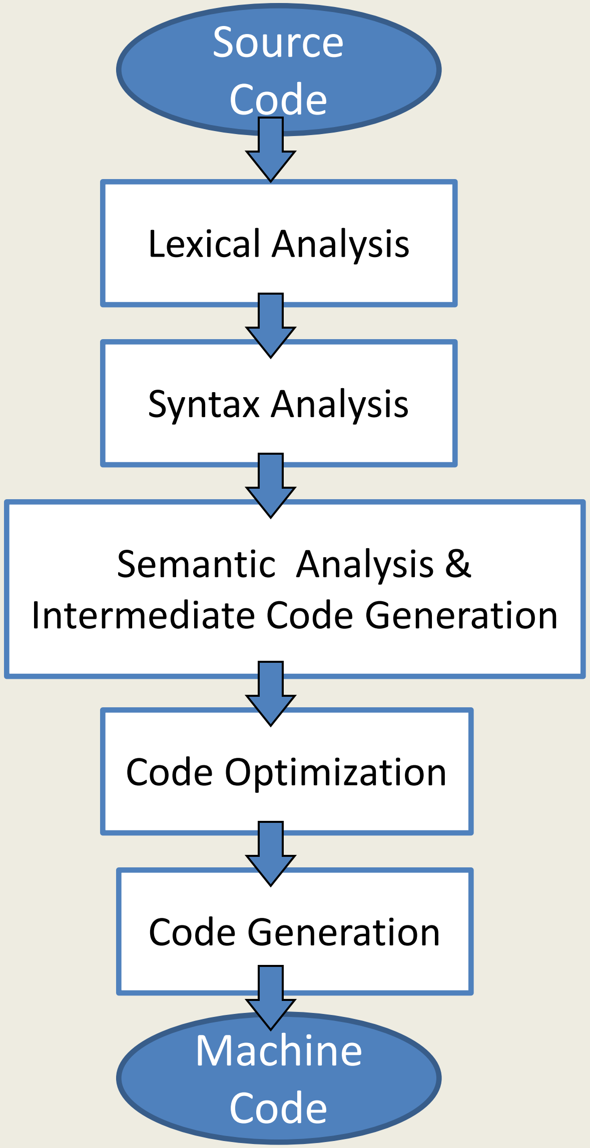
Code Optimization

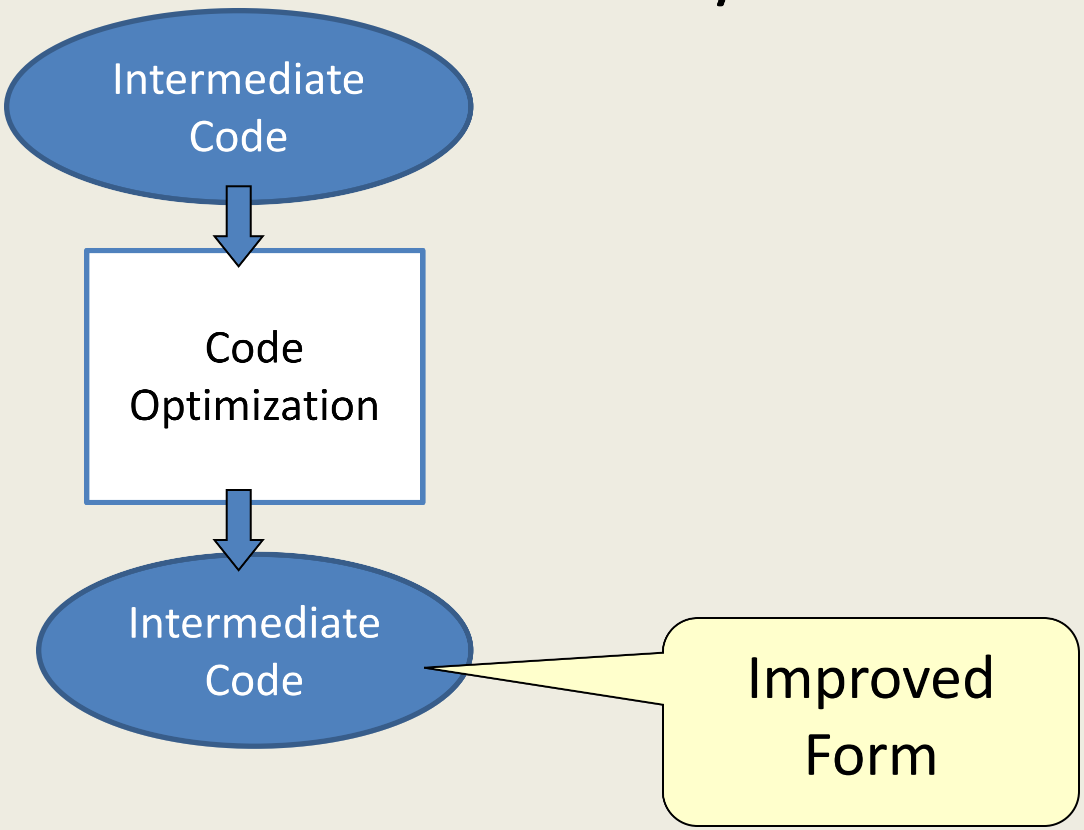
Dinesh Gopalani
dgopalani.cse@mnit.ac.in

Phases in a Compiler



Code Optimization

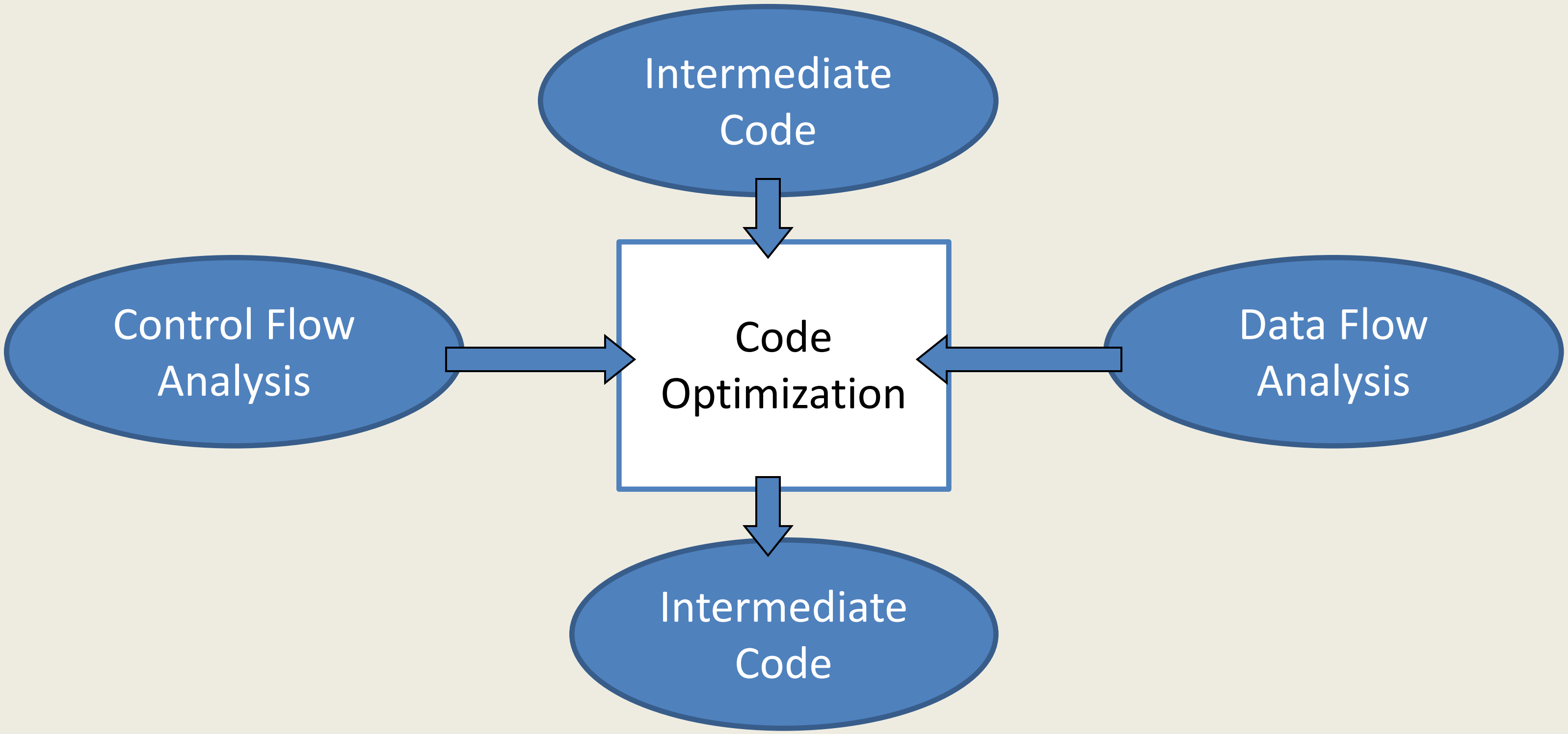
To improve the Intermediate Code so that the final Machine Code runs faster and/or takes less space



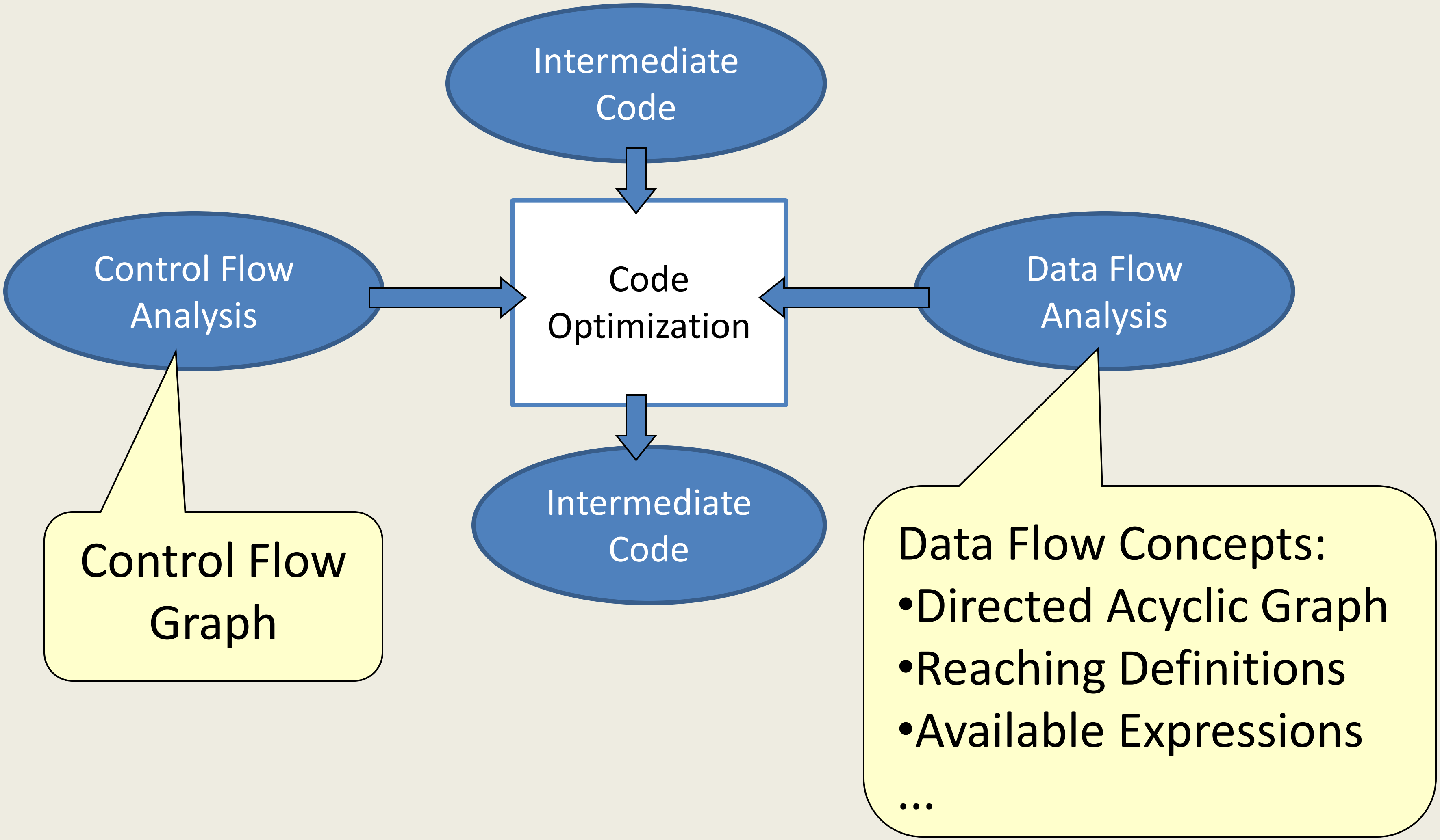
Code Optimization

- The optimization shall capture most of the potential improvements without an unreasonable amount of efforts.
- It must preserve the meaning of the Source Program.
- It shall reduce the execution time and/or space taken by the machine code.

How Code Optimization works?



How Code Optimization works?

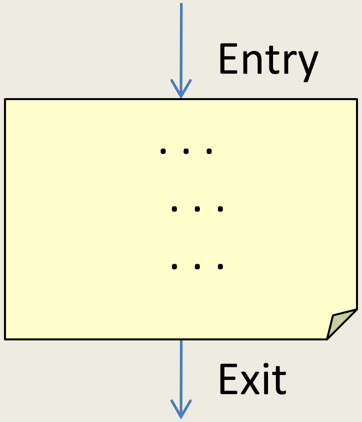


Control Flow Graph

- The Control Flow Graph is a directed graph depicting the flow of the program.
- The nodes of the graph are Basic Blocks.
- The graph depicts how the program control is being passed among the basic blocks.
- It also helps in locating unwanted code (unreachable code) in the program.

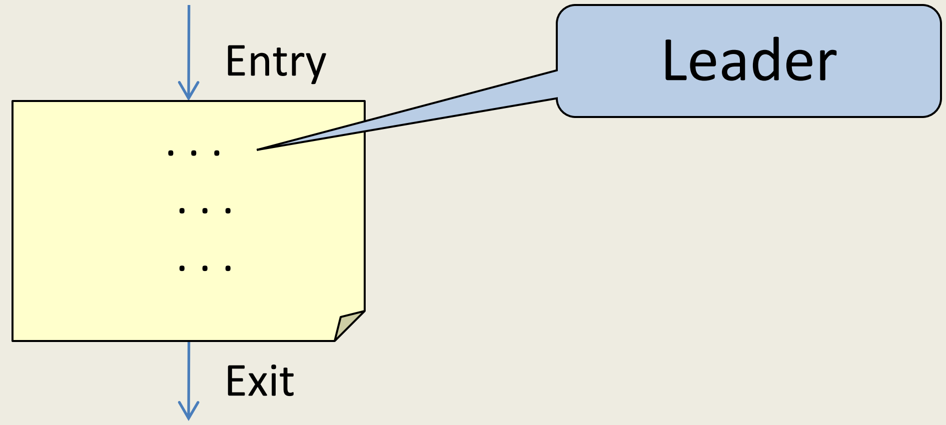
Basic Block

It is a sequence of consecutive statements which can be entered only at the beginning, and when entered are executed in sequence without halt or possibility of branch except at the end.



Basic Block

It is a sequence of consecutive statements which can be entered only at the beginning, and when entered are executed in sequence without halt or possibility of branch except at the end.



Basic Block

Method for Partition of Program into Basic Blocks:

1. Identify all the leaders using the following rules:
   1. The first statement of the program is a leader.
   2. Any statement which is the target of a conditional or unconditional goto is a leader.
   3. Any statement which immediately follows a conditional goto is a leader.
2. For each leader construct its basic block, which consists of the leader and all statements upto but not including the next leader or the end of the program.

