Improving the Gnu Compiler Collection

Uday Khedker

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

Jan 2010

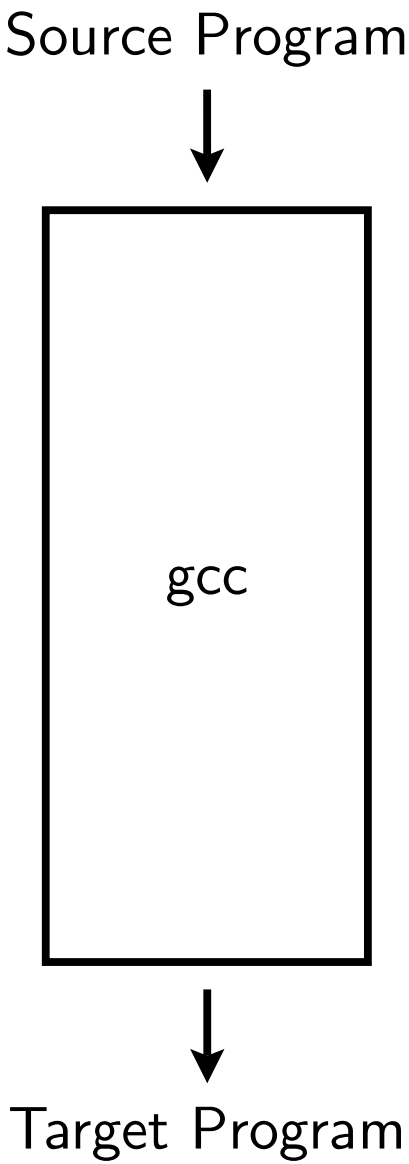
Outline

- GCC: The Great Compiler Challenge
- Understanding and Improving the Gnu Compiler Collection
  - Improving machine independent optimizations
  - Understanding machine descriptions
  - Improving machine descriptions and instruction selection
- Activities of GCC Resource Center
- Conclusions

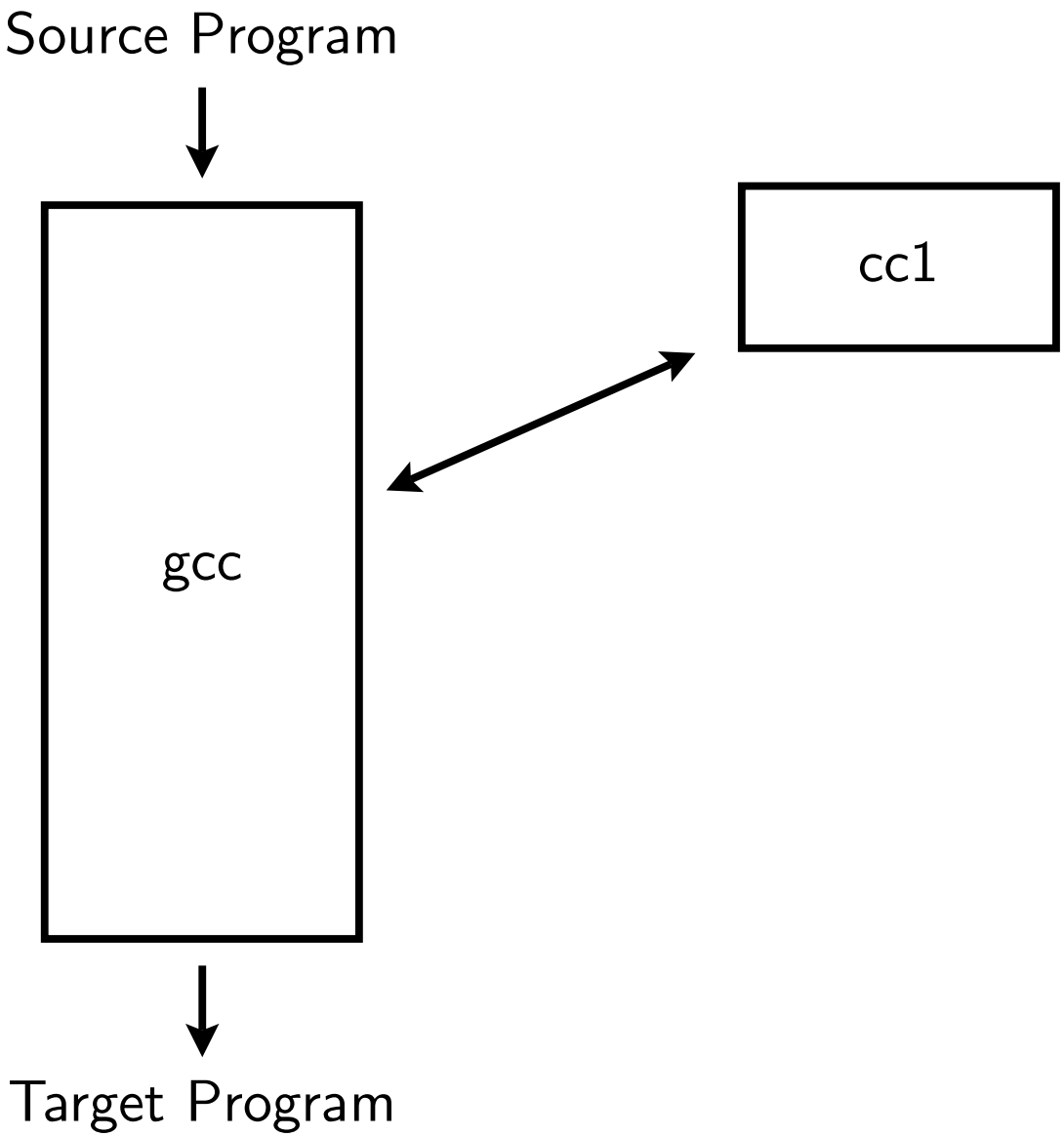
Part 1

GCC $\equiv$ *The Great Compiler Challenge*

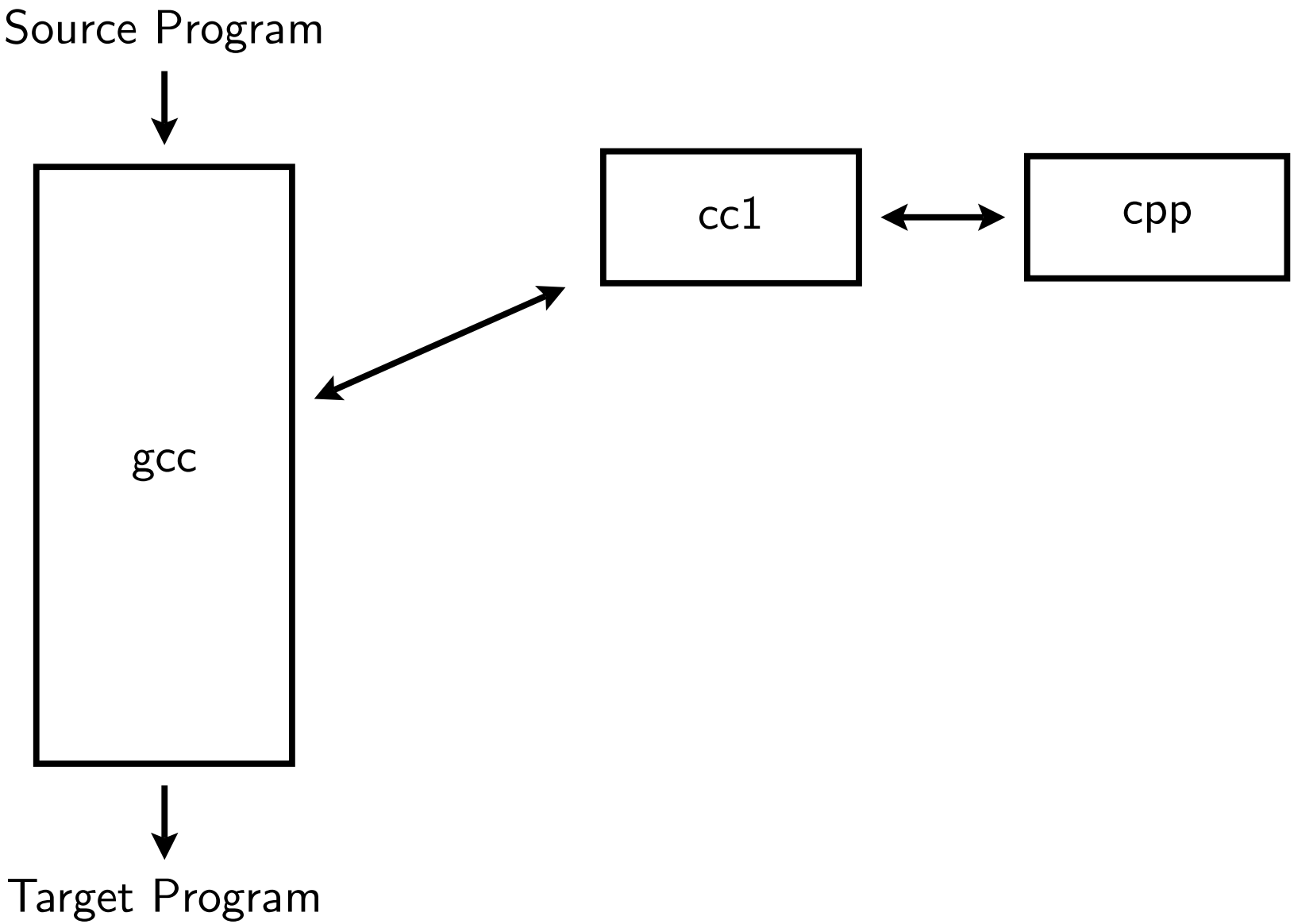
The Gnu Tool Chain



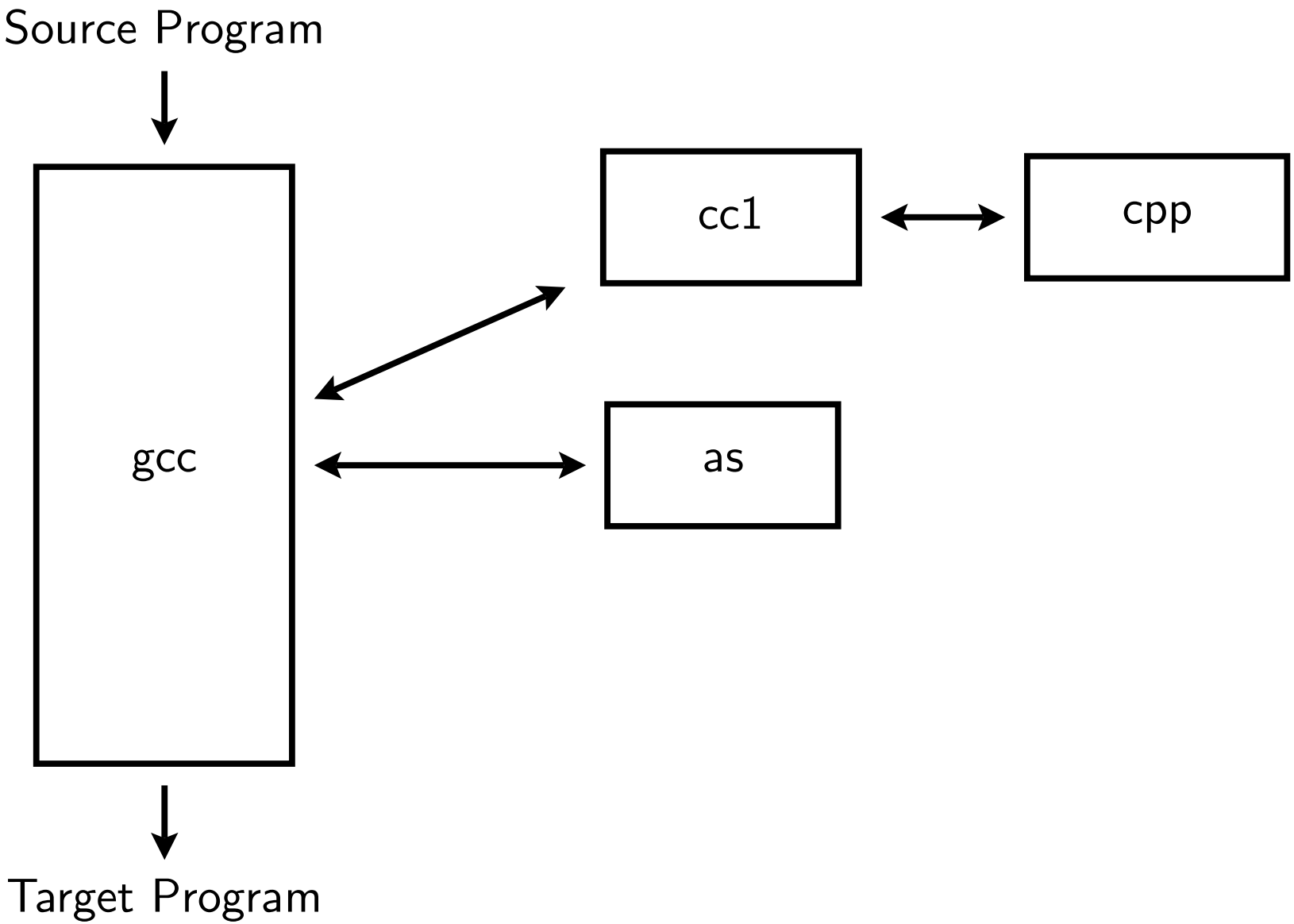
The Gnu Tool Chain



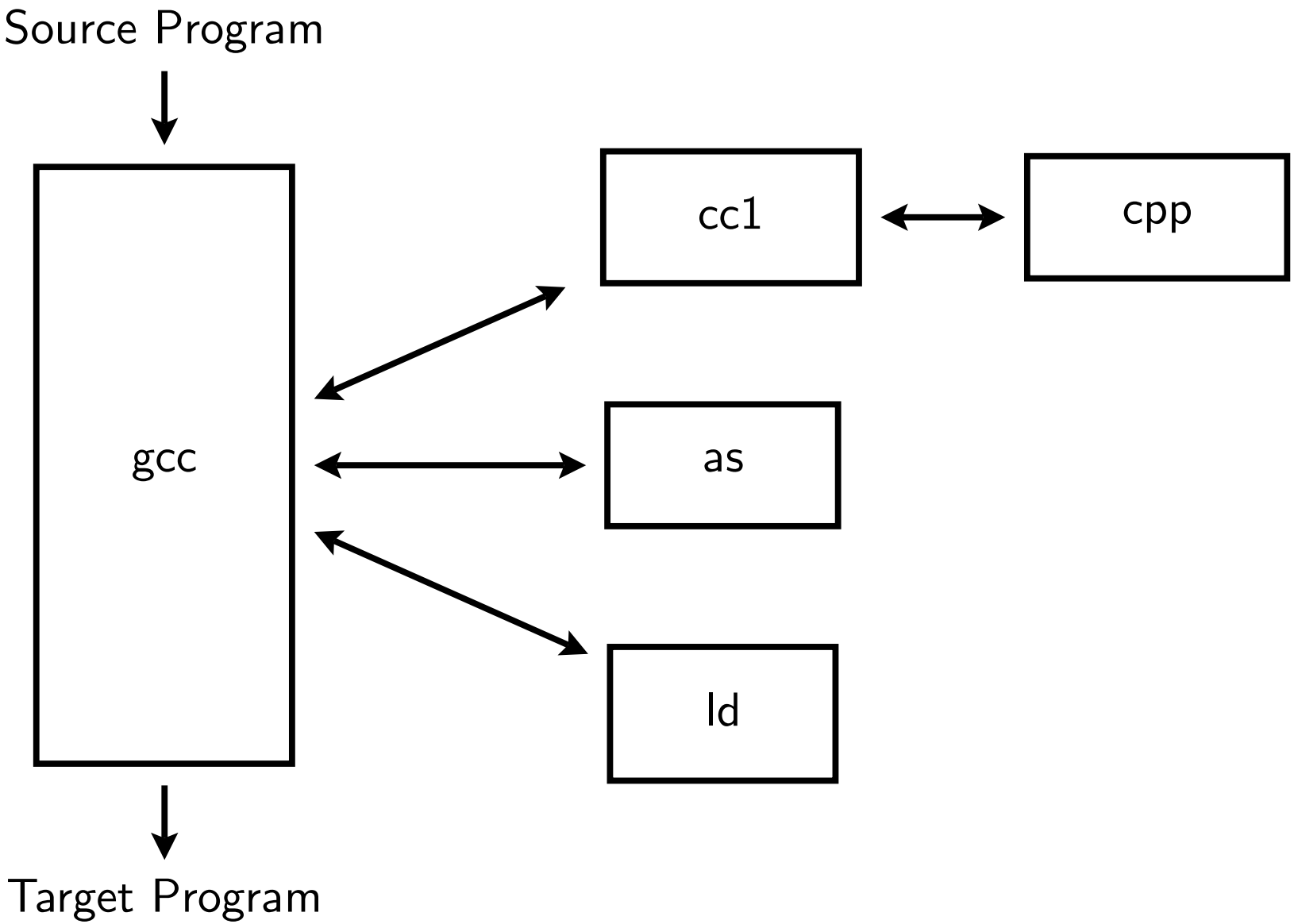
The Gnu Tool Chain



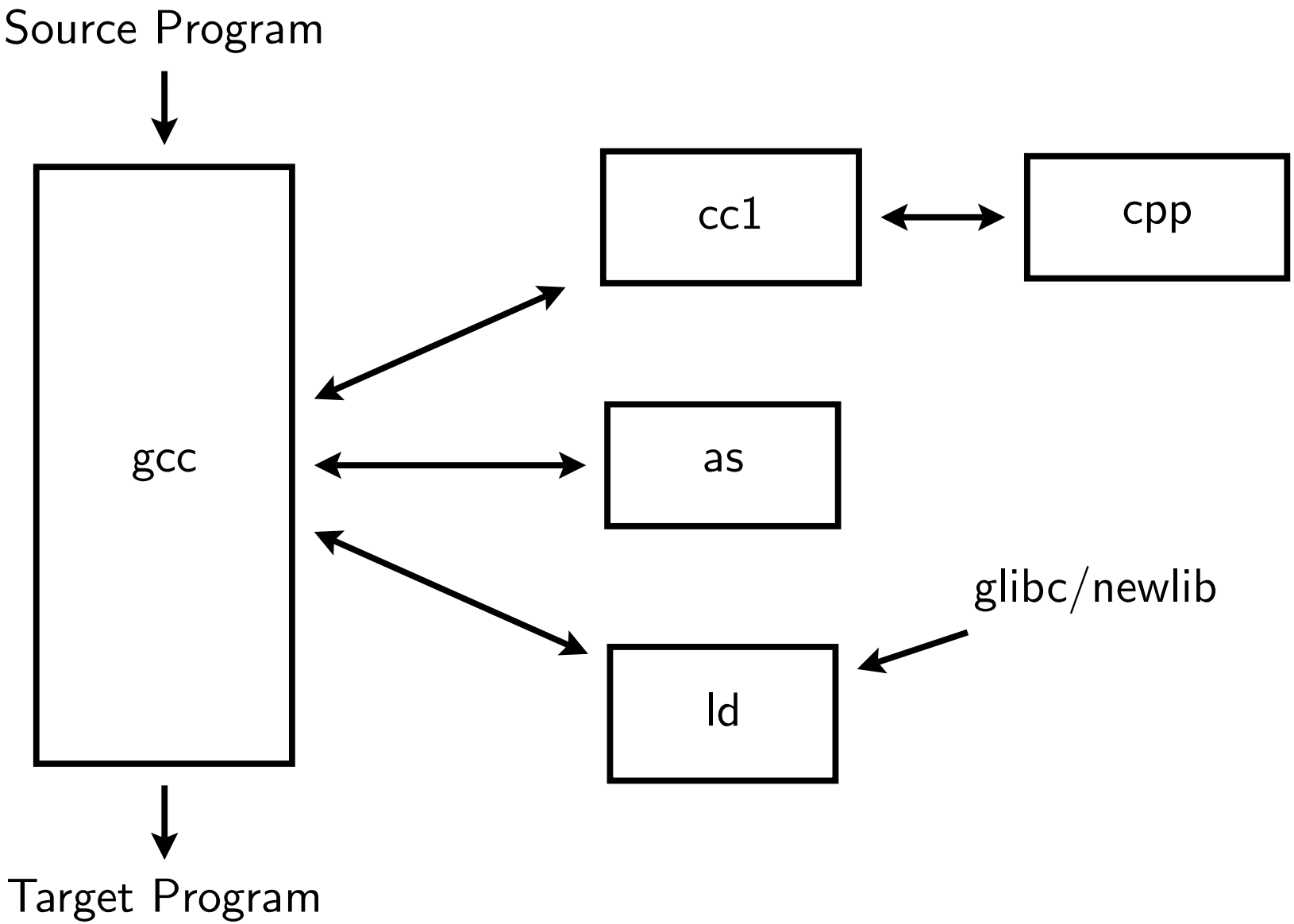
The Gnu Tool Chain



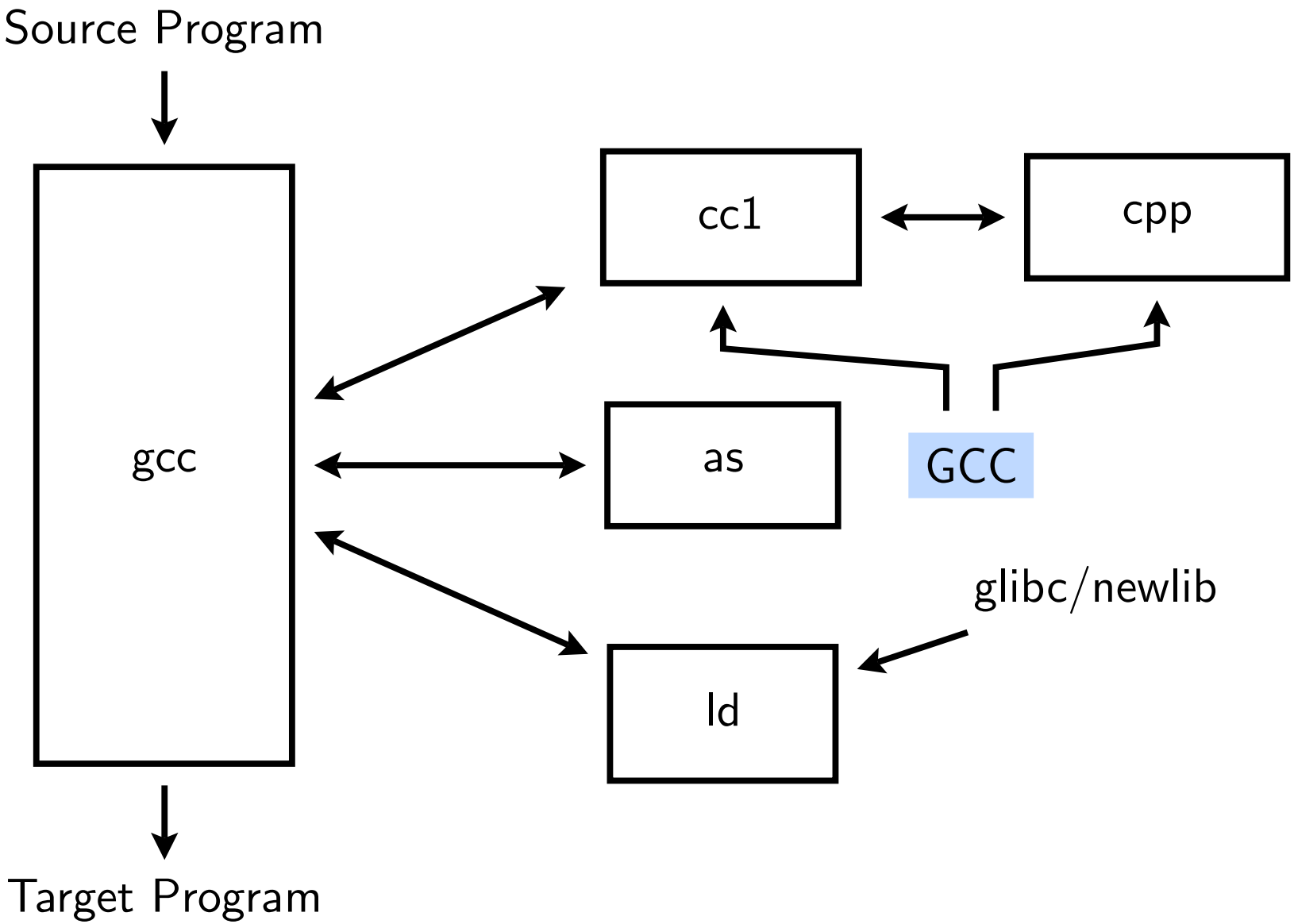
The Gnu Tool Chain



The Gnu Tool Chain



The Gnu Tool Chain



Why is Understanding GCC Difficult?

Some of the obvious reasons:

- Comprehensiveness

  GCC is a production quality framework in terms of completeness and practical usefulness

- Open development model

  Could lead to heterogeneity. Design flaws may be difficult to correct

- Rapid versioning

  GCC maintenance is a race against time. Disruptive corrections are difficult

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86),
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa,
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V,

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP,

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP,

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809,

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze,

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430,

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios,

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10,

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10, TIGCC (m68k variant),

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10, TIGCC (m68k variant), Z8000,

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10, TIGCC (m68k variant), Z8000, PIC24/dsPIC,

Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10, TIGCC (m68k variant), Z8000, PIC24/dsPIC, NEC SX architecture

Comprehensiveness of GCC 4.3.1: Size

Source Lines	Number of lines in the main source	2,029,115
	Number of lines in libraries	1,546,826
Directories	Number of subdirectories	3527
Files	Total number of files	57825
	C source files	19834
	Header files	9643
	C++ files	3638
	Java files	6289
	Makefiles and Makefile templates	163
	Configuration scripts	52
	Machine description files	186

(Line counts estimated by the program `sloccount` by David A. Wheeler)

Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- Cathedral: Total Centralized Control
  Design, implement, test, release
- Bazaar: Total Decentralization
  Release early, release often, make users partners in software development

  "Given enough eyeballs, all bugs are shallow"
  Code errors, logical errors, and architectural errors

Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- Cathedral: Total Centralized Control
  Design, implement, test, release
- Bazaar: Total Decentralization
  Release early, release often, make users partners in software development

  "Given enough eyeballs, all bugs are shallow"
  Code errors, logical errors, and architectural errors

A combination of the two seems more sensible

The Current Development Model of GCC

GCC follows a combination of the Cathedral and the Bazaar approaches

- GCC Steering Committee: Free Software Foundation has given charge
  - Major policy decisions
  - Handling Administrative and Political issues
- Release Managers:
  - Coordination of releases
- Maintainers:
  - Usually area/branch/module specific
  - Responsible for design and implementation
  - Take help of reviewers to evaluate submitted changes

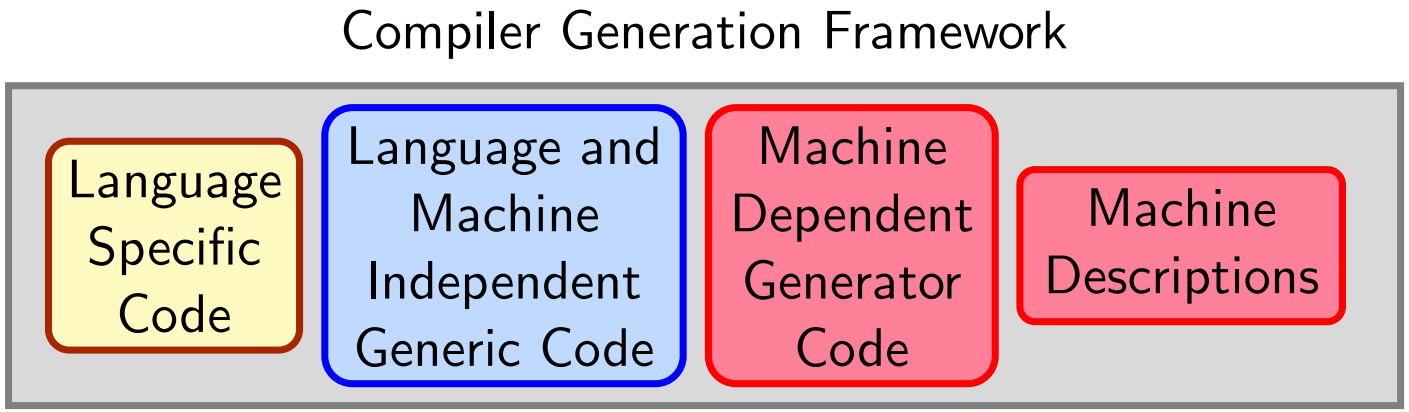
Why is Understanding GCC Difficult?

