HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

SOICT

School of Information and Communication Technology

ONE LOVE. ONE FUTURE.

TRƯỜNG ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

IT3180 – Introduction to Software Engineering

6 – Project Management

ONE LOVE. ONE FUTURE.

Project management concerns

- Manager concerns about following issues:
  - Product quality
  - Risk Assessment
  - Measurement
  - Cost Estimation
  - Project Schedule
  - Customer Communication
  - Staffing
  - Other Resources
  - Project Monitoring

Why Project Fail?

- Changing customer requirement
- Ambiguous/Incomplete requirement
- Unrealistic deadline
- An honest underestimate of effort
- Predictable and/or unpredictable risks
- Technical difficulties
- Miscommunication among project staff

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

To provide visibility about the progress of a project

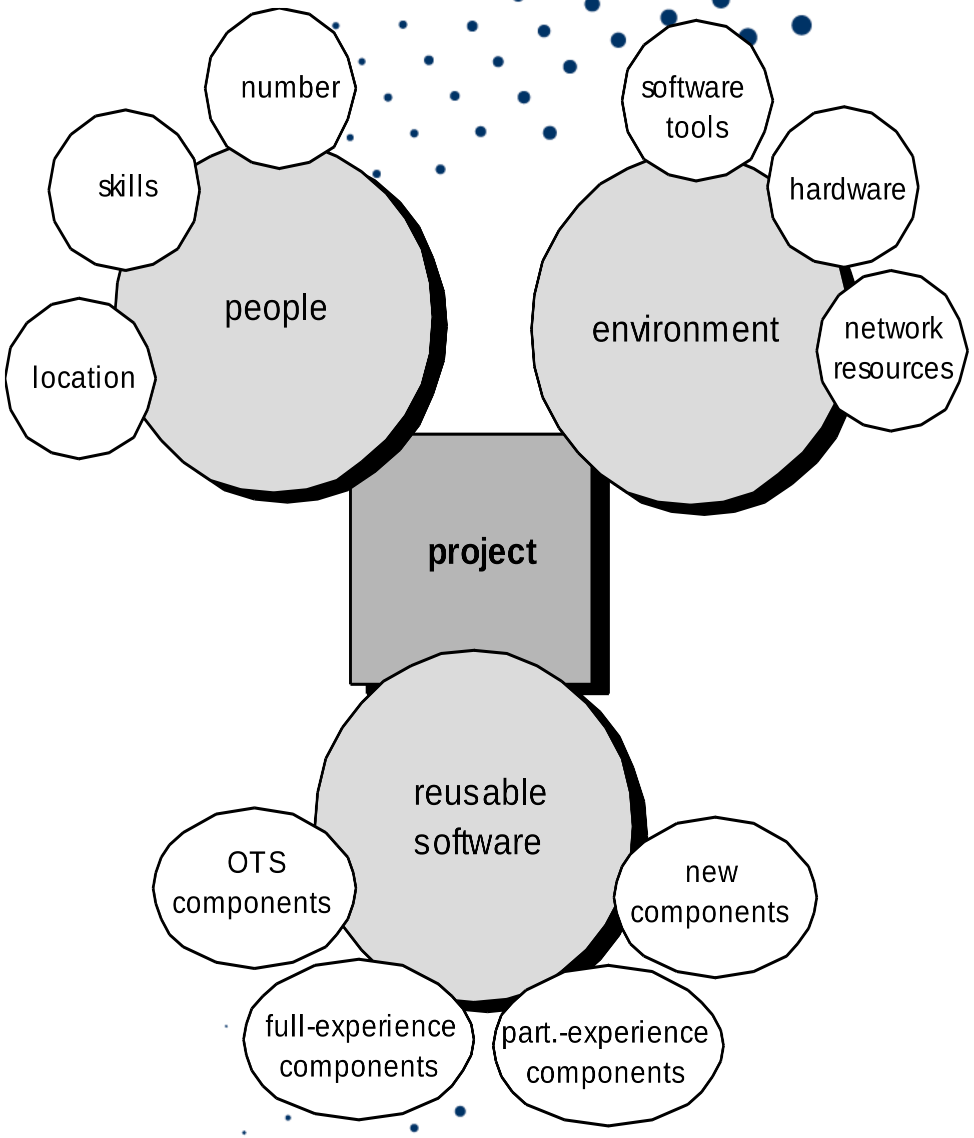
To give early warning of problems so that corrections can be made

The challenge of Project Management (1)

What do clients want to know?

- Will the system do what was promised? (**Function**)
- When will it be delivered? If late, how late? (**Time**)
- How does the cost compare with the budget? (**Cost**)

Resources



ONE LOVE. ONE FUTURE.

The challenge of Project Management (2)

Often, the software is a part of larger activity:

- If the system is a product, marketing and development must be combined (e.g., Microsoft Office)
- If the system has to work with other systems, developments must be coordinated (e.g., embedded systems in an automobile)

The challenge of Project Management (3)

BUT:

- Every software system is different.
- Most systems are not well specified, or the requirements change during development.
- Estimate time and effort is full of errors, even when the system is well understood.

Management Spectrum

- Effective project management focuses on four aspects of the project known as the 4 P's:
  - People - The most important element of a successful project. (recruiting, selection, performance management, training, compensation, career development, organization, work design, team/culture development)
  - Product - The software to be built (product objectives, scope, alternative solutions, constraint)
  - Process - The set of framework activities and software engineering tasks to get the job done (framework activities populated with tasks, milestones, work products, and QA points)
  - Project - All work required to make the product a reality. (planning, monitoring, controlling)

People

- Players of the project:
  - The Stakeholders
  - Team leaders
  - The Software Team
  - Agile Team (Implementer)
  - Coordination and Communication Issues.

Team Leaders

- **MOI** model for leadership
  - Motivation The ability to encourage (by "push or pull") technical people to produce to their best ability.
  - Organization The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product.
  - Ideas or Innovation. The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

- Characteristics of effective project managers (problem solving, managerial identity, achievement, influence and team building)

Software Team

- Organizational Paradigms
  - Closed paradigm — structures a team along a traditional hierarchy of authority. Less likely to be innovative when working within the closed paradigm.
  - Random paradigm — structures a team loosely and depends on individual initiative of the team members. It struggles when "orderly performance" is required.
  - Open paradigm — attempts to structure a team in a manner that achieves some of the controls associated with the closed paradigm but also much of the innovation that occurs when using the random paradigm
  - Synchronous paradigm — relies on the natural compartmentalization of a problem and organizes team members to work on pieces of the problem with little active communication among themselves

Software Team: Agile Team

- Small, Highly motivated project team also called Agile Team, adopts many of the characteristics of successful software projects.
- Team members must have trust in one another.
- The distribution of skills must be appropriate to the problem.
- Unconventional person may have to be excluded from the team, if team organized is to be maintained.
- Team is “self-organizing”
  - An adaptive team structure
  - Uses elements of organizational paradigm’s random, open, and synchronous paradigms
  - Significant autonomy

Product Scope

- Software Scope:
  - Context How does the software to be built fit into a larger system, product, or business context and what constraints are imposed as a result of the context?
  - Information objectives What customer-visible data objects are produced as output from the software? What data objects are required for input?
  - Function and performance What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?
- Software project scope must be unambiguous and understandable at the management and technical levels.

Problem Decomposition

- Sometimes called partitioning or problem elaboration
- Decomposition is applied in 2 major areas
  - Functionality that must be delivered.
  - Process that will be used to deliver it.
- Once scope is defined …
  - It is decomposed into constituent functions
  - It is decomposed into user-visible data objects
  - or
  - It is decomposed into a set of problem classes
- Decomposition process continues until all functions or problem classes have been defined
- Decomposition will make planning easier.

