IF3230 – Sistem Terdistribusi
Naming

Achmad Imam Kistijantoro (imam@informatika.org)
Judhi Santoso (judhi@informatika.org)
Anggrahita Bayu Sasmita (bayu.anggrahita@informatika.org)

Naming

- “My Laptop”
- Komputer no 1 baris 1
- Labsister-1
- Bromo.if.itb.ac.id
- 167.205.32.2
- 00:14:51:ec:fa:1d

Penamaan

- User name
- Machine name
- Files
- Devices
- Variables
- Network services

Naming service

- Layanan yang digunakan untuk lookup names
  - Mengembalikan address atau informasi lain

- Dapat diimplementasikan sebagai
  - Search terhadap isi sebuah file
  - Client/server program
  - Database query
  - ...

Apakah nama?

- Name: mengidentifikasikan apa yang diinginkan/dicari
- Address: identifikasi lokasi/tempat
- Route: identifikasi bagaimana mencapainya
- Binding: asosiasi nama dan address
  - Menentukan implementasi low-level berdasarkan informasi high level

RFC 1498: Internetwork Naming, addresses and routing

Name

- Name digunakan untuk mengidentifikasikan:
  - Layanan: e.g. Time of day
  - Nodes: komputer yang menjalankan service
  - Path: route
  - Object within service: e.g. File dalam sebuah file server

- Naming convention dapat menggunakan berbagai format
  - Menggunakan format yang sesuai dengan kebutuhan aplikasi/user
  - Misal: nama yang mudah dibaca untuk manusia, dan biner untuk mesin

Uniqueness of names

- Pada skala kecil, mudah
- Problematik pada skala besar
  - Sulit untuk menjamin unik untuk global names
- Hierarki memungkinkan pengelolaan keunikan nama
  - Ethernet address: 3 bytes organization, 3 byte controller
  - IP address: network address & host address
  - Domain name
  - URL
  - File path

Terms: naming convention

- Naming system menentukan sintaks nama
  - UNIX file name
    - Parse komponen dari kiri ke kanan, dengan separator /
    - /home/amir/file.txt
  - Internet domain name
    - Urut dari kanan ke kiri dengan delimiter .
    - If.stei.itb.ac.id
  - LDAP names
    - Pasangan atribut/value terurut dari kanan, dengan delimiter ,
    - cn= Achmad Imam, o=STEI, c=ID

Term: Context

- Konteks: himpunan/kumpulan nama => object binding
- Nama unik dalam sebuah konteks
  - e.g. /etc/httpd/conf/httpd.conf pada sebuah komputer tertentu
- Setiap konteks memiliki konvensi nama/naming convention tertentu
- Nama selalu diinterpretasikan relatif terhadap konteks tertentu
  - e.g. Direktori /etc/ pada sebuah UNIX file system

Term: Naming System

- Himpunan/kumpulan context yang terhubung dan bertipe sama, dan menyediakan beberapa operasi yang sama

- Misal:
  - sistem yang mengimplementasikan DNS
  - Sistem yang mengimplementasikan LDAP

Term: Name space

- Container untuk kumpulan nama dalam sebuah naming system

- Sebuah namespace dapat memiliki scope
  - Scope: region dimana nama exists dan mengacu ke object
  - Contoh:
    - Nama semua file dalam sebuah direktori
    - Semua domain name di dalam itb.ac.id
    - Java package, local variables etc.
- Namespace dapat berstruktur tree

Term: resolution

- Resolution: name lookup
  - Mengembalikan binding sebuah nama

- Contoh:
  - www.itb.ac.id => 167.205.1.34

Term: naming service

- Layanan yang menyediakan name resolution

- DNS server akan memetakan
  - www.itb.ac.id => 167.205.1.34

Directory Service

- Extension dari name service
  - Asosiasi nama dengan objek
  - Memungkinkan objek memiliki atribut
  - Dapat melakukan pencarian berdasarkan atribut

- Contoh: LDAP (Lightweight Directory Access Protocol)
- Directory dapat berupa object store

Name resolution

- Untuk mengirimkan data ke service
  - Cari node dimana service tersebut berada
  - Cari alamat/network attachment point untuk node tersebut
  - Temukan path dari lokasi sekarang ke layanan tersebut

Binding

- Asosiasi nama ke alamat/objek
- Static binding
  - Hard coded
- Early binding
  - Lookup binding sebelum digunakan
  - Cache binding yang sebelumnya pernah digunakan
- Late binding
  - Lookup pas sebelum digunakan

Contoh Kasus: DNS

- Lihat materi DNS pada IF3130 (Jaringan Komputer)

Lookup

- Lookup(key, value)
- Kumpulan node bekerjasama menyediakan layanan
- Ideal:
  - Tidak ada koordinator sentral
  - Beberapa node dapat crash

Pendekatan

- Central coordinator
  - Napster
- Flooding
  - Gnutella
- Distributed Hash Tables
  - CAN, Chord, Amazon Dynamo, Tapestry,...

