Cornell University
Computer Systems Laboratory

CSL

# TYPE FREEZING: EXPLOITING ATTRIBUTE TYPE MONOMORPHISM IN TRACING JIT COMPILERS

Lin Cheng[1], Berkin Ilbeyi[1], Carl Friedrich Bolz-Tereick[2], Christopher Batten[1]

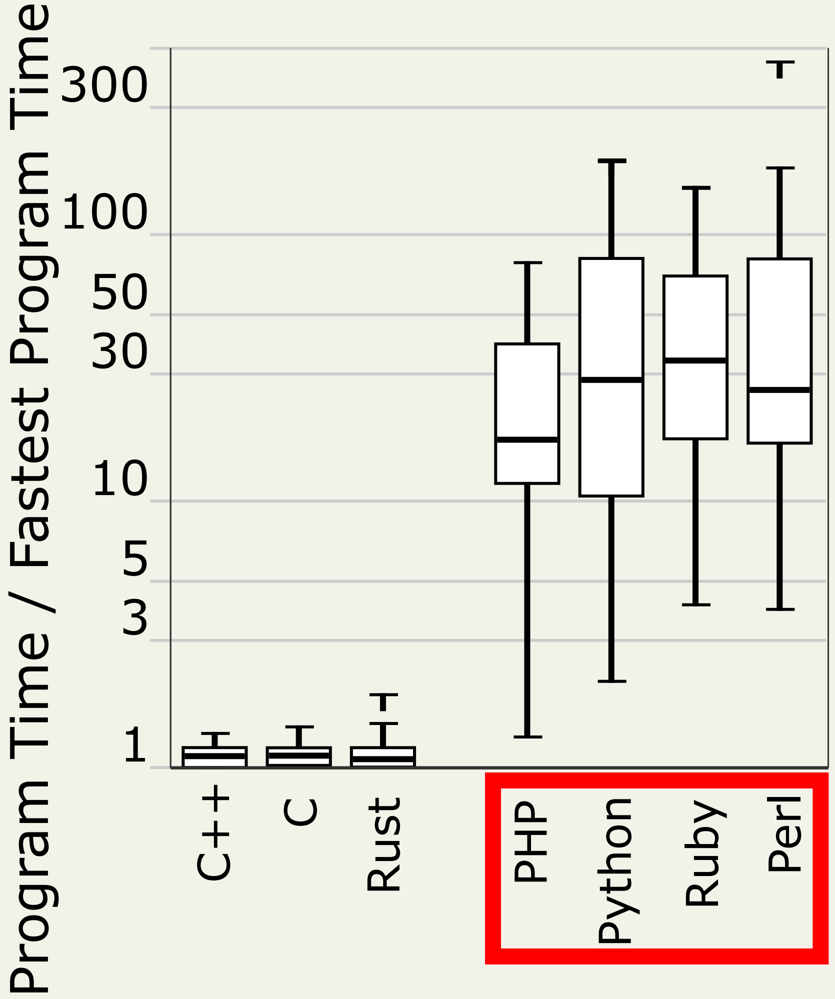
[1]Computer Systems Laboratory
Cornell University
[2]Heinrich-Heine-Universität Düsseldorf

# Dynamic Languages Are Popular and Slow

CSL

| Rank | Language | Type | Score |
|---|---|---|---|
| 1 | Python | | 100.0 |
| 2 | Java | | 96.3 |
| 3 | C | | 94.4 |
| 4 | C++ | | 87.5 |
| 5 | R | | 81.5 |
| 6 | JavaScript | | 79.4 |
| 7 | C# | | 74.5 |
| 8 | Matlab | | 70.6 |
| 9 | Swift | | 69.1 |
| 10 | Go | | 68.0 |

S. Cass. "The Top Programming Languages 2019." IEEE Spectrum.




I. Guoy. "The Computer Languages Benchmarks Game."

# TYPE POLYMORPHISM → TYPE MONOMORPHISM

- An identifier can hold different types of data

```
def foo( pt ):
    x = pt.x
    y = pt.y
    return x + y
```

```
>> pt = Point()
>> pt.x = 14
>> pt.y = 28
>> foo( pt )
42
```

```
>> pt = Point()
>> pt.x = 14.0
>> pt.y = 28.0
>> foo( pt )
42.0
```

```
>> pt = Point()
>> pt.x = "14"
>> pt.y = "28"
>> foo( pt )
“1428”
```

[1] Xia et al, An Empirical Study of Dynamic Types for Python Projects. *Int’l Conf. on Software Analysis, Testing, and Evolution* (Nov 2018)

# Type Polymorphism → Type Monomorphism

CSL

- An identifier can hold different types of data
- At least 79% of the identifiers in real world Python applications are type monomorphic [1]

```
def foo( pt ):
    x = pt.x
    y = pt.y
    return x + y
```

```
>> pt = Point()
>> pt.x = 14
>> pt.y = 28
>> foo( pt )
42
```

```
>> pt = Point()
>> pt.x = 14.0
>> pt.y = 28.0
>> foo( pt )
42.0
```

```
>> pt = Point()
>> pt.x = "14"
>> pt.y = "28"
>> foo( pt )
“1428”
```

**Type Specialization Via JIT Compiling**

```
assert_type( pt, Point )
_x = load_attr( pt, “x” )
_y = load_attr( pt, “y” )
assert_type( _x, int )
assert_type( _y, int )
r = add_int( _x, _y )
return r
```

[1] Xia et al, An Empirical Study of Dynamic Types for Python Projects. *Int’l Conf. on Software Analysis, Testing, and Evolution* (Nov 2018)

# Type Polymorphism → Type Monomorphism

CSL

- An identifier can hold different types of data
- At least 79% of the identifiers in real world Python applications are type monomorphic [1]

```
def foo( pt ):
    x = pt.x
    y = pt.y
    return x + y
```

```
>> pt = Point()
>> pt.x = 14
>> pt.y = 28
>> foo( pt )
42
```

```
>> pt = Point()
>> pt.x = 14.0
>> pt.y = 28.0
>> foo( pt )
42.0
```

```
>> pt = Point()
>> pt.x = "14"
>> pt.y = "28"
>> foo( pt )
“1428”
```

**Type Specialization Via JIT Compiling**

```
assert_type( pt, Point )
_x = load_attr( pt, “x” )
_y = load_attr( pt, “y” )
assert_type( _x, int )
assert_type( _y, int )
r = add_int( _x, _y )
return r
```

[1] Xia et al, An Empirical Study of Dynamic Types for Python Projects. *Int’l Conf. on Software Analysis, Testing, and Evolution* (Nov 2018)

# Type Polymorphism → Type Monomorphism

CSL

- An identifier can hold different types of data
- At least 79% of the identifiers in real world Python applications are type monomorphic [1]
- **Attribute type monomorphism** is a special kind of type monomorphism, in which a certain attribute of a user-defined type only holds a single type of data

```
def foo( pt ):
    x = pt.x
    y = pt.y
    return x + y
```

```
>> pt = Point()
>> pt.x = 14
>> pt.y = 28
>> foo( pt )
42
```

```
>> pt = Point()
>> pt.x = 14.0
>> pt.y = 28.0
>> foo( pt )
42.0
```

```
>> pt = Point()
>> pt.x = "14"
>> pt.y = "28"
>> foo( pt )
"1428"
```

**Type Specialization Via JIT Compiling**

```
assert_type( pt, Point )
_x = load_attr( pt, "x" )
_y = load_attr( pt, "y" )
assert_type( _x, int )
assert_type( _y, int )
r = add_int( _x, _y )
return r
```

Knowing pt is a Point instance implies x and y are integers

[1] Xia et al, An Empirical Study of Dynamic Types for Python Projects. *Int'l Conf. on Software Analysis, Testing, and Evolution* (Nov 2018)

# Attribute Type Monomorphism → 75% of All Reads

CSL

| Benchmark | Total | Attribute Reads: Monomorphic Primitive (%) | Monomorphic User-Defined (%) | Polymorphic (%) |
|---|---|---|---|---|
| deltablue | 524.10 M | 56.6 | 41.7 | 1.7 |
| raytrace | 5.01 B | 2.7 | 9.6 | 87.4 |
| richards | 808.08 M | 64.5 | 7.5 | 24.6 |
| eparse | 20.34 M | 51.6 | 0.1 | 48.4 |
| telco | 376.50 M | 70.9 | 1.6 | 27.5 |
| float | 150.01 M | 100.0 | 0.0 | 0.0 |
| html5lib | 21.17 M | 70.0 | 5.7 | 24.4 |
| chaos | 538.39 M | 86.1 | 0.0 | 13.9 |
| pickle | 55.93 M | 100.0 | 0.0 | 0.0 |
| django | 32.48 M | 71.5 | 14.2 | 14.3 |
| sympy | 6.20 M | 87.3 | 0.0 | 12.6 |
| sympy-opt | 6.20 M | 86.8 | 0.0 | 13.1 |
| gcbench | 37.14 M | 0.0 | 0.0 | 100.0 |
| genshi-xml | 5.84 M | 98.0 | 1.7 | 0.3 |
| chameleon | 451.46 K | 84.0 | 0.4 | 15.6 |
| mako | 260.12 K | 73.7 | 17.5 | 8.8 |
| meteor-contest | 11.52 K | 77.9 | 0.6 | 21.5 |
| nbody-modified | 7.88 K | 68.3 | 0.8 | 30.9 |
| fib | 7.88 K | 68.3 | 0.8 | 30.9 |

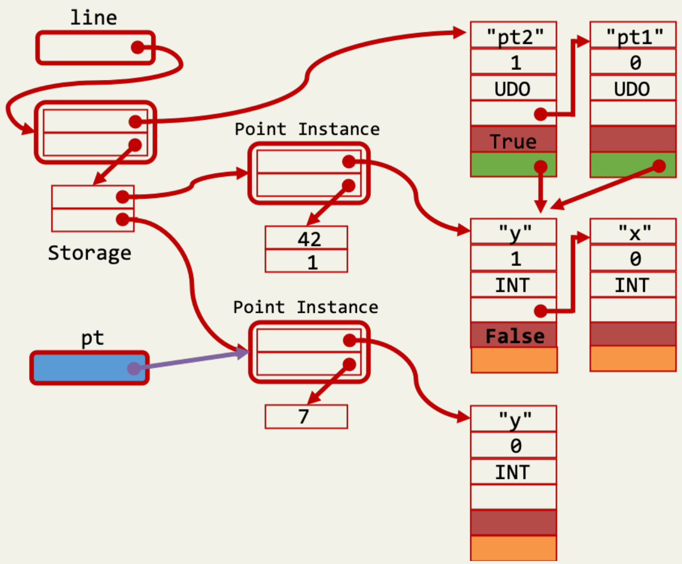
# ATTRIBUTE TYPE MONOMORPHISM → 75% OF ALL READS

CSL

| Benchmark | Attribute Reads: Total | Monomorphic Primitive (%) | Monomorphic User-Defined (%) | Polymorphic (%) |
|---|---|---|---|---|
| deltablue | 524.10 M | 56.6 | 41.7 | 1.7 |
| raytrace | 5.01 B | 2.7 | 9.6 | 87.4 |
| richards | 808.08 M | 64.5 | 7.5 | 24.6 |
| eparse | 20.34 M | 51.6 | 0.1 | 48.4 |
| telco | 376.50 M | 70.9 | 1.6 | 27.5 |
| float | 150.01 M | 100.0 | 0.0 | 0.0 |
| html5lib | 21.17 M | 70.0 | 5.7 | 24.4 |
| chaos | 538.39 M | 86.1 | 0.0 | 13.9 |
| pickle | 55.93 M | 100.0 | 0.0 | 0.0 |
| django | 32.48 M | 71.5 | 14.2 | 14.3 |
| sympy | 6.20 M | 87.3 | 0.0 | 12.6 |
| sympy-opt | 6.20 M | 86.8 | 0.0 | 13.1 |
| gcbench | 37.14 M | 0.0 | 0.0 | 100.0 |
| genshi-xml | 5.84 M | 98.0 | 1.7 | 0.3 |
| chameleon | 451.46 K | 84.0 | 0.4 | 15.6 |
| mako | 260.12 K | 73.7 | 17.5 | 8.8 |
| meteor-contest | 11.52 K | 77.9 | 0.6 | 21.5 |
| nbody-modified | 7.88 K | 68.3 | 0.8 | 30.9 |
| fib | 7.88 K | 68.3 | 0.8 | 30.9 |

