# 第三部分 软件测试应用

## 3.2 集成测试

# 本节教学目标

- 理解单元测试的基本概念
- 理解单元测试的过程

# 目录

# 讨论

- 单元测试通过后，是否需要集成在一起进行测试？
  - 需要
  - 每个模块能够单独工作，但将这些模块集成在一起，某些模块有可能不能正常工作

# 集成测试的基本概念

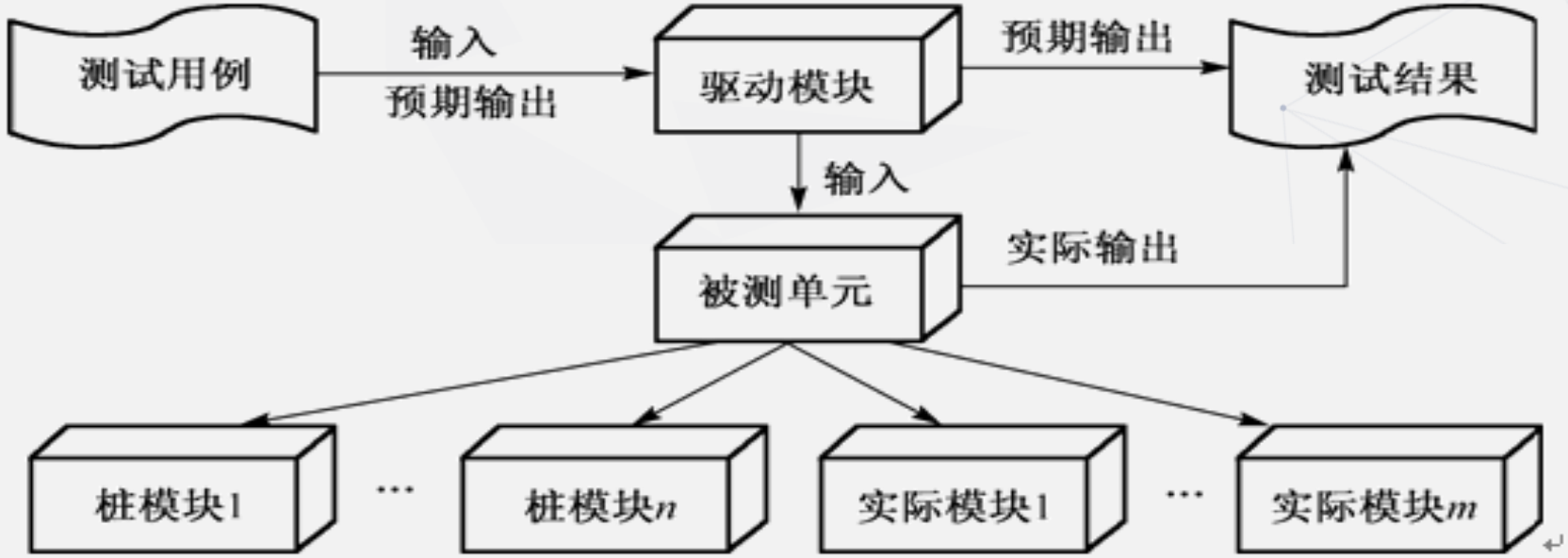
- 什么是集成测试(integration testing)
  - 集成测试就是在单元测试的基础上，将所有已通过单元测试的模块按照概要设计的要求组装为子系统或系统，并进行测试的过程，目的是确保各单元模块组合在一起后能够按既定意图协作运行，并确保增量的行为正确

# 集成测试内容

- 集成测试内容：
  - 将各个具有相互调用关系的模块组装起来时，检查穿越模块接口的数据是否会丢失
  - 判断各子功能组合起来能否达到预期要求的父功能
  - 检查一个模块的功能是否会对其他模块的功能产生不利影响
  - 检查全局数据结构是否正确，以及在完成模块功能的过程中是否会被异常修改
  - 单个模块的误差累积起来，是否会放大到不可接受的程度

# 集成测试内容

- 补充概念：
  - 驱动模块(Driver)：是模拟被测单元的上级模块，用于接收测试数据、启动被测模块和输出结果
  - 桩模块(Stub)：是模拟被测单元所调用的模块。有时，需要使用子模块的接口，才能做少量数据操作，并验证和打印入口处的信息，然后返回。桩模块不包含原模块的所有细节



# 目录

# 集成测试方法

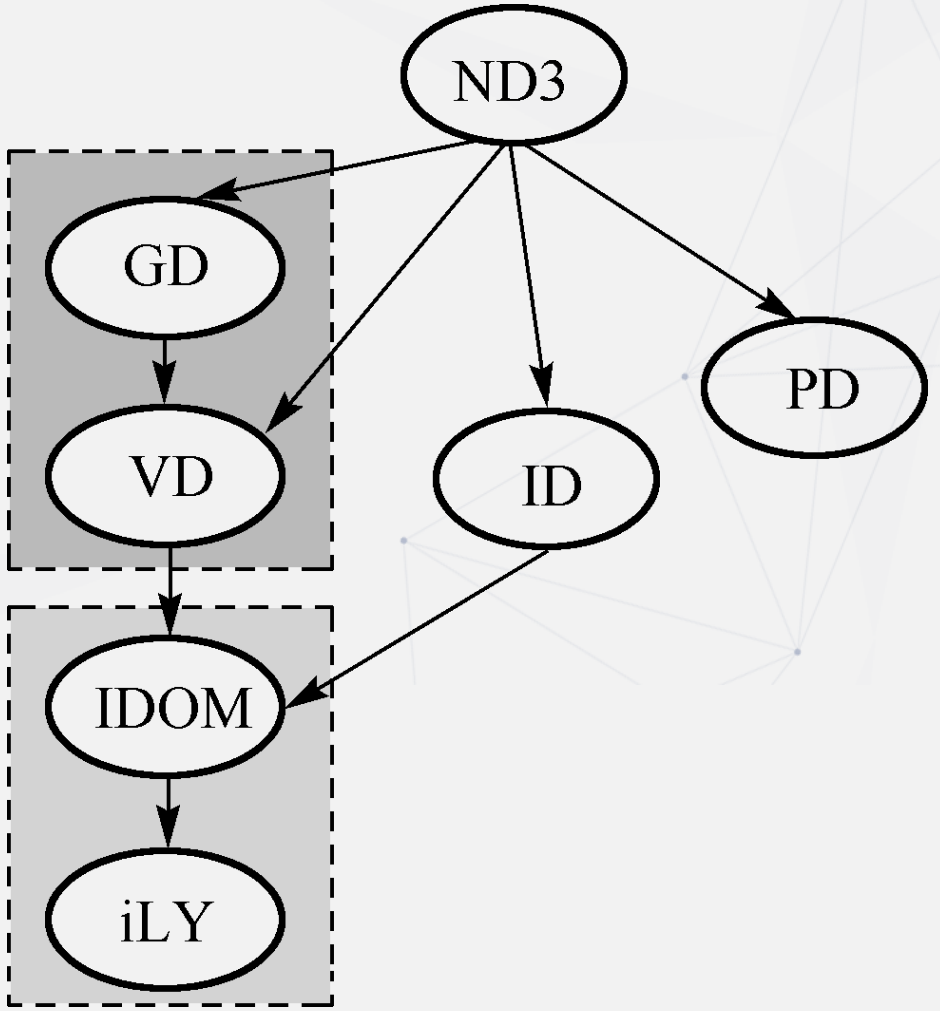
- 成对集成
- 邻居集成
- 基于独立路径的集成

# 集成测试方法—成对集成

- **基本思想**：将每个集成测试用例限定在一对**调用单元**上，每个集成测试用例都是最小的集成单元，仅涉及一对调用的接口
- **测试用例设计**：两个典型的模块成对集成
- **规模估算**：共m个模块，n条边，因每条边对应一对调用接口，确定一个成对测试用例，因此包含n个测试用例

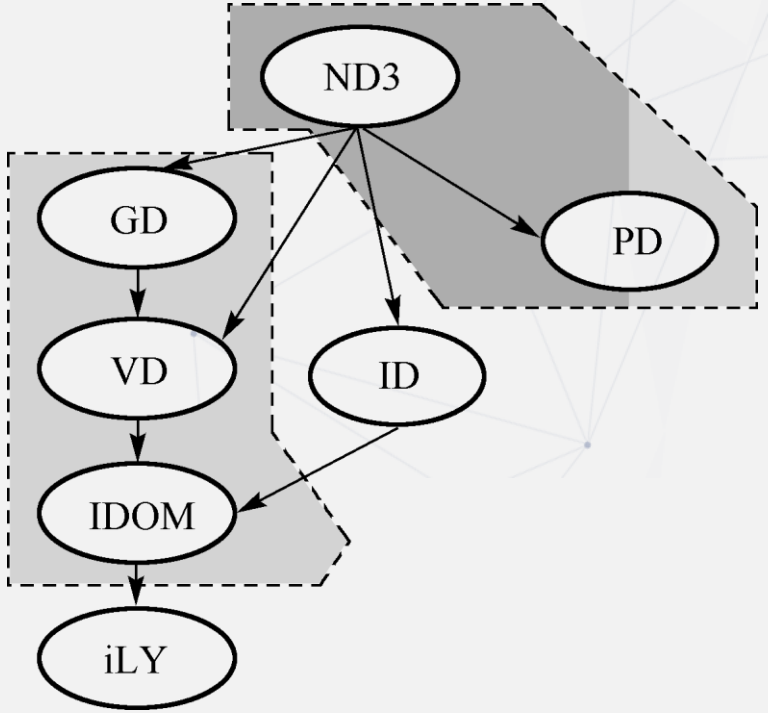
**特点分析**：目的希望避免开发驱动和桩模块，但事实没有做到（4个桩模块，3个驱动模块）

**优势**：容易定位缺陷



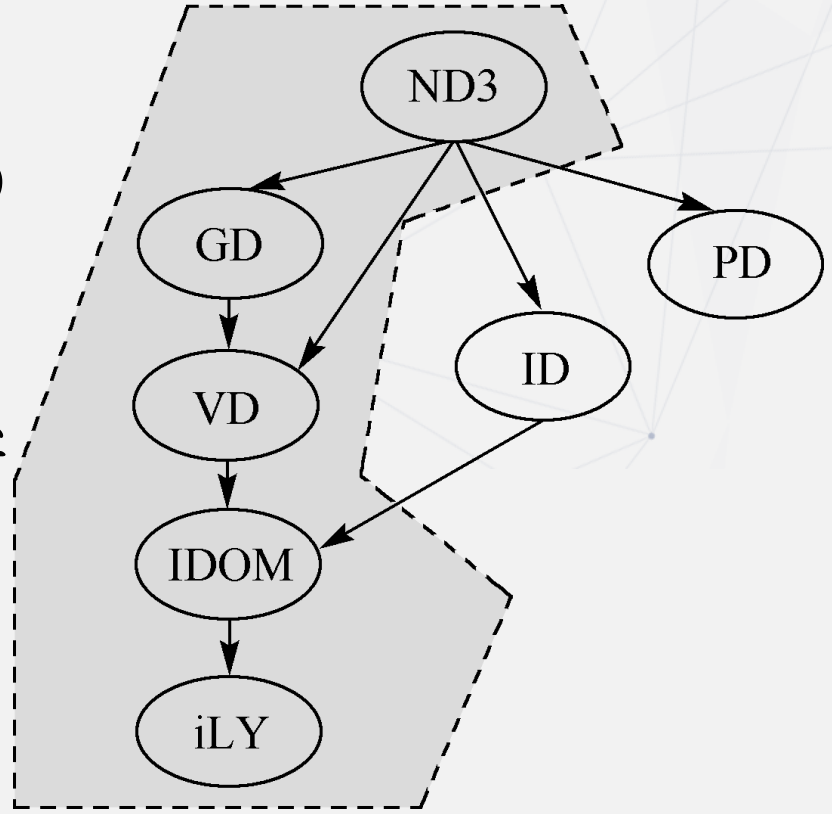
# 集成测试方法—邻居集成

- **概念**：邻居是指某个指定模块及其所有**直接调用**该模块的上层模块以及所有被该模块直接调用的下层模块
- 基本思想：将每个集成测试用例限定在某个节点的邻居上，针对某个模块的集成测试用例应同时包含该模块及其邻居
- **测试用例设计**：如图所示（5条）
- **规模**：共m个模块，其中n个模块是中间层的模块，根节点直接调用叶子节点，用例数量n+1；否则为n个
- **特点分析**：试图通过扩大单个测试用例的范围来减少测试用例的总数，导致的结果是缺陷定位变得困难
-



# 集成测试方法—基于独立路径的测试

- **基本思想**：将函数调用图看做程序的控制流图或程序图，每个从根节点到叶子节点的调用形成了路径，每条独立路径即可构成一个集成测试用例
- **测试用例设计**：
  - ND3-GD-VD-IDOM-iLY
  - ND3-VD-IDOM-iLY
  - ……（根据实际情况，如果存在不可行路径，则去掉）
- **规模估算**：环复杂度V，测试用例数量也是V，但是存在不可能路径，需要去掉或编写桩模块构造可行路径
- **特点分析**：减少桩和驱动模块开发量
- **不足**：缺陷定位困难



# 目录

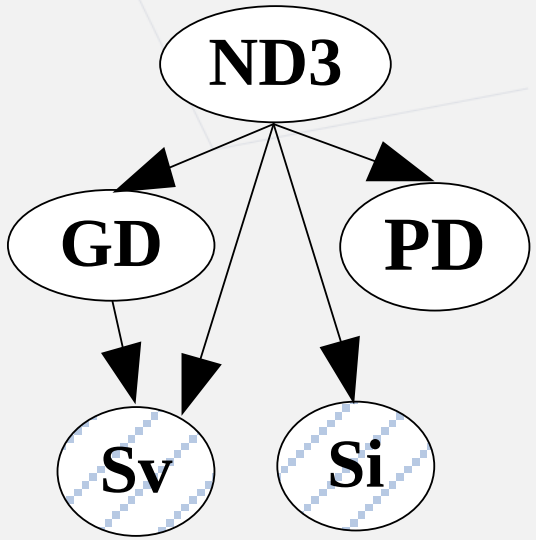
# 集成测试遍历顺序的设计—大爆炸集成

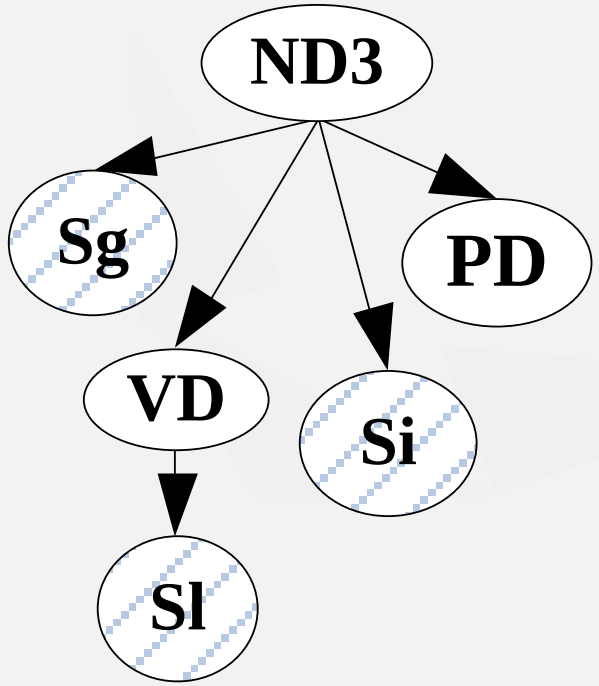
- **基本思想**：将所有经过单元测试的模块一次性组装到被测系统中进行测试，完全不考虑模块之间的依赖性和可能的风险
- **举例**：**将所有7个模块放在一起进行测试，即仅需一个测试用例，达到用例规模的最小化**
- **优点**：测试规模小
- **缺点**：违反了测试从小范围到大范围展开的原则，难以定位问题

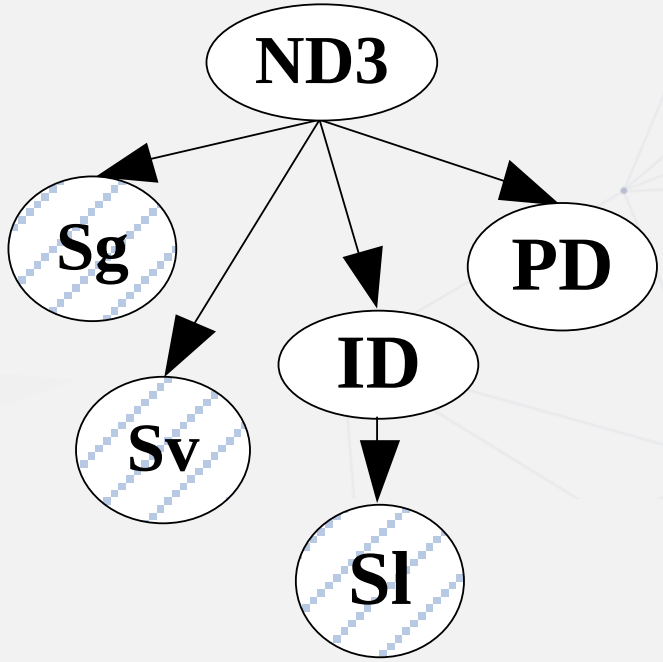
# 自顶向下集成

- **基本思想**： 从主控模块(主程序，即根节点)开始，按照系统程序结构，沿着控制层次从上而下，逐渐将各模块组装起来

- 深度优先：

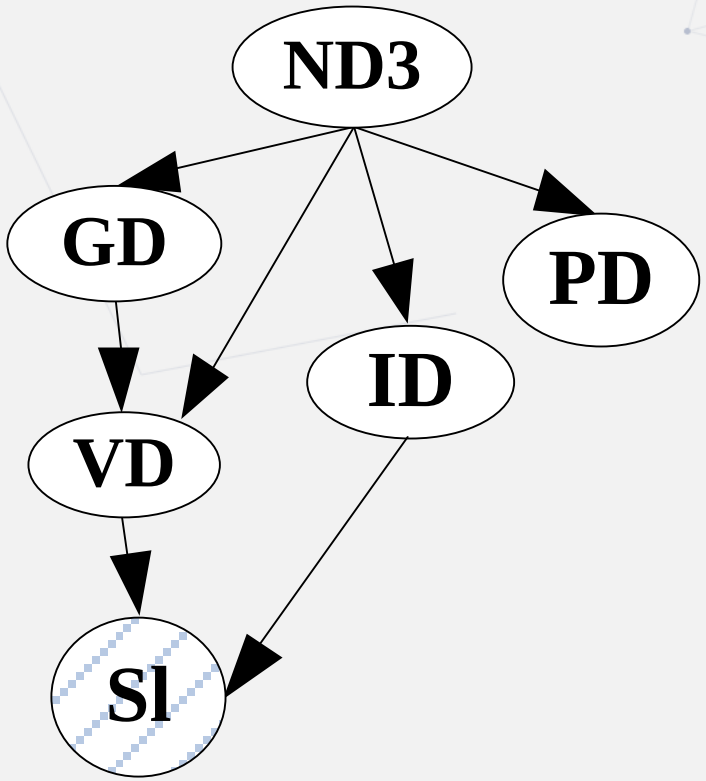


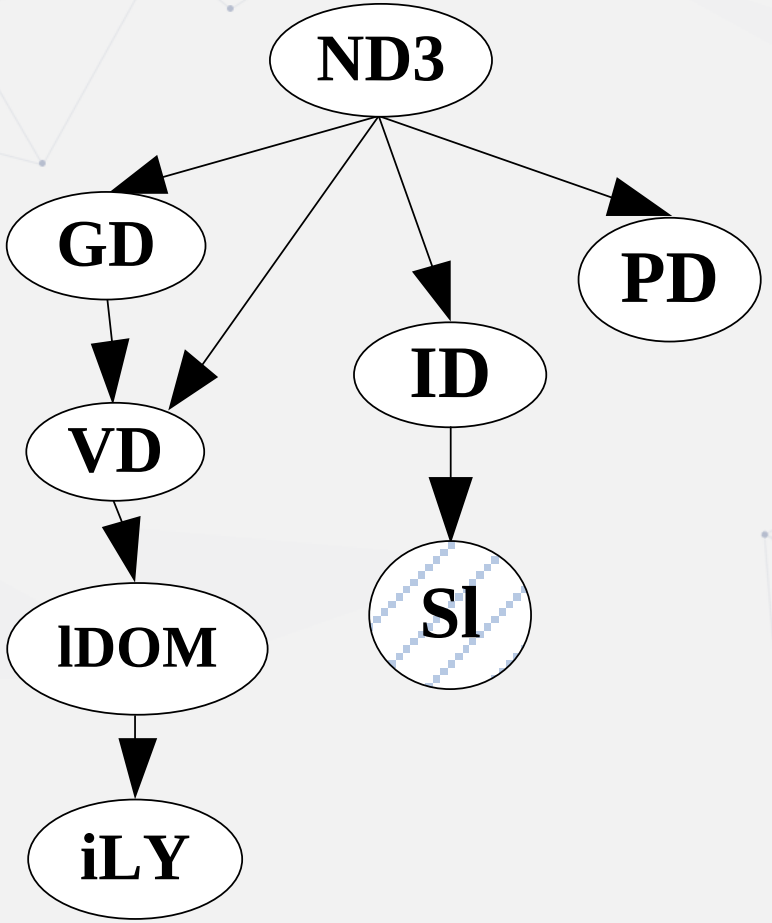



# 自顶向下集成

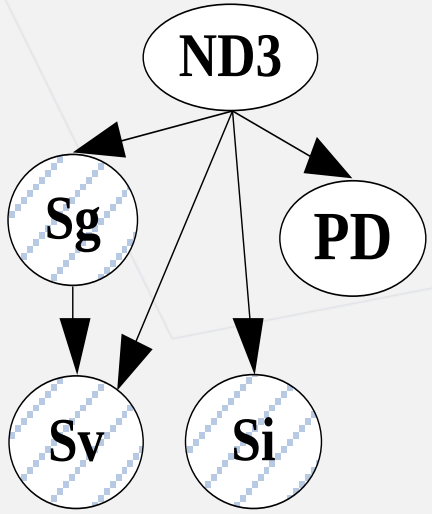
- 继续集成



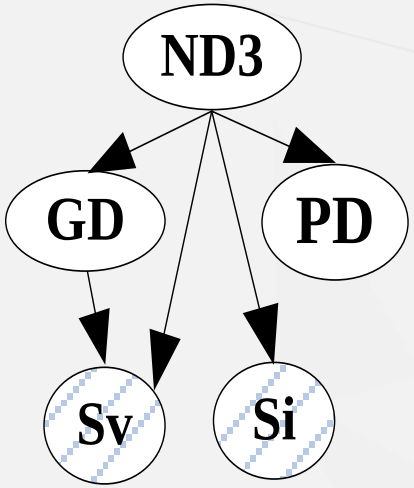



# 自顶向下集成

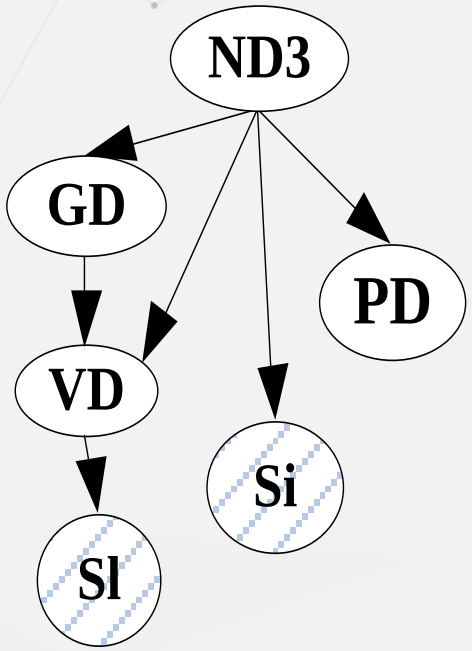
- 以广度优先策略自顶向下集成的测试用例设计：



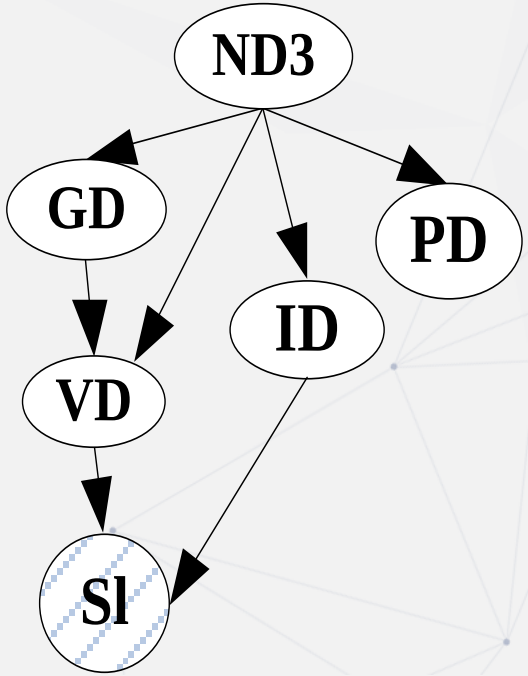

（a）测试NextDate3




（b）加入GetDate




（c）加入ValidDate




（d）加入IncrementDate

# 自顶向下的集成

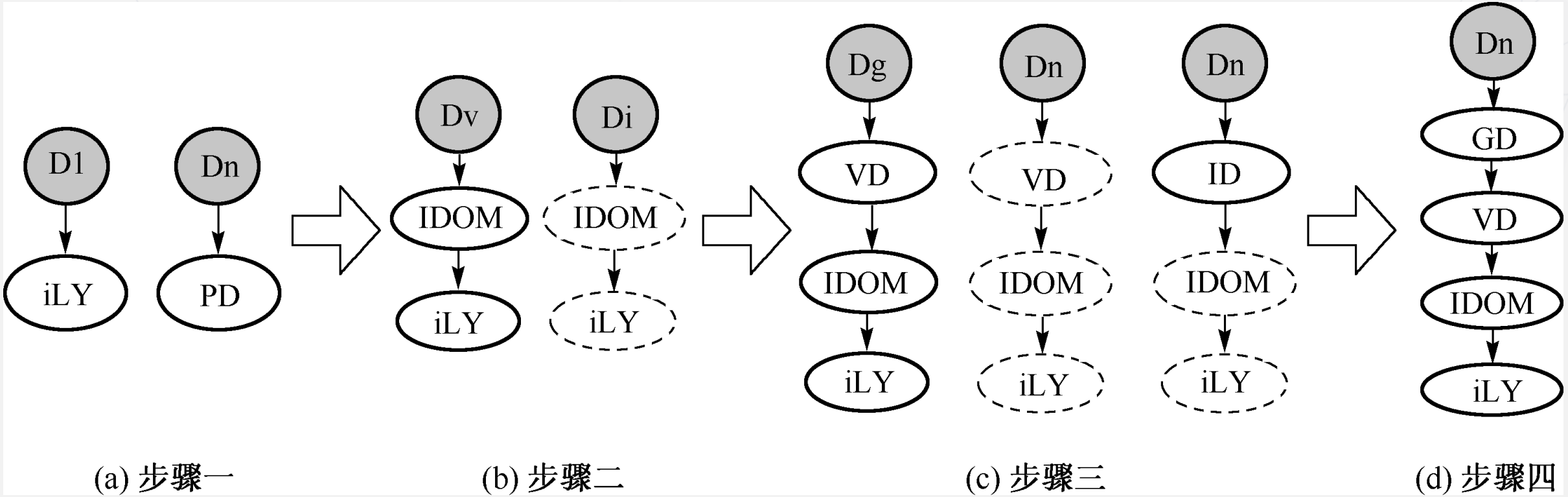
- 优势
  - 优先从根节点开始测试，有助于早期实现并验证系统主要功能，给开发团队和用户带来成功的信心，也便于早期验证主要的控制和判断，避免主控程序的缺陷，确保开发进度
  - 单个测试用例包含多个模块，可从整体上降低测试用例规模
  - 采用递增方式展开测试，每个新的测试用例一般仅加入一个新的模块，便于缺陷定位

# 自顶向下的集成

- 不足
  - 桩模块的开发和维护工作量较大
  - 难以早期发现底层模块中复杂算法的缺陷，且随着测试的进行，系统越来越复杂，底层模块的测试很难保证充分性
  - 不利于测试的并行，难以充分展开人力

# 自底向上的集成

- 自底向上的集成(Bottom Up)
- **基本思想**：从底层模块(即叶子节点)开始，按照调用图的结构，从下而上，逐层将各模块组装起来



# 自底向上的集成

- 优势
  - 优先从叶子节点开始测试，有助于早期发现底层模块中复杂算法的缺陷，且驱动模块的开发有利于规范和约束系统上层模块的设计，在一定程度上增加系统可测试性
  - 单个测试用例包含多个模块，可从整体上降低测试用例规模
  - 多个集成测试可并行展开，确保测试工作进度

# 自底向上的集成

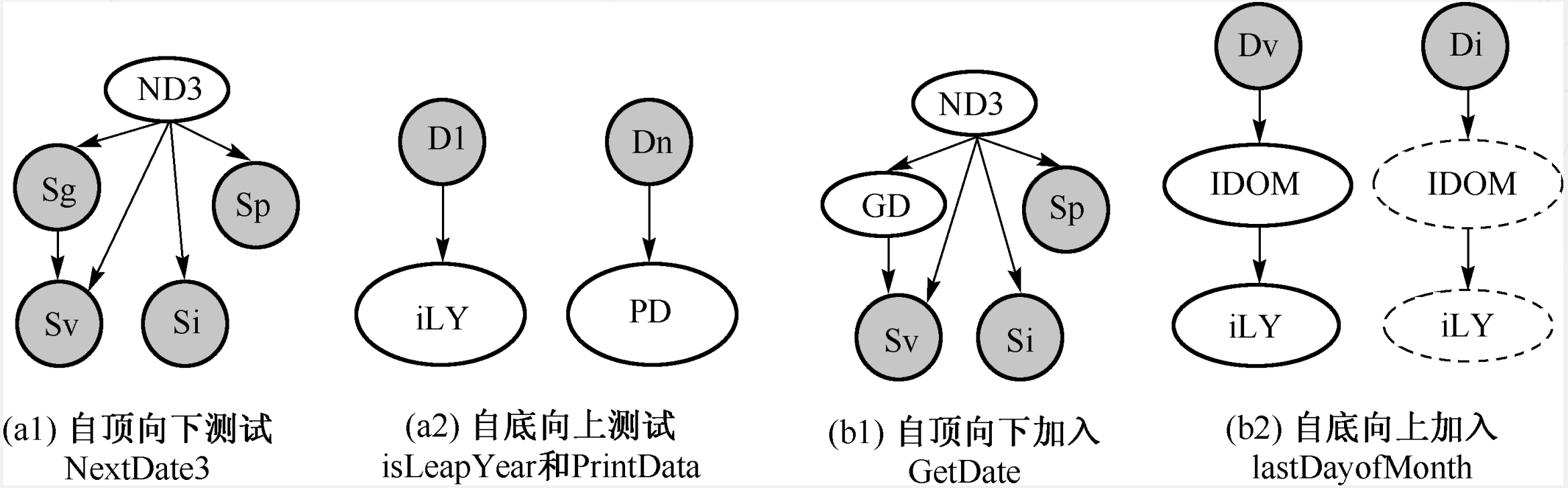
- 不足
  - 驱动模块的开发和维护工作量较大
  - 难以早期发现上层模块中有关逻辑和控制方面的缺陷
  - 直至加入最后一个模块才能看到整个系统框架，难以早期发现时序问题和资源竞争问题

# 混合集成

- 另一种叫法：三明治集成(Sandwich)
- 基本思想：将自顶向下和自底向上集成方法结合起来的集成策略。在调用图上按照一定的策略，分别自顶向下和自底向上展开集成，并在子树上进行大爆炸集成

# 混合集成

- **策略1**：将系统划分为三层，中间层为目标层，测试时对目标层上面的层使用自顶向下的集成策略，对目标层下面的层使用自底向上的集成策略




(a1) 自顶向下测试 NextDate3

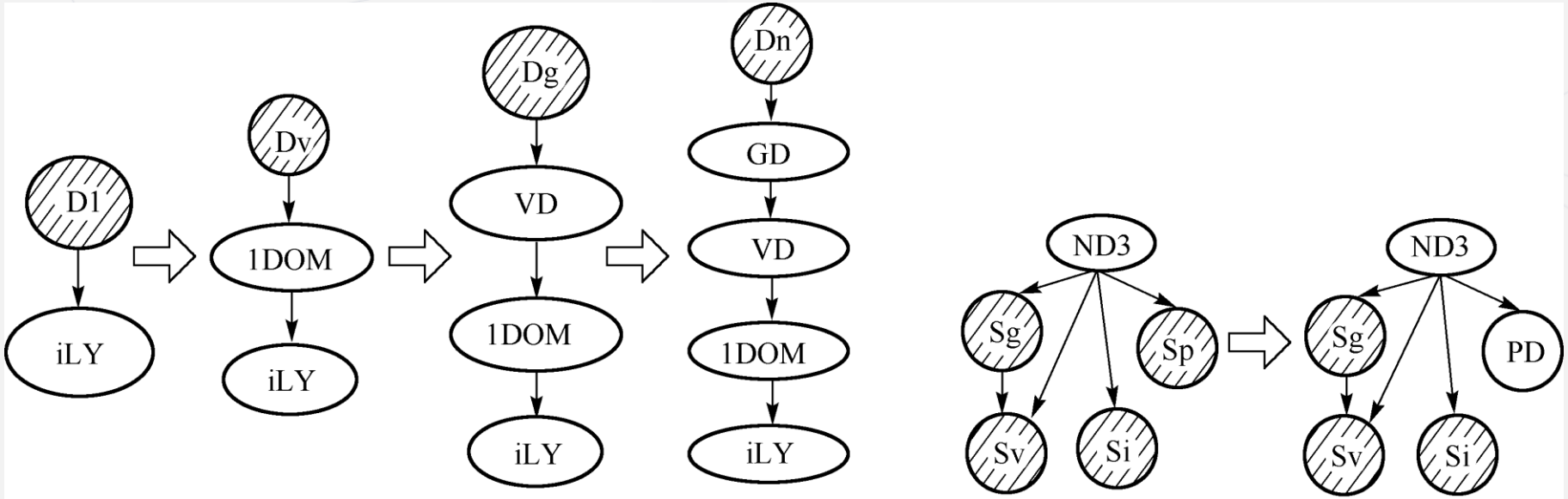
(a2) 自底向上测试 isLeapYear和PrintData

(b1) 自顶向下加入 GetDate

(b2) 自底向上加入 lastDayofMonth

# 混合集成

- 策略2：基于策略1并对目标层采用独立测试策略，确保目标层模块在集成测试之前得到充分的测试
- 策略3：对包含读操作的子系统自底向上集成测试直至根节点，然后对包含写操作的子系统自顶向下集成测试直至叶子节点




(a) 自底向上集成测试　　(b) 自顶向下集成测试

# 混合集成

- 优势：结合了自顶向下和自底向上的集成的优势
- 不足：
  - 中间的目标层可能得不到充分的测试
  - 需要同时开发桩和驱动模块，这部分工作量比较大
  - 需在子树上进行大爆炸集成，一旦发现缺陷，涉及的接口数量较多，增加了缺陷定位难度

# 集成测试策略的比较

| 项目 | 测试用例数目 | 桩模块 | 驱动模块 | 缺陷定位 | 并行测试 | 系统概貌 |
|---|---|---|---|---|---|---|
| 成对集成 | 由边数决定 | 需要 | 需要 | 非常容易 | 可以 | 不确定 |
| 邻居集成 | 主要由中间节点数决定 | 需要 | 需要 | 困难 | 可以 | 不确定 |
| 大爆炸 | 少 | 不需要 | 不需要 | 非常困难 | N/A | 早期 |
| 自顶向下 | 较多 | 需要 | 不需要 | 较容易 | 困难 | 早期 |
| 自底向上 | 较多 | 不需要 | 需要 | 较容易 | 可以 | 较晚 |
| 三明治 | 较多 | 需要 | 需要 | 较困难 | 可以 | 早期 |

# 内容总结

- 集成测试的基本概念
- 集成测试的方法：成对、邻居、基于独立路径
- 集成测试遍历顺序：大爆炸、自顶向下、自底向上、混合
- 集成测试策略的比较

# Question