Lisaac

Efficient compilation strategy for object-oriented languages under the closed-world assumption

Benoît Sonntag – benoit.sonntag@lisaac.org

Object Prototype

`http://www.lisaac.org`

History : Lisaac for IsaacOOS Language

In the past. . .

C language

$\Downarrow$

Unix system

The futur. . .

Lisaac

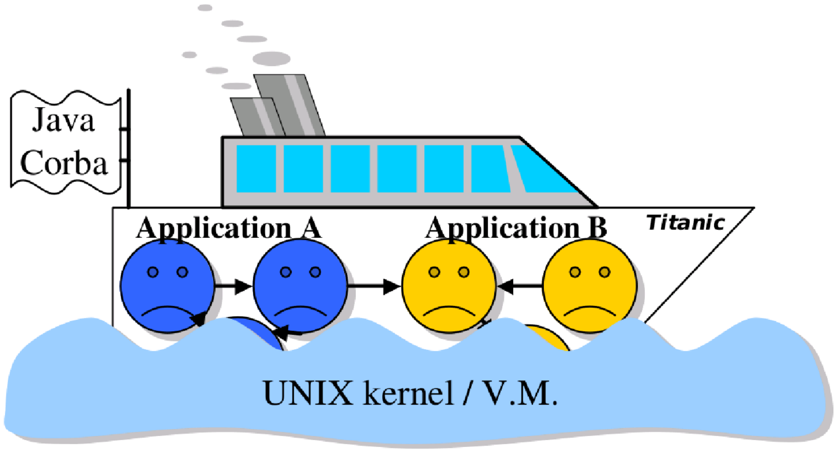
Prototype based Object Oriented Language

$\Downarrow$

IsaacOOS

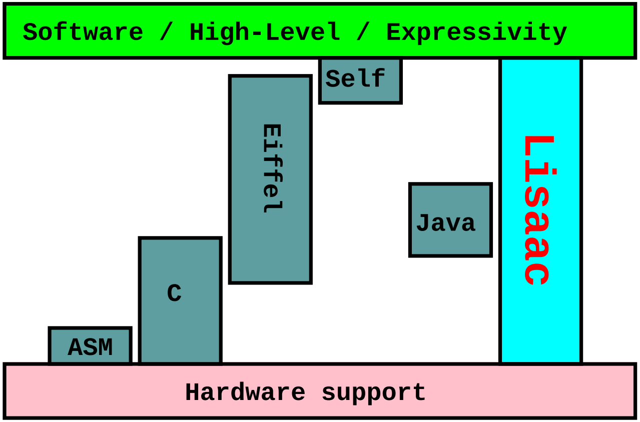
Prototype Object Operating System

Let them sink in a bigger box ?



High-level *vs* Hardware

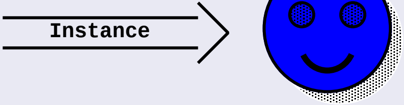
Object Oriented for Hardware



Class *vs* Prototype (1/3)

Class

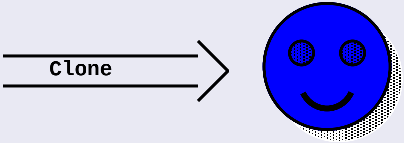
`1 Squeleton`
`(=class)`



`1 Object`

Prototype

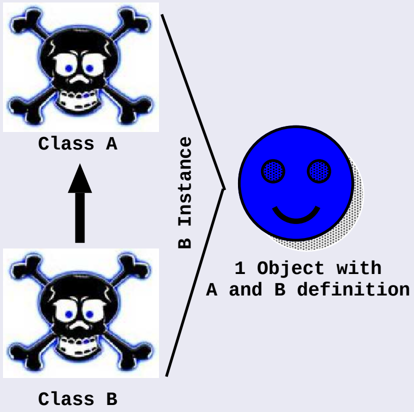
`1 Object prototype`
`(=the One)`



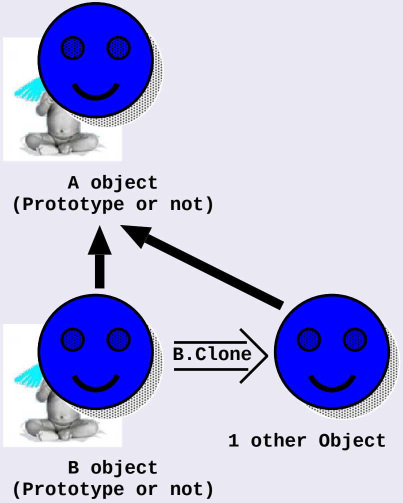
`1 other Object`

Class *vs* Prototype (2/3)

Class

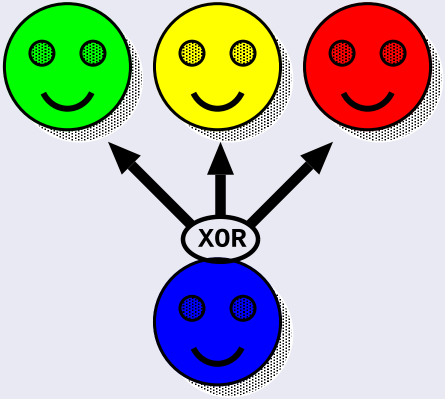


Prototype



Class *vs* Prototype (3/3)

Dynamic inheritance



Example : Hello world!

hello.li

```
Section Header
  + name := HELLO;
Section Public
  - main <-
  (
    (1+2).print;
    'A'.print;
    ``Hello world !\n''.print;
  );
```

Command line : `lisaac hello.li`

Executable result : `hello` *(ou* `hello.exe` *for windows)*

Slot identifier

```
- qsort tab:COLLECTION from low:INTEGER to high:INTEGER <-
( + i,j:INTEGER;
  + x,y:OBJECT;
  i := low;
  j := high;
  x := tab.item ((i + j)>> 1);
  { ...
    (i <= j).if {
      tab.swap j and i;
      ...
    };
  }.do_while {i <= j};
  (low < j).if { qsort tab from low to j; };
  (i < high).if { qsort tab from i to high; };
);
```