The Process

- Process model chosen must be appropriate for the:
  - Customers and developers,
  - Characteristics of the product, and
  - Project development environment
- Once a process framework has been established
  - Consider project characteristics
  - Determine the degree of thoroughness required
  - Define a task set for each software engineering activity
    - Task set =
      - Software engineering tasks
      - Work products
      - Quality assurance points
      - Milestones

Aspects of Project Management (1)

- **Planning**
  - Outline schedule during feasibility study
  - Full schedule for each part of a project (e.g., each process step, iteration, or sprint)
- **Contingency planning**
  - Anticipate possible problems (risk management)

Aspects of Project Management (2)

- **Progress tracking**
  - Regular comparison of progress against plan
  - Regular modification of the plan
  - Changes of scope, etc. made jointly by client and developers
- **Final analysis**
  - Analysis of project for improvements during next project

Terminology (1)

- **Deliverable**
  - Work product that is provided to the client (mock-up, demonstration, prototype, report, presentation, documentation, code, etc.)
  - Release of a system or subsystem to customers and users
- **Milestone**
  - Completion of a specified set of activities (e.g., delivery of a deliverable, completion of a process step, end of a sprint)

Terminology (2)

- Activity
  - Part of a project that takes place over time (also known as a **task**)
  - Release of a system or subsystem to customers and users
- Event
  - The end of a group of activities, e.g., agreement by all parties on the budget and plan
- Dependency
  - An activity that cannot begin until some event is reached
- Resource
  - Staff time, equipment, or other limited resource required by an activity

Standard approach to Project Management

- The **scope** of the project is defined early in the process.
- The development is divided into **tasks** and **milestones**.
- **Estimates** are made of the **time** and **resources** needed for each task.
- The estimates are combined to create a **schedule** and a **plan**.
- **Progress** is continually **reviewed** against the plan, perhaps weekly.
- The **plan** is **modified** by changes to scope, time, resources, etc.

Typically the plan is managed by a **separate project management team**, not by the software developers.

Used with the Modified Waterfall Model and Iterative Refinement.

Agile Approach to Project Management

- Planning is divided into high level **release forecasting** and low level **detailed planning**.
- Release planning is a best guess, high level view of what can be achieved in a sequence of **time-boxes**.
- Release plans are continually **modified**, perhaps daily.
- **Clients** and **developers** take joint control of the release plans and choice of sprints.
- For each **time-box**, the **team** plans what it can achieve.

The team may use **Gantt charts** or other conventional planning tools.

Project Planning Tools

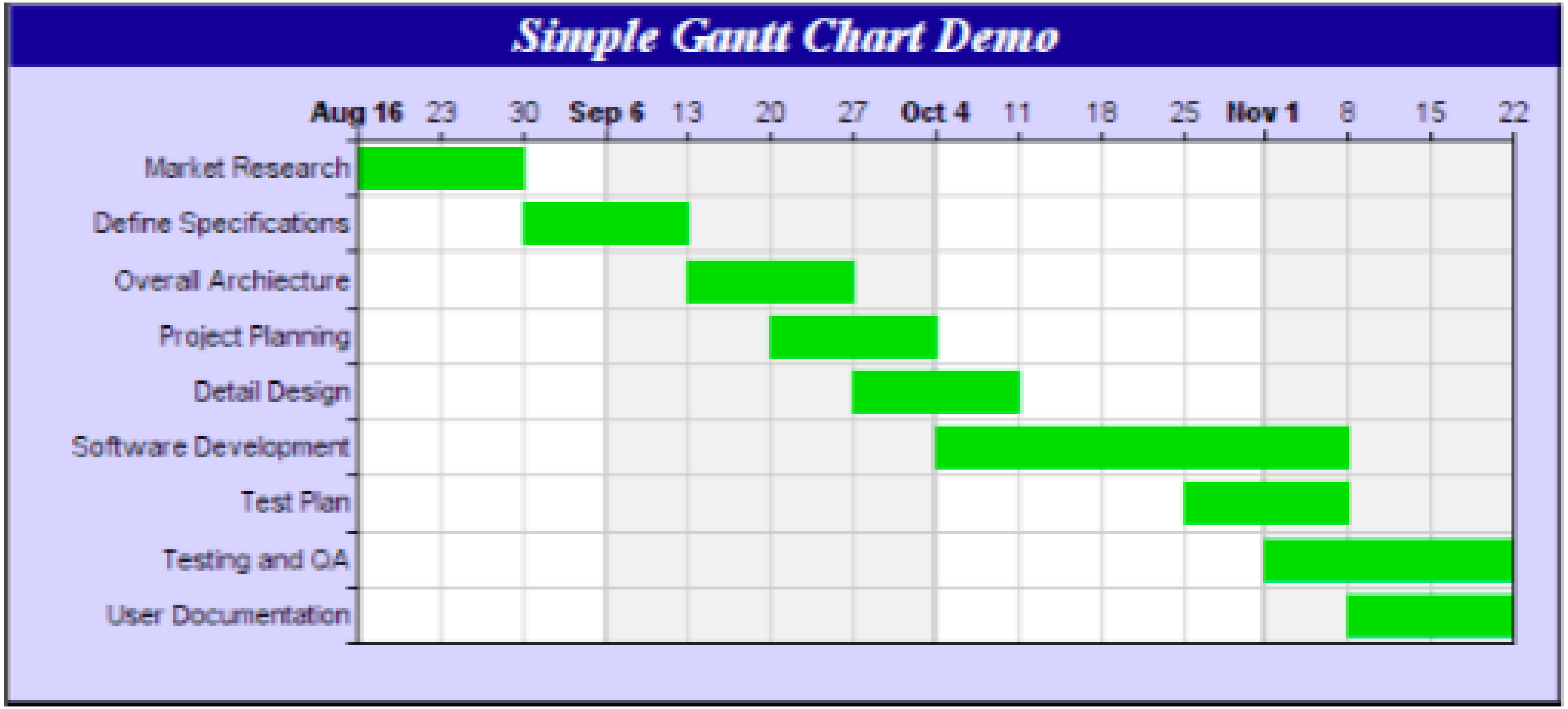
Critical Path Method, Gantt Charts

- Build a work-plan from activity data
- Display work-plan in graphical or tabular form.

Project planning software (e.g., Microsoft Project)

- Maintain a database of activities and related data
- Calculate and display schedules
- Manage progress reports

A Simple Gantt Chart



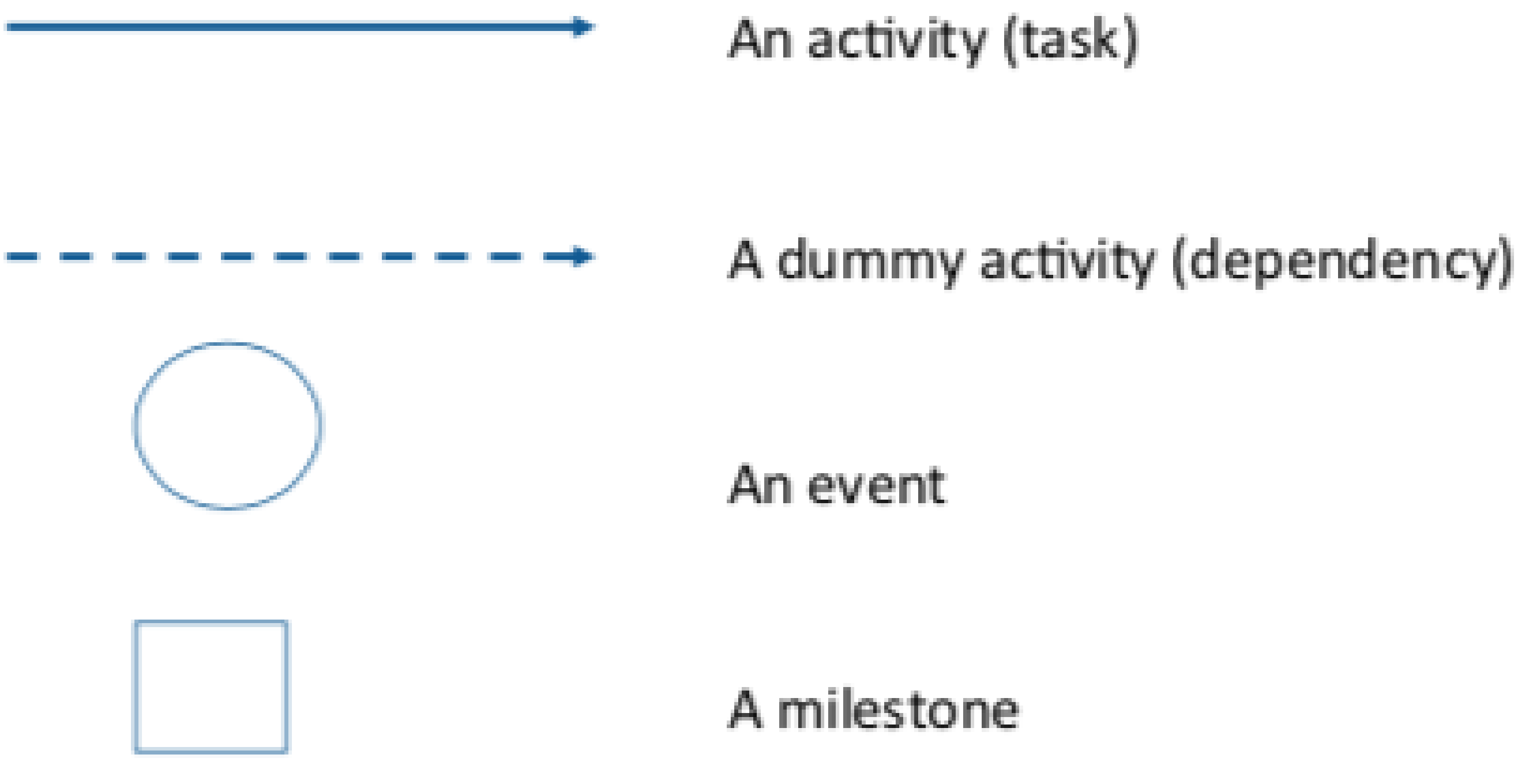
Source: Advanced Software Engineering Limited

