From Encapsulation to Ownership

A (partial) history

SOFTWARE ENGINEERING

Report on a conference sponsored by the

NATO SCIENCE COMMITTEE

Garmisch, Germany, 7th to 11th October 1968

Chairman: Professor Dr. F. L. Bauer

Co-chairmen: Professor L. Bolliet, Dr. H. J. Helms

Editors: Peter Naur and Brian Randell

MASS PRODUCED SOFTWARE COMPONENTS, BY M.D. MCILROY

My thesis is that the software industry is weakly founded, and that one aspect of this weakness is the absence of a software components subindustry.

SYNTACTIC CONTROL OF INTERFERENCE

John C. Reynolds
School of Computer and Information Science
Syracuse University

POPL 78

```
procedure fact(n, f); integer n, f;
    begin integer k;
    k := 0; f := 1;
    while k ≠ n do
        begin k := k + 1; f := k × f end
    end .
```

```
procedure transpose(X, Y); real array X, Y;
    for i := 1 until 50 do
        for j := 1 until 50 do
            Y(i, j) := X(j, i)
```

```
x := x + 1 || y := y × 2
```

```
procedure itersucc(n,p); integer n; procedure p;
    begin integer k;
    k := nodelist(n);
    while k ≠ 0 do
        begin p(item(k)); k := link(k) end
    end
```

Syntactic Control

<δ exp> ::= <δ var>

<integer exp> ::= 0

| <integer exp> + <integer exp>

<sta> ::= <δ var> := <δ exp>

<sta> ::= noaction

| <sta> ; <sta>

| while <Boolean exp> do <sta>

<sta> ::= new <δ var id> in <sta>

<ω> ::= <ω id>

<ω → ω'> ::= λ <ω id>. <ω'>

<ω'> ::= <ω → ω'> (<ω>)

$\langle \Pi(i_1:\omega_1, \ldots, i_n:\omega_n) \rangle$ ::=

$\langle i_1:\langle\omega_1\rangle, \ldots, i_n:\langle\omega_n\rangle \rangle$

$\langle\omega_k\rangle$ ::= $\langle \Pi(i_1:\omega_1, \ldots, i_n:\omega_n)\rangle . i_k$

<ω> ::= if <Boolean exp> then <ω> else <ω>

<ω> ::= Y(<ω → ω>)

```
(λfact: integer exp → (integer var → sta). S)
    (λn: integer exp. λf: integer var.
       new k: integer in
           (k := 0; f := 1;
           while k ≠ n do (k := k+1; f := k×f)))
```

```
let repeat = λ(s: sta, b: Boolean exp).
    (s; while ¬ b do s)   .
```

```
x := n || y := n
```

```
let twice = λs: sta. (s; s) in
    (twice ( x := x+1) || twice(y := y×2))
```

