SOFTWARE TIMELINESS
CS3213 FSE

Prof. Abhik Roychoudhury

National University of Singapore

WHAT WE DID EARLIER

- System Requirements: Use-cases, Scenarios, Sequence Diagrams
- System structure: Class diagrams
- Discussion on semantics
- System behavior: State diagrams
- Discussion of the thinking behind your course project
- Static analysis and vulnerability detection: Secure SE
- Software Debugging
- White-box Testing: test estimation and generation
- Taint propagation, malicious inputs: secure SE

- Today
  - **Software timeliness: performance gaps, and real-time software**

WCET

- Worst Case Execution Time (WCET) of a program for a given hardware platform.
  - Sequential Terminating Programs.
  - Gets input, computes, produces output.
- Many inputs are possible.
  - Leads to different execution times.
- WCET : An upper bound on the execution time for all possible inputs.

WHY NEED ANALYSIS?

- To find WCET of a program, execute it for all possible inputs.
  - WCET by measurement.
  - Exponentially many possible inputs in terms of input size.
    - Insertion sort program
  - Similar problems will be encountered for WCET Analysis via platform simulation.
- Need access to platforms/simulators also!
  - Go for static analysis.

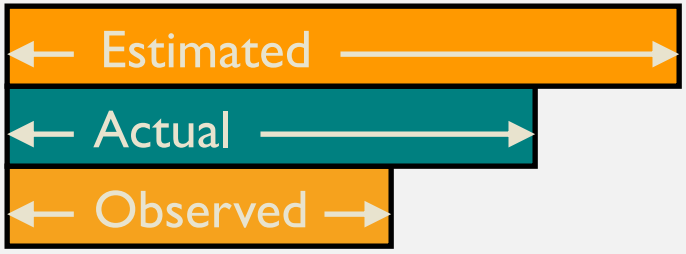
WCET BY MEASUREMENT?

- What about one-path programs such as matrix multiplication?
  - Execution path is independent of input data.
  - Still execution time can be variable.
    - Latency of floating point operation (e.g., multiplication) depends on the input data.
  - Not possible to try it on all possible platforms and then choose one.
    - Often trying to decide the platform as well.

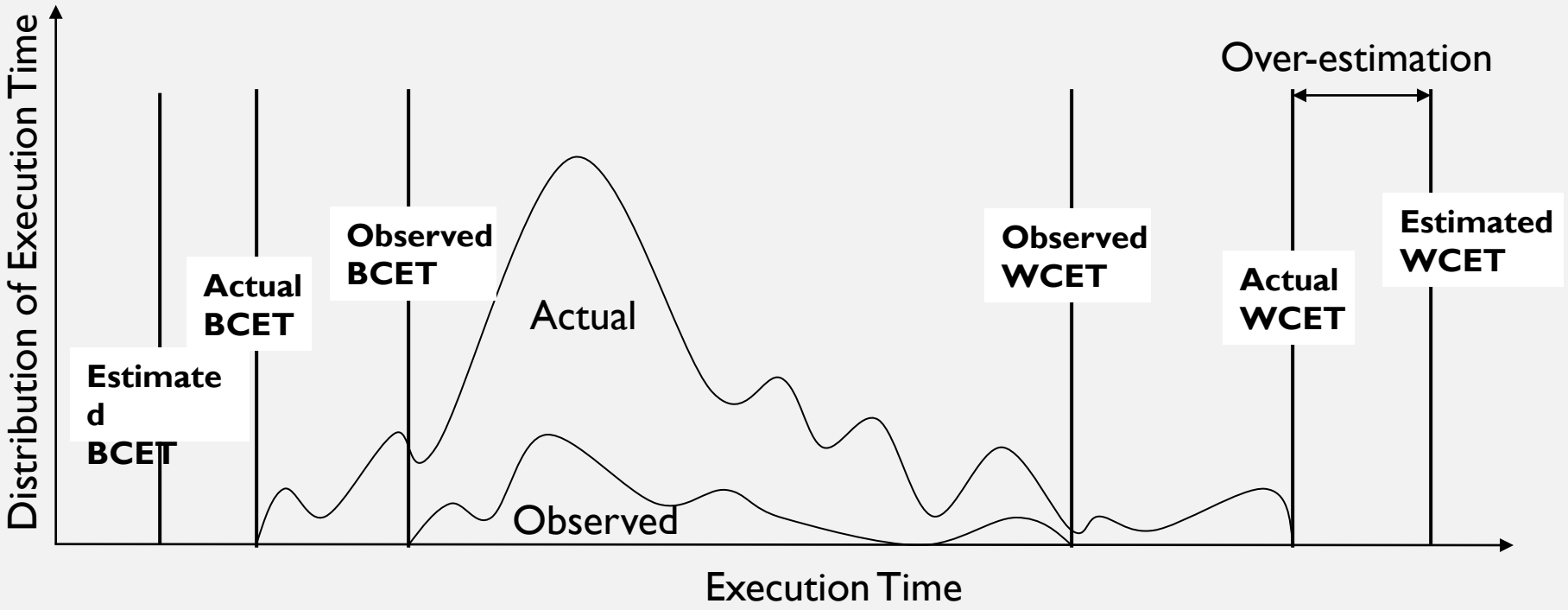
WCET ANALYSIS

- Employ static analysis to compute an upper bound on actual WCET (**Estimated WCET**)
- Run program on selected inputs get a lower bound on actual WCET (**Observed WCET**)

Estimated WCET ≥ Actual WCET ≥ Observed WCET

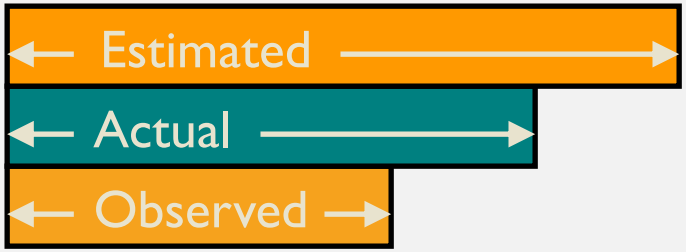


BCET AND WCET



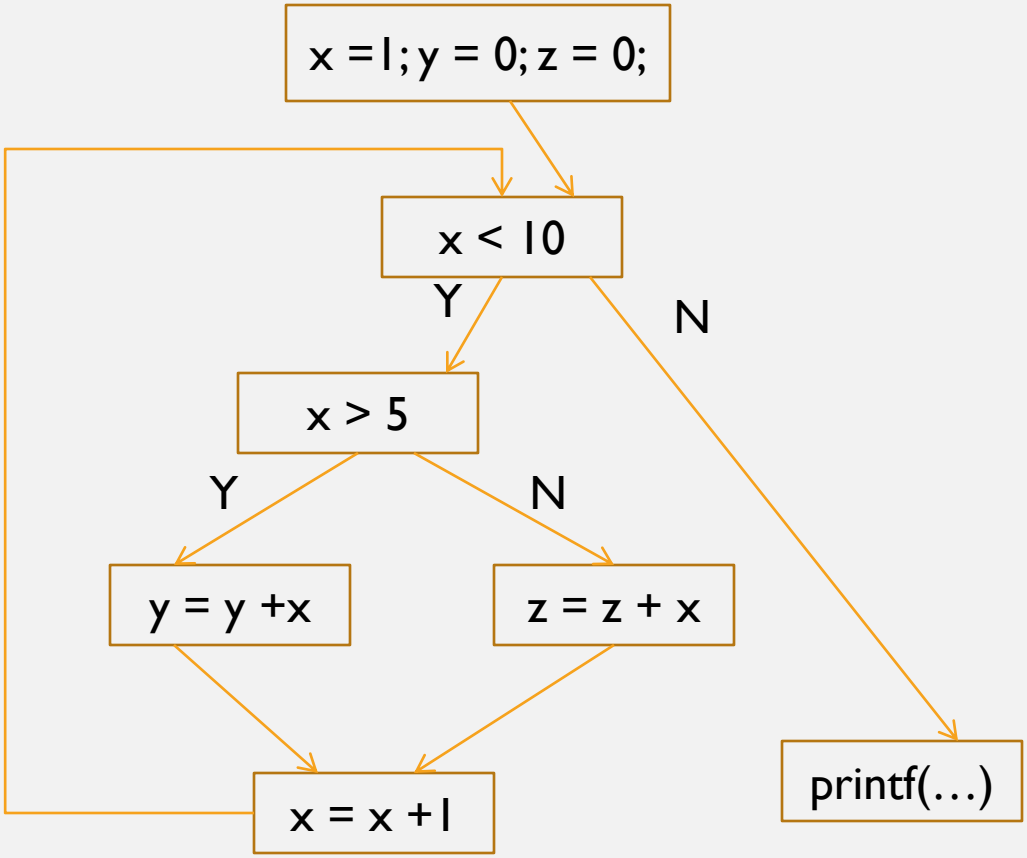
WCET ANALYSIS

- Program path analysis
  - All paths in **control flow graph** are not feasible.
- Micro-architectural modeling
  - Dynamically variable instruction execution time.
    - Cache, Pipeline, Branch Prediction
    - Not covered in our Software Engineering perspective.



