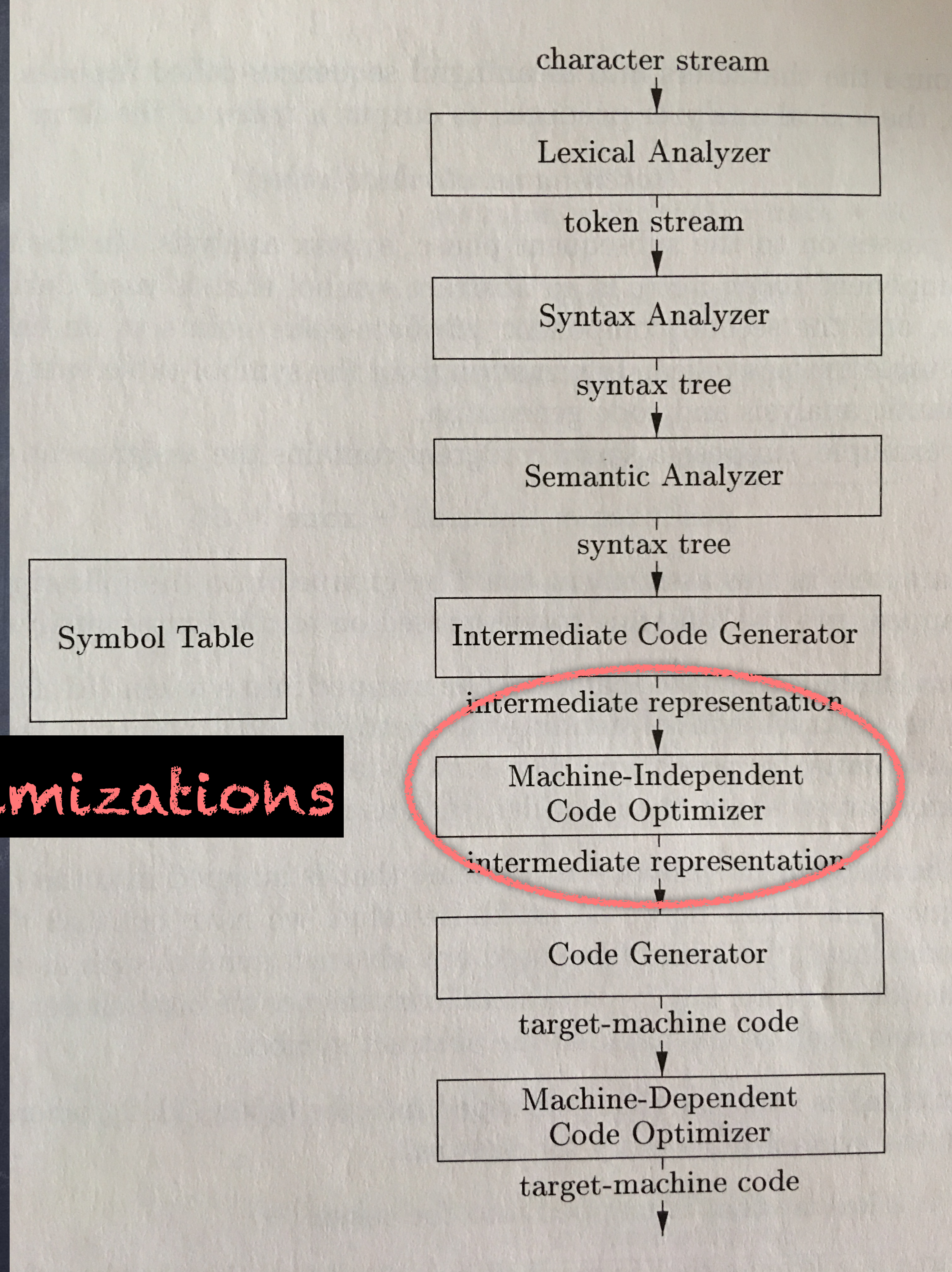
CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

Phases of a compiler



Optimizations

Figure 1.6, page 5 of text

9.2.3 Data-flow schemas on basic blocks

"...data-flow equations usually do not have a unique solution. Our goal is to find the most 'precise' solution that satisfies the two sets of constraints: control-flow and transfer constraints. That is, we need a solution that encourages valid code improvements, but does not justify unsafe transformations..."

[p. 601]

9.2.4 Reaching definitions

"A definition d reaches a point p if there is a path from the point immediately following d to p, such that d is not 'killed' along that path." [p. 601]

"We kill a definition of a variable x if there is any other definition of x anywhere along the path." [p. 601]

9.2.4 Reaching definitions

"A definition of a variable x is a statement that assigns, or may assign, a value to x."

What is meant by "may assign"?

9.2.4 Reaching definitions

"Procedure parameters, array accesses, and indirect references all may have aliases, and it is not easy to tell if a statement is referring to a particular variable x." [p. 601]

"Program analysis must be conservative" [p. 601]

Transfer equations for reaching definitions

For this definition:

$$d: u = v + w$$

σ is a data-flow value

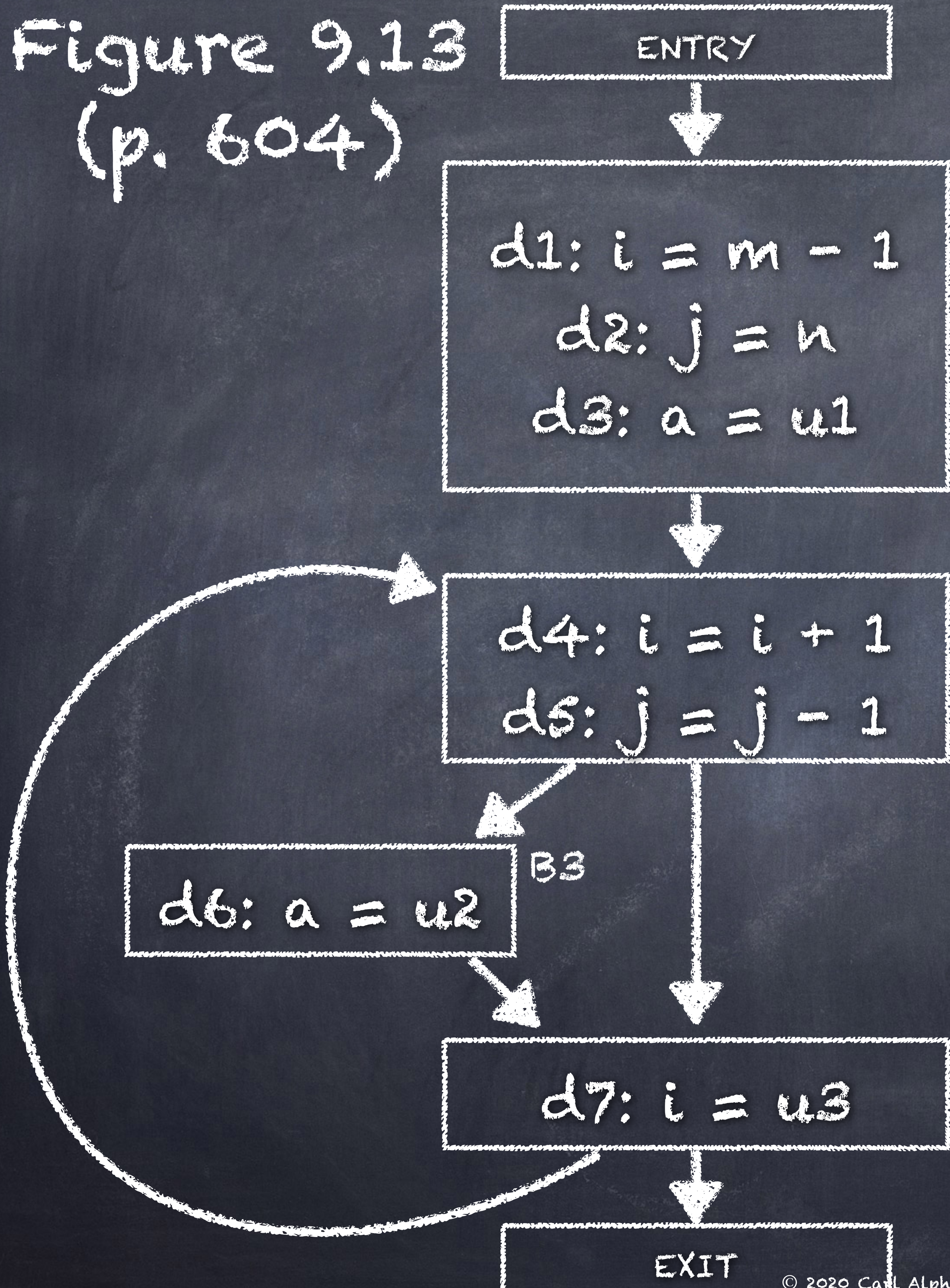
The transfer equation is:

$$f_d(\sigma) = gen_d \cup (\sigma - kill_d)$$

where $gen_d = \{d\}$. $kill_d$ is the set of all other definitions of u in the program

The argument of a transfer function is a data-flow value, which "represents an abstraction of the set of all possible program states that can be observed for that point." [p. 599] Recall too that a program state consists of all the variables in the program along with their current values.

Figure 9.13 (p. 604)



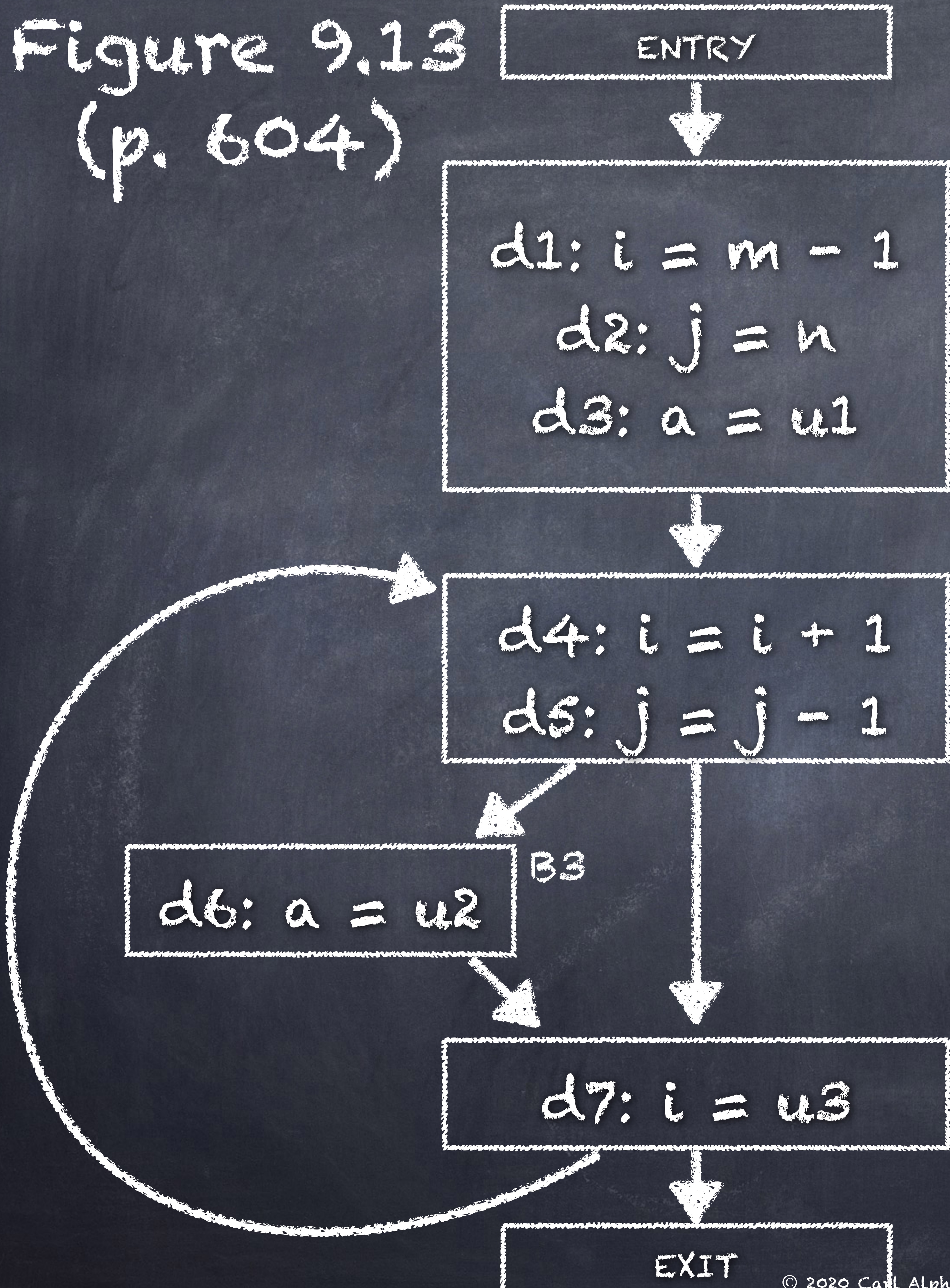
$gen_{B1} = \{ ? \}$
$kill_{B1} = \{ ? \}$

$gen_{B2} = \{ ? \}$
$kill_{B2} = \{ ? \}$

$gen_{B3} = \{ ? \}$
$kill_{B3} = \{ ? \}$

$gen_{B4} = \{ ? \}$
$kill_{B4} = \{ ? \}$

Figure 9.13 (p. 604)



