Sistem Terdistribusi

TIK-604

Penamaan (*Naming*)

Kuliah 05: 11 s.d 13 Maret 2019

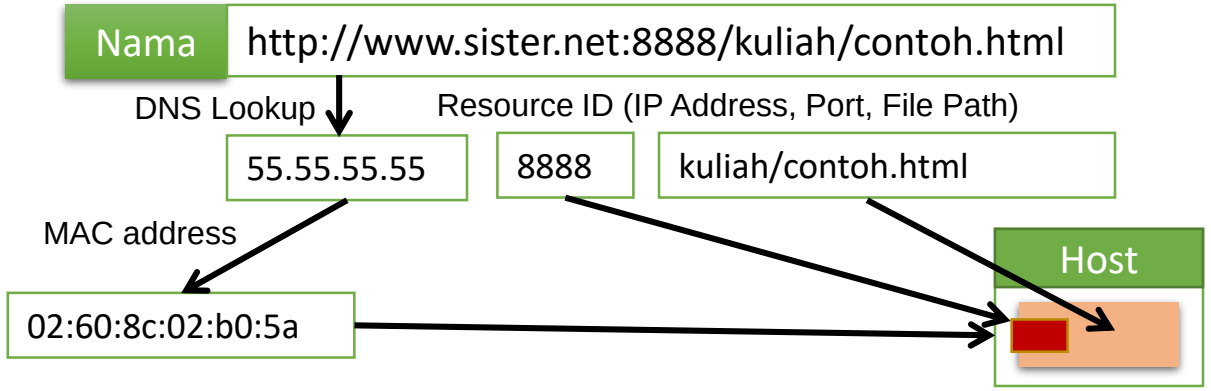
Husni

Hari ini...

- Kuliah sebelumnya:
  - Arsitektur
- Bahasan hari ini:
  - Penamaan (*Naming*)
- Pengumuman:
  - Proyek?
  - Tugas Kelompok!

Penamaan

- Nama digunakan untuk secara unik mengenali entitas di dalam sistem terdistribusi
  - Entitas dapat berupa proses, obyek jauh (*remote*), *newsgroups*, dll.,
- Nama dipetakan ke lokasi entitas menggunakan *name resolution*
- Contoh resolusi nama:



Nama, Alamat & Pengenal

- Suatu entitas dapat dikenali dengan tiga jenis referensi:
  a) Nama
     - Nama adalah sehimpunan bit atau karakter yang mereferensi/mengacu suatu entitas
     - Nama dapat bersifat *human-friendly* (atau tidak)
  b) Alamat (*Address*)
     - Setiap entitas terletak pada suatu *access point* (titik akses), dan *access point* itu mempunyai alamat (*address*)
     - Address dapat bersifat *location-dependent* (atau tidak)
     - Contoh: IP Address + Port
  c) Pengenal (*Identifier*)
     - Identifier adalah nama yang secara unik mengenali entitas
     - Suatu *identifier sejati* adalah nama dengan properti-properti berikut:
       - Suatu *identifier* mengacu ke paling banyak satu entitas
       - Setiap entitas dirujuk oleh paling banyak satu *identifier*
       - Suatu identifier selalu merujuk ke entitas yang sama (tidak pernah digunakan-ulang)

Sistem Penamaan

- *Sistem Penamaan* sederhananya merupakan suatu *middleware* yang membantu urusan resolusi nama

- Sistem penamaan dapat dikategorikan ke dalam tiga kelas, berdasarkan pada cara pemberian namanya:
  a. Penamaan *Flat*
  b. Penamaan Terstruktur
  c. Penamaan berbasis Atribut.

Kelas-kelas Penamaan

- Penamaan *flat*
- Penamaan terstruktur
- Penamaan berbasis atribut

Penamaan *Flat*

- Dalam penamaan *flat*, *identifiers* merupakan bit-bit string yang acak (dikenal sebagai nama *flat* atau tak-terstruktur)

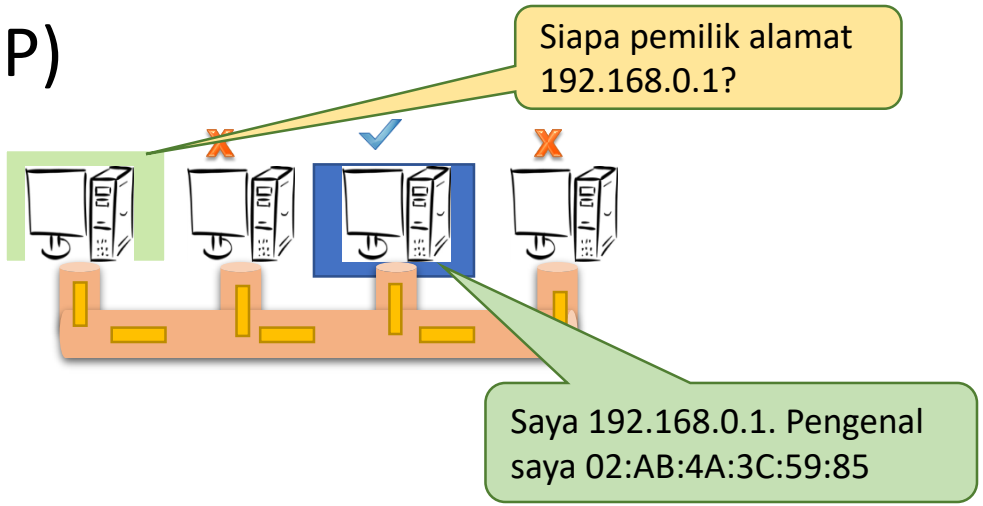
- Nama *flat* tidak mengandung informasi apapun mengenai bagaimana menemukan suatu entitas

- Akan dipelajari empat jenis mekanisme resolusi nama untuk nama *flat*:
  1. *Broadcasting*
  2. *Forwarding pointers*
  3. Pendekatan berbasis *Home*
  4. *Distributed Hash Tables (DHTs)*

1. *Broadcasting*

- Pendekatan: *Broadcast*kan nama/alamat ke seluruh jaringan; Entitas yang berasosiasi dengan nama tersebut memberikan respon berupa pengenal mutakhirnya

- Contoh: ***Address Resolution Protocol*** (ARP)
  - Mencarikan *IP address* ke *MAC address*-nya
  - Dalam sistem ini,
    - IP address adalah alamat dari entitas
    - MAC address adalah *identifier* dari *access point*-nya



- Tantangan:
  - Tidak *scalable* dalam jaringan besar
    - Teknik ini mengakibatkan jaringan banjir dengan **message broadcast**
  - Mengharuskan semua entitas untuk mendengarkan (*atau mengintai*) semua *request*

2. Forwarding Pointers

- *Forwarding pointers* memungkinkan penemuan entitas *mobile*
  - Entitas *mobile* bergerak dari satu *access point* ke *point* lainnya

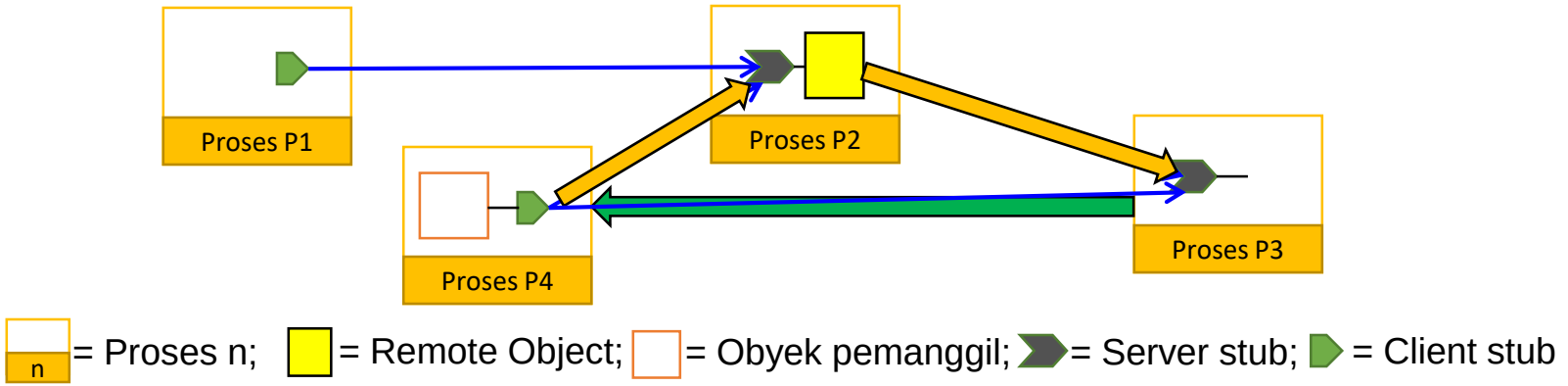
- Saat suatu entitas bergerak dari lokasi A ke lokasi B, ia meninggalkan di belakang (di A) suatu referensi ke lokasi barunya di B
- Mekanisme resolusi nama:
  - Mengikuti *chain of pointers* untuk menjangkau entitas tersebut
  - Mengupdate referensi entitas ketika lokasi terkini ditemukan.

