Lisaac

The power of simplicity at work for operating system

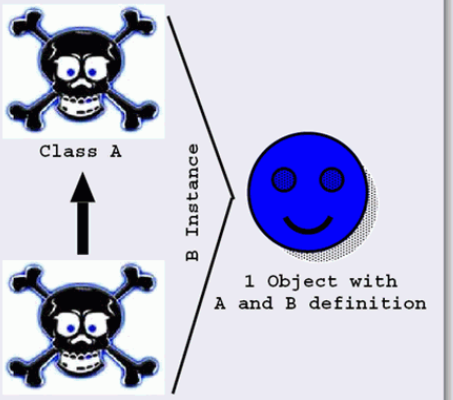
Benoît Sonntag – sonntag@icps.u-strasbg.fr
and the Lisaac team

http://isaacproject.u-strasbg.fr/
ou
http://IsaacOS.com

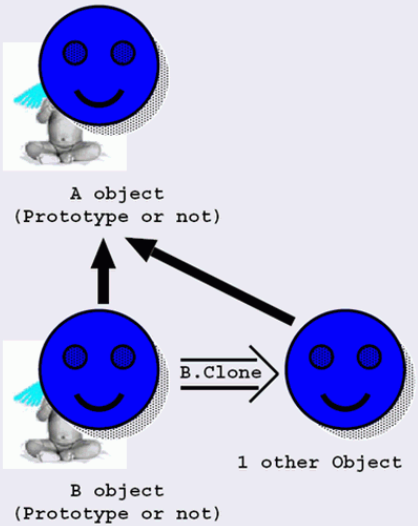
September 27, 2008

Class *vs* Prototype (2/3)

Class

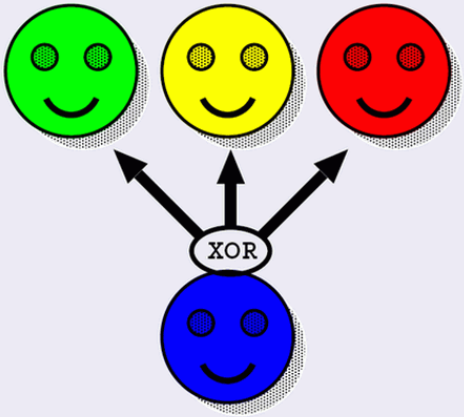


Prototype



Class *vs* Prototype (3/3)

Dynamic inheritance



Inherit Lisaac

- 

Self: Flexibility, simplicity and prototype concept

+

- 

Eiffel: Static type, programming by contract

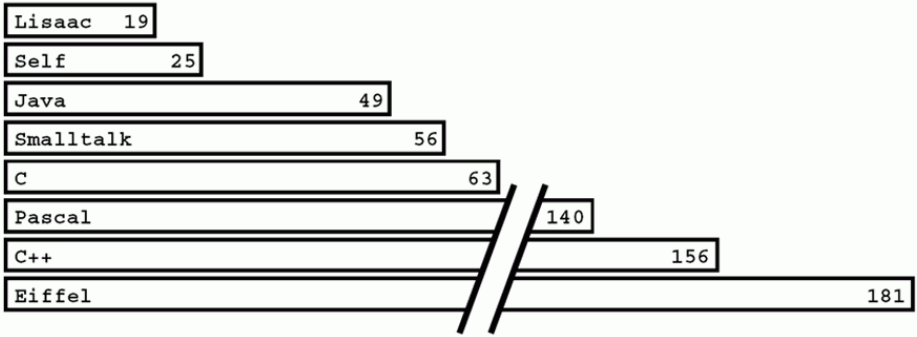
+

- **C**: Interrupt management, memory mapping

=

Lisaac: Full prototype object for hardware

The grammar of Lisaac



Number of gamatical rules

Example : Hello world!

hello.li

```
Section Header
  + name := HELLO;
Section Public
  - main <- 
  (
    ''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; };
);
```

Slot / Unary / Binary

Unary message:

```
- '-':SELF <- zero - Self;
```

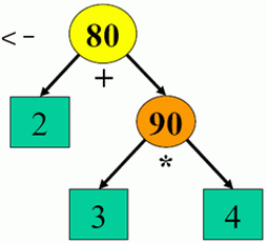
Example: `a:=-3;`

Binary message:

```
- '+' left 80 other:SELF :SELF <-
(Self- -other);

- '*' left 90 other:SELF :SELF <-
( … );
```

Example: `a:= 2 + 3 * 4;`



Partnerships

Core partners

External partners

Assignment : code (3/3)

Example

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

BLOCK : Example (3/4)

Embedded code in object

```
+ display:{(INTEGER,INTEGER); INTEGER};
display := { (x,y:INTEGER); // Vector parameter
             + sum:INTEGER; // One local variable
             x.print;
             ','.print;
             y.print;
             sum := x + y;
             sum // The result block
           };
...
display.value (3,4) .print;
```

BLOCK : Examples (4/4)

For expressions

```
(a != NULL) && {a.value = 3}
```

For conditionals

```
(a > b).if {
  'y'.print;
} else {
  'n'.print;
};
```

For loops

```
{ j := j + 1;
  j.print;
}.do_while {j < 10};
```

For iterations

```
1.to 10 do { j:INTEGER;
  j.print;
};
```

Programming by contract : Require/Ensure (4/5)

