Preparation for the forthcoming **semester-project**

COS**341**

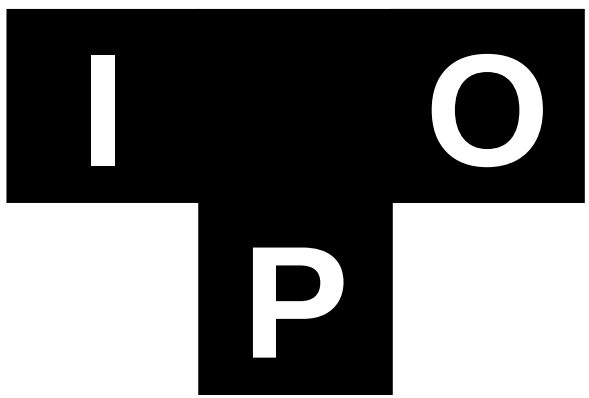
Academic Year 2020

Preparation for the forthcoming **semester-project**

Compilers are Software Systems

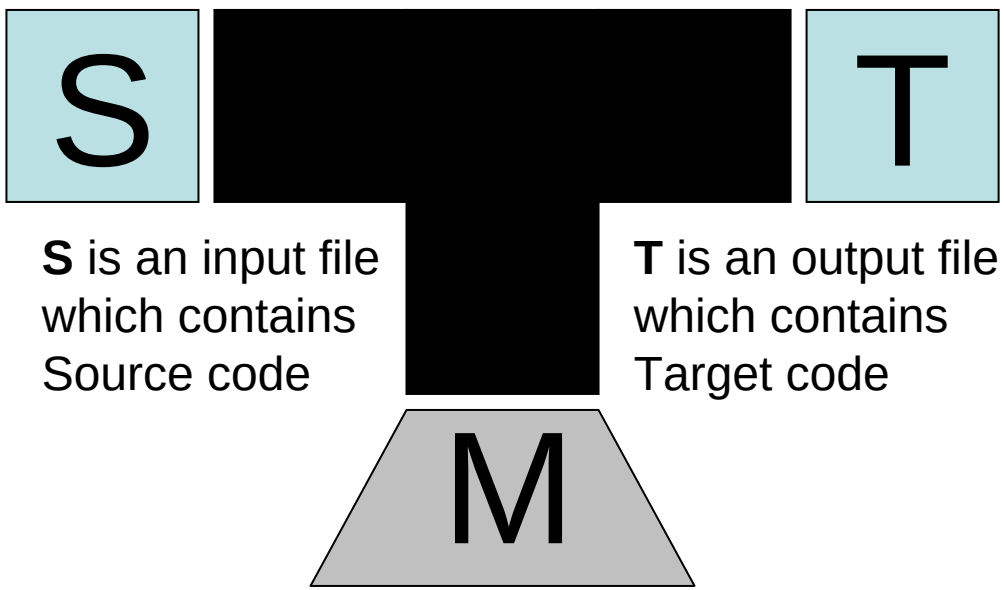
In the well-known “**T-Notation**”, the “**T**”-shape represents a **compiler software system**.

Preparation for the forthcoming **semester-project**



I represents the Input language, for which the compiler shall generate output.
O represents the Output language, which the compiler emits.
P is the programming language, in which this compiler itself is implemented.

Preparation for the forthcoming **semester-project**



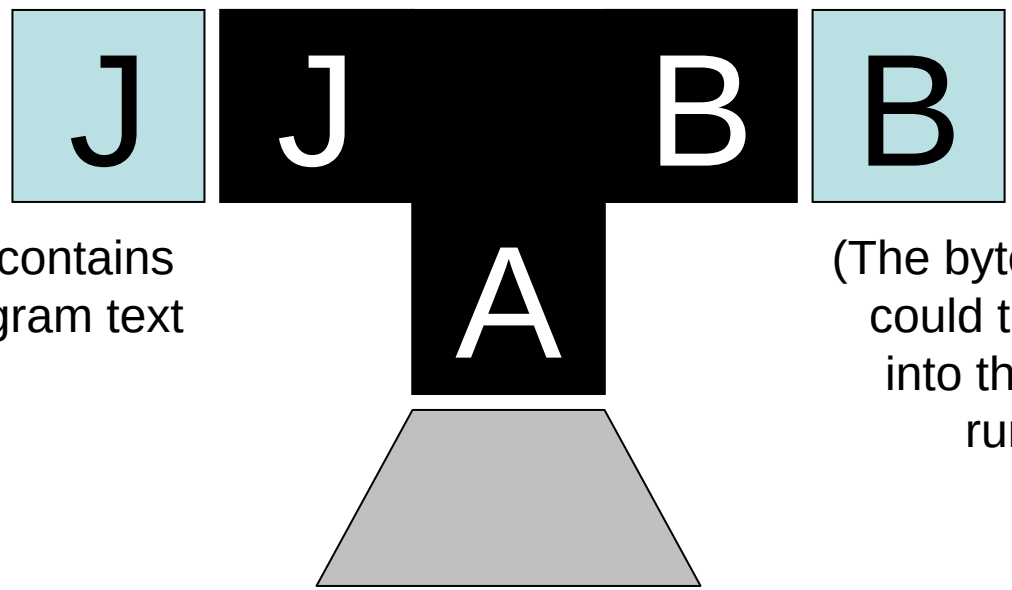
S is an input file which contains Source code

