Integration test planning

Testing takes place throughout the software life cycle.

Testing can apply to:

- design;
- source code;
- manuals; and
- tests themselves (choice of test data, etc.).

Integration test planning is carried out during the design stage.

An *integration test plan* is a collection of *integration tests* that focus on *functionality*.

Bottom up integration testing

There are two basic approaches to integration testing:

1. bottom up testing, and
2. top down testing.

Assume that detailed design consists of a collection of structure charts.

Bottom up integration testing proceeds as follows.

Unit test individual lowest level modules first. Lowest modules are combined to form subsystems, the subsystems tested, and so on.

NOTE: An artificial environment is necessary for each integration test; the environment consists of driver programs and test data, and is called a *test harness*.

Bottom up integration testing: disadvantages

- Must code and debug test harnesses (extra work).

- There can be difficulty in combining subsystems and then testing them. An extreme case: all modules united tested, then combined together; this is called *big bang* integration testing.

- It can be difficult to fully unit test a module. A more complex driver is usually necessary, which can further complicate finding the source of errors.

Top down integration testing

Modules at top of structure chart are tested first, starting with the main or control modules.

NOTE: For called modules not yet written, it is necessary to use *stubs*, i.e., simple dummy modules used to avoid linker errors.

One therefore uses older more reliable modules to test new modules.

There is little emphasis on unit testing, perhaps nothing beyond successful compilation. Instead an integration test is used to test any given module.

Process of designing integration tests

1. Look at design plans.
2. Decide on the functions to test.†
3. Identify *test threads* and necessary scaffolding.
4. Determine test data requirements.

---

†Our next subject is on estimating and scheduling. In determining the functions, one should try for the corresponding test threads that can be worked in in parallel by different test teams, hopefully without any necessary interaction.

Definitions

Integration Test
Tests action of a group of modules accomplishing an identifiable function.

Test Thread
A group of modules being tested.

Scaffolding
Set of modules, stubs and possibly a test harness connected to the test thread, but not in the test thread.

