cs316: Introduction and Lab Plan

Uday Khedker

(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,

Indian Institute of Technology, Bombay

January 2017

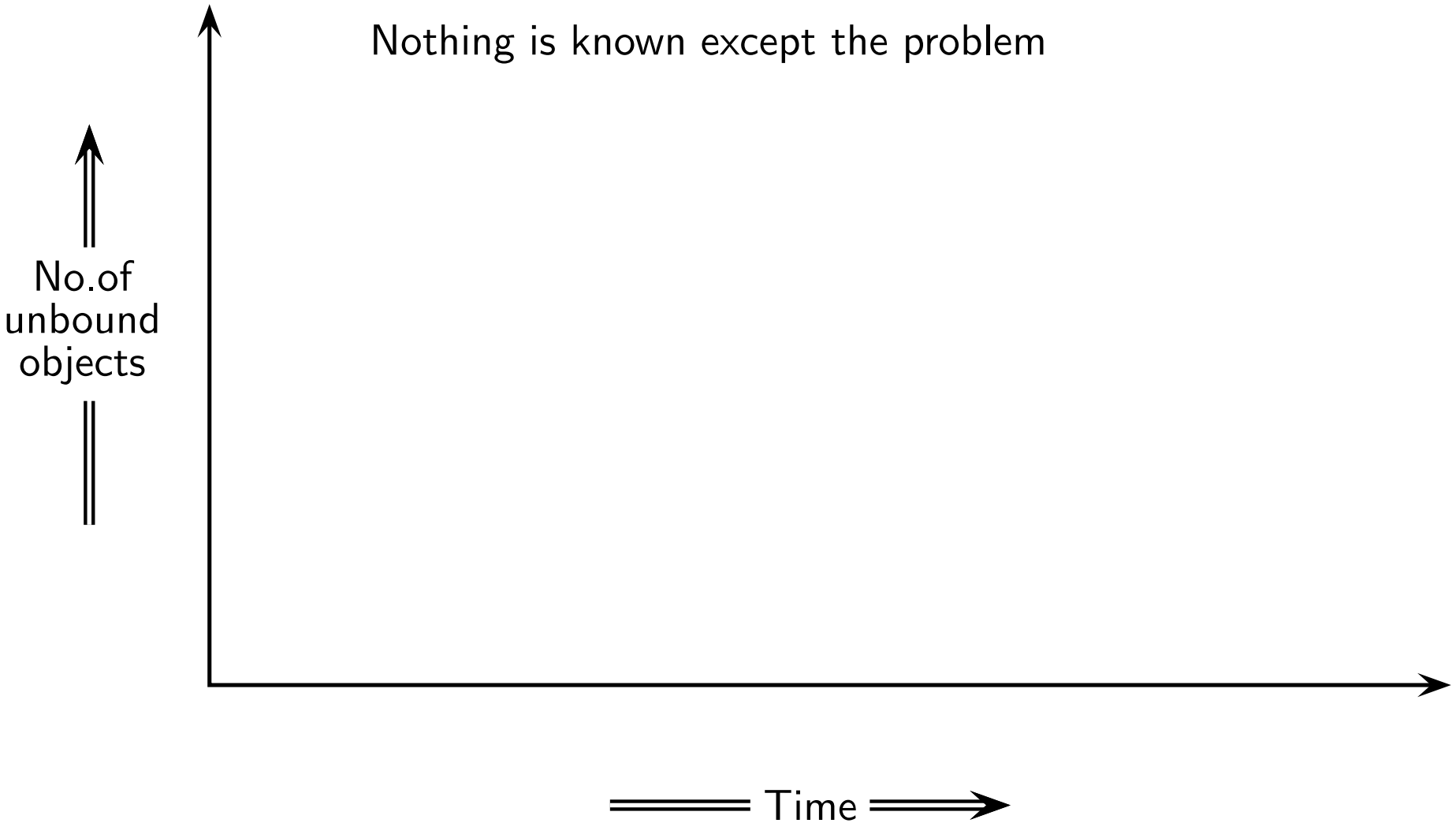
Outline

- Introduction to compilation
- An overview of compilation sequence
- Lab plan

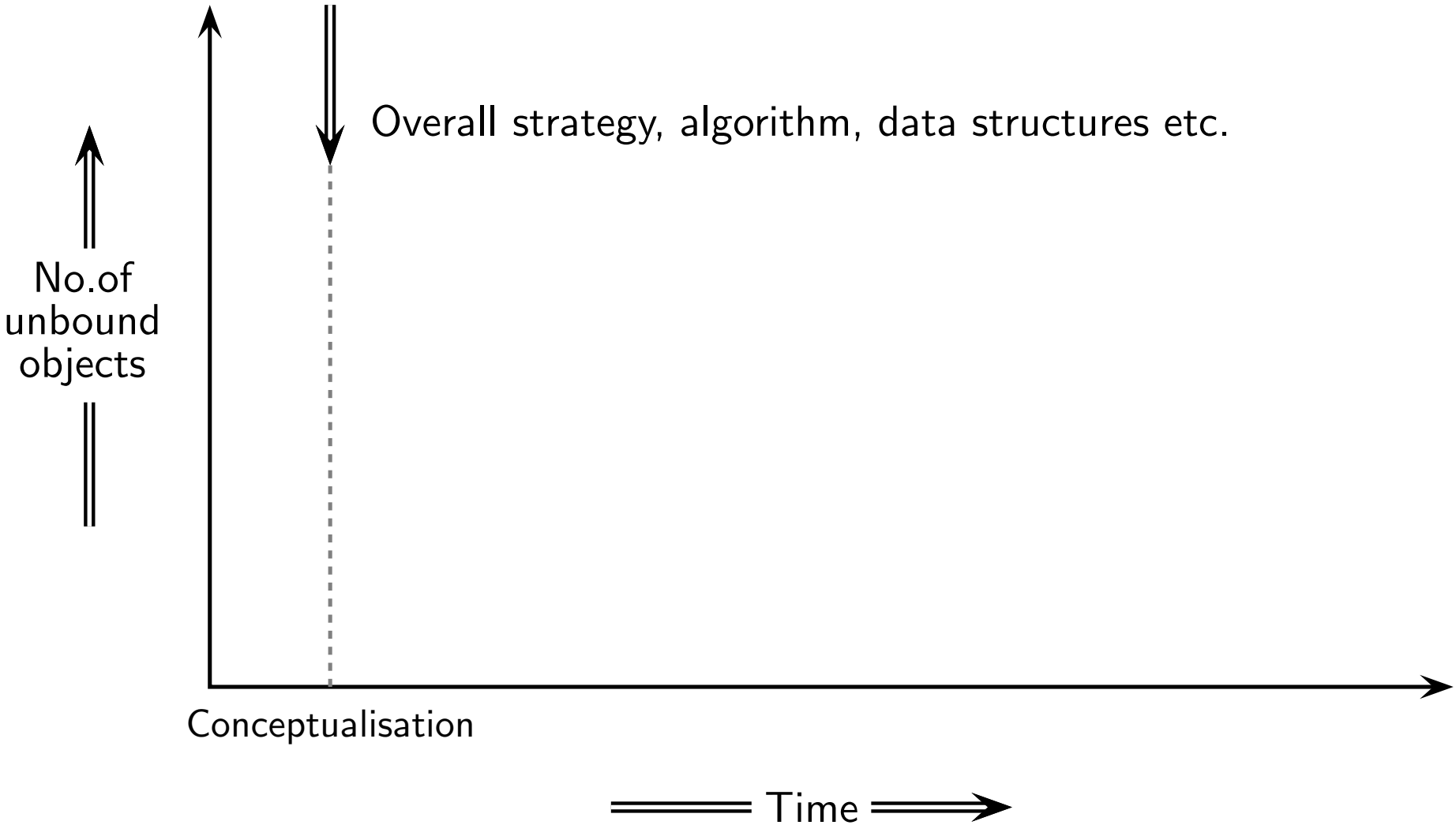
Part 1

Introduction to Compilation

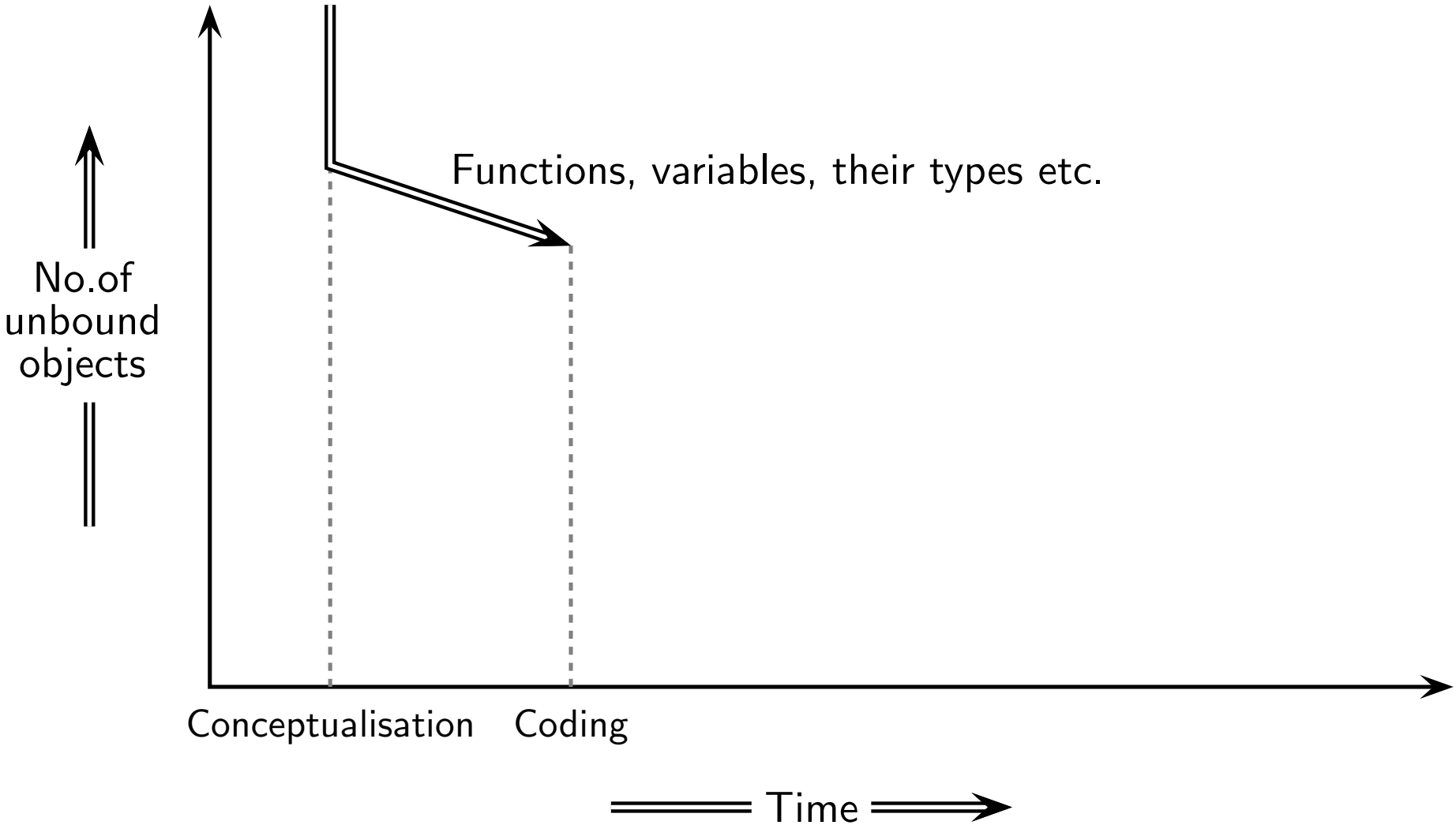
Binding


Nothing is known except the problem
No.of unbound objects
Time

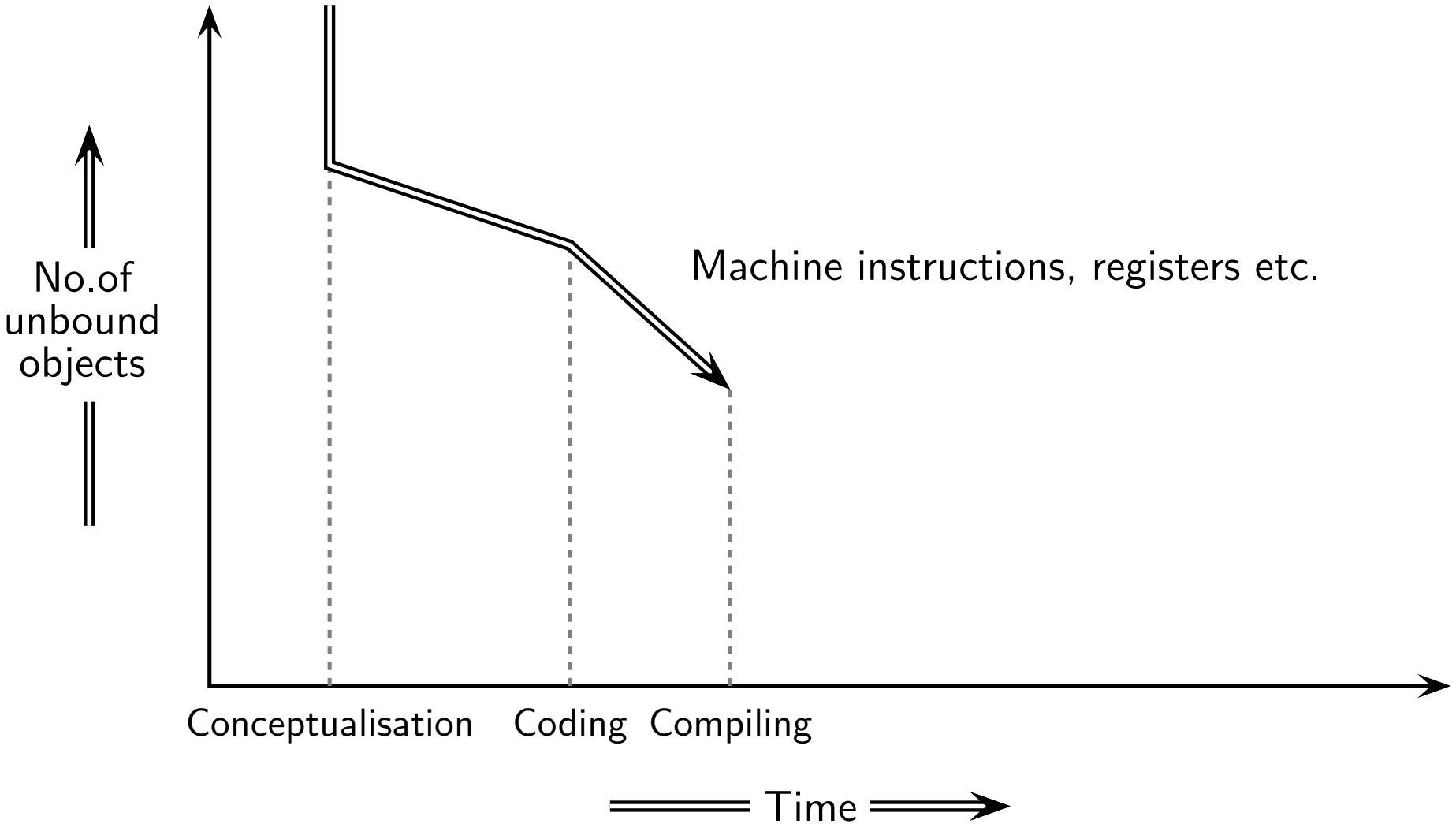
Binding


Overall strategy, algorithm, data structures etc.
No.of unbound objects
Conceptualisation
Time

