

 POLITECNICO DI MILANO

Dipartimento di
Elettronica e Informazione

Planning and Managing Software Projects 2012-13
Class 9

Scheduling

Fundamentals, Techniques, Optimization

Emanuele Della Valle, Lecturer: Dario Cerizza
<http://emanueledellavalle.org>

- This slides are partially based on CEFRIEL' s slides for PMI Certification and largely based on Prof. John Musser class notes on “Principles of Software Project Management”
- Original slides are available at <http://www.projectreference.com/>
- Reuse and republish permission was granted

- **Class 7 and 8 Review**
- Scheduling Fundamentals
- Scheduling Techniques
 - Network Diagrams
 - Bar Charts
- Schedule Optimization Techniques
- Mythical Man-Month

- WBS
- Estimation

- Types: Process, Product, Hybrid
- Formats: Outline or graphical organizational chart
- High-level WBS does not show durations and dependencies
- WBS becomes input to many things, esp. schedule
- What hurts most is what's missing → 100% Rule:
 - the sum of the work on the children must be equal to 100% of the work referred by the parent
 - All work inside the scope of the project is in the WBS
 - No work outside the scope of the project is in the WBS

- Estimation is the process of determining the effort and the duration of activities
- Setting realistic expectations is the single most important task of a project
- “Unrealistic expectations based on inaccurate estimates are the single largest cause of software failure.”
 - Futrell, Shafer, Shafer, “Quality Software Project Management”

- Use multiple methods if possible
 - This reduces your risk
 - If using “experts”, use two
- History is your best ally
 - Especially when using Function Points, LOC (Lines of Code), ...
- Get buy-in
 - Involve who will do the work in the estimation process
- Remember: estimation is an iterative process!
 - Esteems must be updated during project execution
 - First esteems are usually very wrong w.r.t real effort spent during the project execution
 - Last esteems are usually more accurate
- Know your “presentation” techniques
 - Provide esteems ranges where possible
 - Provide different esteems with different probabilities

- Bottom-up
 - More work to create but more accurate
 - Often with Expert Judgment at the task level (buy-in)
- Top-down
 - Used in the earliest phases
 - Usually as is the case with Analogy or Expert Judgment
- Analogy
 - Comparison with previous project: formal or informal
- Expert Judgment
 - Via staff members who will do the work
 - Most common technique along with analogy
 - Best if multiple ‘experts’ are consulted

- Parametric Methods
 - Know the trade-offs of: LOC & Function Points

- Function Points
 - Benefit: relatively independent of the technology used to develop the system
 - We will re-visit this briefly later in semester (when discussing “software metrics”)

- **Scheduling Fundamentals**
- Scheduling Techniques
 - Network Diagrams
 - Bar Charts
- Schedule Optimization Techniques
- Mythical Man-Month

- Initial Planning:
 - It's needed for the SOW and the Project Charter
 - It answers to the question: What/How
 - WBS (1st pass)
 - Should consider activities for other project management documents
 - Software Development Plan, Risk Mgmt., Cfg. Mgmt.
- Estimating
 - It answers to the question: How much/How long
 - Size (quantity/complexity) and Effort (duration)
 - It's an iterative process
- Scheduling
 - It answers to the question: In which order
 - Precedence, concurrences, lag & lead times, slack & float, ...
 - It's an iterative process

- Once tasks (from the WBS) and effort/duration (from estimation) are known: then “scheduling”
- Scheduling is the definition of the start and the end time of each activity/task
 - Requires the definition of **dependencies** among tasks and the **assignment of resources**
- The result of scheduling is a plan that defines:
 - **Who**
 - does **What**
 - by **When**

- Scheduling aims to find a trade-off among six objectives:
 - Primary objectives
 1. Best time
 2. Least cost
 3. Least risk
 - Secondary objectives
 4. Propose and evaluate different schedule alternatives
 5. Make an effective use of resources
 6. Reduce communication overhead among resources

4 reasons for dependencies among tasks

1. Mandatory Dependencies

- These are “Hard logic” dependencies
- Nature of the work dictates an ordering
- Ex: UI design precedes UI implementation
- Ex: Coding has to precede testing (*)

2. Discretionary Dependencies

- “Soft logic” dependencies
 - Determined by the project management team
 - Process-driven
 - Ex: Discretionary order of creating certain modules
 - Ex: First Front-end, then Back-End (because the opposite order is also meaningful)
- (*) NOTE: substantial process innovation often take place when “hard logic” dependencies are shown to be wrong
 - Ex: Test first approaches

3. External Dependencies

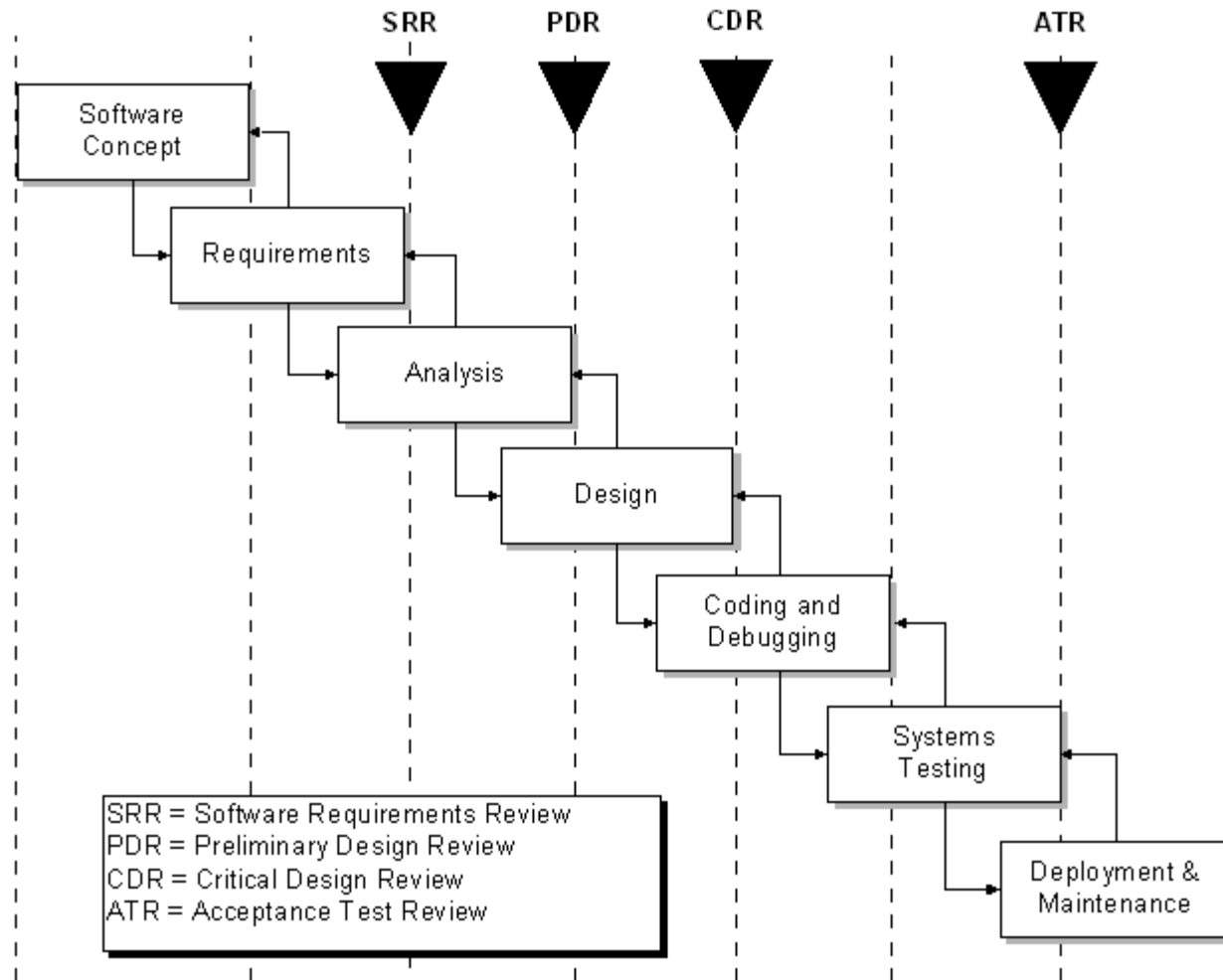
- Outside of the project itself
- Ex: Release of 3rd party product; contract signoff
- Ex: stakeholders, suppliers, year end

4. Resource Dependencies

- Two task rely on the same resource
- Ex: You have only one DBA but multiple DB tasks
- Ex: You have only one server but different software versions needs multiple installations

- Identify crucial points in your schedule

- Have a duration of zero
- Typically shown as inverted triangle or a diamond
- Often used at “review” or “delivery” times
 - Or at end or beginning of phases
 - Ex: Software Requirements Review (SRR)
 - Ex: Delivery of working component
- Can be tied to contract terms
 - Ex: User Sign-off



- Scheduling Fundamentals
- **Scheduling Techniques**
 - Network Diagrams
 - Bar Charts
- Schedule Optimization Techniques
- Mythical Man-Month

- **Network Diagrams**
 - CPM
 - PERT

- **Bar Charts**
 - Milestone Chart
 - Gantt Chart

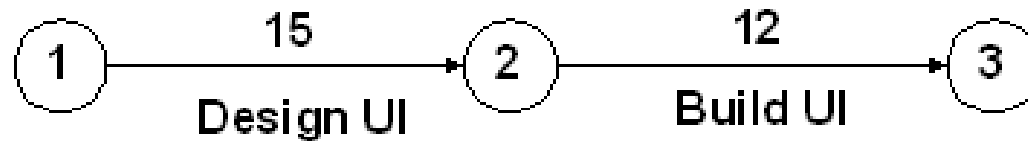
- Developed in the 1950's
- A graphical representation of the tasks necessary to complete a project
- Clearly visualizes the flow of tasks & relationships

