JECRC Foundation

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE

SUBJECT- COMPILER DESIGN

SEMESTER- 5TH SEM

VISSION AND MISSION OF INSTITUTE

To become a renowned center of outcome based learning and work towards academic, professional, cultural and social enrichment of the lives of individuals and communities

M1: Focus on evaluation of learning outcomes and motivate students to inculcate research aptitude by project based learning.

M2: Identify, based on informed perception of Indian, regional and global needs, the areas of focus and provide platform to gain knowledge and solutions.

M3: Offer opportunities for interaction between academia and industry.

M4: Develop human potential to its fullest extent so that intellectually capable and imaginatively gifted leaders can emerge in a range of professions.

VISION OF THE DEPARTMENT

To become renowned Centre of excellence in computer science and engineering and make competent engineers & professionals with high ethical values prepared for lifelong learning.

MISION OF THE DEPARTMENT

M1: To impart outcome based education for emerging technologies in the field of computer science and engineering.

M2: To provide opportunities for interaction between academia and industry.

M3: To provide platform for lifelong learning by accepting the change in technologies.

M4: To develop aptitude of fulfilling social responsibilities

PROGRAM OUTCOMES

Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

COURSE OUTCOME

CO1: Compare different phases of compiler and design lexical analyzer. CO2: Examine syntax and semantic analyzer by understanding grammars.

CO3: Illustrate storage allocation and its organization & analyze symbol table organization.

CO4: Analyze code optimization, code generation & compare various compilers.

CO-PO Mapping

Semester	Subject	Code	L/T/P	CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
V	COMPILER DESIGN	5CS4 - 02	L	1. Compare different phases of compiler and design lexical analyzer.	3	3	3	3	2	1	1	1	1	2	1	3
			L	2. Examine syntax and semantic analyzer and illustrate storage allocation and its organization	3	3	3	3	1	1	1	1	1	2	2	3
			L	3. Analyze symbol table organization, code optimization and code generator	3	3	3	3	2	1	1	1	1	2	2	3
			L	4.Compare and evaluate various compilers and analyzers	3	3	3	3	2	1	1	1	1	2	1	3

PROGRAM EDUCATIONAL OBJECTIVES:

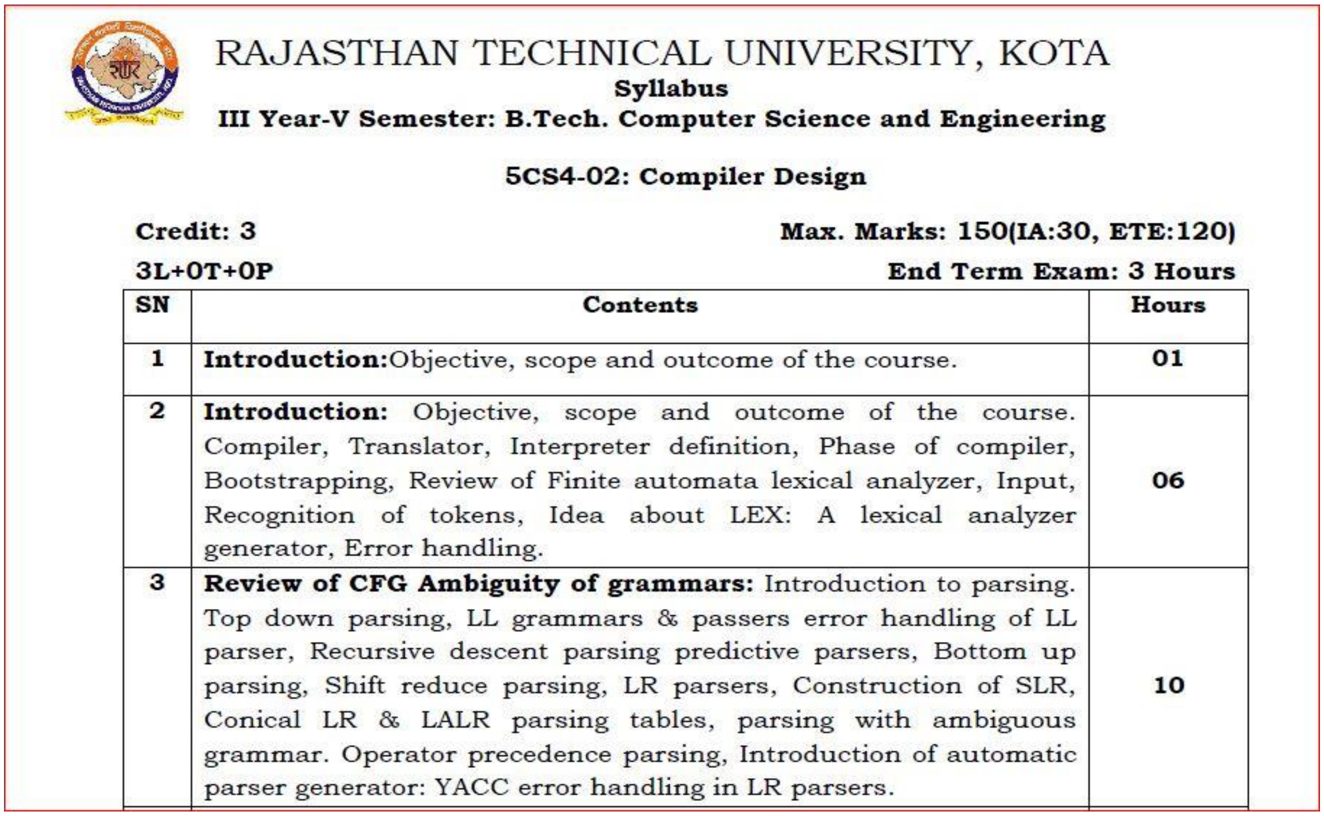
1. To provide students with the fundamentals of Engineering Sciences with more emphasis in **Computer Science &Engineering** by way of analyzing and exploiting engineering challenges.
2. To train students with good scientific and engineering knowledge so as to comprehend, analyze, design, and create novel products and solutions for the real life problems.
3. To inculcate professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, entrepreneurial thinking and an ability to relate engineering issues with social issues.
4. To provide students with an academic environment aware of excellence, leadership, written ethical codes and guidelines, and the self motivated life-long learning needed for a successful professional career.
5. To prepare students to excel in Industry and Higher education by Educating Students along with High moral values and Knowledge

PSO

PSO1. Ability to interpret and analyze network specific and cyber security issues, automation in real word environment.

PSO2. Ability to Design and Develop Mobile and Web-based applications under realistic constraints.

SYLLABUS



RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Syllabus

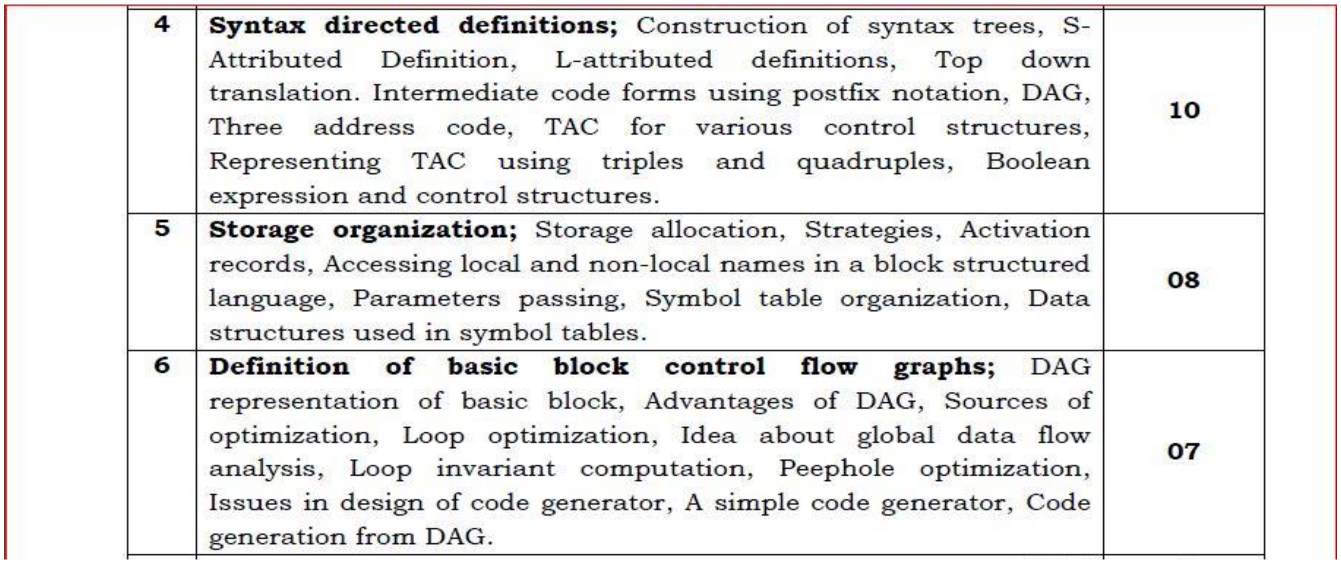
III Year-V Semester: B.Tech. Computer Science and Engineering

5CS4-02: Compiler Design

Credit: 3 **Max. Marks: 150(IA:30, ETE:120)**

3L+0T+0P **End Term Exam: 3 Hours**

SN	Contents	Hours
1	**Introduction:**Objective, scope and outcome of the course.	01
2	**Introduction:** Objective, scope and outcome of the course. Compiler, Translator, Interpreter definition, Phase of compiler, Bootstrapping, Review of Finite automata lexical analyzer, Input, Recognition of tokens, Idea about LEX: A lexical analyzer generator, Error handling.	06
3	**Review of CFG Ambiguity of grammars:** Introduction to parsing. Top down parsing, LL grammars & passers error handling of LL parser, Recursive descent parsing predictive parsers, Bottom up parsing, Shift reduce parsing, LR parsers, Construction of SLR, Conical LR & LALR parsing tables, parsing with ambiguous grammar. Operator precedence parsing, Introduction of automatic parser generator: YACC error handling in LR parsers.	10
4	**Syntax directed definitions;** Construction of syntax trees, S-Attributed Definition, L-attributed definitions, Top down translation. Intermediate code forms using postfix notation, DAG, Three address code, TAC for various control structures, Representing TAC using triples and quadruples, Boolean expression and control structures.	10
5	**Storage organization;** Storage allocation, Strategies, Activation records, Accessing local and non-local names in a block structured language, Parameters passing, Symbol table organization, Data structures used in symbol tables.	08
6	**Definition of basic block control flow graphs;** DAG representation of basic block, Advantages of DAG, Sources of optimization, Loop optimization, Idea about global data flow analysis, Loop invariant computation, Peephole optimization, Issues in design of code generator, A simple code generator, Code generation from DAG.	07



LECTURE PLAN:

Subject: Compiler Design (5CS4 – 02) **Year/Sem: III/V**

Unit No./ Total lec. Req.	Topics	Lect. Req.
Unit-1 (6)	Compiler, Translator, Interpreter definition, Phase of compiler	1
	Introduction to one pass & Multipass compilers, Bootstrapping	1
	Review of Finite automata lexical analyzer, Input, buffering,	2
	Recognition of tokens, Idea about LEX:, GATE Questions	1
	A lexical analyzer generator, Error Handling, Unit Test	1
Unit-2 (17)	Review of CFG Ambiguity of grammars, Introduction to parsing	2
	Bottom up parsing Top down Parsing Technique	5
	Shift reduce parsing, Operator Precedence Parsing	2
	Recursive descent parsing predictive parsers	1
	LL grammars & passers error handling of LL parser	1
	Conical LR & LALR parsing tables	3
	parsing with ambiguous grammar, GATE Questions	2
	Introduction of automatic parser generator: YACC error handling in LR parsers, Unit Test	1
Unit 3- (7)	Syntax directed definitions; Construction of syntax trees	1
	L-attributed definitions, Top down translation	1
	Specification of a type checker, GATE Questions	1
	Intermediate code forms using postfix notation and three address code,	2
	Representing TAC using triples and quadruples, Translation of assignment statement.	1
	Boolean expression and control structures, Unit Test	1
Unit 4- (4)	Storage organization, Storage allocation, Strategies, Activation records,	1
	Accessing local and non local names in a block structured language	1
	Parameters passing, Symbol table organization, GATE Questions	1
	Data structures used in symbol tables, Unit Test	1
Unit 5- (6)	Definition of basic block control flow graphs,	1
	DAG representation of basic block, Advantages of DAG,	1
	Sources of optimization, Loop optimization Idea about global data flow analysis, Loop invariant computation, Loop invariant computation, Tutorial	2
	Peephole optimization, GATE Questions, Tutorial	1
	Issues in design of code generator, A simple code generator, Code generation from DAG., UNIT TEST, Revision	1

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE

Year & Sem – 3rd/ 5th sem

Subject – Compiler Design

Unit – 5

Code Generator: Introduction

- ❖**Code generator** converts the intermediate representation of source code into a form that can be readily executed by the machine.
- ❖A code generator is expected to generate a correct code.
- ❖Designing of code generator should be done in such a way so that it can be easily implemented, tested and maintained.

