

IKI10400 • Struktur Data & Algoritma: B-Tree

Fakultas Ilmu Komputer • Universitas Indonesia

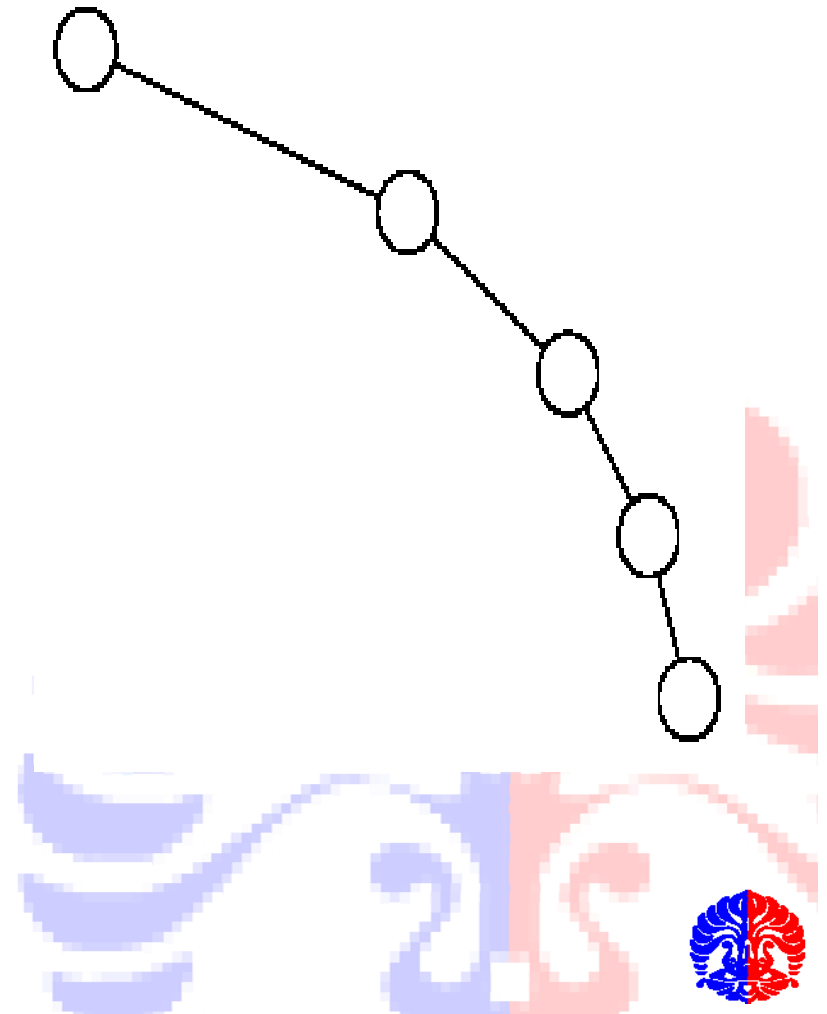
Slide acknowledgments:

Suryana Setiawan, Ade Azurat, Denny, Ruli Manurung, Tisha Melia



Motivasi B Trees

- The problem with Binary Trees is balance, the tree can easily deteriorate to a linked list. Consequently, the reduced search times are lost, this problem is overcome in B-Trees.
 - B stands for Balanced, where all the leaves are the same distance from the root. B-Trees guarantee a predictable efficiency.
- There are several varieties of B-Trees, most applications use the B+Tree.



Motivasi

- Perhatikan kasus berikut ini:
 - Kita harus membuat program basisdata untuk menyimpan data di yellow pages daerah Jakarta, misalnya ada 2.000.000 data.
 - Setiap entry terdapat nama, alamat, nomor telepon, dll. Asumsi setiap entry disimpan dalam sebuah record yang besarnya 512 byte.
 - Total file size = $2,000,000 * 512 \text{ byte} = 1 \text{ GB}$.
 - terlalu besar untuk disimpan dalam memory (primary storage)
 - perlu disimpan di disk (secondary storage)



Motivasi

- Jika kita menggunakan disk untuk penyimpanan, kita harus menggunakan struktur blok pada disk untuk menyimpan basis data tsb.
 - Secondary storage dibagi menjadi blok-blok yang ukurannya sama. Umumnya 512 byte, 2 KB, 4 KB, 8 KB.
 - Block adalah satuan unit transfer antar disk dengan memory. Walaupun program hanya membaca 10 byte dari disk, 1 block akan dibaca dari disk dan disimpan ke memory.
- Misalnya 1 disk block 8.192 byte (8 KB)
 - Maka jumlah blok yang diperlukan:
 $1 \text{ GB} / 8 \text{ KB per block} = 125,000 \text{ blocks.}$
 - Setiap blok menyimpan:
 $8,192 / 512 = 16 \text{ records.}$



Motivasi

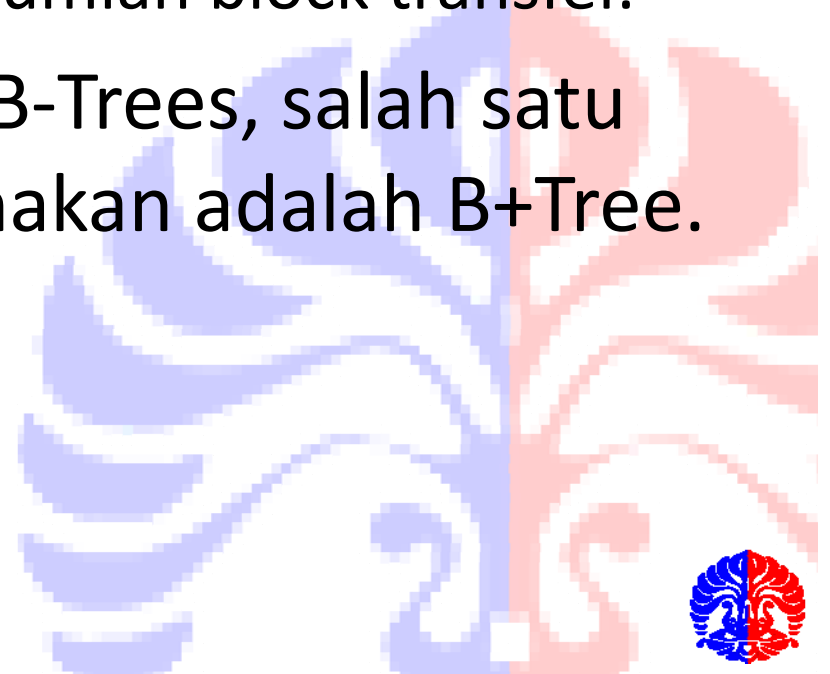
- Karena keterbatasan mekanik, sebuah akses ke harddisk (storage) membutuhkan waktu yang relatif sangat lama.
 - Akses ke disk diperkirakan 10,000 kali lebih lambat dari pada akses ke main memory.
 - Sebuah akses ke disk dapat disetarakan dengan 200,000 buah instruksi.
- Dengan demikian jumlah akses ke disk akan mendominasi running time.
- Kita membutuhkan:

Sebuah teknik pencarian “multiway search tree” yang dapat meminimalkan jumlah akses ke disk.



