CLOS Efficiency: Instantiation

Didier Verna

didier@lrde.epita.fr
http://www.lrde.epita.fr/˜didier

ILC 2009 – Tuesday, March 20th

The context

Don't look at me... like *that*

- Not (particularly) interested in performance
- Not (at all) a Lisp implementer

▶ Merely an observer

Look at me... like *this*

- Surrounded by C++ gurus (Cf. `Olena`)
- Performance does matter to them
- But you should see the code !

▶ This would be so much easier in Lisp, but...

They wouldn’t dare to complain about parens. . .

Because if you can read this,

```
 template <template <class>  class M,  typename T,  typename V>
 struct  ch_value_ <M <tag::value_<T>>, V>
{ typedef  M<V> ret; };

 template <template <class>  class M,  typename I,  typename V>
 struct  ch_value_ <M <tag::image_<I>>, V>
{ typedef  M <mln_ch_value(I, V)> ret; };

 template <template <class, class>  class M,  typename T,
   typename I,  typename V>
 struct  ch_value_ <M <tag::value_<T>, tag::image_<I>>, V>
{ typedef  mln_ch_value(I, V) ret; };

 template <template <class, class>  class M,  typename P,
   typename T,  typename V>
 struct  ch_value_ <M <tag::psite_<P>, tag::value_<T>>, V>
{ typedef  M<P, V> ret; };
```

They wouldn't dare to complain about parens. . .

surely you can read that !

```
(template (template (class) (class M) (typename T) (typename V))
(struct (ch_value_ (M (tag::value_ T))  V)
( typedef (M V) ret)) )

(template (template (class) (class M) (typename I) (typename V))
(struct (ch_value_ (M (tag::image_ I))  V)
( typedef (M (mln_ch_value I  V)) ret)) )

(template (template (class  class) (class M) (typename T)
  (typename I) (typename V))
(struct (ch_value_ (M (tag::value_ T) (tag::image_ I))  V)
( typedef (mln_ch_value I  V) ret)) )

(template (template (class  class) (class M) (typename P)
  (typename T) (typename V))
(struct (ch_value_ (M (tag::psite_ P) (tag::value_ T))  V)
( typedef (M P  V) ret)) )
```

The performance “issue”

Typical conversation

Yobbo: But LISP is slow right?

Me: How do you know that?

Yobbo: `[choose your favorite answer]`

✗ Huh, it’s a well known fact

✗ Well, that’s what I heard

✗ Last time I checked [. . .]

✓ It’s dynamic, so it’s slow

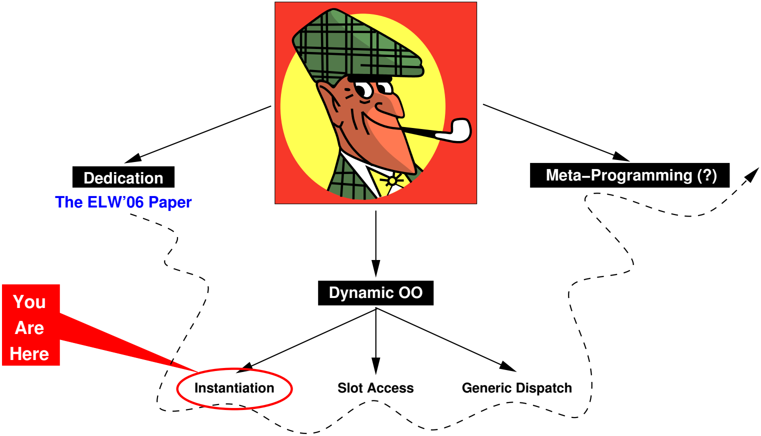
The real problems

- **Lack of strong evidence** (don’t know / don’t care)
- **From the ground up** (micro-benchmarking)

▶ Where are we today in terms of performance?

My (not so) secret agenda

On the behavior and performance of LISP



Dedication

The ELW’06 Paper

Meta–Programming (?)

