Synthesizing Replacement Classes

Malavika Samak, Deokhwan Kim and Martin Rinard

CSAIL, MIT

Library Usage in Applications

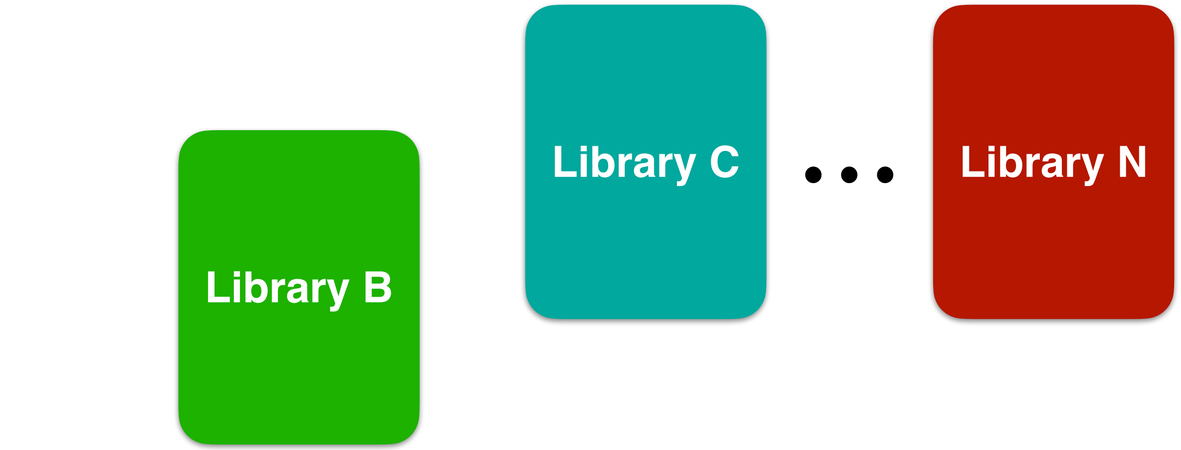
Developer

Library Usage in Applications

Application

Developer

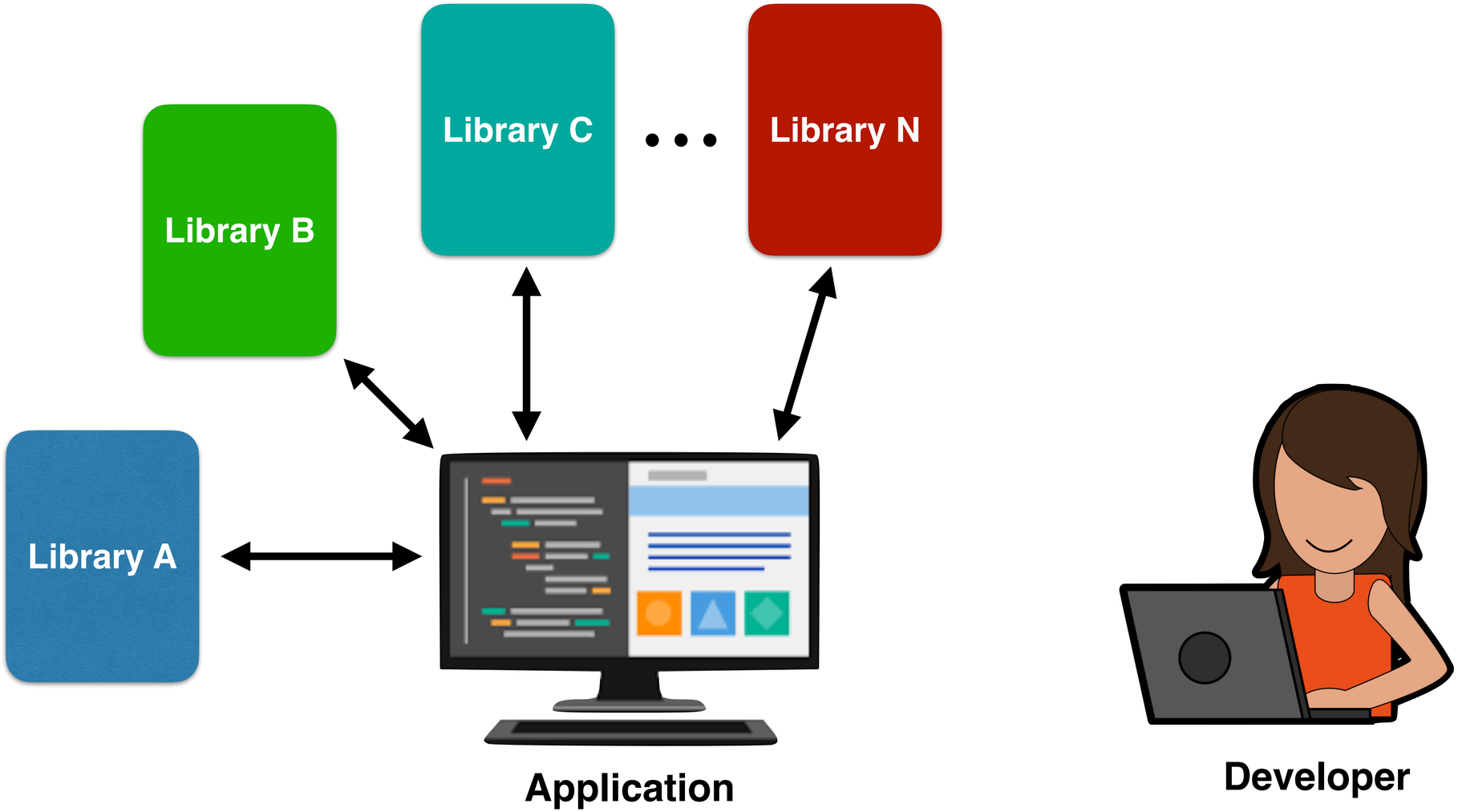
Library Usage in Applications



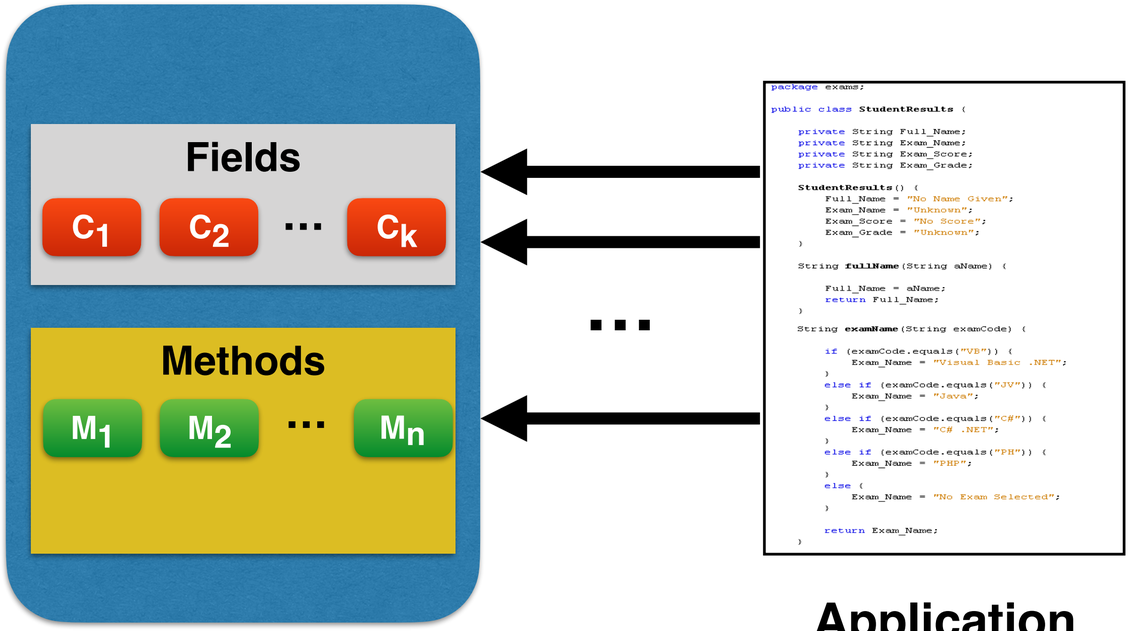
Application

Developer

Library Usage in Applications



Class Updates for Applications



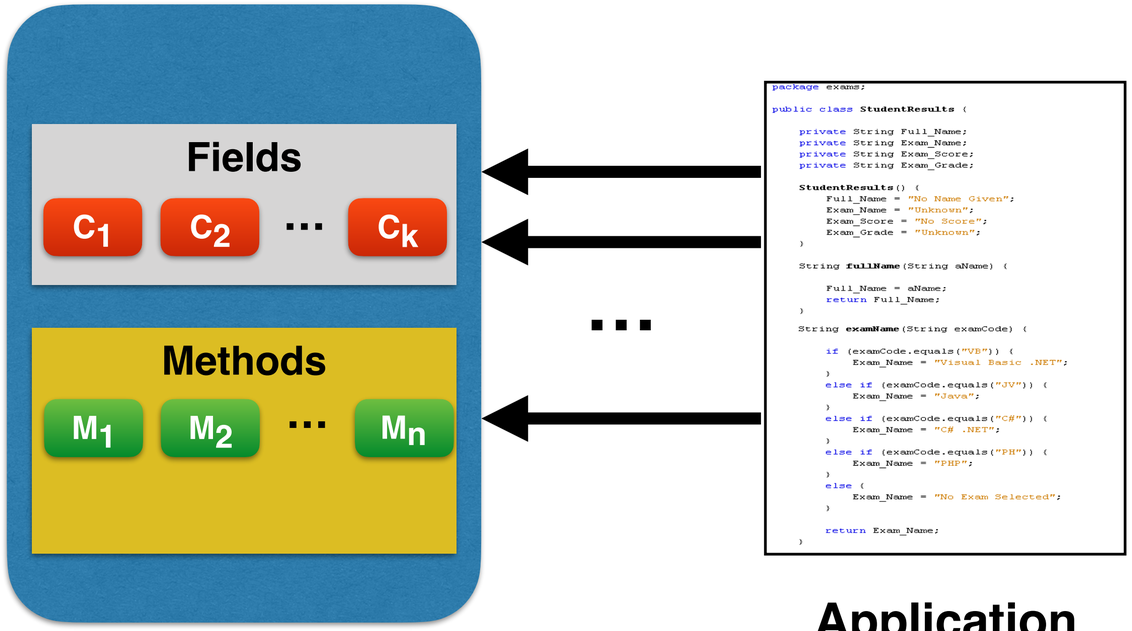
Class O

```
package exams;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fullName(String aName) {

        Full_Name = aName;
        return Full_Name;
    }

    String examName(String examCode) {

        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

Application

Developer

Class Updates for Applications

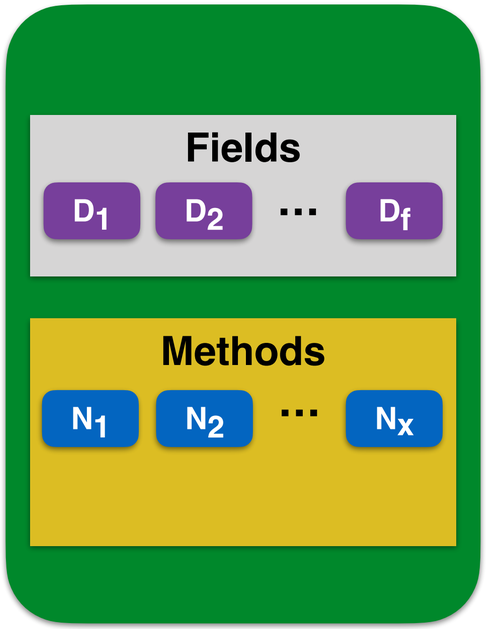


Class O

```
package exams;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fullName(String aName) {

        Full_Name = aName;
        return Full_Name;
    }

    String examName(String examCode) {

        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

...

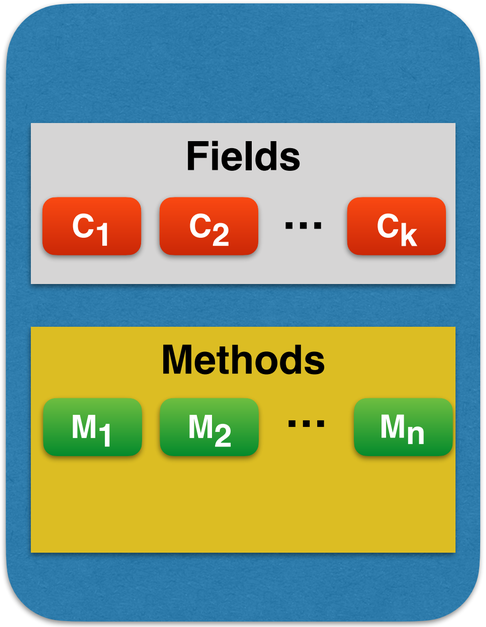
Application



Class R

Developer

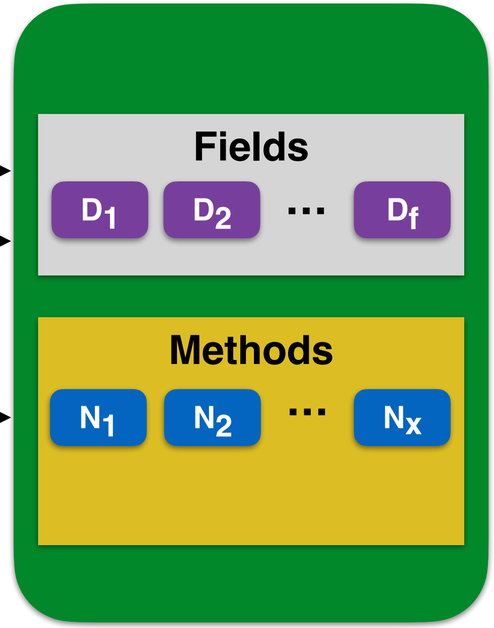
Class Updates for Applications



Class O

```
package exams;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fullName(String aName) {

        Full_Name = aName;
        return Full_Name;
    }

    String examName(String examCode) {

        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

Application

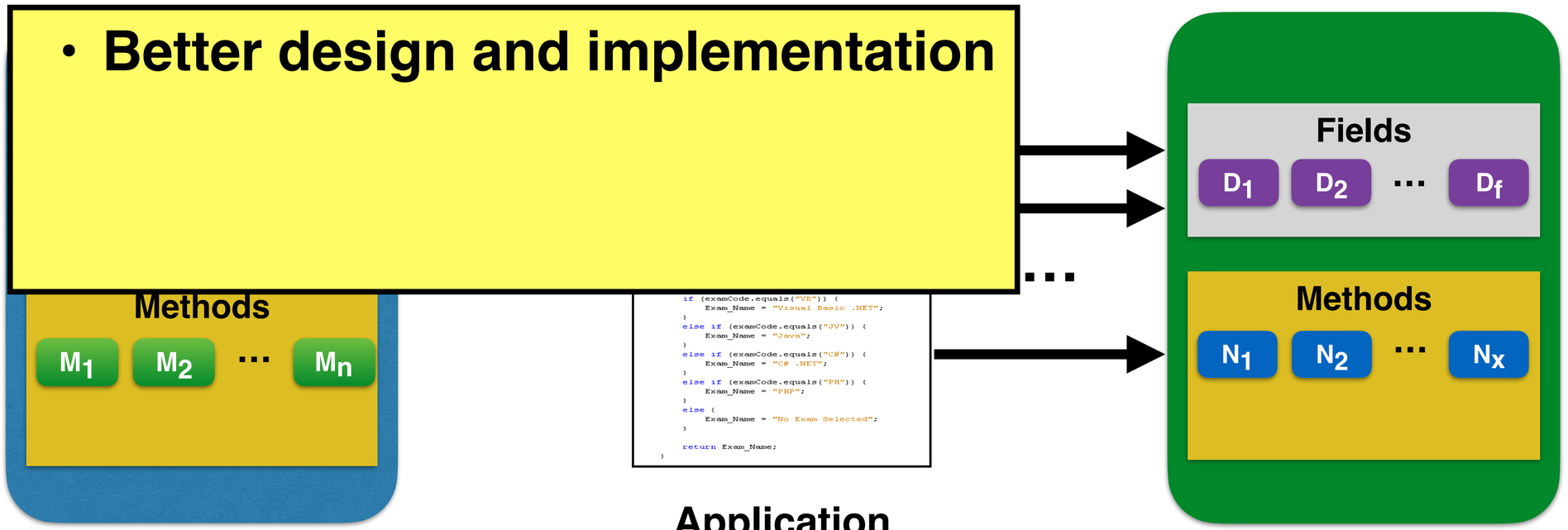


Class R

Developer

Class Updates for Applications

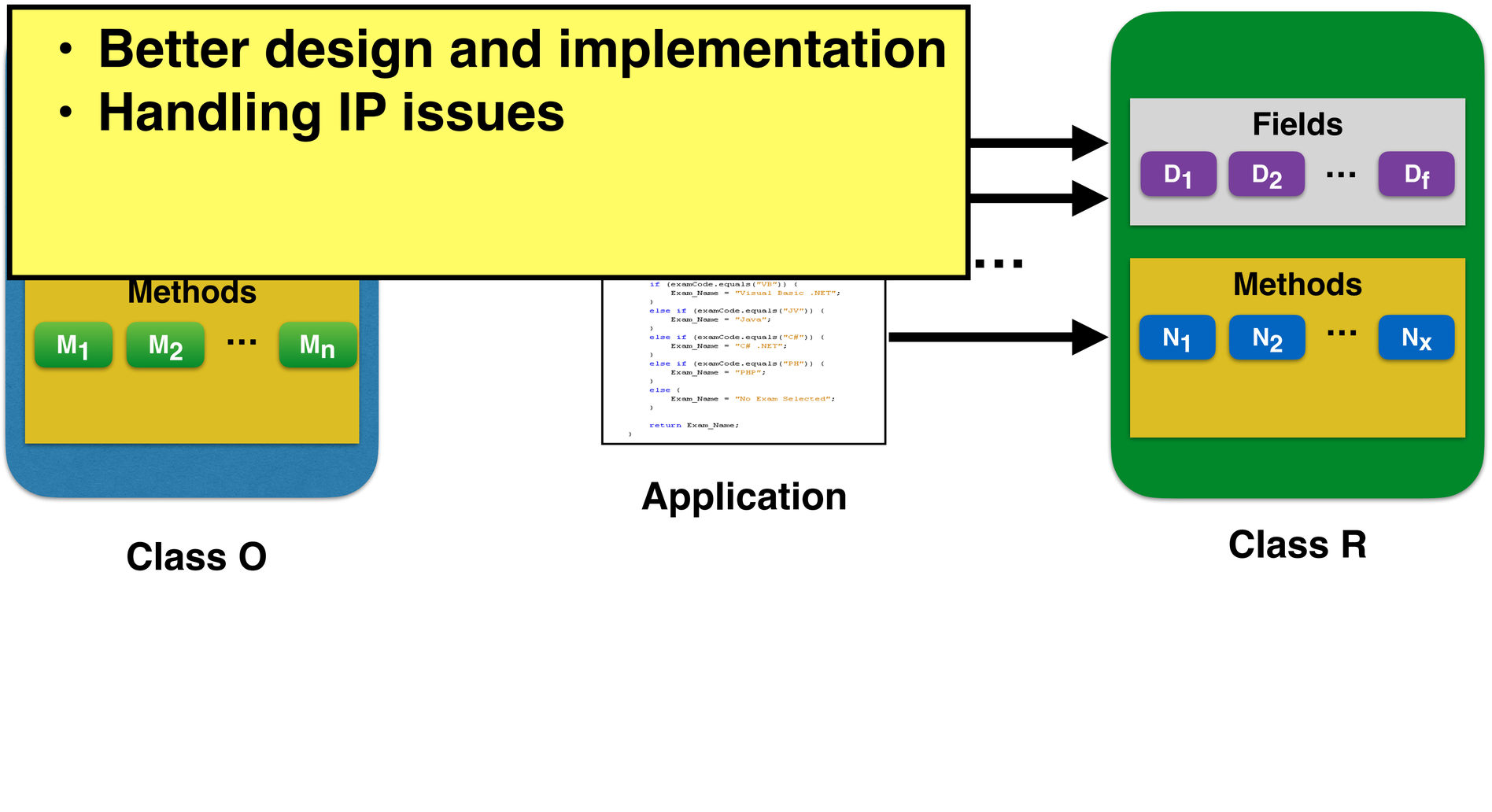
- **Better design and implementation**



Developer

Class Updates for Applications

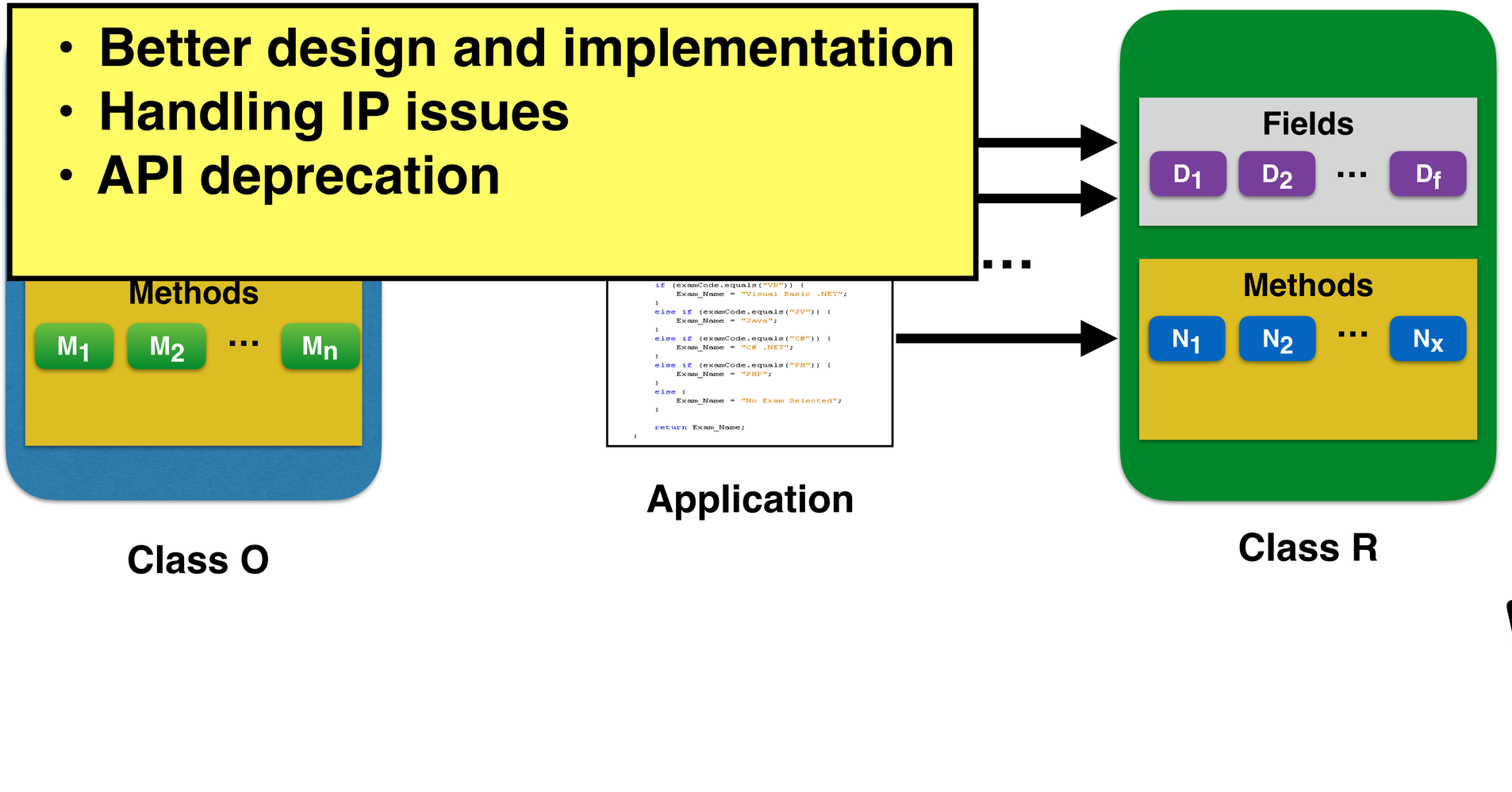
- **Better design and implementation**
- **Handling IP issues**



Developer

Class Updates for Applications

- **Better design and implementation**
- **Handling IP issues**
- **API deprecation**



Developer

Class Updates for Applications

- **Better design and implementation**
- **Handling IP issues**
- **API deprecation**

…

Methods

M_1 M_2 … M_n

Class O

```
if (examCode.equals("VB")) {
    Exam_Name = "Visual Basic .NET";
}
else if (examCode.equals("JV")) {
    Exam_Name = "Java";
}
else if (examCode.equals("C#")) {
    Exam_Name = "C# .NET";
}
else if (examCode.equals("PH")) {
    Exam_Name = "PHP";
}
else {
    Exam_Name = "No Exam Selected";
}

return Exam_Name;
}
```

Application

…

Fields

D_1 D_2 … D_f

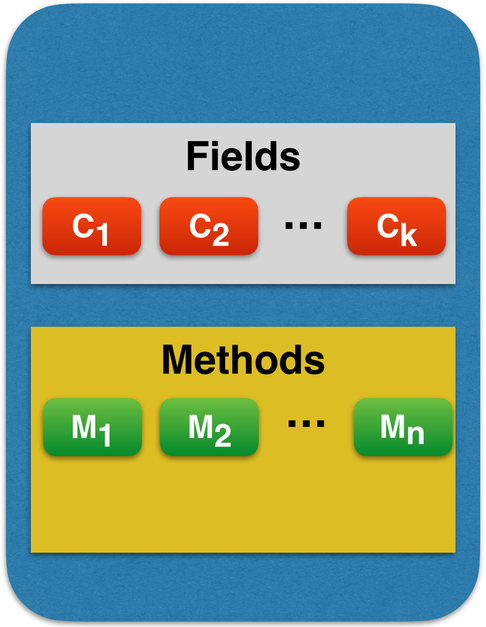
Methods

N_1 N_2 … N_x

Class R

Developer

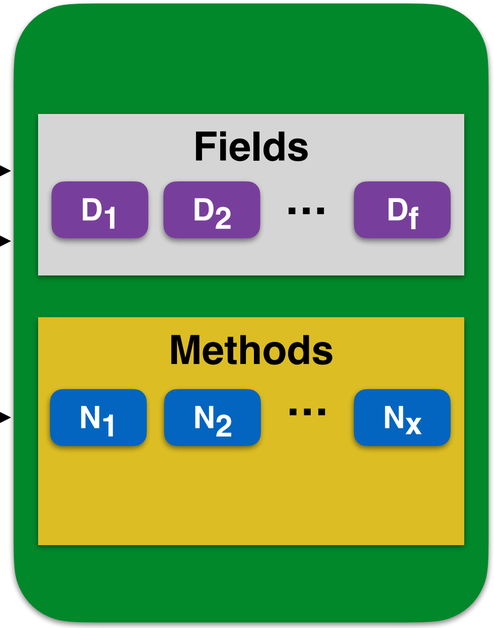
Class Updates for Applications



Class O

```
package exams;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fullName(String aName) {

        Full_Name = aName;
        return Full_Name;
    }

    String examName(String examCode) {

        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

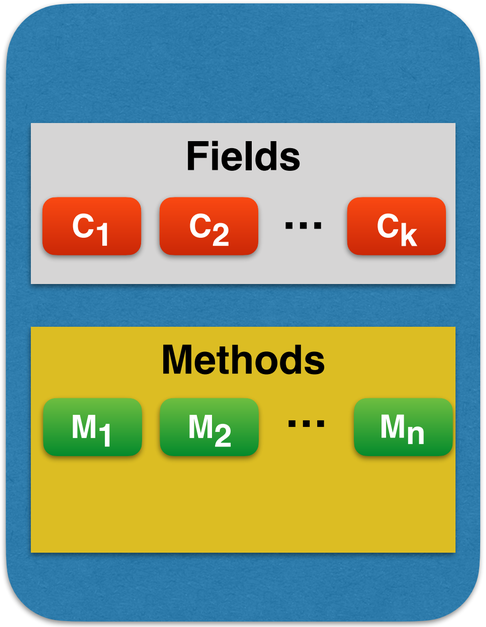
Application



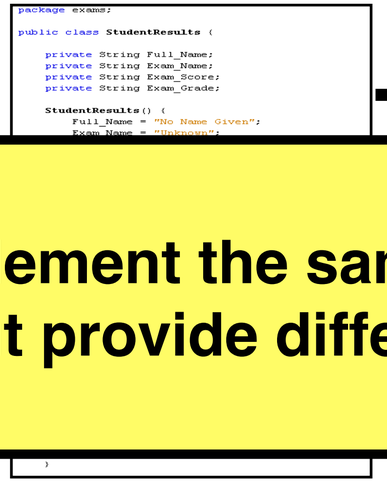
Class R

Developer

Class Updates for Applications

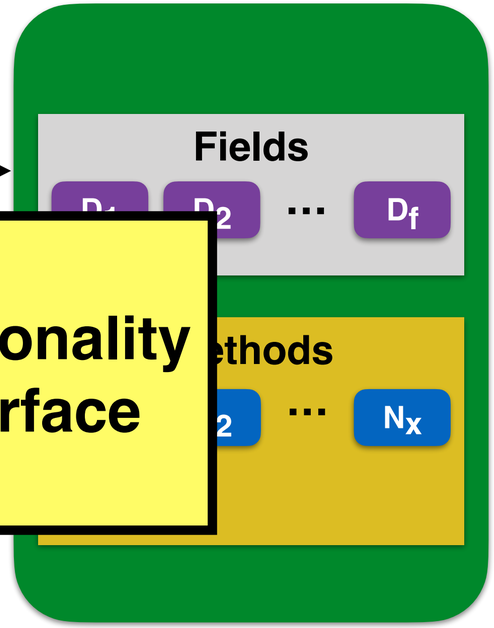


Class O



Application

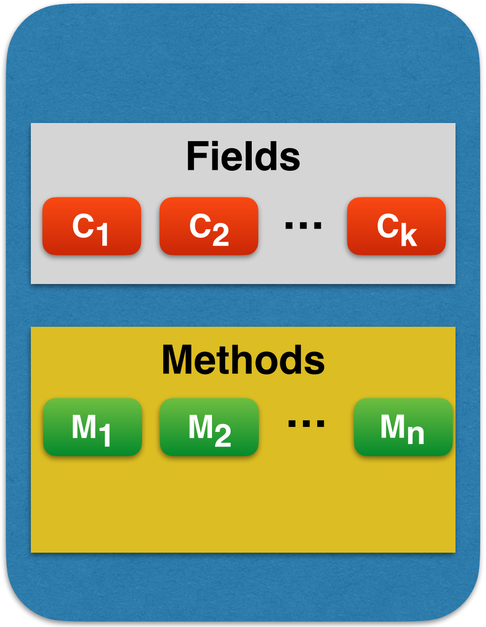
Implement the same functionality but provide different interface



