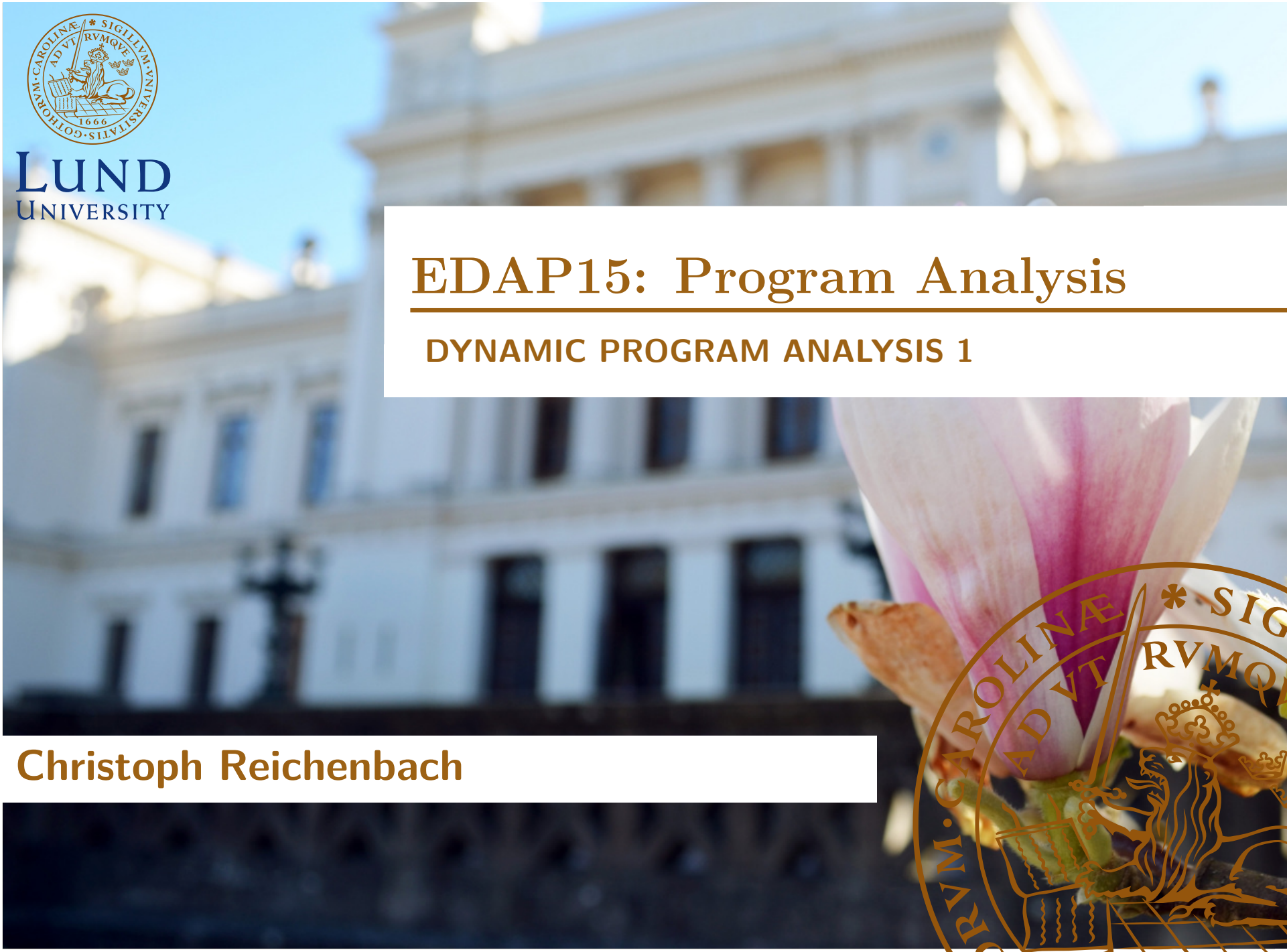


LUND UNIVERSITY

EDAP15: Program Analysis

DYNAMIC PROGRAM ANALYSIS 1

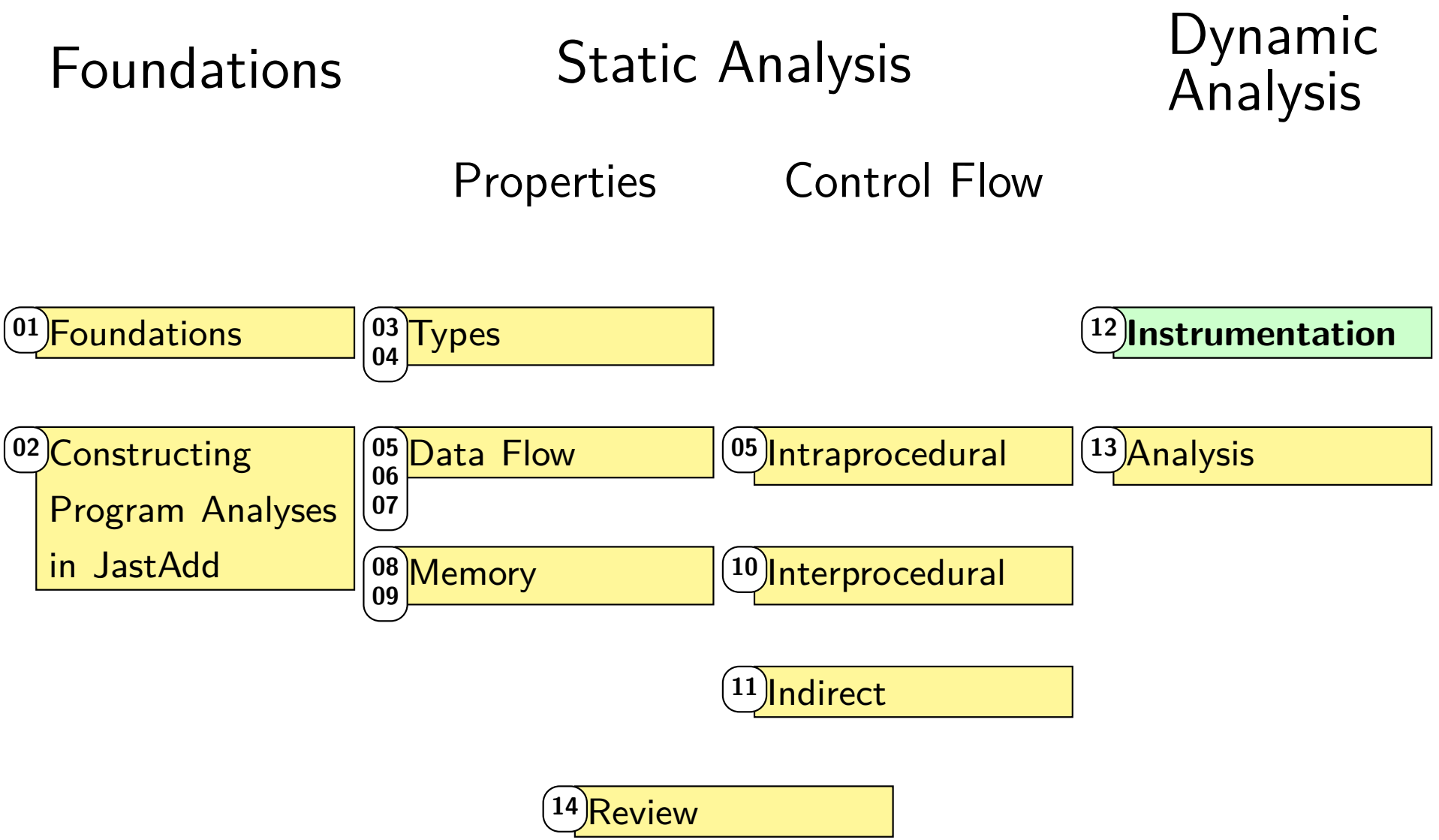
Christoph Reichenbach

Welcome back!

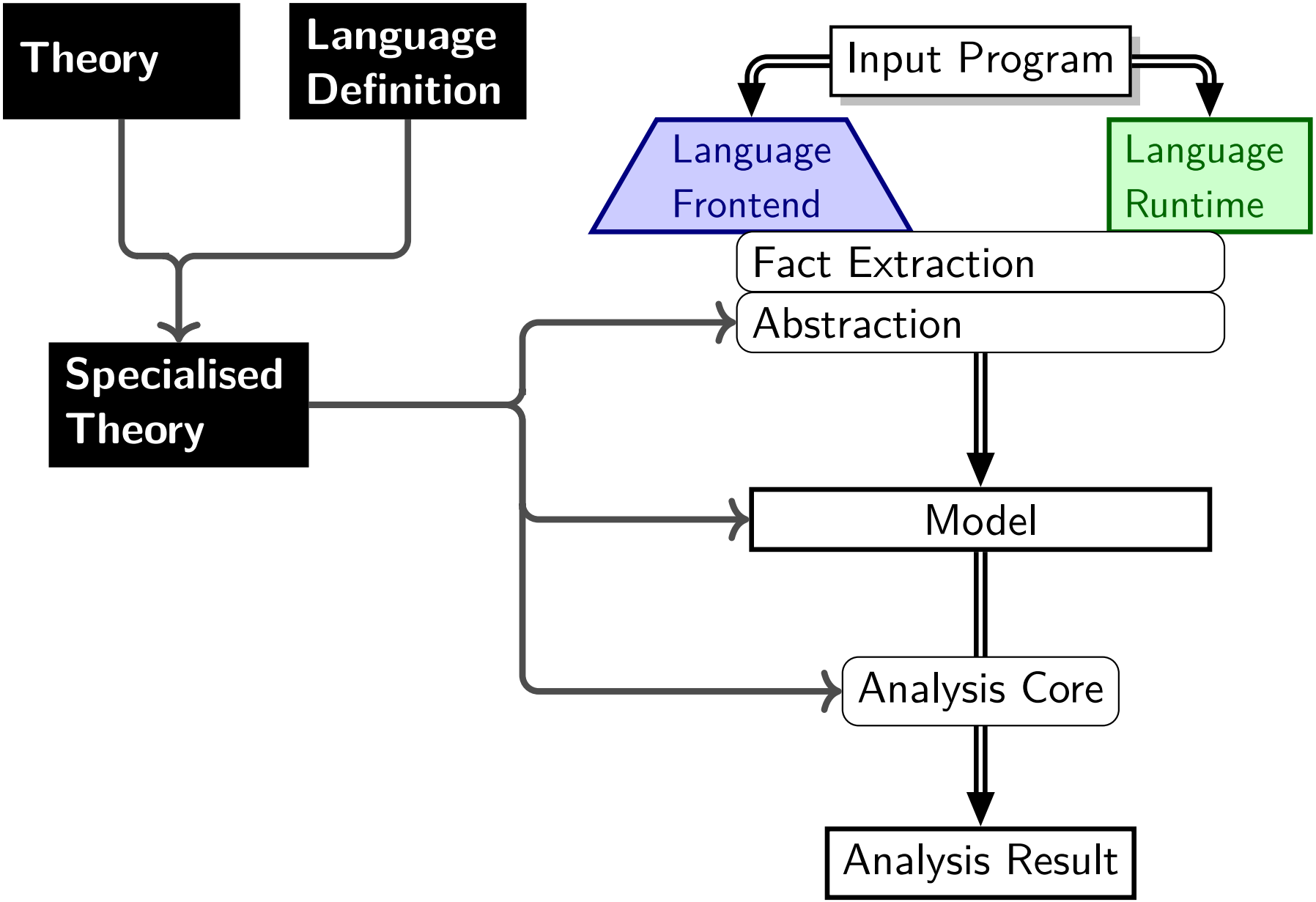
- Lab 3: Fixes available upstream (git) for Task 1
  - `nullReport()` → `nullReports()` in task description and code-prober script
  - If you followed only the `.jrag` / `.java` code, you might not have noticed the discrepancy
  - Fixes merged in if you hadn't pushed any updates yet

Questions?

