Incremental Machine Descriptions for GCC

Sameera Deshpande Uday Khedker.

(www.cse.iitb.ac.in/~{sameera,uday})

Department of Computer Science and Engineering,

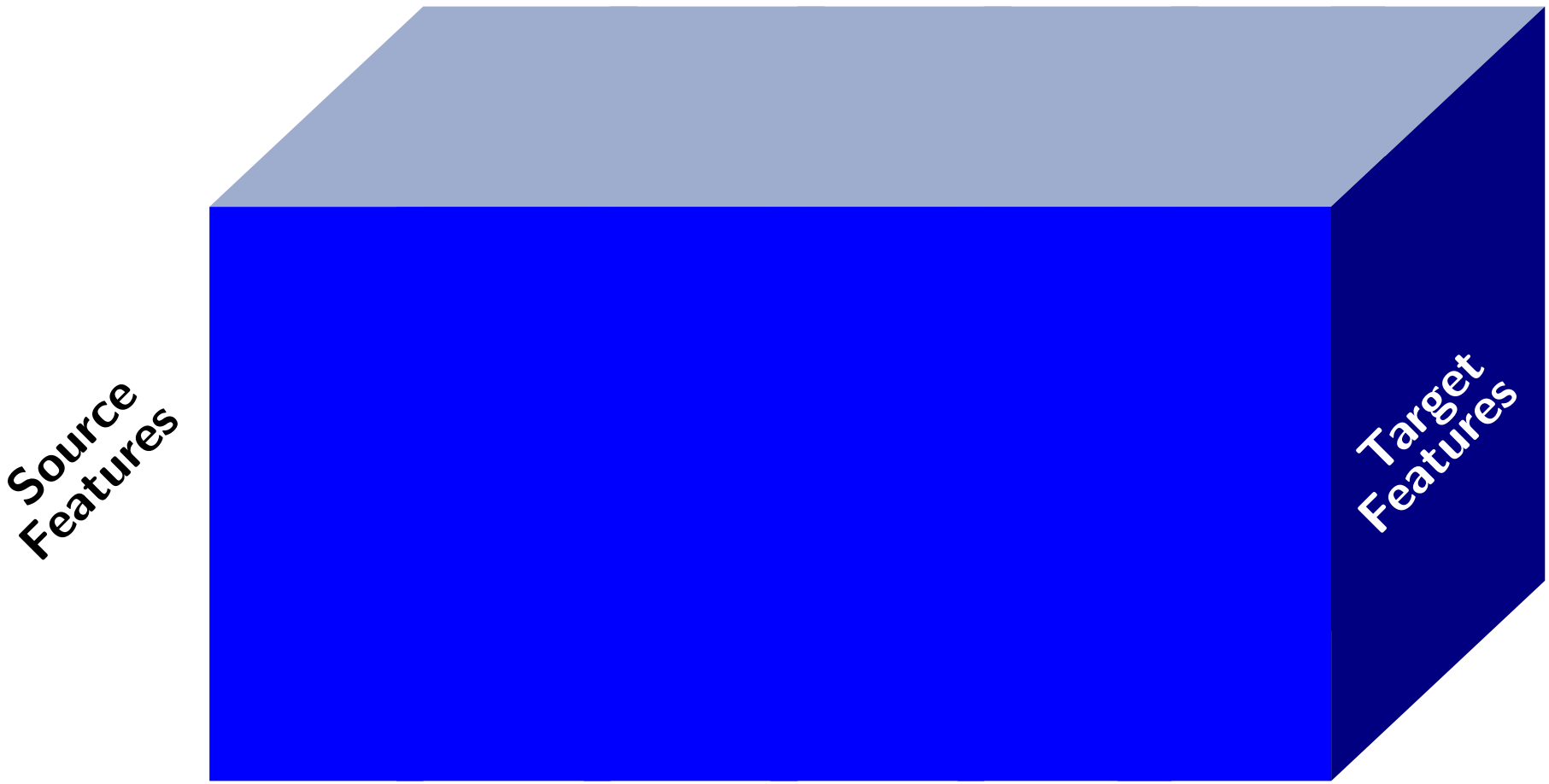
Indian Institute of Technology, Bombay

September 16, 2007

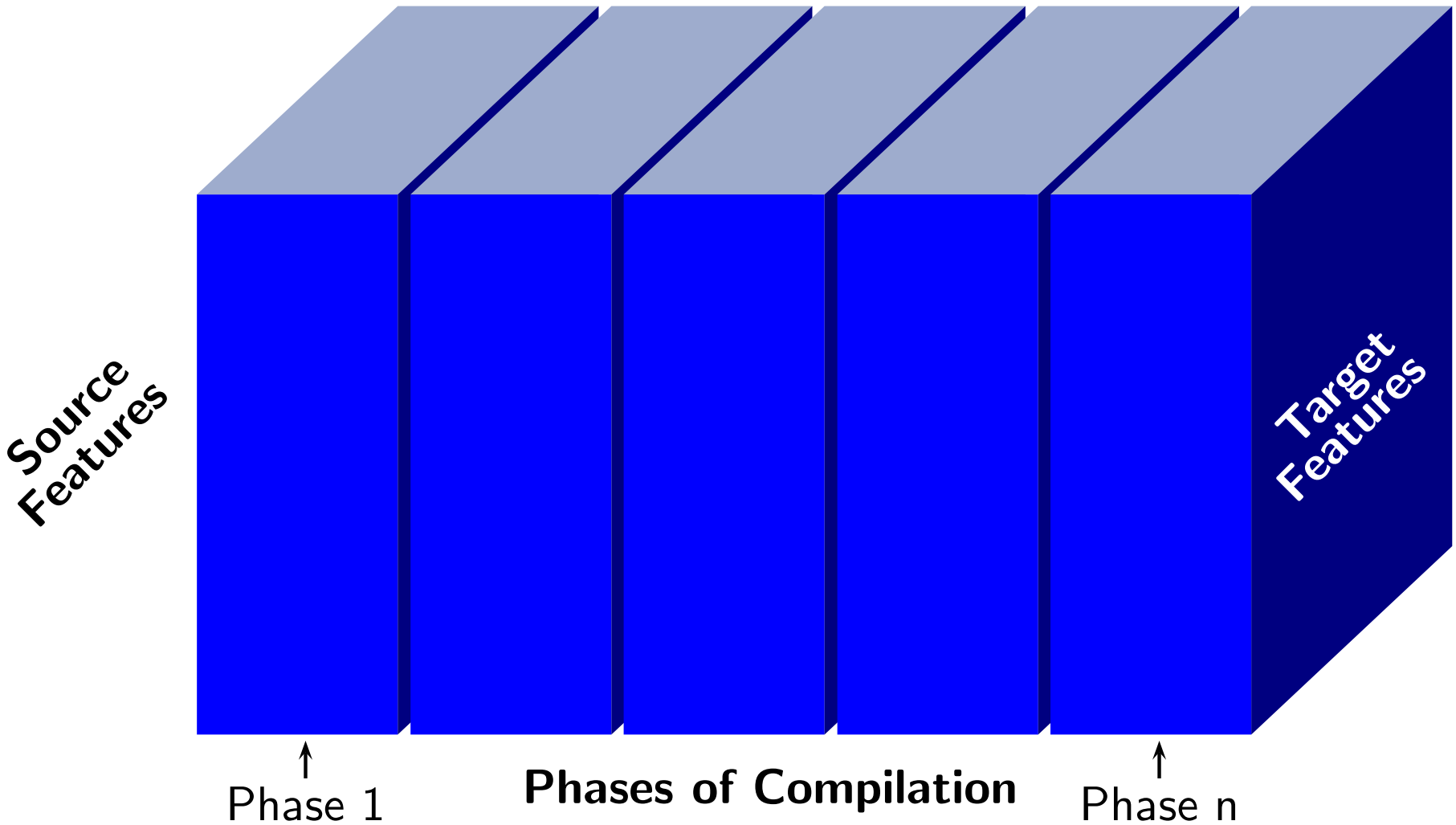
Part 1

Incremental Machine Descriptions

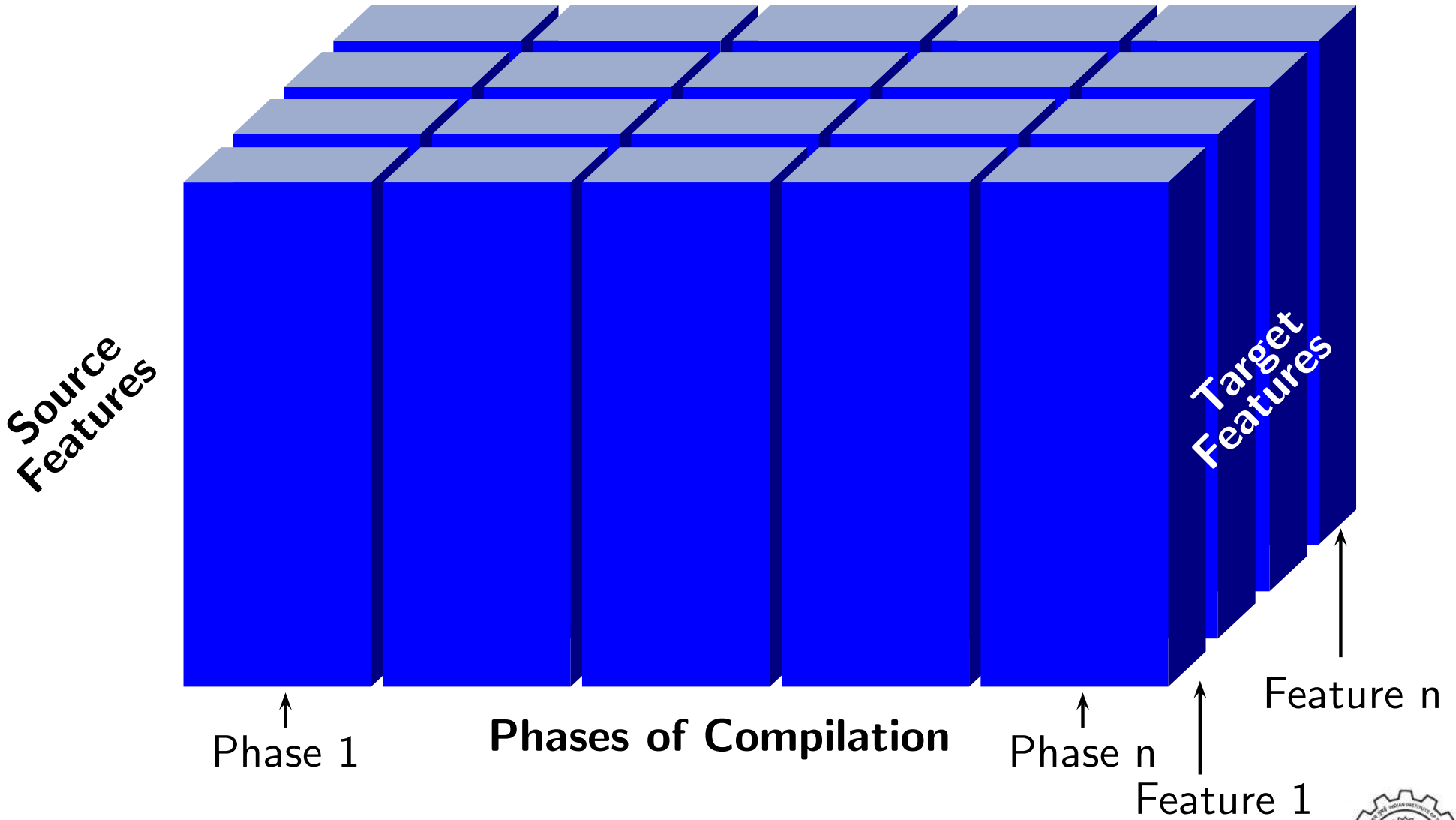
Modularity in Compilation



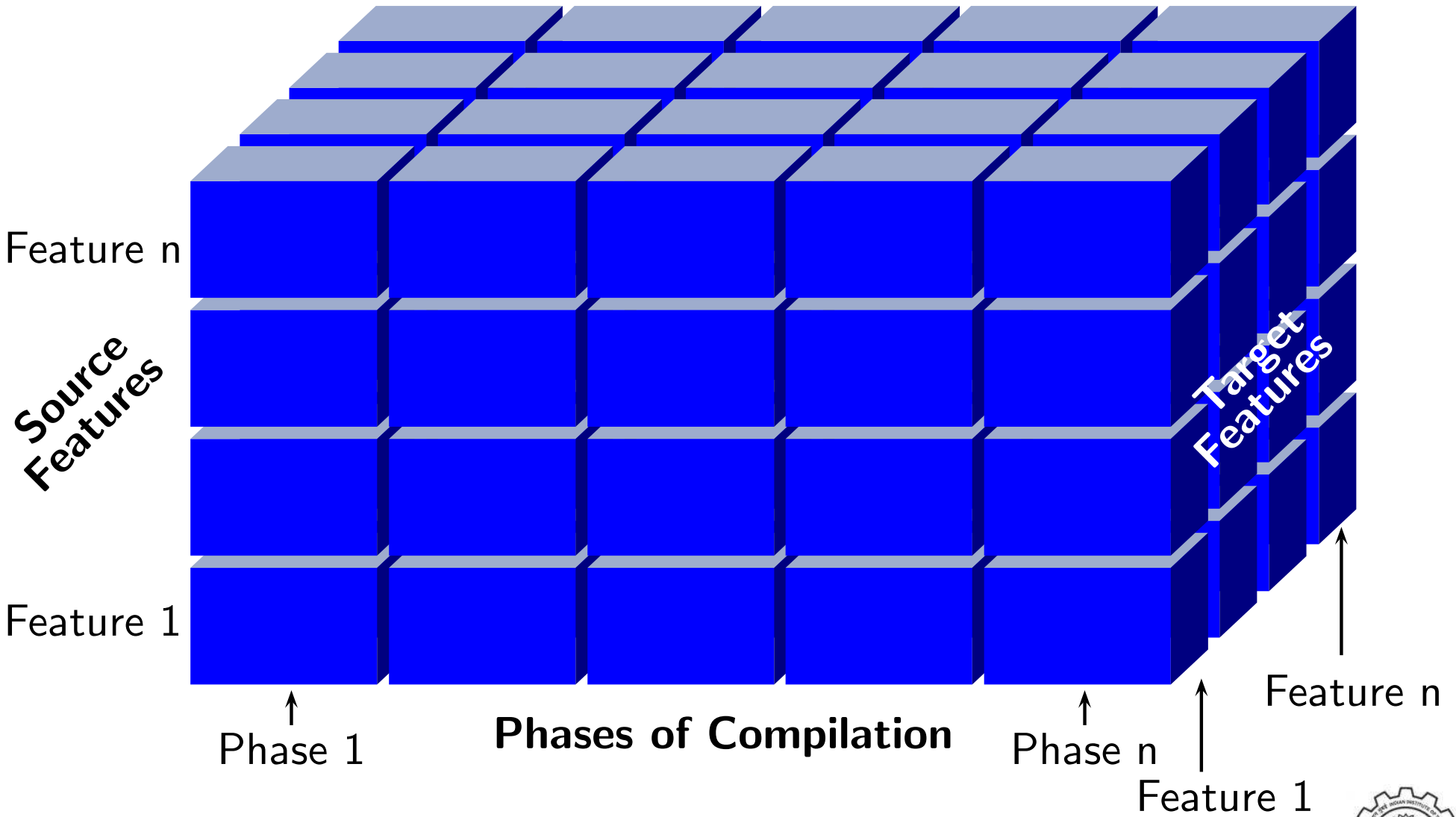
Modularity in Compilation



Modularity in Compilation



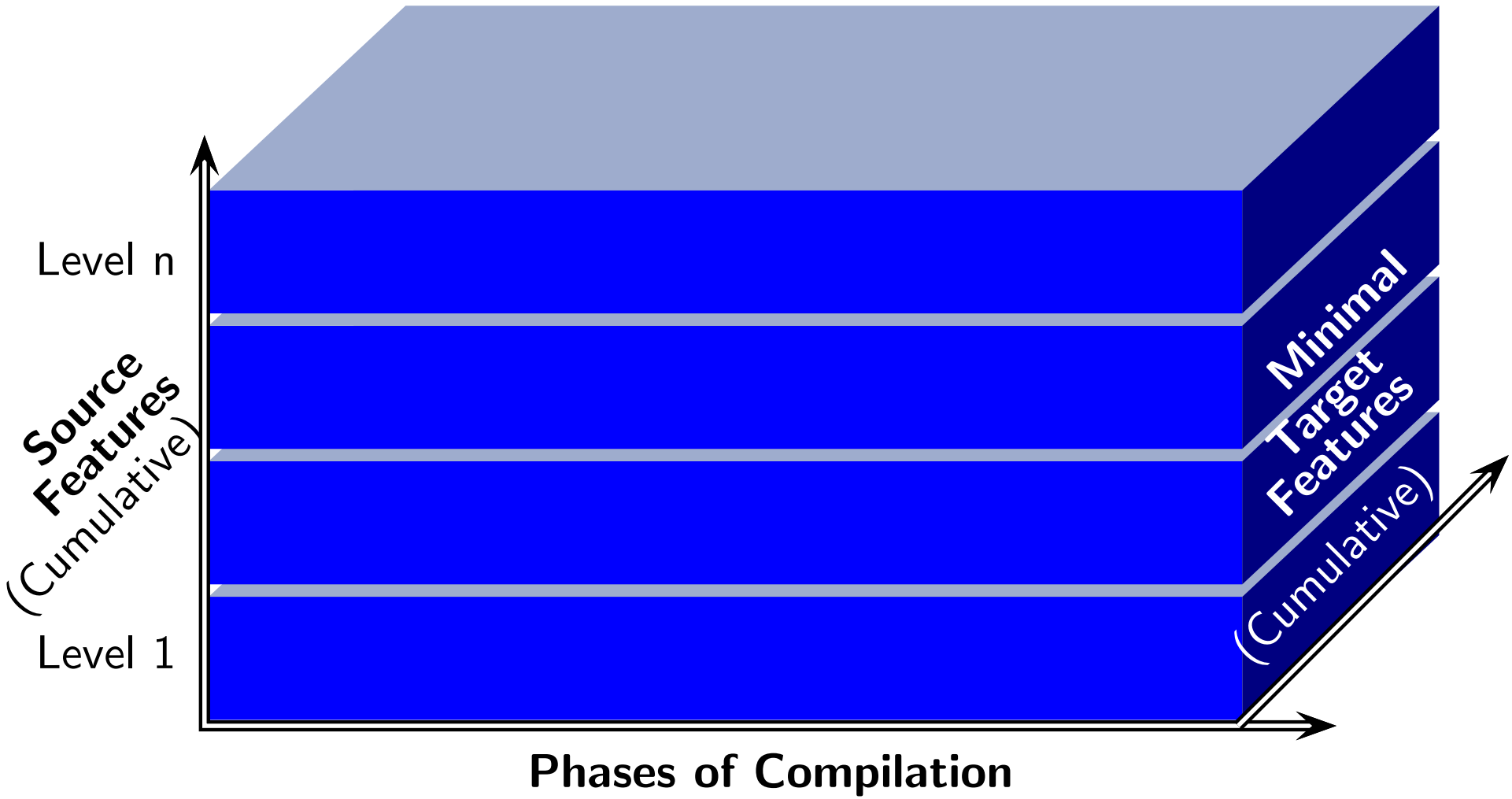
Modularity in Compilation



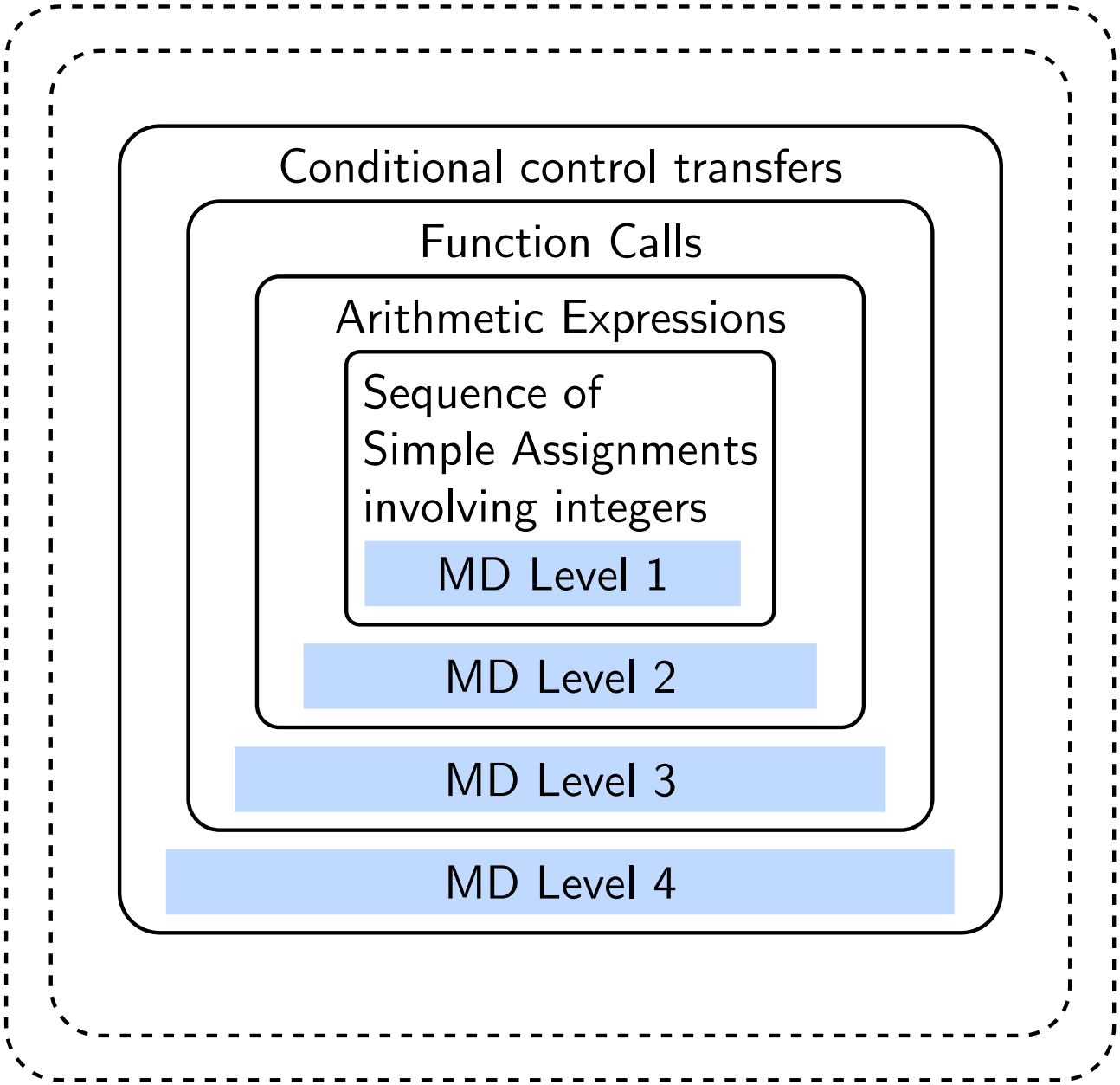
Modularity in Compilation

- Is it possible to achieve such a clean division?
  - Compilation phases may be independent in principle but in practice, the interfaces carry a lot of information making the intermdediate representations quite complex.
