abc: an extensible AspectJ compiler

- Programming Tools Group, University of Oxford
- Sable Research Group, McGill University
- BRICS, Aarhus Universitet

The need for an AOP workbench

new features:
parametric introductions
dataflow pointcut
remote pointcut
predicted cflow
tracecuts
event-based AOP
symmetric composition
AHEAD
...

new analyses:
pure aspects
static cflow
thisJoinPoint escape
...

code generation:
around without closures
inline advice or not?
...

shared tool building helps progress

Goals of *abc*

a compiler workbench for AspectJ to:

- explore AOP language design space (this talk)
- experiment with better code generation
- experiment with static analyses
  for safety checks and optimisations

clarify AspectJ language definition:
- grammar
- scope rules for ITDS, ...

Does *ajc* meet these goals?

proven workbench: AspectJ language was developed on it!

- fast compiler
- incremental compilation
- tight integration with Eclipse

ajc has evolved from a research tool to a production compiler

Difficult to meet *abc* goals:

- 119 changes to the text of the Eclipse Java compiler
- customised BCEL
- no LALR(1) grammar
- no analysis & optimisation framework
- designed for compilation speed

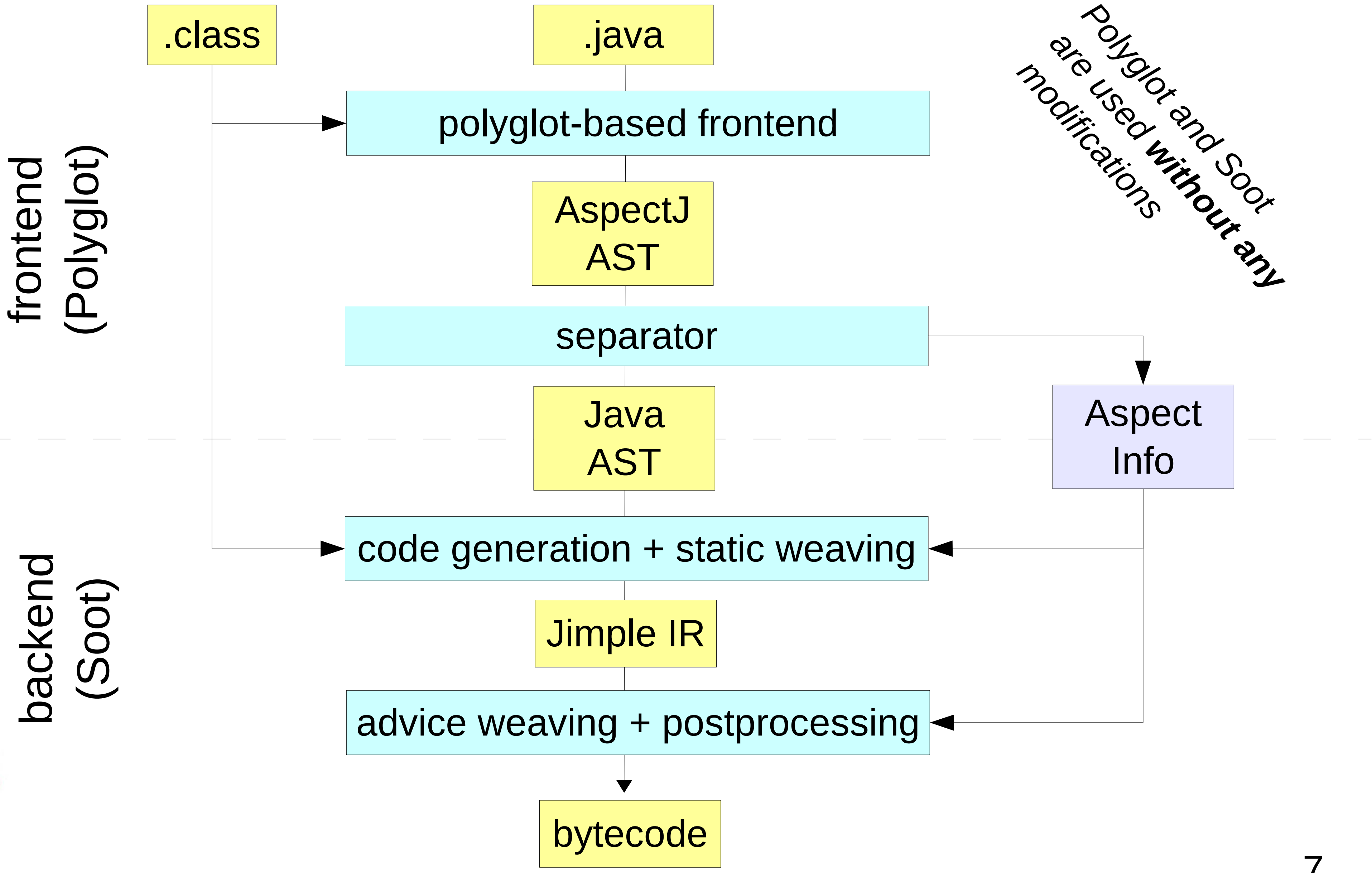
Extensibility of *abc*

- extensible frontend via *Polyglot*:
  - new syntax
  - new types
- extensible backend via *Soot*:
  - new joinpoints ⇐ focus of this talk
  - new analyses
  - new optimisations

How to add new joinpoint+pointcut?

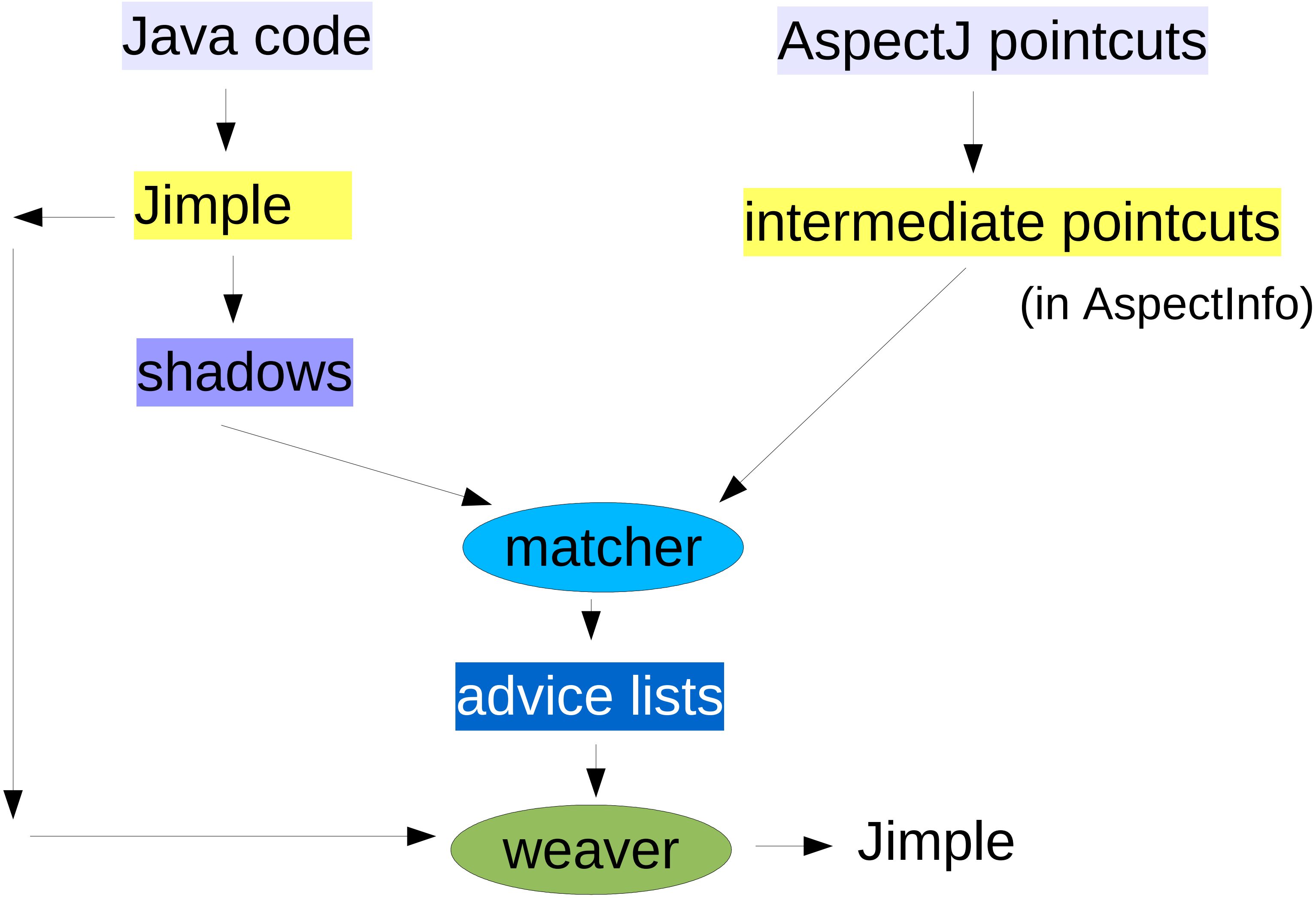
- architecture of abc
- intermediate representation for pointcuts
- how to find shadows in code
- example: array access joinpoint