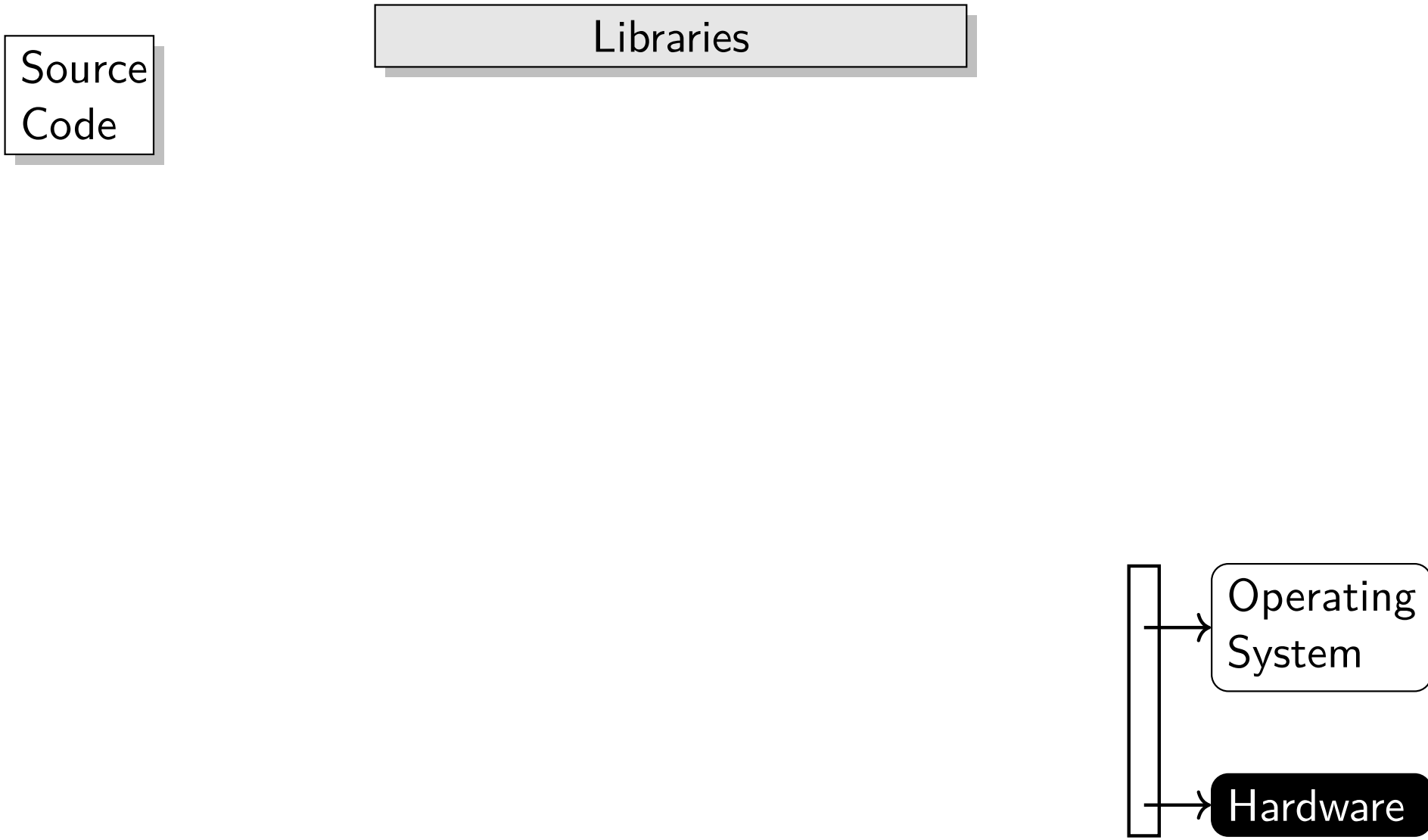
Lecture Overview



Building a Program Analysis



Program Execution Pipeline

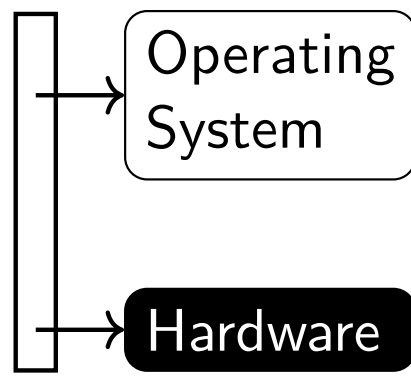


Program Execution Pipeline

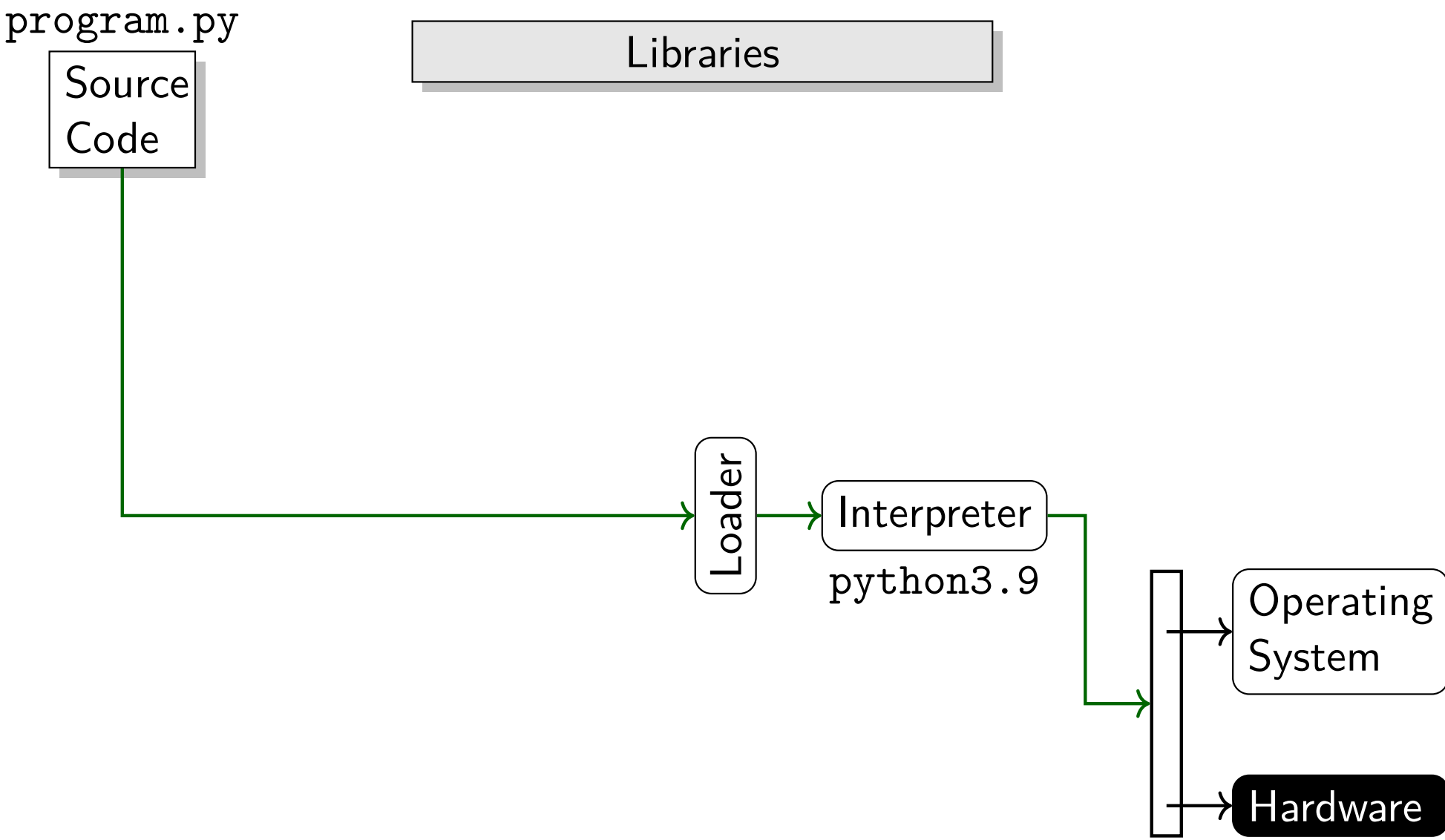
`program.py`

Source Code

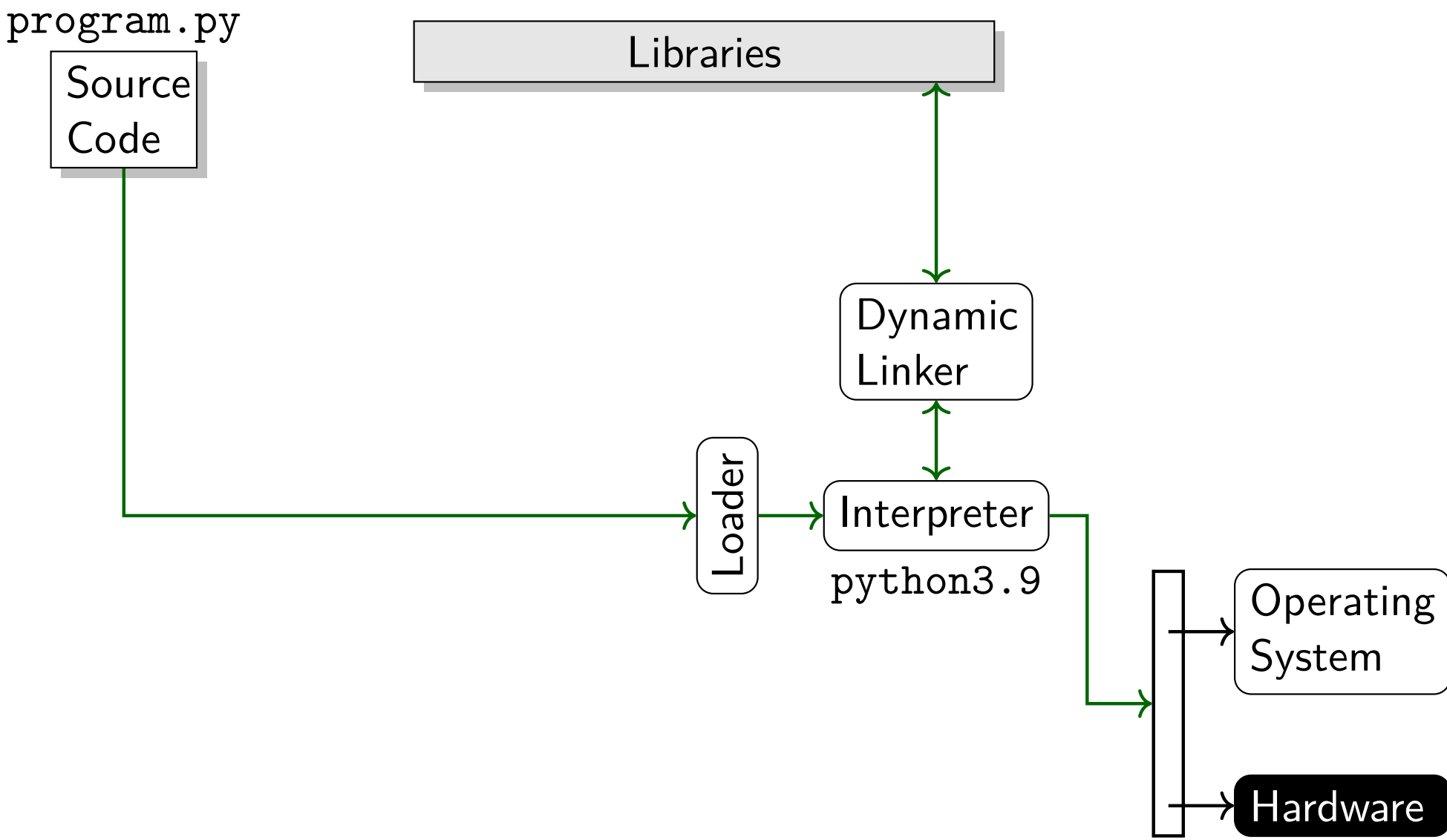
Libraries



Program Execution Pipeline



Program Execution Pipeline