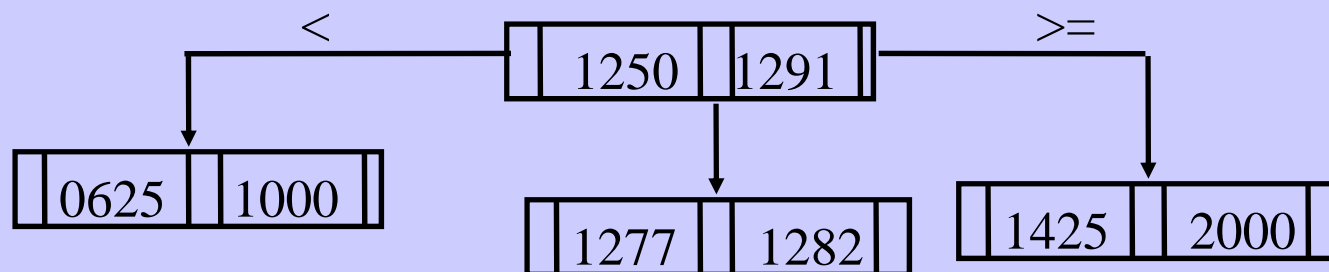
B-Tree

- B-tree banyak digunakan untuk external data structure.
 - Setiap node berukuran sesuai dengan ukuran block pada disk, misalnya 1 block = 8 KB.
 - Tujuannya: meminimalkan jumlah block transfer.
- Ada beberapa variasi dari B-Trees, salah satu contoh yang banyak di gunakan adalah B+Tree.



B Tree

- B Tree dengan degree m memiliki karakteristik sebagai berikut:
 - Setiap non-leaf (internal) nodes (kecuali root) jumlah anaknya (yang tidak null) antara $\lceil m/2 \rceil$ dan m .
 - Sebuah non-leaf (internal) node yang memiliki m cabang memiliki sejumlah $m-1$ keys.
 - Setiap leaves berada pada *level* yang sama, (dengan kata lain, memiliki *depth* yang sama dari *root*).

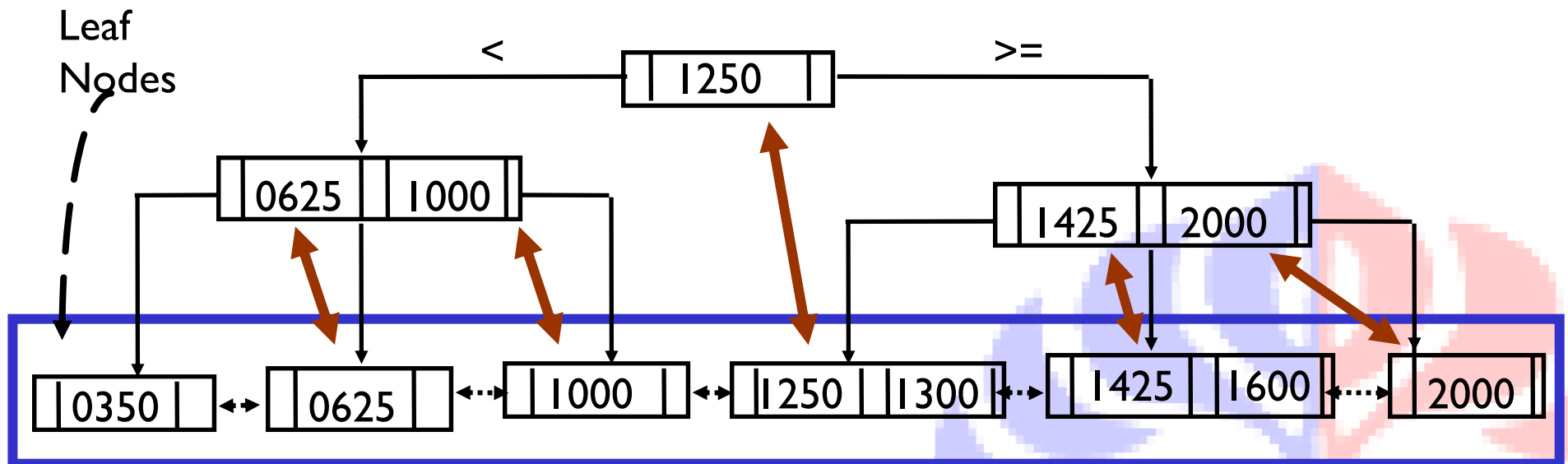


B+Tree

- B+Tree adalah variant dari B-tree, dengan aturan:
 - semua key value sebagai reference terhadap data disimpan dalam leaf.
 - disertakan suatu pointer tambahan untuk menghubungkan setiap leaf node tersebut sebagai suatu linear linked-list.
 - Struktur ini memungkinkan akses sikuensial data dalam B-tree tanpa harus turun-naik pada struktur hirarkisnya.
 - node internal digunakan sebagai ‘indeks’ (dummy key).
 - beberapa *key value* dapat muncul dua kali di dalam tree (sekali pada leaf sebagai reference thd data kedua sebagai indeks pada internal node).

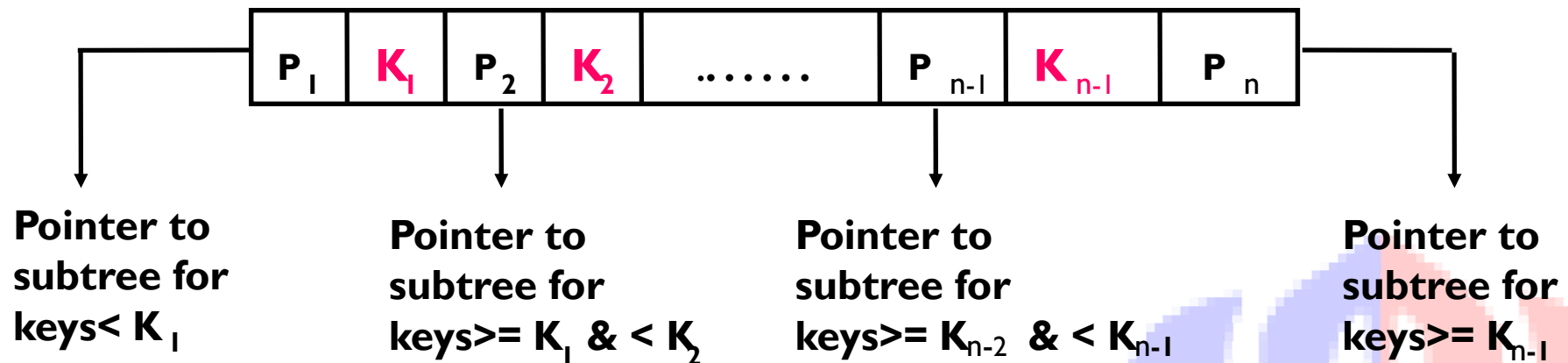


B+Tree

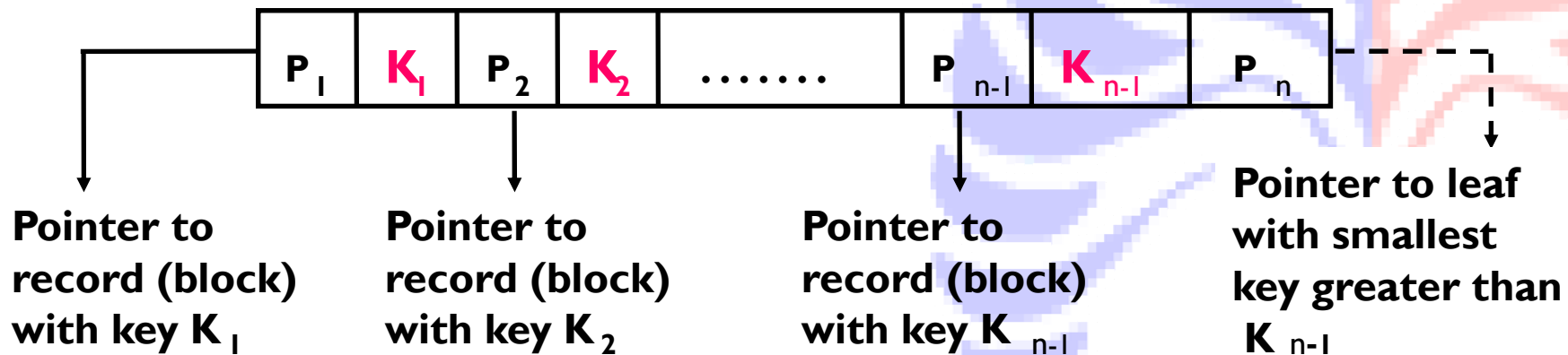


Struktur: B+Tree Node

A high level node (internal node)