Architecture of abc



Polyglot and Soot are used **without any** modifications

Pointcuts and weaving in *abc*



Java code

AspectJ pointcuts

Jimple

intermediate pointcuts

(in AspectInfo)

shadows

matcher

advice lists

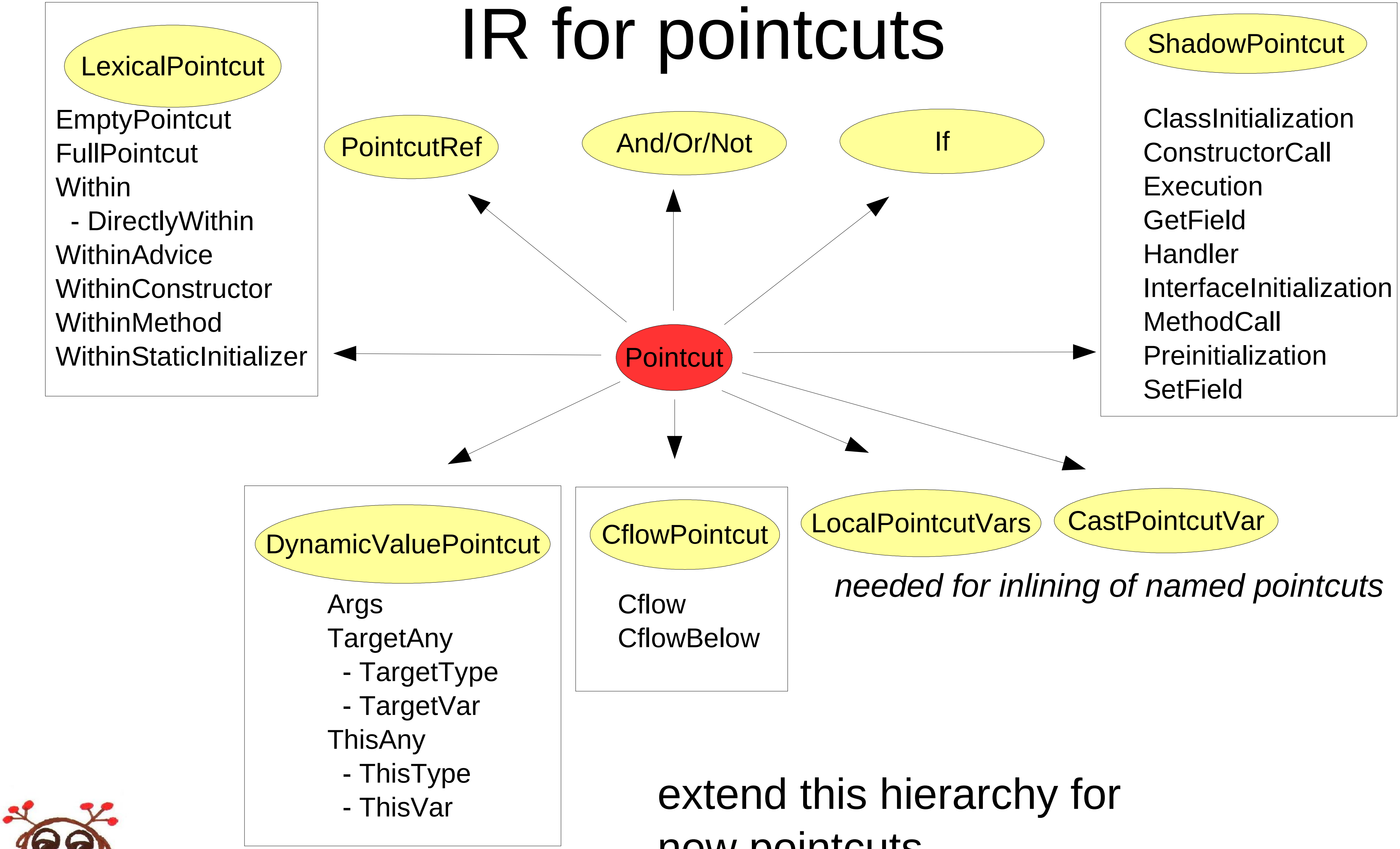
weaver

Jimple

How to add a new joinpoint+pointcut?

- architecture of abc
- intermediate representation for pointcuts
- how to find shadows in code
- example: array access joinpoint

IR for pointcuts



Pointcut

LexicalPointcut

EmptyPointcut
FullPointcut
Within
 - DirectlyWithin
WithinAdvice
WithinConstructor
WithinMethod
WithinStaticInitializer

PointcutRef

And/Or/Not

If

ShadowPointcut

ClassInitialization
ConstructorCall
Execution
GetField
Handler
InterfaceInitialization
MethodCall
