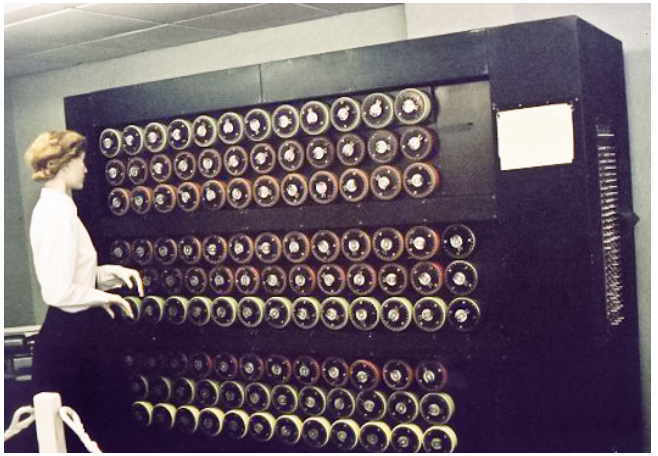
بسم الله الرحمن الرحيم

Introduction

- Course Information
- A Quick History of Compilers
- The Structure of a Compiler

What is a Compiler?

History of High-level Languages



A Short History of Compilers

- First, there was nothing.
- Then, there was machine code.
- Then, there were assembly languages.
- Programming expensive; 50% of costs for machines went into programming.

First Practical Compiler

In his PhD dissertation 1951; published in 1954), Böhm describes for the first time a translation mechanism of a programming language, written in that same language.

https://en.wikipedia.org/wiki/Corrado_B%C3%B6hm
https://en.wikipedia.org/wiki/B%C3%B6hm%27s_language

High-Level Languages

Rear Admiral **Grace Hopper**, inventor of COBOL, and the term "compiler."

Image: http://upload.wikimedia.org/wikipedia/commons/thumb/5/55/Grace_Hopper.jpg/300px-Grace_Hopper.jpg
http://www.nytimes.com/2007/03/20/business/20backus.html

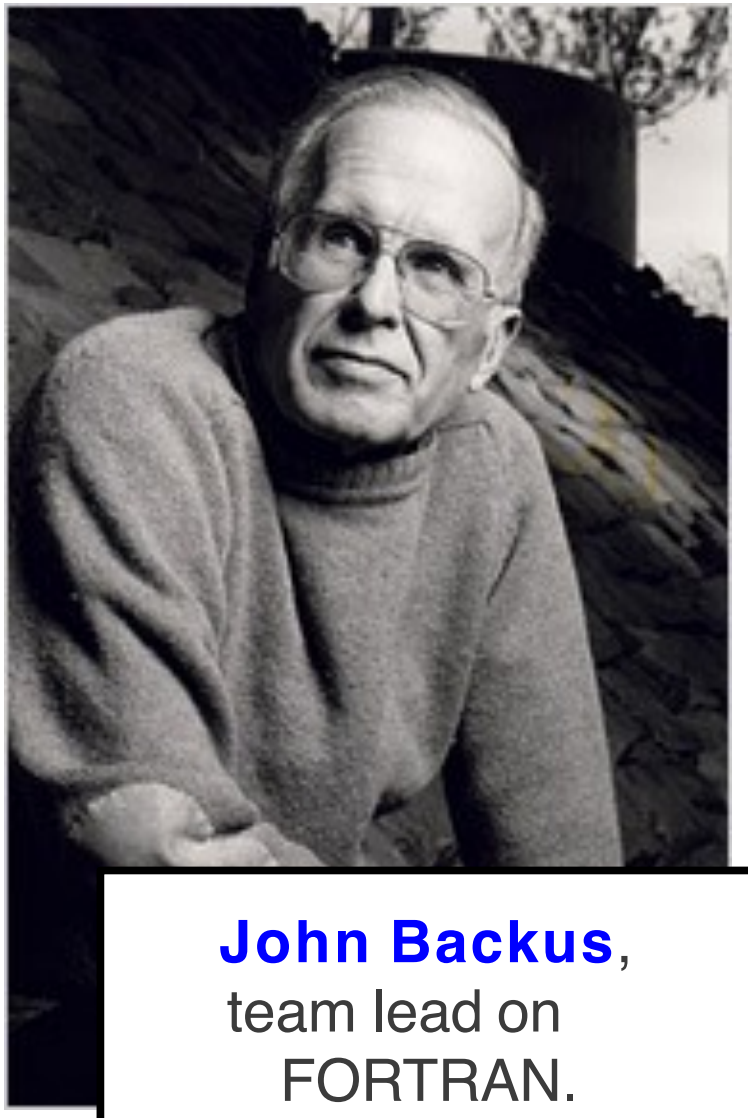
High-Level Languages

Rear Admiral **Grace Hopper**, inventor of COBOL, and the term "compiler."

A programming language based on English

Image: http://upload.wikimedia.org/wikipedia/commons/thumb/5/55/Grace_Hopper.jpg/300px-Grace_Hopper.jpg
http://www.nytimes.com/2007/03/20/business/20backus.html

High-Level Languages



John Backus, team lead on FORTRAN.

Image: http://upload.wikimedia.org/wikipedia/commons/thumb/5/55/[illegible]er.jpg
http://www.nytimes.com/2007/03/20/business/20backus.html

FORTRAN I

- Translate high-level code to assembly.
- Many thought this impossible.
- Had already failed in other projects.
- Development time halved
- Performance is close to hand-written assembly!

Effect on Computer Science

- The first compiler
- Huge impact on computer science.
- Led to an enormous body of theoretical and practical work.
- Modern compilers preserve the outlines of FORTRAN I

```
INTEGER FUNCTION FCN20(NDIMS, X, NFCNS, FUNVLS)
    INTEGER NDIMS, NFCNS
    DOUBLE PRECISION X(*), FUNVLS(*)
    DOUBLE PRECISION Z
    Z = (X(1) + X(2) + X(3)) ** 2
    IF (Z .NE. 0.0) THEN
       FUNVLS(1) = 1.0 / Z
    ELSE
       FUNVLS(1) = 0.0
    ENDIF
    FCN20 = 1
    RETURN
END
```

What is a Compiler?

- Takes as input a program written in one language and **translates** it into a **functionally equivalent** program in another language.
- Source is usually high-level (e.g. Java), target is usually low-level (e.g. Assembly).
-

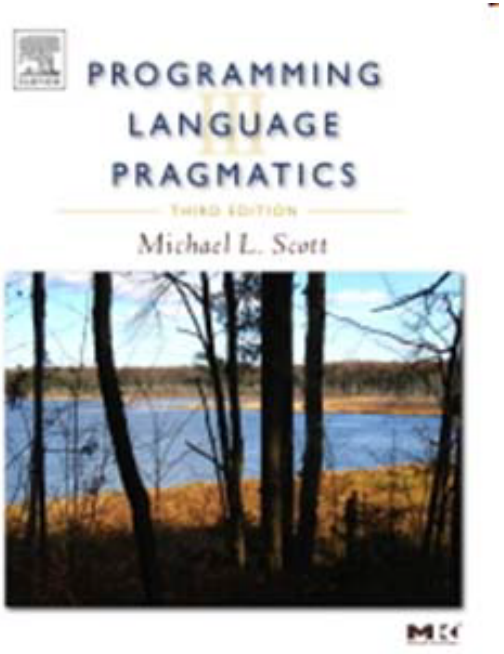
Hierarchy of dependency between concepts related to compiler

- computational model
- programming language
- compiler

Computational Thinking in Programming Language Design

Underlying every programming language is a model of computation:

- **Procedural: C, C++, C#, Java**
- **Declarative: SQL**
- **Logic: Prolog**
- **Functional: Haskell**
- **Scripting: AWK, Perl, Python, Ruby**



Evolutionary Forces on Languages and Compilers

More and different kinds of languages

Increasing diversity of applications

Stress on increasing productivity

Need to improve software reliability

Target machines more diverse

Parallel machine architectures

Massive compiler collections

1970	2010
Fortran	Java
Lisp	C
Cobol	PHP
Algol 60	C++
APL	Visual Basic
Snobol 4	C#
Simula 67	Python
Basic	Perl
PL/1	Delphi
Pascal	JavaScript

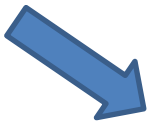
[http://www.tiobe.com]

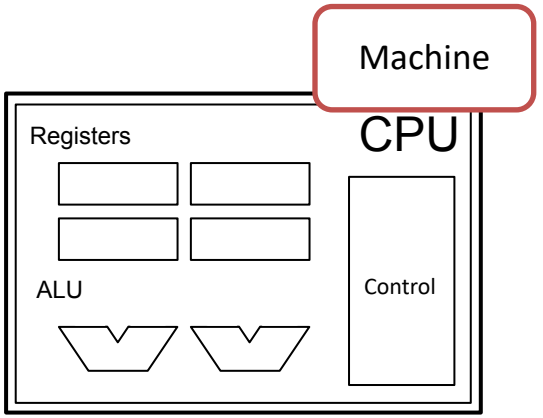
Schema

Language

```
fun(y) {
  x = y - 2;
  return x;
};
```

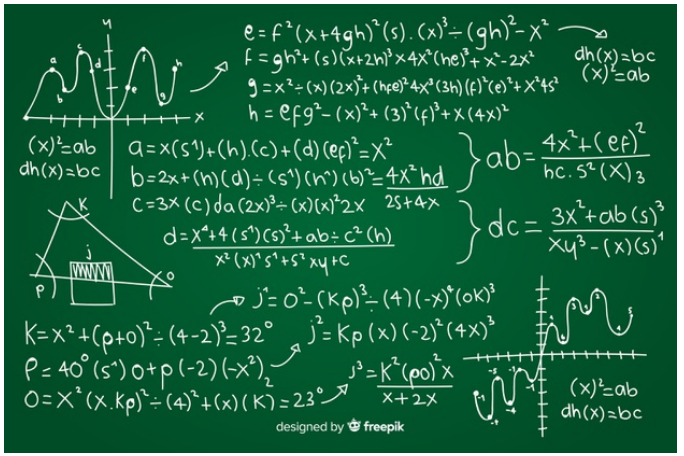
Infinite resources
No performance specification





- Finite resources
- Extremely performance sensitive

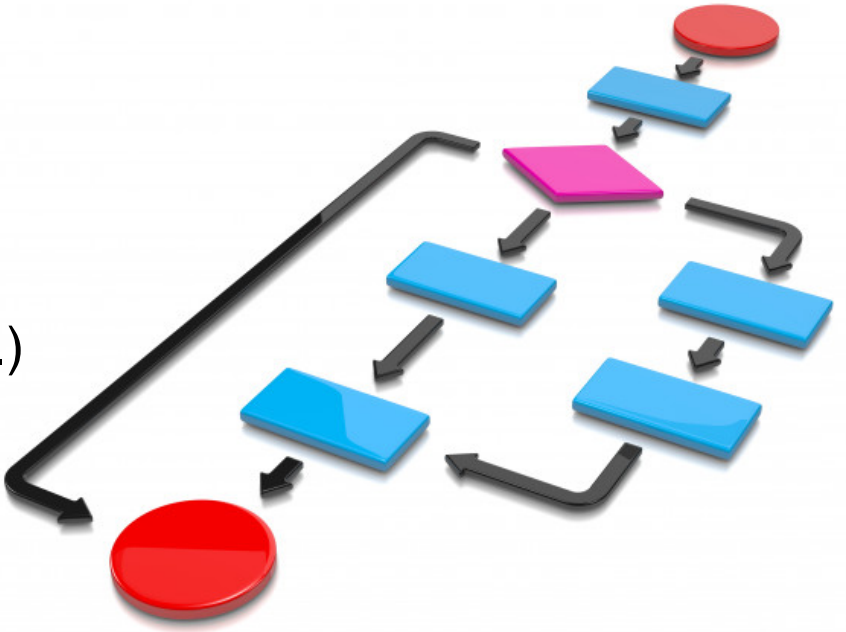
Language Environment Construction touches many topics in Computer Science



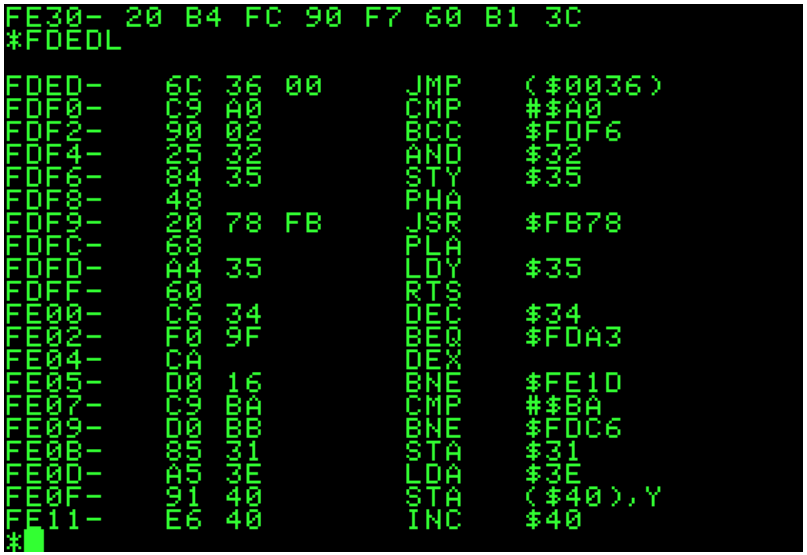
- Theory
  - Finite State Automata, Grammars and Parsing, data-flow
- Algorithms
  - Graph manipulation, dynamic programming
- Data structures
  - Symbol tables, abstract syntax trees
- Systems
  - Allocation and naming, multi-pass systems, compiler construction
- Computer Architecture
  - Memory hierarchy, instruction selection, interlocks and latencies, parallelism
- Security
  - Detection of and Protection against vulnerabilities
- Software Engineering
  - Software development environments, debugging
- Artificial Intelligence
  - Heuristic based search for best optimizations

Inputs

- Standard language
  - State
    - Variables,
    - Structures,
    - Arrays
  - Computation
    - Expressions (arithmetic, logical, etc.)
    - Assignment statements
    - Control flow (conditionals, loops)
    - Procedures



Outputs



- State
  - Registers
  - Memory with Flat Address Space
- Machine code – load/store architecture
  - Load, store instructions
  - Arithmetic, logical operations on registers
  - Branch instructions

Translation Approaches

- Compiler Approach
- Interpreter Approach
- Dynamic Approach

Compiler Approach

Compiler Approach

- The target is not necessarily a machine code.
  - e.g. Assembly language e.g. MIPS or x86
  - e.g. VHDL: the output is C.
  - It might be intermediate code e.g. JBC

Compiler Approach

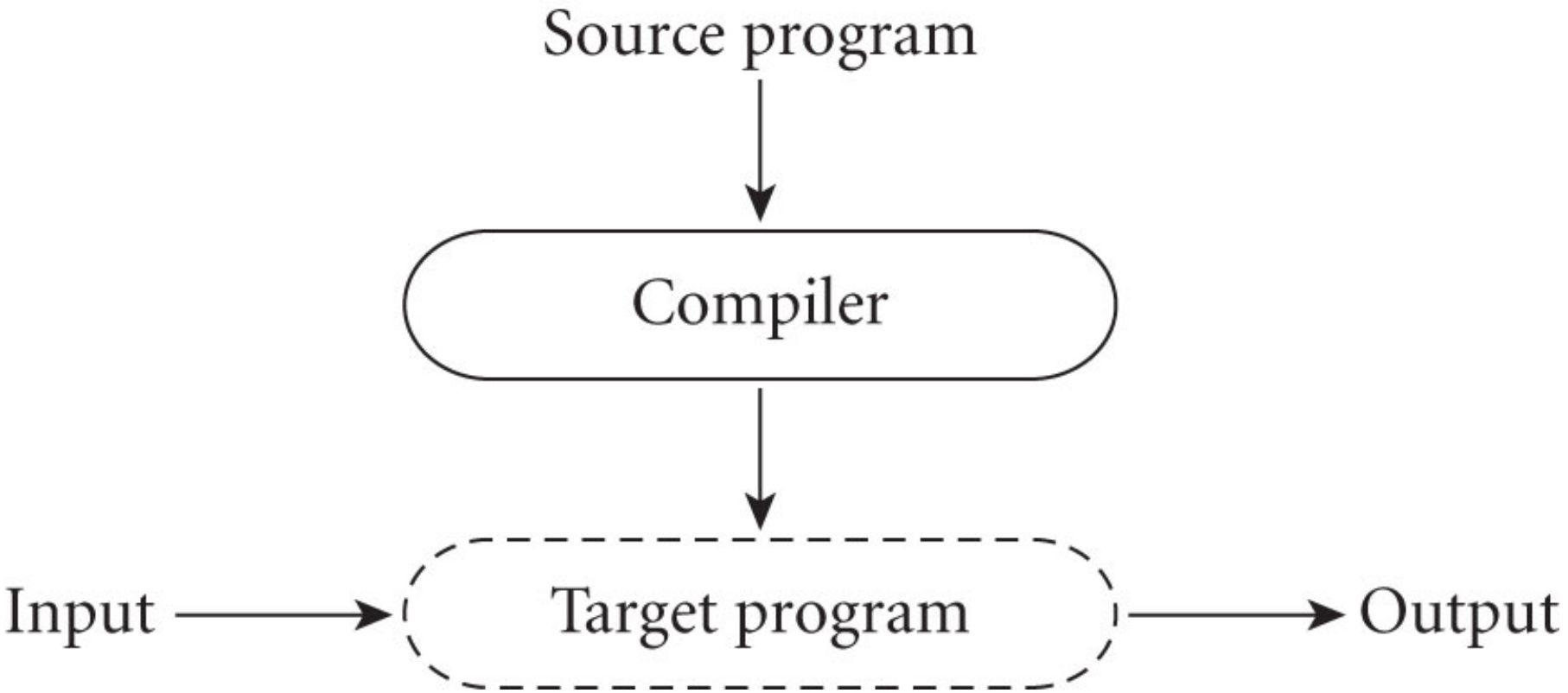
- The target is not necessarily a machine code.
  - e.g. Assembly language e.g. MIPS or x86
  - e.g. VHDL: the output is C.
  - It might be intermediate code e.g. JBC
- By following **physical structure** of program we translate it.

Compiler Approach

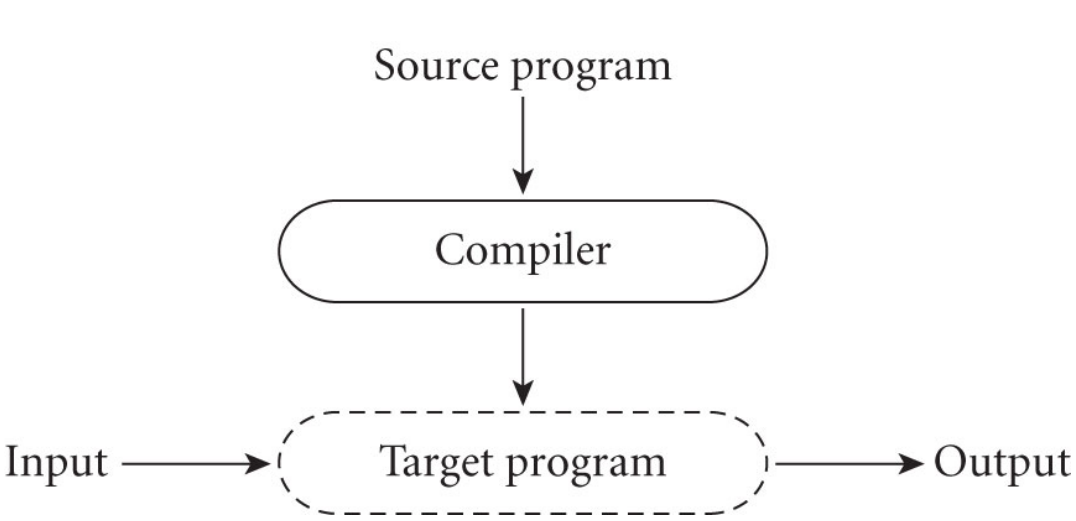
- The target is not necessarily a machine code.
  - e.g. Assembly language e.g. MIPS or x86
  - e.g. VHDL: the output is C.
  - It might be intermediate code e.g. JBC
- By following **physical structure** of program we translate it.
- The generated code is much more faster.

Compiler Approach

- The target is not necessarily a machine code.
  - e.g. Assembly language e.g. MIPS or x86
  - e.g. VHDL: the output is C.
  - It might be intermediate code e.g. JBC
- By following **physical structure** of program we translate it.
- The generated code is much more faster.
- We decide before run the code (e.g. type)


Source program
Compiler
Input
Target program
Output

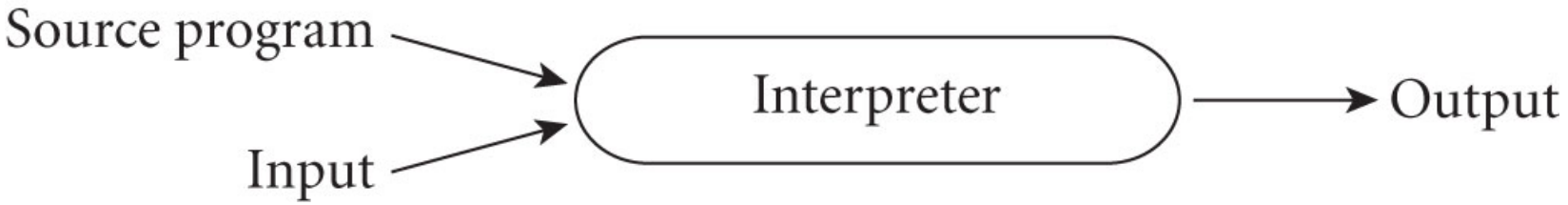
```c
#include <stdio.h>

int getint() {
    int i;
    scanf("%d", &i);
    return i;
}

int gcd() {
    int i = getint(), j = getint(
);
    while (i != j) {
        if (i > j) i = i - j;
        else j = j - i;
    }
     return I;
}
```




```
.LC0:
        .string "%d"
getint():
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        lea     rax, [rbp-4]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    __isoc99_scanf
        mov     eax, DWORD PTR [rbp-4]
        leave
        ret
.LC1:
        .string "%d\n"
putint(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC1
        mov     eax, 0
        call    printf
        leave
        ret
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        call    getint()
        mov     DWORD PTR [rbp-4], ea
x
        call    getint()
        mov     DWORD PTR [rbp-8], ea
x
        jmp     .L5
.L7:
        mov     eax, DWORD PTR [rbp-4
]
        cmp     eax, DWORD PTR [rbp-8
]
        jle     .L6
        mov     eax, DWORD PTR [rbp-8
]
        sub     DWORD PTR [rbp-4], ea
x
        jmp     .L5
.L6:
        mov     eax, DWORD PTR [rbp-4
]
        sub     DWORD PTR [rbp-8], ea
x
.L5:
        mov     eax, DWORD PTR [rbp-4
]
```

Interpreter Approach

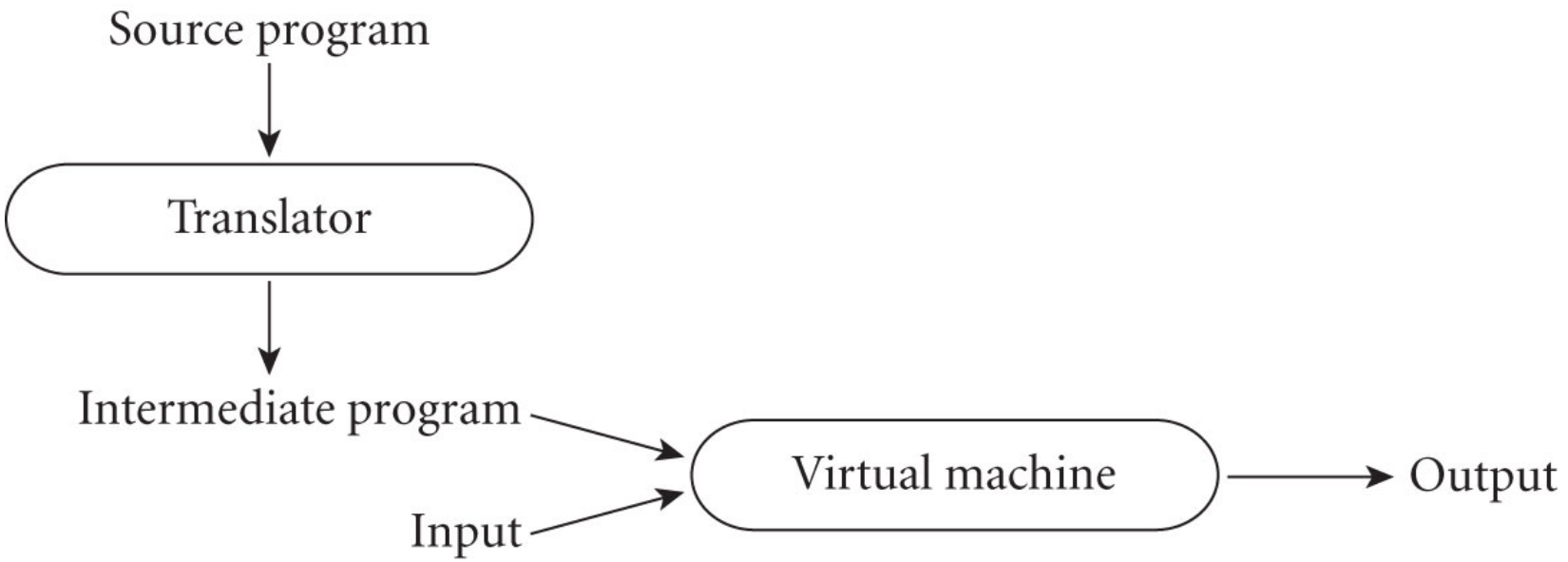
- Interpreter **translate** the source code to machine code **online**.
- The code is **translated** while it is **running.**