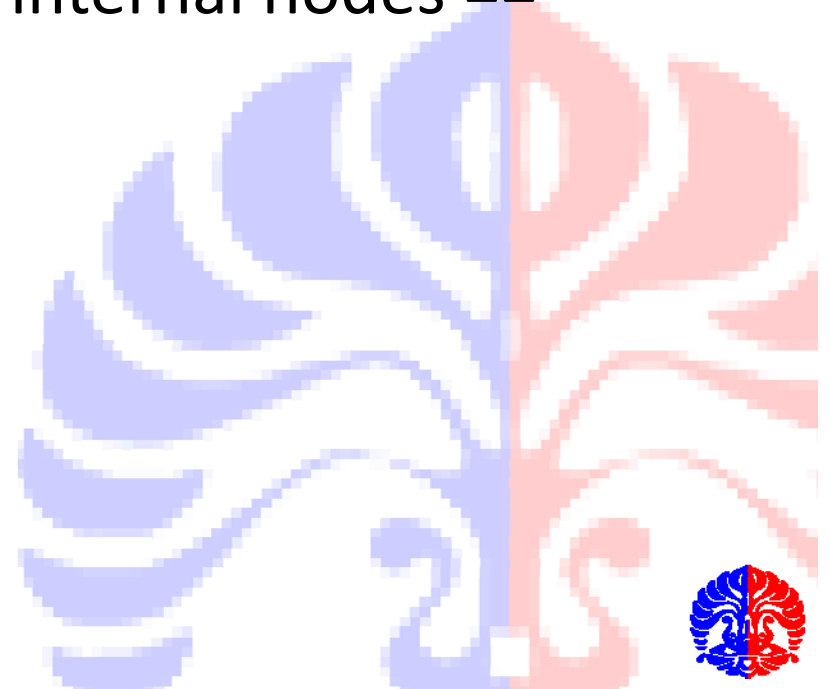


A leaf node (Every key value appears in a leaf node)

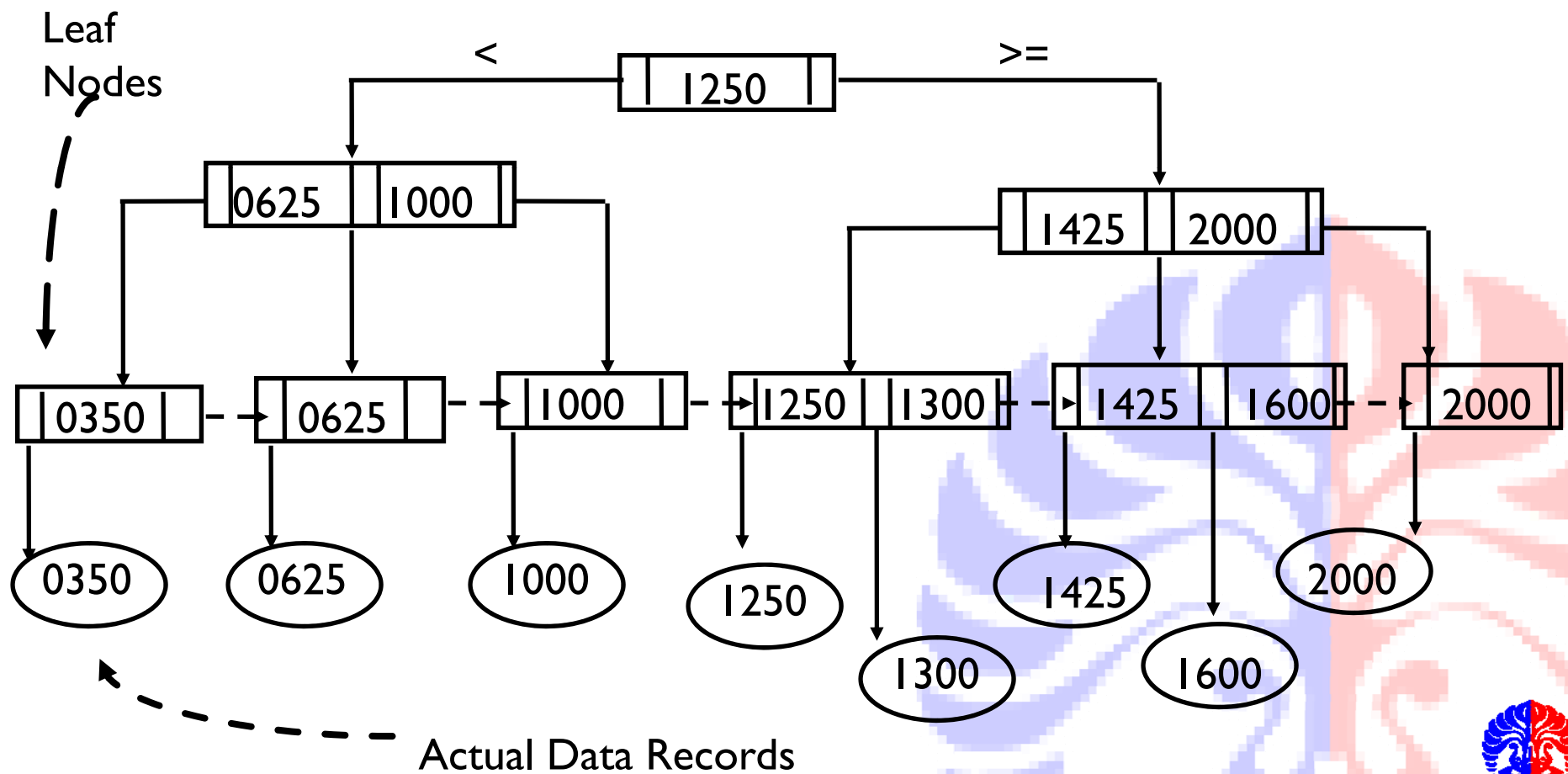


Struktur: B+Tree Node

- Perhatikan bahwa:
 - jumlah pointer/children pada leaf nodes == jumlah key
 - karena pointer terakhir menunjuk ke leaf node berikutnya
 - jumlah pointer/children pada internal nodes == jumlah key + 1



Contoh sebuah B+Tree



Proses pada B+Trees

- Mencari records dengan *search-key value* : k .
 1. Mulai dari root.
 - a. Periksa node tersebut, dan tentukan nilai key terkecil pada node tersebut yang lebih besar dari k .
 - b. Jika key tersebut ditemukan, ada K_i , key terkecil pada node yang memenuhi syarat: $K_i > k$, maka ikuti P_i (rekursif terhadap node yang ditunjukkan oleh P_i)
 - c. Jika tidak ditemukan dan $k \geq K_{m-1}$, serta ada m pointers pada node. Maka ikuti P_m (rekursif terhadap node yang ditunjukkan oleh P_i)
 - Bila node yang ditunjukkan oleh pointer tersebut di atas internal node (non-leaf node), ulangi prosedur a-c.
 - Bila mencapai sebuah *leaf node*. Jika ada i sehingga nilai key $K_i = k$, maka ikuti pointer P_i untuk mendapatkan record (atau *bucket*) yang diinginkan. Jika tidak, maka tidak ada data dengan nilai key k .



