DYPL

Course information

- 14 Lectures
- 3 Assignments (3x1.5 credit)
- 3 credit exam
- Course information on web page
- Discussion in FC and in the weekly seminar

http://people.dsv.su.se/~beatrice/DYPL/

Lectures

1. What do you mean “dynamic”?
2. Shortcomings of dynamic programming languages
3. Duelling Banjos I: Ruby vs. Python
4. Duelling Banjos II: Ruby vs. Python

Lectures, cont'd

5. More on Python
6. More on Ruby
7. Dynamic Languages III: Reflection and Metaprogramming
8. Back to the future: Smalltalk
9. Dynamic Languages IV:Philosophy and all that jazz

Lectures, cont'd

- Lisp
- Javascript
- Smalltalk
- Erlang
- Combining static and dynamic typing

Assignments

- Examines programming part
- 3 x 1.5 credit
- Python, Ruby + other new language
- Must work in pairs and rotate pairs
- Solutions presented and defended

Weekly Seminars

- We prepare some discussion topic and/or programming problem
- The time will be used for discussions about some theoretical problem, programming problem or other course related topics

Literature Seminars

- “Hackers and Painters” - Paul Graham
- “The Cathedral and the Bazaar” - Eric S. Raymond
- “The Best Software Writing” - ed. Joel Spolsky

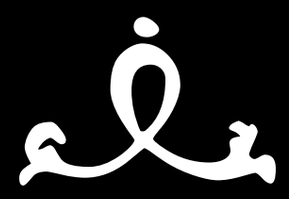
Exam

- "Take-home" exam
- Might be combination of theoretical and practical problems
- Example questions will be distributed
- Bonus credits can be used to "buy" answers
- The exam will be published 2012-03-14 08.00 on the course web page
- The exam should be handed in according to the instructions in the exam document on 2012-03-17

Deadlines

- We have no hard deadlines during this course
- *But*, that doesn't mean that you shouldn't set up deadlines for yourself!
- We recommend you to finish
  - Assignment 1 by February 6
  - Assignment 2 by February 21
  - Assignment 3 by March 7

Questions?



What do you mean “dynamic”?

Dynamic?

Oxford Dictionary

- *[Process or system]* characterised by constant change, activity, or progress
- *[A person]* positive in attitude and full of energy and new ideas
- A force that stimulates change or progress within a system or process

Synonyms

- Energetic, spirited, active, lively, vital, vigourous, forceful, powerful, positive
- High-powered, aggressive, bold, enterprising
- Informal go-getting, peppy, full of get-up-and-go, full of vim and vigour.

Scripting Language

- Script comes from performing arts
- One-off tasks
- Customising admin tasks
- Simple, repetitive tasks

~~"Scripting language"~~

- A belittling term
- Script implies "a small program"
- We use dynamic [programming] language

A Few DYPLs

- LISP (~1956)
- Smalltalk (1972)
- Self (1986)
- Perl (1987)
- Python (1991)
- Ruby (1993)

What does dynamic mean for PLs?

Dynamic typing

Static Typing

- Each variable must be declared with a particular type and it must be used in ways that are appropriate for the type
- Type checking at compiletime
- C, C++, Java, ML, Haskell, etc.

```
public int sumOfWages( BaseballPlayer[] bs )
{
  int sum = 0;
  for ( int i=0; i < bs.length; ++i )
  {
    sum += bs[ i ].wage( );
  }
  return sum;
}
```

Dynamic Typing

- In a dynamically typed language, you can send any message to any object, and the language only cares that the object can accept the message

```
def sumOfWages( aList ):
   sum = 0
   for item in aList:
      sum += item.wage( )
   return sum
```

Dynamic Type

Checking?