Slot identifier

```
– qsort tab:COLLECTION from low:INTEGER to high:INTEGER ←
( + i,j:INTEGER;
  + x,y:OBJECT;
  i := low;
  j := high;
  x := tab.item ((i + j)>> 1);
  { ...
    (i <= j).if {
      tab.swap j and i;
      ...
    };
  }.do_while {i <= j};
  (low < j).if { qsort tab from low to j; };
  (i < high).if { qsort tab from i to high; };
);
```

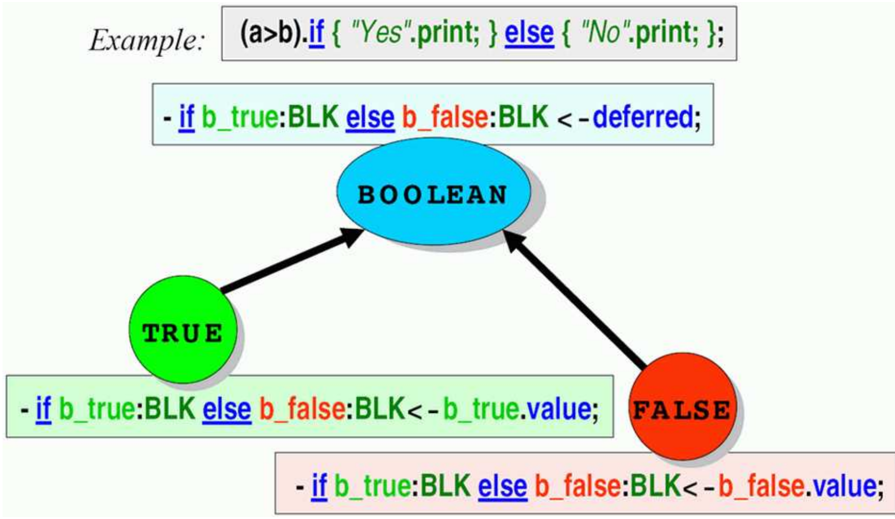
Slot identifier : if

```
- qsort tab:COLLECTION from low:INTEGER to high:INTEGER <-
( + i,j:INTEGER;
  + x,y:OBJECT;
  i := low;
  j := high;
  x := tab.item ((i + j)>> 1);
  { ...
    (i <= j).if {
      tab.textcolorblueswap j and i;
      ...
    };
  }.do_while {i <= j};
  (low < j).if { qsort tab from low to j; };
  (i < high).if { qsort tab from i to high; };
);
```

Slot identifier : loop

```
- qsort tab:COLLECTION from low:INTEGER to high:INTEGER <-
( + i,j:INTEGER;
  + x,y:OBJECT;
  i := low;
  j := high;
  x := tab.item ((i + j)>> 1);
  { ...
    (i <= j).if {
      tab.swap j and i;
      ...
    };
  }.do_while {i <= j};
  (low < j).if { qsort tab from low to j; };
  (i < high).if { qsort tab from i to high; };
);
```

If then else


Example:

```
(a>b).if { "Yes".print; } else { "No".print; };
```

```
- if b_true:BLK else b_false:BLK <-deferred;
```

```
- if b_true:BLK else b_false:BLK<-b_true.value;
```

```
- if b_true:BLK else b_false:BLK<-b_false.value;
```

Assignment : code

Example

```
- color (r,g,b:INTEGER) <-
(
  true_color:=r<<16|g<<8|b;
);
...
(
  color <- (
    gray_color := (r+g+b)/3;
  );
);
```

Inheritance : Dynamic once compute parent

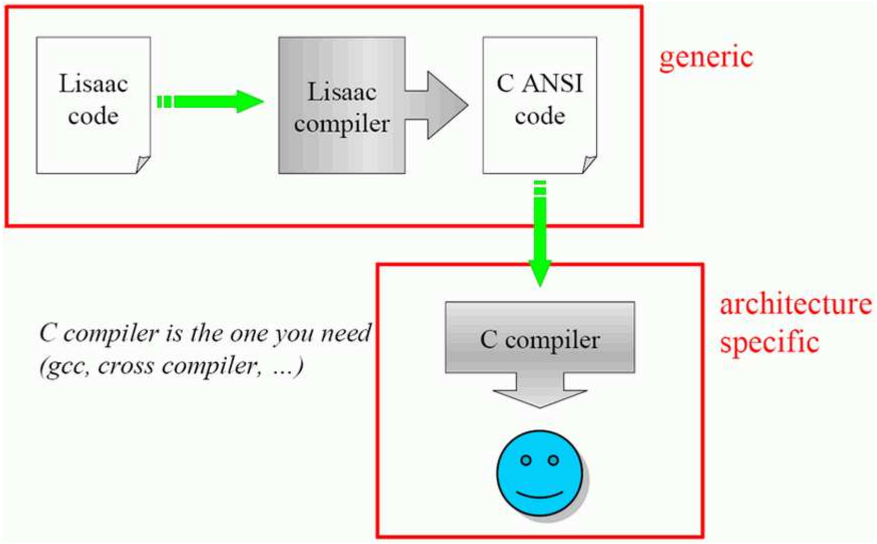
Once execution dynamic parent evaluation

```
Section Inherit
  + parent:OBJECT <-
  ( + result:OBJECT;
    ...// compute my parent
    parent := result // my parent is a data now!!!
  );
```

Note

- The first lookup, the parent is dynamically defined
- The next lookup, the parent is a simple data value

Multi-platform compiler



Global analysis

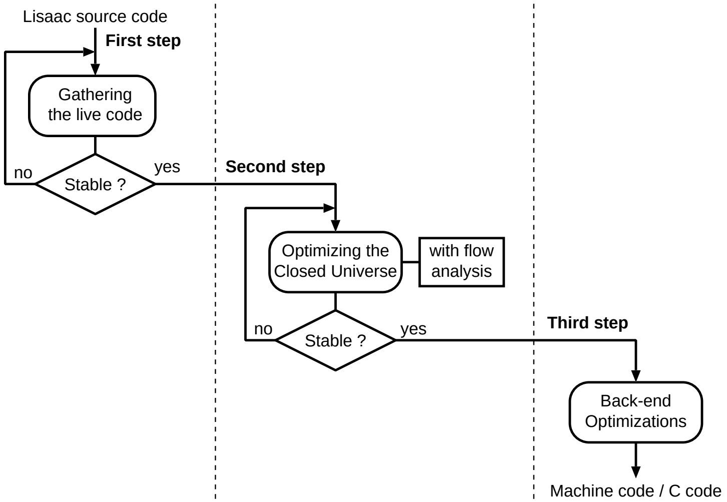
Java, C++ : Classic technical

Virtual Function Table (VFT)
$\Rightarrow$ Pointer of function
$\Rightarrow$ Indirect call
$\Rightarrow$ **No optimization!**

