

Tim Pengajar IF2250

IF2250 – Rekayasa Perangkat Lunak

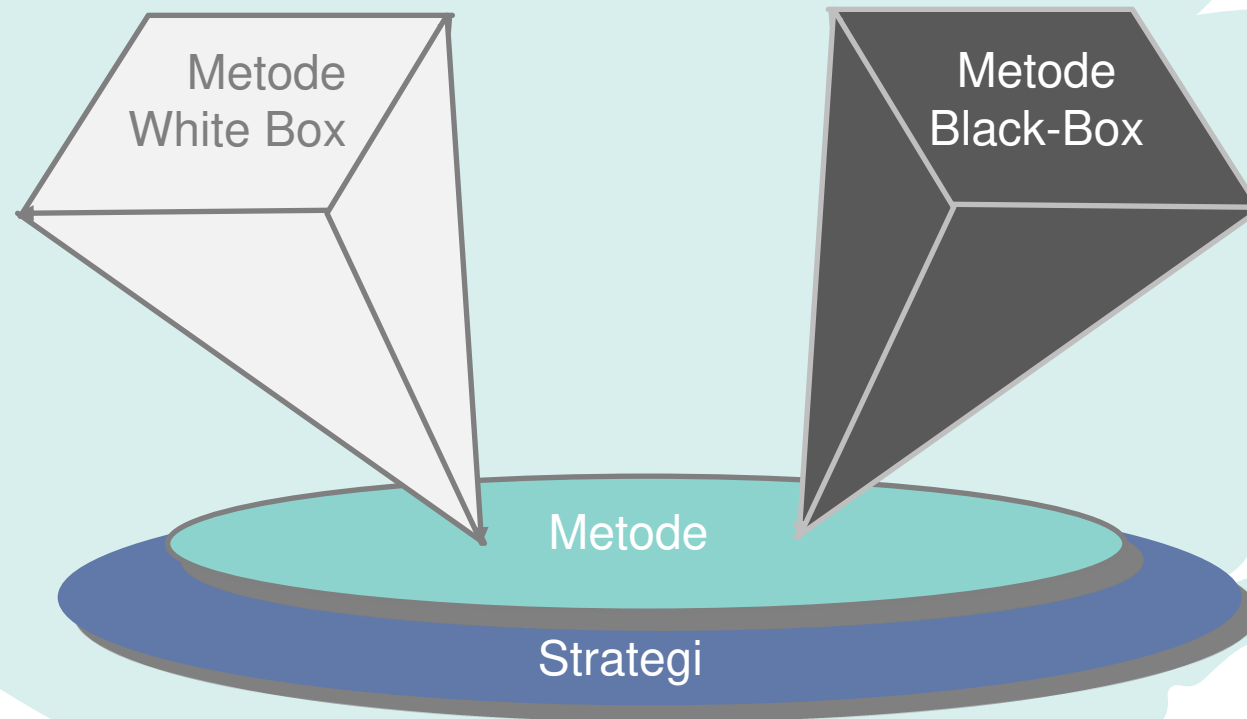
## Pengujian P/L (Bag 2)

SEMESTER II TAHUN AJARAN 2023/2024



KNOWLEDGE & SOFTWARE ENGINEERING

# ***Metode/Strategi Pengujian***



# ***White Box Testing***

Penguji bisa membaca kode program(bisa melihat bagian pengulangan, statement kondisi, pemanggilan prosedur/fungsi, dan lain-lain), sehingga kasus uji dibuat berdasarkan informasi dari kode program ini

## **Black Box Testing**

Penguji tidak membaca source-code, sehingga pengembangan kasus uji berdasarkan spesifikasi fungsional dari program. Sering disebut juga sebagai pengujian fungsional



# ***Pengujian Unit atau Pengujian Modul atau Pengujian Komponen***

(UNIT TESTING, MODULE  
TESTING OR COMPONENT  
TESTING)



# Proses Pengujian

```
1 #include <stdio.h>
2
3 int Kuadrat(int A)
4 {
5     int hasil;
6     hasil = 0;
7     for (int i = 0; i < A; i++)
8     {
9         hasil = hasil + A;
10    }
11    return hasil;
12 }
```

Unit Program  
yang akan  
diuji

## Pembuatan Kasus Uji

Masukan	Harapan Hasil
A=0	return 0
A=1	return 1
A=2	return 4
A=3	return 9
A=10	return 100

Perbaiki Kode

Hasil:

- Kasus uji no x tidak lolos; atau
- Semua kasus uji lolos;

Eksekusi Kasus Uji

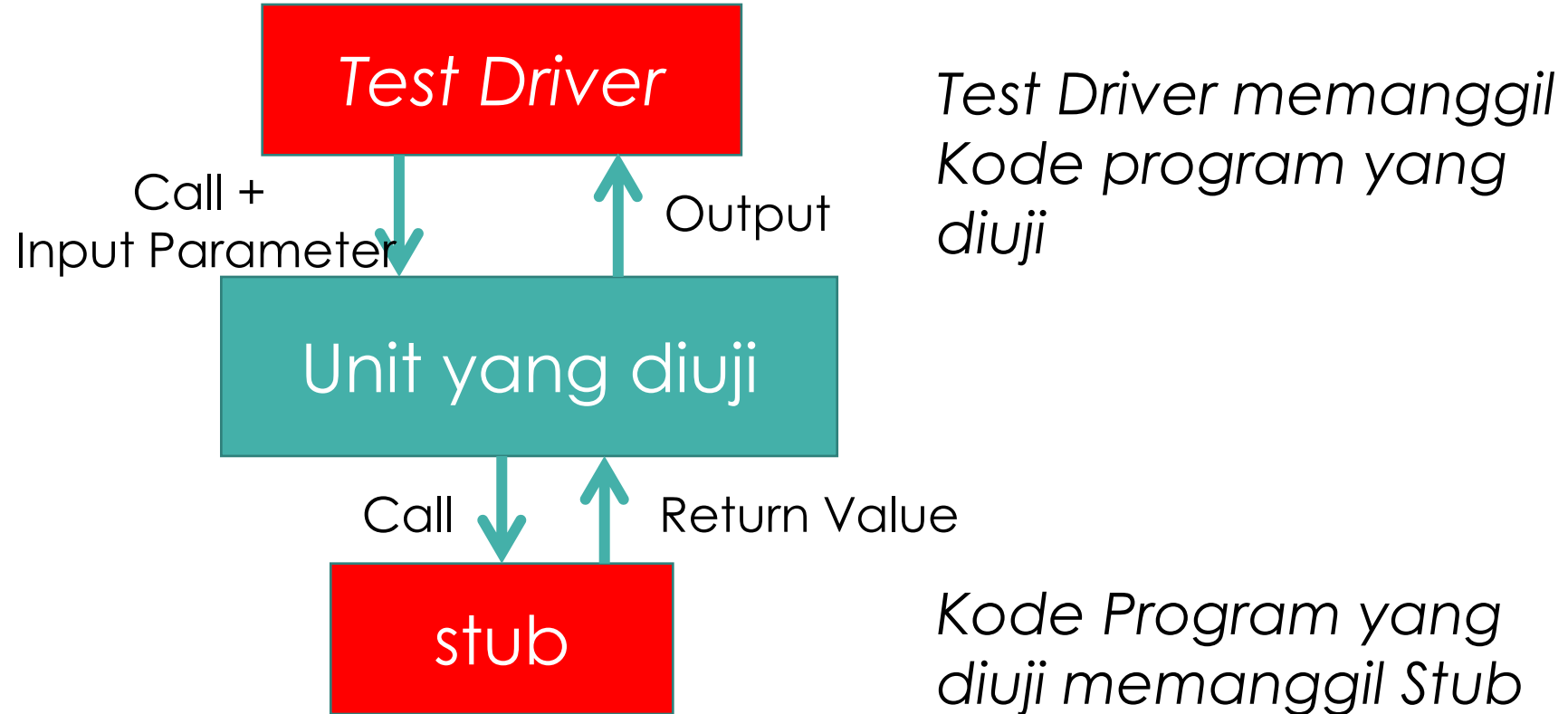
Selesai

# *Lingkungan Pengujian*

- Lingkungan untuk menguji harus disiapkan
  - Perlu computer, perlu compiler, perlu persiapan jaringan, dan lain-lain
- Untuk pengujian Unit, maka mungkin saja ada bagian program lain yang belum siap, tetapi pengujian unit tetap harus dilakukan
  - Sehingga perlu disiapkan **TEST DRIVER** dan **STUB**
- *Test driver* adalah program yang memanggil unit yang sedang diuji, sekaligus memberikan input data, dan memeriksa hasilnya
- Bila unit yang akan diuji, perlu memanggil unit lain yang belum selesai di program, maka dipersiapkan STUB untuk meng-emulasi peran unit lain tadi.
  - Bagian ini disebut juga sebagai program dummy
- Test driver dan stub secara bersama-sama disebut **scaffolding**

# *Lingkungan Pengujian Unit*

7



## ***Unit Under Test***

```
int HitungGajiPegawai(int NIP)
{
    int GaPok = ReadGajiPokok(NIP);
    int GaTun = ReadGajiTun( NIP);

    return GaPok + GaTun;
}
```

## **STUB**

```
int ReadGajiPokok(int NIP)
{
    // execute sql statement
    // select gaji from Pegawai
    return 2500;
}
```

## **Test Driver**

```
int main()
{
    assert(HitungGajiPegawai(111), 10000);
    assert(HitungGajiPegawai(200), 5000);
}
```

```
int ReadGajiTun(int NIP)
{
    execute sql statement
    select tunj from Pegawai
    return 75000;
}
```





## ***Unit Under Test***

```
int HitungGajiPegawai(int NIP)
{
    int GaPok = ReadGajiPokok(NIP);
    int GaTun = ReadGajiTun( NIP);

    return GaPok + GaTun;
}
```

- Procedure HitungGajiPegawai yang ditulis dalam bahasa C ini adalah bagian yang diberikan oleh user untuk diuji oleh penguji
- Dalam pengembangan software skala menengah, pengujian unit ini dapat dilakukan oleh pengembang sendiri. Hal ini dilakukan untuk meminimalisasi biaya. Tentunya, pengembang biasanya menggunakan alat bantu seperti JUnit (java), NUnit (.NET), atau PUnit (PHP).