- CPM
 - Critical Path Method
- PERT
 - Program Evaluation and Review Technique
- Sometimes treated synonymously
- All are models using network diagrams

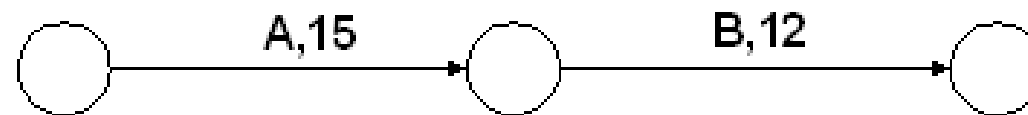
Two classic formats

- Two classic formats
 - AOA: Activity on Arrow
 - AON: Activity on Node
- Each activity labeled with
 - Identifier (usually a letter/code)
 - Duration (in standard unit like days)
- There are other variations of labeling
- There is one start & one end event
- Time goes from left to right

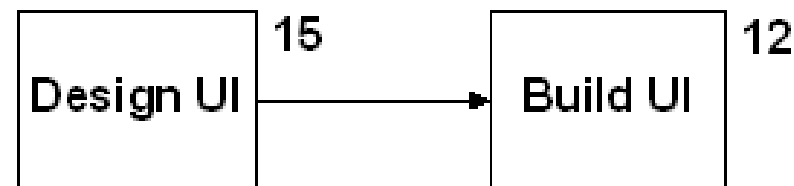
Activity on Arrow (AOA)



or



Activity on Node (AON)

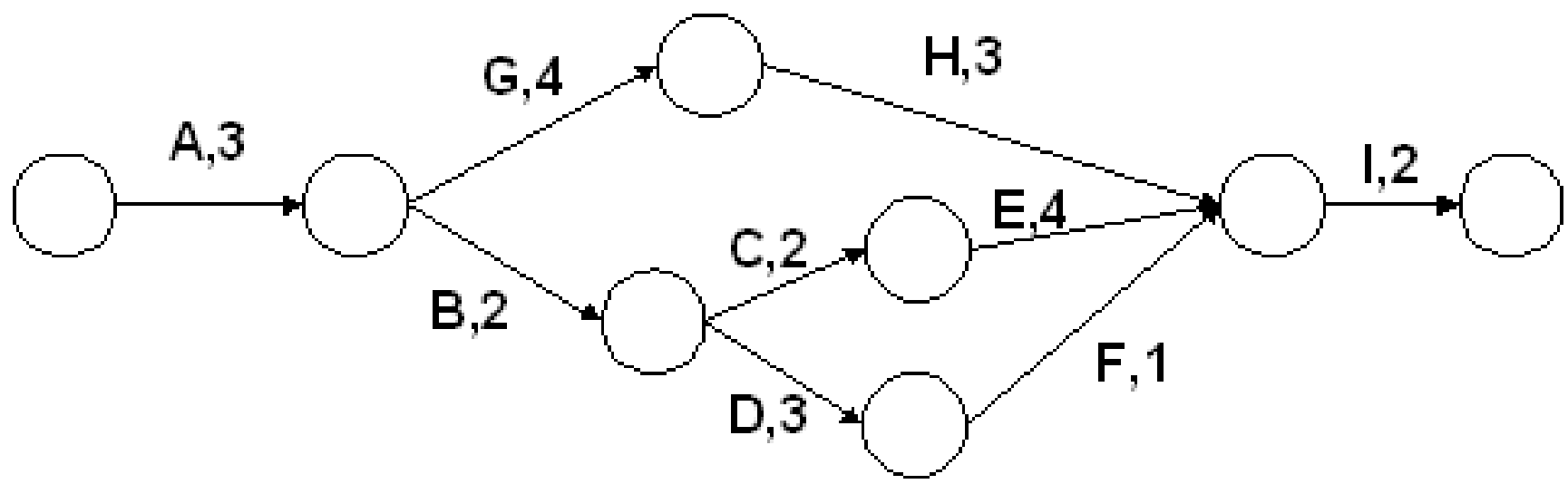


or

| | | |
|-------------|----------|--------------|
| Early Start | Duration | Early Finish |
| Task Name | | |
| Late Start | Slack | Late Finish |

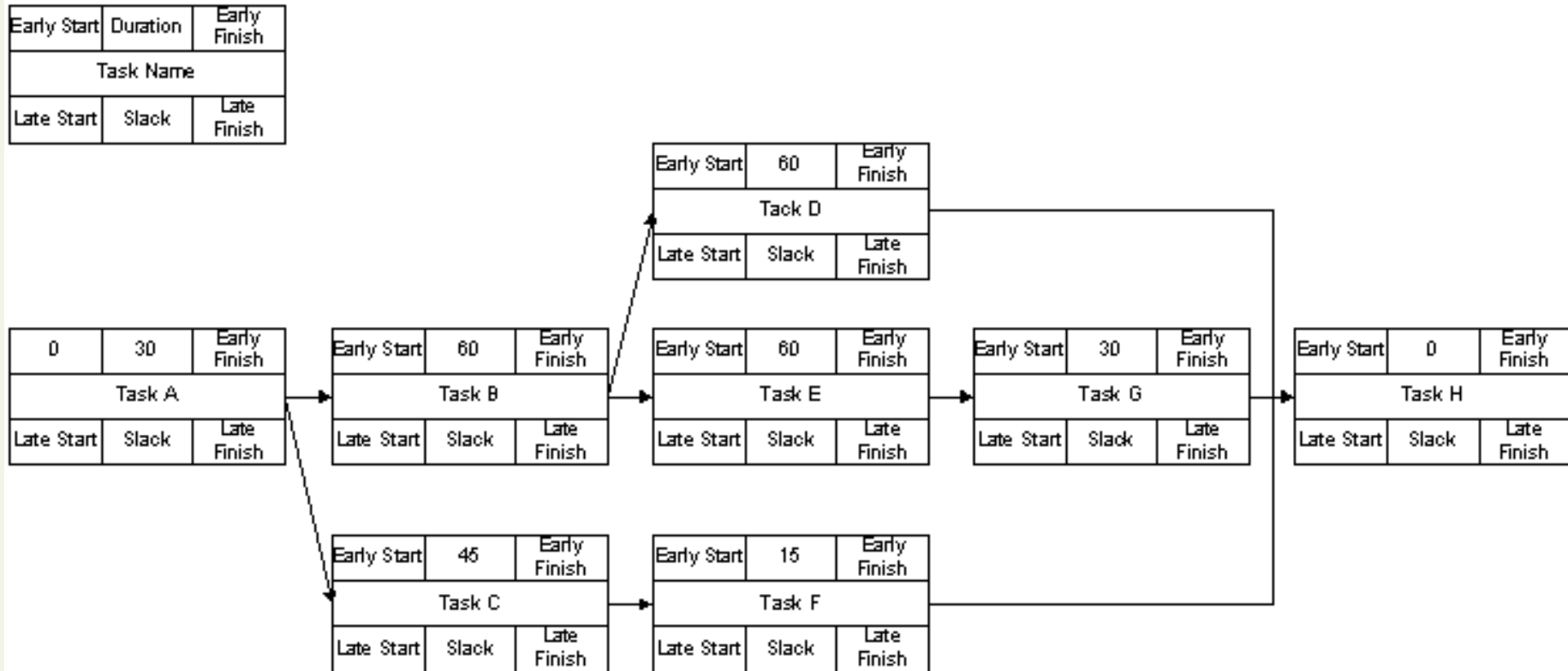
- AOA
 - Activities on Arrows
 - Circles representing Events
 - Such as 'start' or 'end' of a given task
 - Lines representing Tasks
 - Thing being done 'Build UI'
 - a.k.a. Arrow Diagramming Method (ADM)

- AON
 - Activities on Nodes
 - Nodes can be circles or rectangles (usually latter)
 - Task information written on node
 - Arrows are dependencies between tasks
 - a.k.a. Precedence Diagramming Method (PDM)



Network Diagrams

AON Example



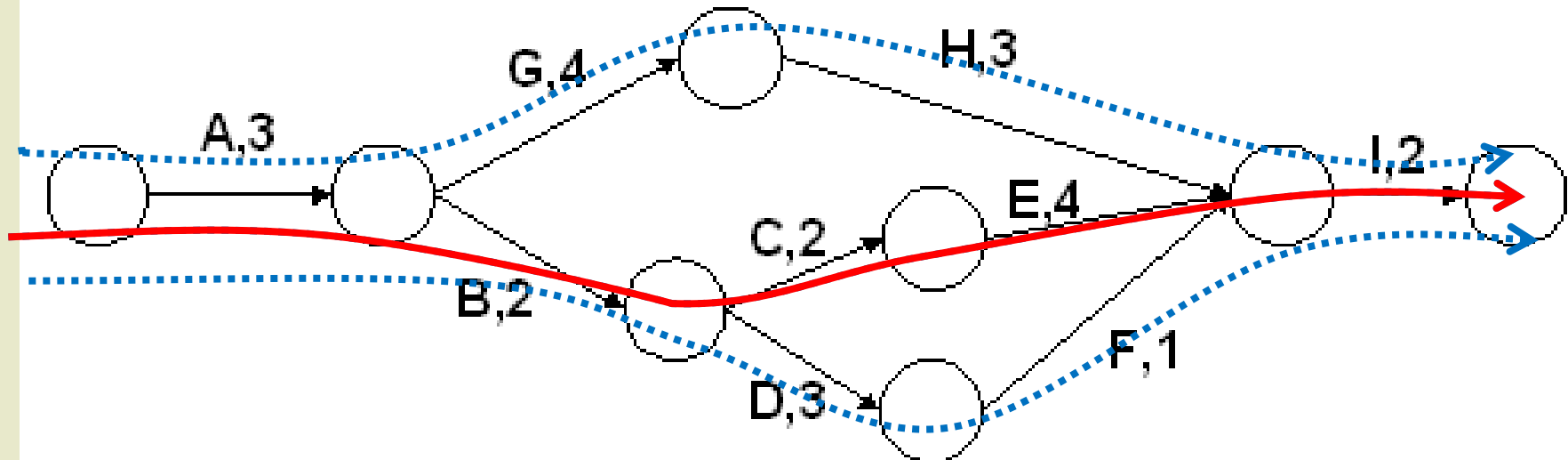
This type of diagrams are often confused as PERT diagrams.
We'll see PERT later

- A Critical Path is a specific set of sequential tasks upon which the project completion date depends

- Tasks on the critical path cannot be delayed without delaying the project completion day
 - If a task on the critical path is delayed by 1 day, then the project completion date is delayed (at least) by 1 day

- All projects have at least one Critical Path
 - Critical Paths are the paths with duration = total project duration

- How many paths are here?
- Which one is the Critical Path?