Deeper reason: GCC is not a *compiler* but a *compiler generation framework*

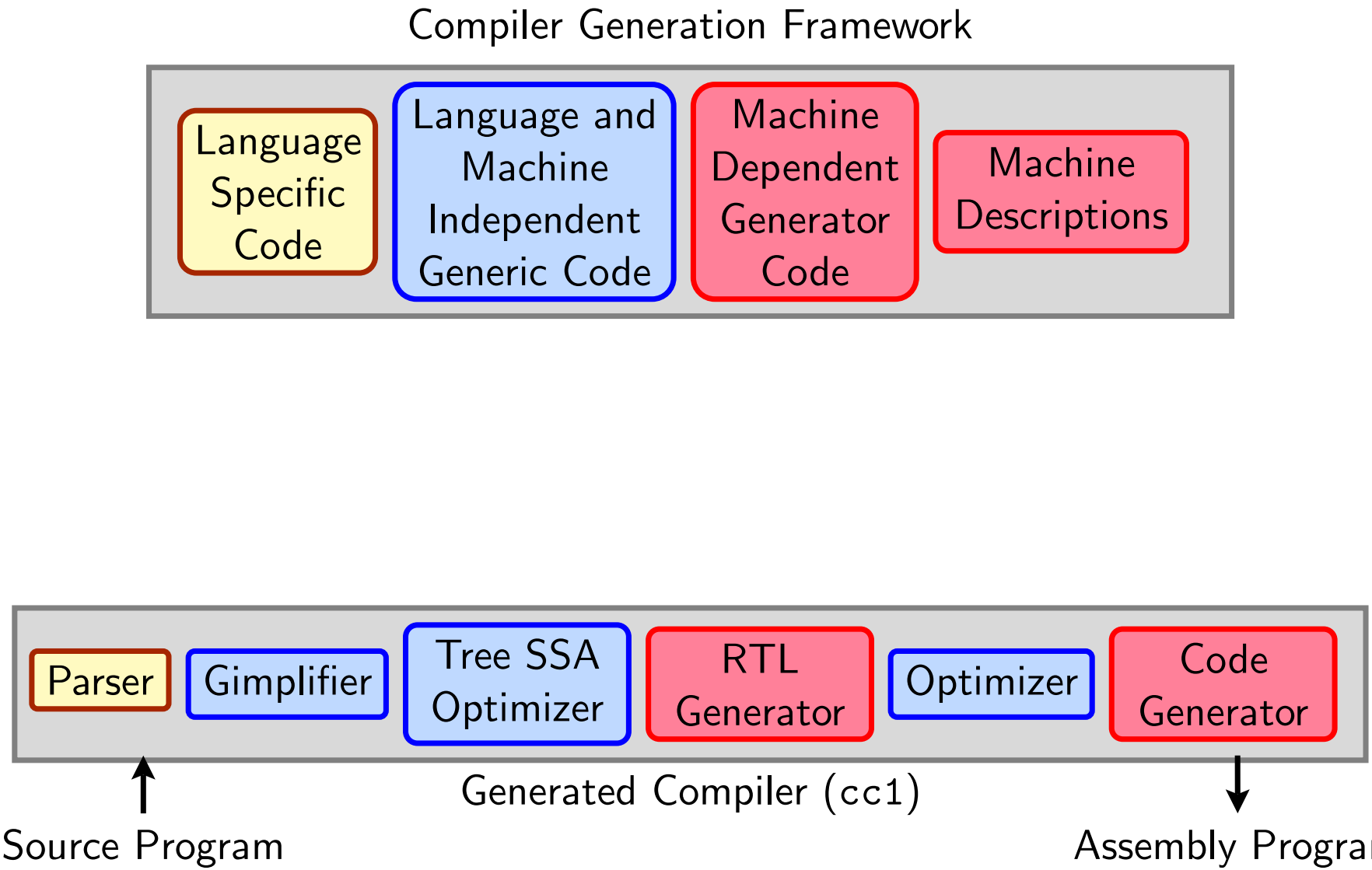
There are two distinct gaps that need to be bridged:

- Input-output of the generation framework: The target specification and the generated compiler
- Input-output of the generated compiler: A source program and the generated assembly program

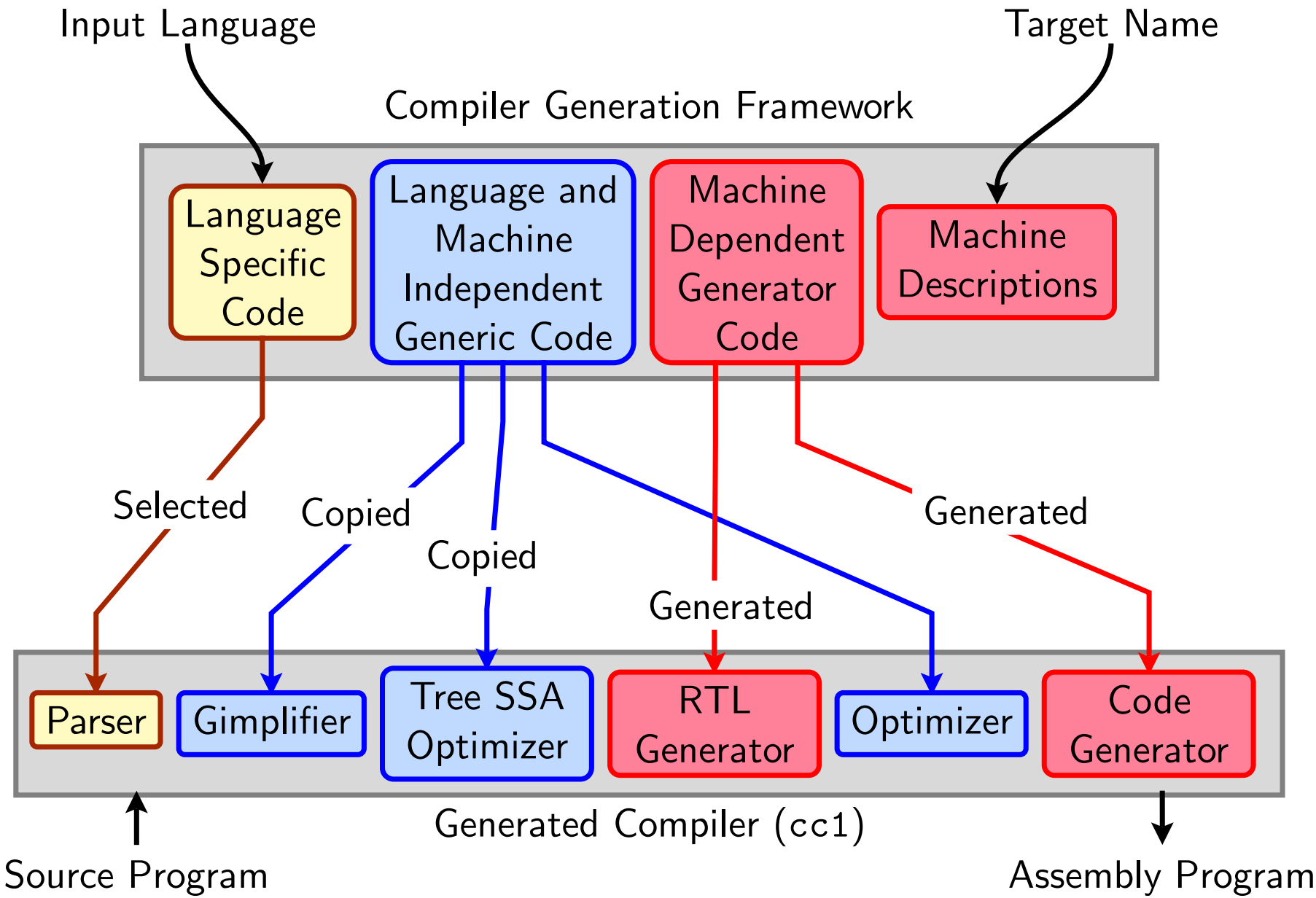
The Architecture of GCC



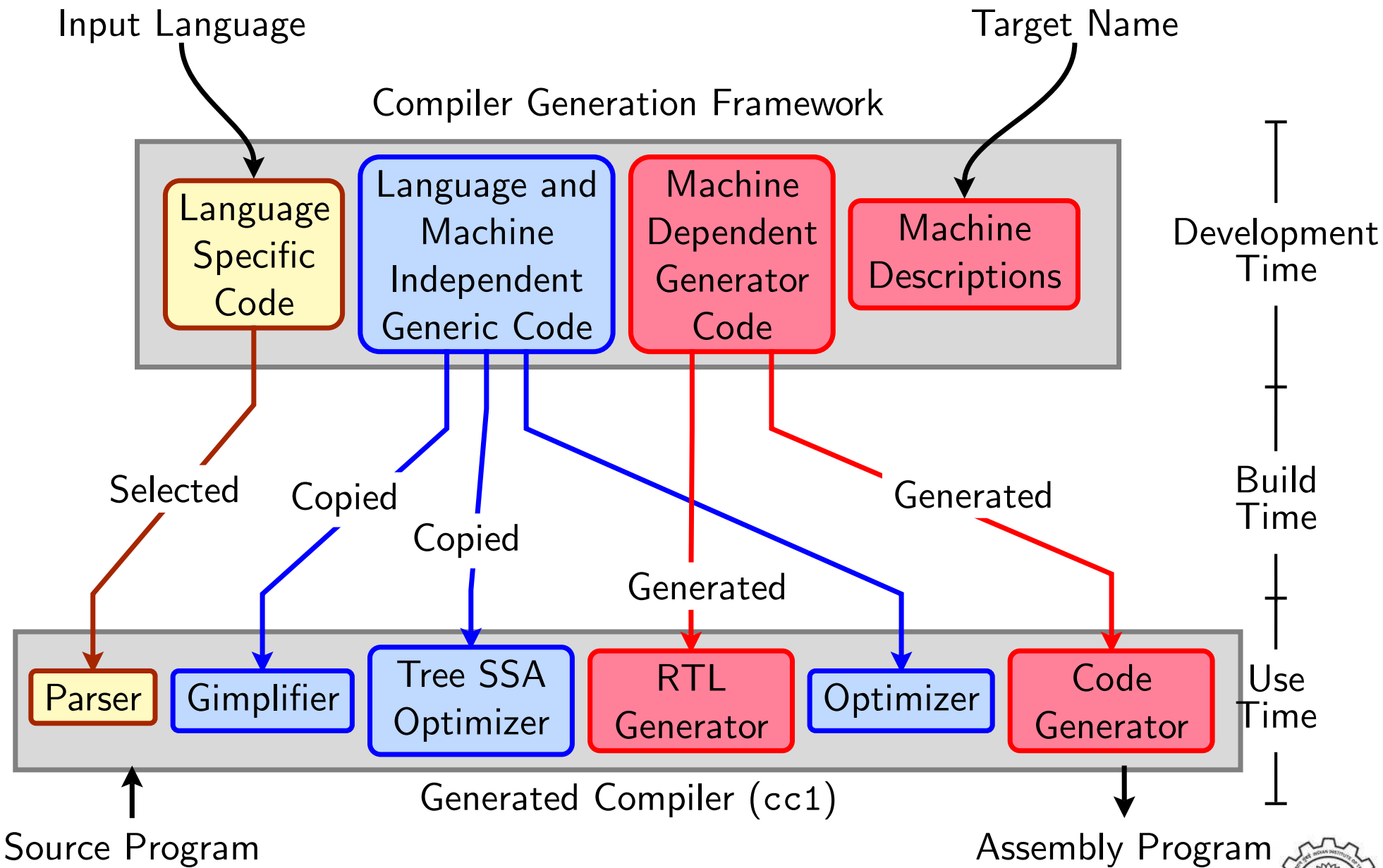
The Architecture of GCC



The Architecture of GCC



The Architecture of GCC



An Example of The Generation Related Gap

- Predicate function for invoking the loop distribution pass

```
static bool
gate_tree_loop_distribution (void)
{
  return flag_tree_loop_distribution != 0;
}
```


An Example of The Generation Related Gap

- Predicate function for invoking the loop distribution pass

```
static bool
gate_tree_loop_distribution (void)
{
  return flag_tree_loop_distribution != 0;
}
```

- There is no declaration of or assignment to variable `flag_tree_loop_distribution` in the entire source!
- It is described in `common.opt` as follows

```
ftree-loop-distribution
Common Report Var(flag_tree_loop_distribution) Optimization
Enable loop distribution on trees
```

- The required C statements are generated during the build

Another Example of The Generation Related Gap

Locating the `main` function in the directory `gcc-4.4.2/gcc` using cscope

Another Example of The Generation Related Gap

Locating the `main` function in the directory `gcc-4.4.2/gcc` using cscope

```
  File              Line
0 collect2.c         766 main (int argc, char **argv)
1 fix-header.c      1074 main (int argc, char **argv)
2 fp-test.c           85 main (void )
3 gcc.c             6216 main (int argc, char **argv)
4 gcov-dump.c         76 main (int argc ATTRIBUTE_UNUSED, char **argv
5 gcov-iov.c          29 main (int argc, char **argv)
6 gcov.c             355 main (int argc, char **argv)
7 gen-protos.c       130 main (int argc ATTRIBUTE_UNUSED, char **argv
8 genattr.c           89 main (int argc, char **argv)
9 genattrtab.c      4438 main (int argc, char **argv)
a genautomata.c     9321 main (int argc, char **argv)
b gencheckSum.c       65 main (int argc, char ** argv)
c gencodes.c          51 main (int argc, char **argv)
d genconditions.c    209 main (int argc, char **argv)
e genconfig.c        261 main (int argc, char **argv)
f genconstants.c      50 main (int argc, char **argv)
```

Another Example of The Generation Related Gap

Locating the `main` function in the directory `gcc-4.4.2/gcc` using cscope

```
g genemit.c          820 main (int argc, char **argv)
h genextract.c       394 main (int argc, char **argv)
i genflags.c         231 main (int argc, char **argv)
j gengenrtl.c        350 main (int argc, char **argv)
k gengtype.c        3584 main (int argc, char **argv)
l genmddeps.c         45 main (int argc, char **argv)
m genmodes.c        1376 main (int argc, char **argv)
n genopinit.c        472 main (int argc, char **argv)
o genoutput.c       1005 main (int argc, char **argv)
p genpeep.c          353 main (int argc, char **argv)
q genpreds.c        1399 main (int argc, char **argv)
r genrecog.c        2718 main (int argc, char **argv)
s main.c              33 main (int argc, char **argv)
t mips-tdump.c      1393 main (int argc, char **argv)
u mips-tfile.c       655 main (void )
v mips-tfile.c      4690 main (int argc, char **argv)
w protoize.c        4373 main (int argc, char **const argv)
```

The GCC Challenge: Poor Retargetability Mechanism

- Symptom of poor retargetability mechanism

  Large size of specifications

The GCC Challenge: Poor Retargetability Mechanism

- Symptom of poor retargetability mechanism

  Large size of specifications

- Size in terms of line counts

Files	i386	mips
*.md	35766	12930
*.c	28643	12572
*.h	15694	5105

Meeting the GCC Challenge

Goal of Understanding	Methodology	Needs Examining		
		Makefiles	Source	MD
Translation sequence of programs	Gray box probing	No	No	No
Build process	Customizing the configuration and building	Yes	No	No
Retargetability issues and machine descriptions	Incremental construction of machine descriptions	No	No	Yes
IR data structures and access mechanisms	Adding passes to massage IRs	No	Yes	Yes
Retargetability mechanism		Yes	Yes	Yes

Broad Research Goals of GCC Resource Center

- Using GCC as a means
  - Adding new optimizations to GCC
  - Adding flow and context sensitive analyses to GCC (In particular, pointer analysis)
  - Translation validation of GCC
  - Linear types in GCC
- Using GCC as an end in itself
  - Changing the retargetability mechanism of GCC
  - Cleaning up the machine descriptions of GCC
  - Systematic construction of machine descriptions
  - Facilitating optimizer generation in GCC

Part 2

Improving Machine Independent Optimizations in GCC

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)
  - Context and flow sensitive 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)
  - Context and flow sensitive whole program analysis does not exist
- Our Goals:
  - Implement scalable context and flow sensitive pointer 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)
  - Context and flow sensitive whole program analysis does not exist
- Our Goals:
  - Implement scalable context and flow sensitive pointer 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:
  - *gdfa*: Generic intraprocedural bit vector data flow analysis (patch released for GCC 4.3.0)
  - Algorithms and formal theory required further is in place

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives:
- Main Challenge:
- The State of Art:
- Our Breakthrough:
- The Consequences:
- Further Goal:

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- Main Challenge:
- The State of Art:
- Our Breakthrough:
- The Consequences:
- Further Goal:

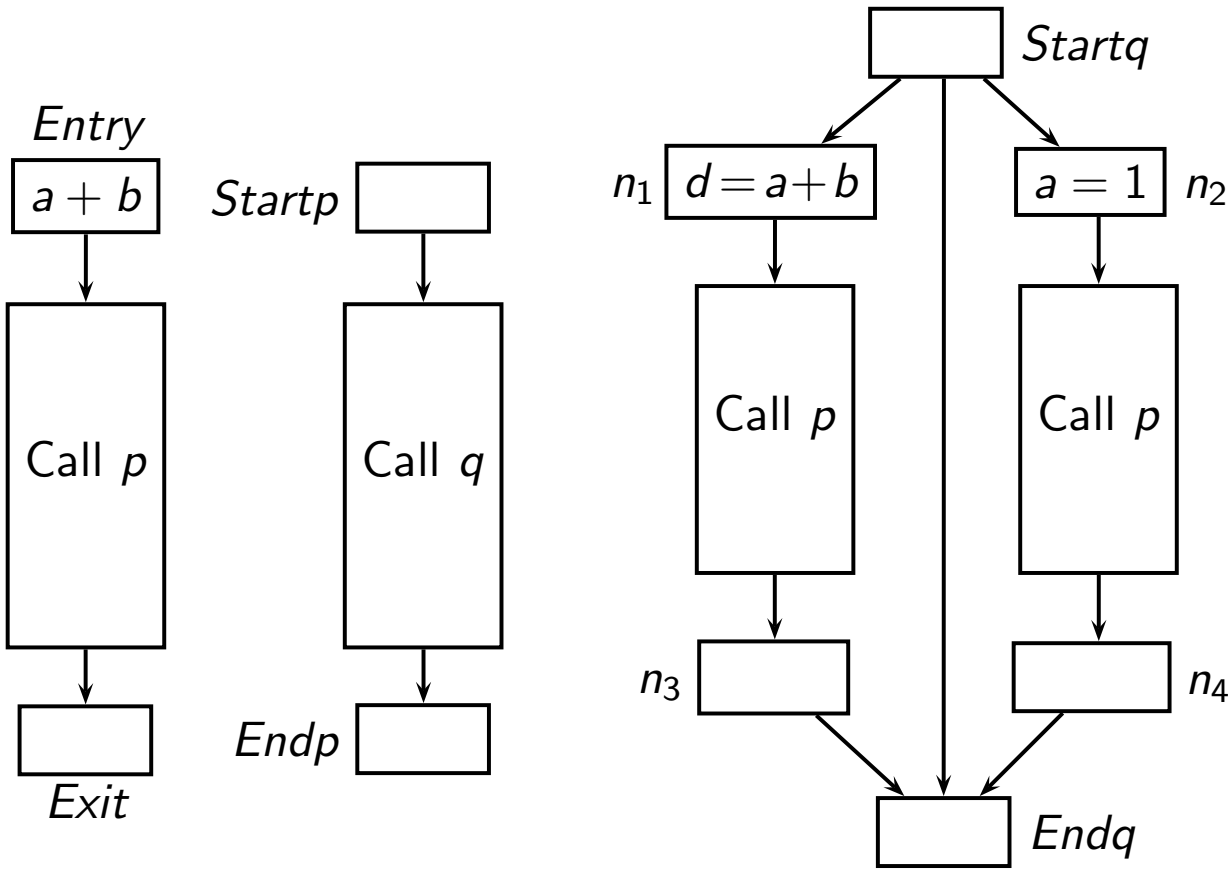
Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point
- The State of Art:
- Our Breakthrough:
- The Consequences:
- Further Goal:

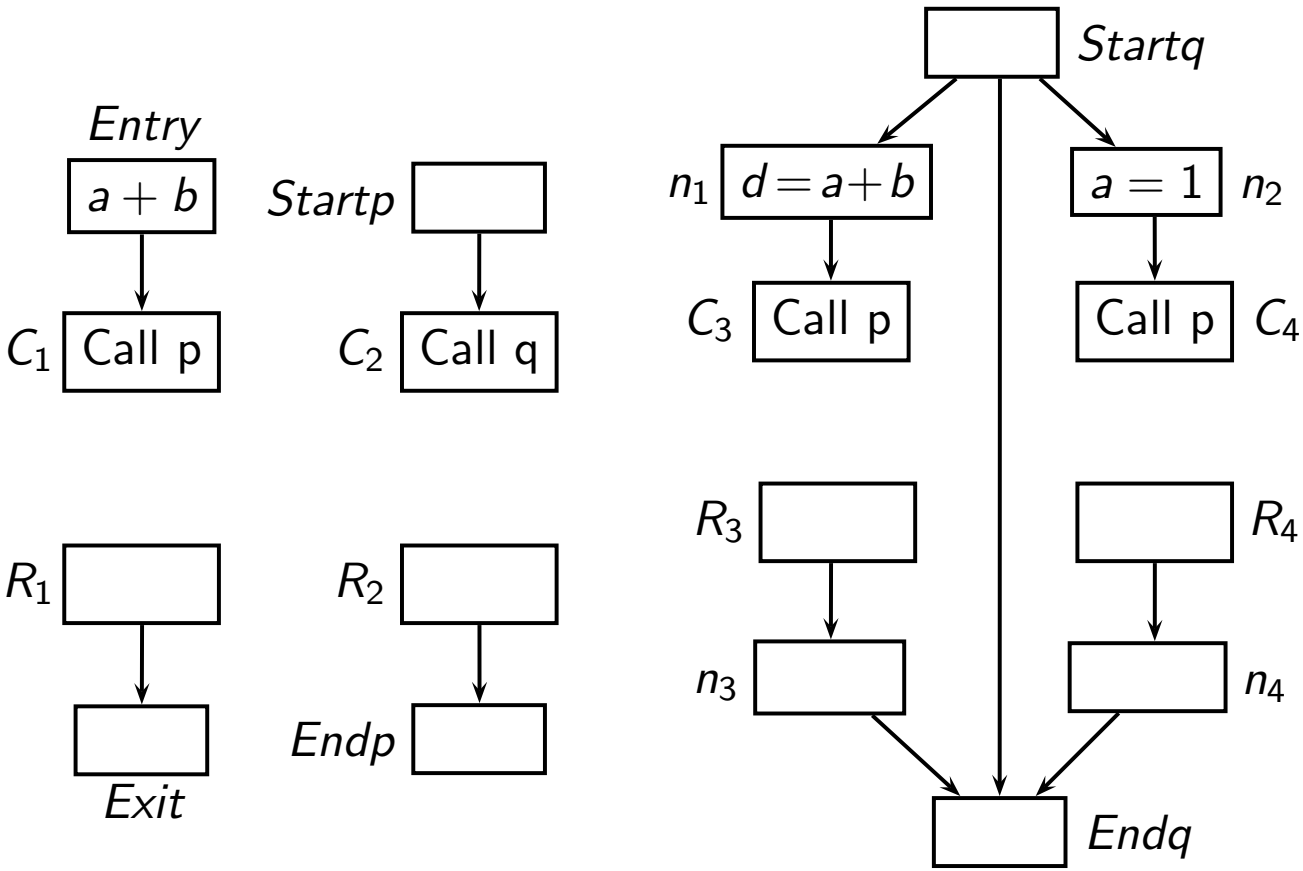
Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point
- The State of Art: Merge information across contexts for efficiency
  $\Rightarrow$ Significant imprecision in recursive programs
- Our Breakthrough:
- The Consequences:
- Further Goal:

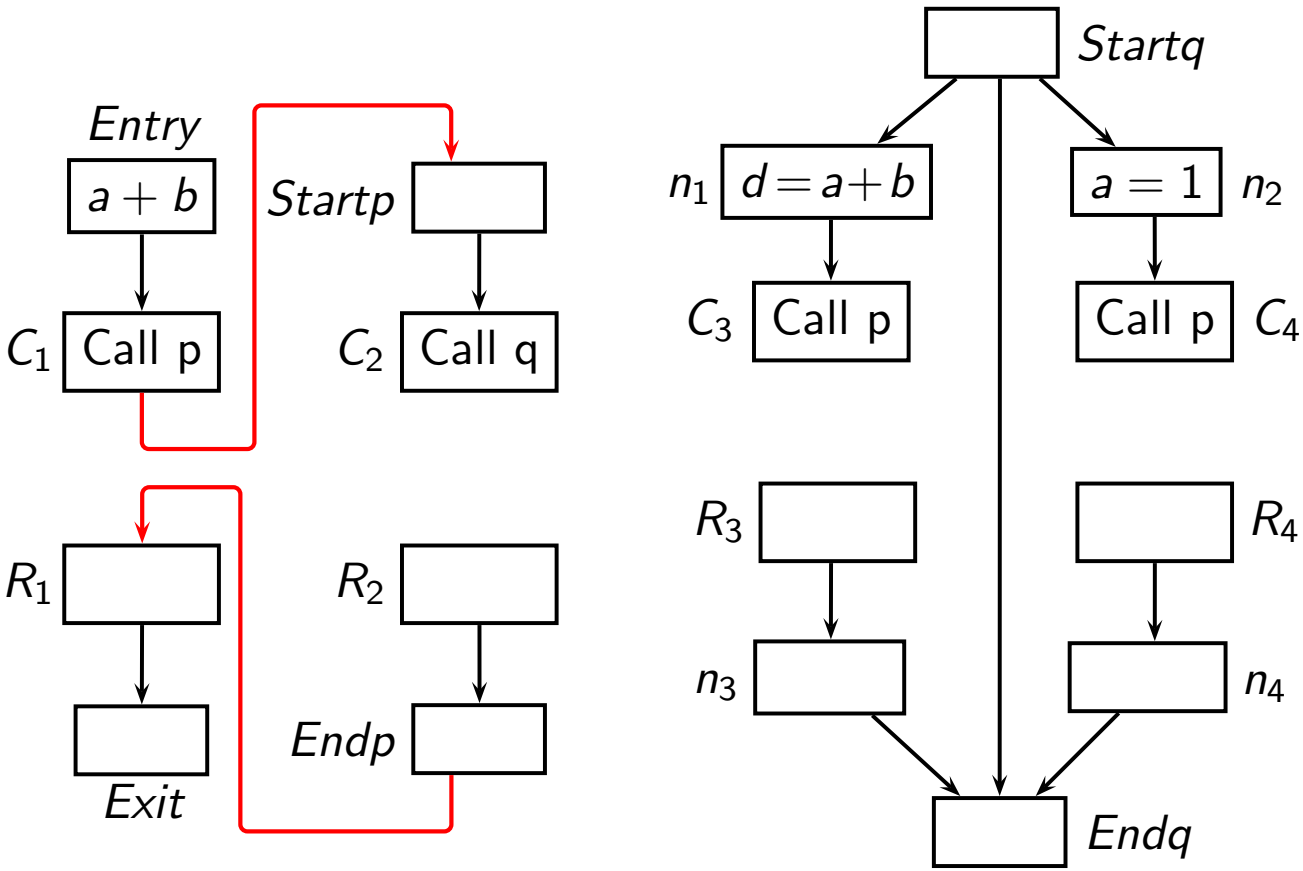
Defining Interprocedural Context for Static Analysis



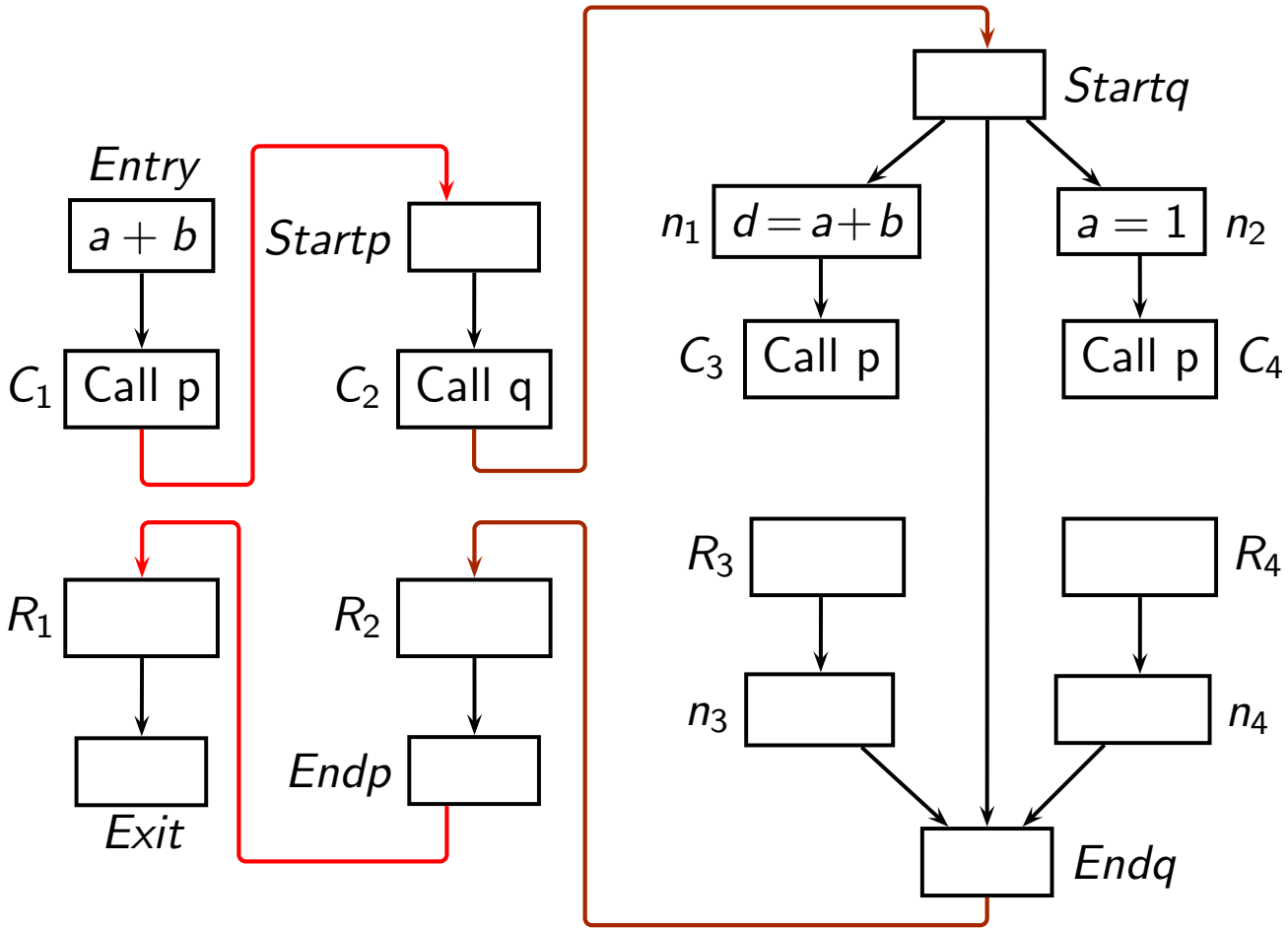
Defining Interprocedural Context for Static Analysis



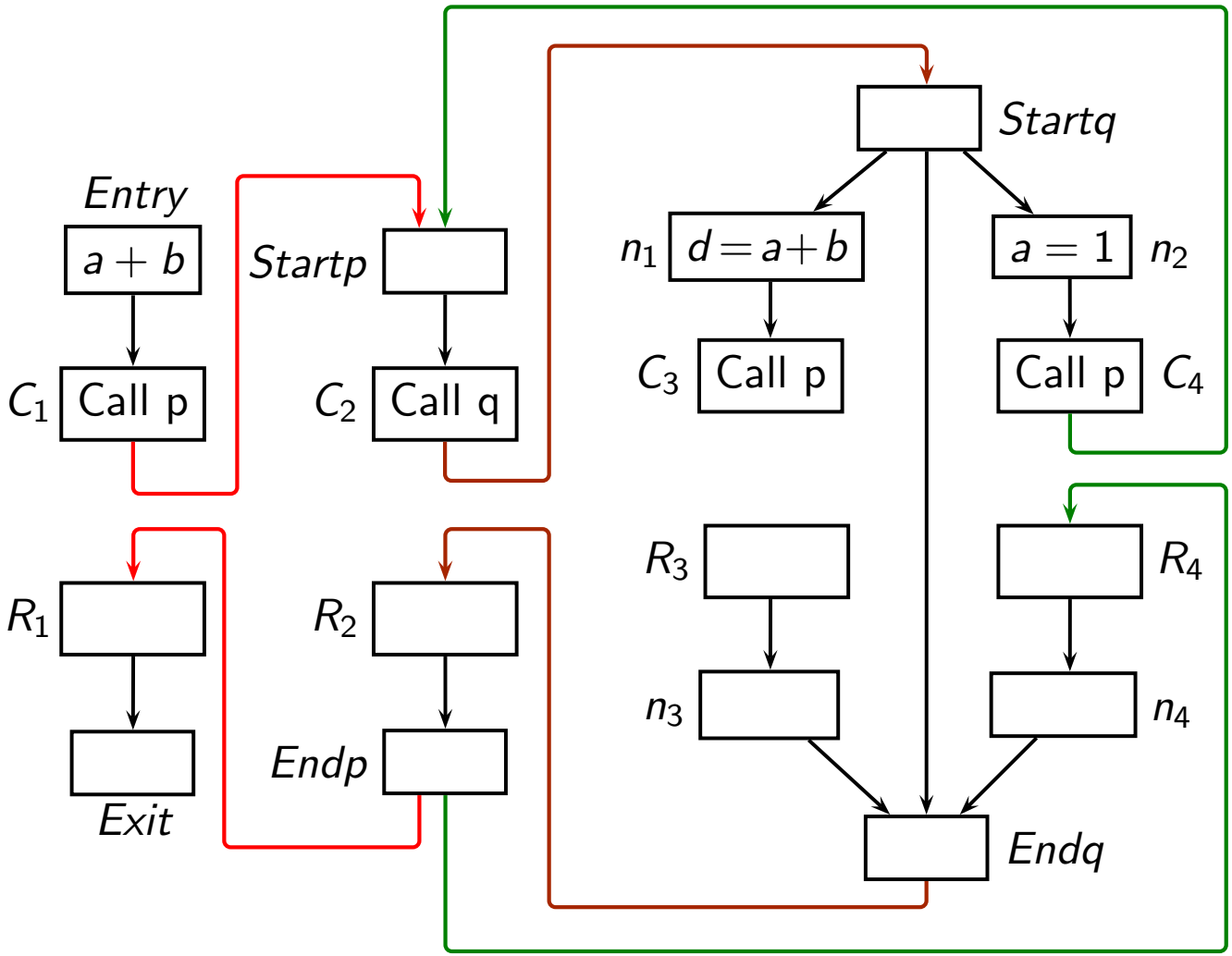
Defining Interprocedural Context for Static Analysis



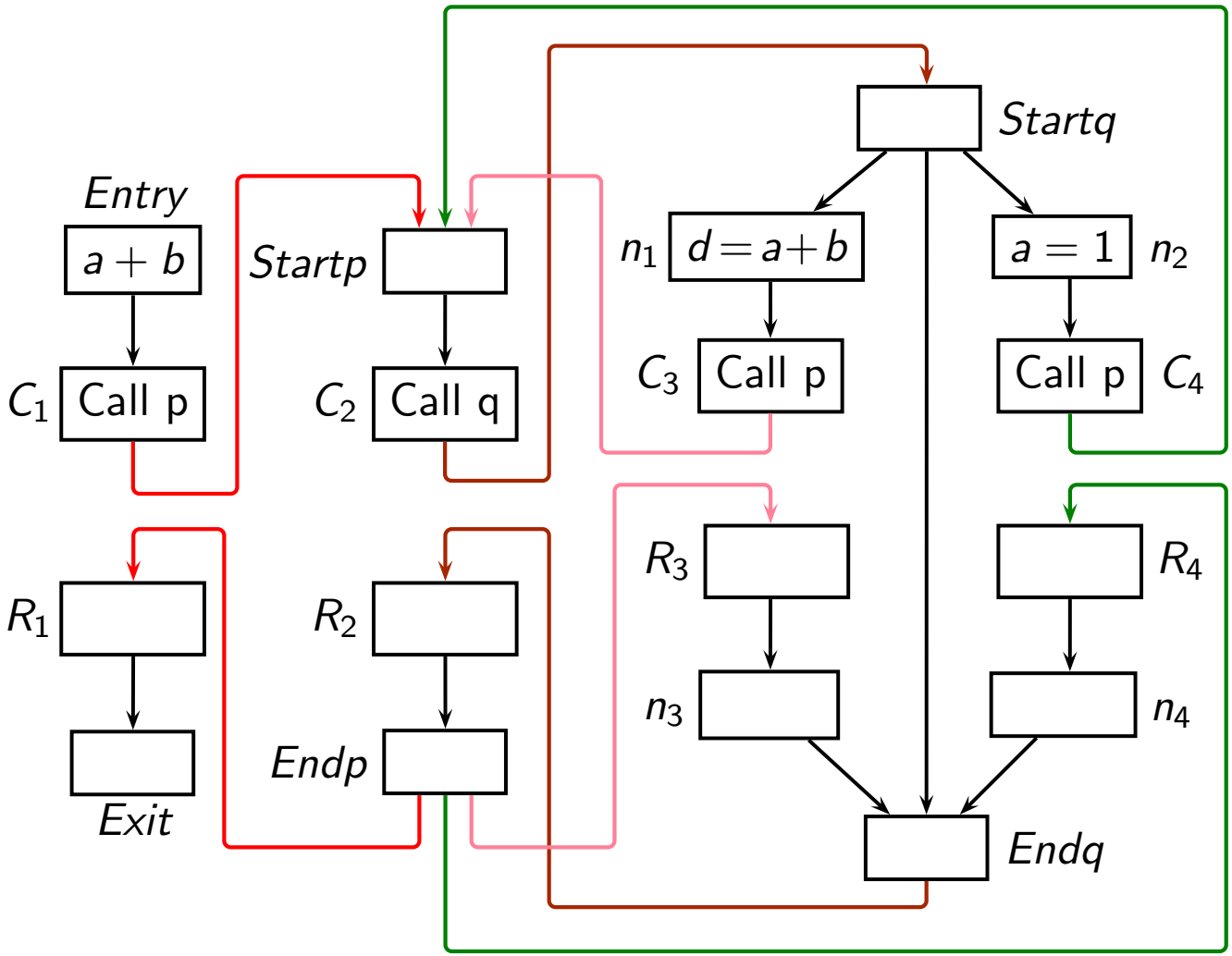
Defining Interprocedural Context for Static Analysis



