11S3112 PPMPL

# Testing in Software Life Cycle

## Week 2, Session 1
## Tahun Ajaran 22/23

**Institut Teknologi Del**
Jl. Sisingamangaraja
Sitoluama, Laguboti 22381
Toba – SUMUT
http://www.pidel.org

Lecturer : ACB
Teaching Assistant: Apria & Aprialdy
- *PPMPL* 11S3112–
- *Sem Ganjil 2022/2023* -

# Content

- Testing Definition
- Different views of testing
- Testing terminologies
  - Waterfall model and V model
  - Roles of Testing
- Details of unit testing, integration testing, and system testing

# Testing Definition

- IEEE Std 829-1983: the process of analysing a software item to detect the difference between existing and required conditions (that is bugs) and to evaluate the features of the software item
- IEEE Std 610.12-1990: the process of operating system or component under specified conditions, observing or recording the results, and making an evaluation of some aspects of the system or component

# Different views of Testing

- *Hetzel, 1973:* establishing confidence that a program does what it is supposed to do
- Myers, 1979: the process of executing a program with intent of finding errors

# Testing Terminologies

- *Error*: the amount by which the result in incorrect (formally)
- *Mistake / error*: a human action that produces an incorrect result
- *Fault*: an incorrect step, process, or data definition in a computer program
- *Failure*: an incorrect result as observed from the program output (the manifestation of faults after the program is run)

# Testing Terminologies

- *A test case:* the input values (test data), expected output (produced by test oracle), the test objective
- *A good test case*: has high chance to reveal errors
- *A successful test case*: test case that uncover new error

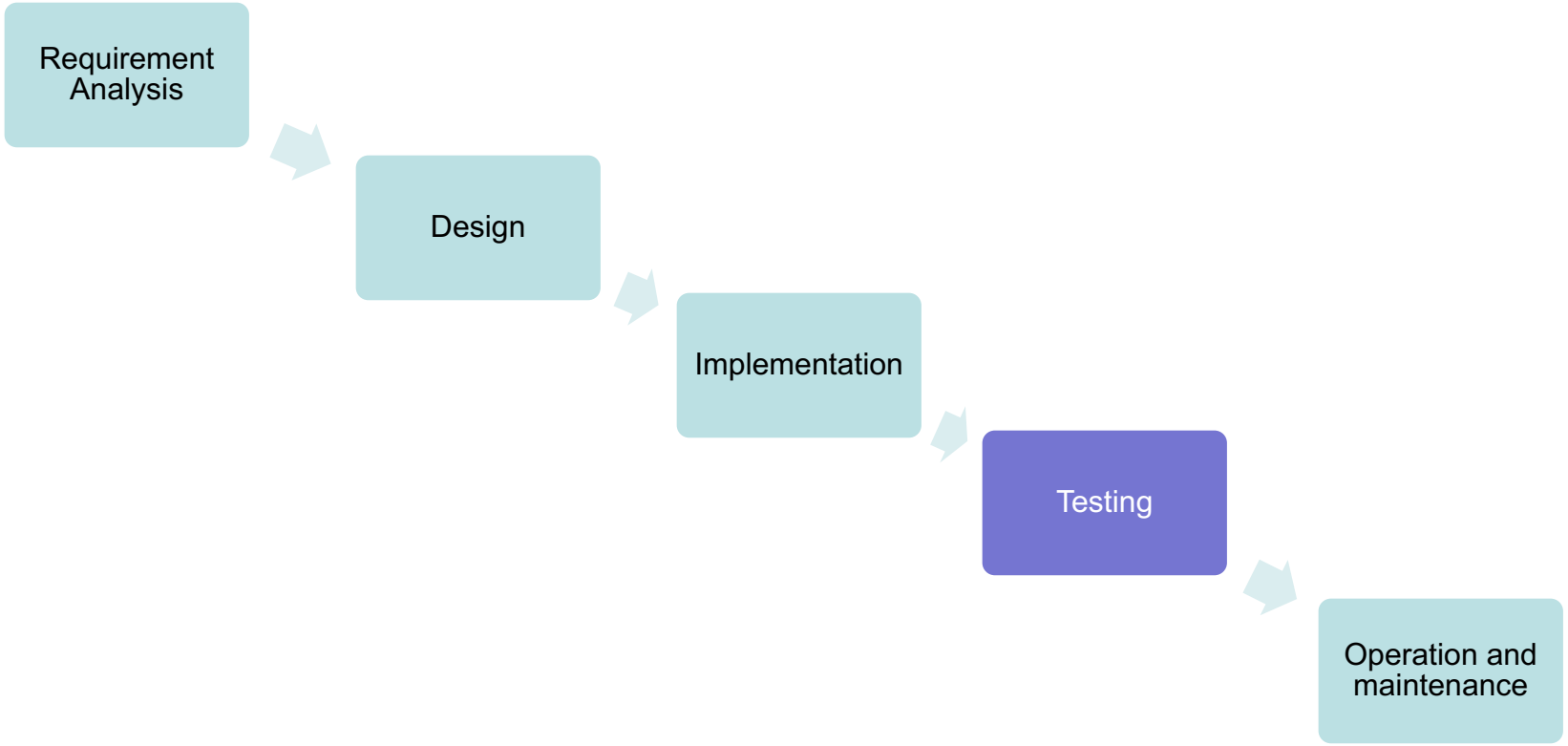
# Testing process

- A process of planning, preparation, and performing the actual testing of a software
- The process should be defined well and monitored so that:
  - *Important test will not be missed*
  - *continuous improvements is facilitated*
  - *We can learn and benefit from previous experience*
  - *We can convince users of the quality of testing*