Basic Block

Example:

```
begin
  i=1;
  sum=0;
  while(i<=100)
  {
    sum=sum+i*i;
    i=i+1;
  }
  print sum;
end
```

Basic Block

Example:

Source Code

```
begin
  i=1;
  sum=0;
  while(i<=100)
  {
    sum=sum+i*i;
    i=i+1;
  }
  print sum;
end
```

Intermediate Code

```
1:   i=1;
2:   sum=0;
3:   if i<=100 goto 5;
4:   goto 9;
5:   t1=i*i;
6:   sum=sum+t1;
7:   i=i+1;
8:   goto 3;
9:   print sum;
10;  END
```

Basic Block

Example:

Leader

```
1:  i=1;
2:  sum=0;
3:  if i<=100 goto 5;
4:  goto 9;
5:  t1=i*i;
6:  sum=sum+t1;
7:  i=i+1;
8:  goto 3;
9:  print sum;
10; END
```

Basic Block

Example:

```
1:   i=1;
2:   sum=0;
3:   if i<=100 goto 5;
4:   goto 9;
5:   t1=i*i;
6:   sum=sum+t1;
7:   i=i+1;
8:   goto 3;
9:   print sum;
10;  END
```

Leader

Leader

Leader

Leader

Basic Block

Example:

```
1:   i=1;
2:   sum=0;
3:   if i<=100 goto 5;
4:   goto 9;
5:   t1=i*i;
6:   sum=sum+t1;
7:   i=i+1;
8:   goto 3;
9:   print sum;
10;  END
```

Leader

Leader

Leader

Leader

Leader

Basic Block

Example:

```
1:   i=1;
2:   sum=0;
3:   if i<=100 goto 5;
4:   goto 9;
5:   t1=i*i;
6:   sum=sum+t1;
7:   i=i+1;
8:   goto 3;
9:   print sum;
10;  END
```

B1

```
1:   i=1;
2:   sum=0;
```

B2

```
3:   if i<=100 goto 5;
```

B3

```
4:   goto 9;
```

B4

```
5:   t1=i*i;
6:   sum=sum+t1;
7:   i=i+1;
8:   goto 3;
```

B5

```
9:   print sum;
10;  END
```

Control Flow Graph

Edges in the Graph:

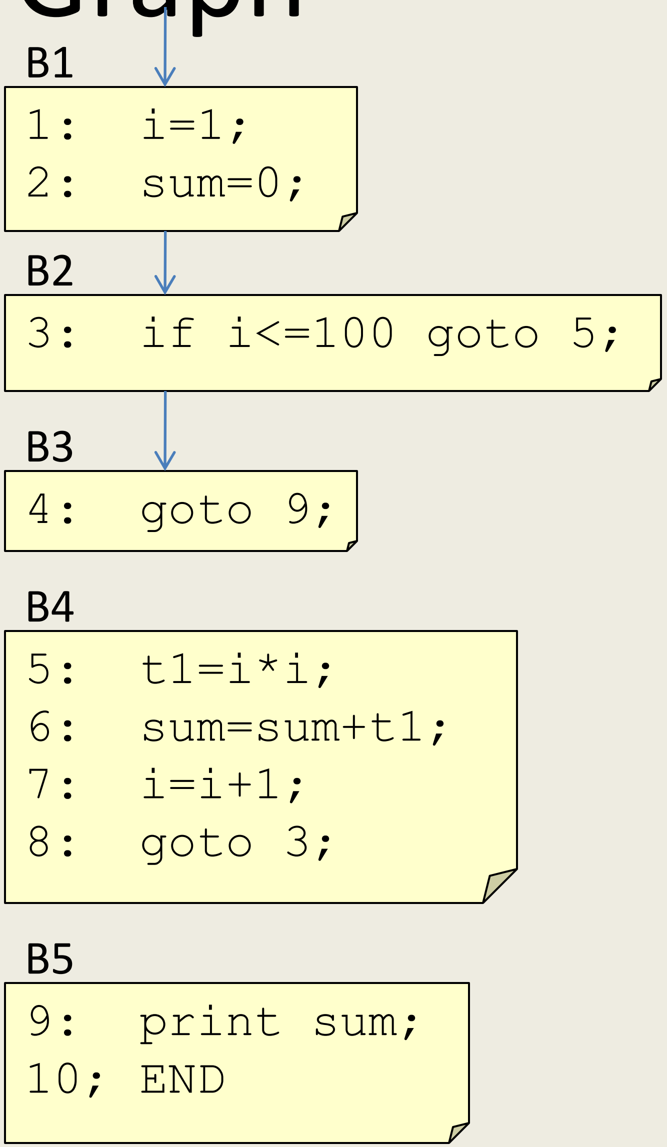
There is a directed edge from block Bi to block Bj, if

1. Bj immediately follows Bi in the order of the program and Bi does not end in an unconditional jump, OR
2. There is a conditional or unconditional jump from the last statement of Bi to the first statement of Bj.

Control Flow Graph

Example:

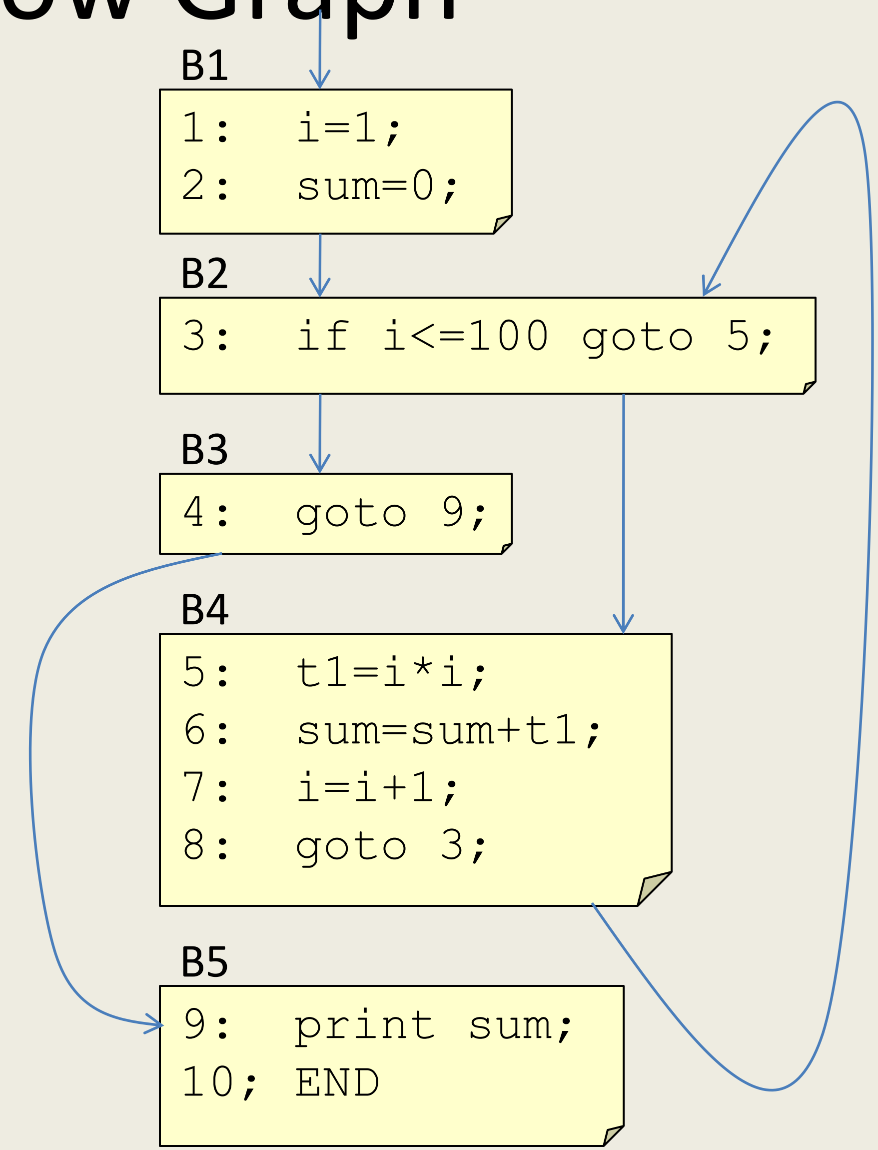
```
1:   i=1;
2:   sum=0;
3:   if i<=100 goto 5;
4:   goto 9;
5:   t1=i*i;
6:   sum=sum+t1;
7:   i=i+1;
8:   goto 3;
9:   print sum;
10;  END
```



Control Flow Graph

Example:

```
1:   i=1;
2:   sum=0;
3:   if i<=100 goto 5;
4:   goto 9;
5:   t1=i*i;
6:   sum=sum+t1;
7:   i=i+1;
8:   goto 3;
9:   print sum;
10;  END
```



Data Flow Analysis

- The Data Flow Analysis is carried out on the Control Flow Graph to perform Optimizations on the code.
- Two Types
  - Local Data Flow Analysis

    DAG (Directed Acyclic Graph)
  - Global Data Flow Analysis

    Reaching Definitions, Available Expressions, Live Variable

DAG (Directed Acyclic Graph)

- A DAG is a useful structure for analyzing a given basic block.
- Each node of a flow graph can be represented by a DAG.
- A DAG is a directed graph with no cycles.

DAG (Directed Acyclic Graph)

Properties:

- Leaves are labeled by identifiers, either variable names or constants.
- Internal nodes are labeled by operator symbols.
- Nodes are also optionally given an extra set of identifiers.

DAG (Directed Acyclic Graph)

Method of Construction:

- The following three-address statements are considered:
  i. A = B op C
  ii. A = op B
  iii. A = B
- Assuming the following function:

  NODE(Id) – returns the most recently created node associated with “Id”.

DAG (Directed Acyclic Graph)

Method of Construction (Steps):

1. If NODE(B) is undefined, create a leaf node labeled B, and let NODE(B) be this node. In case (i), if NODE(C) is undefined, create a leaf node labeled C, and let NODE(C) be this node.

DAG (Directed Acyclic Graph)

Method of Construction (Steps):

2. <u>Case (i)</u>: Determine if there is a node labeled “op”, whose left child is NODE(B) and right child is NODE(C). If not, create such a node. In either event, let n be the node found or created.

   <u>Case (ii)</u>: Determine if there is a node labeled “op”, whose lone child is NODE(B). If not, create such a node. In either event, let n be the node found or created.

   <u>Case (iii)</u>: Let n be the NODE(B).

DAG (Directed Acyclic Graph)

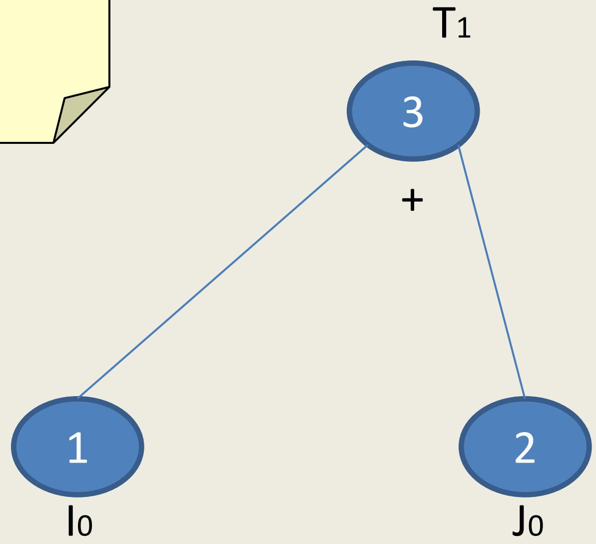
Method of Construction (Steps):

3. Append A to the list of attached identifiers for the node n found or created in Step 2. Delete A from the list of attached identifiers for NODE(A). Finally, set NODE(A) to n.

DAG (Directed Acyclic Graph)

Example:

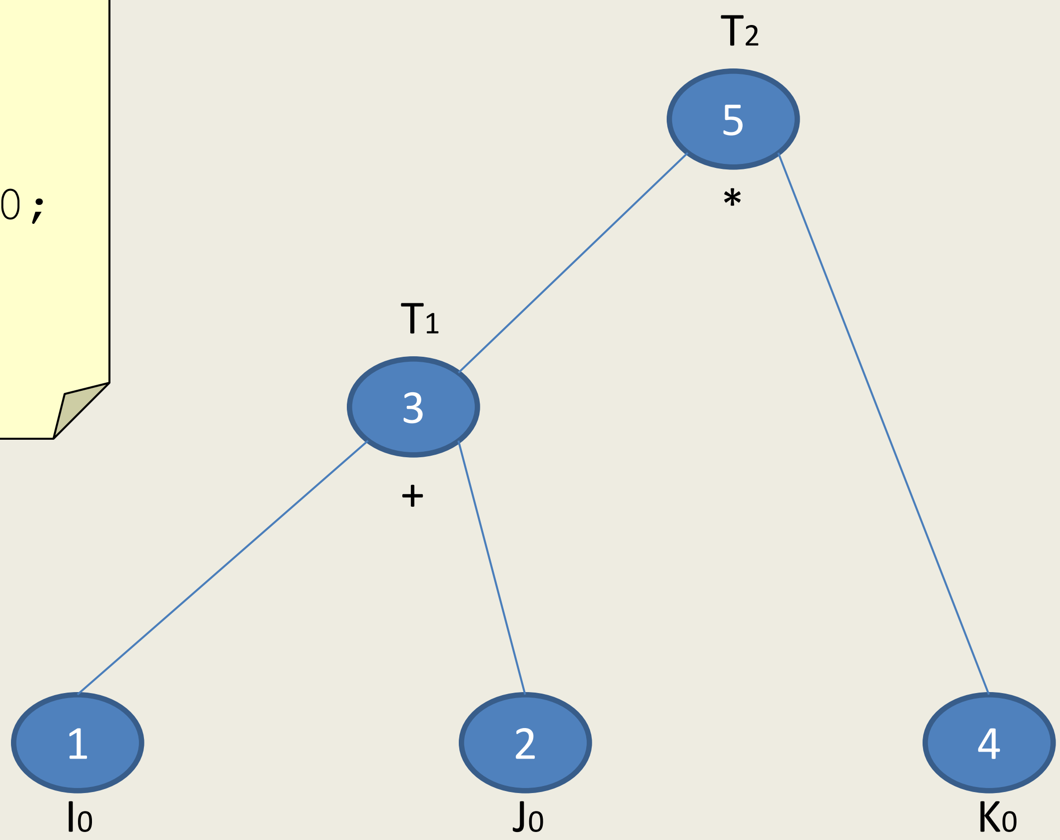
```
1:  T1=I+J;
2:  T2=T1*K;
3:  A=T2;
4:  K=K+10;
5:  T3=I+J;
6:  T4=T3-20;
7:  B=T4;
8:  I=I+1;
9:  J=J+4;
```



DAG (Directed Acyclic Graph)

Example:

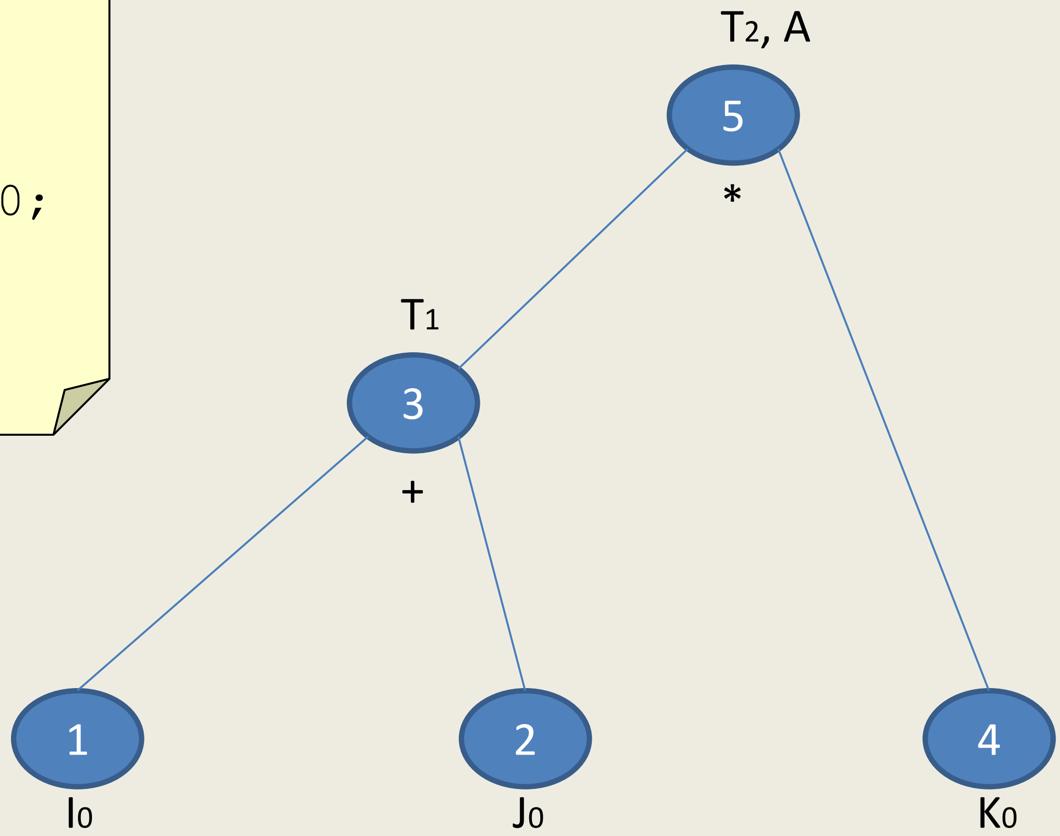
```
1:  T1=I+J;
2:  T2=T1*K;
3:  A=T2;
4:  K=K+10;
5:  T3=I+J;
6:  T4=T3-20;
7:  B=T4;
8:  I=I+1;
9:  J=J+4;
```



