Programming: What? Why? How?

Panchatcharam M

PROGRAMMING: WHAT?

PROGRAMMING: WHAT?

A way to instruct the computer to perform various tasks

✓ Examples:

- Addition of two numbers
- Simple Interest
- Probability
- Simulation
- Microwave Oven
- Washing Machine

PROGRAMMING: WHAT?

- ✍ Programming is the process of designing and creating instructions (code) that a computer can execute to perform specific tasks or solve problems.

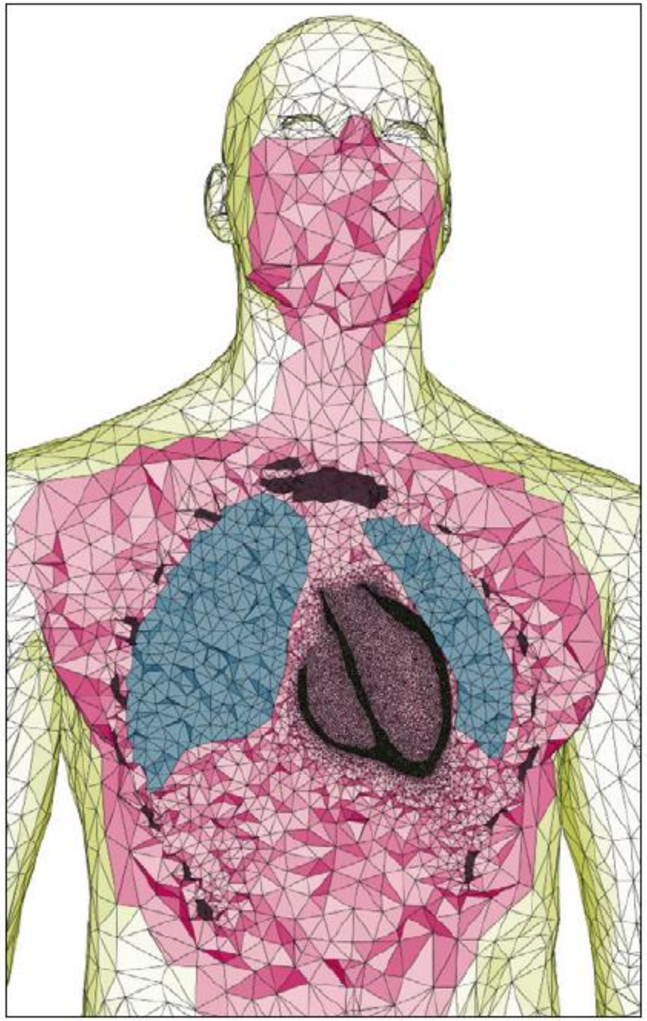
CORE CONCEPTS OF PROGRAMMING

Data Type	Numbers, Text, etc
Control Flow	If – else, for, while loops
Functions and Modularity	
Algorithms and Logic	The flow and structure of the instructions.
Languages	C, C++, Python, JAVA,etc

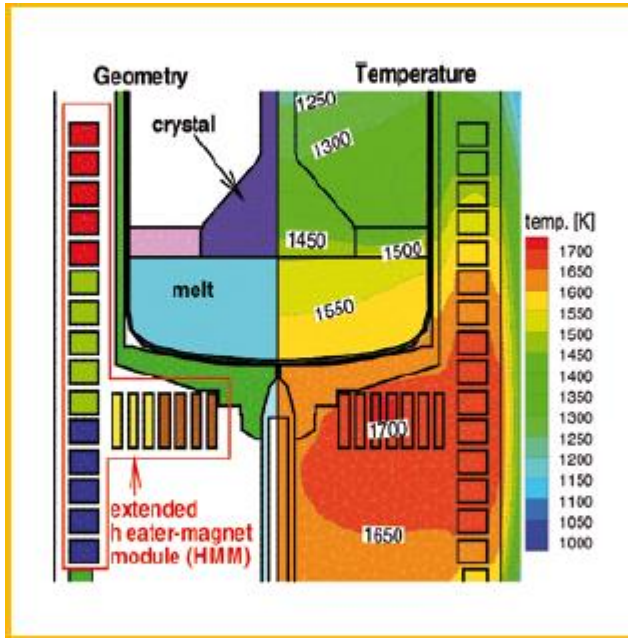
Programming: Where?

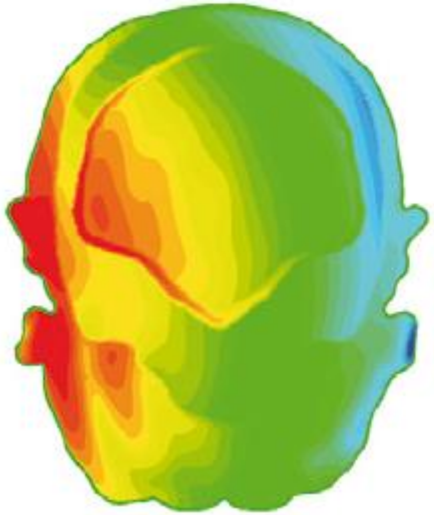
INTERDISCIPLINARY RESEARCH

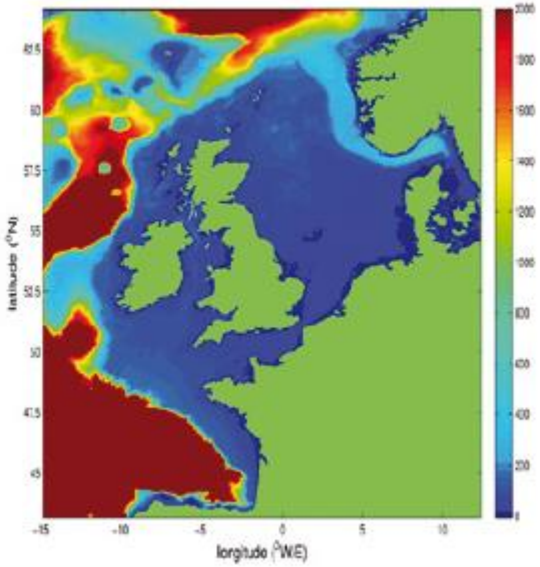
- All Engineering Field
- Image Processing
- Electro Chemistry
- Physics
- Fluid Mechanics
- Atmospheric Science
- Plant Physiology
- Human Physiology
- Medical
- Financial
- .....



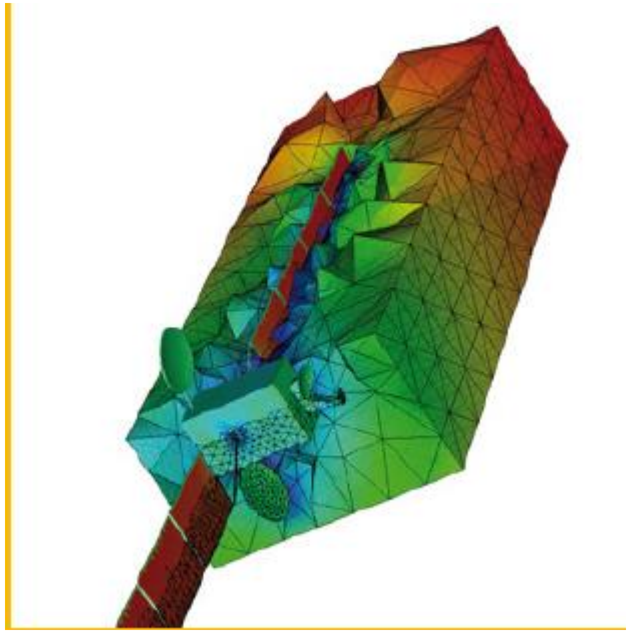
EXAMPLES

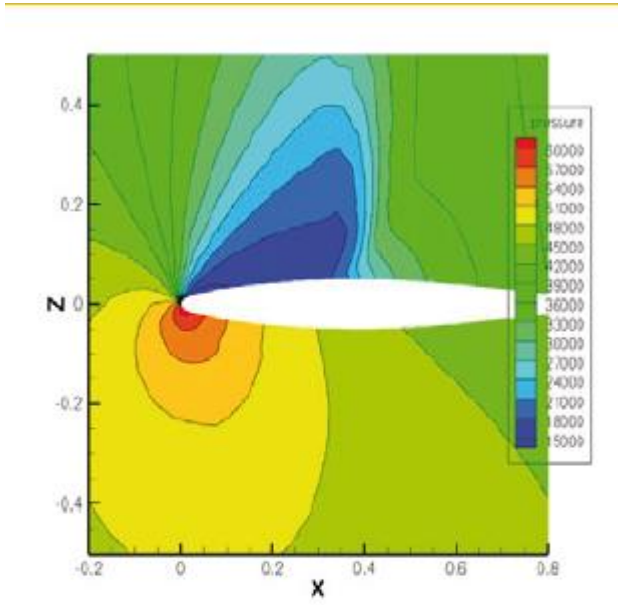

Geometry
Temperature
crystal
melt
1250
1300
1450
1500
1550
1700
1650
extended
h eater-magnet
module (HMM)
temp. [K]
1700
1650
1600
1550
1500
1450
1400
1350
1300
1250
1200
1150
1100
1050
1000



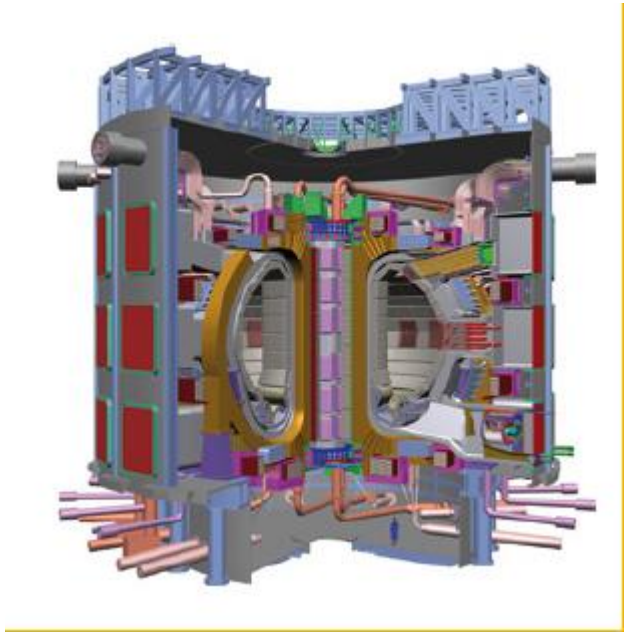

62.5
60
57.5
55
52.5
50
47.5
45
latitude (°N)
-15
-10
-5
0
5
10
longitude (°W/E)
2000
1800
1600
1400
1200
1000
800
600
400
200
0

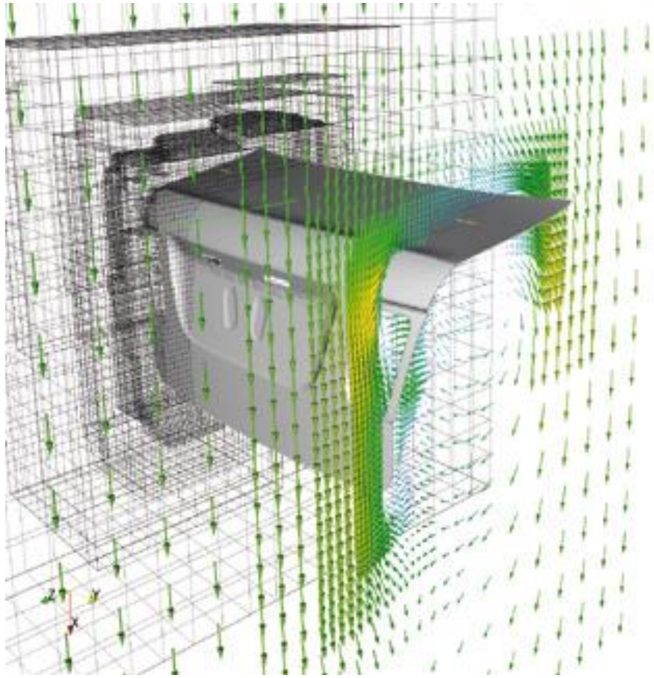
EXAMPLES



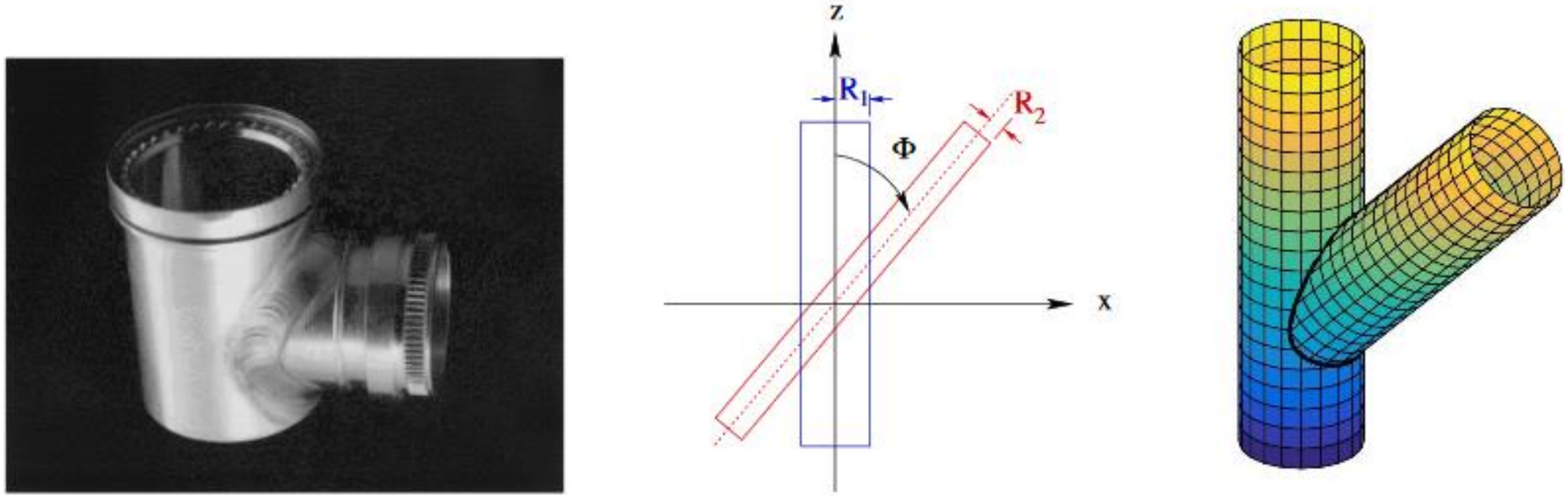

pressure
60000
57000
54000
51000
48000
45000
42000
39000
36000
33000
30000
27000
24000
21000
18000
15000
0.4
0.2
Z 0
-0.2
-0.4
-0.2
0
0.2
0.4
0.6
0.8
X

EXAMPLES





Examples: Robotic Pipe Welding



(a) (b) (c)

Autonomous Things Example:

Drone examines a large field, ready to harvest

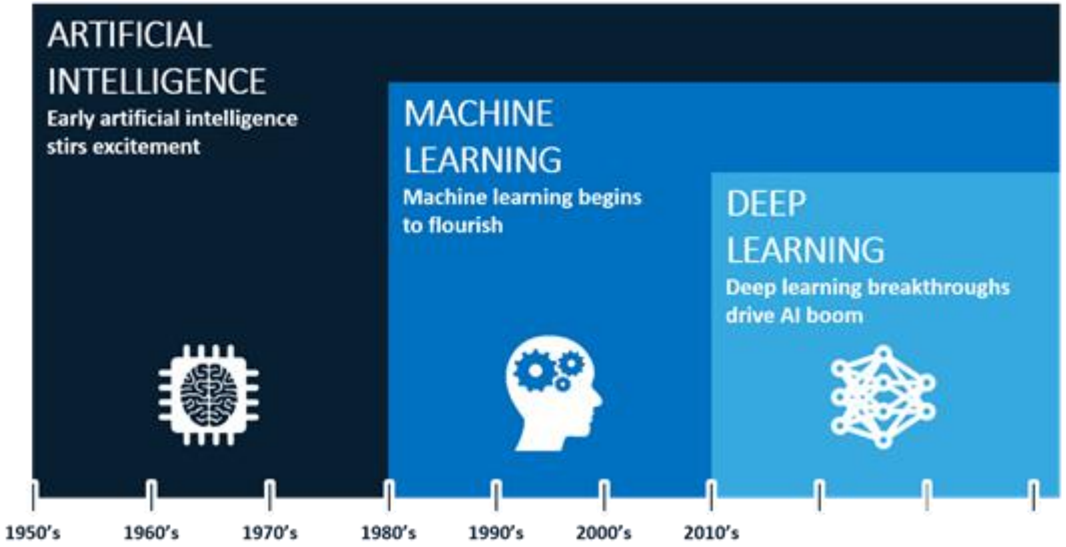
Instruct an autonomous vehicle to harvest

Harvested crops to packaging area

Packaging area to final delivery places

Artificial Intelligence

- Study of intelligent agents
- A system’s ability to correctly interpret external data, to learn from such data, use those learnings to achieve specific goals and tasks through flexible adaption



ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement

MACHINE LEARNING
Machine learning begins to flourish

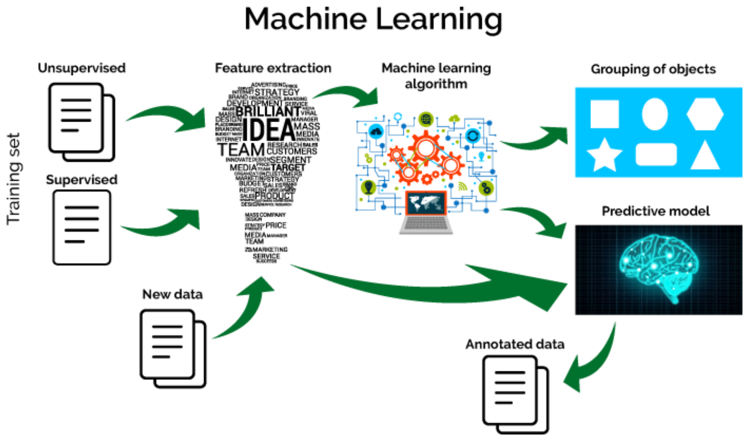
DEEP LEARNING
Deep learning breakthroughs drive AI boom

AI: Intelligence demonstrated by machines rather than humans or animals.

ML: Giving computers the skills to learn without explicit programming

DL: Is an ML subset, examining algorithms that learn and improve on their own.

