

Dasar-dasar Pengujian Perangkat Lunak

Minggu ke 5

Pengujian / testing

- ◆ *Testing is the exposure of a system to trial input to see wheter it produces corect output*
- ◆ *Adalah proses eksekusi suatu program dengan maksud menemukan kesalahan*

Pengujian perangkat lunak

- ◆ *Elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain dan pengkodean*

Proses Testing

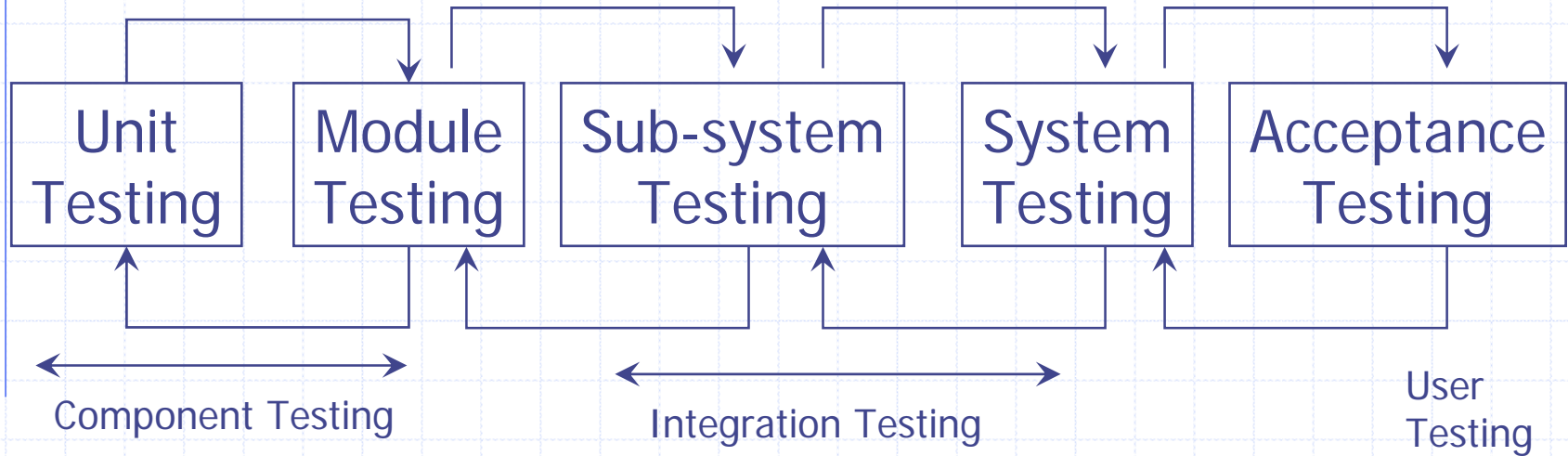
◆ System Testing

- Pengujian terhadap integrasi sub-system, yaitu keterhubungan antar sub-system

◆ Acceptance Testing

- Pengujian terakhir sebelum sistem dipakai oleh user.
- Melibatkan pengujian dengan data dari pengguna sistem.
- Biasa dikenal sebagai "alpha test" ("beta test" untuk software komersial, dimana pengujian dilakukan oleh potensial customer)

Proses Testing



The testing process

◆ Component testing

- Pengujian komponen-komponen program
- Biasanya dilakukan oleh component developer (kecuali untuk system kritis)