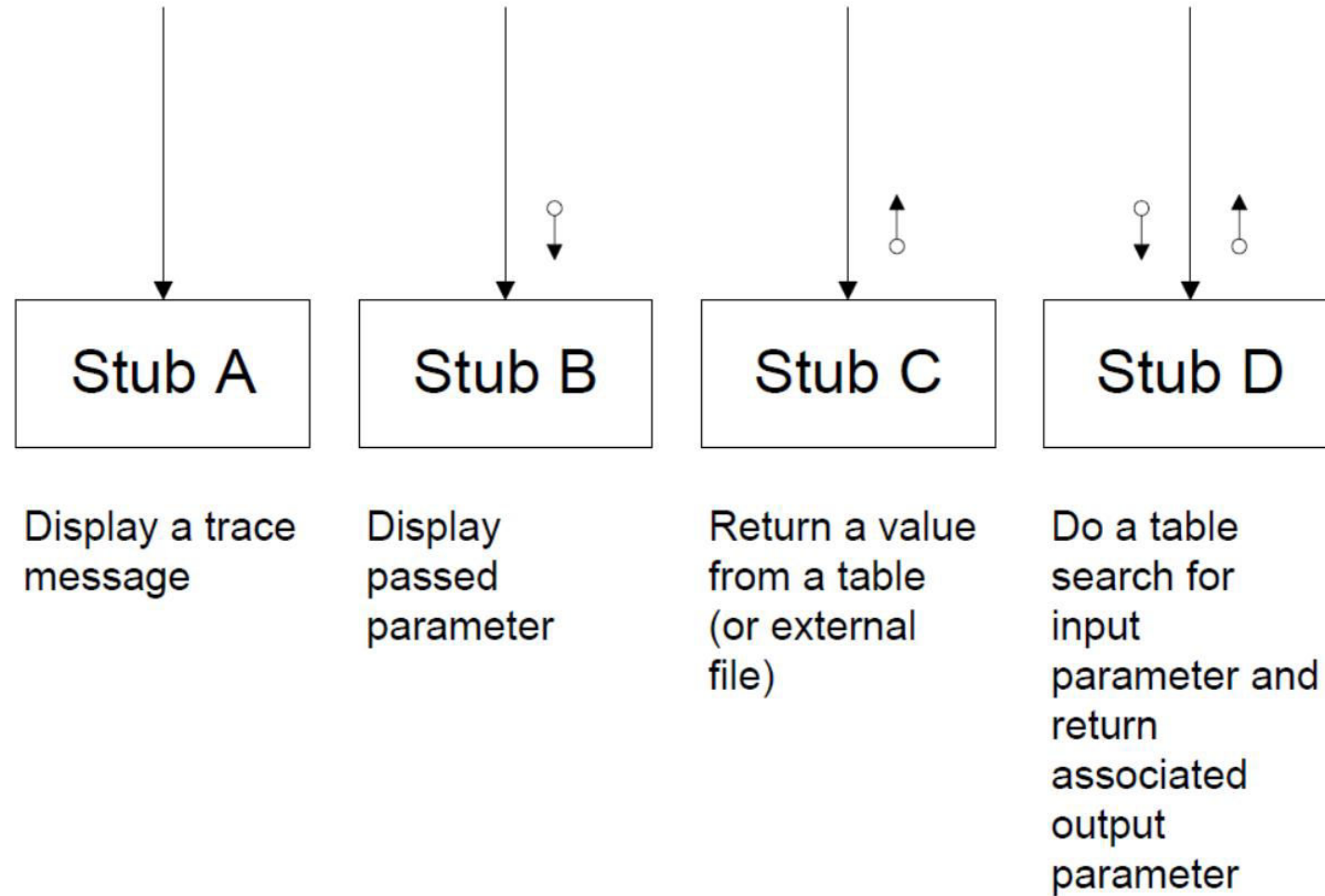
- Stub dibuat oleh penguji
  - Jika pengujian unit dilakukan oleh programmer, maka stub dibuat oleh programmer
- Stub dibuat karena bagian program itu belum siap atau belum selesai dikerjakan
  - Biasanya Stub dibuat untuk bagian yang ber-interface ke sistem lain.
    - Pada contoh ini, kedua stub seharusnya memanggil suatu perintah SQL ke basisdata, tetapi pada saat itu, pengembangan basisdata masih belum siap/selesai.
- Stub dibuat agar memungkinkan program yang diuji dapat di-compile dan melakukan fungsi yang diharapkan
  - Sehingga menghindari ketergantungan pada modul/unit lain yang belum selesai

## STUB

```
int ReadGajiPokok(int NIP)
{
    // execute sql statement
    // select gaji from Pegawai
    return 2500;
}
```

```
int ReadGajiTun(int NIP)
{
    // execute sql statement
    // select tunj from Pegawai
    // return 75000;
}
```