Preinitialization
SetField

DynamicValuePointcut

Args
TargetAny
 - TargetType
 - TargetVar
ThisAny
 - ThisType
 - ThisVar

CflowPointcut

Cflow
CflowBelow

LocalPointcutVars

CastPointcutVar

needed for inlining of named pointcuts

extend this hierarchy for
new pointcuts

Examples: AspectJ pointcut to IR

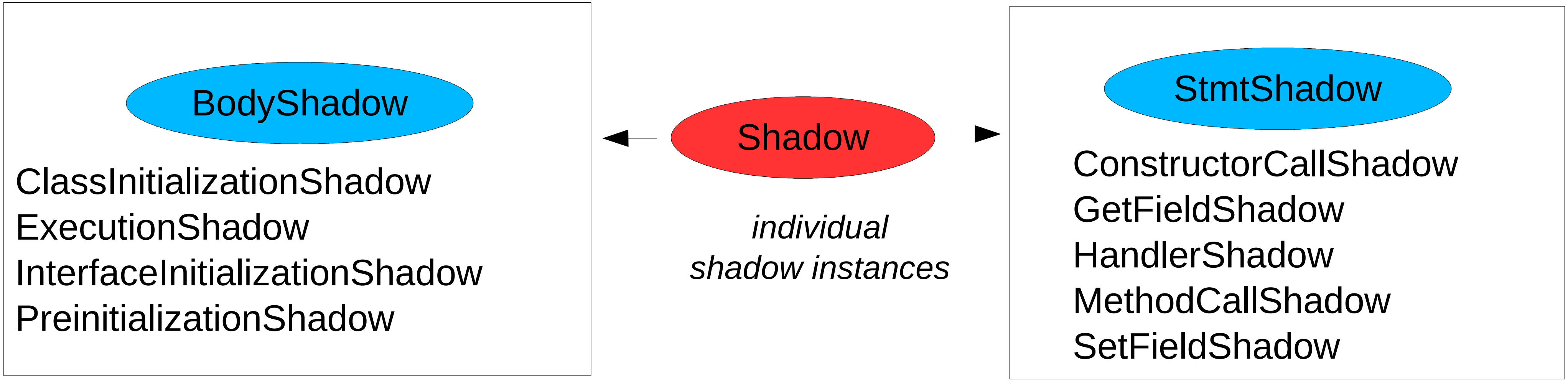
AspectJ	Intermediate Representation
execution(int Foo.foo(char))	withinmethod(int Foo.foo(char)) && execution()
adviceexecution()	withinadvice() && execution()
call(Foo.new(int))	constructorcall(Foo.new(int))
initialization(Foo.new(..))	(withinconstructor(Foo.new(..)) && classinitialization()) || interfaceinitialization(Foo)

new pointcuts also need to be translated to IR

How to add a new joinpoint+pointcut?