Class R

Developer

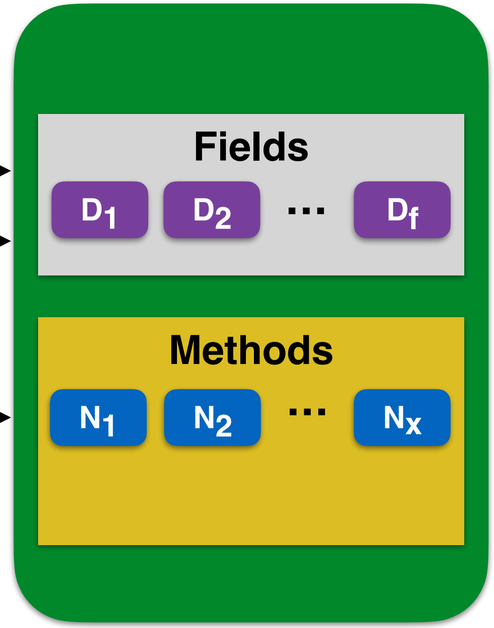
Refactoring challenges



Class O

```
package exams;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fullName(String aName) {

        Full_Name = aName;
        return Full_Name;
    }

    String examName(String examCode) {

        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

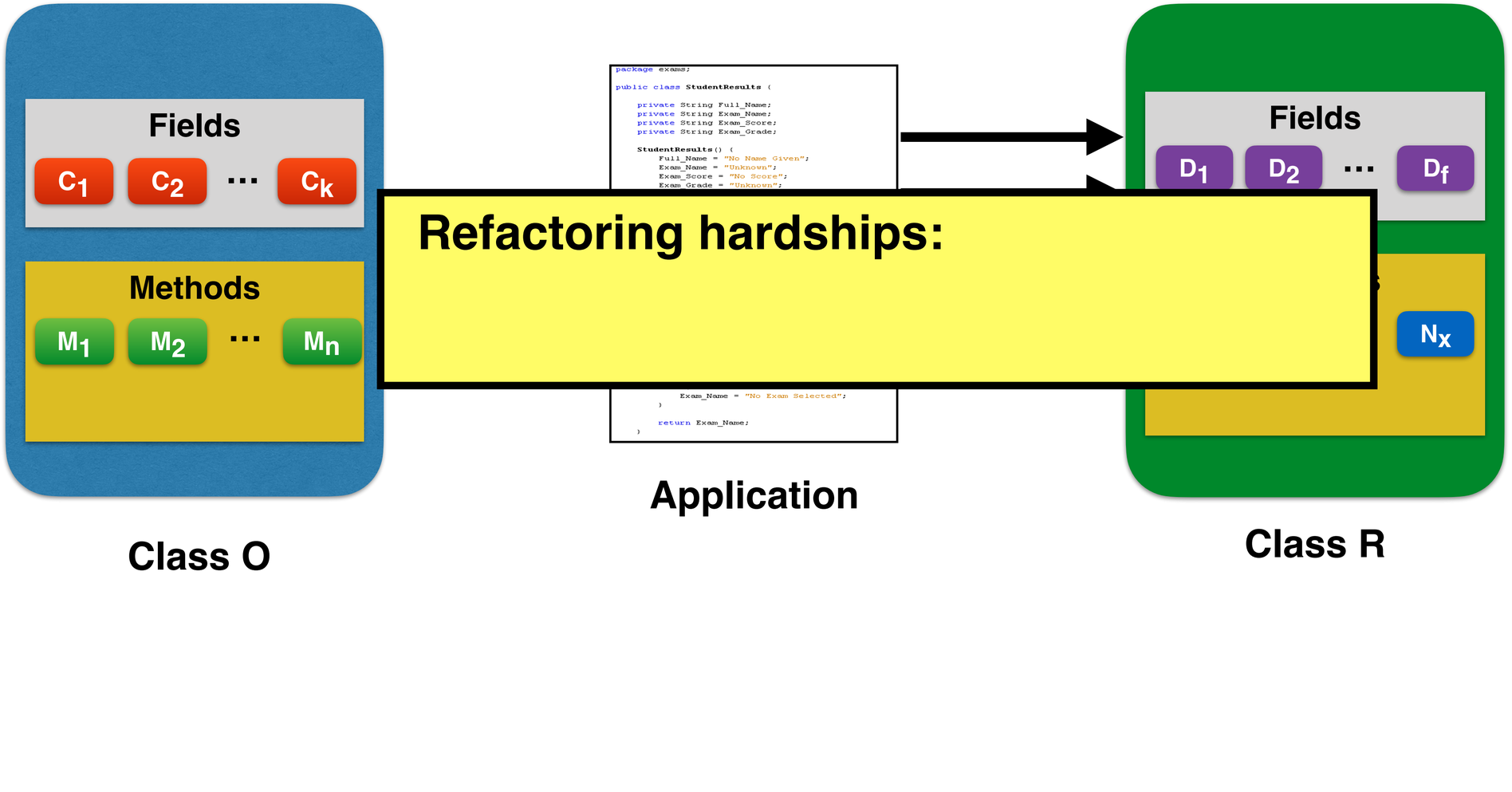
Application



Class R

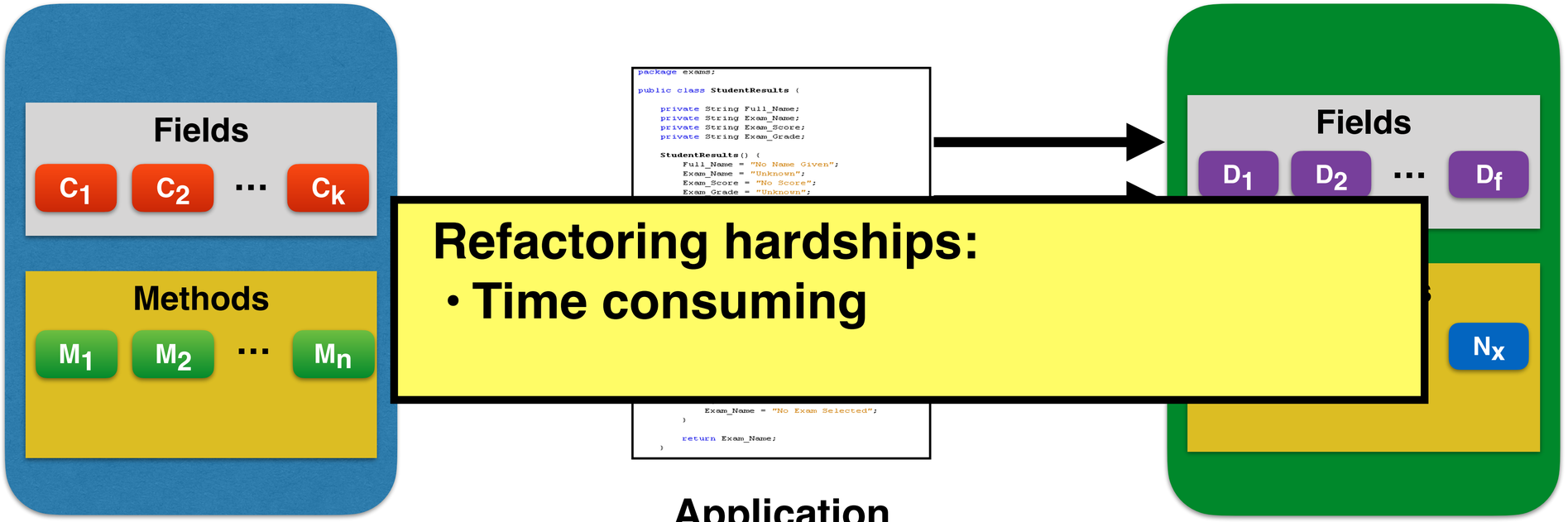
Developer

Refactoring challenges



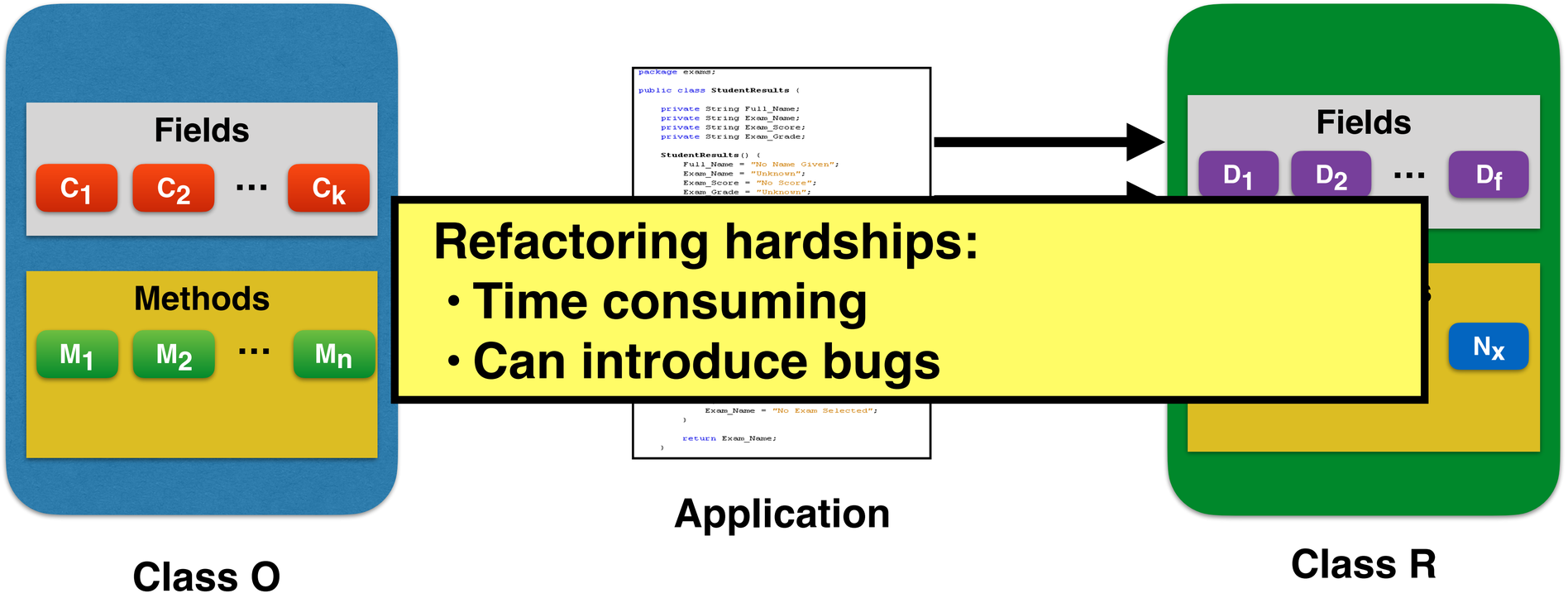
Developer

Refactoring challenges



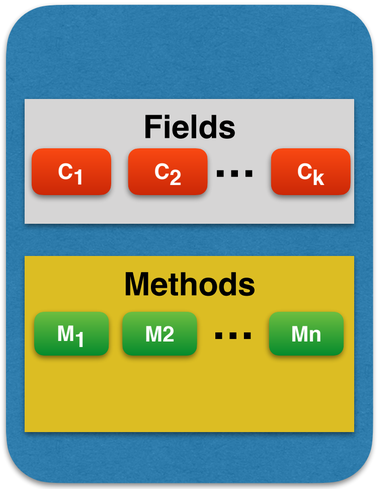
Developer

Refactoring challenges

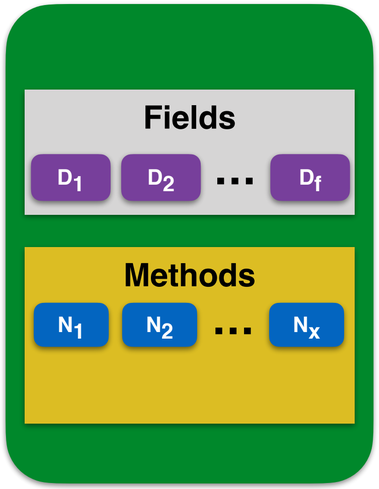


Developer

MASK

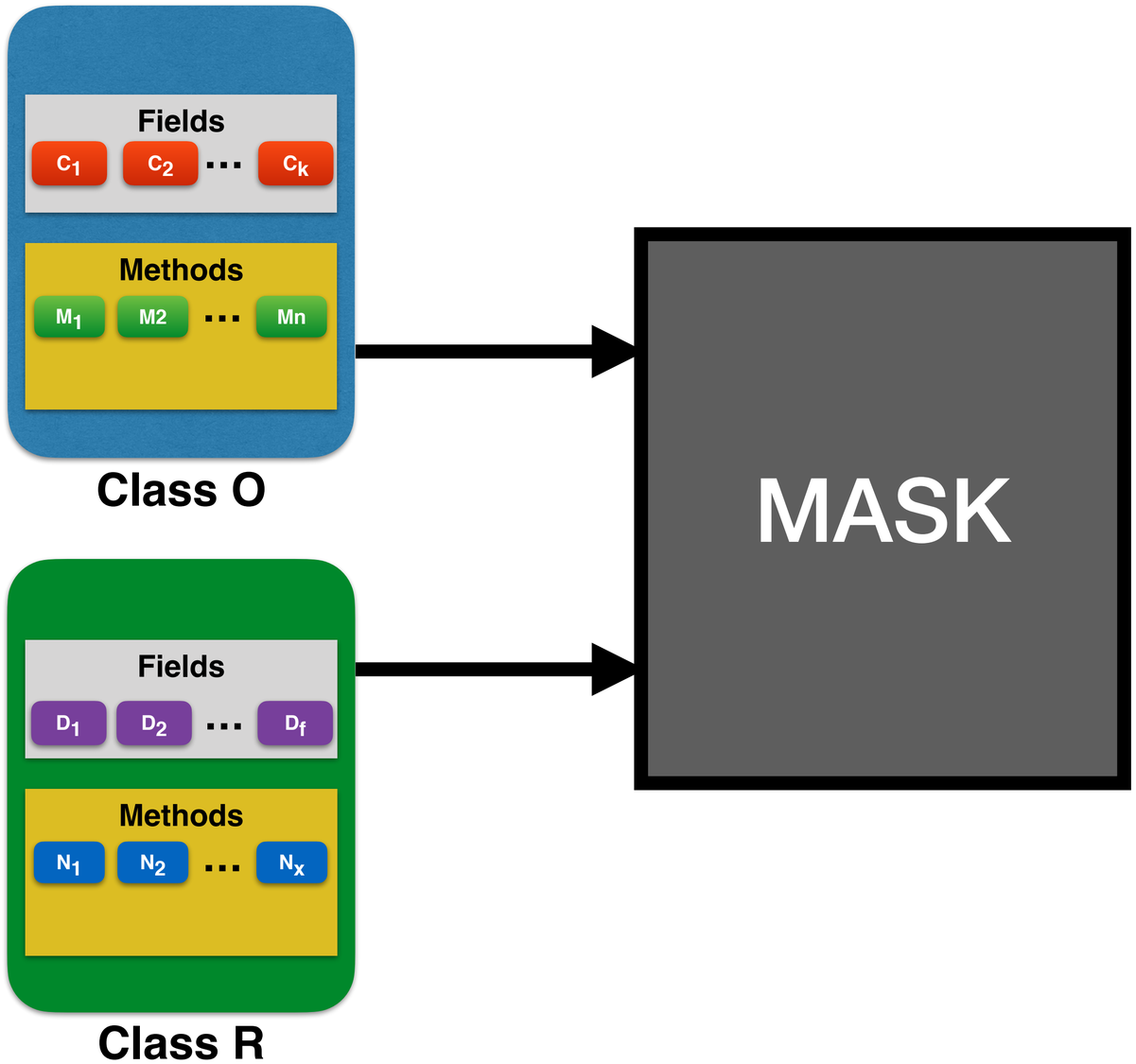


Class O

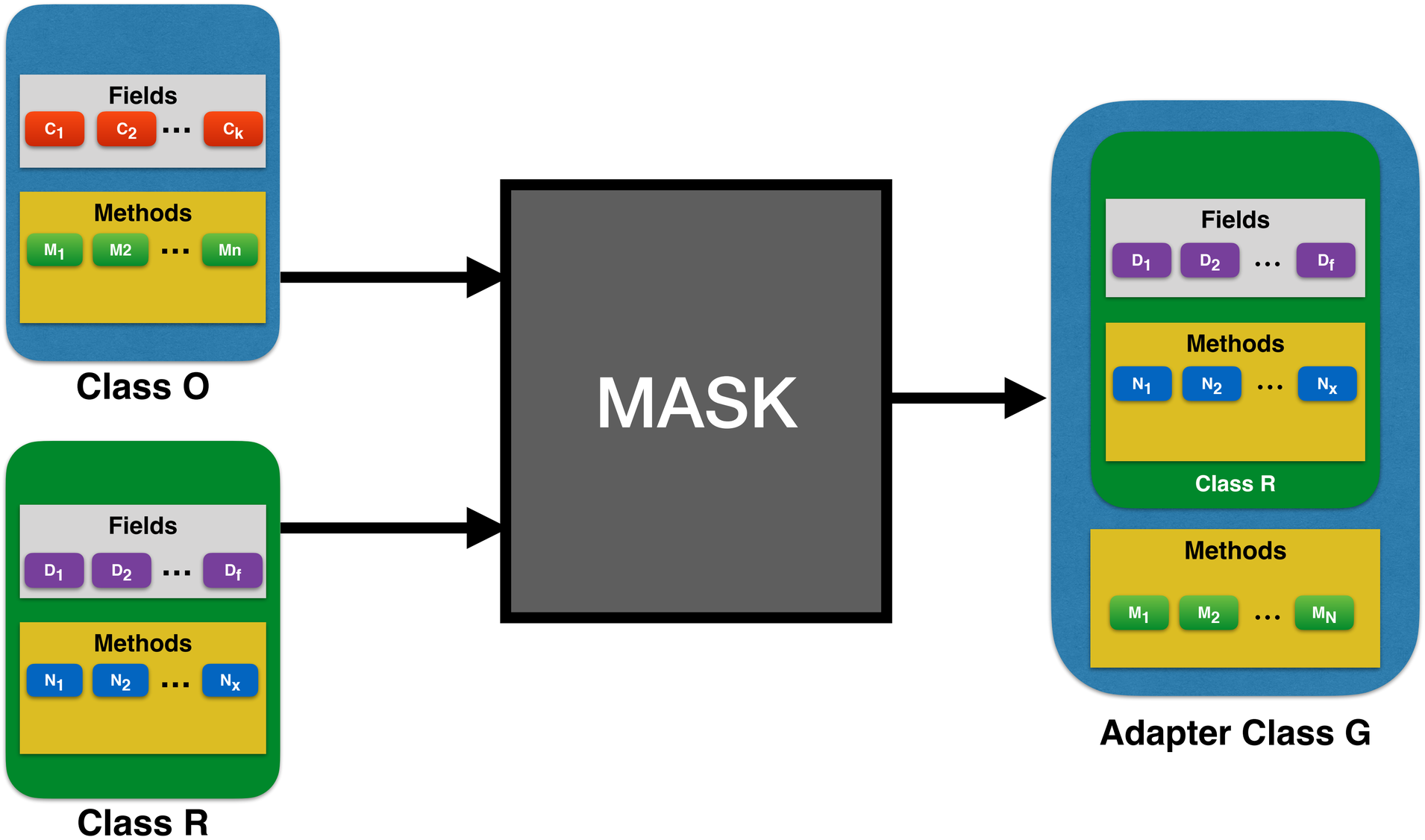


Class R

MASK

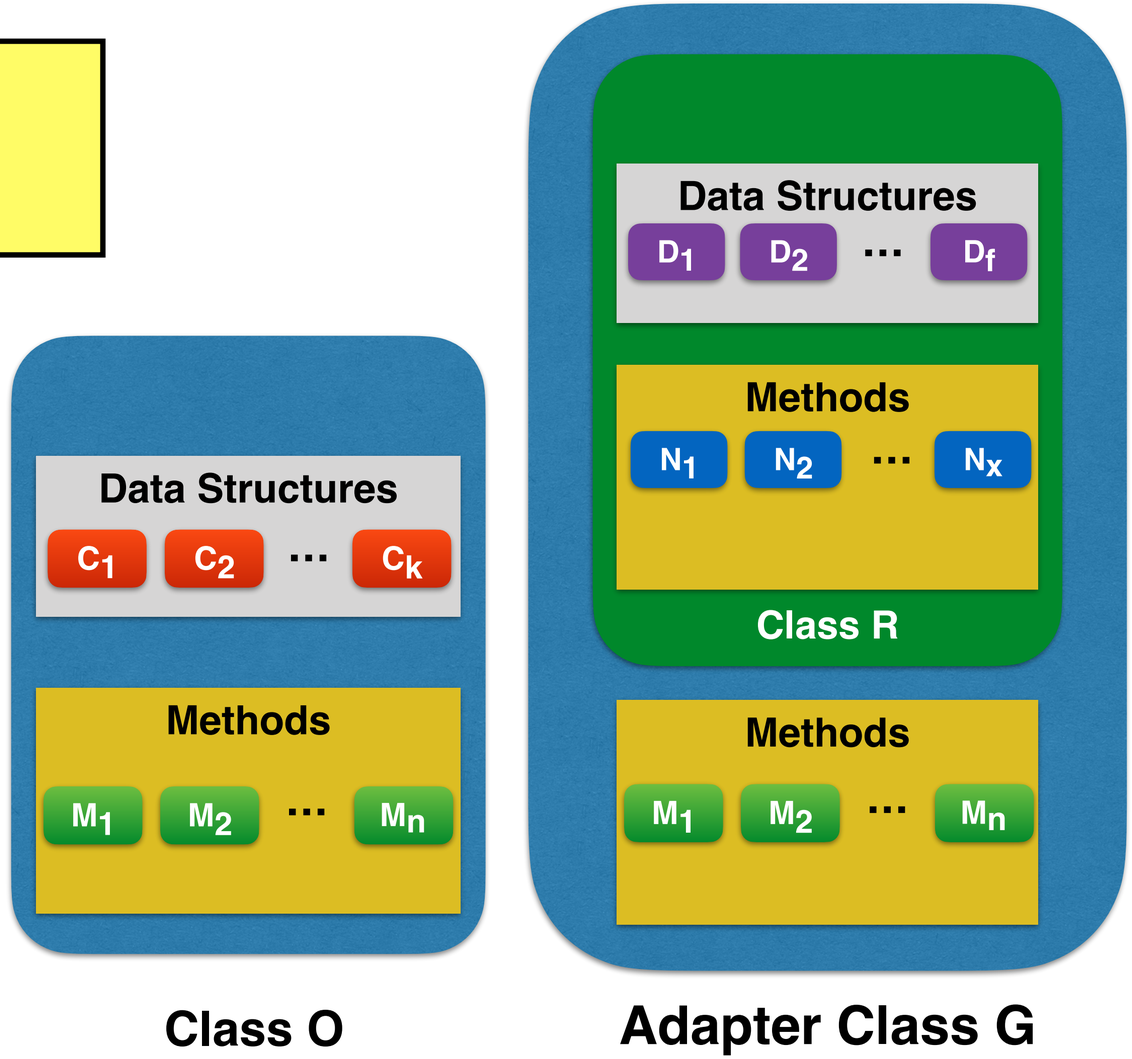

Fields
C1
C2
...
Ck
Methods
M1
M2
...
Mn
Class O
Fields
D1
D2
...
Df
Methods
N1
N2
...
Nx
Class R
MASK

MASK


Fields
C1
C2
Ck
Methods
M1
M2
Mn
Class O
Fields
D1
D2
Df
Methods
N1
N2
Nx
Class R
MASK
Fields
D1
D2
Df
Methods
N1
N2
Nx
Class R
Methods
M1
M2
MN
Adapter Class G

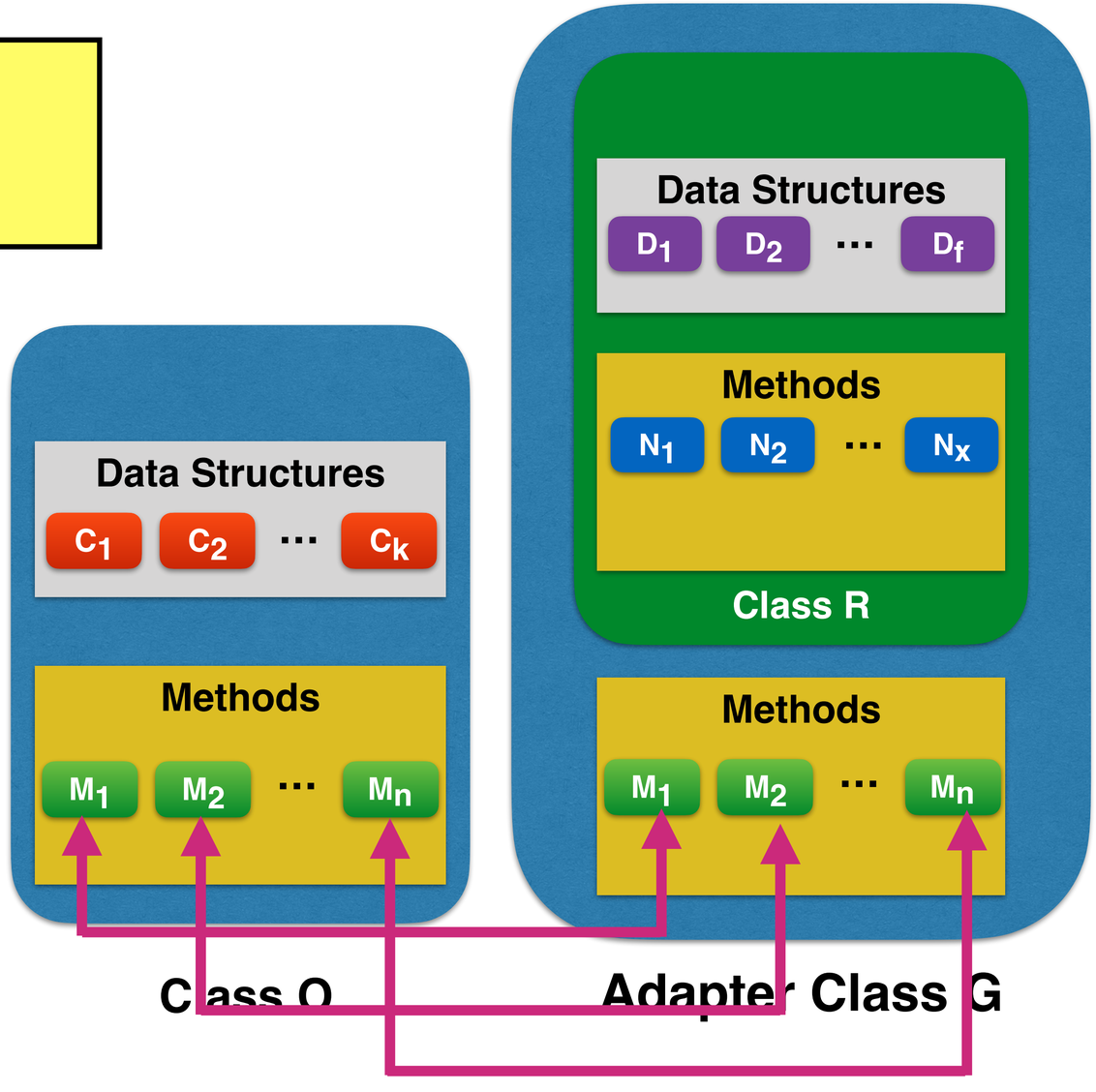
Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts



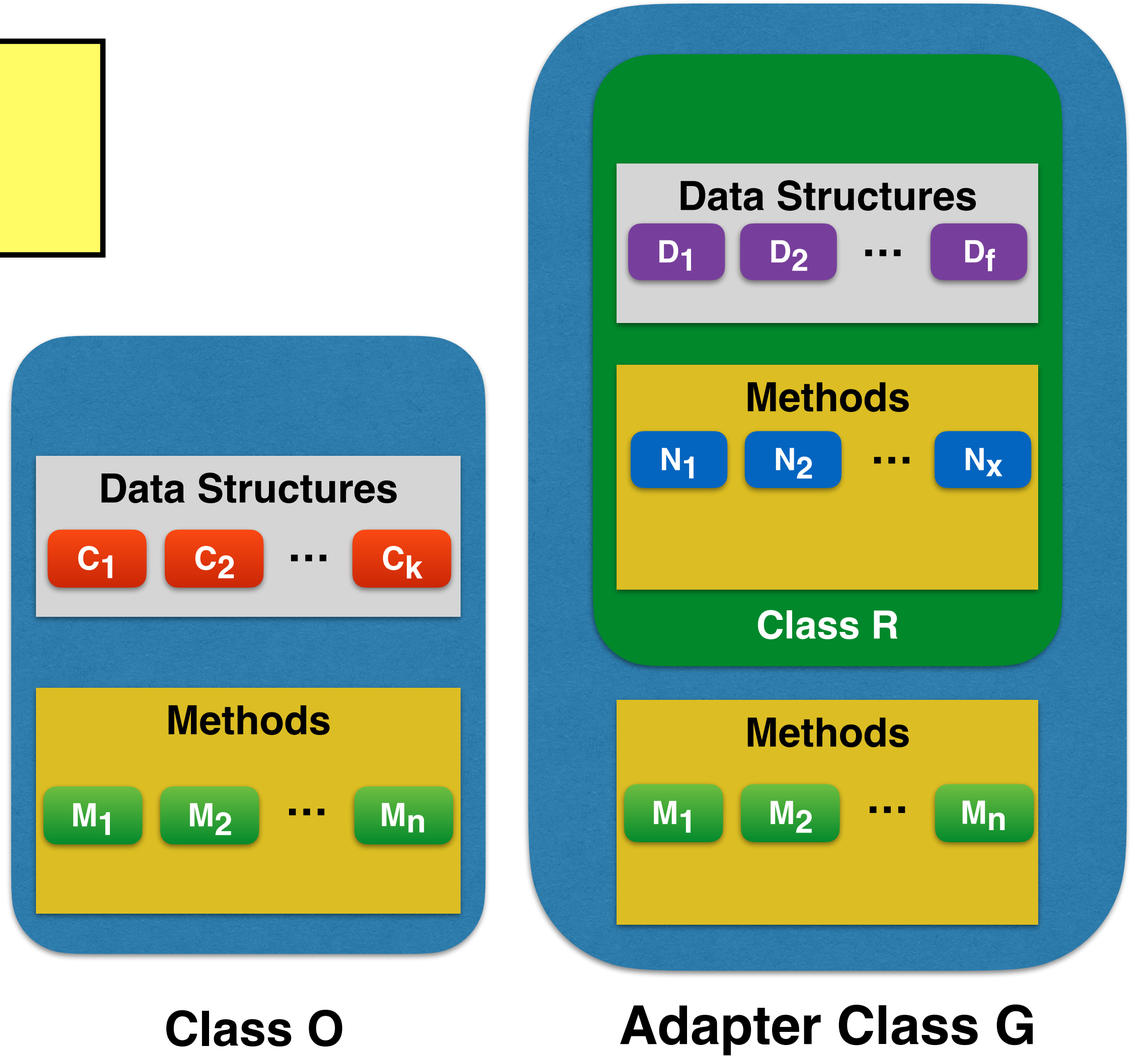
Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts



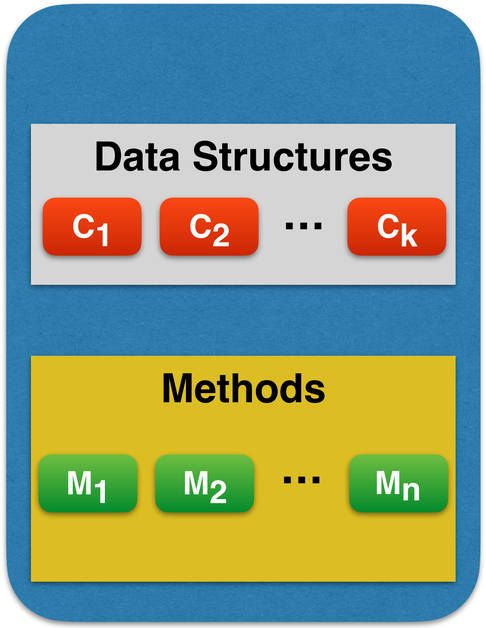
Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts

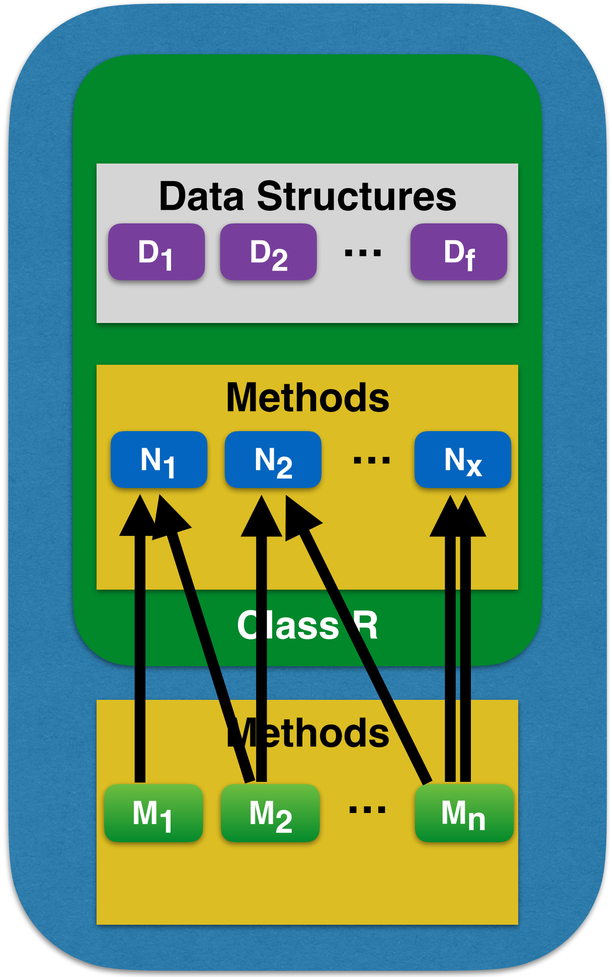


Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts



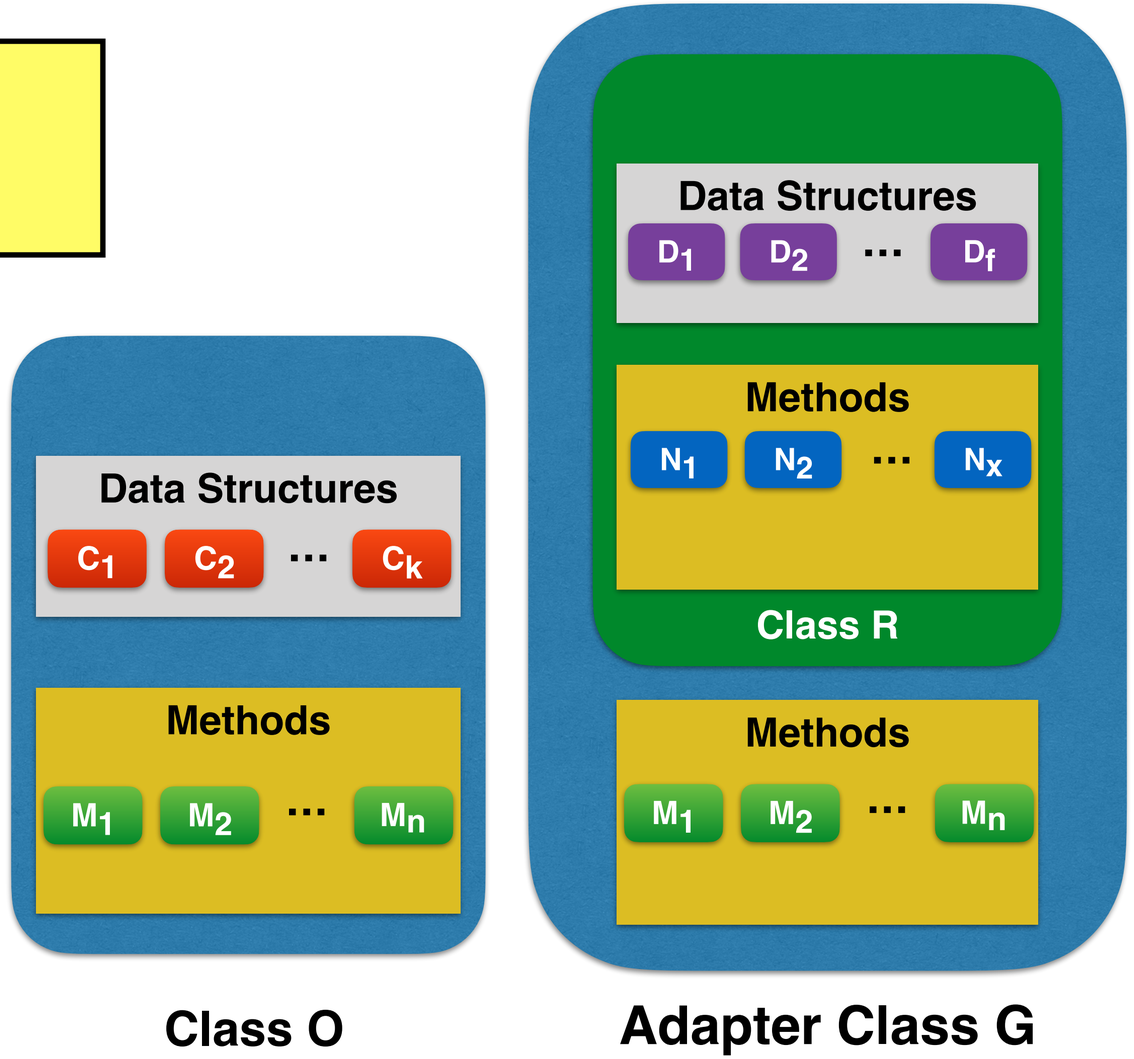
Class O



Adapter Class G

Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts

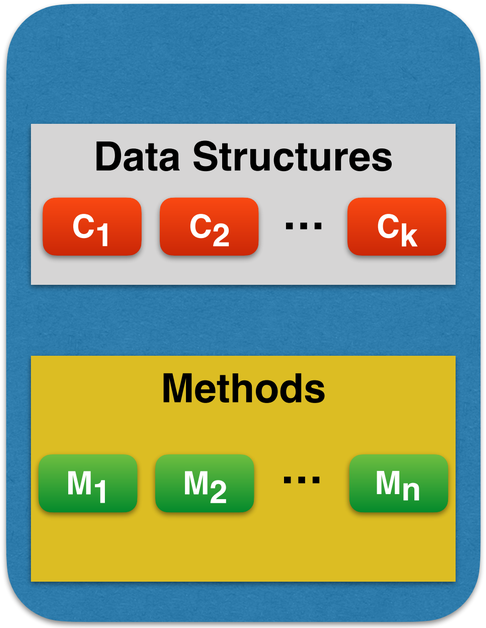


Adapter Class Requirements

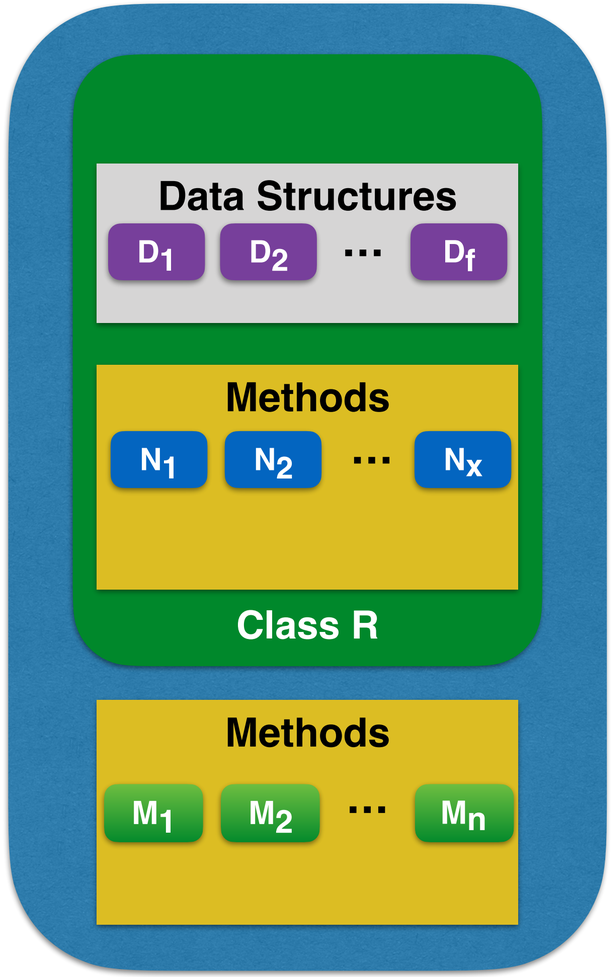
1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts

$r_1 = a.M_i(a_1 \ldots a_n)$

$r_1 = a.M_i(a_1 \ldots a_n)$



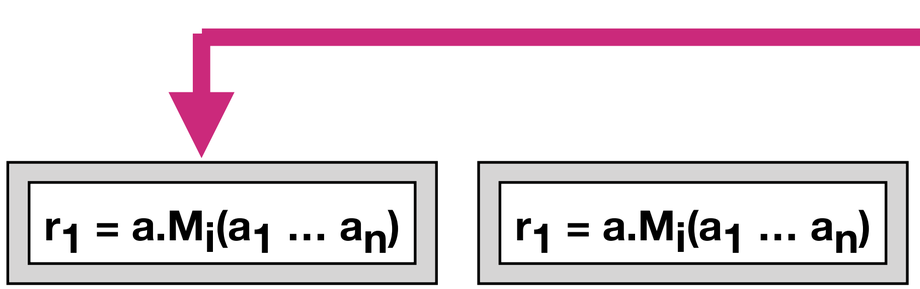
Class O

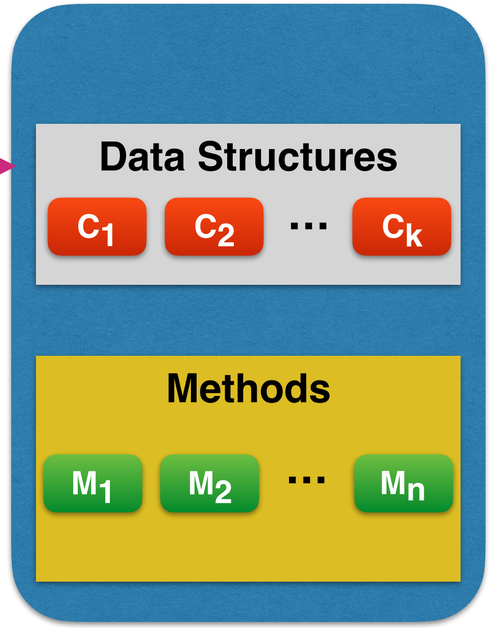


Adapter Class G

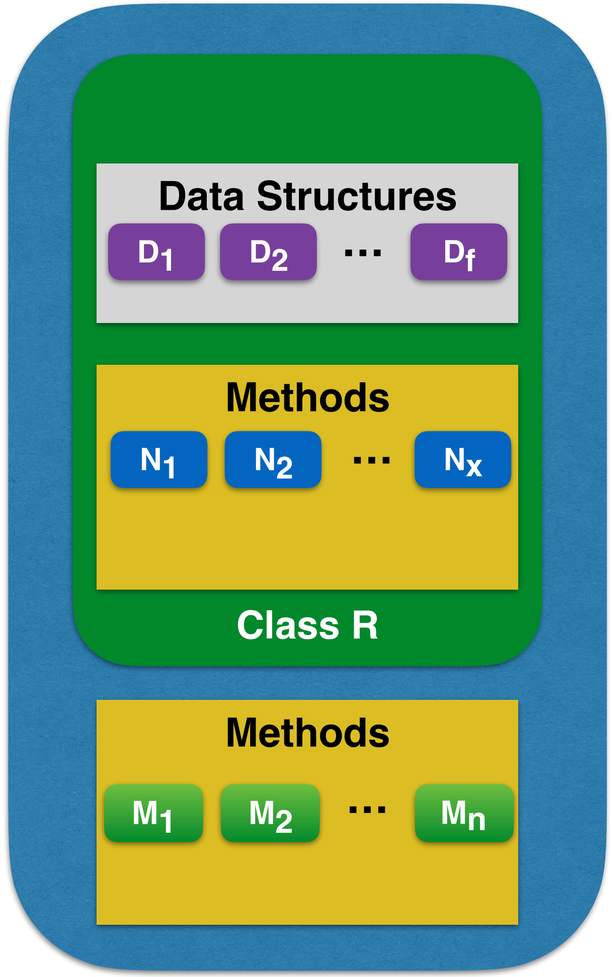
Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts





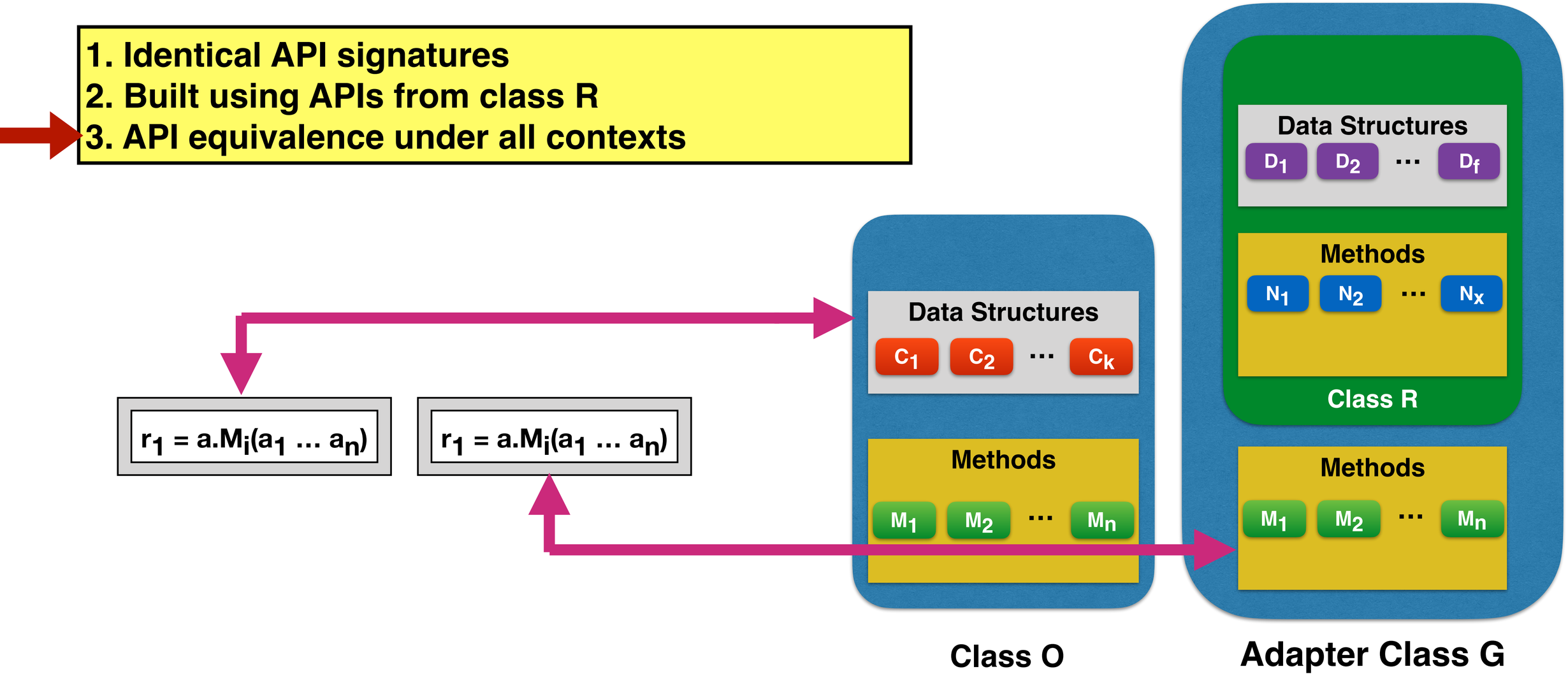
Class O



Adapter Class G

Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts

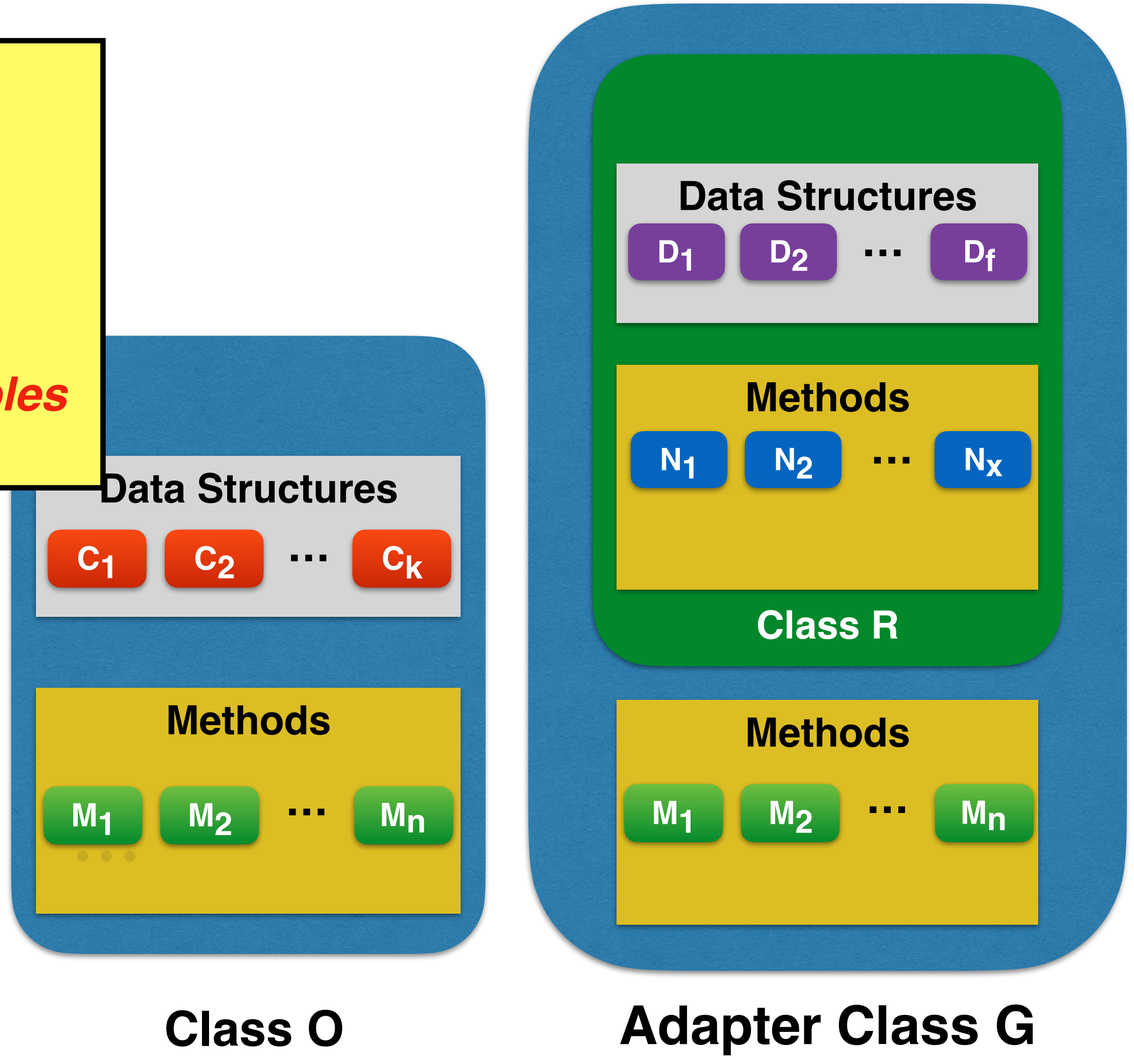


Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts
   a) Return *equivalent value* for equivalent history
   b) Drives *all* objects to *equivalent states*
   c) Creates an equivalent *aliasing* between *all variables* and their *dereferences*

$r_1 = a.M_i(a_1 \ldots a_n)$

$r_1 = a.M_i(a_1 \ldots a_n)$



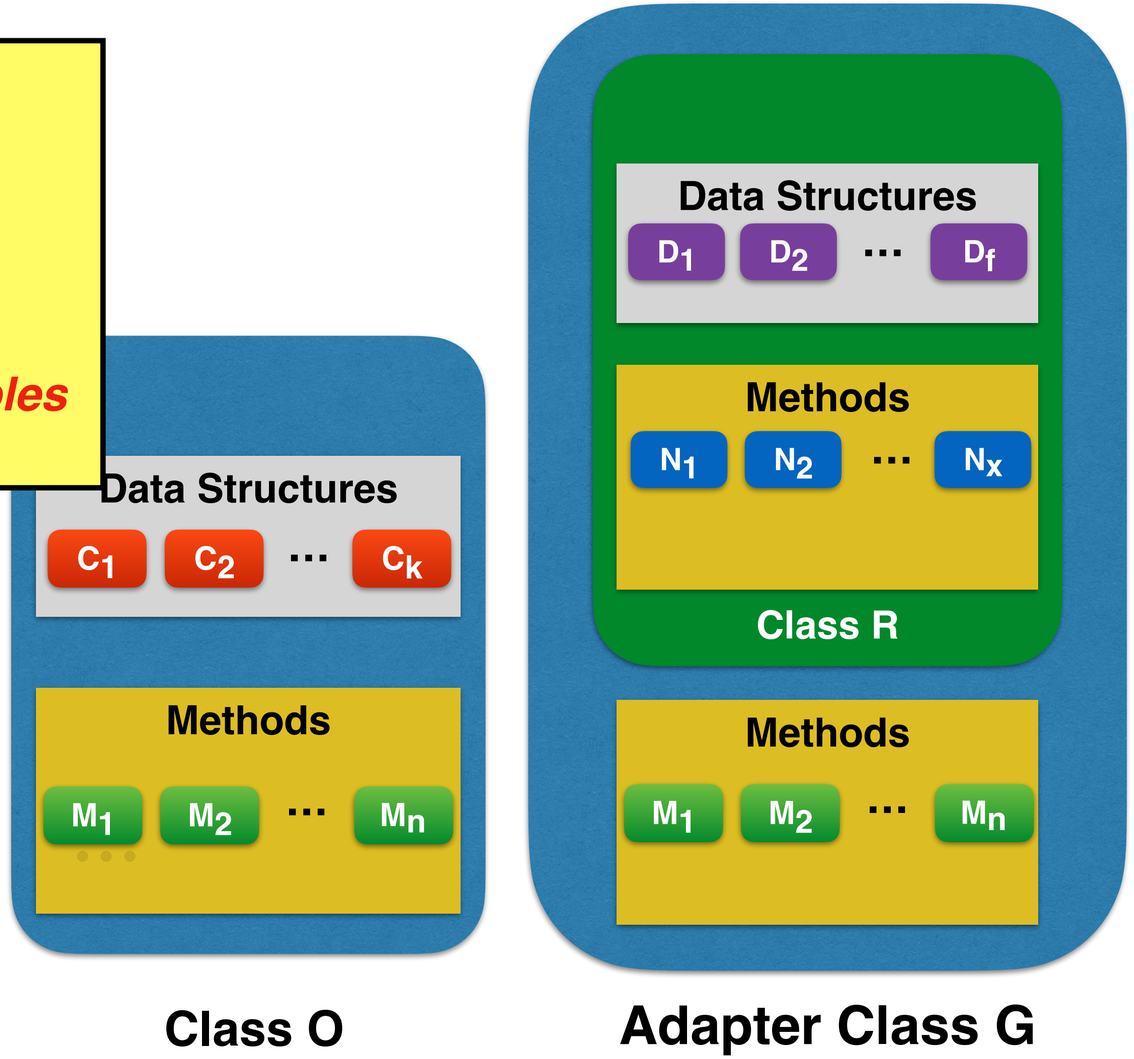
Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts
   a) Return *equivalent value* for equivalent history
   b) Drives *all* objects to *equivalent states*
   c) Creates an equivalent *aliasing* between *all variables* and their *dereferences*

$r_1 = a.M_i(a_1 \ldots a_n)$

$r_1 = a.M_i(a_1 \ldots a_n)$

r1 ≡ r1

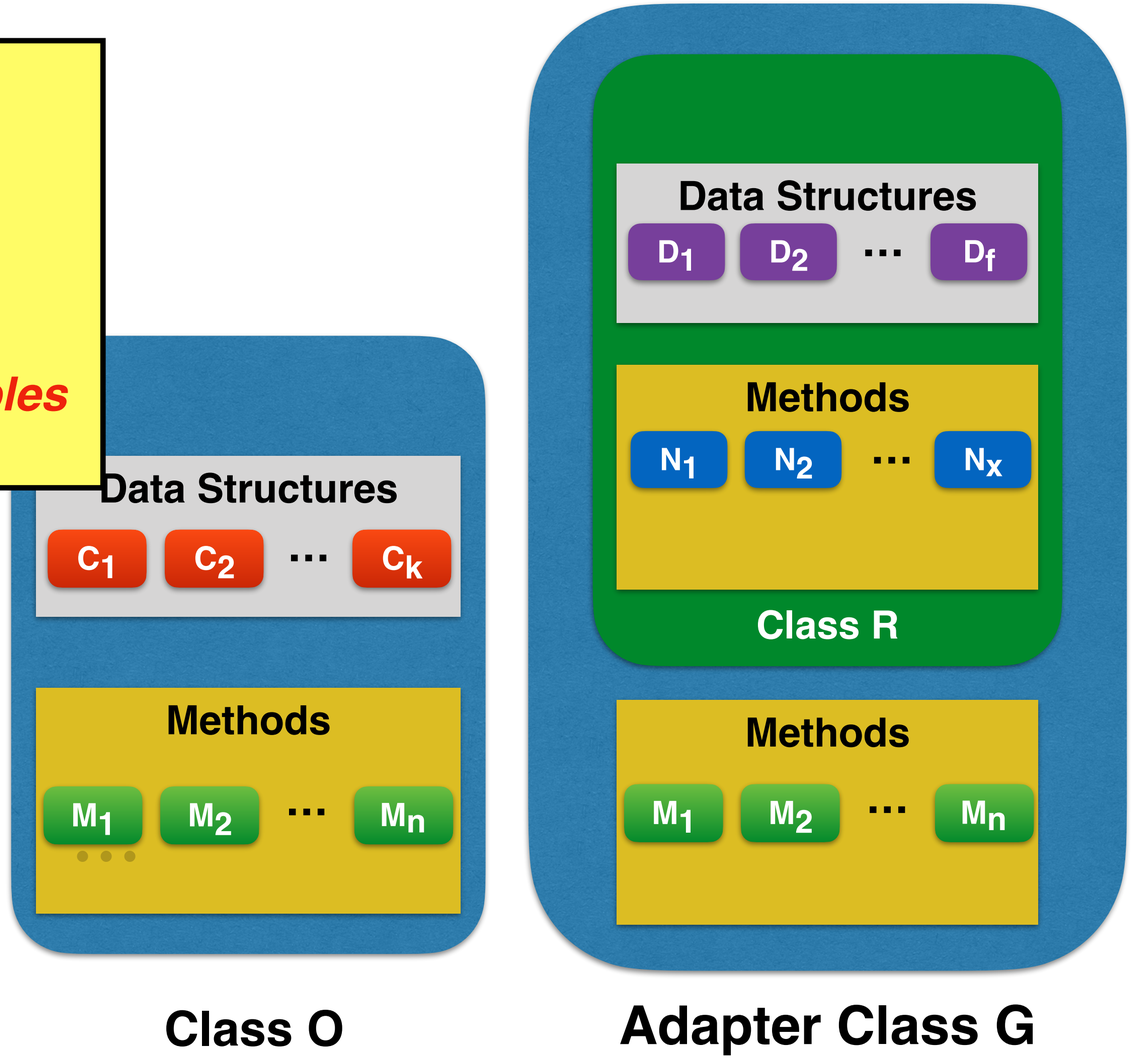


Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts
   a) Return *equivalent value* for equivalent history
   b) Drives *all* objects to *equivalent states*
   c) Creates an equivalent *aliasing* between *all variables* and their *dereferences*

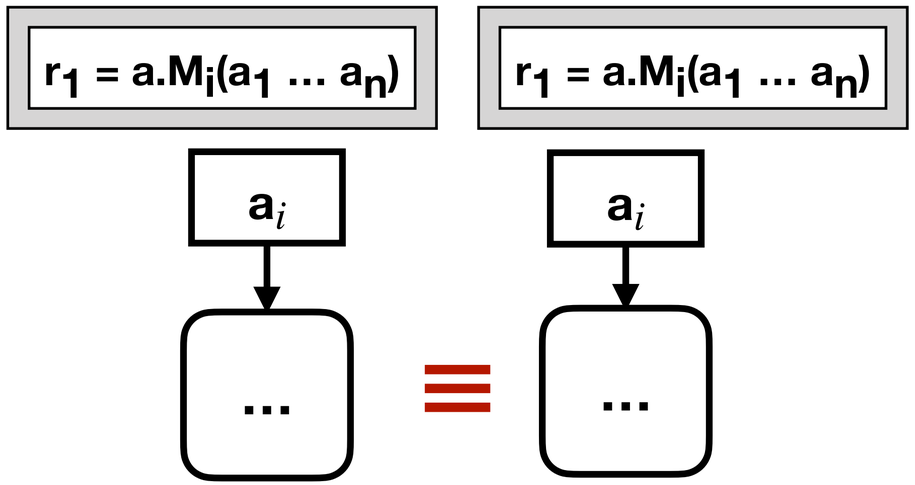
$r_1 = a.M_i(a_1 \ldots a_n)$

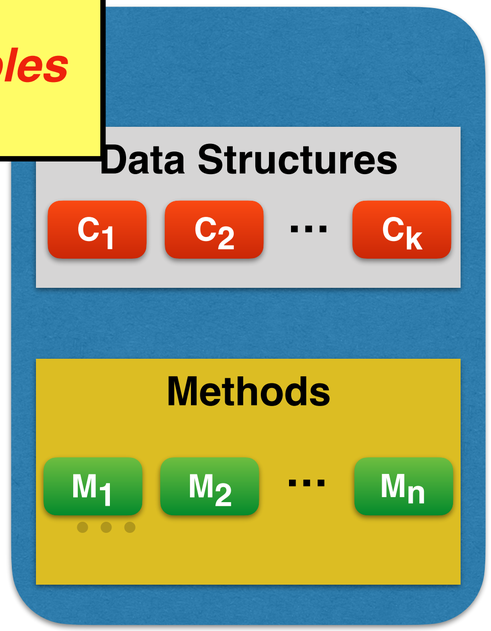
$r_1 = a.M_i(a_1 \ldots a_n)$



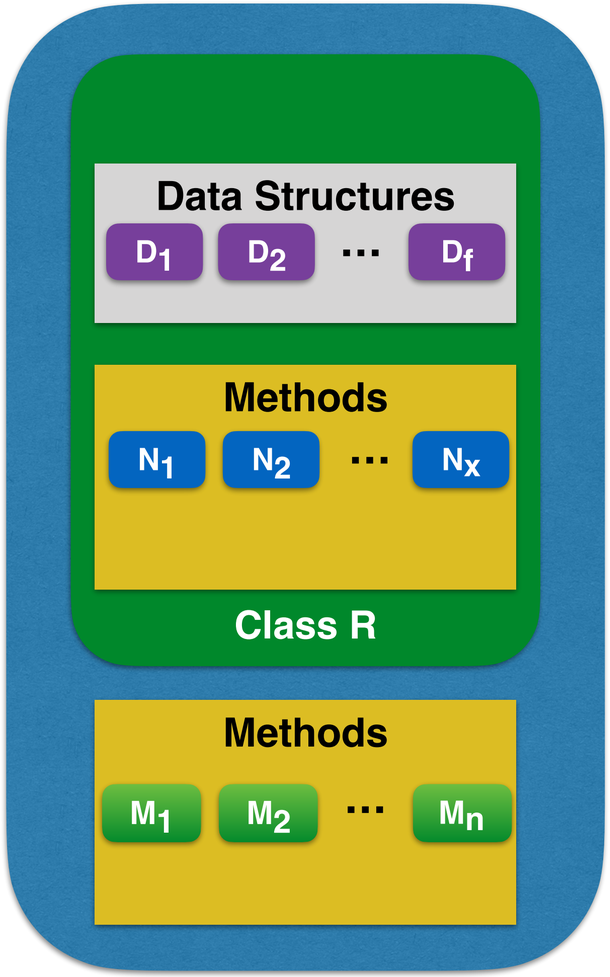
Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts
   a) Return *equivalent value* for equivalent history
   b) Drives *all* objects to *equivalent states*
   c) Creates an equivalent *aliasing* between *all variables* and their *dereferences*





Class O



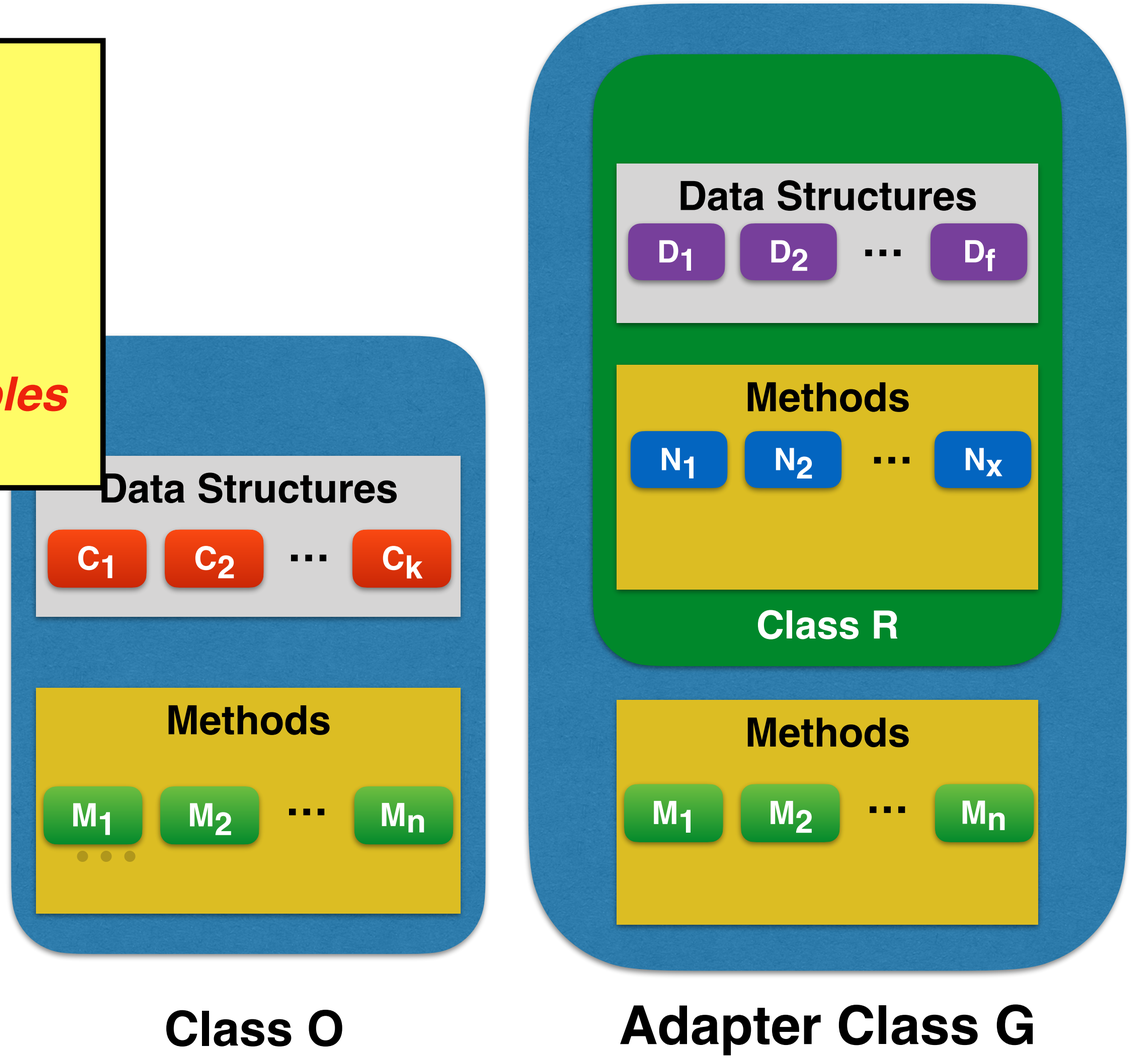
Adapter Class G

Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts
   a) Return *equivalent value* for equivalent history
   b) Drives *all* objects to *equivalent states*
   c) Creates an equivalent *aliasing* between *all variables* and their *dereferences*

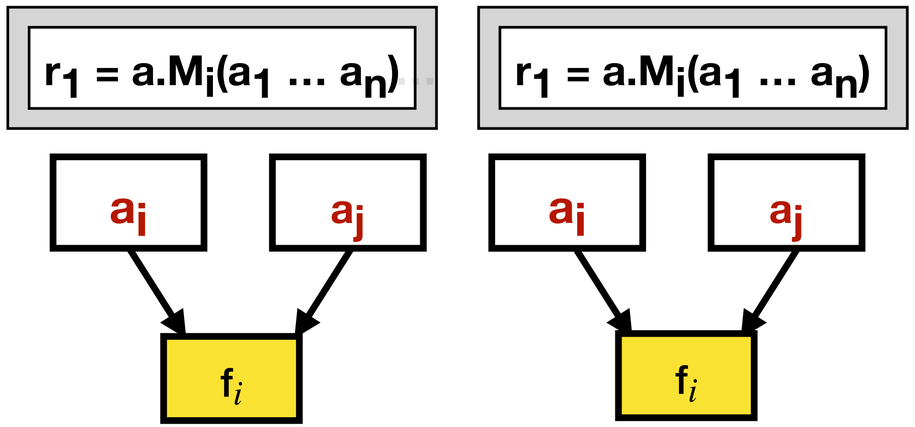
$r_1 = a.M_i(a_1 \ldots a_n)$

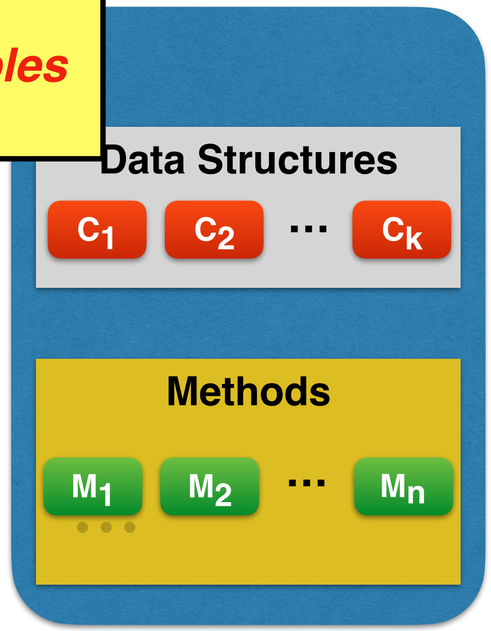
$r_1 = a.M_i(a_1 \ldots a_n)$



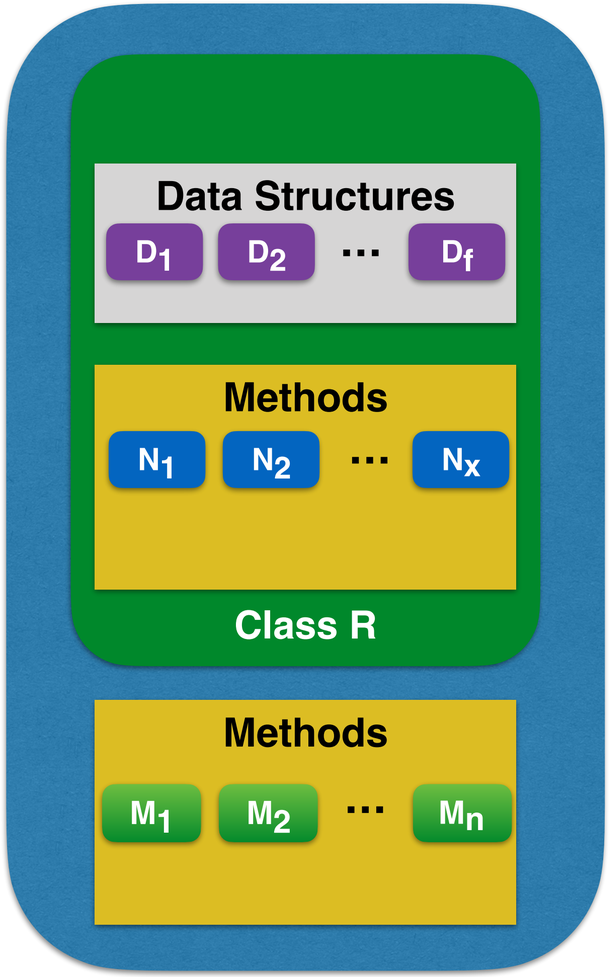
Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts
   a) Return *equivalent value* for equivalent history
   b) Drives *all* objects to *equivalent states*
   c) Creates an equivalent *aliasing* between *all variables* and their *dereferences*





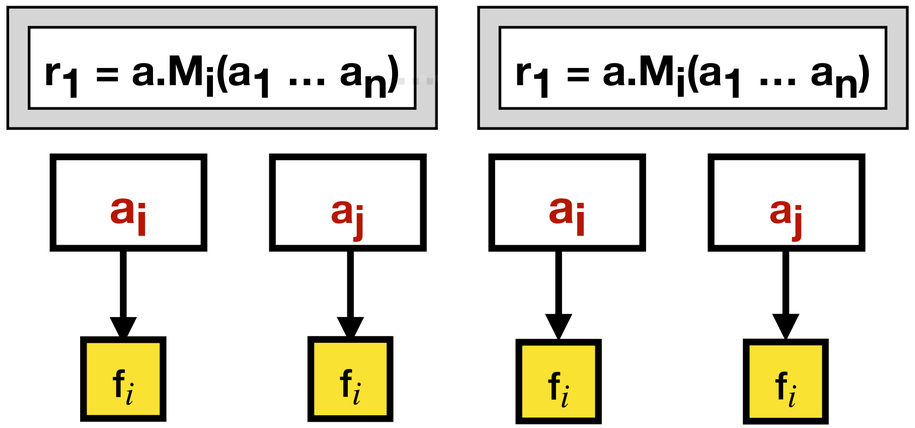
Class O

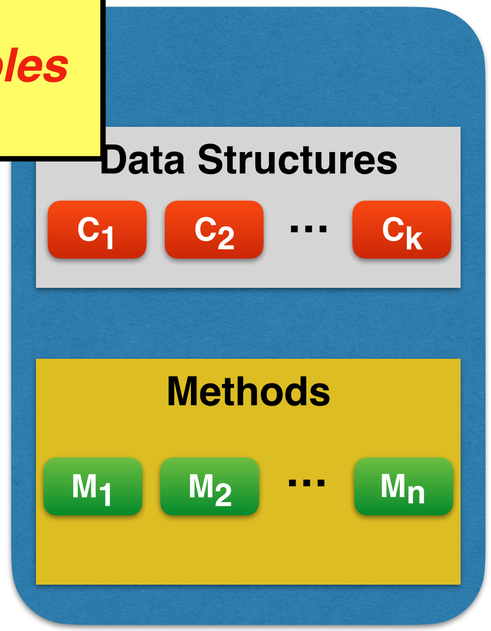


Adapter Class G

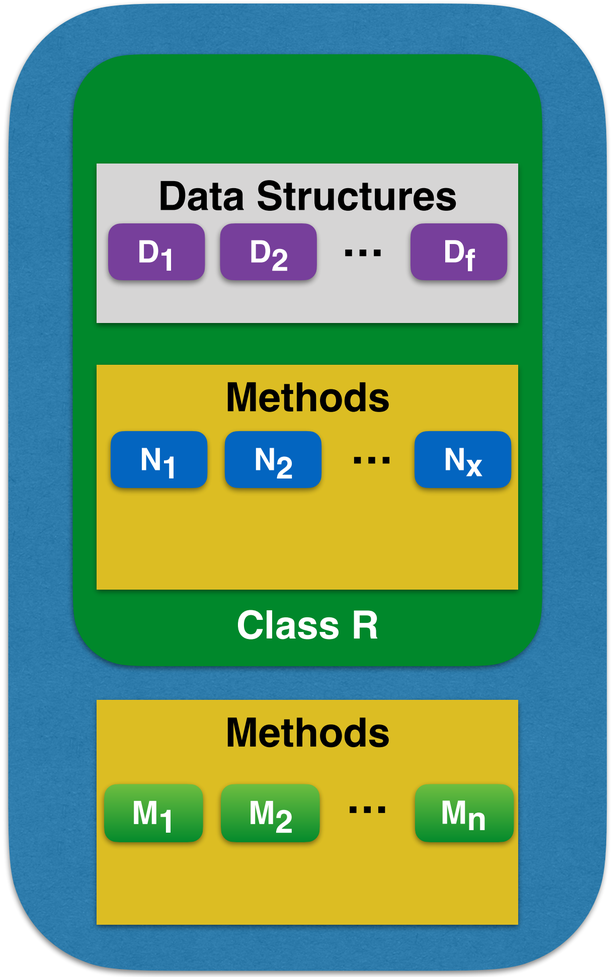
Adapter Class Requirements

1. Identical API signatures
2. Built using APIs from class R
3. API equivalence under all contexts
   a) Return *equivalent value* for equivalent history
   b) Drives *all* objects to *equivalent states*
   c) Creates an equivalent *aliasing* between *all variables* and their *dereferences*





Class O



Adapter Class G

Class replacement example

Class replacement example

