SOFTWARE TESTING

CO3015 / CO5252

CH5. UNIT / INTEGRATION TESTING

Content

- Unit testing
- Test case design
- Incremental testing
- Top-down vs. Bottom-up testing

Unit testing

- A process of testing the individual sub-program, subroutines, class or procedures in a program
- Testing a program = initially focus on small building blocks of the program
- Motivation:
  - *Focus initially on small units*
  - *Help debuging: error => where it is (in the unit-under-test)*
  - *Test multiple modules simultaneously*

Unit testing

- 1. The manner in which test cases are designed.
- 2. The order in which modules should be tested and integrated.
- 3. Advice about performing the tests.

Test case design

- Using white-box and/or black-box testing techniques
- Strategy
  - *Combination of input conditions: cause-effect graphing*
  - *Any event: boundary value analysis (input and output)*
  - *Identify the valid/invalid equivalence classes for input and output*
  - *Use the error-guessing techniques*
  - *Others:*
    - Decision coverage, condition coverage, …

Test case design

- Summary
  - *Logic coverage*
  - *Equivalence partitioning*
  - *Boundary value analysis*
  - *Cause-effect graphing*
  - *Error guessing techniques*

https://reqtest.com/testing-blog/test-case-design-techniques/

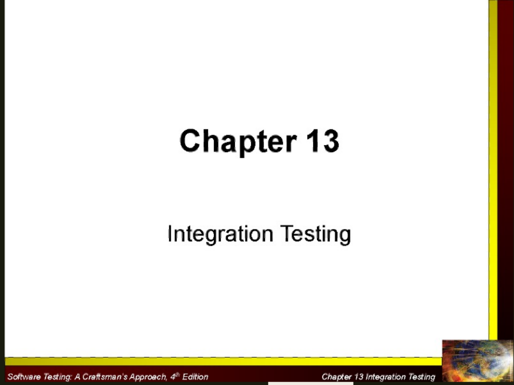
Test case design

- Techniques:
  - *Specification-Based techniques*
  - *Structure-Based techniques*
  - *Experience-Based techniques*
- Specification-Based techniques = Black-box techniques
  - *Boundary Value Analysis*
  - *Equivalence Partitioning*
  - *Decision Table Testing*
  - *State Transition Diagrams*
  - *Use Case Testing*
- Structure-Based techniques = White-box techniques
  - *Statement Testing & Coverage*
  - *Decision Testing Coverage*
  - *Condition Testing*
  - *Multiple Condition Testing*
  - *All Path Testing*
- Experience-Based techniques
  - *Error Guessing*
  - *Exploratory Testing*

Exploratory Testing

- Reading

https://www.guru99.com/exploratory-testing.html



Chapter 13

Integration Testing

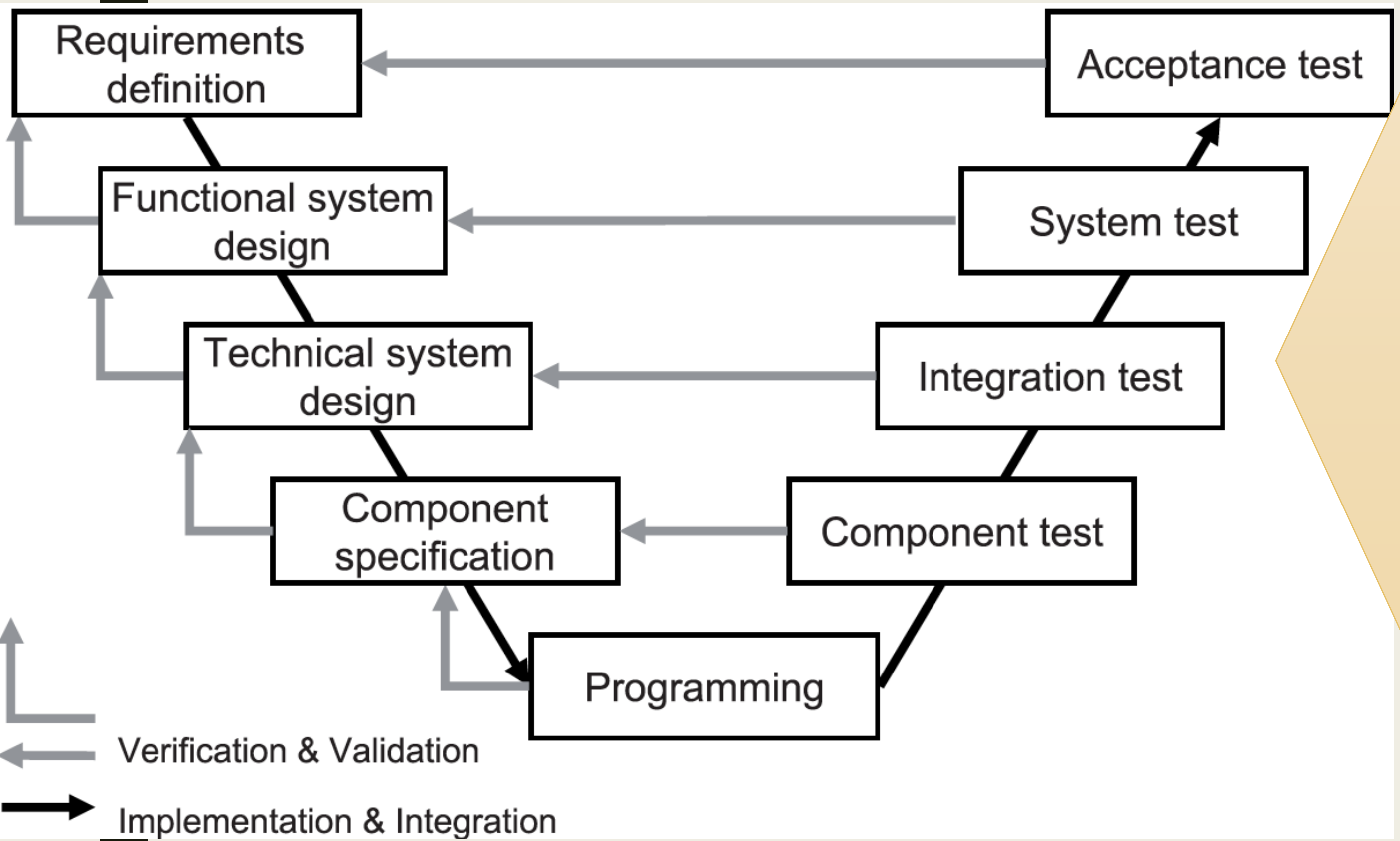
Software Testing: A Craftsman's Approach, 4th Edition

