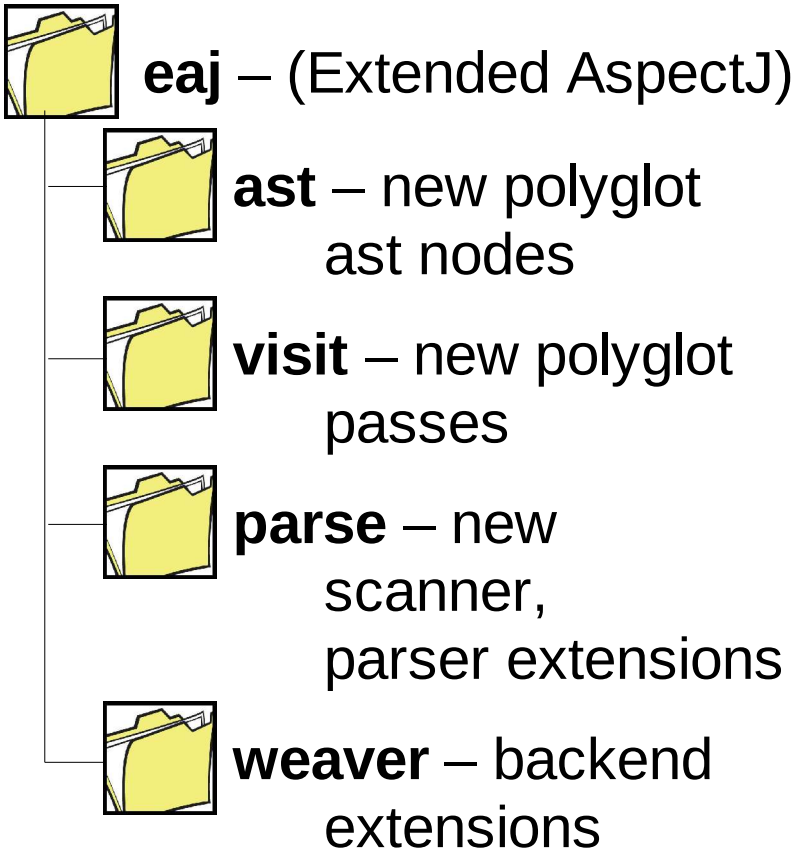
Extending abc

Aspect *Bench* Compiler

- abc...
  - ...is designed to provide a workbed for research and investigation
  - ...therefore must be flexible and extensible
- We ensured that it is by extending it

Layout of an extension

- 3 small extensions
- 2 ½ weeks coding (no prior experience with the codebase)
- ~1000 lines of code
- In self-contained directory structure



Layout of an extension

- *ExtensionInfo* is sub-classed for each extension.
  - Calls a new scanner and an extended parser
  - Creates factories for creating Polyglot AST nodes and type objects
  - (Re)Orders the passes of the compiler

The Cast Pointcut

- Defines a new shadow join point encompassing each explicit or implicit cast, and a pointcut to match it
- Syntax:

```
cast ( TypePattern )
```

matches all casts to a type matching the *TypePattern*

The Cast Pointcut

- For example

```
pointcut int_to_short(int x) :
    cast(short) && args(x);
```

