Mini Workshop on GCC Internals

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

GCC Resource Center,
Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

January 2008

Part 1

Outline

Outline

- Open Source Vs. Free Software
- GCC Internals
- Major Research Initiatives
- Conclusions

Part 2

Open Source Vs. Free Software

Open Source Vs. Free Software

- The open source initiative: (http://www.opensource.org/)
  Emphasis on development methodology
- The Free Software Foundation: (http://www.fsf.org/)
  Emphasis on freedom of the user
- In some cases, open source software has restricted user freedom

Open Source and Free Software Development Model

- The Cathedral and the Bazaar

  Eric S Raymond, 1999.
- Cathedral: Total Centralized Control

  Design, implement, test, release
- Bazaar: Total Decentralization

  Release early, release often, let users fix bugs

The Bazaar Approach

Release early, release often, let users fix bugs

- Brooks' law (The Mythical Man Month, 1975)

The Bazaar Approach

Release early, release often, let users fix bugs

- Brooks' law (The Mythical Man Month, 1975)
  - 12 man month effort

The Bazaar Approach

Release early, release often, let users fix bugs

- Brooks' law (The Mythical Man Month, 1975)
  - 12 man month effort
  - 1 person working for 12 months
    OR
    12 persons working for 1 month?

The Bazaar Approach

Release early, release often, let users fix bugs

- Brooks' law (The Mythical Man Month, 1975)
  - 12 man month effort
  - 1 person working for 12 months
    OR
    12 persons working for 1 month?
- Bazaar approach believes that the two are somewhat equivalent in internet-based distributed development.

The Bazaar Approach

Release early, release often, let users fix bugs

- Brooks' law (The Mythical Man Month, 1975)
  - 12 man month effort
  - 1 person working for 12 months
    OR
    12 persons working for 1 month?
- Bazaar approach believes that the two are somewhat equivalent in internet-based distributed development.
- "Given enough eyeballs, all bugs are shallow".

  Code errors, logical errors, and architectural errors.

The Bazaar Approach

Release early, release often, let users fix bugs

- Brooks' law (The Mythical Man Month, 1975)
  - 12 man month effort
  - 1 person working for 12 months
    OR
    12 persons working for 1 month?
- Bazaar approach believes that the two are somewhat equivalent in internet-based distributed development.
- "Given enough eyeballs, all bugs are shallow".

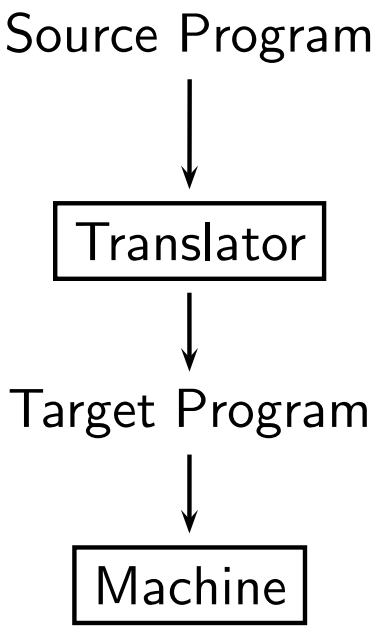
  Code errors, logical errors, and architectural errors.