CONTROL FLOW GRAPH

```
x = 1; y = 0; z = 0;
while (x < 10){
    if (x > 5)
            y = y + x;
    else   z = z + x;
    x = x + 1;
}
printf(….);
```



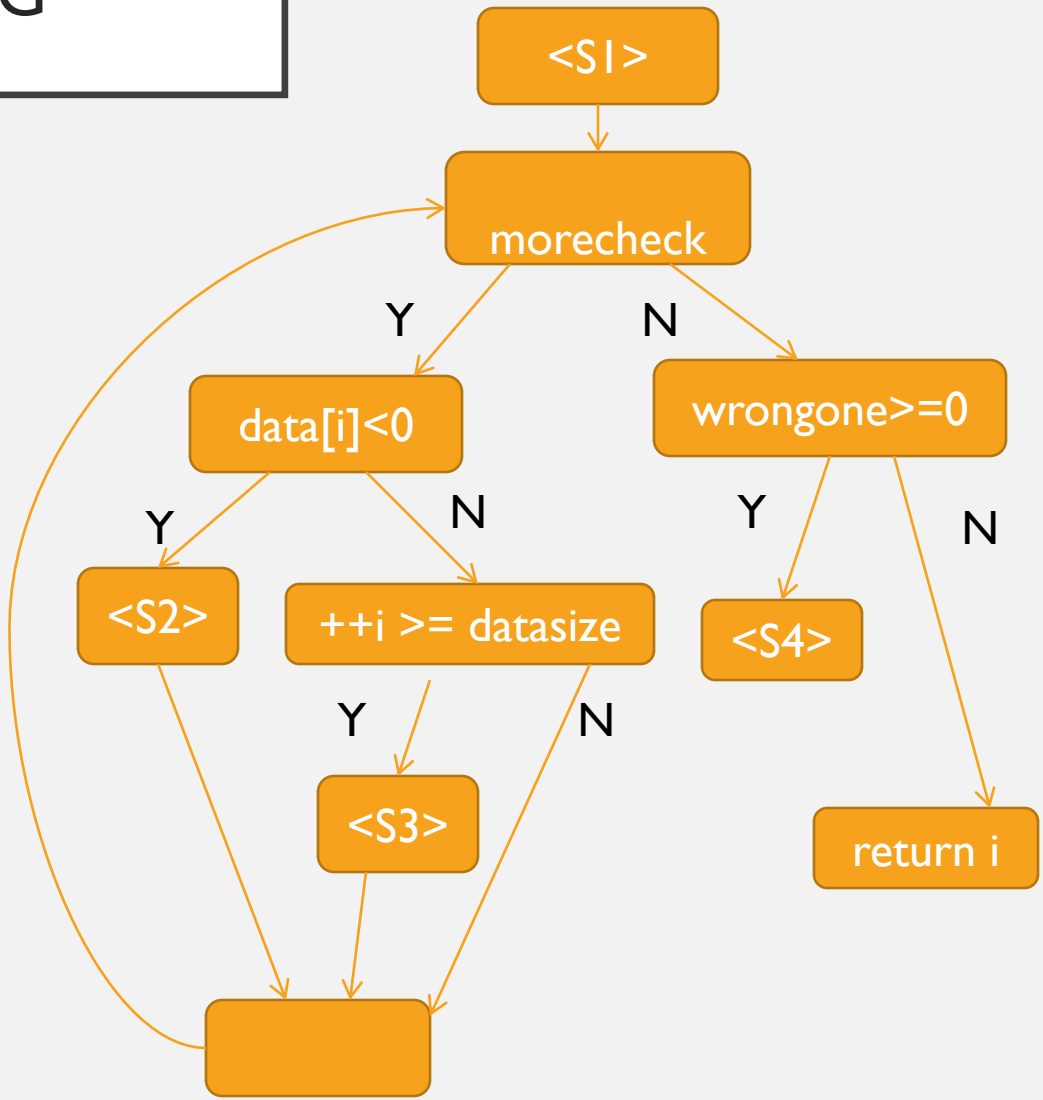
Nodes of the graph, basic blocks, are maximal code fragments executed without control transfer. The edges denote control transfer.

EXERCISE: CFG

```
procedure Check_data()
{       int  i = 0, morecheck = 1, wrongone = -1, datasize = 10;
  L:    while (morecheck)
  LB:   {
              if  (data[i] < 0)
  A:              {  wrongone = i;   morecheck = 0; }
              else
  B:                  if  (++i >= datasize) morecheck = 0;
        }
        if (wrongone >= 0)
  C:            { handle_exception(wrongone); return 0; }
  C':   else  return i;
}
```

EXERCISE: CFG

```
procedure Check_data()
{       …. <S1>
  L:    while (morecheck)
  LB:   {
              if  (data[i] < 0)
  A:              { …. <S2> }
              else
  B:           if  (++i >= datasize)
                          … <S3>;
        }
        if (wrongone >= 0)
   C:             { … <S4> }
   C':   else  return i;
}
```



EXERCISE ON CFG

How to construct an inter-procedural CFG for a program with many procedures?

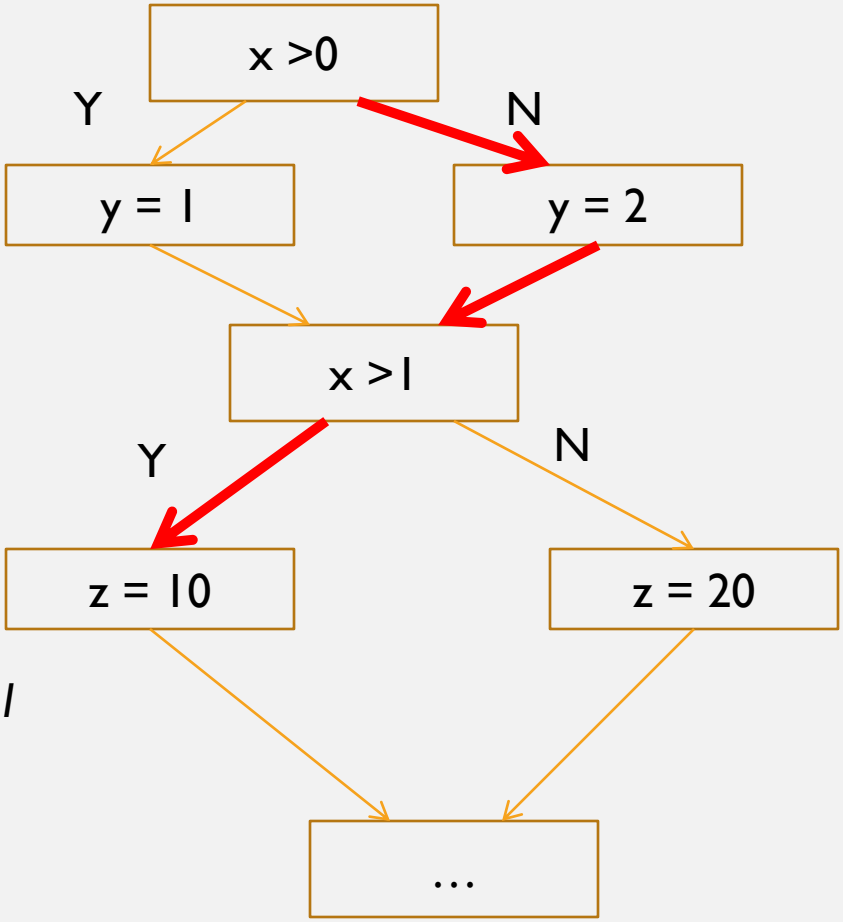
```
main(){
…
   f1();
   f2();
…
}
```

```
f1(){
   …
  f2();
   …
 }
```

```
f2(){
    …
  f2();
   …
 }
```

WHY ALL PATHS MAY NOT BE FEASIBLE?

- if (x > 0{
- y = 1;
- else
- y = 2;
- }
- if (x > 1){
- z = 10;
- else
- z = 20;
- }
- …



$x \leq 0 \ \wedge x > 1$

RESTRICTIONS OF ANALYSIS – (1)

- Static analysis need not be on source program.
  - We can perform static analysis on assembly code of a given program.
  - The analysis is only for time taken, and not for the memory locations / values accessed.
  - No restriction on program data structures used for WCET analysis.
  - What about control flow ?

RESTRICTIONS OF ANALYSIS – (2)

- Restrictions on control flow
  - 1. No unbounded loops
    - Common sense.
      - Otherwise how to guarantee time?
  - 2. No unbounded recursion
    - Similar issue.
  - 3. No dynamic function calls
    - Need to statically know the functions called, and the possible call sites of these functions.

ORGANIZATION OF WCET ANALYSIS

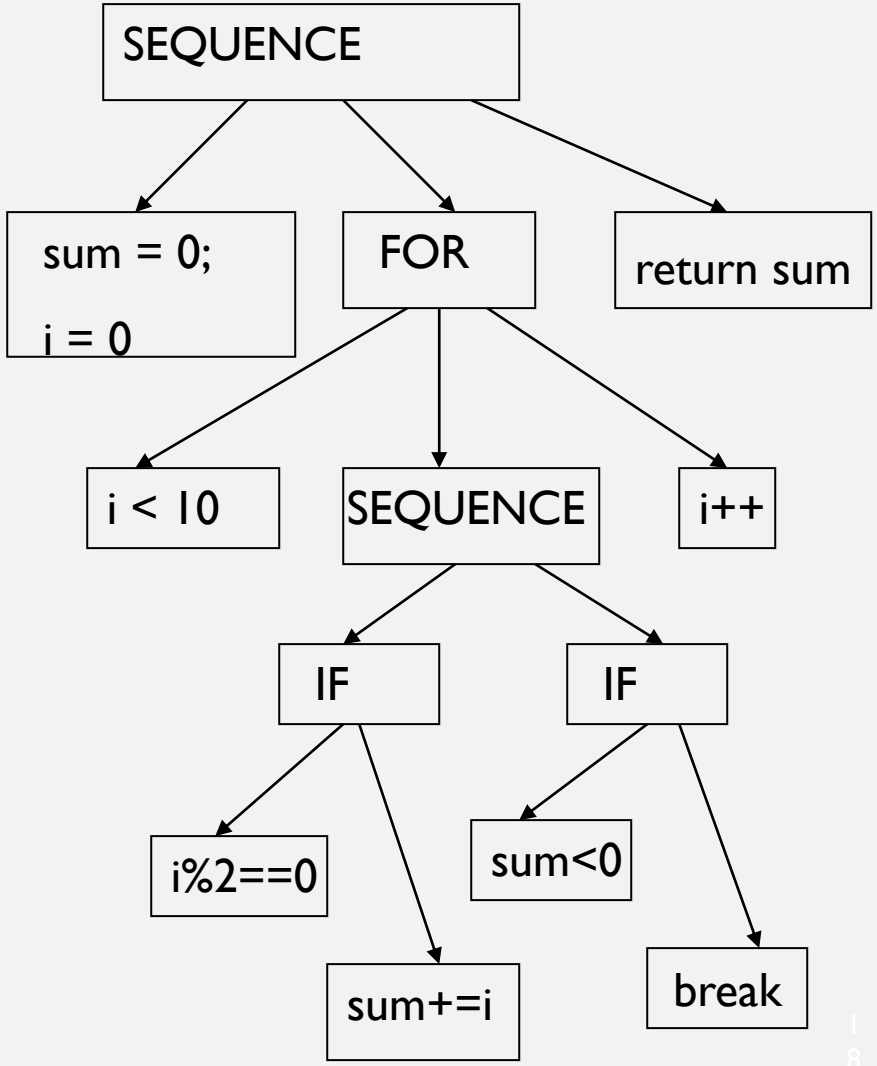
- What is Timing Analysis ?
- An Early solution -- Timing Schema.
- Modeling Program Flows.
  - Primarily Control flow.
- Modeling timing effects of Micro-architecture.
  - Cache, pipeline, Not covered in CS3213 FSE

TIMING SCHEMA

- One of the first works on WCET analysis.
- Basically, perform control flow analysis to find the "longest" program path.
- The notion of "longest" is weighted
  - Take into account the cost of executing individual program elements.
  - Timing schema is a simple way of composing these costs.
- Does not work on Control Flow Graphs
  - Works on Abstract Syntax Tree

