k** teams of a soccer league

Running Example

- Compute the **scores** of the **best k** teams of a soccer league

- **computeKsum:: List -> Int -> Int**
- computeKSum x1 x2 =
- Inputs:
  - x1 = [49, 62, 82, 54, 76]
  - x2 = 2
- Output:
  - 158 (82 + 76)

Sample Domain Specific Language

- **computeKsum:: List -> Int -> Int**
- computeKSum ([49, 62, 82, 54, 76] , 2) = 158

N->	0 | … | 10 | X | last(L) | head(L) | sum(L) | maximum(L) | minimum(L)
L->	take(L,N) | filter(L,T) | sort(L) | reverse(L) | X
T->	geqz | leqz | eqz

Complete Programs

- **computeKsum:: List -> Int -> Int**
- computeKSum x1 x2 =

- - sort x1 in ascending order

L1 <- **sort** x1

- - L2 is x1 in descending order

L2 <- **reverse** L1

- - Take L2's first x2 entries

L3 <- **take** L2 x2

- - Compute sum of all elements in L3

sum L3

Complete Programs

- **computeKsum:: List -> Int -> Int**
- computeKSum x1 x2 =

- - sort x1 in ascending order

L1 <- **sort** x1

- - L2 is x1 in descending order

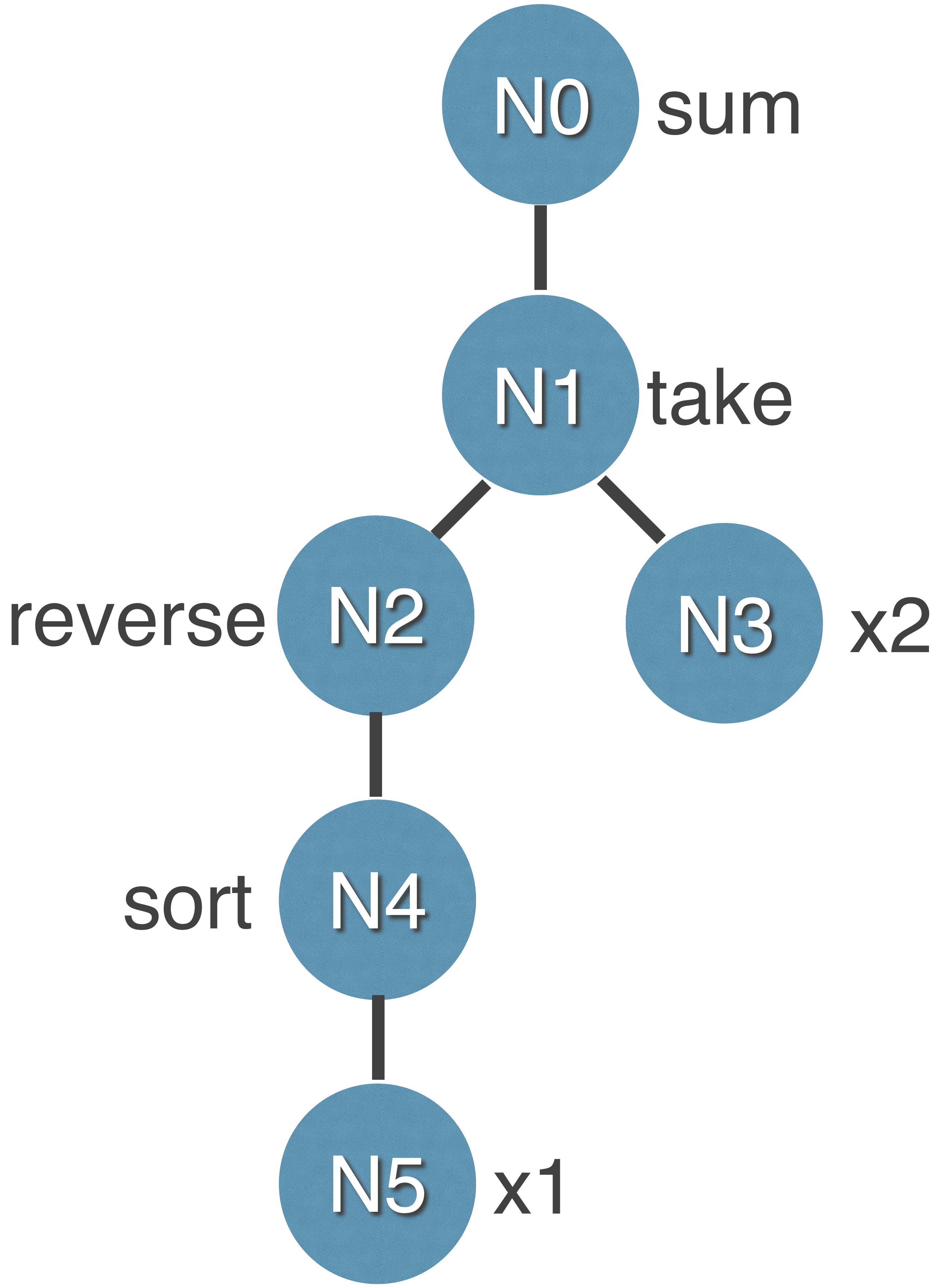
L2 <- **reverse** L1

- - Take L2's first x2 entries

L3 <- **take** L2 x2

- - Compute sum of all elements in L3

sum L3



- **Program:**
  - Abstract Syntax Trees (ASTs)

Partial Programs

- **computeKsum:: List -> Int -> Int**
- computeKSum x1 x2 =

- - sort x1 in ascending order

L1 <- **sort** x1

- - L2 is x1 in descending order

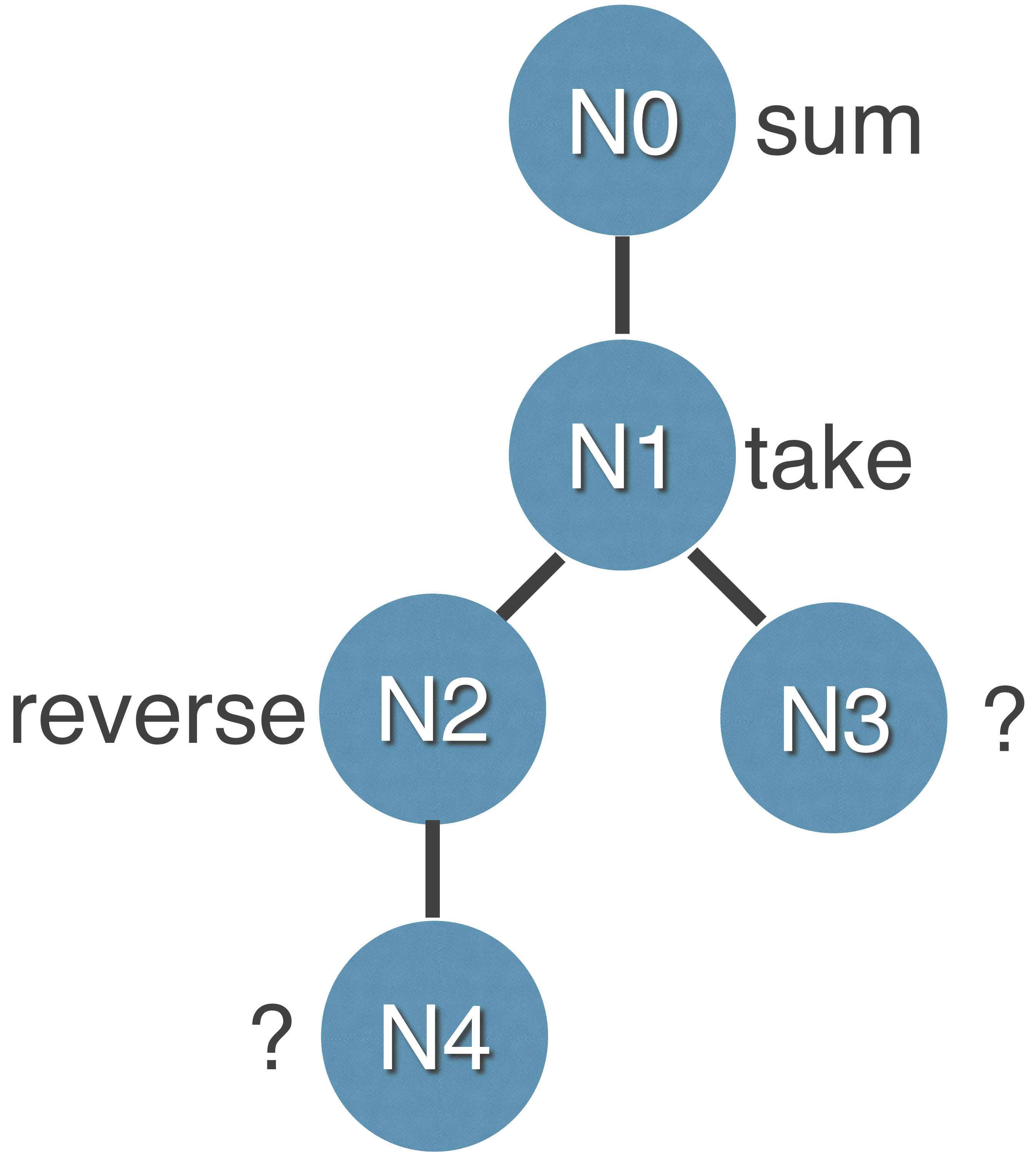
L2 <- **reverse** L1

- - Take L2's first x2 entries

L3 <- **take** L2 x2

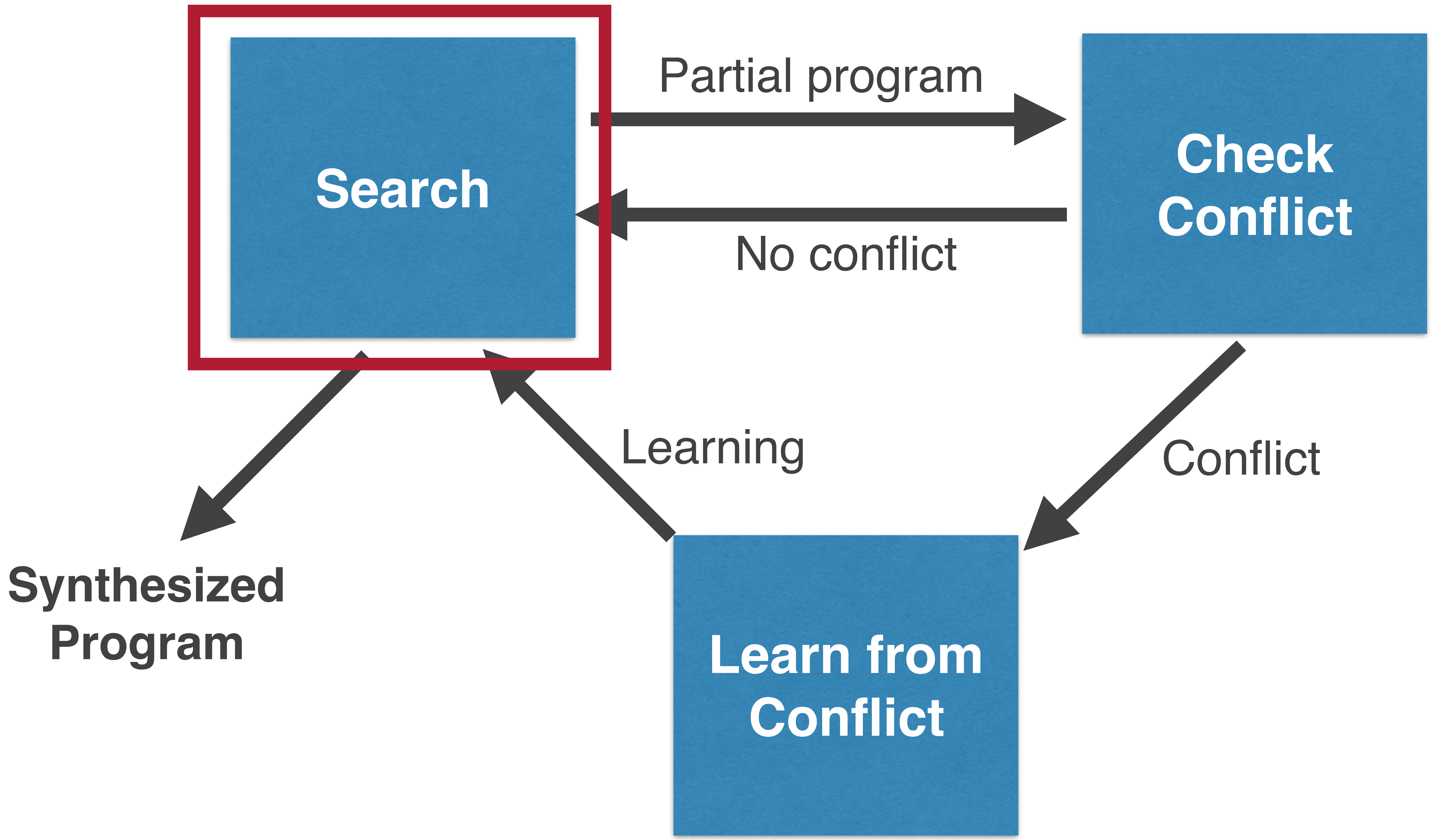
- - Compute sum of all elements in L3

sum L3



- **Partial Program:**
  - Abstract Syntax Trees (ASTs)
  - Some nodes are unknown

Conflict-Driven Synthesis



Guiding the Search

- **Goal:** Choose a component to each node of the program

Guiding the Search

- **Goal:** Choose a component to each node of the program

- Machine learning (ML):
  - N-grams
  - Neural Networks

- Learning from large corpora

Guiding the Search

- **Goal:** Choose a component to each node of the program

- Machine learning (ML):
  - N-grams
  - Neural Networks
- Learning from large corpora

- **computeKsum:: List -> Int -> Int**
- Inputs: [49, 62, 82, 54, 76] , 2
- Output: 158

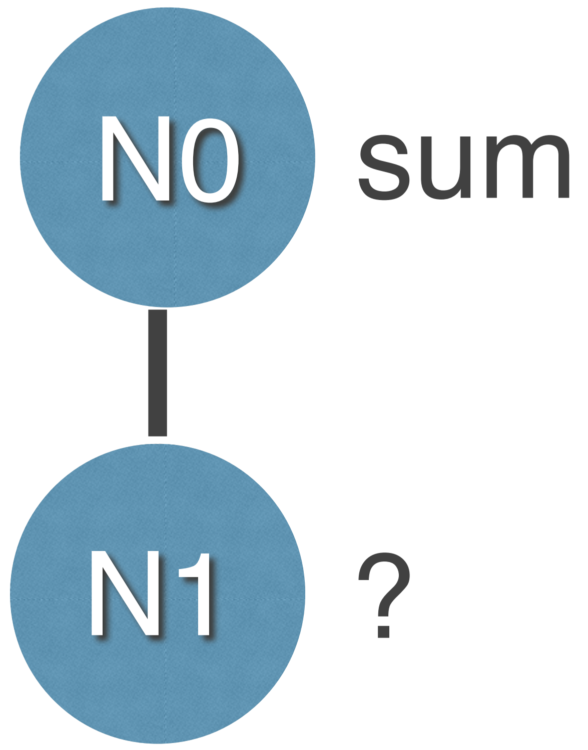
N0 ?

Guiding the Search

- **Goal:** Choose a component to each node of the program

- Machine learning (ML):
  - N-grams
  - Neural Networks
- Learning from large corpora

- computeKsum:: List -> Int -> Int
- Inputs: [49, 62, 82, 54, 76] , 2
- Output: 158

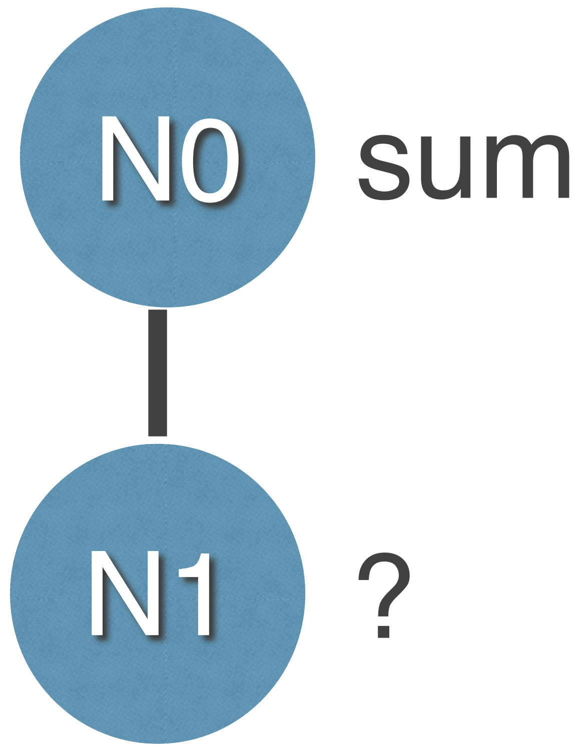


Guiding the Search

- **Goal:** Choose a component to each node of the program

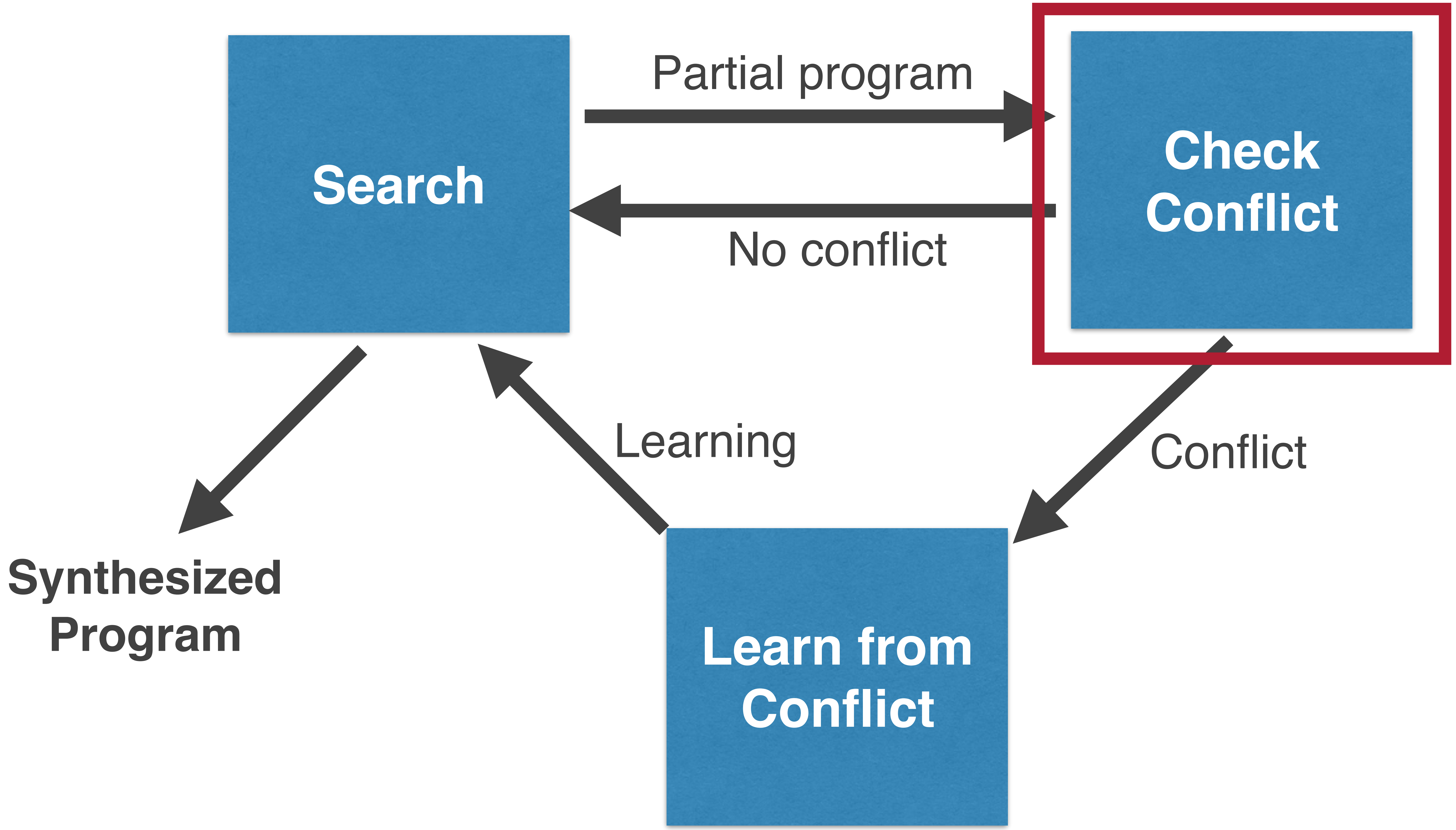
- Machine learning (ML):
  - N-grams
  - Neural Networks
- Learning from large corpora

- computeKsum:: List -> Int -> Int
- Inputs: [49, 62, 82, 54, 76] , 2
- Output: 158



We can only check if the program is correct once we have a **complete program**!