Require / ensure on a slot

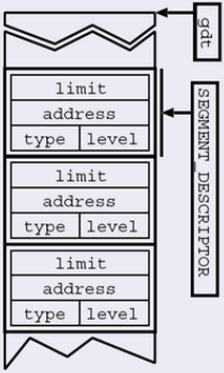
```
- swap idx1:INTEGER with idx2:INTEGER ←
//  Swap item at index 'idx1' with item at index 'idx2'
[ // Require
  ? {valid_index idx1};
  ? {valid_index idx2};
]
( + tmp:E; // Body slot
  tmp := item idx1;
  put (item idx2) to idx1;   put tmp to idx2;
)
[ // Ensure
  ? {item idx1 = Old item idx2};
  ? {item idx2 = Old item idx1};
];
```

Memory Mapping : hardware structure (1/3)

Example for Global Descriptor Table on Intel x86

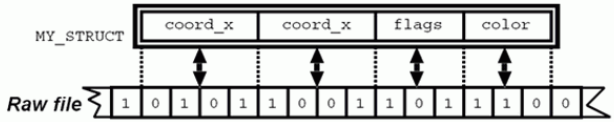
```
Section Header
  + name := SEGMENT_DESCRIPTOR;
Section Mapping
  + limit:UINTEGER_32;
  + address:UINTEGER_32;
  + type:UINTEGER_16;
  + level:UINTEGER_16;

...
  - gdt:NATIVE_ARRAY Expanded SEGMENT_DESCRIPTOR;
```



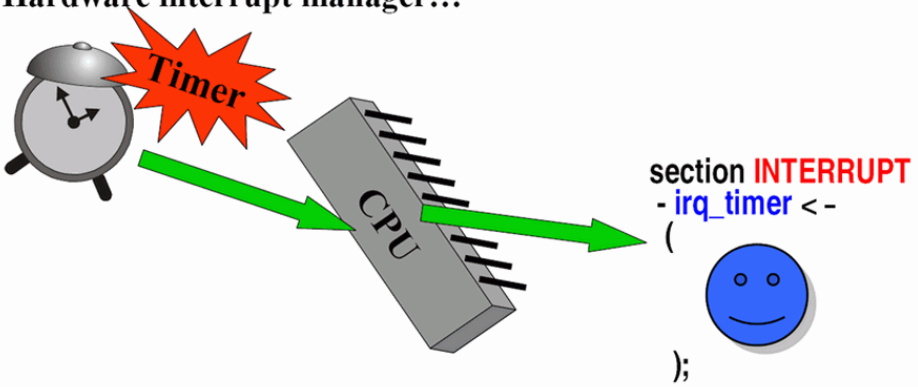
Memory Mapping : binary file structure (2/3)

```
Section Header
  + name := MY_STRUCT;
Section Mapping
  + coord_x:UINTEGER_32;
  + coord_y:UINTEGER_32;
  + flags:UINTEGER_16;
  + color:UINTEGER_16;
Section Public
  - move ← ...
  - set_color ← ...
```



Interrupt object

Hardware interrupt manager…



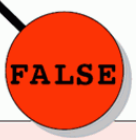
Boolean / if else

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;
```



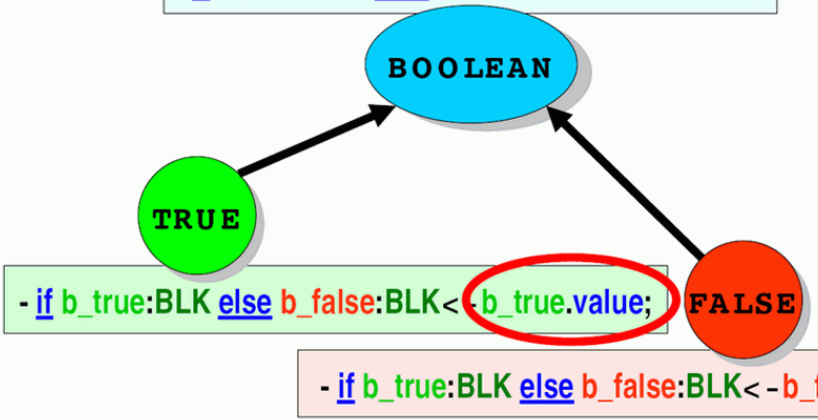

```
- if b_true:BLK else b_false:BLK<- b_false.value;
```

Boolean / if else

Example:

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

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

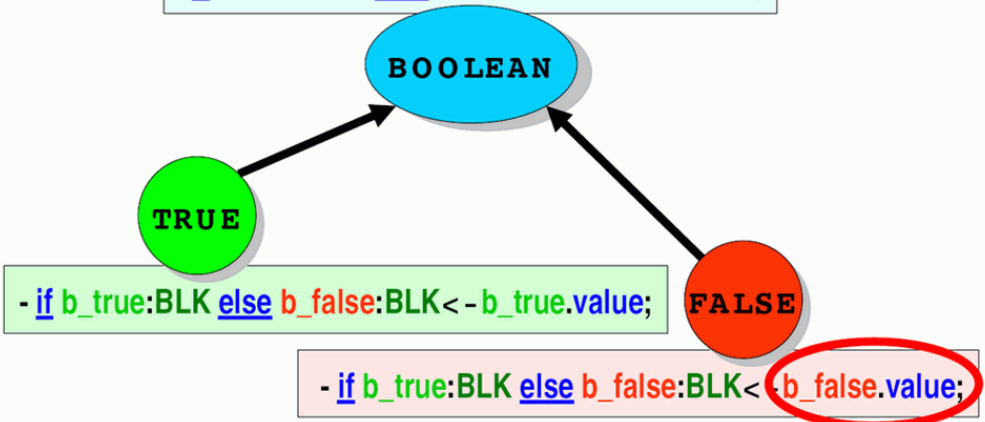


Boolean / if else

Example:

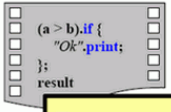
```
(a>b).if { "Yes".print; } else { "No".print; };
```

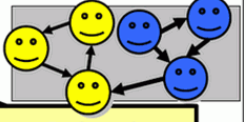
```
- if b_true:BLK else b_false:BLK < - deferred;
```



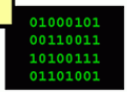
Le Projet Isaac

Technologie à prototype





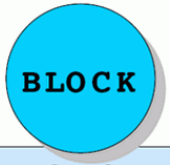
Langage à prototype + compilateur



Système d'exploitation à prototype

Loop definition

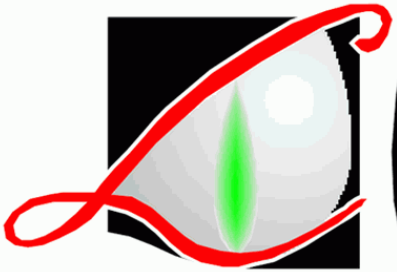
Example: `{a>b}.while_do { b :=b+1; };`




