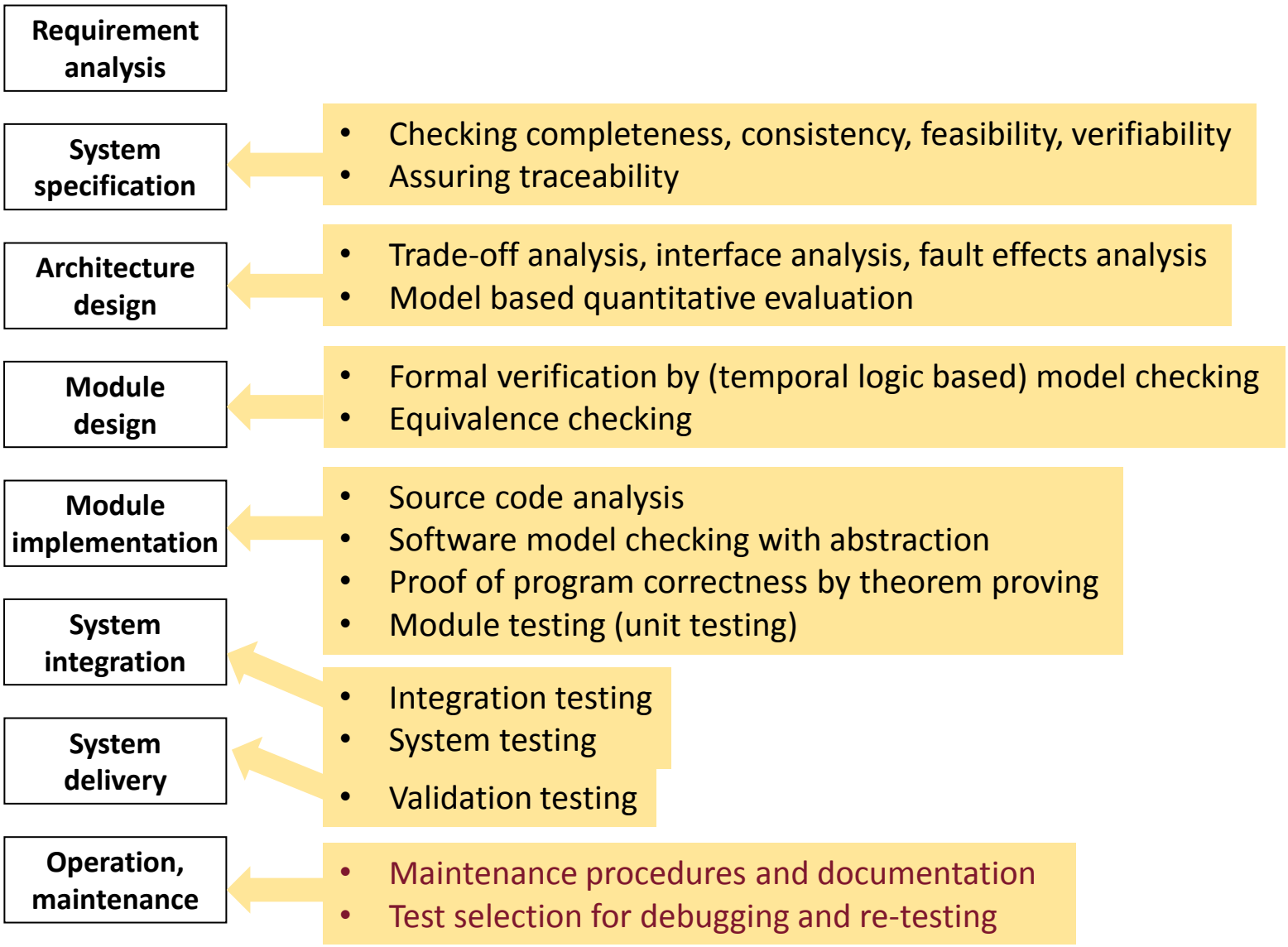
Verification during maintenance

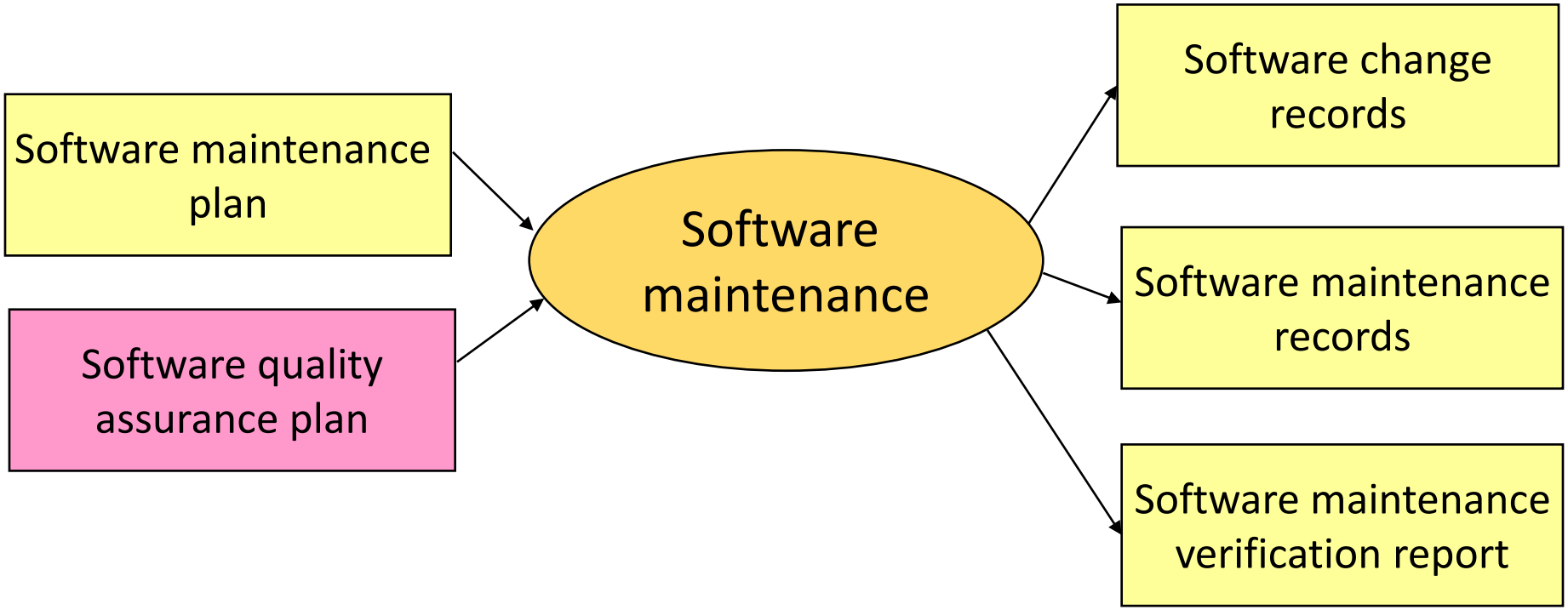
Istvan Majzik
majzik@mit.bme.hu

Budapest University of Technology and Economics
Dept. of Measurement and Information Systems

Typical development steps and V&V tasks



Software maintenance

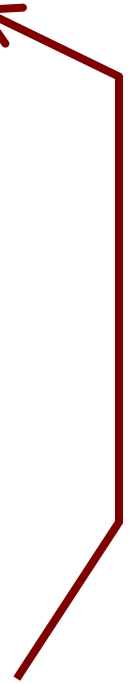


Software maintenance plan

- Procedures to be planned:
  - Submitting bug reports, collecting error logs
  - Planning, implementing, verifying maintenance
  - Approval of maintenance
- The competences, tools, planning, documentation shall be at the same level as in case of development
- Measures and techniques in safety standards:
  - Data collection and analysis
  - Effect analysis
- Documentation:
  - Software change records
  - Software maintenance records

Software change records and maintenance records

- Software change records
  - Belongs to the change activity
  - Request for change/modification
  - Specification of modification
  - Analysis of the effect of modification
  - Verification and validation of modification
- Software maintenance records
  - Belongs to the software element as “history”
  - Reference to the applied change records
  - Information related to the effects of a change
  - Tasks for repeated validation, regression test cases
  - Configuration and its history