Central coordinator

- Contoh: Napster
- Central directory
  - Mencari content/names dan server yang menghost file tersebut
  - Lookup(name) => list of servers
  - Download dari any available server

- Contoh: GFS (Google File Systems)

Query Flooding

- Contoh: Gnutella
- Well-known node berperan sebagai anchor
  - Node yang memiliki file akan memberitahu anchor
  - Node memilih node lain sebagai peer
- Problem:
  - Penggunaan network tidak efisien (flooding)
  - Node kadang down dan dapat lebih lambat dibanding node lain

Distributed Hash Table

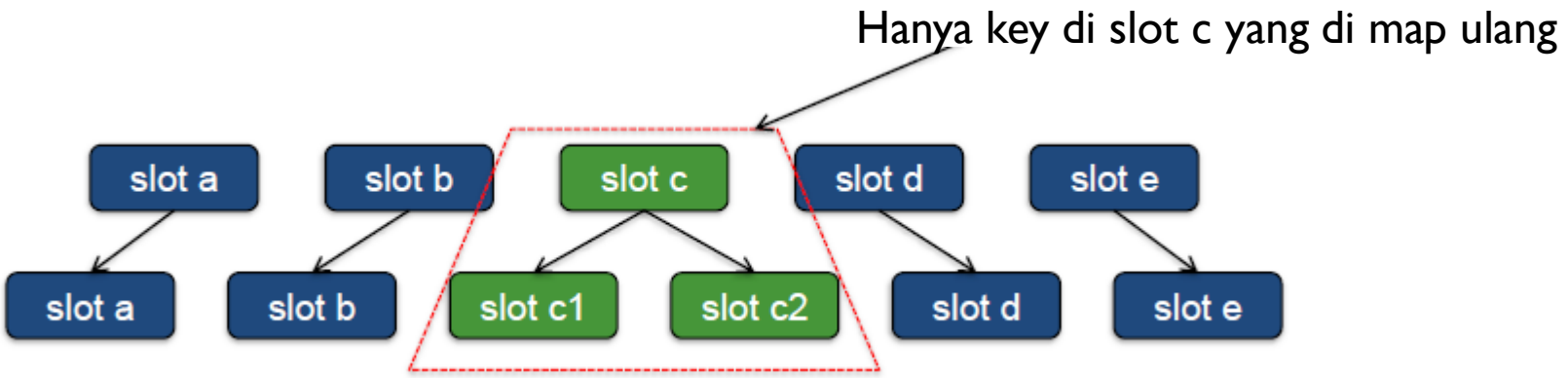
- Hash table: melakukan pencarian dengan O(1)
- Fungsi hash: fungsi yang menerima variable length input (e.g. String) dan menghasilkan fix-length result (e.g. Integer)
- Memilih fungsi hash
  - Ideal: uniform key distribution untuk semua nama yang mungkin
  - Minimal collision: 2 nama berbeda dipetakan ke key yang sama

Distributed Hash Table

- Versi peer-to-peer untuk database key-value
- Cara kerja:
  - Sebuah peer melakukan query ke database untuk sebuah key
  - Database menemukan peer mana yang memiliki key tersebut
  - Peer yang memiliki key tersebut akan mengembalikan <key, value> ke peer yang melakukan query
- Harus efisien, tidak boleh menghasilkan flood

Distributed Hash Table

- Bagaimana menangani penambahan node/slot:
- Consistent Hashing:
  - Most keys akan dipetakan ke node yang sama



Distributed Hash Table

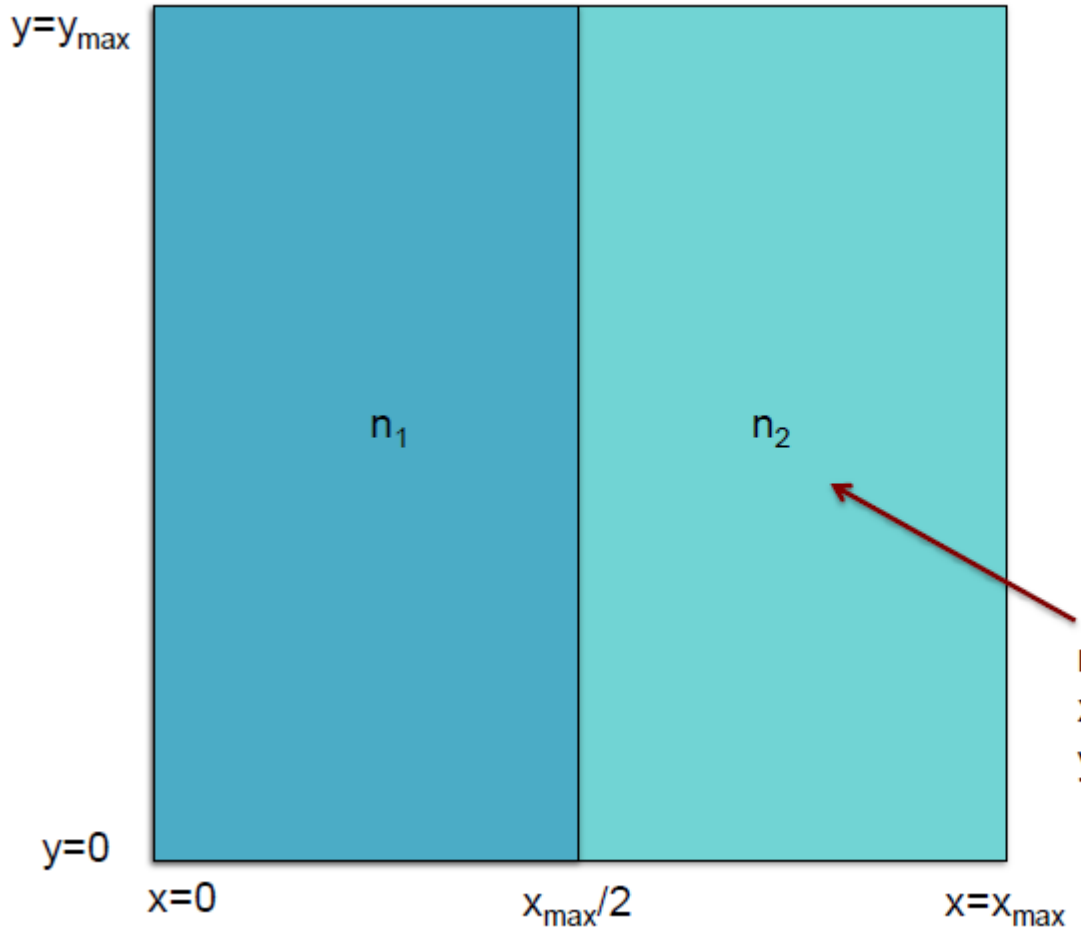
- Sebar hash table ke multiple node
- Setiap node menyimpan sebagian dari key space
- Lookup(key) => node ID yang menyimpan (key, value)

- Problem:
  - Bagaimana mempartisi data dan melakukan lookup?
  - Menjaga sistem tetap terdesentralisasi?
  - Membuat sistem skalabel?
  - Fault tolerant?