```
- while_do statements_to_repeat:BLOCK < -
  (self.value).if {
     statements_to_repeat.value;
     self.while_do statements_to_repeat;
  };
```

The FAST Compiler

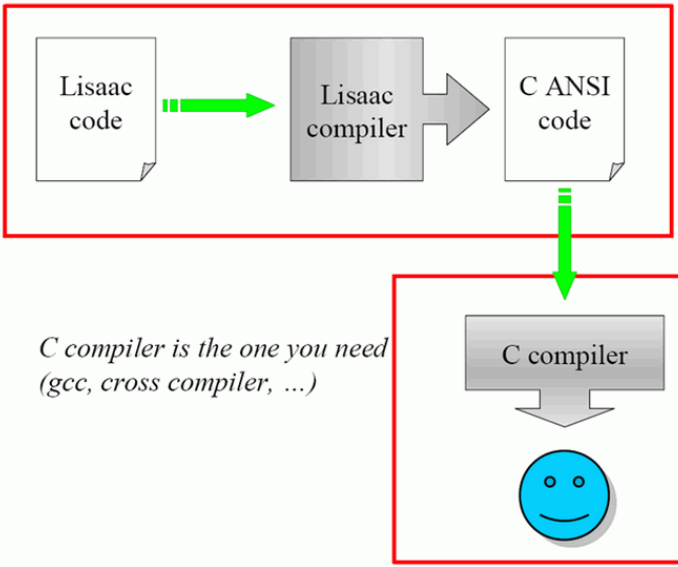
01000101
00110011
10100111
01101001



ISAAC

Prototype Language

Multi-platform compiler



C compiler is the one you need (gcc, cross compiler, ...)

Global analysis

Inter macro-object : *compilation séparée*

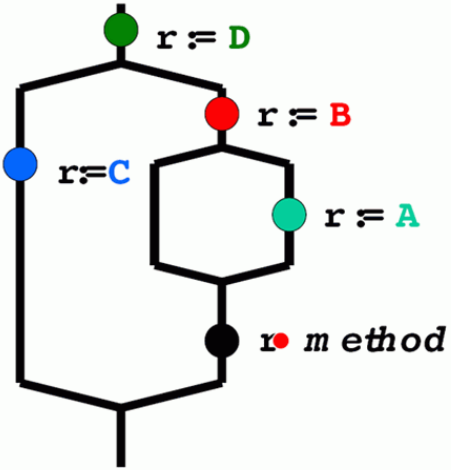
Classic technical (VFT, Lookup, …)

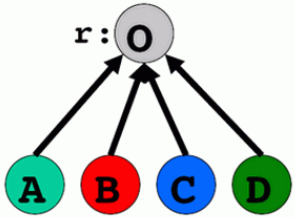
Inter micro-object : *global analysis*

- Recenser les types vivants
- Recenser le code vivant

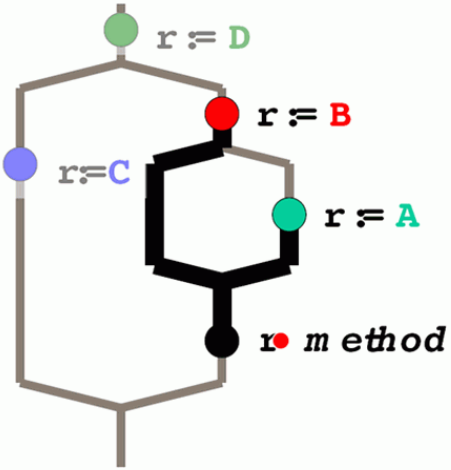
1011
0110

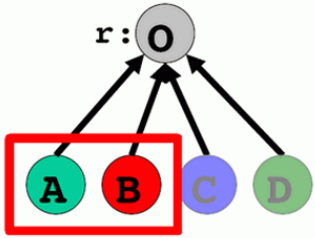
Flow analysis



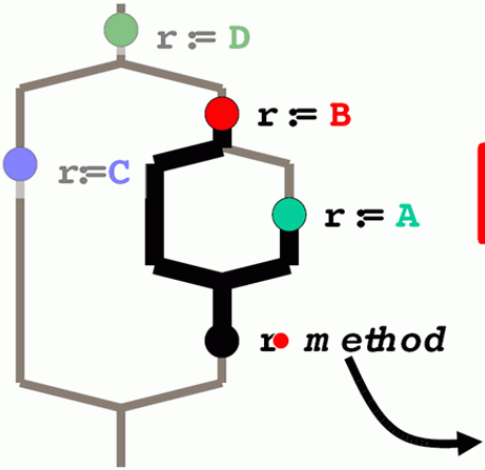


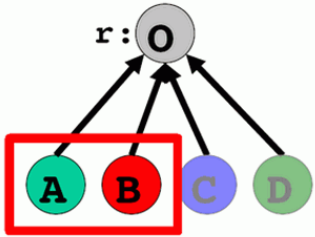
Flow analysis





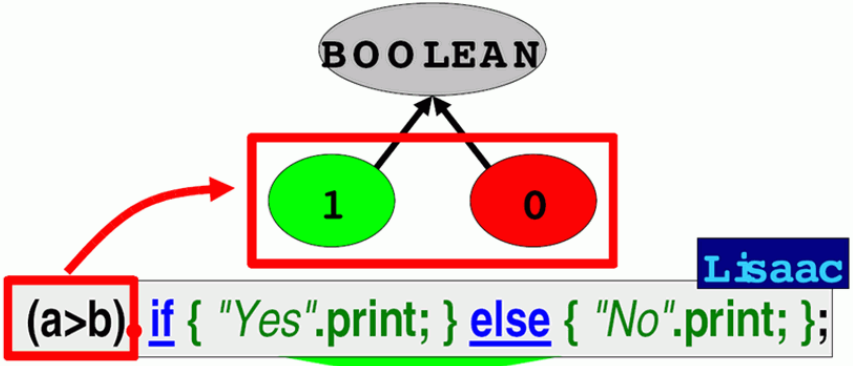
Flow analysis



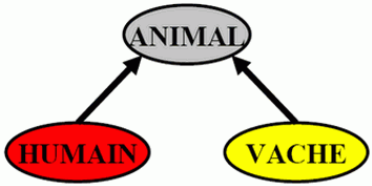


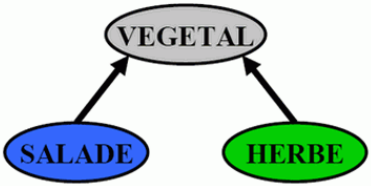