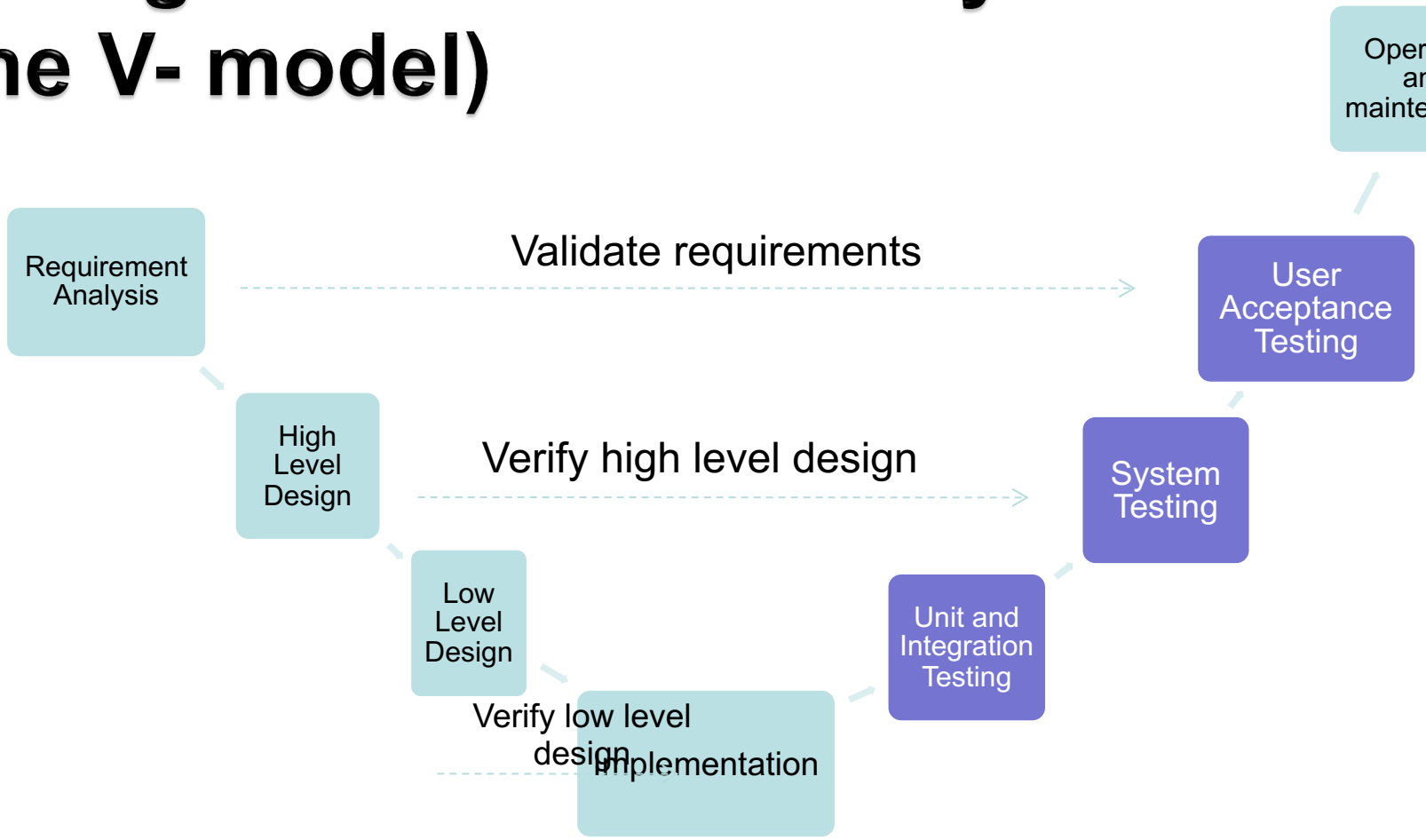
# Testing activity

- Define testing objectives
- Design test cases: valid and invalid situations
- Do the testing:
  - ❑ Prepare and document test data
    - ❑ Including Generation and Selection test data
  - ❑ Execute program using test data
  - ❑ Record and analyze the actual result
  - ❑ Evaluate and document the test results

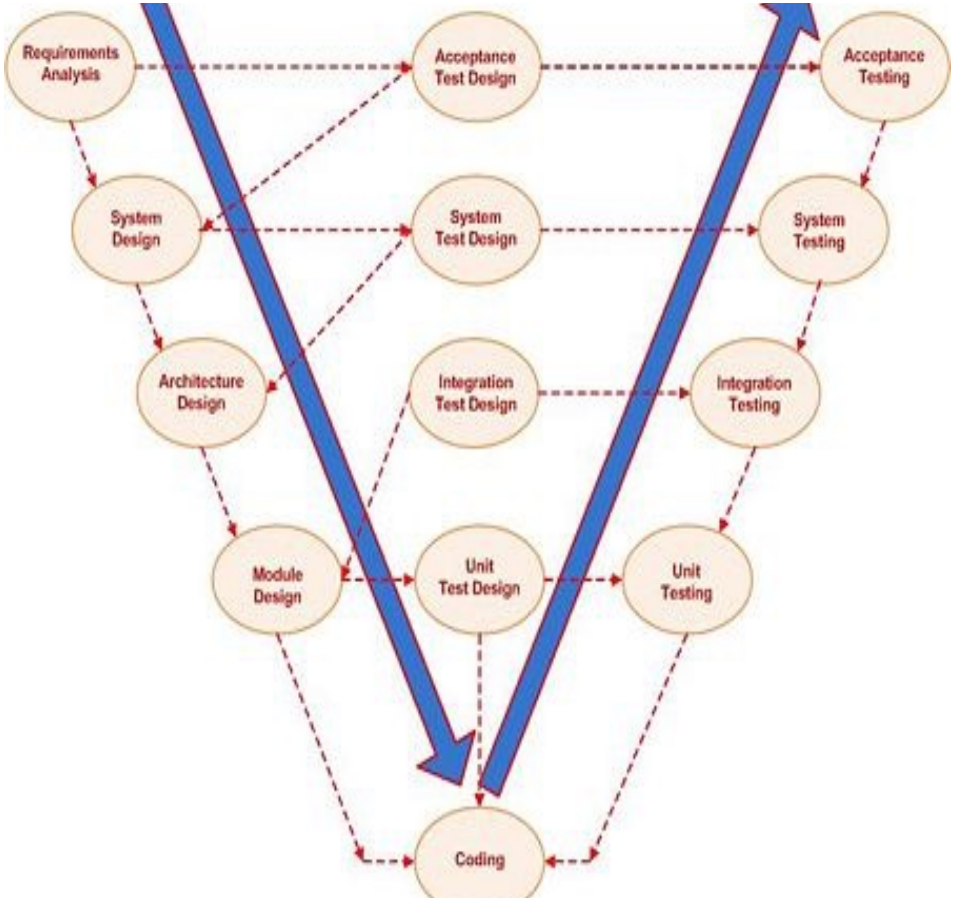
# Testing in Software Life Cycle (Traditional / the waterfall-model)



# Testing in Software Life Cycle (The V- model)



# The V- model…New Approach




**V-Model Phases**

# Requirement Analysis

- Client and developer work together to define the problems to be solved by the software
- SRS (system requirement specification): document specifying the requirements of the software
- SRS contains:

❑Functional requirements

❑Non-functional requirements, such as: performance, reliability, and usability

# Design

- Several possible ways to satisfy the requirements are explored
- One is chosen after considering the resources and constraints
- *High level design (system and architectural design)*: overall architecture of the system is designed
- *Low level design (detailed level design)*: each software component, description of program modules, arguments, and return parameters are produces

# Implementation

- Detailed design is implemented
- Coding by programmers

# Testing (1)

a) *Unit testing*: each module should be unit tested independently so that it conforms its specification in the low level design

b) *Integration testing*: after passing the unit testing, the modules are integrated for the integration testing

# Testing (2)

c) *System testing*: the integrated modules are integrated to form the entire system for the system test. It test the requirements in SRS (functional and non-functional), including performance testing, stress/load testing,

d) *User acceptance testing*: involving customer, includes *alpha test* and *beta test,* should address the acceptance criteria in the SRS

# Operation and Maintenance

- Put into real use
- During the operation, bugs might be discovered and fixed, software might be improved
- When the software is unable to maintain due to whatever reasons, new software is required

# a) Unit Testing activity

- Design test cases to test every module
- Do the testing:
  - Prepare and document test data
    - Including Generation and Selection test data
  - Execute module under testing using test data
  - Record and analyze the actual result
  - Evaluate and document the test results