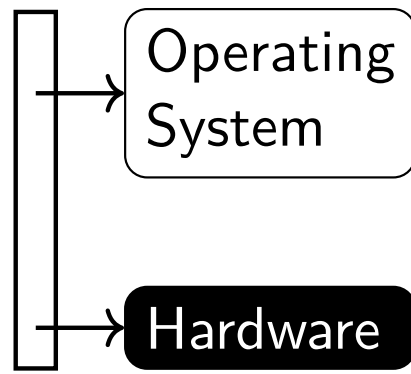


Program Execution Pipeline

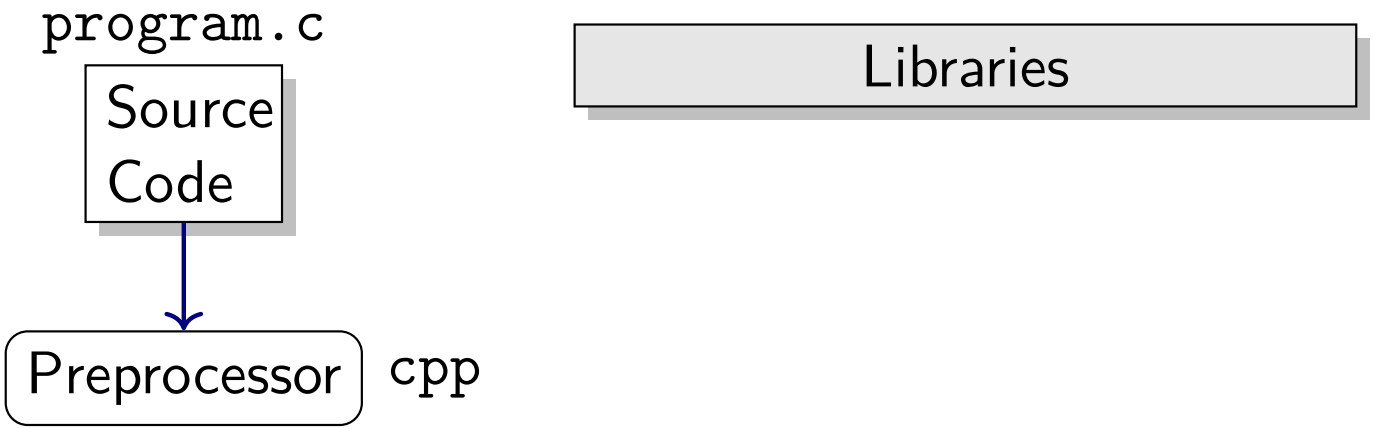
`program.c`

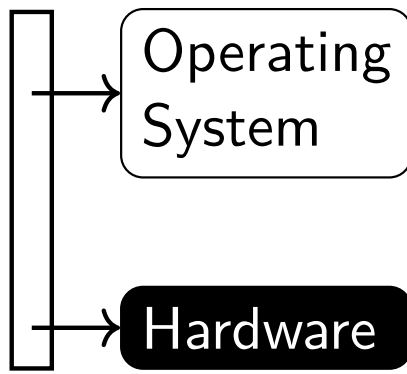
Source Code

Libraries

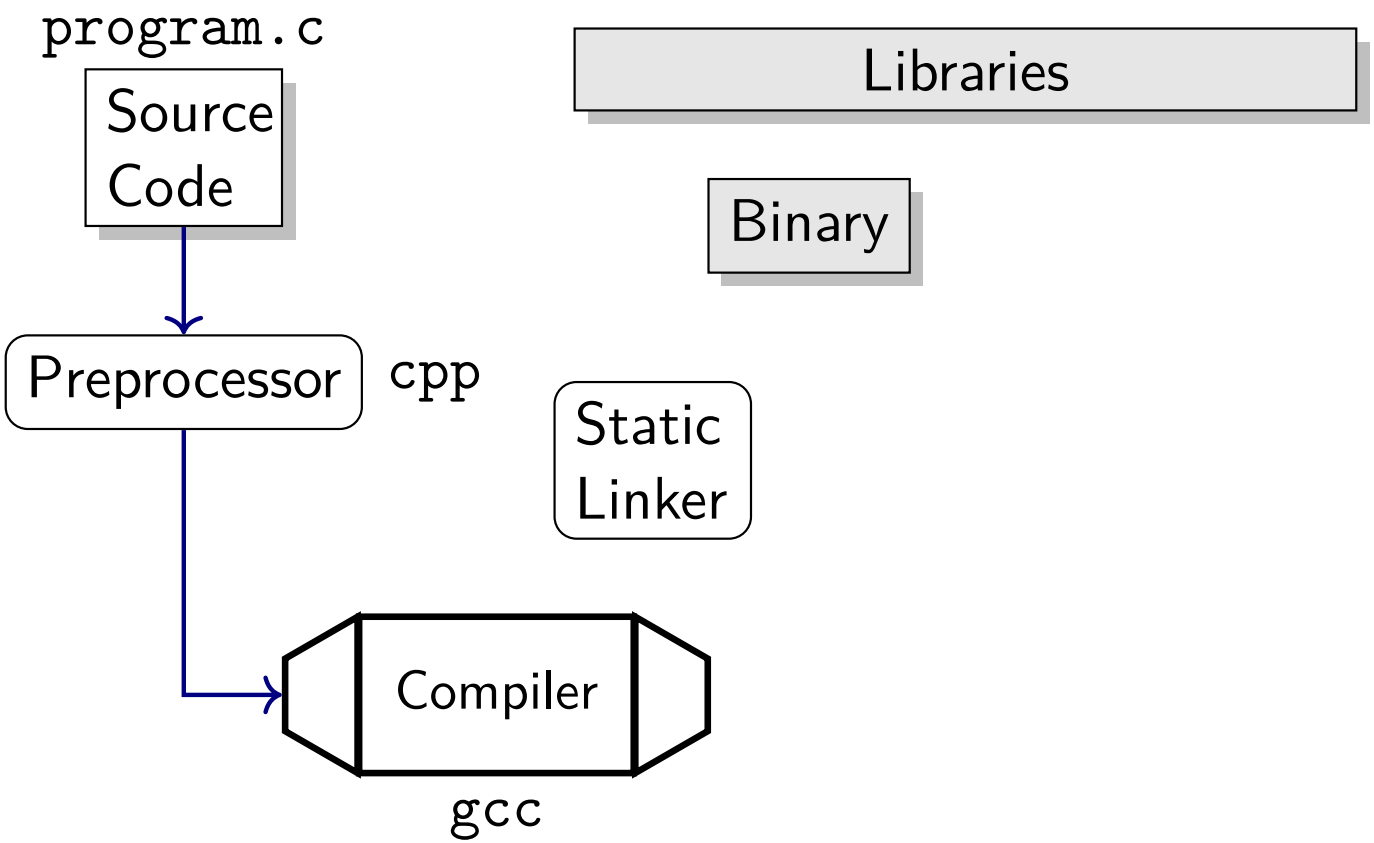


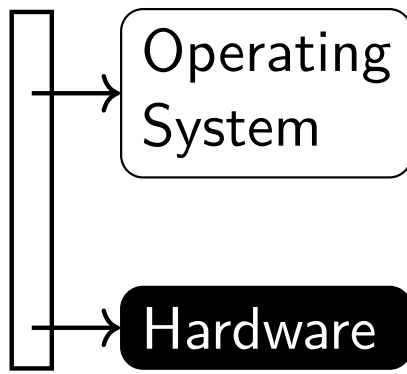
Program Execution Pipeline



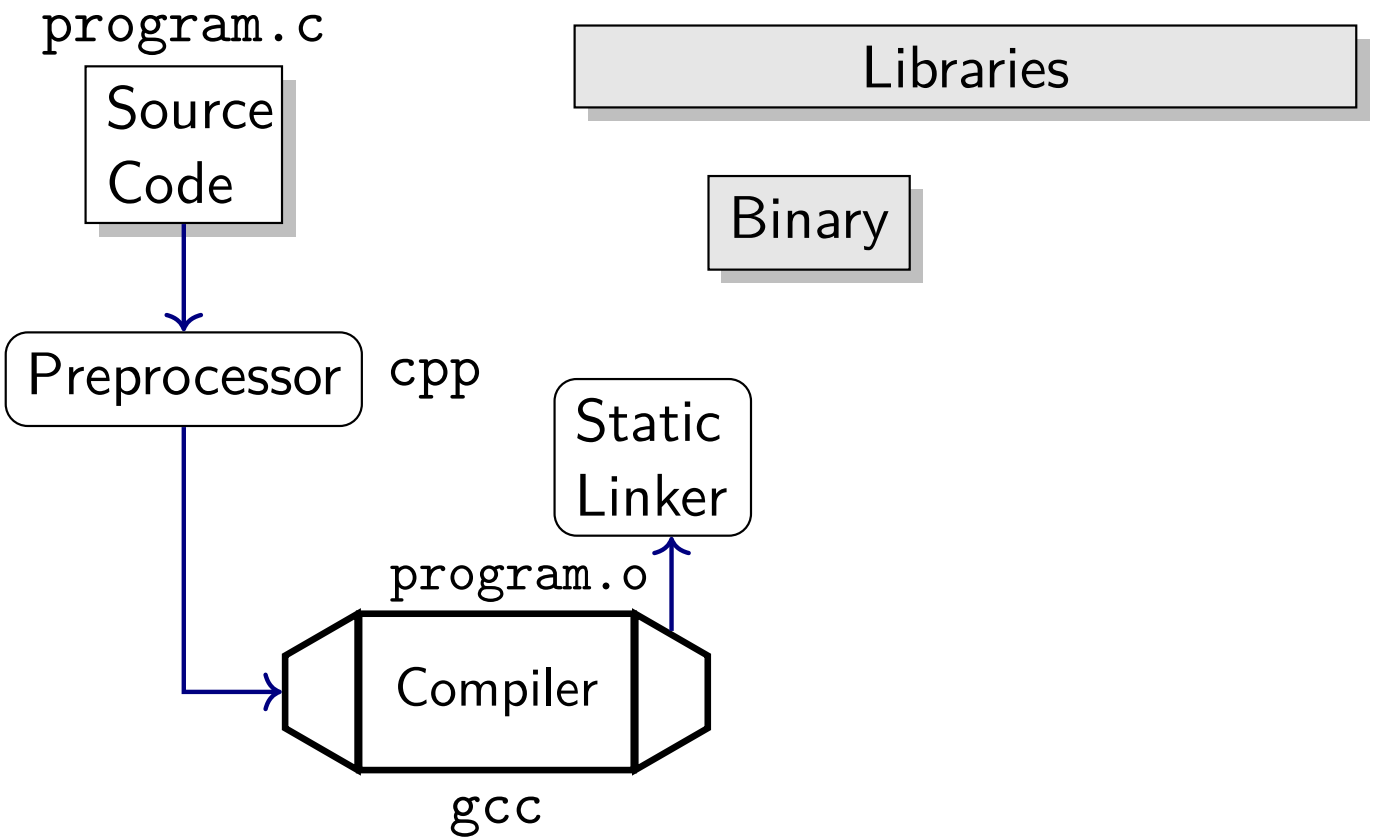