- architecture of abc
- intermediate representation for pointcuts
- how to find shadows in code
- example: array access joinpoint

Shadow Kinds



BodyShadow

ClassInitializationShadow
ExecutionShadow
InterfaceInitializationShadow
PreinitializationShadow

Shadow

individual
shadow instances

StmtShadow

ConstructorCallShadow
GetFieldShadow
HandlerShadow
MethodCallShadow
SetFieldShadow

for blocks of Jimple code

for single Jimple statements
(or a pair, for constructor call)

and a singleton class for each kind named

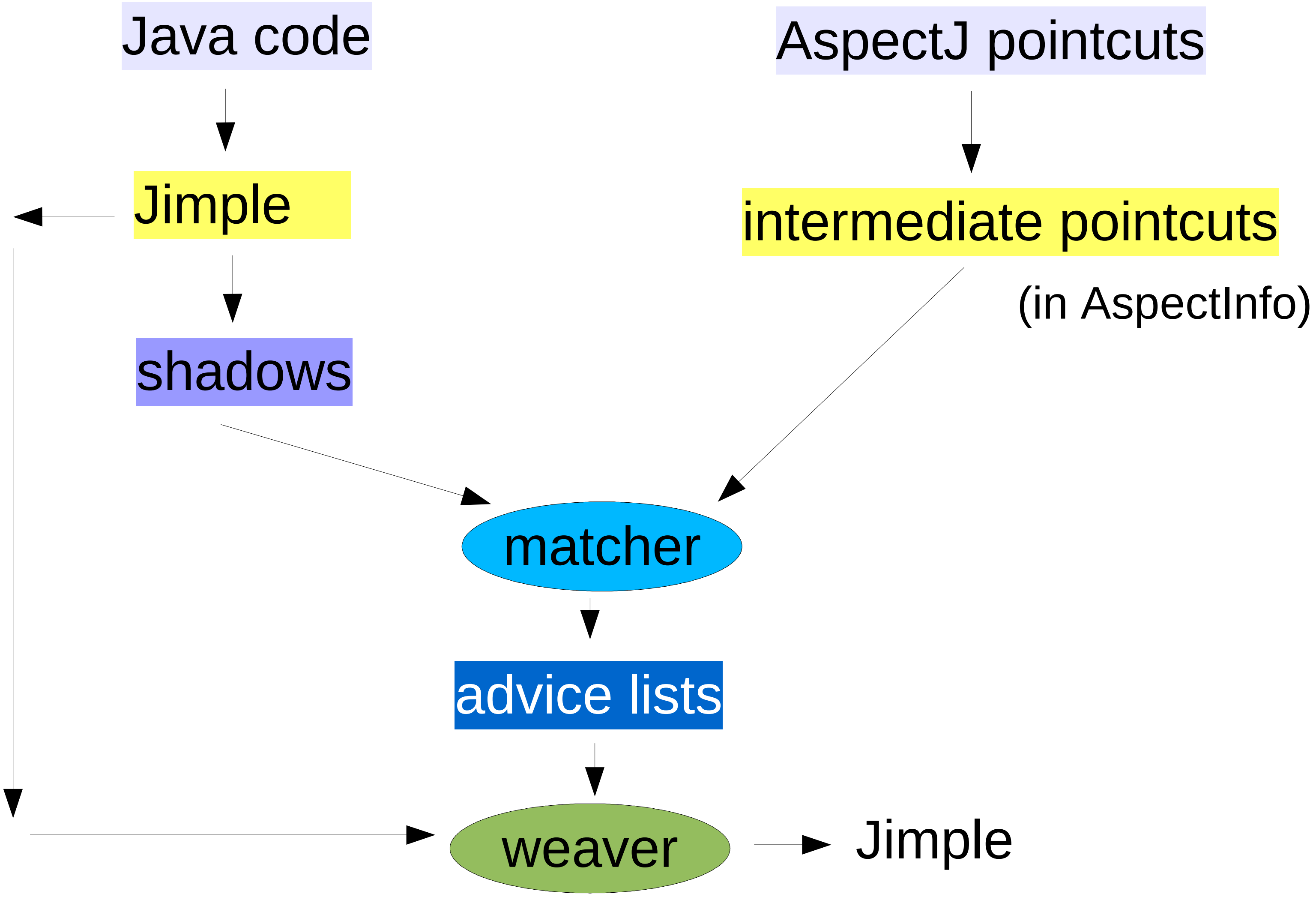
<*kind*>ShadowType (a subtype of ShadowType)

