Aston University

BIRMINGHAM UK

Software Project Management

Unit 3: Software Project Planning (2)

Thais Webber
Richard Lee

Outline

- Frameworks for software project planning
- Selection of software project approaches
  - Analysing other project characteristics
  - Development process models
- Effort estimation for software projects
- Activity planning and resource allocation

Selection of software project approaches

- Look at risks and uncertainties, e.g.
  - Are requirements well understood?
  - Are technologies to be used well understood?
- Look at the type of application being built, e.g.
  - Information system? Embedded system?
  - Safety-critical software? Mobile application?
- Approaches with heavy **structure** versus Approaches with pressure about **speed** of delivery?
- Software development **process that best fit the needs**?

Structure versus speed of delivery

Structured approaches

- Also called heavyweight approaches

- Step-by-step methods where each step and intermediate product is carefully defined

- Emphasis on getting quality right first time

- Example: use of UML (i.e., Unified Modelling Language)

Structure versus speed of delivery

Agile methods

- Emphasis on speed of delivery rather than documentation
- RAD – Rapid application development emphasised use of quickly developed prototypes
- SCRUM (Unit 4)

Outline

- Frameworks for software project planning

- Selection of software project approaches
  - Analysing other project characteristics
  - **Development process models**

- Effort estimation for software projects

- Activity planning and resource allocation

Choice of (development) process models

- Several widely used development process **models**
  - ‘**waterfall**’ model /sequential / also known as ‘one-shot’, ‘once-through’
  - **prototyping** / evolutionary development model / ‘try-feedback-refine’, ‘learn by doing’
  - **incremental** delivery model /’build in pieces’
  - **agile** methods such as extreme programming and SCRUM / ‘iterative-collaborative-adaptive’

Waterfall versus agile

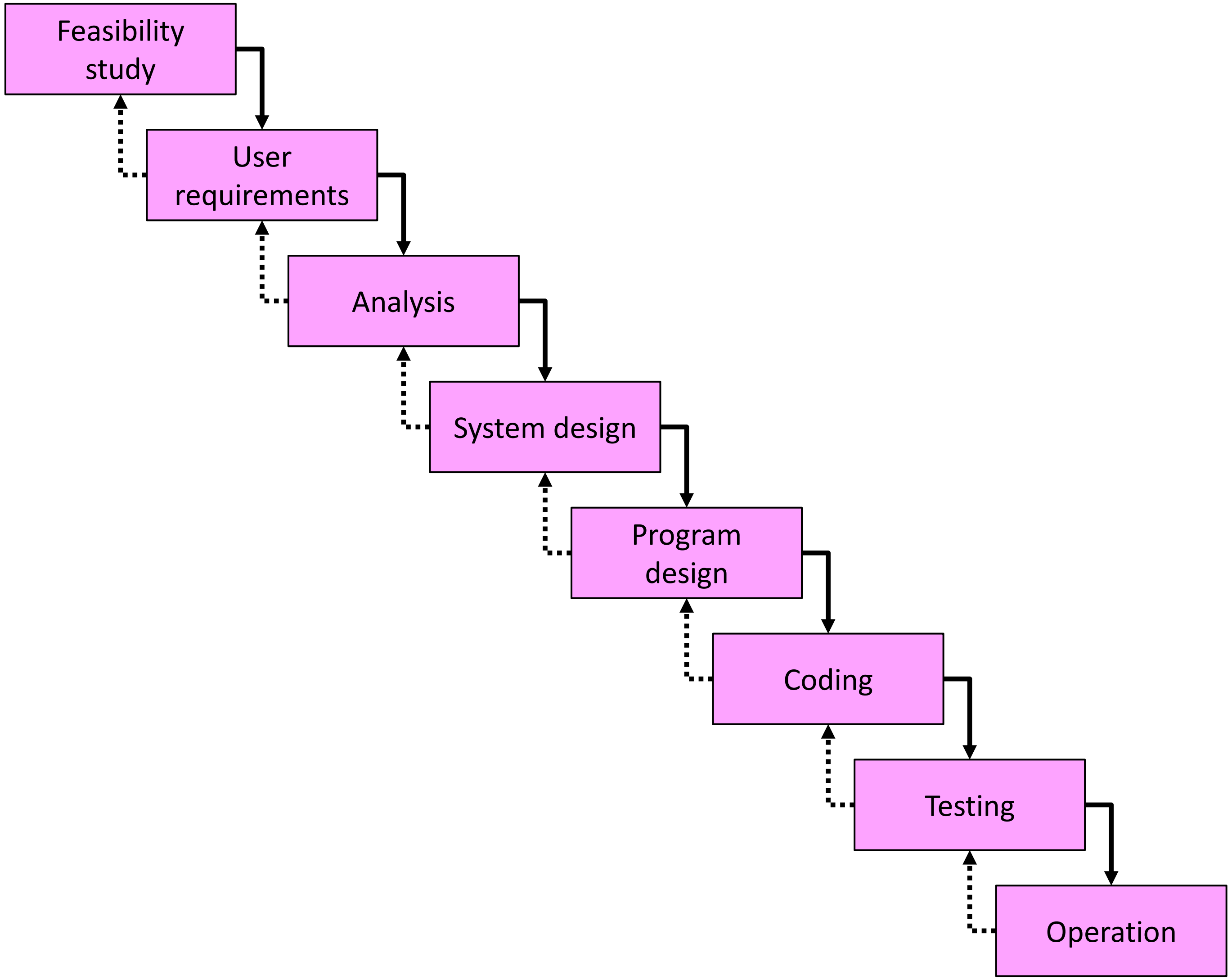
Predictive approach

- "Plan the work, work the plan"
- Everything must be planned
- More control and visibility

Empiric approach

- "Fail fast, fail safe"
- Based on experience
- More flexibility and learning

The waterfall model



- The 'classical' model
- Imposes structure
- Every stage signed off
- <u>Limited scope for iteration</u>

Works well when end-user requirements are clearly defined and stable - no change throughout the project

Waterfall advantages

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model; each phase has specific deliverables and a review process
- Phases are processed and completed one at a time
- Works well for smaller projects where requirements are very well-understood
- Clearly-defined stages
- Well-understood milestones
- Easy-to-arrange tasks
- Process and results are well-documented

Waterfall disadvantages

- No working software is produced until late during the life cycle
- High amounts of risk and uncertainty
- Poor model for long and ongoing projects
- Not suitable for the projects where requirements are at a moderate to high risk of changing; risk and uncertainty are high with this process model
- It is difficult to measure progress within stages
- Cannot accommodate changing requirements
- Adjusting scope during the life cycle can end a project
- Integration is done as a big-bang at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early

Projects suited for agile delivery

Aston University
BIRMINGHAM UK

- The most appropriate projects for agile are ones with aggressive deadlines, high degree of complexity, and high degree of novelty (uniqueness) to them

- Novelty?

- Urgency

- Complexity

Focus is on delivering the highest business value in the shortest time

Agile methods tackling disadvantages

- Agile aims to address several disadvantages of structured development methods:
  - **large amounts of documentation** which can be largely unread
  - need to keep **documentation up to date**
  - **communication stifled** by division into specialist groups and need to follow procedures
  - **user exclusion** from the decision process
  - **long lead times** to deliver anything
  - etc.

Extreme programming (XP)

A type of agile development process

Frequent refactoring
Pair programming
Test-driven development (TDD)
Continuous integration

- Characteristics:
  - Increments of one to three weeks
  - Customer(end users) can suggest improvement at any point
  - Elimination of distinction between design and building of software
  - Code developed to meet current needs only
  - Frequent refactoring to keep code structured
  - Developers work in pairs
  - Test cases and expected results devised *before* software design
  - After testing of increment, test cases added to a consolidated set of test cases – reused as part of continuous integration

Limitations of extreme programming (XP)

- Reliance on availability of **high-quality developers** (e.g. experience, good technical skills)

- **Dependence on personal knowledge** – after development, knowledge of software may decay making future development difficult

- **Rationale for decisions may be lost**, e.g., which test case checks a particular requirement

- **Reuse of existing code less likely**

Prototyping and evolutionary methods

- Main types
  - **‘throw away’ prototypes** – used to learn about an area of uncertainty
  - **evolutionary prototypes** – the prototype is developed and modified until it becomes the operational system
- What is being prototyped?
  - human-computer interface – front end
  - elements/subsets of functionality – back end

Prototype terms – key points

- **Prototype**
  - a model for acquiring requirements
  - a working model built to learn how a work system could operate
  - like physical models in many engineering applications

- **Throwaway Prototype**
  - designed to be discarded
  - suitable when the final design is unclear; useful for comparing alternatives

- **Evolutionary Prototype**
  - designed to be adapted for normal use
  - suitable when the final design is clear
  - built using the intended platform

Reasons for prototyping

- Learning by doing
- Improved communication
- Improved user involvement
- A feedback loop is established
- Reduces the need for documentation
- Reduces maintenance costs due to changes before the application goes live
- Prototype can be used for producing expected results

Dangers of prototyping

- Users may misunderstand the role of the prototype
- Their expectations may get too high
- Lack of project control and standards
- Additional expense of building prototype
- Focus on usable interface could be at expense of machine efficiency

Incremental delivery

- The application to be delivered is broken down into a number of components called **lots** that provide **immediate value** to the customers

- Each component is developed as a separate **increment**

Incremental - pros and cons

- Several important advantages:
  - feedback from early stages used in developing later stages
  - easier to cope with changing requirements
  - user gets some benefits earlier
  - project may be put aside temporarily

- BUT there are some possible disadvantages:
  - loss of economy of scale – some costs will be repeated
  - software breakage – later increments may change earlier increments

Typical increment plan

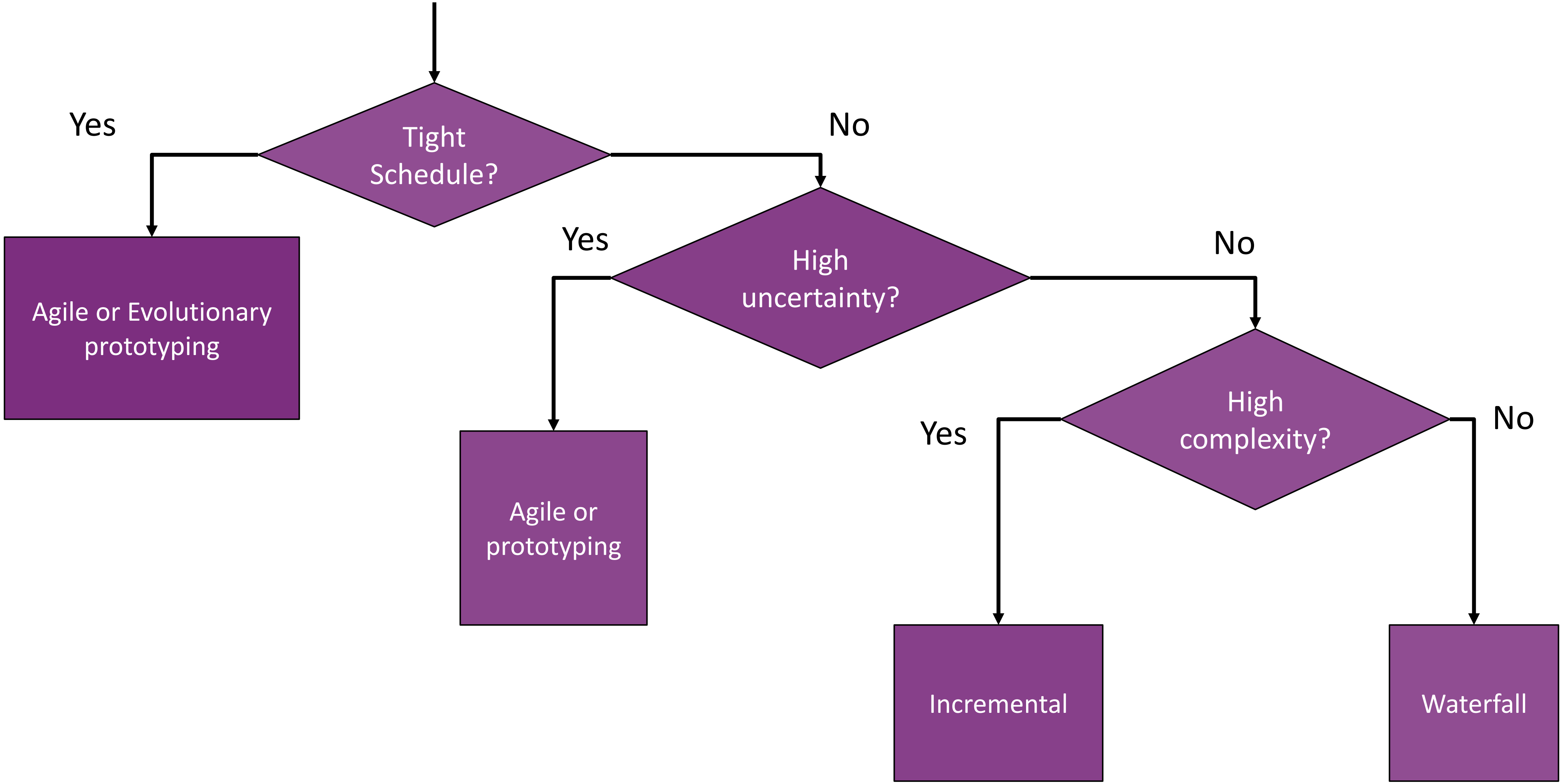
- Each lot should deliver some benefits to the user

- Some lots will be dependent on others, hence there is a natural ordering of these lots

Choosing a model

- If uncertainty is high:
  - Agile or prototyping
- If complexity is high but uncertainty is not:
  - Incremental delivery
- If uncertainty and complexity are both low:
  - Waterfall
- If schedule is tight:
  - Agile or evolutionary prototype

Choosing a model for SW development

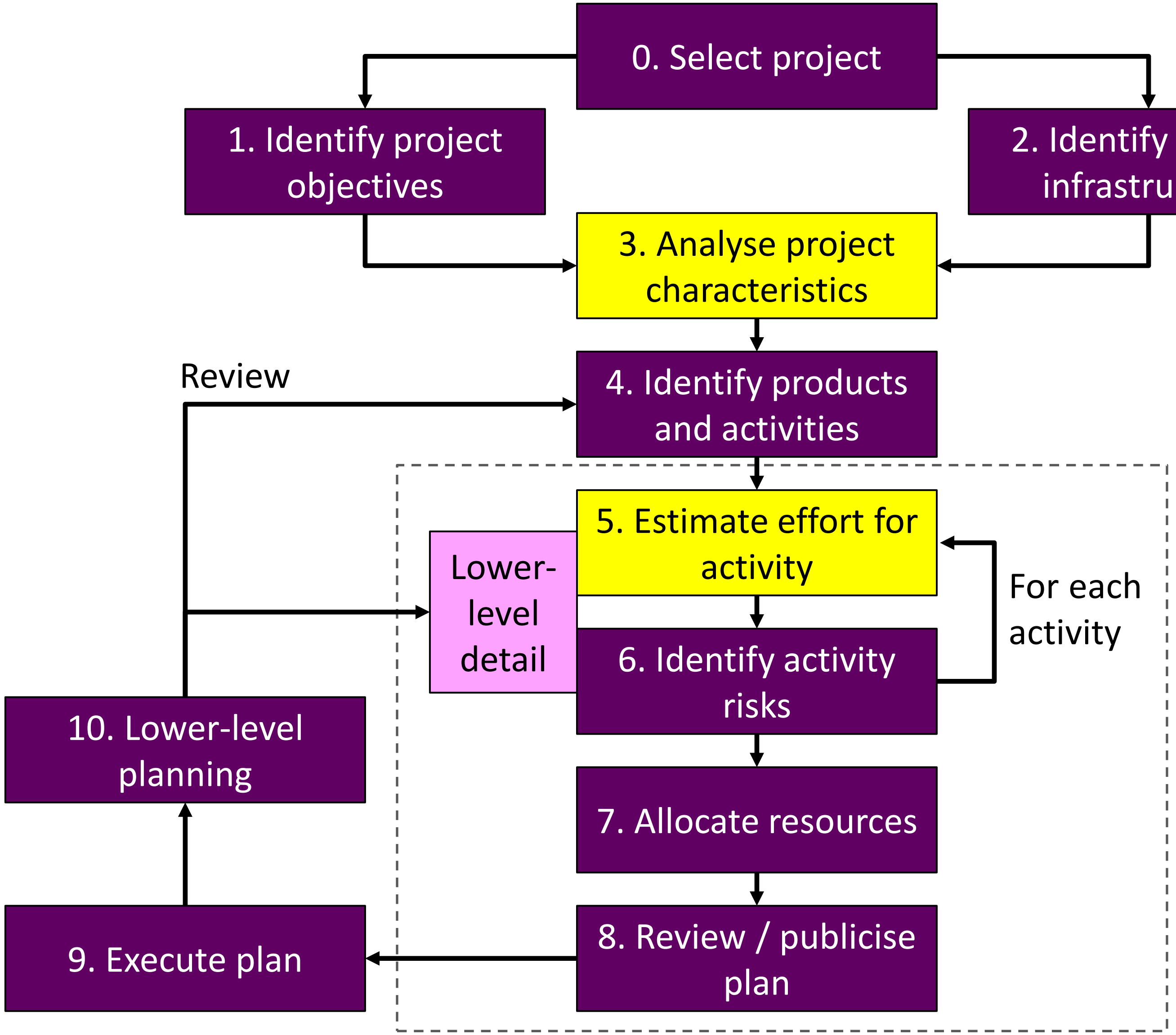


Outline

- Frameworks for software project planning
- Selection of software project approaches
- **Effort estimation for software projects**
- Activity planning and resource allocation

Steps 3 and 5 of Step Wise - Effort

***Effort**
time,
resources,
and cost


0. Select project
1. Identify project objectives
2. Identify project infrastructure
3. Analyse project characteristics
Review
4. Identify products and activities
5. Estimate effort for activity
Lower-level detail
For each activity
6. Identify activity risks
10. Lower-level planning
7. Allocate resources
9. Execute plan
8. Review / publicise plan

<u>Goal</u>:
accurate and realistic estimates

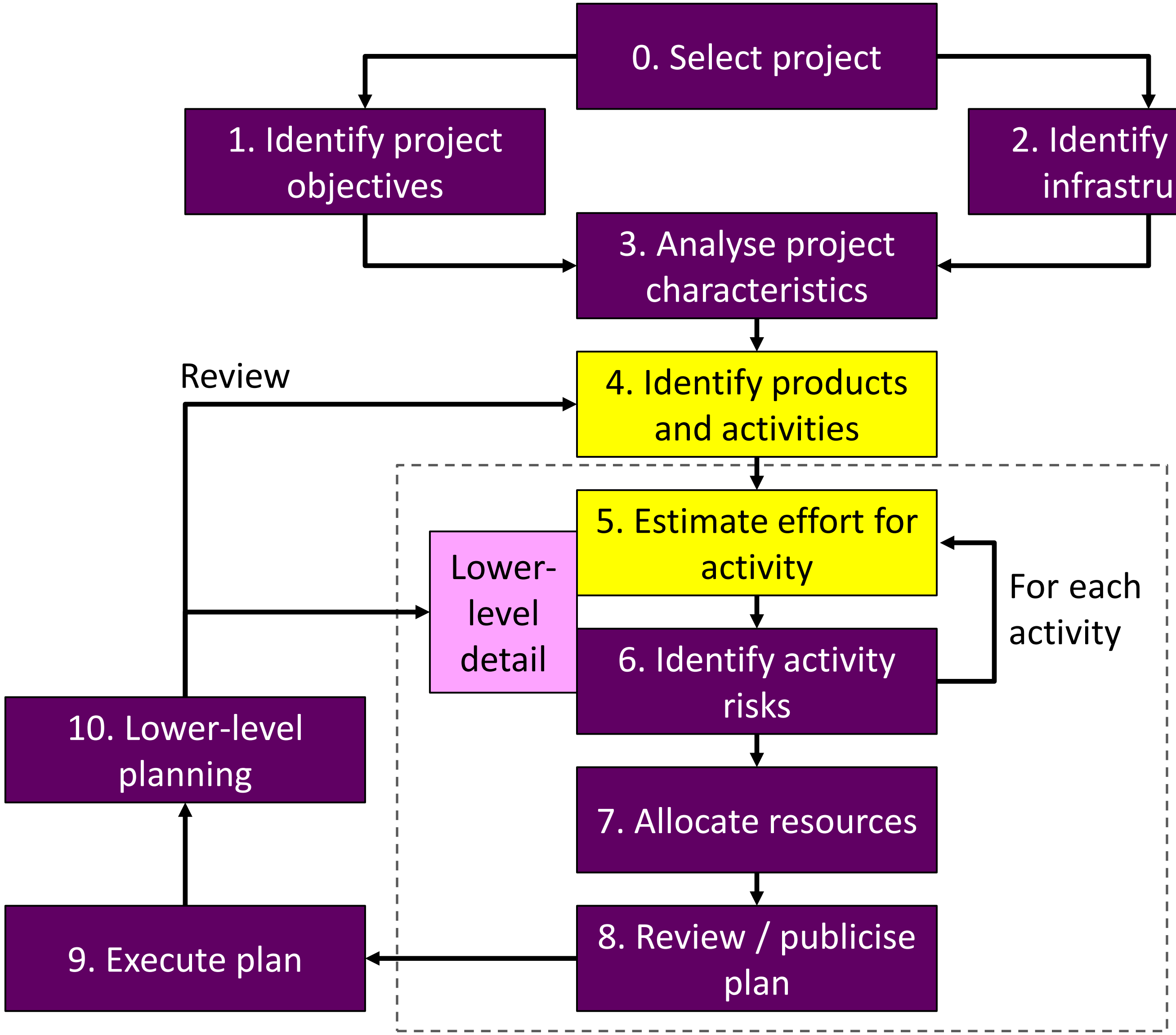
- Size
- Dependencies
- Skill levels

Outline

- Frameworks for software project planning
- Selection of software project approaches
- Effort estimation for software projects
- Activity planning and resource allocation
  - Activity planning
  - Resource allocation

Steps 4 and 5 of Step Wise

Aston University BIRMINGHAM UK


0. Select project
1. Identify project objectives
2. Identify project infrastructure
3. Analyse project characteristics
Review
4. Identify products and activities
5. Estimate effort for activity
Lower-level detail
For each activity
6. Identify activity risks
7. Allocate resources
8. Review / publicise plan
10. Lower-level planning
9. Execute plan

What needs to be planned?