  - Translation of many source features depend on translation of other source features.
  - Many target features are are dependent on each other.
- How does one understand the abstractions which GCC machine descriptions capture?
- How does one partition machine descriptions?

Appropriate Modularity for Machine Descriptions?



Systematic Development of Machine Descriptions



Systematic Development of Machine Descriptions

- Define different levels of source language
- 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.

Part 2

Level 0 of Spim Machine Descriptions

Sub-levels of Level 0

Three sub-levels

- Level 0.0: Merely build GCC for spim simulator
  Does not compile any program (i.e. compilation aborts)
- Level 0.1: Compiles empty parameterless void function

```
void fun()
{
}
```

```
void fun()
{
  L: goto L;
}
```

- Level 0.2: Incorporates complete activation record structure
  Required for Level 1

Category of Macros in Level 0

Category	Level 0.0	Level 0.1	Level 0.2
Memory Layout	complete	complete	complete
Registers	partial	partial	complete
Addressing Modes	none	partial	partial
Activation Record Conventions	dummy	dummy	complete
Calling Conventions	dummy	dummy	partial
Assembly Output Format	dummy	partial	partial

- Complete specification of activation record in level 0.2 is not necessary but is provided to facilitate local variables in level 1.
- Complete specification of registers in level 0.2 follows the complete specification of activation record.

Operations in Level 0.0

- In principle `spim0.0.md` can be empty.

Operations in Level 0.0

- In principle `spim0.0.md` can be empty.
- However, this results in empty data structures in the C source. Empty arrays in declarations are not acceptable in ANSI C.