Contoh kasus: CAN – Content Addressable Network

- Buat logical grid (e.g. Menggunakan 2D: x-y)
- pisahkan hash function untuk setiap dimensi
  - h_x(key) dan h_y(key)
- Sebuah node
  - bertanggungjawab untuk rentang nilai tertentu pada kedua dimensi tadi
  - Mengetahui tetangganya

Contoh: CAN pada 2 node



$x = hash_x(key)$

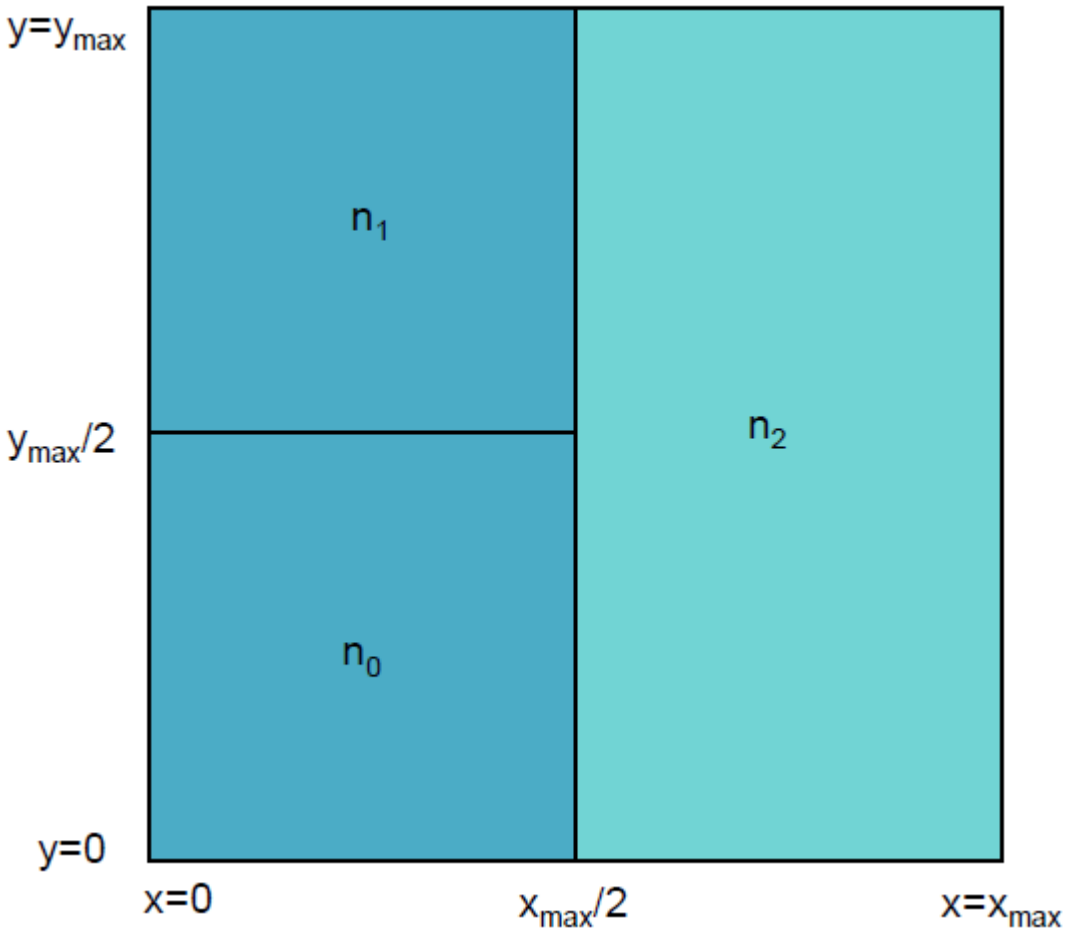
$y = hash_y(key)$

if $x < (x_{max}/2)$
n_1 has (*key, value*)

if $x \geq (x_{max}/2)$
n_2 has (*key, value*)