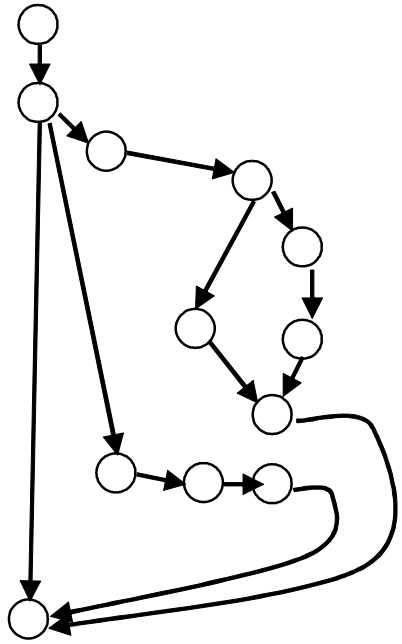


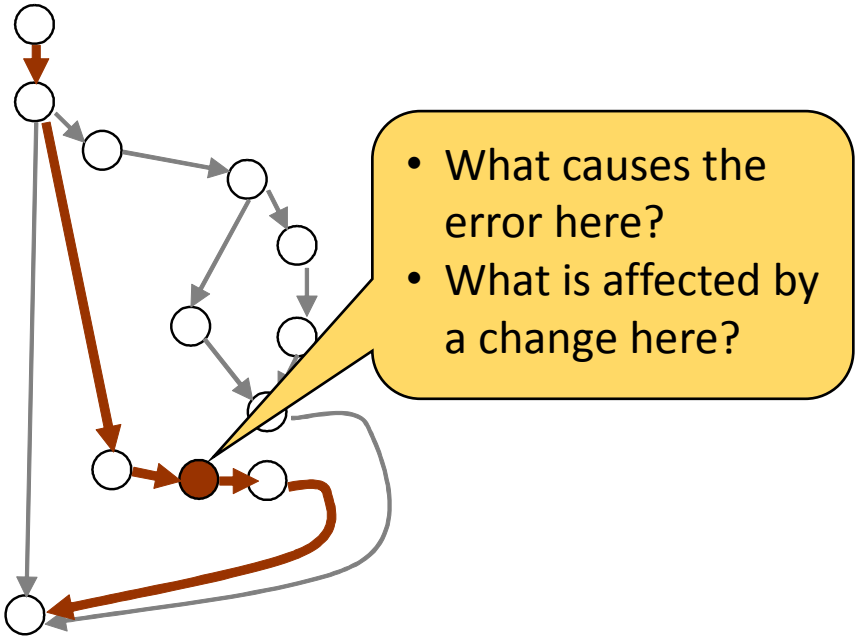
Supporting maintenance

- Tasks:
  - In case of modification: Effect analysis
  - In case of bug report: Debugging, repair
  - In case of both: Verification (testing and re-testing)
- Supporting technology: Program slicing
  - Analysis of the effects of a modification/repair
  - Reducing the complexity of debugging
  - Helping in test selection for testing and regression testing

Selecting relevant parts of the program: Program slicing

- Only a part (“slice”) of the program can be taken into account when debugging, verifying, (re-)testing the program
  - Debugging: What part of the program determines the value of a given variable at a statement?
  - Verification, testing: What is those part of the program that is influenced when a statement is changed?





Definition of static slicing

- Static slicing criterion: C = (V, I)
  - V is a subset of program variables
  - I is a selected statement of the program
- The static slice S of a program M according to the C = (V, I) slicing criterion:

  Executable subprogram of M, for which the following holds:
  - Executing M and S for any program input:
    the variables in V have the same values in both programs
    at the statement I
- Slicing: Selects those statements of program M that influence the values of variables in V at statement I

Example: Static slicing (1)

```
        procedure SumEven
        int n, sum, j
1       sum := 0
2       j := 2
3       n := read()
4       while (n > 0) do
5               sum := sum + j
6               j := j + 2
7               n := n - 1
        endwhile
8       write (sum)
```

- The program summarizes the first n even numbers.

Example: Static slicing (2)

```
     procedure SumEven
     int n, sum, j
1    sum := 0
2    j := 2
3    n := read()
4    while (n > 0) do
5            sum := sum + j
6            j := j + 2
7            n := n - 1
     endwhile
8    write (sum)
```

Criterion:

C=({j}, 6)

Influencing statements:

- 2: assignment to j
- 4: start of the while loop
- 3: influences the loop (n)
- 7: influences the loop (n)

Slice according to C=({j}, 6):