Machine Learning



[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed.
—Arthur Samuel, 1959

A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.
—Tom Mitchell, 1997

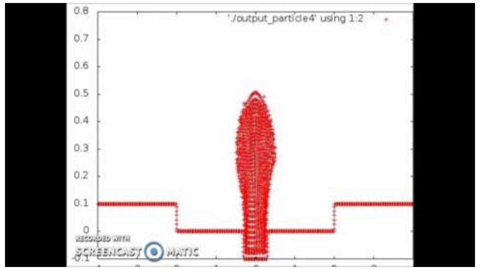
"Algorithms that parse data, learn from that data, and then apply what they've learned to make informed decisions"
https://www.zendesk.com/

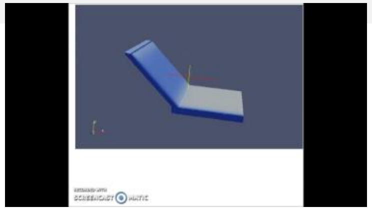
MY EXPERIENCES

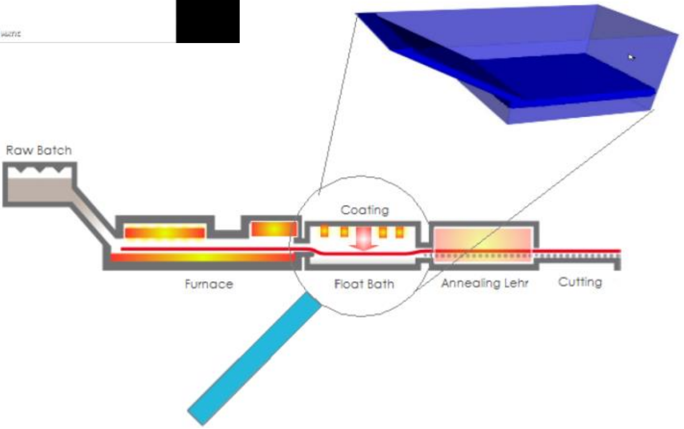
☑PhD & PostDoc: IIT Madras, TU Kaiserslautern, Fraunhofer ITWM

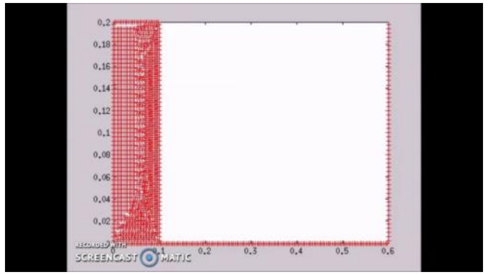
- Sewage Water
  - Manhole Problem
- Darcy Flow
- Schott Glass
  - Streak Formation
- Finite PointSet Method

Manhole Problem



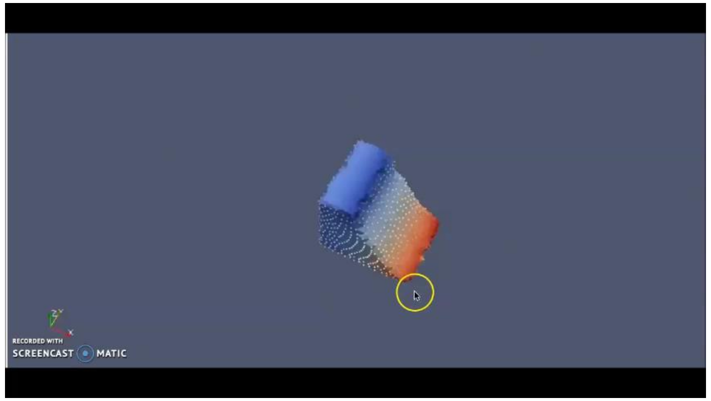




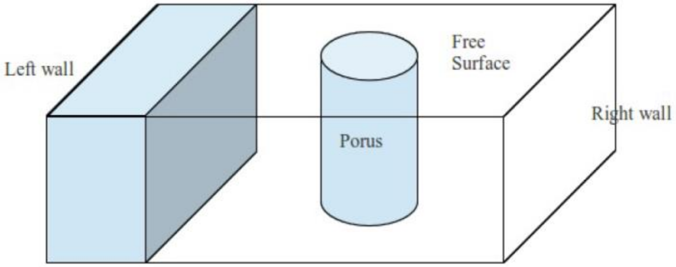


Company: Schott Glass

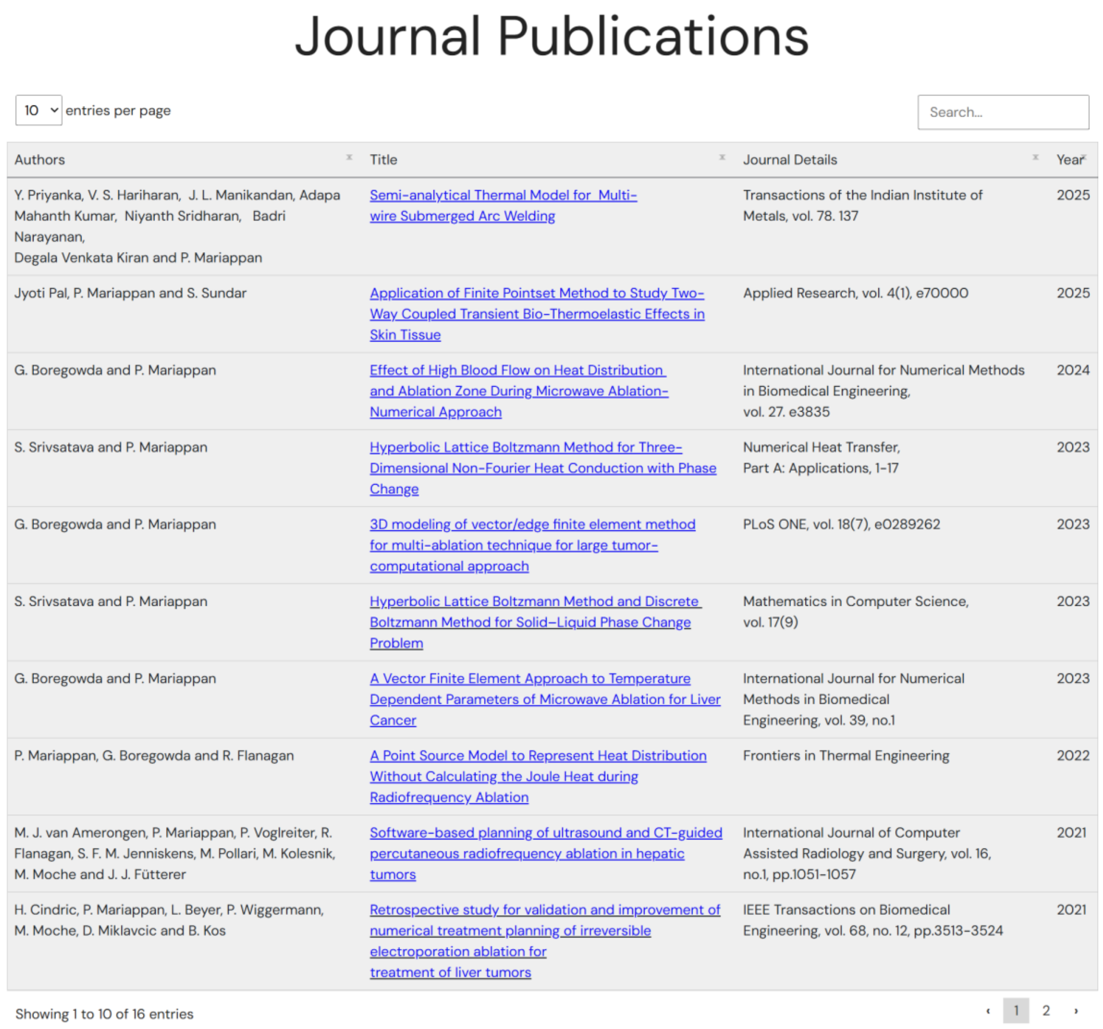
Steak Formation



Darcy Flow



MY EXPERIENCES



Journal Publications

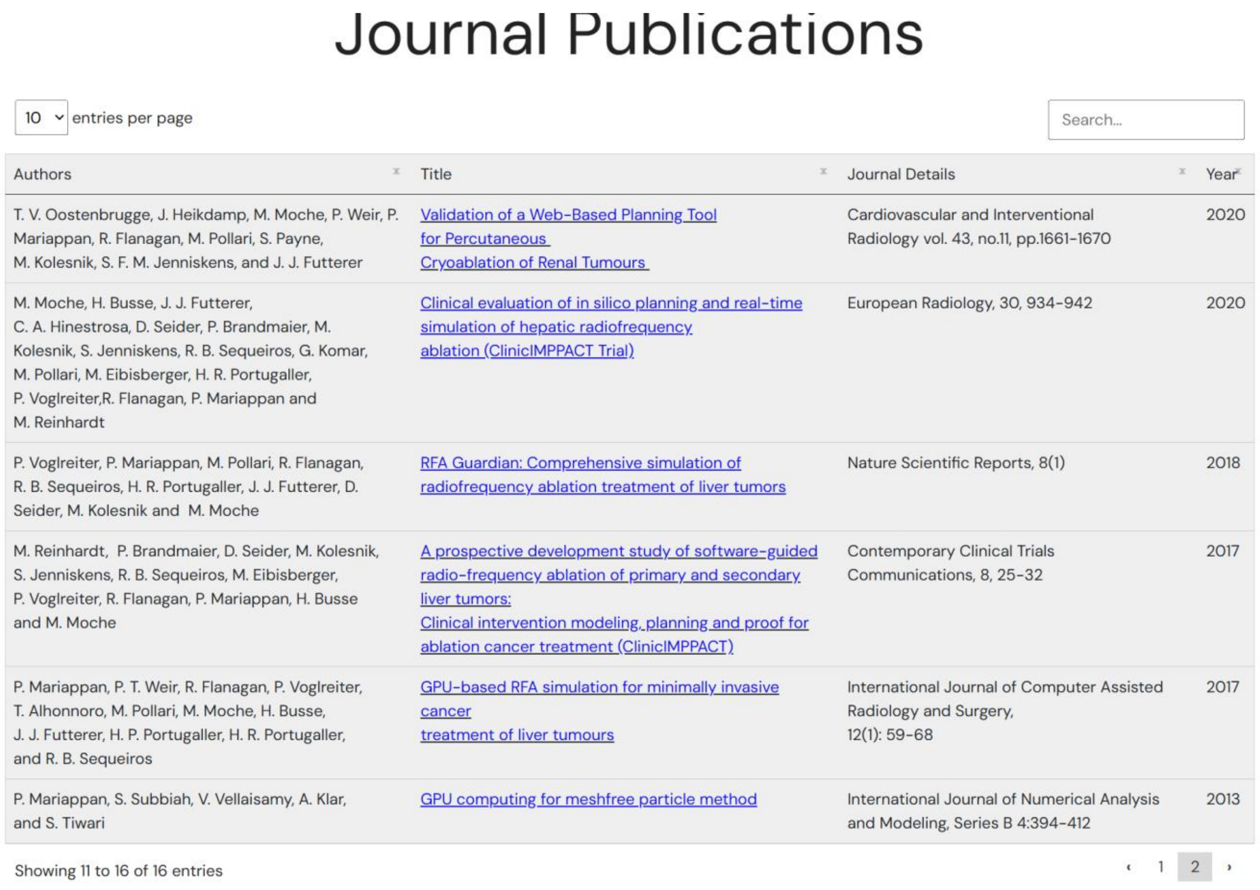
10 entries per page

Search...

Authors	Title	Journal Details	Year
Y. Priyanka, V. S. Hariharan, J. L. Manikandan, Adapa Mahanth Kumar, Niyanth Sridharan, Badri Narayanan, Degala Venkata Kiran and P. Mariappan	Semi-analytical Thermal Model for Multi-wire Submerged Arc Welding	Transactions of the Indian Institute of Metals, vol. 78. 137	2025
Jyoti Pal, P. Mariappan and S. Sundar	Application of Finite Pointset Method to Study Two-Way Coupled Transient Bio-Thermoelastic Effects in Skin Tissue	Applied Research, vol. 4(1), e70000	2025
G. Boregowda and P. Mariappan	Effect of High Blood Flow on Heat Distribution and Ablation Zone During Microwave Ablation-Numerical Approach	International Journal for Numerical Methods in Biomedical Engineering, vol. 27. e3835	2024
S. Srivsatava and P. Mariappan	Hyperbolic Lattice Boltzmann Method for Three-Dimensional Non-Fourier Heat Conduction with Phase Change	Numerical Heat Transfer, Part A: Applications, 1-17	2023
G. Boregowda and P. Mariappan	3D modeling of vector/edge finite element method for multi-ablation technique for large tumor-computational approach	PLoS ONE, vol. 18(7), e0289262	2023
S. Srivsatava and P. Mariappan	Hyperbolic Lattice Boltzmann Method and Discrete Boltzmann Method for Solid–Liquid Phase Change Problem	Mathematics in Computer Science, vol. 17(9)	2023
G. Boregowda and P. Mariappan	A Vector Finite Element Approach to Temperature Dependent Parameters of Microwave Ablation for Liver Cancer	International Journal for Numerical Methods in Biomedical Engineering, vol. 39, no.1	2023
P. Mariappan, G. Boregowda and R. Flanagan	A Point Source Model to Represent Heat Distribution Without Calculating the Joule Heat during Radiofrequency Ablation	Frontiers in Thermal Engineering	2022
M. J. van Amerongen, P. Mariappan, P. Voglreiter, R. Flanagan, S. F. M. Jenniskens, M. Pollari, M. Kolesnik, M. Moche and J. J. Fütterer	Software-based planning of ultrasound and CT-guided percutaneous radiofrequency ablation in hepatic tumors	International Journal of Computer Assisted Radiology and Surgery, vol. 16, no.1, pp.1051-1057	2021
H. Cindric, P. Mariappan, L. Beyer, P. Wiggermann, M. Moche, D. Miklavcic and B. Kos	Retrospective study for validation and improvement of numerical treatment planning of irreversible electroporation ablation for treatment of liver tumors	IEEE Transactions on Biomedical Engineering, vol. 68, no. 12, pp.3513-3524	2021

Showing 1 to 10 of 16 entries

‹ 1 2 ›



Journal Publications

10 entries per page

Search...

Authors	Title	Journal Details	Year
T. V. Oostenbrugge, J. Heikdamp, M. Moche, P. Weir, P. Mariappan, R. Flanagan, M. Pollari, S. Payne, M. Kolesnik, S. F. M. Jenniskens, and J. J. Futterer	Validation of a Web-Based Planning Tool for Percutaneous Cryoablation of Renal Tumours	Cardiovascular and Interventional Radiology vol. 43, no.11, pp.1661-1670	2020
M. Moche, H. Busse, J. J. Futterer, C. A. Hinestrosa, D. Seider, P. Brandmaier, M. Kolesnik, S. Jenniskens, R. B. Sequeiros, G. Komar, M. Pollari, M. Eibisberger, H. R. Portugaller, P. Voglreiter,R. Flanagan, P. Mariappan and M. Reinhardt	Clinical evaluation of in silico planning and real-time simulation of hepatic radiofrequency ablation (ClinicIMPPACT Trial)	European Radiology, 30, 934-942	2020
P. Voglreiter, P. Mariappan, M. Pollari, R. Flanagan, R. B. Sequeiros, H. R. Portugaller, J. J. Futterer, D. Seider, M. Kolesnik and M. Moche	RFA Guardian: Comprehensive simulation of radiofrequency ablation treatment of liver tumors	Nature Scientific Reports, 8(1)	2018
M. Reinhardt, P. Brandmaier, D. Seider, M. Kolesnik, S. Jenniskens, R. B. Sequeiros, M. Eibisberger, P. Voglreiter, R. Flanagan, P. Mariappan, H. Busse and M. Moche	A prospective development study of software-guided radio-frequency ablation of primary and secondary liver tumors: Clinical intervention modeling, planning and proof for ablation cancer treatment (ClinicIMPPACT)	Contemporary Clinical Trials Communications, 8, 25-32	2017
P. Mariappan, P. T. Weir, R. Flanagan, P. Voglreiter, T. Alhonnoro, M. Pollari, M. Moche, H. Busse, J. J. Futterer, H. P. Portugaller, H. R. Portugaller, and R. B. Sequeiros	GPU-based RFA simulation for minimally invasive cancer treatment of liver tumours	International Journal of Computer Assisted Radiology and Surgery, 12(1): 59-68	2017
P. Mariappan, S. Subbiah, V. Vellaisamy, A. Klar, and S. Tiwari	GPU computing for meshfree particle method	International Journal of Numerical Analysis and Modeling, Series B 4:394-412	2013

Showing 11 to 16 of 16 entries

‹ 1 2 ›

Programming: Why?

PROGRAMMING: WHY?

Critical Thinking and Solving Real-World Problems:	Applications in science, engineering, business, entertainment, healthcare, and more
Creativity and Innovation	Develop new algorithms, conduct data analysis, and build artificial intelligence
Career Opportunities	Technology, Data Science, Finance etc
Automation	Enable computers to perform repetitive or complex tasks efficiently
Simulation and Experimentation	Model physical phenomena (e.g., solving PDEs, weather forecasting)

PROGRAMMING: WHY?

- Computers are fast
- Cheap Labor for us: In fact, a slave to human
  - No strike, No hike
- Can work 24x7
  - No Rest, No 8 hour work rules
- Can solve complicated problem
  - Cryptography, bitcoins
  - See earlier applications

Programming: Famous quotes?

PROGRAMMING: QUOTES?

“Whether you want to uncover the secrets of the universe, or you just want to pursue a career in the 21st century, basic computer programming is an essential skill to learn.”
—Stephen Hawking, Theoretical Physicist, Cosmologist, Author

“Learning to write programs stretches your mind, and helps you think better, creates a way of thinking about things that I think is helpful in all domains.”
—Bill Gates, Co-Chairman, Bill & Melinda Gates Foundation, Co-Founder, Microsoft

PROGRAMMING: QUOTES?

“We salute the coders, designers, and programmers already hard at work at their desks, and we encourage every student who can’t decide whether to take that computer science class to give it a try.”
—Michael Bloomberg. Former Mayor, New York City

“Whether we’re fighting climate change or going to space, everything is moved forward by computers, and we don’t have enough people who can code. Teaching young people to code early on can help build skills and confidence and energize the classroom with learning-by-doing opportunities.”
—Richard Branson, Founder, Virgin Group

PROGRAMMING: QUOTES?

“Learning to code is learning to create and innovate.”
—Enda Kenny, Taoiseach, Ireland

“Learning to code is useful no matter what your career ambitions are.”
—Arianna Huffington, Founder, The Huffington Post

Programming: How?

BEGINNERS

- Used to skip the fundamentals and jump directly to the shiny tools, catch words, technology
  - It is vain
  - Can’t perform well in interview
  - Can’t develop a project

- Never jump into program unless you are clear with fundamentals

- Choose a programming language you are most comfortable with
    - Can be C, C++, Fortran, Python etc
- Understand the basic concepts of the languages
    - Syntax
    - Variables
    - Conditionals
    - Operators
    - Loops
    - ….