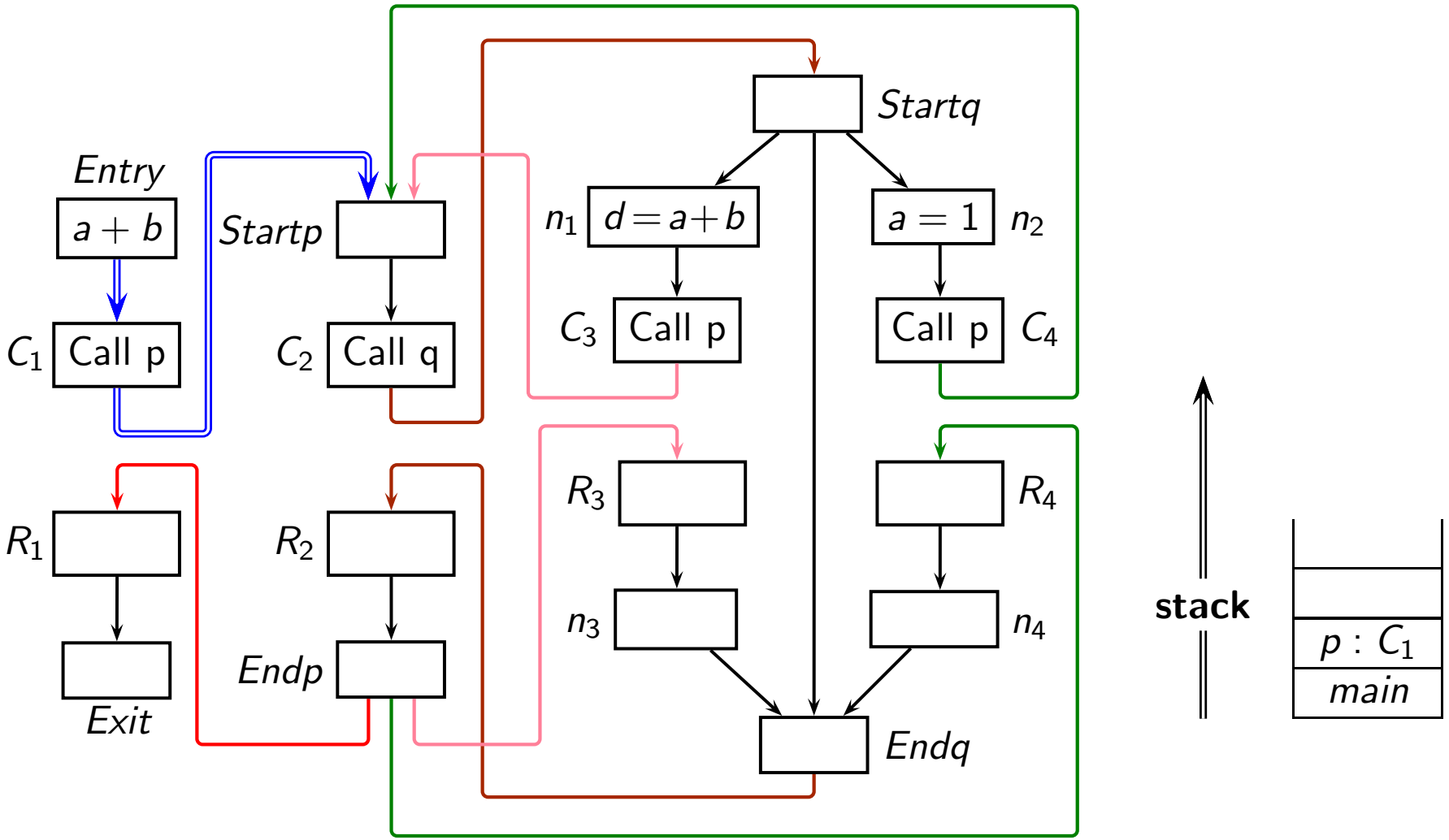
Defining Interprocedural Context for Static Analysis



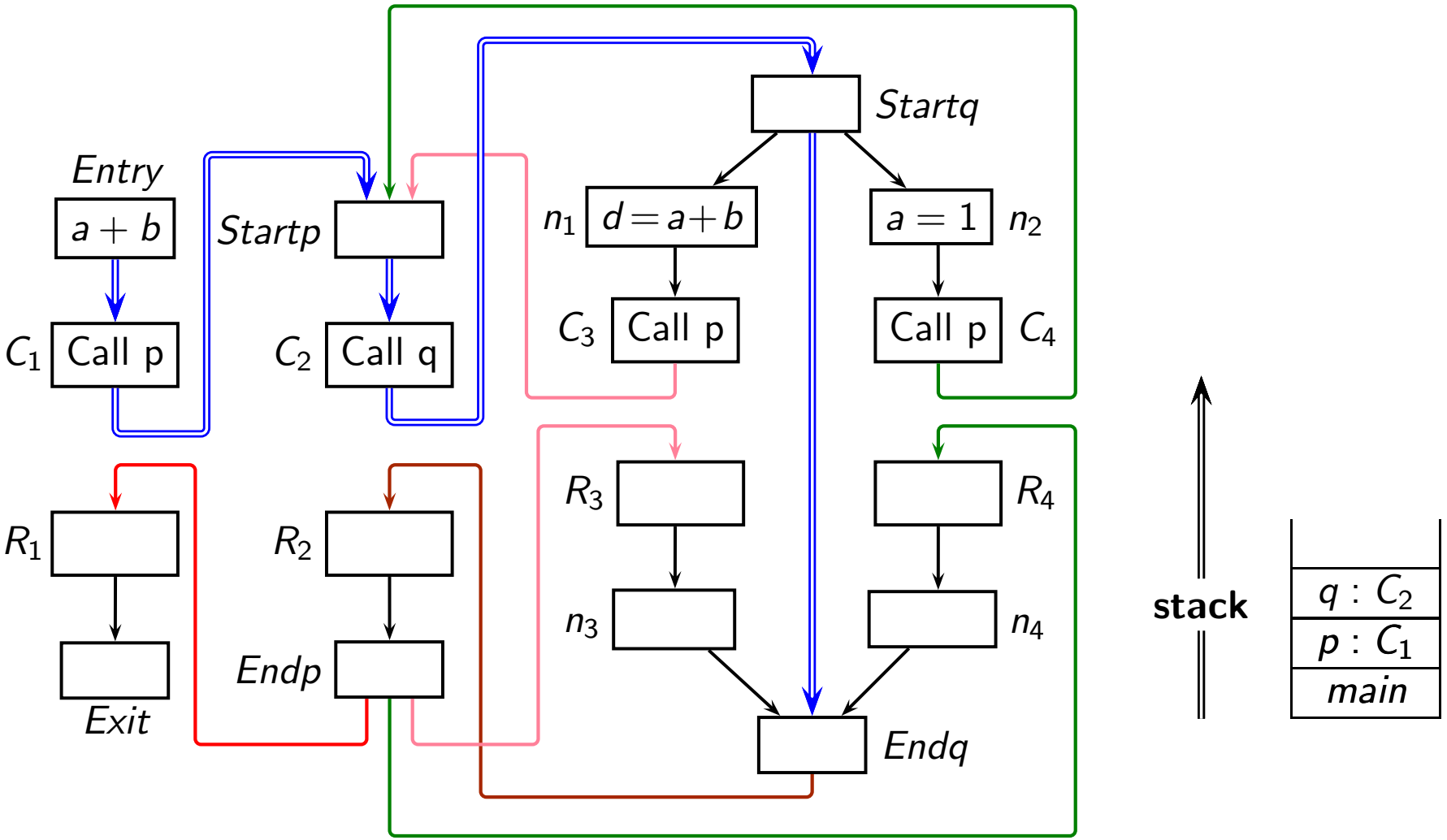
Defining Interprocedural Context for Static Analysis



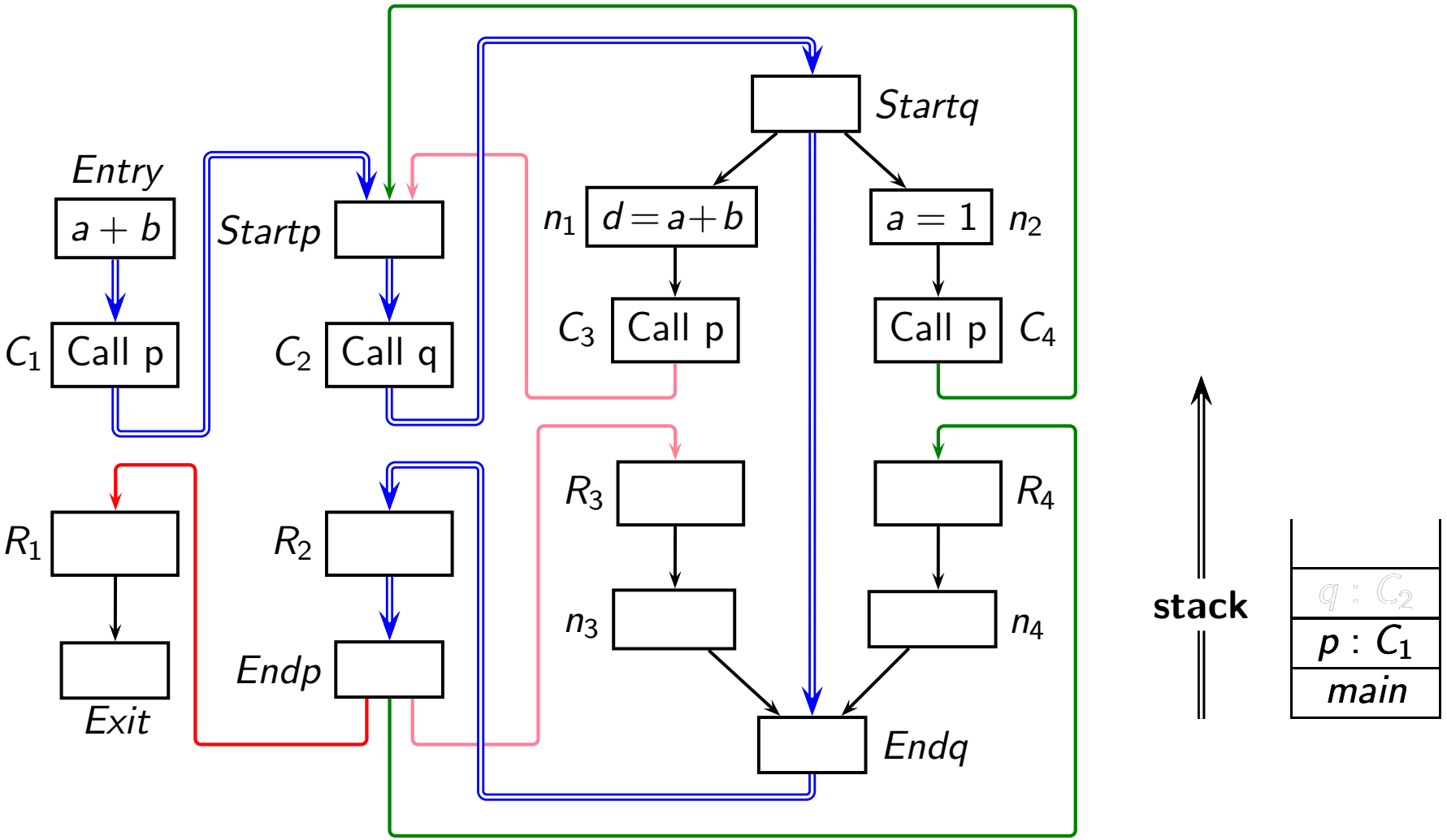
Defining Interprocedural Context for Static Analysis



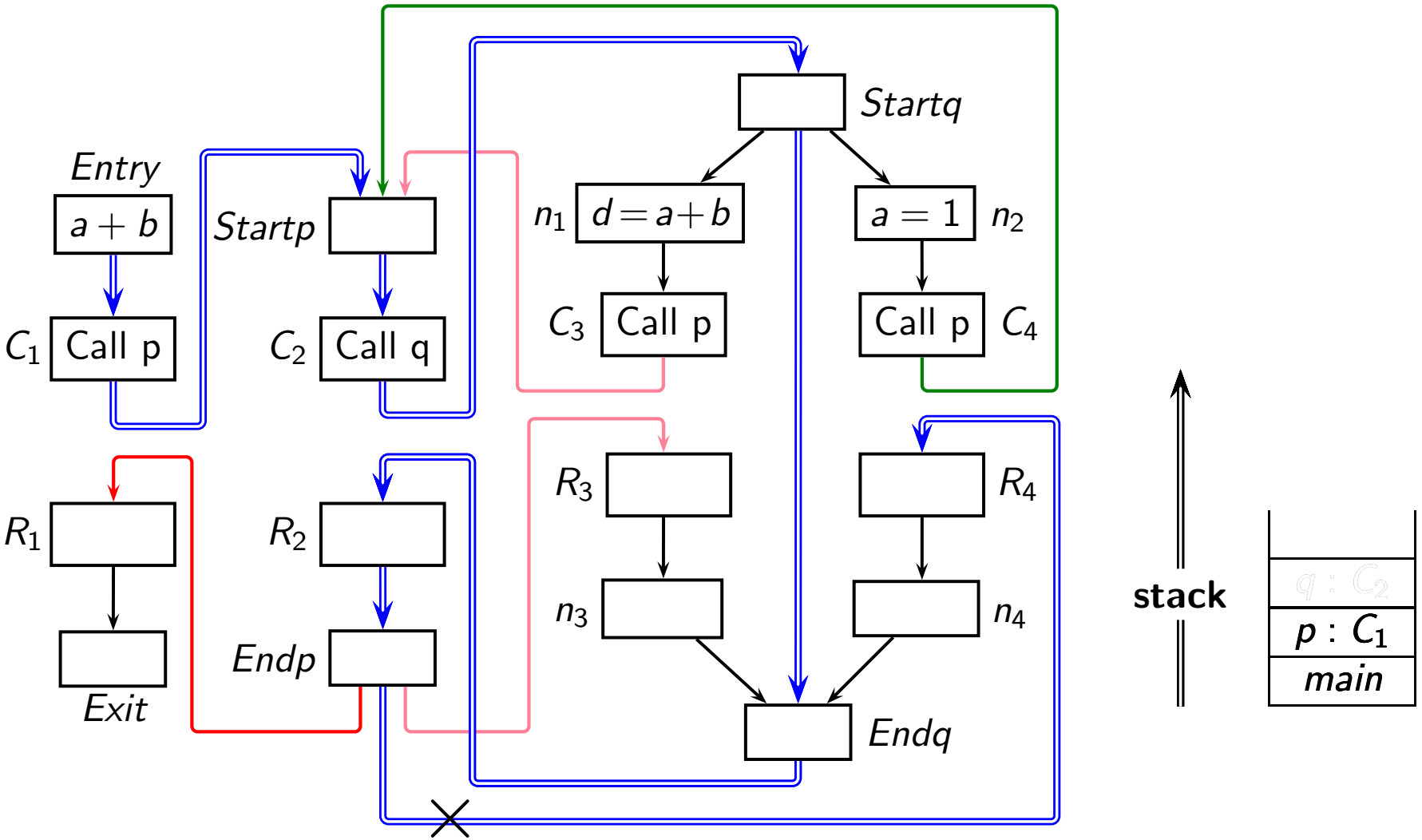
Defining Interprocedural Context for Static Analysis



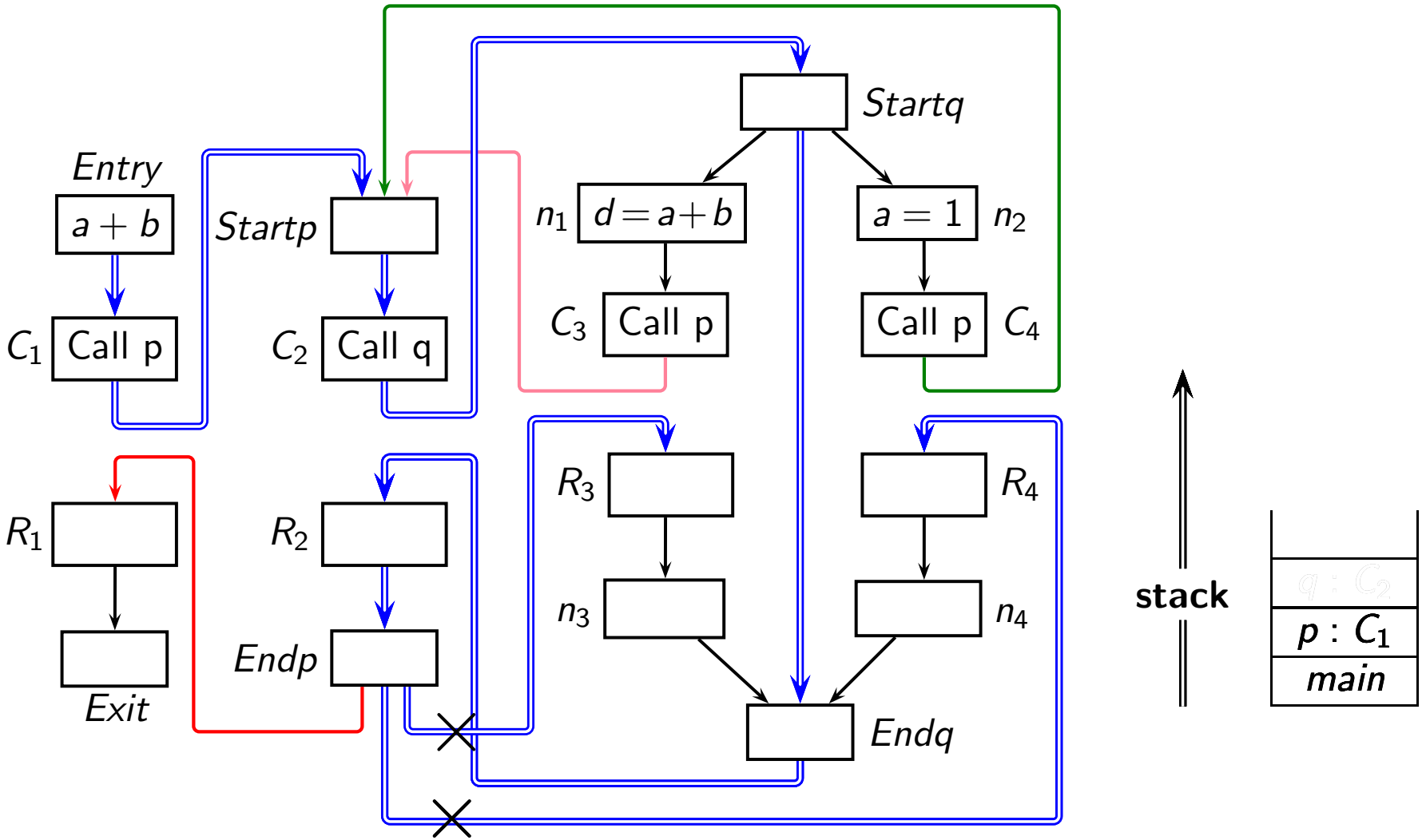
Defining Interprocedural Context for Static Analysis



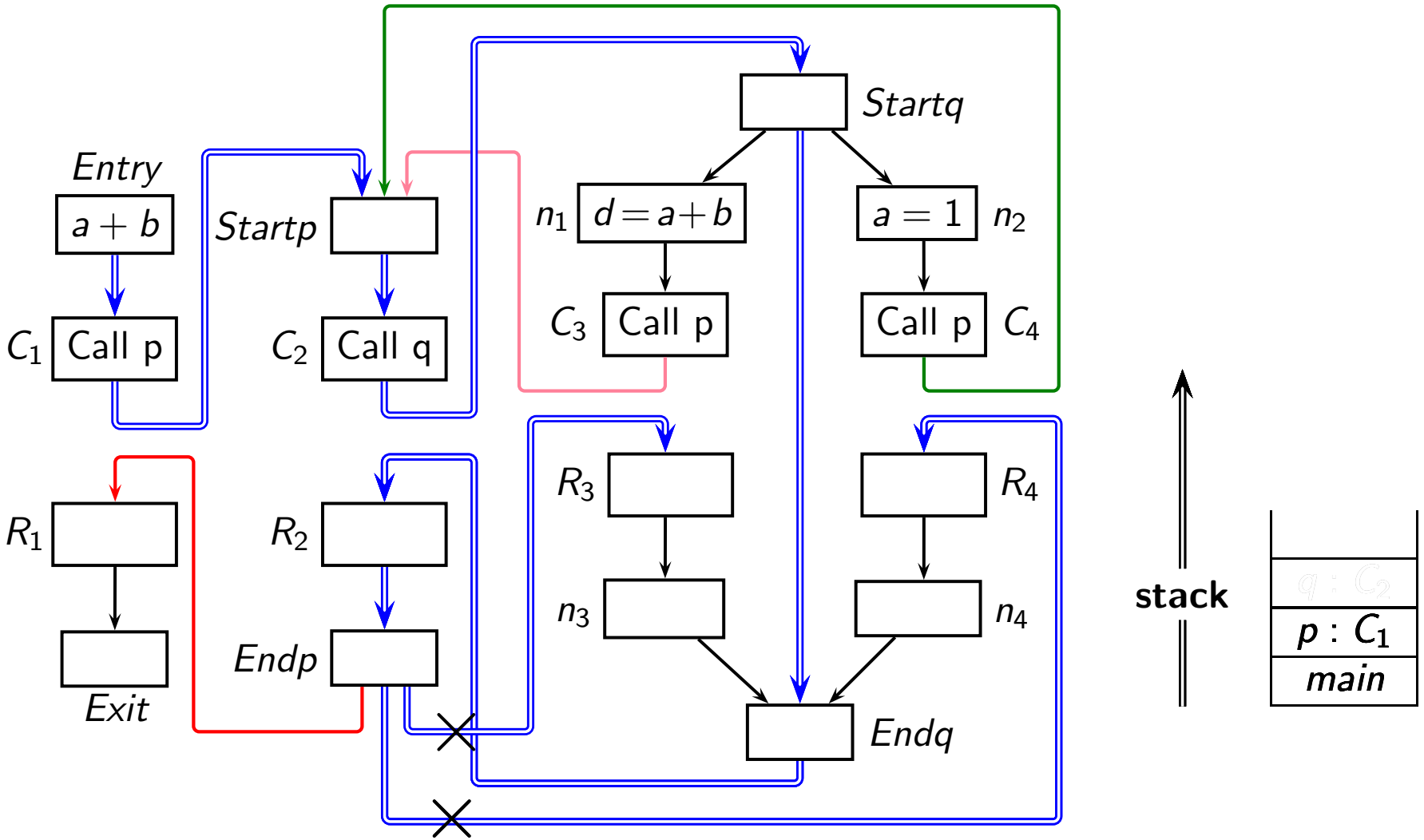
Defining Interprocedural Context for Static Analysis



Defining Interprocedural Context for Static Analysis



Defining Interprocedural Context for Static Analysis



Context is defined by stack snapshot ⇒ Unbounded number of contexts

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point
- The State of Art: Merge information across contexts for efficiency
  $\Rightarrow$ Significant imprecision in recursive programs
- Our Breakthrough:
- The Consequences:
- Further Goal:

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point
- The State of Art: Merge information across contexts for efficiency
  $\Rightarrow$ Significant imprecision in recursive programs
- Our Breakthrough: Clean, formally provable characterizations to
  - discard redundant contexts at the start of every procedure, and
  - simulate regeneration contexts at the end of every procedure
- The Consequences:

- Further Goal:

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point
- The State of Art: Merge information across contexts for efficiency
  $\Rightarrow$ Significant imprecision in recursive programs
- Our Breakthrough: Clean, formally provable characterizations to
  - discard redundant contexts at the start of every procedure, and
  - simulate regeneration contexts at the end of every procedure
- The Consequences: Our implementation in GCC shows that our variant saves time and space by
- Further Goal:

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point
- The State of Art: Merge information across contexts for efficiency
  $\Rightarrow$ Significant imprecision in recursive programs
- Our Breakthrough: Clean, formally provable characterizations to
  - discard redundant contexts at the start of every procedure, and
  - simulate regeneration contexts at the end of every procedure
- The Consequences: Our implementation in GCC shows that our variant saves time and space by a factor of over a million!
- Further Goal:

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point
- The State of Art: Merge information across contexts for efficiency
  $\Rightarrow$ Significant imprecision in recursive programs
- Our Breakthrough: Clean, formally provable characterizations to
  - discard redundant contexts at the start of every procedure, and
  - simulate regeneration contexts at the end of every procedure
- The Consequences: Our implementation in GCC shows that our variant saves time and space by a factor of over a million!
- Further Goal: Design a scalable variant to analyze a million lines of code in a few minutes

Heap Reference Analysis [TOPLAS 2007]

- The Problem:
- Our Objectives:
- Main Challenge:
- Our Key Idea:
- Current status:
- Further Work:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives:
- Main Challenge:
- Our Key Idea:
- Current status:
- Further Work:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge:
- Our Key Idea:
- Current status:
- Further Work:

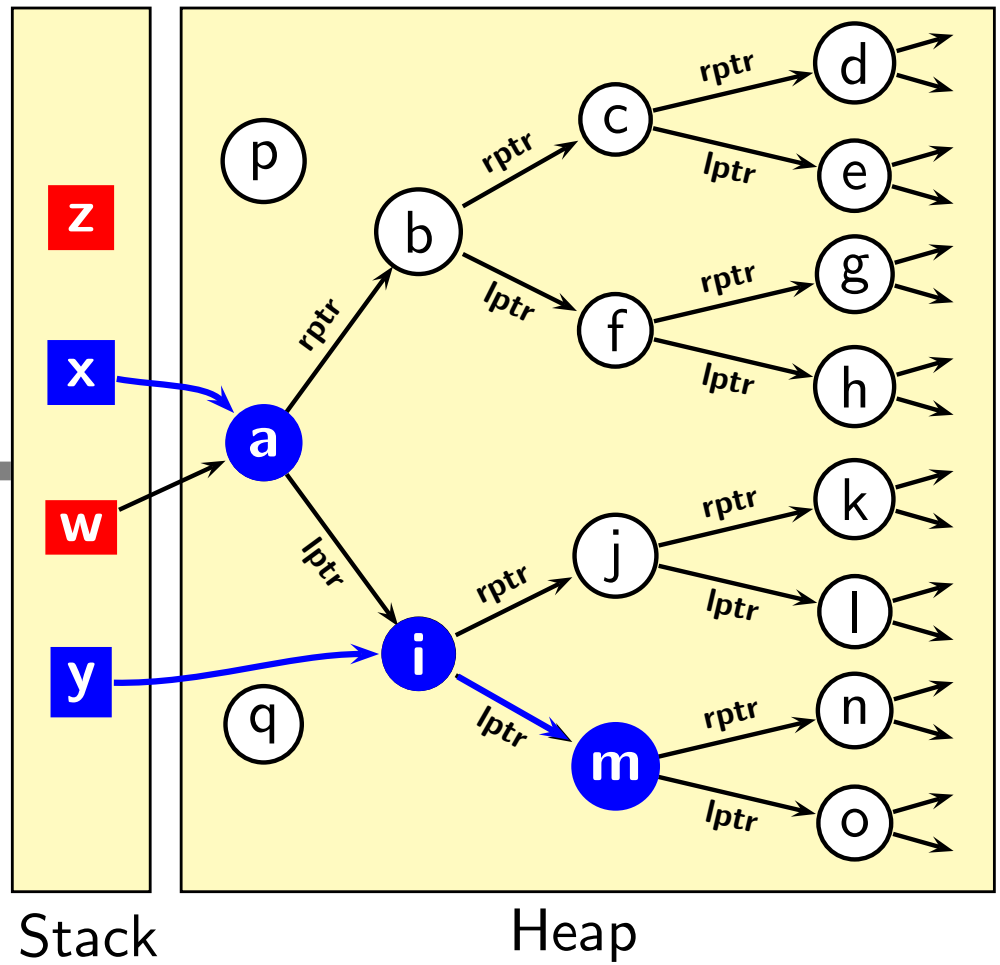
Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.
- Our Key Idea:
- Current status:
- Further Work:

Which Heap Memory Nodes Can be Statically Marked as Live?

If the while loop is not executed even once.

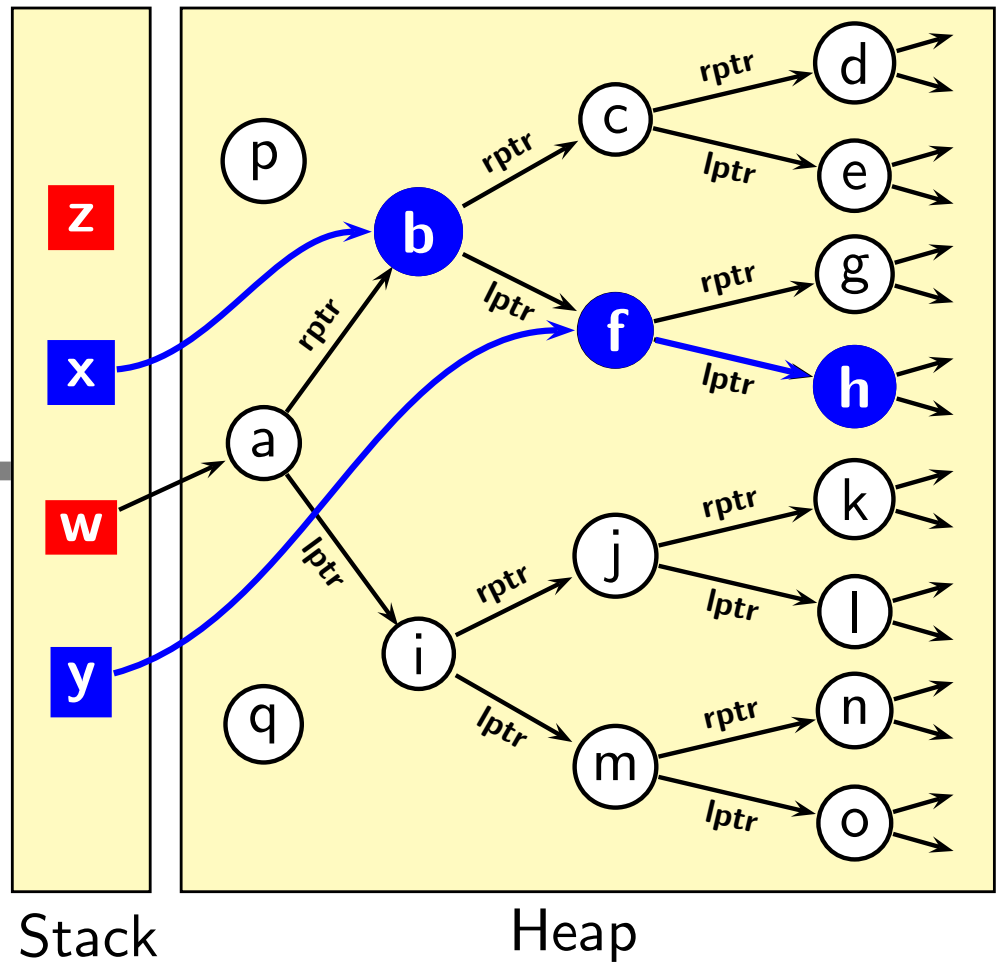
```
1    w = x          // x points to m_a
2    while  (x.data < max)
3           x = x.rptr
4    y = x.lptr
5    z = New  class_of_z
6    y = y.lptr
7    z.sum = x.data + y.data
```



Which Heap Memory Nodes Can be Statically Marked as Live?