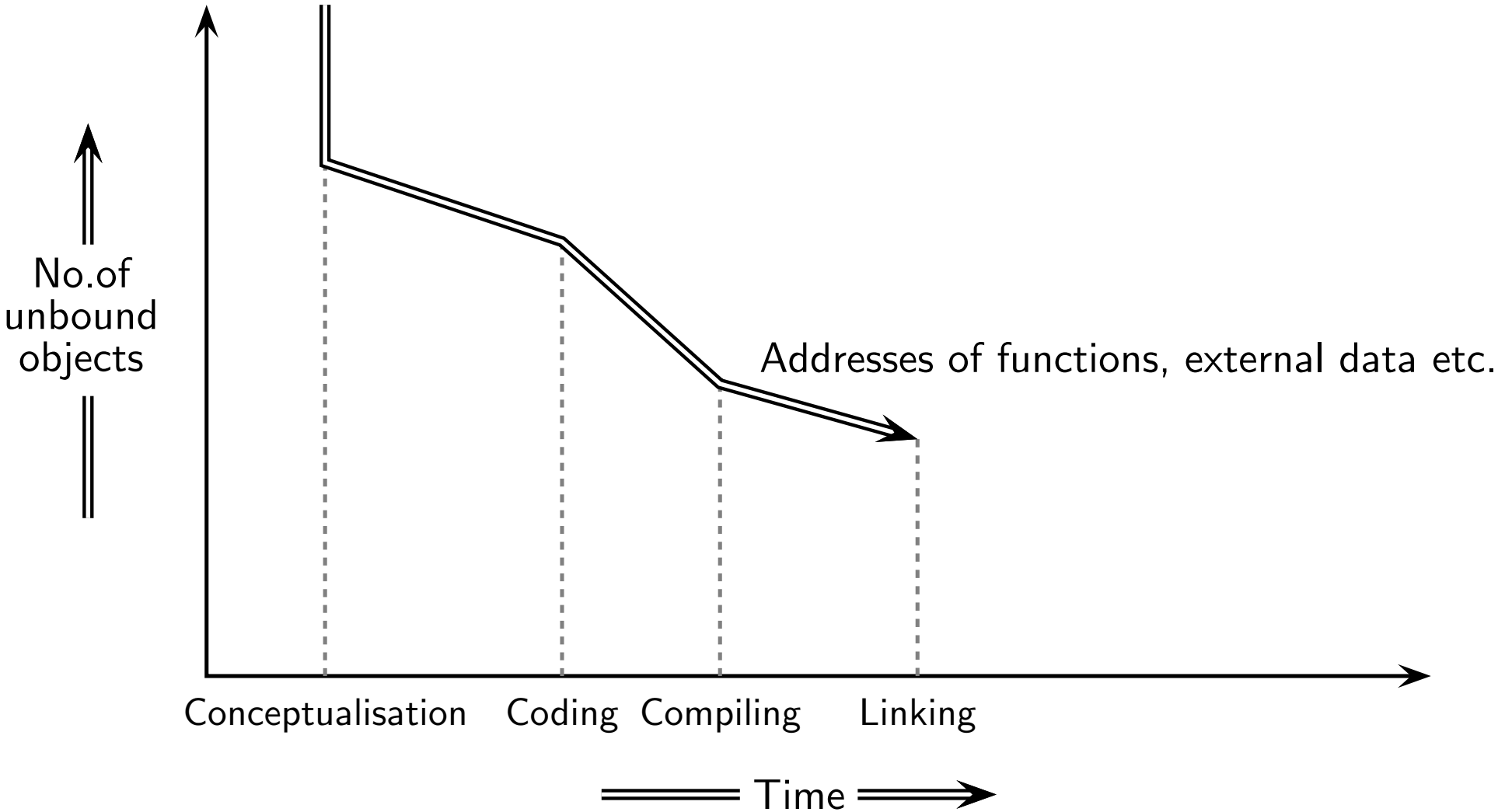
Binding


Functions, variables, their types etc.
No.of unbound objects
Conceptualisation
Coding
Time

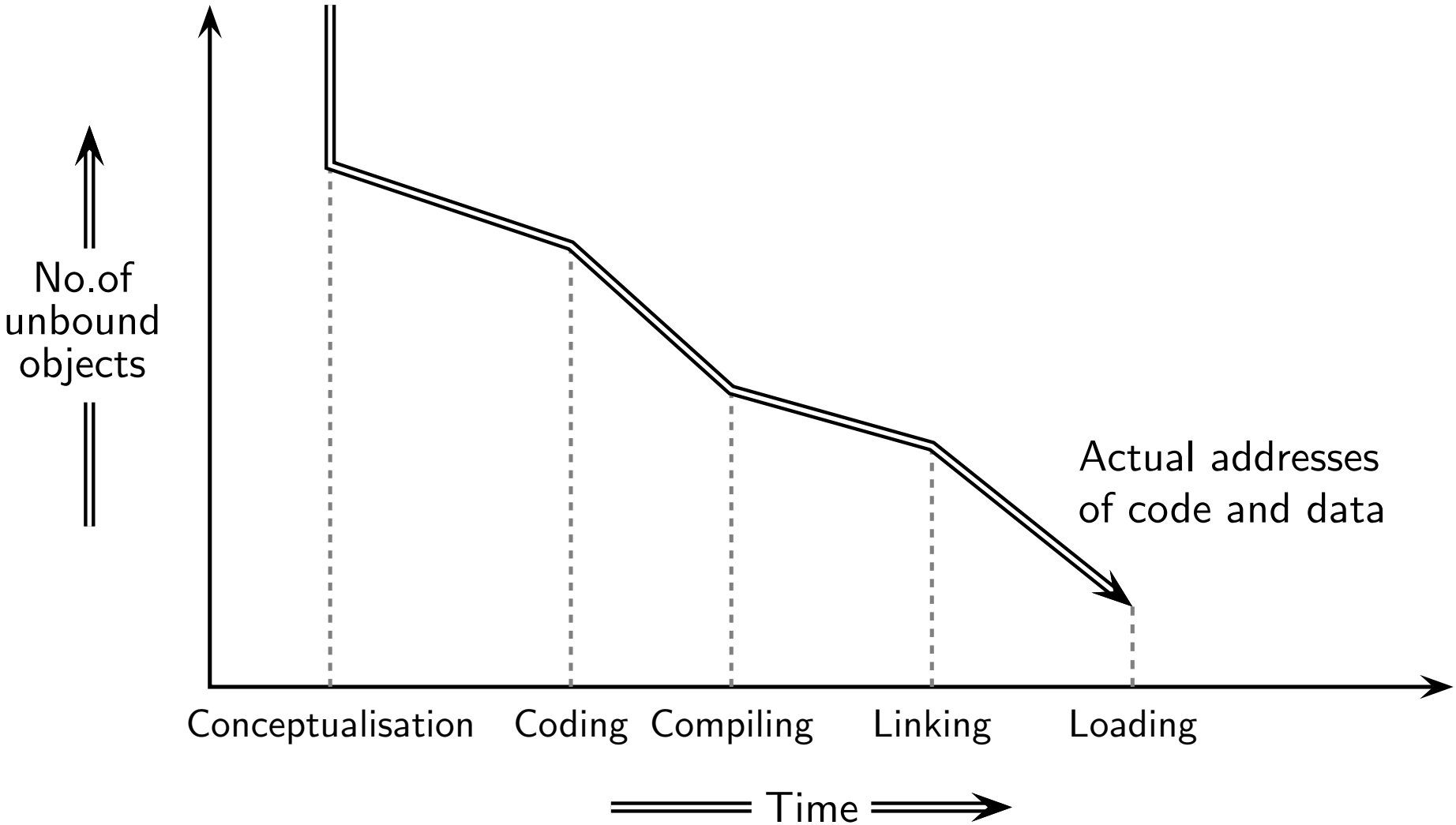
Binding


No.of
unbound
objects
Machine instructions, registers etc.
Conceptualisation
Coding
Compiling
Time

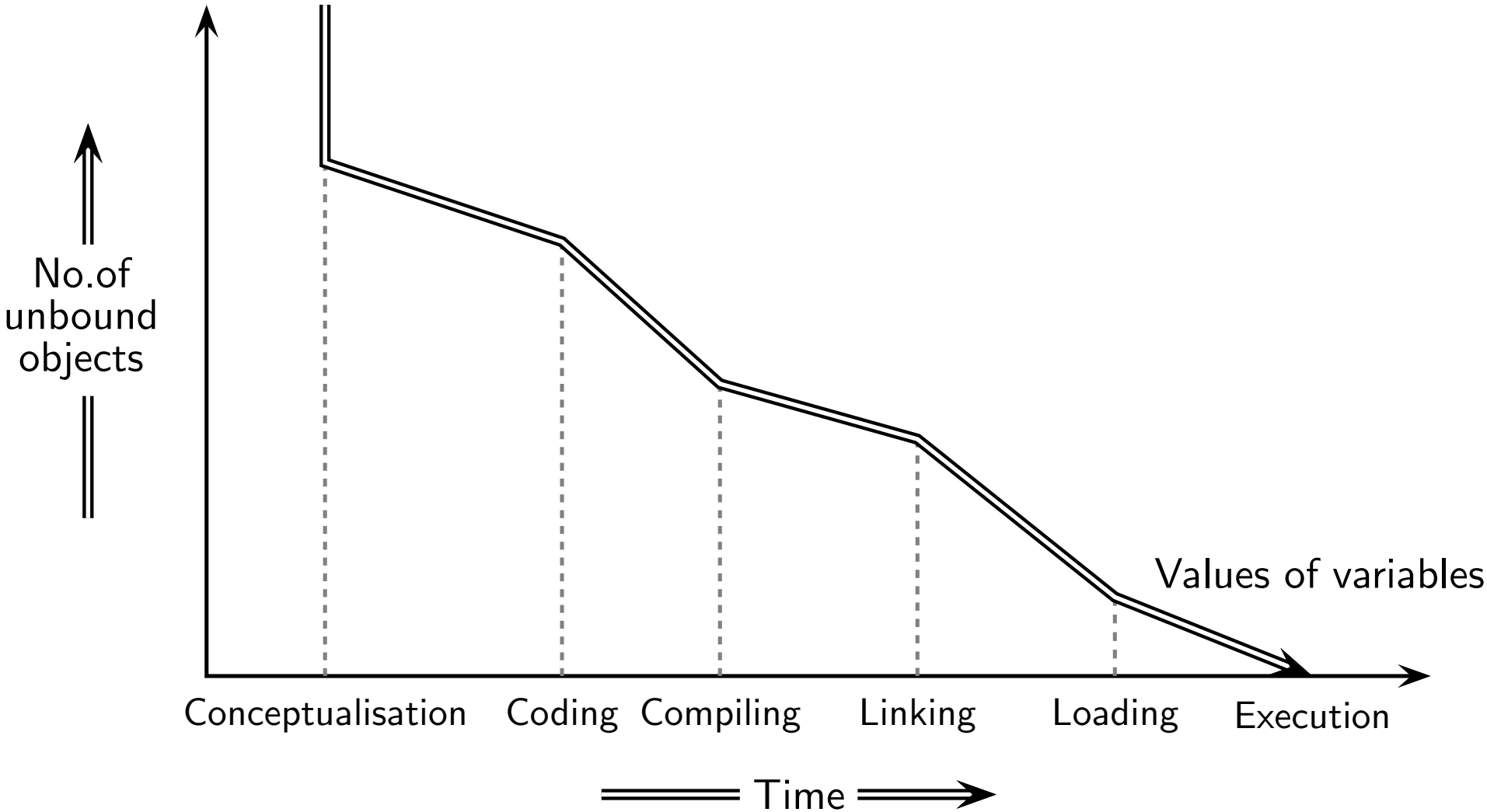
Binding


No.of
unbound
objects
Addresses of functions, external data etc.
Conceptualisation
Coding
Compiling
Linking
Time

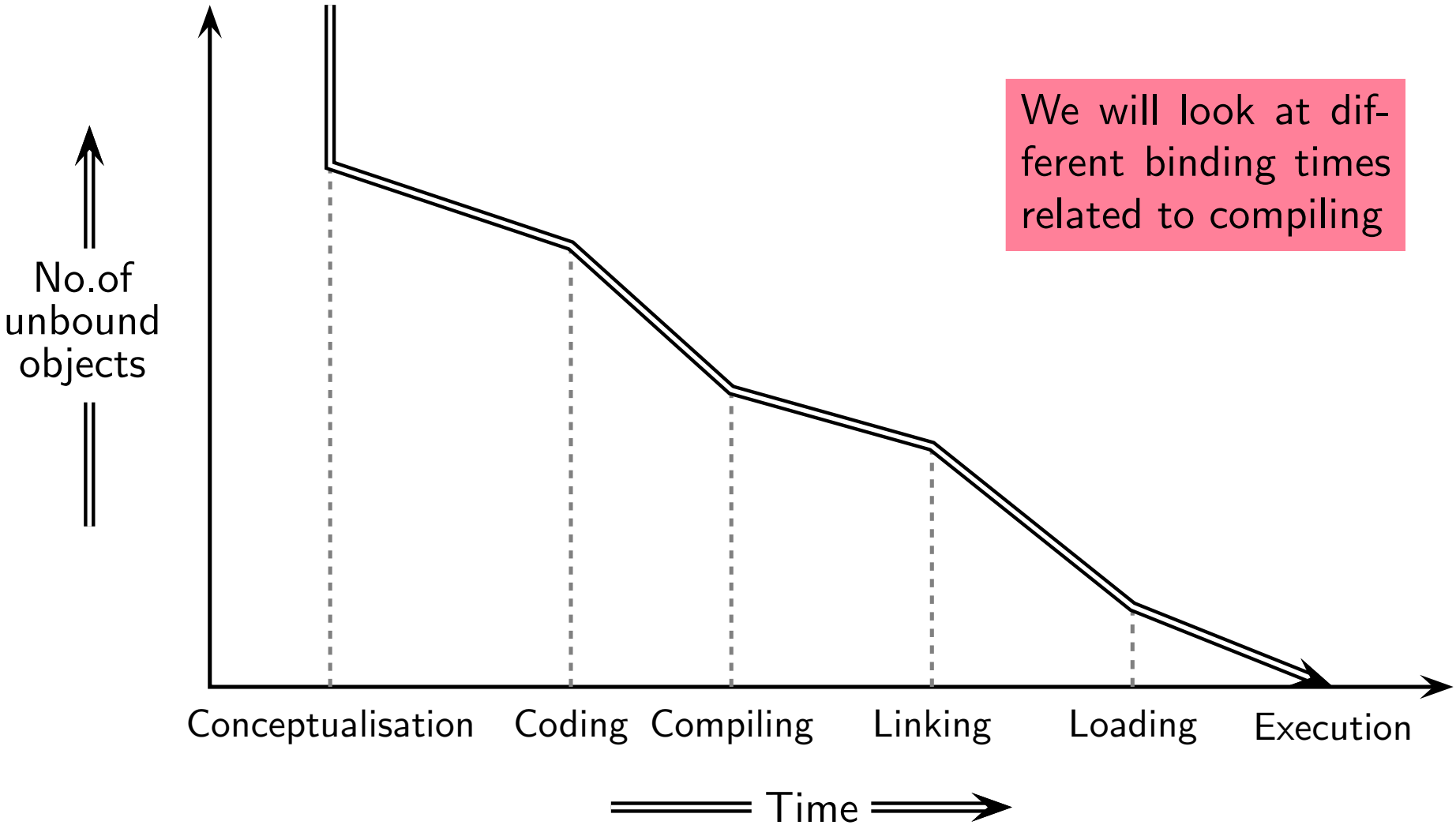
Binding


No.of
unbound
objects
Actual addresses
of code and data
Conceptualisation
Coding
Compiling
Linking
Loading
Time