Gantt Chart (1)

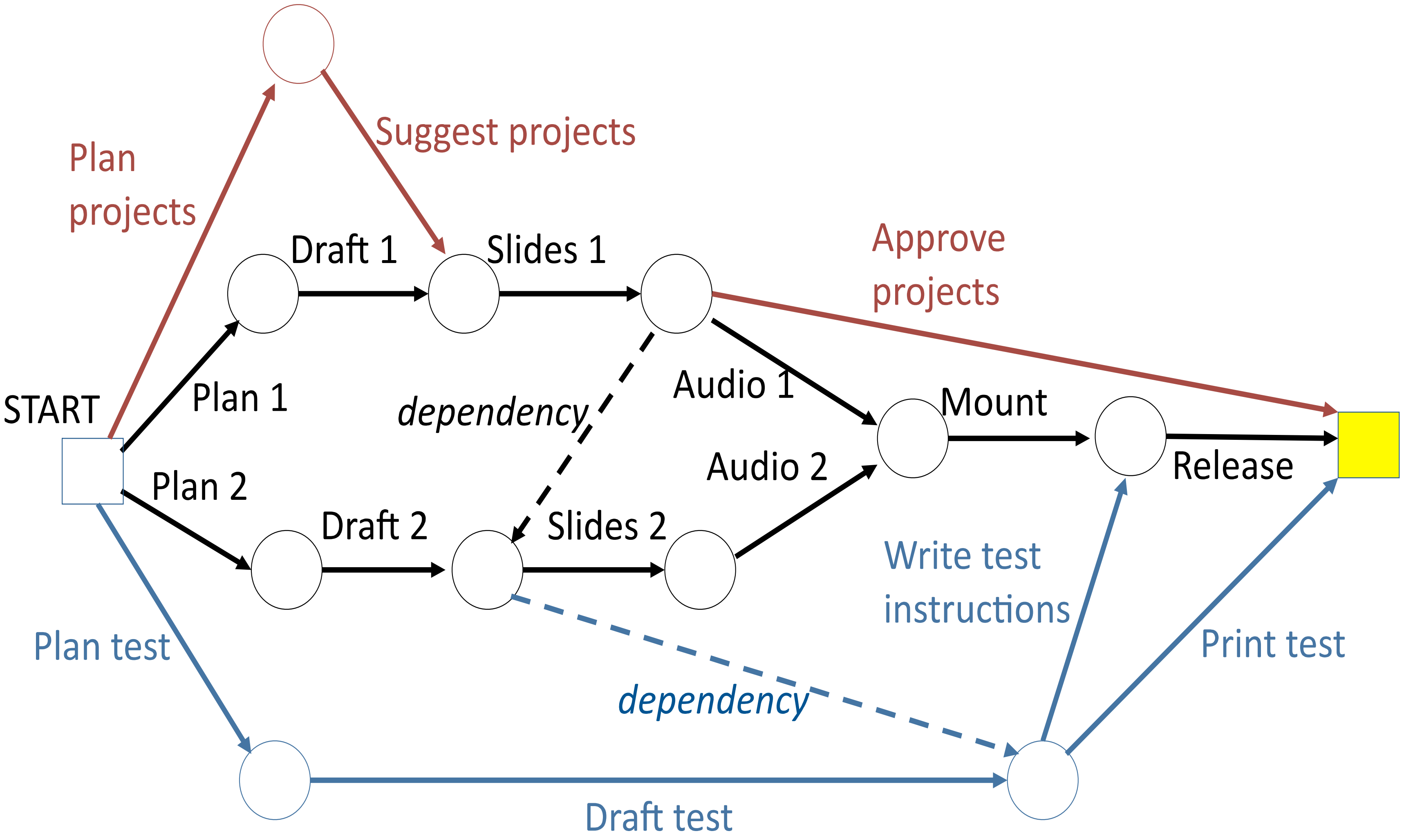
Used for small projects, single time-boxes, and sprints

- Dates run along the top (days, weeks, or months).
- Each row represents an activity.
- Activities may be sequential, in parallel or overlapping.
- The schedule for an activity is a horizontal bar.
- The left end marks the planned beginning of the task.
- The right end marks the expected end date.
- The chart is updated by filling in each activity to a length proportional to the work accomplished. This is often difficult.
- Progress to date can be compared with the plan by drawing a vertical line through the chart at the current date.

A group of scheduling techniques that emphasizes dependencies:



Example: Activity Graph



Scheduling using Activity Graphs: History

PERT

- Program Evaluation and Review Technique introduced by the U.S. Navy in 1957 to support the development of its Polaris submarine missile program.

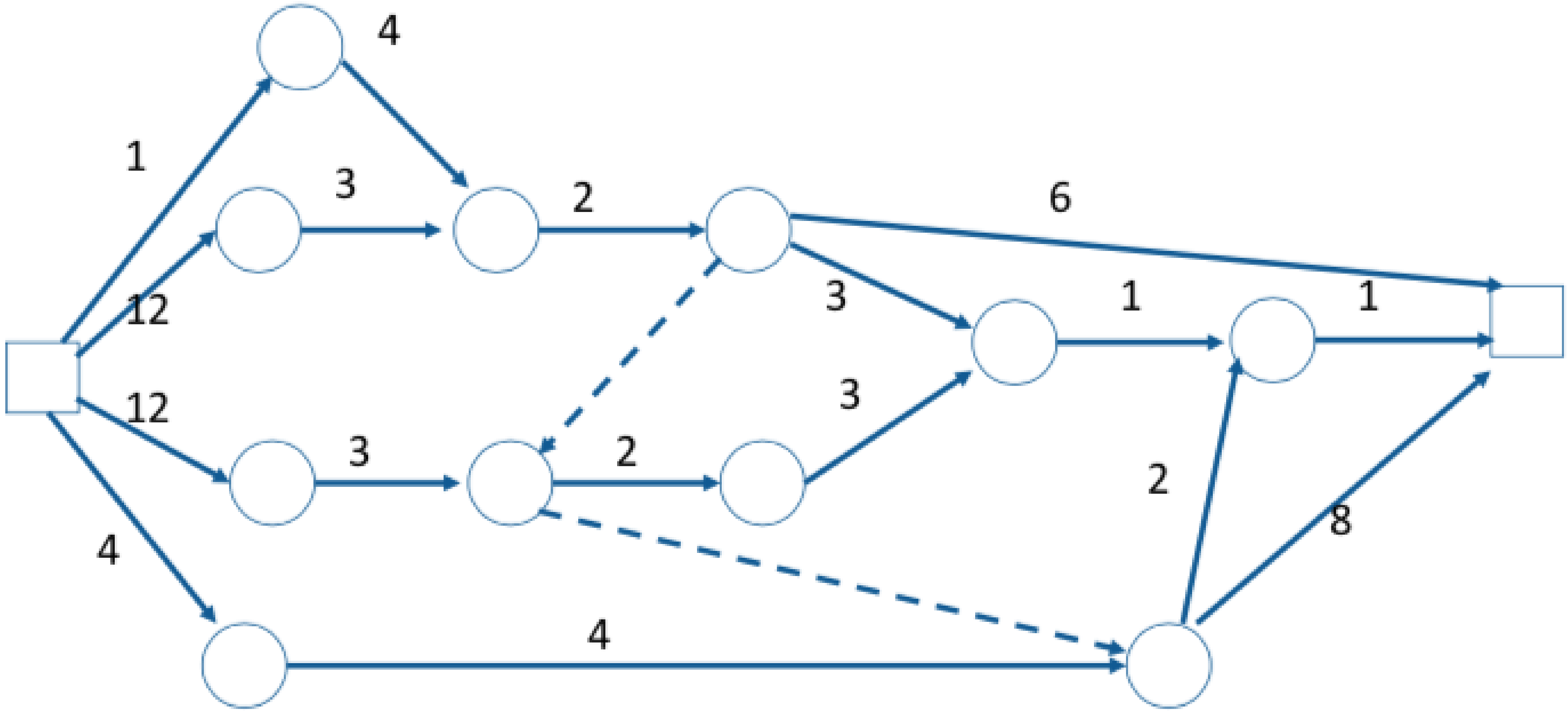
PERT/Time

- Activity graph with three time estimates (shortest, most probable, longest) on each activity to compute schedules.
- Because of the difficulty of obtaining good time estimates, usually only one esDmate is made. This is called the **Critical Path Method**.