# *Jenis STUB*

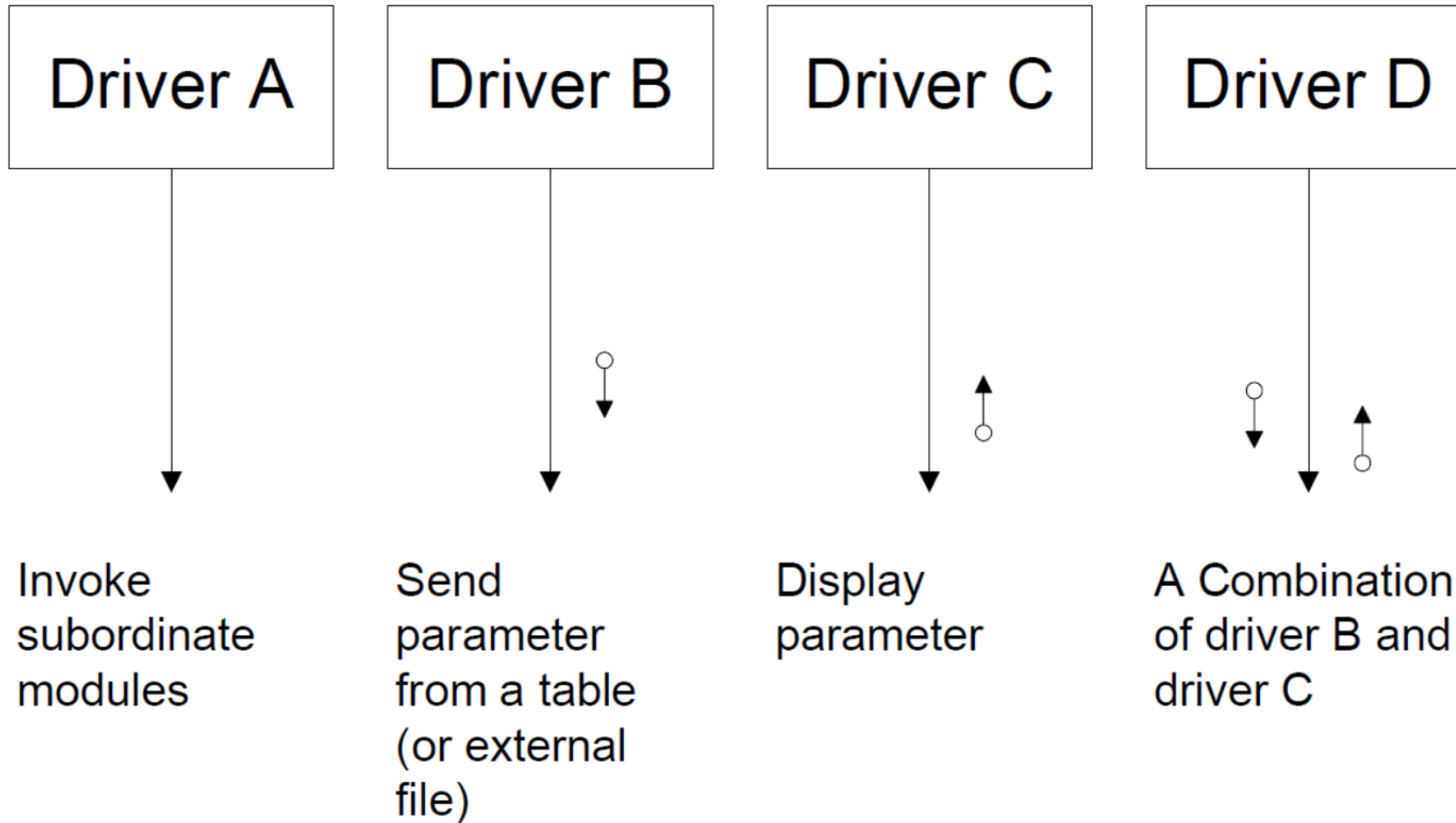


- Test driver dikembangkan oleh penguji
  - Dan kalau program cukup kecil, maka programmer bertanggung jawab untuk mengembangkan test driver ini
- Test driver akan memanggil unit yang akan diuji
  - Test driver ini biasanya berisi beberapa kasus uji yang terkait dengan modul yang akan diuji
  - Test driver juga bisa berisi urutan skenario yang menggunakan unit yang diuji
- Test driver juga perlu memperhatikan penggunaan atau penyiapan stub
  - test driver menyesuaikan pemakaian stub sesuai kebutuhan, atau
  - Stub menyesuaikan dengan kebutuhan test driver
- Pada contoh dibawah ini, test driver menggunakan perintah 'assert' untuk melihat apakah fungsi HitungGajiPegawai mengembalikan suatu nilai
  - Dengan parameter 111, diharapkan mengembalikan nilai 10000

## Test Driver

```
int main()
{
    assert(HitungGajiPegawai(111), 10000);
    assert(HitungGajiPegawai(200), 5000);
}
```

# *Jenis Driver*



# ***WHITE BOX TESTING***

CONTOH:

- CONTROL FLOW TESTING
- DATA FLOW TESTING
- MUTATION TESTING
- DAN LAIN-LAIN



# ***Control Flow Testing***

- Langkah
  - Kode Program diubah menjadi Control Flow Graph
  - Dilihat jalur eksekusinya
  - Setiap jalur eksekusi harus dilewati oleh kasus uji

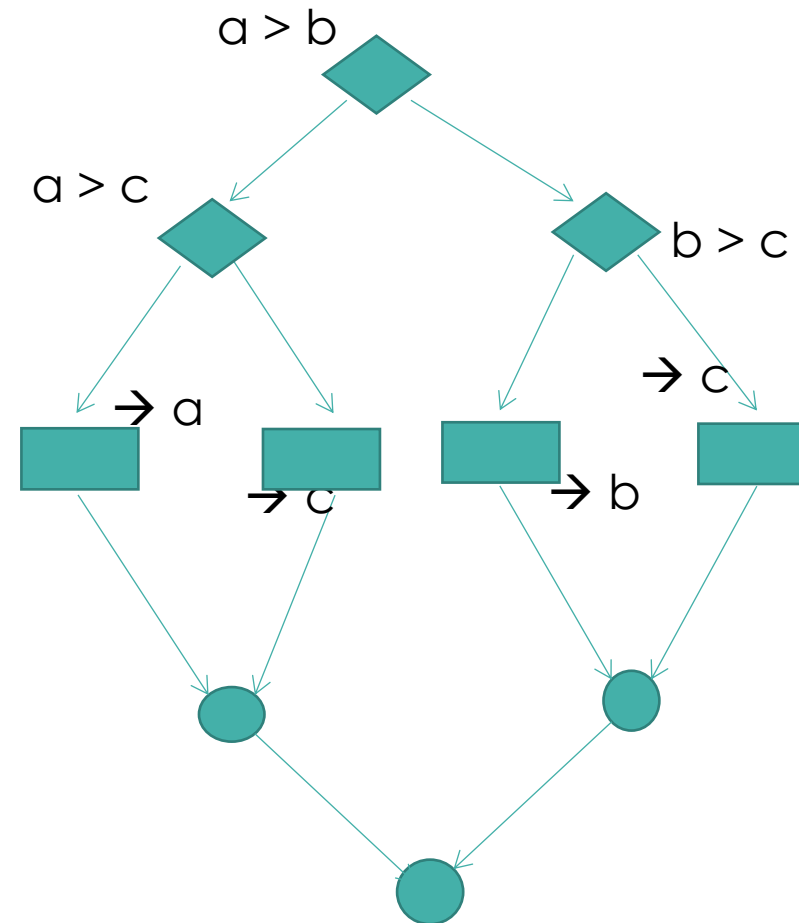
# *Contoh Mencari Bilangan Terbesar*

```
1 Function Largest( a, b, c : integer) : integer
2 Kamus:-
3 Algoritma
4
5     if a > b then
6         if a > c then
7             return a
8         else
9             return c
10    else
11        if b > c then
12            return b
13        else
14            return c
```



# CFG

```
if a > b then
  if a > c then
    return a
  else
    return c
else
  if b > c then
    return b
  else
    return c
```

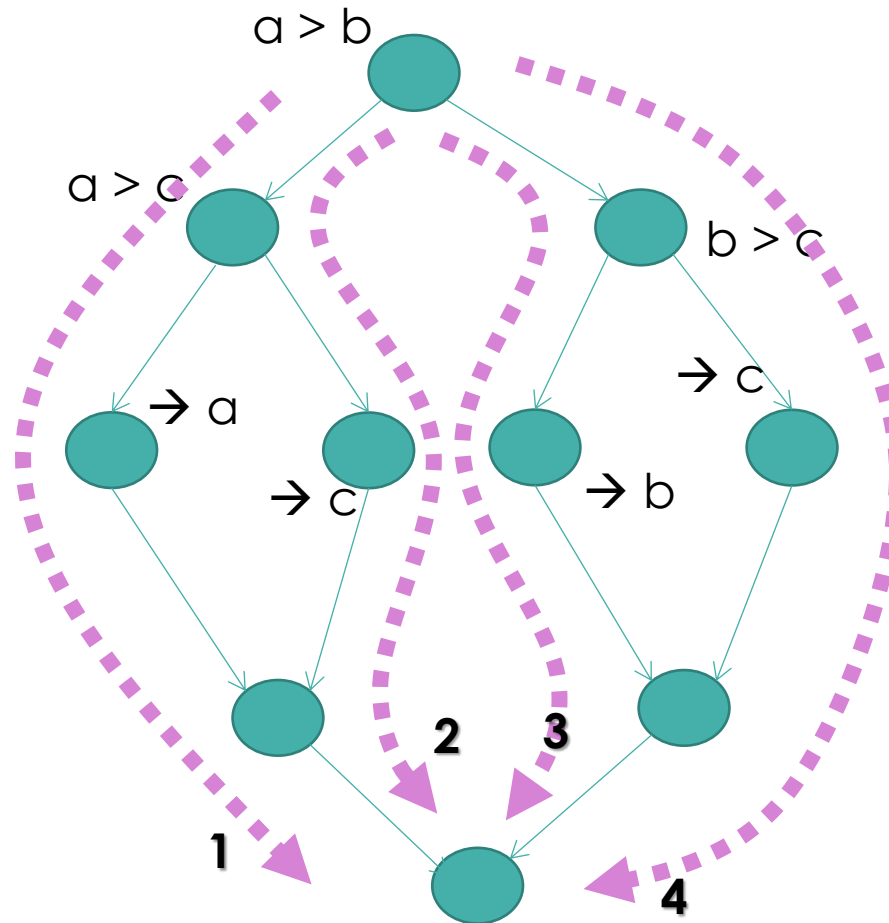


# *CFG dapat disederhanakan menjadi kumpulan nodes saja*

```

if a > b then
  if a > c then
    return a
  else
    return c
else
  if b > c then
    return b
  else
    return c

```



Ada berapa jalur dasar?