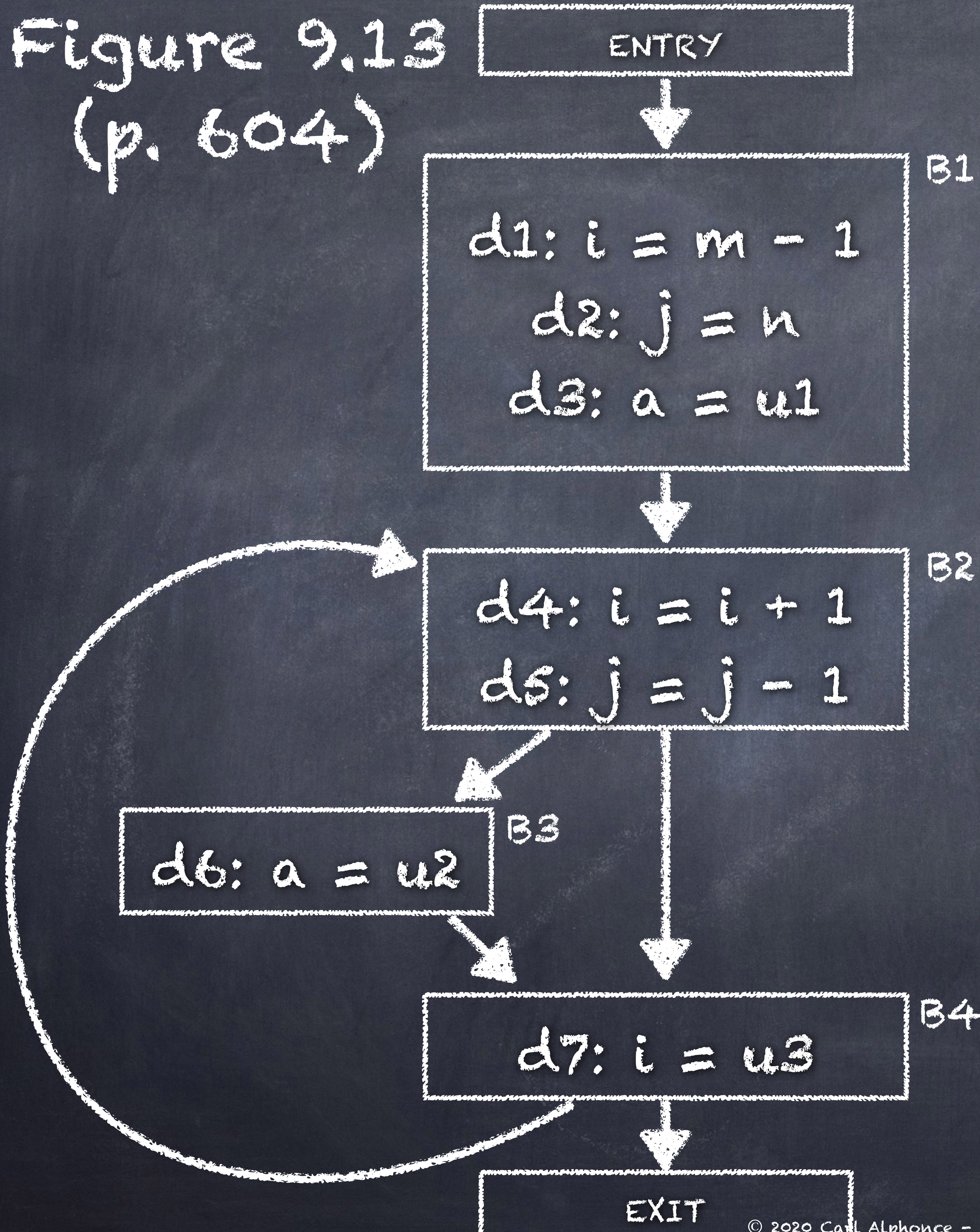
gen_{B1} = { d1, d2, d3 }
$kill_{B1}$ = { ? }

gen_{B2} = { ? }
$kill_{B2}$ = { ? }

gen_{B3} = { ? }
$kill_{B3}$ = { ? }

gen_{B4} = { ? }
$kill_{B4}$ = { ? }

Figure 9.13 (p. 604)



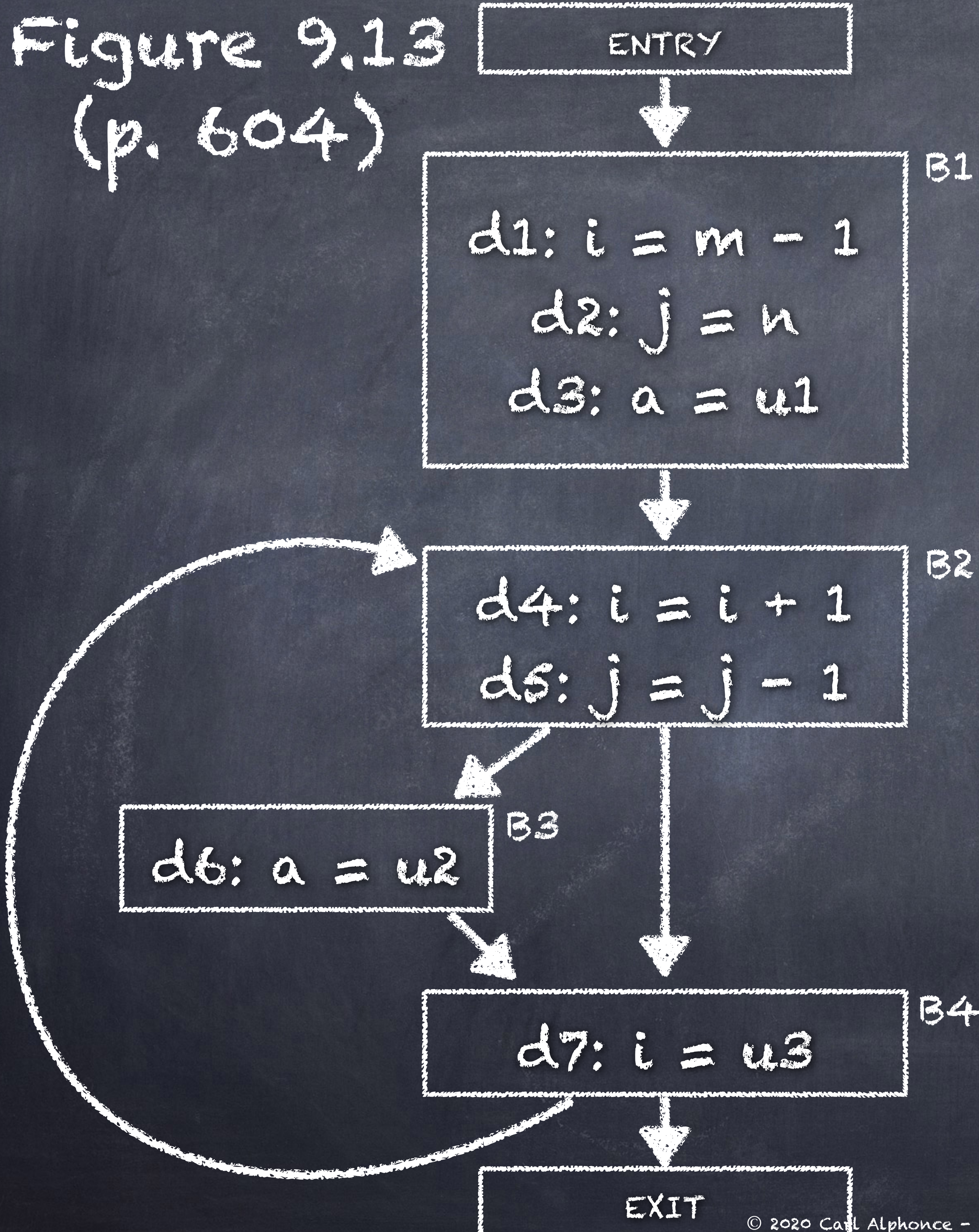
gen_{B1} = { d1, d2, d3 }
$kill_{B1}$ = { d4, d5, d6, d7 }

gen_{B2} = { ? }
$kill_{B2}$ = { ? }

gen_{B3} = { ? }
$kill_{B3}$ = { ? }

gen_{B4} = { ? }
$kill_{B4}$ = { ? }

Figure 9.13 (p. 604)



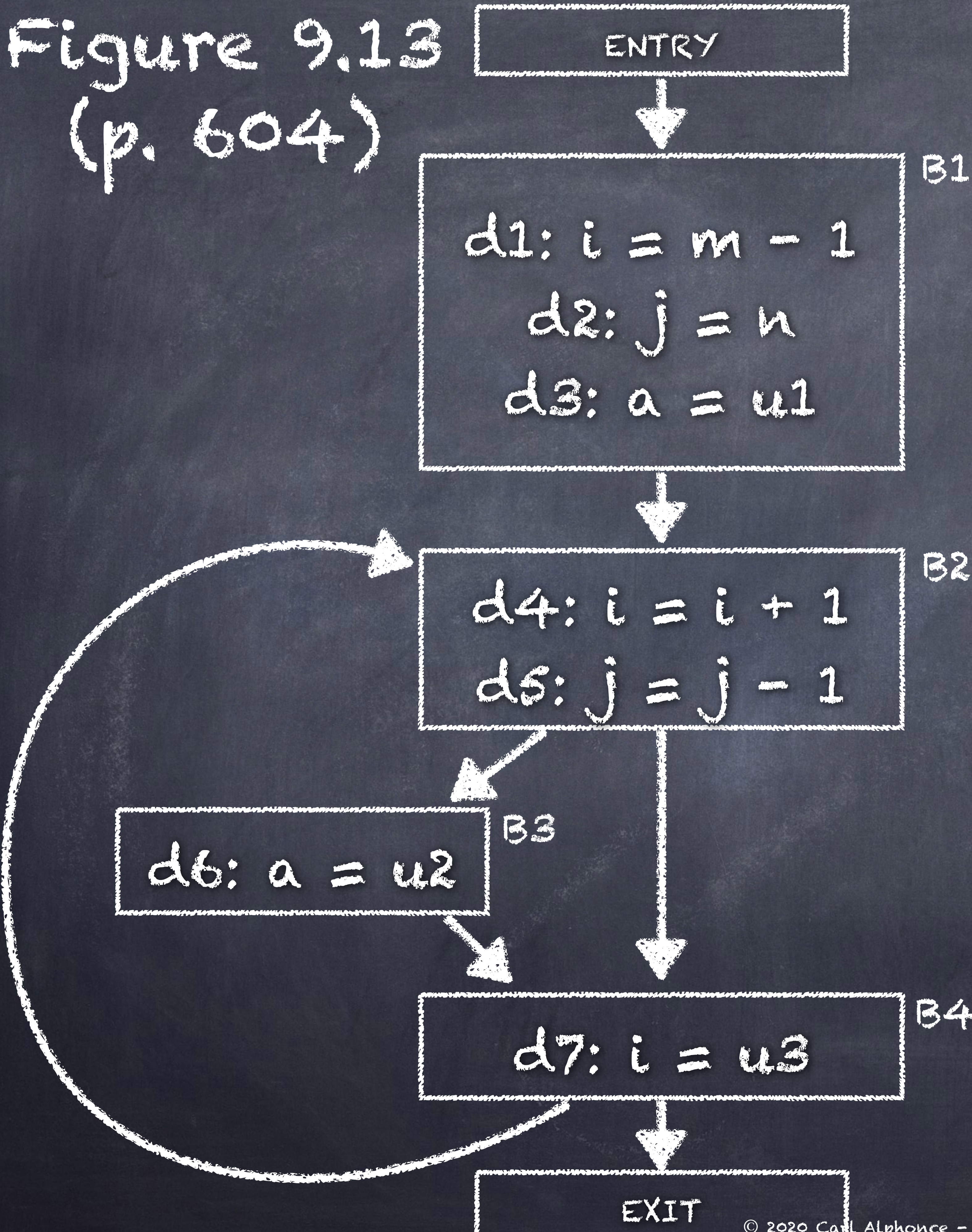
gen_{B1} = { d1, d2, d3 }
$kill_{B1}$ = { d4, d5, d6, d7 }

Q: Why kill d4 - d7 here, since they are not on a path to B1?

gen_{B3} = { ? }
$kill_{B3}$ = { ? }

gen_{B4} = { ? }
$kill_{B4}$ = { ? }

Figure 9.13 (p. 604)



gen_{B1} = { d1, d2, d3 }

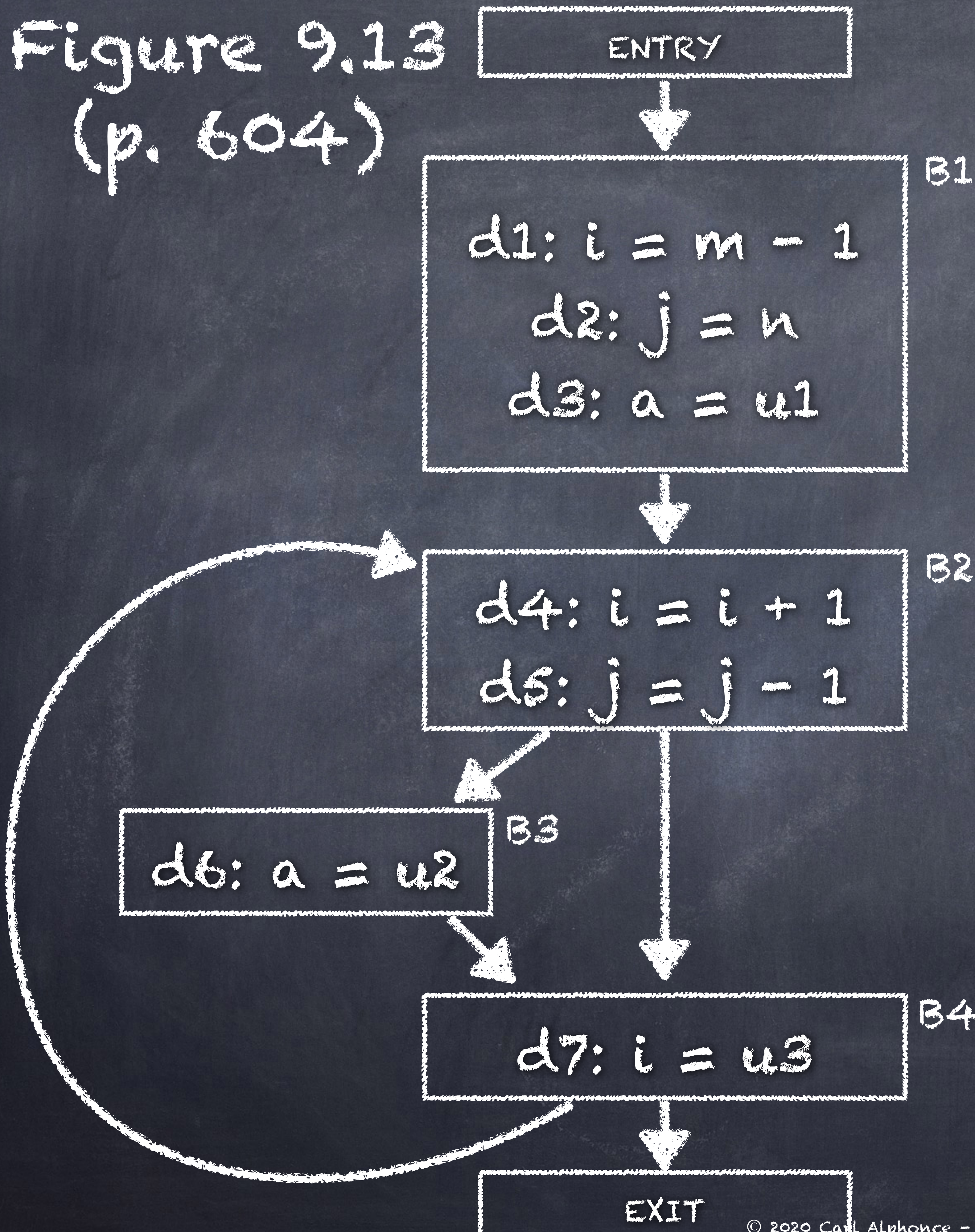
$kill_{B1}$ = { d4, d5, d6, d7 }

Q: Why kill d4 - d7 here, since they are not on a path to B1?

