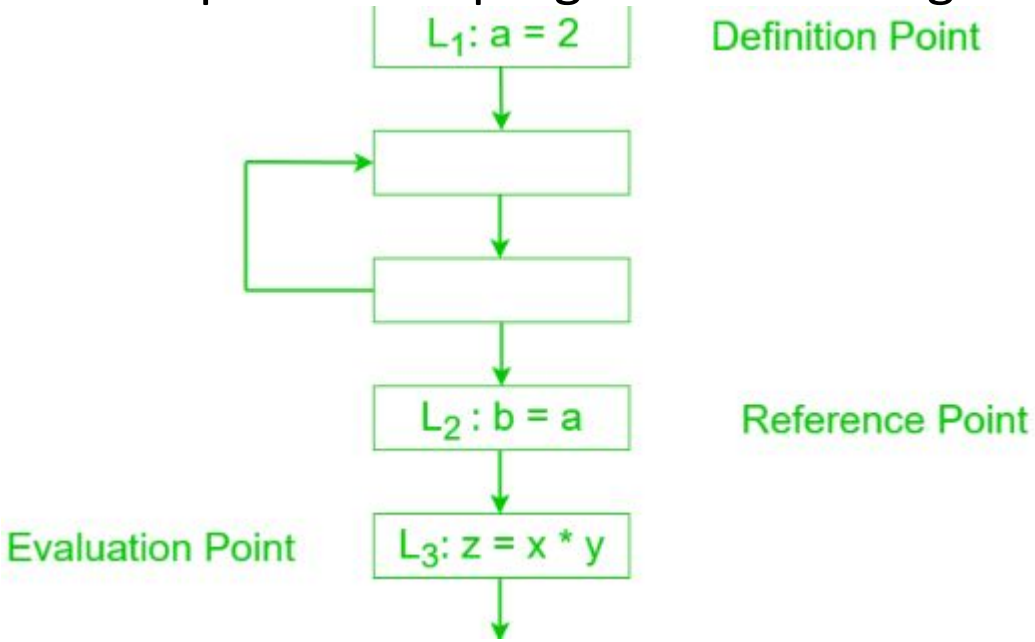
Unit-6
Global Data flow Analysis

Dr.C.Kavitha
AP(SG)
Dept. of CSE
PSG College of Technology

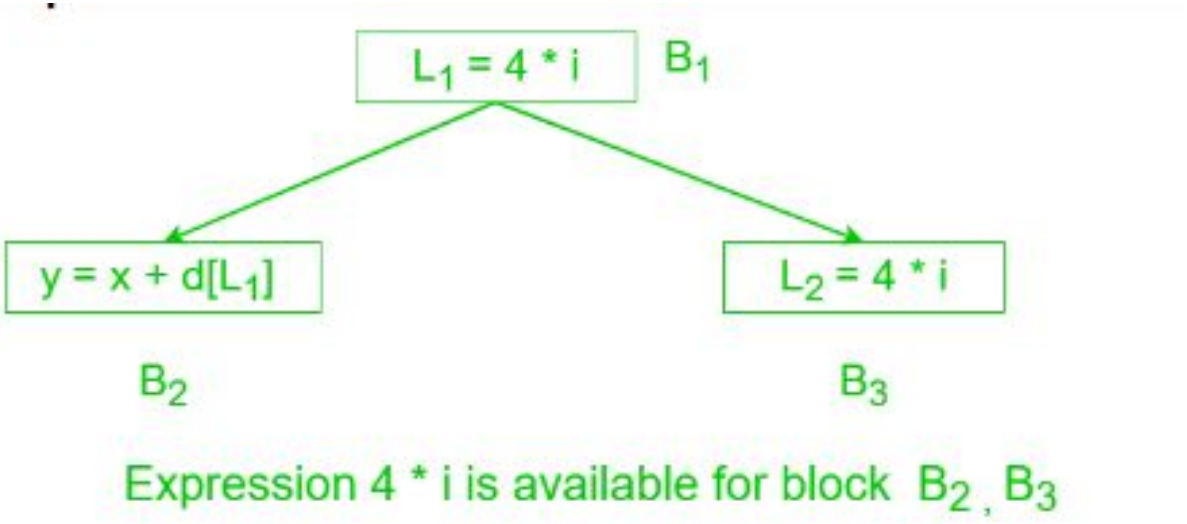
Introduction to Data Flow Analysis

- In order to do code optimization and a good job of code generation , compiler needs to collect information about the program as a whole and to distribute this information to each block in the flow graph.
- Data flow information that can be optimizing compiler collects by a process Data Flow Analysis.
- The generation and killing of process depends on the desired information on the Data Flow Analysis to be solved.
- Data Flow Analysis is affected by the control construct in a problem.