BEGINNERS

- Don't
  - Try learn multiple language at the same time
  - Keep on Jumping from one language to another

- Stick with one language

- Learning the first language is difficult
- Practice every day
- Write programs every single day until you get familiar with it

BEGINNERS

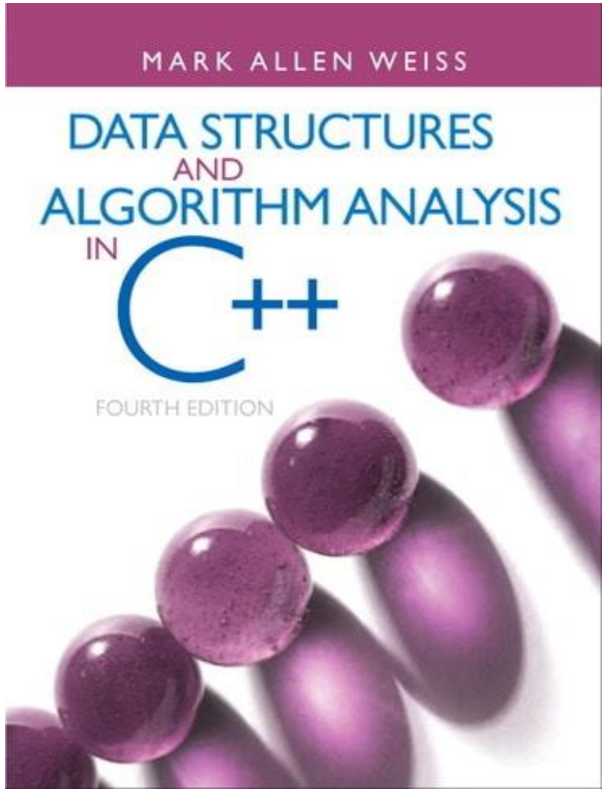
- Don’t
  - Learn all theories and then jump to program

- Learn two hours of conceptual and spend an hour in practical aspects of the learning
- Practice! Practice and Do more Practice!

- Create an application project based on the basics you have learnt
- Simple program: Calculator application
- Use Google, Stackoverflow, and other online resources when you commit mistakes
- Participate in Hackathon and competitive programming

DATA STRUCTURES AND ALGORITHMS

- Never jump into program unless you understand algorithms and data structure
- These two are heart of programming



How to develop code

Remember the syntax

Understand the problem

Identify inputs

Identify outputs

Identify the approach to solve problems

Draw a picture on how to solve? Flowchart?

HOW TO DEVELOP CODE

Write your own algorithm in a paper. Need not be efficient

Create Unit tests and see whether your algorithm provides desired output for given input

Select a programming language of your choice

Convert your algorithm to a code format using the programming language

Test your unit test

Mistakes should/must be there

Debug your code and retest until desired output is obtained

Improve the algorithm, think to make an efficient algorithm and code

DO'S AND DON'TS

Memorize	Never memorize any code instead understand the logic
Look	Never look at a problem in a big picture
Break down	Break down the problem into pieces
Try	Try to solve each pieces
Practice	Practice! More Practice! More and More Practice!
Don't panic	Don't panic while making mistakes, learn from it

Compiler vs Interpreter

Panchatcharam M

COMPUTER BASICS

Computer Systems

- Developed by Academia and Industry
- Daily usage: General Purpose Machines
- Specific applications: Special Purpose Machines
- Defined through their interfaces at a number of layered abstraction levels

Application Programs

High-Level Languages: Set of Machine Instructions

Language Architecture: Interface between Application Program and High-Level Language

Instruction set Architecture: Interface between machine instructions set and runtime, I/O Control

Four Basic Points

- Structure: Interconnection of various hardware components
- Organization Dynamic Interplay and Management of various components
- Implementation: Design of hardware components
- Performance: Behaviour of the computer system

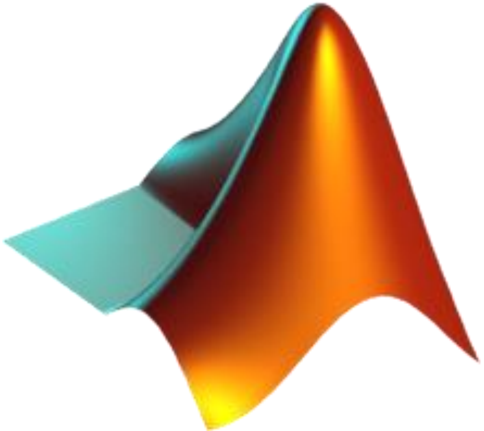
Hardware

- Hardware: Any Physical device used in or with machines

This Photo by Unknown Author is licensed under CC BY-SA

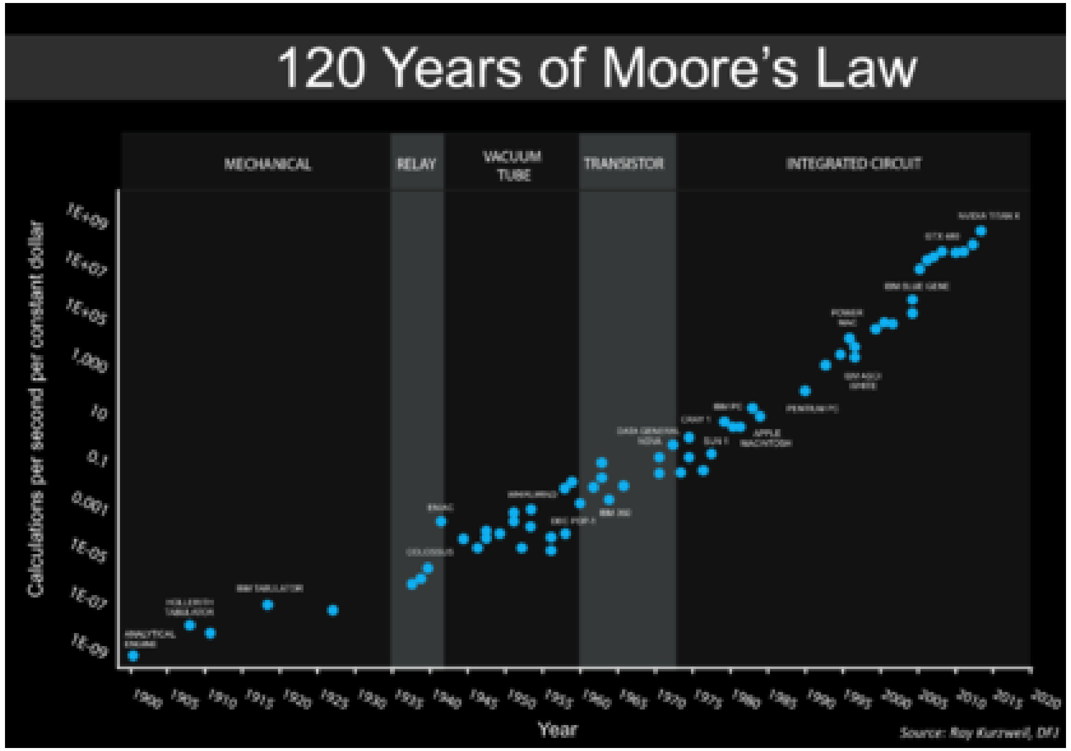
Software

- Software: Collection of Code Installed on computers' hard drive



Moore's Law

- Billions of Calculations in one second
- SuperComputers: Quadrillions of instructions per second
- Computer Programs: Computer processes data under the control of sequences of instructions
- Guides the computers through ordered actions
- Guided by people: Programmers
- Hardware cost decreases rapidly
- Capacities of computers doubles every year
- Number of transistors in dense integrated circuit doubles every year
- SSI,LSI,VLSI,VVLSI,UVLSI,WSI,SOC,3D-IC



120 Years of Moore's Law

Source: Ray Kurzweil, DFJ

LANGUAGES

Machine Language

Computer can directly understand only its own ML

Defined by its hardware design

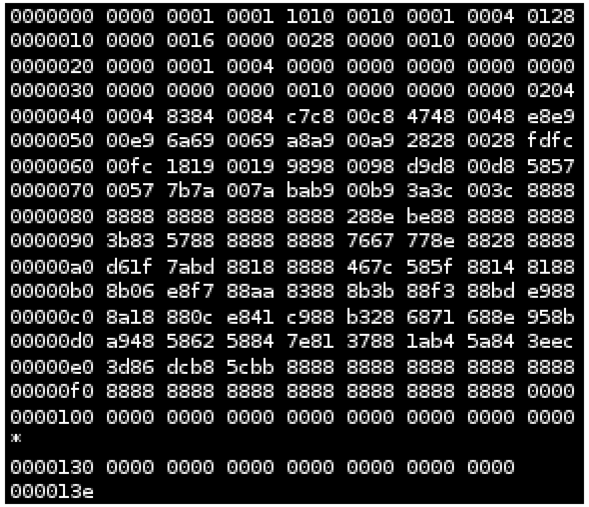
Strings of Numbers (0s and 1s)

Machine Dependent

Difficult for human to understand

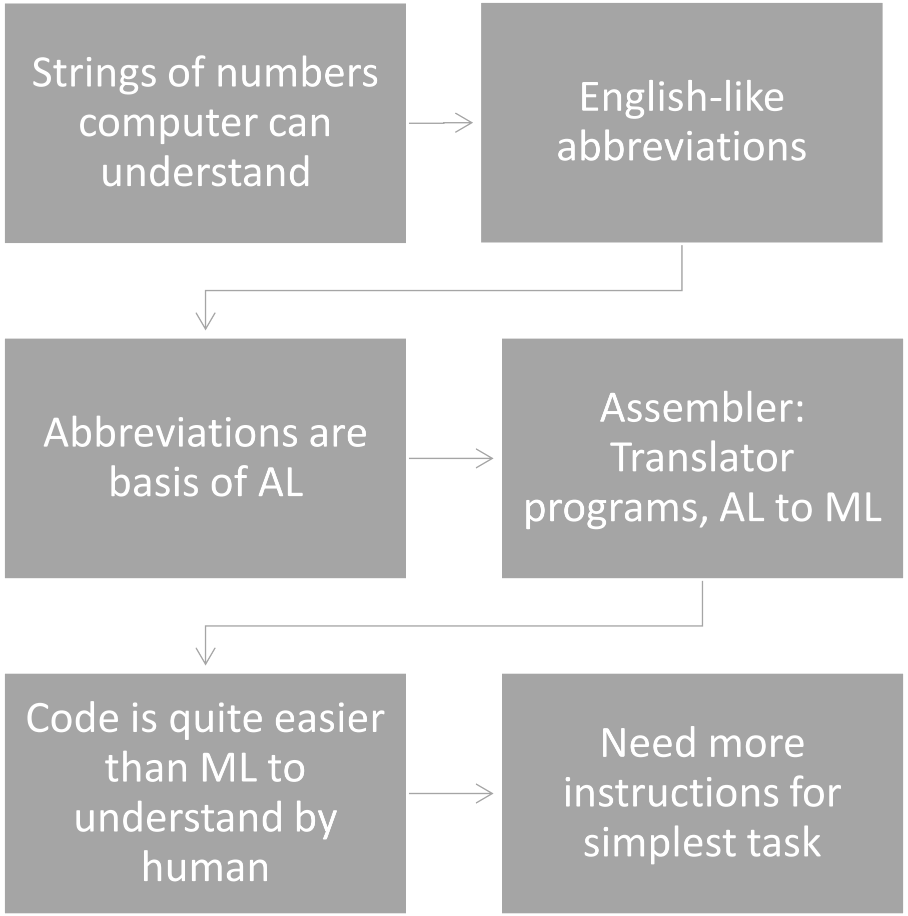
Slow and tedious for a programmer

- It is the lowest-level programming language which only the specific computer can understand, consists of strings of numbers and almost impossible for humans to understand.



```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

Assembly Language



Strings of numbers computer can understand

English-like abbreviations

Abbreviations are basis of AL

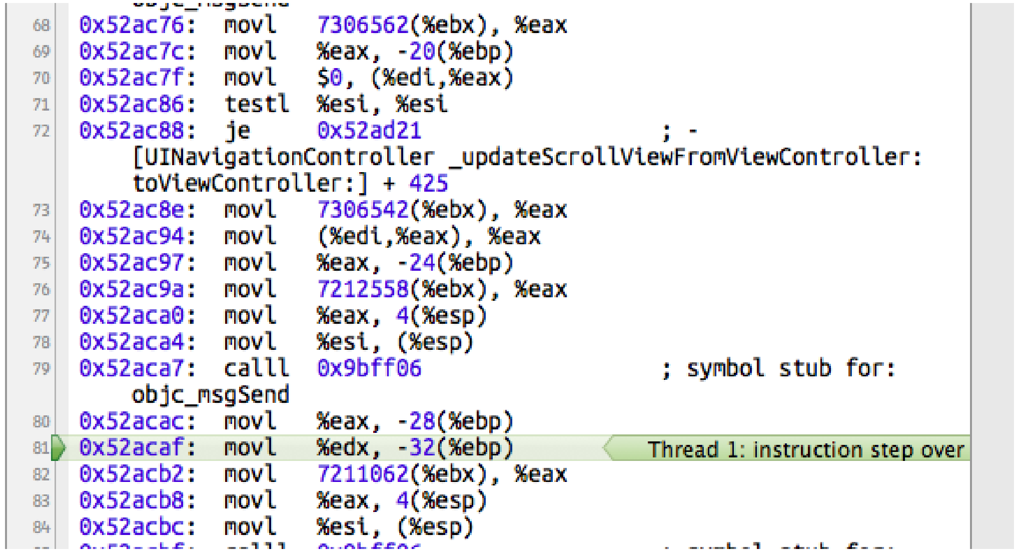
Assembler: Translator programs, AL to ML

Code is quite easier than ML to understand by human

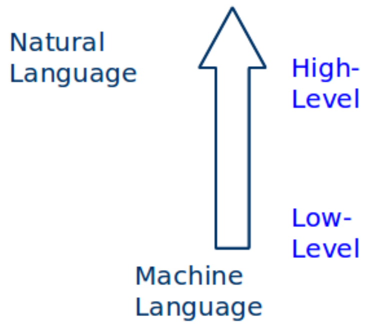
Need more instructions for simplest task

```
MOV AL, 1h ; Load AL with immediate value 1
MOV CL, 2h ; Load CL with immediate value 2
MOV DL, 3h ; Load DL with immediate value 3
```

- It is a low level programming language that allows a user to write a program using alphanumeric mnemonic codes instead of numeric codes for a set of instructions. It can be translated using an assembler into machine language

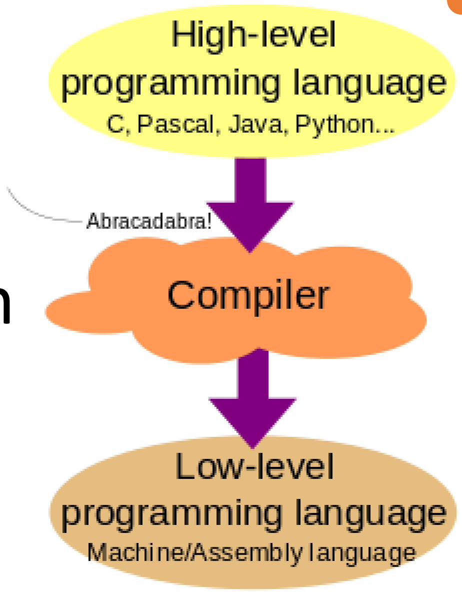


```
0x52ac76:  movl   7306562(%ebx), %eax
0x52ac7c:  movl   %eax, -20(%ebp)
0x52ac7f:  movl   $0, (%edi,%eax)
0x52ac86:  testl  %esi, %esi
0x52ac88:  je     0x52ad21                       ; -
    [UINavigationController _updateScrollViewFromViewController:
    toViewController:] + 425
0x52ac8e:  movl   7306542(%ebx), %eax
0x52ac94:  movl   (%edi,%eax), %eax
0x52ac97:  movl   %eax, -24(%ebp)
0x52ac9a:  movl   7212558(%ebx), %eax
0x52aca0:  movl   %eax, 4(%esp)
0x52aca4:  movl   %esi, (%esp)
0x52aca7:  calll  0x9bff06                       ; symbol stub for:
    objc_msgSend
0x52acac:  movl   %eax, -28(%ebp)
0x52acaf:  movl   %edx, -32(%ebp)
0x52acb2:  movl   7211062(%ebx), %eax
0x52acb8:  movl   %eax, 4(%esp)
0x52acbc:  movl   %esi, (%esp)
```



High Level Language

- It is a programming language that is understood by humans/programmers. It can be translated using a translator, for example, compiler or interpreters, into a simple machine language that computer can understand and execute. It does not depend on specific computer.


High-level programming language
C, Pascal, Java, Python...
Abracadabra!
Compiler
Low-level programming language
Machine/Assembly language

Single Statement to accomplish substantial tasks

Compilers: Translator program HLL to ML

Easy to understand

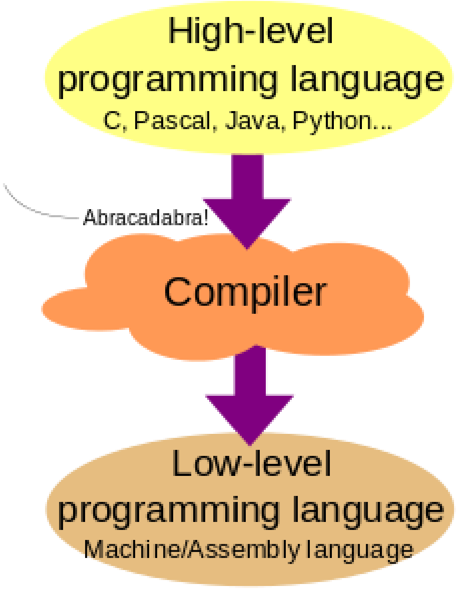
Variables, Arrays, Objects, Loop

Boolean, Functions, threads, abstract

High Level Language

- It is a programming language that is understood by humans/programmers. It can be translated using a translator, for example, compiler or interpreters, into a simple machine language that computer can understand and execute. It does not depend on specific computer.