Hints and notes

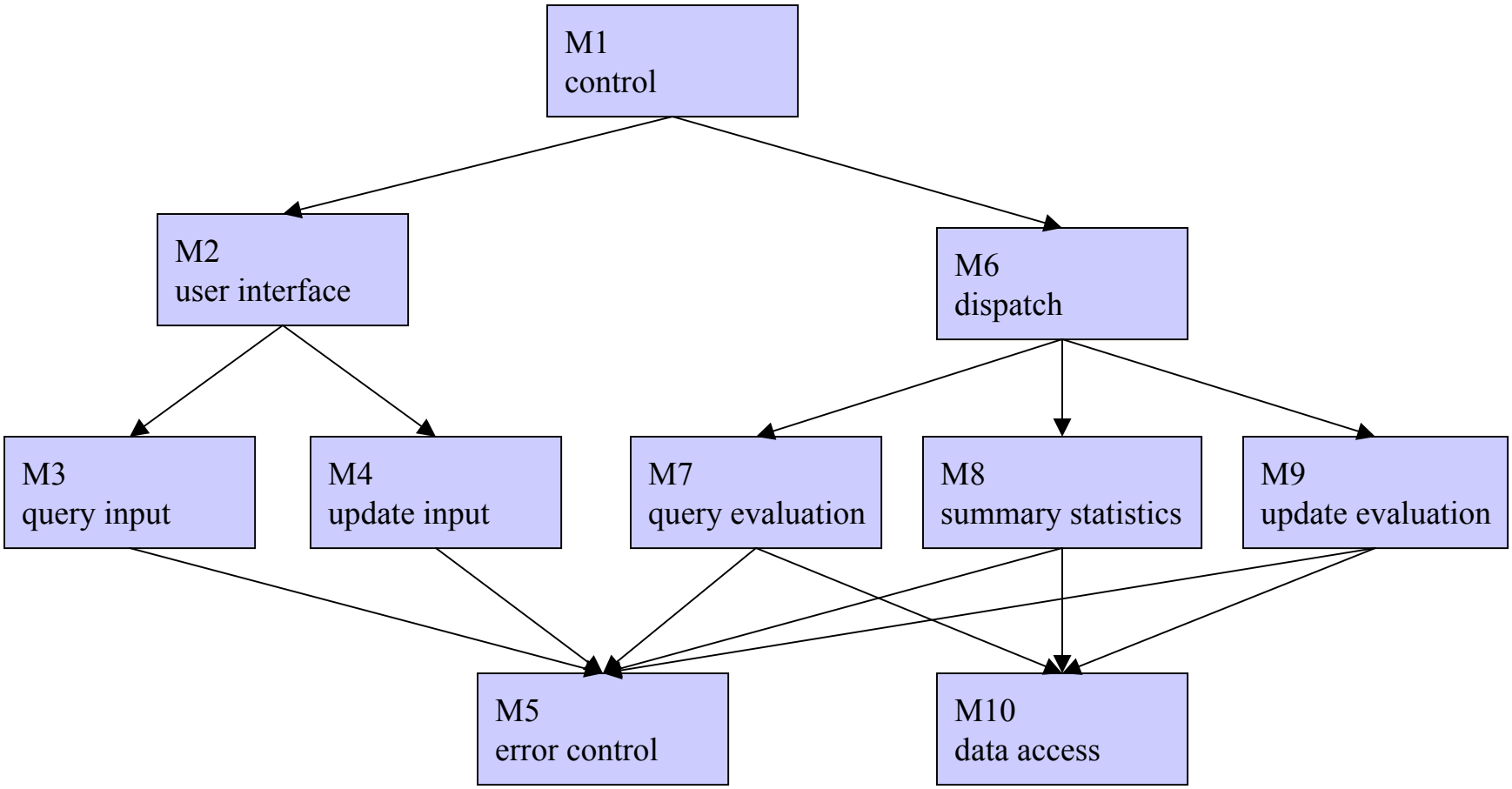
- Use earlier tests to provide scaffolding and data for later tests. This suggests one should test input first.

- As new modules replace stubs, all or part of previous tests should be repeated to ensure no new errors. These are called *regression tests*.

- Although top-down integration testing is usually preferable, it may not be possible in some cases:
  - Module groups performing complex tasks; or
  - Need to test timing requirements (e.g., device drivers).

For many cases, it is possible to isolate and fully test such subsystems first. Thus, some combination of bottom up and top down testing is required; this is called *sandwich testing*.

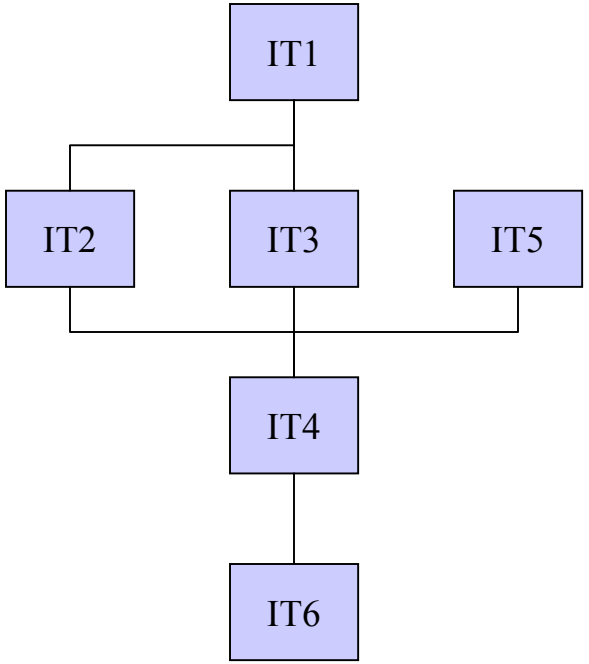
Example


M1
control
M2
user interface
M6
dispatch
M3
query input
M4
update input
M7
query evaluation
M8
summary statistics
M9
update evaluation
M5
error control
M10
data access

Test priority charts

Implies the following.