PERT/Cost

- Added scheduling of resources (e.g., facilities, skilled people, etc.)

Time Estimates for Activities (weeks)



Example: Building a house

- Project activities:
  - install landscaping
  - pour foundations
  - frame walls
  - install plumbing systems
  - get permits
  - install electrical systems
  - move in

Example: Building a house

Activities in order, Durations, Labels, Dependencies

Project tasks	Durations	Labels	Preds.	Post
Get permits				
Pour foundations				
Frame walls				
Install plumbing systems				
Install electrical systems				
Install landscaping				
Move in				

Example: Building a house

Activities in order, Durations, Labels, Dependencies

Project tasks	Durations	Labels	Preds.	Post
Get permits	2			
Pour foundations	6			
Frame walls	5			
Install plumbing systems	4			
Install electrical systems	6			
Install landscaping	9			
Move in	3			

Example: Building a house

Activities in order, Durations, Labels, Dependencies

Project tasks	Durations	Labels	Preds.	Post
Get permits	2	A		
Pour foundations	6	B		
Frame walls	5	C		
Install plumbing systems	4	D		
Install electrical systems	6	E		
Install landscaping	9	F		
Move in	3	G		

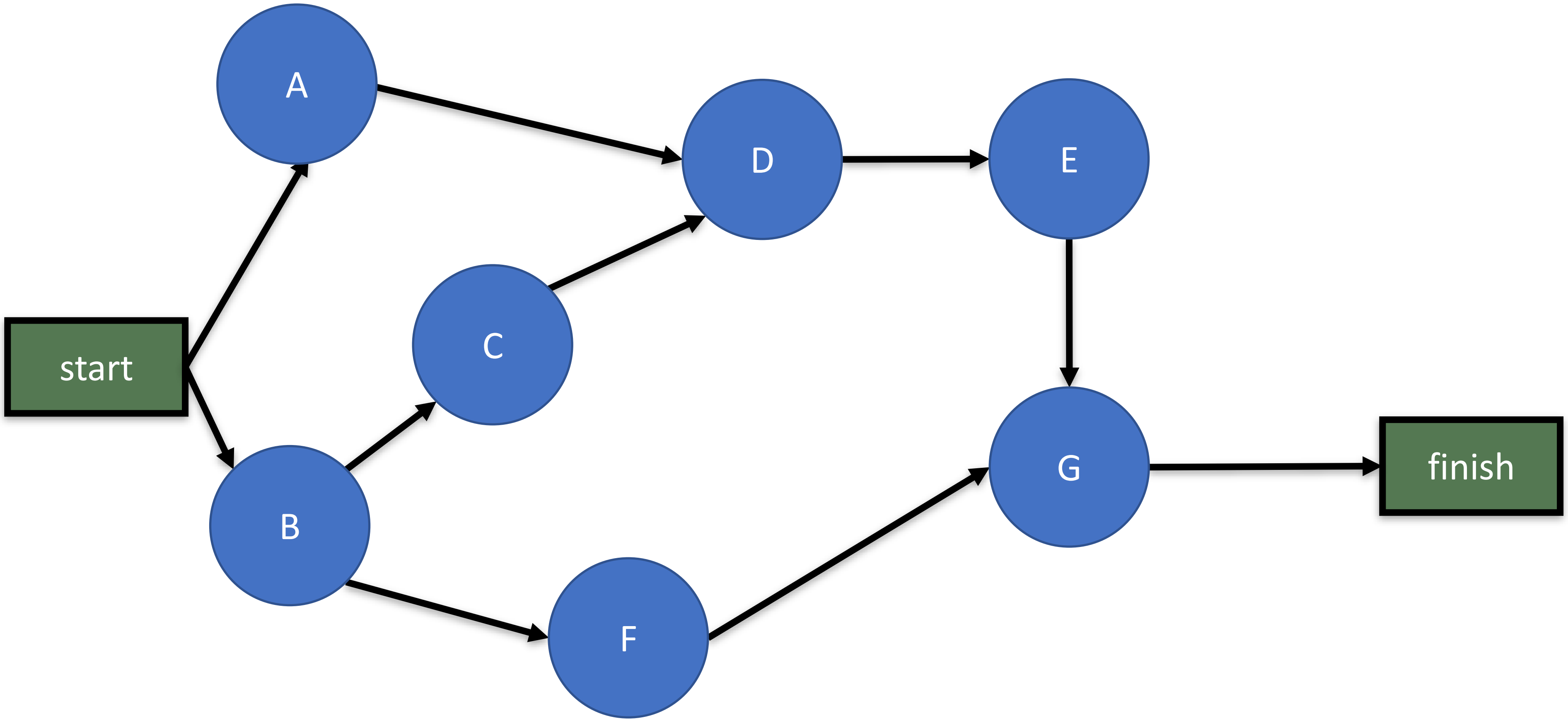
Example: Building a house

Activities in order, Durations, Labels, Dependencies

Project tasks	Durations	Labels	Preds.	Post
Get permits	2	A	--	B
Pour foundations	6	B	--	C, F
Frame walls	5	C	B	D
Install plumbing systems	4	D	A, C	E
Install electrical systems	6	E	D	G
Install landscaping	9	F	B	G
Move in	3	G	E, F	--

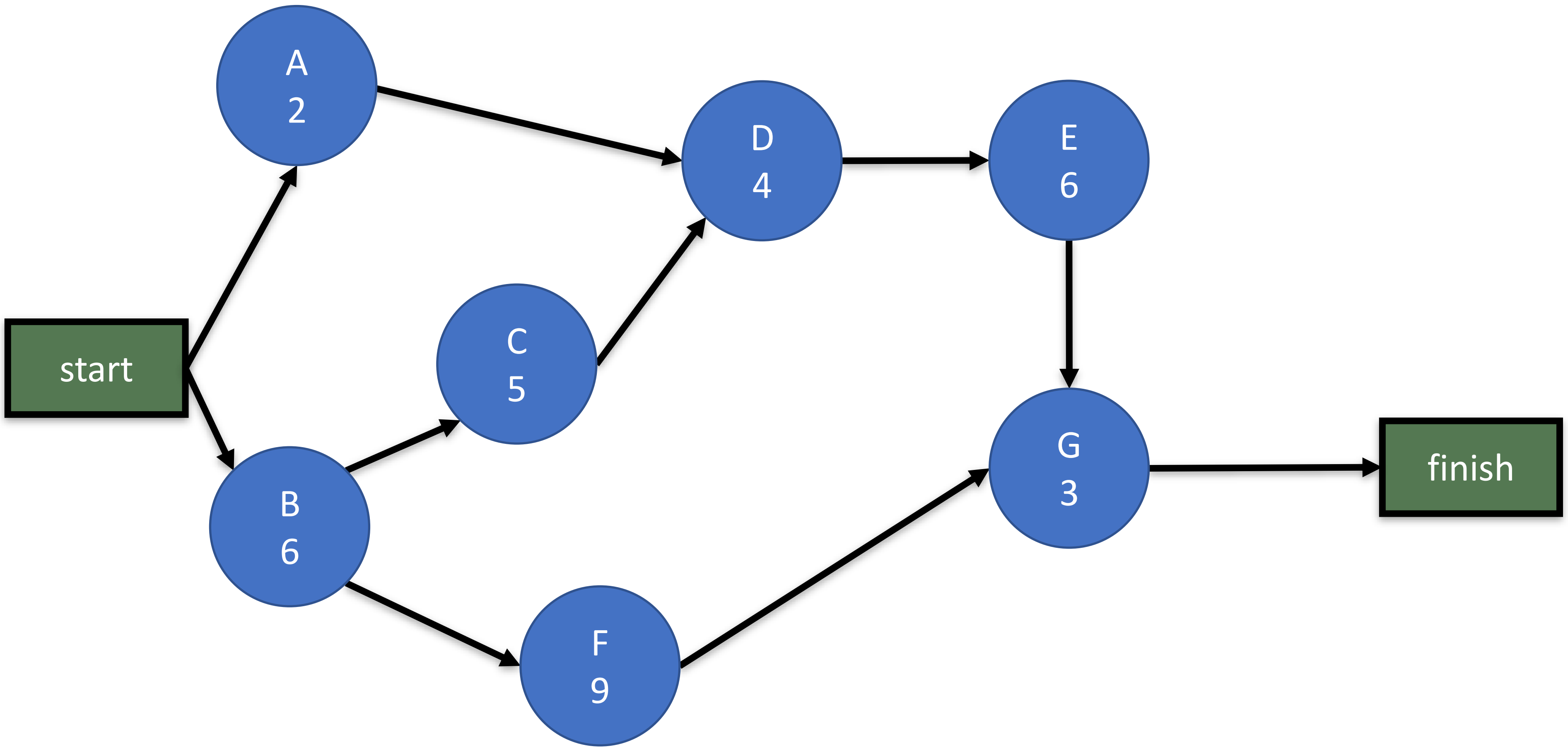
Example: Building a house

Create a precedency diagram



Example: Building a house

Add duration to each node in the diagram



Critical Path Method

- Uses an **Activity Graph** with single time estimate for each activity
- A standard method for managing large construction projects
- On big projects, activity graphs with more than 10,000 activities are common
- Based on the estimated duration, calculate the theoretical Early Start , Early Finish, Late Start and Late Finish for each activity