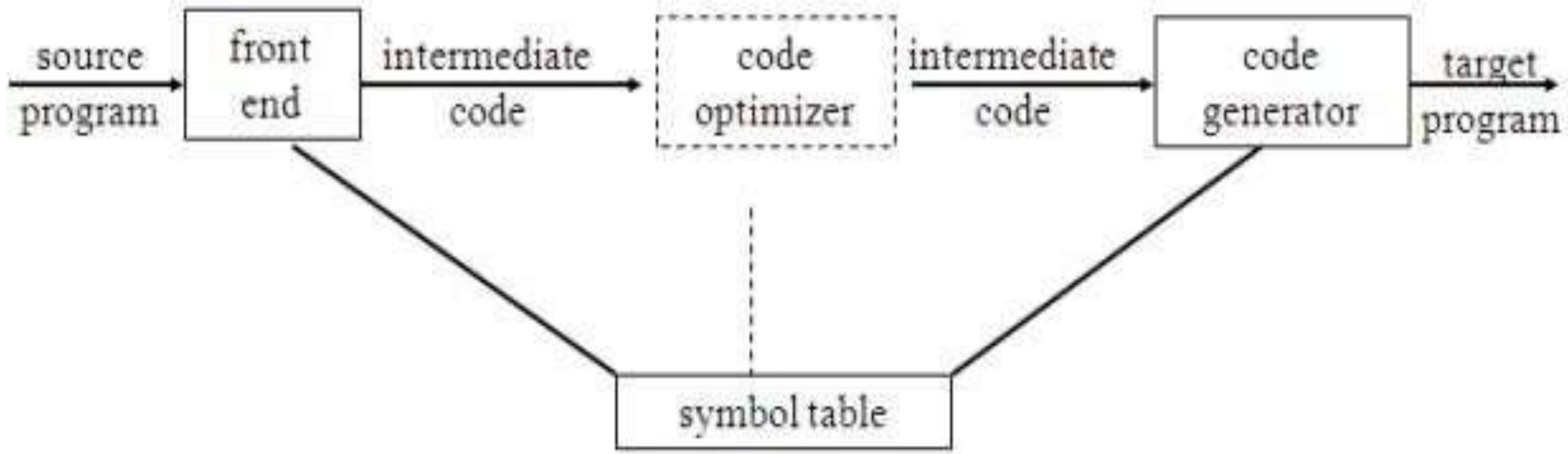
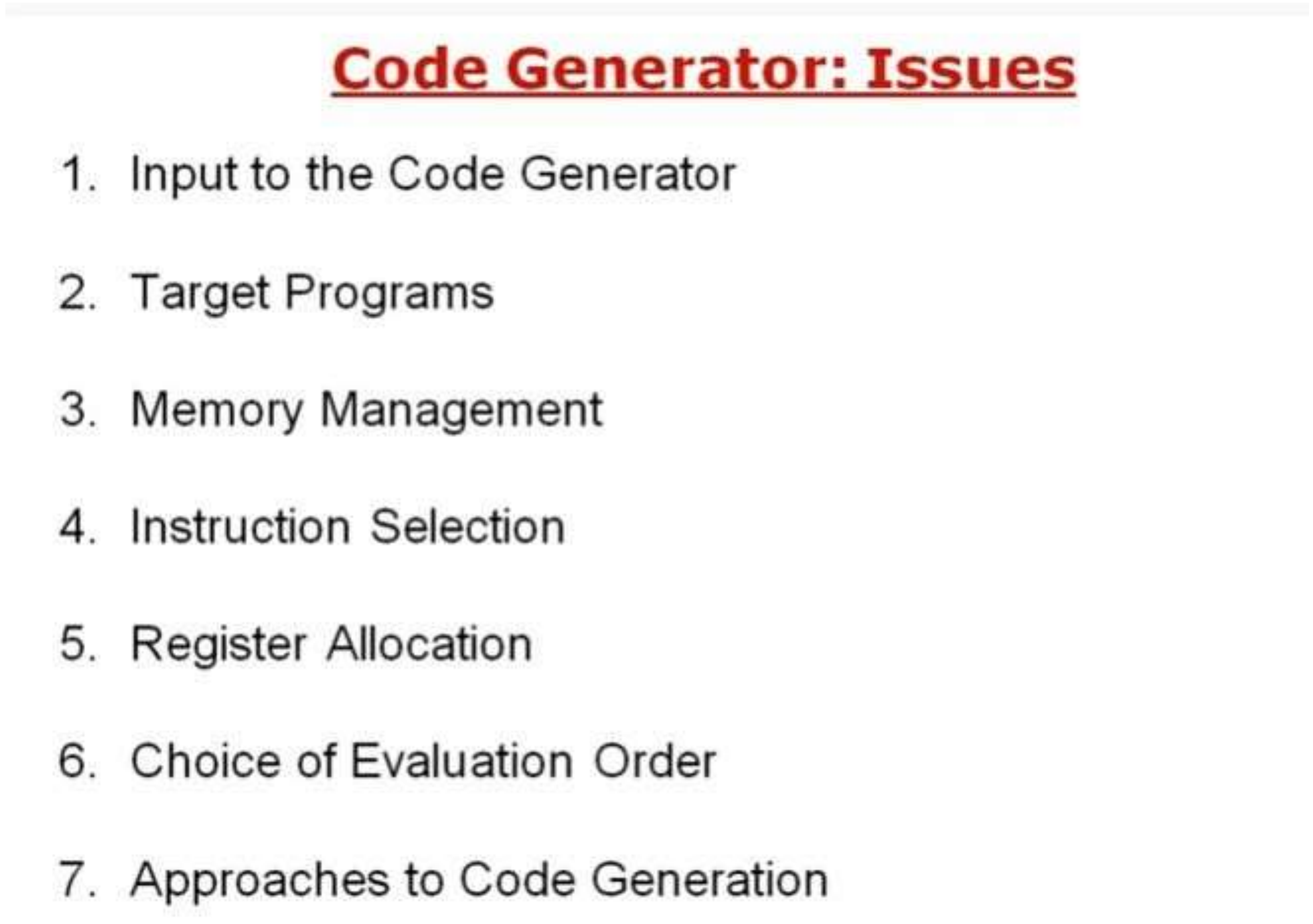
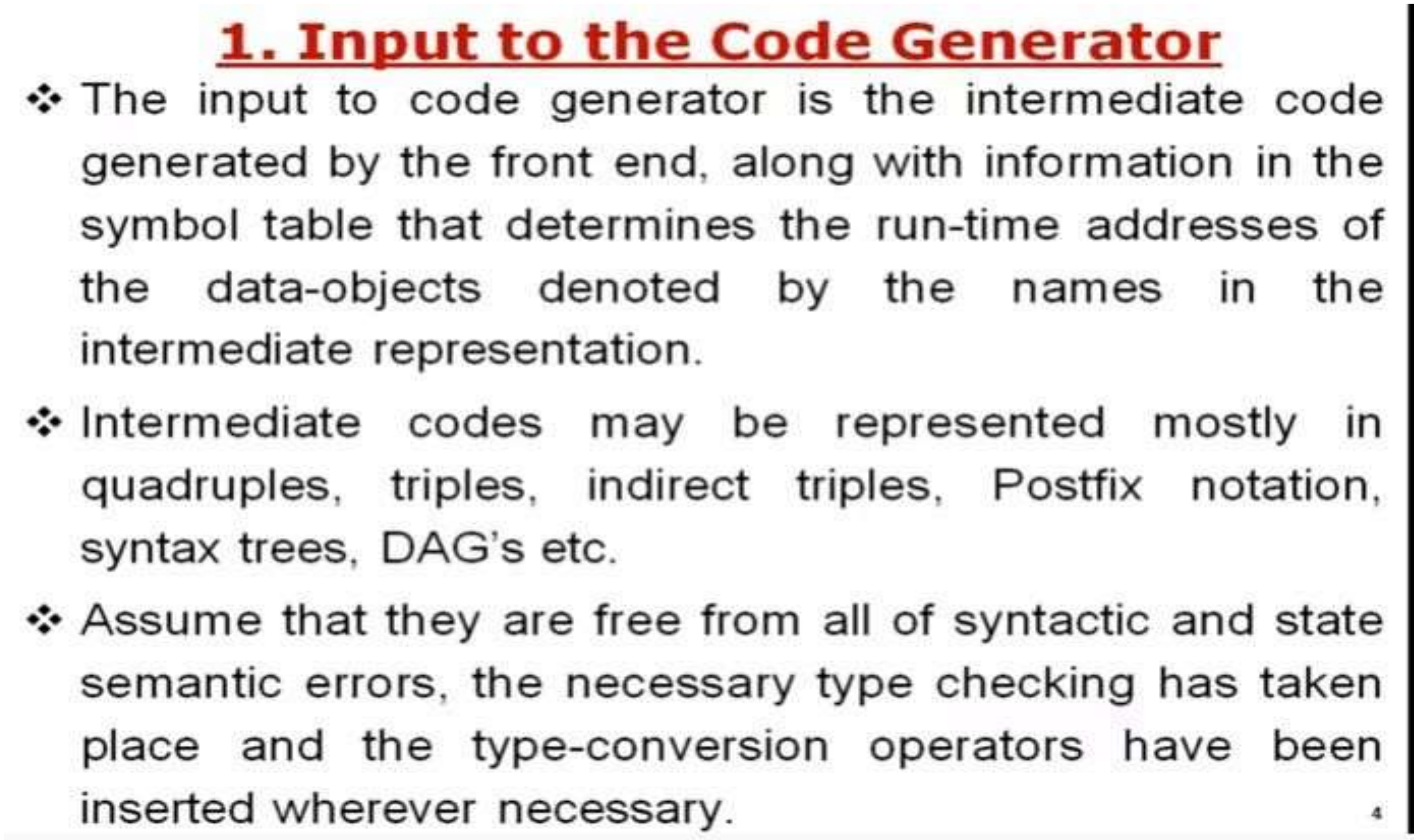


Fig. 4.1 Position of code generator



Code Generator: Issues

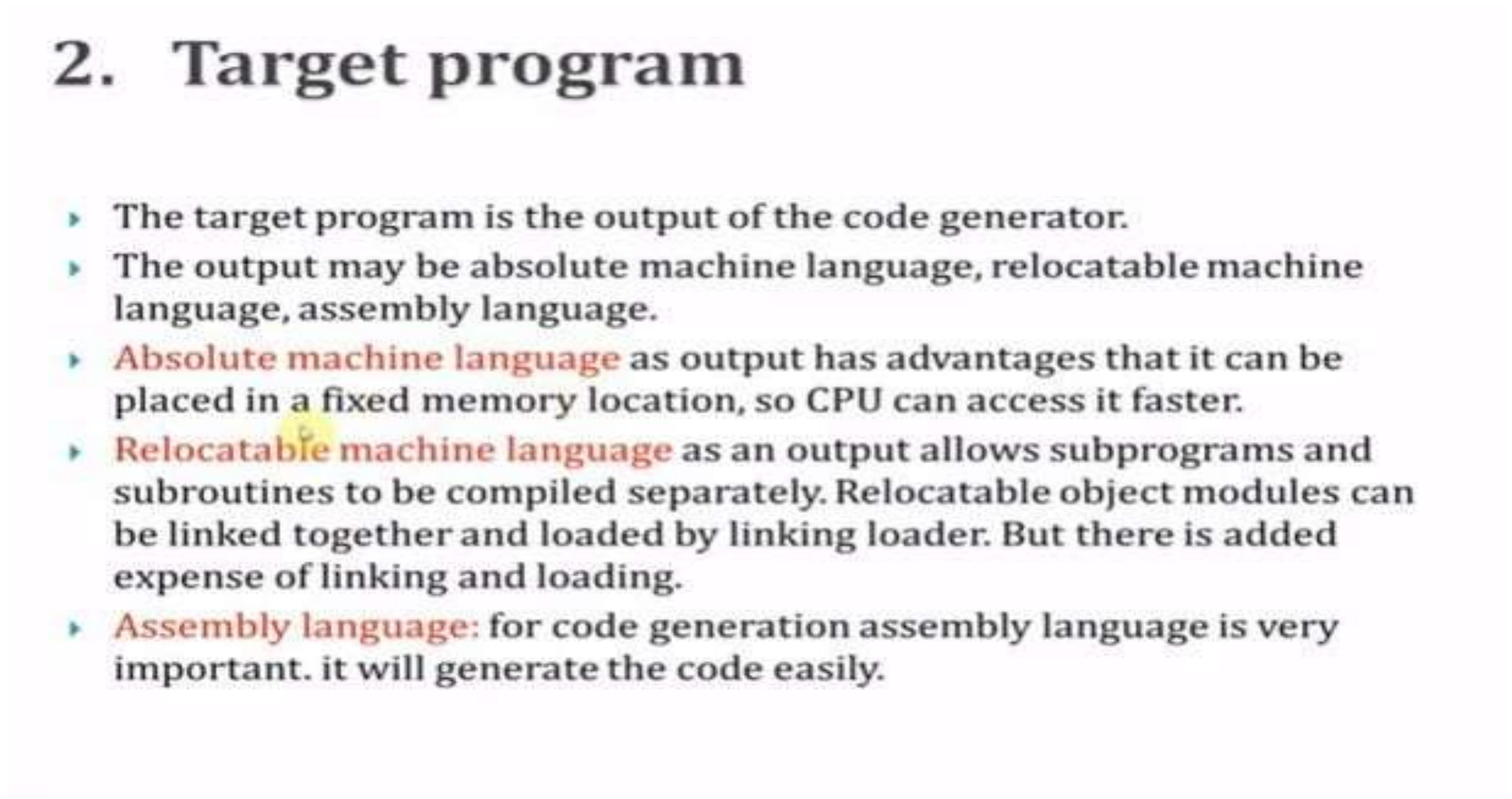
1. Input to the Code Generator
2. Target Programs
3. Memory Management
4. Instruction Selection
5. Register Allocation
6. Choice of Evaluation Order
7. Approaches to Code Generation



1. Input to the Code Generator

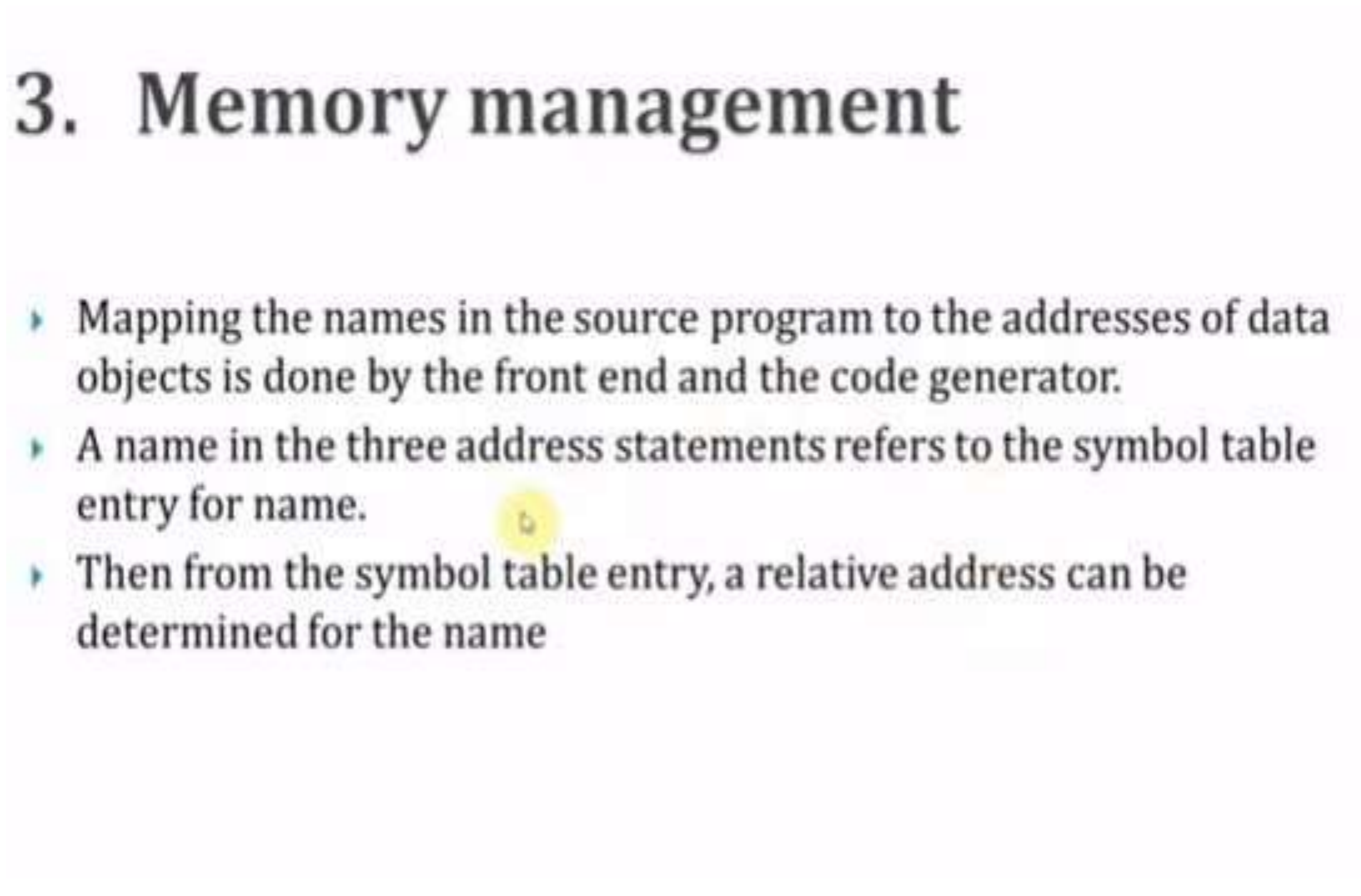
- ❖ The input to code generator is the intermediate code generated by the front end, along with information in the symbol table that determines the run-time addresses of the data-objects denoted by the names in the intermediate representation.
- ❖ Intermediate codes may be represented mostly in quadruples, triples, indirect triples, Postfix notation, syntax trees, DAG's etc.
- ❖ Assume that they are free from all of syntactic and state semantic errors, the necessary type checking has taken place and the type-conversion operators have been inserted wherever necessary.