1. IT1 before IT2 and IT3.
2. IT1 in parallel to IT5.
3. IT2 in parallel to IT3.
4. IT2, IT3 and IT5 before IT4.
5. All before IT6.



Further notes on integration test planning

- Automated tools can aid in integration testing. Examples:
  - test data generators;
  - command language programs;
  - development support libraries;
  - etc.
- Another factor which governs selection of test threads:

module criticality;

that is , module complexity and/or importance in the software system. Such modules should be tested as early as possible and included in many test threads.

More definitions

Build
The task of coding and unit testing a collection of modules, stubs and test harnesses.

Build Plan
A set of builds that include all modules, stubs and test harnesses used in an integration test plan.

Implementation Task
Either a build or an integration test.

Schedule Priority Chart
Requires an integration test plan and a build plan. Corresponds to a priority chart that includes a node for each implementation task.

Estimating

A large subject.

Four general approaches.

1. Expert judgment.
2. Delphi cost estimation.
3. Algorithmic methods.
4. Work breakdown.

For purposes of assignment two, use a combination of work breakdown and expert judgment, and assume you are the expert.

Scheduling: *Critical Path Method* (CPM)

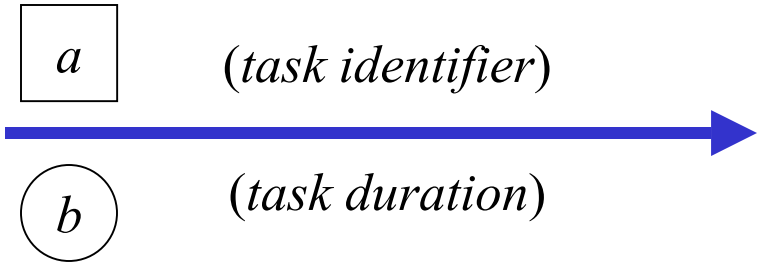
Takes the following as input.

1. A schedule priority chart.
2. Estimates of how long each task in the schedule priority chart will take.
3. An assignment of each task to a programmer.
4. An ordering on the tasks assigned to each programmer.

Computes the following.

1. Earliest start time of each task.
2. Earliest completion time for all tasks.
3. Slack time, the amount of time starting on a task can be delayed without affecting the earliest completion date.

CPM: Notational convention for tasks



$a \equiv$ (earliest possible start time of the task)

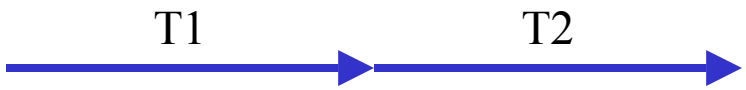
$b \equiv$ (latest possible start time of the task)†

$b - a \equiv$ (slack time for the task)

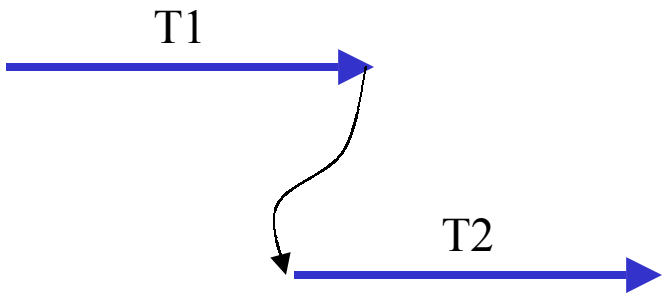
†Without delaying earliest completion time for all tasks.