n_2 is responsible for a zone
$x=(x_{max}/2 .. x_{max})$,
$y=(0 .. y_{max})$

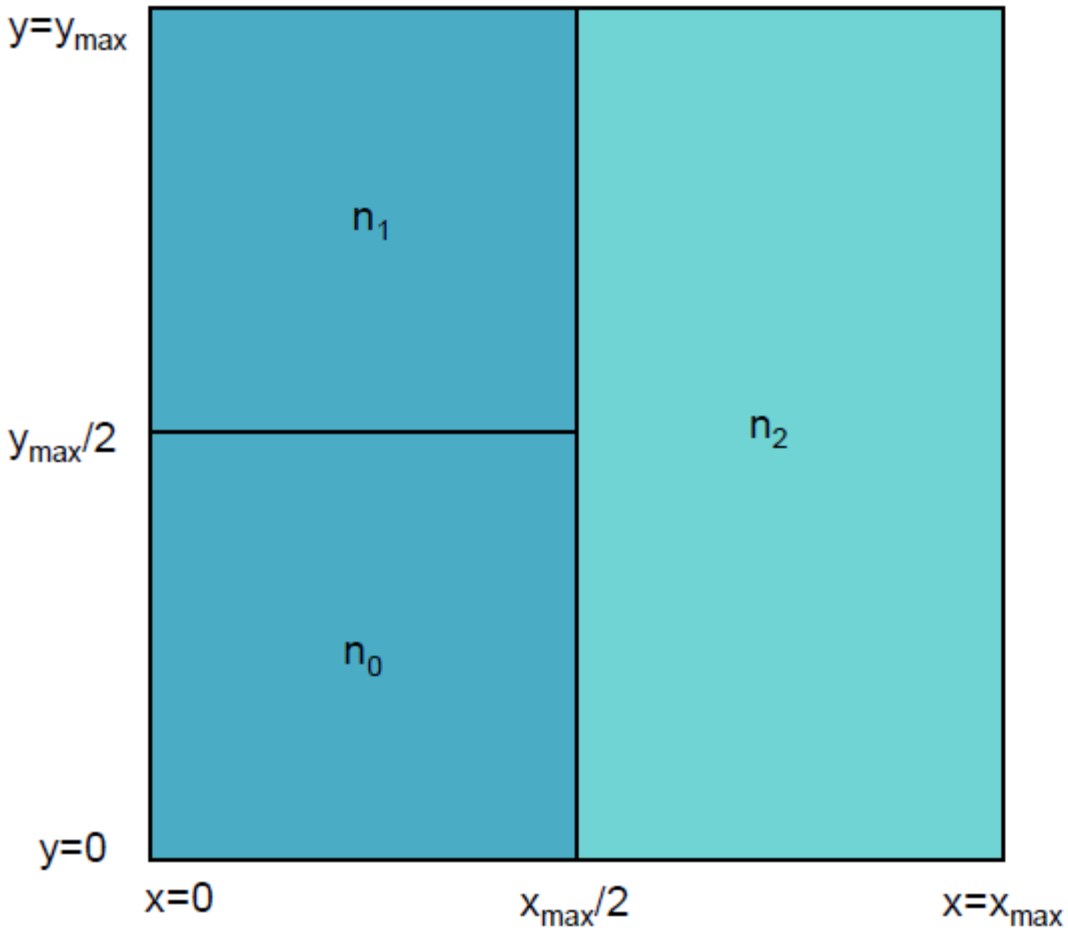
CAN partitioning



Setiap node dapat di-split menjadi 2 – vertikal maupun horizontal

Saat sebuah node ditambahkan, node yang baru dan yang lama akan saling mengetahui rentang nilai masing2

CAN key -> node mapping

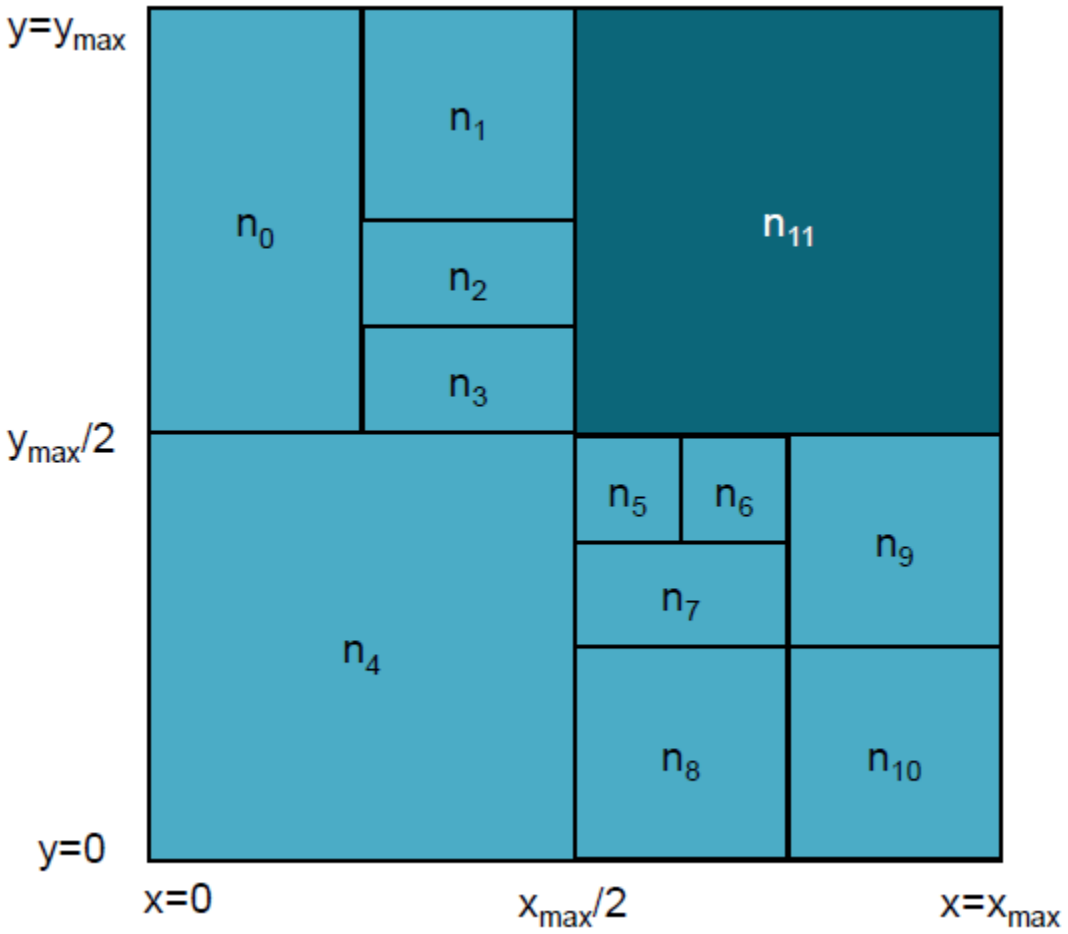


$x = hash_x(key)$

$y = hash_y(key)$

if $x < (x_{max}/2)$ {
  if $y < (y_{max}/2)$
    n_0 has (*key, value*)
  else
    n_1 has (*key, value*)
}