T is an output file which contains Target code

M is a machine (hardware) on which the compiler itself is “running”

Preparation for the forthcoming **semester-project**

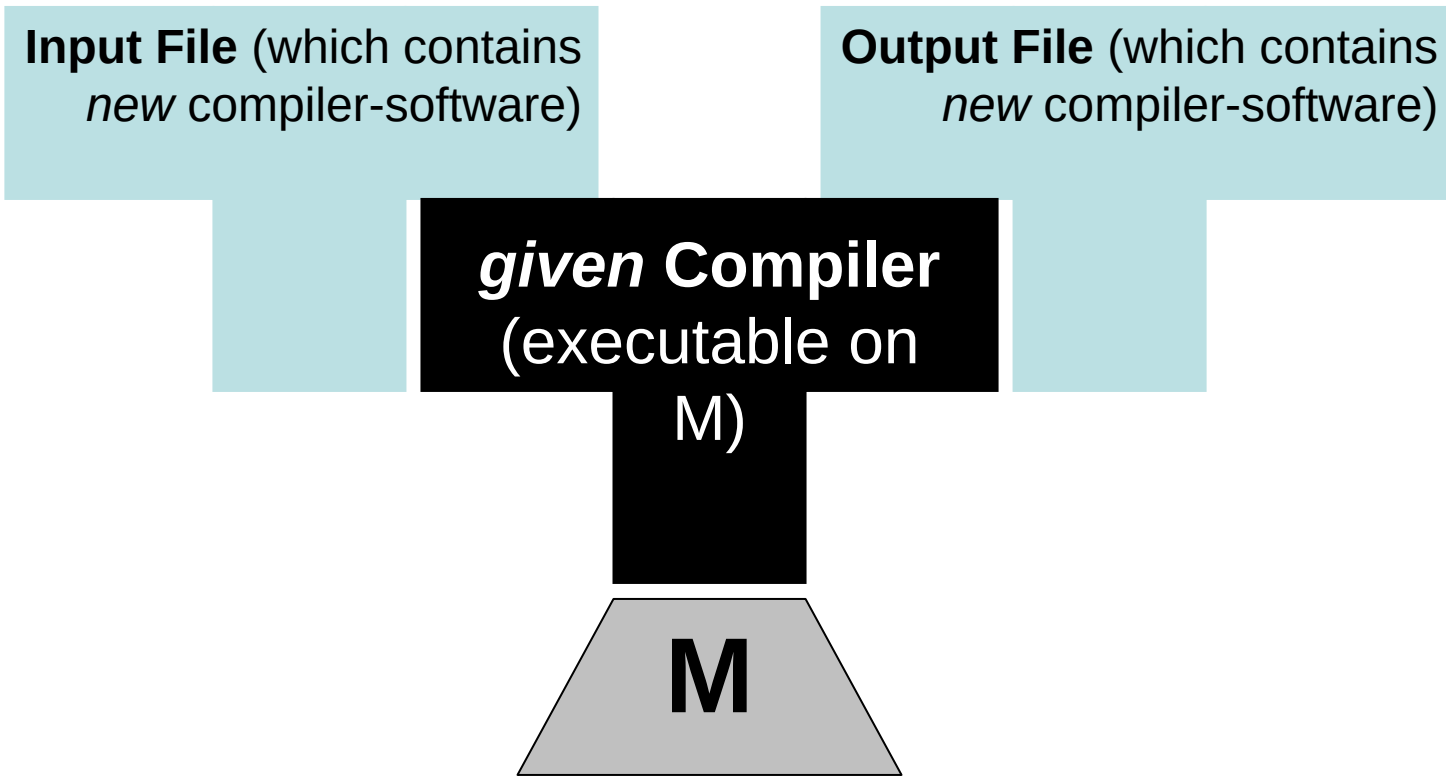
EXAMPLE:
Compiler (in **T-Notation**),
itself implemented in executable Assembler code (A),
which “eats” a Java program file (J)
and emits a Byte-code file (B)