CPM: Representing ordering constraints

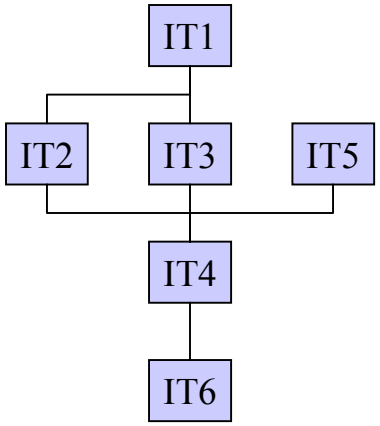
To capture that task T2 must wait for the completion of task T1 before starting:



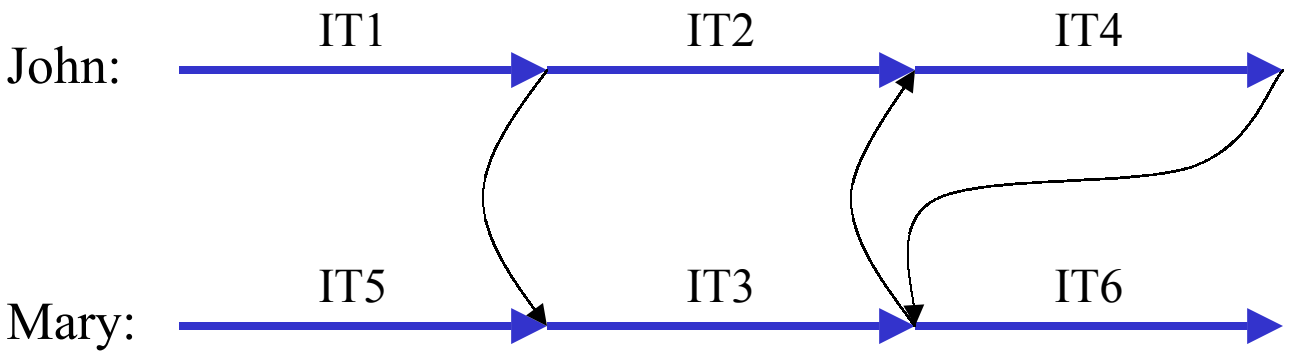
or



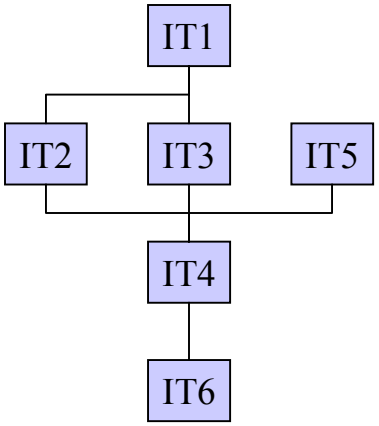
CPM Example