with a method

Shadow matchesAt(pos)

each new joinpoint requires new Shadow and ShadowType class

Pointcuts and weaving in *abc*



Java code

AspectJ pointcuts

Jimple

intermediate pointcuts

(in AspectInfo)

shadows

matcher

advice lists

weaver

Jimple

Computing Advice Lists (simplified)

```
for each weavable class C
    for each method M in C
        for each "position" pos in M
            for each shadow type t
                Shadow sh = t.matchesAt(pos);
                for each advice declaration ad
                    Pointcut pc = ad.getPointcut();
                    Residue r = pc.matchesAt(sh);
                    add (pos,ad,r) to "advice list" of M;
```

“positions” are user-definable

Residue is an IR of the dynamic test to be inserted (e.g. for if, args, ...)

advice lists are applied in a separate weaving pass

How to add a new joinpoint+pointcut?

- architecture of abc
- intermediate representation for pointcuts
- how to find shadows in code
- example: array access joinpoint

Example: array pointcuts

```
Bar around(Bar array[], int index) :
    arrayget() && target(array) && args(index) {
        Bar value=proceed(array,index);
        return value;
}
void around (Bar array[], Bar value, int index) :
    arrayset() && target(array) && args(value, index) {
        proceed(array,value,index);
}
```

general syntax:

```
basic_pointcut_expr ::=
    PC_ARRAYGET LPAREN RPAREN
  | PC_ARRAYSET LPAREN RPAREN
```

with thanks to Bruno Harbulot !

Six steps for adding new joinpoint

General	**This example**
• extend parser, new AST nodes	grammar rule, new pointcut AST class
• AspectInfo: IR for aspect-specific features	new class for *arrayget* pointcut
• extend shadow finder	how to find *arrayget* shadow
• New runtime	dynamic representation of *arrayget* joinpoint
• Extend driver classes to use new runtime	AbcExtension

Finding *arrayget* shadows in Jimple

Java:

```
void shift(Object[] arr) {
        for (int i = arr.length; i>0; i--)
                arr[i] = arr[i-1];
}
```

Jimple:

```
void shift(java.lang.Object[])
{
    java.lang.Object[] arr;
    int i, $i0;
    java.lang.Object $r0;
    arr := @parameter0: java.lang.Object[];
    i = lengthof arr;
    goto label1;
 label0:
    $i0 = i - 1;
    $r0 = arr[$i0];
    arr[i] = $r0;
    i = i - 1;
 label1:
    if i > 0 goto label0;
    return;
}
```

Jimple is:
- typed
- stackless

Finding *arrayget* shadows (1)

```
public static ArrayGetShadow matchesAt(MethodPosition pos)
   {
      if (!(pos instanceof StmtMethodPosition)) return null;
      Stmt stmt = ((StmtMethodPosition) pos).getStmt();

      if (!(stmt instanceof AssignStmt)) return null;
      AssignStmt assign=(AssignStmt)stmt;
      Value rhs = assign.getRightOp();

      if(!(rhs instanceof ArrayRef)) return null;
      ArrayRef ref=(ArrayRef)rhs;

      Value index=ref.getIndex();
      ... restructure if necessary, next slide ....

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

```
$r0 = arr[$i0];
```

Finding *arrayget* shadows (2)

```
// make sure the index is a local.
// restructure if necessary.
if (!(index instanceof Local)) {
    Body body=pos.getContainer().getActiveBody();
    Chain statements=body.getUnits().getNonPatchingChain();
    LocalGeneratorEx lg=new LocalGeneratorEx(body);

