**Errors**

- Conditional dispatch introduces the possibility of new types of errors at run-time
  - **Ambiguity Error** (multiple methods apply to a call, and neither is most specific)
  - **No-such-method Error** (no method can be found for a call)
  - Compiler prevents no-such-method errors by normally forcing all methods to override a *regular Java method*.
  - Programmer can circumvent this by labeling methods “inc” (to allow patterns to be used as preconditions)
  - Compiler can find most ambiguity errors. It cannot find errors caused by:

- **Multiple (interface) inheritance**. E.g. for interfaces A and B:

  ```java
  void f(A a) {}  
  void f(B b) {}  
  ```

- **Different deconstructors** on the same (or a related) class. E.g. for deconstructors d1 and d2:

  ```java
  void f(Point d1()) {}  
  void f(Point d2()) {}  
  ```

- **Deconstructors that modify the heap or aren’t deterministic**:

  ```java
  Deconstructor Point(int x, int y) {  
      Random r = new Random();  
      //Randomly return either 0 or 1  
      //for each of x and y  
      x = r.nextInt(2);  
      y = r.nextInt(2);  
  }
  ```

**Implementation**

- OOMatch has been implemented in Polyglot, a Java-to-Java compiler framework
- i.e. we translate OOMatch to Java
- Translation renames all methods, and creates dispatchers that select which method gets called
- Method calls call the dispatcher, instead of the original method
- Implementation only guaranteed to work for single-threaded programs. Concurrent programs are left for future work.