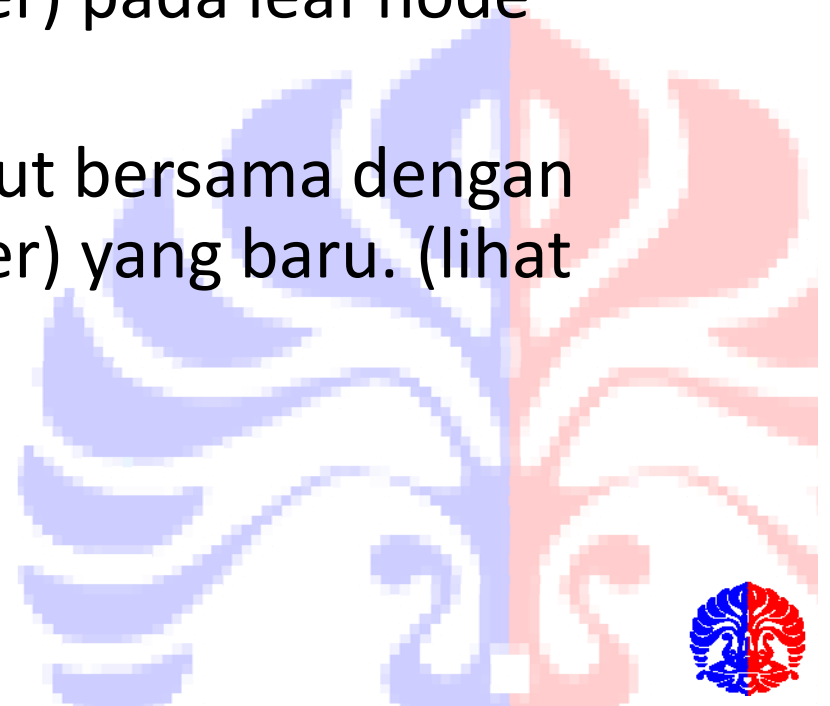
Proses pada B+Trees: *Range Query*

- Mencari seluruh record yang berada diantara k dan l (range query).
 - Mencari record dengan search-key value = k .
 - selama nilai search-key value selanjutnya yang ditunjukkan oleh pointer lebih kecil dari l , ikuti pointer untuk mendapatkan records yang diinginkan
 - jika search-key yang ditemukan adalah search-key yang terakhir dalam node, ikuti pointer yang terakhir (P_n) untuk menuju leaf node selanjutnya.
- Bandingkan dengan proses yang sama pada binary search tree



Insertion on B+Trees

- Cari posisi leaf node yang sesuai agar nilai search-key yang baru dapat diletakkan secara terurut.
 - Jika masih ada tempat pada leaf node, tambahkan pasangan (key-value, pointer) pada leaf node secara terurut
 - Bila tidak, split node tersebut bersama dengan pasangan (key-value, pointer) yang baru. (lihat slide selanjutnya)



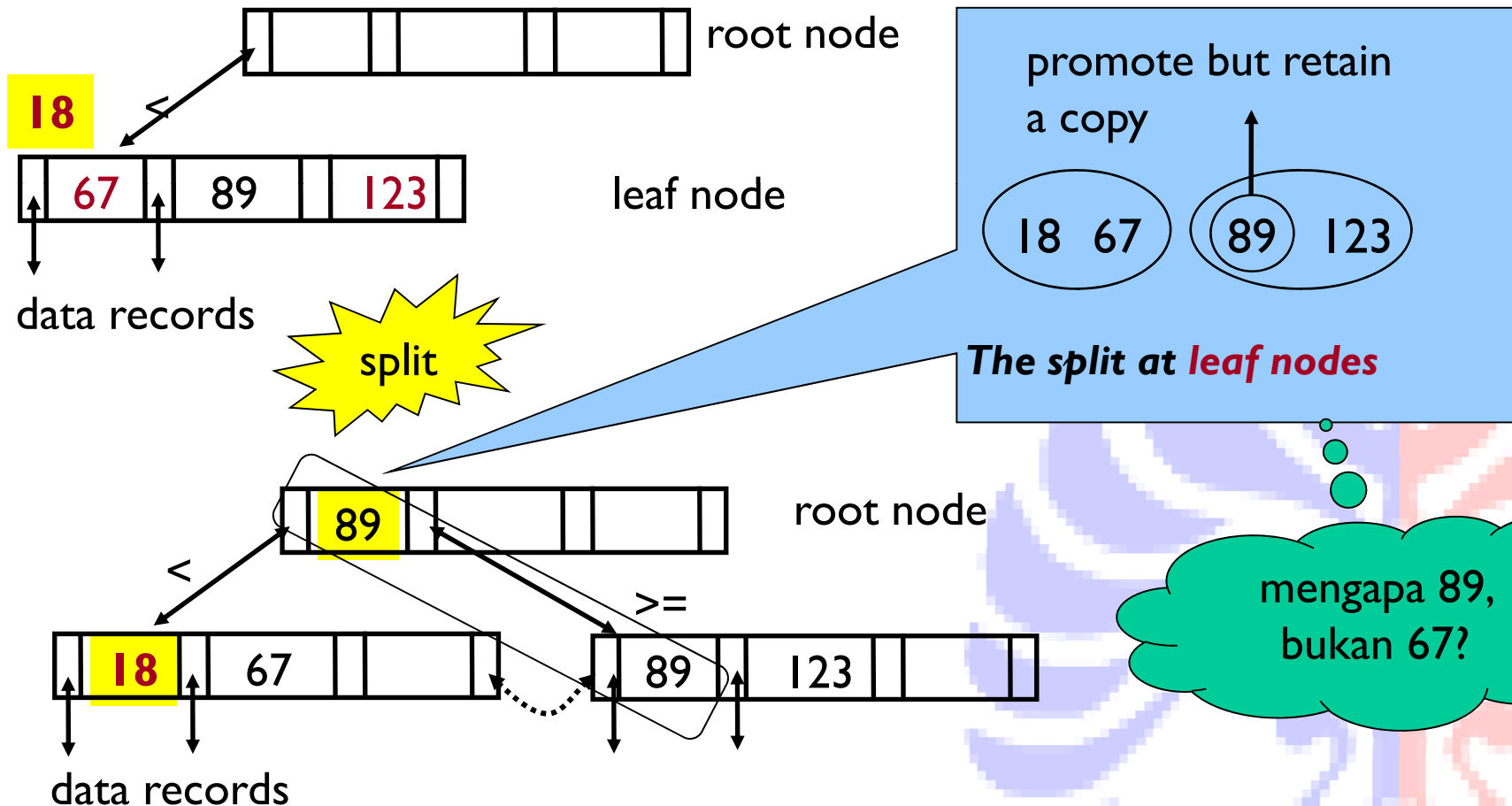
Insertion on B+Trees

- Proses Splitting pada sebuah node:
 - ambil pasangan (search-key value, pointer) **termasuk yang baru ditambahkan**. letakkan search-key $\lfloor m/2 \rfloor$ yang pertama pada node awal, dan pindahkan sisanya pada sebuah node baru.
 - ketika melakukan splitting **pada sebuah leaf**, promosikan middle/median key dari node yang akan di split kepada parent node, **tanpa menghapus pada leaf** tersebut (meng-copy).
 - ketika melakukan splitting **pada internal node**, promosikan the middle/median key dari node yang akan displit kepada parent node, dan **hapus pada node** sebelumnya (tidak membuat copy).
 - Jika hasil promosi membuat parent menjadi full, lakukan proses splitting serupa secara rekursif.

