Back End Phases of Compiler

BITS Pilani
Pilani Campus

Shashank Gupta
Assistant Professor
Department of Computer Science and Information Systems

Code Generation

Abstraction at Source Code Level

- Identifiers
- Operators
- Expressions
- Statements
- Conditionals
- Iterations
- Functions

Abstractions at Target Code Level

- Memory Locations
- Registers
- Stack
- Opcodes
- System Libraries
- Interface to OS

Code Generation

It is a process of mapping from source level abstractions to target machine abstractions.

- However, a two-step process from transforming high level abstractions to low level abstractions.

Initially, Intermediate Code gets generated from the disambiguated parse tree.

Secondly, Machine code would be generated from this intermediate code representation.

Code Generation

Map identifiers to locations (Memory/Storage Allocation)

Map source code operators to opcodes or a sequence of opcodes.

Code Generation

Transform conditionals and iterations to a test/jump or compare instructions.

Layout Parameter Passing Protocols: locations for parameters, return values, etc.

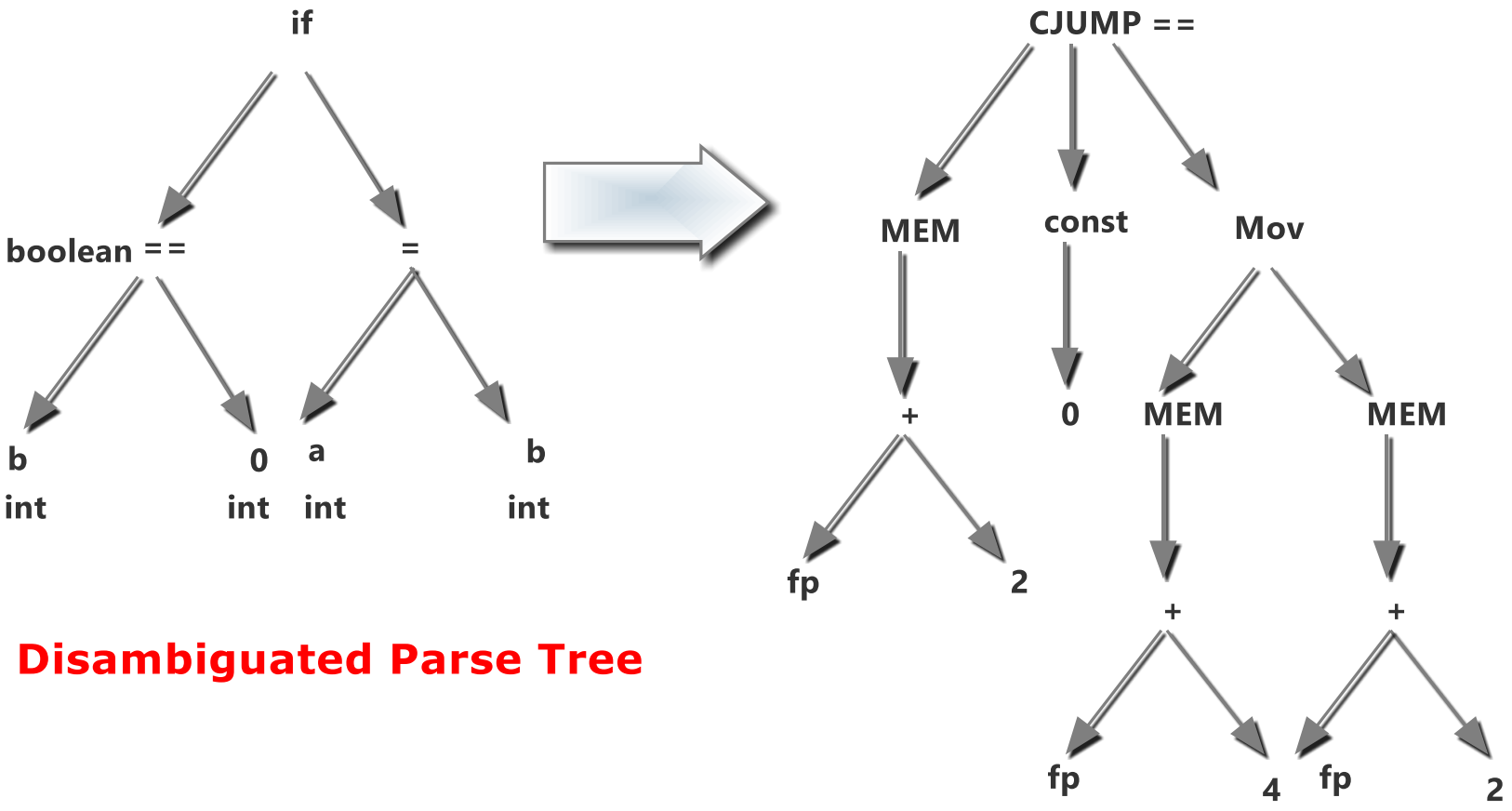
Post Translation Optimizations (Optional)