- Scope
- Tasks

What is the effort required?

Planning your project

- **Objective: produce a <u>schedule</u> that indicates <u>start</u> and <u>completion</u> times for each project activity**

The <u>WHEN/WHO</u> for each activity

- **This enables:**
  - Ensuring that appropriate resources will be available when required
  - Avoiding resource overload
  - Derivation of a timed cash flow forecast
  - Project re-planning to correct drift from the target

Resources ready when needed

When <u>money</u> will be needed?

Schedule allows adjusting the plan when needed

Approaches (3)

- **Activity-based**
  - Draw-up a Work Breakdown Structure listing the work items (activities) needed based on an analysis of similar past products

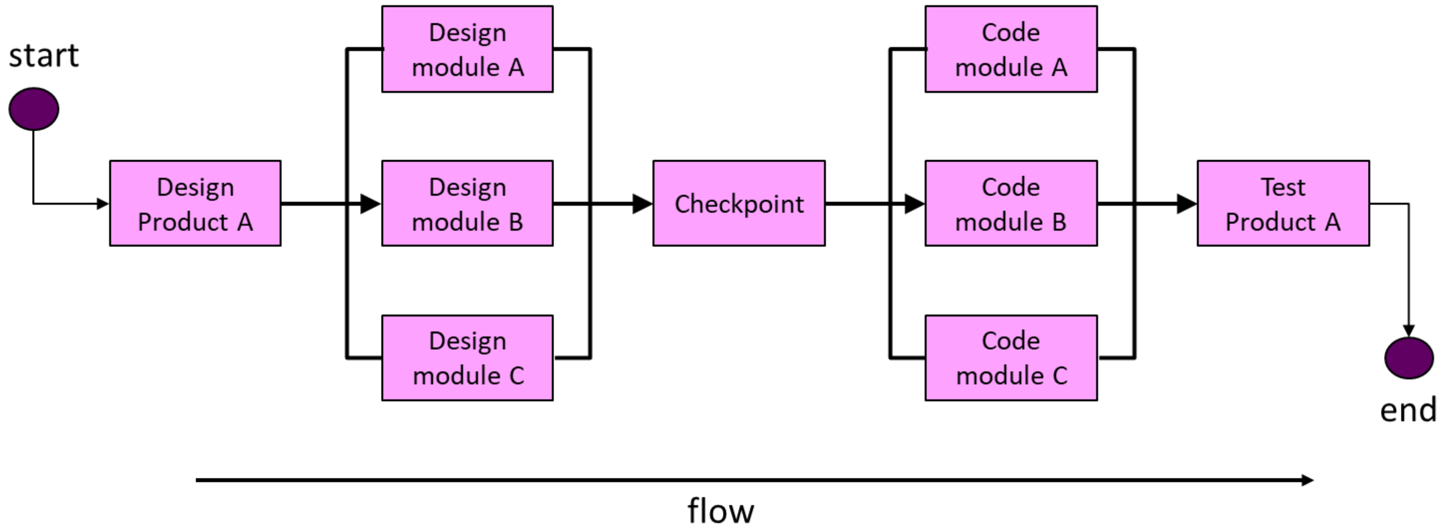
- **Product-based***
  - Used in PRINCE2 and Step Wise
  - **List** the deliverable and intermediate products of project – **Product Breakdown Structure (PBS)**
  - **Identify the order** in which products have to be created – **Product Flow Diagram (PFD)**
  - **Work out the activities** needed to create the products – **e.g. creating Activity Networks**

- **Hybrid**
  - start from a simple list of final deliverables (which are product specific) and apply the **activity-based approach to each deliverable**

Activity networks

- These networks help us to:
  - Assess the feasibility of the planned project completion date
  - Identify when resources will need to be deployed to activities
  - Calculate when costs will be incurred
  - Globally, co-ordinate and motivate of the project team

An activity network: activities order and flow (arrows)

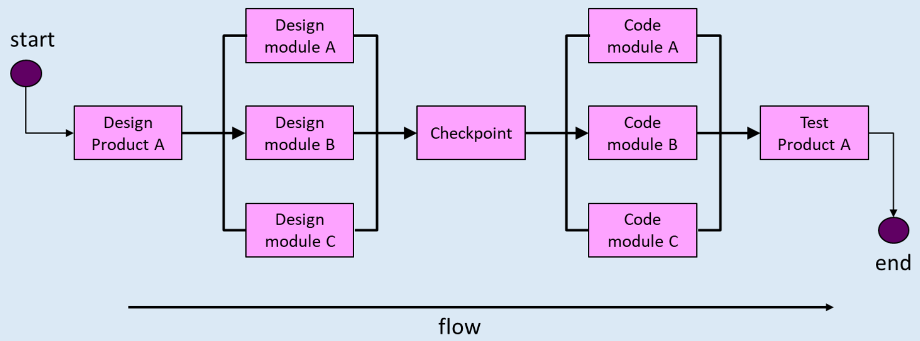


Activity networks

- These help us to:
  - Assess the feasibility of the planned project completion date
  - Identify when resources will need to be deployed to activities
  - Calculate when costs will be incurred
  - Globally, co-ordinate and motivate of the project team

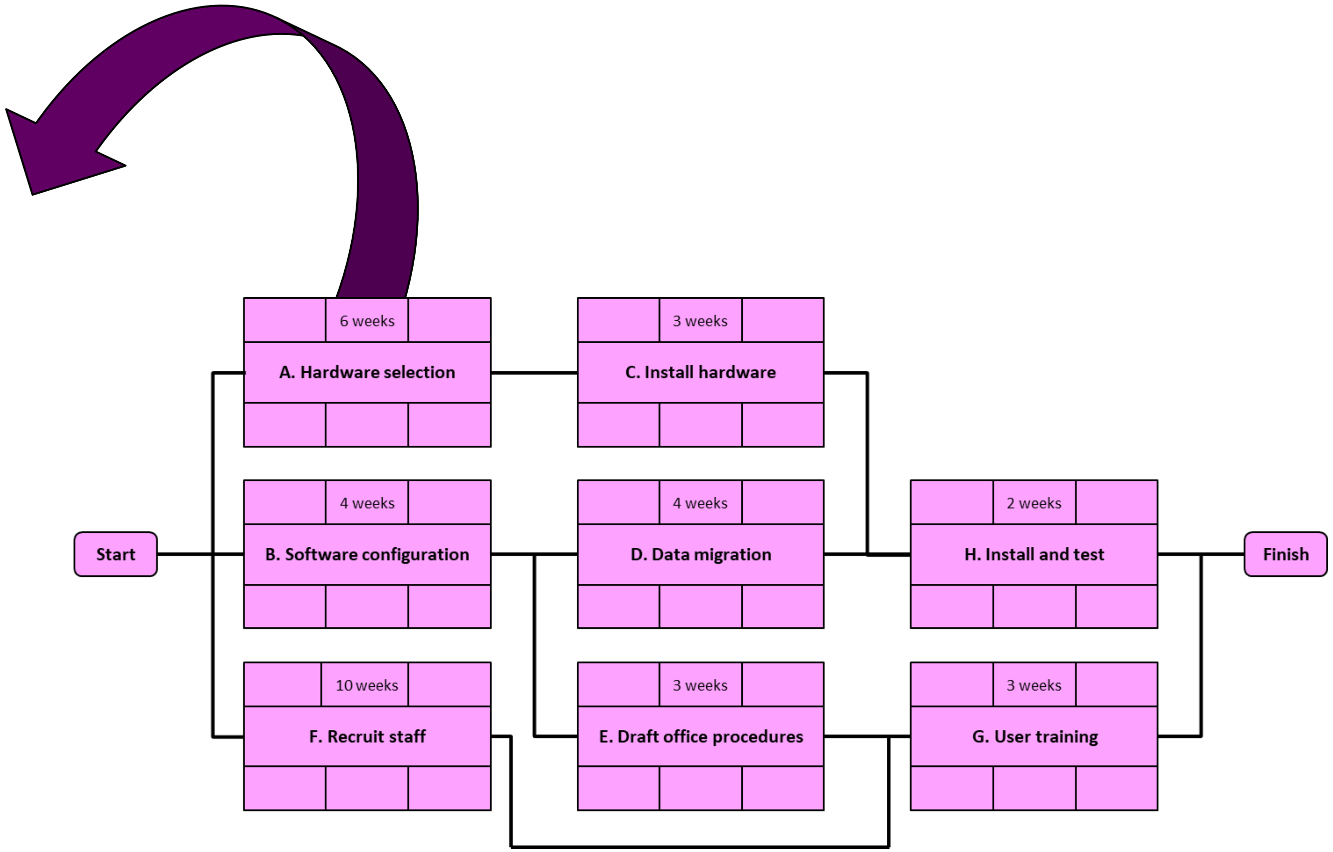
- Assumptions:
  - Each project is composed of a number of activities
  - Can start when at least one activity is ready to start
  - Is completed when all activities are completed

An activity network: activities order and flow (arrows)



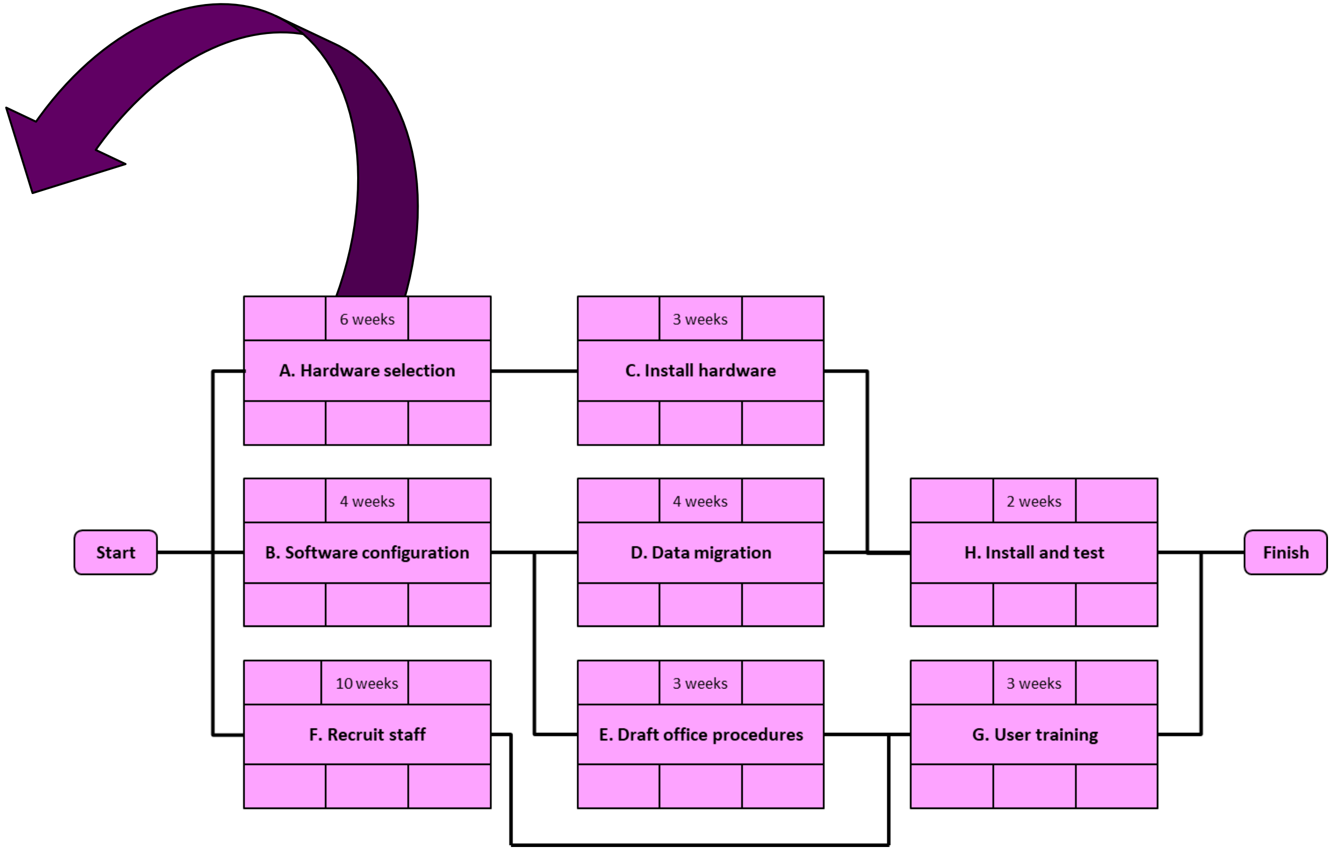
Labelling convention

Earliest start	Duration	Earliest finish
Activity label, activity description		
Latest start	Float	Latest finish



Labelling convention

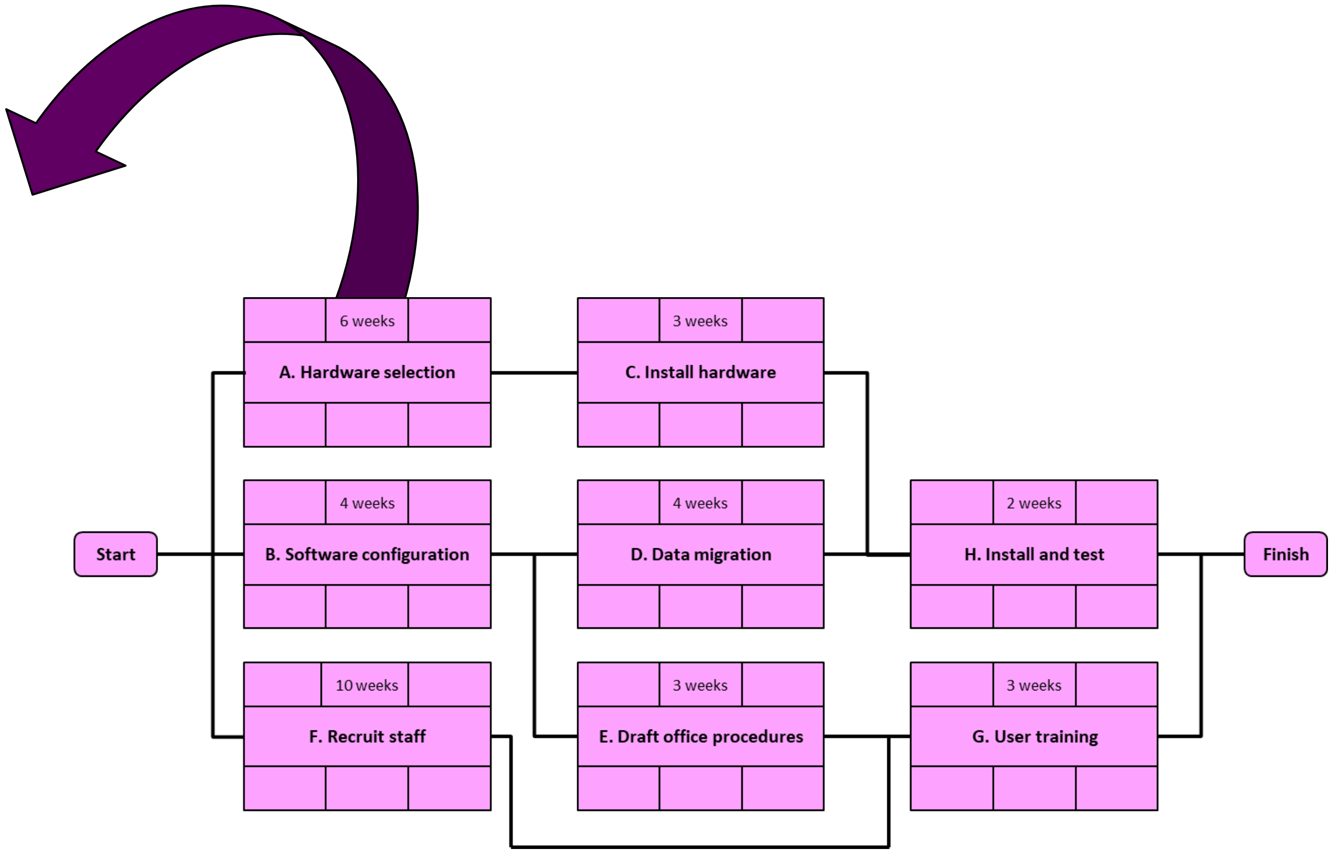
Earliest start	Duration	Earliest finish
Activity label, activity description		
Latest start	Float	Latest finish



Identify **what** the activity is

Labelling convention

Earliest start	Duration	Earliest finish
Activity label, activity description		
Latest start	Float	Latest finish