{2, 3, 4, 6, 7}

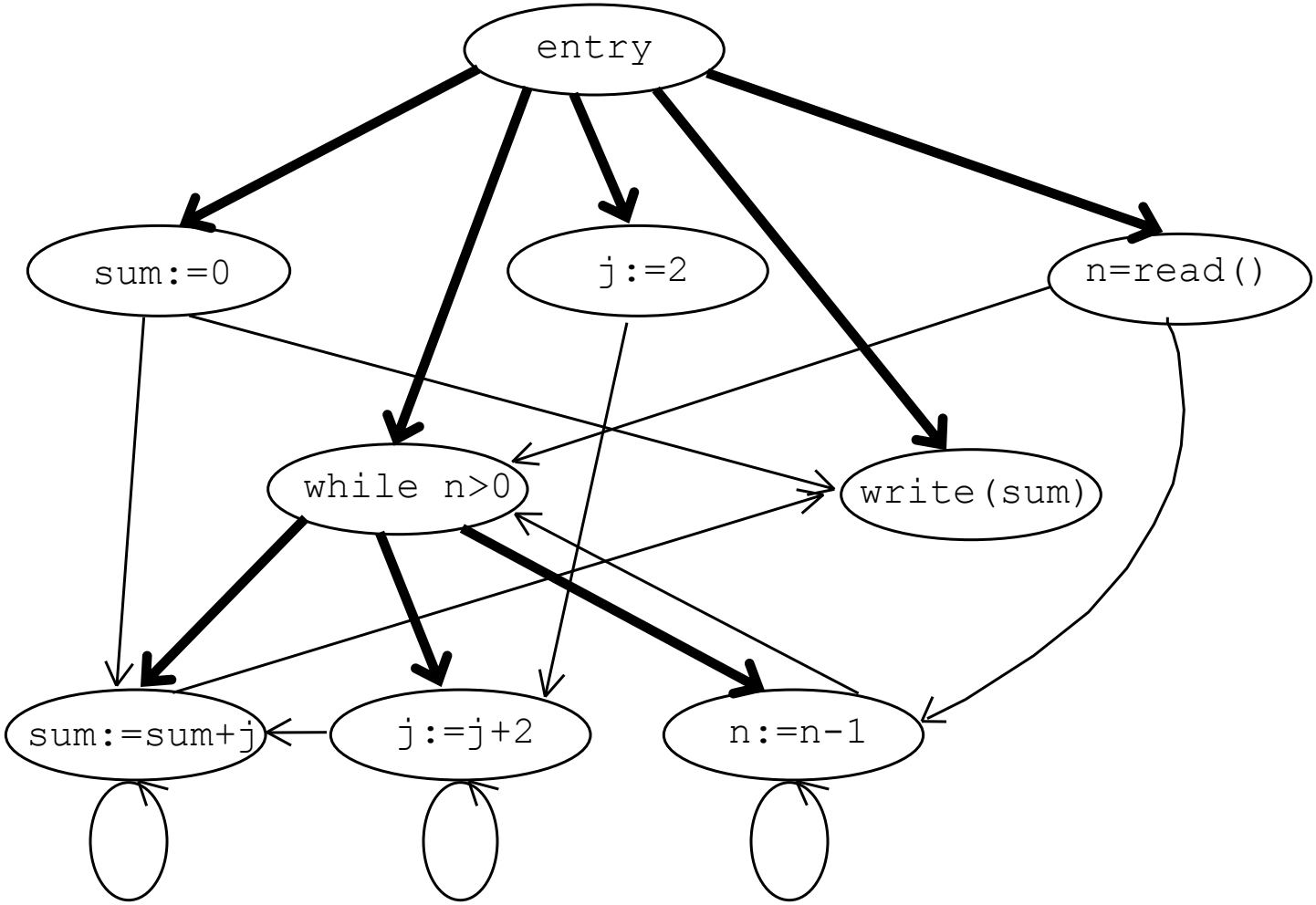
Slice according to C=({n},7):

{3, 4, 7}

Basis for slicing: Dependencies in the program

- A statement b is control dependent from statement a in the CFG of a program, if:
  - There is a program path to b that includes a,
  - There exists a path that includes a but does not reach b
- A statement b is data dependent from statement a, if :
  - The (a, b) pair of statements forms a def-use pair
- The program dependence graph (PDG) of a program:
  - Contains a unique entry node (with control dependences to all)
  - Includes program statements as nodes
  - There is an edge from node representing statement a to node representing statement b:
    - if b is control dependent from a,
    - or b is data dependent from a

Dependence graph of the example program



Control dependences: Thick edges

Data dependences: Thin edges

Determining a static backward slice

Forming a backward slice:

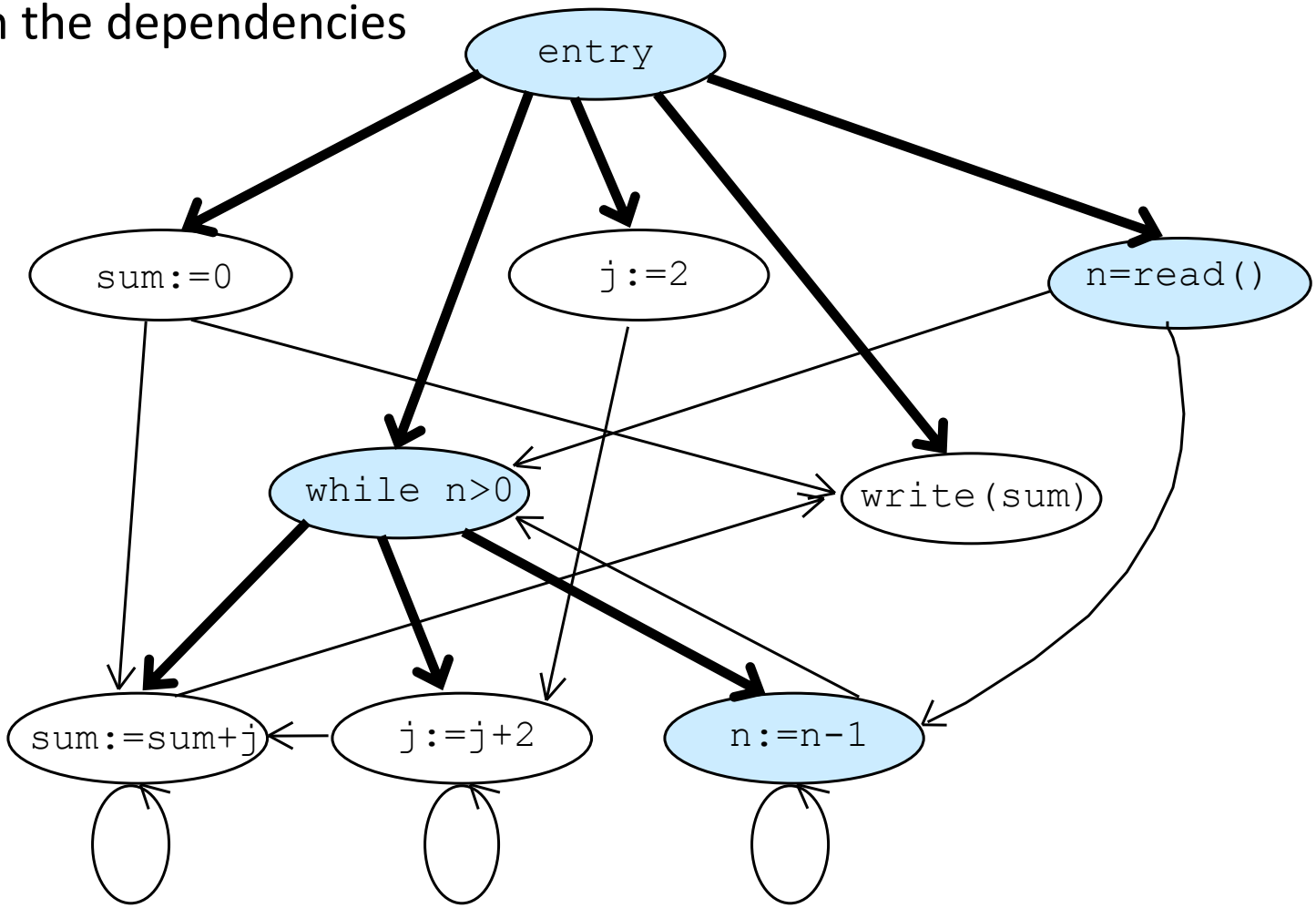
- Constructing the Program Dependence Graph
- Starting from the node representing the statement given in the slicing criterion
- Including in the slice those statements that are on the paths reachable by stepping backward on the dependency edges

Algorithm: List based processing for the PDG

1. Insert into the list the statement in the slicing criterion
2. Taking a statement from the list and including it in the slice
3. Inserting into the list those statements that are at the source of edges leading to the processed statement (and were not processed yet)
4. Continuing the algorithm from step 2 until the list becomes empty