Conflict-Driven Synthesis



Imprecise Specifications

- **Goal:** Prune the search space with imprecise specifications

Imprecise Specifications

- **Goal:** Prune the search space with imprecise specifications
  - Precisely describing a component can be challenging

Imprecise Specifications

- **Goal:** Prune the search space with imprecise specifications
  - Precisely describing a component can be challenging
  - Use simple properties that **over-approximate** the behavior of a component:
    - List properties: size, maximum, etc.

Imprecise Specifications

- **Goal:** Prune the search space with imprecise specifications
  - Precisely describing a component can be challenging
  - Use simple properties that **over-approximate** the behavior of a component:
    - List properties: size, maximum, etc.

L-> filter(L,T)	y.size <= x1.size y.max <= x1.max
L -> take(L,N)	y.size <= x1.size ; y.size = x2 y.max <= x1.max
N -> head(L)	y <= x1.max

Pruning the Search Space

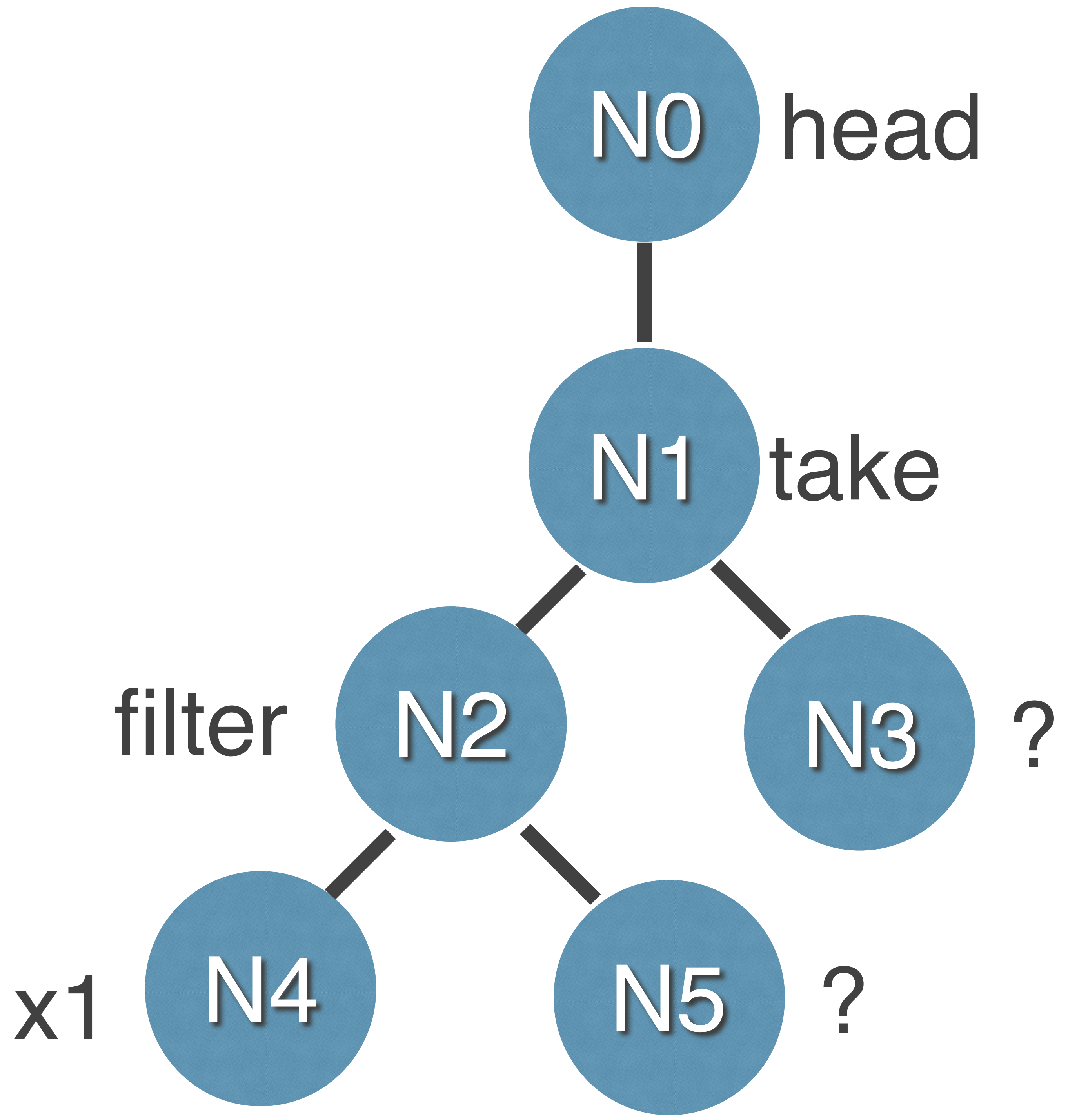
N0 -> (head)

N1 -> (take)

N2 -> (filter)

N4 -> (x1)

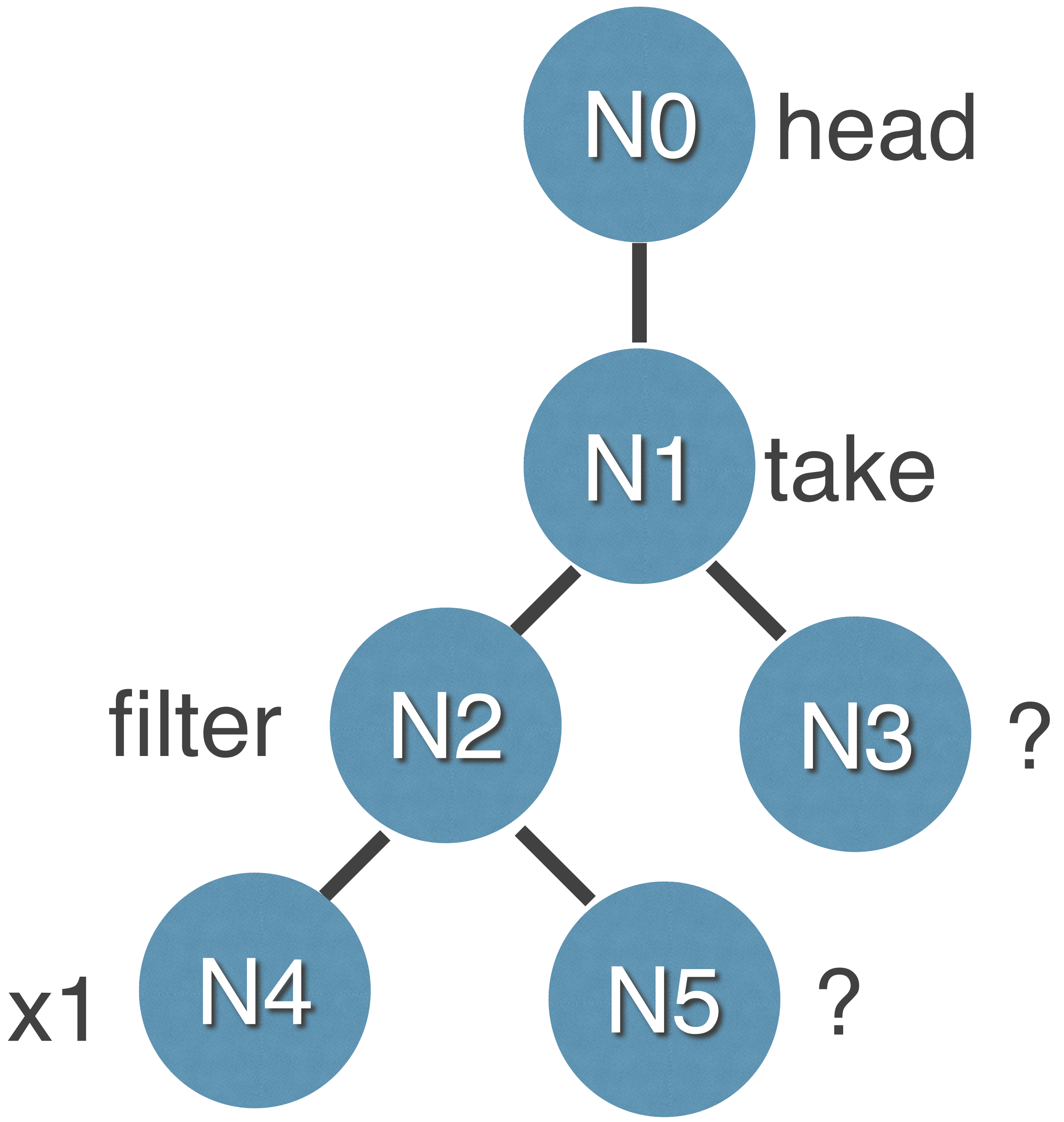
- x1 = [49, 62, 82, 54, 76]
- y = 158



Pruning the Search Space

N0 -> (head)	y <= n1.max
N1 -> (take)	n1.max <= n2.max n1.size <= n2.size n1.size = n3
N2 -> (filter)	n2.size <= n4.size n2.max <= n4.max
N4 -> (x1)	n4 = x1

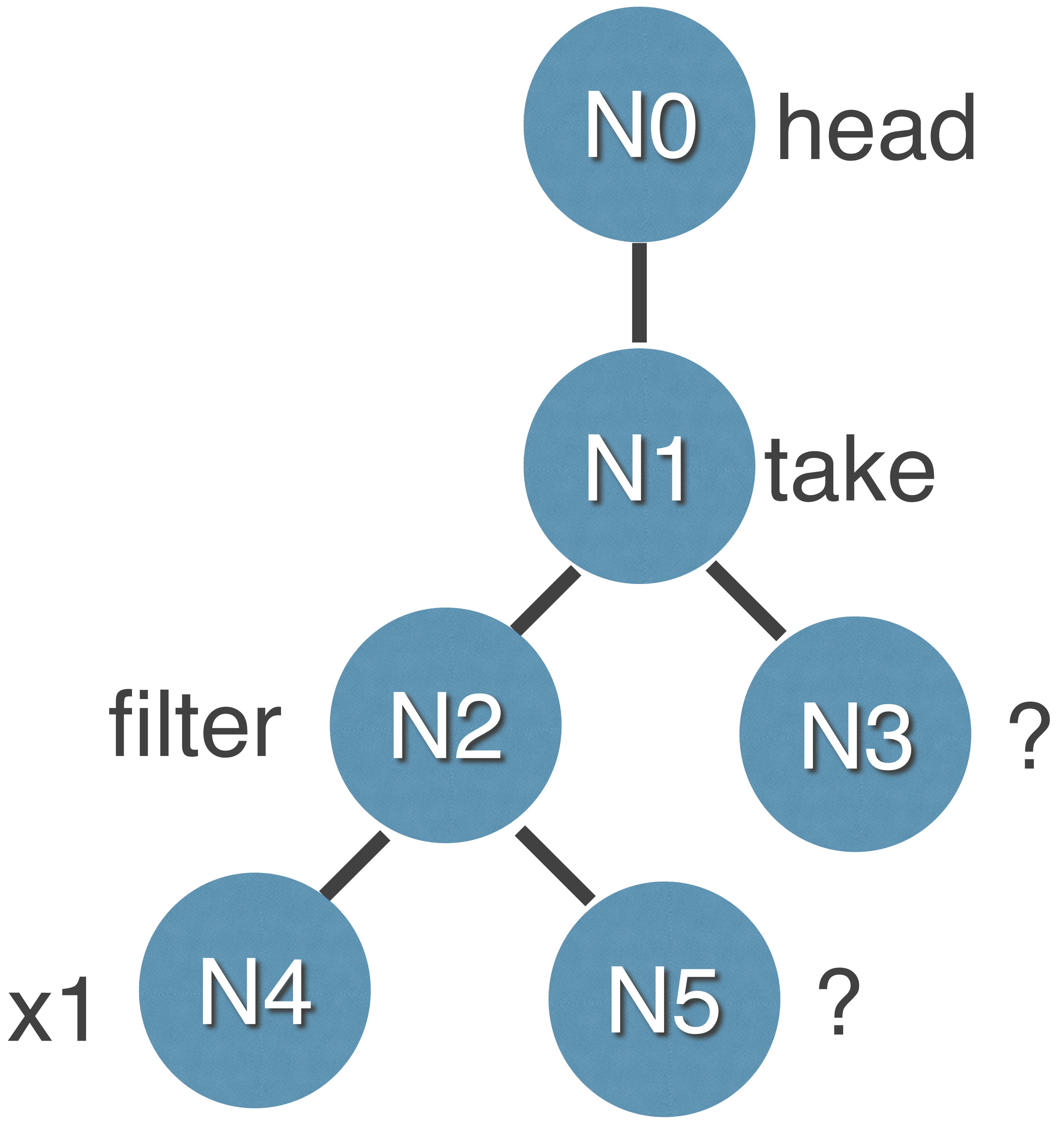
- x1 = [49, 62, 82, 54, 76]
- y = 158



Pruning the Search Space

- x1 = [49, 62, 82, 54, 76]
- y = 158

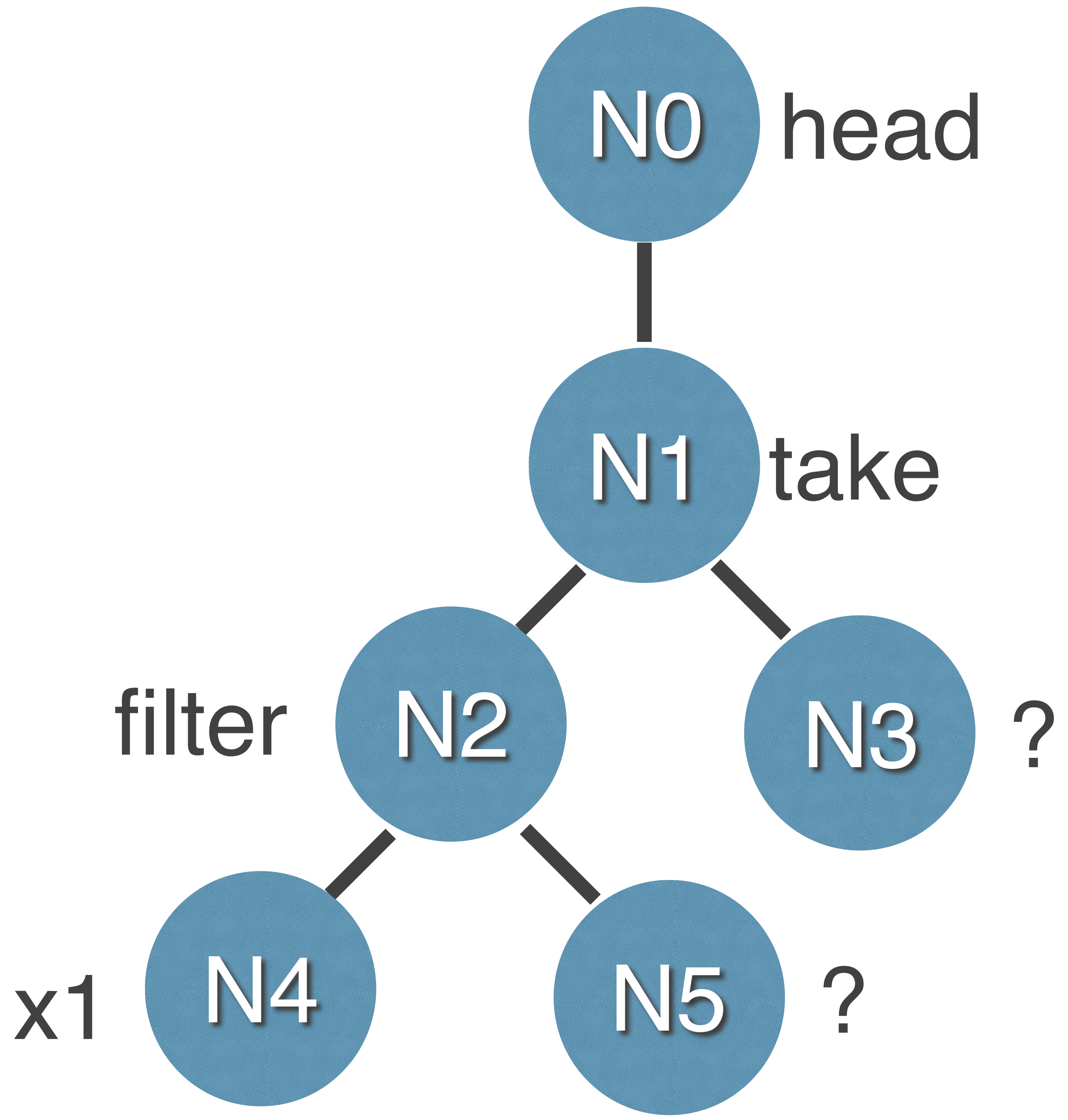
N0 -> (head)	**y <= n1.max** **(158 <= 82)**
N1 -> (take)	**n1.max <= n2.max** n1.size <= n2.size n1.size = n3
N2 -> (filter)	n2.size <= n4.size **n2.max <= n4.max**
N4 -> (x1)	**n4 = x1**



Pruning the Search Space

- x1 = [49, 62, 82, 54, 76]
- y = 158

A partial program represents **many** complete programs!

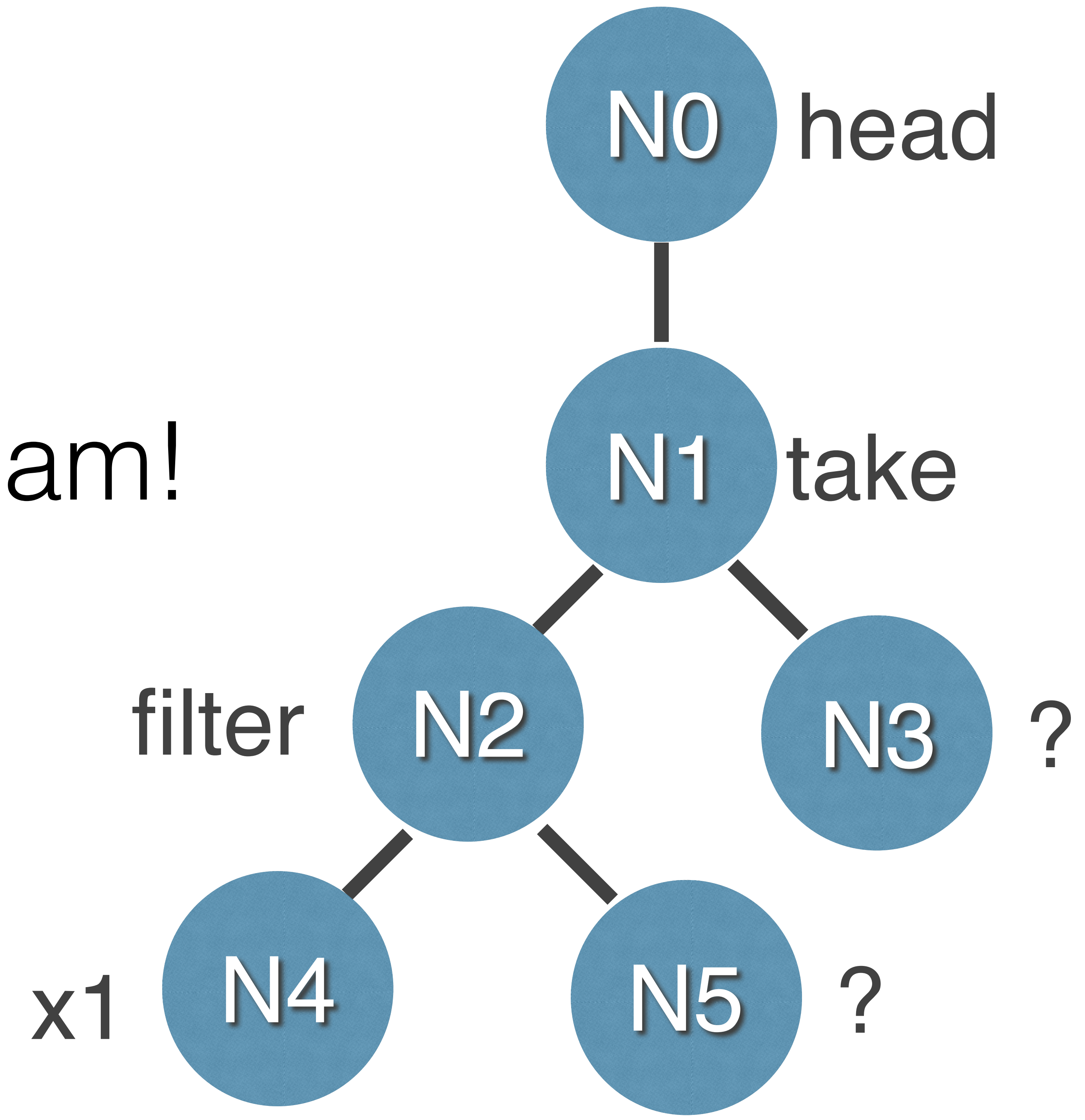


Pruning the Search Space

- x1 = [49, 62, 82, 54, 76]
- y = 158

A partial program represents **many** complete programs!