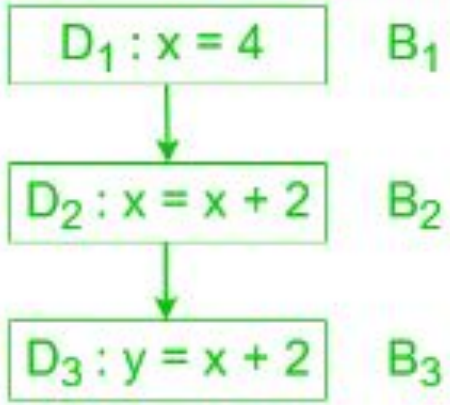
- Within each basic block, a point is assigned between two adjacent statements, before the first statement, and after the last statement
- A path from p1 to pn is a sequence of points p1, p2, …, pn such that for each i, 1 <= i <= n-1, either pi is the point immediately preceding a statement and pi+1 is the point immediately following that statement in the same block, or pi is the end of some block and pi+1 is the beginning of a successor block
- **Definition Point:** a point in a program containing some definition.
- **Reference Point:** a point in a program containing a reference to a data item.
- **Evaluation Point:** a point in a program containing evaluation of expression.



- **Available Expression –** A expression is said to be available at a program point x iff along paths its reaching to x. A Expression is available at its evaluation point.
  A expression a+b is said to be available if none of the operands gets modified before their use.
- **Advantage –**
  It is used to eliminate common sub expressions.

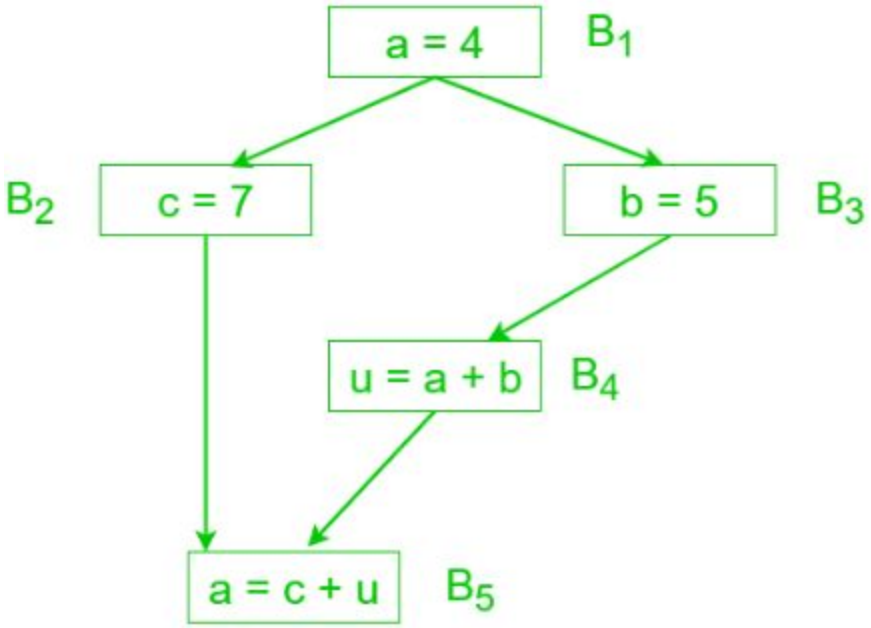


- **Reaching Definition –** A definition D is reaches a point x if there is path from D to x in which D is not killed, i.e., not redefined.
- **Advantage –**
  It is used in constant and variable propagation.



D_1 is reaching definition for B_2 but not for B_3 since it is killed by D_2

- **Live variable –** A variable is said to be live at some point p if from p to end the variable is used before it is redefined else it becomes dead.
- **Advantage –**
- It is useful for register allocation.
- It is used in dead code elimination.



a is live at block B_1 , B_3 , B_4 but killed at B_5

- **Busy Expression –** An expression is busy along a path iff its evaluation exists along that path and none of its operand definition exists before its evaluation along the path.
- **Advantage –**
  It is used for performing code movement optimization.