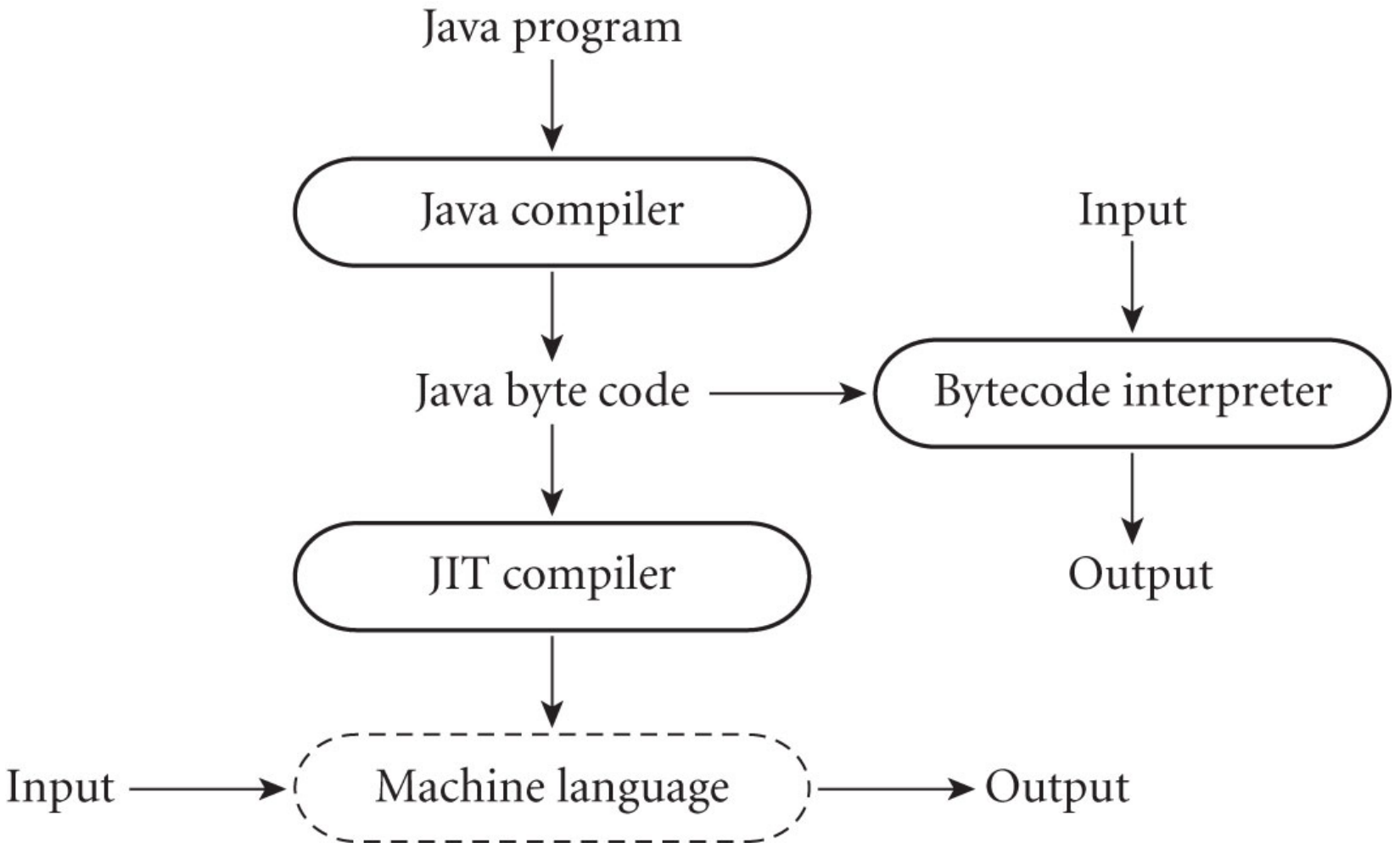
Interpreter Approach

- We follow the **execution path** to translate the code.
- leads to greater **flexibility** and better **diagnostics** (error messages) than does compilation.
- It can also cope with languages in which fundamental characteristics of the program, such as the sizes and types of variables, or even which names refer to which variables, can depend on the input data.

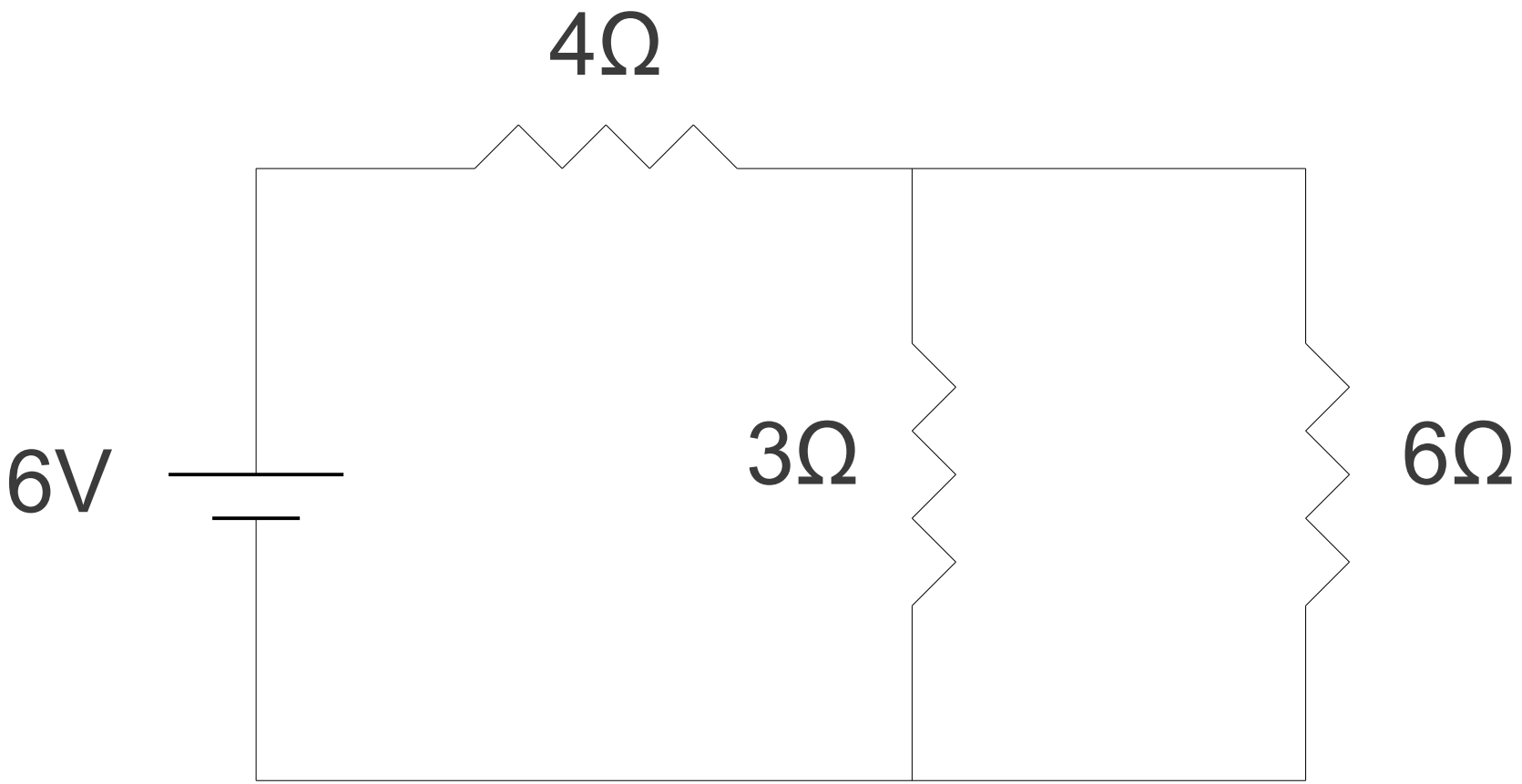
A Hybrid Approach

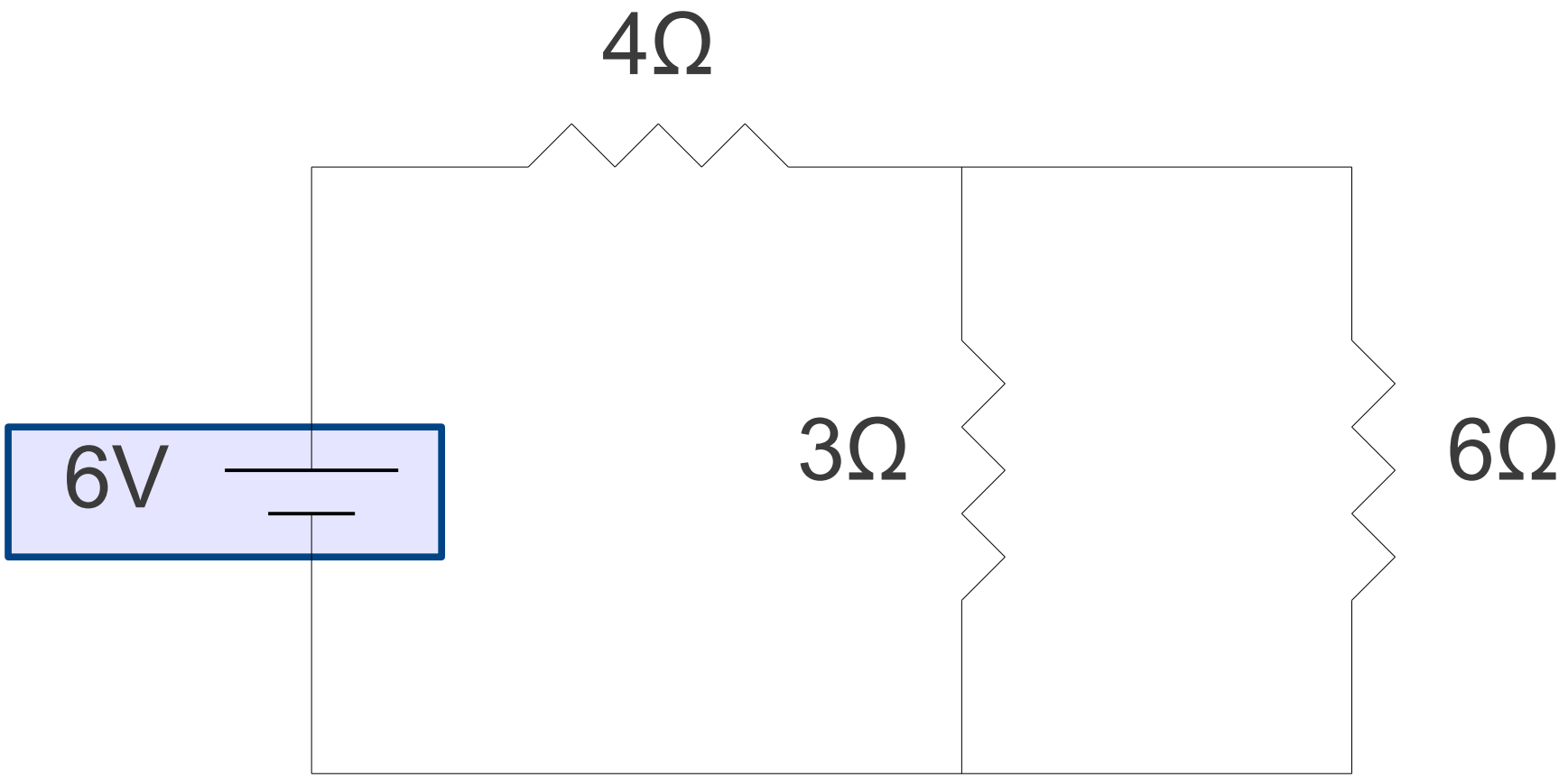


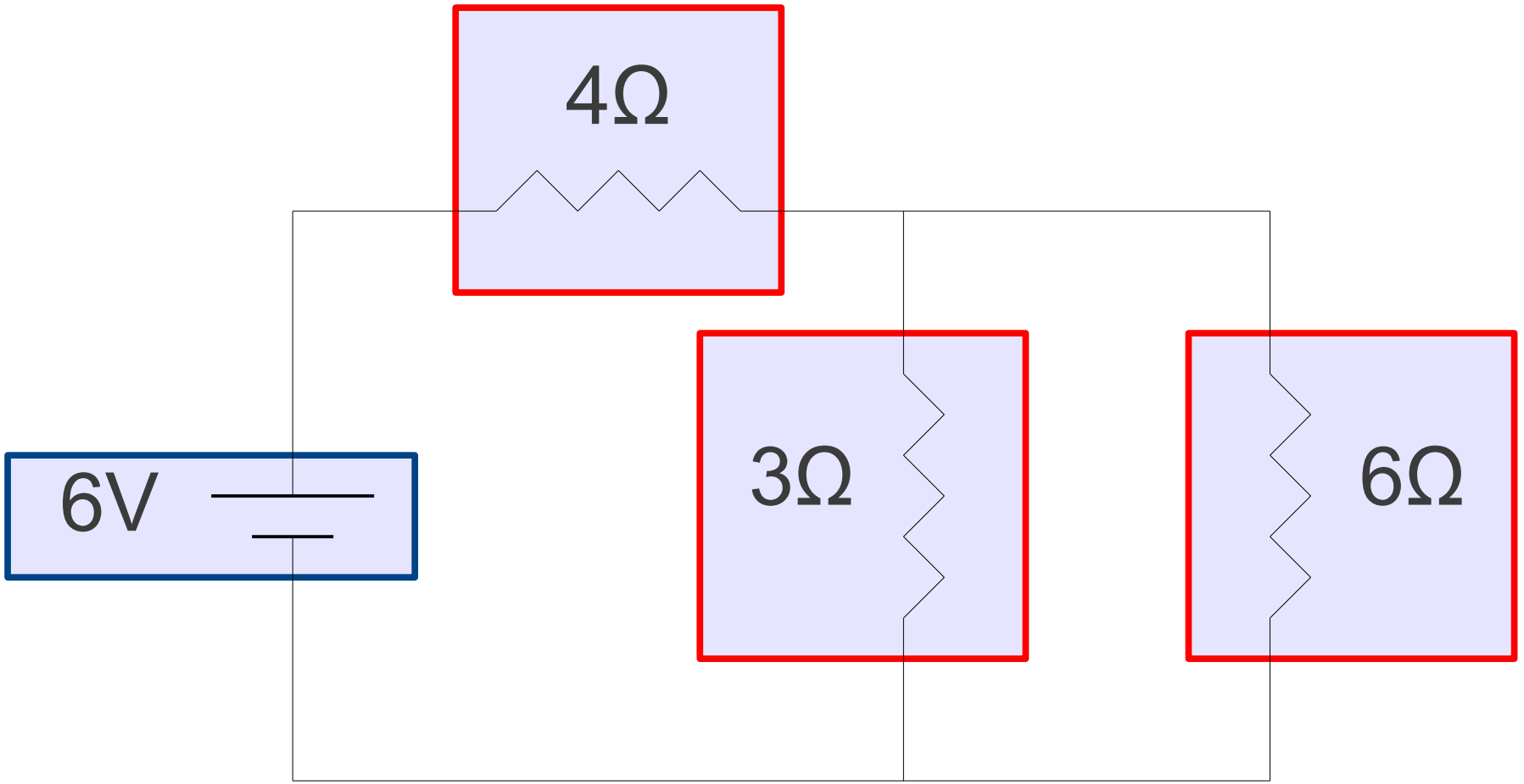
Hybrid Approach

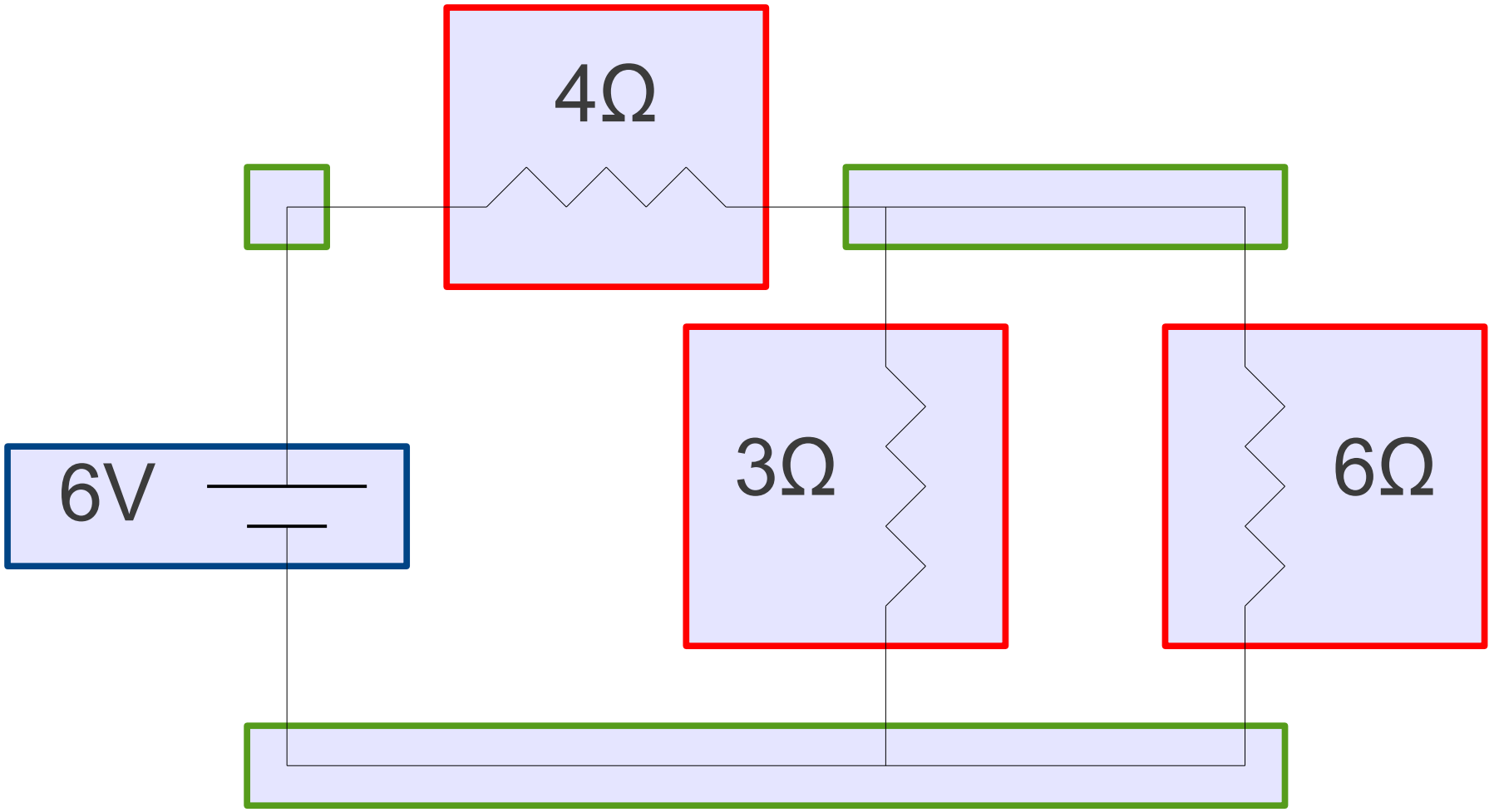


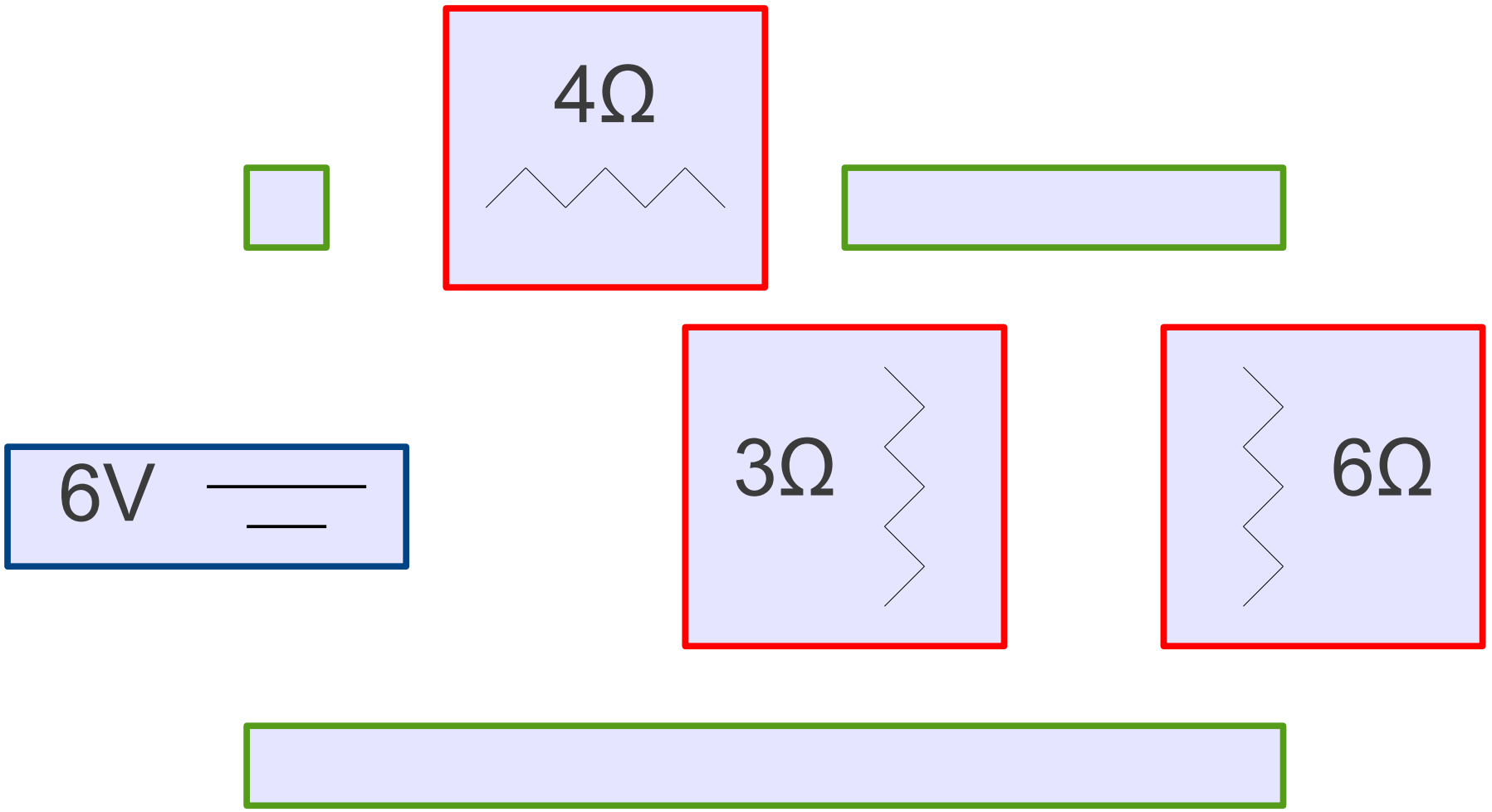
How does a compiler work?