We are only pruning **one** partial program!



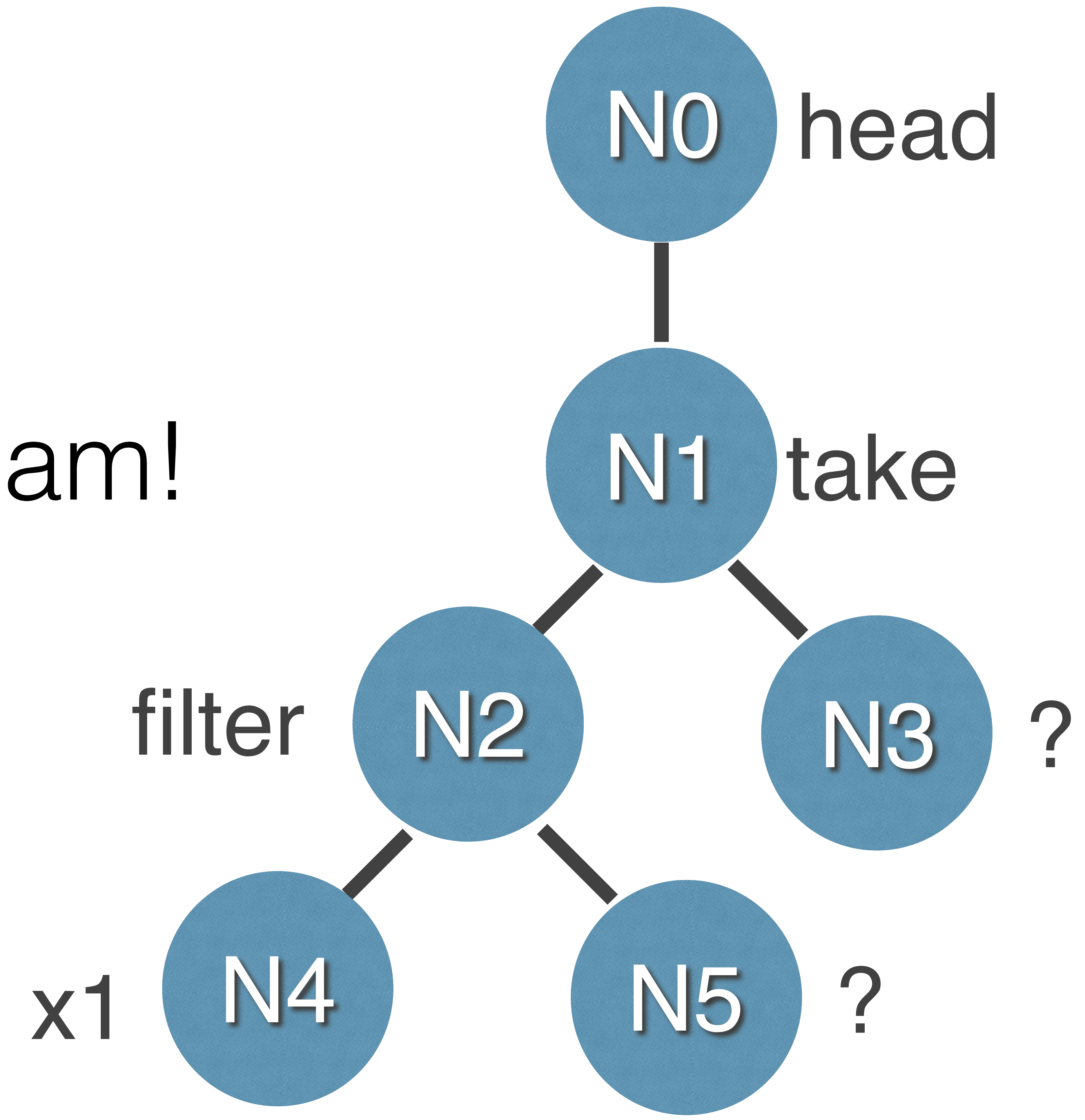
Pruning the Search Space

- x1 = [49, 62, 82, 54, 76]
- y = 158

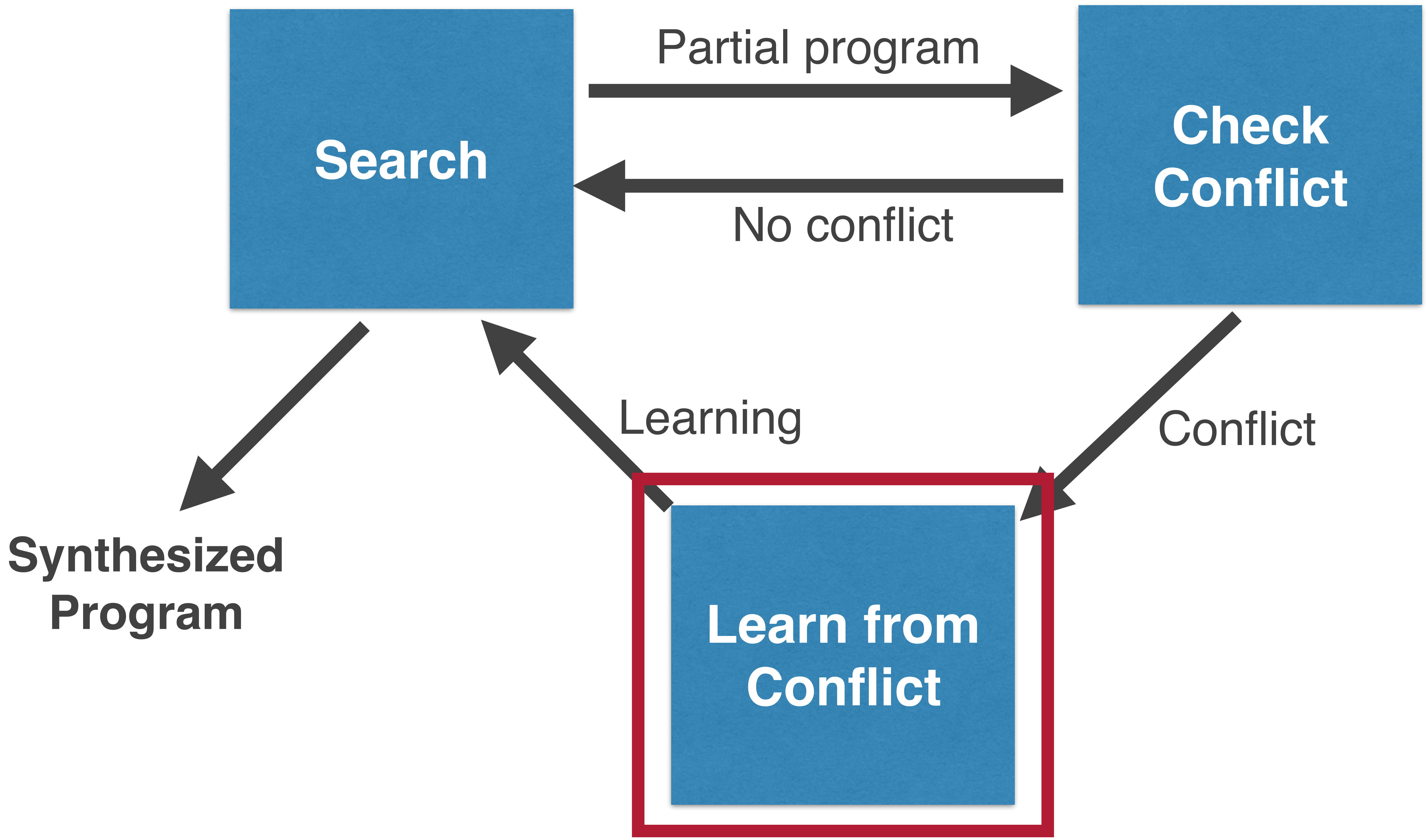
A partial program represents **many** complete programs!

We are only pruning **one** partial program!

Can we prune **equivalent infeasible** partial programs?



Conflict-Driven Synthesis



Learning from Mistakes

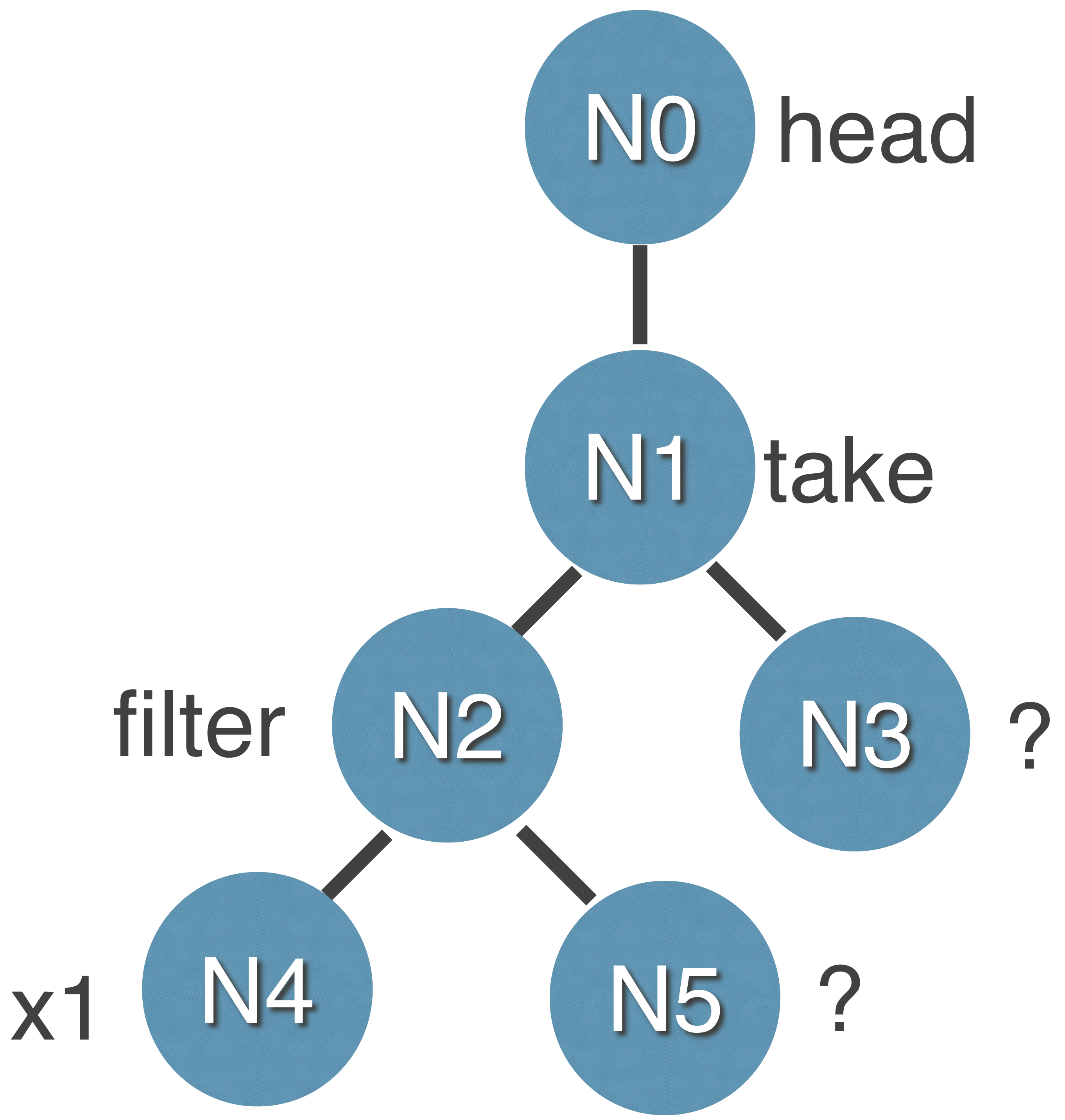
- **Goal:** : Learn equivalent infeasible partial programs

Learning from Mistakes

- **Goal:** : Learn equivalent infeasible partial programs

Equivalent modulo conflict:

- Two components X and X' are **equivalent modulo conflict** at node N if replacing X with X' leads to the same conflict



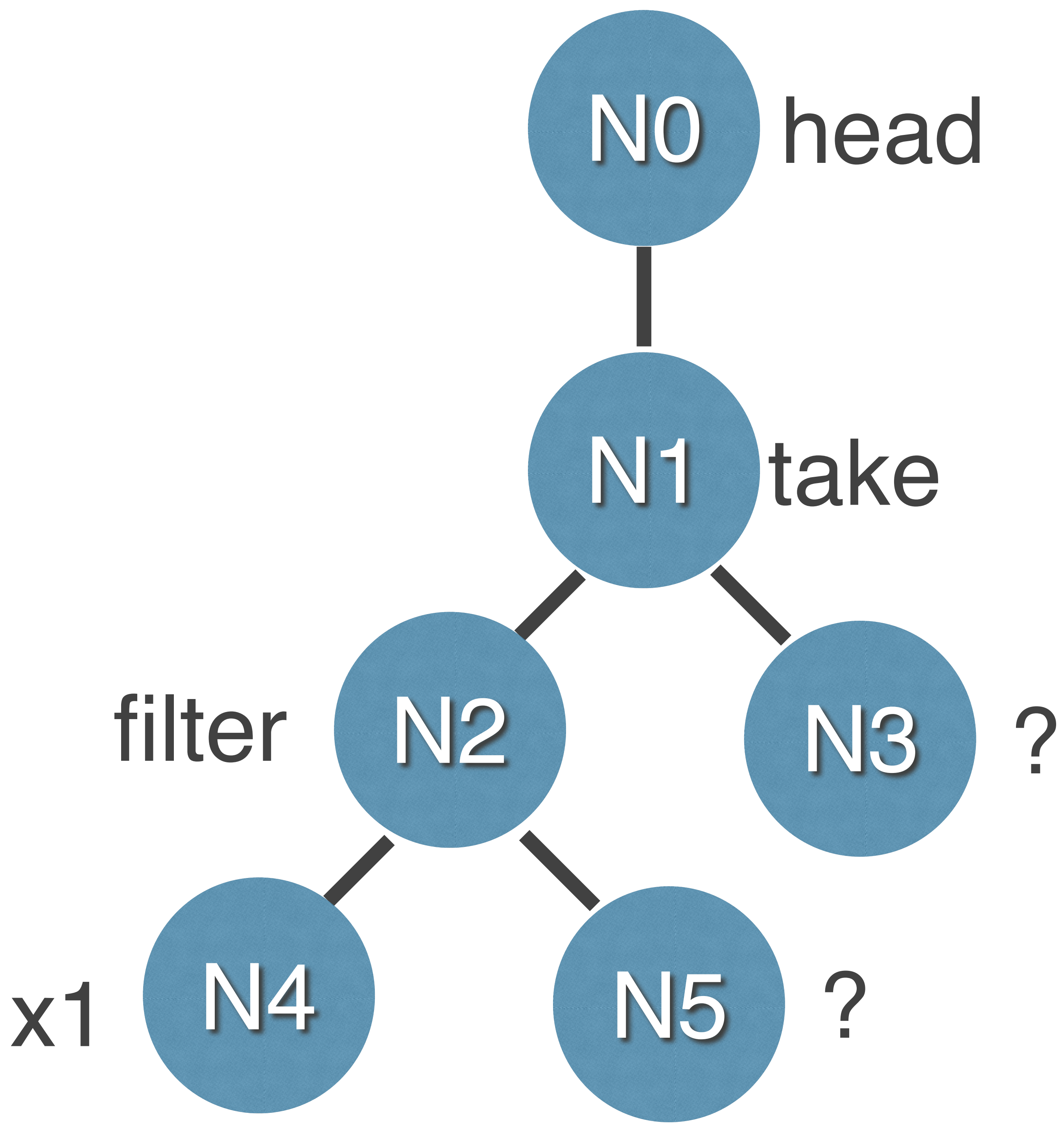
Learning from Mistakes

- **Goal:** : Learn equivalent infeasible partial programs

Equivalent modulo conflict:

- Two components X and X’ are **equivalent modulo conflict** at node N if replacing X with X’ leads to the same conflict

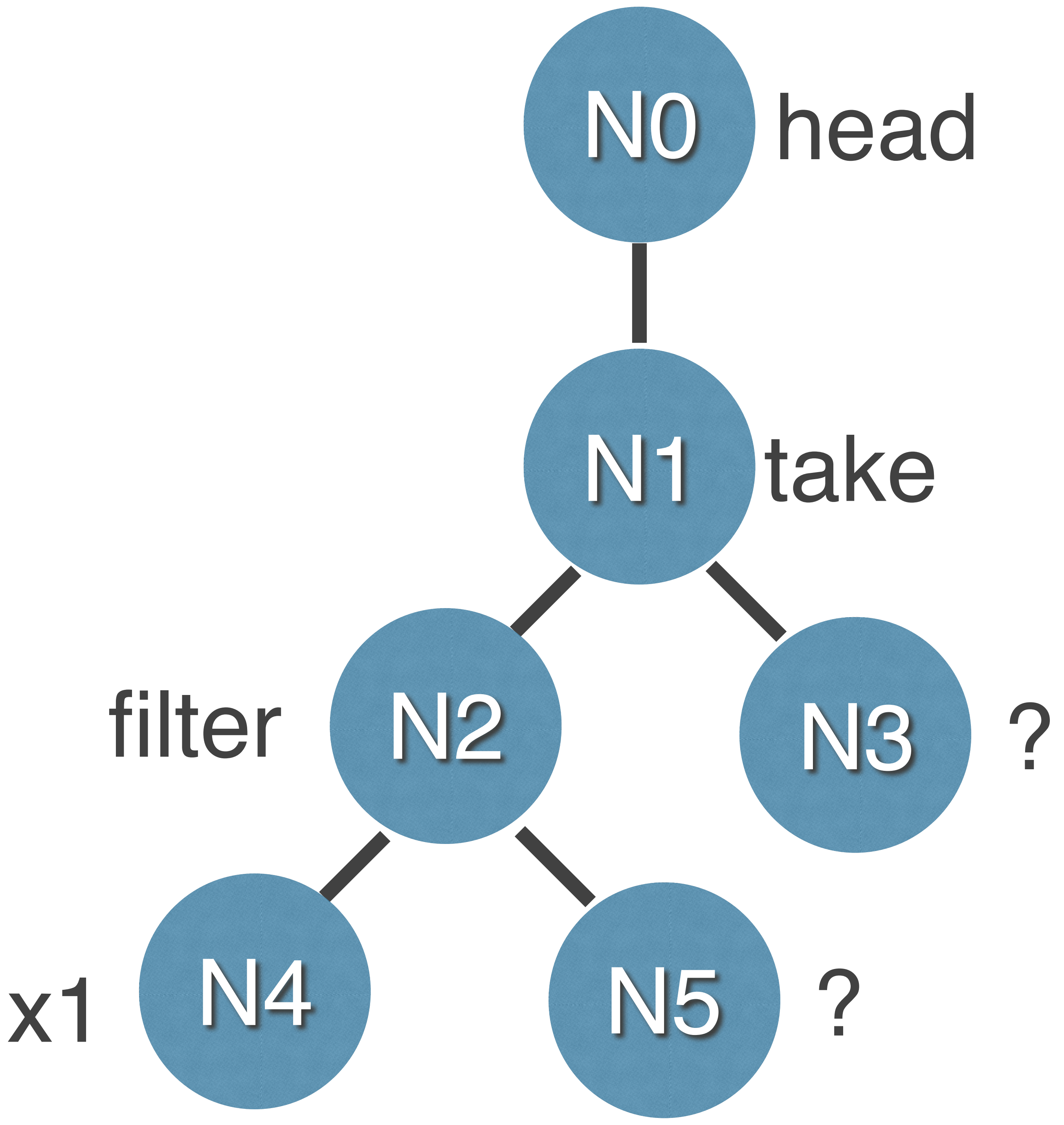
How to detect equivalent modulo conflict components?



Learning from Mistakes

How to detect equivalent modulo conflict components?