Information for Reaching Definitions

- gen[S]: definitions generated within S and reaching the end of S
- kill[S]: definitions killed within S
- in[S]: definitions reaching the beginning of S
- out[S]: definitions reaching the end of S

Data Flow Equation

- Data flow information can be collected by setting up and solving systems of equations that relate information at various points

  out[S] = gen[S] U (in[S] - kill[S])

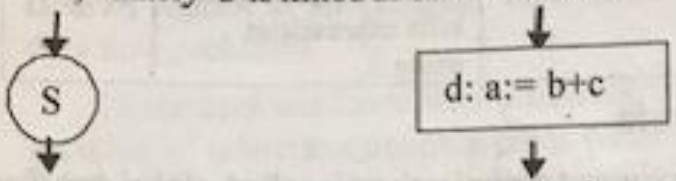
- Let S be a statement, the information at the end of a statement (**out[S])** is either generated within the statement (**gen[S])** or enters at the beginning (**in[S]**) and is not killed as control flows through the statement

The Iterative Algorithm

- Repeatedly compute in and out sets for each node in the control flow graph simultaneously until there is no change
- in[B] = Up ϵ pred(B) out[P]
- out[B] = gen[B] U (in[B] - kill[B])

The data flow equations for reaching definitions for different types of statements are given below:

a. If a block has only one **simple** statement represented by S. Here the definition of 'a', namely 'd' is killed as shown below.

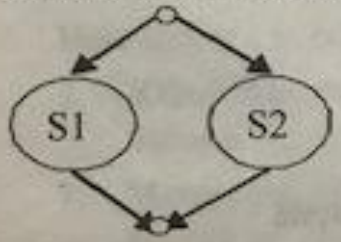


gen[S] = {d}
kill[S] = in[S]-{d}
out[S] = gen[S] U (in[S] - kill[S])

b. If a block has a **sequence** of simple statements S1 and S2. Hence out[S1] becomes in[S2]. Also in[S] is in[S1] and out[S2] is out[S]. Definitions generated and killed within the block are a combination of those for S1 and S2.

gen[S] = gen[S2] U (gen[S1] - kill [S2])
kill[S] = kill[S2] U (kill[S1] - gen [S2])
in[S1] = in[S]
in[S2] = out[S1]
out[S] = out[S2]

c. If a block has a **decision** to choose one of the paths of either S1 or S2, in[S] is in[S1] and in[S2]. Also, out[S] is union of out[S1] and out[S2]. Definitions generated within the block are a combination of those for S1 and S2. Definitions killed within the block are the intersection of those in S1 and S2.

