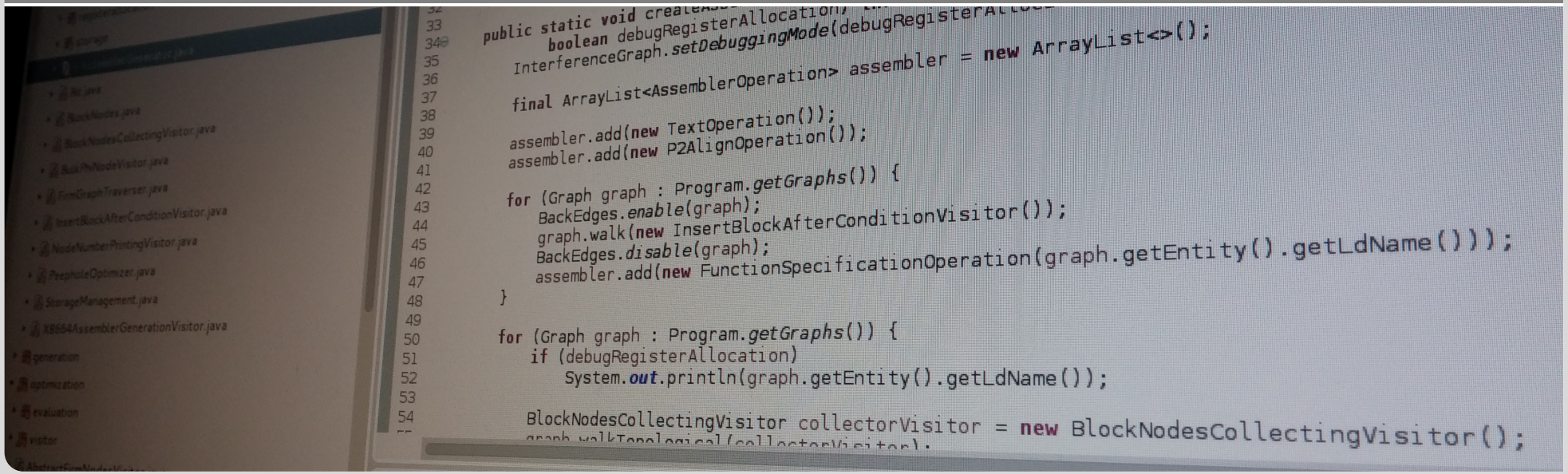
Karlsruhe Institute of Technology

Compiler: Assembler Generation

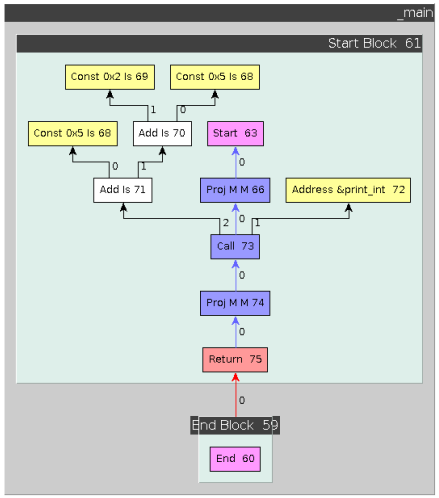
Norman Böwing, Andreas Eberle, Aleksej Frank, Polina Goltsman and Valentin Zickner

KARLSRUHE INSTITUTE OF TECHNOLOGY (KIT)



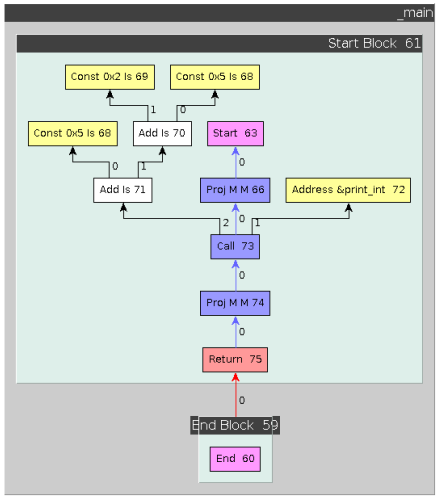
Tasks of Assembler Generation

- Precondition: Firm-Graph



Tasks of Assembler Generation

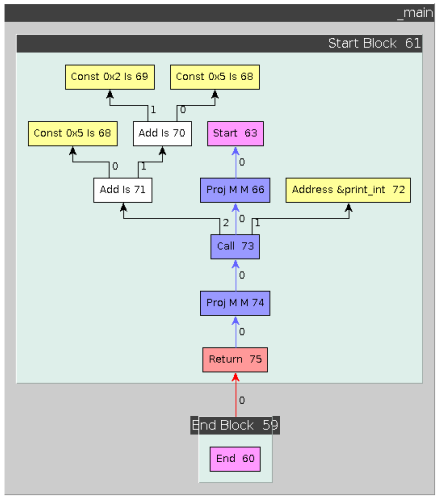
- Precondition: Firm-Graph
- Result: x86_64 Assembler




```
        .text
        .p2align 4,,15
        .globl _main
        .type   _main, @function
.p2align 4
_main:
.L61:
        movl $0x2, %ebp # Const Is<0x2>[69:10]
        add $0x5, %ebp  # Add Is[70:11]
        add $0x5, %ebp  # Add Is[71:12]
        movl %ebp, %edi
        call print_int  # Call T[73:14]
        ret
        .size   _main, .-_main
.L59:
        # end node
```

Tasks of Assembler Generation

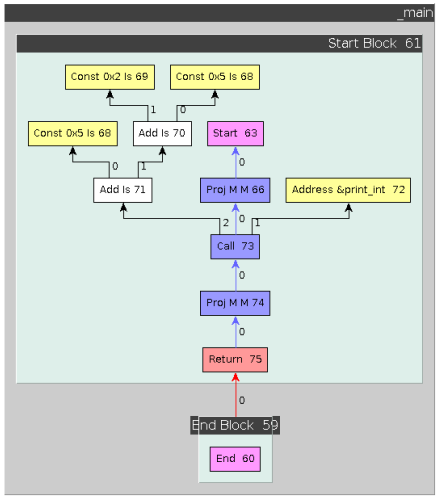
- Precondition: Firm-Graph
- Result: x86_64 Assembler
- Tasks:
  - Instruction selection
  - Instruction scheduling
  - Naive register allocation: Use stack




```
        .text
        .p2align 4,,15
        .globl _main
        .type   _main, @function
.p2align 4
_main:
.L61:
        movl $0x2, %ebp # Const Is<0x2>[69:10]
        add $0x5, %ebp  # Add Is[70:11]
        add $0x5, %ebp  # Add Is[71:12]
        movl %ebp, %edi
        call print_int  # Call T[73:14]
        ret
        .size   _main, .-_main
.L59:
        # end node
```

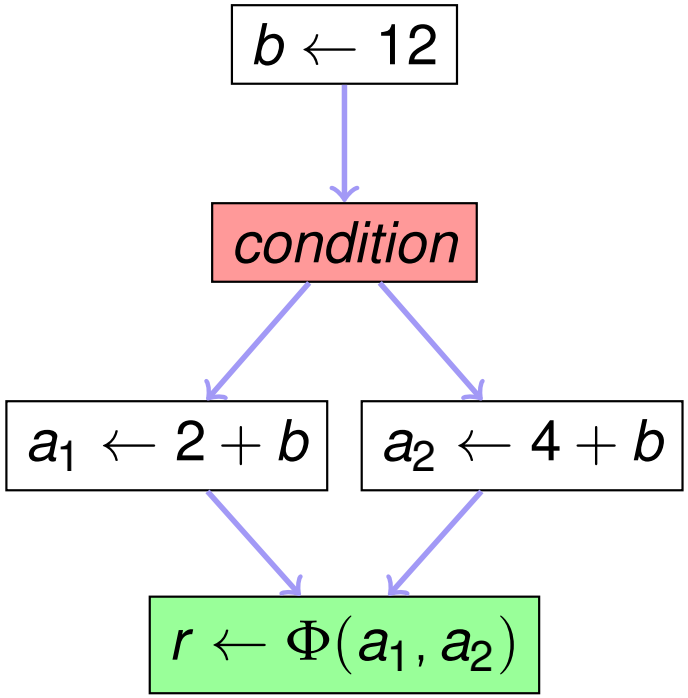
Tasks of Assembler Generation

- Precondition: Firm-Graph
- Result: x86_64 Assembler
- Tasks:
  - Instruction selection
  - Instruction scheduling
  - Naive register allocation: Use stack
- Problems:
  - Memory access is expensive

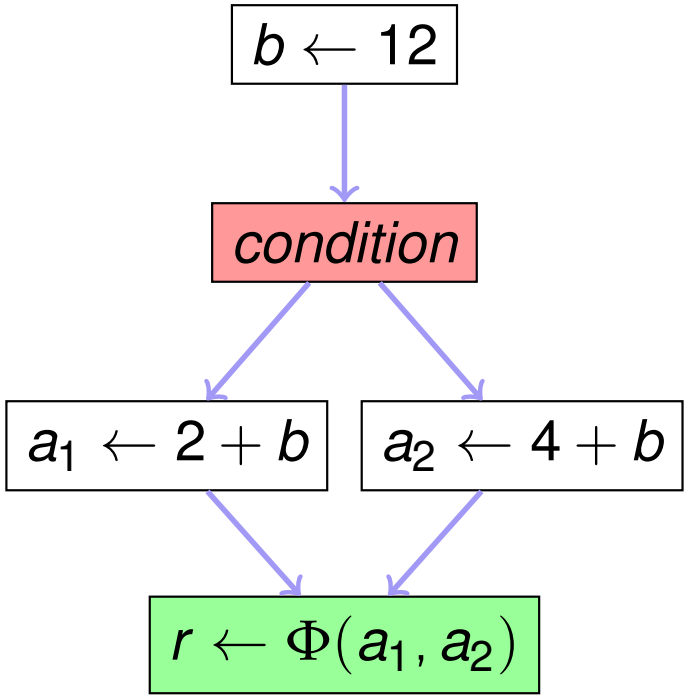



```
        .text
        .p2align 4,,15
        .globl _main
        .type   _main, @function
.p2align 4
_main:
.L61:
        movl $0x2, %ebp # Const Is<0x2>[69:10]
        add $0x5, %ebp  # Add Is[70:11]
        add $0x5, %ebp  # Add Is[71:12]
        movl %ebp, %edi
        call print_int  # Call T[73:14]
        ret
        .size   _main, .-_main
.L59:
        # end node
```