File which contains
a Java program text

(The byte-code file B
could then be fed
into the JVM for
running)

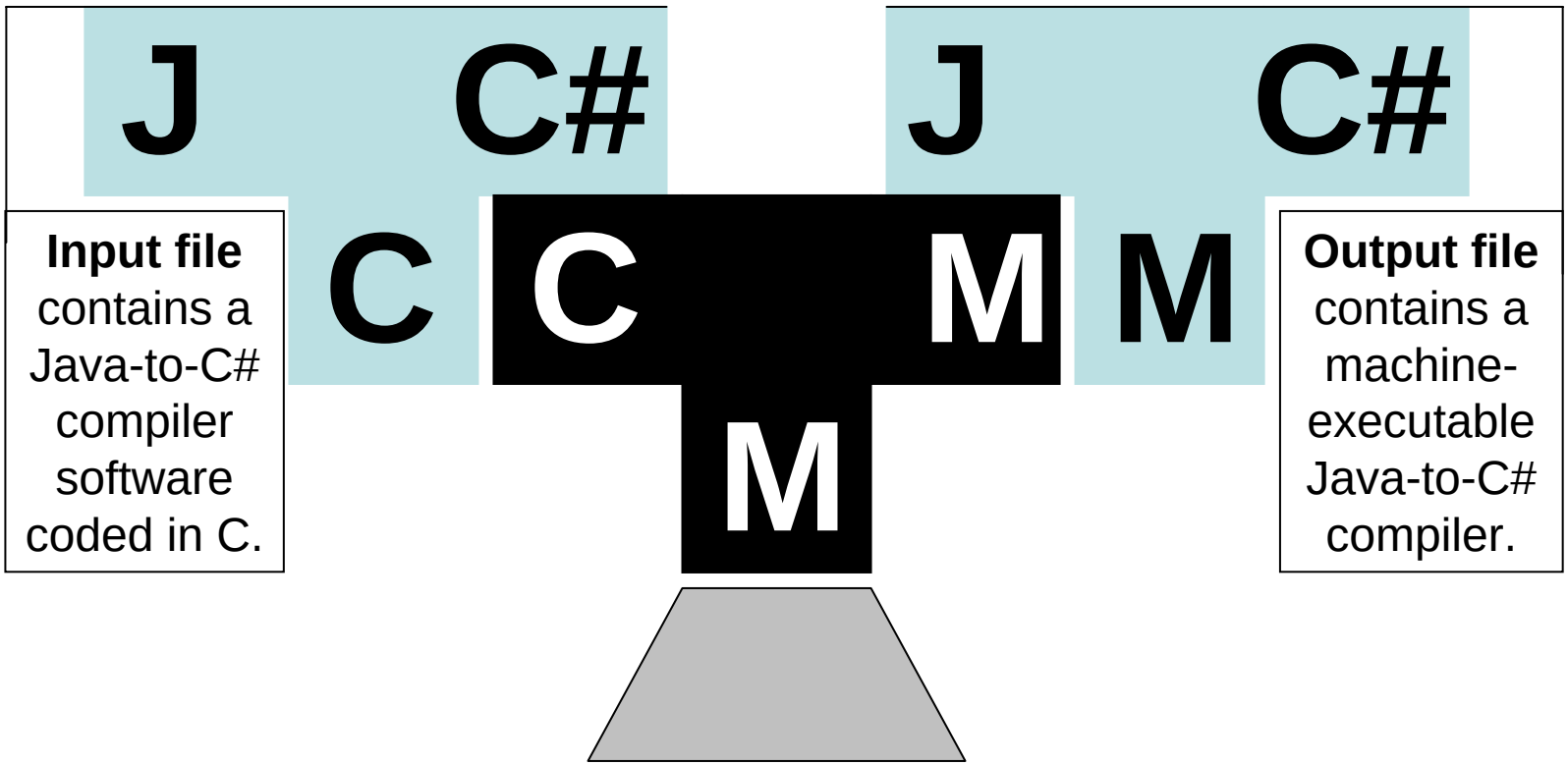
Preparation for the forthcoming **semester-project**



Because compilers are SOFTWARE, a compiler's input and output files can (of course) contain "code"-text which "implements" other compilers.