A: Here we are looking just at this block, and not trying to account for flow between blocks.

Inter-block flow is taken into account later.

Figure 9.13 (p. 604)



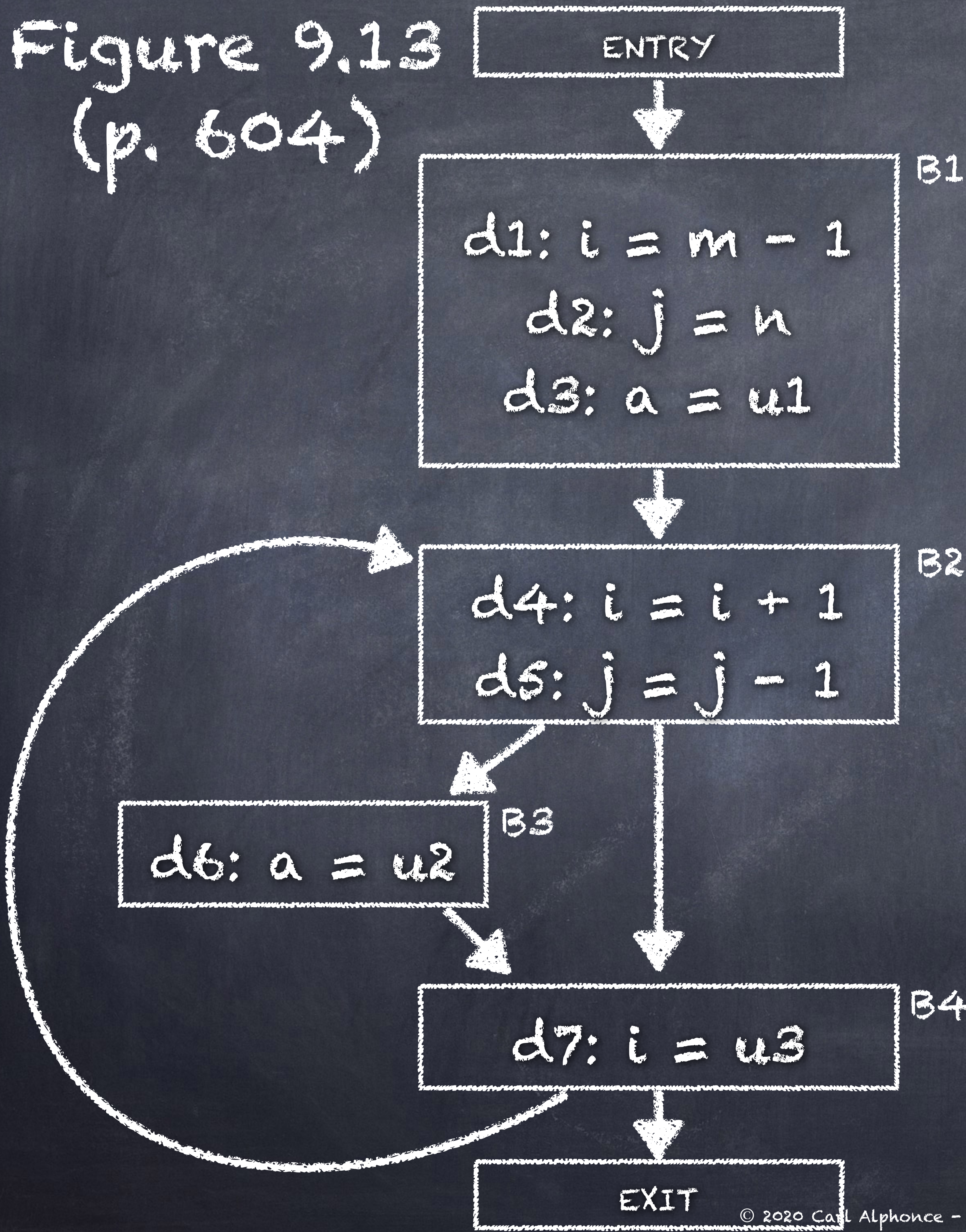
$gen_{B1} = \{ d1, d2, d3 \}$
$kill_{B1} = \{ d4, d5, d6, d7 \}$

$gen_{B2} = \{ ? \}$
$kill_{B2} = \{ ? \}$

$gen_{B3} = \{ ? \}$
$kill_{B3} = \{ ? \}$

$gen_{B4} = \{ ? \}$
$kill_{B4} = \{ ? \}$

Figure 9.13 (p. 604)



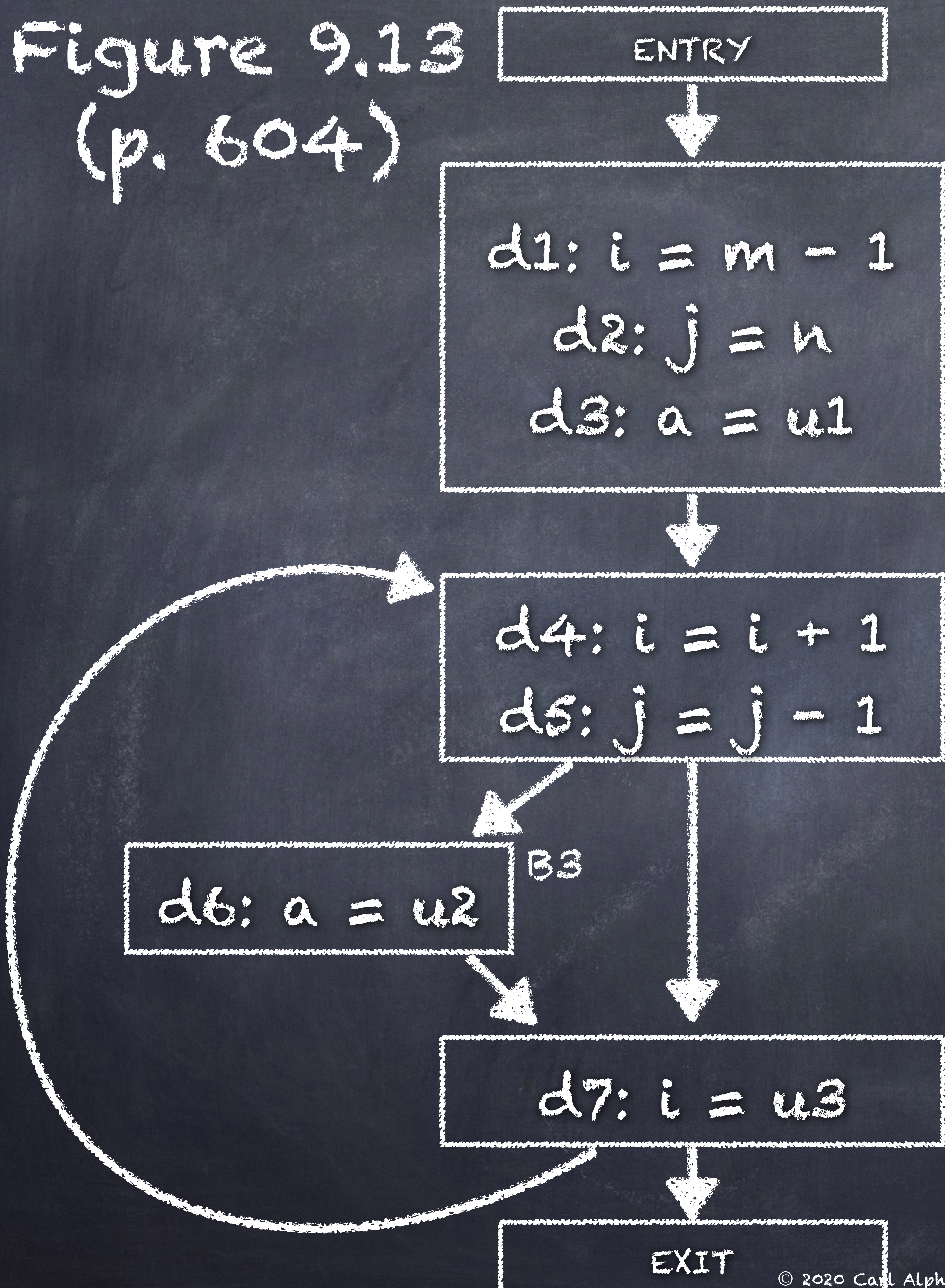
$gen_{B1} = \{ d1, d2, d3 \}$
$kill_{B1} = \{ d4, d5, d6, d7 \}$

$gen_{B2} = \{ d4, d5 \}$
$kill_{B2} = \{ ? \}$

$gen_{B3} = \{ ? \}$
$kill_{B3} = \{ ? \}$

$gen_{B4} = \{ ? \}$
$kill_{B4} = \{ ? \}$

Figure 9.13 (p. 604)



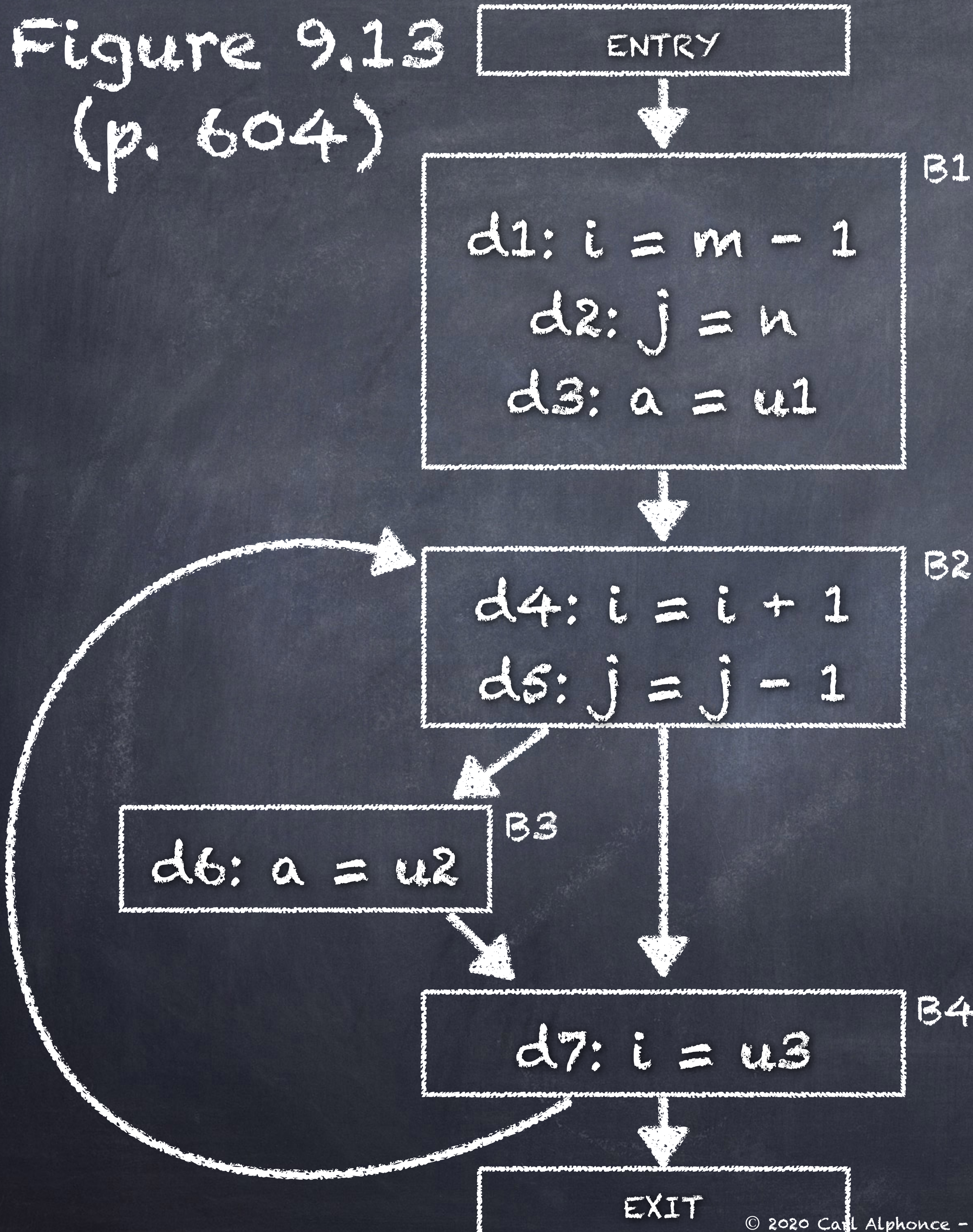
$gen_{B1} = \{ d1, d2, d3 \}$
$kill_{B1} = \{ d4, d5, d6, d7 \}$