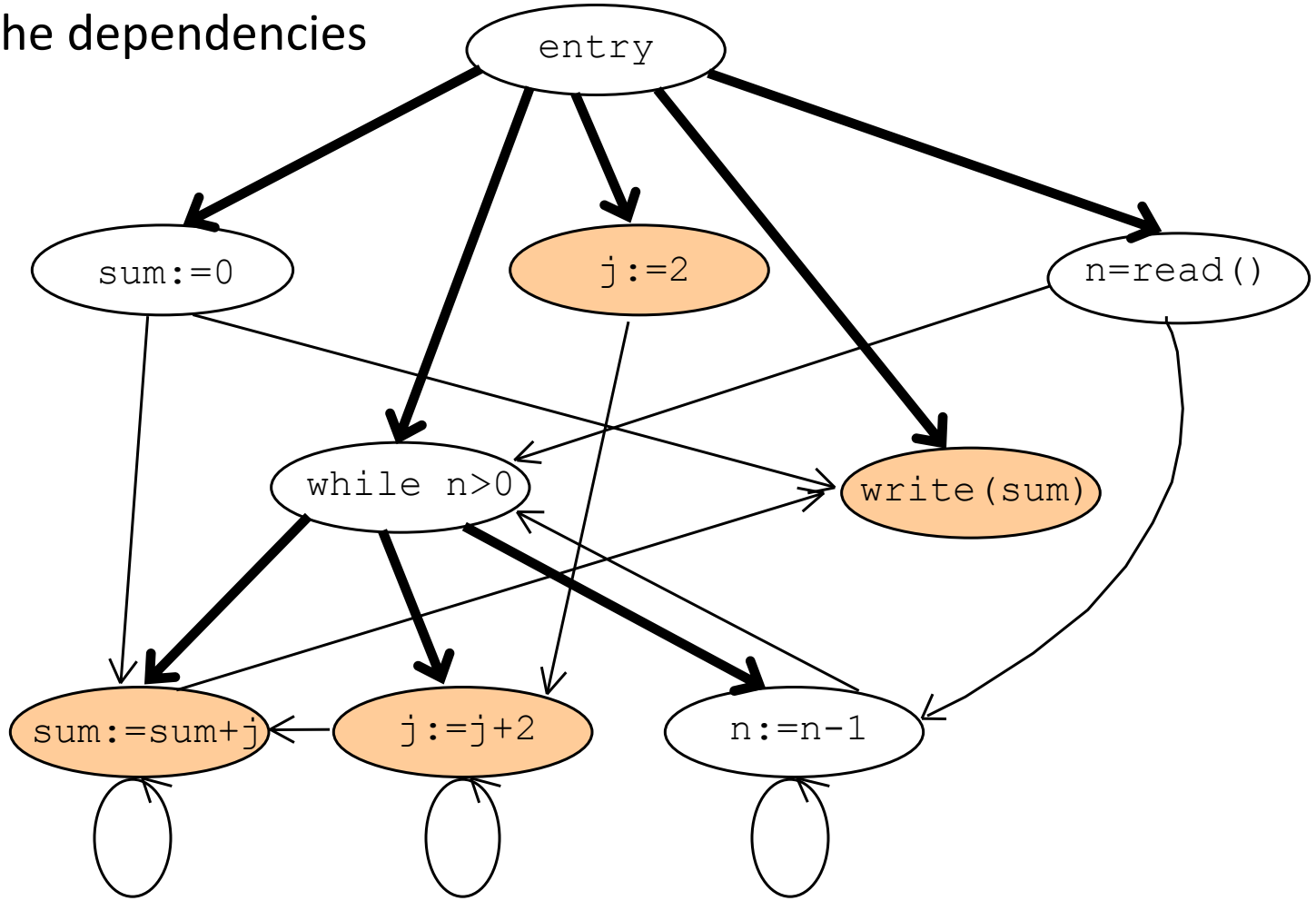
Backward slice for the example program

- Static backward slice for criterion C=({n},7)
- Built by starting from the statement in the criterion (n:=n-1) and stepping backward on the dependencies

