A MOP-Based Implementation for Method Combinations

ELS 2023

Didier Verna

EPITA / LRDE

didier@lrde.epita.fr

lrde/~didier

@didierverna

didier.verna

in/didierverna

Benefits of Method Combinations

Bad: dispatch *code* mixed up with methods code

```
Class::method (...);
super.method (...);
(call-next-method ...)
```

Good: dispatch *policy* declared separately

```
(defgeneric func (...)
  (:method-combination mc ...) ;; declaration
  (:method (...) #| method-specific code only |#)
  ...)
```

- Increased orthogonality / SoC

Benefits of Meta-Object Protocols

What

- Expose a language's implementation
- For introspection / intercession

How

- Standard classes available for sub-classing
  or modification
- Standard (generic) functions available for specialization
  or modification

- Homogeneous behavioral reflection

Except that...

- `method-combination`: abstract class
- `define-method-combination`: macro

That's about it...

Consequences

- No portability to be hoped for
  At least one implementation-specific sub-class
- Missing / ill-designed existing protocols
  E.g. `find-method-combination` / `compute-effective-method`
- Unclear intended nature
  Classes? Instances?
- Unclear intended implementation
  "contains information about both the type of method combination and the arguments being used with that type"

- *Cf.* 2018 paper

Plan

Overview

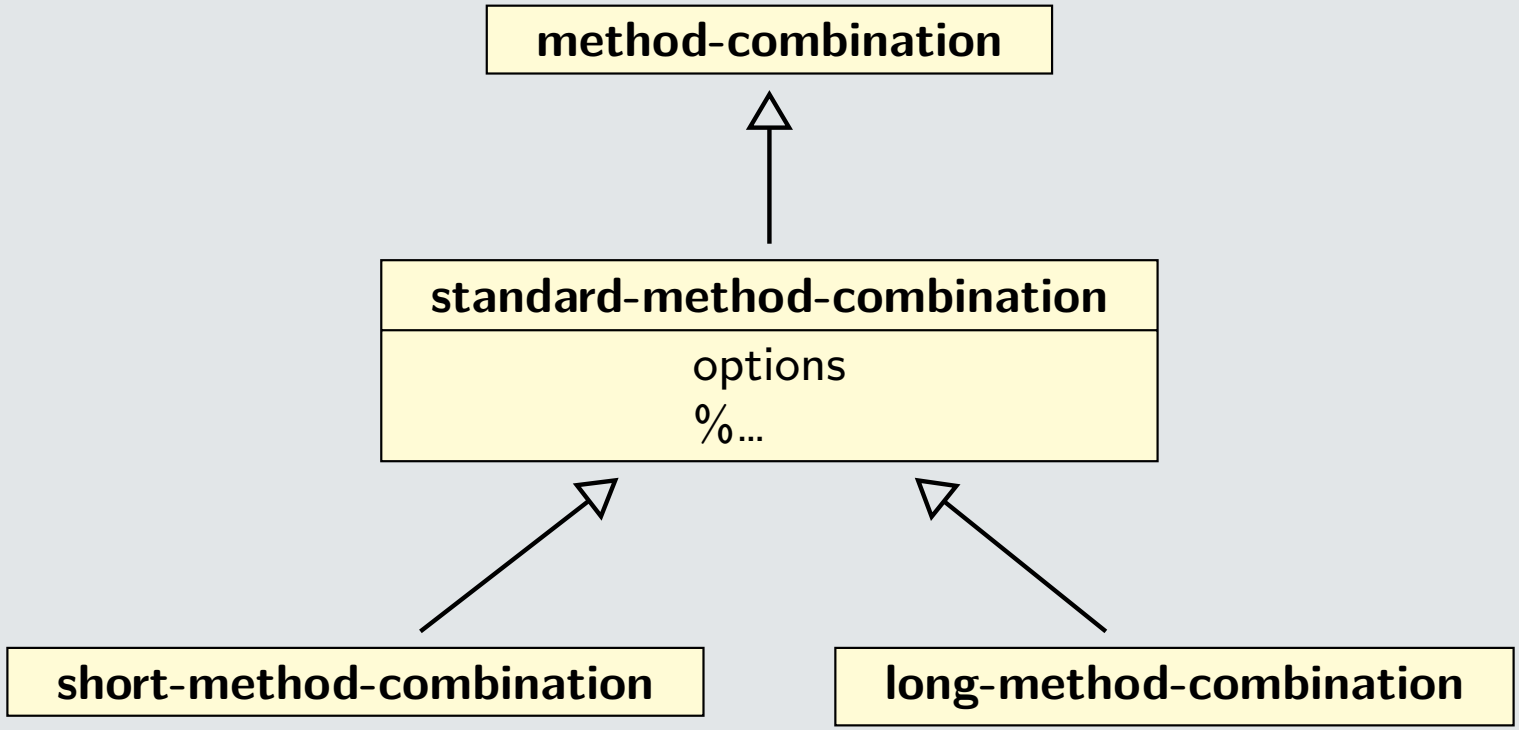
1. The standard consistently speaks of "method combination *types*".
   - Method combinations should be reified as *types*`^D^D^D`*classes*.
   - Generic functions should instantiate those classes.
2. There are two "forms" of method combinations: short & long.
   Note: the PCL hierarchy already reflects that.
   - Method combinations "forms" should also be reified as classes.
   - We have two classes of (classes of) method combinations.