$gen_{B2} = \{ d4, d5 \}$
$kill_{B2} = \{ d1, d2, d7 \}$

$gen_{B3} = \{ ? \}$
$kill_{B3} = \{ ? \}$

$gen_{B4} = \{ ? \}$
$kill_{B4} = \{ ? \}$

Figure 9.13 (p. 604)



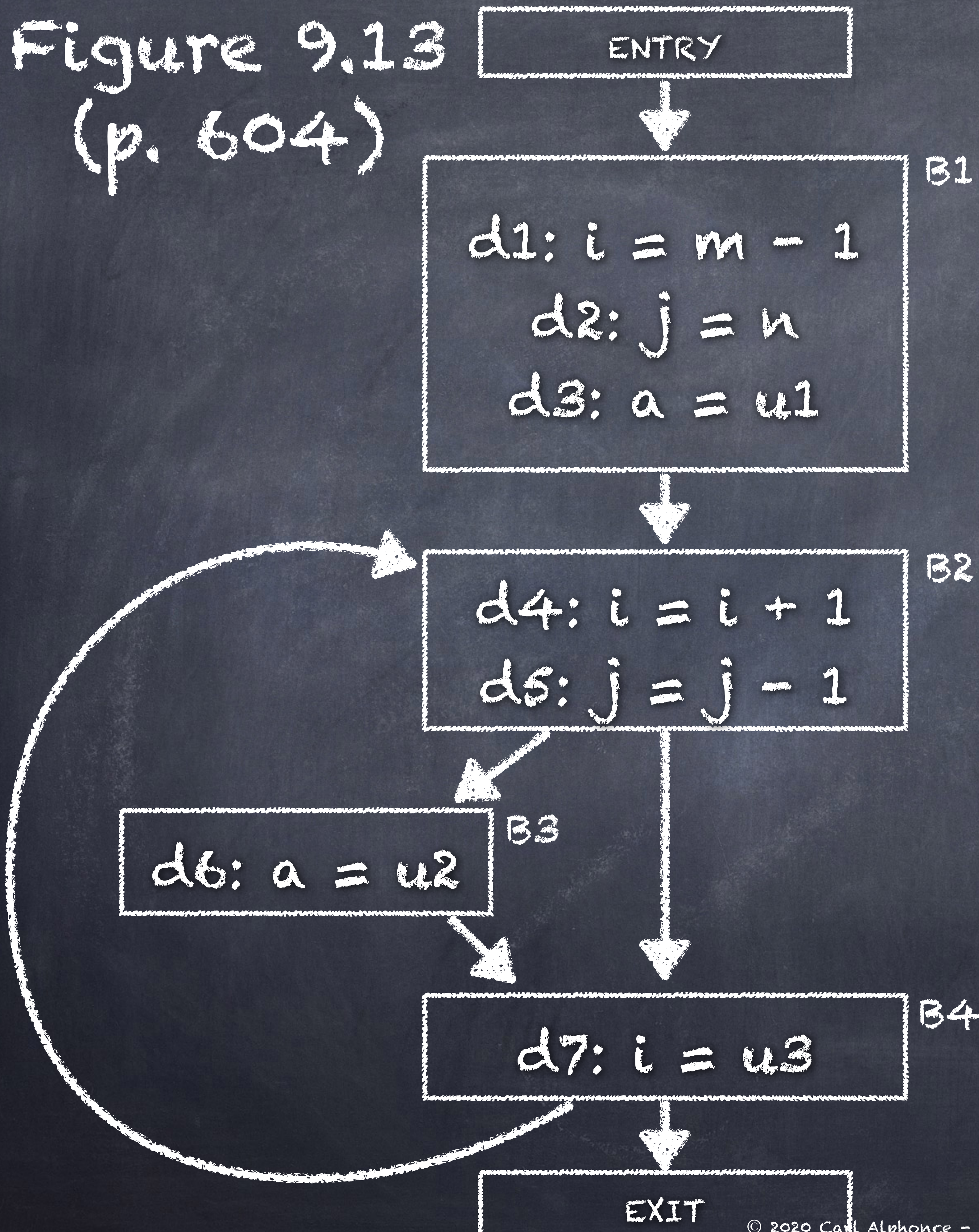
gen_{B1} = { d1, d2, d3 }
$kill_{B1}$ = { d4, d5, d6, d7 }

gen_{B2} = { d4, d5 }
$kill_{B2}$ = { d1, d2, d7 }

gen_{B3} = { d6 }
$kill_{B3}$ = { ? }

gen_{B4} = { ? }
$kill_{B4}$ = { ? }

Figure 9.13 (p. 604)



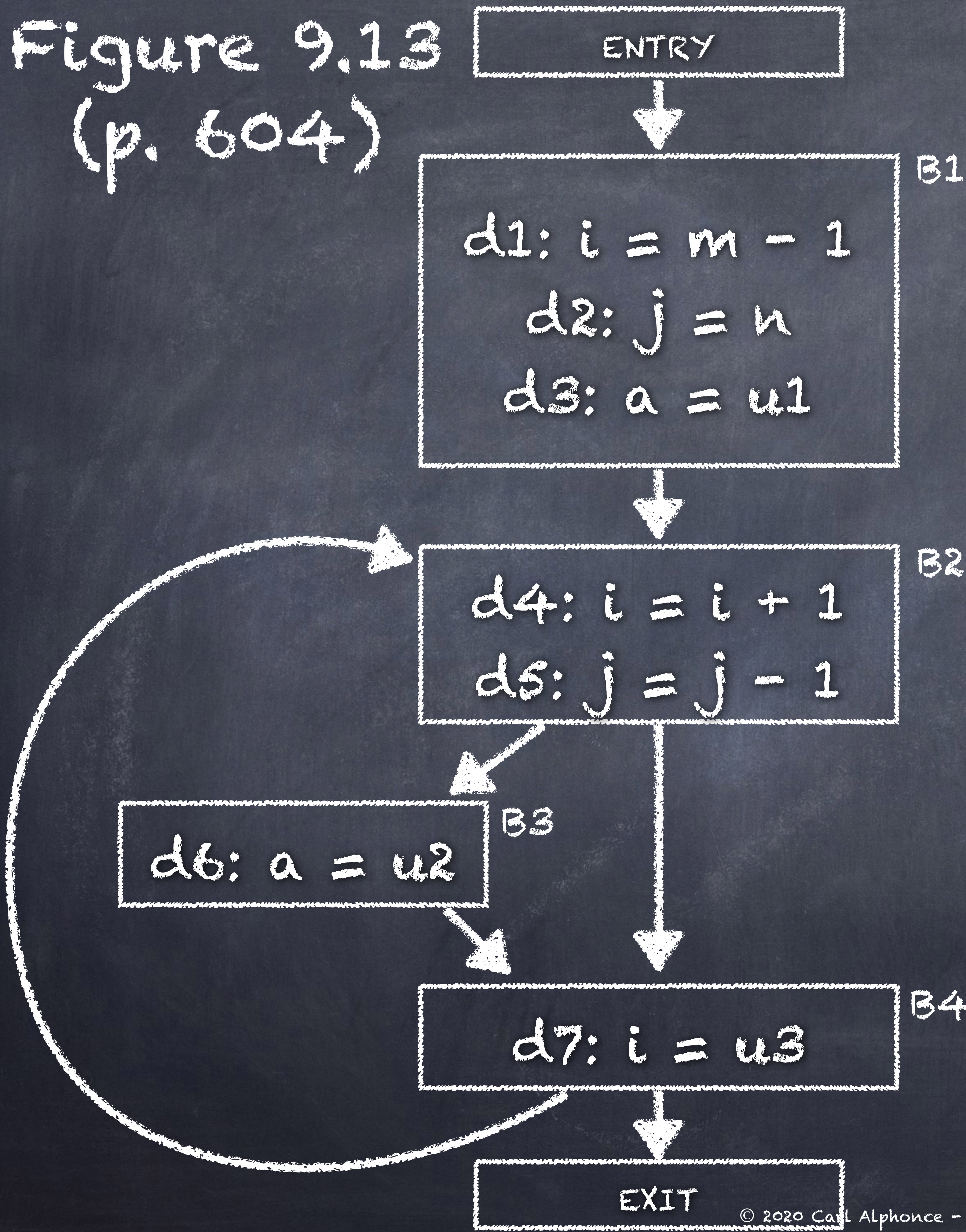
$gen_{B1} = \{ d1, d2, d3 \}$
$kill_{B1} = \{ d4, d5, d6, d7 \}$

$gen_{B2} = \{ d4, d5 \}$
$kill_{B2} = \{ d1, d2, d7 \}$

$gen_{B3} = \{ d6 \}$
$kill_{B3} = \{ d3 \}$

$gen_{B4} = \{ ? \}$
$kill_{B4} = \{ ? \}$

Figure 9.13 (p. 604)



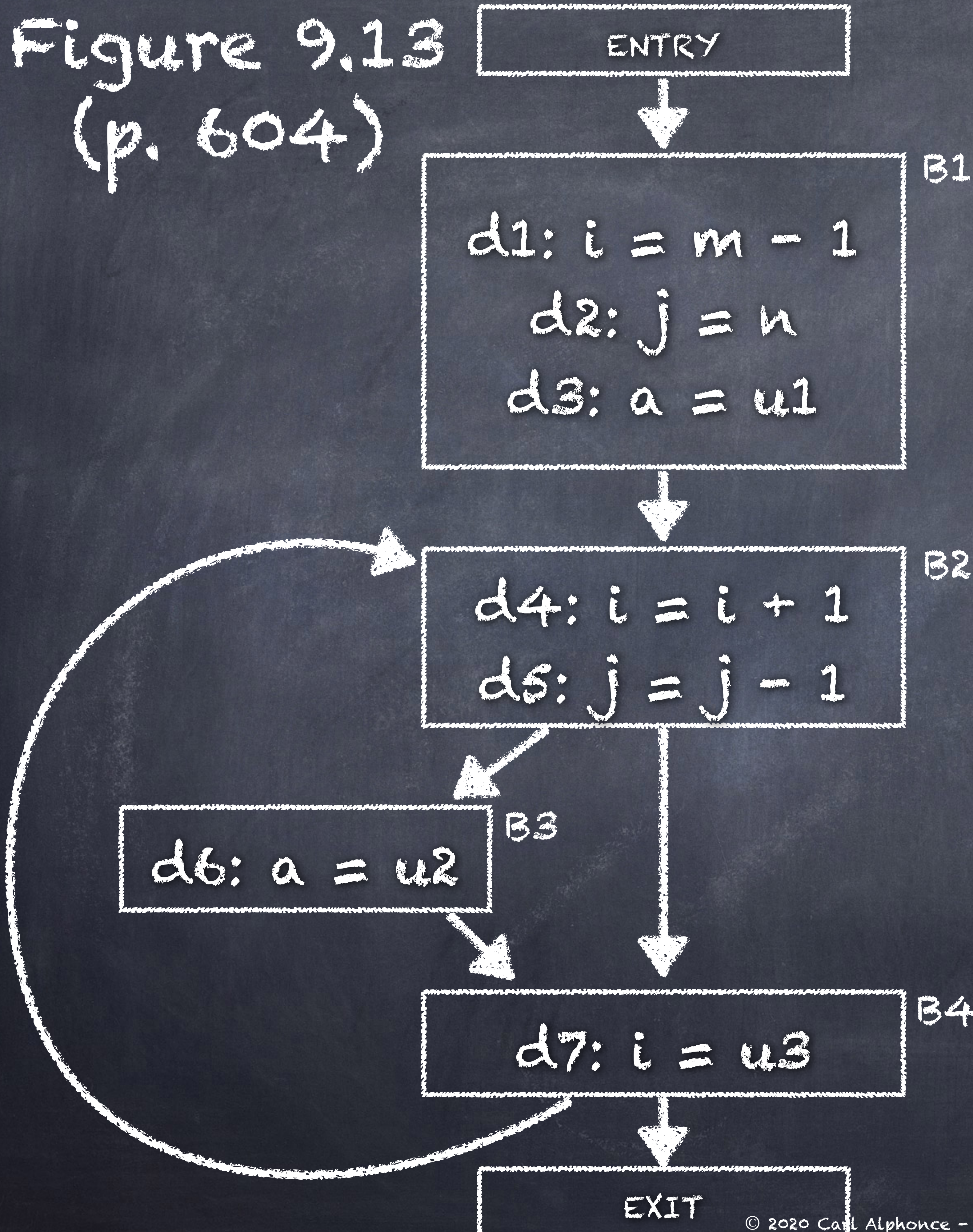
$gen_{B1} = \{ d1, d2, d3 \}$
$kill_{B1} = \{ d4, d5, d6, d7 \}$

$gen_{B2} = \{ d4, d5 \}$
$kill_{B2} = \{ d1, d2, d7 \}$

$gen_{B3} = \{ d6 \}$
$kill_{B3} = \{ d3 \}$

$gen_{B4} = \{ d7 \}$
$kill_{B4} = \{ ? \}$

Figure 9.13 (p. 604)



gen_{B1} = { d1, d2, d3 }
$kill_{B1}$ = { d4, d5, d6, d7 }

gen_{B2} = { d4, d5 }
$kill_{B2}$ = { d1, d2, d7 }

gen_{B3} = { d6 }
$kill_{B3}$ = { d3 }

gen_{B4} = { d7 }
$kill_{B4}$ = { d1, d4 }

Extending transfer equations from statements to blocks

Composition of f_1 and f_2:

$$f_1(x) = gen_1 \cup (x - kill_1)$$

$$f_2(x) = gen_2 \cup (x - kill_2)$$

$$f_2(f_1(x)) = gen_2 \cup ((gen_1 \cup (x - kill_1)) - kill_2)$$

$$= gen_2 \cup ((gen_1 - kill_2) \cup ((x - kill_1) - kill_2))$$

$$= gen_2 \cup (gen_1 - kill_2) \cup (x - (kill_1 \cup kill_2))$$

Extending transfer equations from statements to blocks

In general:

$f_B(x) = gen_B \cup (x - kill_B)$

$kill_B = \bigcup_{i \in n} kill_i$

$$\begin{aligned} gen_B = \; & gen_n \cup \\ & (gen_{n-1} - kill_n) \cup \\ & (gen_{n-2} - kill_{n-1} - kill_n) \cup \\ & \ldots \cup \\ & (gen_1 - kill_2 - kill_3 - \ldots - kill_n) \end{aligned}$$

Extending transfer equations from statements to blocks

"The gen set contains all the definitions inside the block that are "visible" immediately after the block - we refer to them as downwards exposed. A definition is downwards exposed in a basic block only if it is not "killed" by a subsequent definition to the same variable inside the same basic block." [p. 605]

Iterative algorithm for reaching definitions

Algorithm [p. 606]

INPUT: A flow graph for which $kill_B$ and gen_B have been computed for each block B.