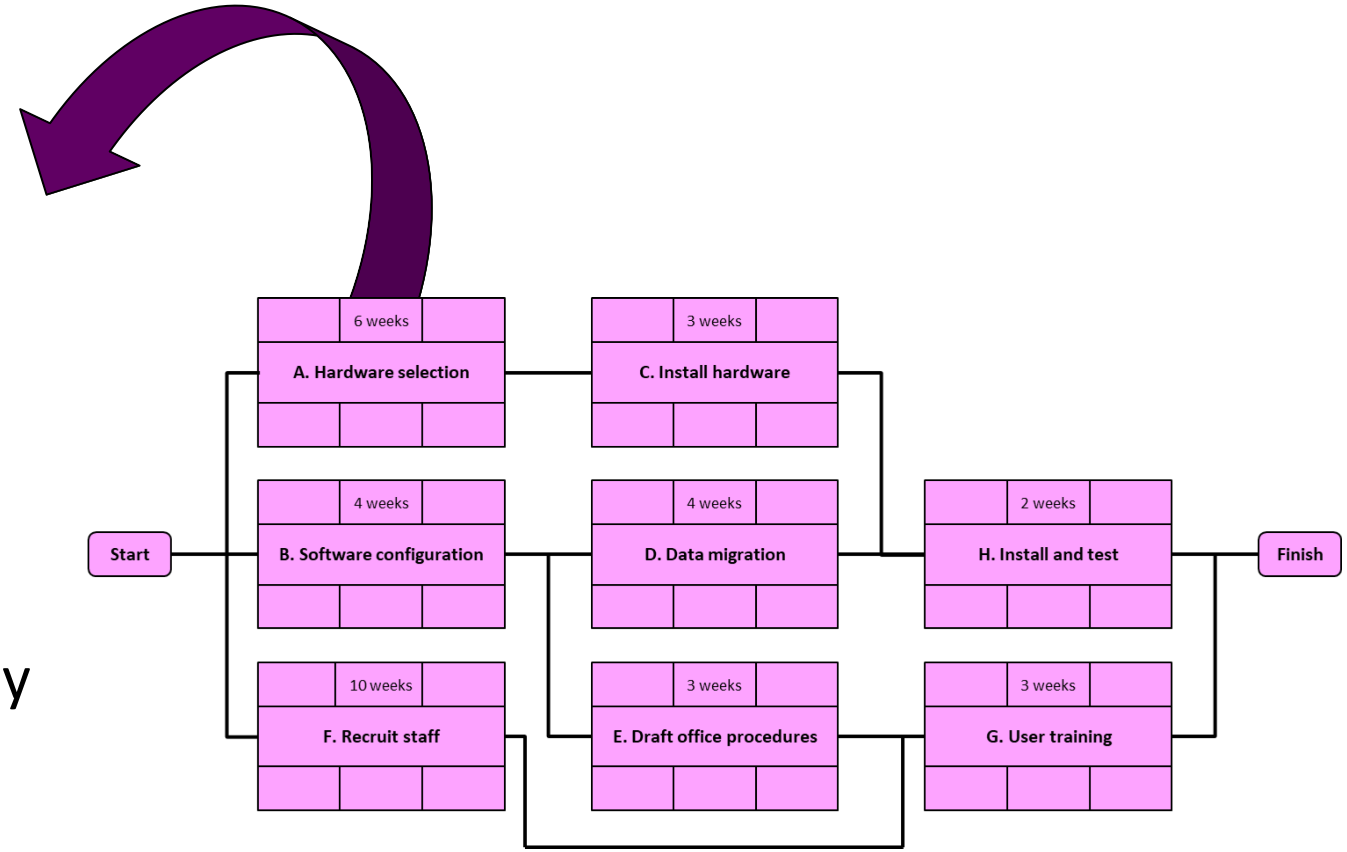


earliest time the activity can begin after the required tasks before it are finished

Labelling convention

Earliest start	**Duration**	Earliest finish
Activity label, activity description		
Latest start	Float	Latest finish

The **time** required **to complete** the activity

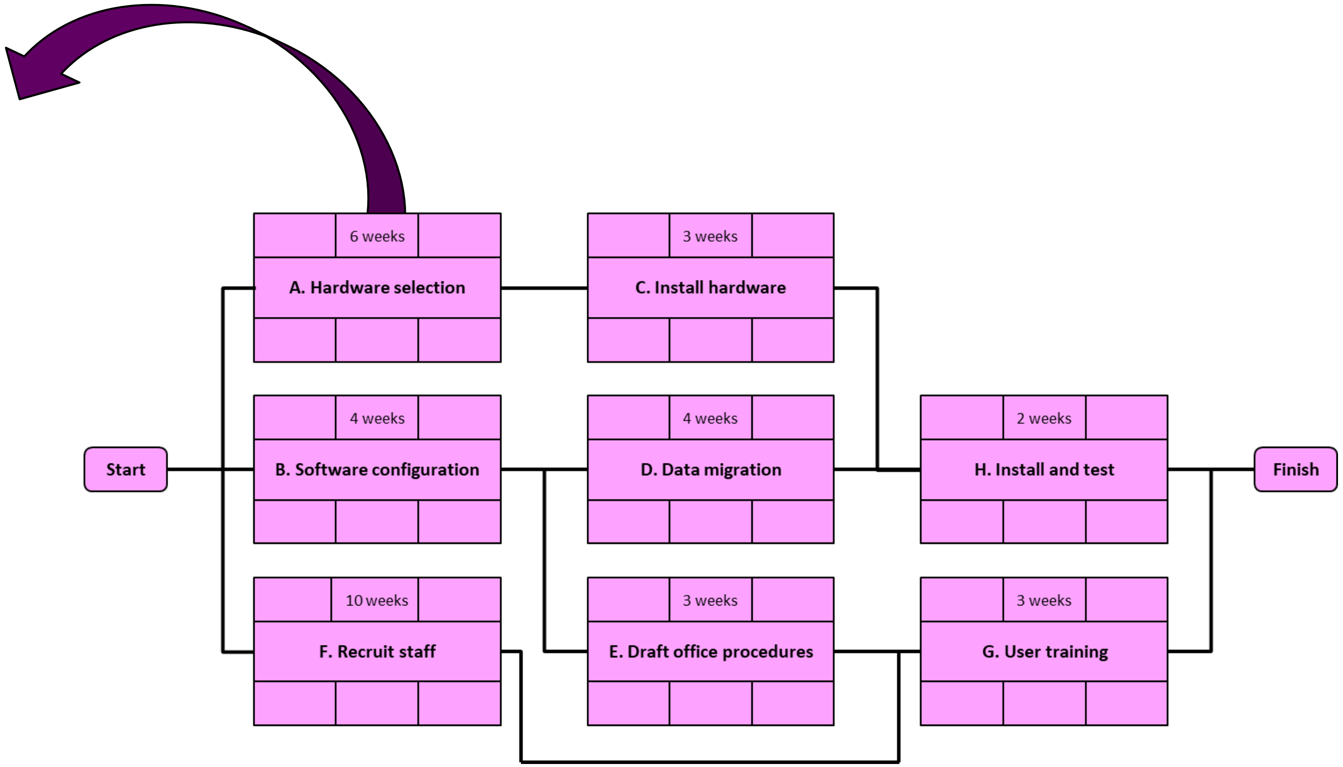


Labelling convention

Earliest start	Duration	Earliest finish
Activity label, activity description		
Latest start	Float	Latest finish

Earliest finish =
Earliest Start + Duration

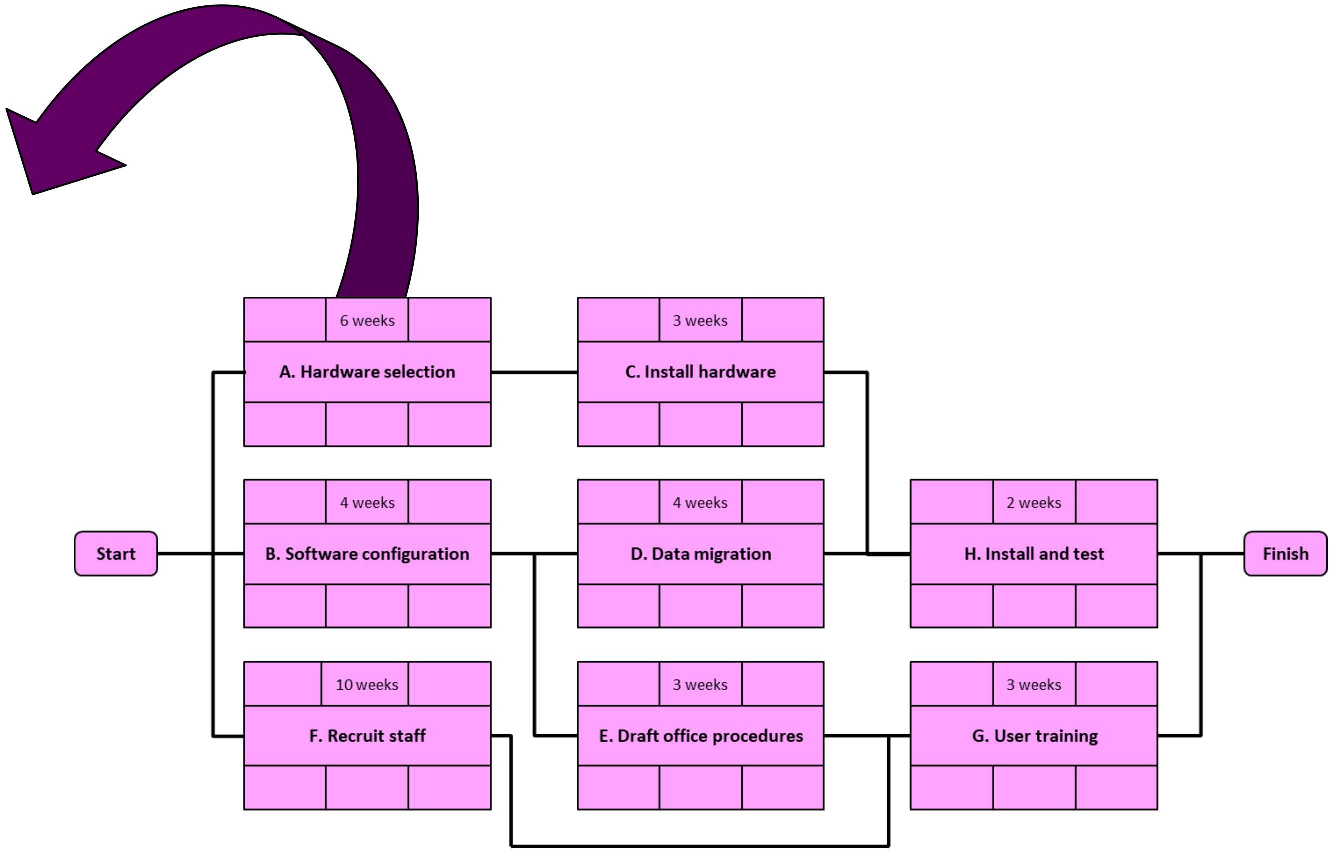
The **earliest time to finish** the activity



Labelling convention

Earliest start	Duration	Earliest finish
Week 0	**10 weeks**	**Week 10**
Activity label, activity description		
Latest start	Float	Latest finish

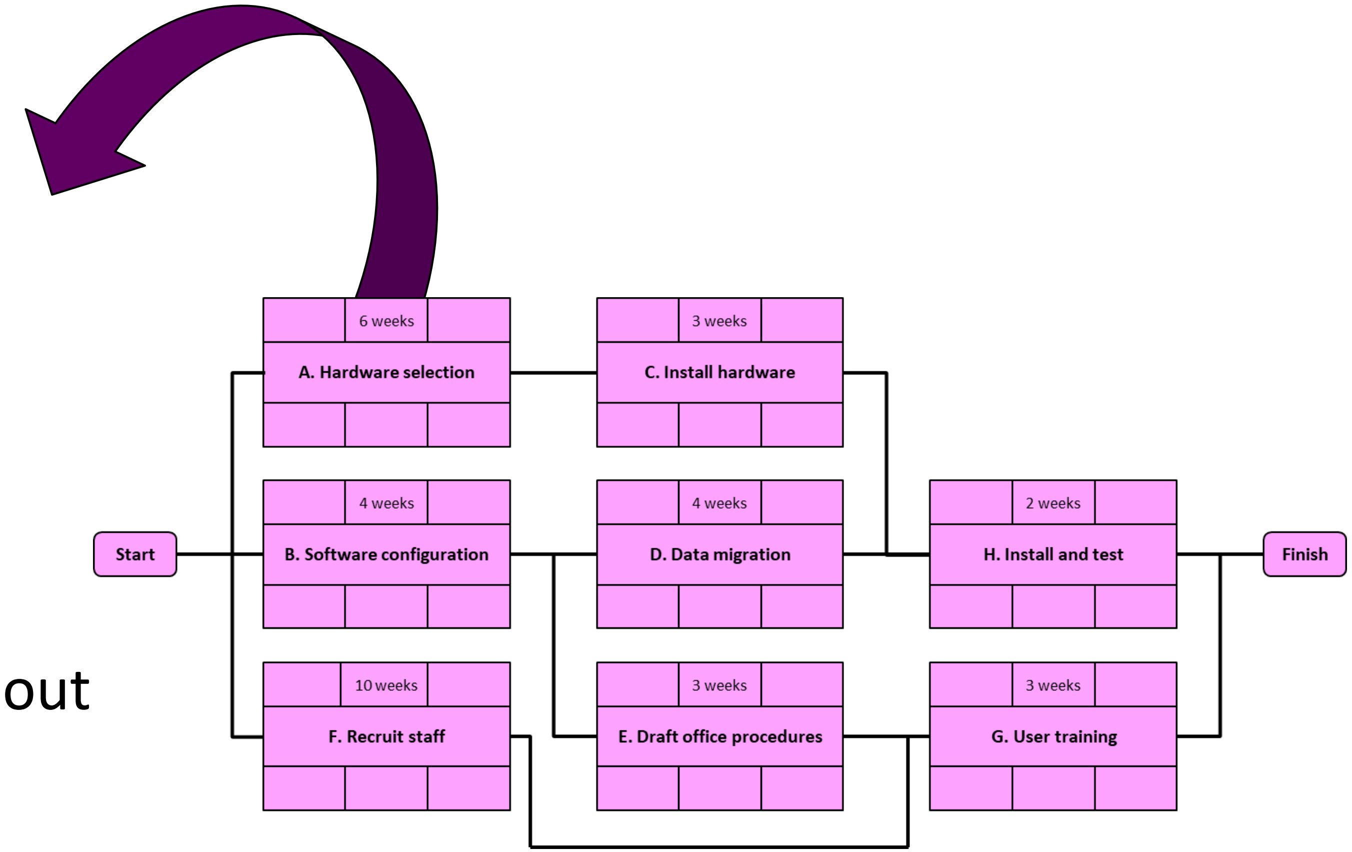
Earliest finish = Earliest Start + Duration



The **earliest time to finish** the activity

Labelling convention

Earliest start	Duration	Earliest finish
Activity label, activity description		
Latest start	Float	**Latest finish**

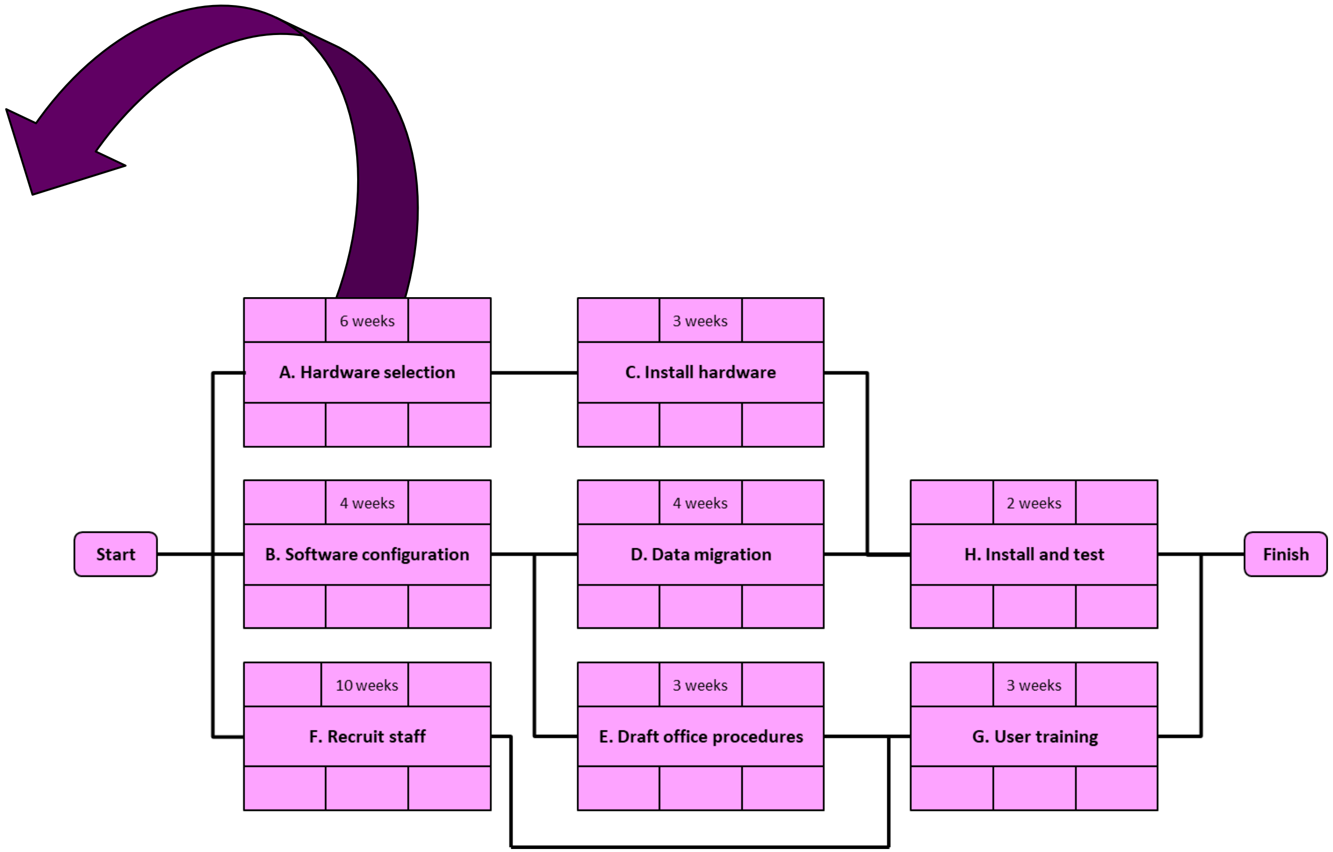


The **latest time** an activity **can finish** without impacting the overall project timeline

Labelling convention

Earliest start	**Duration**	Earliest finish
Activity label, activity description		
Latest start	Float	**Latest finish**

Latest start =
Latest finish - Duration



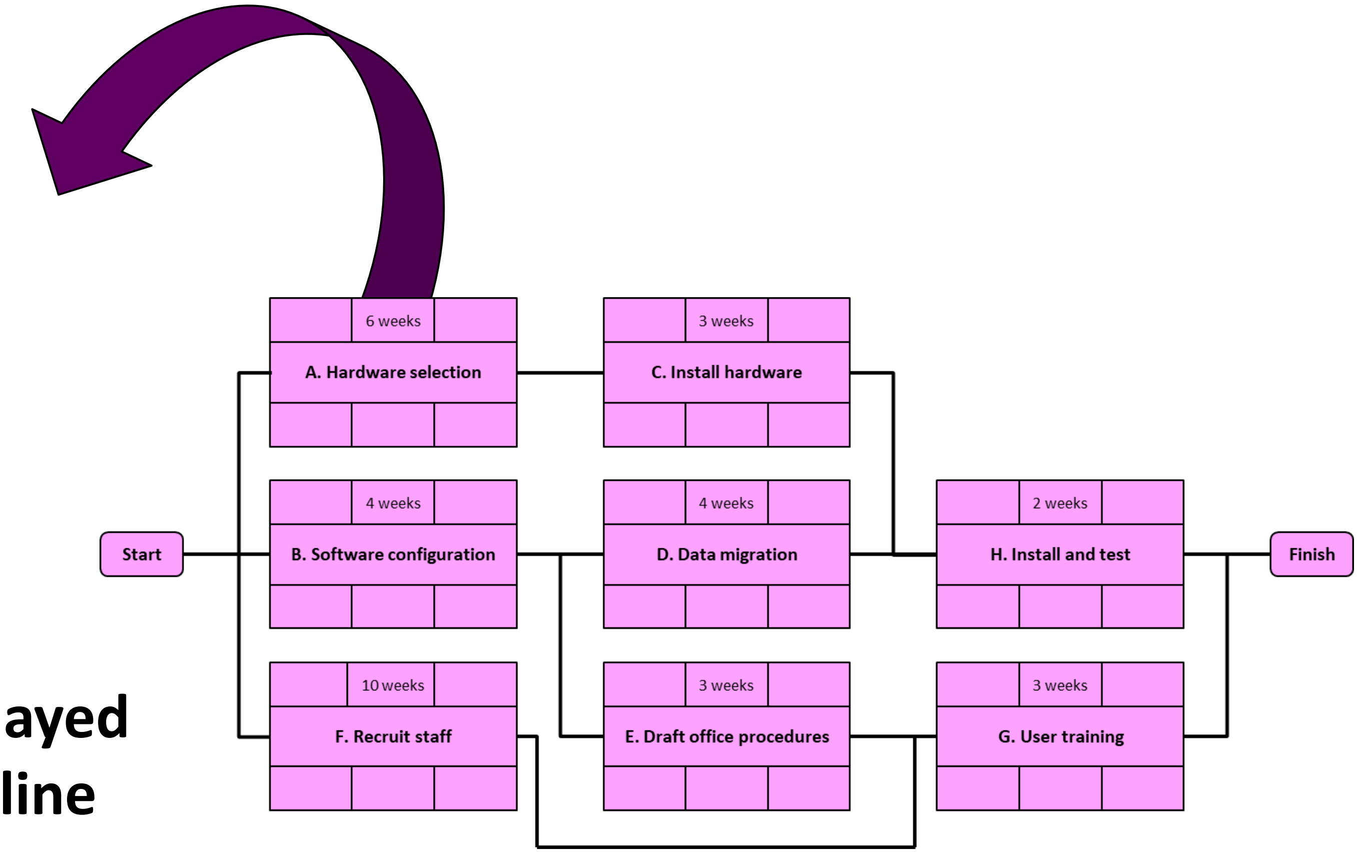
The **latest time** an activity **can begin** without delaying the project

