Week 12 - Wednesday

COMP 3100

Last time

- What did we talk about last time?
- Financial and economic planning
  - Time value of money
  - $F_n = P \cdot (1 + r)^n$

Questions?

Project 4

Finishing Financial Planning

Internal rate of return

- What if you knew how much someone would pay you today and how much you could get paid in the future and needed to compute the rate of return needed to make them match?
  - This helps you look for another way to spend your money with a better interest rate
  - Or it helps you understand the rate of return that a project provides
- It's algebra, solve for *r*:
  - $F_n = P \cdot (1+r)^n$
  - $\frac{F_n}{P} = (1+r)^n$
  - $r = \left(\frac{F_n}{P}\right)^{\frac{1}{n}} - 1$
- If you have multiple stages of costs and revenues, you'll need to do a binary search on *r* values:
  1. Start with a minimum bound for *r* and a maximum bound for *r*
  2. Guess the rate halfway between them
  3. Run through the math on a previous slide to see what the net is
  4. If it's too high, go back to Step 1 with the minimum and the midpoint as your range
  5. If it's too low, go back to Step 1 with the midpoint and the maximum as your range
  6. When the minimum and the maximum are close enough together (like 0.001%), you have a good estimate

Expected value

- Assume each sample point has a value (like the money associated with that outcome)
- The **expected value** is the value of each sample point multiplied by its probability
  - It's the average value of everything, weighted by the probability that it happens
- Example:
  - You're playing roulette, always betting on black
  - An American roulette wheel has 38 outcomes: 18 are red, 18 are black, and two are neither (0 and 00)
  - If you bet $1 on black:
    - You have an 18/38 chance of winning $1
    - You have a 20/38 chance of losing $1
  - Expected value = $\$1 \cdot \frac{18}{38} - \$1 \cdot \frac{20}{38} \approx -\$0.05$
  - Thus, you'll win some and lose some, but on average, you'll lose about $0.05 each time they spin the wheel

Simple example with uncertainty

- Your company needs to install some free software
  - There's a 20% chance that the installation will be effortless and cost about $100 of worker time
  - There's an 80% chance that the installation will be a huge pain and cost about $8,000 of worker time
- Expected cost of the installation is:

$$0.20 \cdot \$100 + 0.80 \cdot \$8{,}000 = \$20 + \$6{,}400 = \$6{,}420$$

Complex example with uncertainty

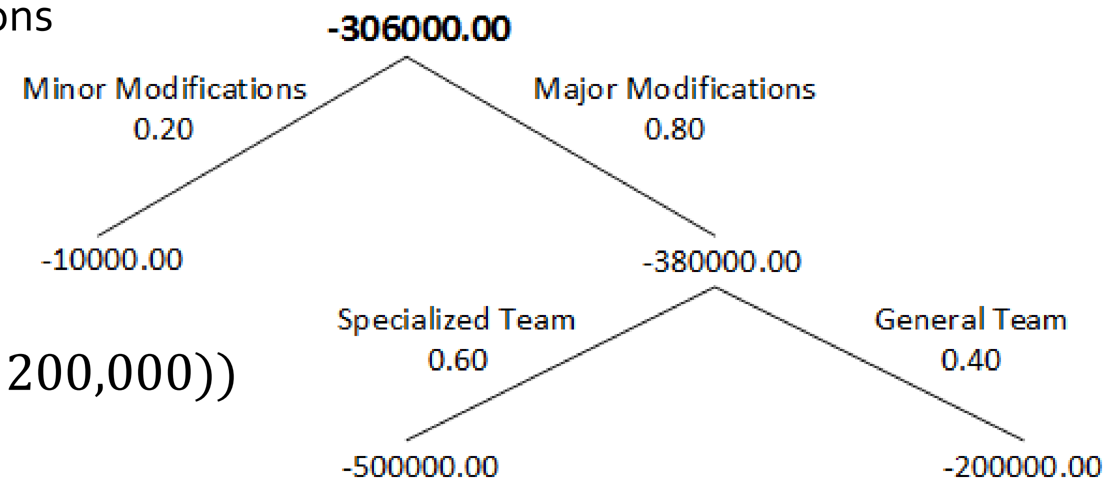
- We can take the go-no go example and add probability to it
- We'll say that there's a 25% chance that the customer goes broke after three years, ending the revenue we'd get (and also the costs)
- The table showing this outcome looks like this:

n	R	C	$(1 + r)^n$	$R / (1 + r)^n$	$C / (1 + r)^n$	Net
0	0.00	-450000.00	1.0	0.00	-450000.00	-450000.00
1	100000.00	-80000.00	1.04	96153.85	-76923.08	19230.77
2	100000.00	-80000.00	1.0816	92455.62	-73964.50	18491.12
3	100000.00	-80000.00	1.124864	88899.64	-71119.71	17779.93
4	0.00		1.16985856	0.00	0.00	0.00
5	0.00		1.2166529024	0.00	0.00	0.00
				277509.11	-672007.29	-394498.18

- The expected value is $0.75 \cdot \$18{,}109.22 + 0.25 \cdot \$-394{,}498.18 = \$-85{,}042.63$
- This negative expected value means the project will likely lose money

Probability trees

- The previous example covered only two different possibilities
- There could be many possibilities, and each possibility might be broken down further into sub-possibilities
- In these situations, we can show the possibilities as a tree
- Working from the bottom of the tree, we can determine the discounted present value of each outcome
- Then, we can weight these outcomes by their probabilities to get an overall expected value
- The following tree lets us understand the expected value of the cost of a project to modify an open source product
  - There's a 20% chance it will require only minor modifications
  - There's an 80% chance it will require major modifications
    - If major, there's a 60% chance it will require a specialized team
    - If major, there's a 40% chance it can use a general team



- $0.2 \cdot \$-10{,}000 + 0.8 \cdot (0.6 \cdot \$-500{,}000 + 0.4 \cdot \$-200{,}000))$
- $= \$-306{,}000$

Scheduling

Scheduling

- Two weeks ago, we talked about effort estimation
- Effort estimation predicts the number of person-months needed to do a project
- Even if we had a perfect estimate of the amount of work to be done, we would still have to take many things into account to predict when the project will be done
  - How many people
  - Details of tasks
  - Task dependencies
  - Personnel capabilities