```
if (r -> id = A) {
  method_A();
} else {
  method_B();
};
```

Conditional: *if else*



Specialisation code





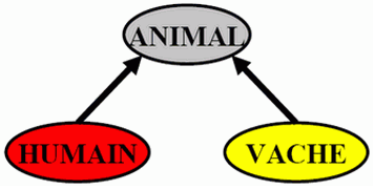
{🔴🟡}.mange {🔵🟢};

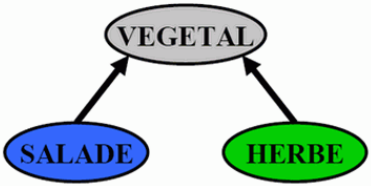
{🔴 }.mange {🔵🟢};

{🔴 }.mange { 🟢};

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

Specialisation code / call 1





{🔴🟡}.mange {🔵🟢};

🔴.mange_1 {🔵🟢};

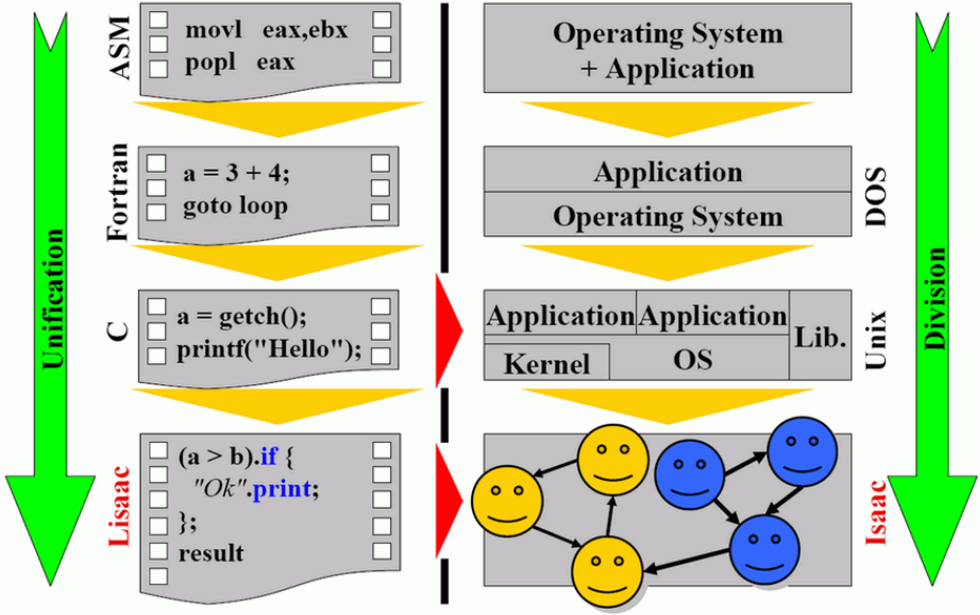
(elt.est_salade);

🟡.mange_2 {🔵🟢};

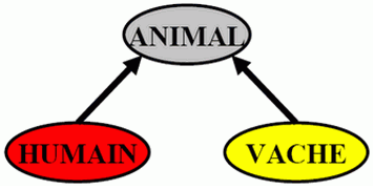
(TRUE);

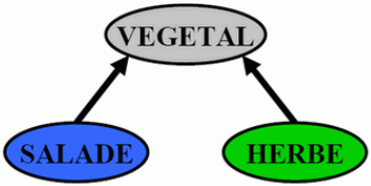
```
- mange elt:VEGETAL :BOOL <-
( + result:BOOL;
  (est_humain).if {
     result := elt.est_salade;
  } else {
     result := TRUE;
  };
  result
);
```

Language *vs* O.S.



Specialisation code / call 2





{🔴 }.mange {🔵🟢};

🔴.mange_1 {🔵🟢};

(elt.est_salade);

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

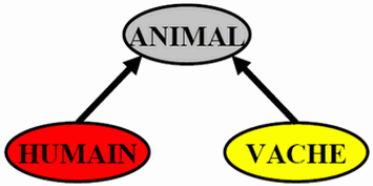
Specialisation code / call 3

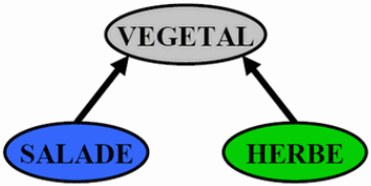