***Cari semua masukan yang memungkinkan semua jalur di lewati***

Jalur	Input (a, b, c)	Hasil yang diharapkan
Jalur 1	(10, 3, 2)	10
Jalur 2	(25, 15, 40)	25
Jalur 3	(10, 15, 6)	15
Jalur 4	(10, 15, 19)	19

# ***Buat CFG!***

## ***Dari C code berikut***

```
#include <stdio.h>

int main()
{
    int x;
    a = 5;
    x = 0;
    while (x < 10)
    {
        printf( "%d %d\n", x, a );
        x = x + 1;
    }
}
```

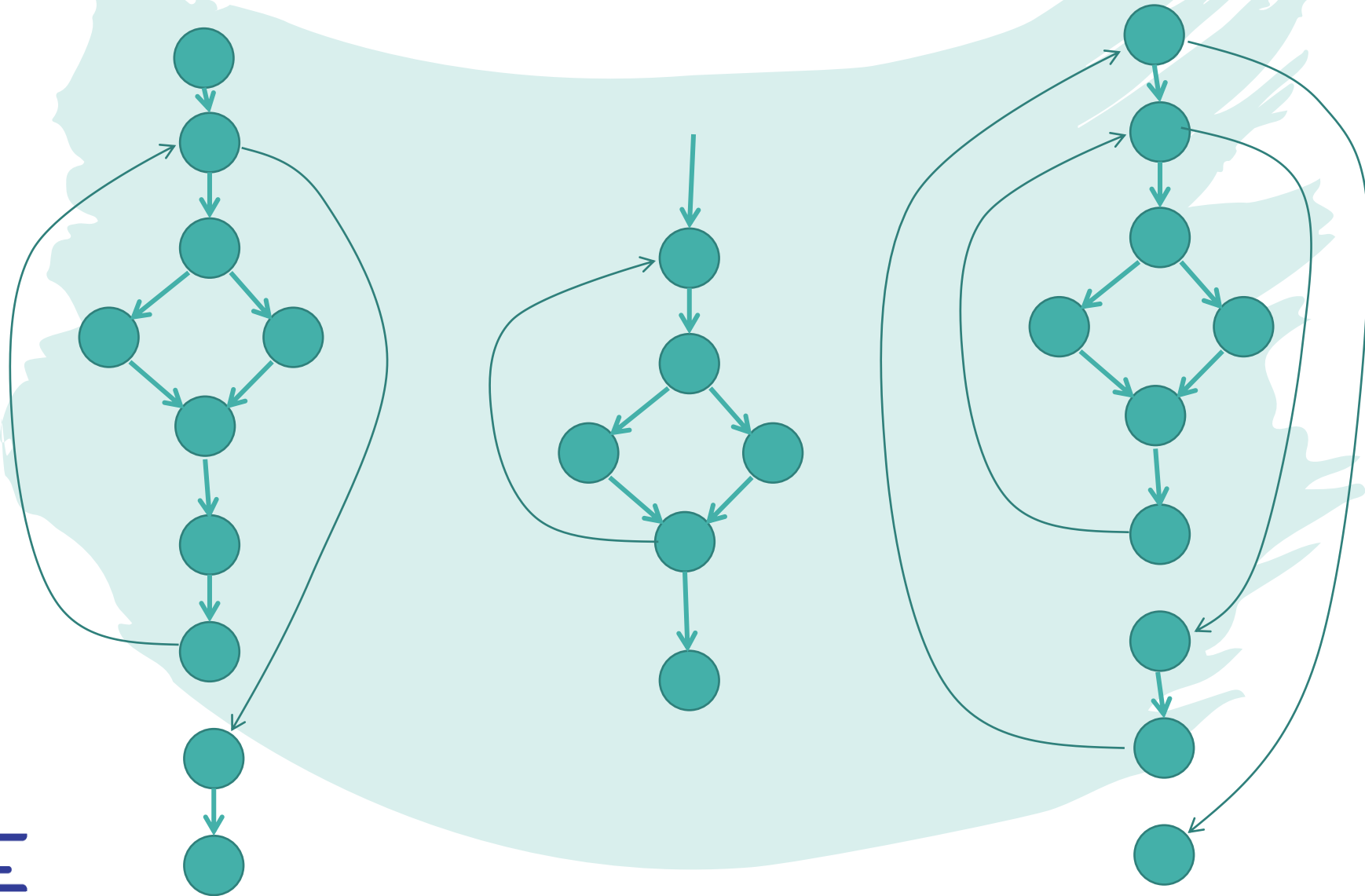
```
#include <stdio.h>

int main()
{
    int x;
    X = 0;
    do {
        if (x < 1)
            printf(“%d”, x);
        else
            printf(“%d”, x * 2);
        x = x + 1;
    } while x < 10;
}
```

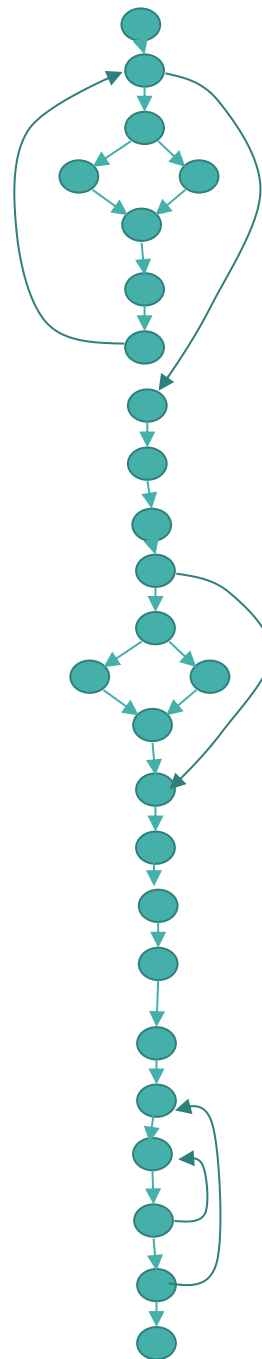
```
#include <stdio.h>

int main()
{
    int x;
    for ( x = 0; x < 10; x++ )
    {
        for (y = 10; y > 1; y--)
            printf(“%d, %d”, x, y );
    }
}
```

# *Bagaimana bentuk program-program ini?*



*...atau yang ini?*



# ***BLACK BOX TESTING***

**EQUIVALENCE CLASS PARTITIONING**

**BOUNDARY VALUE ANALYSIS**

GRAPH BASED METHOD

COMPARISON TESTING

ORTHOGONAL ARRAY TESTING

MODEL BASED TESTING

DAN LAIN-LAIN



# ***Permasalahan Black-Box Testing***

- Mencari masukan apa yang bagus untuk kasus uji
- Apakah suatu masukan data akan memungkinkan unit program yang diuji menjadi salah?
- Berapa jumlah masukan data yang perlu dicoba?
  - Mungkin perlu banyak, tetapi seberapa banyak?
- Ada nilai batas?
  - Apakah nilai-nilai batas sudah diperiksa pada kode program



# ***Beberapa Teknik Black Box***

- **Equivalence Class Partitioning**
- **Boundary Value Analysis**
- Graph Based method
- Comparison Testing
- Orthogonal Array Testing
- Model Based Testing
- Dan lain-lain

# ***Equivalence Class Partitioning dan Boundary Value Analysis (ECP/BVA)***



# Contoh

- Sebuah program digunakan untuk Pemilu. Program akan menerima data masukan umur. Umur yang dibolehkan adalah 17 tahun ke atas agar bisa memilih.
  - Ada dua kelas ekivalen (ada dua partisi)

0	16	17	Tak hingga?
---	----	----	-------------

- Jadi akan hanya diperlukan dua kasus uji

Nomor Test	Data	Hasil Yang Diharapkan
1	12	You cannot vote
2	21	You can vote

- Kedua test itu tentunya belum cukup, karena kita tahu bahwa test tersebut belum mencakup perbedaan batas yang penting yaitu batas umur 17 tahun. Jadi perlu diperiksa yang yang berumur 16, 17 dan 18.
  - Kenapa? Karena pemrogram bisa saja melakukan kesalahan memasukkan kondisi umur tersebut.
    - If (umur >17) atau if (umur >= 17) atau if (umur <= 16) ?
- Jadi kita buat dua test lagi menggunakan ide Boundary Value Analysis.

Nomor Test	Data	Hasil Yang Diharapkan
3	16	You cannot vote
4	17	You can vote

- Kenapa 18 tidak perlu diperiksa lagi?
- Apakah perlu memeriksa umur 0 dan umur 100 misalnya?

## *Contoh lain*

- Test program yang menampilkan bilangan terbesar antara dua input. Kedua input selalu dalam range 0-10000. Jika nilainya sama, maka akan ditampilkan nilainya yang sama.
  - Maka kita bisa mengambil nilai sembarang antara 0-10000 untuk keduanya:
    - Misalnya untuk input 1 diambil 500 dan kedua 1000
    - Kemudian diambil nilai batas, 0 dan 10000
    - Catatan: untuk kasus ini, jika range sudah ditentukan 0 sampai 10000, artinya masukan tidak mungkin kurang dari 0, atau lebih dari 10000

# *Kasus Uji*

Nomor Test	Angka 1	Angka 2	Hasil yang diharapkan
1	0	0	0
2	0	1000	1000
3	0	10000	10000
4	500	0	500
5	500	1000	1000
6	500	10000	10000
7	10000	0	10000
8	10000	1000	10000
9	10000	10000	10000

# *Integration Testing*



# *Pengujian Integrasi (Integration Testing)*

- Pengujian unit hanya fokus pada komponen/unit individual
  - Sesudah semua fault/bug/defect dihilangkan dari setiap unit, maka unit/komponen ini siap diintegrasikan atau digabungkan
- Proses integrasi unit-unit tadi akan memungkinkan terjadinya error yang tidak terdeteksi sebelumnya.
  - Pengujian integrasi dapat menemukan kesalahan atau bahkan dapat mendeteksi fault/bug/defect yang belum terdeteksi saat pengujian unit
  - Pengujian integrasi fokusnya lebih pada sekumpulan komponen/unit
- Idealnya, proses integrasi atau penggabungan, dilakukan satu persatu, setiap kali penggabungan satu unit maka akan dilakukan pengujian.
  - Jika kesalahan tidak ditemukan, maka unit lain akan digabungkan, hingga seluruh unit terintegrasi
  - Strategi penggabungan/integrasi akan menentukan lama tidaknya pengujian integrasi dilakukan
    - Strategi yang salah akan menambah biaya dan waktu