OUTPUT: IN[B] and OUT[B], the set of definitions reaching the entry and exit of each block B of the flow graph

METHOD:

```
OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = ∅ }
while (changes to any OUT occurs) {
    for (each basic block B other than ENTRY) {
        IN[B] = ∪_{P a predecessor of B} OUT[P]
        OUT[B] = gen_B ∪ ( IN[B] - kill_B )
    }
}
```

See footnote 4 on page 606

Iterative algorithm for reaching definitions

Algorithm [p. 606]

INPUT: A flow graph for which k
each block B.

OUTPUT: IN[B] and OUT[B], the
and exit of each block B of the

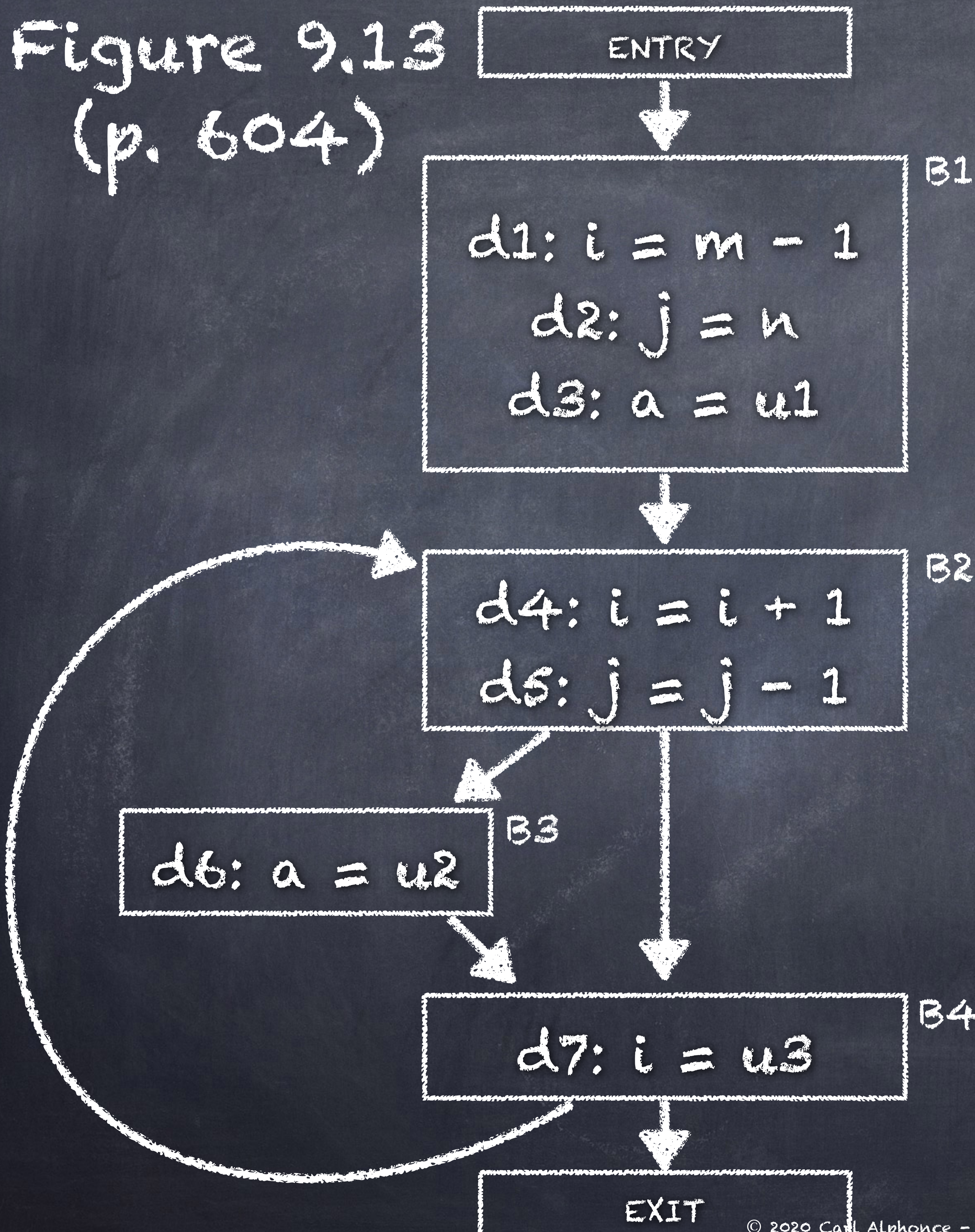
Written this way to allow different entry conditions for different data flow algorithms.

METHOD:

```
OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = ∅ }
while (changes to any OUT occurs) {
    for (each basic block B other than ENTRY) {
        IN[B] = ∪ P a predecessor of B OUT[P]
        OUT[B] = genB ∪ ( IN[B] - killB )
    }
}
```

See footnote 4 on page 606

Figure 9.13 (p. 604)



$gen_{B1} = \{ d1, d2, d3 \}$
$kill_{B1} = \{ d4, d5, d6, d7 \}$

$gen_{B2} = \{ d4, d5 \}$
$kill_{B2} = \{ d1, d2, d7 \}$

$gen_{B3} = \{ d6 \}$
$kill_{B3} = \{ d3 \}$

$gen_{B4} = \{ d7 \}$
$kill_{B4} = \{ d1, d4 \}$

Example 9.12 - building off figure 9.13

```
OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = ∅ }
while (changes to any OUT occurs) {
    for (each basic block B other than ENTRY) {
        IN[B] = ∪P a predecessor of B OUT[P]
        OUT[B] = genB ∪ ( IN[B] - killB )
    }
}
```

	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$
B1					
B2					
B3					
B4					
EXIT					

Represent d_i as a bit vector, where each d is a definition from 9.13

Union of sets A ∪ B: A OR B

Difference of sets A − B: A AND B'

Compute in order B1, B2, B3, B4, EXIT

For example:

$IN[B2]^1 = OUT[B1]^1 \cup OUT[B4]^0 = 111\ 0000 \cup 000\ 0000 = 111\ 0000$

$$\begin{aligned} OUT[B2]^1 &= gen_{B2} \cup (IN[B2]^1 - kill_{B2}) \\ &= 000\ 1100 + (111\ 0000 - 110\ 0001) \\ &= 000\ 1100 + 001\ 0000 = 001\ 1100 \end{aligned}$$

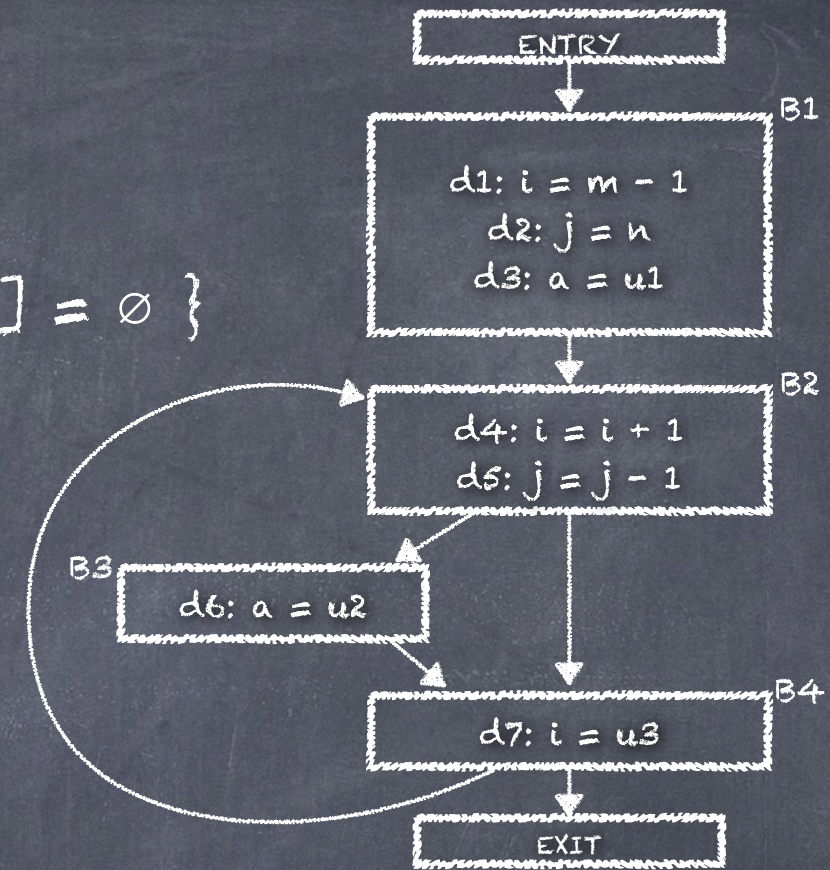
Example 9.12 - building off figure 9.13

```
OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = ∅ }
while (changes to any OUT occurs) {
    for (each basic block B other than ENTRY) {
        IN[B] = ∪P a predecessor of B OUT[P]
        OUT[B] = genB ∪ ( IN[B] - killB )
    }
}
```

	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$
B1	000 0000	[redacted]	[redacted]	[redacted]	[redacted]
B2	000 0000	[redacted]	[redacted]	[redacted]	[redacted]
B3	000 0000	[redacted]	[redacted]	[redacted]	[redacted]
B4	000 0000	[redacted]	[redacted]	[redacted]	[redacted]
EXIT	000 0000	[redacted]	[redacted]	[redacted]	[redacted]

Example 9.12

```
OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = ∅ }
while (changes to any OUT occurs) {
    for (each basic block B other than ENTRY) {
        IN[B] = ∪P a predecessor of B OUT[P]
        OUT[B] = genB ∪ ( IN[B] - killB )
    }
}
```



	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$
B1	000 0000	000 0000	111 0000		
B2	000 0000				
B3	000 0000				
B4	000 0000				
EXIT	000 0000				

$IN[B1] = pred(B1) = ENTRY$

$OUT[B1] = gen_{B1} \cup (IN[B1] - kill_{B1})$

$gen_{B1} = \{ d1, d2, d3 \}$

$kill_{B1} = \{ d4, d5, d6, d7 \}$

Example 9.12

OUT[ENTRY] = ∅

for (each basic block B other than ENTRY) { OUT[B] = ∅ }

while (changes to any OUT occurs) {

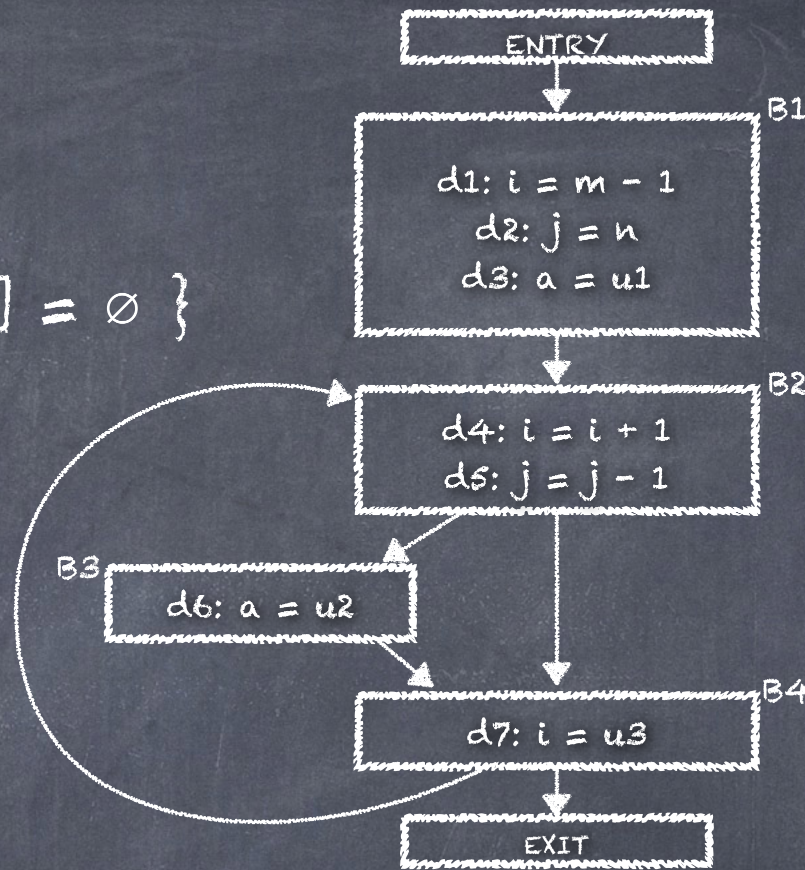
    for (each basic block B other than ENTRY) {

        $IN[B] = \cup_{P \text{ a predecessor of } B} OUT[P]$

        $OUT[B] = gen_B \cup (IN[B] - kill_B)$

    }

}



	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$
B1	000 0000	000 0000	111 0000		
B2	000 0000	111 0000	001 1100		
B3	000 0000				
B4	000 0000				
EXIT	000 0000				

