Aalto University

Compiler Writing

Lecture 10

Part 2

Compilers

- Source Language Issues
  - Size of the source language (bigger = harder)
  - Extent of change during compiler construction (more changes = harder)
- Performance Criteria
  - Compiler Speed
  - Code Quality
  - Error Diagnostics
  - Portability
  - Maintainability

Performance Criteria

- Portability
  - Retargetability
  - Rehostability

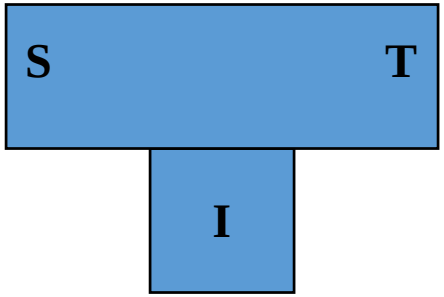
- A Retargetable compiler is one that can be modified easily to generate code for a new target language

- A rehostable compiler is one that can be moved easily to run on a new machine

- A portable compiler may not be as efficient as a compiler designed for a specific machine, because we cannot make any specific assumption about the target machine

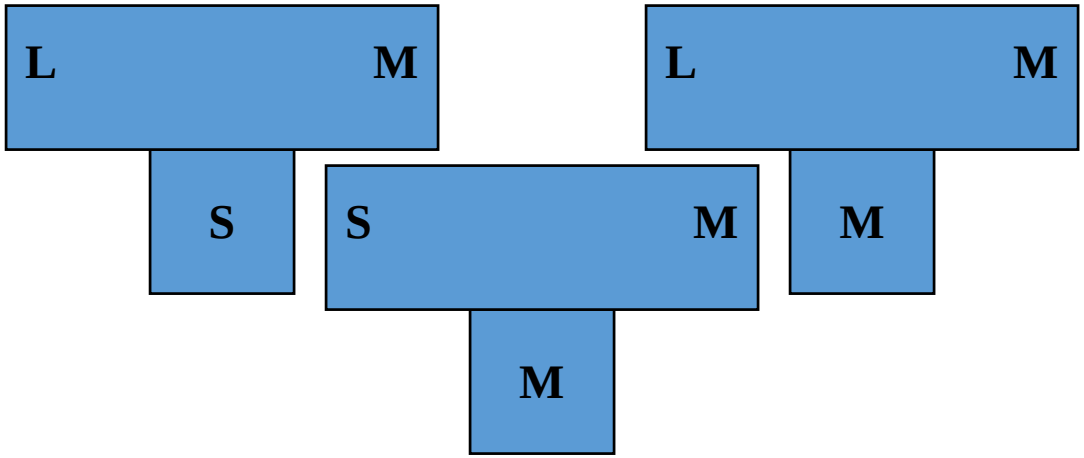
How was the first compiler compiled?

- *Bootstrapping*: using the facilities offered by a language to compile itself is essence of bootstrapping
- There are three languages involved in writing a compiler
  - Source Language (S)
  - Target Language (T)
  - Implementation Language (I)
- T-Diagram



Using Bootstrapping to Compile a Compiler

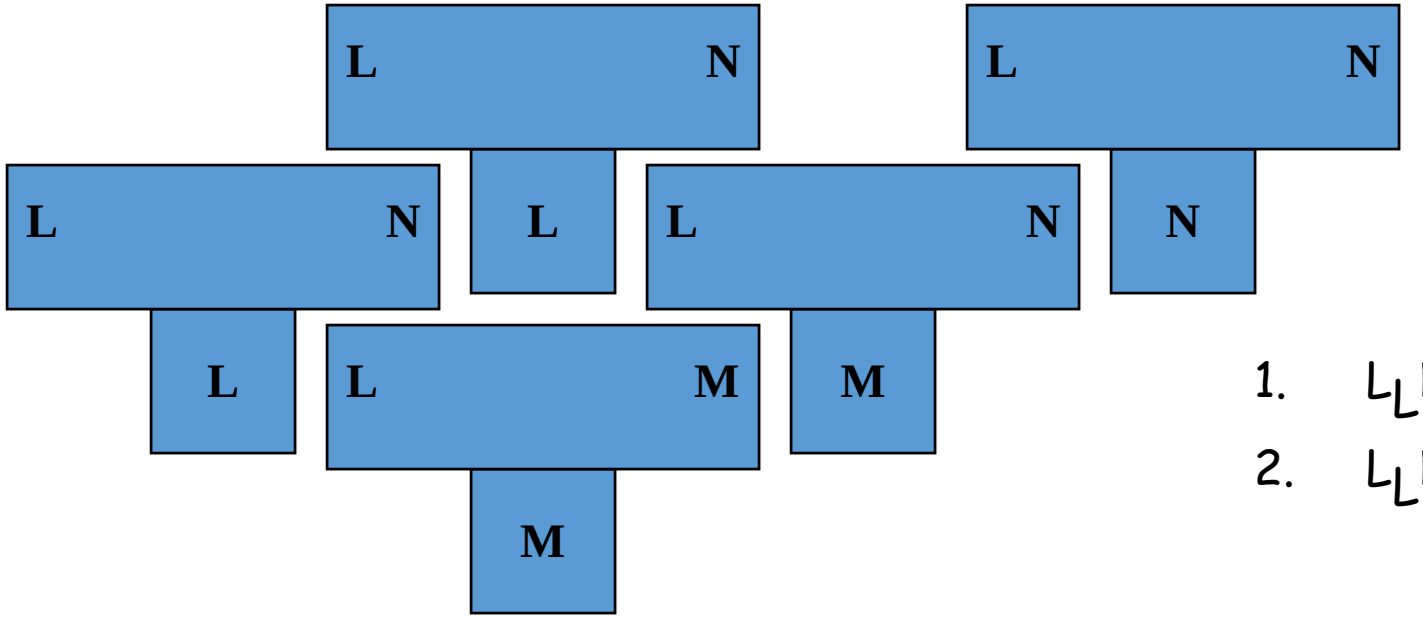
- L is a high level language
- S is a small subset of L, and M is a computer
- $S_M M$ is a running compiler produced using assembly language of machine M
- $L_S M$ is a compiler written in S, which translates L to executable code of M
- $L_M M$ is a running compiler produced by compiling $L_S M$ using $S_M M$



