LECTURE 3

Software Engineering Concepts

MUHAMMAD MUNEEB ARSLAN

Project Management

TO A COMPLETE PROJECT…

The Aim of Project Management

- To complete a project:
  - On time
  - On budget
  - With required functionality
  - To the satisfaction of the client
  - Without exhausting the team

The Project Manager

- Create and maintain the schedule
- Should track progress against schedule
- Keep some slack in the schedule
- Be continually making adjustments:
  - Start activities before previous activity complete
  - Sub-contract activities
  - Renegotiate deliverables
- Keep senior management informed

Project Planning Methods

- The Critical Path Method, Gantt charts, Activity bar charts, etc. are roughly equivalent.
- These methods are best when:
  - Model is updated regularly (e.g., monthly)
  - The structure of the project is well understood
  - The time estimates are reliable
  - Activities do not share resources

An Example

- Deliverables:
  - 16 Written texts (bound in pairs)
  - 8 Television programs
  - 8 Radio programs
  - 4 Computer programs
  - 1 Home experimental kit (scientific calculator)
  - 4 Assignments and sample solutions

Flexibility

Schedule:

- Dates for broadcasting TV and radio programs are fixed. Printing and mailings can be accelerated if overtime is used.

Functionality:

- The course team can decide what goes into the components of the course.

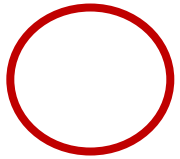
Resources:

- The size of the course team can be increased slightly.