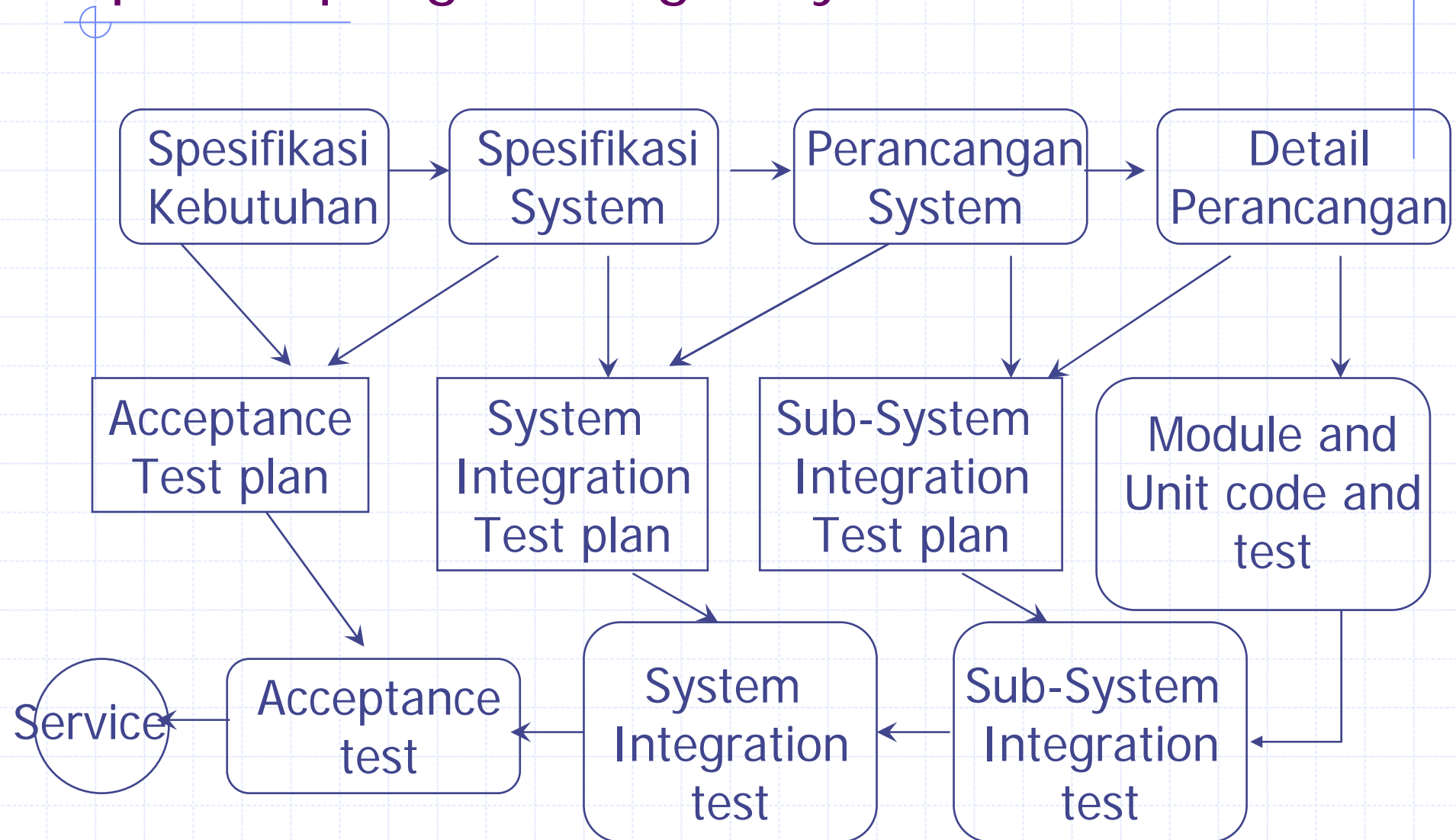
◆ Integration testing

- Pengujian kelompok komponen-komponen yang terintegrasi untuk membentuk sub-system ataupun system
- Dialakukan oleh tim penguji yang independent
- Pengujian berdasarkan spesifikasi sistem

Rencana Pengujian

- ◆ Proses testing
 - Deskripsi fase-fase utama dalam pengujian
- ◆ Pelacakan Kebutuhan
 - Semua kebutuhan user diuji secara individu
- ◆ Item yg diuji
 - Menspesifikasi komponen sistem yang diuji
- ◆ Jadwal Testing
- ◆ Prosedur Pencatatan Hasil dan Prosedur
- ◆ Kebutuhan akan Hardware dan Software
- ◆ Kendala-kendala
 - Mis: kekurangan staff, alat, waktu dll.