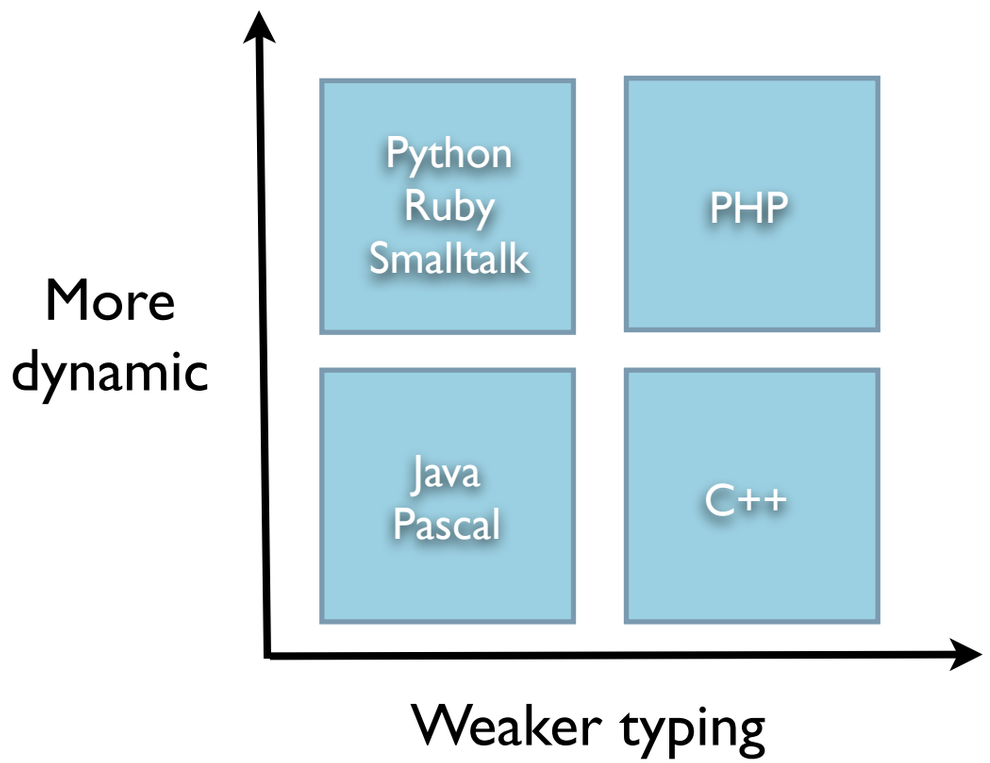
Dynamic Type?

Typing?

- Two forms of typing
  - Strong
  - Weak
- Strong typing and static typing are orthogonal concepts

Weak or Strong Typing?