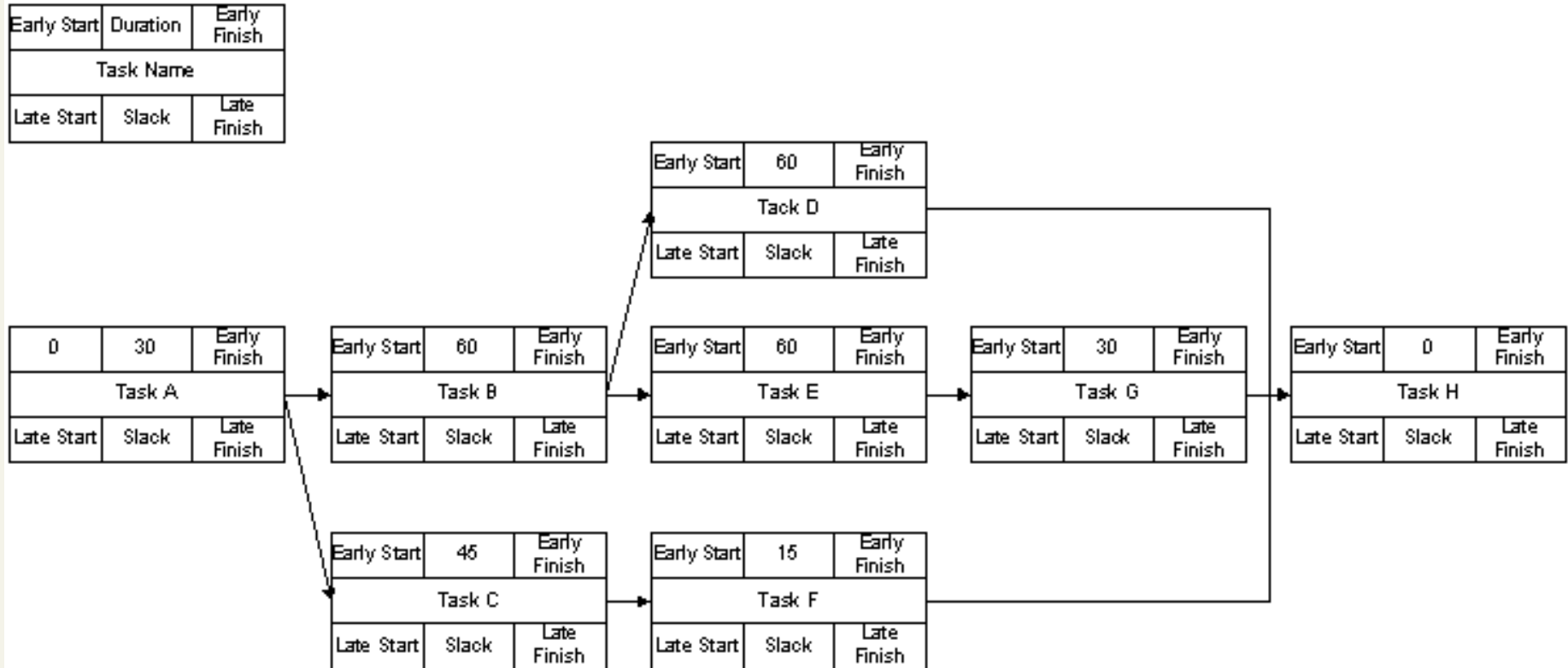


- Delaying tasks on the critical path directly lengthens the schedule
- Accelerating tasks on the critical path does NOT directly shorten the schedule
 - Critical Path may change to another as you shorten the current critical path
- Delaying tasks NOT on the critical path does NOT directly lengthen the schedule
 - Tasks out of the critical path have a little amount of time (named **slack**) that let them to be delayed without impacting on the project completion date
- Accelerating tasks NOT on the critical path does NOT change the project completion date
 - Their slack increases

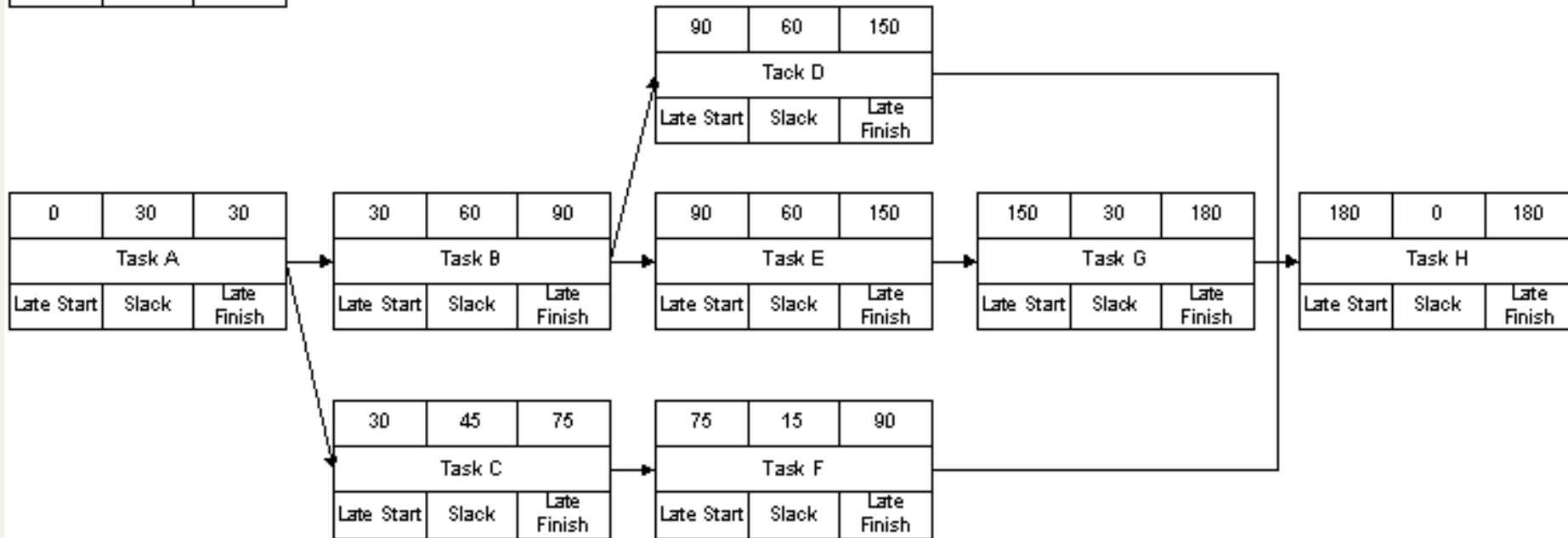
- The process for determining and optimizing the critical path
- Should be done in conjunction with the project manager & the functional manager
- Based upon a **2-passes approach**
 - **Forward Pass** and **Backward Pass**
- As result of the 2-passes, the critical path becomes evident

(This exercise is part of course exams!)

- Used to determine early start (ES) and early finish (EF) times for each task
- Work from left to right
- Adding times to each node and each path
- Rule: when several tasks converge, the ES for the next task is the **largest** of preceding EF times