Chapter 13 Integration Testing

INTEGRATION TESTING

https://www.tutorialspoint.com/software_testing_dictionary/incremental_testing.htm

Integration testing



- The process of verifying the **interfaces** and **interaction between modules**.
- Techniques:
  - *Incremental integration testing: integrate modules one-by-one*
  - *Big bang: all the modules are integrated in one shot.*

Testing Level Assumptions and Objectives

- Unit assumptions
  - *All other units are correct*
  - *Compiles correctly*
- Integration assumptions
  - *Unit testing complete*
- System assumptions
  - *Integration testing complete*
  - *Tests occur at port boundary*

- Unit goals
  - *Correct unit function*
  - *Coverage metrics satisfied*
- Integration goals
  - *Interfaces correct*
  - *Correct function across units*
  - *Fault isolation support*
- System goals
  - *Correct system functions*
  - *Non-functional requirements tested*
  - *Customer satisfaction.*

Approaches to Integration Testing

- **Functional Decomposition**
  - *Top-down*
  - *Bottom-up*
  - *Sandwich*
  - *“Big bang”*
- Call graph
  - *Pairwise integration*
  - *Neighborhood integration*
- Paths
  - ***MM-Paths***
  - *Atomic System Functions*

Basis of Integration Testing Strategies

- Functional Decomposition
  - *applies best to procedural code*
- Call Graph
  - *applies to both procedural and object-oriented code*
- MM-Paths
  - *applies to both procedural and object-oriented code*

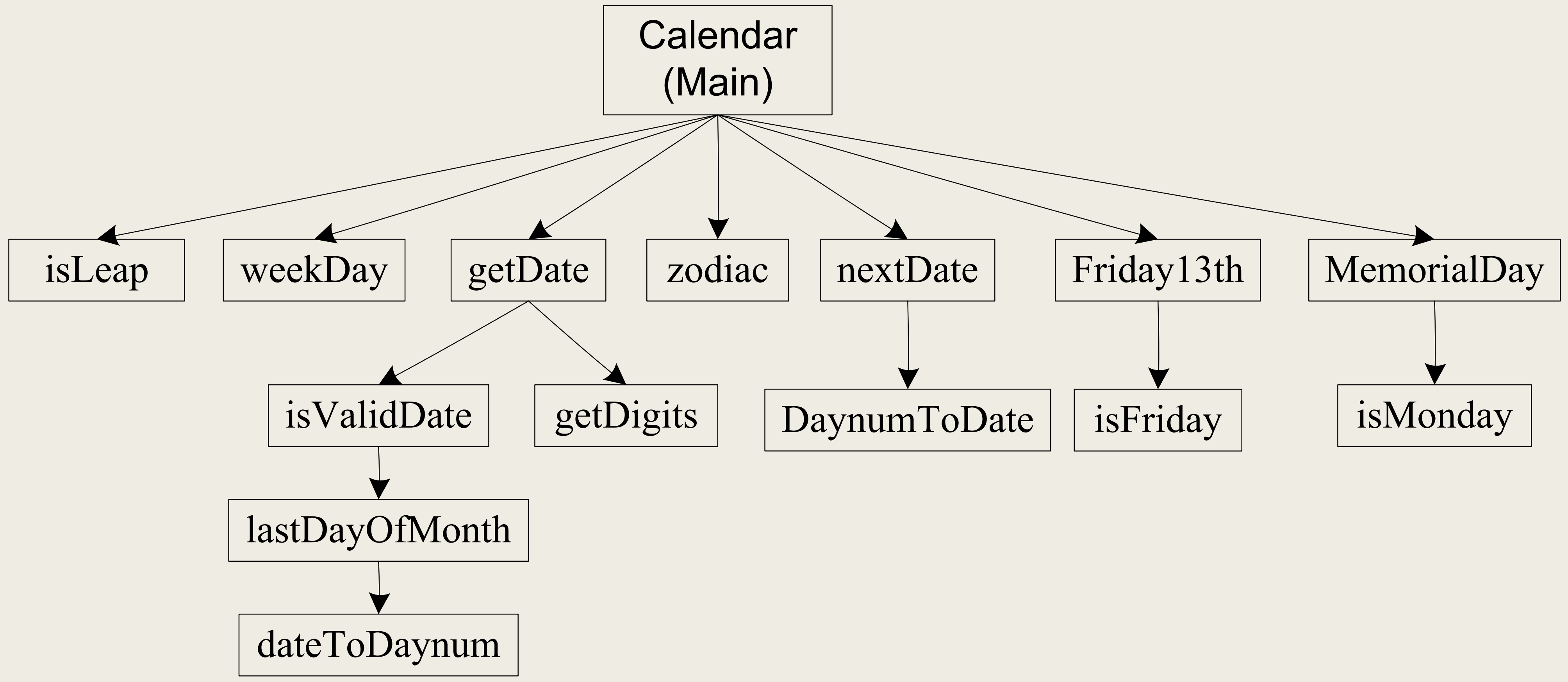
Continuing Example—Calendar Program

- Date in the form mm, dd, yyyy
- Calendar functions
  - *the date of the next day (NextDate)*
  - *the day of the week corresponding to the date*
  - *the zodiac sign of the date*
  - *the most recent year in which Memorial Day was celebrated on May 27*
  - *the most recent Friday the Thirteenth*