Simple Type Freezing

Nested Type Freezing




# Type Freezing

Motivation

Background

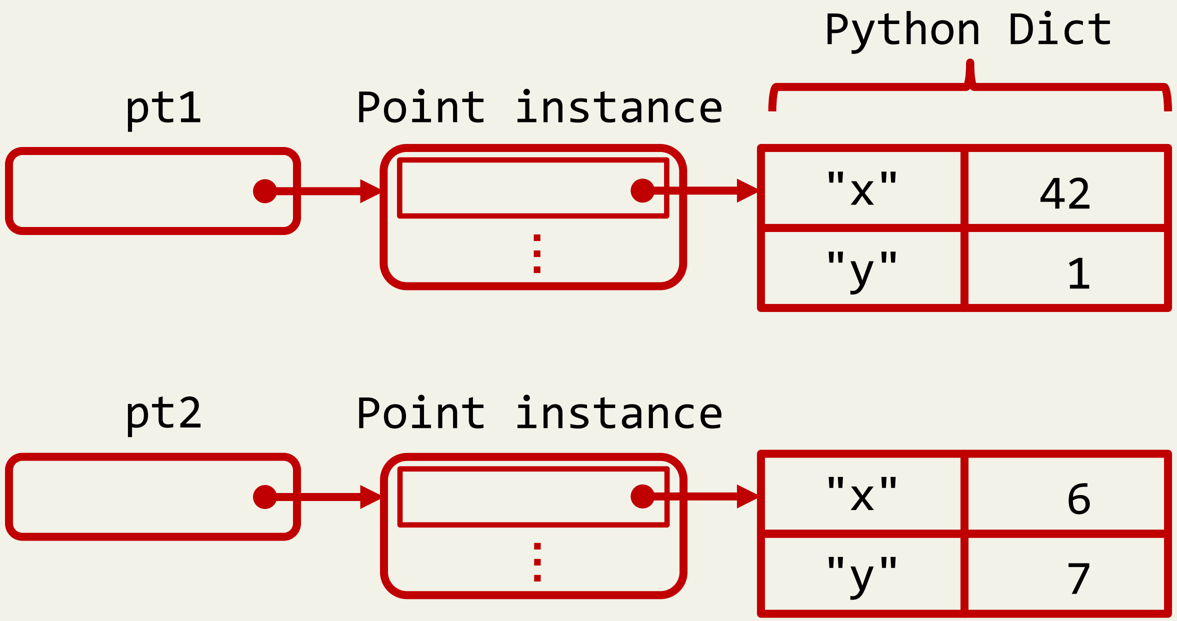
Simple Type Freezing

Nested Type Freezing

Evaluation

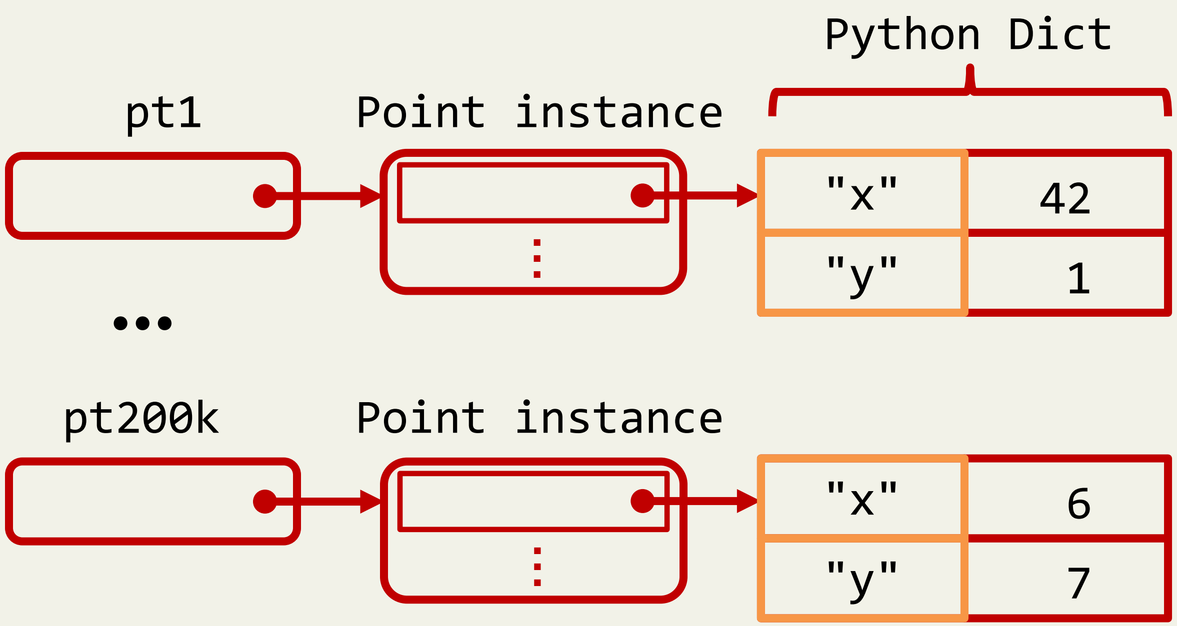
# User Defined Types - The CPython Way

```
class Point( object ):
  def __init__( self, x, y ):
    self.x = x
    self.y = y

pt1 = Point( 42, 1 )
pt2 = Point(  6, 7 )
```




- Attributes can be added to, or removed from, an instance dynamically: It is necessary to keep track of each instance's attribute list
- CPython associates a complex and memory hungry Dict with each instance

```
class Point( object ):
  def __init__( self, x, y ):
    self.x = x
    self.y = y

pt1    = Point( 42, 1 )
. . .
pt200k = Point(  6, 7 )
```



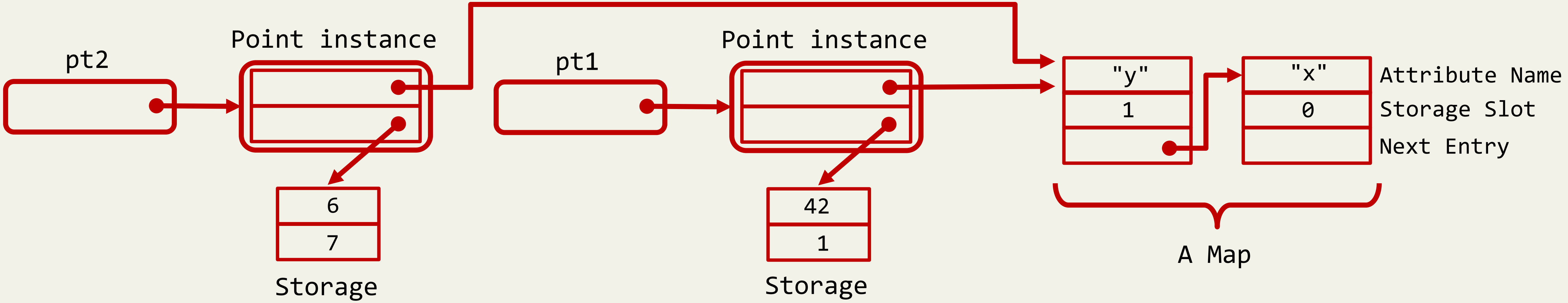

- Attributes can be added to, or removed from, an instance dynamically: It is necessary to keep track of each instance's attribute list

- CPython associates a complex and memory hungry Dict with each instance

# User Defined Types - The PyPy Way

```
class Point( object ):
  def __init__( self, x, y ):
    self.x = x
    self.y = y

pt1 = Point( 42, 1 )
pt2 = Point(  6, 7 )
```

- Instances are likely to have the same set of attributes: modern JIT compilers usually implement an optimization called **Maps** (also known as **Hidden Classes** or **Shapes**)
- User-defined types are structural: an instance's map determines its type



```
class Point( object ):
  def __init__( self, x, y ):
    self.x = x
    self.y = y

class Line( object ):
  def __init__( self, pt1, pt2 ):
    self.pt1 = pt1
    self.pt2 = pt2

def create_lines( n ):
  lines = []
  for i in range( n ):
    pt1 = Point( i, n-i )
    pt2 = Point( 2*i-n, i-n )
    lines.append( Line( pt1, pt2 ) )
  return lines
```

```
def total_length( n, lines ):
  length = 0
  i = 0
  while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = ( pt1.x - pt2.x ) ** 2
    b_side = ( pt1.y - pt2.y ) ** 2
    length += math.sqrt( a_side + b_side )
  return length
```

# Running Example

CSL

```
def total_length( n, lines ):
  length = 0
  i = 0
  while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = ( pt1.x - pt2.x ) ** 2
    b_side = ( pt1.y - pt2.y ) ** 2
    length += math.sqrt( a_side + b_side )
  return length
```

```
p7 = get_array_item( p0, i1 )     # line = lines[i]
guard_class( p7, W_ObjectObject ) #
```

# Running Example

```
def total_length( n, lines ):
  length = 0
  i = 0
  while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = ( pt1.x - pt2.x ) ** 2
    b_side = ( pt1.y - pt2.y ) ** 2
    length += math.sqrt( a_side + b_side )
  return length
```

```
p7 = get_array_item( p0, i1 )     # line = lines[i]
guard_class( p7, W_ObjectObject ) #

p8 = get( p7, Map )               # pt1 = line.pt1
guard_value( p8, Map of Line )    #
guard_not_invalidated()           #
p9 = get( p7, slot0 )             #
guard_class( p9, W_ObjectObject ) #
```

# Running Example

```
def total_length( n, lines ):
  length = 0
  i = 0
  while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = ( pt1.x - pt2.x ) ** 2
    b_side = ( pt1.y - pt2.y ) ** 2
    length += math.sqrt( a_side + b_side )
  return length
```

```
p7 = get_array_item( p0, i1 )     # line = lines[i]
guard_class( p7, W_ObjectObject ) #

p8 = get( p7, Map )               # pt1 = line.pt1
guard_value( p8, Map of Line )    #
guard_not_invalidated()           #
p9 = get( p7, slot0 )             #
guard_class( p9, W_ObjectObject ) #

p10 = get( p7, slot1 )            # pt2 = line.pt2
guard_class( p7, W_ObjectObject ) #
```

# Running Example

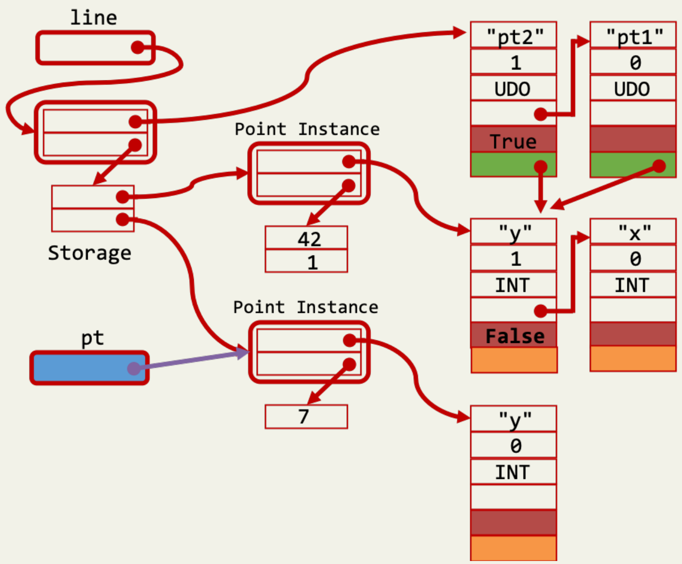
```
def total_length( n, lines ):
  length = 0
  i = 0
  while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = ( pt1.x - pt2.x ) ** 2
    b_side = ( pt1.y - pt2.y ) ** 2
    length += math.sqrt( a_side + b_side )
  return length
```

```
p7 = get_array_item( p0, i1 )     # line = lines[i]
guard_class( p7, W_ObjectObject ) #

p8 = get( p7, Map )               # pt1 = line.pt1
guard_value( p8, Map of Line )    #
guard_not_invalidated()           #
p9 = get( p7, slot0 )             #
guard_class( p9, W_ObjectObject ) #

p10 = get( p7, slot1 )            # pt2 = line.pt2
guard_class( p7, W_ObjectObject ) #

p11 = get( p9, Map )              # pt1.x
guard_value( p11, Map of Point)   #
p12 = get( p9, slot0 )            #
guard_class( p12, W_IntObject )   #

p13 = get( p10, Map )             # pt2.x
guard_value( p13, Map of Point)   #
p14 = get( p10, slot0 )           #
guard_class( p14, W_IntObject )   #
```