4



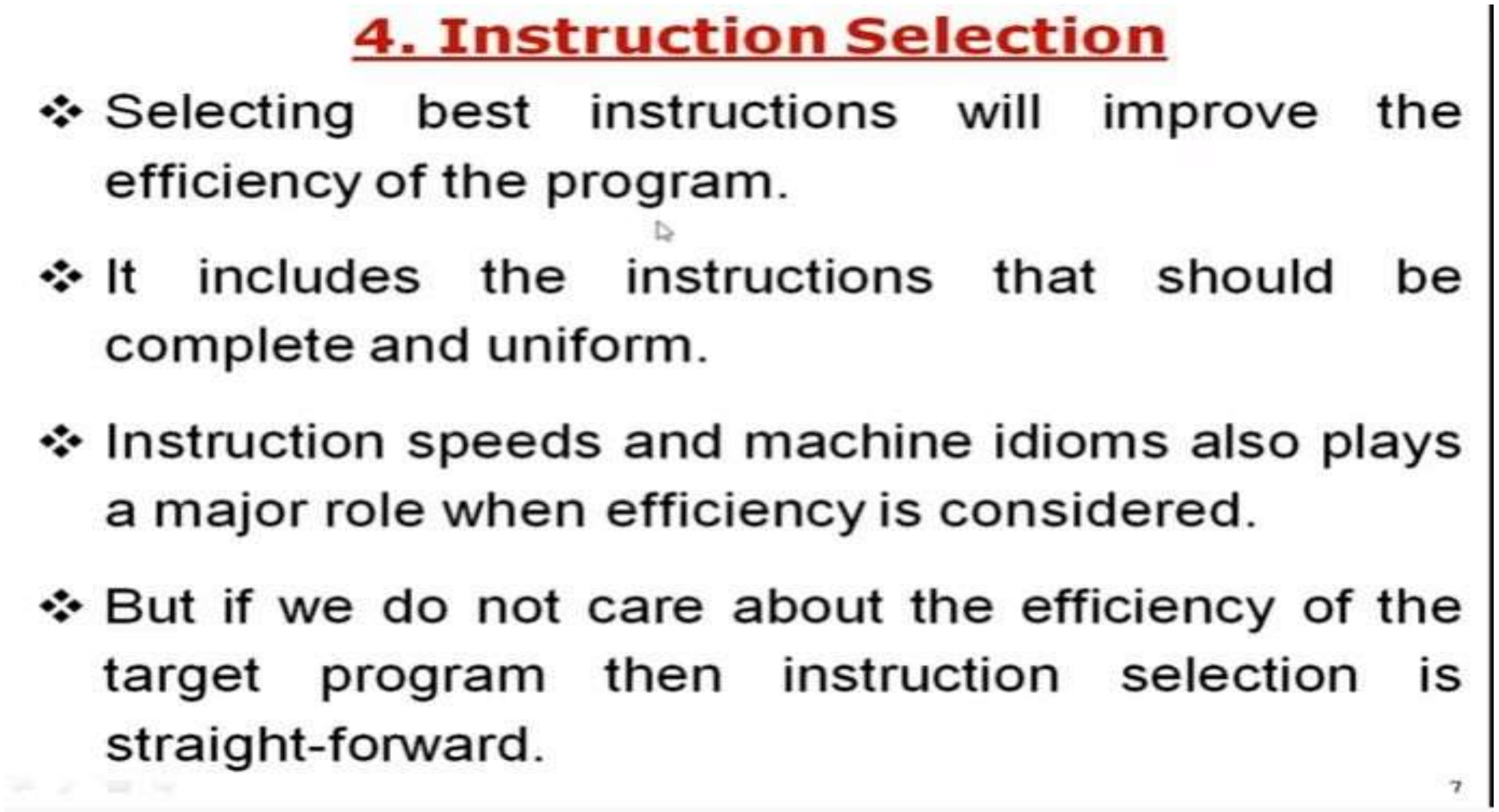
2. Target program

- The target program is the output of the code generator.
- The output may be absolute machine language, relocatable machine language, assembly language.
- Absolute machine language as output has advantages that it can be placed in a fixed memory location, so CPU can access it faster.
- Relocatable machine language as an output allows subprograms and subroutines to be compiled separately. Relocatable object modules can be linked together and loaded by linking loader. But there is added expense of linking and loading.
- Assembly language: for code generation assembly language is very important. it will generate the code easily.



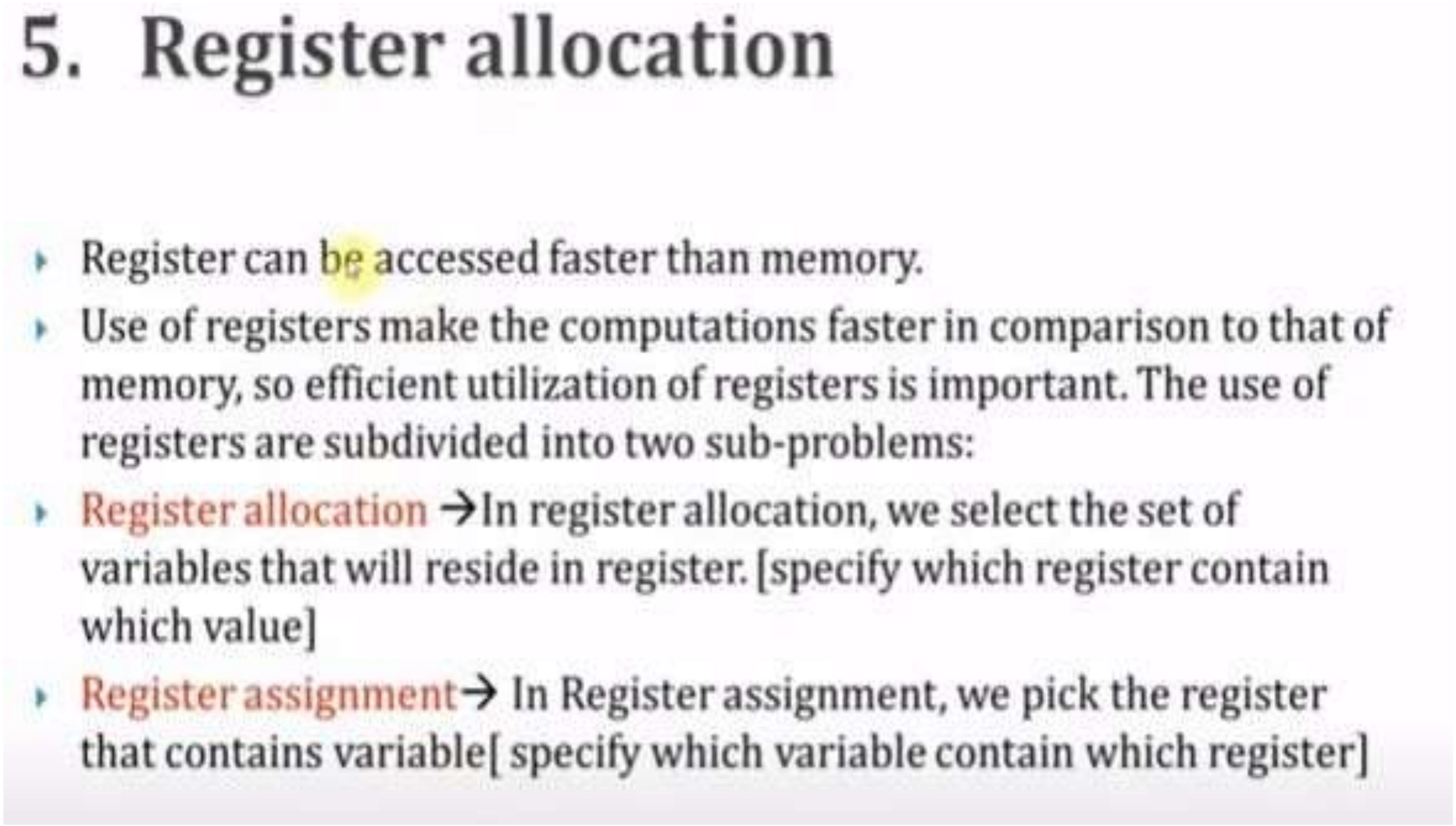
3. Memory management

- Mapping the names in the source program to the addresses of data objects is done by the front end and the code generator.
- A name in the three address statements refers to the symbol table entry for name.
- Then from the symbol table entry, a relative address can be determined for the name



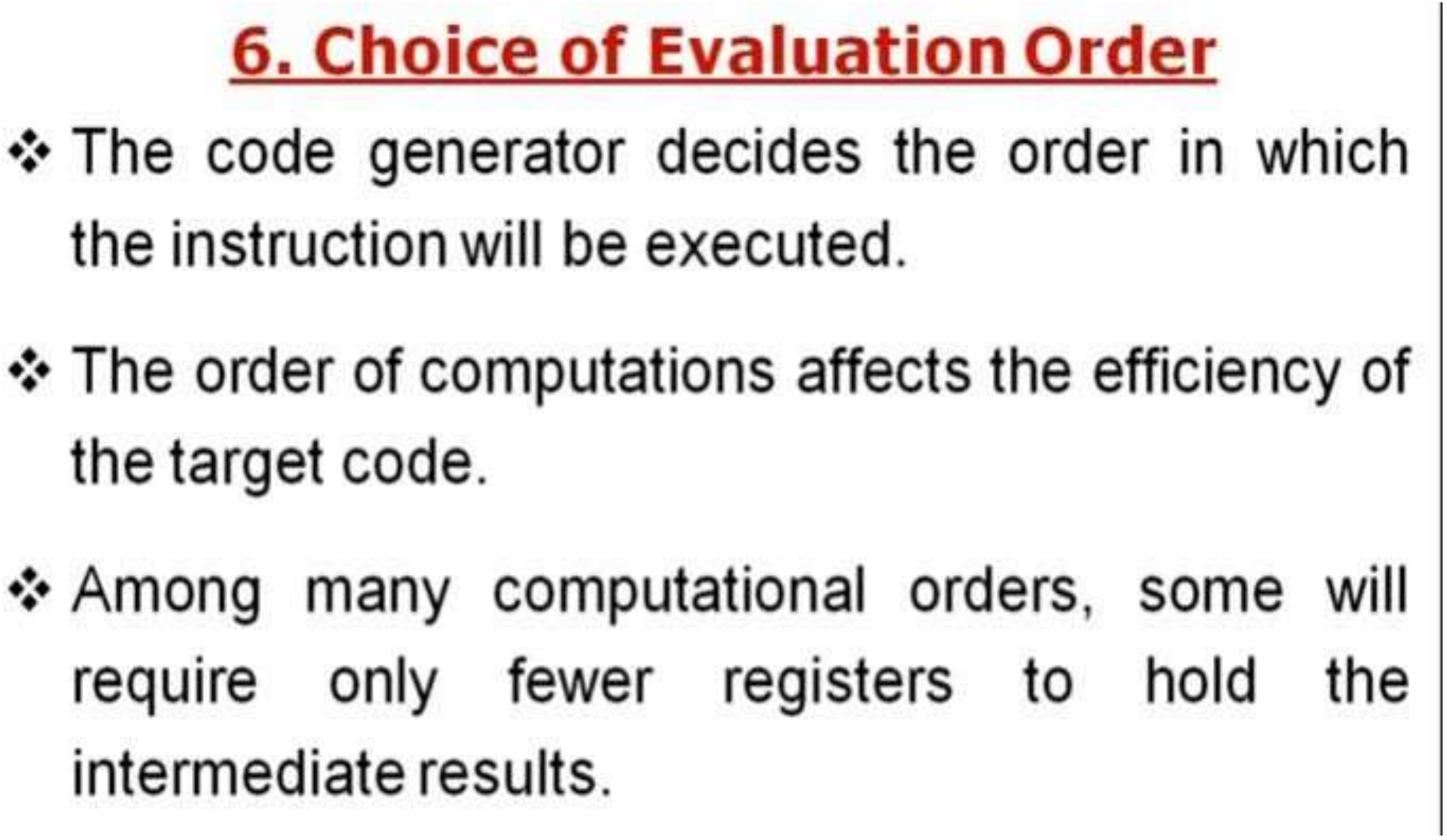
4. Instruction Selection

- Selecting best instructions will improve the efficiency of the program.
- It includes the instructions that should be complete and uniform.
- Instruction speeds and machine idioms also plays a major role when efficiency is considered.
- But if we do not care about the efficiency of the target program then instruction selection is straight-forward.