Summary

A method combination (class) is:

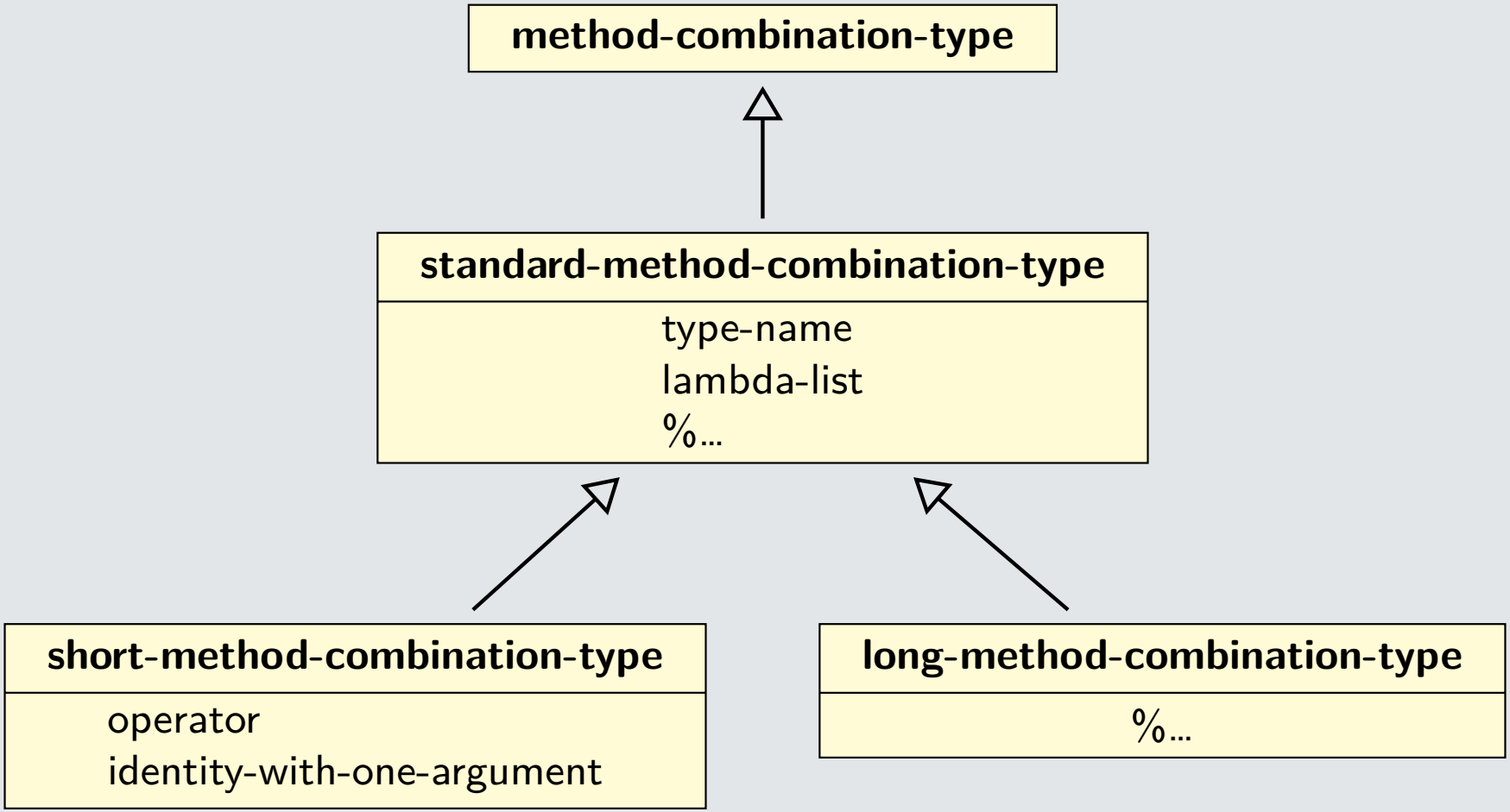
- a short / long one (inheritance)
- of the short / long kind (implementation)