ES, EF, LF, LS

- **Earliest start date (ES)**: the earliest date that it is possible to start an activity, given that its precedent activities must be completed first
- **Earliest finish date (EF)**: the date that all the activities ending at that node will be completed, assuming that every activity begins at its earliest start date
  - Equal to the earliest start time for the activity plus the time required to complete the activity
- **Latest finish time (LF):** the latest time at which the activity can be completed without delaying the project
- **Latest start time (LS):** equal to the latest finish time minus the time required to complete the activity

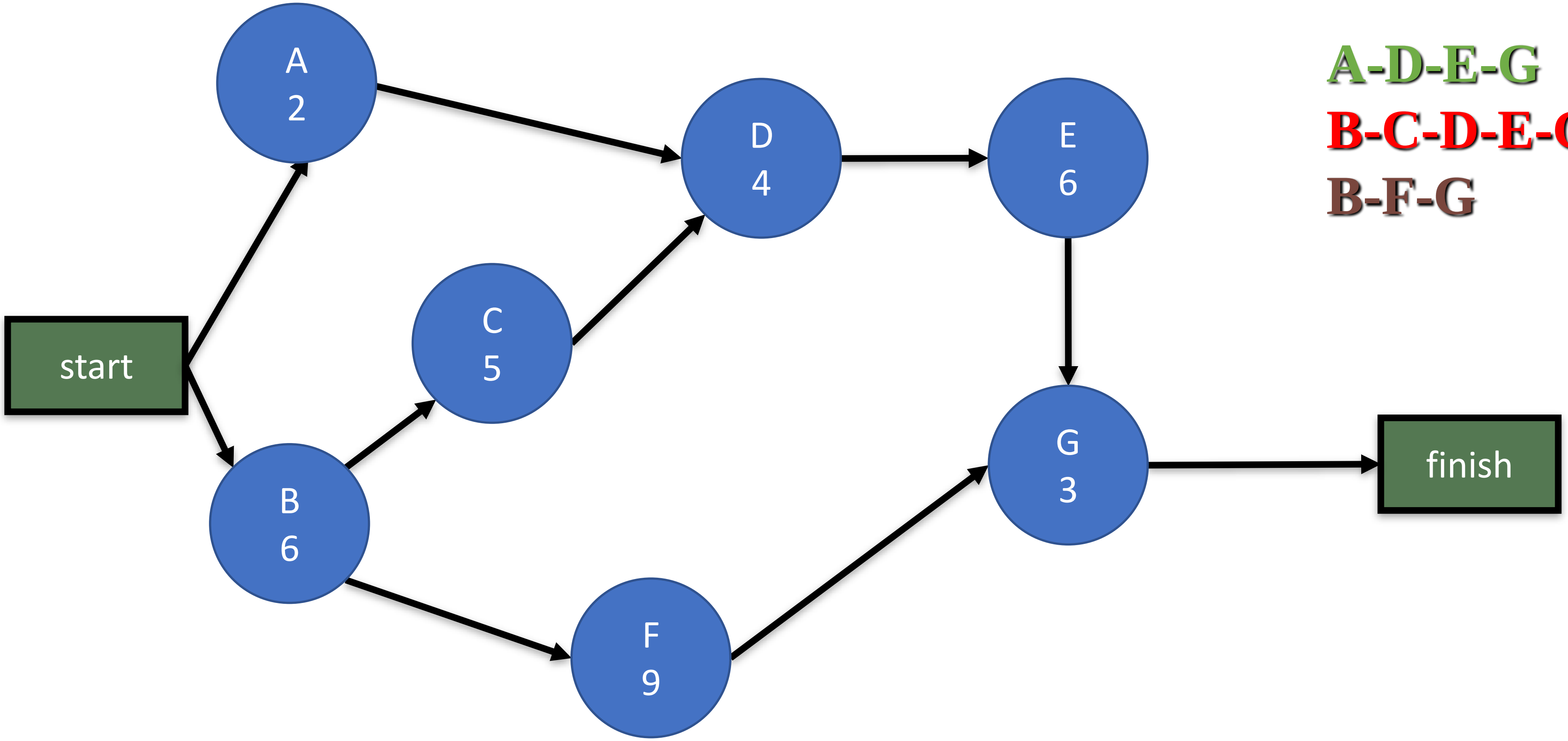
Identify Critical Path

- The critical path is the longest-duration path through the network
- Determining the following four parameters for each activity

- **Slack time** (float time): how much extra time you have available for a particular activity?

Example: Building a house

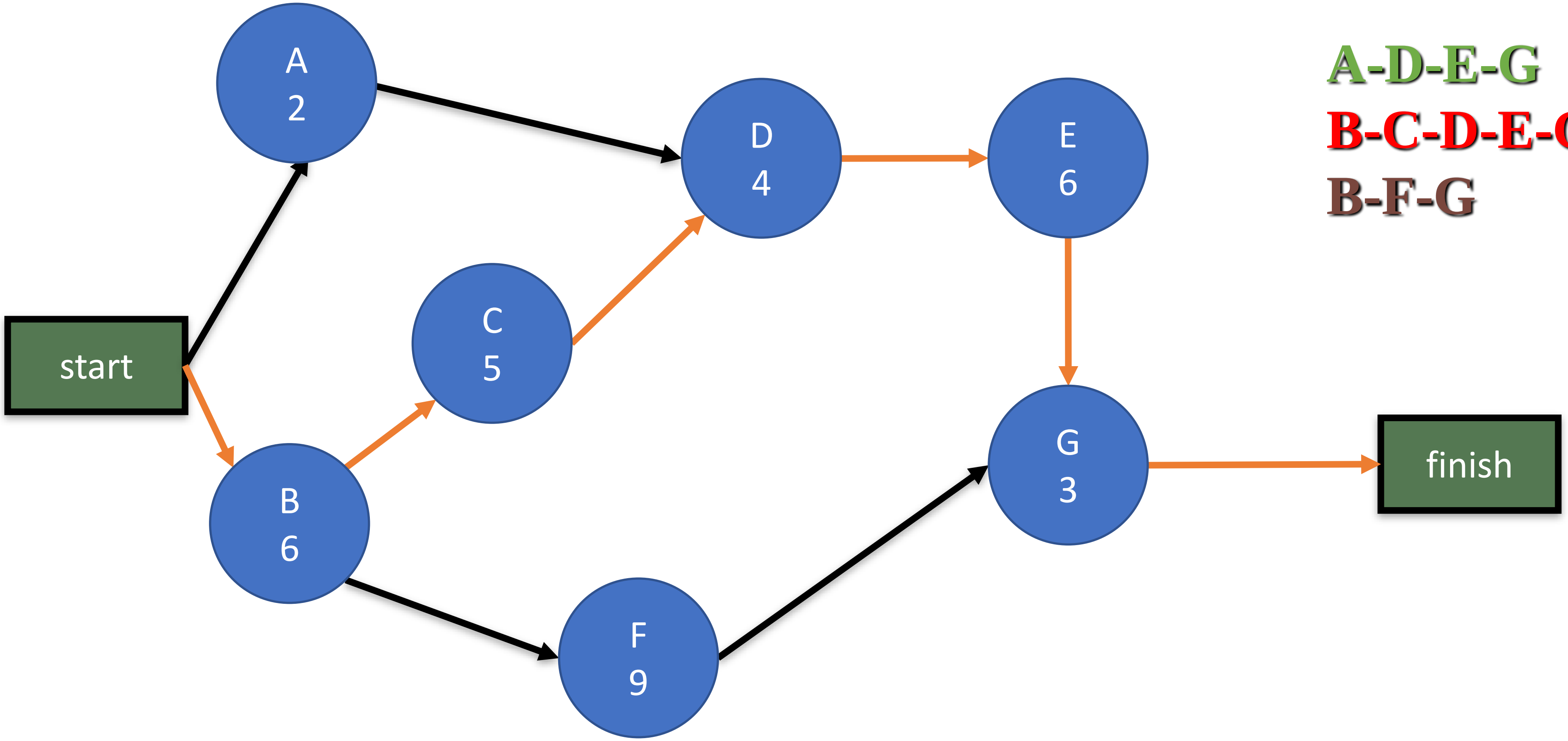
How many paths are there in this network?