Operations in Level 0.0

- In principle `spim0.0.md` can be empty.
- However, this results in empty data structures in the C source. Empty arrays in declarations are not acceptable in ANSI C.
- If we remove `-pedantic` option while building `gcc`, the compiler gets built.

Operations in Level 0.0

- In principle `spim0.0.md` can be empty.
- However, this results in empty data structures in the C source. Empty arrays in declarations are not acceptable in ANSI C.
- If we remove `-pedantic` option while building `gcc`, the compiler gets built.
- If we do not want to change the configuration system, `spim0.0.md` can be defined to contain

```
(define_insn "dummy_pattern"
        [(reg:SI 0)]
        "1"
        "This stmnt should not be emitted!"
)
```


Operations in Level 0

Operations	Level 0.0	Level 0.1	Level 0.2
JUMP direct	dummy	actual	actual
JUMP indirect	dummy	dummy	dummy
RETURN	not required	required	required

Operations in Level 0

Operations	Level 0.0	Level 0.1	Level 0.2
JUMP direct	dummy	actual	actual
JUMP indirect	dummy	dummy	dummy
RETURN	not required	partial	partial

spim0.2.md

```
(define_insn "jump"
   [(set (pc)
     (label_ref (match_operand 0 "" ""))
    )]
   ""
   "ja %l0"
)
```

Operations in Level 0

Operations	Level 0.0	Level 0.1	Level 0.2
JUMP direct	dummy	actual	actual
JUMP indirect	dummy	dummy	dummy
RETURN	not required	partial	partial

spim0.0.c

```
rtx gen_jump (...)
{ return 0; }
rtx gen_indirect_jump (...)
{ return 0; }
```

spim0.0.h

```
#define CODE_FOR_indirect_jump 8
```

spim0.2.md

```
(define_insn "jump"
   [(set (pc)
     (label_ref (match_operand 0 "" ""))
    )]
   ""
   "j %l0"
)
```