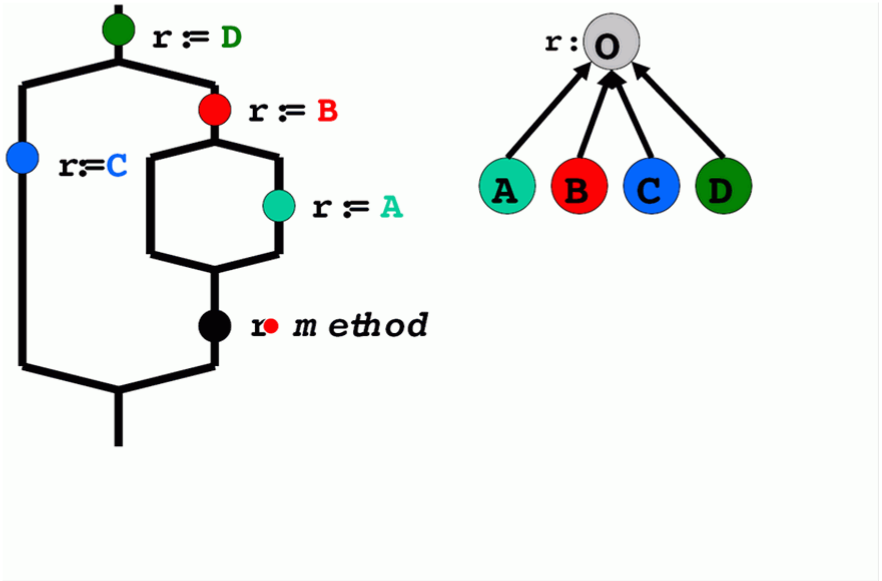
Lisaac : Global analysis

Transitive closure
$\Rightarrow$ Dispatch Binary Branch (DBB)
$\Rightarrow$ Static call
$\Rightarrow$ **Full optimization!**

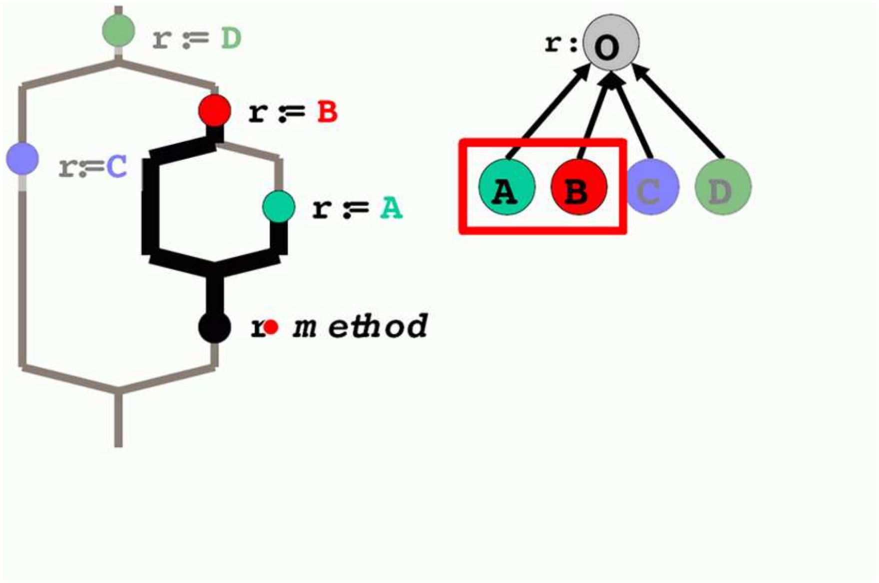
Global overview



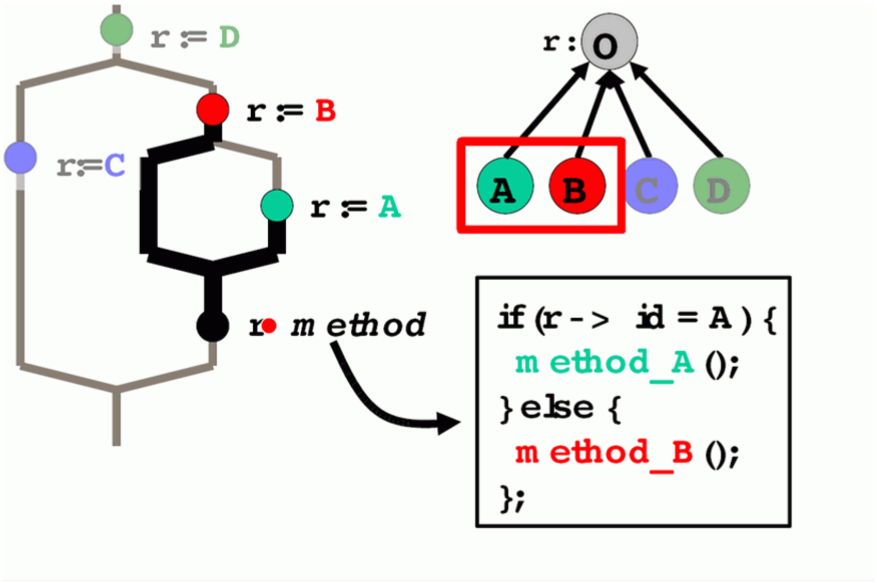
Dispatch Binary Branch (1/4)



Dispatch Binary Branch (2/4)