Binding

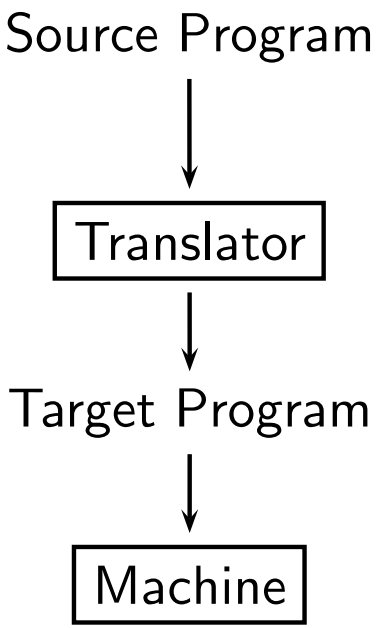

No.of
unbound
objects
Values of variables
Conceptualisation
Coding
Compiling
Linking
Loading
Execution
Time

Binding

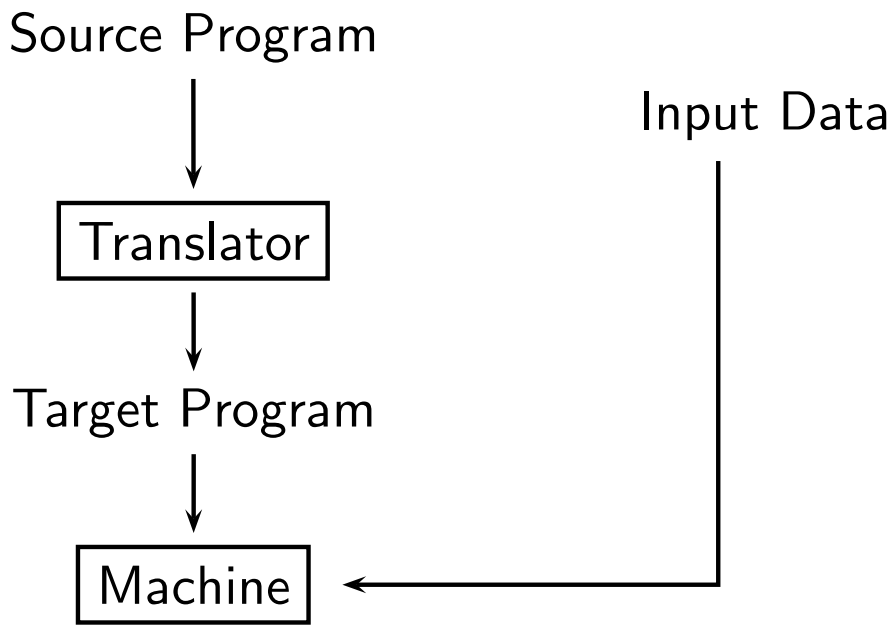


We will look at different binding times related to compiling

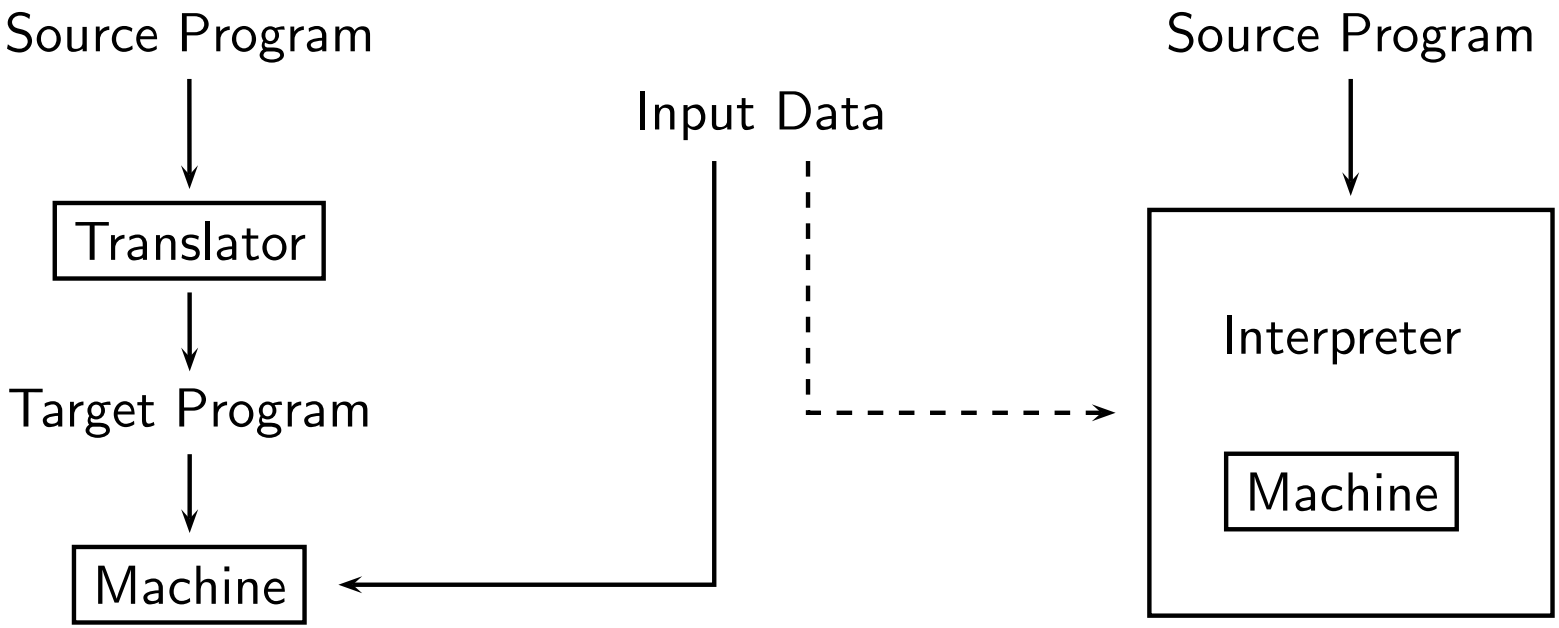
Implementation Mechanisms



Implementation Mechanisms

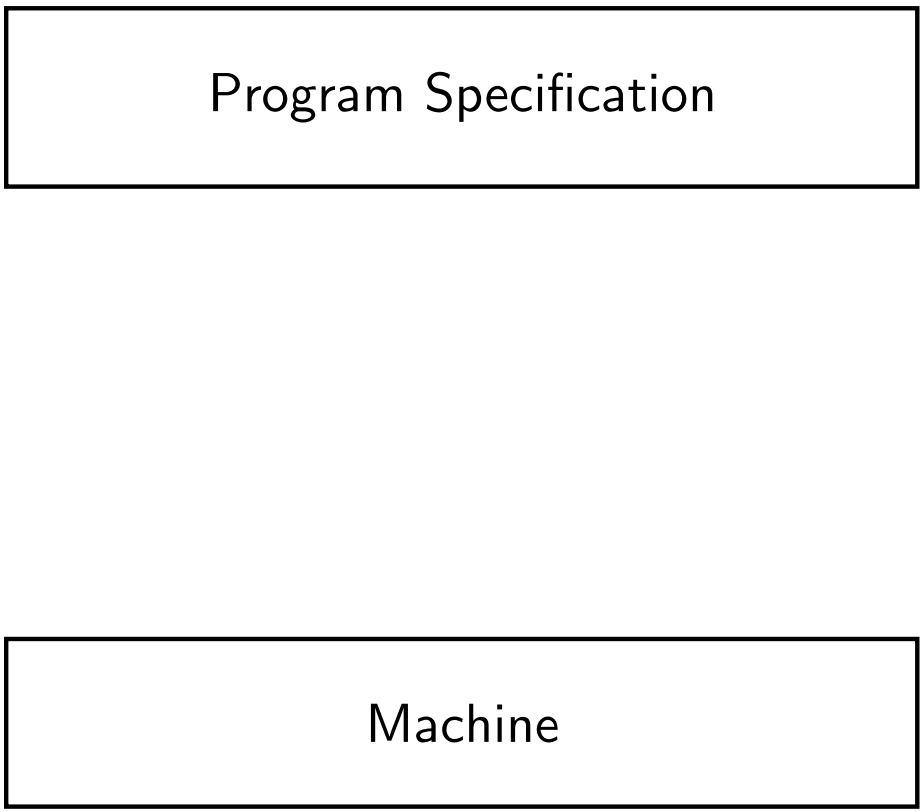


Implementation Mechanisms



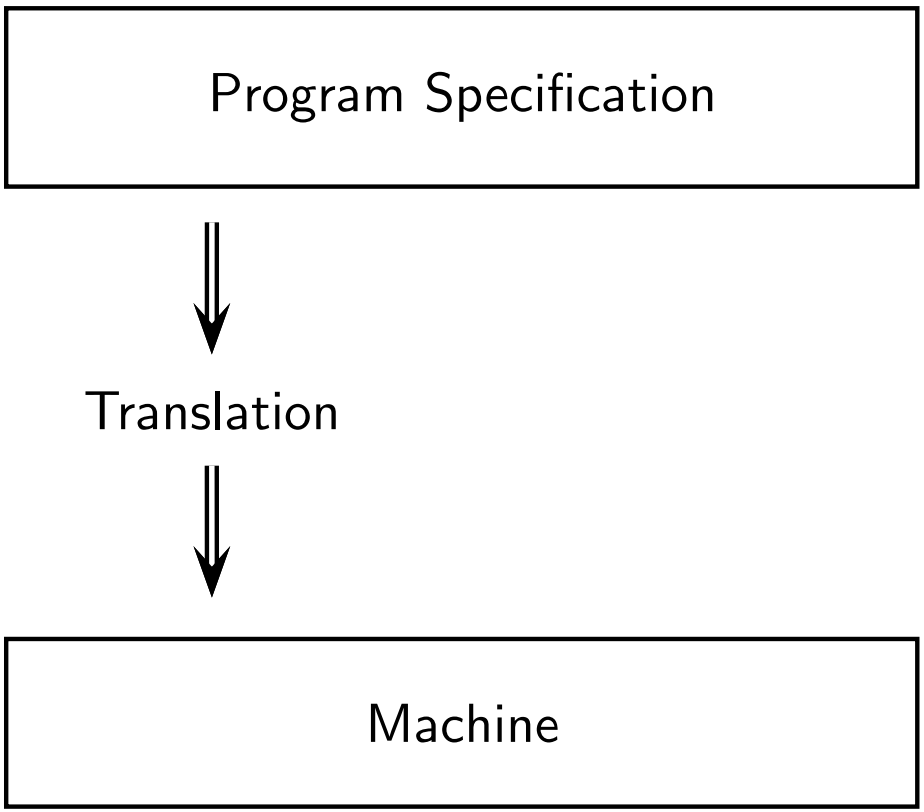
Implementation Mechanisms as "Bridges"

- "Gap" between the "levels" of program specification and execution



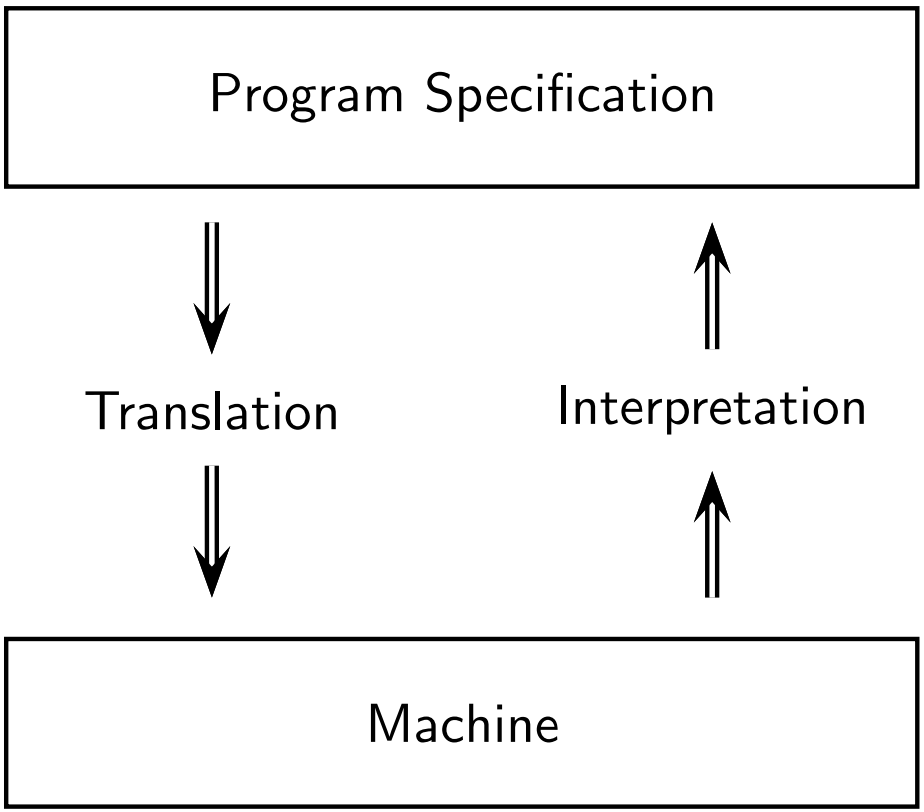
Implementation Mechanisms as "Bridges"

- "Gap" between the "levels" of program specification and execution



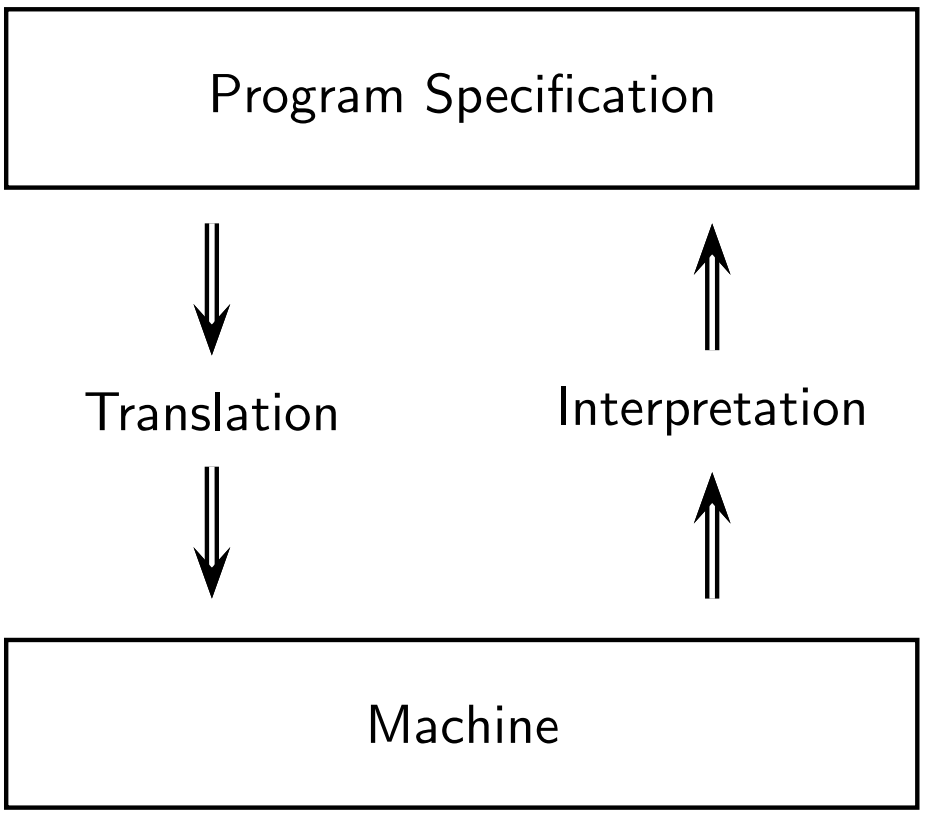
Implementation Mechanisms as "Bridges"