Method Combinations Hierarchy



- `(defgeneric ... (:method-combination mc options) ...)`

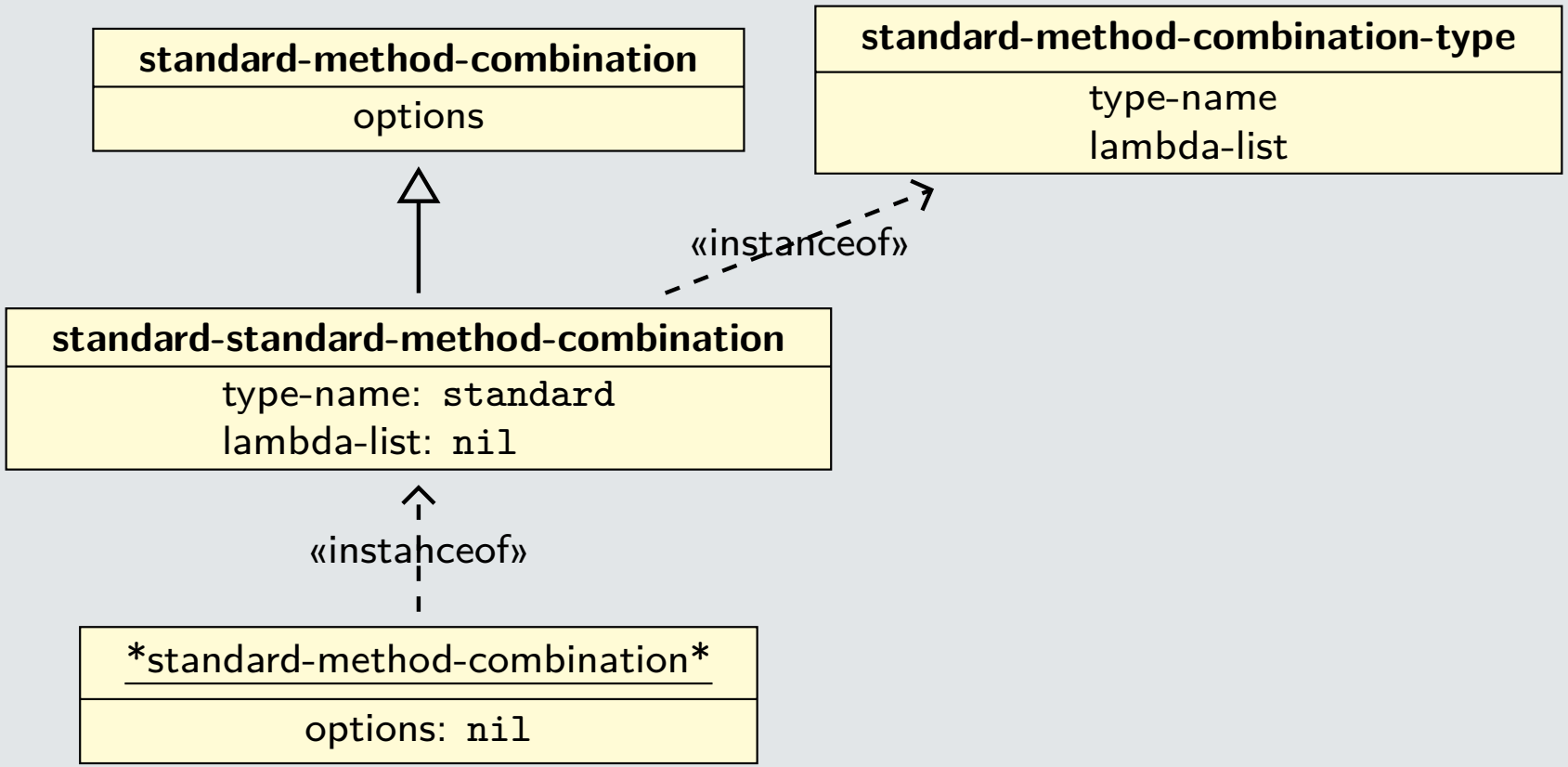
Method Combination Types Hierarchy



Plan

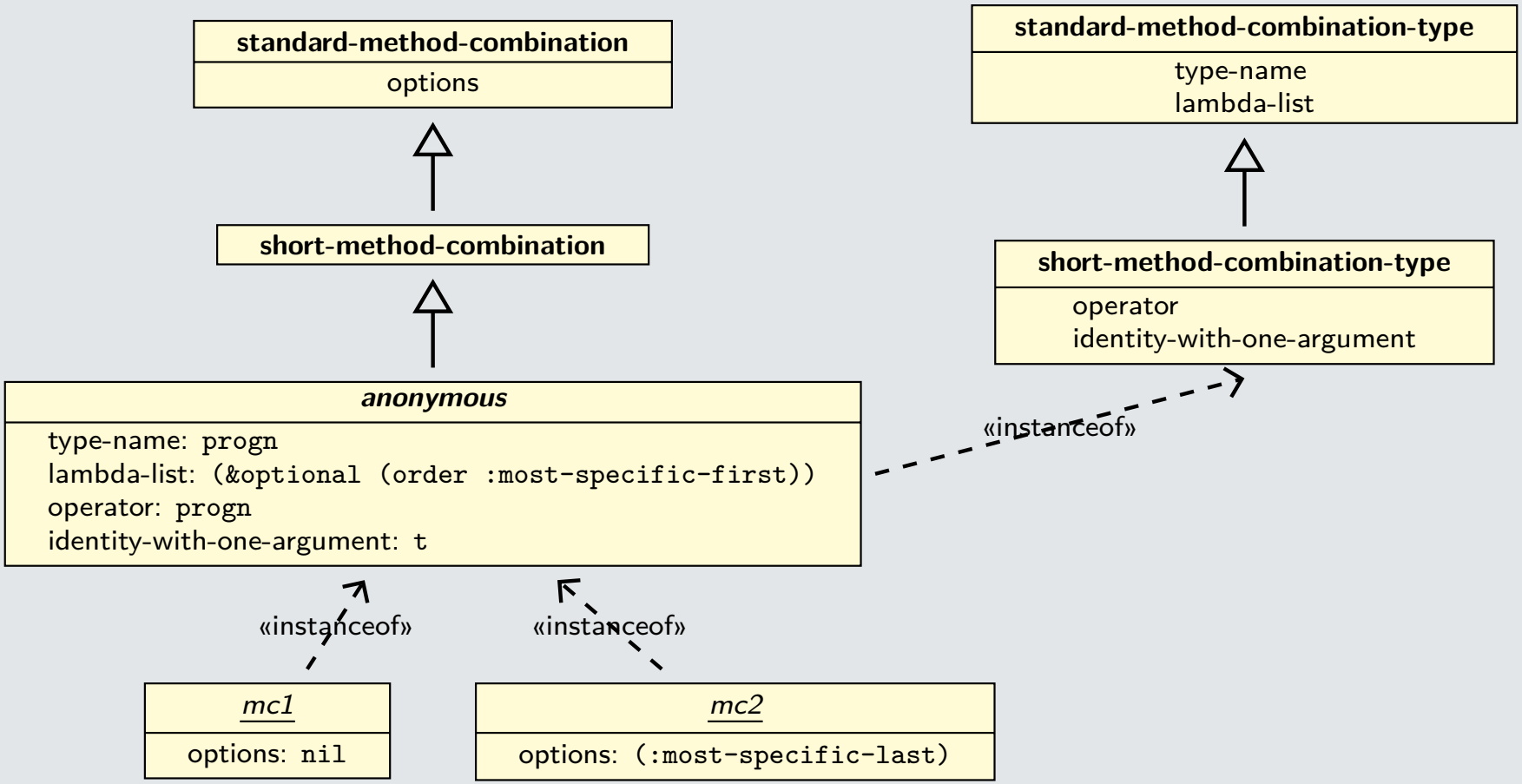
