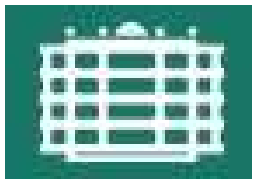


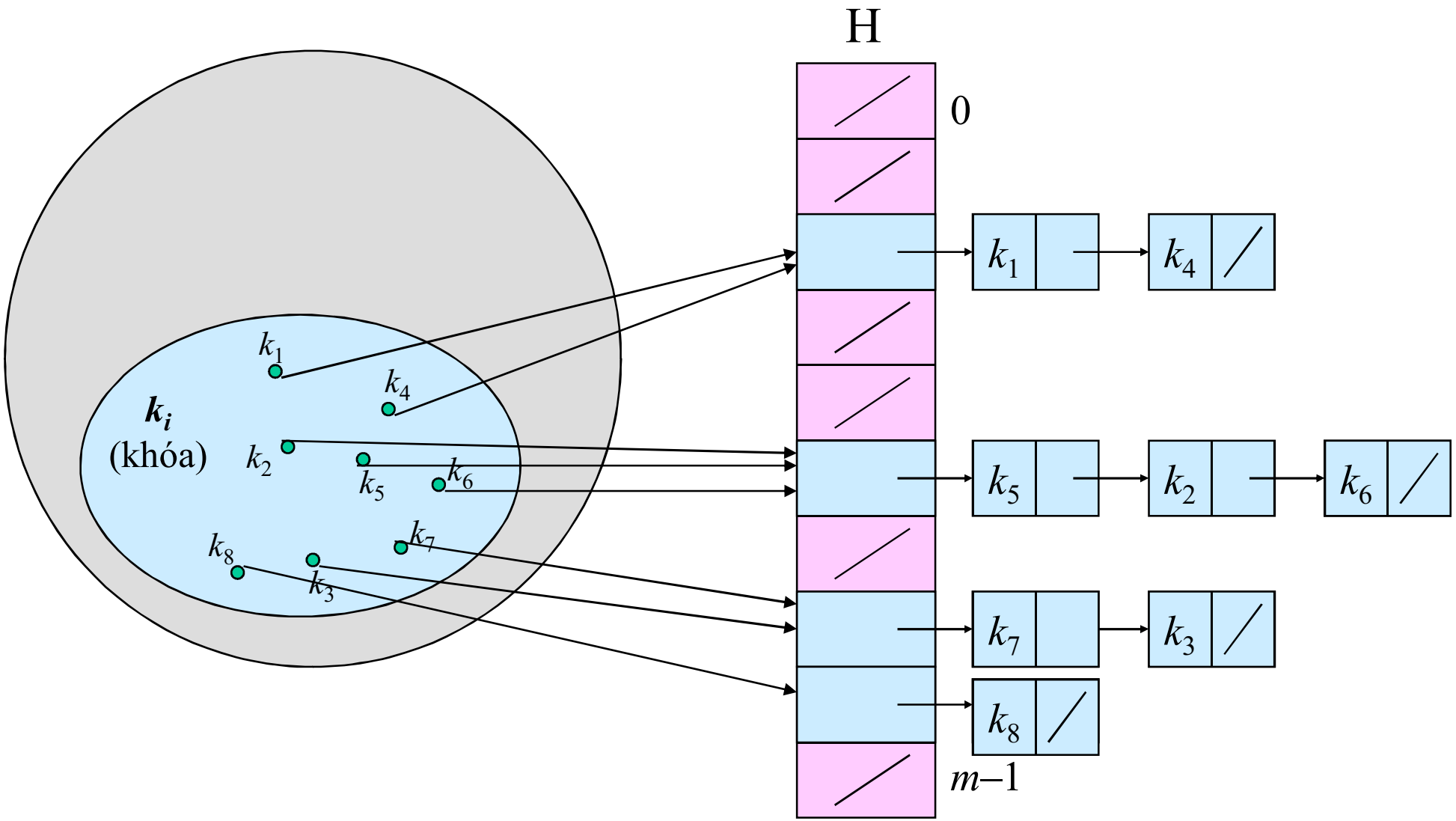
Bảng Băm Phân Tán (DHT)
và
Mạng Ngang Hàng Chord

Phan Anh - Nguyễn Đình Nghĩa

Nhắc lại kiến thức

Bảng băm (Hash Table)

Cấu trúc dữ liệu bảng băm



Cấu trúc dữ liệu bảng băm

- Dùng để lưu trữ một tập các khoá: k_1, k_2,…
- Khoá k_i được lưu trữ ở vị trí $h(k_i)$ của mảng H, h là hàm băm.
- Các khoá có cùng giá trị băm với khoá k_i sẽ được lưu trữ trong một danh sách liên kết được quản lý bởi phần tử $H(h(k_i))$ (hiện tượng xung đột).
- Hàm băm là một hàm biến đổi khoá k sang một số nguyên là vị trí trong mảng H.
  - VD: h(“Ha Noi”)= 16
- Thích hợp với các thao tác chèn, tìm kiếm, xoá một khoá.

Ví dụ một bảng băm đơn giản

Khóa k sẽ được lưu trữ tại vị trí **k mod M** (M kích thước mảng).

0	1	2	3	4	5	6	7	8	9
					95				

Thêm phần tử x = 95 vào mảng
95 mod 10 = 5.

Ví dụ một bảng băm đơn giản

Các giá trị: 31, 10 , 14, 93, 82, 95, 79, 18, 27

0	1	2	3	4	5	6	7	8	9
10	31	82	93	14	95	46	27	18	79

Vấn đề nảy sinh

Giả sử thêm 55 vào mảng:

0	1	2	3	4	5	6	7	8	9
		82			95		27		

+ 55 phải lưu vào vị trí 5. Tuy nhiên vị trí này đã có chứa 95.

=> Giải quyết đụng độ.

Vấn đề xung đột khi xử lý bảng băm

- Trong thực tế có nhiều trường hợp có nhiều hơn 2 phần tử sẽ được "băm" vào cùng 1 vị trí.

- Hiển nhiên phần tử được "băm" đầu tiên sẽ chiếm lĩnh vị trí đó, các phần tử sau cần phải được lưu vào các vị trí trống khác sao cho vấn đề truy xuất và tìm kiếm phải dễ dàng.

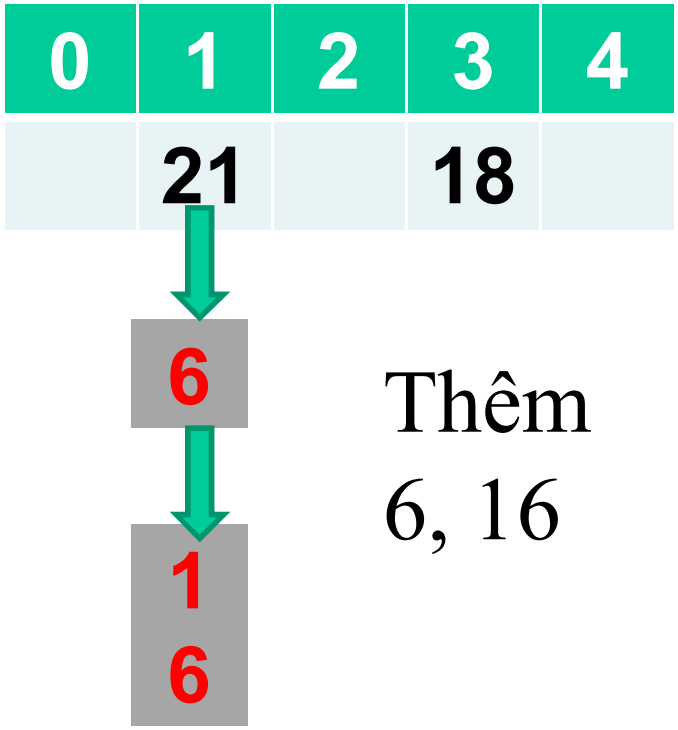
a. Làm giảm xung đột

- Hàm băm cần được chọn sao cho:
  - Xác xuất phân bố khoá là đều nhau.
  - Dễ dàng tính toán thao tác.

Thông thường, hàm băm sử dụng các số nguyên tố (vì xác suất ngẫu nhiên phân bố các số nguyên tố là đều nhất).

I. Sử dụng danh sách liên kết (nối kết)

- Ý tưởng: "Các phần tử băm vào trùng vị trí k được nối vào DS nối kết" tại vị trí đó.



Thêm
6, 16

Hàm băm:
F(k) = k mod 5

* Phân tích

* PP DSLK có nhiều khuyết điểm:

- Khi có quá nhiều khoá vào cùng vị trí, DSLK thì tại vị trí đó sẽ **rất dài** => Tăng chi phí tìm kiếm.
- Các **ô trống** còn dư nhiều => lãng phí về thời gian tìm kiếm và không gian lưu trữ.