Scheduling: Critical Path Method

An activity

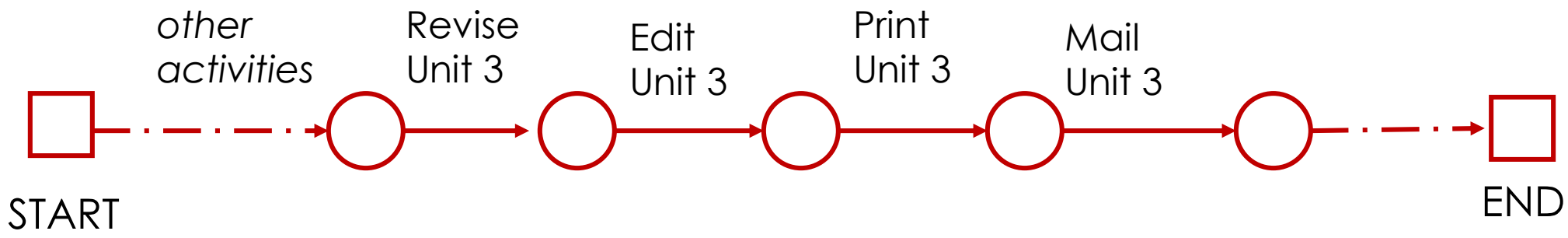
A dummy activity



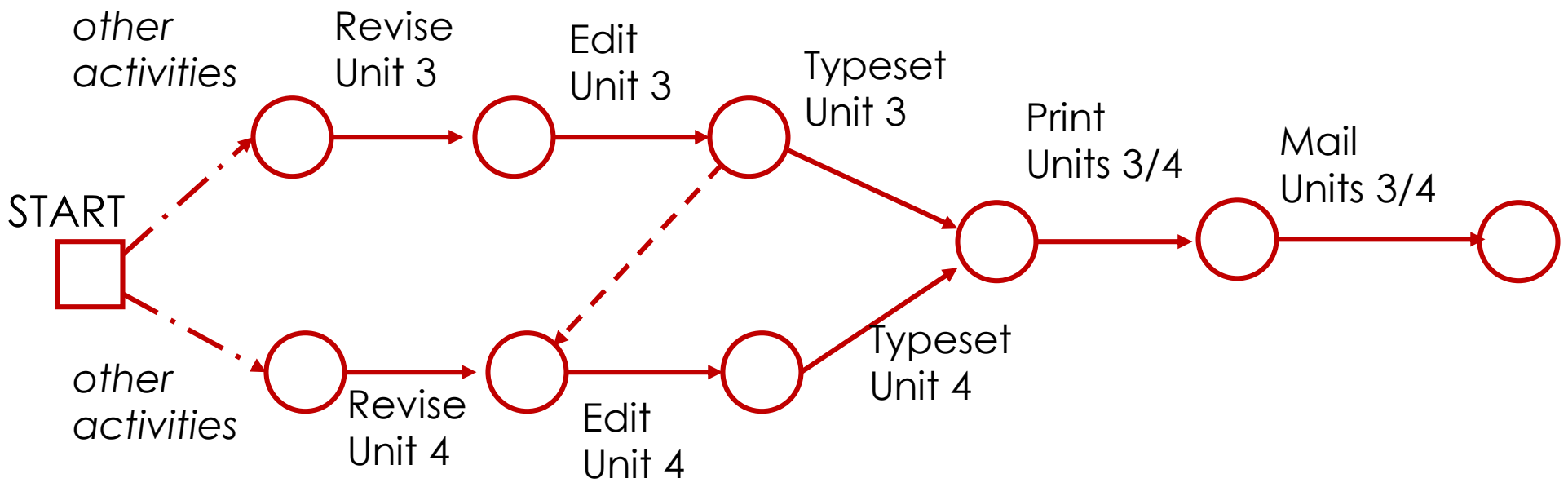
An event

A milestone

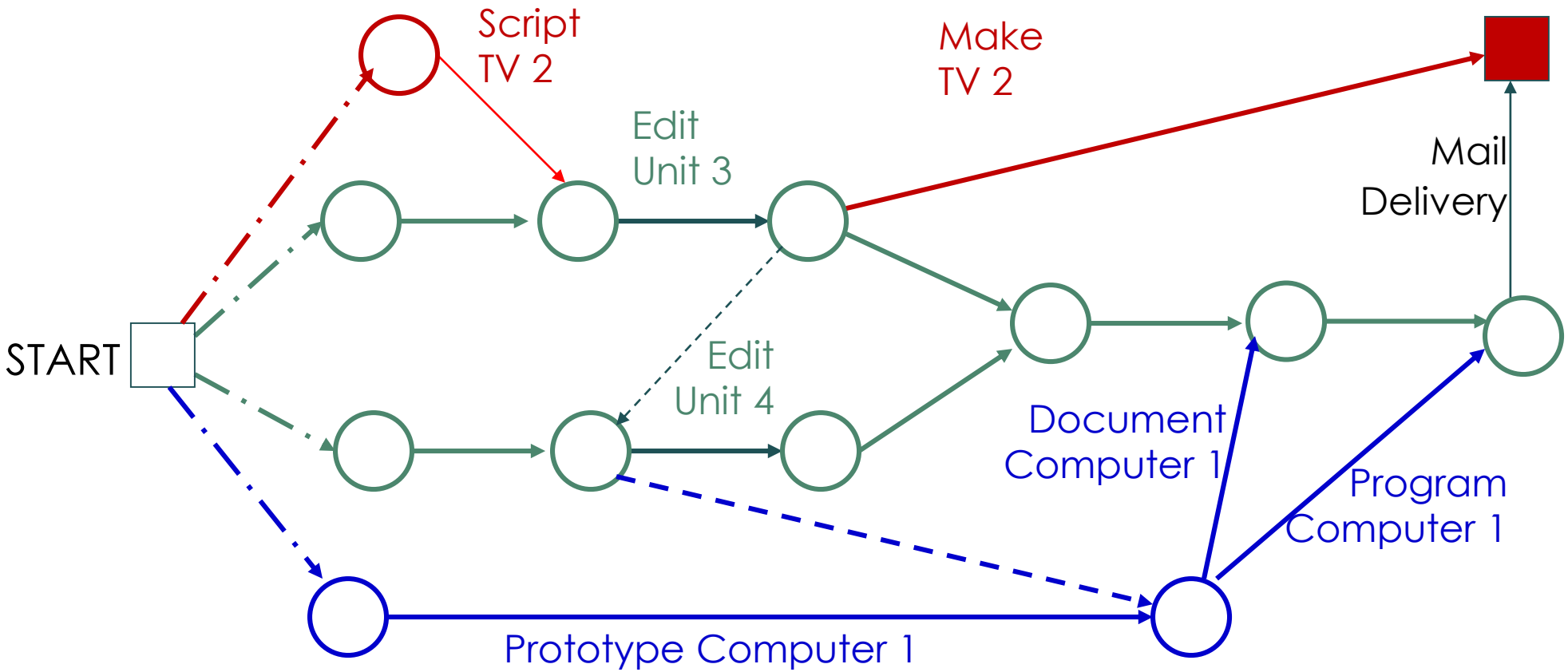
Critical Path Method



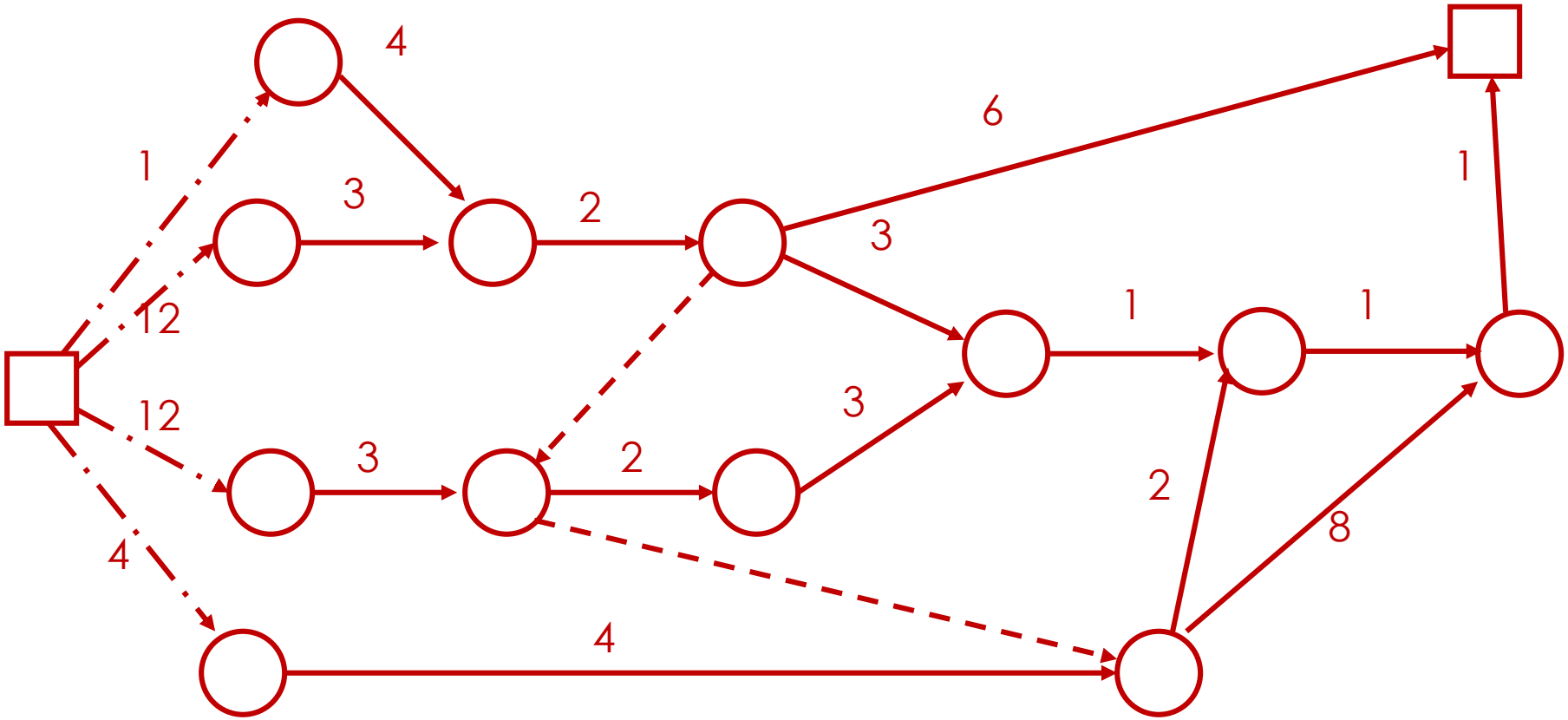
Critical Path Method



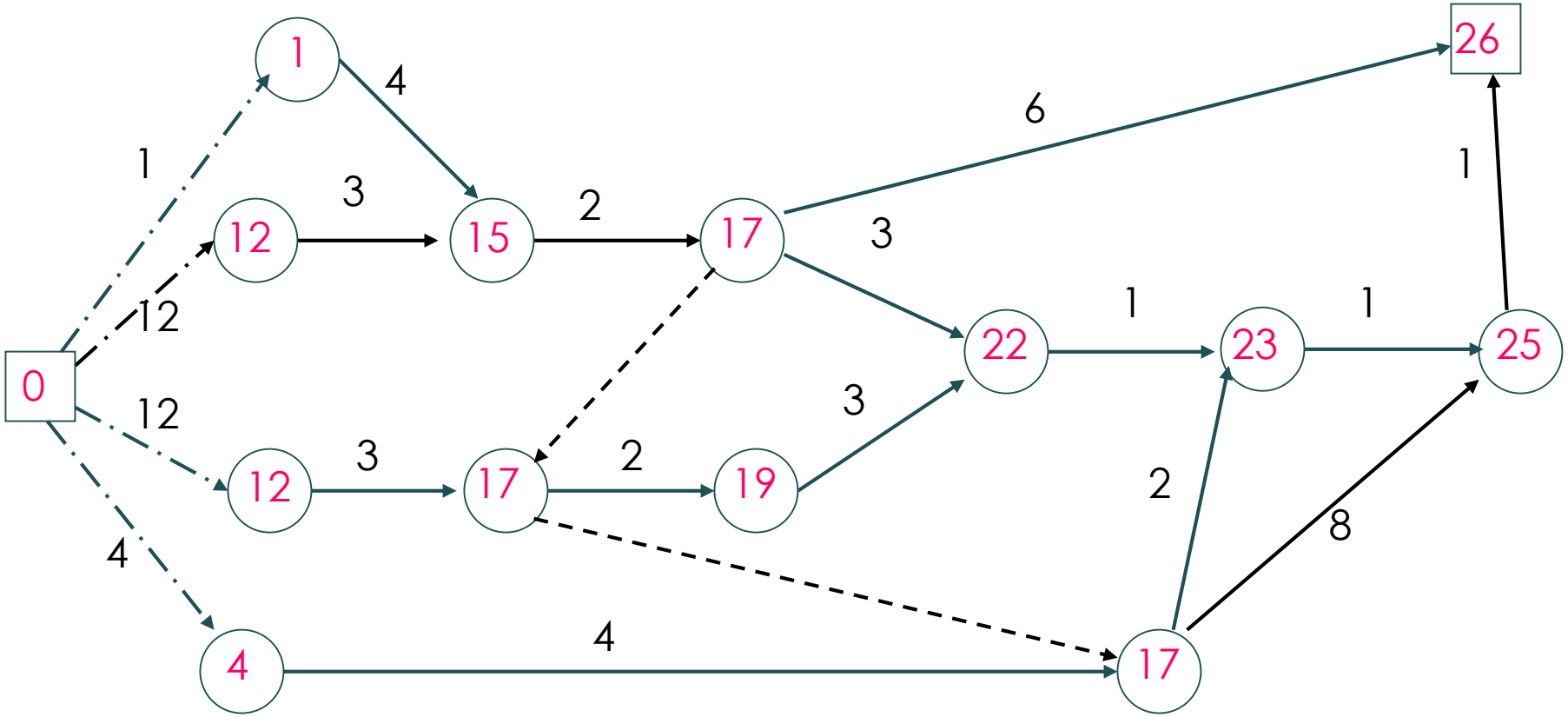
Critical Path Method



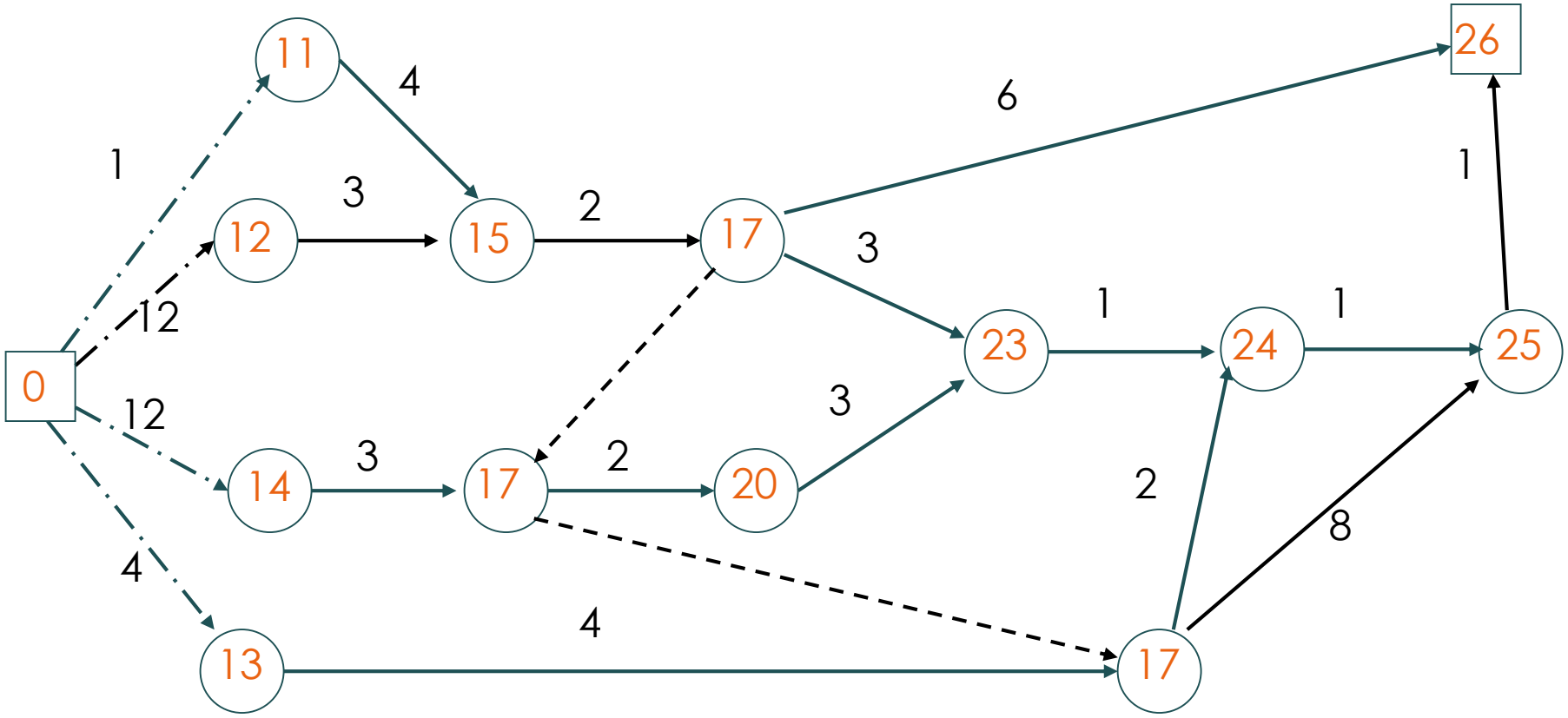
Time Estimates for Activities



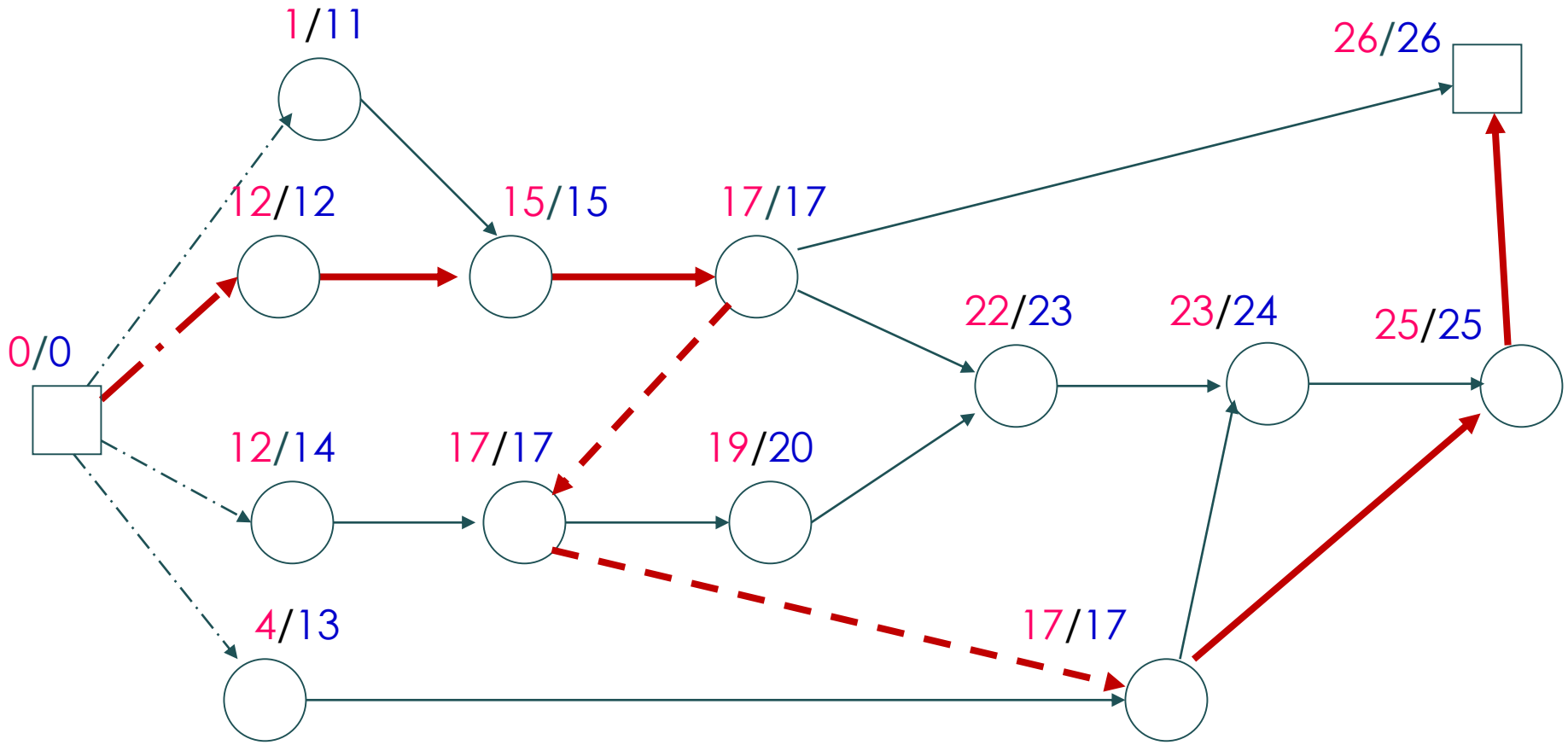
Earliest Start Dates



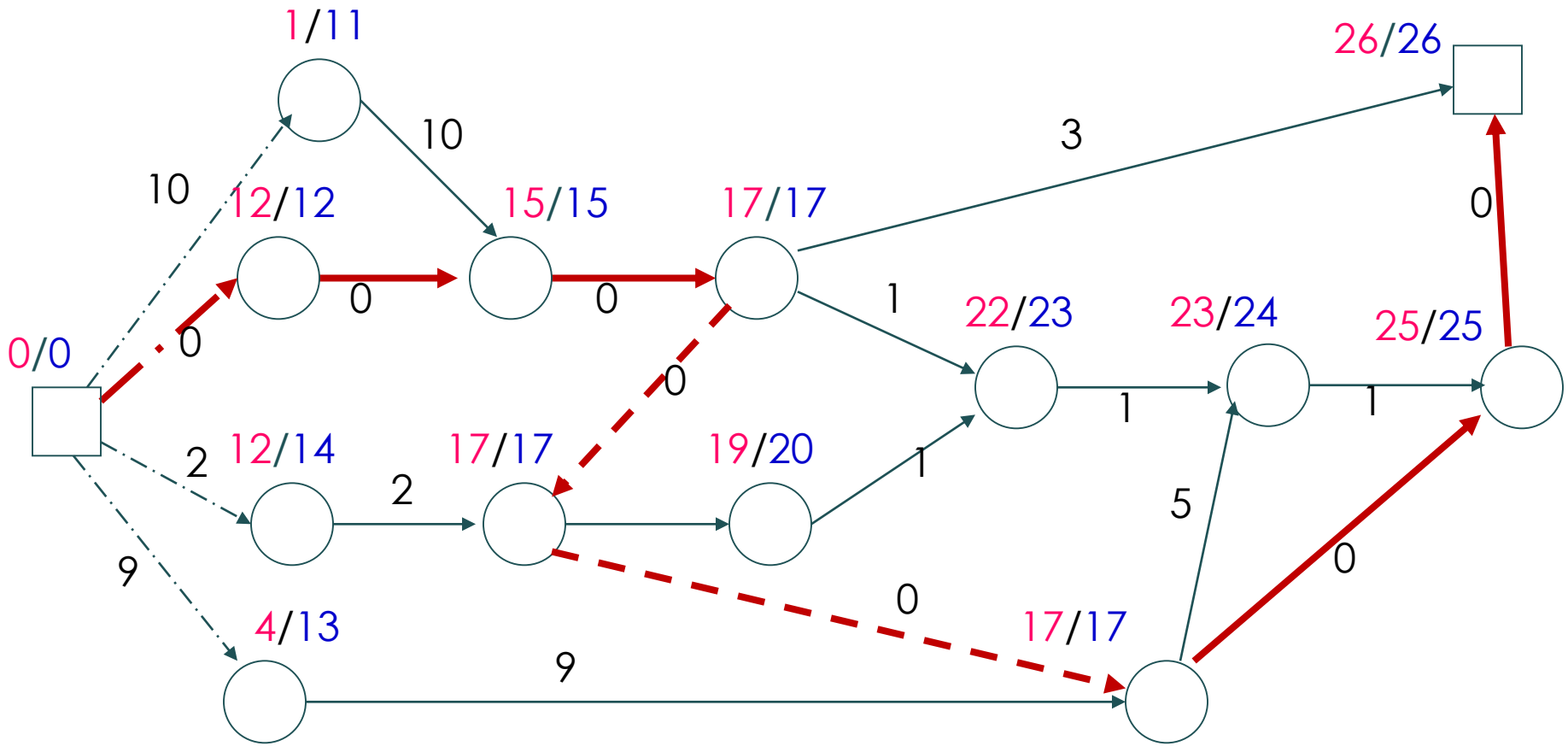
Latest Start Dates



Critical Path



Slack


1/11
26/26
10
3
10
12/12
15/15
17/17
0
0
0
1
0
0
22/23
23/24
25/25
0/0
0