Principial procedure

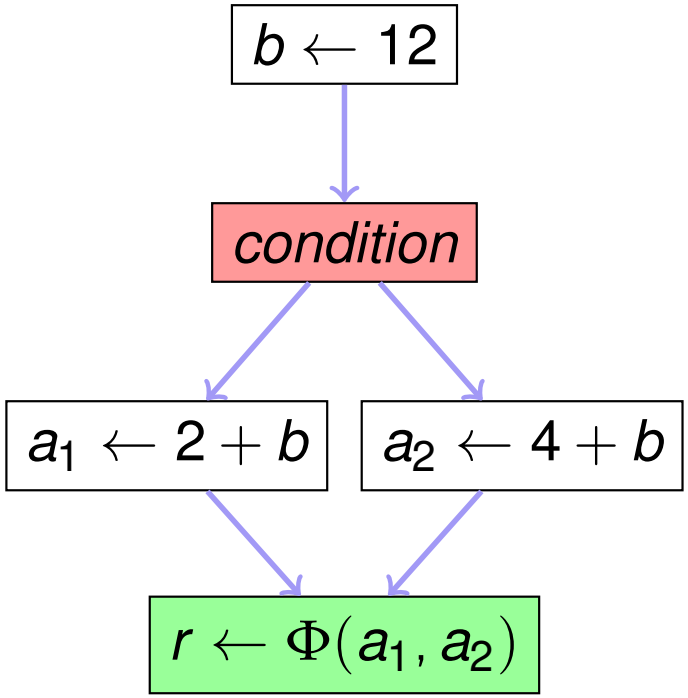


Principial procedure




```
        R1 ← 12
        condition
        jnz ifcase
        jmp elsecase
ifcase :    R2 ← add 2, R1
        jmp end
elsecase :  R3 ← add 4, R1
        jmp end
end :       R4 ← Φ(R2, R3)
```

Principial procedure

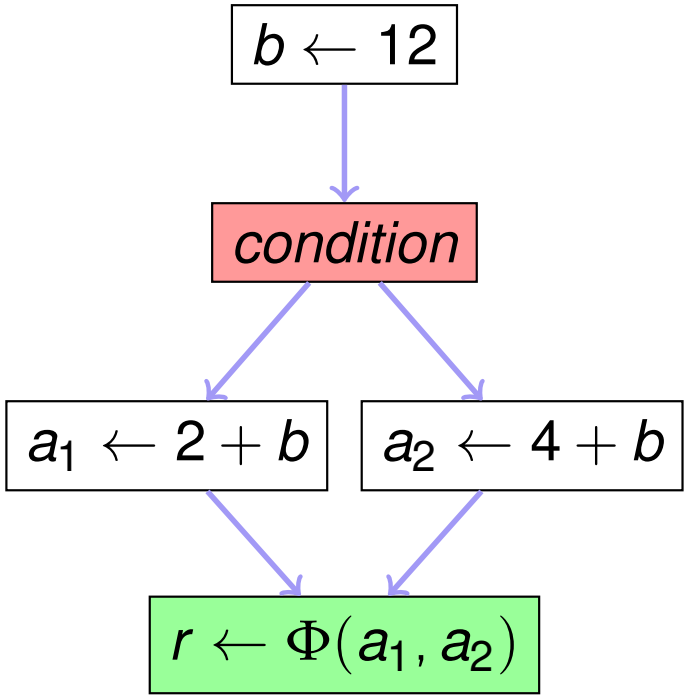


$$
\begin{array}{ll}
 & R1 \leftarrow 12 \\
 & condition \\
 & jnz\ ifcase \\
 & jmp\ elsecase \\
ifcase: & R2 \leftarrow add\ 2, R1 \\
 & jmp\ end \\
elsecase: & R3 \leftarrow add\ 4, R1 \\
 & jmp\ end \\
end: & R4 \leftarrow \Phi(R2, R3)
\end{array}
$$

Challenges:

- Firm nodes not extendable in jFirm

Principial procedure

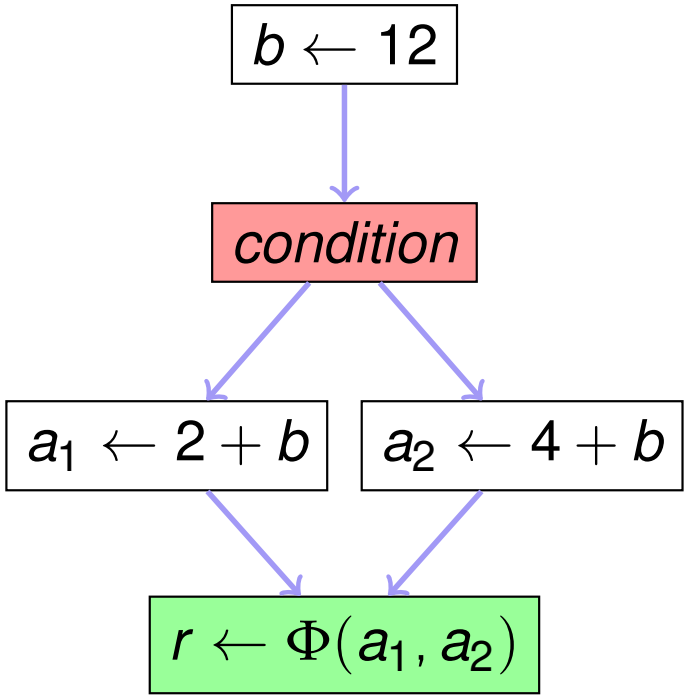


$$
\begin{array}{ll}
 & R1 \leftarrow 12 \\
 & condition \\
 & jnz\ ifcase \\
 & jmp\ elsecase \\
ifcase: & R2 \leftarrow add\ 2, R1 \\
 & jmp\ end \\
elsecase: & R3 \leftarrow add\ 4, R1 \\
 & jmp\ end \\
end: & R4 \leftarrow \Phi(R2, R3)
\end{array}
$$

Challenges:

- Firm nodes not extendable in jFirm
- Two operand code in x86_64 assembler:
  $R2 \leftarrow add\ 2, R1 \Rightarrow mov\ R1, R2; add\ 2, R2$

Principial procedure



$$
\begin{array}{ll}
 & R1 \leftarrow 12 \\
 & condition \\
 & jnz\ ifcase \\
 & jmp\ elsecase \\
ifcase: & R2 \leftarrow add\ 2, R1 \\
 & jmp\ end \\
elsecase: & R3 \leftarrow add\ 4, R1 \\
 & jmp\ end \\
end: & R4 \leftarrow \Phi(R2, R3)
\end{array}
$$

Challenges:

- Firm nodes not extendable in jFirm
- Two operand code in x86_64 assembler:
  $R2 \leftarrow add\ 2, R1 \Rightarrow mov\ R1, R2; add\ 2, R2$
- Register allocation

Register Allocation: Linear Scan

- Precondition: Linear block order with virtual registers

Register Allocation: Linear Scan

- Precondition: Linear block order with virtual registers

```
1                 R1 ← 12
2                 condition
3                 jnz ifcase
4                 jmp elsecase
5   ifcase :      R2 ← add 2, R1
6                 jmp end
7   elsecase :    R3 ← add 4, R1
8                 jmp end
9   end :         R4 ← Φ(R2, R3)
```

Register Allocation: Linear Scan

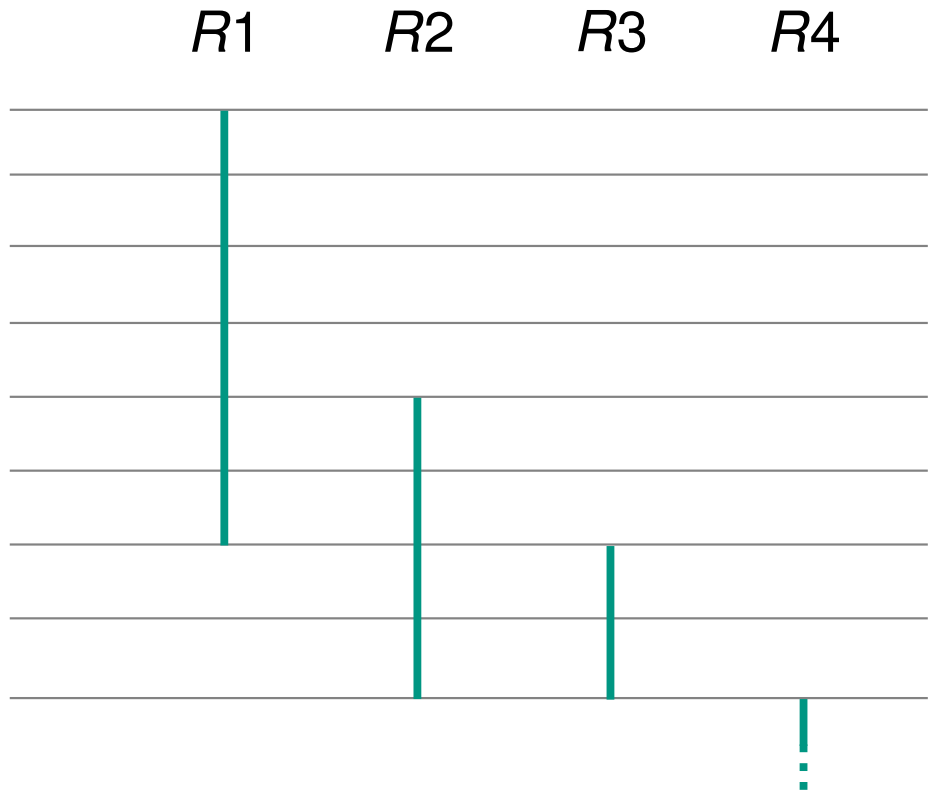
- Precondition: Linear block order with virtual registers

			$R1$	$R2$	$R3$	$R4$
1		$R1 \leftarrow 12$				
2		*condition*				
3		*jnz ifcase*				
4		*jmp elsecase*				
5	*ifcase* :	$R2 \leftarrow$ *add* 2, $R1$				
6		*jmp end*				
7	*elsecase* :	$R3 \leftarrow$ *add* 4, $R1$				
8		*jmp end*				
9	*end* :	$R4 \leftarrow \Phi(R2, R3)$				