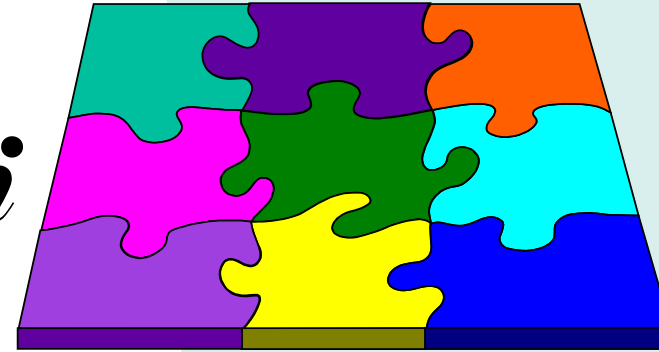


# *Strategi Pengujian Integrasi*

Pendekatan incremental

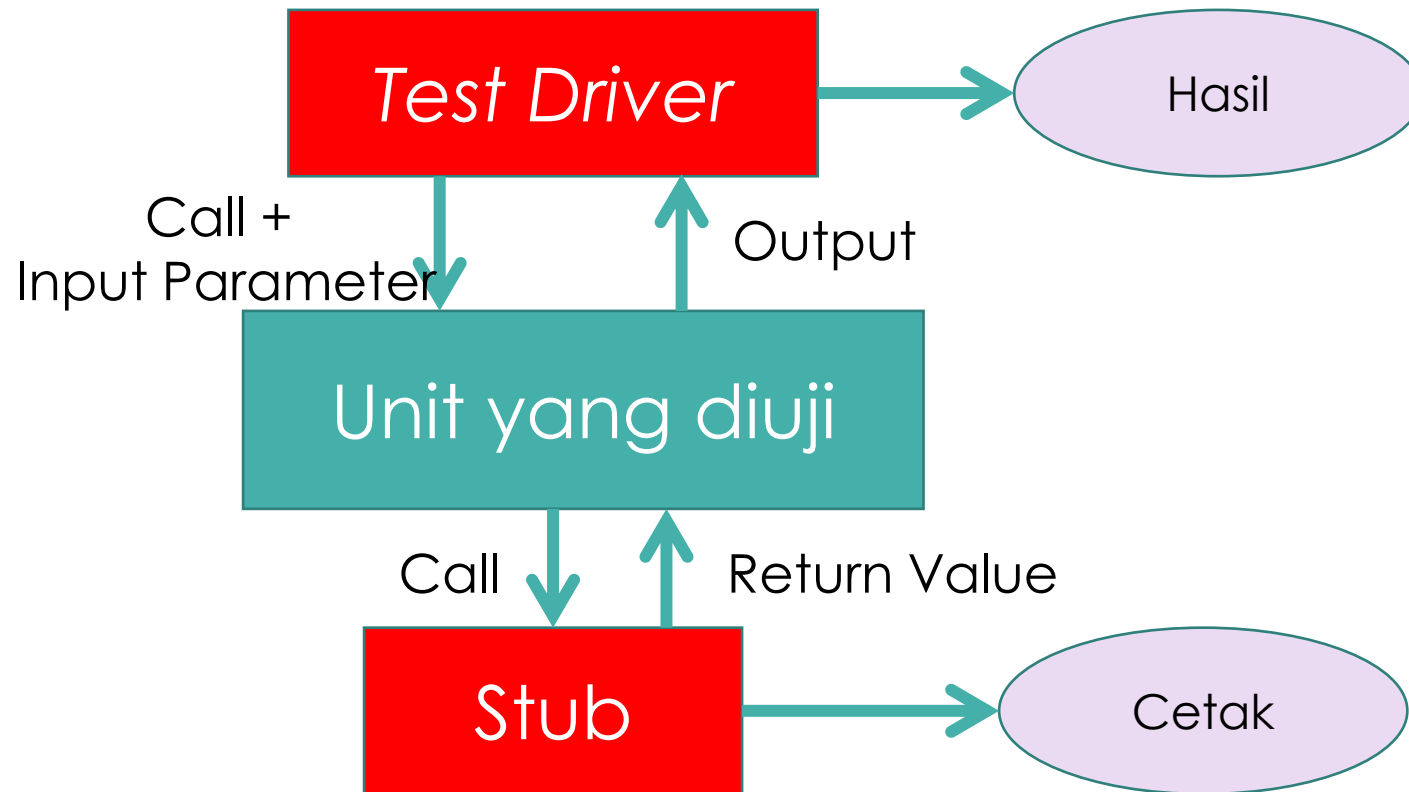
- Top Down testing
- Bottom Up testing
- Sandwich Testing

Pendekatan 'BIG BANG'



# ***Review: Lingkungan Pengujian***

34



## ***Unit Under Test***

```
int HitungGajiPegawai(int NIP)
{
    int GaPok = ReadGajiPokok(NIP);
    int GaTun = ReadGajiTun( NIP);

    return GaPok + GaTun;
}
```

## **STUB**

```
int ReadGajiPokok(int NIP)
{
    // execute sql statement
    // select gaji from Pegawai
    return 2500;
}
```

```
int ReadGajiTun(int NIP)
{
```

```
    // execute sql statement
    // select tunj from Pegawai
    return 7500;
}
```

## **Test Driver**

```
int main()
{
    assert(HitungGajiPegawai(111), 10000);
    assertNotTrue(HitungGajiPegawai(200), 5000);
}
```



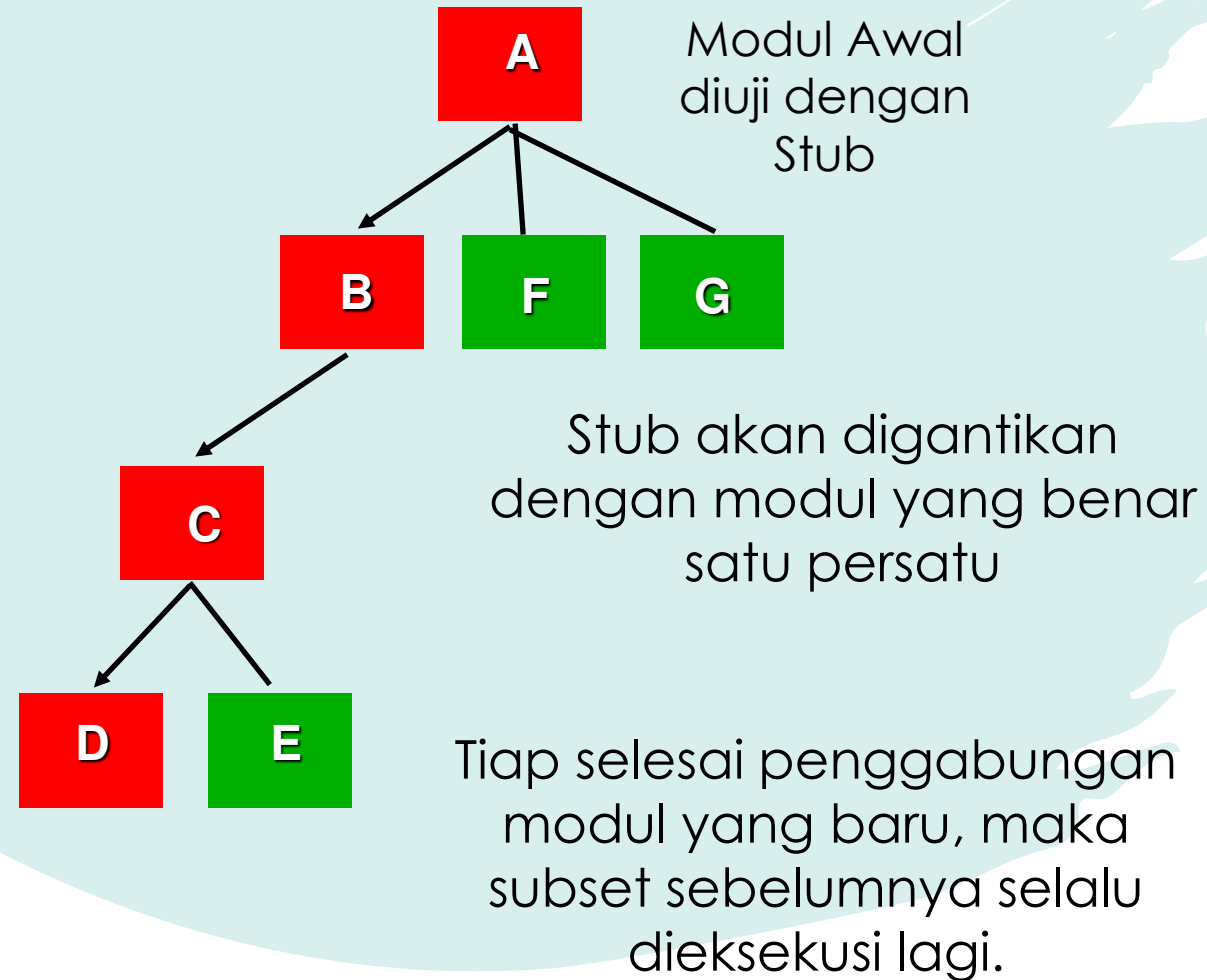
# ***Strategi Top Down Testing***

- Pengujian dimulai dari komponen level atas, biasanya bagian User Interface (UI), lalu diikuti komponen-komponen di bawahnya, hingga semua sudah diuji.
- Butuh Stub untuk komponen di bawahnya, test driver tidak diperlukan