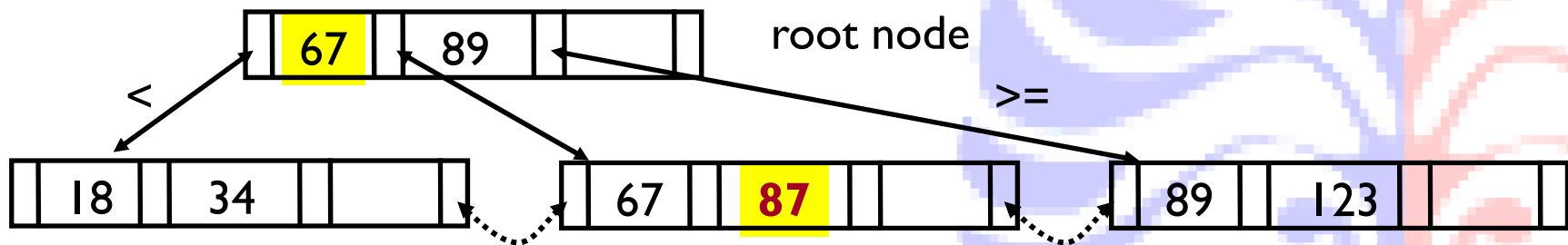
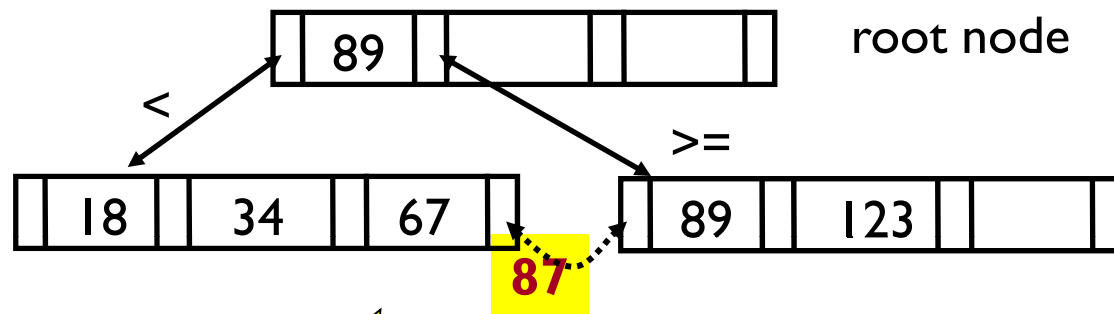


Building a B+Tree (m=4)

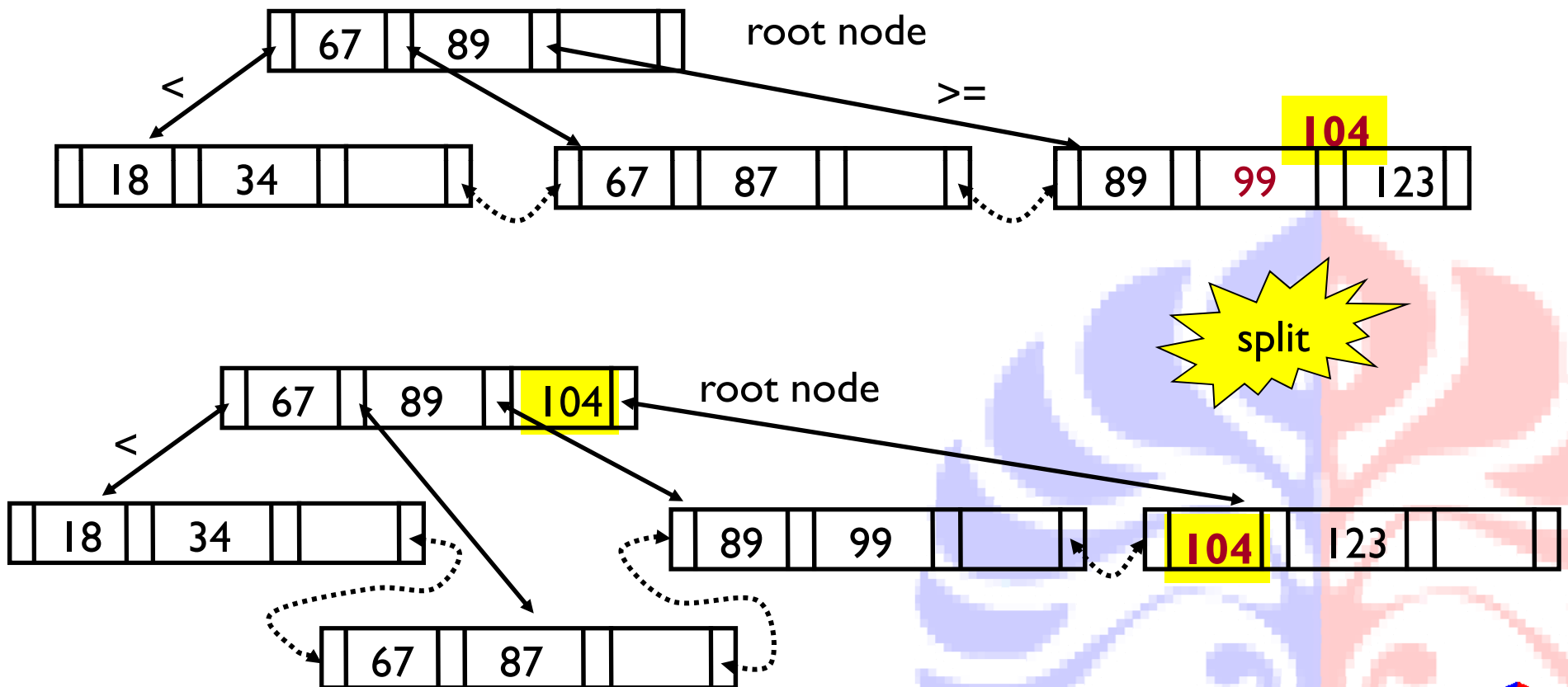
67, 123, 89, 18, 34, 87, 99, 104, 36, 55, 78, 9



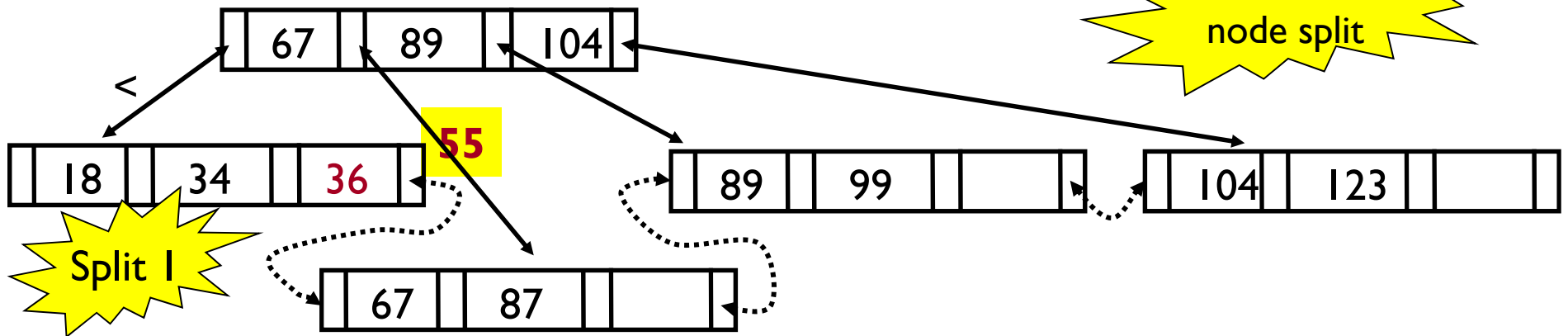
67, 123, 89, 18, 34, **87**, 99, 104, 36, 55, 78, 9



