Pirouette

Ethan Canton, Claire Huyck, Alex Kim, Julia Montuori, Yining Zhang

CSE 302/402 Experiential Learning/Research (Instructors: Lukasz Ziarek, Andrew Hirsch)

Choreographic Programming

Simplifies **distributed** and **concurrent** systems by defining the entire program behavior from the start.

- Specify **all** participants & their computations
- Define all **communication** patterns clearly
- Automatically generates code for each node/thread

Pirouette Language

Pirouette is a programming language that **eliminates deadlock** by defining a single, choreographed "master" program for the entire system.

- The master program describes **global behavior** across all computers.
- Every computer follows a **predefined script**, preventing miscommunications.
- **Guarantees deadlock freedom** by construction.

Overview of Semester

- Tested the following components with the OUnit2 testing module:
  - Type Checker
  - Core AST
  - CLI tools
  - Net Lexer/Parser
  - Code generation
  - Pretty Printer (Choreo)
- Added support for floating-point numbers
- Extended compiler with Foreign Function Interface (FFI) capabilities
- Implemented a Parser for netIR t(Intermediate Representation)
- Translated HasChor examples into pirouette
- Started to add variant types into Pirouette

Present & Future Goals

Short term Goals

- Finish adding variant types into Pirouette and test it
- Finish unit testing the rest of the existing code base
- Write documentation for undocumented code sections

Long term goals:

- Implement process polymorphism for Pirouette
- Rewrite existing code to create a more robust code base to avoid future tech debt

Deadlock Example

A deadlock happens when two or more processes wait on each other indefinitely, causing the system to freeze because no one can proceed.

A well-known example is the dining philosophers problem.