EXAMPLE

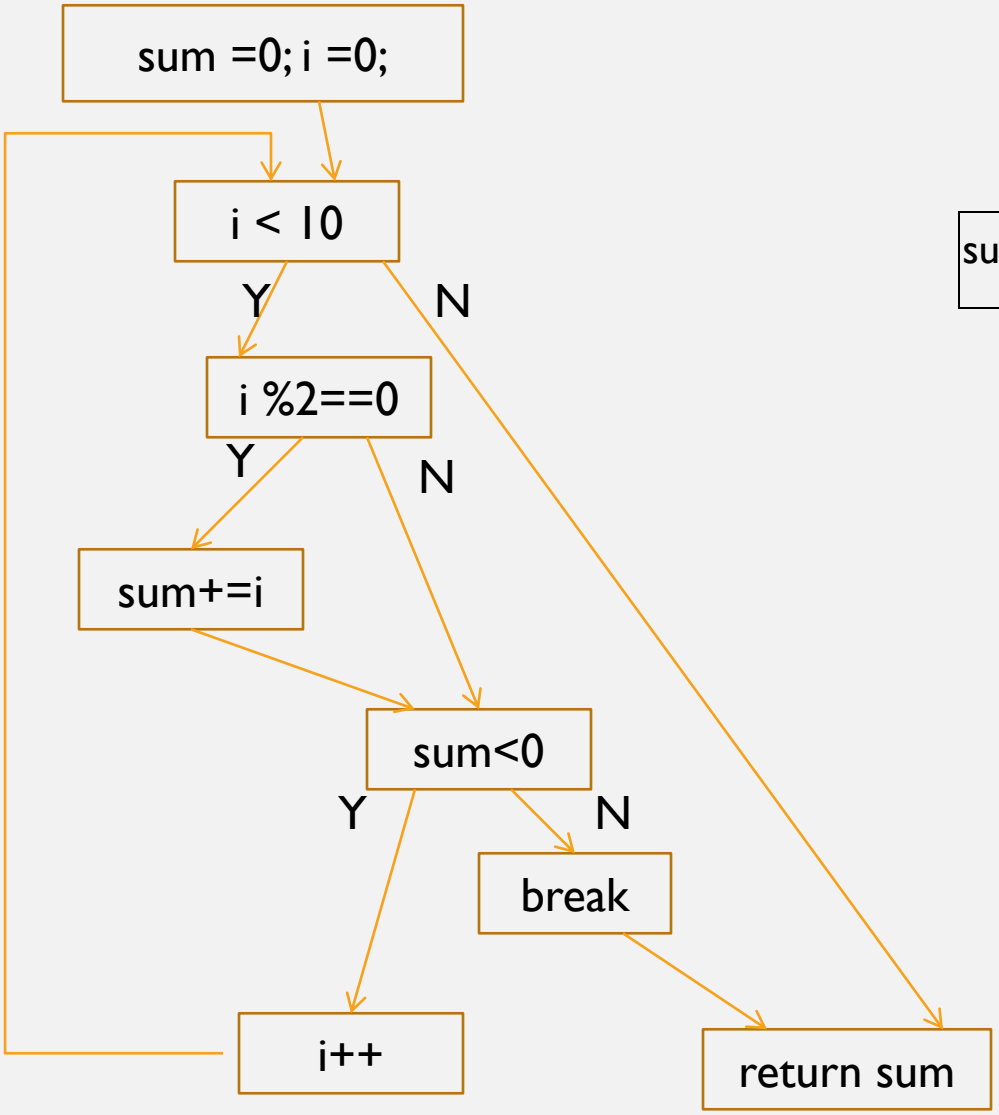
```
sum = 0;
for (i=0; i< 10; i++){
    if (i % 2 == 0)
        sum += i;
    if (sum < 0)
        break;
}
return sum;
```

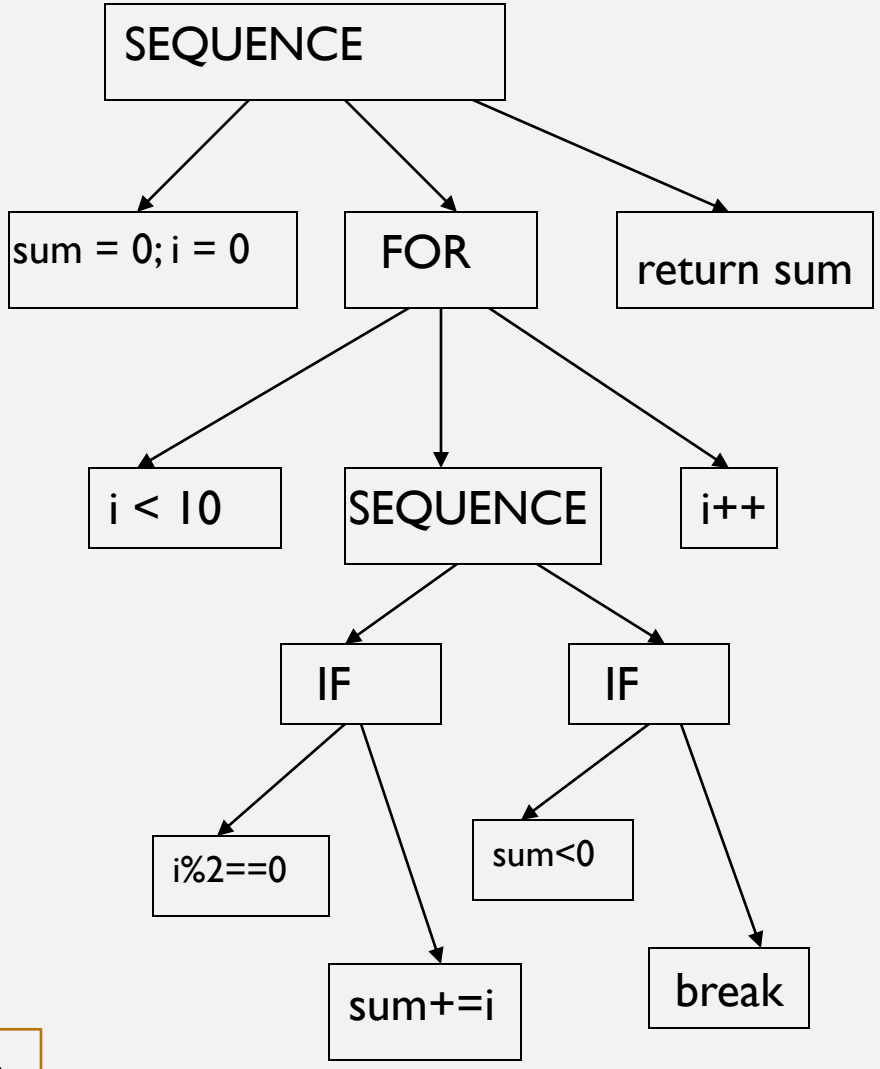


AST AND CFG

- Hierarchy
  - AST shows the different scopes at different levels
  - CFG has no hierarchy.
- Loops
  - AST is tree, free from cycles
  - Any loop in the program is a cycle in the CFG.

REPRESENTATIONS: AST AND CFG





TIMING SCHEMA – (WORKS ON AST)

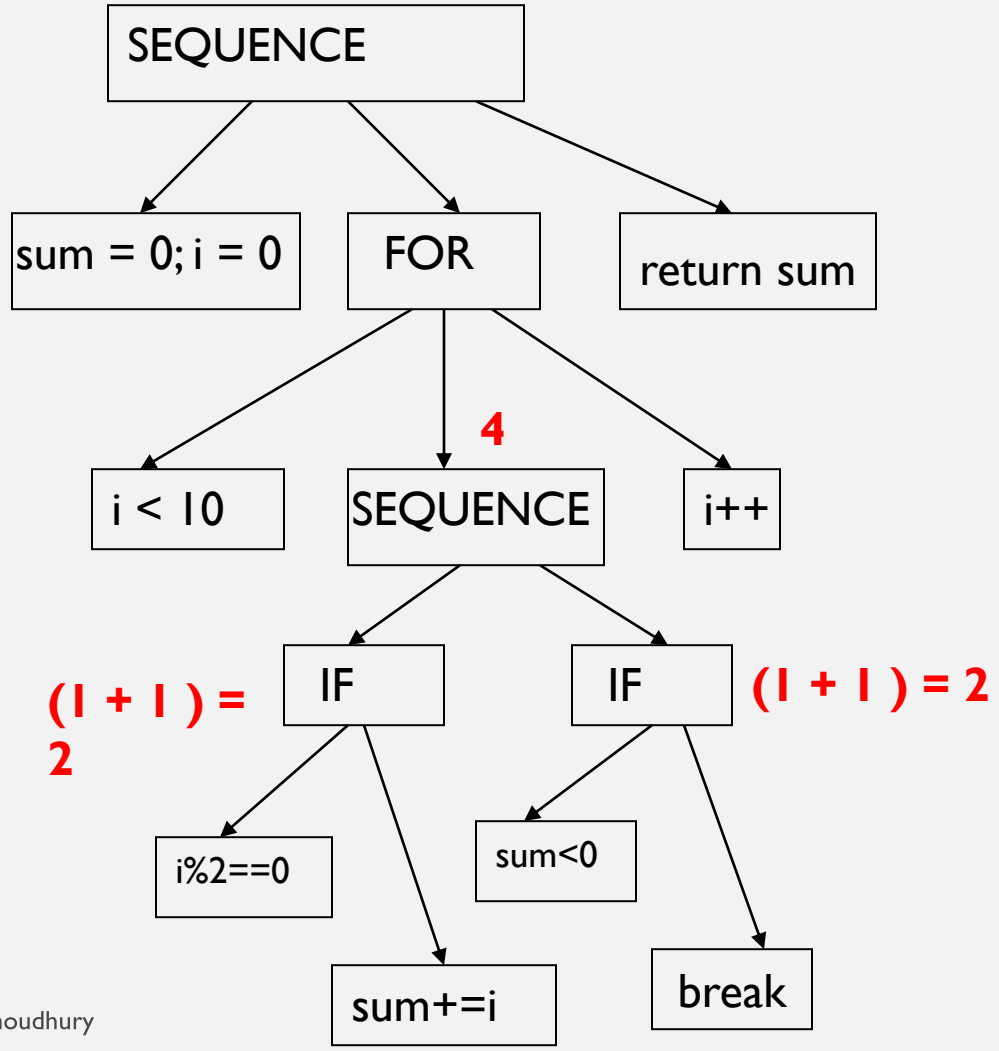
- Time(S1;S2) = Time(S1) + Time(S2)
- Time(if B {S1} else {S2})
  - = Time (B) + max(Time(S1), Time(S2))
- Time(while B {S1})
  - = (n+1) * Time(B) + n * Time(S1)
  - n is the loop bound.