# b) Integration Testing

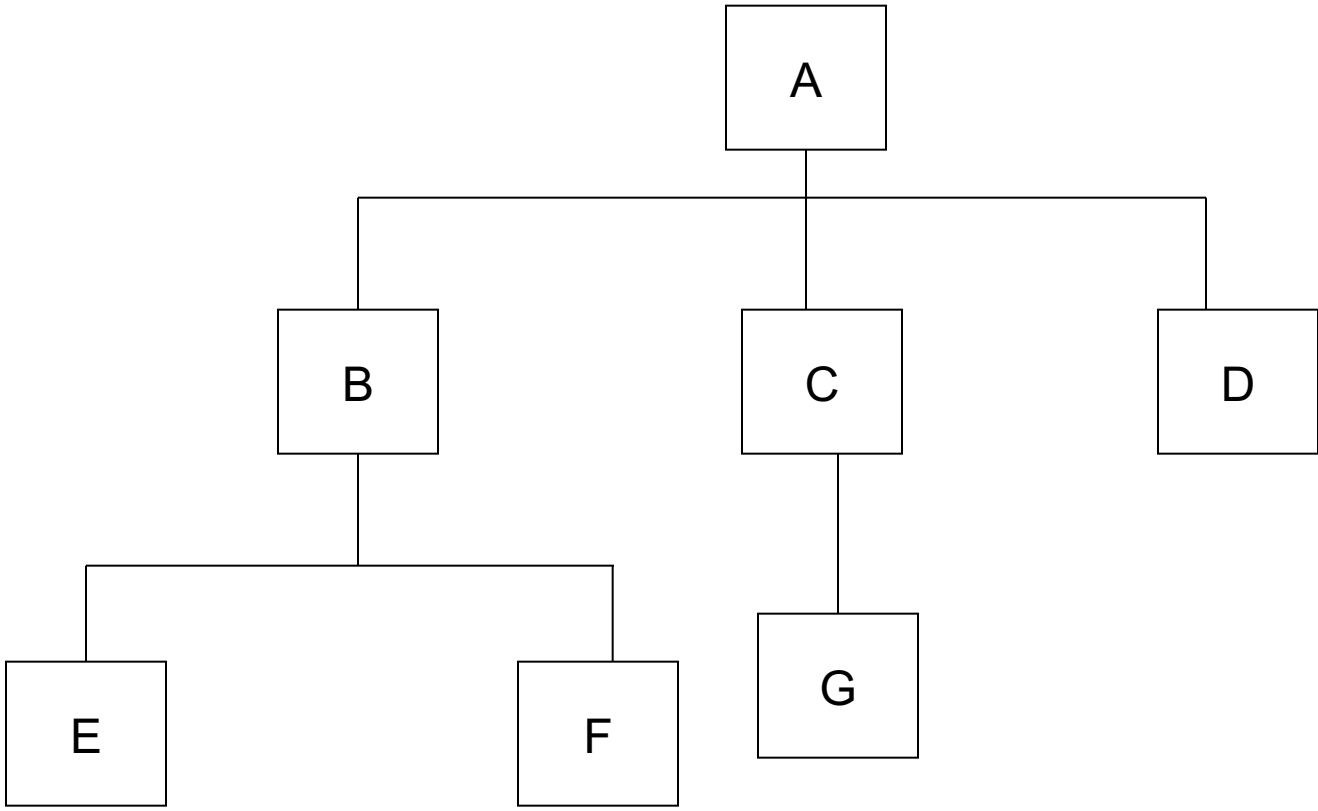
- Non incremental: big bang integration
- Incremental:

❑Top down integration

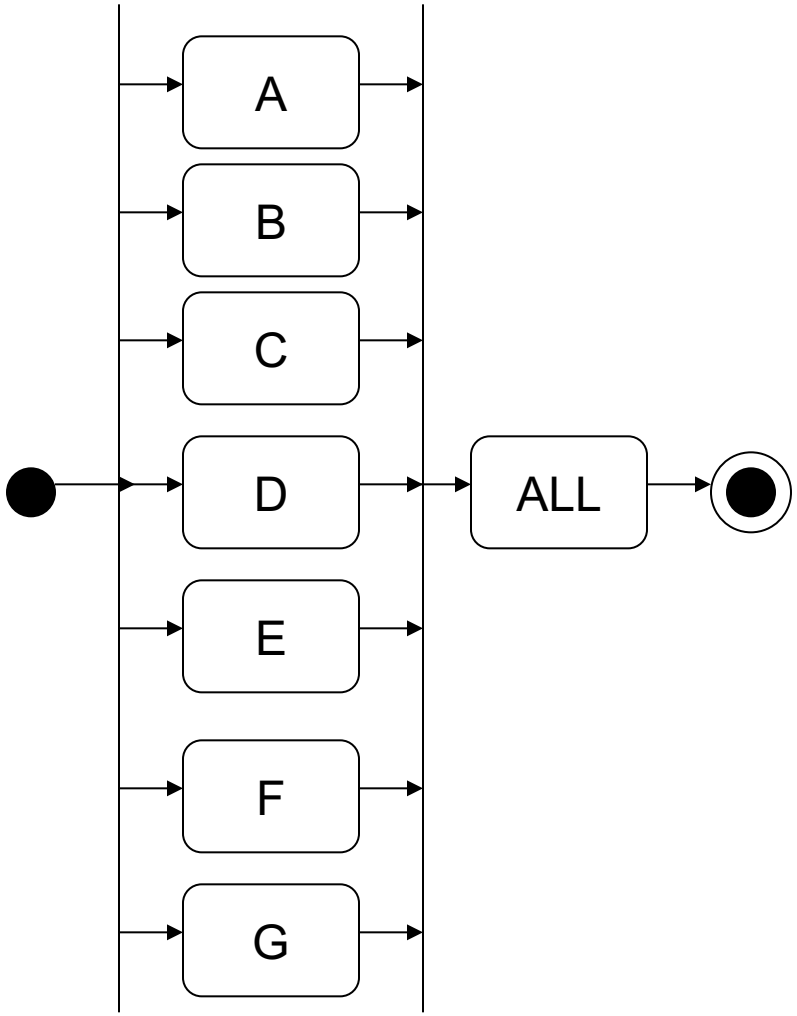
❑Bottom up integration

❑Hybrid integration

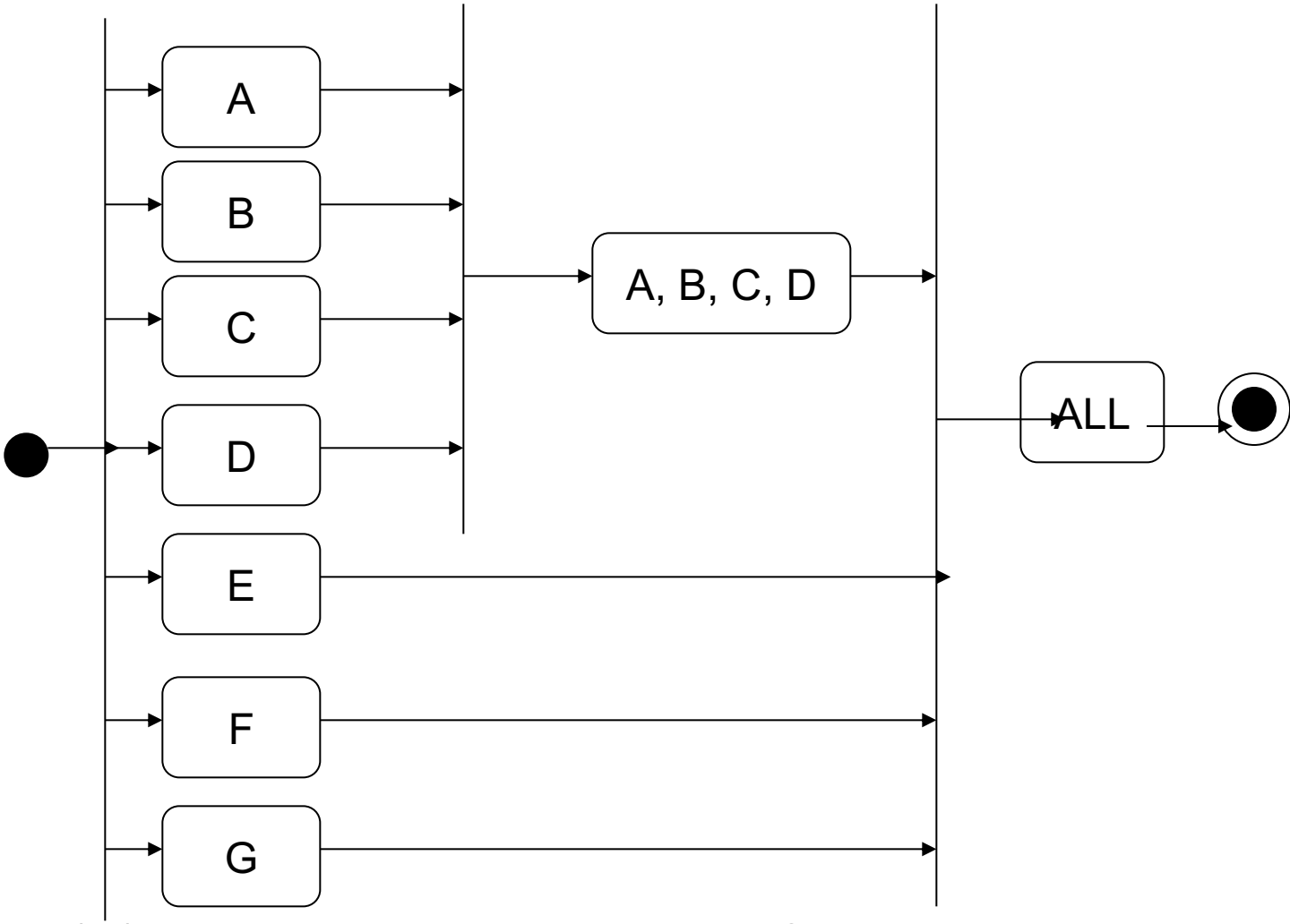
# An example



# Big-bang integration testing



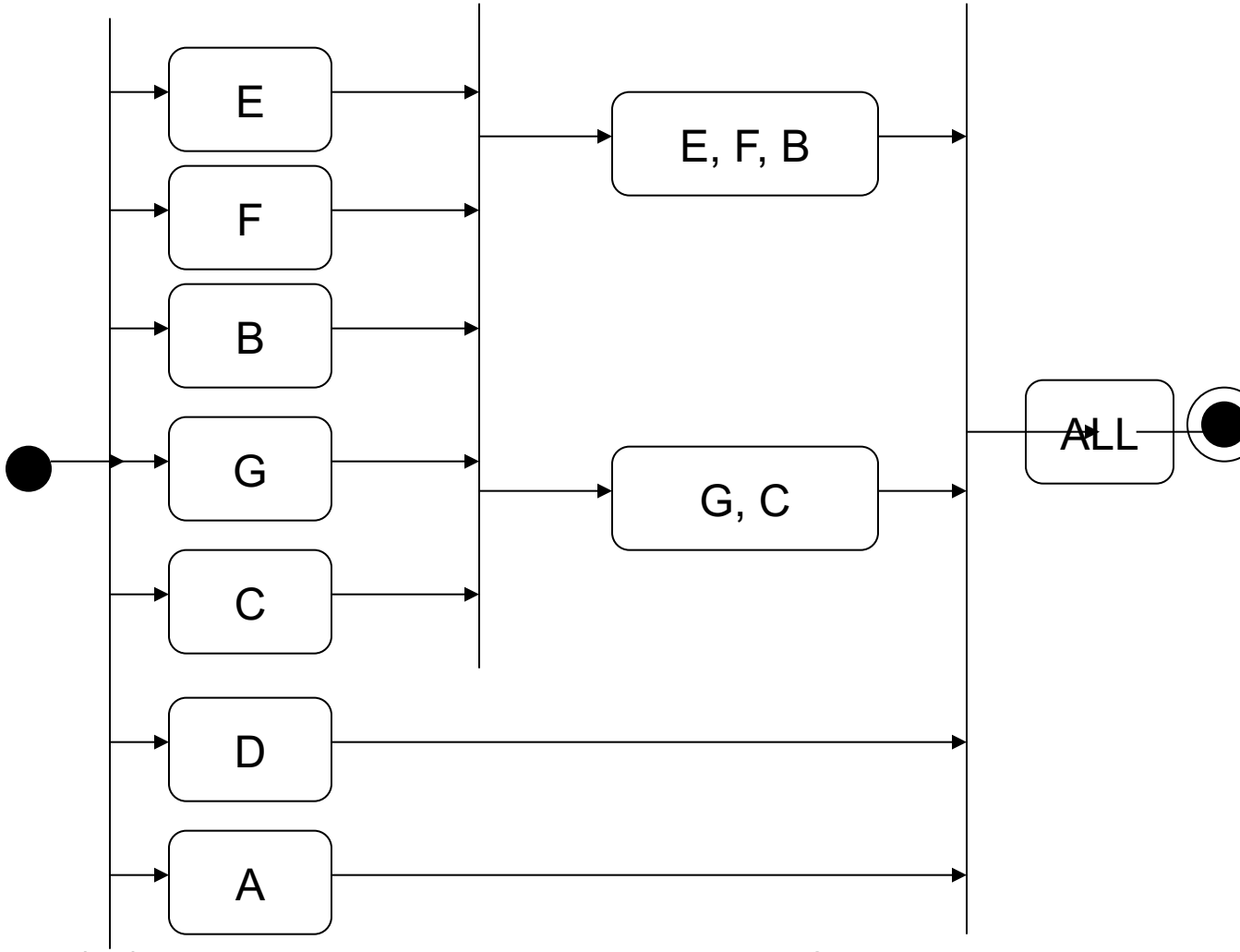
# Top down integration testing



# Top down integration testing

- Help the progress to users and senior management
- It is more likely to discover errors at early stage
- Test stubs are needed (what is a test stub?)
- The actual output cannot be observed until the lowest level modules have been finished

# Bottom up integration testing



# Bottom up integration testing

- The higher level view is not available and cannot be tested at early stage
- More likely to detect errors at the lower level
- Test drivers are needed (*what is the different between test drivers and test stubs?)*