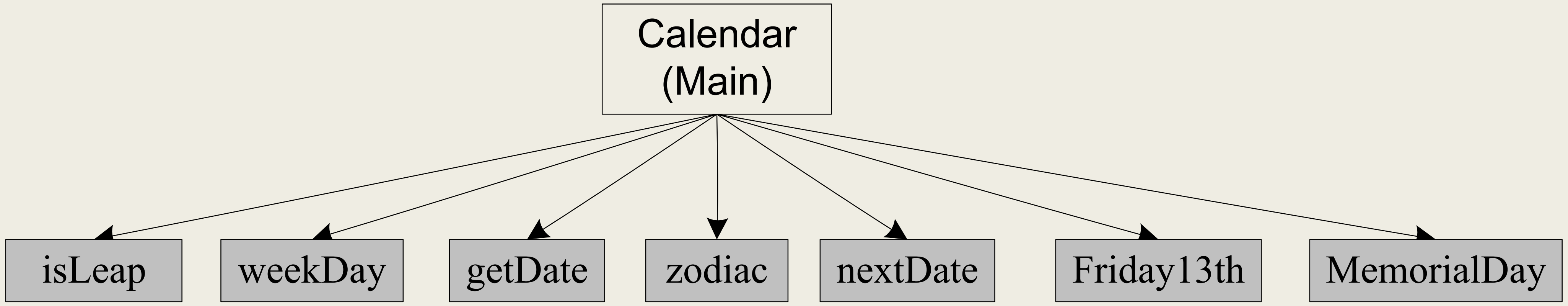
Calendar Program Units

```
Main    Calendar
    Function isLeap
    Procedure weekDay
    Procedure getDate
        Function isValidDate
            Function lastDayOfMonth
        Procedure getDigits
    Procedure memorialDay
            Function isMonday
    Procedure friday13th
        Function isFriday
    Procedure nextDate
        Procedure dayNumToDate
    Procedure zodiac
```

Functional Decomposition of Calendar



First Step in Top-Down Integration



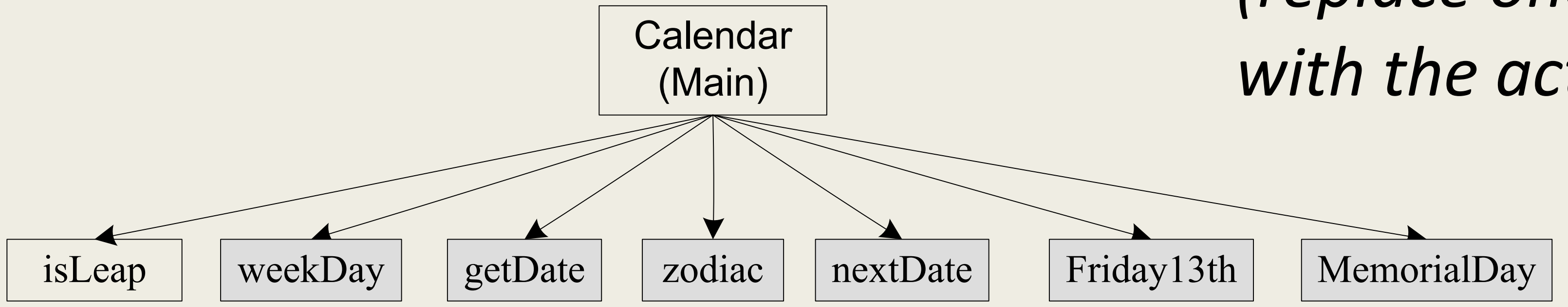
"Grey" units are **stubs** that return the correct values when referenced. This level checks the main program logic.