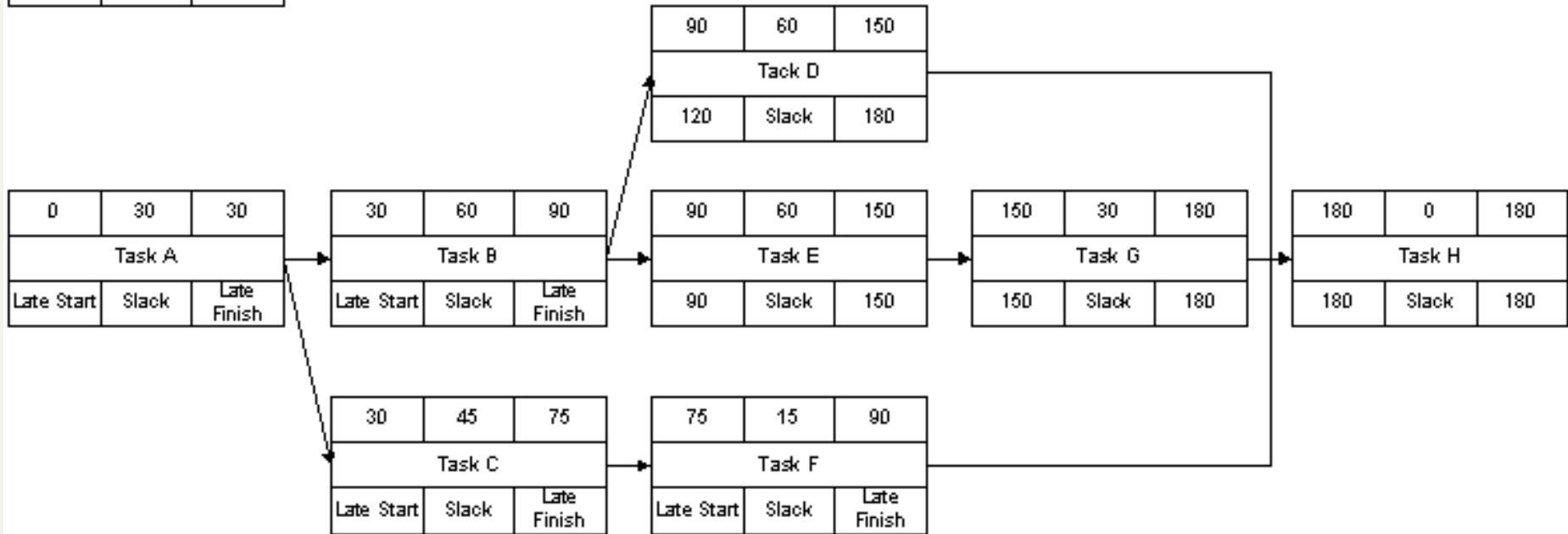


| | | |
|-------------|----------|--------------|
| Early Start | Duration | Early Finish |
| Task Name | | |
| Late Start | Slack | Late Finish |

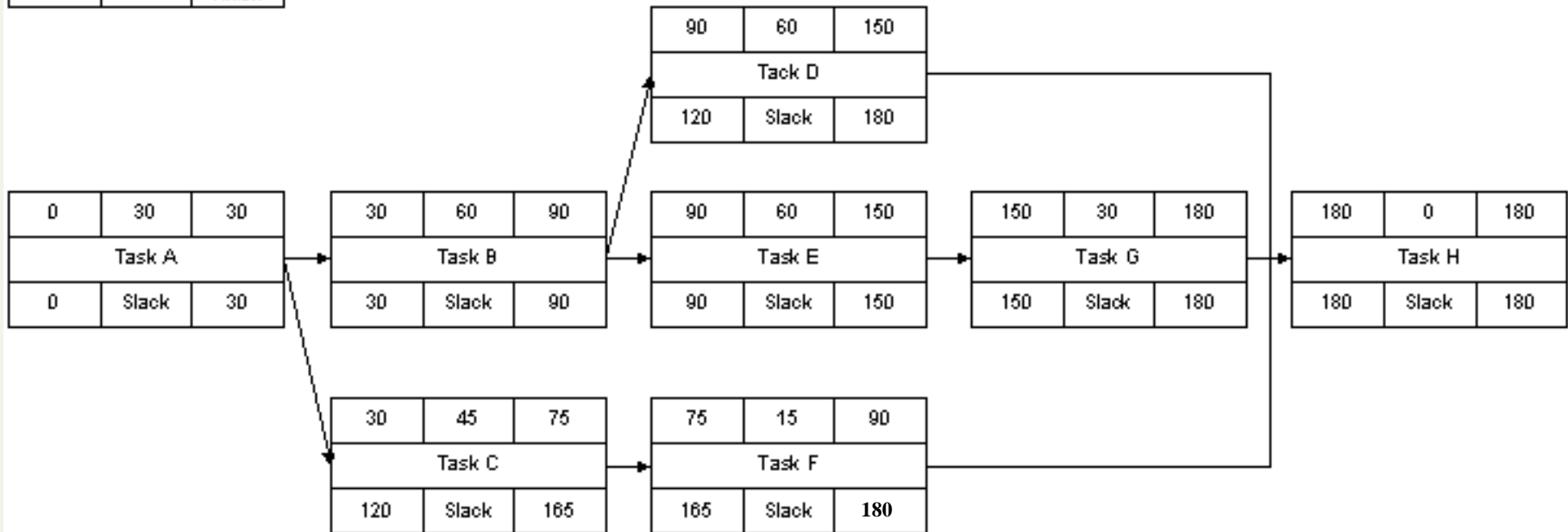


- Used to determine the late finish (LF) and late start (LS) times
- Start at the end node and move backward left
- Subtract duration from connecting node's earliest start time
- Rule: when several tasks converge, the last finish for the previous task is the **smallest** of following last start times

| | | |
|-------------|----------|--------------|
| Early Start | Duration | Early Finish |
| Task Name | | |
| Late Start | Slack | Late Finish |



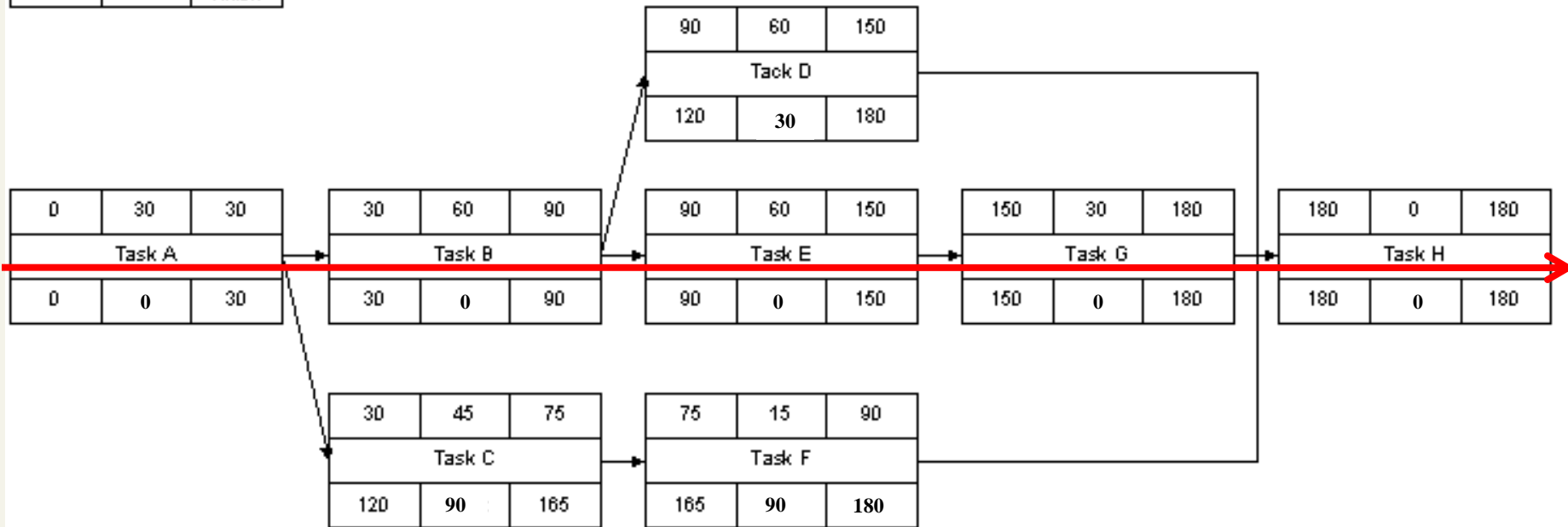
| | | |
|-------------|----------|--------------|
| Early Start | Duration | Early Finish |
| Task Name | | |
| Late Start | Slack | Late Finish |