Remove/Simplify operations like

- Addition with 0
- Multiplication by 0
- Multiplication by 1

This is also know Machine-dependent code optimization.

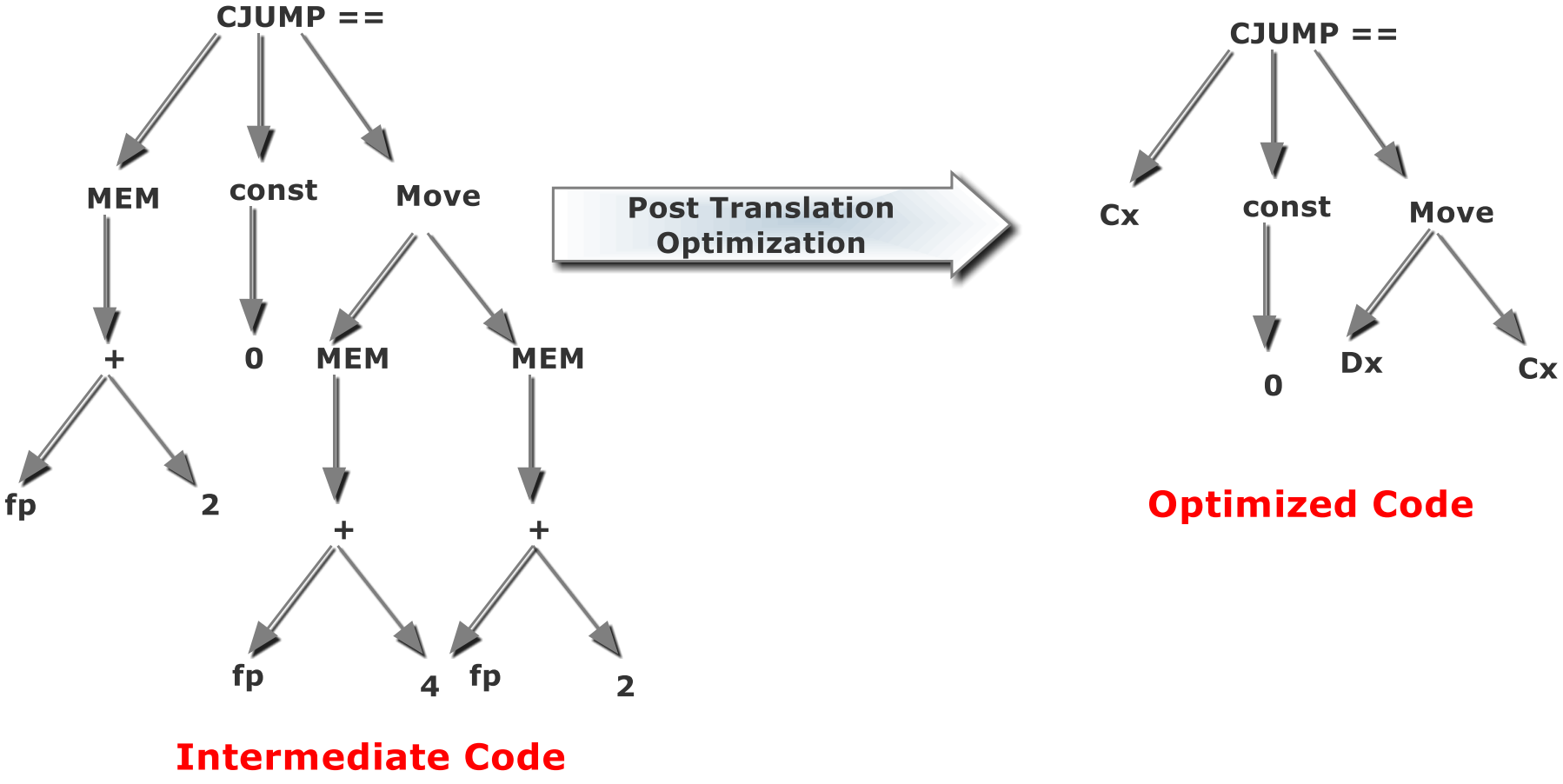
Example of Intermediate Code Generation



Disambiguated Parse Tree

Intermediate Code

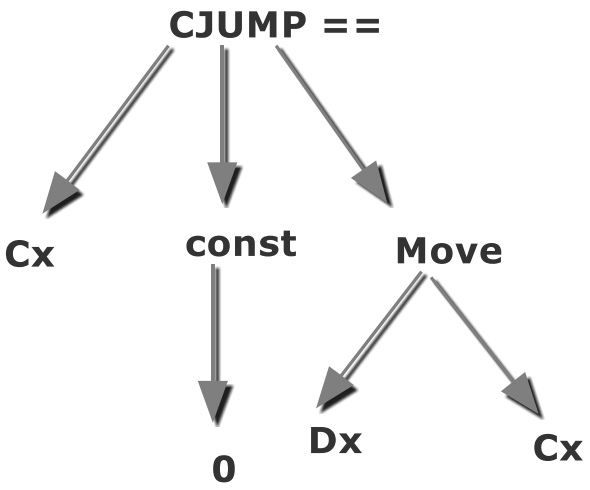
Optimization of Intermediate Code



Intermediate Code

Optimized Code

Target Code Generation

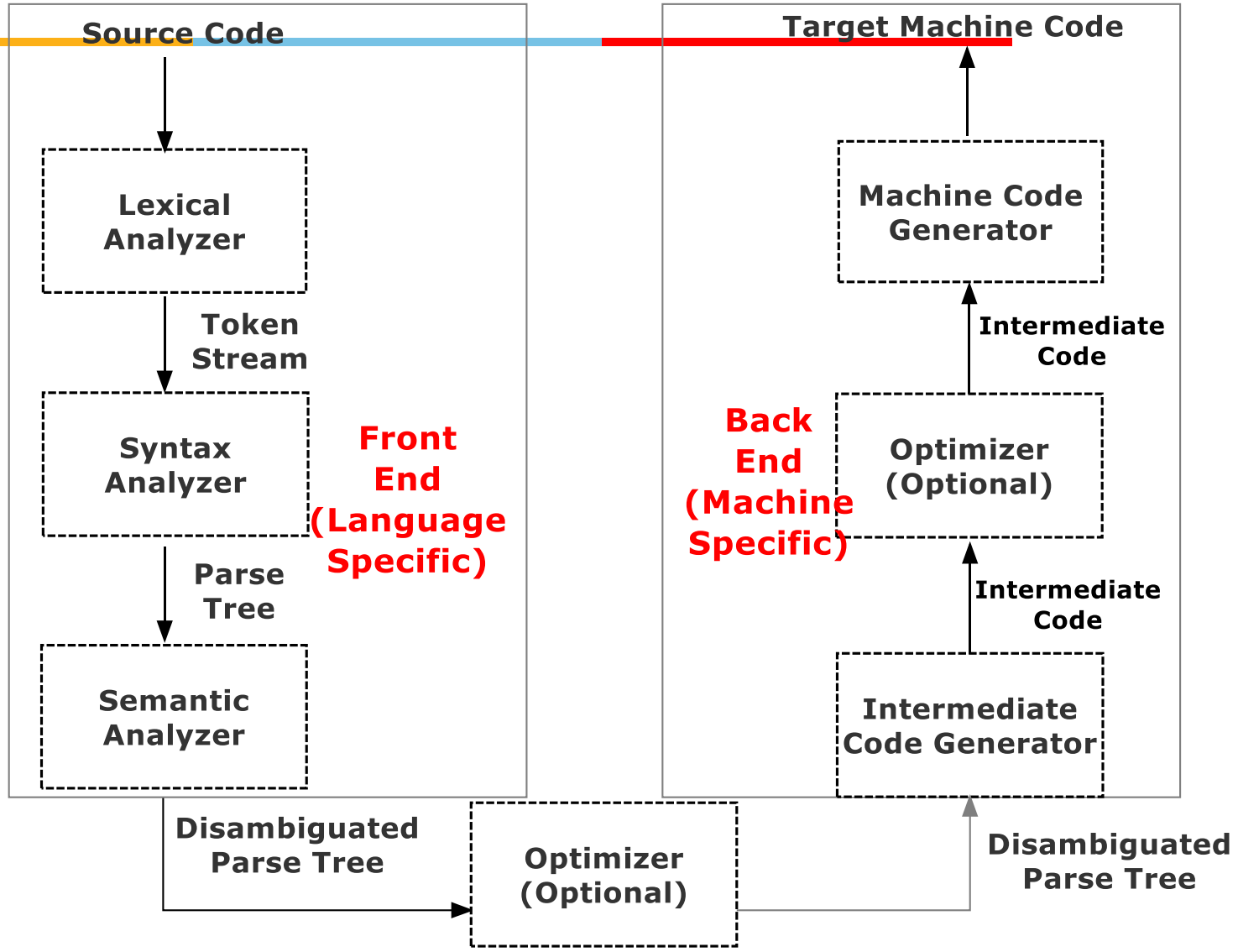