1
1
2
12/14
17/17
19/20
2
1
5
9
0
0
4/13
17/17
9

Key Personnel

In computing, not all people are equal:

- The best are at least 5 times more productive
- Some tasks are too difficult for everybody

Adding more people adds communications complexity

- Some activities need a single mind
- Sometimes, the elapsed time for an activity can not be shortened.

What happens to the project if a key person is sick or quits?

Software Design

THE LIFE BELT…

Then and Now…

- At first, struggling with programming languages, small programs, math algorithms.
  - Worried about giving instructions to machine (efficiency)
  - "Think like a computer“
- Found that life cycle costs depend far more on how well communicates with people than how fast it runs.
- Separated the two and more emphasis began on
  - How to write software to communicate algorithms and structure to humans
  - How to structure design process itself.

Structured Programming

LETS MASTER THE COMPLEXITY.

Dijkstra, Hoare, Wirth…

- Construction of correct programs requires that programs be intellectually manageable
- Key to intellectual manageability is the structure of the program itself.
- Disciplined use of a few program building blocks facilitates correctness arguments.

Brought…

- Restricted control structures
- Levels of abstraction
- Stepwise refinement
- Program families
- Abstract data types

Dijkstra

- 3 main tools
  - Enumerative reasoning
  - Mathematical induction
  - Abstraction (e.g., variable, procedure, data type)

- Restrict programs to constructs that allow us to use these aids.
  - Sequencing and alternation (enumeration)
  - Iteration and recursion (induction)
  - Procedures, macros, and programmer-defined data types

Dijkstra

- Designed using "levels of abstraction"
  - System design described in layers
  - Higher levels could use services of lower levels
  - Lower levels could not access higher levels
- Lowest level implemented first
  - Provided a "virtual machine" for implementation of next level
  - Process continued until highest level completed.
- A "bottom up" technique

Wirth

- Divide and conquer
- A top-down technique for decomposing a system from preliminary design specification of functionality into more elementary levels.
- Program construction consists of sequence of refinement steps.

Program Families

Software will inevitably exist in many versions

- Different services for slightly different markets
- Different hardware or software platforms
- Different resource tradeoffs (speed vs. space)
- Different external events and devices
- Bug fixes

Think of development as a tree rather than a line

- Never modify a completed program
- Always begin with one of intermediate forms

System Structure

Structuring a large set of modules to form a system is an essentially distinct and different intellectual activity from that of constructing the individual modules.

Activity of producing detailed designs and implementations is programming in the small.

Modularization

Want to minimize, order, and make explicit the connections between modules.

Combining modularity with hierarchical abstraction turned out to be a very powerful combination

Module Specification

Started to distinguish between design and "packaging"

Design is process of partitioning a problem and its solution into significant pieces.