Example #1

Standard (Standard) Method Combination



Example #2

Short Method Combinations



Plan

Design

Examples

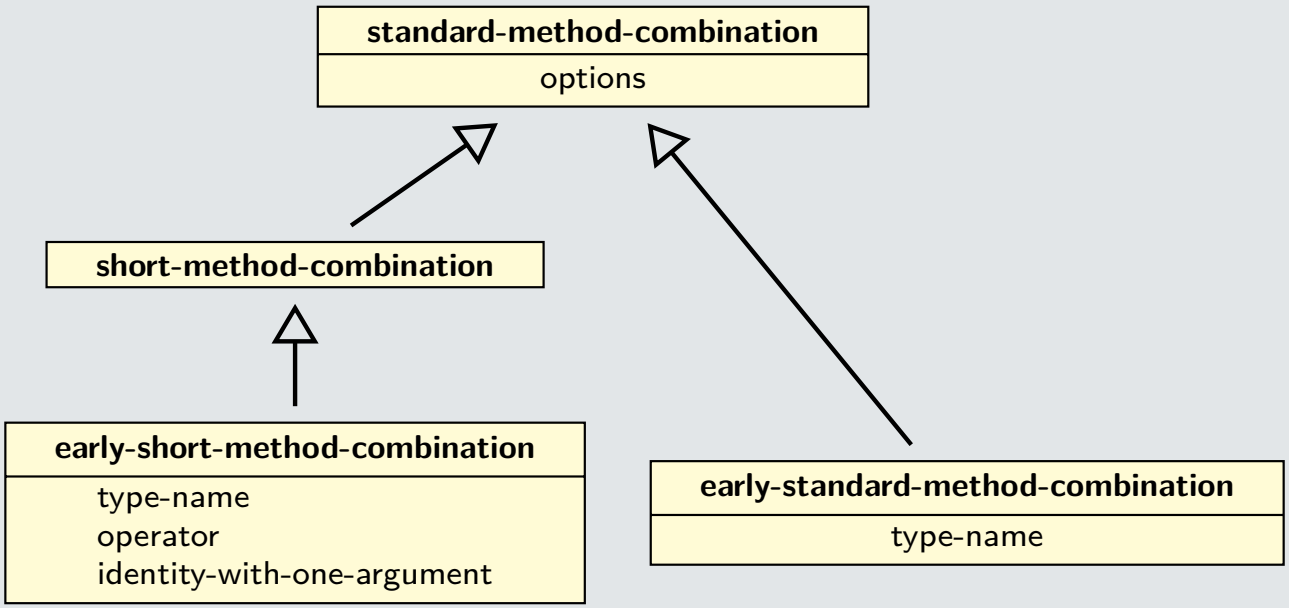
Implementation (SBCL)

Extensibility

Performance

Bootstrap

Early Method Combinations



- Two “early” method combinations: `standard` & `or`
- Bootstrap uses them
- Post-bootstrap conversion to the full architecture