**Keuntungan:** Biasanya dimulai dari UI, (atau menu) atau dari program utama sehingga mudah (tidak butuh test driver)

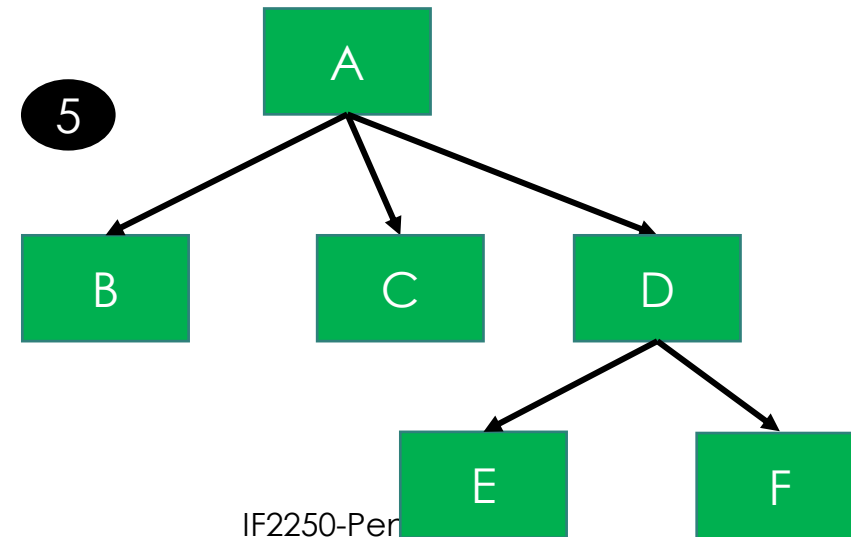
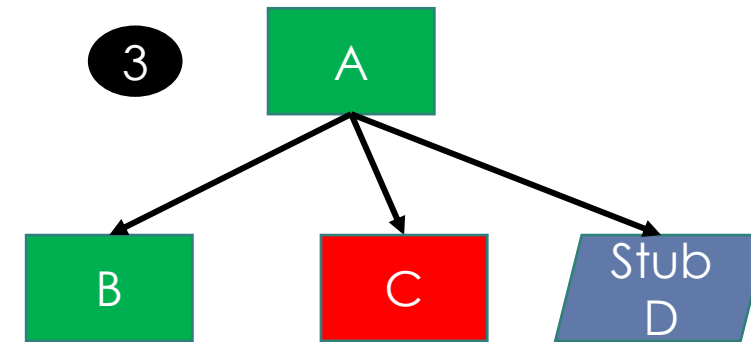
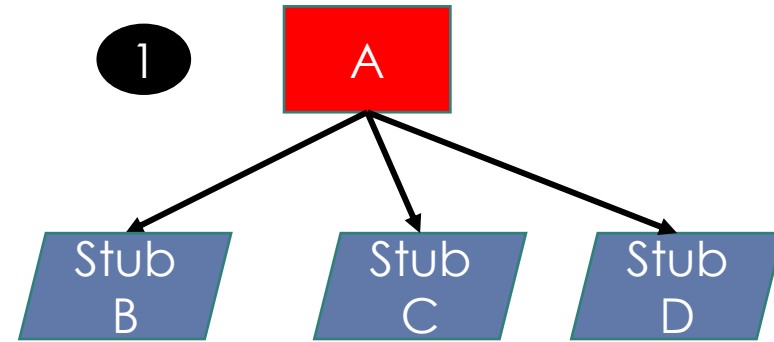
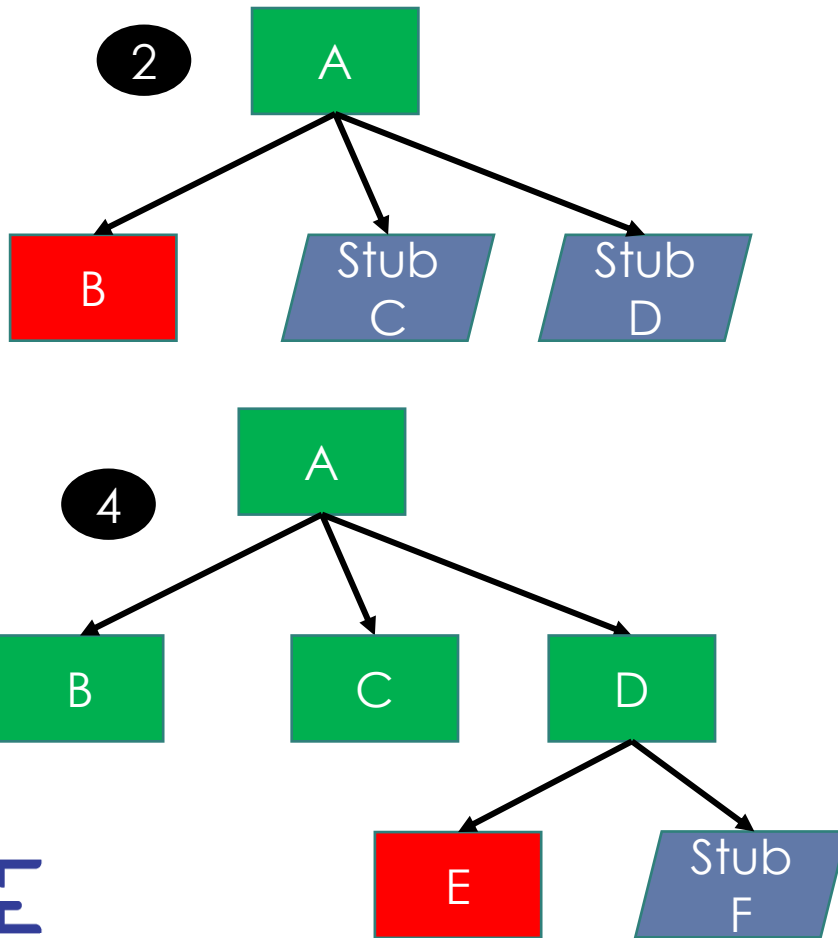
**Kerugian:** kadang banyak stub dibutuhkan, padahal pengembangan stub butuh waktu dan stub sering ada error