Register Allocation: Linear Scan

- Precondition: Linear block order with virtual registers

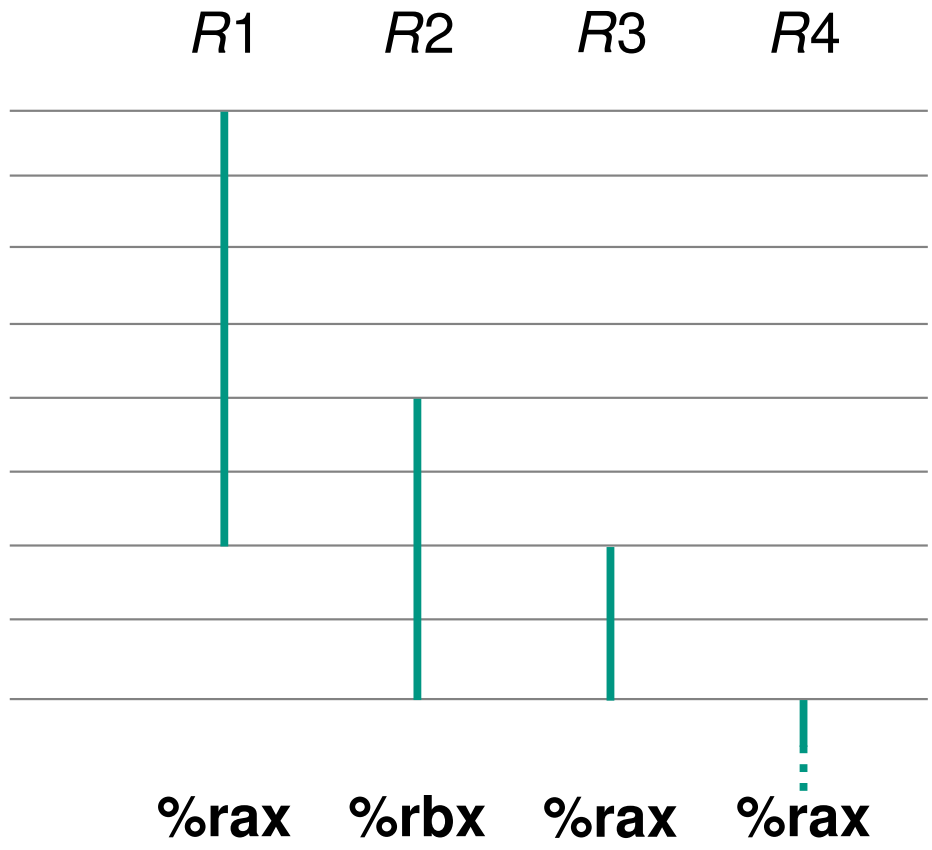
```
1                  R1 ← 12
2                  condition
3                  jnz ifcase
4                  jmp elsecase
5  ifcase :        R2 ← add 2, R1
6                  jmp end
7  elsecase :      R3 ← add 4, R1
8                  jmp end
9  end :           R4 ← Φ(R2, R3)
```



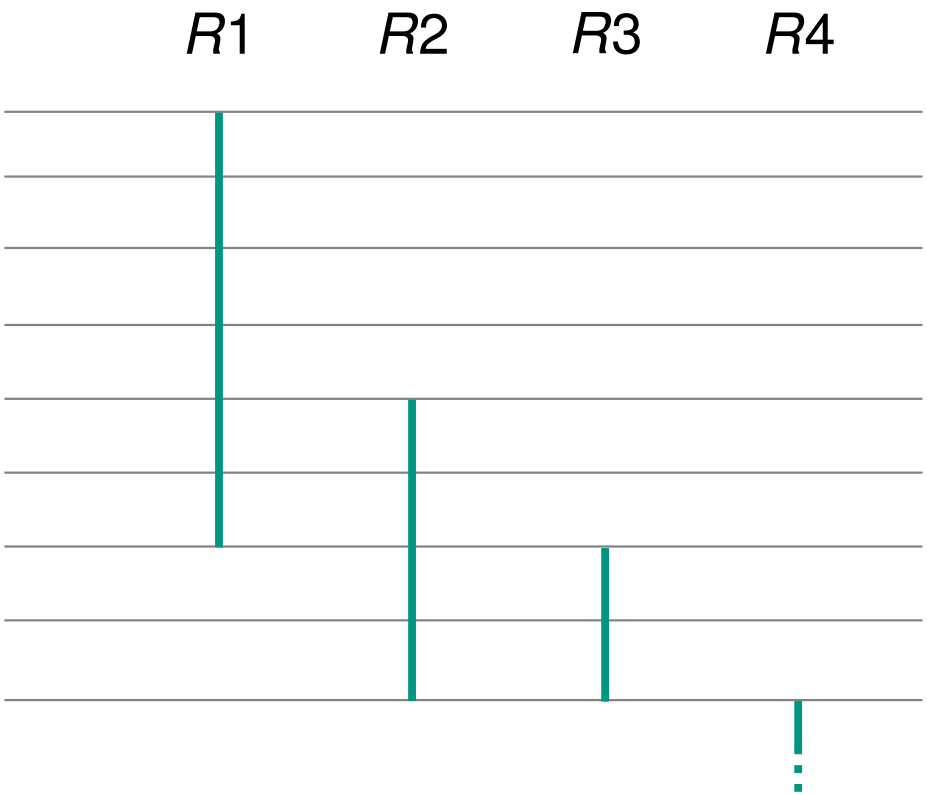
Register Allocation: Linear Scan

- Precondition: Linear block order with virtual registers
- Result: Mapping of Registers to Hardware-Registers

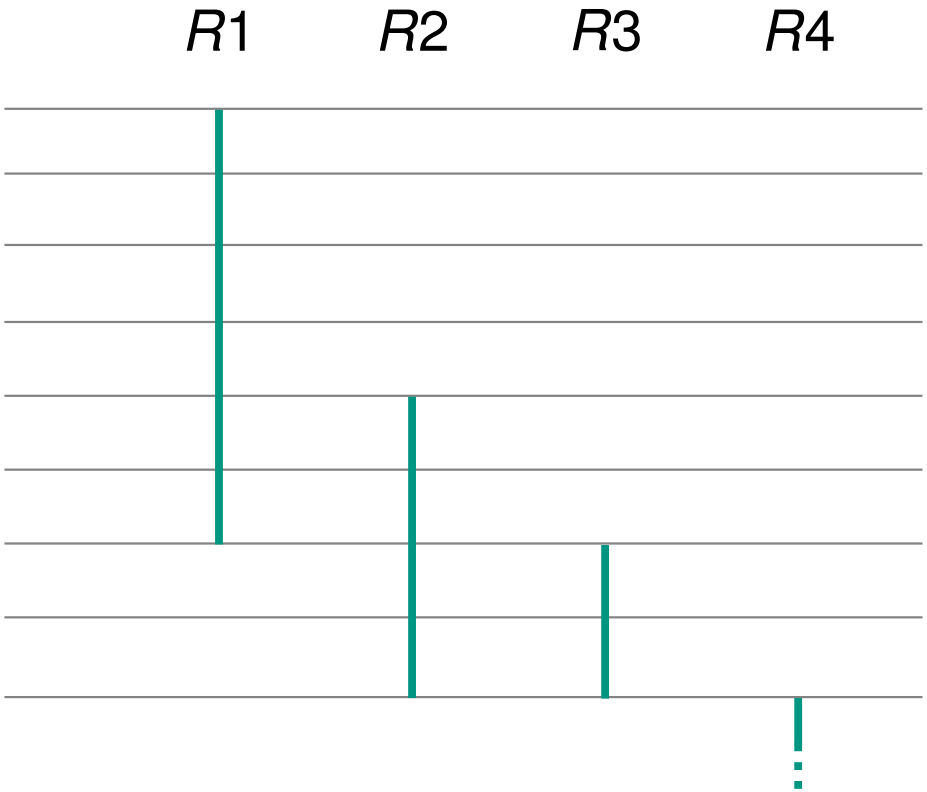
```
1                  R1 ← 12
2                  condition
3                  jnz ifcase
4                  jmp elsecase
5  ifcase :        R2 ← add 2, R1
6                  jmp end
7  elsecase :      R3 ← add 4, R1
8                  jmp end
9  end :           R4 ← Φ(R2, R3)
```