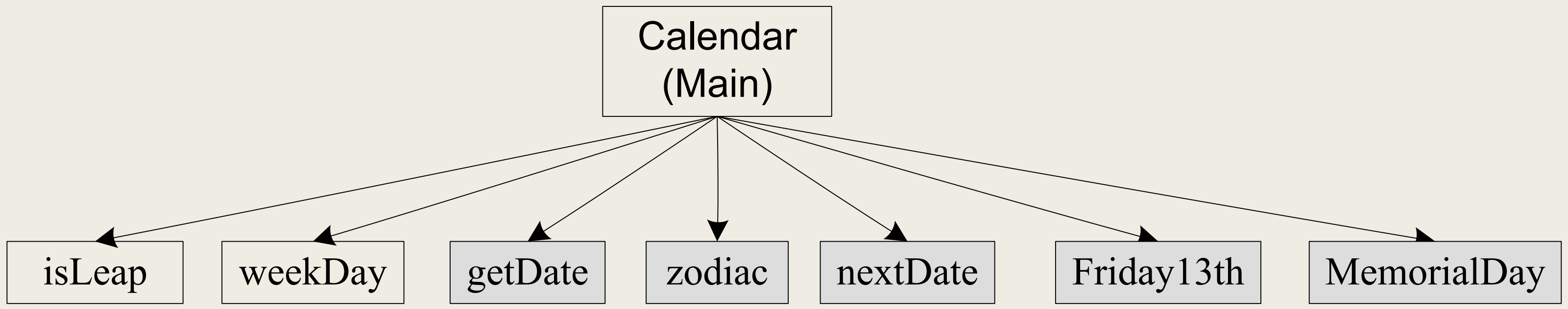
weekDayStub

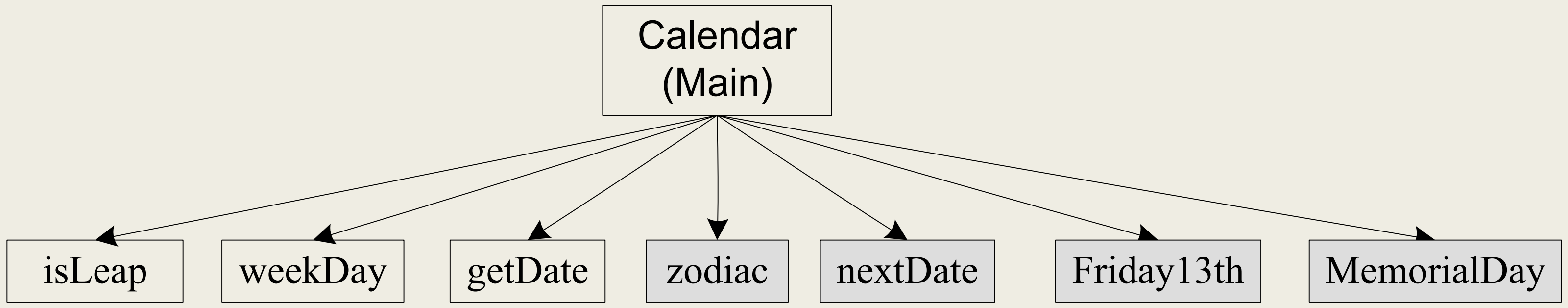
```
Procedure weekDayStub(mm, dd, yyyy, dayName)
If ((mm = 10) AND (dd = 28) AND (yyyy = 2013))
    Then dayName = “Monday”
EndIf
.
.
.
If ((mm = 10) AND (dd = 30) AND (yyyy = 2013))
    Then dayName = “Wednesday”
EndIf
```

Next Three Steps

(replace one stub at a time with the actual code.)







Top-Down Integration Mechanism

- Breadth-first traversal of the functional decomposition tree.
- First step: Check main program logic, with all called **units replaced by stubs** that always return correct values.
- Move down one level
  - ***replace*** *one* ***stub*** *at a time with* ***actual code****.*
  - *any fault must be in the newly integrated unit*

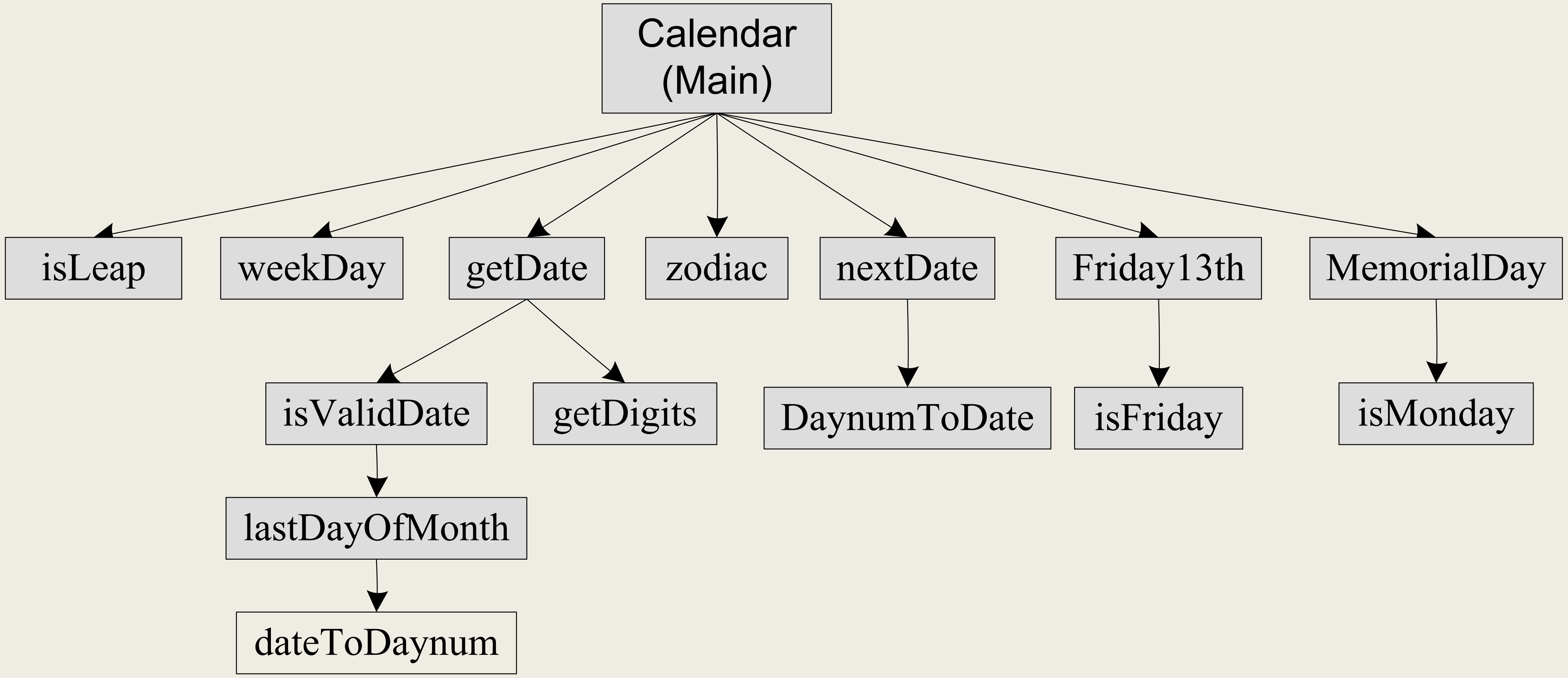
Bottom-Up Integration Mechanism

- Reverse of top-down integration
- Start at leaves of the functional decomposition tree.
- **Driver** units...
  - *call next level unit*
  - *serve as a small test bed*
  - *"drive" the unit with inputs*
  - *drivers know expected outputs*
- As with top-down integration, one driver unit at a time is replaced with actual code.
- Any fault is (most likely) in the newly integrated code.