DAG (Directed Acyclic Graph)

Example:

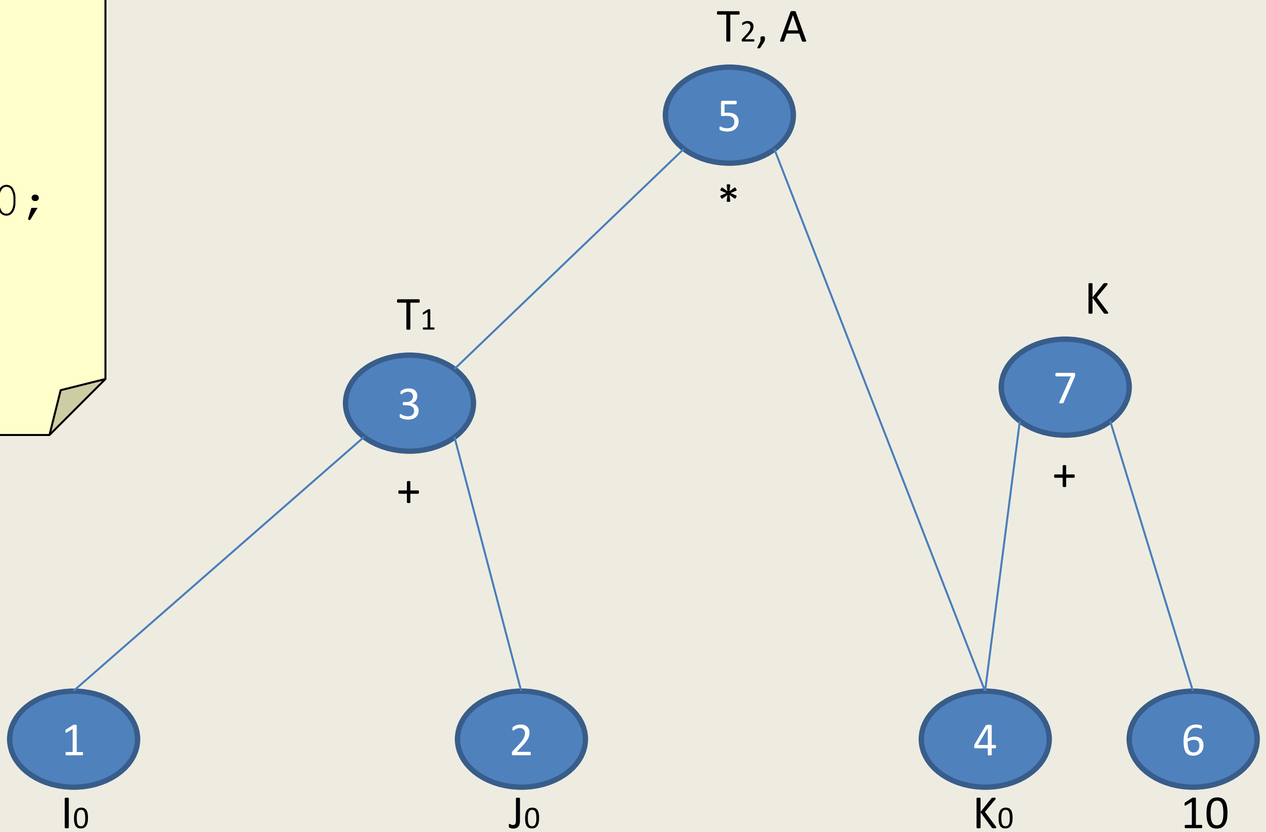
```
1:  T1=I+J;
2:  T2=T1*K;
3:  A=T2;
4:  K=K+10;
5:  T3=I+J;
6:  T4=T3-20;
7:  B=T4;
8:  I=I+1;
9:  J=J+4;
```



DAG (Directed Acyclic Graph)

Example:

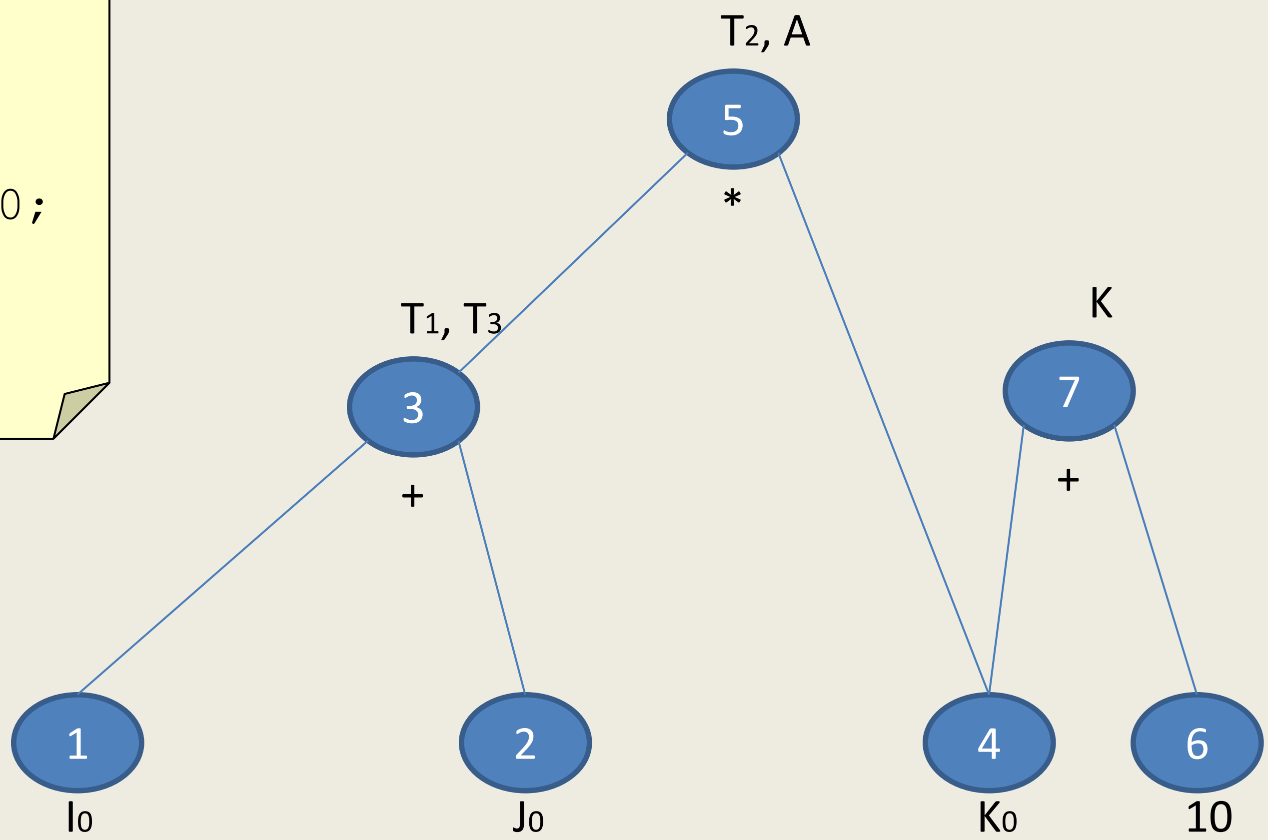
```
1:   T1=I+J;
2:   T2=T1*K;
3:   A=T2;
4:   K=K+10;
5:   T3=I+J;
6:   T4=T3-20;
7:   B=T4;
8:   I=I+1;
9:   J=J+4;
```



DAG (Directed Acyclic Graph)

Example:

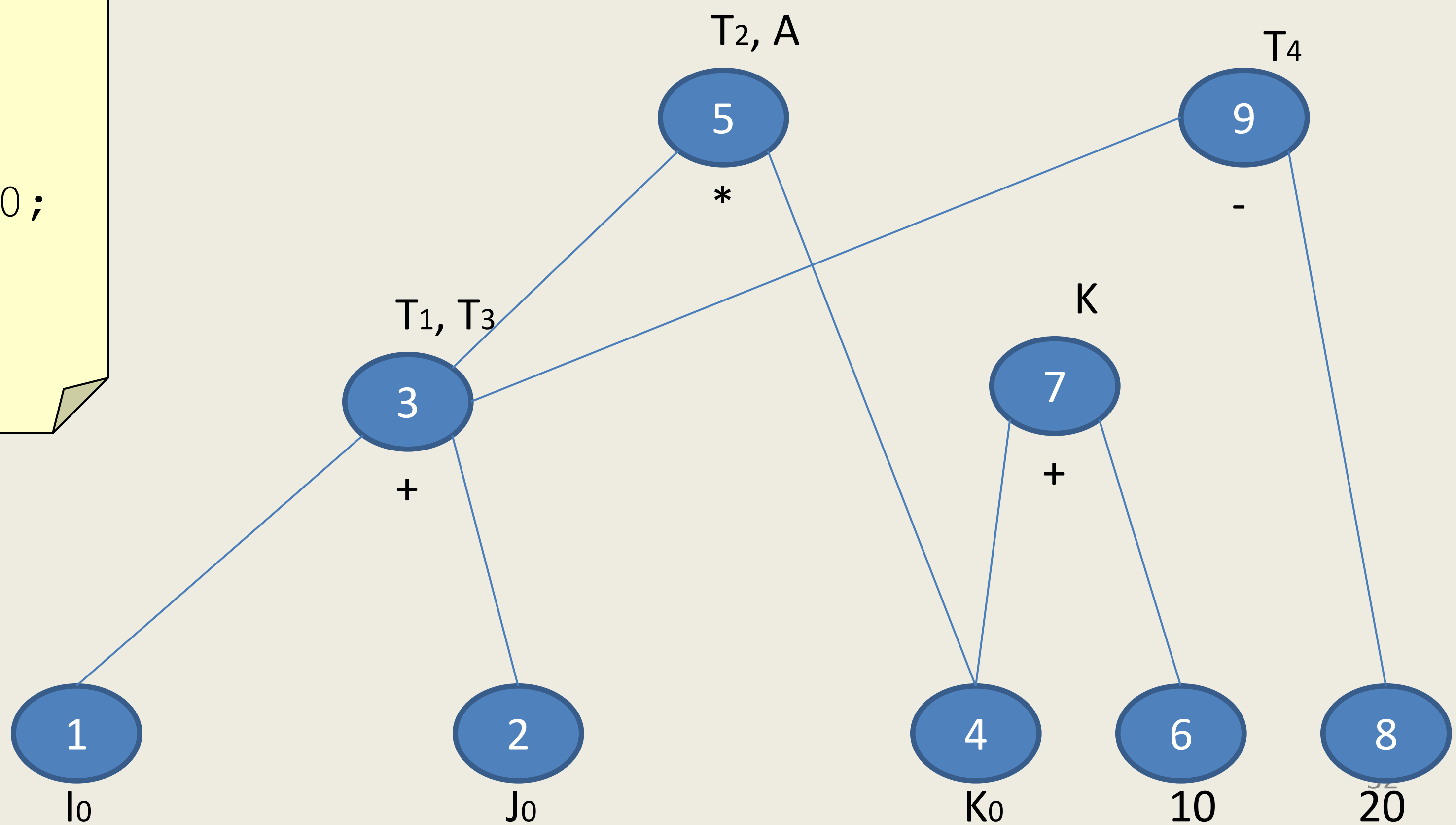
```
1:  T1=I+J;
2:  T2=T1*K;
3:  A=T2;
4:  K=K+10;
5:  T3=I+J;
6:  T4=T3-20;
7:  B=T4;
8:  I=I+1;
9:  J=J+4;
```



DAG (Directed Acyclic Graph)

Example:

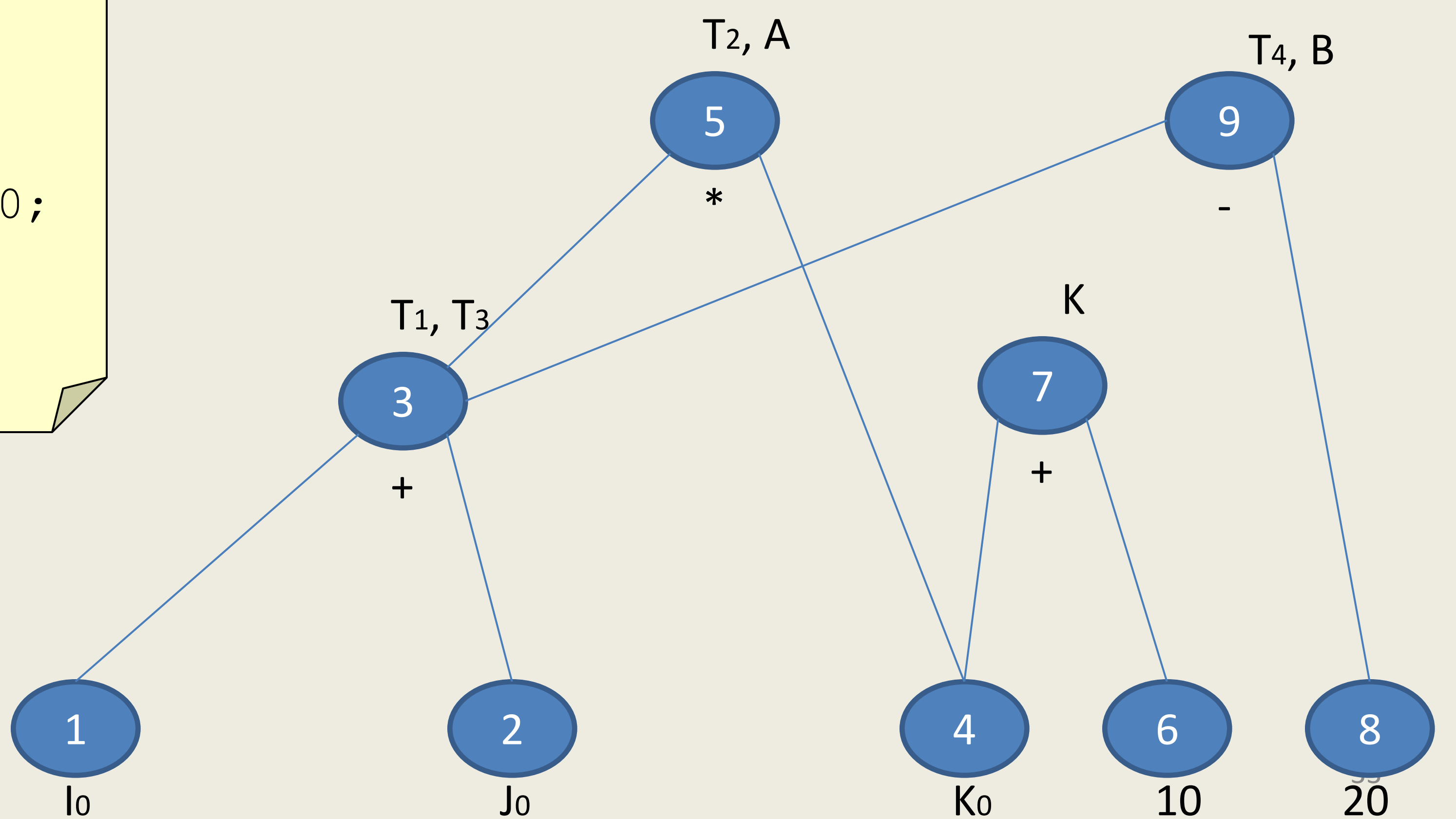
```
1:  T1=I+J;
2:  T2=T1*K;
3:  A=T2;
4:  K=K+10;
5:  T3=I+J;
6:  T4=T3-20;
7:  B=T4;
8:  I=I+1;
9:  J=J+4;
```