# *Strategi Top Down Testing (2)*



# Top Down Testing

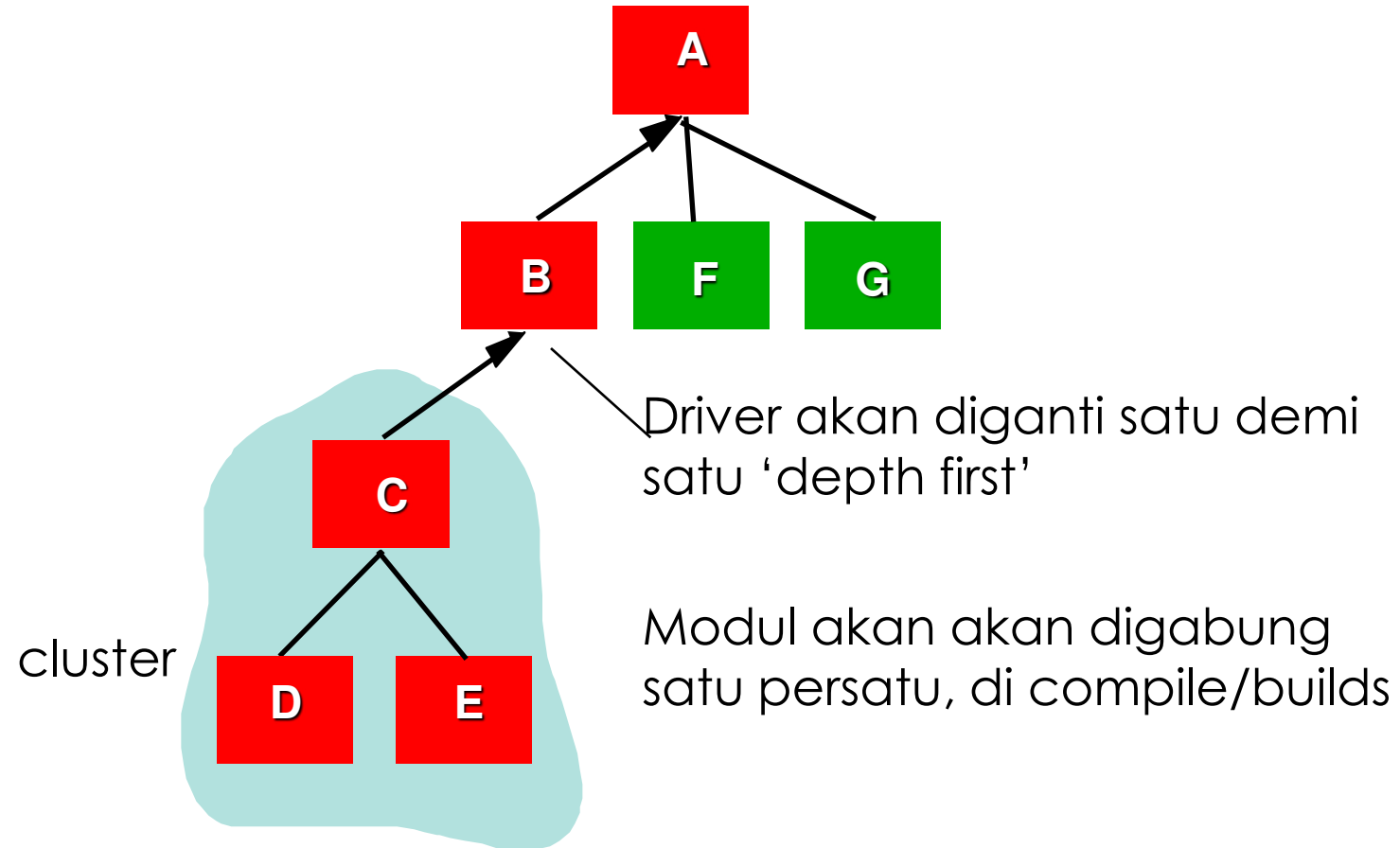
 Modul yang baru digabung, perlu diuji  
 Modul yang sudah lolos uji



# ***Strategi Bottom-Up Testing***

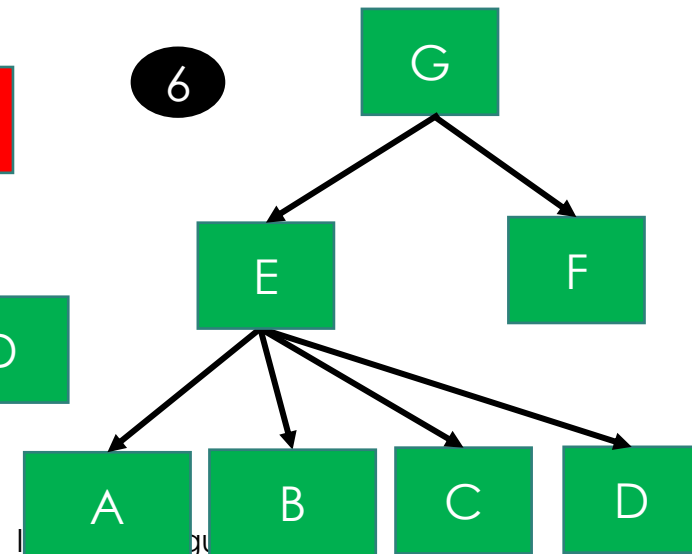
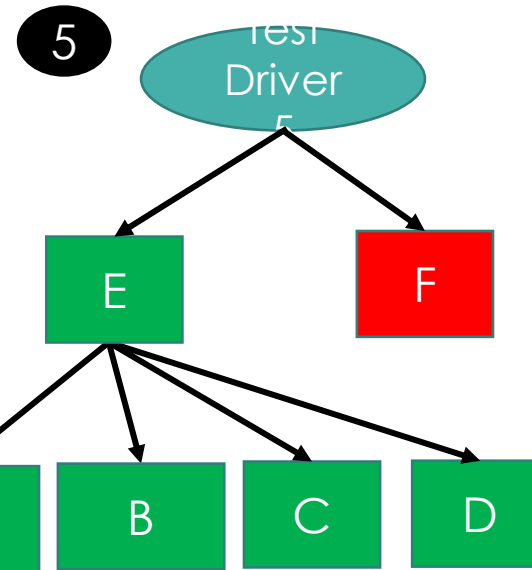
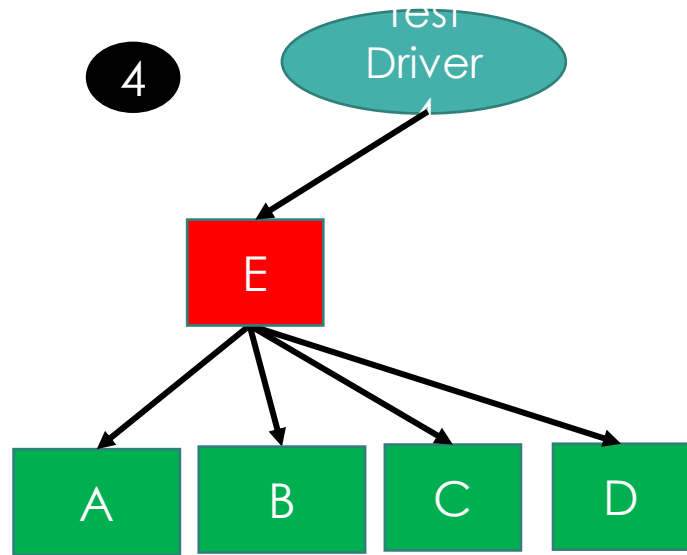
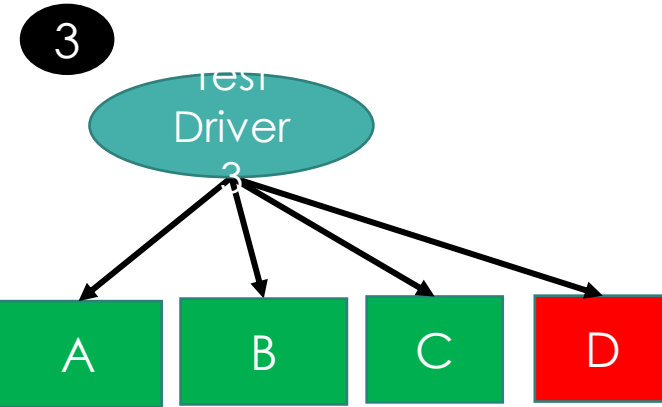
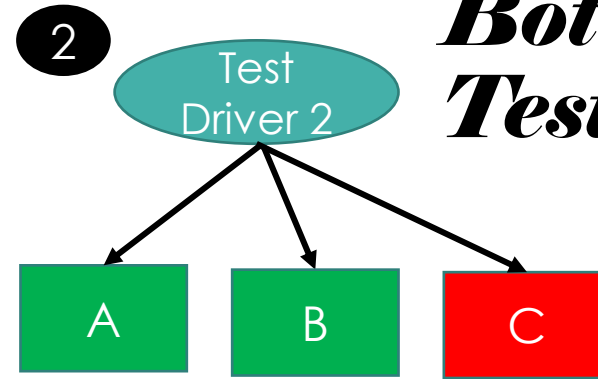
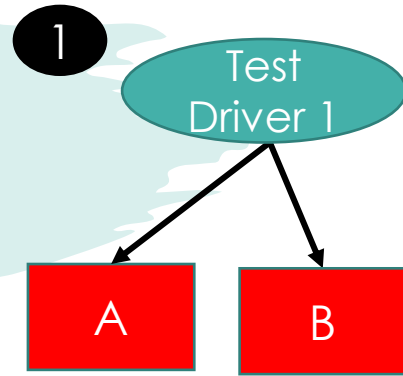
- Setelah tiap komponen/unit di lapisan bawah di lakukan unit-test, maka komponen/unit ini diintegrasikan dengan komponen di lapisan atasnya.
  - Diulang hingga semua level di atasnya terintegrasi
  - Butuh test driver untuk men-simulasikan komponen/unit di level atasnya
    - Test Stub tidak diperlukan