A-D-E-G = 15
B-C-D-E-G = 24
B-F-G = 18

Example: Building a house

Which is the longest path?



A-D-E-G = 15

B-C-D-E-G = 24

B-F-G = 18

Calculate Slack time

For each activity, calculate ES, EF, LS, LF and slack time

ES	**Duration**	**EF**
Activity		
LS	**Slack**	**LF**

Calculate Slack time (2)

- The float is how long an activity's duration can extend before it lengthens the project duration

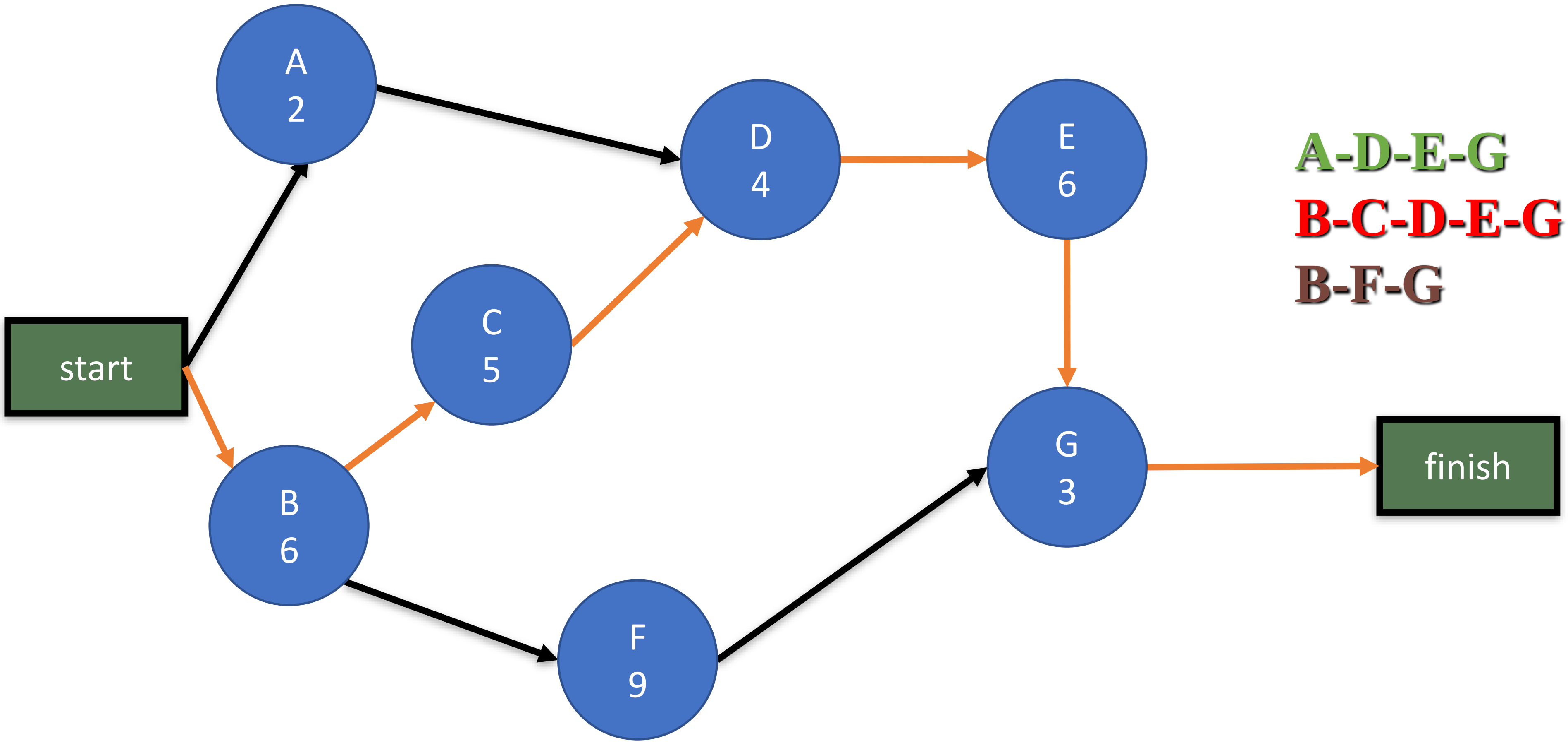
- The float for any activity on the critical path is **zero**

- The float for **non-critical activities** is the critical path duration ***minus*** the duration of the activity's path

- If **an activity** is on **multiple paths**, its float is the one that is **least**

Calculate Slack time (3)

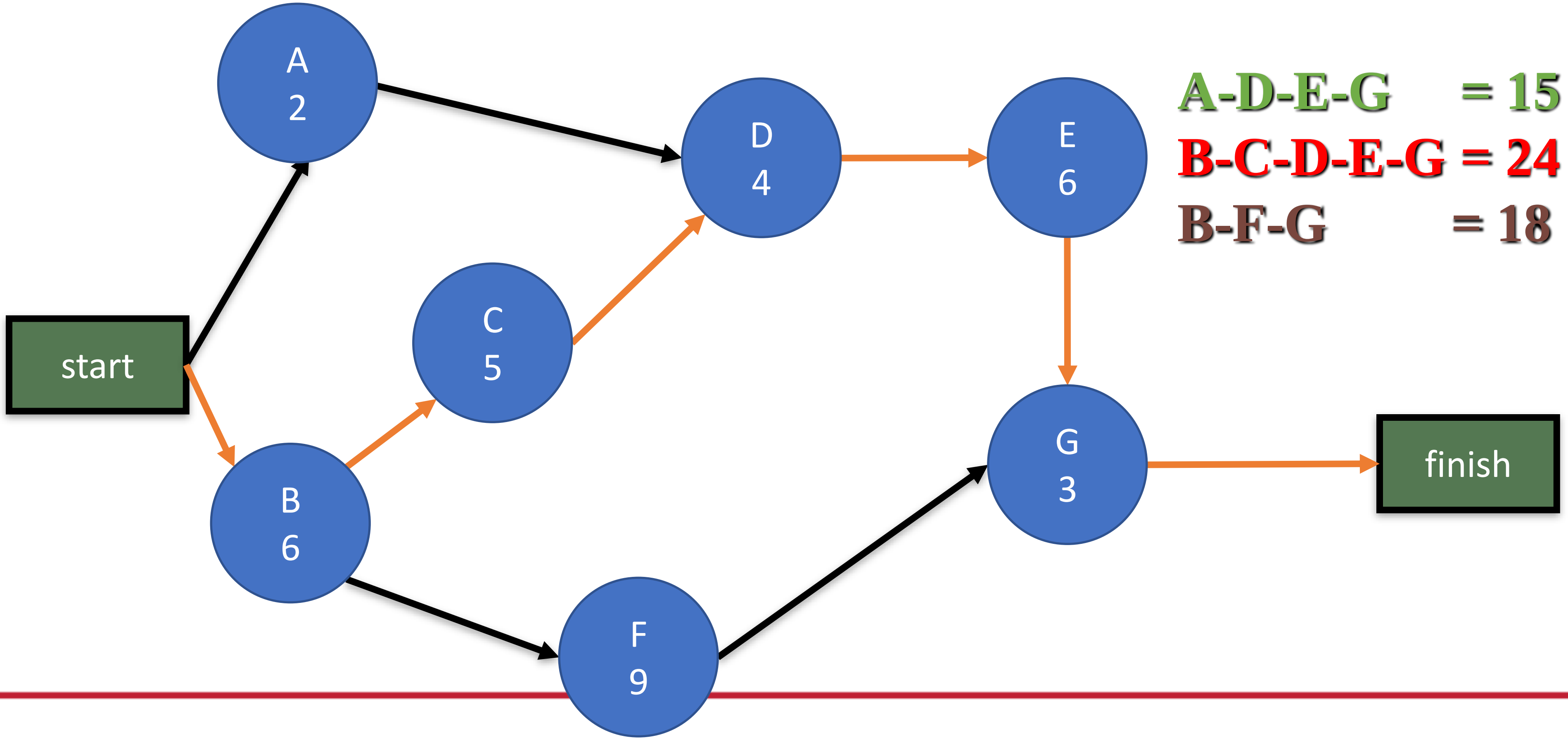
- The critical path has a duration of 24
- The Slack time of activities B, C, D, E, G are all 0.