```
{●   }.mange {   ●};
```

```
●.mange_3 {   ●};
( FALSE );
```

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

Specialisation code






```
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
);
```

Specialisation code

Lisaac	SmartEiffel	CPA
{}.mange {}; {}.mange {}; {}.mange { };	{}.mange {}; {}.mange {};	{}.mange {}; {}.mange {}; {}.mange {}; {}.mange {};

Specialisation code

Lisaac	SmartEiffel	CPA
{●}.mange {●●}; {●}.mange {●●}; {●}.mange {●}; {●}.mange {●}; {●}.mange {●}; {●}.mange {●};	{●}.mange {●●}; {●}.mange {●●};	{●}.mange {●}; {●}.mange {●}; {●}.mange {●}; {●}.mange {●};

As fast a C language

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

Lisaac code

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

Lisaac compiler

C code

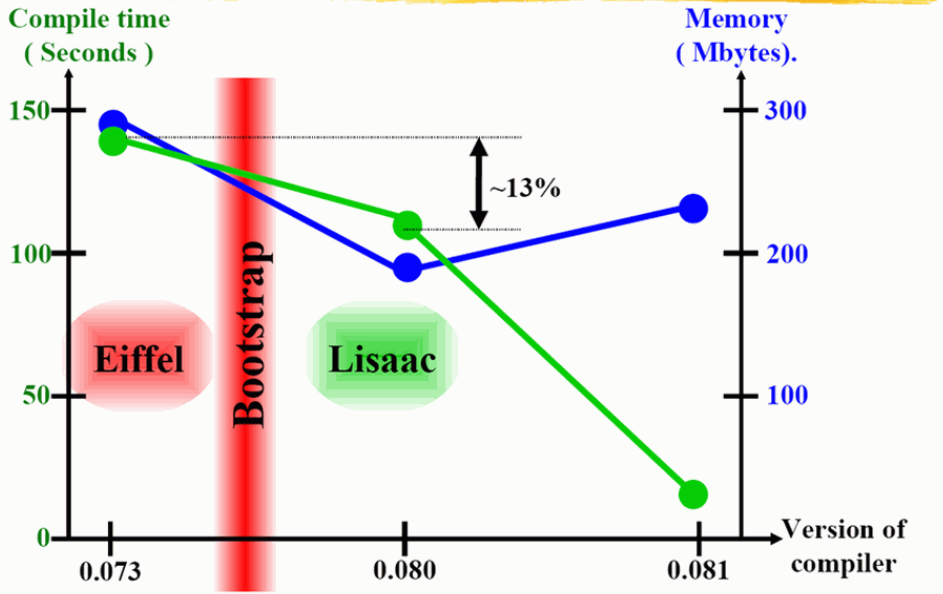
```
j = 0;
while (j<10) {
 putc('H    ',STD_OUT);
 putc('e    ',STD_OUT);
 putc('l    ',STD_OUT);
 putc('l    ',STD_OUT);
 putc('o    ',STD_OUT);
 j = j + 1;
};
```

C, Eiffel, Java *vs* Lisaac

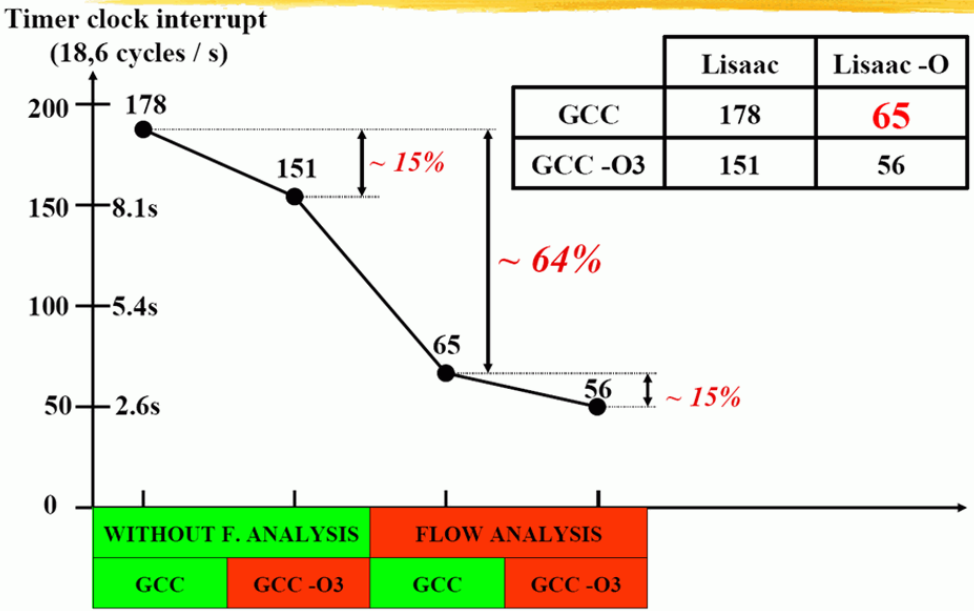
Benchmark runtime on a quick-sort program.

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 benchmark



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

3. Compilateur Lisaac

MPEG2 Decoder

	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 +**