- Tantangan:
  - Rantai panjang mengakibatkan delay resolusi lebih panjang pula
  - Rantai panjang mudah gagal dikarenakan rusaknya *link-link*.

Forwarding Pointer: Contoh

- Rantai *Stub-Scion Pair* (SSP) menerapkan *remote invocations* bagi entitas *mobile* menggunakan *forwarding pointers*
  - Server stub dirujuk sebagai *Scion* dalam paper aslinya
- Setiap *forwarding pointer* diimplementasikan sebagai suatu pasangan:

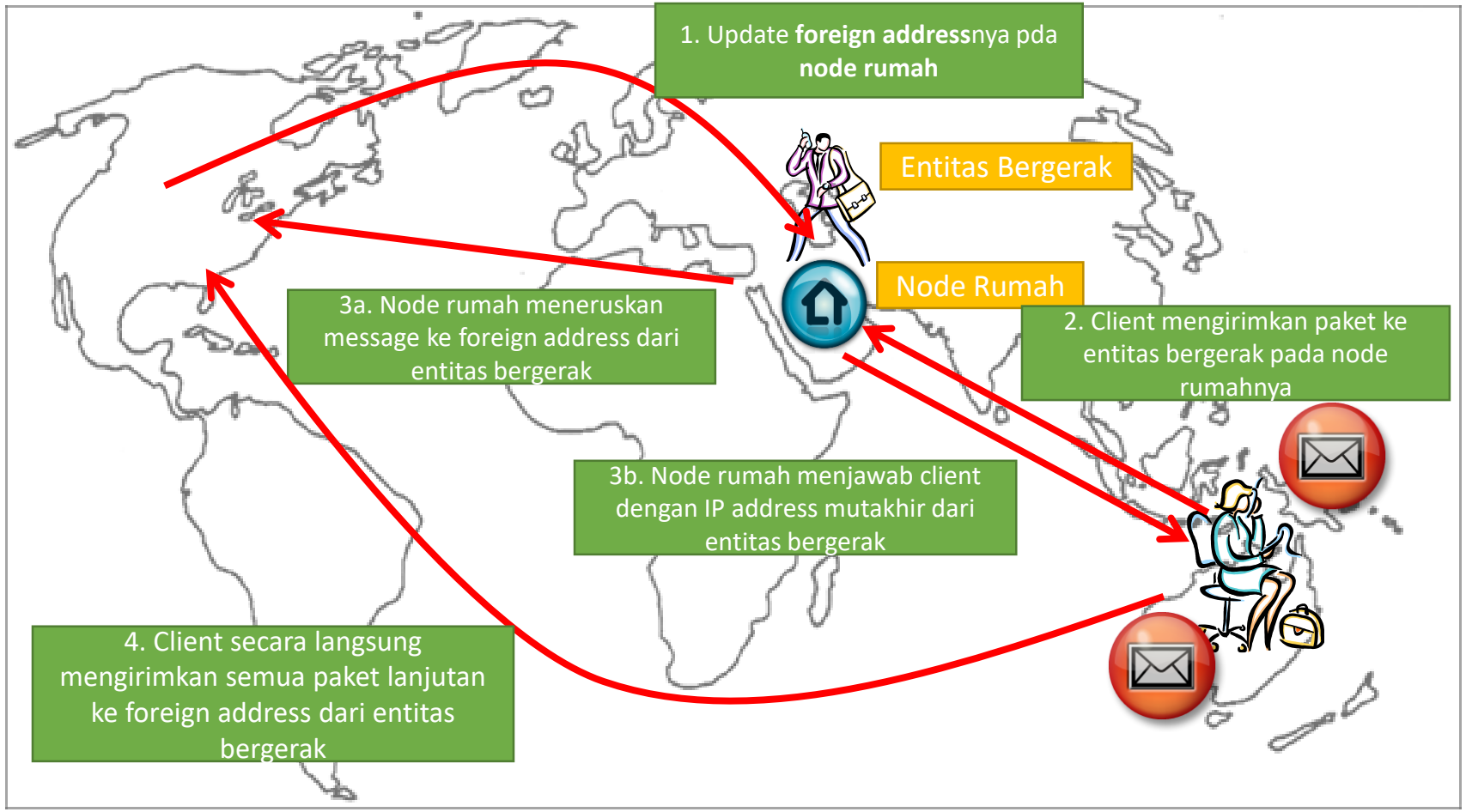
  `(client stub, server stub)`
  - Server stub berisi referensi lokal ke obyek aktual atau client stub lain
- Ketika obyek berpindah dari A (misalnya P2) ke B (misalnya P3),
  - Ia meninggalkan suatu *client stub* di A (yaitu P2)
  - Ia menginstall suatu *server stub* di B (yaitu P3)



3. Pendekatan *Home-Based*

- Setiap entitas diberikan suatu node *home*
  - Node home bersifat *static* (mempunyai *access point* dan *address* tetap)
  - Ia memelihara *track* menuju alamat terkini dari entitas
- Interaksi entitas-home:
  - Alamat rumah dari entitas didaftarkan pada suatu *naming service*
  - Entitas itu mengupdate *home* berkenaan dengan alamat terkininya (*foreign address*) kapanpun ia bergerak
- Resolusi nama:
  - Client menghubungi *home* untuk mendapatkan *foreign address*-nya
  - Client kemudian menghubungi entitas pada *foreign location* tersebut.

3. Pendekatan Berbasis *Home*: Contoh



3. Pendekatan Berbasis *Home*: Tantangan

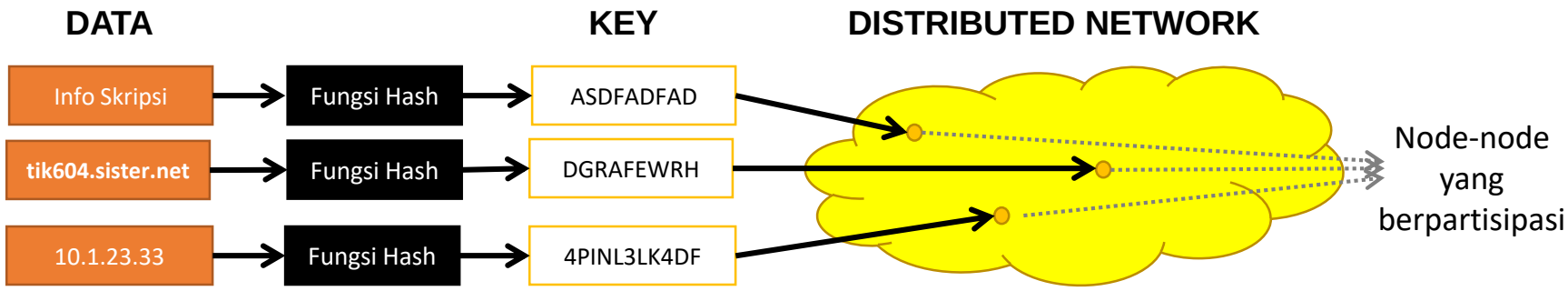
- Alamat rumah statis bersifat permanen selama masa hidup entitas
  - Jika entitas dipindahkan secara permanen, maka pendekatan berbasis *home* sederhana ini menghadirkan biaya komunikasi yang lebih tinggi

- Biaya setup koneksi dikarenakan komunikasi antara client dan rumahnya dapat menjadi mahal
  - Pikirkan skenario dimana *clients* lebih dekat ke entitas *mobile* daripada entitas *home*.