- "Gap" between the "levels" of program specification and execution



Implementation Mechanisms as "Bridges"

- "Gap" between the "levels" of program specification and execution



State : Variables
Operations: Expressions, Control Flow

State : Memory, Registers
Operations: Machine Instructions

High and Low Level Abstractions

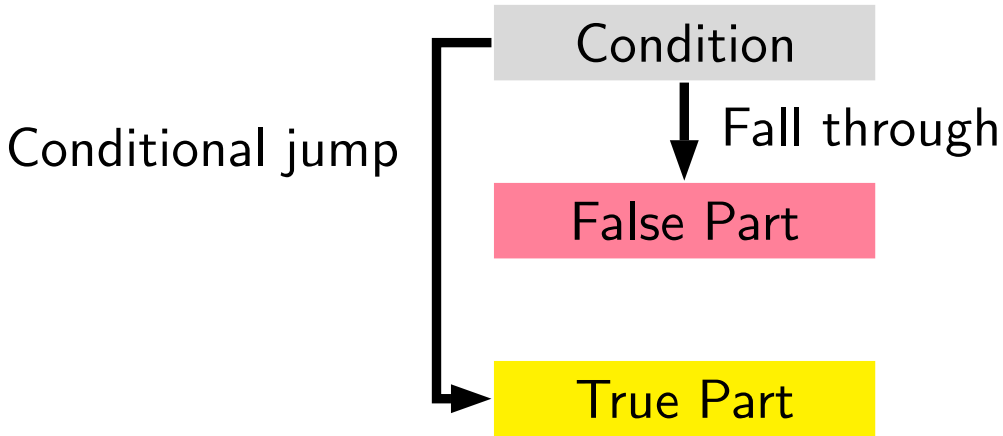
Input C statement

```
a = b<10?b:c;
```

Spim Assembly Equivalent

```
     lw   $t0, 4($fp)   ;     t0 <- b          # Is b smaller
     slti $t0, $t0, 10  ;     t0 <- t0 < 10    # than 10?
     not  $t0, $t0      ;     t0 <- !t0
     bgtz $t0, L0:      ;     if t0>0 goto L0
     lw   $t0, 4($fp)   ;     t0 <- b          # YES
     b    L1:           ;     goto L1
L0:  lw   $t0, 8($fp)   ;L0:  t0 <- c          # NO
L1:  sw   0($fp), $t0   ;L1:  a <- t0
```


High and Low Level Abstractions



Input C statement

```
a = b<10?b:c;
```

Spim Assembly Equivalent

```
     lw   $t0, 4($fp)   ;     t0 <- b          # Is b smaller
     slti $t0, $t0, 10  ;     t0 <- t0 < 10    # than 10?
     not  $t0, $t0      ;     t0 <- !t0
     bgtz $t0, L0:      ;     if t0>0 goto L0
     lw   $t0, 4($fp)   ;     t0 <- b          # YES
     b    L1:           ;     goto L1
L0:  lw   $t0, 8($fp)   ;L0:  t0 <- c          # NO
L1:  sw   0($fp), $t0   ;L1:  a <- t0
```


High and Low Level Abstractions

NOT Condition

True Part

False Part

Input C statement

```
a = b<10?b:c;
```

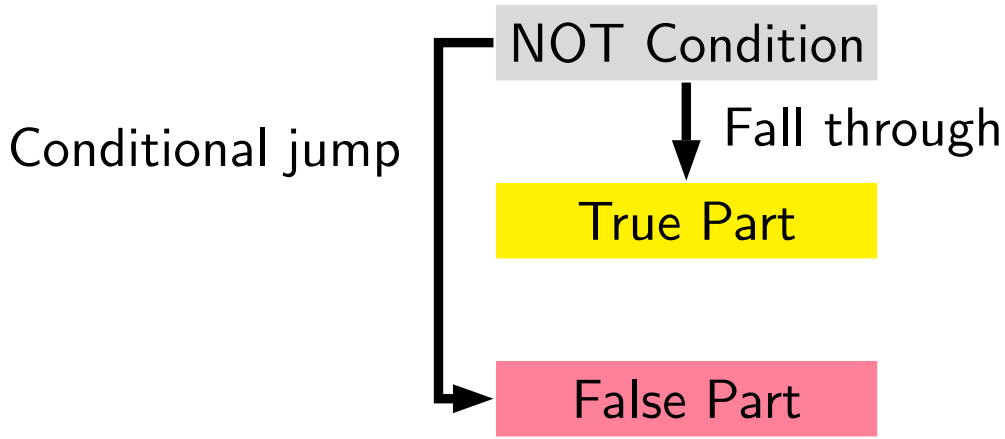
Spim Assembly Equivalent

```
     lw   $t0, 4($fp)   ;     t0 <- b          # Is b smaller
     slti $t0, $t0, 10  ;     t0 <- t0 < 10    # than 10?
     not  $t0, $t0      ;     t0 <- !t0
     bgtz $t0, L0:      ;     if t0>0 goto L0
     lw   $t0, 4($fp)   ;     t0 <- b          # YES
     b    L1:           ;     goto L1
L0:  lw   $t0, 8($fp)   ;L0:  t0 <- c          # NO
L1:  sw   0($fp), $t0   ;L1:  a <- t0
```


High and Low Level Abstractions

Input C statement

```
a = b<10?b:c;
```



Spim Assembly Equivalent

```
     lw   $t0, 4($fp)   ;     t0 <- b          # Is b smaller
     slti $t0, $t0, 10  ;     t0 <- t0 < 10    # than 10?
     not  $t0, $t0      ;     t0 <- !t0
     bgtz $t0, L0:      ;     if t0>0 goto L0
     lw   $t0, 4($fp)   ;     t0 <- b          # YES
     b    L1:           ;     goto L1
L0:  lw   $t0, 8($fp)   ;L0:  t0 <- c          # NO
L1:  sw   0($fp), $t0   ;L1:  a <- t0
```

Implementation Mechanisms

- Translation = Analysis + Synthesis

  Interpretation = Analysis + Execution

Implementation Mechanisms

- Translation = Analysis + Synthesis

  Interpretation = Analysis + Execution

- Translation Instructions $\Longrightarrow$ Equivalent Instructions

Implementation Mechanisms

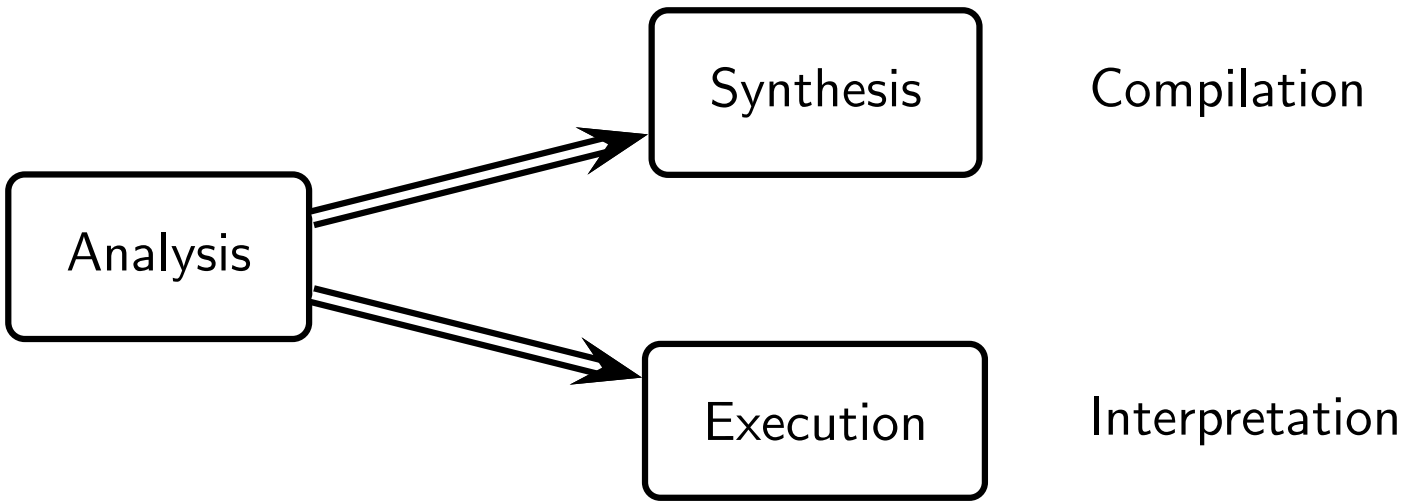
- Translation = Analysis + Synthesis

  Interpretation = Analysis + Execution

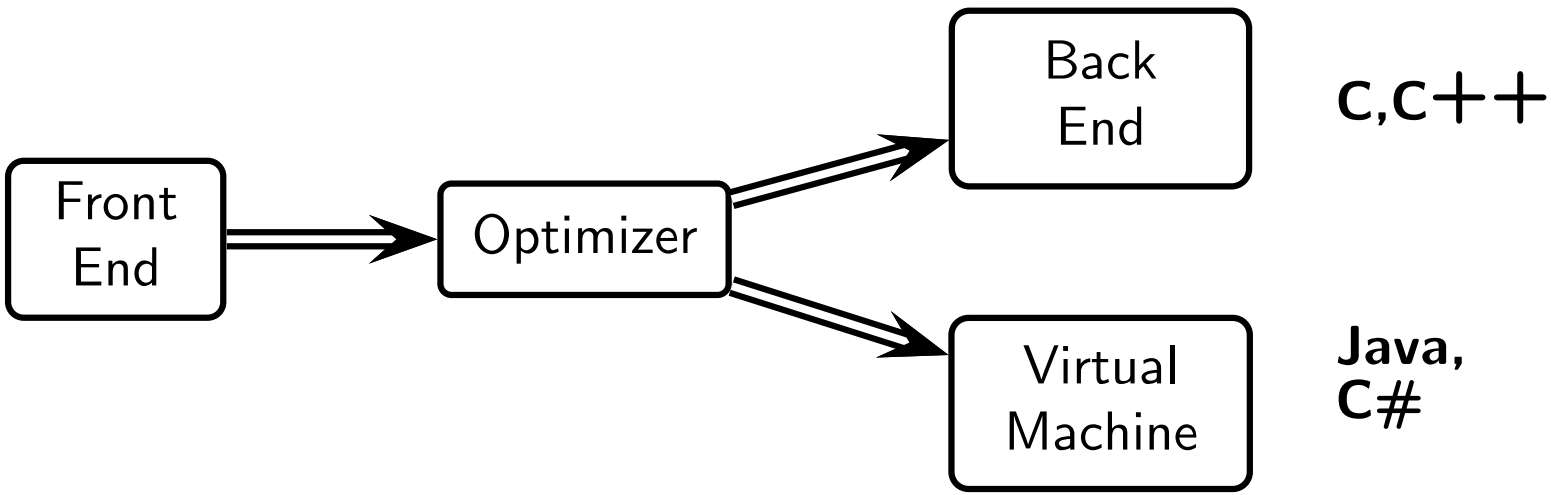
- Translation Instructions $\Longrightarrow$ Equivalent Instructions

