# LLVM Architecture Overview

Software Systems Architecture

- José Francisco Veiga (up202108753)
- Marco Vilas Boas (up202108774)
- Pedro Lima (up202108806)
- Pedro Januário (up202108768)
- Pedro Marcelino (up202108754)

# What is LLVM?

- Open-source compiler infrastructure
- Provides modular and reusable compiler and toolchain technologies

# Architecture Summary

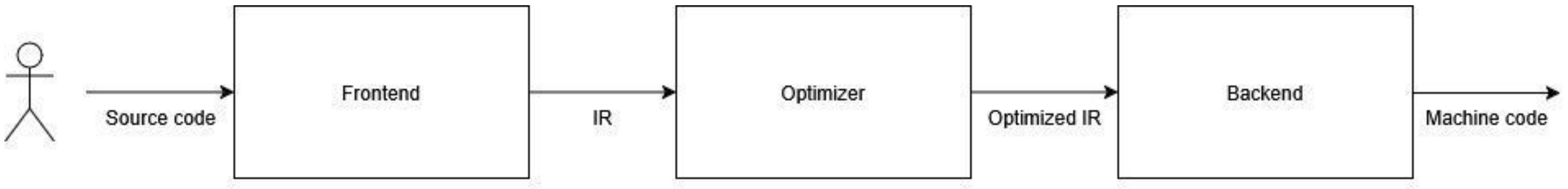
**LLVM** as a **modular**, **library-based design** for a **compiler**, instead of the traditional monolith.

Allows for **easy integration** of new source languages and target architectures, as well as **reusability of modules** in new projects.

**LLVM IR** is a key secret for its success: a **self-contained**, **first-class** language that provides a common interface for all compiler components.

Many advantages over traditional architecture (e.g. GCC) because of customizability and reusability.

# 3-Phase Compiler Architecture



*IR = Internal Representation*

**Frontend**

- Parses and validates input code
- Translates to LLVM Intermediate Representation (IR)

**Optimizer**

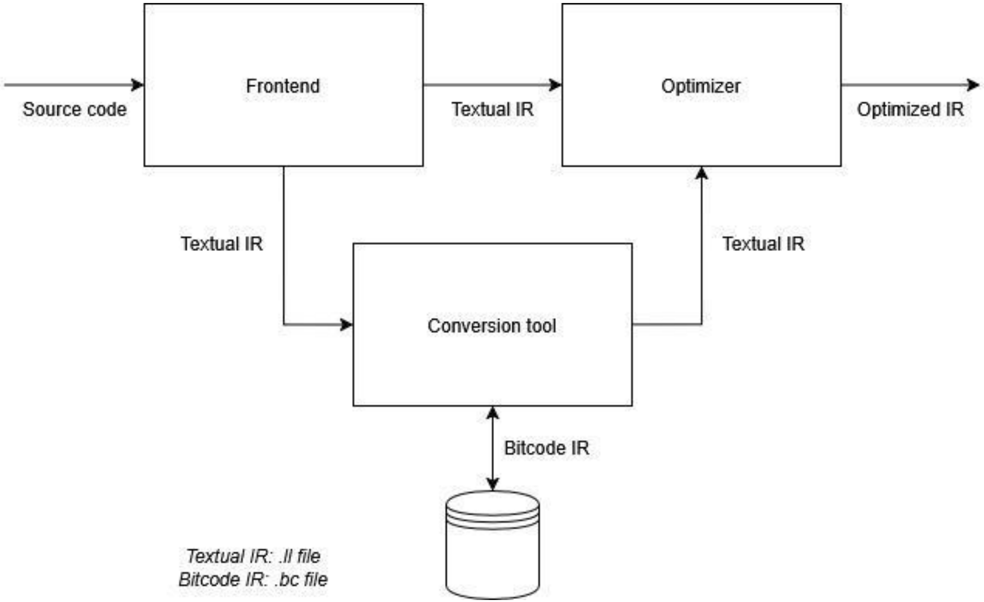
- Performs analysis and optimization passes
- Improves performance and efficiency