# Running Example

```python
def total_length( n, lines ):
  length = 0
  i = 0
  while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = ( pt1.x - pt2.x ) ** 2
    b_side = ( pt1.y - pt2.y ) ** 2
    length += math.sqrt( a_side + b_side )
  return length
```

```
p7 = get_array_item( p0, i1 )     # line = lines[i]
guard_class( p7, W_ObjectObject ) #

p8 = get( p7, Map )               # pt1 = line.pt1
guard_value( p8, Map of Line )    #
guard_not_invalidated()           #
p9 = get( p7, slot0 )             #
guard_class( p9, W_ObjectObject ) #

p10 = get( p7, slot1 )            # pt2 = line.pt2
guard_class( p7, W_ObjectObject ) #

p11 = get( p9, Map )              # pt1.x
guard_value( p11, Map of Point)   #
p12 = get( p9, slot0 )            #
guard_class( p12, W_IntObject )   #

p13 = get( p10, Map )             # pt2.x
guard_value( p13, Map of Point)   #
p14 = get( p10, slot0 )           #
guard_class( p14, W_IntObject )   #
. . .
p19 = get( p9, slot1 )            # pt1.y
guard_class( p19, W_IntObject )   #

p20 = get( p10, slot1 )           # pt2.y
guard_class( p20, W_IntObject )   #
. . .
```




# Type Freezing

Motivation

Background

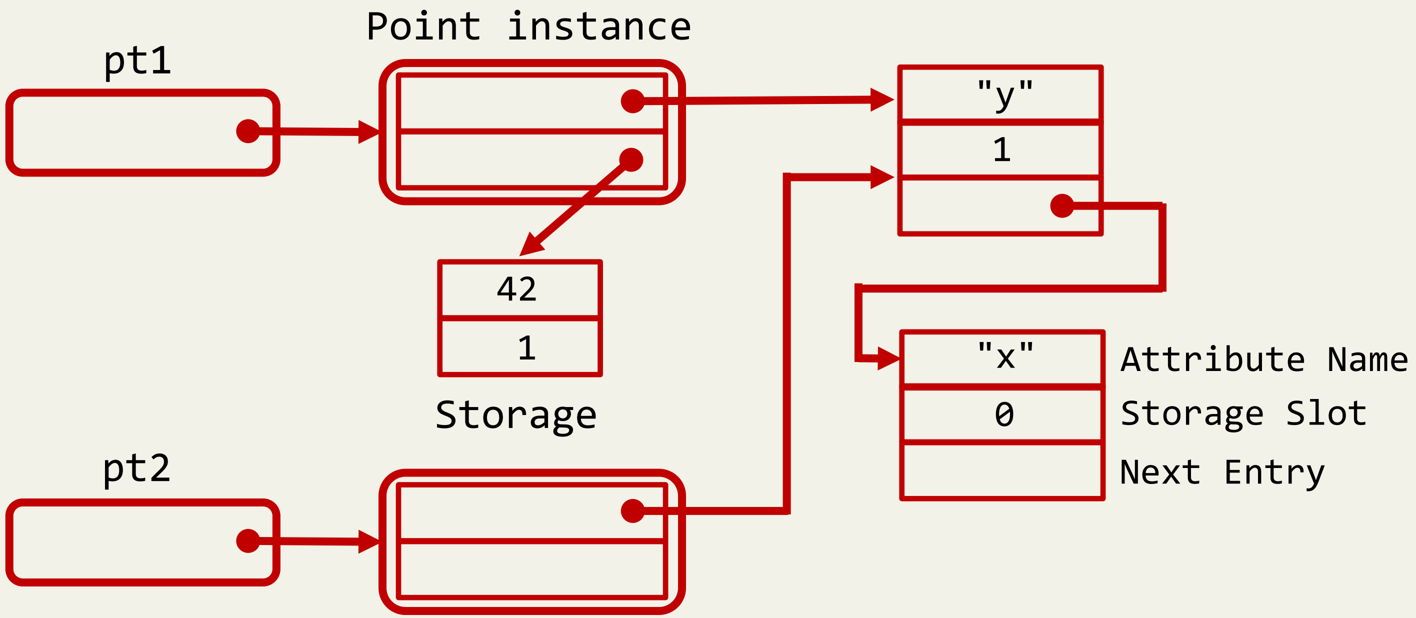
Simple Type Freezing

Nested Type Freezing

Evaluation

# Technique: Simple Type Freezing

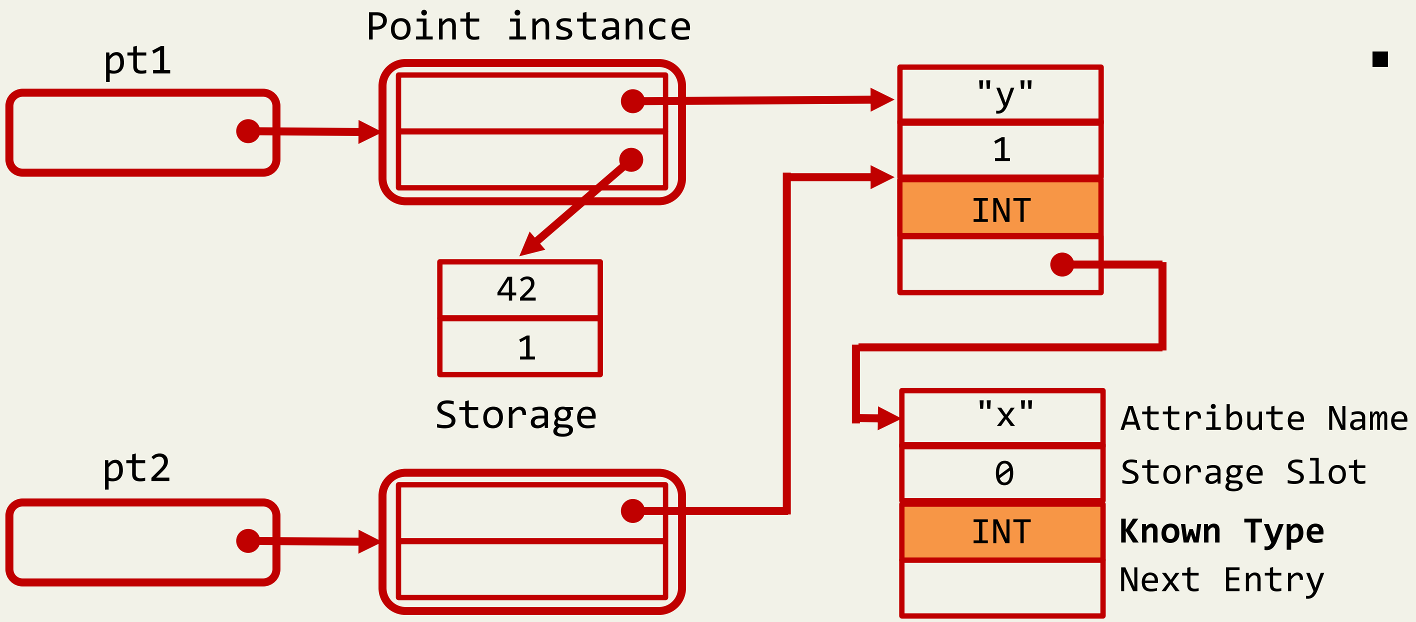
```
class Point( object ):
  def __init__( self, x, y ):
    self.x = x
    self.y = y

pt1 = Point( 42, 1 )
pt2 = Point(  6, 7 )
```




- To exploit attribute type monomorphism, we need to keep track of which attributes are type monomorphic

# Technique: Simple Type Freezing

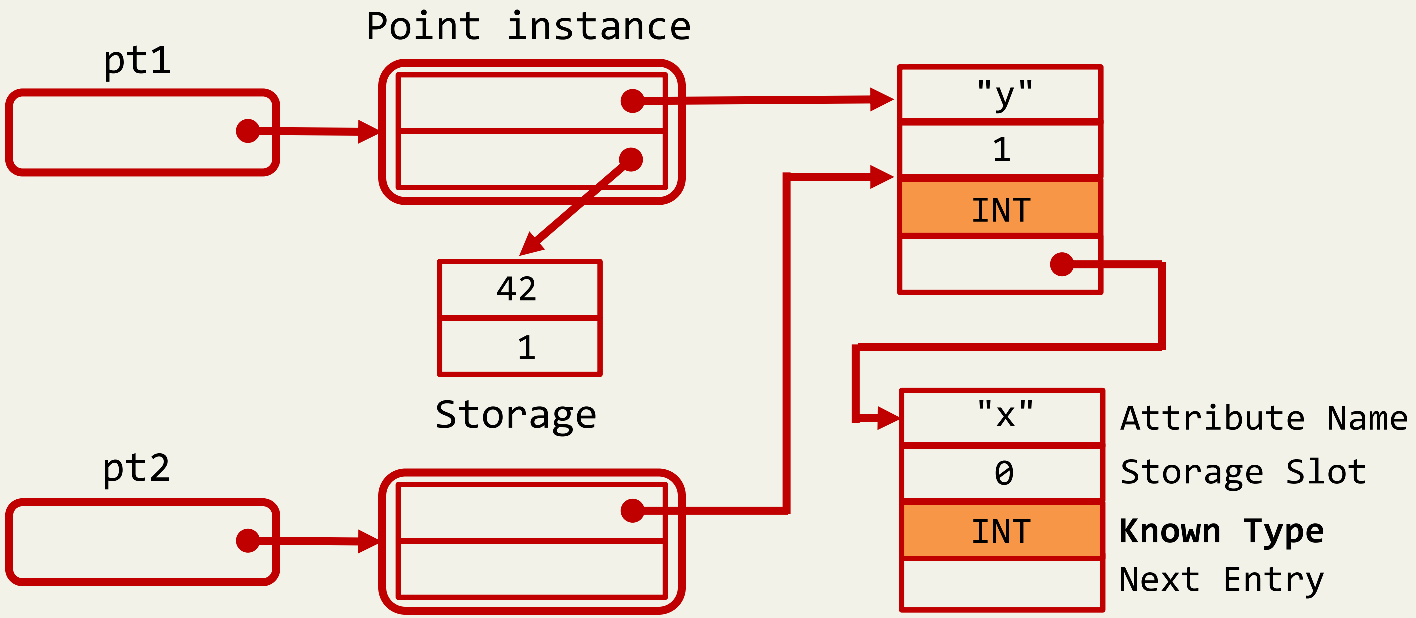
```
class Point( object ):
  def __init__( self, x, y ):
    self.x = x
    self.y = y

pt1 = Point( 42, 1 )
pt2 = Point(  6, 7 )
```




- To exploit attribute type monomorphism, we need to keep track of which attributes are type monomorphic
- We "freeze" the type information of attributes into the map with an auxiliary field, **known type**
- With this extra info, **unmodified** PyPy JIT compiler is able to prove the type of these attributes at compiling time and eliminate type guards on them