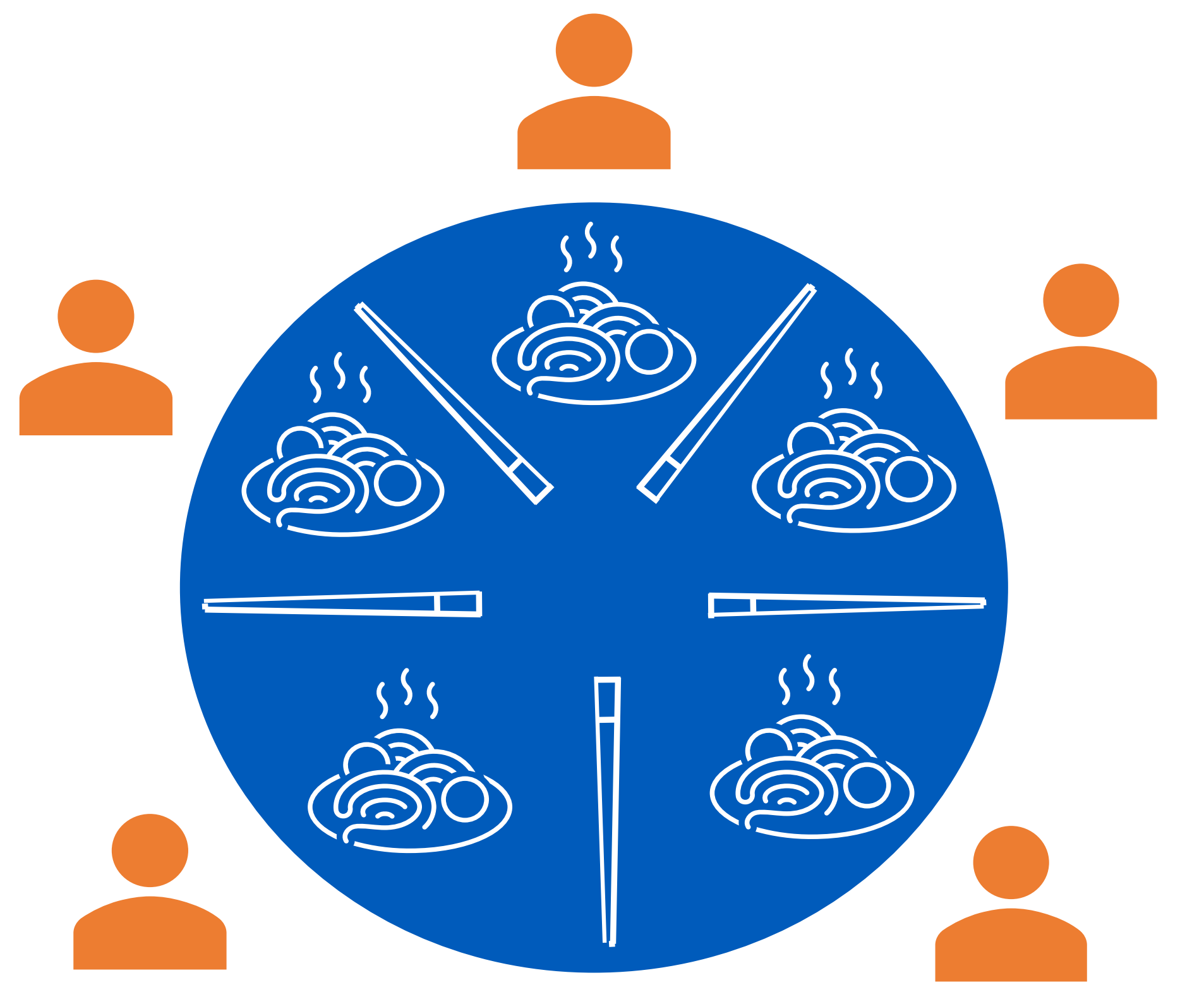


Figure 1: Dining philosophers problem [1]

- 5 philosophers are eating and thinking together
- Each of them requires both chopsticks on either side of their bowls to eat
- The 2 chopsticks on either side is only usable when their neighbors are thinking
- How can we ensure all philosophers think while not starving?

- P.S: In Pirouette, this wouldn't even exist!

Compiler Toolchain

A compiler is a tool that **translates** code **written** by a **human**, into something that a **computer** can read and **run**.

- It also checks for errors before running.
- It optimizes code for better performance.

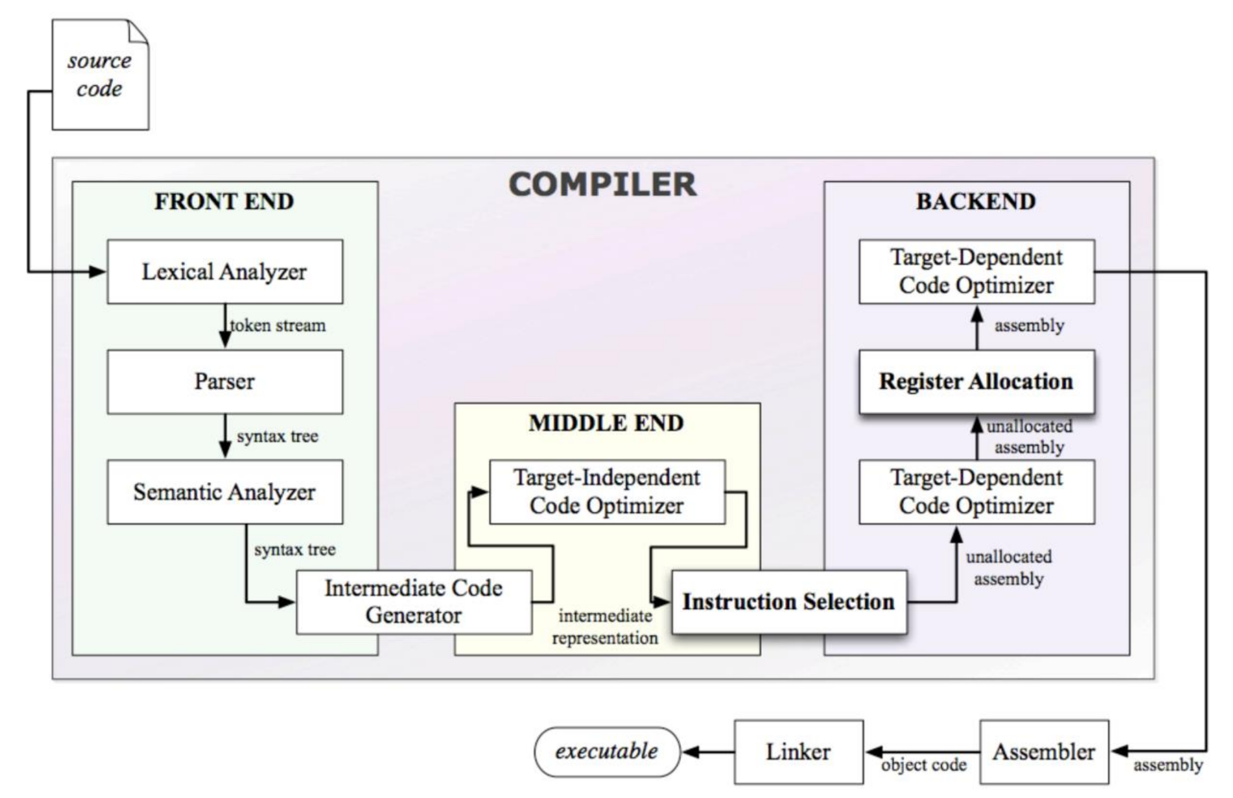


Figure 2: Path of a typical Compiler Toolchain [2]

- Most compilers produce program for a **single** computer
- But our compiler produces **multiple** programs that run across **different** machines
- All while automatically coordinating with each other

Multiple backends

Shared Memory:

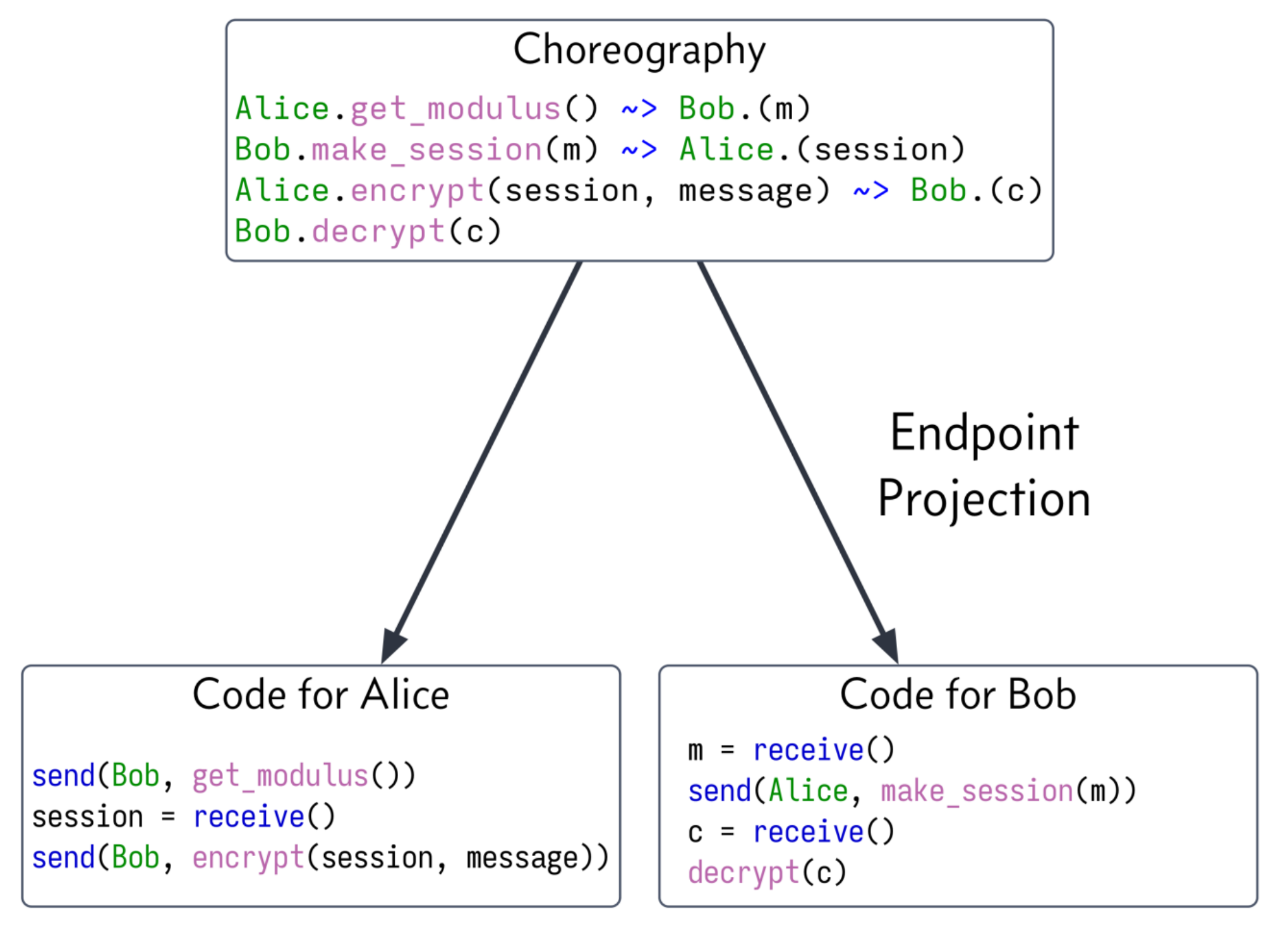
- Uses in-memory channels for efficient communication
- Works within same machine
- Provides low-latency performance

HTTP Backend:

- Enables cross-machine communication
- Works across network boundaries
- Suitable for distributed systems

Endpoint Projection

- Using **endpoint projection**, the compiler automatically splits into **local programs** for each computer
- Transforms a global program into localized behaviors
- Ensures endpoints can operate autonomously



```
Alice.get_modulus() ~> Bob.(m)
Bob.make_session(m) ~> Alice.(session)
Alice.encrypt(session, message) ~> Bob.(c)
Bob.decrypt(c)
```

```
send(Bob, get_modulus())
session = receive()
send(Bob, encrypt(session, message))
```

```
m = receive()
send(Alice, make_session(m))
c = receive()
decrypt(c)
```

Figure 3: Example of Endpoint Projection [3]

Conclusion

Current choreographic programming libraries are difficult to use or understand. Pirouette on the other hand prioritizes simplicity for the programmer while maintaining all the benefits of choreographic programming.

References