- The actual output can be observed once the lowest level modules have been finished

# Hybrid Integration Testing

- Mix between top down and bottom up (sandwich)
- One possible way:
  - ❑ B-E-F and C-G
  - ❑ All
- Other possible way:
  - ❑ A-B, A-B-E, A-B-E-F
  - ❑ A-B-E-F-C, A-B-E-F-C-G
  - ❑ All

# What are STUBS and DRIVERS?

**Stubs** and **Drivers** are used in **integration testing** for **TOP-DOWN** and **BOTTOM-UP** testing respectively

# What is Top-Down Development?

- Top-Down development: We first create the “root” module and after that each module (child) that is called by the “root”.
- Next we move down the tree and consider each child as a root for the next modules that need to be developed. We repeat these steps until we reach the “leaf” modules

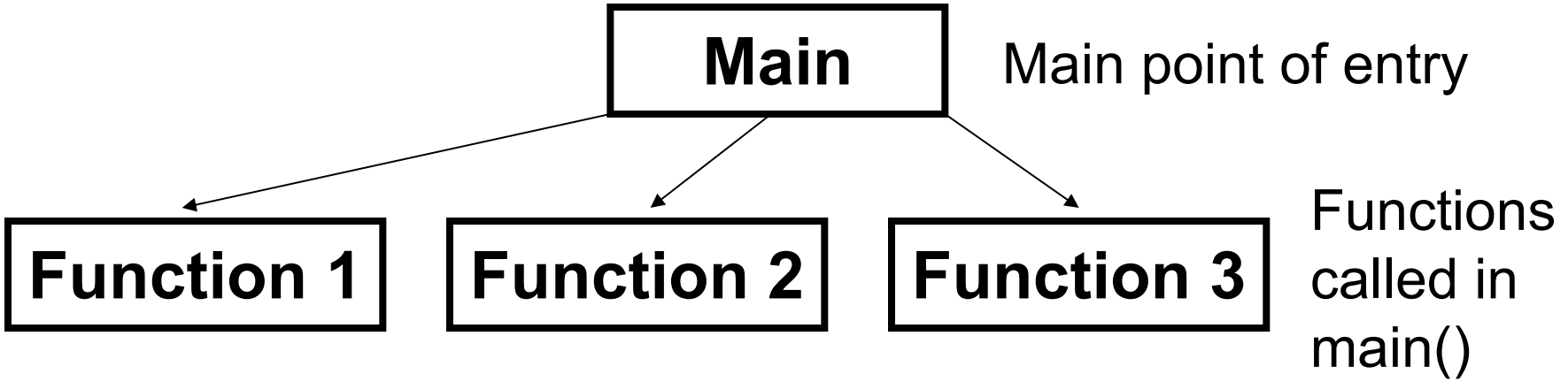
# What is Bottom-Up Development?