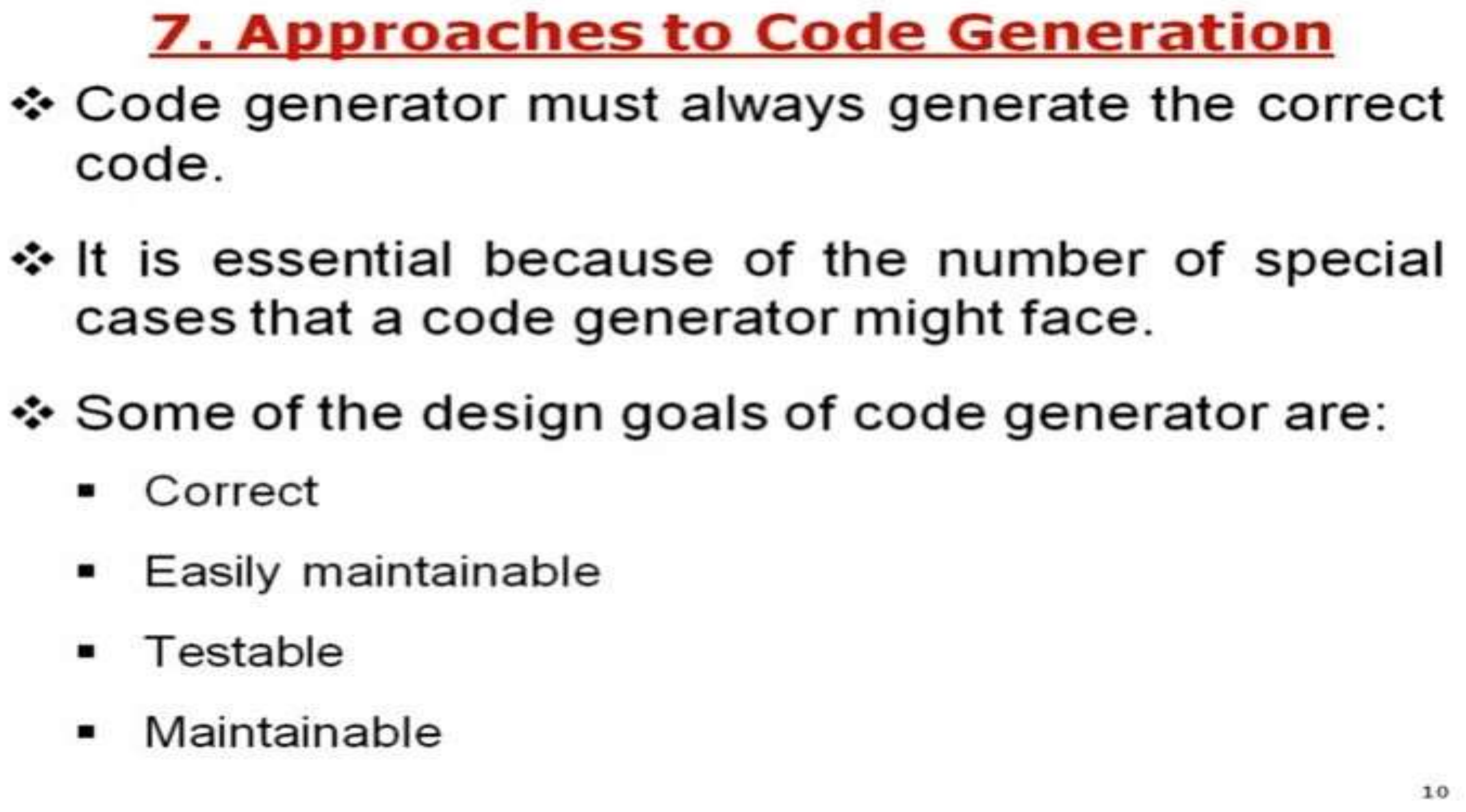
5. Register allocation

- Register can be accessed faster than memory.
- Use of registers make the computations faster in comparison to that of memory, so efficient utilization of registers is important. The use of registers are subdivided into two sub-problems:
- Register allocation →In register allocation, we select the set of variables that will reside in register. [specify which register contain which value]
- Register assignment→ In Register assignment, we pick the register that contains variable[specify which variable contain which register]



6. Choice of Evaluation Order

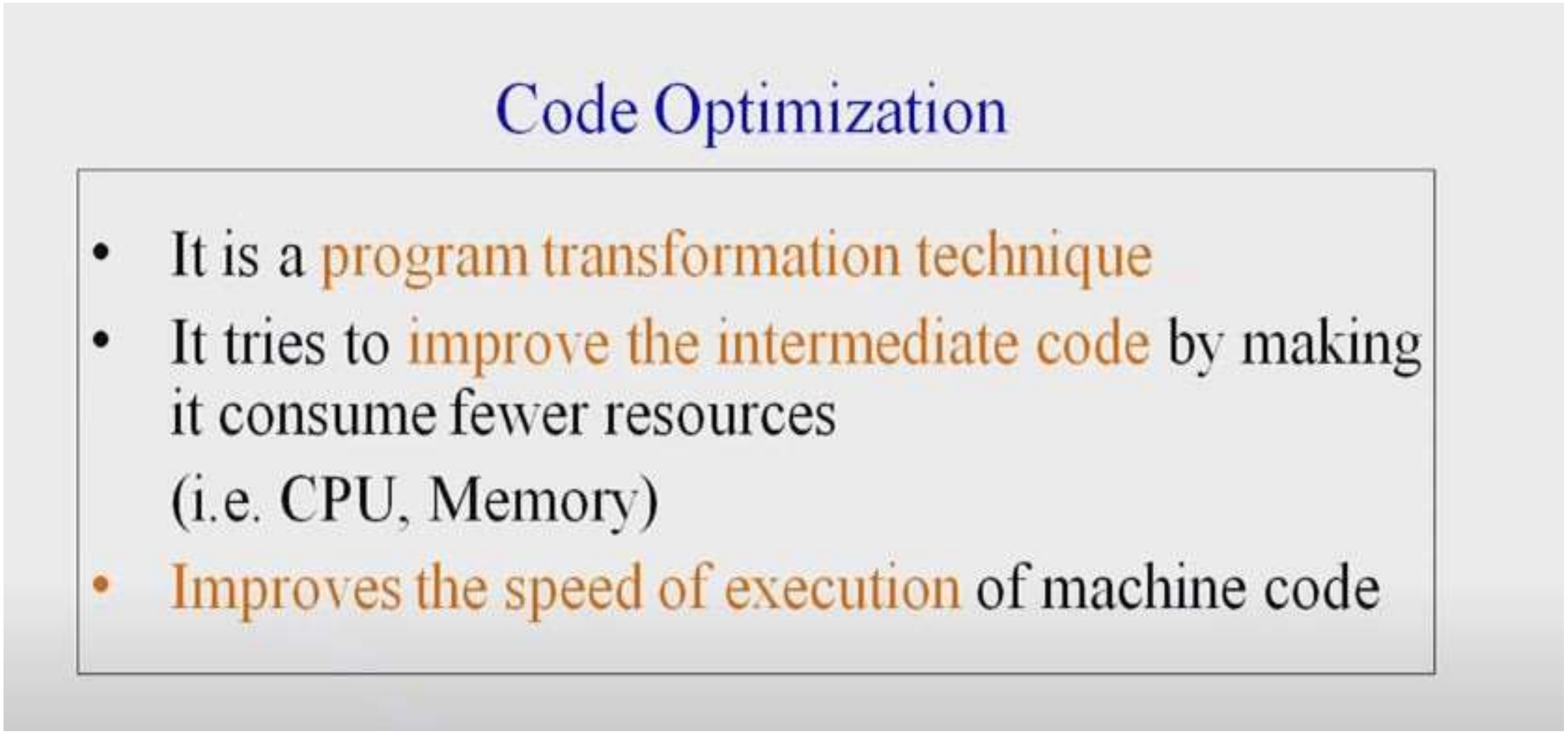
- ❖ The code generator decides the order in which the instruction will be executed.
- ❖ The order of computations affects the efficiency of the target code.
- ❖ Among many computational orders, some will require only fewer registers to hold the intermediate results.



7. Approaches to Code Generation

- Code generator must always generate the correct code.
- It is essential because of the number of special cases that a code generator might face.
- Some of the design goals of code generator are:
  - Correct
  - Easily maintainable
  - Testable
  - Maintainable

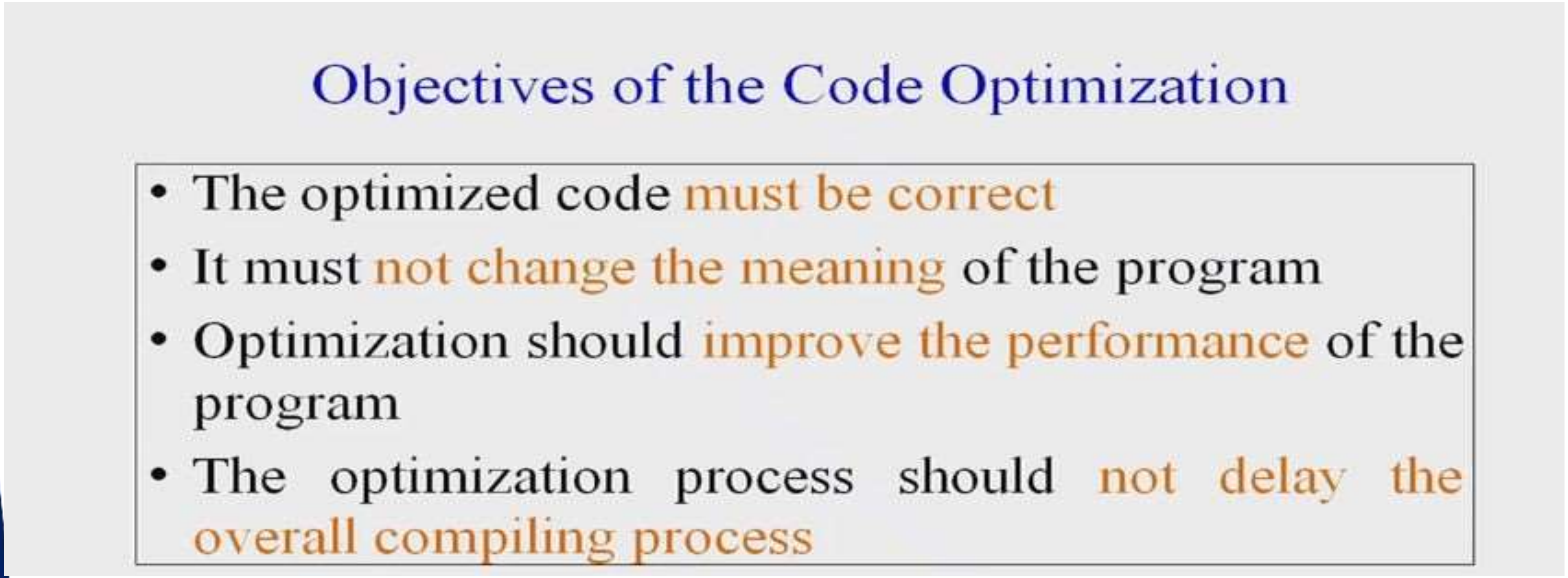
10



Code Optimization

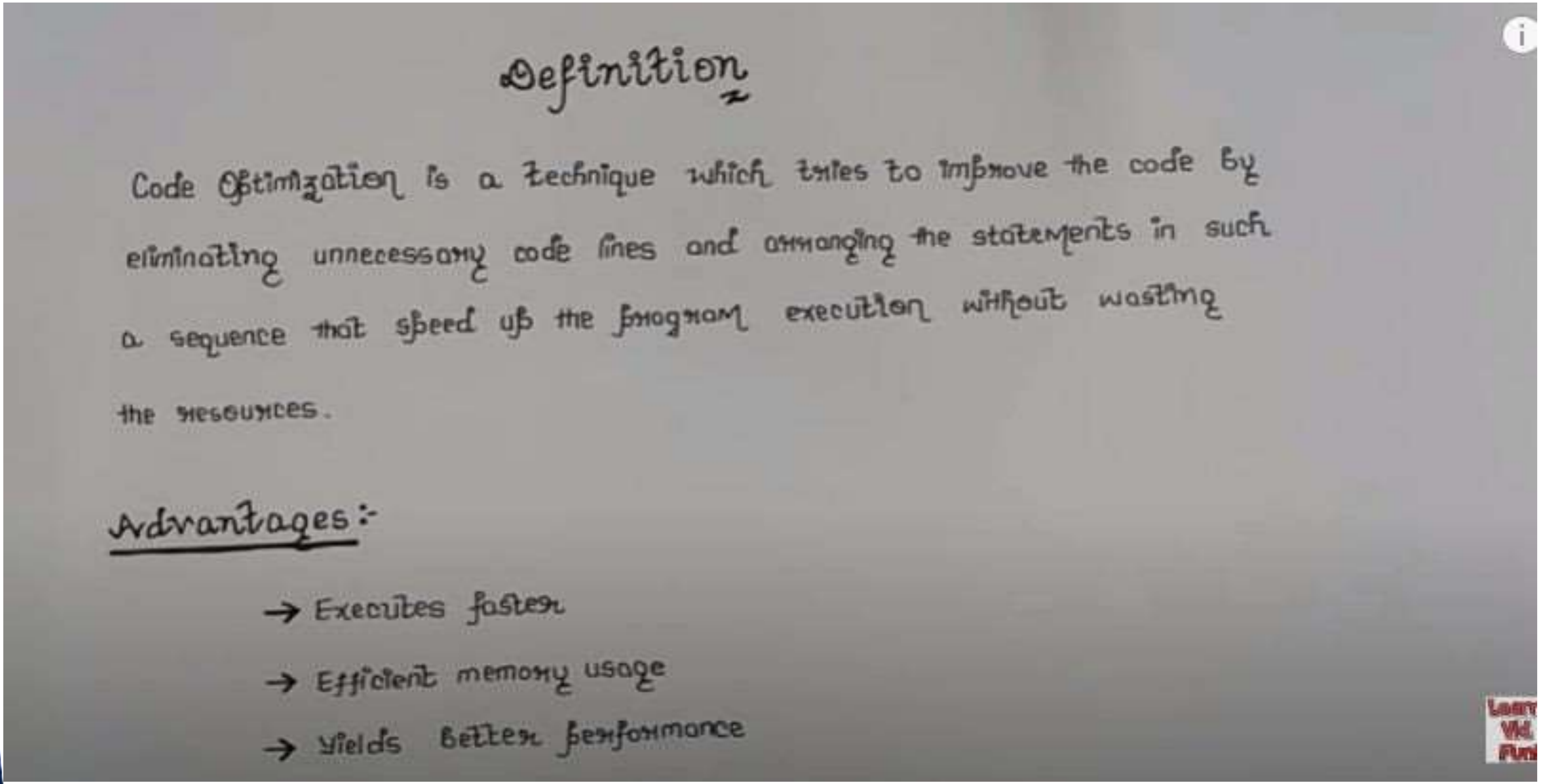
- It is a program transformation technique
- It tries to improve the intermediate code by making it consume fewer resources

  (i.e. CPU, Memory)
- Improves the speed of execution of machine code



Objectives of the Code Optimization

- The optimized code must be correct
- It must not change the meaning of the program
- Optimization should improve the performance of the program
- The optimization process should not delay the overall compiling process