```
int x1, y1, x2, y2;

Box (int a, int b, int c, int d) {
   x1 = a; y1 = b;  x2 = c; y2 = d;
}

int length () {
  return x2 - x1;
}

Box expand (int val) {
  Box b = new Box (x1 - val, y1 - val,
                   x2 + val, y2 + val);
  return b;
}

...
```

BOX

Class replacement example

int x1, y1, x2, y2;

```
Box (int a, int b, int c, int d) {
   x1 = a; y1 = b;  x2 = c; y2 = d;
}

int length () {
  return x2 - x1;
}

Box expand (int val) {
  Box b = new Box (x1 - val, y1 - val,
                   x2 + val, y2 + val);
  return b;
}
```

…

BOX

int x, y, width, height;

```
Rectangle (Rectangle r) {
 this (r.x, r.y, r.width, r.height);
}

Rectangle (int a, int b, int c, int d) {
 x = a; y = b; width = c; height = d;
}

int width () { return width;}

Rectangle shrink (int h, int v) {
 x += h; width -= (h+h);
 y += v; height -=(v+v);
 return this;
}
```

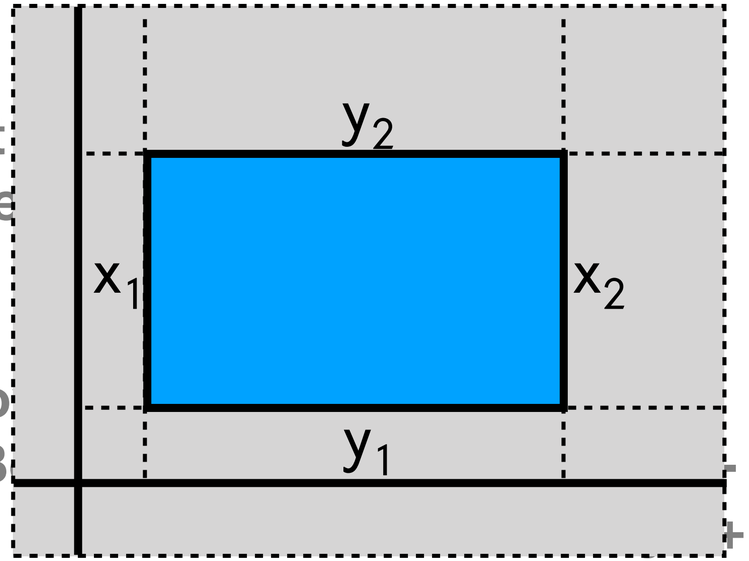
…

RECTANGLE

Class replacement example

int x1, y1, x2, y2;