Hubungan antara rencana pengujian dan proses pengembangan system

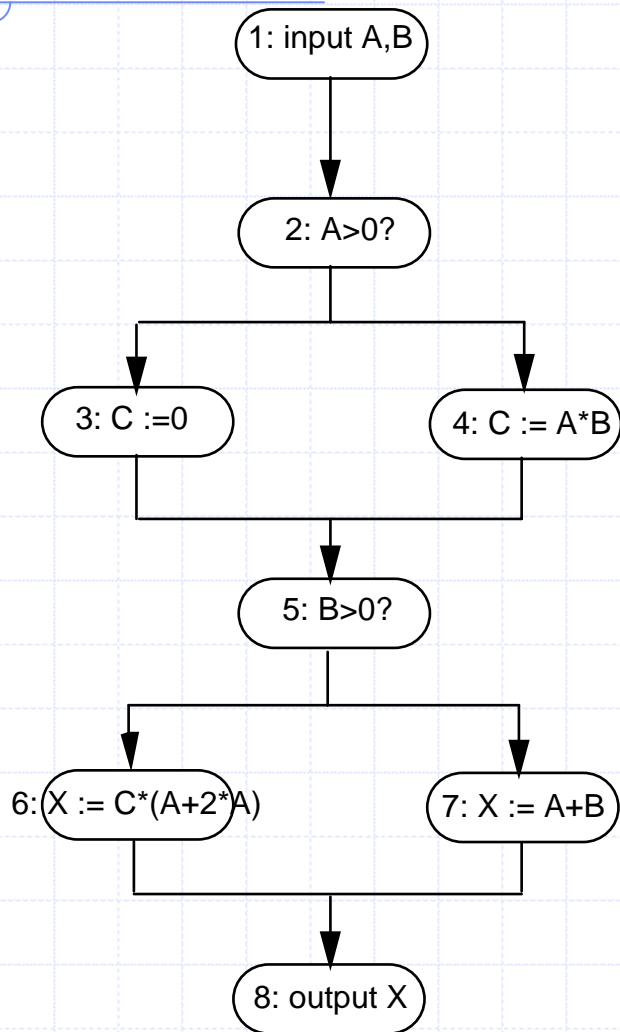


Failures, Faults

- ◆ Failure: output yang tidak benar/tidak sesuai ketika sistem dijalankan
- ◆ Fault: kesalahan dalam source code yang mungkin menimbulkan failure ketika code yg fault tsb dijalankan

Failure Class	Deskripsi
Transient	Muncul untuk input tertentu
Permanent	Muncul untuk semua input
Recoverable	Sistem dapat memperbaiki secara otomatis
Unrecoverable	Sistem tidak dapat memperbaiki secara otomatis
Non-corrupting	Failure tidak merusak data
Corrupting	Failure yang merusak sistem data

Contoh: Faults, Errors, and Failures



- Suppose node 6 should be $X := C * (A + 2 * B)$
 - Failure-less fault:
 - » executing path (1,2,4,5,7,8) will not reveal this fault because 6 is not executed
 - » nor will executing path (1,2,3,5,6,8) because $C = 0$
- Need to make sure proper test cases are selected
 - the definitions of C at nodes 3 and 4 both affect the use of C at node 6
 - » executing path (1,2,4,5,6,8) will reveal the failure, but only if $B \neq 0$