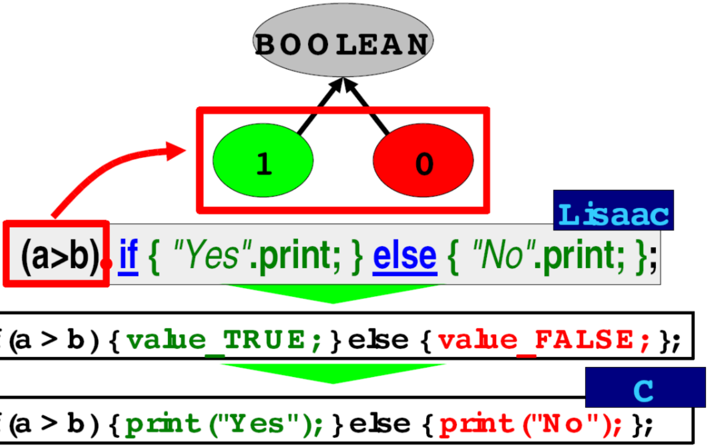


Dispatch Binary Branch (3/4)



```
if (r -> id = A) {
 method_A();
}else {
 method_B();
};
```

DBB: If then else

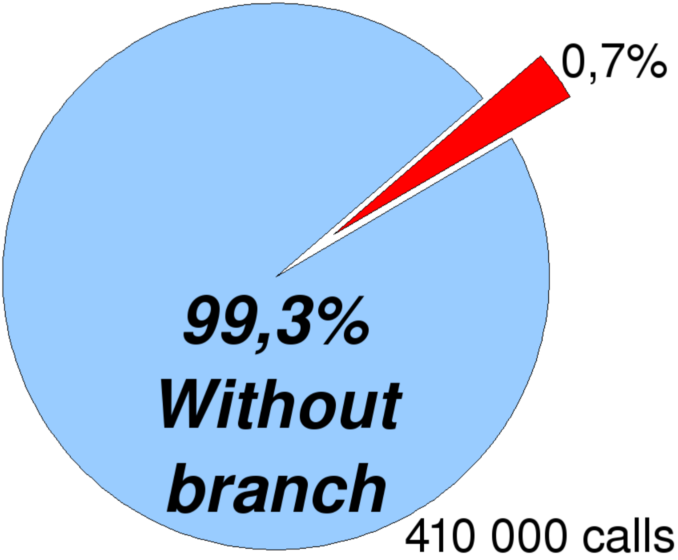


```
(a>b).if { "Yes".print; } else { "No".print; };
```

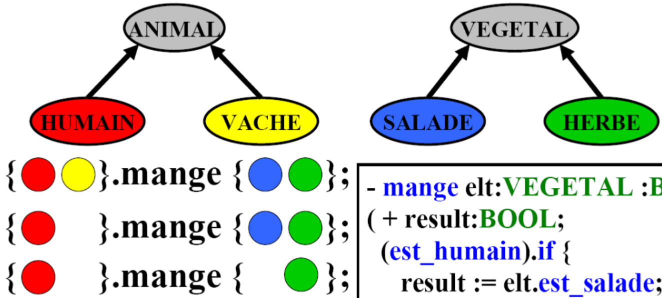
```
if (a > b) {value_TRUE;} else {value_FALSE;};
```

```
if (a > b) {print("Yes");} else {print("No");};
```

Dispatch Binary Branch (4/4)

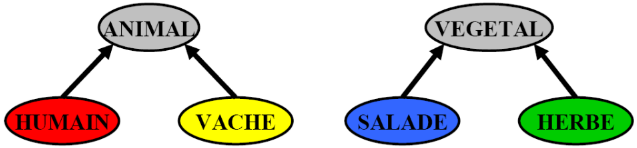


Customization (1/6)



```
- mange elt:VEGETAL :BOOL <-
( + result:BOOL;
  (est_humain).if {
    result := elt.est_salade;
  } else {
    result := TRUE;
  };
  result
);
```

Customization: Call #1 (2/6)



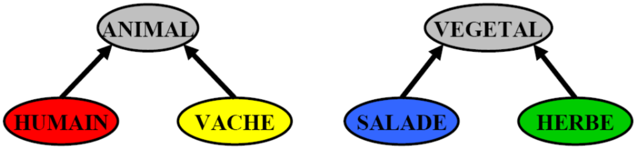
{🔴🟡}.mange {🔵🟢};

🔴.mange_1 {🔵🟢};
(elt.est_salade);

🟡.mange_2 {🔵🟢};
(TRUE);

```
- mange elt:VEGETAL :BOOL <-
( + result:BOOL;
  (est_humain).if {
    result := elt.est_salade;
  } else {
    result := TRUE;
  };
  result
);
```

Customization: Call #2 (3/6)

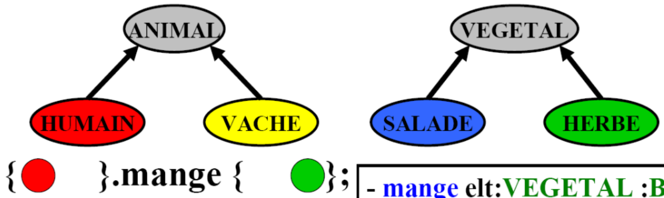


{🔴 }.mange {🔵🟢};

```
🔴.mange_1 {🔵🟢};
( elt.est_salade );
```

```
- mange elt:VEGETAL :BOOL <-
( + result:BOOL;
  (est_humain).if {
    result := elt.est_salade;
  } else {
    result := TRUE;
  };
  result
);
```

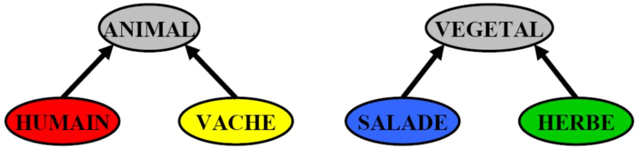
Customization: Call #3 (4/6)



```
🔴.mange_3 { 🟢};
( FALSE );
```

```
- mange elt:VEGETAL :BOOL <-
( + result:BOOL;
  (est_humain).if {
    result := elt.est_salade;
  } else {
    result := TRUE;
  };
  result
);
```

Customization (5/6)



{🔴🟡}.mange {🔵🟢};

{🔴 }.mange {🔵🟢};

{🔴 }.mange { 🟢};