People

- Effort *E* is given in person-days or person months
- Thus, time *T* could be given by the following equation where *N* is the number of people:

$$T = E / N$$

- Unfortunately, this ideal equation is unlikely to work out for a couple of reasons

Details of tasks

- Some tasks are easy to split up, and others are not
- If it takes 5 minutes to pump up a bicycle tire, you can't do it 100 times faster by using 100 people instead of one
- Small tasks can usually only be done by a single person at a time
- Larger tasks generally obey the $T = E / N$ rule, but there are diminishing returns for large values of N

Task dependencies

- Looking at the time to do individual tasks isn't enough
- Consider tasks A, B, and C with the following amounts of effort:
  - A: 5 person-months
  - B: 3 person-months
  - C: 4 person-months
- If we have three employees, one can work on each task
- If the tasks are independent, the project will take 5 months to do, and tasks B and C can even run late without delaying the total project
- What if C requires B to be done and B requires A to be done?
  - If any task is delayed, the whole project will be delayed
  - We have to share work on each task

Personnel capabilities

- Some developers are better than others
  - This messes with the overall $T = E / N$ rule
- Some developers have specialized in certain areas
  - A tester might be great at testing but not so good at development
  - Only one person on the team might have experience with GUIs
- As a consequence, it might not be possible to have multiple people working on a given task, and one person might be needed for two different tasks
- Agile methodologies supposedly improve these issues by trying to make everyone work on everything and grow their skills

Simplifying assumptions

- We assume that we have a good estimate of the relationship between effort and time
- We assume small tasks assigned to one person
- We assume a dependency between two tasks if only one person has the skills needed to do both
  - This allows us to look at the problem of specific skillsets as the more general problem of dependencies
- With these assumptions, we can organize our tasks by duration and dependency

Dependency example

- The following example shows 14 tasks
- The time for each task is given
- The prerequisite tasks that must be done first are listed too
- Tasks are numbered so that higher number tasks are dependent on lower number tasks

Task Number	Duration (Days)	Prerequisite Tasks
1	6	-
2	5	1
3	2	1
4	6	1
5	4	2, 3
6	1	4
7	2	4
8	4	4
9	3	2
10	4	5, 6
11	1	7, 8
12	4	9, 10
13	2	6, 11
14	1	12, 13

Another view of the same tasks

- The previous slide has all the information we need
- But it's not displayed in a way that is helpful for every kind of analysis
- For example, it's hard to figure out how long the whole project will take
- It's also hard to identify **critical tasks**, the ones that determine the minimum time for the project
- Another thing we want to see is **slack** (or **float**), the amount of time non-critical tasks can slip without delaying the project
- To the right is another view that shows which tasks are dependent on a given task

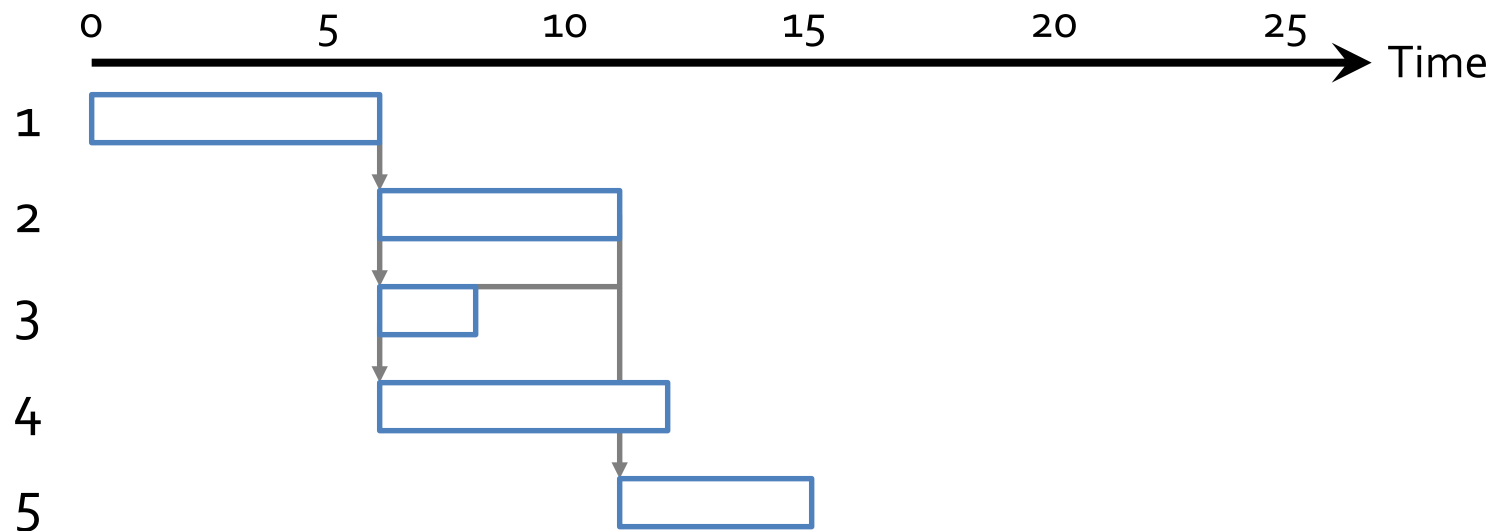
Task Number	Subsequent Tasks
1	2, 3, 4
2	5, 9
3	5
4	6, 7, 8
5	10
6	10, 13
7	11
8	11
9	12
10	12
11	13
12	14
13	14
14	-

Gantt charts

- Gantt charts let us find total time, critical tasks, and float times
  - Tasks are represented as rectangles with length proportional to duration
  - Dependencies between tasks are arrows
  - Time increases from left to right
  - We put the task starts as early as possible, immediately after their last prerequisite finishes

