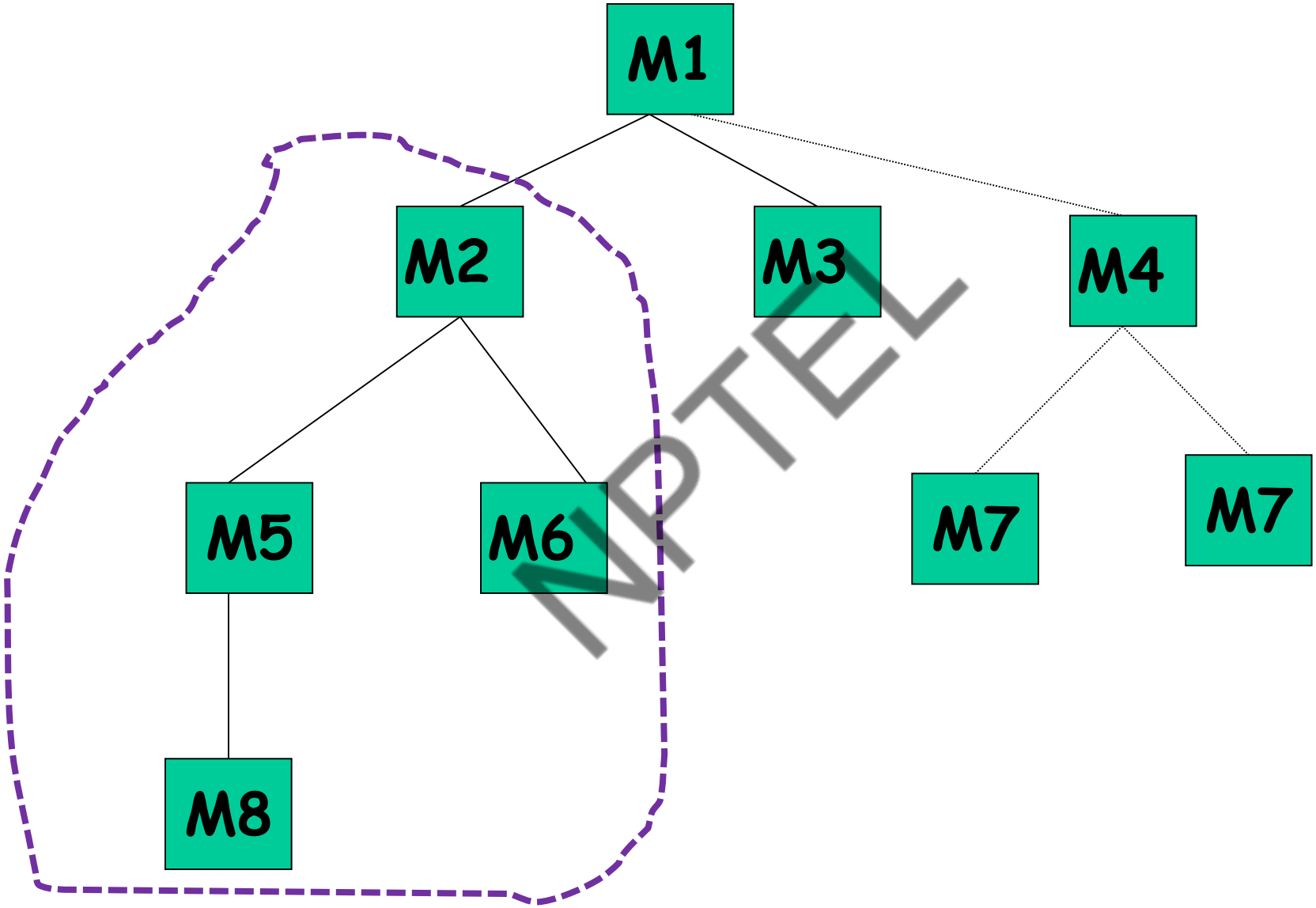
Integration Testing

Integration Testing Approaches

- Develop the integration plan by examining the structure chart :
  - big bang approach
  - top-down approach
  - bottom-up approach
  - mixed approach

Example Structured Design



Big Bang Integration Testing

- Big bang approach is the simplest integration testing approach:
  - All the modules are simply put together and tested.
  - This technique is used only for very small systems.

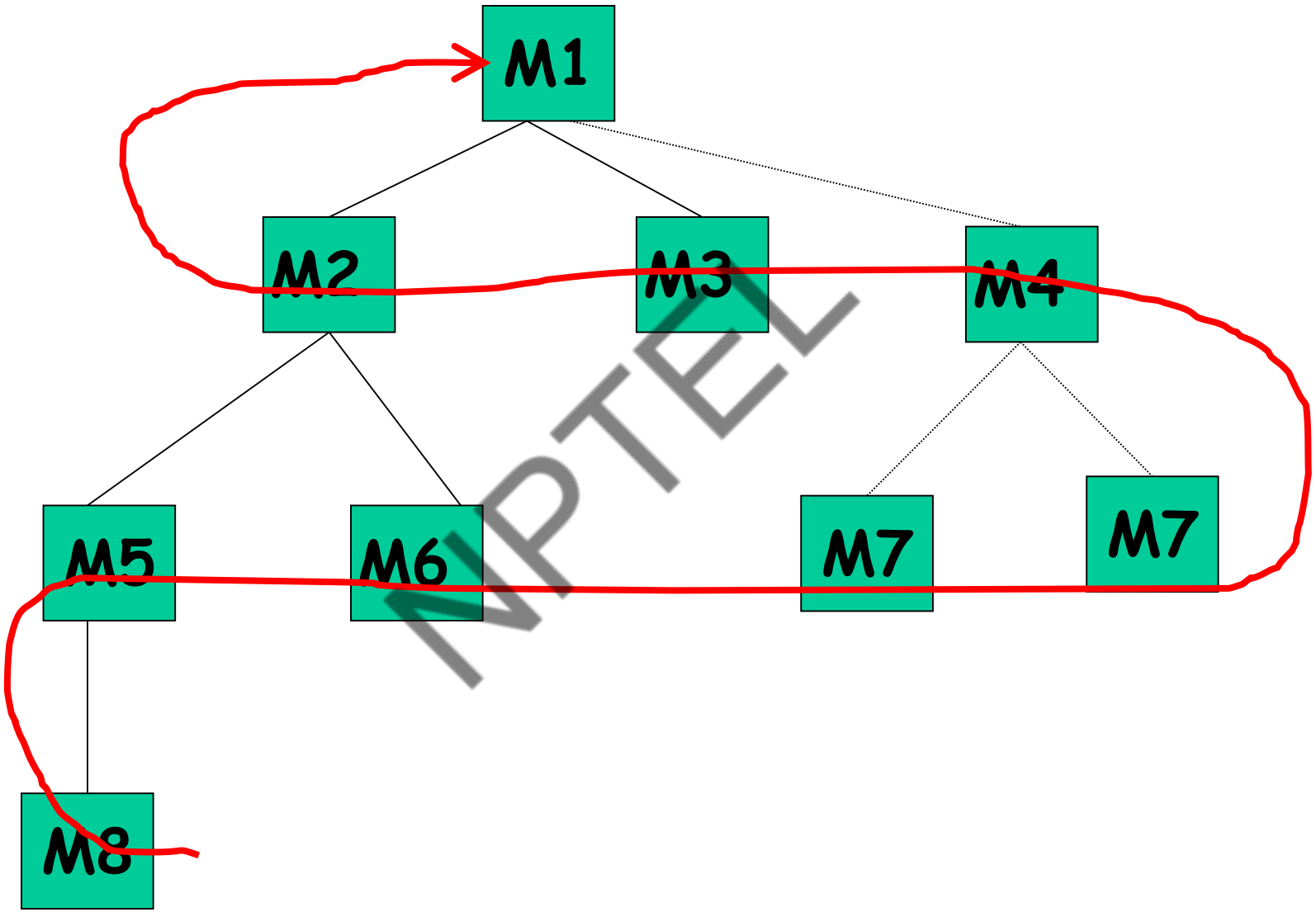
Big Bang Integration Testing

- Main problems with this approach:
  - If an error is found:
    - It is very difficult to localize the error
    - The error may potentially belong to any of the modules being integrated.
  - Debugging becomes very expensive.

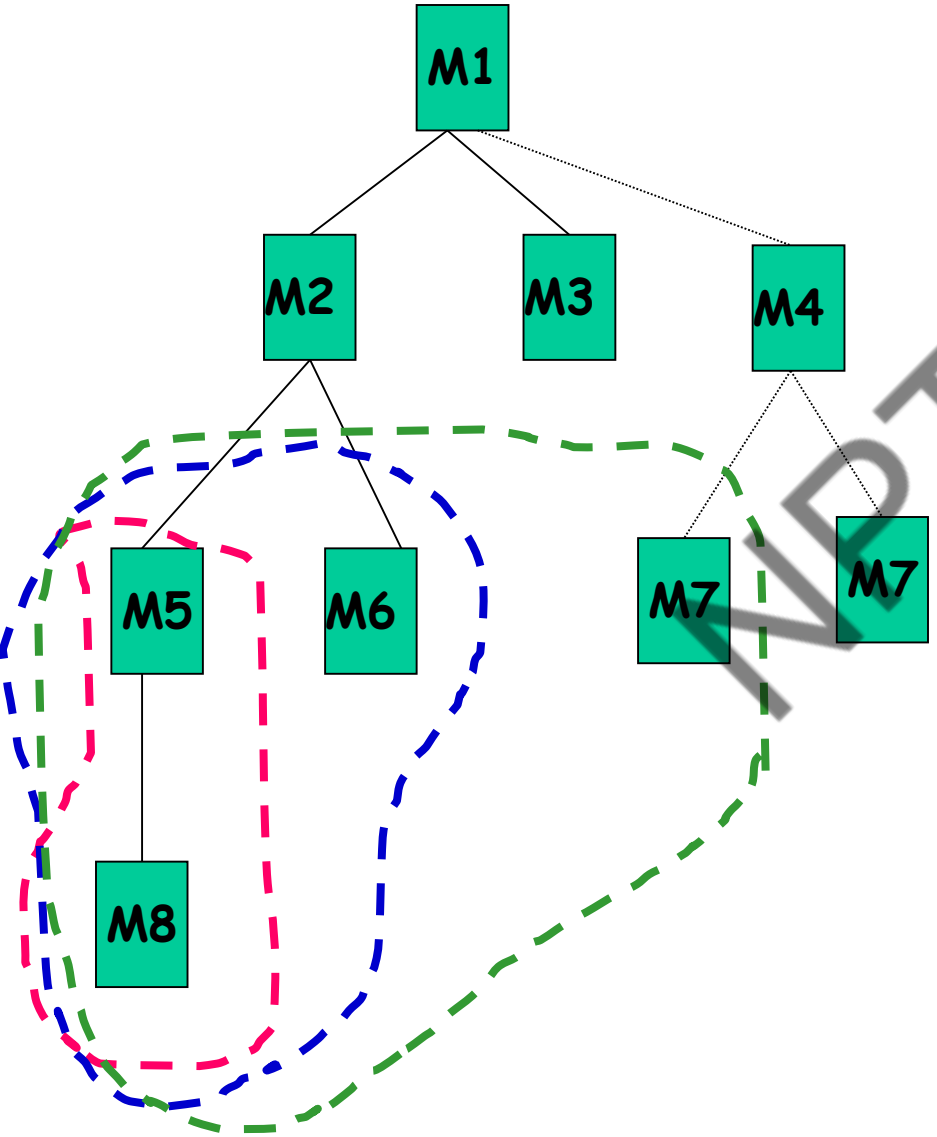
Bottom-up Integration Testing

- Integrate and test the bottom level modules first.
- Disadvantages of bottom-up testing:
  - Drivers have to be written.
  - Test engineers cannot observe system level functions from a partly integrated system.

Bottom-up testing



Example Bottom-up Testing



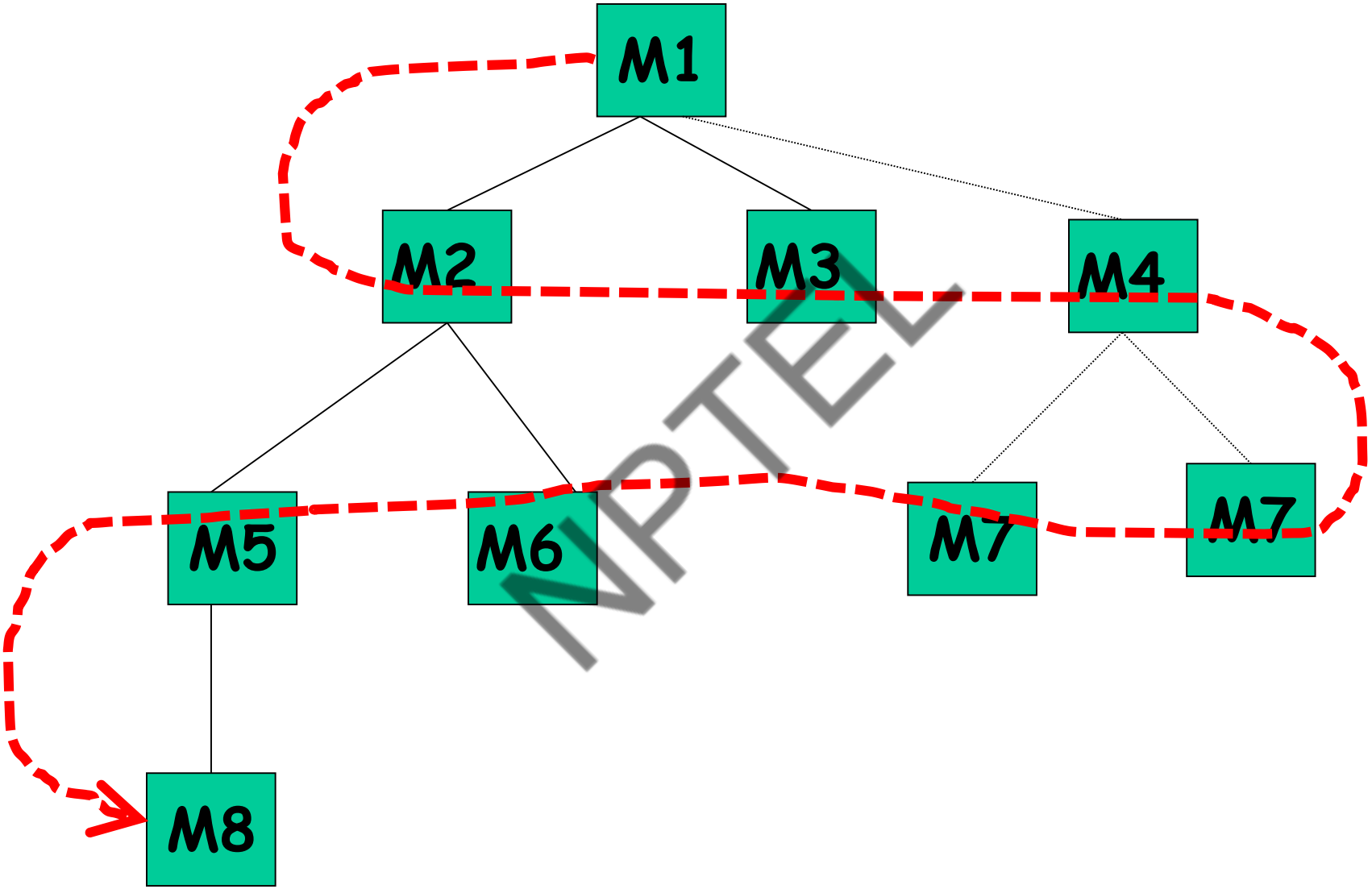
In Bottom Up testing :

- M5-M8 is tested with drivers for M5
- M5-M6-M8 is tested with drivers for M5-M6
- …

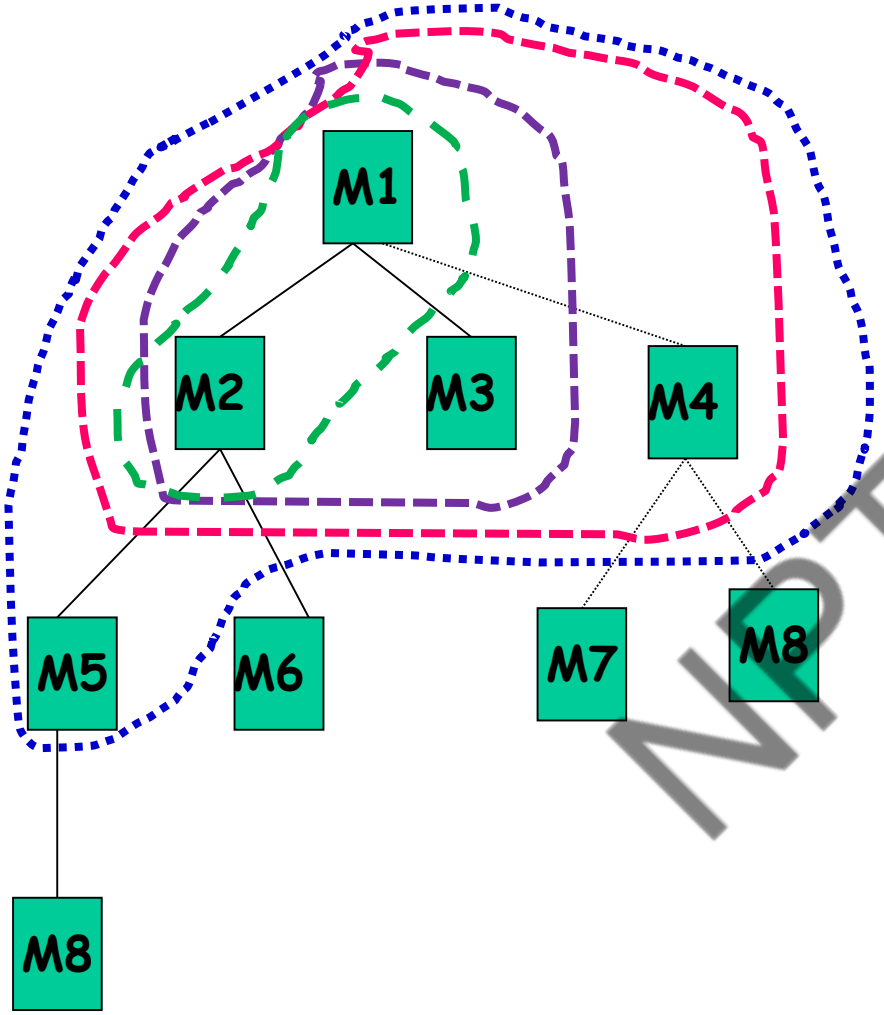
Top-down Integration Testing

- Top-down integration testing starts with the top module:
  - and one or two subordinate modules
- After the top-level 'skeleton' has been tested:
  - Immediate subordinate modules of the 'skeleton' are combined with it and tested.

Top-down testing



Top-down Integration Testing



- In Top Down test:
  - M1-M2 tested with stubs for M3, M4, M5 and M6
  - M1-M2- M3 tested with stubs for M4 M5, and M6
  - Then M1-M2-M3-M4 tested with stubs for M5, M6, M7 and M8

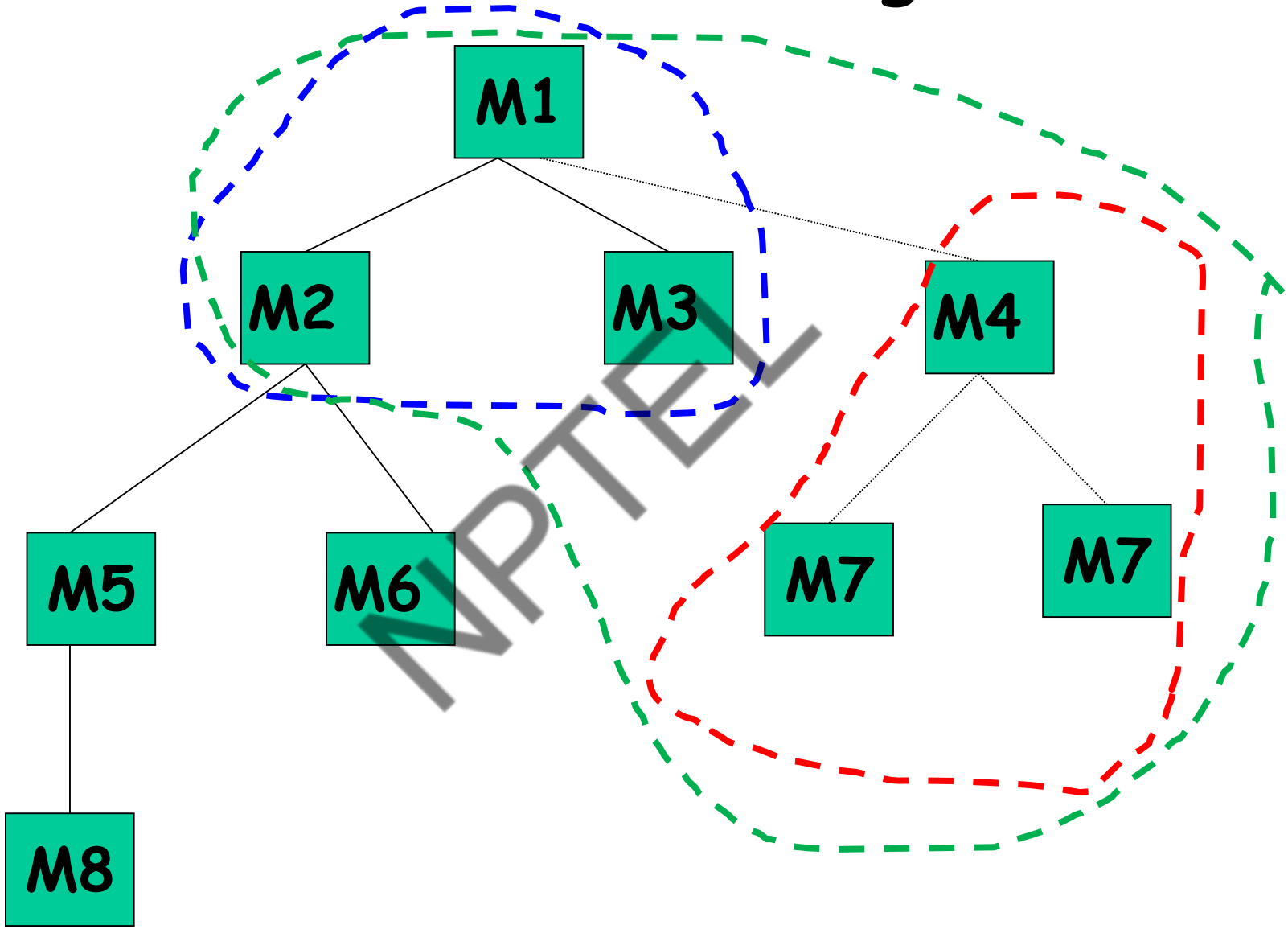
Top-Down Integration

- **Advantages of top down integration testing:**
  - Test cases designed to test the integration of some module are reused after integrating other modules at lower level.
  - Advantageous if major flaws occur toward the top of the program.
- **Disadvantages of top down integration testing:**
  - It may not be possible to observe meaningful system functions because of an absence of lower level modules which handle I/O.
  - Stub design become increasingly difficult when stubs lie far away from the level of the module.

Mixed Integration Testing

- Mixed (or sandwiched) integration testing:
  - Uses both top-down and bottom-up testing approaches.
  - Requires less stubs and drivers

Sandwich testing



Integration Testing

- In top-down approach:
  - Integration testing waits till all top-level modules are coded and unit tested.
- In bottom-up approach:
  - Testing can start only after bottom level modules are ready.

System Testing

- Objective:
  - **Validate a fully developed software against its requirements.**

System Testing

- There are three main types of system testing:
  - **Alpha Testing**
  - **Beta Testing**
  - **Acceptance Testing**

Alpha Testing

- System testing carried out by the test team within the developing organization.
  - Test cases are designed based on the SRS document

Beta Testing

- System testing performed by a select group of friendly customers.

Acceptance Testing

- System testing performed by the customer himself:
  - **To determine whether the system should be accepted or rejected.**

System Testing

Types of System Testing

- Two types:
  - **Functionality:** Black-box test cases are designed to test the system functionality against the requirements.
  - **Performance:** Tests are designed against the non-functional requirements documented in the SRS document.

Performance Tests

- Stress tests
- Volume tests
- Configuration tests
- Compatibility tests
- Security tests
- Load test
- Recovery tests
- Maintenance tests
- Documentation tests
- Usability tests
- Environmental tests

Stress Testing

- Stress testing (also called endurance testing):
  - Impose abnormal input to stress the capabilities of the software.
  - **Input data volume, input data rate, processing time, utilization of memory, etc. are tested beyond the designed capacity.**

Stress Testing

- Stress testing usually involves an element of time or size,
  - Such as the number of records transferred per unit time,
  - The maximum number of users active at any time, input data size, etc.
- Therefore stress testing may not be applicable to many types of systems.

Stress Testing Examples

- If an operating system is supposed to support 15 multiprogrammed jobs,
  - The system is stressed by attempting to run 15 or more jobs simultaneously.
- A real-time system might be tested
  - To determine the effect of simultaneous arrival of several high-priority interrupts.

Load testing

- Load testing:
  - Determines whether the performance of the system under different loads acceptable?
  - **Example:** For a web-based application, what is the performance of the system under some specified hits?
- Tool:
  - JMeter:
    http://jakarta.apache.org/jmeter/

Volume Testing

- Tests whether handling large amounts of data in the system is satisfactory:
  - Whether data structures (e.g. queues, stacks, arrays, etc.) are large enough to handle all possible situations.
  - **Fields, records, and files are stressed to check if their size can accommodate all possible specified data volumes.**

Configuration Testing

- Sometimes systems are built in various configurations for different users
  - for instance, a minimal system may serve a single user, other configurations for additional users.
- Test system behavior:
  - in various hardware and software configurations specified in requirements

Compatibility Testing

- These tests are needed when the system interfaces with other systems:
  - Check whether the interface functions as required.
- **Example:** For a web-based application, check whether the application works satisfactorily with various web browsers.

Compatibility Testing: Another Example

- If a system is to communicate with a large database system to retrieve information:
  - A compatibility test examines speed and accuracy of retrieval.

Recovery Testing

- These tests check response to:
  - The loss of data, power, devices, or services
  - Subject system to loss of resources
    - Check if the system recovers properly.

Maintenance Testing

- Diagnostic tools and procedures help find source of problems.
  - It may be required to supply
    - Default configurations
    - Diagnostic programs
    - Traces of transactions,
    - Schematic diagrams, etc.
- Verify that:
  - all required artefacts for maintenance exist
  - they function properly

Documentation tests

- Check whether required documents exist and are consistent:
  - user guides,
  - maintenance guides,
  - technical documents
- Sometimes requirements specify:
  - Format and audience of specific documents
  - Documents are evaluated for compliance

Usability tests

- All aspects of user interfaces are tested:
  - Display screens
  - messages
  - report formats
  - navigation and selection problems

Environmental test

- These tests check the system's ability to perform at the installation site.
- Requirements might include tolerance for
  - heat
  - humidity
  - chemical presence
  - portability
  - electrical or magnetic fields
  - disruption of power, etc.

Test Summary Report

- Generated towards the end of testing phase.
- Covers each subsystem:
  - A summary of tests which have been applied to the subsystem.

Test Summary Report

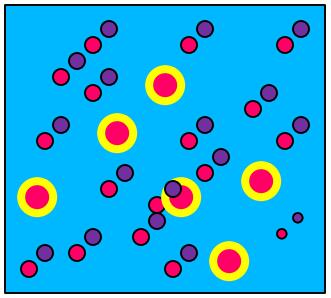
- Specifies:
  - how many tests have been applied to a subsystem,
  - how many tests have been successful,
  - how many have been unsuccessful, and the degree to which they have been unsuccessful,
    - e.g. whether a test was an outright failure
    - or whether some expected results of the test were actually observed.

Regression Testing

Latent Errors: How Many Errors are Still Remaining?

- Make a few arbitrary changes to the program:
  - **Artificial errors are seeded into the program.**
  - **Check how many of the seeded errors are detected during testing.**

Error Seeding



- Let:
  - N be the total number of errors in the system
  - n of these errors be found by testing.
  - S be the total number of seeded errors,
  - s of the seeded errors be found during testing.

Error Seeding

- n/N = s/S
- N = S n/s
- remaining defects:

$$N - n = n\ ((S - s)/ s)$$

Quiz 1

- 100 errors were introduced.
- 90 of these errors were found during testing
- 50 other errors were also found.
- Find error estimate for the code.

Quiz 1: Solution

- 100 errors were introduced.
- 90 of these errors were found during testing
- 50 other errors were also found.
- Remaining errors=

  50 (100-90)/90 = 6

Error Seeding: An Issue

- The kinds of seeded errors should match closely with existing errors:
  - However, it is difficult to predict the types of errors that exist.
- **Working solution:**
  - **Estimate by analyzing historical data from similar projects.**

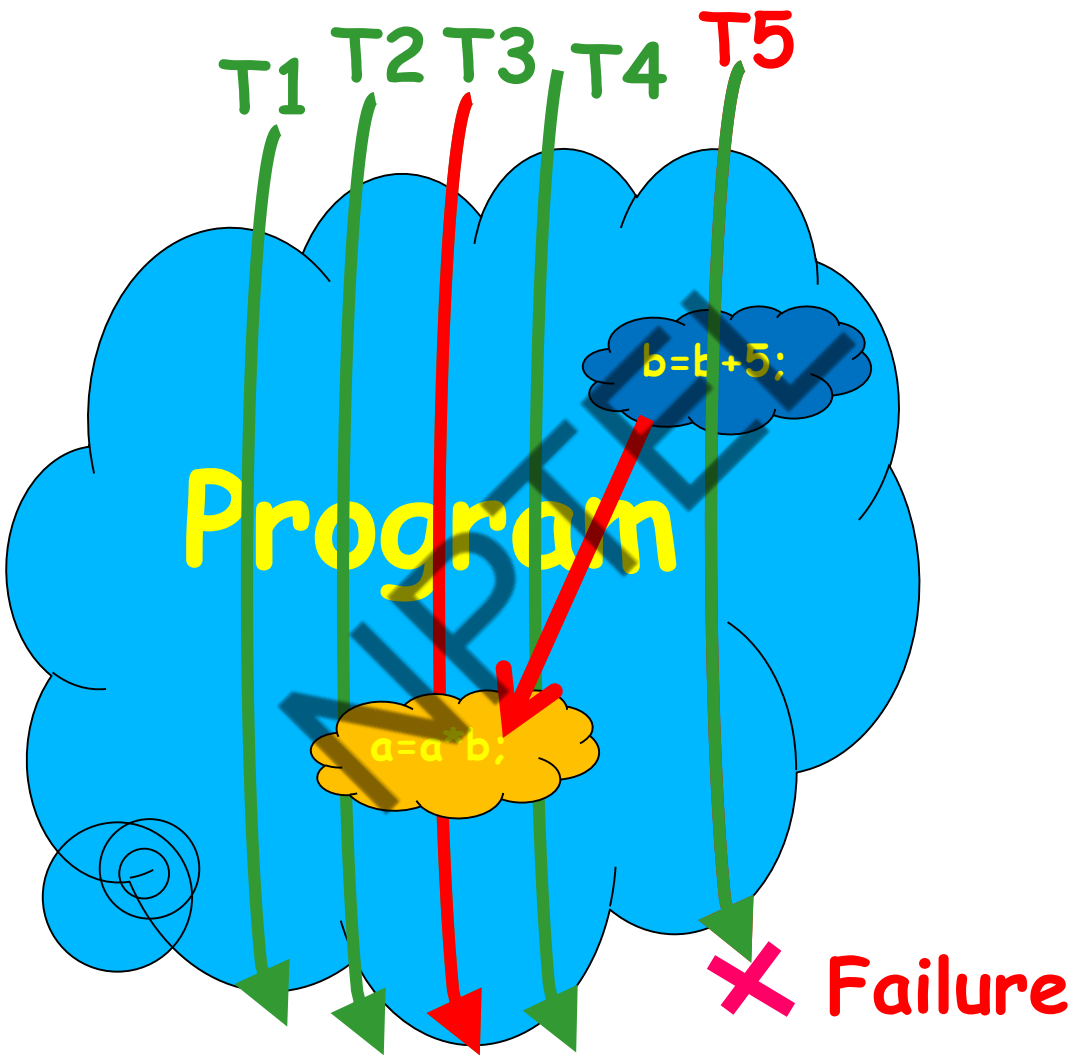
Quiz 2

- Before system testing 100 errors were seeded by the manager.
- During system testing 60 of these were detected.
- 150 other errors were also detected
- **How many unknown errors are expected to remain after system testing?**

What is regression testing?

Regression testing is testing done to check that a system update does not cause new errors or re-introduce errors that have been corrected earlier.

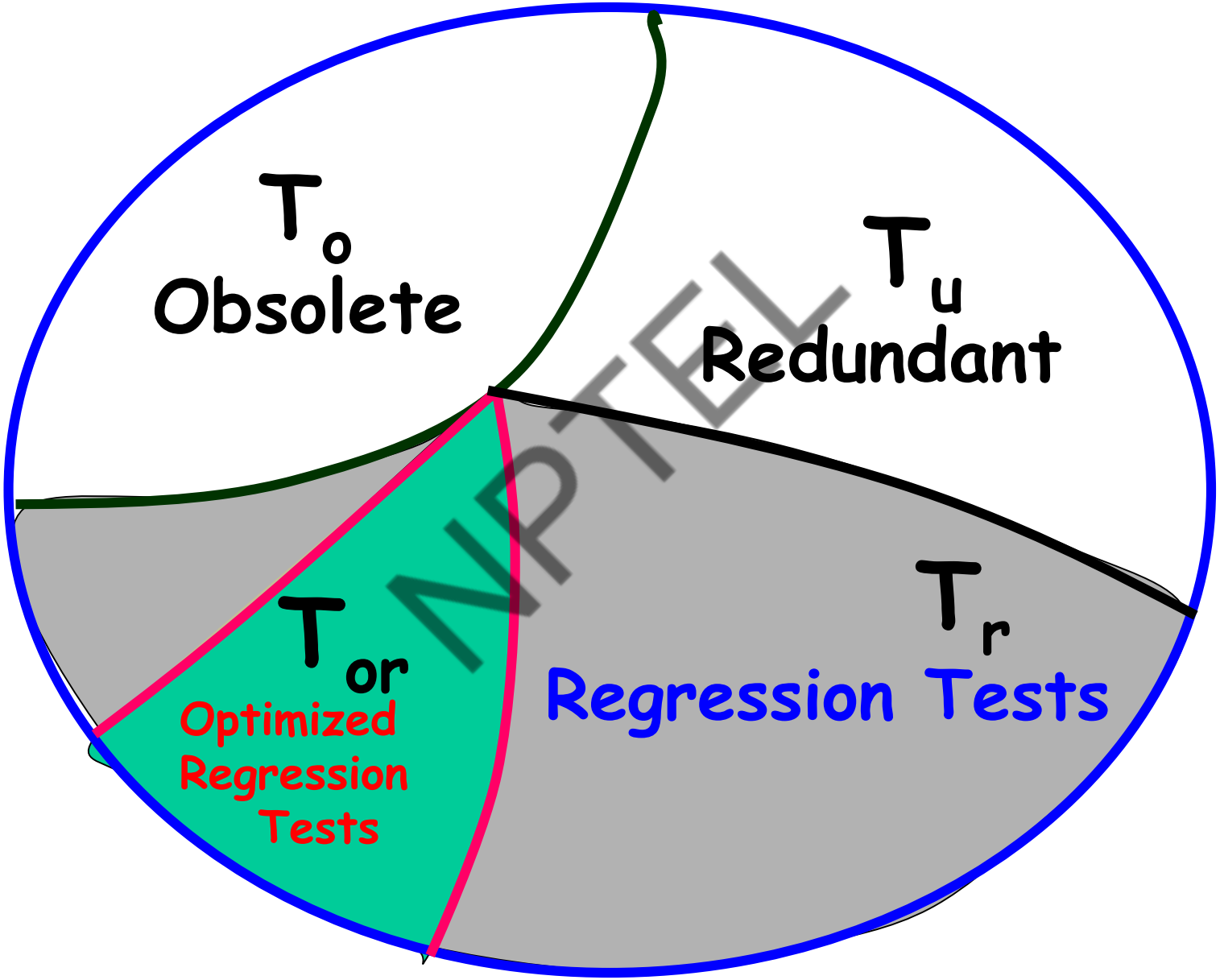
Why regression testing?



Need for Regression Testing

- Any system during use undergoes frequent code changes.
  - Corrective, Adaptive, and Perfective changes.
- Regression testing needed after every change:
  - Ensures unchanged features continue to work fine.

Partitions of an Existing Test Suite



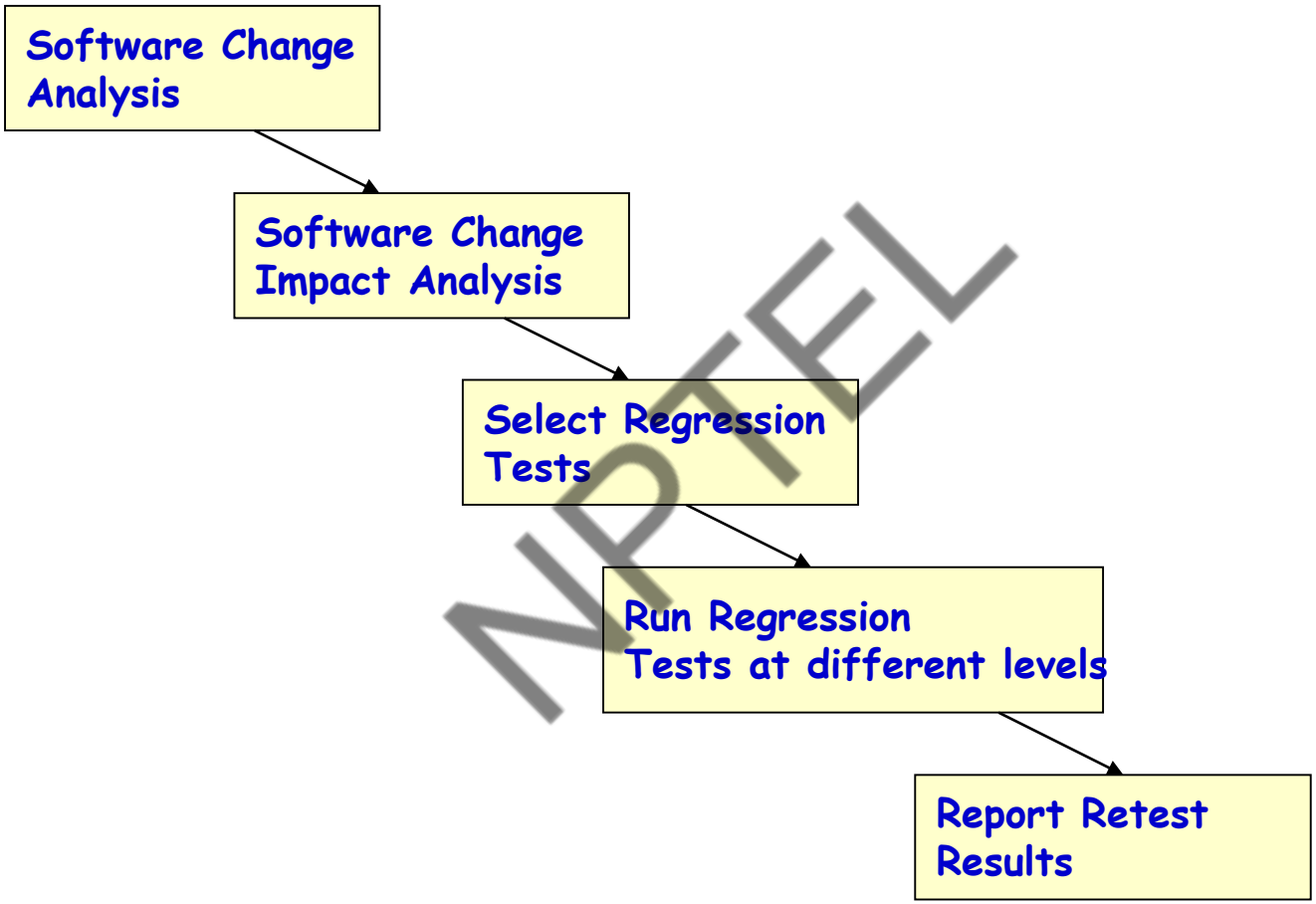
Automated Regression Testing

- Test cases that have already run once:
  - May have to be run again and again after each change
  - Test cases may be executed hundreds of time
  - Automation very important…
- Fortunately, capture and replay type tools appear almost a perfect fit:
  - However, test cases may fail for reasons such as date or time change
  - Also test cases may need to be maintained after code change

Major Regression Testing Tasks

- Test revalidation (RTV):
  - Check which tests remain valid
- Test selection (RTS):
  - Identify tests that execute modified portions.
- Test minimization (RTM):
  - Remove redundant tests.
- Test prioritization (RTP):
  - Prioritize tests based on certain criteria.

Software Regression Process



Testing Object-Oriented Programs

Quiz

- Is regression testing a unit, integration, or system testing technique?
- Answer: It is used in all of these testing. Actually it is a different dimension of testing.

Introduction

- More than 50% of development effort is being spent on testing.
- Quality and effective reuse of software depend to a large extent:
  - on thorough testing.
- It was expected during initial years that OO would help substantially reduce testing effort:
  - But, as we find it out today --- it only complicates testing.

Challenges in OO Testing

- What is an appropriate unit for testing?
- Implications of OO features:
  - Encapsulation
  - Inheritance
  - Polymorphism & Dynamic Binding, etc.
- State-based testing
- Test coverage analysis
- Integration strategies
- Test process strategy

What is a Suitable Unit for Testing?

- What is the fundamental unit of testing for conventional programs?
  - A function.
- However, as far as OO programs are concerned:
  - Methods are not the basic unit of testing.
- **Weyukar's Anticomposition axiom**:
  - Any amount of testing of individual methods can not ensure that a class has been satisfactorily tested.

Suitable Unit for Testing OO Programs

- **Class level:**
  - Testing interactions between attributes and methods must be addressed.
  - State of the object must be considered.

Weyukar's Anticomposition Axion (IEEE TSE Dec. 1986)

- Adequate testing of each individual program components does not necessarily suffice to adequate test the entire program.
- **Consider P and Q as components:**
  - P has the opportunity to modify the context seen by Q in a more complex way than could be done by stubs during testing of components in isolation.

- **What is the interpretations for OO programs?**

Encapsulation

- Encapsulation is not a source of errors:
  - However, an obstacle to testing.
  - It prevents accessing attribute values by a debugger.
- While testing:
  - Precise information about current state is necessary.

Solving Encapsulation-Related Problems

Several solutions are possible:

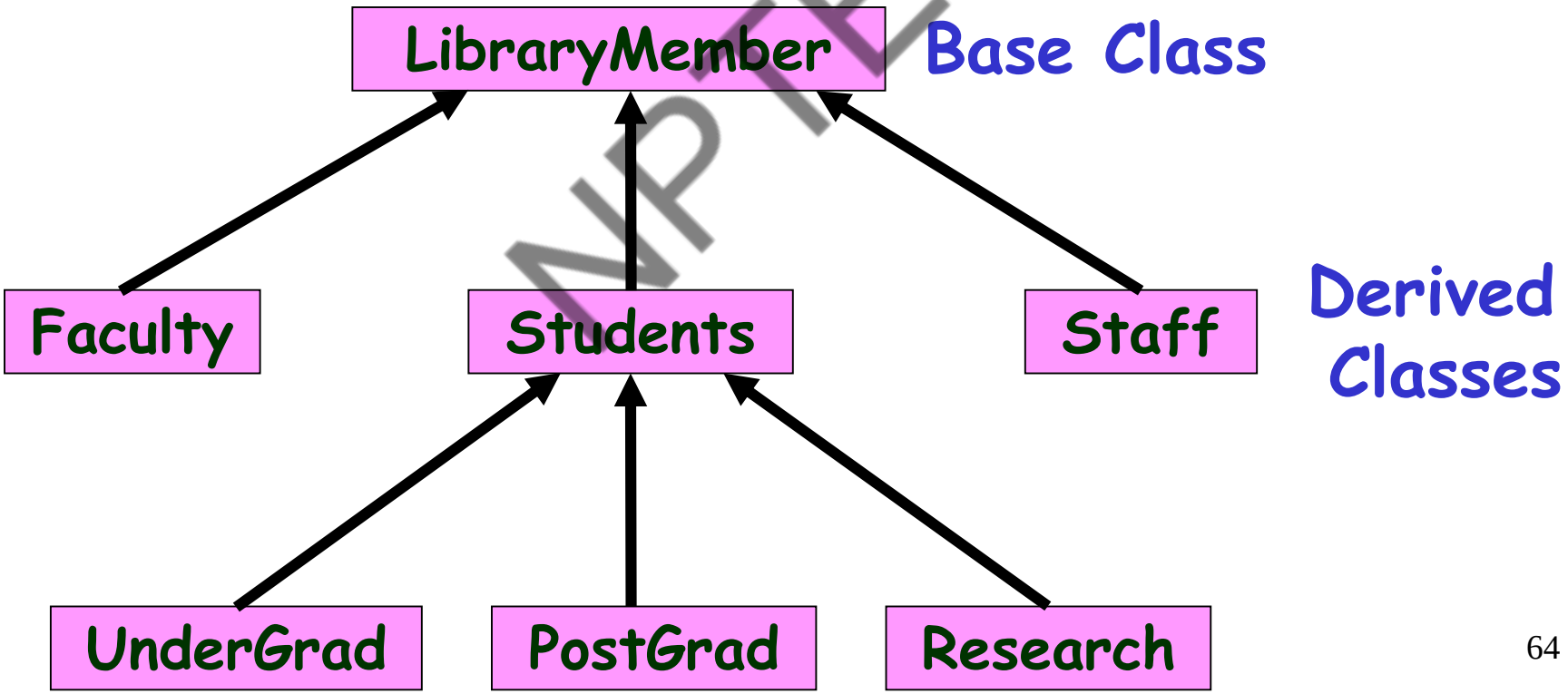
- Built-in or inherited state reporting methods.
- Low level probes to manually inspect object attributes.
- Proof-of-correctness technique (formal).

Solving Encapsulation-Related Problems

- Most feasible way:
  - State reporting methods.
- Reliable verification of state reporting methods is a problem.

Inheritance

- Should inherited methods be retested?
  - Retesting of inherited methods is the rule, rather than an exception.



Should Inherited Methods be Retested?

- Retesting required:
  - Because a new context of usage results when a subclass is derived. (Anticomposition axiom)

- Correct behavior at an upper level:
  - Does not guarantee correct behavior at a lower level.

Example

```
Class A{
Protected int x=200; //invariant x>100
Void m(){//correctness depends on
invariant}
}
Class B extends A{
    void m1(){x=1; …} …}
```

- Execution of m1() causes a bug in m()
- Breaks the invariant, m is incorrect in the context of B, even though it is correct in A:
  - Therefor m should be retested in B

Another Example

```
Class A{
Void m(){ … m2(); … }
Void m2() {…} }
Class B extends A{
    void m2(){…} …}
```

- M2 has been overridden in B, can affect other methods inherited from A such as m()
  - m() would now call B. m2.
  - So, we cannot be sure that m is correct anymore, we need to retest it with B instance

Inheritance --- Overriding

- In case of method overriding:
  - Need to retest the classes in the context of overriding.
  - An overridden method must be retested even when only minor syntactic changes are made.

Which Methods to Test Within A Class?

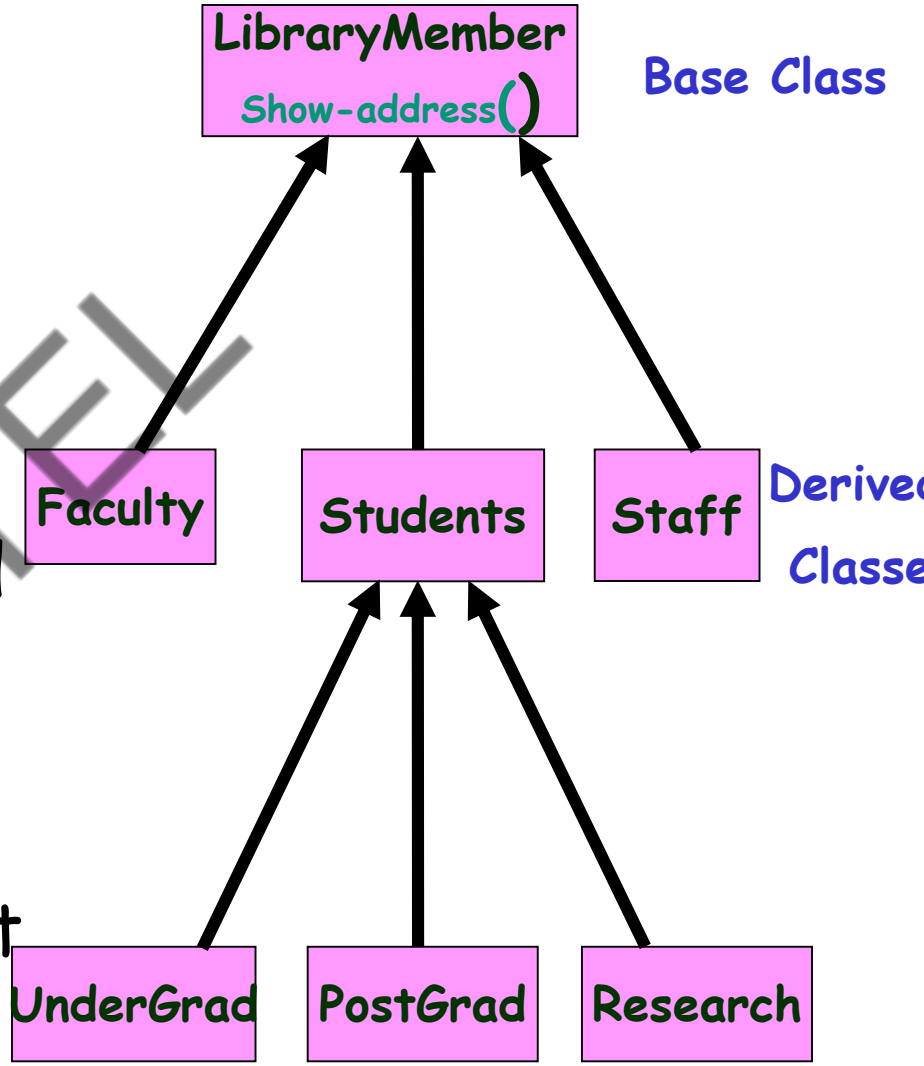
- New methods:
  - Defined in the class under test not inherited or overloaded by methods in a super class:
  - Complete testing
- Inherited methods:
  - Defined in a superclass of the class under test:

    Retest only if the methods interacts with new or redefined method.
- Redefined methods:

- Defined in a superclass of but redefined class under test: complete Retest

Regression Testing Derived Class: Example

- Principle: inherited methods should be retested in the context of a subclass
- Example: if we change a method show-address() in a super class, we need to retest show-address() and other dependent methods inside all subclasses that inherit it.



Deep Inheritance Hierarchy

- A subclass at the bottom of a deep hierarchy:
  - May have only one or two lines of code.
  - But may inherit hundreds of features.
- This situation creates fault hazards:
  - Similar to unrestricted access to global data in procedural programs.
- **Inheritance weakens encapsulation.**

Deep Inheritance Hierarchy cont...

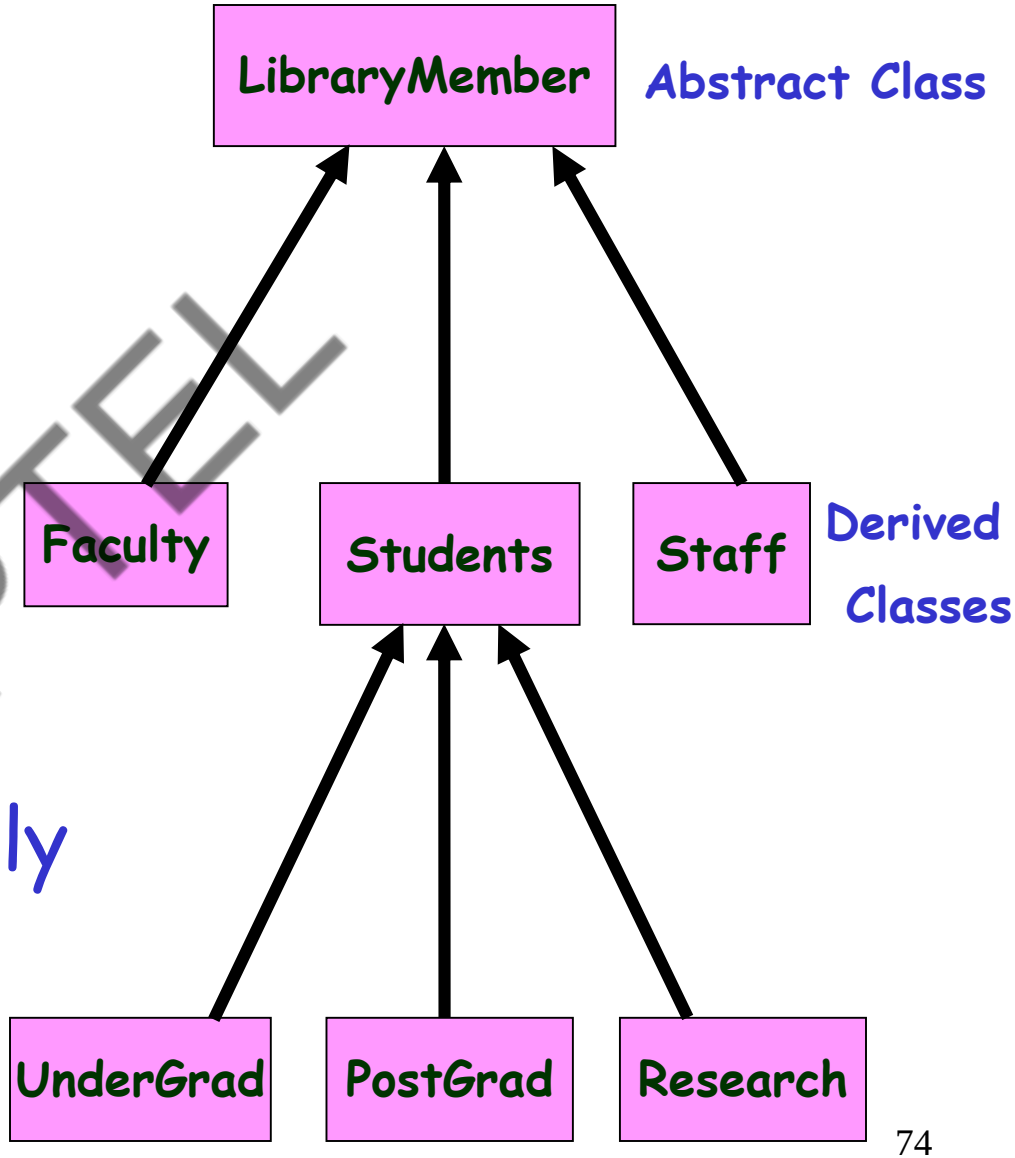
- A deep and wide inheritance hierarchy can defy comprehension:
  - Lead to bugs and reduce testability.
  - Incorrect initialization and forgotten methods can result.
  - Class flattening may increase understandibility.
- Multiple Inheritance:
  - Increases number of contexts to test.

Abstract and Generic Classes

- Unique to OO programming:
  - Provide important support for reuse.
- Must be extended and instantiated to be tested.
- May never be considered fully tested:
  - Since need retesting when new subclasses are created.

Testing an Abstract Class

- Not possible to directly test it.
  - Can only be indirectly tested through classes derived from it.
  - So can never be considered as fully tested.



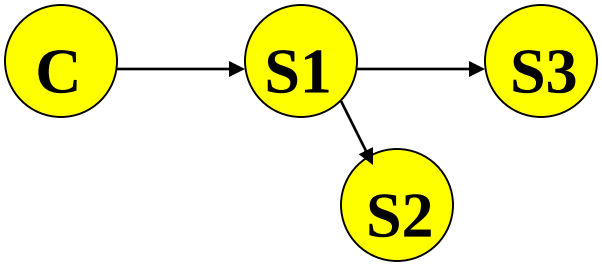
Polymorphism

- Each possible binding of a polymorphic component requires separate testing:
  - Often difficult to find all bindings that may occur.
  - Increases the chances of bugs .
  - An obstacle to reaching coverage goals.

Polymorphism

cont…

- Polymorphism complicates integration planning:
  - Many server classes may need to be integrated before a client class can be tested.

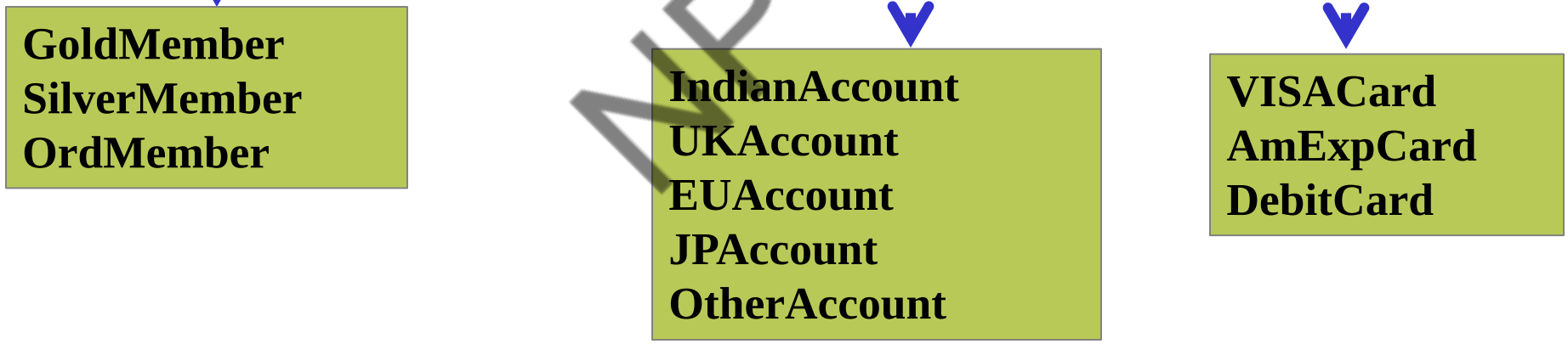


Dynamic Binding

- Dynamic binding implies:
  - The code that implements a given function is unknown until run time.
  - Static analysis cannot be used to identify the precise dependencies in a program.
- It becomes difficult to identify all possible bindings and test them.

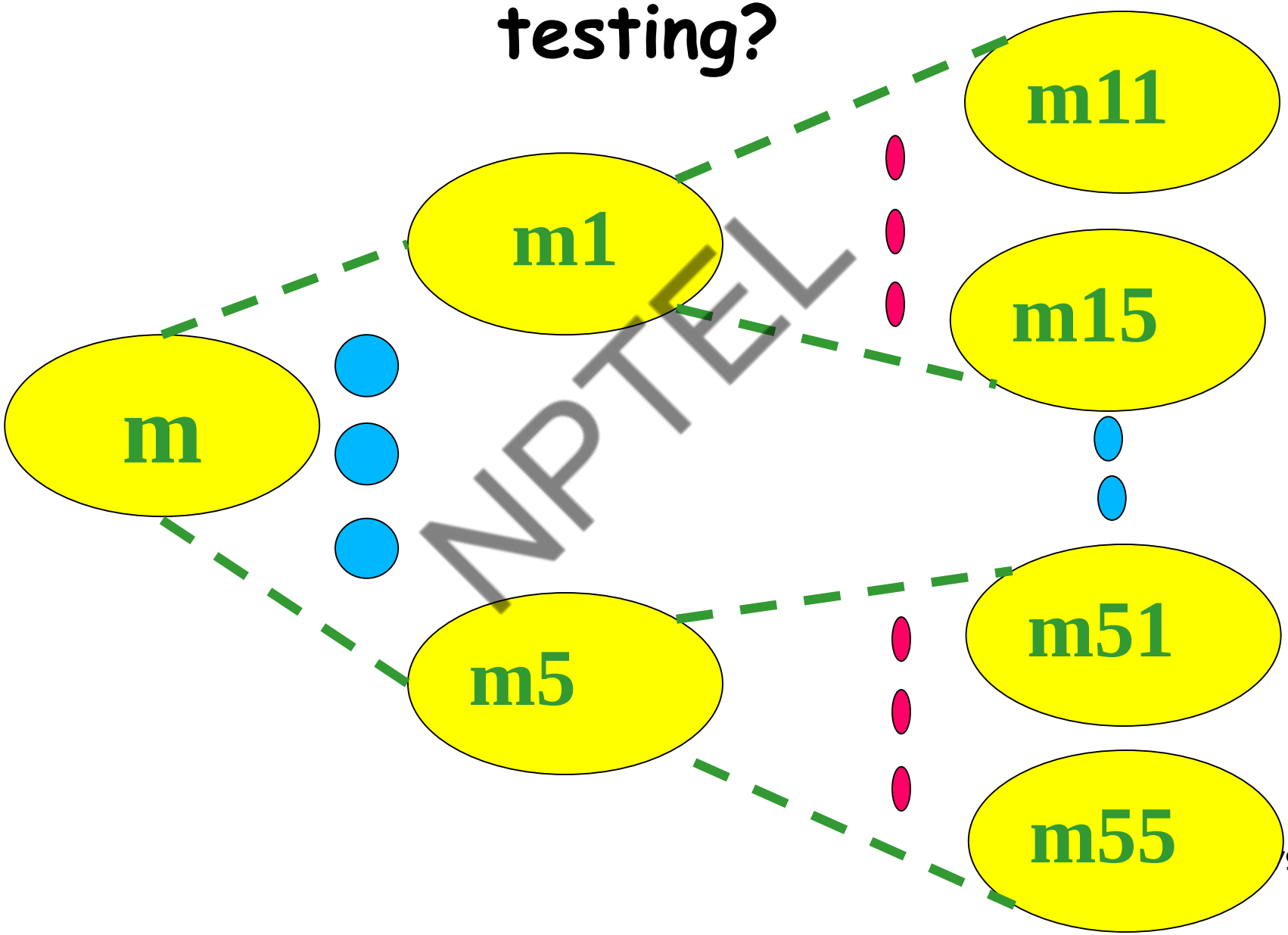
Dynamic Binding: the combinatorial explosion problem

```
Class member{
...
 abstract boolean validatePayment(Account a, int amt, Card c) ;
...
}
```



The combinatorial problem: $3 \times 5 \times 3 = 45$ possible combinations of dynamic bindings (just for this one method!)

How many test cases for pair wise testing?



State-Based Testing

- The concept of control flow of a conventional program :
  - Does not map readily to an OO program.
- In a state model:
  - We specify how the object's state would change under certain conditions.

State-Based Testing

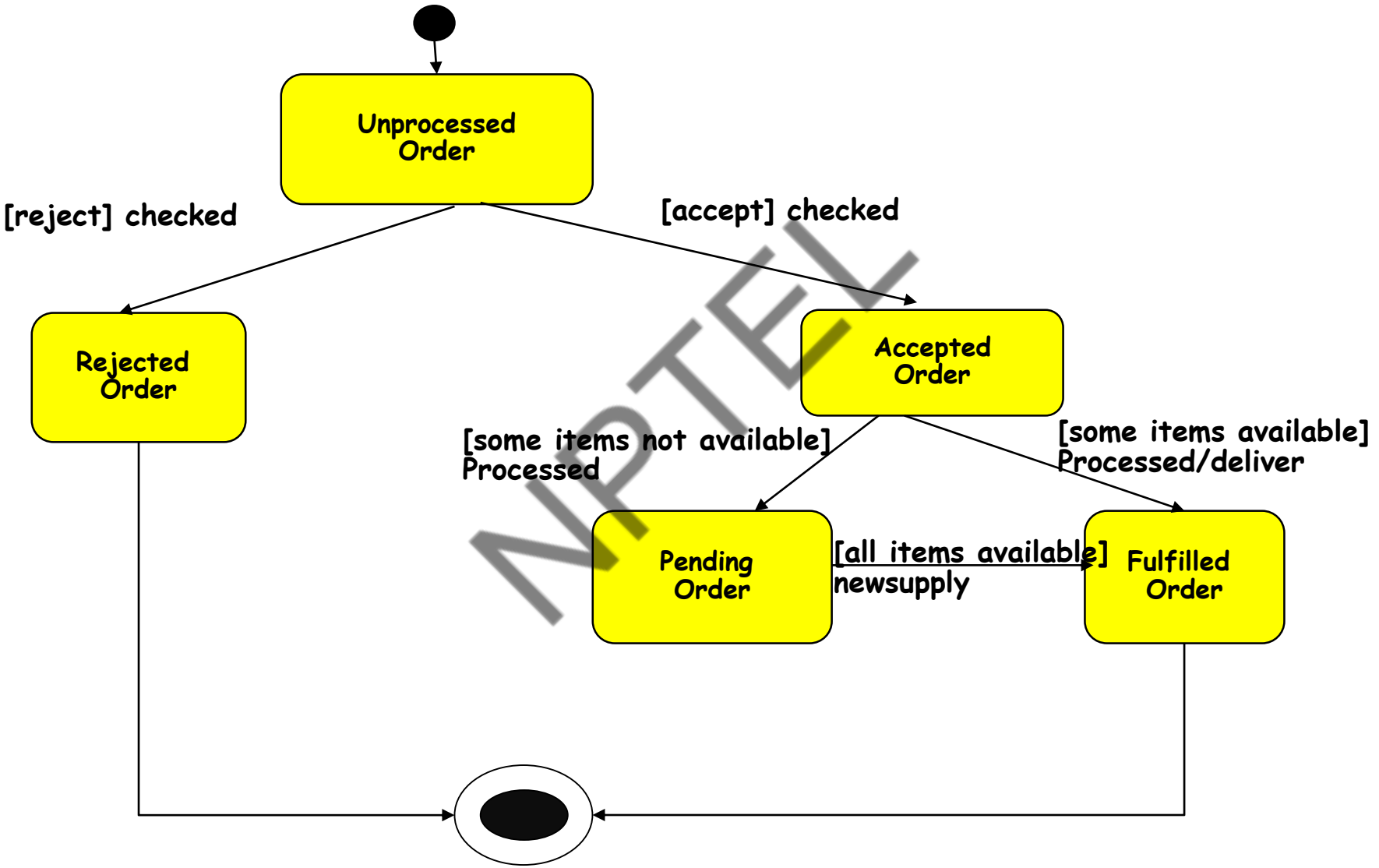
- Flow of control in OO programs:
  - Message passing from one object to another
- Causes the receiving object to perform some operation,
  - can lead to an alteration of its state.

State-Based Testing

cont...

- The state model defines the allowable transitions at each state.
- States can be constructed:
  - Using equivalence classes defined on the instance variables.
- Jacobson's OOSE advocates:
  - Design test cases to cover all state transitions.

An example of A State Model



Example: State chart diagram for an order object

State-Based Integration Testing

cont...

- Test cases can be derived from the state machine model of a class:
  - Methods result in state transitions.
  - Test cases are designed to exercise each transition at a state.
- However, the transitions are tied to user-selectable activation sequences:
  - Use Cases

Difficulty With State Based Testing

- The locus of state control is distributed over an entire OO application.
  - Cooperative control makes it difficult to achieve system state and transition coverage.
- A global state model becomes too complex for practical systems.
  - Rarely constructed by developers.
  - A global state model is needed to show how classes interact.

Test Coverage

- Test coverage analysis:
  - Helps determine the "thoroughness" of testing achieved.
- Several coverage analysis criteria for traditional programs have been proposed:
  - What is a coverage criterion?
- Tests that are adequate w.r.t a criterion:
  - Cover all elements of the domain determined by that criterion.

Test Coverage Criterion cont...

- But, what are the elements that characterize an object-oriented program?
  - Certainly different from procedural programs.
  - For example: Statement coverage is not appropriate due to inheritance and polymorphism.
- Appropriate test coverage criteria are needed.

Test Process Strategy

- Object-oriented development tends toward:
  - Shorter methods.
  - Complexity shifts from testing methods to class relations
  - In this context model based testing (also called grey box testing) of object-oriented programs assumes importance.

Integration Testing

- OO programs do not have a hierarchical control structure:
  - So conventional top-down and bottom-up integration tests have little meaning
- Integration applied three different incremental strategies:
  - Thread-based testing: integrates classes required to respond to one input or event
  - Use-based testing: integrates classes required by one use case
  - Cluster testing: integrates classes required to demonstrate one collaboration