# Benefit: Eliminating Type Guards

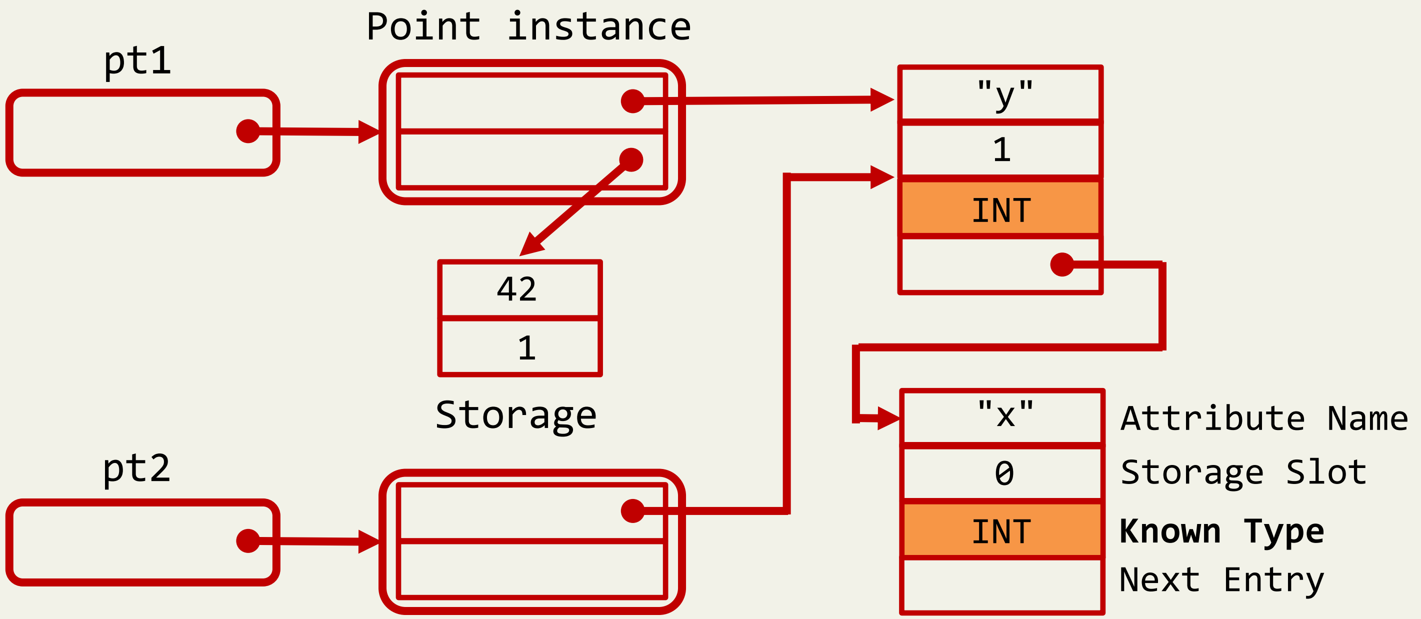
```
while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = ( pt1.x - pt2.x ) ** 2
    b_side = ( pt1.y - pt2.y ) ** 2
    length += math.sqrt( a_side + b_side )
```




```
p7 = get_array_item( p0, i1 )     # line = lines[i]
guard_class( p7, W_ObjectObject ) #

p8 = get( p7, Map )               # pt1 = line.pt1
guard_value( p8, Map of Line )    #
guard_not_invalidated()           #
p9 = get( p7, slot0 )             #
guard_class( p9, W_ObjectObject ) #

p10 = get( p7, slot1 )            # pt2 = line.pt2
guard_class( p7, W_ObjectObject ) #

p11 = get( p9, Map )              # pt1.x
guard_value( p11, Map of Point)   #
p12 = get( p9, slot0 )            #
guard_class( p12, W_IntObject )   #

p13 = get( p10, Map )             # pt2.x
guard_value( p13, Map of Point)   #
p14 = get( p10, slot0 )           #
guard_class( p14, W_IntObject )   #
. . .
p19 = get( p9, slot1 )            # pt1.y
guard_class( p19, W_IntObject )   #

p20 = get( p10, slot1 )           # pt2.y
guard_class( p20, W_IntObject )   #
. . .
```

# Benefit: Eliminating Type Guards

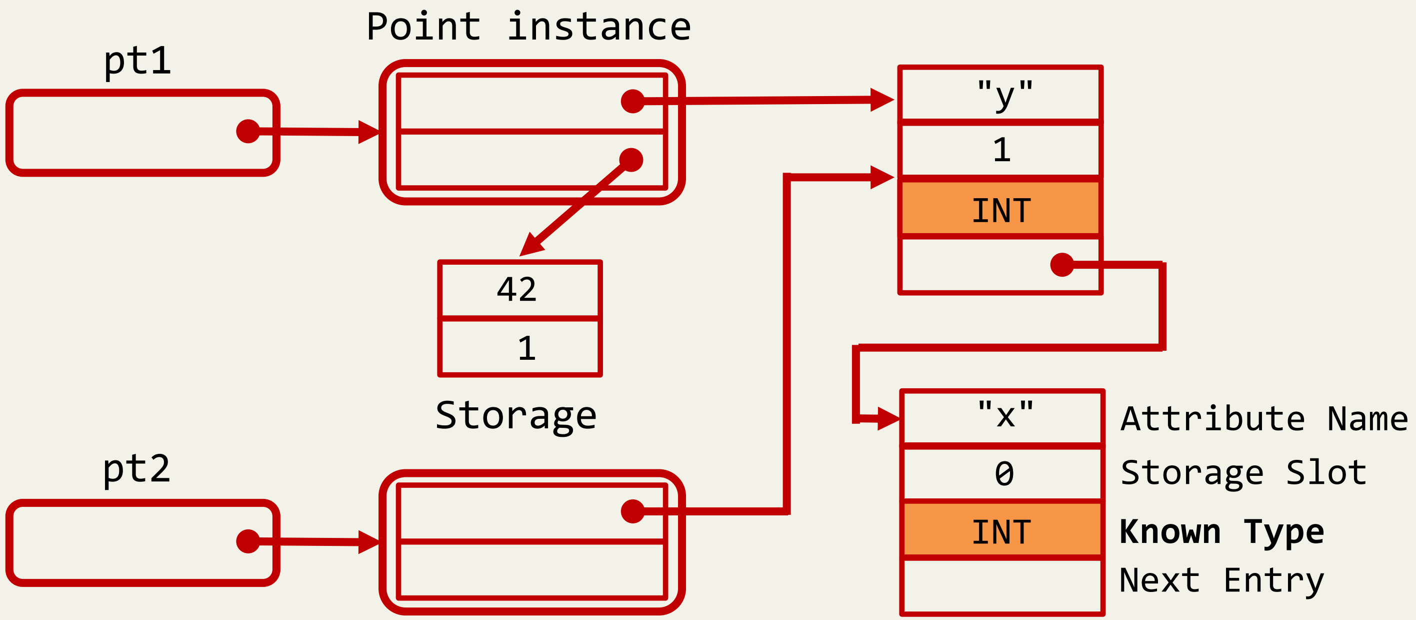
```
while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = ( pt1.x - pt2.x ) ** 2
    b_side = ( pt1.y - pt2.y ) ** 2
    length += math.sqrt( a_side + b_side )
```




```
p7 = get_array_item( p0, i1 )       # line = lines[i]
guard_class( p7, W_ObjectObject ) #

p8 = get( p7, Map )                 # pt1 = line.pt1
guard_value( p8, Map of Line )      #
guard_not_invalidated()             #
p9 = get( p7, slot0 )               #
guard_class( p9, W_ObjectObject ) #

p10 = get( p7, slot1 )              # pt2 = line.pt2
guard_class( p7, W_ObjectObject ) #

p11 = get( p9, Map )                # pt1.x
guard_value( p11, Map of Point)     #
p12 = get( p9, slot0 )              #
guard_class( p12, W_IntObject )     #

p13 = get( p10, Map )               # pt2.x
guard_value( p13, Map of Point)     #
p14 = get( p10, slot0 )             #
guard_class( p14, W_IntObject )     #
. . .
p19 = get( p9, slot1 )              # pt1.y
guard_class( p19, W_IntObject )     #

p20 = get( p10, slot1 )             # pt2.y
guard_class( p20, W_IntObject )     #
. . .
```

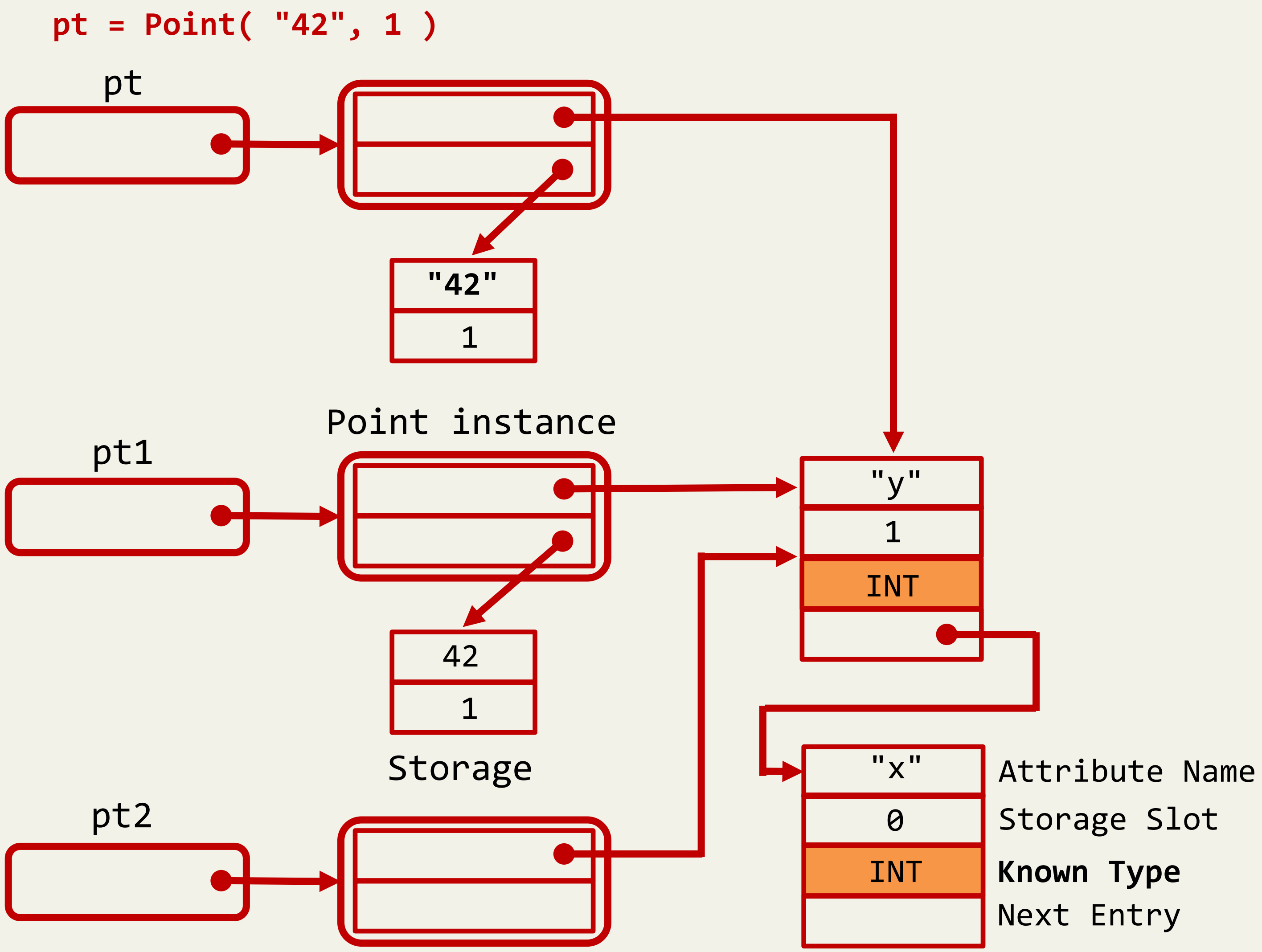
# CHALLENGE: ATTRIBUTES MAY BECOME POLYMORPHIC

```
while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = ( pt1.x - pt2.x ) ** 2
    b_side = ( pt1.y - pt2.y ) ** 2
    length += math.sqrt( a_side + b_side )
pt = Point( "42", 1 )
```




```
p7 = get_array_item( p0, i1 )      # line = lines[i]
guard_class( p7, W_ObjectObject ) #

p8 = get( p7, Map )                # pt1 = line.pt1
guard_value( p8, Map of Line )     #
guard_not_invalidated()            #
p9 = get( p7, slot0 )              #

p10 = get( p7, slot1 )             # pt2 = line.pt2

p11 = get( p9, Map )               # pt1.x
guard_value( p11, Map of Point)    #
p12 = get( p9, slot0 )             #

p13 = get( p10, Map )              # pt2.x
guard_value( p13, Map of Point)    #
p14 = get( p10, slot0 )            #

. . .
p19 = get( p9, slot1 )             # pt1.y

p20 = get( p10, slot1 )            # pt2.y

. . .
```