Bottom-Up development: Considering the same tree of modules described in the previous screen, we first develop the leaf modules. Next, we move up one level and develop the parent-modules (the ones that call the leaf modules). After each level is complete, we move up one level and develop it, until we reach the “root” module.

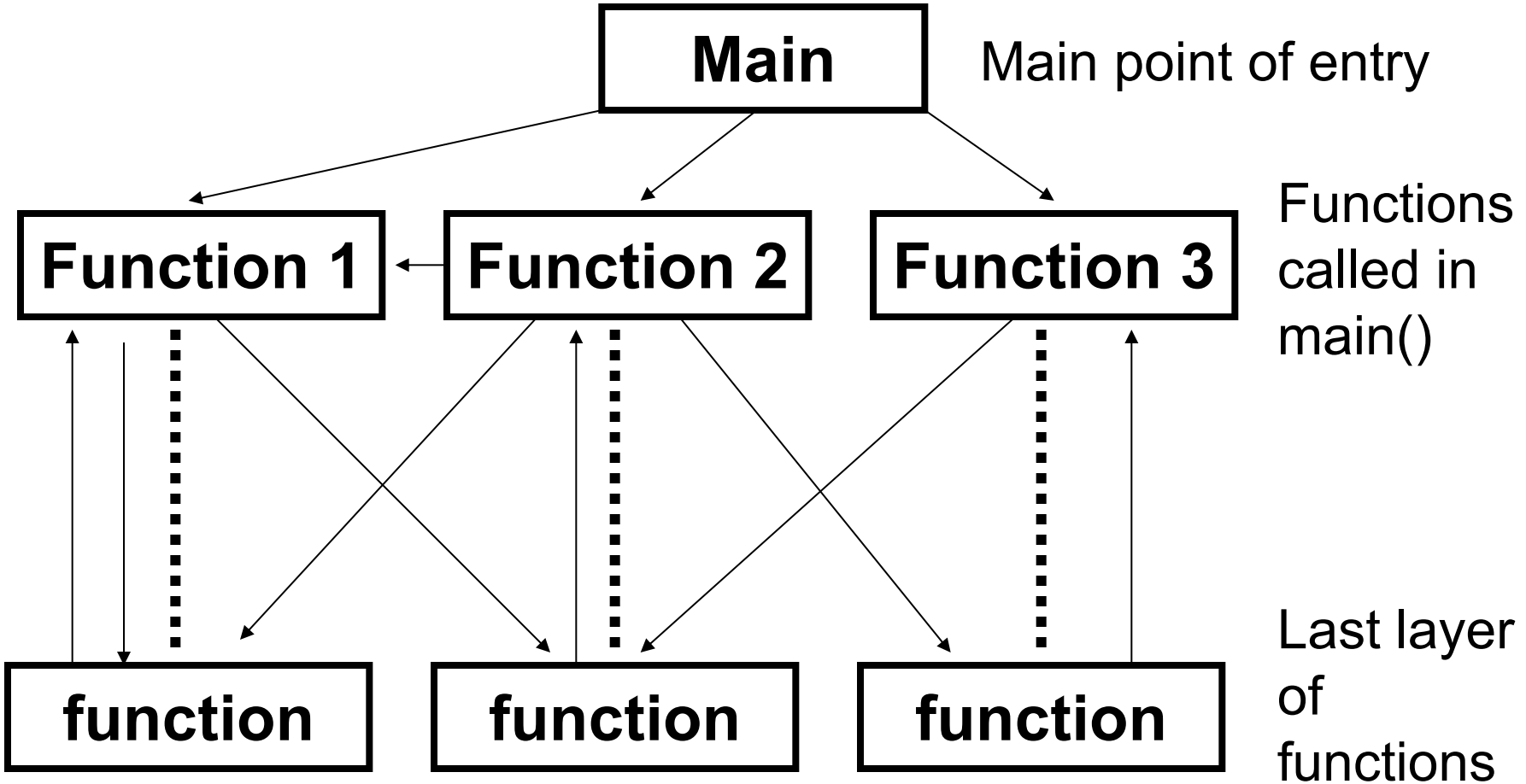
# Context –A Normal Program


Main point of entry