Protocol Additions

```
find-method-combination-type (name &optional (errorp t))
"Find a NAMEd method combination type.
If ERRORP (the default), throw an error if no such
method combination type is found.
Otherwise, return NIL."
```

```
find-method-combination* (name &optional options (errorp t))
"Find a method combination object for NAME and OPTIONS.
If ERRORP (the default), throw an error if no NAMEd
method combination type is found.
Otherwise, return NIL.

Note that when a NAMEd method combination type exists,
asking for a new set of (conformant) OPTIONS will
always instantiate the combination again, regardless
of the value of ERRORP."
```

Plan

Design

Examples

Implementation (SBCL)

Extensibility

Performance

Sub-Classing Method Combinations (Types)

Short Form

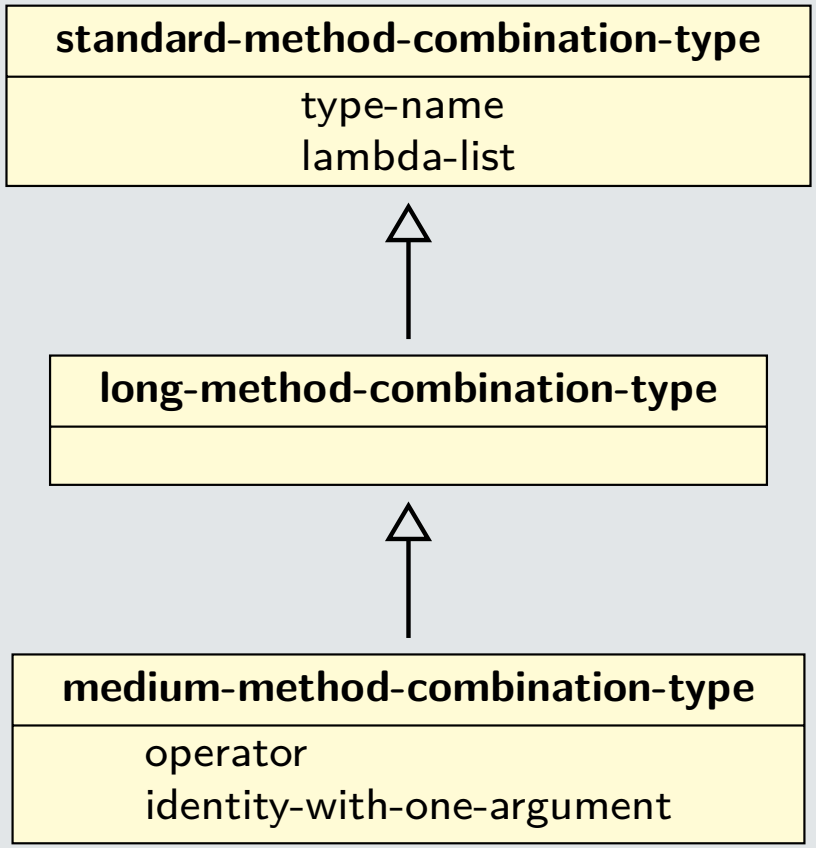
```
(define-method-combination ...
  :method-combination-class name
  :method-combination-type-class name) ;; or (name initargs*)
```

Long Form

```
(define-method-combination ...
  (:method-combination-class name)
  (:method-combination-type-class name initargs*))
```

- Remains standard-compliant

Example: Medium MC Types



Example: Medium MC Types

A macro...