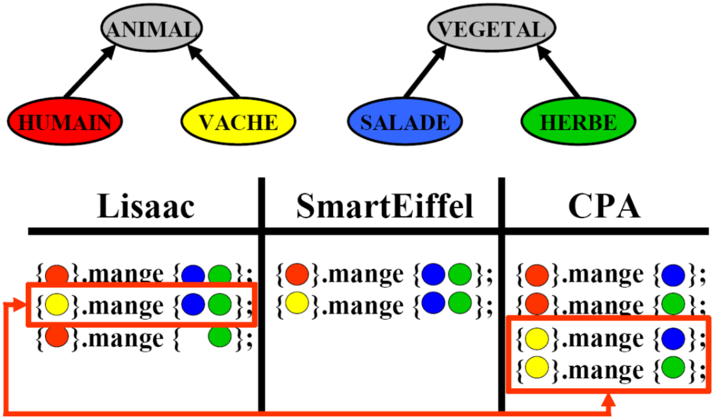
```
mange_1(elt)
( elt.est_salade );
```

```
mange_2( )
( TRUE );
```

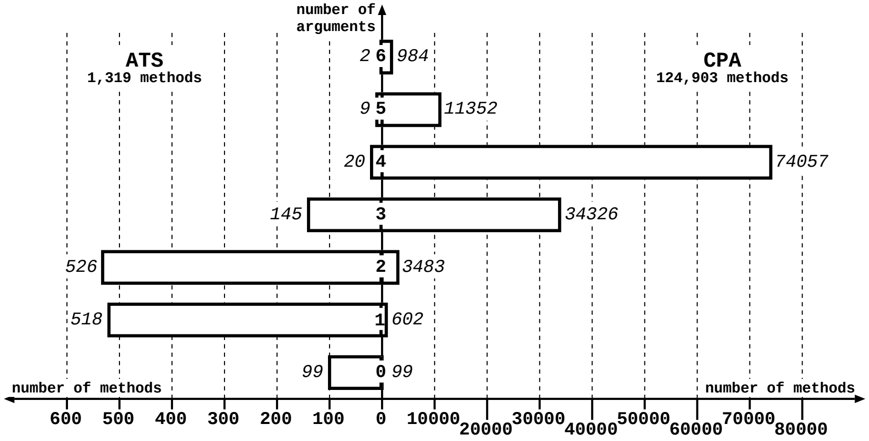
```
mange_3( )
( FALSE );
```

```
- mange elt:VEGETAL :BOOL <-
( + result:BOOL;
  (est_humain).if {
    result := elt.est_salade;
  } else {
    result := TRUE;
  };
  result
);
```

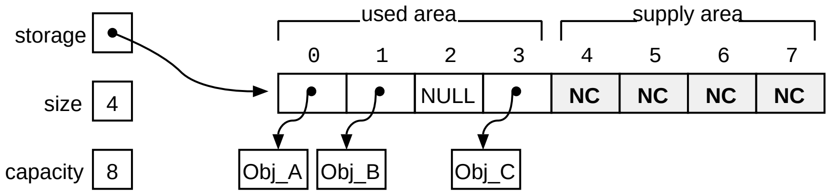
Customization (6/6)



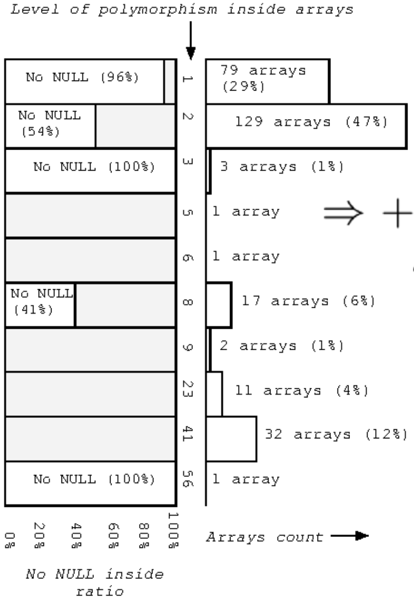
Customization *vs* CPA



Array: Pattern Matching control (1/2)



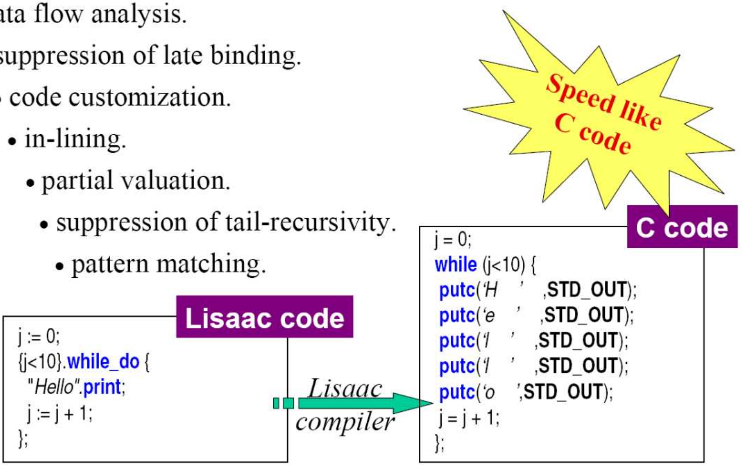
Array: Pattern Matching control (2/2)



$\Rightarrow$ + Optimization GC: 40% off mark

As fast a C language

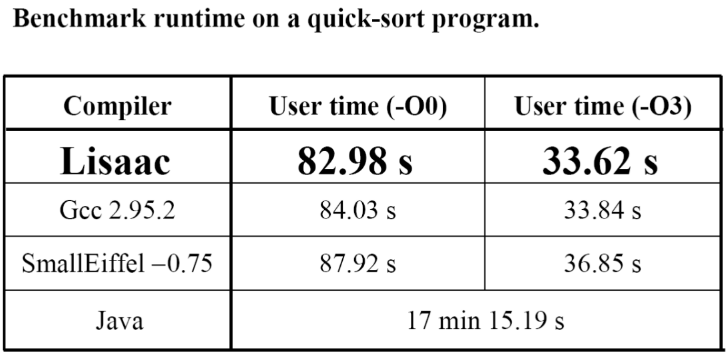
- data flow analysis.
  - suppression of late binding.
    - code customization.
      - in-lining.
        - partial valuation.
          - suppression of tail-recursivity.
            - pattern matching.



```
j := 0;
{j<10}.while_do {
  "Hello".print;
  j := j + 1;
};
```

```
j = 0;
while (j<10) {
 putc('H'   ,STD_OUT);
 putc('e'   ,STD_OUT);
 putc('l'   ,STD_OUT);
 putc('l'   ,STD_OUT);
 putc('o'   ,STD_OUT);
 j = j + 1;
};
```