Why a new language ? (1/2)

- C language

advantage

Memory mapping, interrupt management, ASM glue, multiple kind of integer, compiled, very good performance

inconvenience

Not high level language

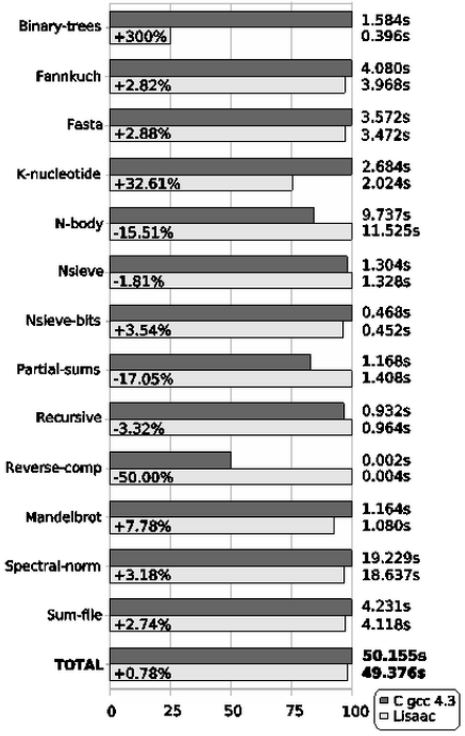
- SmartEiffel language

advantage

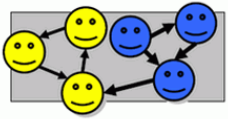
High level language, genericity, uniformity, static type, programming by contract, compiled, good performance

inconvenience

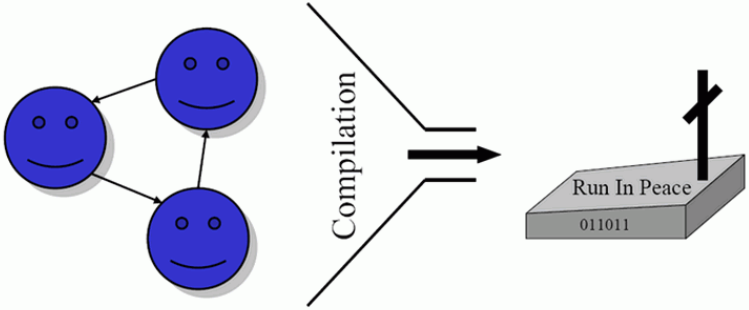
Not prototype object oriented, lack of OS programming facility


Binary-trees
+300%
1.584s
0.396s
Fannkuch
+2.82%
4.080s
3.968s
Fasta
+2.88%
3.572s
3.472s
K-nucleotide
+32.61%
2.684s
2.024s
N-body
-15.51%
9.737s
11.525s
Nsieve
-1.81%
1.304s
1.328s
Nsieve-bits
+3.54%
0.468s
0.452s
Partial-sums
-17.05%
1.168s
1.408s
Recursive
-3.32%
0.932s
0.964s
Reverse-comp
-50.00%
0.002s
0.004s
Mandelbrot
+7.78%
1.164s
1.080s
Spectral-norm
+3.18%
19.229s
18.637s
Sum-file
+2.74%
4.231s
4.118s
TOTAL
+0.78%
50.155s
49.376s
0
25
50
75
100
C gcc 4.3
Lisaac

Object Operating System



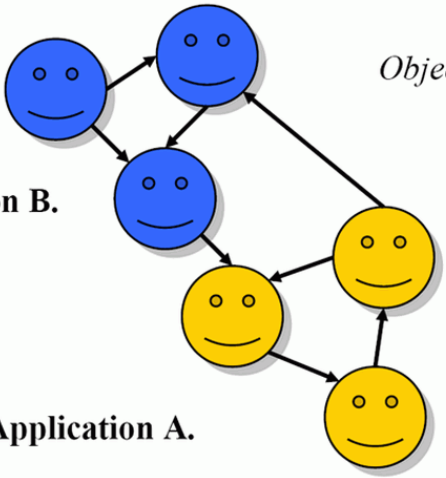
We need OS to be changed



Object Oriented source code of the application A.

Black box of the application A. (OO ?)

… to free objects

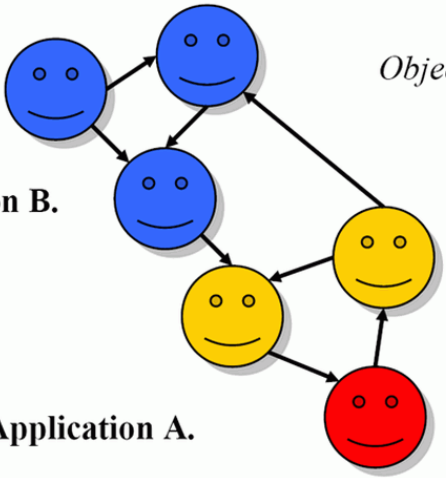