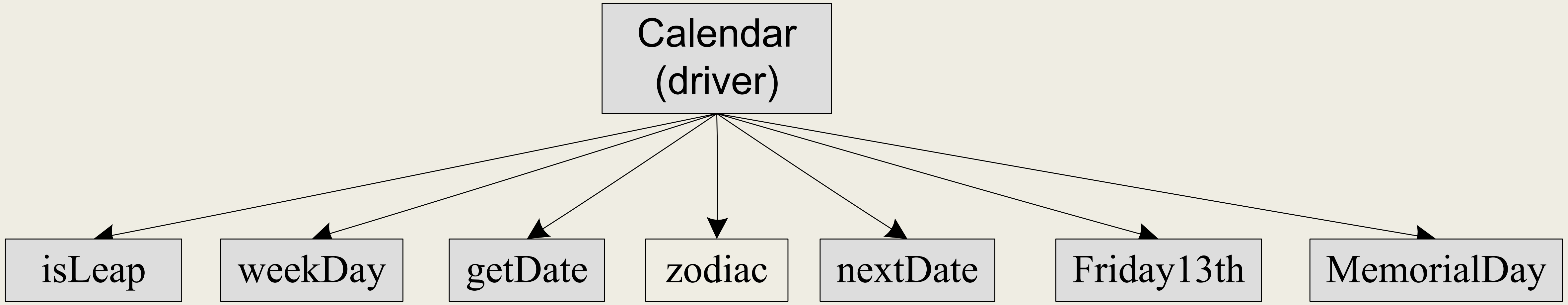
Top-Down and Bottom-Up Integration

- Both depend on throwaway code.
  - *drivers are usually more complex than stubs*
- Both test just the interface between two units at a time.
- In Bottom-Up integration, a driver might simply reuse unit level tests for the “lower” unit.
- Fan-in and fan-out in the decomposition tree results is some redundancy.