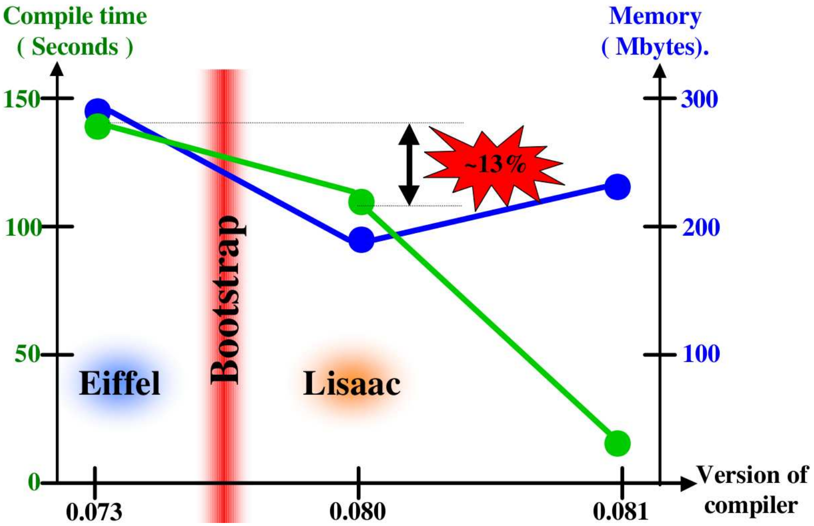
Tiny test: Quicksort



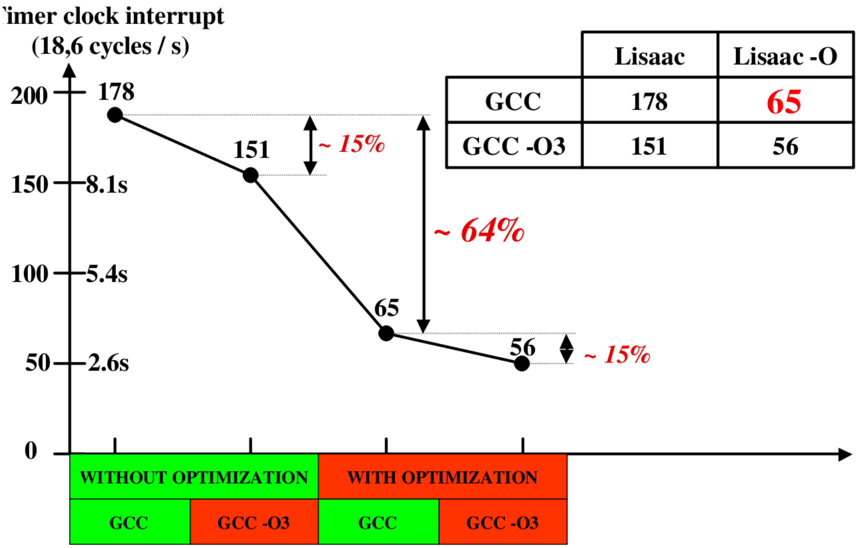
Benchmark runtime on a quick-sort program.

Compiler	User time (-O0)	User time (-O3)
Lisaac	**82.98 s**	**33.62 s**
Gcc 2.95.2	84.03 s	33.84 s
SmallEiffel –0.75	87.92 s	36.85 s
Java	17 min 15.19 s	

Compiler / Bootstrap

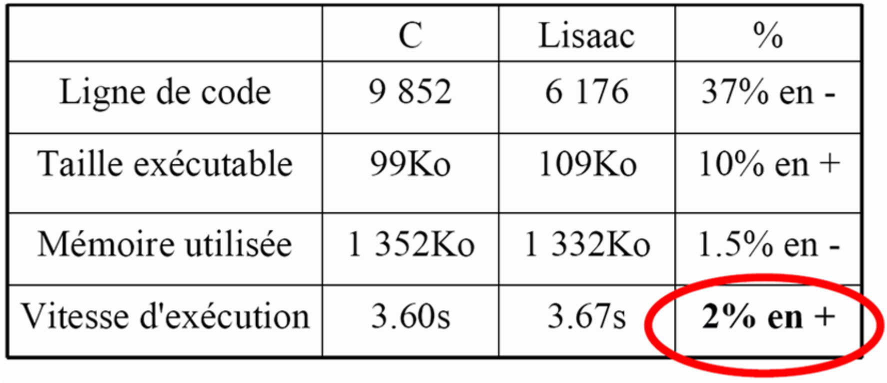


Isaac OS benchmark



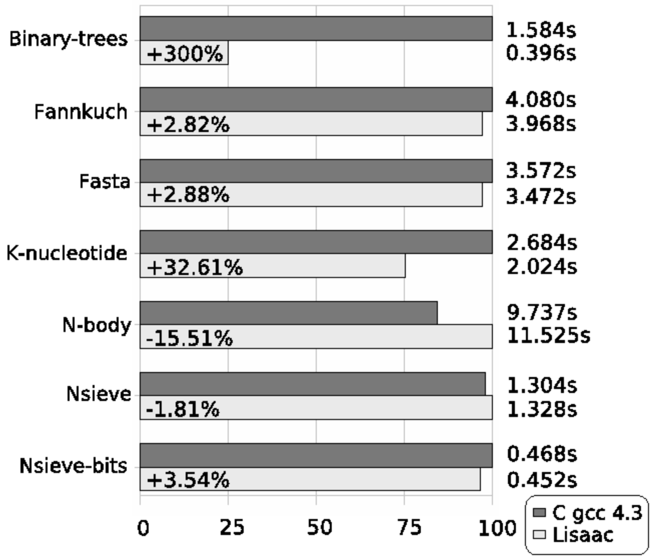
	Lisaac	Lisaac -O
GCC	178	65
GCC -O3	151	56

