Programming (ERIM)

Lecture 1: Introduction to programming paradigms and typing systems

Tommi Tervonen

Econometric Institute, Erasmus School of Economics

Course learning objectives

After this course, you should be able to:

- Understand and use basic constructs of procedural weakly typed programming languages (such as Matlab, R and Python)
- Program simple computational tests and model estimation algorithms
- Visualize test results
- Code according to the "contract programming" approach

Course organization

- 8 lectures (weeks 1-4, 6-8, maybe 10)
  - Theoretical contents
  - Provide background for the exercises
- 8 exercise sessions (weeks 2-9)
  - 7 exercises done individually or in pairs
  - Come to exercises to ask questions and get help with your code

Study load

- 4 ECTS = 112h
- 8 lectures = 8h
- 8 exercise sessions = 16h
- $\Rightarrow$ Independent programming 85h $\approx$ 11 h/w

Grading

- Exercises: 100% (14.29% each)
  - Done in pairs or individually
  - Exercises will be online at the beginning of the lecture
  - Strict deadline on Sundays @ 23.59 (fourth exercise will run for 2 weeks)
  - Submission via BB: *only* the source file(s) in the root of a zip. Include a comment in the beginning with your name(s) and student number(s)

Plagiarism

- Do not submit anything you haven't written yourself
- Do not submit anything that is not your idea
- We will not give you answers in the tutorials, but merely help you to find the answer
- "But I could've solved this problem myself, it was just faster to google the solution"

Course staff

- Me
- You! Participate in course discussion forums in BB to get and provide help with the exercises

Course contents

Week 1 Introduction to programming paradigms and weakly typed languages

- Practicalities
- Programming paradigms
- Scripting languages
- Types and variables

W2 Control flow, branching, loop constructs

W3 Subroutines and scoping

W4 Side effects, functions and procedures

—

W6 Programming by contract

W7 Test-driven development

W8 Anonymous functions and function references

—

W10 Version control

Literature

- Matlab book can be useful to own for the Matlab users
- R users: Introduction to programming with R (`http://cran.r-project.org/doc/manuals/R-intro.html`)
- LN-TT-22012-3 as background material, available @ `http://smaa.fi/tommi/courses/prog2/`
- All course material is posted in `http://smaa.fi/tommi/courses/erimprog/`, and links to exercises also in BB

Software

- The exercise sessions will be guided with Matlab or R
- For R users: RStudio (`http://www.rstudio.com/ide/download/desktop`)
- You can also do the exercises with Python or Octave (though visualization in Octave sucks)

Q?

"The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague."

E.W. Dijkstra

Programming paradigms

- Programming paradigms refer to the philosophy behind designing programming languages

- When you know to program with 1 language of a paradigm, others of the same paradigm are easy to learn (mostly just syntax)

Programming paradigms

1. Procedural / imperative paradigm (C, Pascal, Matlab, R, Fortran, Algol, Python)

2. Object-oriented paradigm (Java, Smalltalk, C++ partially)

3. Declarative paradigm, including
   - Functional programming (ML, Lisp, Haskell, Erlang, Scala, Scheme)
   - Logic programming (Prolog)

Our first program: hello world

```
disp('Hello World!');
```

```
message('Hello World!')
```

- Single-line script
- Contains a single *statement*
- Calls function `disp` (Matlab) / `message` (R) with parameter 'Hello World!'
- 'Hello World!' is a string

Compilation of languages

- Before source code can be executed, it needs to be *compiled* into an executable format