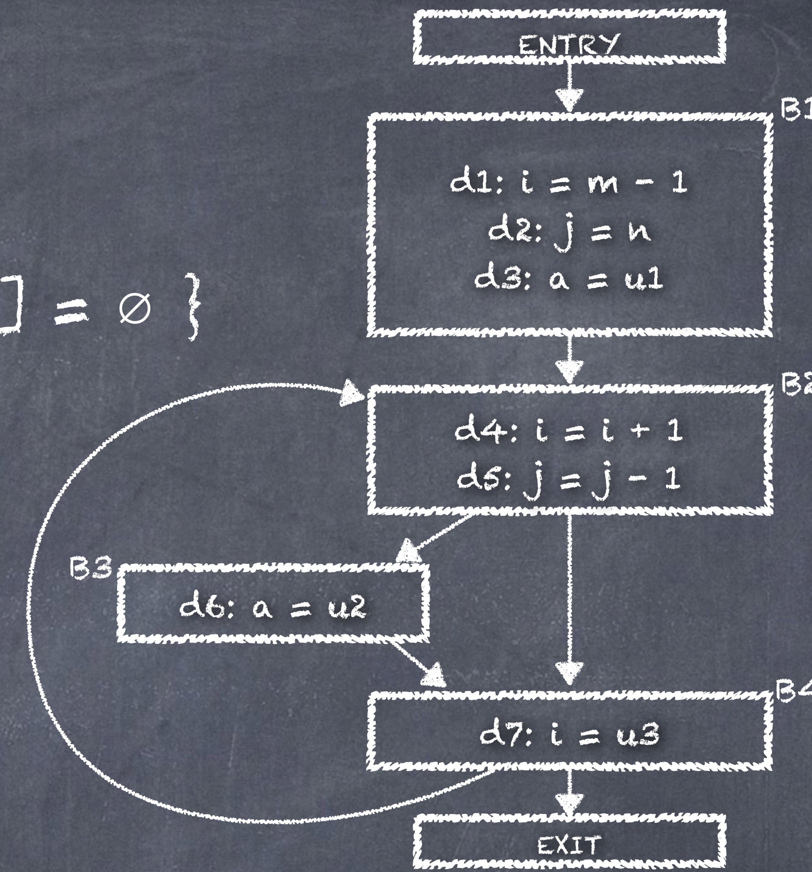
$IN[B2] = pred(B2) = OUT[B1] \cup OUT[B4]$

$OUT[B2] = gen_{B2} \cup (IN[B2] - kill_{B2})$

$gen_{B2} = \{ d4, d5 \}$

$kill_{B2} = \{ d1, d2, d7 \}$

Example 9.12




```
OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = ∅ }
while (changes to any OUT occurs) {
    for (each basic block B other than ENTRY) {
        IN[B] = ∪P a predecessor of B OUT[P]
        OUT[B] = genB ∪ ( IN[B] - killB )
    }
}
```

	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$
B1	000 0000	000 0000	111 0000	[redacted]	[redacted]
B2	000 0000	111 0000	001 1100	[redacted]	[redacted]
B3	000 0000	001 1100	000 1110	[redacted]	[redacted]
B4	000 0000	[redacted]	[redacted]	[redacted]	[redacted]
EXIT	000 0000	[redacted]	[redacted]	[redacted]	[redacted]

IN[B3] = pred(B3) = OUT[B2]

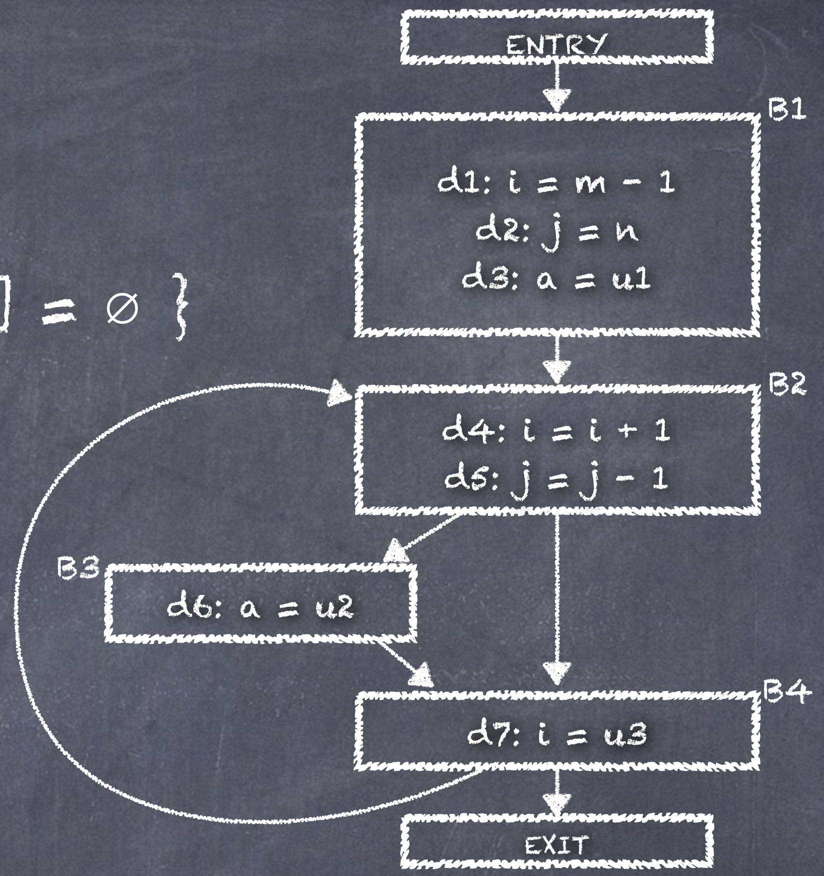
OUT[B3] = gen_{B3} ∪ (IN[B3] - $kill_{B3}$)

gen_{B3} = { d6 }

$kill_{B3}$ = { d3 }

Example 9.12

OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = ∅ }
while (changes to any OUT occurs) {
    for (each basic block B other than ENTRY) {
        $IN[B] = \cup_{P \text{ a predecessor of } B} OUT[P]$
        $OUT[B] = gen_B \cup (IN[B] - kill_B)$
    }
}



	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$
B1	000 0000	000 0000	111 0000		
B2	000 0000	111 0000	001 1100		
B3	000 0000	001 1100	000 1110		
B4	000 0000	001 1110	001 0111		
EXIT	000 0000				

$gen_{B4} = \{ d7 \}$
$kill_{B4} = \{ d1, d4 \}$

$IN[B4] = OUT[B2] \cup OUT[B3]$
$OUT[B4] = gen_{B4} \cup (IN[B4] - kill_{B4})$

Example 9.12

$OUT[ENTRY] = \emptyset$

for (each basic block B other than ENTRY) { $OUT[B] = \emptyset$ }

while (changes to any OUT occurs) {

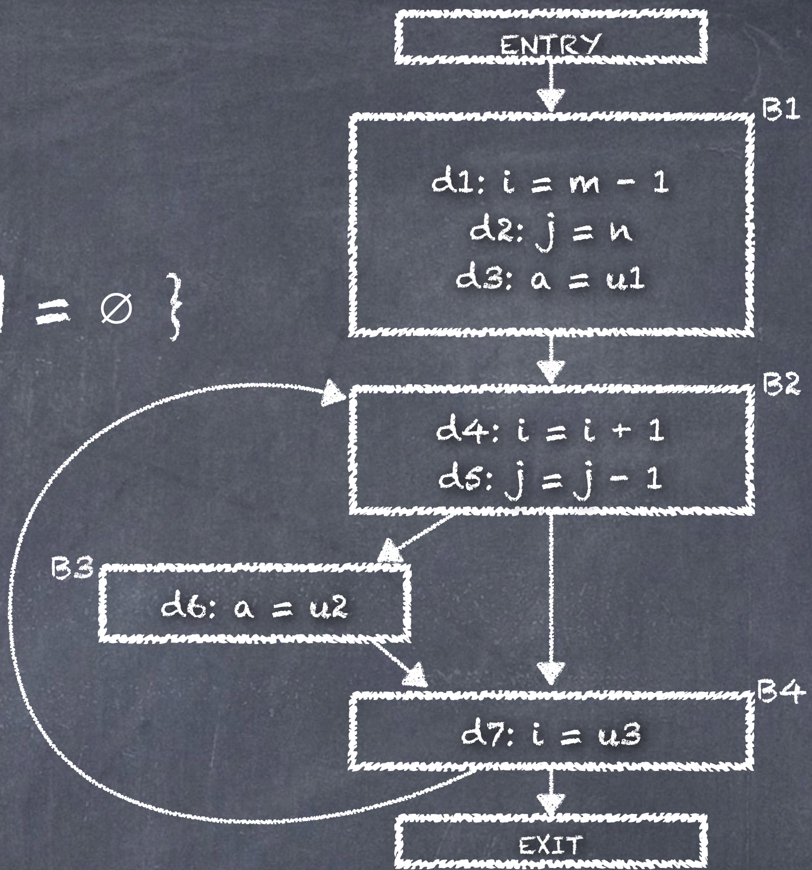
  for (each basic block B other than ENTRY) {

    $IN[B] = \cup_{P \text{ a predecessor of } B} OUT[P]$

    $OUT[B] = gen_B \cup (IN[B] - kill_B)$

  }

}

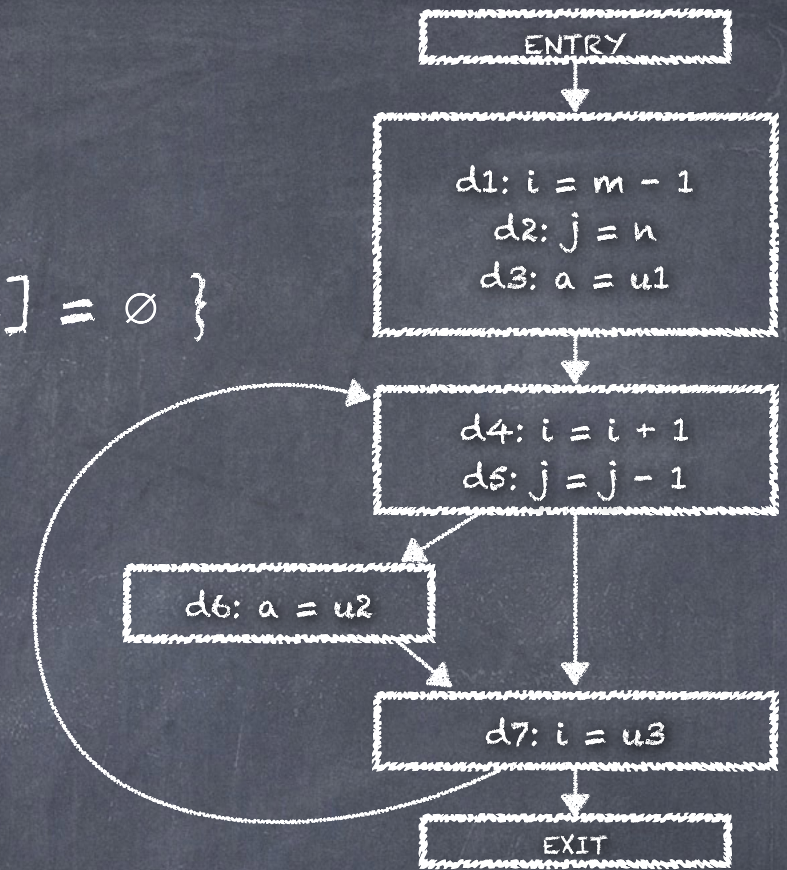


	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$
B1	000 0000	000 0000	111 0000	[redacted]	[redacted]
B2	000 0000	111 0000	001 1100	[redacted]	[redacted]
B3	000 0000	001 1100	000 1110	[redacted]	[redacted]
B4	000 0000	001 1110	001 0111	[redacted]	[redacted]
EXIT	000 0000	001 0111	001 0111	[redacted]	[redacted]

$IN[EXIT] = OUT[B4]$

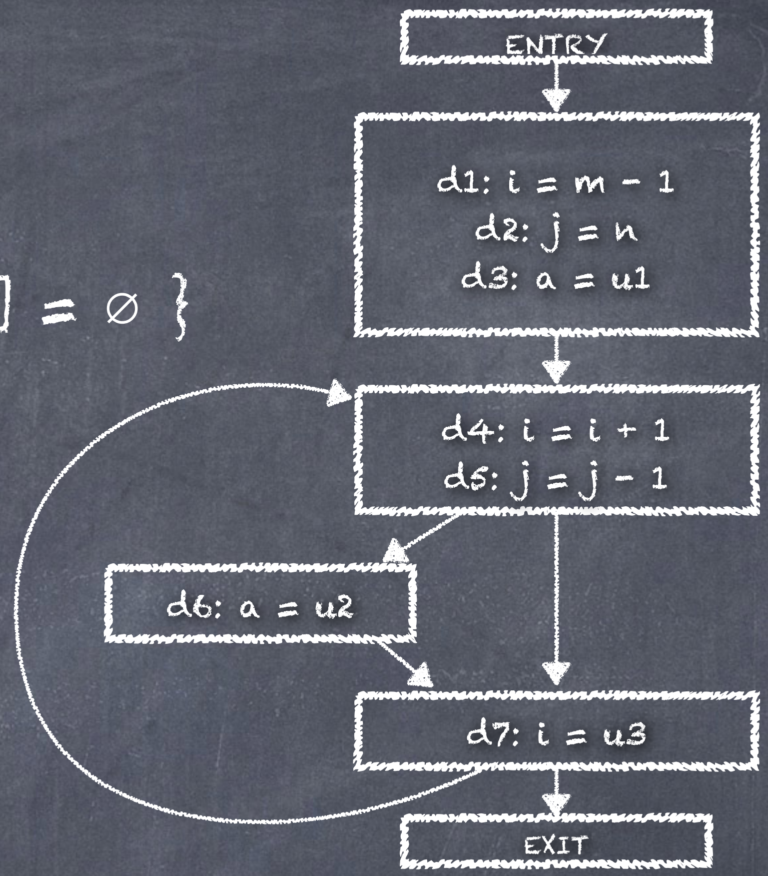
$OUT[EXIT] = IN[EXIT]$

```
OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = ∅ }
while (changes to any OUT occurs) {
    for (each basic block B other than ENTRY) {
        IN[B] = ∪P a predecessor of B OUT[P]
        OUT[B] = genB ∪ ( IN[B] - killB )
    }
}
```