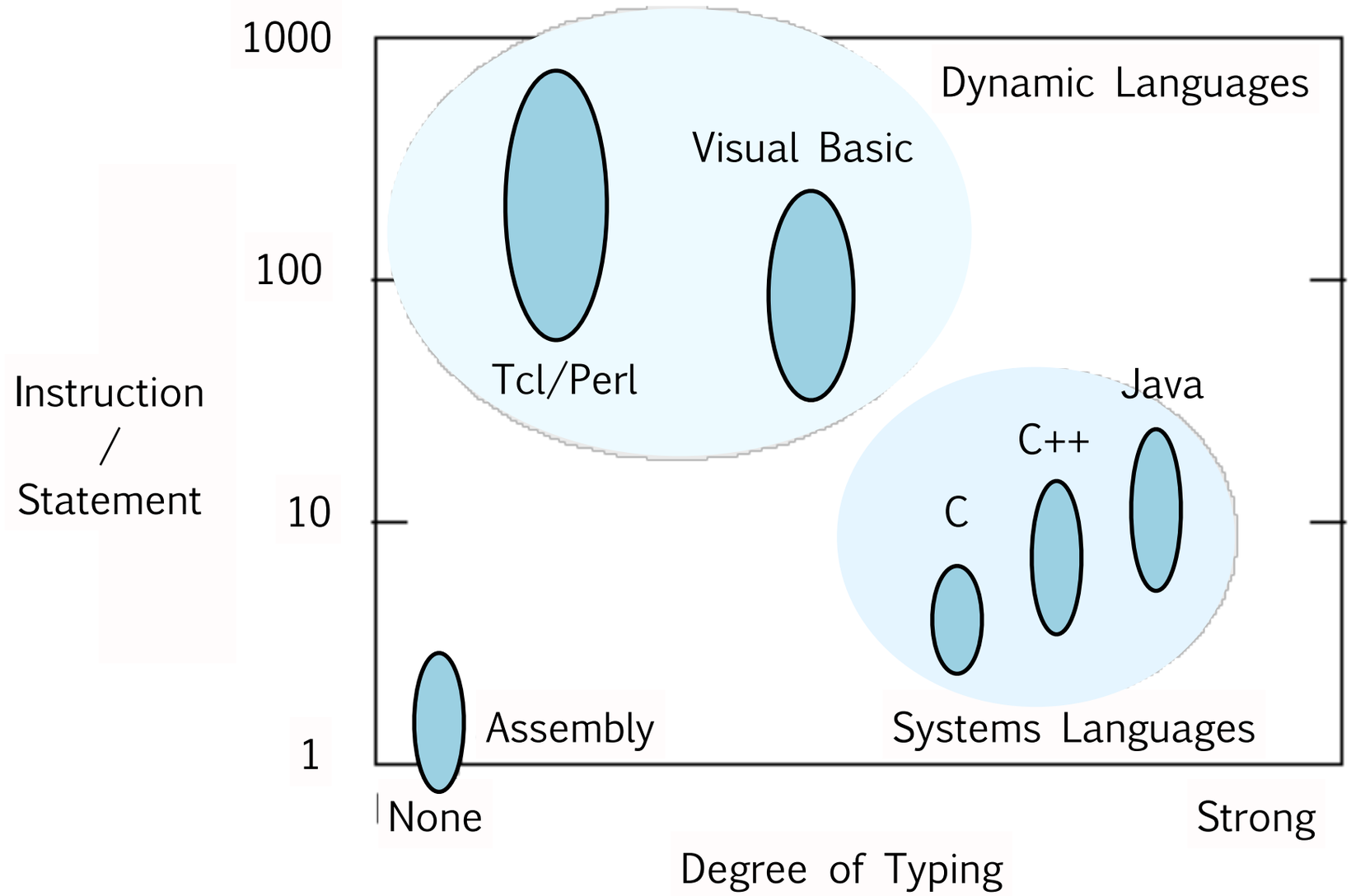
- Decides how a type is enforced, or interpreted
- In a weakly typed language, variables can be coerced easily -- interpreted as something else



```
>> 1+"2"
TypeError: String can't be coerced into Fixnum
     from (irb):1:in `+'
     from (irb):1
>> "1"+2
TypeError: can't convert Fixnum into String
     from (irb):2:in `+'
     from (irb):2
```

```
>>> "1"+2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> 1+"2"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

High-level



Slightly modified illustration from [1]

Productive

Briefly: LOC

- LOC/hour is invariant of PL [11, 2 via 1]
- 1 LOC C ≈ 3–7 assembly instr. [1]
- 1 LOC Python ≈ 3 LOC C [11]
- 1 LOC TCL ≈ 3–47 LOC C [1]

PLs Comparison

- “An Empirical Comparison of Seven Programming Languages”, Lutz Prechelt [11]
- Designing and writing the program in Perl, Python, Rexx, or Tcl takes no more than half as much time as writing it in C, C++, or Java — and the resulting program is only half as long.