Packaging is process of clustering pieces of a problem solution into computer load modules that run within system time and space constraints without unduly compromising integrity of original design.

Reuse

- Assumed hundreds of reusable building-block module could be abstracted and added to program libraries.
- Why didn't happen?

Stepwise Refinement vs. Module Specification

Intermediate steps are programs that are complete except for implementation of certain operators and operands.

Intermediate stages are not programs. Instead they are specifications of externally visible collective behavior of program groups called modules.

Similarities

Precise representation of intermediate stages in program design.

Postponement of decisions: Important decisions postponed until late stages or confined to well-delineated subset of code.

Differences

Decision Making

SR: Decision-making order critical. May have to backtrack more than really want. Sequencing decisions made early because intermediate reps are programs.

MS: May be easier to reverse decisions without repeating so much work. Sequencing decisions made last.

Effort

SR: Less work than either classical approach (because keeps complexity in control).

MS: Significant amount of extra effort because only works if external characteristics of each module sufficiently well specified that code can be written without looking at implementation of other modules. In return, get independent development potential.

Minimizing Connectivity

Cohesion

Relationship between functions a module provides

Coupling

Relationship between modules, inter-module connections

Inter-module Friction

Smaller modules tend to be interfaced by "larger surfaces"

Replacement of module with large interface causes friction, requiring rewrites in other modules.

Advantages of Reducing Connectivity

Independent development (decisions made locally, do not interfere with correctness of other modules).

Correctness proofs easier to derive

Potential reusability increased.

Reduction in maintenance costs (less likely changes will propagate to other modules)

Comprehensibility (can understand module independent of environment in which used).

1980s

- OO Design comes:
  - Added inheritance, multiple inheritance, and polymorphism to ADT.
  - In process added complexity and increased some types of connectivity.
  - Lots of claimed advantages -- so far empirical evaluation is not supporting them well.

1990s

- Architecture
- Patterns
- Frameworks
- Kits
- etc.

Software Design Principles

Design is a creative, problem-solving activity.

No recipe for doing it - always need some type of "magic".

Quality and expertise of designers is determinant for success.

Simon

An expert has over 50,000 chunks of domain knowledge at hand.

Solving a problem involves mapping into knowledge available.

The larger this knowledge and the more accessible, the more successful the process will be.

Brooks, Curtis

Successful software development often depends on small number of exceptional designers who "think on a system level."

Curtis: Such people might not be particularly good programmers.

Design problem

- How to decompose system into parts
  - each with a lower complexity than system as a whole
  - while minimizing interaction between the parts such that the parts together solve the problem.

Four Primary Design Principles

Separation of concerns

Abstraction

Simplicity

Restricted visibility

DEAL WITH SEPARATE ASPECTS OF A PROBLEM SEPARATE.

Separation of concerns

Abstraction

- Identify important aspects of a phenomenon and ignore details that are irrelevant at this stage.
- Hierarchical abstraction: build hierarchical layers of abstraction
  - Procedural (functional) abstraction
  - Data abstraction
  - Control abstraction (abstract from precise sequence of events handled, e.g., nondeterminacy)

Simplicity

EMPHASIS ON SOFTWARE THAT IS CLEAR, SIMPLE, AND THEREFORE EASY TO CHECK, UNDERSTAND, AND MODIFY.

LOCALITY OF INFORMATION

Restricted visibility

Design Methods

Set of guidelines, heuristics, and procedures on how to go about designing a system.

Usually offer a notation to express result of design process.

Trying to provide a systematic means for organizing the design process and its products.

Design Methods

Design method may be based on:

- Functional decomposition
- Data flow
- Data structures
- Control flow
- Objects

Vary in degree of prescriptiveness

Life Belt or Leg Iron

- Will the adoption of a design method help the software development process (be a "life belt") or is there significant risk that its use will lead to suboptimum solution (be a "leg iron")?

David Budgen, Software Design Methods: Life Belt or Leg Iron (IEEE Software, Sept/Oct. 99)

Two general design characteristics

"Wicked" nature of any design process:

- Adopting a particular solution approach to a problem may make task of solving it more intractable, i.e., the design process is not neutral.

Expert designers engage in opportunistic behavior:

- As solution's form emerges, problem solving strategy is adapted to meet new characteristics that are revealed, i.e., expert designers do not follow a single method.

Conclusion!

Questions???