Starting Point of Bottom-Up Integration



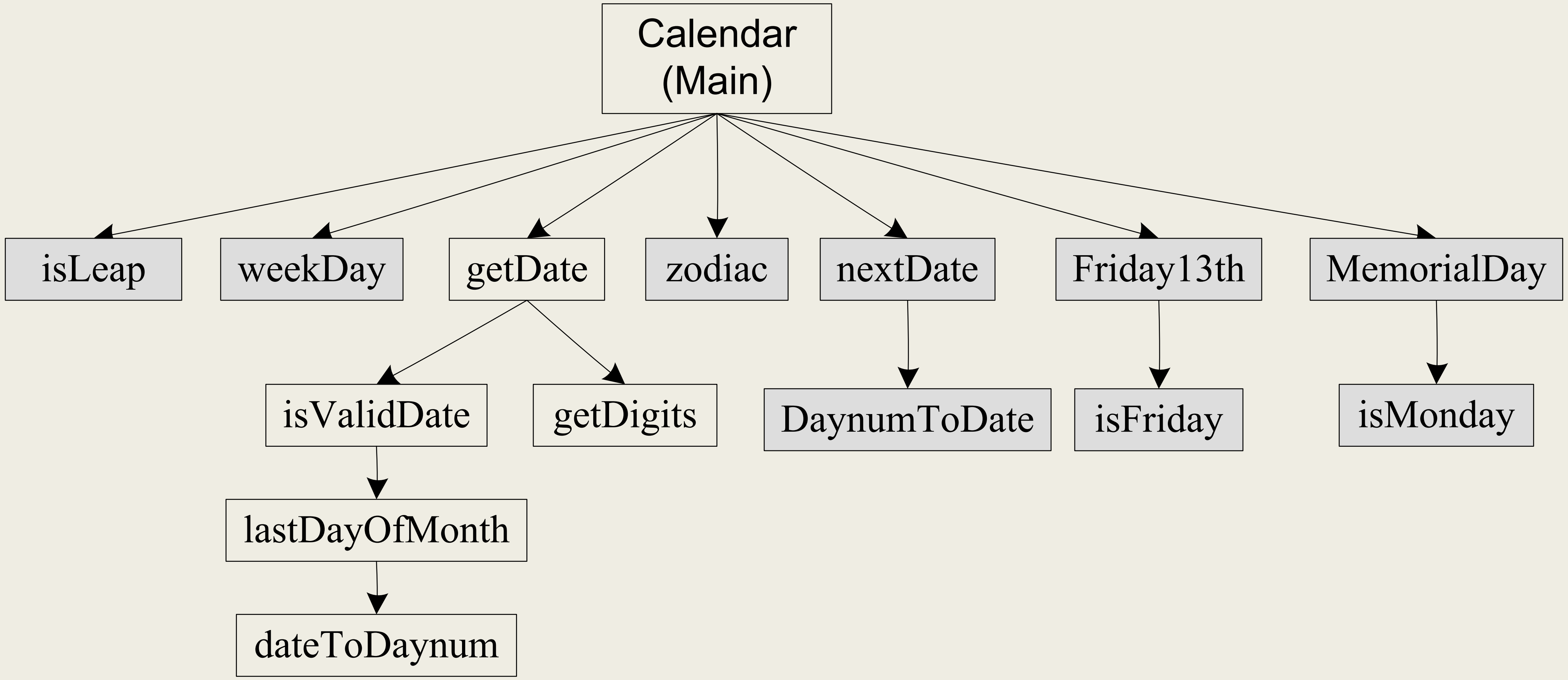
Bottom-Up Integration of Zodiac



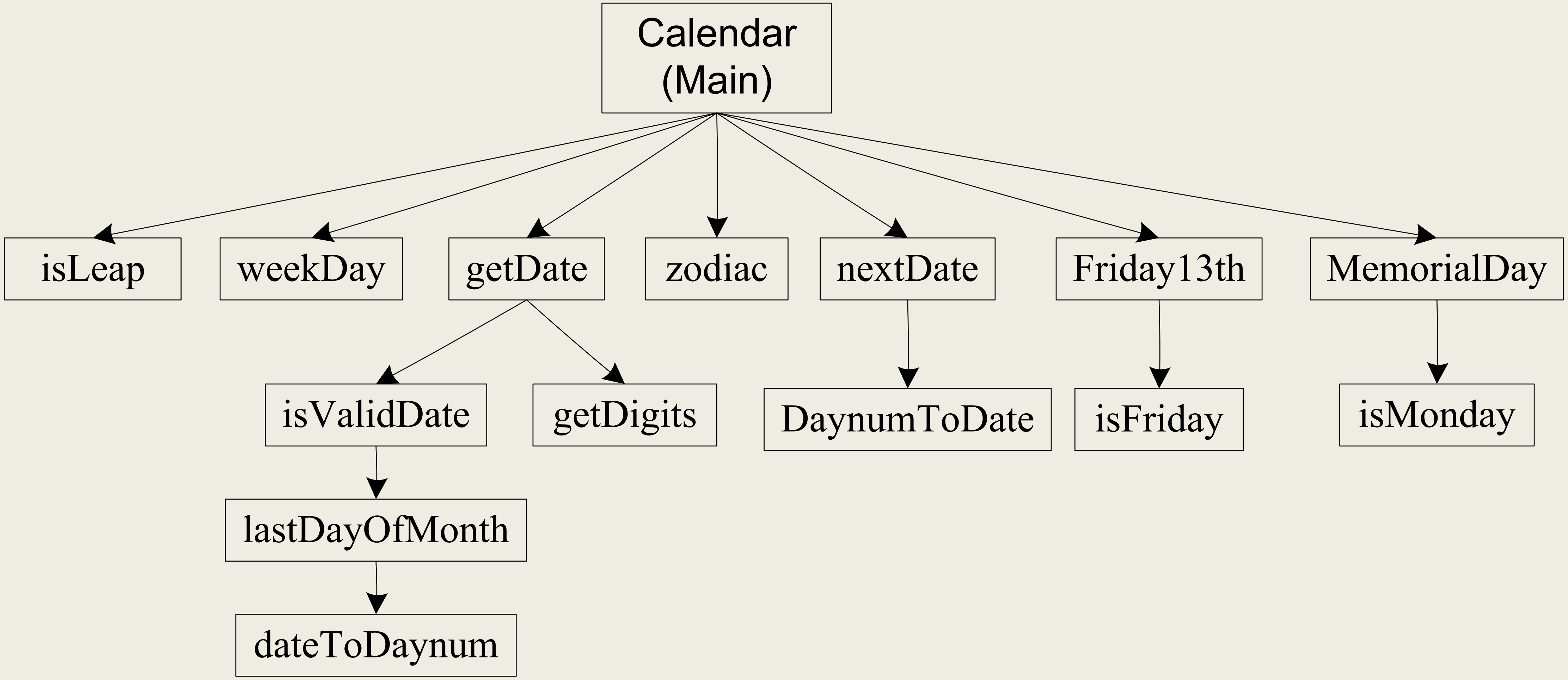
Sandwich Integration

- Avoids some of the repetition on both top-down and bottom-up integration.
- Nicely understood as a depth-first traversal of the functional decomposition tree.
- A “sandwich” is one path from the root to a leaf of the functional decomposition tree.
- Avoids stub and driver development.
- More complex fault isolation.

A Sample Sandwich



“Big Bang” Integration



“Big Bang” Integration

- No...
  - *stubs*
  - *drivers*
  - *strategy*
- And very difficult fault isolation
- This is the practice in an agile environment with a daily run of the project to that point.

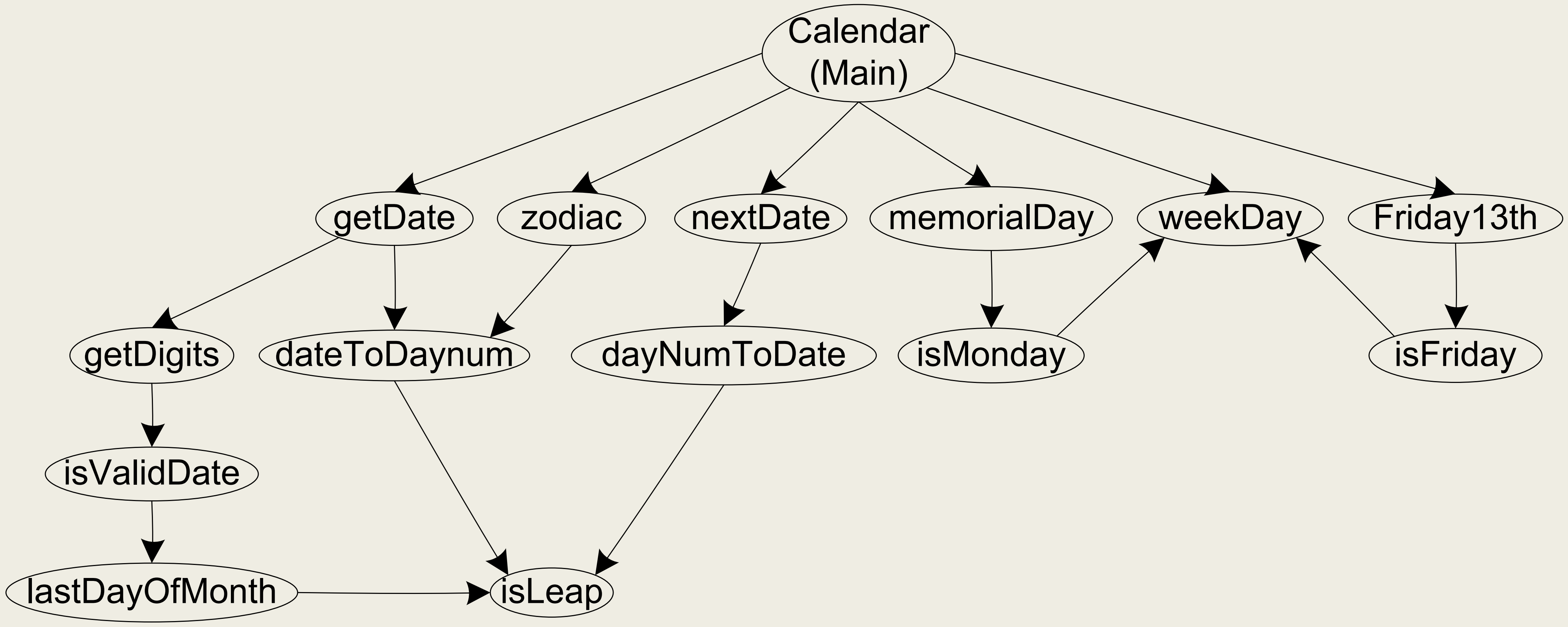
Pros and Cons of Decomposition-Based Integration