    Local l=lg.generateLocal(index.getType());
    AssignStmt as=Jimple.v().newAssignStmt(l, index);

    statements.insertBefore(as,stmt);
    stmt.redirectJumpsToThisTo(as);

    ref.setIndex(l);
}
```

```
label:  $r0 = arr[0]
⇒
        int $i3; ...
label': $i3 = 0;
label:  $r0 = arr[$i3];
```

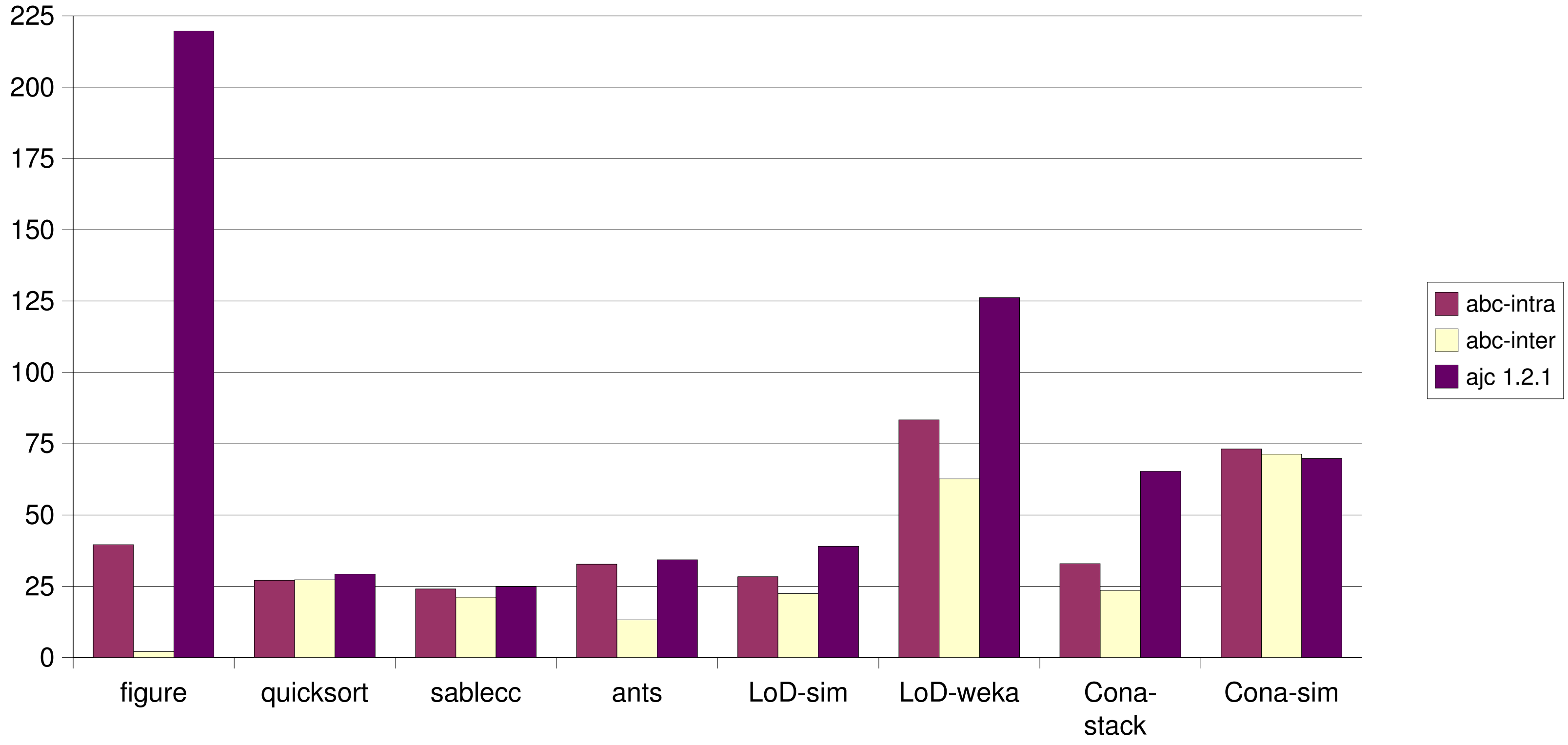
Goals of *abc* revisited

a compiler workbench for AspectJ to:

- explore AOP language design space (this talk)
- experiment with better code generation
- experiment with static analyses for safety checks and optimisations

Other goals of abc addressed in PLDI 2005

PLDI 2005 results: runtime speed



compile-time speed is *not* a goal of abc

Papers by users of *abc*

Bruno Harbulot and John Gurd:
A join point for loops in AspectJ.
FOAL 2005.

Tomoyuki Aotani and Hidehiko Masuhara:
Compiling conditional pointcuts for user-level semantic pointcuts.
SPLAT 2005.

Eric Bodden:
Concern-specific languages and their implementation with abc.
SPLAT 2005.