```
 #include <iostream>
using namepsace std;
int main()
{
        int a=3,b=4;
        cout<<"Hello"<<endl;
        cout<<a+b<<endl;
        retun 0;
}
```

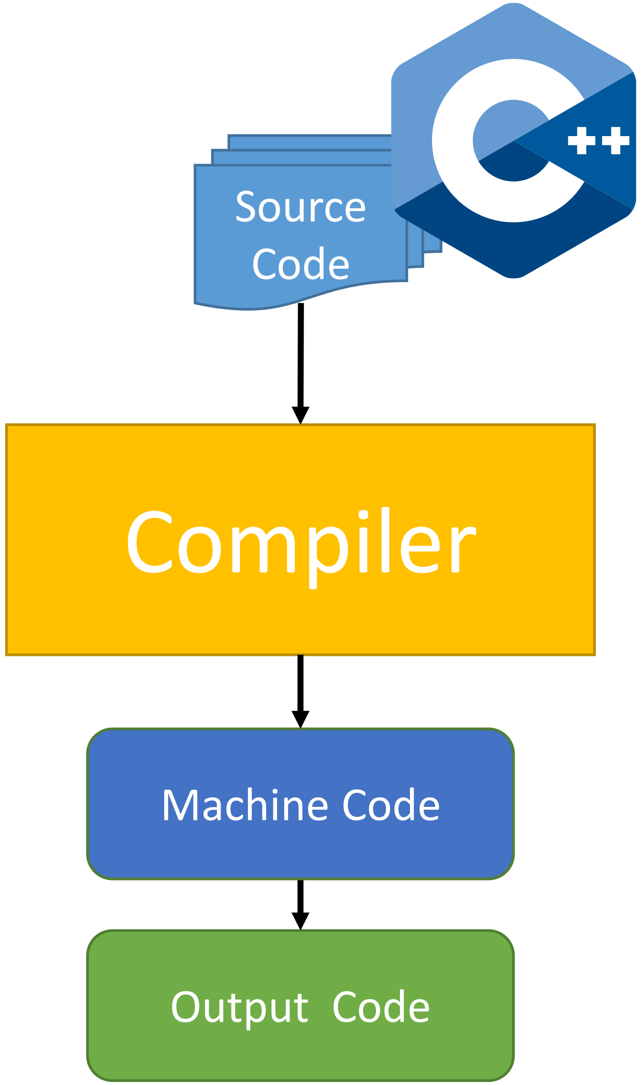

High-level programming language
C, Pascal, Java, Python...
Abracadabra!
Compiler
Low-level programming language
Machine/Assembly language

COMPILER

Compiler

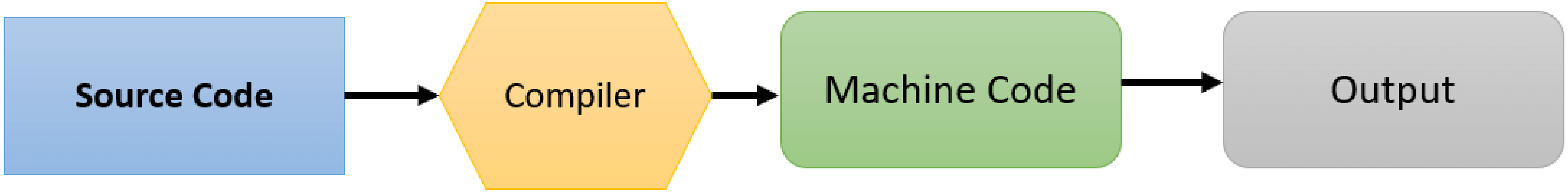
- A compiler is a program that reads a program written in the high-level language and converts it into the machine or low-level language and reports the errors present in the program.

It converts the entire source code in one go or could take multiple passes to do so, but at last, the user gets the compiled code which is ready to execute.



Compiler

How Compiler Works

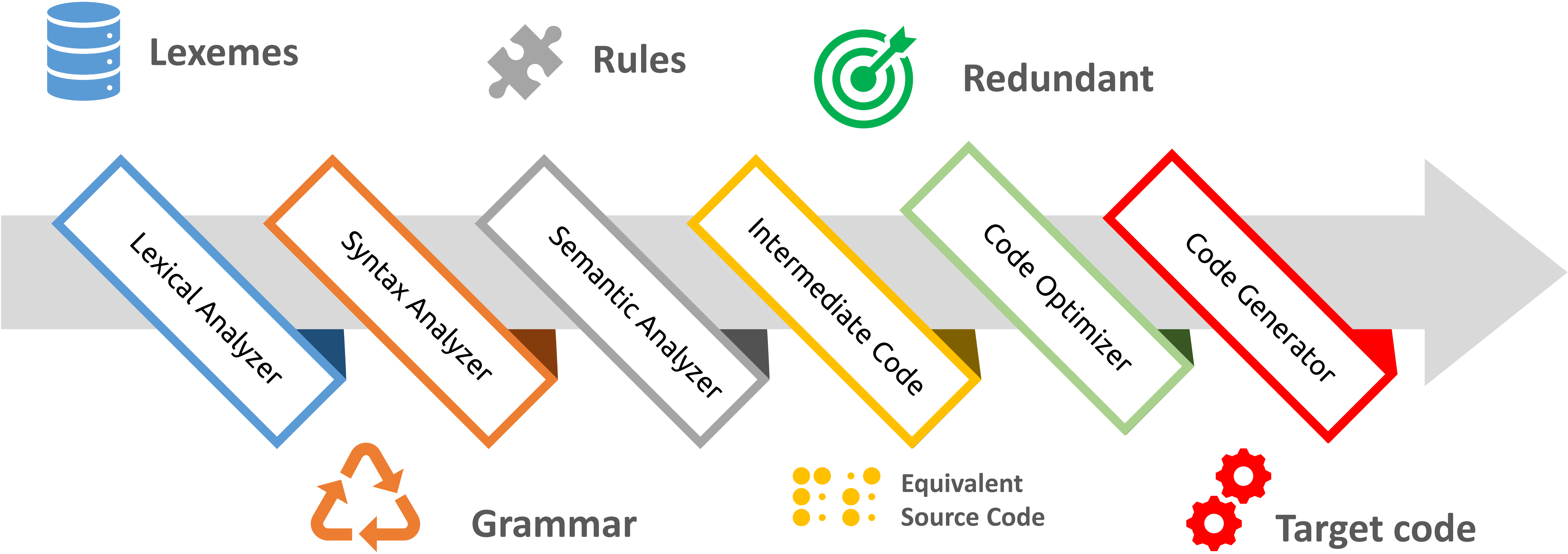


Source Code → Compiler → Machine Code → Output

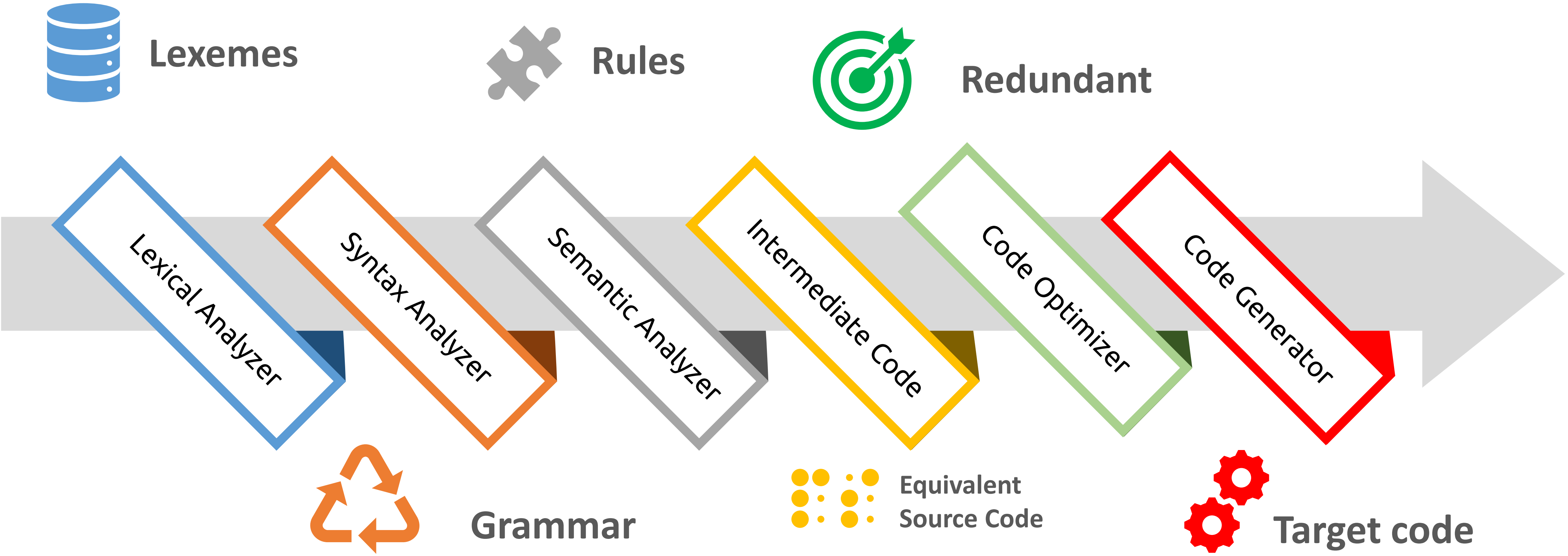
C++

Scala

6 Phases Compiler

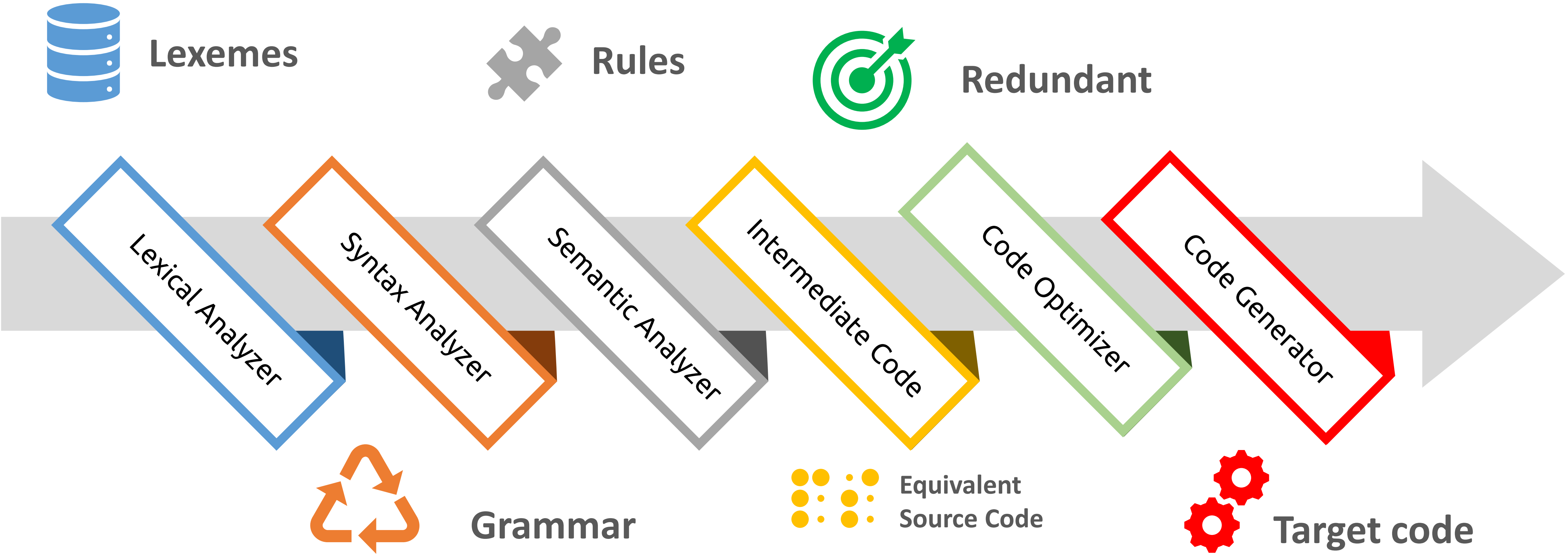


Lexical Analyzer (Scanning)



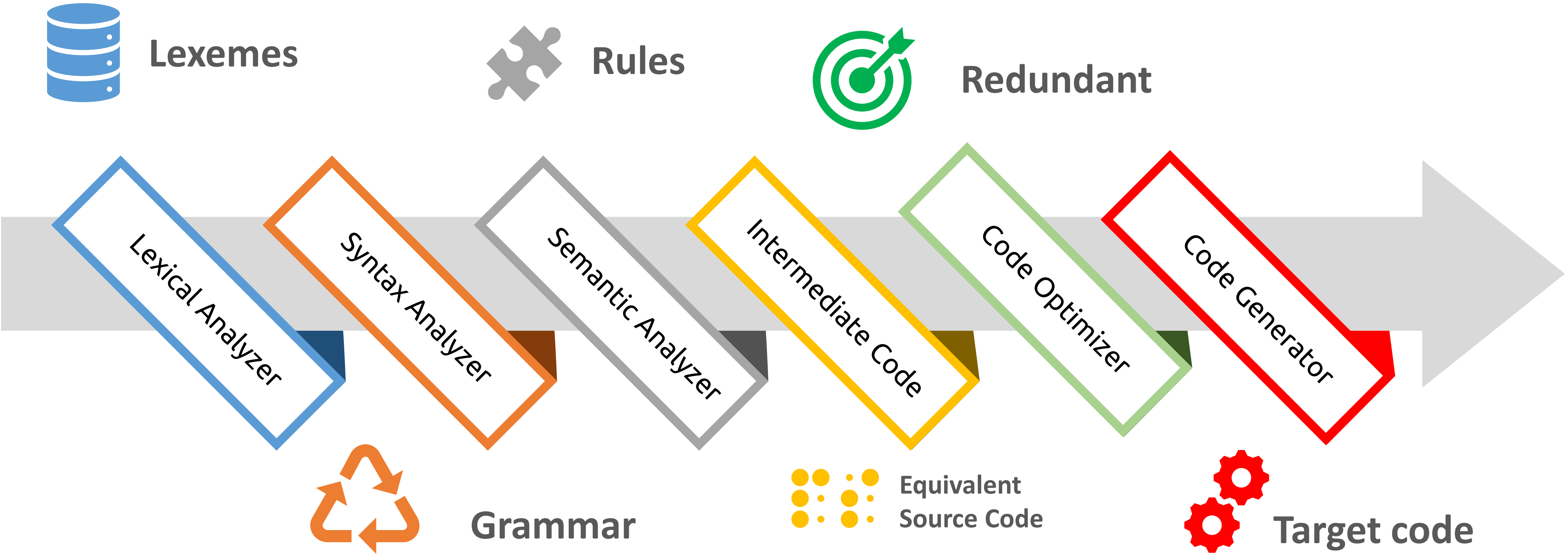
Scans the code as a stream of characters into lexemes. Output: Sequence of tokens with reference to the programming languages

Lexical Analyzer (Scanning)



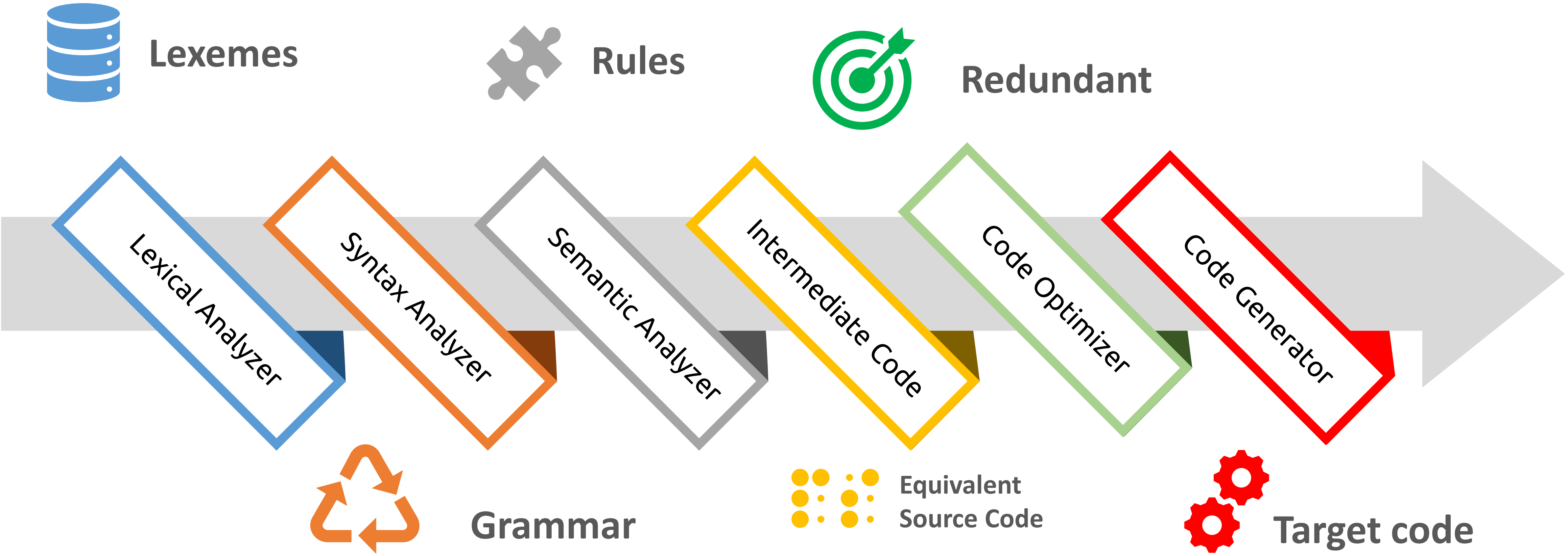
Input: int a=b+1;, Output: Keyword [int], identifier [a,b], operator [=,+] Number[1], Symbol [;]. Error: Unrecognized symbols, like @$

Syntax Analyzer (Parser))



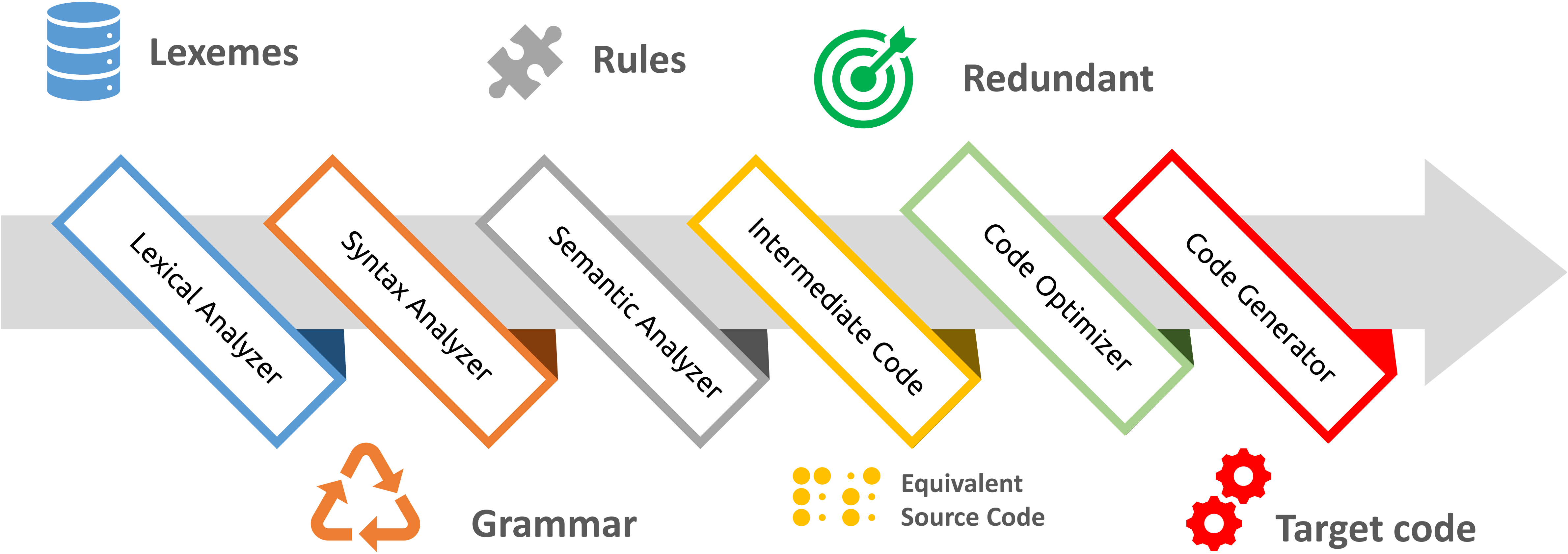