4. *Distributed Hash Table* (DHT)

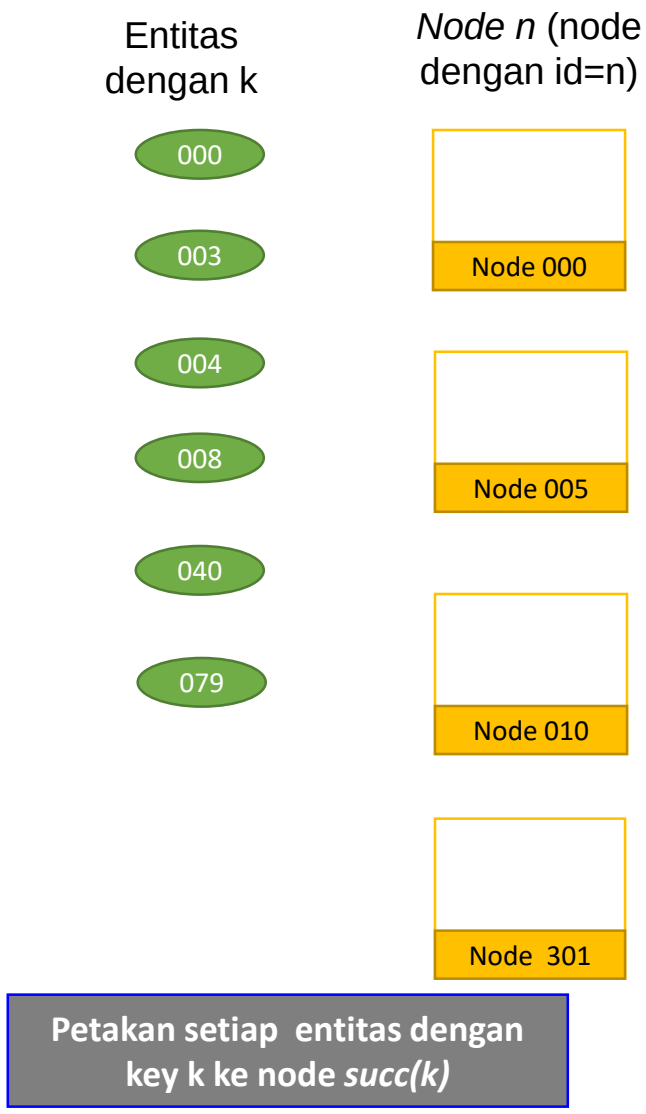
- DHT merupakan suatu sistem terdistribusi yang menyediakan layanan *lookup* serupa dengan *hash table*
  - Pasangan *(key, value)* disimpan dalam node-node yang berpartisipasi dalam DHT
  - Tanggungjawab untuk memelihara pemetaan dari *key* ke *value* didistribusikan antar node-node tersebut
  - Node apapun yang berpartisipasi dapat melayani pengambilan nilai untuk suatu *key* yang diberikan

- Akan didiskusikan DHT representatif dikenal sebagai *Chord*



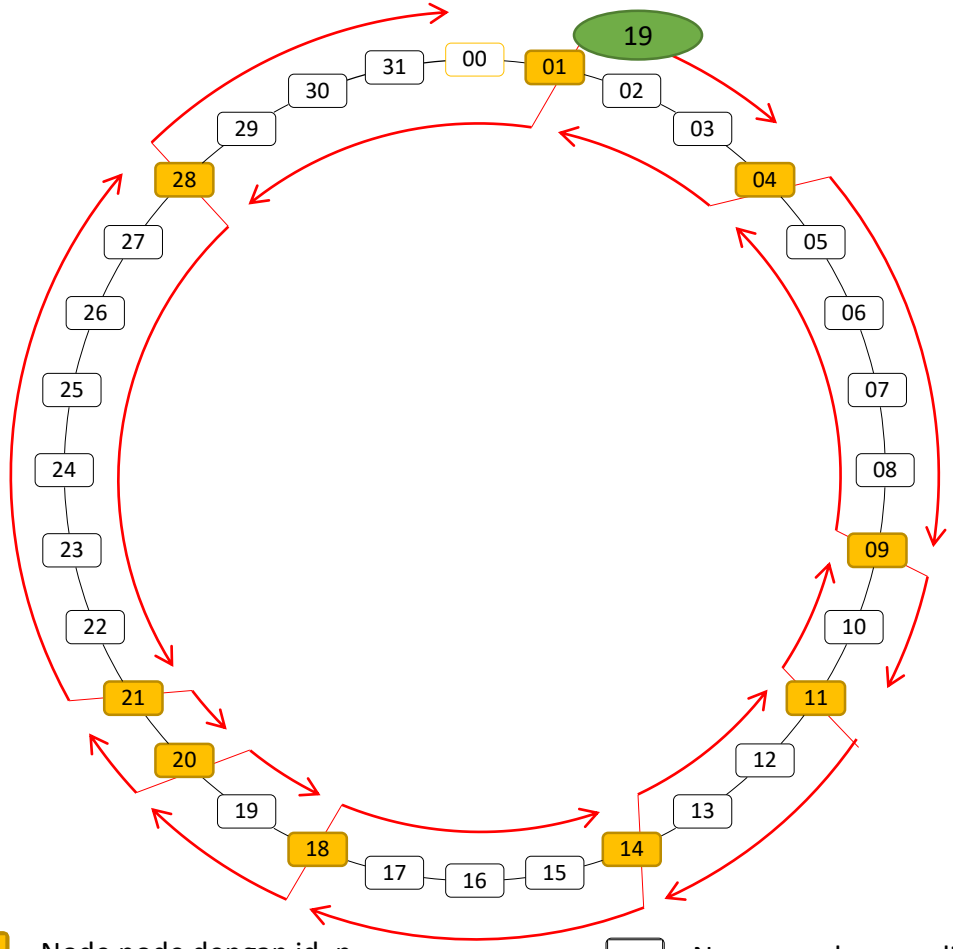
Chord

- Chord menyematkan suatu *m-bit identifier* (dipilih secara acak) kepada setiap node
  - Suatu node dapat dihubungi melalui alamat jaringannya
- Selain itu, ia memetakan setiap entitas ke suatu node
  - Entitas dapat berupa proses, file, dll.,
- Pemetaan entitas ke node
  - Setiap node bertanggungjawab untuk sehimpunan entitas
  - Suatu entitas dengan *key k* jatuh di bawah dari node dengan pengenal terkecil *id >= k*. Node ini dikenal sebagai *successor* dari *k*, dan dinotasikan dengan *succ(k)*



Algoritma Resolusi Kunci Naïve

- Isu utama dalam DHT adalah efisiensi pemetaan suatu key *k* ke lokasi jaringan dari *succ(k)*
  - Diberikan suatu entitas dengan *key* ***k***, bagaimana mencari ***node succ(k)***?



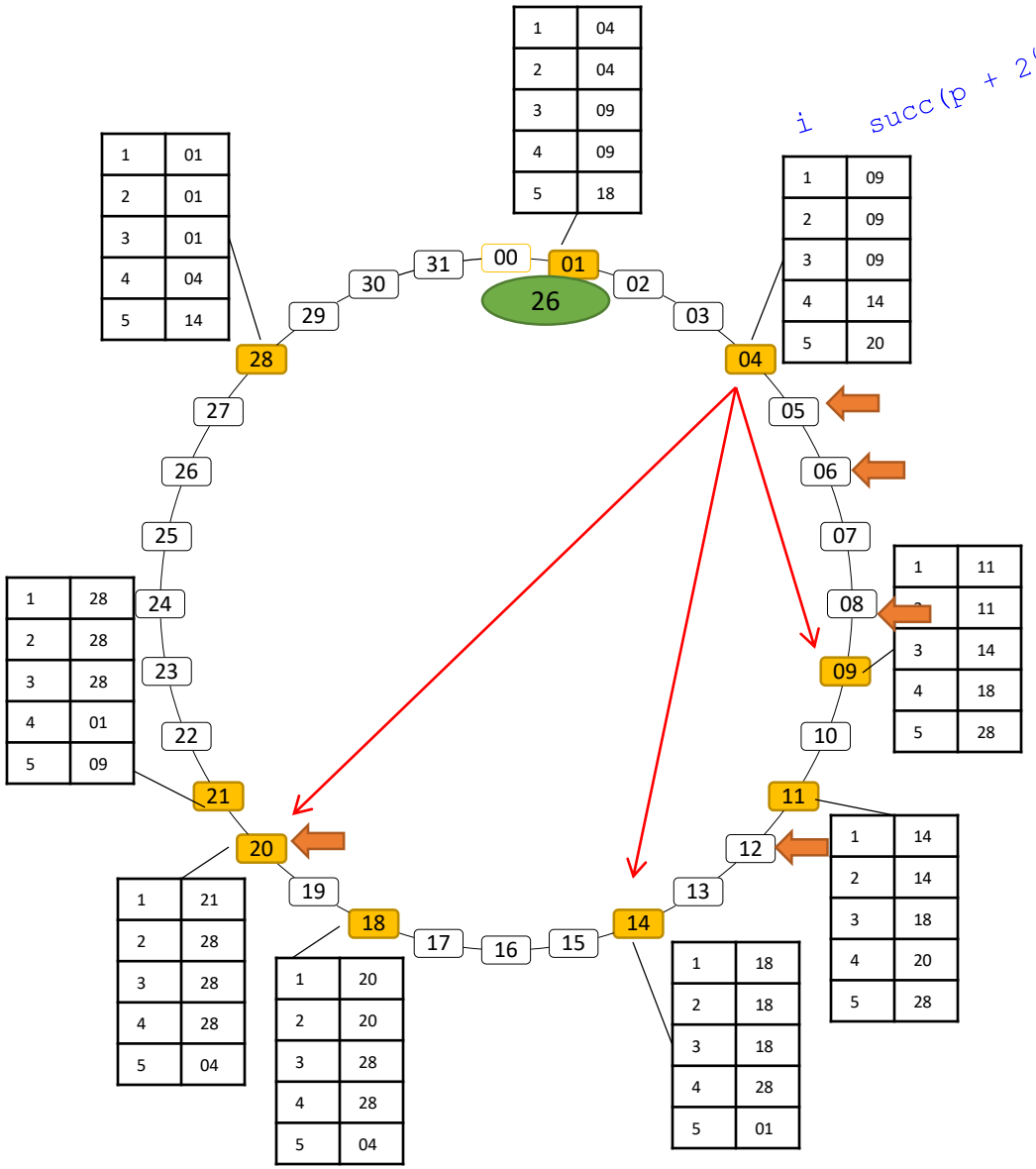
1. Semua node disusun dalam suatu *logical ring* sesuai dengan ID mereka
2. Setiap node ‘p’ memegang *track* dari tetangga langsungnya: *succ(p)* dan *pred(p)*
3. Jika ‘p’ menerima request untuk me*resolve* key ‘k’:
   - Jika *pred(p) < k <=p*, node p akan menanganinya
   - Jika tidak, akan diteruskan ke *succ(n)* atau *pred(n)*