- Pros
  - *intuitively clear*
  - *“build” with proven components*
  - *fault isolation varies with the number of units being integrated*
- Cons
  - *based on lexicographic inclusion (a purely structural consideration)*
  - *some branches in a functional decomposition may not correspond with actual interfaces.*
  - *stub and driver development can be extensive*

Call Graph-Based Integration

- Definition: The Call Graph of a program is a directed graph in which
  - *nodes are unit*
  - *edges correspond to actual program calls (or messages)*
- Call Graph Integration avoids the possibility of impossible edges in decomposition-based integration.
- Can still use the notions of stubs and drivers.
- Can still traverse the Call Graph in a top-down or bottom-up strategy.

Call Graph of the Calendar Program

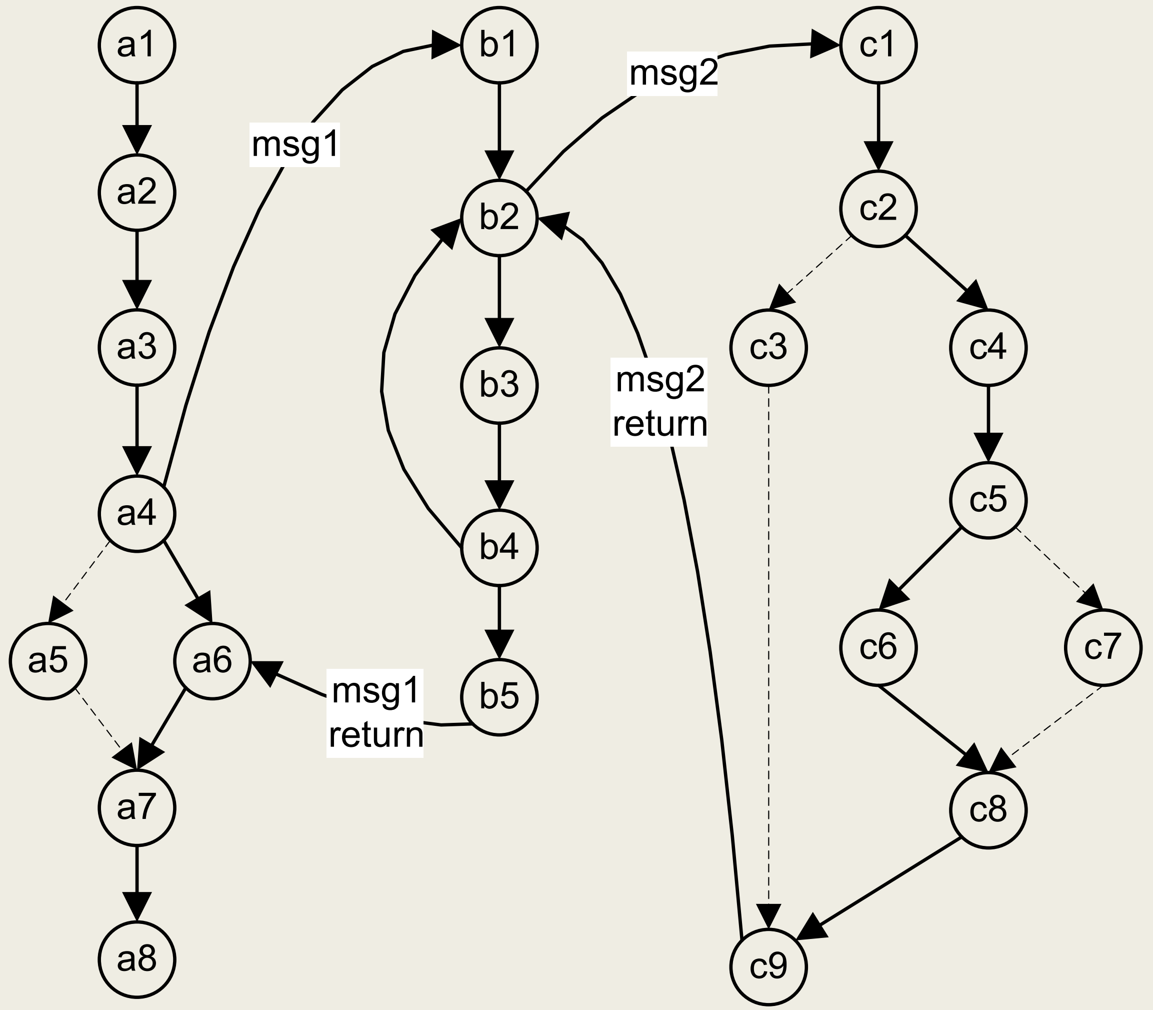


Call Graph-Based Integration (continued)