Definition

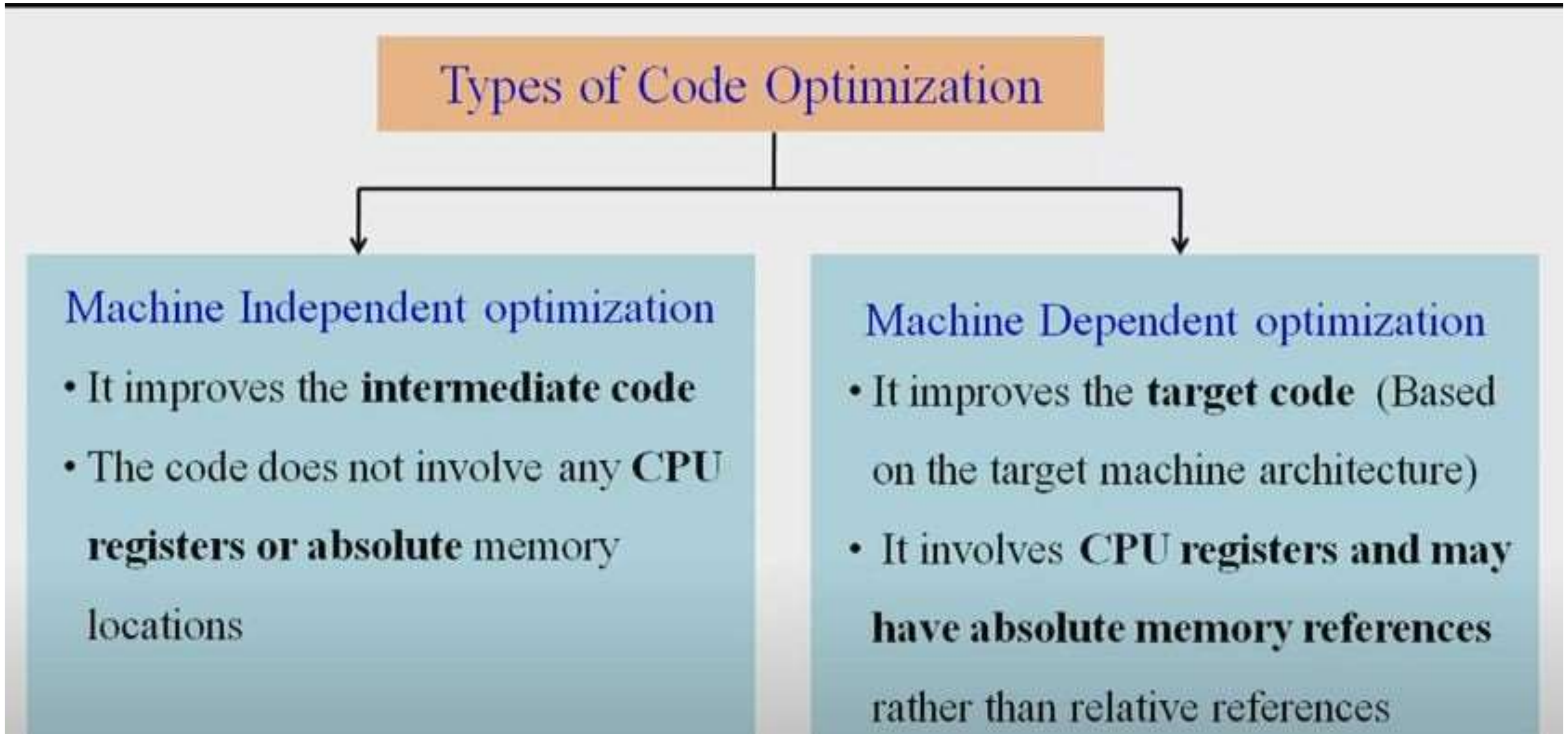
Code Optimization is a technique which tries to improve the code by eliminating unnecessary code lines and arranging the statements in such a sequence that speed up the program execution without wasting the resources.

Advantages:-

→ Executes faster

→ Efficient memory usage

→ Yields better performance



Types of Code Optimization

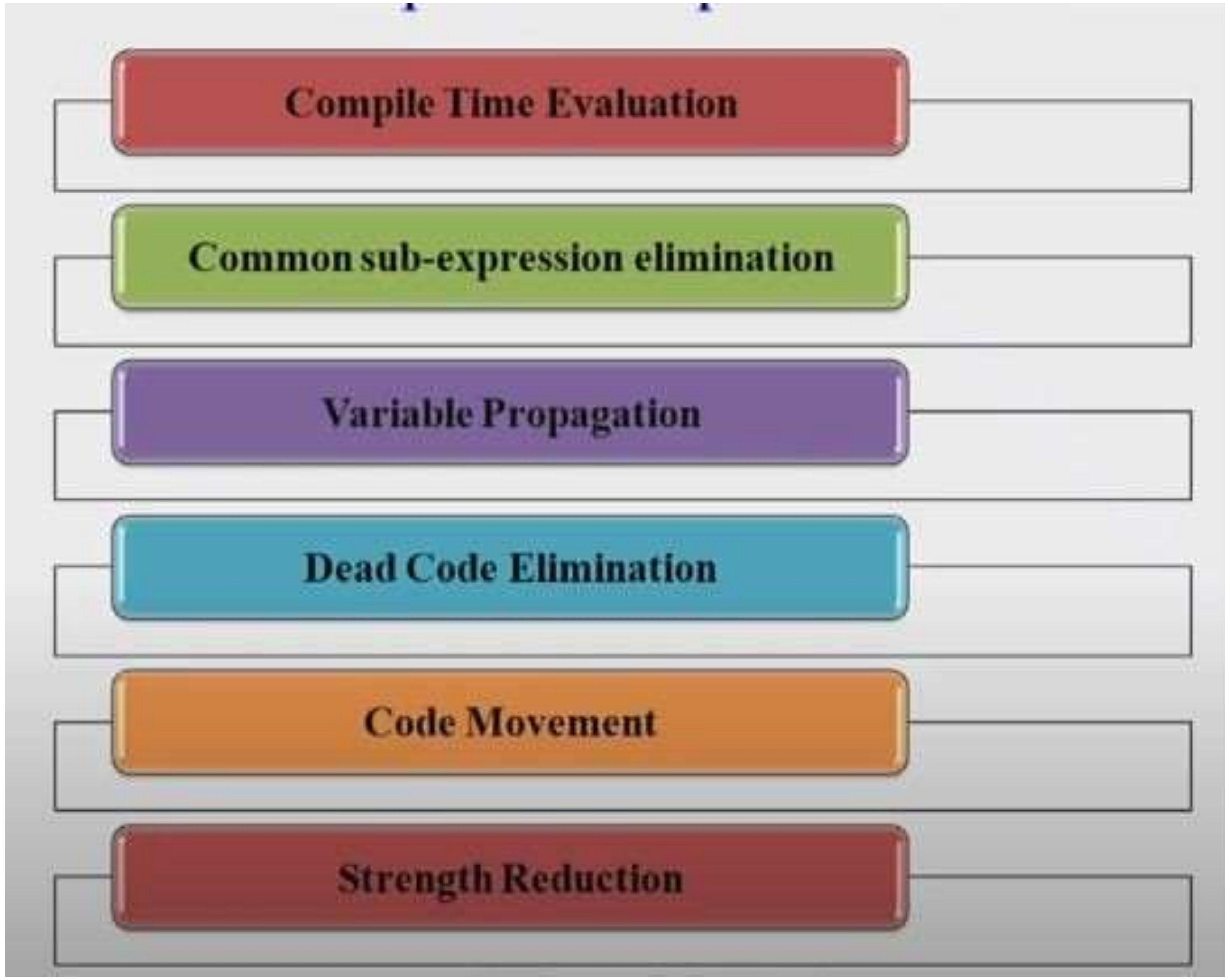
Machine Independent optimization

- It improves the **intermediate code**
- The code does not involve any **CPU registers or absolute** memory locations

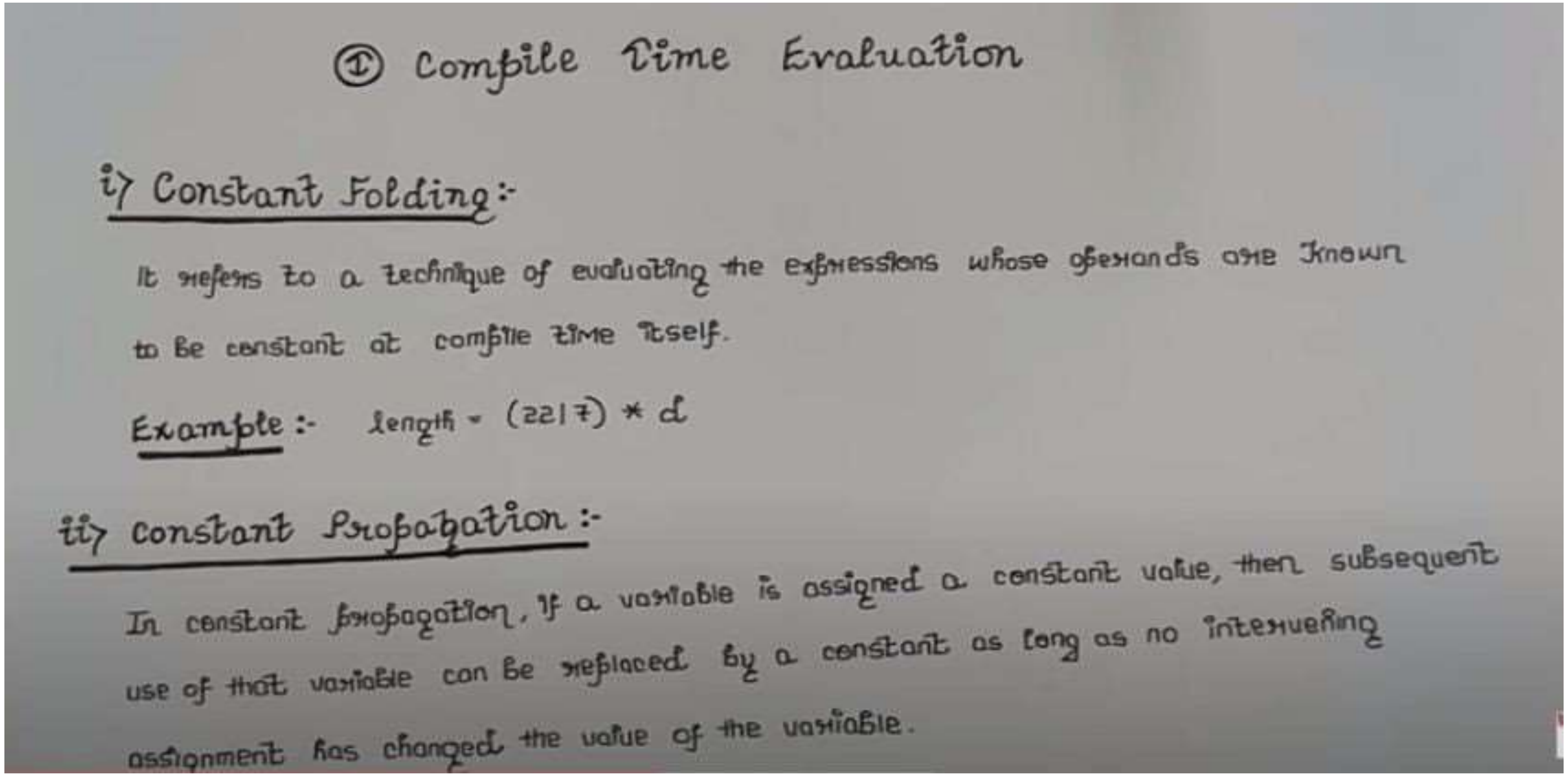
Machine Dependent optimization

- It improves the **target code** (Based on the target machine architecture)
- It involves **CPU registers and may have absolute memory references** rather than relative references

TECHNIQUES OF MACHINE DEPENDENT OPTIMIZATION



- Compile Time Evaluation
- Common sub-expression elimination
- Variable Propagation
- Dead Code Elimination
- Code Movement
- Strength Reduction



① Compile Time Evaluation

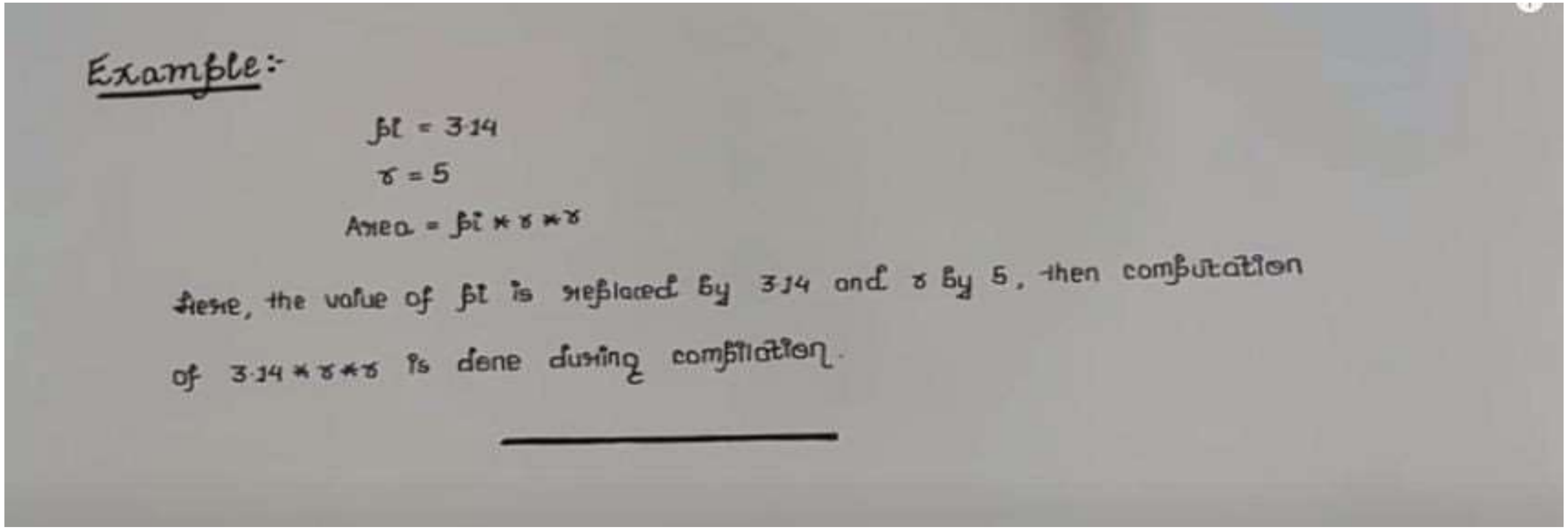
i> Constant Folding :-

It refers to a technique of evaluating the expressions whose operands are known to be constant at compile time itself.

Example :- length = (22/7) * d

ii> Constant Propagation :-

In constant propagation, if a variable is assigned a constant value, then subsequent use of that variable can be replaced by a constant as long as no intervening assignment has changed the value of the variable.