Solusi tidak *scalable*:
- Sejalan pertumbuhan jaringan, terjadi kenaikan delay *forwarding*
  - Resolusi *key* kompleksitas waktunya O(n)

n = Node node dengan id=n

p = Nomor node yang diberikan ke key p

Resolusi Kunci dalam Chord



- Chord memperbaiki resolusi key dengan mengurangi kompleksitas waktu menjadi O(log n)

1. Semua node disusun dalam suatu *logical ring* sesuai dengan ID-nya
2. Setiap node 'p' mempunyai suatu tabel FT_p dari paling banyak *m* entri. Tabel ini dinamakan Finger Table.

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

   Catatan: $FT_p[i]$ bertambah secara eksponensial

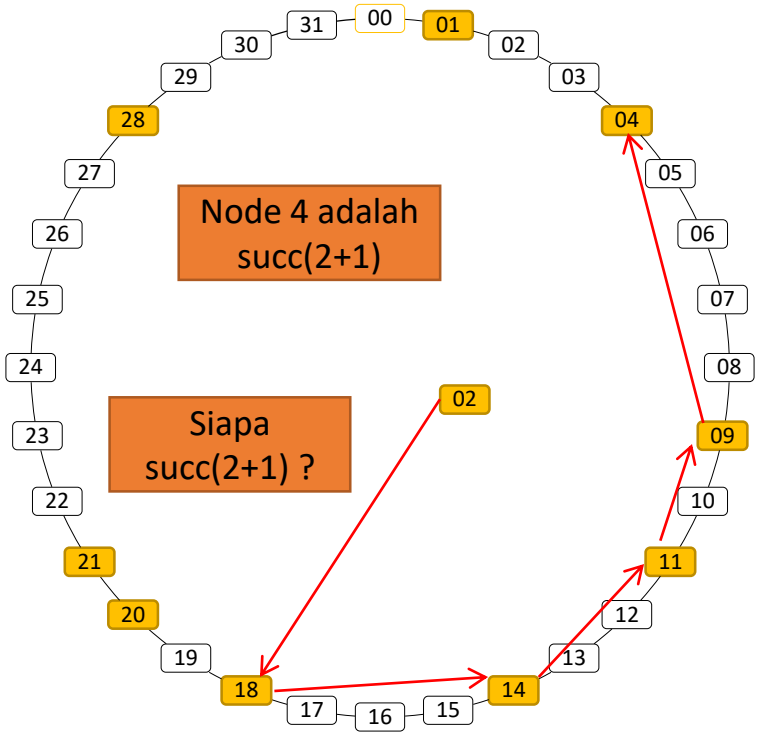
3. Jika node 'p' menerima request untuk meresolve key 'k':
   - Node p akan meneruskannya ke node q dengan index j dalam F_p dimana

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

     - Jika $k > FT_p[m]$, maka node p akan meneruskannya ke $FT_p[m]$
     - Jika $k < FT_p[1]$, maka node p akan meneruskannya ke $FT_p[1]$

Chord: Protokol Join & Leave

- Dalam sistem terdistribusi skala besar, node-node secara dinamis *join* dan *leave* (secara suka rela atau karena kegagalan)
- Jika suatu node p ingin bergabung (*join*):
  - Ia menghubungi *arbitrary node*, mencarikan succ(p+1), dan menyisipkan dirinya ke dalam ring tersebut
- Jika node p ingin meninggalkan (*leave*):
  - Ia menghubungi *pred(p)* dan *succ(p+1)* dan mengupdatenya.



Chord: Protokol Update *Finger Table*

- Untuk suatu node q, $FT_q[1]$ harus *up-to-date*
  - Ia merujuk ke *next node* dalam ring tersebut
  - Protokol:
    - Secara berkala, request *succ(q+1)* untuk return *pred(succ(q+1))*
    - Jika *q = pred(succ(q+1))*, maka informasinya up-to-date
    - Jika tidak, suatu node baru p telah ditambahkan ke ring sehingga *q < p < succ(q+1)*
      - $FT_q[1] = p$
      - Request p untuk update *pred(p) = q*
    - Dengan cara serupa, node p mengupdate setiap entri i dengan mencari

      $succ(p + 2^{(i-1)})$

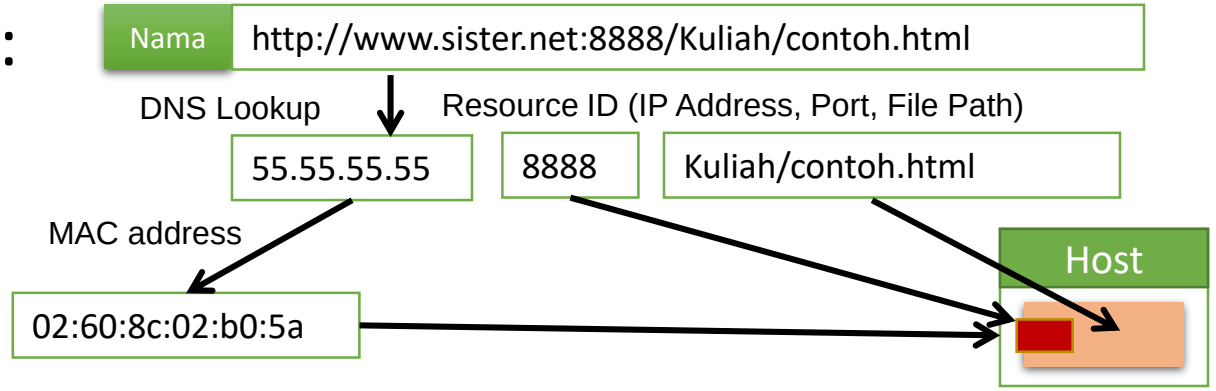
Kedekatan Jaringan dalam Chord

- Organisasi logis dari node-node dalam jaringan *overlay* dapat mengakibatkan ketidak-efisienan transfer *message*
  - Node *k* dan node *succ(k +1)* mungkin terpisah jauh

- Chord dapat dioptimalkan dengan mempertimbangkan lokasi jaringan dari node-node
  1. Penugasan Node Sadar Topologi
     - Dua node berdekatan memperoleh identifiers yang saling berdekatan
  2. Routing kedekatan
     - Setiap node *q* memelihara '*r*' suksesor untuk entri ke-i dalam *finger table*
     - $FT_q[i]$ sekarang merujuk ke sebanyak r node suksesor di dalam rentang $[p + 2^{(i-1)}, p + 2^i - 1]$
     - Untuk meneruskan request *lookup*, ambil satu dari r suksesor terdekat ke node *q* tersebut.