If the while loop is executed once.

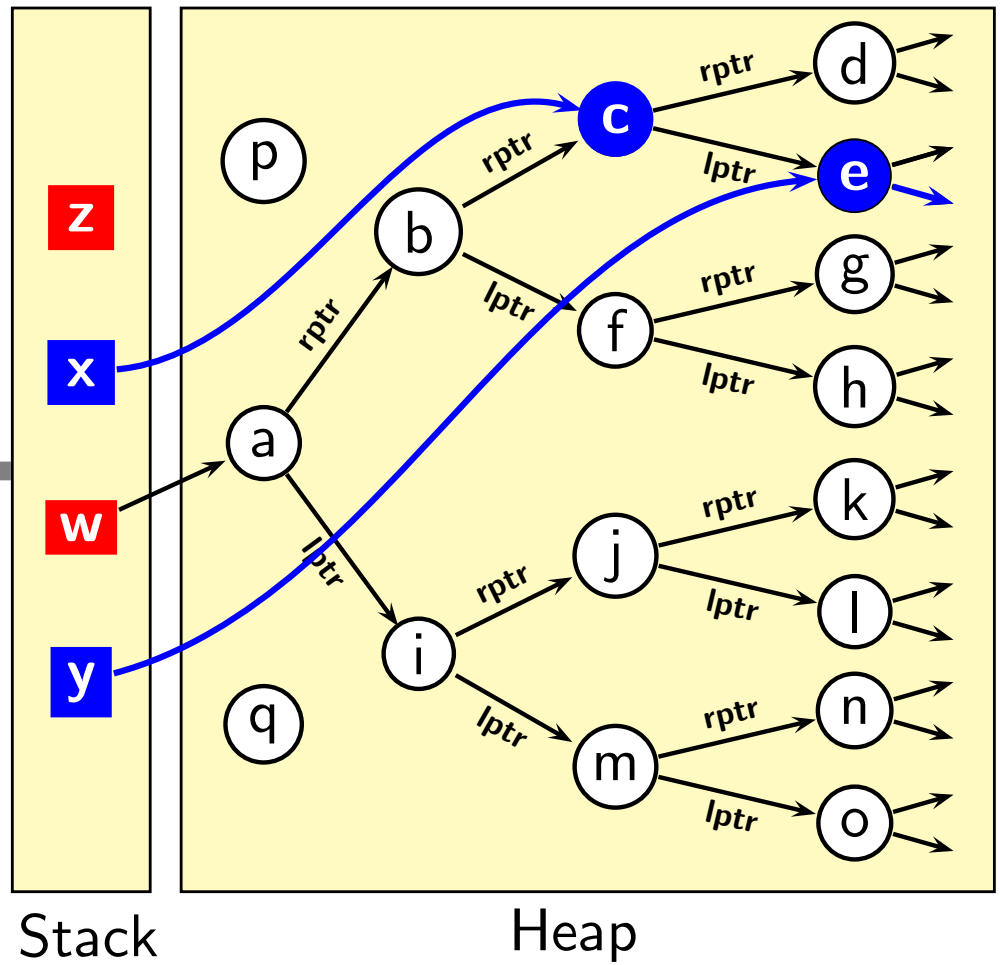
```
1   w = x          // x points to m_a
2   while (x.data < max)
3         x = x.rptr
4   y = x.lptr
5   z = New class_of_z
6   y = y.lptr
7   z.sum = x.data + y.data
```



Which Heap Memory Nodes Can be Statically Marked as Live?

If the while loop is executed twice.

```
1   w = x          // x points to m_a
2   while (x.data < max)
3         x = x.rptr
4   y = x.lptr
5   z = New class_of_z
6   y = y.lptr
7   z.sum = x.data + y.data
```



Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.
- Our Key Idea:
- Current status:
- Further Work:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.
- Our Key Idea: Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.
- Current status:

- Further Work:

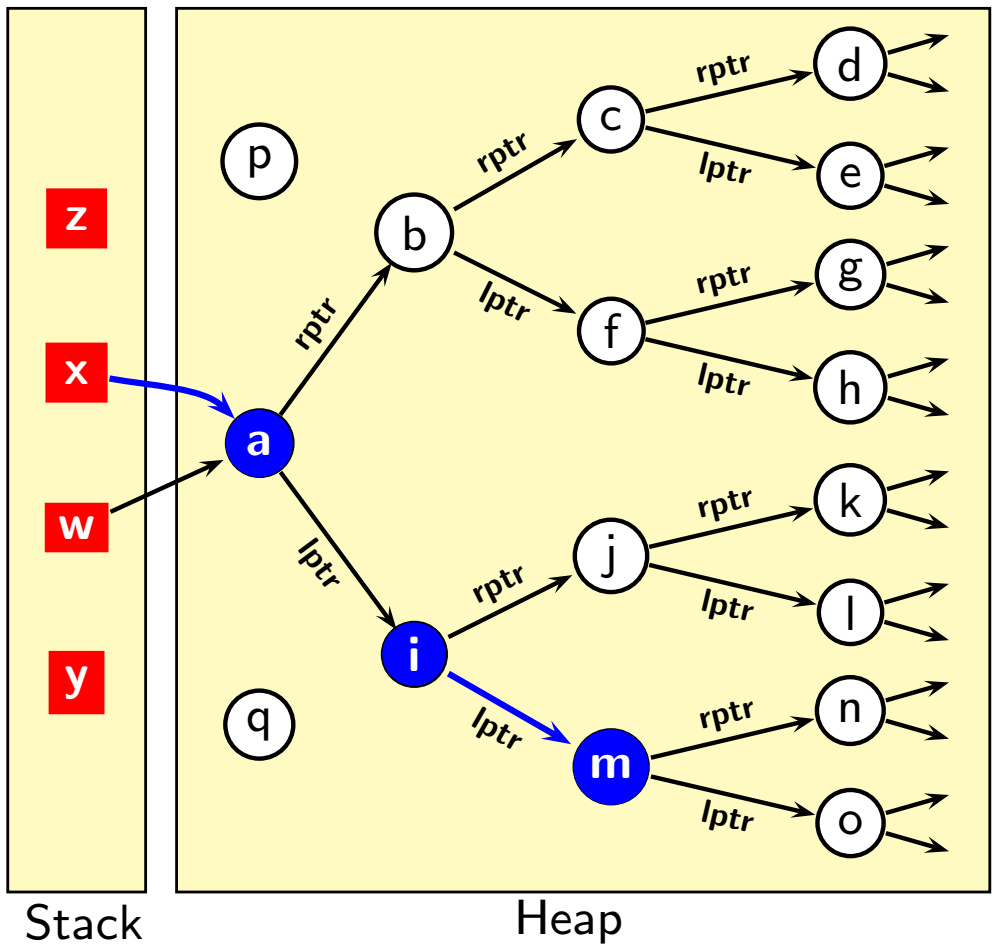
Our Solution

		y = z = null
1	w = x	
		w = null
2	while (x.data < max)	
	{	x.lptr = null
3	x = x.rptr }	
		x.rptr = x.lptr.rptr = null
		x.lptr.lptr.lptr = null
		x.lptr.lptr.rptr = null
4	y = x.lptr	
		x.lptr = y.rptr = null
		y.lptr.lptr = y.lptr.rptr = null
5	z = *New class_of_z*	
		z.lptr = z.rptr = null
6	y = y.lptr	
		y.lptr = y.rptr = null
7	z.sum = x.data + y.data	
		x = y = z = null

Heap Reference Analysis: Our Solution

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr      }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

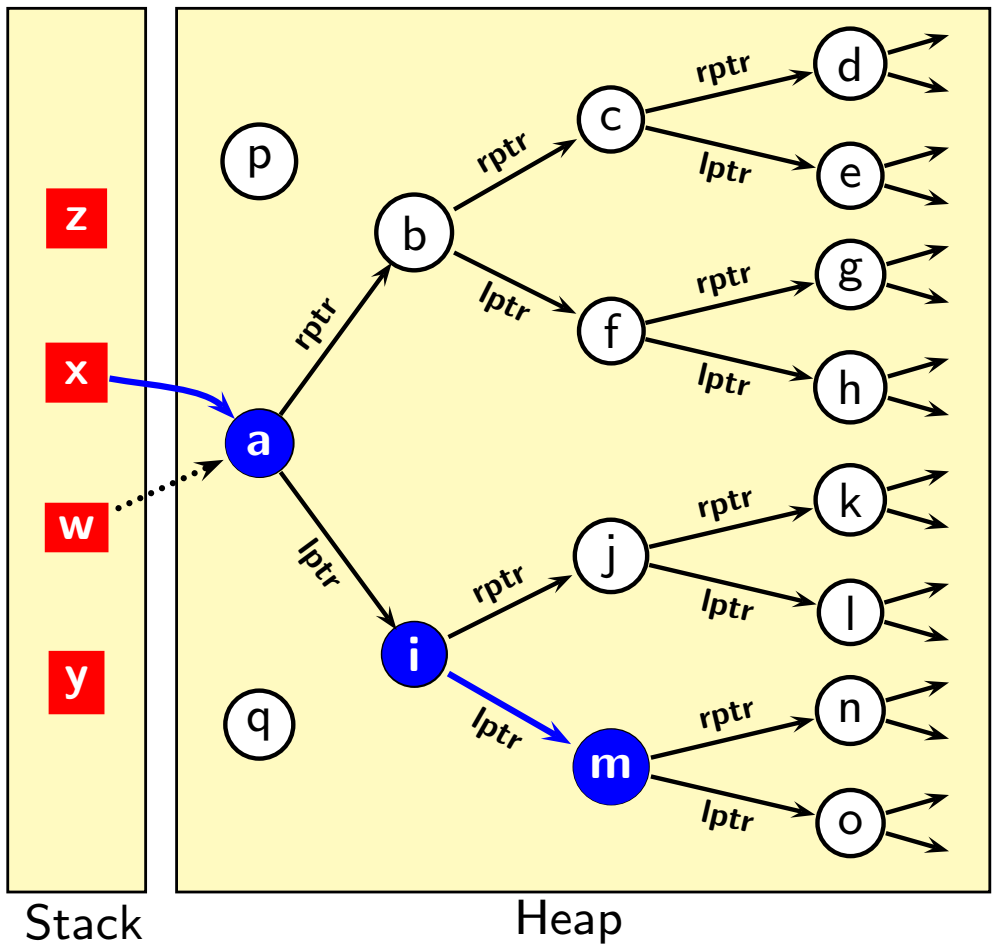
While loop is not executed even once



Heap Reference Analysis: Our Solution

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr      }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

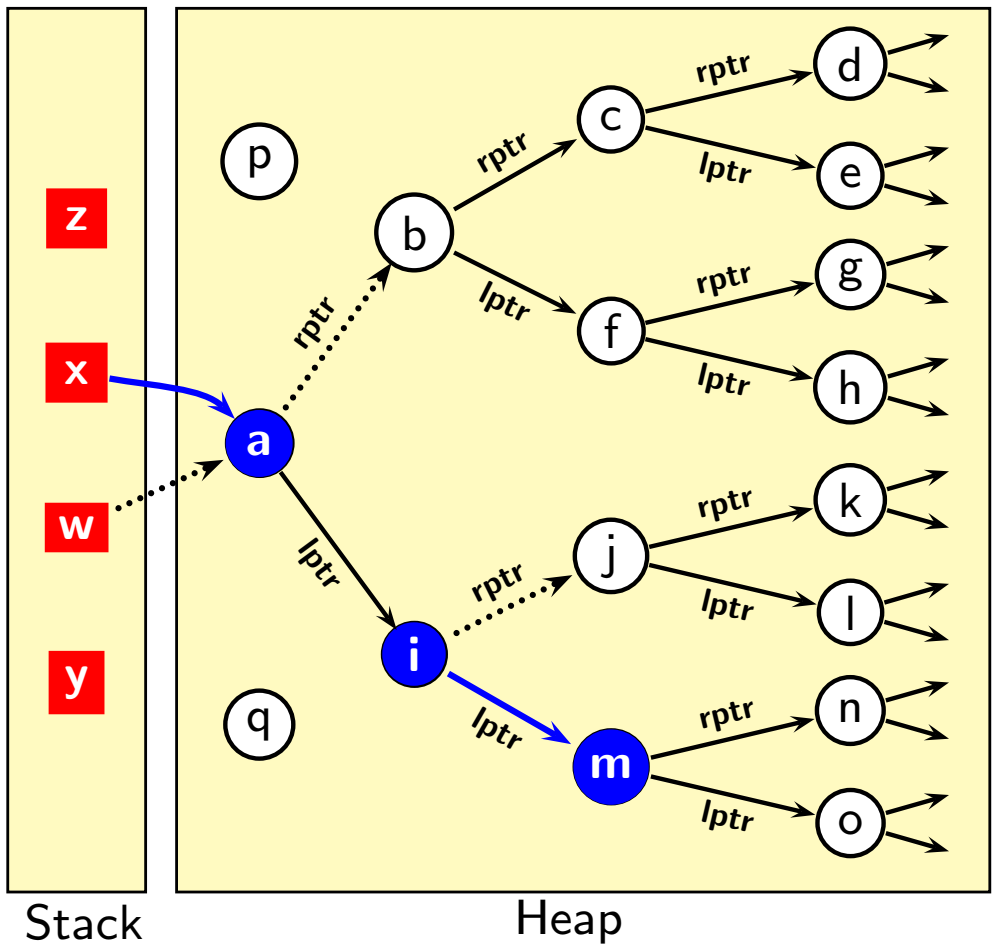
While loop is not executed even once



Heap Reference Analysis: Our Solution

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr      }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

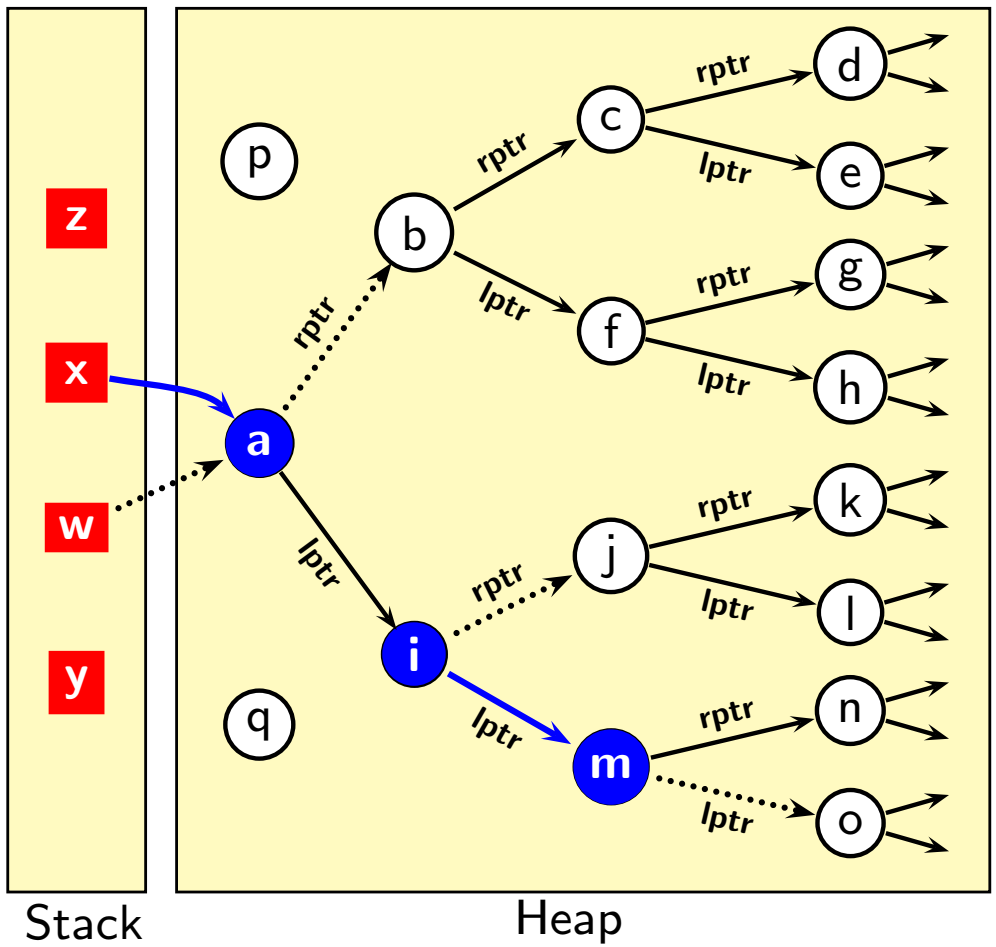
While loop is not executed even once



Heap Reference Analysis: Our Solution

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr      }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

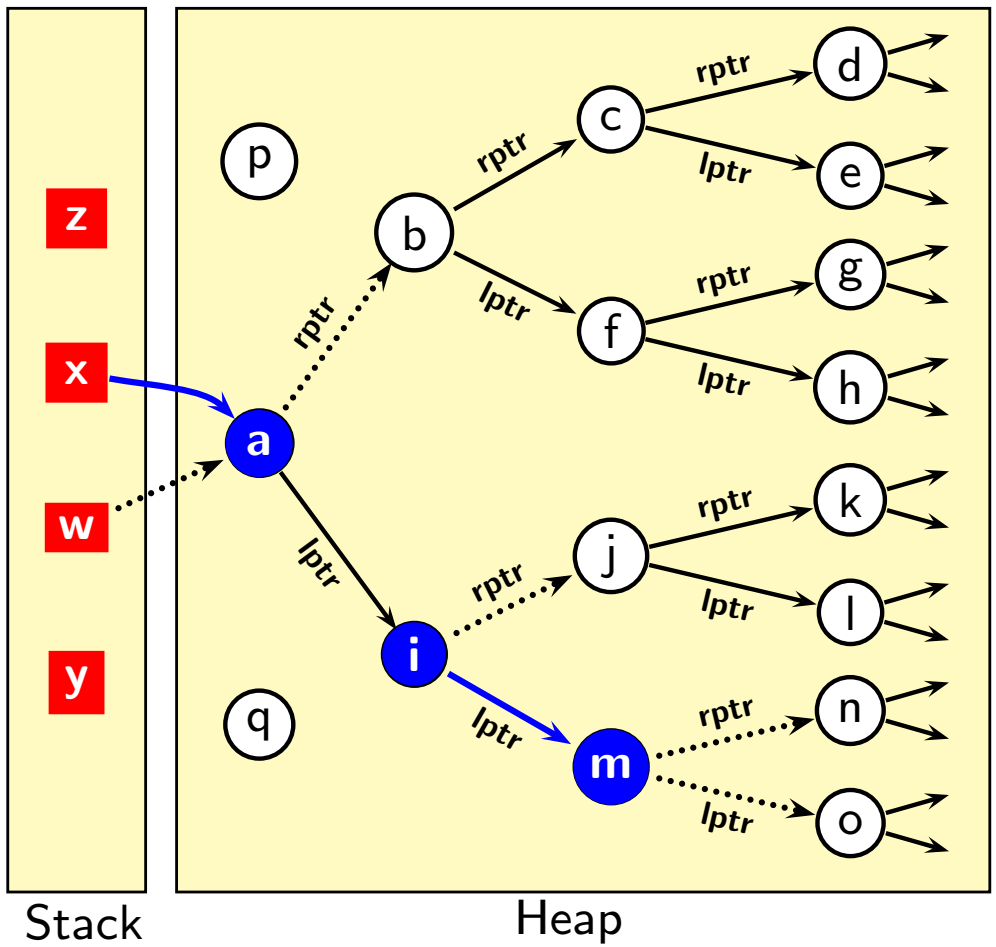
While loop is not executed even once



Heap Reference Analysis: Our Solution

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr      }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

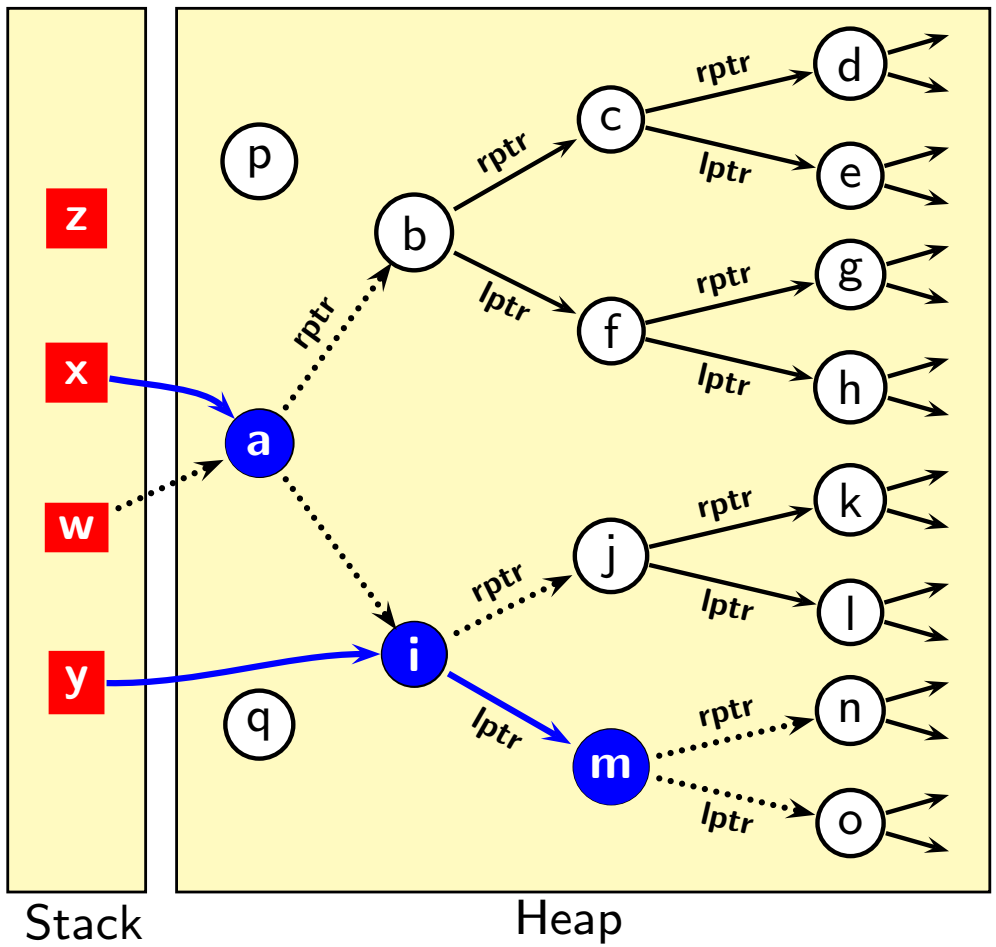
While loop is not executed even once



Heap Reference Analysis: Our Solution

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr      }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

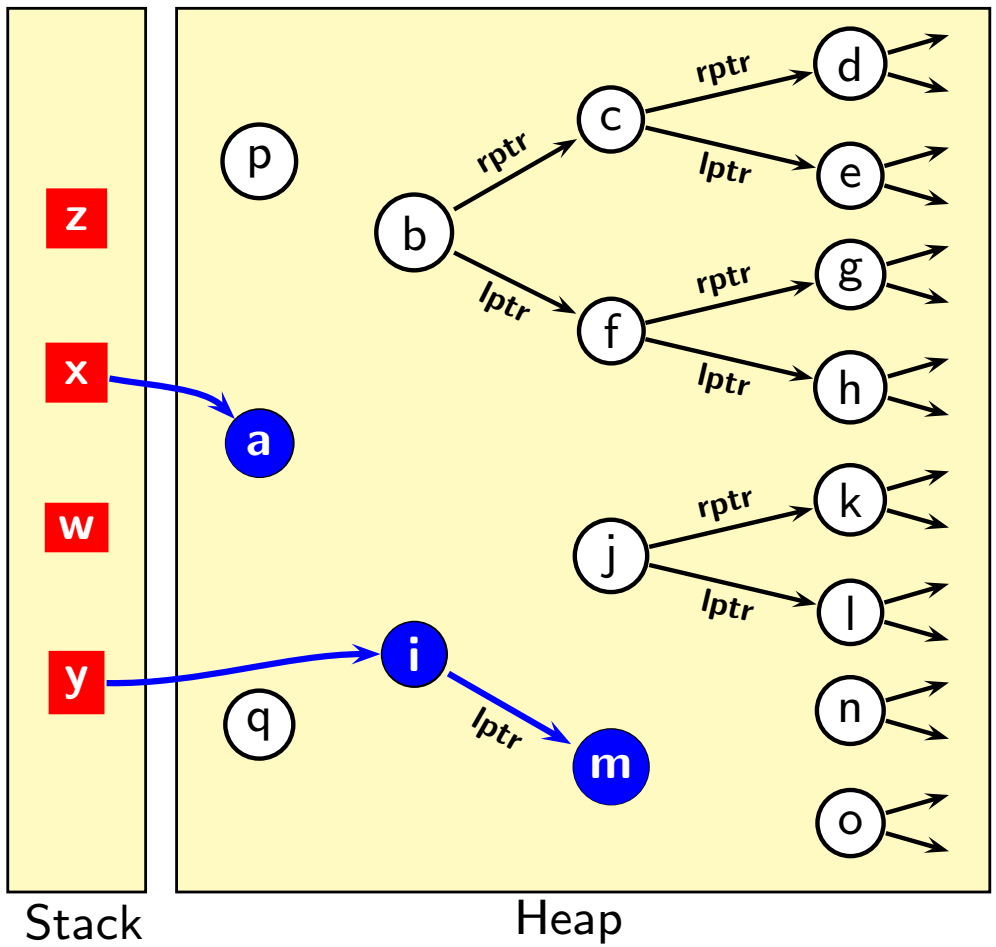
While loop is not executed even once



Heap Reference Analysis: Our Solution

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr      }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

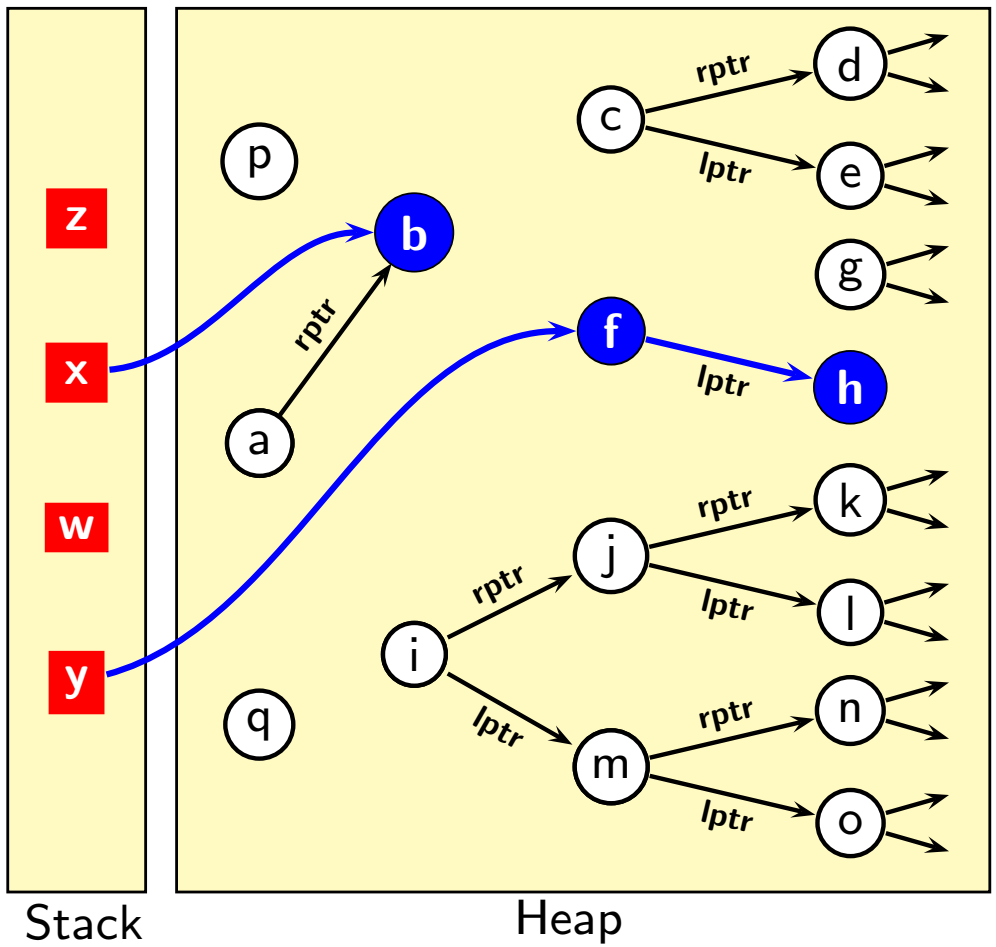
While loop is not executed even once



Heap Reference Analysis: Our Solution

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr      }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