**Backend**

- Generates native machine code from IR

# IR Transactions

1. Frontend generates IR
2. Optimizer refines and improves IR
3. IR can be saved as bitcode for reuse



*Textual IR: .ll file*
*Bitcode IR: .bc file*

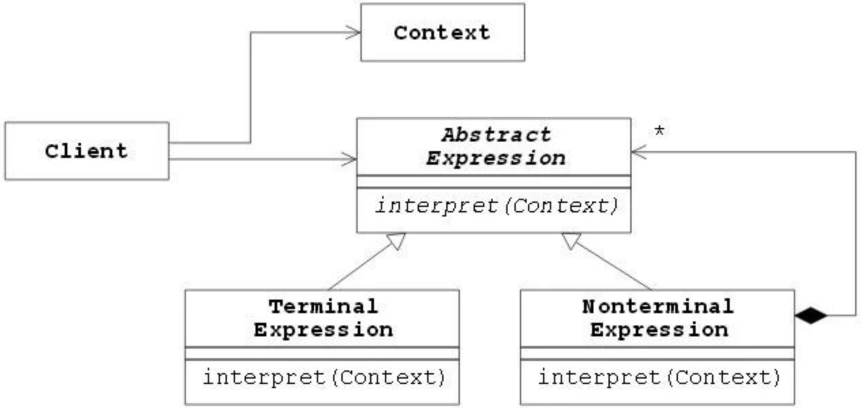
# Architectural Patterns

- Interpreter
- Pipes and Filters
- Layers Patterns
- Broker Pattern
- Encapsulated Implementation

# Interpreter

Defines a way to interpret and evaluate language grammar or expressions.

1. High-level language to IR
2. IR to machine code

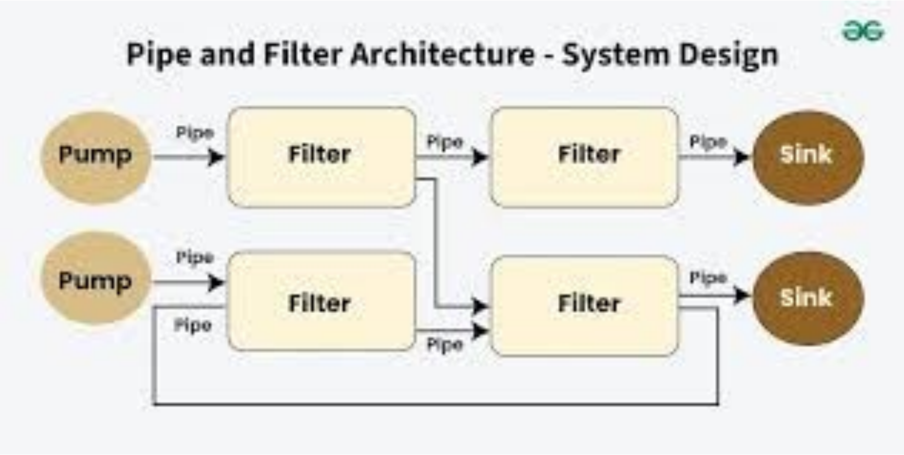


# Pipes and Filters

- Data flows through pipes and transformed by filters

- Programming languages (pipes) flow through frontend, optimizer and backend (filters)