MPEG2 benchmark

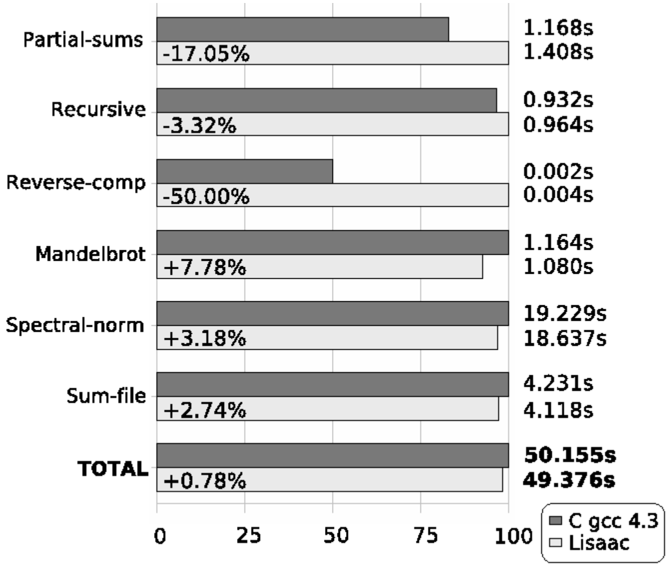


	C	Lisaac	%
Ligne de code	9 852	6 176	37% en -
Taille exécutable	99Ko	109Ko	10% en +
Mémoire utilisée	1 352Ko	1 332Ko	1.5% en -
Vitesse d'exécution	3.60s	3.67s	**2% en +**

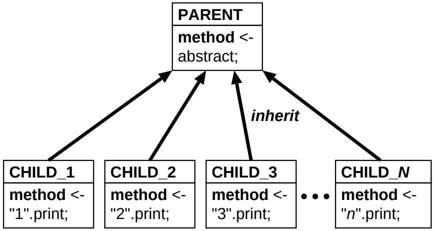
Shootout benchmark (1/2)



Shootout benchmark (2/2)



Horizontal inheritance

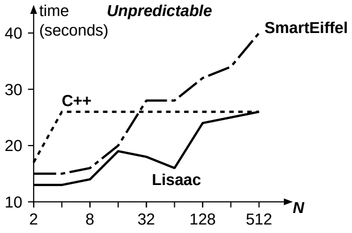


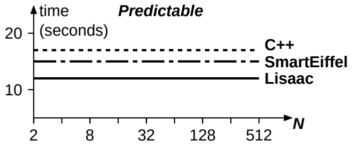
Unpredictable MAIN

```
main <-
1_000_000_000.times {
  array.item(random).method;
};
```

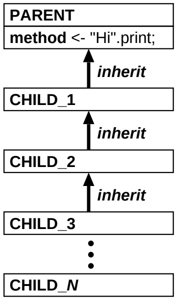
Predictable MAIN

```
main <-
1_000_000_000.times {
  array.item(random & 1).method;
};
```





Vertical inheritance

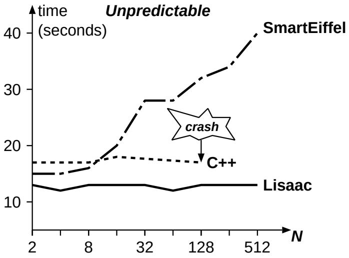


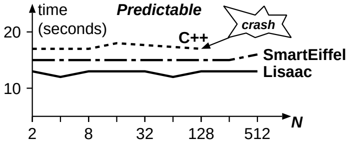
***Unpredictable* MAIN**

```
main <-
1_000_000_000.times {
  array.item(random).method;
};
```

***Predictable* MAIN**

```
main <-
1_000_000_000.times {
  array.item(random & 1).method;
};
```





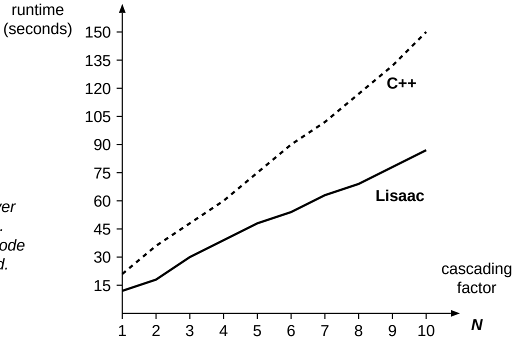
Auto-cascading

Automatic cascading-calls detection.

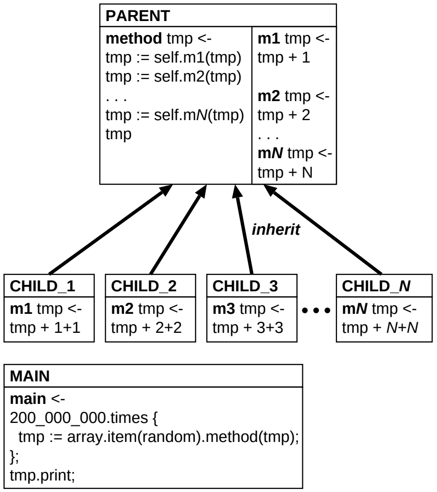
MAIN

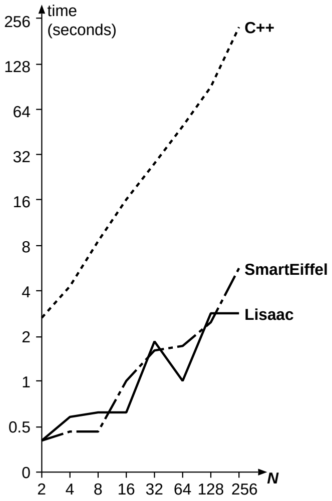
```
main <-
500_000_000.times {
  receiver := array.item(random);
  receiver.method1;
  receiver.method2;
  receiver.method3;
  ...
  ...
  receiver.methodN;
};
```

Same receiver for all calls. Dispatching code is factorized.