II. Sử dụng PP "Dò tuyến tính"

- Ý tưởng: "Nếu có 1 khóa bị băm vào vị trí đã có phần tử thì nó sẽ được chèn vào ô trống gần nhất theo phía bên phải (hoặc trái)".

0	1	2	3	4
	21	6	18	16

Hàm băm:

$F(k) = k \bmod 5$

Thêm

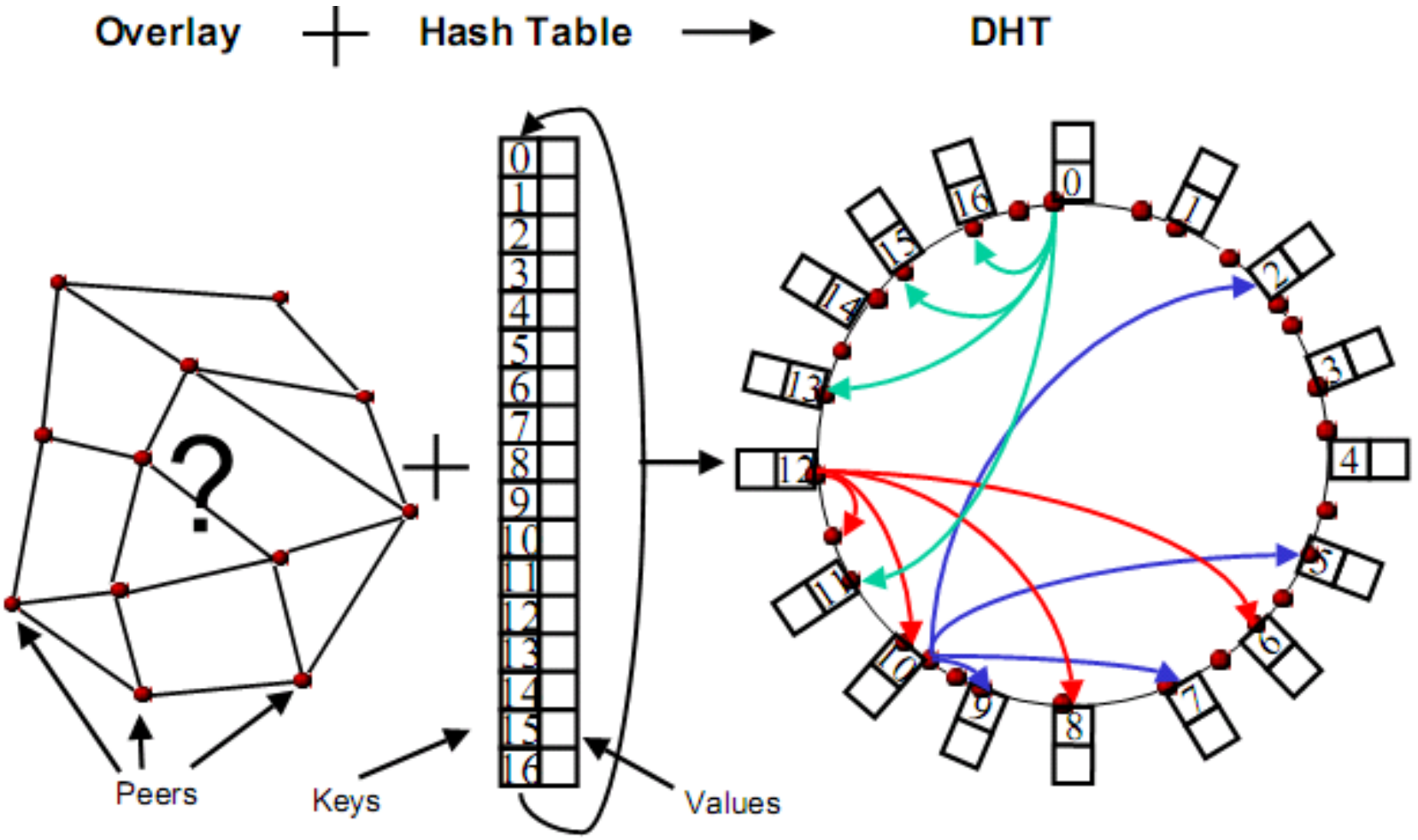
6, 16

* Phân tích

- Phương pháp này dễ thực hiện.
- Nếu có nhiều phần tử băm trùng nhau thì đặc tính bảng băm bị mất đi.
- Trong trường hợp xấu nhất tìm kiếm trên bảng băm thành tìm kiếm tuyến tính trên mảng.

Bảng Băm Phân Tán (Distributed Hash Tables - DHTs)

Bảng băm phân tán (Distributed Hash Table - DHT)

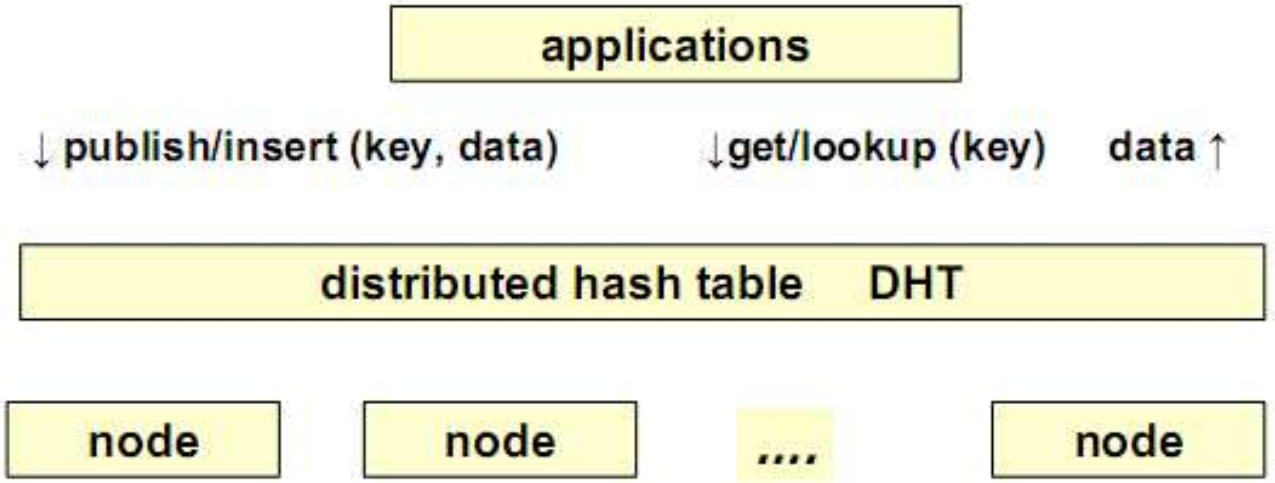


Bảng băm phân tán (Distributed Hash Table - DHT)

- DHTs là một lớp (class) của hệ thống phân tán có cấu trúc, cung cấp khả năng tìm kiếm (lookup) tương tự như bảng hash:
  - Là một dạng của cấu trúc bảng băm thông thường.
  - Cặp (khóa - key, giá trị - value) được lưu trữ ở DHTs và bất kì node nào cũng có thể truy vấn lấy value một cách hiệu quả thông qua key đã cho.
  - Hỗ các 3 thao tác: chèn, tìm kiếm, xoá các cặp (key, value).

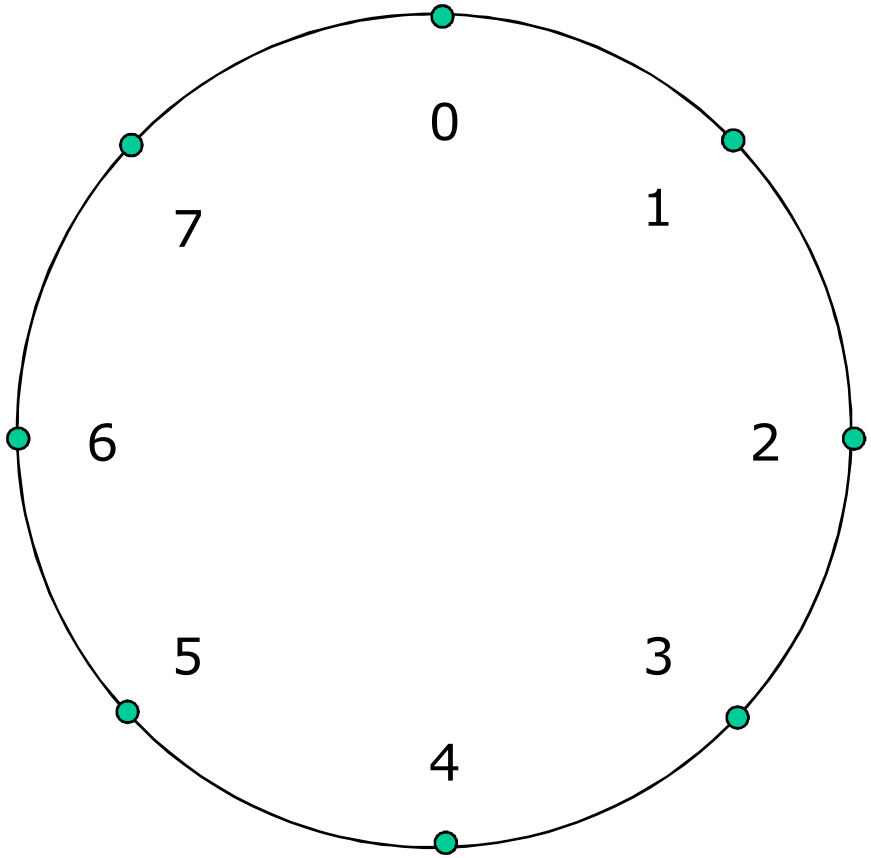
Bảng băm phân tán (Distributed Hash Table - DHT)