```
Box (int a, int b, int c, int d) {
   x1 = a; y1 = b;  x2 = c; y2 = d;
}

int
  re
}

Bo
  B                                  - val,
                                     + val);
  return b;
}
```

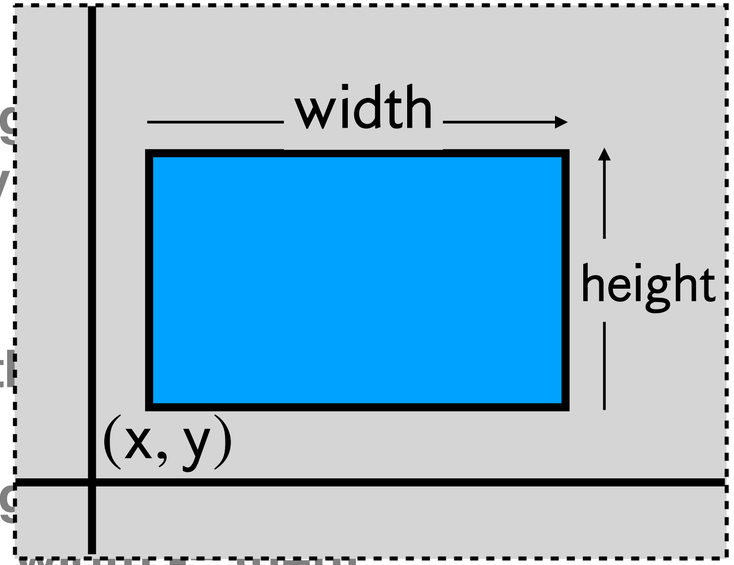


...

BOX

int x, y, width, height;

```
Rectangle (Rectangle r) {
 this (r.x, r.y, r.width, r.height);
}

Rectang
 x = a; y
}

int widt

Rectang
 x += h; width -= (h+h);
 y += v; height -=(v+v);
 return this;
}
```



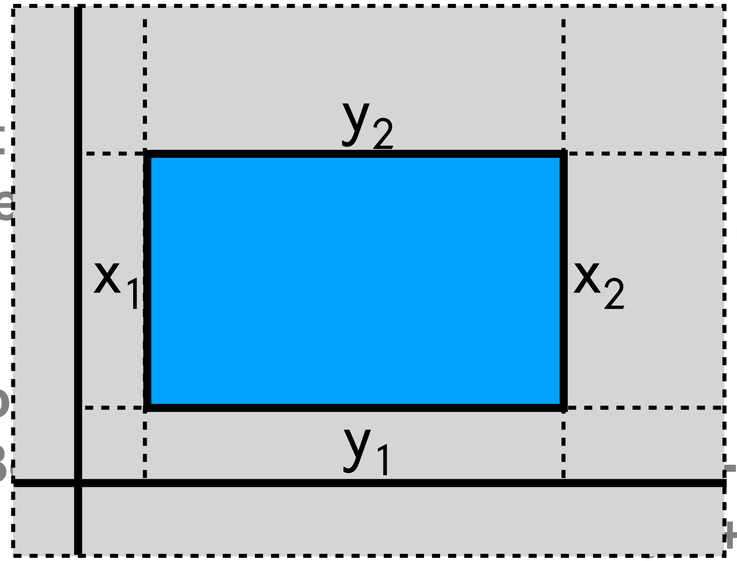
...

RECTANGLE

Class replacement example

int x1, y1, x2, y2;

```
Box (int a, int b, int c, int d) {
   x1 = a; y1 = b;  x2 = c; y2 = d;
}

int
  re
}

Bo
  B                                  - val,
                                     + val);
  return b;
}
```



...

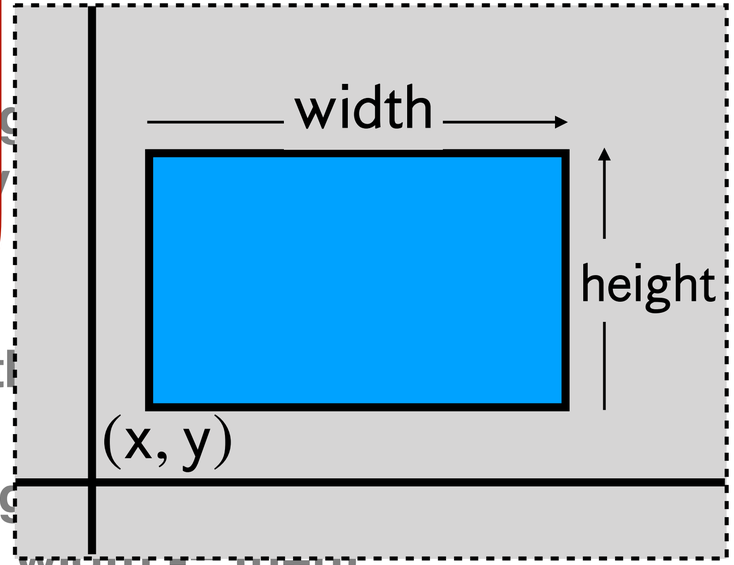
BOX

INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + width$$
$$y_2 = y + height$$

int x, y, width, height;

```
gle (Rectangle r) {
x, r.y, r.width, r.height);

int widt

Rectang
 x += h; width -= (h+h);
 y += v; height -=(v+v);
 return this;
}
```



...

RECTANGLE

Class replacement example

int x1, y1, x2, y2;

```
Box (int a, int b, int c, int d) {
   x1 = a; y1 = b;  x2 = c; y2 = d
}

int length () {
  return x2 - x1;
}

Box expand (int val) {
  Box b = new Box (x1 - val, y1 - val,
                   x2 + val, y2 + val);
  return b;
}
```

…

BOX

INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + \text{width}$$
$$y_2 = y + \text{height}$$

int x, y, width, height;

```
gle (Rectangle r) {
x, r.y, r.width, r.height);


gle (int a, int b, int c, int d) {
= b; width = c; height = d;


int width () { return width;}

Rectangle shrink (int h, int v) {
 x += h; width -= (h+h);
 y += v; height -=(v+v);
 return this;
}
```

…

RECTANGLE

Class replacement example

int x1, y1, x2, y2;

```
Box (int a, int b, int c, int d) {
   x1 = a; y1 = b;  x2 = c; y2 = d
}

int length () {
  return x2 - x1;
}

Box expand (int val) {
  Box b = new Box (x1 - val, y1 - val,
                   x2 + val, y2 + val);
  return b;
}
```

…

BOX

INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$

$$y_1 = y$$

$$x_2 = x + \text{width}$$

$$y_2 = y + \text{height}$$

int x, y, width, height;

```
gle (Rectangle r) {
x, r.y, r.width, r.height);


gle (int a, int b, int c, int d) {
 = b; width = c; height = d;

int width () { return width;}

Rectangle shrink (int h, int v) {
 x += h; width -= (h+h);
 y += v; height -=(v+v);
 return this;
}
```

…

RECTANGLE

Class replacement example

```
int x1, y1, x2, y2;

Box (int a, int b, int c, int d) {
   x1 = a; y1 = b;  x2 = c; y2 = d;
}

int length () {
  return x2 - x1;
}

Box expand (int val) {
  Box b = new Box (x1 - val, y1 - val,
                   x2 + val, y2 + val);
  return b;
}

...
```

BOX

```
Rectangle rect;

Adapter (Rectangle r) { rect = r; }

Adapter (int a, int b, int c, int d) {
   rect = new Rectangle(a, b, c-a, d-b);
}

int length () {
  int ret = rect.width();
  return ret;
}

Adapter expand (int val) {
  Rectangle ret = new Rectangle (rect);
  ret.shrink (-val, -val);
  return new Adapter (ret);
}

...
```

Adapter

Class replacement example

```
int x1, y1, x2, y2;

Box (int a, int b, int c, int d) {
   x1 = a; y1 = b;  x2 = c; y2 = d;
}

int length () {
  return x2 - x1;
}

Box expand (int val) {
  Box b = new Box (x1 - val, y1 - val,
                   x2 + val, y2 + val);
  return b;
}

...
```

BOX

```
Rectangle rect;

Adapter (Rectangle r) { rect = r; }

Adapter (int a, int b, int c, int d) {
   rect = new Rectangle(a, b, c-a, d-b);
}

int length () {
  int ret = rect.width();
  return ret;
}

Adapter expand (int val) {
  Rectangle ret = new Rectangle (rect);
  ret.shrink (-val, -val);
  return new Adapter (ret);
}

...
```

Adapter

Class replacement example

int x1, y1, x2, y2;

```
Box (int a, int b, int c, int d) {
   x1 = a; y1 = b;  x2 = c; y2 = d;
}

int length () {
  return x2 - x1;
}

Box expand (int val) {
  Box b = new Box (x1 - val, y1 - val,
                   x2 + val, y2 + val);
  return b;
}

                   ...
```

Rectangle rect;

```
Adapter (Rectangle r) { rect = r; }

Adapter (int a, int b, int c, int d) {
   rect = new Rectangle(a, b, c-a, d-b);
}

int length () {
  int ret = rect.width();
  return ret;
}

Adapter expand (int val) {
  Rectangle ret = new Rectangle (rect);
  ret.shrink (-val, -val);
  return new Adapter (ret);
}                                    Adapter
                   ...
```

Class replacement example

int x1, y1, x2, y2;

```
Box (int a, int b, int c, int d) {
   x1 = a; y1 = b;  x2 = c; y2 = d;
}

int length () {
  return x2 - x1;
}

Box expand (int val) {
  Box b = new Box (x1 - val, y1 - val,
                   x2 + val, y2 + val)
  return b;
}

...
```

Rectangle rect;

```
Adapter (Rectangle r) { rect = r; }

Adapter (int a, int b, int c, int d) {
   rect = new Rectangle(a, b, c-a, d-b);
}

int length () {
  int ret = rect.width();
  return ret;
}

Adapter expand (int val) {
  Rectangle ret = new Rectangle (rect);
  ret.shrink (-val, -val);
  return new Adapter (ret);
}
                                    Adapter
...
```

Challenges for MASK

1. Establish an inter-class equivalence predicate

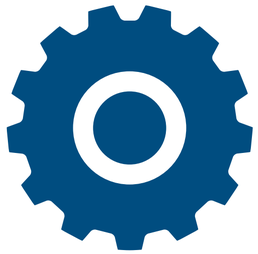
2. Synthesize the adapter class

    Verify correctness of ***all*** adapter methods with inter-class equivalence

Synthesizing the adapter class via Sketching

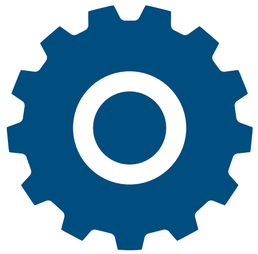
Synthesizing adapter class via sketching

Inter-class Equivalence Predicate

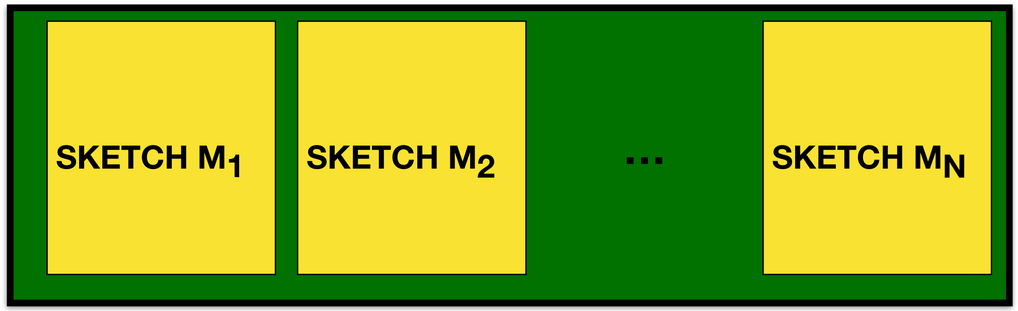


Synthesizing adapter class via sketching

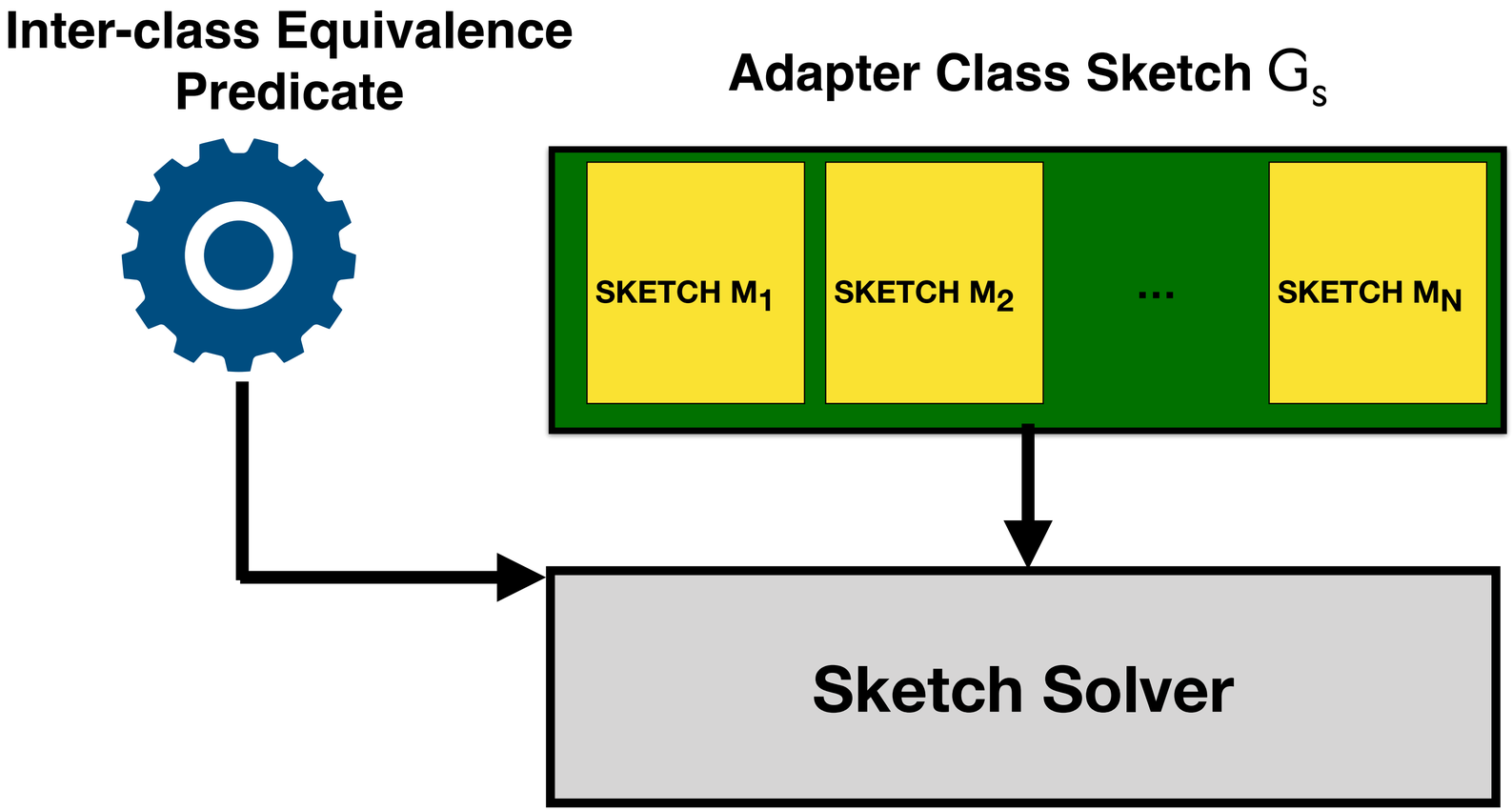
Inter-class Equivalence Predicate



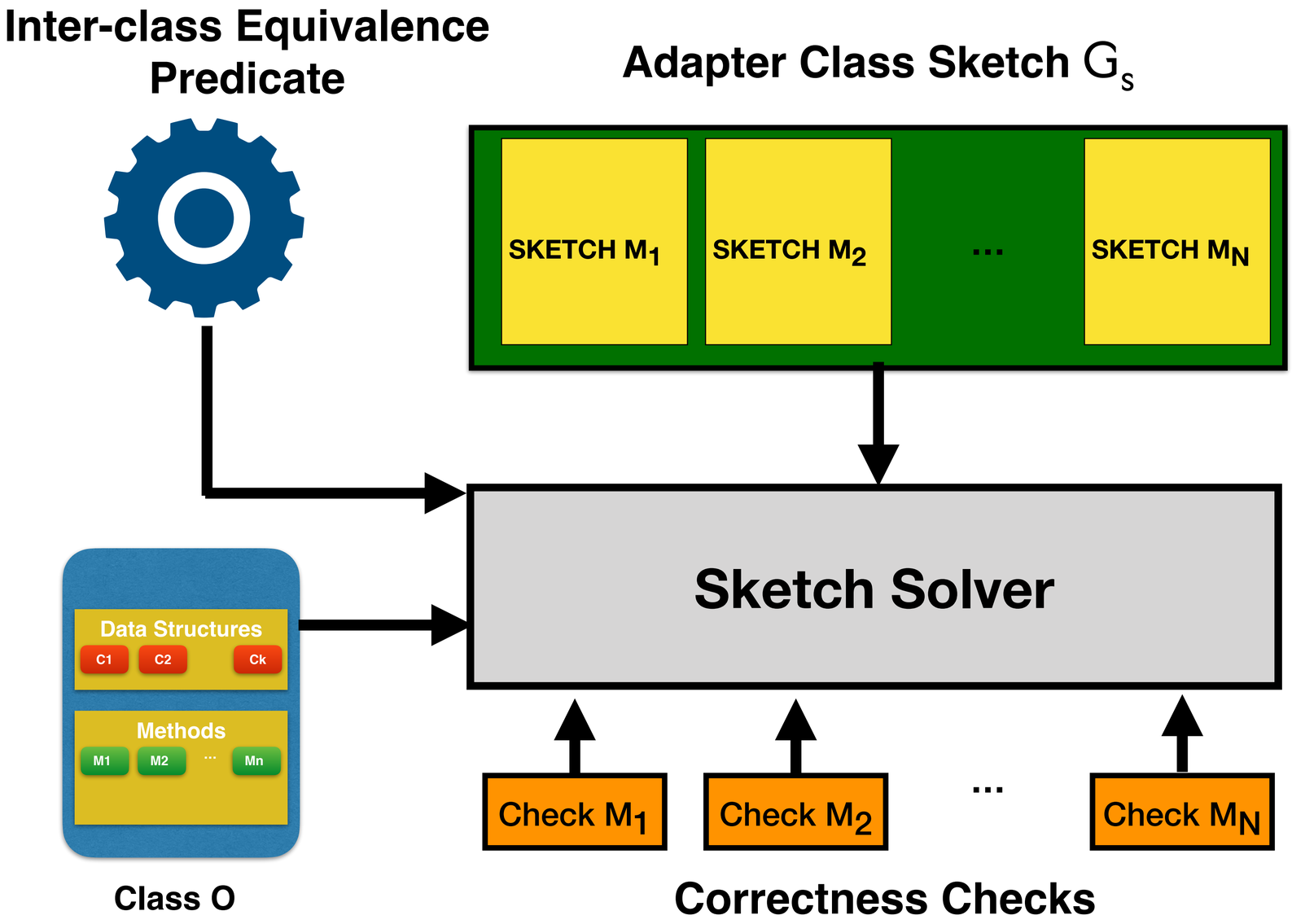
Adapter Class Sketch G_s



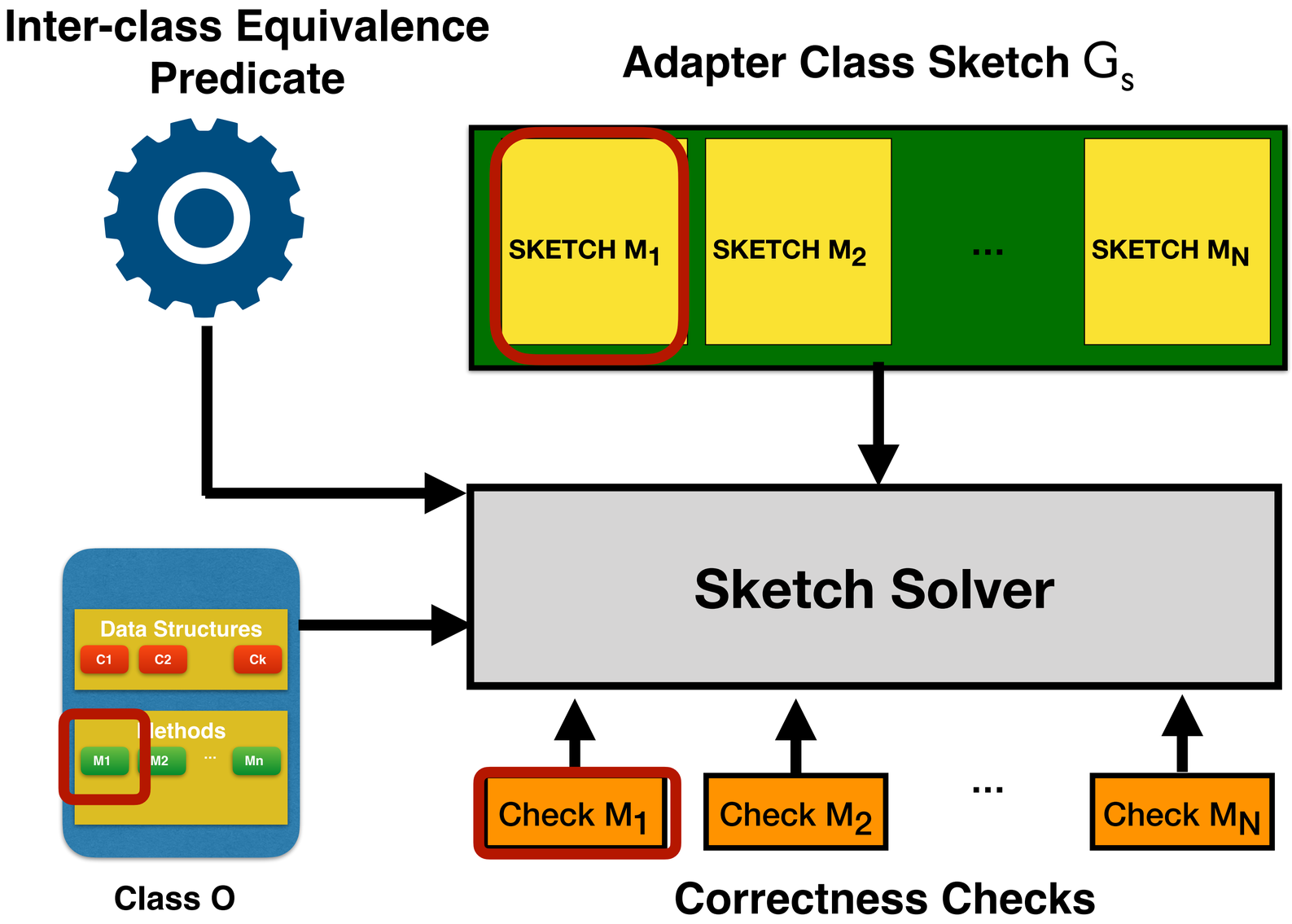
Synthesizing adapter class via sketching



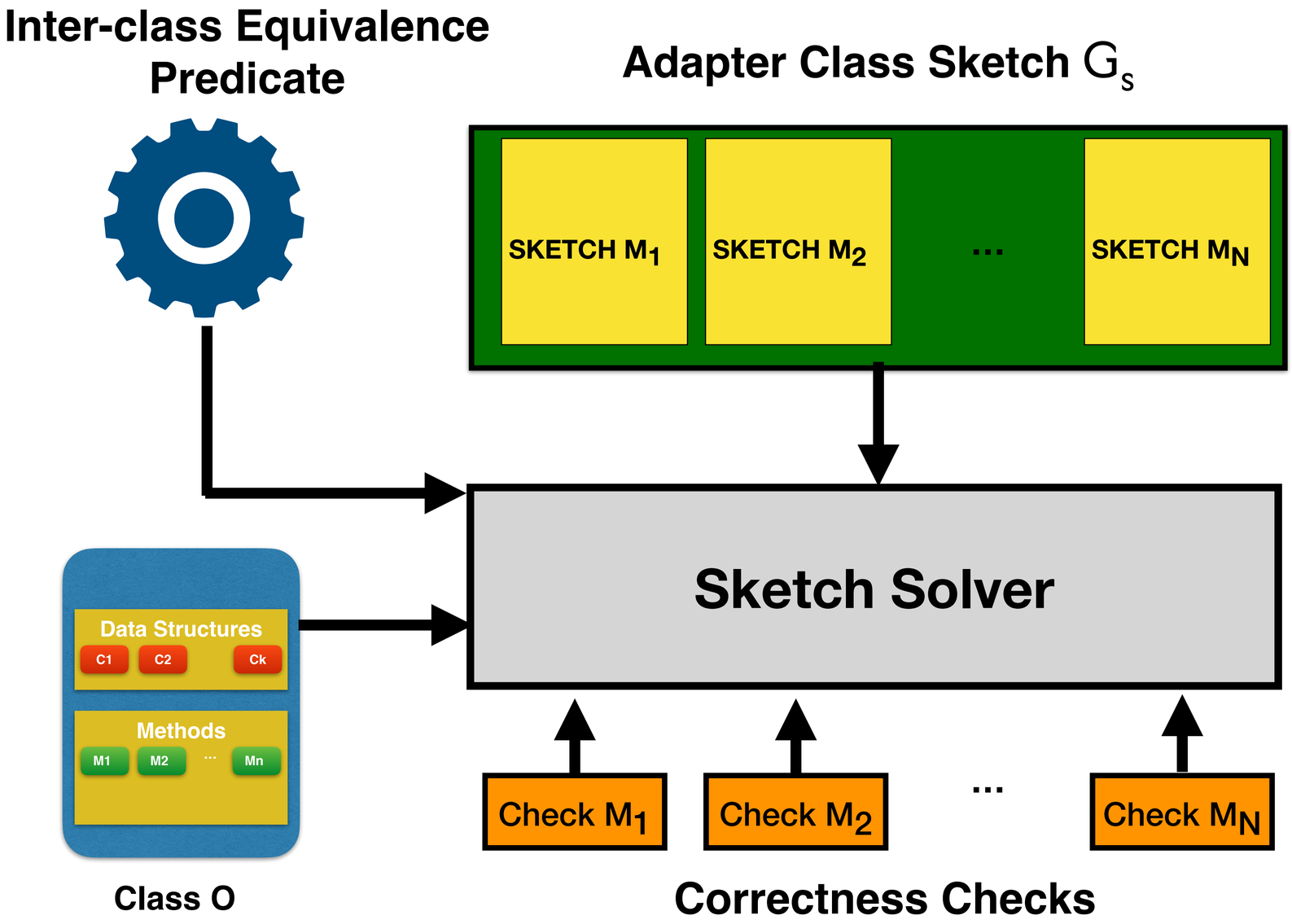
Synthesizing adapter class via sketching



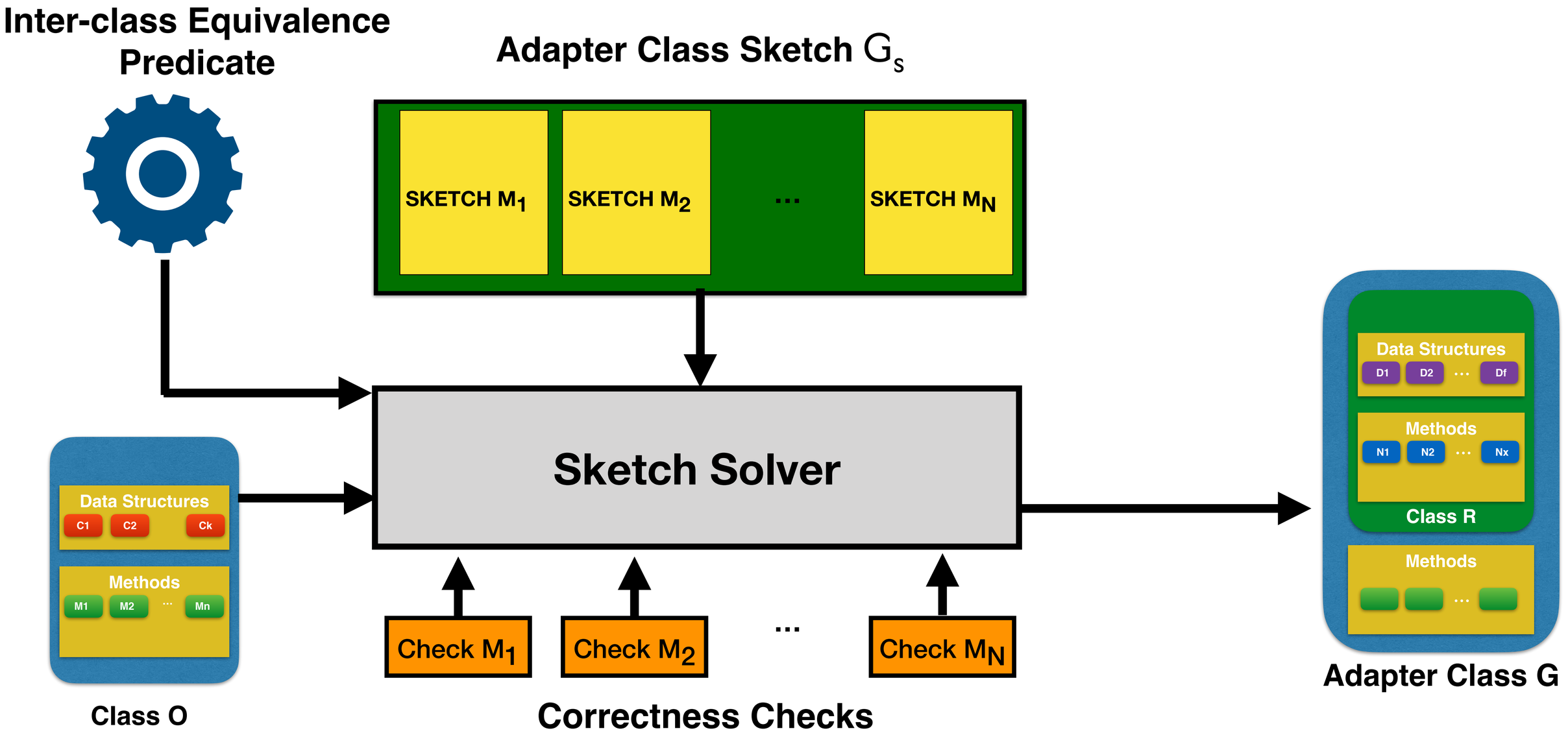
Synthesizing adapter class via sketching



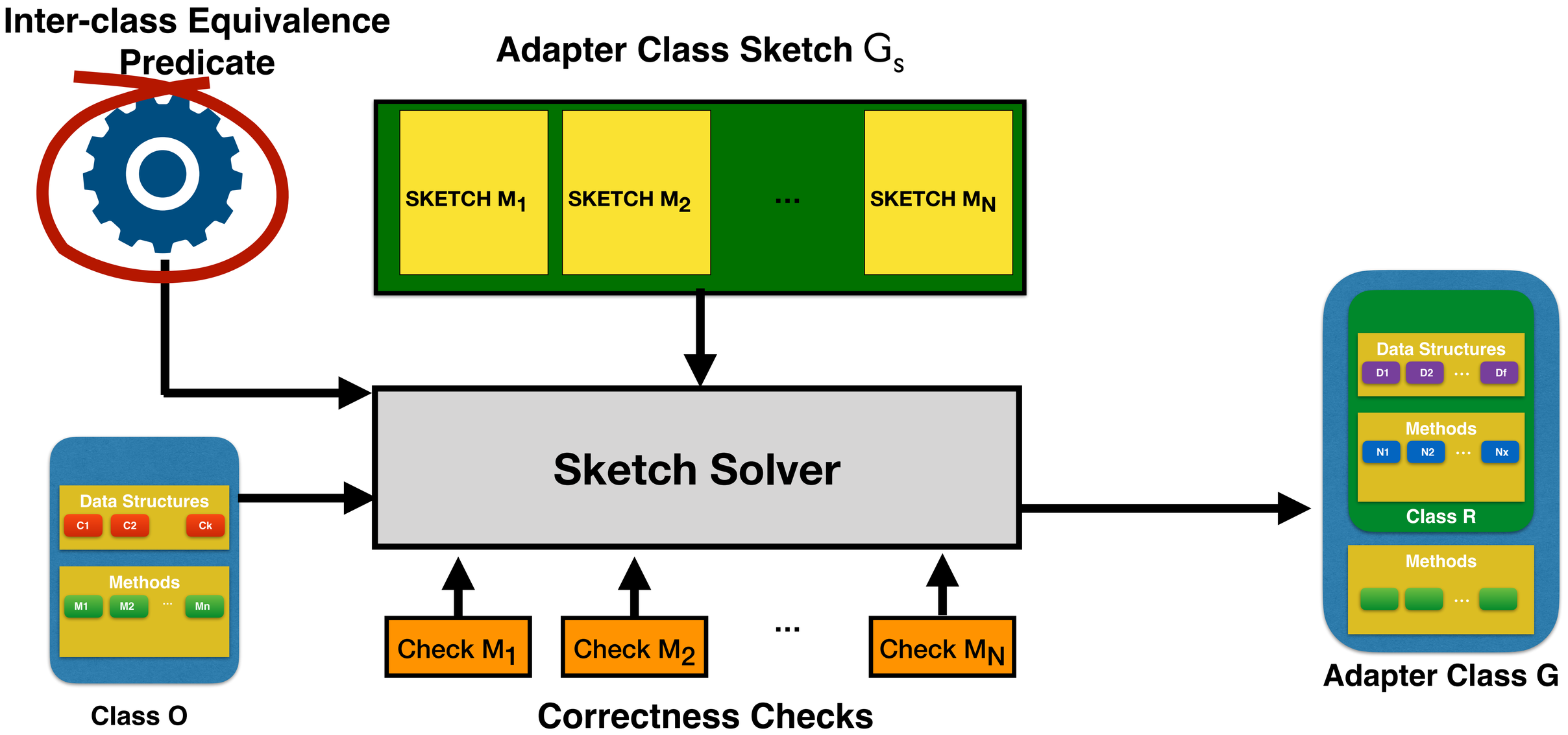
Synthesizing adapter class via sketching



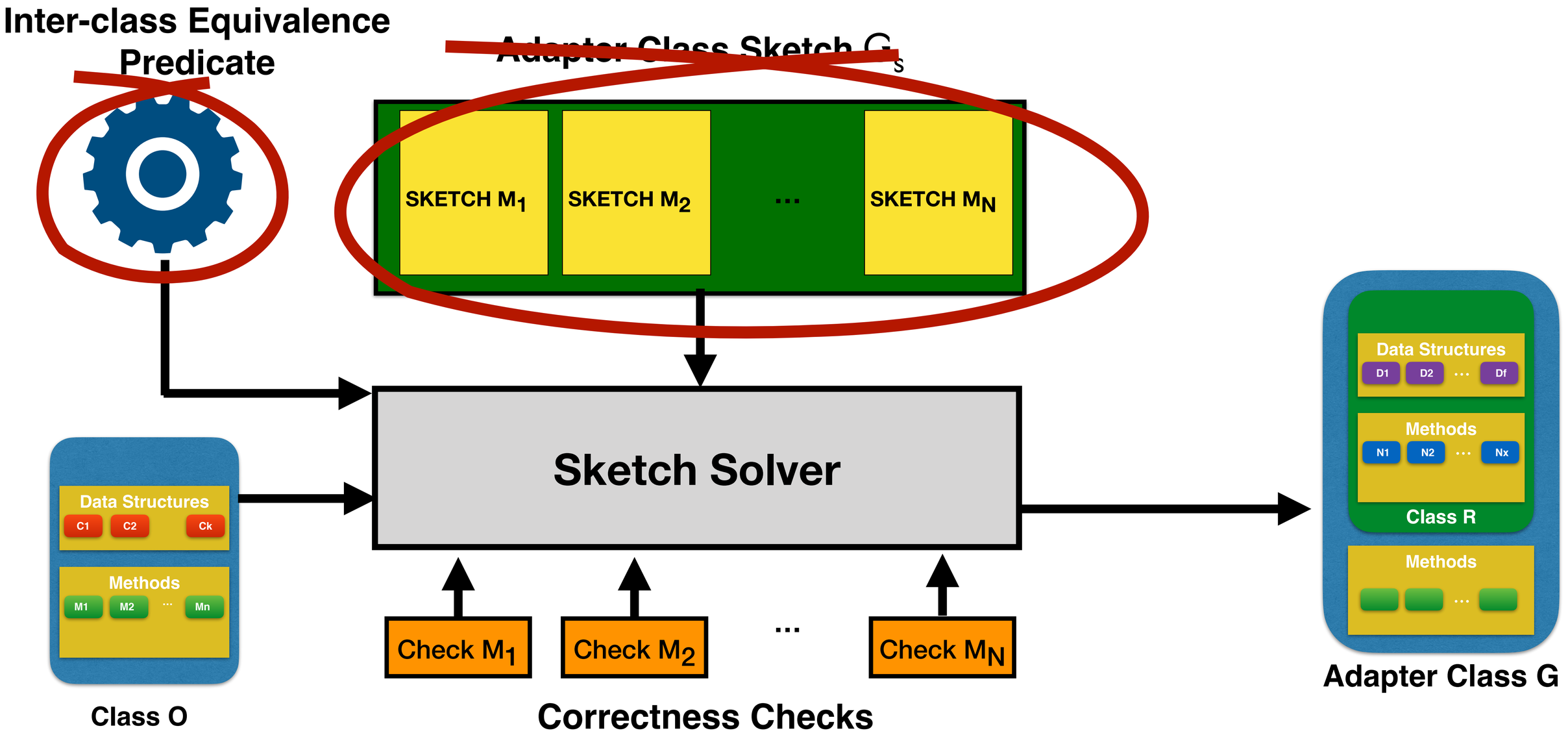
Synthesizing adapter class via sketching



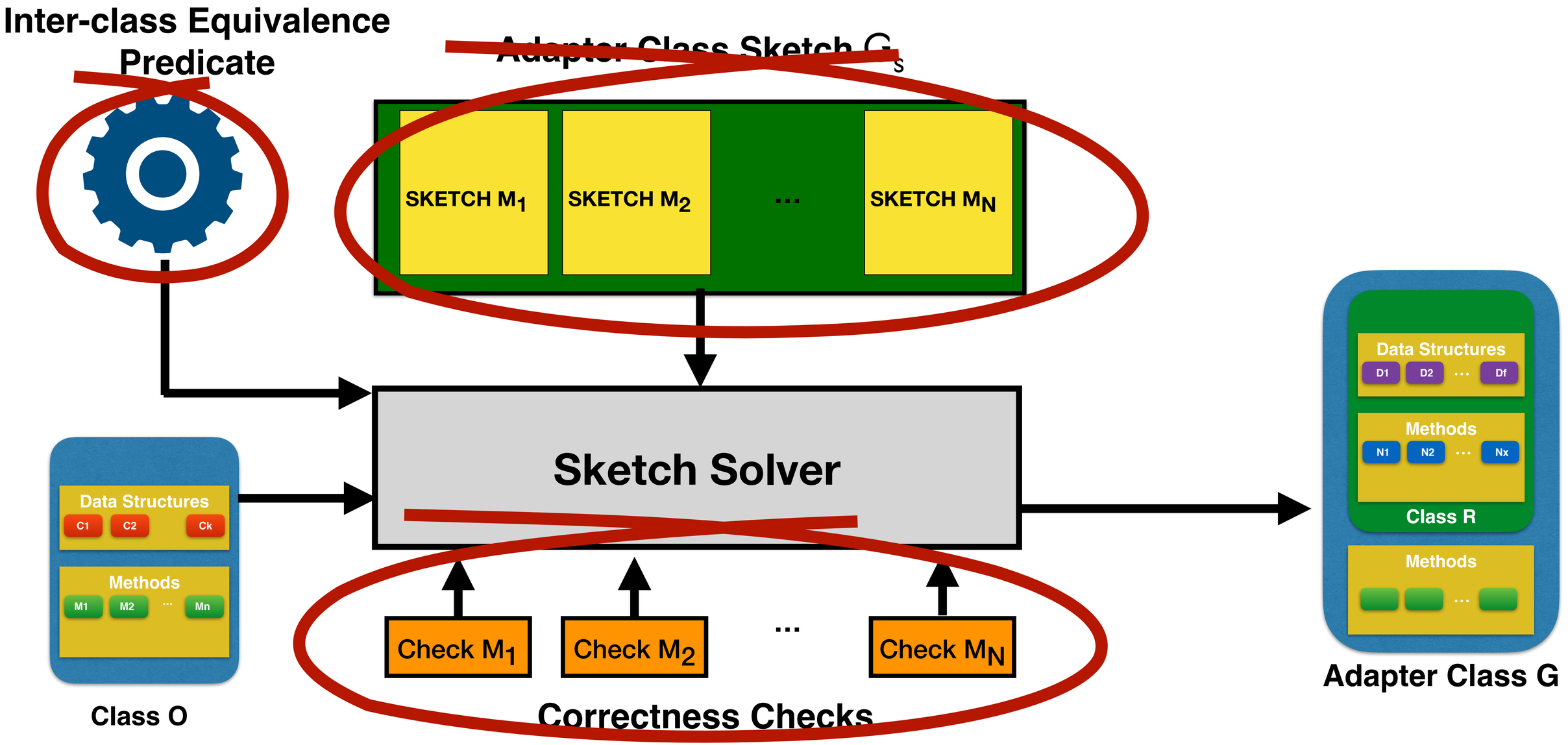
Synthesizing adapter class via sketching



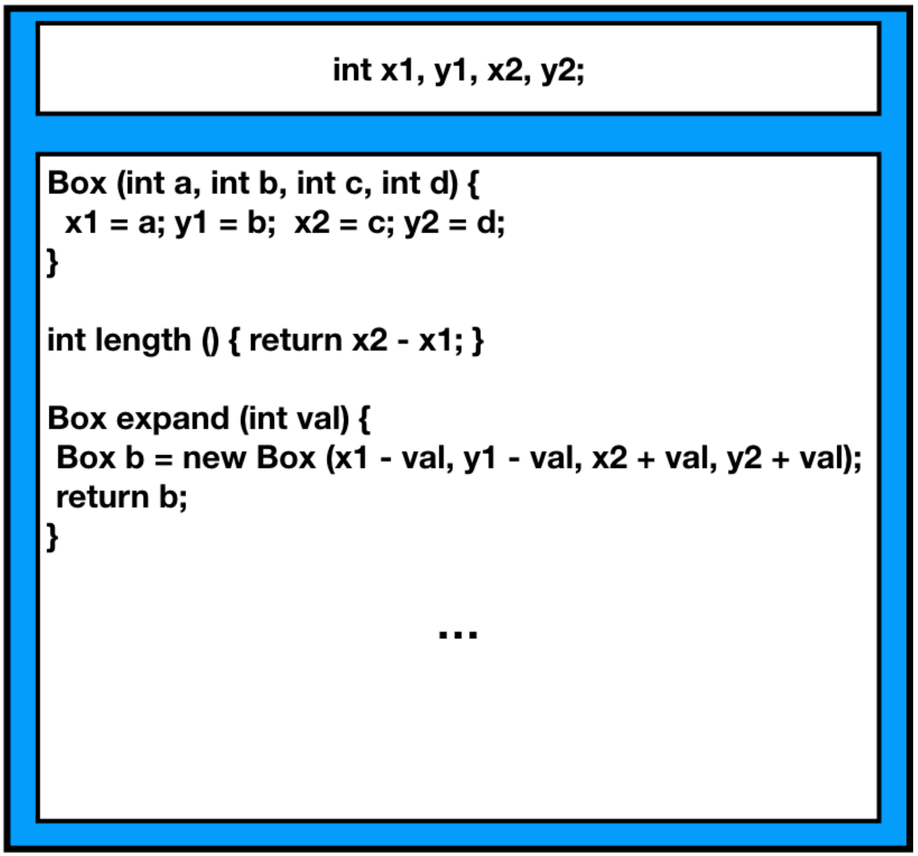
Synthesizing adapter class via sketching



Synthesizing adapter class via sketching

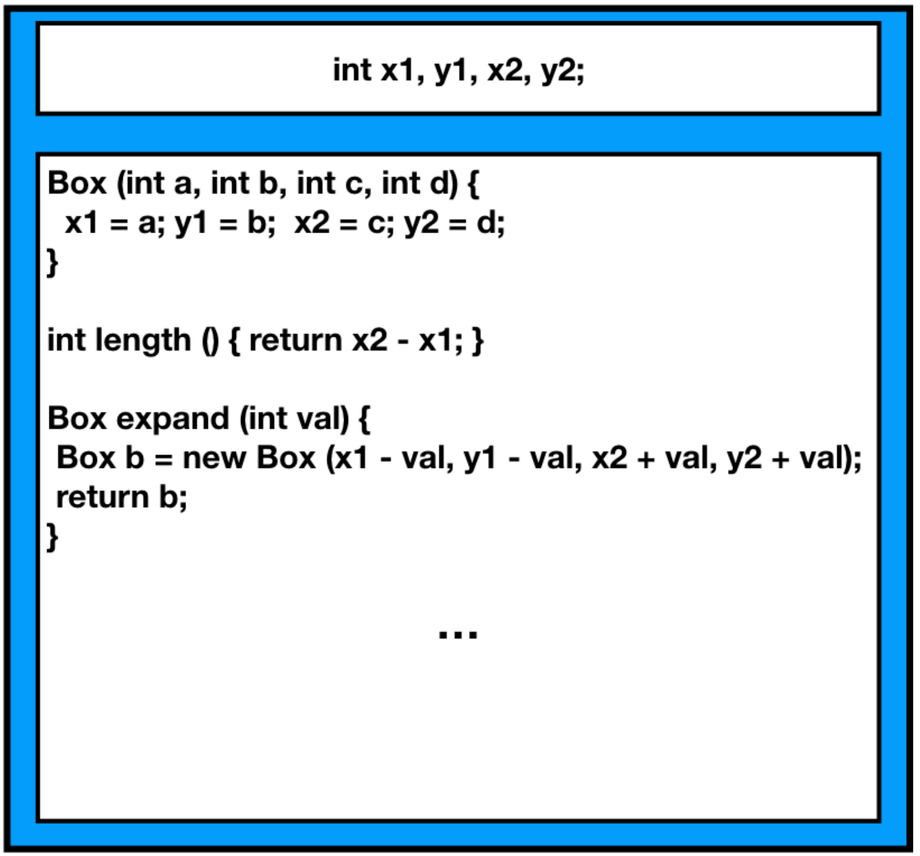


Building the adapter method sketch



Building the adapter method sketch

Box expand(int val)



Box

Building the adapter method sketch

Box expand(int val)