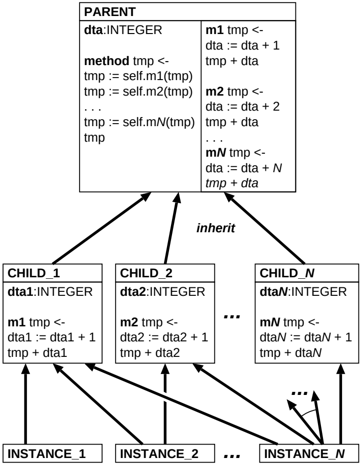


Call on self *(this)*

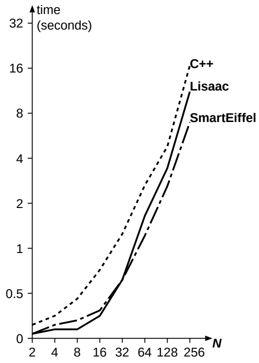




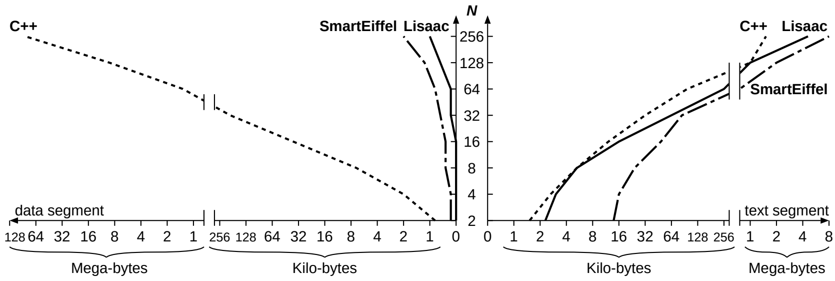
Multiple inheritance (1/2)


```
PARENT
dta:INTEGER

method tmp <-
tmp := self.m1(tmp)
tmp := self.m2(tmp)
. . .
tmp := self.mN(tmp)
tmp

m1 tmp <-
dta := dta + 1
tmp + dta

m2 tmp <-
dta := dta + 2
tmp + dta
. . .
mN tmp <-
dta := dta + N
tmp + dta
```
```
CHILD_1
dta1:INTEGER

m1 tmp <-
dta1 := dta1 + 1
tmp + dta1
```
```
CHILD_2
dta2:INTEGER

m2 tmp <-
dta2 := dta2 + 1
tmp + dta2
```
```
CHILD_N
dtaN:INTEGER

mN tmp <-
dtaN := dtaN + 1
tmp + dtaN
```

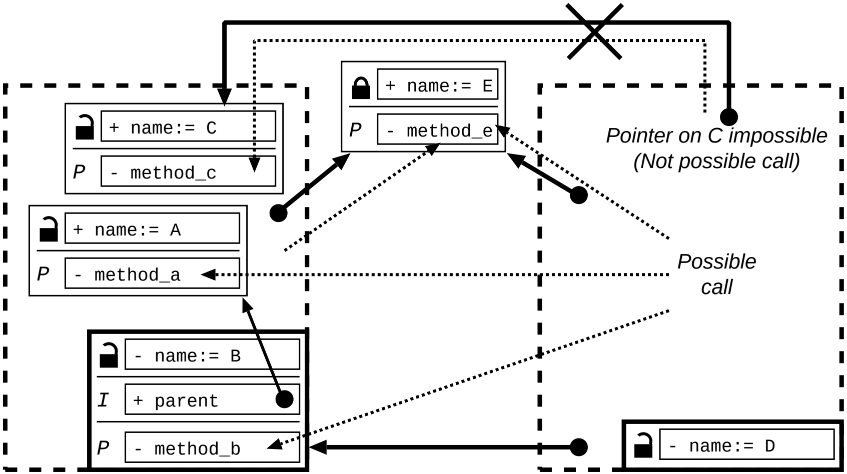
```
MAIN
main <-
1_000_000.times {
  tmp := array.item(random).method(tmp);
};
tmp.print;
```



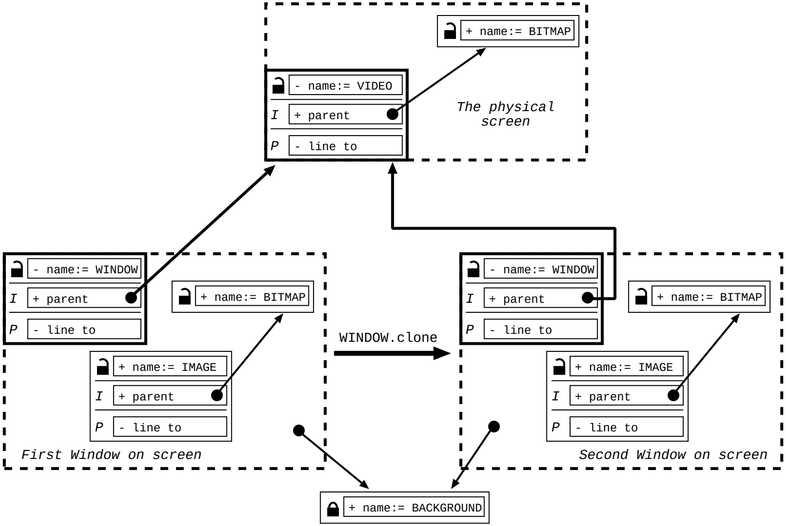
Multiple inheritance (2/2)



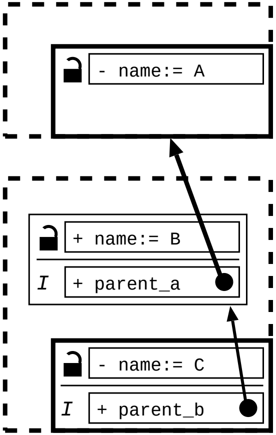
COP : Concurrent Object Prototypes (1/3)



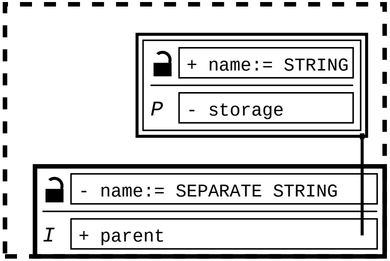
COP : Concurrent Object Prototypes (2/3)



COP : Concurrent Object Prototypes (3/3)



COP : Concurrent Object Prototypes



Question ?

IRC

- Server: `irc.oftc.net`
- Channel: `#isaac`

Information & contacts

- **Wiki** : `http://www.lisaac.org/documentation/wiki`
- **Mailing list** : `lisaac-announce@lists.alioth.debian.org`

Object Prototype

`http://www.lisaac.org`