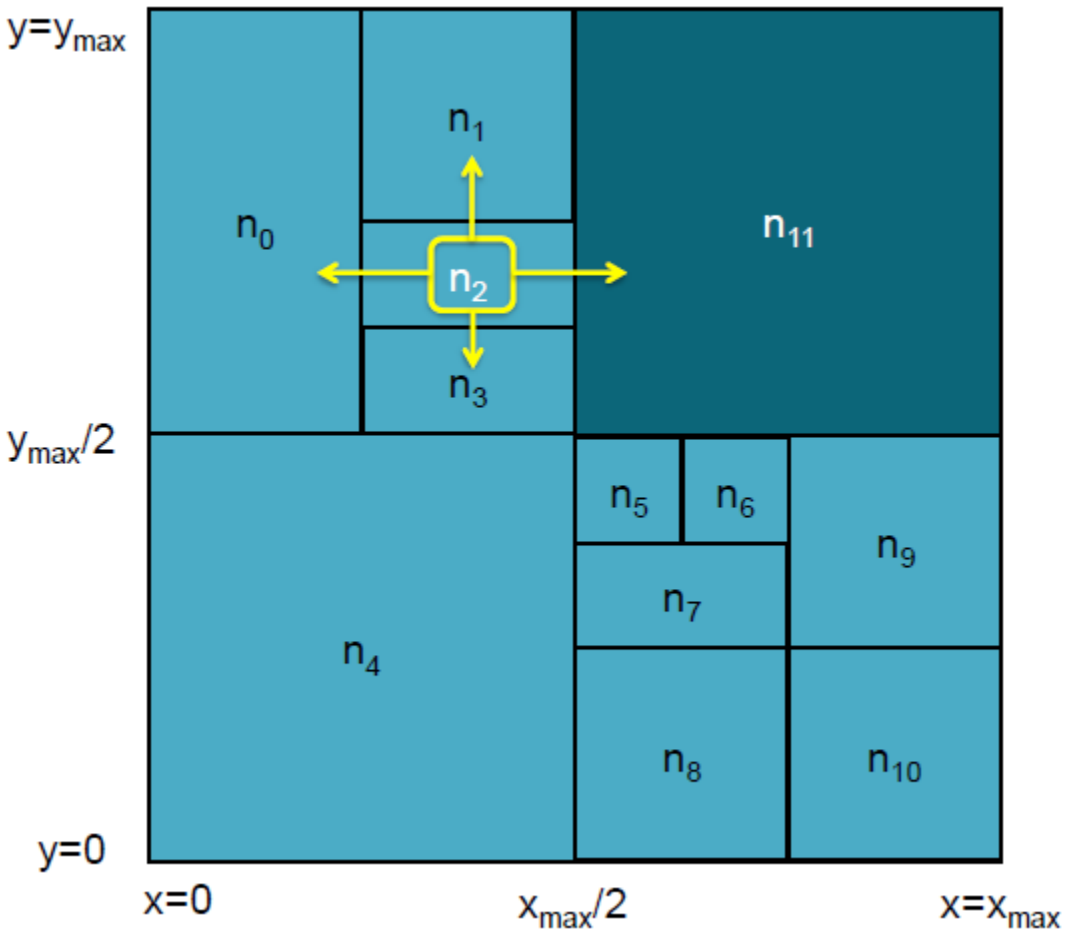
if $x \geq (x_{max}/2)$
    n_2 has (*key, value*)