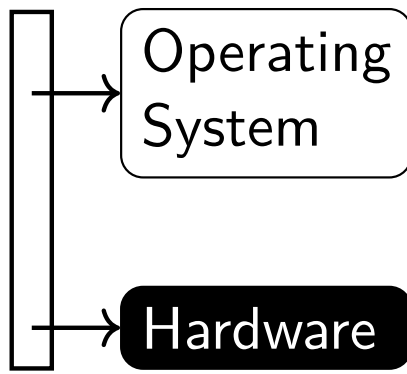
Program Execution Pipeline



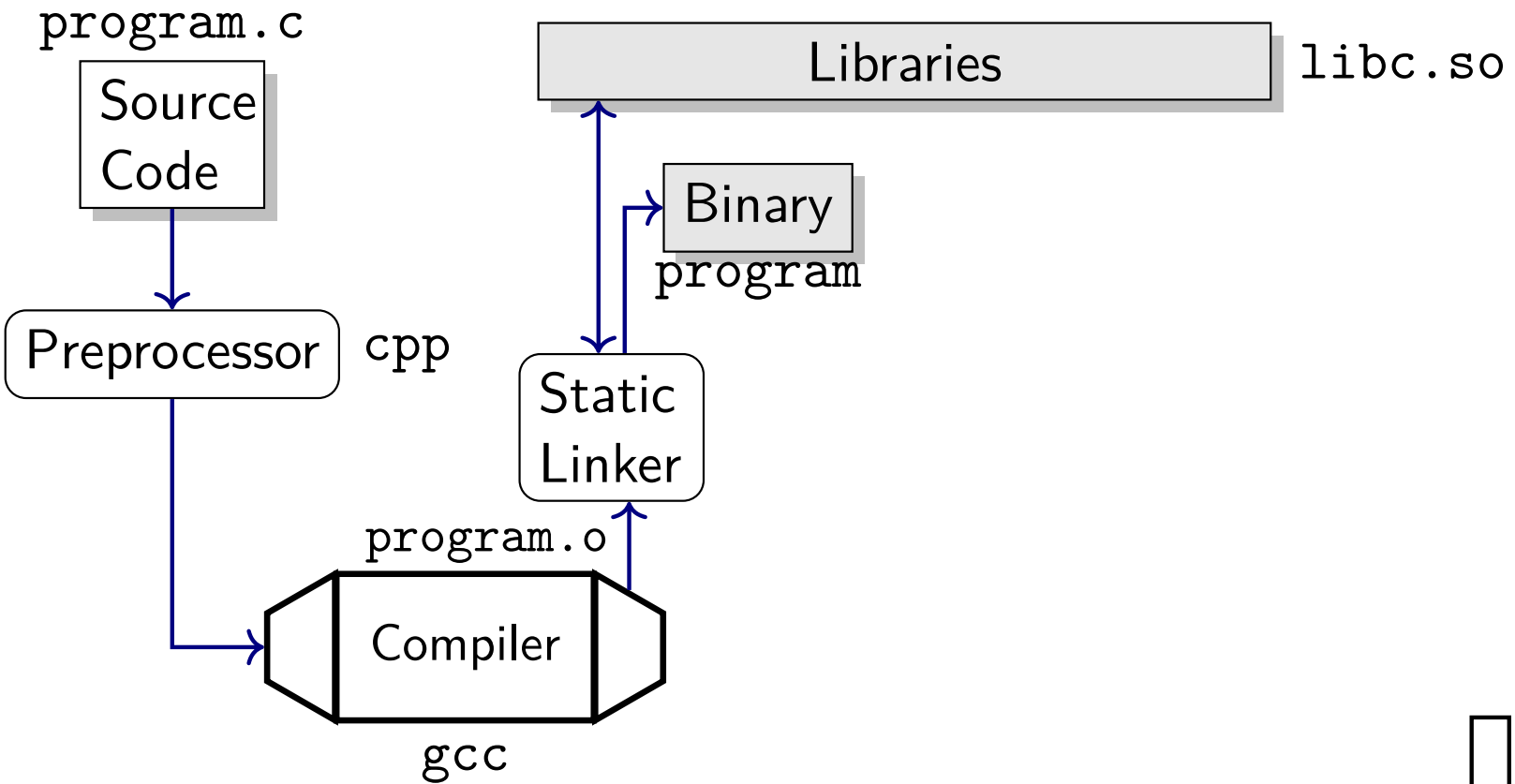


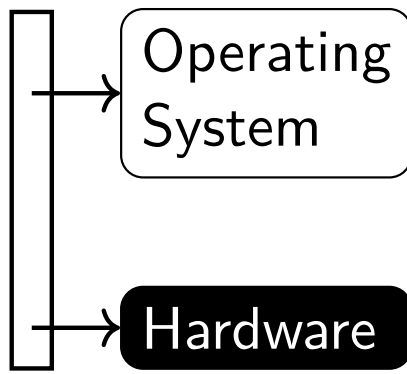
Program Execution Pipeline



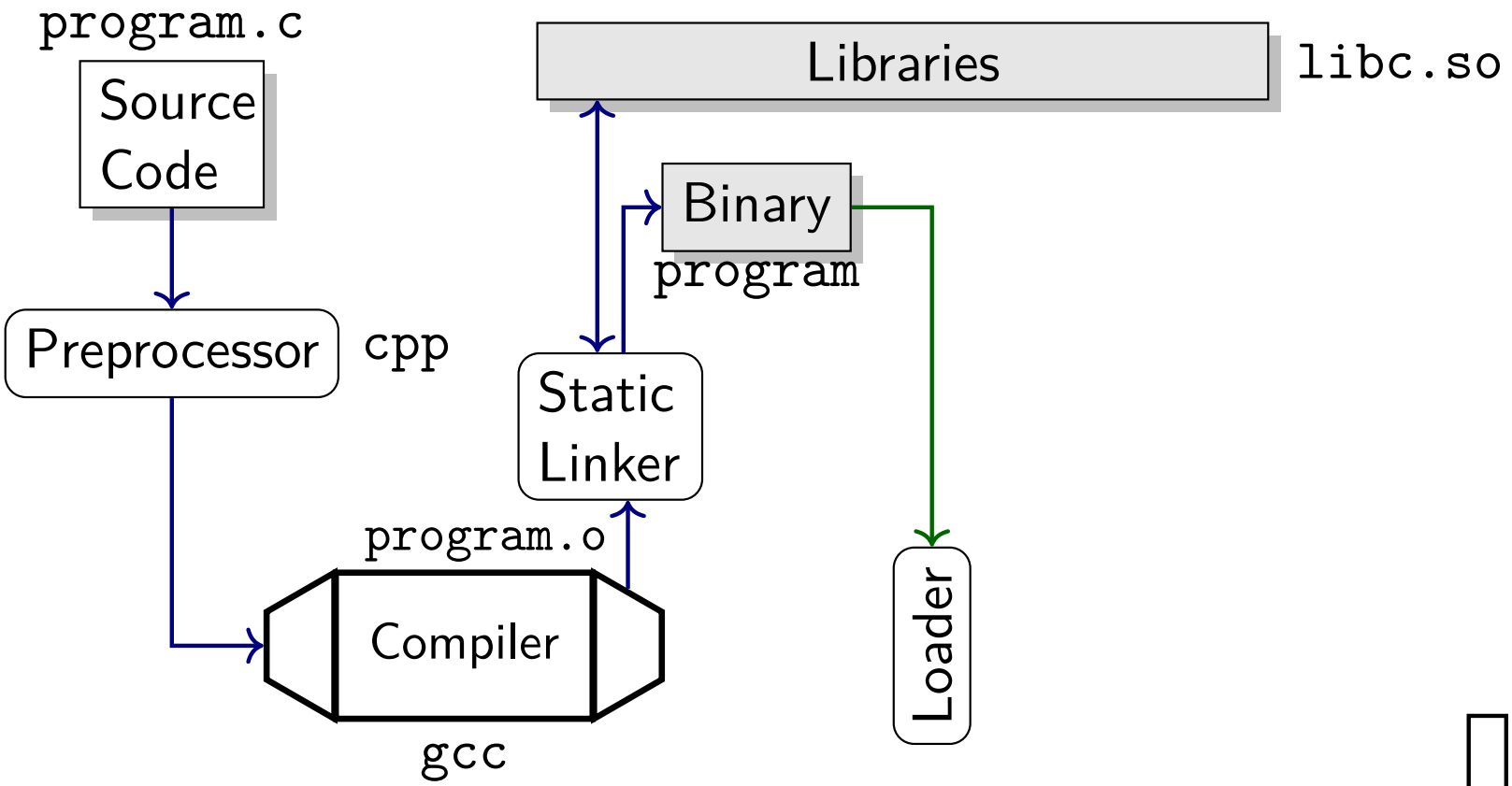


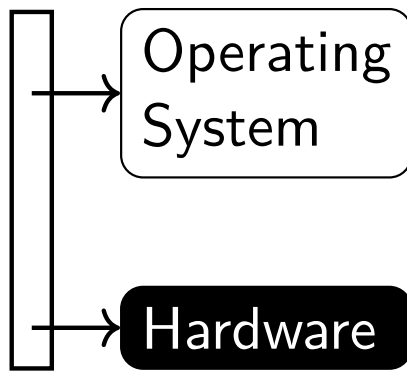
Program Execution Pipeline



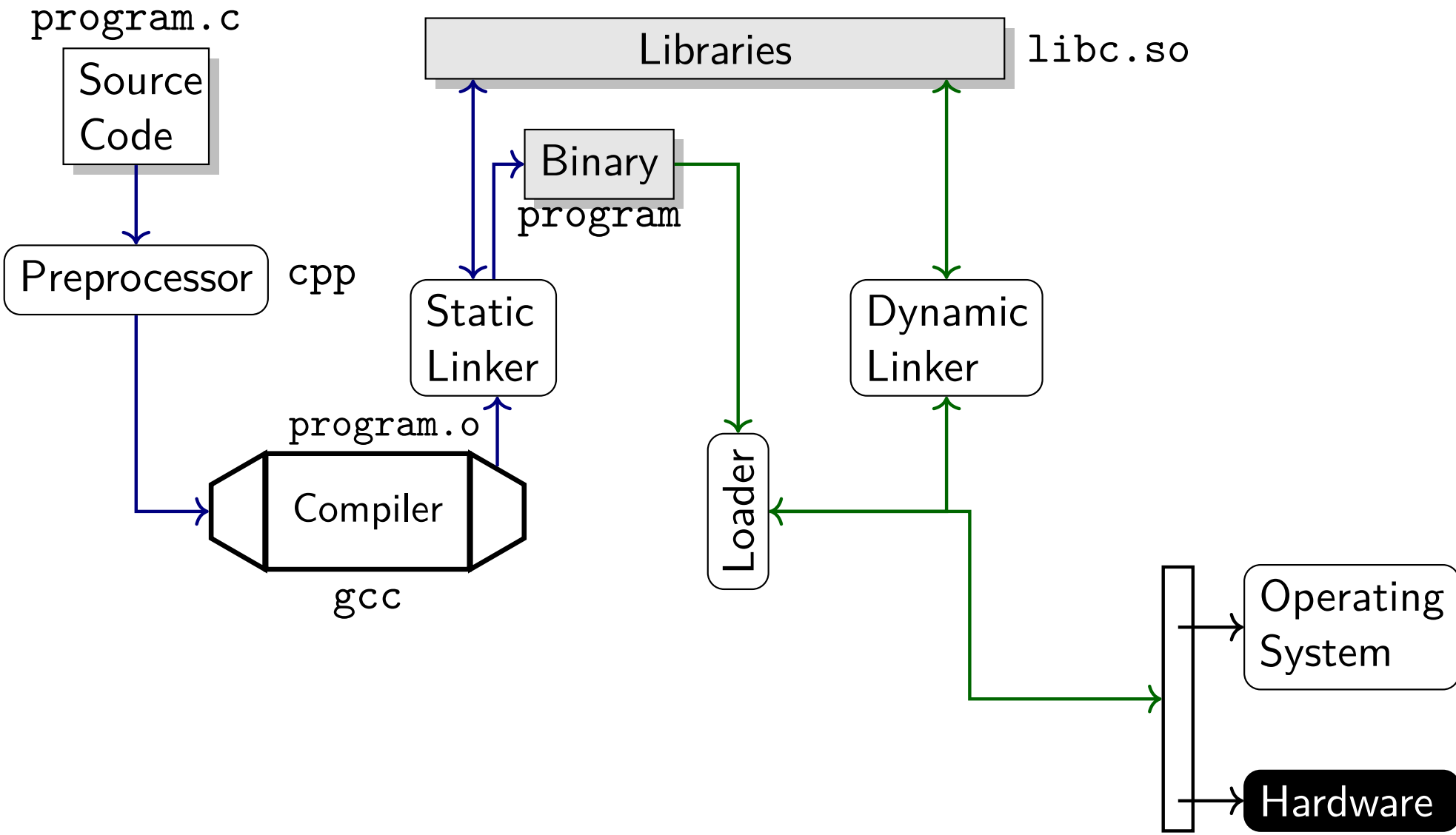