$$\underline{Y}(f) \Rightarrow f(\underline{Y}(f))$$

Passive

The Geneva Convention On The Treatment of Object Aliasing

John Hogg
Bell-Northern Research

Doug Lea
SUNY Oswego

Alan Wills
University of Manchester

Dennis deChampeaux
Hewlett-Packard

Richard Holt
University of Toronto

ECOOP 91,
in Geneva

Islands: Aliasing Protection In Object-Oriented Languages

John Hogg

Bell-Northern Research
Ottawa, Ontario

OOPSLA 1991

Side-effect Free Functions

class name Complex

instance variable names x

y

instance methods

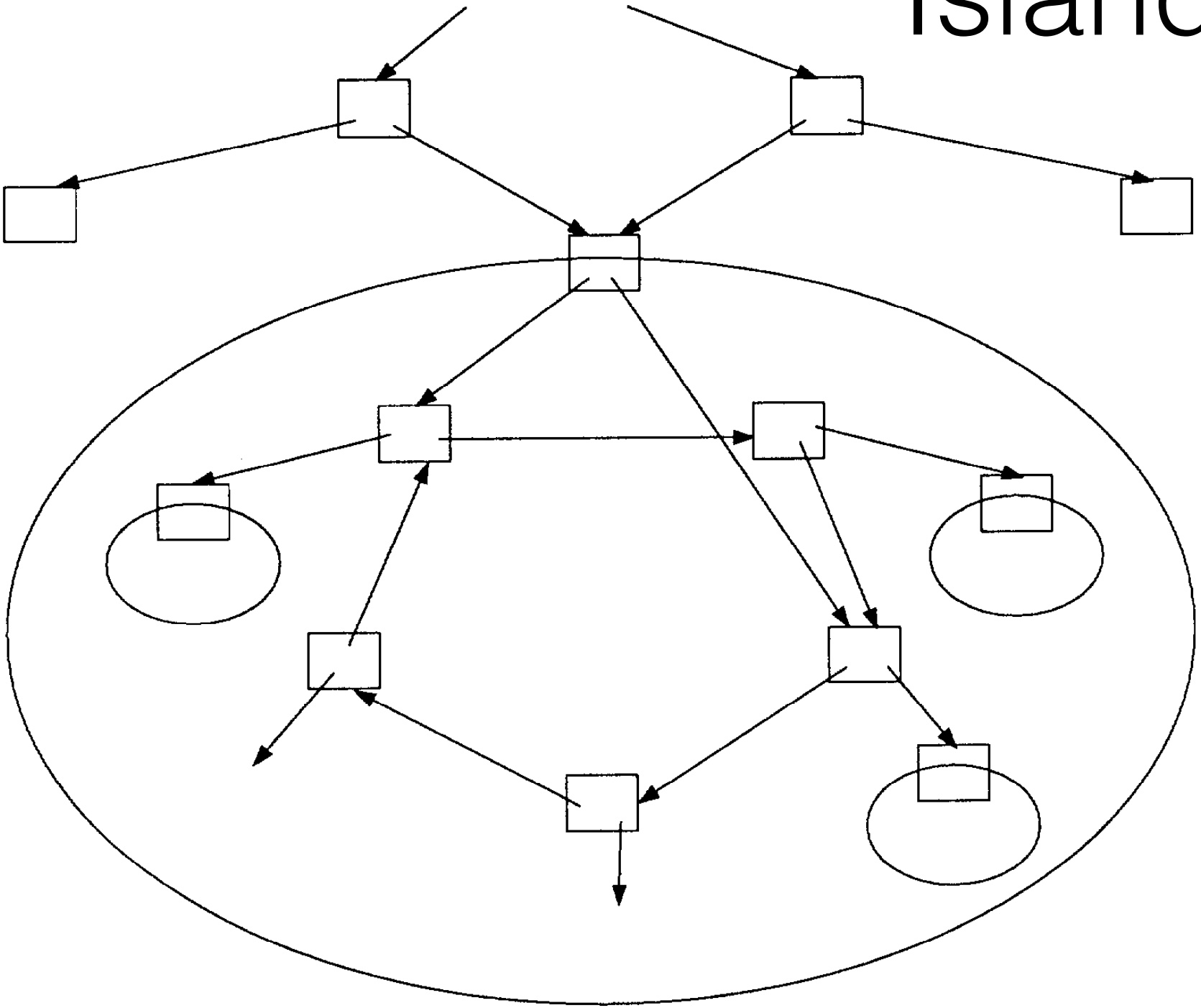
...

```
%read MaxAbs: aComplex%read :%read
    ((self Abs) > (aComplex Abs))
        ifTrue: [↑self]
        ifFalse: [↑aComplex]

%read Abs :%read
    ↑((x*x) + (y*y)) Sqrt
```

...

```
%read Max: aComplex%read
    ((self Abs) > (aComplex Abs))
        ifTrue: [↑self Copy]
        ifFalse: [↑aComplex Copy]

%read Abs :%read
    ↑((x*x) + (y*y)) Sqrt

%read Copy
    ↑(Complex new) x: x y: y

%read x: newX%read y: newY%read
    x ← newX Copy.
    y ← newY Copy
```