```
int x1, y1, x2, y2;

Box (int a, int b, int c, int d) {
  x1 = a; y1 = b;  x2 = c; y2 = d;
}

int length () { return x2 - x1; }

Box expand (int val) {
 Box b = new Box (x1 - val, y1 - val, x2 + v
 return b;
}

...
```

Box

```
Adapter expand(int val) {
   Rectangle retRect1 = null; int retInt1 = 0;
   switch (??) {
      case 1: retRect1 = new Rectangle ( ); break;
      case 2: retRect1 = new Rectangle (this.rect); break;
      case 3: retInt1 = width(); break;
      case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
break;
                              ...
      default:
   }

   Rectangle retRect2 = null; int retInt2 = 0;
   switch (??) { ... }

   return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

SKETCH

Building the adapter method sketch

Box expand(int val)

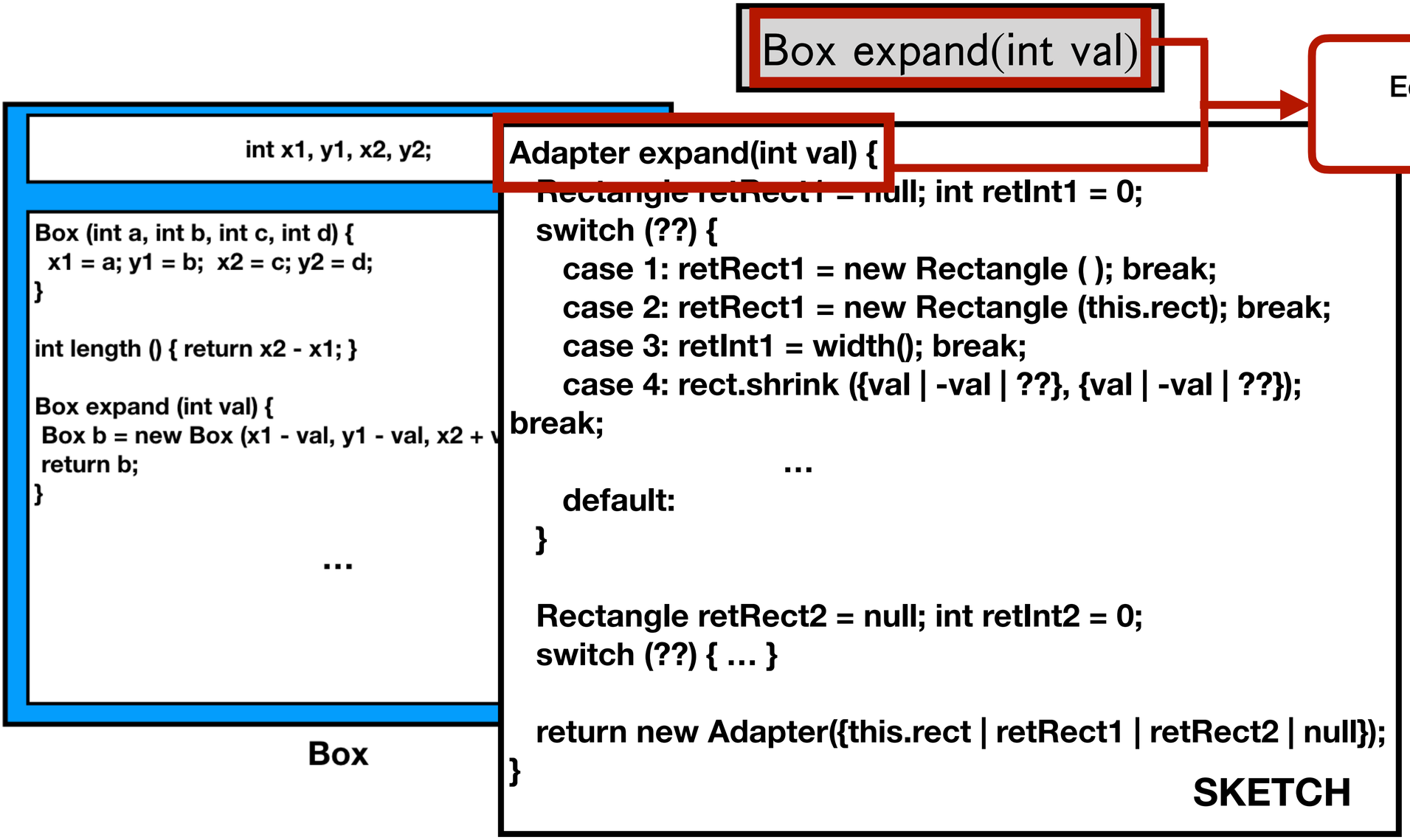
```
int x1, y1, x2, y2;

Box (int a, int b, int c, int d) {
  x1 = a; y1 = b;  x2 = c; y2 = d;
}

int length () { return x2 - x1; }

Box expand (int val) {
 Box b = new Box (x1 - val, y1 - val, x2 + v
 return b;
}

...
```

Box

```
Adapter expand(int val) {
  Rectangle retRect1 = null; int retInt1 = 0;
  switch (??) {
     case 1: retRect1 = new Rectangle ( ); break;
     case 2: retRect1 = new Rectangle (this.rect); break;
     case 3: retInt1 = width(); break;
     case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
break;
                          ...
     default:
  }

  Rectangle retRect2 = null; int retInt2 = 0;
  switch (??) { ... }

  return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

SKETCH

Building the adapter method sketch



Building the adapter method sketch

Box expand(int val)

```
int x1, y1, x2, y2;

Box (int a, int b, int c, int d) {
  x1 = a; y1 = b;  x2 = c; y2 = d;
}

int length () { return x2 - x1; }

Box expand (int val) {
 Box b = new Box (x1 - val, y1 - val, x2 + v
 return b;
}

...
```

Box

```
Adapter expand(int val) {
   Rectangle retRect1 = null; int retInt1 = 0;
   switch (??) {
      case 1: retRect1 = new Rectangle ( ); break;
      case 2: retRect1 = new Rectangle (this.rect); break;
      case 3: retInt1 = width(); break;
      case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
break;
                              ...
      default:
   }

   Rectangle retRect2 = null; int retInt2 = 0;
   switch (??) { ... }

   return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

SKETCH

Building the adapter method sketch

Box expand(int val)

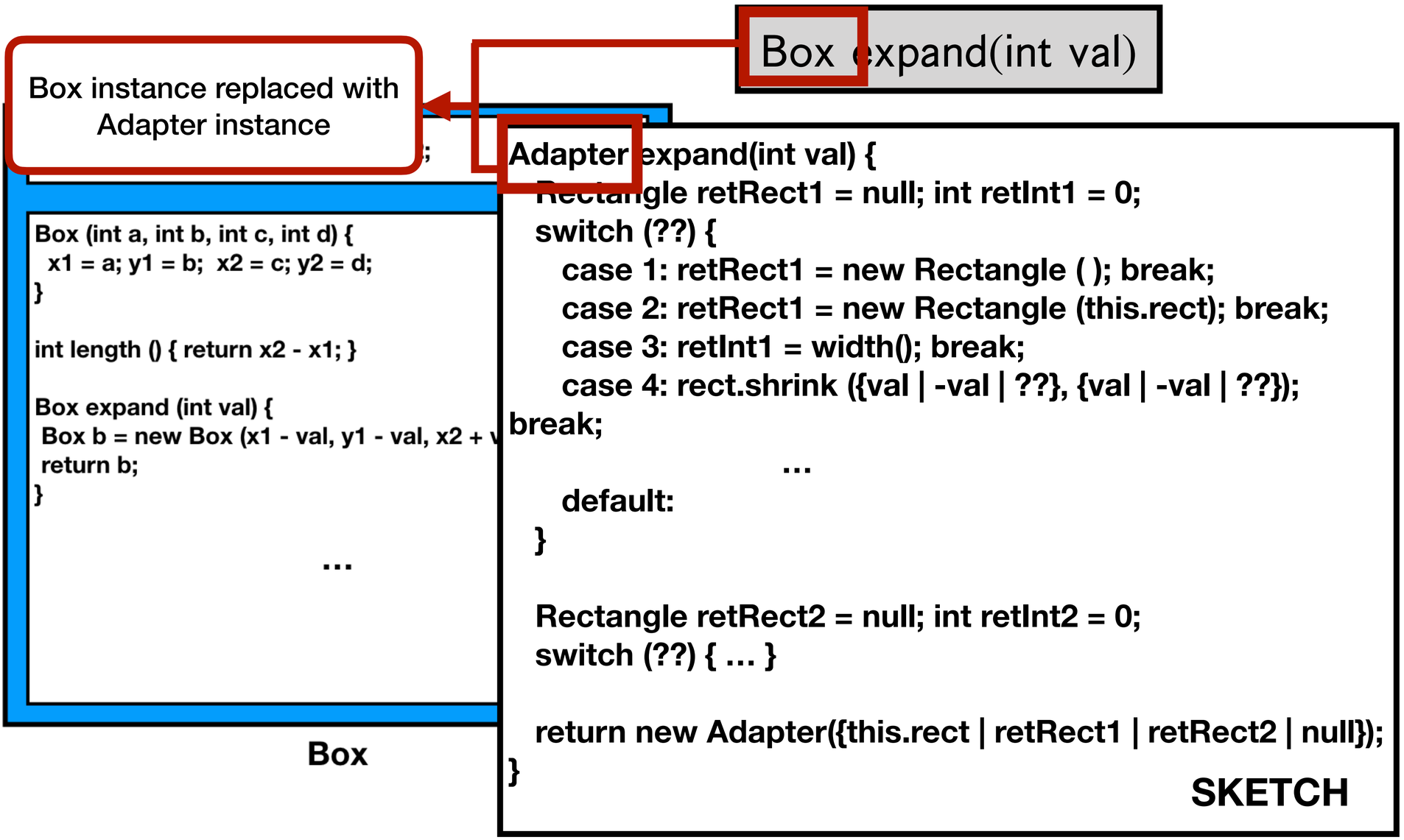
```
int x1, y1, x2, y2;

Box (int a, int b, int c, int d) {
  x1 = a; y1 = b;  x2 = c; y2 = d;
}

int length () { return x2 - x1; }

Box expand (int val) {
 Box b = new Box (x1 - val, y1 - val, x2 + v
 return b;
}

...
```

Box

```
Adapter expand(int val) {
  Rectangle retRect1 = null; int retInt1 = 0;
  switch (??) {
     case 1: retRect1 = new Rectangle ( ); break;
     case 2: retRect1 = new Rectangle (this.rect); break;
     case 3: retInt1 = width(); break;
     case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
break;
                          ...
     default:
  }

  Rectangle retRect2 = null; int retInt2 = 0;
  switch (??) { ... }

  return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

SKETCH

Building the adapter method sketch



Building the adapter method sketch

Box expand(int val)

```
int x1, y1, x2, y2;

Box (int a, int b, int c, int d) {
  x1 = a; y1 = b;  x2 = c; y2 = d;
}

int length () { return x2 - x1; }

Box expand (int val) {
 Box b = new Box (x1 - val, y1 - val, x2 + v
 return b;
}

...
```

Box

```
Adapter expand(int val) {
   Rectangle retRect1 = null; int retInt1 = 0;
   switch (??) {
      case 1: retRect1 = new Rectangle ( ); break;
      case 2: retRect1 = new Rectangle (this.rect); break;
      case 3: retInt1 = width(); break;
      case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
break;
                              ...
      default:
   }

   Rectangle retRect2 = null; int retInt2 = 0;
   switch (??) { ... }

   return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

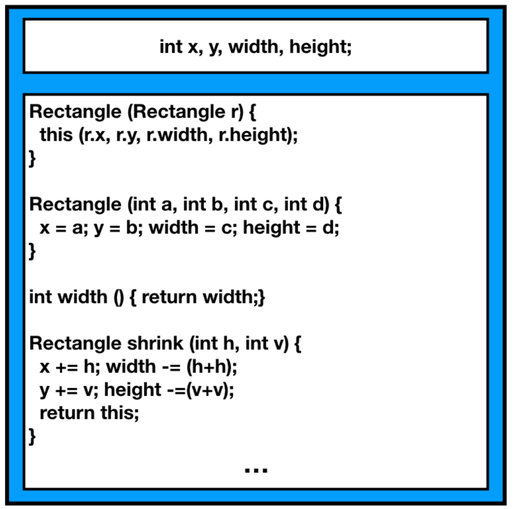
SKETCH

Building the adapter method sketch

Box expand(int c)

```
Adapter expand(int val) {
  Rectangle retRect1 = null; int retInt1 = 0;
  switch (??) {
    case 1: retRect1 = new Rectangle ( ); break;
    case 2: retRect1 = new Rectangle (this.rect); break;
    case 3: retInt1 = width(); break;
    case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
break;
                          …
    default:
  }

  Rectangle retRect2 = null; int retInt2 = 0;
  switch (??) { … }

  return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

SKETCH



Rectangle

Building the adapter method sketch

Box expand(int c)

```
Adapter expand(int val) {
  Rectangle retRect1 = null; int retInt1 = 0;
  switch (??) {
    case 1: retRect1 = new Rectangle ( ); break;
    case 2: retRect1 = new Rectangle (this.rect); break;
    case 3: retInt1 = width(); break;
    case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
break;
                          …
    default:
  }

  Rectangle retRect2 = null; int retInt2 = 0;
  switch (??) { … }

  return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

SKETCH

```
int x, y, width, height;

Rectangle (Rectangle r) {
  this (r.x, r.y, r.width, r.height);
}

Rectangle (int a, int b, int c, int d) {
  x = a; y = b; width = c; height = d;
}

int width () { return width;}

Rectangle shrink (int h, int v) {
  x += h; width -= (h+h);
  y += v; height -=(v+v);
  return this;
}
                          …
```

Rectangle

Building the adapter method sketch

Box expand(int c)

Choice to invoke a public method from Class R

```
Adapter expand(int val) {
  Rectangle retRect1 = null; int retInt1 = 0;
  switch (??) {
    case 1: retRect1 = new Rectangle ( ); break;
    case 2: retRect1 = new Rectangle (this.rect); break;
    case 3: retInt1 = width(); break;
    case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
... break;
                          ...
    default:
  }

  Rectangle retRect2 = null; int retInt2 = 0;
  switch (??) { ... }

  return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

SKETCH

```
int x, y, width, height;

Rectangle (Rectangle r) {
  this (r.x, r.y, r.width, r.height);
}

Rectangle (int a, int b, int c, int d) {
  x = a; y = b; width = c; height = d;
}

int width () { return width;}

Rectangle shrink (int h, int v) {
  x += h; width -= (h+h);
  y += v; height -=(v+v);
  return this;
}
                    ...
```

Rectangle

Building the adapter method sketch

Box expand(int c)

Choice to invoke a public method from Class R

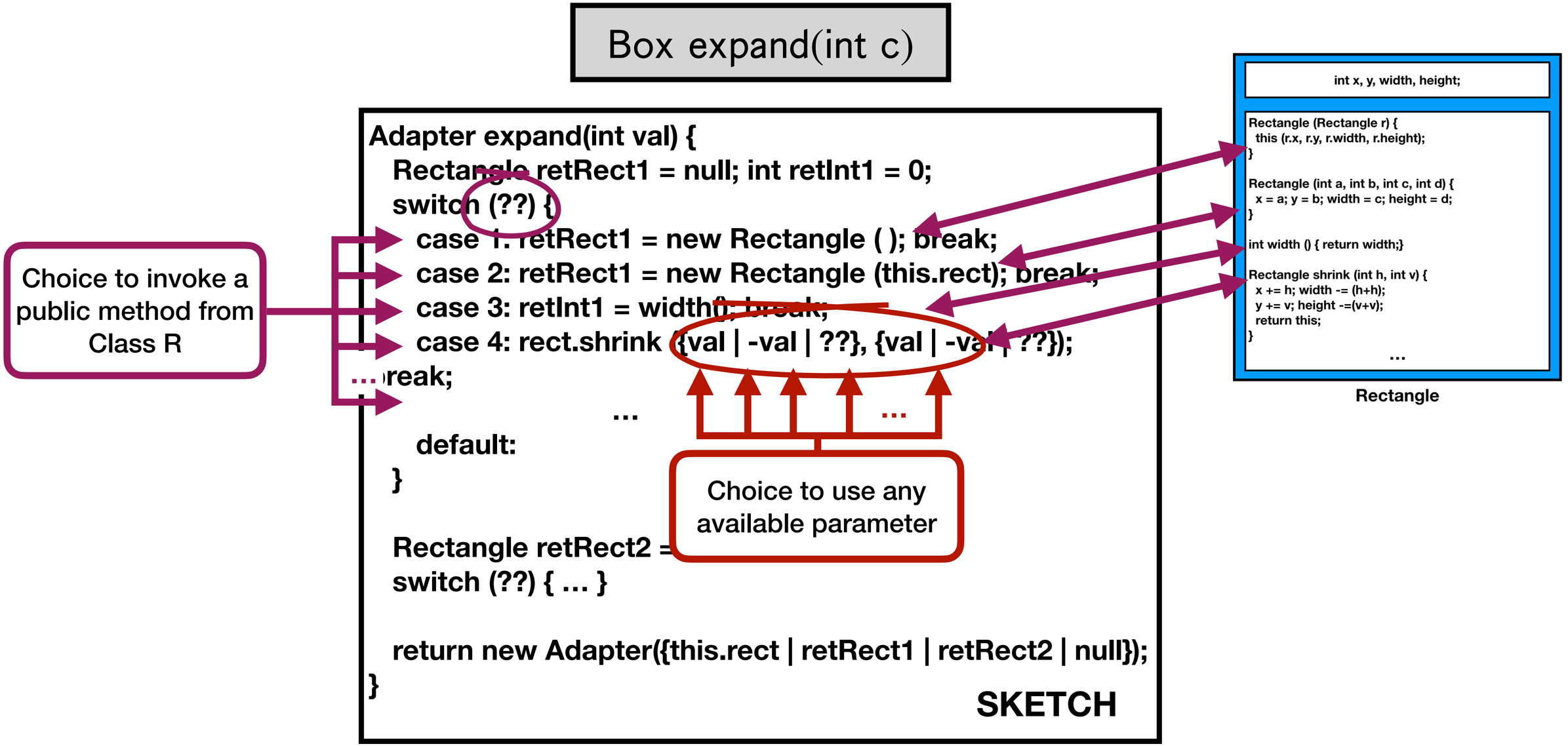
```
Adapter expand(int val) {
  Rectangle retRect1 = null; int retInt1 = 0;
  switch (??) {
    case 1: retRect1 = new Rectangle ( ); break;
    case 2: retRect1 = new Rectangle (this.rect); break;
    case 3: retInt1 = width(); break;
    case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
... break;
                    ...
    default:
  }

  Rectangle retRect2 = null; int retInt2 = 0;
  switch (??) { ... }

  return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

SKETCH

```
int x, y, width, height;

Rectangle (Rectangle r) {
  this (r.x, r.y, r.width, r.height);
}

Rectangle (int a, int b, int c, int d) {
  x = a; y = b; width = c; height = d;
}

int width () { return width;}

Rectangle shrink (int h, int v) {
  x += h; width -= (h+h);
  y += v; height -=(v+v);
  return this;
}
                    ...
```

Rectangle

Building the adapter method sketch

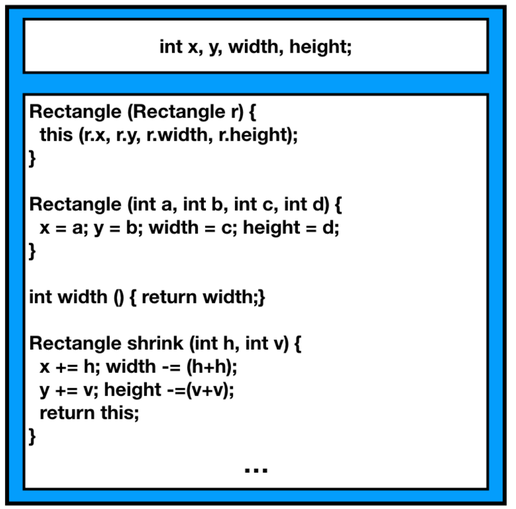


Building the adapter method sketch

Box expand(int c)

```
Adapter expand(int val) {
  Rectangle retRect1 = null; int retInt1 = 0;
  switch (??) {
    case 1: retRect1 = new Rectangle ( ); break;
    case 2: retRect1 = new Rectangle (this.rect); break;
    case 3: retInt1 = width(); break;
    case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
break;
                          …
    default:
  }

  Rectangle retRect2 = null; int retInt2 = 0;
  switch (??) { … }

  return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

SKETCH



Rectangle

Building the adapter method sketch

Box expand(int c)

```
Adapter expand(int val) {
  Rectangle retRect1 = null; int retInt1 = 0;
  switch (??) {
    case 1: retRect1 = new Rectangle ( ); break;
    case 2: retRect1 = new Rectangle (this.rect); break;
    case 3: retInt1 = width(); break;
    case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
break;
                          …
    default:
  }

  Rectangle retRect2 = null; int retInt2 = 0;
  switch (??) { … }

  return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

SKETCH

Create a invocation sequence

```
int x, y, width, height;

Rectangle (Rectangle r) {
  this (r.x, r.y, r.width, r.height);
}

Rectangle (int a, int b, int c, int d) {
  x = a; y = b; width = c; height = d;
}

int width () { return width;}

Rectangle shrink (int h, int v) {
  x += h; width -= (h+h);
  y += v; height -=(v+v);
  return this;
}
                          …
```

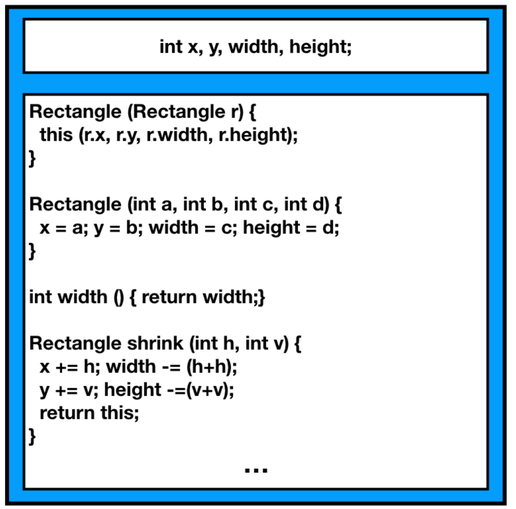
Rectangle

Building the adapter method sketch

Box expand(int c)

```
Adapter expand(int val) {
  Rectangle retRect1 = null; int retInt1 = 0;
  switch (??) {
    case 1: retRect1 = new Rectangle ( ); break;
    case 2: retRect1 = new Rectangle (this.rect); break;
    case 3: retInt1 = width(); break;
    case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
break;
                          …
    default:
  }

  Rectangle retRect2 = null; int retInt2 = 0;
  switch (??) { … }

  return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

SKETCH



Rectangle

Building the adapter method sketch

Box expand(int c)

```
Adapter expand(int val) {
   Rectangle retRect1 = null; int retInt1 = 0;
   switch (??) {
      case 1: retRect1 = new Rectangle ( ); break;
      case 2: retRect1 = new Rectangle (this.rect); break;
      case 3: retInt1 = width(); break;
      case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
break;
                              ...
      default:
   }

   Rectangle retRect2 = null; int retInt2 = 0;
   switch (??) { ... }

   return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

SKETCH

Choice to return any suitable value

```
int x, y, width, height;

Rectangle (Rectangle r) {
  this (r.x, r.y, r.width, r.height);
}

Rectangle (int a, int b, int c, int d) {
  x = a; y = b; width = c; height = d;
}

int width () { return width;}

Rectangle shrink (int h, int v) {
  x += h; width -= (h+h);
  y += v; height -=(v+v);
  return this;
}
                              ...
```

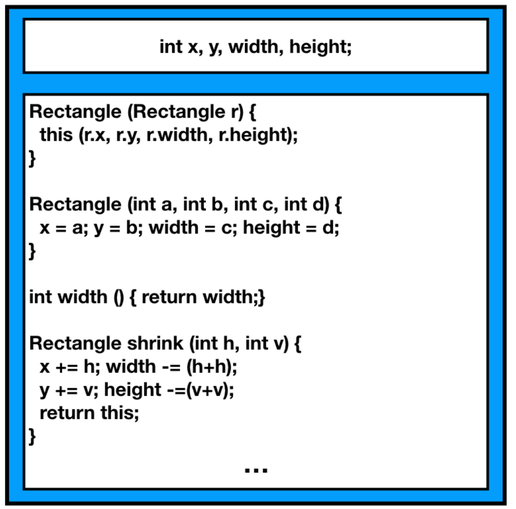
Rectangle

Building the adapter method sketch

Box expand(int c)

```
Adapter expand(int val) {
  Rectangle retRect1 = null; int retInt1 = 0;
  switch (??) {
    case 1: retRect1 = new Rectangle ( ); break;
    case 2: retRect1 = new Rectangle (this.rect); break;
    case 3: retInt1 = width(); break;
    case 4: rect.shrink ({val | -val | ??}, {val | -val | ??});
break;
                          …
    default:
  }

  Rectangle retRect2 = null; int retInt2 = 0;
  switch (??) { … }

  return new Adapter({this.rect | retRect1 | retRect2 | null});
}
```

SKETCH



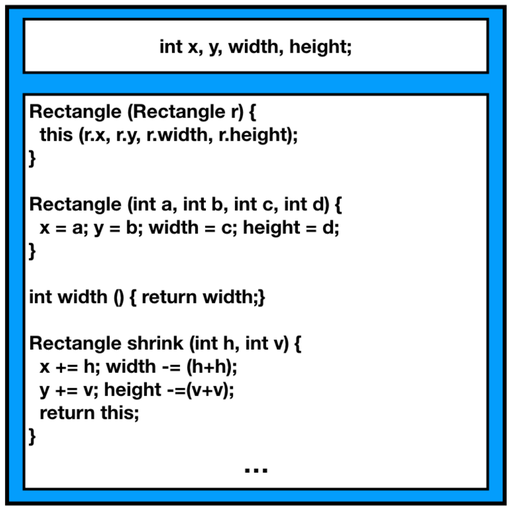
Rectangle

Building the correctness check/harness

ASSUME AVAILABILITY OF INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + \text{width}$$
$$y_2 = y + \text{height}$$

```
harness static void check-expand ( int s[], int r[] ) {
  Box a1 = new Box (); a1.x1 = s[0]; a1.y1 = s[1];
  a1.x2 = s[2]; a1.y2 = s[3];
  int a2 = s[4];

  Rectangle b1 = new Rectangle(); b1.x = r[0]; b1.y = r[1];
  b1.width = r[2]; b1.height = r[3];
  Adapter w1 = new Adapter(b1);
  int b2 = r[4];

  assume(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assume(a2 == b2);

  Box2 a3 = a1.expand(a2);  Adapter w3 = w1.expand(b2);

  Rectangle b3 = w3.rect;
  assert(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assert(a2 == b2);
  assert(a3.x1 == b3.x && a3.y1 = b3.y
  && a3.x2 == b3.x+ b3.width && a3.y2 == b3.y+ b3.height);
  assert((a1 == a3 && b1 == b3) || (a1 != a3 && b1 != b3) );
}
```



Rectangle

Building the correctness check/harness

ASSUME AVAILABILITY OF INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + \text{width}$$
$$y_2 = y + \text{height}$$

```
harness static void check-expand ( int s[], int r[] ) {
  Box a1 = new Box (); a1.x1 = s[0]; a1.y1 = s[1];
  a1.x2 = s[2]; a1.y2 = s[3];
  int a2 = s[4];

  Rectangle b1 = new Rectangle(); b1.x = r[0]; b1.y = r[1];
  b1.width = r[2]; b1.height = r[3];
  Adapter w1 = new Adapter(b1);
  int b2 = r[4];

  assume(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assume(a2 == b2);

  Box2 a3 = a1.expand(a2);  Adapter w3 = w1.expand(b2);

  Rectangle b3 = w3.rect;
  assert(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assert(a2 == b2);
  assert(a3.x1 == b3.x && a3.y1 = b3.y
  && a3.x2 == b3.x+ b3.width && a3.y2 == b3.y+ b3.height);
  assert((a1 == a3 && b1 == b3) || (a1 != a3 && b1 != b3) );
}
```

Create parameters for the original method and symbolize them.

```
Rectangle (int a, int b, int c, int d) {
  x = a; y = b; width = c; height = d;
}

int width () { return width;}

Rectangle shrink (int h, int v) {
  x += h; width -= (h+h);
  y += v; height -=(v+v);
  return this;
}
                ...
```

Rectangle

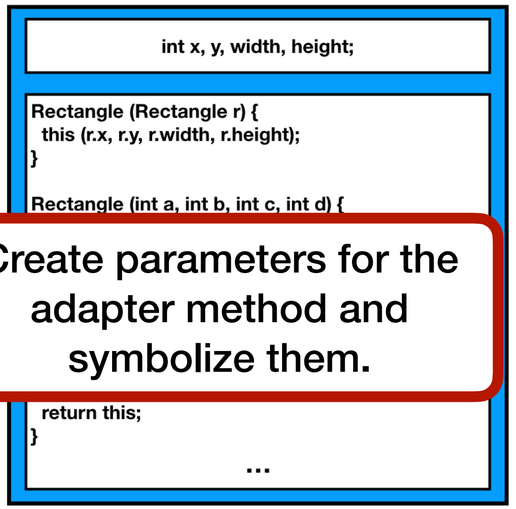
Building the correctness check/harness

ASSUME AVAILABILITY OF INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + \text{width}$$
$$y_2 = y + \text{height}$$

```
harness static void check-expand ( int s[], int r[] ) {
  Box a1 = new Box (); a1.x1 = s[0]; a1.y1 = s[1];
  a1.x2 = s[2]; a1.y2 = s[3];
  int a2 = s[4];

  Rectangle b1 = new Rectangle(); b1.x = r[0]; b1.y = r[1];
  b1.width = r[2]; b1.height = r[3];
  Adapter w1 = new Adapter(b1);
  int b2 = r[4];

  assume(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assume(a2 == b2);

  Box2 a3 = a1.expand(a2);  Adapter w3 = w1.expand(b2);

  Rectangle b3 = w3.rect;
  assert(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assert(a2 == b2);
  assert(a3.x1 == b3.x && a3.y1 = b3.y
  && a3.x2 == b3.x+ b3.width && a3.y2 == b3.y+ b3.height);
  assert((a1 == a3 && b1 == b3) || (a1 != a3 && b1 != b3) );
}
```

Create parameters for the adapter method and symbolize them.