Penamaan

- Nama digunakan untuk secara unik mengidentifikasi entitas-entitas di dalam sistem terdistribusi
  - Entitas dapat berupa proses, *remote objects*, *newsgroups*, dll.,
- Nama dipetakan ke lokasi entitas menggunakan resolusi (*name resolution*)
- Contoh resolusi nama:



Kategori Pemanaan

- Penamaan *Flat*
- Penamaan Terstruktur
- Penamaan Berbasis Atribut

Penamaan Terstruktur

- Nama terstruktur tersusun dari nama-nama *human-readable* sederhana
  - Nama-nama diatur dalam suatu struktur tertentu

- Contoh:
  - File-systems menggunakan nama terstruktur untuk mengidentifikasi file-file
    - /home/userid/work/dist-systems/naming.txt

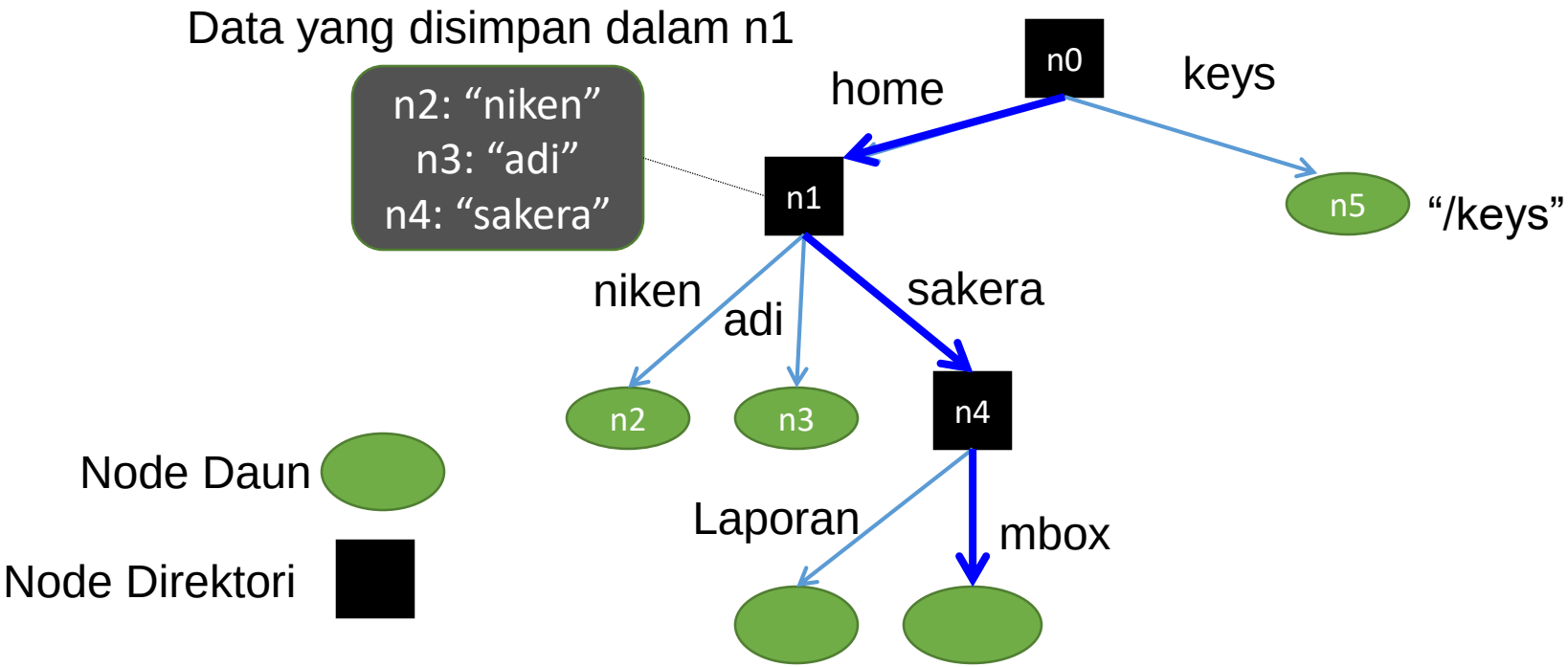
  - Websites dapat diakses melalui nama terstruktur
    - Husni.trunojoyo.ac.id

Ruang Nama

- Nama terstruktur diorganisir dalam ruang nama (*name spaces*)
- *Name space* adalah suatu *directed graph* yang terdiri dari:
  - Node daun (*leaf*)
    - Setiap node daun merepresentasikan suatu entitas
    - Node daun biasanya menyimpan *address* dari suatu entitas (misalnya dalam DNS), atau *state* dari (atau *path* menuju) suatu entitas (misalnya di dalam *file systems*)
  - Node direktori
    - Node direktori merujuk ke node leaf atau direktori lain
    - Setiap *outgoing edge* diwakili oleh (*edge label, node identifier*)
- Setiap node dapat menyimpan tipe data tertentu
  - Yaitu State dan/atau address (*misalnya untuk suatu mesin*) dan/atau path.

Ruang Nama: Contoh

Mencari entitas dengan nama “/home/sakera/mbox”



Resolusi Name

- Proses pencarian nama disebut resolusi (*name resolution*)
- Mekanisme pengakhiran (*closure*):
  - Resolusi nama tidak dapat diselesaikan tanpa suatu *initial directory node*
  - Mekanisme ***closure*** memilih konteks implisit dari mana memulai resolusi nama
  - Contoh:
    - husni.trunojoyo.ac.id: mulai pada **DNS Server**
    - /home/sakera/mbox: mulai pada ***root*** dari file-system

Pengaitan Nama

- Ruang nama dapat secara efektif digunakan untuk mengaitkan (menghubungkan, membuat *link*) dua entitas berbeda

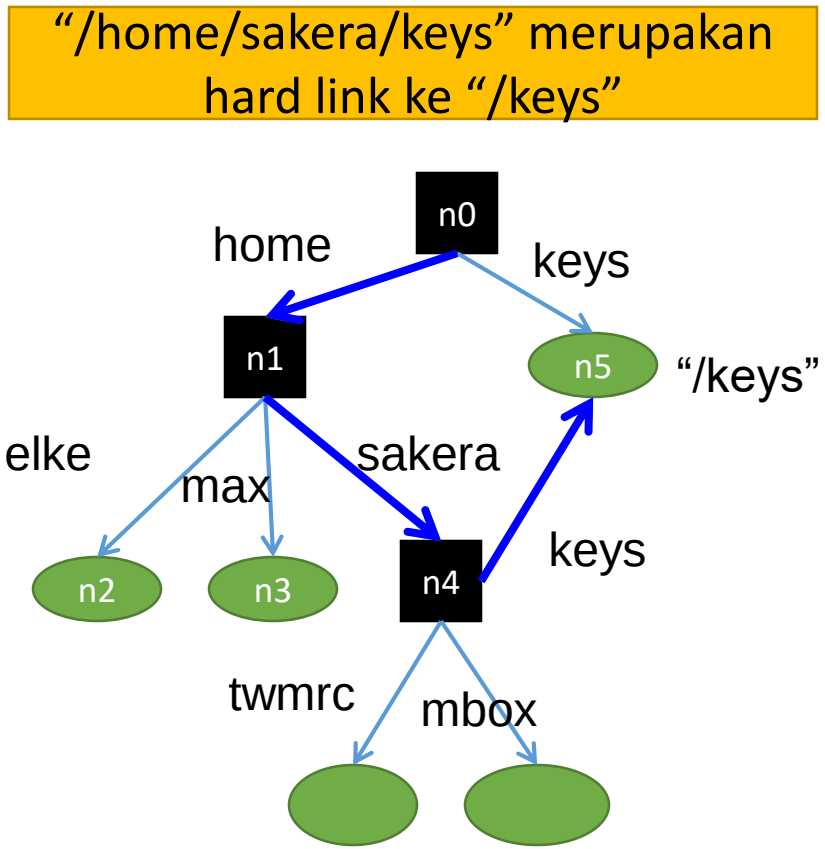
- Dua jenis link dapat hadir antara node-node:
  1. *Hard Links*
  2. *Symbolic Links*

1. Hard Links

- Ada suatu *directed link* dari *hard link* (nama link) ke *actual node* (node sebenarnya)