Partial Gantt chart example

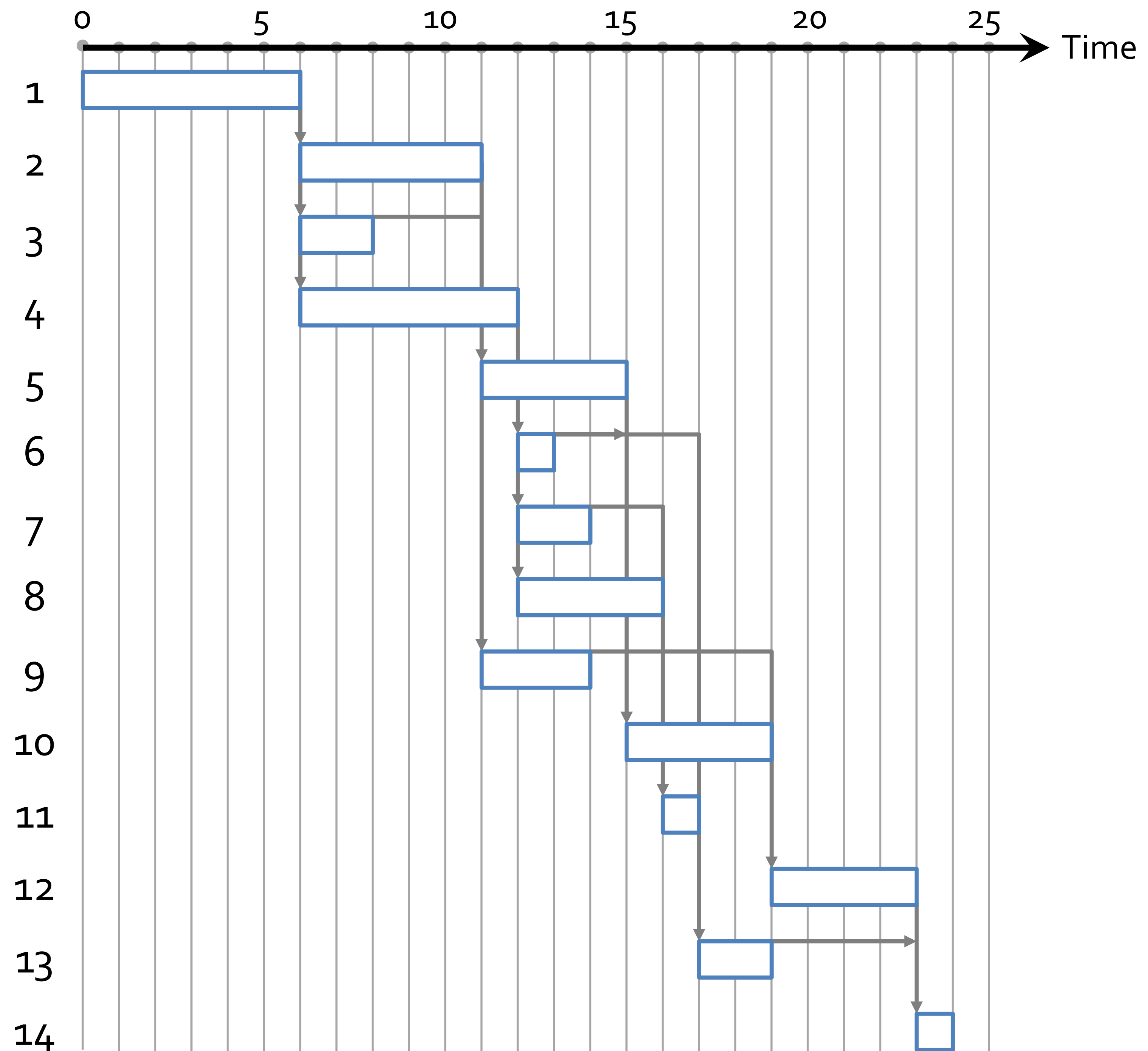
- The first five tasks from our earlier table have the following characteristics
- Corresponding Gantt chart:



Task	Duration	Prerequisites
1	6	-
2	5	1
3	2	1
4	6	1
5	4	2, 3

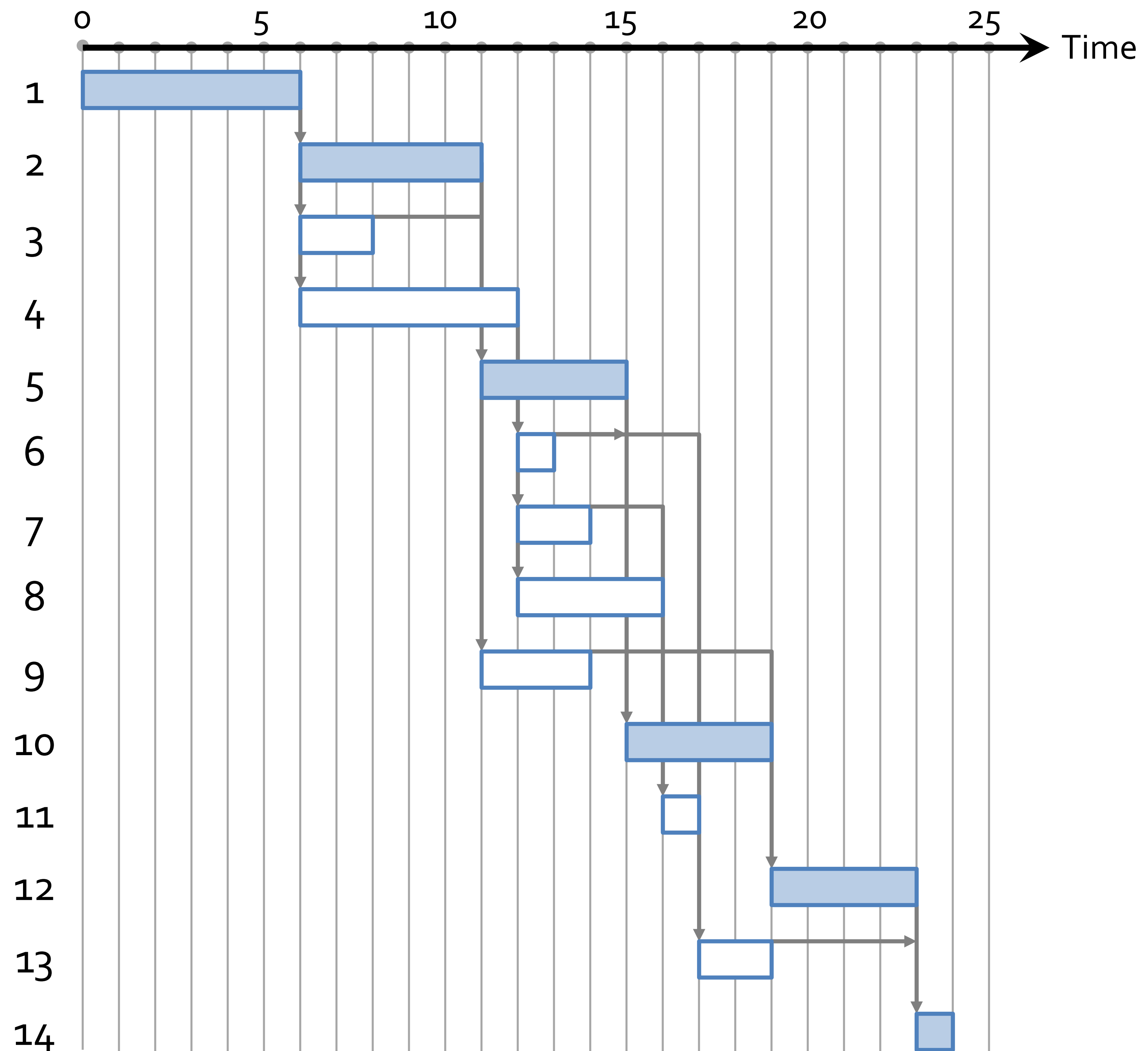
Full Gantt chart

- Here is the full Gantt chart
- People don't always draw the arrows, but we're doing so here to be explicit
- Looking carefully at the chart, it's clear that the project will take 24 days



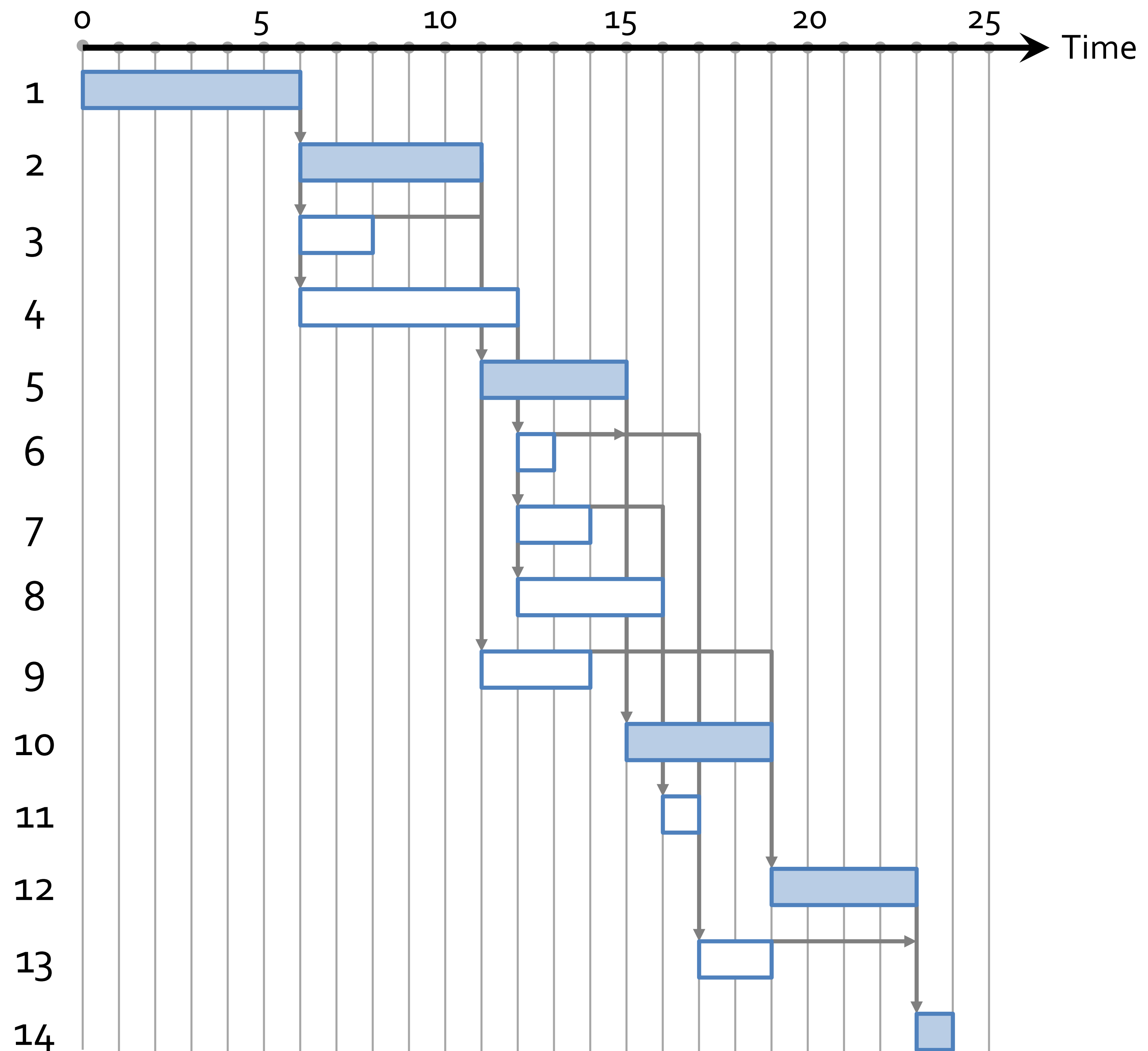
Critical tasks

- We can find the critical tasks by working backward from the task(s) with the latest finish time
- Whichever of its predecessors have the latest end time are also critical
- If any of these are delayed, the whole project will be delayed



Slack time

- Non-critical tasks have **slack**, an amount of time they can slip by and still not delay the project
- Horizontal arrows show slack times:
  - Task 3: 3 units
  - Task 6: 2 units
  - Task 7: 6 units
    - Tricky! It's 6 because that's what's needed before a critical task will be delayed
  - Task 9: 5 units
  - Task 13: 4 units



Gantt tools

- Tools exist to make Gantt charts automatically from duration and prerequisite data
- Such tools can identify critical tasks and slack times
- They're only as good as the input you give them
- They won't help you:
  - Break your project into meaningful tasks
  - Estimate how long those tasks take
  - Come up with task prerequisites