- matches a cast from an `int` to a `short` and binds `x` to the original `int`

Check bounds with Cast Pointcut

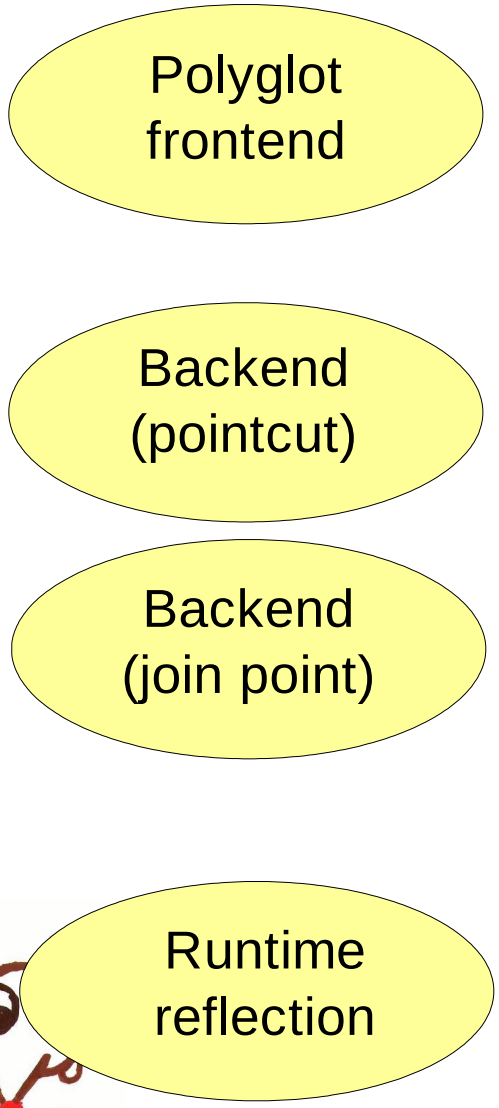
```
import uk.ac.ox.comlab.abc.eaj.lang.reflect.CastSignature;

aspect BoundsCheck
{
    before(int x) :
        cast(short) && args(x)
    {
        CastSignature s = (CastSignature)
            thisJoinPointStaticPart.getSignature();

        if (x > Short.MAX_VALUE || x < Short.MIN_VALUE) {
            System.out.println(
                "Warning: information lost casting " +
                x + " to a " + s.getCastType().getName());
        }
    }
}
```

Check bounds with Cast Pointcut

```
class LoseInformation
{
    public static void main(String[] args)
    {
        int x = 50000;
        short y;

        y = (short) x;
    }
}
```

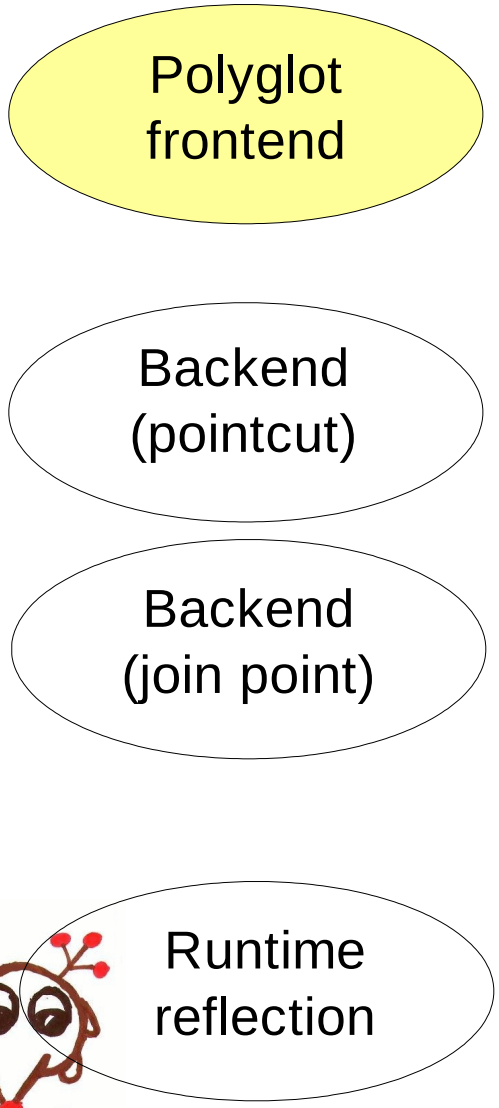
```
$ java LoseInformation

Warning: information lost casting 50000 to a short
```