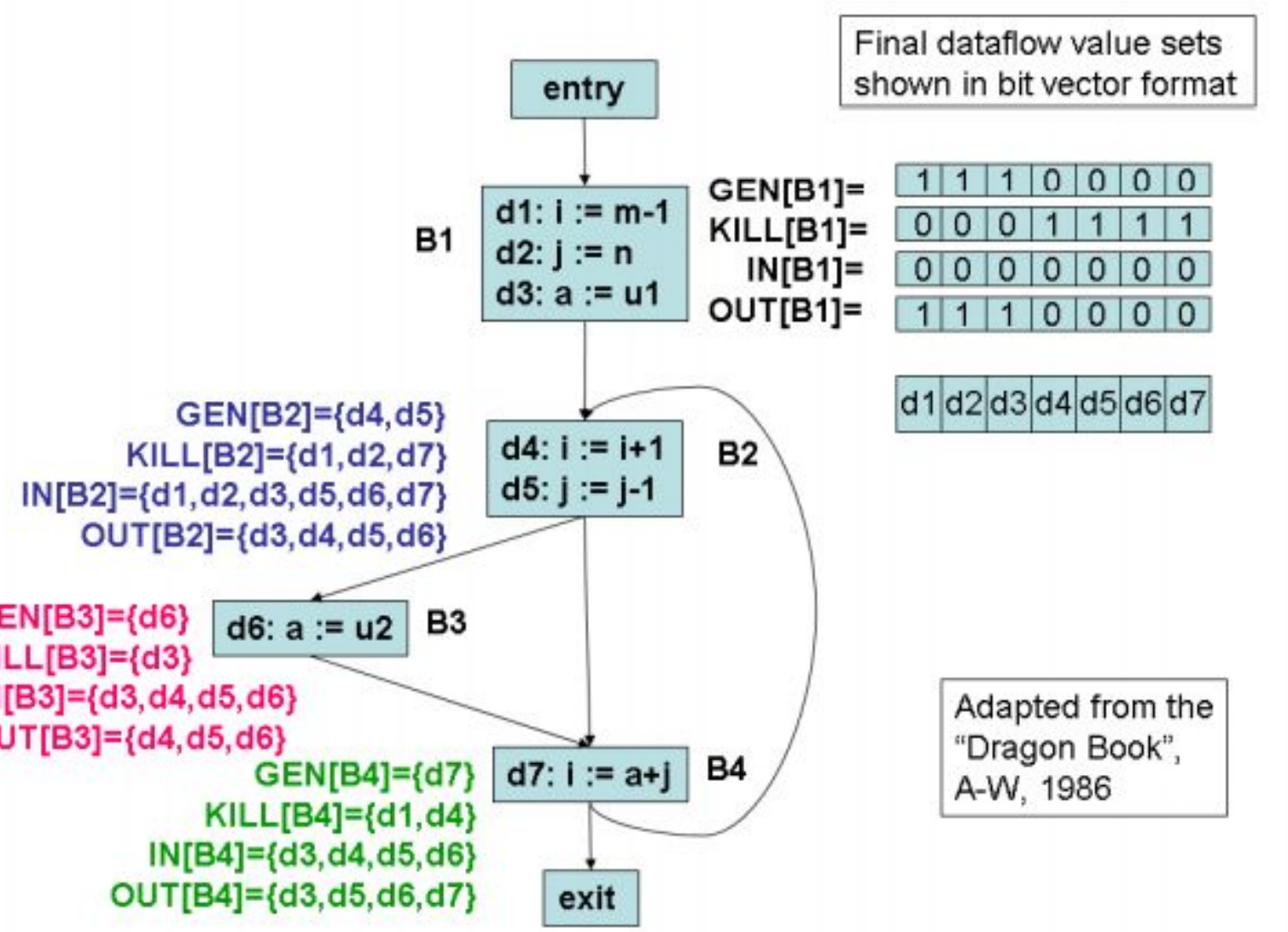


gen[S] = gen[S1] U gen[S2]
kill[S] = kill[S1] ∩ kill[S2]
in[S1] = in[S]
in[S2] = in[S]
out[S] = out[S1] U out [S2]

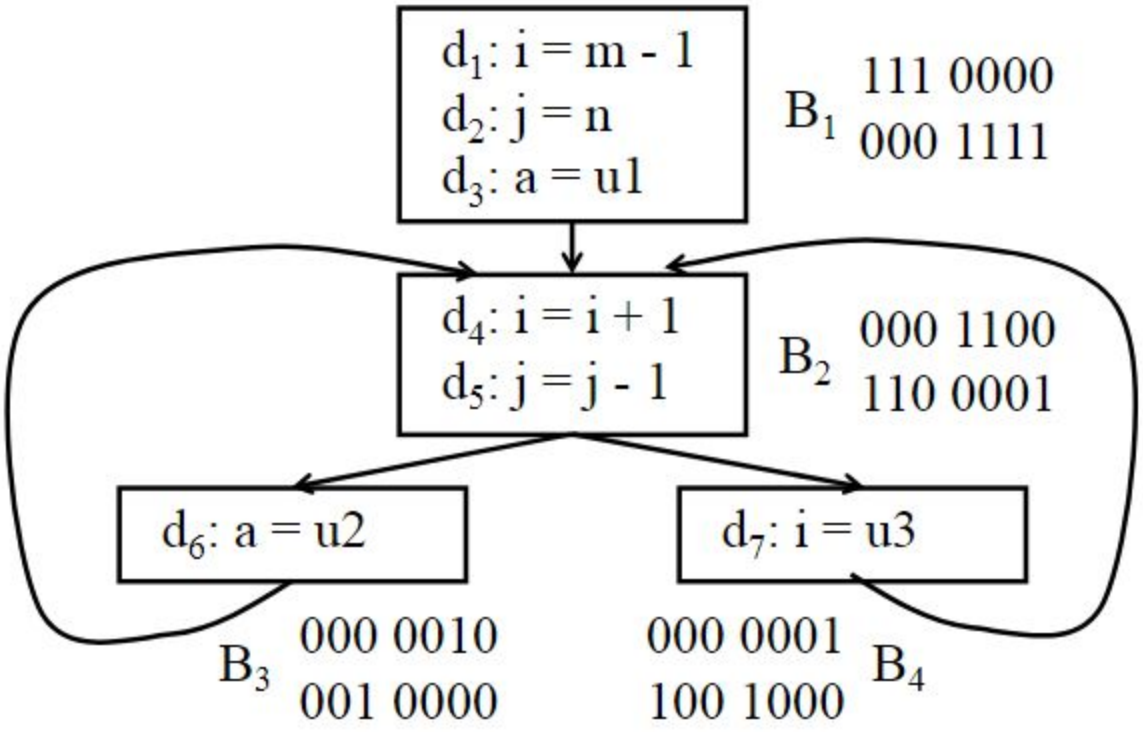
Algorithm: Reaching Definitions

```
/* Assume in[B] = ∅ for all B */
for each block B do out[B] := gen[B]
change := true;
while change do begin
    change := false;
    for each block B do begin
        in[B] := ∪_{p ∈ pred(B)} out[p]
        oldout := out[B]
        out[B] := gen[B] ∪ (in[B] - kill[B])
        if out[B] ≠ oldout then change := true
    end
end
```