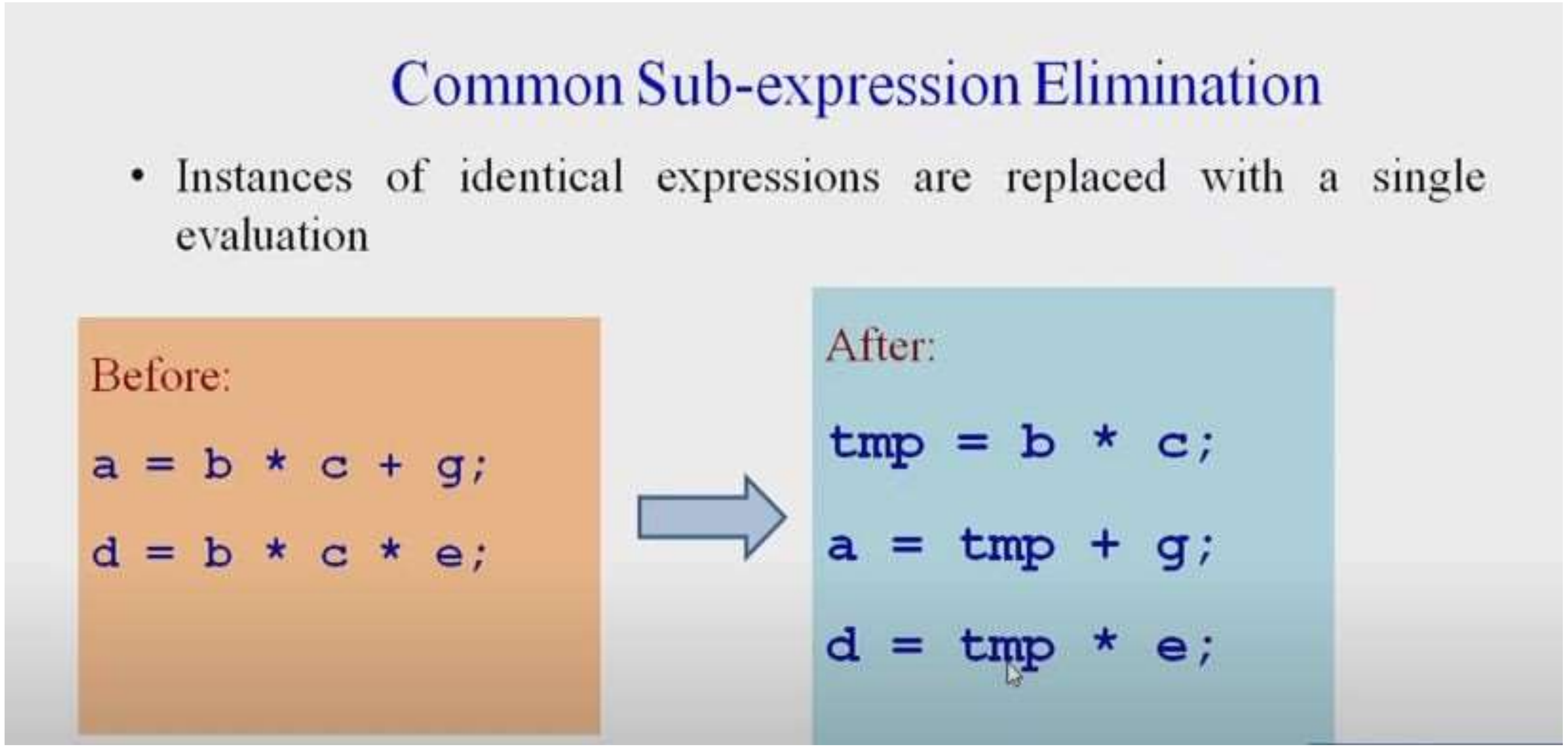
Example:-

pi = 3.14

r = 5

Area = pi * r * r

Here, the value of pi is replaced by 3.14 and r by 5, then computation of 3.14 * r * r is done during compilation.



Common Sub-expression Elimination

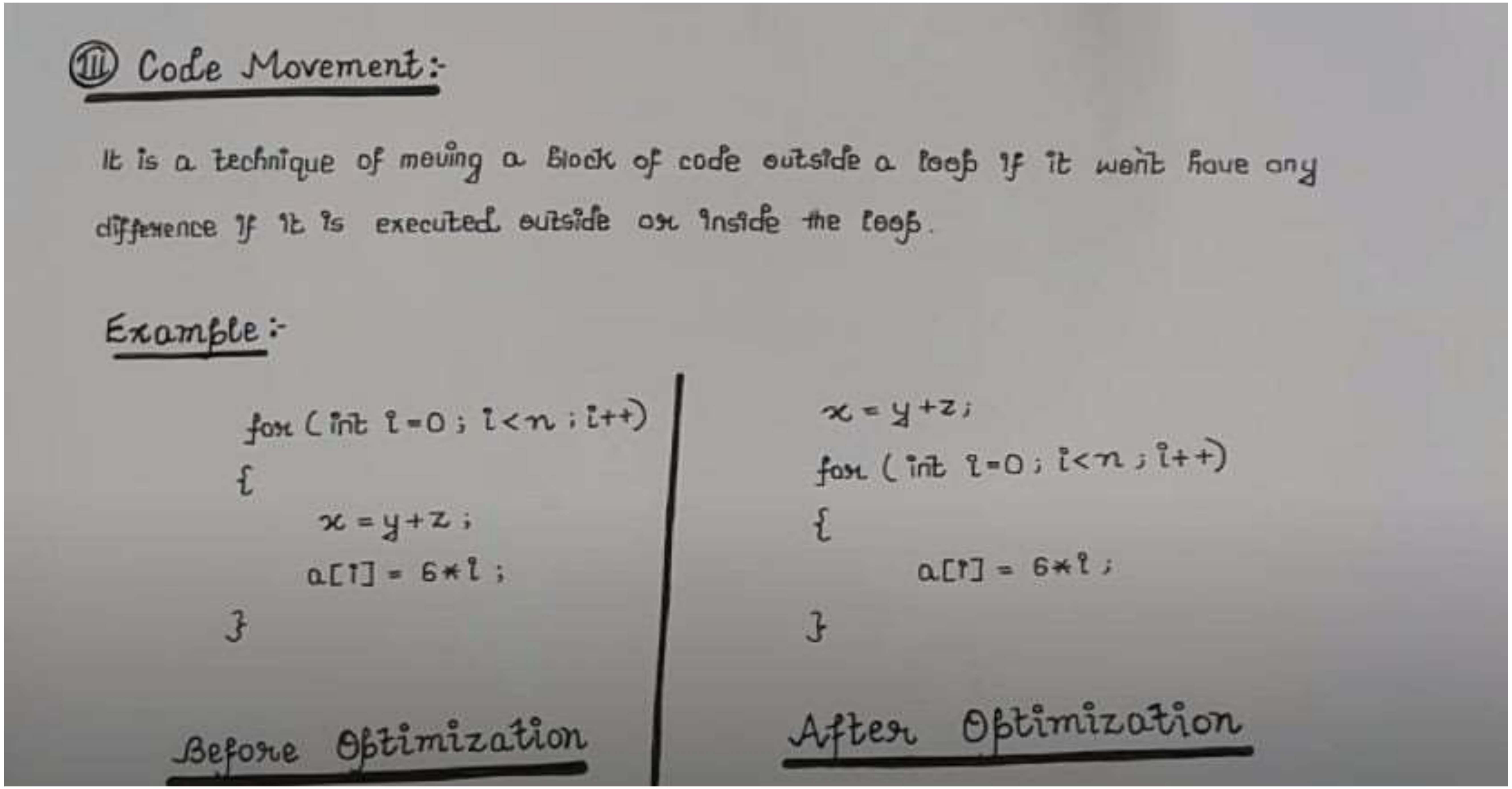
- Instances of identical expressions are replaced with a single evaluation

Before:

```
a = b * c + g;
d = b * c * e;
```

After:

```
tmp = b * c;
a = tmp + g;
d = tmp * e;
```



(11) Code Movement :-

It is a technique of moving a Block of code outside a loop if it won't have any difference if it is executed outside or inside the loop.

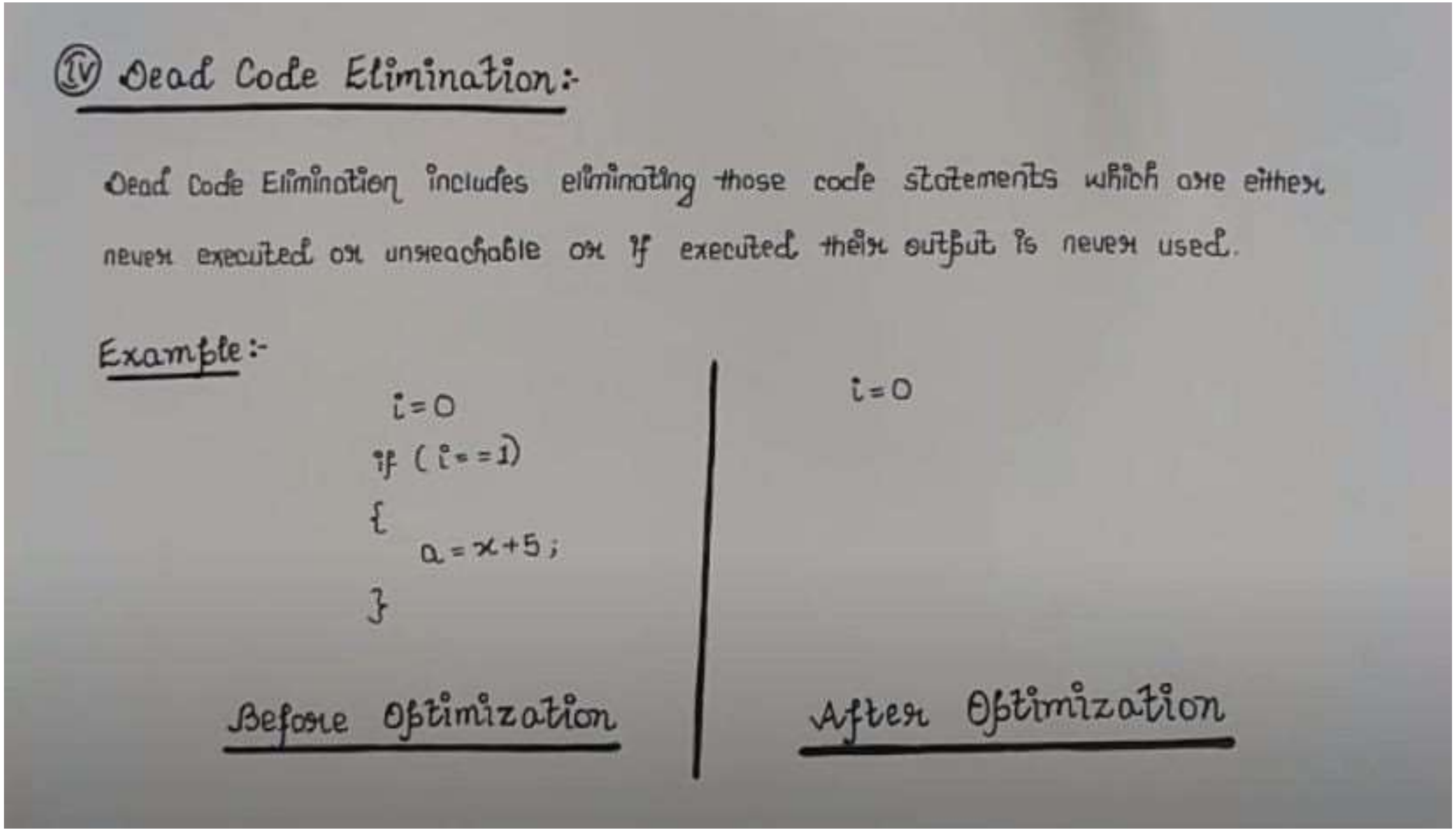
Example :-

```
for ( int i=0 ; i<n ; i++)
{
      x = y+z ;
      a[i] = 6*i ;
}
```

Before Optimization

```
x = y+z ;
for ( int i=0 ; i<n ; i++)
{
      a[i] = 6*i ;
}
```

After Optimization



(IV) Dead Code Elimination:-

Dead Code Elimination includes eliminating those code statements which are either never executed or unreachable or if executed their output is never used.

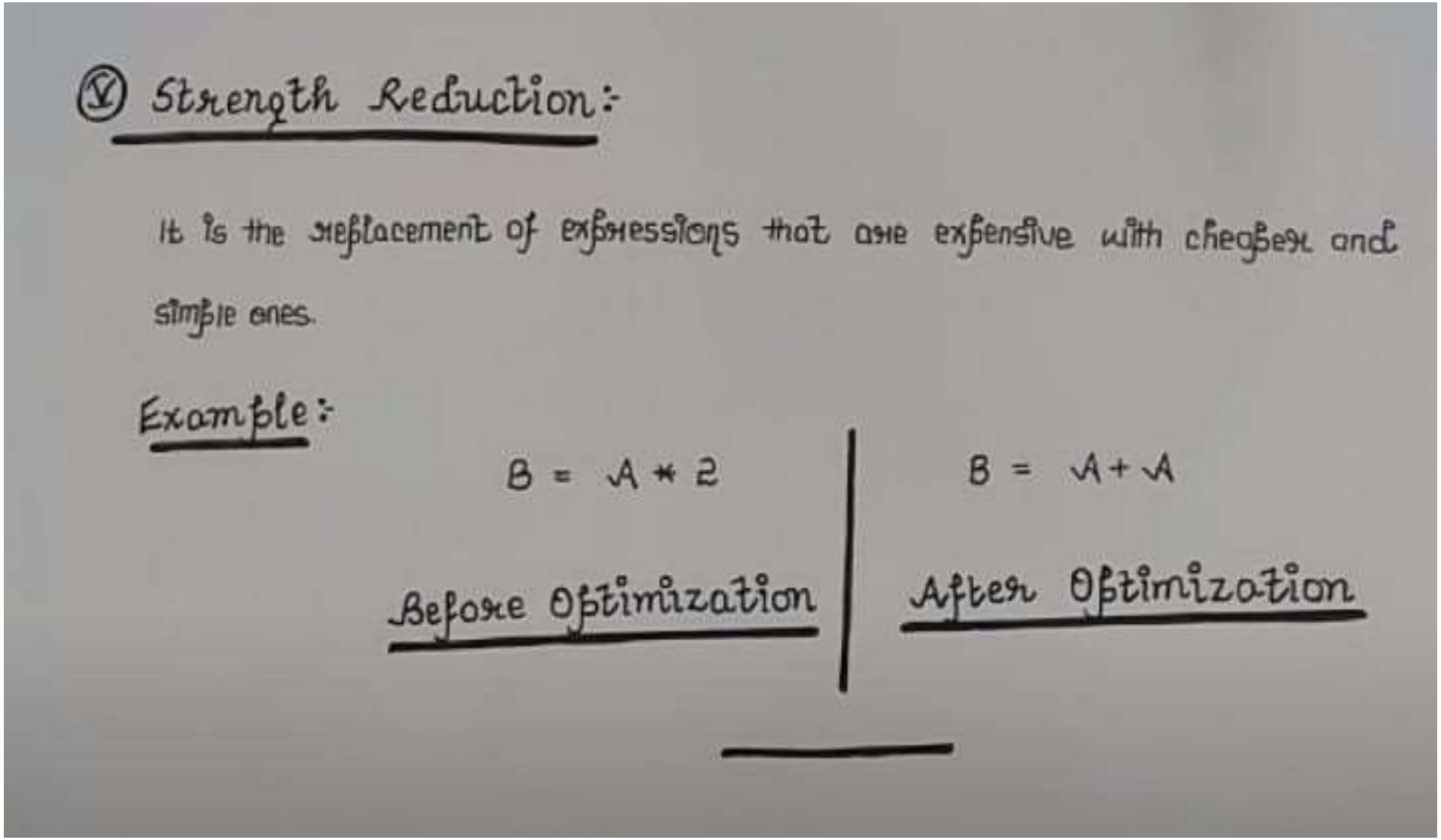
Example:-

```
i=0
if (i==1)
{
    a=x+5;
}
```