Program Execution Pipeline

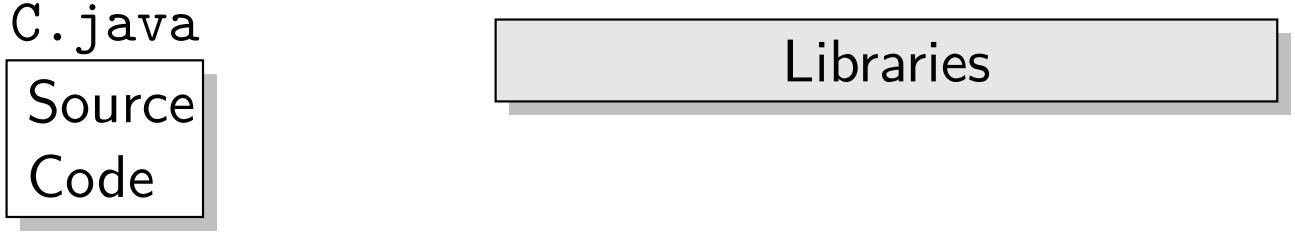


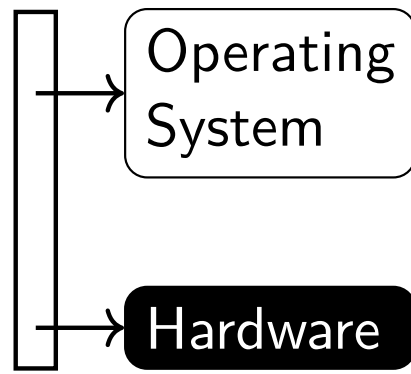


Program Execution Pipeline

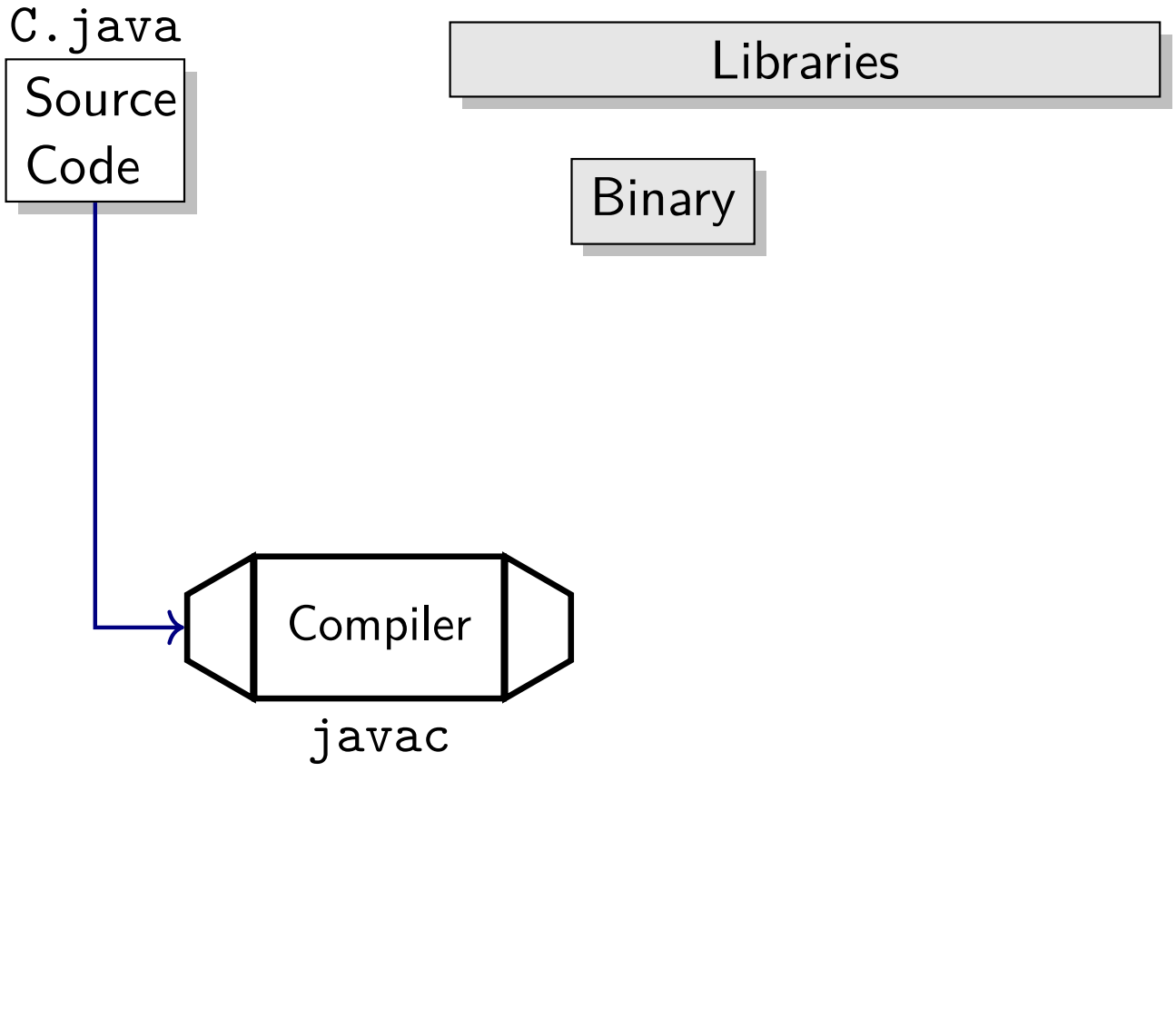


Program Execution Pipeline

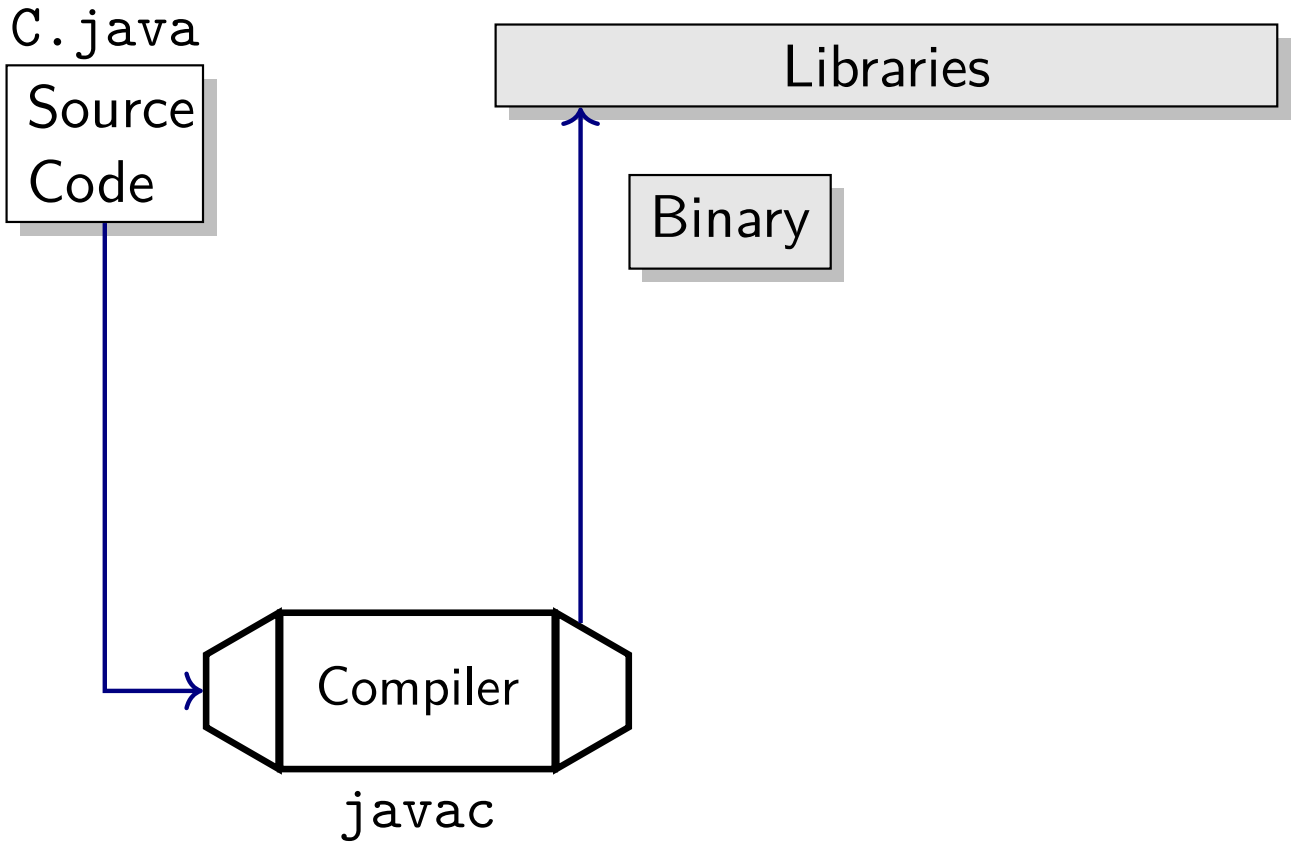


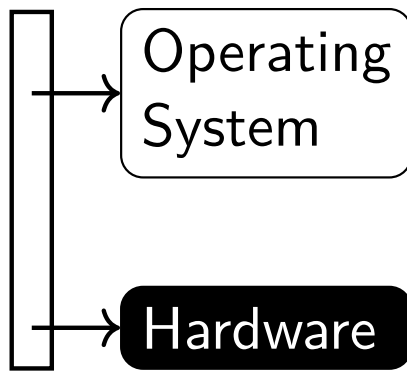


Program Execution Pipeline

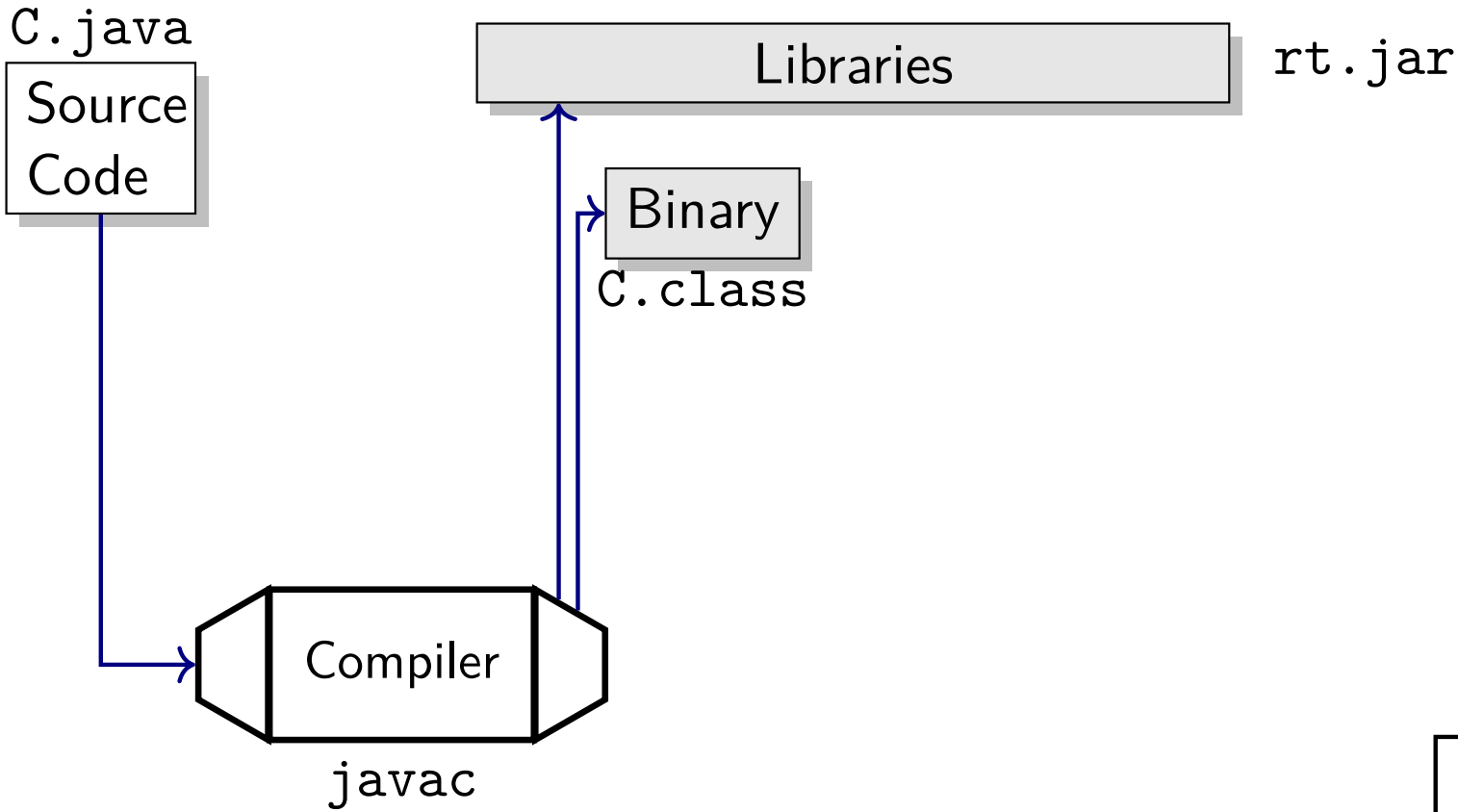