Critical path methods

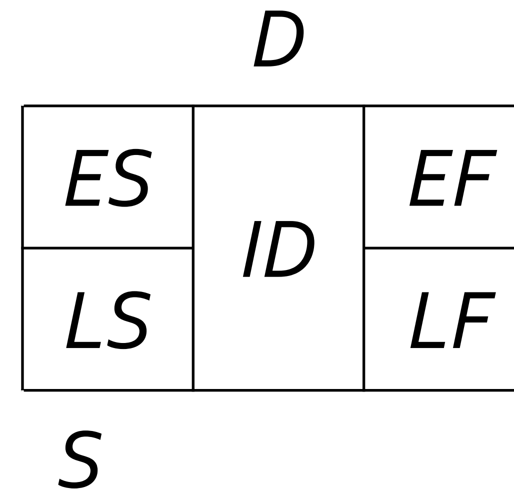
- Computer scientists love to use computer science for everything, even project management problems
- In addition to Gantt charts, similar information can be represented using graphs
  - Then, graph theory tools can be applied to the information
- These approaches are called critical path methods (CPM) because they focus on making the critical path as short as possible

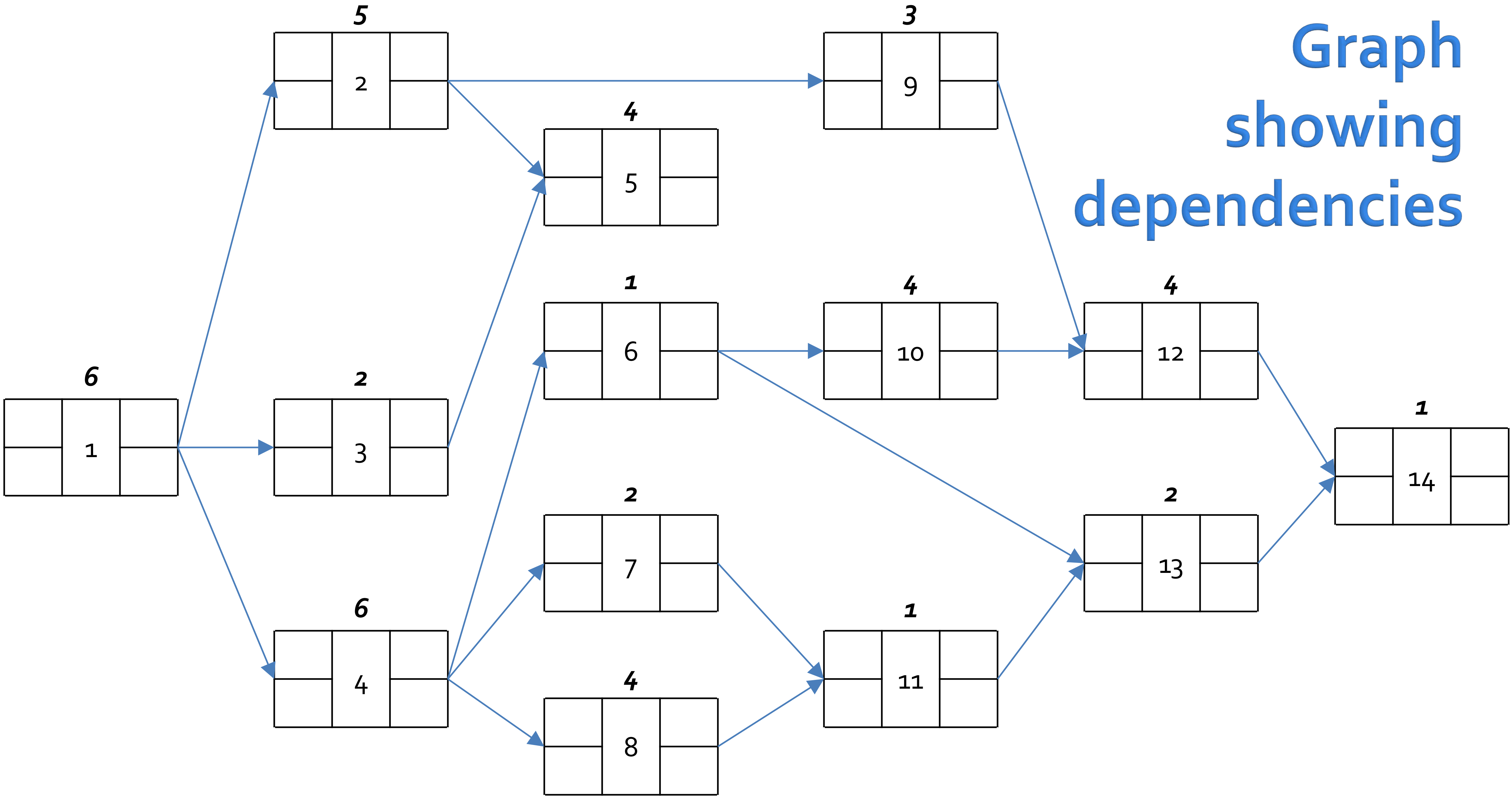
More on critical path

- An important idea that critical path methods add to the mix is a tradeoff between time and cost
- Each task has:
  - A **normal time** that the task would take
  - A **crash time** which is the fastest a task could possibly be done by spending more resources
  - A (usually linear) relationship between putting resources in and getting the task done quicker
- By using linear programming, a technique for finding optimal solutions to linear systems of equations, the cheapest way to finish a project by a given deadline could be determined
  - Maybe rushing Task 7 is worth the extra money but rushing Task 10 isn't