Islands

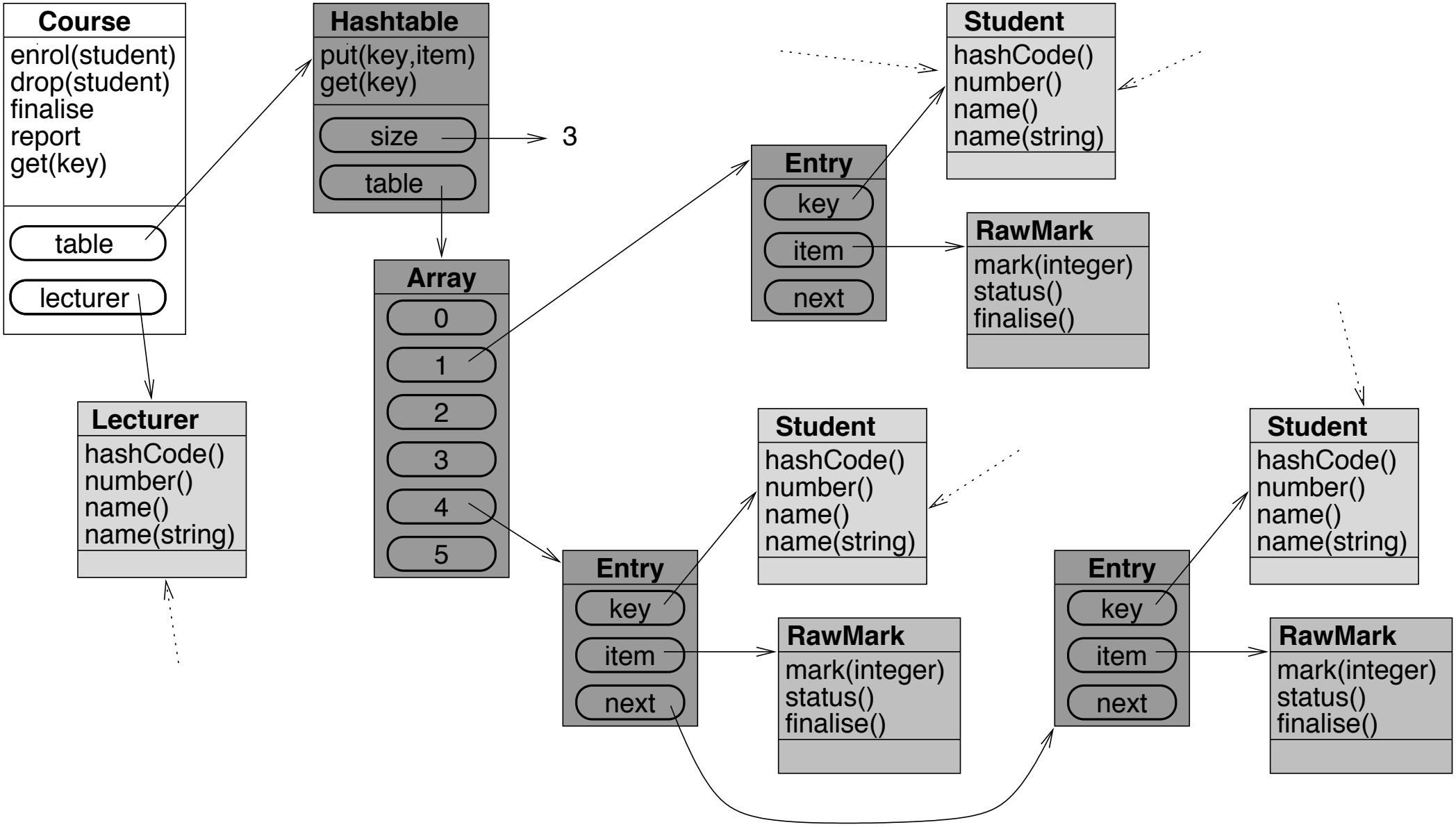


Flexible Alias Protection

James Noble[1], Jan Vitek[2], and John Potter[1]

[1] Microsoft Research Institute, Macquarie University, Sydney
kjx,potter@mri.mq.edu.au
[2] Object Systems Group, Université de Genève, Geneva.
Jan.Vitek@cui.unige.ch

ECOOP 98


Course
enrol(student)
drop(student)
finalise
report
get(key)
table
lecturer
Hashtable
put(key,item)
get(key)
size
3
table
Array
0
1
2
3
4
5
Entry
key
item
next
Student
hashCode()
number()
name()
name(string)
RawMark
mark(integer)
status()
finalise()
Lecturer
hashCode()
number()
name()
name(string)
Entry
key
item
next
Student
hashCode()
number()
name()
name(string)
RawMark
mark(integer)
status()
finalise()
Entry
key
item
next
Student
hashCode()
number()
name()
name(string)
RawMark
mark(integer)
status()
finalise()

```
class  Course<arg s Student> {
   private rep Hashtable<arg s Student, rep RawMark> marks =
         new Hashtable();
      ...
   public void enrol (arg s Student s) {
      rep RawMark r = new RawMark();
      marks.put(s, r);
   }
   public void
      recordMarkFor(arg s Student s, val String workUnit, val int mark) {
      marks.get(s).recordMarkFor(workUnit, mark);
   }
   public void finalReport (arg s Student s) {
      marks.get(s).finalReport();
   }
}
```

Ownership Types for Flexible Alias Protection

David G. Clarke, John M. Potter, James Noble

Microsoft Research Institute, Macquarie University, Sydney, Australia
{clad,potter,kjx}@mri.mq.edu.au

OOPSLA 98

```
class Engine {
   void start() { ...  }
   void stop() { ...  }
}

class Driver { ...  }

class Car {
   rep Engine engine;          // representation
   norep Driver driver;        // not representatio

   Car() {                     // constructor
      // new engine as part of representation
      engine = new rep Engine();
      driver = null;
   }

   rep Engine getEngine() { return engine; }

   void setEngine(rep Engine e) { engine = e; }

   void go() {
      if (driver != null) engine.start();
   }
}
```

```
class Main {
   void main()
   {
      norep Driver bob = new norep Driver();
      norep Car car = new norep Car();

      car.driver = bob;        // (*)
      car.go();
      car.engine.stop();       // fails
      car.getEngine().stop(); // fails
                               //  but not in
      rep Engine e = new rep Engine();
      car.setEngine(e);        // fails
                               //  different
   }
}
```

```
class Pair<m,n> {
   m X fst;
   n Y snd;
}

class Intermediate {
   rep   Pair<rep, norep> pair1;
   norep Pair<rep, norep> pair2;

   rep   Pair<rep, norep> a() { return pair1; }
   norep Pair<rep, norep> b() { return pair2; }
   rep   X x() { return pair1.fst; }
   norep Y y() { return pair1.snd; }

   void updateX() {
      pair1.fst = new rep X();
   }
}
```