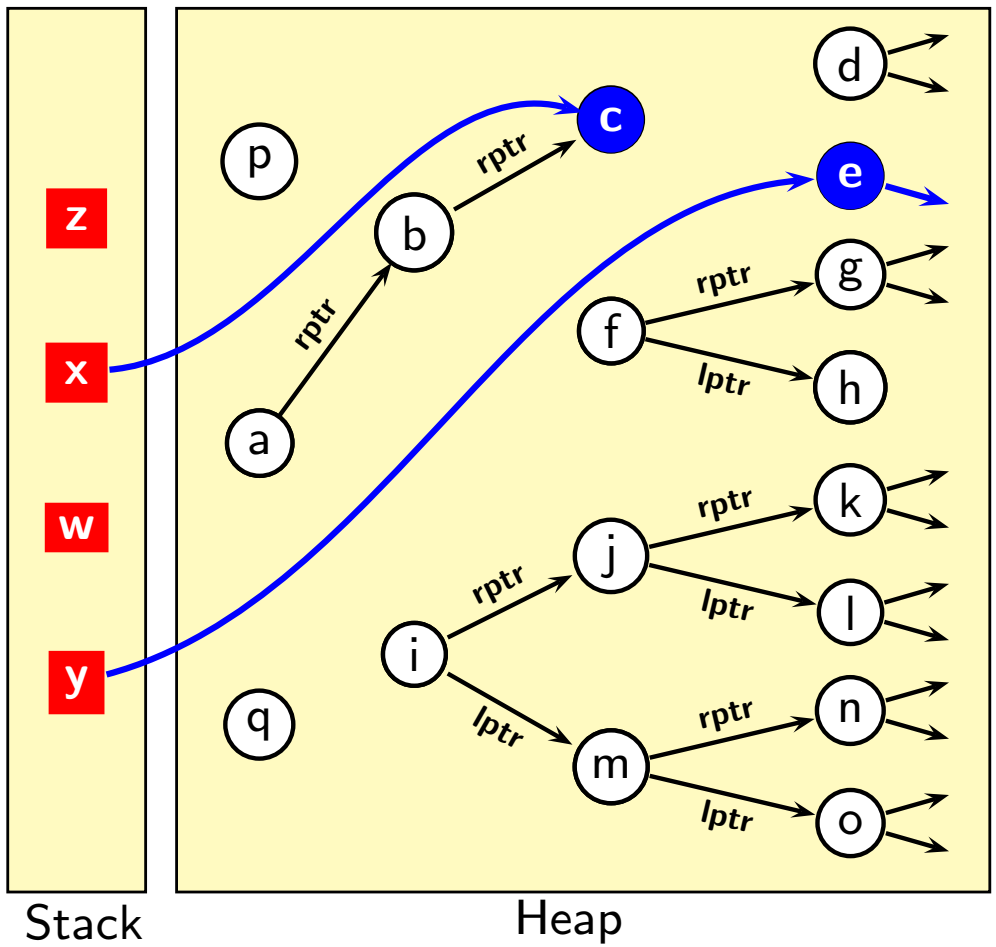
While loop is executed once



Heap Reference Analysis: Our Solution

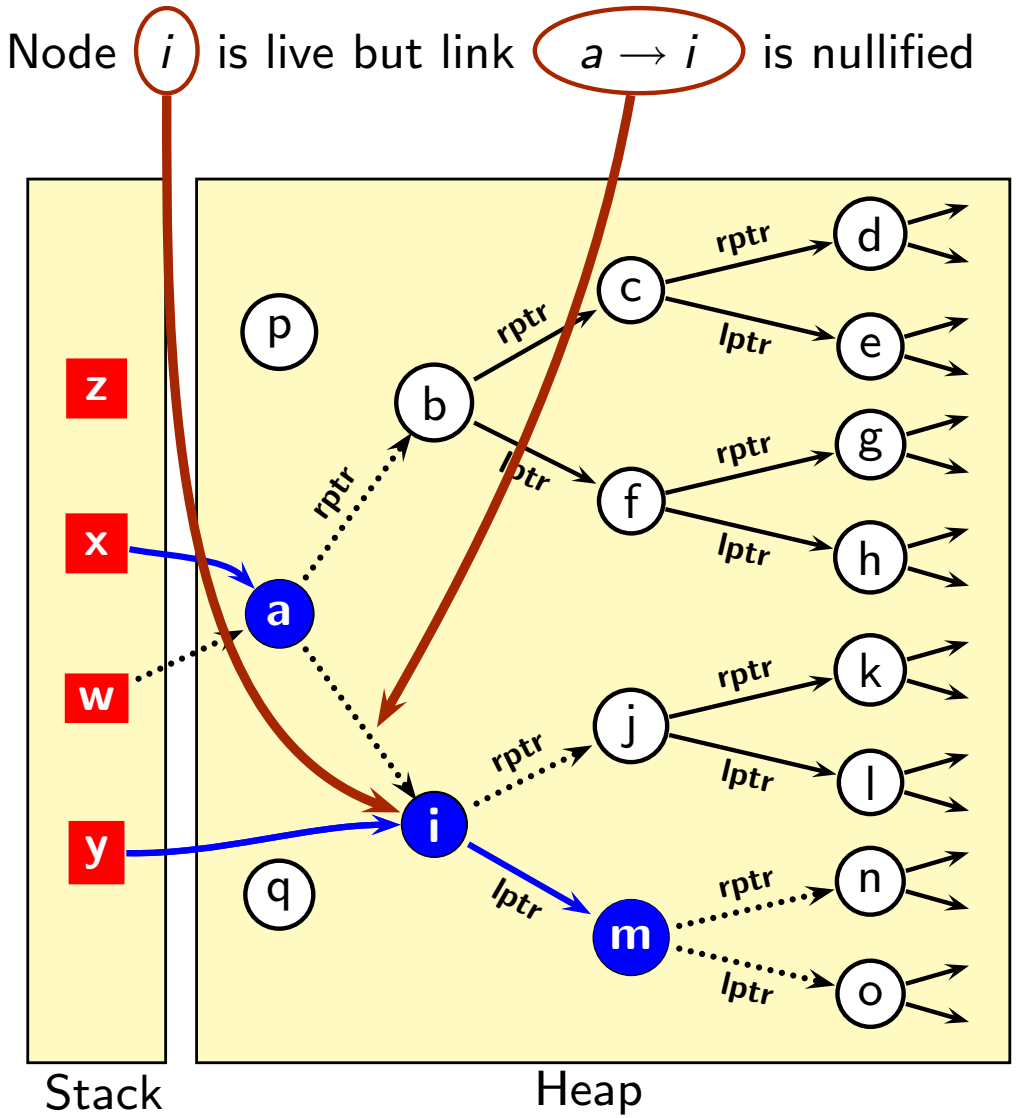
```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr      }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

While loop is executed twice



Some Observations

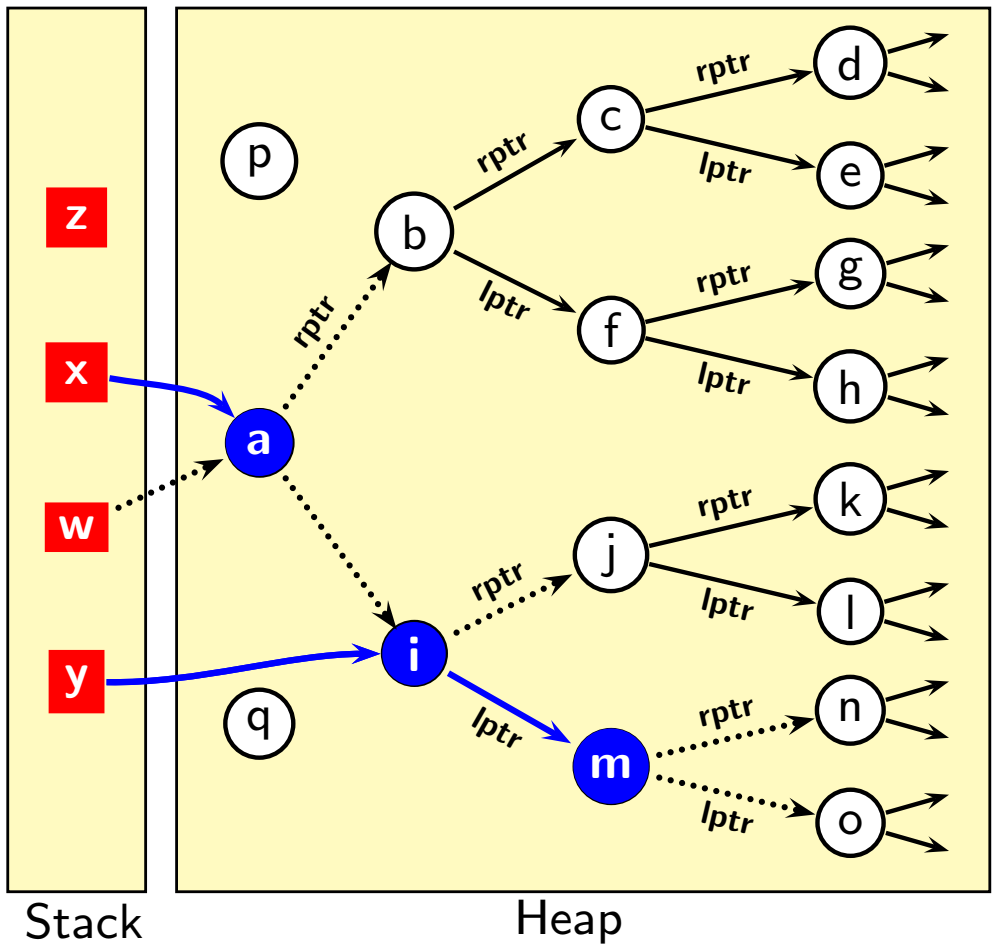
```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr      }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```



Some Observations

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr        }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

New access expressions are created.
Can they cause exceptions?



Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.
- Our Key Idea: Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.
- Current status:

- Further Work:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.
- Our Key Idea: Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.
- Current status: Theory and prototype implementation (at the intraprocedural level) ready for Java.
- Further Work:

Heap Reference Analysis [TOPLAS 2007]

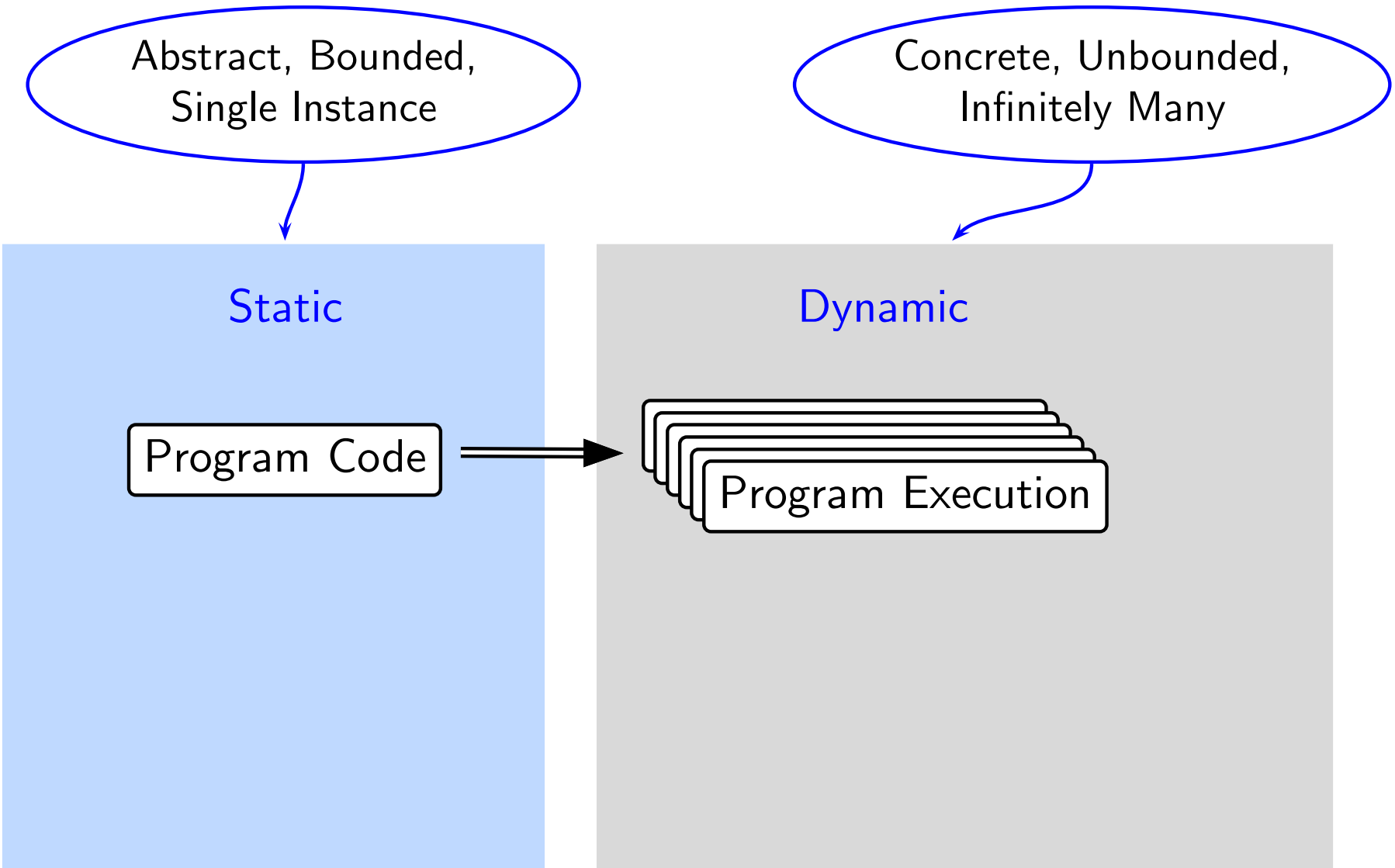
- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.
- Our Key Idea: Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.
- Current status: Theory and prototype implementation (at the intraprocedural level) ready for Java.
- Further Work:
  - Improve alias analysis
  - Interprocedural implementation and Performance tuning

BTW, What is Static Analysis of Heap?

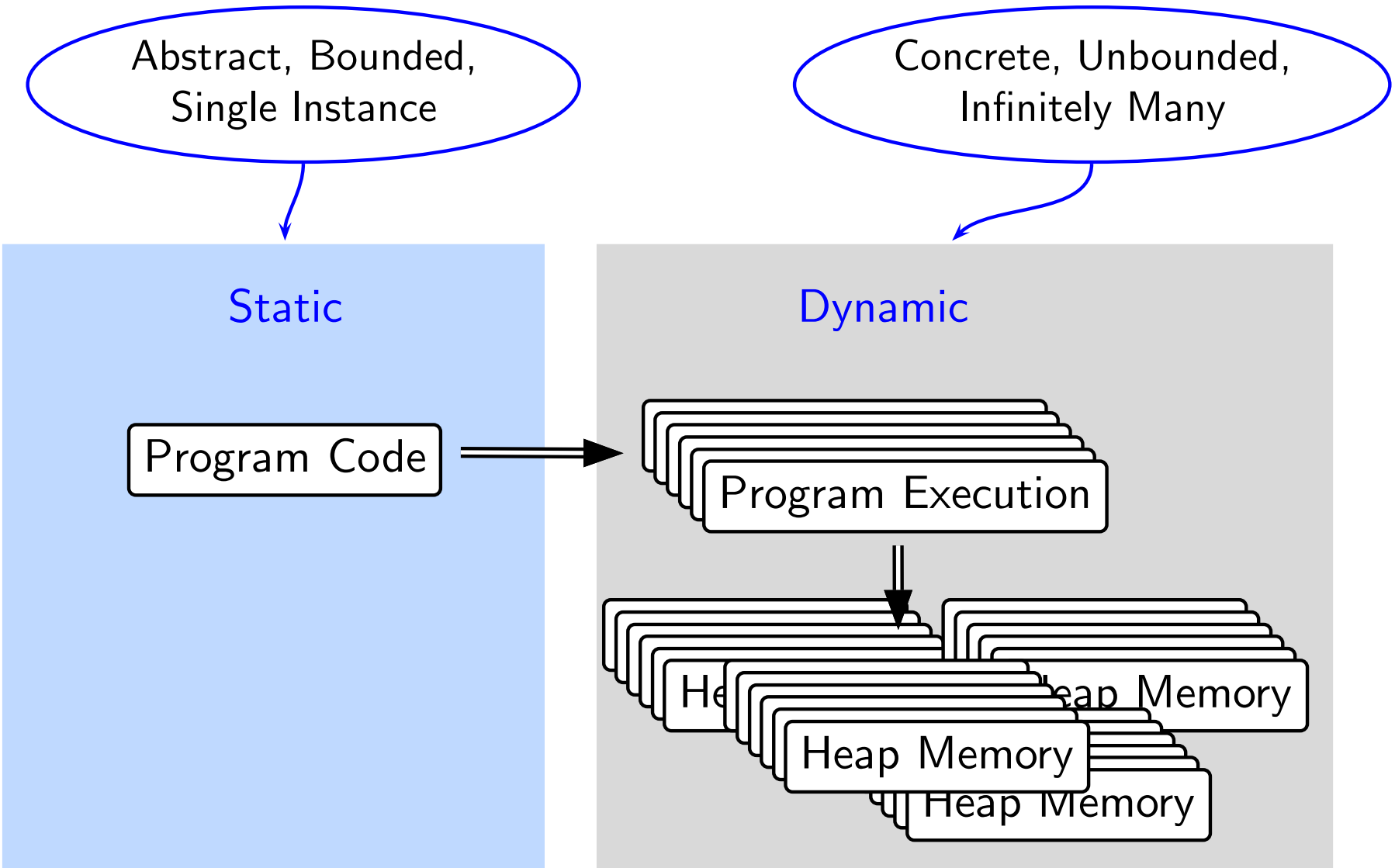
Static

Dynamic

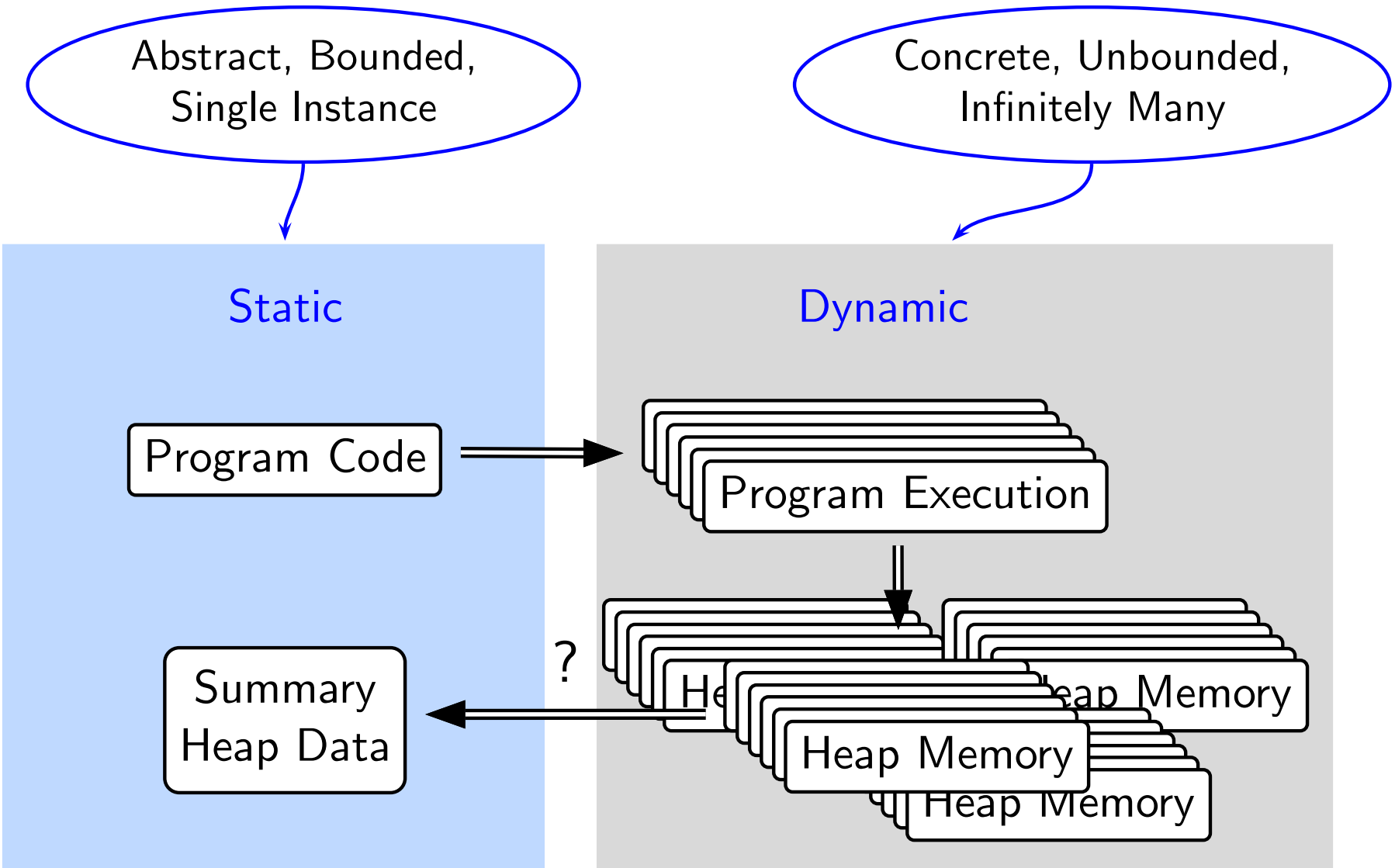
BTW, What is Static Analysis of Heap?



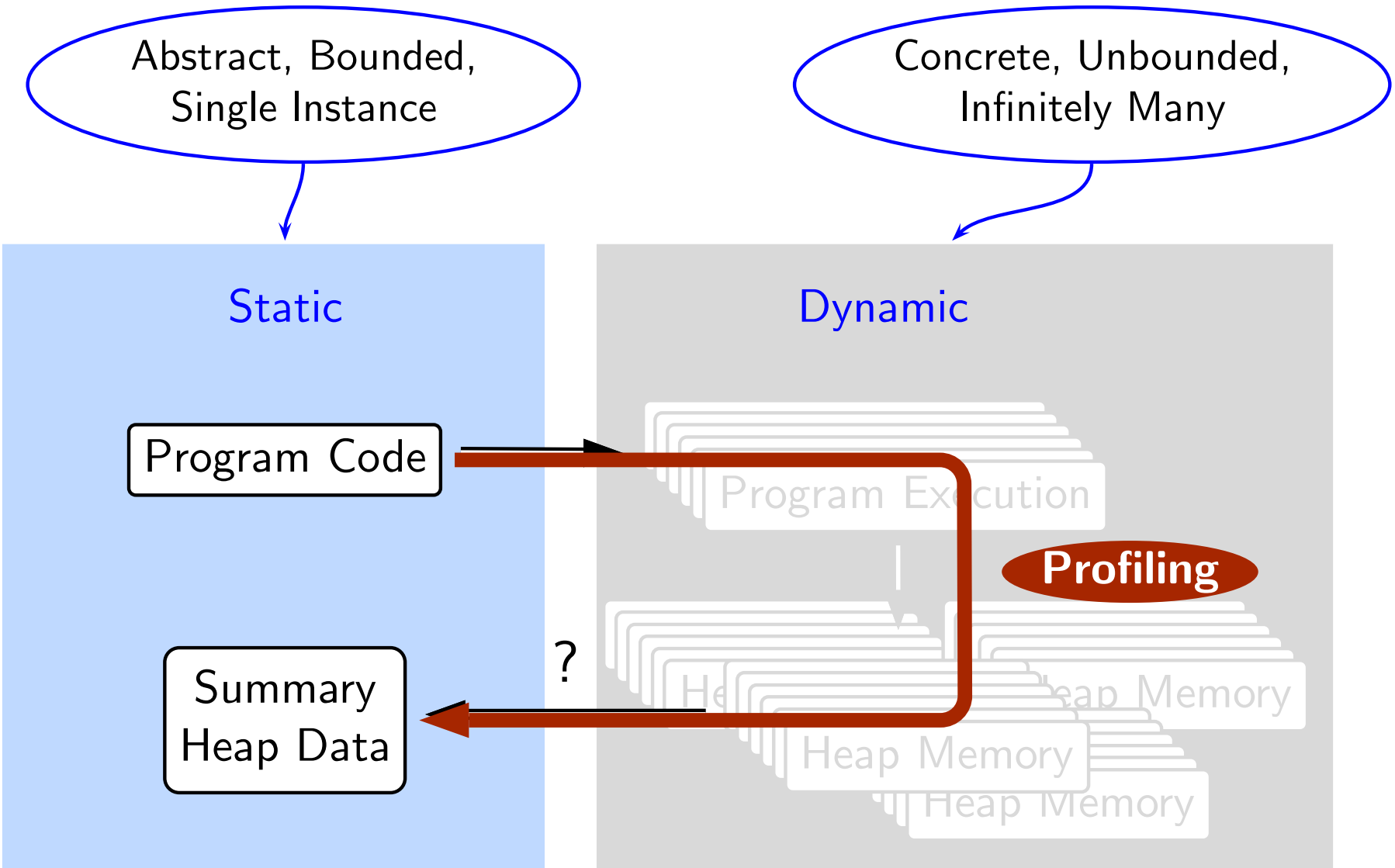
BTW, What is Static Analysis of Heap?



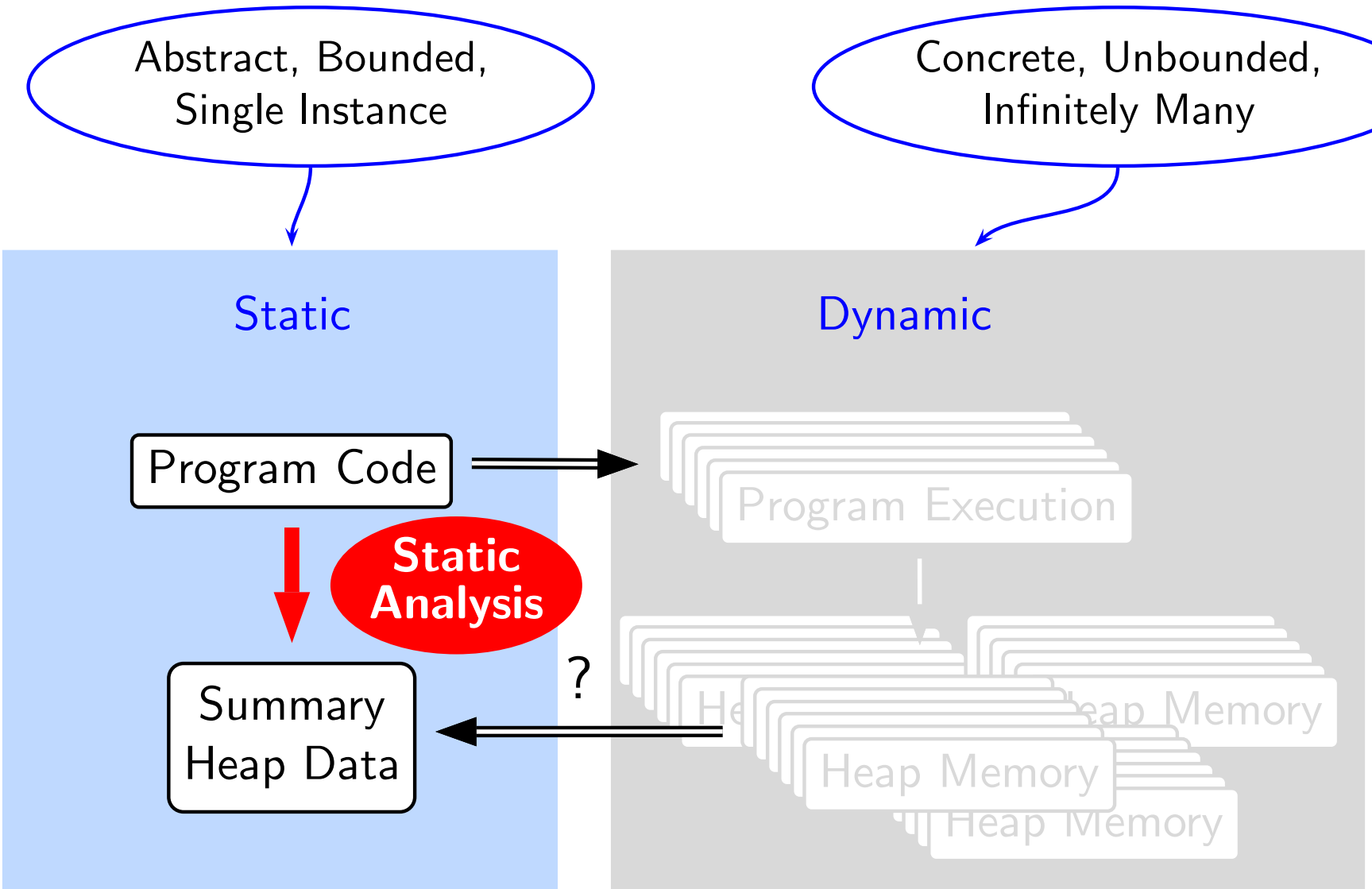
BTW, What is Static Analysis of Heap?



BTW, What is Static Analysis of Heap?



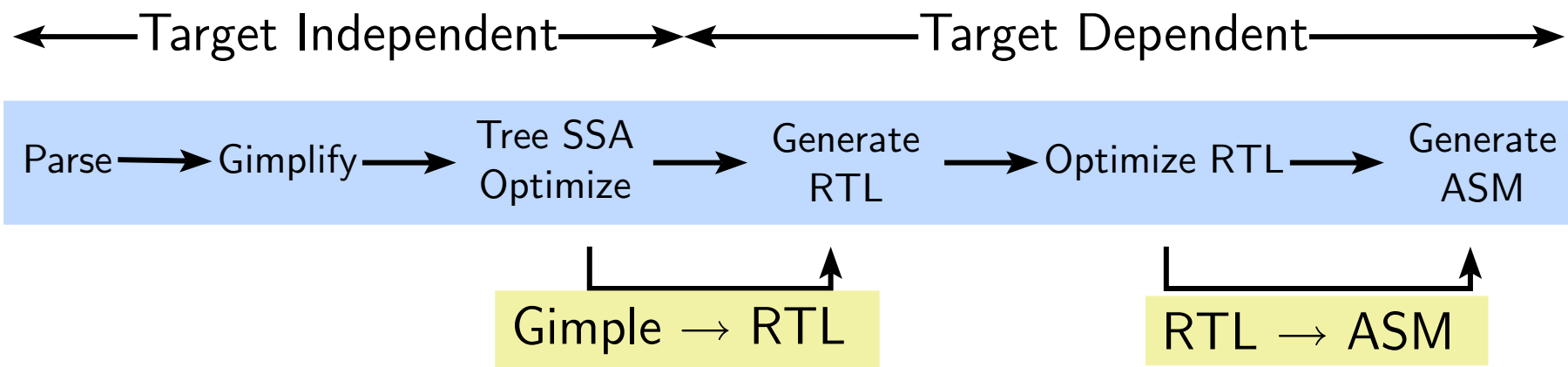
BTW, What is Static Analysis of Heap?



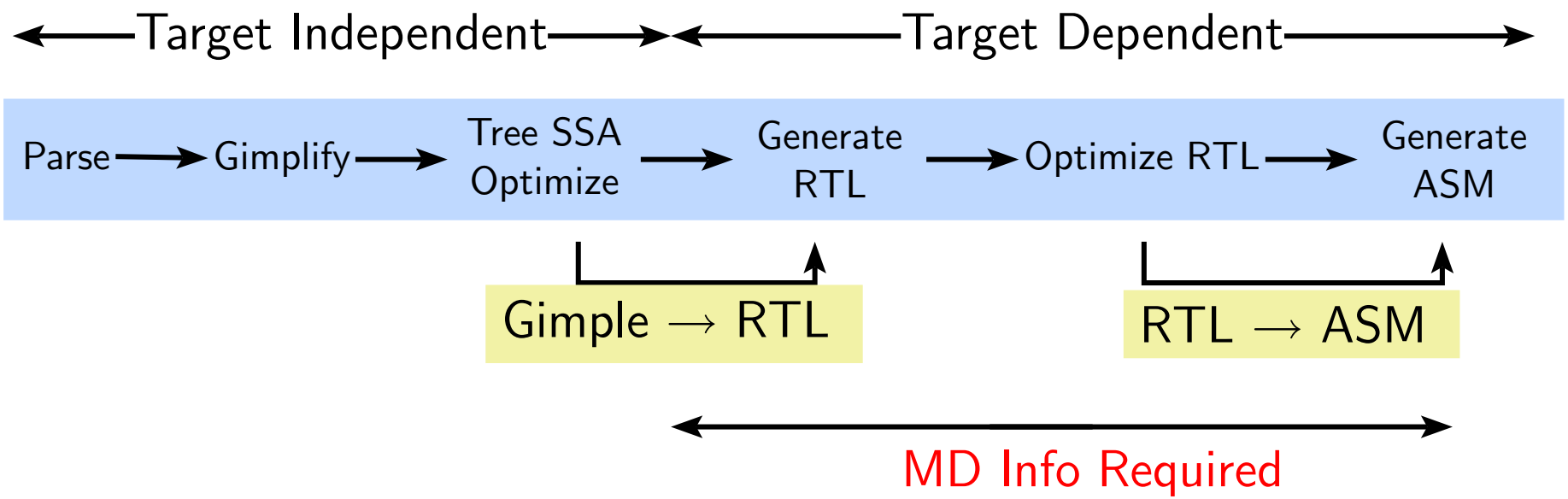
Part 3

Understanding GCC Machine Descriptions

Role of Machine Descriptions in Translation



Role of Machine Descriptions in Translation



A Target Instruction in Machine Descriptions

```
(define_insn
    "movsi"
     (set
       (match_operand 0 "register_operand" "r")
       (match_operand 1 "const_int_operand" "k")
       )
    "" /*  C boolean expression, if required */
    "li %0, %1"
)
```


A Target Instruction in Machine Descriptions

Define instruction pattern

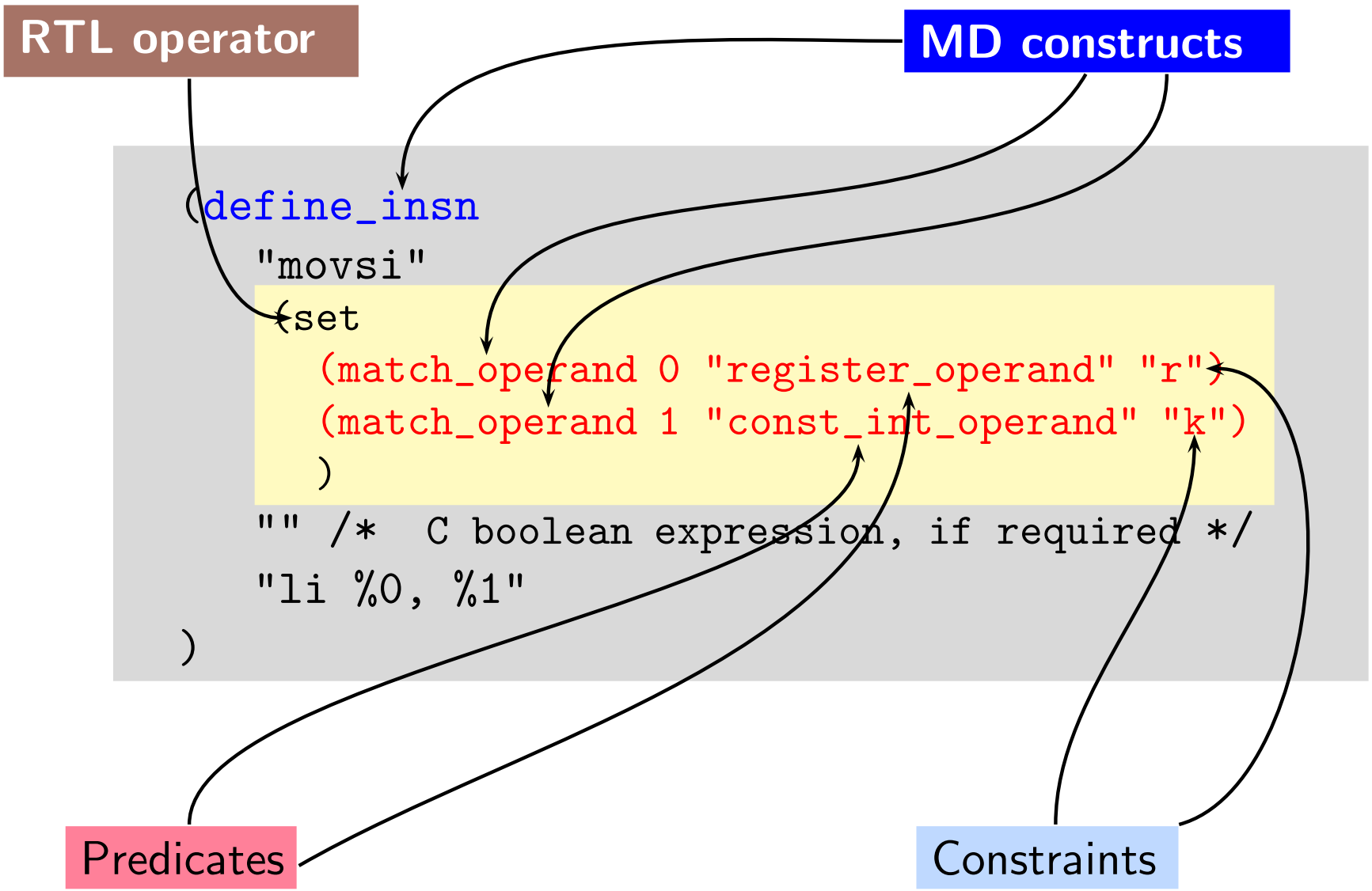
Standard Pattern Name

```
(define_insn
    "movsi"
    (set
      (match_operand 0 "register_operand" "r")
      (match_operand 1 "const_int_operand" "k")
      )
    "" /*  C boolean expression, if required */
    "li %0, %1"
)
```

RTL Expression (RTX):
Semantics of target instruction

target asm inst. =
Concrete syntax for RTX

A Target Instruction in Machine Descriptions



An Example of Translation

```
(define_insn
   "movsi"
    (set
      (match_operand 0 "register_operand" "r")
      (match_operand 1 "const_int_operand" "k")
      )
   "" /*  C boolean expression, if required */
   "li %0, %1"
)
```

An Example of Translation

```
(define_insn
   "movsi"
    (set
      (match_operand 0 "register_operand" "r")
      (match_operand 1 "const_int_operand" "k")
      )
   "" /*  C boolean expression, if required */
   "li %0, %1"
)
```

Development

```
D.1283 = 10;  ⇒  (set
                    (reg:SI 58 [D.1283])
                    (const_int 10:  [0xa])
                 )  ⇒  li $t0, 10
```

Use

The Essence of Retargetability

When are the machine descriptions read?