Dynamic OO

Instantiation

Slot Access

Generic Dispatch

You Are Here

Table of contents

1. The experiments
2. C++ Grounding
3. LISP surprises
   - Structures
   - Classes
4. Cross-language comparison

Experimental protocol

```
Class *instance = new Class;
(make-instance ...)
```

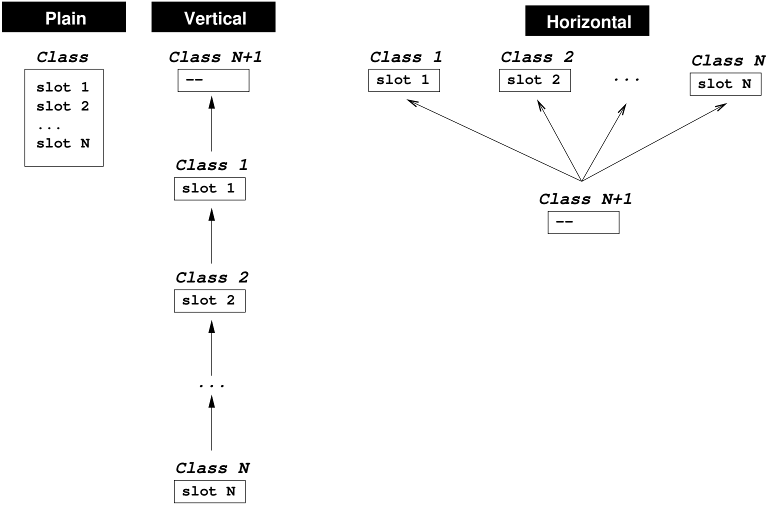
- ≠ compilers
- Class size (1, 7, 49 slots)
- Class hierarchy (plain, vertical, horizontal)
- Slot type (fixnums, single-floats)
- Slot initialization (yes, no)
- Slot allocation (instance, class)
- Optimization level (safe, optimized, inline)

▶ 1300+ individual tests

Compilers

- **C++:** GCC 4.3.2 (Debian package 4.3.2-1)
- **LISP:**
  - CMU-CL 19d (Debian package)
  - SBCL 1.0.22.17
  - ACL 8.1 Express Edition

Class hierarchies



Plain

Class

slot 1
slot 2
...
slot N

Vertical

Class N+1

--

Class 1

slot 1

Class 2

slot 2

...

Class N

slot N

Horizontal

Class 1

slot 1

Class 2

slot 2

...

Class N

slot N

Class N+1

--

Slot initialization / allocation

Initialization

- Compile-time constants
- **LISP:** `:initform` only
- **C++:** inside a provided constructor with no argument

Shared slots

- **C++:** strictly compile-time
- **LISP:** run-time, but hopefully during class finalization or first instance creation

Optimization modes

C++

`-O3 -DNDEBUG`

LISP

- Not inlined: `(make-instance some-class)`
  - “safe”: `(safety 3) (... 0)`
  - “optimized”: `(speed 3) (... 0)`
- “inline”:
  - “optimized” settings
  - `(make-instance 'myclass)`

Final remarks

structures *vs* classes

- **C++:** `struct` $\Longleftrightarrow$ `class`
- **LISP:** `struct` $\neq$ `class`