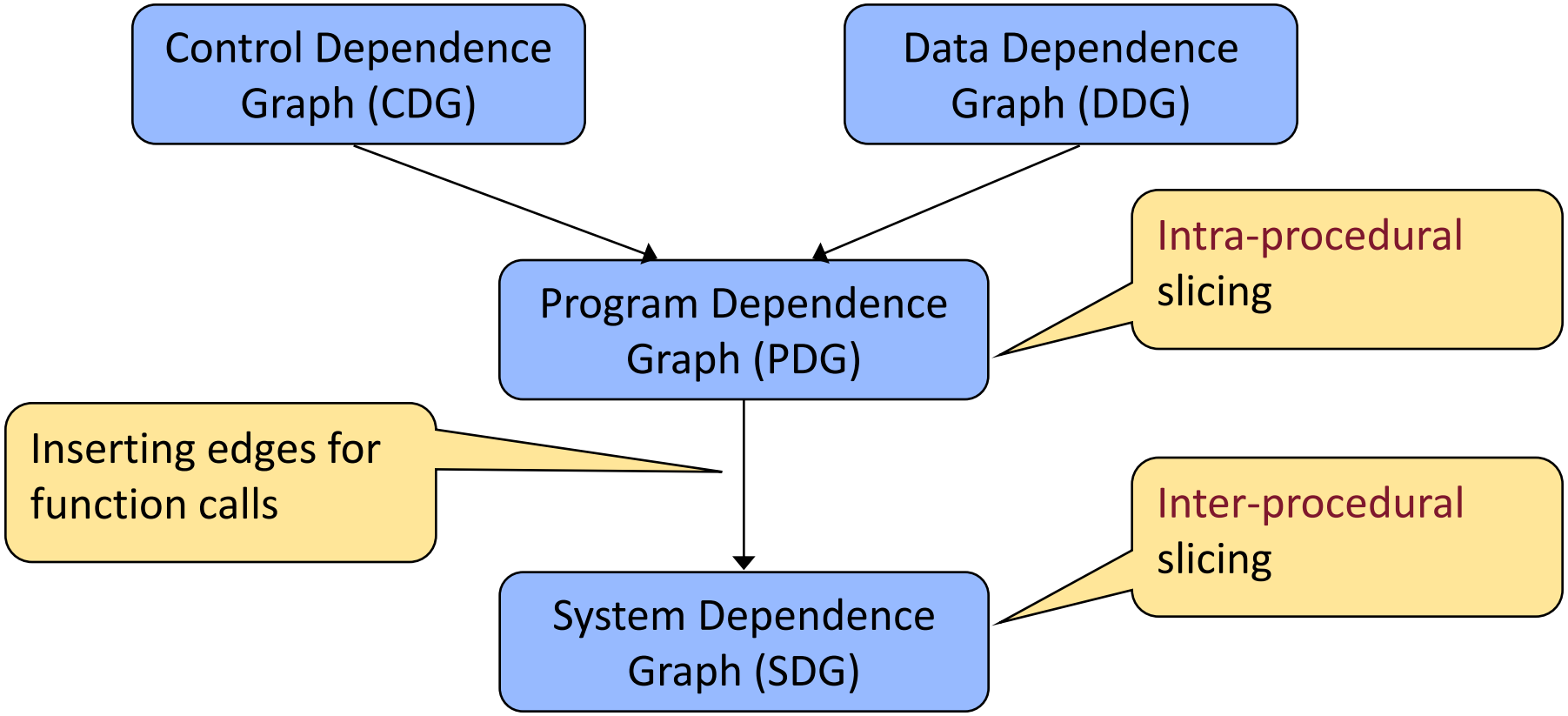


Forward slice for the example program

- Static forward slice for criterion C=({j},2)
- Built by starting from the statement in the criterion (j:=2) and stepping forward on the dependencies



Program structures for slicing



The construction of slices is a reachability problem

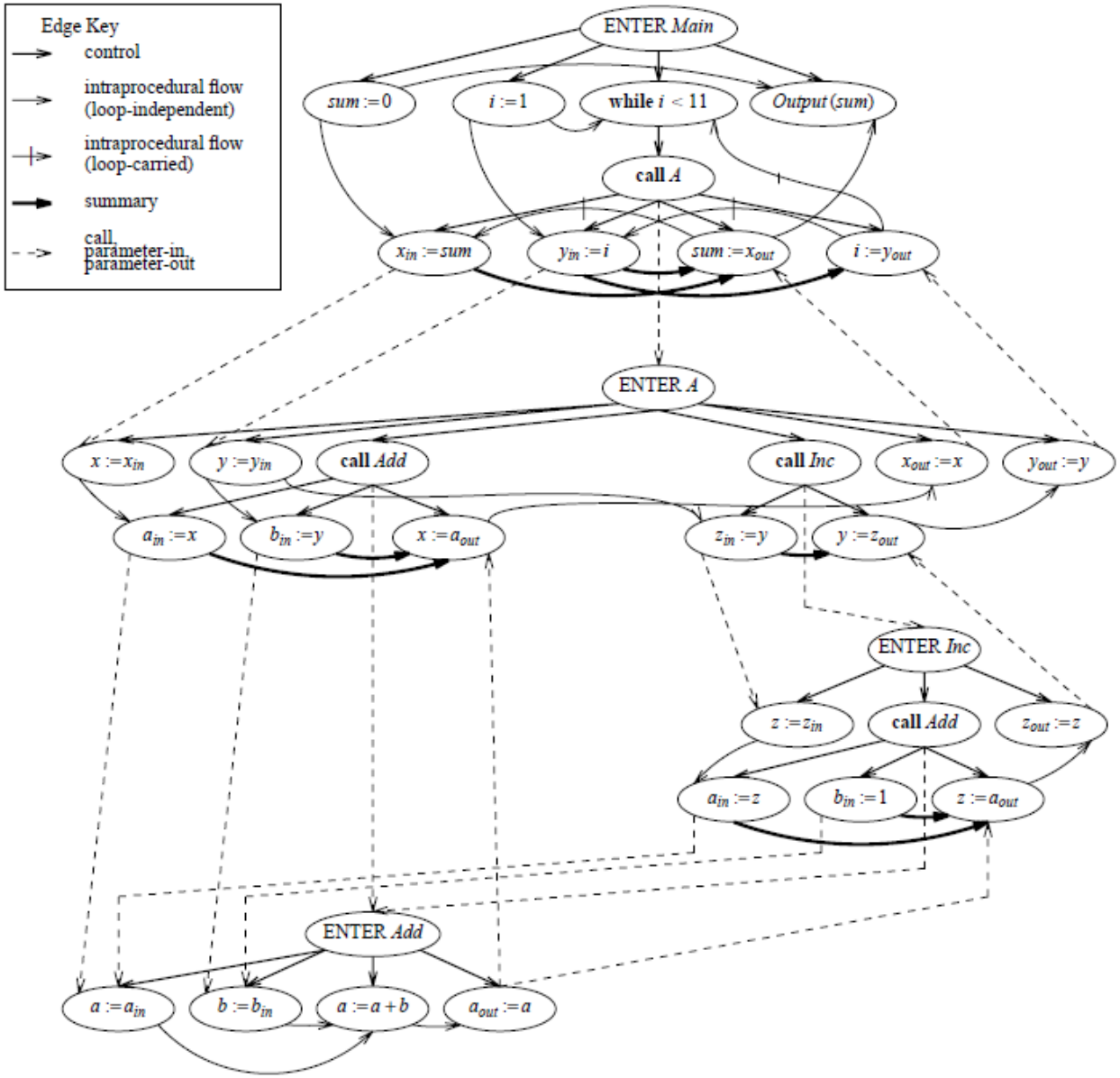
System Dependence Graph (example)

```
procedure Main
  sum := 0
  i := 1
  while i < 11 do
      call A(sum, i)
  od
  print(sum)
end

procedure A(x, y)
  call Add(x, y)
  call Increment(y)
return

procedure Add(a, b)
  a := a + b
return

procedure Increment(z)
  call Add(z, 1)
return
```