Labelling convention

Earliest start	Duration	Earliest finish
Activity label, activity description		
Latest start	**Float**	**Latest finish**

Float is a calculated value

Float = latest finish - earliest finish



The **amount of time** an activity can be **delayed without affecting** the overall project **timeline**

Example activity network

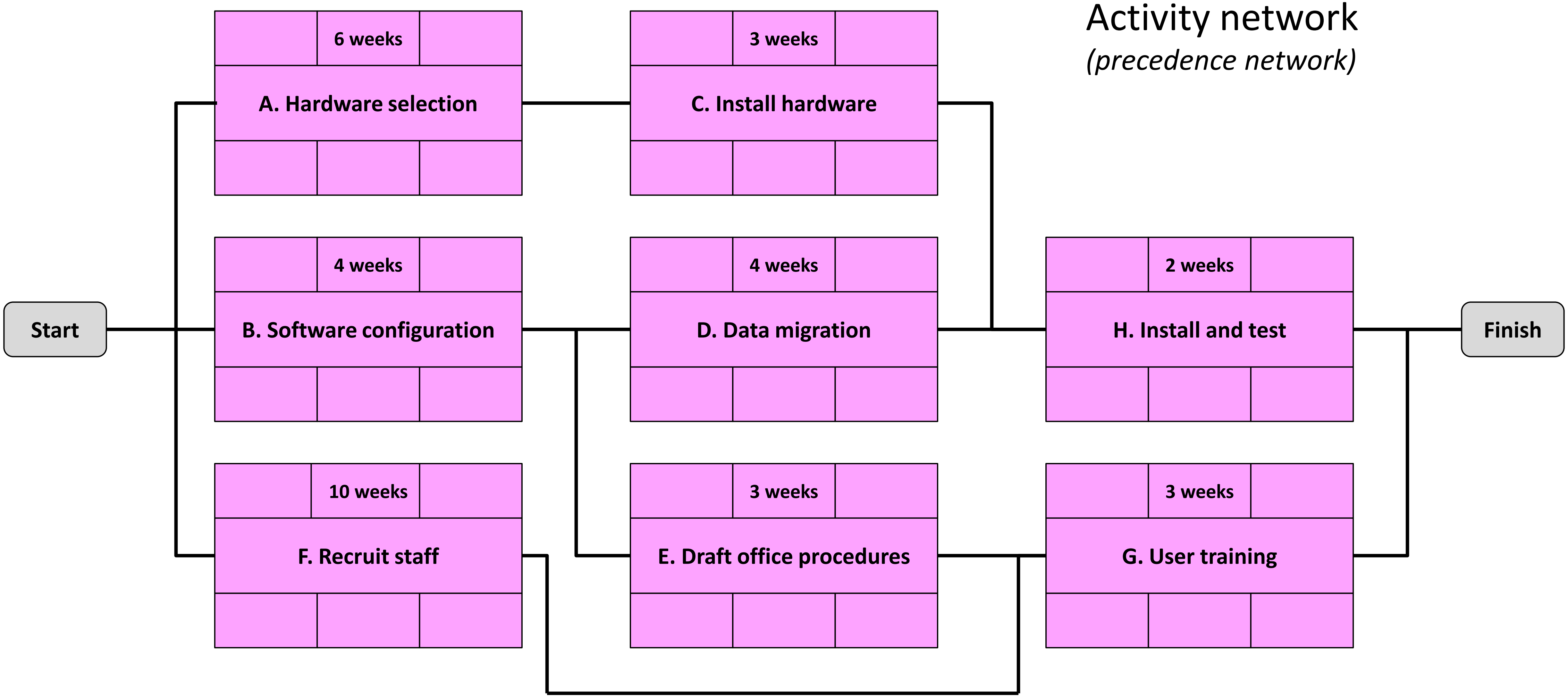
Project specification with estimated activity durations and dependencies

Activity	Description	Duration (Weeks)	Precedence requirements
A	Hardware selection	6	-
B	System configuration	4	-
C	Install hardware	3	A
D	Data migration	4	B
E	Draft office procedures	3	B
F	Recruit staff	10	-
G	User training	3	E, F
H	Install and test system	2	C, D

Example activity network

Activity network

(precedence network)



Example activity network

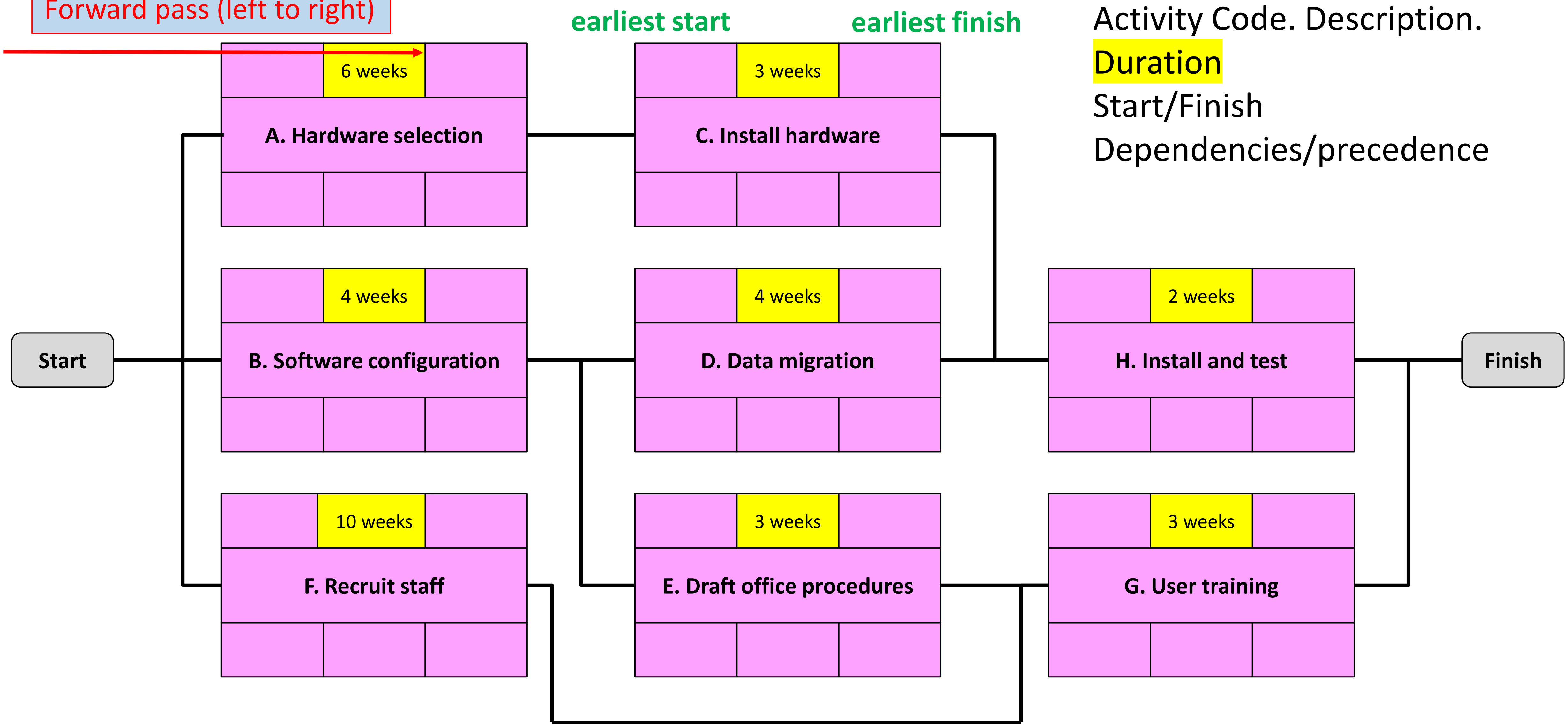
Forward pass (left to right)

Activity Code. Description.

Duration

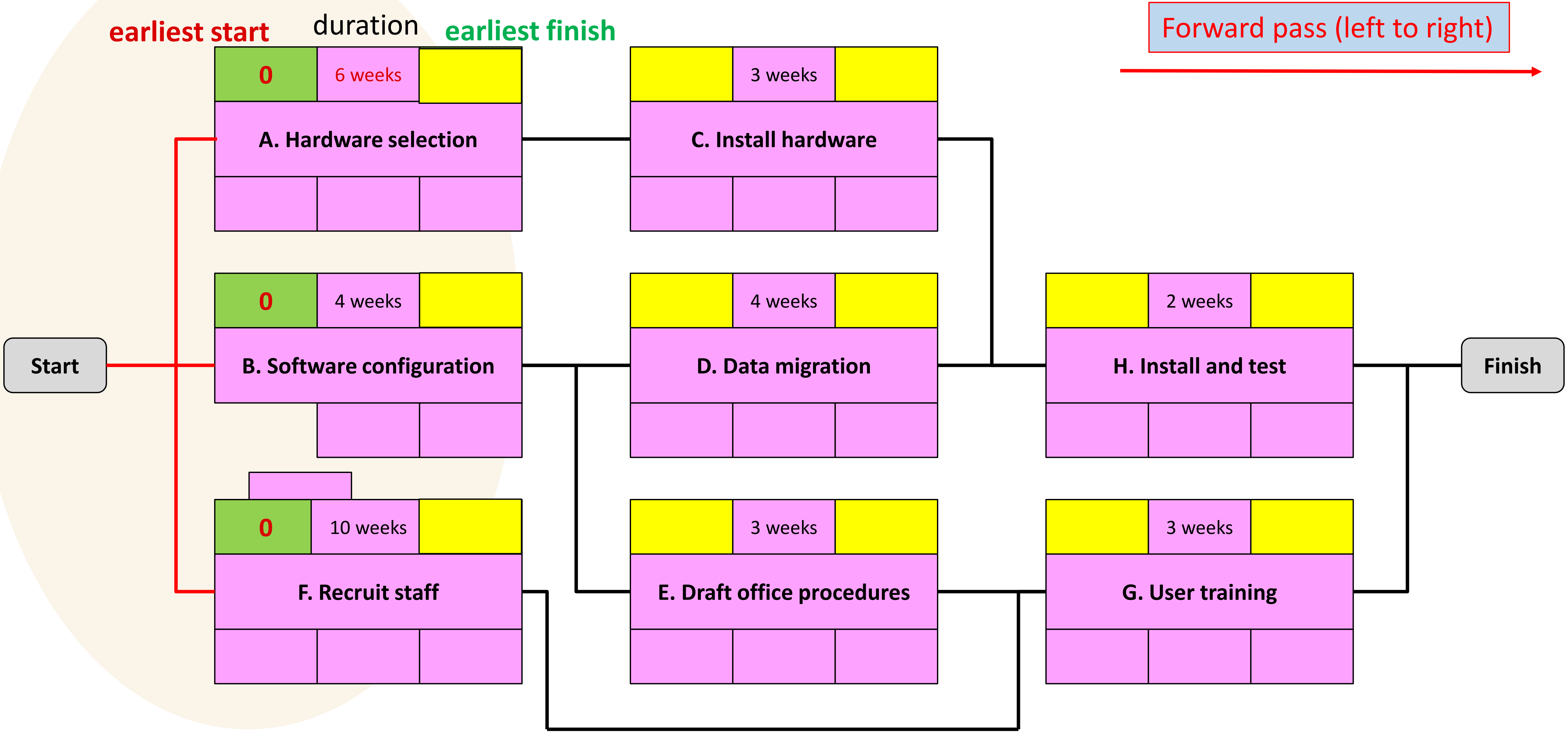
Start/Finish

Dependencies/precedence

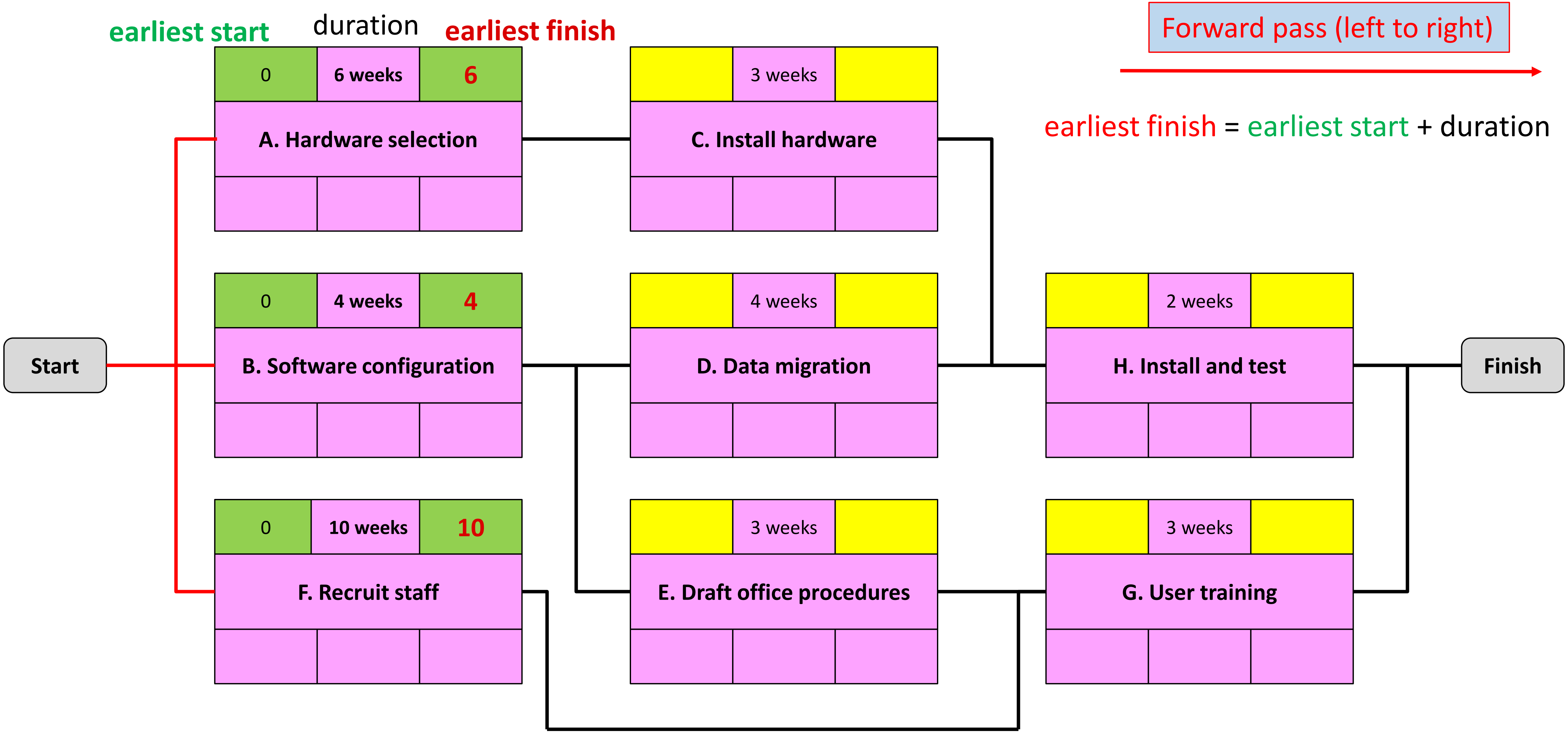


Earliest start/finish calculated

Forward pass (left to right)



Earliest start/finish calculated



Forward pass (left to right)