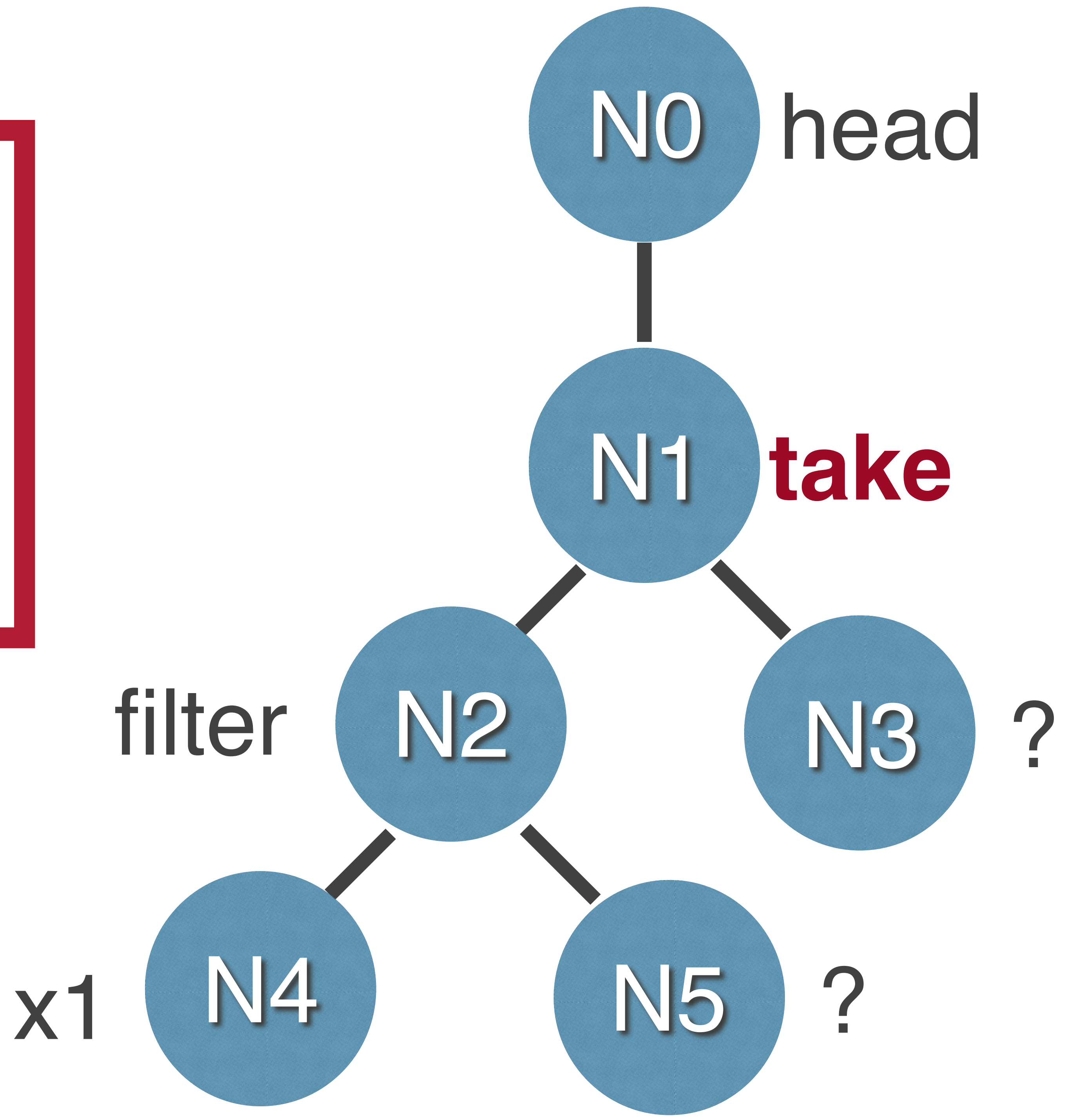
N0 -> (head)	**y <= n1.max**
N1 -> (take)	**n1.max <= n2.max** n1.size < n2.size n1.size = n3
N2 -> (filter)	n2.size < n4.size **n2.max <= n4.max**
N4 -> (x1)	**n4 = x1**



Learning from Mistakes

How to detect equivalent modulo conflict components?

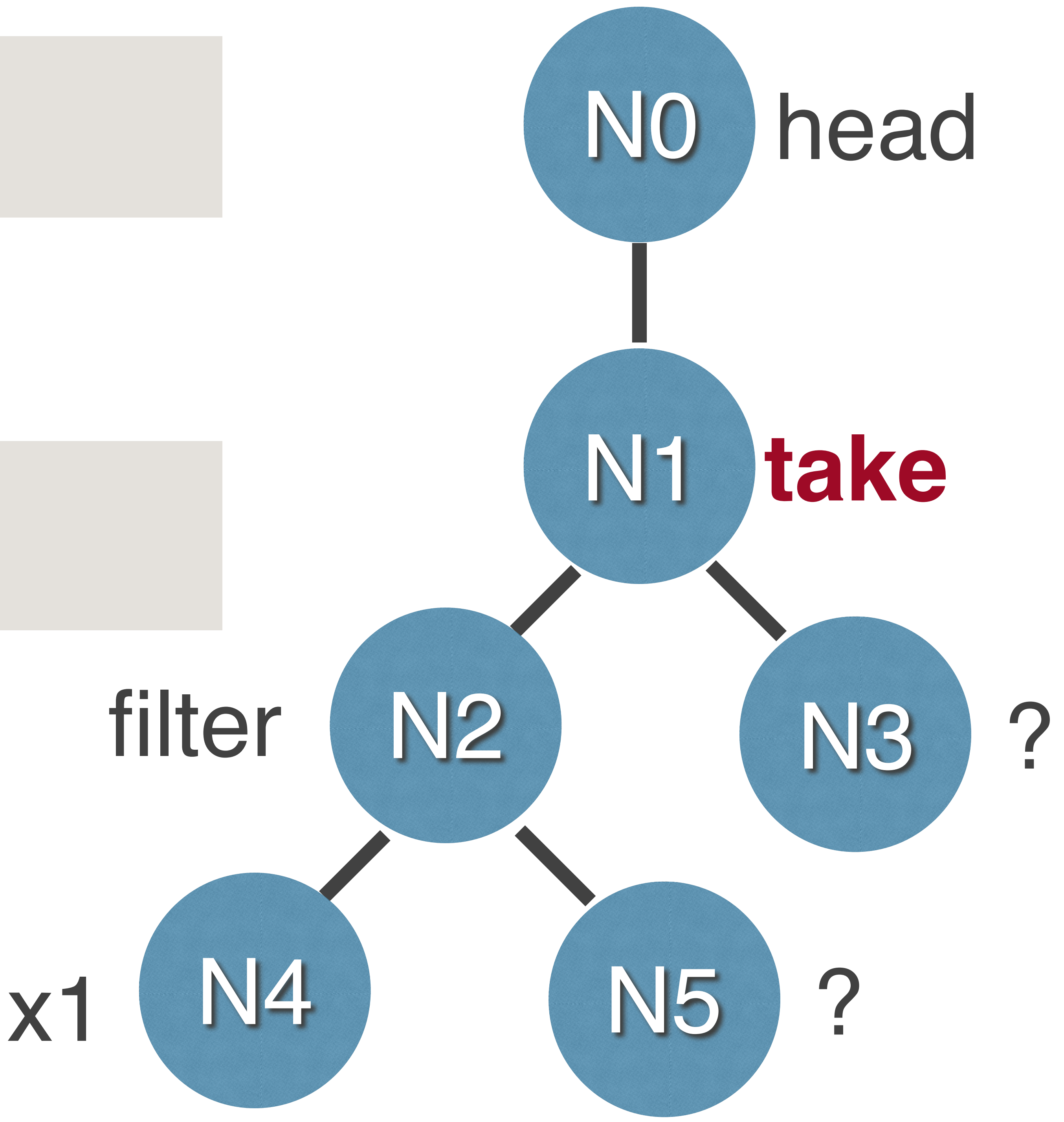
N0 -> (head)	**$y <= n1.max$**
N1 -> (take)	**$n1.max <= n2.max$** $n1.size < n2.size$ $n1.size = n3$
N2 -> (filter)	$n2.size < n4.size$ **$n2.max <= n4.max$**
N4 -> (x1)	**$n4 = x1$**



Learning from Mistakes

How to detect equivalent modulo conflict components?

Take	**n1.max <= n2.max**
Sort	n1.max = n2.max
Reverse	n1.max = n2.max
Filter	n1.max <= n2.max



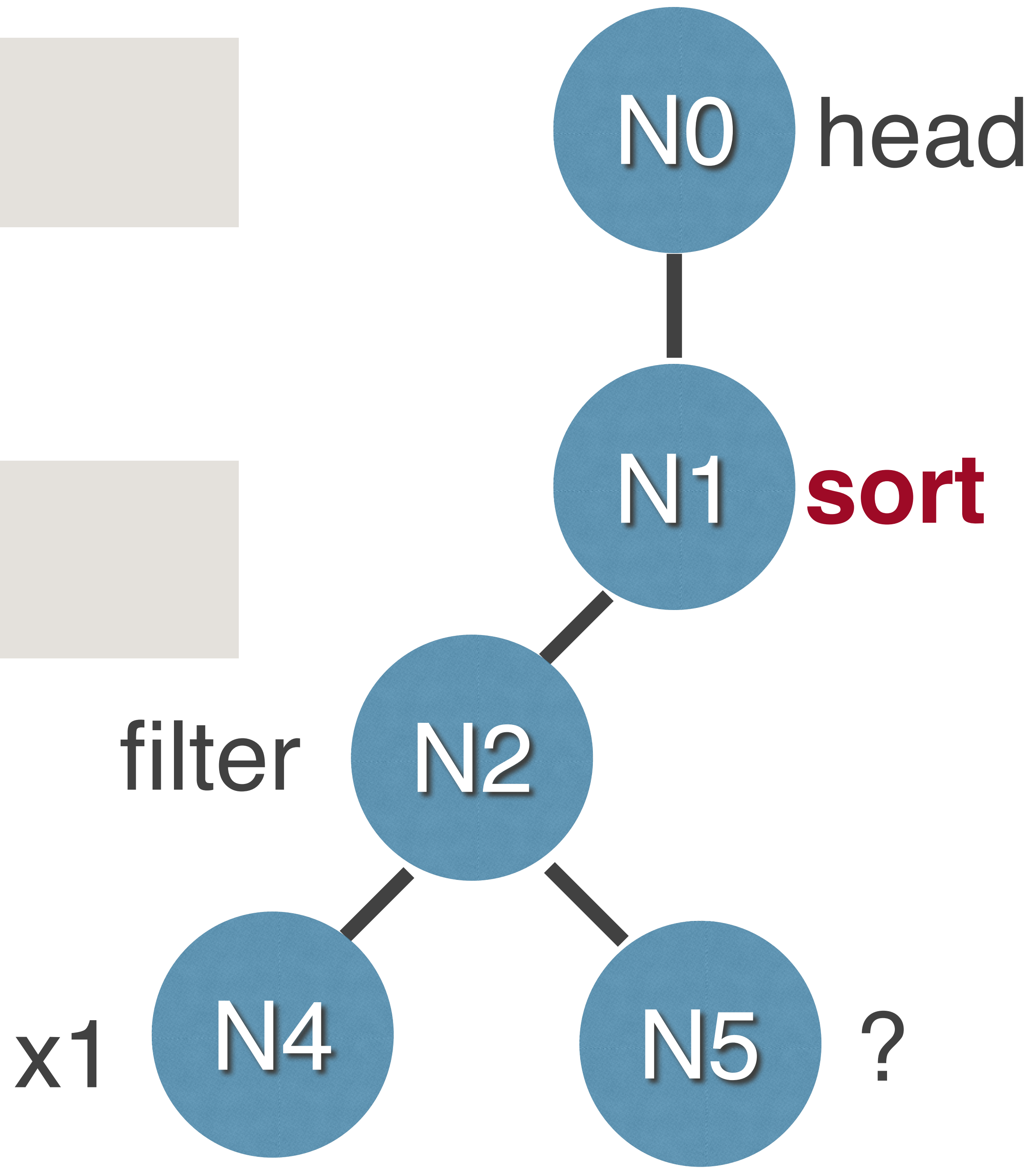
If the specification of X’ implies X then X ≡ X’:

- take ≡ sort ≡ reverse ≡ filter

Learning from Mistakes

How to detect equivalent modulo conflict components?

Take	n1.max <= n2.max
Sort	**n1.max = n2.max**
Reverse	n1.max = n2.max
Filter	n1.max <= n2.max



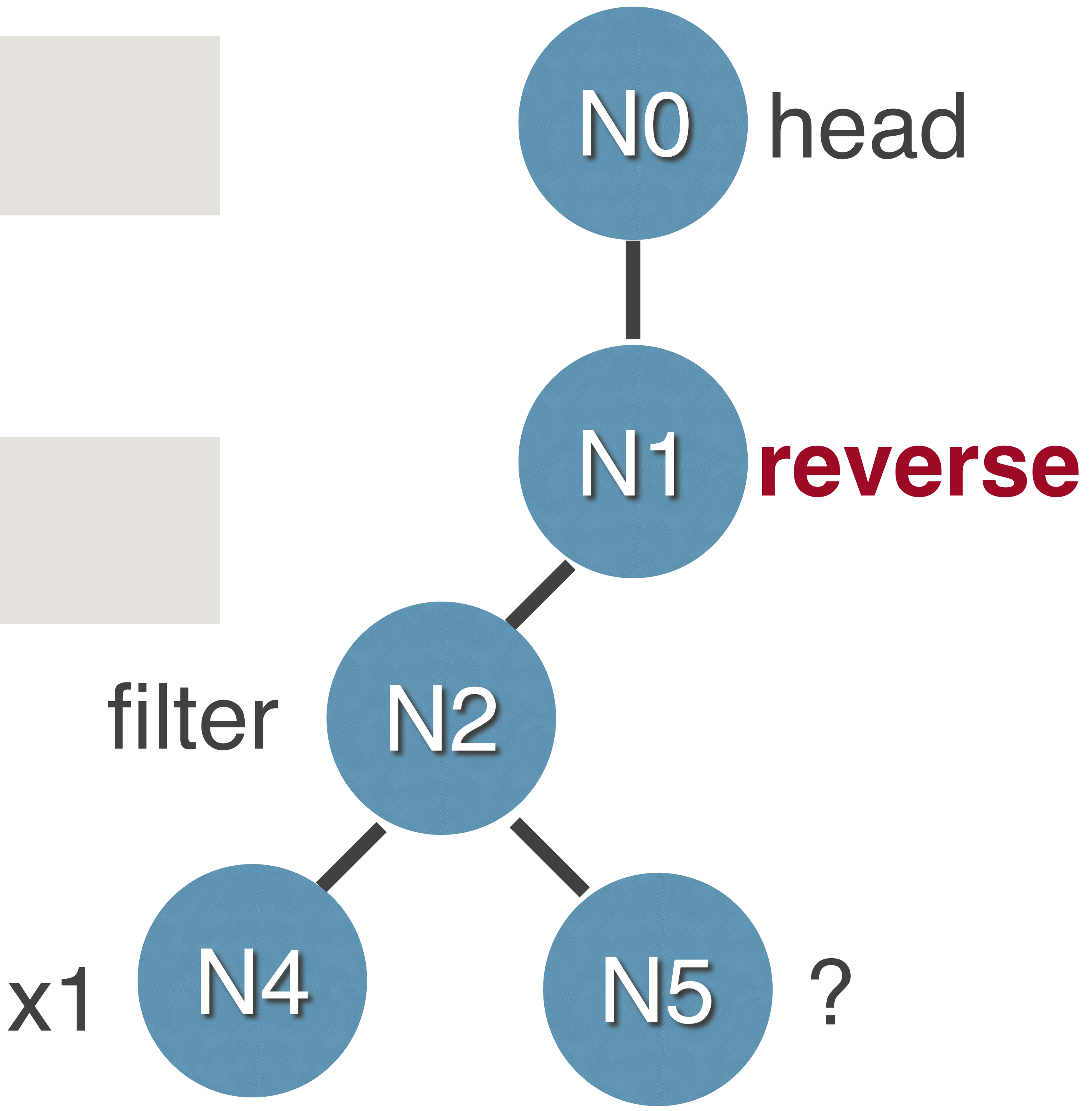
If the specification of X' implies X then X ≡ X':

- take ≡ sort ≡ reverse ≡ filter

Learning from Mistakes

How to detect equivalent modulo conflict components?

Take	n1.max <= n2.max
Sort	n1.max = n2.max
Reverse	**n1.max = n2.max**
Filter	n1.max <= n2.max



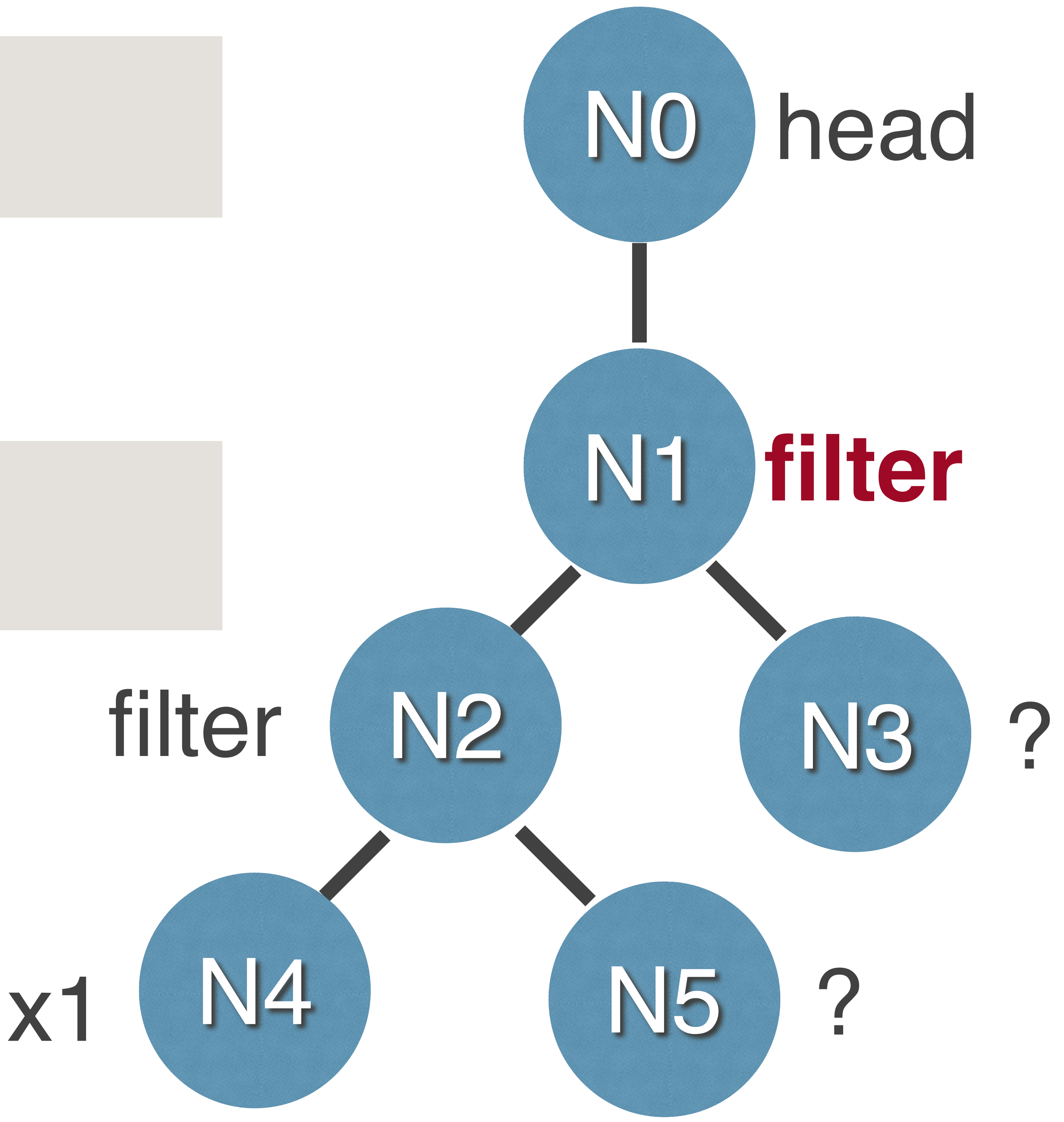
If the specification of X’ implies X then X ≡ X’:

- take ≡ sort ≡ reverse ≡ filter

Learning from Mistakes

How to detect equivalent modulo conflict components?