The Essence of Retargetability

When are the machine descriptions read?

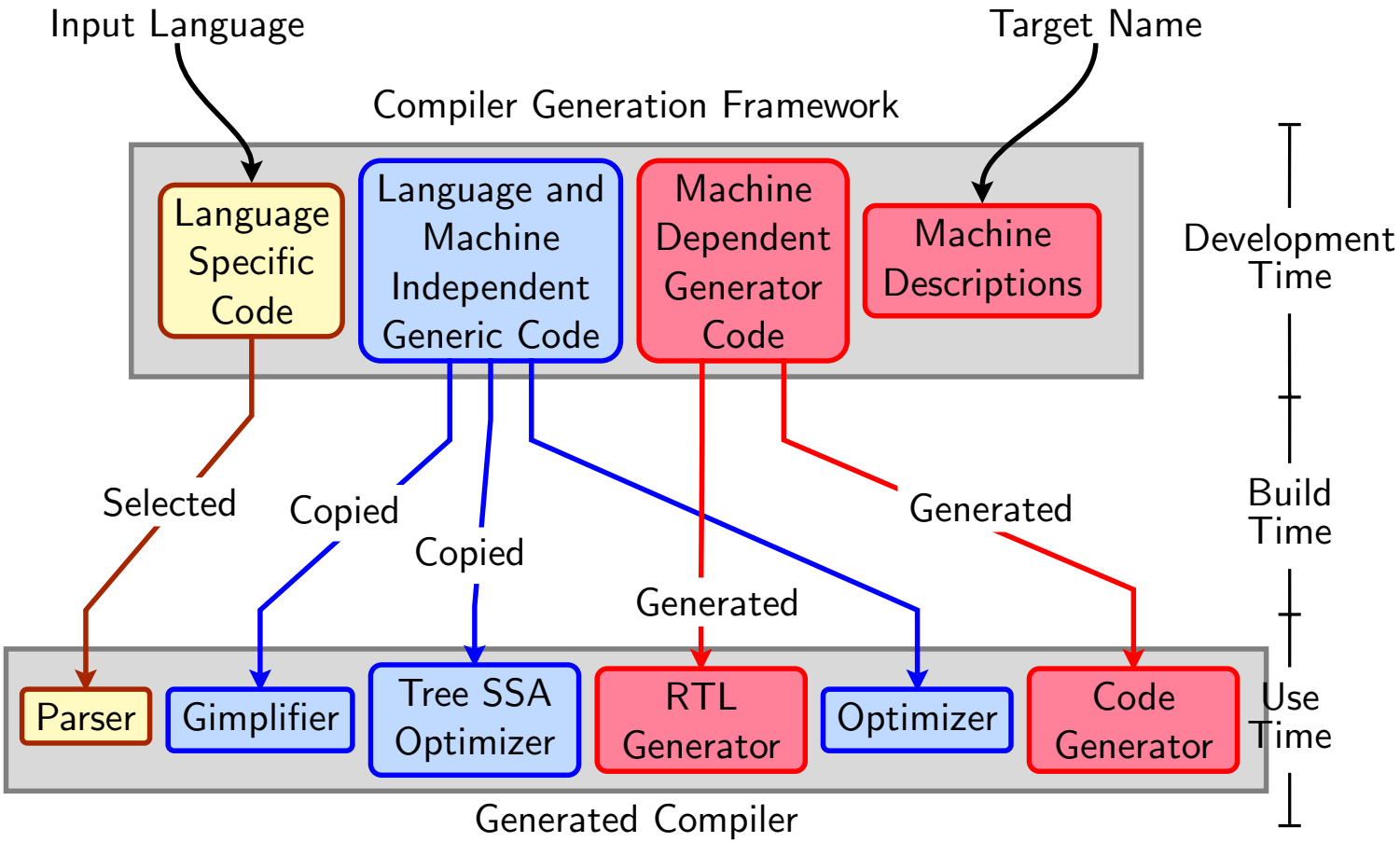
- During the build process

The Essence of Retargetability

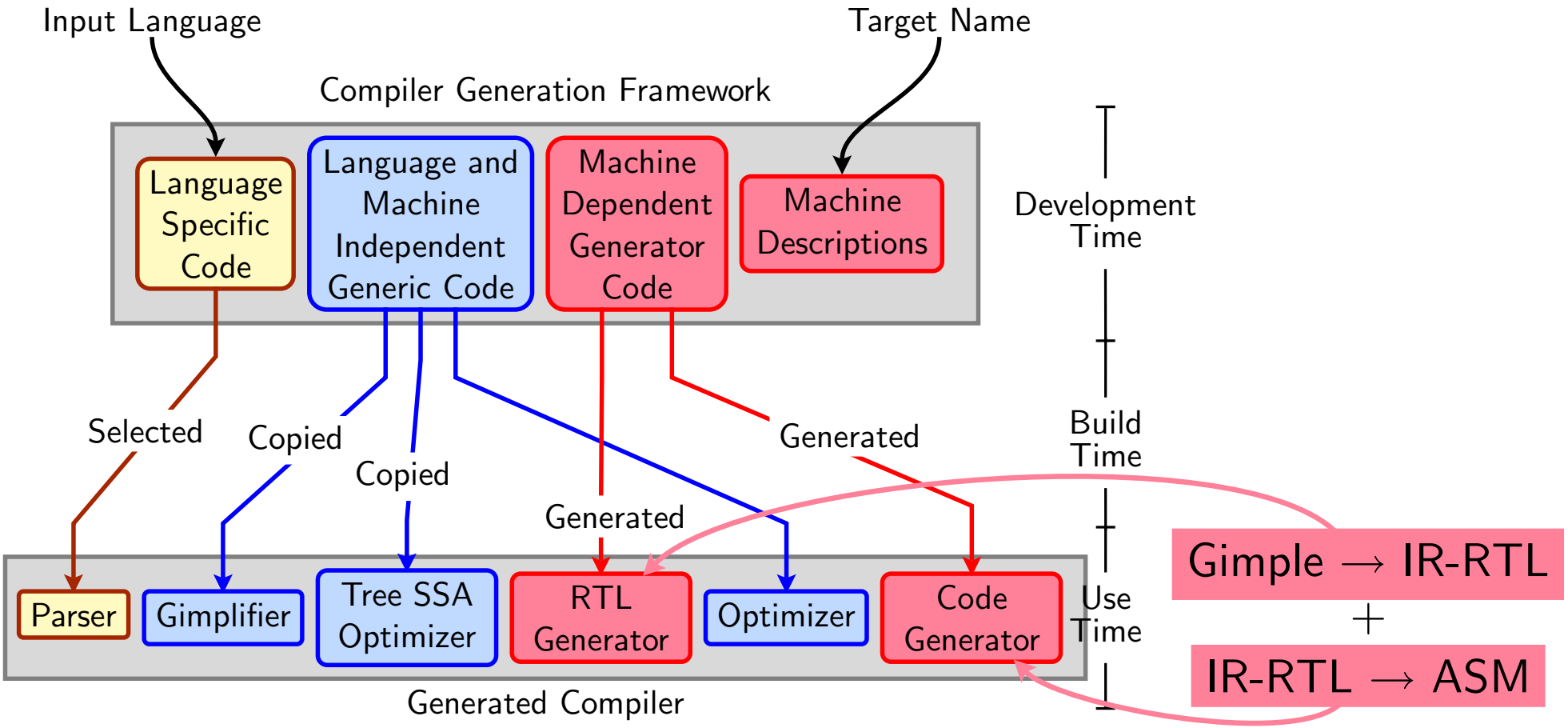
When are the machine descriptions read?

- During the build process
- When a program is compiled by gcc the information gleaned from machine descriptions is consulted

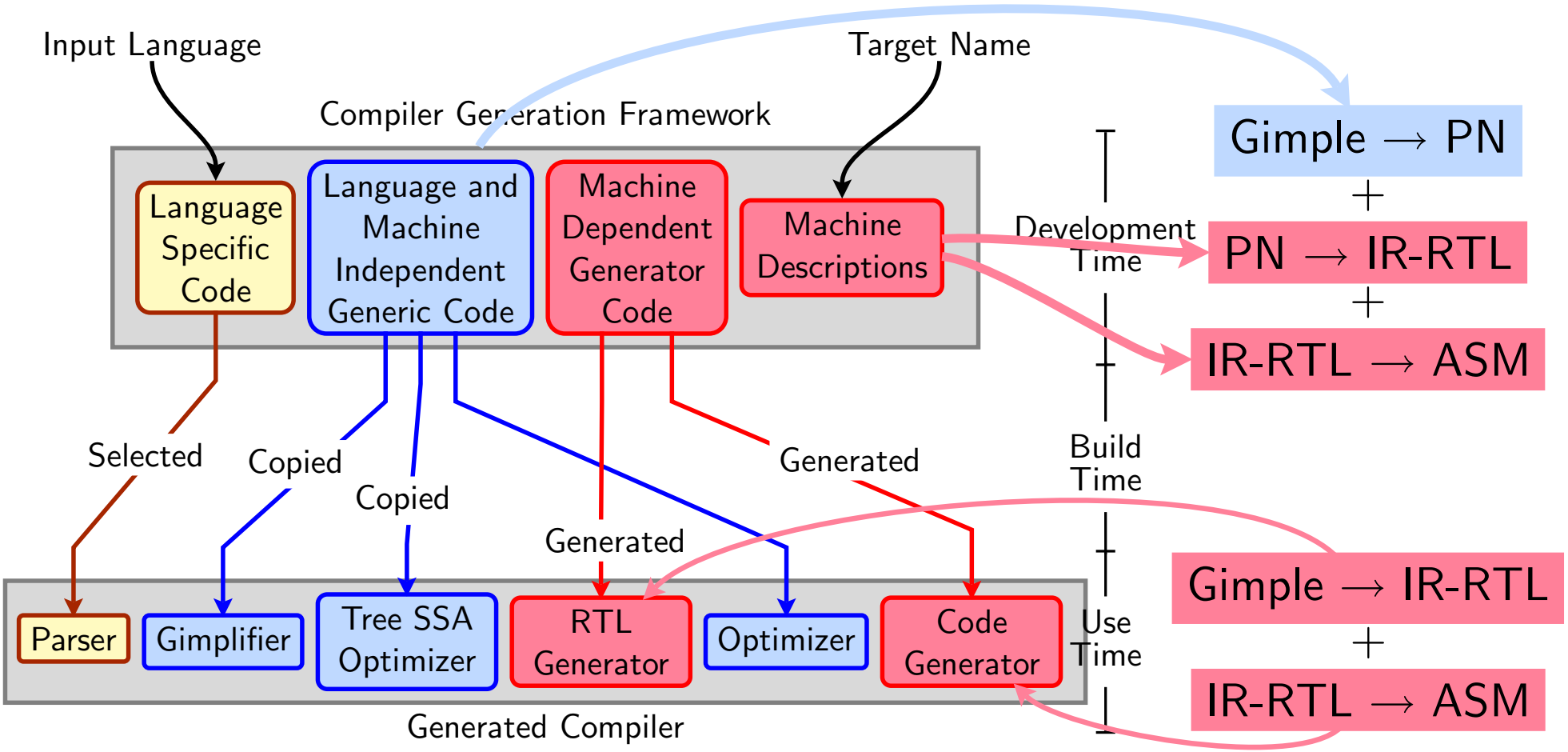
Retargetability Mechanism of GCC



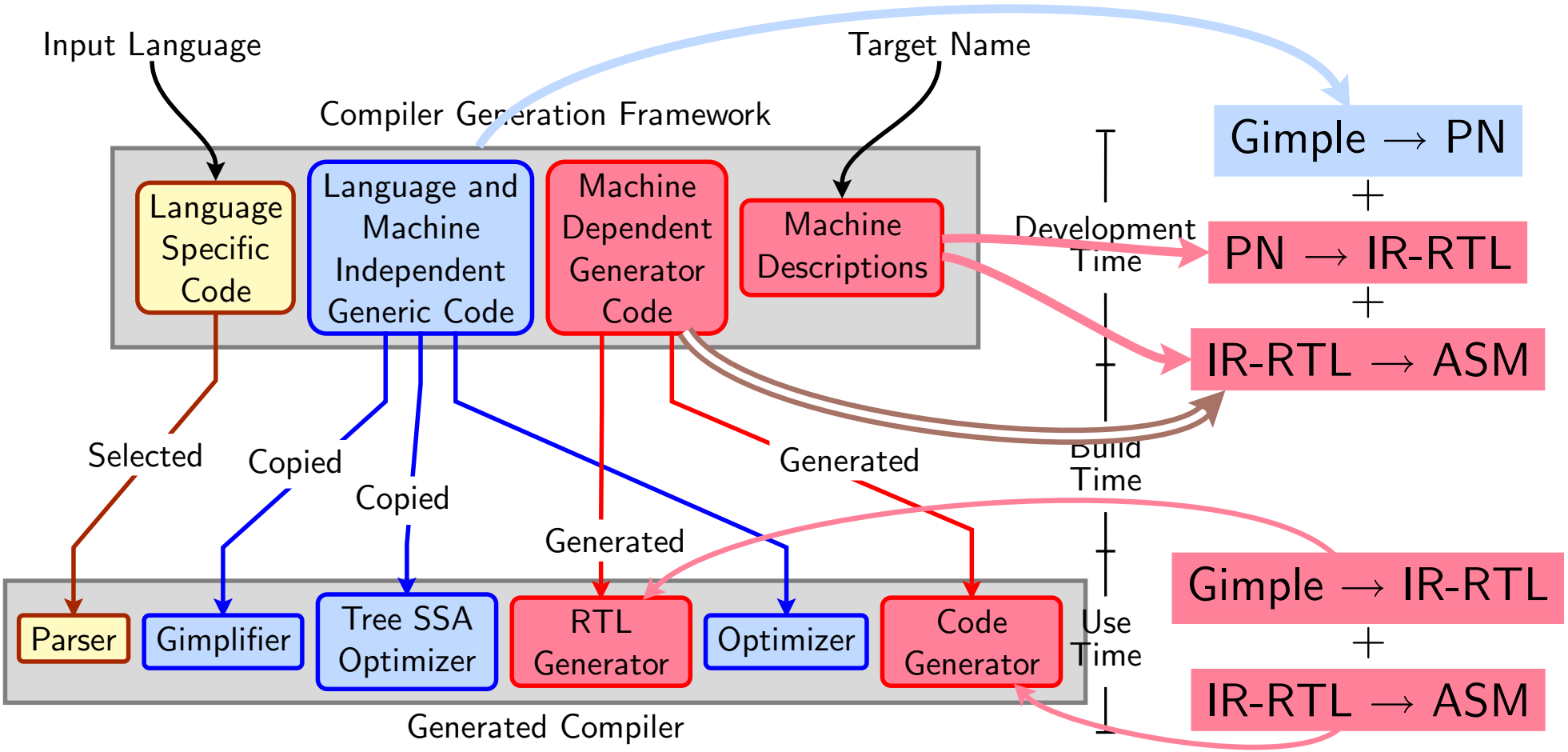
Retargetability Mechanism of GCC



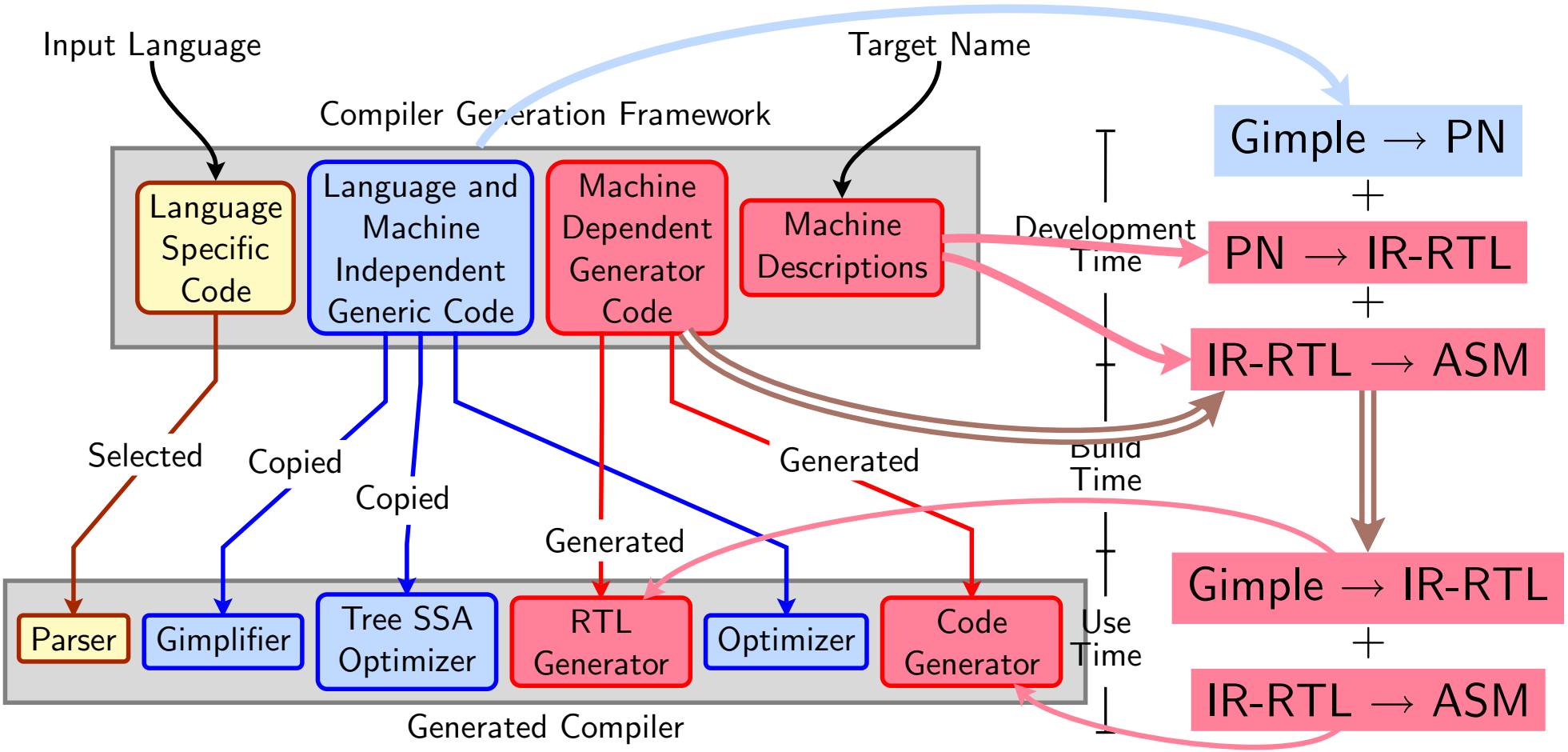
Retargetability Mechanism of GCC



Retargetability Mechanism of GCC



Retargetability Mechanism of GCC



Some Other RTL Constructs

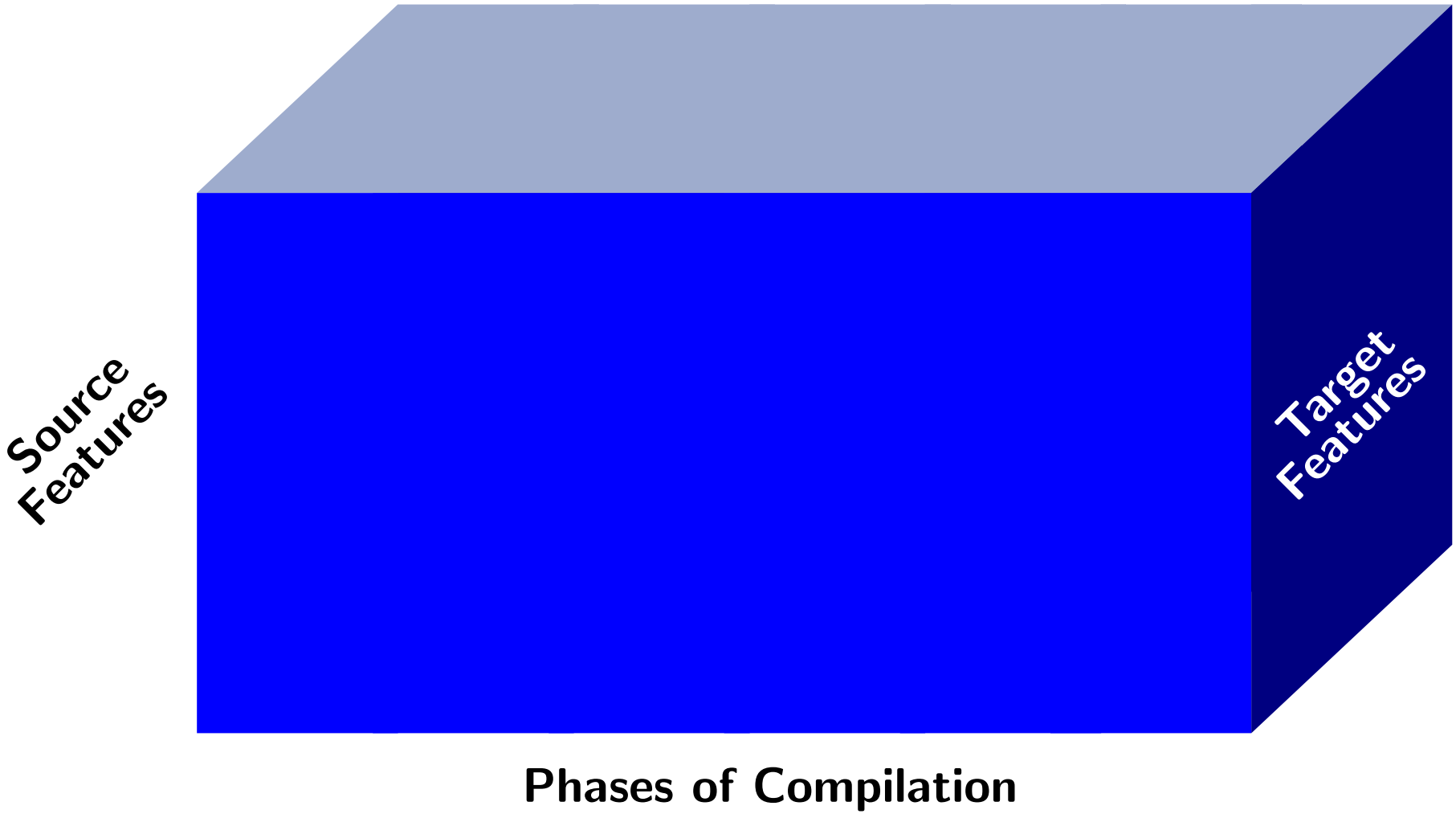
- `define_expan`: Generate possibly multiple RTL statements by using associated C code
- `define_attr`: Define attributes and specify their values
- `define_split`: Split complex insn into simpler ones
  e.g. for better use of delay slots
- `define_insn_and_split`: A combination of `define_insn` and `define_split`
  Used when the split pattern matches and insn exactly.
- `define_peephole2`: Peephole optimization over insns that substitutes insns. Run after register allocation, and before scheduling.
- `define_constants`: Use literal constants in rest of the MD.

The Size of Machine Descriptions

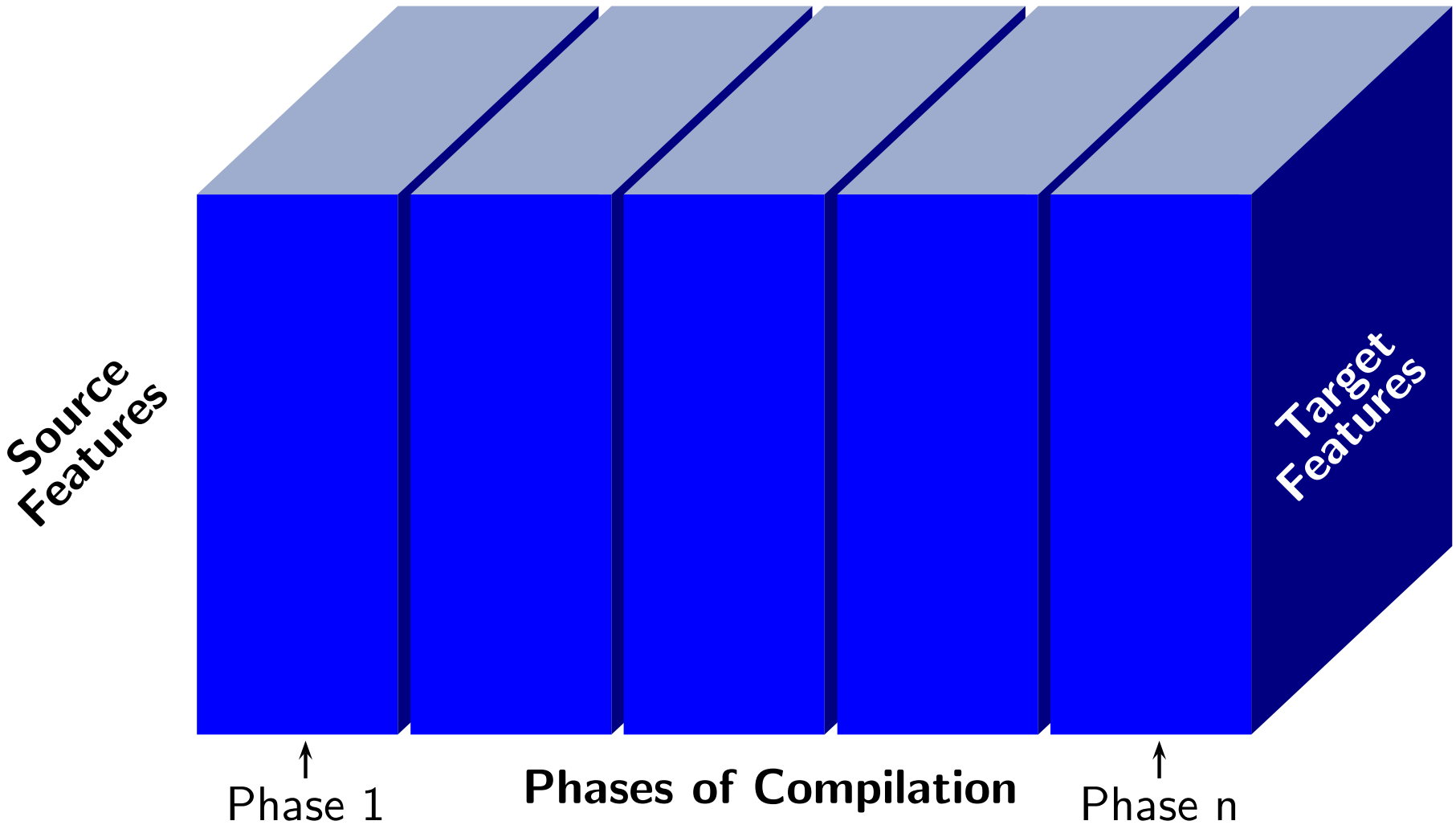
Size in terms of line counts

Files	i386	mips
*.md	35766	12930
*.c	28643	12572
*.h	15694	5105

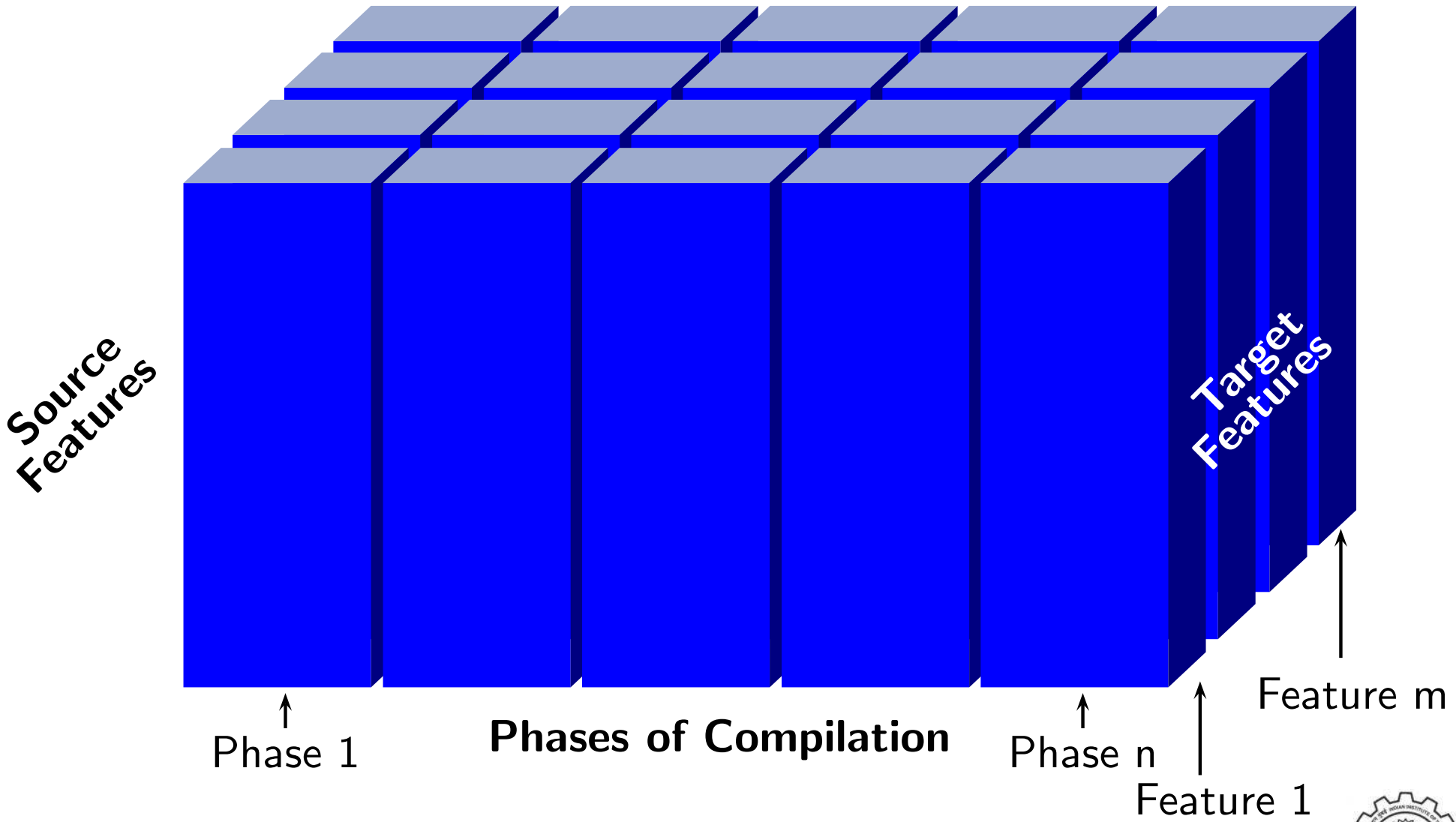
In Search of Modularity in Retargetable Compilation