DAG (Directed Acyclic Graph)

Example:

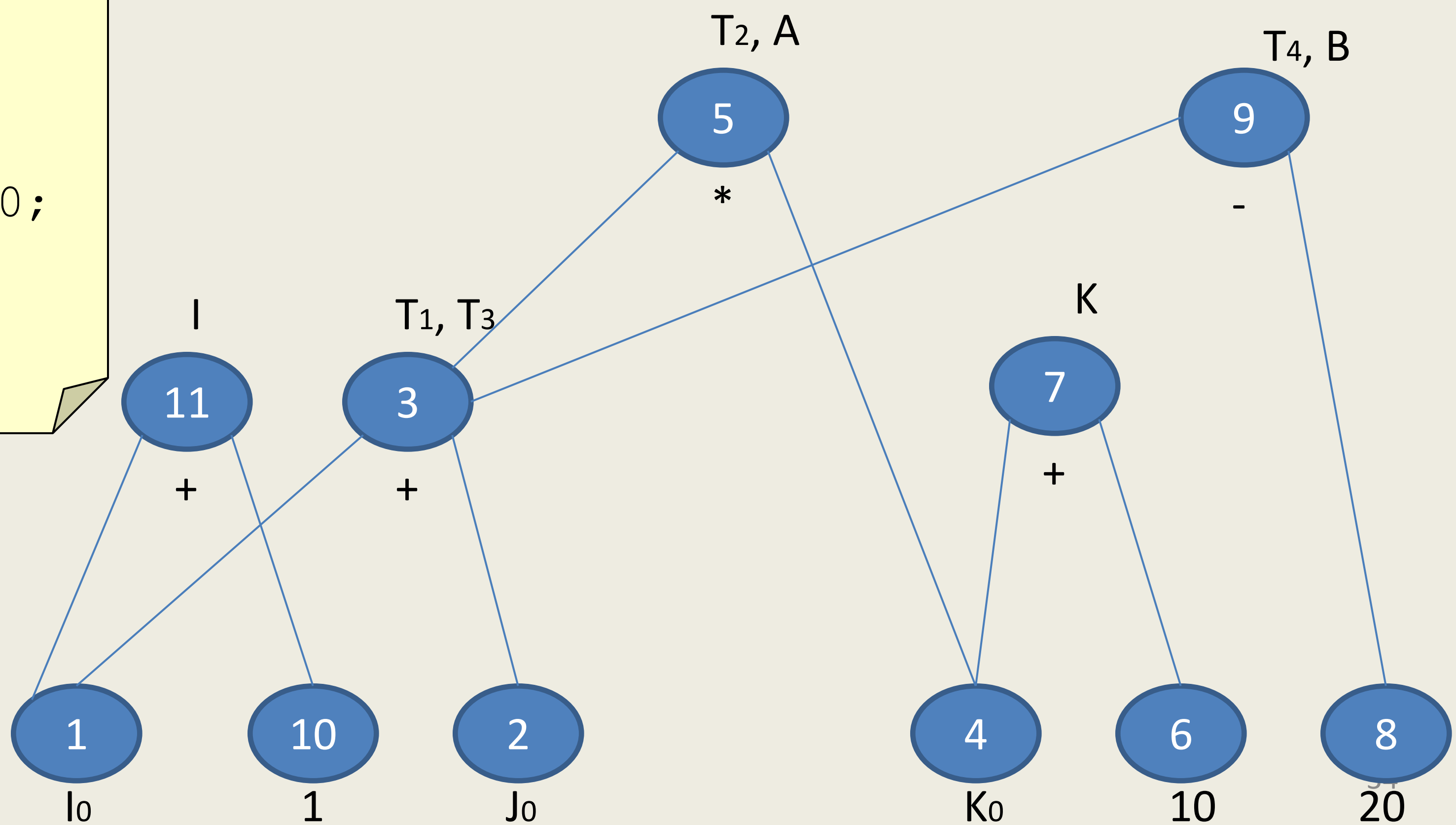
```
1:  T1=I+J;
2:  T2=T1*K;
3:  A=T2;
4:  K=K+10;
5:  T3=I+J;
6:  T4=T3-20;
7:  B=T4;
8:  I=I+1;
9:  J=J+4;
```



DAG (Directed Acyclic Graph)

Example:

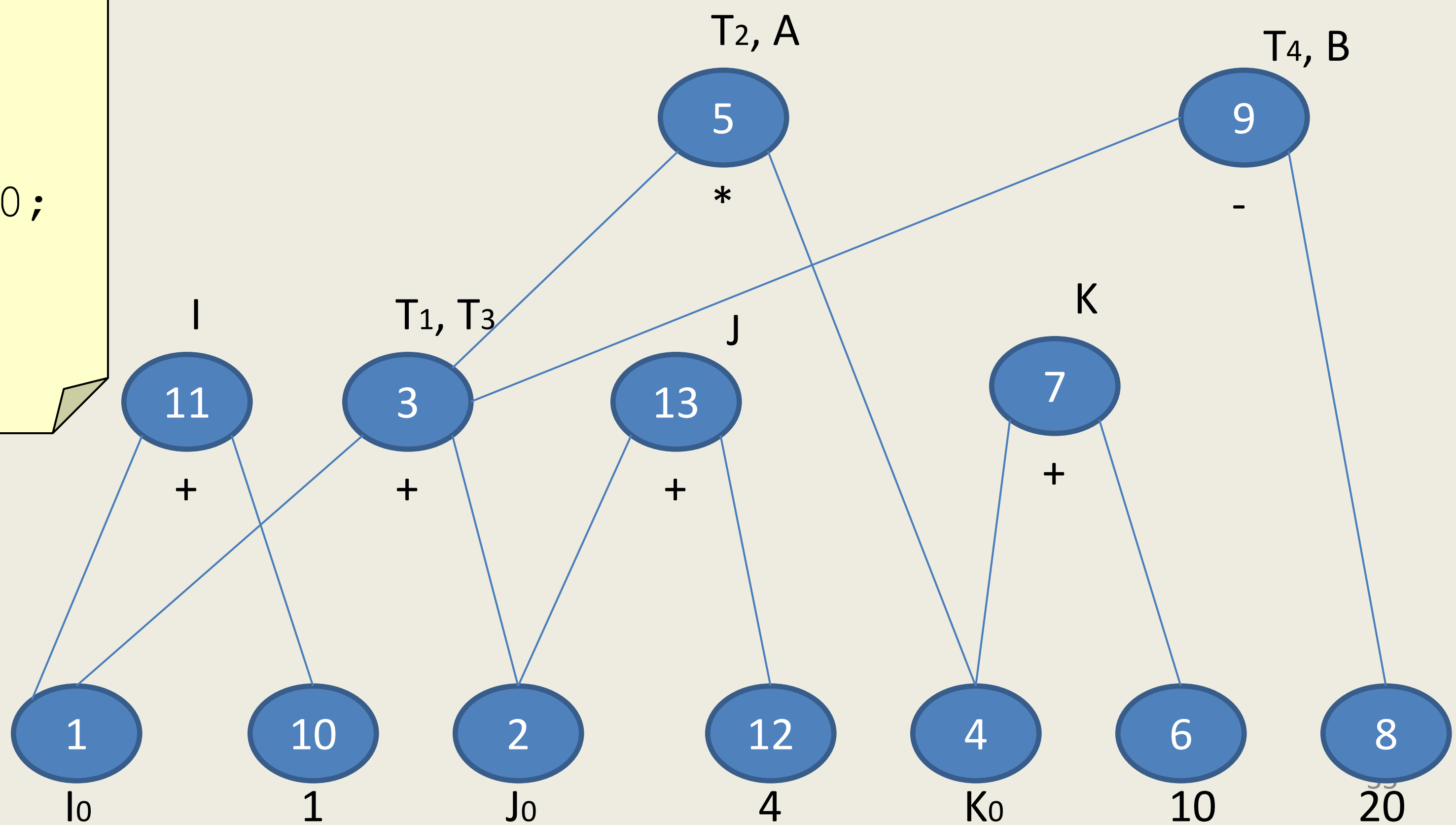
```
1:   T1=I+J;
2:   T2=T1*K;
3:   A=T2;
4:   K=K+10;
5:   T3=I+J;
6:   T4=T3-20;
7:   B=T4;
8:   I=I+1;
9:   J=J+4;
```



DAG (Directed Acyclic Graph)

Example:

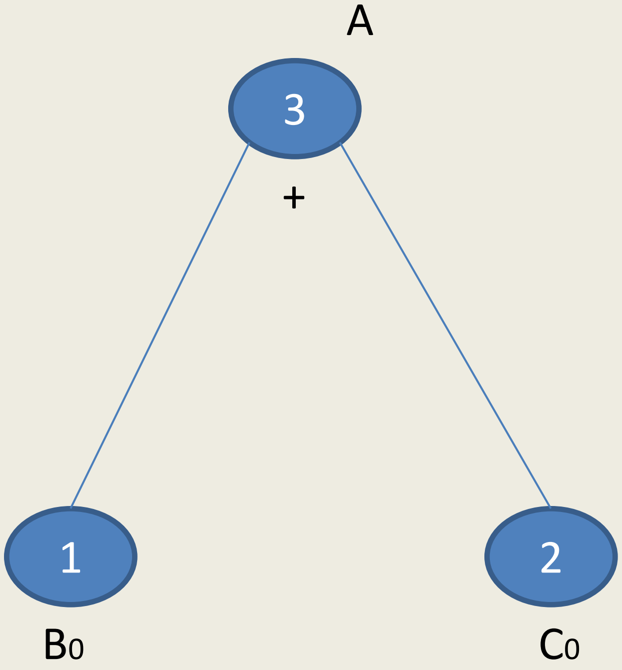
```
1:   T1=I+J;
2:   T2=T1*K;
3:   A=T2;
4:   K=K+10;
5:   T3=I+J;
6:   T4=T3-20;
7:   B=T4;
8:   I=I+1;
9:   J=J+4;
```



DAG (Directed Acyclic Graph)

Example:

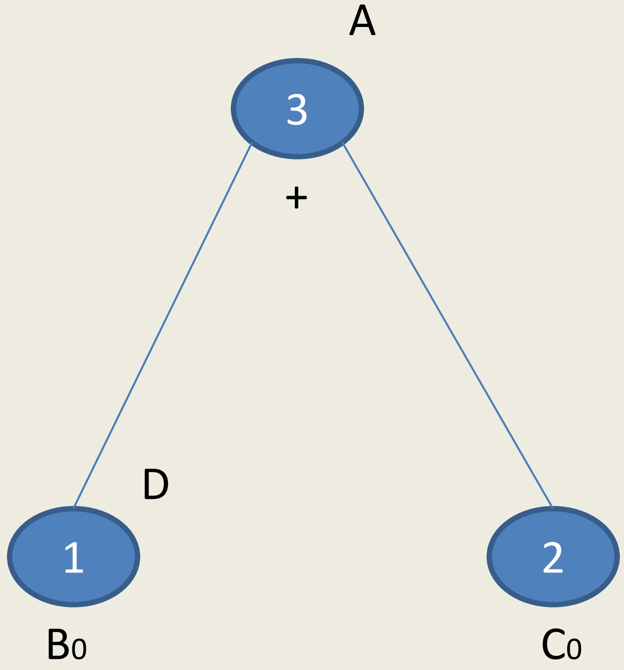
```
1:  A=B+C;
```



DAG (Directed Acyclic Graph)

Example:

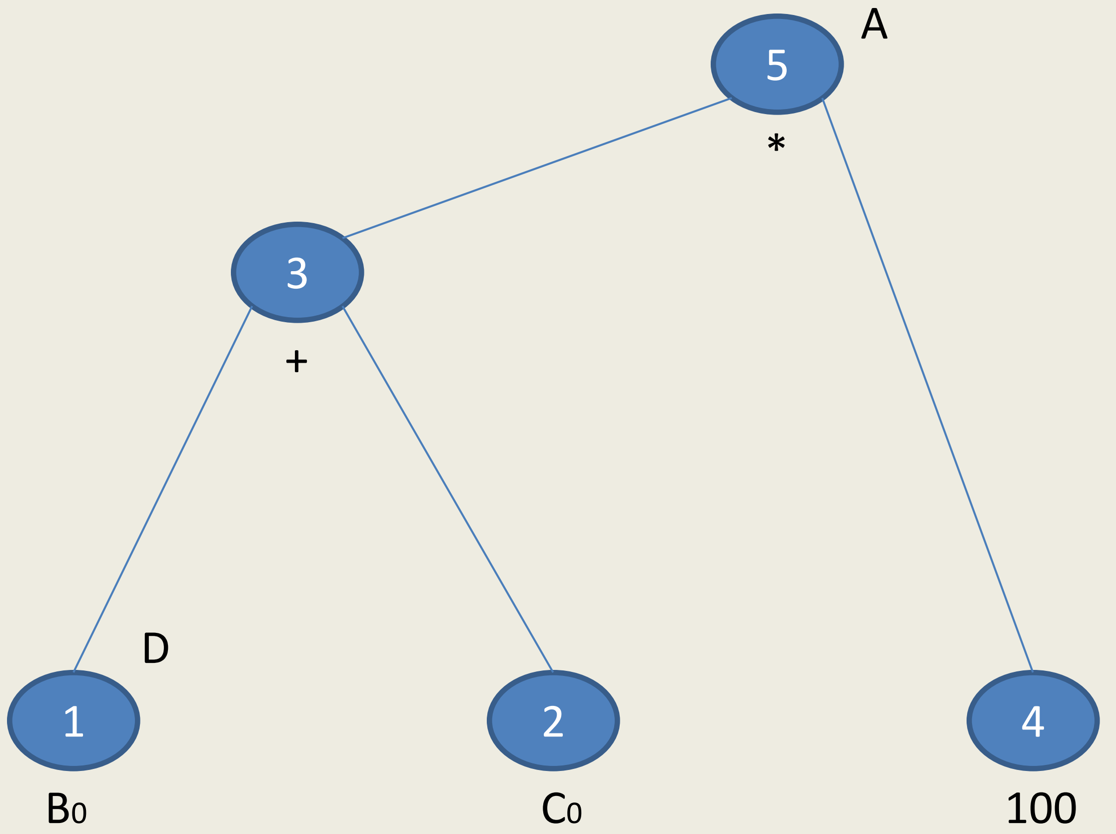
```
1:   A=B+C;
2:   D=B;
```



DAG (Directed Acyclic Graph)