- Setiap node ditambahkan, data terkait harus dipindahkan

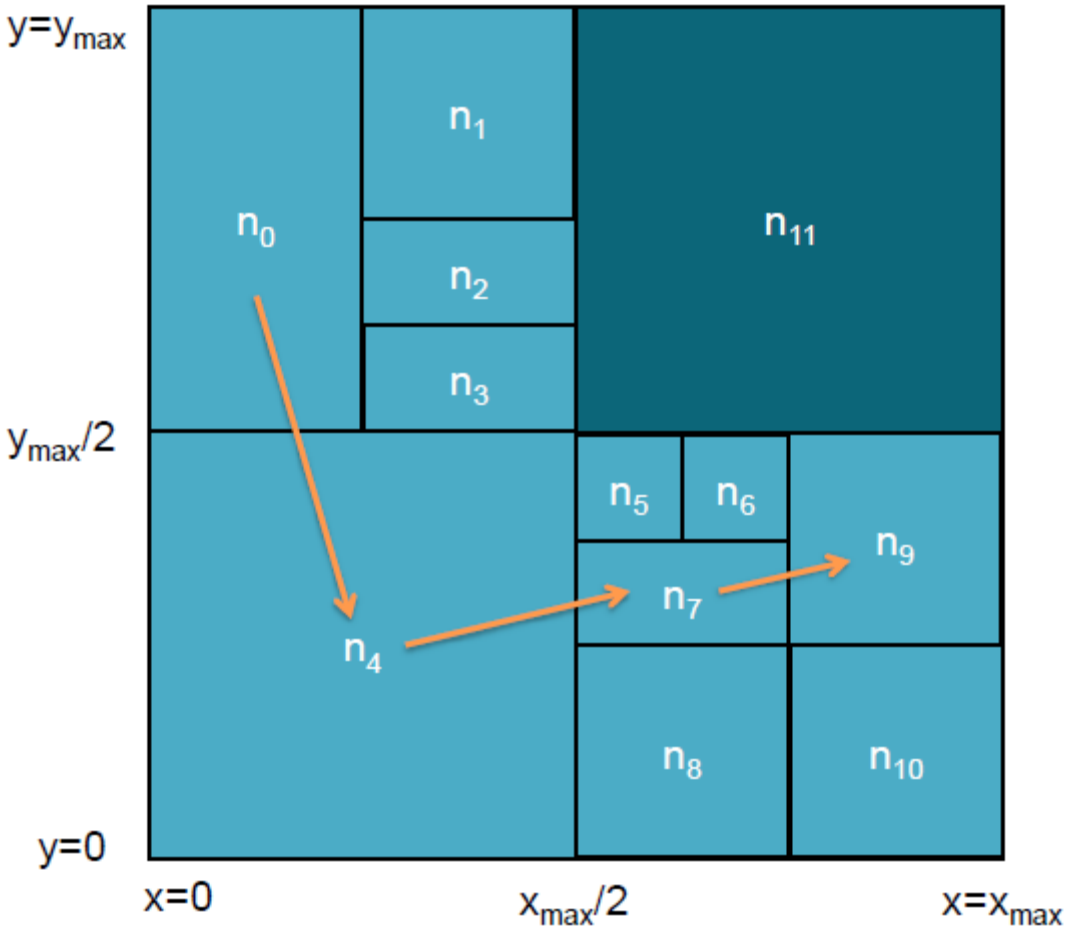
- Neighbor harus diberitahu tentang node baru

- Sebuah node hanya mengetahui tetangganya



Tetangga: node yang memiliki zone bersebelahan

CAN routing



Lookup(key) pada sebuah node dilakukan dengan menghitung hash untuk masing2 dimensi (x-y), dan route request ke tetangga

Ideal: route yang meminimalkan jarak ke tujuan

Distributed Hash Tables (DHT)

Chord

Consider the organization of many nodes into a logical ring

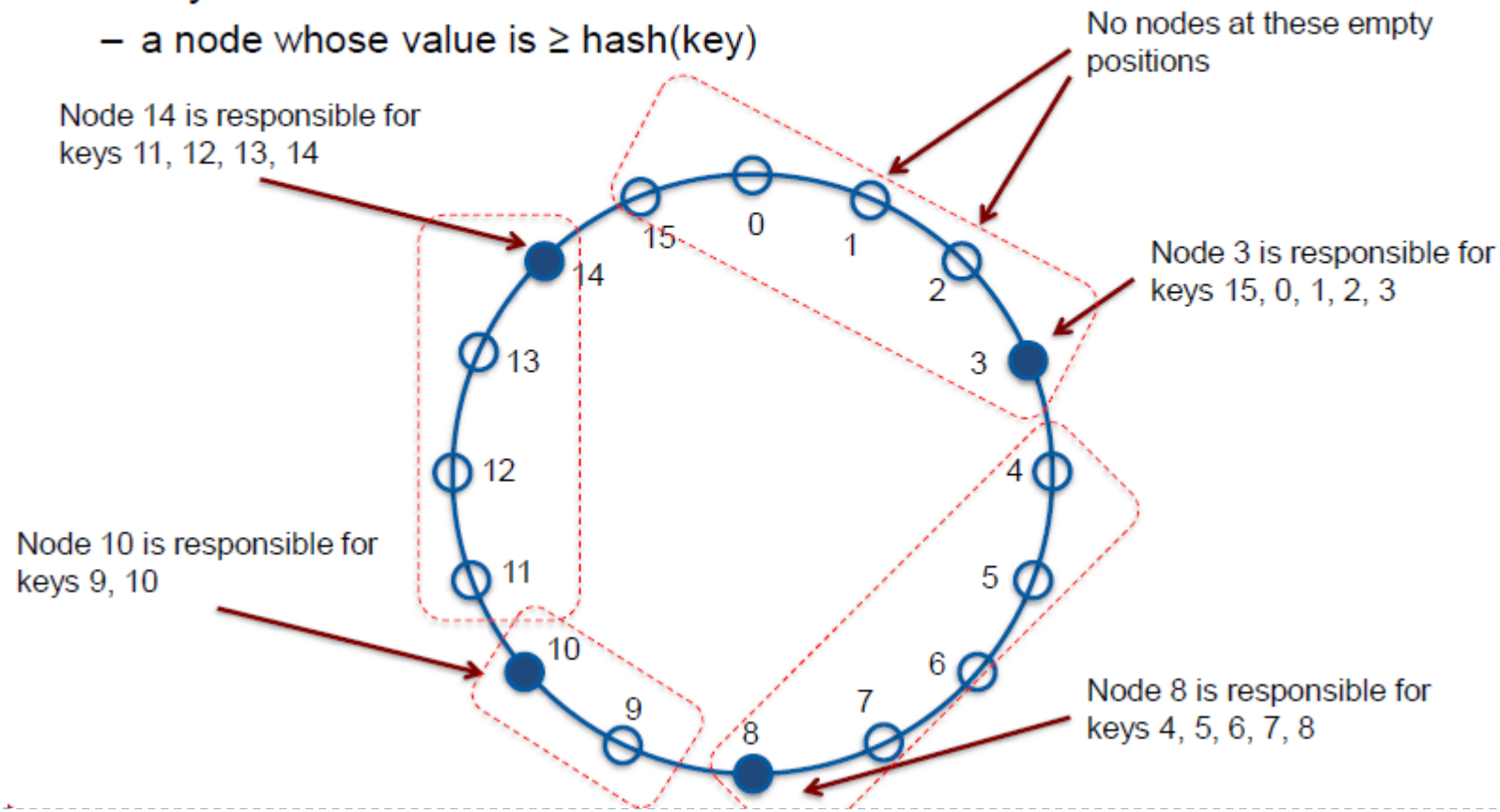
- Each node is assigned a random m-bit identifier.
- Every entity is assigned a unique m-bit key.
- Entity with key k falls under jurisdiction of node with smallest $id \geq k$ (called its successor).