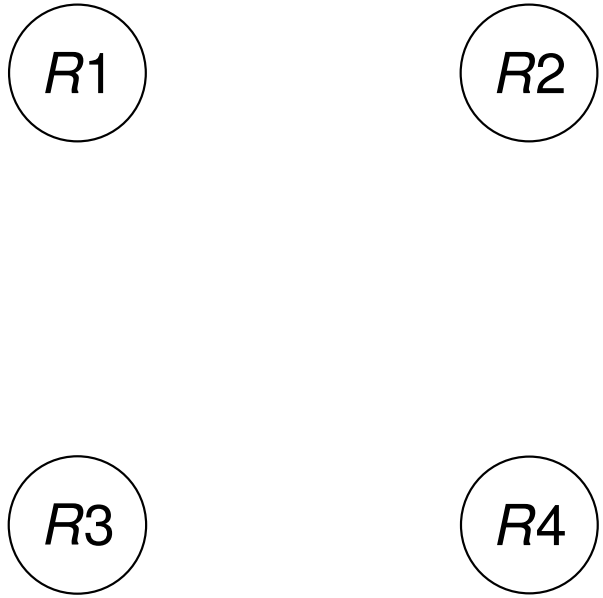


Linear Scan: Extension Graph Coloring

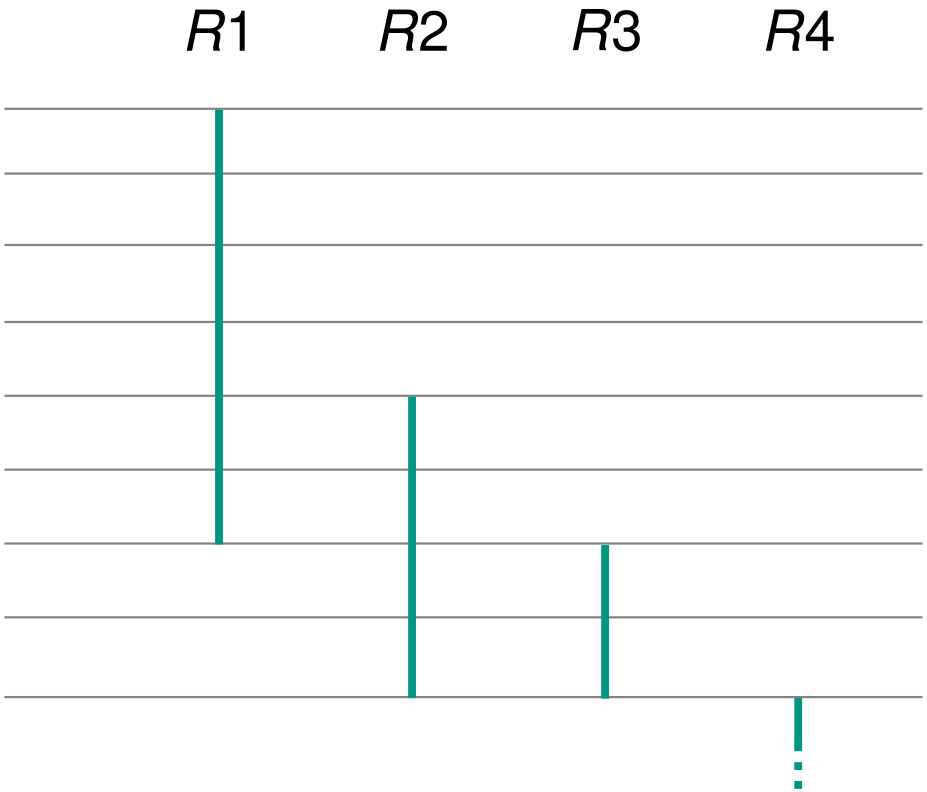


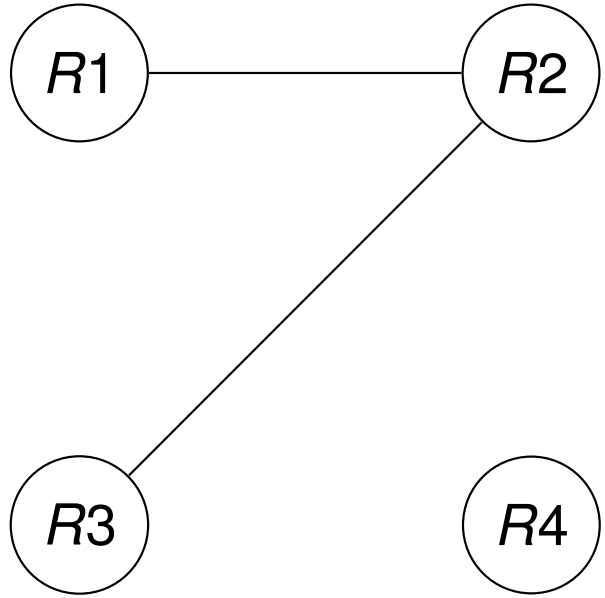
Linear Scan: Extension Graph Coloring



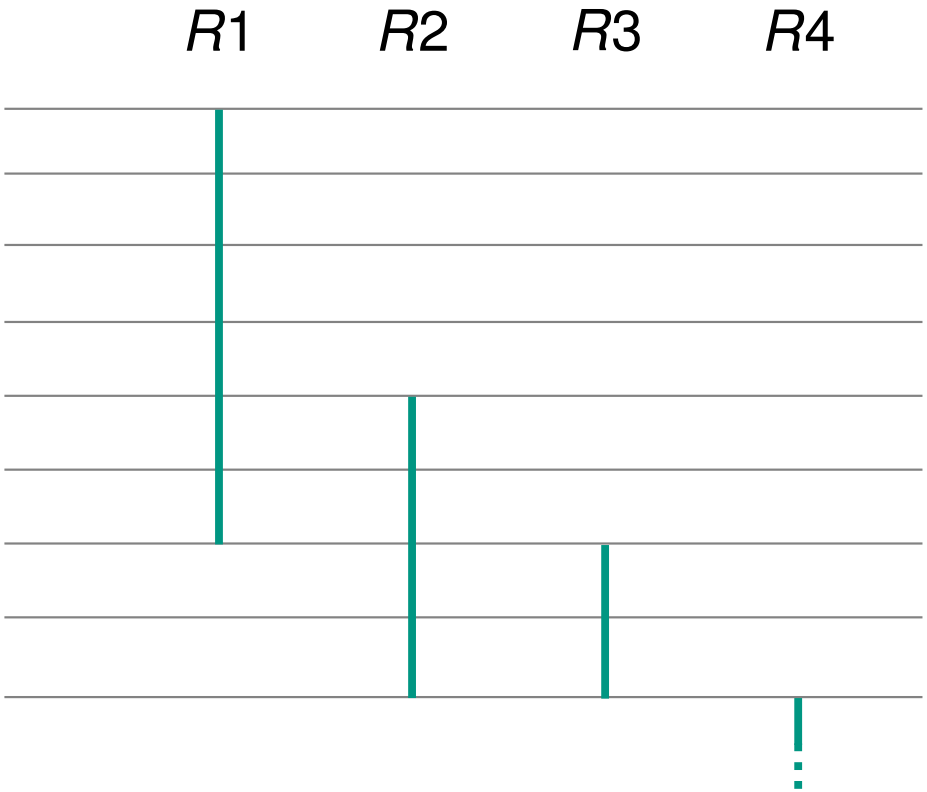


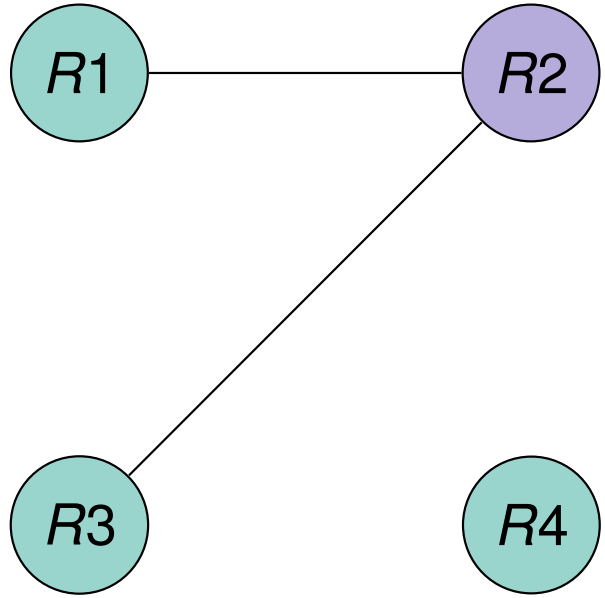
Linear Scan: Extension Graph Coloring



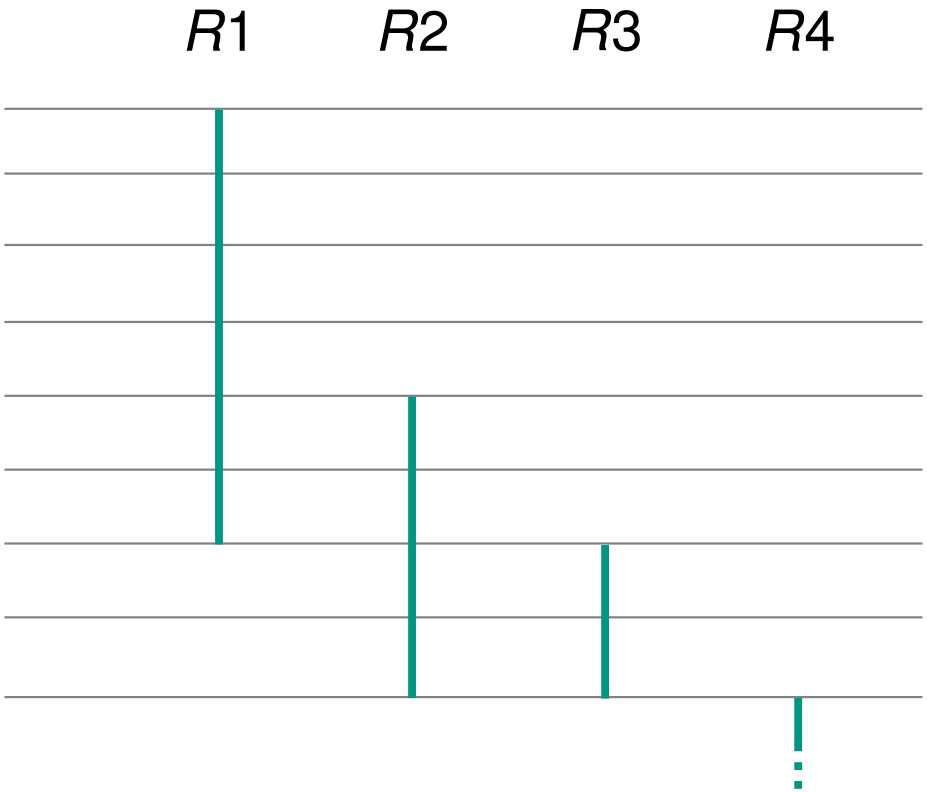


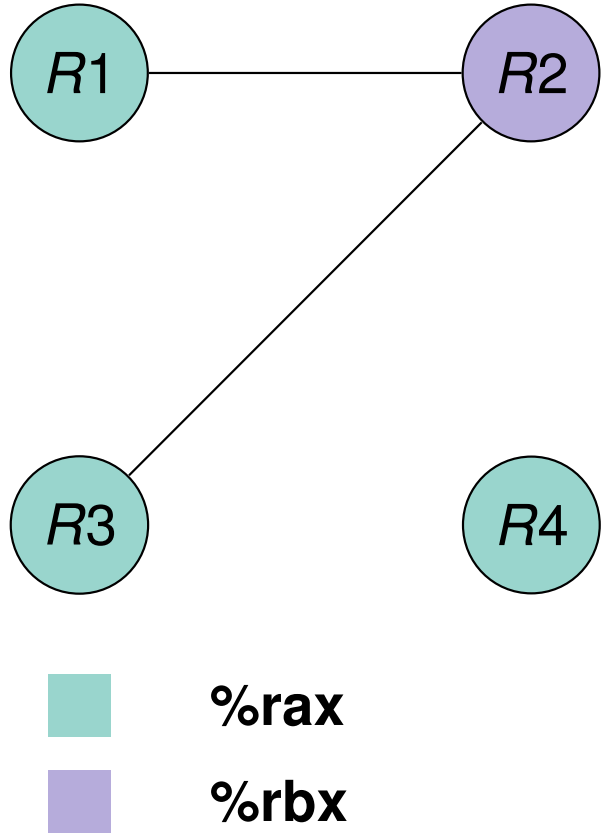
Linear Scan: Extension Graph Coloring