Tokens generated in Lexical analyzer phase are against grammar of programming language. Checks whether the expressions are syntactically correct or not. It makes parse trees

Syntax Analyzer (Parser)



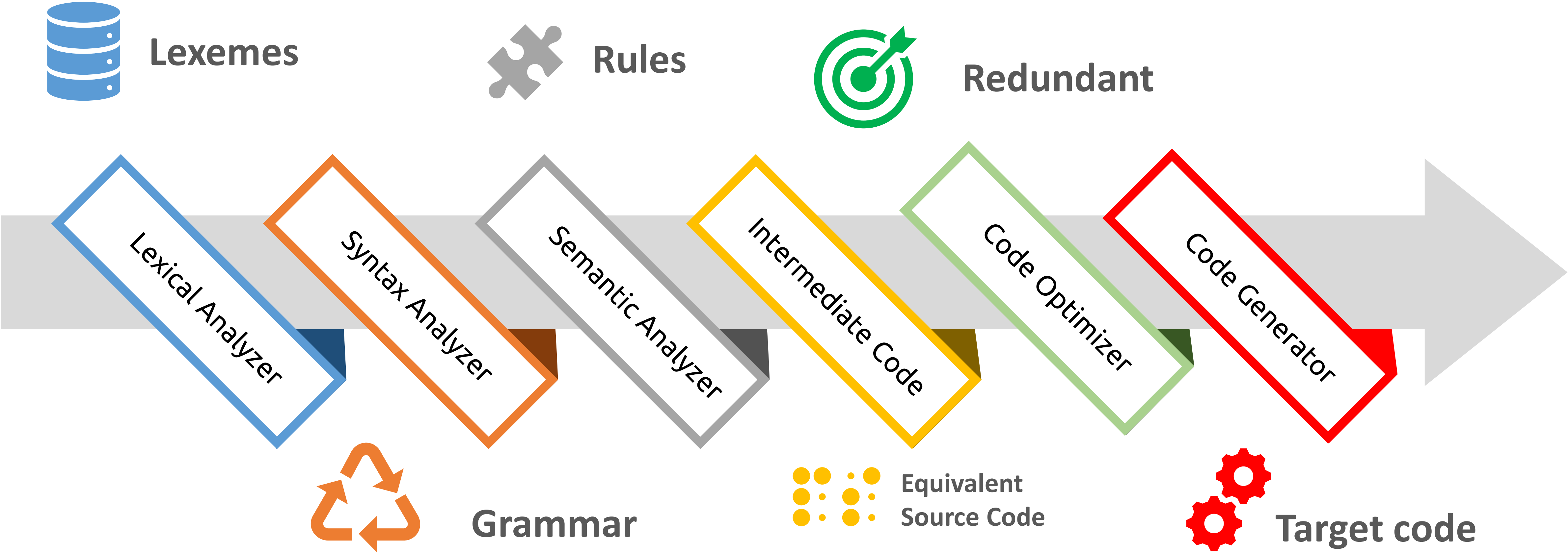
Input: int = x 3; Error will be thrown. Missing semicolon or mismatched brackets

Semantic Analyzer (Meaning)



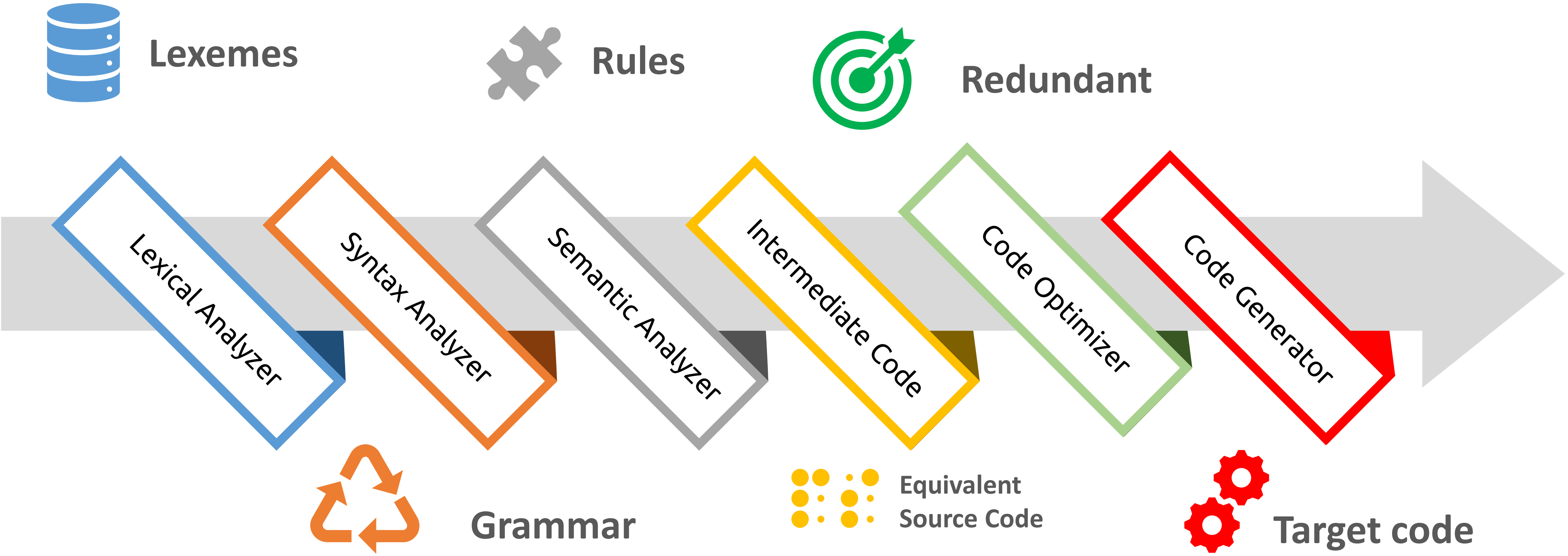
Checks whether the expressions and statements generated by previous phase follow the rule of programming language or not. Creates annotated parse trees

Semantic Analyzer (Meaning)



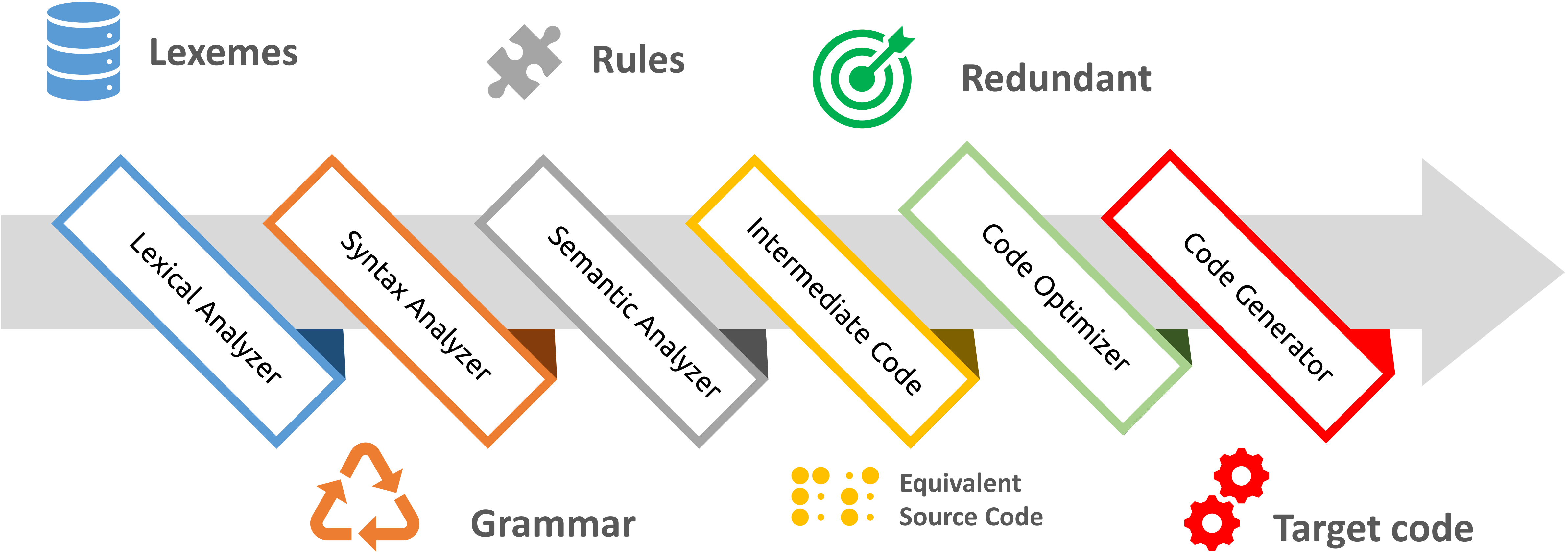
Input: x=5+”hello”. Error as string and integer addition is an error. Type mismatch or undeclared variable.

Intermediate Code Generation (IR)



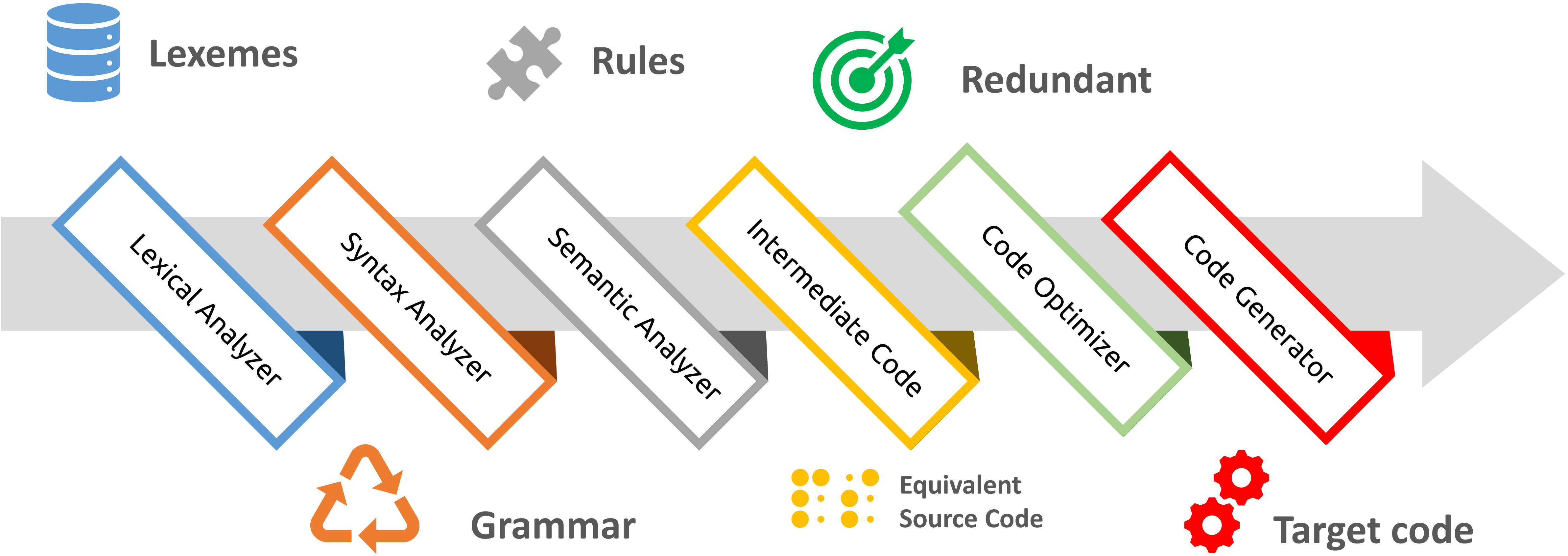
Equivalent intermediate code of the source code

Intermediate Code Generation (IR)



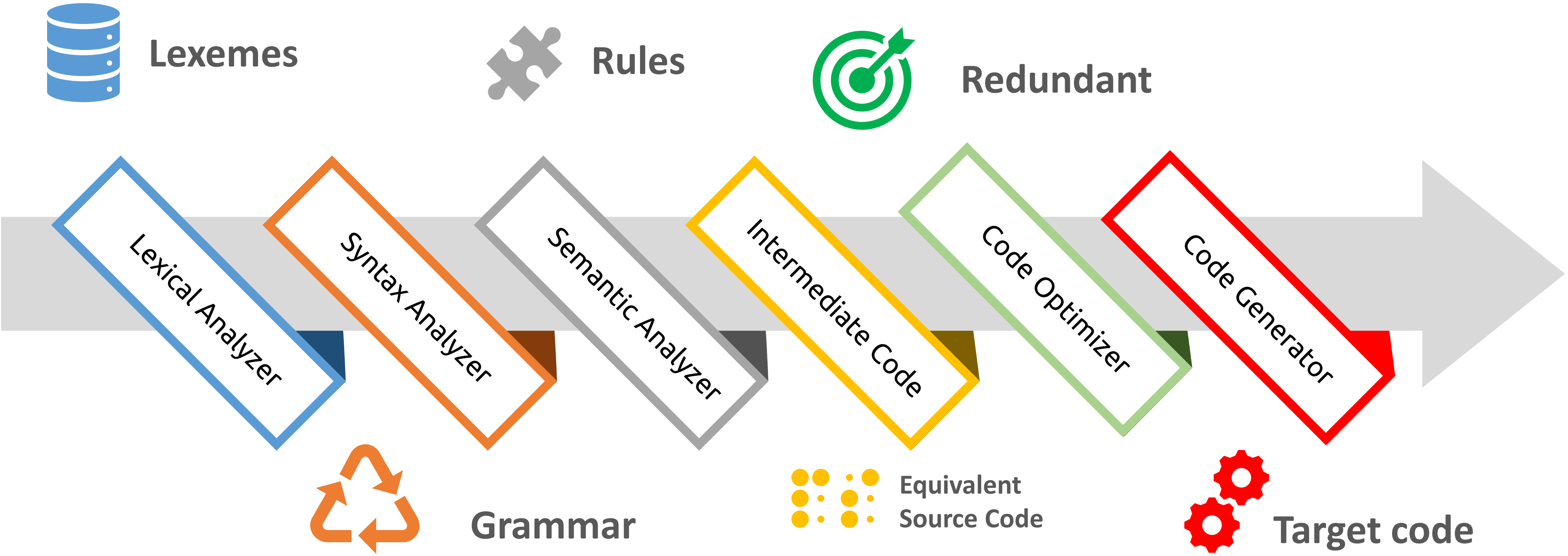
a=b+c, IR : t1=b+c, a=t1. Optimization and Portability

Code Optimizer



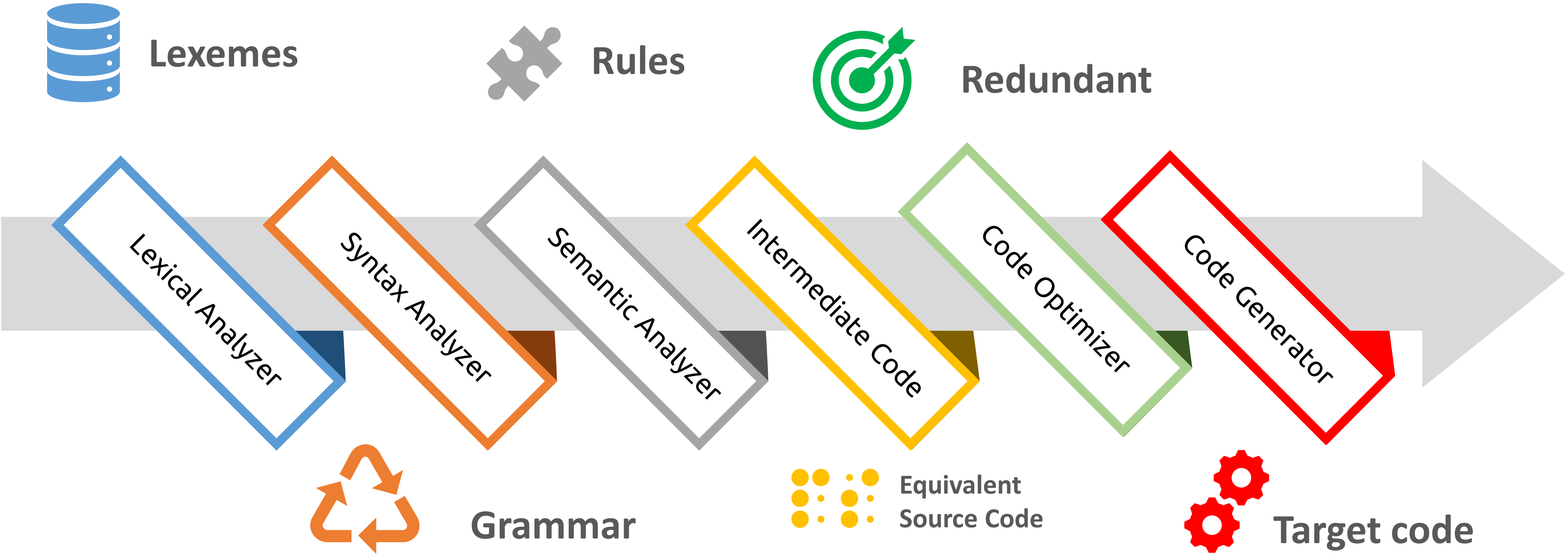
Improves the space and time requirements of the program.
Eliminates the redundant code, unused variables, dead code.

Code Optimizer



int a=6*0, is optimized by a=0; (Not always guaranteed)

Code Generator

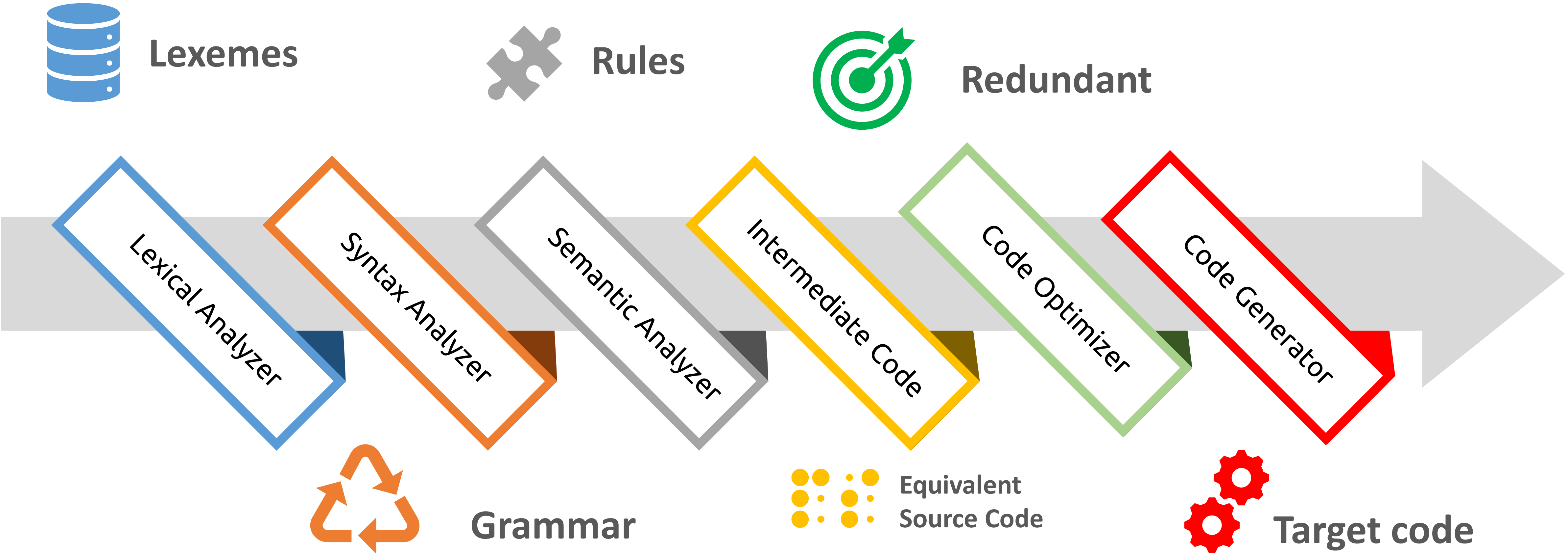


Final phase.

Target code for a particular machine is generated. Executable Binary or Assembly

Performs memory, register management and machine specific optimization

Code Generator



t1=b+1:

```
MOV R1, b
ADD R1, c
MOV a, R1
```

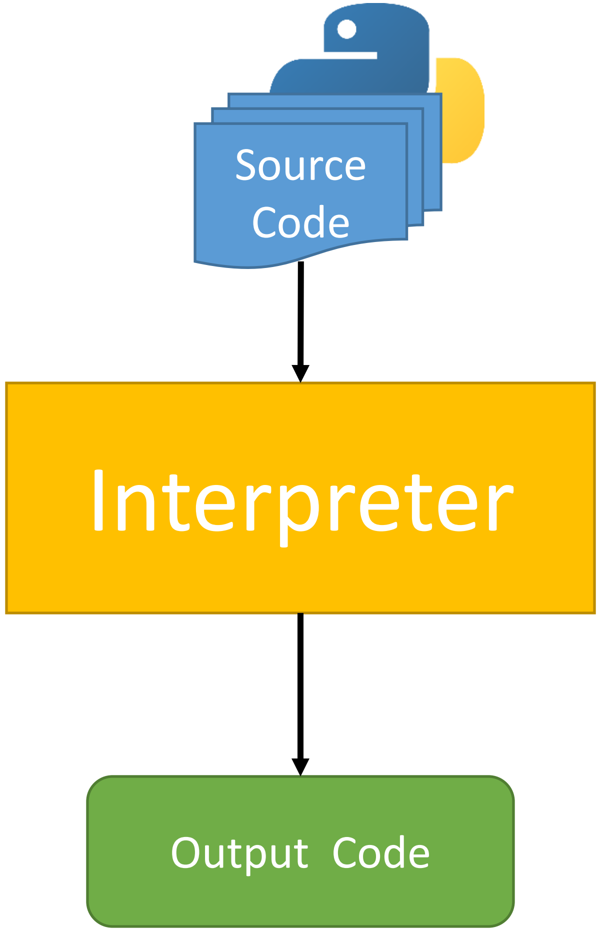
6 Phases of Compilers

Process	Key Task	Output
Lexical Analysis	Break source into tokens	Tokens
Syntax Analysis	Check the Grammar Rules	Parse Tree
Semantic Analysis	Check meaning and type rules	Annotated Tree
Intermediate Code Generation	Convert to Intermediate Representation form	IR (3-Address Code)
Code Optimization	Improve Code Performance	Optimized IR
Code Generation	Generate Machine/Assembly Code	Target Code

INTERPRETER

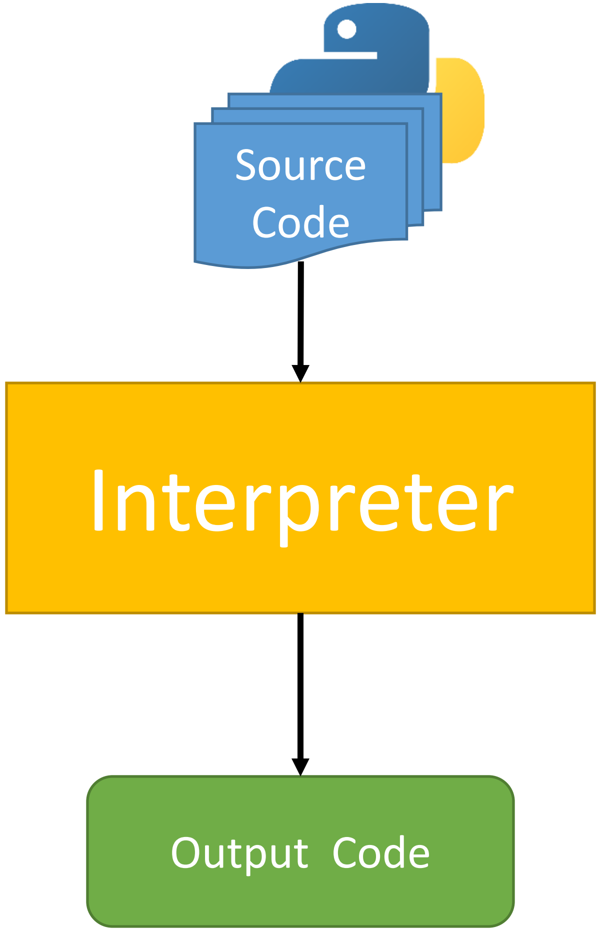
Interpreter

- An alternative for implementing a programming language and does the same work as compiler
- It Performs lexing, parsing and type checking similar to compiler.



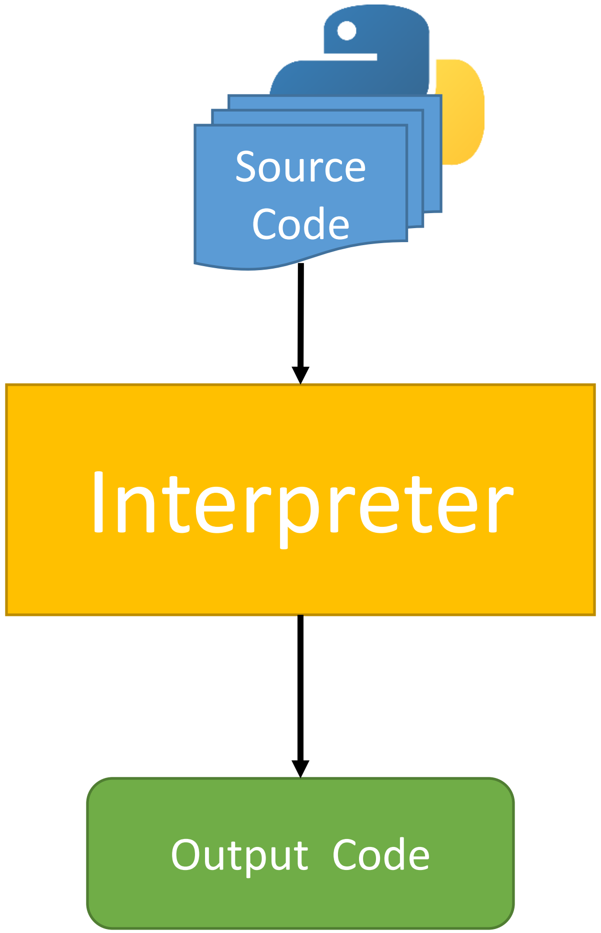
Interpreter

- Processes syntax tree directly access expressions and executes statements rather than generating code from the syntax tree
- Require processing same syntax tree more than once. Slower than compiler



Interpreter

- Large HLL to ML takes more time to compile
- Interpreters: Developed to execute HLL directly
- No compilation delay
- Slower than compiled programs

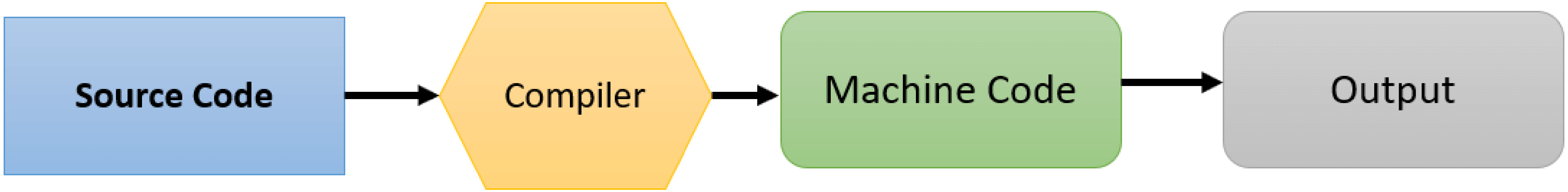

Source Code

Interpreter

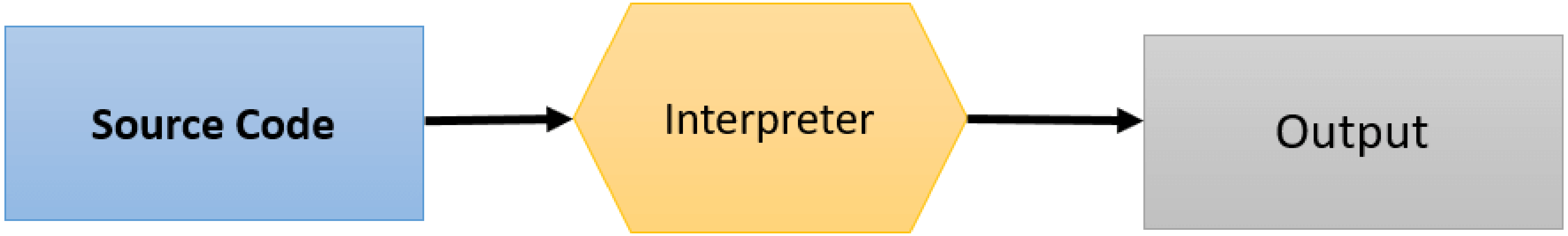
Output Code

Compiler vs Interpreter

How Compiler Works



How Interpreter Works



Compiler vs Interpreter

Process	Compiler	Interpreter
Input	Takes an entire program at a time	Takes a single line of code at a time
Output	Generates intermediate object code	Won’t produce any intermediate object code
When?	Before execution	Simultaneous compilation and execution
Speed	Faster	Slower
Memory Requirement	More for object code	less, no object code
Errors	All errors at a time after compilation, difficult	Error, line by line, easier

PROGRAM PARADIGMS

Program Paradigms

01 Declarative

- Focus more on specifying what a language is supported to accomplish rather than by what means it is suppose to accomplish.
- Use to avoid undesired side-effects

02 Functional

- It is a subset of declarative programming
- Tries to express problems in mathematical equations & functions
- Goes out of its way to avoid concepts of states, mutable variables

03 Generic

- Focus on writing skeleton algorithms in terms of types that will be specified when the algorithm is actually used.
- Allows leniency to programmers to avoid strict strong typing rules
- Powerful paradigm if well-implemented

04 Imperative

- Allow programmers to give the computer-ordered list of instructions without necessarily have to effectively state the task
- Opposite of declarative languages

Program Paradigms

05

Structured

- Provide some form of noteworthy structure to language
- Intuitive control over the order in which statements are executed
- Examples: C, C++

06

Procedural

- Imperative structured programming language
- Support concepts of procedure, subroutines and functions
- Examples: C++, C, Fortran, Python

07

Object Oriented

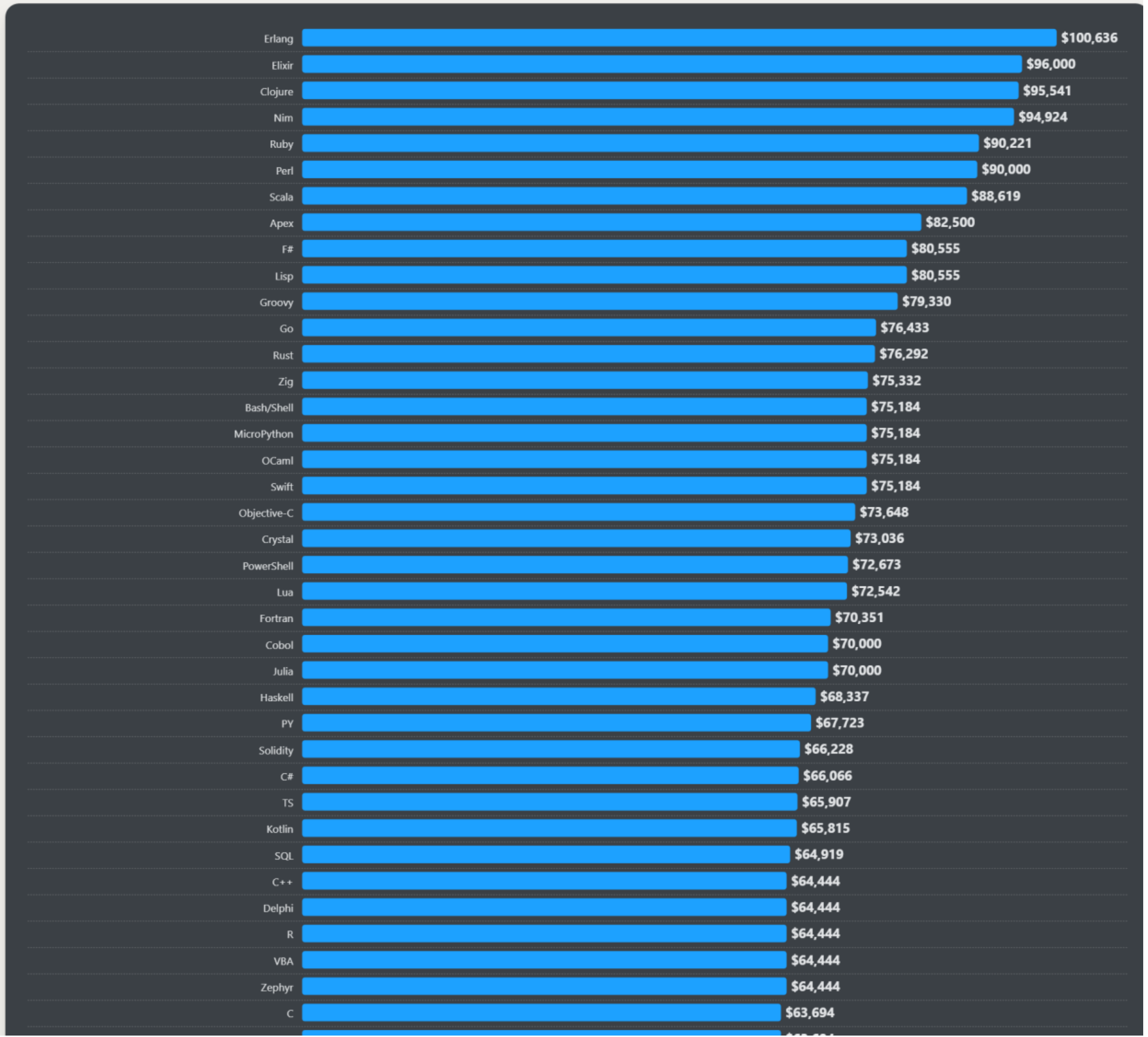
- Subset of structured
- Expresses in terms of objects
- Objects mean to objects in the real world

07

Object Oriented

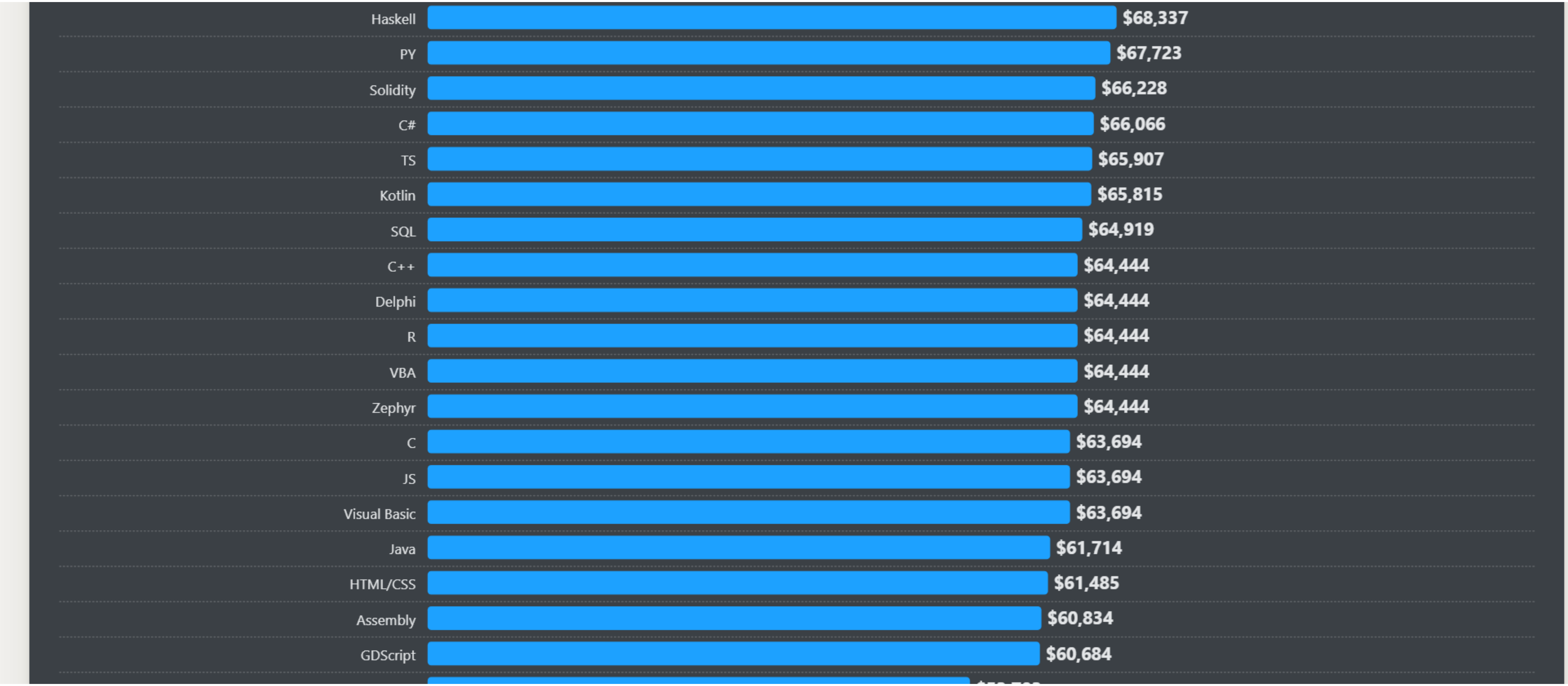
- Reusable, remarkable
- Easy to understand and use

Top Salaried Languages



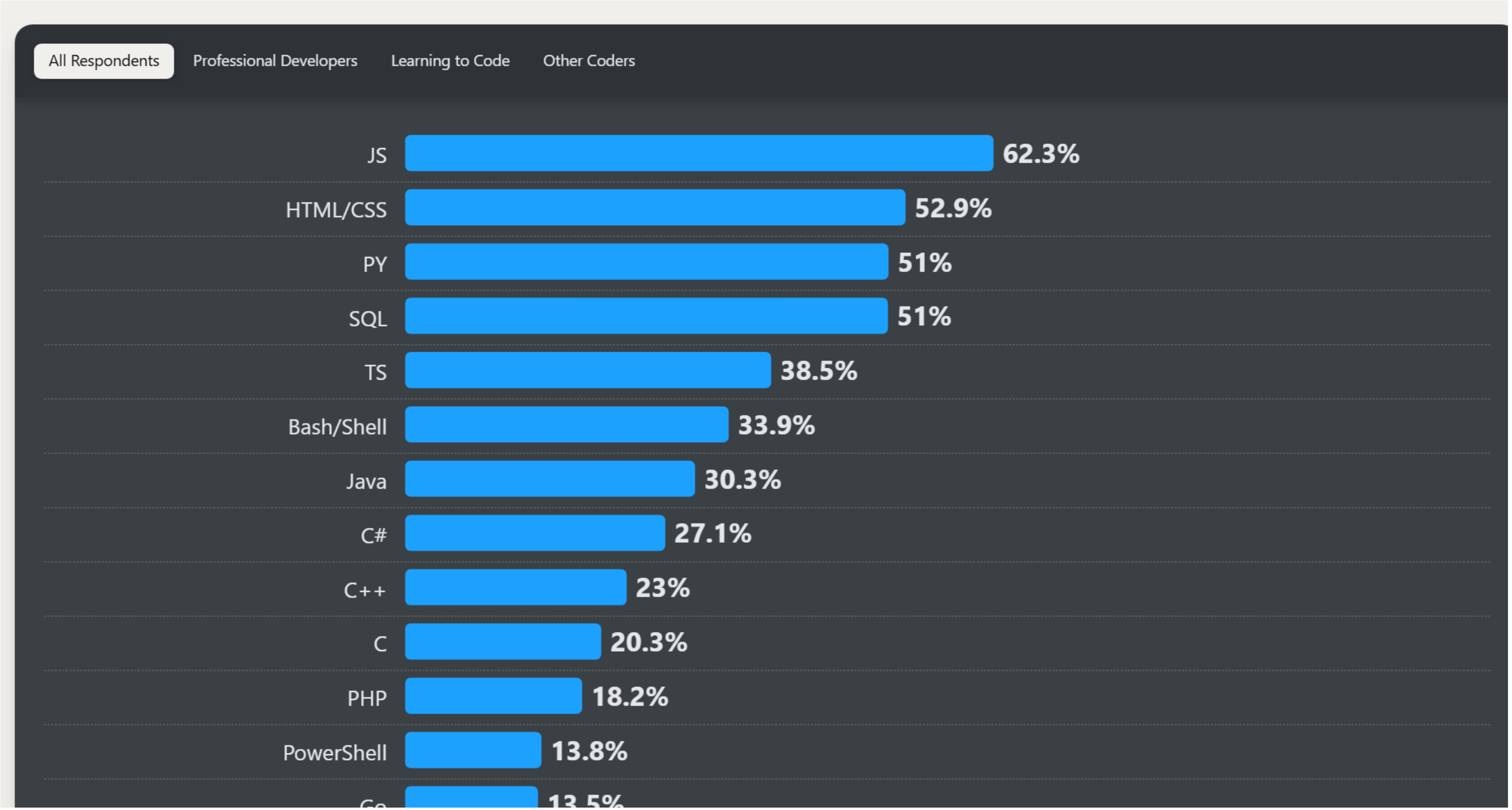
https://survey.stackoverflow.co/2024/technology#4-top-paying-technologies

Top Salaried Languages



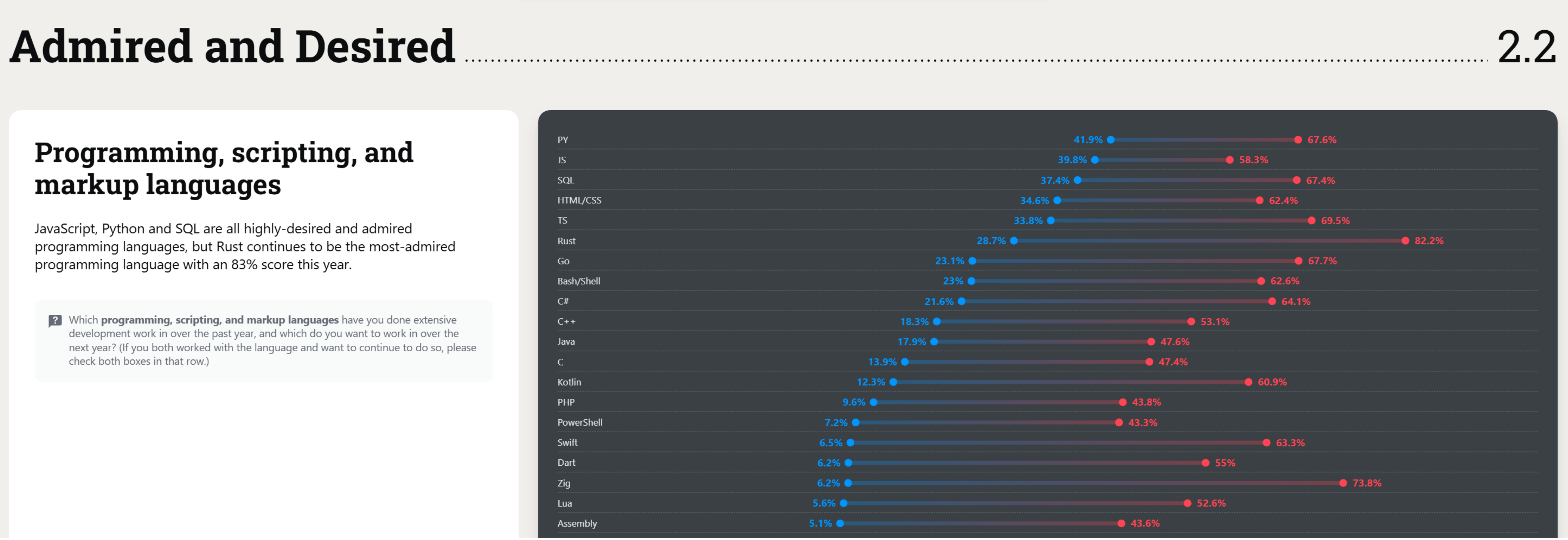
https://survey.stackoverflow.co/2024/technology#4-top-paying-technologies

Most Popular Languages



https://survey.stackoverflow.co/2022#most-popular-technologies-language

Admired and Desired



Admired and Desired 2.2

Programming, scripting, and markup languages

JavaScript, Python and SQL are all highly-desired and admired programming languages, but Rust continues to be the most-admired programming language with an 83% score this year.

Which **programming, scripting, and markup languages** have you done extensive development work in over the past year, and which do you want to work in over the next year? (If you both worked with the language and want to continue to do so, please check both boxes in that row.)

https://survey.stackoverflow.co/2024/technology

Further Reading

https://www.fullstackacademy.com/blog/nine-best-programming-languages-to-learn

https://cs.lmu.edu/~ray/notes/paradigms/

https://insights.stackoverflow.com/survey/2020#technology-what-languages-are-associated-with-the-highest-salaries-worldwide-global

Basics of C++

Panchatcharam M

C++

What is C++?

- A programming language
- Open ISO-Standardized language: Since 1998
- A compiled language

Features

- ✓Strongly-type unsafe language
- ✓Supports both manifest and inferred typing
- ✓Supports both static and dynamic type checking
- ✓Offers many paradigm of choices: procedural, generic, OOPS

Features