Nonsolution

Let node id keep track of $succ(id)$ and start linear search along the ring.

Contoh

- Example: *n=16;* system with 4 nodes (so far)
- A key is stored at a successor
  - a node whose value is ≥ hash(key)



Chord

- Untuk menangani request, cara yang sederhana adalah mengatur agar setiap node mencatat siapa node successor nya.
- Saat sebuah peer menerima request, dia akan memeriksa apakah key yang dicari berada dalam rentang yang dikelola node tersebut atau tidak. Jika tidak, maka request akan diforward ke successor nya
  - Worst case: untuk ring dengan p node, traversal dilakukan pada p-1 node
  - Average case: p/2 node

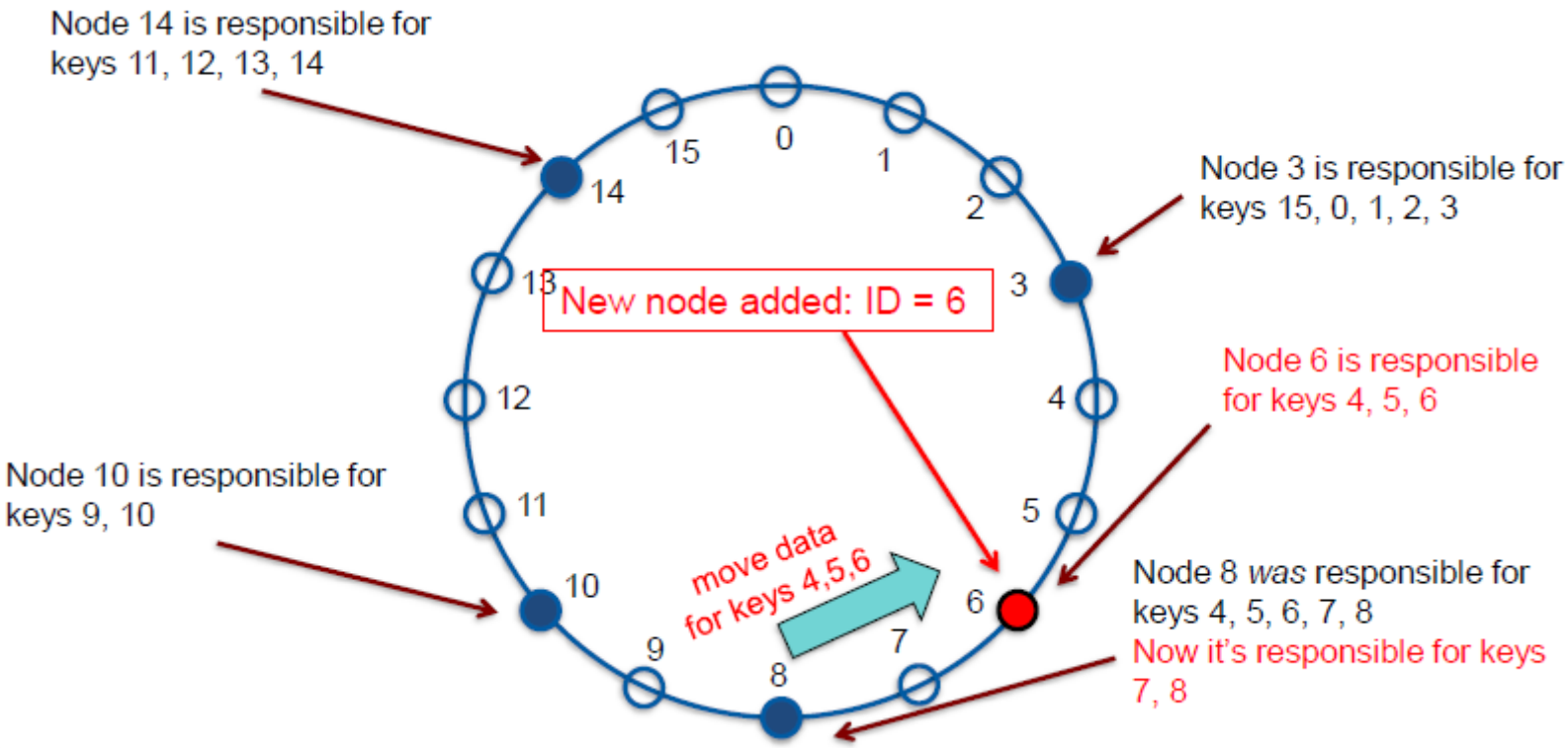
Chord

- Apa yang harus dilakukan untuk:

- Penambahan/penghapusan node
- Improving lookup time
- Fault tolerance

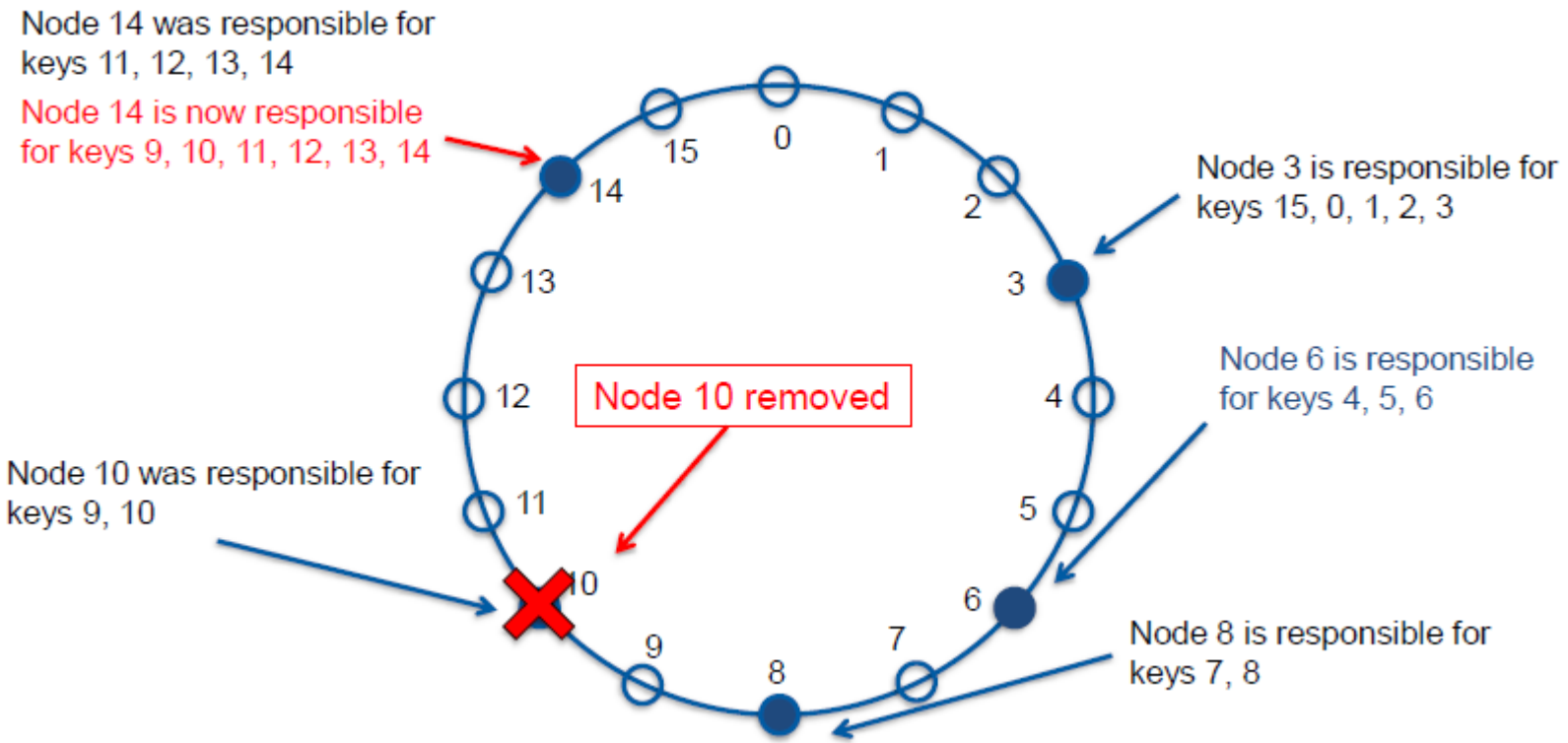
Penambahan node

- Beberapa key yang di-assign ke successor sebuah node dialihkan ke node baru
- Data terkait (key,value) harus dipindahkan ke node baru



Penghapusan node

- Key di re-assign ke node's successor
- Data terkait dipindahkan



Fault tolerance

- Node dapat crash/mati
  - (key, value) harus di-replikasi
  - Buat R- replika, dan dan disimpan pada r-1 successor pada ring

- Problem
  - Node harus mengetahui successor nya successor
    - Mudah jika semua node diketahui
  - Saat sebuah node kembali up, harus mencek successor jika ada update baru
  - Setiap perubahan harus di-propagasi ke semua replika

Chord Performance

- O(n) lookup => tidak bagus

- Cara sederhana:
  - Semua node saling mengetahui satu sama lain
  - Lookup untuk mencari node yang menyimpan data dapat dilakukan dengan O(1)
  - Setiap perubahan node harus diberitahukan ke semua node
  - Tidak tepat jika jumlah node sangat besar

Finger table

- Solusi: kompromi untuk tidak menggunakan tabel besar pada setiap node, gunakan finger table yang berukuran m entries
- Finger table (FT) berisi partial list dari node
- Pada setiap node, entri ke i pada FT menunjukkan node yang menjadi successor paling tidak 2^{i-1} pada ring
  - FT[0]: suksesor langsung
  - FT[1]: suksesor ke dua
  - FT[2]: suksesor ke 4
  - FT[3]: suksesor ke 8
- Kompleksitas lookup: O(log n)

DHTs: Finger Tables

Principle

- Each node p maintains a finger table $FT_p[]$ with at most m entries:

$$FT_p[i] = succ(p + 2^{i-1})$$

  Note: $FT_p[i]$ points to the first node succeeding p by at least 2^{i-1}.
- To look up a key k, node p forwards the request to node with index j satisfying

$$q = FT_p[j] \leq k < FT_p[j+1]$$

- If $p < k < FT_p[1]$, the request is also forwarded to $FT_p[1]$

DHTs: Finger Tables