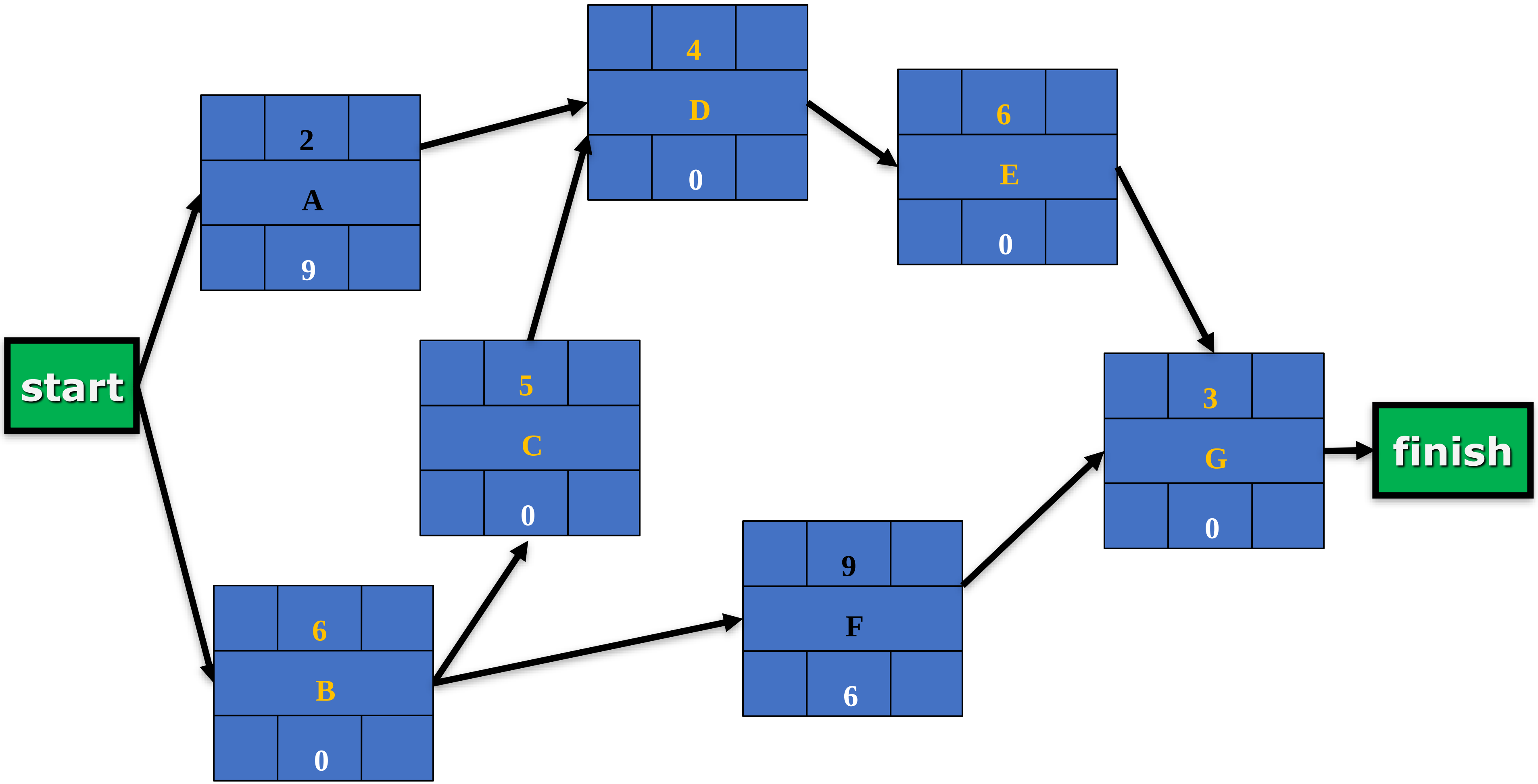
A-D-E-G = 15
B-C-D-E-G = 24
B-F-G = 18

Calculate Slack time (4)

- With path B-F-G has a duration of 18, the Slack time of F (non-critical path activities) is 24–18 = 6
- What's about the activity A?



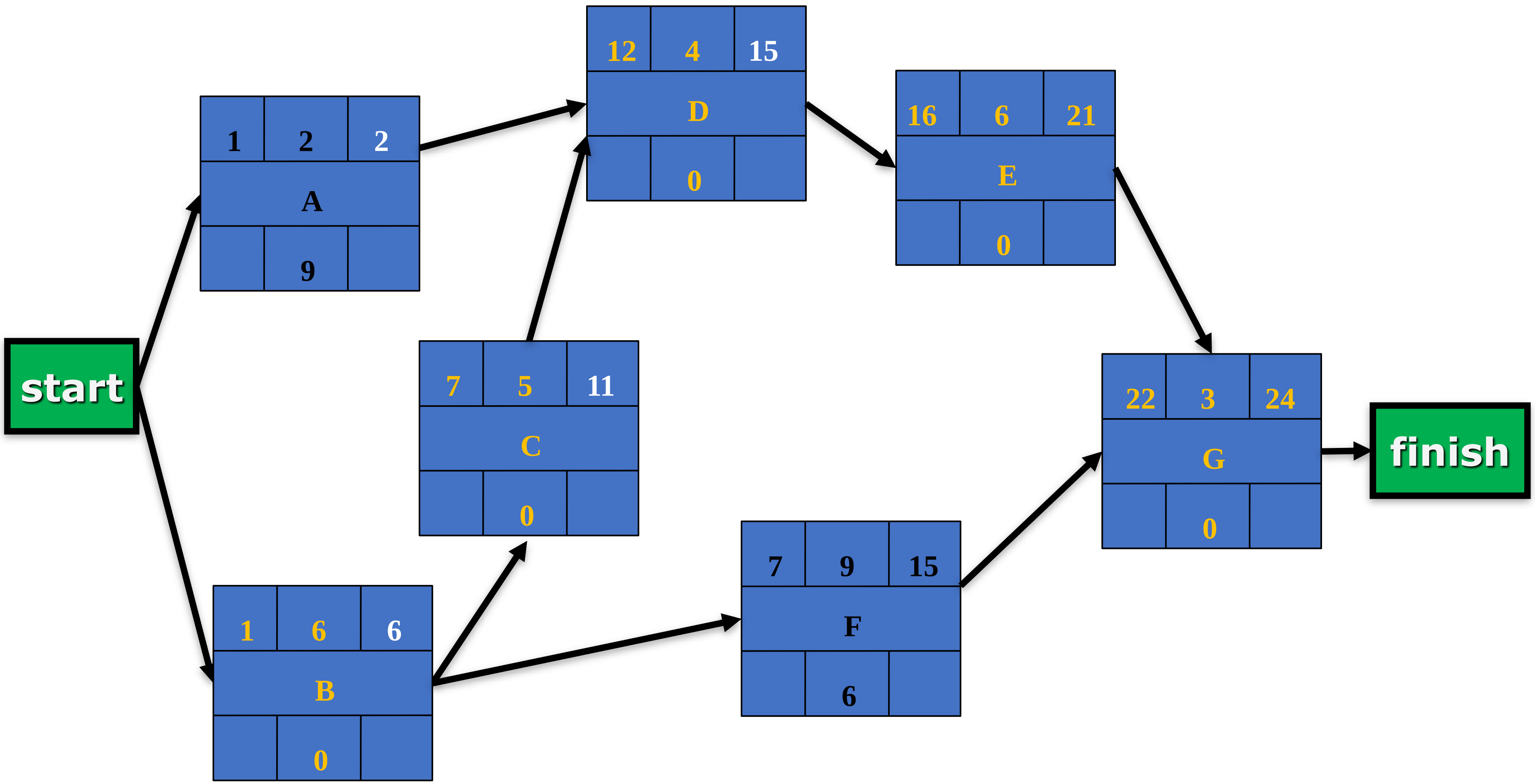
Calculate Slack time (5)



Calculate ES and EF (1)

- ES and EF are calculated by doing a **forward** pass through the diagram
- The **ES** of *activities after the start node* is **1**
- The **EF** of an activity is its **ES** *plus* **its duration** *minus* **1**
- The **ES** is the **EF** of the predecessor activity *plus* **1**
- If there are multiple predecessor activities, use the *greatest EF*

Calculate ES and EF (2)



Calculating LS and LF (1)

- Late start is the latest time that an activity can start
  - If an activity is on a path that's much **shorter** than the critical path, then it can start very **late without delaying** the project

- Late finish is the latest time that an activity can finish
  - If an activity is on a **shorter** path than the critical path and all of the other activities on that path start and finish early, then it can finish very **late without delaying** the project

Calculating LS and LF (2)

- LS and LF are calculated by doing a backward pass through the diagram

- Start with the longest path and work your way from the end node to the start node
  - Do the same thing for the next longest path, and so on
  - Don't recalculate the LS or LF for an activity that's already been calculated on a prior backward pass.