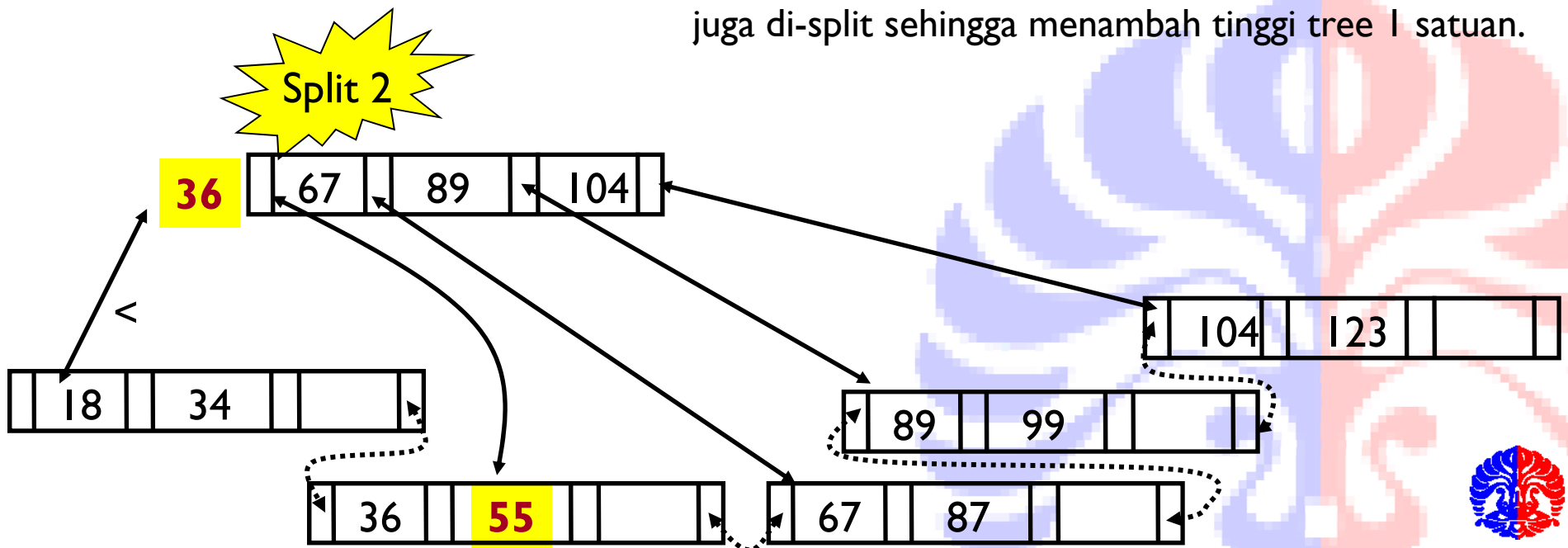
67, 123, 89, 18, 34, 87, 99, **104**, 36, 55, 78, 9



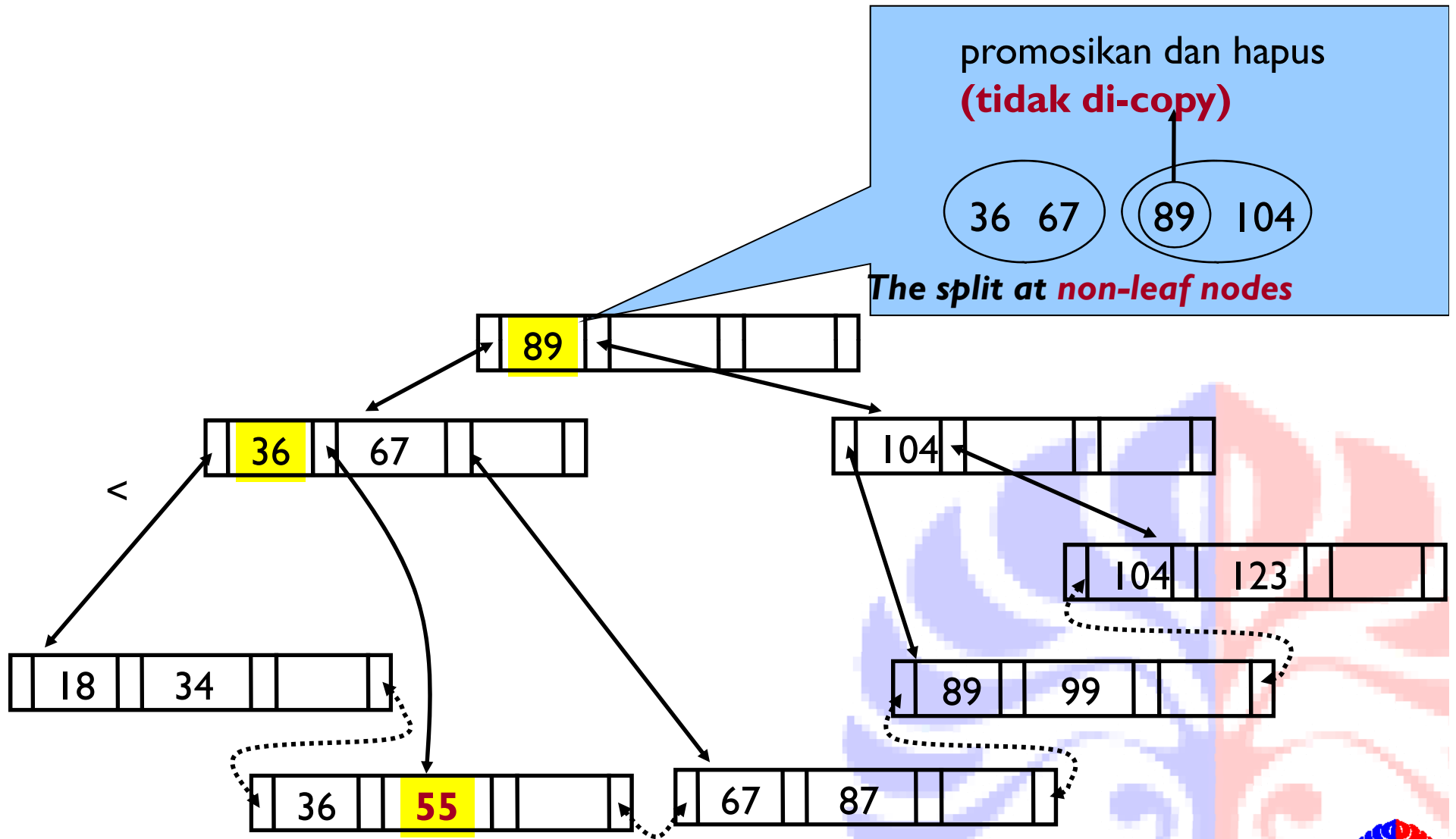
67, 123, 89, 18, 34, 87, 99, 104, 36, **55**, 78, 9



Proses splitting diteruskan ke atas hingga node tersebut tidak full. Pada worst case, root node bisa juga di-split sehingga menambah tinggi tree 1 satuan.