Meta-classes

LISP-specific

Memory management

- **C++:** manual
- **LISP:** automatic through (different) GC

▶ Avoid benchmarking

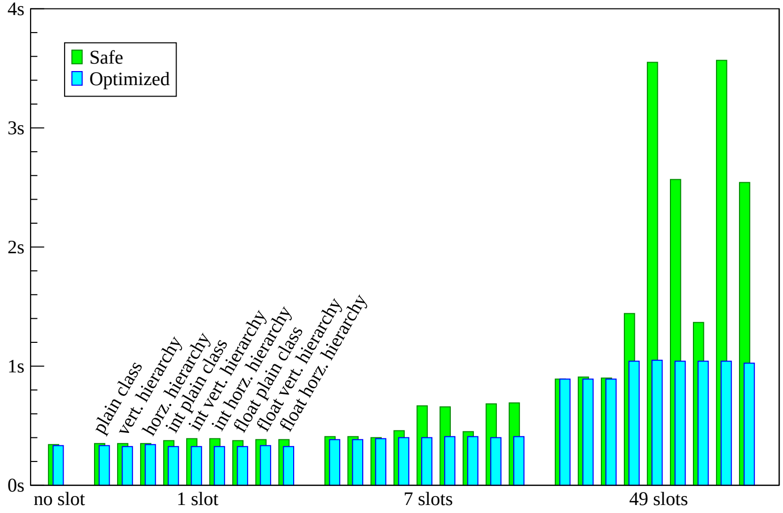
Experimental conditions

- Debian GNU Linux / 2.6.26-1-686 packaged kernel
- i686 DualCore CPU
  - 2.13GHz
  - 2GB RAM
  - 2MB level 2 cache
- Single user mode
- All benchmarks at least 1s
- Avoid memory exhaustion / swapping (C++)

▶ 10% significance margin

C++ Results

5,000,000 objects, local slots



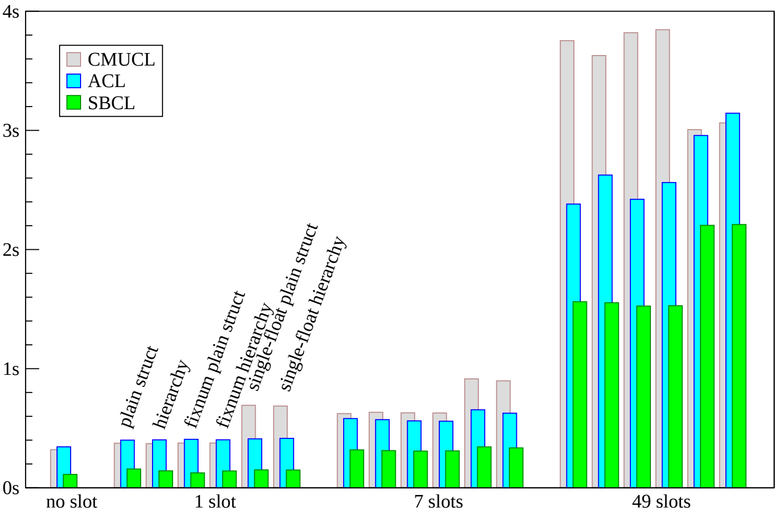
C++ behavior

- Immune to slot type
- Optimization mode *flattens* timings
  - Small effect of initialization remains
- Safe mode very sensitive to:
  - Slot initialization
  - Class hierarchy

  ▶ Morphology of constructor call chain
- Shared slots: *all flat*

LISP structure results

10,000,000 objects, inline mode



LISP structure behavior

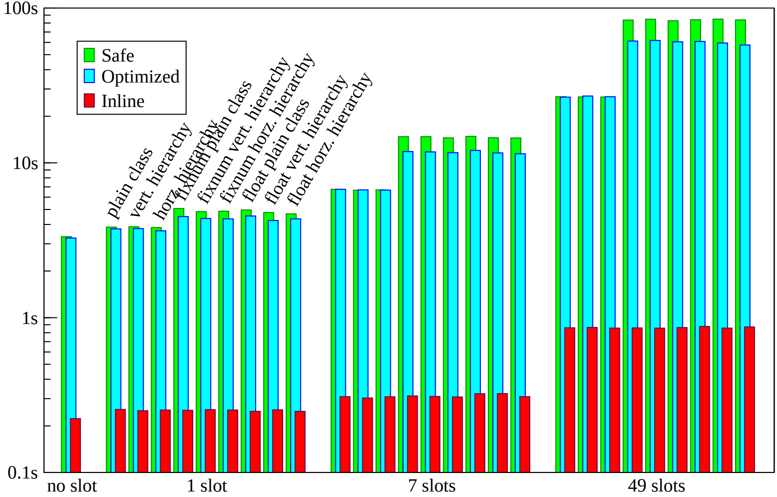
- **Dependence on slot type**
  Internal representation / (un)boxing
- **Immune to (`fixnum`) slot initialization**
  Slots always initialized to `nil` (not required)
- **Immune to structure hierarchy**
  `struct` ⟺ `vector`

Discrepancies

- **Type checking:**
  - CMU-CL: always (except `fixnums` in 19d)
  - SBCL: depends on compiler settings
  - ACL: never
- CMU-CL on `single-float` ???

