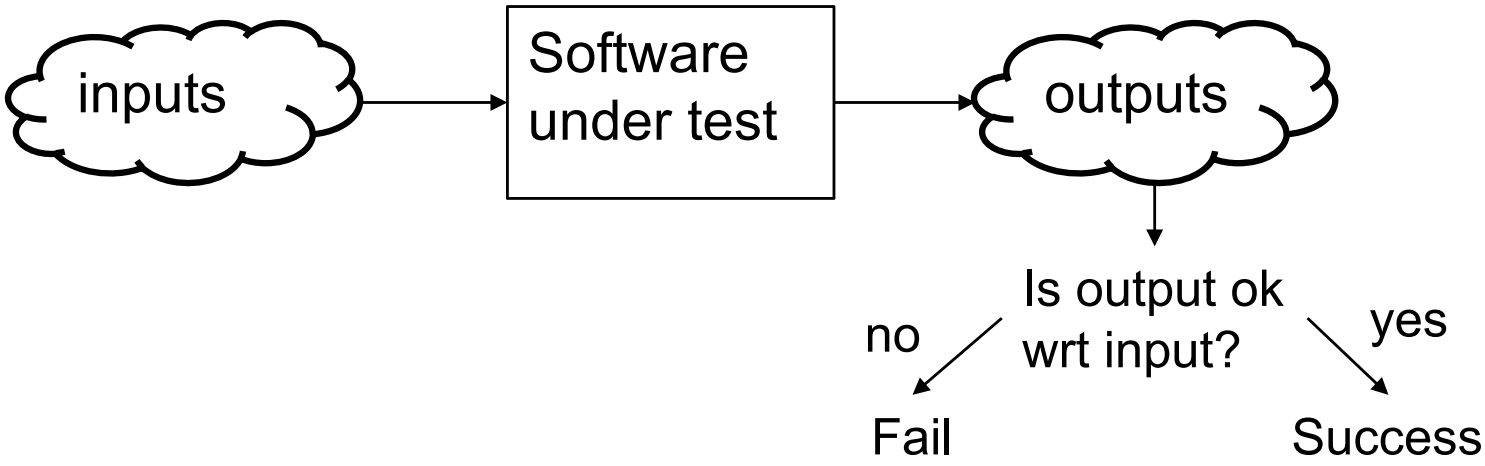
Politecnico
di Milano

Testing

Testing

- Program testing can be used to show the presence of bugs, but never to show their absence. (Dijkstra 1972)



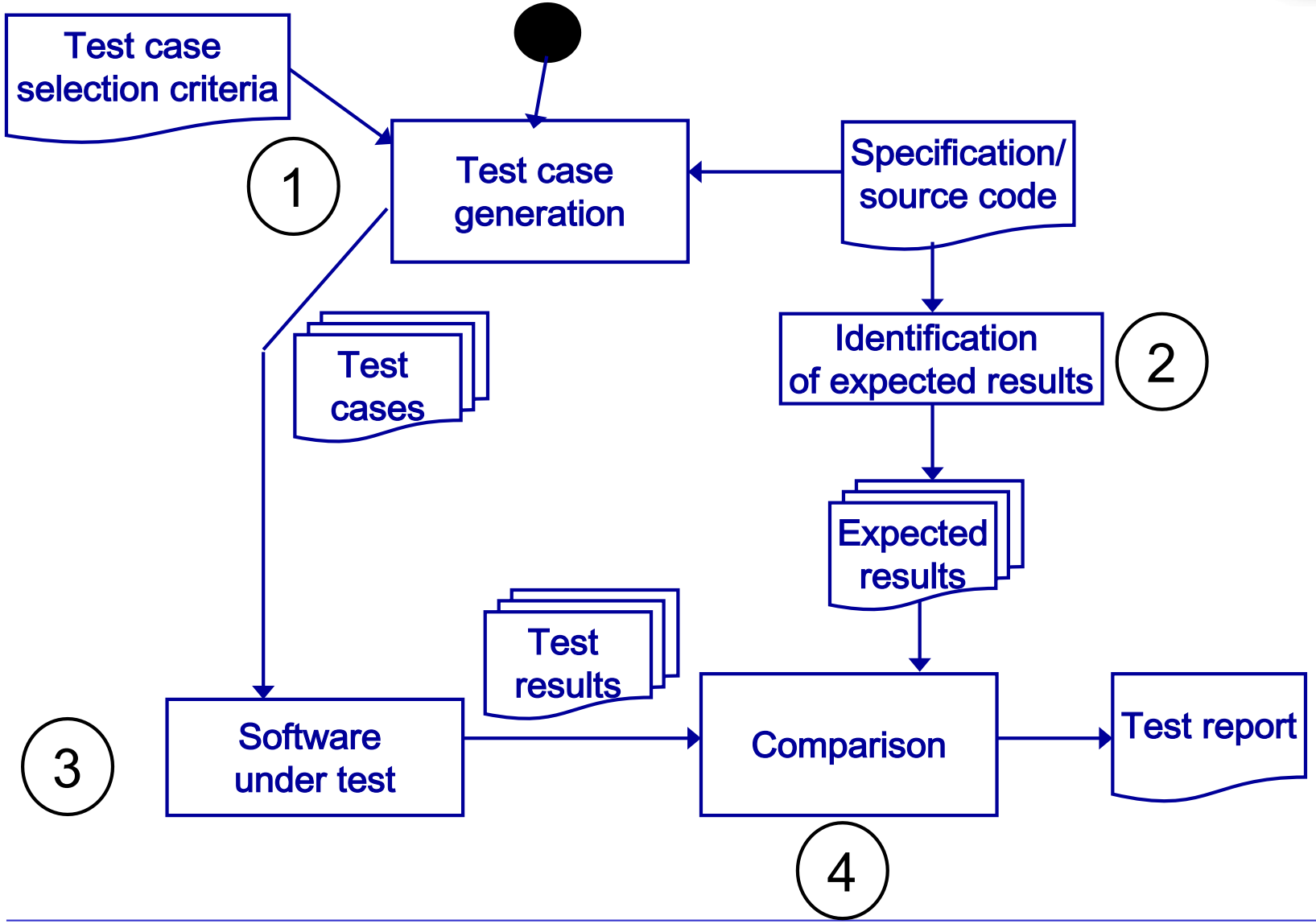
Definitions

- A test case t includes
  - A set of inputs for the system
  - A hypothesis on the state of the system at the time of the test case execution
  - The expected output
- Test set T
  - A set of test cases
- If P is our system
  - A test case t is *successful* if P(t) is correct
  - A test set T is *successful* if P correct for all t in T

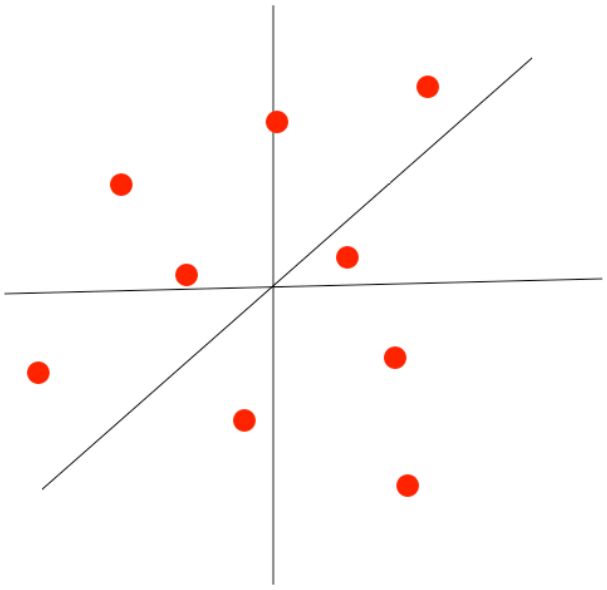
How to select test sets?