A combination of the two seems more sensible

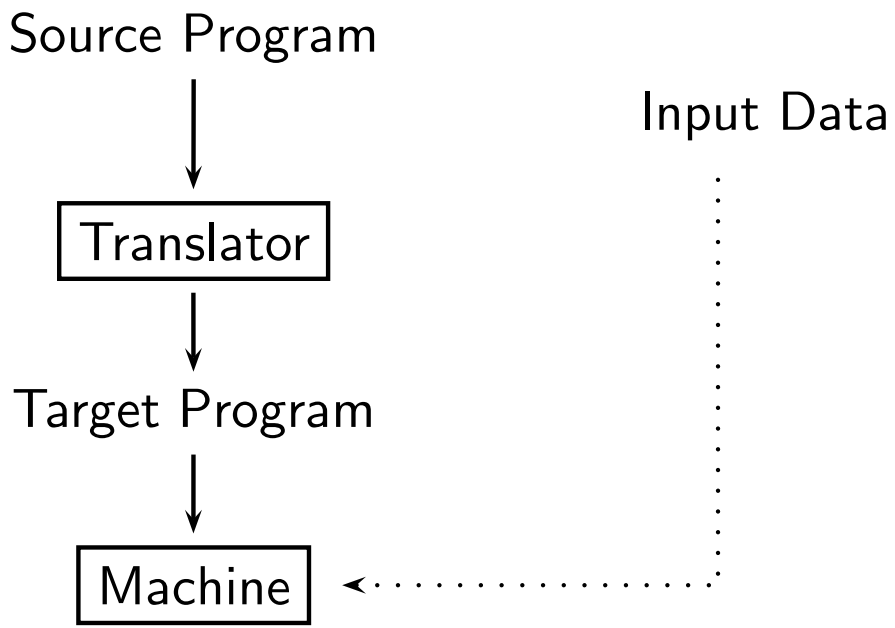
Part 3

Introduction to Compilation

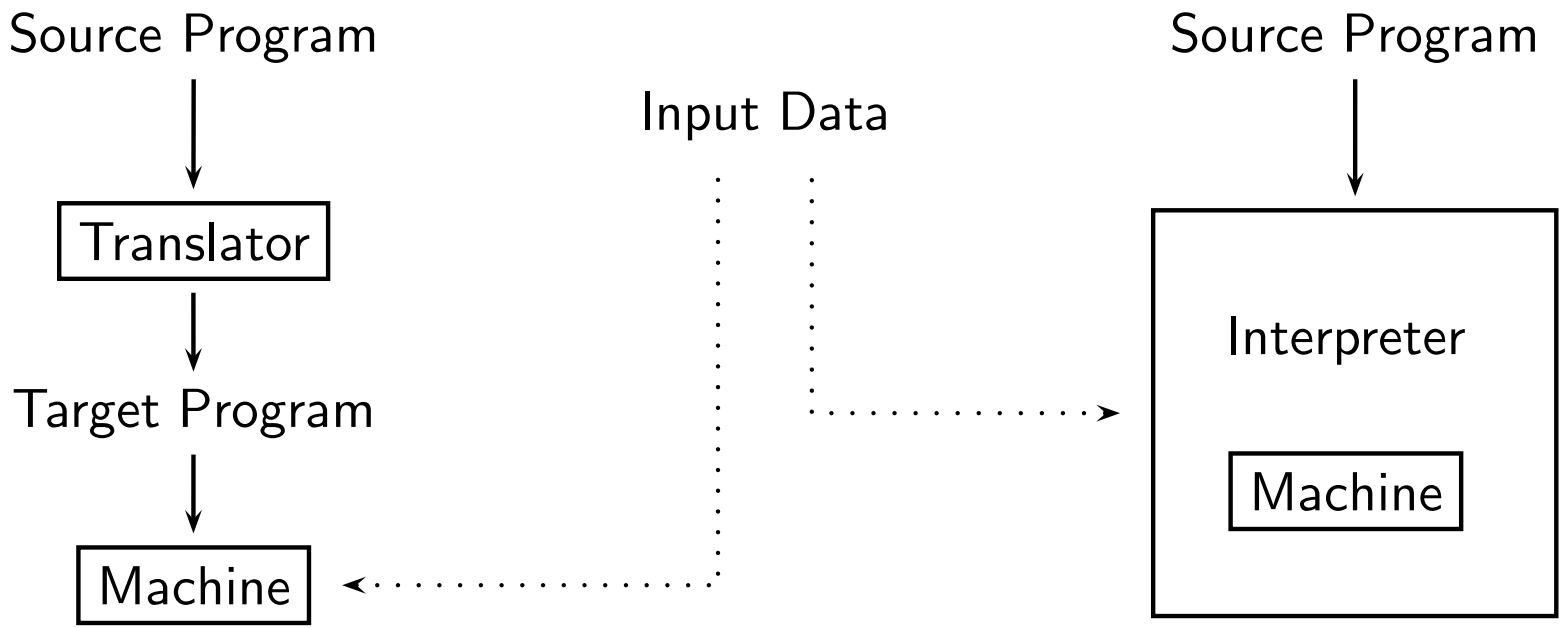
Implementation Mechanisms



Implementation Mechanisms

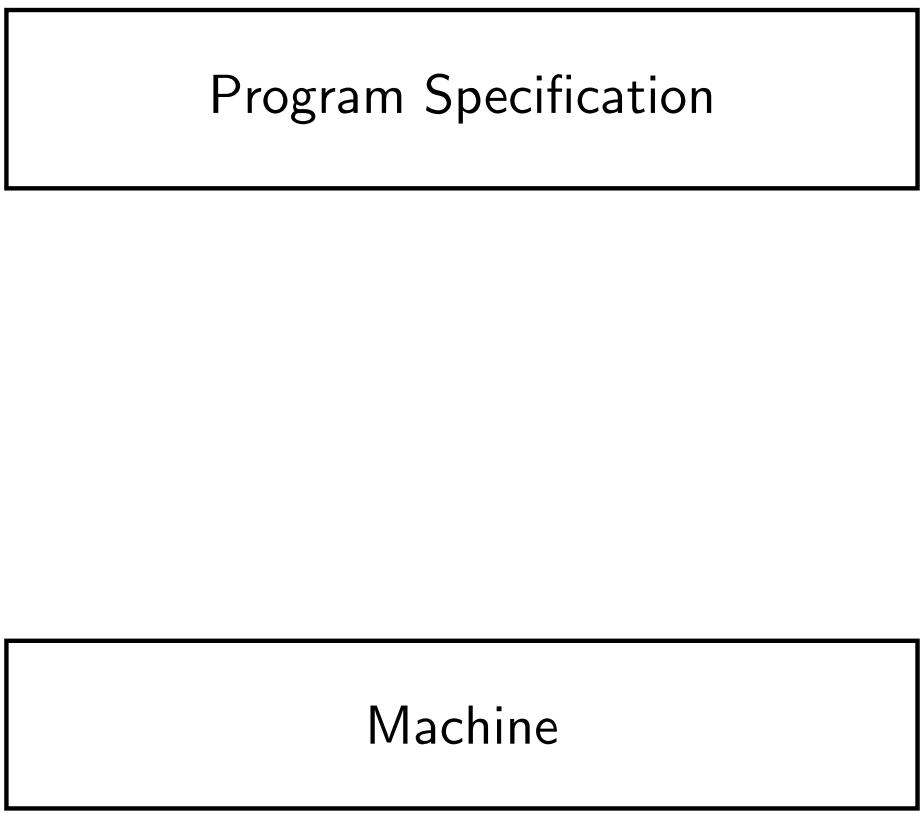


Implementation Mechanisms



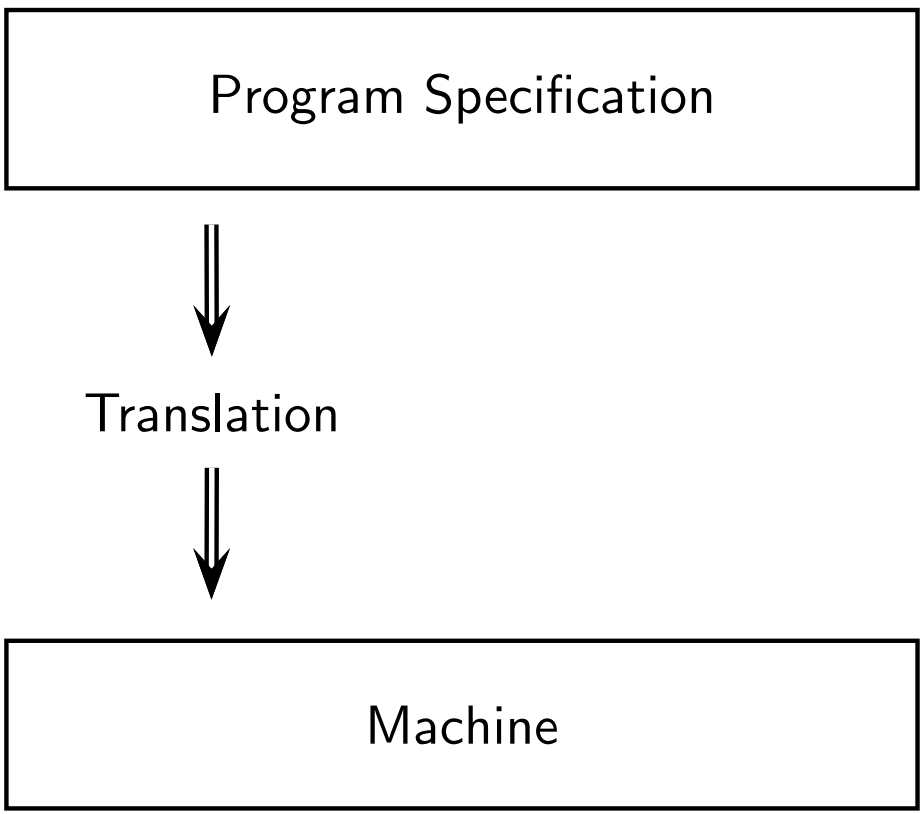
Implementation Mechanisms as "Bridges"

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



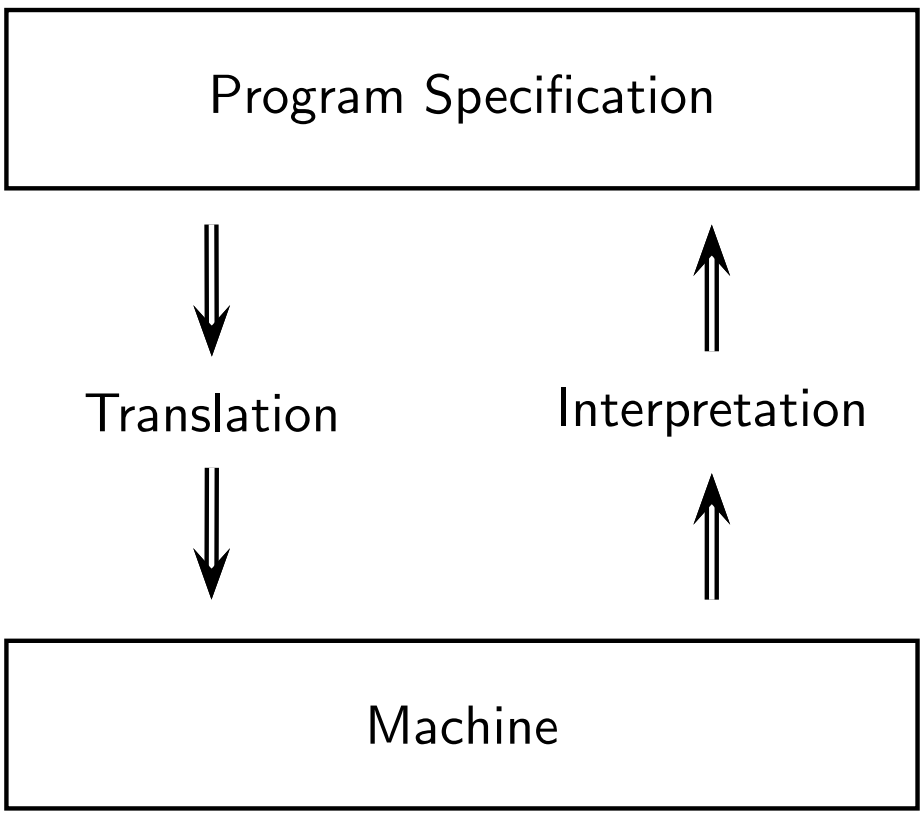
Implementation Mechanisms as "Bridges"

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



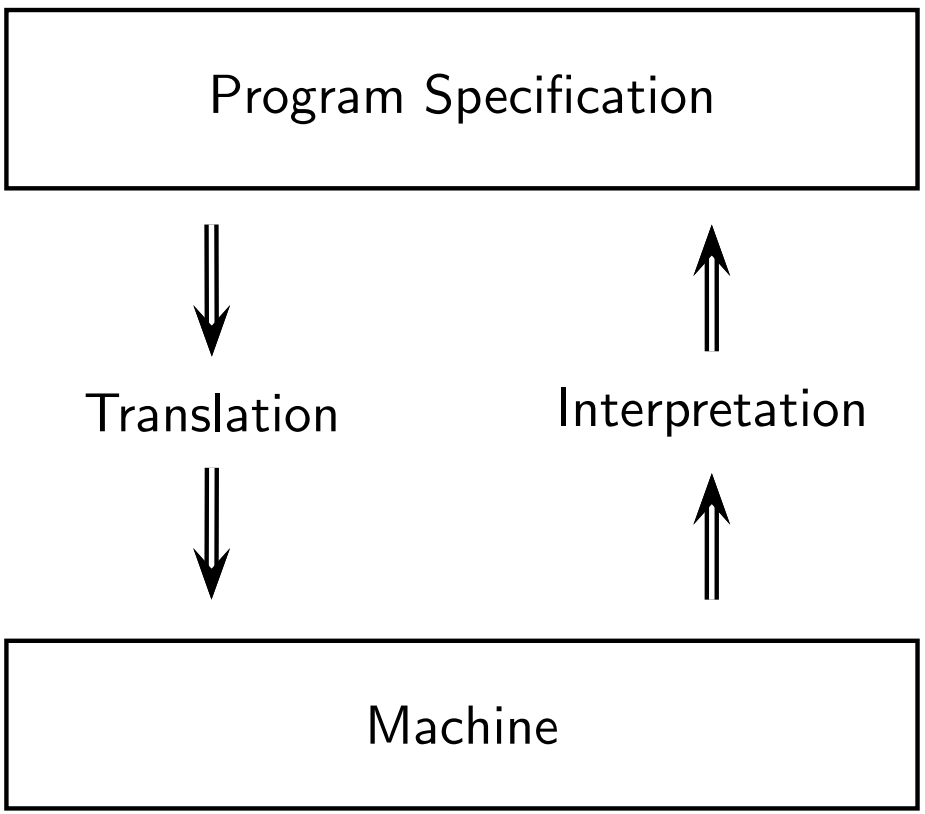
Implementation Mechanisms as "Bridges"