Example:

```
1:   A=B+C;
2:   D=B;
3:   A=A*100;
```



Application of DAG

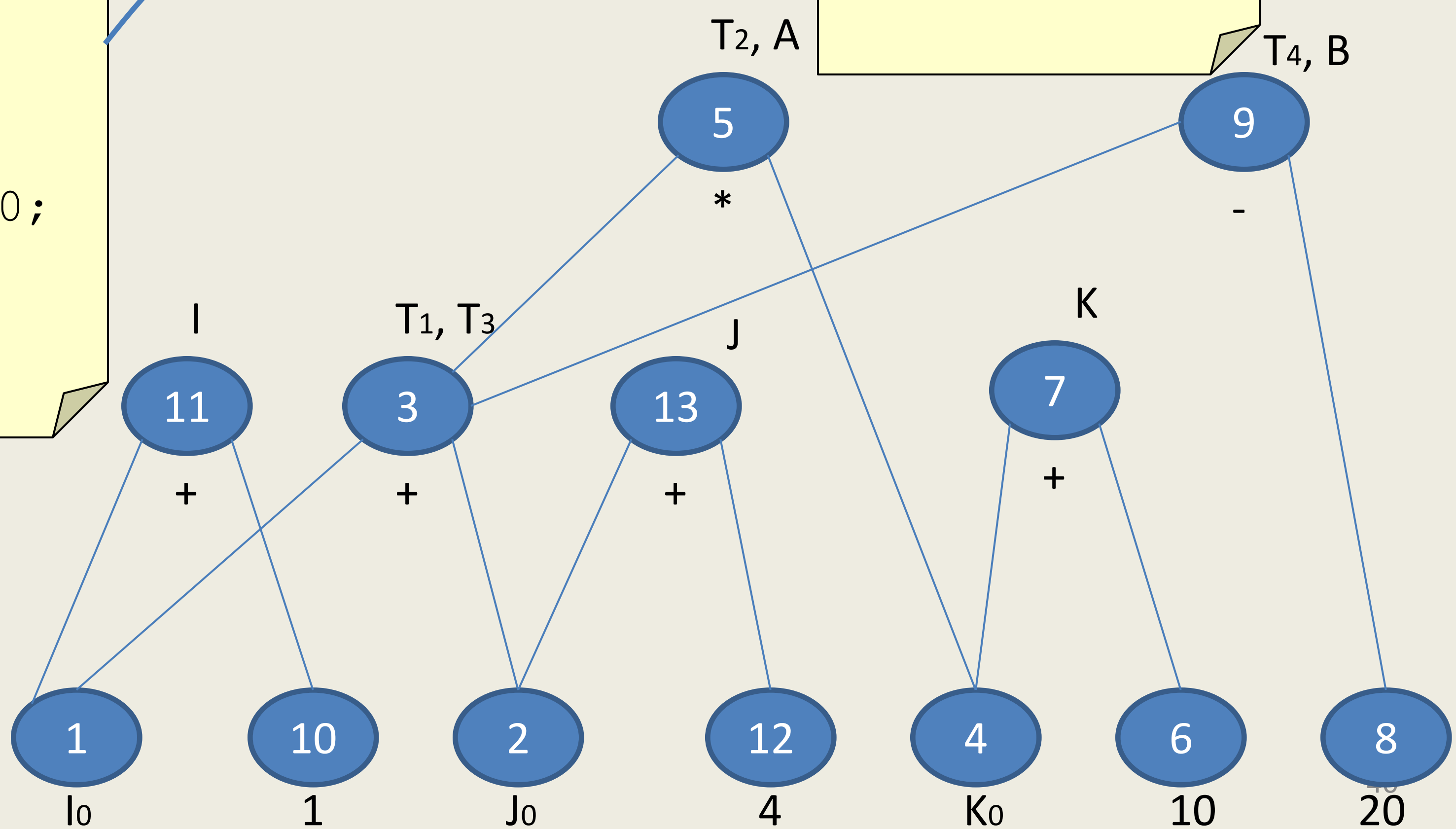
- A simplified list of sequence of three-address statements can be constructed taking advantage of Common Sub-expressions and not performing copy statements of the form A=B unless absolutely necessary.
- Whenever a node has more than one identifiers on its attached list, we can check which of these identifiers are needed outside the block.

Application of DAG

Example:

```
1:   T1=I+J;
2:   T2=T1*K;
3:   A=T2;
4:   K=K+10;
5:   T3=I+J;
6:   T4=T3-20;
7:   B=T4;
8:   I=I+1;
9:   J=J+4;
```

```
1:   T1=I+J;
```

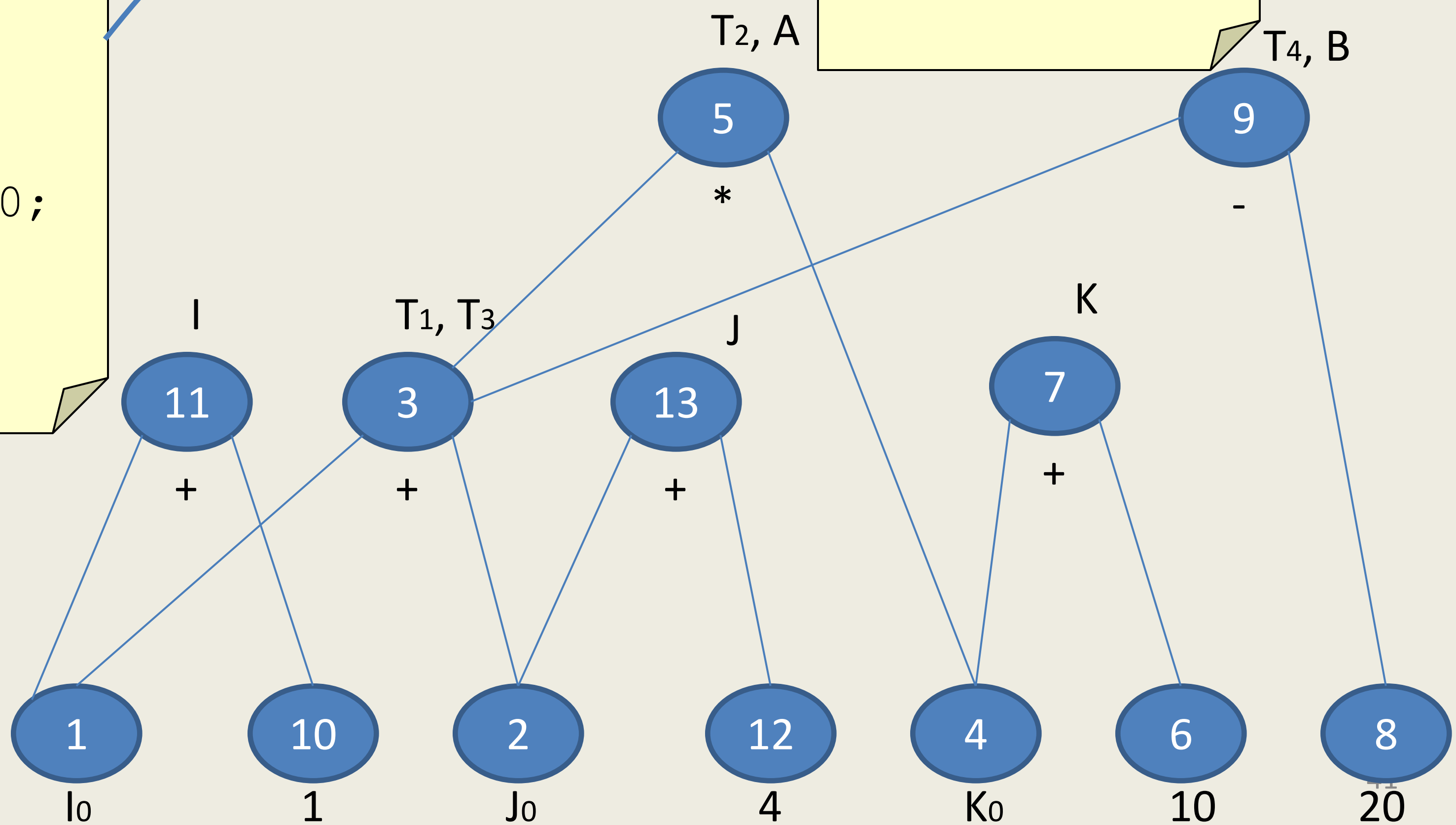


Application of DAG

Example:

```
1:   T1=I+J;
2:   T2=T1*K;
3:   A=T2;
4:   K=K+10;
5:   T3=I+J;
6:   T4=T3-20;
7:   B=T4;
8:   I=I+1;
9:   J=J+4;
```

```
1:   T1=I+J;
2:   A=T1*K;
```

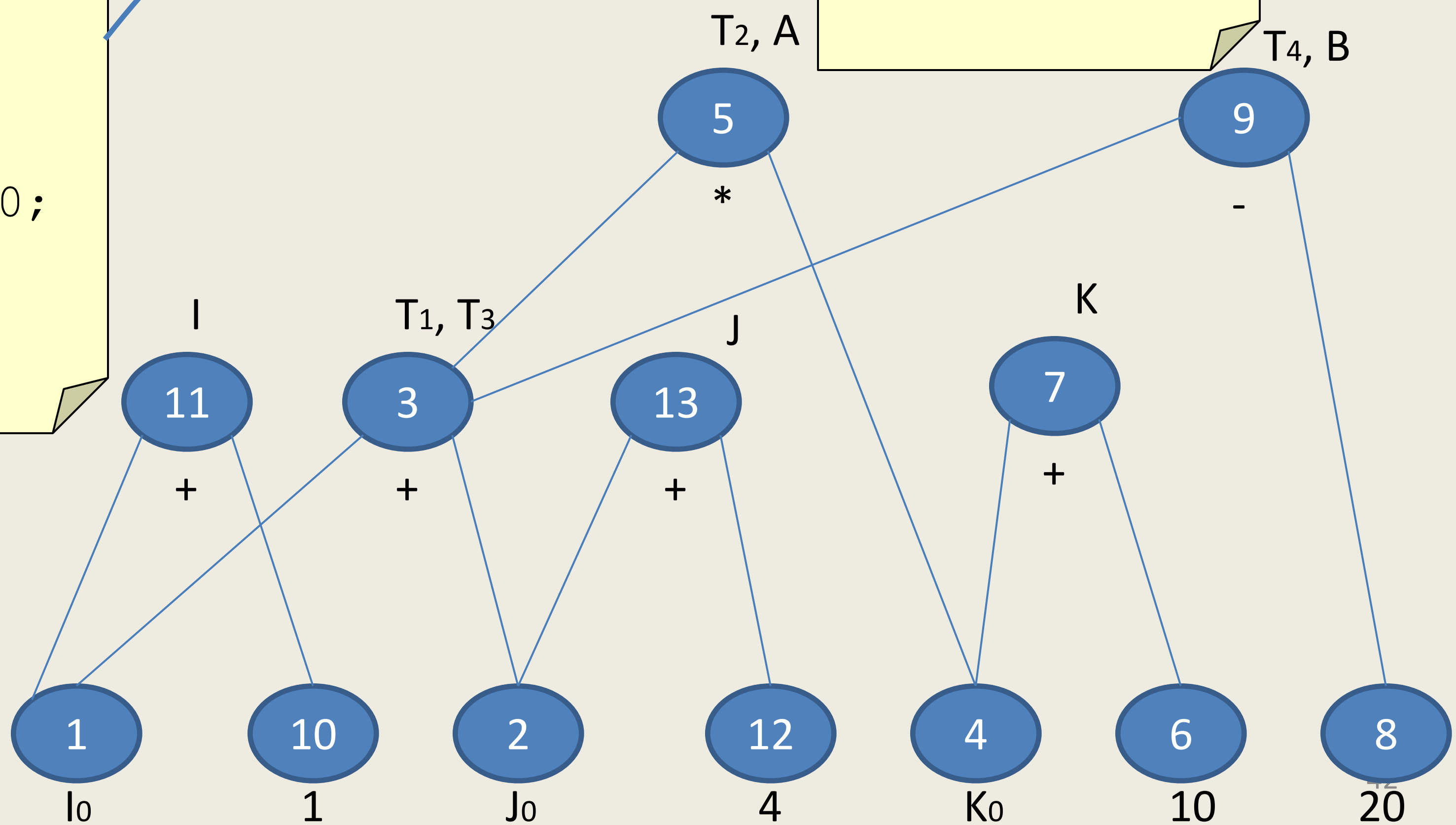


Application of DAG

Example:

```
1:   T1=I+J;
2:   T2=T1*K;
3:   A=T2;
4:   K=K+10;
5:   T3=I+J;
6:   T4=T3-20;
7:   B=T4;
8:   I=I+1;
9:   J=J+4;
```

```
1:   T1=I+J;
2:   A=T1*K;
3:   K=K+10;
```

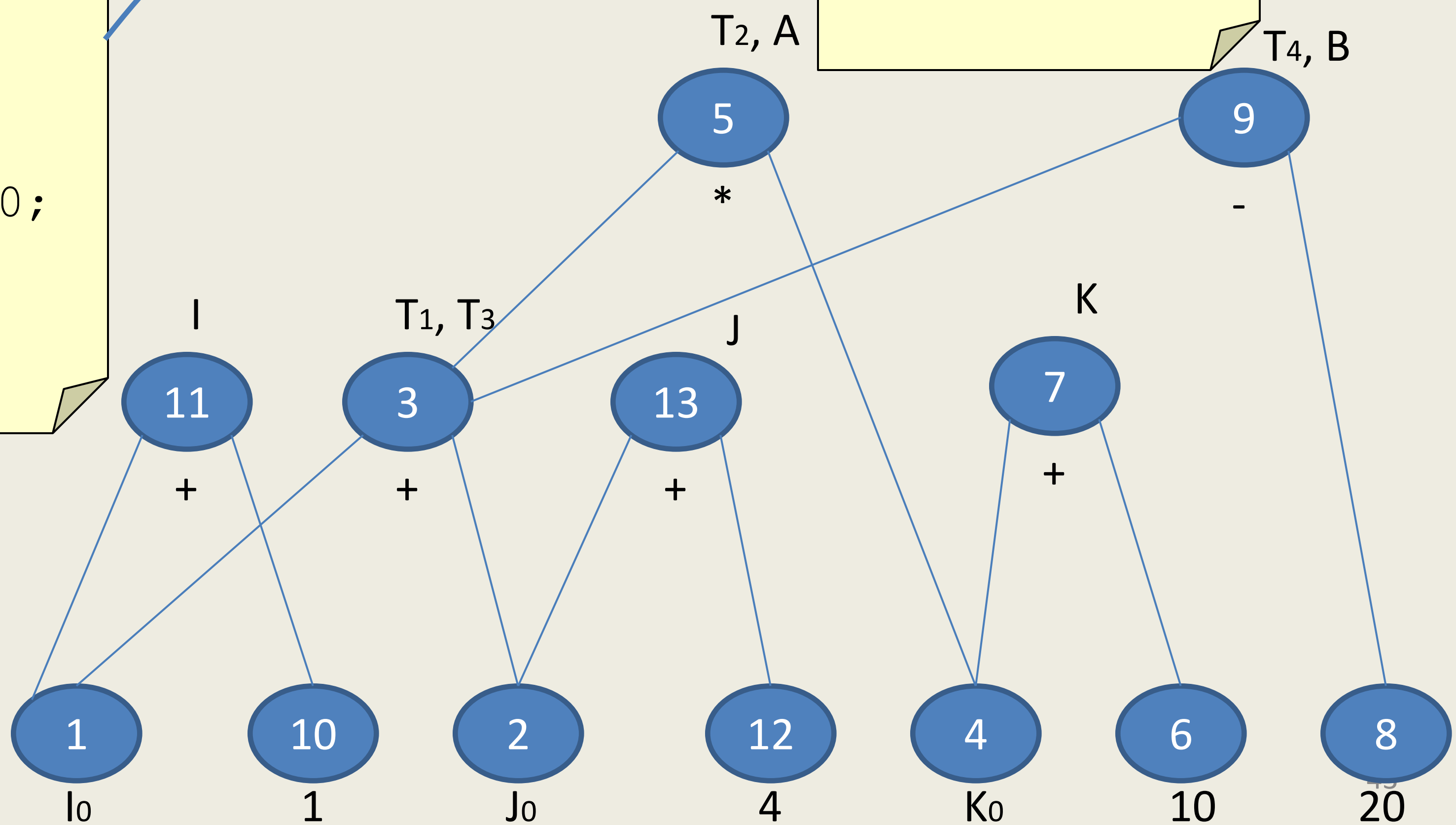


Application of DAG

Example:

```
1:   T1=I+J;
2:   T2=T1*K;
3:   A=T2;
4:   K=K+10;
5:   T3=I+J;
6:   T4=T3-20;
7:   B=T4;
8:   I=I+1;
9:   J=J+4;
```

```
1:   T1=I+J;
2:   A=T1*K;
3:   K=K+10;
4:   B=T1-20;
```