```
`cat wordlist`.split("\n").collect do |w|
  w.strip if w =~ /^#{match}/
end.compact
```

Interpreted (rather than compiled)

Interpreted Languages

- Interpreting is the process of mapping encoded instructions to some actual operations associated with them
- This way interpretation and execution are basically the same thing -- the only difference is the level at which the interpretation is done
- Most interpreted languages combine both compilation and interpretation

Interpreted Languages

- Platform independence
- Reflective usage of the evaluator
- Dynamic typing
- Ease of debugging

Flexible at run-time

What is run-time?

Extensible through metaprogramming

```
2.weeks + 4.days + 3.minutes + 8.seconds
3.weeks.from_today.on_sunday
```

```
# Augment the built-in classes
# (As numeric is already defined --
# it is standard--
# these methods are added to the
# Numeric class)

class Numeric
    def minutes; self * 60; end
    def hours; self * 60.minutes; end
    def days; self * 24.hours; end
    # etc.
end

# A time interval
3.years + 13.days + 2.hours

# Four months from now, on a Monday
4.months.from_now.next_week.monday
```

Fun

REPL?

Garbage collected

Suitable for non-programmers also

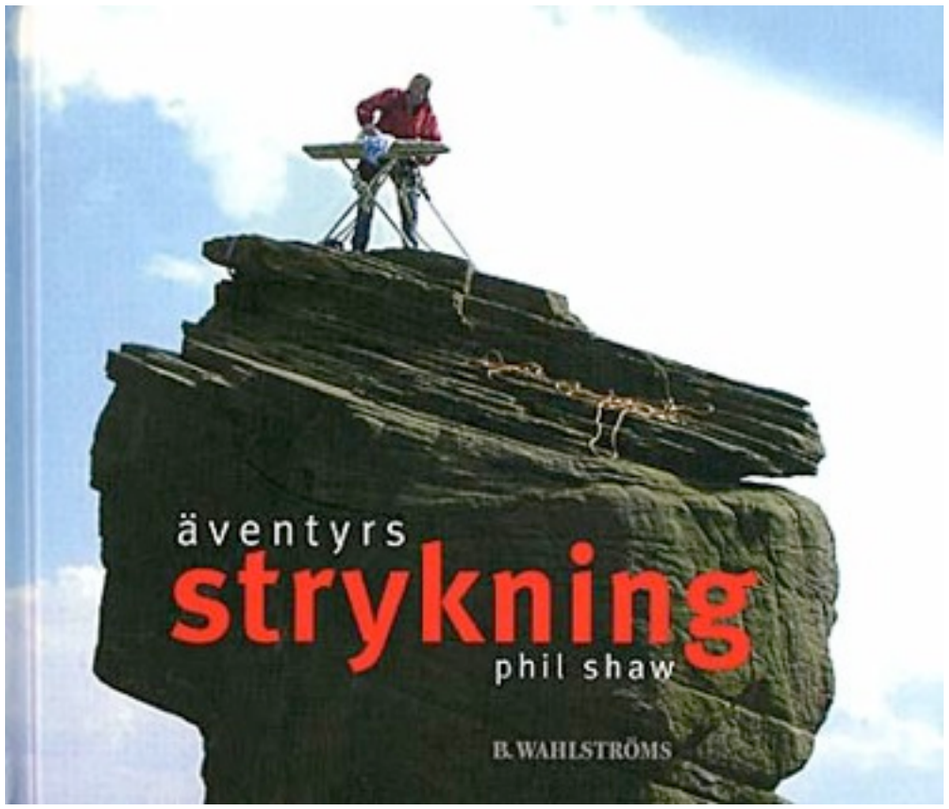
Pragmatic

Why are DYPLs not as successful as system PLs?

Origin of Languages

- Statically typed languages:

- Dynamically typed languages:



System PLs