- "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

- A source statement

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

- Spim assembly equivalent

```
        lw   $t0, 4($fp)    #       $t0 <- b
        slti $t0, $t0, 10   #       $t0 <- $t0 < b
        not  $t0, $t0       #       $t0 <- ! $t0
        bgtz $t0, L0:       #       if $t0 >=  0 goto L0:
        lw   $t0, 4($fp)    #       $t0 <- b
        b    L1:            #       goto L1:
L0:     lw   $t0, 8($fp)    # L0:   $t0 <- c
L1:     sw   0($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

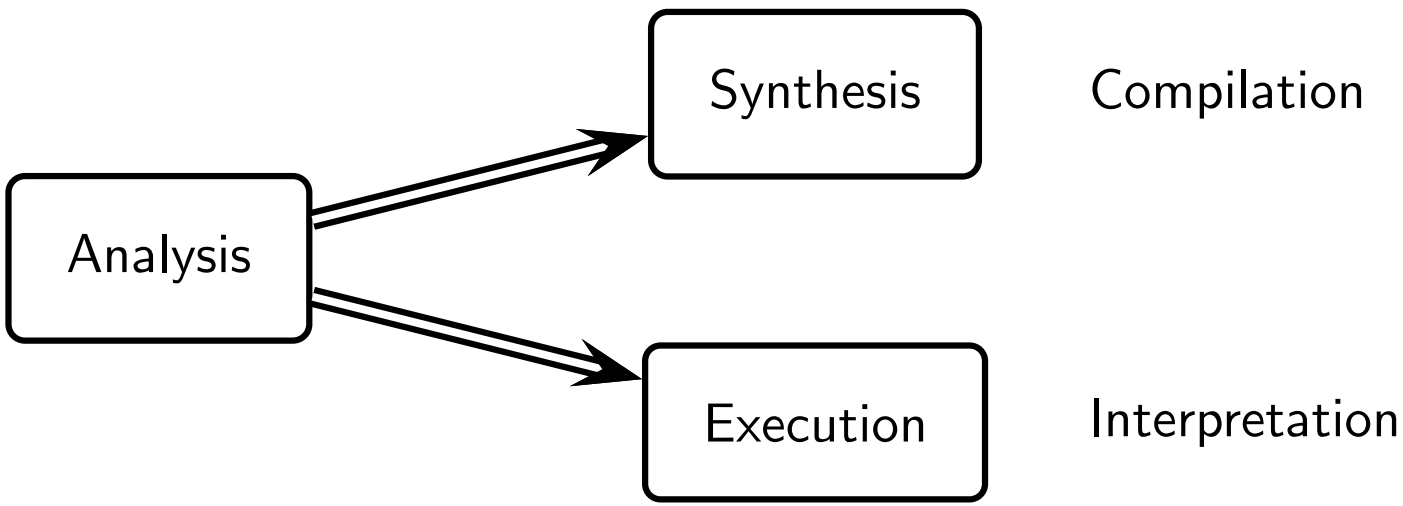
- Translation = Analysis + Synthesis

  Interpretation = Analysis + Execution

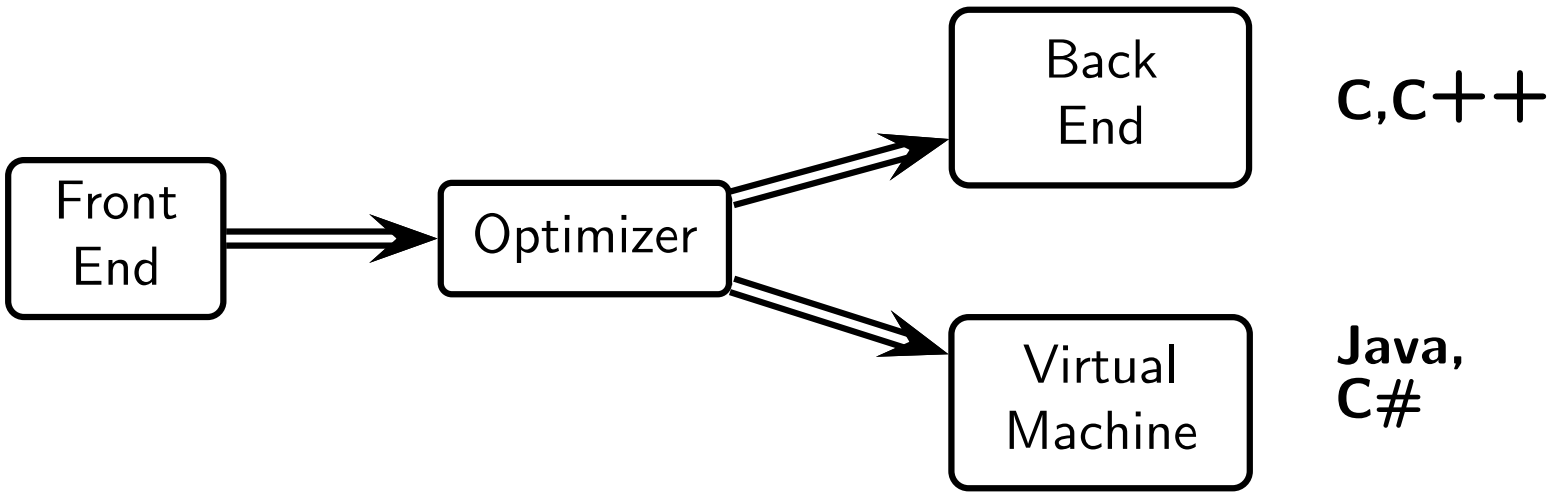
- Translation Instructions $\Longrightarrow$ Equivalent Instructions

  Interpretation Instructions $\Longrightarrow$ Actions Implied by Instructions

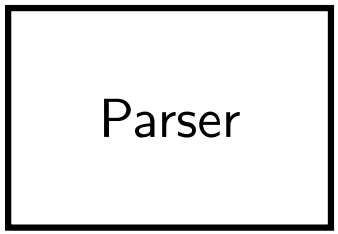
Language Implementation Models



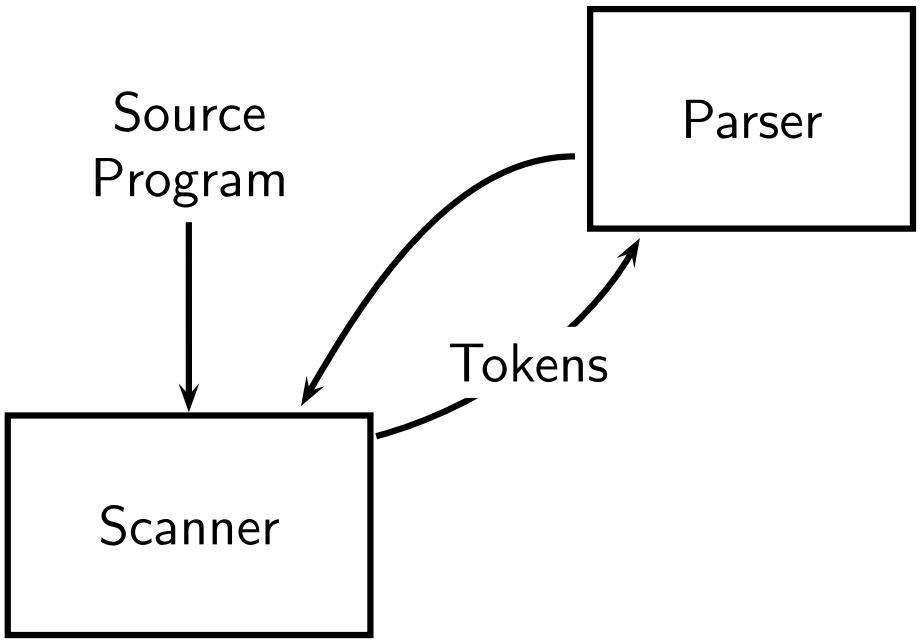
Language Processor Models



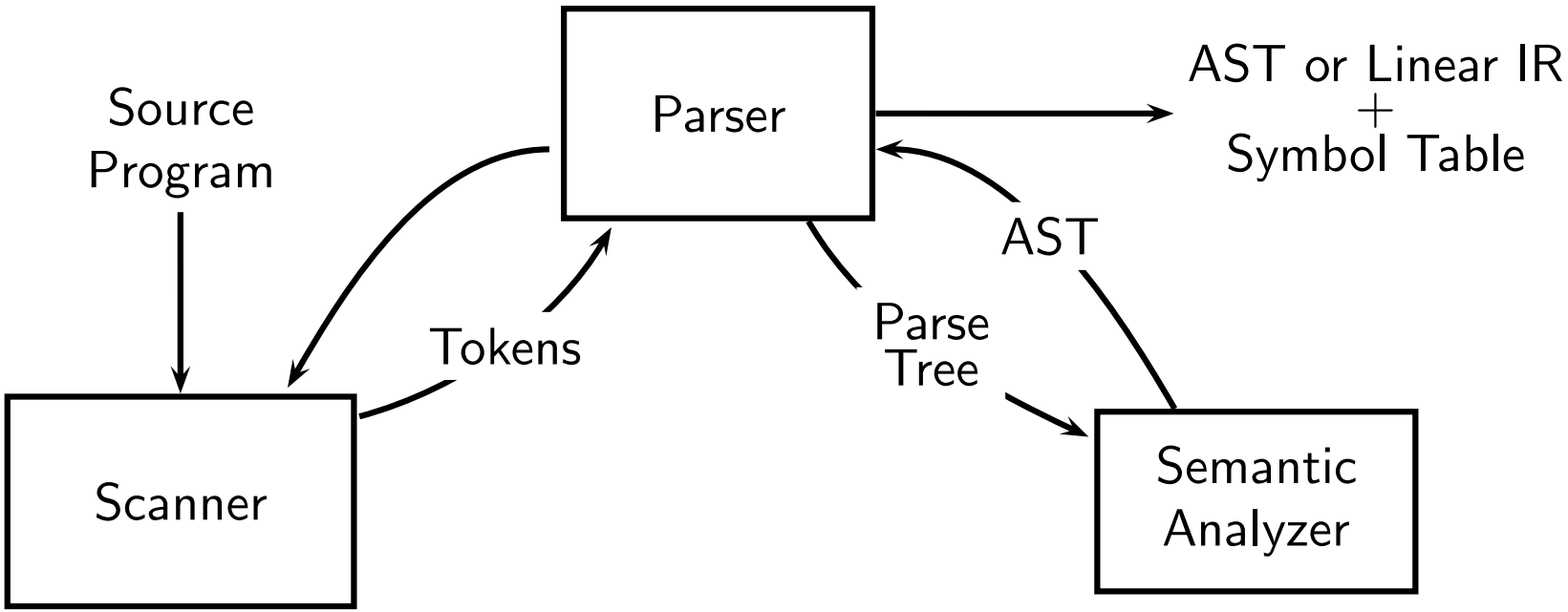
Typical Front Ends



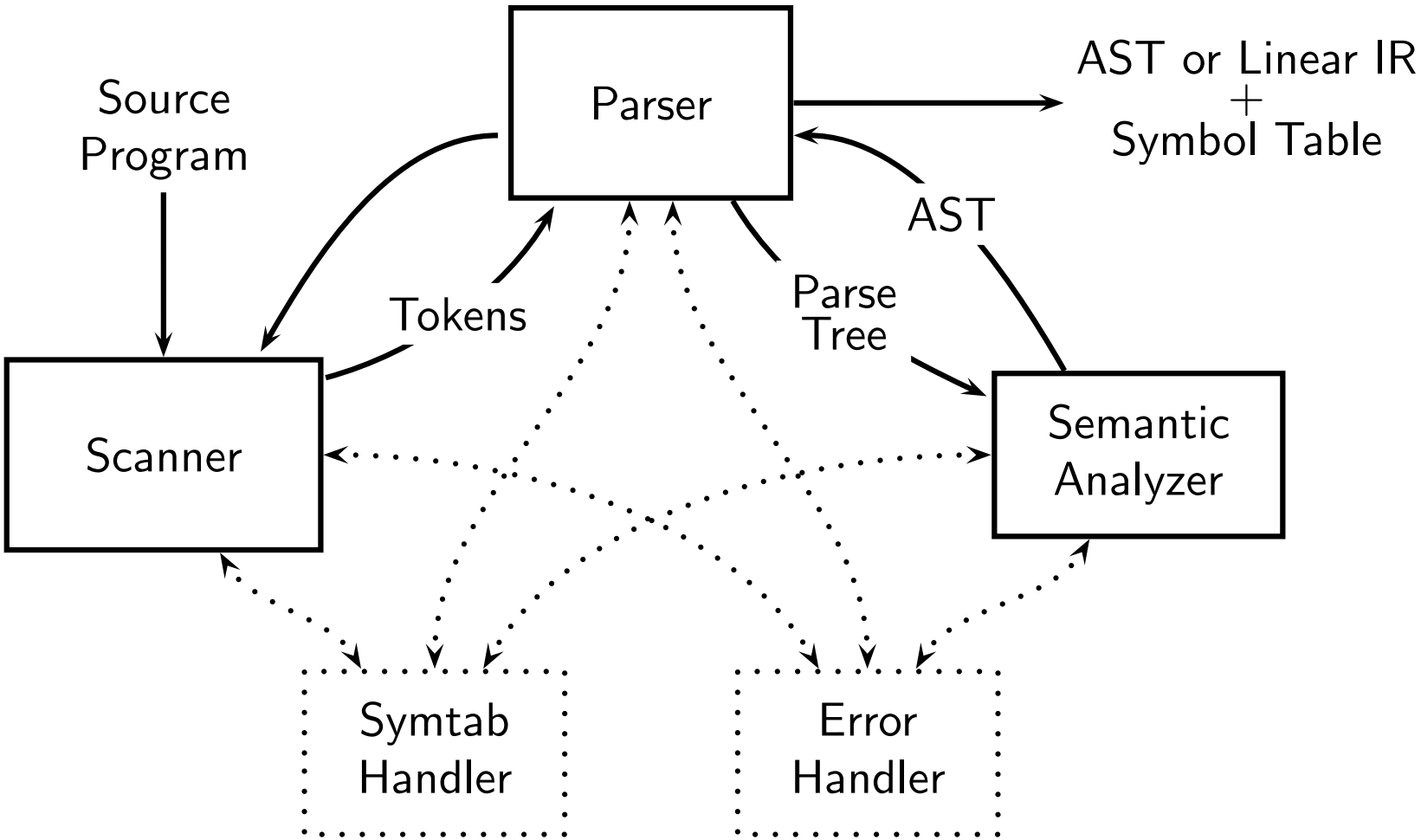
Typical Front Ends



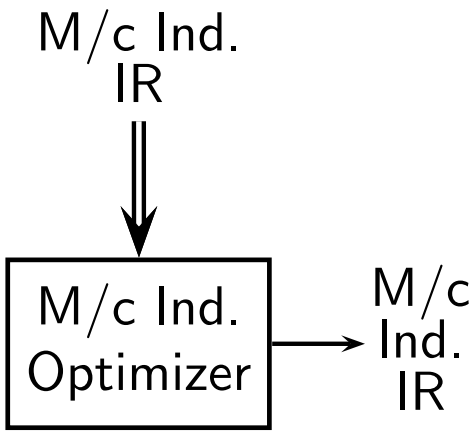
Typical Front Ends



Typical Front Ends

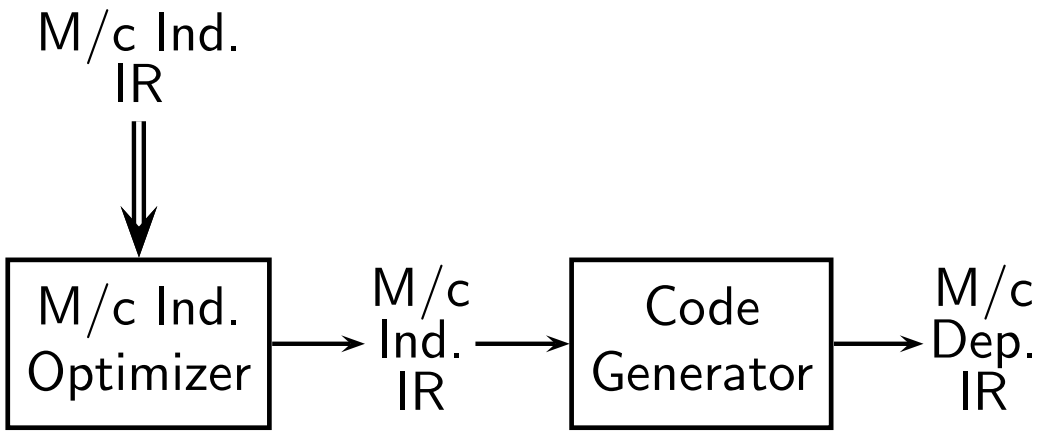


Typical Back Ends



- Compile time evaluations
- Eliminating redundant computations

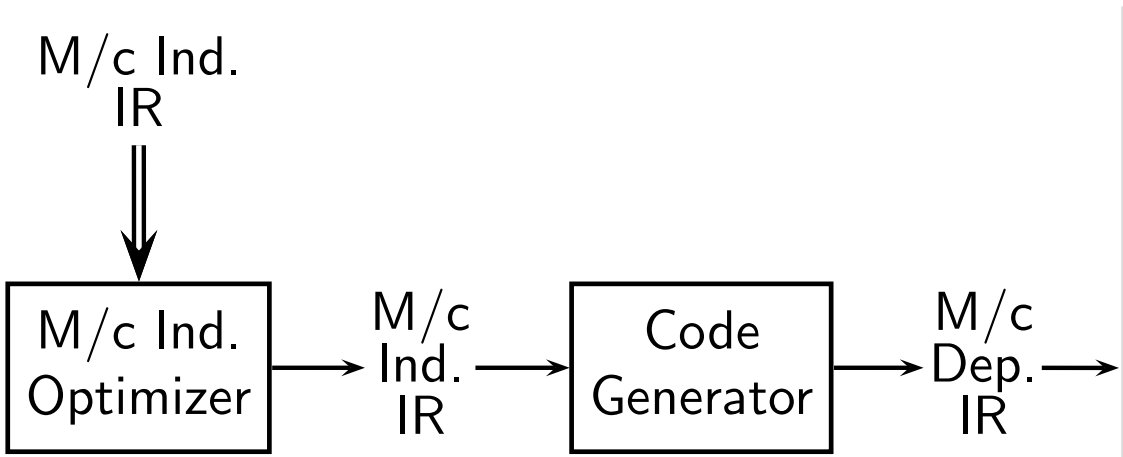
Typical Back Ends



- Compile time evaluations
- Eliminating redundant computations

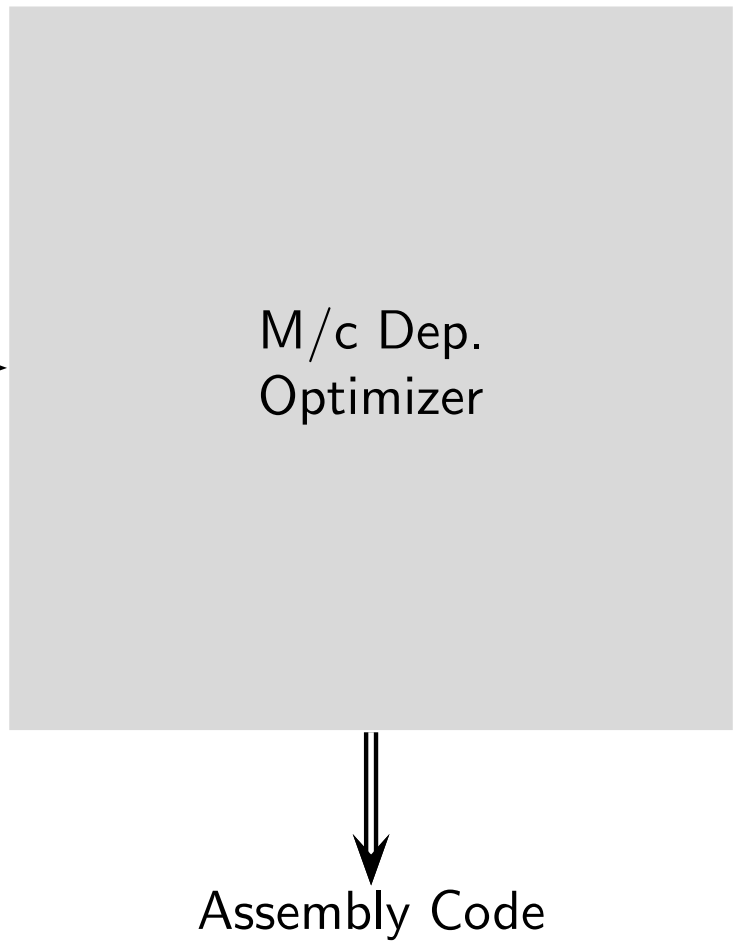
- Instruction Selection
- Local Reg Allocation
- Choice of Order of Evaluation

Typical Back Ends

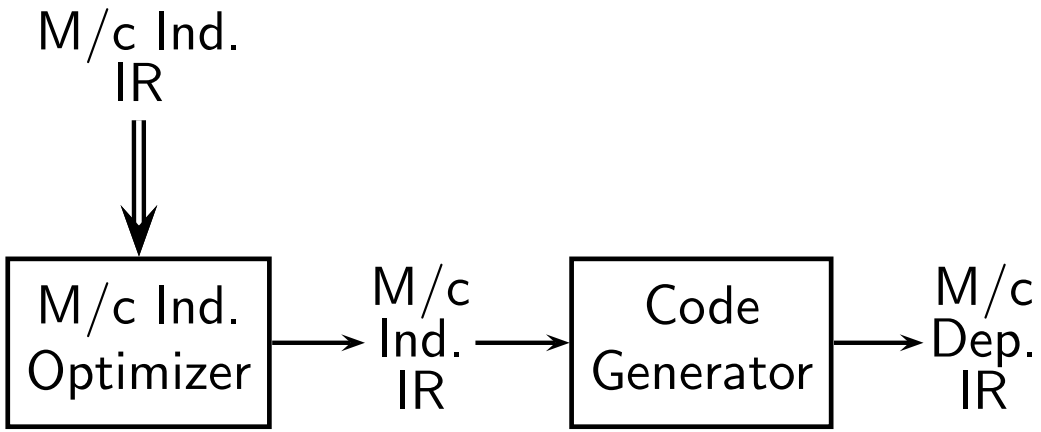


- Compile time evaluations
- Eliminating redundant computations

- Instruction Selection
- Local Reg Allocation
- Choice of Order of Evaluation

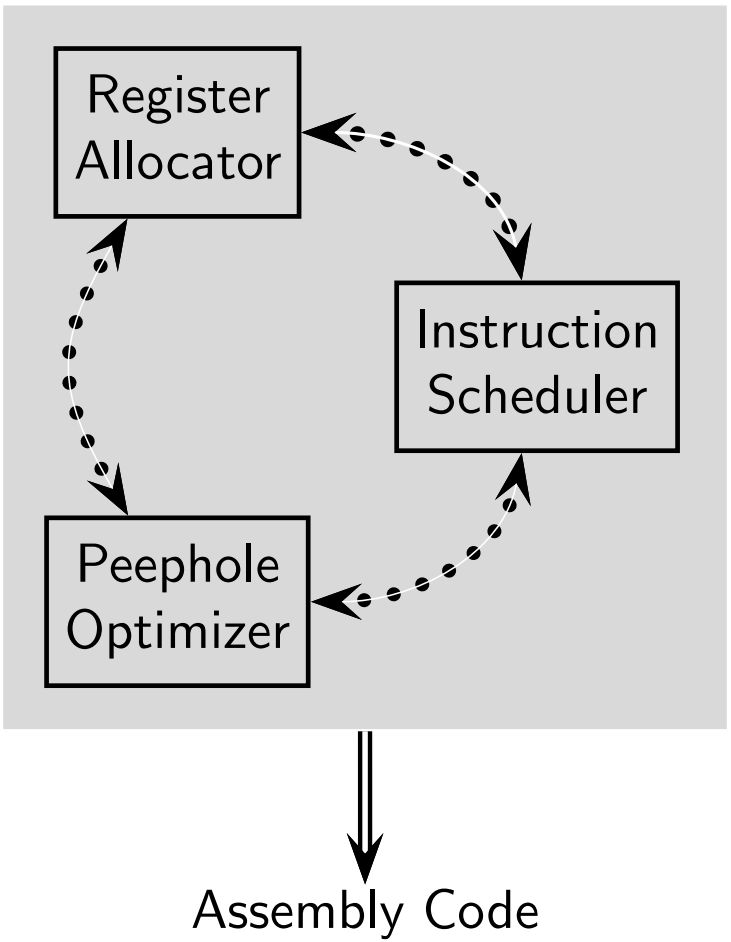


Typical Back Ends



- Compile time evaluations
- Eliminating redundant computations

- Instruction Selection
- Local Reg Allocation
- Choice of Order of Evaluation



Part 4

Major R&D Initiatives

Improving Retargetability and Instruction Selection

- The Problem:
- The Consequences:

Improving Retargetability and Instruction Selection

- The Problem: Instruction selection algorithms in GCC are very primitive (employ full tree matching instead of tree tiling).
- The Consequences:

Improving Retargetability and Instruction Selection

- The Problem: Instruction selection algorithms in GCC are very primitive (employ full tree matching instead of tree tiling).
- The Consequences:
  - A compiler developer needs to visualize and specify meaningful combinations of instructions for generating good quality code.

Improving Retargetability and Instruction Selection

- The Problem: Instruction selection algorithms in GCC are very primitive (employ full tree matching instead of tree tiling).
- The Consequences:
  - A compiler developer needs to visualize and specify meaningful combinations of instructions for generating good quality code.
  - The machine descriptions are difficult to construct, understand, maintain, and enhance.

Improving Retargetability and Instruction Selection

- The Problem: Instruction selection algorithms in GCC are very primitive (employ full tree matching instead of tree tiling).
- The Consequences:
  - A compiler developer needs to visualize and specify meaningful combinations of instructions for generating good quality code.
  - The machine descriptions are difficult to construct, understand, maintain, and enhance.
  - GCC has become a hacker's paradise instead of a clean, production quality compiler generation framework.

Improving Retargetability and Instruction Selection

- Our Goals:

- Current Status:

Improving Retargetability and Instruction Selection

- Our Goals:
  - Discover the abstractions required in machine descriptions and develop a systematic methodology of constructing them.

- Current Status:

Improving Retargetability and Instruction Selection

- Our Goals:
  - Discover the abstractions required in machine descriptions and develop a systematic methodology of constructing them.
  - Use tree tiling based instruction selection algorithms to allow for cleaner and simpler machine descriptions.
- Current Status:

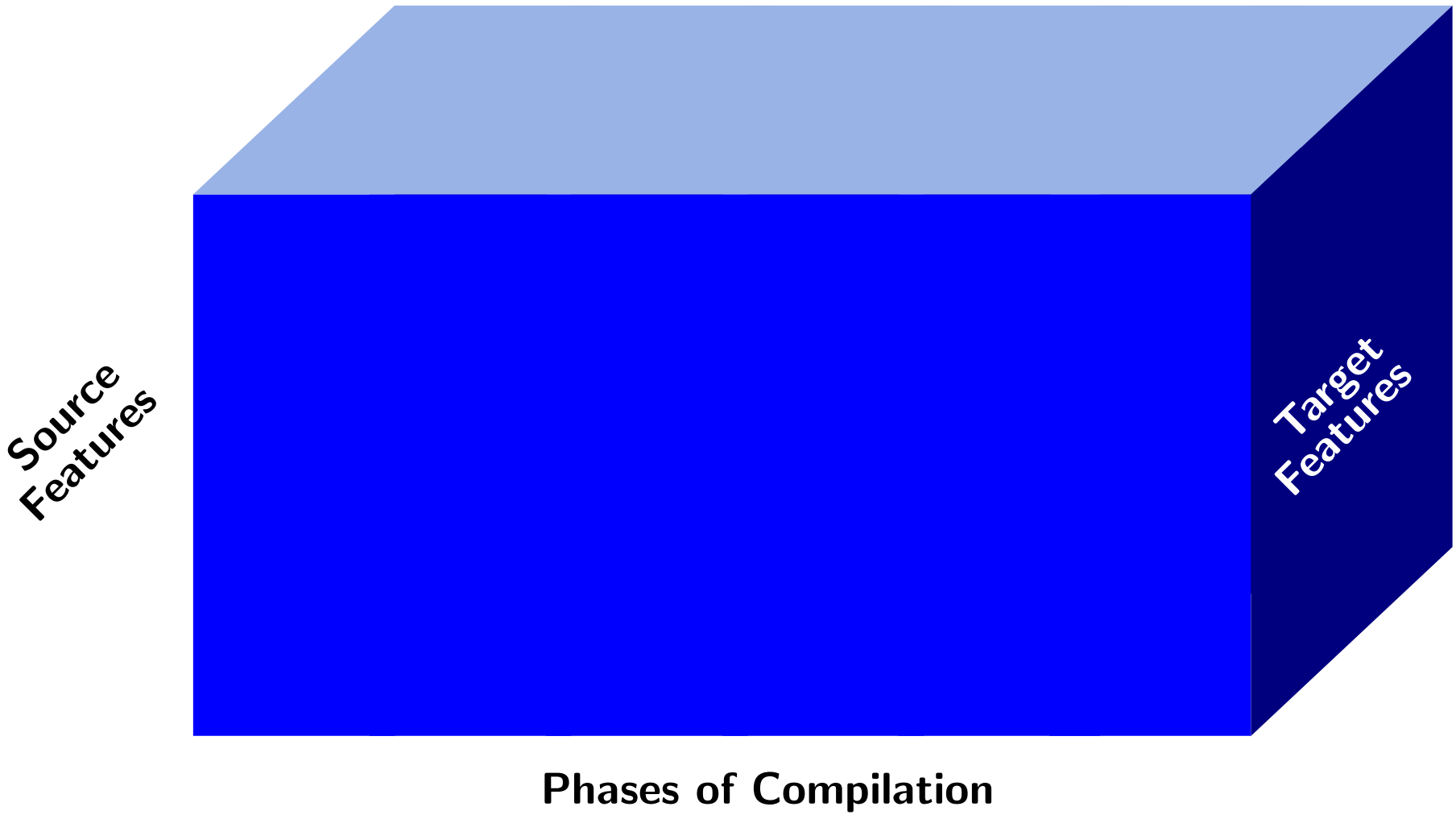
Improving Retargetability and Instruction Selection

- Our Goals:
  - Discover the abstractions required in machine descriptions and develop a systematic methodology of constructing them.
  - Use tree tiling based instruction selection algorithms to allow for cleaner and simpler machine descriptions.
- Current Status:
  - A methodology of incremental construction has been devised.

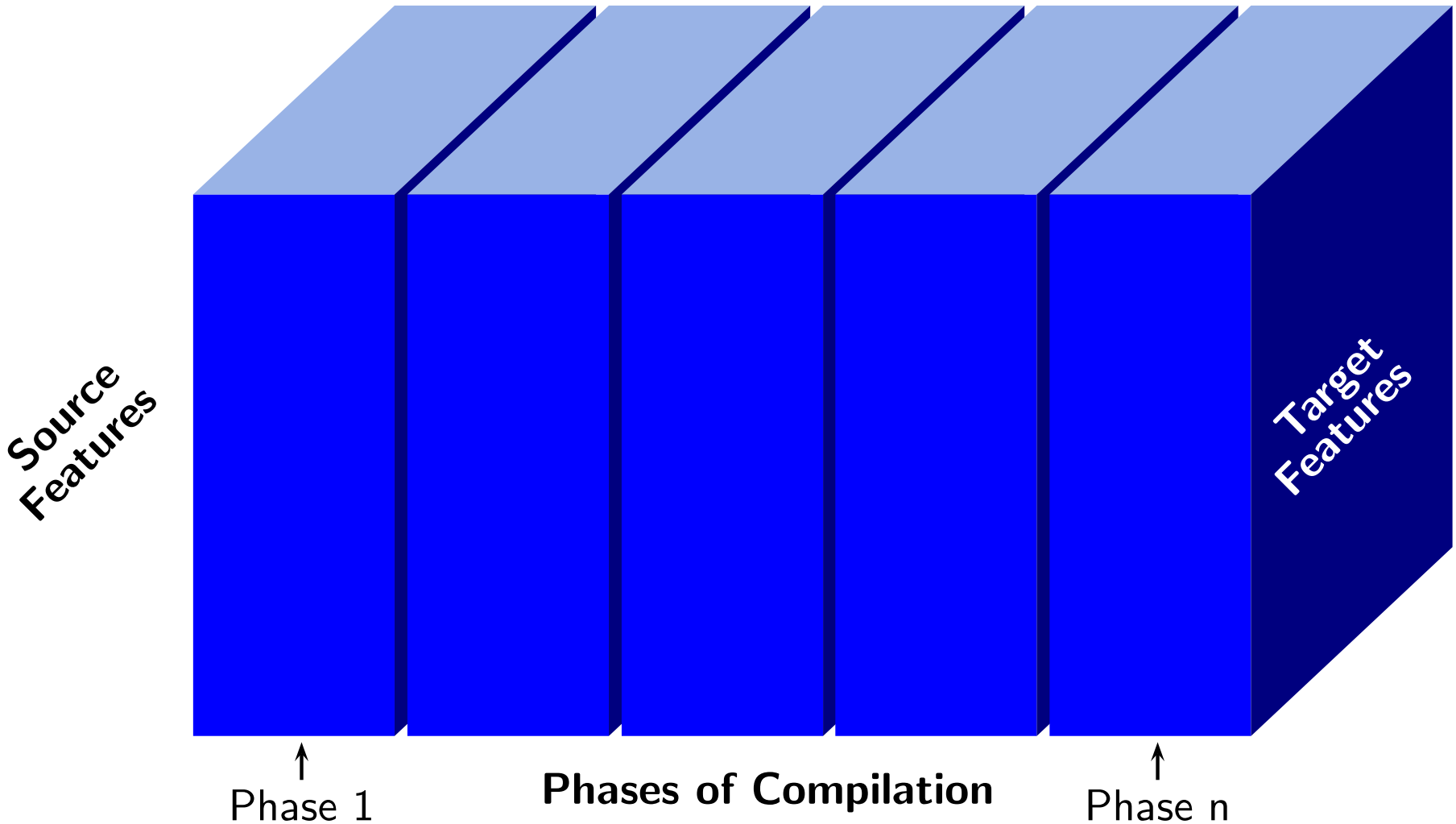