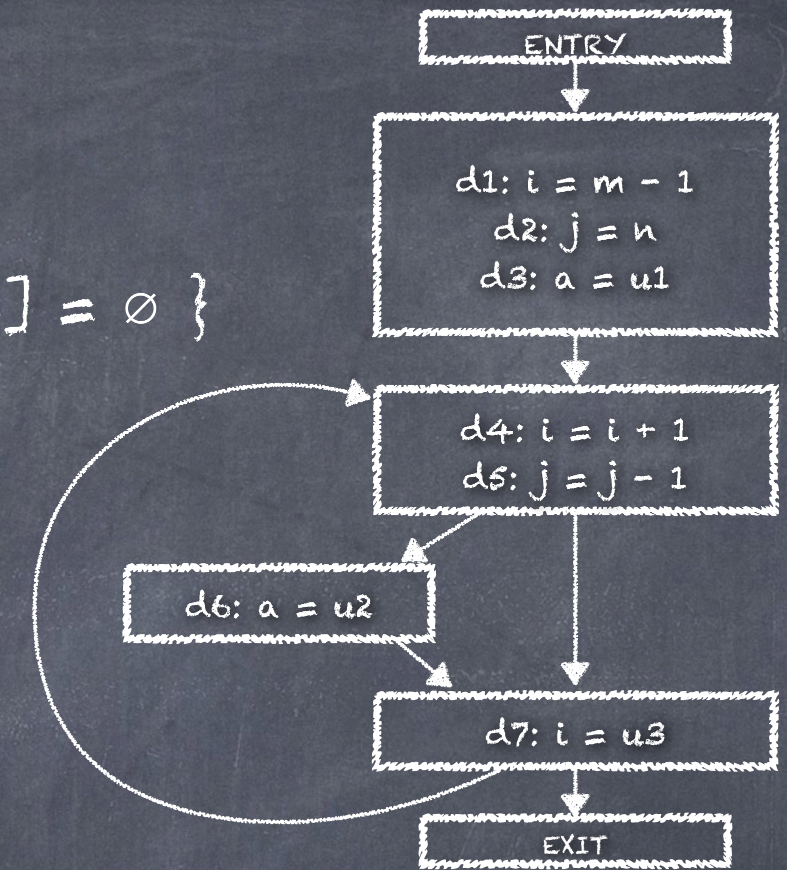
	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$
B1	000 0000	000 0000	111 0000	000 0000	111 0000
B2	000 0000	111 0000	001 1100	[redacted]	[redacted]
B3	000 0000	001 1100	000 1110	[redacted]	[redacted]
B4	000 0000	001 1110	001 0111	[redacted]	[redacted]
EXIT	000 0000	001 0111	001 0111	[redacted]	[redacted]

```
OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = ∅ }
while (changes to any OUT occurs) {
    for (each basic block B other than ENTRY) {
        IN[B] = ∪P a predecessor of B OUT[P]
        OUT[B] = genB ∪ ( IN[B] - killB )
    }
}
```



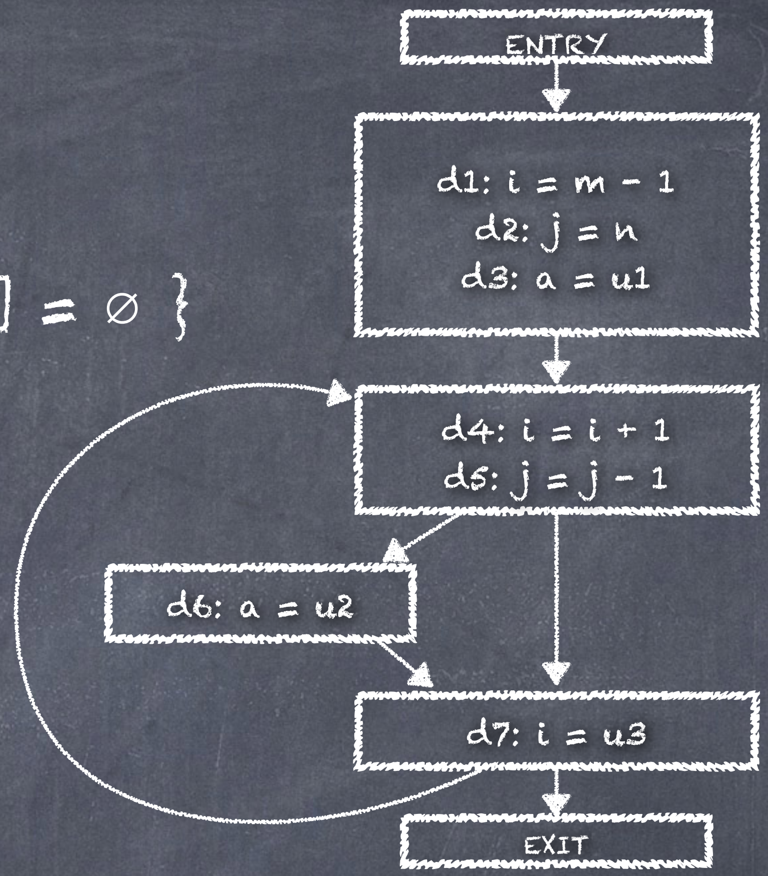
	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$
B1	000 0000	000 0000	111 0000	000 0000	111 0000
B2	000 0000	111 0000	001 1100	111 0111	001 1110
B3	000 0000	001 1100	000 1110	[redacted]	[redacted]
B4	000 0000	001 1110	001 0111	[redacted]	[redacted]
EXIT	000 0000	001 0111	001 0111	[redacted]	[redacted]

OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = ∅ }
while (changes to any OUT occurs) {
    for (each basic block B other than ENTRY) {
        $IN[B] = \cup_{P \text{ a predecessor of } B} OUT[P]$
        $OUT[B] = gen_B \cup (IN[B] - kill_B)$
    }
}



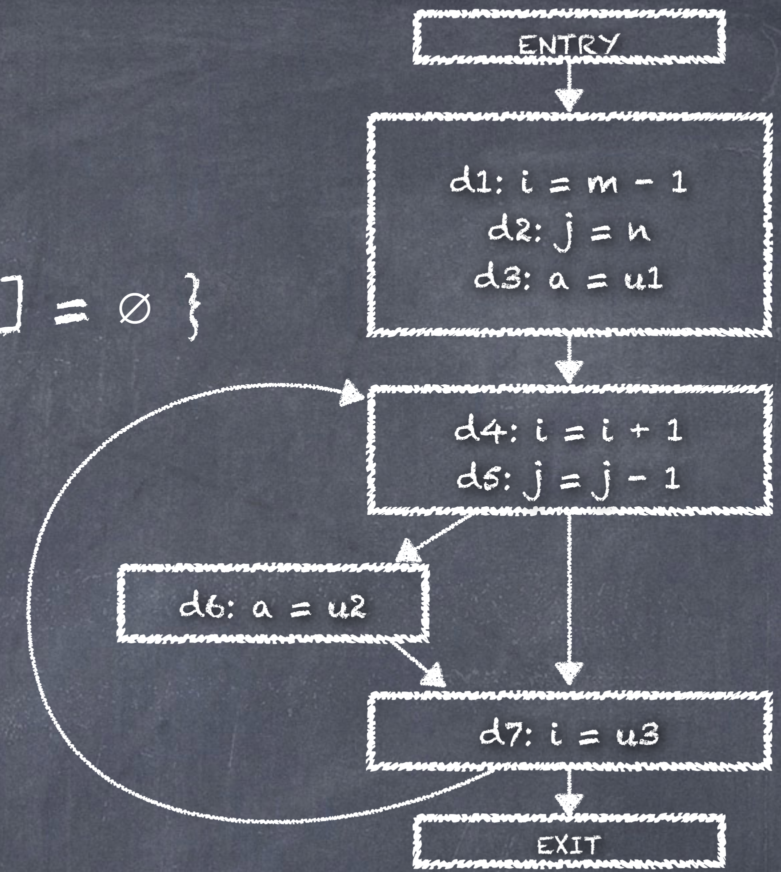
	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$
B1	000 0000	000 0000	111 0000	000 0000	111 0000
B2	000 0000	111 0000	001 1100	111 0111	001 1110
B3	000 0000	001 1100	000 1110	001 1110	000 1110
B4	000 0000	001 1110	001 0111	[redacted]	
EXIT	000 0000	001 0111	001 0111		

```
OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = ∅ }
while (changes to any OUT occurs) {
    for (each basic block B other than ENTRY) {
        IN[B] = ∪P a predecessor of B OUT[P]
        OUT[B] = genB ∪ ( IN[B] - killB )
    }
}
```



	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$
B1	000 0000	000 0000	111 0000	000 0000	111 0000
B2	000 0000	111 0000	001 1100	111 0111	001 1110
B3	000 0000	001 1100	000 1110	001 1110	000 1110
B4	000 0000	001 1110	001 0111	001 1110	001 0111
EXIT	000 0000	001 0111	001 0111	[redacted]	

```
OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = ∅ }
while (changes to any OUT occurs) {
    for (each basic block B other than ENTRY) {
        IN[B] = ∪P a predecessor of B OUT[P]
        OUT[B] = genB ∪ ( IN[B] − killB )
    }
}
```



	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$
B1	000 0000	000 0000	111 0000	000 0000	111 0000
B2	000 0000	111 0000	001 1100	111 0111	001 1110
B3	000 0000	001 1100	000 1110	001 1110	000 1110
B4	000 0000	001 1110	001 0111	001 1110	001 0111
EXIT	000 0000	001 0111	001 0111	001 0111	001 0111

9.2.4 Reaching definitions

Useful for constant propagation and constant folding (§8.5.4 - p. 536, §9.4 - p. 632). Additional discussion and examples:

en.wikipedia.org/wiki/Constant_folding

Useful for global common subexpression elimination (§9.1.4 - p. 588, §9.2.6 - p. 610, §9.5 - p. 639). Additional discussion and examples:

en.wikipedia.org/wiki/Common_subexpression_elimination

9.2.5 Live variable analysis

Useful for effective register management.

"After a value is computed in a register, and presumably used within a block, it is not necessary to store that value if it is dead at the end of the block. Also, if all registers are full and we need another register, we should favor using a register with a dead value, since that value does not have to be stored." [p. 608]

9.2.5 Live variable analysis

"In live variable analysis we wish to know for variable x and point p whether the value of x at p could be used along some path in the flow graph starting at p. If so, we say x is live at p; otherwise, x is dead at p." [p. 608]

In contrast to reaching analysis, which used a forward transfer function, live variable analysis uses a backward transfer function.

9.2.5 Live variable analysis

definitions, page 609

def_B is "the set of variables defined in B prior to any use of that variable in B"

use_B is "the set of variables whose values may be used in B prior to any definition of the variable"

9.2.5 Live variable analysis

definitions, page 609

$IN[EXIT] = \emptyset$

$IN[B] = use_B \cup (OUT[B] - def_B)$

$OUT[B] = \bigcup_{S \text{ a successor of } B} IN[S]$

9.2.5 Live variable analysis

Algorithm [p. 610]

INPUT: A flow graph with def and use computed for each block.

OUTPUT: IN[B] and OUT[B], the set of variables live on entry and exit of each block of the flow graph.

METHOD:

```
IN[EXIT] = ∅
for (each basic block B other than EXIT) { IN[B] = ∅ }
while (changes to any IN occur) {
    for (each basic block B other than EXIT) {
        OUT[B] = ∪_{S a successor of B} IN[S]
        IN[B] = use_B ∪ (OUT[B] - def_B)
    }
}
```

9.2.6 Available expressions

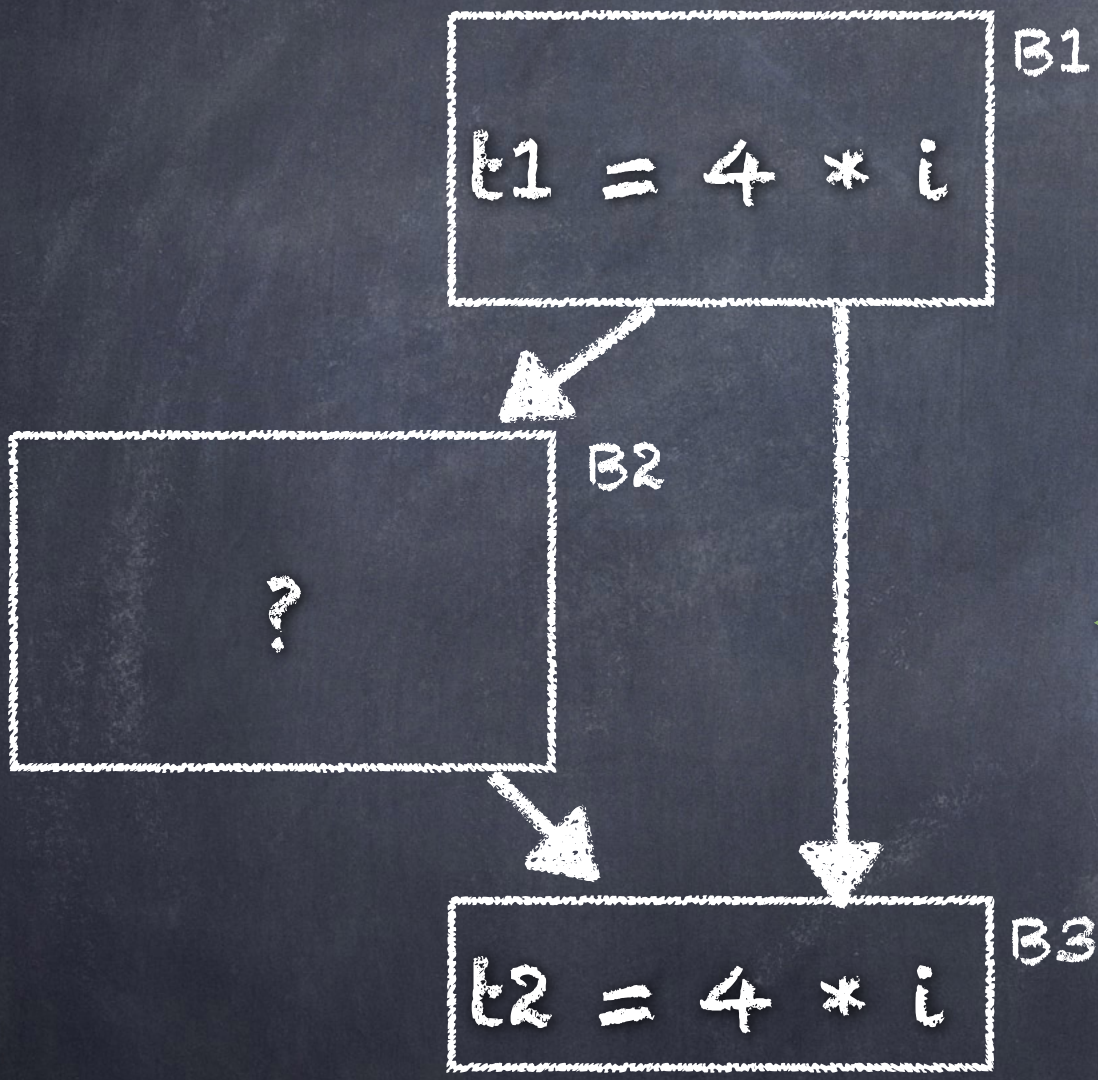
"An expression x+y is available at a point p if every path from the entry node to p evaluates to x+y, and after the last such evaluation prior to reaching p, there are no subsequent assignments to x or y." [p. 610]