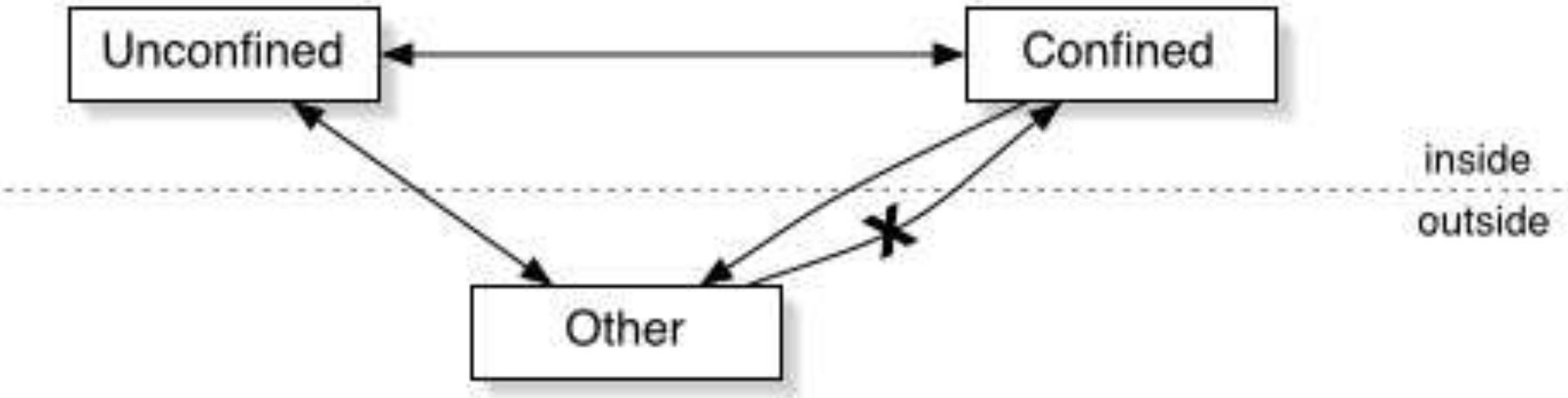
```
class Main {
   norep Intermediate safe;

   void main() {
      rep   Pair<rep, norep> a;
      norep Pair<rep, norep> b;
      rep   X x;
      norep Y y;

      a = safe.a();            //
      b = safe.b();            //
      x = safe.x();            //
      y = safe.y();            //

      safe.updateX();          //
   }
```

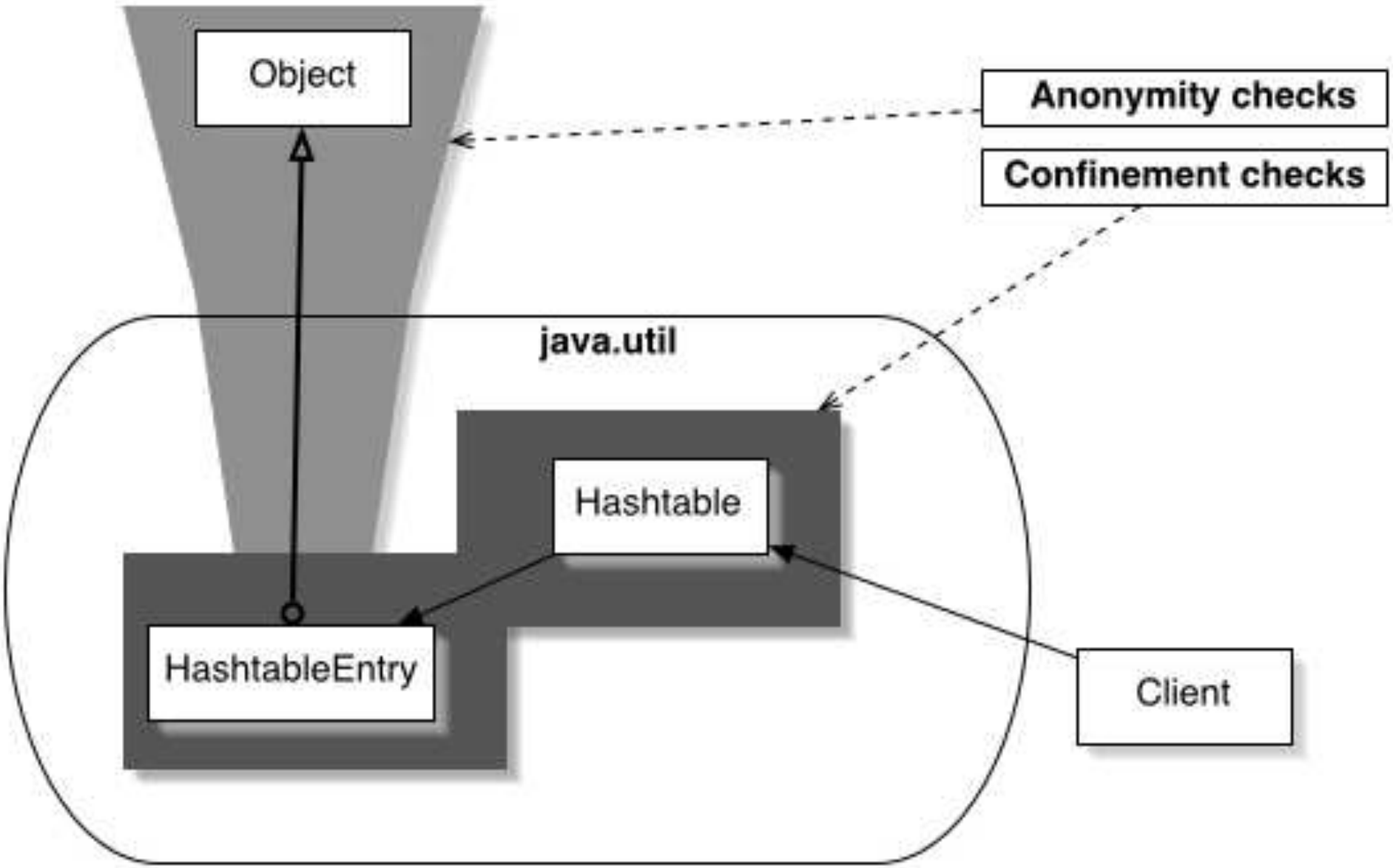
Encapsulating Objects with Confined Types