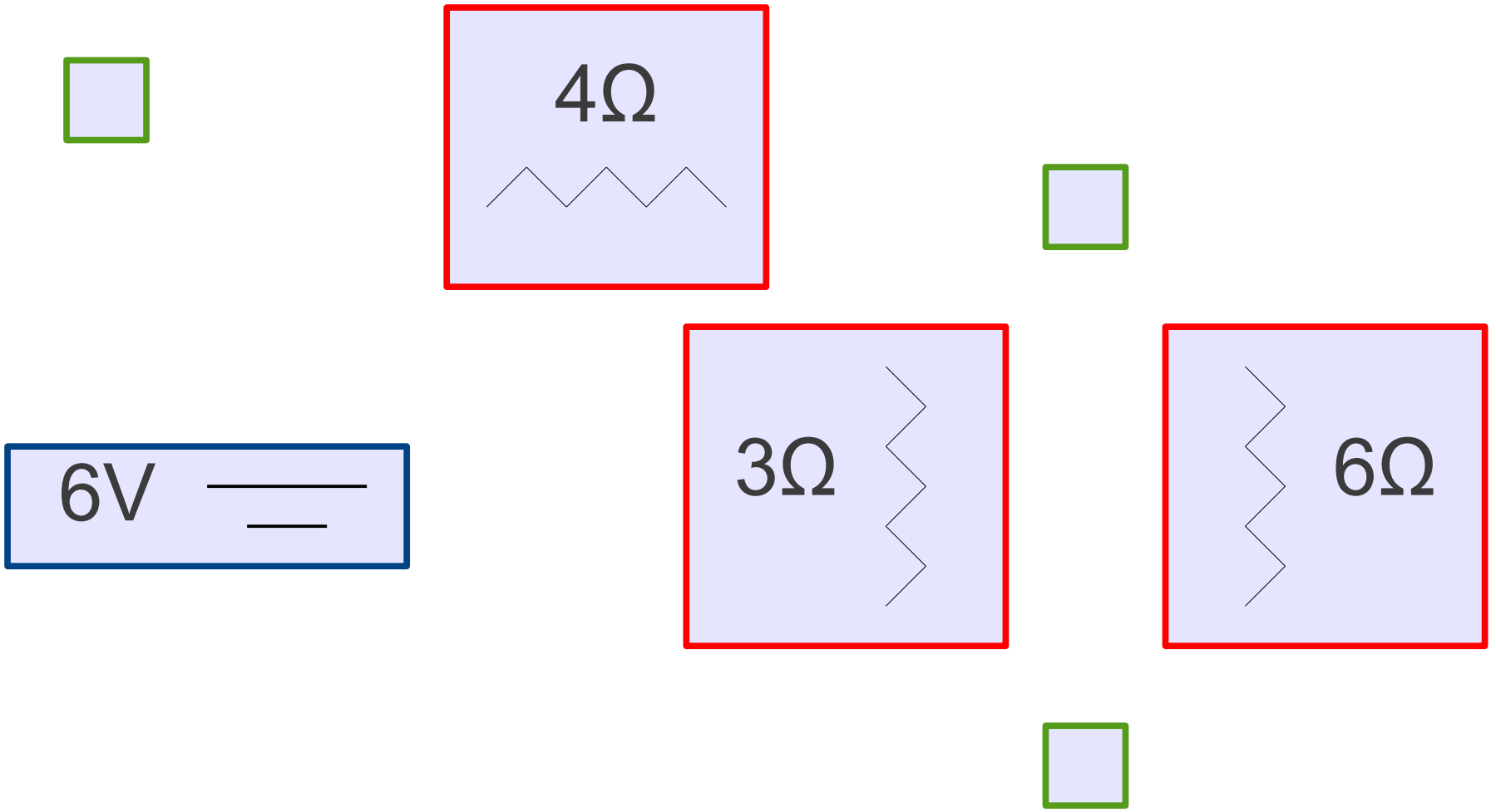

4Ω
6V
3Ω
6Ω

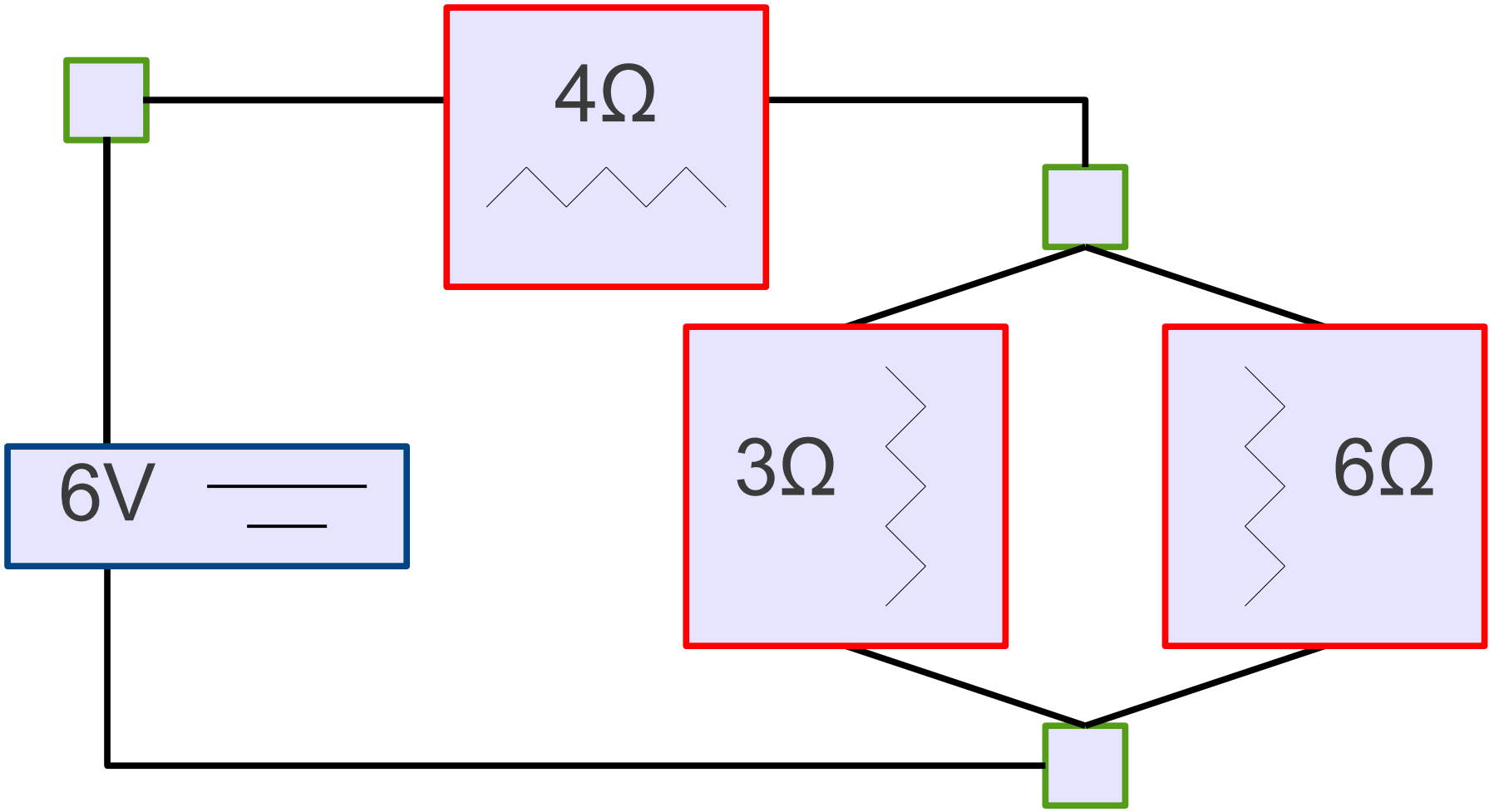

4Ω
6V
3Ω
6Ω

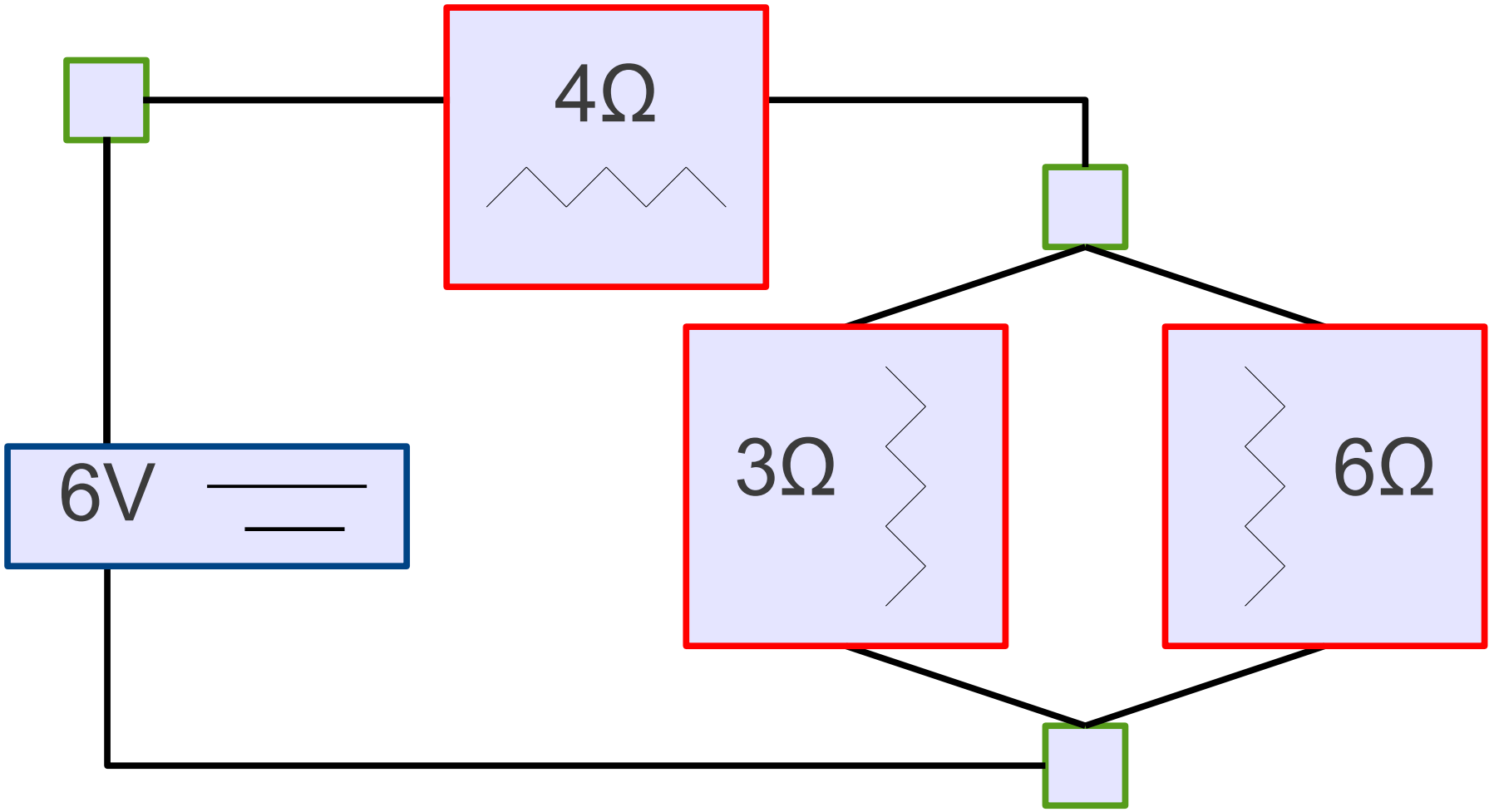

4Ω
3Ω
6Ω
6V


4Ω
3Ω
6Ω
6V

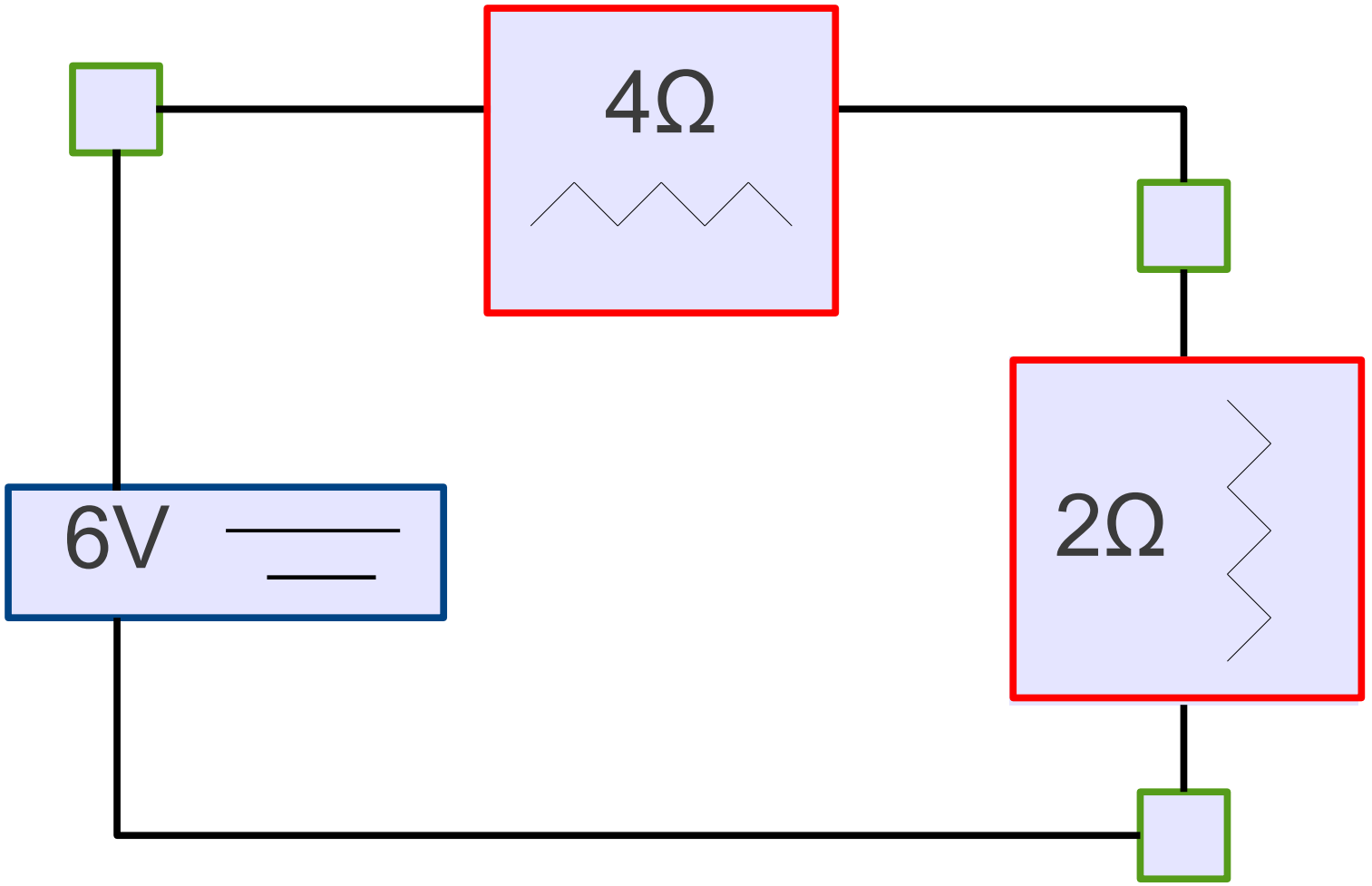

4Ω
6V
3Ω
6Ω


4Ω
6V
3Ω
6Ω

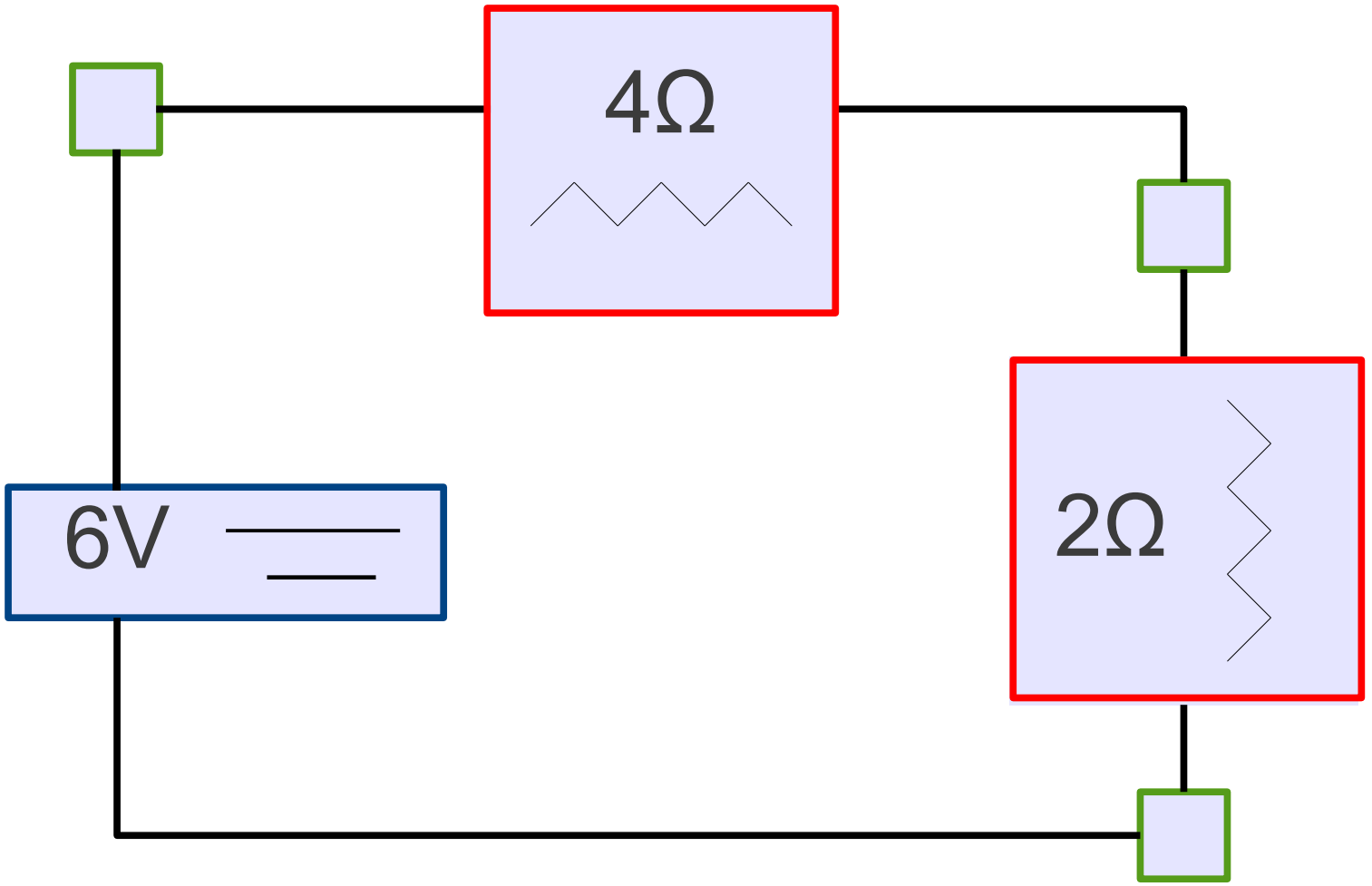

4Ω
3Ω
6Ω
6V

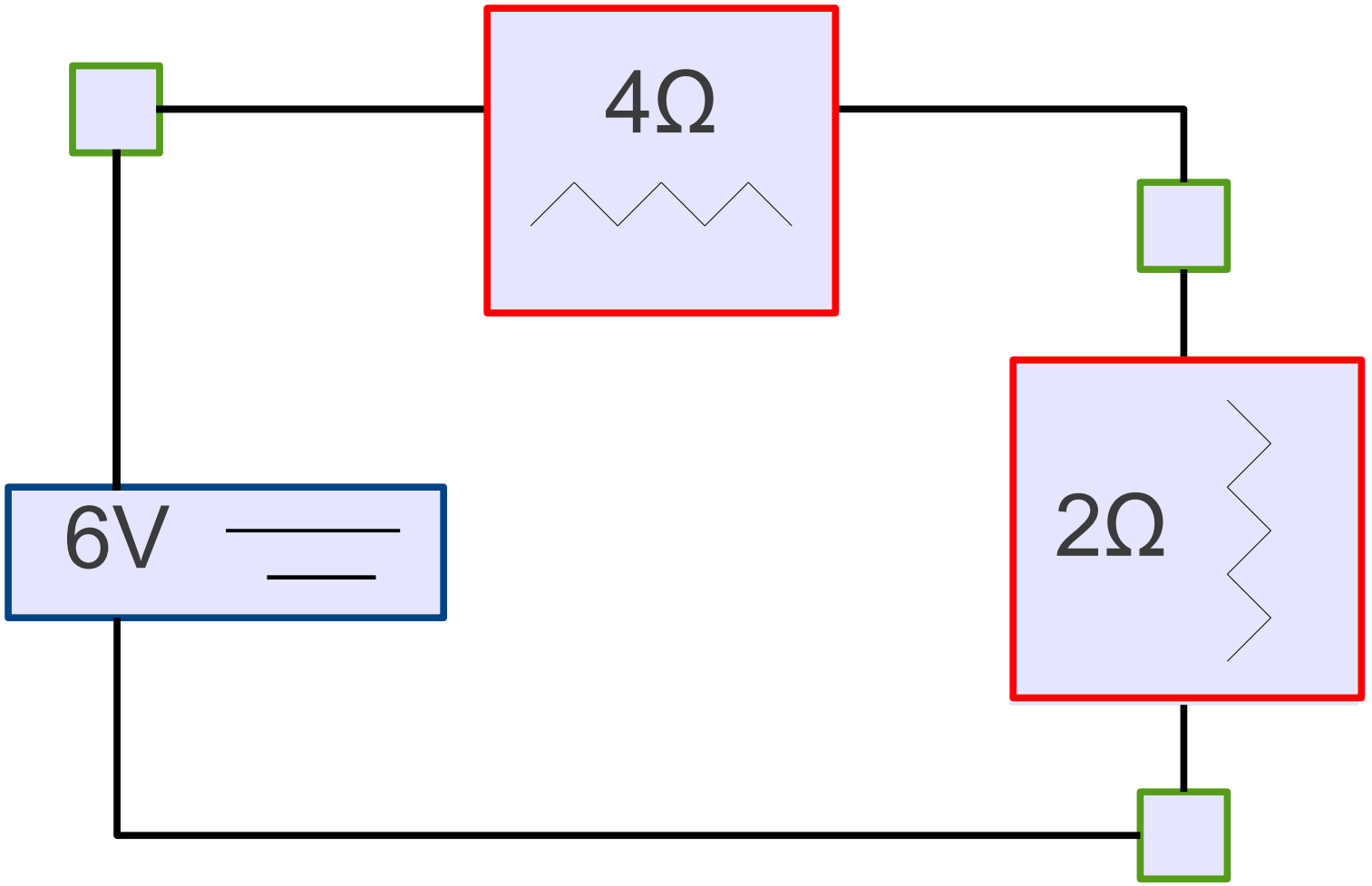


$$\frac{1}{\frac{1}{3\,\Omega}+\frac{1}{6\,\Omega}}=2\ \Omega$$

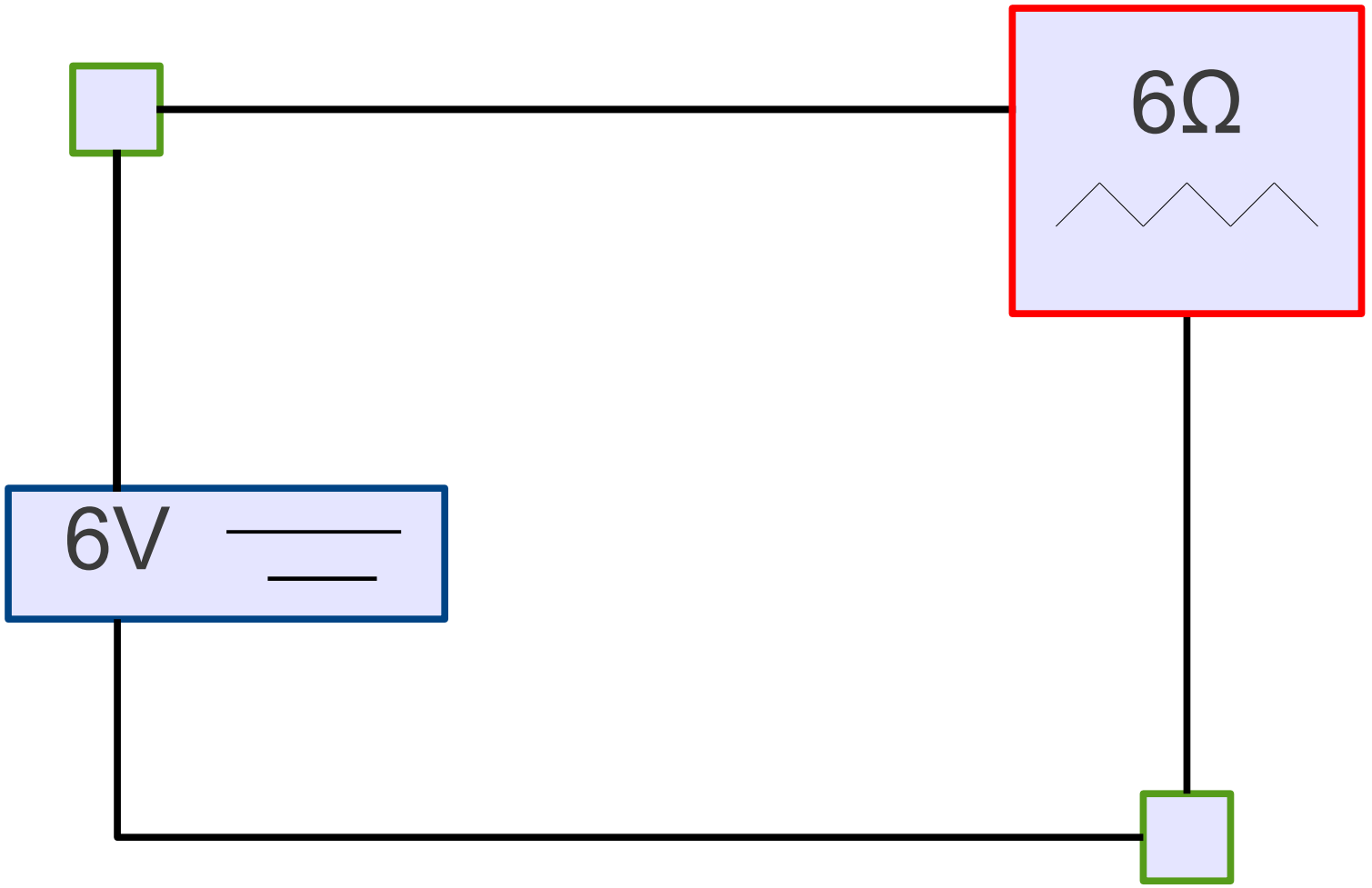


$$\frac{1}{\frac{1}{3\,\Omega}+\frac{1}{6\,\Omega}}=2\ \Omega$$

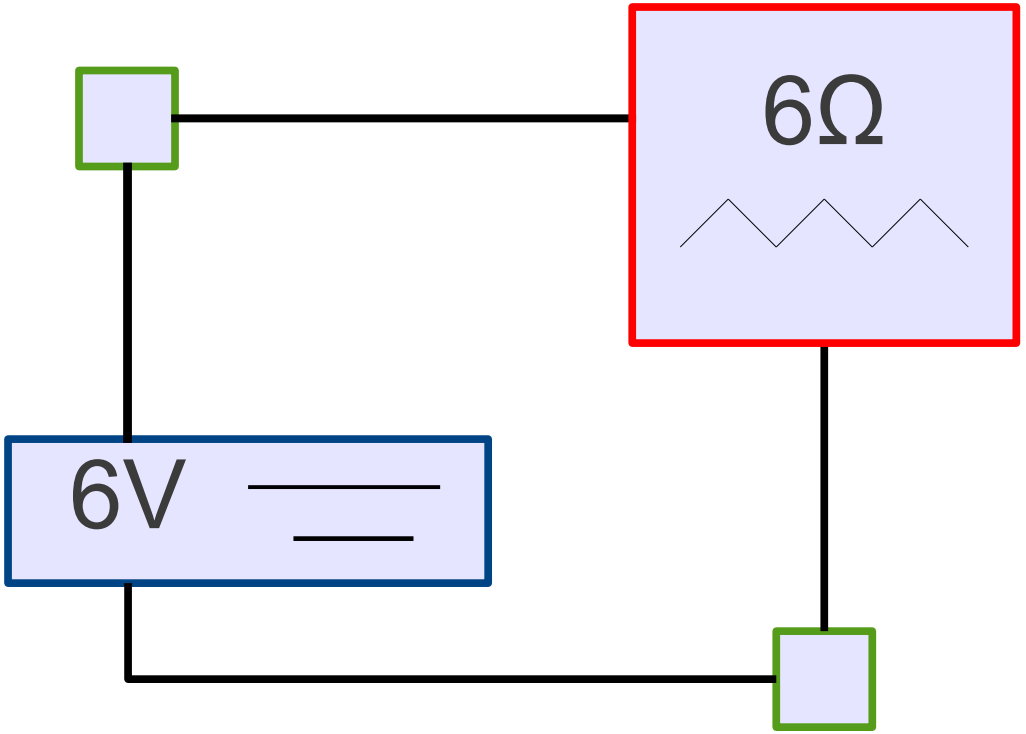

4Ω
6V
2Ω

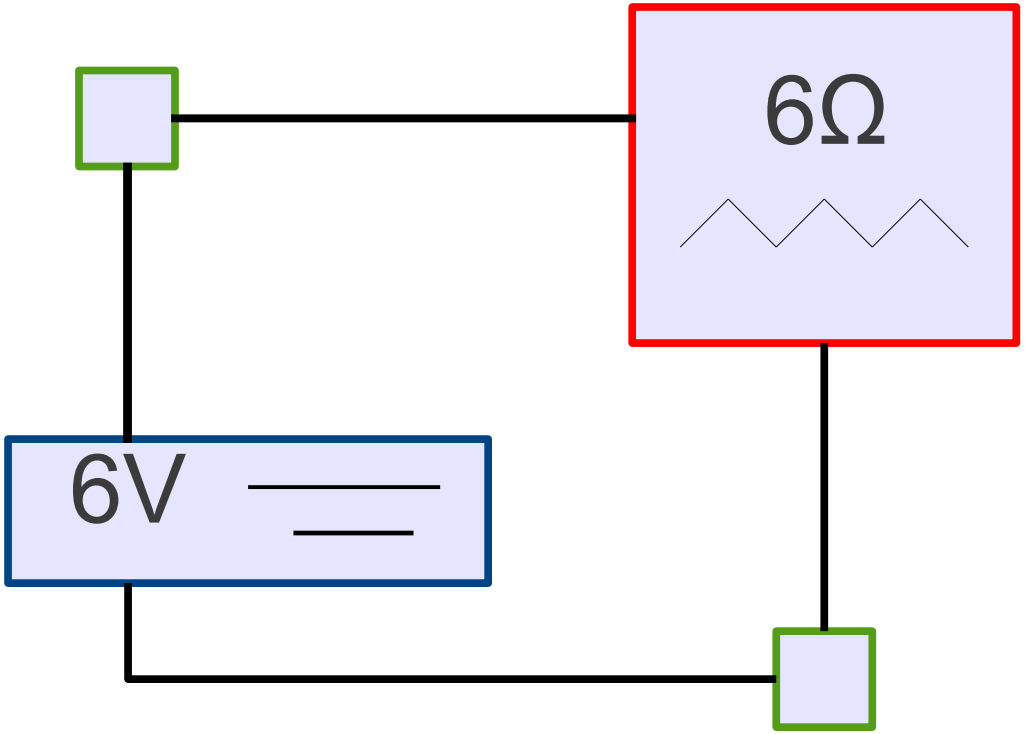