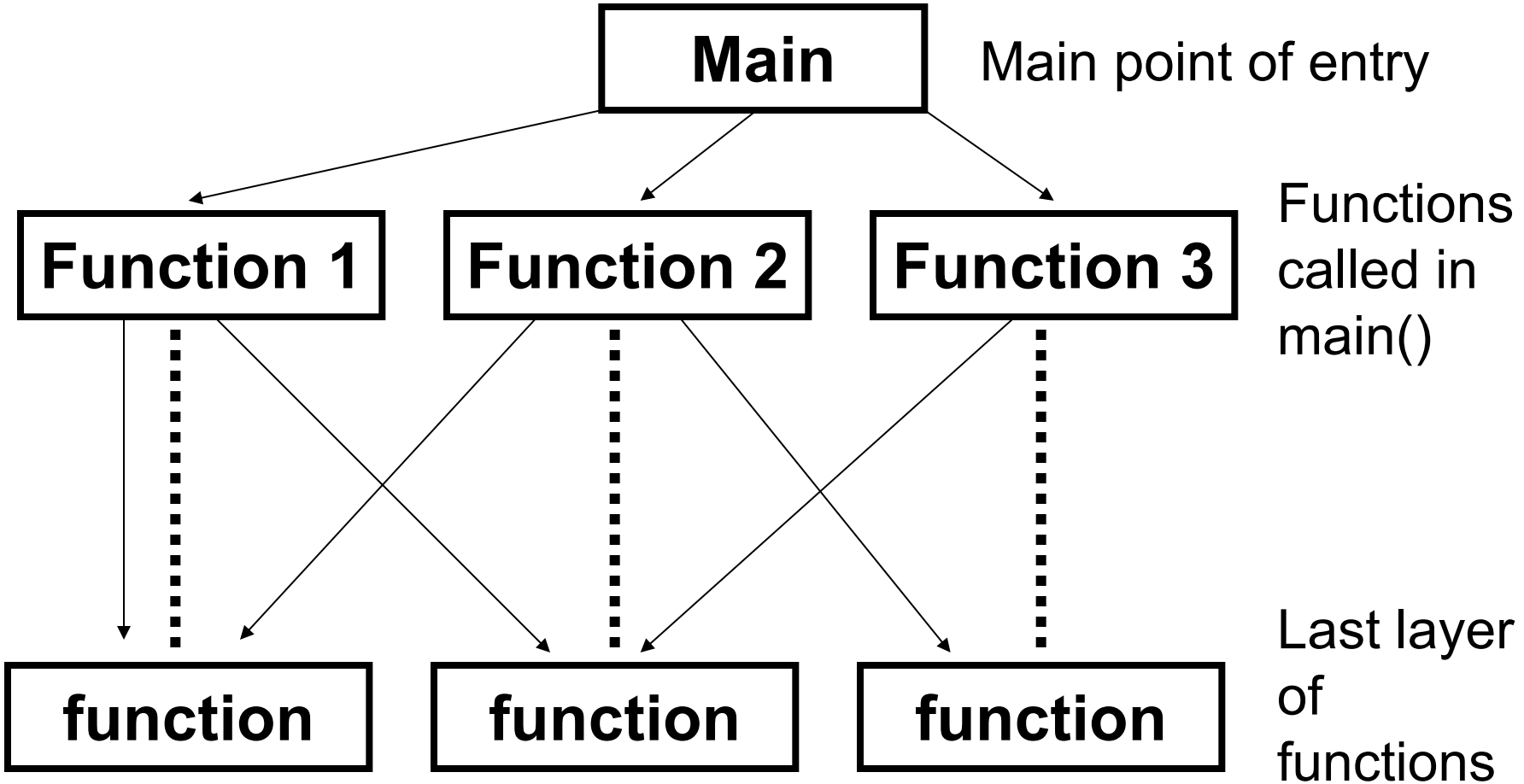
# Context –A Normal Program

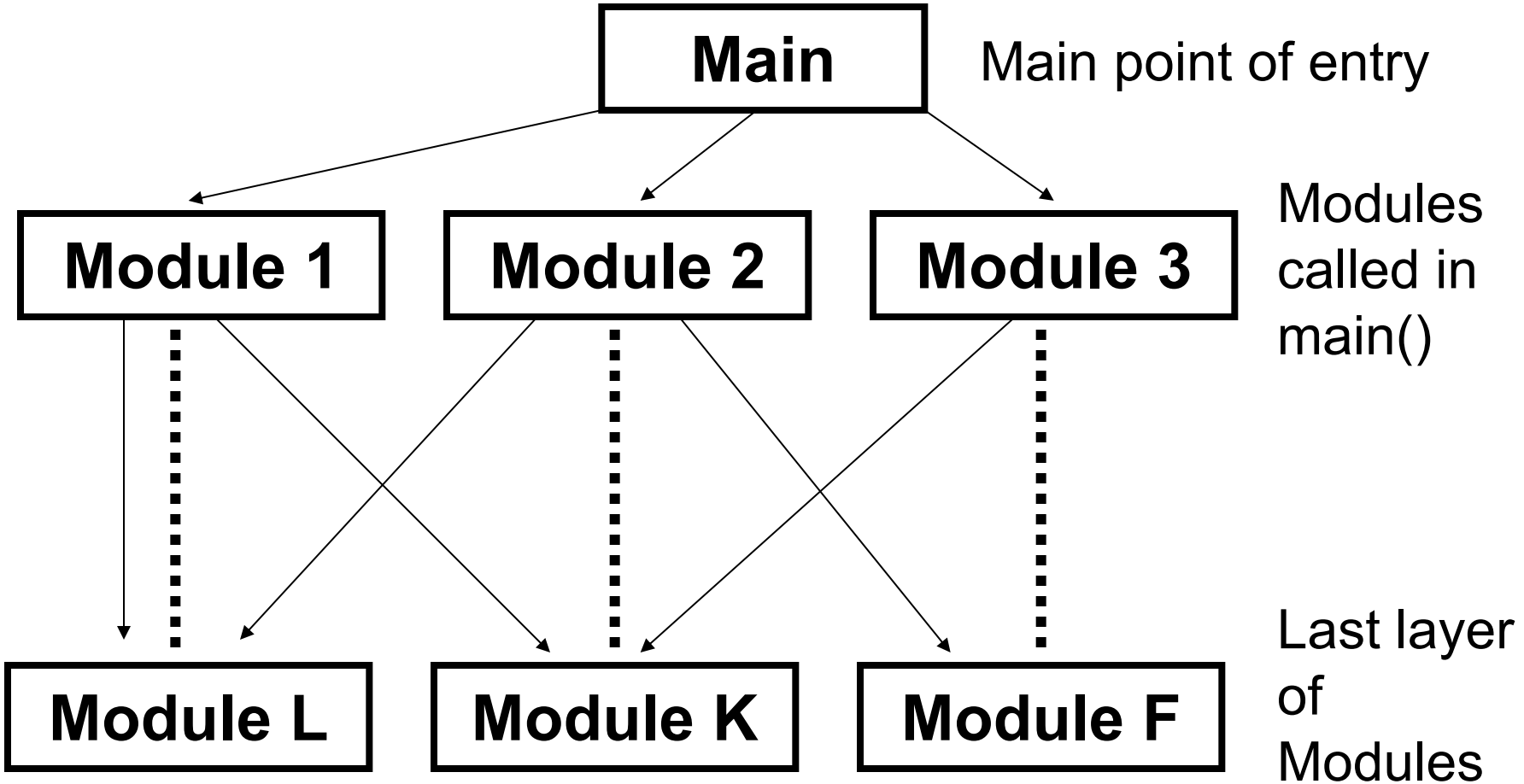


# Context –A Normal Program

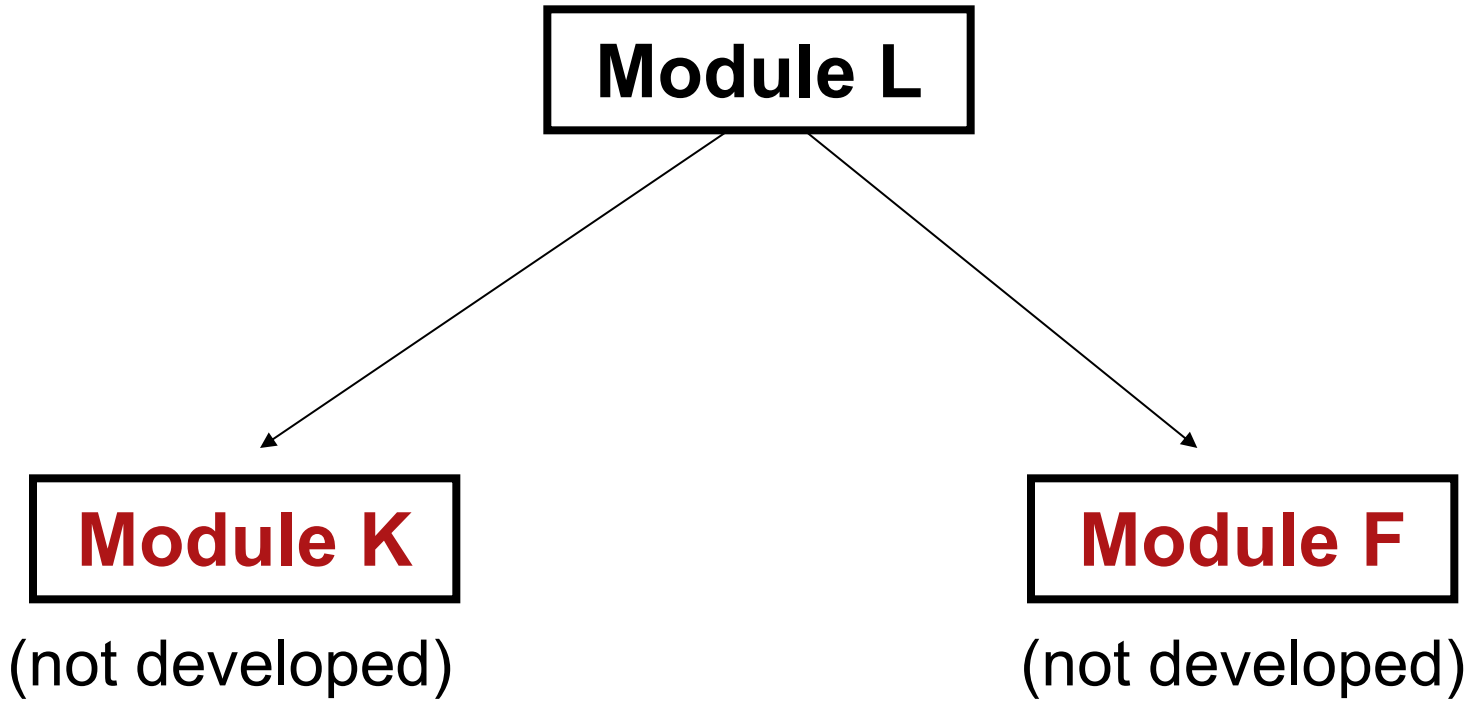


# Context – No circular dependencies



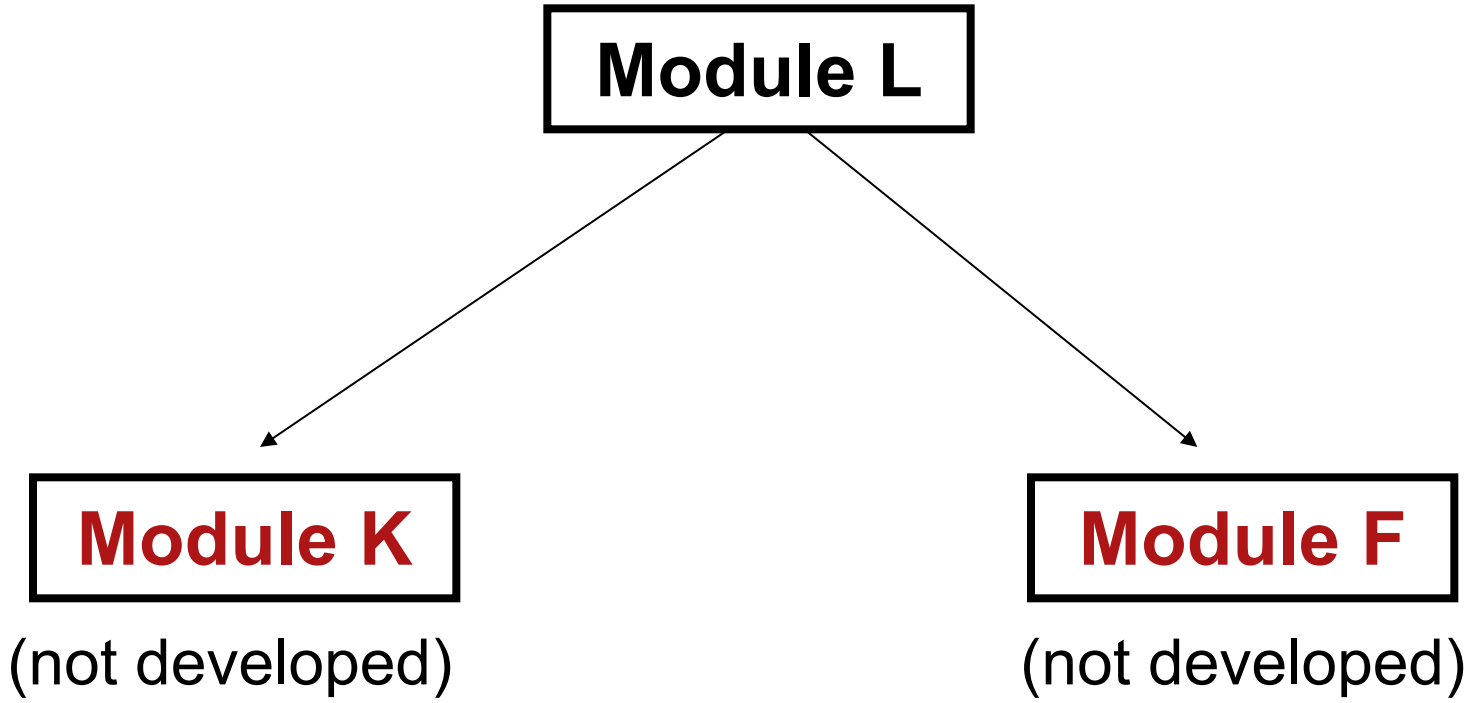

Main
Main point of entry
Module 1
Module 2
Module 3
Modules called in main()
Module L
Module K
Module F
Last layer of Modules