Christian Grothoff[1] Jens Palsberg[1] Jan Vitek[2]

[1] UCLA Computer Science Department
University of California, Los Angeles
christian@grothoff.org, palsberg@ucla.edu
[2] Department of Computer Science
Purdue University, West Lafayette
jv@cs.purdue.edu

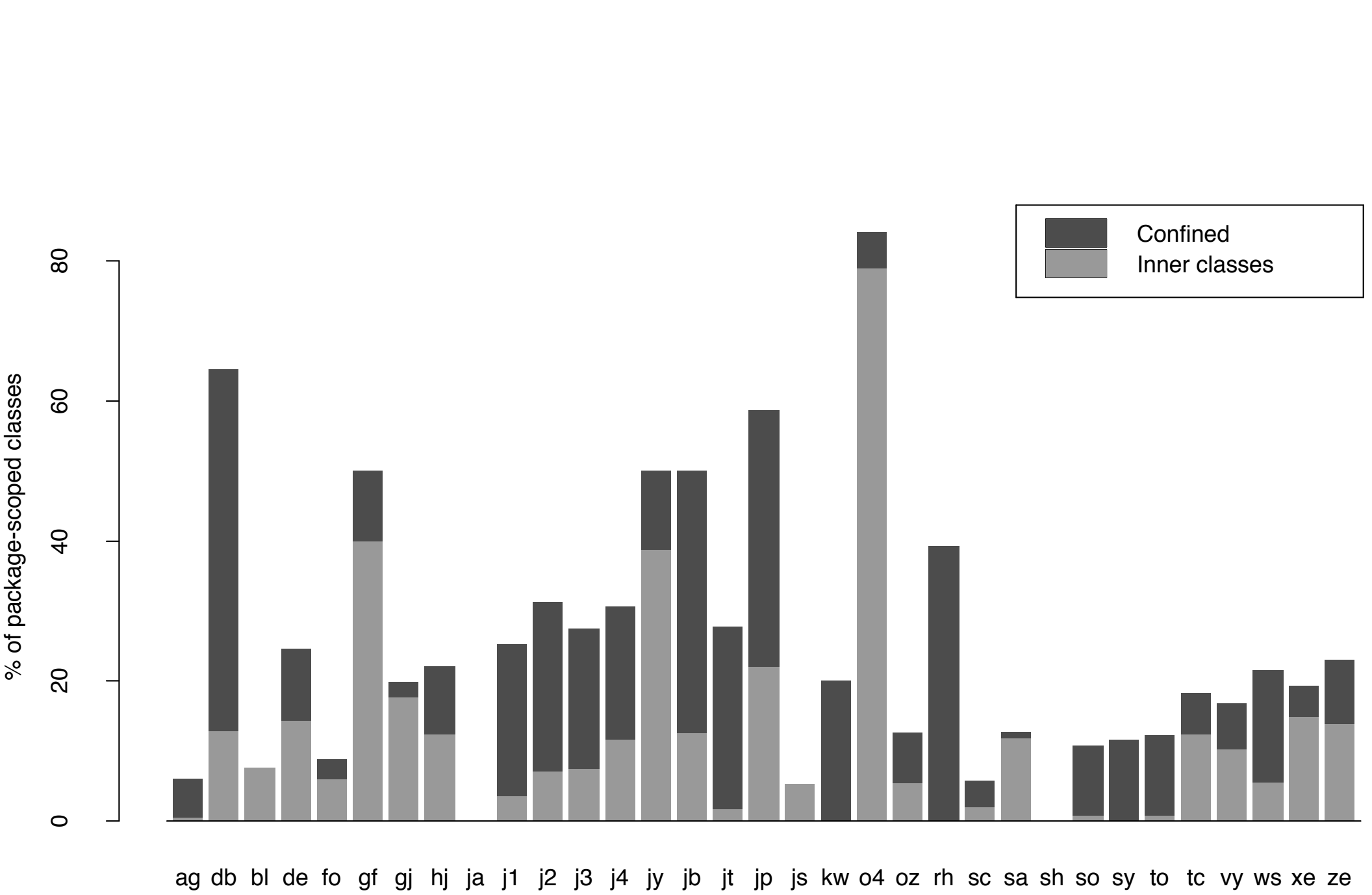
OOPSLA 01


Unconfined
Confined
inside
outside
Other


Object
Anonymity checks
Confinement checks
java.util
Hashtable
HashtableEntry
Client

$\mathcal{A}1$	An anonymous method cannot widen `this` to a non-confined type.
$\mathcal{A}2$	An anonymous method cannot be `native`.
$\mathcal{A}3$	An anonymous method cannot invoke non-anonymous methods on `this`.

$\mathcal{C}1$	All methods invoked on a confined type must be anonymous.
$\mathcal{C}2$	A confined type cannot be public.
$\mathcal{C}3$	A confined type cannot appear in the type of a public (or protected) field or the return type of a public (or protected) method of a non-confined type.
$\mathcal{C}4$	Subtypes of a confined type must be confined.
$\mathcal{C}5$	A confined type cannot be widened to a non-confined type.