Optimized Code

```
CMP Cx, 0
CMOVZ Dx, Cx
```

Target Code

Compiler Structure



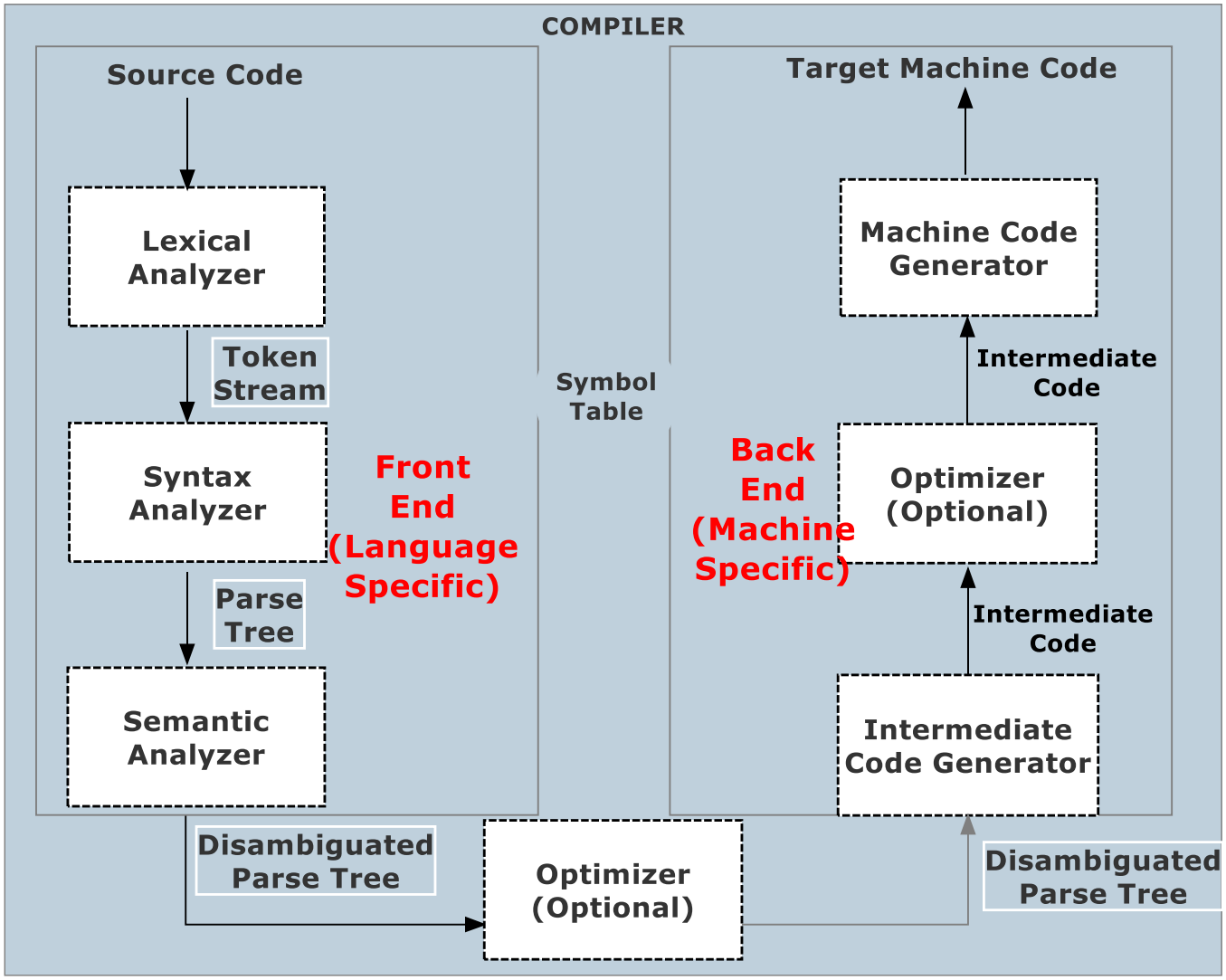
Symbol Table

Information required about the source program identifiers during compilation.

- **Category of Variable: Keyword, Identifier, etc.**
- **Data types of Variables: int, float, etc.**
- **Quantity of storage required**
- **Address in the Memory**
- **Scope Information**