# A Tipical Example

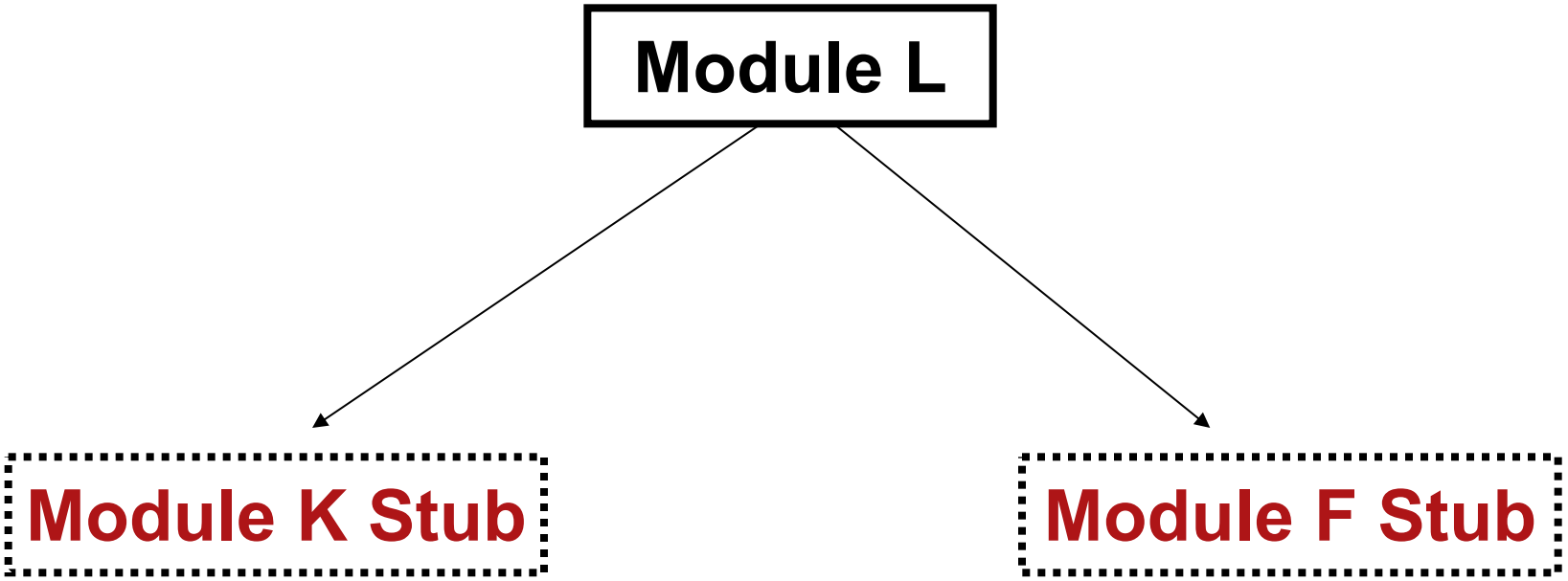


# A Tipical Example

So how can we test **MODULE L?**



# A Tipical Example

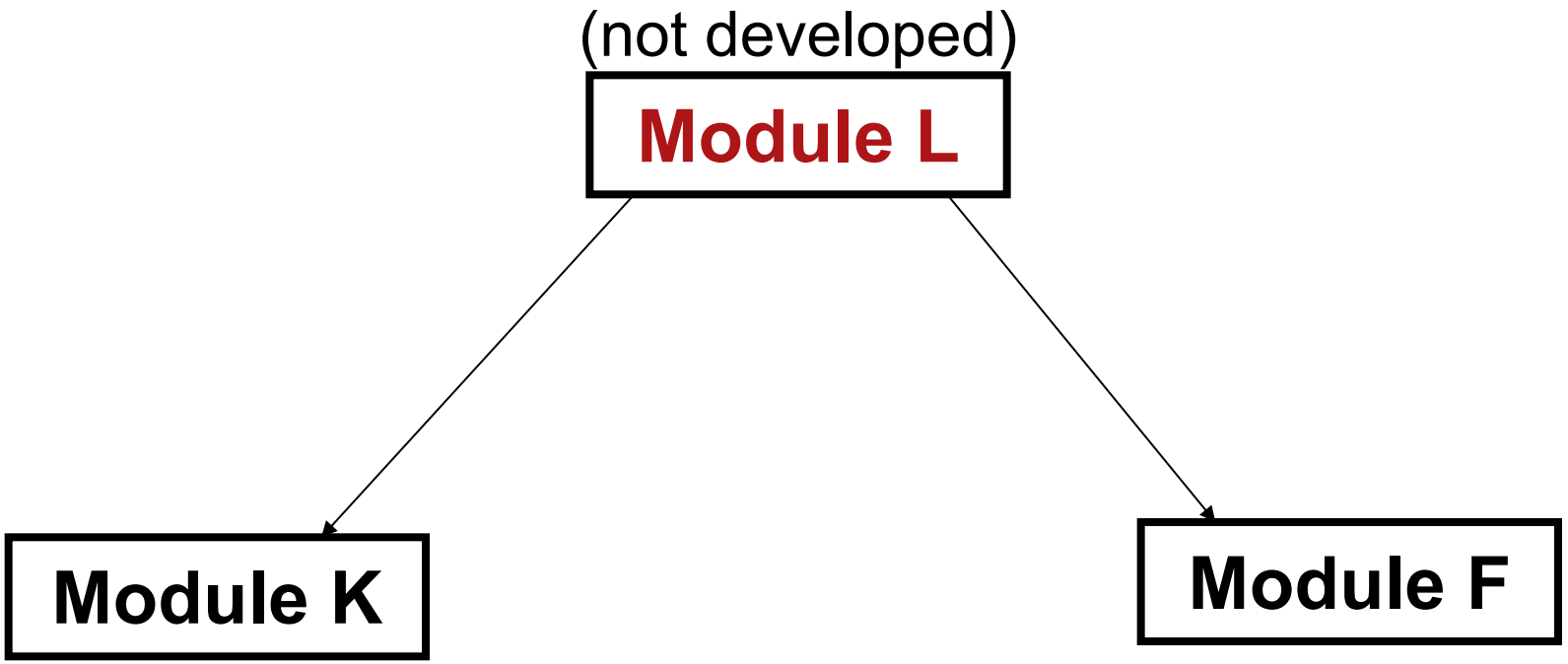



Dummy code that **SIMULATES** the functionality of the undeveloped modules

# Code Example - Stub

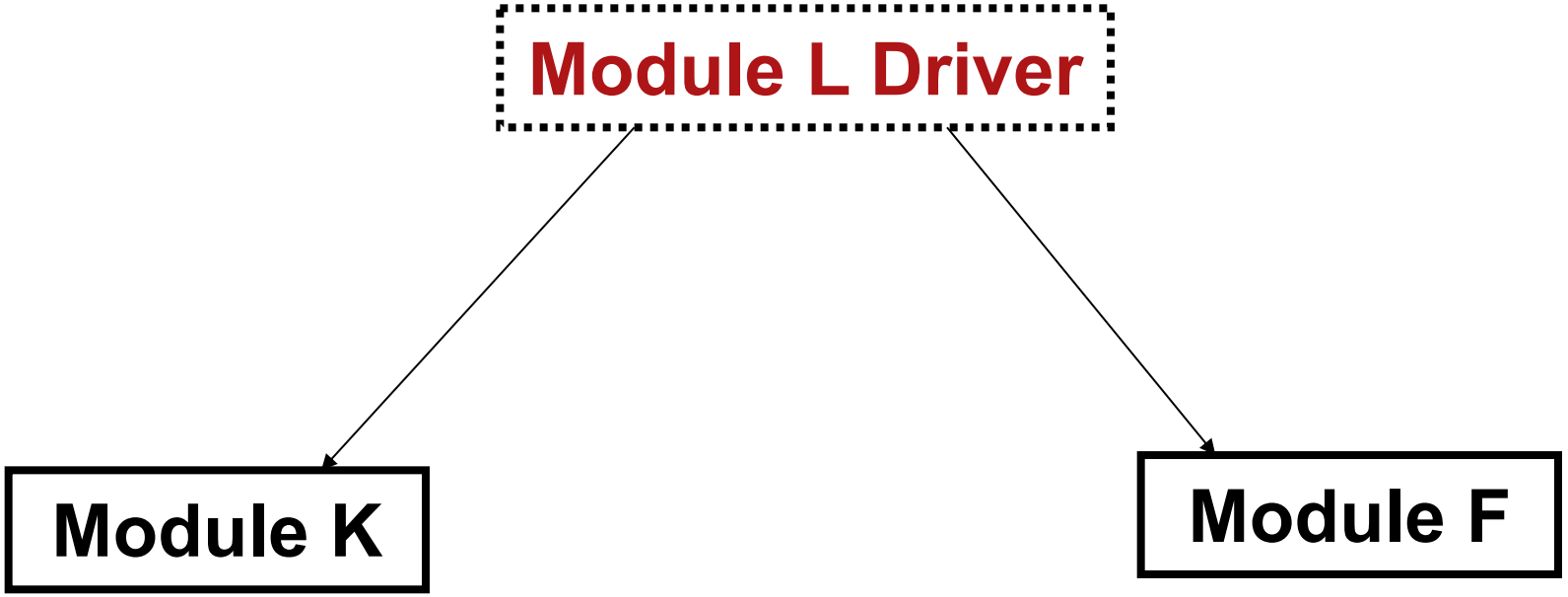
```
void functionWeTest(params..) {

        ……...

        int p = price(param1);

        ……..

}

void price(int param) { //this is the stub

        return 10; // We don't care what the price is. We just need a value so we can test the other function

}
```

# Let's Move to Drivers



# Let's Move to Drivers

Dummy code that **returns** values from Module L and Module K



# A Tipical Example

```
void functionThatCallsPrice (params..)
{ //this is the driver
        int p = price(param1);
        printf("Price is: %d", p);
}
void price(int param) {
        //complex equations and DB
integrations that determine the real
price
}
```

# Conclusions

1. No need to write all the modules in order to begin testing
2. Stubs and Drivers are usually easy to create
3. The hard part is identifying what is essential and what can be simulated
4. Stubs and Drivers can be thrown away once the equivalent module is created

# c) System Testing

- Function testing: black box testing is included here by some experts. Related to the functional testing
- Performance Testing: non functional testing. This includes: volume testing, load/stress testing, reliability testing, security testing

# When testing is stopped ?

- No definite answer
- Possible answers:

- When testing resources are run out
- When the level of reliability as specified in SRS are reached
- When the test manager is confident on the quality of the software

# References


- Hetzel, W. (1973) Program Testing Method, Prentice-Hall
- IEEE(1983) IEEE standard for software testing documentation, IEEE Std 829-1983
- IEEE(1990) IEEE standard for software testing documentation, IEEE Std 610.12-1990
- Pfleeger, S.L.(2001) Software Engineering: Theory and Practice, 2nd ed. Prentice-Hall

Thank You ☺