Confined
Inner classes
% of package-scoped classes
0
20
40
60
80
ag db bl de fo gf gj hj ja j1 j2 j3 j4 jy jb jt jp js kw o4 oz rh sc sa sh so sy to tc vy ws xe ze

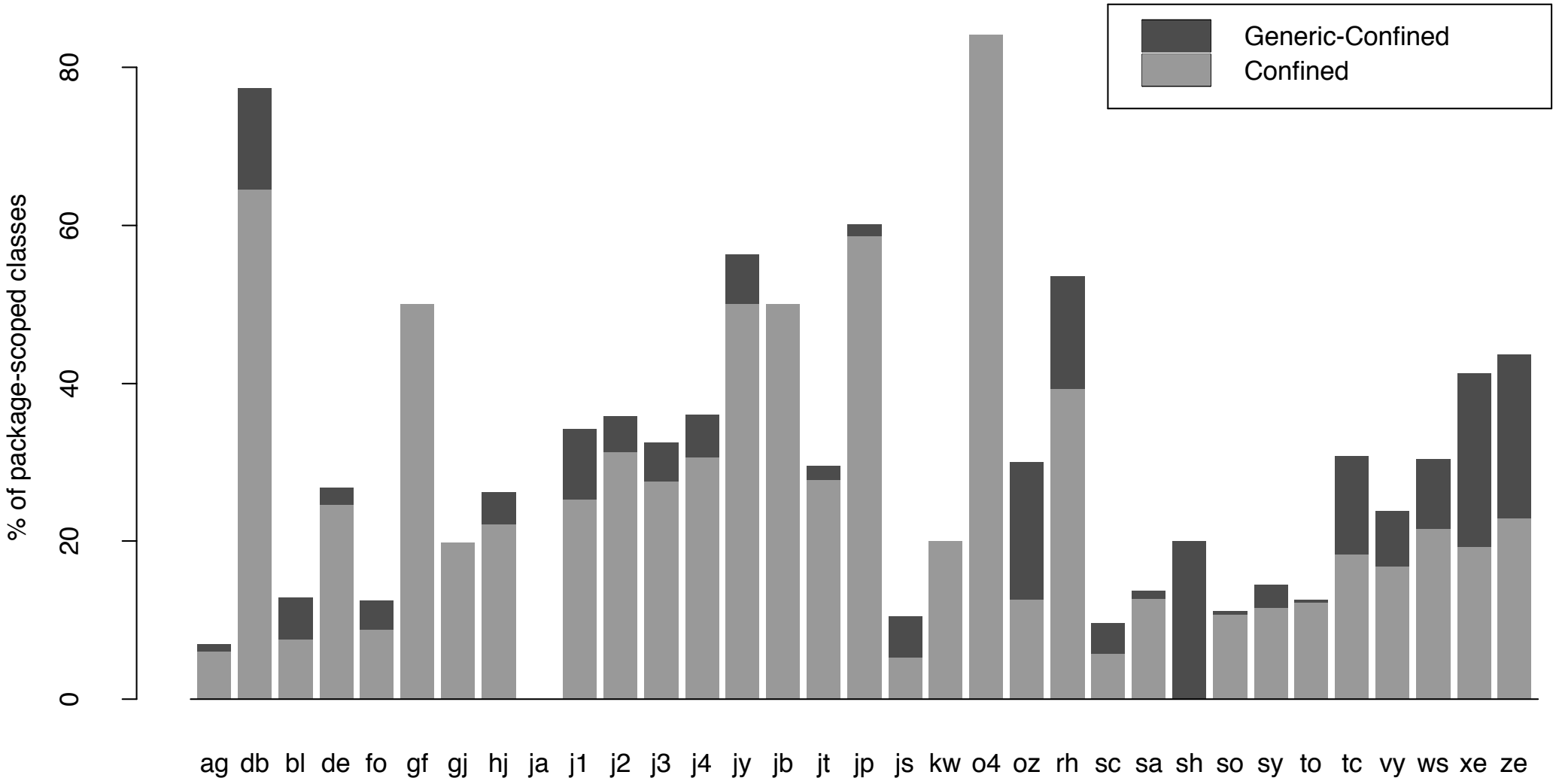


Fig. 15. Generic-confined types.