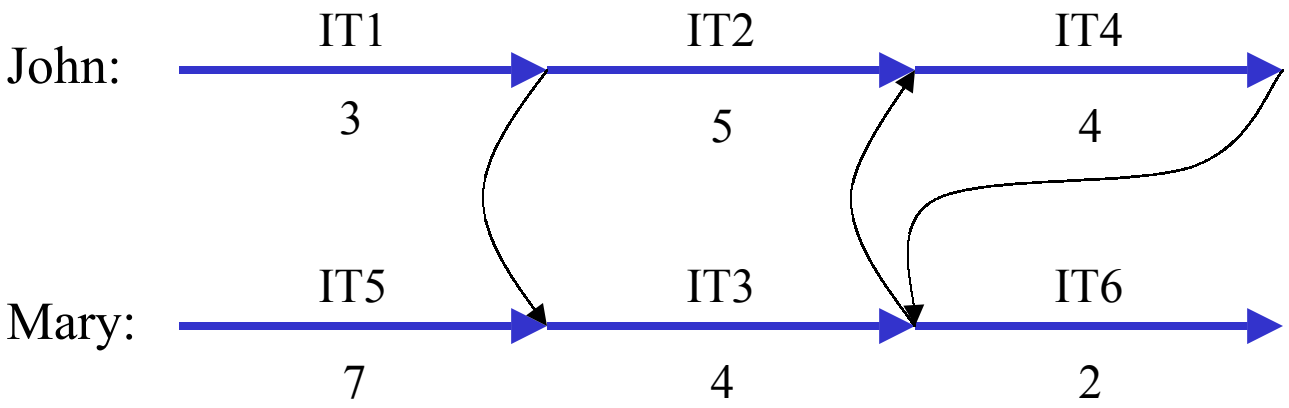
CPM Example



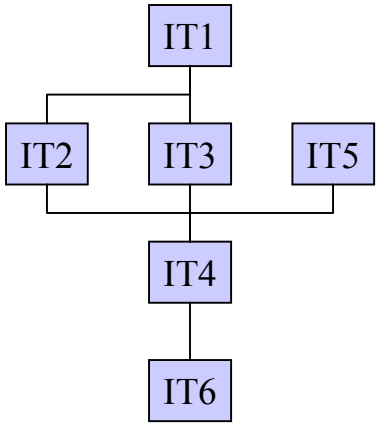
(*adding ordering constraints*)



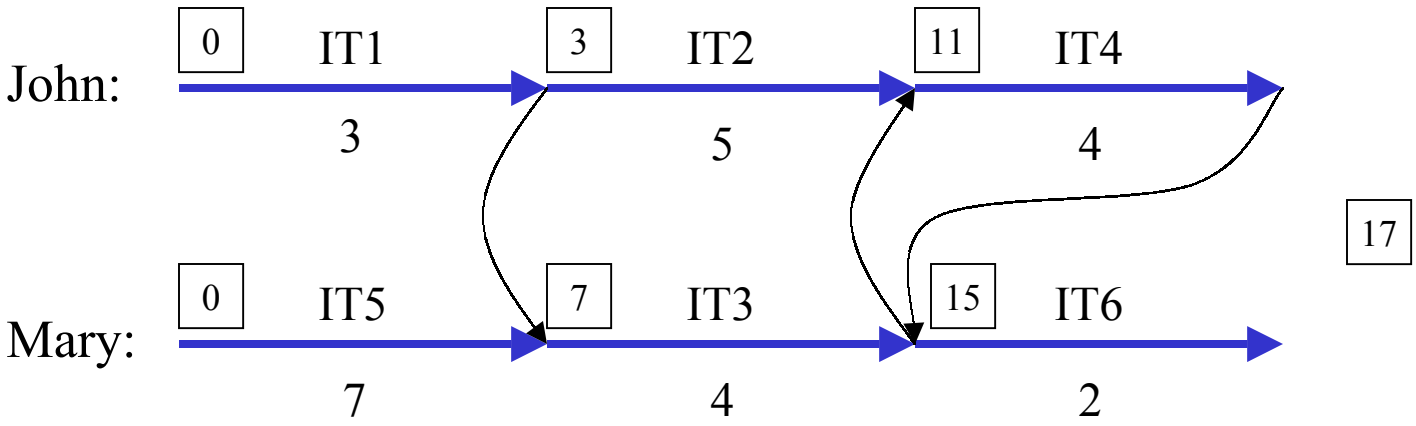
CPM Example

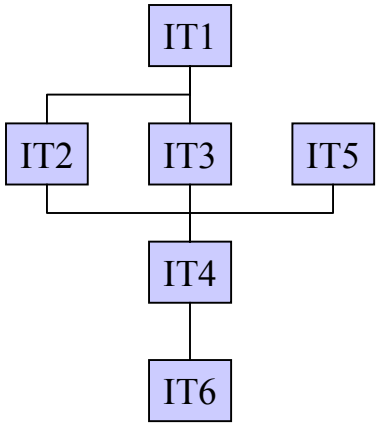


(*adding task durations*)