Implementing the Cast Pointcut



- Frontend
  - New polyglot AST node: *PCCast*
- Backend
  - Cast pointcut class
  - Cast shadow join point class
- Runtime
  - Cast signature

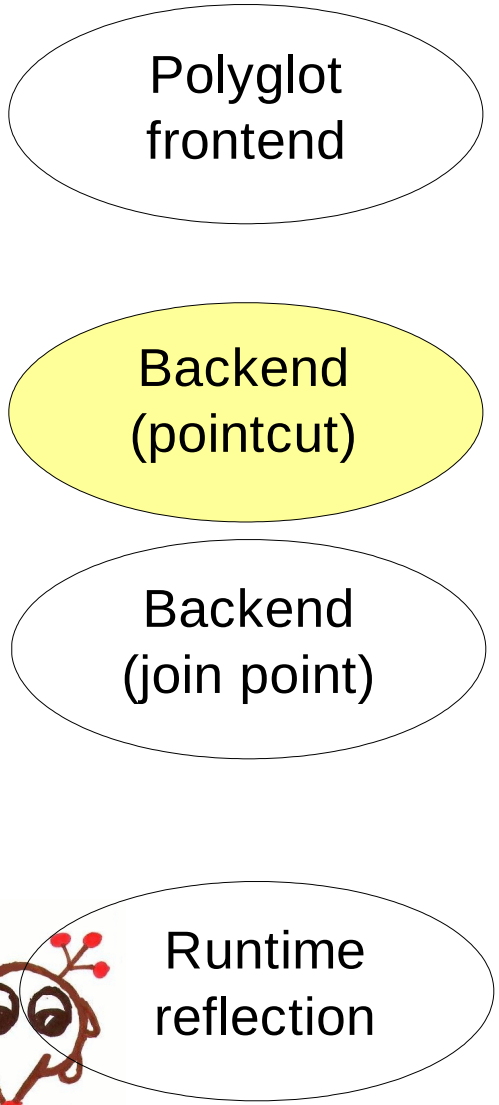
Implementing the Cast Pointcut



- Create a polyglot AST node which stores the *TypePattern*

```
Class PCCast_c extends Pointcut_c
                implements PCCast
{
  protected TypePatternExpr type_pattern;
     .
     .
     .
  public abc.weaving.aspectinfo.Pointcut makeAIPointcut()
  {
    return new
      abc.eaj.weaving.aspectinfo.Cast
         (type_pattern.makeAITypePattern(), position());
  }
}
```