- DHTs là cơ sở để xây dựng các hệ thống ứng dụng phân tán như distributed file systems, peer-to-peer file sharing và content distribution systems. Bên cạnh đó là các hệ thống web caching, multicast, anycast, domain name services, và instant messaging.
- Các hệ thống ứng dụng sử dụng DHTs đáng chú ý có BitTorrent, eDonkey ….



DHT: Không gian địa chỉ

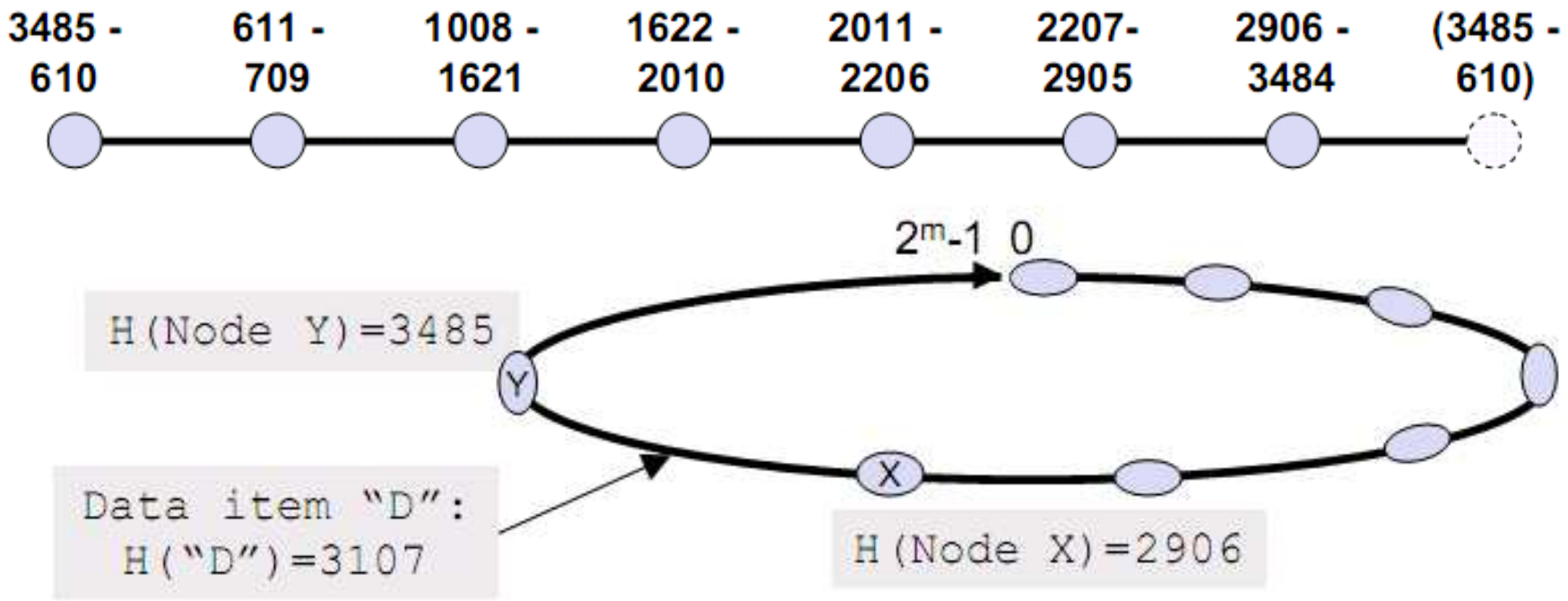
- Không gian địa chỉ của DHT là một tập gồm nhiều số nguyên, vd: từ 0 … 2^3-1, 0 … 2^{160}-1, v.v….



DHT: Không gian địa chỉ

- Các Node và dữ liệu (data items) được ánh xạ vào cùng một không gian địa chỉ.
- Sử dụng hàm băm bảo mật SHA-1 (sinh ra một số 160 bit).
- Đầu vào của hàm băm
  - Địa chỉ IP của một Node.
  - Tên các files dữ liệu.
  - Hoặc nội dung của dữ liệu.

DHT: Không gian địa chỉ



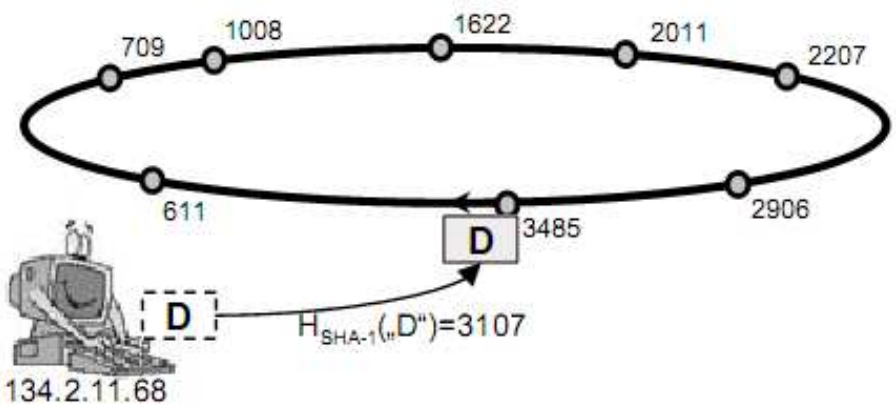
- Trong hình vẽ:
  - Không gian địa chỉ: 0…65535 (2^{16} -1)
  - Được phân hoạch cho 8 Node

DHT: Quản lý dữ liệu

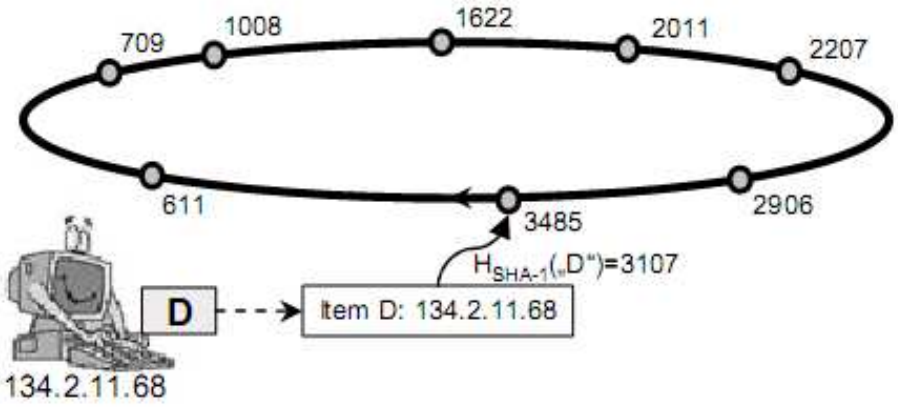
- Địa chỉ IP của một node được băm để xác định vị trí của nó trong bảng băm.
  - NodeID = SHA-1(Node IP Address)
- Mỗi file dữ liệu được gán một số định danh (Key)
  - Key = SHA-1(tên file) hoặc SHA-1(nội dung file).
  - Key là giá trị duy nhất trong không gian địa chỉ.
- Mỗi node quản lý một khoảng giá trị trong không gian địa chỉ.
- Dữ liệu được lưu trữ ở node và được quản lý khoá của dữ liệu.

DHT: Quản lý dữ liệu

- Dữ liệu có thể được lưu trữ trực tiếp hoặc gián tiếp thông qua địa chỉ IP.



(a) Lưu trữ trực tiếp



(b) Lưu trữ gián tiếp

DHT: Tìm kiếm dữ liệu

- Thông điệp tìm kiếm khoá K sẽ được chuyển đi lần lượt đến trong node trong DHT cho đến khi gặp node quản lý khoá K.