Summary: Using static slices

- Slicing result in smaller programs
  - Easier to handle and understand during debugging
  - Smaller code in case of testing
- Questions that can be answered using slices:
  - Backward slice: What are the statements that influence an erroneous value?
  - Forward slice: What are the statements that are influenced when a statement is changed? What shall be (re)tested?

Related problem:

Debugging on the basis of a concrete test input (that failed)

- The slice shall not consider any input but the concrete one
- The size of the slice can be further reduced

Definition of dynamic slicing

- Slicing is performed on the basis of a program path executed in case of a given input
  - Loops: May be executed several times in the path
- Dynamic slicing criterion: $C = (t, I^q, V)$
  - t is the input of the program (test input)
  - I^q is a selected statement (executed q times)
  - V is a subset of program variables
- Definition: Dynamic slice S of program M according to slicing criterion $C=(t, I^q, V)$:
  Executable subprogram of M for which the following holds:
  - Executing M and S for the given input t,

    the variables in V have the same values in both programs

    at the q-th execution of statement I

Dynamic slice of the example program (1)

```
   procedure SumEven
   int n, sum, j
1  sum := 0
2  j := 2
3  n =: read()
4¹ while (n > 0) do
5¹       sum := sum + j
6¹       j := j + 2
7¹       n := n - 1
   endwhile
8  write (sum)
```

Criterion:

$C=(n=1, 8^1, \{sum\})$

The loop is executed once (n=1).

Statements that influence 8:

- $5^{1:}$ value assignment (sum)
- 3: reading n (for the loop)
- 1 and 2: initial assignments

Dynamic slice:

{1, 2, 3, 5, 8}

Dynamic slice of the example program (2)

```
   procedure SumEven
   int n, sum, j
1  sum := 0
2  j := 2
3  n =: read()
4⁰ while (n > 0) do
5⁰       sum := sum + j
6⁰       j := j + 2
7⁰       n := n - 1
   endwhile
8  write (sum)
```

Criterion:

$C=(n=0, 8^1, \{sum\})$

The loop is not executed.

Statements that influence 8:

- 3: reading n (for the loop)
- 1: initial assignment

Dynamic slice:

{1, 3, 8}

Using dynamic slices

- Differences regarding program paths:
  - Static slice:
    All potential inputs (program executions)
    and all related dependencies
    are taken into account (there is no specific input)
  - Dynamic slice:
    Restricted to a specific input,
    that defined a concrete execution path,
    thus resulting in smaller slice than the static slice
- Debugging after a failed test
  - Looking for bugs in case of the given test input
  - Dynamic slice can be used

Overview of slicing types

- Types of slicing:
  - Executable – not executable (just for understanding)
  - Static – dynamic
  - Forward – backward
  - Interprocedural – intraprocedural
- The type of slicing depends on the usage
  - Debugging
  - Effect analysis, dependency analysis
  - Program understanding
  - Testing and retesting

Tools supporting slicing

- WPS - The Wisconsin Program Slicing System
  - Classic tool
- CodeSurfer (GrammaTech)
  - Static slicing on C programs
  - Impact Analysis: See what statements depend on a selected statement or instruction
  - Control Dependence Analysis: See the code that influences a statement's execution
- Unravel (NIST)
  - Program slicing tool that can be used to statically evaluate ANSI C source code
- MS Software Reengineering Toolkit (Semantics Designs)
  - General machinery for program control and data flow analysis
- Frama-C platform
- …