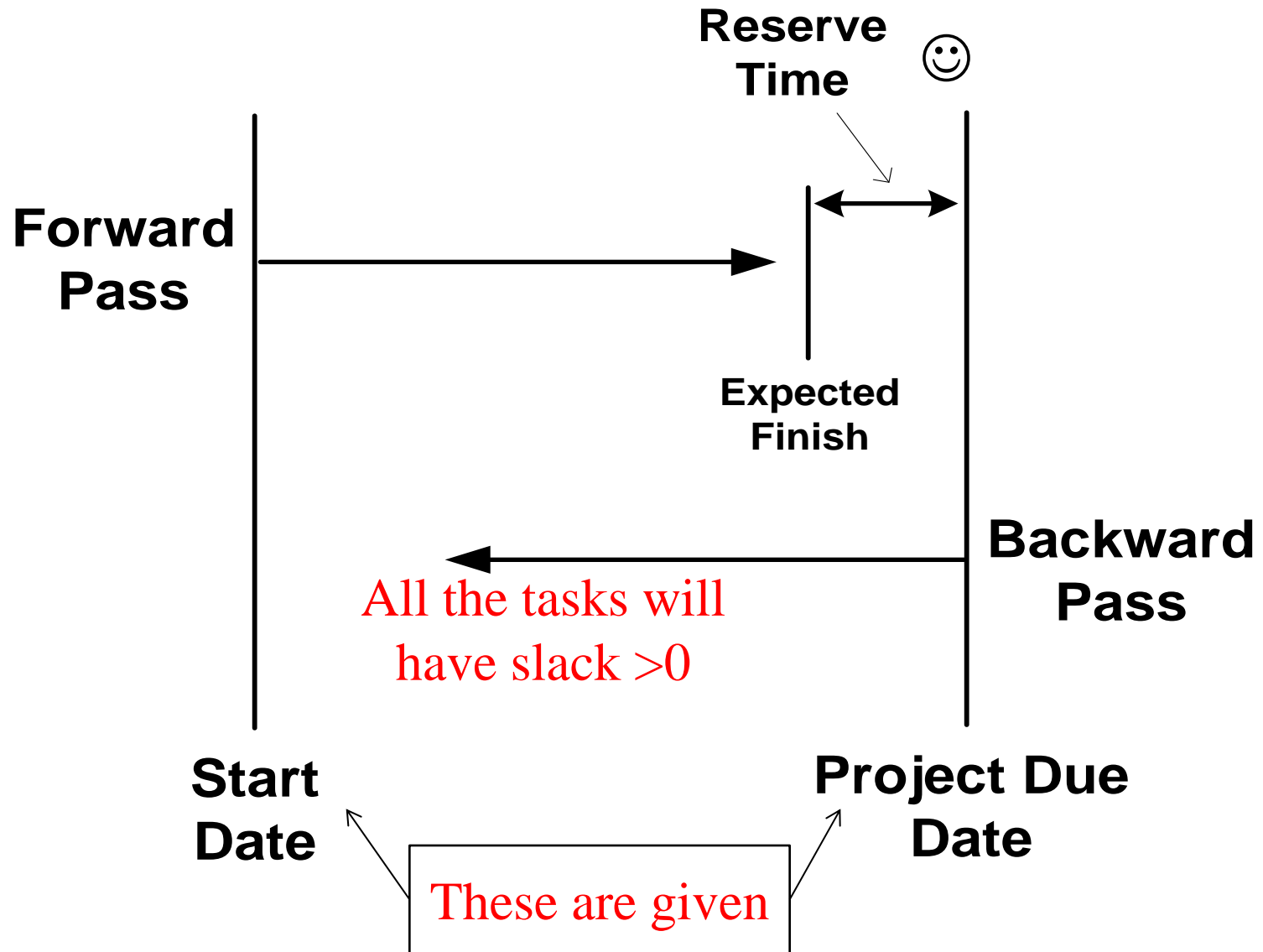


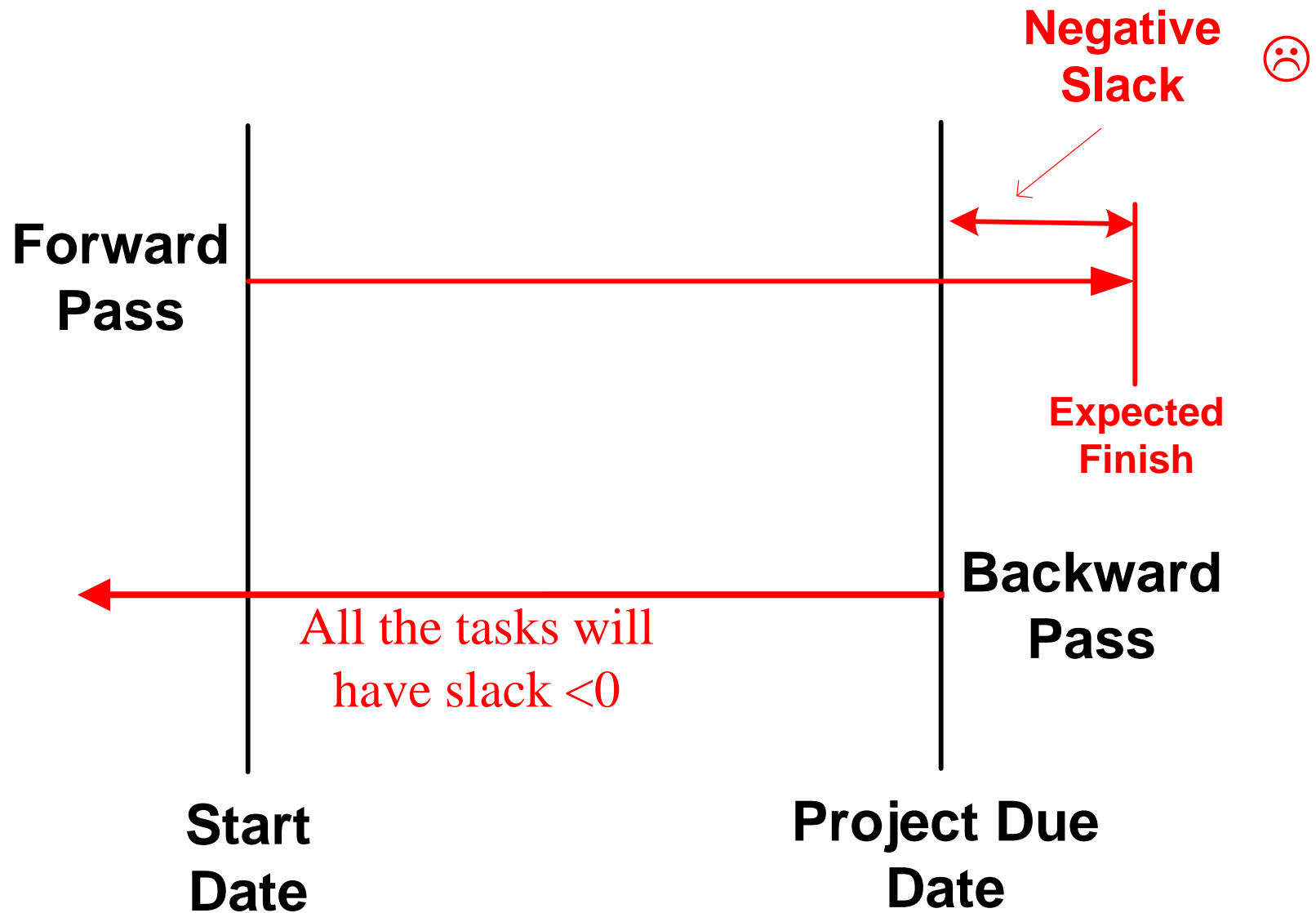
Slack = Late Finish - Early Finish = Late Start - Early Start

Critical Path: all tasks with slack = 0

| | | |
|-------------|----------|--------------|
| Early Start | Duration | Early Finish |
| Task Name | | |
| Late Start | Slack | Late Finish |







- Advantages
 - Show precedence well
 - Reveal interdependencies not shown in other techniques
 - Ability to calculate critical path
 - Ability to perform “what if” exercises

- Disadvantages
 - Default model assumes resources are unlimited
 - You need to incorporate this yourself (Resource Dependencies) when determining the “real” Critical Path
 - Difficult to draw and to read on large projects

- **Program Evaluation and Review Technique**
- Based on idea that estimates are uncertain
 - Therefore uses duration ranges
 - And the probability of falling to a given range
- First is done on each task, then at project level

For each task:

1) Start with 3 estimates for each task

- Optimistic
 - Would likely occur 1 time in 20
- Most likely
 - Modal value of the distribution
- Pessimistic
 - Would be exceeded only one time in 20

2) Calculate the expected time of each task:

$$t_e = \frac{a + 4m + b}{6}$$

where

t_e = expected time

a = optimistic time estimate

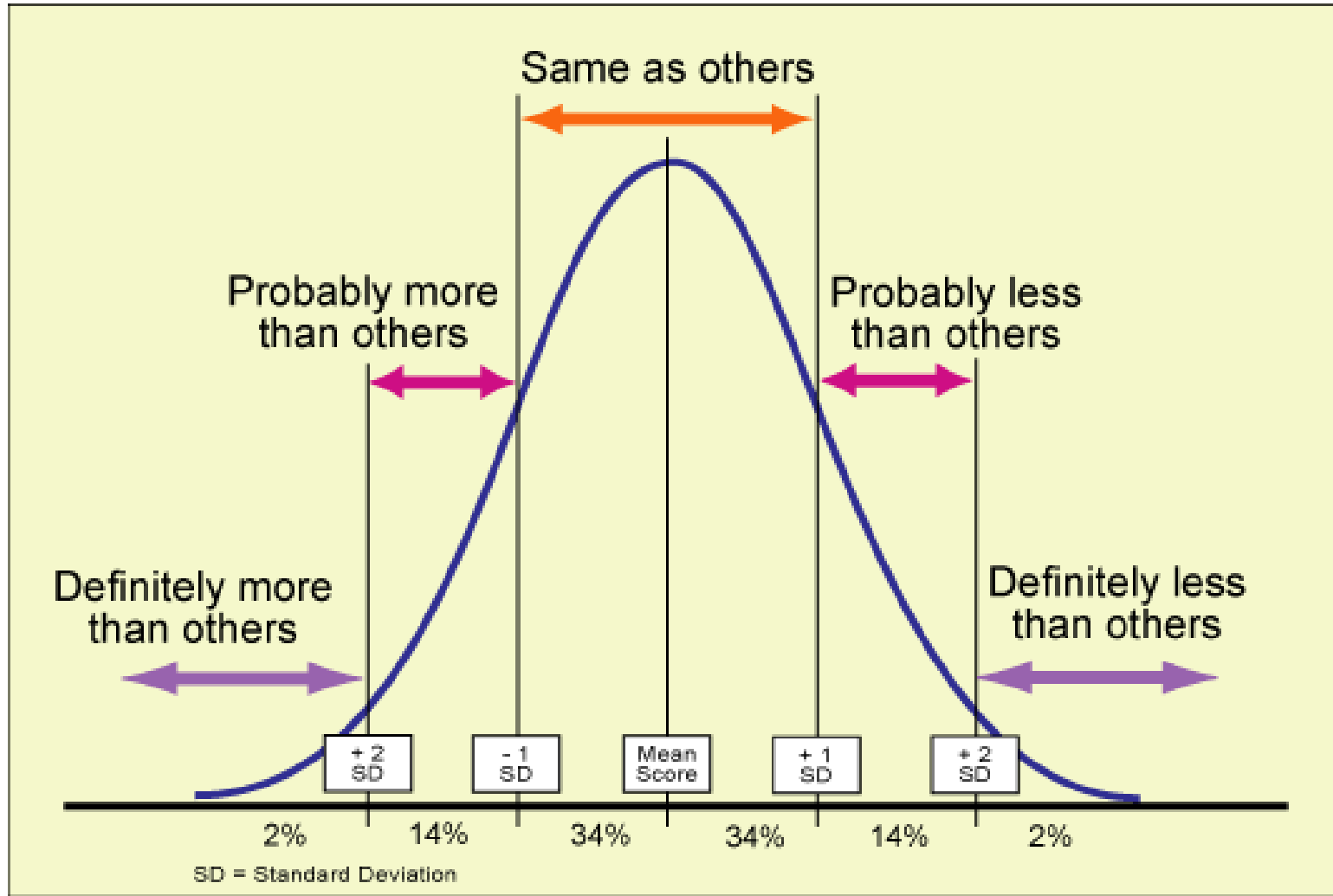
m = most likely time estimate

b = pessimistic time estimate

3) Calculate the standard deviation of each task:

$$s_i = \frac{b_i - a_i}{6}$$

MEMO: bell curve (normal distribution)