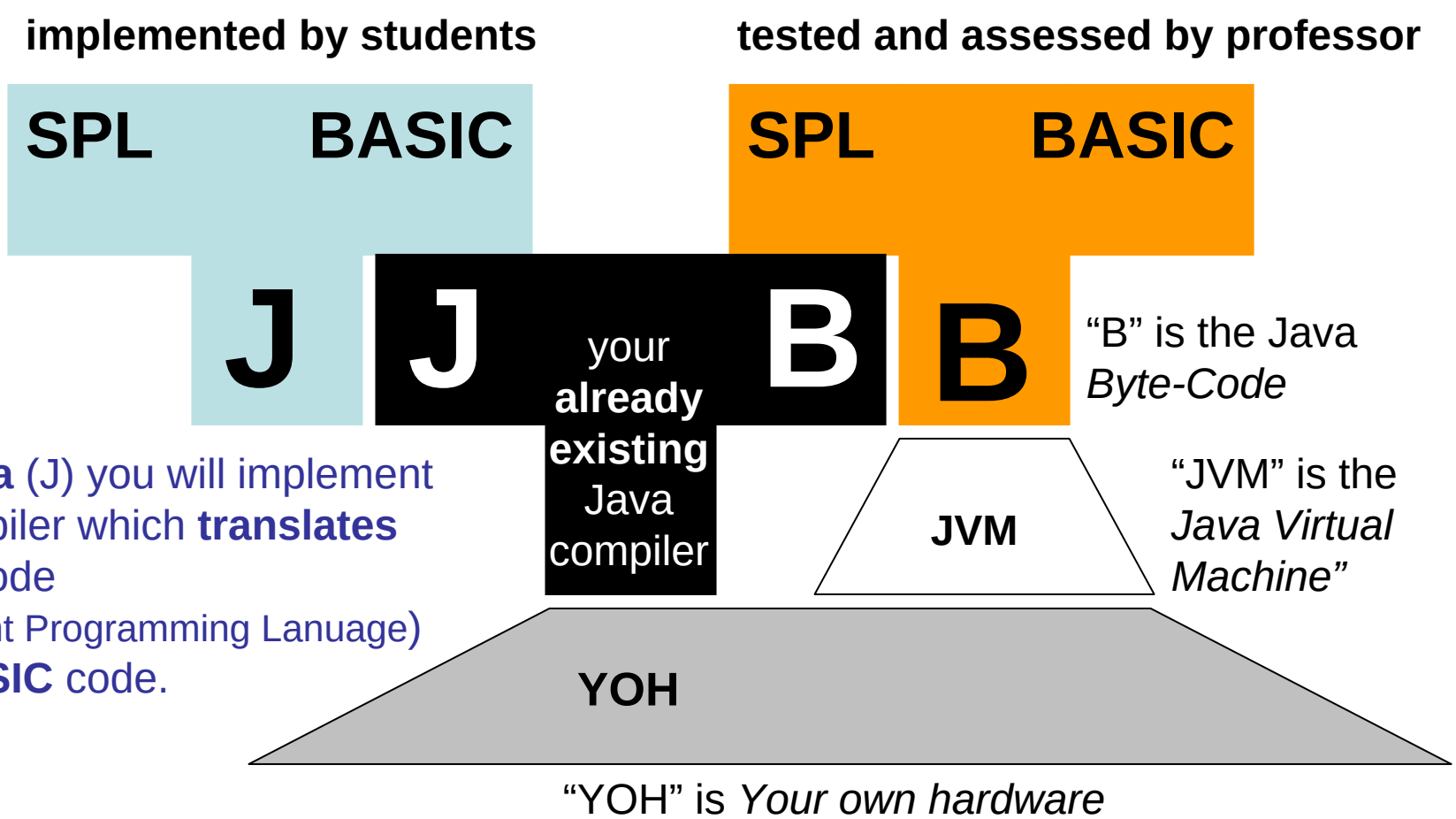
Preparation for the forthcoming **semester-project**