DHT: Cơ chế quản lý

- Một node ra nhập (join) hoặc rời bỏ (leave) hệ thống được quản lý như thế nào?
- Node Join: 4 bước
  - Step 1: liên lạc với một node tồn tại trong DHT.
  - Step 2: xác định khoảng địa chỉ mà nó quản lý.
  - Step 3: cập nhật lại thông tin phục vụ cho việc tìm kiếm.
  - Step 4: chuyển tất cả các cặp (Key, Value) thuộc quyền quản lý từ node trước về nó.

DHT: Cơ chế quản lý

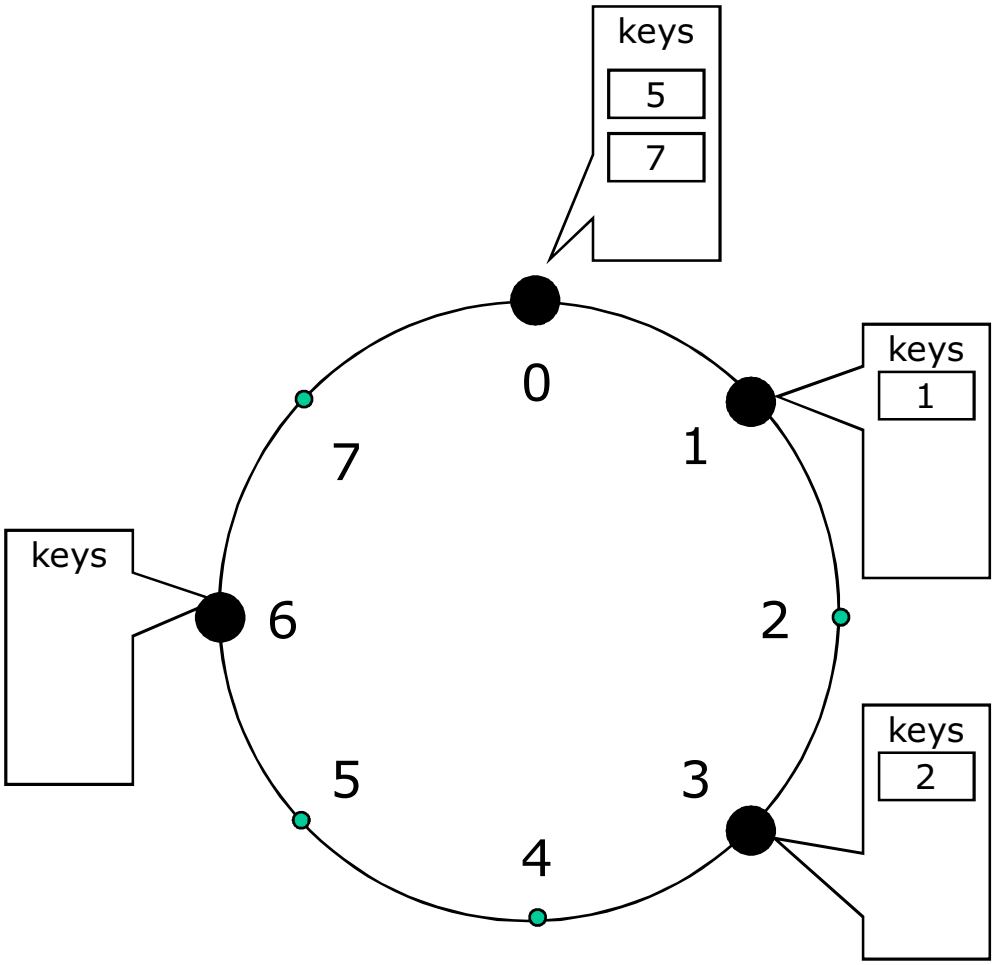
- Node Leave: 2 bước
  - Step 2: chuyển các cặp (Key, Value) của nó về node trước nó.
  - Step 2: cập nhật lại thông tin phục vụ cho việc tìm kiếm.

Giới thiệu giao thức DHT

Chord

I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. **Chord: A scalable Peer-To-Peer lookup service for internet applications**. In Proceedings of the 2001 ACM SIGCOMM Conference, p 149–160, 2001.

Mạng ngang hàng Chord



Mạng ngang hàng Chord

- Là một trong nhiều giao thức sử dụng DHTs (các giao thức sử dụng DHTs khác như CAN, Pastry, Viceroy…).
- Hệ thống tìm kiếm và lưu trữ thông tin P2P.
- Cho một khoá (data item), nó ánh xạ khoá đó vào một node.
- Sử dụng cùng một hàm băm để gán các khoá cho các node.
- Giải quyết được vấn đề tìm kiếm khoá trong một tập các node phân tán.
- Duy trì thông tin tìm đường khi một node tham gia và rời hệ thống.

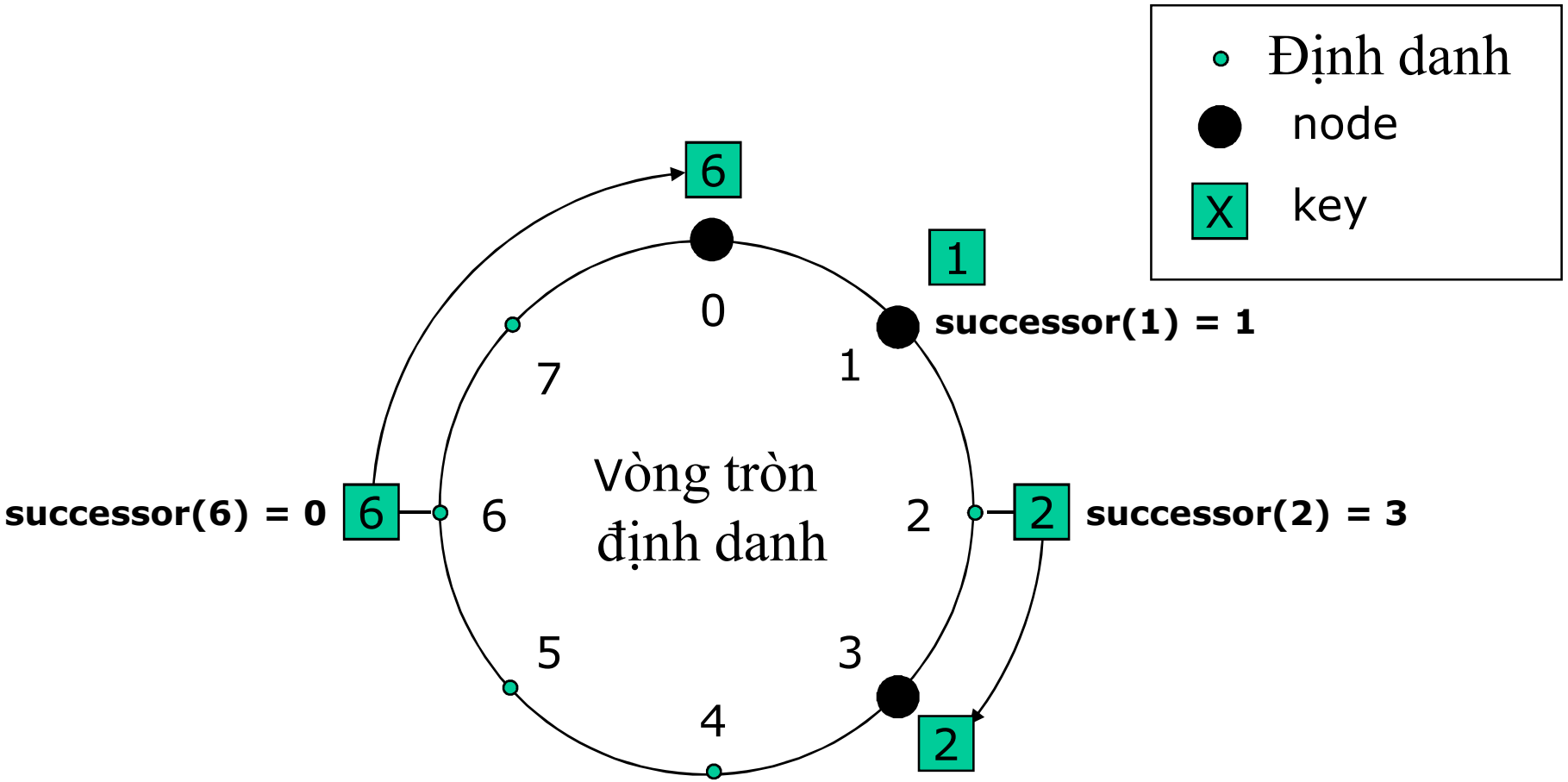
Chord: Hàm băng đồng nhất

- Gán cho mỗi node và mỗi khoá một số định danh m – bit.
- Sử dụng hàm băm SHA-1.
- Định danh của một node là giá trị băm địa chỉ IP của node đó.
  - ID(Node) = SHA-1(IP Address)
- Định danh của một mục dữ liệu (Key) là giá trị băm của tên hoặc nội dung dữ liệu (phụ thuộc vào ứng dụng).
  - ID(Key) = SHA-1(tên file)
  - ID(key) = SHA-1(nội dung file)