CPM Example

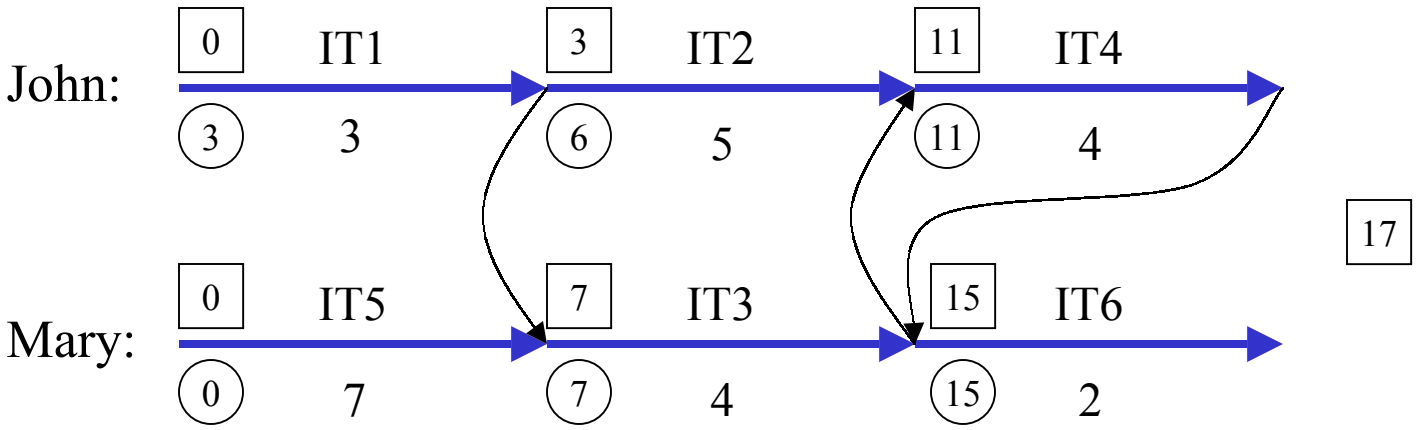




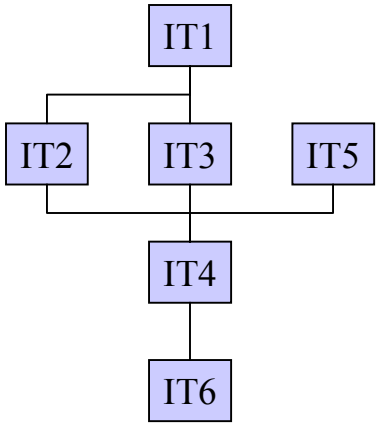
(*calculating earliest start times*)

(*calculating earliest completion time*)

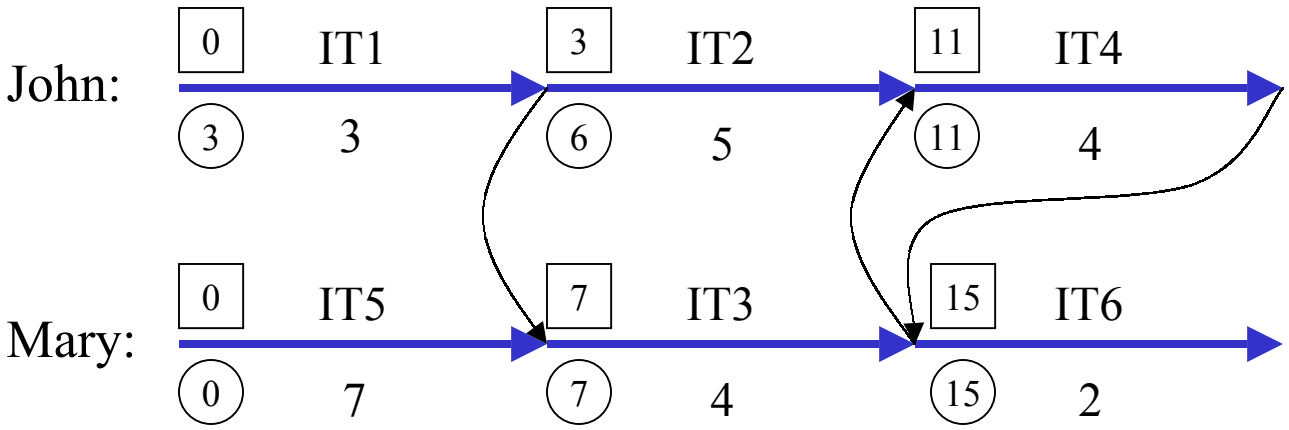
CPM Example

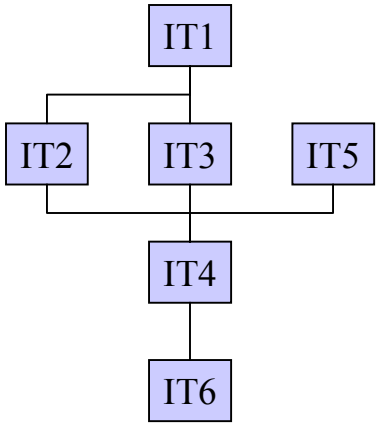


(*calculating latest possible start times*)