Before Optimization

```
i=0
```

After Optimization

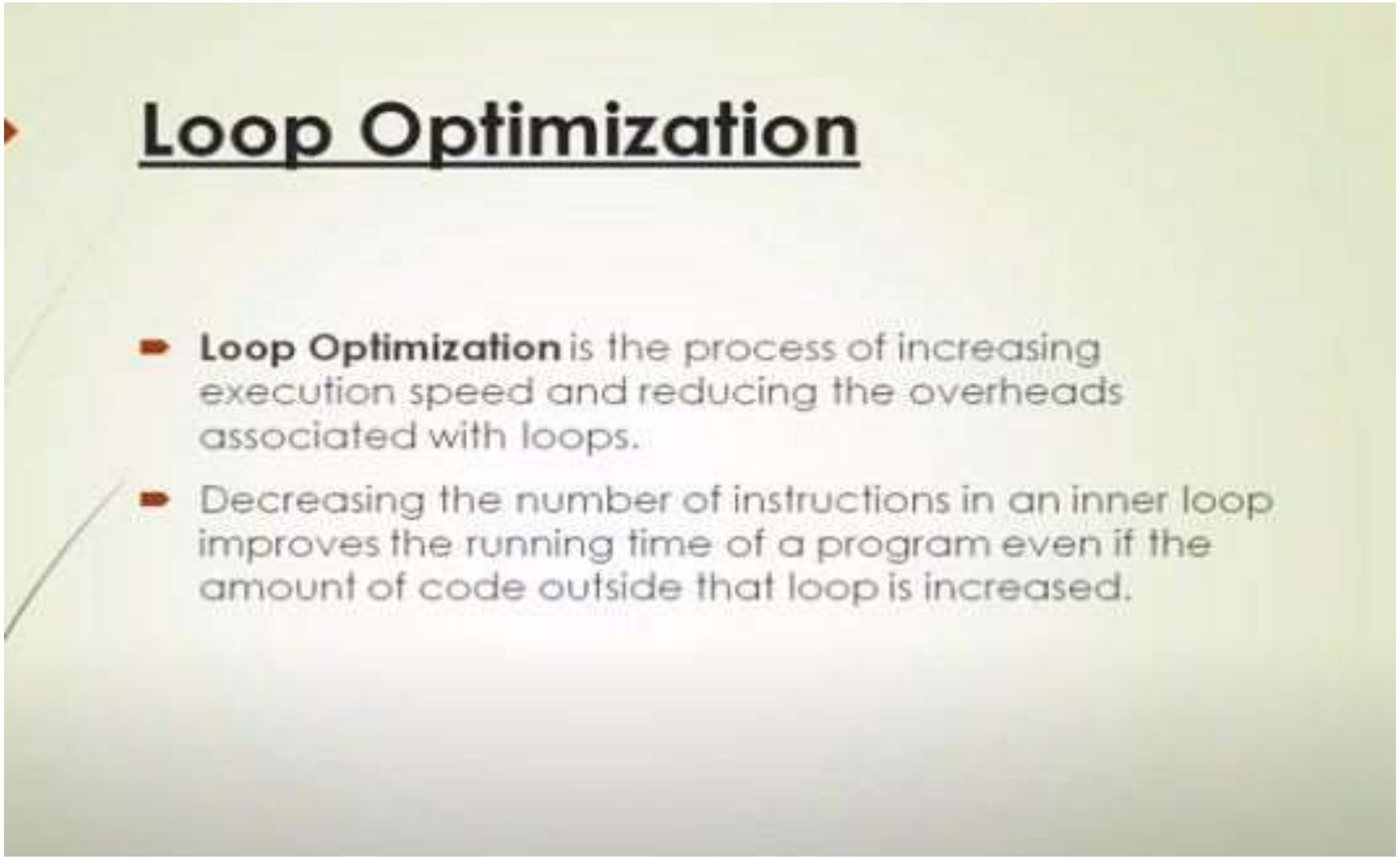


(V) Strength Reduction:-

It is the replacement of expressions that are expensive with cheaper and simple ones.

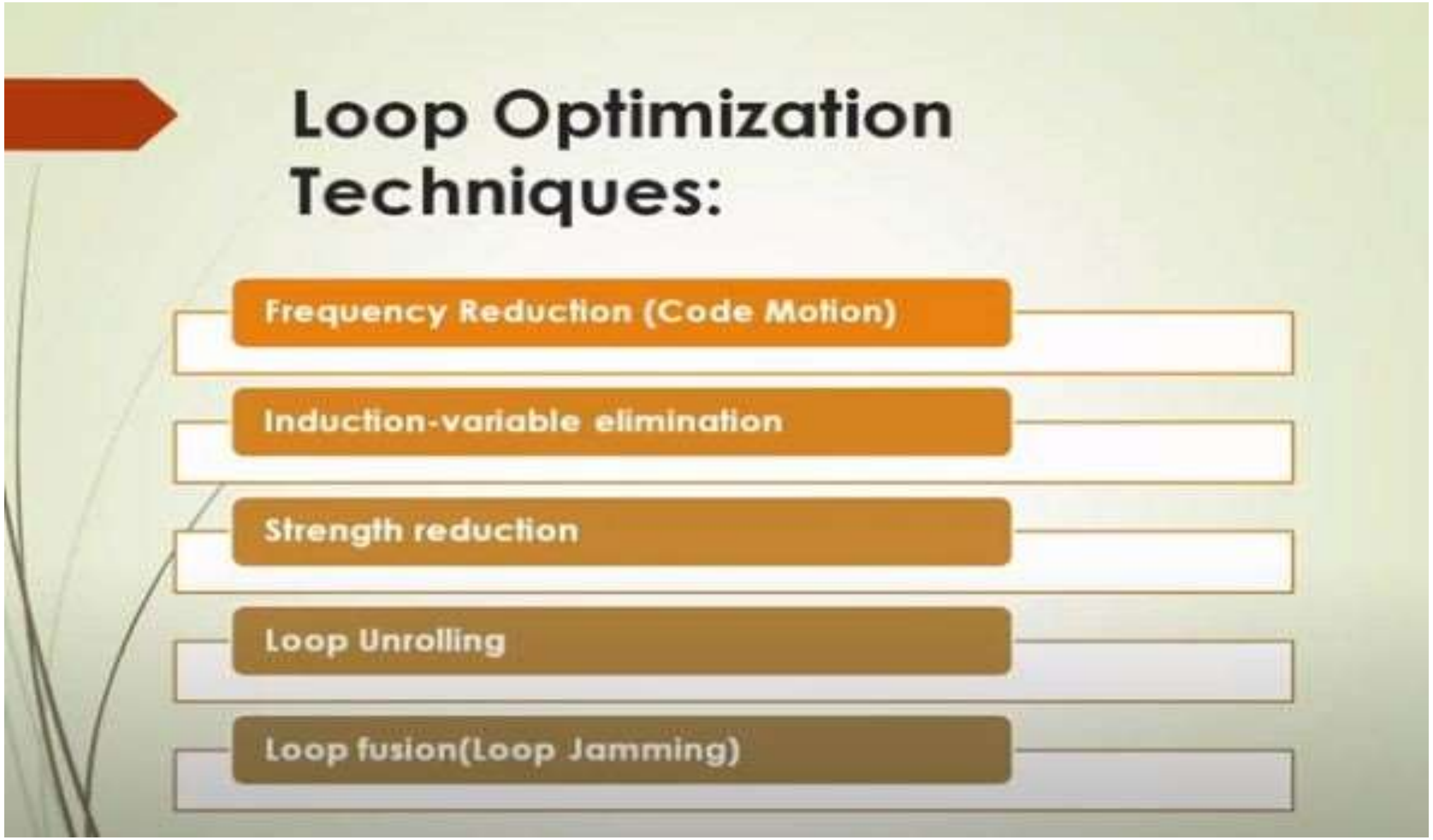
Example:-

Before Optimization	After Optimization
B = A * 2	B = A + A



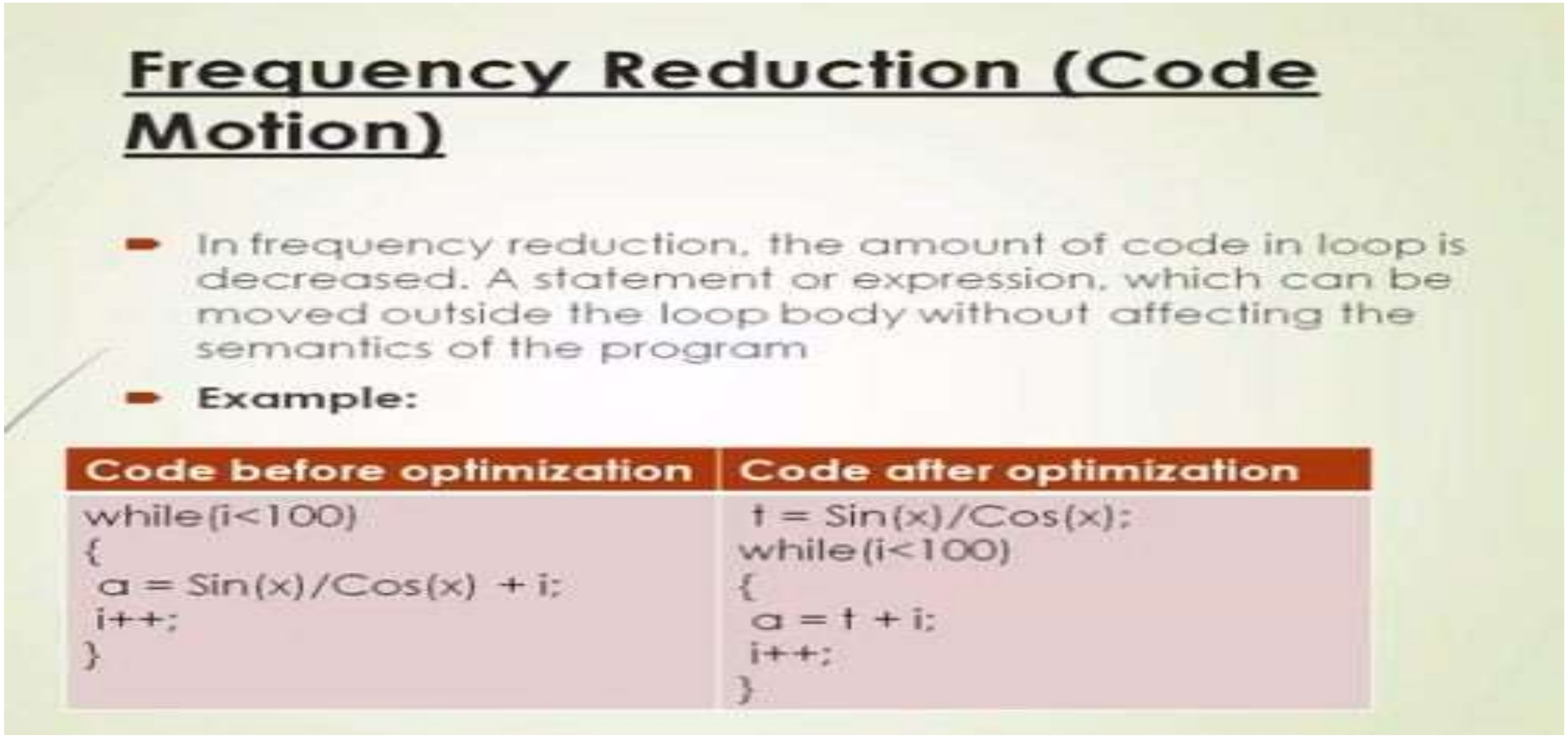
Loop Optimization

- **Loop Optimization** is the process of increasing execution speed and reducing the overheads associated with loops.
- Decreasing the number of instructions in an inner loop improves the running time of a program even if the amount of code outside that loop is increased.



Loop Optimization Techniques:

- Frequency Reduction (Code Motion)
- Induction-variable elimination
- Strength reduction
- Loop Unrolling
- Loop fusion(Loop Jamming)

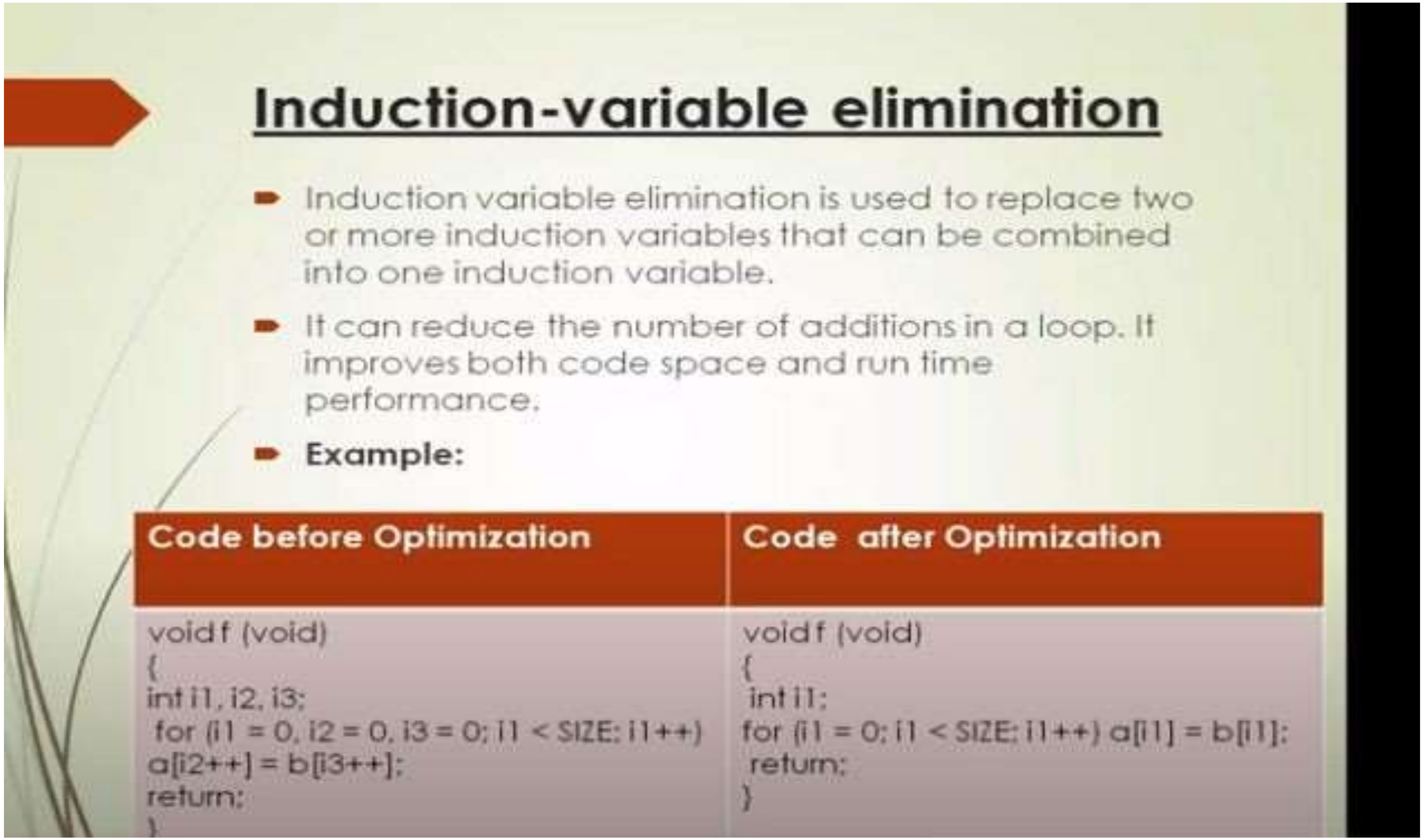


Frequency Reduction (Code Motion)