Operations in Level 0

Operations	Level 0.0	Level 0.1	Level 0.2
JUMP direct	dummy	actual	actual
JUMP indirect	dummy	dummy	dummy
RETURN	not required	partial	partial

spim0.2.md

```
(define_expand "epilogue"
   [(clobber (const_int 0))]
   ""
   { spim_epilogue();
     DONE;
   }
)
(define_insn "IITB_return"
   [(return)]
   ""
   "jr \\$ra"
)
```

spim0.2.c

```
void spim_epilogue()
{
    emit_insn(gen_IITB_return());
}
```

Part 3

Level 1 of Spim Machine Descriptions

Increments for Level 1

- Addition to the source language
  - Assignment statements involving integer constant, integer local or global variables.
  - Returning values. (No calls, though!)
- Changes in machine descriptions
  - Minor changes in macros required for level 0
    - `$zero` now belongs to new class
    - Assembly output needs to change
  - Some function bodies expanded
  - New operations included in the `.md` file

`diff -w` shows the changes!

Operations Required in Level 1

Operation	Primitive Variants	Implementation
$Dest \leftarrow Src$	$R_i \leftarrow R_j$	`move rj, ri`
	$R \leftarrow M_{global}$	`lw r, m`
	$R \leftarrow M_{local}$	`lw r, c($fp)`
	$R \leftarrow C$	`li r, c`
	$M \leftarrow R$	`sw r, m`
RETURN Src	RETURN Src	`$v0` $\leftarrow Src$ `j $ra`
$Dest \leftarrow Src_1 + Src_2$	$R_i \leftarrow R_j + R_k$	`add ri, rj, rk`
	$R_i \leftarrow R_j + C$	`addi ri, rj, c`

Part 4

Level 2 of Spim Machine Descriptions

Increments for Level 2

- Addition to the source language :
  - Arithmetic Operations.
  - Bitwise Operations.
- Sub-levels of Level 2
  - Level 2.0: Compound instructions handled at assembly emission time.
  - Level 2.1: Compound instructions emitted seperately at RTL code generation time.
  - Level 2.2: Compound instructions are emitted atomically, but 'splitted' in later phases.

Arithmetic Operations Required in Level 2

Operation	Primitive Variants	Implementation
$Dest \leftarrow Src_1 - Src_2$	$R_i \leftarrow R_j - R_k$	`sub ri, rj, rk`
$Dest \leftarrow -Src$	$R_i \leftarrow -R_j$	`neg ri, rj`
$Dest \leftarrow Src_1/Src_2$	$R_i \leftarrow R_j/R_k$	`div rj, rk` `mflo ri`
$Dest \leftarrow Src_1\%Src_2$	$R_i \leftarrow R_j\%R_k$	`rem ri, rj, rk`
$Dest \leftarrow Src_1 * Src_2$	$R_i \leftarrow R_j * R_k$	`mul ri, rj, rk`

Bitwise Operations Required in Level 2

Operation	Primitive Variants	Implementation
$Dest \leftarrow Src_1 \ll Src_2$	$R_i \leftarrow R_j \ll R_k$	`sllv ri, rj, rk`
	$R_i \leftarrow R_j \ll C_5$	`sll ri, rj, c`
$Dest \leftarrow Src_1 \gg Src_2$	$R_i \leftarrow R_j \gg R_k$	`srav ri, rj, rk`
	$R_i \leftarrow R_j \gg C_5$	`sra ri, rj, c`
$Dest \leftarrow Src_1 \& Src_2$	$R_i \leftarrow R_j \& R_k$	`and ri, rj, rk`
	$R_i \leftarrow R_j \& C$	`andi ri, rj, c`
$Dest \leftarrow Src_1 \mid Src_2$	$R_i \leftarrow R_j \mid R_k$	`or ri, rj, rk`
	$R_i \leftarrow R_j \mid C$	`ori ri, rj, c`
$Dest \leftarrow Src_1 \ \hat{} \ Src_2$	$R_i \leftarrow R_j \ \hat{} \ R_k$	`xor ri, rj, rk`
	$R_i \leftarrow R_j \ \hat{} \ C$	`xori ri, rj, c`
$Dest \leftarrow \sim Src$	$R_i \leftarrow \sim R_j$	`not ri, rj`

Changes in machine descriptions

MD	Level 2.0	Level 2.1	Level 2.2
Macros	Level 1	Changed	Level 2.1
Function definitions	Level 1	Added	Level 2.1
md instructions	Added SPNs	SPNs + Support	SPNs + Support

Part 5

Level 3 of Spim Machine Descriptions

Operations Required in Level 3

Operation	Primitive Variants	Implementation	Remark

$Dest \leftarrow fun(P_1, \ldots, P_n)$	call L_{fun}, n	lw r_i, [SP]
		sw r_i, [SP]
		:
		lw r_i, [SP-n*4]
		sw r_i, [SP-n*4]
			Level 1