- Time(for(Init; B; Incr.){ S })
  - = Time(Init) + (n+1)*Time(B) + n*Time(S) + n*Time(Incr.)
- Time(if (B) { S}) = Time(B) + Time(S)

TIMING SCHEMA

Time(for-loop)
= Time(i= 0) +
11 * Time(i < 10) +
10 * 4 + 10 * Time(i++)
= 1 + 11 + 10*4 + 10*1
= 62 time units

Assumption:
Each assignment/condition takes 1 time unit (not realistic in practice).



PROBLEMS WITH TIMING SCHEMA

- Language Level:
  - Just a control flow analysis.
  - Insensitive to knowledge of infeasible paths.
- Compiler level:
  - How to integrate effect of compiler opt?
    - Easy to handle – schema on optimized code.
- Architecture level:
  - Instructions take constant time – Not true.
  - Cache hits, pipelining and other performance enhancing features.

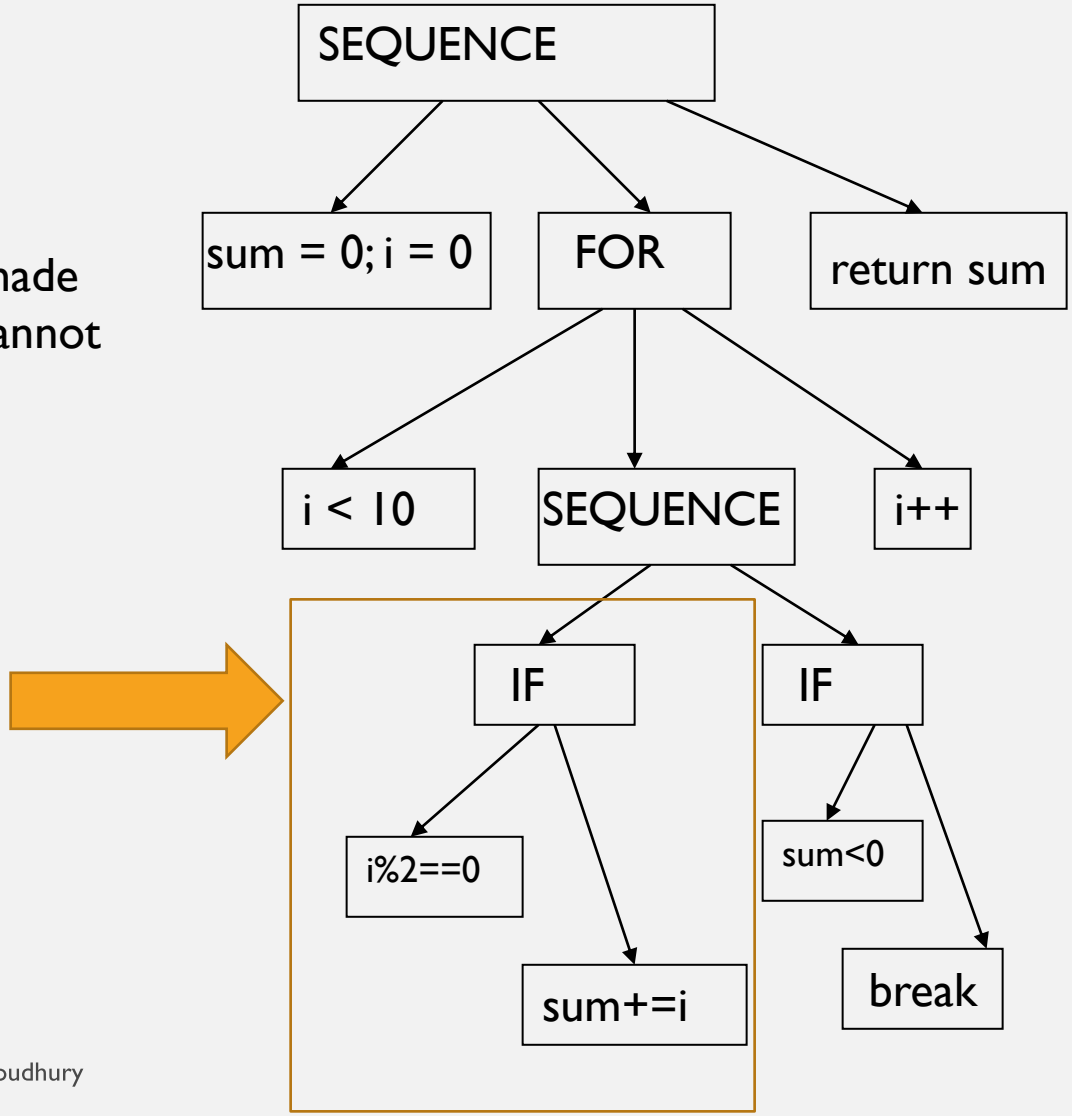
INFEASIBLE PATHS NOT CONSIDERED

It is impossible to execute i%2==0 to be true in two consecutive iterations.

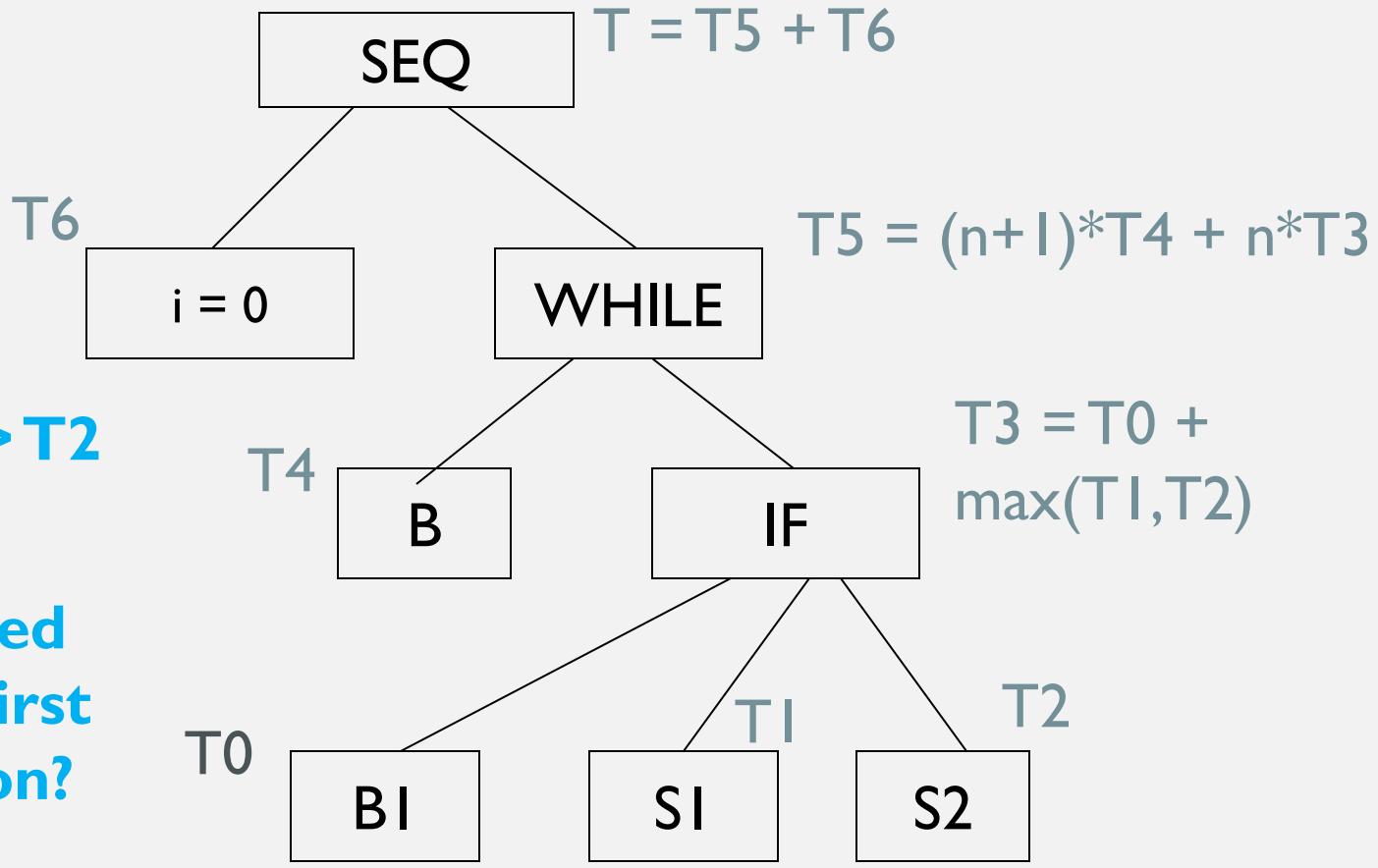
Even if this information is made available – timing schema cannot use it.

The time assigned to this block of code is Time(i%2==0) + Time(sum+=i)

This is the max. time taken by this statement in *any* iteration.



INFEASIBLE PATHS



What if T1 > T2 and

S1 is executed only in the first loop iteration?