```
irror_mod = m
set mirror object to mirror
mirror_mod.mirror_object
operation == "MIRROR_X":
irror_mod.use_x = True
irror_mod.use_y = False
irror_mod.use_z = False
_operation == "MIRROR_Y":
irror_mod.use_x = False
irror_mod.use_y = True
irror_mod.use_z = False
_operation == "MIRROR_Z":
irror_mod.use_x = False
irror_mod.use_y = False
irror_mod.use_z = True

selection at the end -add
_ob.select= 1
er_ob.select=1
ntext.scene.objects.active
("Selected" + str(modifier
irror_ob.select = 0
bpy.context.selected_obj
ata.objects[one.name].sel

int("please select exactl

-- OPERATOR CLASSES

types.Operator):
X mirror to the selected
ject.mirror_mirror_x"
ror X"

ontext):
ext.active obje
```

✓Portable: same code may work with different C++ compilers, e.g, code developed in g++ can run on MSVC

✓Upwards compatible with C: Can use C libraries with few or no modifications

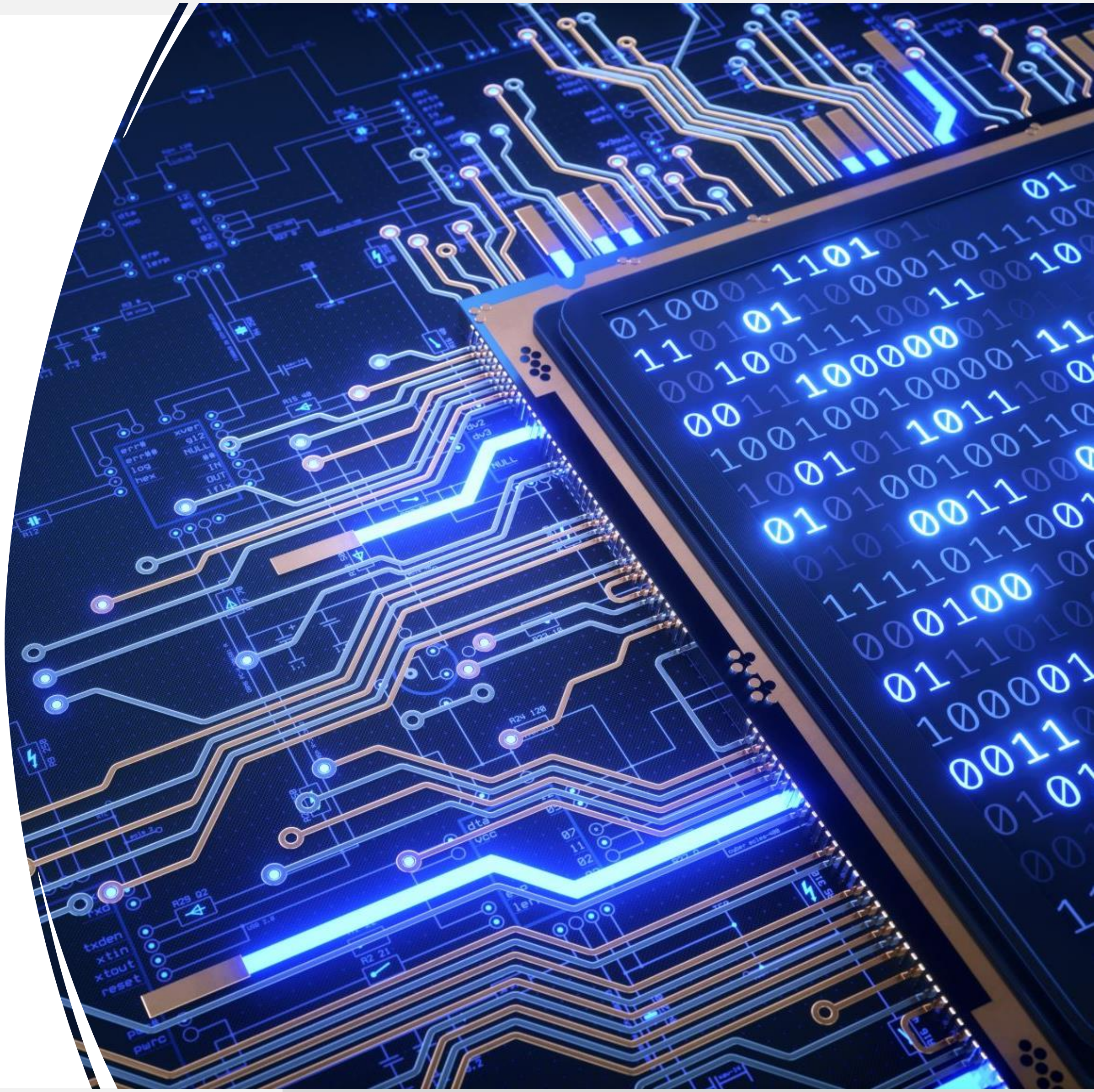
Features

- ✓Incredible library support: More than 3000 C++ libraries in Sourceforge
- ✓Classes, Inheritance, inline, default function arguments, virtual function, function overloading, references, operator overloading,

HISTORY

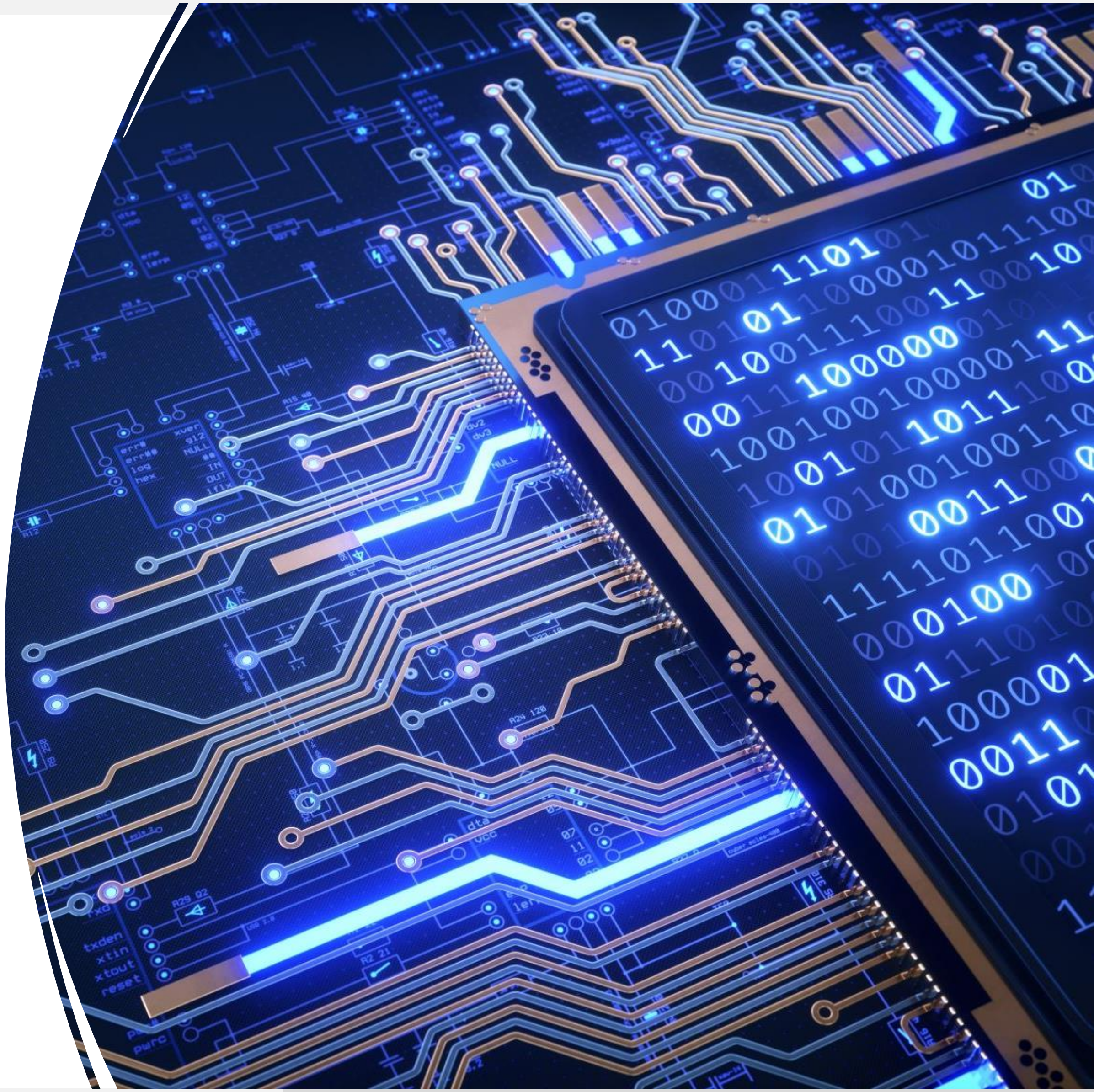
History

- 1979: Bjarne Stroustrup, Ph. D Thesis
- Worked with Simula 67 language (designed for simulations, a first OOP paradigm)
- Worked on "C with classes"
- Constructed a superset of C language
- Included classes, inheritance, default function arguments



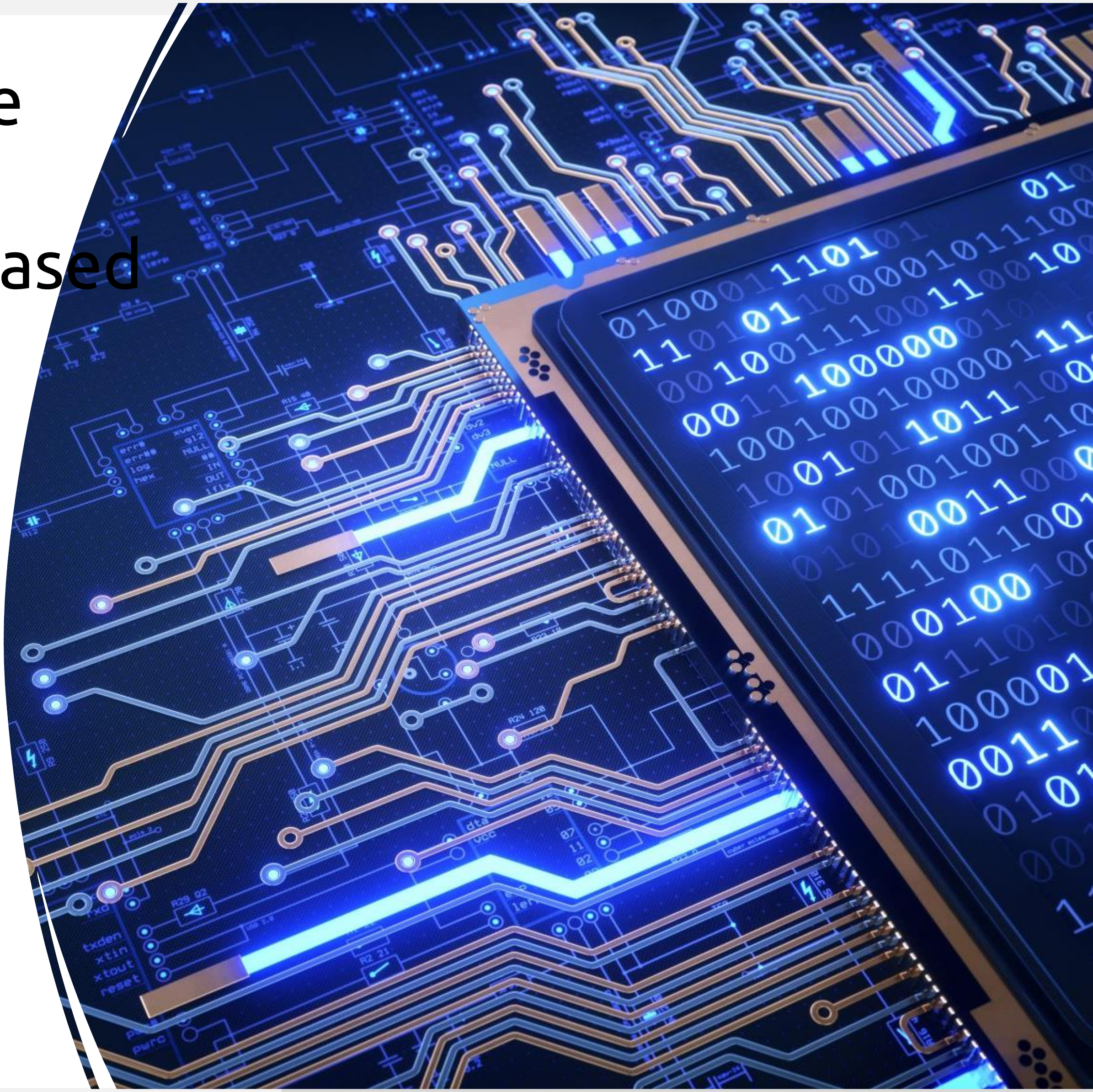
History

- First C with classes compiler: Cfront
- 1983: C with classes became C++
- ++ is an increment operator in C language to denote that many features added to C language
- 1985: The C++ Programming language by Stroustrup was published



History

- 1990: The Annotaed C++ Reference Manual was released
- 1990: Turbo C++ commercially released
- 1998: Standardized, C++ISO/IEC 14882:1998 or C++98
  - 2003: C++03
  - 2005: C++0x
  - 2011: C++11
  - 2014: C++14
  - 2017: C++17
  - 2020: C++20
  - 2023: C++23 (Dec)



APPLICATIONS

Applications

- Operating System Development
- Embedded systems
- Real-time systems
- Communication Systems

ubuntu

Applications of C++

Web and Internet Development

Education

Scientific and Numeric

Database

Networking

Robotics

GUIs

Gaming

Software Development

Business applications

AI & ML

Applications of C++

Google
facebook
Instagram
Pinterest
box
Dropbox
TiVo
comcast
at&t
SiriusXM SATELLITE RADIO
ebay
reddit
digg
Concur
hotels.com
salesforce
slideshare
Chef
openstack
vmware
The New York Times
BBC
THE HUFFINGTON POST
The Weather Channel
NHS
GOV.UK
NAVTEQ
IBM
Instagram
NASA
YouTube
Quora
COMPANIES USING PYTHON
UBER
NETFLIX
mozilla
lyft
C.
des
ZUMBA
S
Utrip
Tictail
h.
CP
!t
HOMEJOY
OpenX
kippt
custora
Eki
unc
L2
exelate
DataHero
eyereturn
divio

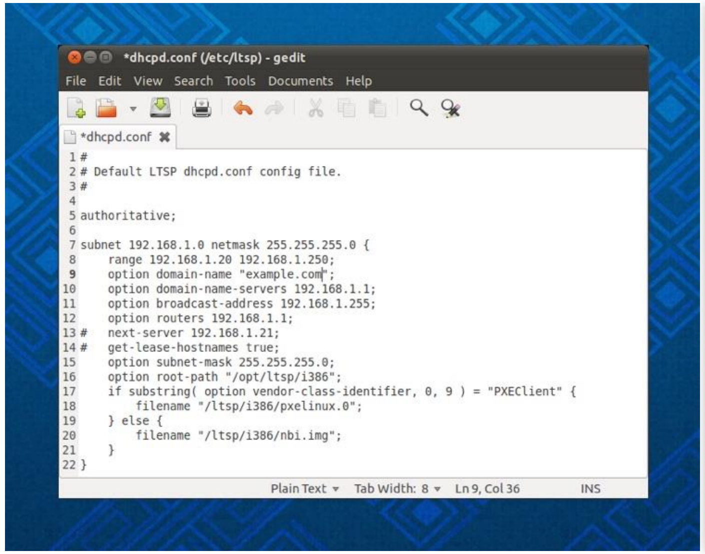
C++ Development Stages

STAGES

- Editing
- Preprocessing
- Compiling
- Linking
- Loading
- Executing
- Debugging

Phase-1: Editing

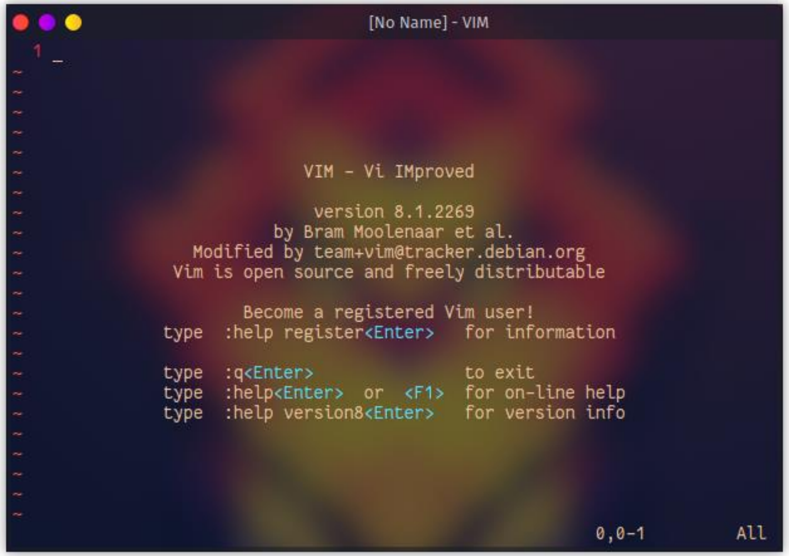
- Editing or creating a C++ file
- gedit, vim, emacs
- Eclipse, MSVC, geany, DevC
- Store the program on secondary hard disk
- Save the file name with an extension .cpp


*dhcpd.conf (/etc/ltsp) - gedit
File Edit View Search Tools Documents Help
*dhcpd.conf
Plain Text
Tab Width: 8
Ln 9, Col 36
INS