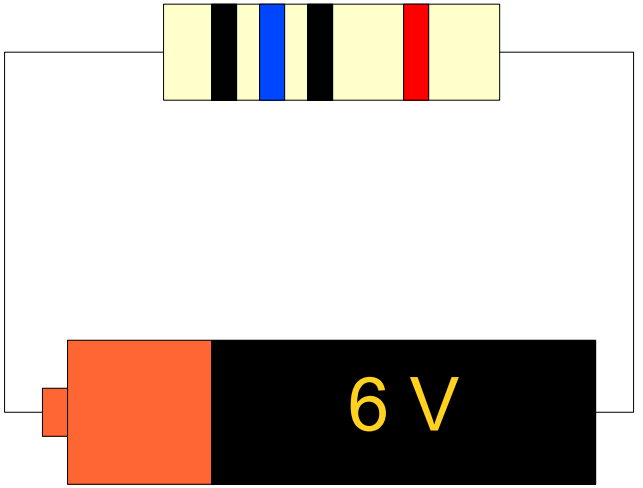
$$4\,\Omega + 2\,\Omega = 6\,\Omega$$

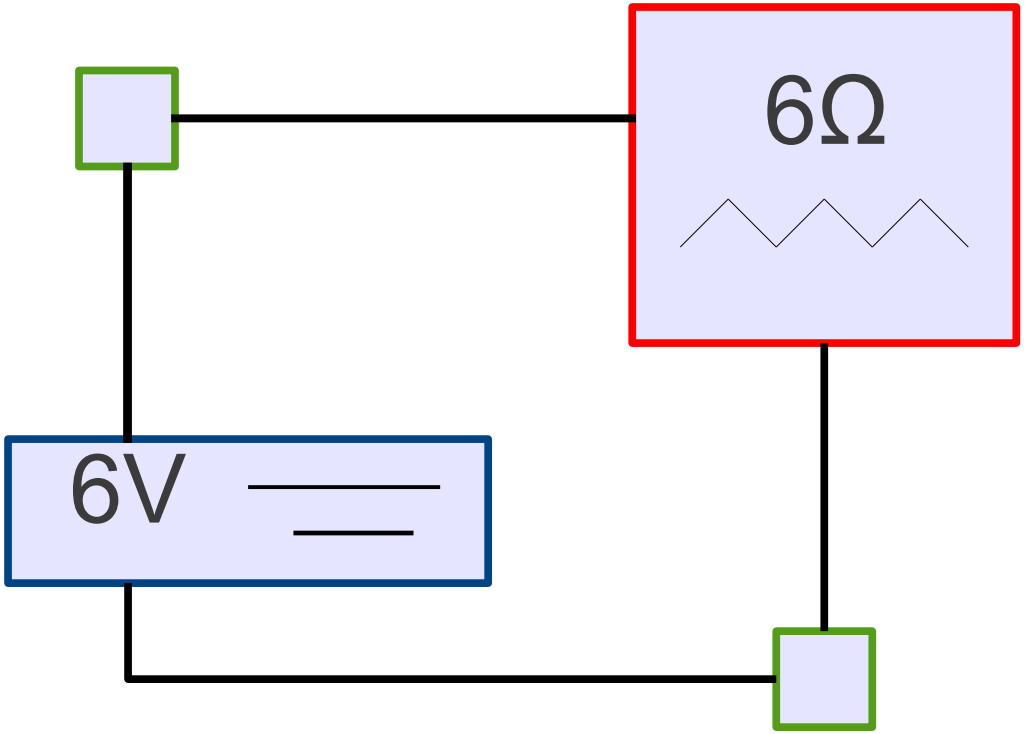


$4\,\Omega + 2\,\Omega = 6\,\Omega$

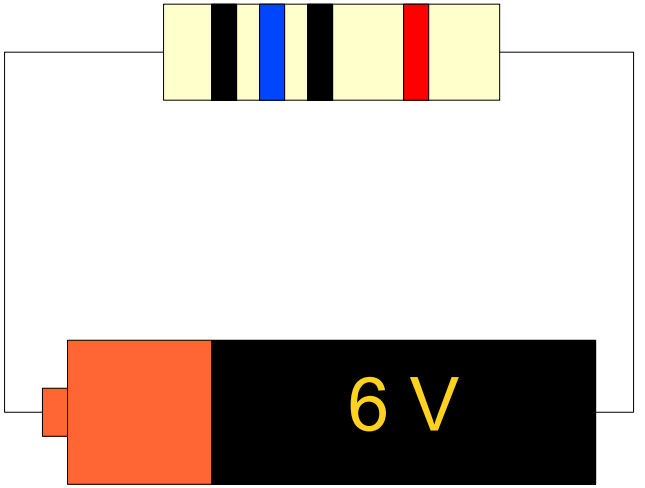

6Ω
6V

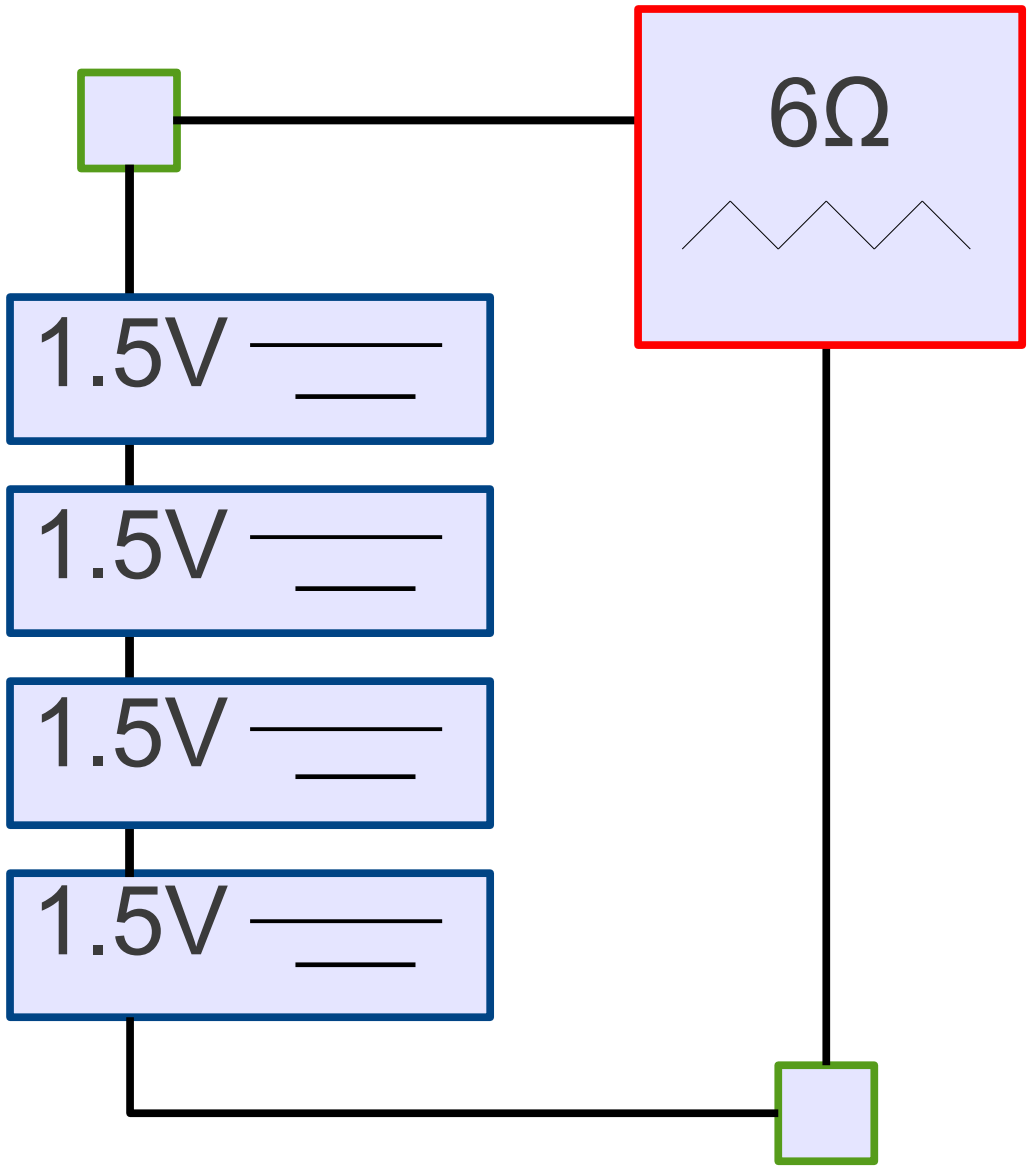

6Ω
6V

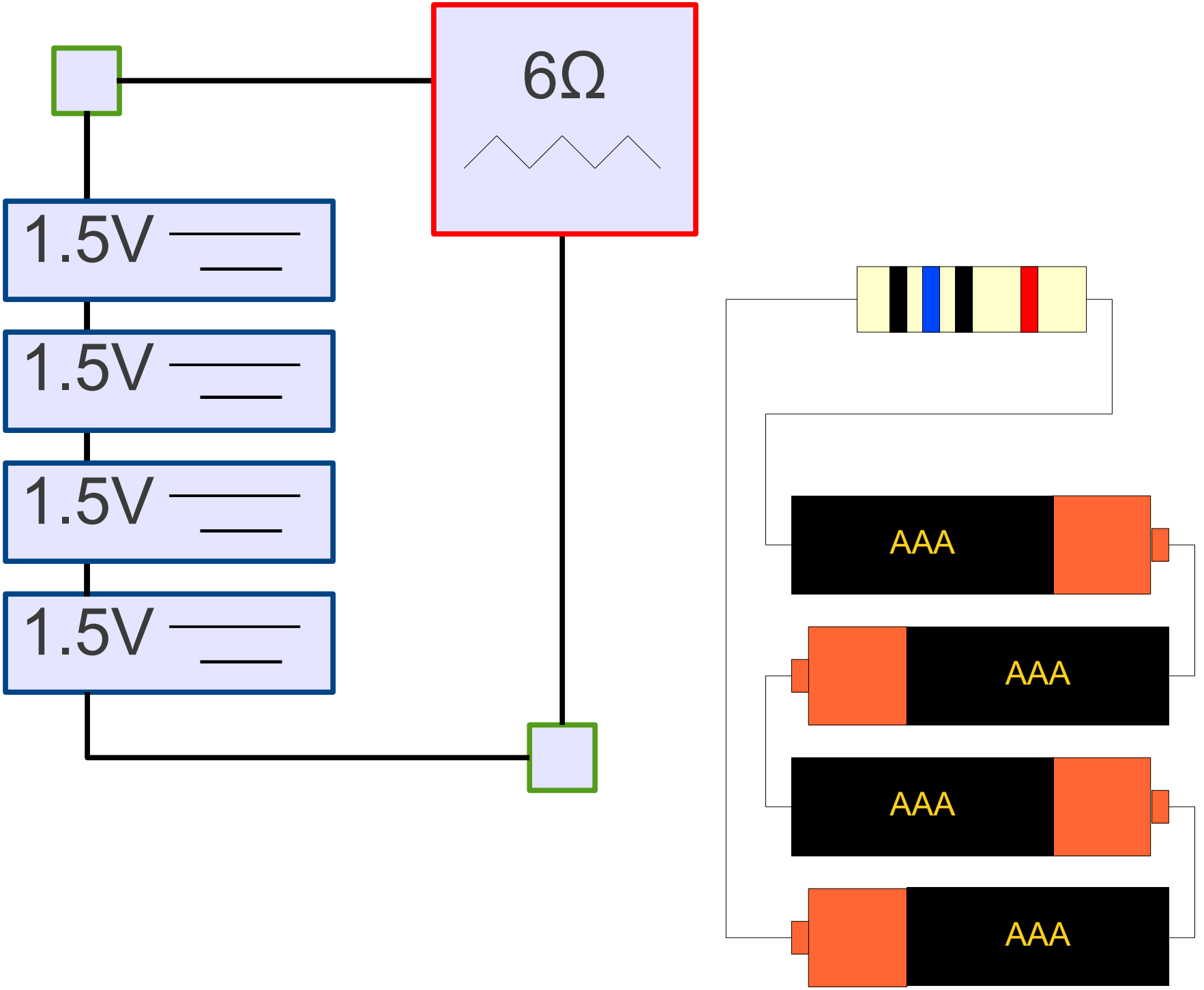

6 V

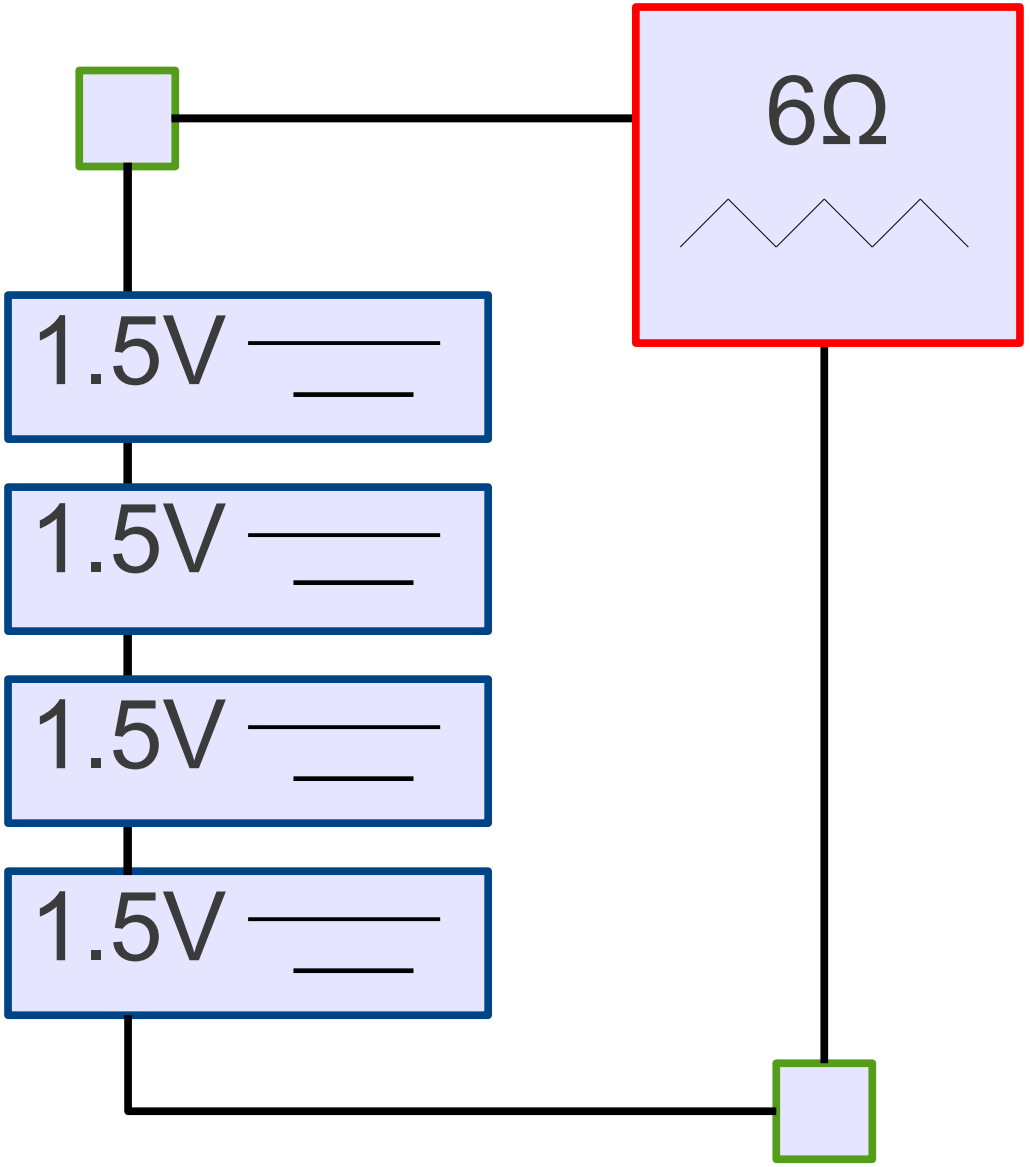


Total Cost: $4.75

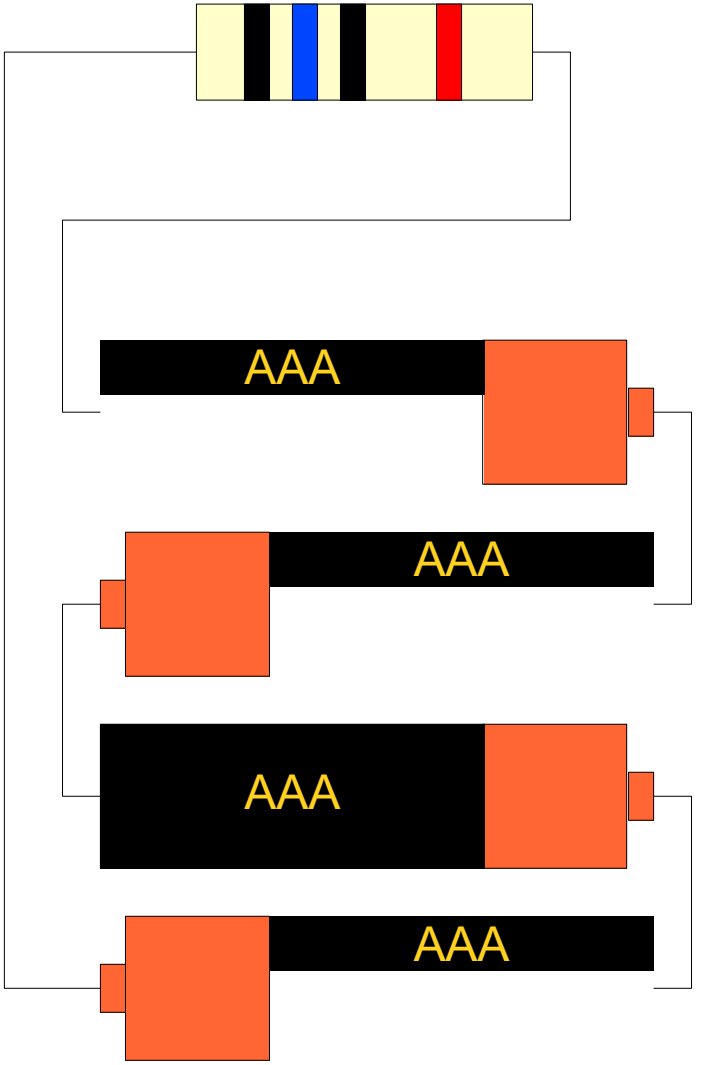



6Ω
1.5V
1.5V
1.5V
1.5V


6Ω
1.5V
1.5V
1.5V
1.5V
AAA
AAA
AAA
AAA



Total Cost: $1.00



From Description to Implementation

From Description to Implementation

- **Lexical analysis (Scanning):** Identify logical pieces of the description.