  Interpretation Instructions $\Longrightarrow$ Actions Implied by Instructions

Language Implementation Models



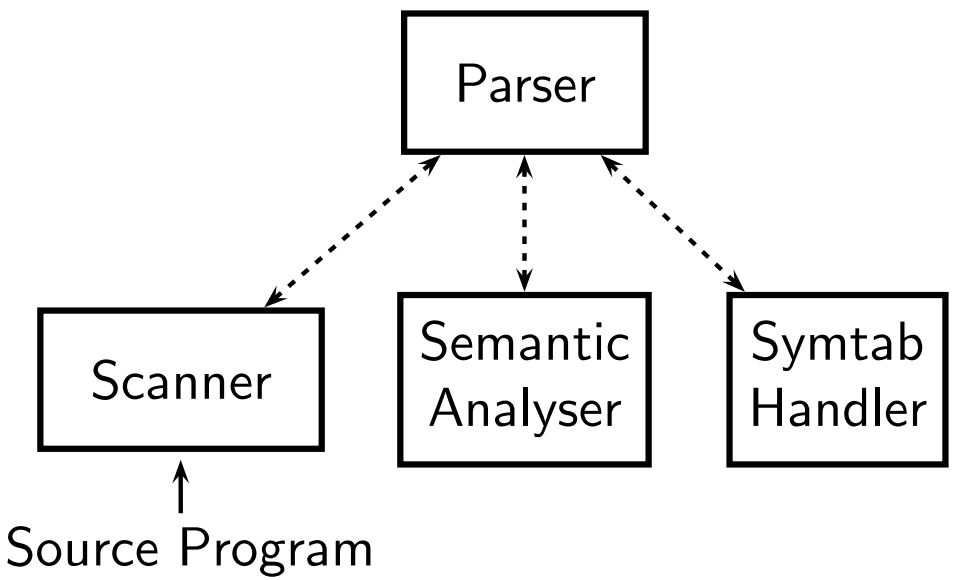
Language Processor Models



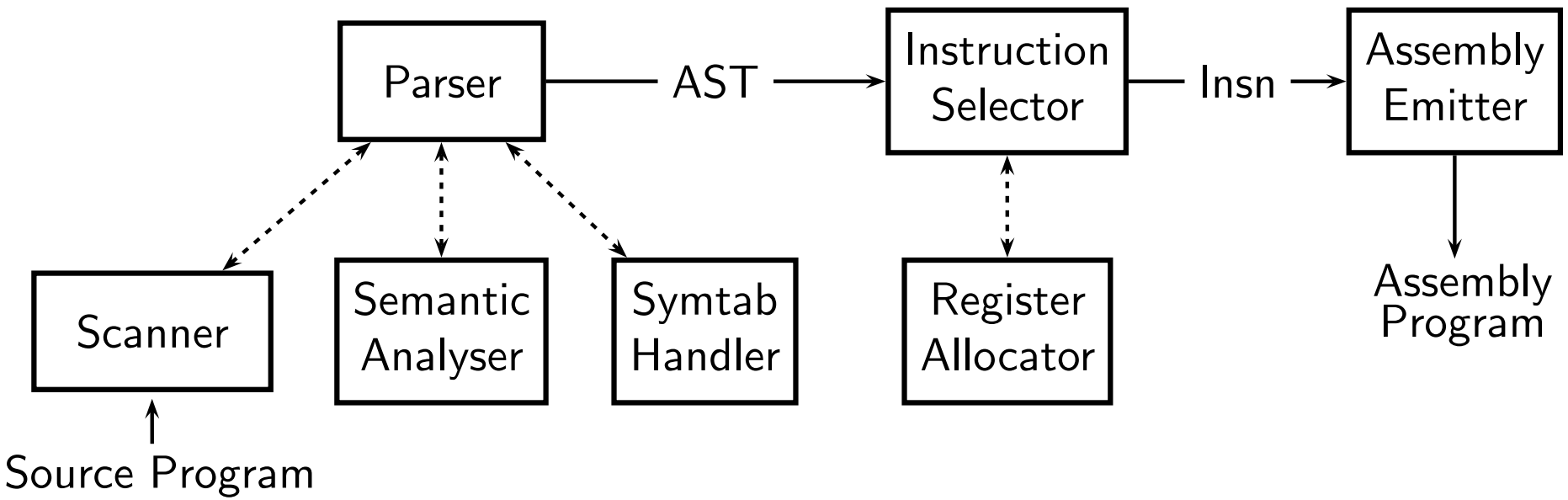
Part 2

An Overview of Compilation Phases

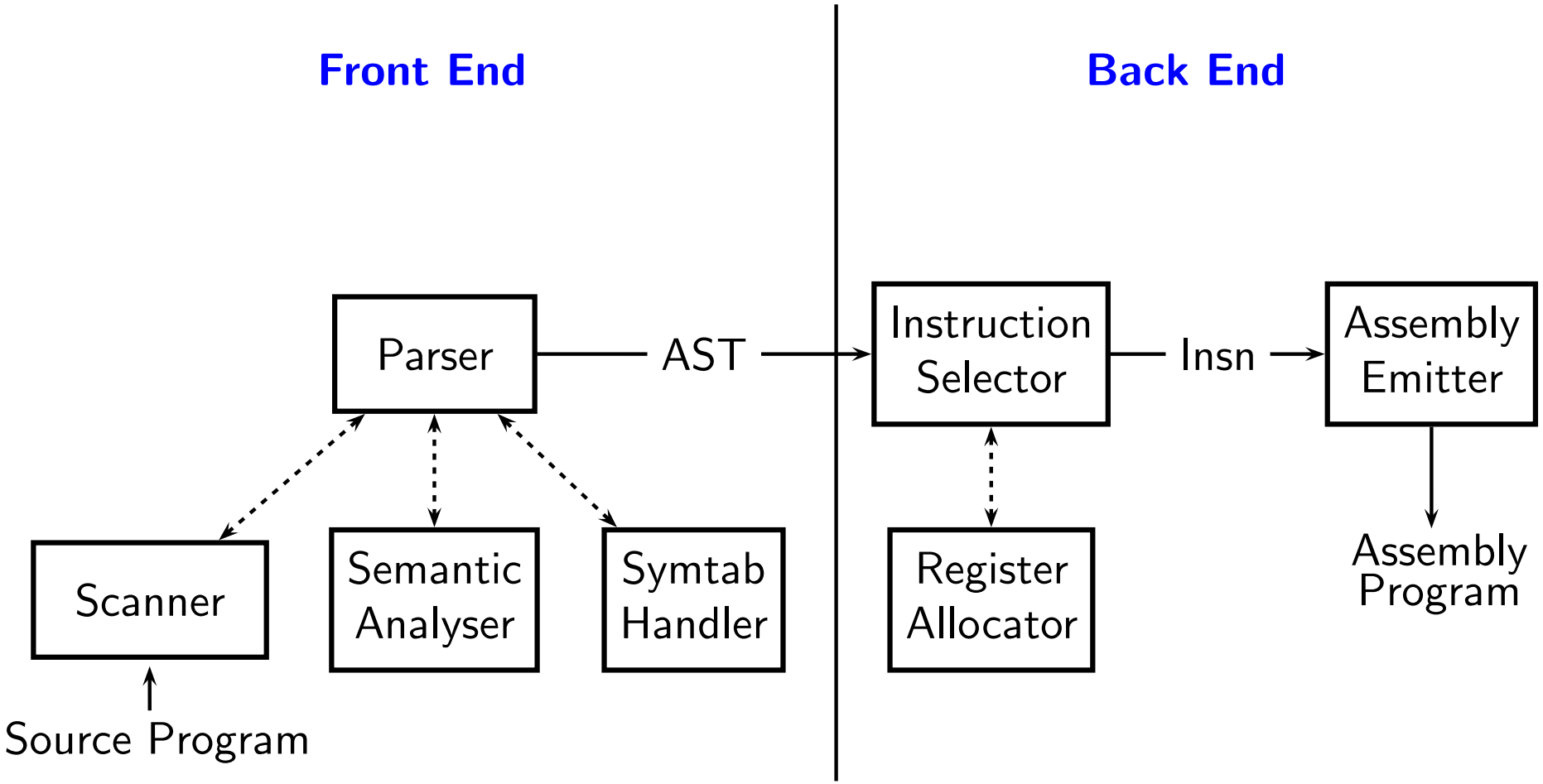
The Structure of a Simple Compiler



The Structure of a Simple Compiler



The Structure of a Simple Compiler

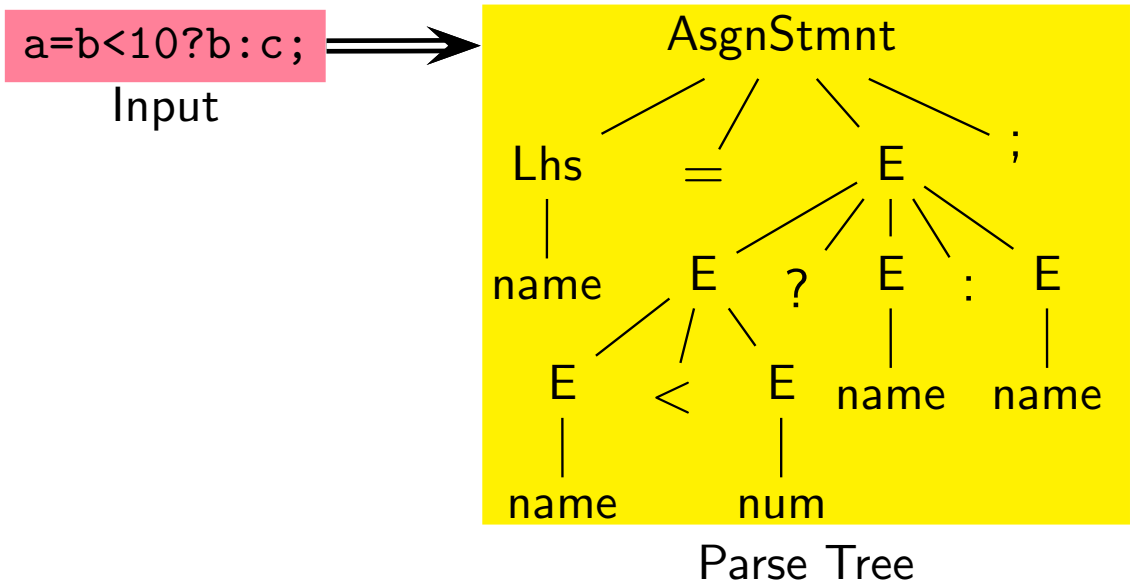


Translation Sequence in Our Compiler: Parsing

```
a=b<10?b:c;
```

Input

Translation Sequence in Our Compiler: Parsing



Parse Tree

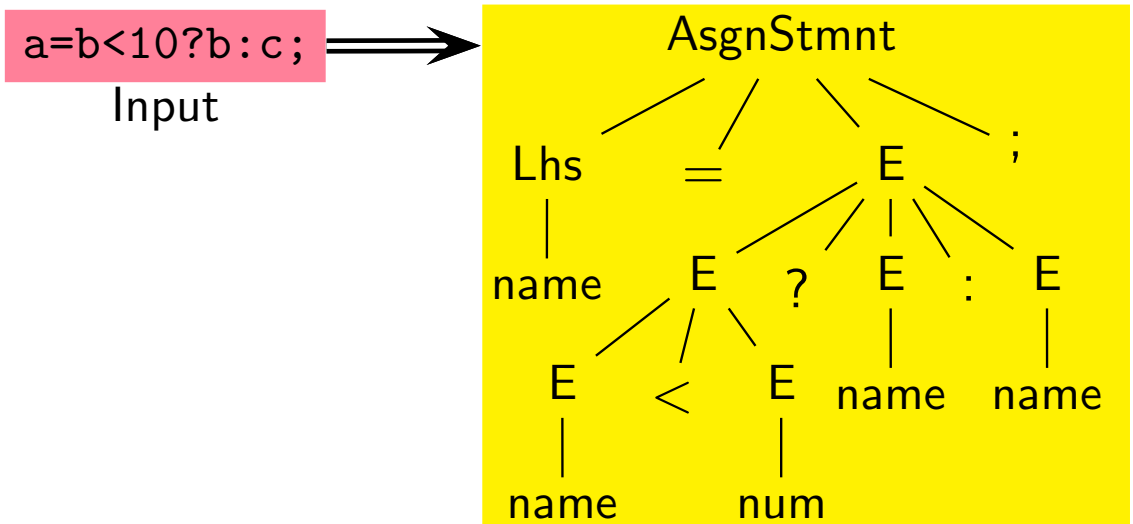
Issues:

- Grammar rules, terminals, non-terminals
- Order of application of grammar rules

  eg. is it `(a = b<10?)` followed by `(b:c)`?
- Values of terminal symbols

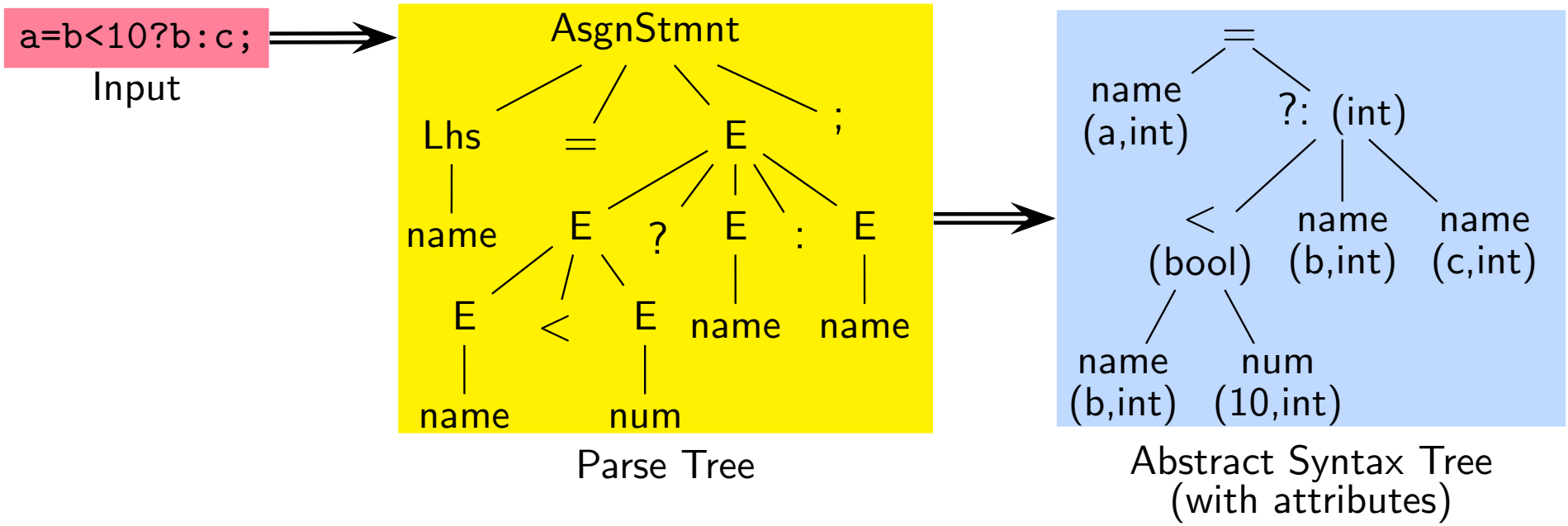
  eg. string "10" vs. integer number 10.

Translation Sequence in Our Compiler: Semantic Analysis



Parse Tree

Translation Sequence in Our Compiler: Semantic Analysis



Issues:

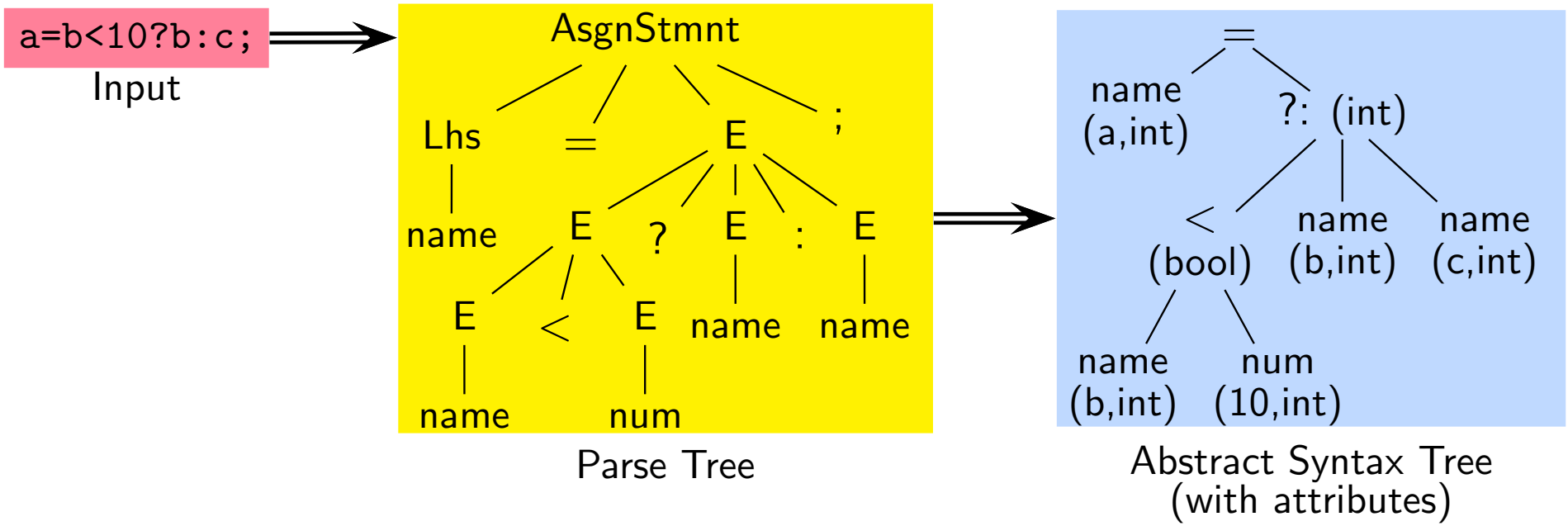
- Symbol tables

  Have variables been declared? What are their types? What is their scope?