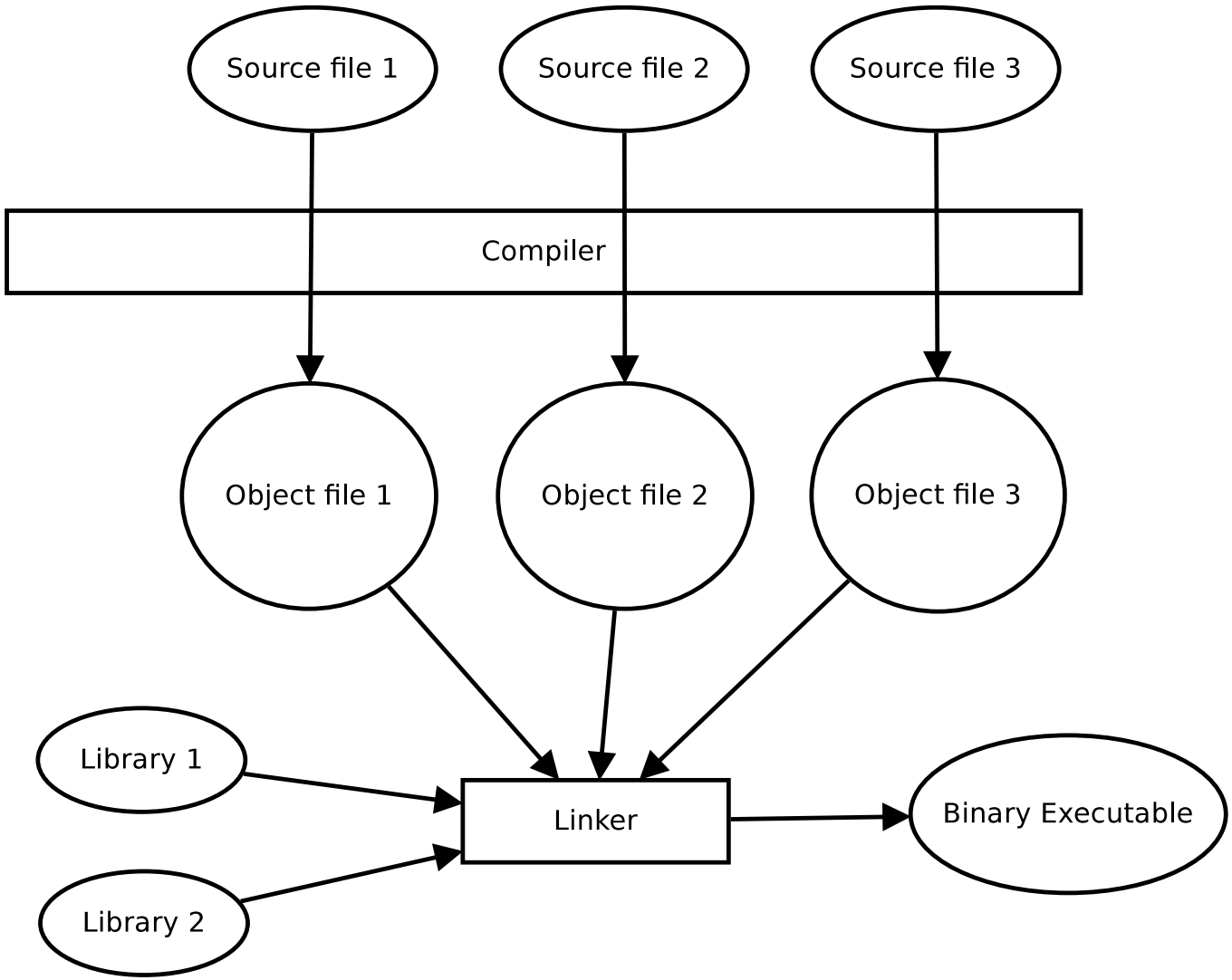
- The compilation can be made

  1. Completely in advance to a binary executable (fast)