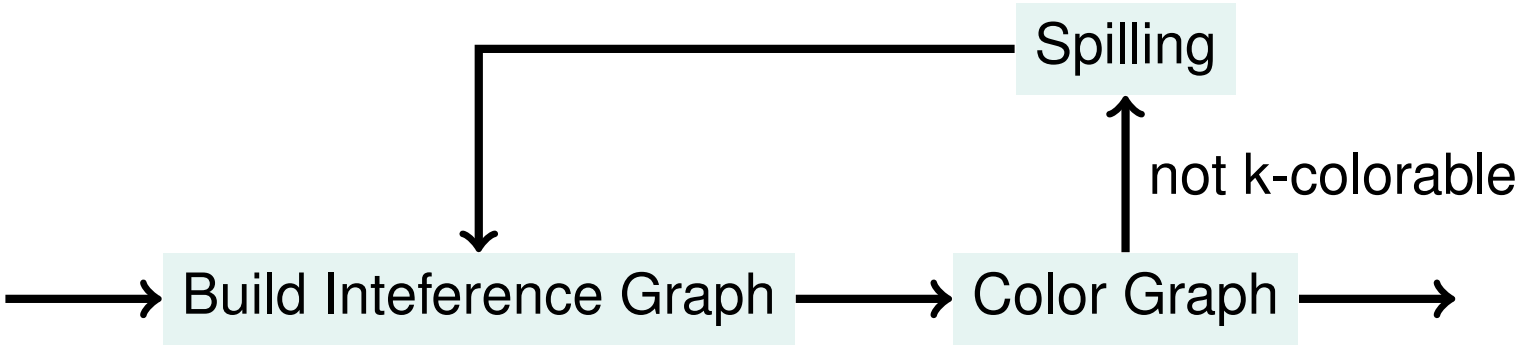
Linear Scan: Extension Graph Coloring





Register Allocation with Graph Coloring

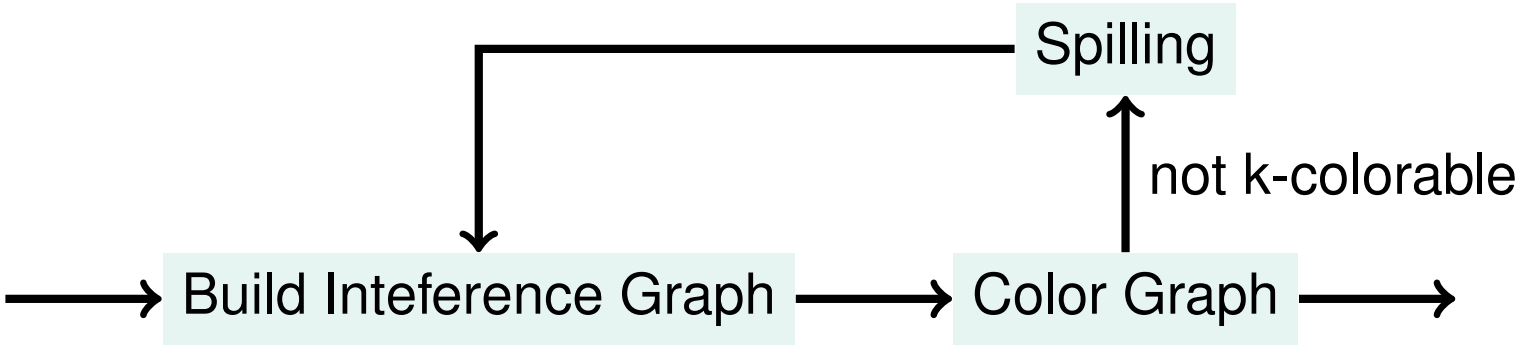
Architecture without SSA



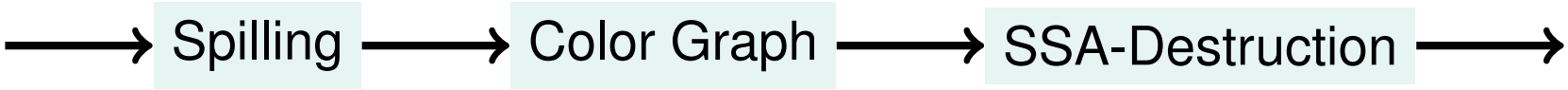
(source: Slides of lecture Compiler, IPD)

Register Allocation with Graph Coloring

Architecture without SSA

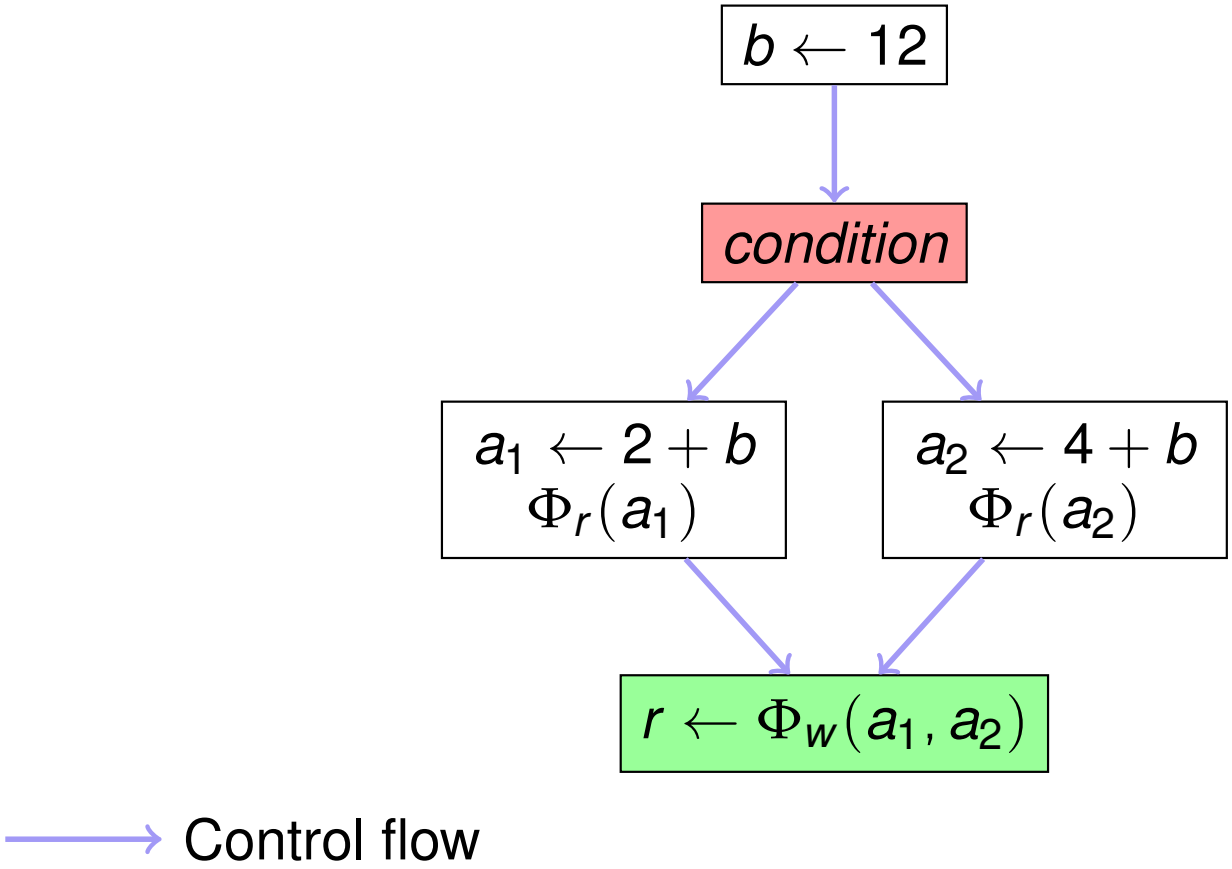


Architecture with SSA

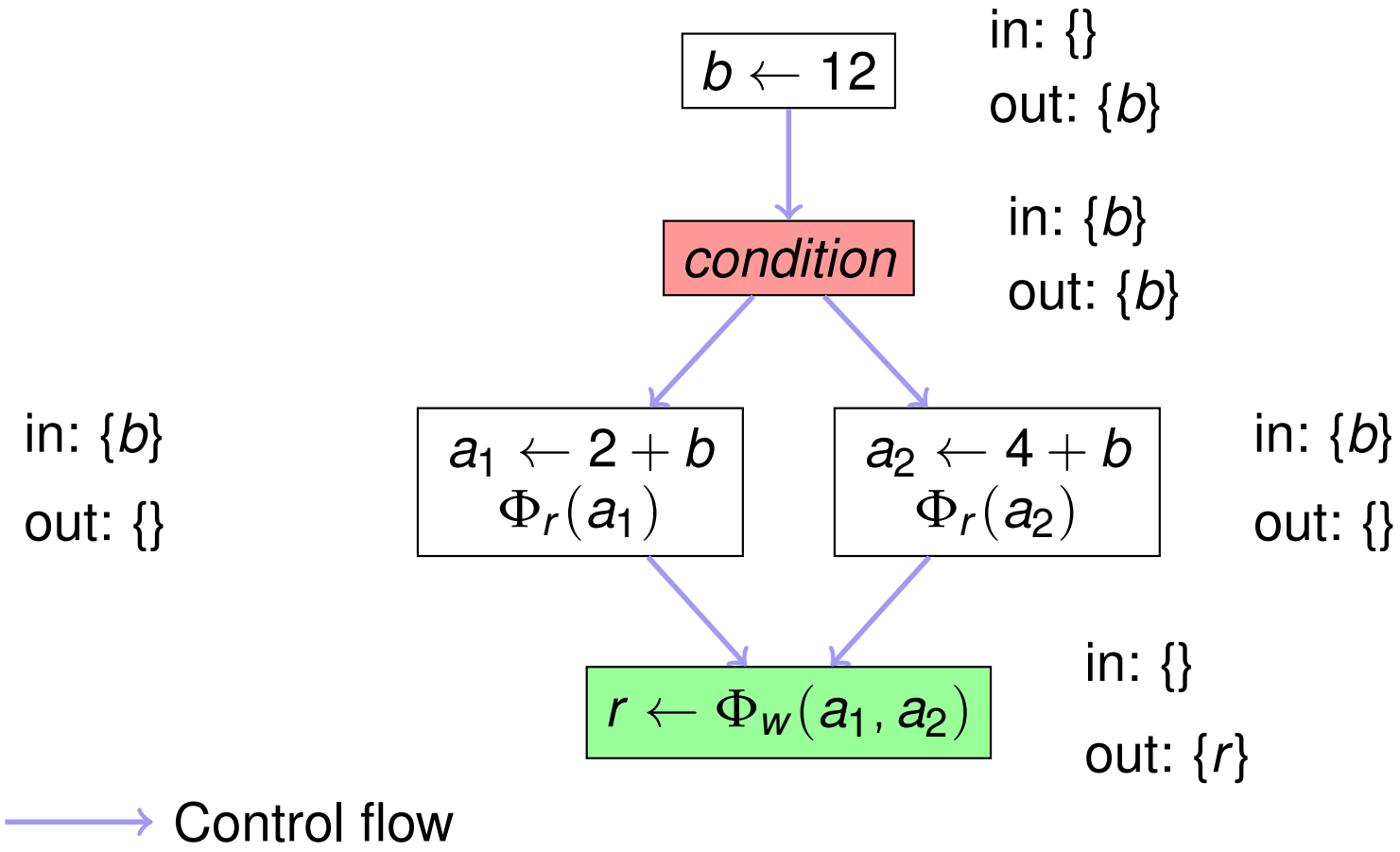


(source: Slides of lecture Compiler, IPD)