Take	n1.max <= n2.max
Sort	n1.max = n2.max
Reverse	n1.max = n2.max
Filter	**n1.max <= n2.max**



If the specification of X' implies X then X ≡ X':

- take ≡ sort ≡ reverse ≡ filter

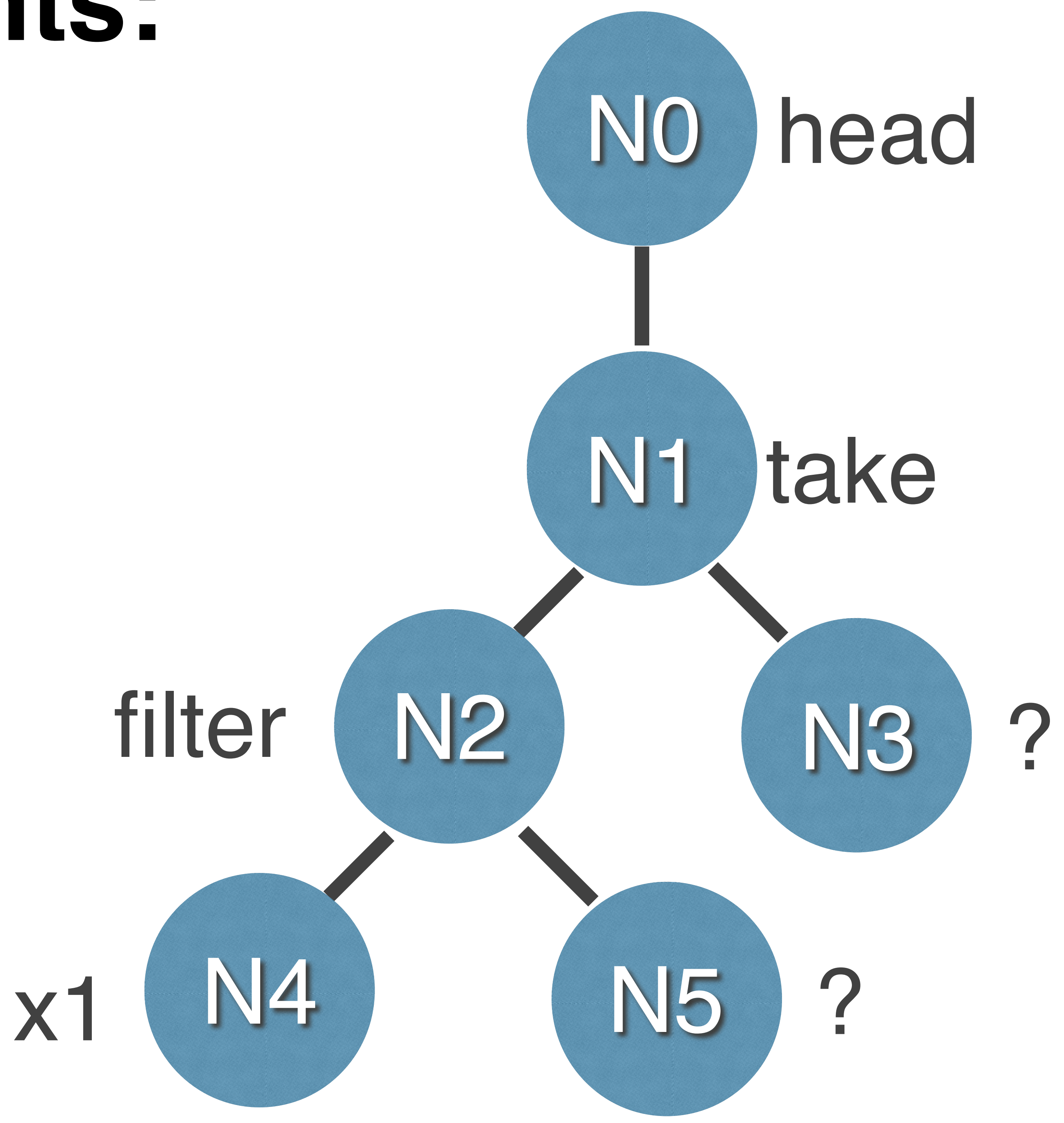
Learning from Mistakes

How to detect equivalent modulo conflict components?

Using modulo conflict components:

- We can learn a lemma that allow us to rule out **63** other partial programs!

- Learning allows the synthesis algorithm to avoid similar mistakes in the future!



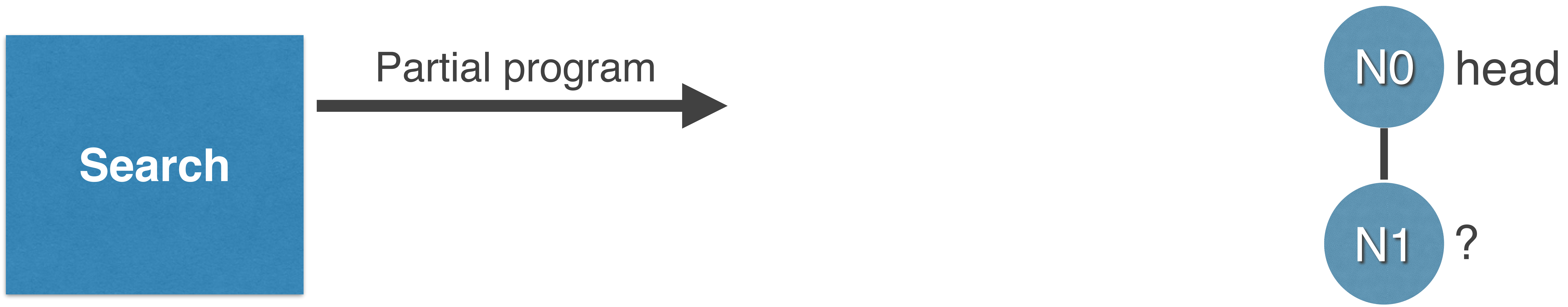
Conflict-Driven Synthesis

Conflict-Driven Synthesis

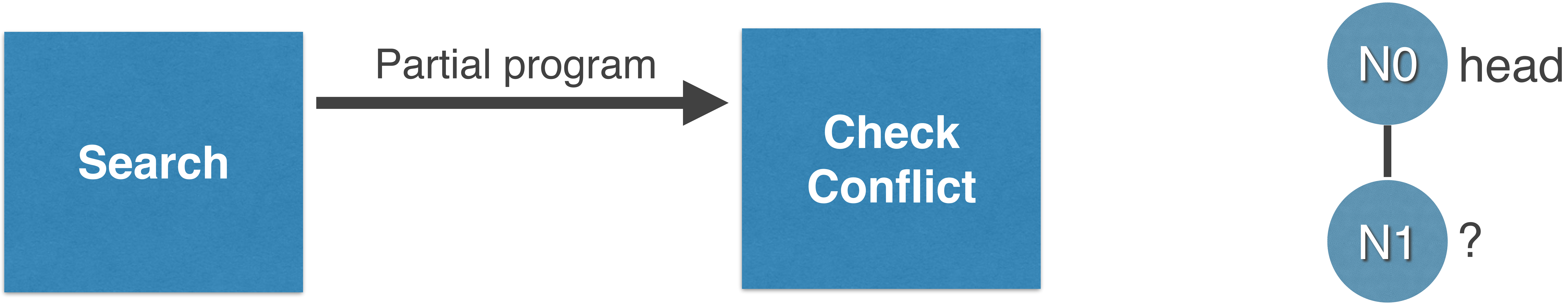
Search

N0 ?