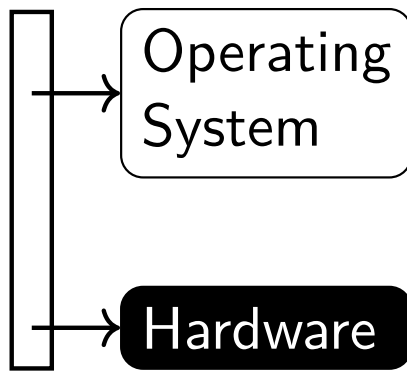
Program Execution Pipeline



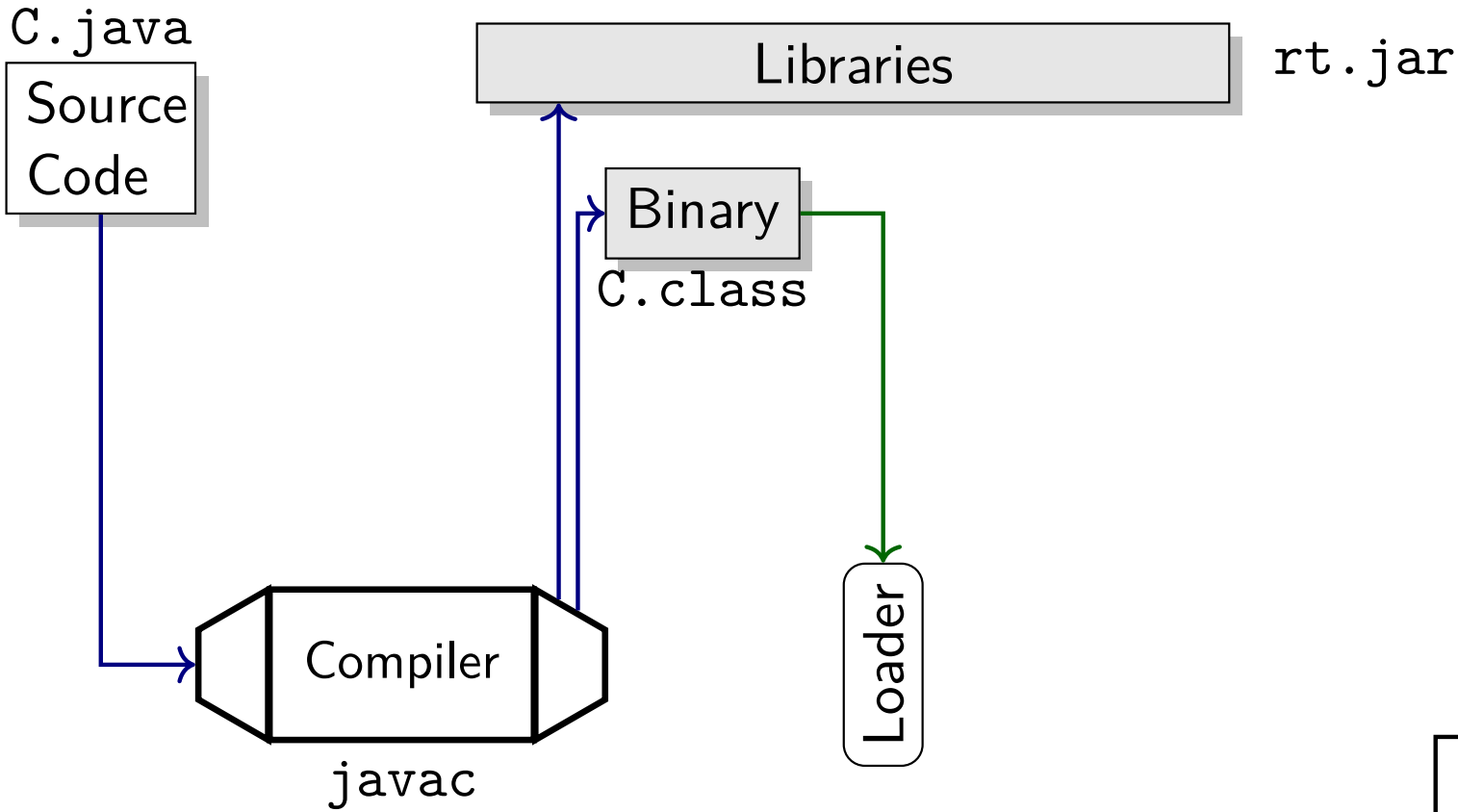


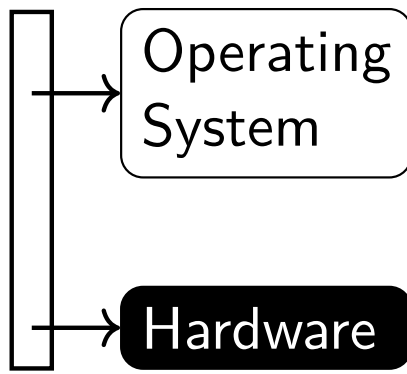
Program Execution Pipeline



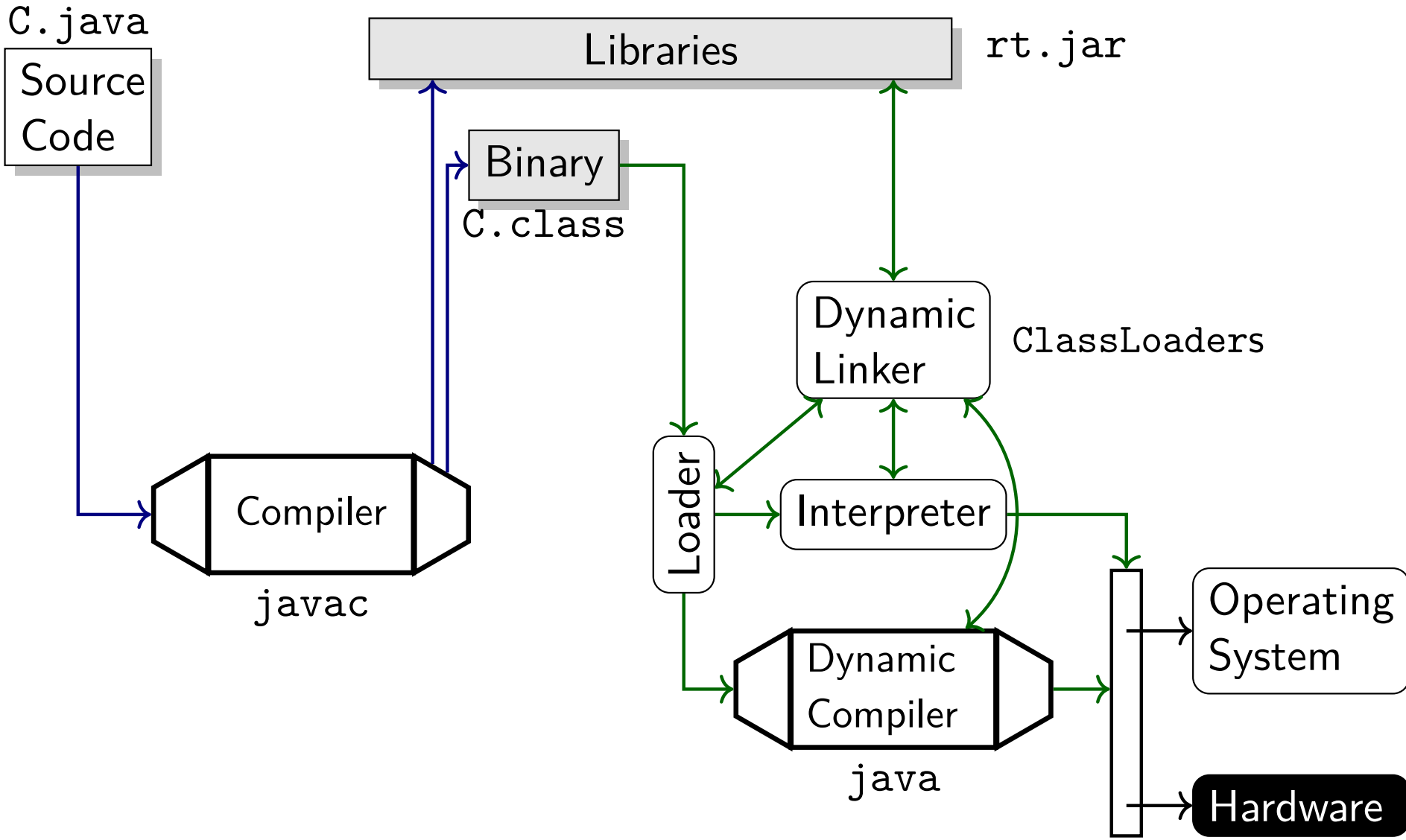


Program Execution Pipeline

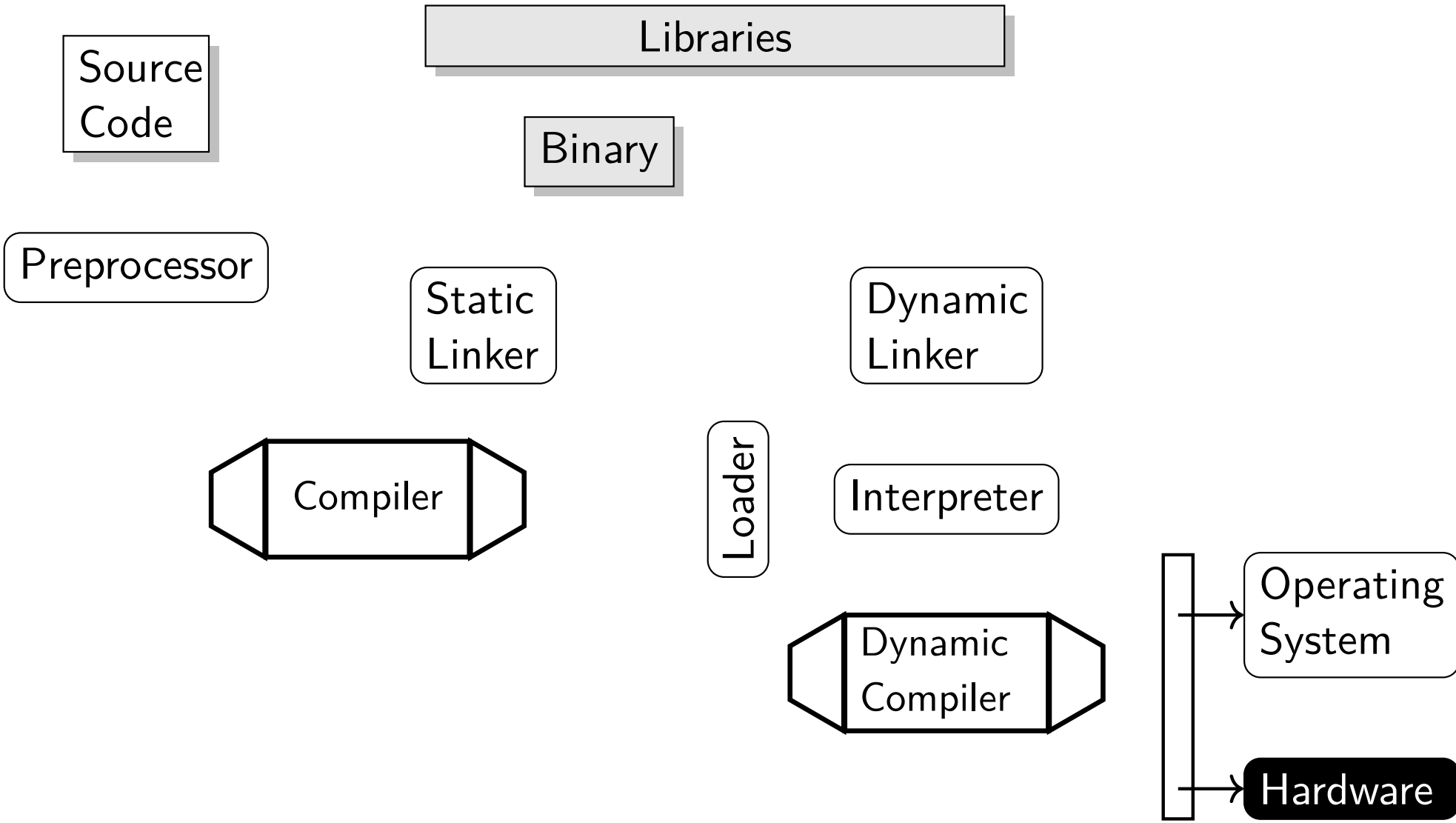




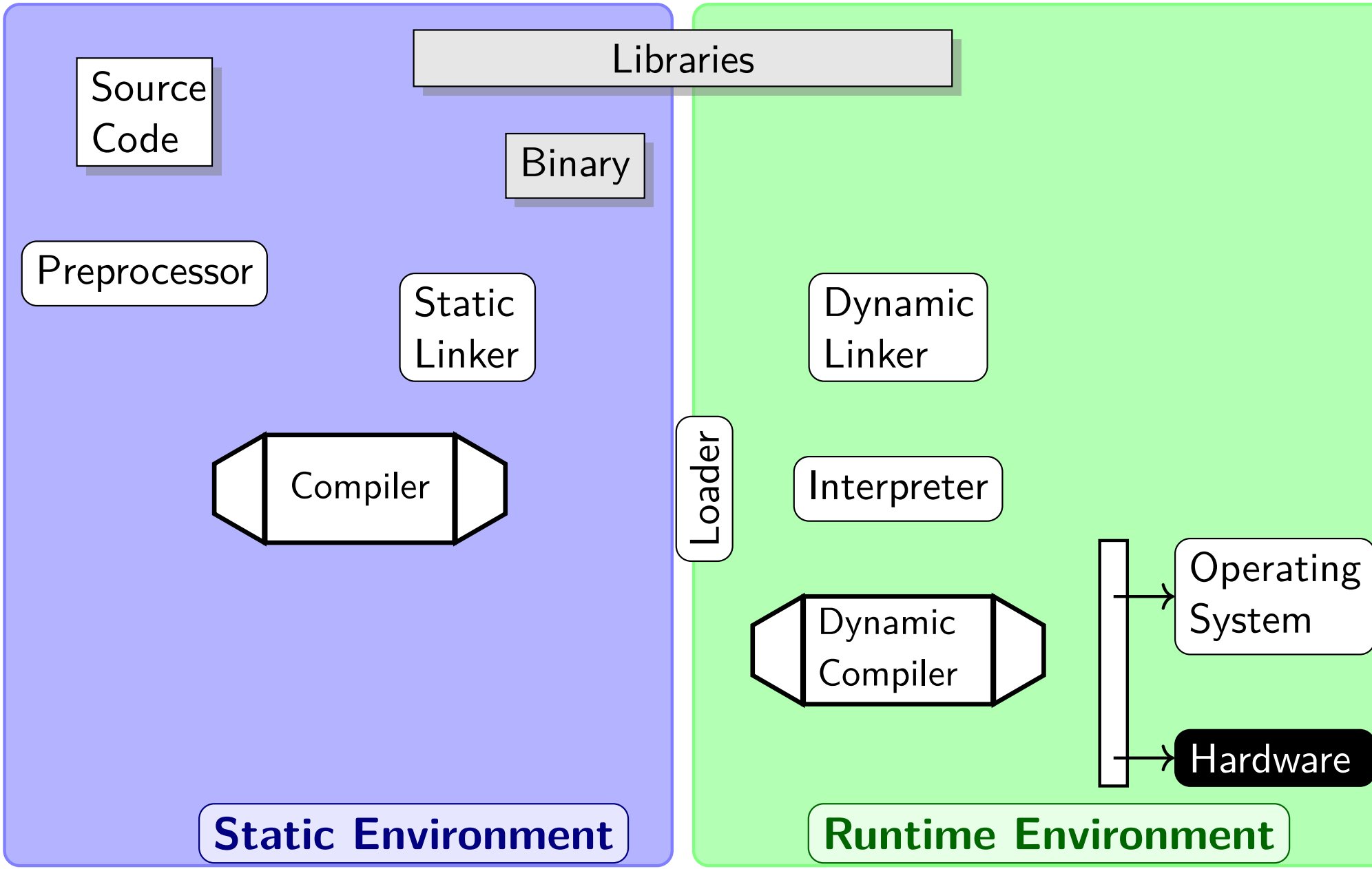
Program Execution Pipeline