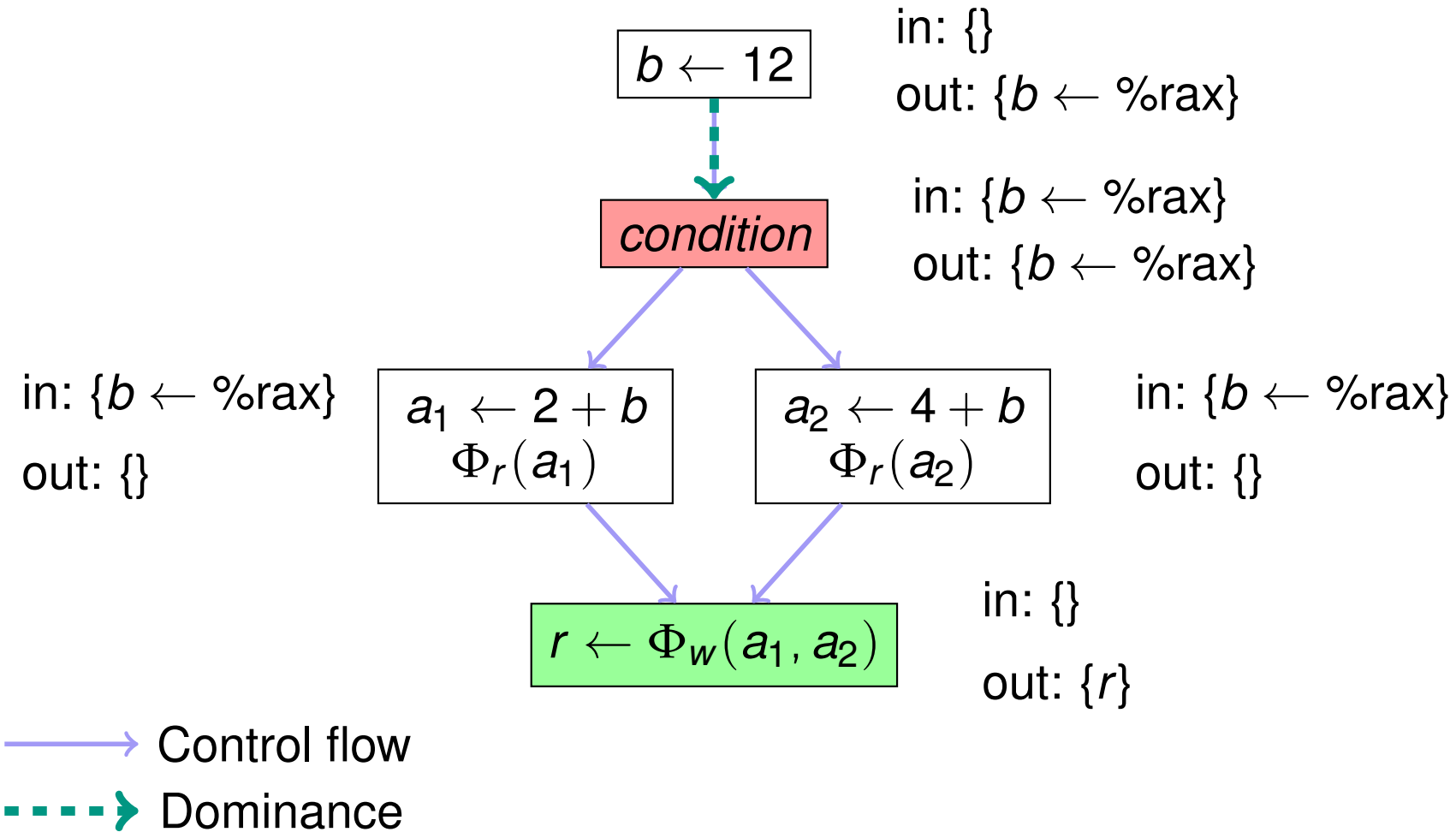
SSA-Based Register Allocation



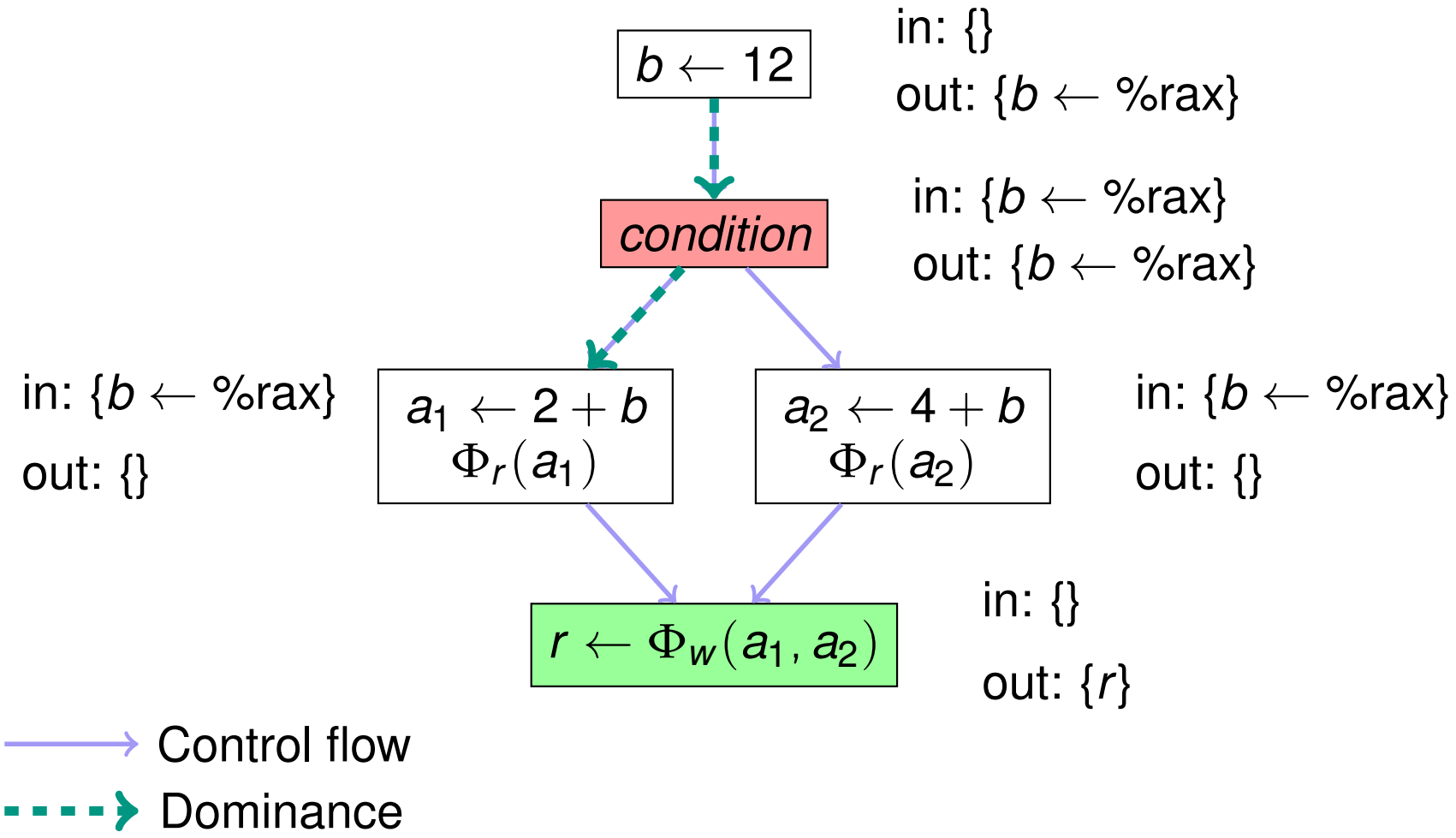
SSA-Based Register Allocation



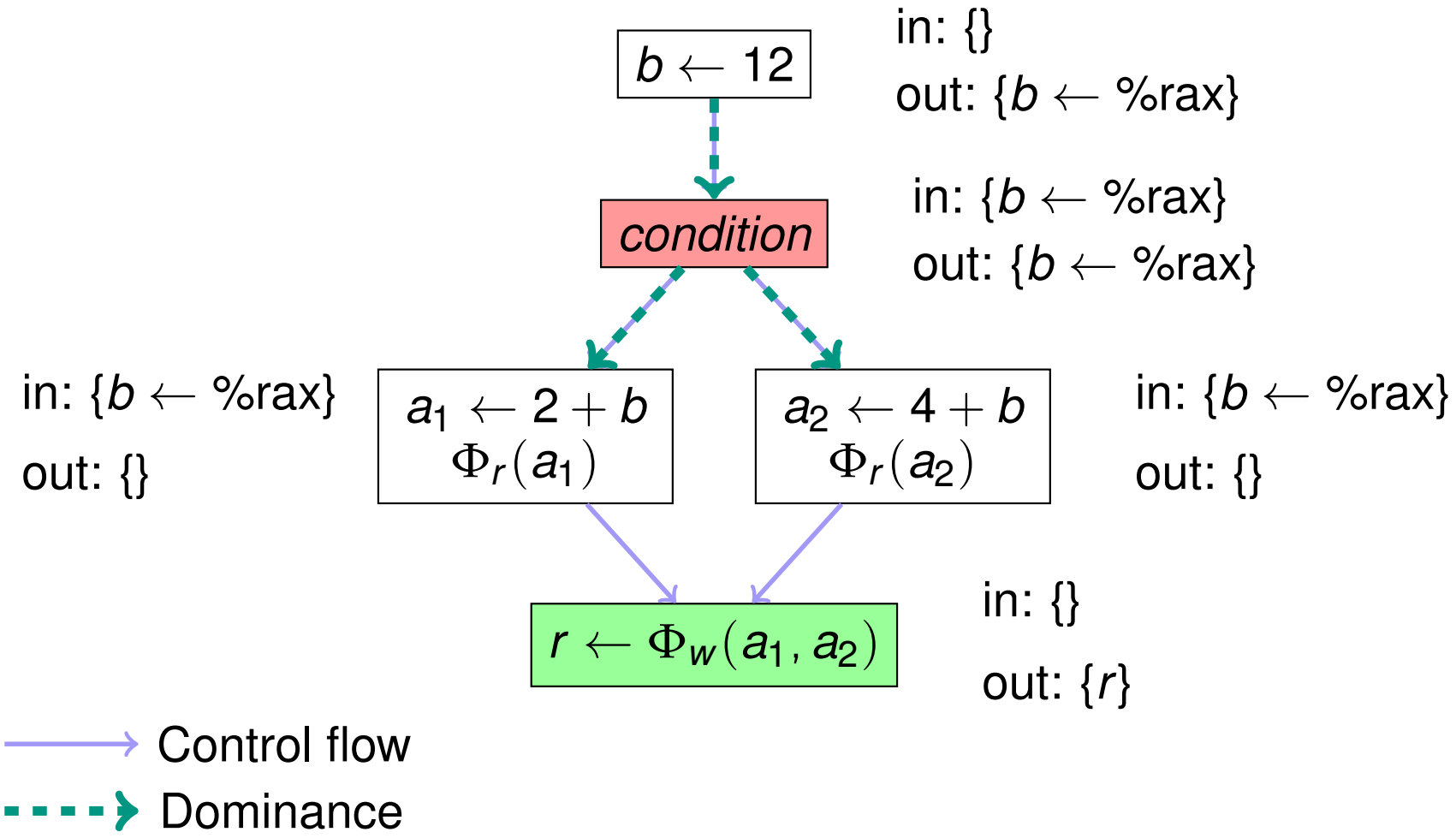
SSA-Based Register Allocation



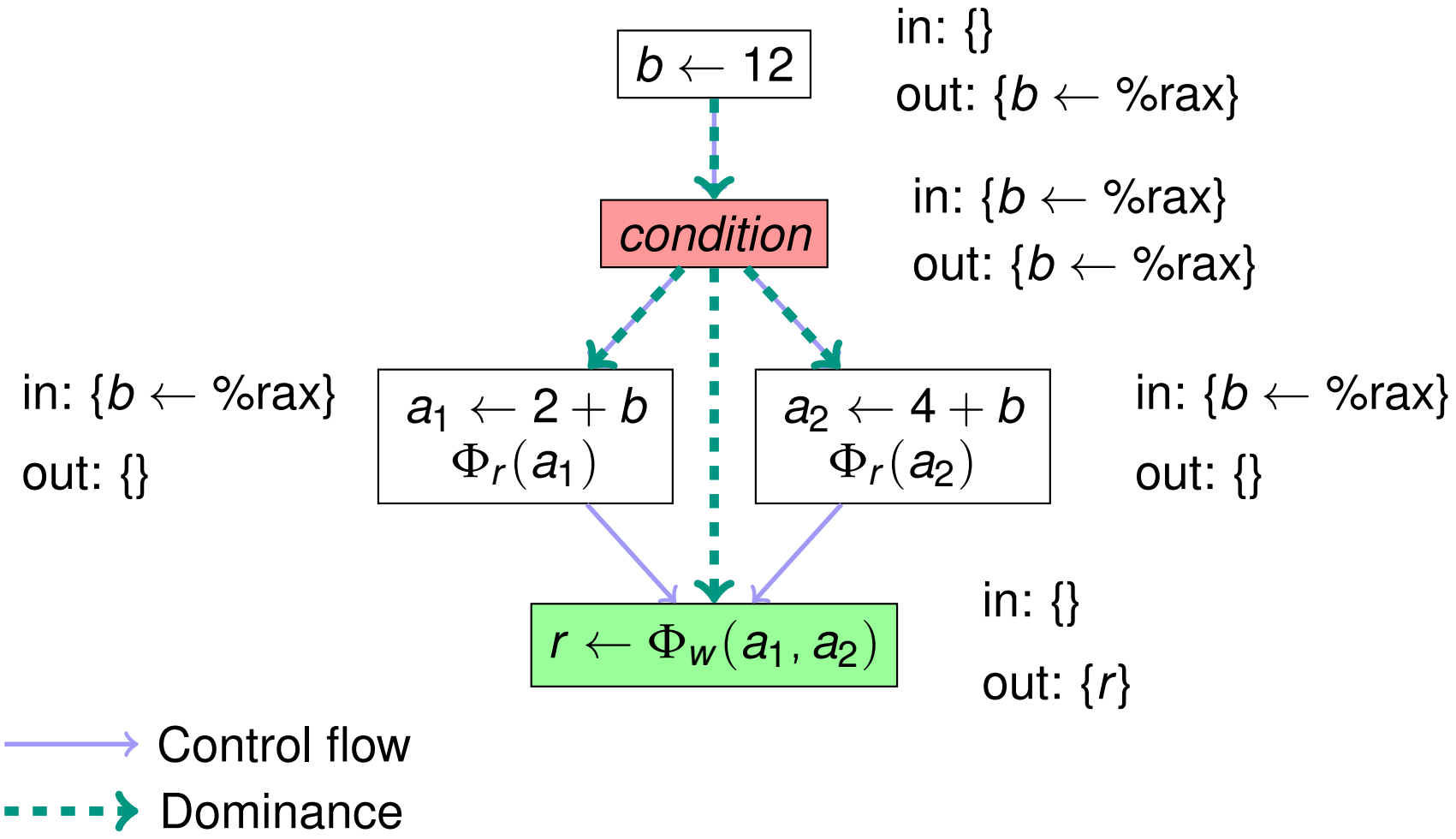
SSA-Based Register Allocation



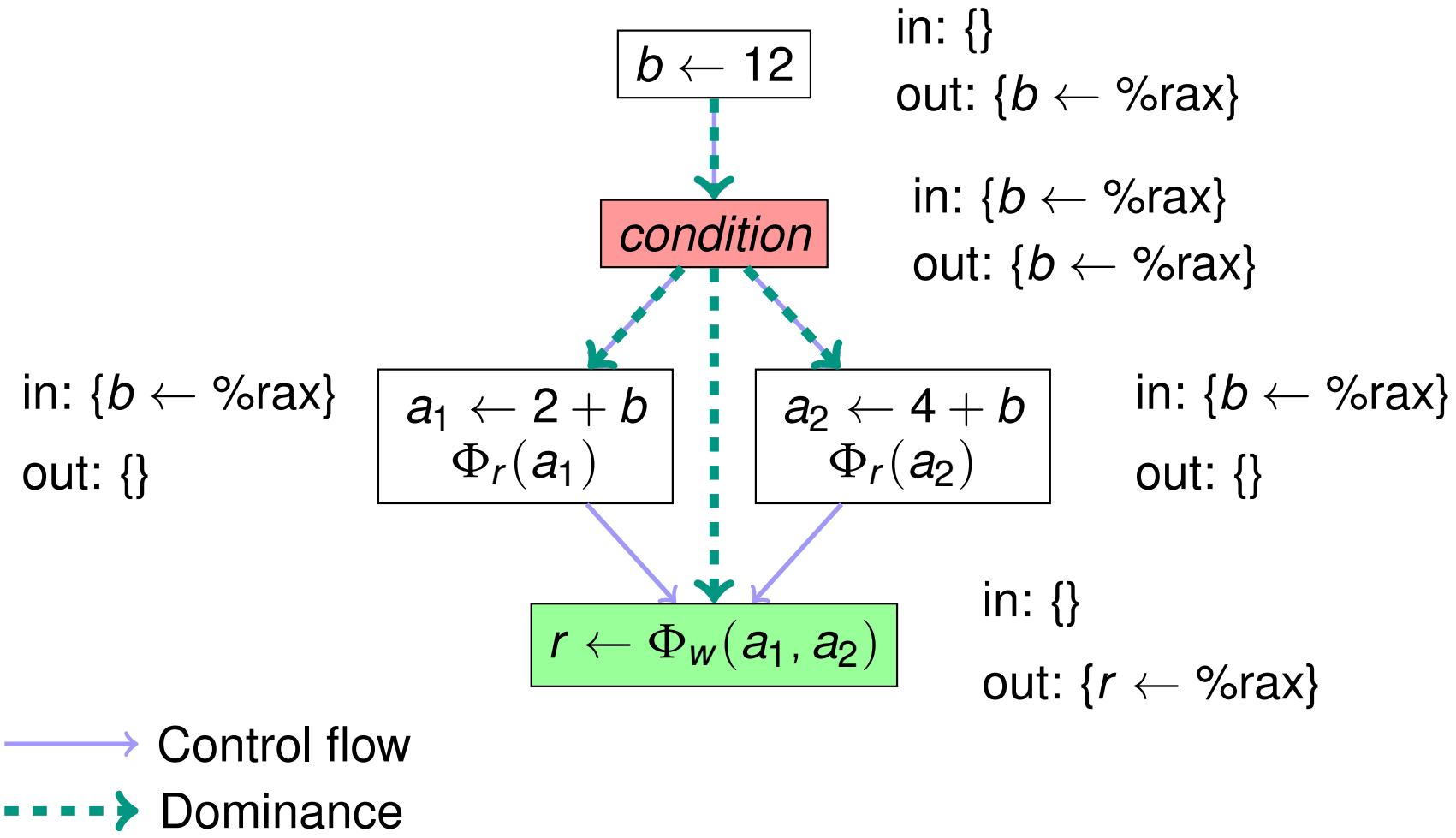
SSA-Based Register Allocation



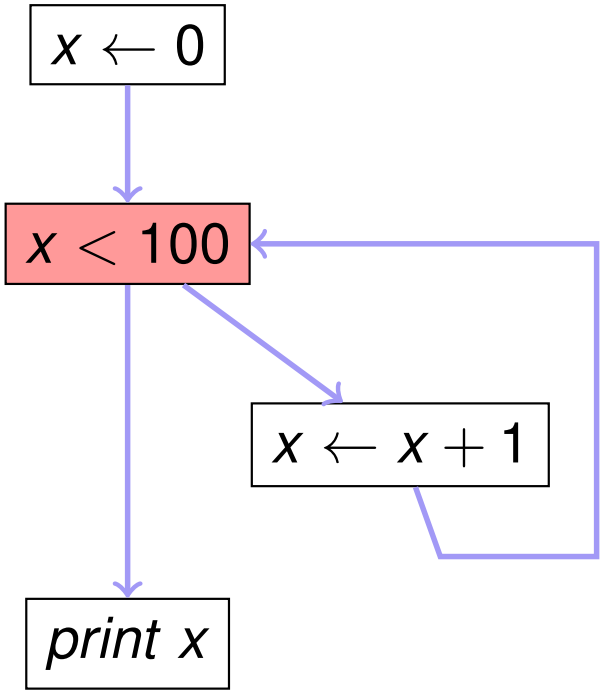
SSA-Based Register Allocation



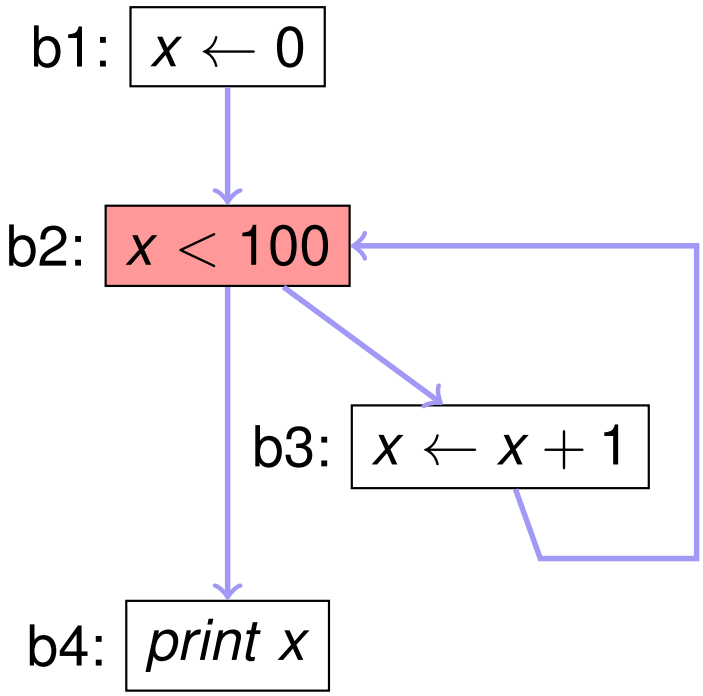
SSA-Based Register Allocation



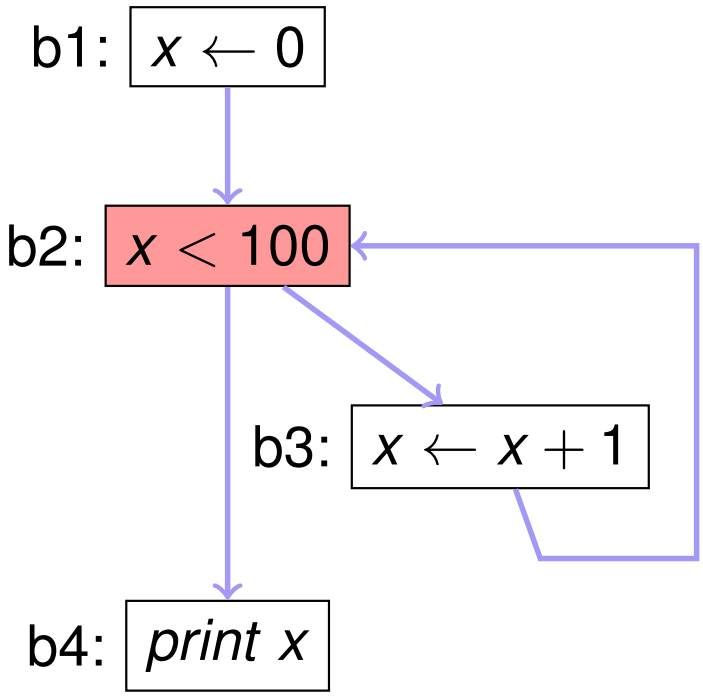
Block Ordering



Block Ordering

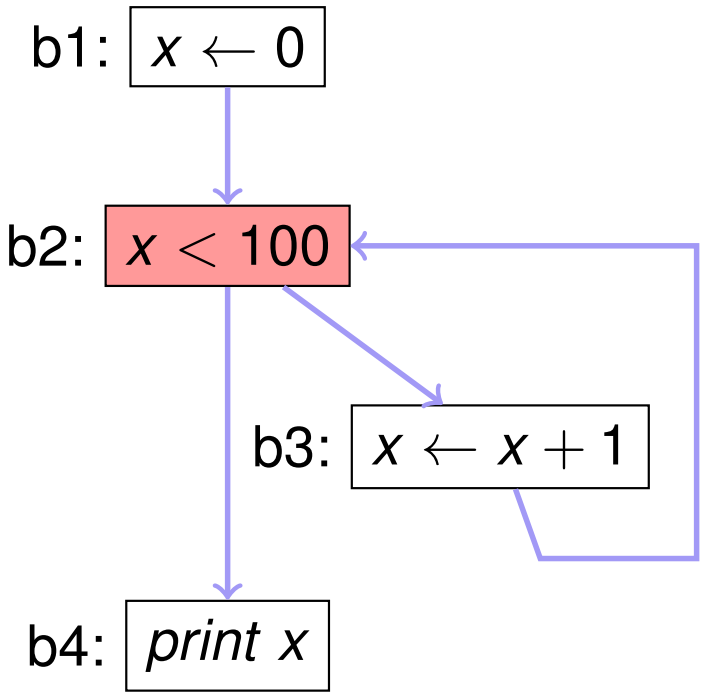


Block Ordering



$$
\begin{array}{lcl}
b1 & : & x \leftarrow 0 \\
 & & jmp\ b2 \\
b2 & : & cmp\ x, 100 \\
 & & jl\ b3 \\
 & & jmp\ b4 \\
b3 & : & x \leftarrow x + 1 \\
 & & jmp\ b2 \\
b4 & : & print\ x
\end{array}
$$

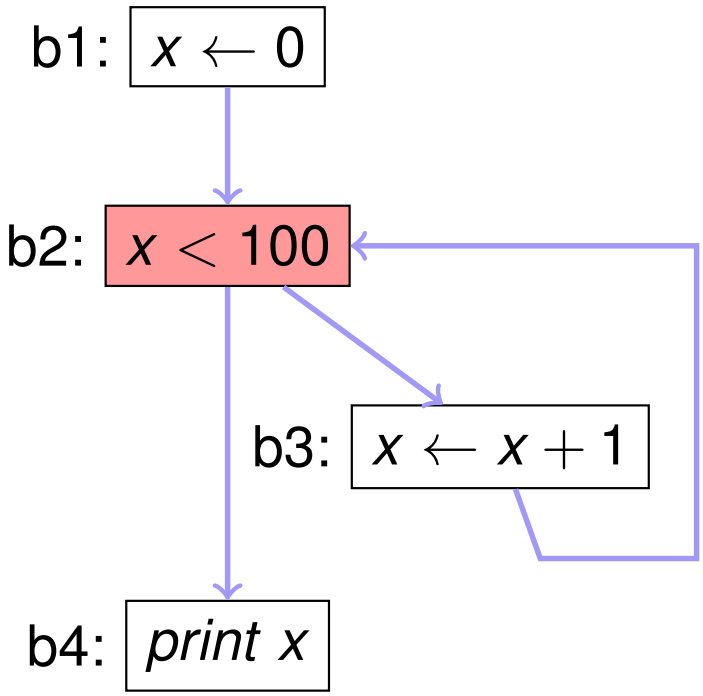
Block Ordering



$$
\begin{array}{lll}
b1 & : & x \leftarrow 0 \\