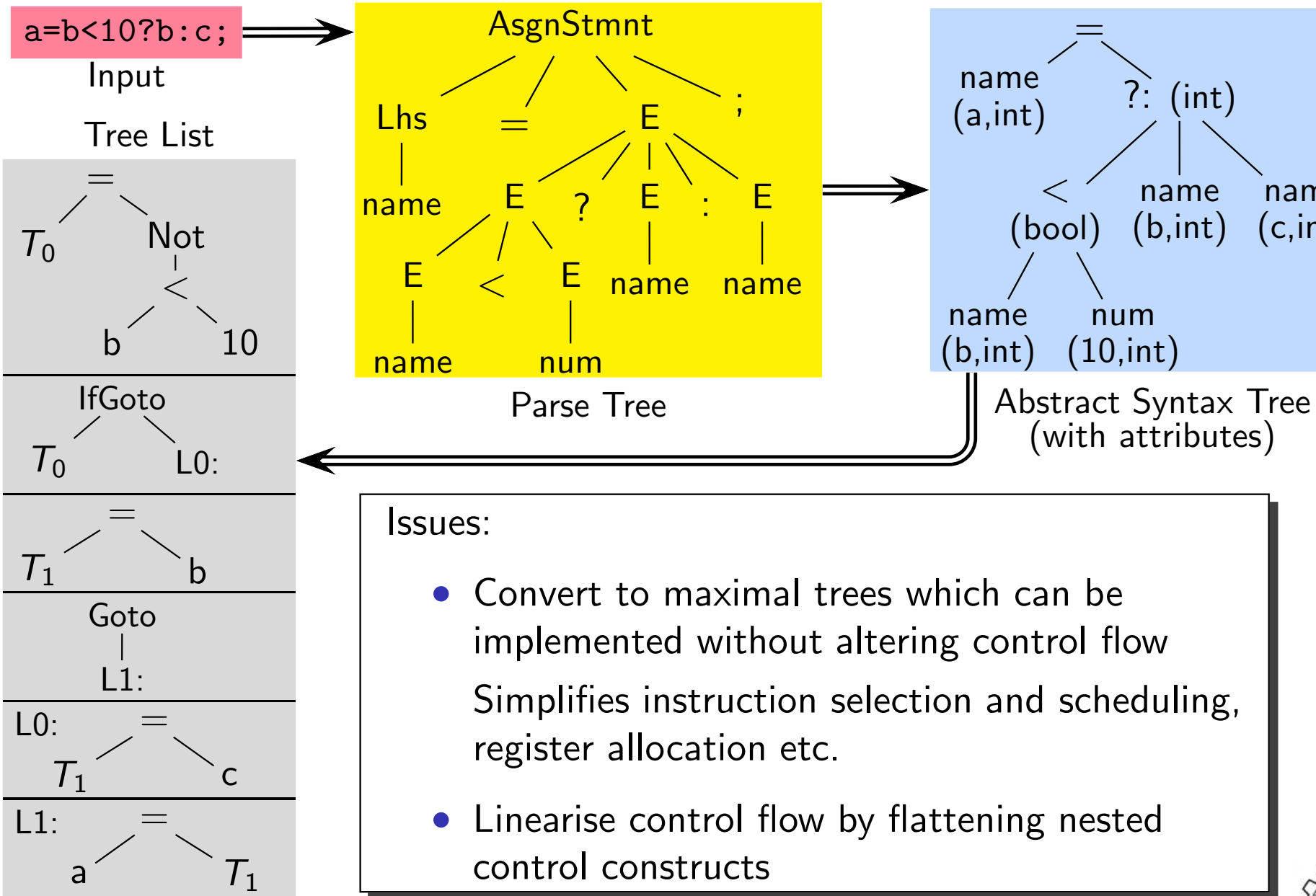
- Type consistency of operators and operands

  The result of computing b<10? is bool and not int

Translation Sequence in Our Compiler: IR Generation



Translation Sequence in Our Compiler: IR Generation



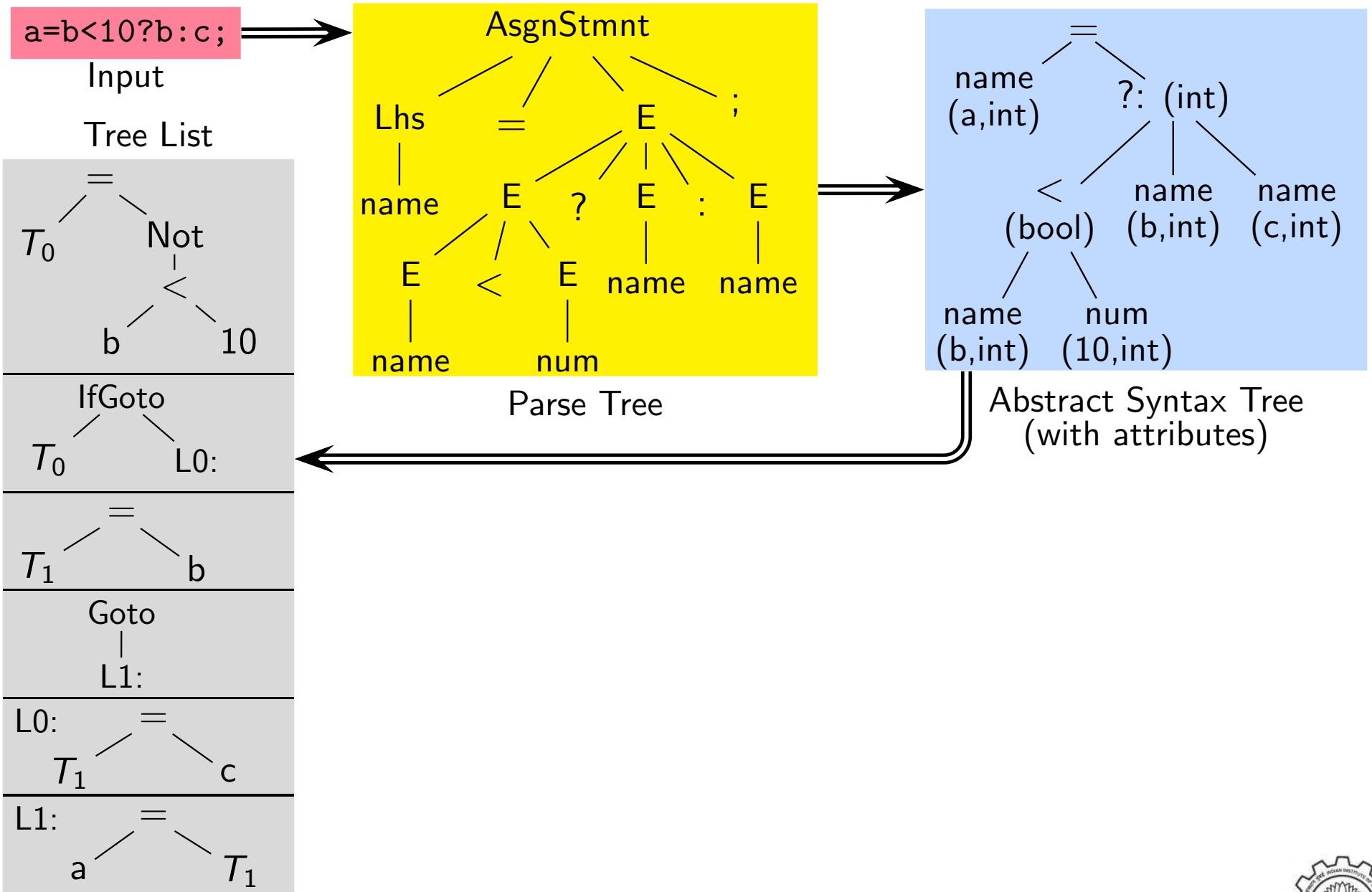
Issues:

- Convert to maximal trees which can be implemented without altering control flow

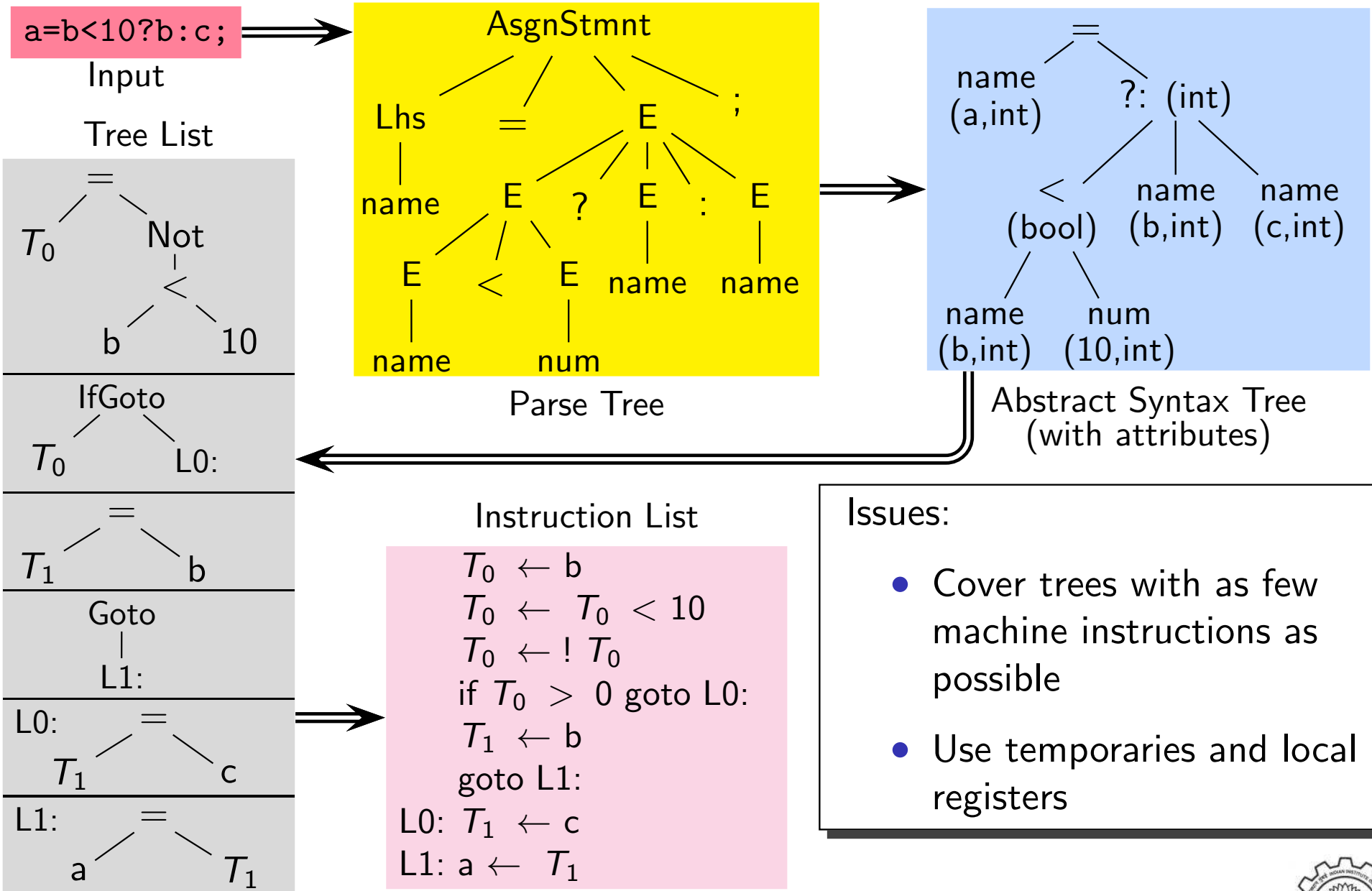
  Simplifies instruction selection and scheduling, register allocation etc.

- Linearise control flow by flattening nested control constructs

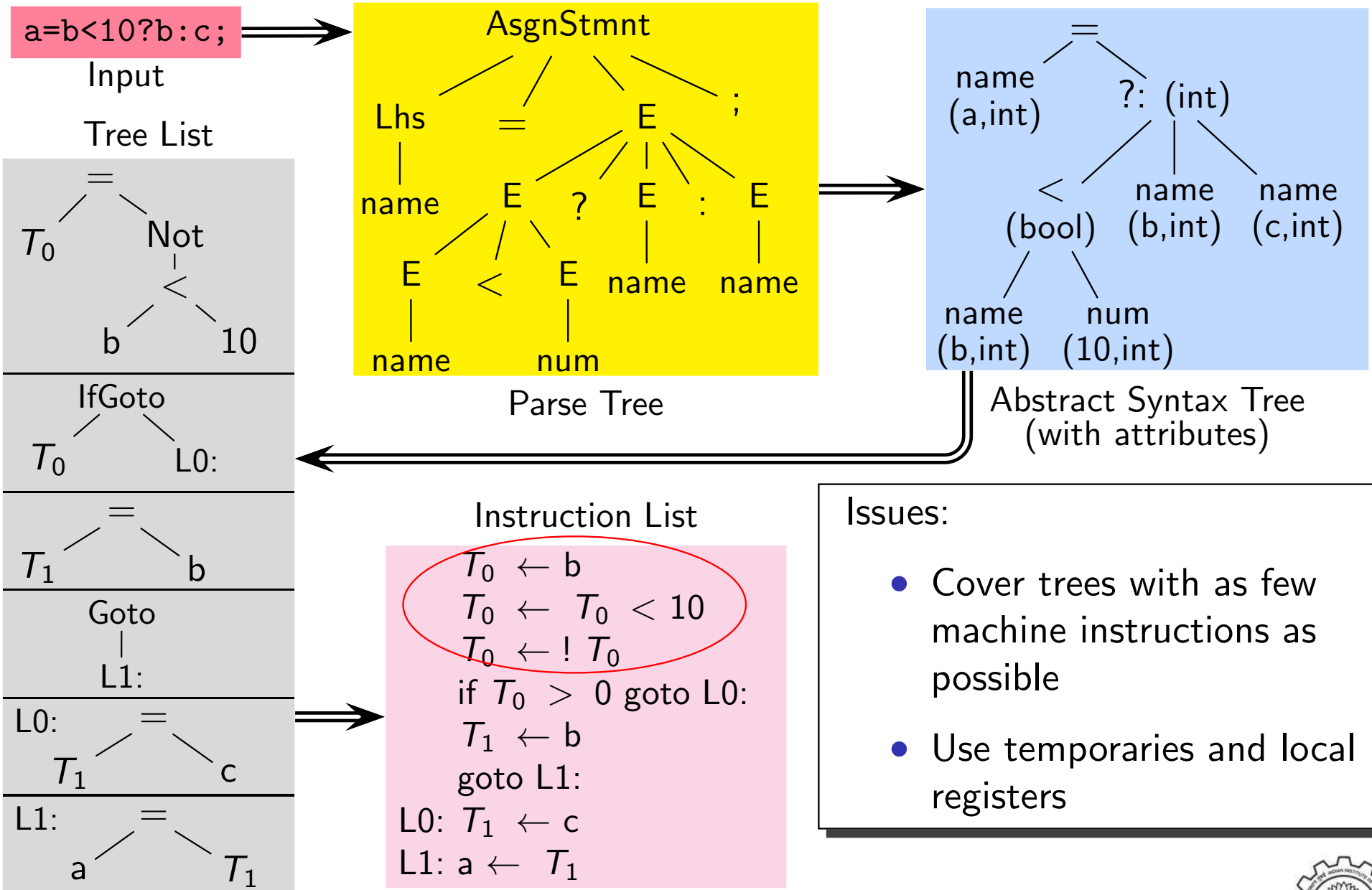
Translation Sequence in Our Compiler: Instruction Selection