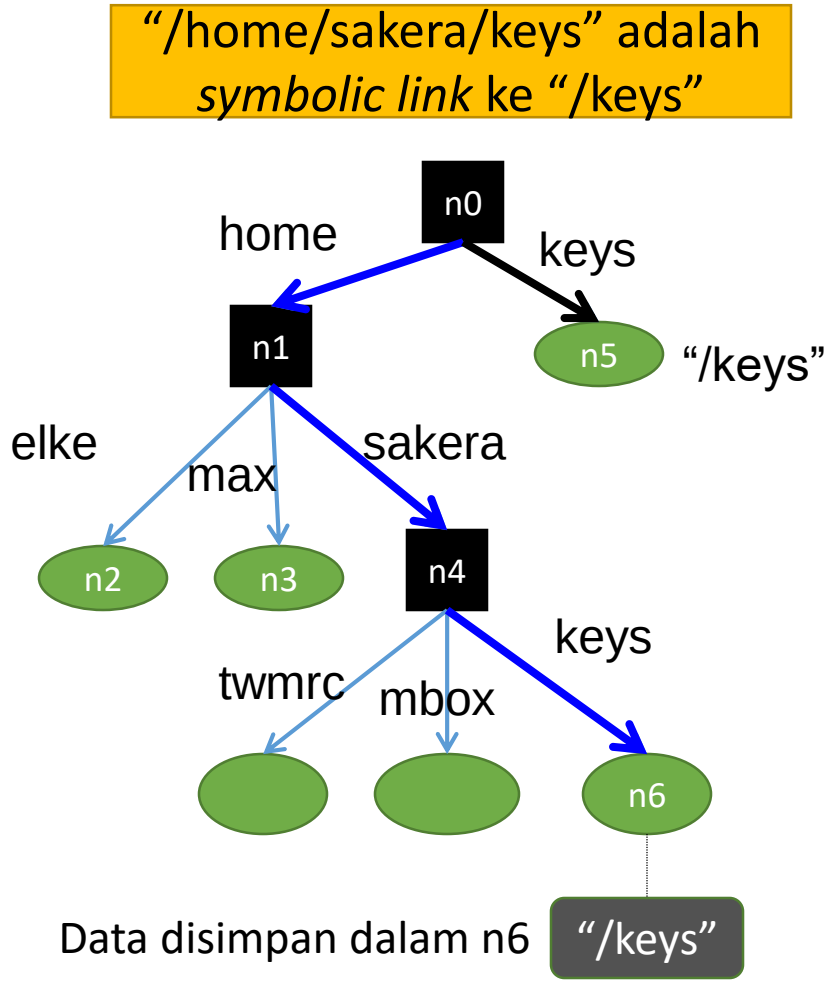
- Resolusi nama:
  - Serupa dengan resolusi nama umum

- Aturan:
  - Harus tidak ada siklus di dalam graf.



2. Symbolic Links

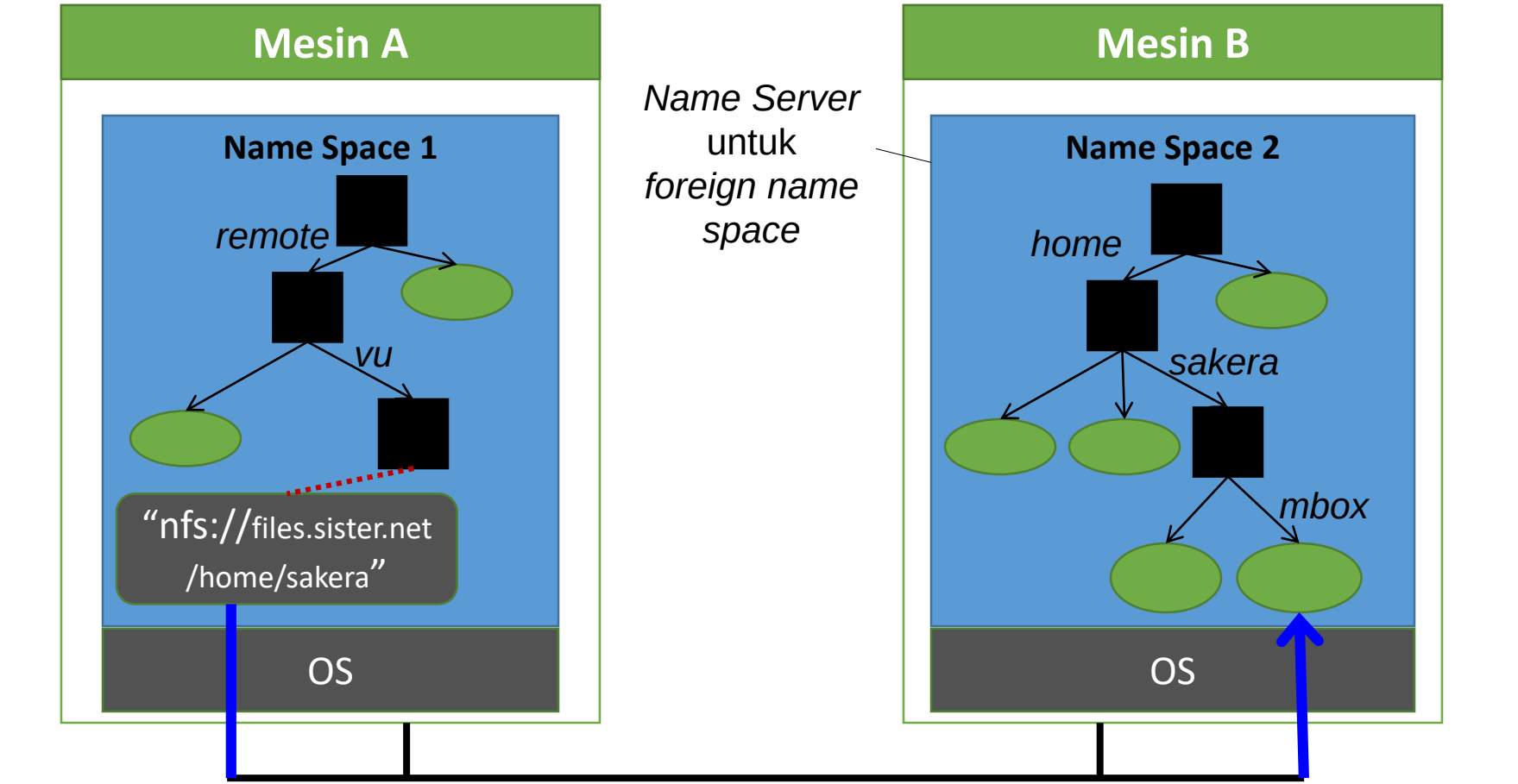
- Symbolic link menyimpan nama dari node asli sebagai *data*
- Resolusi nama bagi suatu *symbolic link* SL
  - Pertama, resolve nama SL
  - Baca isi dari SL
  - Resolusi nama berlanjut dengan isi dari SL
- Aturan:
  - Harus tidak muncul siklus referensi



Penggabungan Ruang Nama

- Dua atau lebih ruangnama dapat digabungkan (*merged*) secara transparan dengan suatu teknik yang dikenal sebagai *mounting*
- Dengan *mounting*, suatu *directory node* dalam satu ruang nama akan menyimpan *identifier* dari *directory node* dari suatu ruang nama lain
- Network File System (NFS) adalah contoh dimana ruang nama berbeda digabungkan (*mounted*)
  - NFS memungkinkan akses transparan ke file-file *remote*.

Contoh *Mounting* Ruang Nama dalam NFS



Resolusi nama untuk "/remote/vu/home/sakera/mbox" dalam suatu sistem file terdistribusi

Ruang Nama Terdistribusi

- Dalam sistem terdistribusi skala besar, penting sekali mendistribusikan ruang nama ke banyak server nama (*name servers*)
  - Mendistribusikan node-node dari graf penamaan
  - Mendistribusikan manajemen ruang nama tersebut
  - Mendistribusikan mekanisme resolusi nama.