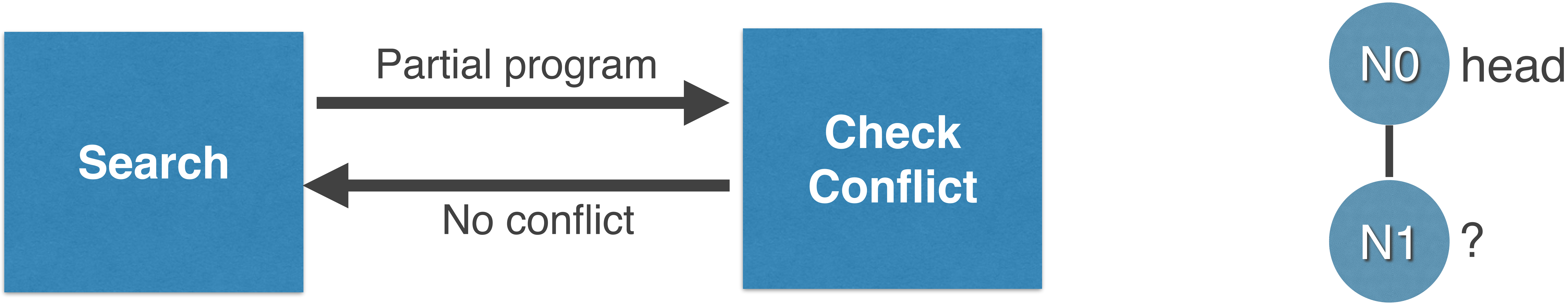
Conflict-Driven Synthesis



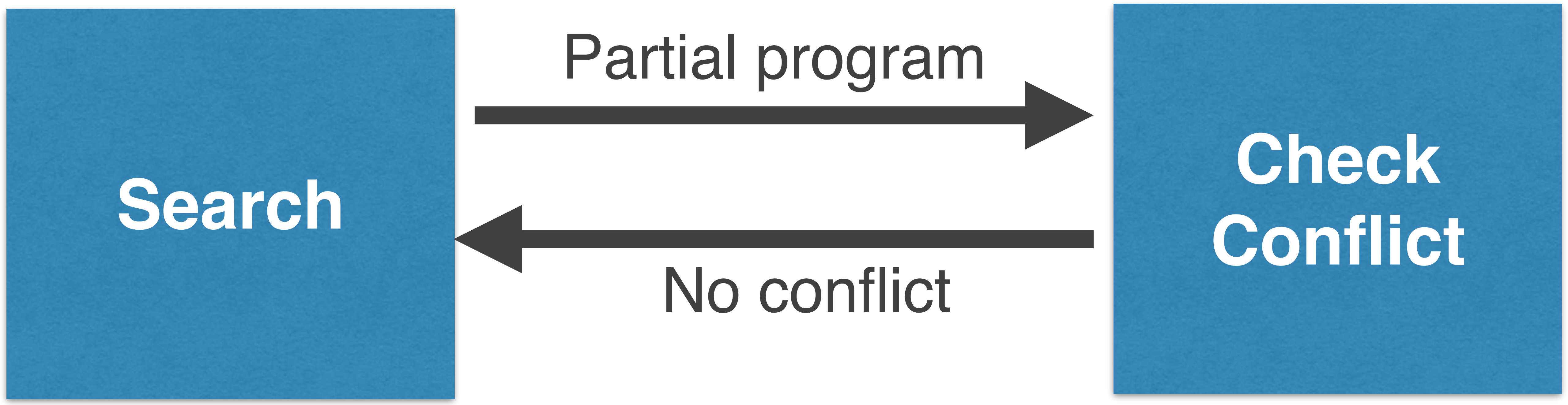
Conflict-Driven Synthesis

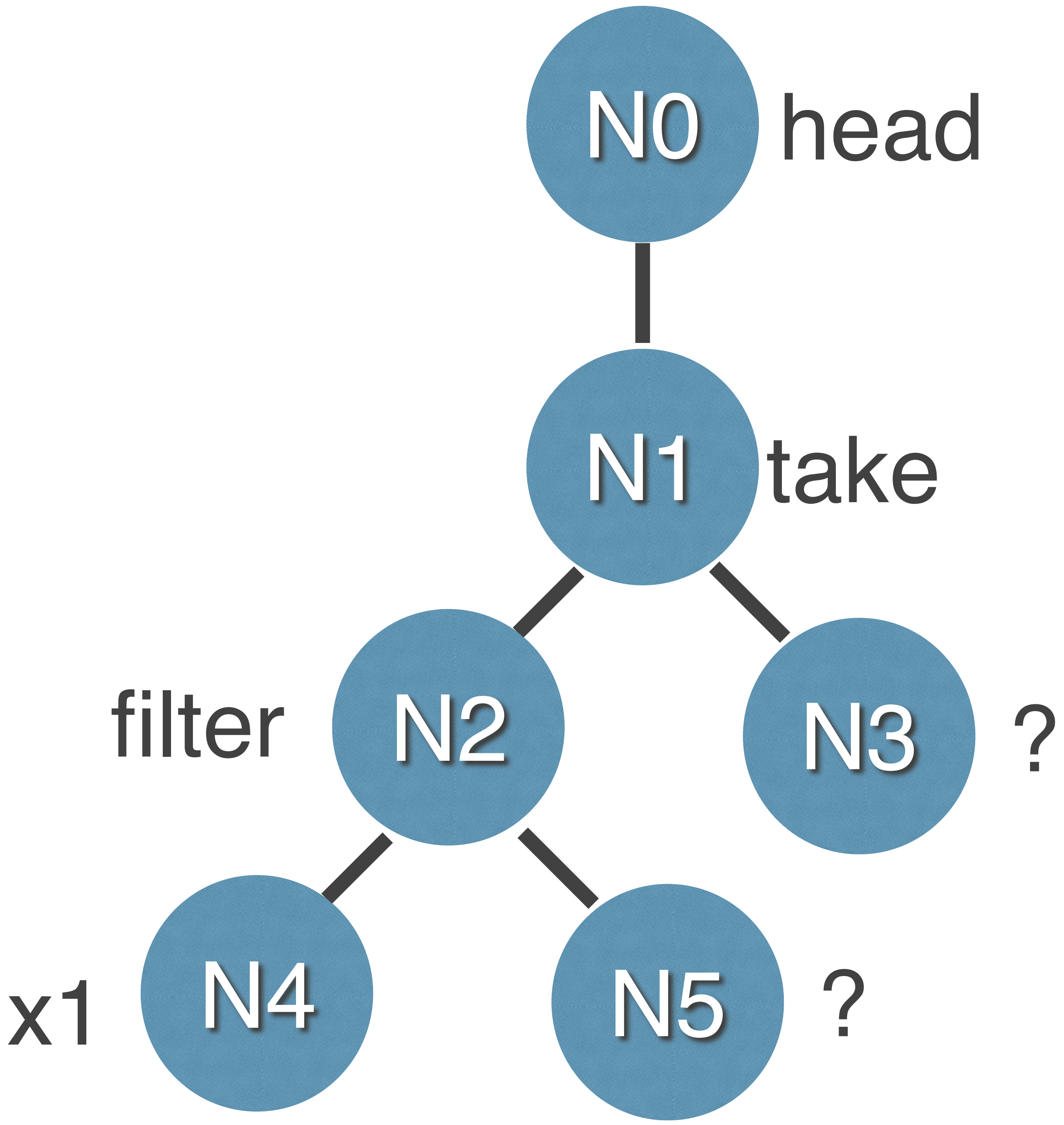


Conflict-Driven Synthesis

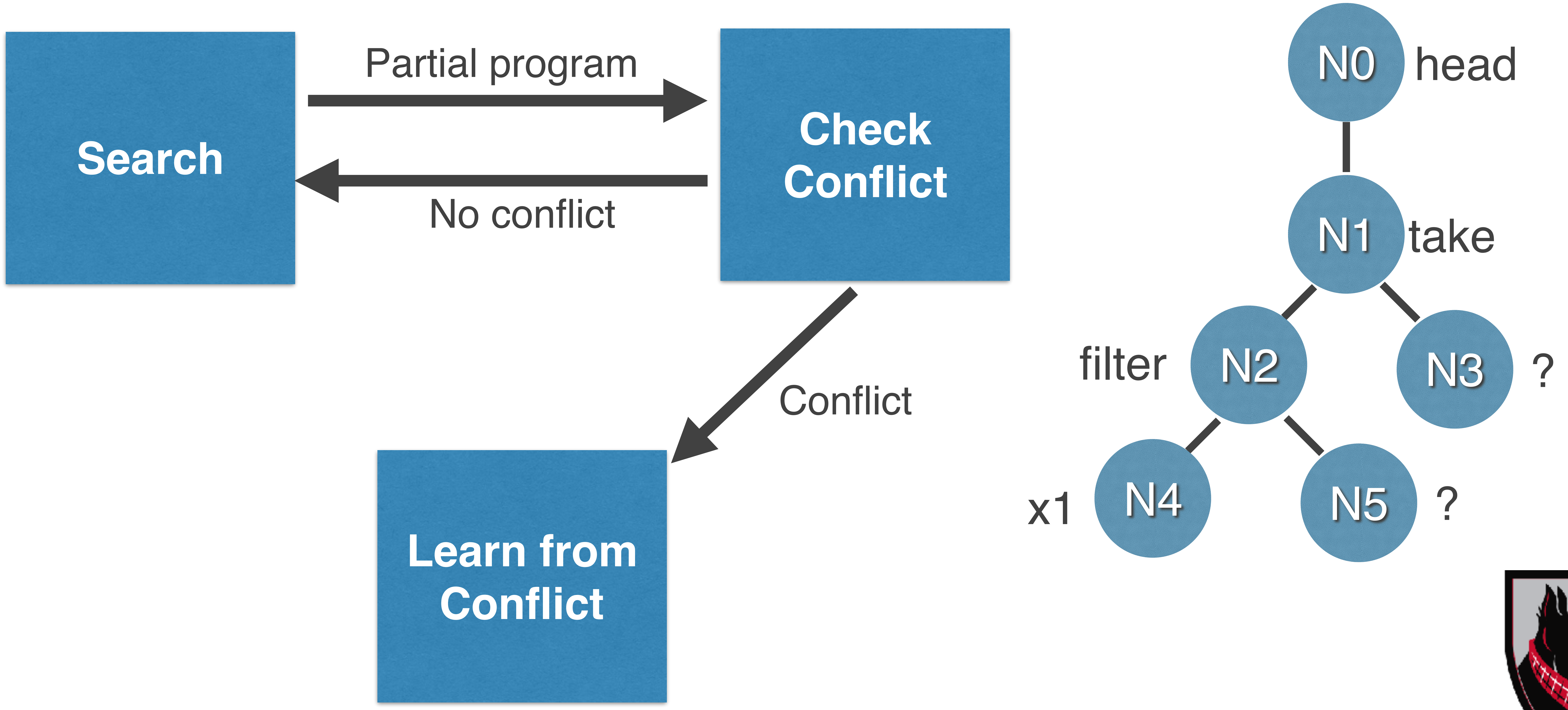


Conflict-Driven Synthesis

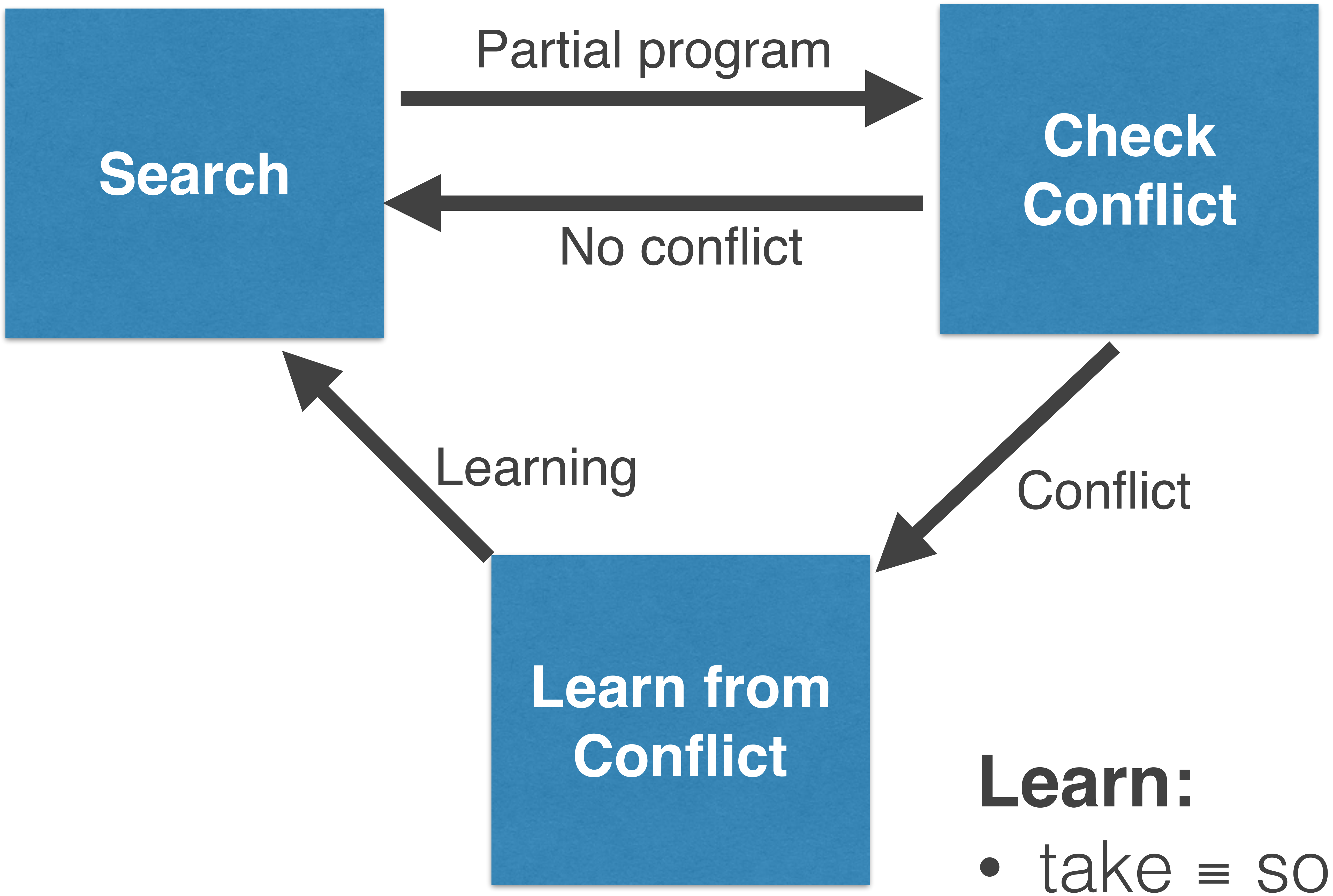


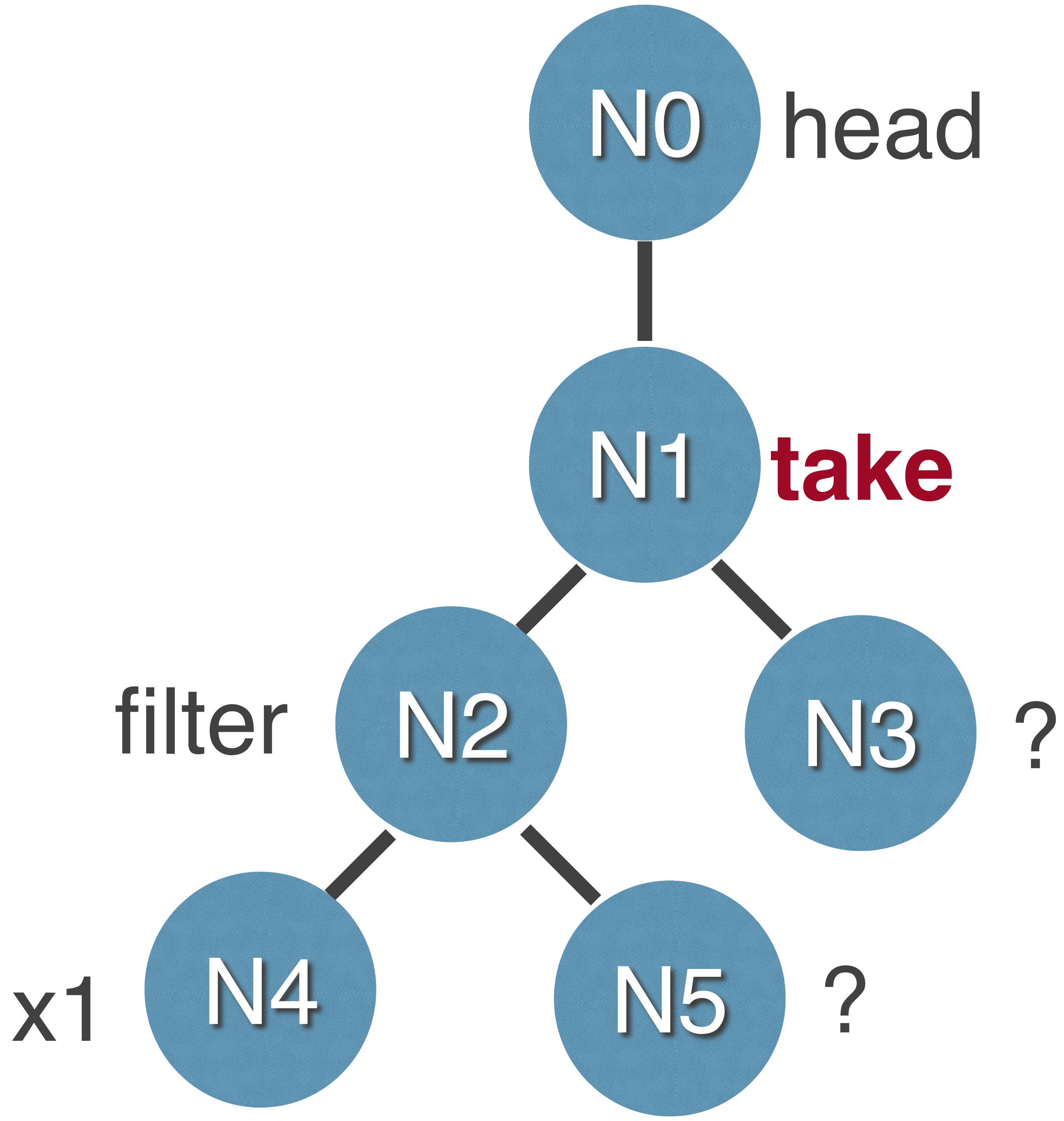


Conflict-Driven Synthesis



Conflict-Driven Synthesis

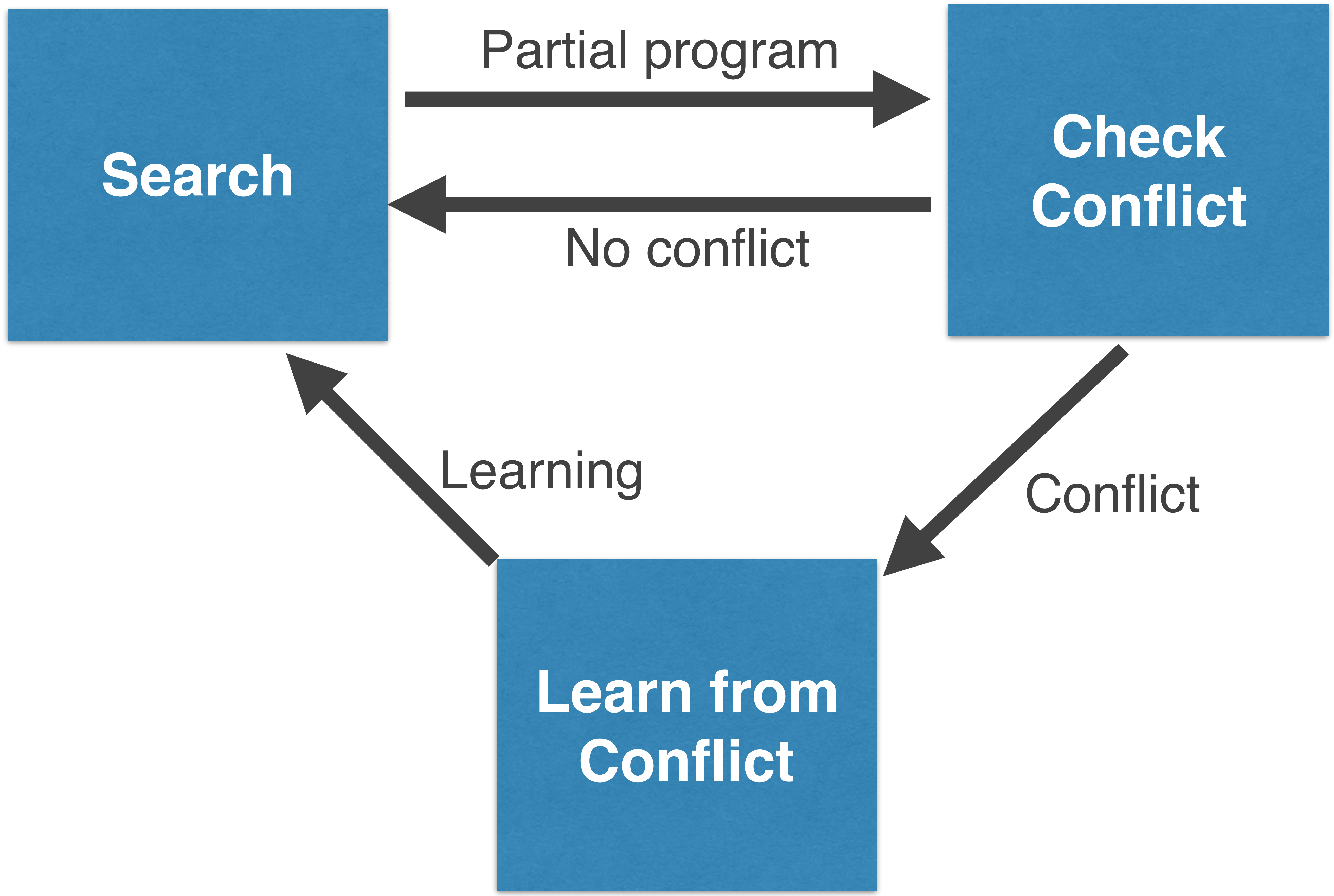




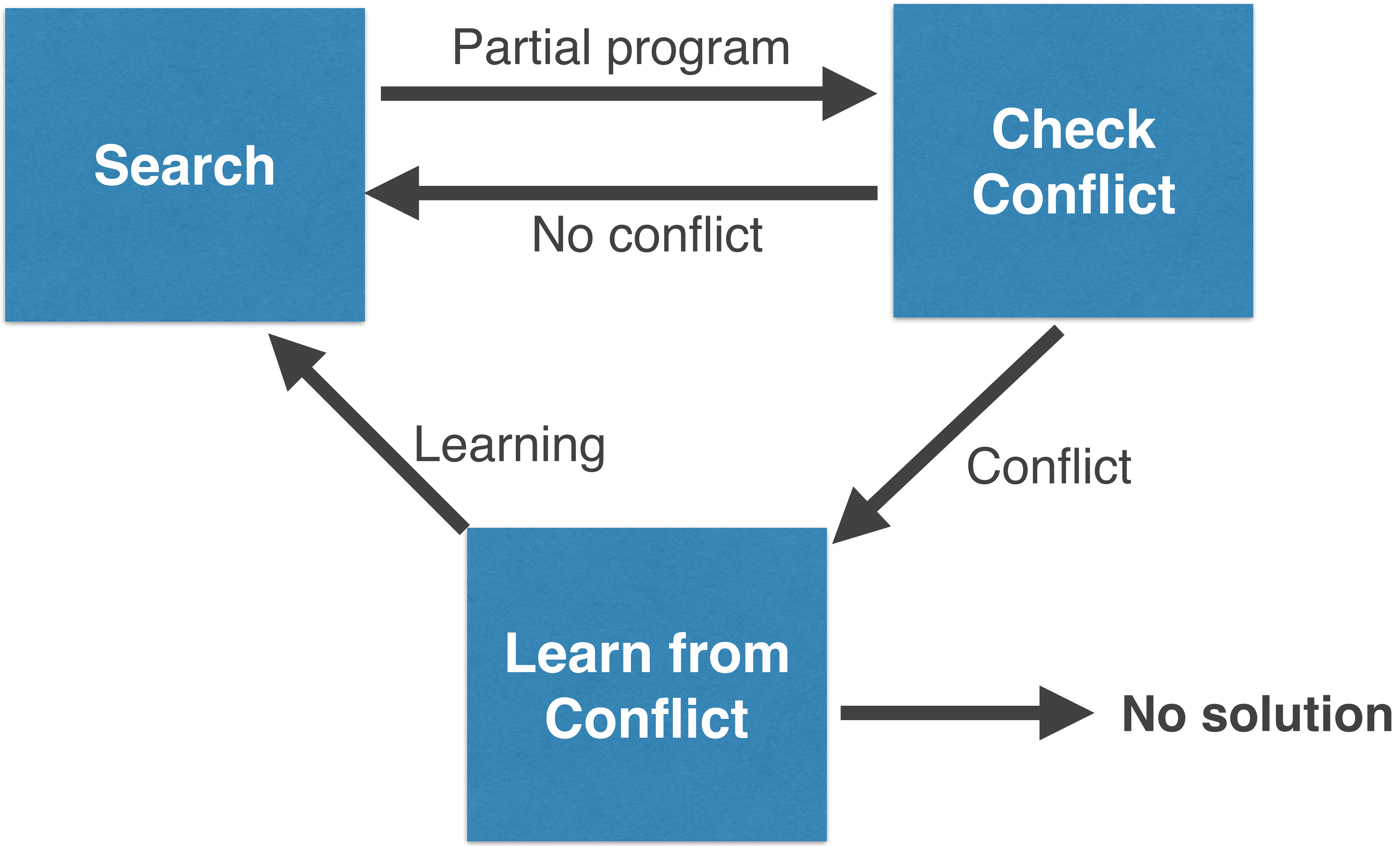
Learn:

- take ≡ sort ≡ reverse ≡ filter

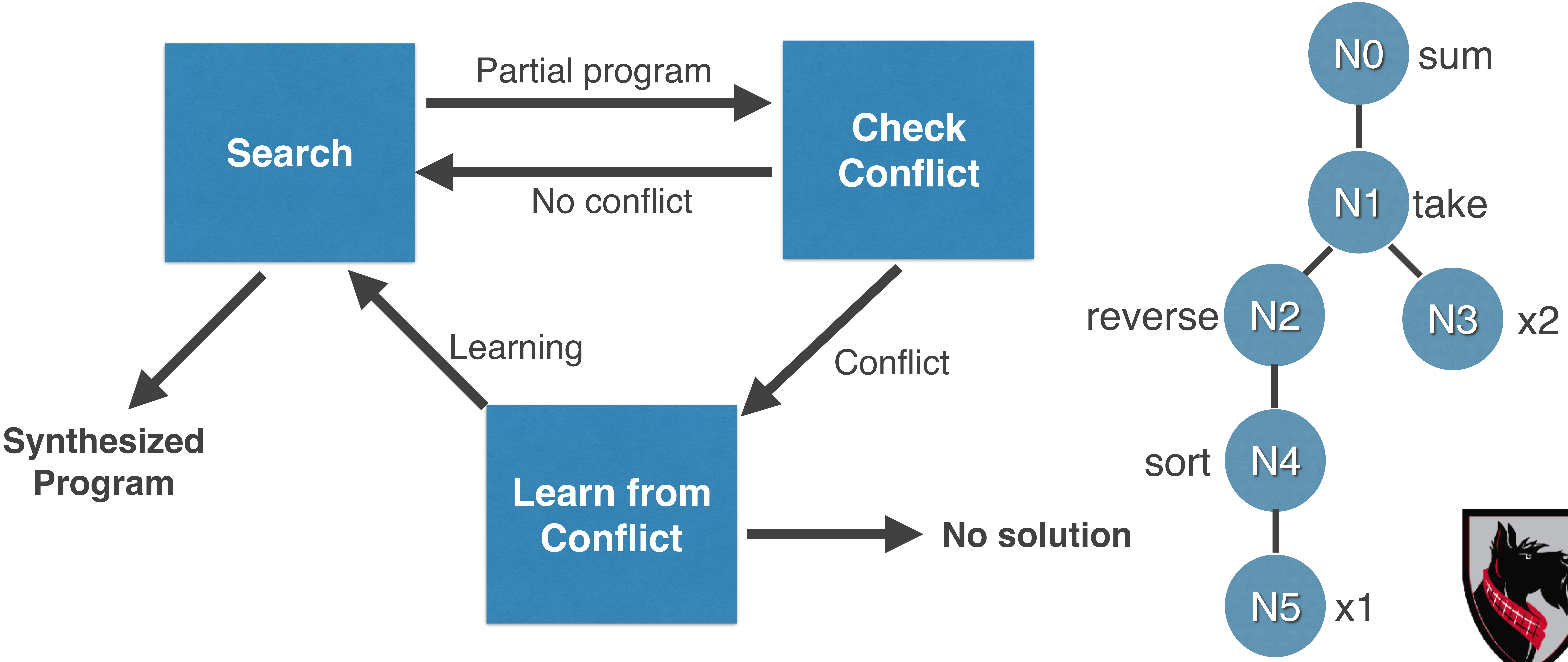
Conflict-Driven Synthesis



Conflict-Driven Synthesis



Conflict-Driven Synthesis



Experimental Evaluation

DeepCoder (Microsoft Research):

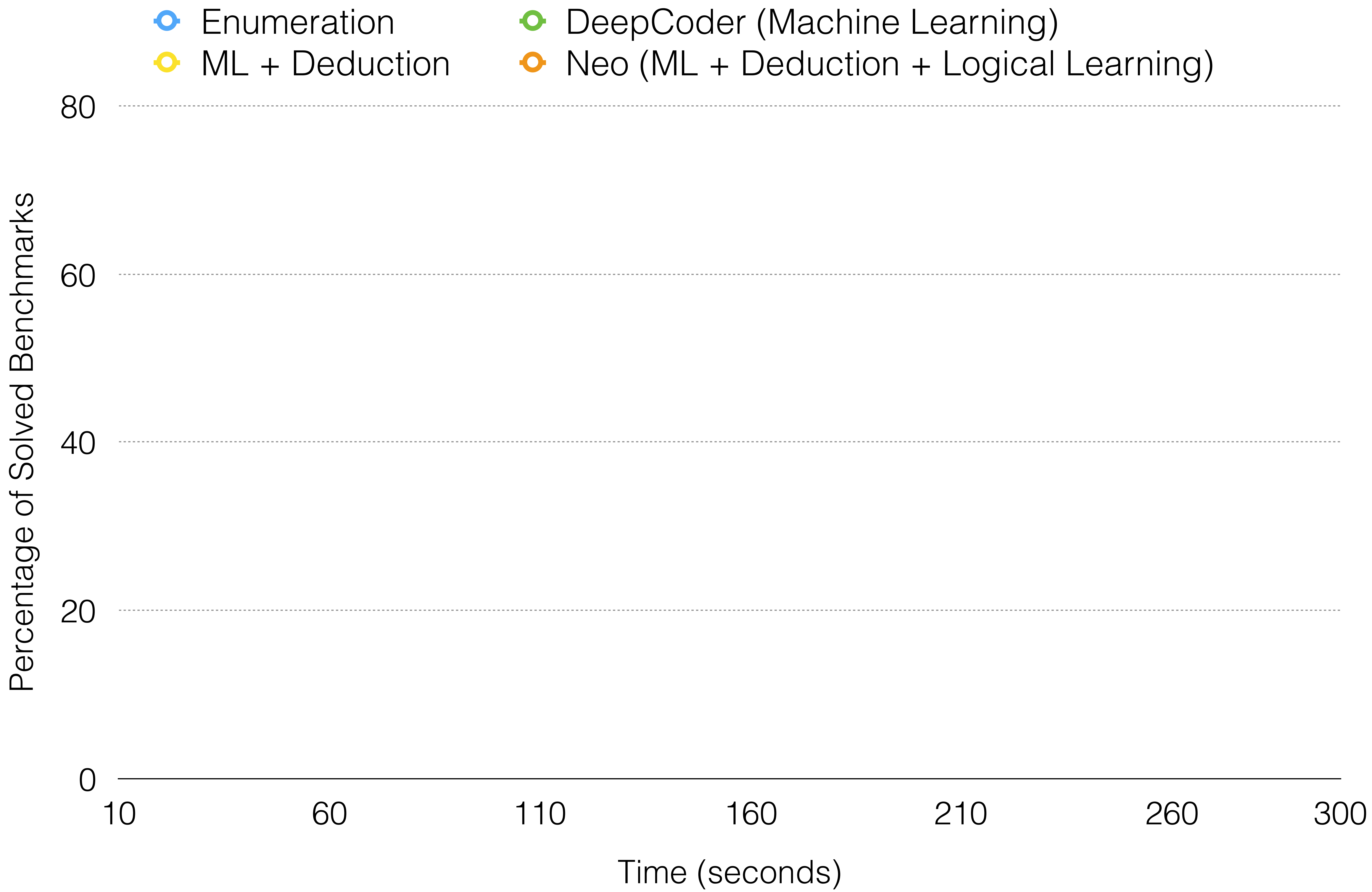
- List manipulation synthesizer
- Uses deep learning to guide the search
- We reimplemented DeepCoder statistical model in our Conflict-Driven Synthesis Framework

Benchmarks:

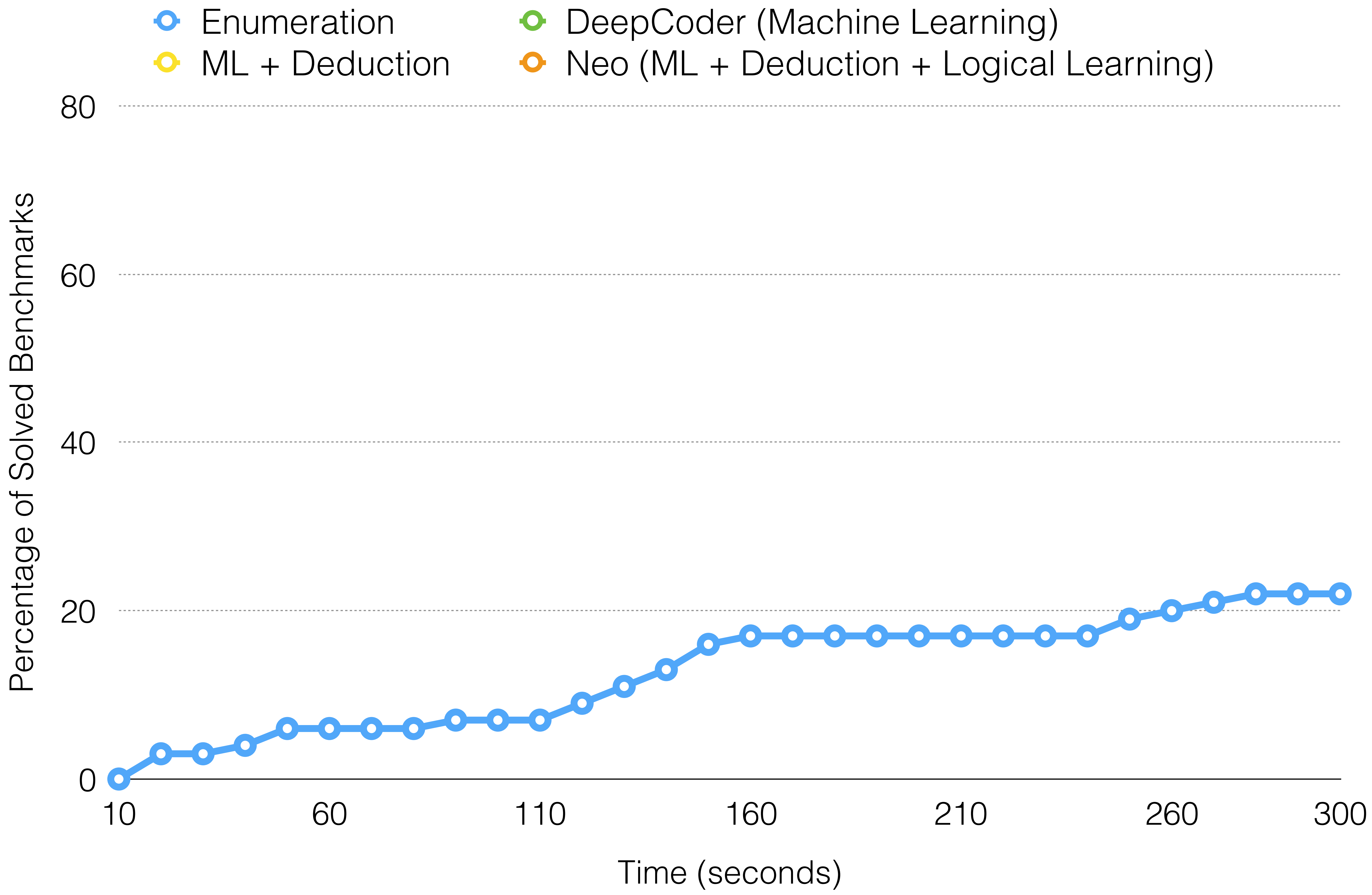
- 100 challenging benchmarks described in DeepCoder's paper