67, 123, 89, 18, 34, 87, 99, 104, 36, **55**, 78, 9



Observasi mengenai B+Trees

- B+Tree umumnya memiliki jumlah levels yang rendah (logarithmic pada jumlah data), sehingga proses pencarian dapat dilakukan dengan efisien.
 - Saat memproses sebuah query, sebuah jalur (path) dijalani pada tree dari root hingga leaf node.
 - jika terdapat sejumlah K search-key pada sebuah file, maka jalur pencarian tidak akan lebih besar dari $\lceil \log_{\lceil m/2 \rceil}(K) \rceil$, m adalah degree B+ Tree.
 - pertanyaan: Mengapa “ $\log_{\lceil m/2 \rceil}$ ”, bukan “ \log_m ” ?



Deletion on B+Trees

- hapus pasangan (search-key value, pointer) dari leaf node
- jika node ternyata memiliki pasangan yang terlalu sedikit, dan jika jumlah pasangan pada node dan sibling-nya **dapat** digabungkan pada sebuah node, maka
 - gabungkan kedua node tersebut
 - hapus pasangan (K_{i-1}, P_i) , pada parent node-nya dimana P_i adalah pointer terhadap node yang dihapus.
 - Lakukan secara recursively hingga root.
- minimum requirement:
 - $\lceil m/2 \rceil$ children pada internal node
 - $\lceil m/2 \rceil$ children pada leaf, #children == #keys pada leaf

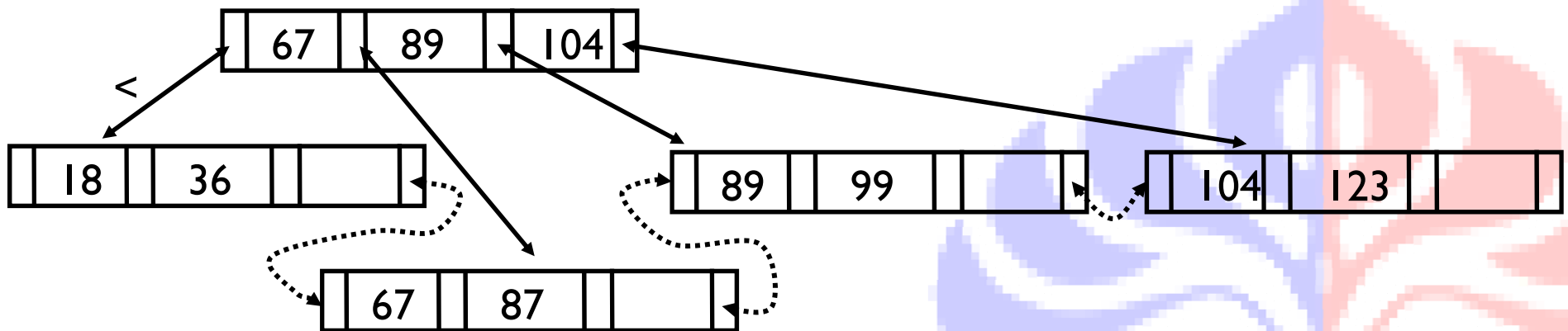
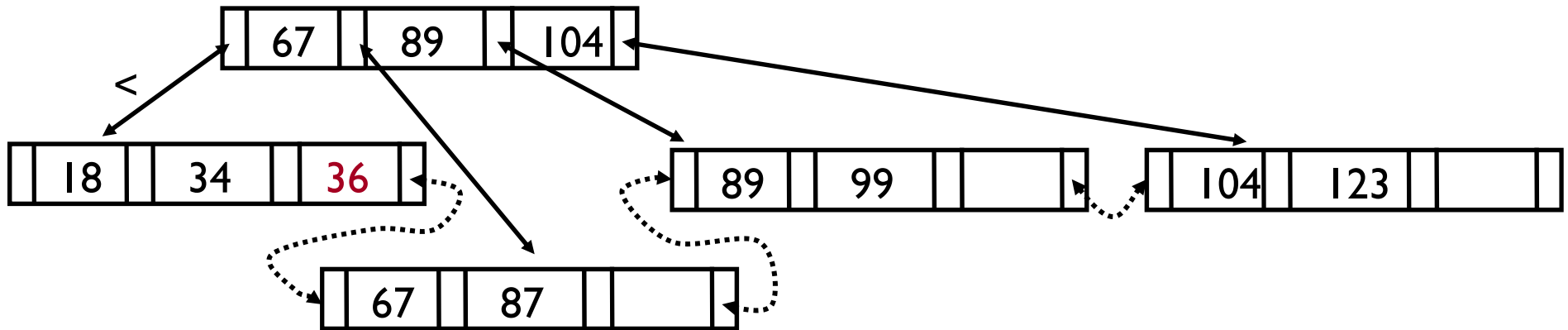


Deletion on B+Trees

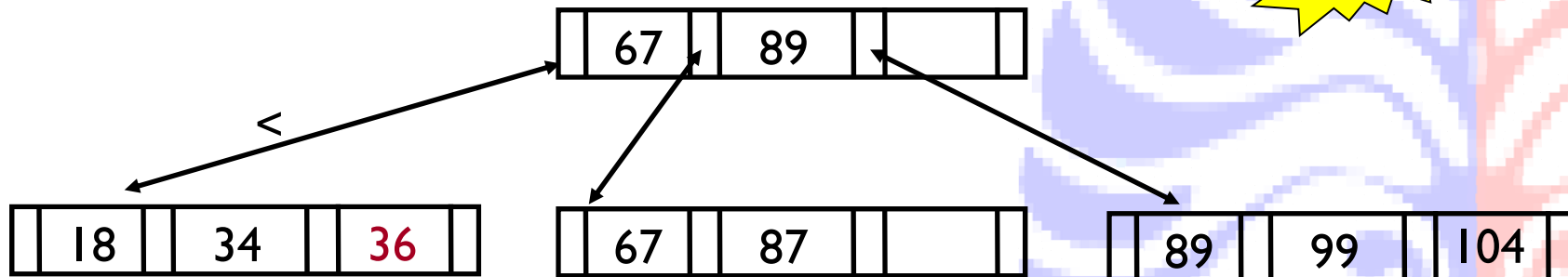
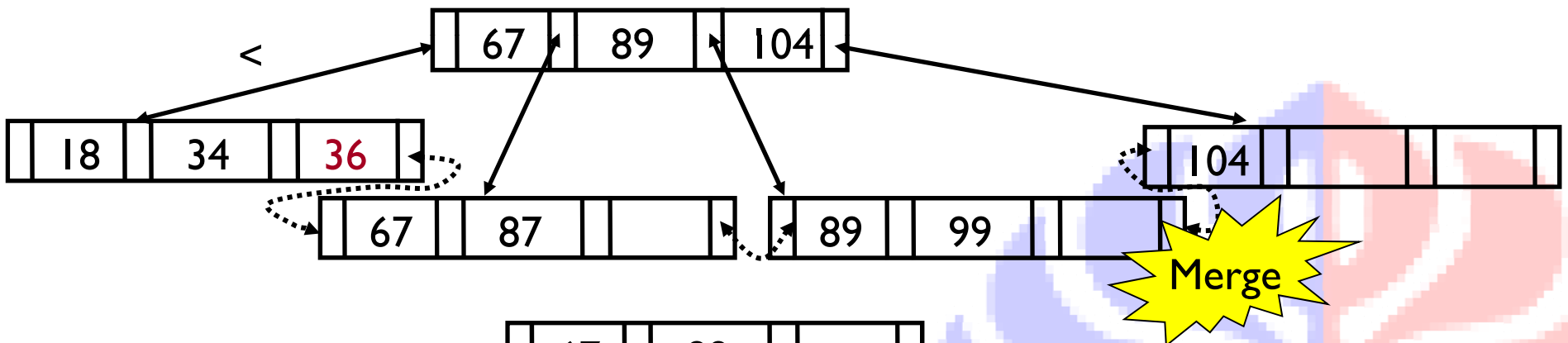
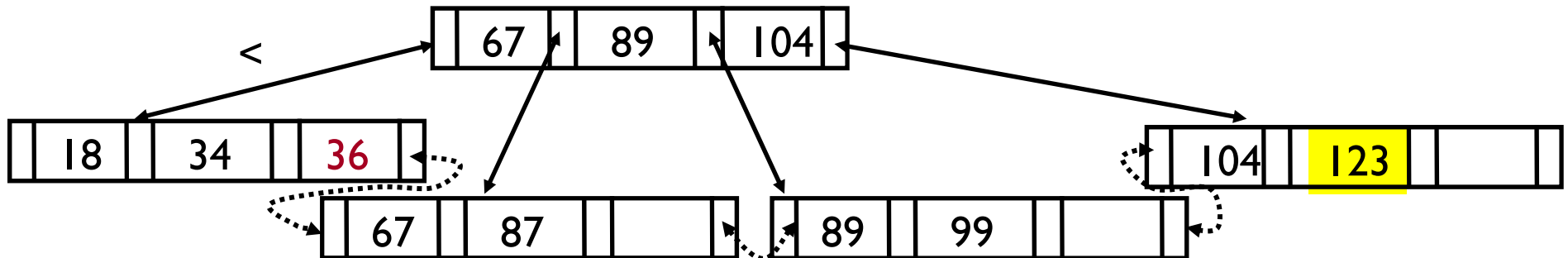
- jika jumlah pasangan pada node dan sibling-nya **tidak dapat** digabungkan pada sebuah node, maka
- re-distribusikan pointers diantara node tersebut dan sibling-nya sehingga mereka memiliki jumlah element yang lebih dari minimum.
 - Update search-key yang berkesesuaian pada parent .
 - sibling → memiliki parent yang sama
- proses ini dapat terus berlaku rekursif ke parent hingga ditemukan node yang memiliki pasangan sejumlah $\lceil m/2 \rceil$ atau lebih.