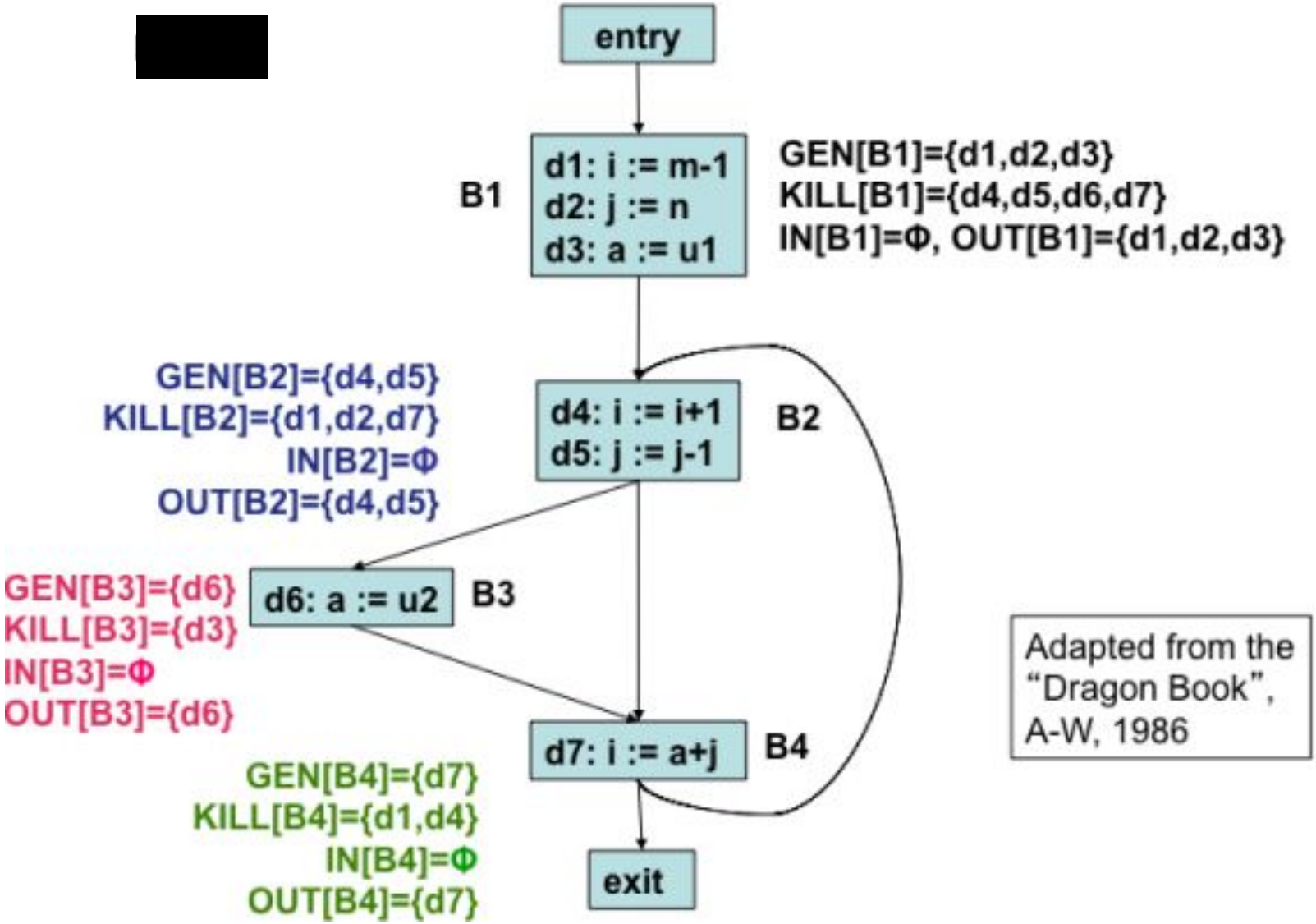
Reaching Definitions: Bit Vector Representation