Chord: Không gian địa chỉ

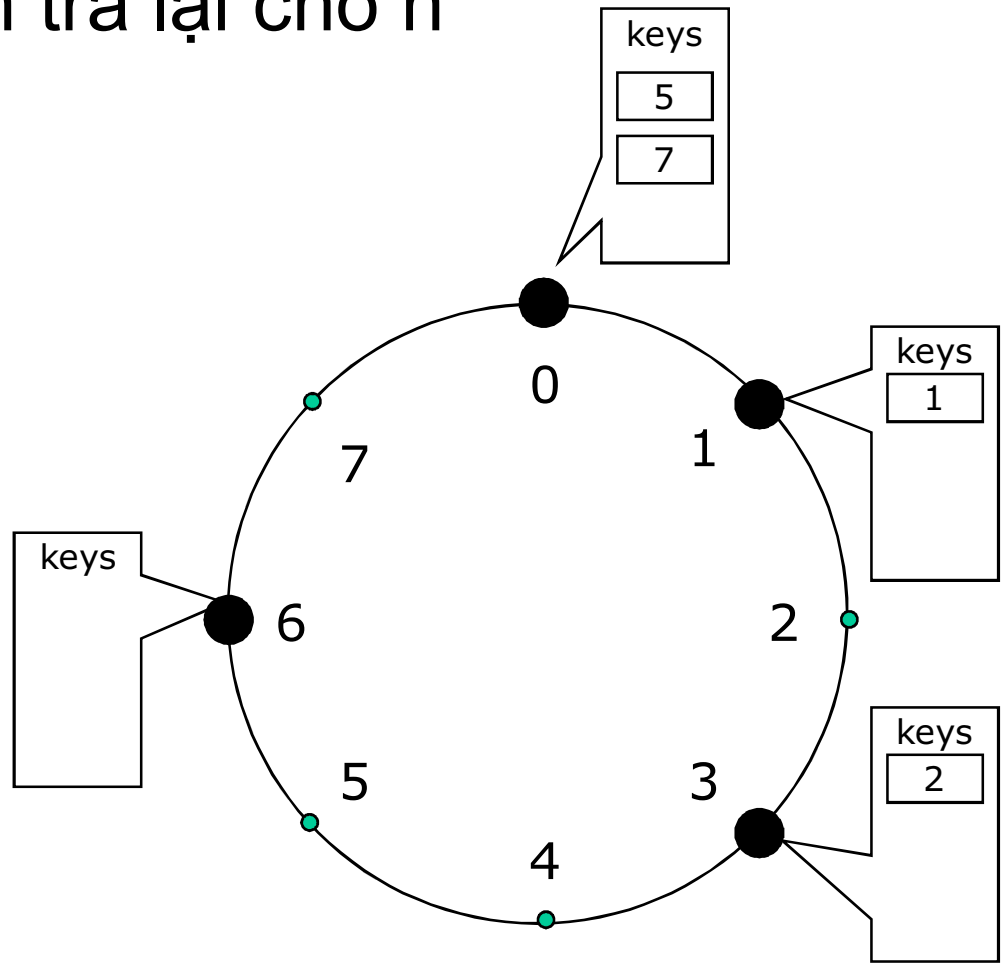
- Trong một không gian định danh m-bit sẽ có 2^m định danh.
- Các định danh được xếp theo thứ tự vòng tròn modulo 2^m.
- Vòng tròn định danh được gọi là vòng tròn *Chord*.
- Cặp (k,v) được lưu ở node đầu tiên có định danh lớn hơn hoặc bằng key trong không gian định danh.
- Node như vậy được gọi là successor k, được ký hiệu là *successor(k)*.

Chord: Successor Nodes



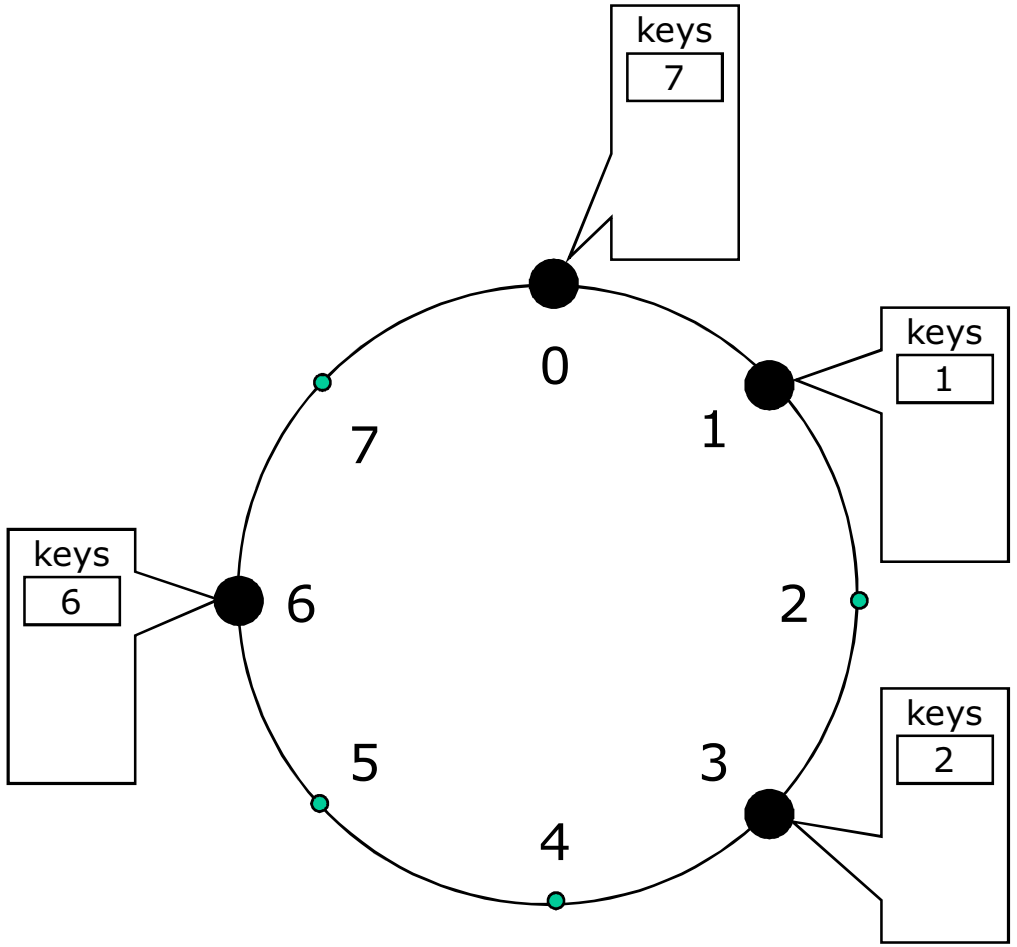
Chord: Join and Departure

- Khi node n join vào mạng, các khoá do n quản lý đã được gán cho successor của n bay giờ sẽ được gán trả lại cho n



Chord: Join and Departure

- Khi node n rời mạng thì tất cả khoá do nó quản lý sẽ được chuyển cho successor của nó.