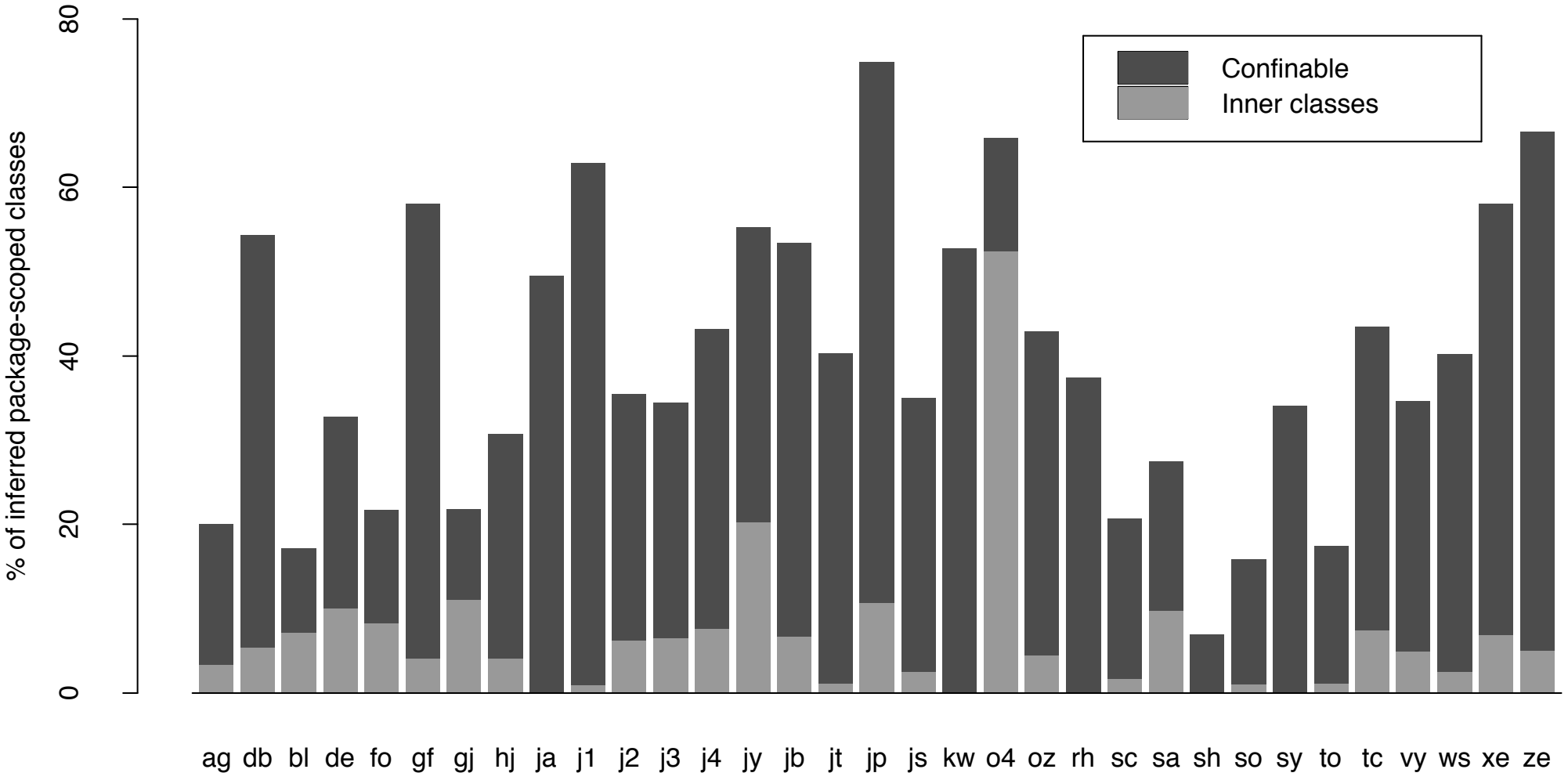


Fig. 16. Confinable types.