Layer dalam Ruang Nama Terdistribusi

- Ruang nama terdistribusi dapat dibagi ke dalam 3 (tiga) *layer*:

Lapisan Global

- Terdiri dari node-node direktori level tinggi
- Node-node direktori bersama-sama dikelola oleh administrasi berbeda

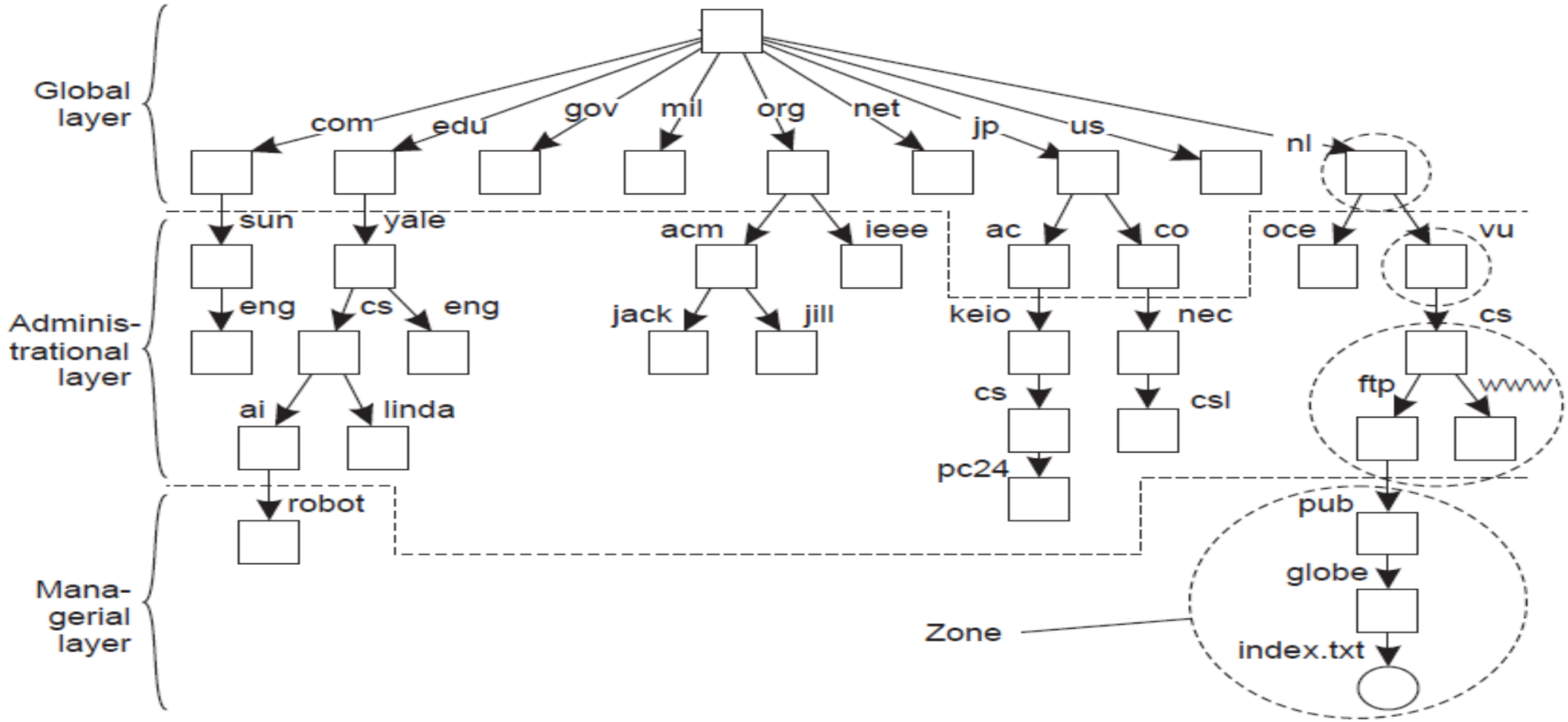
Lapisan Administrasional

- Mengandung node-node direktori level tengah
- Node-node direktori dikelompokkan bersama-sama dengan cara tertentu yang setiap grup dikelola oleh suatu administrasi

Lapisan Manajerial

- Berisi node-node direktori level bawah (rendah) di dalam suatu administrasi tunggal
- Persoalan utama adalah bagaimana secara efisien memetakan node-node direktori ke server nama lokal.

Ruang Nama Terdistribusi: Contoh



Perbandingan Server Nama pada Layer Berbeda

	Global	Administrational	Managerial
Skala geografis dari jaringan	Dunia	Organisasi	Departemen
Total jumlah node	Sedikit	Banyak	Sangat banyak
Jumlah replika	Banyak	Tidak ada / Sedikit	Tidak ada
Propagasi update	Malas	Segera	Segera
Apakah caching sisi client diterapkan?	Ya	Ya	Kadang kala
Kemampuan merespon pencarian	Detik	Milidetik	Segera

Resolusi Nama Terdistribusi

- Resolusi nama terdistribusi bertanggungjawab untuk memetakan nama-nama ke alamat dalam suatu sistem dimana:
  - Server-server nama didistribusikan antar node-node yang berpartisipasi
  - Setiap server nama mempunyai suatu *name resolver* lokal.

- Akan dibahas dua algoritma resolusi nama terdistribusi:
  1. Resolusi nama iteratif
  2. Resolusi nama rekursif

1. Resolusi Nama Iteratif

1. Client menyerahkan nama lengkap yang akan dipecahkan ke *root name server*

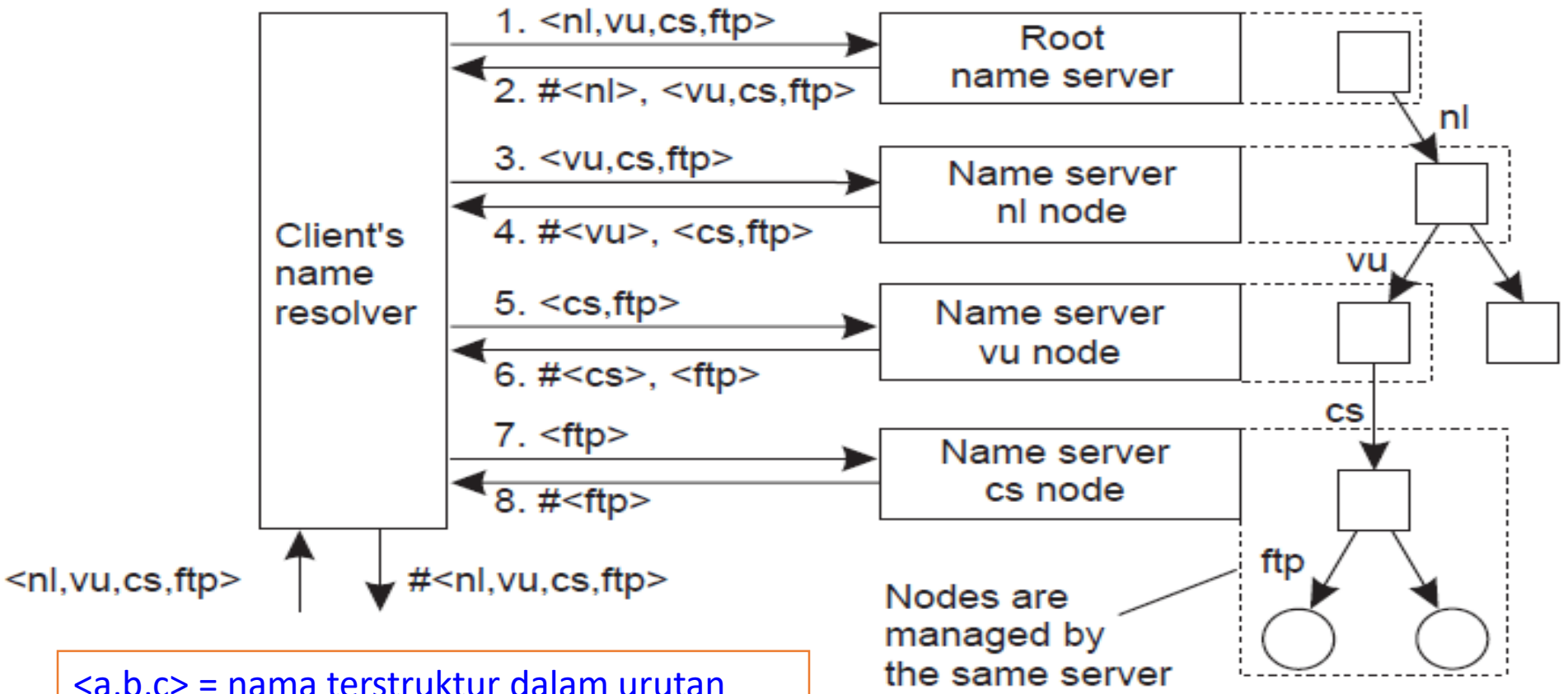
2. *Root name server* memecahkan (*resolve*) nama sejauh kemampuannya dan mengembalikan hasilnya kepada client
   - *Root name server* juga mengembalikan alamat dari server nama level selanjutnya (disingkat NLNS) jika alamat tidak terpecahkan secara lengkap

3. Client melewatkan bagian yang tidak terpecahkan dari nama ke NLNS

4. NLNS memecahkan nama sejauh kemampuannya dan mengembalikan hasilnya kepada client (bersama dengan *next-level name server*-nya)

5. Proses ini berlanjut sampai nama lengkap terpecahkan.

1. Resolusi Nama Iteratif: Contoh



<a,b,c> = nama terstruktur dalam urutan
#<a> = alamat dari node dengan nama "a"