INFEASIBLE PATHS

- Infeasible sequence of statements in general

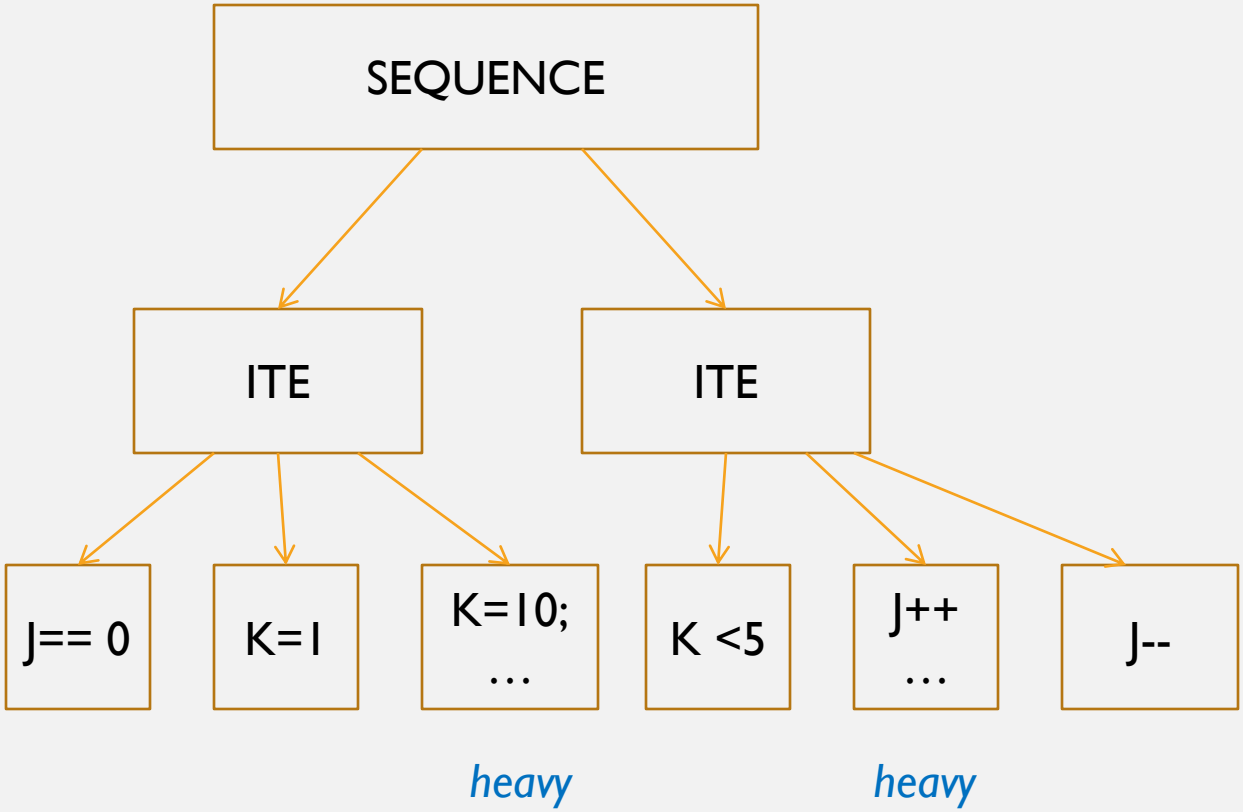
```
if (J== 0) {
        K = 1
} else {
        K = 10
}
if (K < 5){
        J++;
} else {
      J--;
}
```

Cannot be executed together

Such infeasible paths should not be a witness to our WCET estimate.

INFEASIBLE PATH HANDLING IN TIMING SCHEMA

```
if  (J== 0) {
        K = 1
} else {
    K = 10; …
}
if  (K < 5){
        J++;…
} else {
      J--;
}
```



How will timing schema work on this example?

WORKING OF TIMING SCHEMA

Time(first-if-statement)
= 1 + max(1,5) = 6

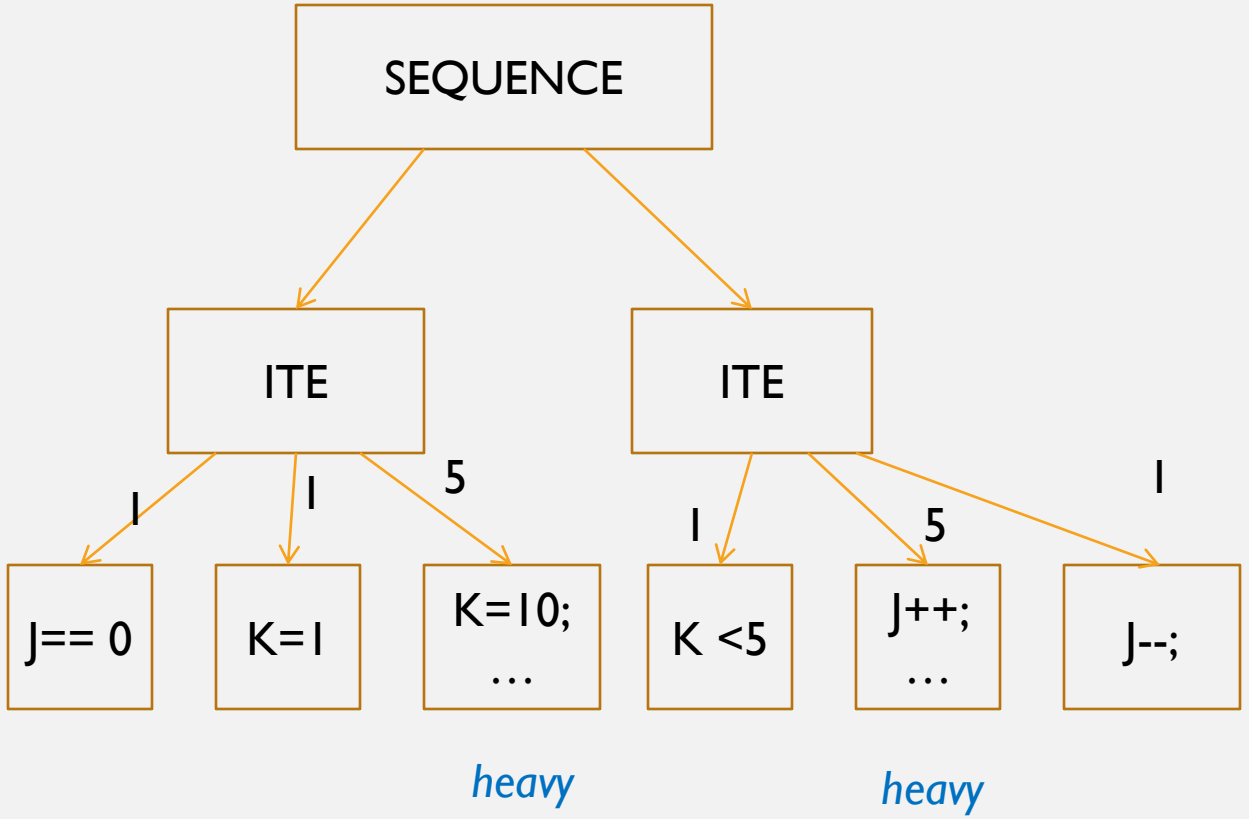
Time(second-if-statement)
= 1 + max(5,1) = 6

Estimated worst case time = 6 + 6 = 12

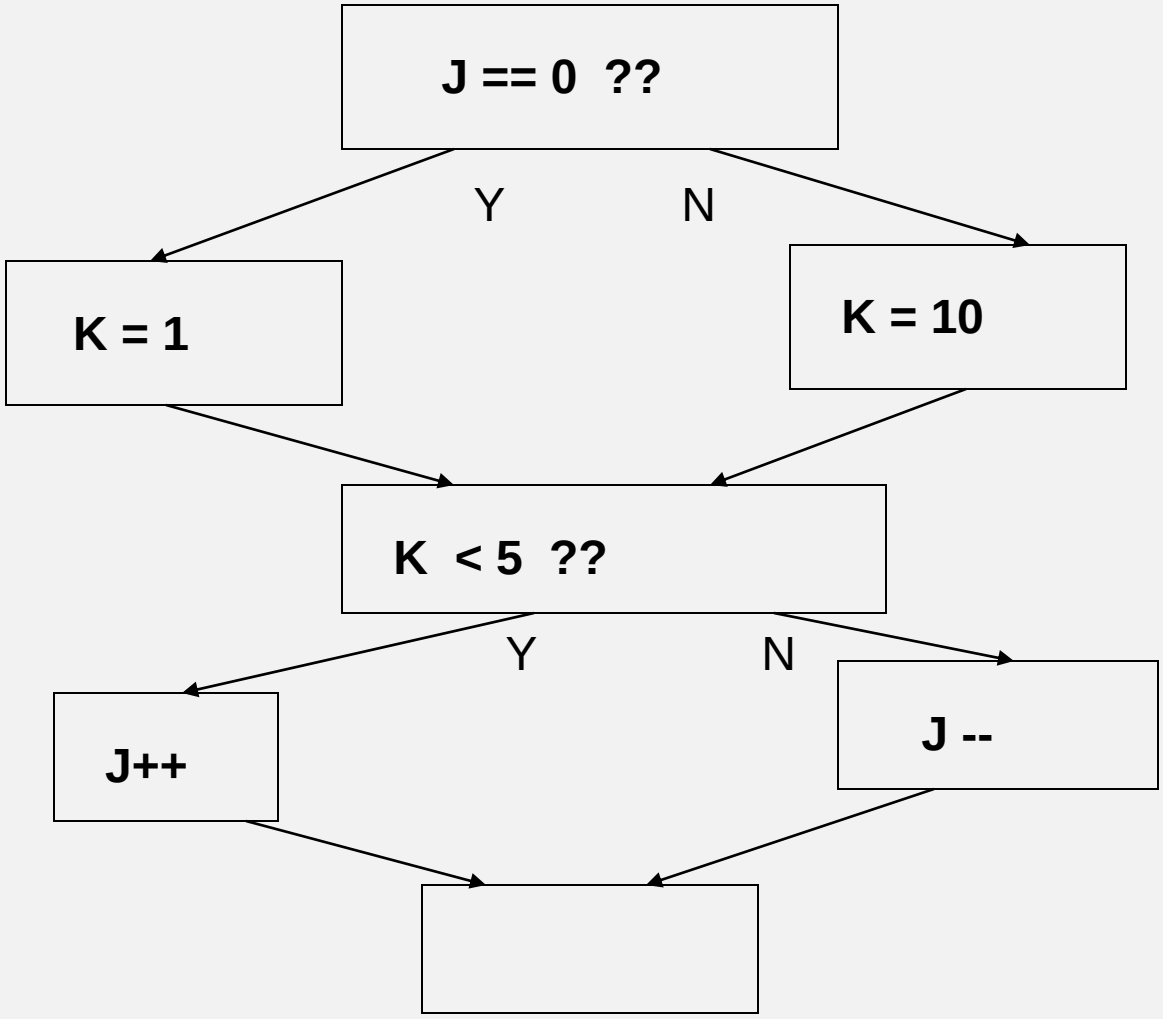
Actual worst-case time = 2 +6 = 8

Why?