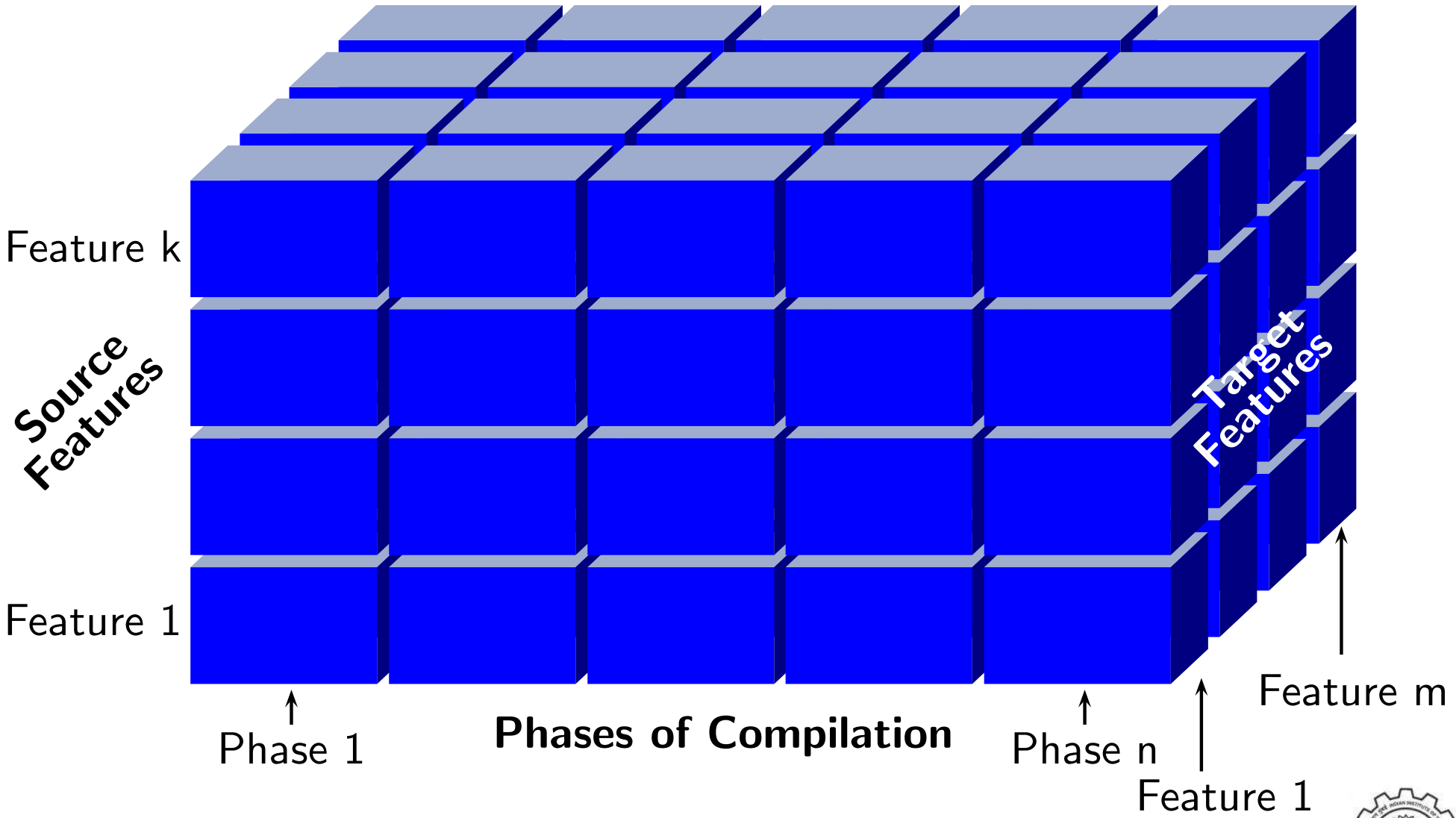
In Search of Modularity in Retargetable Compilation



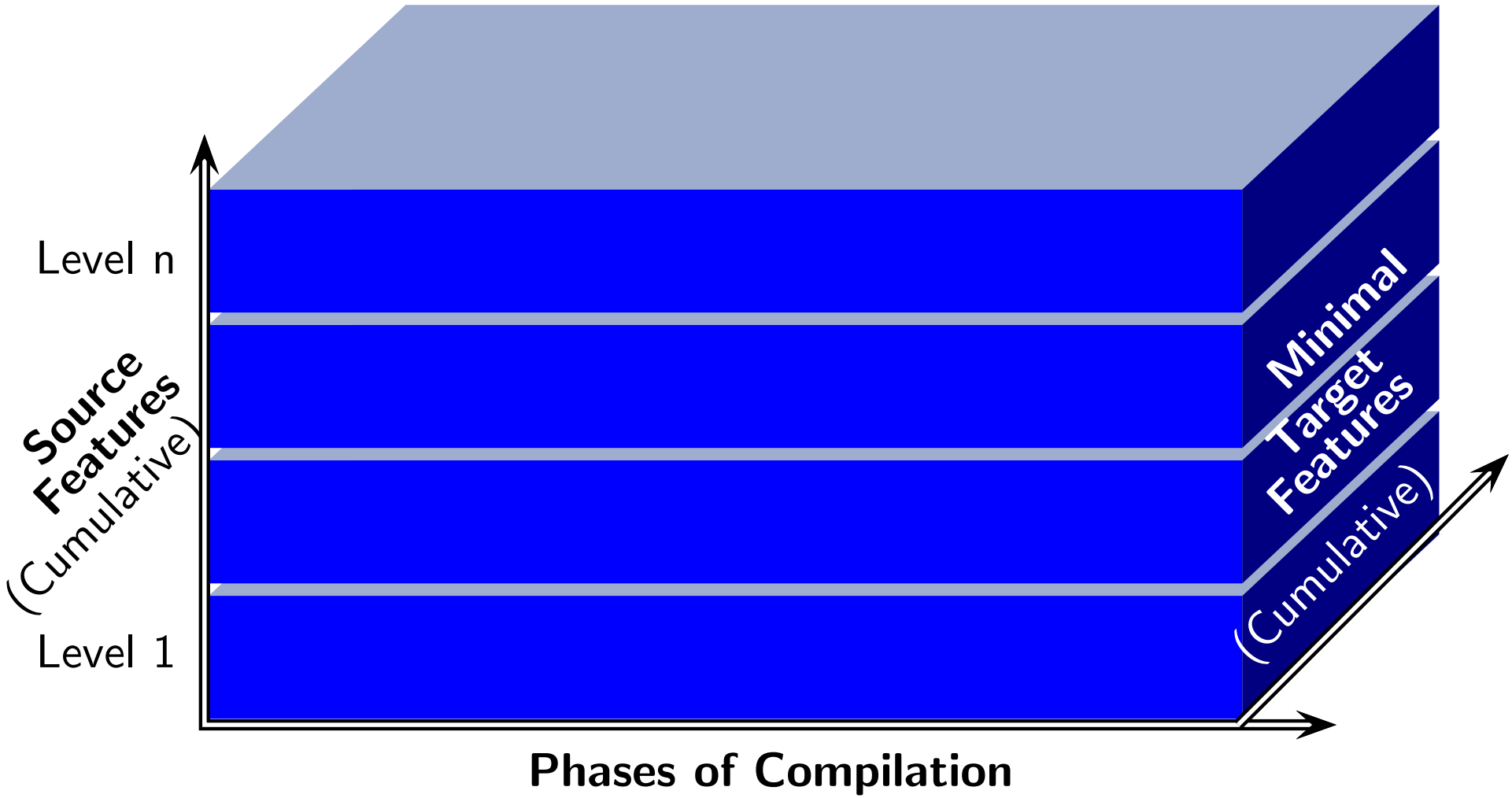
In Search of Modularity in Retargetable Compilation



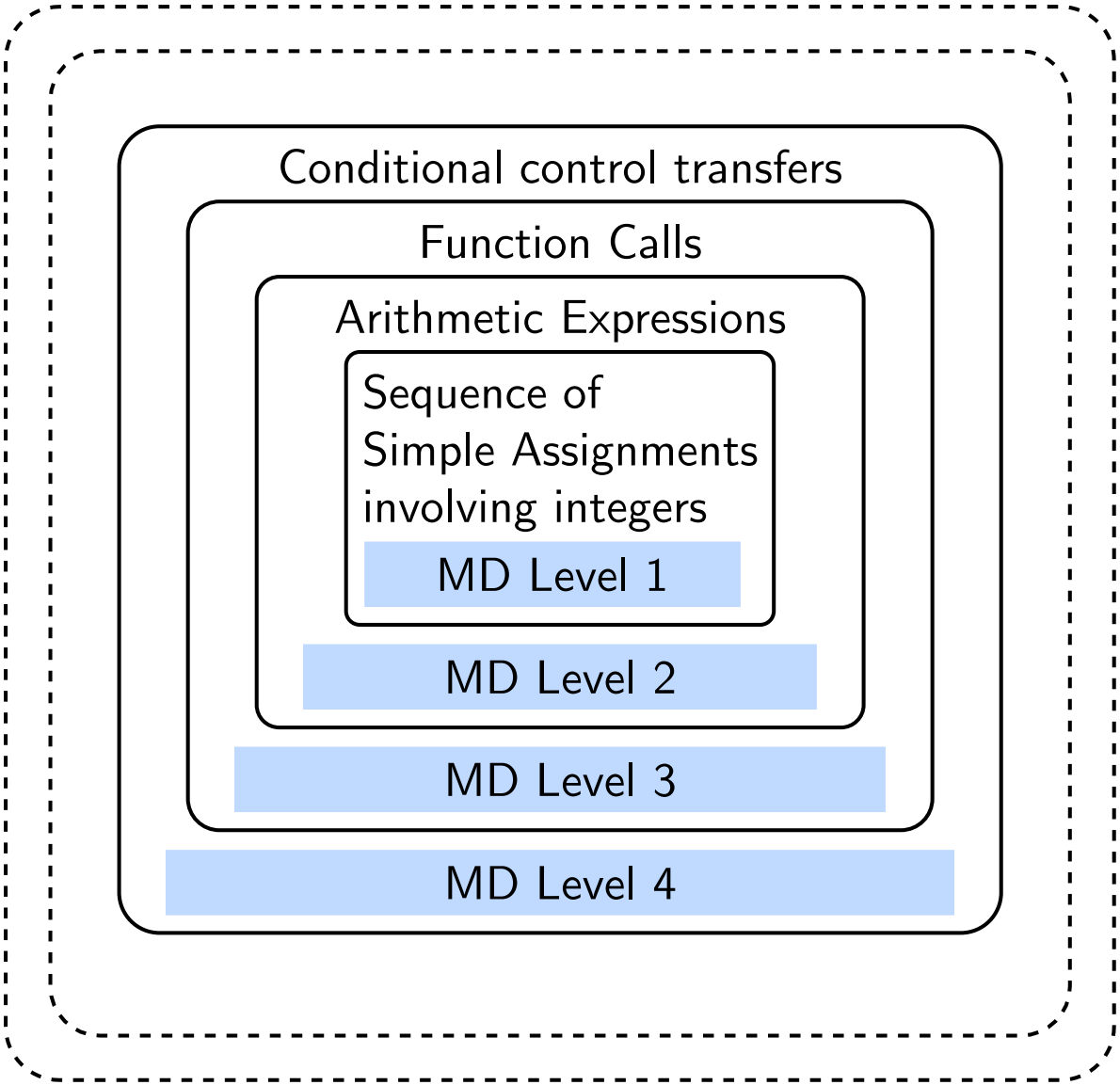
In Search of Modularity in Retargetable Compilation



In Search of Modularity in Retargetable Compilation



Incremental Construction of Machine Descriptions [GREPS 2007]



Incremental Construction of Machine Descriptions

- Define different levels of source language
- Identify the minimal set of features in the target required to support each level
- Identify the minimal information required in the machine description to support each level
  - Successful compilation of any program, and
  - correct execution of the generated assembly program
- Interesting observations
  - It is the increment in the source language which results in understandable increments in machine descriptions rather than the increment in the target architecture
  - If the levels are identified properly, the increments in machine descriptions are monotonic

Incremental Construction of Machine Descriptions

- Consequence

  The usual ramp up period into GCC machine descriptions has been brought down from over six months to a couple of weeks

- Current Status
  - Has been extended to GCC 4.3.1
  - Has been extended to floating point operations

- Further Work
  - Extending it to optimizations specified in machine descriptions
  - Adapting it to possible simplifications in machine descriptions

Part 4

Improving Instruction Selection and Machine Descriptions

Improving Machine Descriptions and Instruction Selection

The Problems:

- Instruction selection algorithms are quite adhoc
- The specification mechanism for Machine descriptions is quite adhoc
- Adhoc design decisions

Improving Machine Descriptions and Instruction Selection

The Problems:

- Instruction selection algorithms are quite adhoc
  - Full tree maching instead of tree tiling
- The specification mechanism for Machine descriptions is quite adhoc
- Adhoc design decisions

Improving Machine Descriptions and Instruction Selection

The Problems:

- Instruction selection algorithms are quite adhoc
  - Full tree maching instead of tree tiling
- The specification mechanism for Machine descriptions is quite adhoc
  - Only syntax borrowed from LISP, neither semantics not spirit!
  - Non-composable rules
- Adhoc design decisions

Improving Machine Descriptions and Instruction Selection

The Problems:

- Instruction selection algorithms are quite adhoc
  - Full tree maching instead of tree tiling
- The specification mechanism for Machine descriptions is quite adhoc
  - Only syntax borrowed from LISP, neither semantics not spirit!
  - Non-composable rules
- Adhoc design decisions
  - Honouring operand constraints delayed to global register allocation
    When required during gimple to RTL translation, a lot of C code is required
  - Choice of insertion of NOPs

Design Flaws in Machine Descriptions

Multiple patterns with same structure

- Repetition of almost similar RTL expressions across multiple `define_insn` an `define_expand patterns`
  - Only Modes, Predicates, Constraints, Boolean Condition, or RTL Expression may differ
  - One RTL expression may appears as a sub-expression of some other RTL expression
- Repetition of C code along with RTL expressions in these patterns.

Consequence of Design Flaws in Machine Descriptions

- The machine descriptions are too verbose, detailed, repetitive and require a lot of C code
- 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

Step 1: Avoiding Verbosity in Machine Description

- New contructs to facilitate more concise machine descriptions
  - `define_rtltemplate`
    Introduces non-terminals for common RTL expressions instead of rewriting them in each `define_insn` or `define_expand` pattern
  - `define_code`
    Introduces non-terminals for C/Assembly code instead of rewriting them in each `define_insn` or `define_expand` pattern
  - `define_pattern`
    Allows specification of multiple `define_insn` and `define_expand` sharing RTL template, assembly template, or C code

Step 1: Avoiding Verbosity in Machine Description

- New contructs to facilitate more concise machine descriptions
  - `define_rtltemplate`
    Introduces non-terminals for common RTL expressions instead of rewriting them in each `define_insn` or `define_expand` pattern
  - `define_code`
    Introduces non-terminals for C/Assembly code instead of rewriting them in each `define_insn` or `define_expand` pattern
  - `define_pattern`
    Allows specification of multiple `define_insn` and `define_expand` sharing RTL template, assembly template, or C code
- Generate exisiting machine descriptions from new descriptions

$\Rightarrow$ No change in GCC source

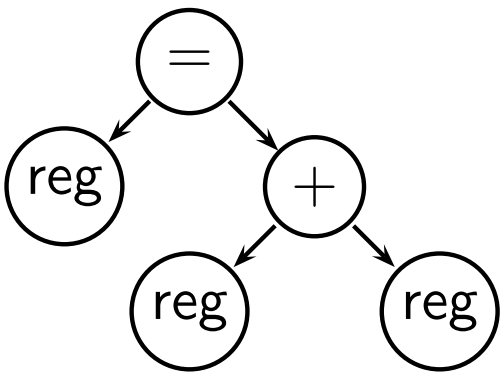
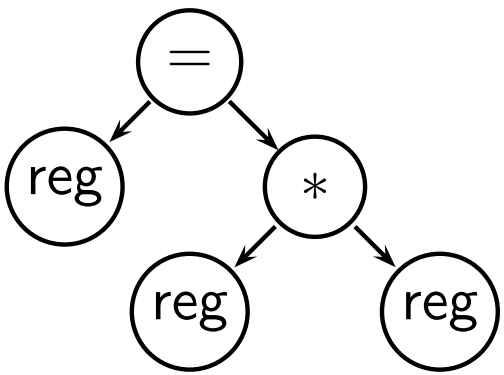
$\Rightarrow$ Incremental changes with gradual transition to new descriptions

Step 2: Improving Instruction Selection

- Since rules become composable, tree tiling based instruction selection algorithms can be used
  Currently rules are non-composable and GCC uses full tree maching algorithm

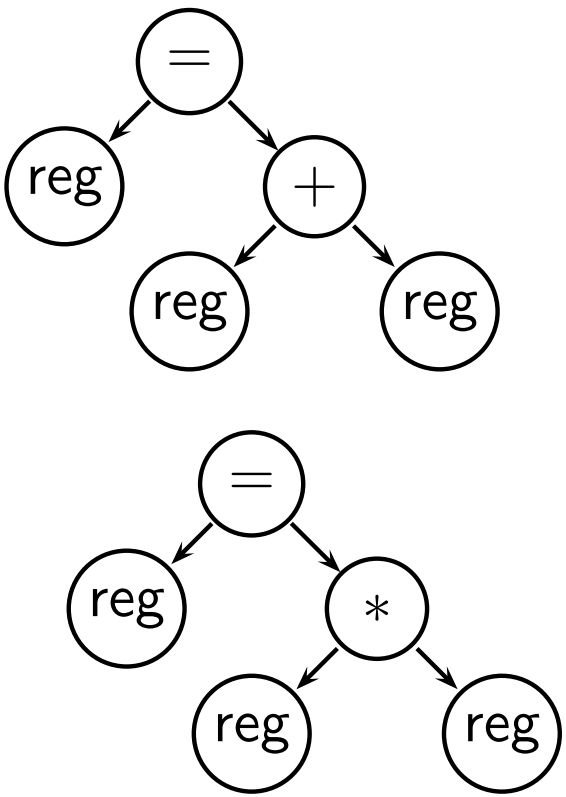
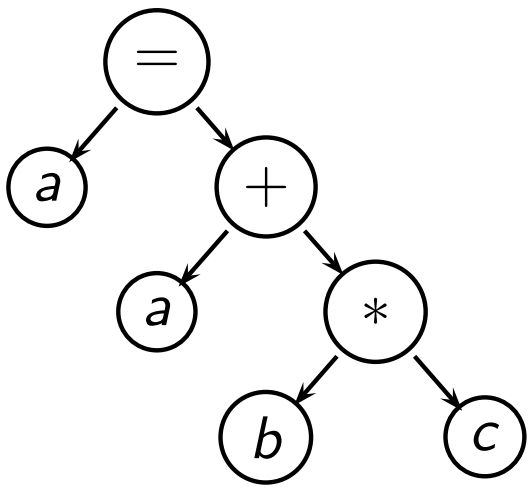
Full Tree Matching

Instructions are viewed as independent non-composable rules

Instructions	Subject Tree	Modified Trees
 		

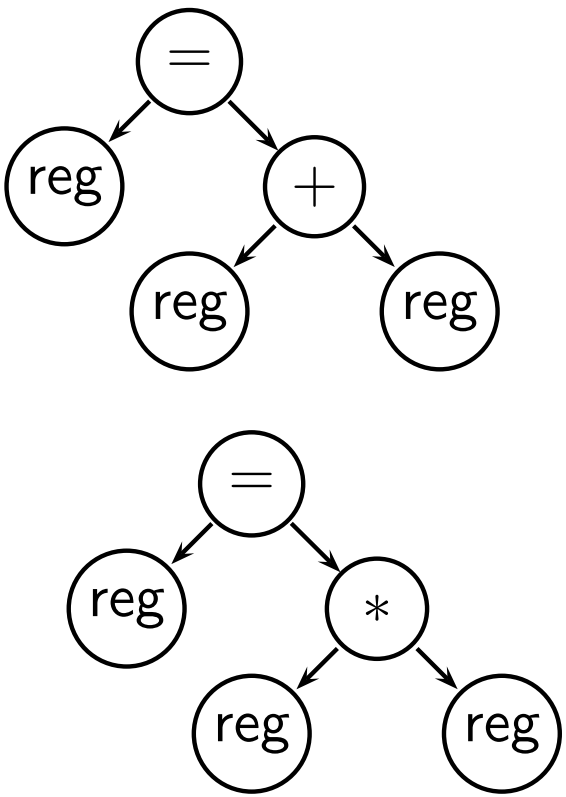
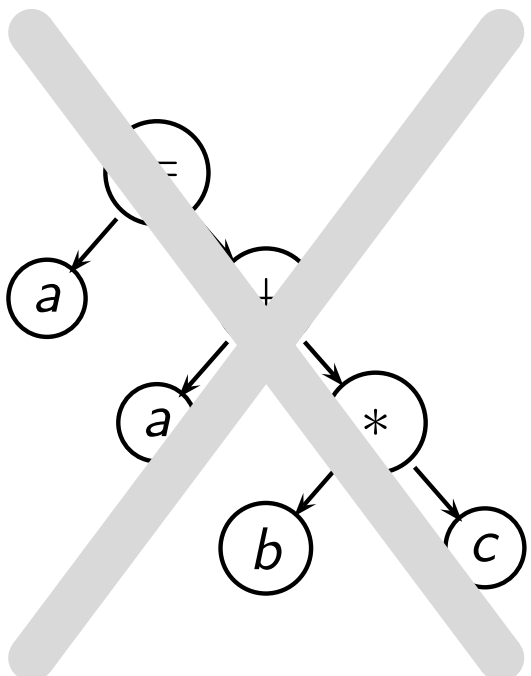
Full Tree Matching

Instructions are viewed as independent non-composable rules

Instructions	Subject Tree	Modified Trees
		

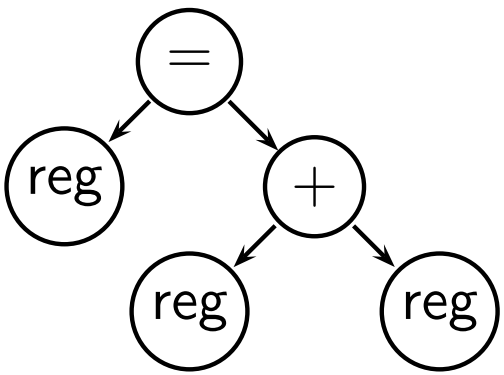
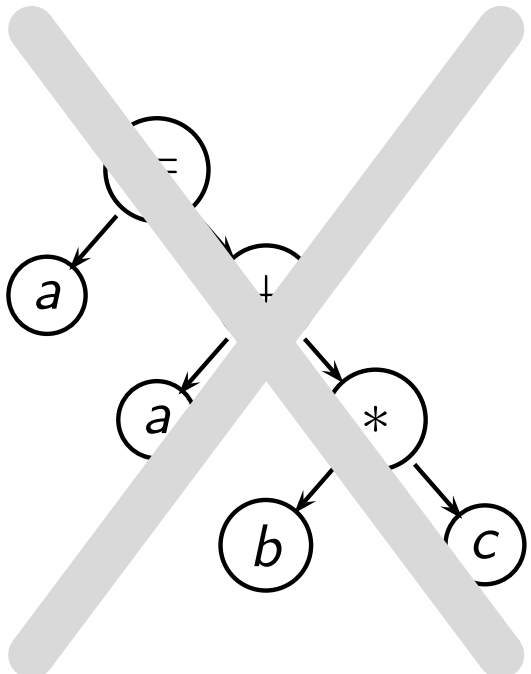
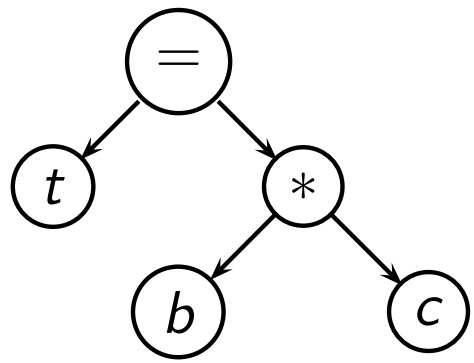
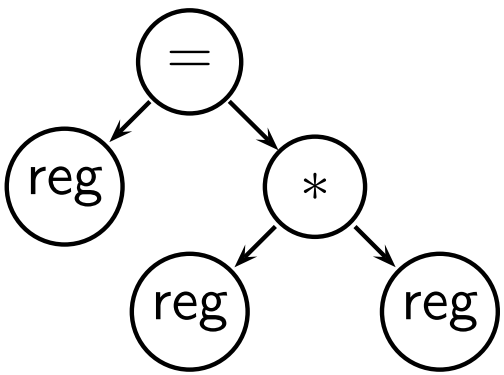
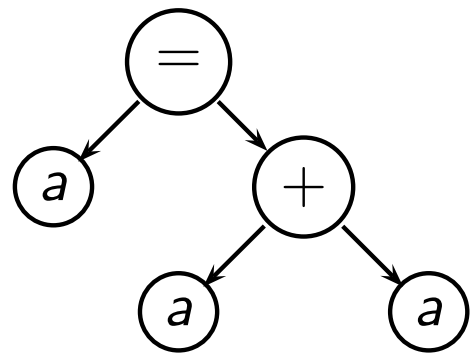
Full Tree Matching

Instructions are viewed as independent non-composable rules

Instructions	Subject Tree	Modified Trees
		

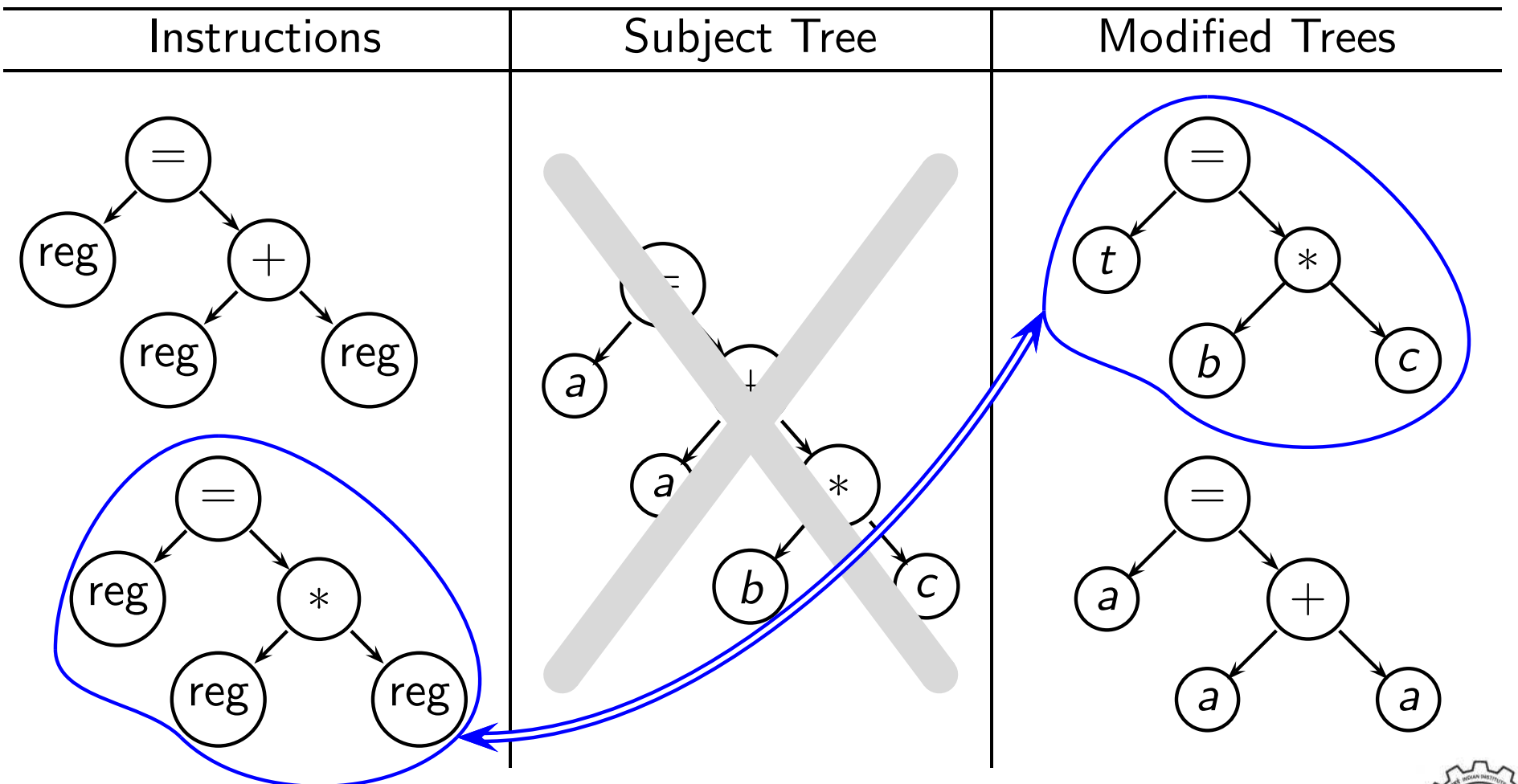
Full Tree Matching

Instructions are viewed as independent non-composable rules

Instructions	Subject Tree	Modified Trees
		
		

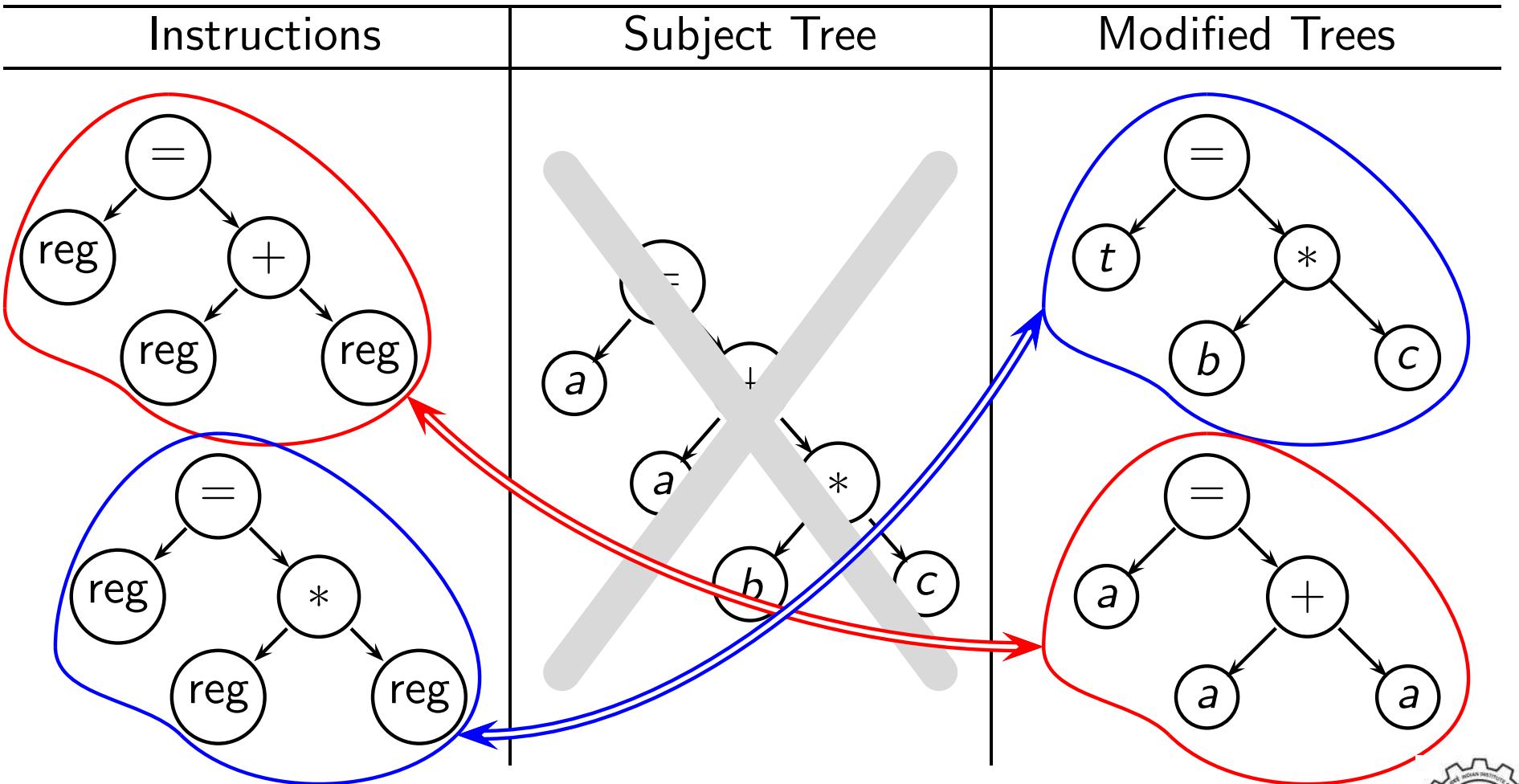
Full Tree Matching

Instructions are viewed as independent non-composable rules



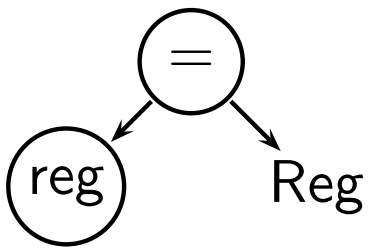
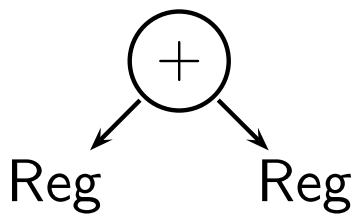
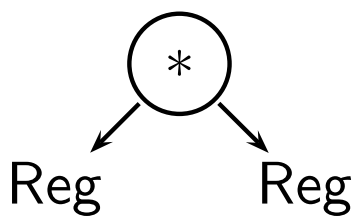
Full Tree Matching