[source : http://www.fontys.nl/lerarenopleiding/tilburg/engels/Toetsing/bell_curve2.gif]

PERT Example on a single task

- Planner 1 (P1) and Planner 2 (P2) are asked to estimate m , a and b for a given task

| | Planner 1 | Planner 2 |
|----------|------------------|------------------|
| m | 10d | 10d |
| a | 9d | 9d |
| b | 12d | 20d |

- Calculate estimated time and standard deviation

| | Planner 1 | Planner 2 |
|------------------|------------------|------------------|
| PERT time | 10.2d | 11.5d |
| Std. Dev. | 0.5d | 1.8d |

- With the $(34\%+34\%=)$ 68% of probability
 - Planner 1 says that task will last between 9.7 to 10.7 days
 - Planner 2 says that task will last between 9.7 to 13.3 days

For the whole project:

- 1) Update the network diagram with the expected time of each task

- 2) Calculate the critical path with the CPM method
→ Result is the expected time of the whole project

- 3) For each task in the critical path, calculate the standard deviation of the project as:

$$s_{cp} = \sqrt{s_1^2 + s_2^2 + \dots + s_n^2}$$

- Advantages
 - Accounts for uncertainty

- Disadvantages
 - Time and labor intensive
 - Assumption of unlimited resources is big issue
 - Lack of functional ownership of estimates
 - Mostly only used on large, complex project

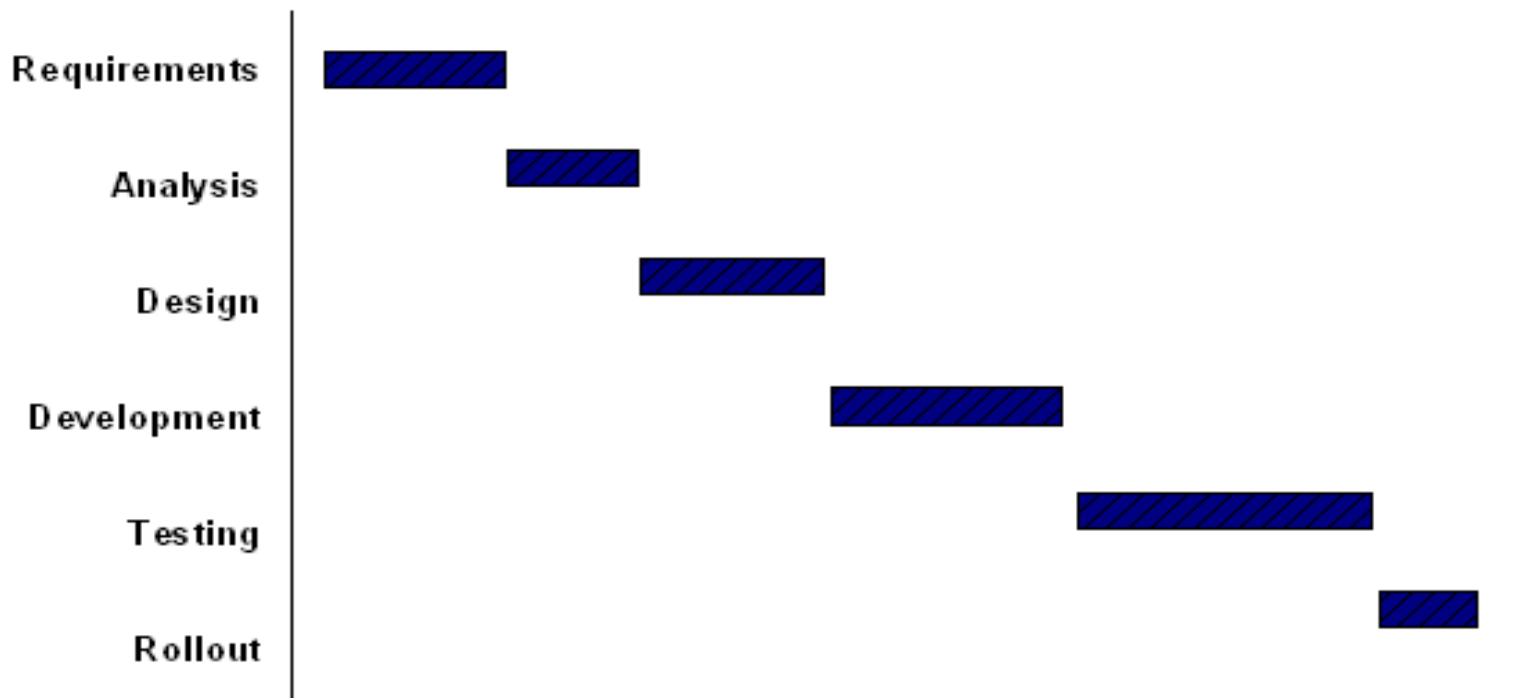
- Get PERT software to calculate it for you

- Both use Network Diagrams
- CPM: deterministic
- PERT: probabilistic
- CPM: one estimate, PERT, three estimates
- PERT is infrequently used

- Network Diagrams
 - CPM
 - PERT

- **Bar Charts**
 - Milestone Chart
 - Gantt Chart

- It shows the temporal sequence of tasks
 - One bar per each task
 - Bar length is proportional to task duration
- It shows only more general tasks of the WBS



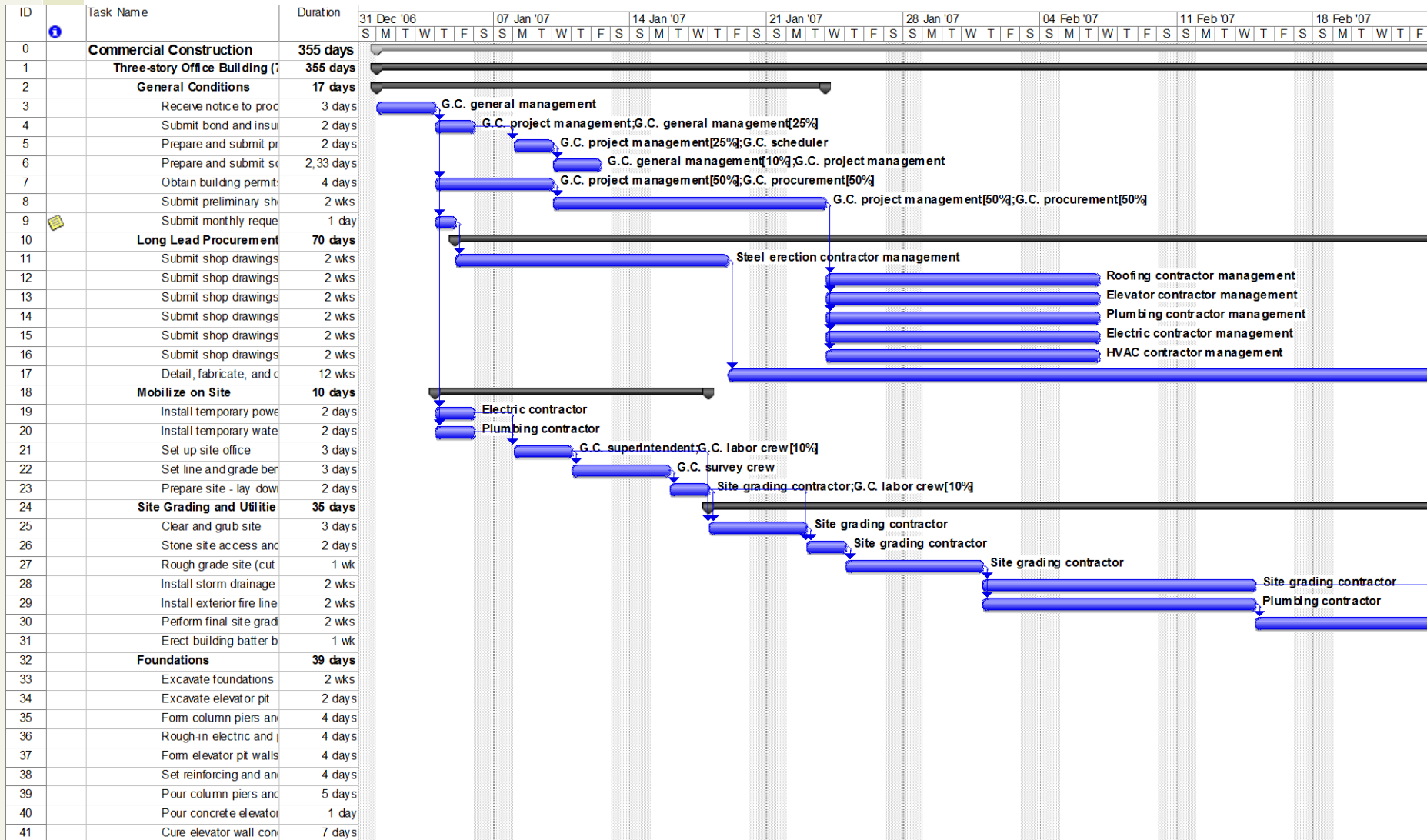
- As the milestone chart, it shows the temporal sequence of tasks and their duration
- Moreover
 - It consider all tasks organized in groups and subgroups
 - It shows dependencies among tasks
- It may also show:
 - Start and end dates of each task
 - Resources involved in each task
 - Percentage of completion of each task (useful during project controlling phases)
 - Today date