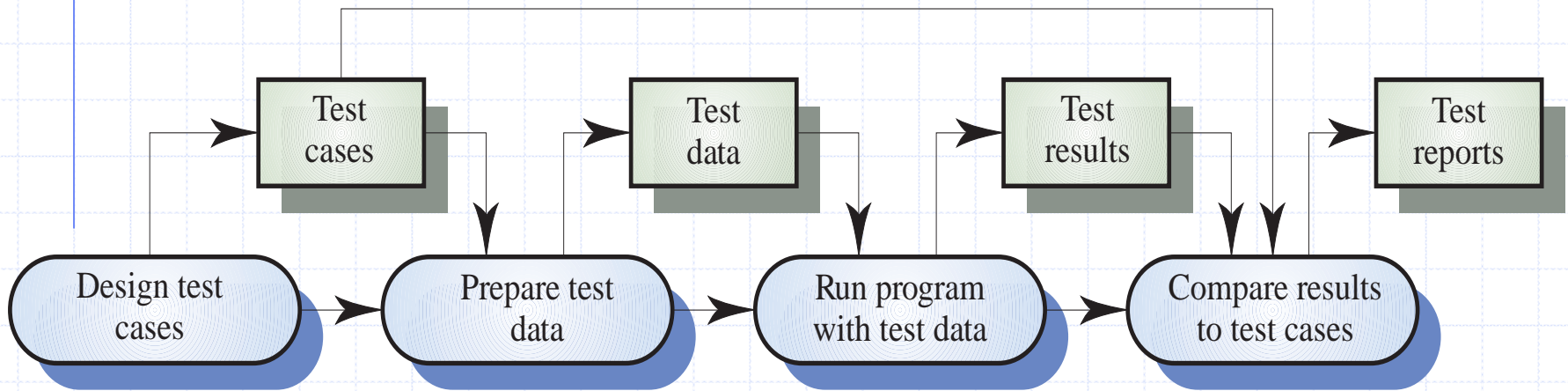
Prioritas Testing

- ◆ Hanya test yang lengkap yg dapat meyakinkan sistem terbebas dari kesalahan, tetapi hal ini sangat sulit dilakukan.
- ◆ Prioritas dilakukan terhadap pengujian kemampuan sistem, bukan masing-masing komponennya.
- ◆ Pengujian untuk situasi yg tipikal lebih penting dibandingkan pengujian terhadap nilai batas.

Test data dan kasus test

- ◆ *Test data*: Input yang yang direncanakan digunakan oleh sistem.
- ◆ *Test cases*: Input yang digunakan untuk menguji sistem dan memprediksi output dari input jika sistem beroperasi sesuai dengan spesifikasi.