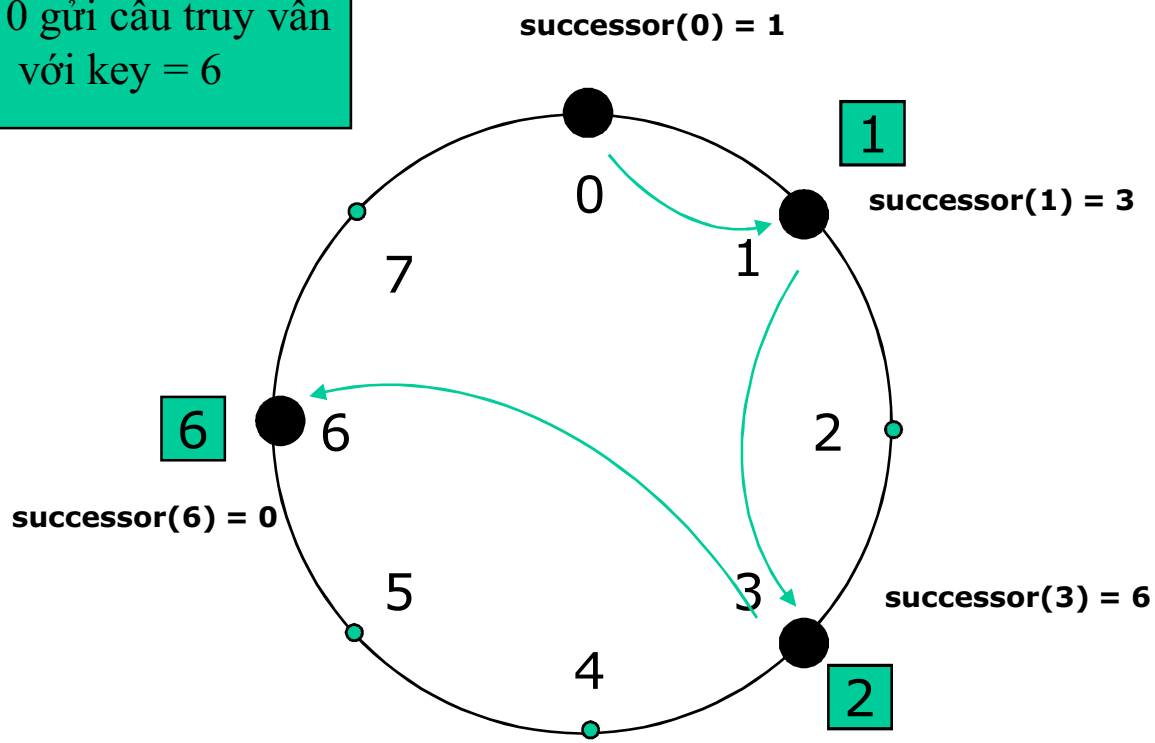


Chord: Tìm kiếm đơn giản

- Mỗi node chỉ biết successor của nó trong vòng tròn định danh, như vậy có thể duyệt qua các node theo thư tự tuyến tính.
- Các câu truy vấn với một định danh cho trước được chuyển quanh vòng tròn thông qua các con trỏ successor cho đến khi gặo node chứa khoá cần tìm.

Chord: Tìm kiếm đơn giản

Node 0 gửi câu truy vấn với key = 6



Chord: Tìm kiếm đơn giản

- Đoạn giả Code để tìm successor:

```
// ask node n to find the successor of id
n.find_successor(id)
    if (id ∈ (n, successor])
        return successor;
    else
        // forward the query around the circle
        return successor.find_successor(id);
```

Chord: Tìm kiếm nâng cao

- Thông tin bổ sung này không phải là bản chất cho vấn đề tính đúng đắn.

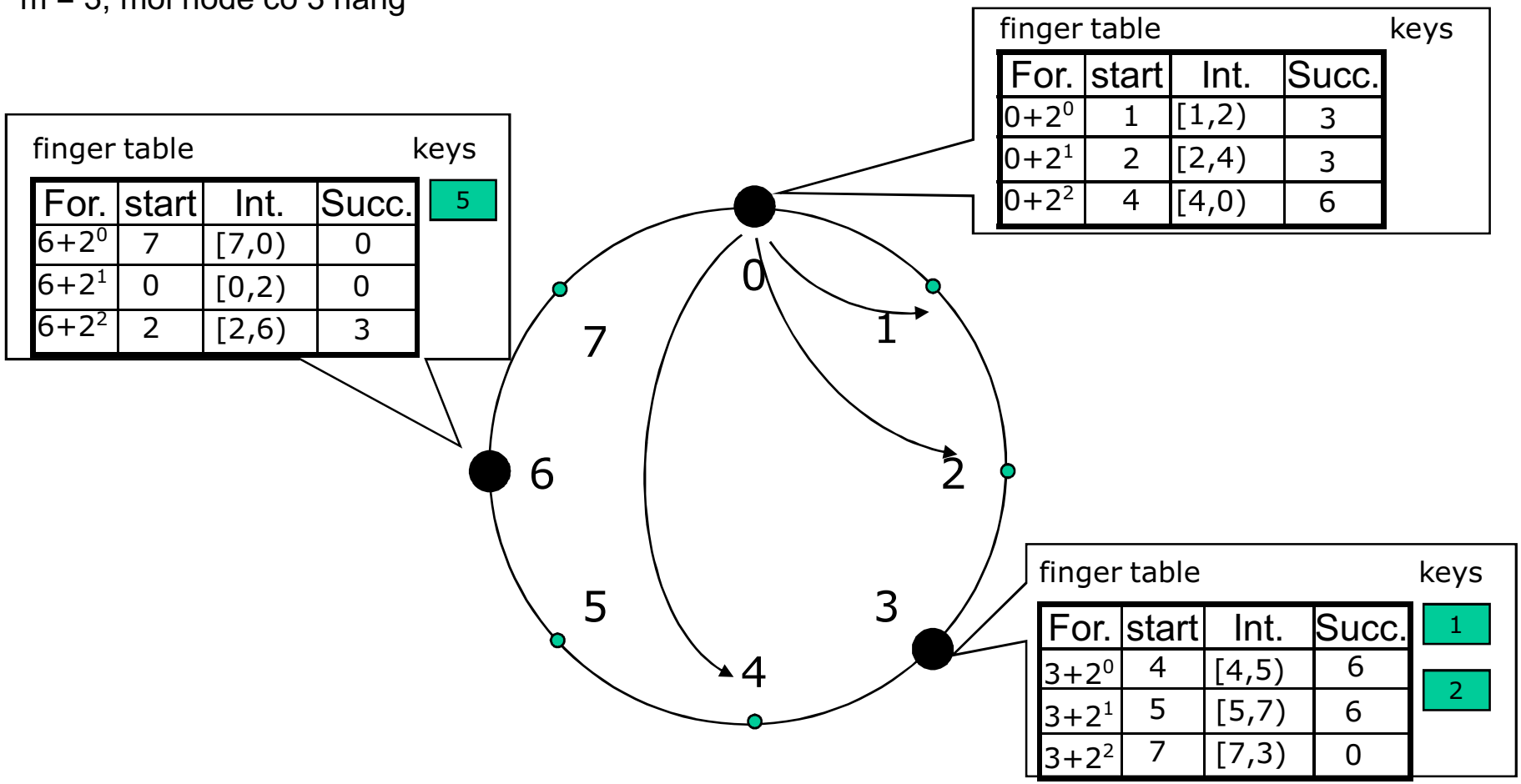
Chord – Finger Tables

- Để tăng tốc độ tìm kiếm, Chord duy trì thêm thông tin tìm đường.
- Mỗi n duy trì một bảng tìm đường gồm m hàng (m là số bit biểu diễn vòng tròn định danh), được gọi là *finger table.*
- Hàng thứ i trong bảng *finger table* của node n xác định node đầu tiên s theo sau node n bởi ít nhất là 2^{i-1} trong vòng tròn định danh.
  - s = successor($n+2^{i-1}$)
- *s* được gọi là *finger thứ i* của node *n*, và được ký hiệu là n.*finger(i)*
- Finger đầu tiên của node n là successor trực tiếp của n trong vòng tròn.