EXAMPLE



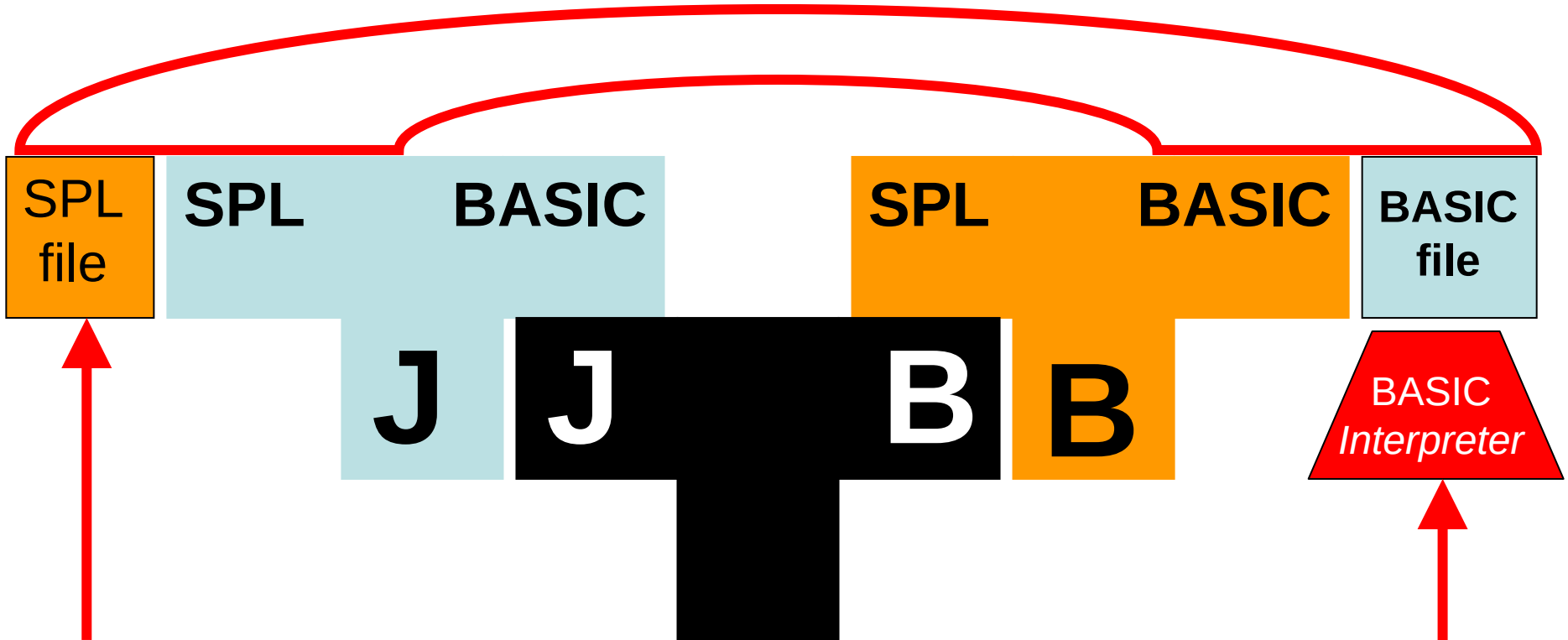
In this example, an **already existing** (machine-executable) **“C”-compiler** is used to create a machine-executable compiler that produces “**C#**” out of “**Java**”(**J**) code.

Your forthcoming **semester-project** will be structured as follows:



In Java (J) you will implement a compiler which **translates** **SPL** code (Student Programming Lanuage) **to BASIC** code.

Your forthcoming **semester-project** will be *assessed* as follows:



The professor will provide SPL programs which your compiler must "eat" and compile...

The professor will use a BASIC *interpreter* to test if the BASIC programs generated by your SPL-to-BASIC compiler can run and function correctly.

Features of the **semester-project** will include:

- **FRONT END**
  - **Lexical Analysis**: *RegExpr ➔ NFA ➔ DFA ➔ MinDFA ➔ Matching Strategy*
  - **Syntax Analysis**: *Removing ambiguity from Grammar, and selecting an appropriate parsing strategy for this grammar's class*
  - **Static Semantic Analysis**: *Type checking, identification of un-defined or un-used variables, scoping analysis (which variable is in which scope), etc...*
- **BACK END**
  - **Generation** *of BASIC code*
  - **Code Analysis** *to identify the minimum number of needed registers, as well as to identify possibilities of code optimisation*
  - **Code Optimisation** *to produce "better" BASIC code from the initially generated "sub-optimal" BASIC code.*