```
#
# Default LTSP dhcpd.conf config file.
#

authoritative;

subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.20 192.168.1.250;
    option domain-name "example.com";
    option domain-name-servers 192.168.1.1;
    option broadcast-address 192.168.1.255;
    option routers 192.168.1.1;
#   next-server 192.168.1.21;
#   get-lease-hostnames true;
    option subnet-mask 255.255.255.0;
    option root-path "/opt/ltsp/i386";
    if substring( option vendor-class-identifier, 0, 9 ) = "PXEClient" {
        filename "/ltsp/i386/pxelinux.0";
    } else {
        filename "/ltsp/i386/nbi.img";
    }
}
```



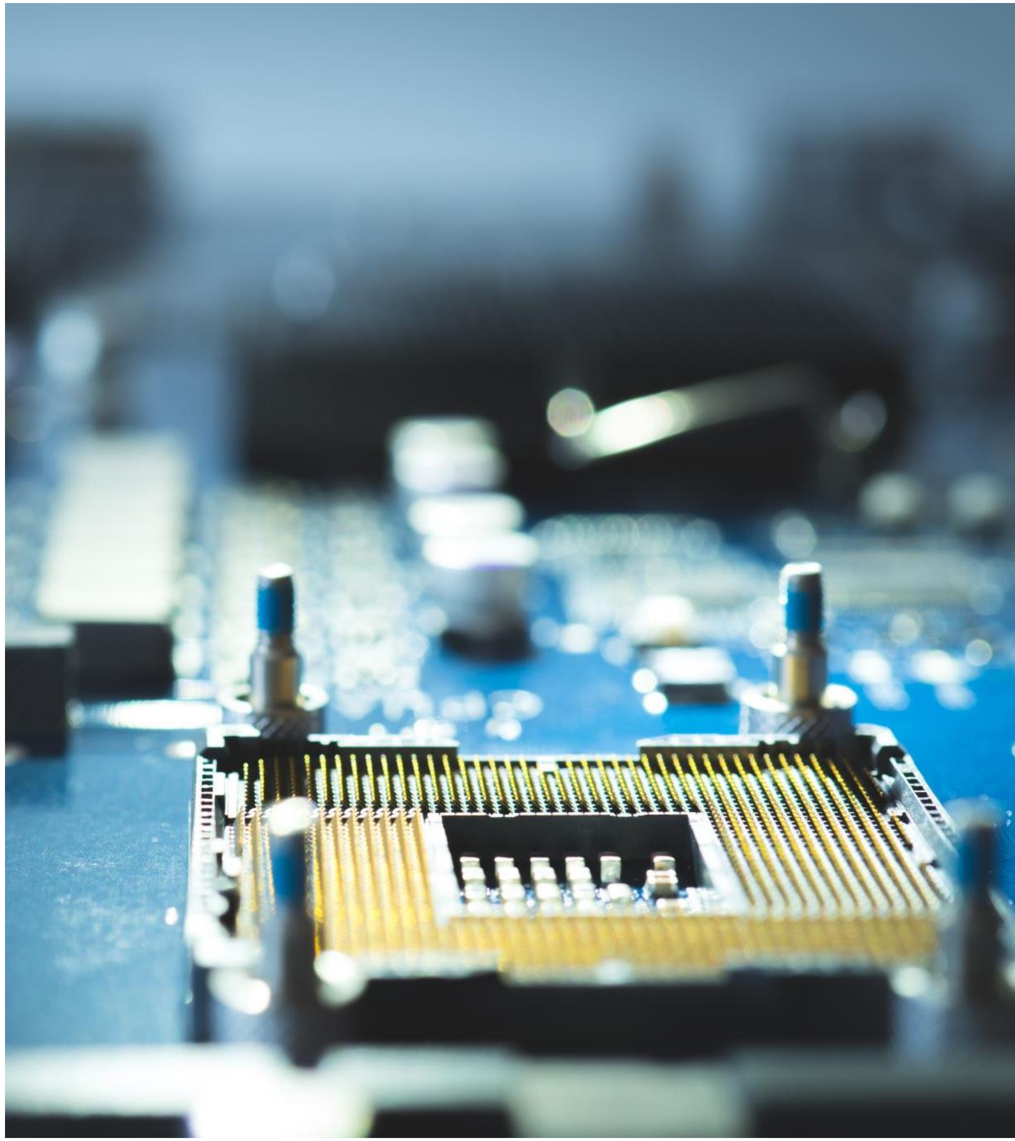
```
                 VIM - Vi IMproved

                 version 8.1.2269
             by Bram Moolenaar et al.
      Modified by team+vim@tracker.debian.org
    Vim is open source and freely distributable

             Become a registered Vim user!
   type  :help register<Enter>   for information

   type  :q<Enter>               to exit
   type  :help<Enter>  or  <F1>  for on-line help
   type  :help version8<Enter>   for version info

                                          0,0-1         All
```



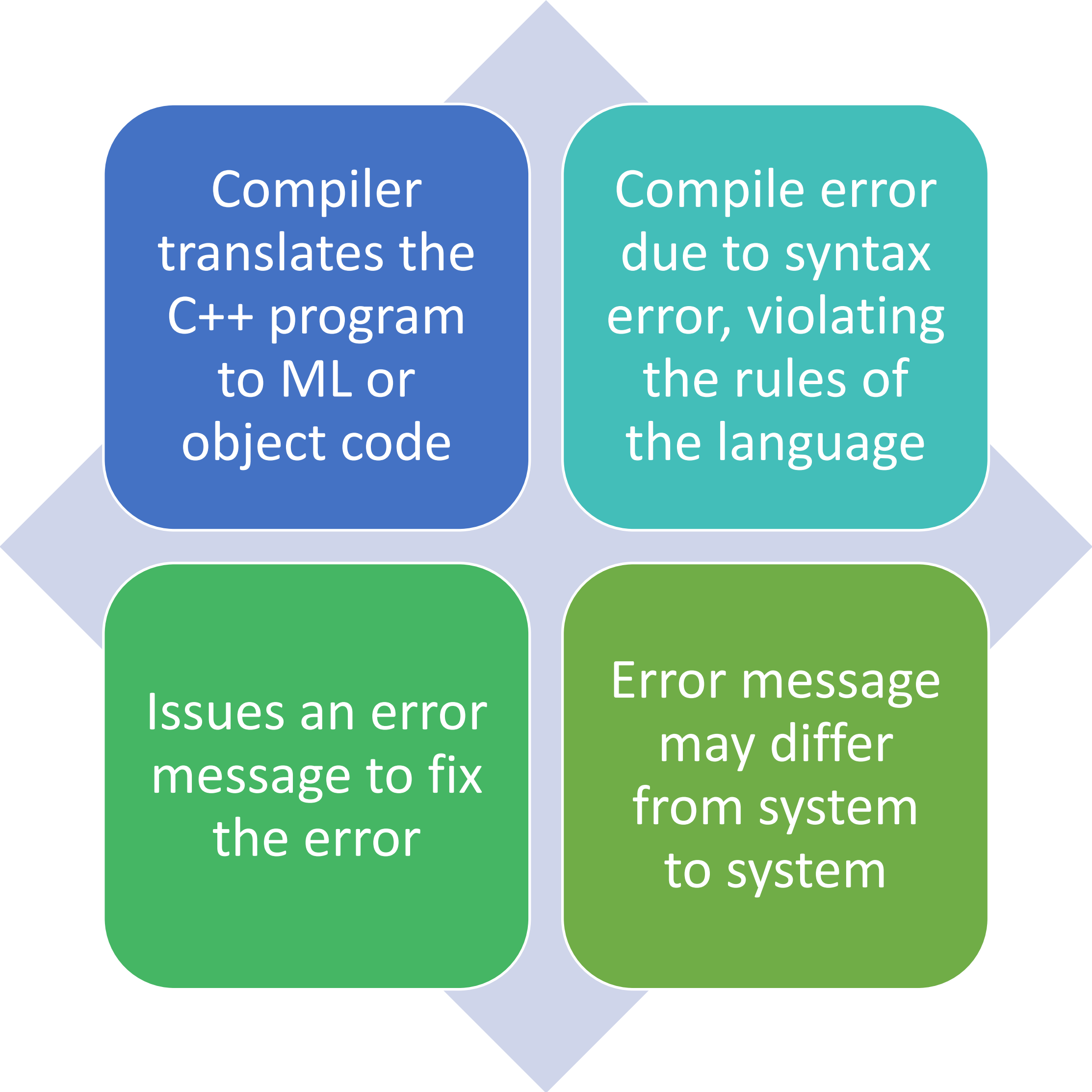
Phase-2: Preprocessing

While the compiler translates the C++ program to ML or object code

Including other files for compilation

Preprocessor program obeys preprocessor directives

Phase-3: Compiling



Compiler translates the C++ program to ML or object code

Compile error due to syntax error, violating the rules of the language

Issues an error message to fix the error

Error message may differ from system to system

Phase-4: Linking

- References to functions defined elsewhere
- Object code provided by many programmers are linked
- Object code produced by the compiler has holes due to missing parts
- Links the object code for the missing functions
- Produces an executable image

Phase-4: Linking

1. Usually, Phase 2,3 and 4 can be done by a single command for smaller program
2. g++ FileName.cpp
3. It compiles, links and creates an executable a.out

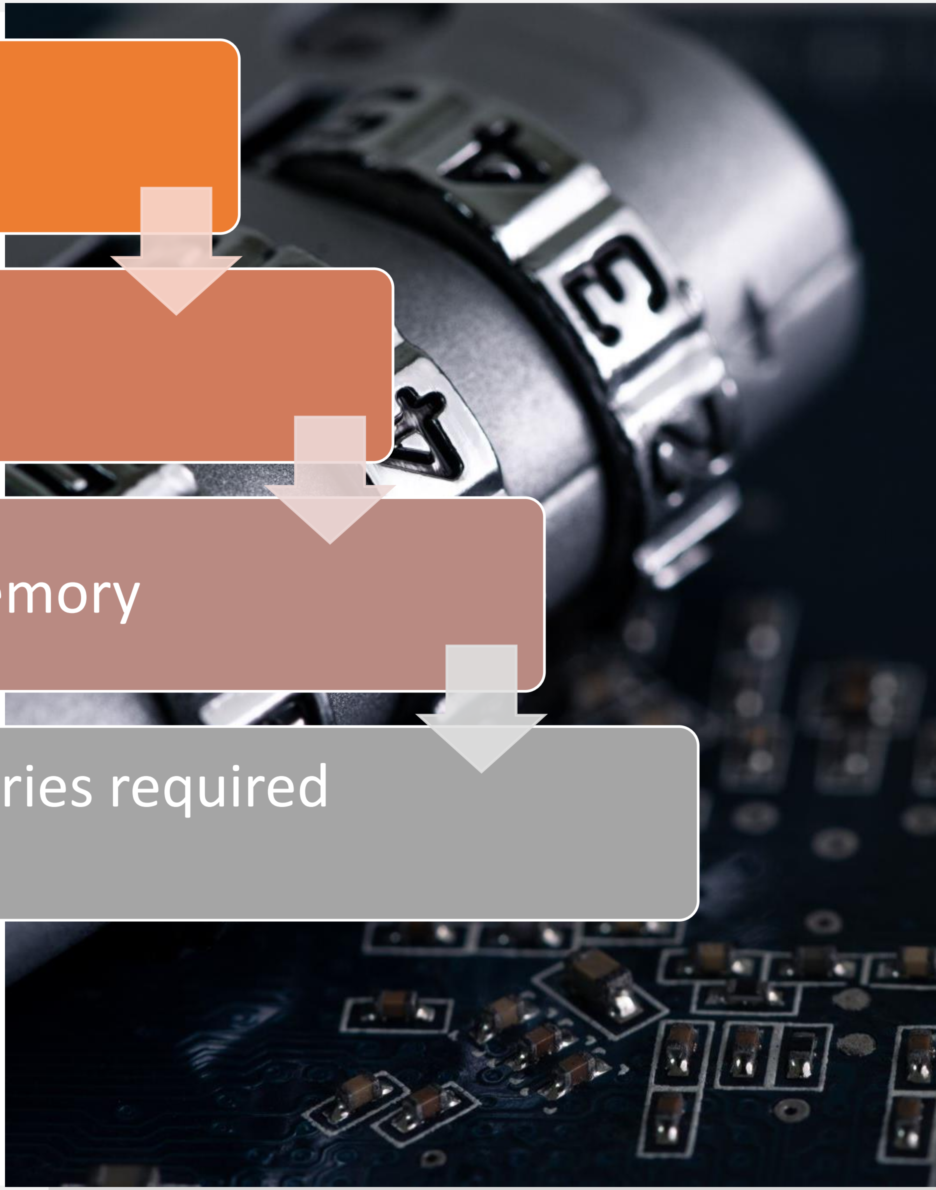
Phase-5: Loading

* Before Execution

* Must be placed in memory first

* Loader loads executable image from disk to memory

* Additional components from shared libraries required for the program



Phase-6: Executing

Under the control of CPU

Executes one instruction at a time

To load and execute, ./a.out

Provides necessary input from stdin(a keyboard)

Produces output to stdout(a computer screen)

stderr: to display the error to the screen

Debugging

Not necessary to produce error free code in first attempt

Syntax error, runtime error, segmentation fault

Make necessary corrections depending on the code and repeat all steps

GNU G++

GNU C++

- ✓ GNU is an operating system that is free software, contains no Unix code
- ✓ Contains many GNU packages
- ✓ GNU's Not Unix!. It is a recursive acronym
- ✓ Its design is Unix-like, but differ from Unix

GNU Compile Collection

- ✓Contains collection of compilers

* C, C++, Objective-C, Fortran, Ada, Go,

Further Reading

https://devdocs.io/cpp/
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4713.pdf
http://www.cplusplus.com/