Chord – tìm kiếm mở rộng

Finger table

m = 3, mỗi node có 3 hàng



Chord – tìm kiếm mở rộng

Giả code để tìm successor của một định danh

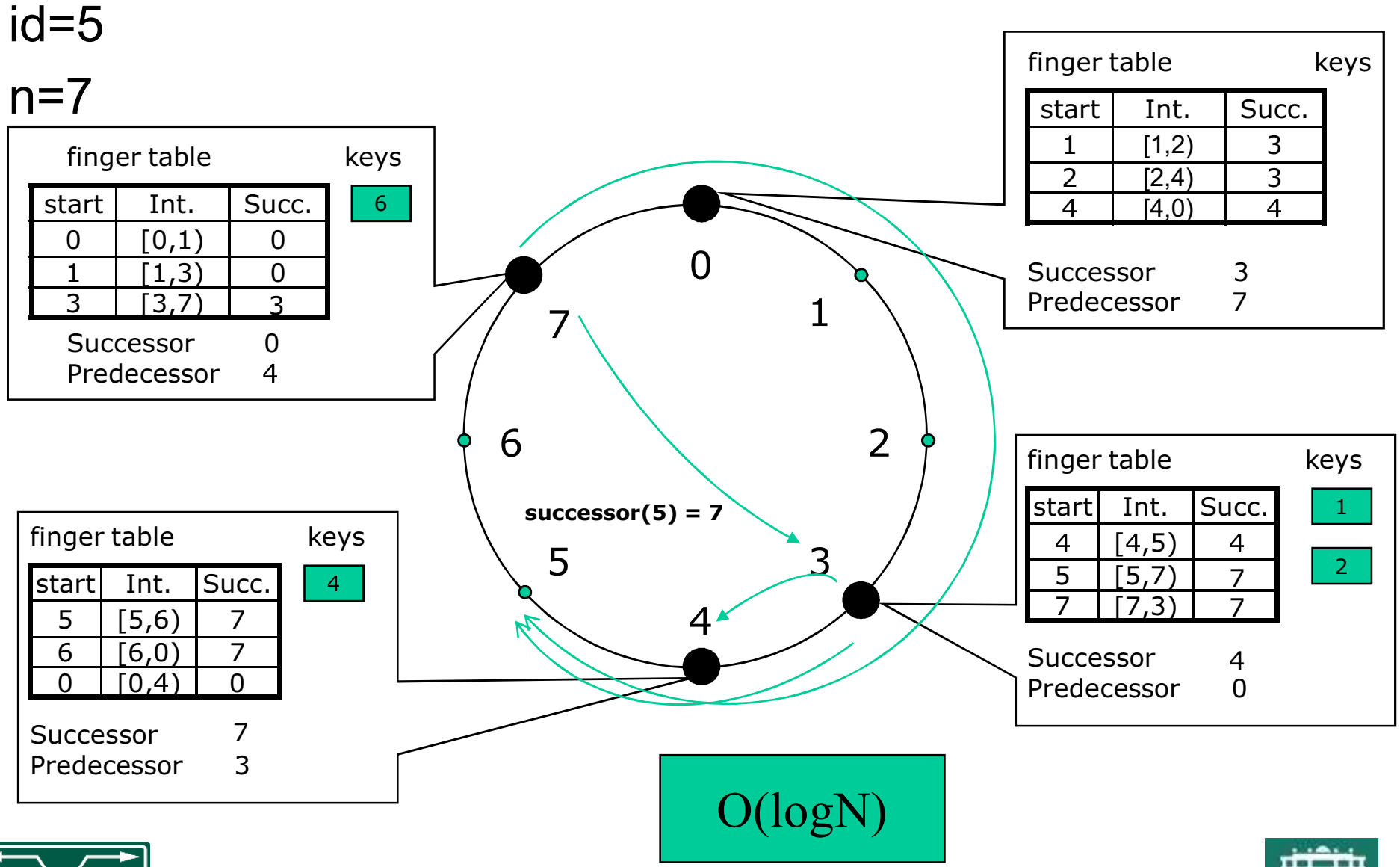
```
// ask node n to find id's successor
n.find_successor(id)
   n' = find_predecessor(id);
   return n'.successor;

// ask node n to find id's predecessor
n.find_predecessor(id)
   n' = n;
   while (id ∉ (n', n'.successor])
      n' = n'.closest_preceding_finger(id);
   return n';

// return closest finger preceding id
n.closest_preceding_finger(id)
   for i = m downto 1
      if (finger[i].node ∈ (n, id))
         return finger[i].node;
   return n;
```

Find id's successor by finding the immediate predecessor of the id

Walk clockwise to find the node which precedes id and whose successor succeeds id

Start with the m^{th} finger of node n
See if it comes between node n and the id, if not, check the $m-1^{th}$ finger until we find one wich does.
This is the closest node preceding id among all the fingers of n

Chord – Tìm kiếm mở rộng

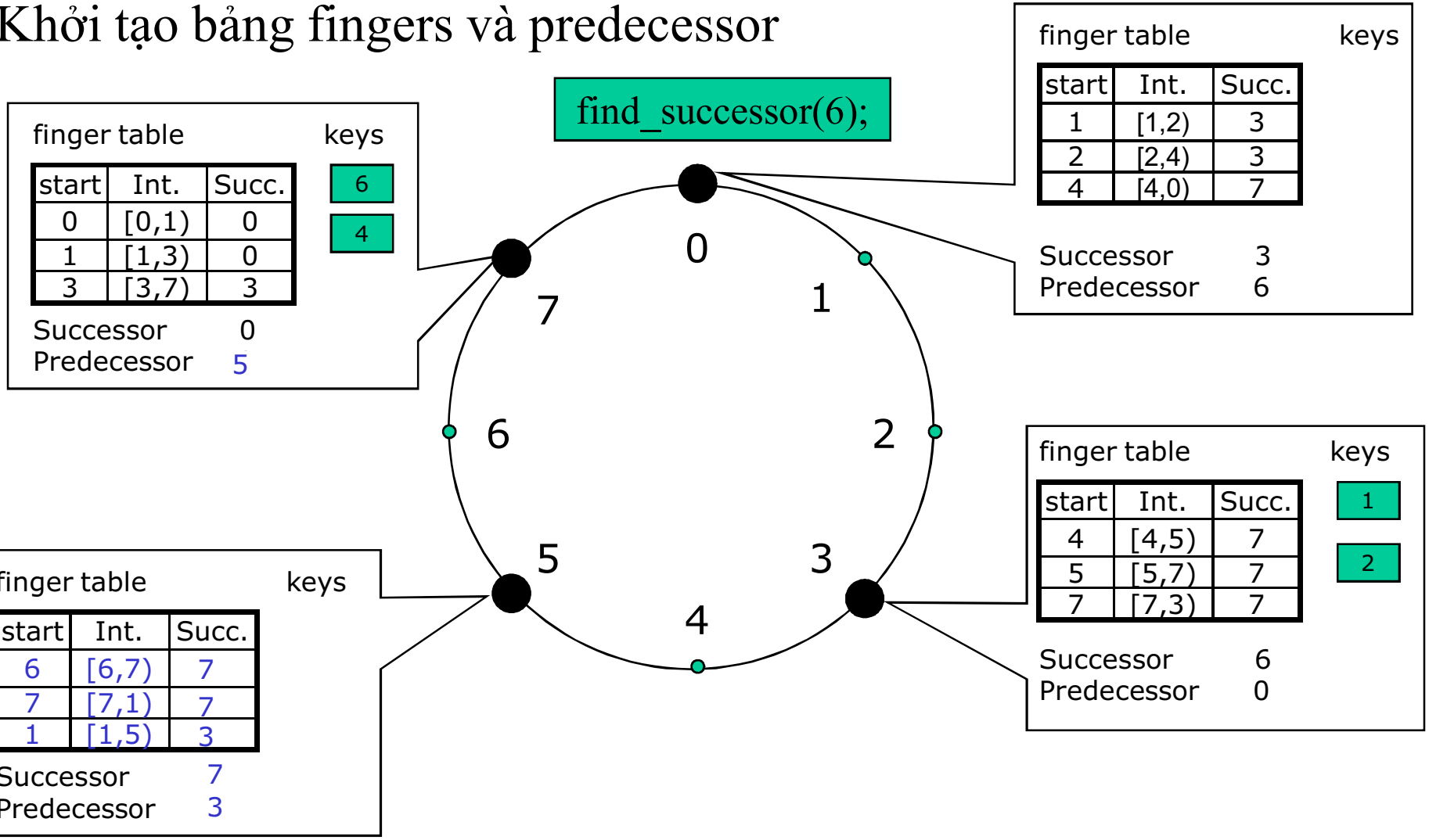


Chord - Node joins

- Các công việc được thực hiện
  - Khởi tạo predecessor và bảng finger table của node n.
  - Cập nhật lại bảng finger table và predecessor của các node tồn tại khi có thêm node n.
  - Chuyển các khoá thích hợp cho node n quản lý.