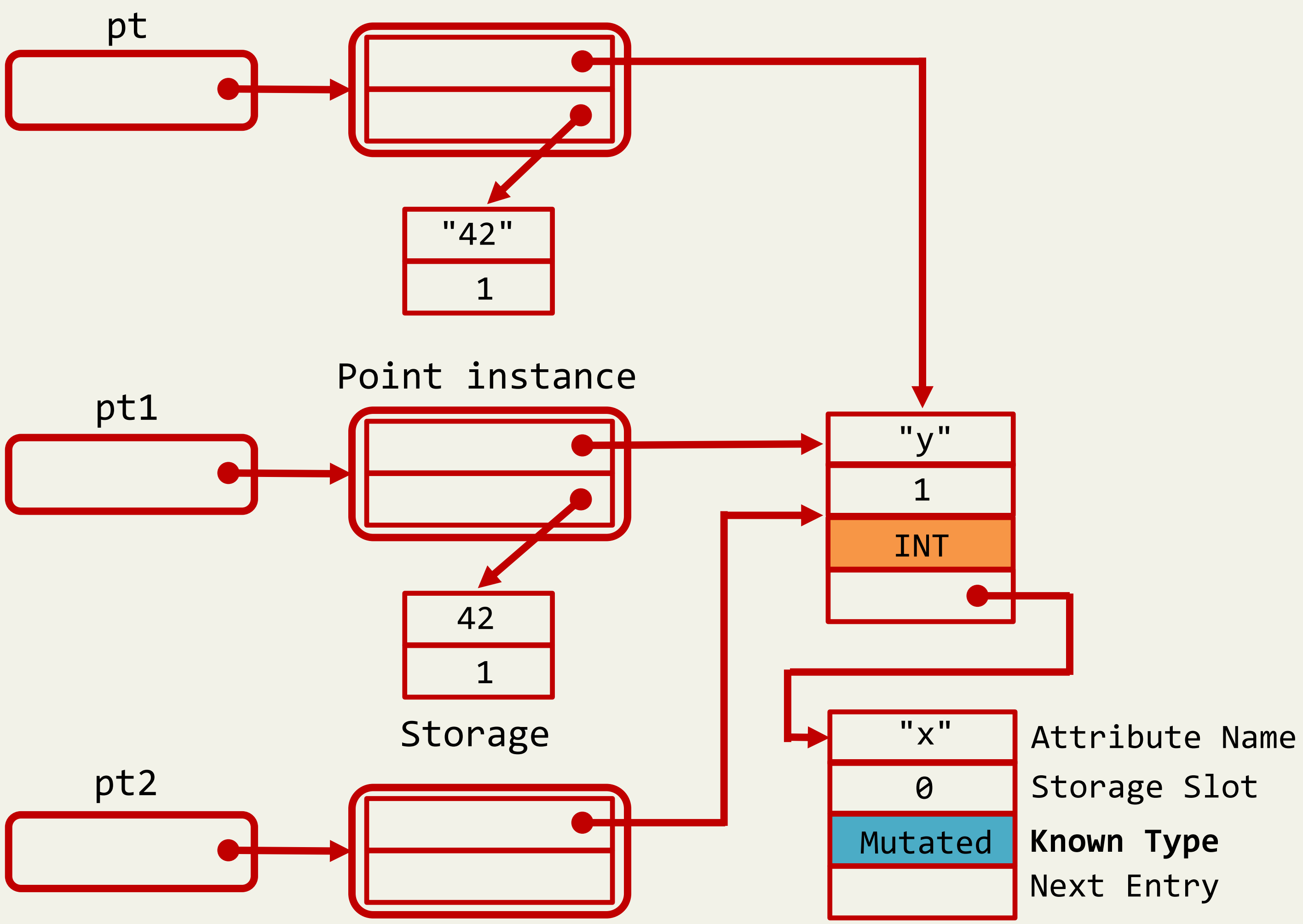
# CHALLENGE: ATTRIBUTES MAY BECOME POLYMORPHIC



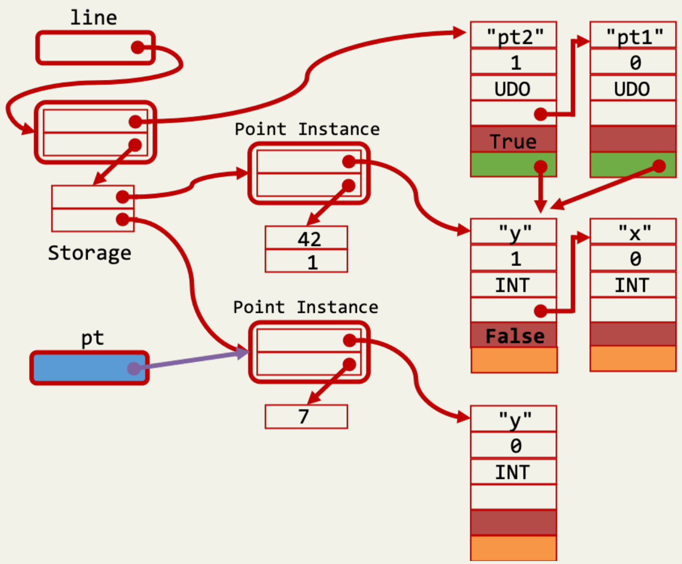

```
p7 = get_array_item( p0, i1 )      # line = lines[i]
guard_class( p7, W_ObjectObject ) #

p8 = get( p7, Map )                # pt1 = line.pt1
guard_value( p8, Map of Line )     #
guard_not_invalidated()            #
p9 = get( p7, slot0 )              #

p10 = get( p7, slot1 )             # pt2 = line.pt2

p11 = get( p9, Map )               # pt1.x
guard_value( p11, Map of Point)    #
p12 = get( p9, slot0 )             #

p13 = get( p10, Map )              # pt2.x
guard_value( p13, Map of Point)    #
p14 = get( p10, slot0 )            #

. . .
p19 = get( p9, slot1 )             # pt1.y

p20 = get( p10, slot1 )            # pt2.y

. . .
```

# Solution: Invalid Traces Through Quasi-Immutable

```
pt = Point( "42", 1 )
```




```
p7 = get_array_item( p0, i1 )      # line = lines[i]
guard_class( p7, W_ObjectObject )  #
p8 = get( p7, Map )                # pt1 = line.pt1
guard_value( p8, Map of Line )     #
guard_not_invalidated()            #
p9 = get( p7, slot0 )              #

p10 = get( p7, slot1 )             # pt2 = line.pt2

p11 = get( p9, Map )               # pt1.x
guard_value( p11, Map of Point )   #
p12 = get( p9, slot0 )             #

p13 = get( p10, Map )              # pt2.x
guard_value( p13, Map of Point)    #
p14 = get( p10, slot0 )            #

. . .
p19 = get( p9, slot1 )             # pt1.y

p20 = get( p10, slot1 )            # pt2.y

. . .
```




# Type Freezing

Motivation

Background

Simple Type Freezing

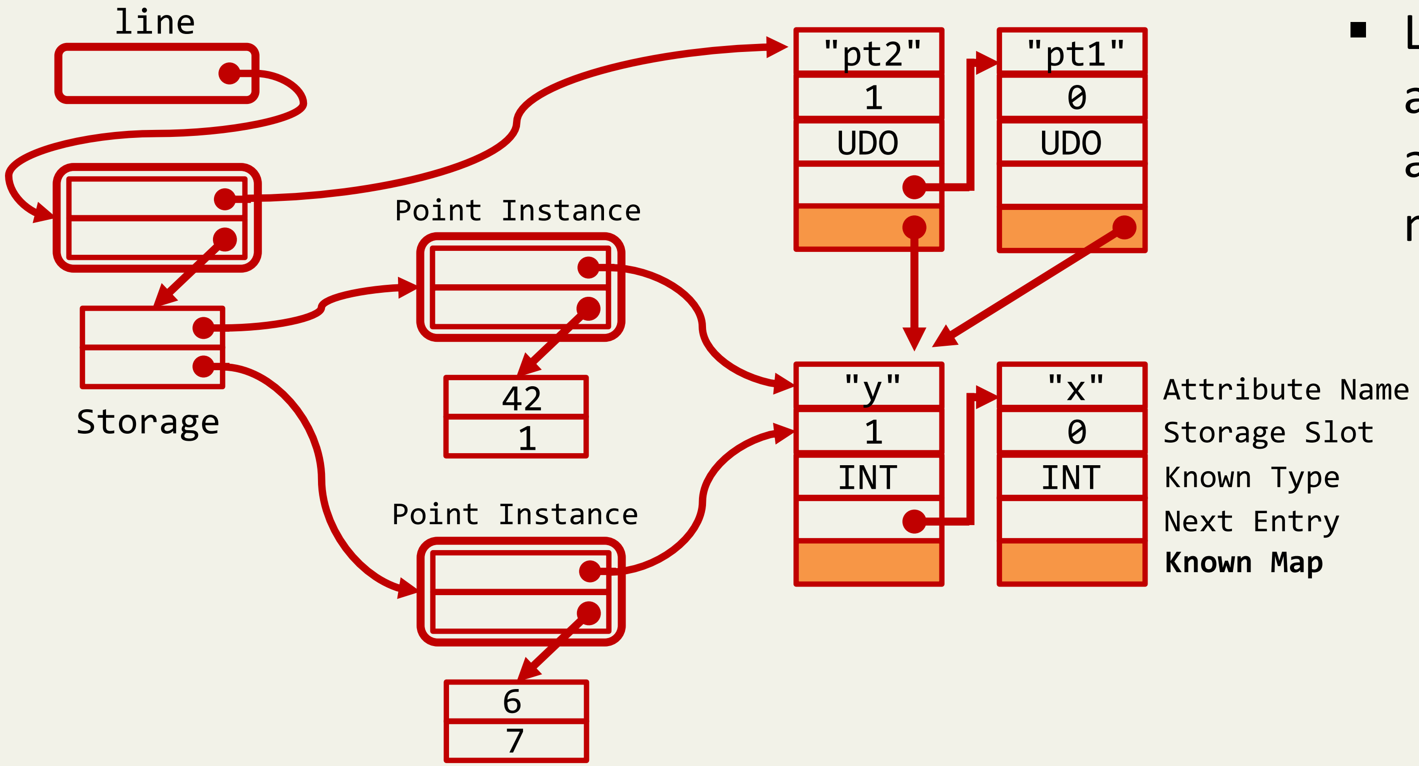
Nested Type Freezing

Evaluation

# TECHNIQUE: NESTED TYPE FREEZING

CSL

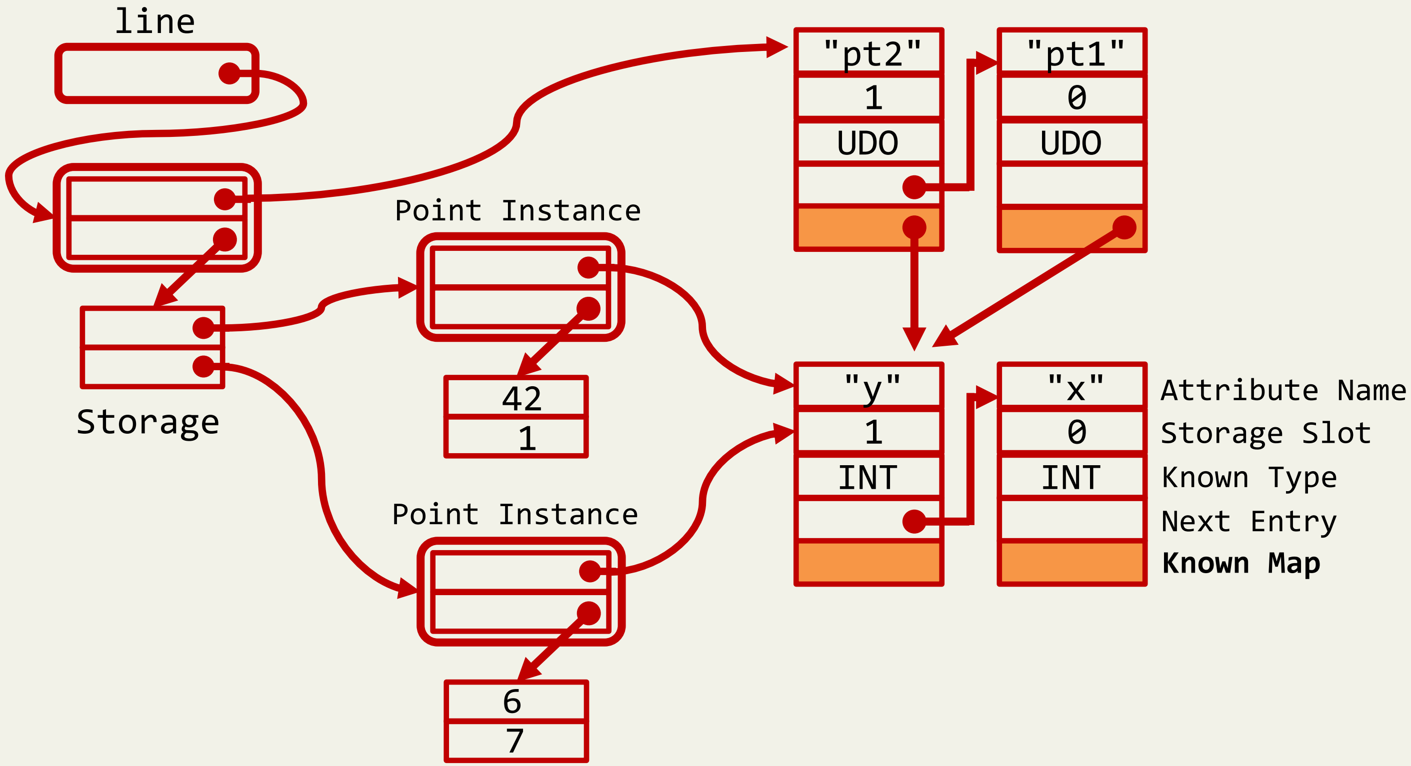
```
class Line( object ):
  def __init__( self, pt1, pt2 ):
    self.pt1 = pt1
    self.pt2 = pt2

line = Line( Point(42, 1), Point(6, 7) )
```




- If a type monomorphic attribute stores another kind of user-defined objects, knowing the type of top-level object implicitly tells the layout of nested objects
- Like the case in simple type freezing, we associate another quasi-immutable auxiliary field, **known map**, with each map entry.