- Optimisation
- Fine-grained resource control
- Static checking
- Typing is documentation
- Conservative is safe

John K. Ousterhout

- Scripting languages are designed for “gluing” applications; they use type- less approaches to achieve a higher level of programming and more rapid application development than system programming languages. [1]

[DYPLs are] old in years but young in maturity [1]

Use DYPLs when [1]

- mainly connecting components
- manipulating many different things
- needing a GUI
- rapid evolution is expected
- extensibility is important

Don’t use DYPLs [1]

- when complex data structures and algorithms are needed
- when manipulating large datasets
- under heavy time-constraints
- when all functions are well-defined and slow evolution is expected

David Ascher [2]

Dynamic languages:

- Technical "purity" — not pushing X
- Optimising person time, not machine time
- Open source rooted
- Evolution by meritocracy and natural selection
- Platform neutrality

Use DYPLs for [2]

- Scripting tasks
- Prototyping
- When needing loose coupling (distributed programs)
- Business logic

Don’t use DYPLs [2]

- when you only have a little memory
- for (some) high-performance tasks

Myths about DYPLs [2]

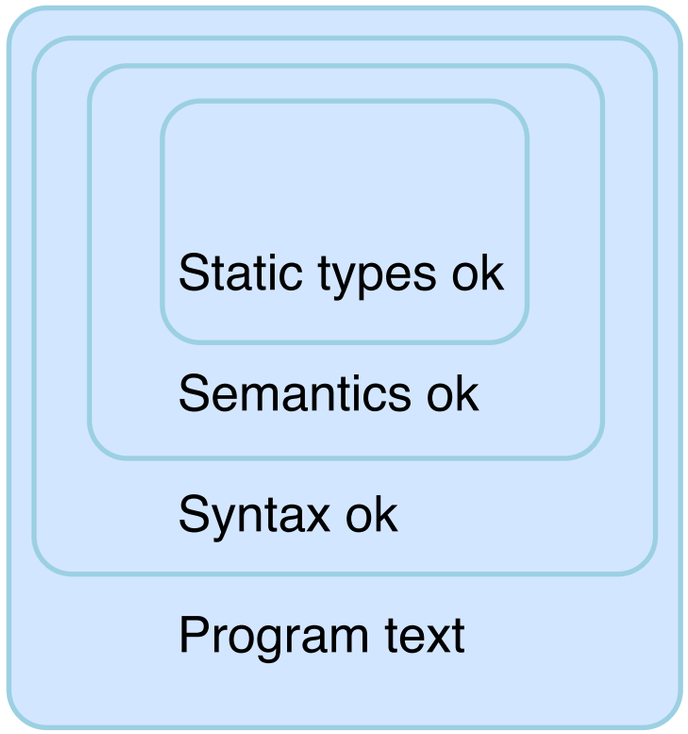
- Cannot be used for real applications
- They are brittle
- Cannot be used for large systems

Myths about DYPLs [2]

- They are not well-supported