Nodes in a CPM graph

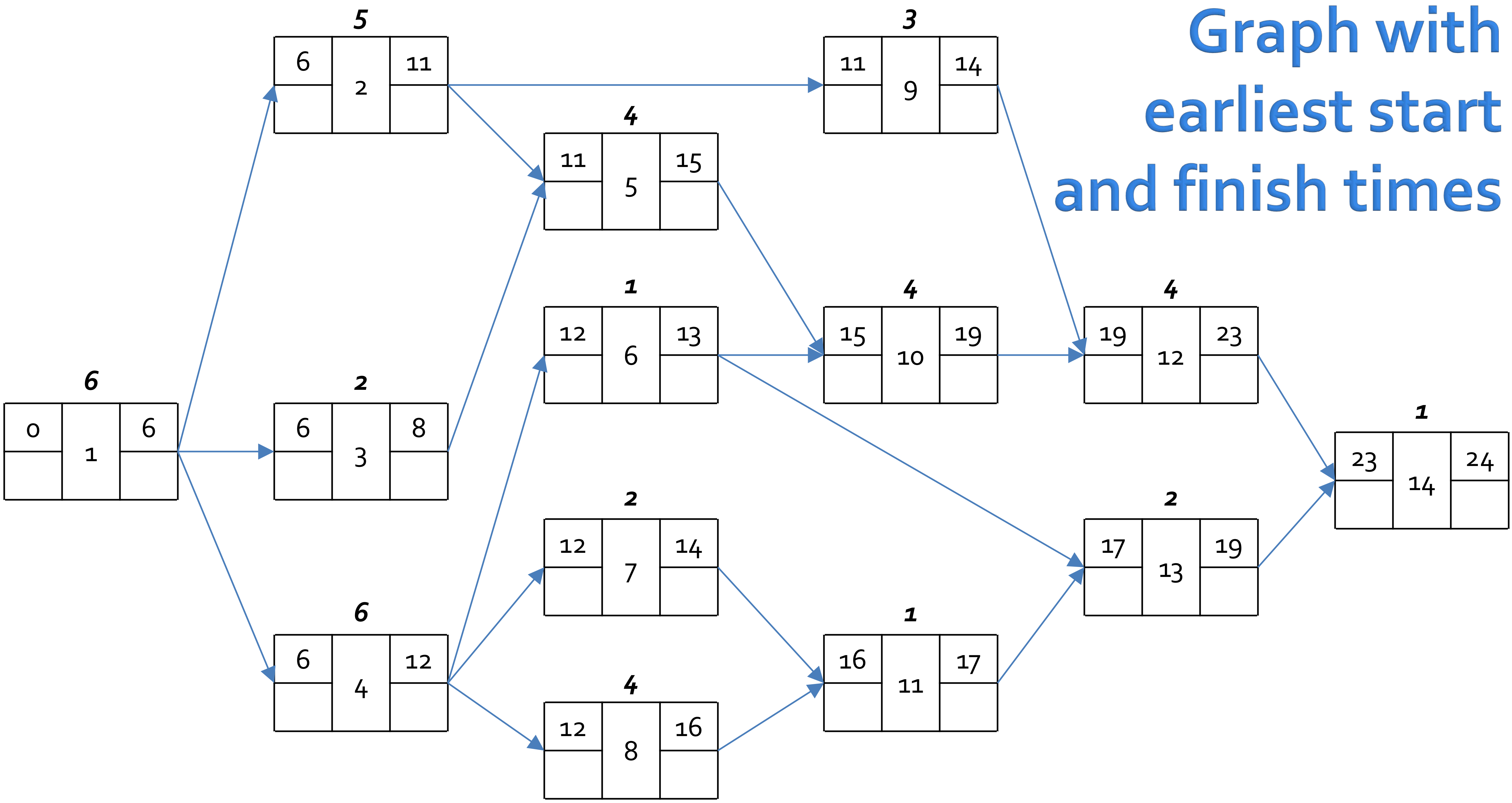
- The CPM we will talk about has nodes containing seven pieces of information, written in a peculiar way
  - *ID*: Task identifier
  - *D*: Task duration
  - *ES*: Earliest start time
  - *EF*: Earliest finish time
  - *LS*: Latest start time
  - *LF*: Latest finish time
  - *S*: Slack




Graph showing dependencies
6
1
5
2
2
3
6
4
4
5
1
6
2
7
4
8
3
9
4
10
1
11
4
12
2
13
1
14