- Randomly → random testing
  - It is "blind", it does not "look for bugs"
  - But it can be effective as it allows testers to generate and execute hundreds of thousand of test cases
  - Using statistics it can be a powerful tool
- Systematically → systematic testing
  - Use characteristics/structure of the software artifacts (e.g., code) – white-box testing
  - Use information on the behavior of the system (e.g., specifications) – black-box testing

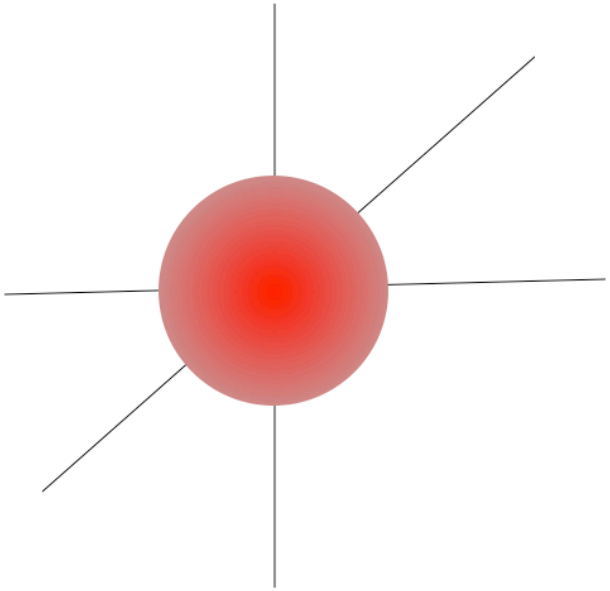
The testing process



Testing vs Analysis



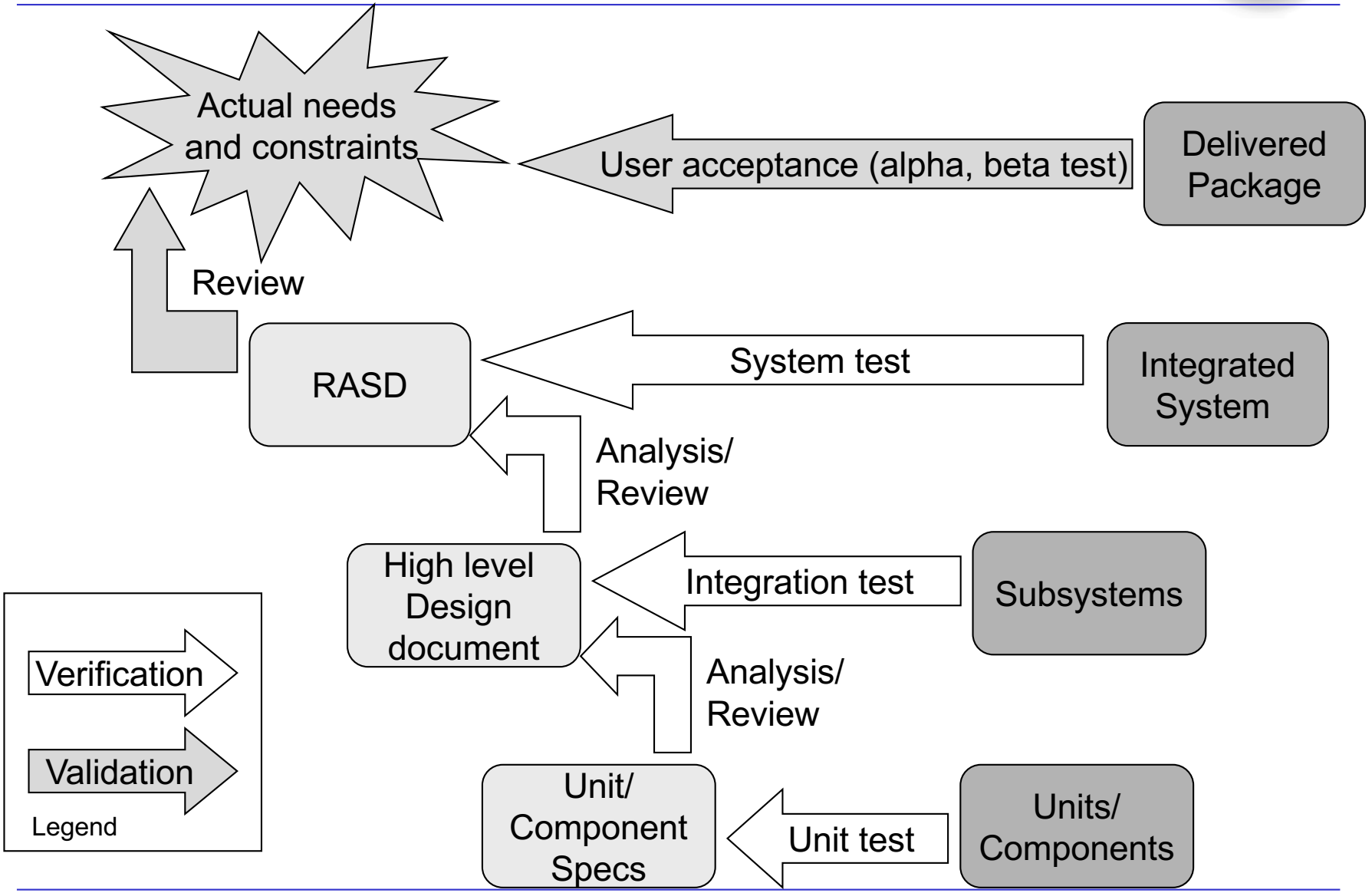
testing:
a few cases of arbitrary size



scope-complete:
all cases within a small bound

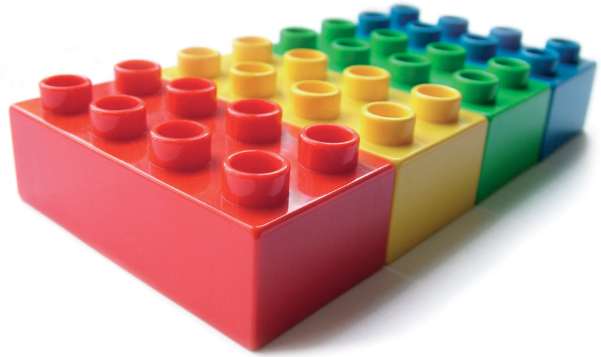
Testing activities

V&V activities and software artifacts (the V model)



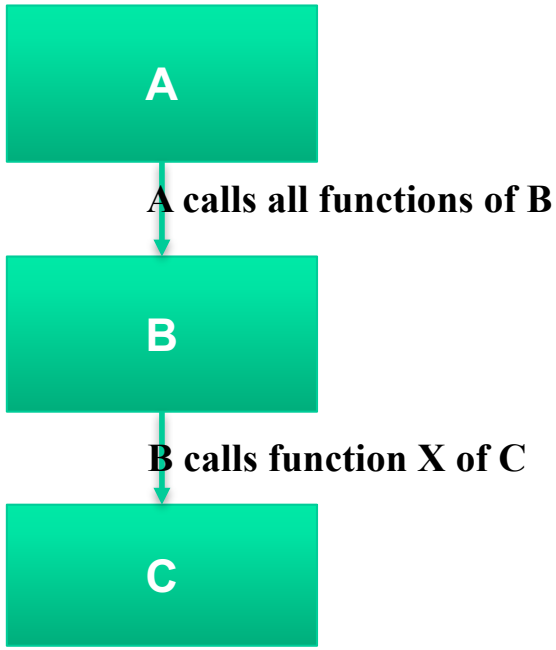
Unit Testing

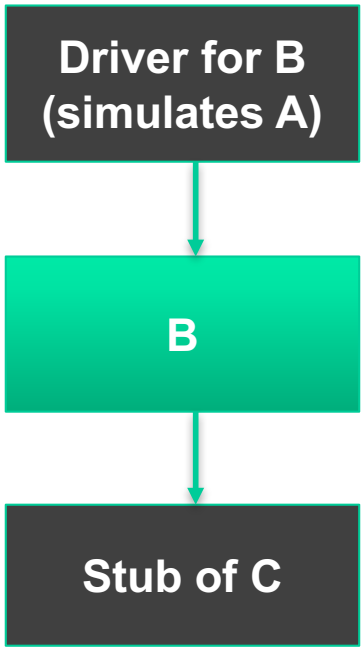
- Conducted by the developers themselves
- Aimed at testing sections of code
- Why unit testing?
  - Find problems early
  - Facilitate changes
  - Guide the design
  - Visual feedback



Unit testing and scaffolding

- The problem: components may not work in isolation
- Thus, we need to simulate components that are missing.
  - Example: B is used by A and uses C. We want to unit test B





Oracle: checks that, after test execution, the system is in a correct state and the output is correct

Integration Testing

- Aimed at exercising
  - interfaces
  - modules' interactions

Integration Faults

- Inconsistent interpretation of parameters or values
  - Example: Mixed units (meters/yards) in Mars Climate Orbiter
- Violations of value domains, capacity, or size limits
  - Example: Buffer overflow
- Side effects on parameters or resources
  - Example: Conflict on (unspecified) temporary file
- Omitted or misunderstood functionality
  - Example: Inconsistent interpretation of web hits
- Nonfunctional properties
  - Example: Unanticipated performance issues
- Dynamic mismatches
  - Example: Incompatible polymorphic method calls

Example from the Apache web server

- Apache web server, version 2.0.48
- Response to normal page request on secure (https) port
- Which problem do you have here?

Impossible to find it with unit testing.
Inspection and some dynamic techniques can find it.

```
static void ssl_io_filter_disable(ap_filter_t *f)
{   bio_filter_in_ctx_t *inctx = f->ctx;

    inctx->ssl = NULL;
    inctx->filter_ctx->pssl = NULL;
}
```

No obvious error, but Apache leaked memory slowly (in normal use) or quickly (if exploited for a DOS attack)

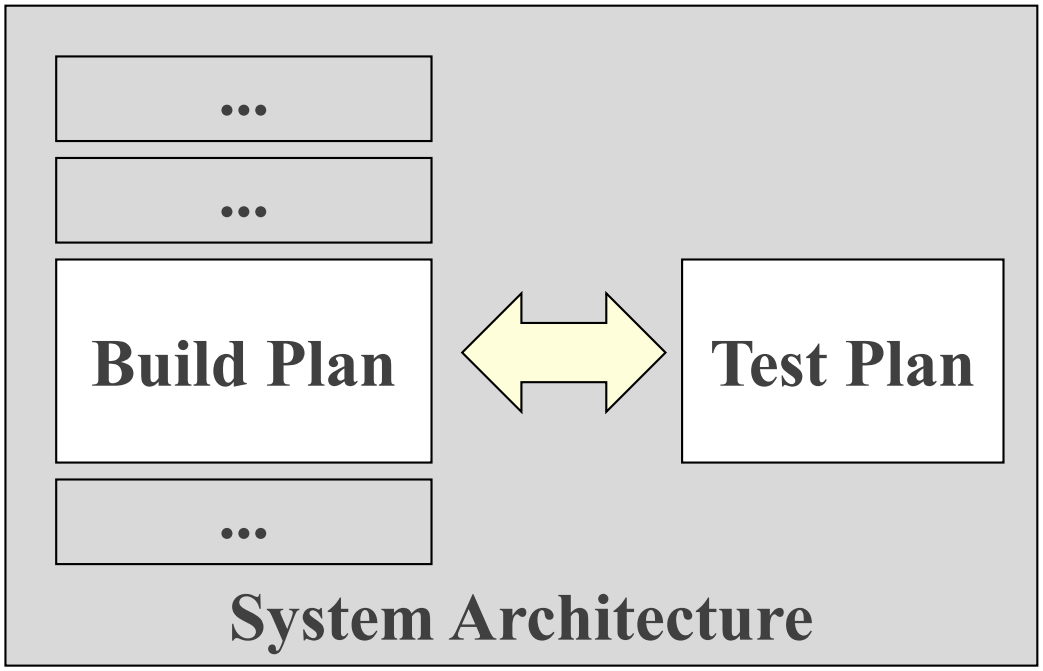
Example from the Apache web server

- Apache web server, version 2.0.48
- Response to normal page request on secure (https) port

```
static void ssl_io_filter_disable(ap_filter_t *f)
{   bio_filter_in_ctx_t *inctx = f->ctx;
    SSL_free(inctx->ssl);
    inctx->ssl = NULL;
    inctx->filter_ctx->pssl = NULL;
}
```

If the interface of the module where f is defined offers SSL_free and clearly specifies how it should be used, the problem could be found during integration testing

Integration Plan + Test Plan



- Integration test plan drives and is driven by the project "build plan"
  - A key feature of the system architecture and project plan

How to achieve integration & testing

- Big bang (better to avoid it)
- Incrementally (facilitates bug tracking)

Big Bang integration & testing

- An extreme and desperate approach: Test only after integrating all modules
  - Does not require stubs/drivers/oracles
    - The only excuse, and a bad one
  - Minimum observability, diagnosability, efficacy, feedback
  - High cost of repair
    - Recall: Cost of repairing a fault rises as a function of time between error and repair

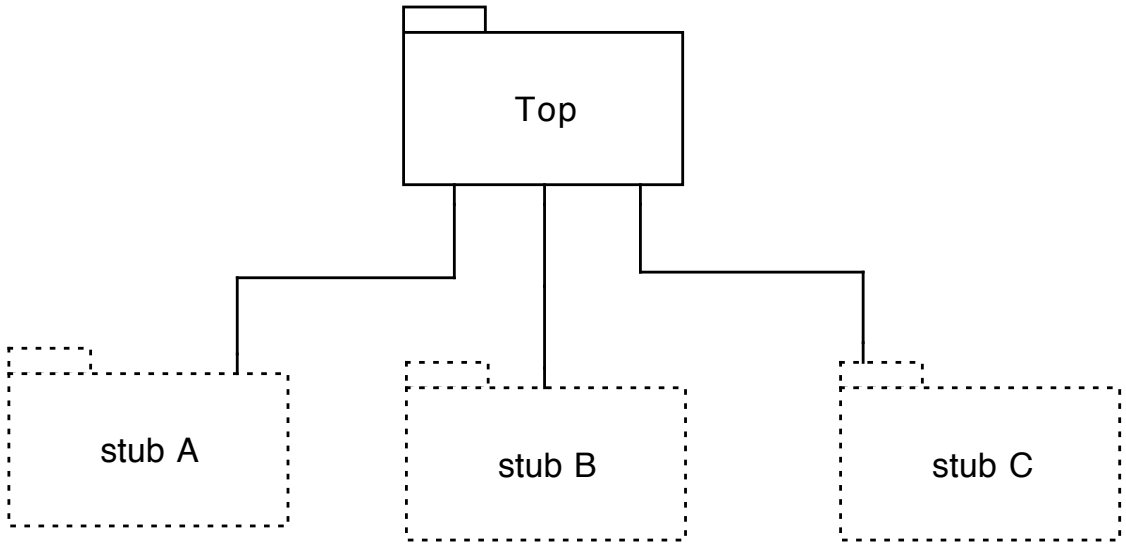
Incremental integration & testing

- It occurs while components are released
- E.g., as soon as a first version of components A and B is released, we integrate them and test the integration
- Then, if tests pass, we integrate also the first version of C that has been released in the meanwhile
  - And we test such new integration

Some strategies for integration & testing

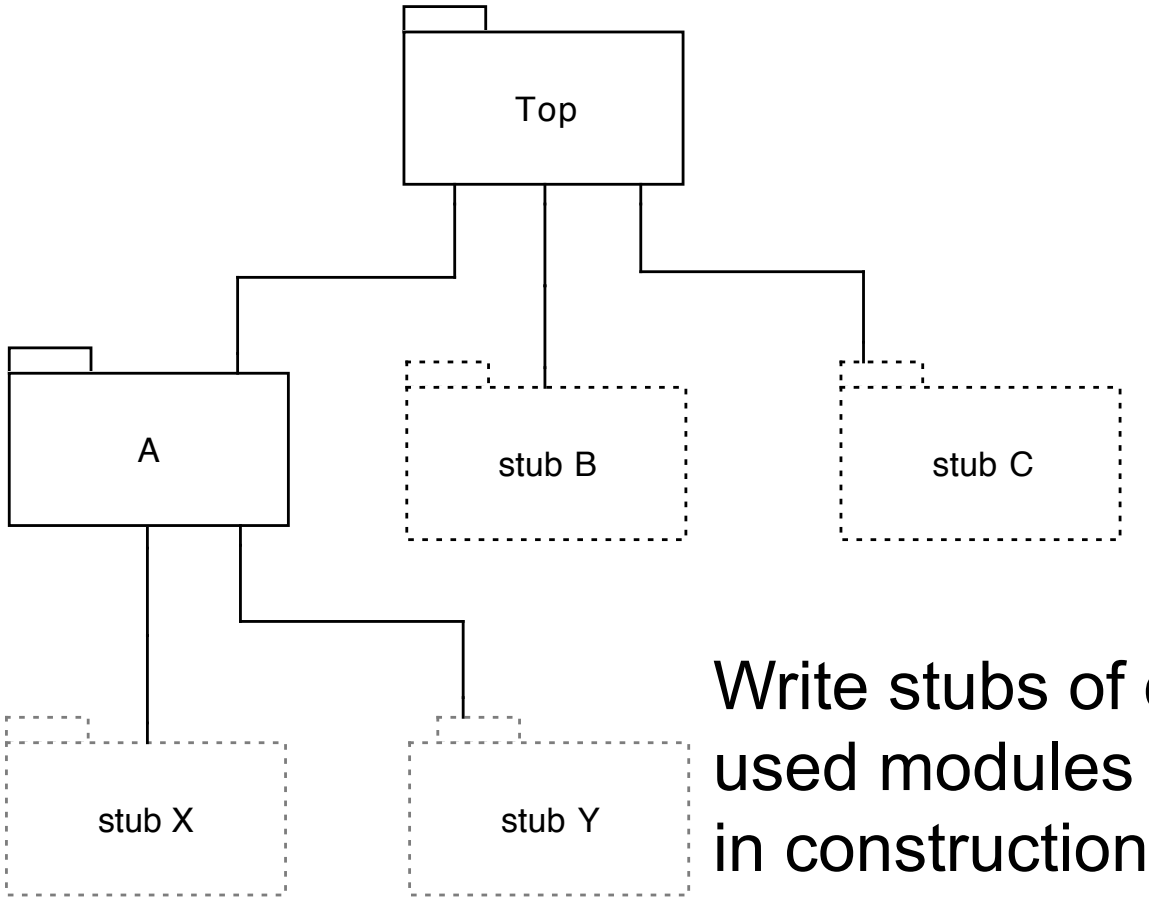
- Based on the hierarchical structure of the system
  - Top-down
  - Bottom-up
- Threads: a portion of several modules that offers a user-visible function
- Critical modules

Top down .



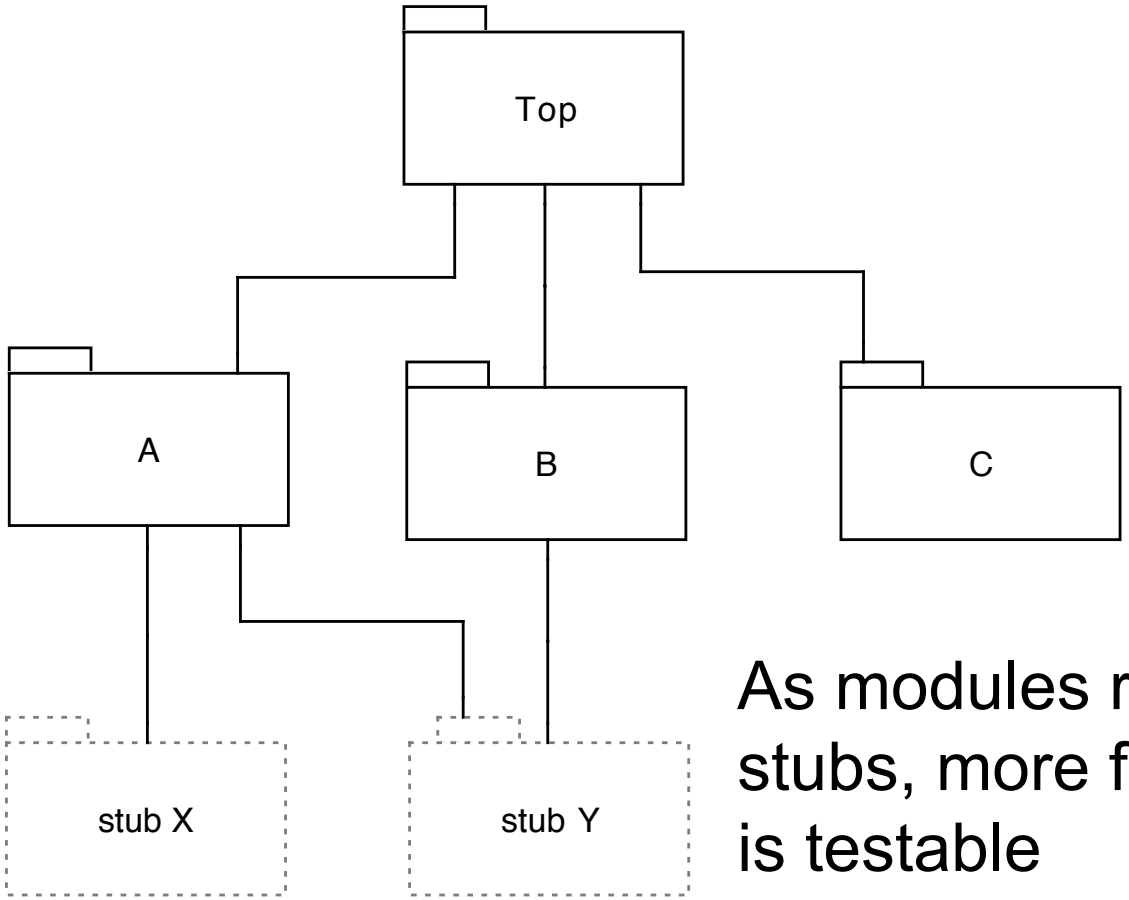
Working from the top level (in terms of “use” or “include” relation) toward the bottom.
No drivers required if program tested from top-level interface (e.g., GUI, CLI, web app, etc.)

Top down ..



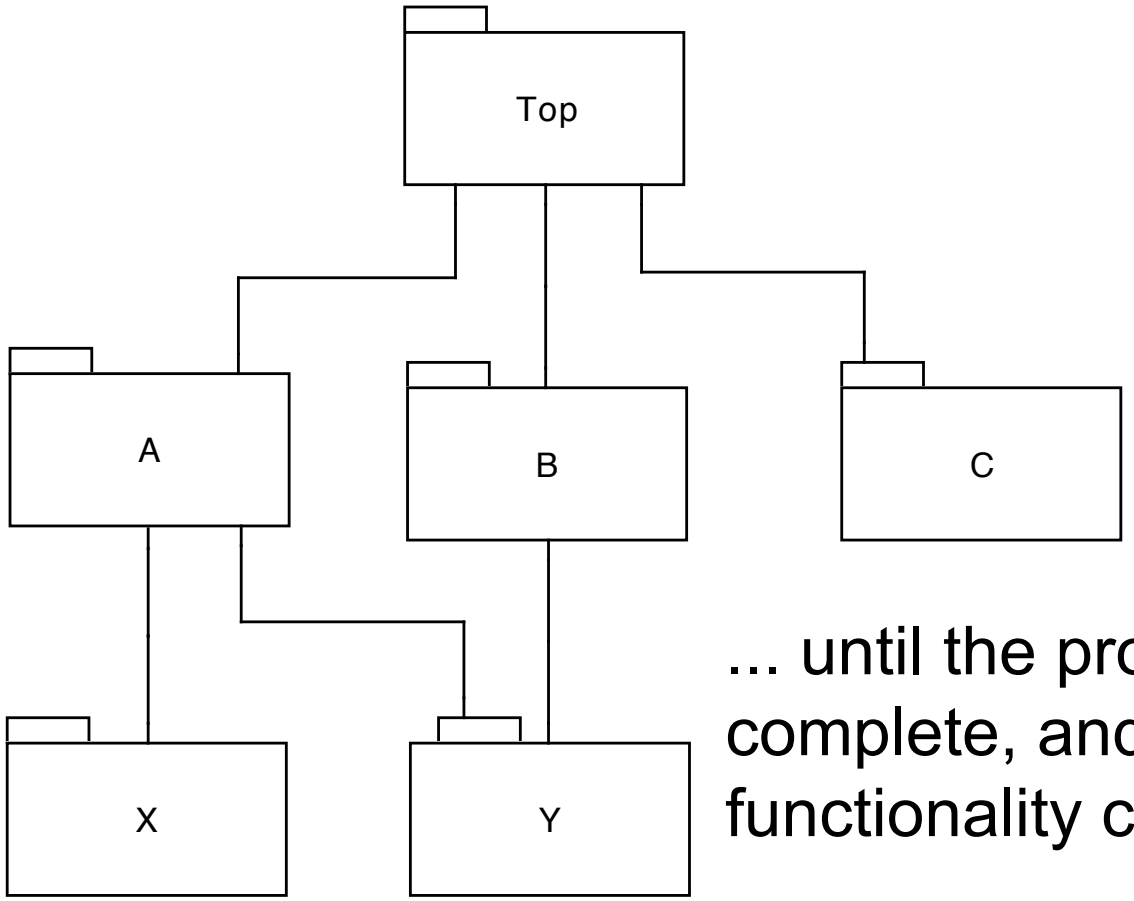
Write stubs of called or used modules at each step in construction

Top down ...



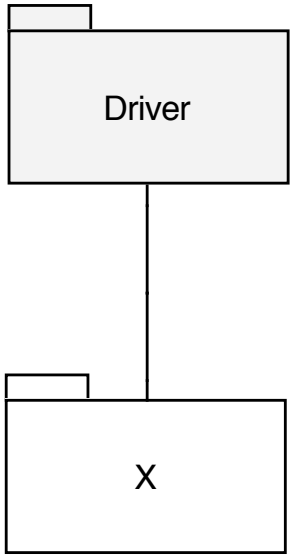
As modules replace stubs, more functionality is testable

Top down ... complete



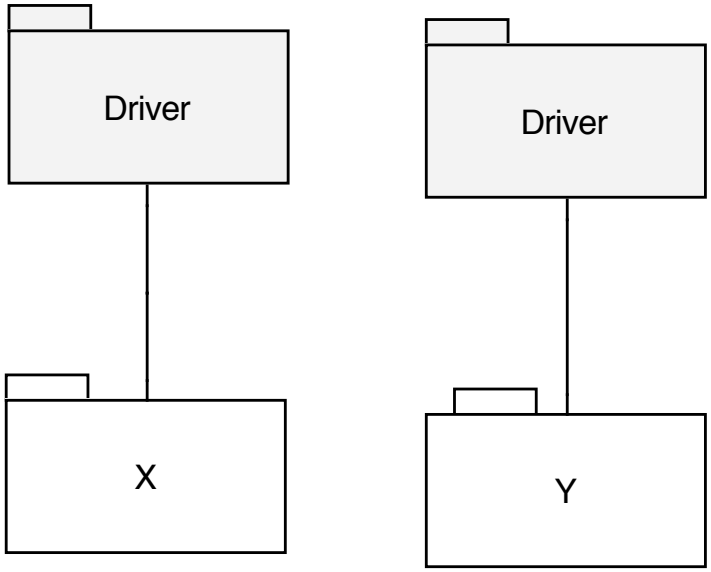
... until the program is complete, and all functionality can be tested

Bottom Up .



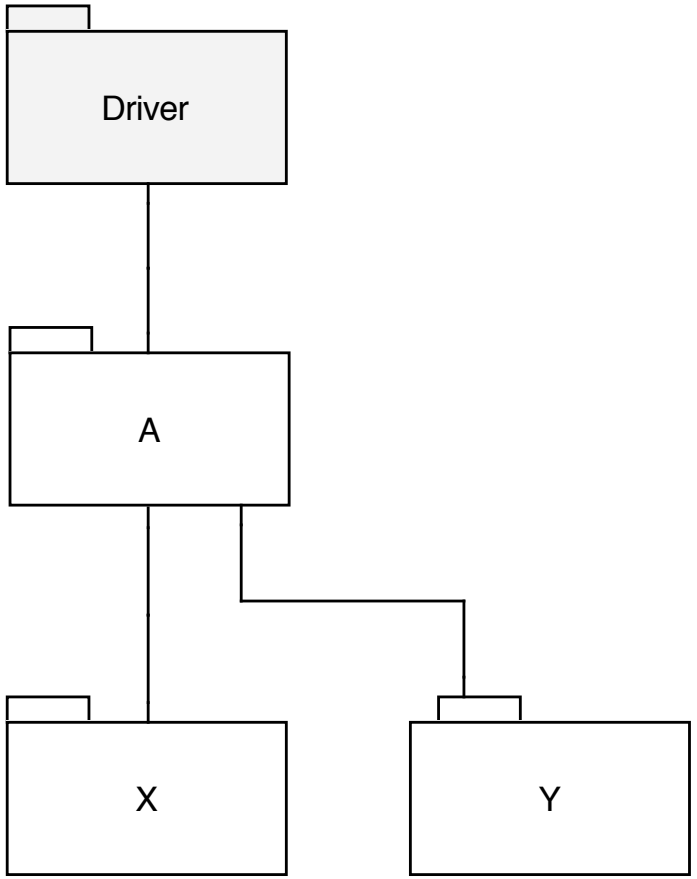
Starting at the leaves of the “uses” hierarchy, we never need stubs

Bottom Up ..



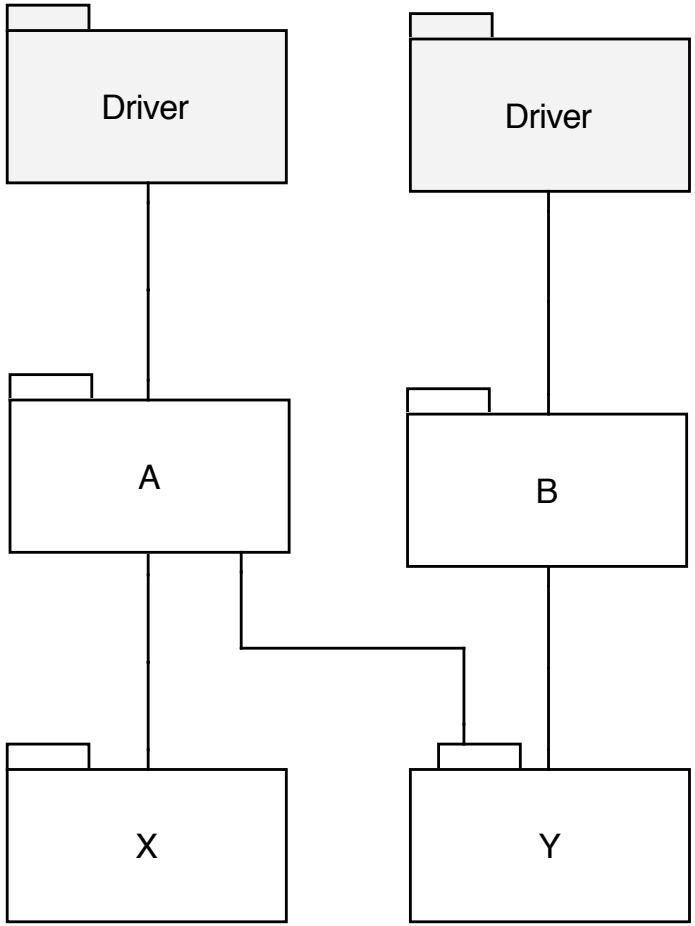
... but we must construct drivers for each module (as in unit testing) ...

Bottom Up ...

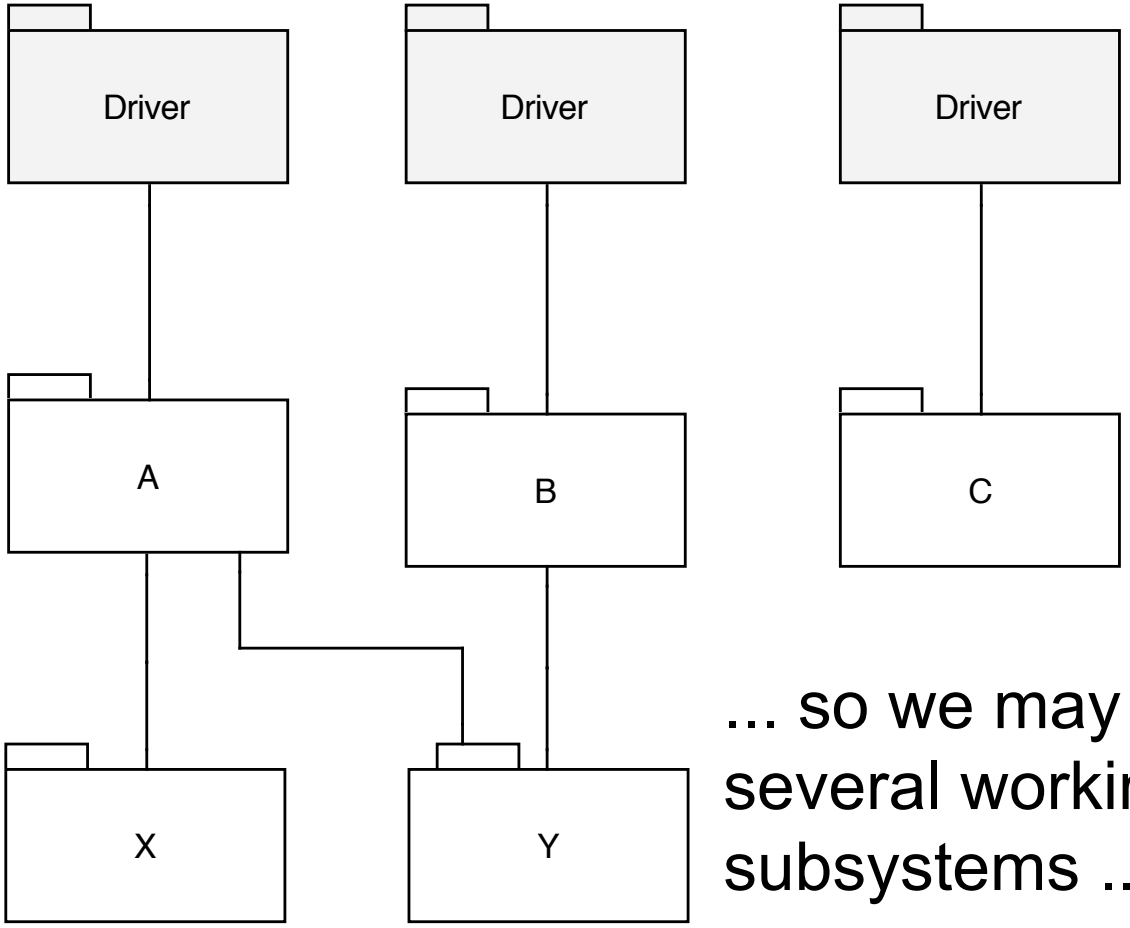


... an intermediate module replaces a driver, and needs its own driver ...

Bottom Up ....

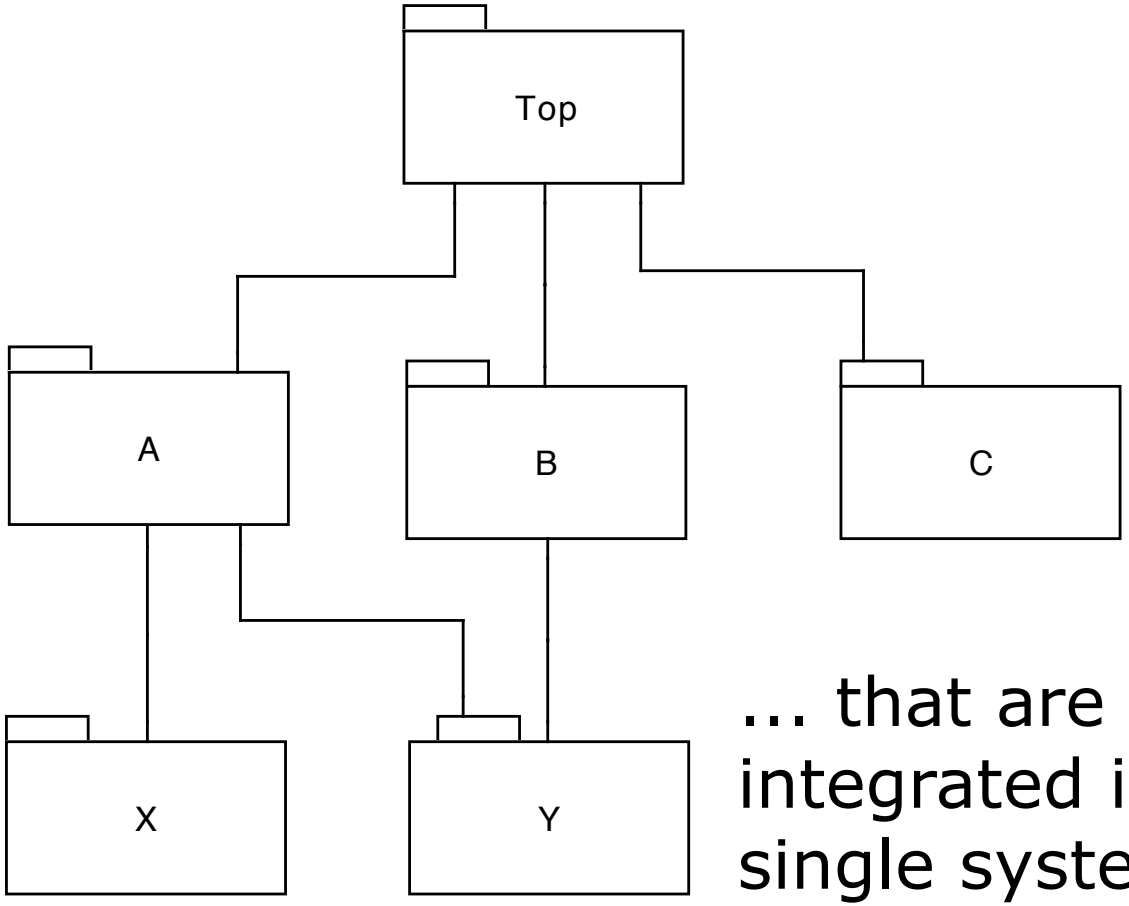


Bottom Up .....



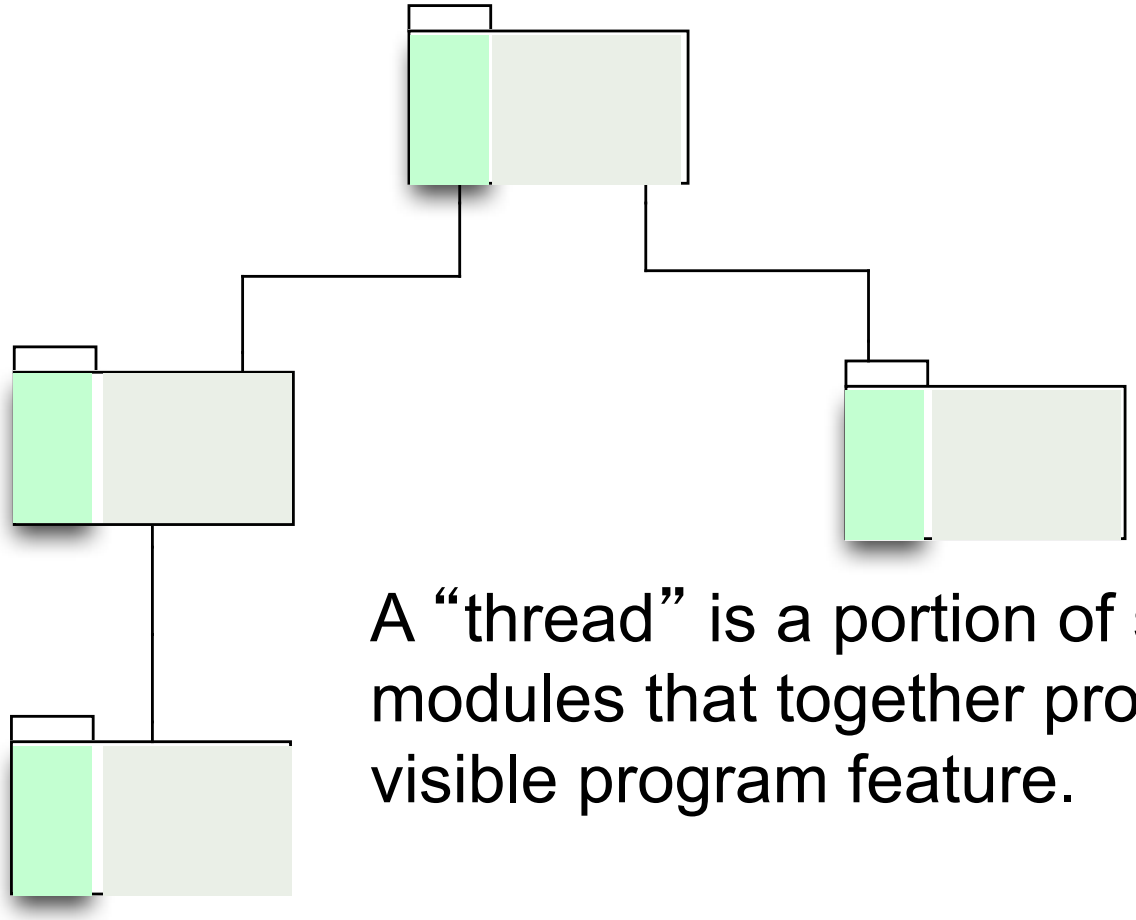
... so we may have several working subsystems ...

Bottom Up (complete)



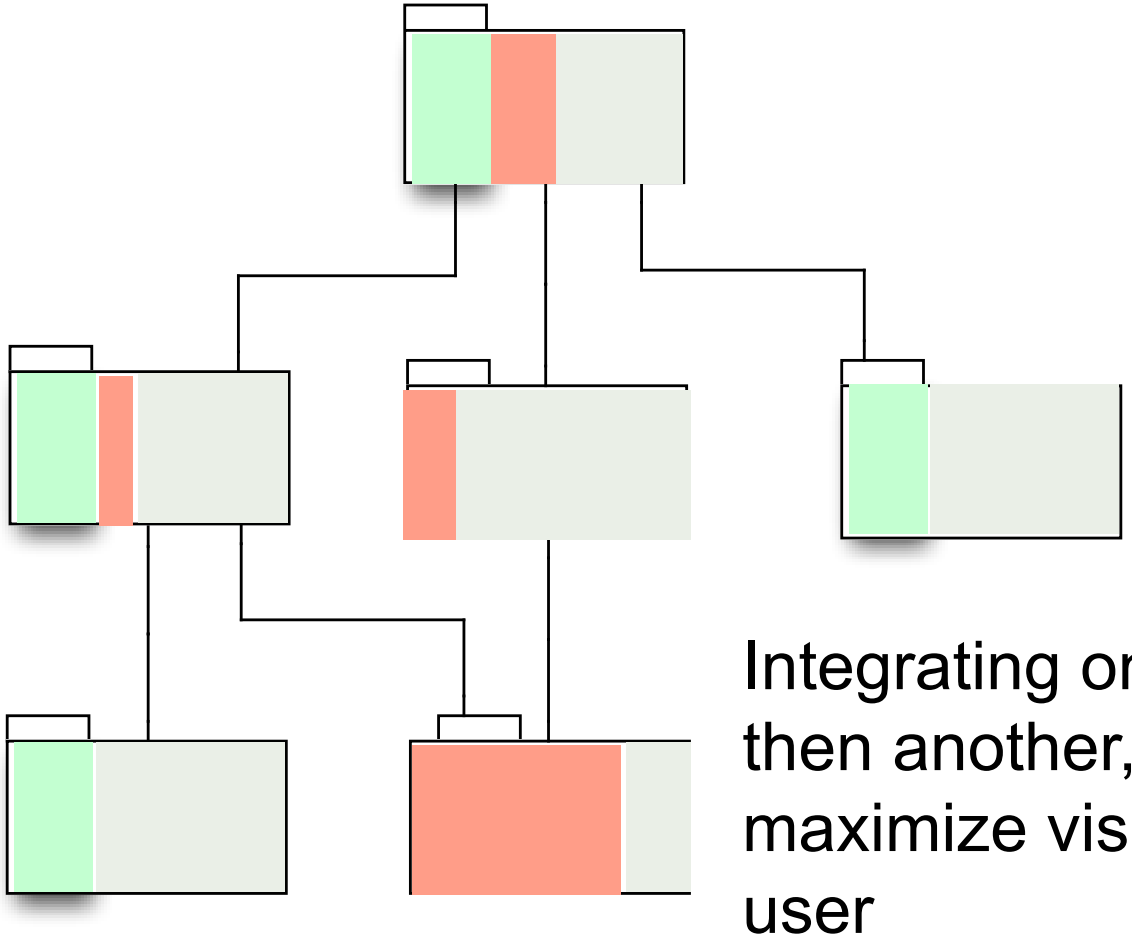
... that are eventually integrated into a single system.

Thread ...



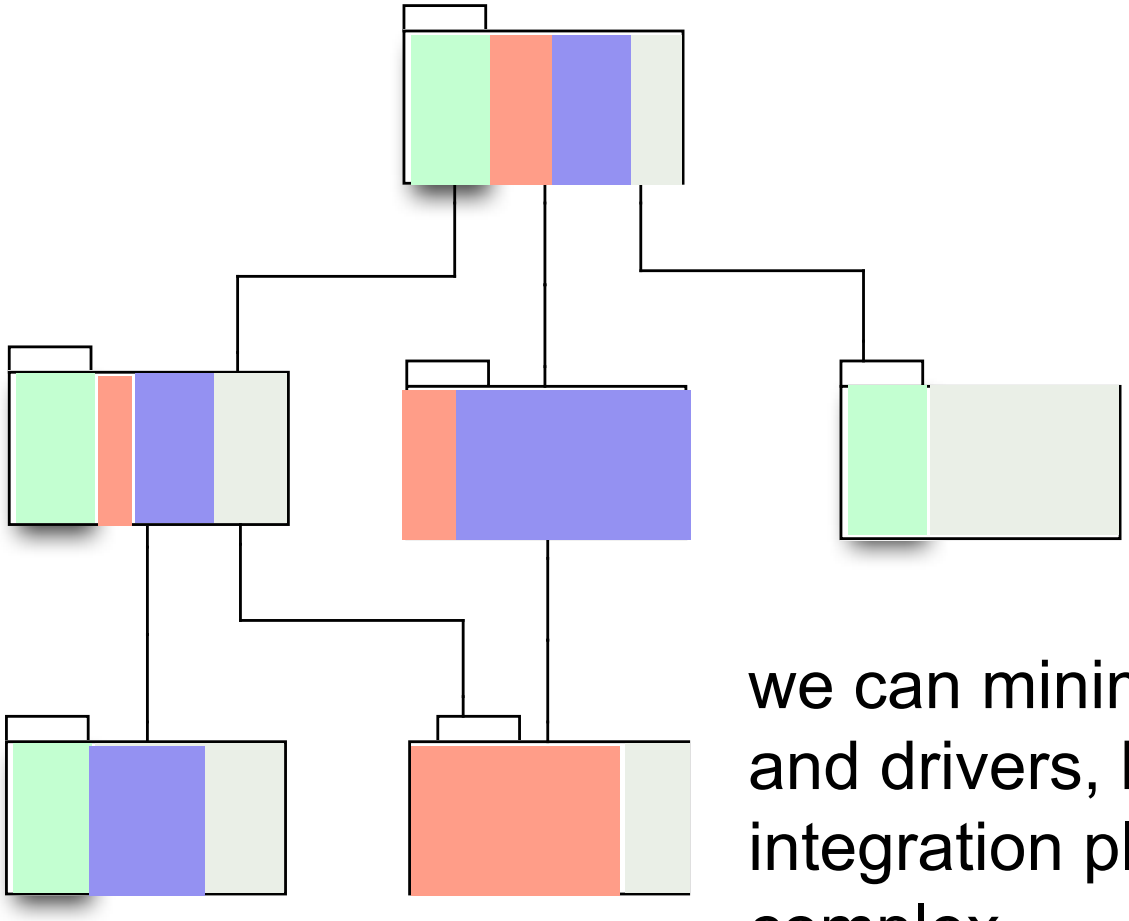
A “thread” is a portion of several modules that together provide a user-visible program feature.

Thread ...



Integrating one thread, then another, etc., we maximize visibility for the user

Thread ...



we can minimize stubs and drivers, but the integration plan may be complex

Critical Modules

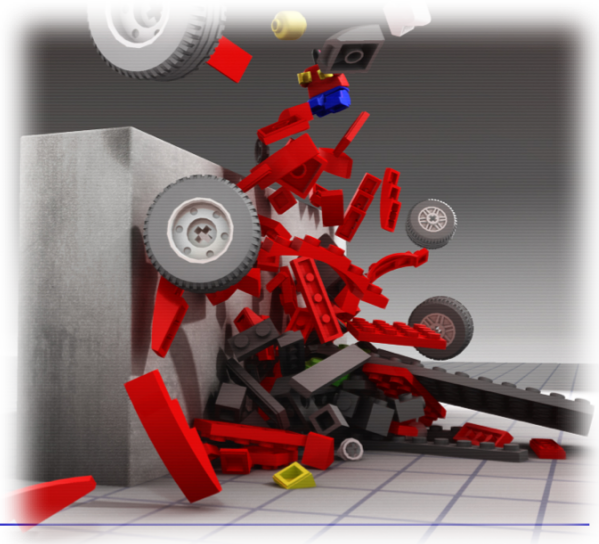
- Strategy: Start with riskiest modules
  - Risk assessment is necessary first step
  - May include technical risks (is X feasible?), process risks (is schedule for X realistic?), other risks
- May resemble thread process in tactics for flexible build order
  - E.g., constructing parts of one module to test functionality in another
- Key point is risk-oriented process
  - Integration & testing as a risk-reduction activity, designed to deliver any bad news as early as possible

Choosing a Strategy

- Structural strategies (bottom up and top down) are simpler
- But thread and critical modules testing provide better process visibility, especially in complex systems
- Possible to combine
  - Top-down and bottom-up are reasonable for relatively small components and subsystems
  - Combinations of thread and critical modules integration testing are often preferred for larger subsystems

System Testing

- Conducted on a complete integrated system
- Independent teams (black box)
- Functional and non-functional requirements
- Testing environment should be as close as possible to production environment
- System testing can be functional or... see next slides



Types of System Testing: Performance Testing

- Purpose
  - Identify bottlenecks affecting response time, utilization, throughput
  - Benchmarking, i.e., establish a performance baseline and possibly compare it with different versions of the same product or a different competitive product
- Prerequisites
  - Expected workload
  - Acceptable performance
- Identifying
  - Inefficient algorithms
  - Query optimization possibilities
  - Hardware/Network issues

Types of System Testing: Load Testing

- Purpose
  - Expose bugs such as memory leaks, mismanagement of memory, buffer overflows
  - Identify upper limits of components
- How
  - Increase the load until threshold
  - Load the system with the maximum load it can operate for a long period

Types of System Testing: Stress Testing

- Purpose
  - Make sure that the system recovers gracefully after failure
- How
  - Trying to break the system under test by overwhelming its resources or by taking resources away from it
- Examples
  - Double the baseline number for concurrent users/HTTP connections
  - Randomly shut down and restart ports on the network switches/routers that connects servers