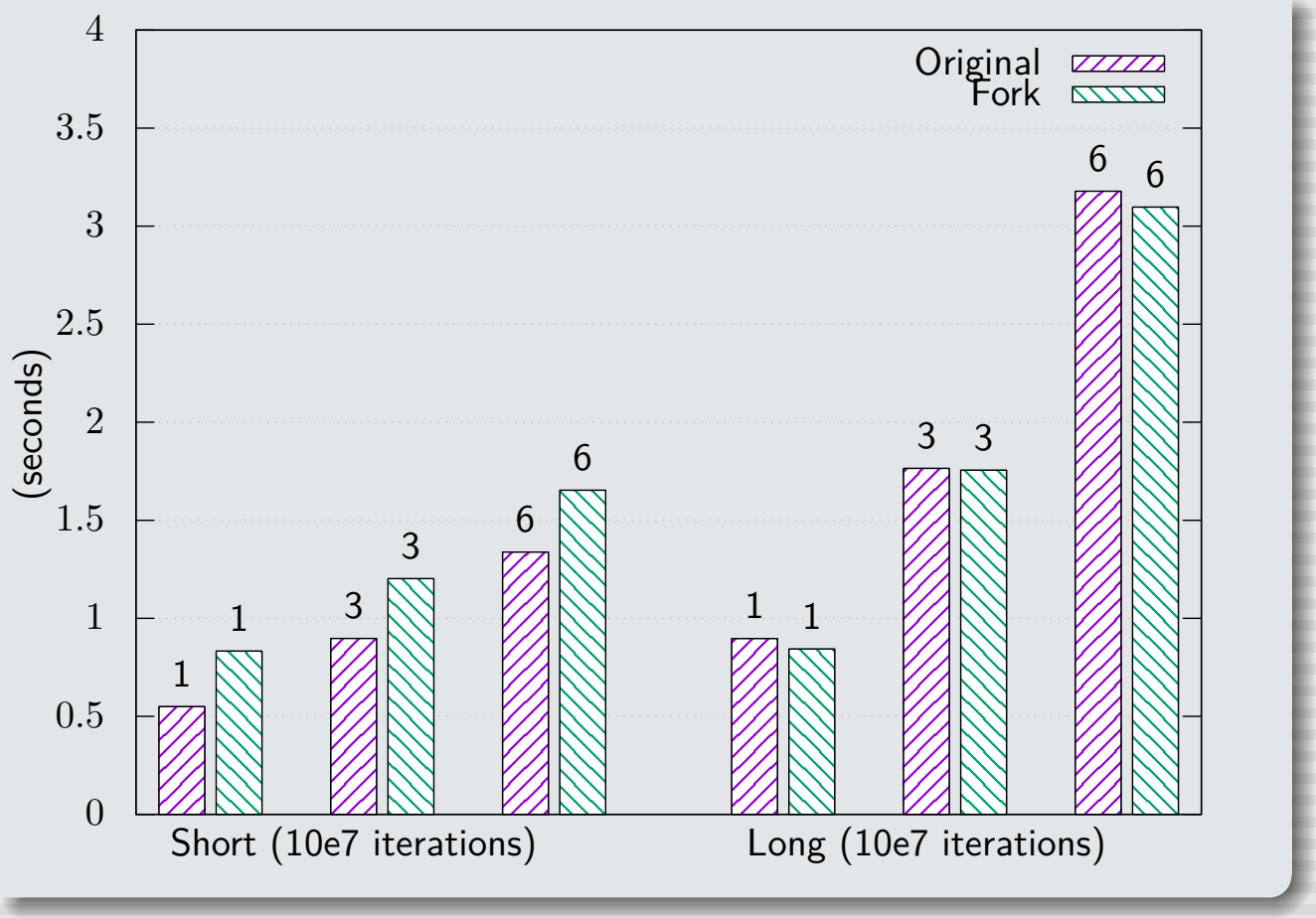
```
(define-medium-method-combination-type my-progn
  :operator progn :identity-with-one-argument t)
```

... which expands to:

```
(define-method-combination myprogn (&optional (order :most-specific-first))
  ((around (:around))
   (before (:before))
   (primary () :order order :required t)
   (after (:after)))
   (:method-combination-type-class medium-method-combination-type
     :operator progn :identity-with-one-argument t)
  ...)
```

Plan

Performance



Conclusion

- Method combination *types* reified
- Standard-compliant implementation
- Close to PCL's original design
- Brings portable MOP-based extensibility to method combinations
- SBCL fork & examples available on GitHub
  - github/didierverna/sbcl/tree/method-combination-types
  - github/didierverna/els2023-method-combinations
- A lot more details in the paper...

Thank you!