(Next lab lesson will entirely focus in drawing Gantt charts)

(A Gantt chart is required by last homework)

Gantt Chart

An example



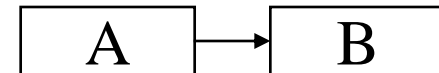
- Precedence:

- A task that must occur before another is said:

- To have **precedence** of the other

- or

- To be a **predecessor** of the other



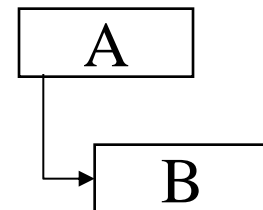
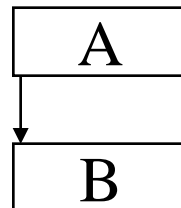
- Concurrency:

- Concurrent tasks are those that can occur in parallel at the same time

- Completely overlapped

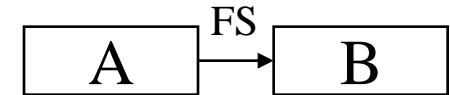
or

- Partially overlapped

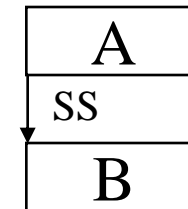


Dependency types

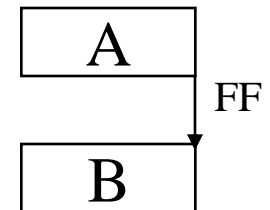
- *Finish-to-start*: "A f-to-s B": B cannot start until A finishes. B can start after A finishes.



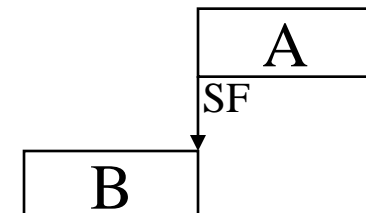
- *Start-to-start*: "A s-to-s B": B cannot start until A starts. B can start after A starts.



- *Finish-to-finish*: "A f-to-f B": B cannot finish until A finishes. B can finish after A finishes.

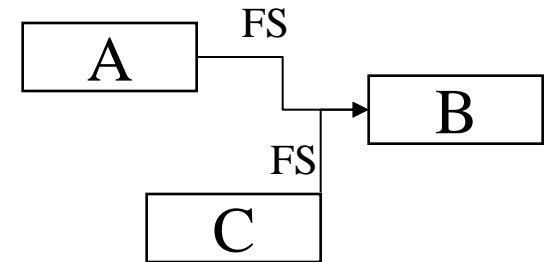
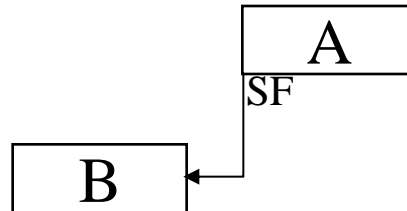
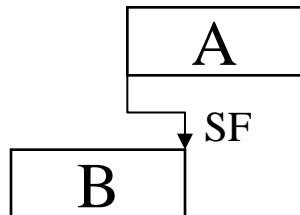
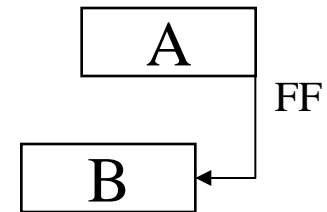
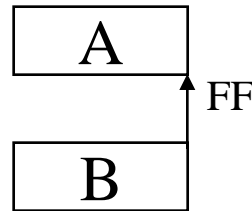
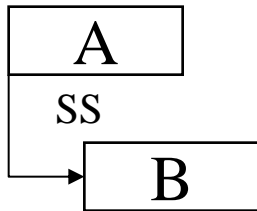


- *Start-to-finish*: "A s-to-f B": B cannot finish until A starts. B can finish after A starts.

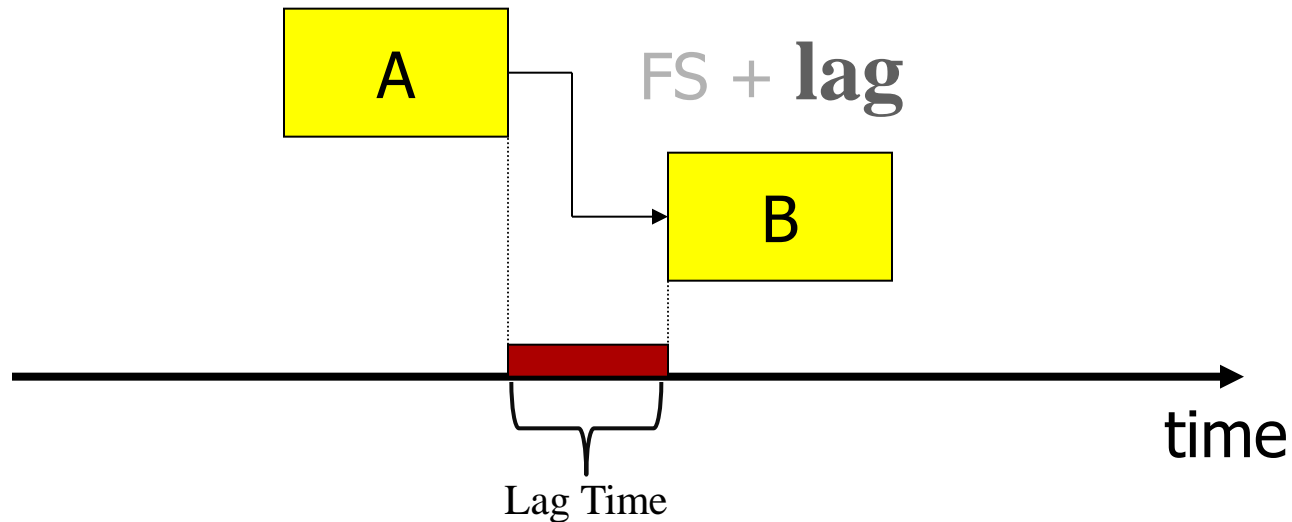


Dependency types (test)

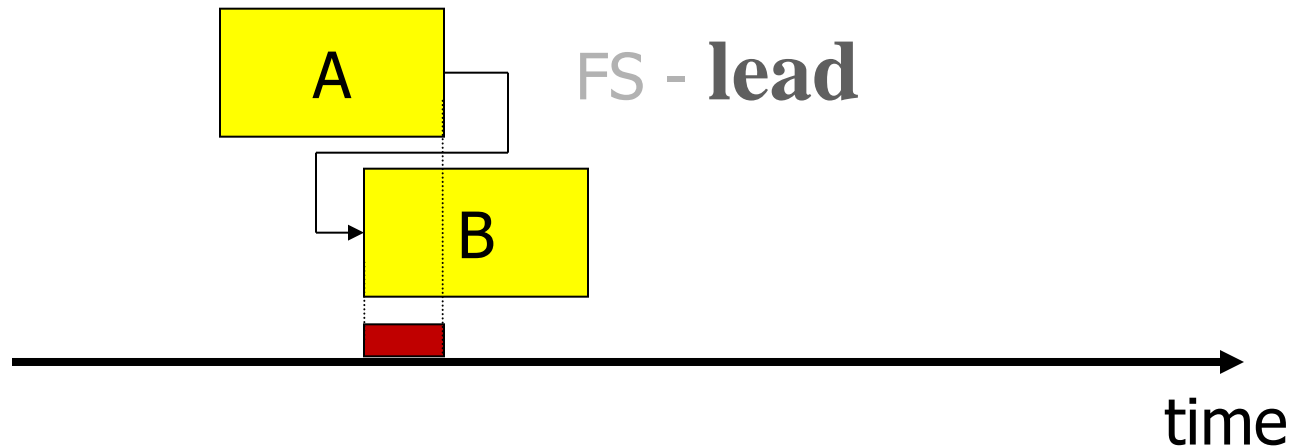
- Which ones are wrong?



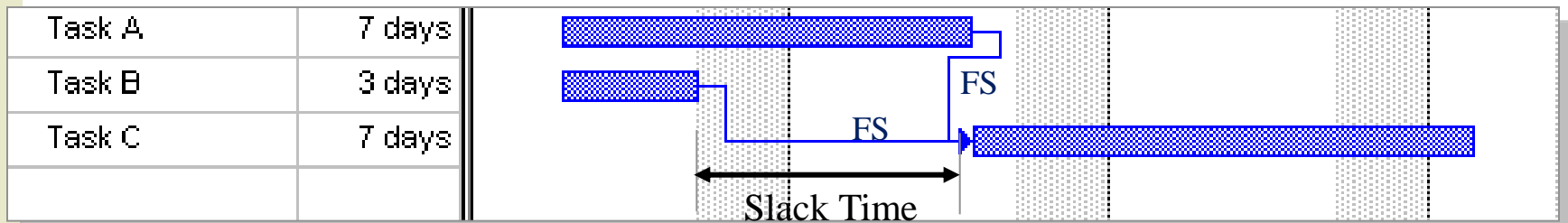
- It means a delay between tasks in sequence
- Example: if "A f-to-s B" and lag is equal to 10, B can start only 10 days after A end



- It means an advance between tasks in sequence
- Example: if "A f-to-s B" and lead is equal to 10, B can start 10 days before A end



- When the schedule contains some concurrent tasks, it may happen that some “free” time is between tasks

