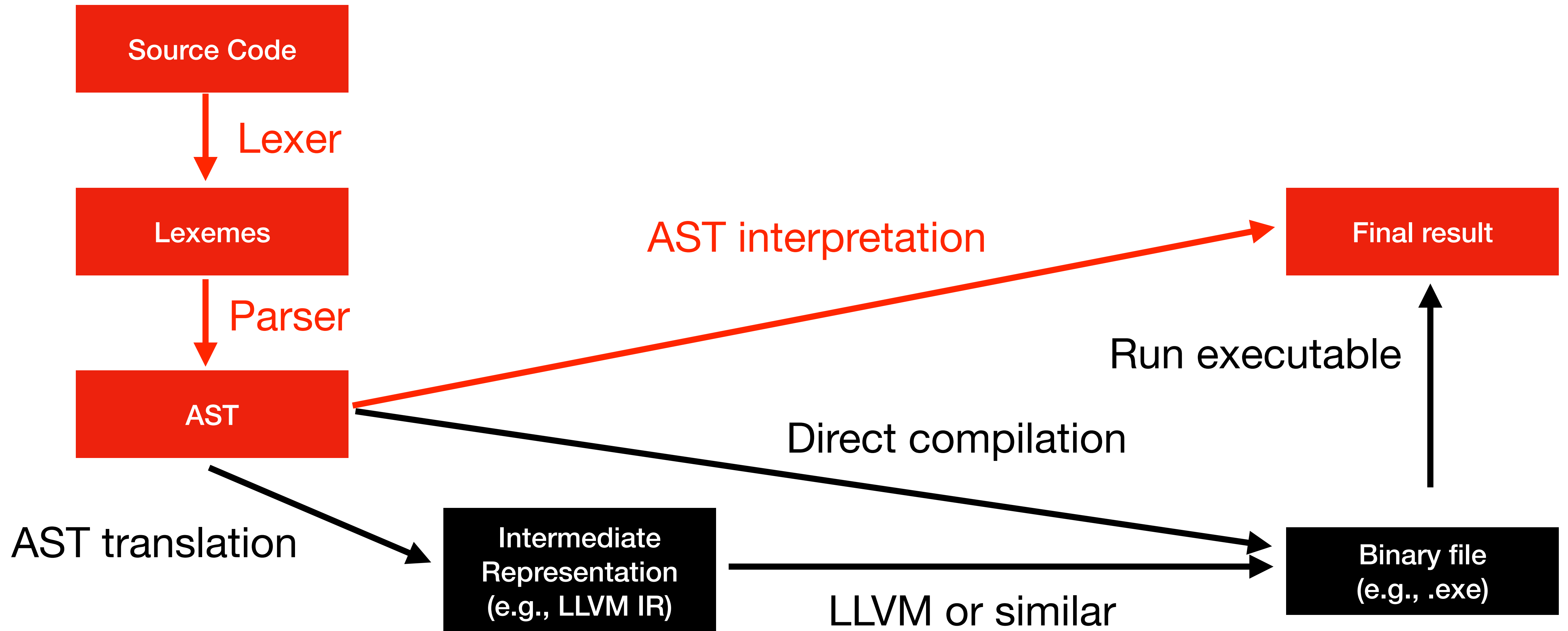


Process overview - review



Running the steps in sequence

```
def interpret(source_code, verbose=False):
    try:
        lexemes = lexer.lex(source_code) ← Lex
        if verbose:
            print("Lexemes:")
            print(lexemes)
        try:
            ast = parser.parse(lexemes) ← Parse
            if verbose:
                print("AST:")
                print(ast)
            try:
                return process_ast_exp(ast) ← Interpret
            except Exception as e:
                if verbose:
                    print("Encountered interpreting error: {}".format(repr(e)))
                    print("Returning None")
                return None
        except Exception as e:
            if verbose:
                print("Encountered parsing error: {}".format(repr(e)))
                print("Returning None")
            return None
    except Exception as e:
        if verbose:
            print("Encountered lexing error: {}".format(repr(e)))
            print("Returning None")
        return None
```

Error handling

The diagram illustrates the sequence of steps in a compiler: Lex, Parse, and Interpret. It also shows how errors are handled. The code defines a function `interpret` that takes `source_code` and `verbose` as arguments. It uses a `try` block to handle exceptions. The `try` block contains three nested `try` blocks. The first `try` block handles the `lexer.lex` call, which is labeled **Lex**. The second `try` block handles the `parser.parse` call, which is labeled **Parse**. The third `try` block handles the `process_ast_exp` call, which is labeled **Interpret**. The `except` blocks handle exceptions from these calls. The `except` block for `process_ast_exp` is labeled **Error handling**. Three arrows point from the **Error handling** label to the three `except` blocks, indicating that the error handling logic is shared across all three steps.