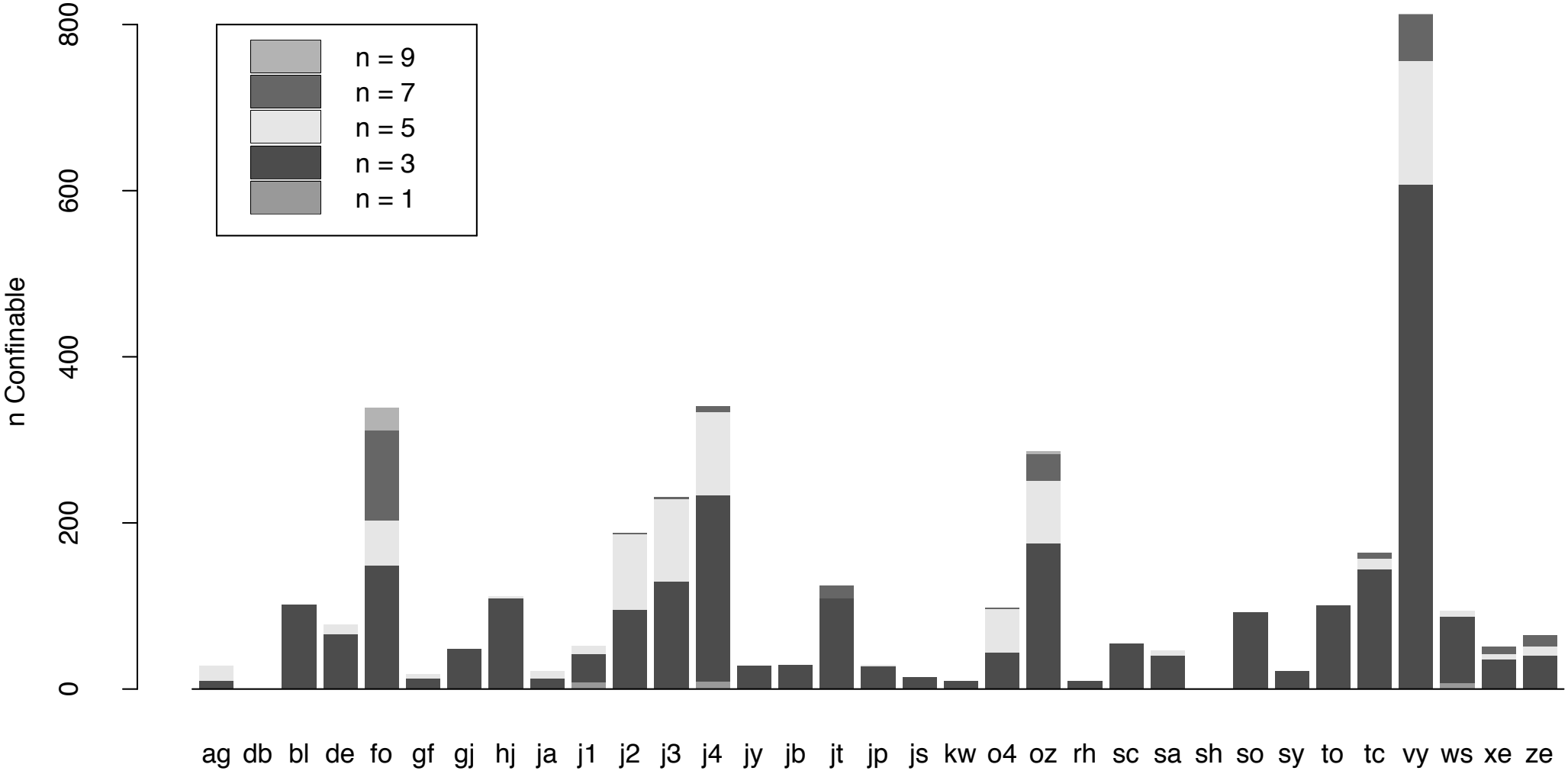
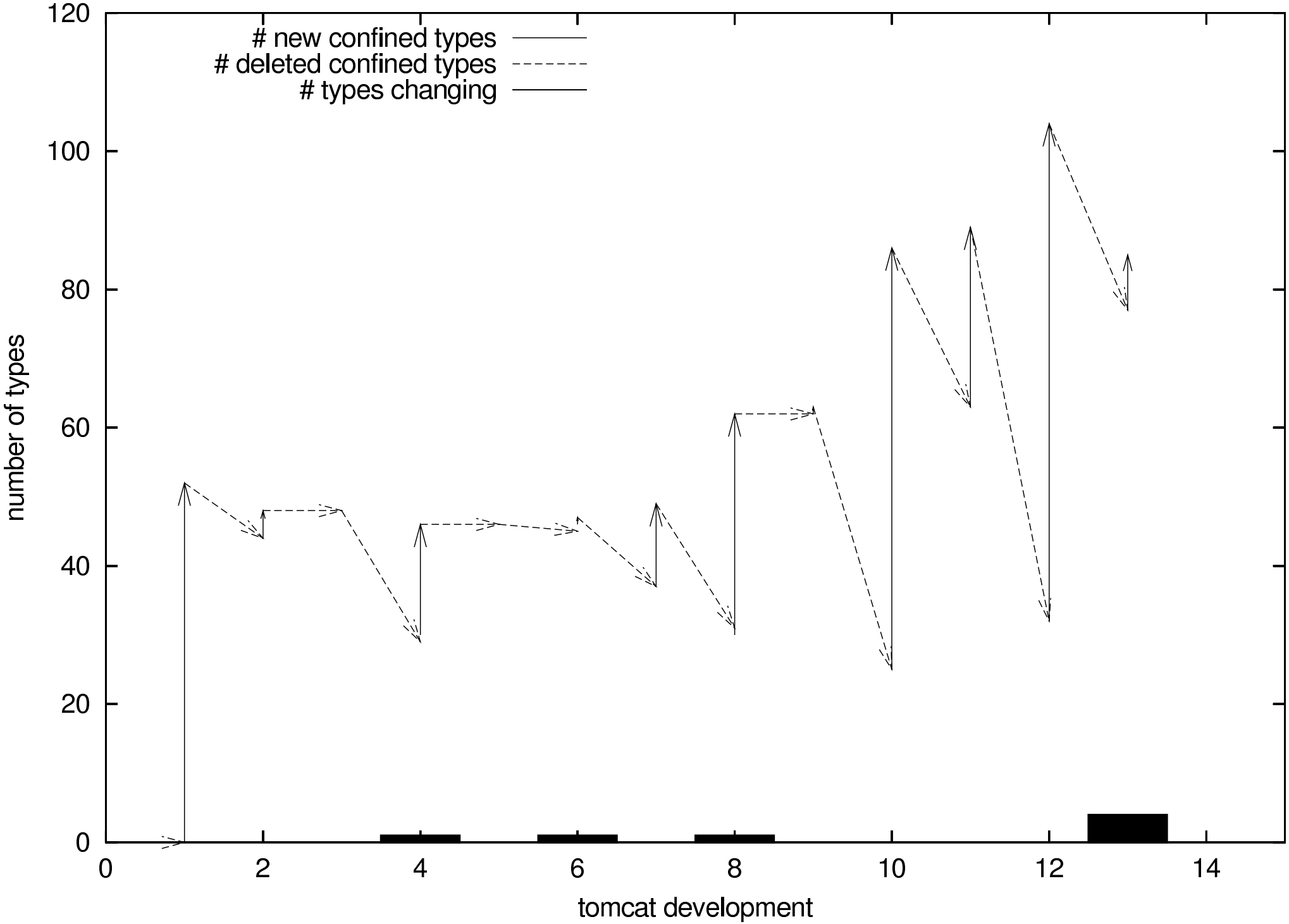


Fig. 18. Confinement with hierarchical packages.


new confined types
deleted confined types
types changing
number of types
tomcat development
0
20
40
60
80
100
120
0
2
4
6
8
10
12
14