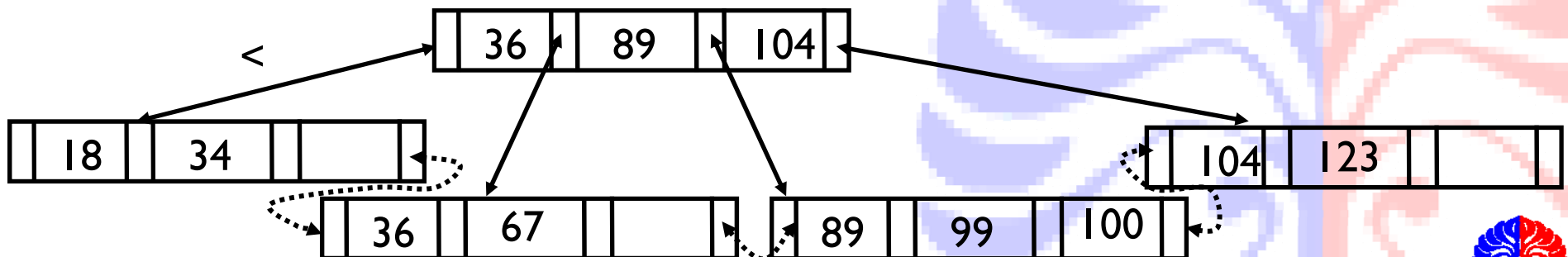
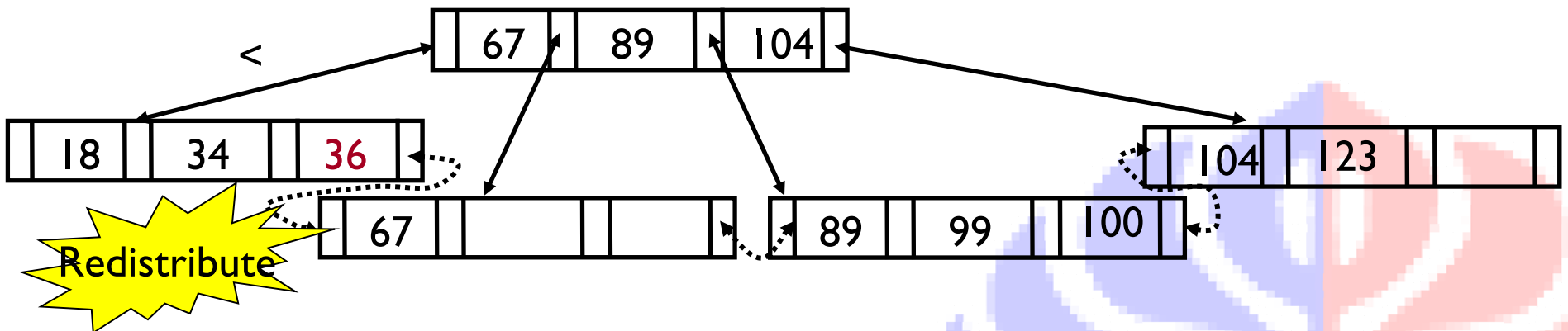
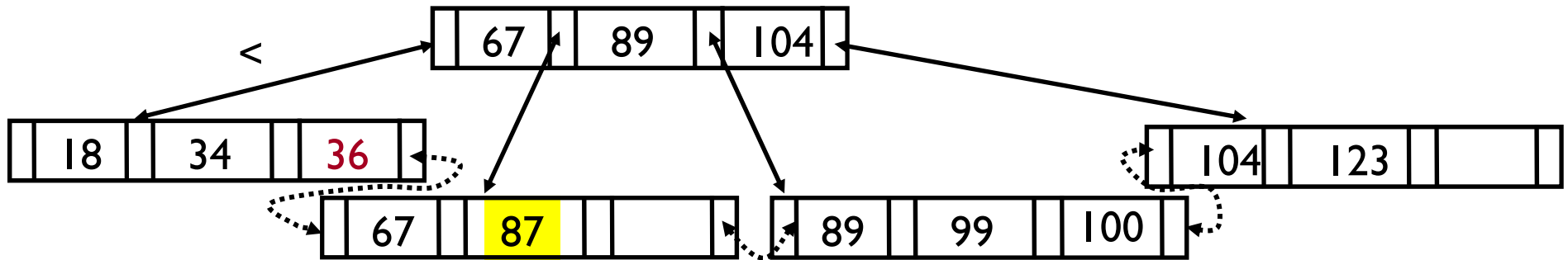
Hapus 34 → OK



Hapus 123 → Merge



Hapus 87 → Redistribute



Rangkuman

- B Tree biasanya digunakan sebagai struktur data external pada databases.
- B Tree dengan degree m memiliki aturan seperti berikut:
 - Setiap non-leaf (internal) nodes (kecuali root) jumlah anaknya (yang tidak null) antara $\lceil m/2 \rceil$ dan m .
 - Sebuah non-leaf (internal) node yang memiliki m cabang memiliki sejumlah $m-1$ keys.
 - Setiap leaves berada pada *level* yang sama, (dengan kata lain, memiliki *depth* yang sama dari *root*).
- B+Tree adalah variant dari B Tree dimana seluruh key terletak pada leaves.



Tambahan informasi dan latihan

- applet simulasi B+Tree
 - <http://slady.net/java/bt/view.php?w=600&h=450>