Instructions are viewed as independent non-composable rules



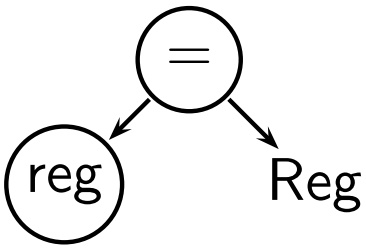
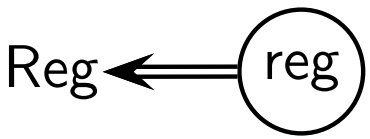
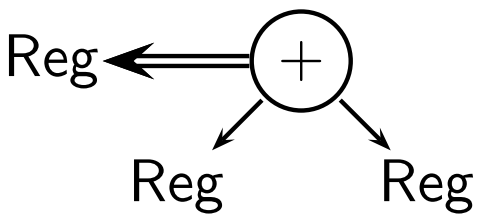
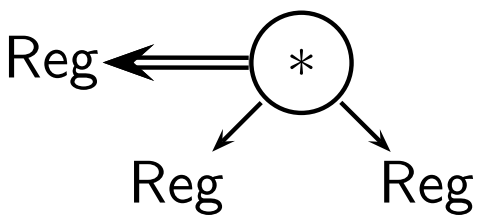
Tree Tiling

Instructions are viewed as composable rules

Instructions	Subject Tree
	
	
	
	

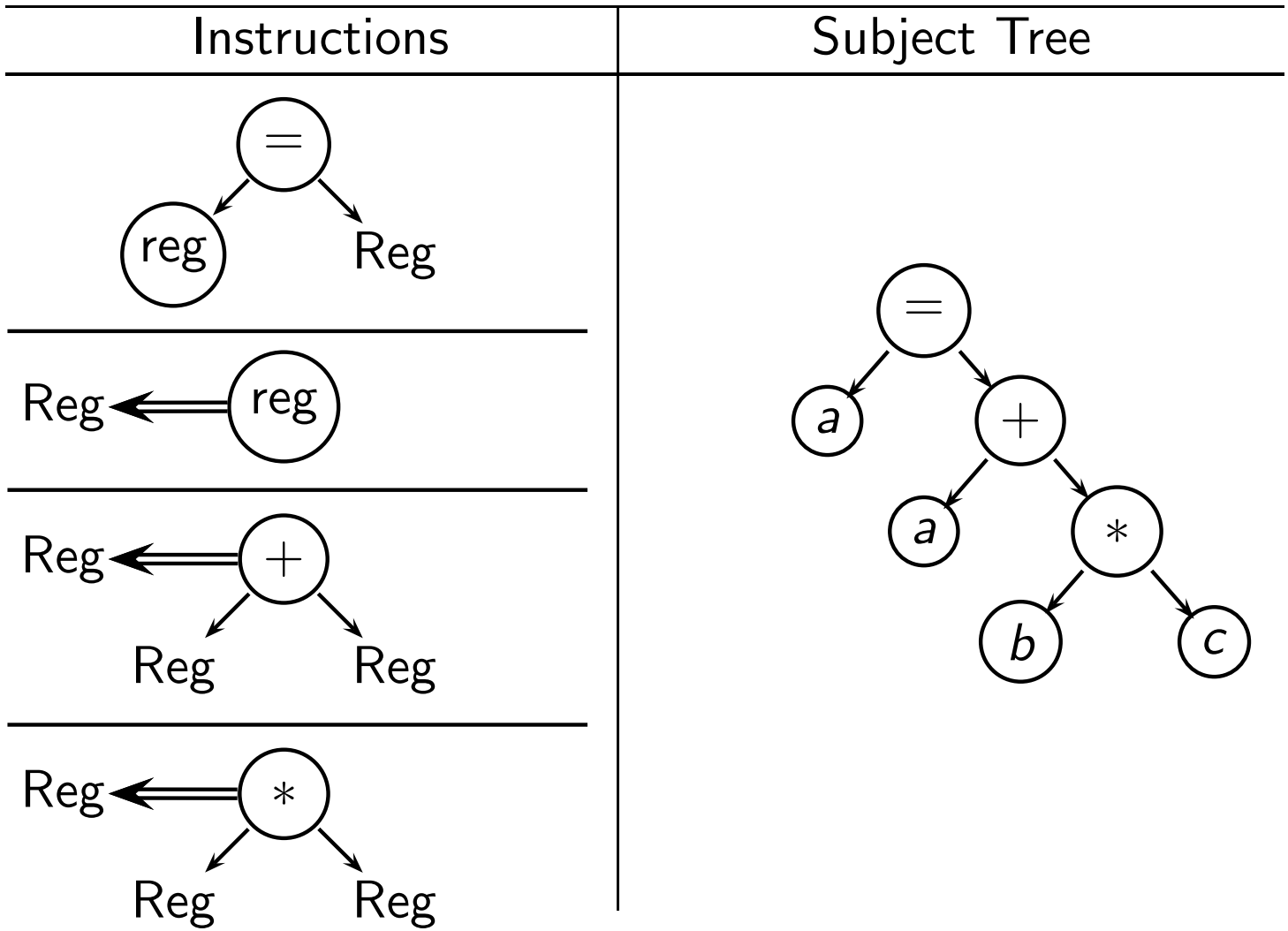
Tree Tiling

Instructions are viewed as composable rules

Instructions	Subject Tree
	
	
	
	

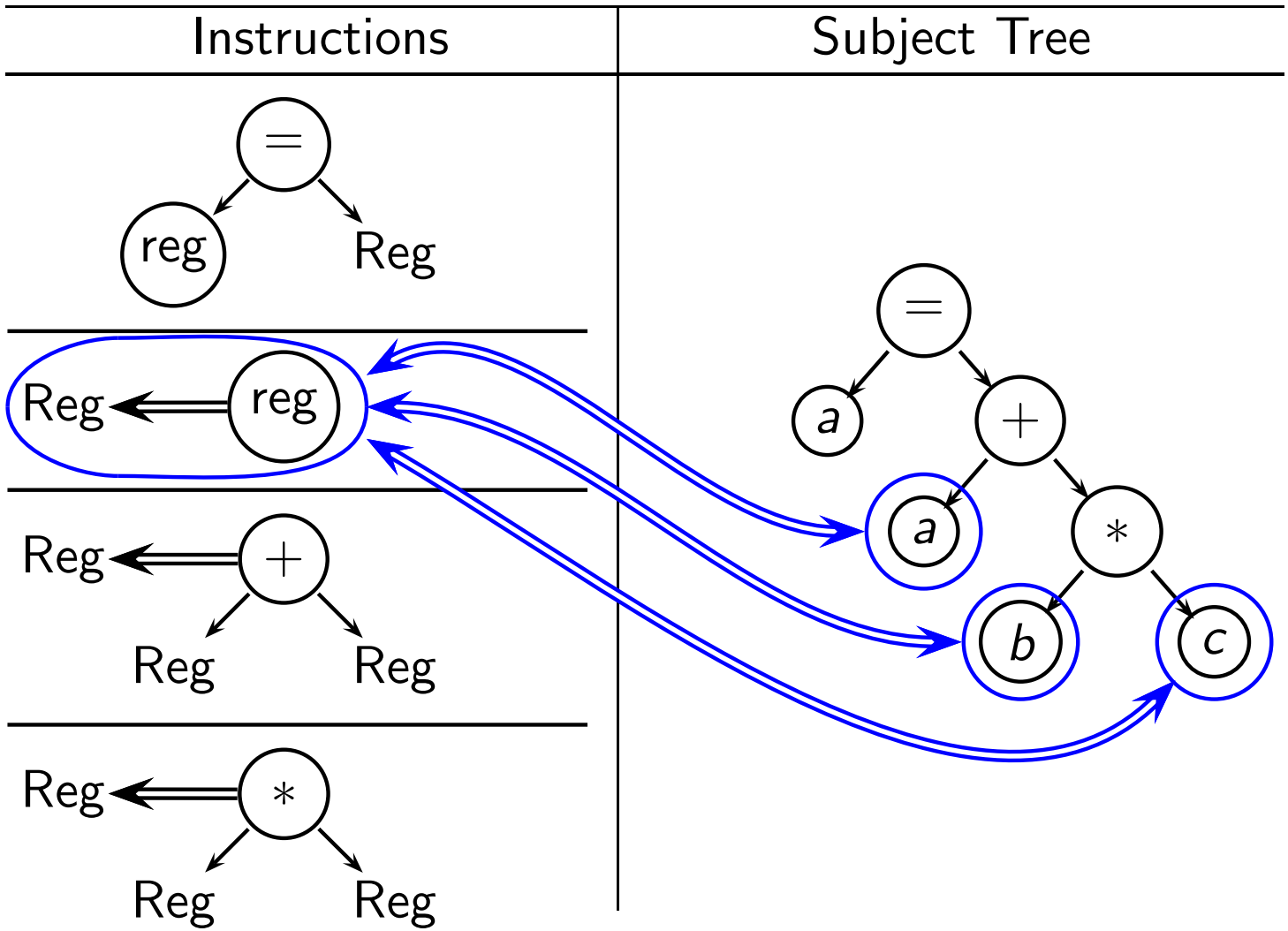
Tree Tiling

Instructions are viewed as composable rules



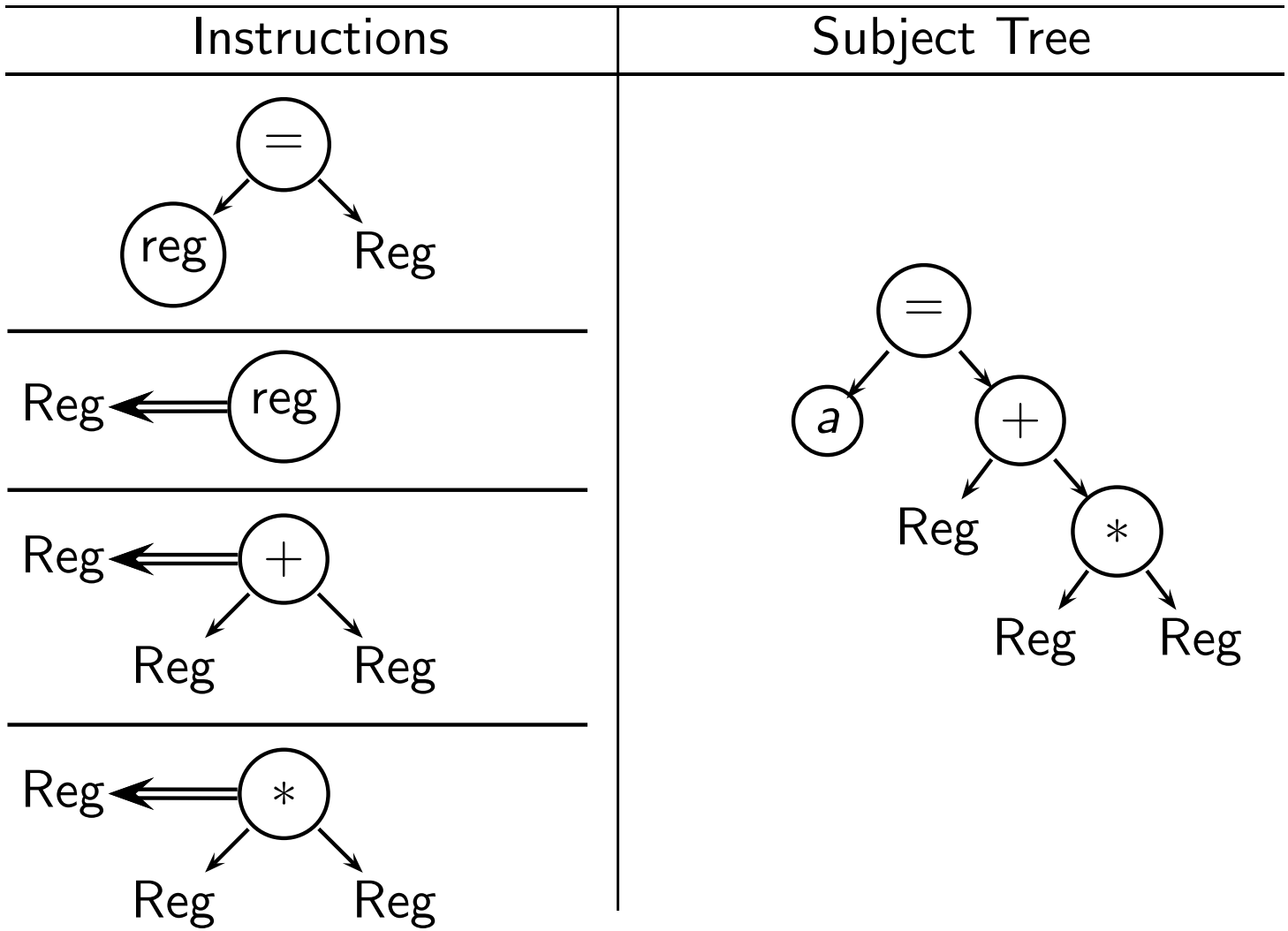
Tree Tiling

Instructions are viewed as composable rules



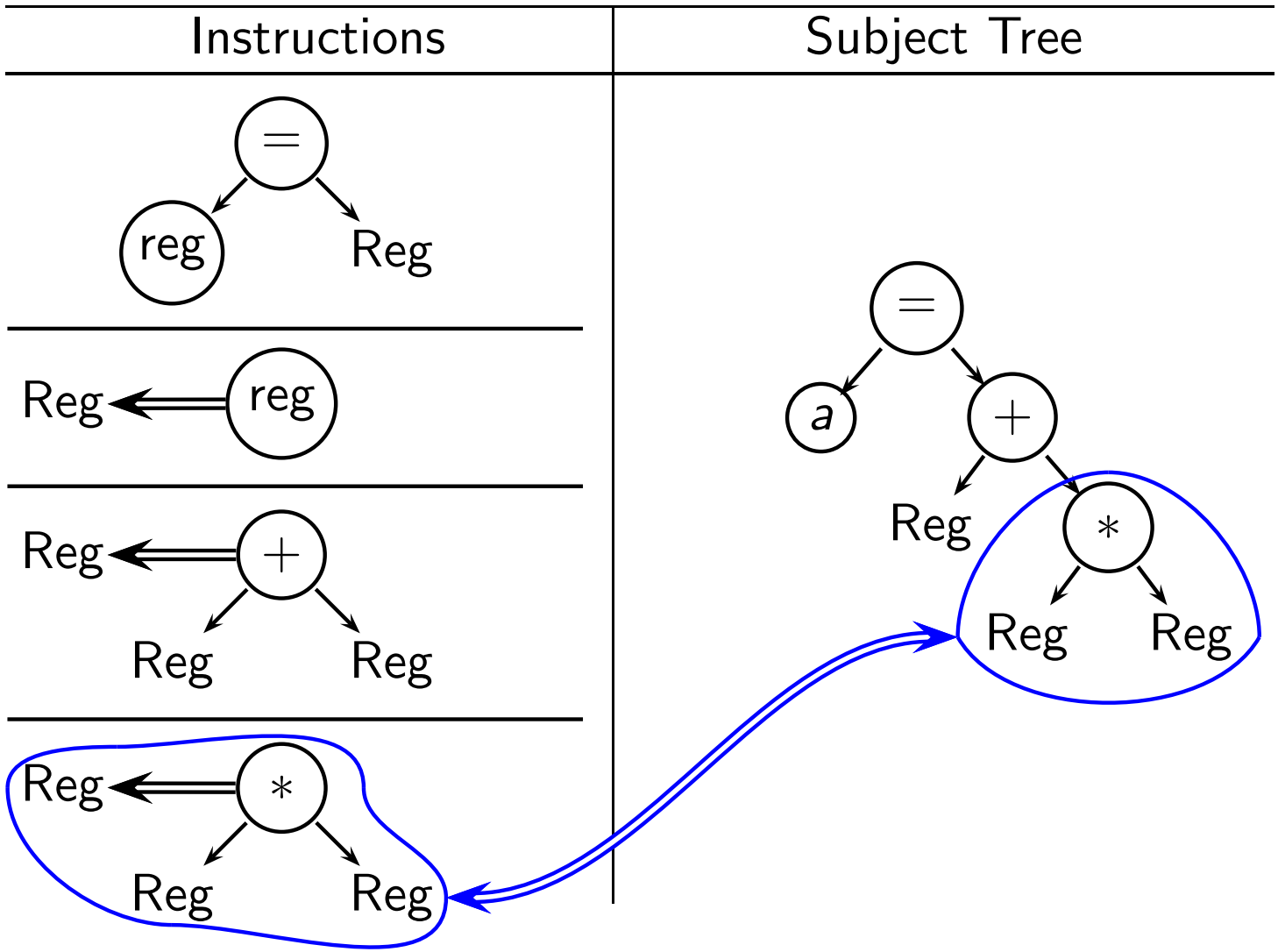
Tree Tiling

Instructions are viewed as composable rules



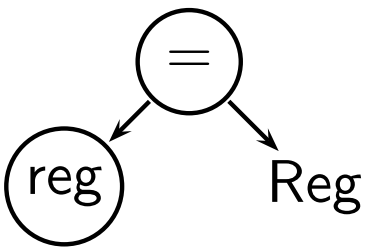
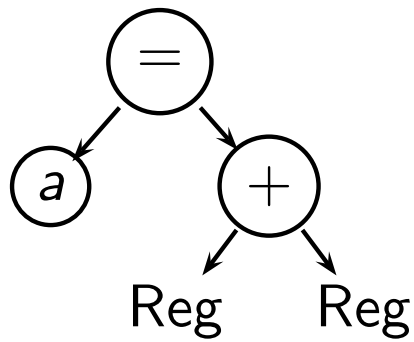
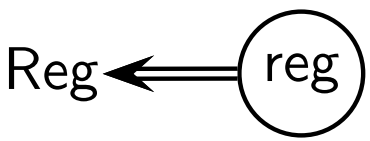
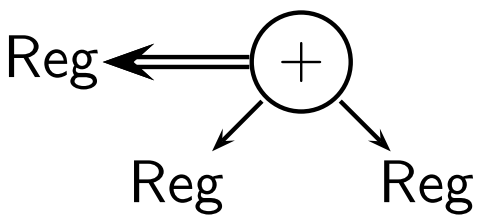
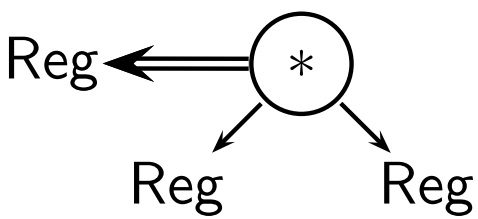
Tree Tiling

Instructions are viewed as composable rules



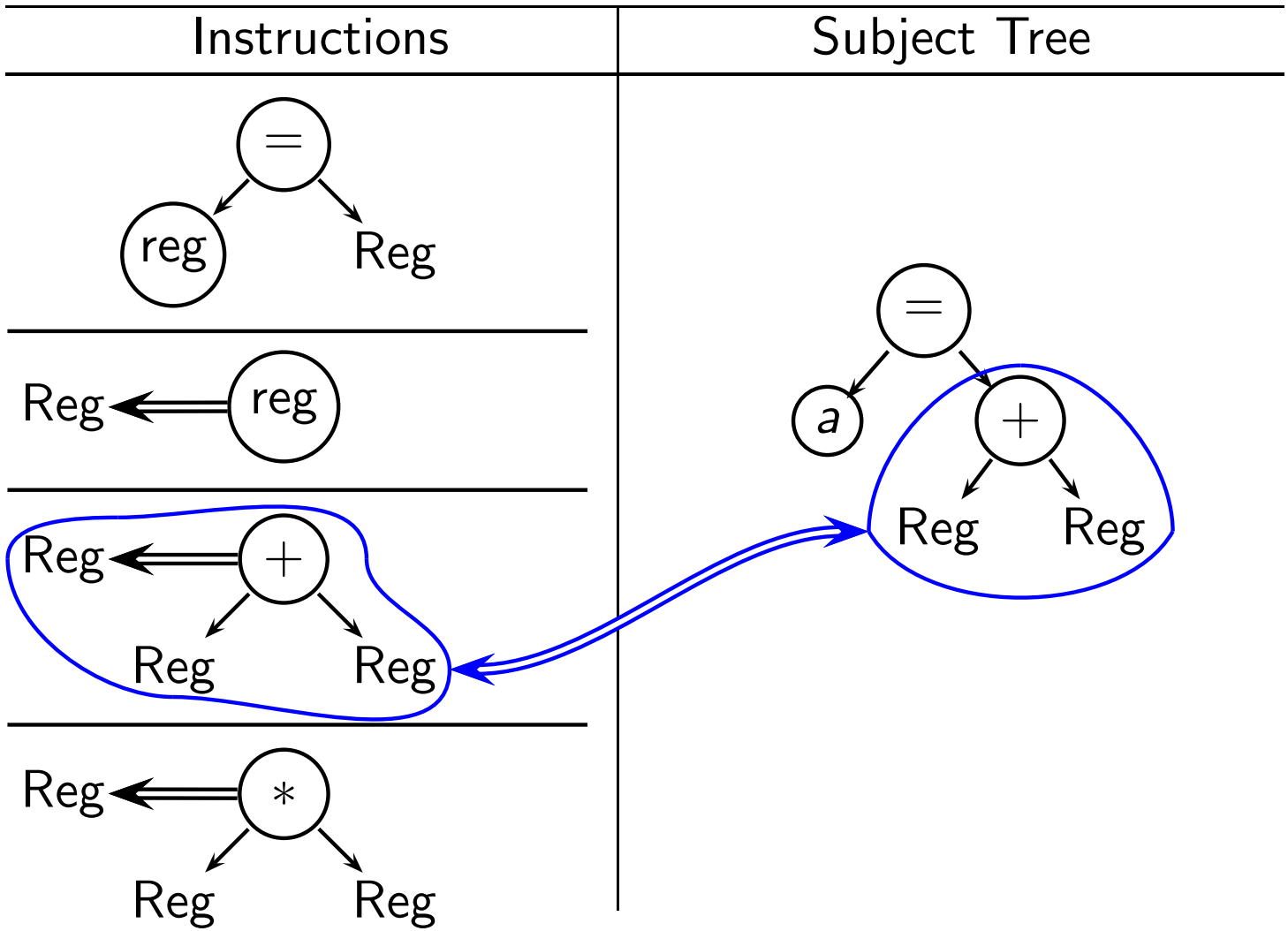
Tree Tiling

Instructions are viewed as composable rules

Instructions	Subject Tree
	
	
	
	

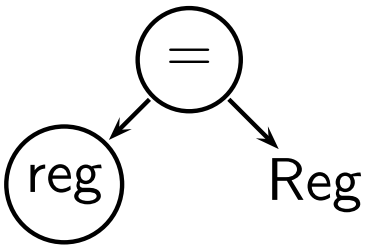
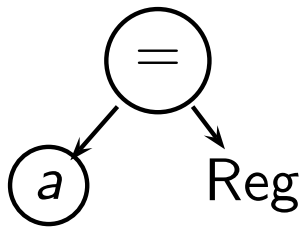
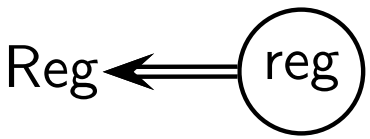
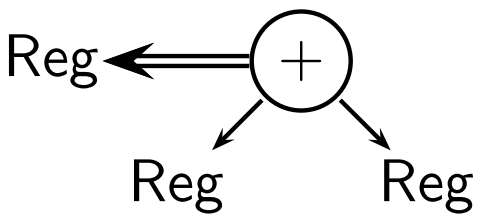
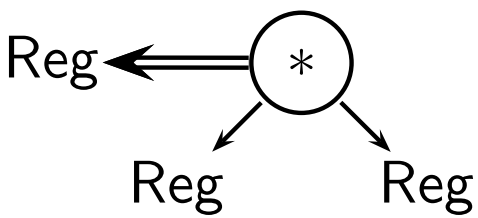
Tree Tiling

Instructions are viewed as composable rules



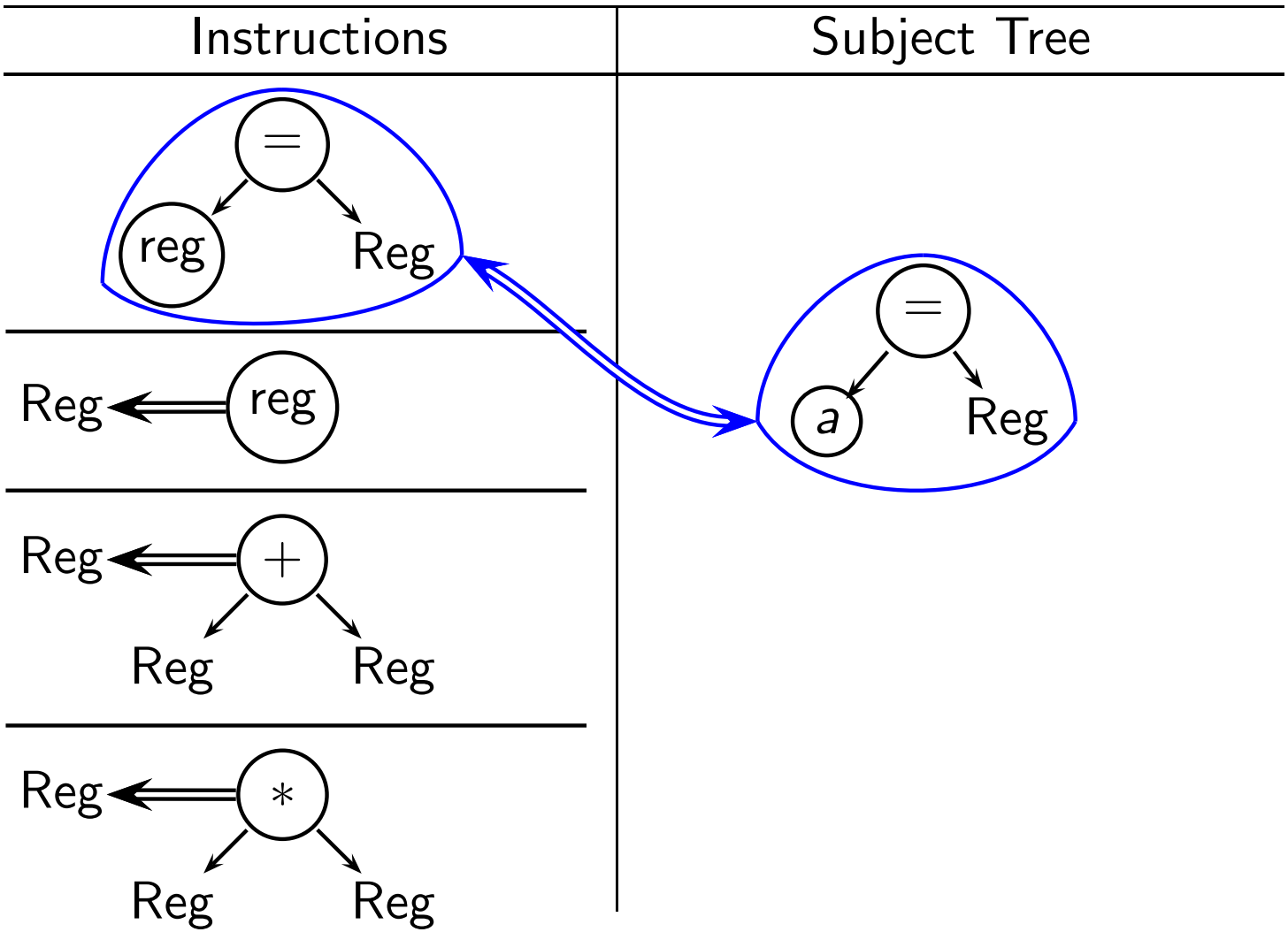
Tree Tiling

Instructions are viewed as composable rules

Instructions	Subject Tree
	
	
	
	

Tree Tiling

Instructions are viewed as composable rules



Improving Machine Descriptions and Instruction Selection

Current Status:

- Preliminary investigations seem very promising
  - Fewer rules
  - Simple rules

Improving Machine Descriptions and Instruction Selection

Current Status:

- Preliminary investigations seem very promising
  - Fewer rules
  - Simple rules
- Prototype of new code generator generator (cgg) is being tested in a toy compiler set up

Part 5

GCC Resource Center

National Resource Center for F/OSS, Phase II

- Sponsored by Department of Information Technology (DIT), Ministry of Information and Communication Technology
- CDAC Chennai is the coordinating agency
- Participating agencies

Organization	Focus
CDAC Chennai	SaaS Model, Mobile Internet Devices on BOSS, BOSS applications
CDAC Mumbai	FOSS Knowledge Base, FOSS Desktops
CDAC Hyderabad	E-Learning
IIT Bombay	Gnu Compiler Collection
IIT Madras	OO Linux kernel
Anna University	FOSS HRD

Objectives of GCC Resource Center

1. To support the open source movement
   Providing training and technical know-how of the GCC framework to academia and industry.
2. To include better technologies in GCC
   Whole program optimization, Optimizer generation, Tree tiling based instruction selection.
3. To facilitate easier and better quality deployments/enhancements of GCC
   Restructuring GCC and devising methodologies for systematic construction of machine descriptions in GCC.
4. To bridge the gap between academic research and practical implementation
   Designing suitable abstractions of GCC architecture

GRC Training Programs

Title	Target	Objectives	Mode	Duration
Workshop on Essential Abstractions in GCC	People interested in deploying or enhancing GCC	Explaining the essential abstractions in GCC to ensure a quick ramp up into GCC Internals	Lectures, demonstrations, and practicals (experiements and assignments)	Three days
Tutorial on Essential Abstractions in GCC	People interested in knowing about issues in deploying or enhancing GCC	Explaining the essential abstractions in GCC to ensure a quick ramp up into GCC Internals	Lectures and demonstrations	One day
Workshop on Compiler Construction with Introduction to GCC	College teachers	Explaining the theory and practice of compiler contruction and illustrating them with the help of GCC	Lectures, demonstrations, and practicals (experiements and assignments)	Seven days
Tutorial on Demystifying GCC Compilation	Students	Explaining the translation sequence of GCC through gray box probing (i.e. by examining the dumps produced by GCC)	Lectures and demonstrations	Half day

GRC Training Programs

Title	Target	Objectives	Mode	Duration
Workshop on Essential Abstractions in GCC	People interested in deploying or enhancing	Explaining the essential	Lectures, demonstrations, practicals (experiements and assignments)	Three days
Tutorial on Essential Abstractions in GCC	People interested in knowing issues i deploying or enhancing GCC	Explaining the essential	Lectures and demonstrations	One day
Workshop on Compiler Construction with Introduction to GCC	College teachers	Explaining the theory and help of GCC	Lectures, demonstrations, and practicals (experiements and assignments)	Seven days
Tutorial on Demystifying GCC Compilation	Students	Explaining the translation produced by GCC)	Lectures and demonstrations	Half day

3, 4, and 5 July, 2009
IIT Bombay, Mumbai

(modified version) 9 Jan 2010
ACM PPoPP, Banglore

7-13 Dec 2009,
IIT Bombay, Mumbai

20 Jan 2010,
Cummins College, Pune

GRC Training Programs

CS 715: The Design and Implementation of GNU Compiler Generation Framework

- 6 credits semester long course for M.Tech. (CSE) students at IIT Bombay
- Significant component of experimentation with GCC
- Introduced in 2008-2009

Part 6

Conclusions

Conclusions

GCC is a strange paradox

- Practically very successful
  - Readily available without any restrictions
  - Easy to use
  - Easy to examine compilation without knowing internals
  - Available on a wide variety of processors and operating systems
  - Can be retargeted to new processors and operating systems

Conclusions

GCC is a strange paradox

- Practically very successful
  - Readily available without any restrictions
  - Easy to use
  - Easy to examine compilation without knowing internals
  - Available on a wide variety of processors and operating systems
  - Can be retargeted to new processors and operating systems
- Quite adhoc

Conclusions

GCC is a strange paradox

- Practically very successful
  - Readily available without any restrictions
  - Easy to use
  - Easy to examine compilation without knowing internals
  - Available on a wide variety of processors and operating systems
  - Can be retargeted to new processors and operating systems
- Quite adhoc
  - Needs significant improvements in terms of design
    Machine description specification, IRs, optimizer generation

Conclusions

GCC is a strange paradox

- Practically very successful
  - Readily available without any restrictions
  - Easy to use
  - Easy to examine compilation without knowing internals
  - Available on a wide variety of processors and operating systems
  - Can be retargeted to new processors and operating systems
- Quite adhoc
  - Needs significant improvements in terms of design
    Machine description specification, IRs, optimizer generation
  - Needs significant improvements in terms of better algorithms
    Retargetability mechanism, interprocedural optimizations, parallelization, vectorization,

Conclusions

GCC Resource Center 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

Last but not the least ...

Thank You!