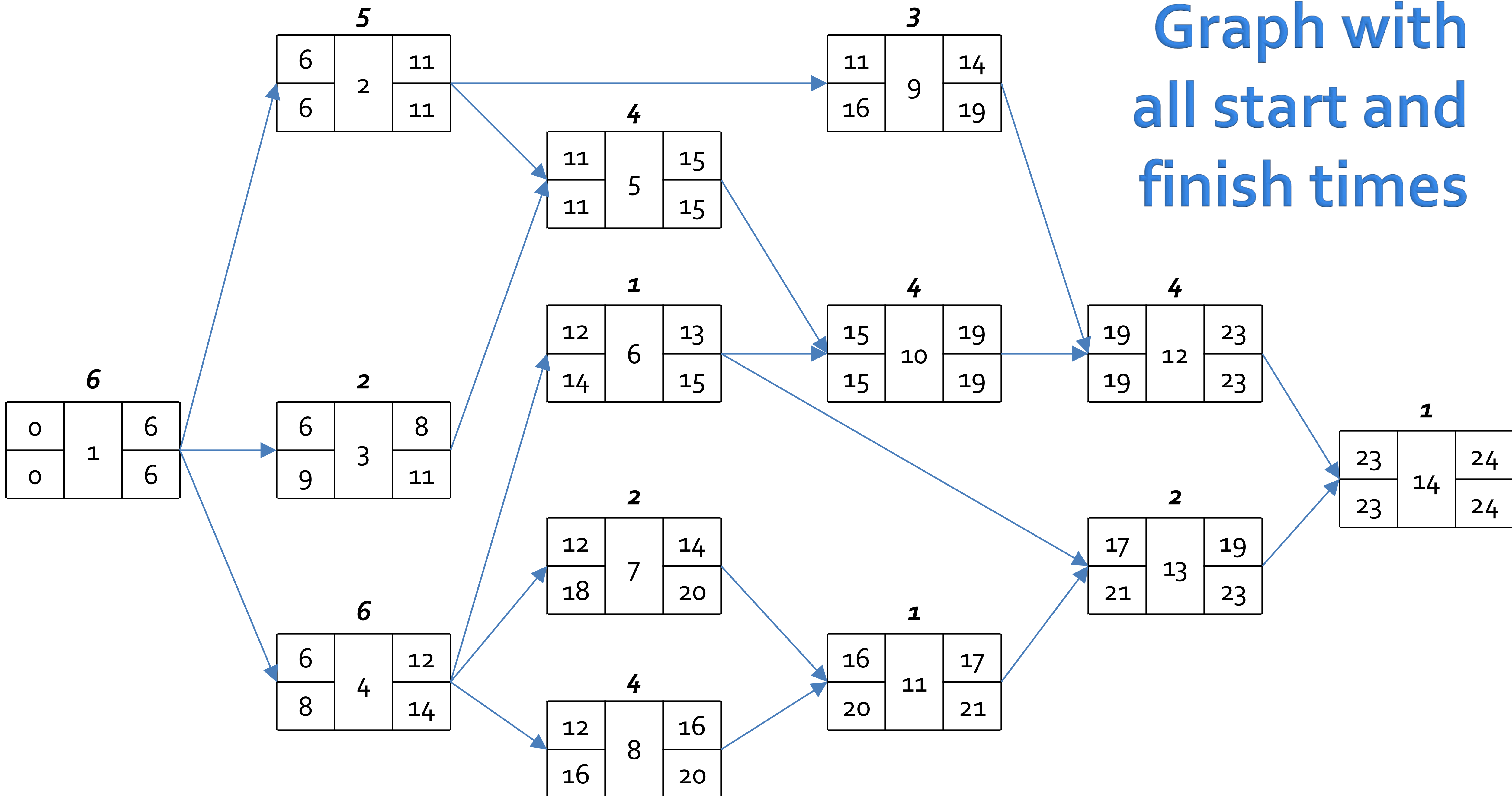
Earliest start and finish times

- Every task with no prerequisite has an *ES* of 0
- For a task with prerequisites, its *ES* is the maximum *EF* of all of its prerequisites
- For each task, $EF = ES + D$
- Using these relationships, we can fill in the *ES* and *EF* for each task, starting from those with no prerequisites and working through the rest of the graph


Graph with earliest start and finish times
6
0 1 6
5
6 2 11
2
6 3 8
6
6 4 12
3
11 9 14
4
11 5 15
1
12 6 13
2
12 7 14
4
12 8 16
4
15 10 19
1
16 11 17
4
19 12 23
2
17 13 19
1
23 14 24

Latest start and finish times

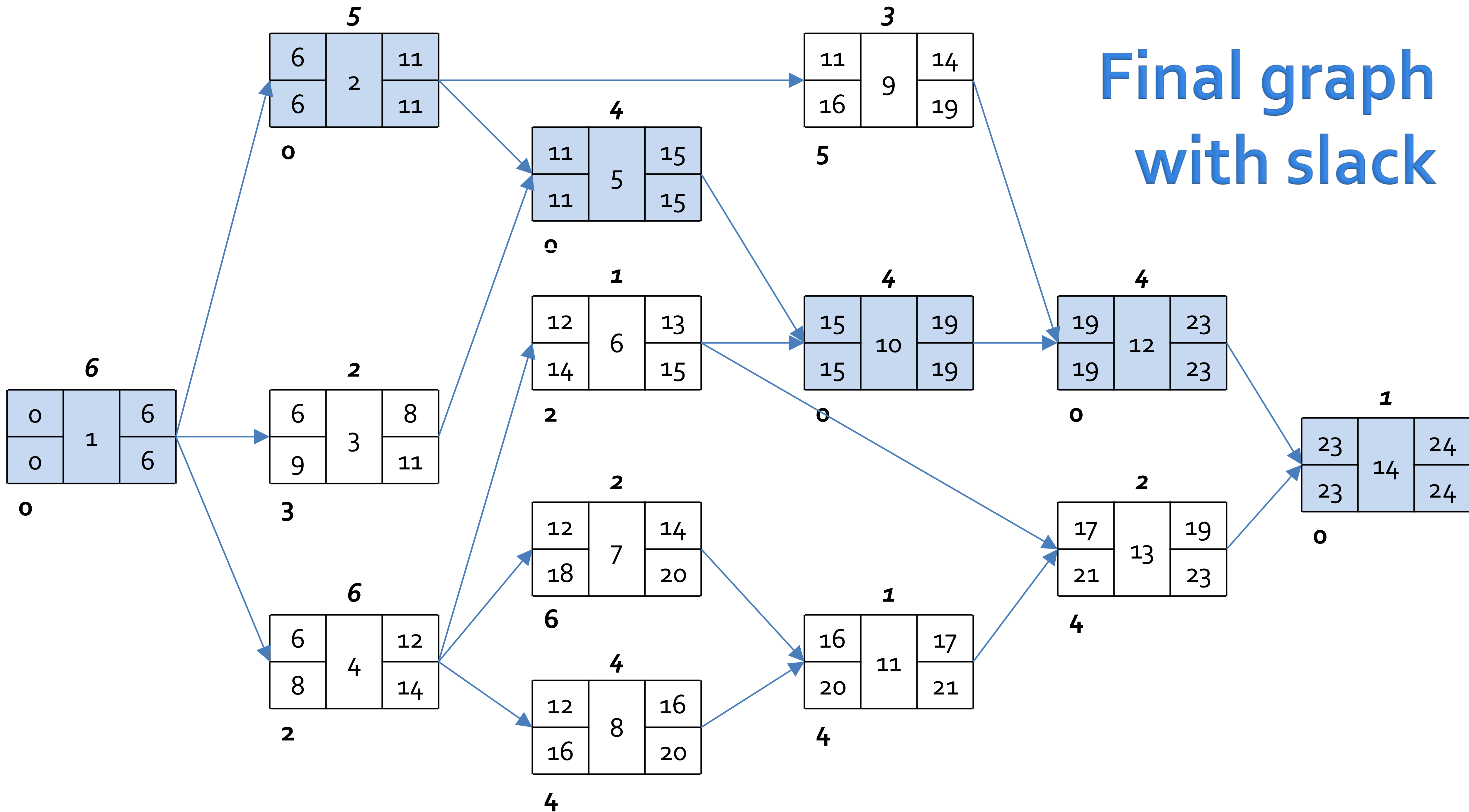
- Every task that isn't the prerequisite for anything has an $LF = EF$
- For a task that is the prerequisite for something, its LF is the minimum LS of the tasks it's a prerequisite for
- For each task, $LS = LF - D$
- Using these relationships, we can fill in the LF and LS for each task, starting from those that aren't the prerequisites for anything and working through the rest of the graph