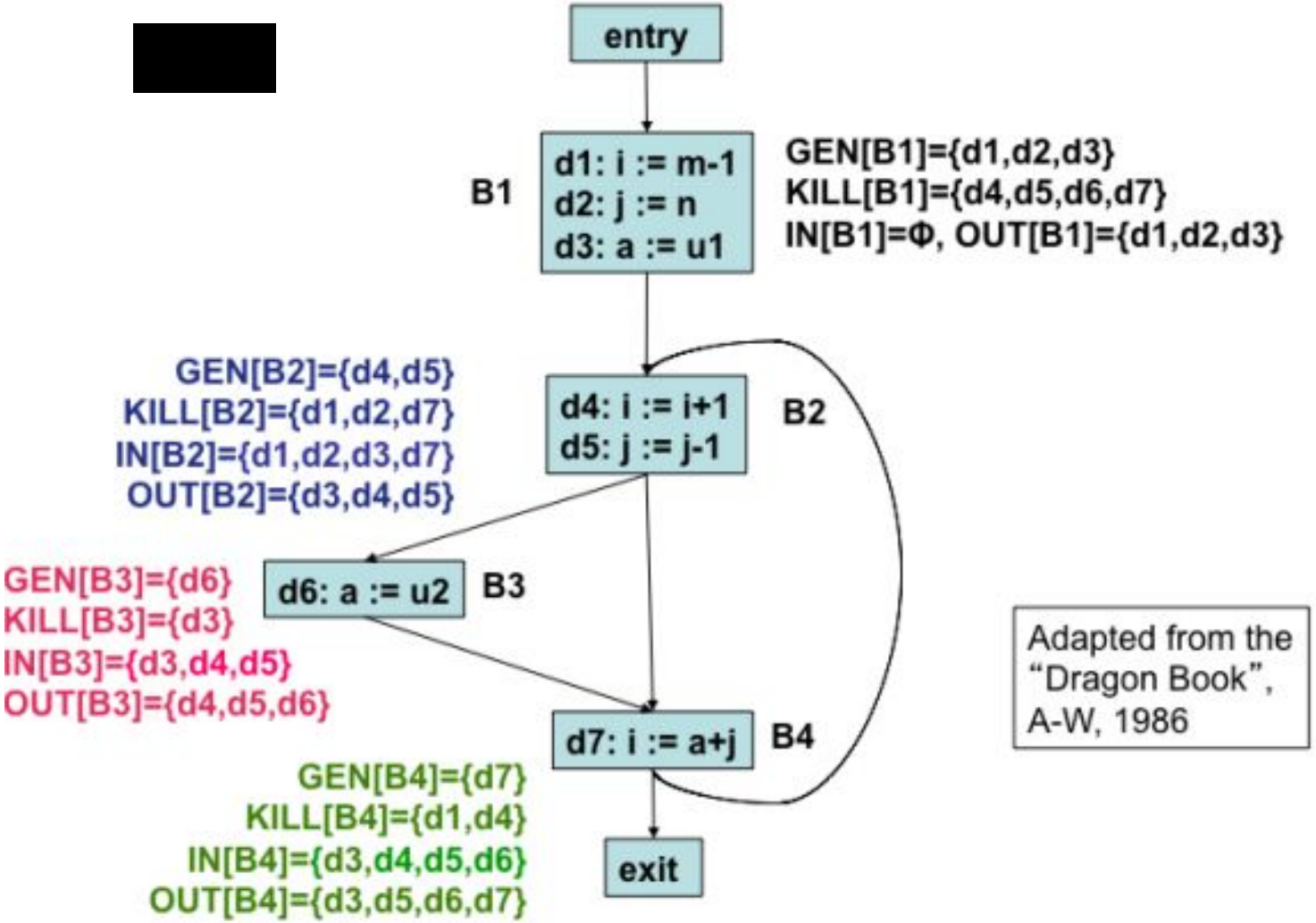
An Example


d_1: i = m - 1
d_2: j = n
d_3: a = u1
B_1
111 0000
000 1111
d_4: i = i + 1
d_5: j = j - 1
B_2
000 1100
110 0001
d_6: a = u2
d_7: i = u3
B_3
000 0010
001 0000
000 0001
100 1000
B_4