Neo vs DeepCoder

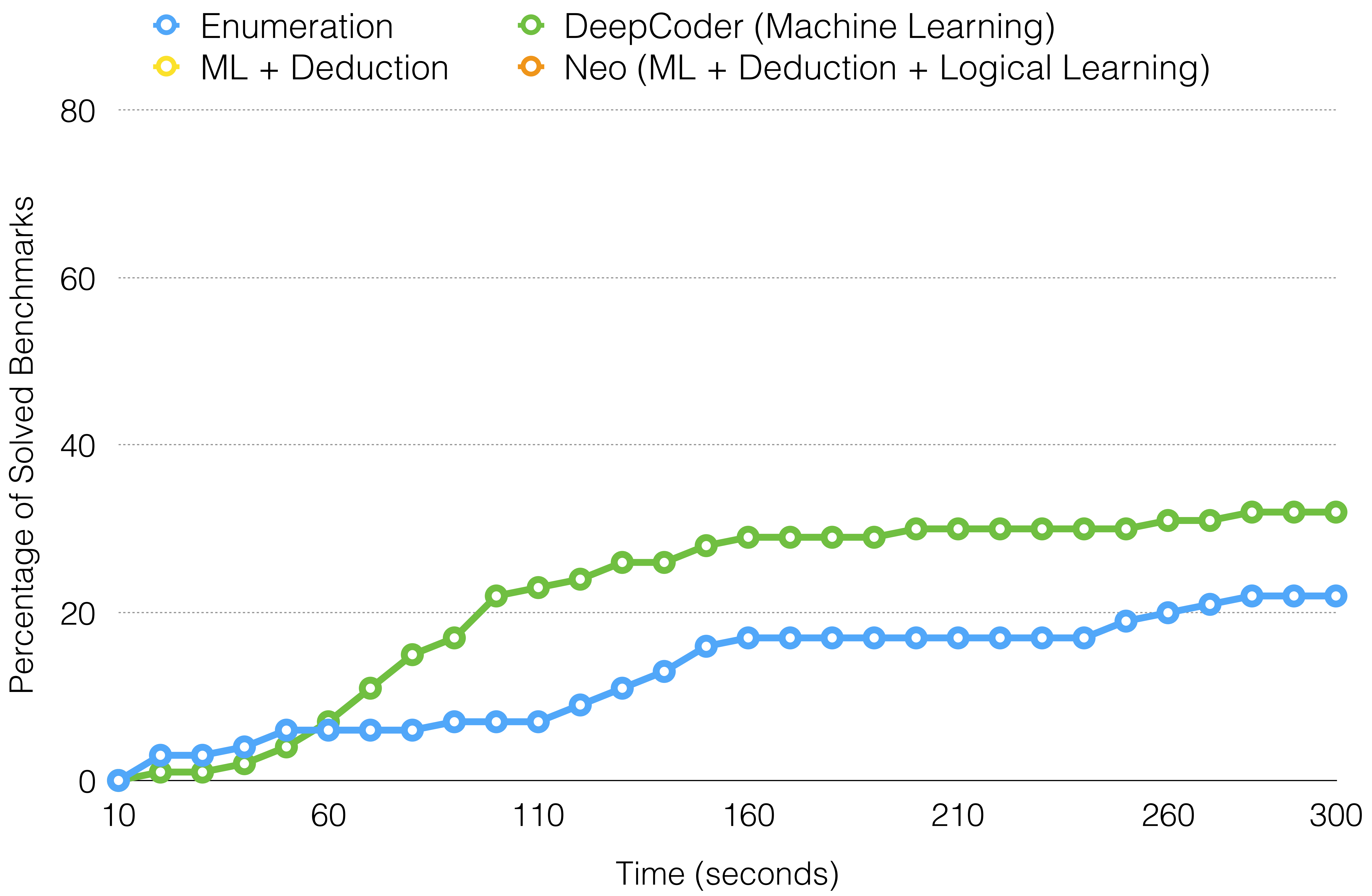
Neo vs DeepCoder



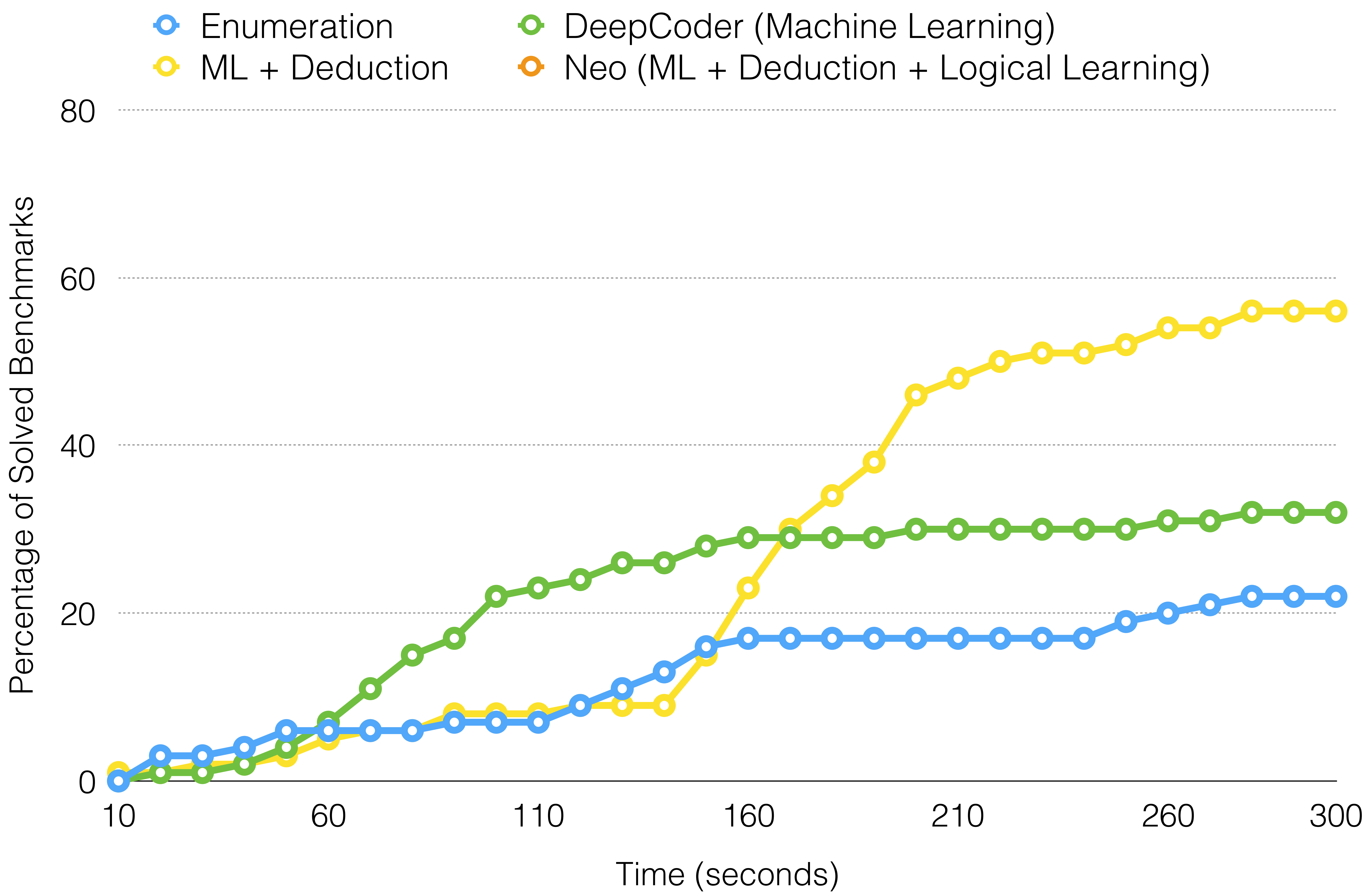
Neo vs DeepCoder



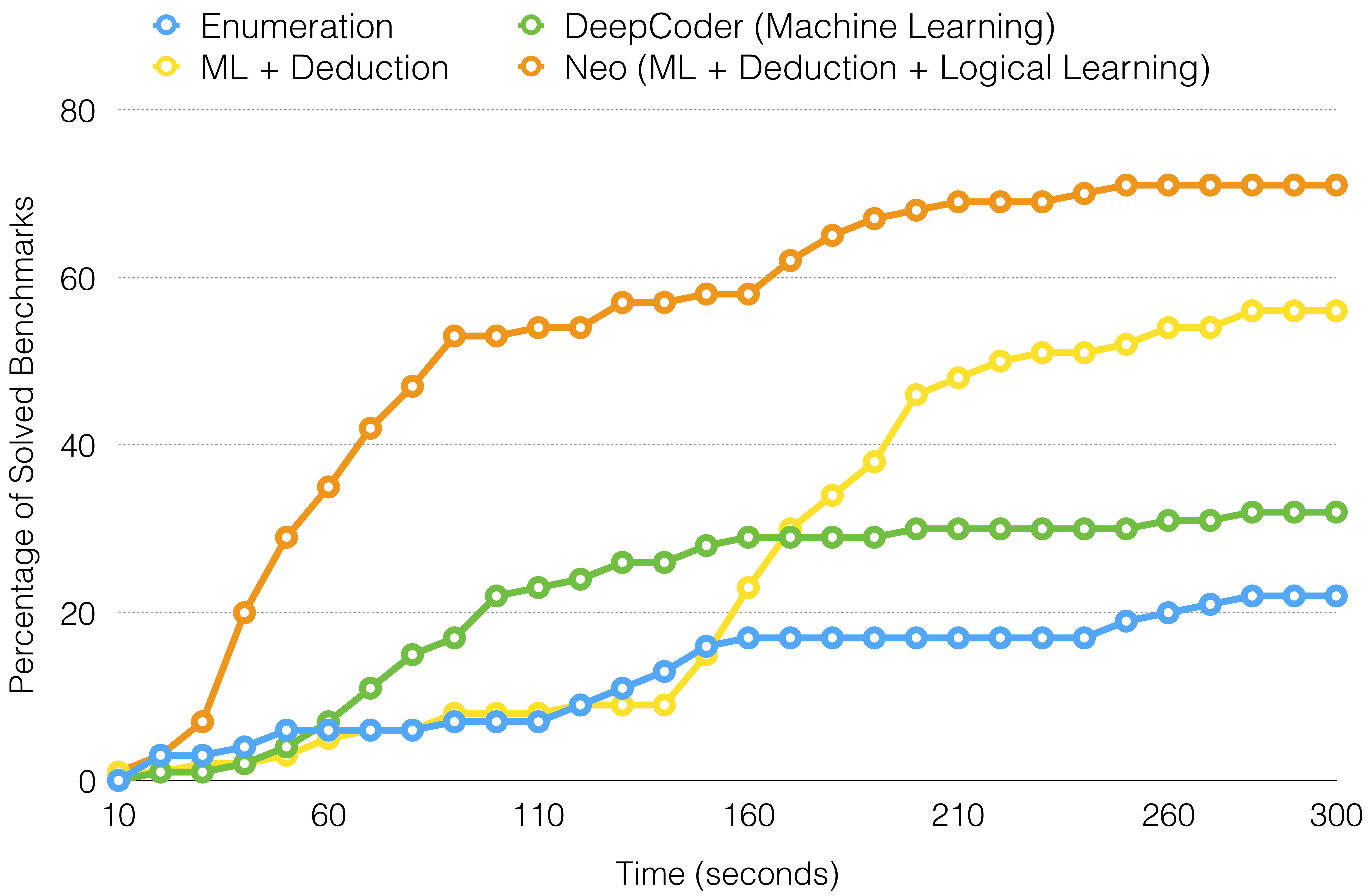
Neo vs DeepCoder



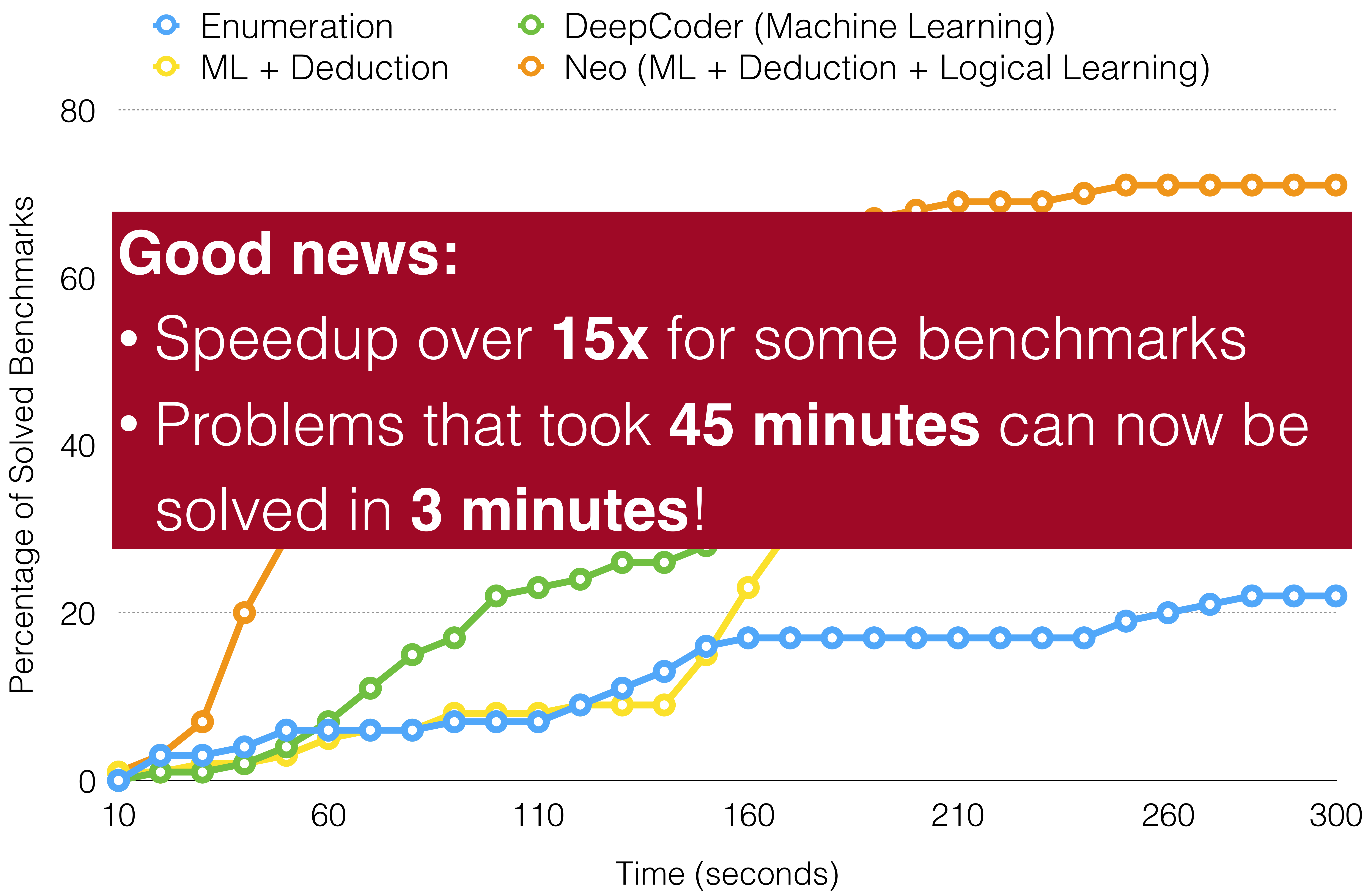
Neo vs DeepCoder



Neo vs DeepCoder



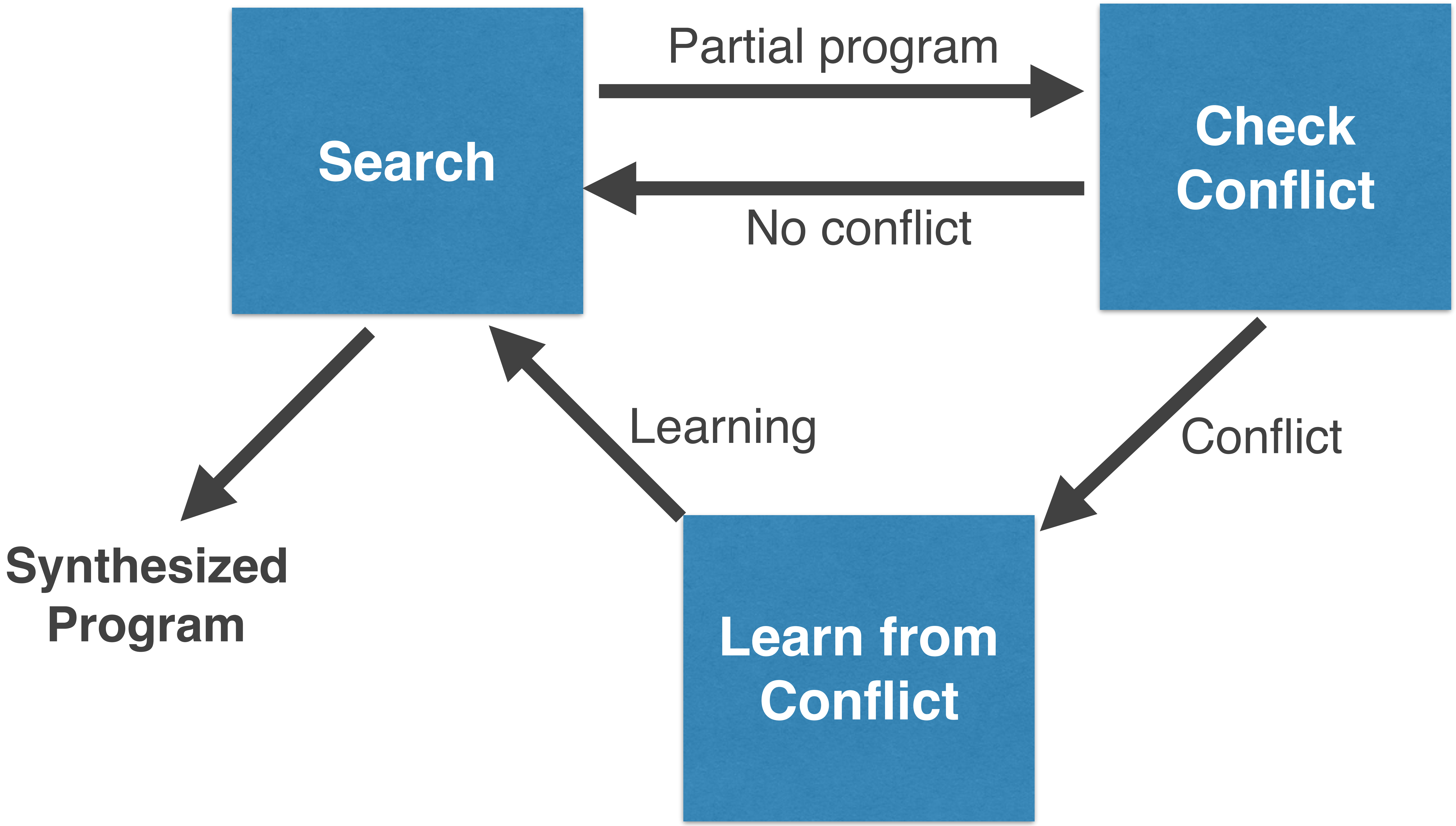
Neo vs DeepCoder



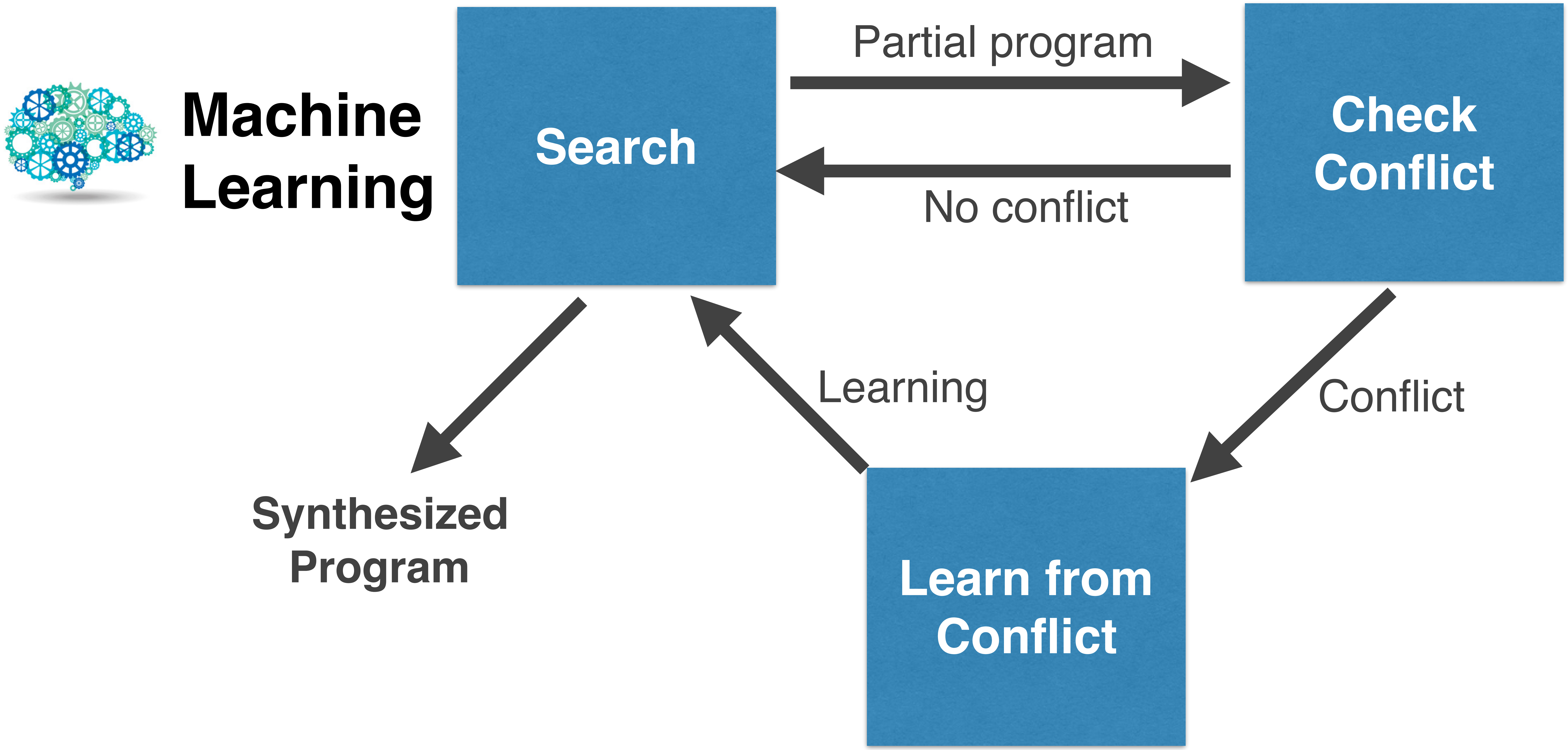
Good news:

- Speedup over **15x** for some benchmarks
- Problems that took **45 minutes** can now be solved in **3 minutes**!

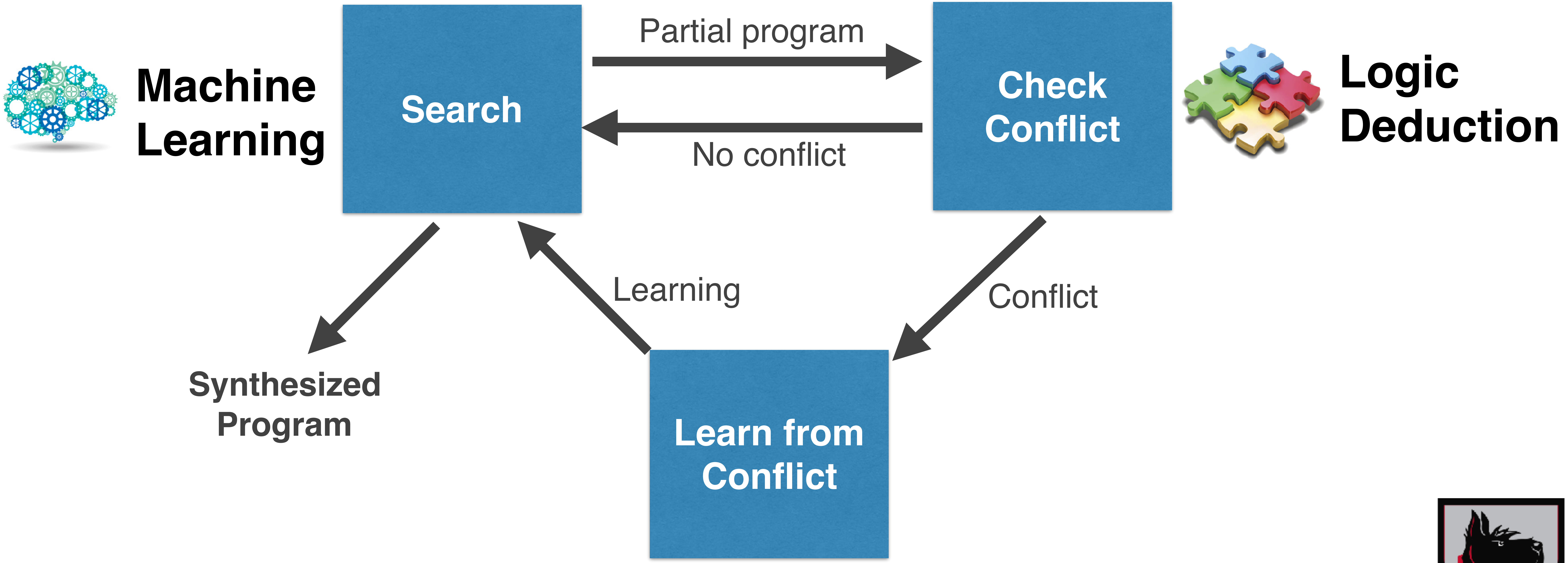
Neo: Conflict-Driven Synthesis



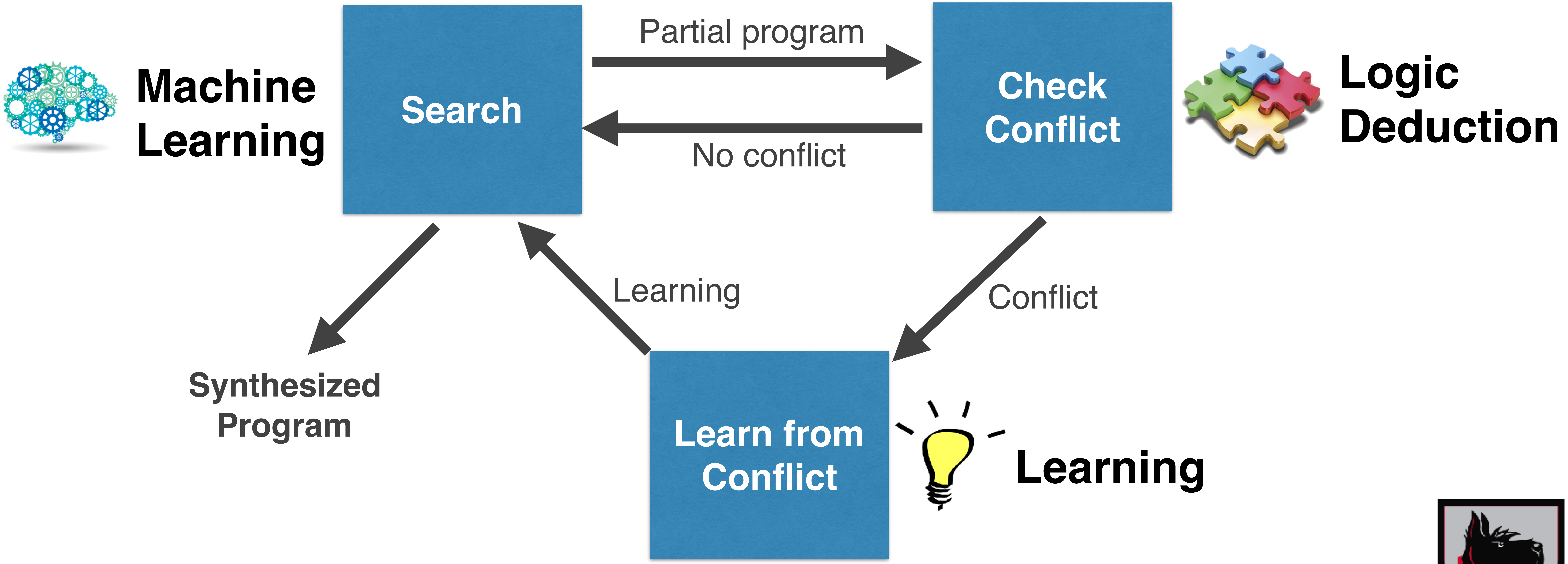
Neo: Conflict-Driven Synthesis



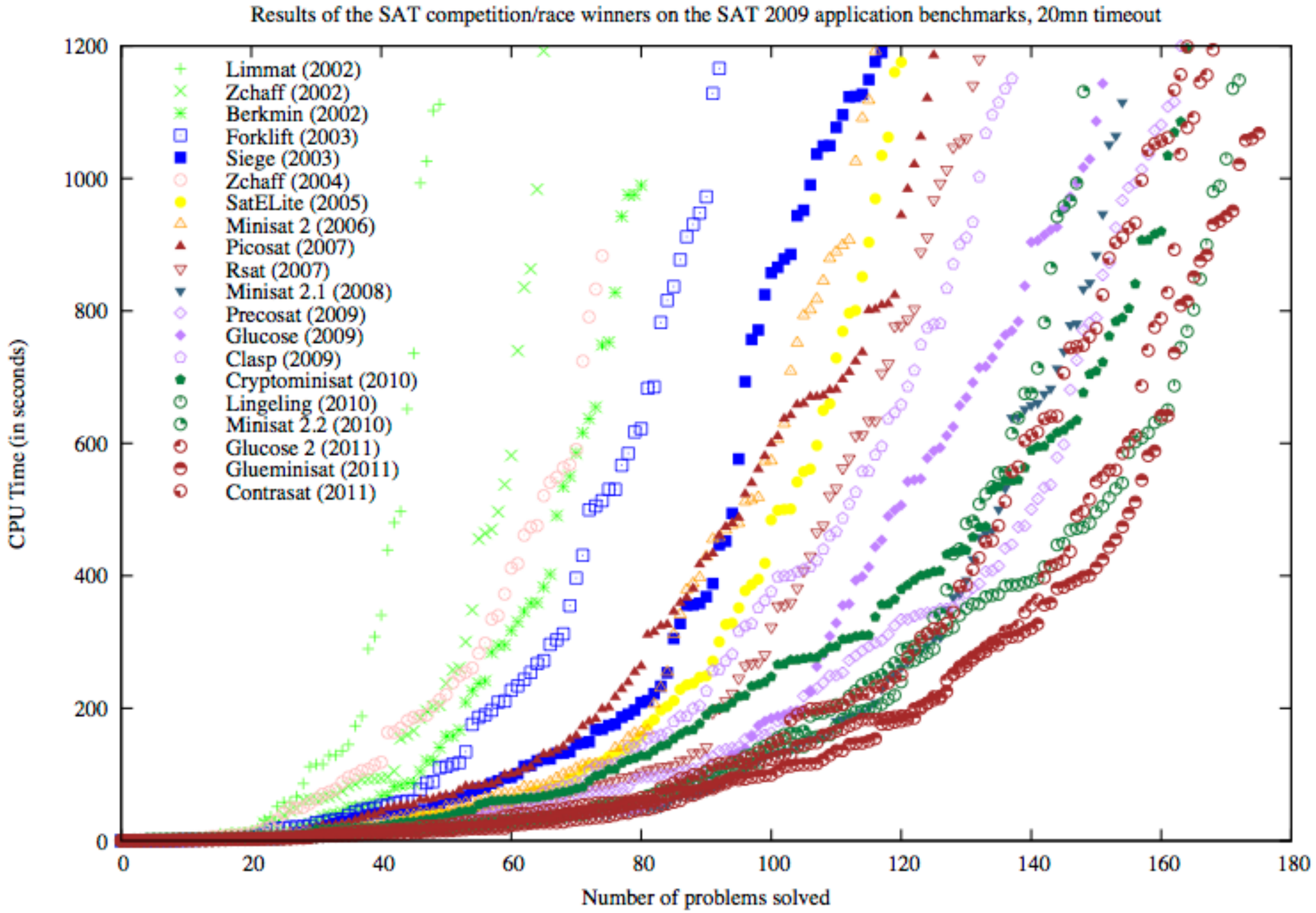
Neo: Conflict-Driven Synthesis



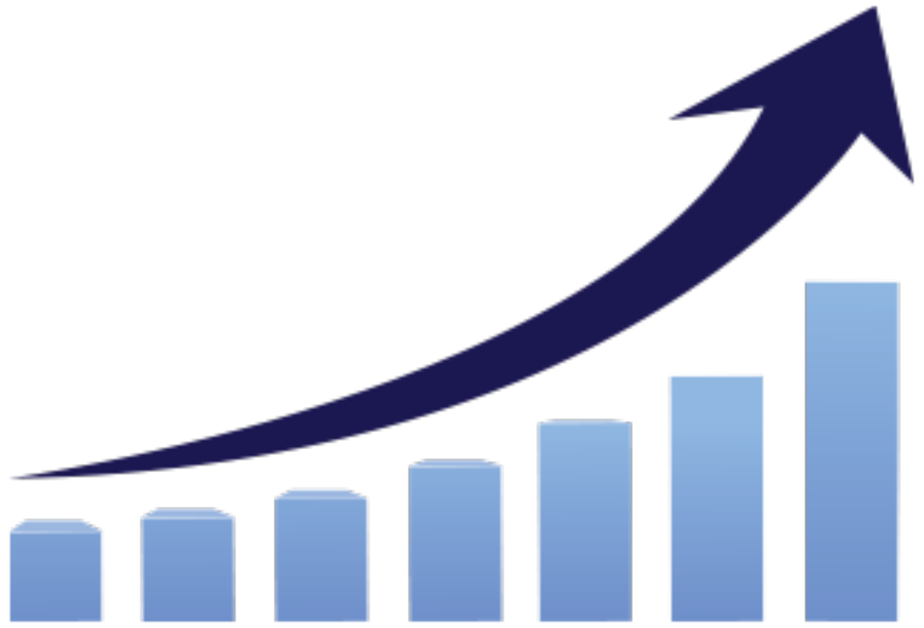
Neo: Conflict-Driven Synthesis



Scalability Through Learning

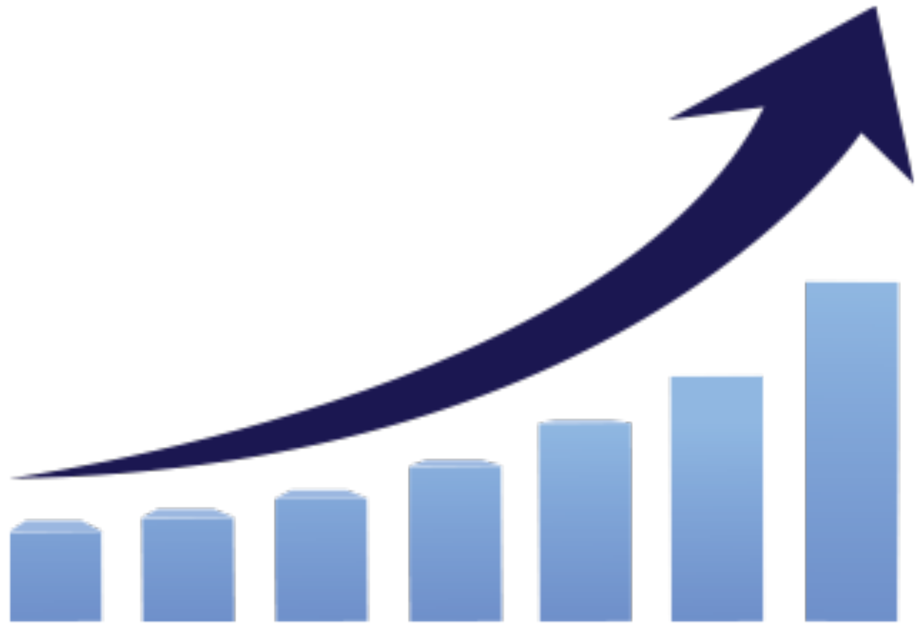


Scalability of Program Synthesis



- **SAT Solving:**
  - <1996: SAT solving was seen as intractable!
  - 1996-now: Conflict-Driven Clause Learning SAT solvers revolutionized the field!

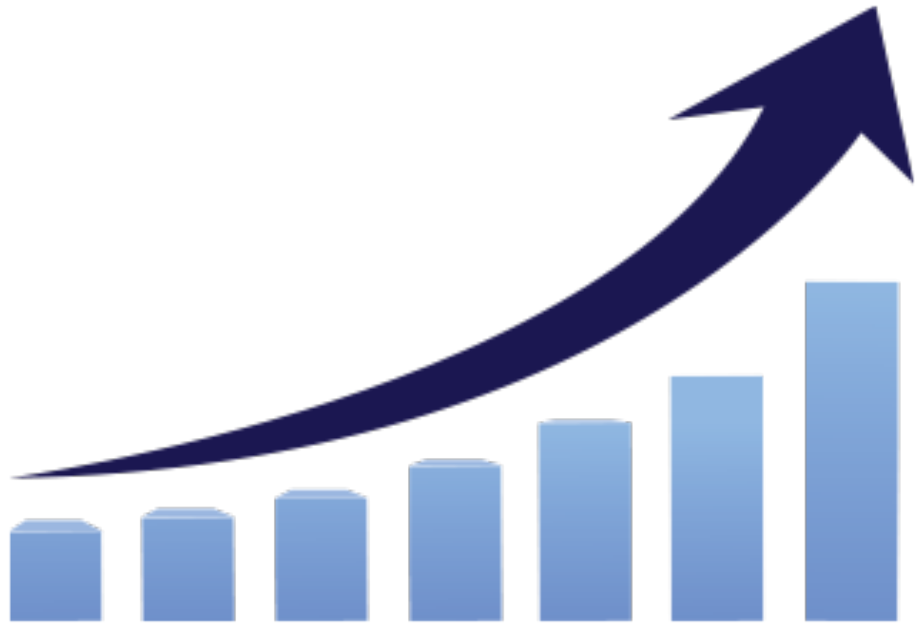
Scalability of Program Synthesis



- **SAT Solving:**
  - <1996: SAT solving was seen as intractable!
  - 1996-now: Conflict-Driven Clause Learning SAT solvers revolutionized the field!

- **Program Synthesis:**
  - First step towards learning from mistakes

Scalability of Program Synthesis



- **SAT Solving:**
  - <1996: SAT solving was seen as intractable!
  - 1996-now: Conflict-Driven Clause Learning SAT solvers revolutionized the field!

- **Program Synthesis:**
  - First step towards learning from mistakes
  - Can **learning** push the **boundaries** of program synthesis?

Applications of Program Synthesis

Data Science

Databases

Program Repair

Security

Computer-Aided Education

And many others!