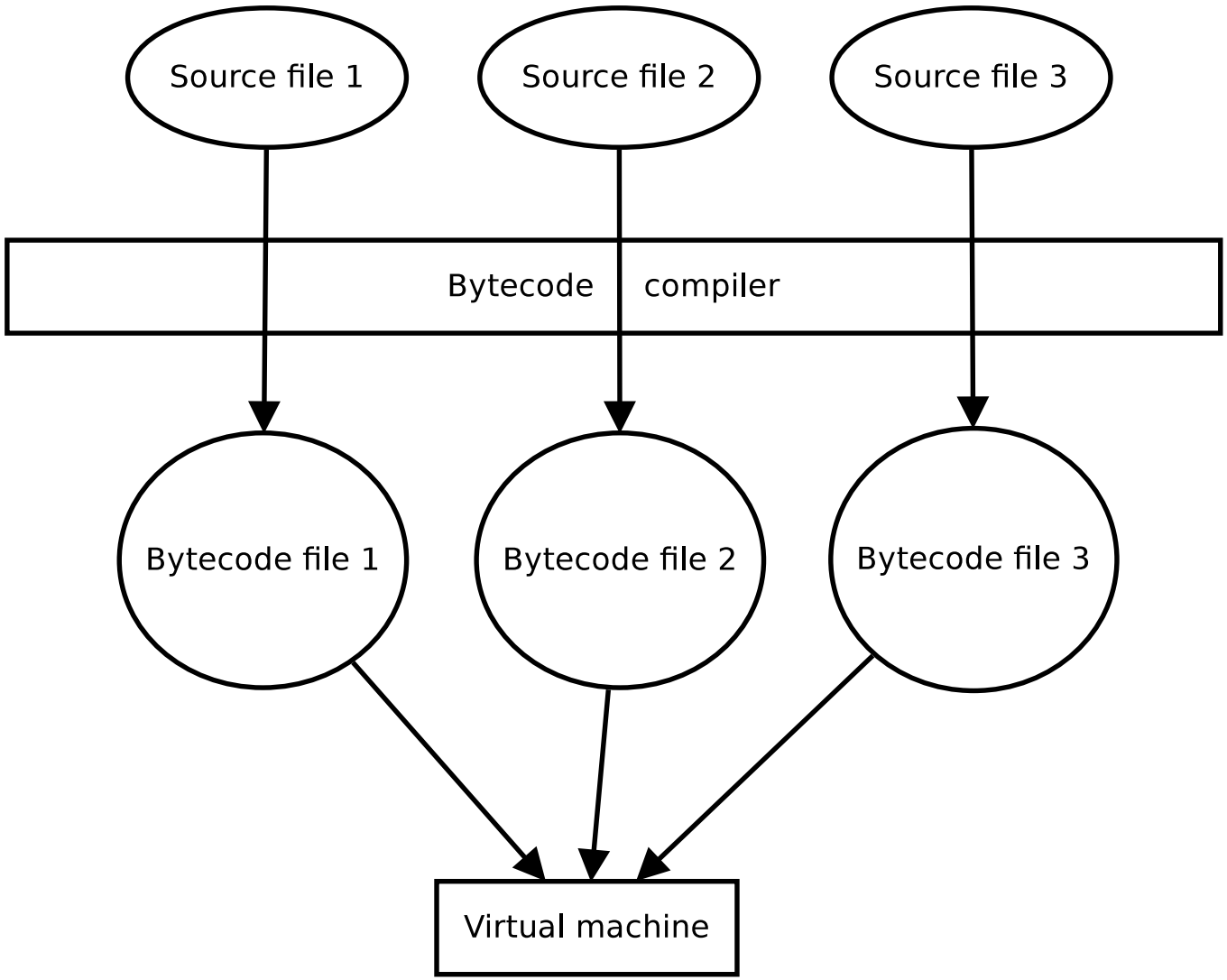
  2. Partially in advance to bytecode to be executed in a virtual machine (Java, quite fast and portable)