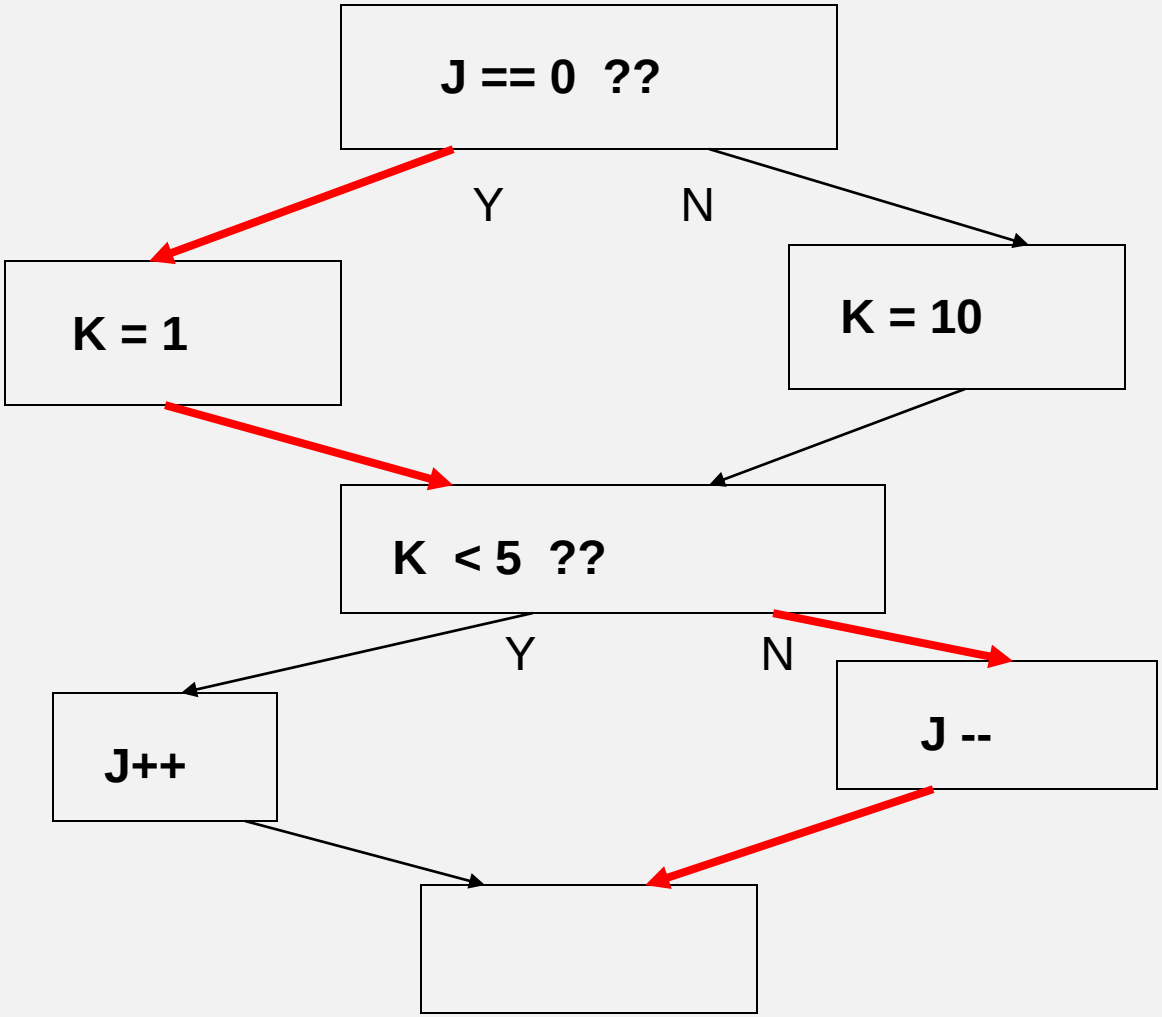
Where is the overestimate from?



CONTROL FLOW GRAPH (CFG)



INFEASIBLE PATH IN CFG



MODELING OF CONTROL FLOW

- Path-based
  - Enumerate paths and find longest path
    - Expensive !
    - Need to remove longest path if it is infeasible.
- Tree-based
  - Bottom-up pass of Syntax Tree
    - Timing Schema
  - Difficult to integrate infeasible path info
- Integer Linear Programming
  - Can take into account certain infeasible path information if available.
  - Efficient solvers available e.g. CPLEX
    - Forms the back-end of most state-of-the-art timing analyzers.

INTEGER LINEAR PROGRAMMING

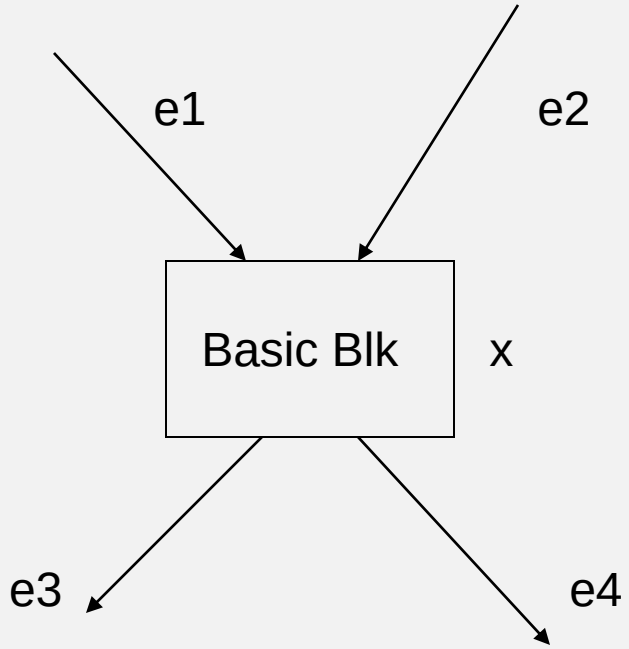
- ILP: Integer Linear Programming
  - Variables and linear constraints on them.
  - Cost function (linear) to optimize.

```
f = 3x + 5y + z
0 <= x, y, z <= 100
x + y + z = 200
x + 2y <= 160
```

Optimal: **f = 520; x = 40; y = 60; z = 100**

Non-Optimal: **f = 480; x = 80; y = 30; z = 90**

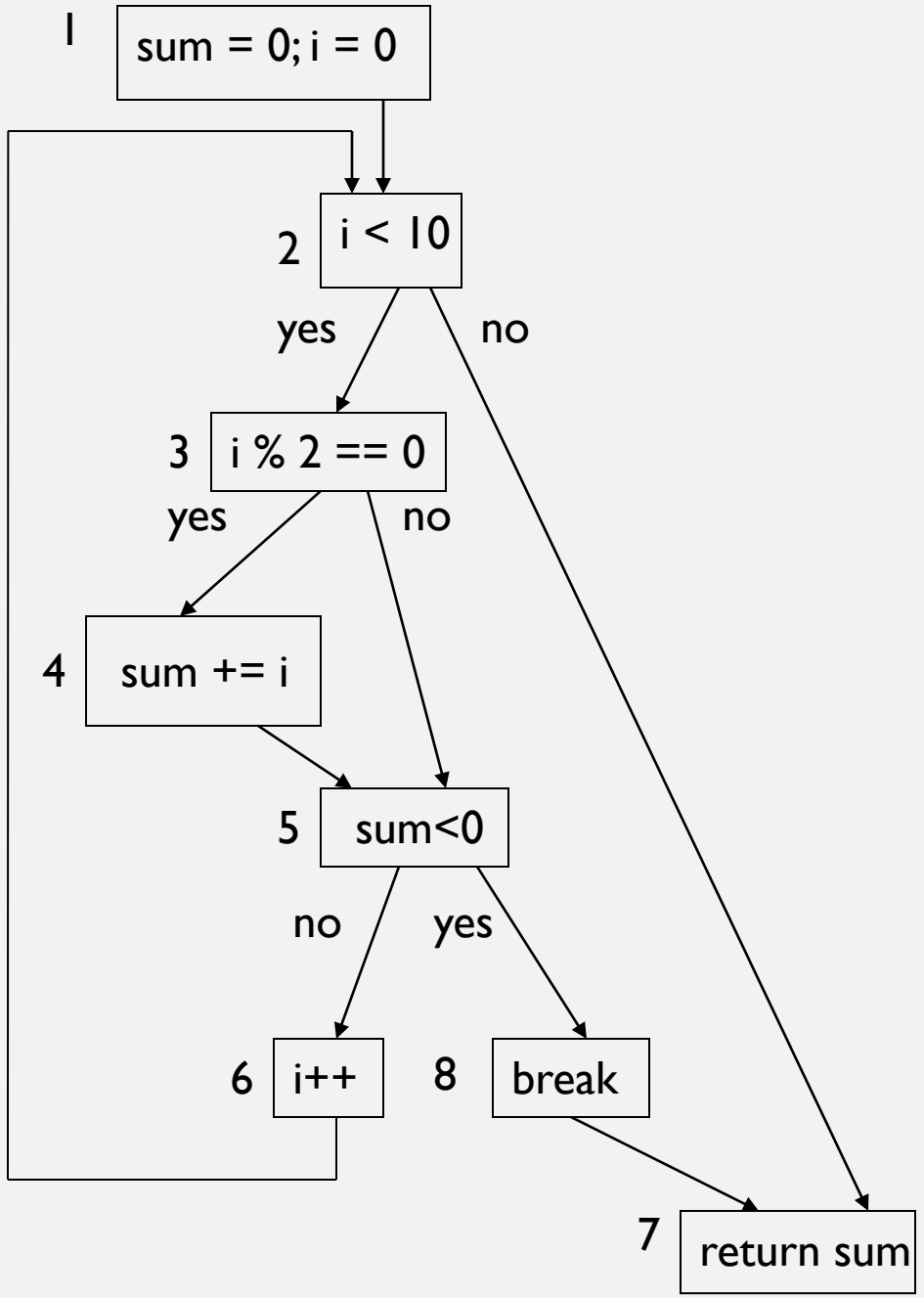
ILP MODELING



$$x = e1 + e2$$
$$= e3 + e4$$

We are dealing with aggregated execution counts of nodes/edges of CFG.