Object expressivity remains after compilation.

Application B.

Application A.

… to free objects



Object expressivity remains after compilation.

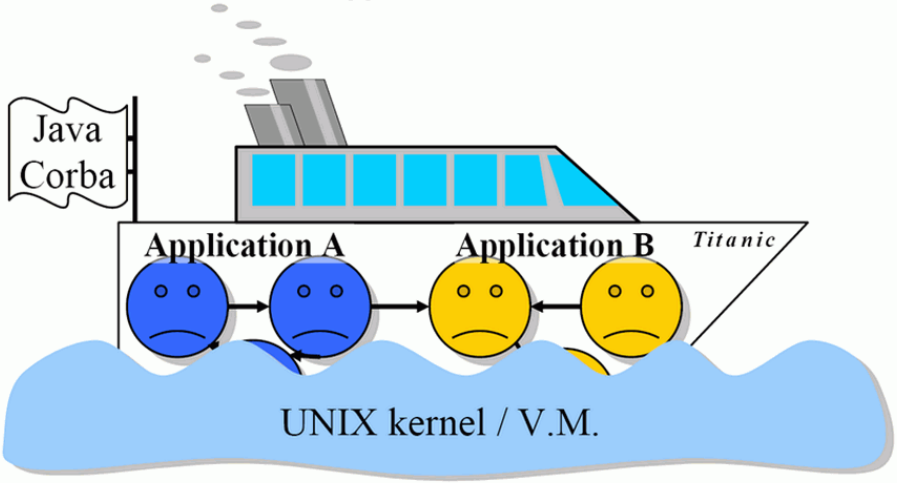
Application B.

Application A.

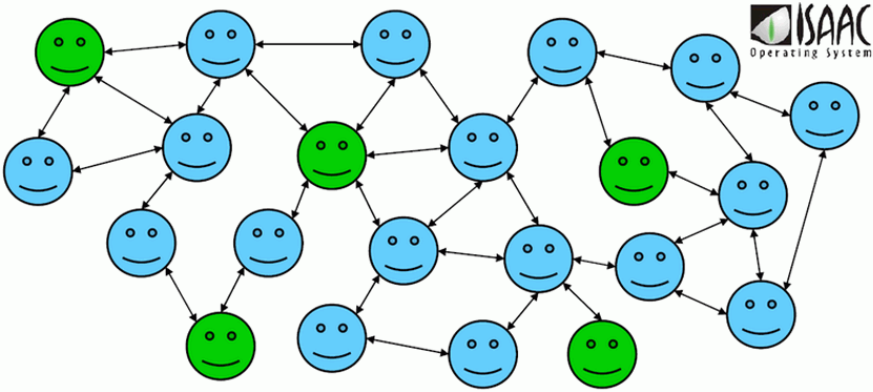
Dynamically replace an object.

Who wants to be uniform?

Let them sink in a bigger box ?



Adaptability



Object of the system

Hardware specific object (architecture 1)

Few hardware specific objects

Adaptability

Current version of Isaac: 24 generic objects (2385 lines)

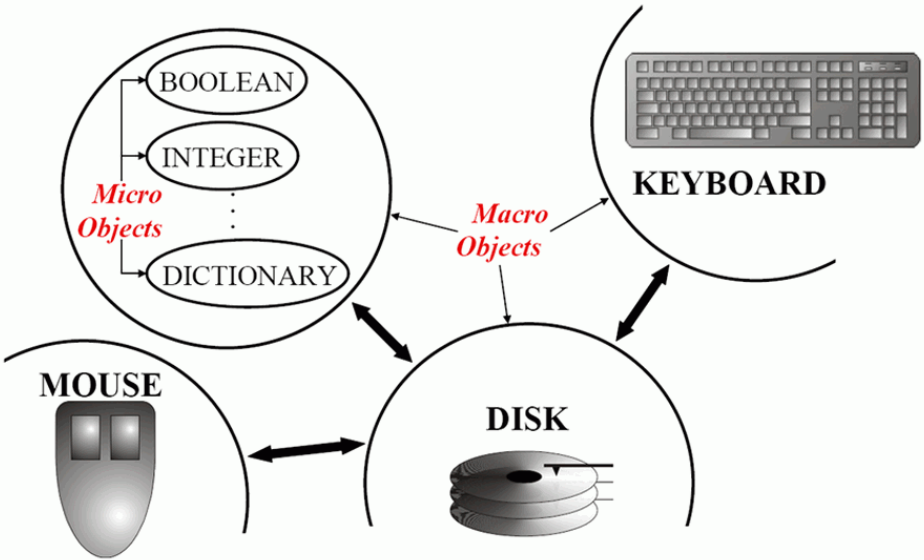
	Hardware Specific Objects	Development time	Size of executable
Intel x86 *Pc*	**37** **4583 lines (63 ASM)**	**First implementation**	**331 Ko**
Motorola DragonBall *Sony PDA*	**12** **815 lines**	**3 month**	**148 Ko**
Intel StrongArm *Ipaq PDA*	**11** **738 lines**	**3 weeks**	**134 Ko**
Intel x86 *Linux version*	**6** **315 lines**	**8 hours !!!**	**194 Ko**
ST processor	**9** **980 lines (340 ASM)**	**2 weeks**	**283 Ko**

Light size

Size of Isaac OS on a Intel x86 architecture

331 Ko	+ Graphic Interface
82 Ko	+ Video Graphic Mode + Mouse
72 Ko	+ File System (FAT 32, FAT 16, FAT 12)
22 Ko	+ Video Text mode + Keyboard
17 Ko	Processor + Memory

Adaptability *vs* Security



Why a new language ? (2/2)

- Self language