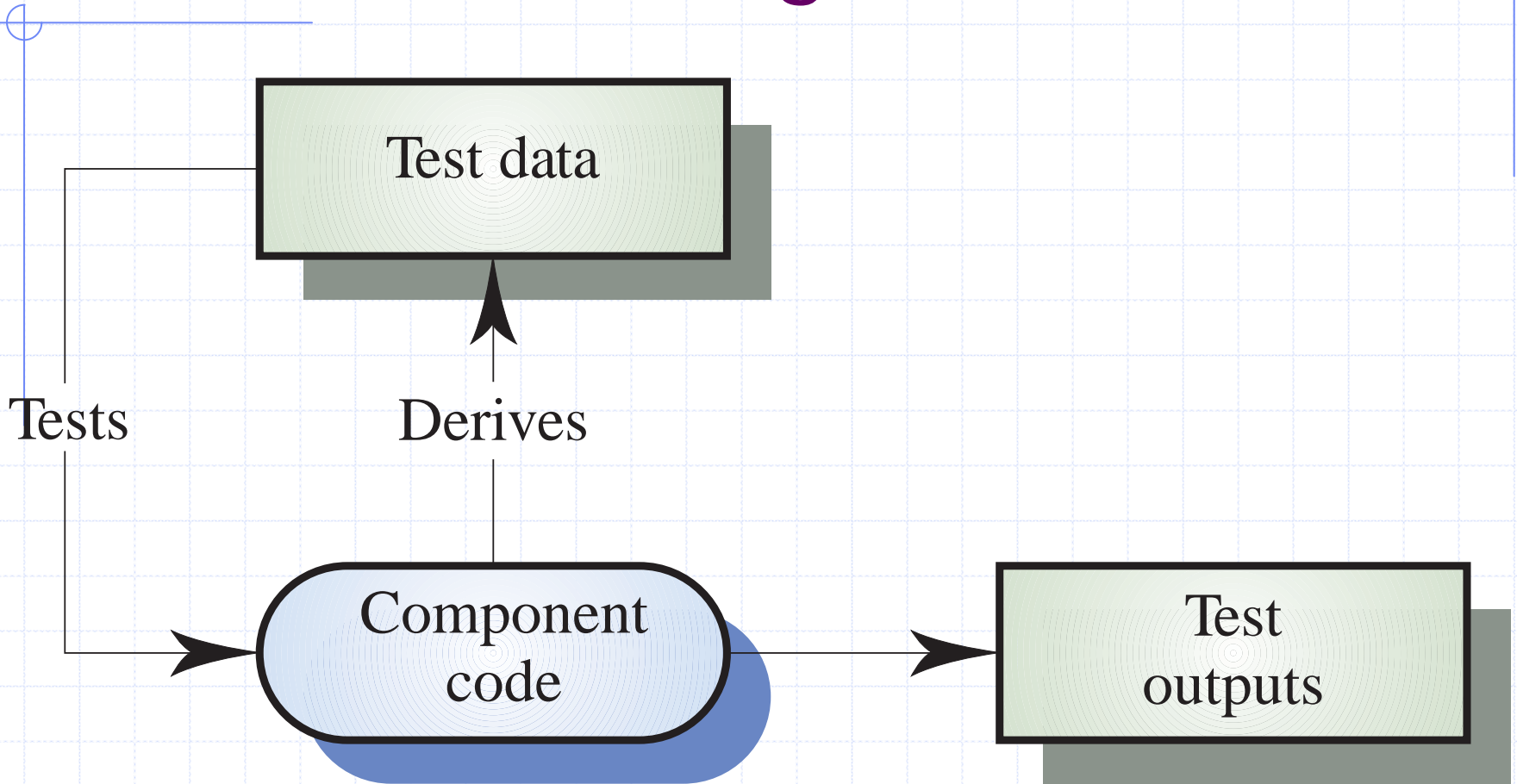
Proses defect testing



Structural testing

- ◆ Disebut juga white-box testing
- ◆ Penentuan test case disesuaikan dengan struktur sistem. Knowledge program digunakan untuk mengidentifikasi test case tambahan.
- ◆ Tujuannya untuk menguji semua statement program (debug).