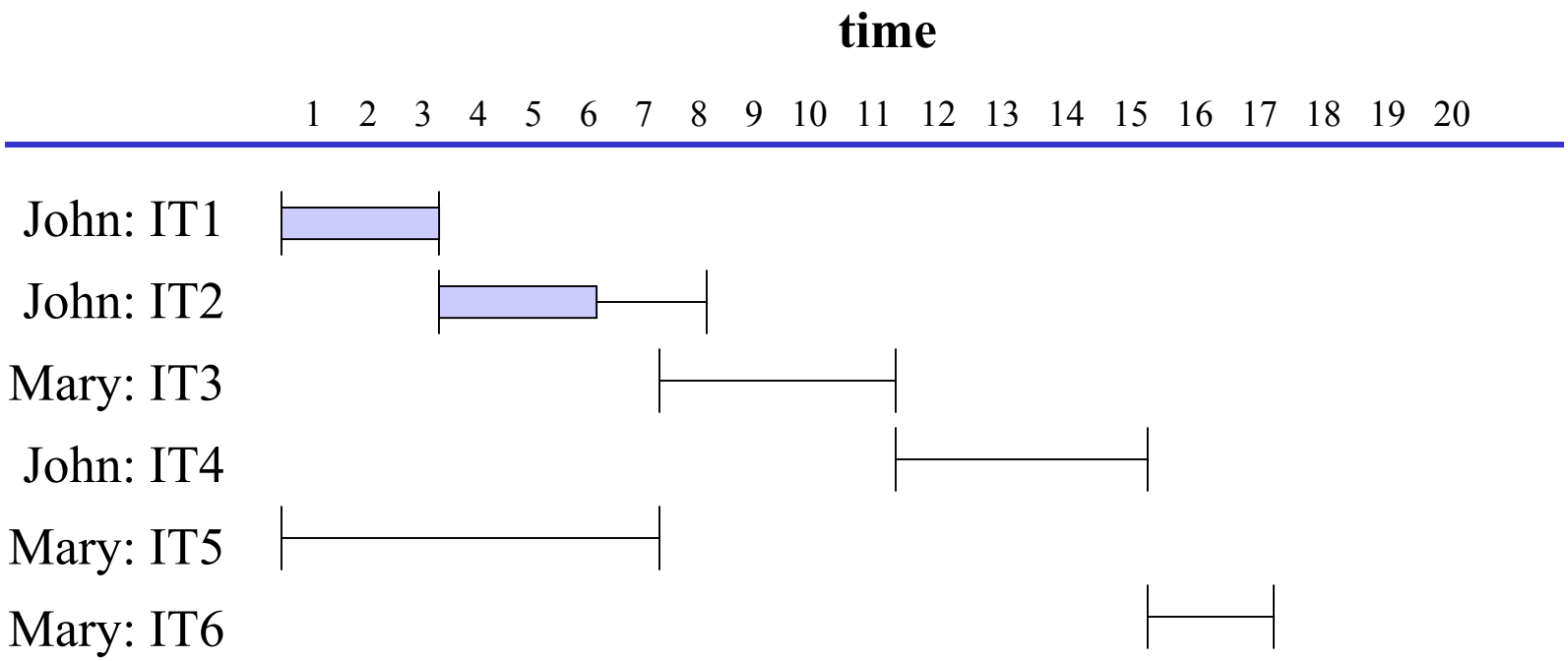
CPM Example





Tasks on the critical path: IT4, IT5, IT3 and IT6.

Gantt charts: example



Gantt charts: notation