- In frequency reduction, the amount of code in loop is decreased. A statement or expression, which can be moved outside the loop body without affecting the semantics of the program
- **Example:**

Code before optimization	Code after optimization

while(i<100)
{
 a = Sin(x)/Cos(x) + i;
 i++;
}
	t = Sin(x)/Cos(x);
	while(i<100)
	{
	 a = t + i;
	 i++;
	}

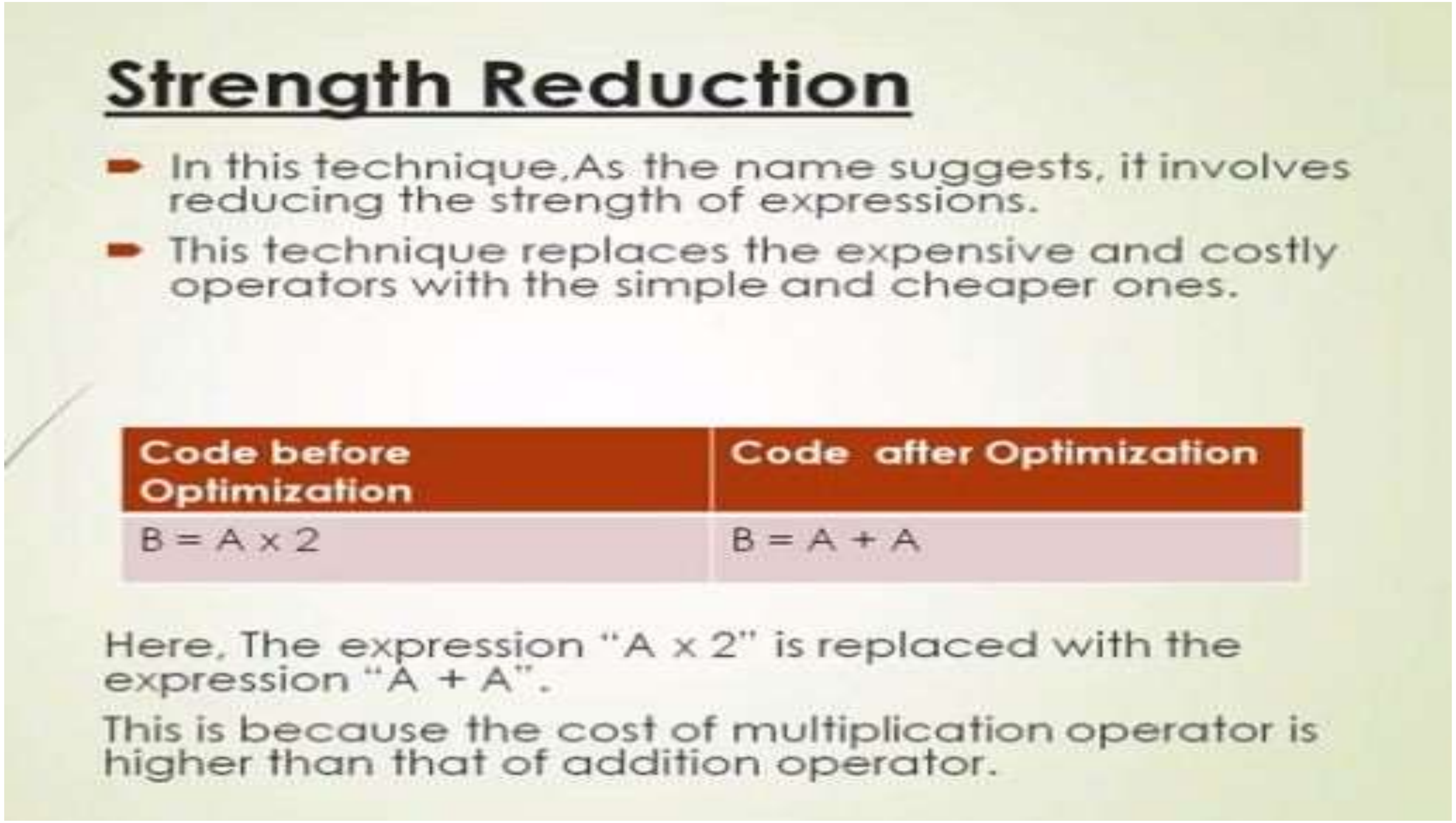


Induction-variable elimination

- Induction variable elimination is used to replace two or more induction variables that can be combined into one induction variable.
- It can reduce the number of additions in a loop. It improves both code space and run time performance.
- **Example:**

Code before Optimization	Code after Optimization

void f (void)
{
int i1, i2, i3;
for (i1 = 0, i2 = 0, i3 = 0; i1 < SIZE; i1++)
a[i2++] = b[i3++];
return;
}
	void f (void)
	{
	int i1;
	for (i1 = 0; i1 < SIZE; i1++) a[i1] = b[i1];
	return;
	}



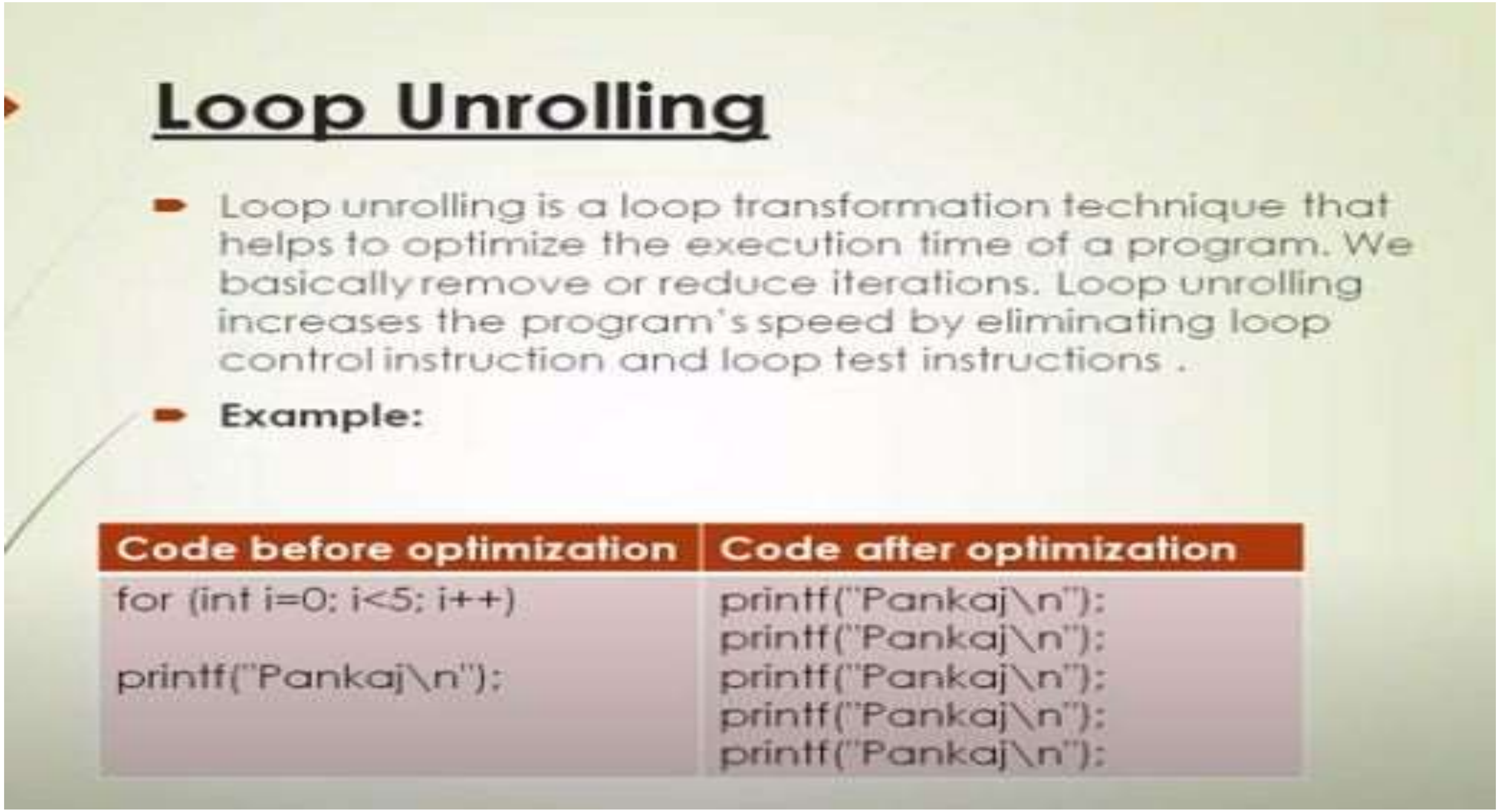
Strength Reduction

- In this technique,As the name suggests, it involves reducing the strength of expressions.
- This technique replaces the expensive and costly operators with the simple and cheaper ones.

Code before Optimization	Code after Optimization
B = A x 2	B = A + A

Here, The expression "A x 2" is replaced with the expression "A + A".

This is because the cost of multiplication operator is higher than that of addition operator.

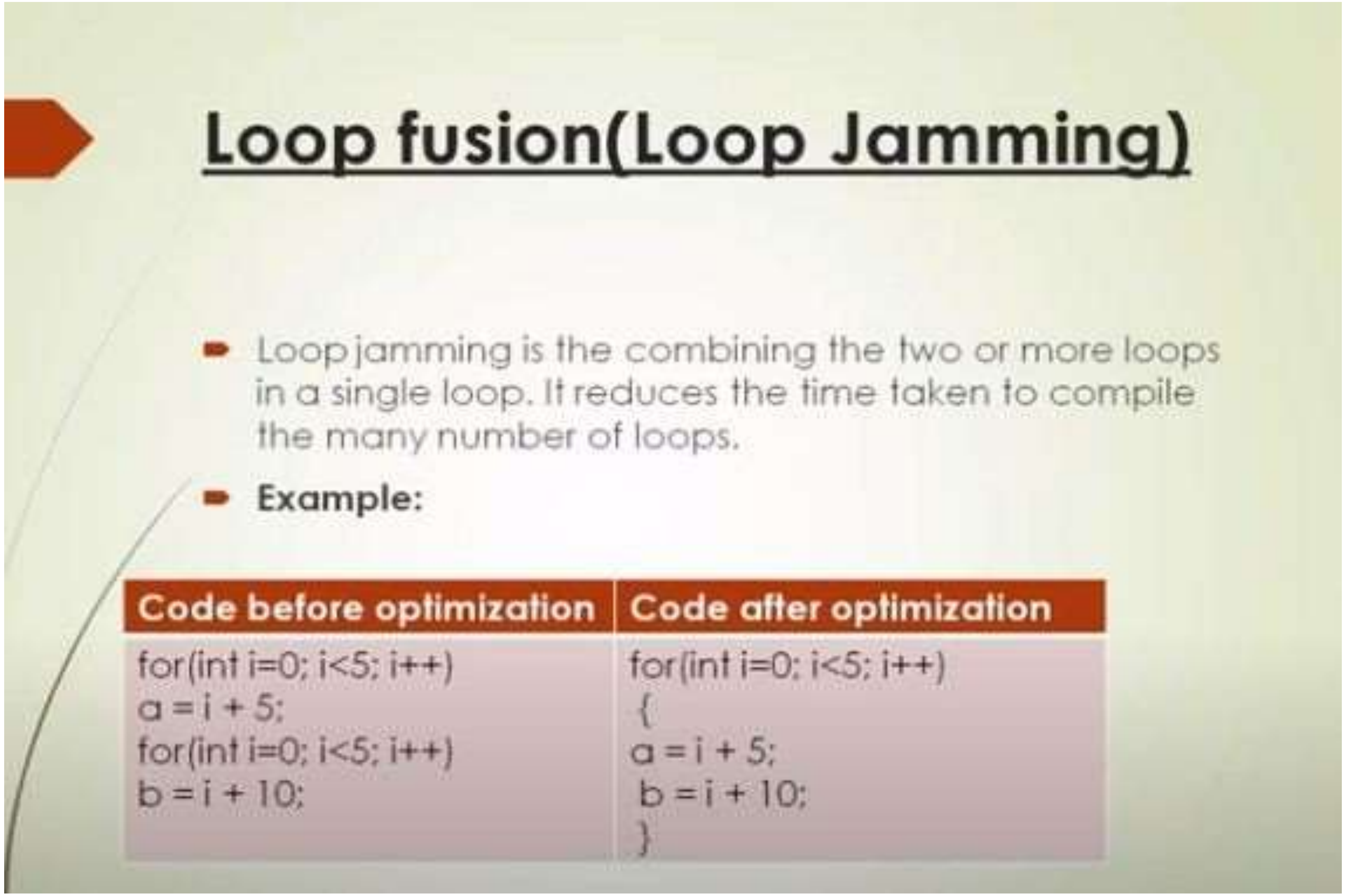


Loop Unrolling

- Loop unrolling is a loop transformation technique that helps to optimize the execution time of a program. We basically remove or reduce iterations. Loop unrolling increases the program's speed by eliminating loop control instruction and loop test instructions .
- **Example:**

Code before optimization	Code after optimization

for (int i=0; i<5; i++)
 	
printf("Pankaj\n");
	printf("Pankaj\n");
	printf("Pankaj\n");
	printf("Pankaj\n");
	printf("Pankaj\n");
	printf("Pankaj\n");



Loop fusion(Loop Jamming)

- Loop jamming is the combining the two or more loops in a single loop. It reduces the time taken to compile the many number of loops.
- **Example:**

Code before optimization	Code after optimization

for(int i=0; i<5; i++)
a = i + 5;
for(int i=0; i<5; i++)
b = i + 10;
	for(int i=0; i<5; i++)
	{
	a = i + 5;
	b = i + 10;
	}

JECRC Foundation

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE

Thank you!