$L_S M + S_M M = L_M M$

Using Bootstrapping to Port a Compiler

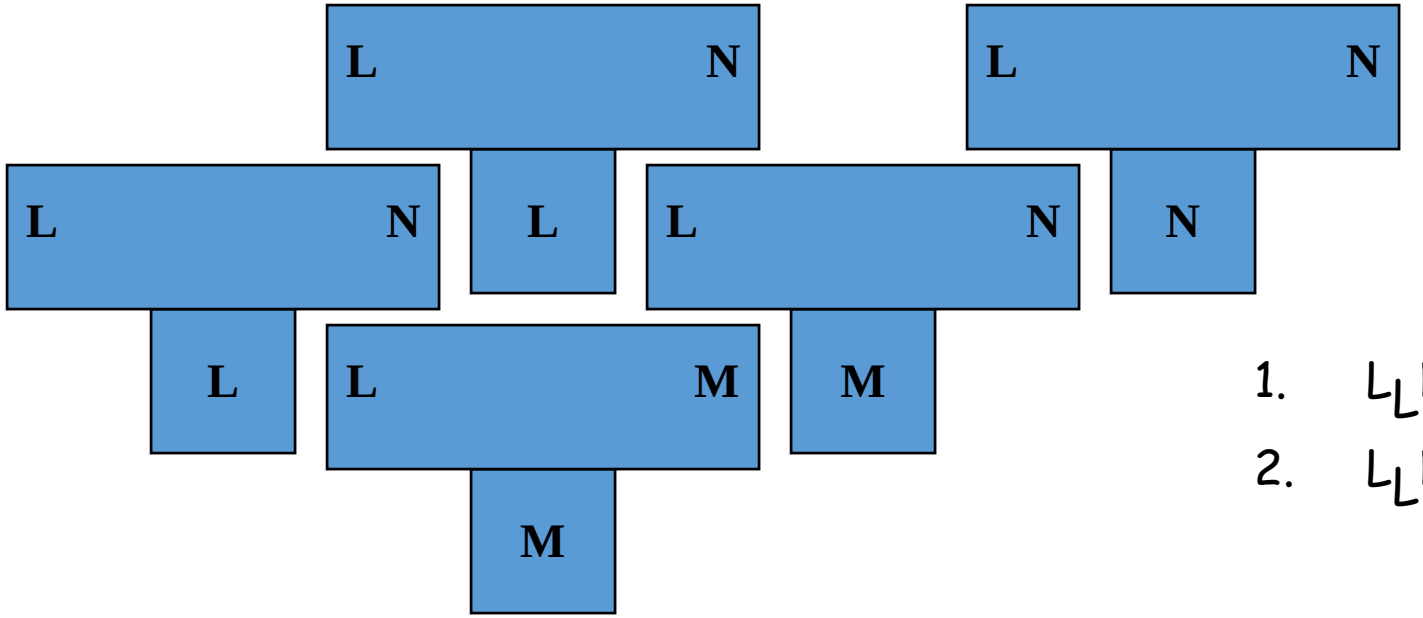
- L is a high level language, M and N are computers
- $L_M M$ is a running compiler, which can translate L to executable code of M
- $L_L N$ is a compiler written in L, which translates L to executable code of N
- $L_M N$ is a cross compiler produced by compiling $L_L N$ using $L_M M$
- $L_N N$ is a running compiler produced by compiling $L_L N$ again using $L_M N$



1. $L_L N + L_M M = L_M N$
2. $L_L N + L_M N = L_N N$

Using Bootstrapping to Optimize a Compiler

- L is a high level language, M slow executable code, and N fast executable code
- $L_M M$ is a running compiler, which can translate L to slow executable code of M
- $L_L N$ is a compiler written in L, which translates L to fast executable code of N
- $L_M N$ is a slow compiler that generate fast codes (by compiling $L_L N$ using $L_M M$)
- $L_N N$ is a fast compiler produced by compiling $L_L N$ again using $L_M N$



1. $L_L N + L_M M = L_M N$
2. $L_L N + L_M N = L_N N$