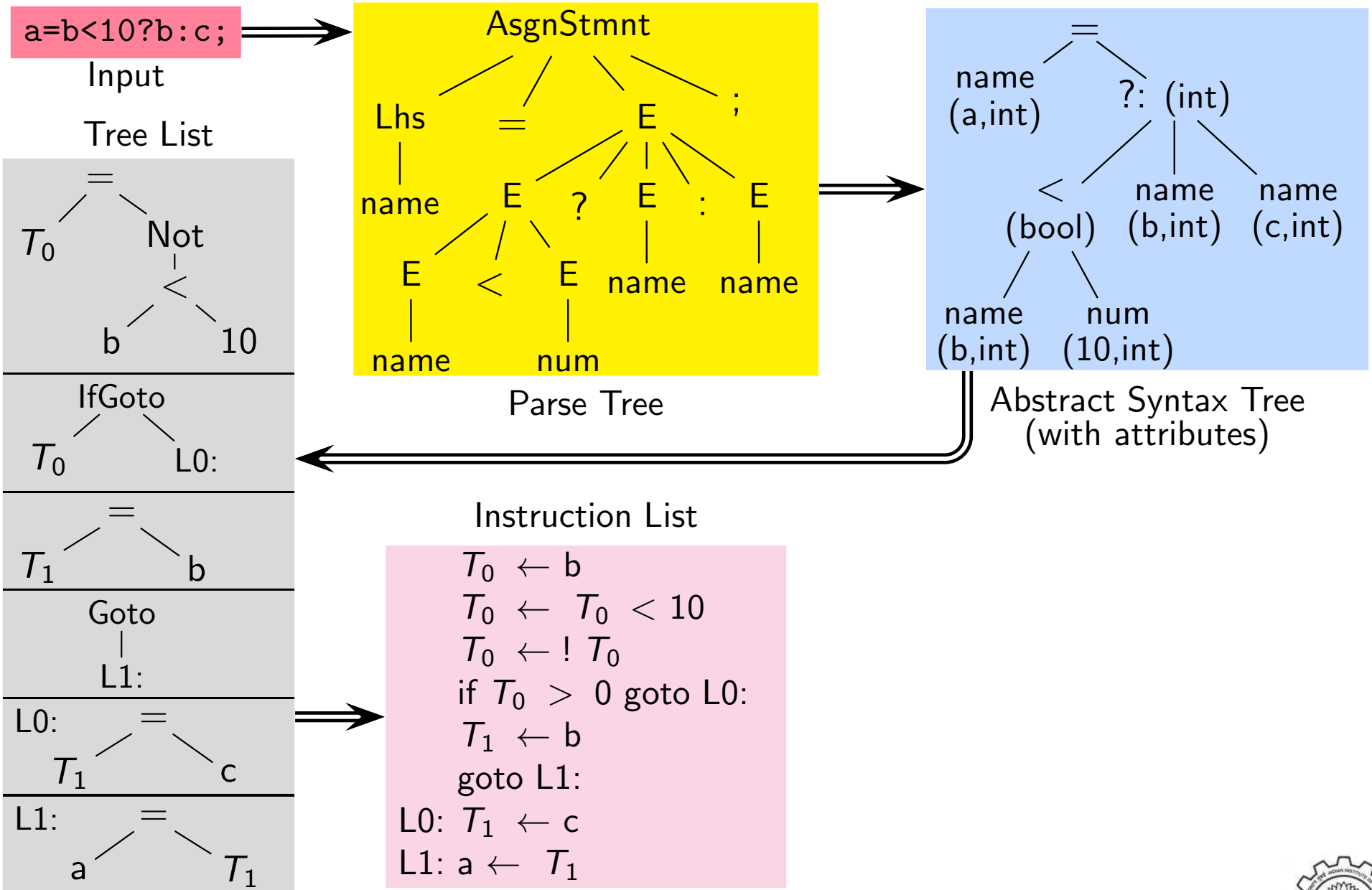
Translation Sequence in Our Compiler: Instruction Selection



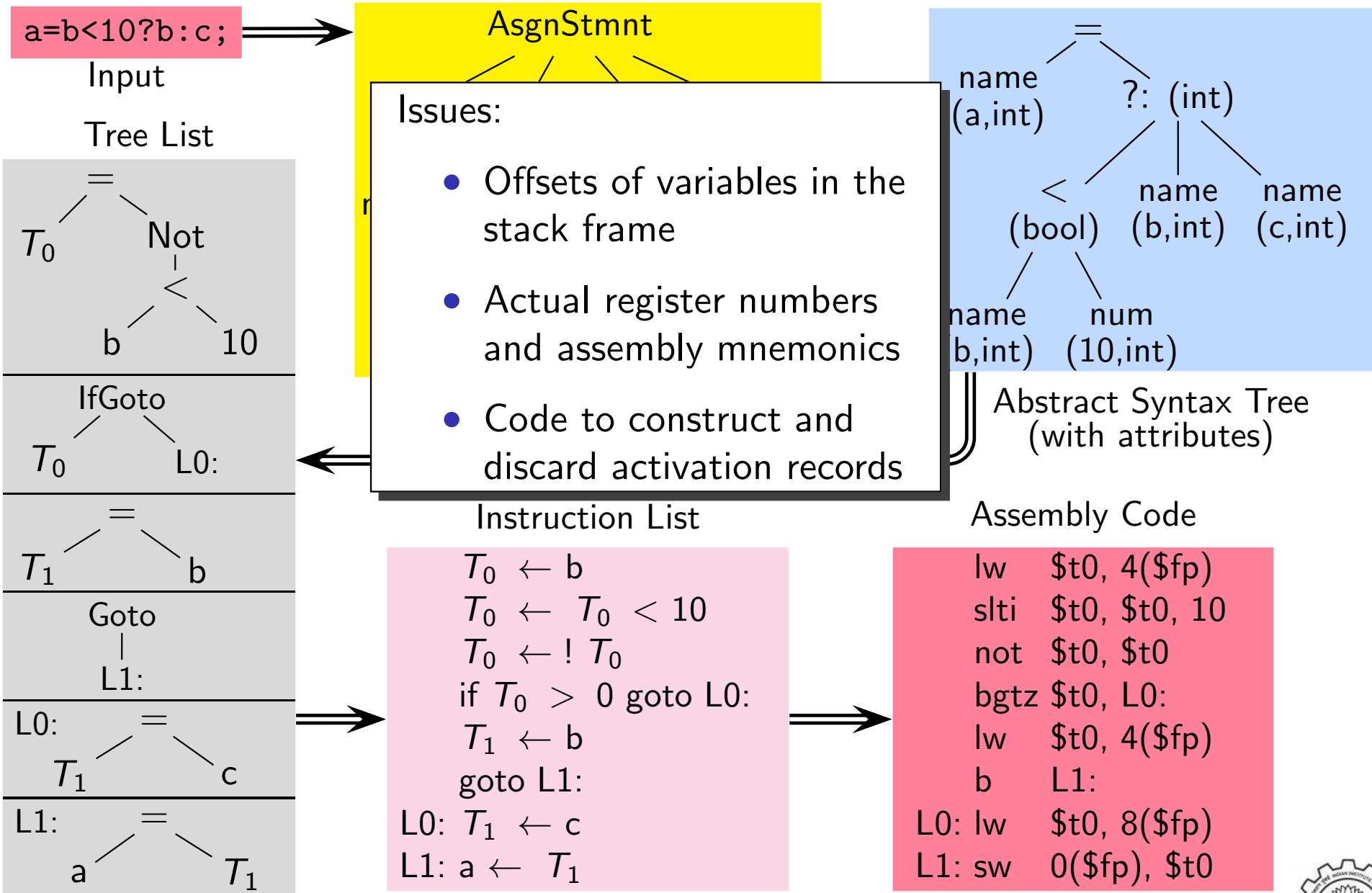
Translation Sequence in Our Compiler: Instruction Selection



Translation Sequence in Our Compiler: Emitting Instructions



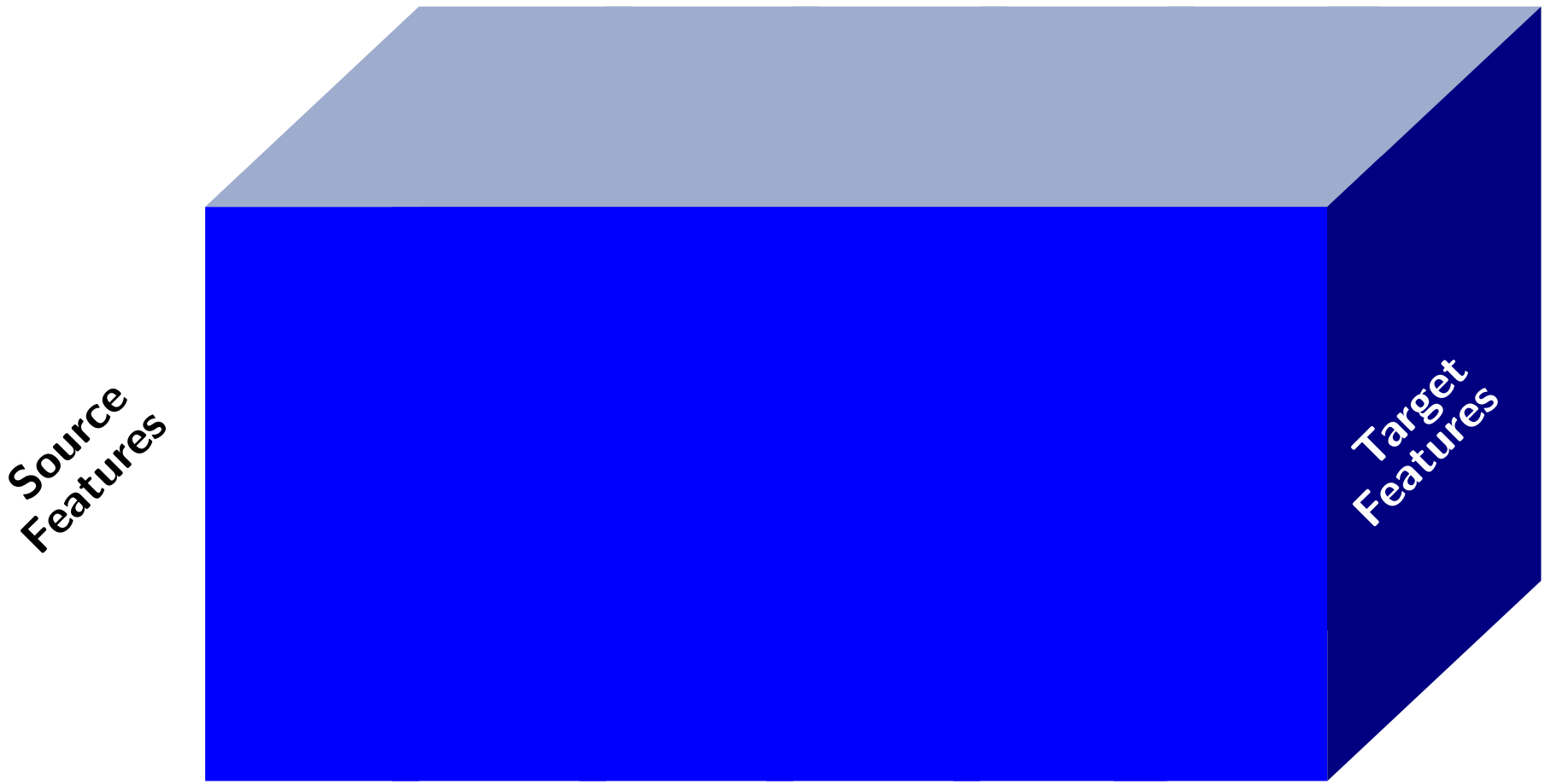
Translation Sequence in Our Compiler: Emitting Instructions