A **Symbol Table** is a data structure that stores all this information in a central repository and every phase of the compiler refers to this repository whenever information is required.

Compiler Structure



Advantages of Compiler

Highly Modular in Nature.

It is re-targetable.

- If there is a single language and multiple machines, then same front end can be used for all machines.
- If there are many languages and single machine, then same back end can be used.

Advantages of Compiler (Continued….)

One can use part of the front end and part of the back end and the whole compiler can be re-targetable.

Source and machine independent code optimization is possible.

- On the other hand, some industries first focus on making the functional specifications of compiler correct and then they work on optimization.

Limitations of Compiler

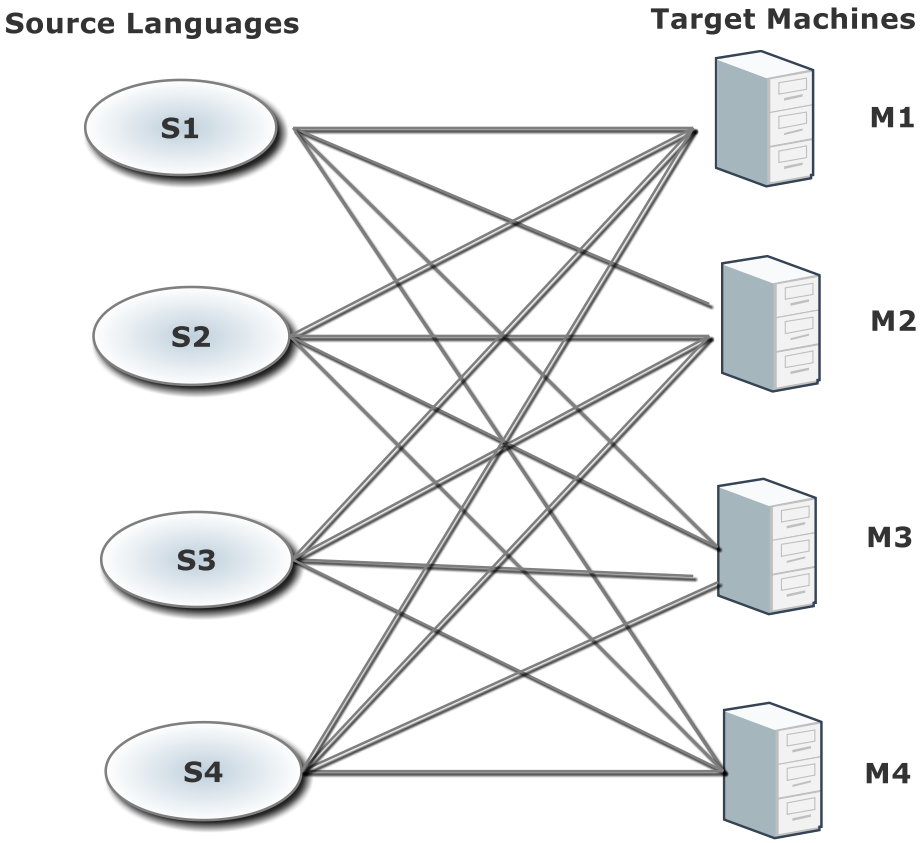
Design of programming languages has a big impact on the complexity of the compiler