From Description to Implementation

- **Lexical analysis (Scanning):** Identify logical pieces of the description.
- **Syntax analysis (Parsing):** Identify how those pieces relate to each other.

From Description to Implementation

- **Lexical analysis (Scanning):** Identify logical pieces of the description.
- **Syntax analysis (Parsing):** Identify how those pieces relate to each other.
- **Semantic analysis:** Identify the meaning of the overall structure.

From Description to Implementation

- **Lexical analysis (Scanning):** Identify logical pieces of the description.
- **Syntax analysis (Parsing):** Identify how those pieces relate to each other.
- **Semantic analysis:** Identify the meaning of the overall structure.
- **IR Generation:** Design one possible structure.

From Description to Implementation

- **Lexical analysis (Scanning):** Identify logical pieces of the description.
- **Syntax analysis (Parsing):** Identify how those pieces relate to each other.
- **Semantic analysis:** Identify the meaning of the overall structure.
- **IR Generation:** Design one possible structure.
- **IR Optimization:** Simplify the intended structure.

From Description to Implementation

- **Lexical analysis (Scanning):** Identify logical pieces of the description.
- **Syntax analysis (Parsing):** Identify how those pieces relate to each other.
- **Semantic analysis:** Identify the meaning of the overall structure.
- **IR Generation:** Design one possible structure.
- **IR Optimization:** Simplify the intended structure.
- **Generation:** Fabricate the structure.

From Description to Implementation

- **Lexical analysis (Scanning):** Identify logical pieces of the description.
- **Syntax analysis (Parsing):** Identify how those pieces relate to each other.
- **Semantic analysis:** Identify the meaning of the overall structure.
- **IR Generation:** Design one possible structure.
- **IR Optimization:** Simplify the intended structure.
- **Generation:** Fabricate the structure.
- **Optimization:** Improve the resulting structure.

Lexical Analysis

- First step: recognize words.
  - Smallest unit above letters

This is a sentence.

- Lexical analysis is not trivial. Consider:

ist his ase nte nce

Real world Example: Watch it On YouTube!

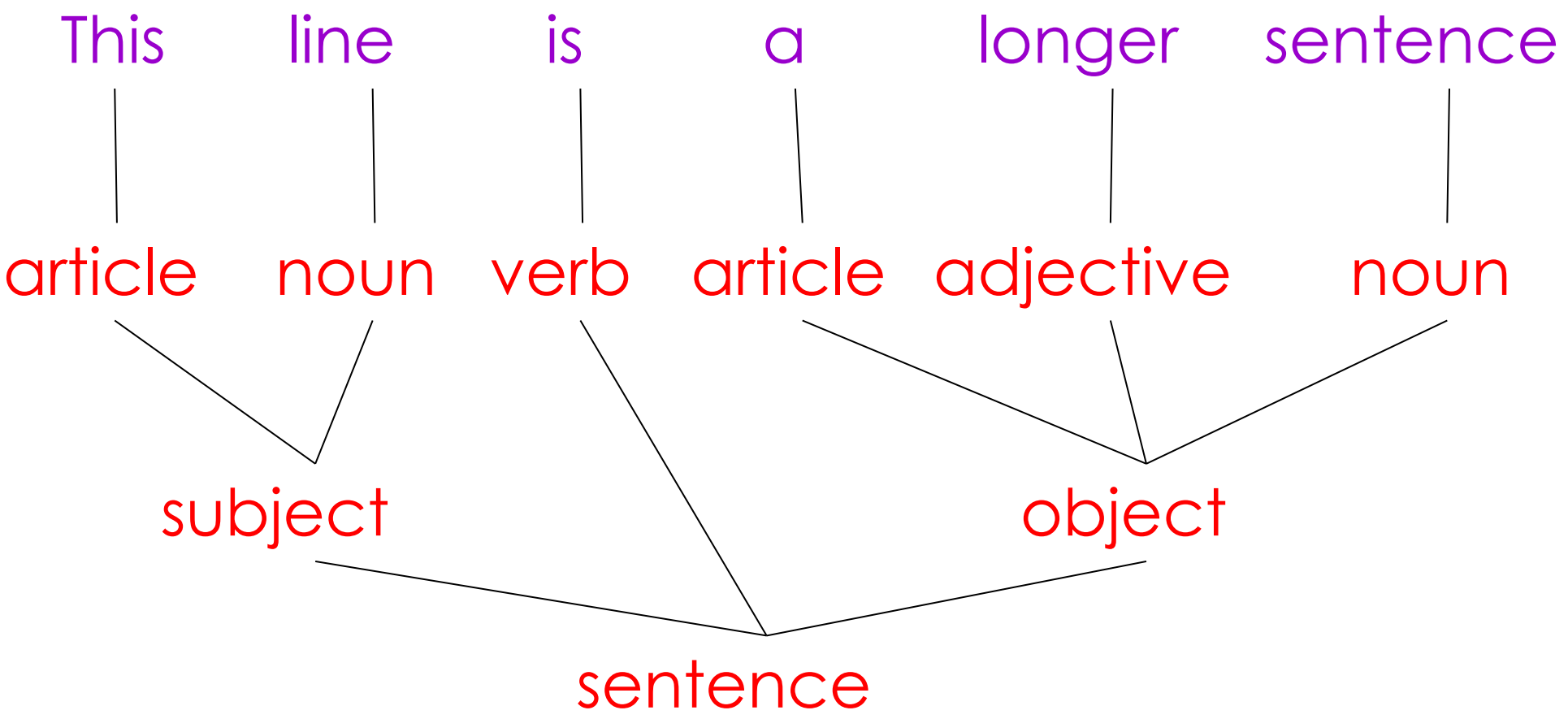
- Lexical analyzer divides program text into "words" or "tokens"

  If x == y then z = 1; else z = 2;

- Units:

- Once words are understood, the next step is to understand sentence structure

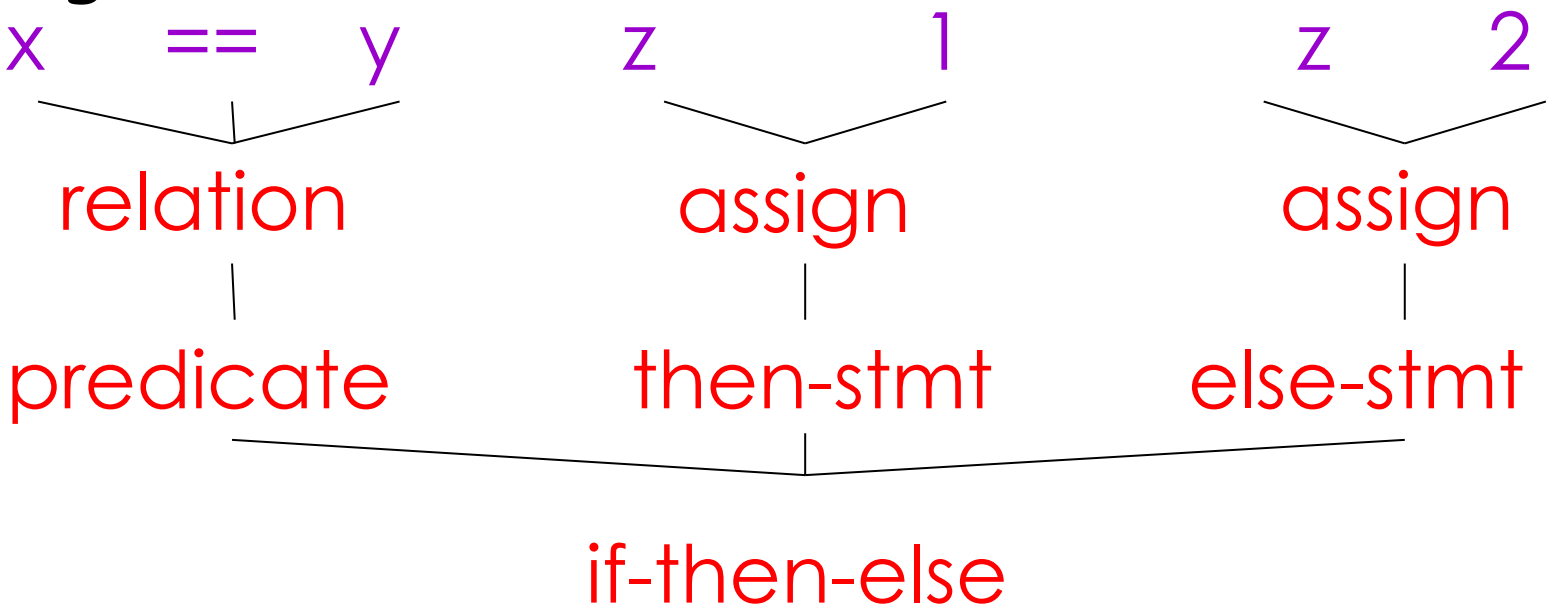
- Parsing = Diagramming Sentences
  - The diagram is a tree


This
line
is
a
longer
sentence
article
noun
verb
article
adjective
noun
subject
object
sentence

- Parsing program expressions is the same
- Consider:

If x == y then z = 1; else z = 2;

- Diagrammed:



- Once sentence structure is understood, we can try to understand “meaning”
  - But meaning is too hard for compilers
- Compilers perform limited semantic analysis to catch inconsistencies

Semantic Analysis in English

- Example:

  Jack said Jerry left his assignment at home.

  What does "his" refer to? Jack or Jerry?

- Even worse:

  Jack said Jack left his assignment at home?

  How many Jacks are there?

  Which one left the assignment?

Semantic Analysis in Programming

- Programming languages define strict rules to avoid such ambiguities
- This C++ code prints "4"; the inner definition is used

```
{
    int Jack = 3;
    {
        int Jack = 4;
        cout << Jack;
    }
}
```

More Semantic Analysis

- Compilers perform many semantic checks besides variable bindings

- Example:

  Jack left her homework at home.

- Possible type mismatch between her and Jack
  - If Jack is male

Optimization

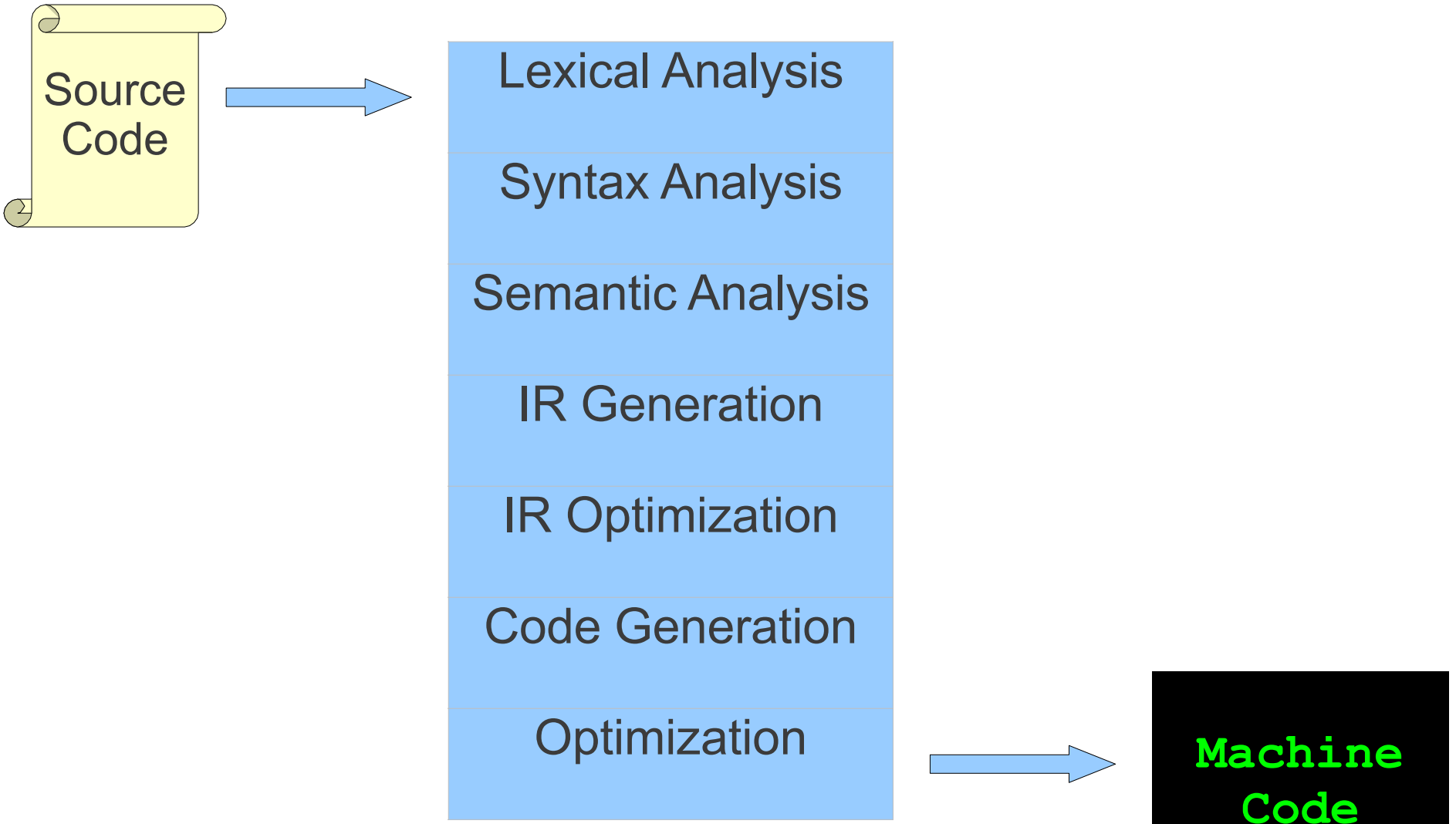
- No strong counterpart in English, but akin to editing

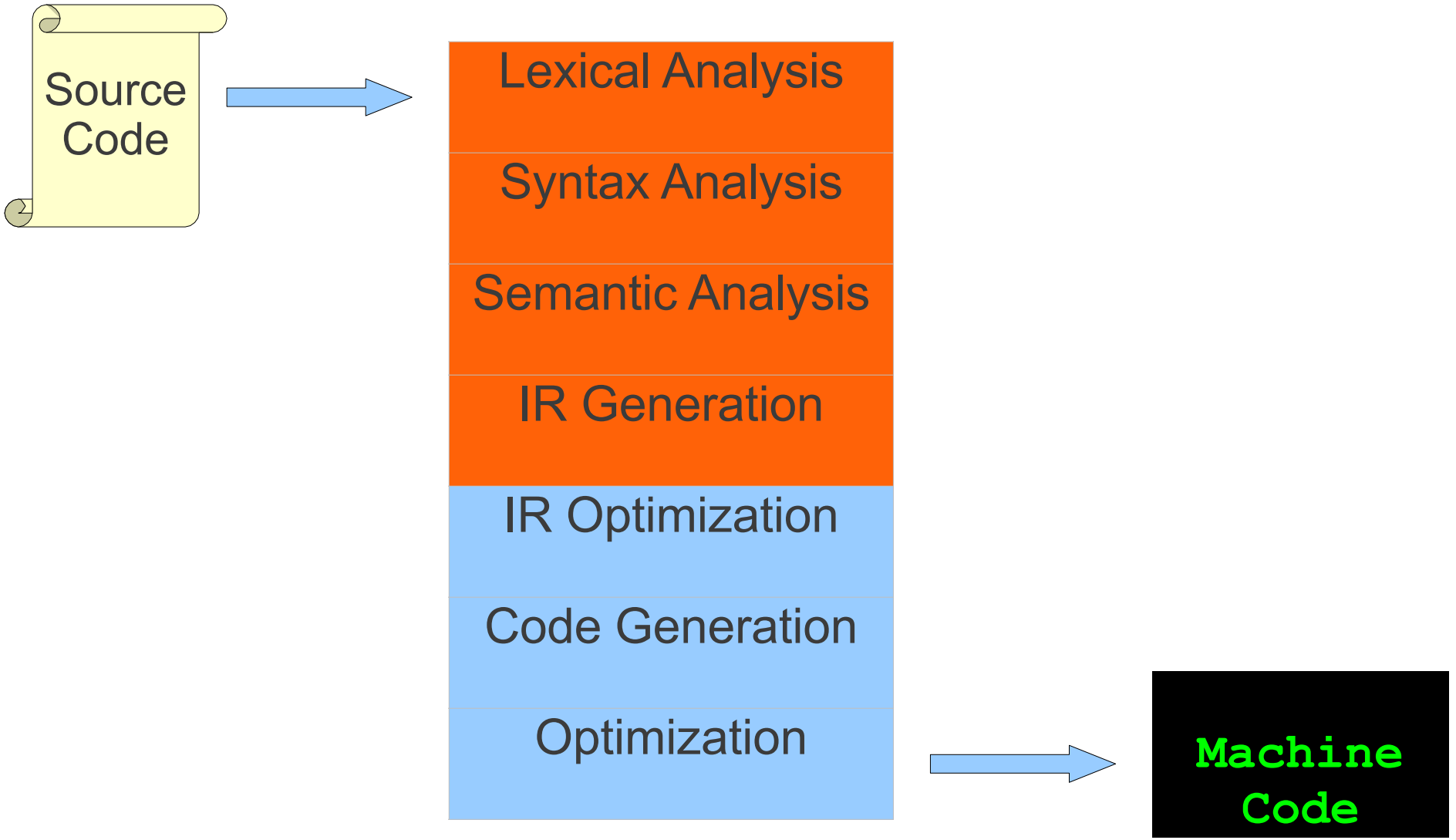
- Automatically modify programs so that they
  - Run faster
  - Use less memory
  - In general, to use or conserve some resource

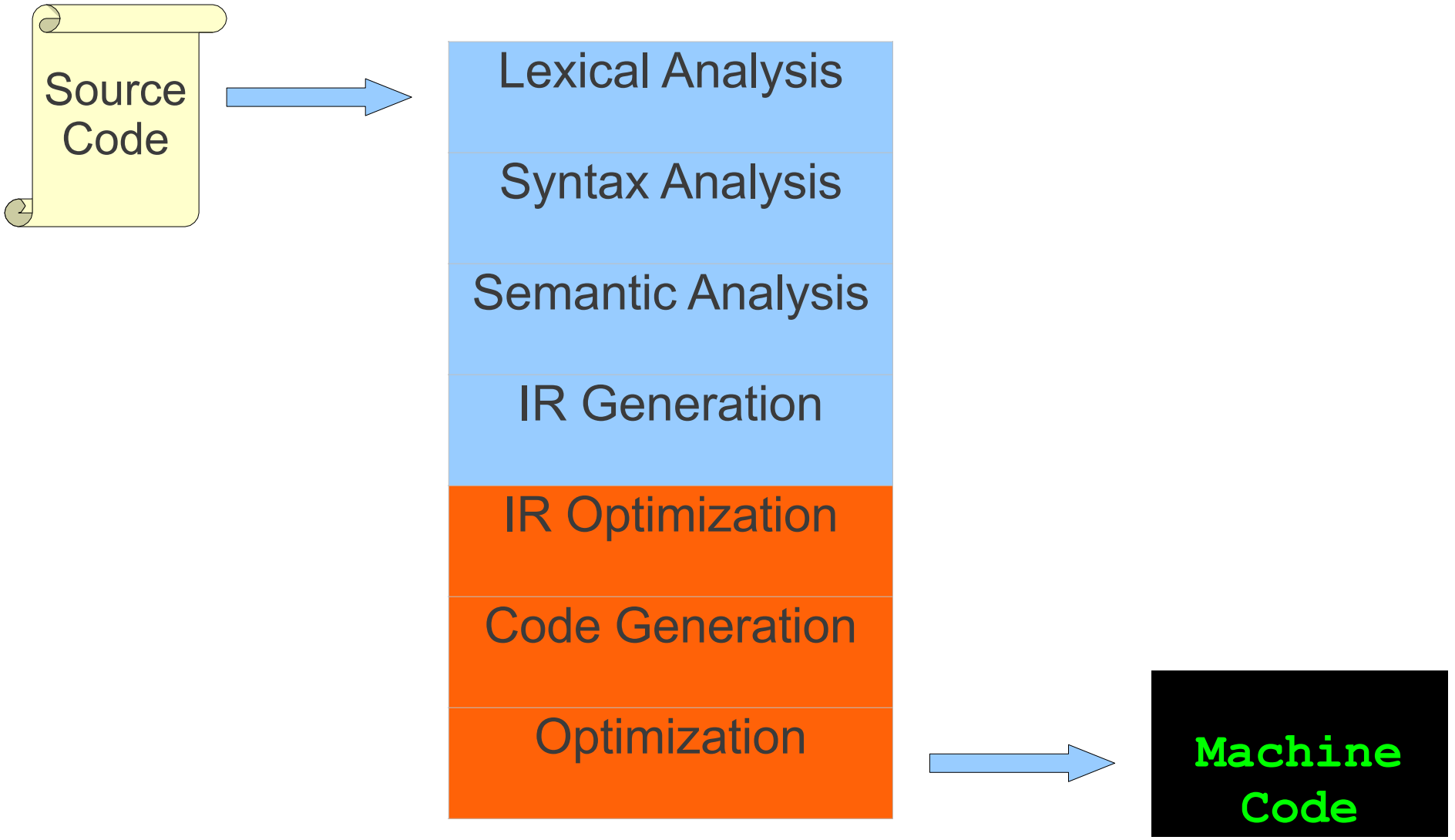
Optimization Example

X = Y * 0 is the same as X = 0

(the * operator is annihilated by zero)


Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code


Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code


Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code

```
while (y < z) {
  int x = a + b;
  y += x;
}
```

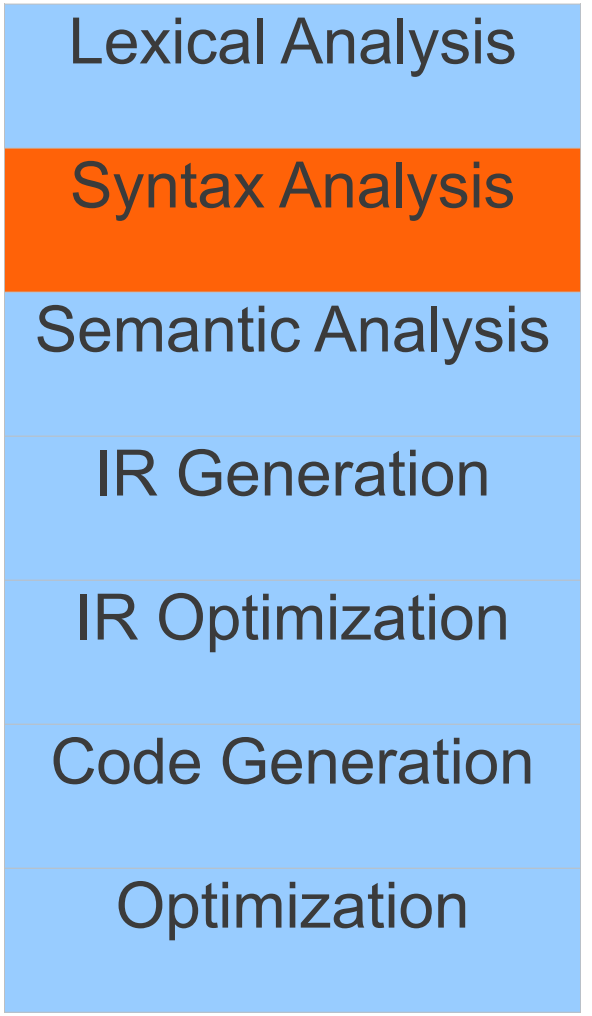
```
while (y < z) {
  int x = a + b;
  y += x;
}
```

```
while (y < z) {
  int x = a + b;
  y += x;
}
```