Rectangle

Building the correctness check/harness

ASSUME AVAILABILITY OF INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + \text{width}$$
$$y_2 = y + \text{height}$$

```
harness static void check-expand ( int s[], int r[] ) {
  Box a1 = new Box (); a1.x1 = s[0]; a1.y1 = s[1];
  a1.x2 = s[2]; a1.y2 = s[3];
  int a2 = s[4];

  Rectangle b1 = new Rectangle(); b1.x = r[0]; b1.y = r[1];
  b1.width = r[2]; b1.height = r[3];
  Adapter w1 = new Adapter(b1);
  int b2 = r[4];

  assume(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assume(a2 == b2);

  Box2 a3 = a1.expand(a2);  Adapter w3 = w1.expand(b2);

  Rectangle b3 = w3.rect;
  assert(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assert(a2 == b2);
  assert(a3.x1 == b3.x && a3.y1 = b3.y
  && a3.x2 == b3.x+ b3.width && a3.y2 == b3.y+ b3.height);
  assert((a1 == a3 && b1 == b3) || (a1 != a3 && b1 != b3) );
}
```

```
int x, y, width, height;

Rectangle (Rectangle r) {
  this (r.x, r.y, r.width, r.height);
}

Rectangle (int a, int b, int c, int d) {
  x = a; y = b; width = c; height = d;
}

int width () { return width;}

Rectangle shrink (int h, int v) {
  x += h; width -= (h+h);
  y += v; height -=(v+v);
  return this;
```

Enforce the equivalence between corresponding parameters

Building the correctness check/harness

ASSUME AVAILABILITY OF INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + \text{width}$$
$$y_2 = y + \text{height}$$

```
harness static void check-expand ( int s[], int r[] ) {
  Box a1 = new Box (); a1.x1 = s[0]; a1.y1 = s[1];
  a1.x2 = s[2]; a1.y2 = s[3];
  int a2 = s[4];

  Rectangle b1 = new Rectangle(); b1.x = r[0]; b1.y = r[1];
  b1.width = r[2]; b1.height = r[3];
  Adapter w1 = new Adapter(b1);
  int b2 = r[4];

  assume(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assume(a2 == b2);

  Box2 a3 = a1.expand(a2);  Adapter w3 = w1.expand(b2);

  Rectangle b3 = w3.rect;
  assert(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assert(a2 == b2);
  assert(a3.x1 == b3.x && a3.y1 = b3.y
  && a3.x2 == b3.x+ b3.width && a3.y2 == b3.y+ b3.height);
  assert((a1 == a3 && b1 == b3) || (a1 != a3 && b1 != b3) );
}
```

```
int x, y, width, height;

Rectangle (Rectangle r) {
  this (r.x, r.y, r.width, r.height);
}

Rectangle (int a, int b, int c, int d) {
  x = a; y = b; width = c; height = d;
}

int width () { return width;}

Rectangle shrink (int h, int v) {
  x += h; width -= (h+h);
  y += v; height -=(v+v);
  return this;
```

Enforce the equivalence between corresponding parameters

Building the correctness check/harness

ASSUME AVAILABILITY OF INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + \text{width}$$
$$y_2 = y + \text{height}$$

```
harness static void check-expand ( int s[], int r[] ) {
  Box a1 = new Box (); a1.x1 = s[0]; a1.y1 = s[1];
  a1.x2 = s[2]; a1.y2 = s[3];
  int a2 = s[4];

  Rectangle b1 = new Rectangle(); b1.x = r[0]; b1.y = r[1];
  b1.width = r[2]; b1.height = r[3];
  Adapter w1 = new Adapter(b1);
  int b2 = r[4];

  assume(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assume(a2 == b2);

  Box2 a3 = a1.expand(a2);  Adapter w3 = w1.expand(b2);

  Rectangle b3 = w3.rect;
  assert(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assert(a2 == b2);
  assert(a3.x1 == b3.x && a3.y1 = b3.y
  && a3.x2 == b3.x+ b3.width && a3.y2 == b3.y+ b3.height);
  assert((a1 == a3 && b1 == b3) || (a1 != a3 && b1 != b3) );
}
```

```
int x, y, width, height;

Rectangle (Rectangle r) {
  this (r.x, r.y, r.width, r.height);
}

Rectangle (int a, int b, int c, int d) {
  x = a; y = b; width = c; height = d;
}

int width () { return width;}

Rectangle shrink (int h, int v) {
  x += h; width -= (h+h);
  y += v; height -=(v+v);
  return this;
```

Enforce the equivalence between corresponding parameters

Building the correctness check/harness

ASSUME AVAILABILITY OF INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + \text{width}$$
$$y_2 = y + \text{height}$$

```
harness static void check-expand ( int s[], int r[] ) {
  Box a1 = new Box (); a1.x1 = s[0]; a1.y1 = s[1];
  a1.x2 = s[2]; a1.y2 = s[3];
  int a2 = s[4];

  Rectangle b1 = new Rectangle(); b1.x = r[0]; b1.y = r[1];
  b1.width = r[2]; b1.height = r[3];
  Adapter w1 = new Adapter(b1);
  int b2 = r[4];

  assume(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assume(a2 == b2);

  Box2 a3 = a1.expand(a2);  Adapter w3 = w1.expand(b2);

  Rectangle b3 = w3.rect;
  assert(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assert(a2 == b2);
  assert(a3.x1 == b3.x && a3.y1 = b3.y
  && a3.x2 == b3.x+ b3.width && a3.y2 == b3.y+ b3.height);
  assert((a1 == a3 && b1 == b3) || (a1 != a3 && b1 != b3) );
}
```

```
int x, y, width, height;

Rectangle (Rectangle r) {
  this (r.x, r.y, r.width, r.height);
}

Rectangle (int a, int b, int c, int d) {
  x = a; y = b; width = c; height = d;
}

int width () { return width;}

Rectangle shrink (int h, int v) {
  x += h; width -= (h+h);
  y += v; height -=(v+v);
  return this;
```

Enforce the equivalence between corresponding parameters

Building the correctness check/harness

ASSUME AVAILABILITY OF INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + \text{width}$$
$$y_2 = y + \text{height}$$

```
harness static void check-expand ( int s[], int r[] ) {
  Box a1 = new Box (); a1.x1 = s[0]; a1.y1 = s[1];
  a1.x2 = s[2]; a1.y2 = s[3];
  int a2 = s[4];

  Rectangle b1 = new Rectangle(); b1.x = r[0]; b1.y = r[1];
  b1.width = r[2]; b1.height = r[3];
  Adapter w1 = new Adapter(b1);
  int b2 = r[4];

  assume(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assume(a2 == b2);

  Box2 a3 = a1.expand(a2);  Adapter w3 = w1.expand(b2);

  Rectangle b3 = w3.rect;
  assert(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assert(a2 == b2);
  assert(a3.x1 == b3.x && a3.y1 = b3.y
  && a3.x2 == b3.x+ b3.width && a3.y2 == b3.y+ b3.height);
  assert((a1 == a3 && b1 == b3) || (a1 != a3 && b1 != b3) );
}
```

```
int x, y, width, height;

Rectangle (Rectangle r) {
  this (r.x, r.y, r.width, r.height);
}

Rectangle (int a, int b, int c, int d) {
  x = a; y = b; width = c; height = d;
}

int width () { return width;}

Rectangle shrink (int h, int v) {
  x += h; width -= (h+h);
  y += v; height -=(v+v);
  return this;
```

Enforce the equivalence between corresponding parameters

Building the correctness check/harness

ASSUME AVAILABILITY OF INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + \text{width}$$
$$y_2 = y + \text{height}$$

```
harness static void check-expand ( int s[], int r[] ) {
  Box a1 = new Box (); a1.x1 = s[0]; a1.y1 = s[1];
  a1.x2 = s[2]; a1.y2 = s[3];
  int a2 = s[4];

  Rectangle b1 = new Rectangle(); b1.x = r[0]; b1.y = r[1];
  b1.width = r[2]; b1.height = r[3];
  Adapter w1 = new Adapter(b1);
  int b2 = r[4];

  assume(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assume(a2 == b2);

  Box2 a3 = a1.expand(a2);  Adapter w3 = w1.expand(b2);

  Rectangle b3 = w3.rect;
  assert(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assert(a2 == b2);
  assert(a3.x1 == b3.x && a3.y1 = b3.y
  && a3.x2 == b3.x+ b3.width && a3.y2 == b3.y+ b3.height);
  assert((a1 == a3 && b1 == b3) || (a1 != a3 && b1 != b3) );
}
```

Execute original method and the adapter method

Building the correctness check/harness

ASSUME AVAILABILITY OF INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + \text{width}$$
$$y_2 = y + \text{height}$$

```
harness static void check-expand ( int s[], int r[] ) {
  Box a1 = new Box (); a1.x1 = s[0]; a1.y1 = s[1];
  a1.x2 = s[2]; a1.y2 = s[3];
  int a2 = s[4];

  Rectangle b1 = new Rectangle(); b1.x = r[0]; b1.y = r[1];
  b1.width = r[2]; b1.height = r[3];
  Adapter w1 = new Adapter(b1);
  int b2 = r[4];

  assume(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assume(a2 == b2);

  Box2 a3 = a1.expand(a2);  Adapter w3 = w1.expand(b2);

  Rectangle b3 = w3.rect;
  assert(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assert(a2 == b2);
  assert(a3.x1 == b3.x && a3.y1 = b3.y
  && a3.x2 == b3.x+ b3.width && a3.y2 == b3.y+ b3.height);
  assert((a1 == a3 && b1 == b3) || (a1 != a3 && b1 != b3) );
}
```

Assert Equivalence between corresponding values

Building the correctness check/harness

ASSUME AVAILABILITY OF INTER-CLASS EQUIVALENCE PREDICATE

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + \text{width}$$
$$y_2 = y + \text{height}$$

```
harness static void check-expand ( int s[], int r[] ) {
  Box a1 = new Box (); a1.x1 = s[0]; a1.y1 = s[1];
  a1.x2 = s[2]; a1.y2 = s[3];
  int a2 = s[4];

  Rectangle b1 = new Rectangle(); b1.x = r[0]; b1.y = r[1];
```

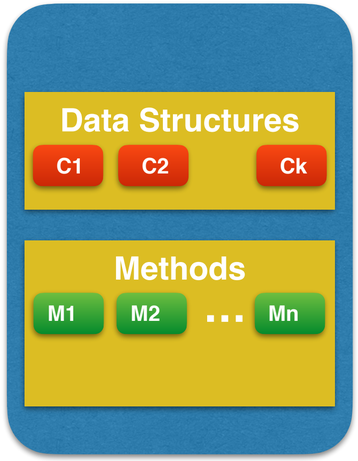
- **Reference class fields**
- **Reference method parameters**
- **Aliasing between variables and other side effects**

```
  Box2 a3 = a1.expand(a2);  Adapter w3 = w1.expand(b2);

  Rectangle b3 = w3.rect;
  assert(a1.x1 == b1.x && a1.y1 = b1.y
  && a1.x2 == b1.x+b1.width && a1.y2 == b1.y+b1.height);
  assert(a2 == b2);
  assert(a3.x1 == b3.x && a3.y1 = b3.y
  && a3.x2 == b3.x+ b3.width && a3.y2 == b3.y+ b3.height);
  assert((a1 == a3 && b1 == b3) || (a1 != a3 && b1 != b3) );
}
```

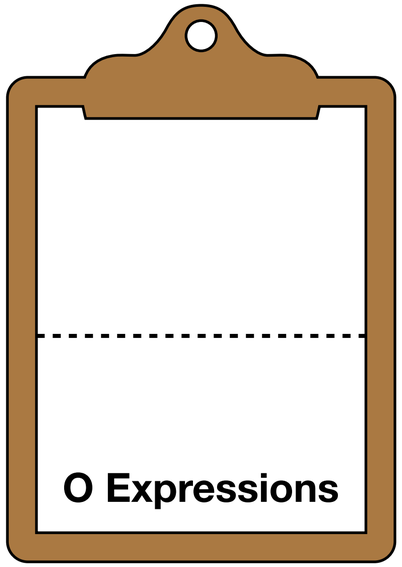
Assert Equivalence between corresponding values

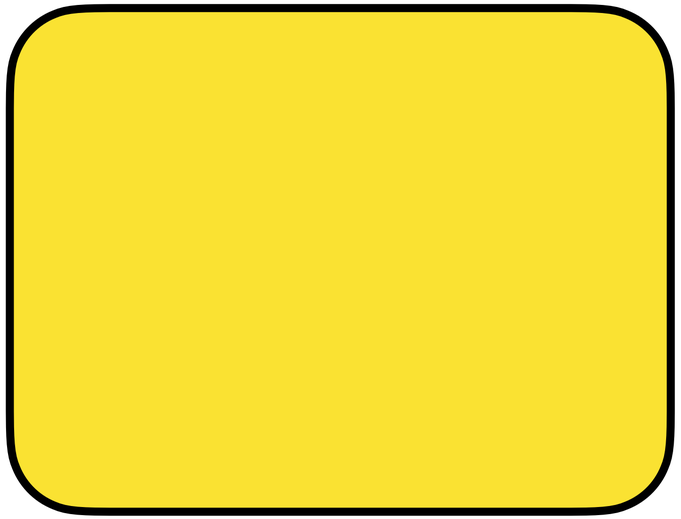
Building Inter-class Equivalence

Step 1:Symbolic Execution



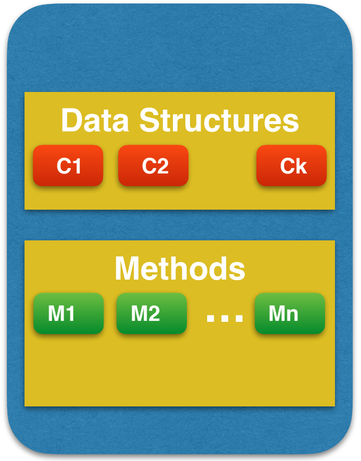
Class O



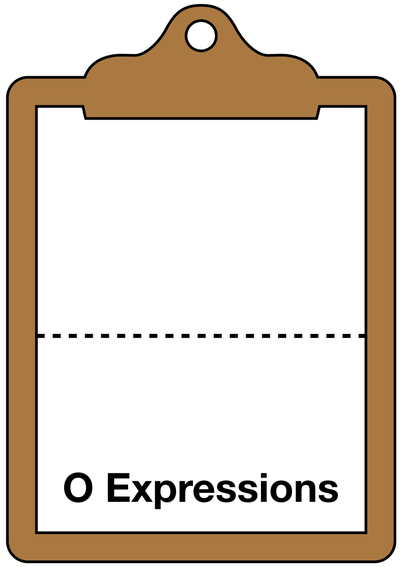


Step 1:Symbolic Execution

- ***All*** public methods in class O are symbolically executed



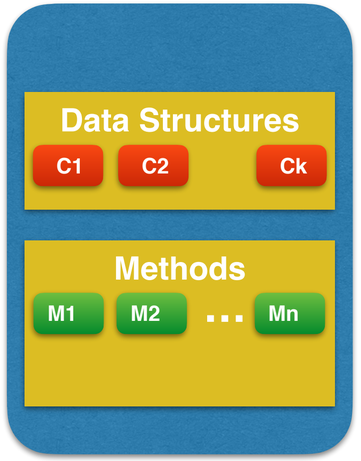
Class O



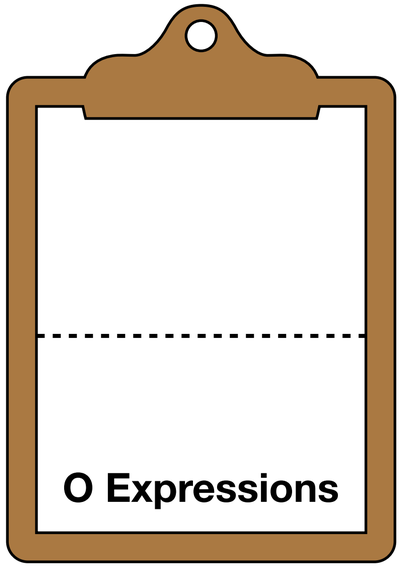
Method : m_i

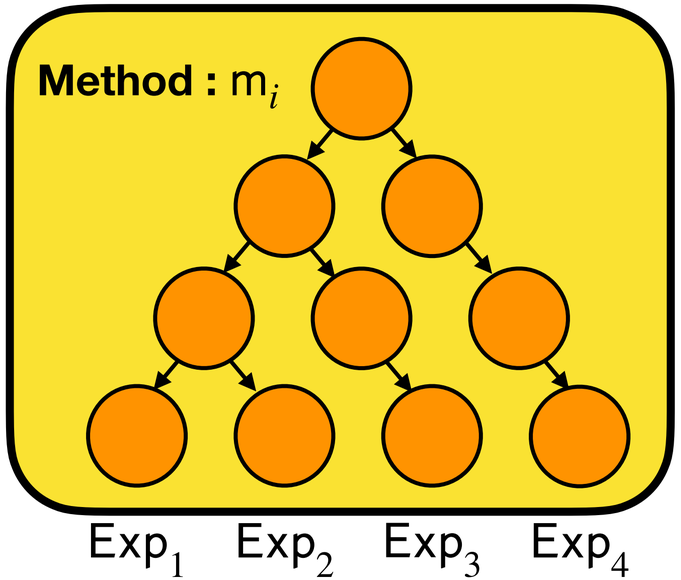
Step 1:Symbolic Execution

- ***All*** public methods in class O are symbolically executed



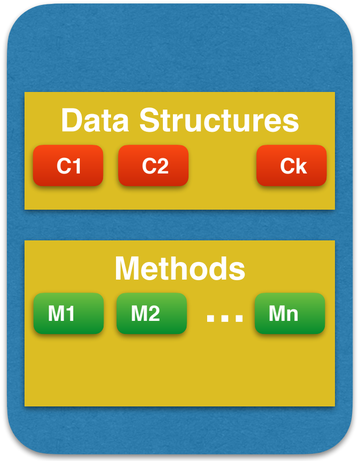
Class O



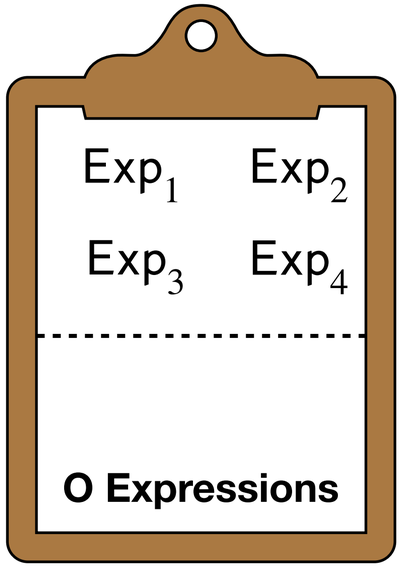


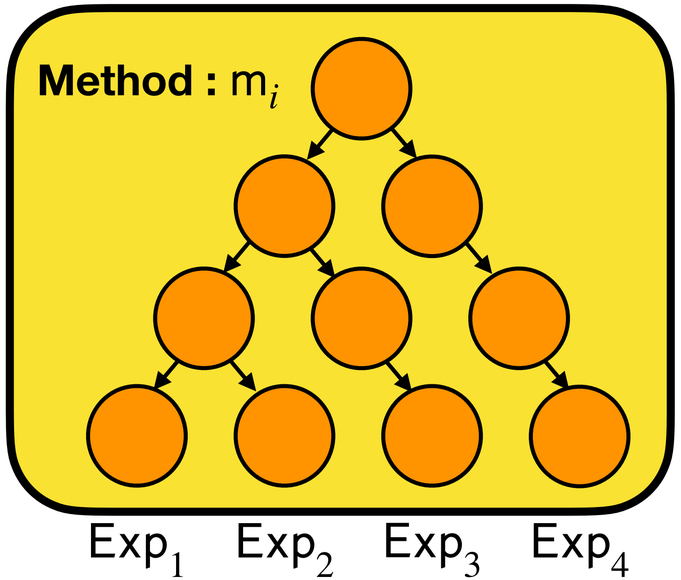
Step 1:Symbolic Execution

- ***All*** public methods in class O are symbolically executed
- Identify all symbolic expressions ***returned***



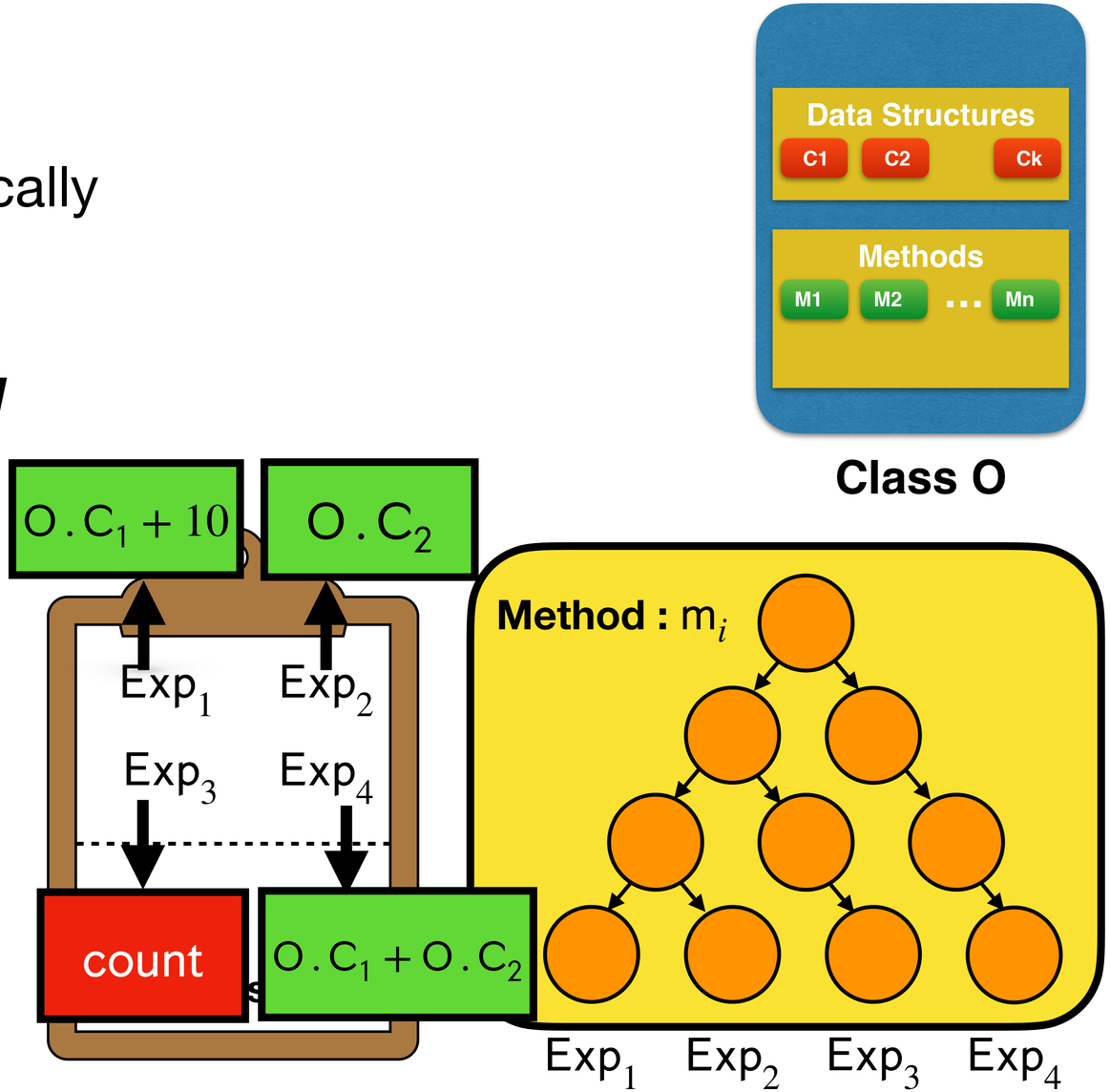
Class O





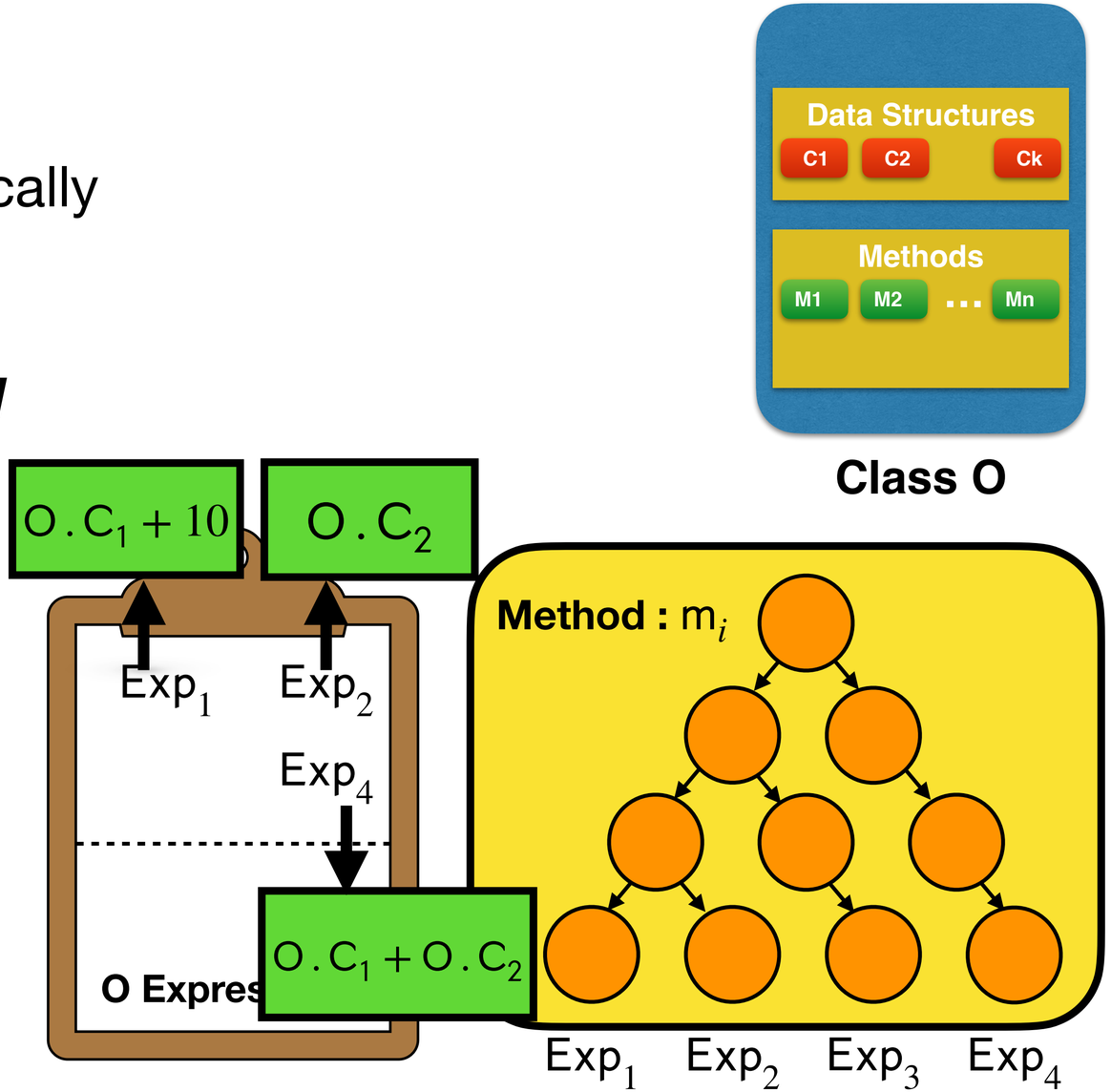
Step 1:Symbolic Execution

- ***All*** public methods in class O are symbolically executed
- Identify all symbolic expressions ***returned***
- Retain useful expressions
  - Contains only O fields and constants



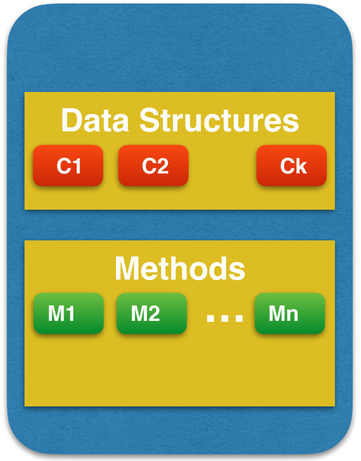
Step 1:Symbolic Execution

- ***All*** public methods in class O are symbolically executed
- Identify all symbolic expressions ***returned***
- Retain useful expressions
  - Contains only O fields and constants

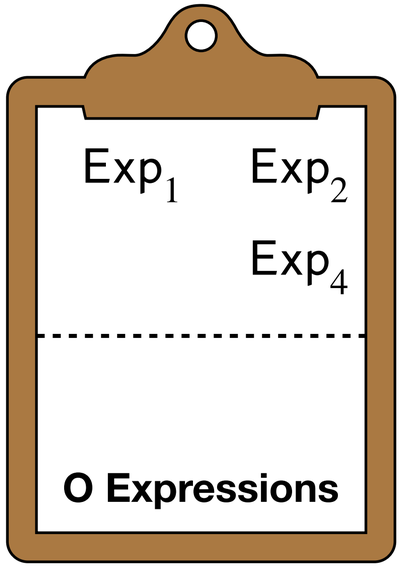


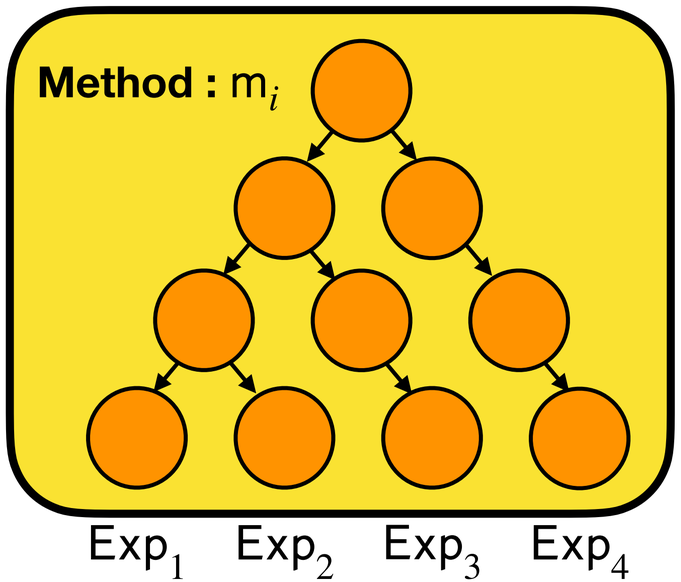
Step 1:Symbolic Execution

- ***All*** public methods in class O are symbolically executed
- Identify all symbolic expressions ***returned***
- Retain useful expressions
  - Contains only O fields and constants



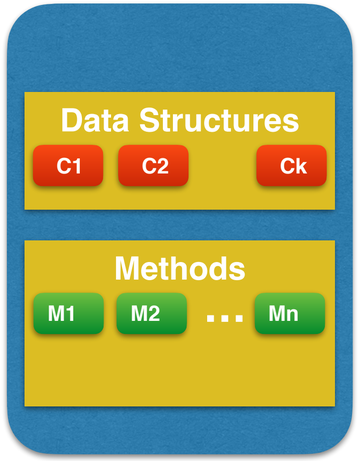
Class O



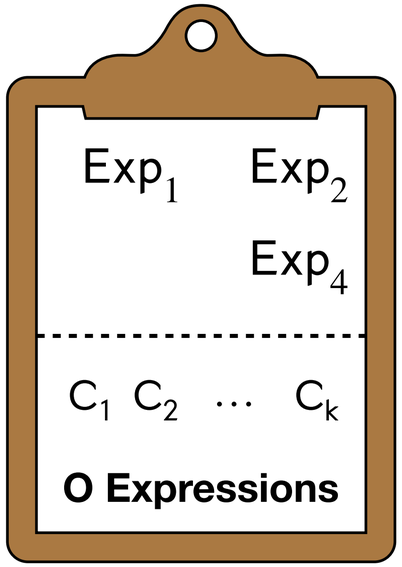


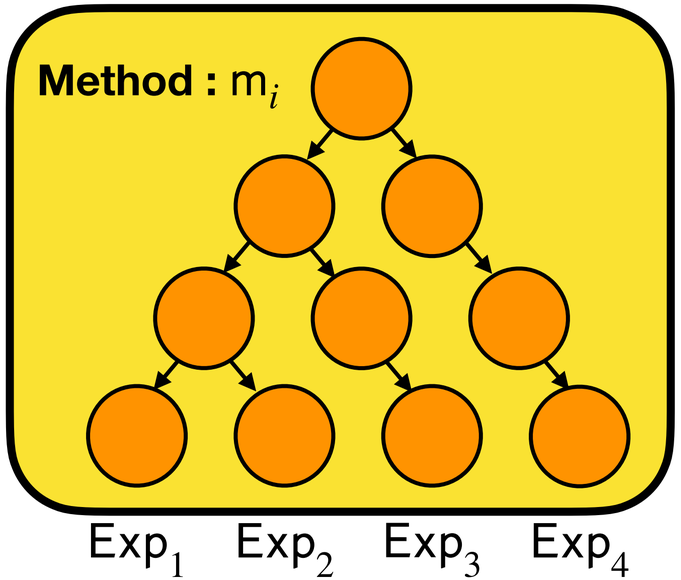
Step 1:Symbolic Execution

- ***All*** public methods in class O are symbolically executed
- Identify all symbolic expressions ***returned***
- Retain useful expressions
  - Contains only O fields and constants
- Identify all field dereferences of class O.