Lot of work is repeatable.

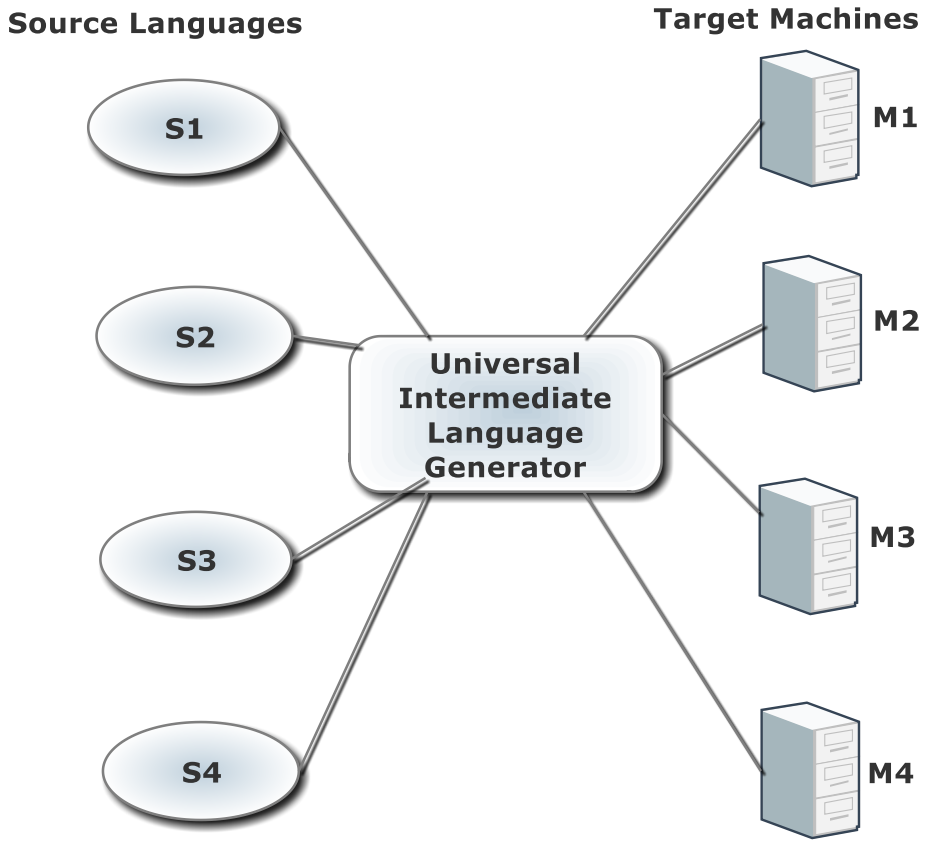
- For ‘S’ languages and ‘M’ target machines, we need to develop S*M Compilers.

S*M Problem of Compiler

S*M Problem



Resolve by S+M



Universal Intermediate Language was proposed in 1958 to reduce the development effort of compiling many different languages to different architectures.

Commonly used Machine Independent Intermediate Code Generation Techniques

Postfix Notation

Three Address Code

Syntax Tree

Directed Acyclic Graph (DAG)

Other Components Supporting the Compiler

Compiler is a part of program development environment.

- This environment contains tools like editor, assembler, debugger, etc.

Such tools including compiler should support each other for efficient program development.

Compilers in Current Era

Overall structure of almost all the compilers is similar to the existing structure

Front end phases are typically a smaller fraction of the total time

- Today back end phases dominate all other phases.