earliest finish = earliest start + duration

earliest start | duration | earliest finish

Start

0	6 weeks	6

A. Hardware selection

0	4 weeks	4

B. Software configuration

0	10 weeks	10

F. Recruit staff

3 weeks

C. Install hardware

4 weeks

D. Data migration

3 weeks

E. Draft office procedures

2 weeks

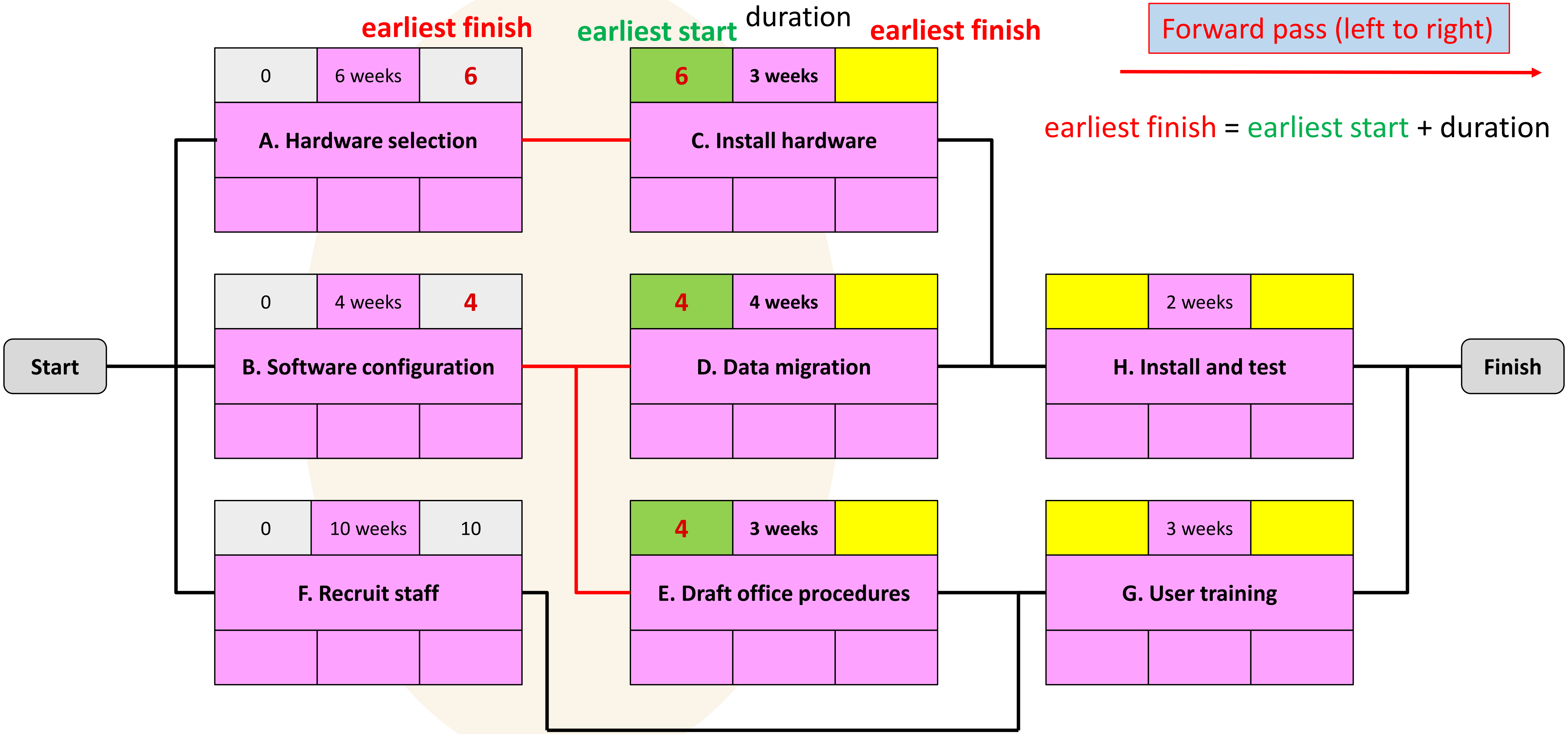
H. Install and test

3 weeks

G. User training

Finish

Earliest start/finish calculated



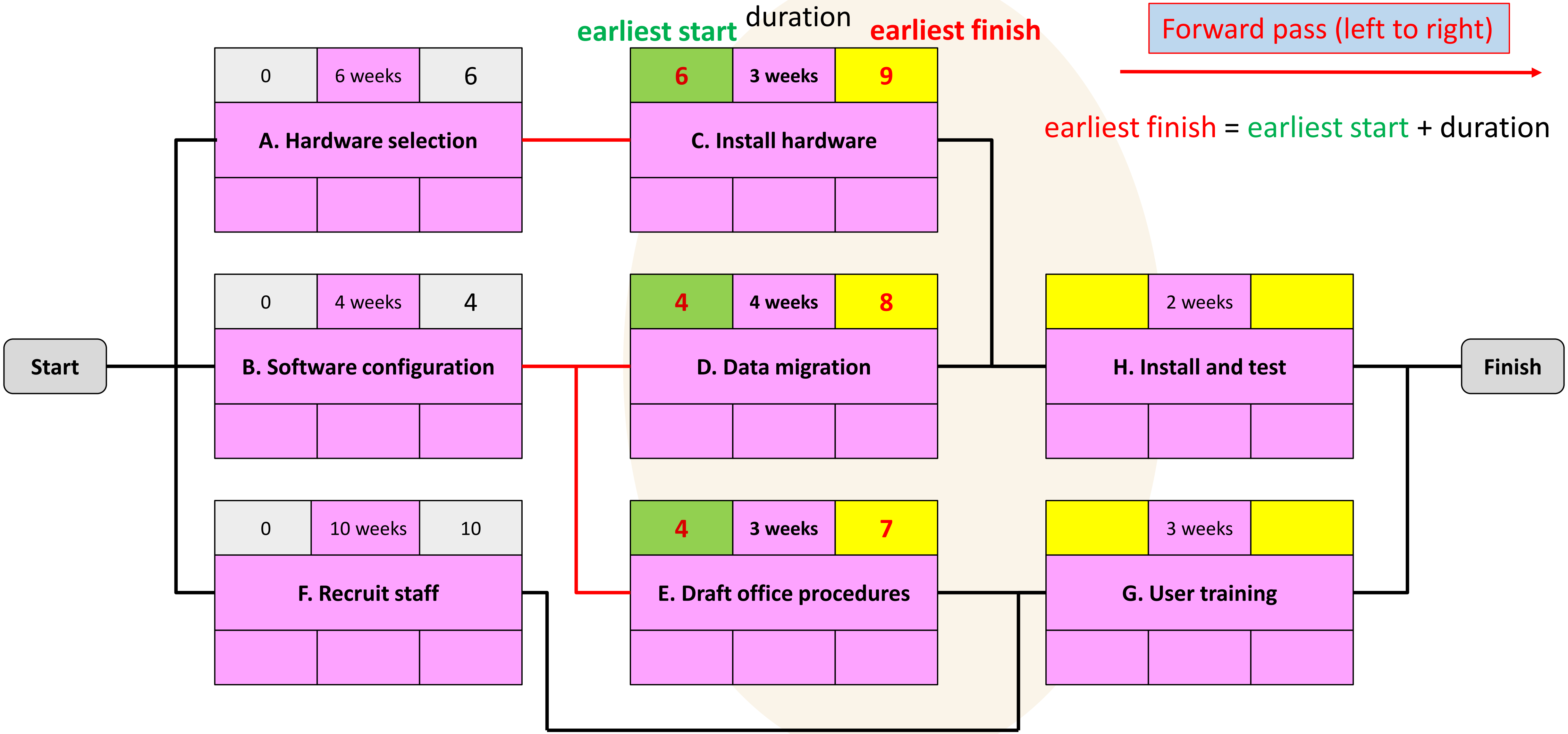
Forward pass (left to right)

earliest finish = earliest start + duration

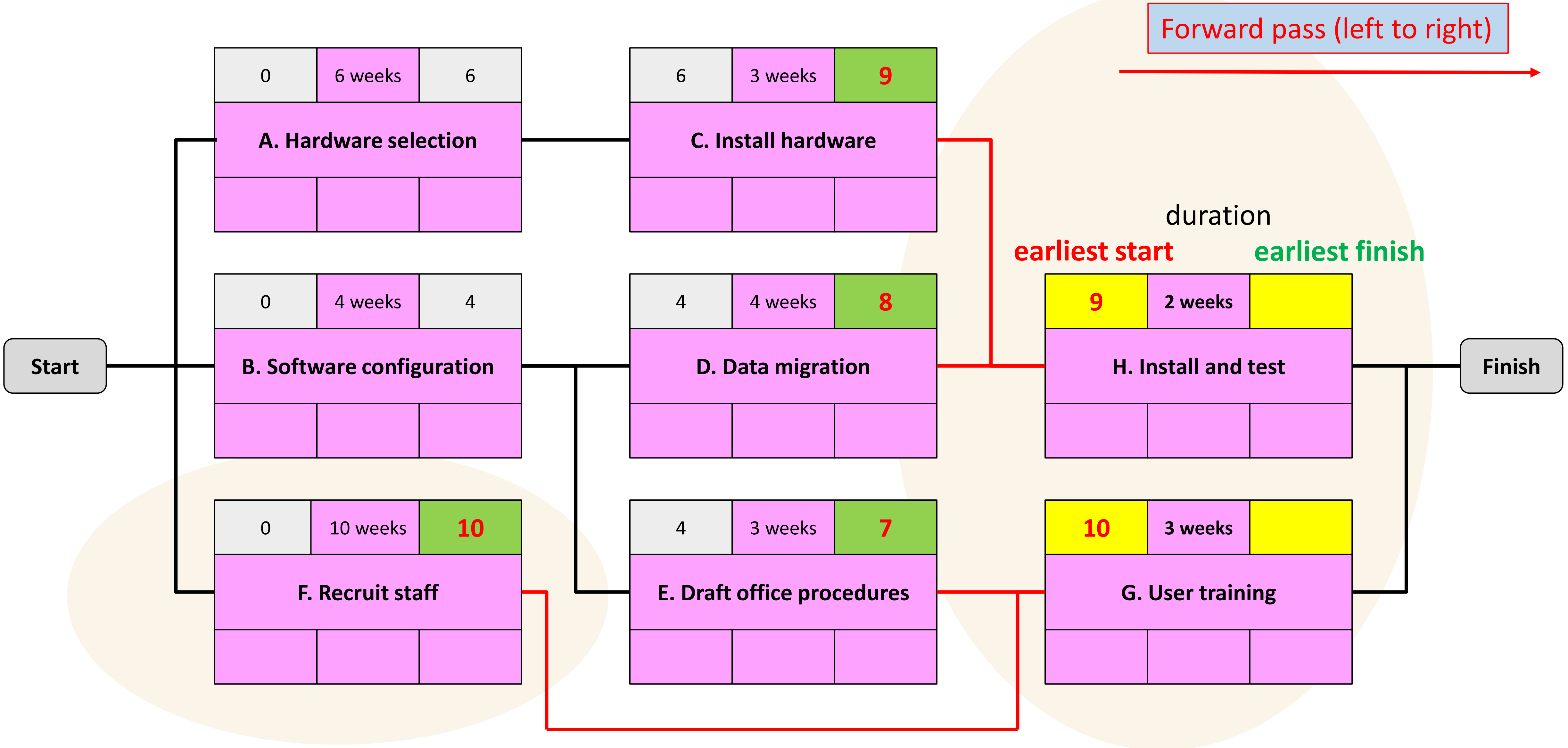
Earliest start/finish calculated

Forward pass (left to right)

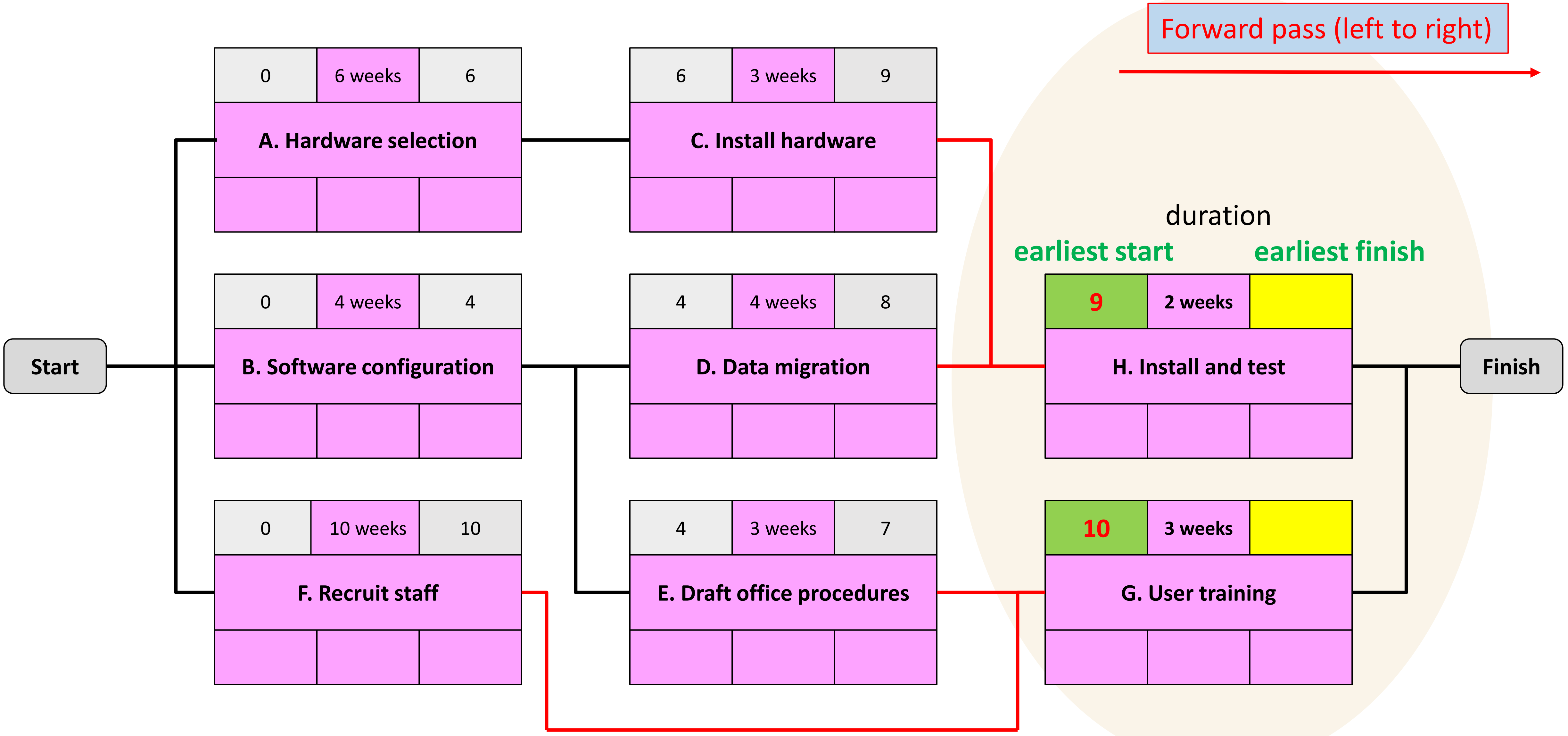
earliest finish = earliest start + duration



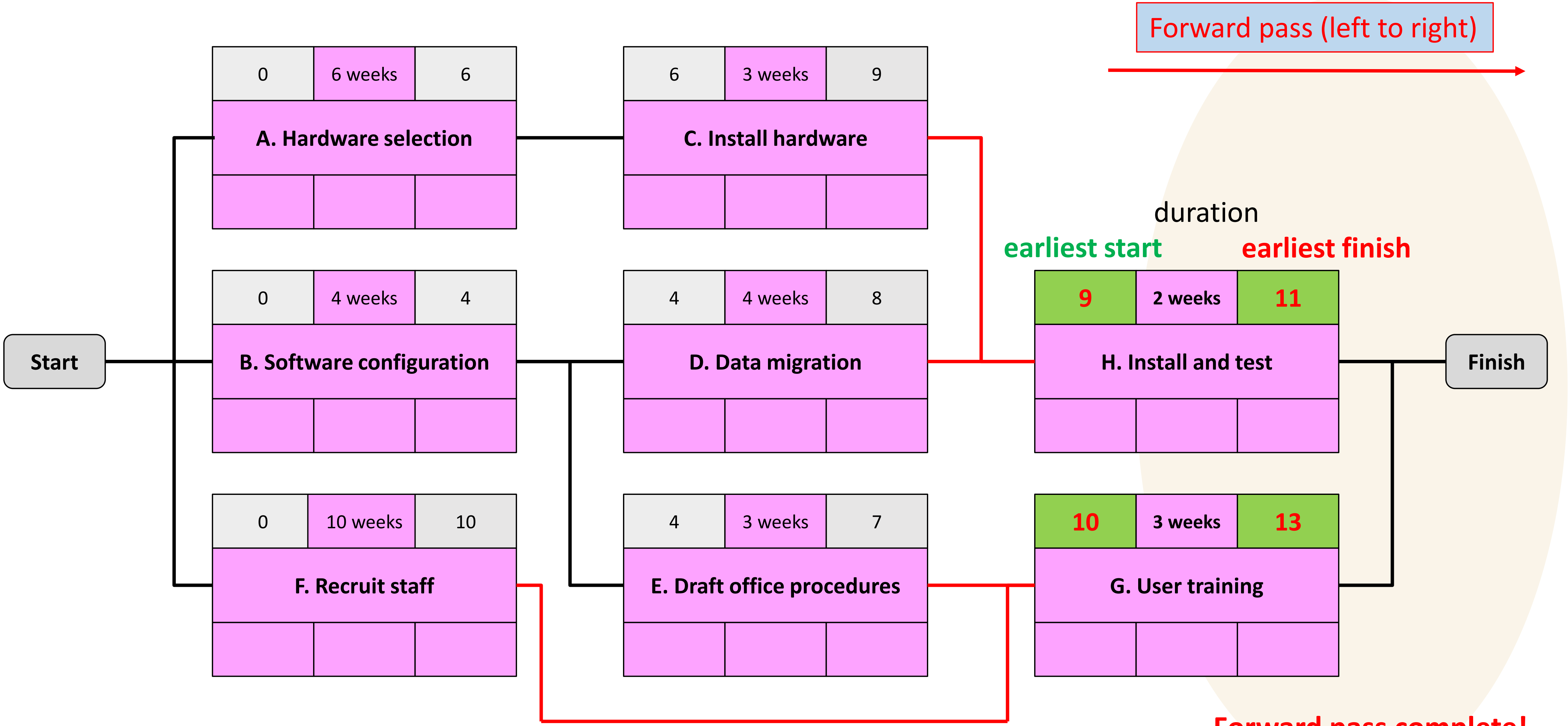
Earliest start/finish calculated



Earliest start/finish calculated



Earliest start/finish calculated



Forward pass (left to right)

duration

earliest start

earliest finish

Start

0 | 6 weeks | 6
A. Hardware selection

6 | 3 weeks | 9
C. Install hardware

0 | 4 weeks | 4
B. Software configuration

4 | 4 weeks | 8
D. Data migration

9 | 2 weeks | 11
H. Install and test

0 | 10 weeks | 10
F. Recruit staff

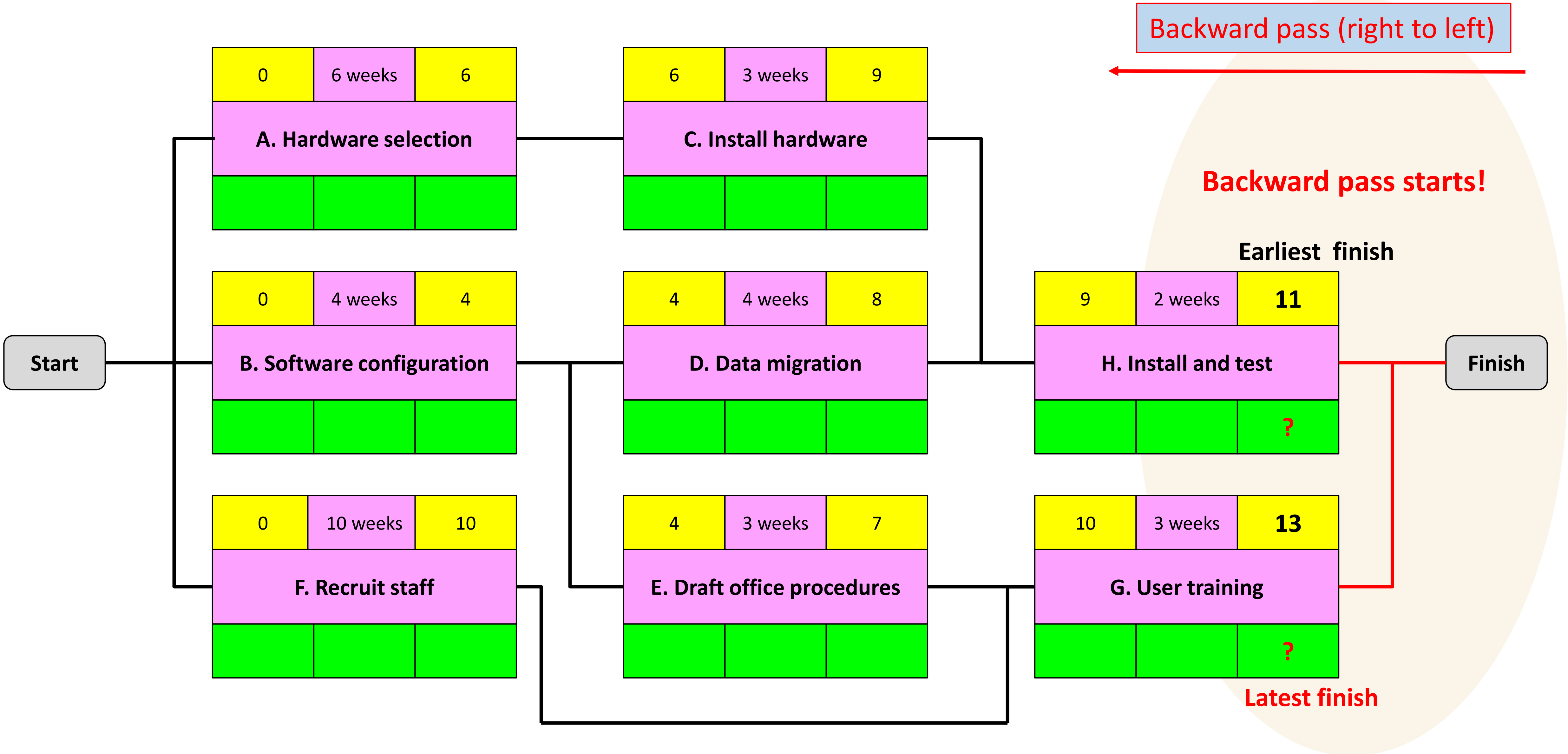
4 | 3 weeks | 7
E. Draft office procedures

10 | 3 weeks | 13
G. User training

Finish

Forward pass complete!

Latest start/finish and float calculated



Backward pass (right to left)

Backward pass starts!

Earliest finish

Start

0 | 6 weeks | 6
A. Hardware selection

6 | 3 weeks | 9
C. Install hardware

0 | 4 weeks | 4
B. Software configuration

4 | 4 weeks | 8
D. Data migration

9 | 2 weeks | **11**
H. Install and test
?