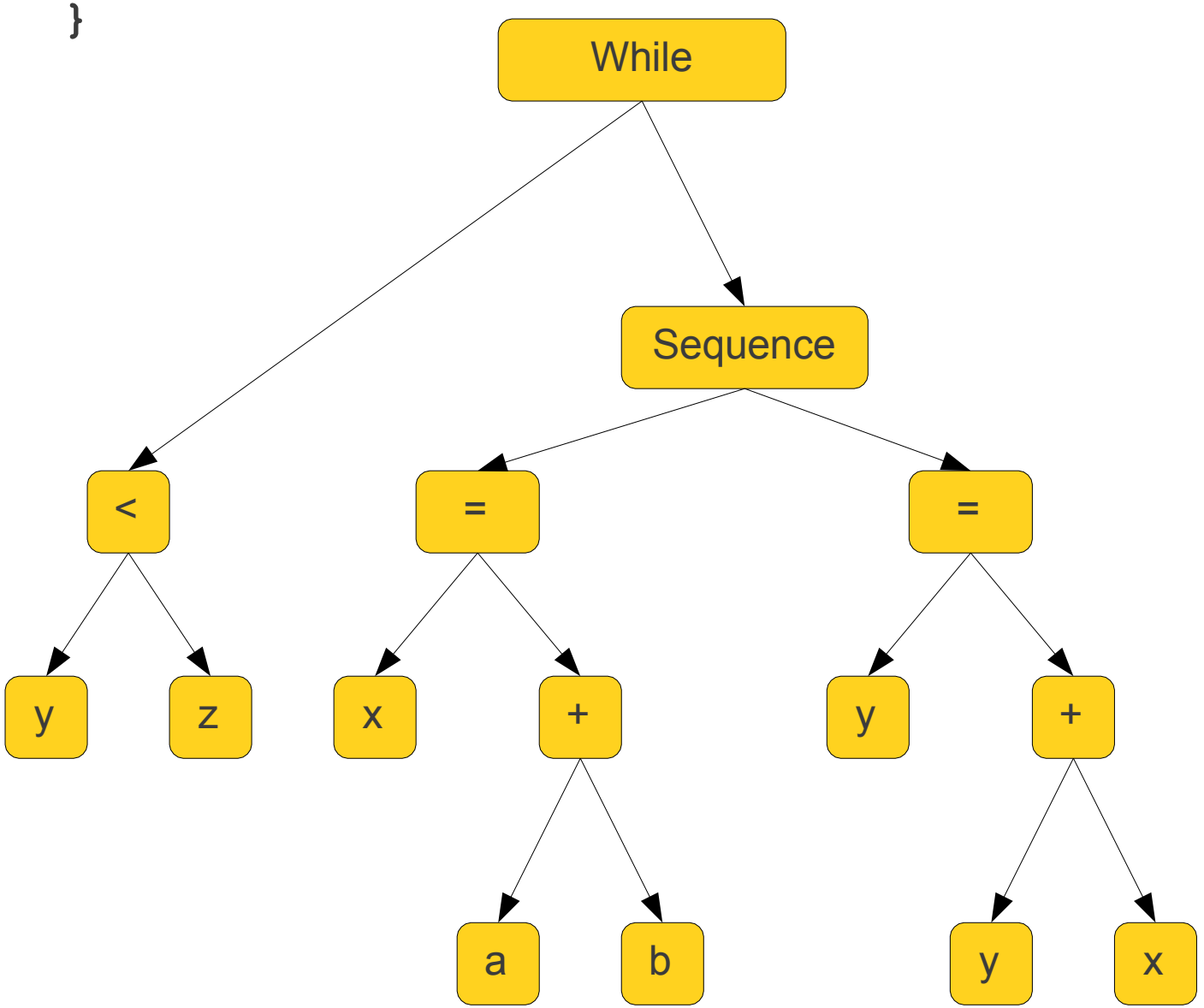
```
T_While
T_LeftParen
T_Identifier y
T_Less
T_Identifier z
T_RightParen
T_OpenBrace
T_Int
T_Identifier x
T_Assign
T_Identifier a
T_Plus
T_Identifier b
T_Semicolon
T_Identifier y
T_PlusAssign
T_Identifier x
T_Semicolon
T_CloseBrace
```

```
while (y < z) {
  int x = a + b;
  y += x;
}
```

```
T_While
T_LeftParen
T_Identifier y
T_Less
T_Identifier z
T_RightParen
T_OpenBrace
T_Int
T_Identifier x
T_Assign
T_Identifier a
T_Plus
T_Identifier b
T_Semicolon
T_Identifier y
T_PlusAssign
T_Identifier x
T_Semicolon
T_CloseBrace
```

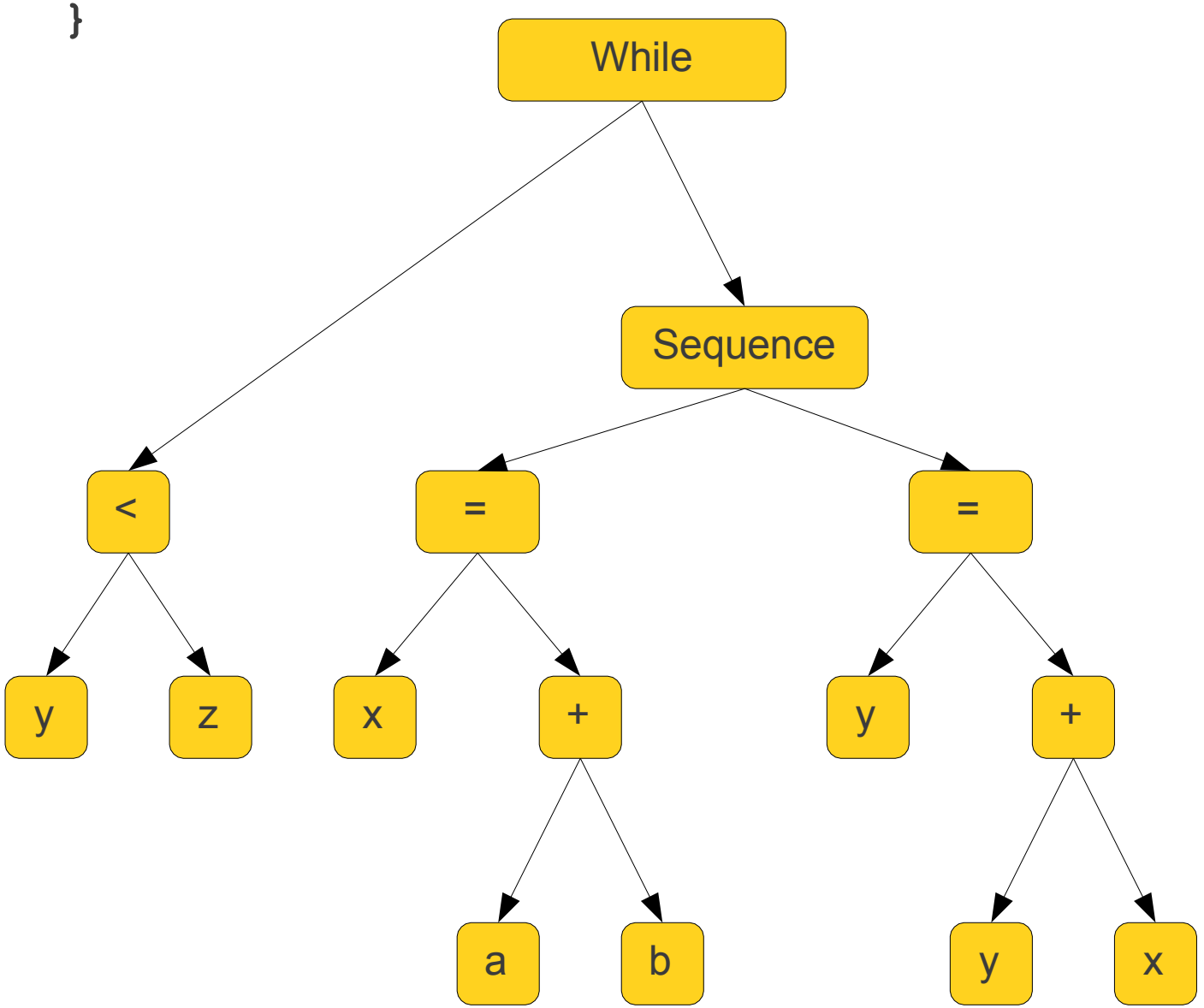


```
while (y < z) {
  int x = a + b;
  y += x;
}
```



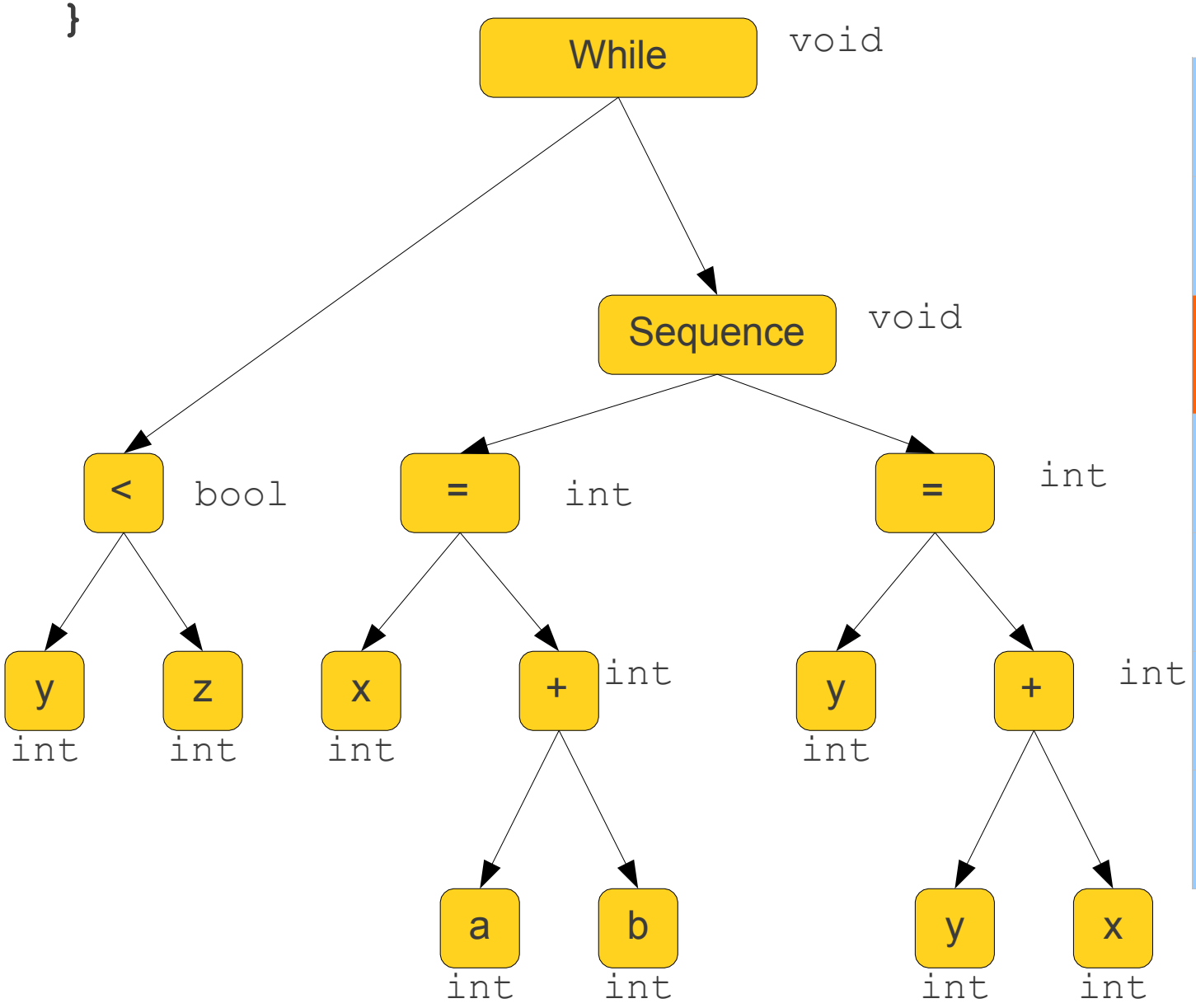
- Lexical Analysis
- **Syntax Analysis**
- Semantic Analysis
- IR Generation
- IR Optimization
- Code Generation
- Optimization

```
while (y < z) {
  int x = a + b;
  y += x;
}
```



- Lexical Analysis
- Syntax Analysis
- **Semantic Analysis**
- IR Generation
- IR Optimization
- Code Generation
- Optimization

```
while (y < z) {
  int x = a + b;
  y += x;
}
```



Lexical Analysis

Syntax Analysis

Semantic Analysis

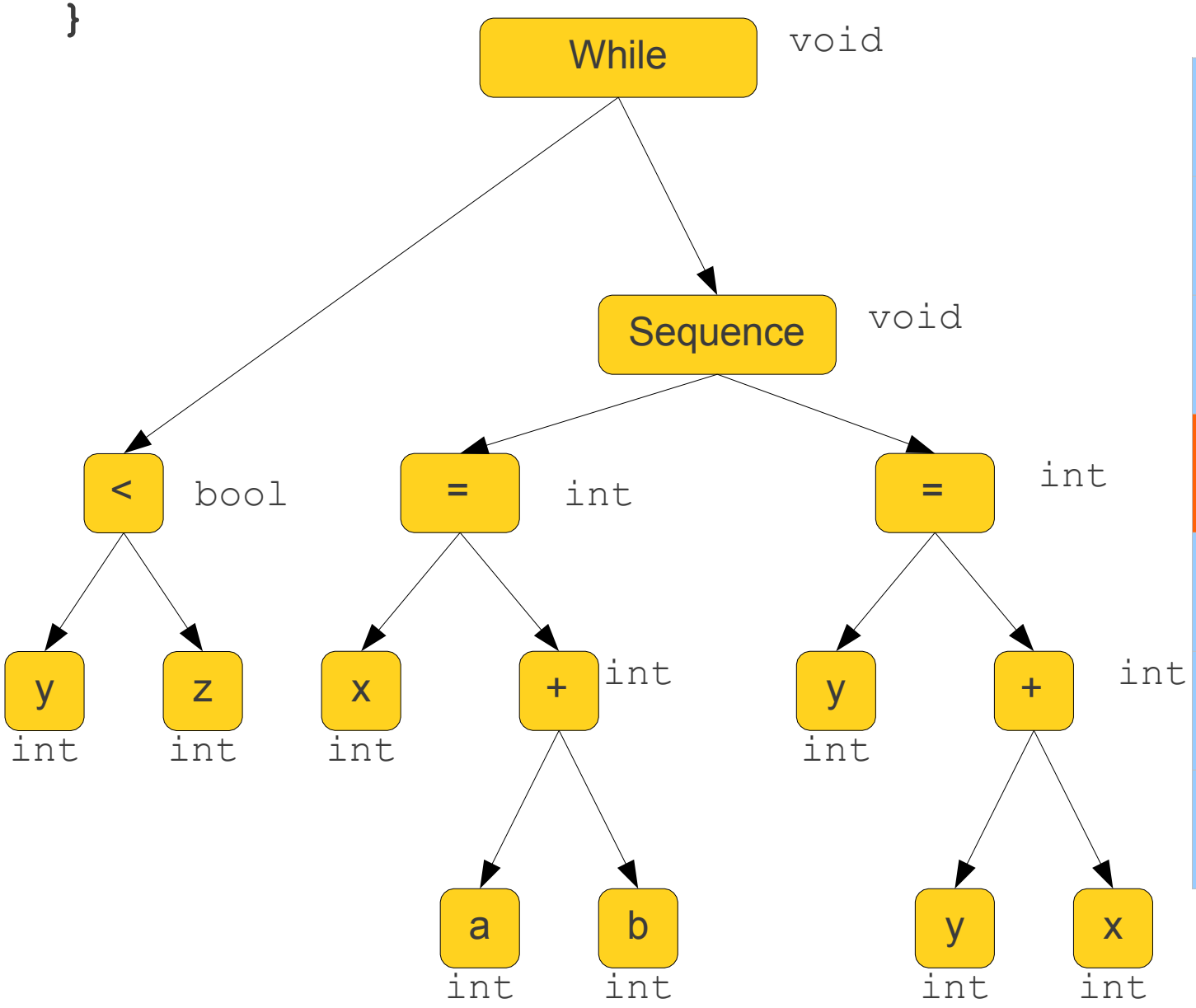
IR Generation

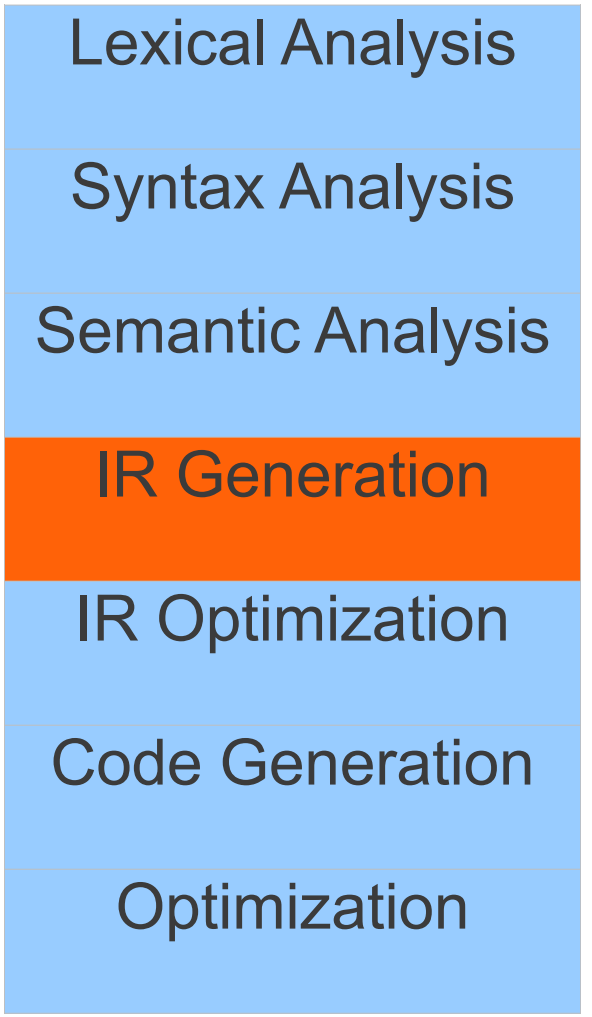
IR Optimization

Code Generation

Optimization

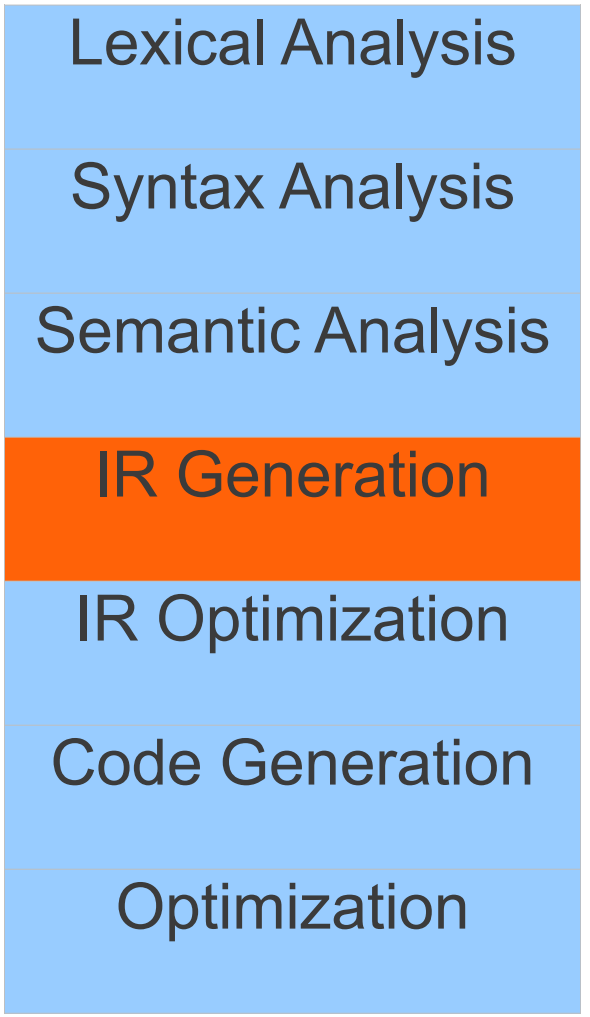
```
while (y < z) {
  int x = a + b;
  y += x;
}
```





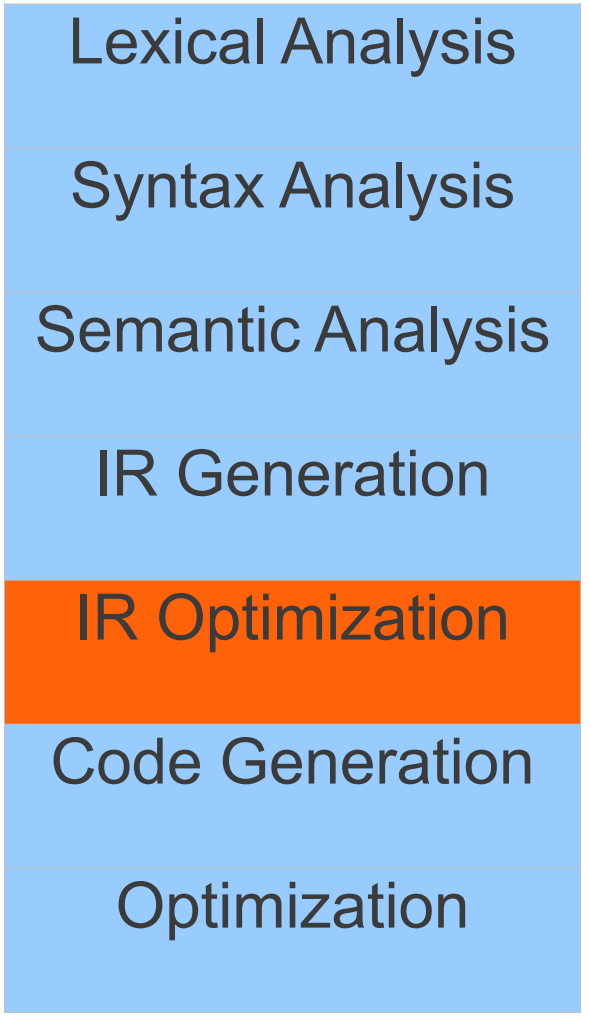
```
while (y < z) {
  int x = a + b;
  y += x;
}
```

```
Loop: x    = a+b
      y    = x+y
      _t1 = y < z
      if _t1 goto Loop
```



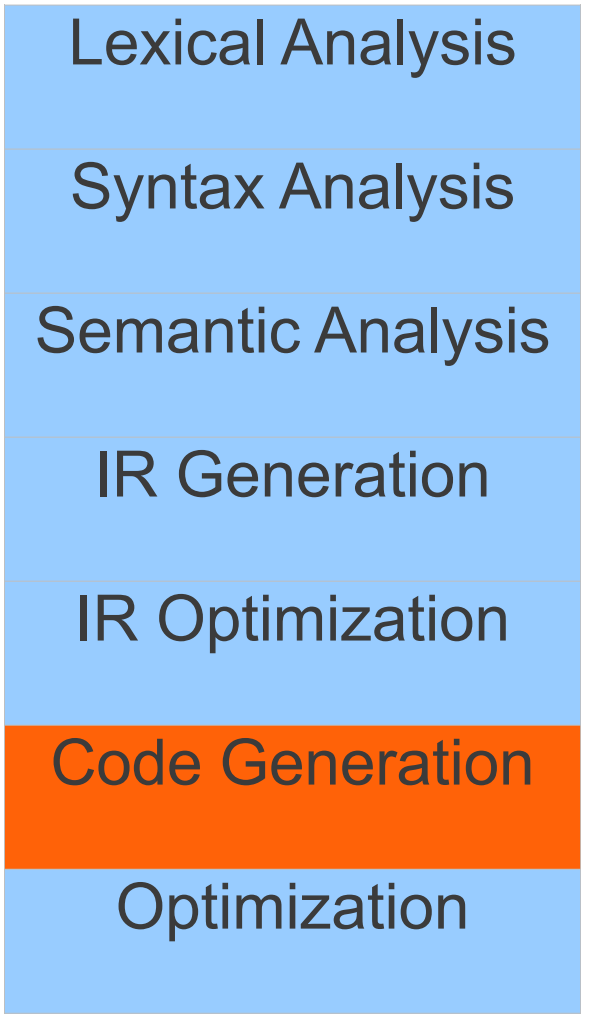
```
while (y < z) {
  int x = a + b;
  y += x;
}
```

```
      x    = a+b
Loop: y    = x+y
      _t1  = y < z
      if _t1 goto Loop
```