# Benefit: Removing Guards on Maps

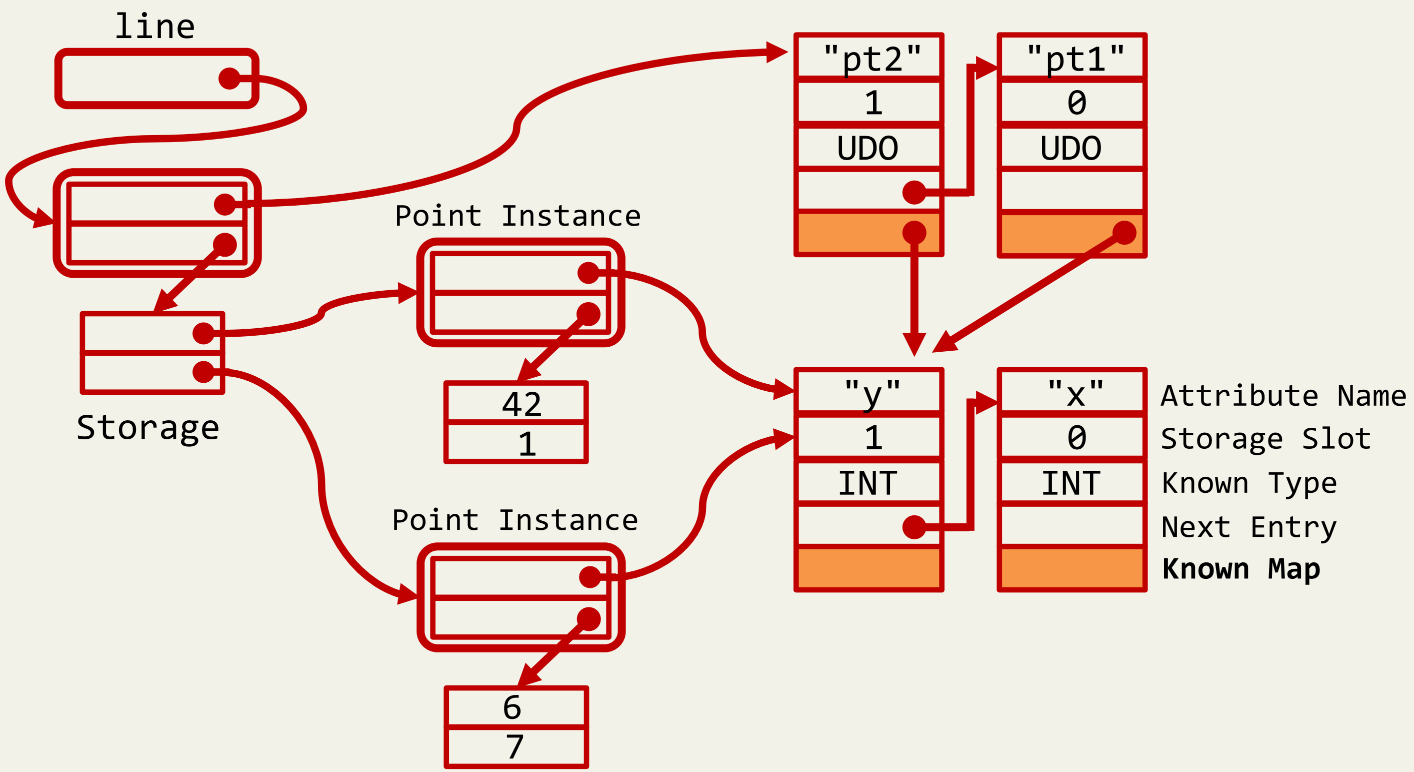
```
while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = ( pt1.x - pt2.x ) ** 2
    b_side = ( pt1.y - pt2.y ) ** 2
    length += math.sqrt( a_side + b_side )
```




```
p7 = get_array_item( p0, i1 )       # line = lines[i]
guard_class( p7, W_ObjectObject )  #

p8 = get( p7, Map )                 # pt1 = line.pt1
guard_value( p8, Map of Line )      #
guard_not_invalidated()             #
p9 = get( p7, slot0 )               #

p10 = get( p7, slot1 )              # pt2 = line.pt2

p11 = get( p9, Map )                # pt1.x
guard_value( p11, Map of Point)     #
p12 = get( p9, slot0 )              #

p13 = get( p10, Map )               # pt2.x
guard_value( p13, Map of Point)     #
p14 = get( p10, slot0 )             #

. . .
p19 = get( p9, slot1 )              # pt1.y

p20 = get( p10, slot1 )             # pt2.y

. . .
```

# Benefit: Removing Guards on Maps

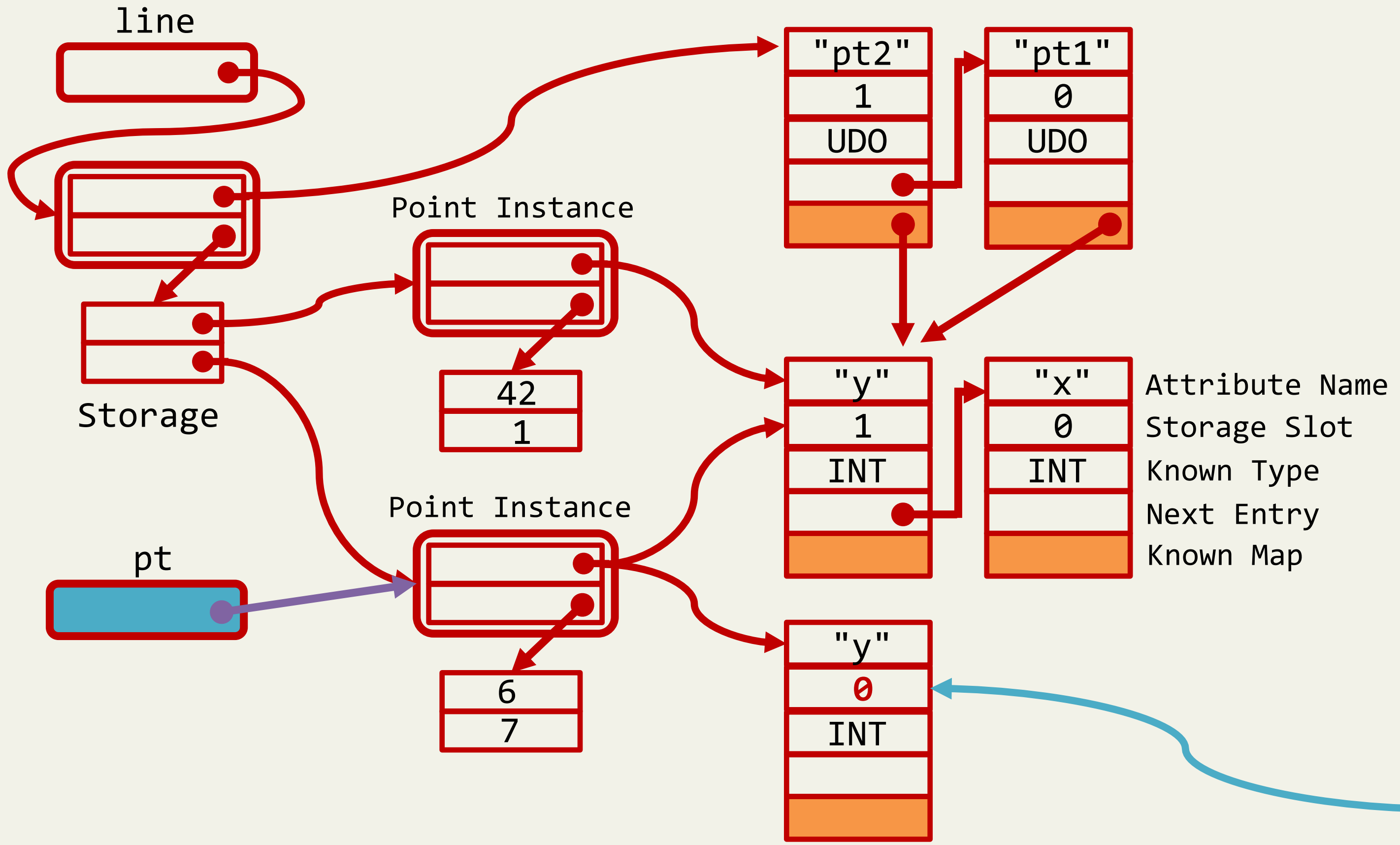
```
while( i < n ):
    line = lines[i]
    pt1 = line.pt1
    pt2 = line.pt2
    a_side = ( pt1.x - pt2.x ) ** 2
    b_side = ( pt1.y - pt2.y ) ** 2
    length += math.sqrt( a_side + b_side )
```




```
p7 = get_array_item( p0, i1 )      # line = lines[i]
guard_class( p7, W_ObjectObject ) #

p8 = get( p7, Map )                # pt1 = line.pt1
guard_value( p8, Map of Line )     #
guard_not_invalidated()            #
p9 = get( p7, slot0 )              #

p10 = get( p7, slot1 )             # pt2 = line.pt2

p11 = get( p9, Map )               # pt1.x
guard_value( p11, Map of Point)    #
p12 = get( p9, slot0 )             #

p13 = get( p10, Map )              # pt2.x
guard_value( p13, Map of Point)    #
p14 = get( p10, slot0 )            #

. . .
p19 = get( p9, slot1 )             # pt1.y

p20 = get( p10, slot1 )            # pt2.y

. . .
```

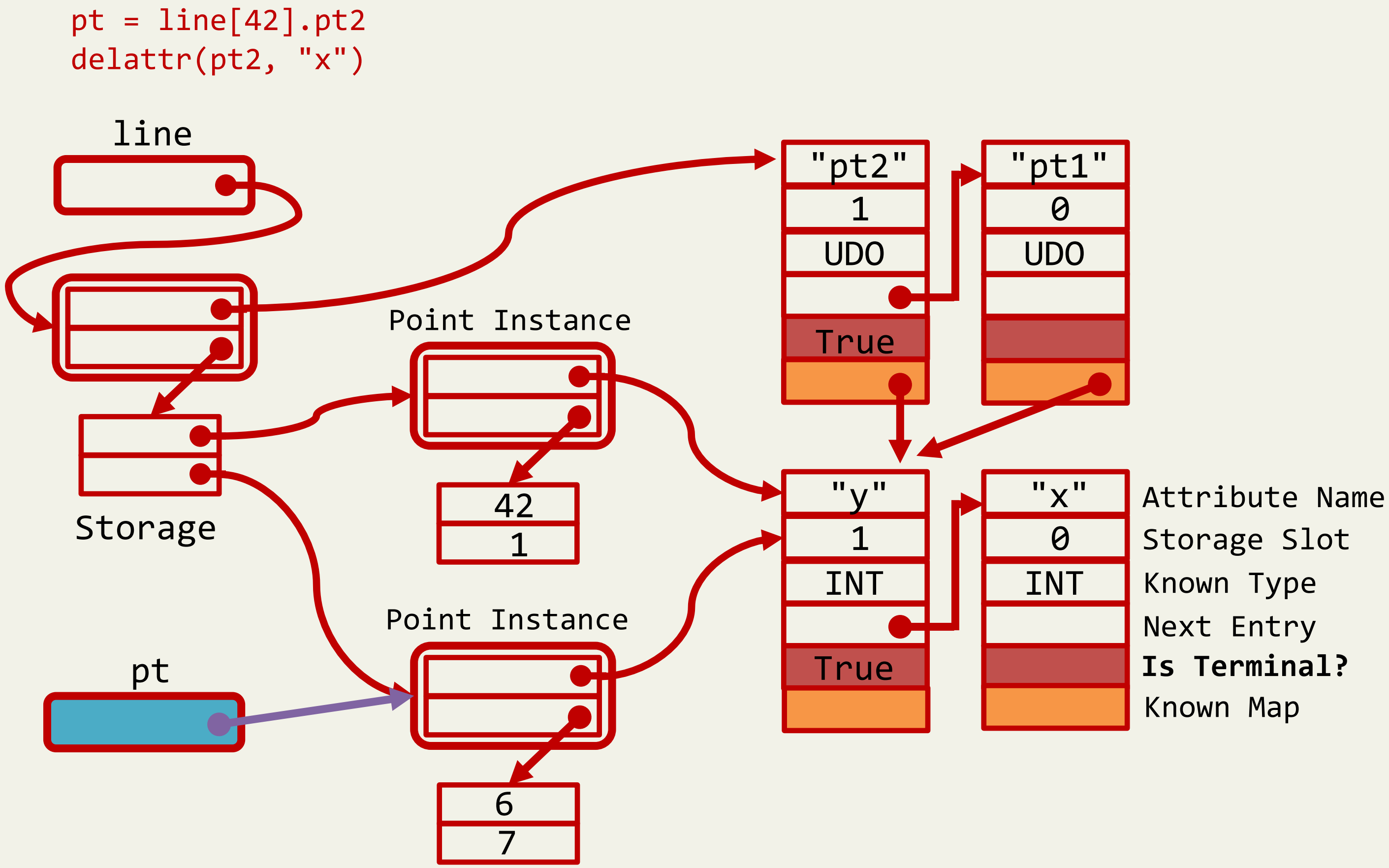
# Challenge: Two References To A Single Object

```
pt = line[42].pt2
delattr(pt2, "x")
```




```
p7 = get_array_item( p0, i1 )       # line = lines[i]
guard_class( p7, W_ObjectObject ) #

p8 = get( p7, Map )                 # pt1 = line.pt1
guard_value( p8, Map of Line )      #
guard_not_invalidated()             #
p9 = get( p7, slot0 )               #

p10 = get( p7, slot1 )              # pt2 = line.pt2

p12 = get( p9, slot0 )              #

p14 = get( p10, slot0 )             #

. . .
p19 = get( p9, slot1 )              # pt1.y

p20 = get( p10, slot1 )             # pt2.y

. . .
```