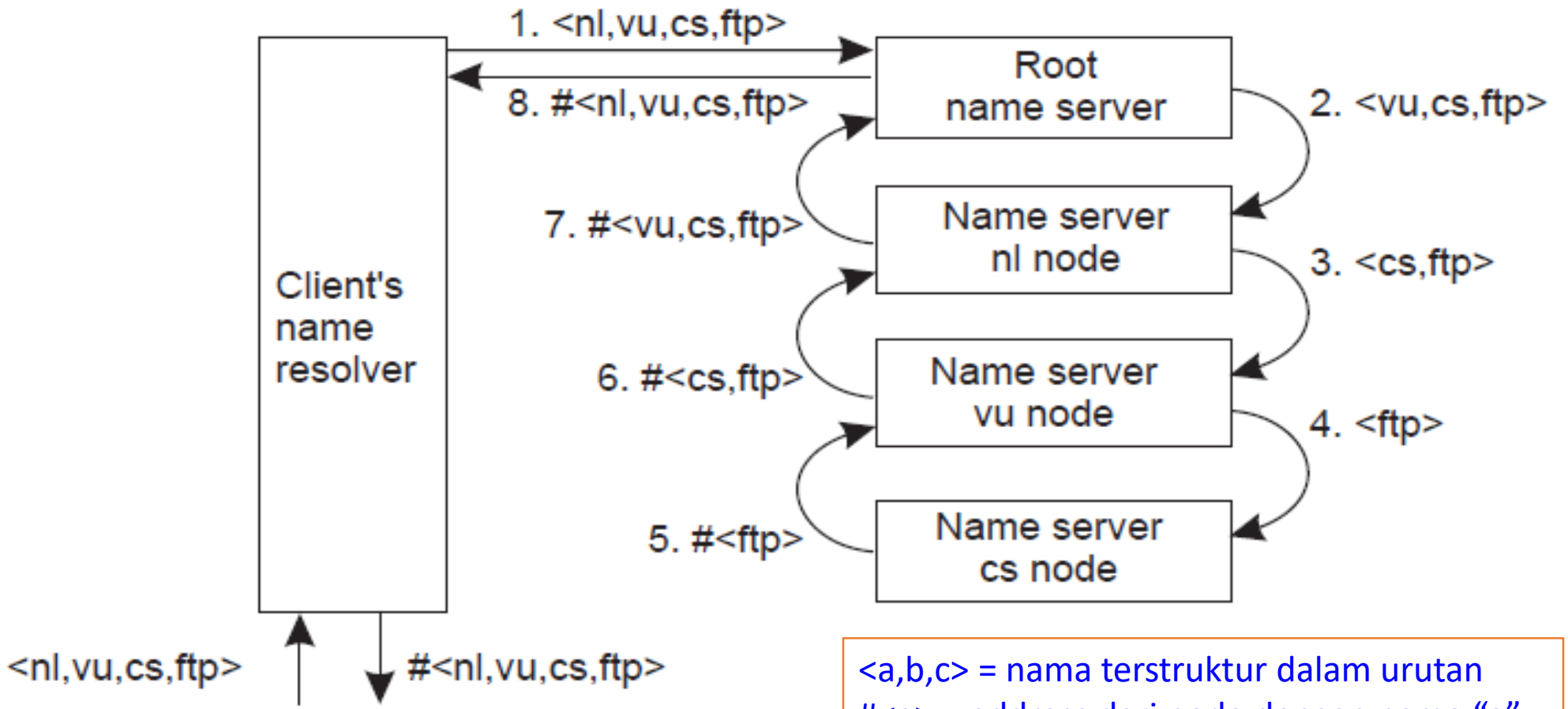
Pemecahan nama "*ftp.cs.vu.nl*"

2. Resolusi Nama Rekursif

- Pendekatan:
  - Client menyerahkan nama yang akan dipecahkan kepada *root name server*
  - *Root name server* tersebut melewatkan hasilnya ke *next name server* yang ditemukannya
  - Proses berlanjut sampai nama tersebut secara lengkap terpecahkan

- Kekurangan:
  - Biaya besar pada server nama (terutama pada server nama tingkat tinggi)

2. Resolusi Nama Rekursif: Contoh



<a,b,c> = nama terstruktur dalam urutan
#<a> = address dari node dengan nama "a"

Pemecahan nama "*ftp.cs.vu.nl*"

Kategori Penamaan

- Penamanaan *flat*
- Penamaan terstruktur
- Penamaan berbasis atribut

Penamaan Berbasis Atribut

- Dalam banyak kasus, adalah jauh lebih tepat menamai dan mencari entitas berrdasarkan arti dari atribut-atributnya
  - Mirip dengan layanan direktori tradisional (misal: *yellow pages*)

- Namun, operasi pencarian (*look-up*) dapat menjadi sangat mahal
  - Harus mencocokkan nilai-nilai atribut yang direquest terhadap nilai-nilai atribut aktual, yang mungkin memerlukan pemeriksaan semua entitas

- Solusi: Implementasikan layanan direktori dasar sebagai sebuah database, dan kombinasikan itu dengan sistem penamaan terstruktur tradisional

- Akan dibahas *Light-weight Directory Access Protocol* (LDAP); suatu sistem contoh yang menggunakan penamaan berbasis atribut.

Light-weight Directory Access Protocol (LDAP)

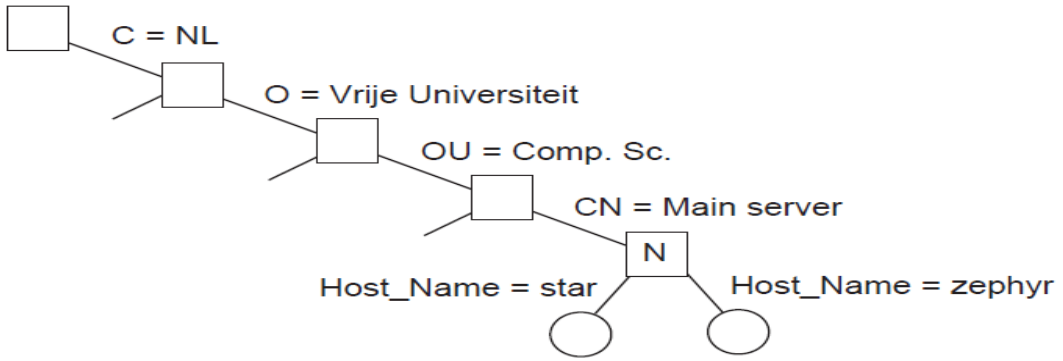
- Layana direktori LDAP terdiri dari sejumlah record bernama "*directory entries*"
  - Setiap record tersusun dari pasangan *(attribute, value)*
  - Standard LDAP menetapkan lima atribut untuk setiap record
- Directory Information Base (DIB) adalah koleksi semua *directory entries*
  - Setiap record dalam DIB bersifat unik
  - Setiap record direpresentasikan oleh suatu nama yang membedakan

Misal: `/C=NL/O=Vrije Universiteit/OU=Comp. Sc.`

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	star
Host_Address	192.31.231.42

Pohon Informasi Direktori dalam LDAP

- Semua record dalam DIB dapat ditata ke dalam suatu pohon hirarkis bernama *Directory Information Tree (DIT)*
- LDAP menyediakan mekanisme pencarian lanjut berdasarkan pada atribut dengan melintasi DIT tersebut
- Sintaks contoh untuk pencarian semua Main_Servers di dalam Vrije Universiteit:



Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	star
Host_Address	192.31.231.42

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	zephyr
Host_Address	137.37.20.10

```
search("&(C = NL) (O = Vrije Universiteit) (OU = *) (CN = Main server)")
```

Rangkuman

- Penamaan dan resolusi nama memungkinkan pengaksesan entitas-entitas dalam suatu sistem terdistribusi

- Tiga tipe penamaan:
  - Penamaan *flat*
    - Broadcasting, forward pointer, pendekatan *home-based*, Distributed Hash Tables (DHTs)
  - Penamaan terstruktur
    - Mengorganisir nama-nama ke dalam *Name Spaces*
    - *Name Spaces* Terdistribusi
  - Penamaan berbasis atribut
    - Entitas-entitas dicari menggunakan atribut-atributnya

Kuliah Selanjutnya...

- Konkurensi dan Sinkronisasi
  - Menjelaskan perlunya sinkronisasi
  - Menganalisis bagaimana komputer menyerempakkan jam dan akses konkuren ke sumber daya
    - Algoritma Sinkronisasi Jam
    - Algoritma Mutual Exclusion