  3. Run-time (slow but allows easy "modify & execute" cycles)

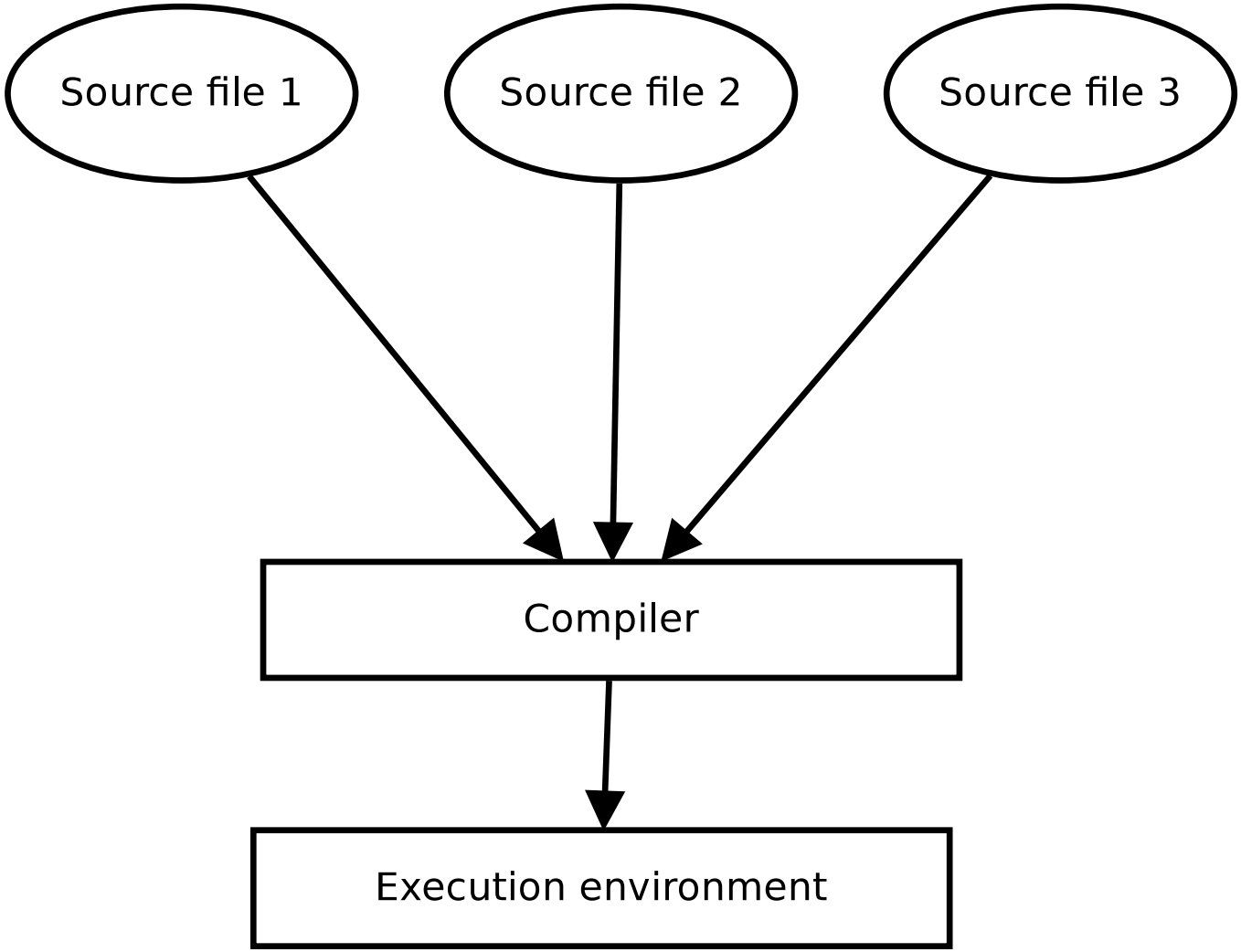
Fully compiled languages (e.g. C)



Bytecode compiled languages (e.g. Java)



Runtime compiled languages (e.g. Matlab)



Scripting languages

- In scripting languages the instructions are compiled run-time into execution statements
- Slow, as less optimization can be made
- In languages of statistical / scientific computation, you have to understand what happens "under the hood" to make efficient *and* correct code

Our second program: store and print variables

```
x = 2;
y = (x + 2) * 2;
x = y + 2 * 2; % y + 4
x
```

```
x <- 2
y <- (x + 2) * 2
x <- y + 2 * 2 # y + 4
print(x)
```

- Script with four statements
- Statements executed one by one from top to bottom
- *Variable* `x` is *declared* and a value 2 is *assigned* to it
- (x + 2) * 2 is an *expression* which is evaluated and its result assigned to `x`
- Operation precedence: assignment is always the last, multiplication/division before addition/substraction

Introduction to types

- Typing systems form the core of programming languages - they allow construction of abstractions

- Differences in electric currency $\rightarrow$ bits $\rightarrow$ numbers $\rightarrow$ characters $\rightarrow$ data records/structures

Types in Matlab / R

- Integers: `x = 2; x <- 2`
- Floating point numbers: `x = 4.123; x <- 4.123`
- Strings: `x = 'my string'; x <- 'my string'`
- Arrays: `x = [1 2 3]; x <- c(1, 2, 3)`
- Matrices / Matlab: `x = [ 1 2 3; 4 5 6];`
- Matrices / R:

```
x <- matrix(c(1, 2, 3, 4, 5, 6),
ncol=2, byrow=TRUE)
```

Strong and weak typing

Strong typing: each variable has a type associated with it

```
int x = 2; // ok
x = 3; // ok
x = 's'; // error
```

Weak typing : a single variable can be assigned varying types of values

```
y = 3; % ok - no type declaration required
y = 't'; % ok
```

Type conversion in Matlab

- Matlab is a weakly typed language, and the following are valid expressions:

```
x = 1;
y = '1';
z = x + y;
```

- Now z = ?

Next

- This week there's no exercise
- Next week onwards: 1h lecture followed by 2h exercise
- Make sure you can run RStudio/Matlab/whatever