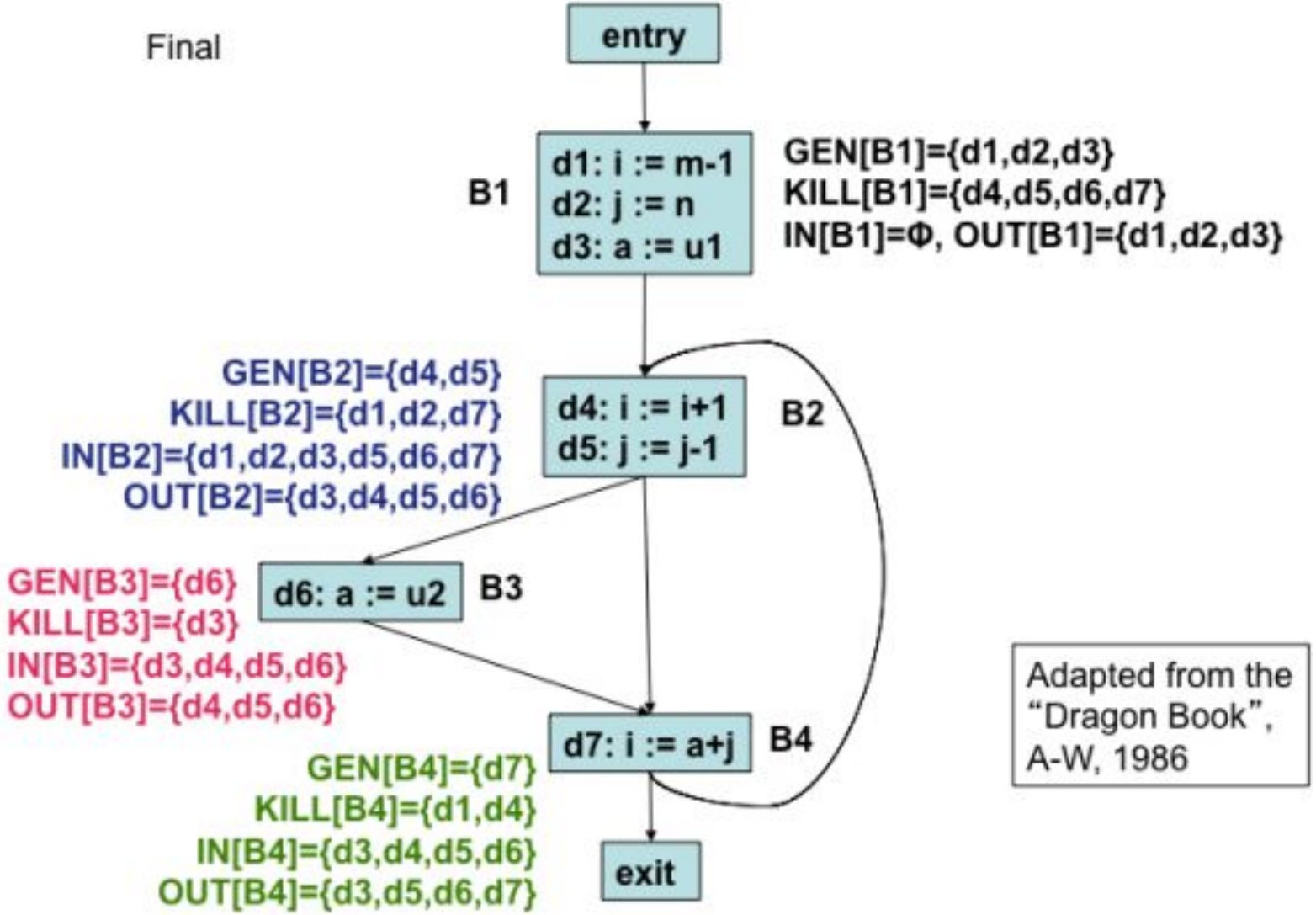
Pass 0


entry
B1
d1: i := m-1
d2: j := n
d3: a := u1
GEN[B1]={d1,d2,d3}
KILL[B1]={d4,d5,d6,d7}
IN[B1]=Φ, OUT[B1]={d1,d2,d3}
GEN[B2]={d4,d5}
KILL[B2]={d1,d2,d7}
IN[B2]=Φ
OUT[B2]={d4,d5}
d4: i := i+1
d5: j := j-1
B2
GEN[B3]={d6}
KILL[B3]={d3}
IN[B3]=Φ
OUT[B3]={d6}
d6: a := u2
B3
d7: i := a+j
B4
GEN[B4]={d7}
KILL[B4]={d1,d4}
IN[B4]=Φ
OUT[B4]={d7}
exit
Adapted from the "Dragon Book", A-W, 1986