Improving Retargetability and Instruction Selection

- Our Goals:
  - Discover the abstractions required in machine descriptions and develop a systematic methodology of constructing them.
  - Use tree tiling based instruction selection algorithms to allow for cleaner and simpler machine descriptions.
- Current Status:
  - A methodology of incremental construction has been devised.
  - Preliminary investigations in using `iburg` seem very promising. (Only 200 rules required for `i386` instead of over a 1000!)

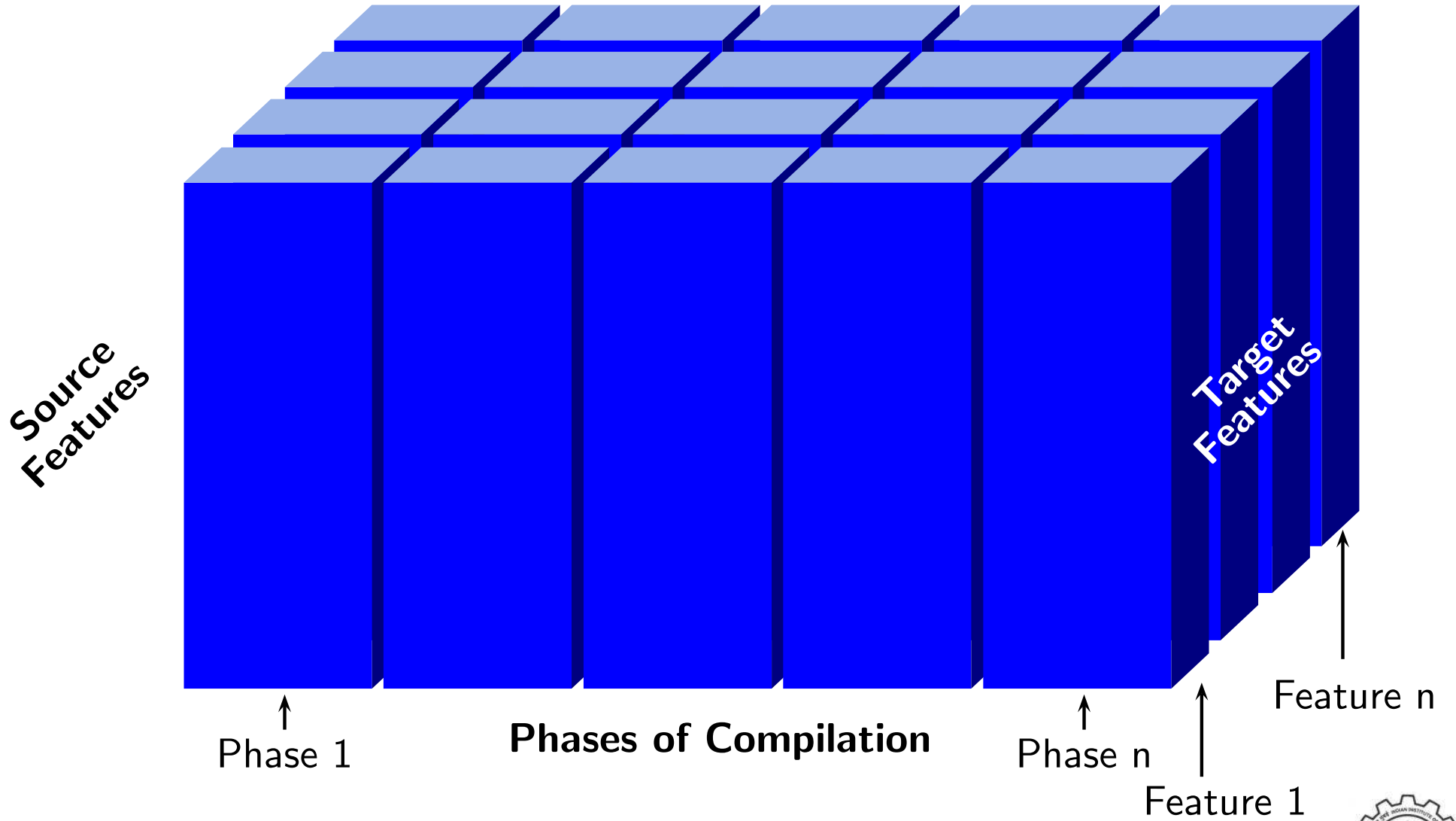
In Search of Modularity in Retargetable Compilation



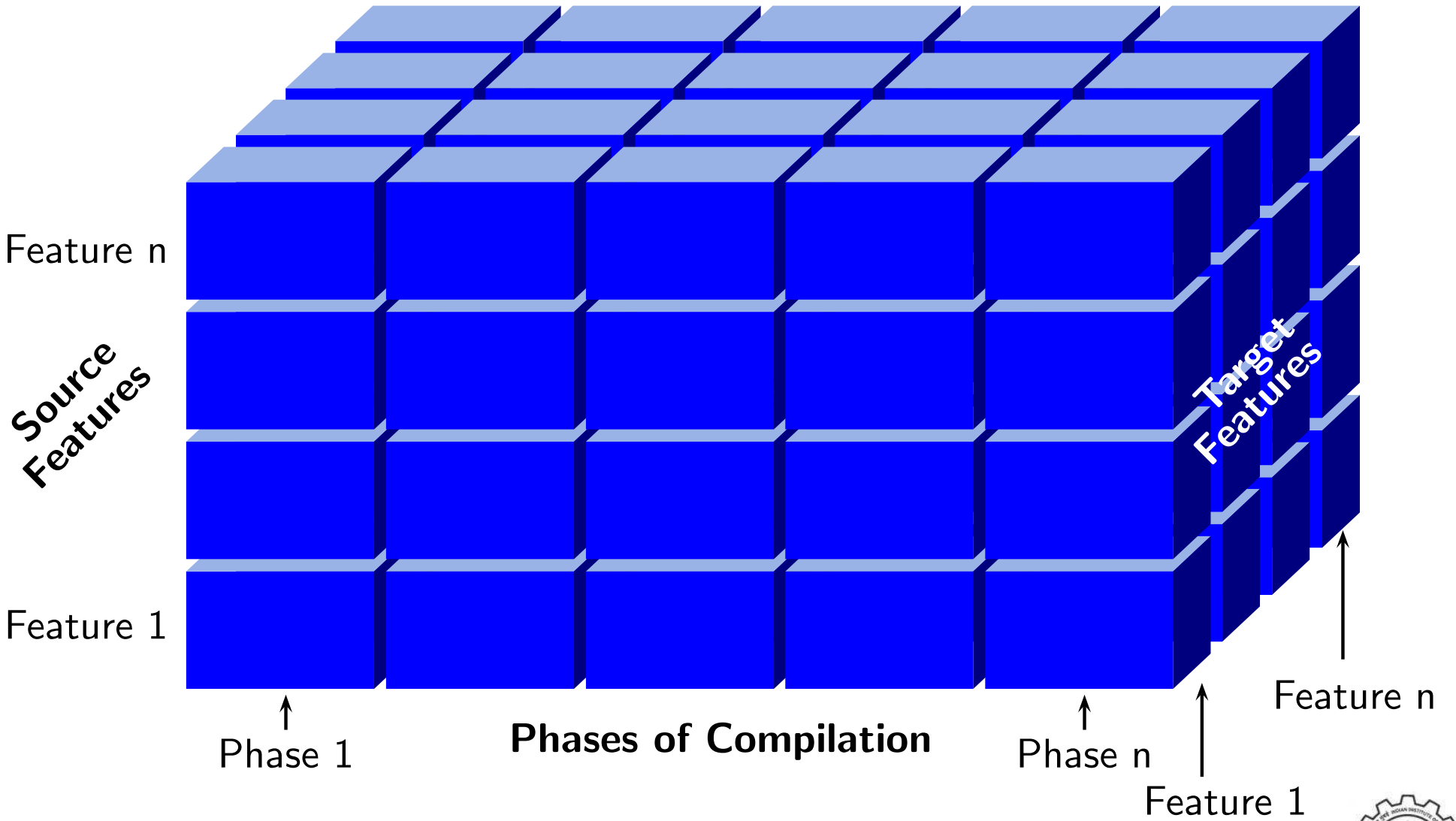
In Search of Modularity in Retargetable Compilation



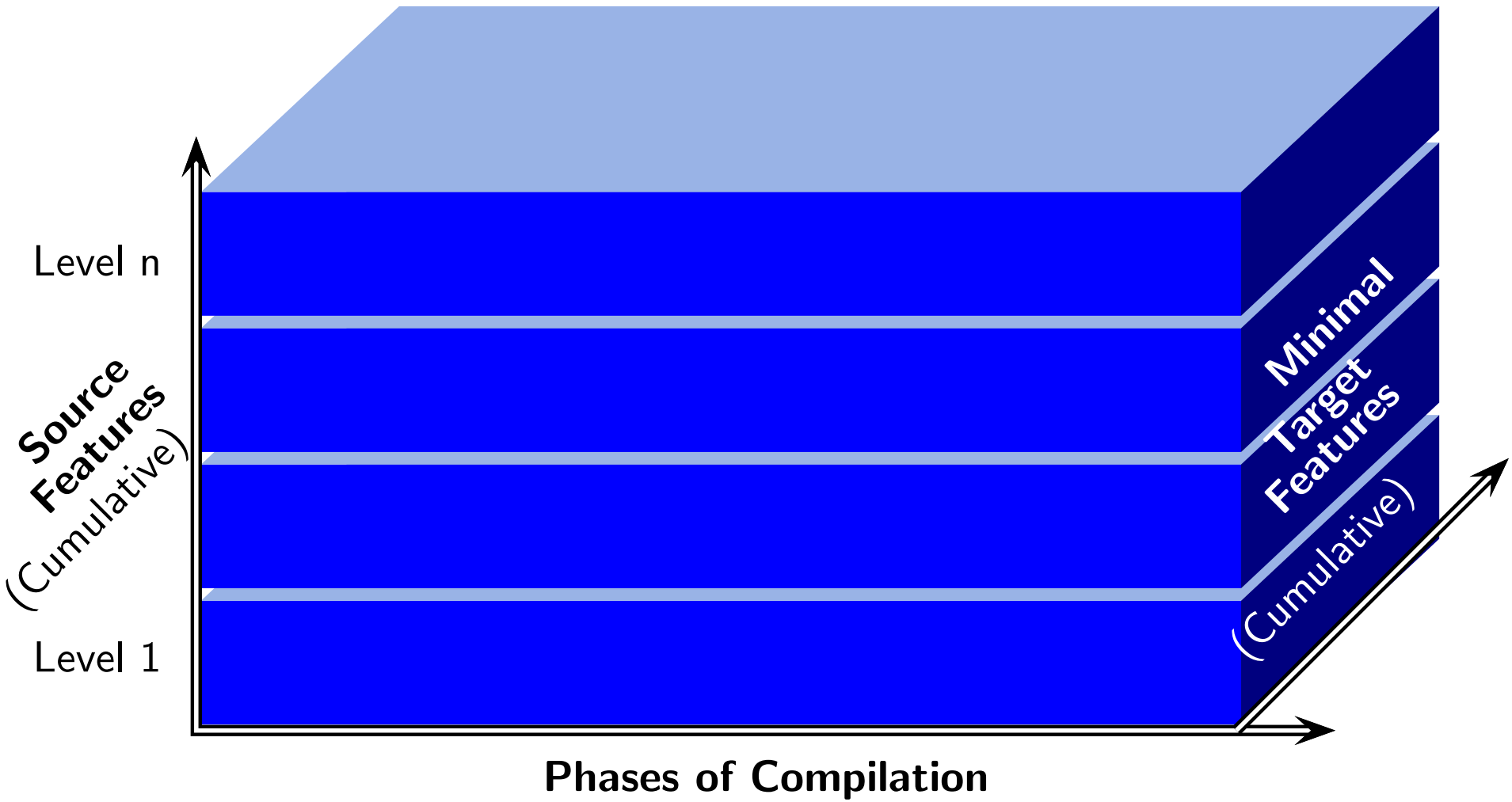
In Search of Modularity in Retargetable Compilation



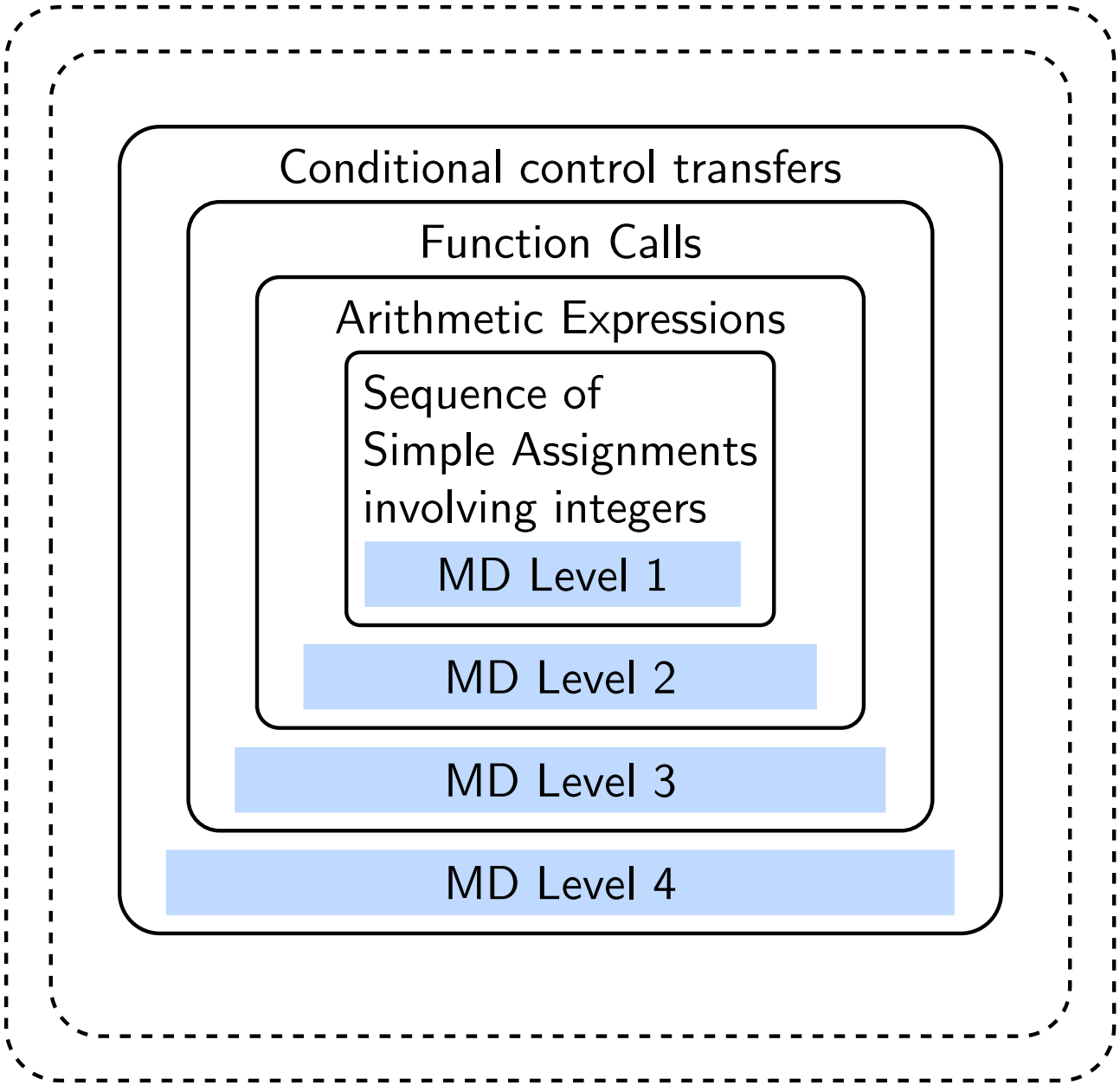
In Search of Modularity in Retargetable Compilation



In Search of Modularity in Retargetable Compilation



Systematic Development of Machine Descriptions



Improving Machine Independent Optimizations in GCC

- The Problems:
- Our Goals:
- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR).
- Our Goals:
- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR).
  - Whole program analysis does not exist.
- Our Goals:
- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR).
  - Whole program analysis does not exist.
- Our Goals:
  - Implement our algorithms of interprocedural analysis.
- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR).
  - Whole program analysis does not exist.
- Our Goals:
  - Implement our algorithms of interprocedural analysis.
  - Facilitate generation of optimizers from specifications.
    - Clean specifications
    - Systematic local, global, and interprocedural analysis
    - Simple, efficient, generic, and precise algorithms
    - Incremental analyses for aggressive optimizations
- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR).
  - Whole program analysis does not exist.
- Our Goals:
  - Implement our algorithms of interprocedural analysis.
  - Facilitate generation of optimizers from specifications.
    - Clean specifications
    - Systematic local, global, and interprocedural analysis
    - Simple, efficient, generic, and precise algorithms
    - Incremental analyses for aggressive optimizations
- Current Status:
  - The required algorithms and their formal theory is in place.

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR).
  - Whole program analysis does not exist.
- Our Goals:
  - Implement our algorithms of interprocedural analysis.
  - Facilitate generation of optimizers from specifications.
    - Clean specifications
    - Systematic local, global, and interprocedural analysis
    - Simple, efficient, generic, and precise algorithms
    - Incremental analyses for aggressive optimizations
- Current Status:
  - The required algorithms and their formal theory is in place.
  - The results of proof of concept implementations are very encouraging.

Translation Validation of GCC

- Problem:
- Our Objectives:
- Our approach:
- Current Status:
- Future work:

Translation Validation of GCC

- Problem:
  - Establishing correctness of compilers is important.
  - Verifying a real compiler is very difficult.
- Our Objectives:
- Our approach:
- Current Status:
- Future work:

Translation Validation of GCC

- Problem:
  - Establishing correctness of compilers is important.
  - Verifying a real compiler is very difficult.
- Our Objectives: To build a system to verify the correctness of the translation of given program.
- Our approach:
- Current Status:
- Future work:

Translation Validation of GCC

- Problem:
  - Establishing correctness of compilers is important.
  - Verifying a real compiler is very difficult.
- Our Objectives: To build a system to verify the correctness of the translation of given program.
- Our approach:
  - Define suitable observation points and observables
  - Establish the conditions under which the observables correspond at the end of the program.
  - Derive the conditions under which the observables correspond at the start of the program.
- Current Status:
- Future work:

Translation Validation of GCC

- Problem:
  - Establishing correctness of compilers is important.
  - Verifying a real compiler is very difficult.
- Our Objectives: To build a system to verify the correctness of the translation of given program.
- Our approach:
  - Define suitable observation points and observables
  - Establish the conditions under which the observables correspond at the end of the program.
  - Derive the conditions under which the observables correspond at the start of the program.
- Current Status: Formal theory and prototype implementation to show the correctness of translation of a few programs exist.