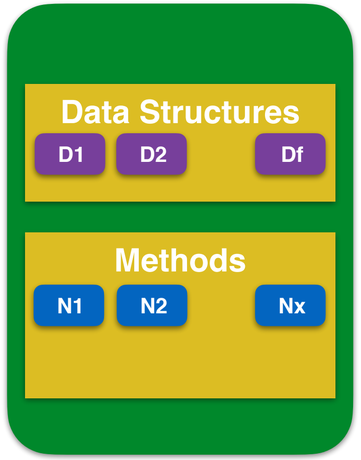
Class O



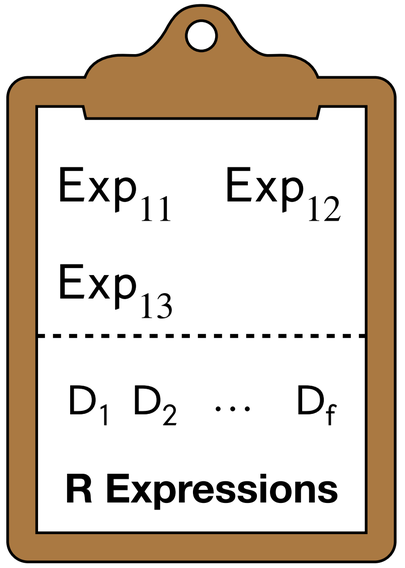


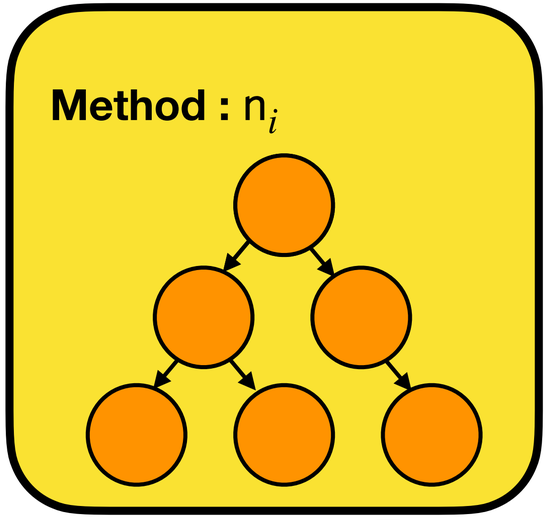
Step 1:Symbolic Execution

- Repeat the same process for class R



Class R

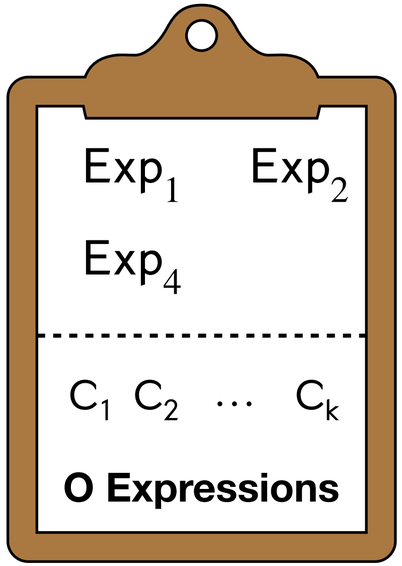


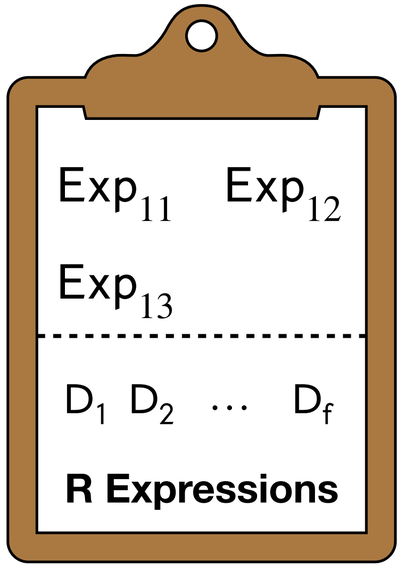


Exp_{11} Exp_{12} Exp_{13}

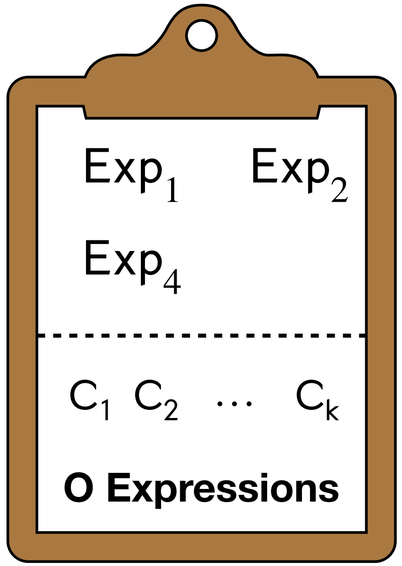
Step 2: Candidate Equivalence Generation

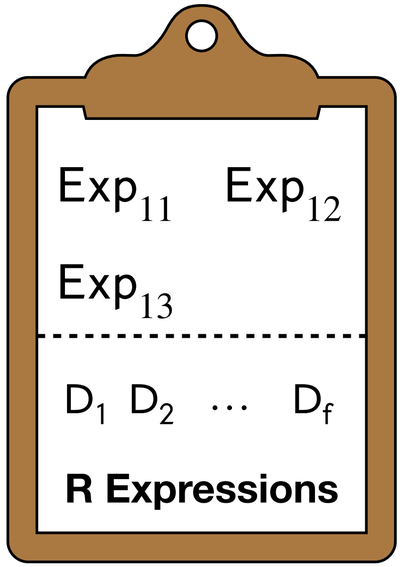
Step 2: Candidate Equivalence Generation





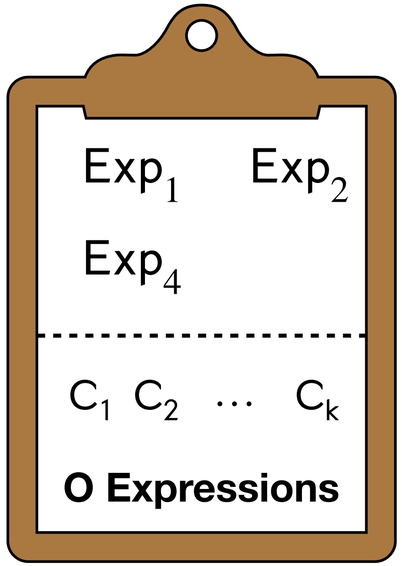
Step 2: Candidate Equivalence Generation

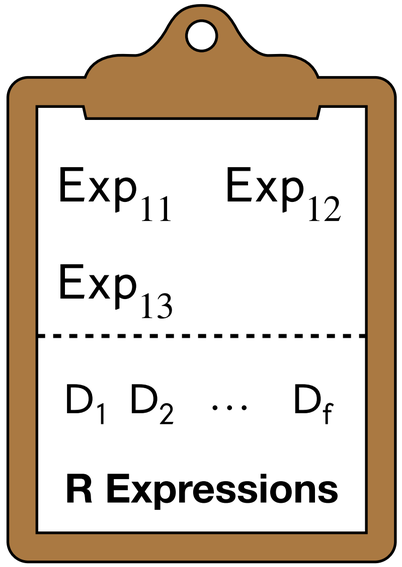




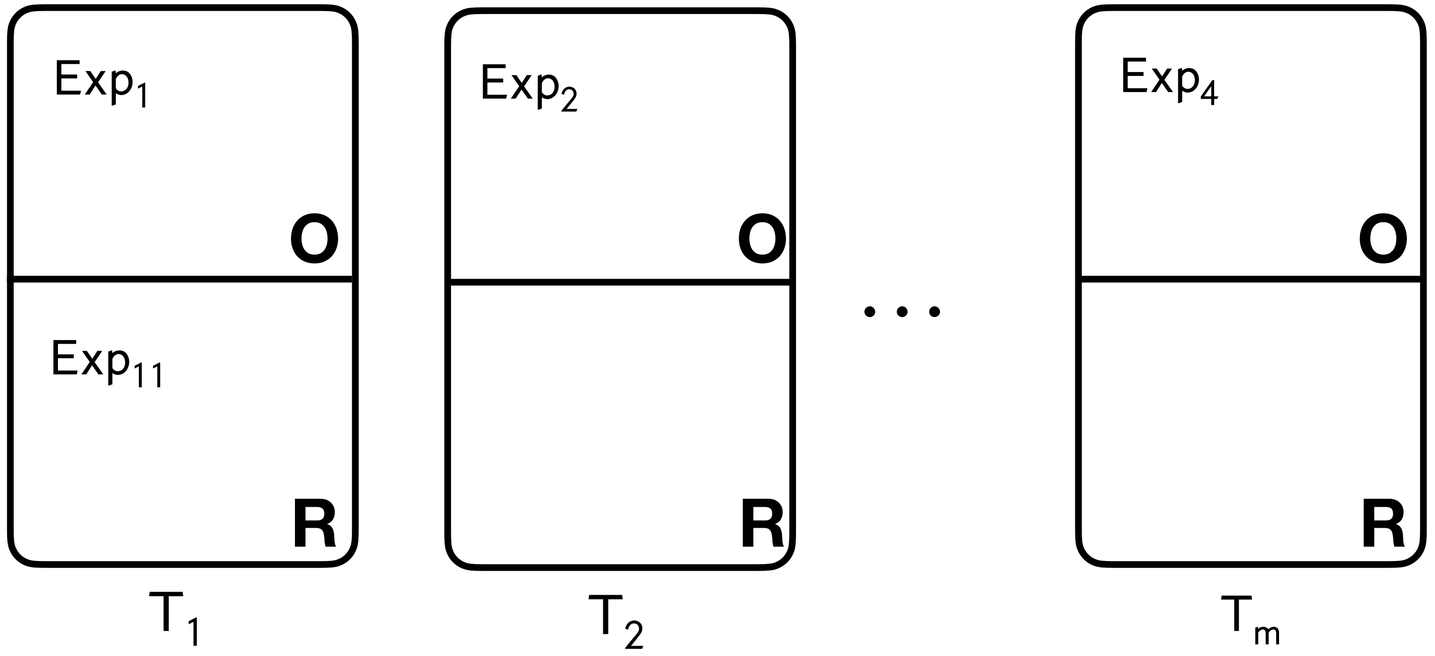
- **Sort expressions based on types**

Step 2: Candidate Equivalence Generation

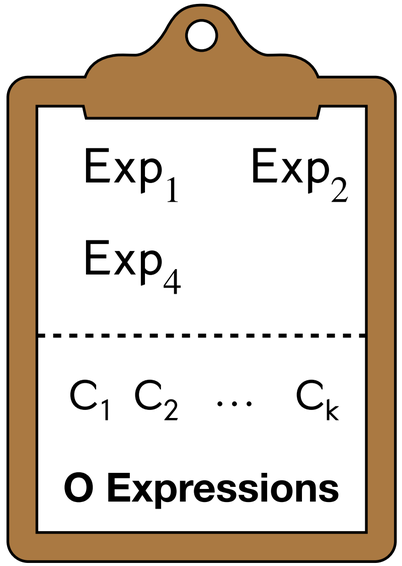


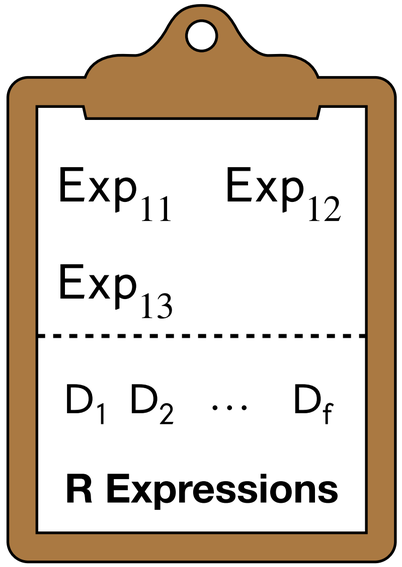


- **Sort expressions based on types**

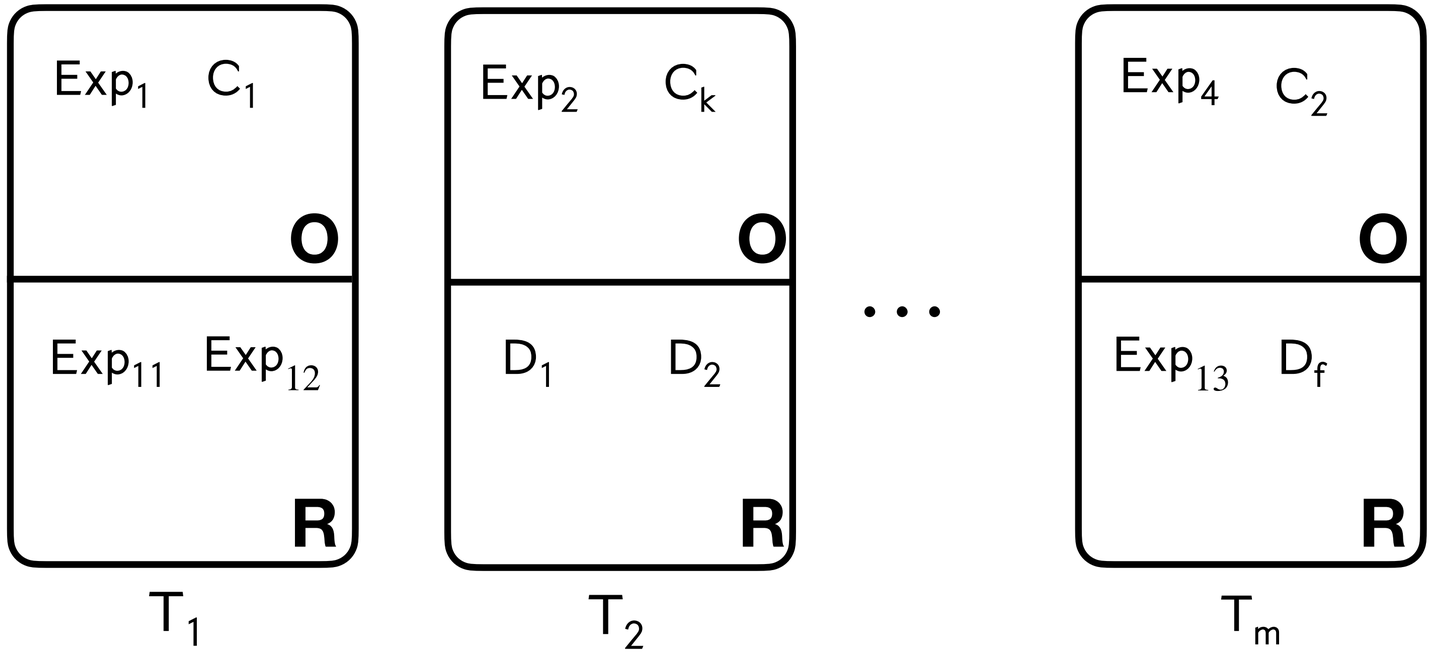


Step 2: Candidate Equivalence Generation

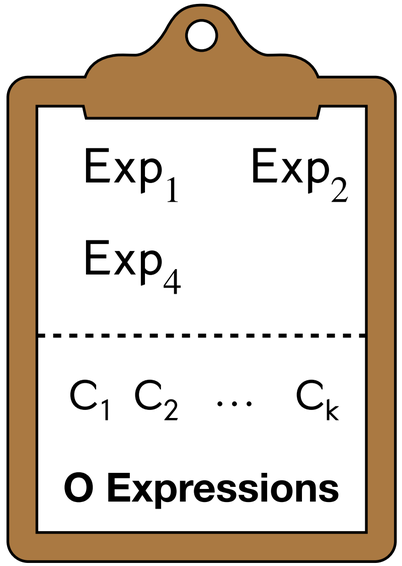


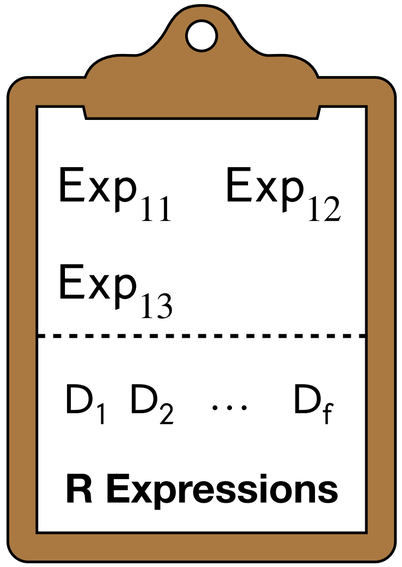


- **Sort expressions based on types**

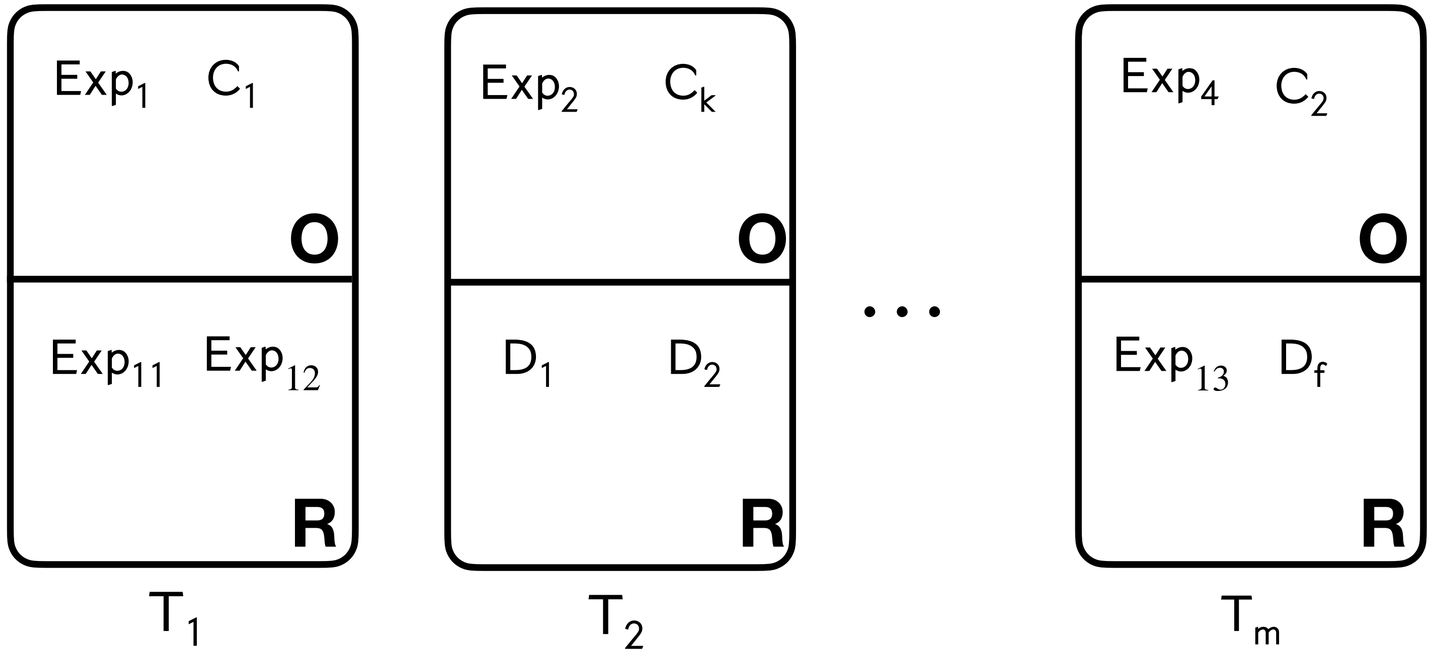


Step 2: Candidate Equivalence Generation

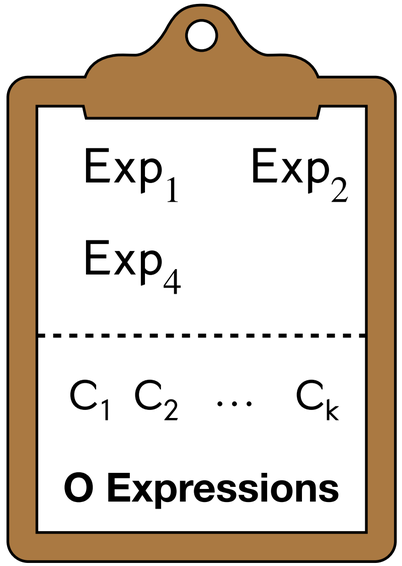


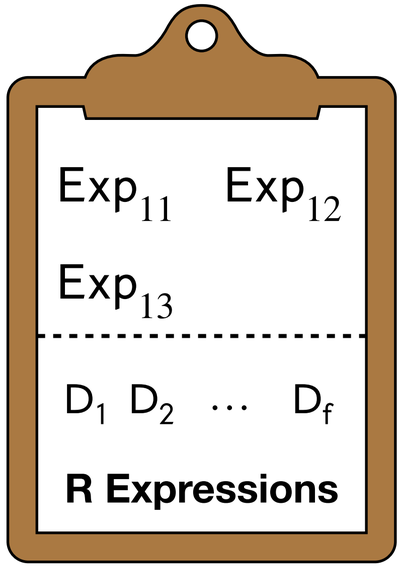


- **Sort expressions based on types**
- **Equate expressions and build predicates**

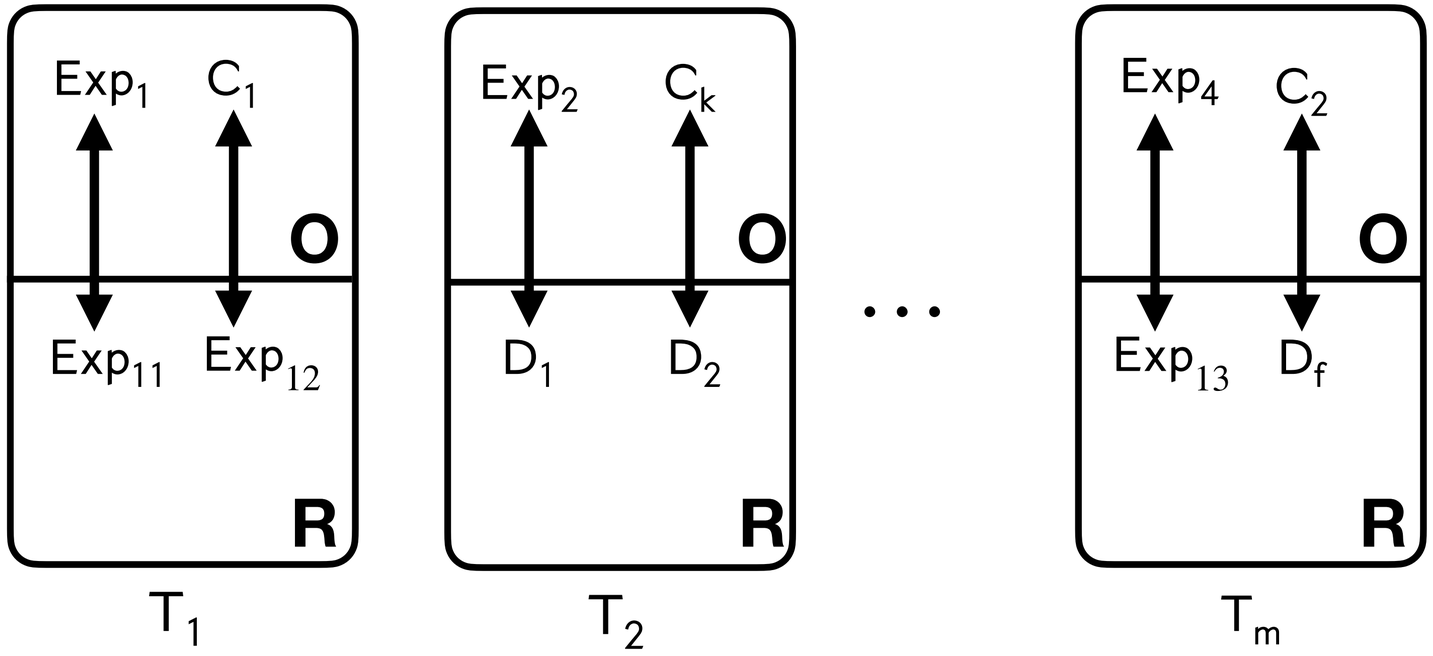


Step 2: Candidate Equivalence Generation

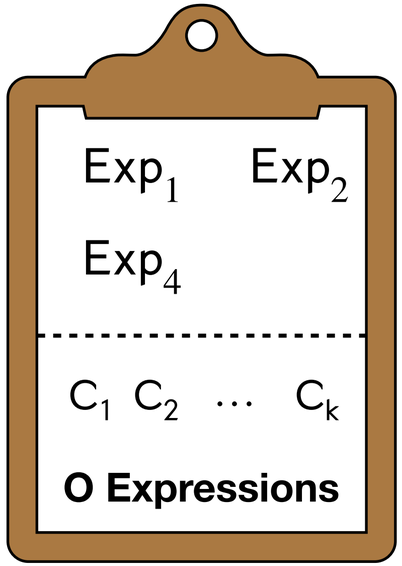


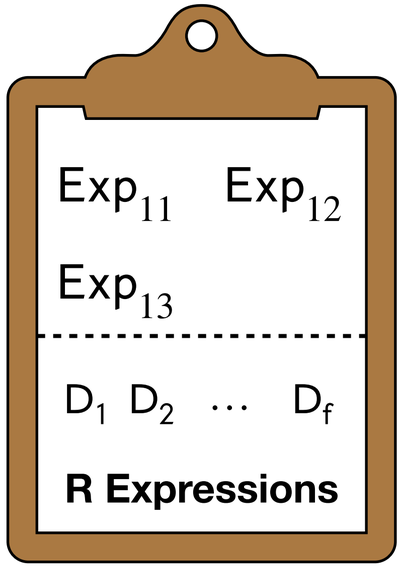


- **Sort expressions based on types**
- **Equate expressions and build predicates**

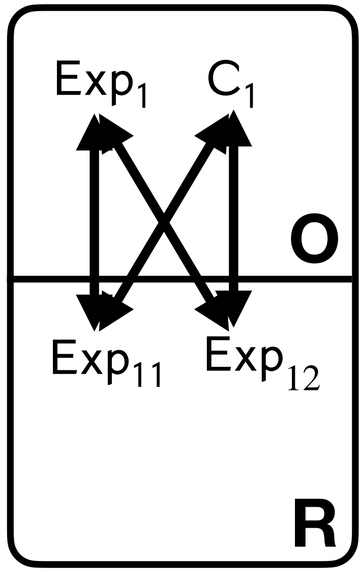


Step 2: Candidate Equivalence Generation

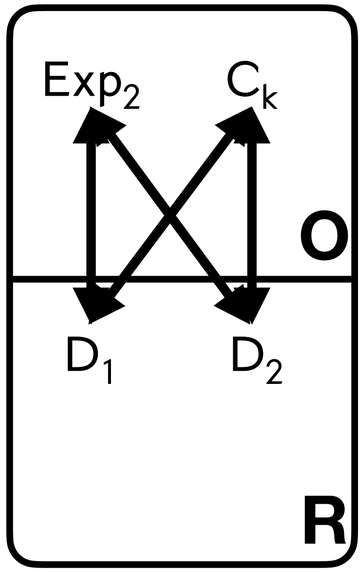




- **Sort expressions based on types**
- **Equate expressions and build predicates**

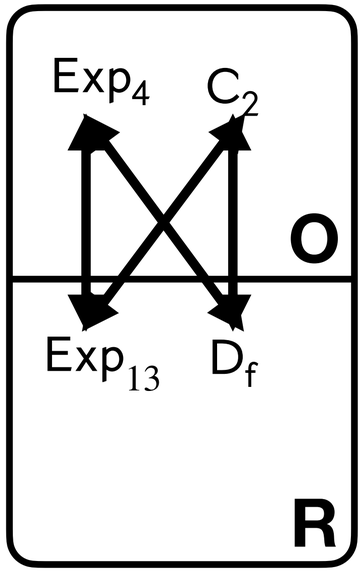


T_1



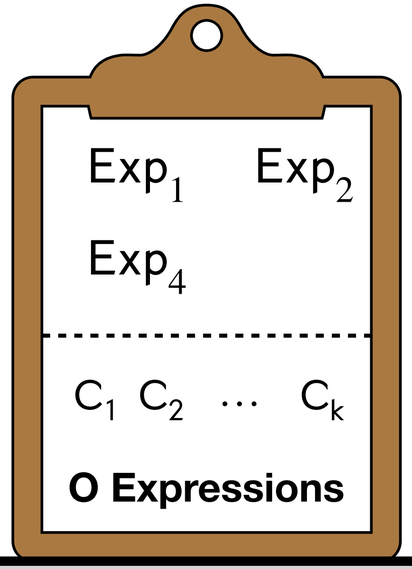
T_2

…



T_m

Step 2: Candidate Equivalence Generation



- **Sort expressions based on types**
- **Equate expressions and build predicates**

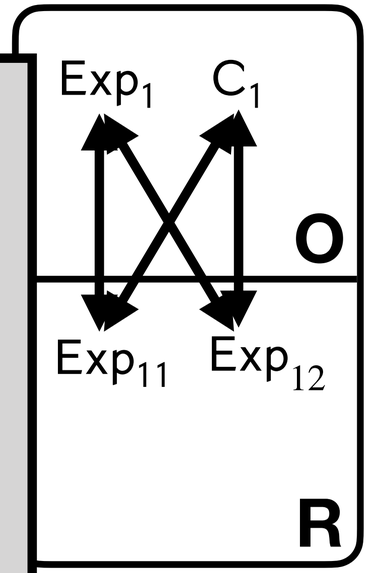
$Exp_1 = Exp_{11} \wedge Exp_2 = D_1 \wedge Exp_4 = Exp_{13}$

$Exp_1 = Exp_{12} \wedge Exp_2 = D_1 \wedge Exp_4 = Exp_{13}$

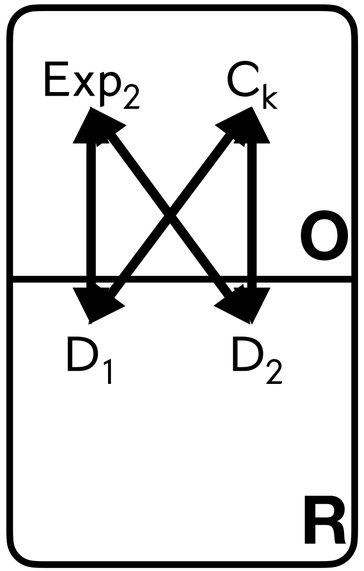
$C_1 = Exp_{11} \wedge Exp_2 = D_1 \wedge Exp_4 = Exp_{13}$

• • •

true

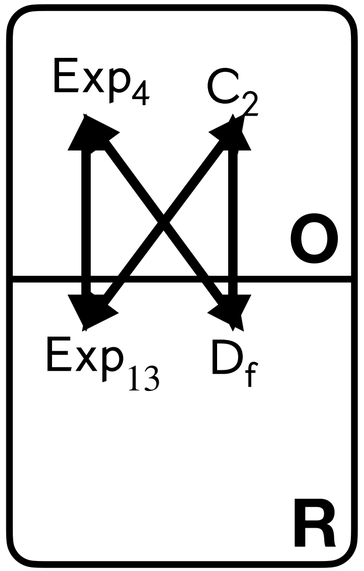


T_1



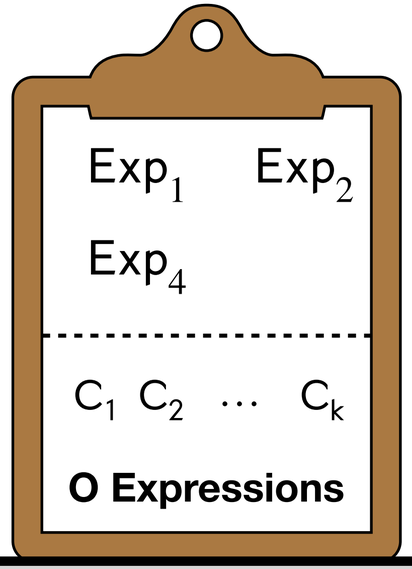
T_2

• • •



T_m

Step 2: Candidate Equivalence Generation



- **Sort expressions based on types**
- **Equate expressions and build predicates**
- **Class O return expressions must be mapped**

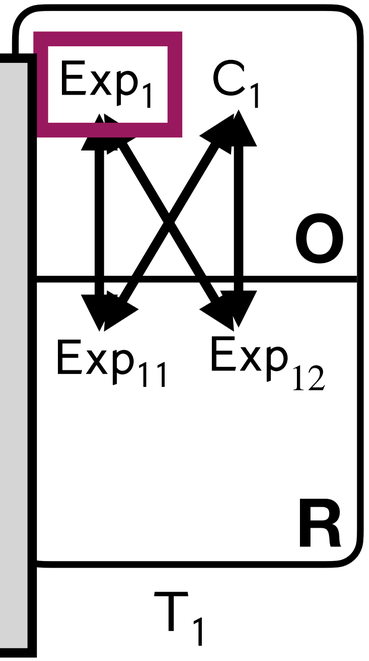
$Exp_1 = Exp_{11} \wedge Exp_2 = D_1 \wedge Exp_4 = Exp_{13}$

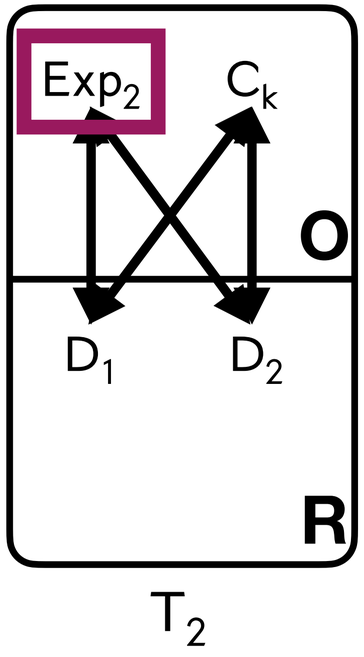
$Exp_1 = Exp_{12} \wedge Exp_2 = D_1 \wedge Exp_4 = Exp_{13}$

$C_1 = Exp_{11} \wedge Exp_2 = D_1 \wedge Exp_4 = Exp_{13}$

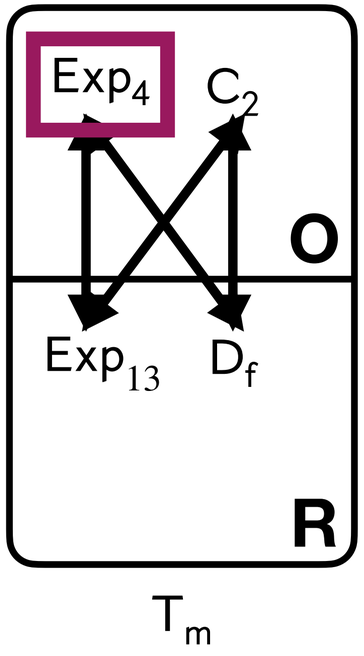
• • •

true

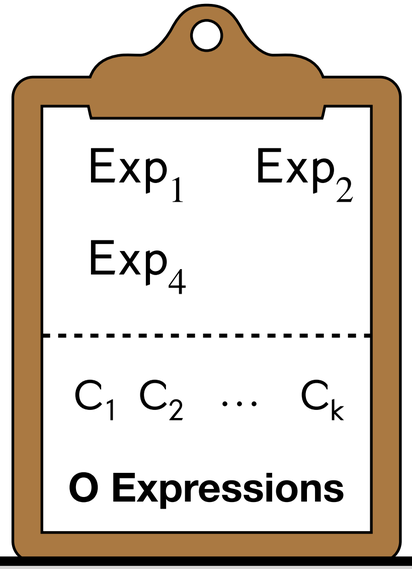




• • •



Step 2: Candidate Equivalence Generation



- **Sort expressions based on types**
- **Equate expressions and build predicates**
- **Class O return expressions must be mapped**

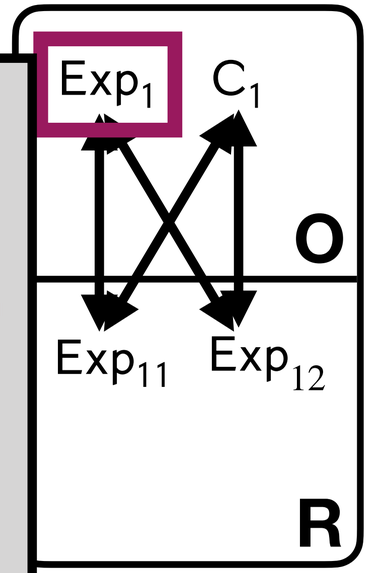
$Exp_1 = Exp_{11} \wedge Exp_2 = D_1 \wedge Exp_4 = Exp_{13}$

$Exp_1 = Exp_{12} \wedge Exp_2 = D_1 \wedge Exp_4 = Exp_{13}$

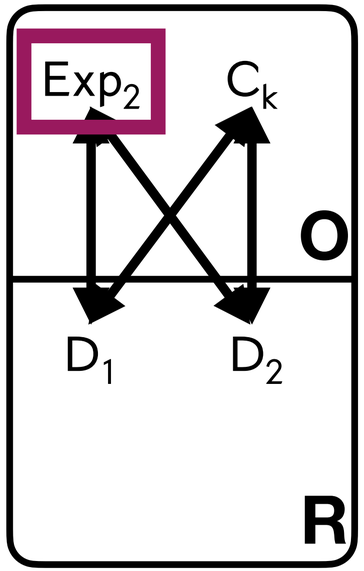
~~$C_1 = Exp_{11} \wedge Exp_2 = D_1 \wedge Exp_4 = Exp_{13}$~~