Graph with all start and finish times
6
0 1 6
0 6
5
6 2 11
6 11
2
6 3 8
9 11
6
6 4 12
8 14
4
11 5 15
11 15
1
12 6 13
14 15
2
12 7 14
18 20
4
12 8 16
16 20
3
11 9 14
16 19
4
15 10 19
15 19
1
16 11 17
20 21
4
19 12 23
19 23
2
17 13 19
21 23
1
23 14 24
23 24

Slack

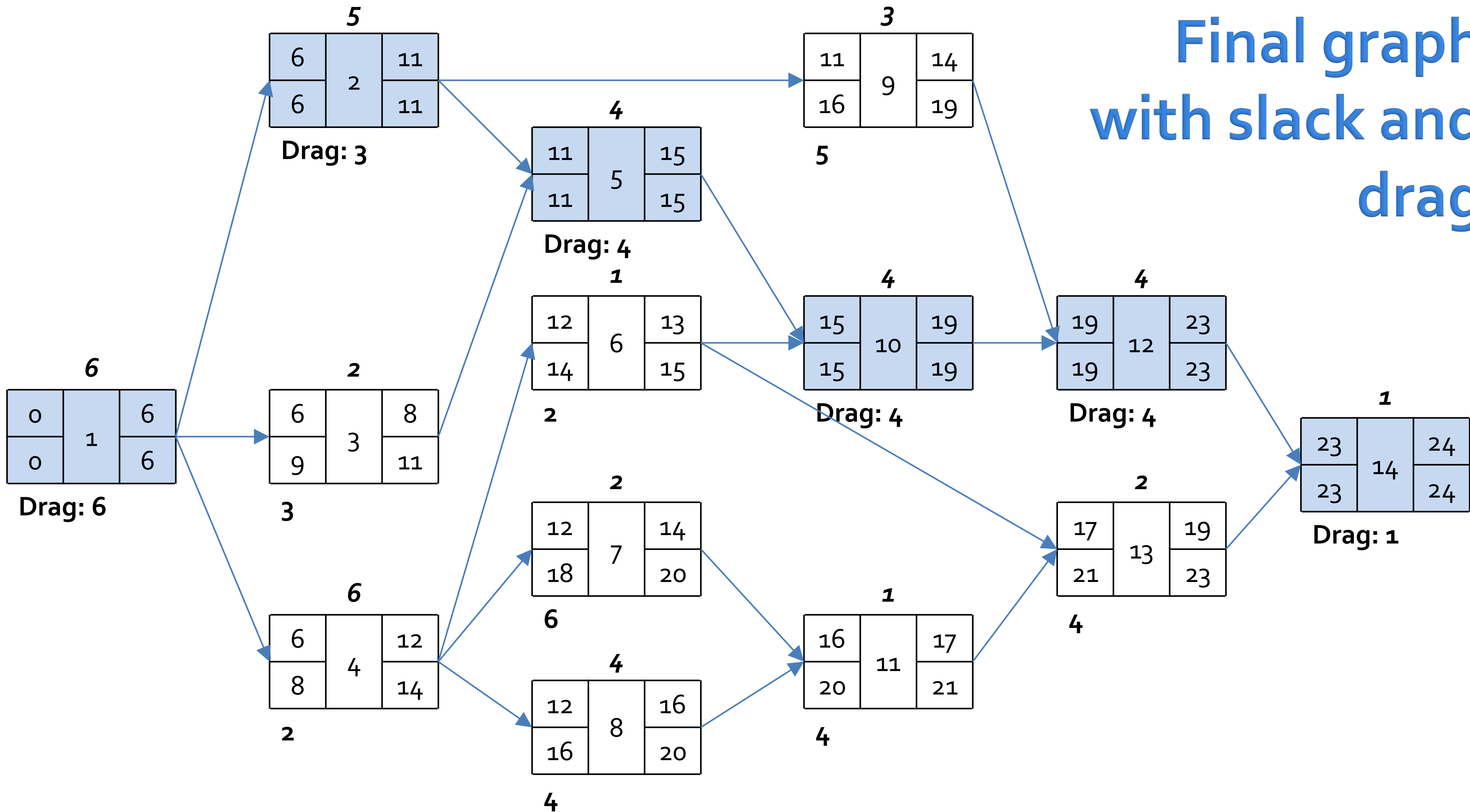
- For each task, the slack time $S = LF - EF$
- We can run through the graph and mark that as well
- Nodes with no slack are on the critical path



Final graph
with slack
6
0 1 6
0 6
0
5
6 2 11
6 11
0
2
6 3 8
9 11
3
6
6 4 12
8 14
2
4
11 5 15
11 15
0
1
12 6 13
14 15
2
2
12 7 14
18 20
6
4
12 8 16
16 20
4
3
11 9 14
16 19
5
4
15 10 19
15 19
0
1
16 11 17
20 21
4
4
19 12 23
19 23
0
2
17 13 19
21 23
4
1
23 14 24
23 24
0

Prioritization

- This approach clearly shows the relationship between tasks and allows us to focus on the critical paths
- Managers might try to shorten the critical path by "crashing" it, putting more resources on tasks with no slack
- The book doesn't mention it, but critical path tasks also have **drag**, the amount by which they are delaying the project
  - If no other tasks are done in parallel with the critical task, its drag is its duration
  - If other non-critical tasks are done in parallel, the drag is the minimum of all of the parallel floats and the duration of the critical task



Final graph with slack and drag
6
0 1 6
0 6
Drag: 6
5
6 2 11
6 11
Drag: 3
2
6 3 8
9 11
3
6
6 4 12
8 14
2
4
11 5 15
11 15
Drag: 4
1
12 6 13
14 15
2
2
12 7 14
18 20
6
4
12 8 16
16 20
4
3
11 9 14
16 19
5
4
15 10 19
15 19
Drag: 4
1
16 11 17
20 21
4
4
19 12 23
19 23
Drag: 4
2
17 13 19
21 23
4
1
23 14 24
23 24
Drag: 1

Quiz

Upcoming

Next time…

- Friday is a work day
- Next Monday:
  - Execution and control

Reminders

- **Work on Project 4**
- Read Chapter 15: Execution and Control for Monday