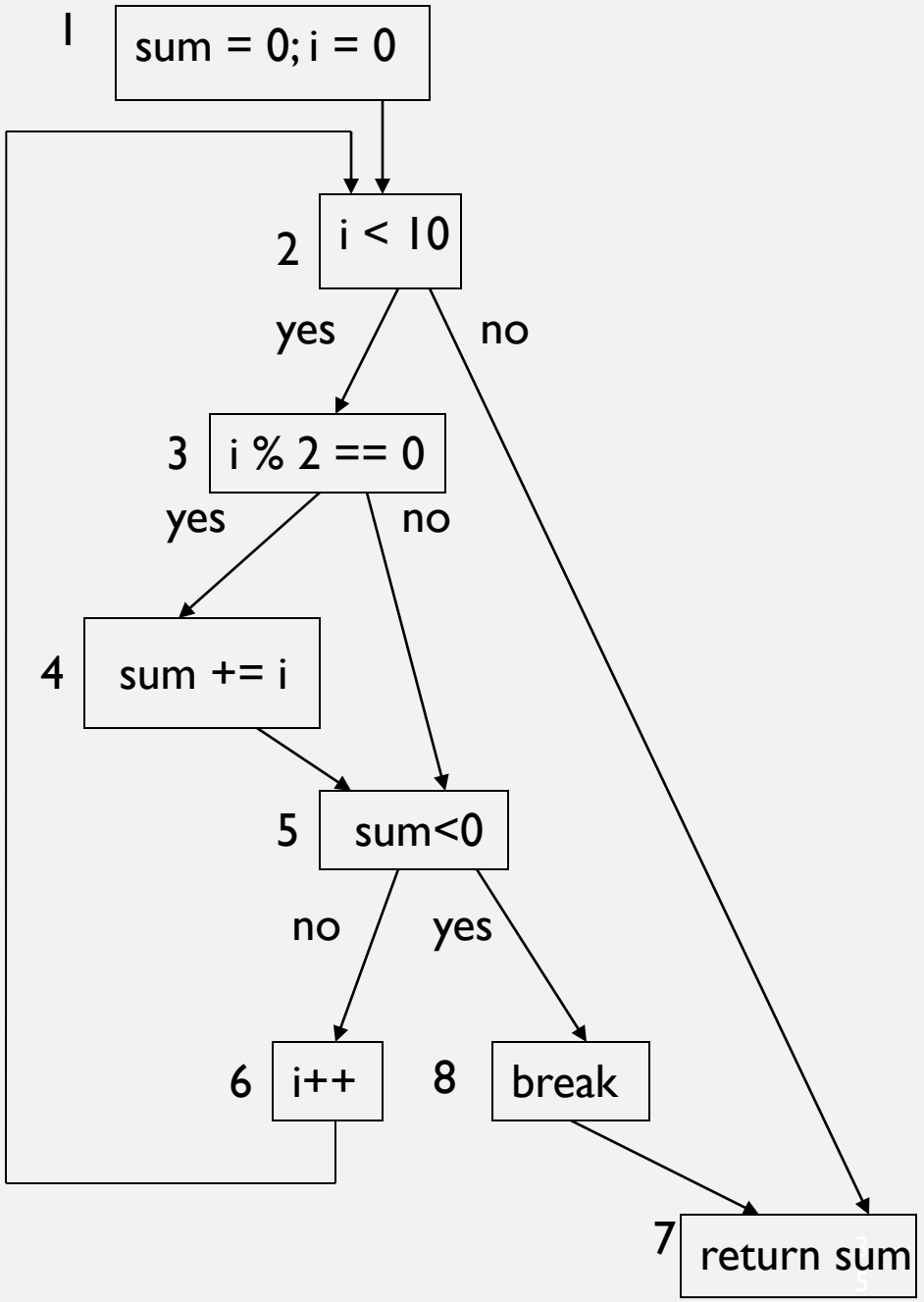
```
sum = 0;
for (i=0; i< 10; i++){
    if (i % 2 == 0)
            sum += i;
    if (sum < 0)
            break;
 }
 return sum;
```



Maximize
Time =
$c_1N_1 + c_2N_2 + c_3N_3 + c_4N_4 +$
$c_5N_5 + c_6N_6 + c_7N_7 + c_8N_8$

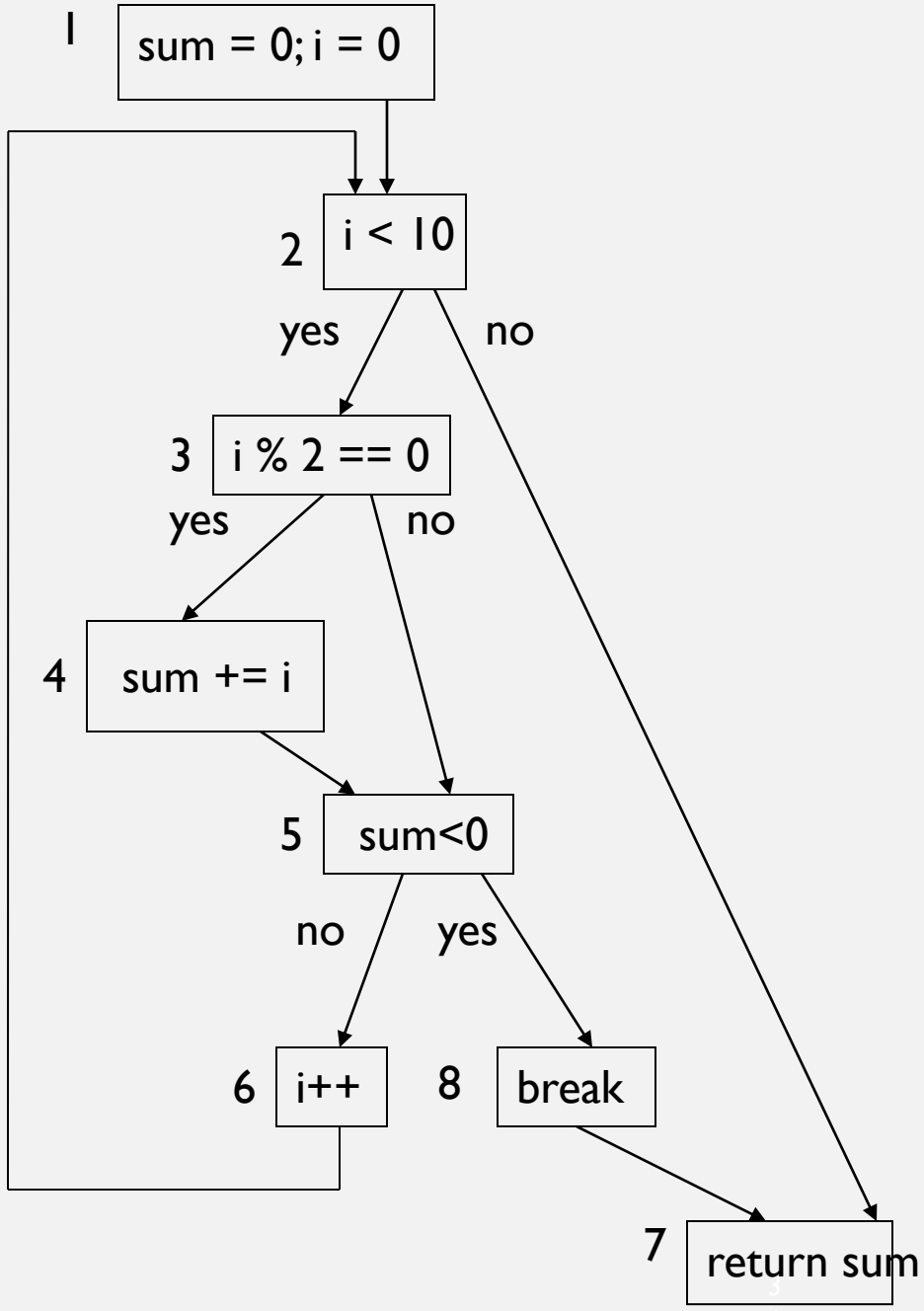
$1 = N_1 = E_{1,2}$
$E_{6,2} + E_{1,2} = N_2 = E_{2,3} + E_{2,7}$
$E_{2,3} = N_3 = E_{3,4} + E_{3,5}$
$E_{3,4} = N_4 = E_{4,5}$
$E_{3,5} + E_{4,5} = N_5 = E_{5,6} + E_{5,8}$
$E_{5,6} = N_6 = E_{6,2}$
$E_{5,8} = N_8 = E_{8,7}$
$E_{8,7} + E_{2,7} = N_7 = 1$

$E_{6,2} \leq 10$
$N_4 \leq 5$



- The break statement is executed at most once.
  - $N_8 \leq 1$

INFEASIBLE PATH



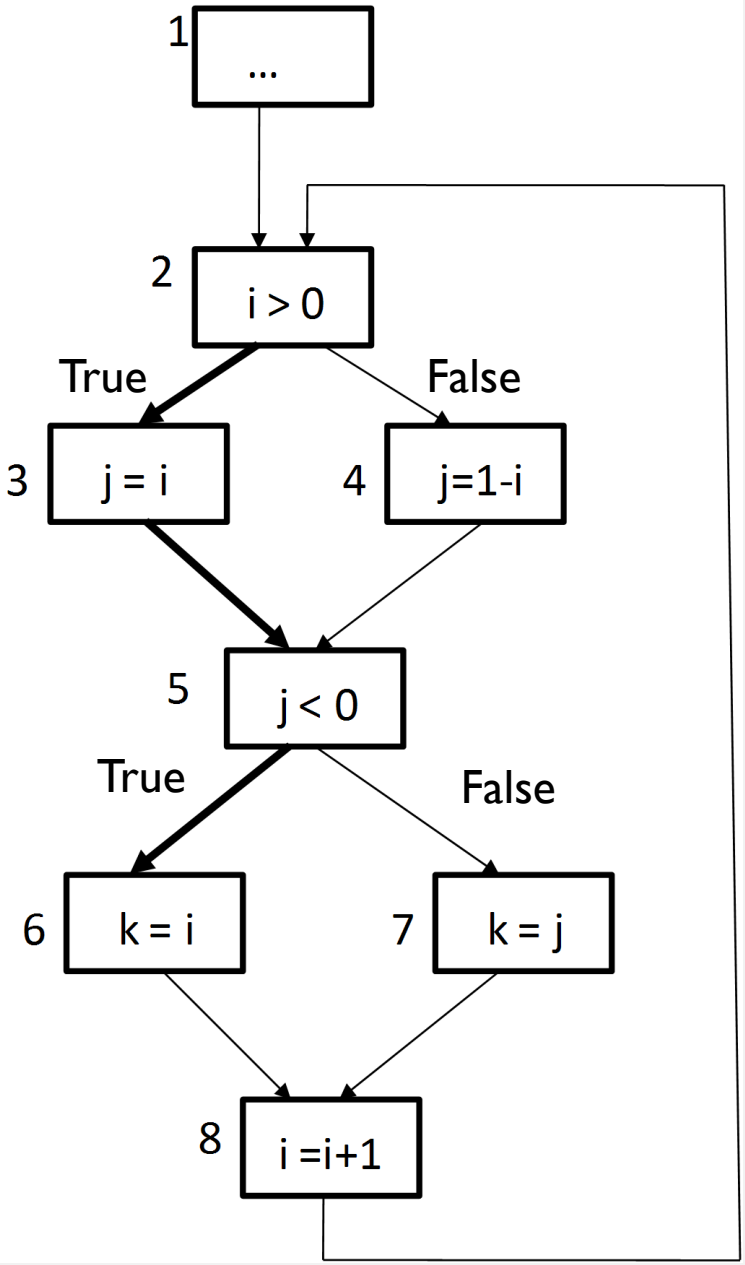
Blocks 3 and 6 are never executed in same loop iteration