0 | 10 weeks | 10
F. Recruit staff

4 | 3 weeks | 7
E. Draft office procedures

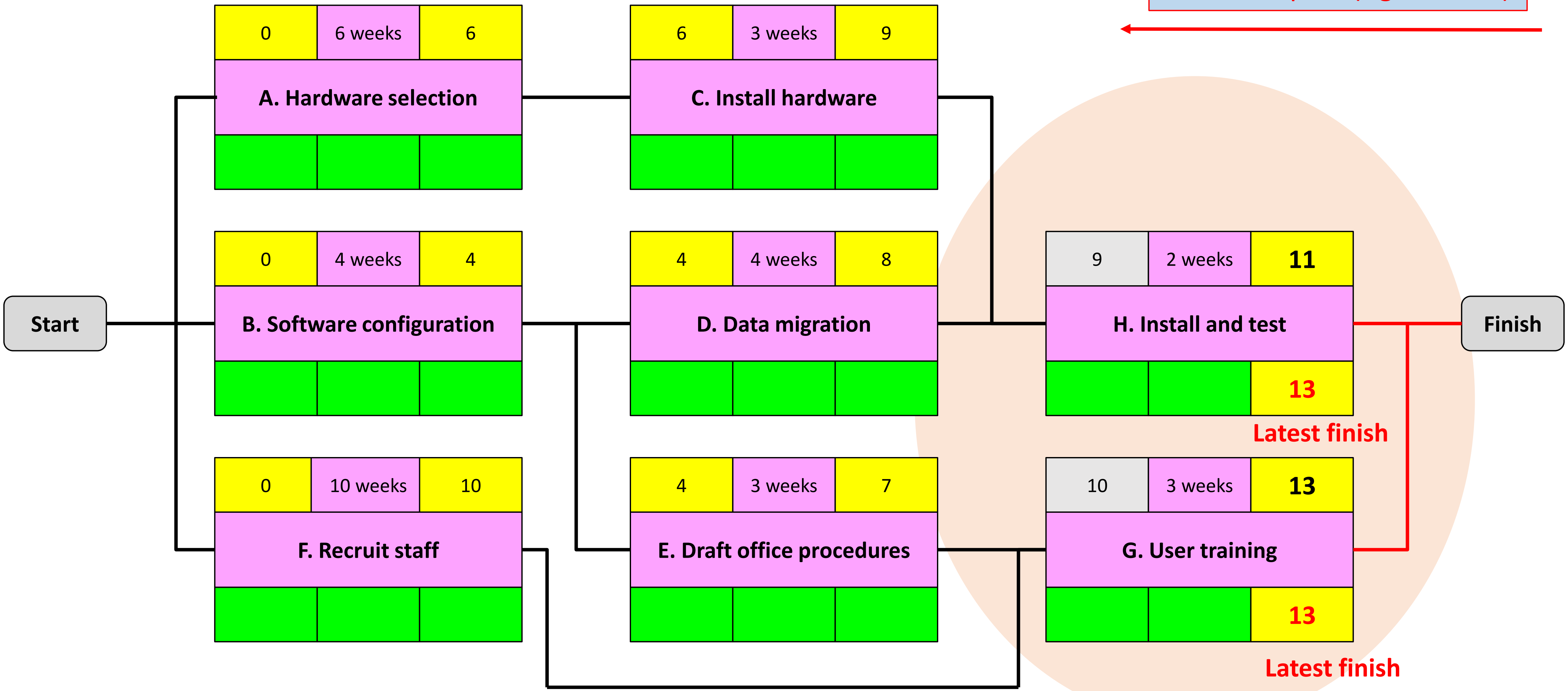
10 | 3 weeks | **13**
G. User training
?

Finish

Latest finish

Latest start/finish and float calculated

Backward pass (right to left)



Start

0	6 weeks	6
A. Hardware selection		

6	3 weeks	9
C. Install hardware		

0	4 weeks	4
B. Software configuration		

4	4 weeks	8
D. Data migration		

9	2 weeks	11
H. Install and test		
		13

Latest finish

0	10 weeks	10
F. Recruit staff		

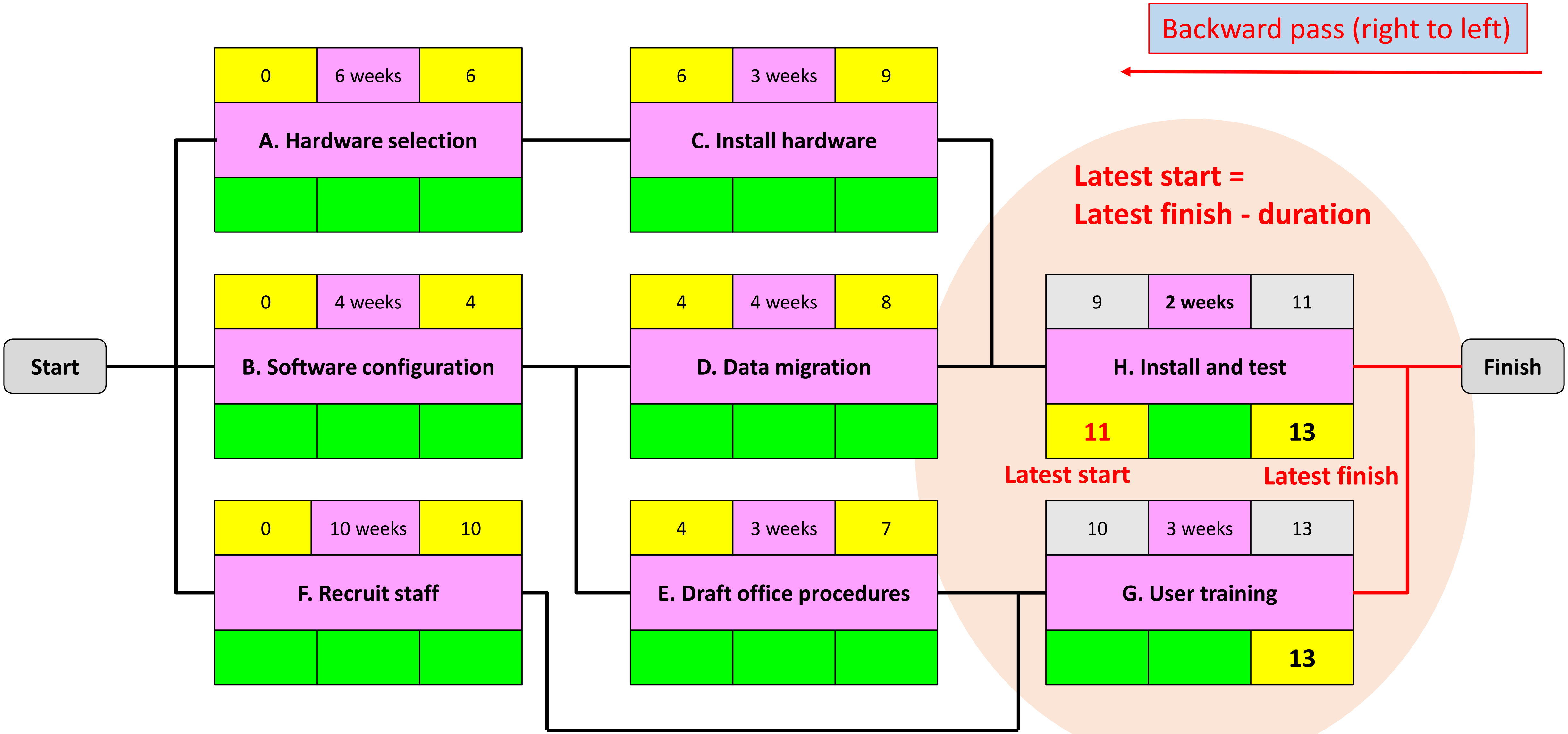
4	3 weeks	7
E. Draft office procedures		

10	3 weeks	13
G. User training		
		13

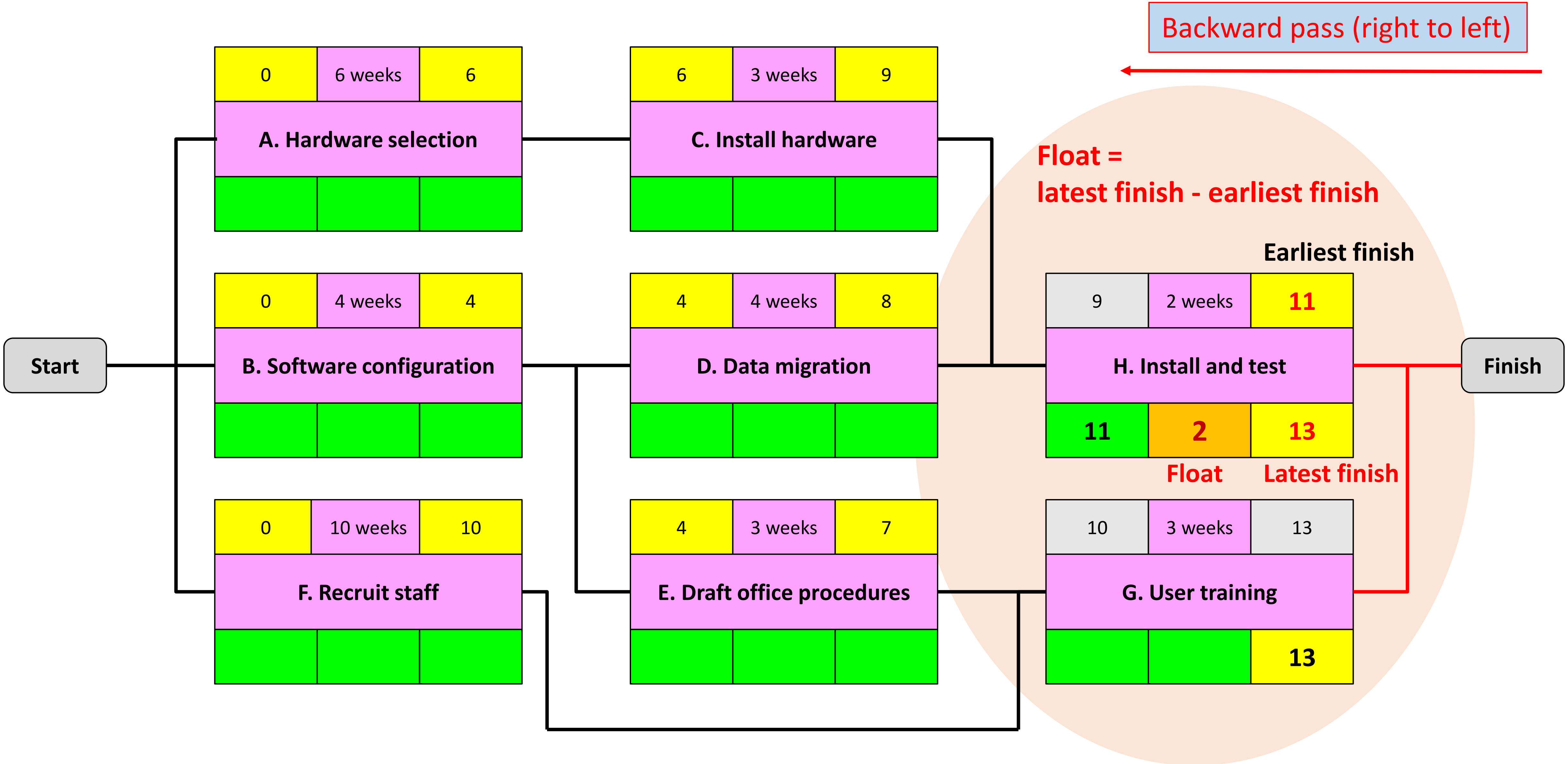
Latest finish

Finish

Latest start/finish and float calculated



Latest start/finish and float calculated



Backward pass (right to left)

Float =
latest finish - earliest finish

Start

0 | 6 weeks | 6
A. Hardware selection

0 | 4 weeks | 4
B. Software configuration

0 | 10 weeks | 10
F. Recruit staff

6 | 3 weeks | 9
C. Install hardware

4 | 4 weeks | 8
D. Data migration

4 | 3 weeks | 7
E. Draft office procedures

Earliest finish

9 | 2 weeks | 11
H. Install and test
11 | 2 | 13

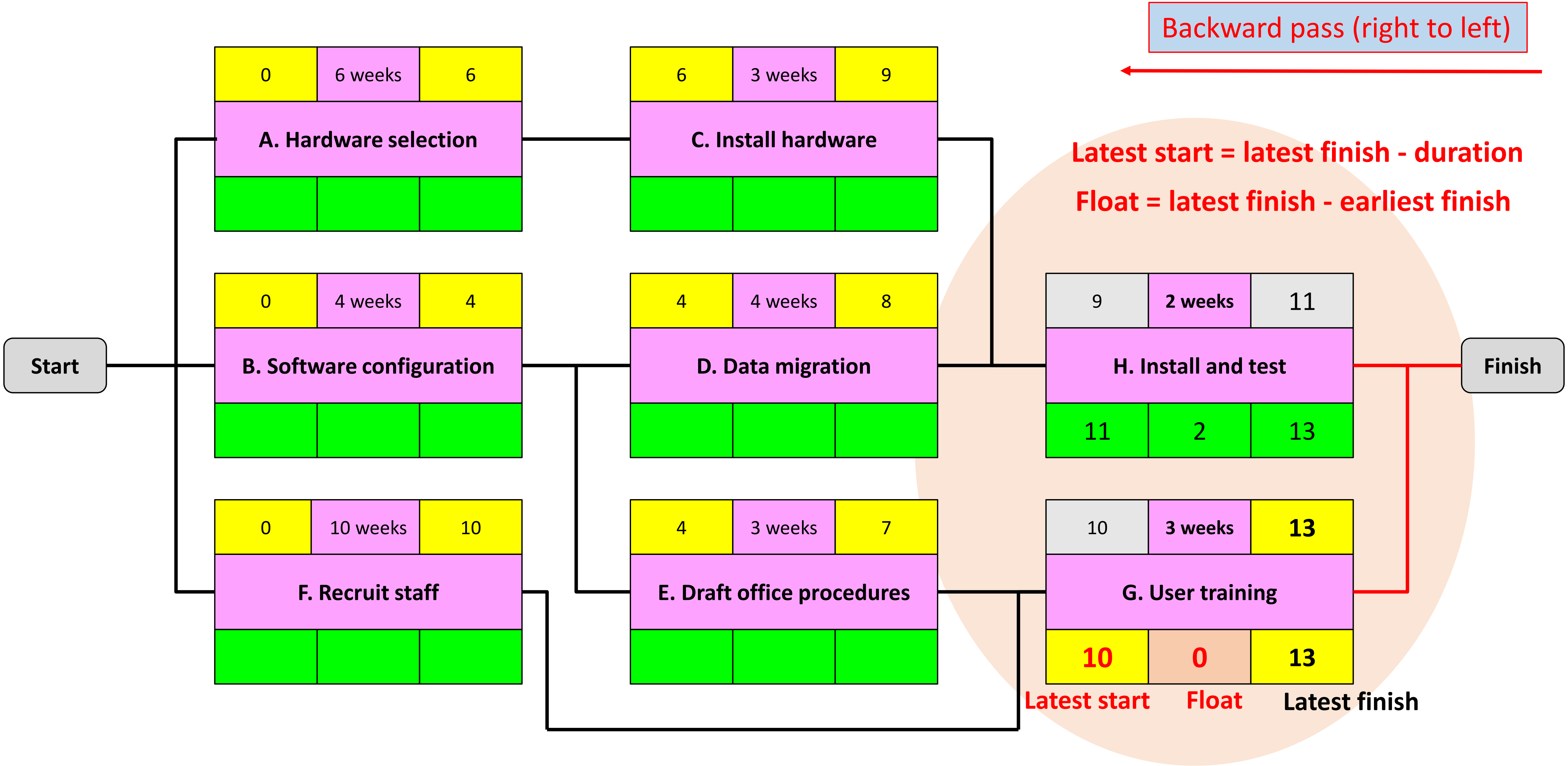
Float

Latest finish

10 | 3 weeks | 13
G. User training
13

Finish

Latest start/finish and float calculated



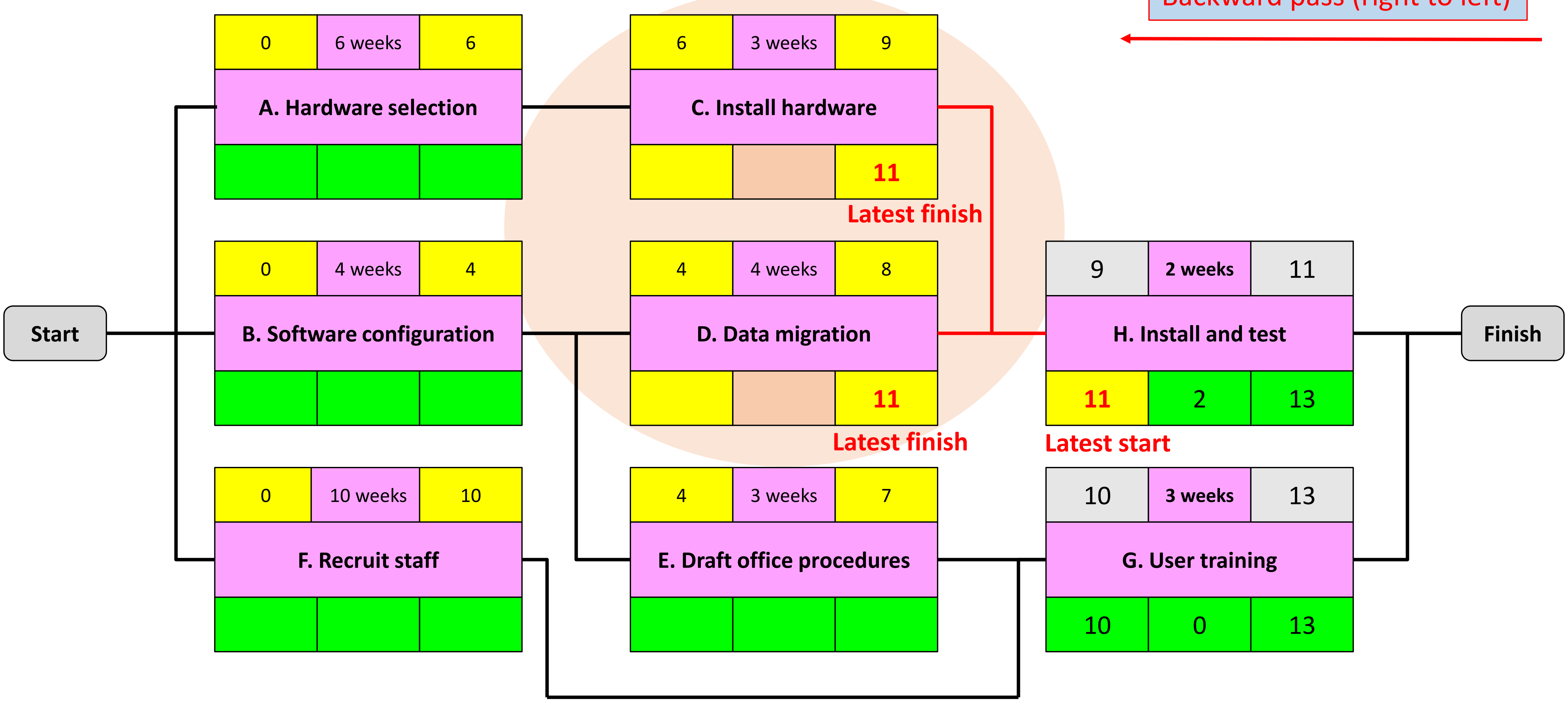
Backward pass (right to left)

Latest start = latest finish - duration

Float = latest finish - earliest finish

Latest start/finish and float calculated

Backward pass (right to left)