White-box testing

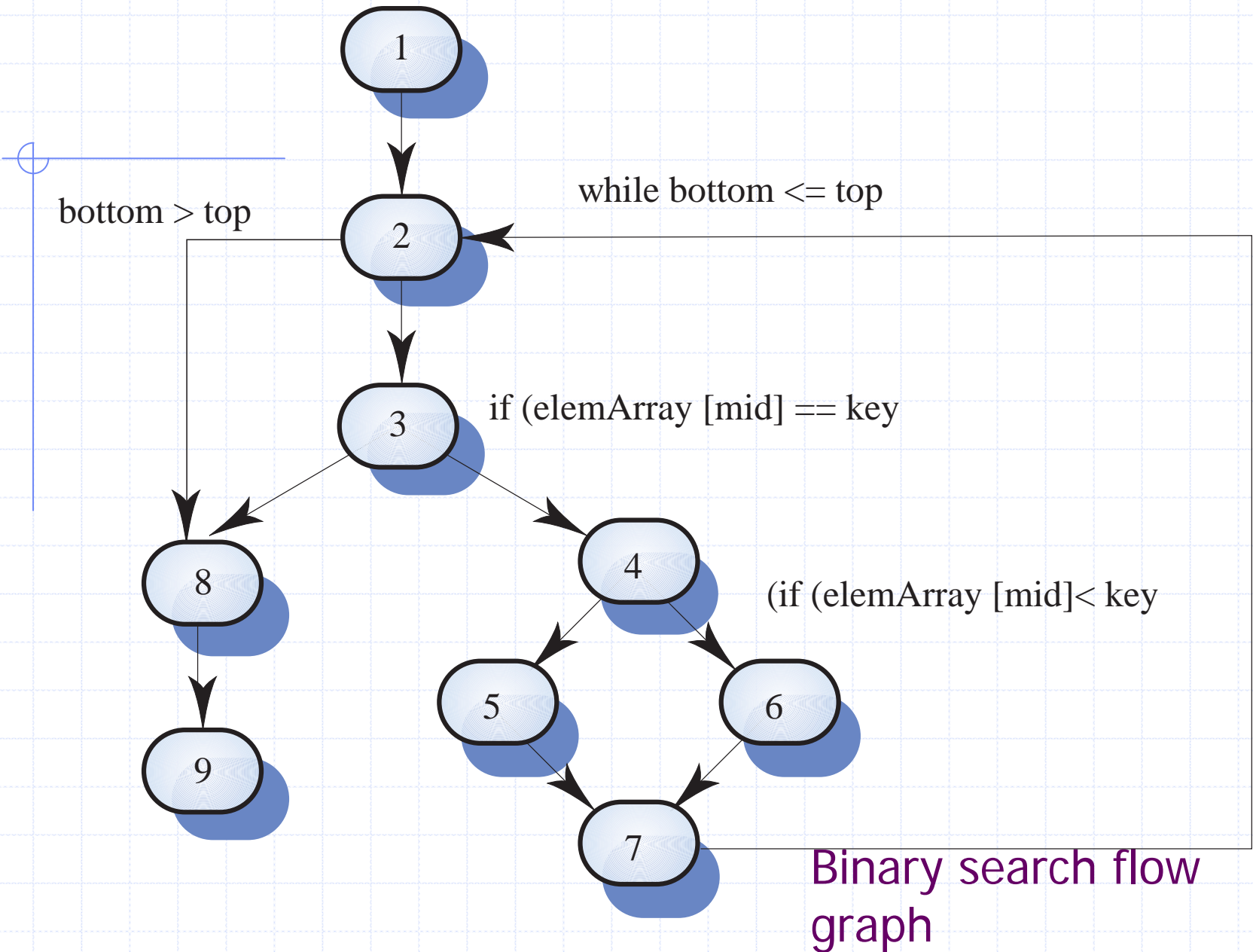


Path testing

- ◆ Tujuannya meyakinkan bahwa himpunan test case akan menguji setiap path pada suatu program paling sedikit satu kali.
- ◆ Titik awal untuk path testing adalah suatu program flow graph yang menunjukkan node-node yang menyatakan program decisions (mis.: if-then-else condition) dan busur menyatakan alur kontrol
- ◆ Statements dengan conditions adalah node-node dalam flow graf.

Program flow graphs

- ◆ Menggambarkan alur kontrol. Setiap cabang ditunjukkan oleh path yg terpisah dan loop ditunjukkan oleh arrows looping kembali ke loop kondisi node.
- ◆ Digunakan sebagai basis untuk menghitung cyclomatic complexity
- ◆ Cyclomatic complexity = Jumlah edges – Jumlah Node + 2
- ◆ Cyclomatic complexity menyatakan jumlah test untuk menguji control statements



Independent paths

◆ 1, 2, 3, 8, 9

◆ 1, 2, 3, 4, 6, 7, 2

◆ 1, 2, 3, 4, 5, 7, 2

◆ 1, 2, 3, 4, 6, 7, 2, 8, 9

◆ Test cases harus ditentukan sehingga semua path tsb tereksekusi.