# *Strategi Bottom-Up Testing (2)*





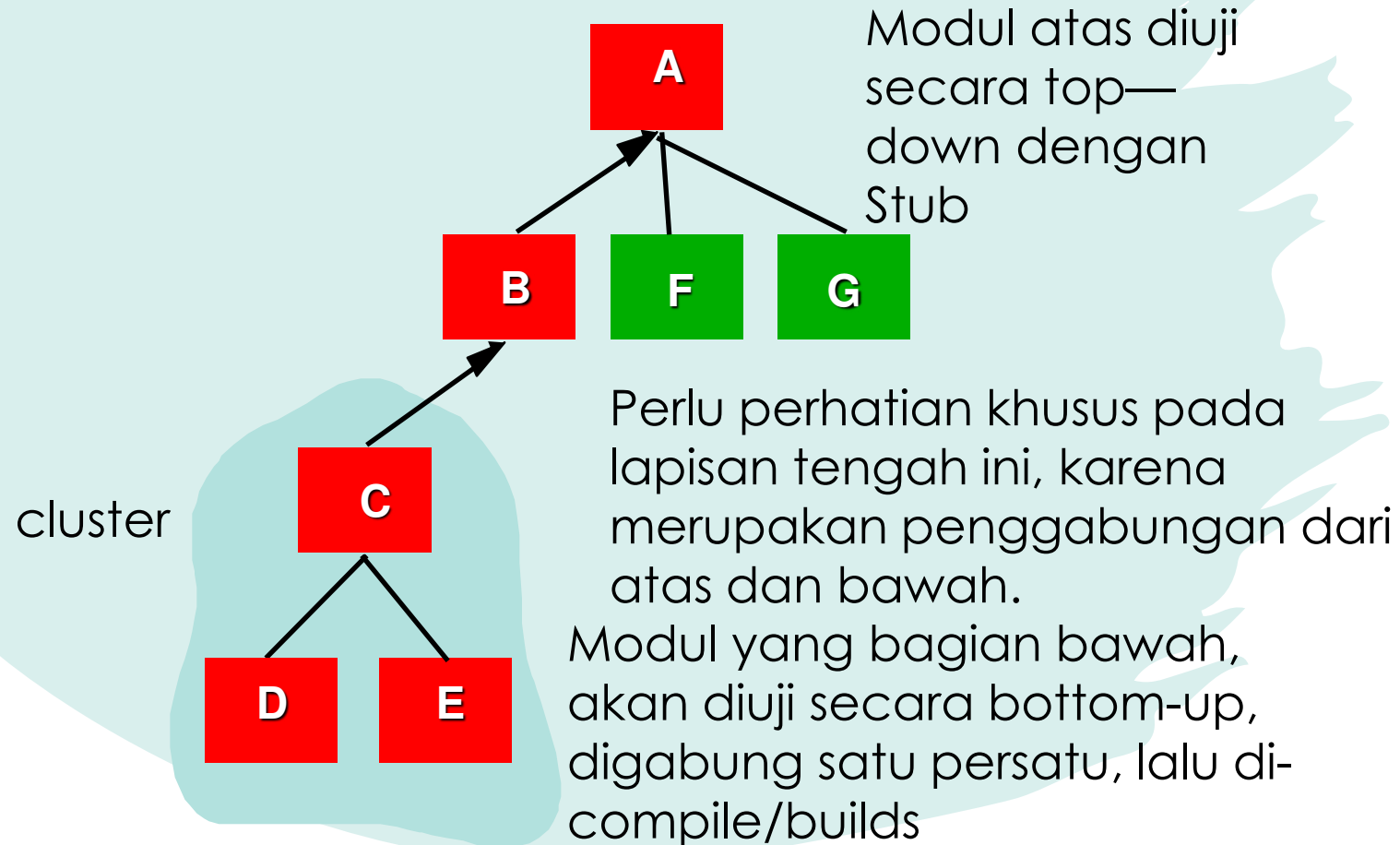
# Bottom Up Testing



# ***Strategi Pengujian Sandwich (Sandwich Testing)***

- Penggabungan strategy Top-down dan Bottom-Up
  - Sistem seolah dibagi menjadi tiga lapisan
    - Lapisan atas
    - Lapisan bawah
    - Lapisan target (yang terletak antara lapisan atas dan bawah)
  - Dengan fokus pada lapisan target, maka top-down/bottom-up testing dilakukan secara bersamaan (paralel)

# ***Pengujian 'Sandwich'*** ***(Sandwich Testing)***



# ***Pengujian Regresi (Regression Testing)***

- Pengujian Regresi melakukan re-eksekusi terhadap beberapa subset test yang sudah dilakukan sebelumnya untuk meyakinkan bahwa perubahan yang baru tidak berakibat pada hasil pengujian sebelumnya
- Jika software diperbaiki, beberapa aspek dalam konfigurasi juga perlu diperbaiki (dokumentasi, program, ataupun data)
- Pengujian regresi akan meyakinkan bahwa perubahan tidak memberikan perilaku yang tidak diinginkan atau tidak mempengaruhi modul-modul sebelumnya.
- Pengujian regresi dapat dilakukan secara manual dengan mengeksekusi ulang semua kasus uji tetapi menggunakan tools akan memungkinkan otomatisasi sehingga akan lebih cepat dan mudah
  - Misalnya penggunaan JUnit (untuk java), NUnit (.NET) atau PUnit (Php).
- Pengujian Regresi juga perlu dilakukan untuk Unit Testing



# ***Big Bang Testing***

- Pada Big Bang testing, pengujian dengan langsung menggabung semua unit/komponen tadi sebagai sistem yang utuh
  - **Keuntungan:** tidak perlu stub atau driver
  - **Kerugian:**
    - Bila terjadi suatu error mungkin sulit dicari bug-nya
      - Sulit diketahui apakah faultnya terletak di interface atau terletak di bagian dalam unit/komponenennya.
- Big Bang Testing bukanlah cara yang ideal, nampaknya cepat, tetapi untuk software skala menengah hingga besar, cara ini tidak di rekomendasikan.

# *Pengujian System*

- Validation testing
  - Fokus pada kebutuhan perangkat lunak (software requirements)
- System testing
  - Fokus pada integrasi sistem
- Alpha/Beta testing
  - Fokus pada pengguna (bagaimana mereka menggunakan)
- Recovery testing
  - Software dibuat 'gagal' dan melihat apakah proses recovery sudah dilakukan.
    - Contoh: software yang memanfaatkan internet dilihat perilaku recovery-nya bila internet tiba-tiba putus
- Security testing
  - Memverifikasi apakah mekanisme proteksi sudah melindungi suatu 'penetrasi'
- Stress testing
  - Sistem diuji bagaimana jika menghadapi permintaan pemakaian sumberdaya yang melebihi jumlah normal, atau frekuensi atau volume normal
- Performance Testing
  - Menguji performansi software saat dieksekusi pada suatu konteks tertentu.

# ***Pengujian Regresi (Regression Testing)***

- Pengujian Regresi melakukan re-eksekusi terhadap beberapa subset test yang sudah dilakukan sebelumnya untuk meyakinkan bahwa perubahan yang baru tidak berakibat pada hasil pengujian sebelumnya
- Jika software diperbaiki, beberapa aspek dalam konfigurasi juga perlu diperbaiki (dokumentasi, program, ataupun data)
- Pengujian regresi akan meyakinkan bahwa perubahan tidak memberikan perilaku yang tidak diinginkan atau tidak mempengaruhi modul-modul sebelumnya.
- Pengujian regresi dapat dilakukan secara manual dengan mengeksekusi ulang semua kasus uji tetapi menggunakan tools diharapkan akan lebih otomatis
  - Misalnya penggunaan JUnit (untuk java), NUnit (.NET) atau PUnit (Php).



# V & V

- *Verifikasi*
  - Aktivitas untuk menjamin bahwa **proses implementasi** suatu fungsi sudah di implementasikan dengan cara yang benar
- *Validasi*
  - Sekumpulan aktivitas untuk menjamin bahwa **fungsi sudah dibuat dengan benar** sesuai dengan kebutuhan pengguna
    - Boehm [Boe81]:
      - *Verification*: "Are we building the product right?"
      - *Validation*: "Are we building the right product?"



# V vs. V

	Verification	Validation
Tujuan	Mengevaluasi pengembangan produk sesuai dengan spesifikasi kebutuhan dan perancangan	Mengevaluasi produk terhadap kebutuhan/harapan pengguna
Aktivitas	Review , Meeting, Inspeksi	Black box/white box testing
Pelaku	Tim QA (Quality Assurance)	Tim Penguji (Testing Tim)
Eksekusi Kode	Tidak ada eksekusi kode program, hanya memeriksa dokumen, model	Perlu ada eksekusi kode program
Biaya perbaikan kesalahan	Biaya lebih murah, kesalahan dapat ditemukan pada tahapan lebih awal	Biaya lebih mahal Karena kesalahan ditemukan di akhir

# *Strategi Pengujian*

- Dimulai dari yang kecil ('testing-in-the-small') hingga bagian yang lebih besar ('testing-in-the-large')
- Untuk software biasa
  - Fokus awal adalah pengujian modul (unit)
  - Diikuti dengan pengujian integrasi modul
- Untuk software OO
  - Fokusnya pada pengujian pada setiap kelas yang melibatkan atribut dan operasinya
  - Berikut komunikasi/kolaborasi antar kelas

# ***Pengujian yang efektif***

- Lakukan **Technical Review**
  - Banyak 'error' akan ditemukan sebelum pengujian benar-benar dilakukan
- Pengujian dimulai dari elemen yang paling dasar hingga integrasi antar modul dan integrasi dengan sistem
- Teknik pengujian yang dipilih disesuaikan dengan teknik pendekatan software engineering
  - Konvensional? OO ?
- Pengujian dilakukan oleh developer
  - Untuk skala besar dilakukan oleh kelompok pengujian yang independen dari pengembang