…

~~true~~

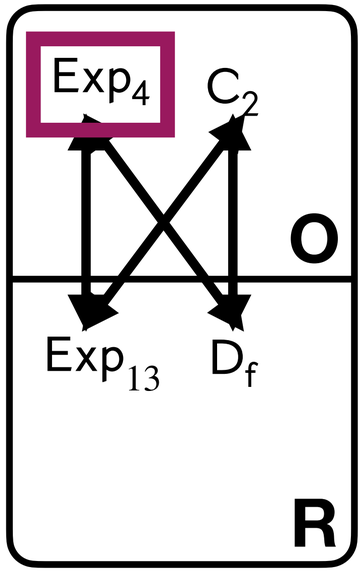


T_1



T_2

…



T_m

Step 2: Candidate Equivalence Generation

- **Sort expressions based on types**
- **Equate expressions and build predicates**
- **Class O return expressions must be mapped**
- **Build inter-class equivalence predicates**

$$Exp_1 = Exp_{11} \wedge Exp_2 = D_1 \wedge Exp_4 = Exp_{13}$$

$$Exp_1 = Exp_{12} \wedge Exp_2 = D_1 \wedge Exp_4 = Exp_{13}$$

$$\bullet \bullet \bullet$$

$$Exp_1 = Exp_{12} \wedge Exp_2 = D_2 \wedge Exp_4 = Exp_{13}$$

Step 2: Candidate Equivalence Generation

- **Sort expressions based on types**
- **Equate expressions and build predicates**
- **Class O return expressions must be mapped**
- **Build inter-class equivalence predicates**

$$Exp_1 = Exp_{11} \wedge Exp_2 = D_1 \wedge Exp_4 = Exp_{13}$$

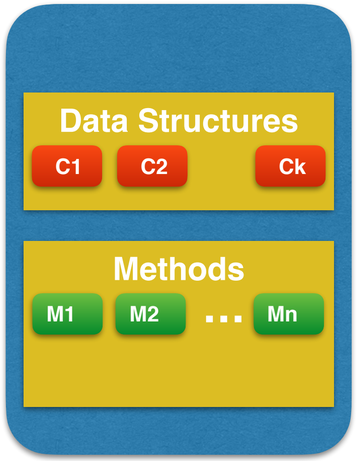
$$Exp_1 = Exp_{12} \wedge Exp_2 = D_1 \wedge Exp_4 = Exp_{13}$$

$$\cdots$$

$$Exp_1 = Exp_{12} \wedge Exp_2 = D_2 \wedge Exp_4 = Exp_{13}$$

Step 2: Candidate Equivalence Generation

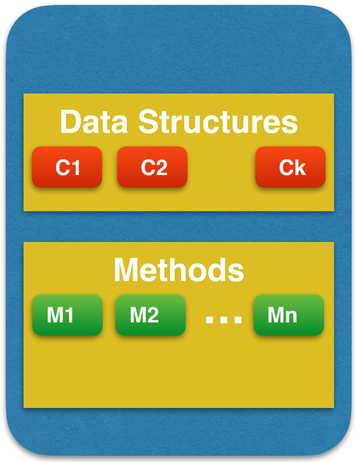
- Sort expressions based on types
- Equate expressions and build predicates
- Class O return expressions must be mapped
- Build inter-class equivalence predicates
- Eliminate invalid equivalence predicates



Class O

Step 2: Candidate Equivalence Generation

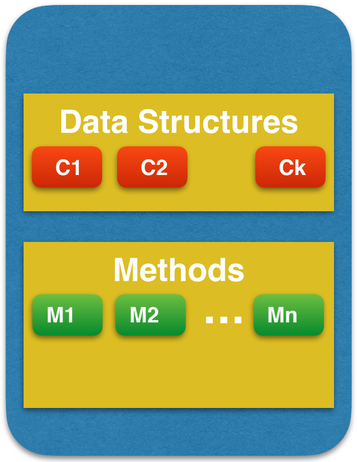
- Sort expressions based on types
- Equate expressions and build predicates
- Class O return expressions must be mapped
- Build inter-class equivalence predicates
- Eliminate invalid equivalence predicates



Class O

Step 2: Candidate Equivalence Generation

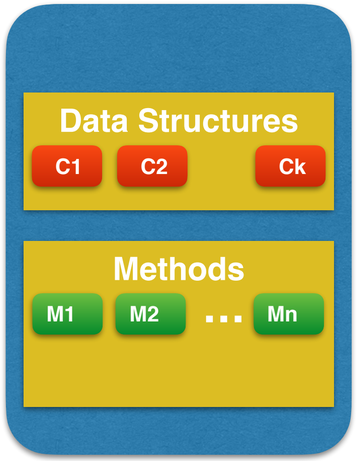
- Sort expressions based on types
- Equate expressions and build predicates
- Class O return expressions must be mapped
- Build inter-class equivalence predicates
- Eliminate invalid equivalence predicates



Class O

Step 2: Candidate Equivalence Generation

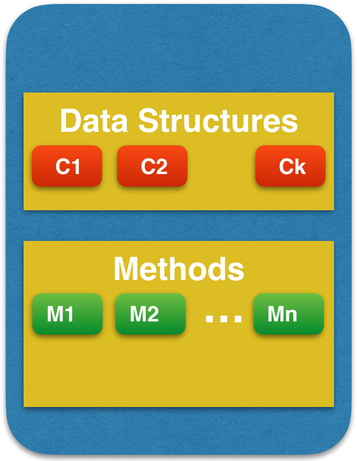
- Sort expressions based on types
- Equate expressions and build predicates
- Class O return expressions must be mapped
- Build inter-class equivalence predicates
- Eliminate invalid equivalence predicates



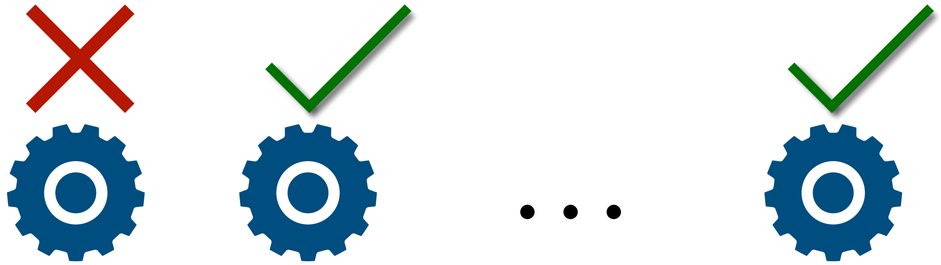
Class O

Step 2: Candidate Equivalence Generation

- Sort expressions based on types
- Equate expressions and build predicates
- Class O return expressions must be mapped
- Build inter-class equivalence predicates
- Eliminate invalid equivalence predicates

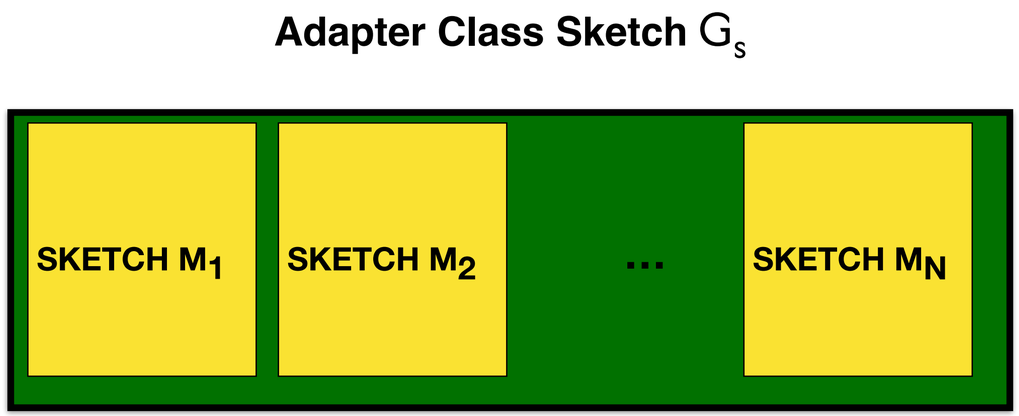


Class O

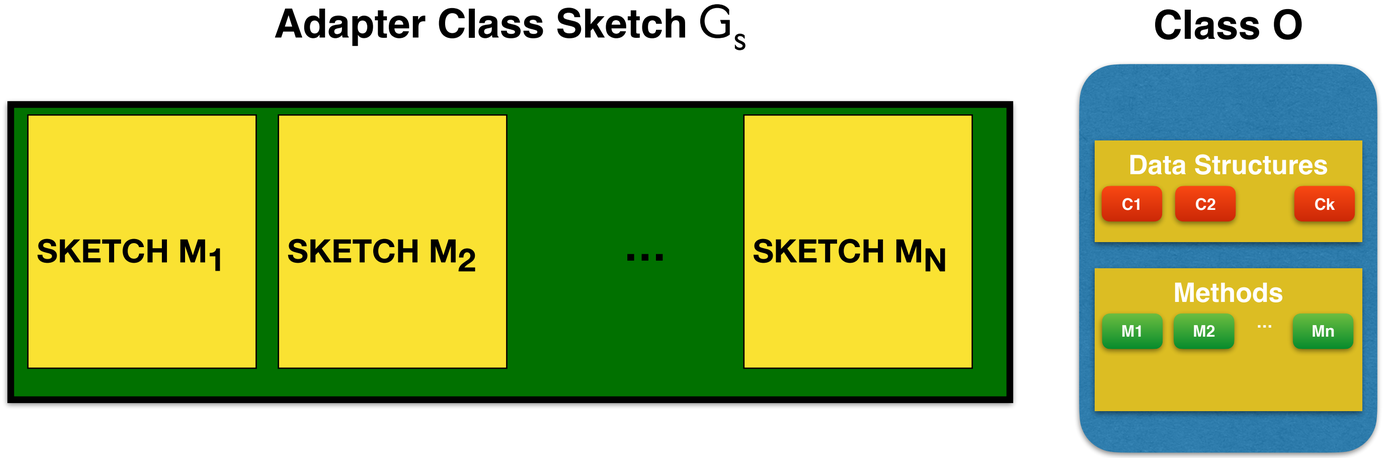


Iteration 1: Solving the Adapter Class Sketch

Iteration 1: Solving the Adapter Class Sketch



Iteration 1: Solving the Adapter Class Sketch

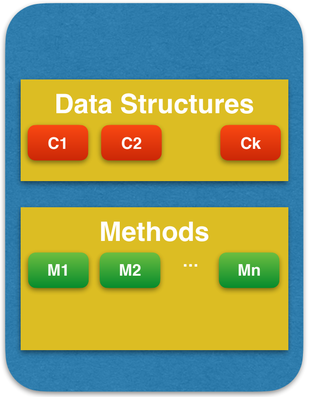


Iteration 1: Solving the Adapter Class Sketch

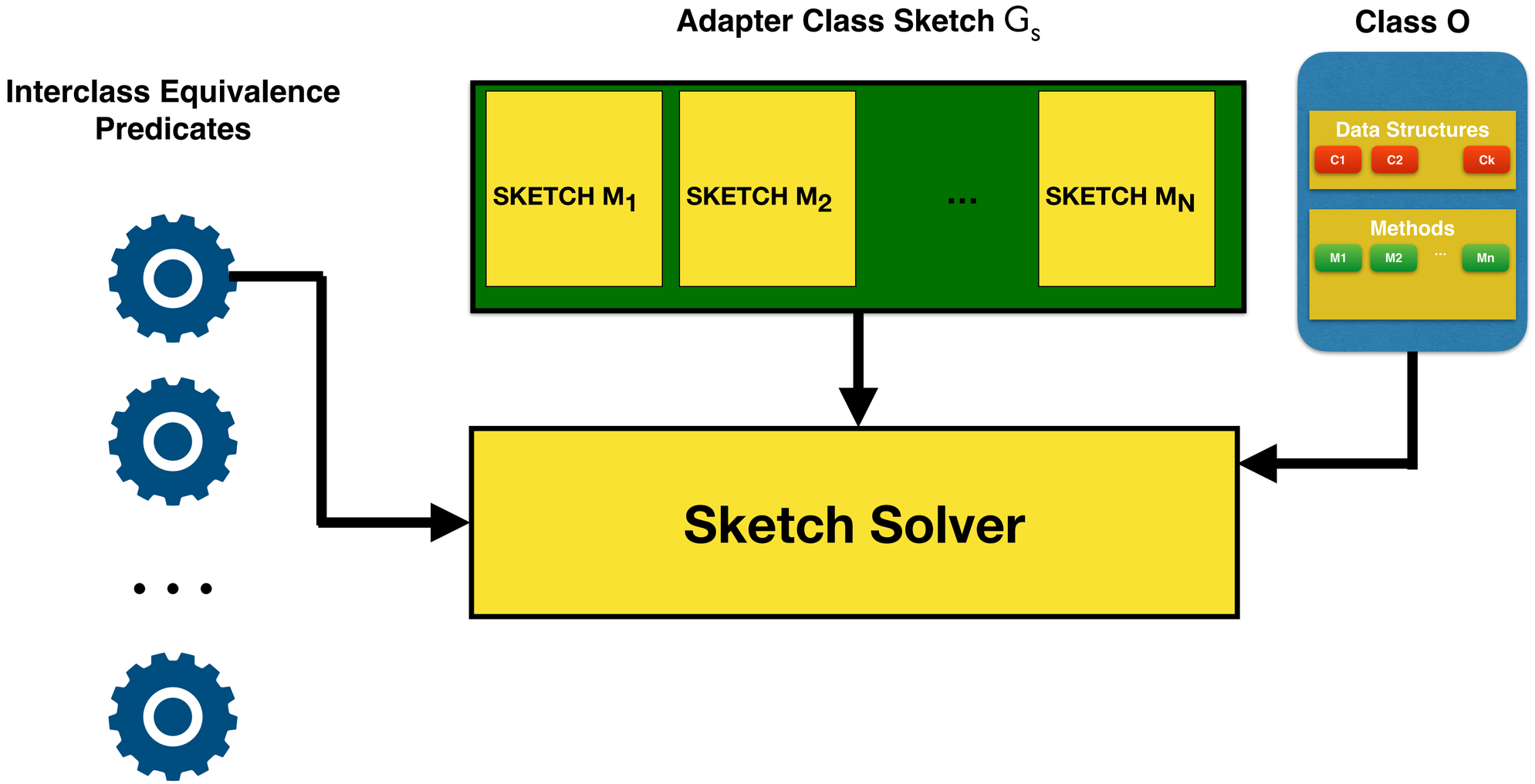
Interclass Equivalence Predicates

Adapter Class Sketch G_s

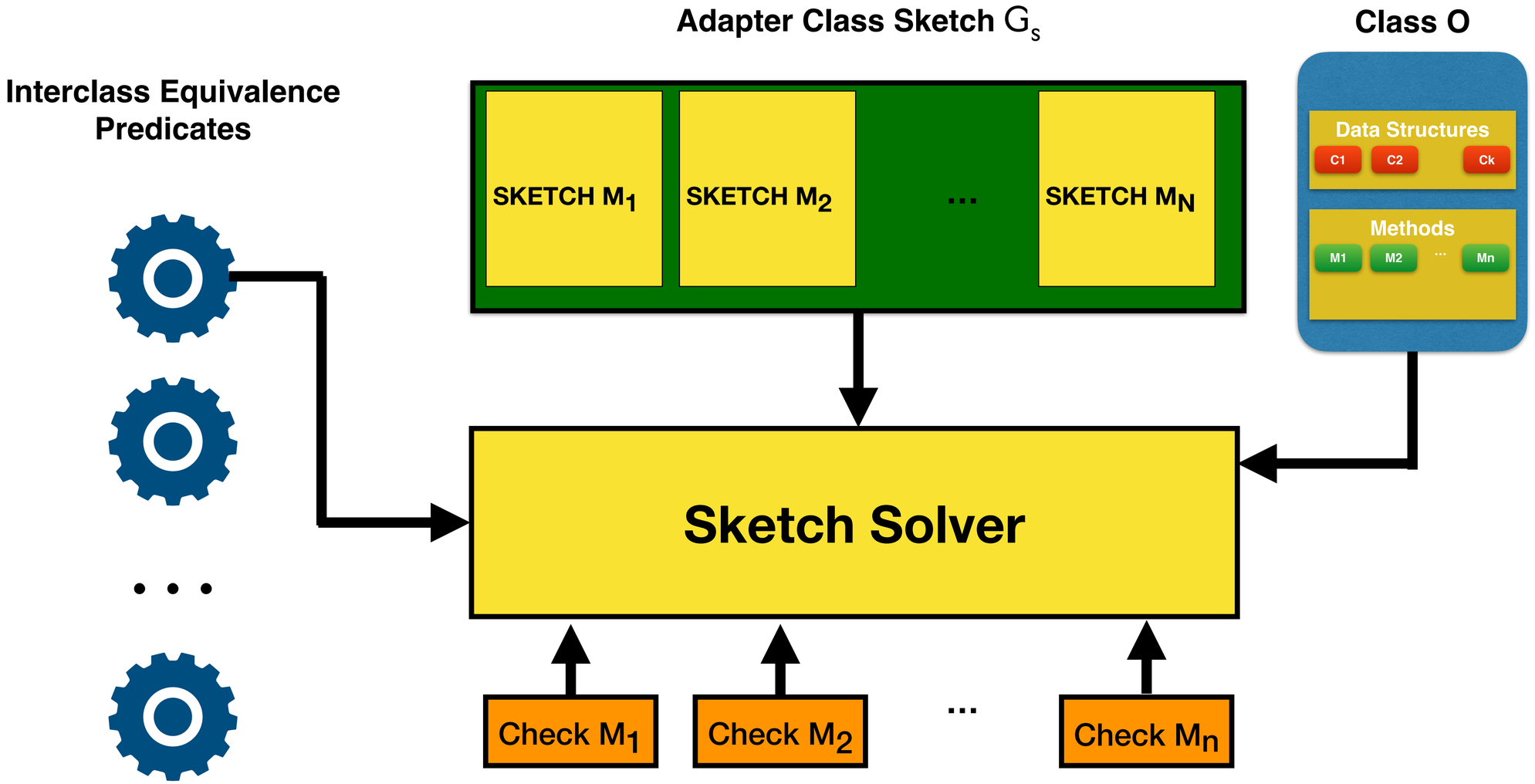
Class O



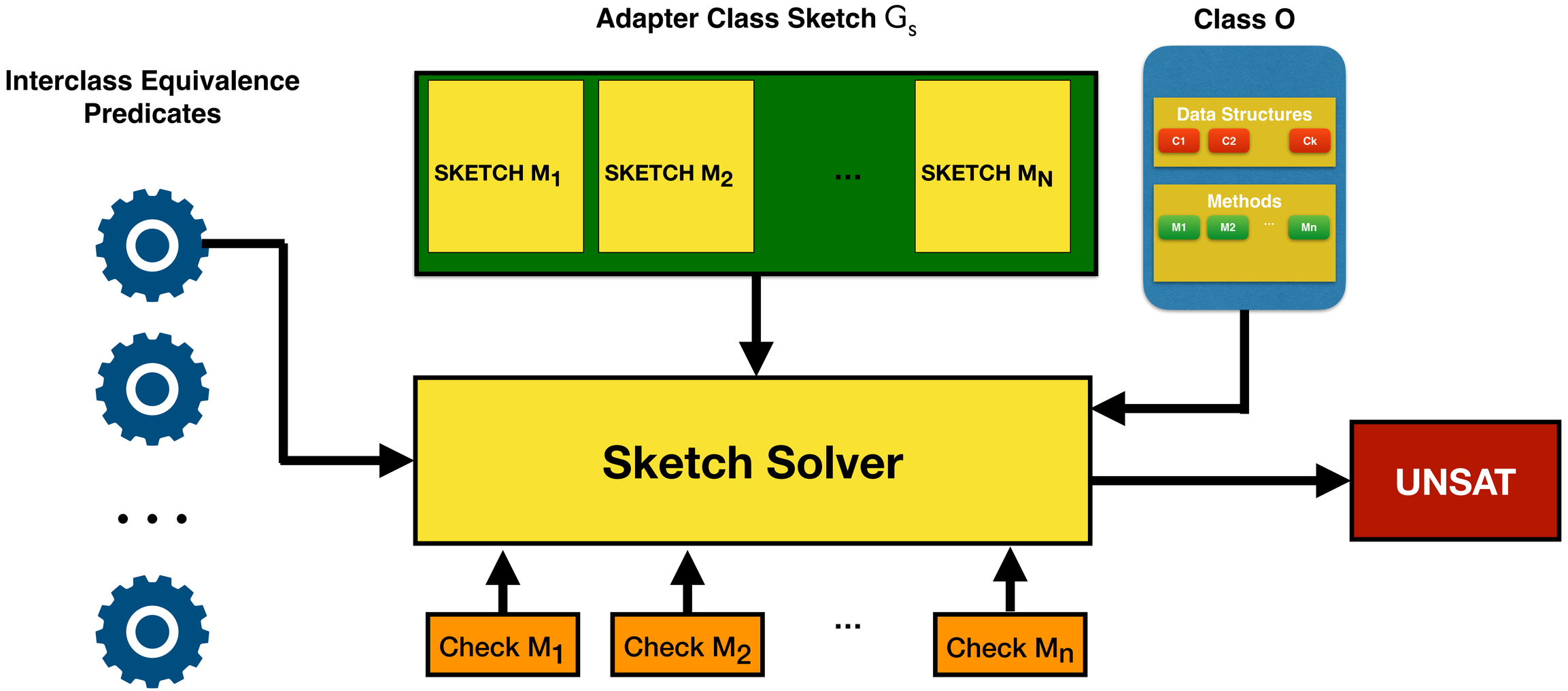
Iteration 1: Solving the Adapter Class Sketch



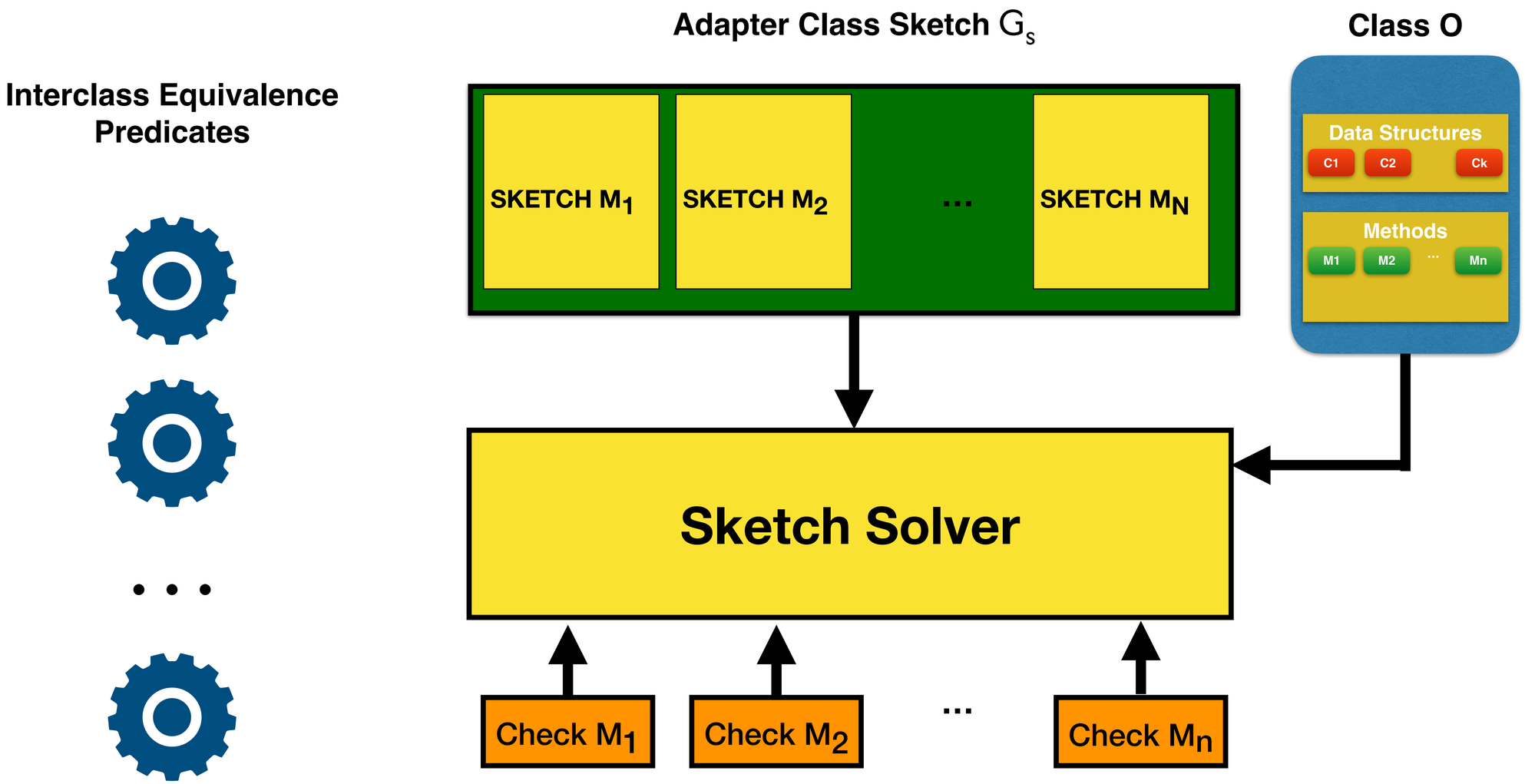
Iteration 1: Solving the Adapter Class Sketch



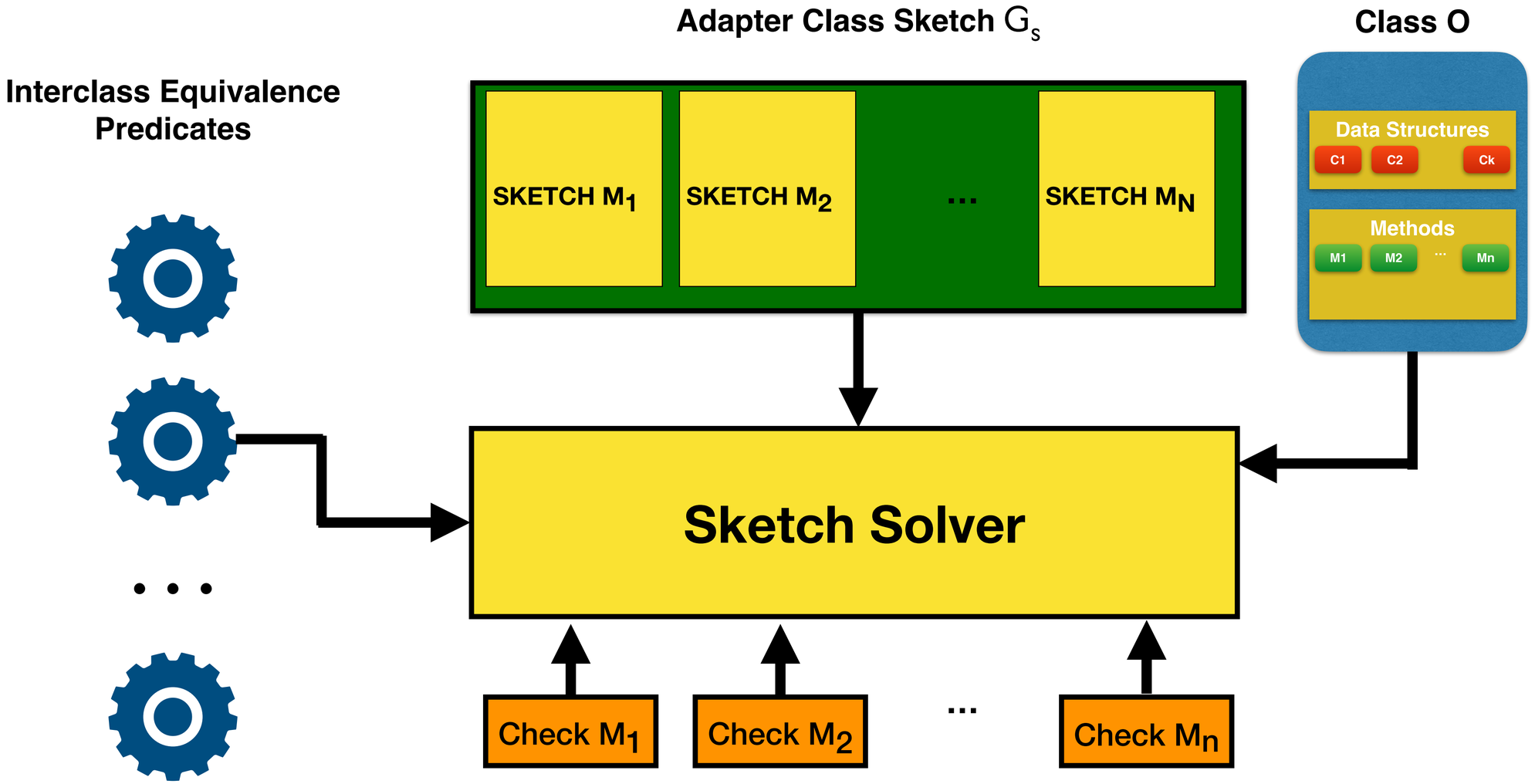
Iteration 1: Solving the Adapter Class Sketch



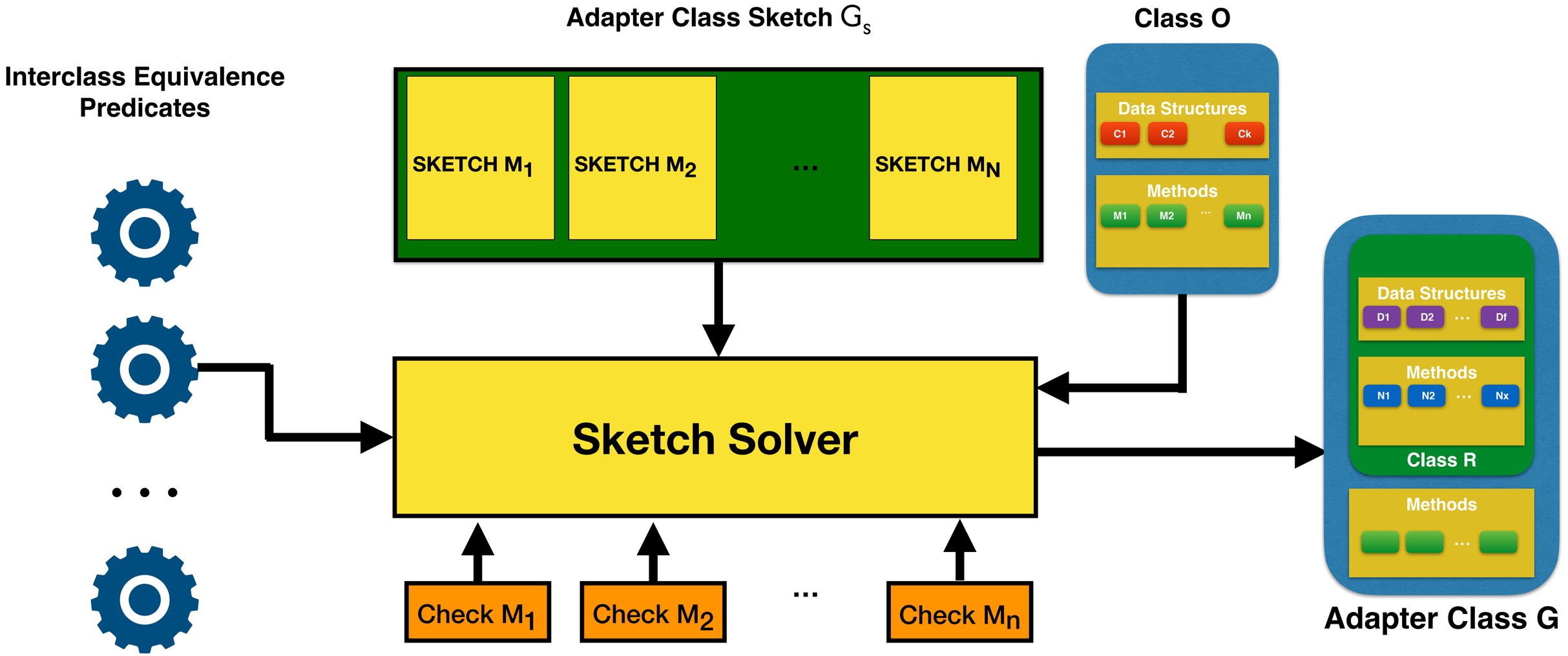
Iteration 2: Solving the Adapter Class Sketch



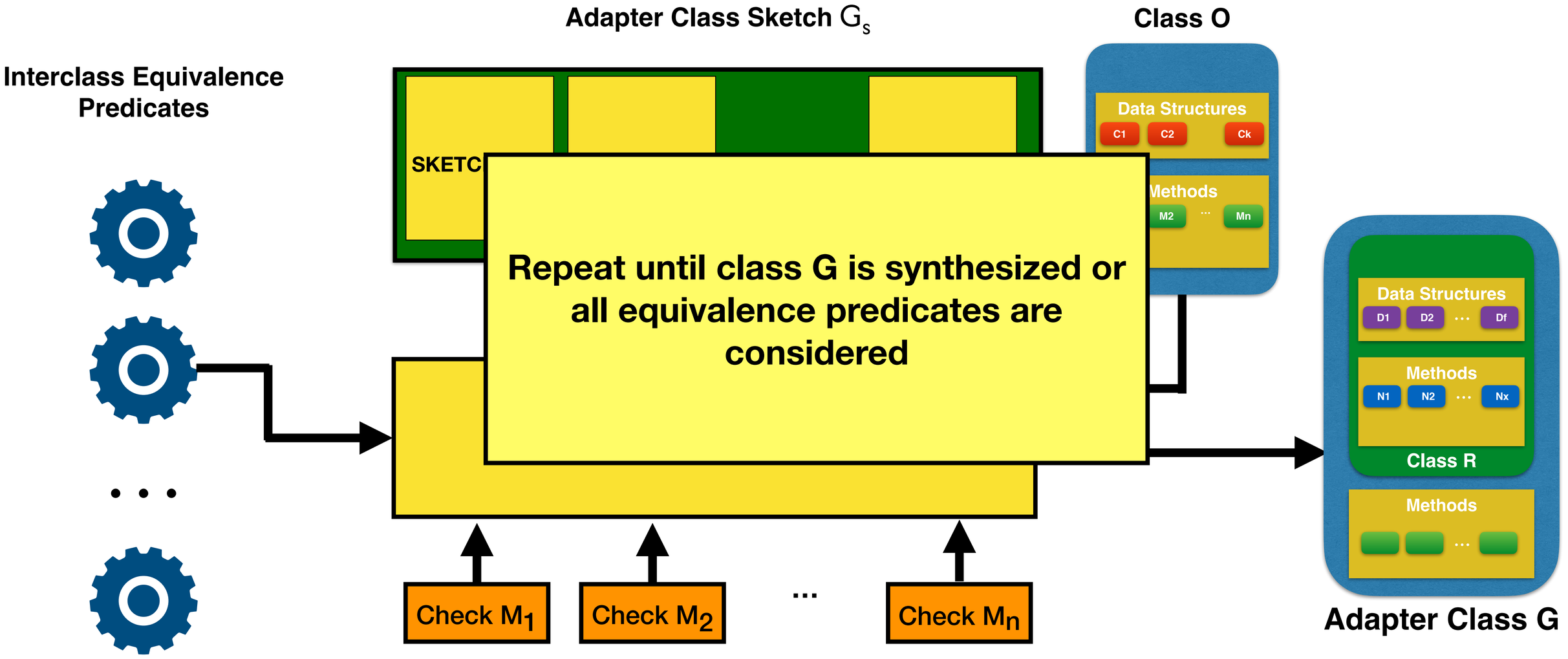
Iteration 2: Solving the Adapter Class Sketch



Iteration 2: Solving the Adapter Class Sketch



Iteration 2: Solving the Adapter Class Sketch



Implementation

- Built on JPF, Mistral and JavaSketch frameworks
- Exceptions are handled using optional try-catch blocks
- Maximum length for arrays and field dereferences: 5
  - Manually verified correctness beyond the limit
- Generic types are instantiated before synthesis
- Constructors are treated as special methods; no assume clause on receivers

Evaluation

- Evaluated on OpenSource Java Classes
- Replacing a class with a similar class
  - Considered 10 classes and 20 class replacement experiments
  - **Inter-class equivalence predicates generated**, Maximum: 8, Average: 2.6
  - **Method invocation sequence** identified as replacement, Maximum: 6, Average: 1.7
  - Allowed partial class synthesis
    - Synthesized **193/375** adapter methods

Evaluation

- Replacing an older version of the class with newer version
  - Considered 6 classes from GitHub > 450 stars
  - Replacement found for 4 classes.
  - 2 cases failed due to bug fixes
  - Fields added/removed/updated, changes to method signature, changes to implementation and cosmetic changes

Broader Implication

- Software is widely available, free and built by others
- Many implementations of the same functionality are available
- Key Question: How can I make sure my software always incorporates best implementation?
- MASK takes a step towards answering this question
- We envision MASK and it’s successors to enable softwares to better themselves

Conclusion

- MASK: Automatic context-agnostic class replacement
  - Synthesize a drop-in adapter class
  - Equivalence of original class and the adapter class is verified
  - Leverages symbolic execution, constraint solving and sketch based synthesis
- Evaluated on open source Java classes
- Can synthesize non-trivial replacements