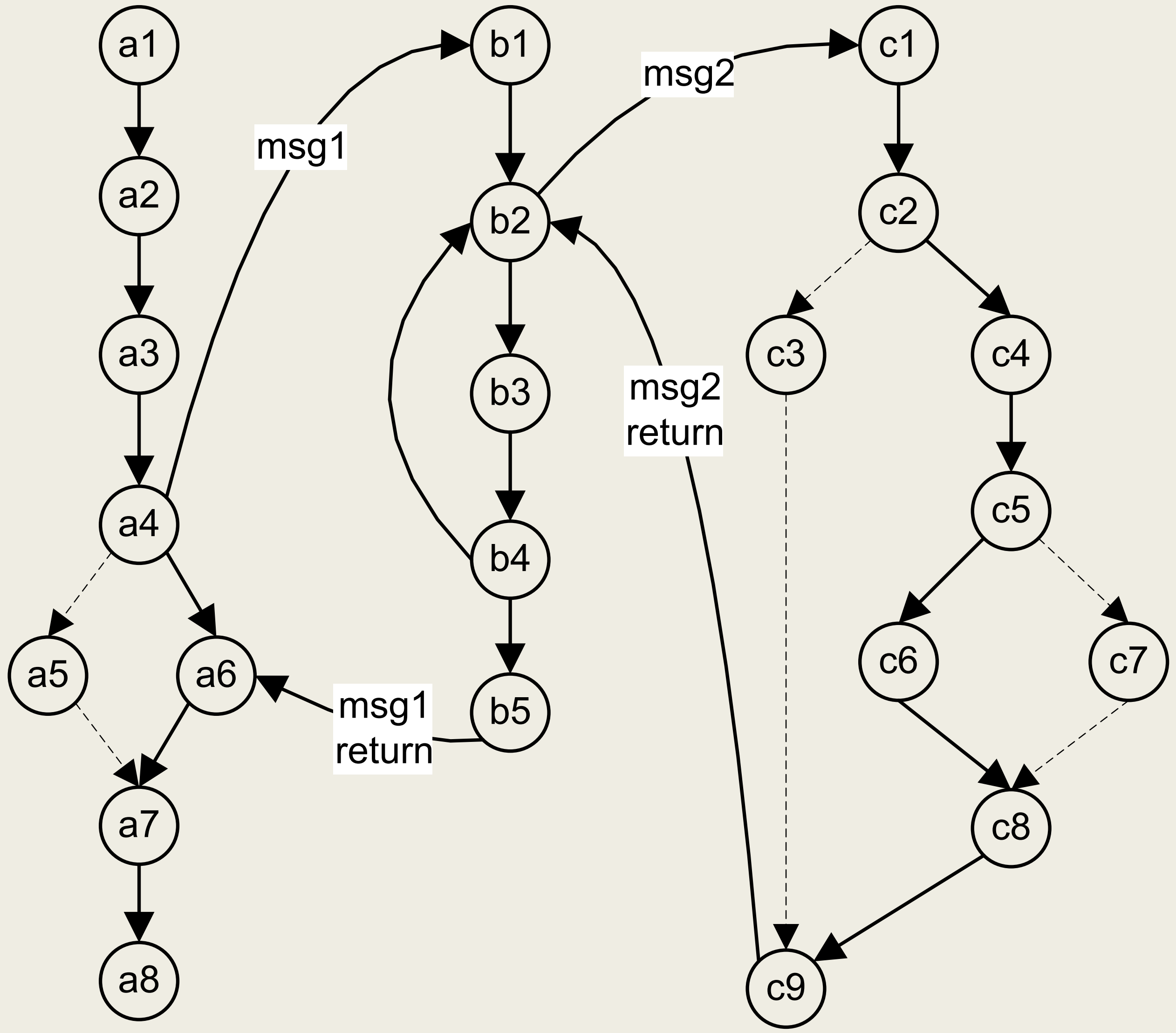
- Two strategies
  - *Pair-wise integration*
  - *Neighborhood integration*
- Degrees of nodes in the Call Graph indicate integration sessions
  - *isLeap and weekDay are each used by three units*
- Possible strategies
  - *test high indegree nodes first, or at least,*
  - *pay special attention to “popular” nodes*

MM-Path Definition and Example

- An MM-Path is an interleaved sequence of module execution paths and messages.
- An MM-Path across three units:



MM-Path Across Three Units



Exercise: List the source and sink nodes in the previous example.

Unit	Source Nodes	Sink Nodes
A		
B		
C		

Details of the Example MM-Path

The node sequence in the example MM-Path is:

<a1, a2, a3, a4>

    message msg1

<b1, b2>

    message msg2

<c1, c2, c4, c5, c6, c8, c9>

    msg2 return

<b3, b4, (b2, b3, b4)*, b5>

    msg1 return

<a6, a7, a8>

Note: the (b2, b3, b4)* is the Kleene Star notation for repeated traversal of the loop.

About MM-Paths

- Message quiescence: in a non-trivial MM-Path, there is always (at least one) a point at which no further messages are sent.
- In the example MM-Path, unit C is the point of message quiescence.
- In a data-driven program (such as NextDate), MM-Paths begin (and end) in the main program.
- In an event program (such as the Saturn Windshield Wiper), MM-Paths begin with the unit that senses an input event and end in the method that produces the corresponding output event.

Some Sequences...

- A DD-Path is a sequence of source statements.
- A unit (module) execution path is a sequence of DD-Paths.
- An MM-Path is a sequence of unit (module) execution paths.
- A (system level) thread is a sequence of MM-Paths.
- Taken together, these sequences are a strong “unifying factor” across levels of testing. They can lead to the possibility of trade-offs among levels of testing (not explored here).

Comparison of Integration Testing Strategies

Strategy Basis	Ability to test interfaces	Ability to test co-functionality	Fault isolation and resolution
Functional Decomposition	acceptable, but can be deceptive (phantom edges)	limited to pairs of units	good to a faulty unit
Call Graph	acceptable	limited to pairs of units	good to a faulty unit
MM-Path	excellent	complete	excellent to a faulty unit execution path

Summary

- Unit testing
- Test case design
  - *Specification-Based techniques: Black-box*
  - *Structure-Based techniques: White-box*
  - *Experience-Based techniques*
- Incremental testing: Integration testing
  - *Top-down vs. Bottom-up testing*
  - *MM-Path*