Pass 1


entry
B1
d1: i := m-1
d2: j := n
d3: a := u1
GEN[B1]={d1,d2,d3}
KILL[B1]={d4,d5,d6,d7}
IN[B1]=Φ, OUT[B1]={d1,d2,d3}
GEN[B2]={d4,d5}
KILL[B2]={d1,d2,d7}
IN[B2]={d1,d2,d3,d7}
OUT[B2]={d3,d4,d5}
d4: i := i+1
d5: j := j-1
B2
GEN[B3]={d6}
KILL[B3]={d3}
IN[B3]={d3,d4,d5}
OUT[B3]={d4,d5,d6}
d6: a := u2
B3
d7: i := a+j
B4
GEN[B4]={d7}
KILL[B4]={d1,d4}
IN[B4]={d3,d4,d5,d6}
OUT[B4]={d3,d5,d6,d7}
exit
Adapted from the "Dragon Book", A-W, 1986

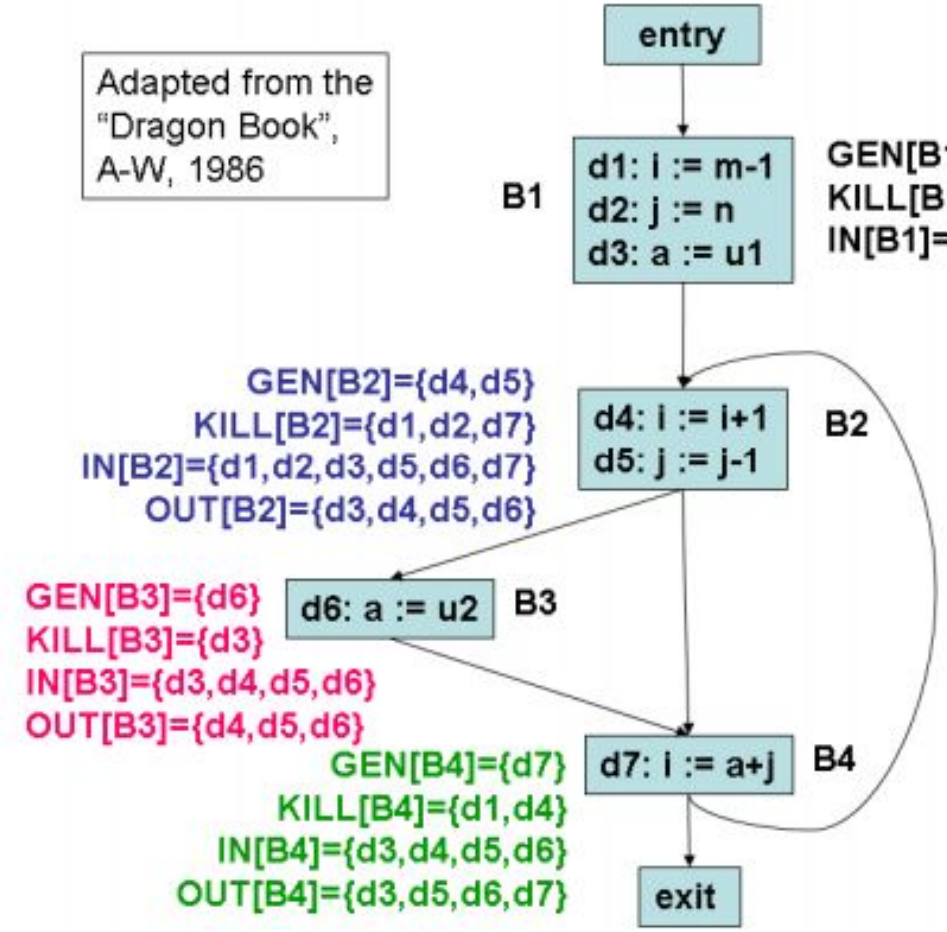
Final



Adapted from the "Dragon Book", A-W, 1986

An Example

Block	Initial		Pass 1		Pass 2	
	In[B]	Out[B]	In[B]	Out[B]	In[B]	Out[B]
B_1	000 0000	111 0000	000 0000	111 0000	000 0000	111 0000
B_2	000 0000	000 1100	111 0011	001 1110	111 1111	001 1110
B_3	000 0000	000 0010	001 1110	000 1110	001 1110	000 1110
B_4	000 0000	000 0001	001 1110	001 0111	001 1110	001 0111



use	u-d chain
(i,d4)	(d1,d7)
(j,d5)	(d2,d5)
(a,d7)	(d3,d6)
(j,d7)	(d5)

Block B2

In[B2]=1110011 (d1,d2,d3,d6,d7} From program flow graph {d2,d3,d6} do not define i. Hence ud-chain for I defined in B2 is only {d1,d7}

Thank You

- https://tutorialspoint.dev/computer-science/compiler-design/data-flow-analysis-compiler
- https://slideplayer.com/slide/3418022/