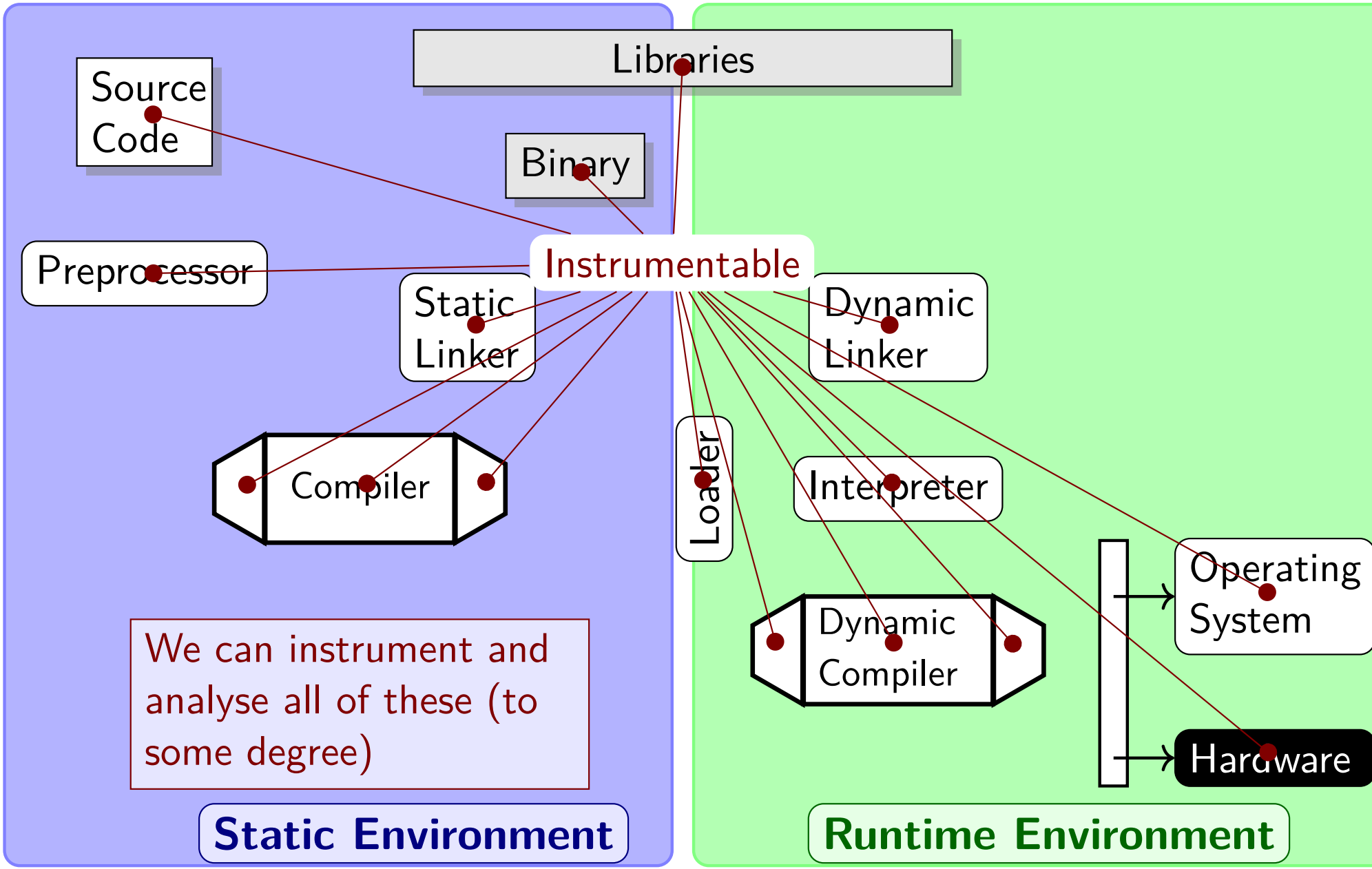
Program Execution Pipeline



Program Execution Pipeline



Program Execution Pipeline



Focus on Dynamic Analysis

- Recall: **Precise** but **Unsound**
- False positives: *none*
  (if what you are measuring is *observable*!)
- False negatives: *unbounded*
  (no insight over how much we are missing)