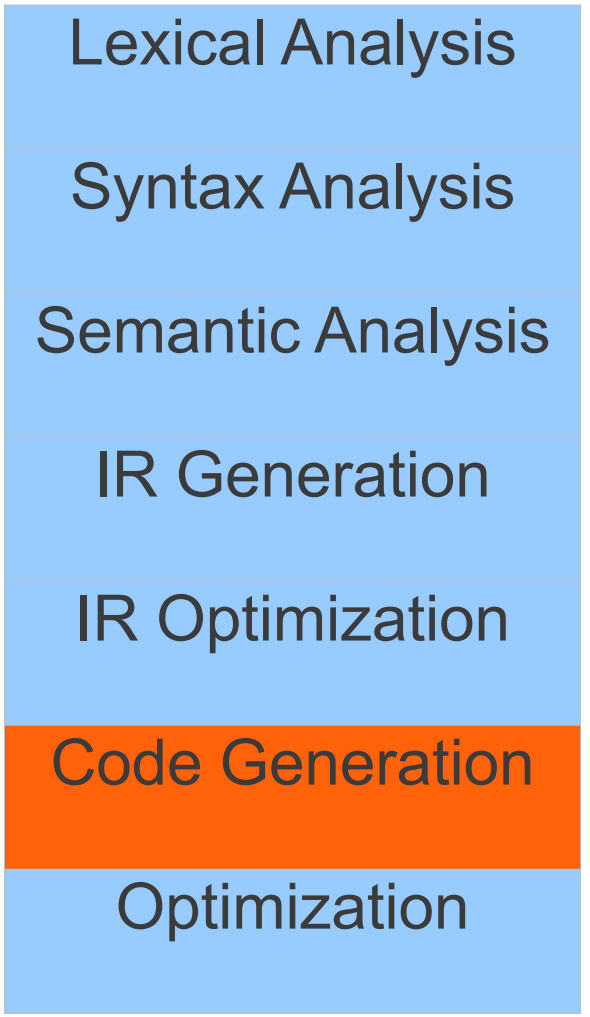
```
while (y < z) {
  int x = a + b;
  y += x;
}
```

```
        x    = a+b
Loop:   y    = x+y
        _t1  = y < z
        if _t1 goto Loop
```



```
while (y < z) {
  int x = a + b;
  y += x;
}
```

```
        add $1, $2, $3
Loop:   add $4, $1, $4
        slt $6, $4, $5
        beq $6, loop
```

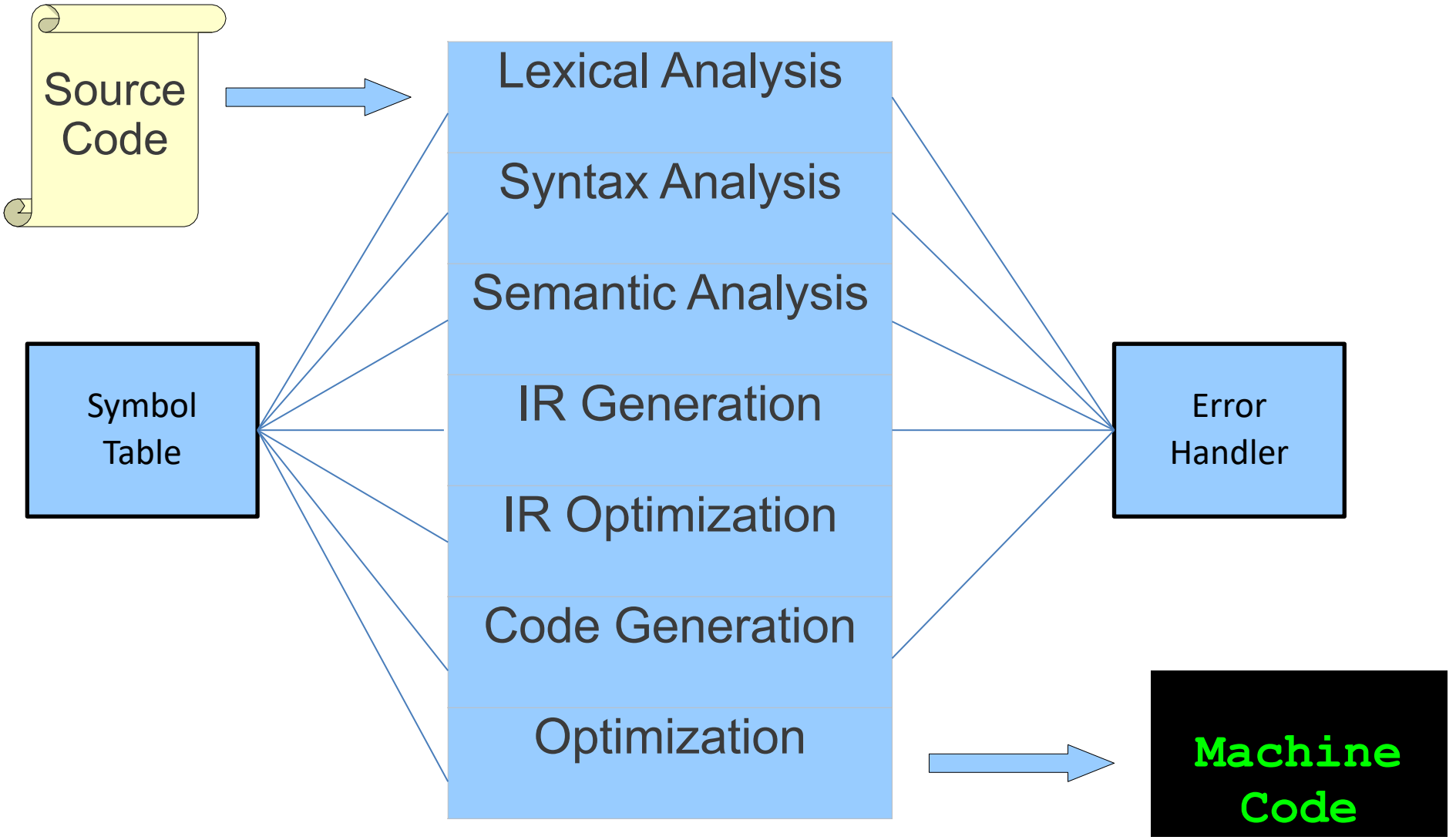


```
while (y < z) {
  int x = a + b;
  y += x;
}
```

```
      add $1, $2, $3
Loop: add $4, $1, $4
      slt $6, $4, $5
      beq $6, loop
```

```
while (y < z) {
  int x = a + b;
  y += x;
}
```

```
      add $1, $2, $3
Loop: add $4, $1, $4
      blt $4, $5, loop
```

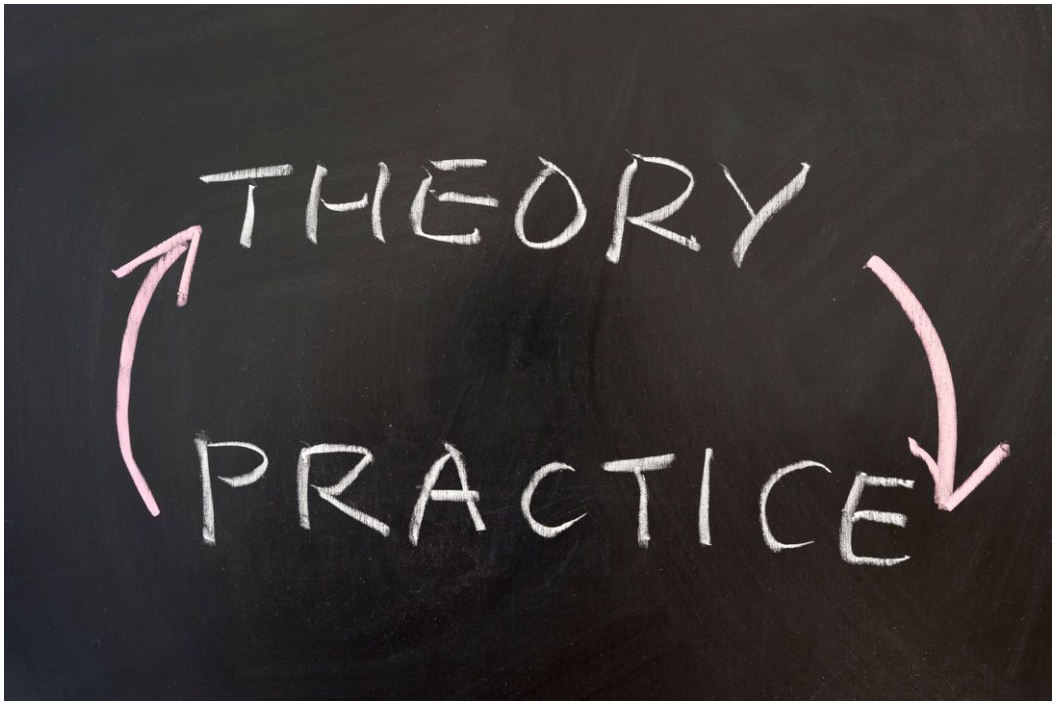

Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Symbol Table
Error Handler
Machine Code

Course Theme

- Studying design and implementation a very complicated software.
- We break it apart!
- We are going to meet weird-useful language features.

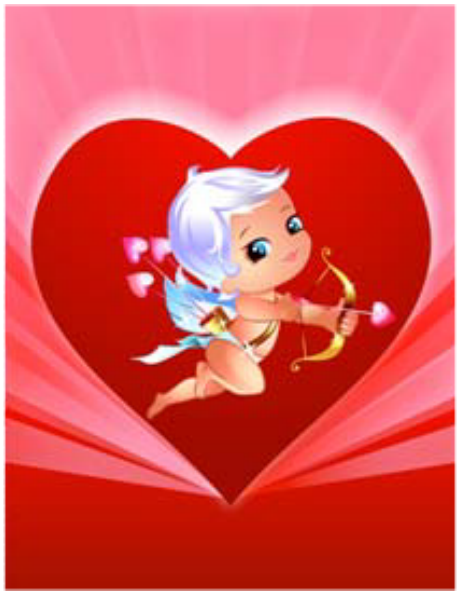
Course Theme

- We use theory as much as possible to survive.
- Excellency at programming is needed.



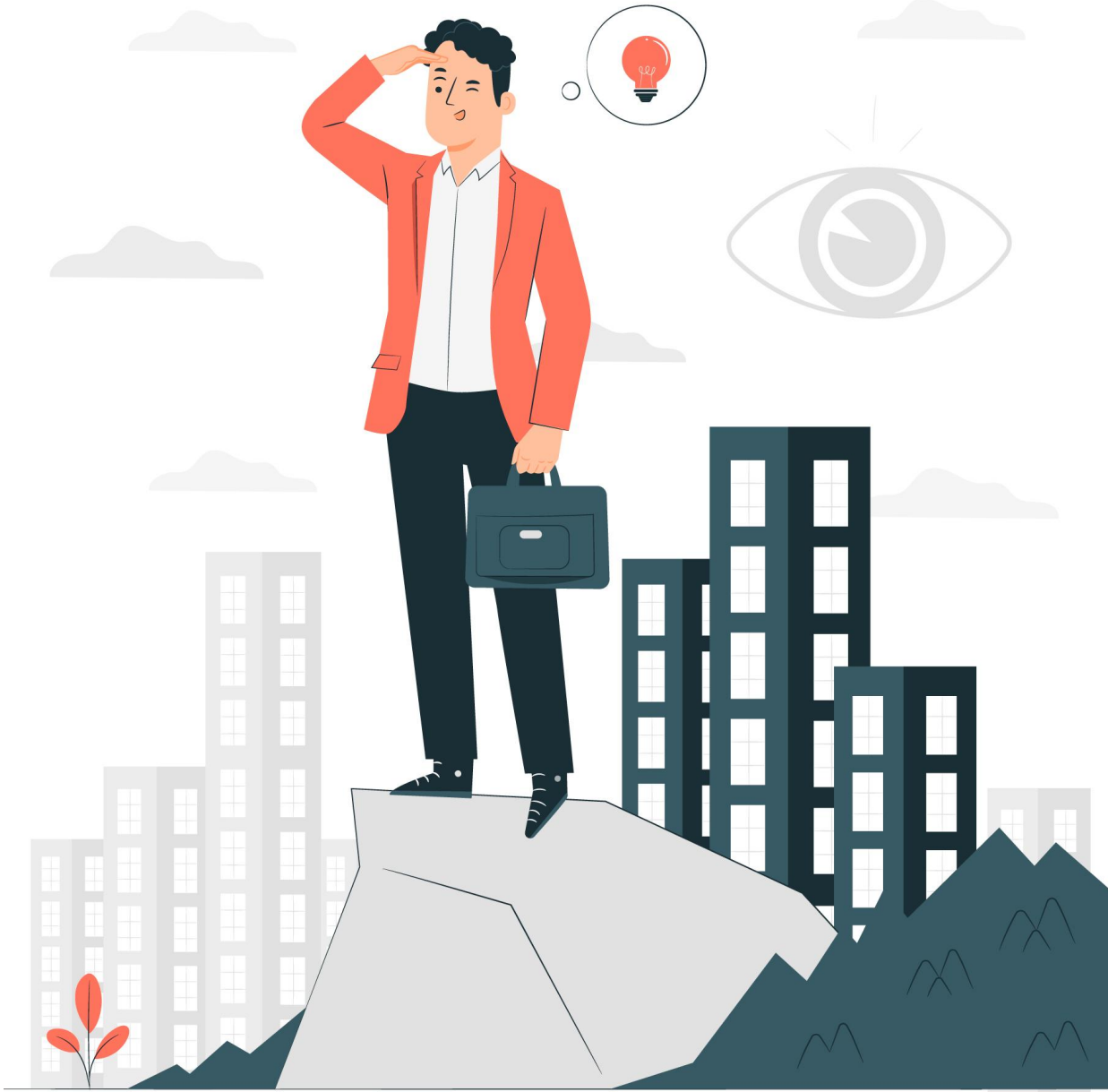
Why Take Programming Languages and Compilers?

To appreciate the marriage of theory and practice



"Theory and practice are not mutually exclusive; they are intimately connected. They live together and support each other." **[D.E. Knuth, 1989]**

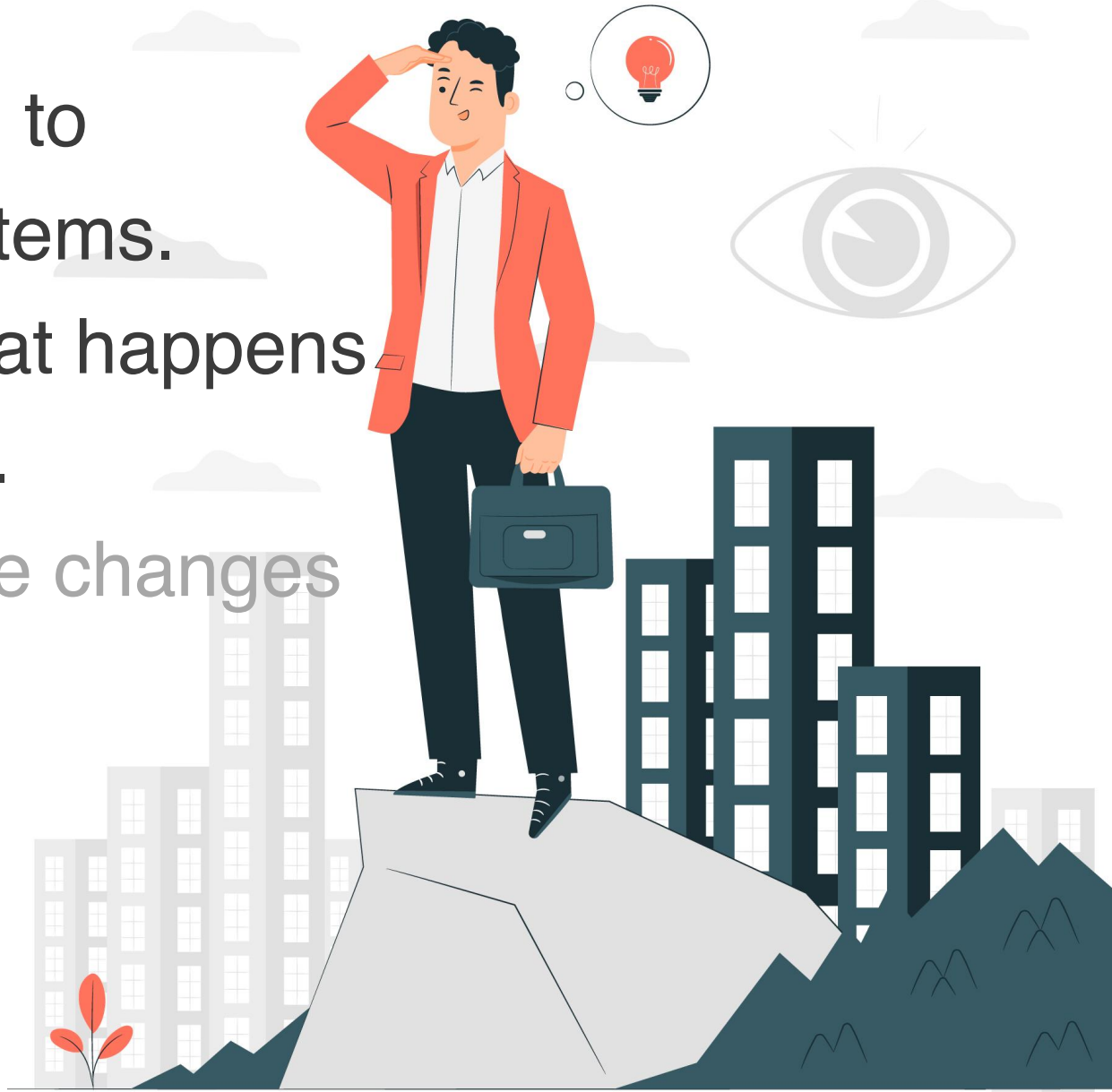
Why Study Compilers?



Why Study Compilers?

We Seek for a vision to

- build better systems.
- Understand what happens under the hood.
- Cope with future changes and era.



Why Study Compilers?

Why Study Compilers?

- Build a **large, ambitious software system**.

Why Study Compilers?

- Build a **large, ambitious software system**.
- Compiler Study Trains **Good Developers**.

Why Study Compilers?

- Build a **large, ambitious software system**.
- Compiler Study Trains **Good Developers**.
- See theory **come to life**.

Why Study Compilers?

Why Study Compilers?

- Learn how to **build programming languages**.

Why Study Compilers?

- Learn how to **build programming languages**.
- Learn **how programming languages work**.

Why Study Compilers?

- Learn how to **build programming languages**.
- Learn **how programming languages work**.
- Learn **tradeoffs in language design**.

Why Study Compilers?

Why Study Compilers?

- Reasoning about programs makes better programmers.

Why Study Compilers?

- Reasoning about programs makes better programmers.
- **Tool building**: there are programmers and there are tool builders…

Why Study Compilers?

- Reasoning about programs makes better programmers.
- **Tool building**: there are programmers and there are tool builders…
- Transformable Skills; It is not all about programming: Javadoc comments to HTML, Server responds to net protocols and etc.

Why Study Compilers?

- Reasoning about programs makes better programmers.
- **Tool building**: there are programmers and there are tool builders…
- Transformable Skills; It is not all about programming: Javadoc comments to HTML, Server responds to net protocols and etc.

This Course Adapted form Stanford CS 143 And MIT CS 6.s081

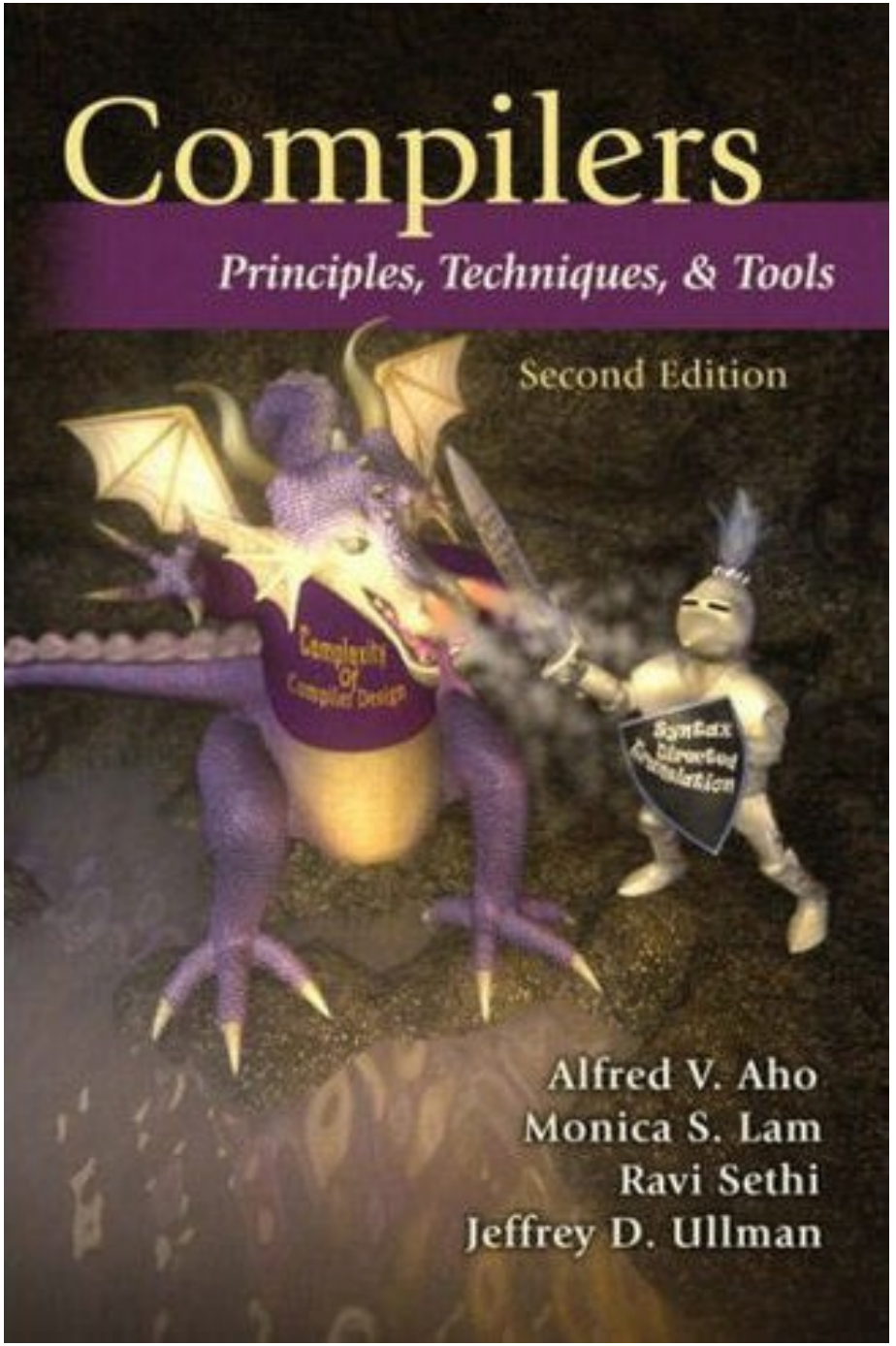
(but with changes!)

TA: Soroosh Zare

(ihaveint [] gmail)

Course Page: http://foroughmand.ir/?page_id=819

- Course info
- Important Announcements
- Calendar
- Some resources


Compilers
Principles, Techniques, & Tools
Second Edition
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman

- We use Quera and CW.

- We use Quera and CW.
- Quera for project, CW for homeworks and exams.

- We use Quera and CW.
- Quera for project, CW for homeworks and exams.