Calculating LS and LF (3)

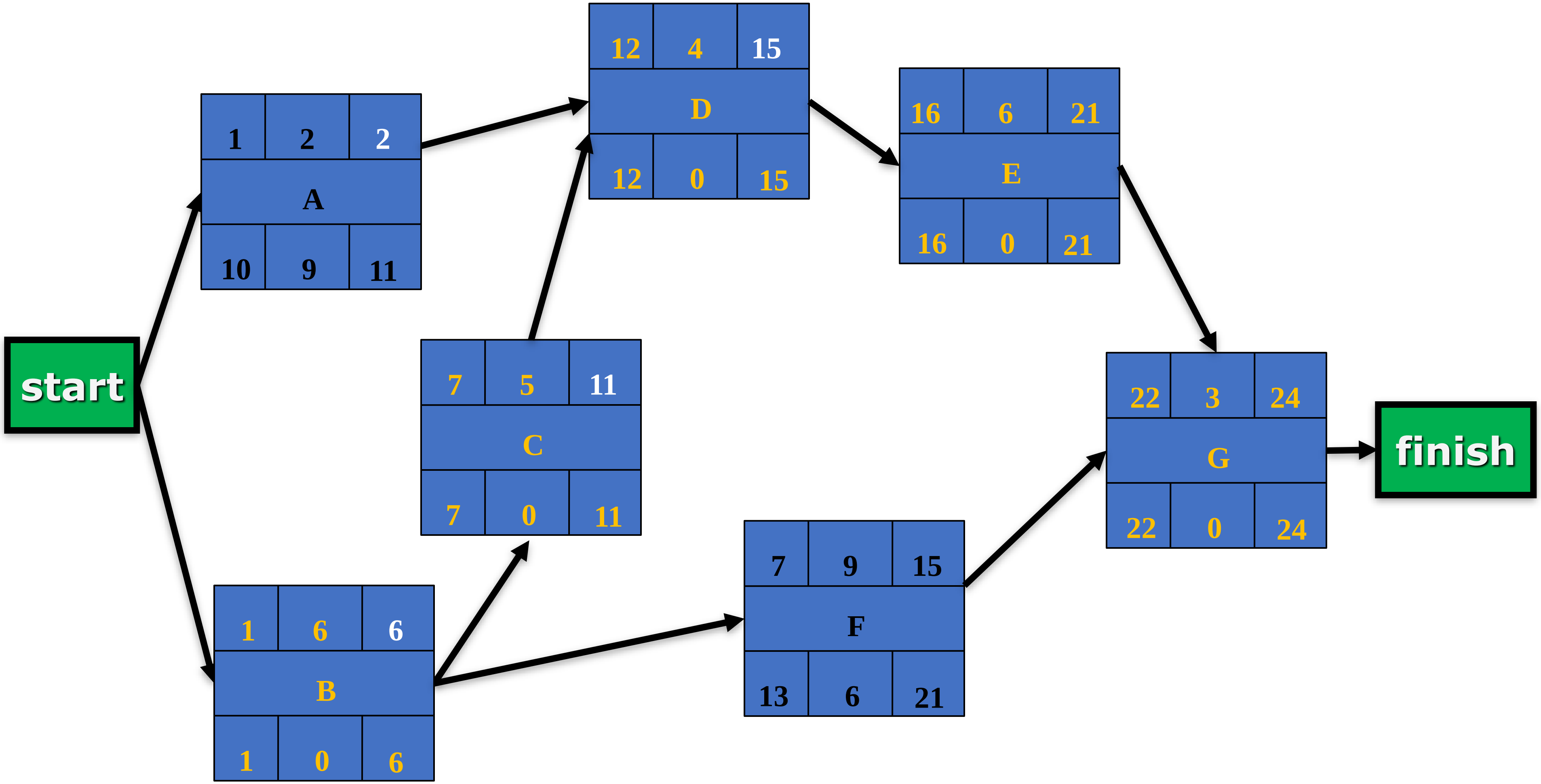
- The **LS** and **LF** of the **last activity** in the **critical path** will be *the same as its ES and EF*

- The **LF** of **non-critical activities** with the end node as their successor will be the **LF** of the last critical path activity

- The **LF** of an activity is the **LS** of its successor *minus* **1**
  - If there are multiple successor activities, use the *least LS*

- The **LS** is the **LF** of the activity *minus* its **duration** *plus* **1**

Calculating LS and LF (4)



Discussion

- What are the critical activities?
- How long will it take to complete this project?
- Can activity D be delayed without delaying the entire project? If so, how many weeks?
- Can activity F be delayed without delaying the entire project? If so, how many weeks
- What is the schedule for activity C?

Software Risks

- Risk always involves two characteristics:
  - Uncertainty —the risk may or may not happen; that is, there are no 100% probable risks.
  - Loss—if the risk becomes a reality, unwanted consequences or losses will occur
- When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk.
- To accomplish this, different categories or types of risks are considered.
  - Project Risks
  - Technical Risks
  - Business Risks
  - Known Risks.
  - Predictable Risks
  - Unpredictable Risks

Reactive Risk Management

- Project team reacts to risks when they occur.
- More commonly, the software team does nothing about risks until something goes wrong.
- Then, the team involved into action in an attempt to correct the problem rapidly. This is often called a *fire fighting mode*.
- When this fails, "crisis management" takes over and the project is in real jeopardy.

Proactive Risk Management

- A proactive strategy begins long before technical work is initiated.
- Potential risks are identified, their probability and impact are assessed, and they are ranked by importance.
- Then, the software team establishes a plan for managing risk.
- The primary objective is to avoid risk, but because not all risks can be avoided, the team works to develop a contingency plan that will enable it to respond in a controlled and effective manner.

Developing Risk Table

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
• • •				

Impact values:
1—catastrophic
2—critical
3—marginal
4—negligible

Determine the overall consequences of a risk

- Determine the average probability of occurrence value for each risk component.
- Determine the impact for each component based on the criteria.
- Complete the risk table and analyze the results as described

Now measure, Risk exposure (RE).

$$RE = P \times C$$

P is the probability of occurrence for a risk and *C* is the cost to the project.

Example

The software team defines a project risk in the following manner

- **Risk Identification** - Only 70 percent of the software components scheduled for reuse and remaining functionality will have to be custom developed.
- **Risk probability.** 80% (likely).
- **Risk Impact** – Assume total no. of component is 60. If only 70 percent can be used, 18 components would have to be developed from scratch.
- Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is $14.00,
- the overall cost (impact) to develop the components would be

  18 x 100 x 14 = $25,200.
- **Risk exposure.** *RE* = 0.80 x 25,200 ~ $20,200.

Risk Mitigation, Monitoring, and Management

- Risk analysis goal - to assist the project team in developing a strategy for dealing with risk. An effective strategy must consider three issues:
  - Risk avoidance or mitigation.
  - Risk monitoring
  - Risk management and contingency planning
- Proactive approach to risk, avoidance is always the best strategy. This is achieved by developing a plan for risk mitigation.
- For example, assume that high staff turnover (i.e. revenue) is noted as a project risk.

RMMM Plan

Risk information sheet

Risk ID: P02-4-32	Date: 5/9/02	Prob: 80%	Impact: high

Description:
Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.

Refinement/context:
Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards.
Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.
Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.

Mitigation/monitoring:
1. Contact third party to determine conformance with design standards.
2. Press for interface standards completion; consider component structure when deciding on interface protocol.
3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.

Management/contingency plan/trigger:
RE computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly.
Trigger: Mitigation steps unproductive as of 7/1/02

Current status:
5/12/02: Mitigation steps initiated.

Originator: D. Gagne	Assigned: B. Laster

7. Project Management

(end of lecture)

ONE LOVE. ONE FUTURE.