Start

0	6 weeks	6
A. Hardware selection		

0	4 weeks	4
B. Software configuration		

0	10 weeks	10
F. Recruit staff		

6	3 weeks	9
C. Install hardware		
		11

Latest finish

4	4 weeks	8
D. Data migration		
		11

Latest finish

4	3 weeks	7
E. Draft office procedures		

9	2 weeks	11
H. Install and test		
11	2	13

Latest start

10	3 weeks	13
G. User training		
10	0	13

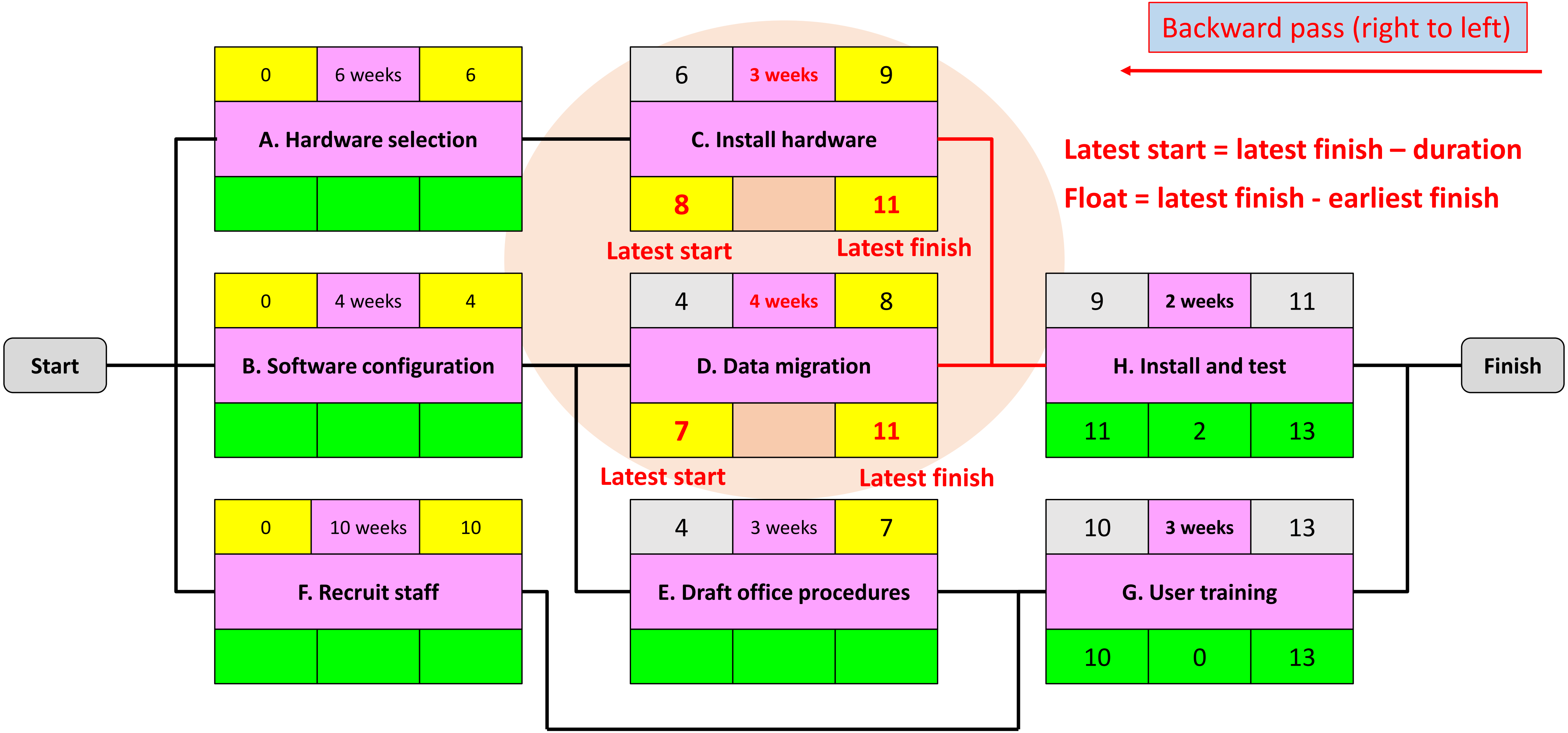
Finish

Latest start/finish and float calculated

Backward pass (right to left)

Latest start = latest finish – duration

Float = latest finish - earliest finish



0	6 weeks	6
A. Hardware selection		

0	4 weeks	4
B. Software configuration		

0	10 weeks	10
F. Recruit staff		

6	3 weeks	9
C. Install hardware		
8		11

4	4 weeks	8
D. Data migration		
7		11

4	3 weeks	7
E. Draft office procedures		

9	2 weeks	11
H. Install and test		
11	2	13

10	3 weeks	13
G. User training		
10	0	13

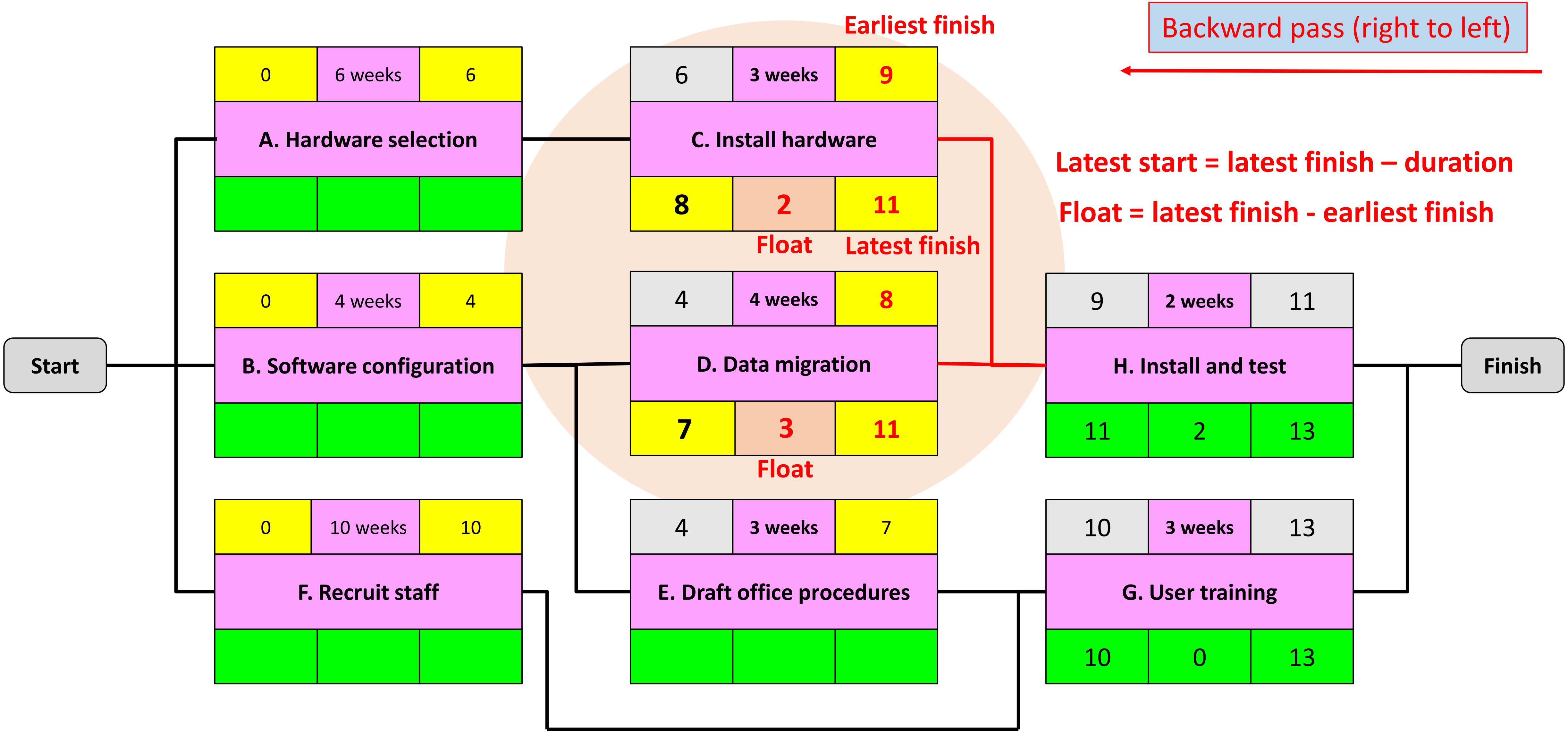
Latest start/finish and float calculated

Aston University
BIRMINGHAM UK

Backward pass (right to left)

Latest start = latest finish – duration

Float = latest finish - earliest finish



0	6 weeks	6
A. Hardware selection		

0	4 weeks	4
B. Software configuration		

0	10 weeks	10
F. Recruit staff		

6	3 weeks	9
C. Install hardware		
8	2	11

4	4 weeks	8
D. Data migration		
7	3	11

4	3 weeks	7
E. Draft office procedures		

9	2 weeks	11
H. Install and test		
11	2	13

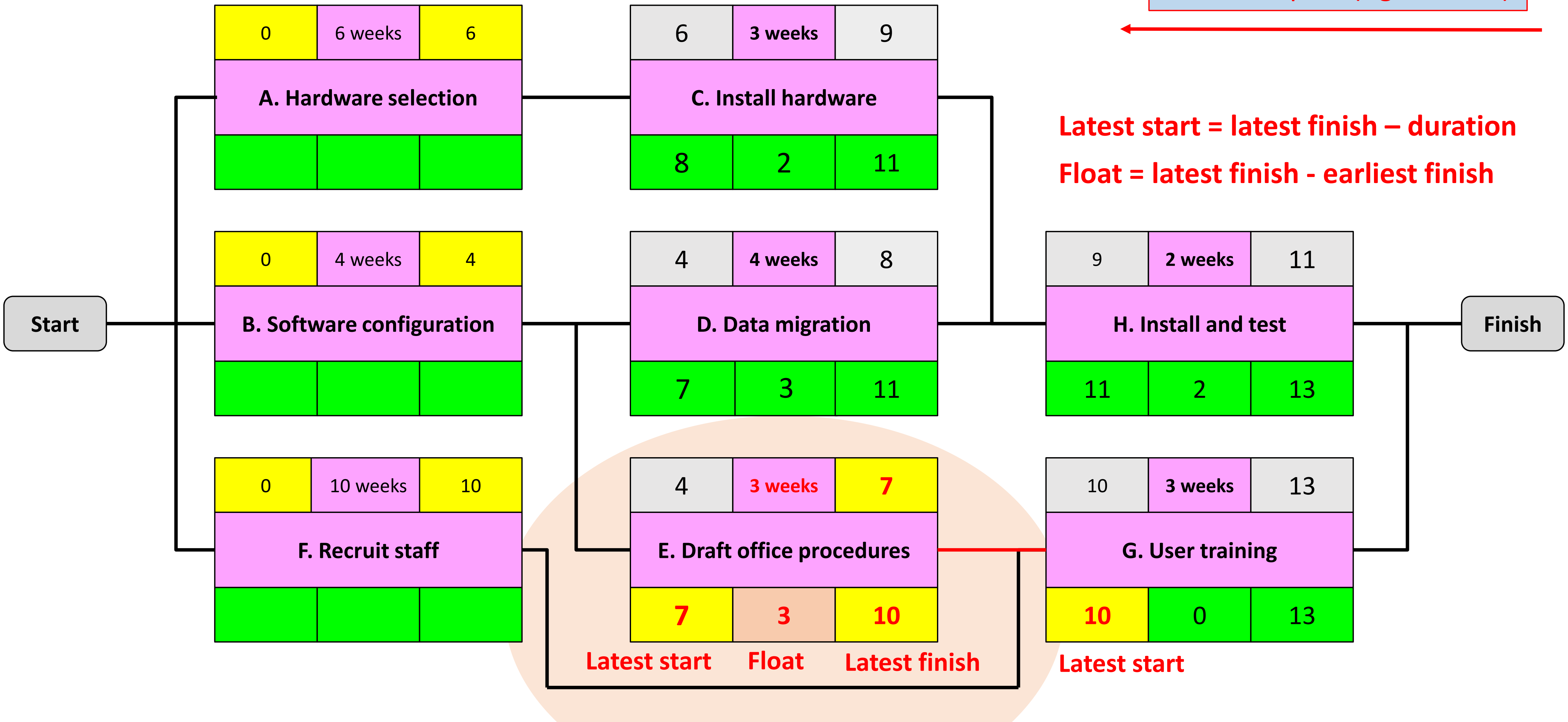
10	3 weeks	13
G. User training		
10	0	13

Latest start/finish and float calculated

Backward pass (right to left)

Latest start = latest finish – duration

Float = latest finish - earliest finish



Start

0 | 6 weeks | 6
A. Hardware selection

0 | 4 weeks | 4
B. Software configuration

0 | 10 weeks | 10
F. Recruit staff

6 | 3 weeks | 9
C. Install hardware
8 | 2 | 11

4 | 4 weeks | 8
D. Data migration
7 | 3 | 11

4 | 3 weeks | 7
E. Draft office procedures
7 | 3 | 10

Latest start Float Latest finish

9 | 2 weeks | 11
H. Install and test
11 | 2 | 13

10 | 3 weeks | 13
G. User training
10 | 0 | 13

Latest start

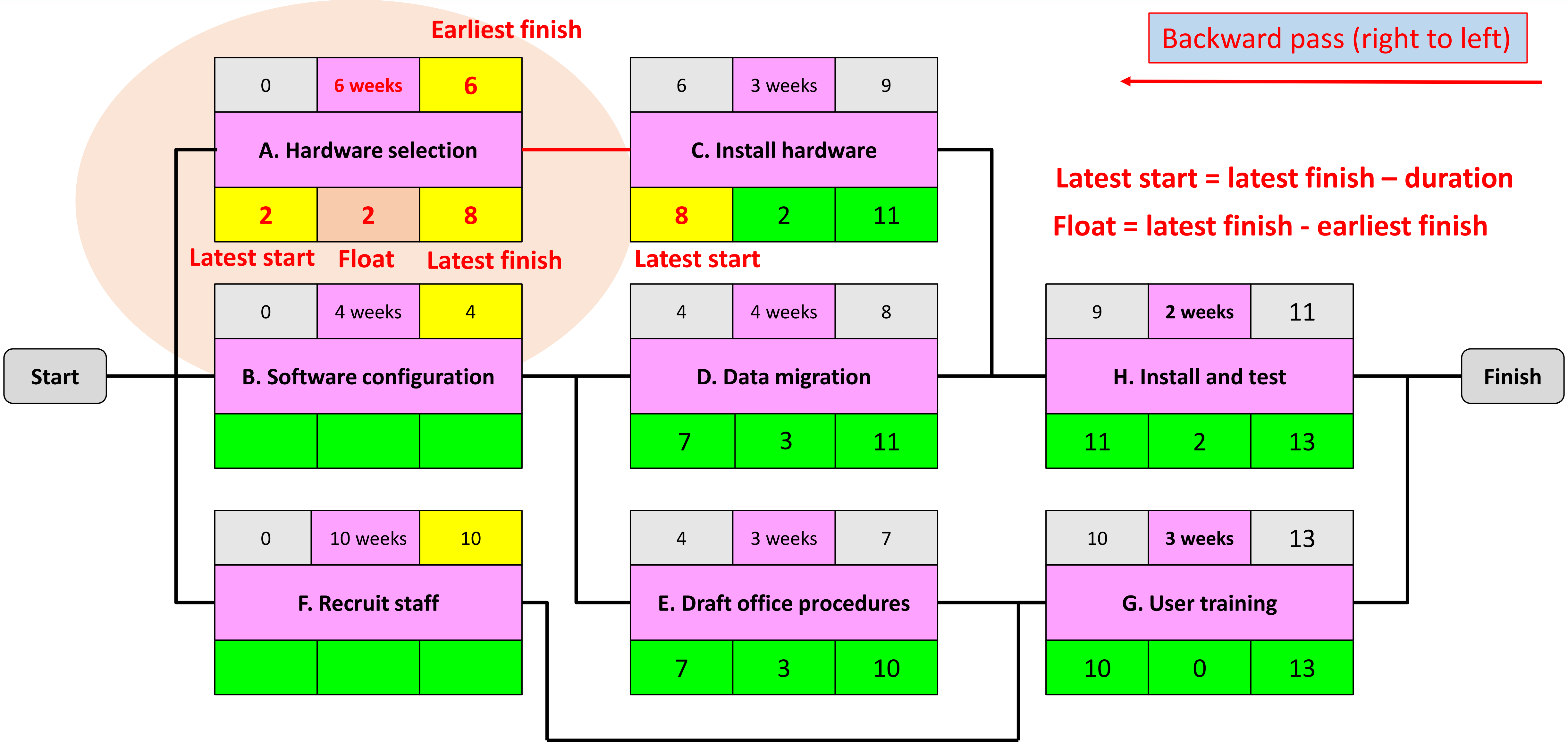
Finish

Latest start/finish and float calculated

Backward pass (right to left)

Latest start = latest finish – duration

Float = latest finish - earliest finish



Earliest finish

0	6 weeks	6
A. Hardware selection		
2	2	8

Latest start Float Latest finish

6	3 weeks	9
C. Install hardware		
8	2	11

Latest start

0	4 weeks	4
B. Software configuration		

4	4 weeks	8
D. Data migration		
7	3	11

9	2 weeks	11
H. Install and test		
11	2	13

0	10 weeks	10
F. Recruit staff		

4	3 weeks	7
E. Draft office procedures		
7	3	10

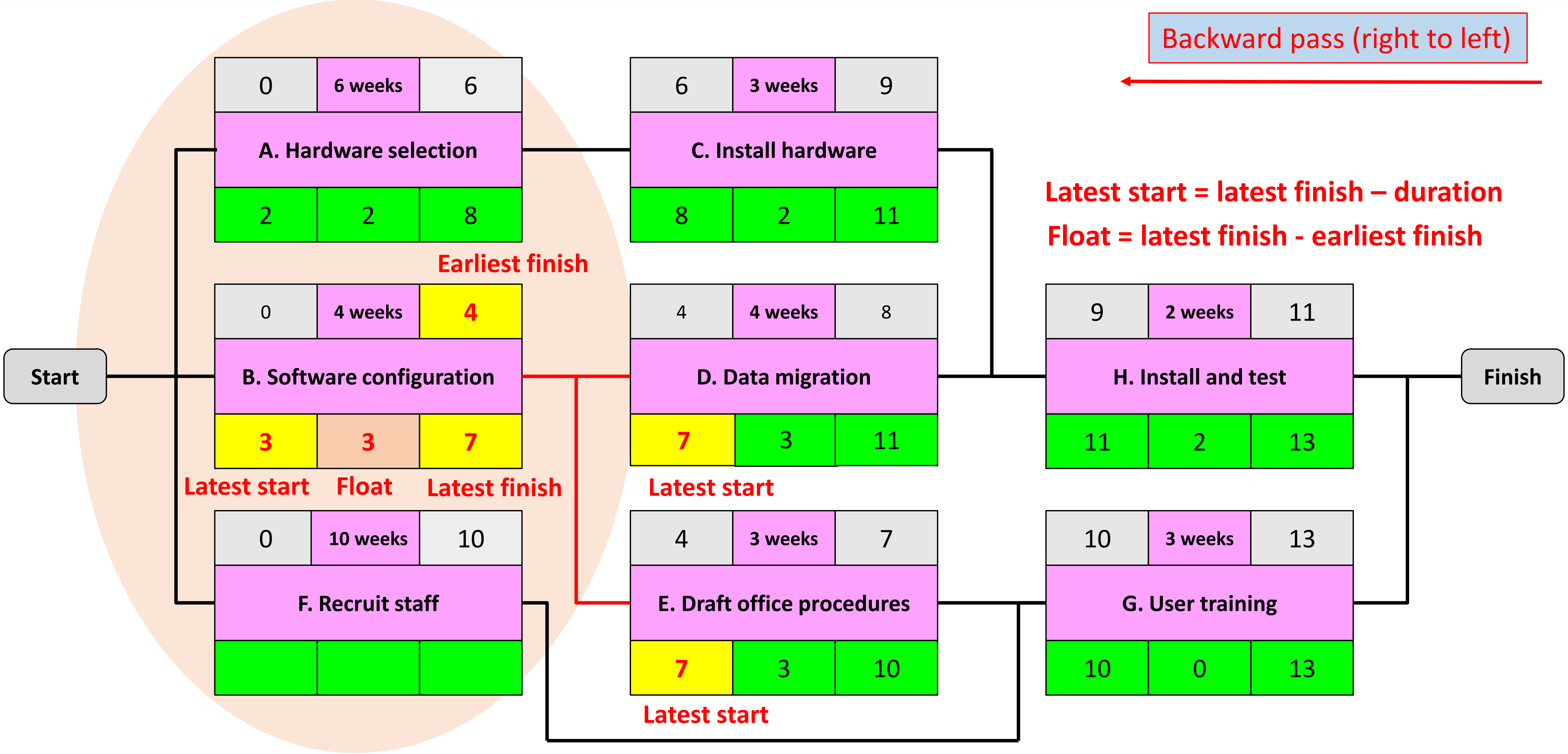
10	3 weeks	13
G. User training		
10	0	13

Latest start/finish and float calculated

Backward pass (right to left)