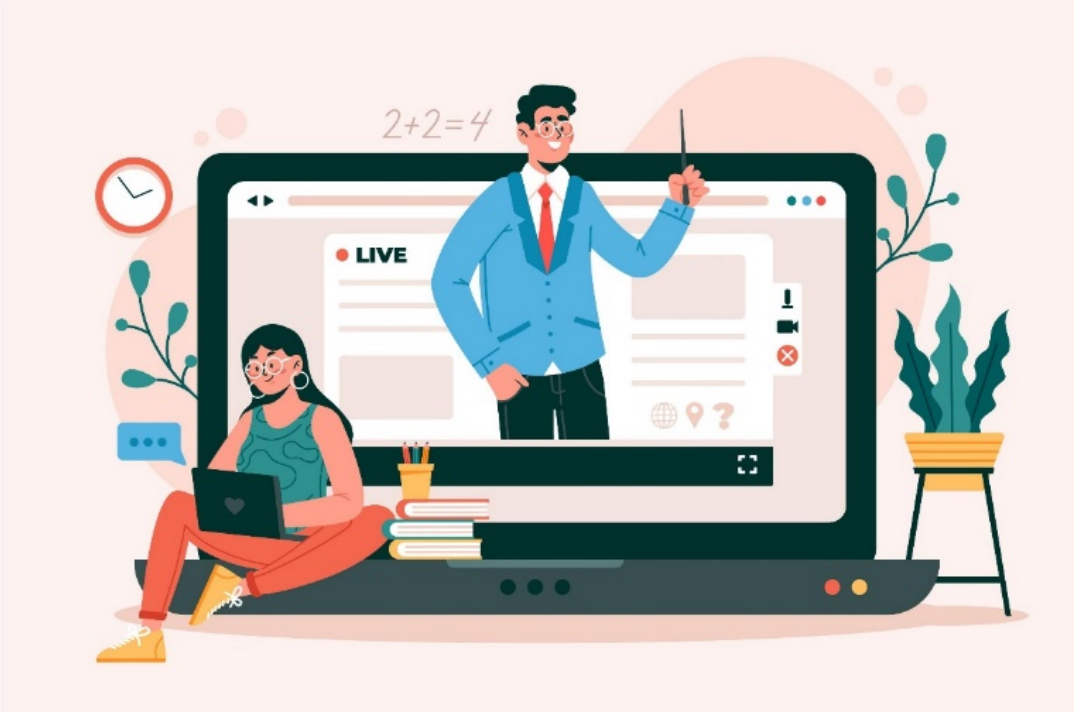

2+2=4
LIVE

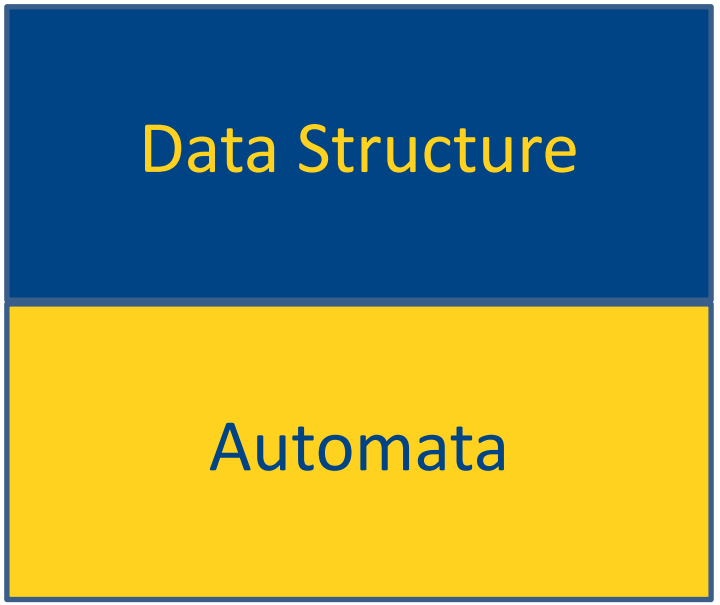
- We are all adults, no mandatory attendance.

- We are all adults, no mandatory attendance.
- But you are responsible for all announcements (noclass, exam date, projects deadline, HWs and etc.).

- We are all adults, no mandatory attendance.
- But you are responsible for all announcements (noclass, exam date, projects deadline, HWs and etc.).
- I can’t help you with your grade at the very end of the term.

- We are all adults, no mandatory attendance.
- But you are responsible for all announcements (noclass, exam date, projects deadline, HWs and etc.).
- I can’t help you with your grade at the very end of the term.
- I’ll try to record.





Maturity in programming and patience is also required.

بارم‌بندی

- تمرین: ۴ نمره
- آزمونک: ۲ نمره
- پروژه: ۶ نمره
- میان‌ترم: ۲ نمره
- پایان‌ترم: ۶ نمره

Homework

- About 7 assignments.
- About 2 weeks each
- About 3 problems.

- Do not postpone until the deadline!

- You have 10 days for late submission allowance.

LaTeX

Exams (mid-term, final, and Quiz)

Exams (mid-term, final, and Quiz)

Exams (mid-term, final, and Quiz)

- Reasonable exams, if you study you can get a good mark.

Exams (mid-term, final, and Quiz)

- Reasonable exams, if you study you can get a good mark.
- They are normal

Exams (mid-term, final, and Quiz)

- Reasonable exams, if you study you can get a good mark.
- They are normal
- You have samples

Exams (mid-term, final, and Quiz)

- Reasonable exams, if you study you can get a good mark.
- They are normal
- You have samples
- **No collaboration**!!!

Exams (mid-term, final, and Quiz)

- Reasonable exams, if you study you can get a good mark.
- They are normal
- You have samples
- **No collaboration**!!!

Exams (mid-term, final, and Quiz)

- Reasonable exams, if you study you can get a good mark.
- They are normal
- You have samples
- **No collaboration**!!!

Exams

Exams

- Open book and lectures in online exam.

Exams

- Open book and lectures in online exam.
- On regular, you can bring on paper.

Exams (mid-term, final, and Quiz)

Exams (mid-term, final, and Quiz)

- Mid-term: ٠٠/١٢/٢٣

Exams (mid-term, final, and Quiz)

- Mid-term: ٠٠/١٢/٢٣

Mini Quiz

- 2 Quizes
- With announcement.
- Takes 30 min.
- Not in group. Open slides.
- From previous or current lectures.
- Rules of mid-term and final applies.

Project

Project

- It is a complete compiler in 3-4 phases.
  - Lexical Analyzer (Scanner)
  - Parser
  - Code Generation

Project

- It is a complete compiler in 3-4 phases.
  - Lexical Analyzer (Scanner)
  - Parser
  - Code Generation

Project

- It is a complete compiler in 3-4 phases.
  - Lexical Analyzer (Scanner)
  - Parser
  - Code Generation

Project

LINUX

Project

- The generated code **must** run on a MIPS machine.

LINUX

Project

- The generated code **must** run on a MIPS machine.
- It must be a compiler!

LINUX

Project

- The generated code **must** run on a MIPS machine.
- It must be a compiler!
- It must **WORK** as the instructions.

LINUX

Project

- The generated code **must** run on a MIPS machine.
- It must be a compiler!
- It must **WORK** as the instructions.
- Linux is strongly recommended.

LINUX

Project

- The generated code **must** run on a MIPS machine.
- It must be a compiler!
- It must **WORK** as the instructions.
- Linux is strongly recommended.

LINUX

Project

- The generated code **must** run on a MIPS machine.
- It must be a compiler!
- It must **WORK** as the instructions.
- Linux is strongly recommended.

LINUX

Warning

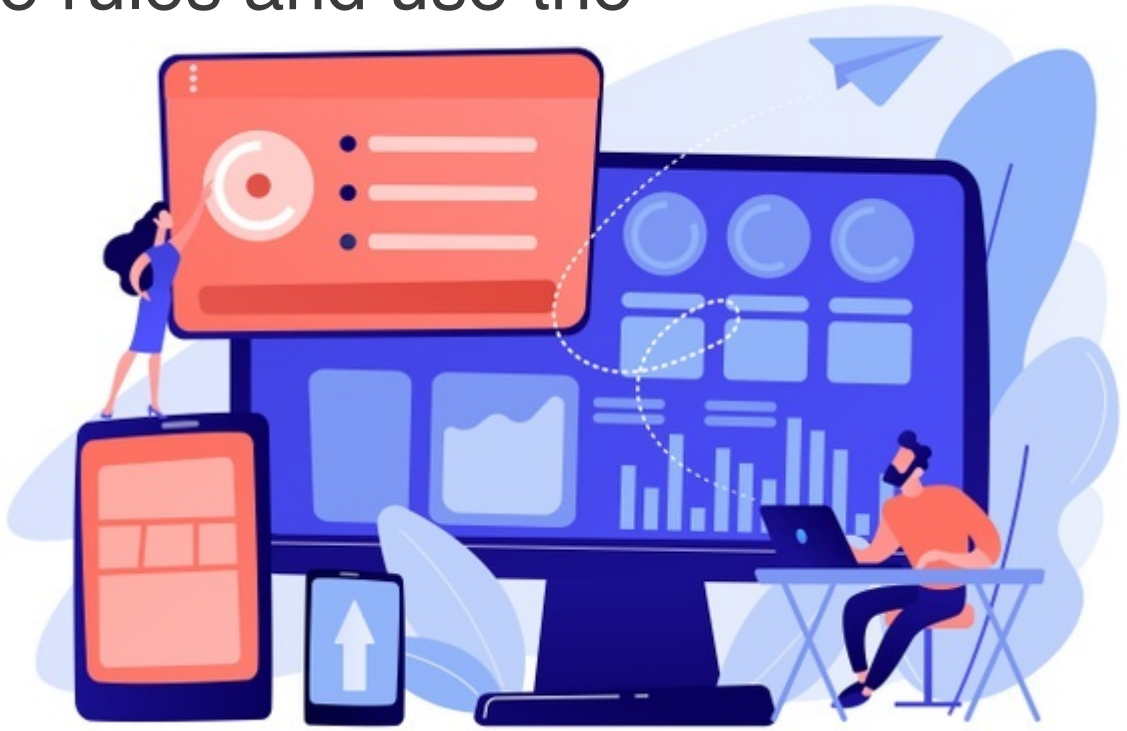
Warning

- This course has a big project. It is not easy. And, It might be time-consuming. Also, It requires patience and carefulness.

Warning

- This course has a big project. It is not easy. And, It might be time-consuming. Also, It requires patience and carefulness.

- Project must follow the rules and use the framework.



Warning

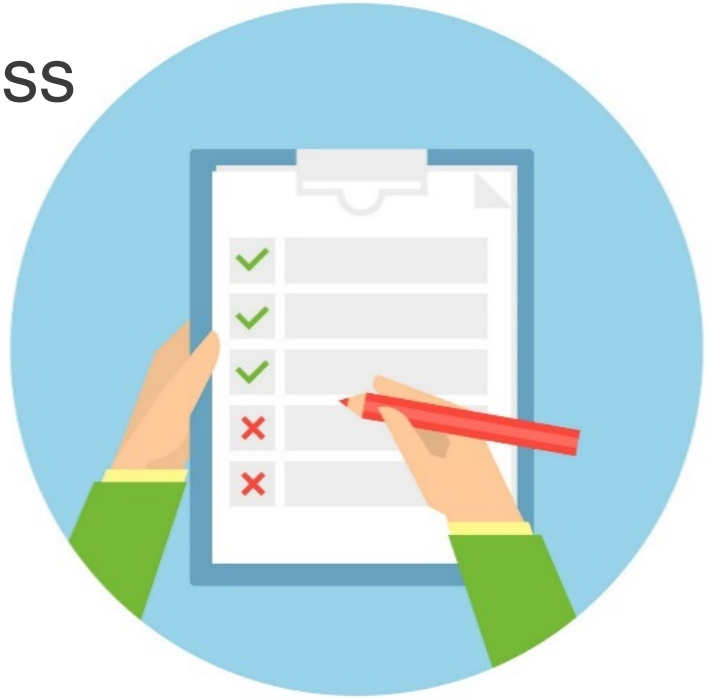
Warning

- We check just your outputs while grading. Especially in phase 3, your generated program **MUST** work correctly.

Warning

- We check just your outputs while grading. Especially in phase 3, your generated program **MUST** work correctly.

- We have many tests either you pass or fail each one.



Submission Guide line

Submission Guide line

- Just submit once unless you contact with the corresponding assistant.

Submission Guide line

- Just submit once unless you contact with the corresponding assistant.
- Don’t postpone everything to the last second (Parkinson’s Law).

Submission Guide line

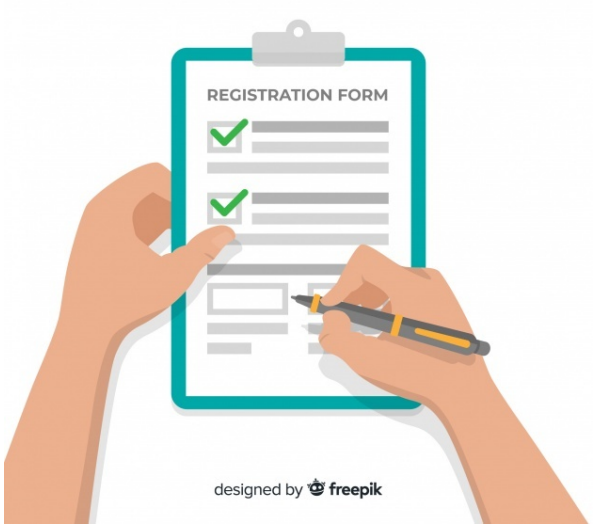
- Just submit once unless you contact with the corresponding assistant.
- Don’t postpone everything to the last second (Parkinson’s Law).
- Just upload it to **Quera** or **CW**. E-mails lost usually.

Project team

Project team

- Team of two or one person.

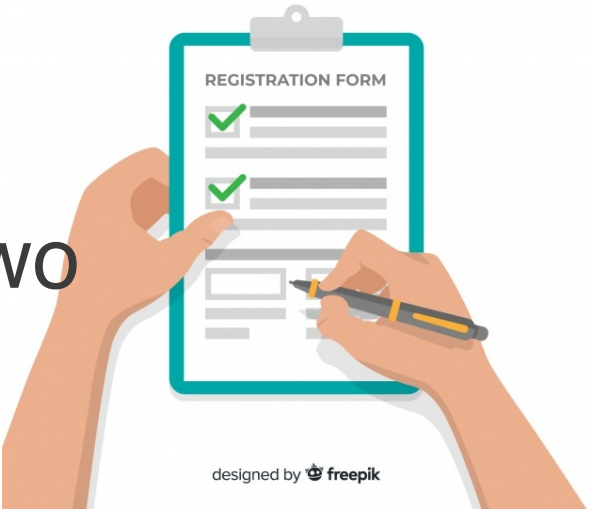
Group Rules


REGISTRATION FORM
designed by freepik


HELPFUL
TIPS!
designed by freepik.com

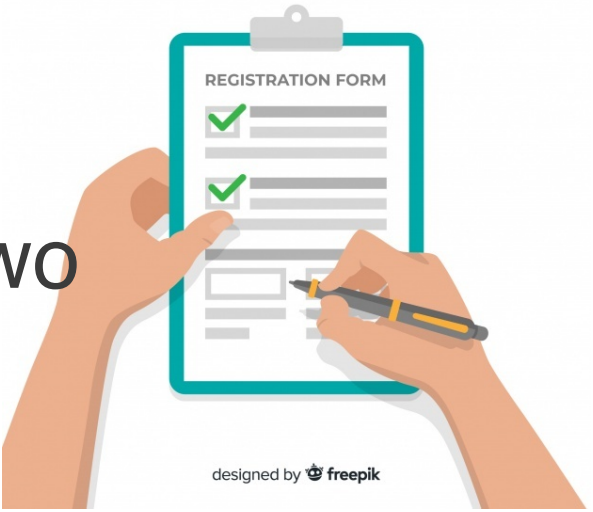
Group Rules

- Projects are for groups of one or two students.



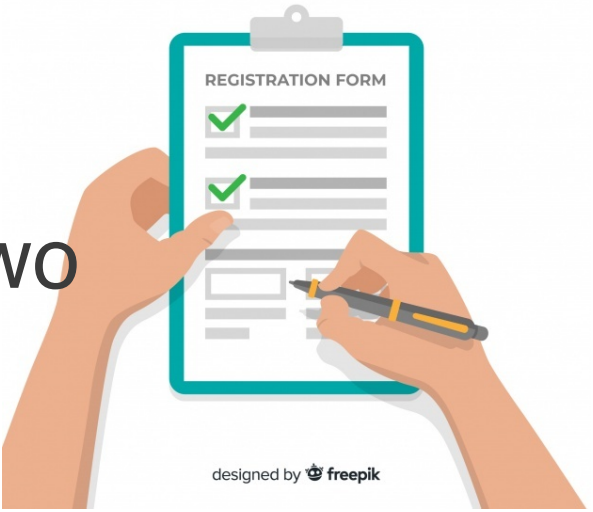
Group Rules

- Projects are for groups of one or two students.
- Pick an innovative name.



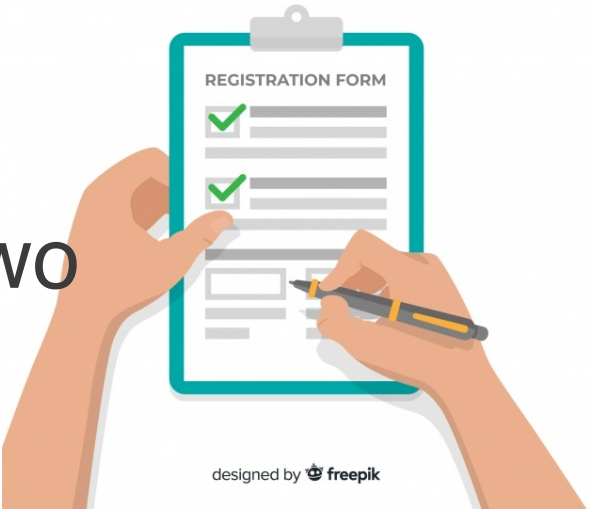
Group Rules

- Projects are for groups of one or two students.
- Pick an innovative name.
- Register Here.

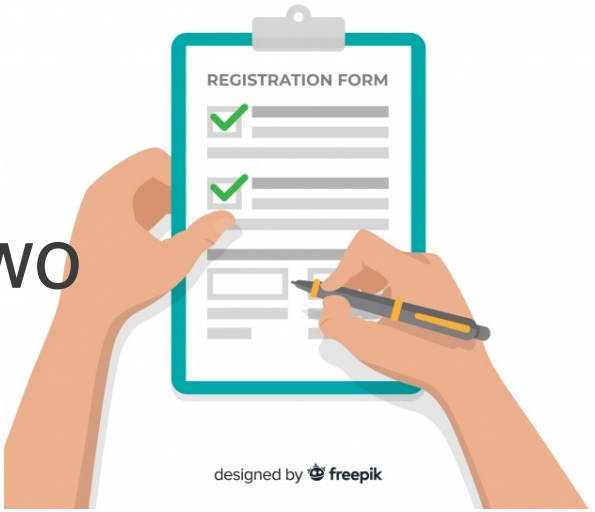


Group Rules

- Projects are for groups of one or two students.
- Pick an innovative name.
- Register Here.



Group Rules



- Projects are for groups of one or two students.
- Pick an innovative name.
- Register Here.

- A piece of advice: create a group with friends with whom you can work easily.

Some Issues

Some Issues

- Every one must contribute, or we reduce his/her grade.

Some Issues

- Every one must contribute, or we reduce his/her grade.

- You must do a fair portion and have a good vision about everything else.

Some Issues

- Every one must contribute, or we reduce his/her grade.
- You must do a fair portion and have a good vision about everything else.
- There is a delivery and we check it there.

تقلب

Use your own knowledge

Use your own knowledge

- Any Communication during exams is not allowed in any way using any media.

Use your own knowledge

- Any Communication during exams is not allowed in any way using any media.

- Searching the internet is not allowed neither.

Use your own knowledge

- Any Communication during exams is not allowed in any way using any media.
- Searching the internet is not allowed neither.
- You can use your book and notes during online exams. But you must obtain them before the exam.

Be Professional

Be Professional

- You must take exams **individually**.

Be Professional

- You must take exams **individually**.
- For the project:

Be Professional

- You must take exams **individually**.
- For the project:
  - Sharing/reading **any piece of code** is not allowed.

Be Professional

- You must take exams **individually**.
- For the project:
  - Sharing/reading **any piece of code** is not allowed.
  - You can learn from other materials, but you have to point out the source and the exact contribution of it in your project

Be Professional

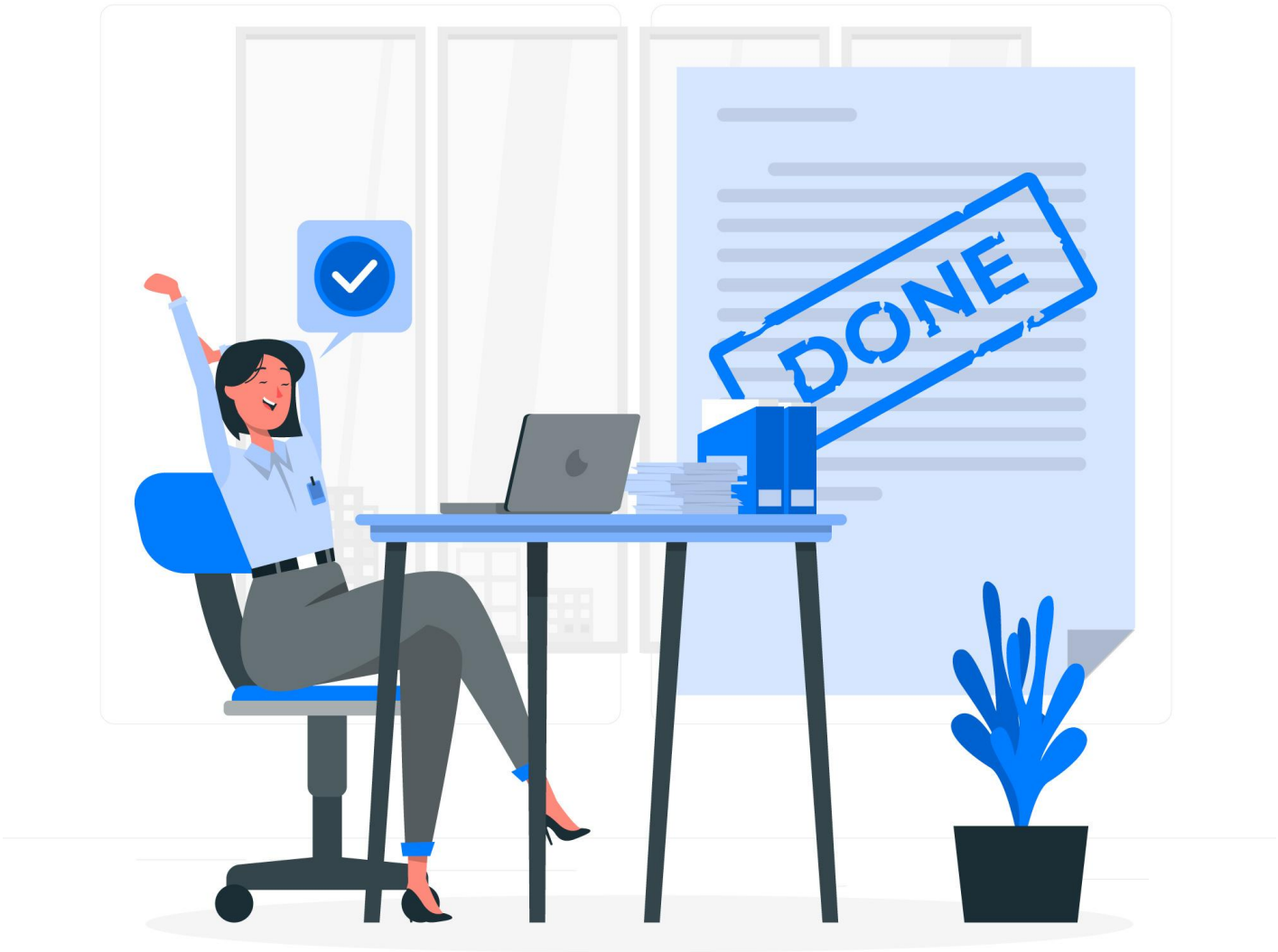
- You must take exams **individually**.
- For the project:
  - Sharing/reading **any piece of code** is not allowed.
  - You can learn from other materials, but you have to point out the source and the exact contribution of it in your project
- If we see any violation our reaction will be hard.

فیلم دکتر خسروی

What happens if…

- In projects the penalty is a -100 grade.
- In homework, plagiarism have a -100 grade.
- In the exams/quiz (One of these can occurs):
  - Send it to education affairs (you may receive 0.25 as the final grade).
  - Fail the course.
  - 0/-100 for the whole exam.
  - -100 for the question
    - in exceptional cases
- And no extra grade, if we decide.

PHD Comics: Extension


DONE