Unit Tests

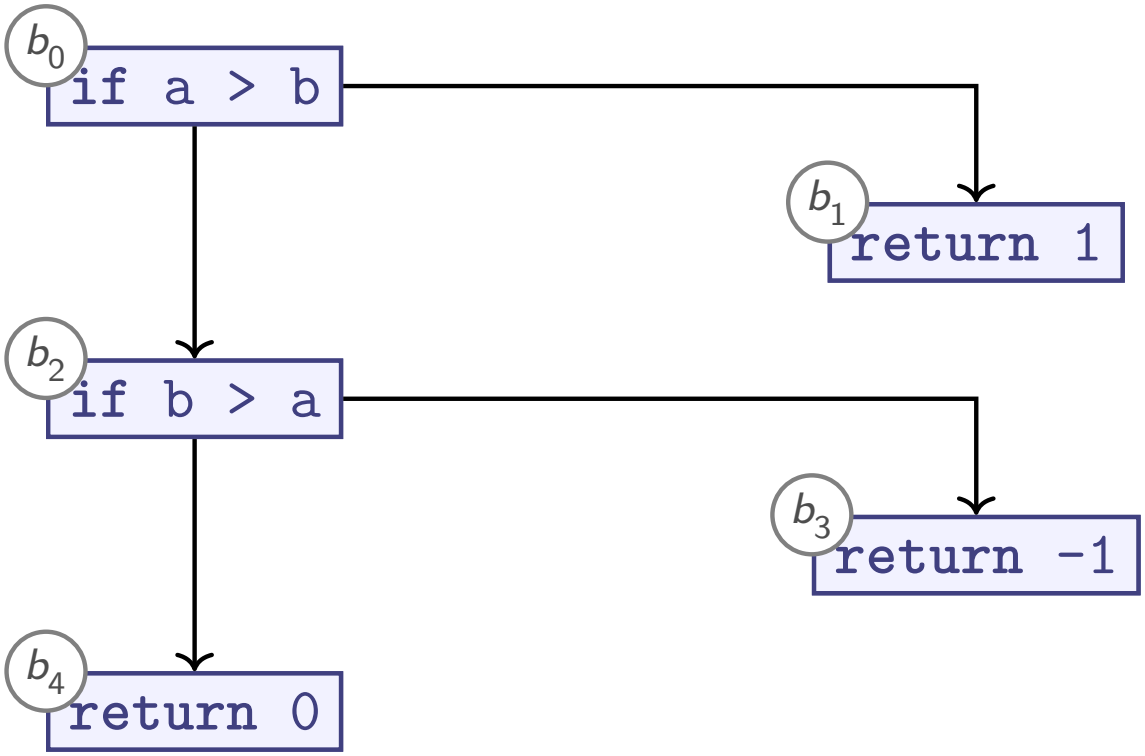
Teal

```
fun cmp(a, b) = {
  if a > b {
    return 1;
  }
  if a < b {
    return -1;
  }
  return 0;
}

fun test() = {
  assert cmp(1, 2) == -1;
  assert cmp(2, 1) == 1;
}
```

Unit tests are a simple form of dynamic program analysis

Unit Test Quality

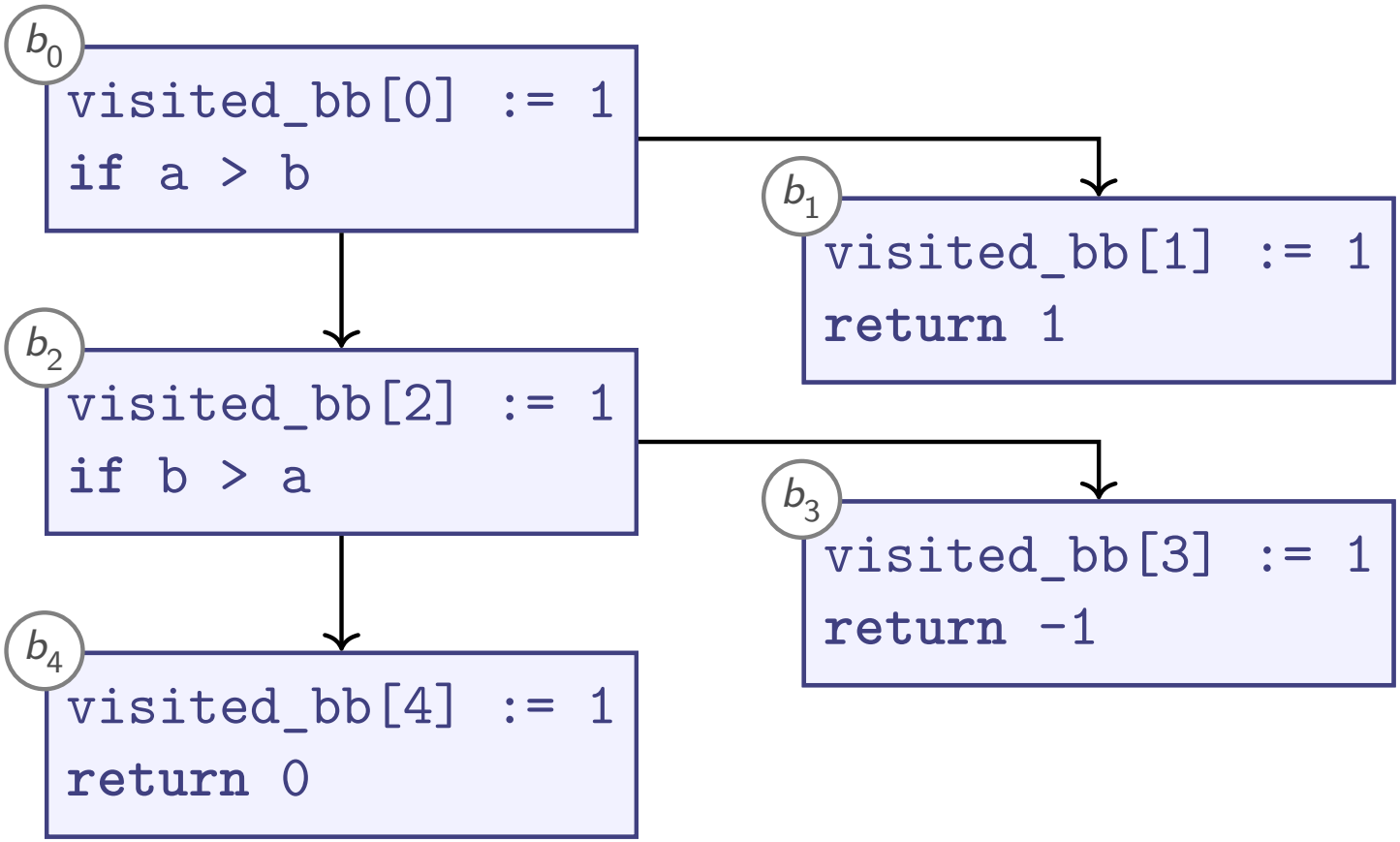


Teal

```
fun test() = {
  assert cmp(1, 2) == -1;
  assert cmp(2, 1) == 1;
}
```

Have I tested all behaviours?

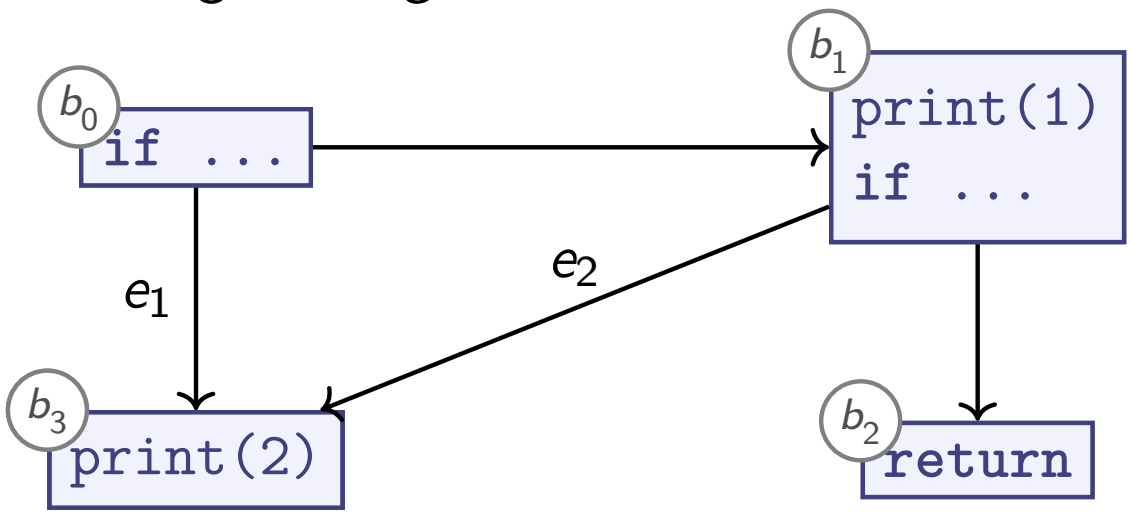
Test Coverage



- Test coverage = fraction of `visited_bb` elements updated

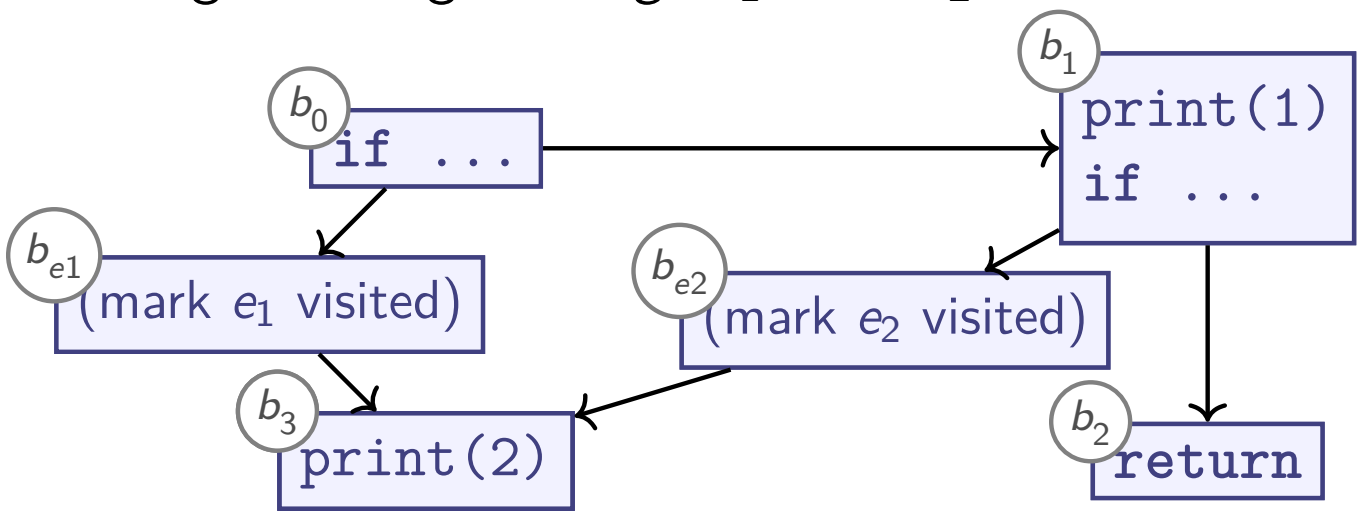
Test Coverage Properties

- **Statement Coverage**: % of executed CFG nodes
  or "Basic Blocks" of contiguous non-branching operations
  - Mark nodes/blocks as visited while testing
- **Edge Coverage**: % of taken CFG edges
  - Challenge: distinguish *edge* e_1 from e_2?



Test Coverage Properties

- **Statement Coverage**: % of executed CFG nodes
  or "Basic Blocks" of contiguous non-branching operations
  - Mark nodes/blocks as visited while testing
- **Edge Coverage**: % of taken CFG edges
  - Challenge: distinguish *edge* e_1 from e_2?



- Alternative: track last CFG node ID
- **Path Coverage**: % of CFG paths (less common)

Summary

- **Unit Tests** are a simple form of dynamic program analysis
  - Minimal tooling needed
  - Custom checks
  - Limited to what underlying language can express directly
- **Test Coverage** tells us how much of our code gets analysed by at least one unit test
- Implement by setting markers on relevant CFG nodes / blocks
  - Source-level: e.g. via DMCE (C/C++)
  - Binary-level: e.g. via JaCoCo/JCov (Java)
- Different criteria, such as:
  - **Statement Coverage**
  - **Edge Coverage**: may require helper CFG nodes
  - **Path Coverage**: paths through CFG (usually excluding loops)

Outlook

- No quizzes for today

`http://cs.lth.se/EDAP15`