 & & jmp\ b2 \\
b2 & : & cmp\ x, 100 \\
 & & jl\ b3 \\
 & & jmp\ b4 \\
b3 & : & x \leftarrow x + 1 \\
 & & jmp\ b2 \\
b4 & : & print\ x
\end{array}
$$

- Move condition to the end of the loop

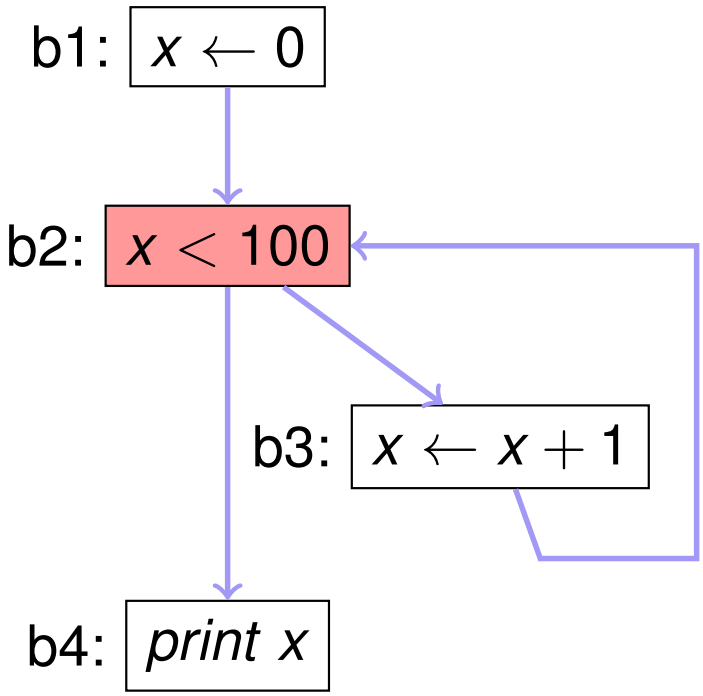
Block Ordering



$$
\begin{array}{lcl}
b1 & : & x \leftarrow 0 \\
 & & jmp\ b2 \\
b3 & : & x \leftarrow x + 1 \\
 & & jmp\ b2 \\
b2 & : & cmp\ x, 100 \\
 & & jl\ b3 \\
 & & jmp\ b4 \\
b4 & : & print\ x
\end{array}
$$

- Move condition to the end of the loop

Block Ordering



$$
\begin{array}{lcl}
b1 & : & x \leftarrow 0 \\
 & & jmp\ b2 \\
b3 & : & x \leftarrow x + 1 \\
 & & \sout{jmp\ b2} \\
b2 & : & cmp\ x, 100 \\
 & & jl\ b3 \\
 & & \sout{jmp\ b4} \\
b4 & : & print\ x
\end{array}
$$

- Move condition to the end of the loop
- Peephole-Optimizations: Eliminate unneccessary jumps

Implemented Optimizations

- Common Subexpression Elimination
- Constant Folding
- Control Flow Optimiziation
- Load Store Optimiziation
- Local Optimization
- Loop Fusion
- Loop Invariant Code Motion
- Loop Unrolling
- Method Inlining
- Normalization
- Peephole Optimizations
- Procedure specialization and removal of side-effect free calls
- Strength Reduction

Facts and Numbers

- Lines of code: 30765
- Classes: 201
- Active days: 107 (93.86%)
- Commits: 1973
  - average 18.4 commits per active day
  - 17.3 per all days
- Craziest commit time: 5am
- Hours spend: way to much
- Longest work streak: 36/48 hours

Conclusion

- Firm makes life easier (and more complicated)
  - Especially on Windows
- Testing is important
- GIT is nice, but using it can be tricky for beginners
- Team coordination essential

Conclusion

- Firm makes life easier (and more complicated)
  - Especially on Windows
- Testing is important
- GIT is nice, but using it can be tricky for beginners
- Team coordination essential

Figure: Source: www.real-drive.de

Thank you for your attention!

Questions?