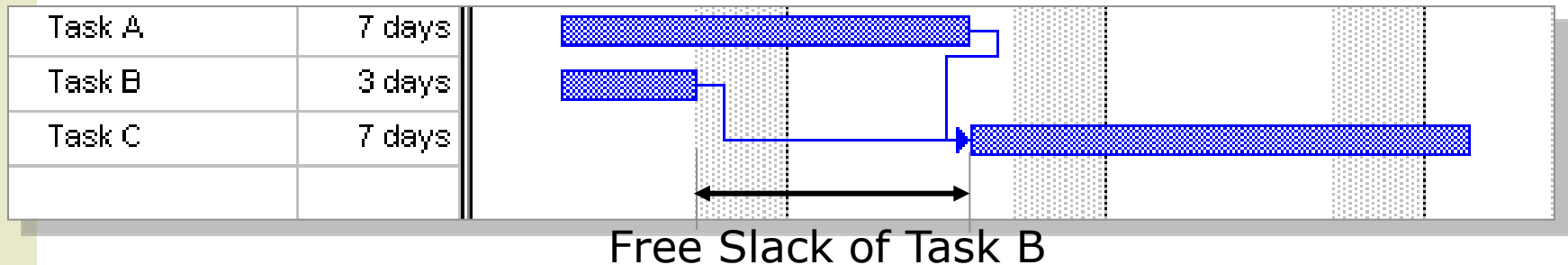


- A **Slack** is the amount of time that a task can be delayed without causing a delay to:
 - any downstream task (named **Free Slack**)
 - the end of the project (named **Total Slack**)
- Slack & Float are synonymous terms
 - We'll use Slack

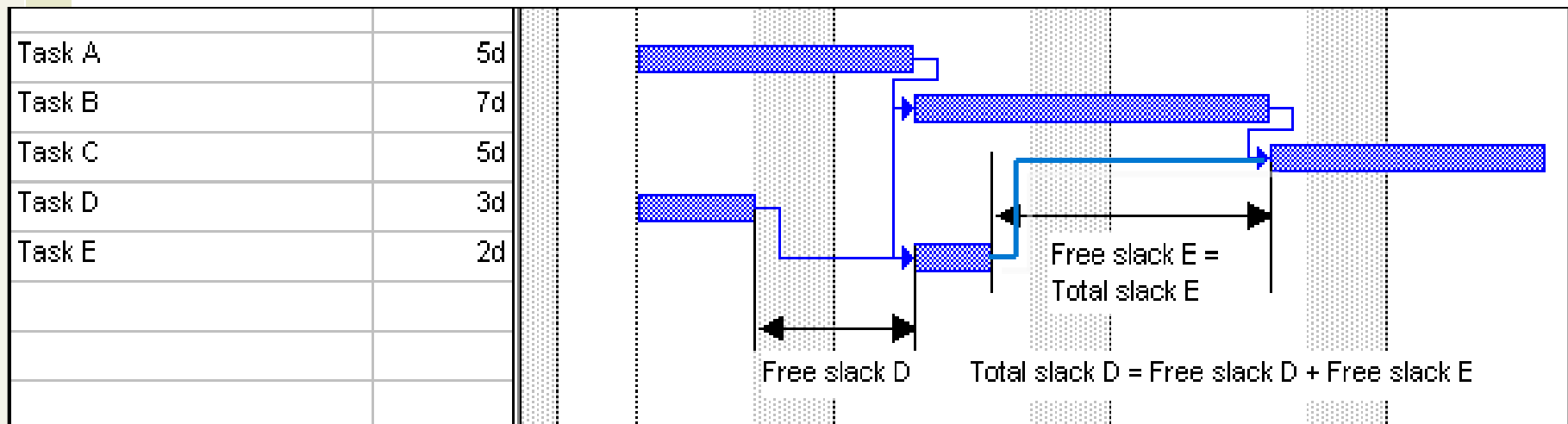
Gantt Chart

Free Slack

- Free Slack: Maximum delay of a task without causing a delay for any downstream task
- Example
 - A and B can be activated as soon as possible
 - Relationships
 - “A finish-to-start C”
 - “B finish-to-start C”



- Total Slack: Maximum delay of a task without causing a delay for the end of the project
 - It can cause delays to some downstream tasks



- Normally, if not specified, **slack** stands for **total slack** (as we did in the CPM)

Advantages and Disadvantages

- Advantages
 - Easily understood
 - Easily created and maintained
 - Good support from software tools

→ **Largely used**
- Disadvantages
 - It has difficulties to show complex relationships among tasks (network diagrams are better in this)
 - It does not show uncertainty of a given activity (as does PERT)

- Scheduling Fundamentals
- Scheduling Techniques
 - Network Diagrams
 - Bar Charts
- **Schedule Optimization Techniques**
- Mythical Man-Month

- Several techniques may be adopted to shorten the schedule
 - They act on the Trade-off Triangle (time, scope, cost)

1. Re-organize schedule:

- *Split and merge tasks* to reduce dependencies and slacks
- *Move resources among tasks* to better spread their effort in the time
 - A senior developer may act as a junior admin if needed
- Advantage: focus on a pure schedule optimization with no impact on scope and cost
- Disadvantages: risks increase
 - The schedule may become difficult to respect
 - Delays have stronger impact on the project (since slacks are reduced)
 - Resources may lose their motivation when moved across tasks

2. Reduce scope:

- *Reduce requirements* and consequently remove whole tasks
 - Advantage: cost and time reduction
 - Disadvantage: customers may not accept this
 - e.g. less tests, worsen performances, less graphics

- *Reduce quality* of software doing some tasks faster
 - e.g. less tests, worsen performances, less graphics
 - Disadvantages:
 - Customers and marketers may not accept this
 - Poorly tested software may cost more in the long time due to dependencies with other parts

3. Increase cost:

- *Crashing: Add resources to tasks*
 - To shorten tasks
 - E.g. 2 developers may do the same work in less time (but not in the half of the time)
 - To break resource dependencies
 - E.g. another developer permits to parallelize tasks that would otherwise be sequential
 - Disadvantage: increased costs
- *Fast Tracking: Break dependencies (both discretionary and some mandatory dependencies)*
 - Such tasks become overlapped even if they should be sequential
 - E.g. developing data model and business logic in parallel
 - Disadvantages:
 - New resources may be needed on concurrent tasks (cost increases)
 - Since some tasks need inputs from other parallel tasks, some effort is spent in working with unstable inputs and some rework may occur (cost increases)

- Scheduling Fundamentals
- Scheduling Techniques
 - Network Diagrams
 - Bar Charts
- Schedule Optimization Techniques
- **Mythical Man-Month**

- Book: “The Mythical Man-Month”
 - Author: Fred Brooks
 - <http://www.amazon.com/exec/obidos/ASIN/0201835959/qid%3D1022856693/sr%3D1-1/ref%3Dsr%5F1%5F1/103-4280067-9687806>
 - <http://my.safaribooksonline.com/0201835959>
- “The classic book on the human elements of software engineering”
- First two chapters are full of terrific insight (and quotes)

- “Cost varies as product of men and months, progress does not”
- “Hence the man-month as a unit for measuring the size of job is a dangerous and deceptive myth”
- “Good cooking fakes time. If you are made to wait, it is to serve you better, and to please you - Menu of Restaurant Antoine, New Orleans -”

- Why is software project disaster so common?
 1. Estimation techniques are poor & assume things will go well (an 'unvoiced' assumption)
 2. Estimation techniques fallaciously confuse effort with progress, hiding the assumption that men and months are interchangeable
 3. Because of estimation uncertainty, managers lack courteous stubbornness of Antoine's chef
 4. Schedule progress is poorly monitored
 5. When schedule slippage is recognized, the natural response is to add manpower. Which, is like dousing a fire with gasoline

- Optimism
 - “All programmers are optimists”
 - 1st false assumption: “all will go well” or “each task takes only as long as it ‘ought’ to take”
 - **The Fix: Consider the larger probabilities**
- Cost (overhead) of communication (and training)
 - Overhead: $n(n-1)/2$
 - How long does a 12 month project take?
 - 1 person: 12 month
 - 2 persons = $12/2 + 2(2-1)/2 = 6+1 = 7$
 - 2 man-month extra
 - 3 persons = $12/3 + 3(3-1)/2 = 4 + 3 = 7$
 - 9 man-months extra
 - 4 persons = $12/4 + 4(4-1)/2 = 3 + 6 = 9$
 - **The Fix: don't assume adding people will solve the problem, add people in well-separated tasks**

- Sequential nature of the process
 - “The bearing of a child takes nine months, no matter how many women are assigned”
- What is the most mis-scheduled part of process?
 - Testing
- Why is this particularly bad?
 - Occurs late in process and without early-warning
 - Higher costs
- **The Fix: Allocate more test time**
 - **Understand task dependencies, test and fix before use**

- Reliance on hunches and guesses
 - What is 'gutless estimating'?
 - Urgency of Client causes Optimistic Estimates
 - E.g., omelet and chef analogy
 - <http://my.safaribooksonline.com/0201835959/ch02lev1sec4>
 - Regardless of Urgency, tasks require the same amount of time

- The myth of additional manpower
 - Brooks Law
 - "Adding manpower to a late project makes it later"
 - http://en.wikipedia.org/wiki/Brooks%27s_law

- Q: “How does a project get to be a year late”?
 - A: “One day at a time”
- Studies
 - Each task: twice as long as estimated
 - Only 50% of work week is real coding
 - The rest 50% is communication, negotiation, documentation, debugging, ...
- **The Fixes**
 - Consider the 50% not-coding time
 - Define clearly measurable milestones
 - No “fuzzy” milestones
 - Reduce the role of conflict among persons
 - Identify the “true status” of a task
 - It’s impressive how much effort is needed to move a 90% done task to a 100% done task

Questions?