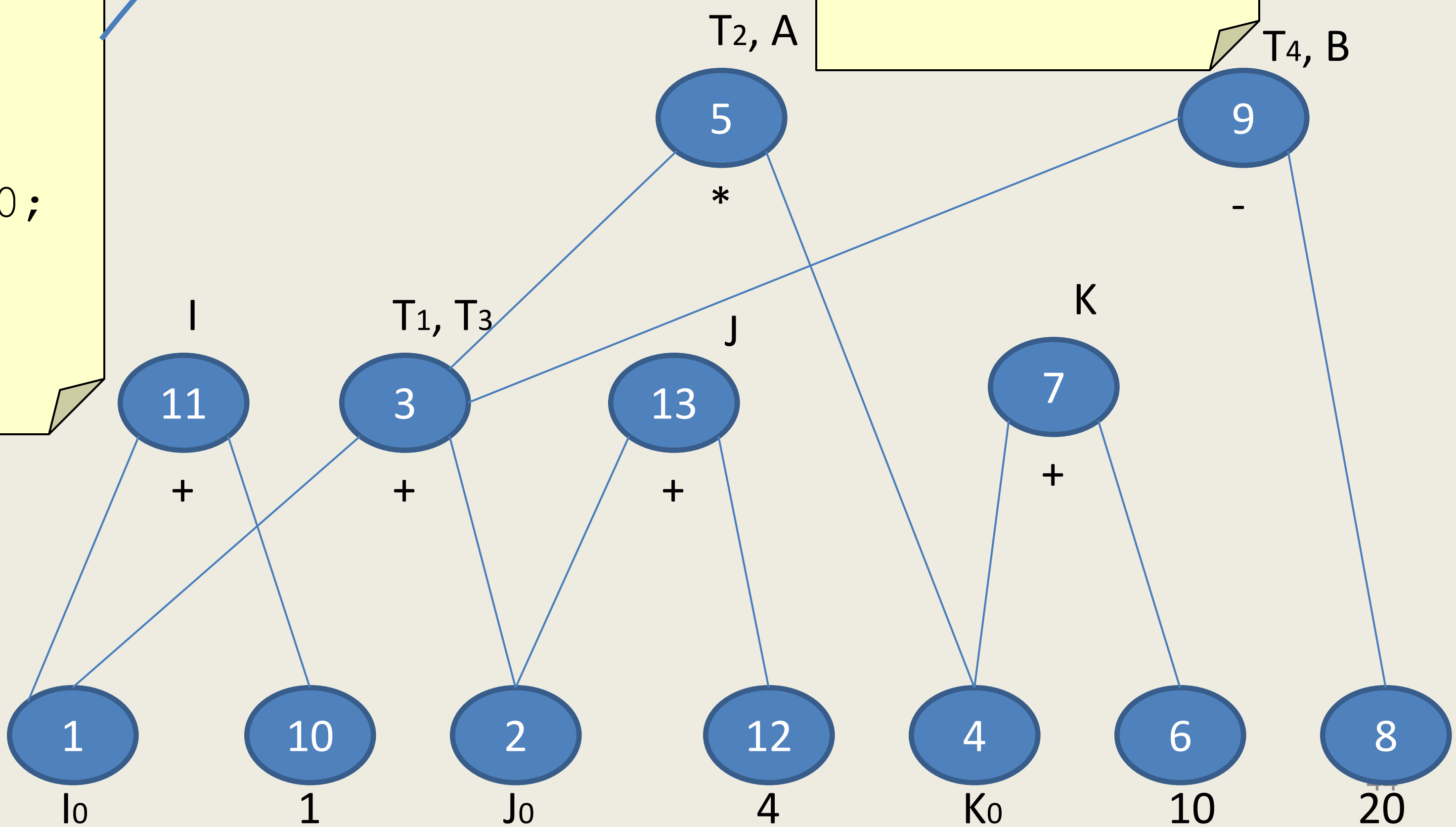


Application of DAG

Example:

```
1:   T1=I+J;
2:   T2=T1*K;
3:   A=T2;
4:   K=K+10;
5:   T3=I+J;
6:   T4=T3-20;
7:   B=T4;
8:   I=I+1;
9:   J=J+4;
```

```
1:   T1=I+J;
2:   A=T1*K;
3:   K=K+10;
4:   B=T1-20;
5:   I=I+1;
```

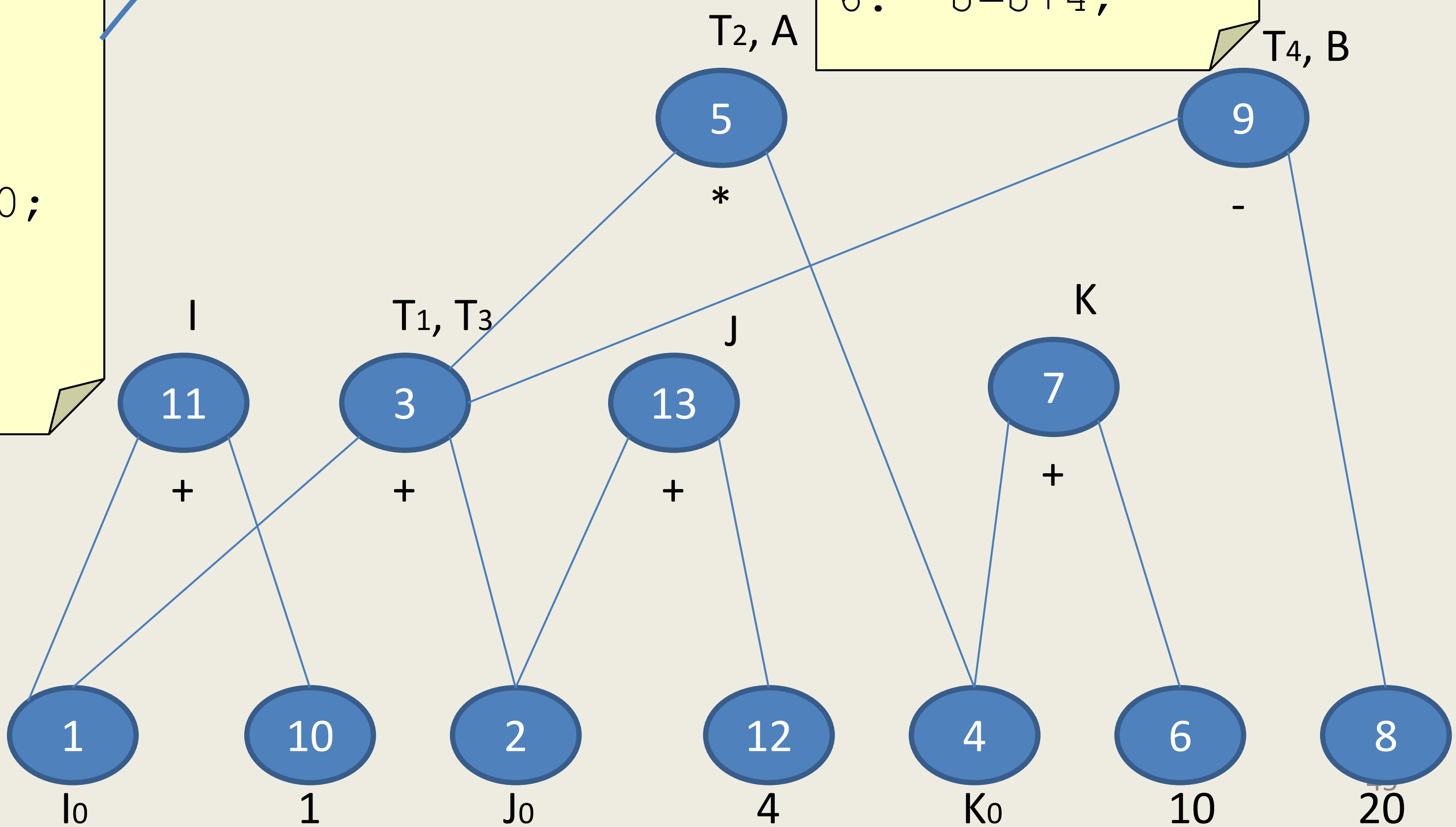


Application of DAG

Example:

```
1:  T1=I+J;
2:  T2=T1*K;
3:  A=T2;
4:  K=K+10;
5:  T3=I+J;
6:  T4=T3-20;
7:  B=T4;
8:  I=I+1;
9:  J=J+4;
```

```
1:  T1=I+J;
2:  A=T1*K;
3:  K=K+10;
4:  B=T1-20;
5:  I=I+1;
6:  J=J+4;
```



Reaching Definitions

A Definition of an identifier A reaches a point p if there is a path in the flow graph from that definition to p such that no other definitions of A appear on the path.

Reaching Definitions

- GEN[B] : is the set of generated definitions within block B that reach end of the block.
- KILL[B] : is the set of definitions outside of B that define identifiers that also have definitions within B.

Reaching Definitions

- IN[B] : is the set of all definitions reaching the point just before the first statement of block B.
- OUT[B] : is the set of all definitions reaching the point just after the last statement of block B.

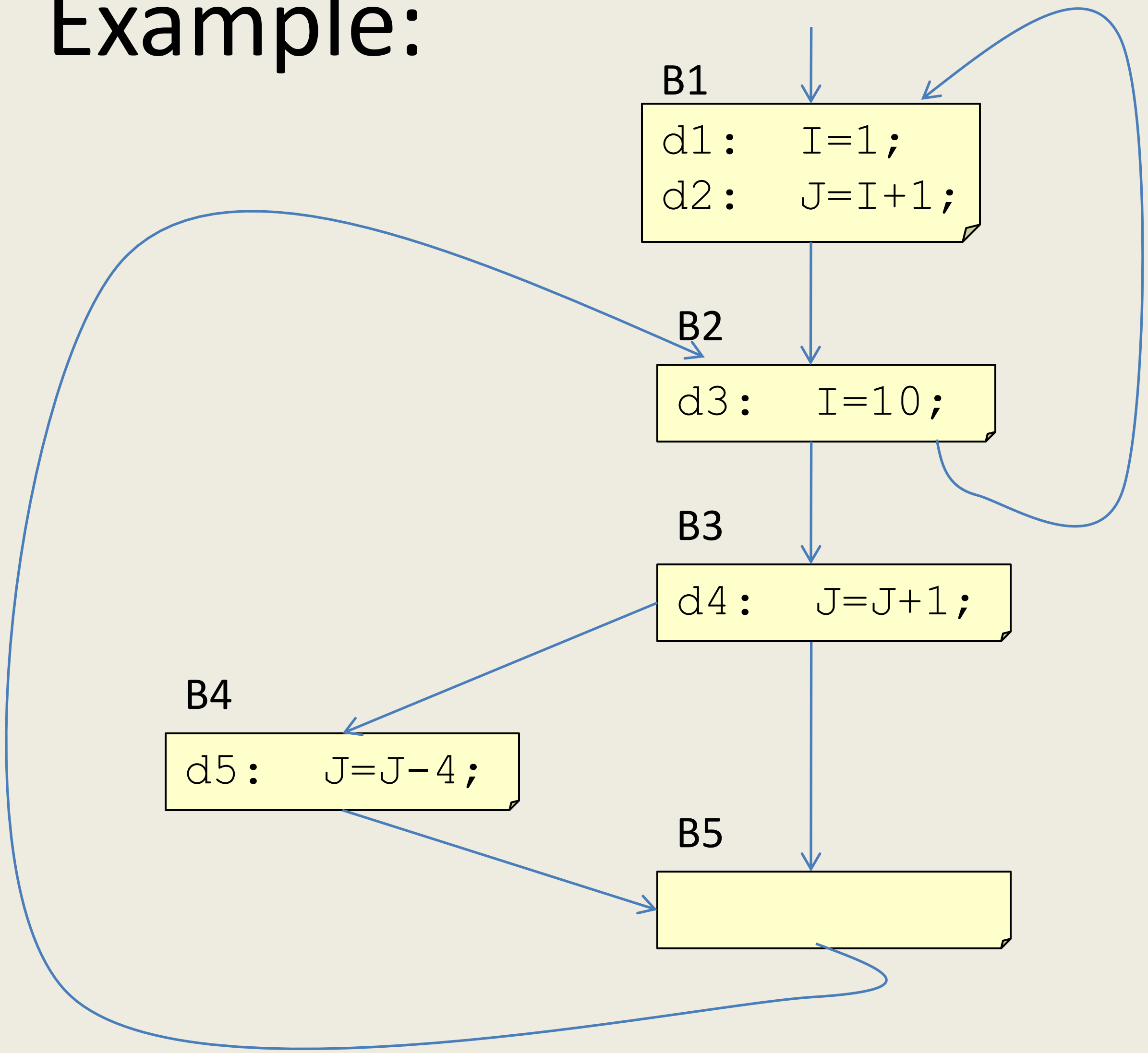
Reaching Definitions

Data Flow Equations :

- $OUT[B] = (IN[B] - KILL[B]) \cup GEN[B]$
- $IN[B] = \bigcup_{\text{P a predecessor of B}} OUT[P]$

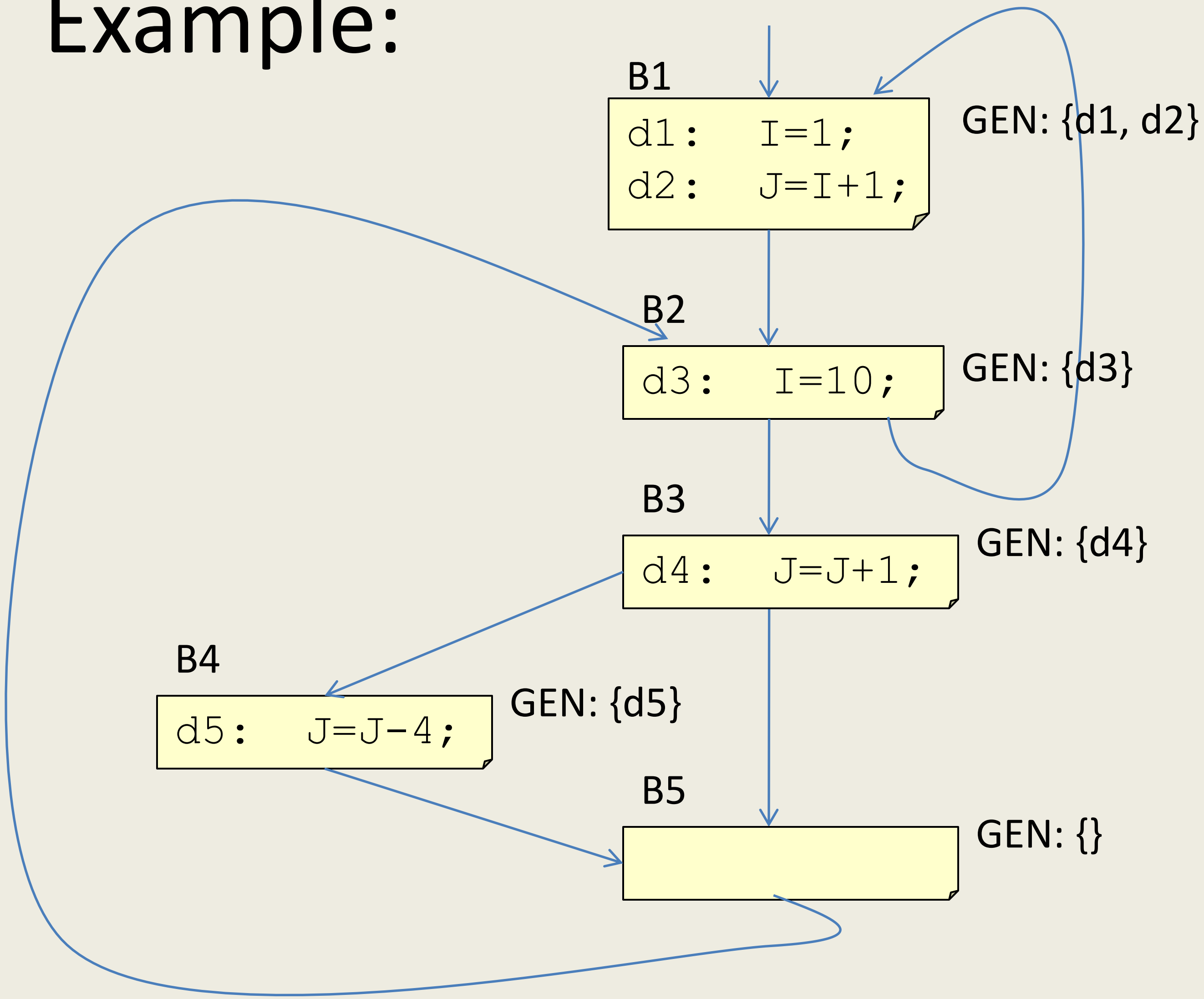
Reaching Definitions

Example:



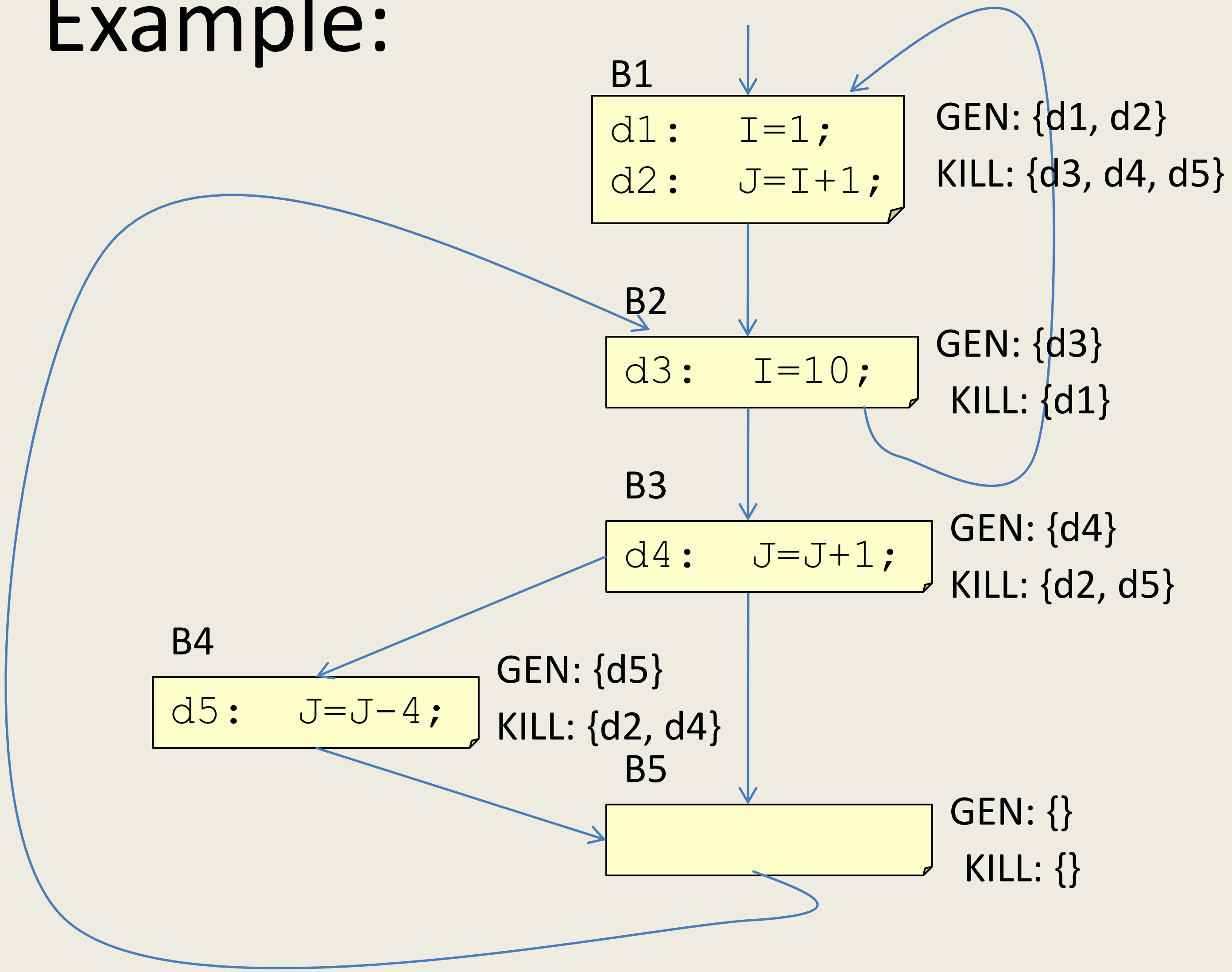
Reaching Definitions

Example:



Reaching Definitions

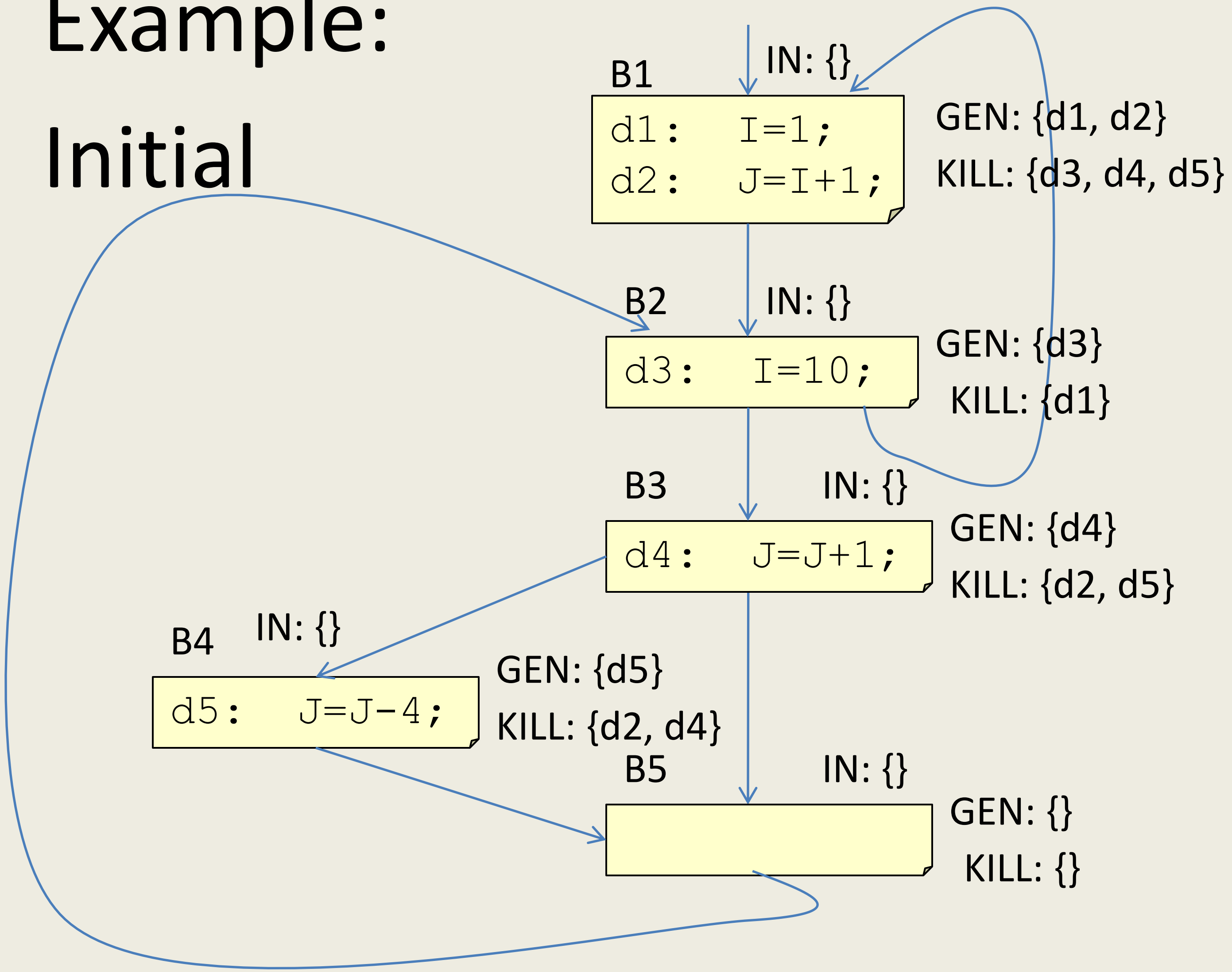
Example:



Reaching Definitions

Example:

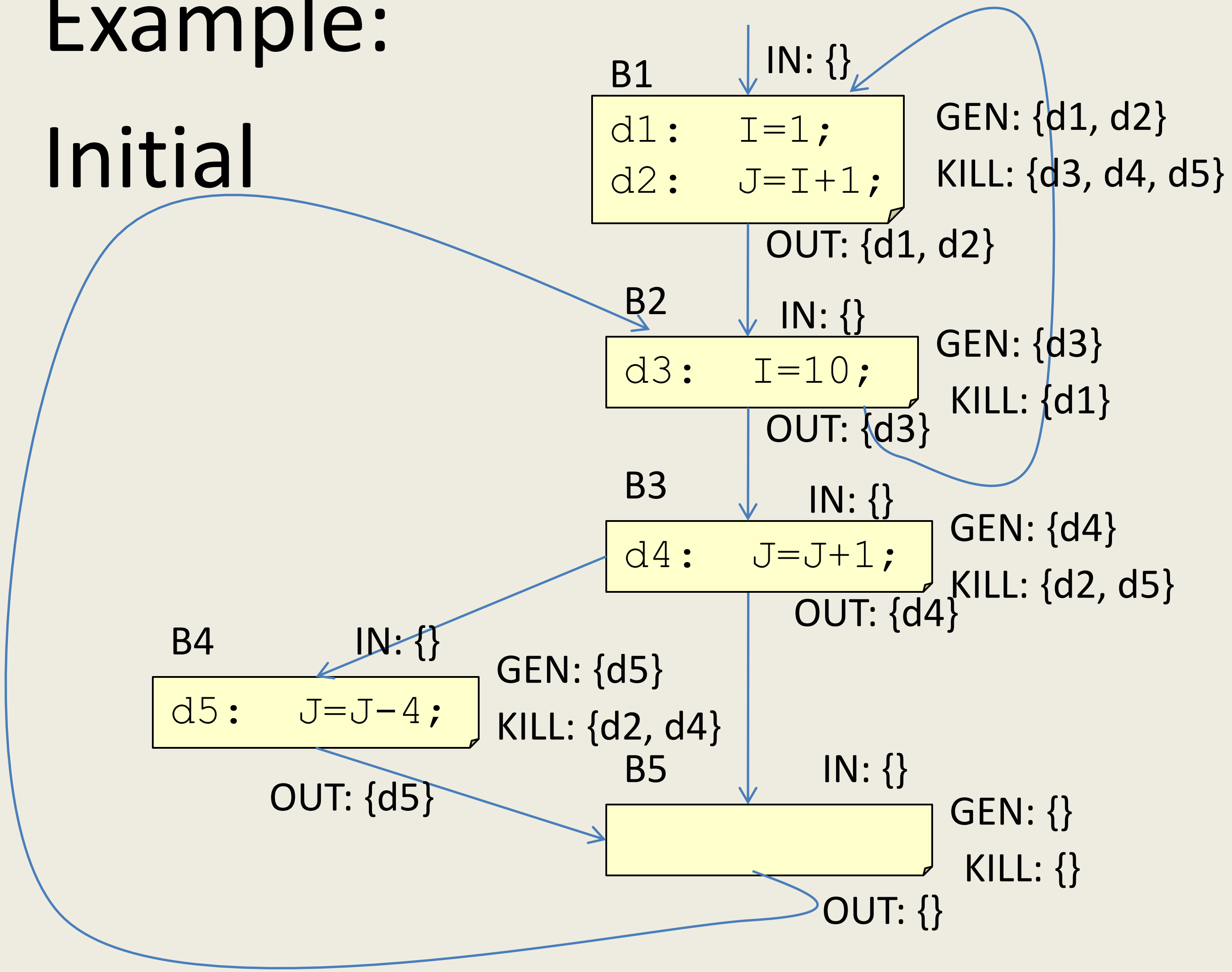
Initial



Reaching Definitions

Example:

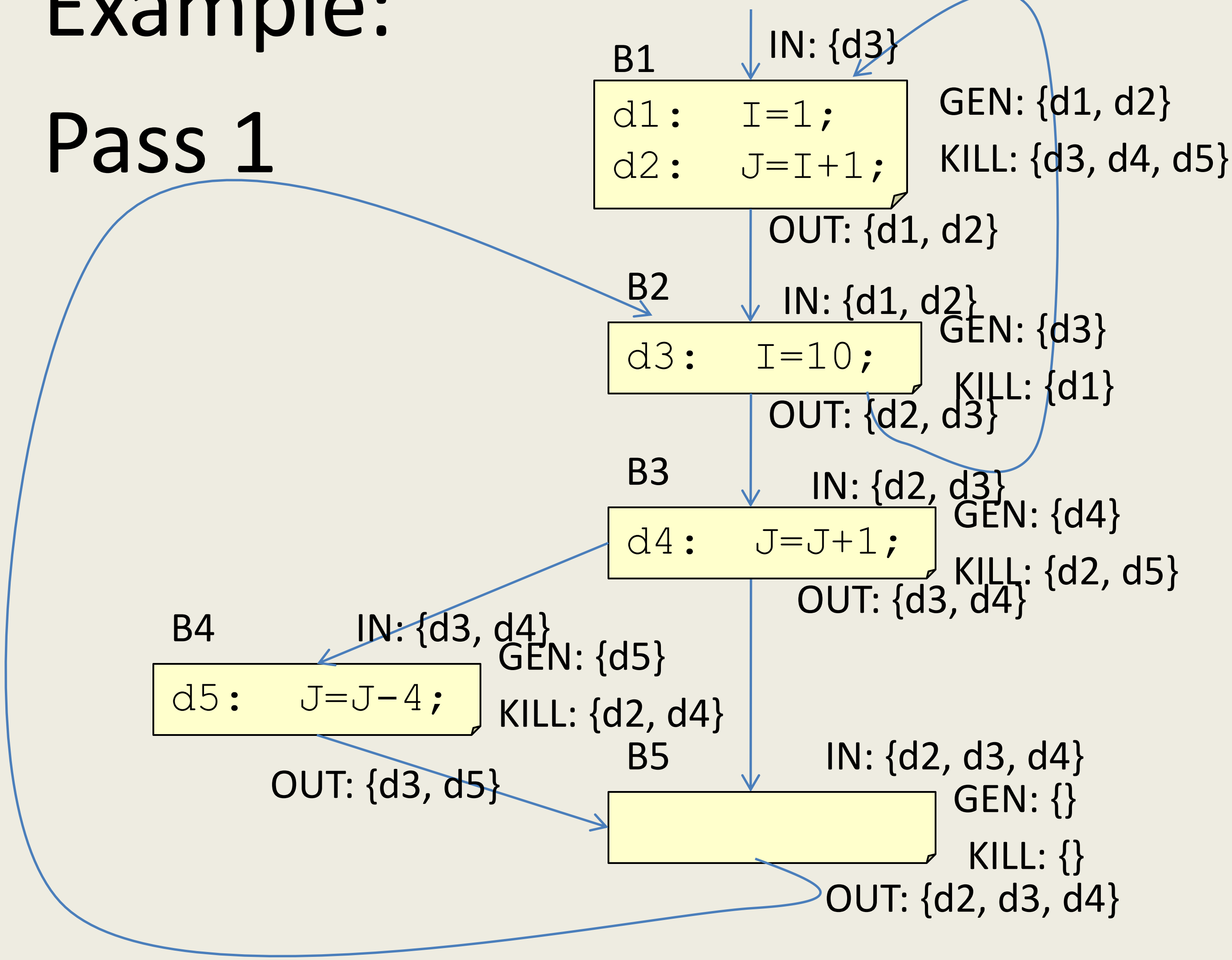
Initial



Reaching Definitions

Example:

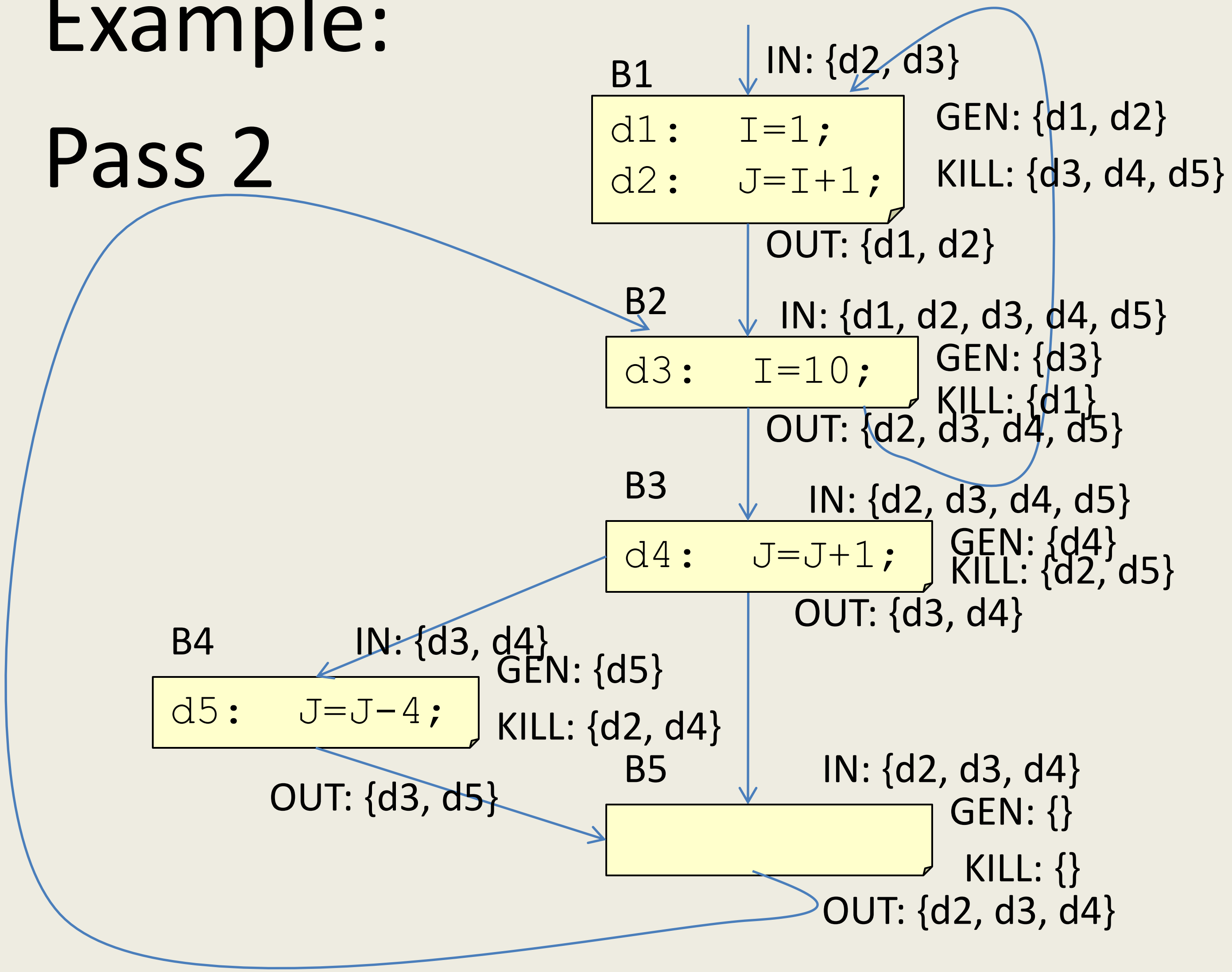
Pass 1



Reaching Definitions

Example:

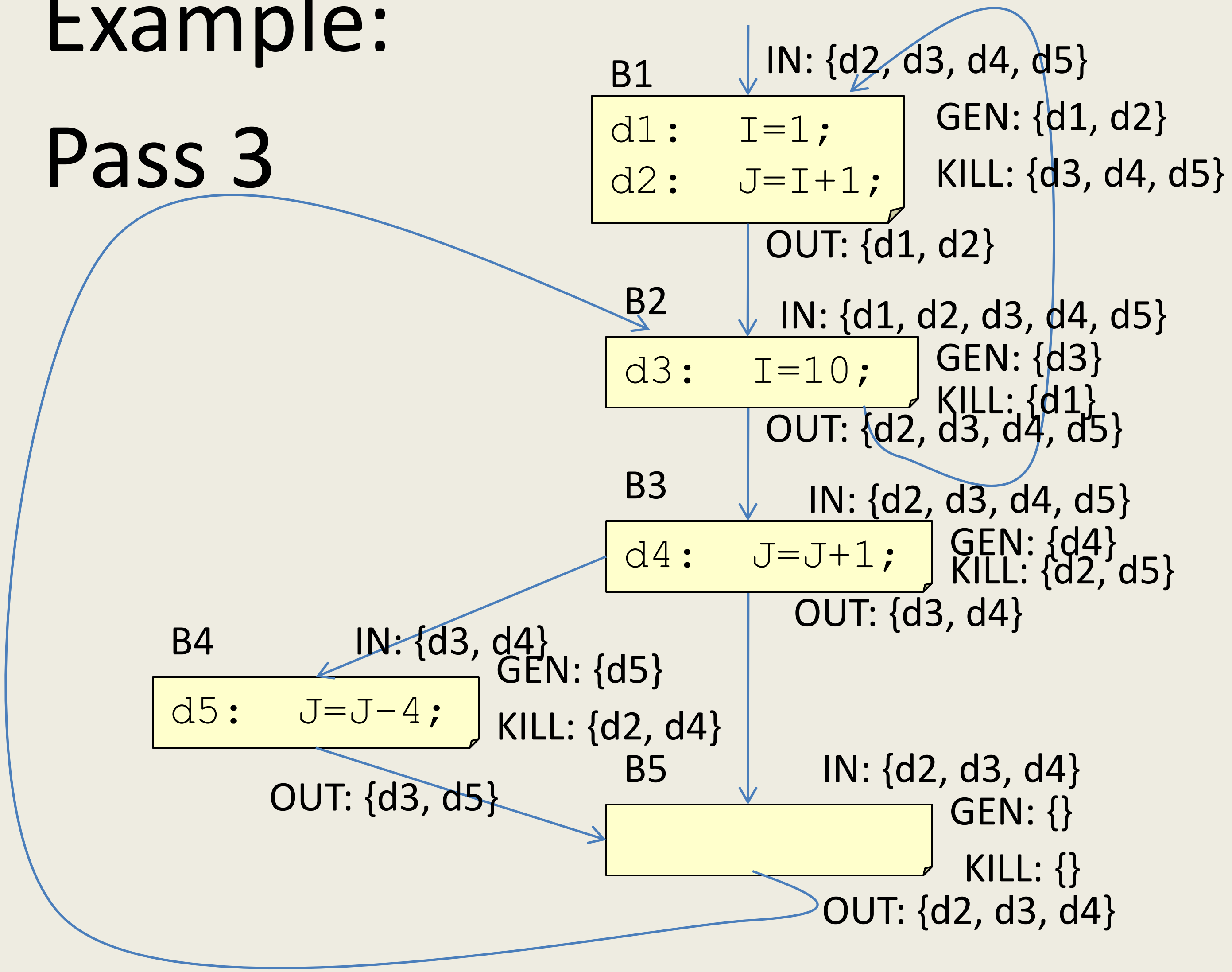
Pass 2



Reaching Definitions

Example:

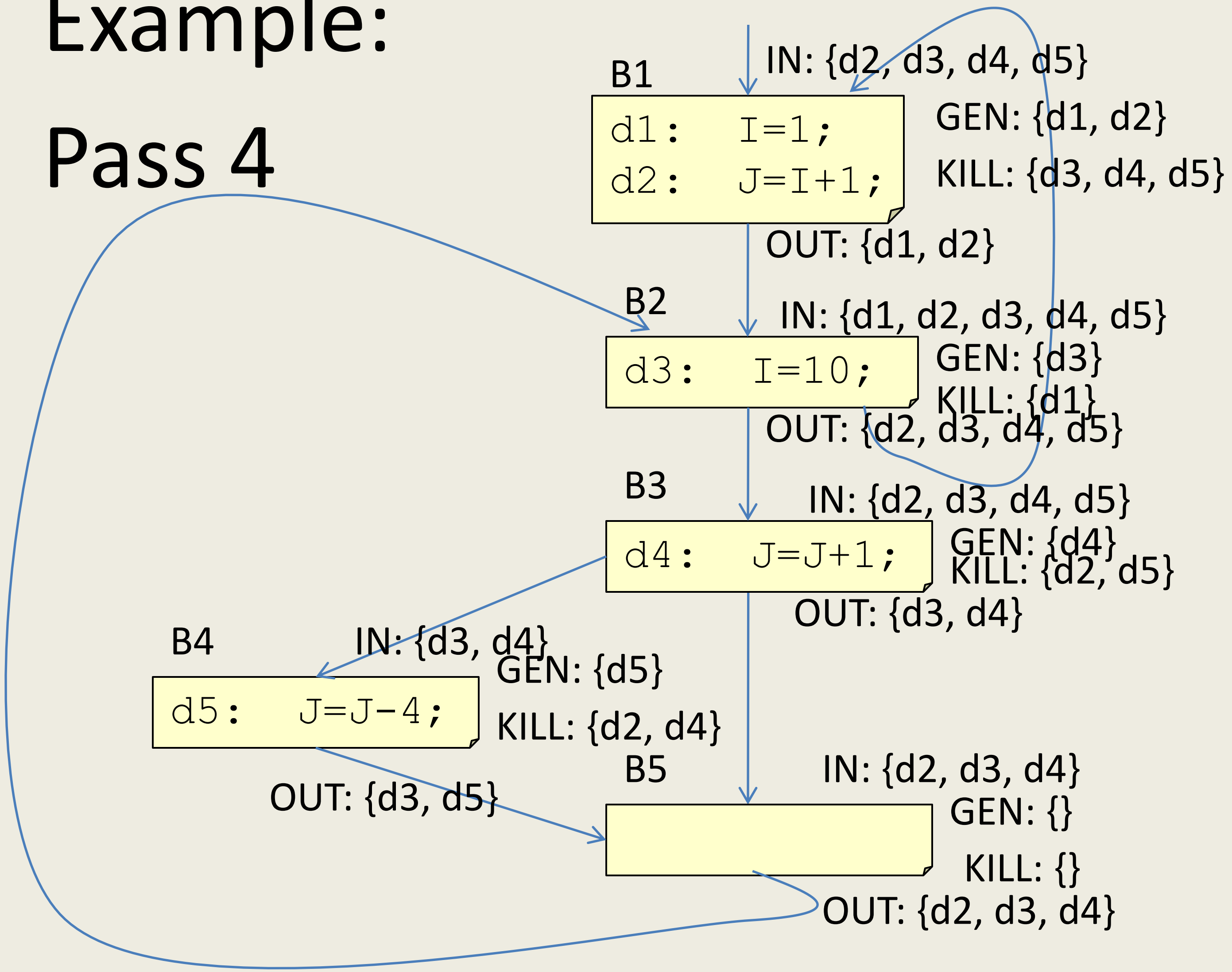
Pass 3



Reaching Definitions

Example:

Pass 4

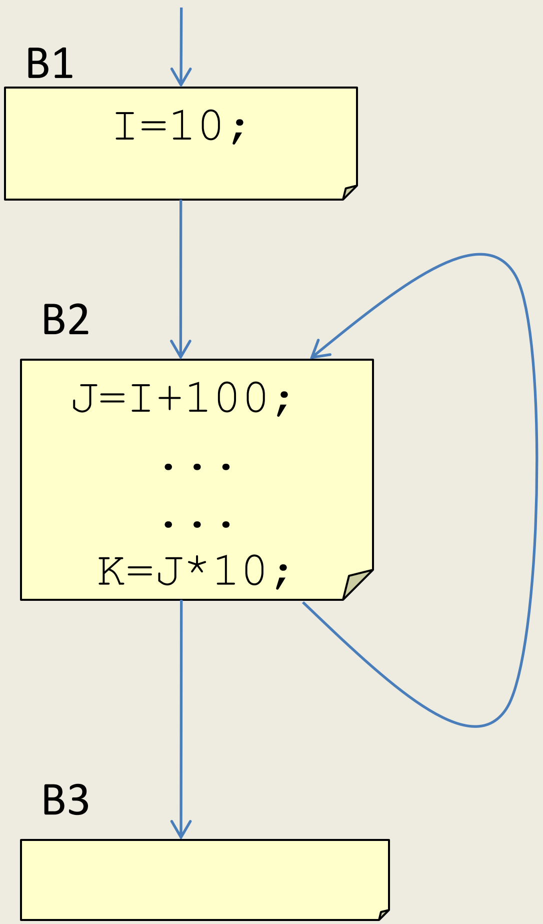


Applications of Reaching Definitions

- Using this information, Constant Folding can be done.
- If there is only one definition of B reaches to point p and the definition is B=c for a constant c then B may be replaced by c.

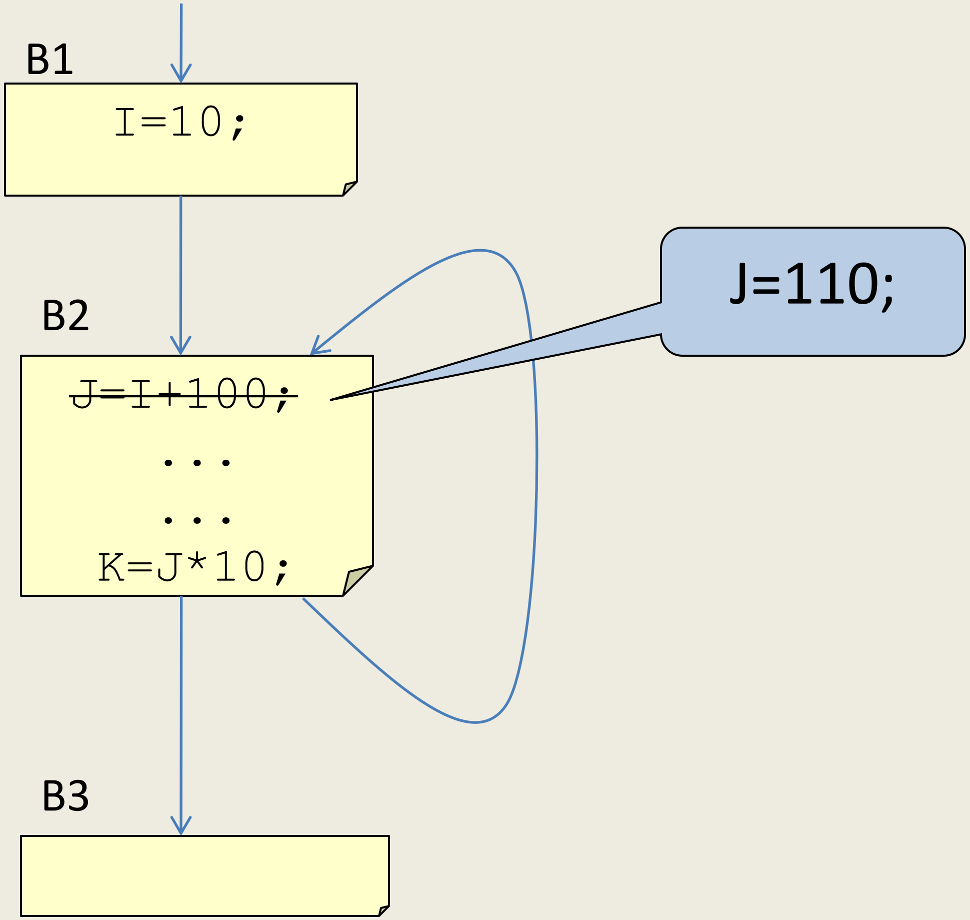
Applications of Reaching Definitions

Example:



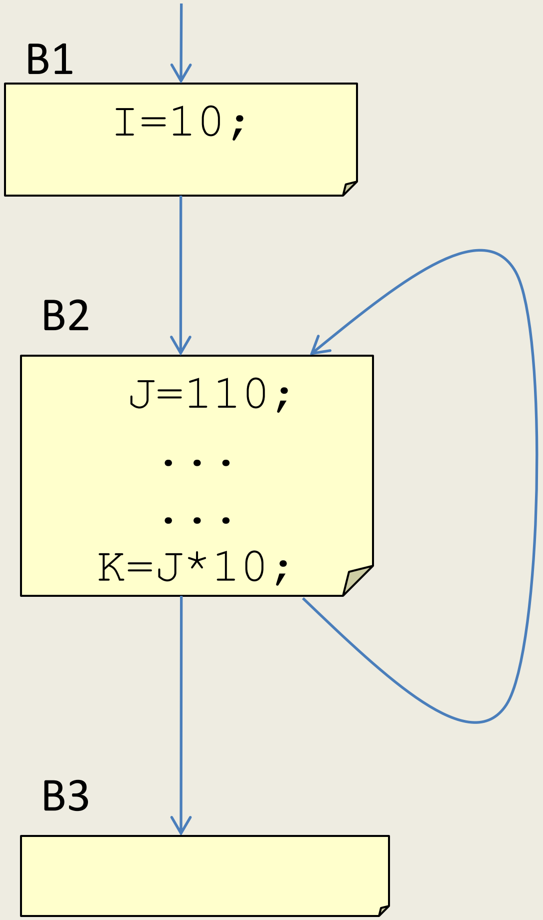
Applications of Reaching Definitions

Example:



Applications of Reaching Definitions

Example:



Applications of Reaching Definitions

Example: