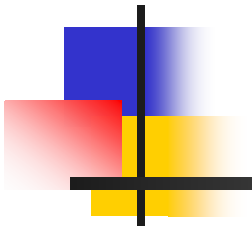
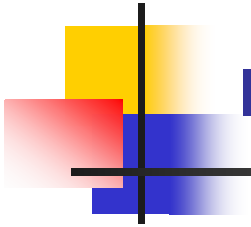


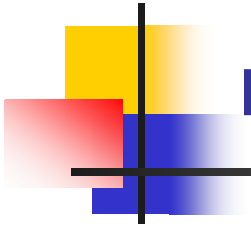
# Integration Testing
# Functional Decomposition Based

Chapter 13



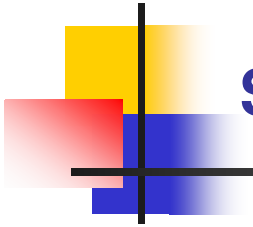
# Integration Testing

- Test the interfaces and interactions among separately tested units
- Three different approaches
  - **Based on functional decomposition**
  - **Based on call graphs**
  - **Based on paths**



# Functional Decomposition

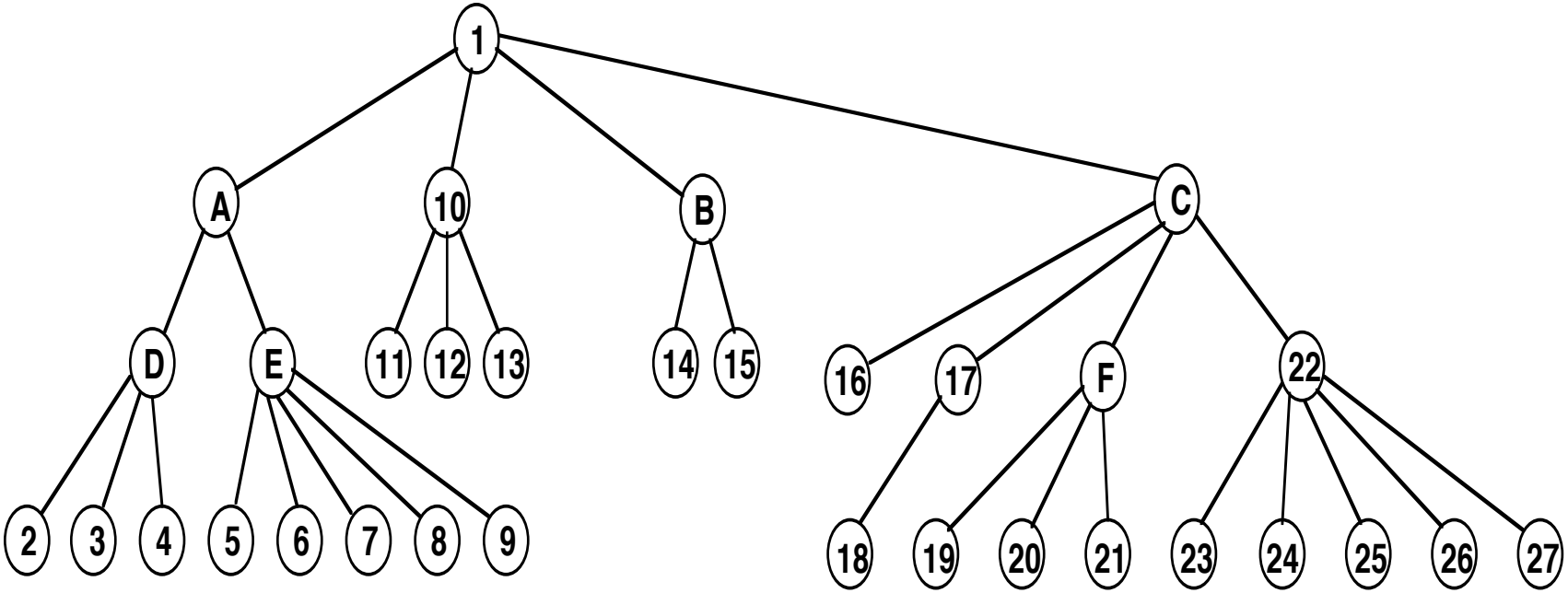
- Functional Decomposition
  - **Create a functional hierarchy for the software**
  - **Problem is broken up into independent task units, or functions**
  - **Units can be run either**
    - **Sequentially and in a synchronous call-reply manner**
    - **Or simultaneously on different processors**
- Used during planning, analysis and design

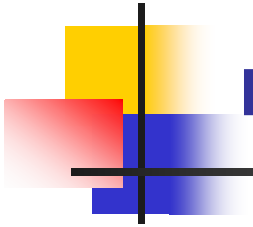


# SATM Units

| Unit | Level | Name |
|---|---|---|
| 1 | 1 | SATM system |
| A | 1.1 | Device sense & control |
| D | 1.1.1 | Door sense & control |
| 2 | 1.1.1.1 | Get door status |
| 3 | 1.1.1.2 | Control door |
| 4 | 1.1.1.3 | Dispense cash |
| E | 1.1.2 | Slot sense & control |
| 5 | 1.1.2.1 | Watch card slot |
| 6 | 1.1.2.2 | Get deposit slot status |
| 7 | 1.1.2.3 | Control card Roller |
| 8 | 1.1.2.4 | Control Envelope Roller |
| 9 | 1.1.2.5 | Read card strip |
| 10 | 1.2 | Central bank comm. |
| 11 | 1.2.1 | Get PIN for PAN |
| 12 | 1.2.2 | Get account status |
| 13 | 1.2.3 | Post daily transactions |
| B | 1.3 | Terminal sense & control |
| 14 | 1.3.1 | Screen door |
| 15 | 1.3.2 | Key sensor |
| C | 1.4 | Manage session |
| 16 | 1.4.1 | Validate card |
| 17 | 1.4.2 | Validate PIN |
| 18 | 1.4.2.1 | Get PIN |
| F | 1.4.3 | Close session |
| 19 | 1.4.3.1 | New transaction request |
| 20 | 1.4.3.2 | Print receipt |
| 21 | 1.4.3.3 | Post transaction local |
| 22 | 1.4.4 | Manage transaction |
| 23 | 1.4.4.1 | Get transaction type |
| 24 | 1.4.4.2 | Get account type |
| 25 | 1.4.4.3 | Report balance |
| 26 | 1.4.4.4 | Process deposit |
| 27 | 1.4.4.5 | Process withdrawal |

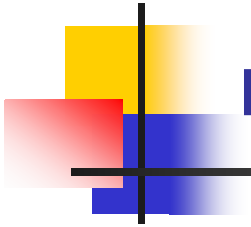
# SATM functional decomposition tree





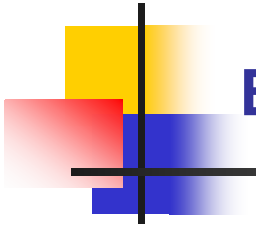
# Decomposition-based integration strategies

- **What are the decomposition-based integration strategies?**



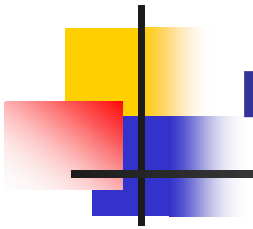
# Decomposition-based integration strategies – 2

- Top-down
- Bottom-up
- Sandwich
- Big bang



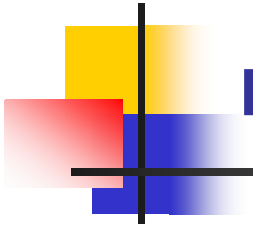
# Big bang integration process

- **What is the big bang integration process.**



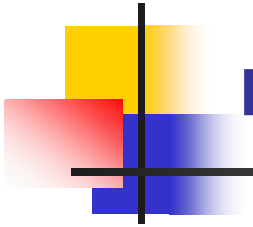
# Big bang integration process – 2

- All units are compiled together
- All units are tested together



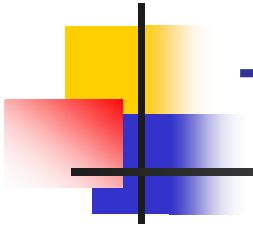
# Big bang integration issues

- **What are the issues (advantages and drawbacks)?**



# Big bang integration issues – 2

- Failures will occur!
- No clues to isolate location of faults
- No stubs or drivers to write

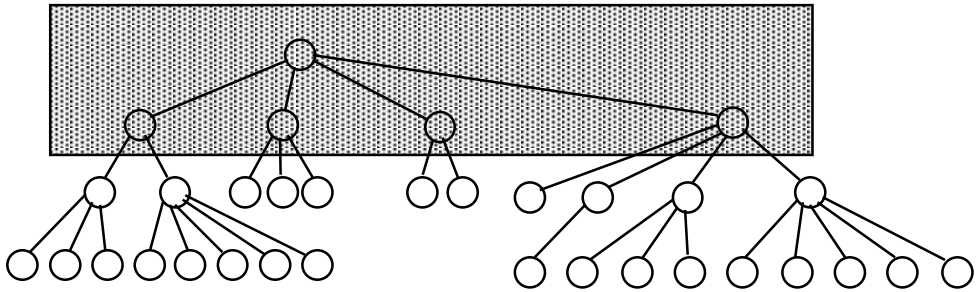


# Top-down integration

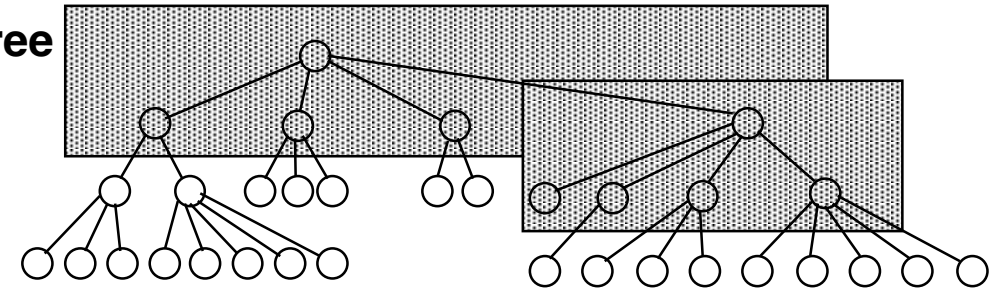
- **What is the top-down integration process?**

# Top-Down integration example

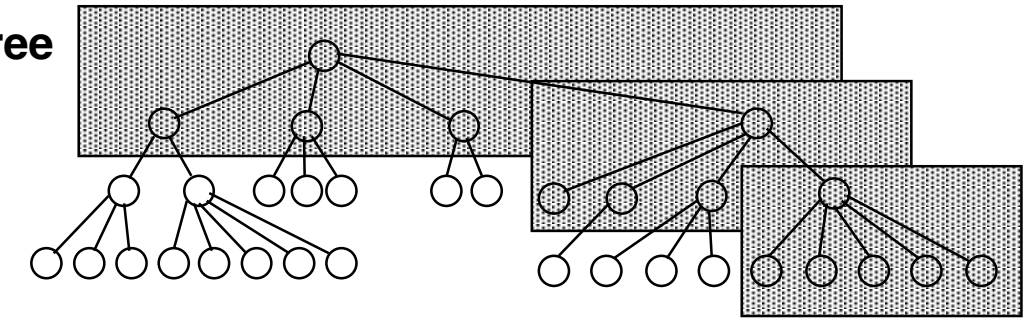
**Top Subtree**
**Sessions 1-4**

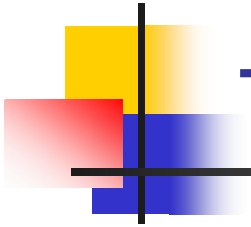


**Second Level Subtree**
**Sessions 5-8**



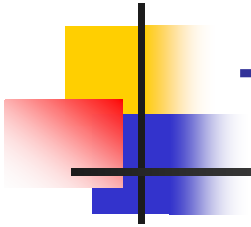
**Bottom Level Subtree**
**Sessions 9-13**





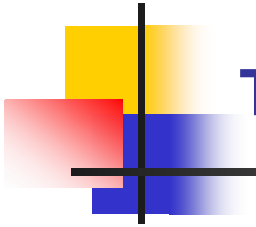
# Top-Down integration process

- Strategy
  - **Focuses on testing the top layer or the controlling subsystem first**
    - **The main, or the root of the call tree**
- General process is
  - **To gradually add more subsystems that are referenced/required by the already tested subsystems when testing the application**
  - **Do this until all subsystems are incorporated into the test**



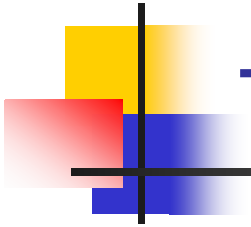
# Top-Down integration process – 2

- **Stubs** are needed to do the testing
  - **A program or a method that simulates the input-output functionality of a missing subsystem by answering to the decomposition sequence of the calling subsystem and returning back simulated data**



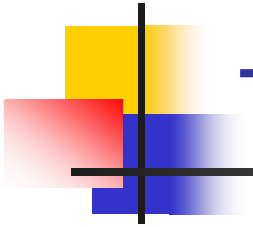
# Top-Down integration issues

- **What are the issues?**



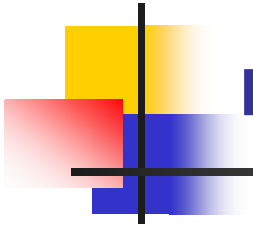
# Top-Down integration issues – 2

- Writing stubs can be difficult
  - **Especially when parameter passing is complex.**
  - **Stubs must allow all possible conditions to be tested**
- Possibly a very large number of stubs may be required
  - **Especially if the lowest level of the system contains many functional units**



# Top-Down integration issues – 3

- One solution to avoid too many stubs
  - **Modified top-down testing strategy**
  - **Test each layer of the system decomposition individually before merging the layers**
  - **Disadvantage of modified top-down testing**
    - **Both stubs and drivers are needed**

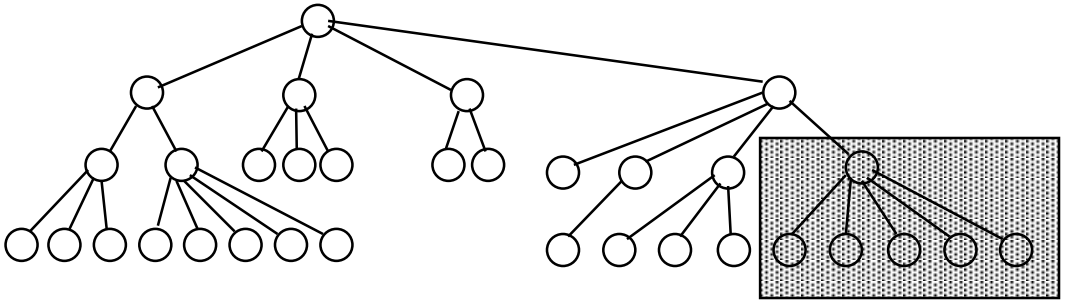


# Bottom-up integration

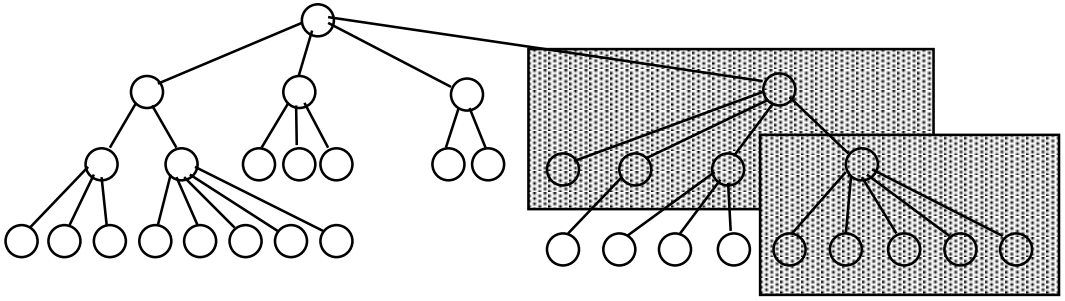
- **What is the bottom-up integration process?**

# Bottom-up integration example

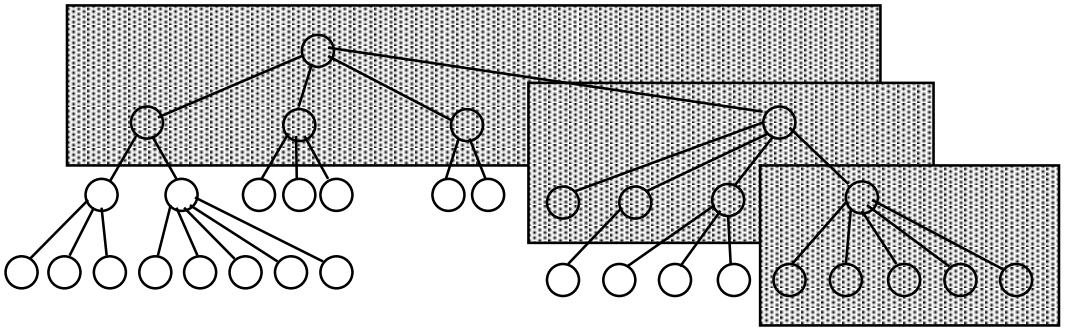
**Bottom Level Subtree**
**Sessions 1-5**

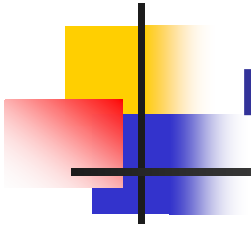


**Second Level Subtree**
**Sessions 6-9**



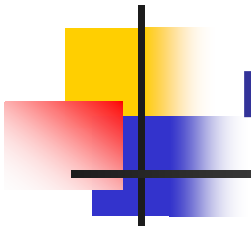
**Top Subtree**
**Sessions 10-13**





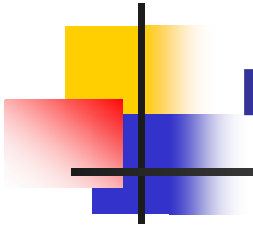
# Bottom-Up integration process

- Bottom-Up integration strategy
  - **Focuses on testing the units at the lowest levels first**
  - **Gradually includes the subsystems that reference/require the previously tested subsystems**
  - **Do until all subsystems are included in the testing**



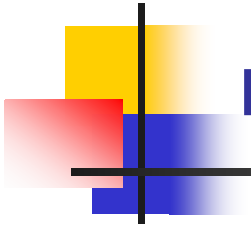
# Bottom-Up integration process – 2

- **Drivers** are needed to do the testing
  - **A driver is a specialized routine that passes test cases to a subsystem**
    - **Subsystem is not everything below current root module, but a sub-tree down to the leaf level**



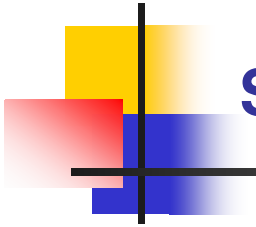
# Bottom-up integration issues

- What are the issues?



# Bottom-Up Integration Issues

- Not an optimal strategy for functionally decomposed systems
  - **Tests the most important subsystem (user interface) last**
- More useful for integrating object-oriented systems
- Drivers may be more complicated than stubs
- Less drivers than stubs are typically required

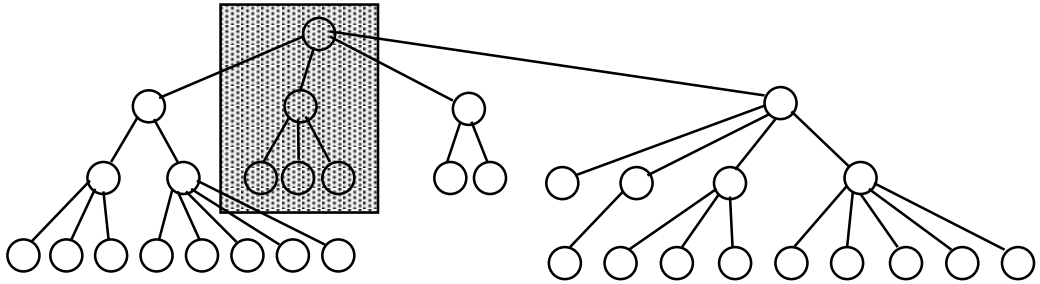


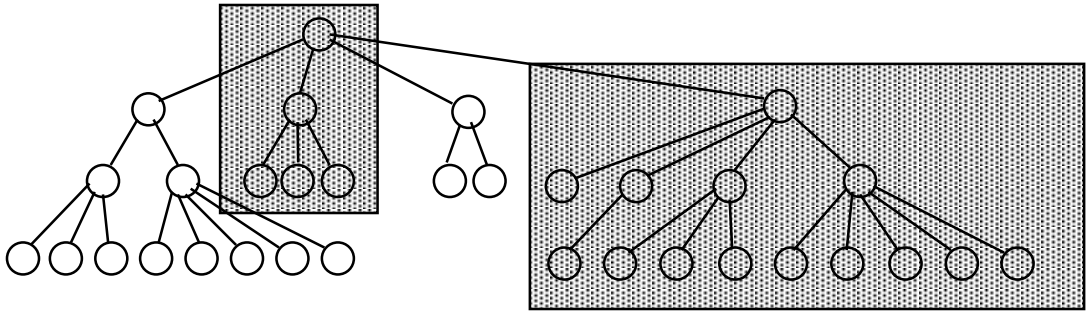
# Sandwich integration

- **What is the sandwich integration process?**

# Sandwich integration example

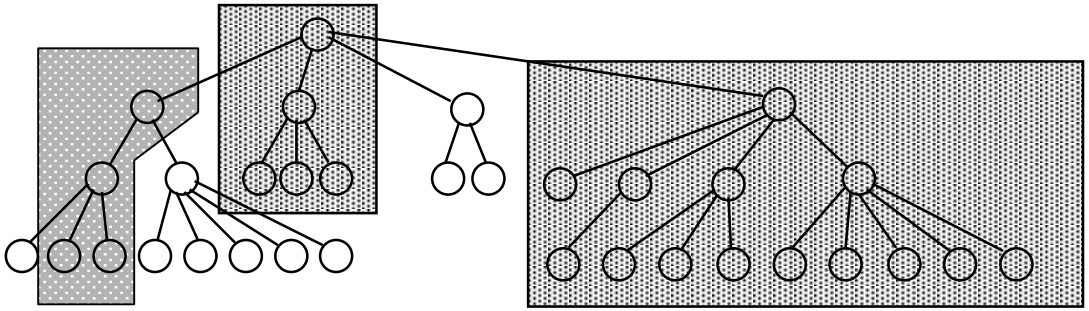
**Sandwich 1**
**Sessions 1-3**

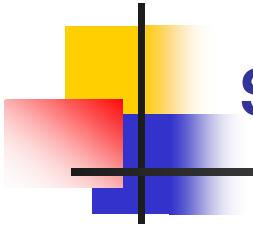




**Sandwich 2**
**Sessions 4-13**

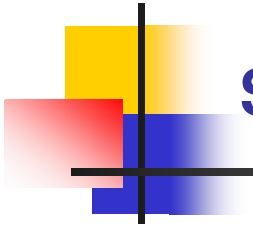
**Sandwich 3**
**Sessions 14-15**





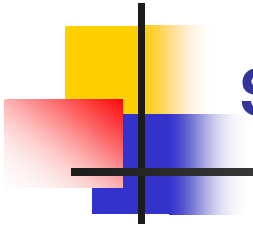
# Sandwich integration process

- Combines top-down strategy with bottom-up strategy
  - **Doing big bang on a subtree**



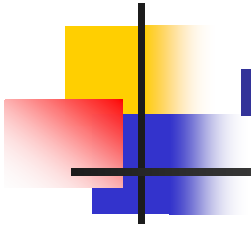
# Sandwich integration issues

- **What are the issues?**



# Sandwich integration issues – 2

- Less stub and driver development effort
- Added difficulty in fault isolation



# Integration test session

- A session is a test suite that tests one edge in the tree
  - **Each session tests the combining of two parts**

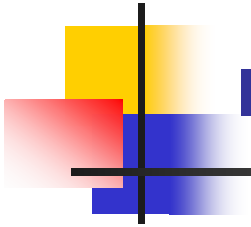
    **#sessions = #edges**

  - **This is different from the textbook**

    **#sessions = #nodes – #leaves + #edges**
    **= 2 #edges – #leaves + 1**

    **Alternately**

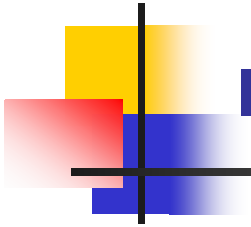
    **#sessions = #internal_nodes + #edges**



# Integration work numbers

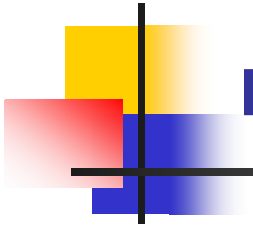
- For top-down integration
  - **#nodes – 1 = #edges  stubs are needed**
- For bottom-up integration
  - **#nodes – #leaves = #internal_nodes**
    **drivers are needed**
- The number integrated units for top-down and bottom-up

  **#integrated_units = #internal_nodes**



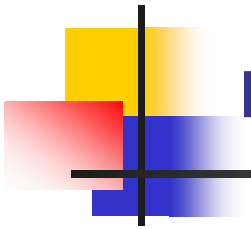
# Integration work numbers

- For SATM have 32 integration test sessions
  - **Correspond to 32 separate sets of test cases**
- For top-down integration
  - **32 stubs are needed**
- For bottom-up integration
  - **10 drivers are needed**
- For top-down and bottom-up
  - **10 integration units**



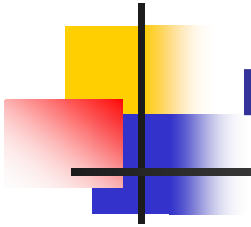
# Decomposition-based drawback

- **What is the major drawback of decomposition-based integration?**



# Decomposition-based drawback – 2

- It is functionally based
  - **Has the problems of all functional testing**
  - **How do we overcome the problems?**



# Decomposition-based drawback – 3

- It is functionally based
  - **Has the problems of all functional testing**
  - **How do we overcome the problems?**
    - **Move to structural-based testing**