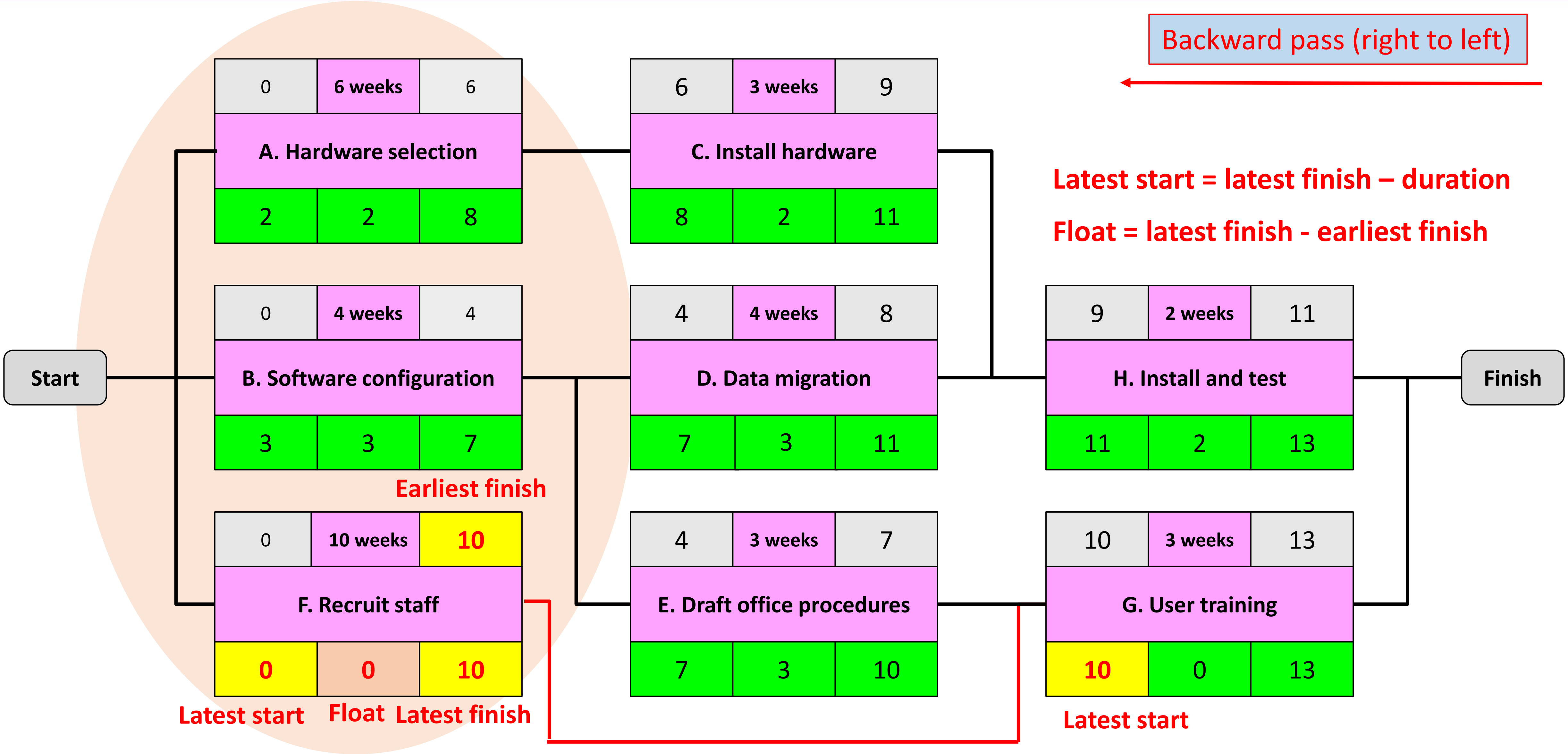
Latest start = latest finish – duration

Float = latest finish - earliest finish



A. Hardware selection	0	6 weeks	6	2	2	8
B. Software configuration	0	4 weeks	4	3	3	7
F. Recruit staff	0	10 weeks	10			
C. Install hardware	6	3 weeks	9	8	2	11
D. Data migration	4	4 weeks	8	7	3	11
E. Draft office procedures	4	3 weeks	7	7	3	10
H. Install and test	9	2 weeks	11	11	2	13
G. User training	10	3 weeks	13	10	0	13

Latest start/finish and float calculated



Backward pass (right to left)

Latest start = latest finish – duration

Float = latest finish - earliest finish

Start

0	**6 weeks**	6
	A. Hardware selection	
2	2	8

0	**4 weeks**	4
	B. Software configuration	
3	3	7

Earliest finish

0	**10 weeks**	**10**
	F. Recruit staff	
0	**0**	**10**

Latest start **Float** **Latest finish**

6	**3 weeks**	9
	C. Install hardware	
8	2	11

4	**4 weeks**	8
	D. Data migration	
7	3	11

4	**3 weeks**	7
	E. Draft office procedures	
7	3	10

9	**2 weeks**	11
	H. Install and test	
11	2	13

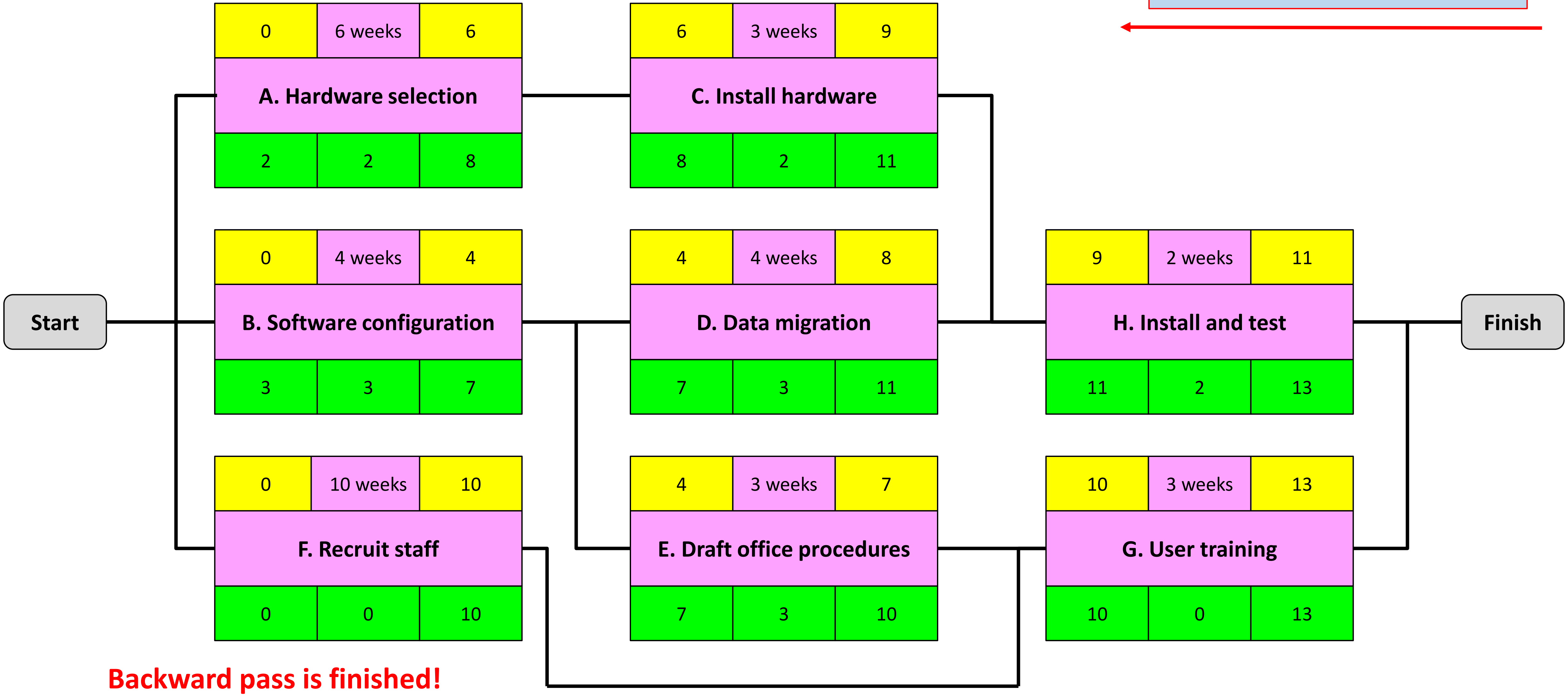
10	**3 weeks**	13
	G. User training	
10	0	13

Latest start

Finish

Latest start/finish and float calculated

Backward pass (right to left)



0	6 weeks	6
A. Hardware selection		
2	2	8

6	3 weeks	9
C. Install hardware		
8	2	11

0	4 weeks	4
B. Software configuration		
3	3	7

4	4 weeks	8
D. Data migration		
7	3	11

9	2 weeks	11
H. Install and test		
11	2	13

0	10 weeks	10
F. Recruit staff		
0	0	10

4	3 weeks	7
E. Draft office procedures		
7	3	10

10	3 weeks	13
G. User training		
10	0	13

Backward pass is finished!

The Critical Path Analysis (CPA)

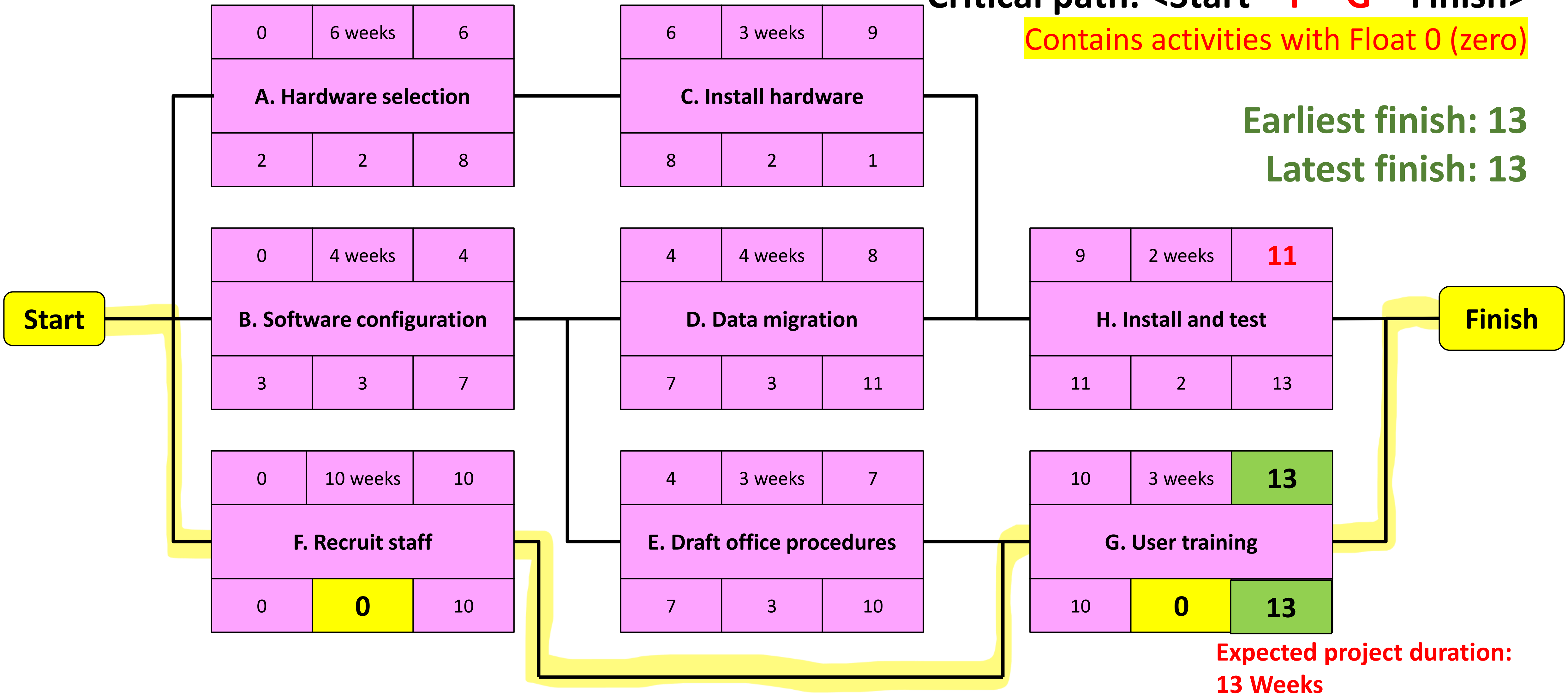
Aston University
BIRMINGHAM UK

Critical path: <Start – F – G – Finish>

Contains activities with Float 0 (zero)

Earliest finish: 13

Latest finish: 13



Start

0	6 weeks	6
A. Hardware selection		
2	2	8

6	3 weeks	9
C. Install hardware		
8	2	1

0	4 weeks	4
B. Software configuration		
3	3	7

4	4 weeks	8
D. Data migration		
7	3	11

9	2 weeks	**11**
H. Install and test		
11	2	13

0	10 weeks	10
F. Recruit staff		
0	**0**	10

4	3 weeks	7
E. Draft office procedures		
7	3	10

10	3 weeks	**13**
G. User training		
10	**0**	**13**

Finish

Expected project duration: 13 Weeks

Activities

- Must have clearly defined start and end-points

- Must have resource requirements that can be forecasted: these are assumed to be constant throughout the project

- Must have a duration that can be forecasted

- May be dependent on other activities being completed first (precedence networks)

Rules for constructing activity networks

- Only one start and one end node – to avoid confusion
- Activities have durations – they do in real-world projects
- Time moves from left to right – convention to aid readability
- Loops are not allowed – to enable analysis
  - Finite loops specifying iterations can be “unfolded”

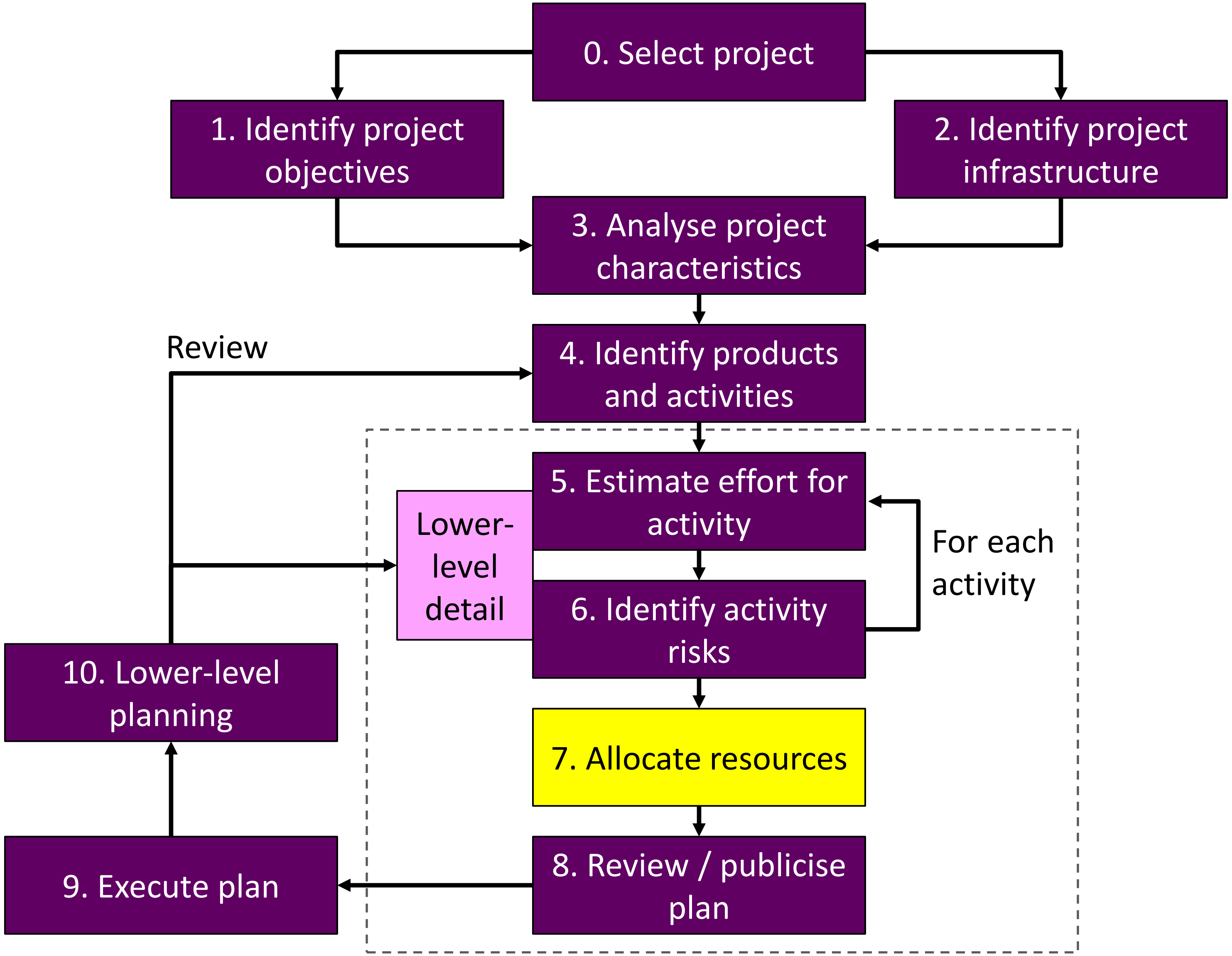
Steps to reduce the Critical Path

- Eliminate tasks on the critical path
- Re-plan serial paths to be in parallel
- Overlap sequential tasks
- Shorten the duration of critical path tasks
- Shorten early tasks
- Shorten longest tasks
- Shorten easiest tasks
- Shorten tasks that cost the least to speed up

Outline

- Frameworks for software project planning
- Selection of software project approaches
- Effort estimation for software projects
- Activity planning and resource allocation
  - Activity planning
  - **Resource allocation**

Steps 4 and 5 of step wise


0. Select project
1. Identify project objectives
2. Identify project infrastructure
3. Analyse project characteristics
Review
4. Identify products and activities
5. Estimate effort for activity
Lower-level detail
For each activity
6. Identify activity risks
7. Allocate resources
8. Review / publicise plan
9. Execute plan
10. Lower-level planning

Objectives of resource allocation

- Identification of the resources required for the project

- Making the demand for resources more even throughout the lifespan of the project

- Production of a work plan and resource schedule

Resources categories

- These include in general:
  - **Labour**
  - Equipment
  - Materials
  - Space
  - **Services**
  - Time
  - Money

Examples resources:

- **Staff, team, developers, managers…**
- computers, hardware, software, tools, or frameworks
- raw materials, components, or consumables
- access to information and knowledge
- licenses, permits, or regulatory approvals
- communication tools
- basic utilities
- **suppliers, vendors, contractors, technical support …**
- training materials
- risk assessment tools
- stakeholders support

Among others…

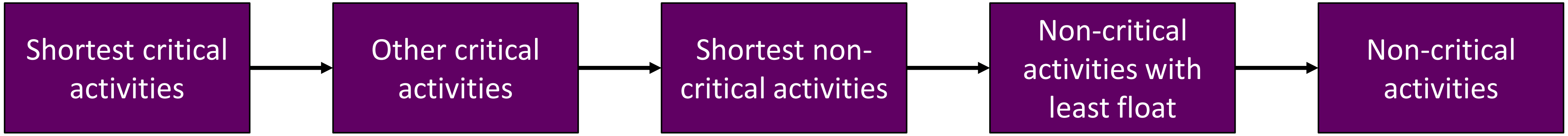
Resource allocation

- The project manager needs to:
  - Identify the resources needed for each activity and create a resource requirement list
  - Identify resource types (e.g. ‘VB programmers’ as opposed to ‘software developers’) – as individuals are interchangeable within the group
  - Allocate resource types to activities

- Resource clashes
  - Where the same resource is needed in more than one place at the same time
  - Can be resolved by:
    - Delaying one of the activities
    - Taking advantage of float to change start date
    - Delaying start of one activity until finish of the other activity that resource is being used on - puts back project completion
  - Can be resolved by moving resource from a non-critical activity
  - Can be resolved by bringing in additional resource - increases costs

Prioritising activities

- **Prioritising** is required when several activities are **competing for the same limited resource at the same time**
- Resources are allocated to activities in **decreasing priority order**. There are two main ways of doing this:
  - **Total float priority** – the activities with the smallest float have the highest priority; activities with the same float are processed in any order
  - **Ordered list priority** – this takes account of the duration of the activity as well as the float
- Typical priority list:



Shortest critical activities → Other critical activities → Shortest non-critical activities → Non-critical activities with least float → Non-critical activities

Aston University
BIRMINGHAM UK

Software Project Management

Unit 3: Software Project Planning (2)

Thais Webber
Richard Lee