LISP class results

SBCL, 5,000,000 objects, standard class, local slots



LISP class behavior

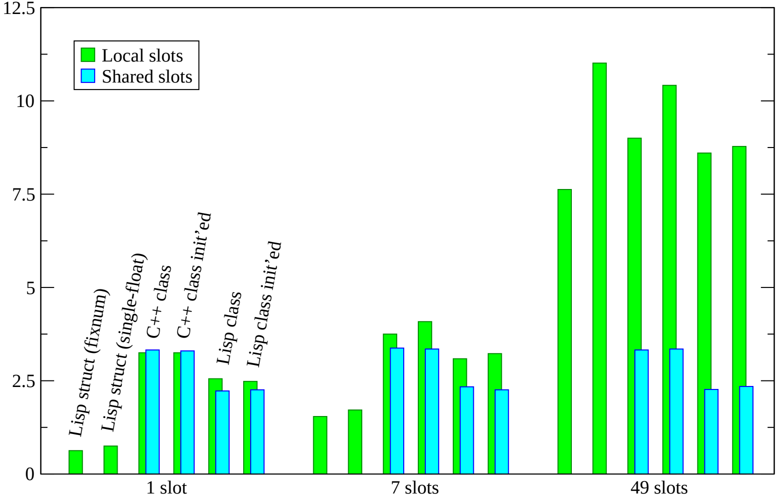
- **Immune to slot type / class hierarchy**
  No special representation, instance vector lookup + access
- **Slots always initialized** (secret unbound value)
  But only slot access time visible
- **Inline mode:** `(make-instance ’class)`
  Improvement 15x to 100x !!
- **Shared slots:** all flat
  Bug (fixed): dependent on class size

Discrepancies

- **Type checking:**
  - CMU-CL: not in safe mode, in contradiction with the manual (fixed)
  - SBCL: missing on shared slots (fixed)
  - ACL: never
- **Meta-class:**
  - CMU-CL sensitive (30 – 50% degradation)
- **Slot initialization:**
  Makes ACL faster (20% in inline mode)
- **ACL on shared slots:**
  - Dependence on class size (10x from small to big class)
  - Dependence on slot initialization
    - Safe/optimized mode: degradation of 3.5x
    - Inline mode: improvement by 2x
  - Sometimes slower than local slots

Cross-language comparison

5,000,000 objects, inline mode



Cross-language behavior

- Lisp structures instantiate faster for smaller objects
- Lisp instantiation is *faster* than in C++ (1.2x)
- Even more so with shared slots (30%)

Conclusion

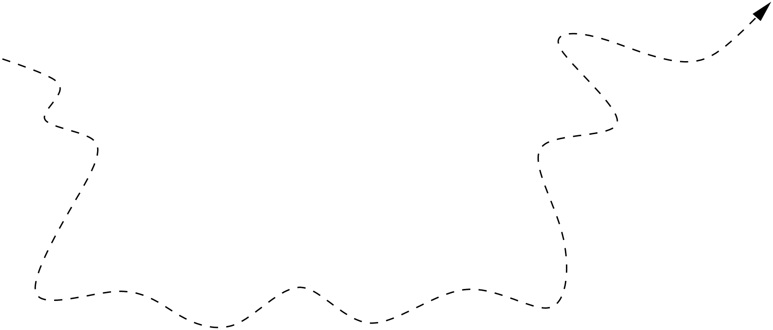
- **Safe mode:** LISP and C++ behave differently
  - C++ sensitive to class hierarchy
  - LISP sensitive to slot type
- **Optimized mode:**
  - Convergence in both behavior and performance
  - `(make-instance 'class)` !!
  - *faster* instantiation in LISP

▶ Kudos to LISP implementers…

- **The dark side of the force:**
  - Type checking (has an impact on performance)
  - COMMON-LISP standard underspecified

Perspectives

- Finish investigation
- Other compilers
- Other architectures
- Regression surveillance
- The rest of the path…



Thanks!

Any quesλions?

- Nikodemus Siivola
- Raymond Toy
- Duane Rettig

This is not a work of fiction. Any resemblance between the characters and persons, living or dead, is purely intentional.