- Allows modular and sequential data processing



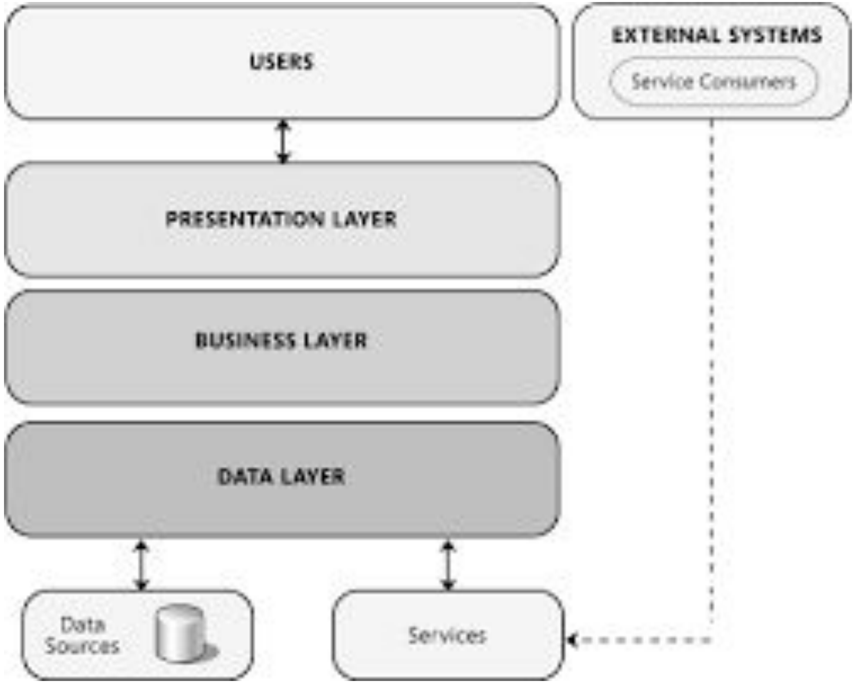
# Layers Patterns

- System divided into clear layers interacting only with adjacent layers

---

- Frontend
- Optimizer
- Backend

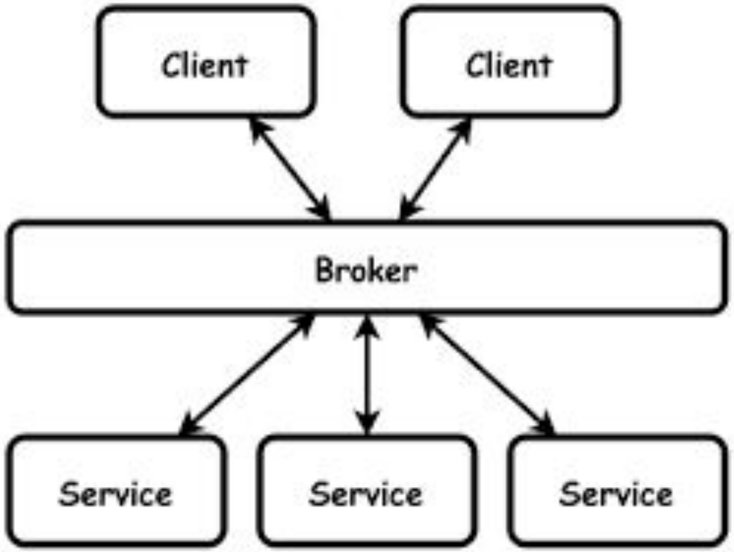
- IR acts as interface
- + Maintainable
- + Scalable



# Broker Pattern

- Mediator facilitates communication between components

- Optimizer acts as a broker between various frontends and backends
- Modular enough to allow distributed execution

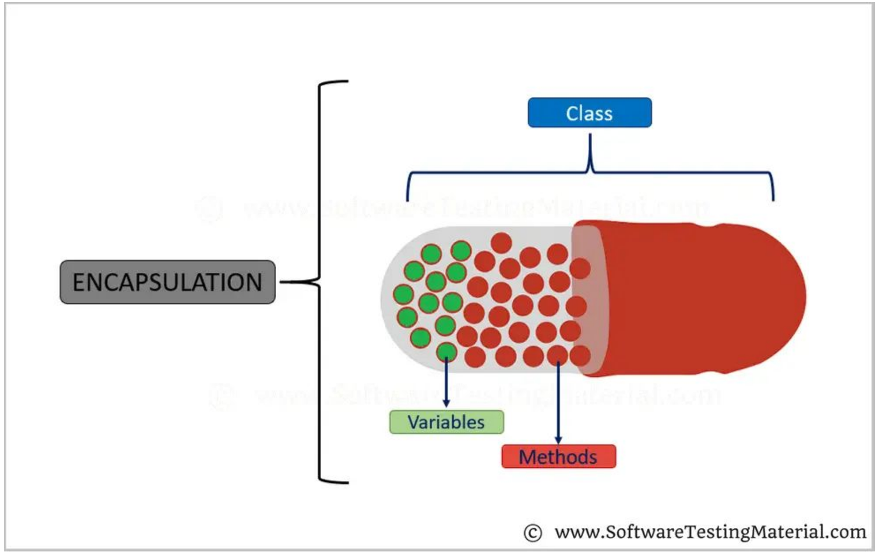


# Encapsulated Implementation

Hide implementation details behind interfaces

- Pass classes for optimizations
- Template-based approach

- + Flexibility
- + Resilience



# Quality Attributes

## Modularity

- Reusable  libraries with well-defined interfaces

## Reusability

- Supports various languages and architectures
- Neutral IR format

## Performance

- Powerful Optimization Pipeline
- Link-time and install-time optimizations

# Conclusion

- LLVM is a prime example of a modular and efficient compiler architecture
- Multiple architectural patterns
- Focus on Adaptability, Modularity and Performance

↓

Globally used