# SOLUTION: TERMINAL MAP

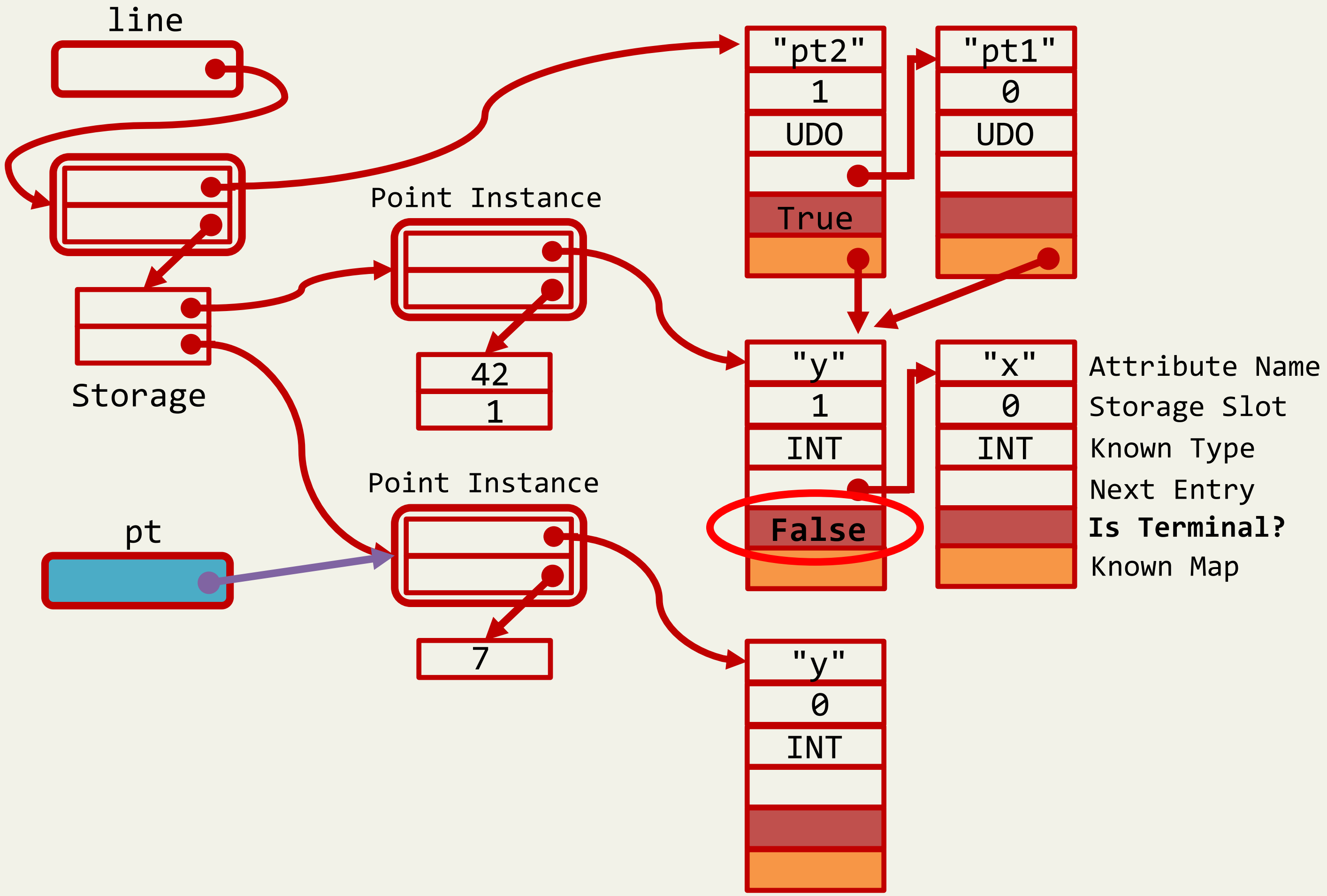



- A map is said to be **terminal**, if and only if none of the instances that point to this map have added or removed any attributes

```
p7 = get_array_item( p0, i1 )         # line = lines[i]
guard_class( p7, W_ObjectObject )     #

p8 = get( p7, Map )                   # pt1 = line.pt1
guard_value( p8, Map of Line )        #
guard_not_invalidated()               #
p9 = get( p7, slot0 )                 #

p10 = get( p7, slot1 )                # pt2 = line.pt2

p12 = get( p9, slot0 )                #

p14 = get( p10, slot0 )               #

. . .
p19 = get( p9, slot1 )                # pt1.y
p20 = get( p10, slot1 )               # pt2.y
. . .
```

# SOLUTION: TERMINAL MAP

```
pt = line[42].pt2
delattr(pt2, "x")
```




- A map is said to be **terminal**, if and only if none of the instances that point to this map have added or removed any attributes

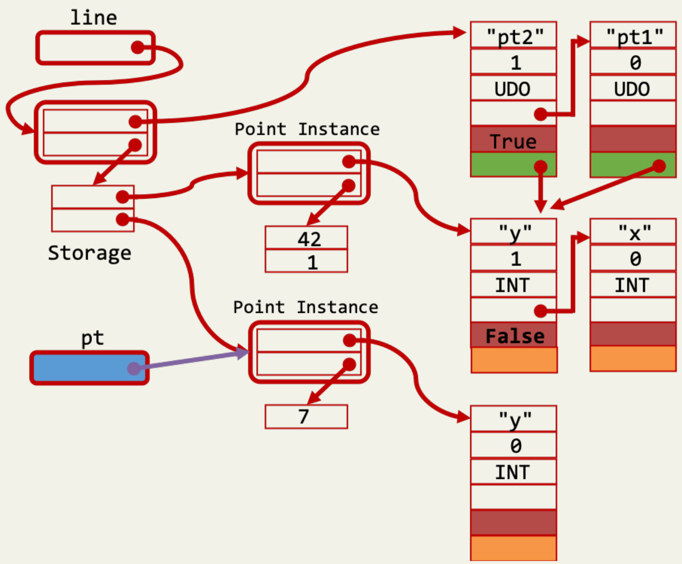
```
p7 = get_array_item( p0, i1 )      # line = lines[i]
guard_class( p7, W_ObjectObject ) #
p8 = get( p7, Ma                   # pt1 = line.pt1
guard_value( p8, M    of Line      #
guard_not_invalidat    )           #
p9 = get( p7, slot0  )             #
p10 = get( p7, slot1 )             # pt2 = line.pt2
p12 = get( p9, slot0  )            #
p14 = get( p10, slo    )           #
. . .
p19 = get( p9, s     )             # pt1.y
p20 = get( p10, slot1 )            # pt2.y
. . .
```

# Write Time Type Checking

- We move runtime type checking from **read time** to **write time** for two reasons
  - There are generally more reads than writes
  - Certain write time type checking can be optimized away, since type of the value to be written is known at JIT compile time
- We used the running example as a micro-benchmark
  - Type Freezing saves up to 30% of dynamic instructions
  - Type Freezing improves performance by up to 6%

| Benchmark | Attribute Reads | Attribute Writes | Reads/Writes |
|---|---|---|---|
| deltablue | 524.10 M | 107.53 M | 4.9 |
| raytrace | 5.01 B | 1.23 B | 4.1 |
| richards | 808.08 M | 297.25 M | 2.7 |
| eparse | 20.34 M | 4.12 M | 4.9 |
| telco | 376.50 M | 103.42 M | 3.6 |
| float | 150.01 M | 90.00 M | 1.7 |
| html5lib | 21.17 M | 4.88 M | 4.3 |
| chaos | 538.39 M | 110.09 M | 4.9 |
| pickle | 55.93 M | 452.44 K | 123.6 |
| django | 32.48 M | 26.67 K | 1217.5 |
| sympy | 6.20 M | 1.44 M | 4.3 |
| sympy-opt | 6.20 M | 1.46 M | 4.2 |
| gcbench | 37.14 M | 190.47 M | 0.2 |
| genshi-xml | 5.84 M | 11.71 K | 498.7 |
| chameleon | 451.46 K | 488.86 K | 0.9 |
| mako | 260.12 K | 109.34 K | 2.4 |
| meteor-contest | 11.52 K | 4.67 K | 2.5 |
| nbody-modified | 7.88 K | 2.27 K | 3.5 |
| fib | 7.88 K | 2.27 K | 3.5 |

```
def create_lines( n ):
  lines = []
  for i in range( n ):
    pt1 = Point( i, n-i )
    pt2 = Point( 2*i-n, i-n )
    lines.append( Line( pt1, pt2 ) )
  return lines
```