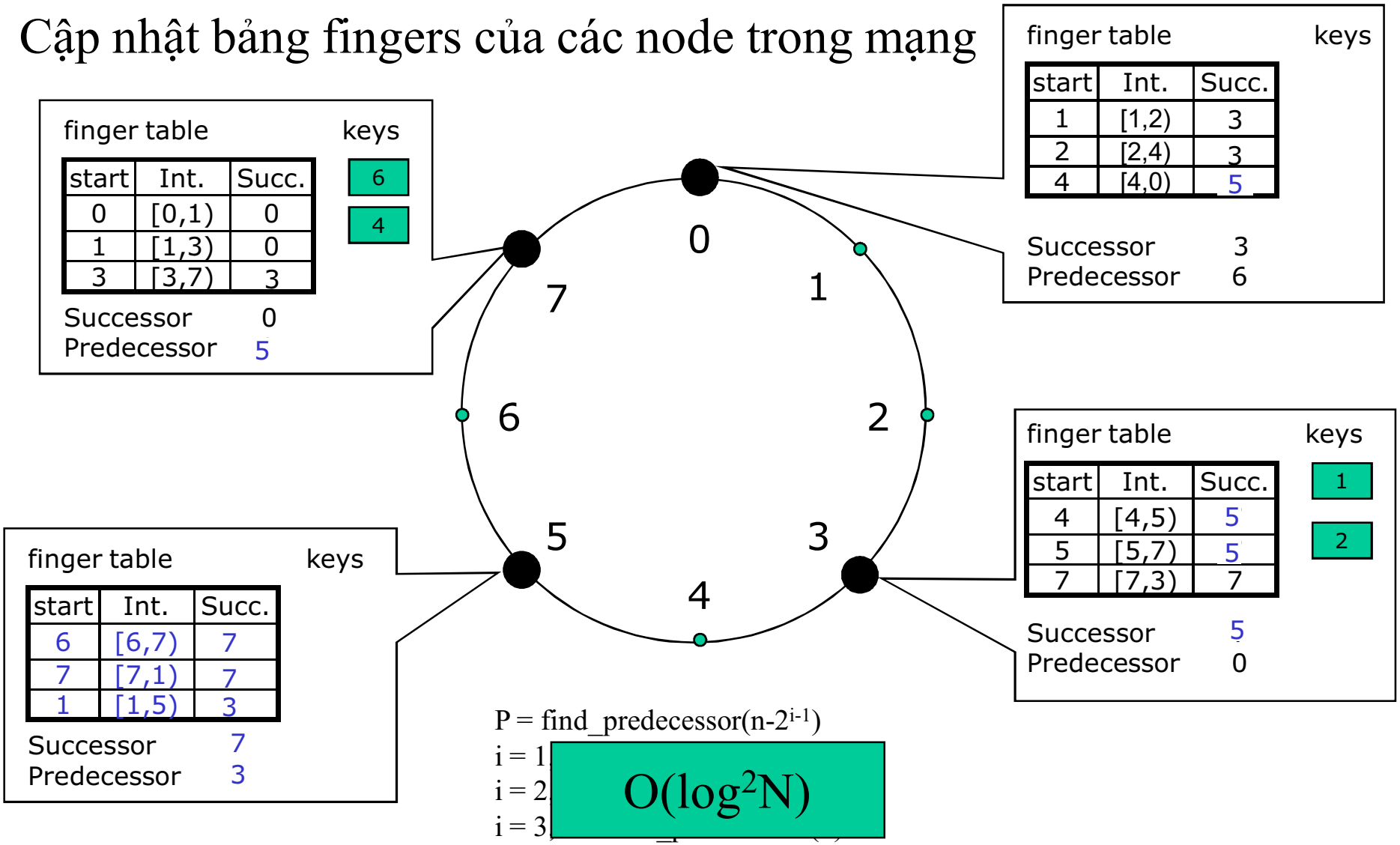
Chord Algorithm - Node joins

Khởi tạo bảng fingers và predecessor

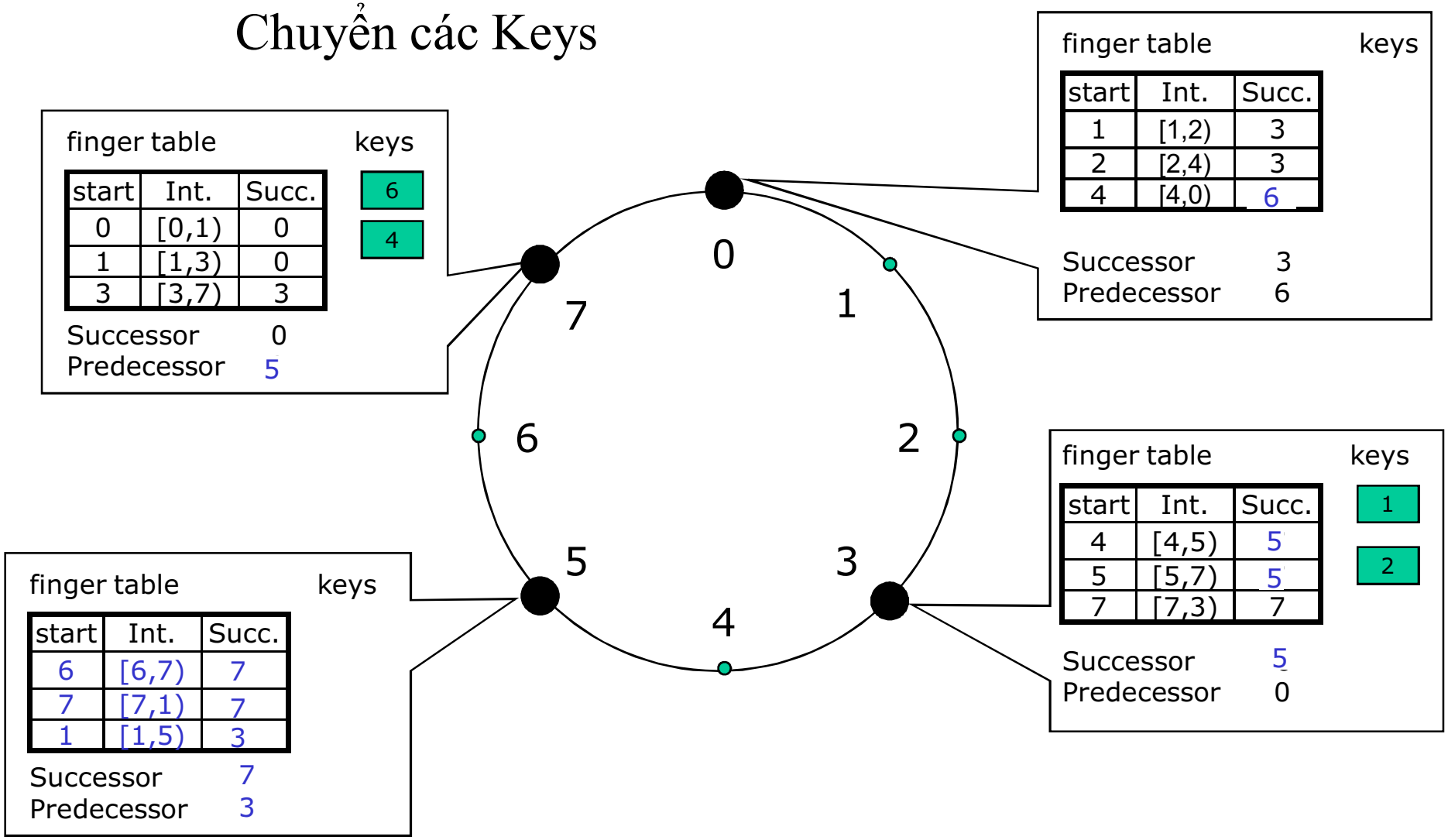


Chord - Node joins

Cập nhật bảng fingers của các node trong mạng



Chord - Node joins



Chord – node joins

```
#define successor finger[1].node

// node n joins the network;
// n' is an arbitrary node in the network
n.join(n')
  if (n')
    init_finger_table(n');
    update_others();
    // move keys in (predecessor, n] from successor
  else // n is the only node in the network
    for i = 1 to m
      finger[i].node = n;
    predecessor = n;

// initialize finger table of local node;
// n' is an arbitrary node already in the network
n.init_finger_table(n')
  finger[1].node = n'.find_successor(finger[1].start);
  predecessor = successor.predecessor;
  successor.predecessor = n;
  for i = 1 to m − 1
    if (finger[i + 1].start ∈ [n,finger[i].node))
      finger[i + 1].node = finger[i].node;
    else
      finger[i + 1].node =
          n'.find_successor(finger[i + 1].start);
```

```
// update all nodes whose finger
// tables should refer to n
n.update_others()
  for i = 1 to m
    // find last node p whose i^th finger might be n
    p = find_predecessor(n − 2^(i−1));
    p.update_finger_table(n, i);

// if s is i^th finger of n, update n's finger table with s
n.update_finger_table(s, i)
  if (s ∈ [n,finger[i].node))
    finger[i].node = s;
    p = predecessor; // get first node preceding n
    p.update_finger_table(s, i);
```

Pseudocode for the node join operation

Chord - Stabilization

- Stabilization
  - Đảm bảo tính đúng đắn và hiệu quả.
  - Đảm bảo tính cập nhật cho các node‘s successor.
  - Sử dụng successor pointers để đảm bảo tính đúng đắn của các bản ghi trong các bảng finger.

Chord - Stabilization

Giả code cho quá trình stabilization

```
n.join(n')
    predecessor = nil;
    successor = n'.find_successor(n);

// periodically verify n's immediate successor,
// and tell the successor about n.
n.stabilize()
    x = successor.predecessor;
    if (x ∈ (n, successor))
        successor = x;
    successor.notify(n);

// n' thinks it might be our predecessor.
n.notify(n')
    if (predecessor is nil or n' ∈ (predecessor, n))
        predecessor = n';

// periodically refresh finger table entries.
n.fix_fingers()
    i = random index > 1 into finger[];
    finger[i].node = find_successor(finger[i].start);
```

Join does not make the rest of the network aware of n

Every node runs stabilize periodically, to verify the successor

Node n asks its successor for the successor's predecessor x. See if x should be n's successor instead. (happens if x recently joined the system)

Notify n's successor of n's exist. Successor changes its predecessor to n if it knows no closer predecessor than n.

Use successor pointers to update finger tables.

Tham khảo

- I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In Proceedings of the 2001 ACM SIGCOMM Conference, pages 149–160, 2001.

- R. Steinmetz, K. Wehrle (Edt.): "Peer-to-Peer Systems and Applications", LNCS 3485, Springer, Chapter 7-8, 2005.
- http://www.wikipedia.org
- http://www.google.com