- Future work:

Translation Validation of GCC

- Problem:
  - Establishing correctness of compilers is important.
  - Verifying a real compiler is very difficult.
- Our Objectives: To build a system to verify the correctness of the translation of given program.
- Our approach:
  - Define suitable observation points and observables
  - Establish the conditions under which the observables correspond at the end of the program.
  - Derive the conditions under which the observables correspond at the start of the program.
- Current Status: Formal theory and prototype implementation to show the correctness of translation of a few programs exist.
- Future work:
  - Cleaning up the theory to systematize the termination criteria.
  - Extending the approach to include more optimizations.

Typed Intermediate Representations in GCC

- The Problems:
- Our Goals:
- Current Status:

Typed Intermediate Representations in GCC

- The Problems:
  - The intermediate representations are typeless.
- Our Goals:
- Current Status:

Typed Intermediate Representations in GCC

- The Problems:
  - The intermediate representations are typeless.
  - Enforcing semantic correctness and consistency of IRs is left to te developer.
- Our Goals:
- Current Status:

Typed Intermediate Representations in GCC

- The Problems:
  - The intermediate representations are typeless.
  - Enforcing semantic correctness and consistency of IRs is left to te developer.
- Our Goals:
  - Change the Gimple IR to bring in type sensitivity.
- Current Status:

Typed Intermediate Representations in GCC

- The Problems:
  - The intermediate representations are typeless.
  - Enforcing semantic correctness and consistency of IRs is left to te developer.
- Our Goals:
  - Change the Gimple IR to bring in type sensitivity.
  - Define appropriate type system.
  - The access functions for Gimple IR should enforce type consistency.
- Current Status:

Typed Intermediate Representations in GCC

- The Problems:
  - The intermediate representations are typeless.
  - Enforcing semantic correctness and consistency of IRs is left to te developer.
- Our Goals:
  - Change the Gimple IR to bring in type sensitivity.
  - Define appropriate type system.
  - The access functions for Gimple IR should enforce type consistency.
- Current Status:
  - Problem definition has to be made more precise.

Linear Types in GCC

- The Problems:
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
  - Use linear types to prohibit aliasing.
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
  - Use linear types to prohibit aliasing.
  - Allow reasonable limited relaxations of linearity constraints.
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
  - Use linear types to prohibit aliasing.
  - Allow reasonable limited relaxations of linearity constraints.
  - Define appropriate type system and enforce it.
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
  - Use linear types to prohibit aliasing.
  - Allow reasonable limited relaxations of linearity constraints.
  - Define appropriate type system and enforce it.
- Current Status:
  - Linearity aspects in C have been studied in details.

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
  - Use linear types to prohibit aliasing.
  - Allow reasonable limited relaxations of linearity constraints.
  - Define appropriate type system and enforce it.
- Current Status:
  - Linearity aspects in C have been studied in details.
  - Variants of linearity have been identified.

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
  - Use linear types to prohibit aliasing.
  - Allow reasonable limited relaxations of linearity constraints.
  - Define appropriate type system and enforce it.
- Current Status:
  - Linearity aspects in C have been studied in details.
  - Variants of linearity have been identified.
  - An initial draft of the type system is in place.

Part 5

Conclusions

Conclusions

- Our group on GCC at IIT Bombay
  - Synergy from group activities
  - Long term commitment to challenging research problems
  - A desire to explore real issues in real compilers
    A dream to improve GCC

Conclusions

- Our group on GCC at IIT Bombay
  - Synergy from group activities
  - Long term commitment to challenging research problems
  - A desire to explore real issues in real compilers
    A dream to improve GCC

Conclusions

- Our group on GCC at IIT Bombay
  - Synergy from group activities
  - Long term commitment to challenging research problems
  - A desire to explore real issues in real compilers
    A dream to improve GCC
- *Would you like to be a part of this dream?*

Last but not the least ...

Thank You!