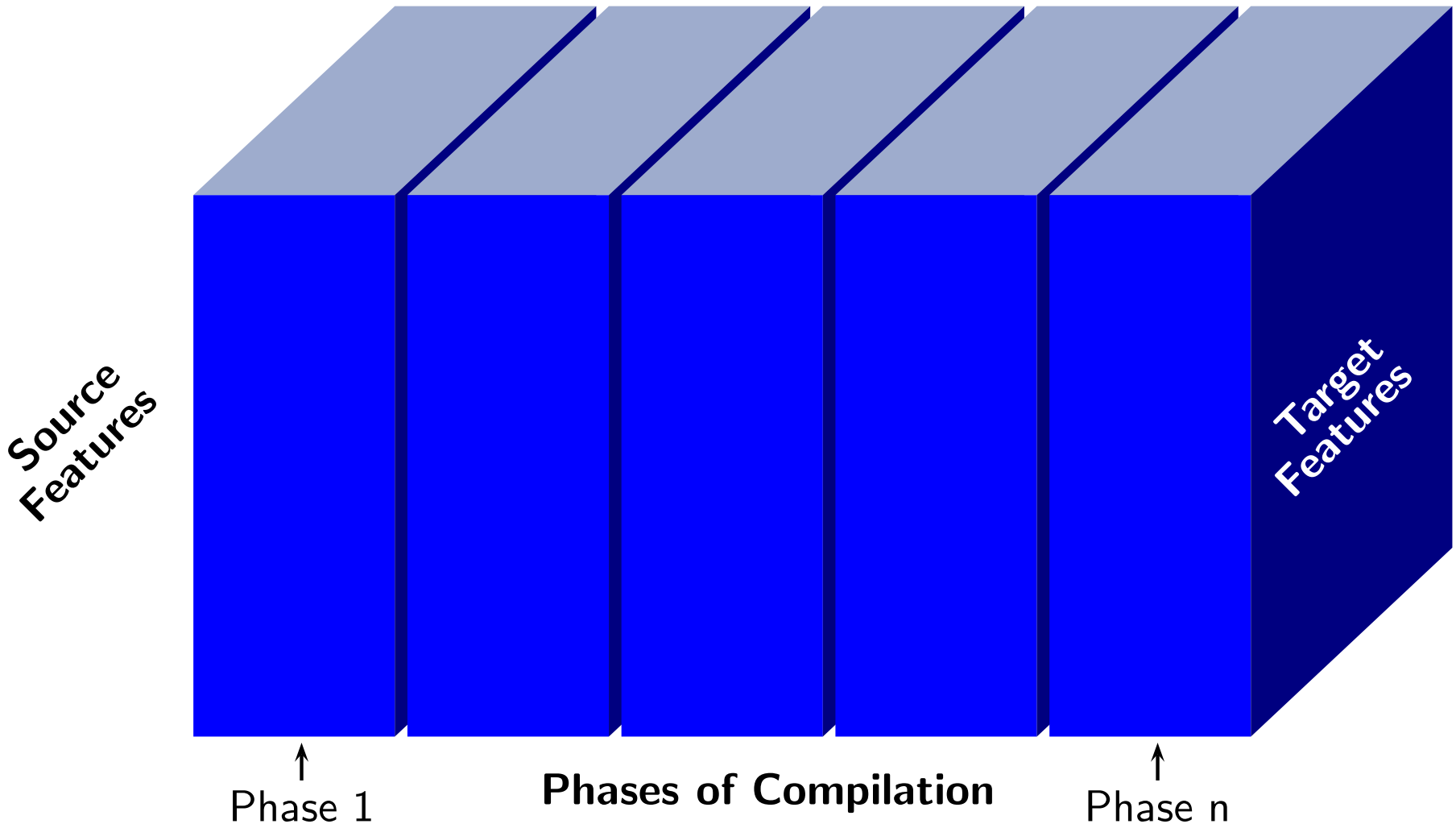
Part 3

Incremental Construction

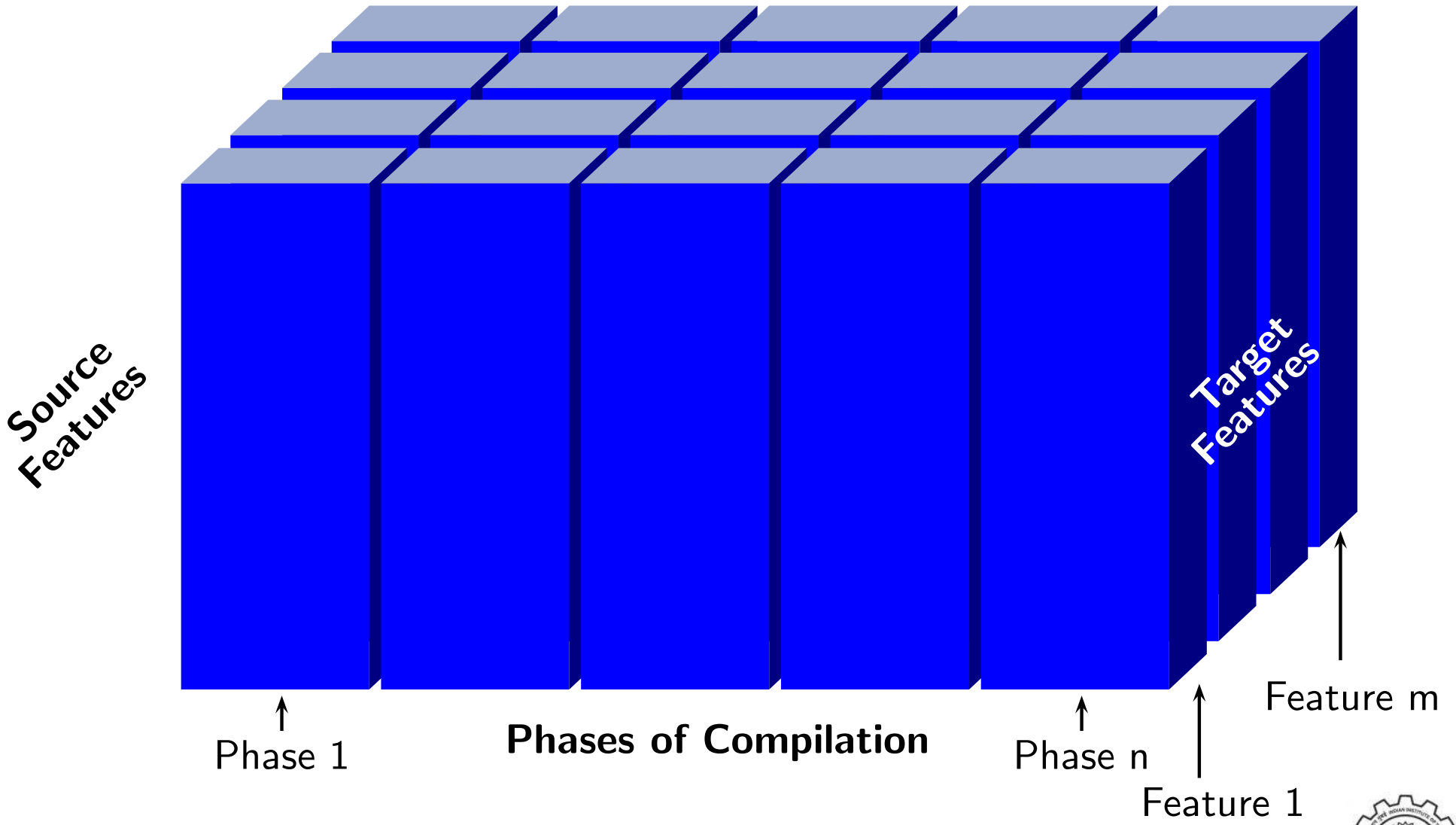
In Search of Modularity in Compilation



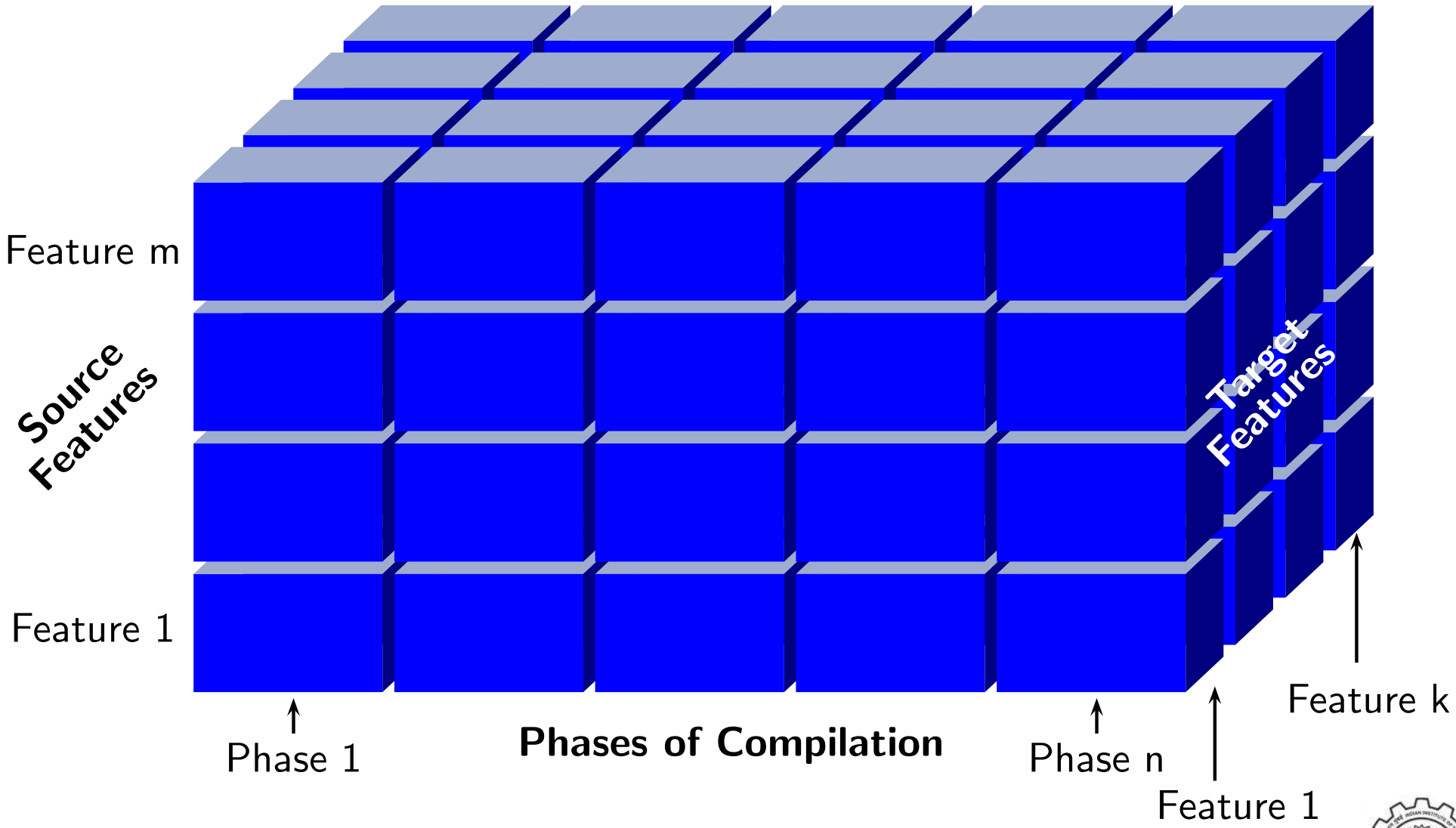
In Search of Modularity in Compilation



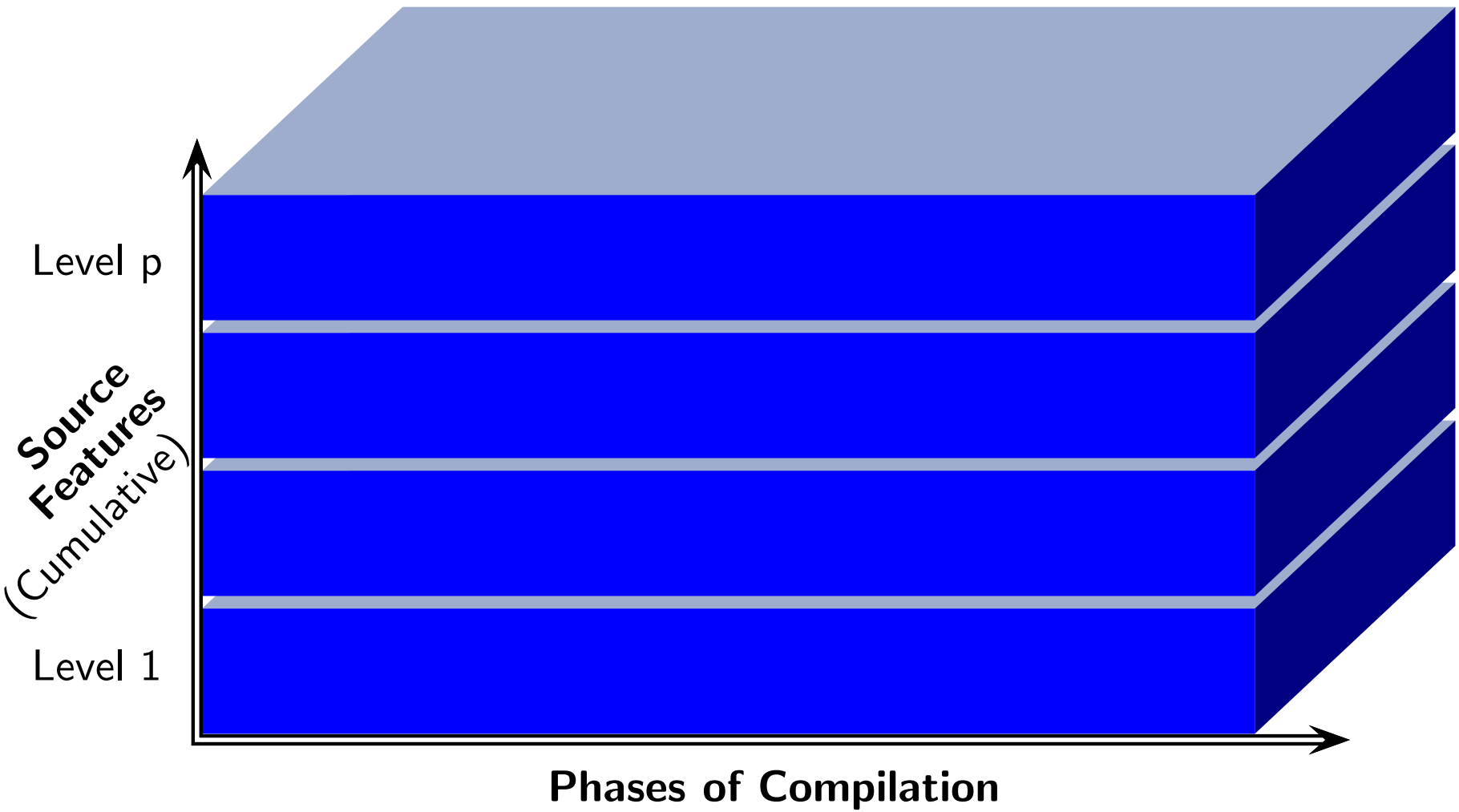
In Search of Modularity in Compilation



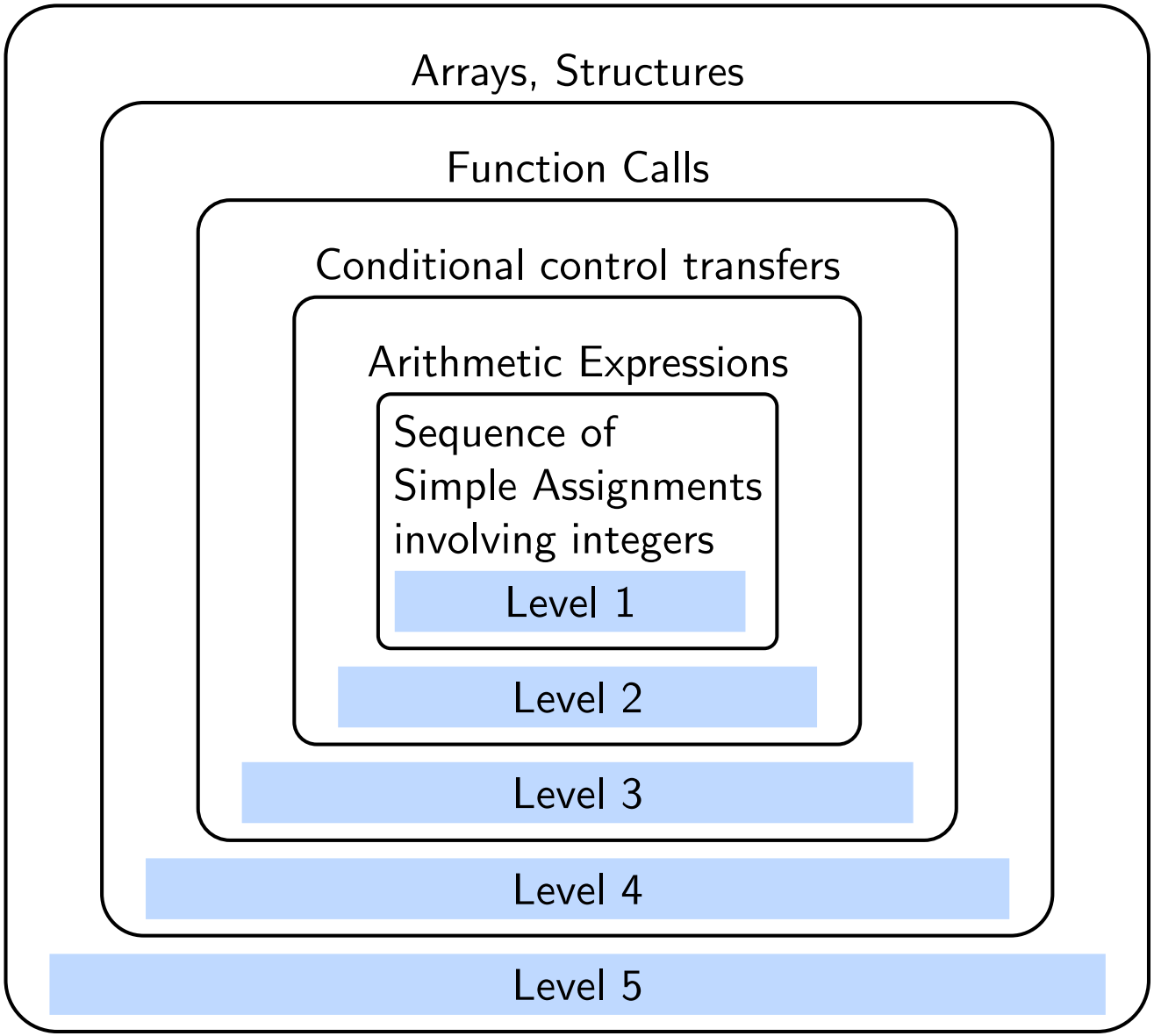
In Search of Modularity in Retargetable Compilation



In Search of Modularity in Compilation



Language Increments



Proposed Assignment Plan: Deadlines

- Submission through moodle on alternate Fridays by 5:00 pm.

Assignment	Input	Evaluation-output	Deadline
1	Level 1	AST	Fri 20 Jan
2	Level 2	AST	Fri 03 Feb
3	Level 3	AST	Fri 17 Feb
Med-Semester Examination Week			
4	Level 3	Intermediate Code Generation	Fri 10 Mar
5	Level 3	Spim code	Fri 24 Mar
6	Level 4	Spim code	Fri 10 Apr

Incentives for AP grade

Handle additional features

- Arrays and structs
- Input output
- Control flow graphs and data flow analysis

Proposed Assignment Plan: Modalities

- Groups of two each
- A reference implementation will be provided for each assignment
- Base source code will be provided for the first assignment

  For other assignments, a library module will be provided for phases that are not to be implemented

Proposed Assignment Plan: Evaluation

- Evaluation will be by running diff on the output
  - Standard file names and directory names must be used
  - It will not be possible to entertain violations
  - Use your creativity inside your code, not in file names, Makefile commands and program output.
- Some marks will be reserved for
  - Participation in moodle discussion forum
  - Reporting bugs in the reference implementation on moodle

Questions

Assignment 0

- You are given five C programs
- Observe the following outputs generated by gcc

Representation	Enabling Switch
GIMPLE	`-fdump-tree-gimple`
CFG	`-fdump-tree-cfg`
Assembly	`-S`