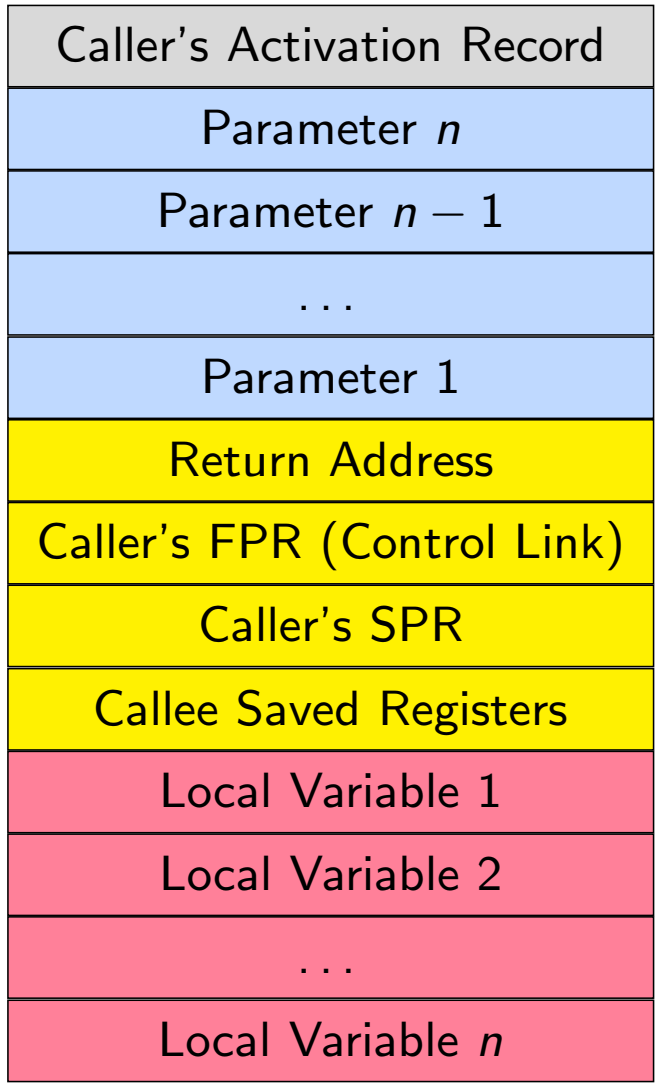
		jal L	New
		$Dest \leftarrow \$v0$	level 1

$fun(P_1, P_2, \ldots, P_n)$	call L_{fun}, n	lw r_i, [SP]
		sw r_i, [SP]
		:
		lw r_i, [SP-n*4]
		sw r_i, [SP-n*4]
			Level 1

		jal L	New

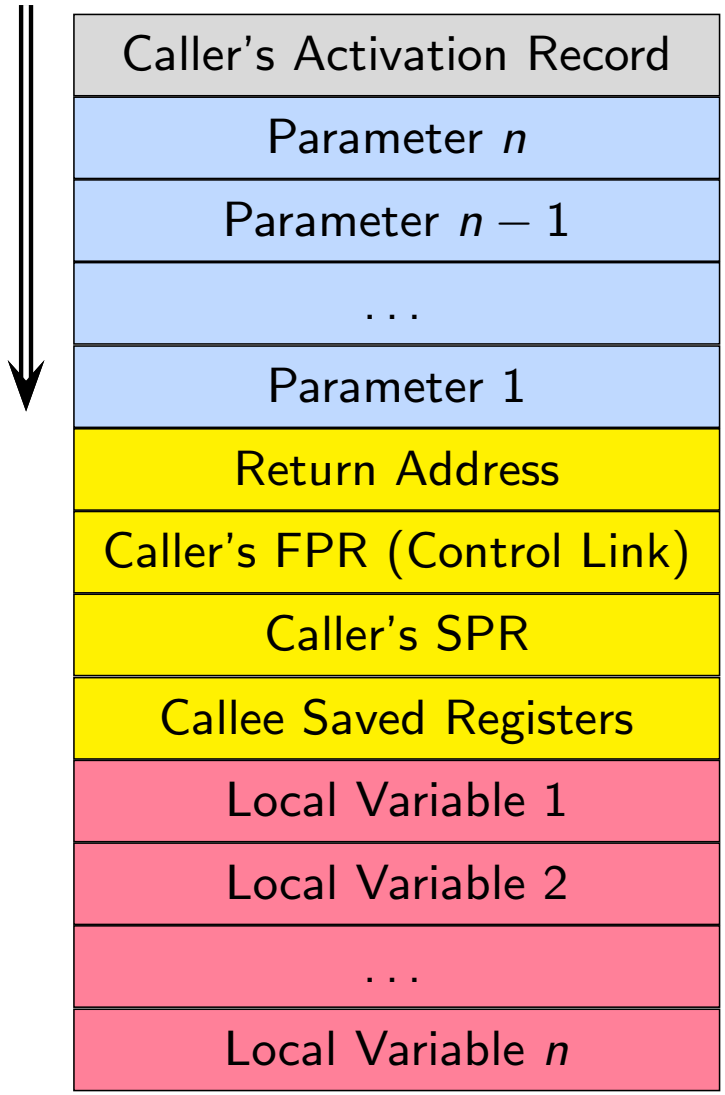
$$a = fun(b + c, b, c) \rightarrow \begin{cases} addi & \$sp, \$sp, -16 \\ move & \$a0, \$sp \\ sw & \$v0, 0(\$a0) \\ lw & \$v0, -20(\$fp) \\ sw & \$v0, 4(\$a0) \\ lw & \$v0, -16(\$fp) \\ sw & \$v0, 8(\$a0) \\ jal & \_fun \\ addi & \$sp, \$sp, 16 \\ sw & \$v0, -24(\$fp) \end{cases}$$