9.2.6 Available expressions

"...a block **kills** expression x+y if it assigns (or may assign) x or y and does not subsequently recompute x+y." [p. 610]

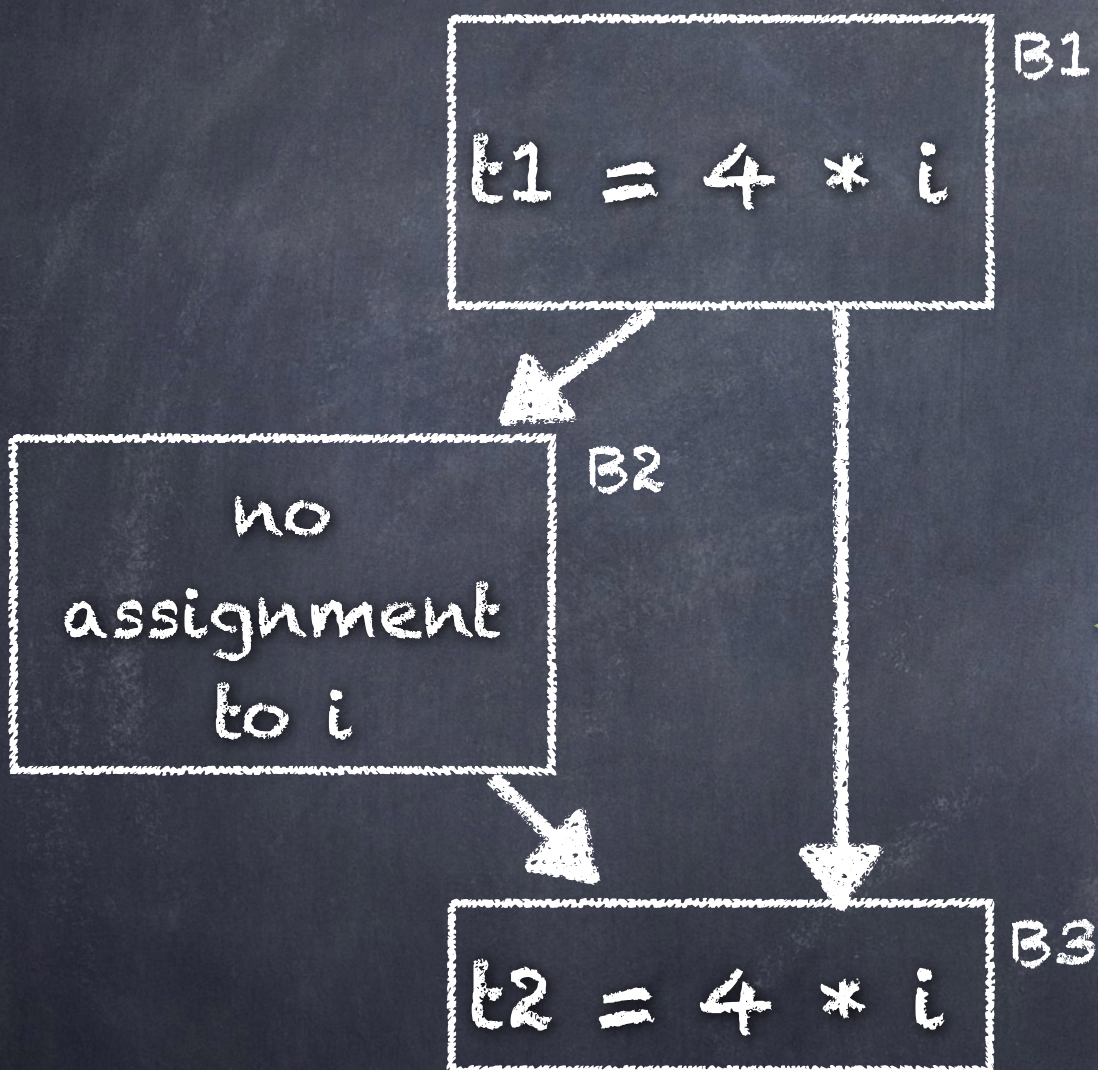
"A block **generates** expression x+y if it definitely evaluates x+y and does not subsequently define x or y." [p. 611]

Figure 9.17



"...the expression 4 * i in block B3 will be a common subexpression if 4 * i is available at the entry point of block B3."
[p 611]

Figure 9.17



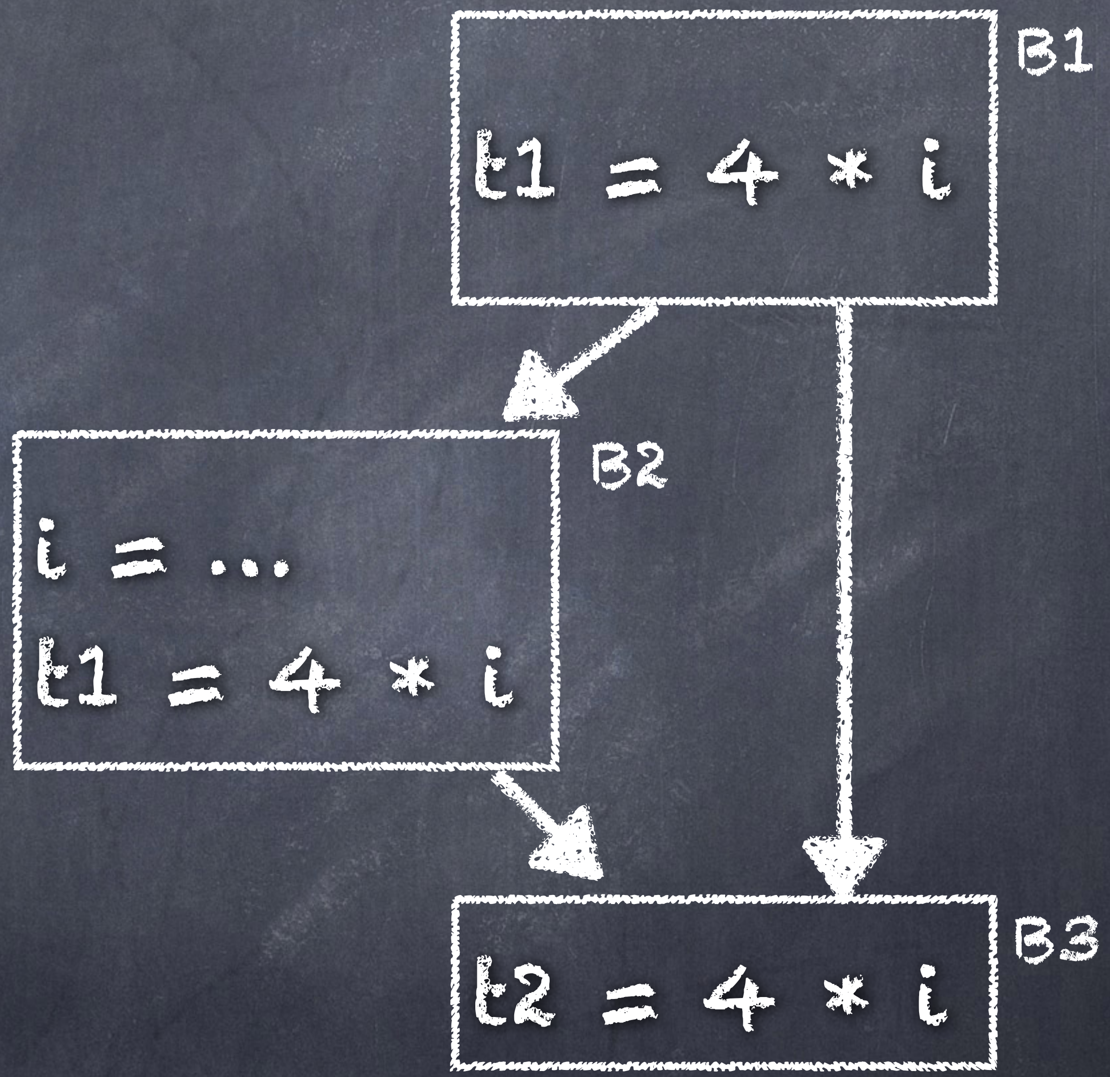
"It will be available if i is not assigned a new value in block B2, ..." [p 611]

Here 4 * i in B3 can be replaced by value of t1, regardless of which branch is taken.

Figure 9.17

"... or if ... 4 * i is recomputed after i is assigned in B2." [p 611]

Again, 4 * i in B3 can be replaced by value of t1, regardless of which branch is taken (since t1 contains the correct value of 4 * i in both cases)



9.2.6 Available expressions

Informally:

"If at point p set S of expressions is available, and q is the point after p, with statement $x=y+z$ between them, then we form the set of expressions available at q by the following steps:

1. Add to S the expression $y+z$.
2. Delete from S any expression involving variable x."

[p. 611]

Example 9.15

Statement	Available expressions
	$\varnothing$
$a = b + c$	
	$\{ b + c \}$
$b = a - d$	
	$\{ a - d \}$
$c = b + c$	
	$\{ a - d \}$
$d = a - d$	
	$\varnothing$

9.2.6 Available expressions

"We can find available expressions in a manner reminiscent of the way reaching definitions are computed. Suppose U is the 'universal' set of all expressions appearing on the right of one or more statement of the program. For each block B, let IN[B] be the set of expressions in U that are available at the point just before the beginning of B. Let OUT[B] be the same for the point following the end of B. Define e_gen_B to be the expressions generated by B and e_kill_B to be the set of expressions in U killed in B. Note that IN, OUT, e_gen, and e_kill can all be represented by bit vectors." [p. 612]

9.2.6 Available expressions

definitions, page 612

$OUT[ENTRY] = \emptyset$

$OUT[B] = e\_gen_B \cap (IN[B] - e\_kill_B)$

$IN[B] = \bigcap_{P \text{ a predecessor of } B} OUT[P]$

9.2.6 Available expressions

definitions, page 612

$OUT[ENTRY] = \emptyset$

$OUT[B] = e\_gen_B \cup (IN[B] - e\_kill_B)$

$IN[B] = \bigcap_{P \text{ a predecessor of } B} OUT[P]$

Note use of ∩ rather than ∪.

"...an expression is available at the beginning of a block only if it is available at the end of ALL its predecessors." [p. 612]

9.2.6 Available expressions

Algorithm [p. 614]

INPUT: A flow graph with e_kill_B and e_gen_B computed for each block B. The initial block is B1.

OUTPUT: IN[B] and OUT[B], the set of expressions available at the entry and exit of each block of the flow graph.

METHOD:

```
OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = U }
while (changes to any OUT occur) {
    for (each basic block B other than EXIT) {
        IN[B] = ∩_{P a predecessor of B} OUT[P]
        OUT[B] = e_gen_B ∩ (IN[B] - e_kill_B)
    }
}
```

9.2.6 Available expressions

Algorithm [p. 614]

INPUT: A flow graph with e_kill_B and e_gen_B computed for each block B. The initial block is B1.

OUTPUT: IN[B] and OUT[B], the set of expressions available at the entry and exit of each block of the flow graph.

METHOD:

$\quad$ OUT[ENTRY] = $\emptyset$
$\quad$ for (each basic block B other than ENTRY) { OUT[B] = U }
$\quad$ while (changes to any OUT occur) {
$\quad\quad$ for (each basic block B other than EXIT) {
$\quad\quad\quad$ $IN[B] = \bigcap_{P \text{ a predecessor of } B} OUT[P]$
$\quad\quad\quad$ $OUT[B] = e\_gen_B \cap (IN[B] - e\_kill_B)$
$\quad\quad$ }
$\quad$ }

Recall: U is set of all expressions

9.2 Summary

	Reaching definitions	Live variables	Available expressions
Domain	sets of definitions	sets of variables	sets of expressions
Direction	forward	backward	forward
Transfer function	$gen_B \cup (x - kill_B)$	$use_B \cup (x - def_B)$	$e\_gen_B \cap (x - e\_kill_B)$
Boundary	$OUT[ENTRY] = \varnothing$	$IN[EXIT] = \varnothing$	$OUT[ENTRY] = \varnothing$
Meet ($\wedge$)	$\cup$	$\cup$	$\cap$
Equations	$OUT[B] = f_B(IN[B])$ $IN[B] = \wedge_{P,pred(B)} OUT[P]$	$IN[B] = f_B(OUT[B])$ $OUT[B] = \wedge_{S,succ(B)} IN[S]$	$OUT[B] = f_B(IN[B])$ $IN[B] = \wedge_{P,pred(B)} OUT[P]$
Initialize	$OUT[B] = \varnothing$	$IN[B] = \varnothing$	$OUT[B] = U$