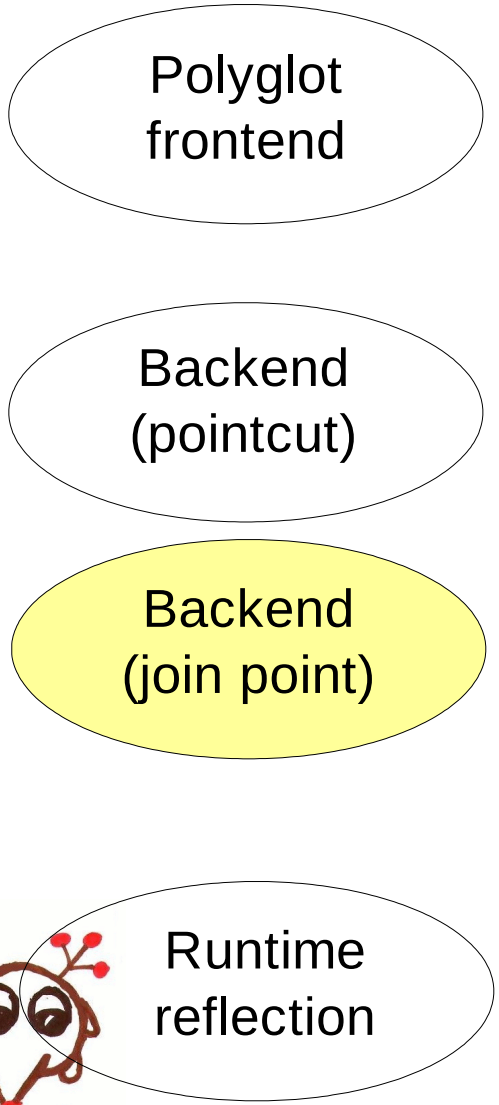
Implementing the Cast Pointcut



- The cast pointcut matches cast join points if they cast a type matching a *TypePattern*

```
class Cast extends ShadowPointcut
{
  private TypePattern type_pattern;
    .
    .
    .
  protected Residue matchesAt(ShadowMatch sm)
  {
    if (! (sm instanceof CastShadowMatch)) return null;
    Type cast_to = ((CastShadowMatch) sm).getCastType();

    if (!getPattern().matchesType(cast_to)) return null;
    return AlwaysMatch.v;
  }
}
```

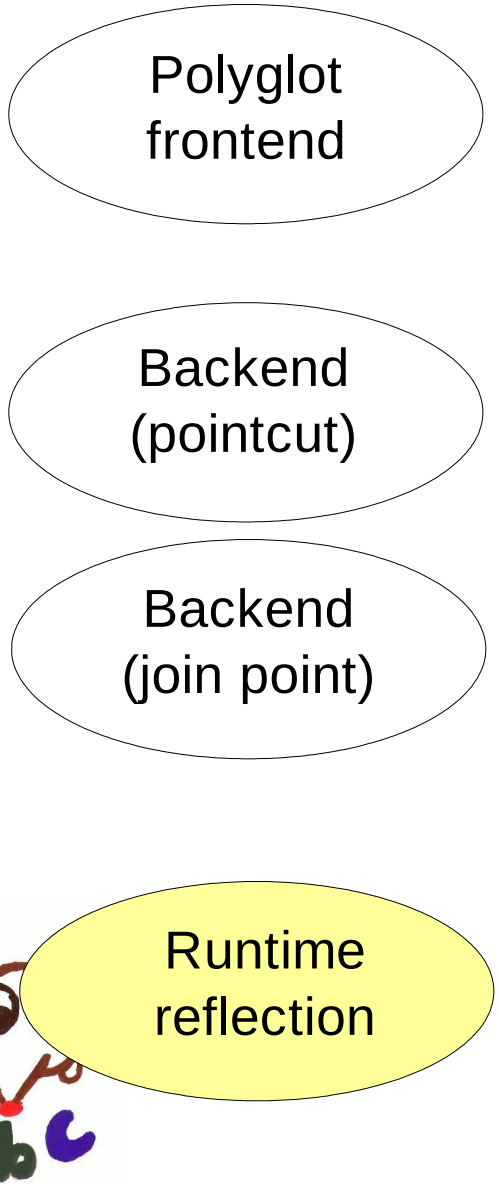
Implementing the Cast Pointcut



- Casts only occur on the right-hand-side of assignments in Jimple

```
class CastShadowMatch extends StmtShadowMatch
{
  private Type cast_to;
    .
    .
  public static CastShadowMatch(MethodPosition pos)
  {
    if (!(pos instanceof StmtMethodPosition)) return null;
    .
    .
    if (!(rhs instanceof CastExpr)) return null;
    Type cast_to = ((CastExpr) rhs).getCastType();

    return new CastShadowMatch(pos.getContainer(),
                               stmt, cast_to);
  }
}
```

Implementing the Cast Pointcut



- *CastSignature,* in the runtime library, allows the retrieval of the type of a cast at runtime
- The information needed by the runtime is encoded by the compiler in the same way that ajc does

Future extensibility

- AspectJ
  - When making compiler extensions you often want to change a class in the compiler source.
  - If you do, this leads to maintenance problems.
  - If you don't, you may have to subclass whole class hierarchies.
  - A possible solution is to use Intertype declarations.