Activation Record Generation during `Call`



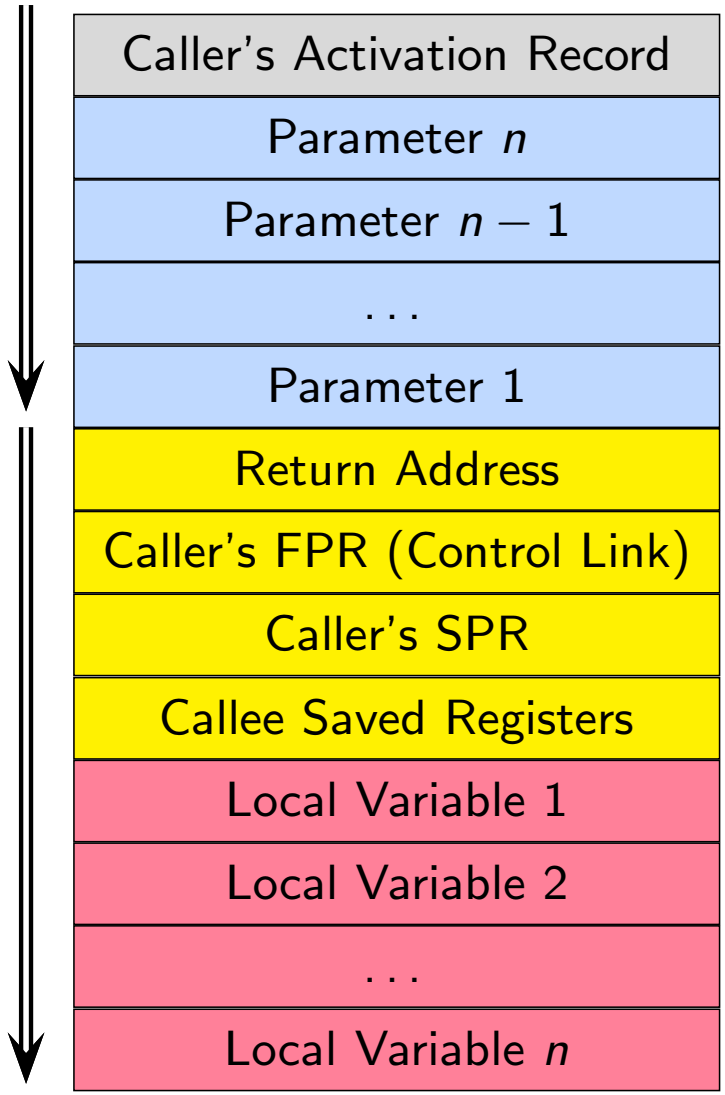
Activation Record Generation during `Call`

- Operations performed by caller
  - Push parameters on stack.
  - Load return address in return address register.
  - Transfer control to Callee.



Activation Record Generation during `Call`

- Operations performed by caller
  - Push parameters on stack.
  - Load return address in return address register.
  - Transfer control to Callee.
- Operations performed by callee
  - Push Return address stored by caller on stack.
  - Push caller's Frame Pointer Register.
  - Push caller's Stack Pointer.
  - Save callee saved registers, if used by callee.
  - Create local variables frame.
  - Start callee body execution.



Part 6

Level 4 of Spim Machine Descriptions

Operations Required in Level 4

Operation	Primitive Variants	Implementation
$Src_1 < Src_2$? goto L : PC	$CC \leftarrow R_i < R_j$ CC < 0 ? goto L : PC	blt r_i, r_j,L
$Src_1 > Src_2$? goto L : PC	$CC \leftarrow R_i > R_j$ CC > 0 ? goto L : PC	bgt r_i, r_j,L
$Src_1 \leq Src_2$? goto L : PC	$CC \leftarrow R_i \leq R_j$ CC ≤ 0 ? goto L : PC	ble r_i, r_j,L
$Src_1 \geq Src_2$? goto L : PC	$CC \leftarrow R_i \geq R_j$ CC ≥ 0 ? goto L : PC	bge r_i, r_j,L

Operations Required in Level 4

Operation	Primitive Variants	Implementation
$Src_1 == Src_2$? goto L : PC	$CC \leftarrow R_i == R_j$ CC == 0 ? goto L : PC	beq r_i, r_j,L
$Src_1 \neq Src_2$? goto L : PC	$CC \leftarrow R_i \neq R_j$ CC $\neq$ 0 ? goto L : PC	bne r_i, r_j,L

Part 7

Summary

Summary of Incremental Construction of MD

- What is a minimal machine description?

Summary of Incremental Construction of MD

- What is a minimal machine description?
- We compared ALL machine descriptions available in the 4.0.2 distribution
  - Number of macros common to these descriptions: 63

Summary of Incremental Construction of MD

- What is a minimal machine description?
- We compared ALL machine descriptions available in the 4.0.2 distribution
  - Number of macros common to these descriptions: 63
  - Number of distinct macros in all these descriptions: 732!

Summary of Incremental Construction of MD

- What is a minimal machine description?
- We compared ALL machine descriptions available in the 4.0.2 distribution
  - Number of macros common to these descriptions: 63
  - Number of distinct macros in all these descriptions: 732!
  - Number of macros in spim0.0 : 92

Summary of Incremental Construction of MD

- What is a minimal machine description?
- We compared ALL machine descriptions available in the 4.0.2 distribution
  - Number of macros common to these descriptions: 63
  - Number of distinct macros in all these descriptions: 732!
  - Number of macros in spim0.0 : 92
- Incremental construction of MD governed by increments in source language features is a powerful methodology of learning the abstractions implicitly embodied by the machine descriptions!

Part 8

Thank You!