- There are no good tools
- Not fitting for JVM, CLR, etc.

Sam Wilmott

- Dynamic programming languages are especially useful for trying out new ideas [3]


Static types ok
Semantics ok
Syntax ok
Program text

David Ungar

- [...] the system gets out of the way, and the true creative abilities of your users are let loose to make magic. That's why I believe in dynamic languages. [4]

Bruce Tate [12]

- The change from Java to the “next language” is imminent
- Ruby is a likely candidate
- DYPLs rarely need things like AOP
- Java was once like kayaking

Will DYPLs ever be mainstream?

Some that made it

- (LISP)
- (Smalltalk)
- JavaScript
- Perl
- PHP

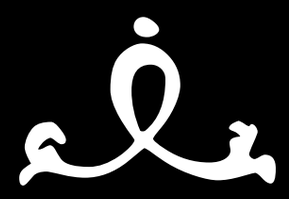
Growing Support

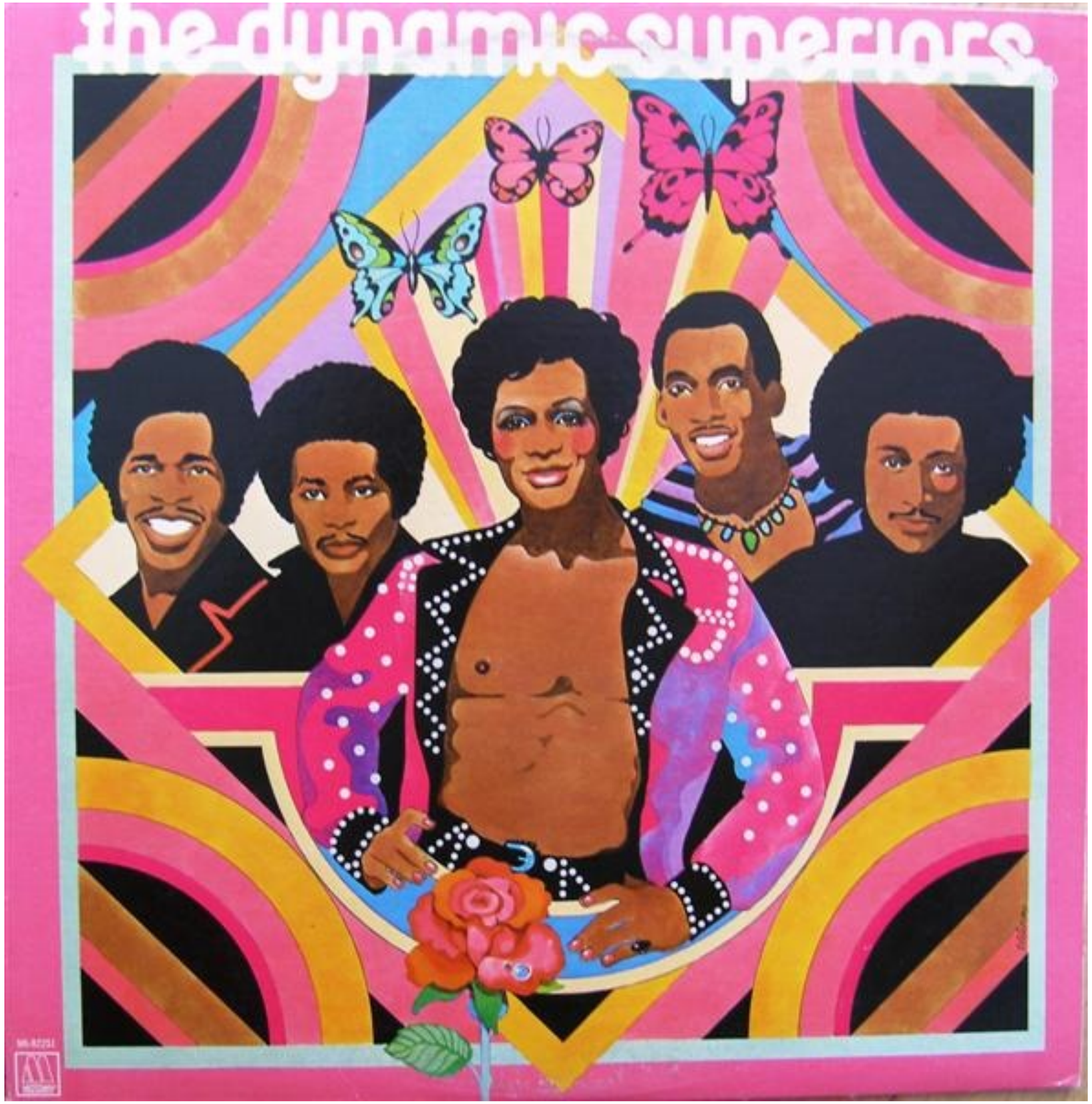
Bye to system PLs?

- Who needs C/C++ anymore?
- JVM is the new OS

Programmers were reluctant to move to macro assembler

Questions?




the dynamic superiors

The End

References

[1] John K. Ousterhout, *Scripting: Higher Level Programming for the 21st Century*

[2] David Ascher, *Dynamic Languages, Ready for the Next Challenges, by Design*

[3] Sam Wilmott, *When Is A Dynamic Programming Language Not Dynamic*

[4] David Ungar, *Why I believe in dynamic languages*

[11] Lutz Prechelt, *An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program*

[12] Bruce Tate, *Beyond Java*

(Reference numbers corresponds to number in Article Index on the course's web page)