# Type Freezing

Motivation

Background

Simple Type Freezing

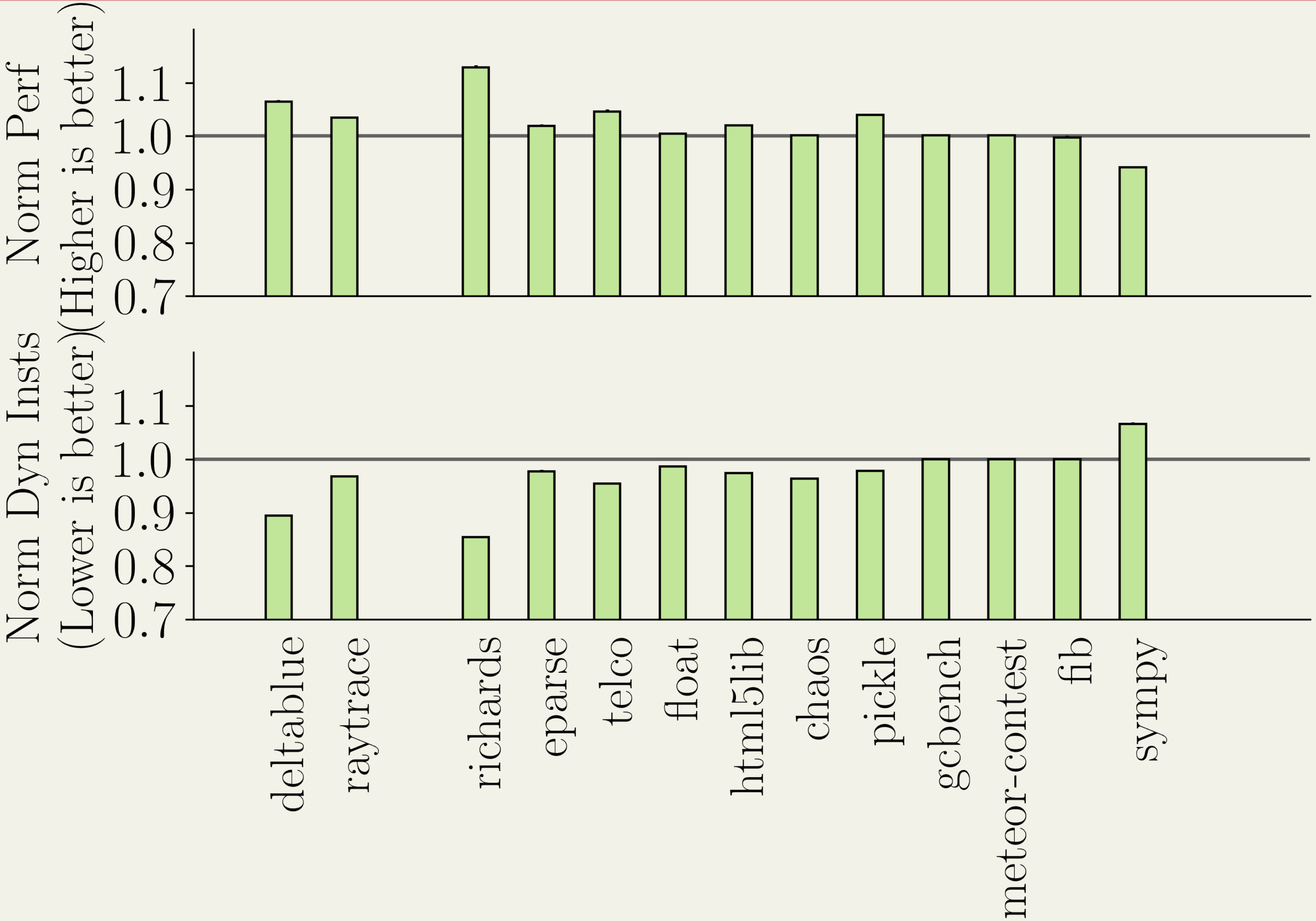
Nested Type Freezing

Evaluation

# PyPy Benchmark Suite

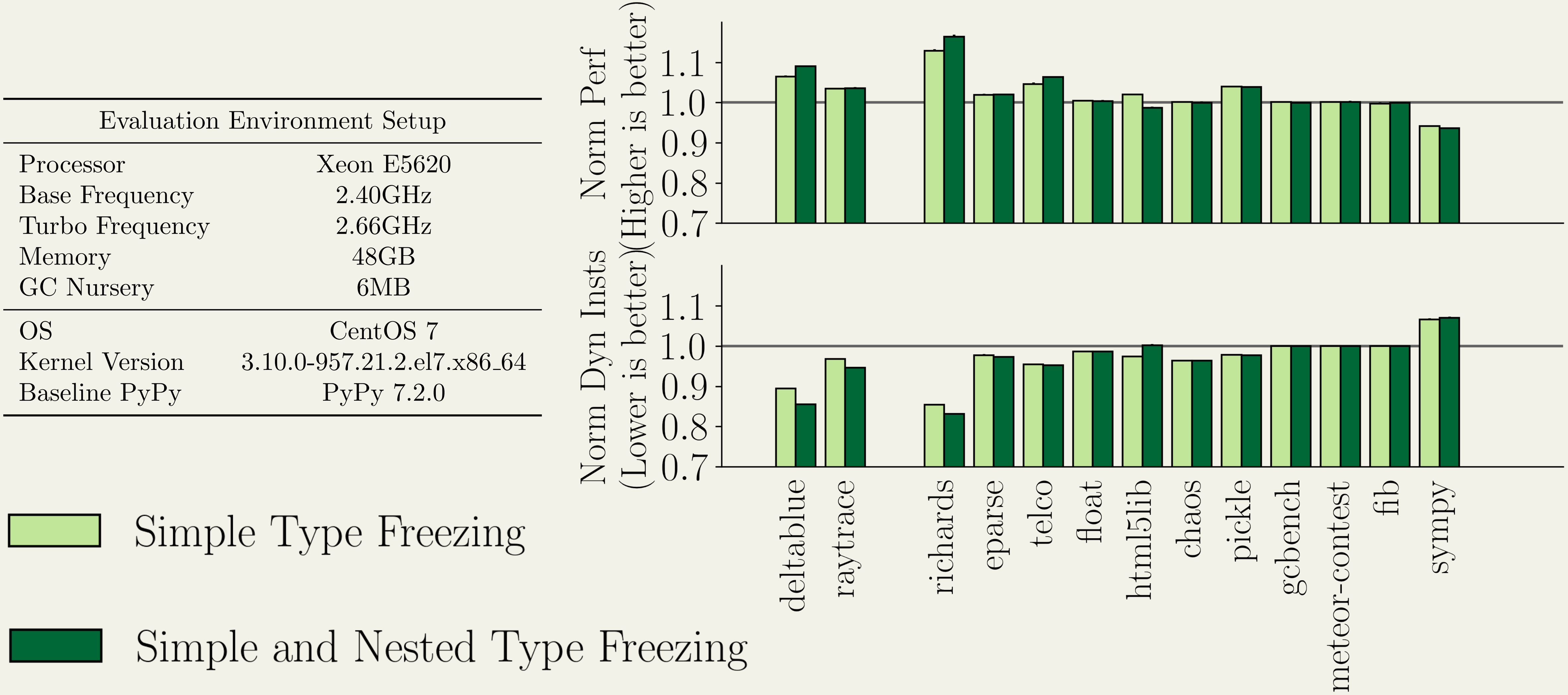
| Evaluation Environment Setup | |
|---|---|
| Processor | Xeon E5620 |
| Base Frequency | 2.40GHz |
| Turbo Frequency | 2.66GHz |
| Memory | 48GB |
| GC Nursery | 6MB |
| OS | CentOS 7 |
| Kernel Version | 3.10.0-957.21.2.el7.x86_64 |
| Baseline PyPy | PyPy 7.2.0 |

Simple Type Freezing



# PyPy Benchmark Suite

| Evaluation Environment Setup | |
|---|---|
| Processor | Xeon E5620 |
| Base Frequency | 2.40GHz |
| Turbo Frequency | 2.66GHz |
| Memory | 48GB |
| GC Nursery | 6MB |
| OS | CentOS 7 |
| Kernel Version | 3.10.0-957.21.2.el7.x86_64 |
| Baseline PyPy | PyPy 7.2.0 |



CSL

- sympy is the only case where type freezing constantly performs worse than baseline
  - We took a deep dive and found the performance cliff is related to class `Rational`
  - Create attribute type polymorphism on purpose
- Many type mutation is due to straight forward casting [1]. This is also the case in `Raytrace`
  - Classes like Point usually hold Floats, but they are initialized to Integers
  - Convert init values to floats so they don't introduce attribute type polymorphism

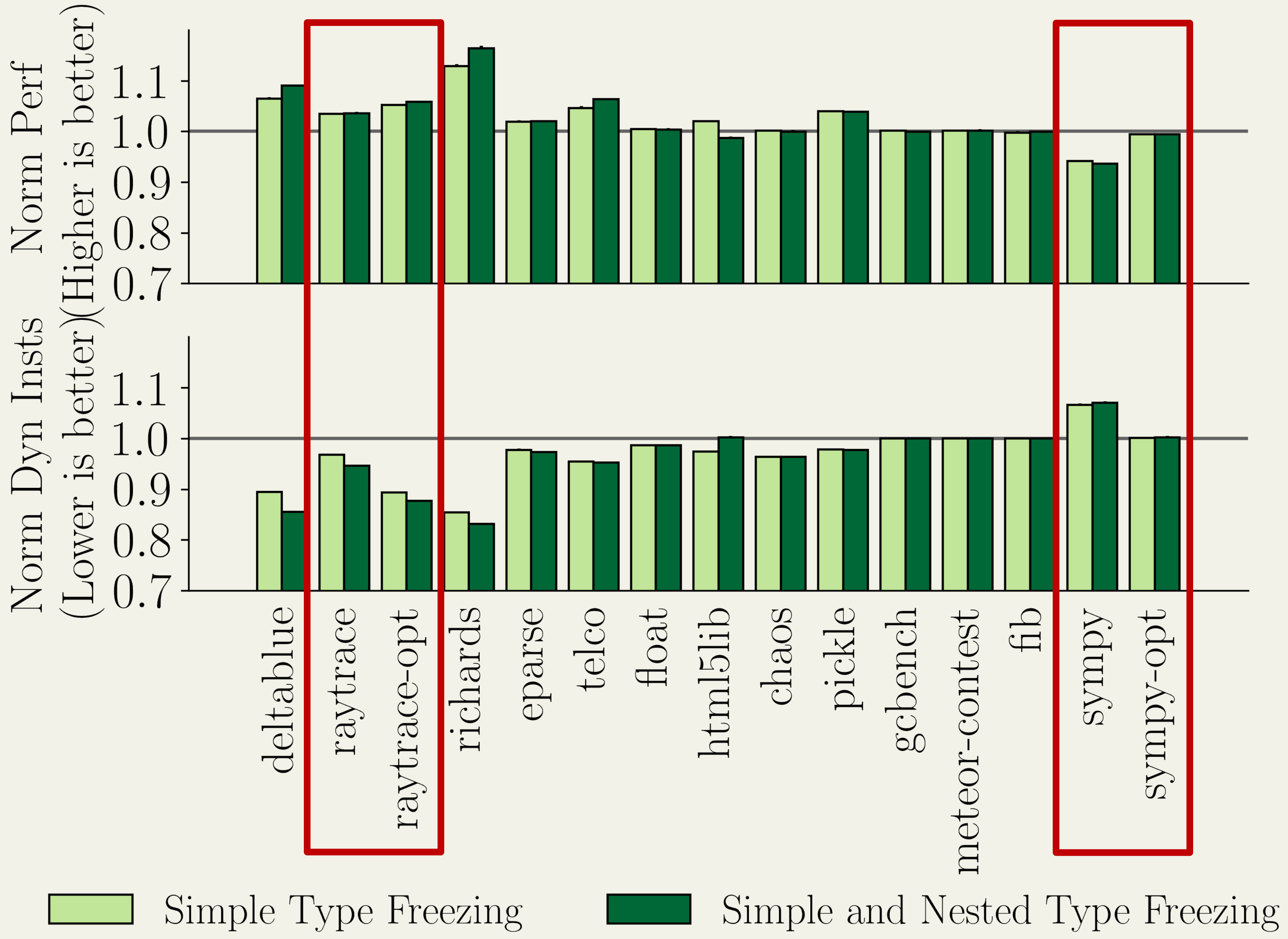
```
class Rational( Number ):
  def __new__( cls, p, q ):
    . . .
    obj.p = p
    obj.q = q
    . . .
dummy = Rational(None, None)
```

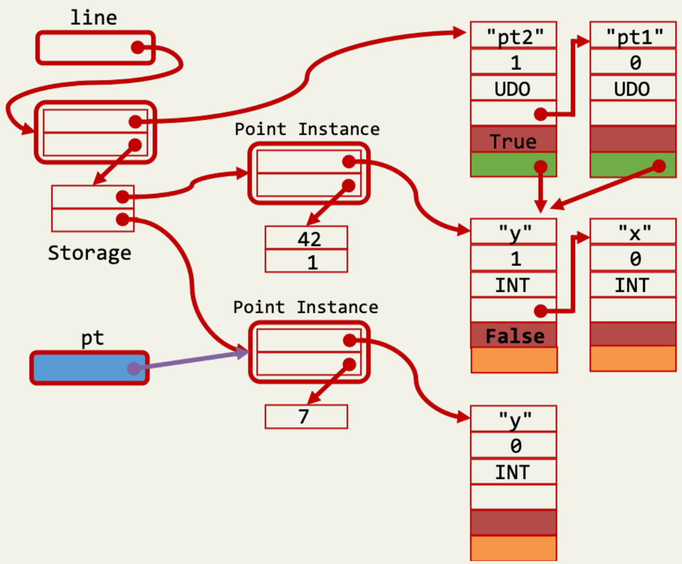
Small Int → Long Int

```
class Point( object ):
  def __init__( cls, x, y, z ):
    self.x = x
    self.y = y
    self.z = z
    . . .
 s.addLight( Point(30,30,10) )
 s.addLight( Point(30.0, 30.0, 10.0) )
```

[1] Alex Holkner and James Harland. 2009. Evaluating The Dynamic Behaviour of Python Applications. *Australasian Conference on Computer Science* (Jan 2009)

# Make Apps Type Freezing Friendly






- Attribute type monomorphism exists in real world applications
- We propose simple and nested type freezing to exploit attribute type monomorphism
- As a pure software technique, type freezing achieves most of the performance benefit of a prior SW/HW co-design work[1]

[1] Dot et al, Removing Checks in Dynamically Typed Languages Through Efficient Profiling. *Int'l Conf. on Code Generation and Optimization* (CGO) (Feb 2017)