advantage

Uniformity, expressivity, simplicity, prototype object oriented

inconvenience

Not compiled, lack of protection (no type), lack of OS programming facility

- Java language

advantage

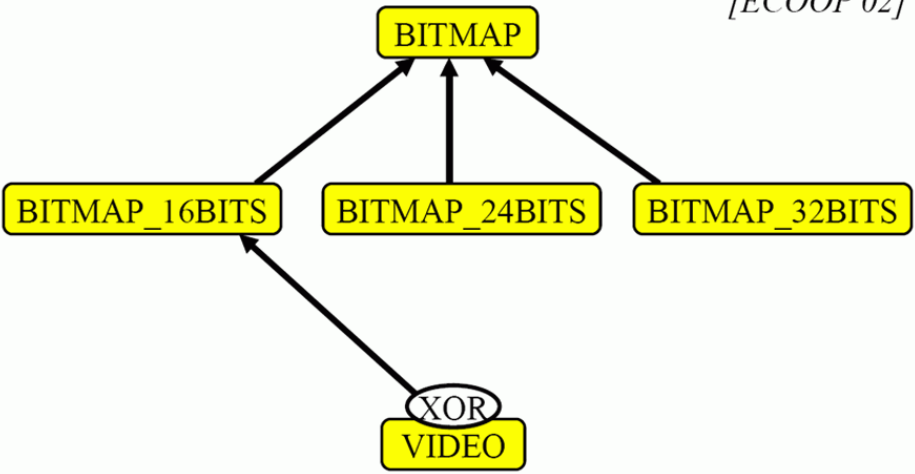
C-like syntax, static type, internet facility

inconvenience

Not prototype object oriented, lack of OS programming facility, not good performance, lack of uniformity and expressivity

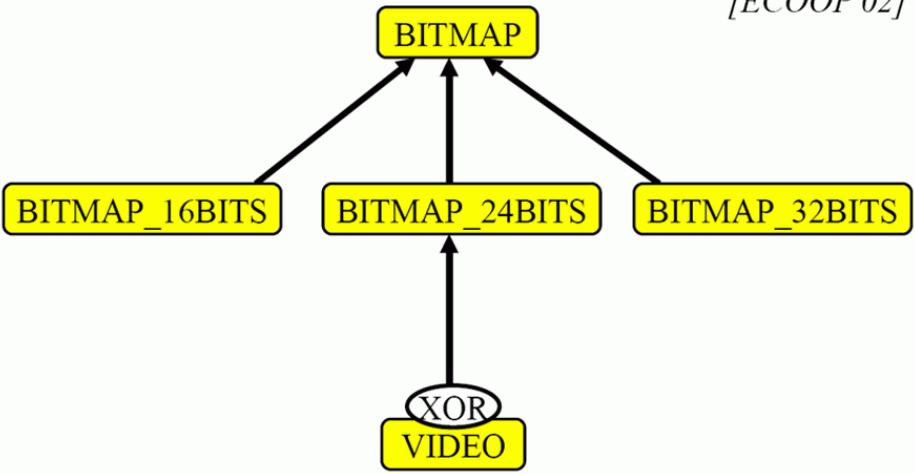
Why dynamic inheritance?

[ECOOP'02]

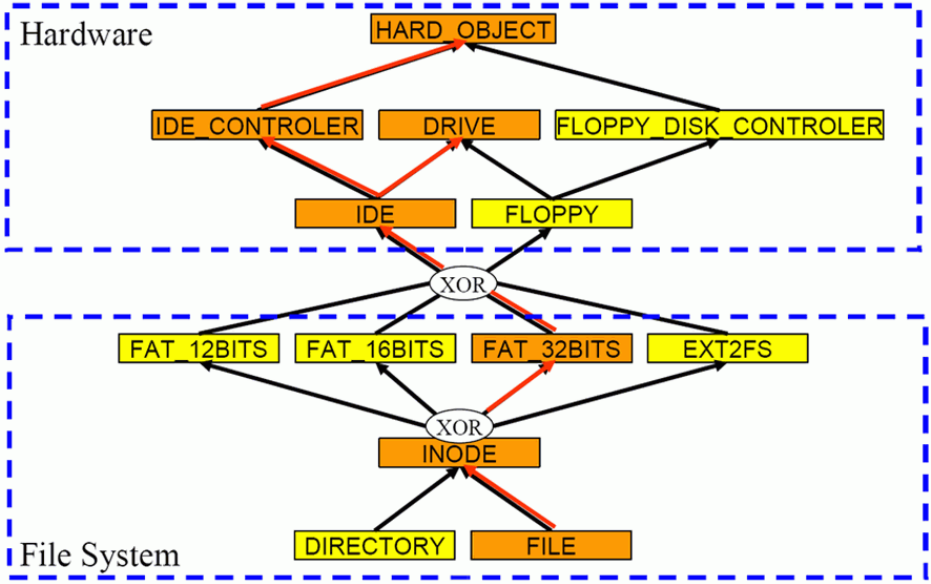


Why dynamic inheritance?

[ECOOP'02]



D.I. : applied to F.S.



Question ?

IRC

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

Information & contacts

- **Wiki** : `http://lisaac.u-strasbg.fr/index.php/Accueil`
- **Mailing list** : `http://isaacproject.u-strasbg.fr/community/mailing_lists.html`

Good luck!

History : Lisaac for IsaacOOS Language

In the past...

C language

⇓

Unix system

The futur...

Lisaac

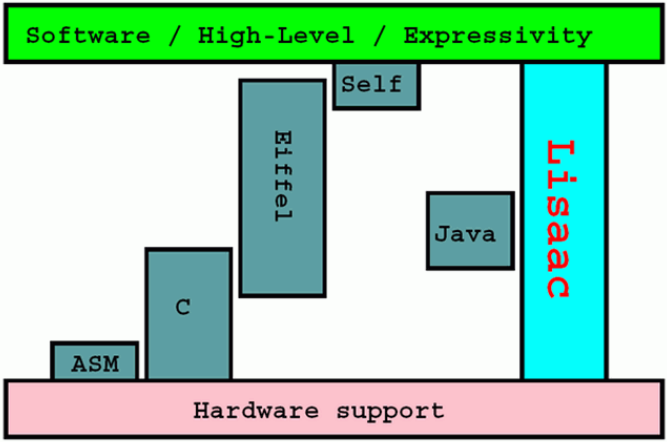
Prototype based Object Oriented Language

⇓

IsaacOOS

Prototype Object Operating System

High-level *vs* Hardware
Object Oriented for Hardware



Class *vs* Prototype (1/3)

Class

Instance

1 Squeleton
(=class)

1 Object

Prototype

Clone

1 Object prototype
(=the One)

1 other Object