$N_3 + N_6 \leq$ loopbound

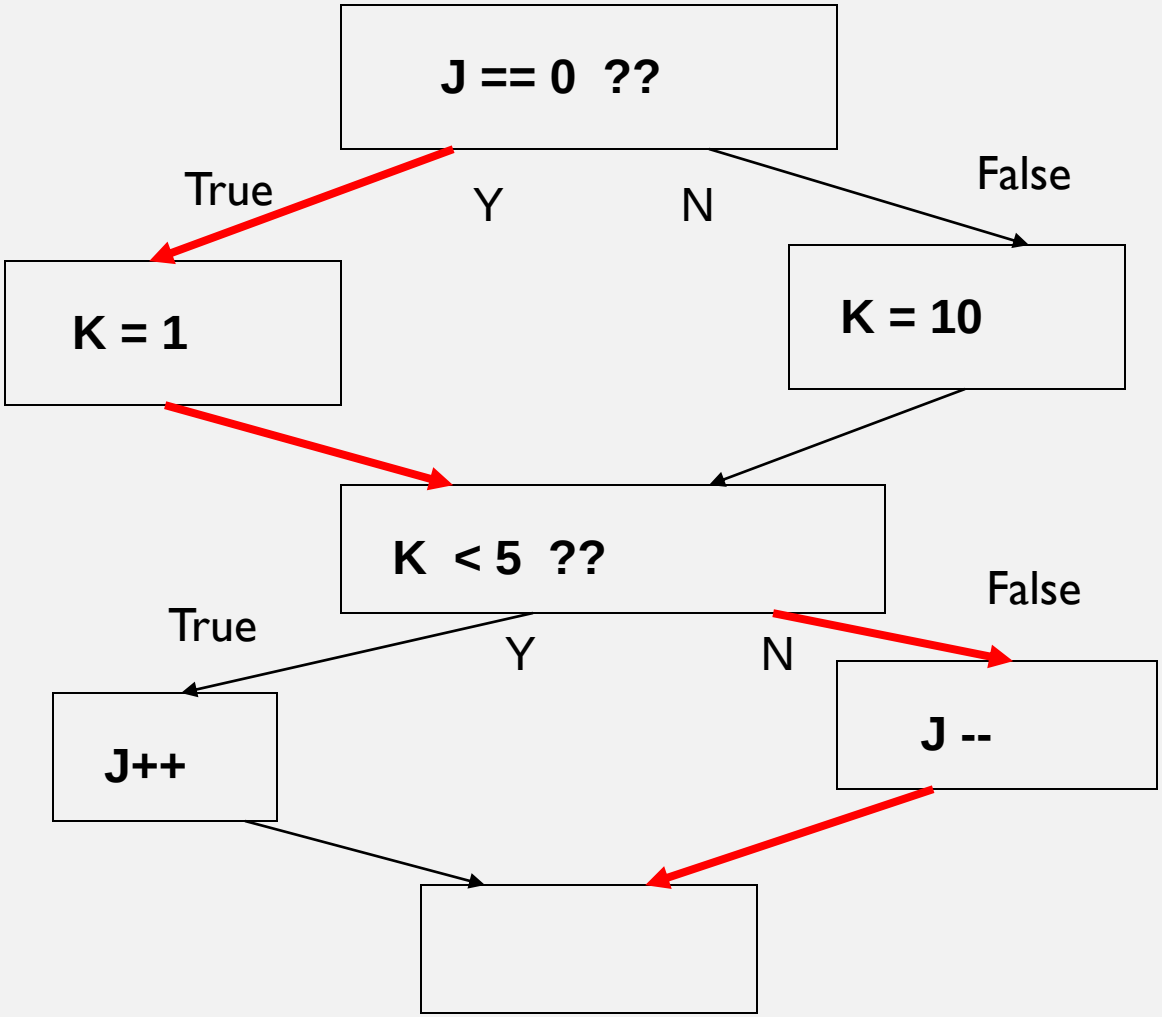
Edges (2,3) and (5,6) are not executed together.

$E_{2,3} = E_{5,7}$

```
while (...){
        if ( i > 0 ){
                j = i;
        } else{
                j = 1-i;
        }
        if  (j < 0){
                k = i;
        } else{
                k = j;
        }
        i = i+1;
}
```



HOW TO EXPRESS THIS INF. PATH CONSTRAINT?



EXERCISE: WHAT ARE THE INFEASIBLE PATHS?

```
procedure Check_data()
{       int  i = 0, morecheck = 1, wrongone = -1, datasize = 10;
        while (morecheck)
        {
             if  (data[i] < 0)
                 {  wrongone = i;   morecheck = 0; }
             else
                  if  (++i >= datasize) morecheck = 0;
          }
          if (wrongone >= 0)
                { handle_exception(wrongone); return 0; }
          else  return i;
 }
```

REVISION OF TIMING ANALYSIS

Abhik Roychoudhury

National University of Singapore

Q1. TIMING SCHEMA

Consider the following program fragment that computes in z the product of x and y. Thus, x and y serve as inputs to the program fragment, and z serves as the output of the program fragment. Both the inputs are positive integers, given as unsigned 8 bit numbers (when represented in binary). Using *Timing Schema* WCET analysis method discussed in class, derive the maximum execution time of the program fragment. Each assignment/return/condition-evaluation takes 1 time unit.

```
z = 0;
while (x !=0){
   if (x %2 != 0){ z = z +y; }
   y = 2 * y; x = x/2;
}
return z;
```

ANSWER TO Q1

T(Program) = T(z= 0) + T(while) + T(return) = 1 + T(while) + 1 = 2 + T(while)

To estimate the time for the while-loop, we need the loop bound LB, which here is log x.

Since x is an 8 bit number, this gives us a loop bound of LB= 8.

T(while) = (LB+1)T(x!=0) + LB*(T(if) + T(y = 2*y) + T(x = x/2))

= 9*1 + 8*(T(if) + 1 +1)

Now, T(if) = T(x%2 !=0) + T(z = z+y) = 1 + 1 = 2

So, T(while) = 9*1 + 8*(2 + 1 + 1) = 9 + 32 = 41 time units

So, T(program) = 2 + T(while) = 2 + 41 = 43 time units.

Q2: ILP MODELING

Formulate the maximum execution time estimation of the program fragment in Question 3(A) using Integer Linear Programming (ILP). Clearly show the objective function and all constraints. Your ILP problem should only perform program path analysis and not micro-architectural modeling. The estimate produced by your ILP problem should be as tight as possible.

ANSWER TO Q2

```
z = 0;
while (x !=0){
      if (x %2 != 0){ z = z +y; }
      y = 2 * y; x = x/2;
 }
 return z;
```

Based on the control flow graph, the flow constraints are as follows. N_i is the execution count of basic block i, and $E_{i,j}$ is the execution count of the edge from basic block i to basic block j

$I = N_1 = E_{1,2}$

$E_{1,2} + E_{5,2} = N_2 = E_{2,6} + E_{2,3}$

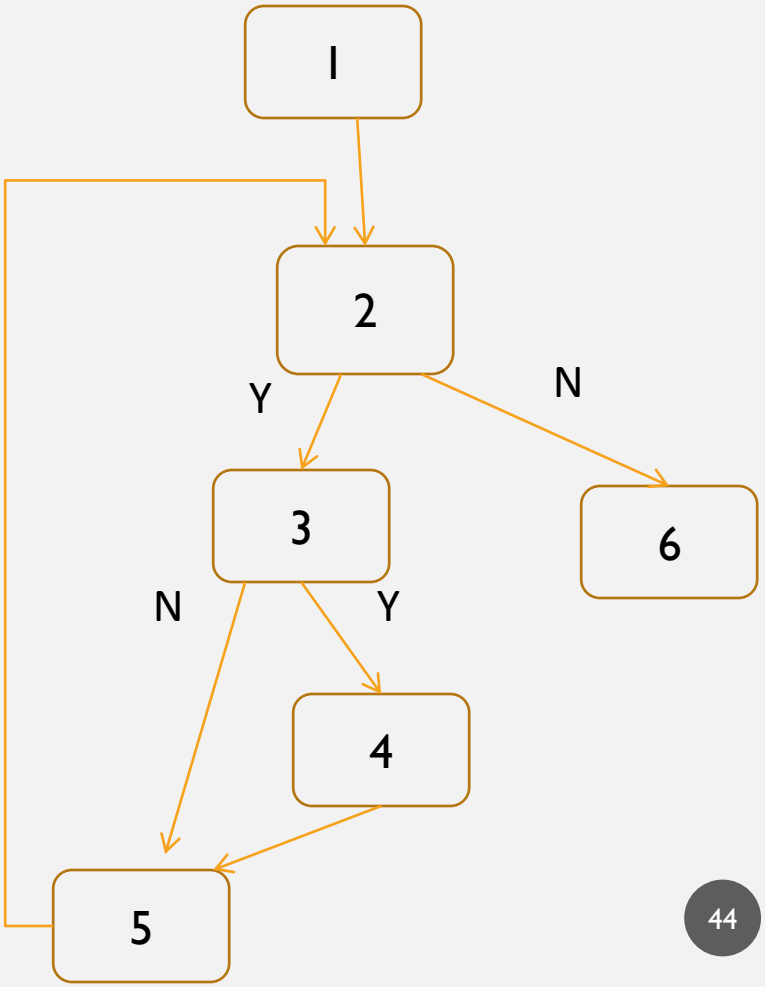
$E_{2,3} = N_3 = E_{3,4} + E_{3,5}$

$E_{3,4} = N_4 = E_{4,5}$

$E_{4,5} + E_{3,5} = N_5 = E_{5,2}$

$E_{2,6} = N_6 = I$

The loop bound accounts for the additional constraint $\mathbf{E_{5,2} \leq 8}$



ANSWER TO Q2

The objective function is $c1*N_1 + c2*N_2 + c3*N_3 + c4*N_4 + c5*N_5 + c6*N_6$

c1 is the execution cost of basic block 1, c2 is the execution cost of basic block 2 and so on.

Since assignments/conditions/returns all take 1 time unit, we get

c1 = 1, c2 = 1, c3 = 1, c4 = 1, c5 = 2, c6 = 1

The objective function can now be maximized w.r.t. flow constraints and loop bounds. We will use an ILP solver for this purpose.

The result from the ILP problem given above should be same as the result returned from timing schema.

Q3: COMPARISON

- Also, comment on how the estimate from your ILP problem will compare with the estimate you produced using Timing schema.

- Answer: The result from the ILP problem is exactly the same as the result returned from timing schema. There is no infeasible path information which can be given to the ILP solver as a constraint – to get a smaller estimate.