Lezione n.5
DISTIBUTED HASH TABLES: CHORD

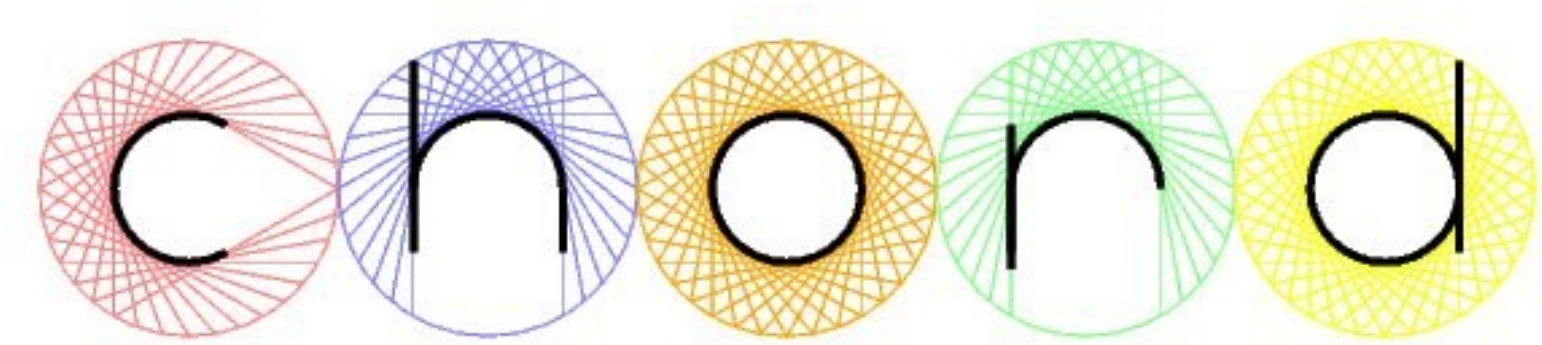
Laura Ricci

Materiale didattico:
Peer-to-Peer Systems
and Applications
Capitolo 8

RIASSUNTO DELLA PRESENTAZIONE

- Chord: idee generali
- Topologia
- Routing
- Auto Organizzazione
  - Arrivo nuovi nodi
  - Partenza volontaria
  - faults

CHORD: INTRODUZIONE



- Materiale didattico sul libro, fonte:

Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. IEEE/ACM Transactions on Networking

•Sviluppato nel 2001 da un gruppo composto da ricercatori del MIT, Università della California

- Download all'indirizzo http://pdos.csail.mit.edu/chord/

CHORD: INTRODUZIONE

- Basato su pochi semplici concetti
  - Facilità di comprensione e di implementazione
  - Eleganza
  - Possibilità di definire ottimizzazioni

- Caratteristiche Principali:
  - Routing
    - Spazio Logico degli indirizzi piatto: gli indirizzi sono identificatori di l-bits invece che indirizzi IP
    - Routing efficiente: log(N) hops se N è il numero totale di nodi del sistema. Dimensione delle tabelle di routing log(N)
  - Auto-organizzazione
    - Gestisce inserzione di nuovi nodi, ritiri volontari dal sistema, fallimenti

CHORD: APPLICAZIONI

- Cooperative Mirroring: distribuire l'informazione gestita da un insieme di servers sui nodi di una rete CHORD
- Time-shared Storage:
  - Scopo: mantenere on line il contenuto fornito da un nodo che si connette in modo intermittente alla rete
  - Il contenuto può essere memorizzato su altri nodi della rete
  - Cooperazione: utilizzo lo spazio disco di altri peer per memorizzare il mio contenuto, offro il mio spazio disco per memorizzare il contenuto di altri
- Indici Distribuiti
  - Ad esempio definire un indice distribuito per Gnutella
  - Chiavi: parole chiave che identificano un file Gnutella
  - Valori associati alle chiavi:indirizzi delle macchine che memorizzano quel contenuto

CHORD: LA TOPOLOGIA

- Hash-table storage
  - put (key, value) per inserire dati in Chord
  - value = get (key) per ricercare dati

- Generazione degli Identificatori mediante Hash. Si utilizza SHA-1 (Secure Hash Standard)

- Ad ogni dato viene associata una chiave key
  - Es: key = sha-1 (dato)

- Ad ogni nodo(host) viene associato un identificatore ID
  - Es: id = sha-1 (indirizzo IP, porta)

- Chiavi ed identificatori sono mappati nello stesso spazio degli indirizzi

CHORD: SHA1 E CONSISTENT HASHING

- Si utilizza SHA1 per assegnare identificatori unici a chiavi e nodi
  - la probabilità che due nodi diversi o due chiavi diverse generino lo stesso identificatore deve essere trascurabile
  - Per questo è necessario utilizzare identificatori sufficientemente lunghi
  - Ad esempio 160-bit (0 <= identificatore < 2^160)

- L'associazione delle chiavi ai nodi avviene mediante consistent hashing che garantisce
  - bilanciamento del carico: le chiavi vengono distribuite uniformemente tra i nodi
  - L'inserimento/ eliminazione di un nodo comporta lo spostamento di un numero limitato di chiavi. Se inserisco l'N-esimo nodo sposto 1/N chiavi

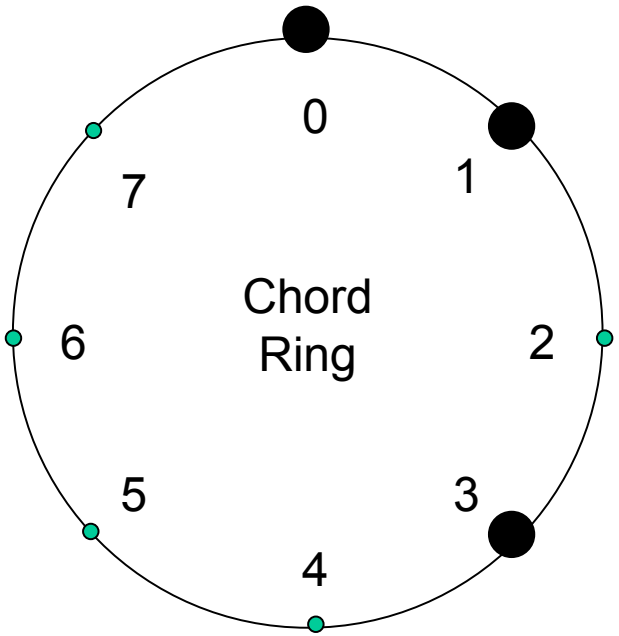
CHORD: LA TOPOLOGIA

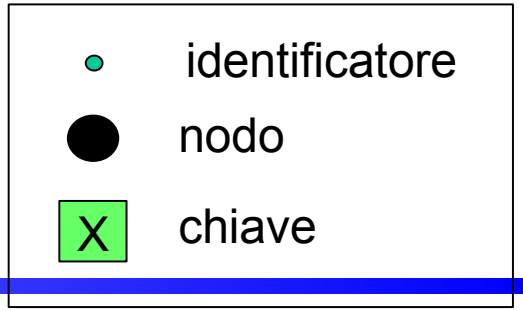
- Ipotesi: consideriamo identificatori di m bits, $[0, 2^m-1]$
- Si stabilisce un ordinamento tra gli identificatori, in base al loro valore numerico. Il successore di 2^m-1 è 0.
- L'ordinamento può essere rappresentato mediante un anello (Chord Ring)
- Consistent hashing: assegna le chiavi ai nodi

  Una chiave di valore K viene assegnata al primo nodo N dell'anello il cui identificatore ID risulta maggiore o uguale a K.
- N viene indicato come nodo successore della chiave key, successor(key)
- N è il primo nodo individuato partendo da K e proseguendo sull'anello Chord in senso orario.

CHORD: LA TOPOLOGIA

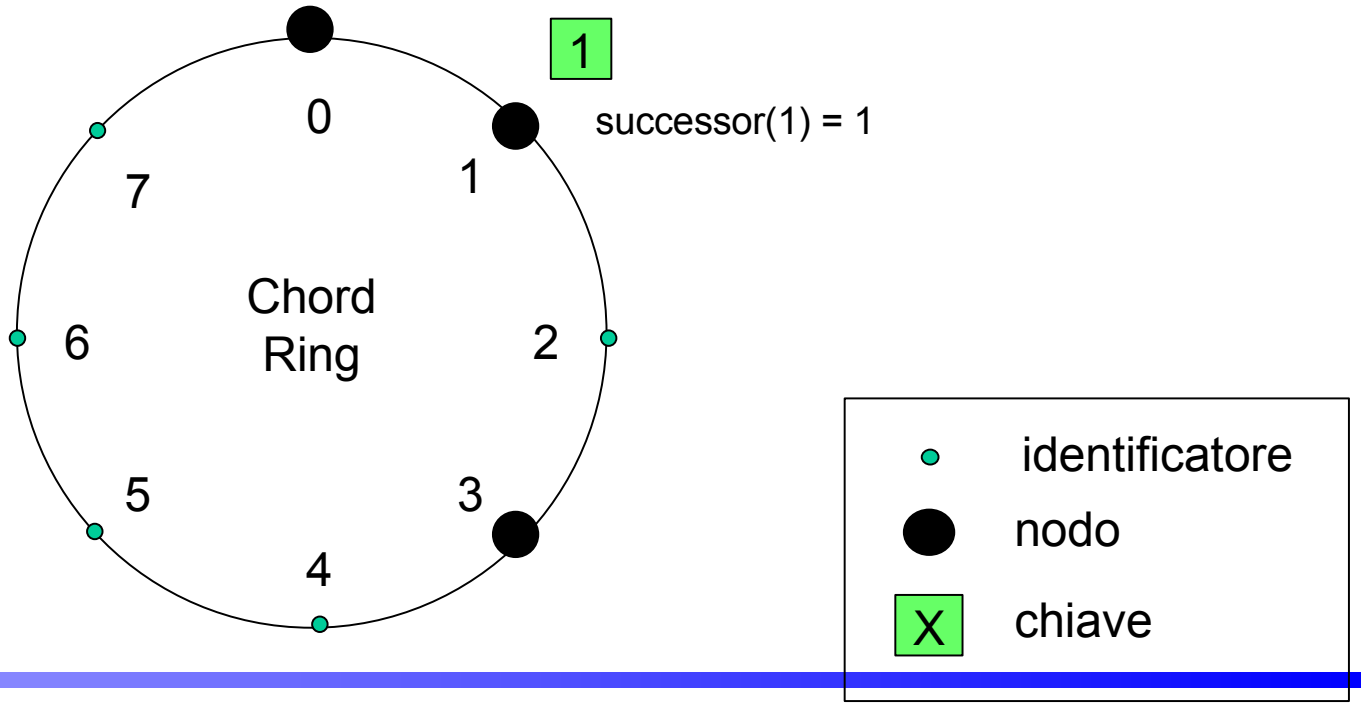
- L'anello contiene chiavi ed IDs, i.e., tutta l'aritmetica è modulo 2^{160}
- La chiave ed il valore ad essa associato sono gestiti dal nodo successivo della chiave in senso orario sull'anello Chord





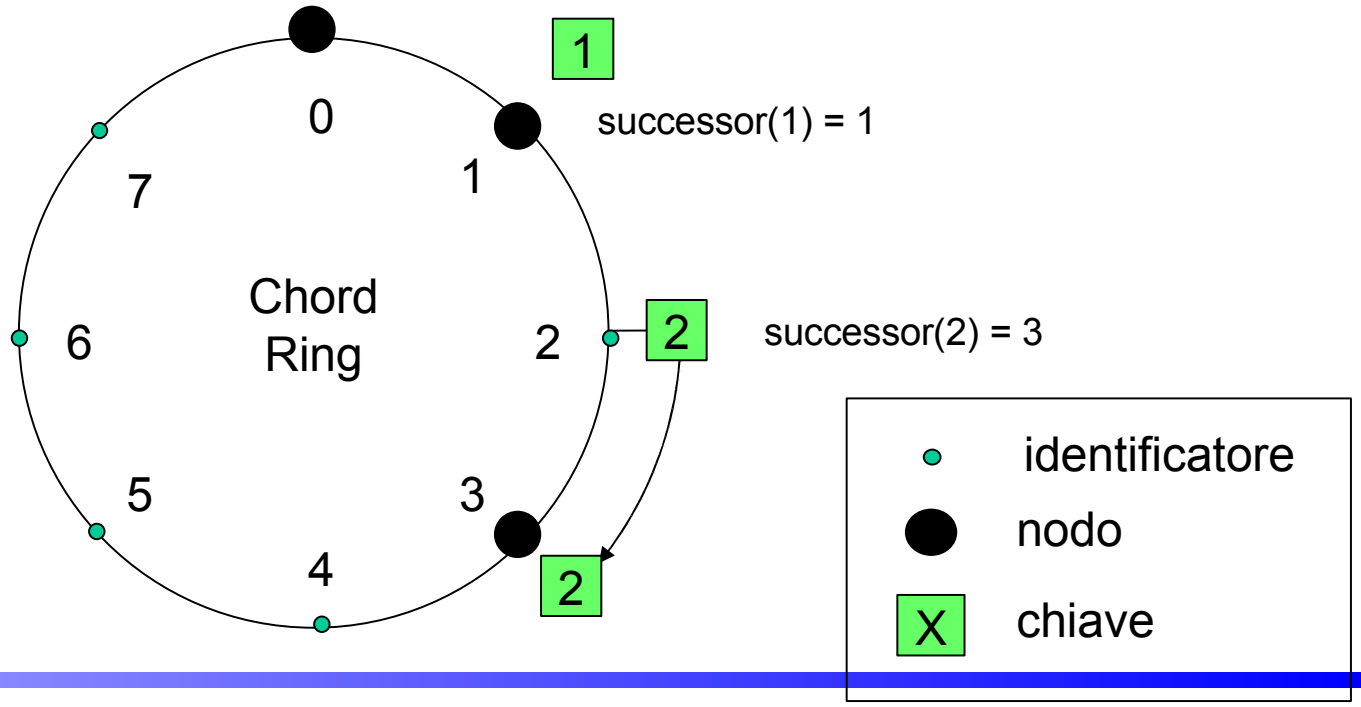
CHORD: LA TOPOLOGIA

- L'anello contiene chiavi ed IDs, i.e., tutta l'aritmetica è modulo 2^{160}
- La coppia (chiave, valore) è gestita dal nodo successivo della chiave in senso orario sull'anello Chord



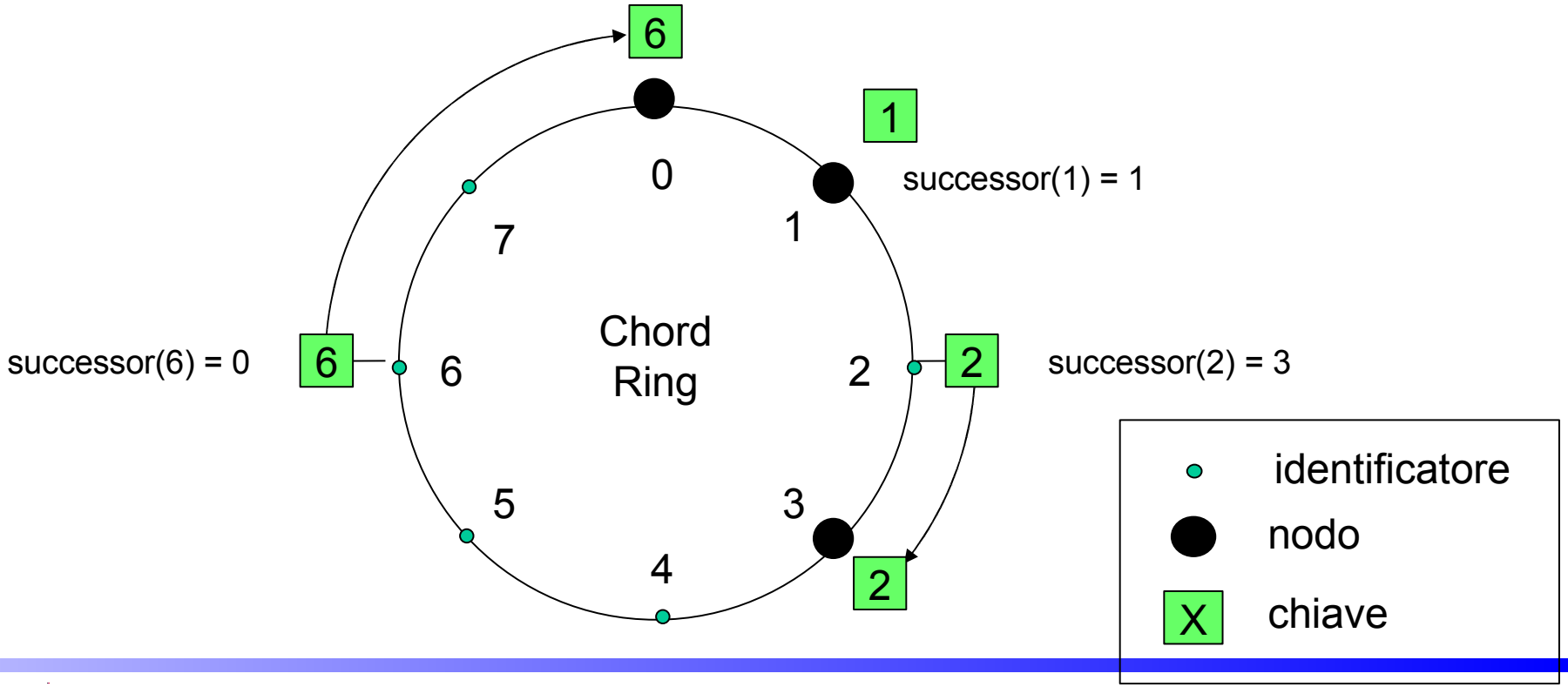
CHORD: LA TOPOLOGIA

- L'anello contiene chiavi ed IDs, i.e., tutta l'aritmetica è modulo 2^{160}
- La coppia (chiave, valore) è gestita dal nodo successivo della chiave in senso orario sull'anello Chord



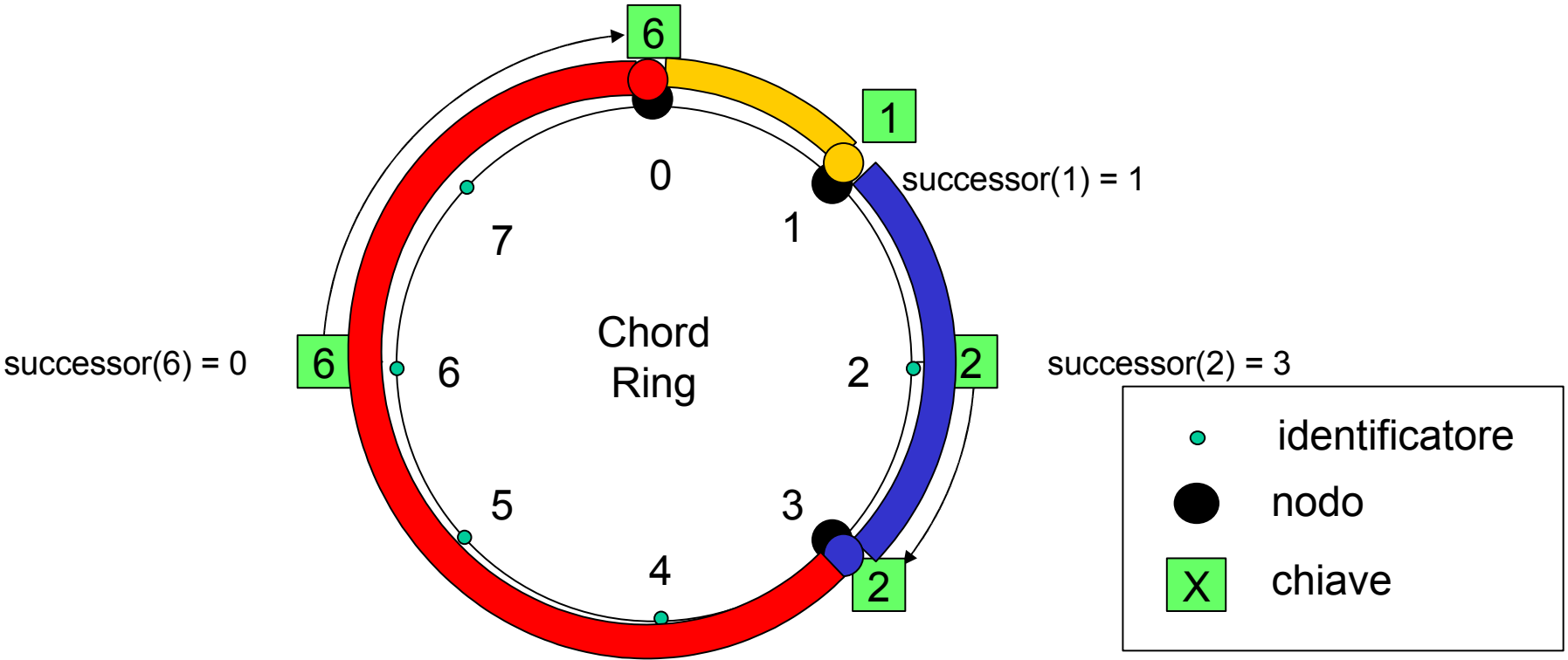
CHORD: LA TOPOLOGIA

- L'anello contiene chiavi ed IDs, i.e., tutta l'aritmetica è modulo 2^{160}
- La coppia (chiave, valore) è gestita dal nodo successivo della chiave in senso orario sull'anello Chord



CHORD: LA TOPOLOGIA

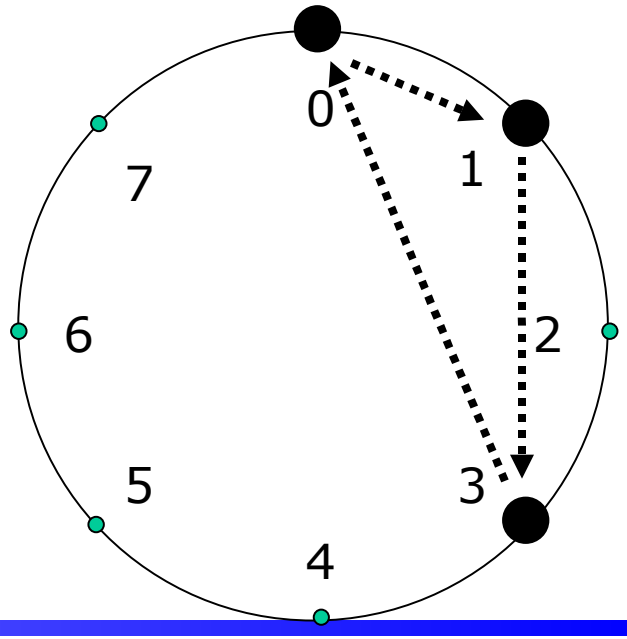
- L'anello contiene chiavi ed IDs, i.e., tutta l'aritmetica è modulo 2^{160}
- La coppia (chiave, valore) è gestita dal nodo successivo della chiave in senso orario sull'anello Chord



CHORD LA TOPOLOGIA

- La topologia è determinata dai links stabiliti tra i nodi dell'anello
  - I links memorizzati in ogni nodo rappresentano la conoscenza che un nodo possiede dell'anello
  - Sono memorizzati nella Routing Table di ogni nodo
- La topologia più semplice: lista circolare
  - Ogni nodo possiede un link verso il nodo successivo, in senso orario
  - In figura le linee tratteggiate rappresentano

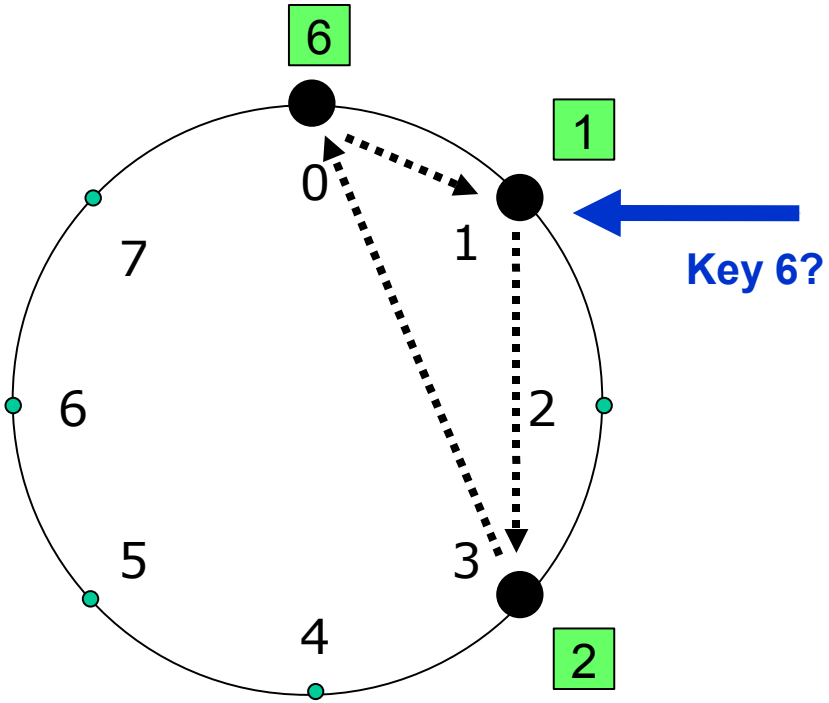
  l'overlay network definita da Chord



CHORD: IL ROUTING

- Un semplice algoritmo di Routing:
  - Ogni nodo possiede solo un link verso il suo successore
  - Inoltra la query per la chiave x al suo successore finchè non individua successor(x)
  - Restituisce i risultati della query
- Vantaggi:
  - Semplice
  - Utilizza poche informazioni
- Svantaggi:
  - Bassa efficienza:
    in media O(1/2 * N), con N nodi
  - Il fallimento di un nodo interrompe

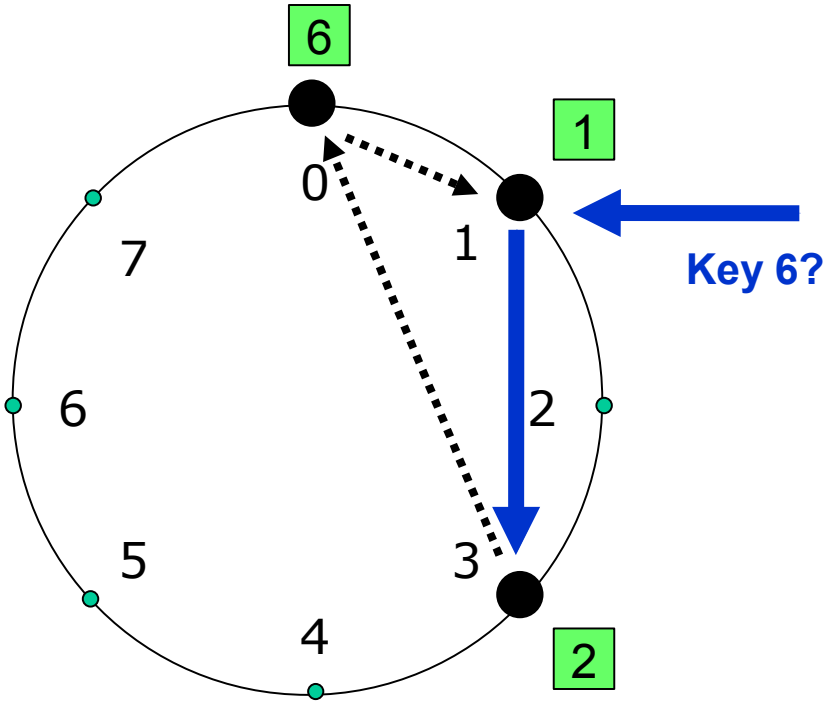
  i collegamenti sull'anello



CHORD: IL ROUTING

- Un semplice algoritmo di Routing:
  - Ogni nodo possiede solo un link verso il suo successore
  - Inoltra la query per la chiave x al suo successore finchè non individua successor(x)
  - Restituisce i risultati della query
- Vantaggi:
  - Semplice
  - Utilizza poche informazioni
- Svantaggi:
  - Bassa efficienza:
    in media O(1/2 * N), con N nodi
  - Il fallimento di un nodo interrompe

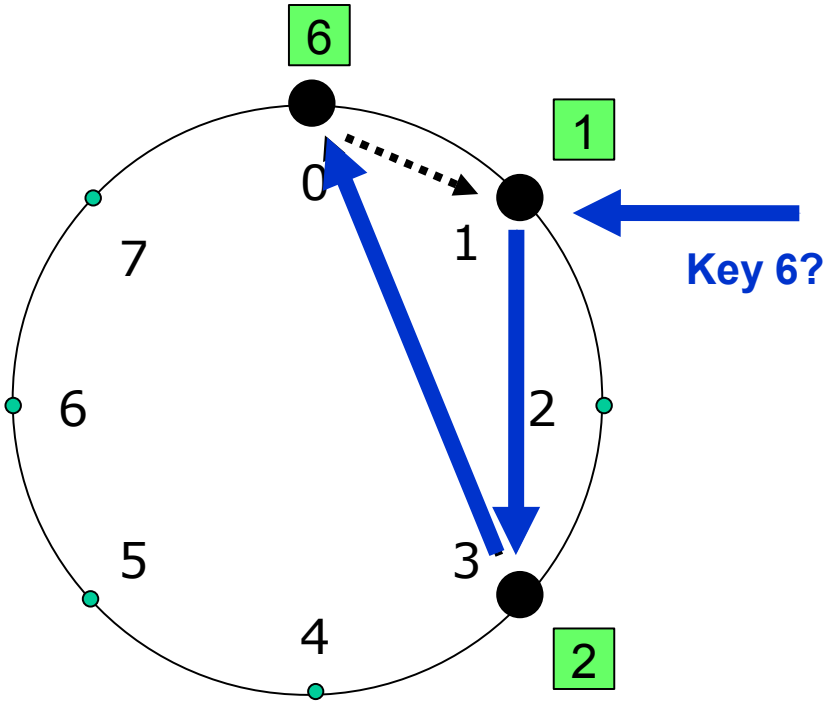
  i collegamenti sull'anello



CHORD: IL ROUTING

- Un semplice algoritmo di Routing:
  - Ogni nodo possiede solo un link verso il suo successore
  - Inoltra la query per la chiave x al suo successore finchè non individua successor(x)
  - Restituisce i risultati della query
- Vantaggi:
  - Semplice
  - Utilizza poche informazioni
- Svantaggi:
  - Bassa efficienza:
    in media O(1/2 * N), con N nodi
  - Il fallimento di un nodo interrompe

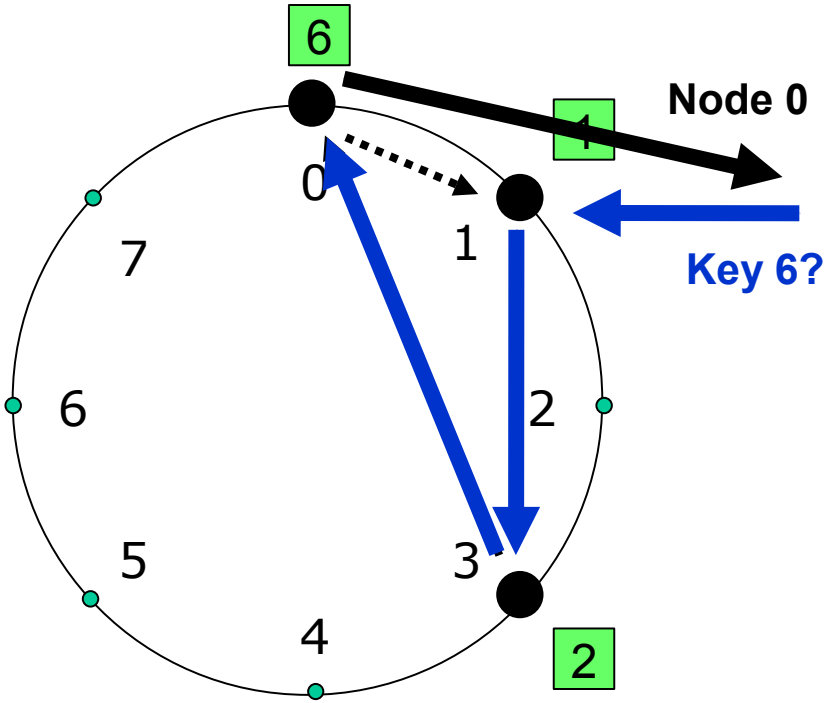
  i collegamenti sull'anello



CHORD: IL ROUTING

- Un semplice algoritmo di Routing:
  - Ogni nodo possiede solo un link verso il suo successore
  - Inoltra la query per la chiave x al suo successore finchè non individua successor(x)
  - Restituisce i risultati della query
- Vantaggi:
  - Semplice
  - Utilizza poche informazioni
- Svantaggi:
  - Bassa efficienza:
    in media O(1/2 * N), con N nodi
  - Il fallimento di un nodo interrompe

  i collegamenti sull'anello



CHORD: IL ROUTING

- Ogni nodo memorizza i links a z vicini

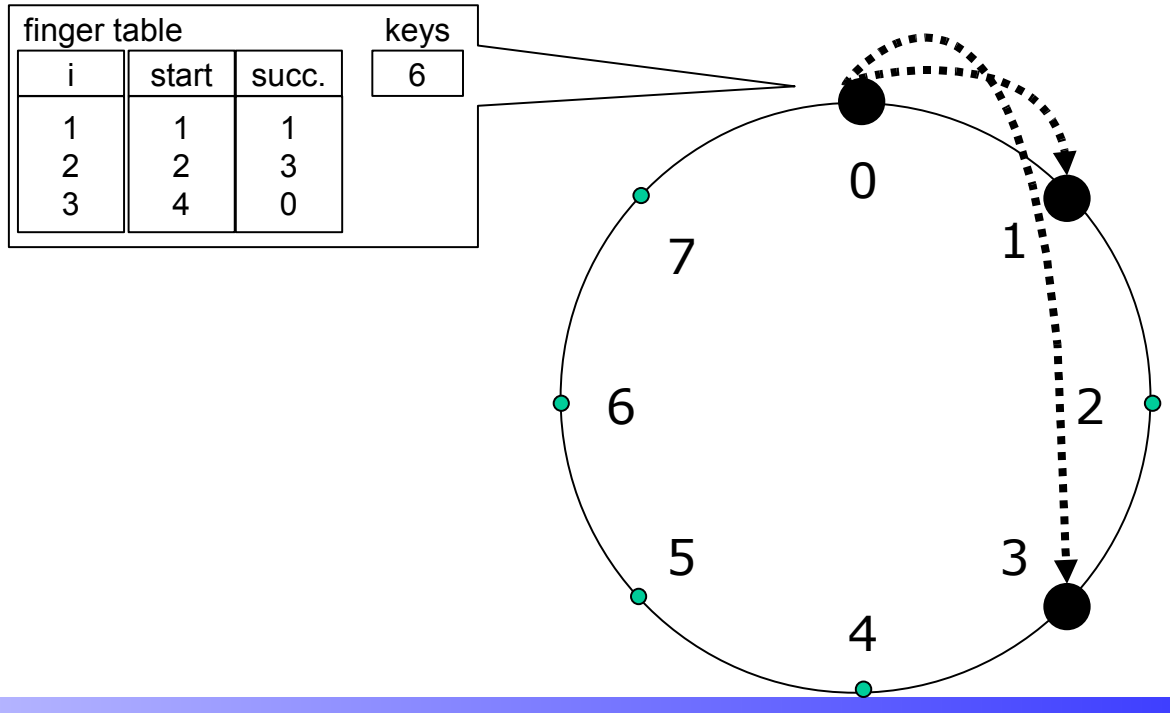
- Se z = N: si ottiene un mesh completo
  - Efficienza della ricerca: O(1), dimensione delle tabelle di routing: O(N)
  - Scalabilità limitata

- Compromesso: ogni nodo memorizza diversi di links verso alcuni nodi vicini (nell'anello) e solo alcuni verso nodi lontani
  - numero limitato di links per ogni nodo
  - routing accurato in prossimità di un nodo, più approssimato verso nodi lontani
  - algoritmo di routing:
    - inoltrare la query per una chiave k al predecessore di k più lontano, conosciuto

CHORD: IL ROUTING

Routing Table: ogni nodo utilizza una finger table

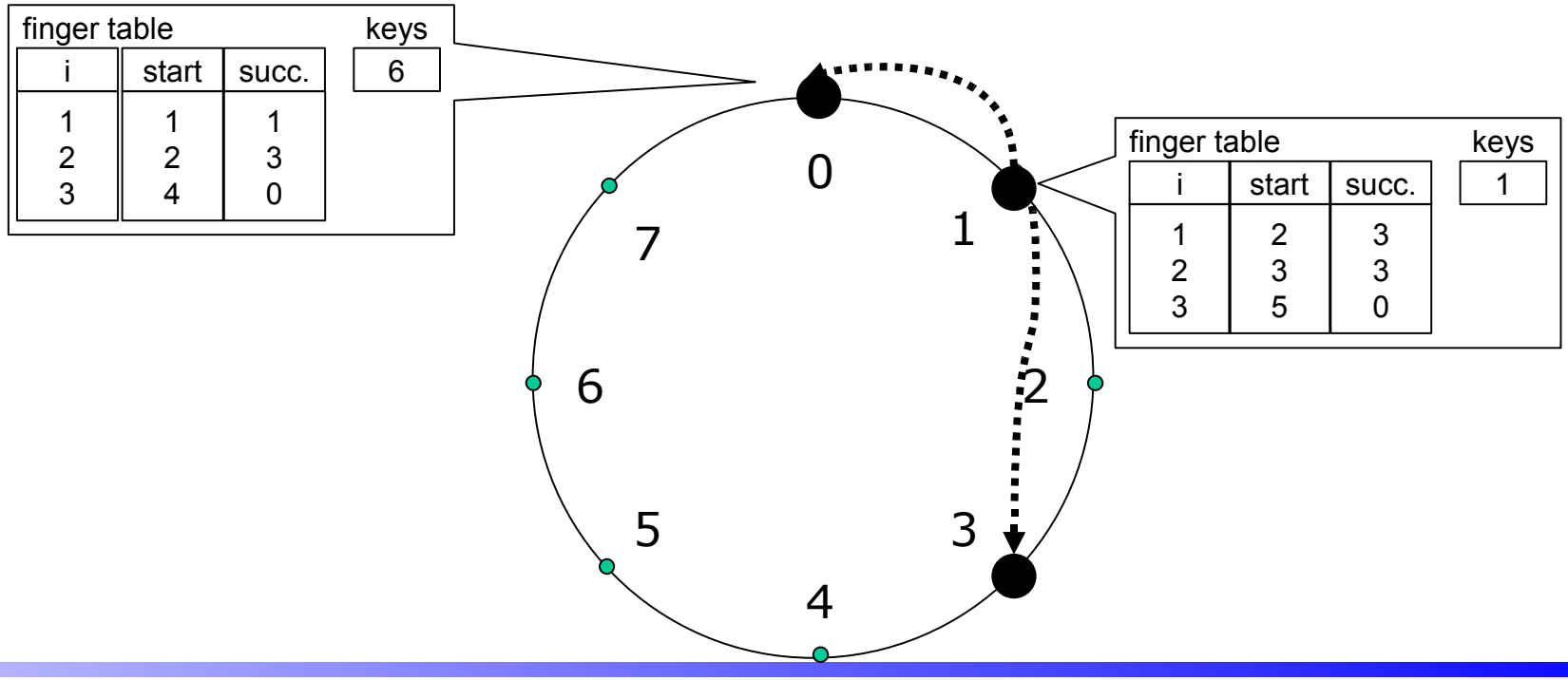
- se m è il numero di bits utilizzati per gli identificatori, la tabella contiene al massimo m links che contengono riferimenti ad altri nodi Chord
- sul nodo n: l'entrata finger[i] punta a successor($n + 2^{i-1}$), $1 \leq i \leq m$



CHORD: IL ROUTING

Routing Table: ogni nodo utilizza una finger table

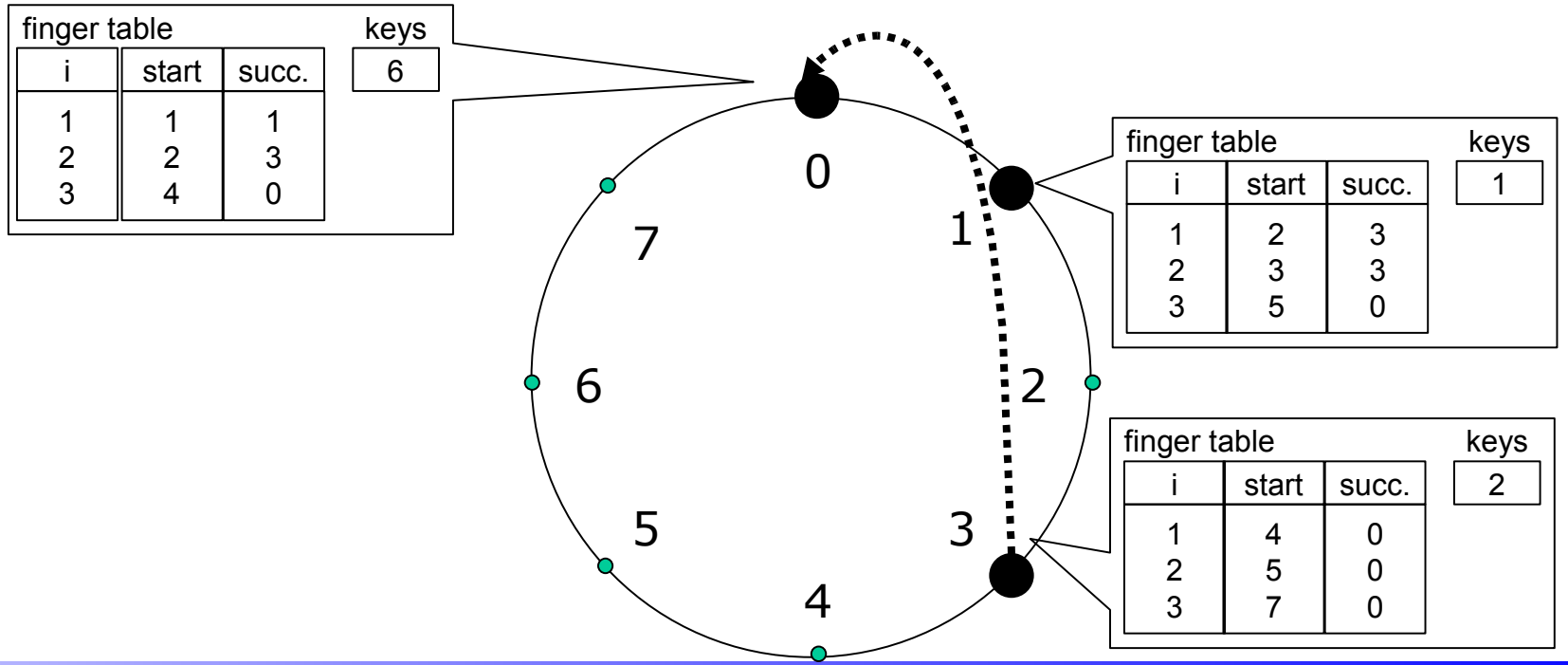
- se m è il numero di bits utilizzati per gli identificatori, la tabella contiene al massimo m links che contengono riferimenti ad altri nodi Chord
- sul nodo n: l'entrata finger[i] punta a successor($n + 2^{i-1}$), $1 \leq i \leq m$



CHORD: IL ROUTING

Routing Table: ogni nodo utilizza una finger table

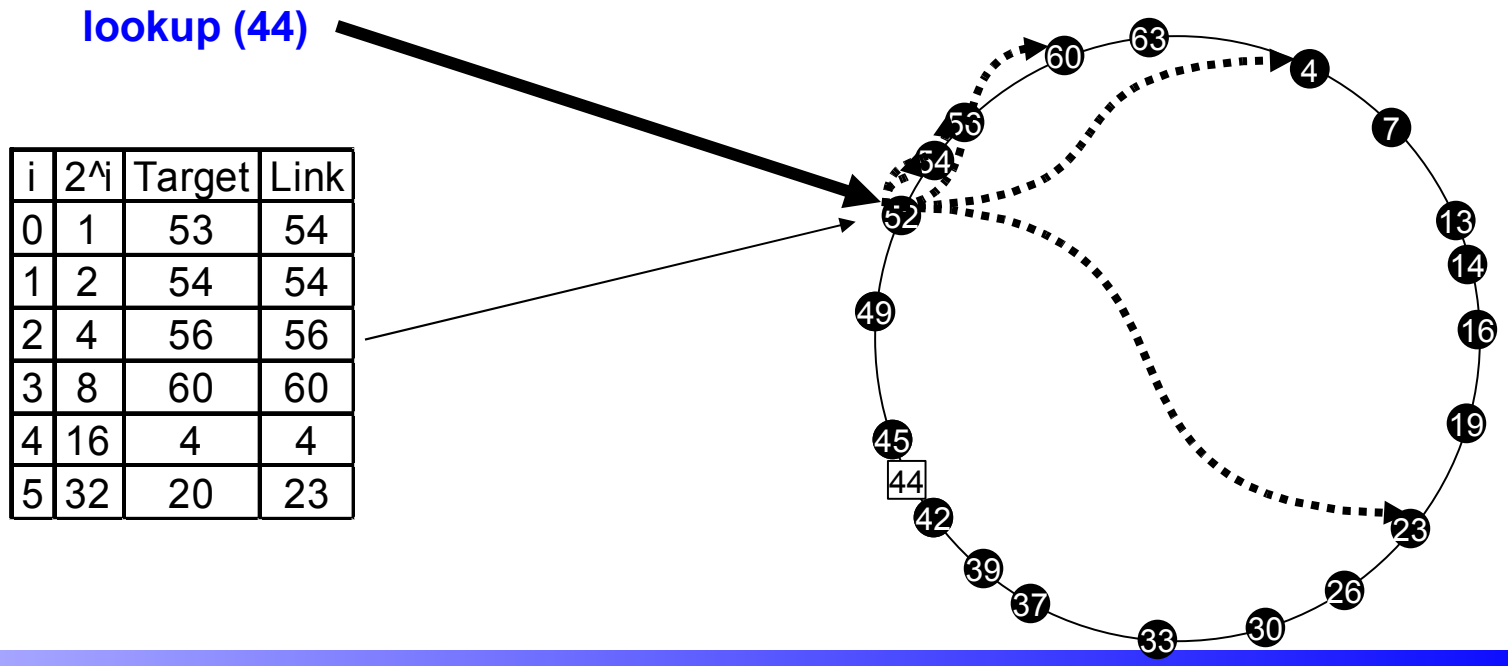
- se m è il numero di bits utilizzati per gli identificatori, la tabella contiene al massimo m links che contengono riferimenti ad altri nodi Chord
- sul nodo n: l'entrata finger[i] punta a successor($n + 2^{i-1}$), $1 \leq i \leq m$



CHORD: ROUTING

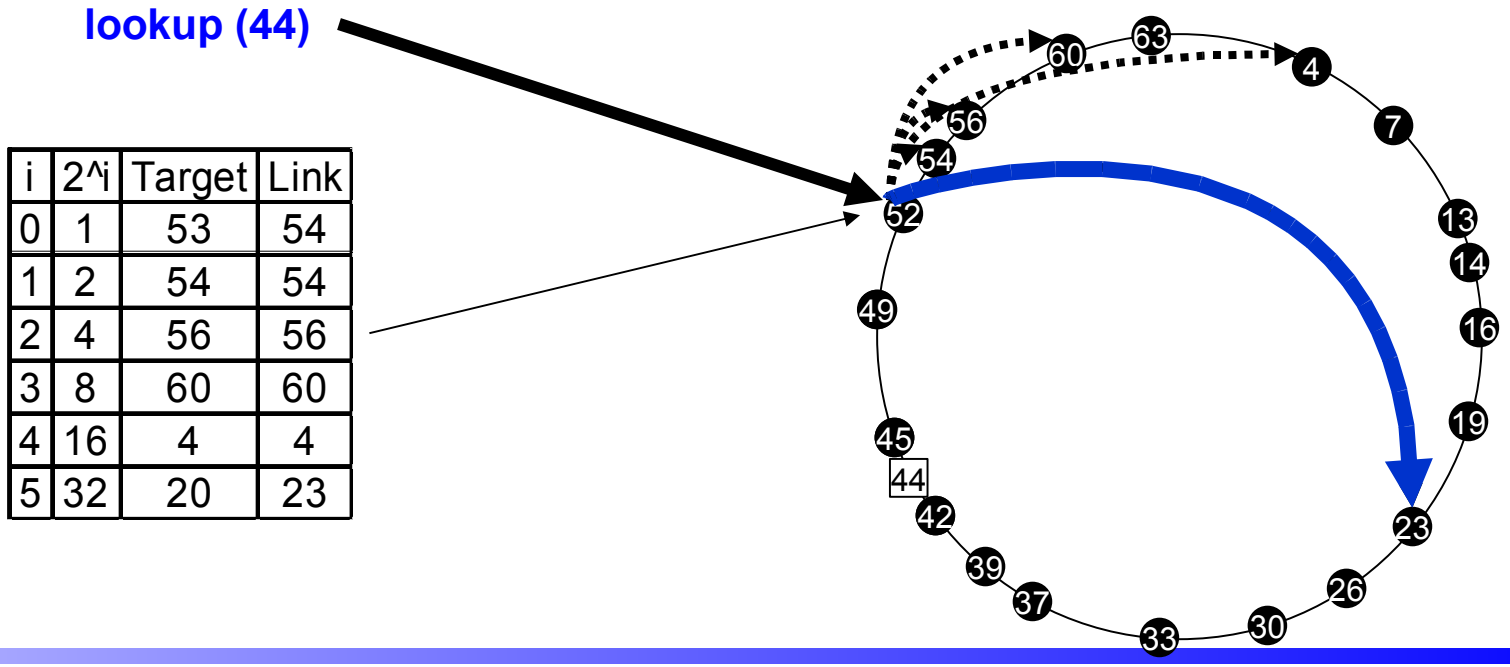
- Algoritmo di Routing : ogni nodo n propaga la query per la chiave k al finger più distante che precede k, in senso orario
- La propagazione continua fino al nodo n tale che

  n = predecessore(k) e successore(n) = successore(k)
- n restituisce il suo successore



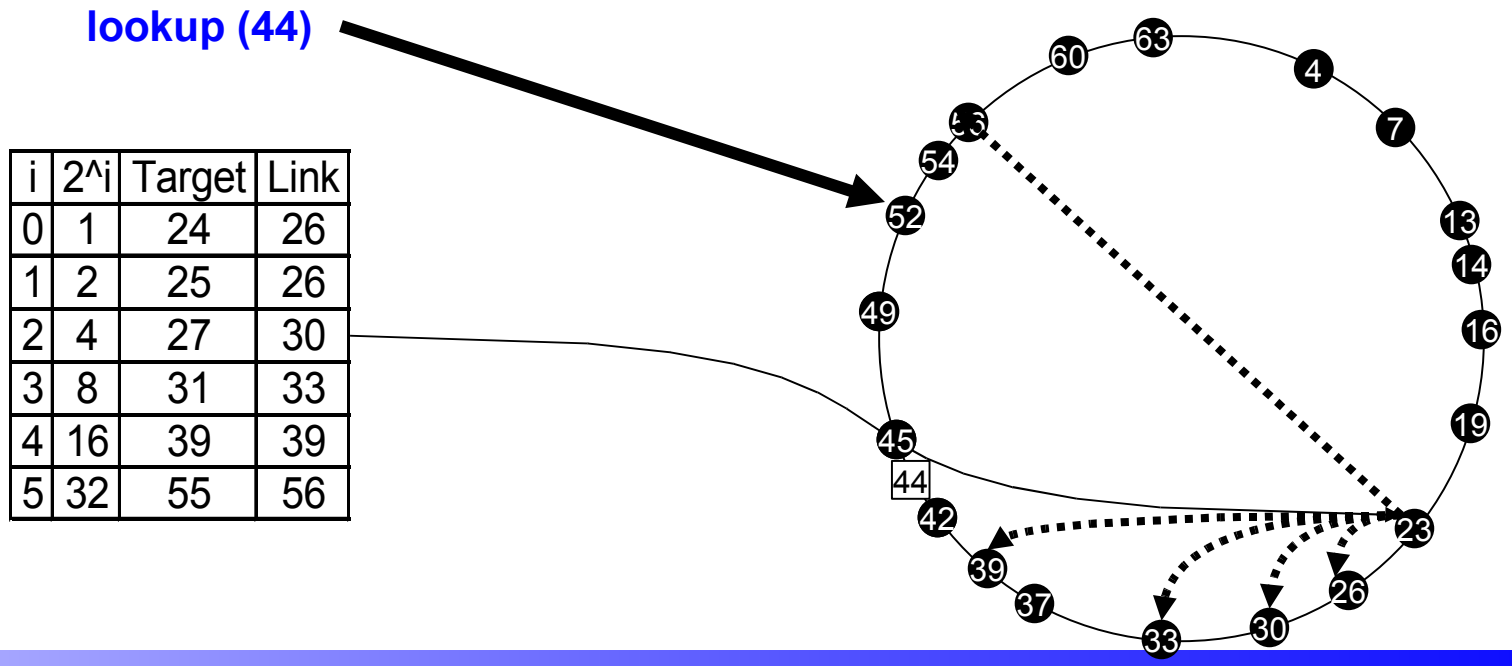
CHORD: ROUTING

- Algoritmo di Routing di Chord: ogni nodo n propaga la query per la chiave k al finger più distante che precede k, in senso orario
- La propagazione continua fino al nodo n tale che n = predecessore(k) e successore(n) = successore(k)
- n restituisce il suo successore



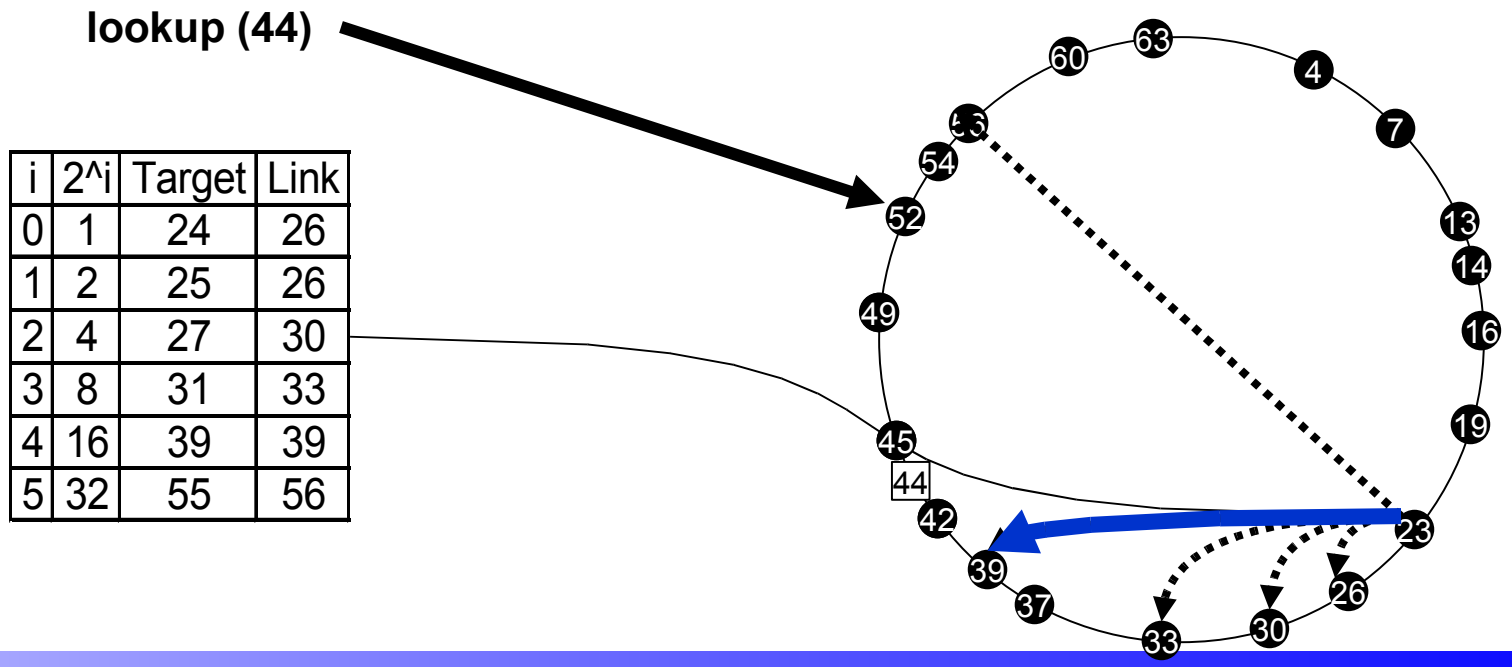
CHORD: ROUTING

- Algoritmo di Routing di Chord: ogni nodo n propaga la query per la chiave k al finger più distante che precede k, in senso orario
- La propagazione continua fino al nodo n tale che n = predecessore(k) e successore(n) = successore(k)
- n restituisce il suo successore



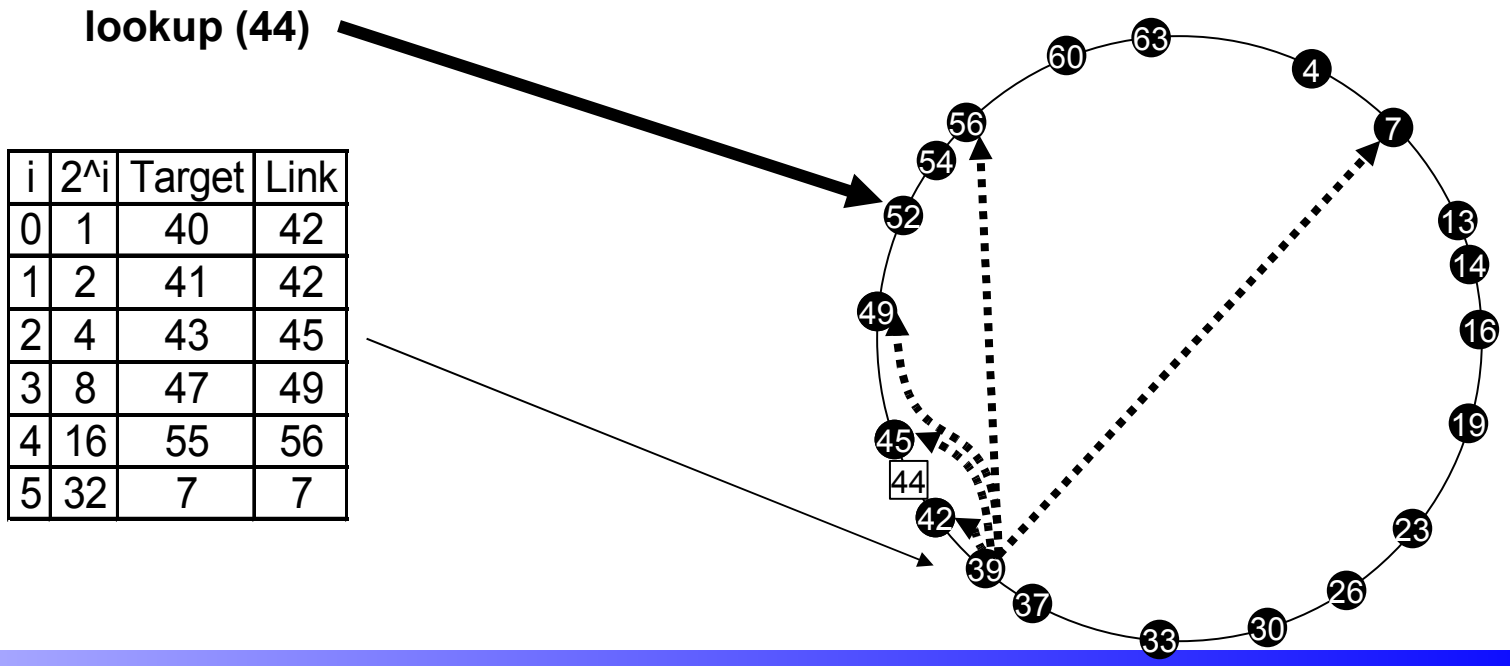
CHORD: ROUTING

- Algoritmo di Routing di Chord: ogni nodo n propaga la query per la chiave k al finger più distante che precede k, in senso orario
- La propagazione continua fino al nodo n tale che n = predecessore(k) e successore(n) = successore(k)
- n restituisce il suo successore



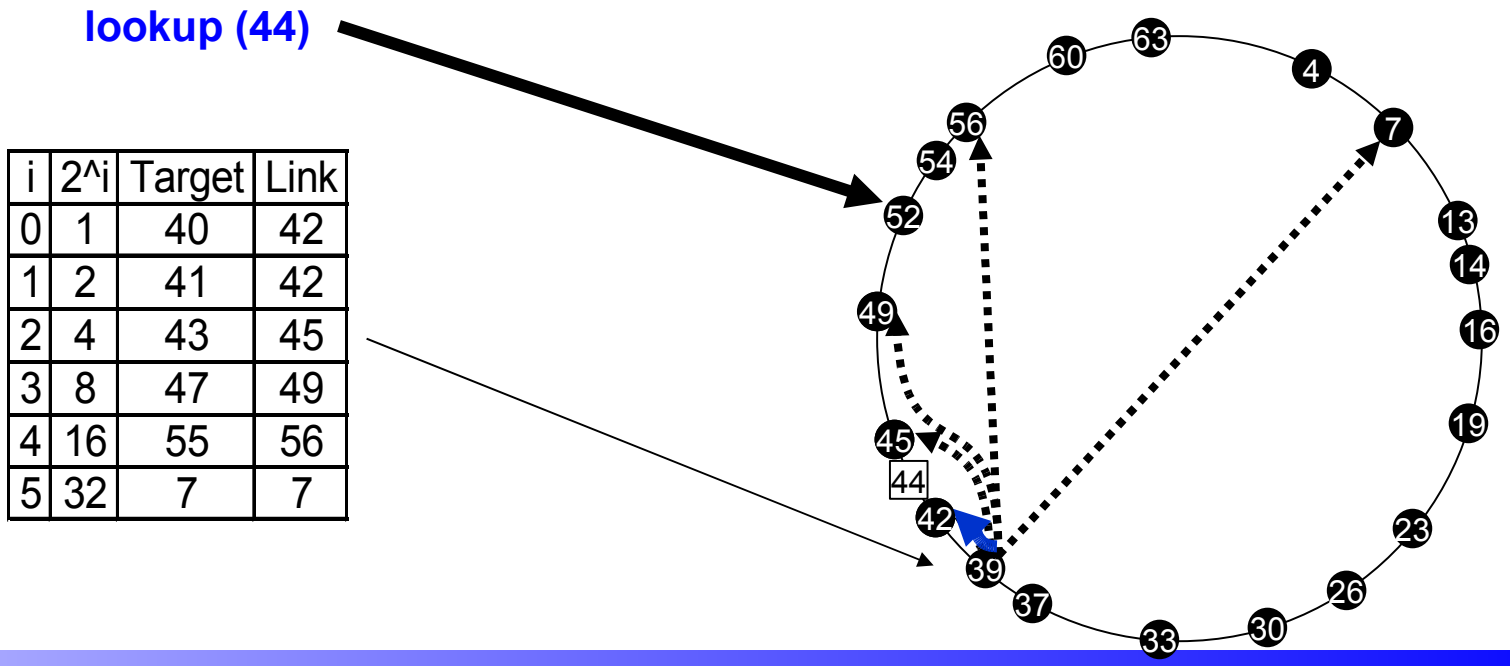
CHORD: ROUTING

- AAlgoritmo di Routing di Chord: ogni nodo n propaga la query per la chiave k al finger più distante che precede k, in senso orario
- La propagazione continua fino al nodo n tale che n = predecessore(k) e successore(n) = successore(k)
- n restituisce il suo successore



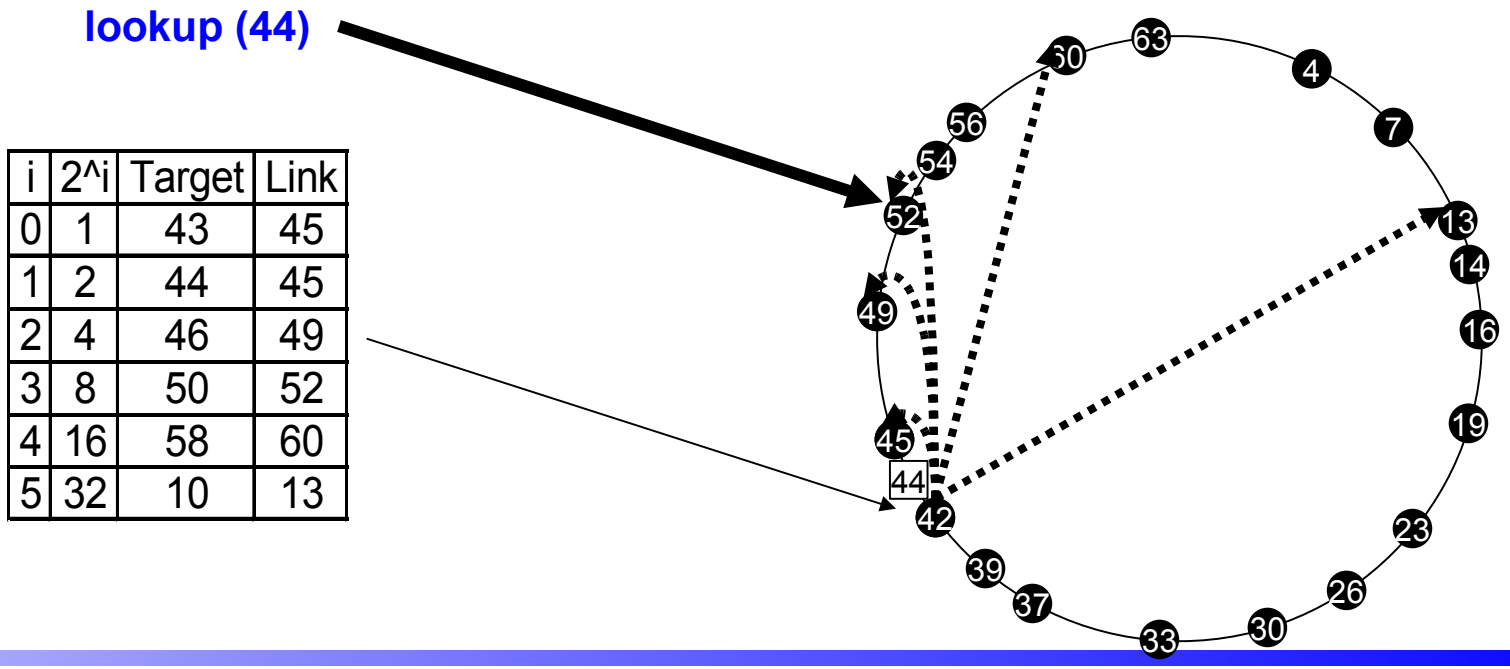
CHORD: ROUTING

- Algoritmo di Routing di Chord: ogni nodo n propaga la query per la chiave k al finger più distante che precede k, in senso orario
- La propagazione continua fino al nodo n tale che n = predecessore(k) e successore(n) = successore(k)
- n restituisce il suo successore



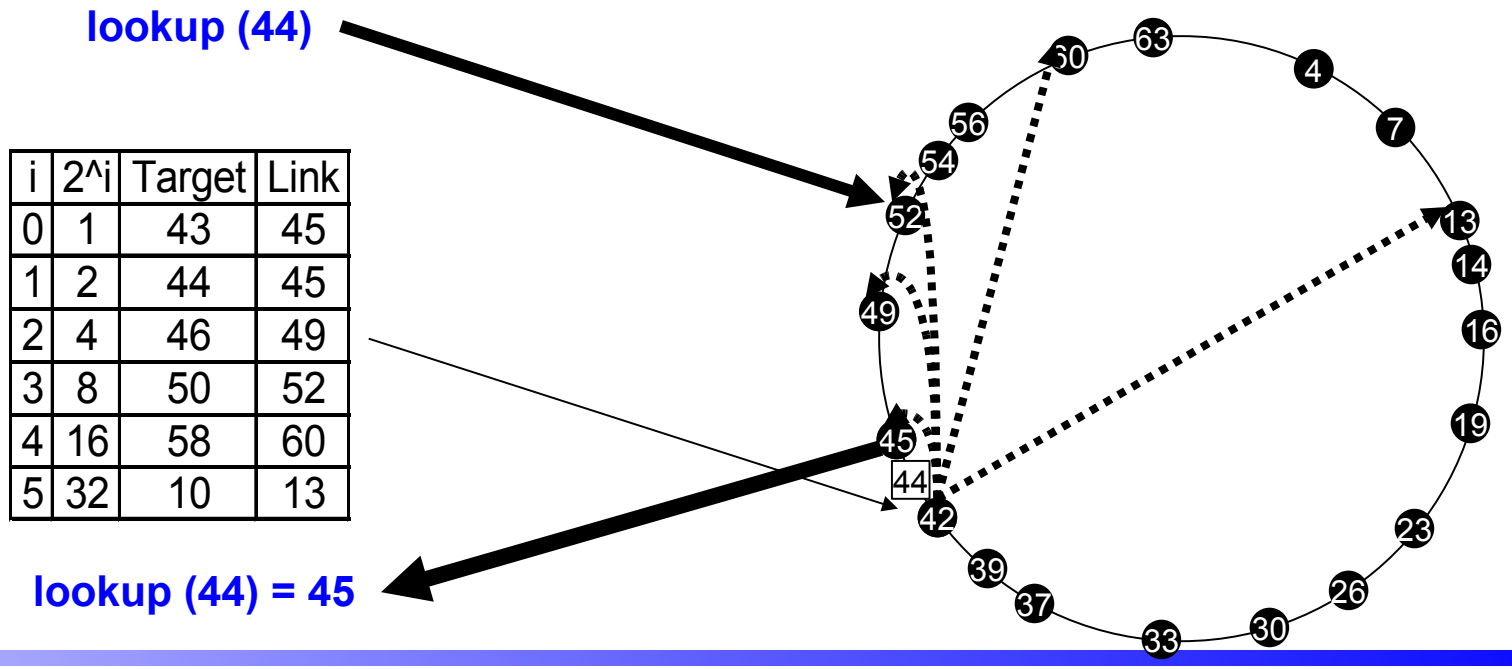
CHORD: ROUTING

- Algoritmo di Routing di Chord: ogni nodo n propaga la query per la chiave k al finger più distante che precede k, in senso orario
- La propagazione continua fino al nodo n tale che n = predecessore(k) e successore(n) = successore(k)
- n restituisce il suo successore



CHORD: ROUTING

- Algoritmo di Routing di Chord: ogni nodo n propaga la query per la chiave k al finger più distante che precede k, in senso orario
- La propagazione continua fino al nodo n tale che n = predecessore(k) e successore(n) = successore(k)
- n restituisce il suo successore



i	2^i	Target	Link
0	1	43	45
1	2	44	45
2	4	46	49
3	8	50	52
4	16	58	60
5	32	10	13

CHORD: L'ALGORITMO DI ROUTING

```
n.find_successor(id)
    if (id ∈(n, successor]
     return successor
    else
       n' = closest_preceding_node(id);
       return n'.find_successor(id)

n.closest_preceding_node(id)
     for i=m downto 1
     if finger[i] ∈(n,id)
          return finger[i]
```

- successor = nodo successore sull'anello
- n.nomefunzione = invocazione di procedura remota sul nodo n

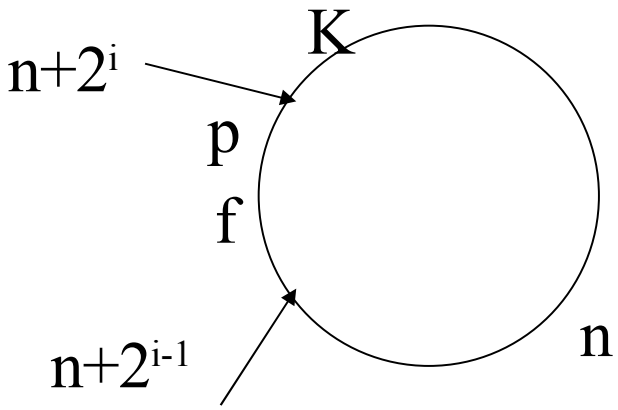
CHORD: LA COMPLESSITA' DEL ROUTING

Teorema: Per una rete Chord con identificatori di m bits, il numero di nodi che devono essere contattati per effettuare il routing di una query è al massimo m

Dimostrazione:

- Supponiamo che un nodo n debba ricercare la chiave k e che $p \neq n$ sia il predecessore di k sull'anello
- Consideriamo la finger table di n. Esiste un i tale che $n+2^{i-1} \leq p < n+2^i$
- Poiché esiste almeno il nodo p nell'intervallo $[n+2^{i-1}, n+2^i)$, finger[i] punta a quel nodo o ad un suo predecessore appartenente a quell'intervallo
- Supponiamo che finger[i]= f, dove f è un predecessore di p
- Calcoliamo la distanza tra n ed f, d(n,f) e quella tra f e p, d(f,p) e dimostriamo che $d(f,p) \leq 1/2\ d(n,p)$
- Intuizione: la distanza tra n e la query si dimezza ad ogni hop

CHORD: LA COMPLESSITA' DEL ROUTING



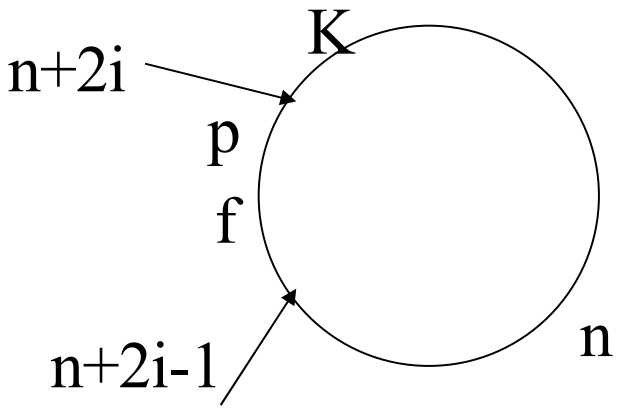
- p = predecessore della chiave K, $n+2^i$, $n+2^{i-1}$ corrispondono a due entrate della finger table di n.

- Definiamo la distanza tra due nodi come il numero di identificatori Chord compresi tra i due nodi

- d (n, f) ≥ 2^{i-1}. Infatti f ≥ $n+2^{i-1}$, poiché f è l'i-simo finger. Inoltre d(n,f) = 2^{i-1} se e solo se f coincide con il primo estremo dell'intervallo

- d(f,p) ≤ 2^{i-1} . Infatti d(f,p) $(n+2^i) - (n+2^{i-1}) \leq 2^i - 2^{i-1} = 2^{i-1}$

CHORD: LA COMPLESSITA' DEL ROUTING



- d (n, f) ≥ 2^{i-1} e d(f,p) ≤ 2^{i-1}
- Dalle relazioni precedenti si deduce che f si trova più vicino a p che ad n
- La distanza tra f e p è al più la metà di quella tra n e p
- La distanza tra il nodo che propaga la query e p si dimezza ad ogni passo
- Poiché la distanza tra n e p è al massimo 2^m, all'inizio, entro m passi la distanza diventa 1, ovvero si è raggiunto p.

CHORD: LA COMPLESSITA' DEL ROUTING

Lemma :

Dato un qualunque intervallo di ampiezza $2^m/N$ il numero atteso di nodi che cadono in questo intervallo è, con alta probabilità, $O(\log N)$

Lemma:

La distanza minima tra due nodi Chord è, con alta probabilità almeno $2^m/N^2$

Le dimostrazioni dei lemmi richiedono risultati di calcolo delle probabilità

Teorema:

Data una query q il numero di nodi che devono essere contattati per individuare il successore di q su un anello Chord contenente N nodi è, con alta probabilità $O(\log N)$.

CHORD: LA COMPLESITA' DEL ROUTING

Teorema:

Data una query q il numero di nodi che devono essere contattati per individuare il successore di q su un anello Chord contenente N nodi è, con alta probabilità O(log N).

Dimostrazione:

- Sappiamo che la distanza, in termini di identificatori Chord tra la sorgente e destinazione si dimezza
- Supponiamo di effettuare log N hops.Dopo questi passi la distanza tra sorgente e destinazione si è ridotta a $2^m/2^{\log N} = 2^m/N$
- Dal Lemma del lucido precedente sappiamo che in un intervallo di ampiezza $2^m/N$ ci sono al più log N nodi
- Quindi effettuando altri log N passi arriviamo a destinazione
- In totale occorrono log N + O(log N) = O(log N) passi di routing

CHORD: DIMENSIONE DELLE FINGER TABLES

Teorema: La finger table di ogni nodo n contiene al massimo O(log N) entrate diverse. L'i-esimo finger, i <m-2log N è uguale al suo immediato successore con alta probabilità.

Dimostrazione:

- Abbiamo detto che la dimensione della finger table è m
- D'altra parte è possibile dimostrare che non tutti gli m fingers sono distinti
- Sappiamo che il finger i del nodo n è un successore dell'id $n+2^{i-1}$
- la distanza minima tra due nodi Chord è, con alta probabilità almeno $2^m/N^2$ (vedi lemma)
- Quindi tutti i fingers di n per cui risulta $2^{i-1}<2^m/N^2$ puntano al successore di n
- Quanti sono questi fingers? Ricaviamo la i

  $\log 2^{i-1}<\log 2^m/N^2 \Rightarrow i-1 < \log 2^m - \log N^2 \Rightarrow i-1 < m-2\log N \Rightarrow i < m - 2\log N +1$
- In totale il numero di fingers distinti che devono essere mantenuti nella finger table è m -(m-2log N) = O(log N)

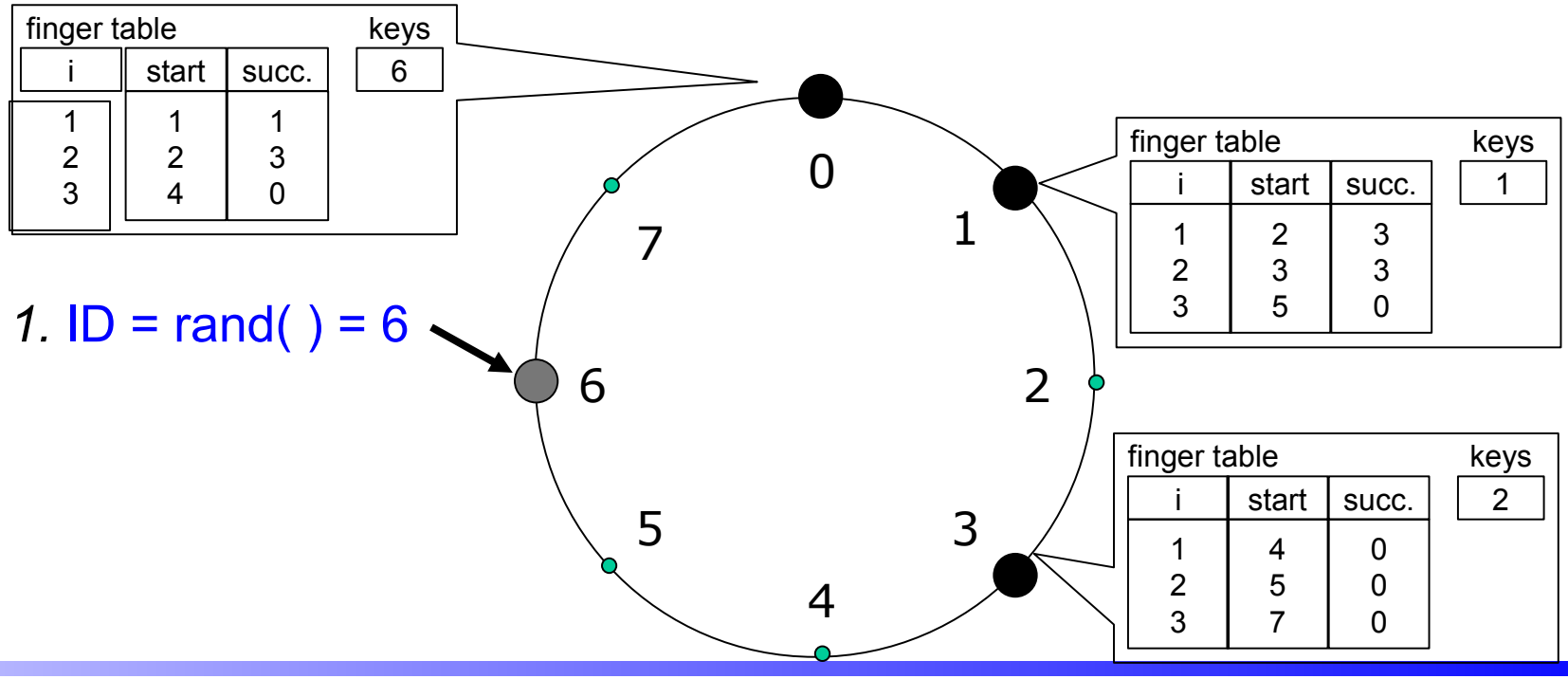
CHORD: AUTO ORGANIZZAZIONE

- Chord è in grado di gestire in modo dinamico i cambiamenti della rete
  - Fallimento di nodi
  - Fallimenti nella rete
  - Arrivo di nuovi nodi
  - Ritiro volontario dei nodi dalla rete

- Problema: mantenere consistente lo stato del sistema in presenza di cambiamenti dinamici
  - Aggiornare l'informazione necessaria per il routing
    - Correttezza del Routing: è necessario che ogni nodo mantenga aggiornato il suo successore effettivo sull'anello
    - Efficienza del Routing: dipende dall'aggiornamento tempestivo delle finger tables
  - Tolleranza ai guasti

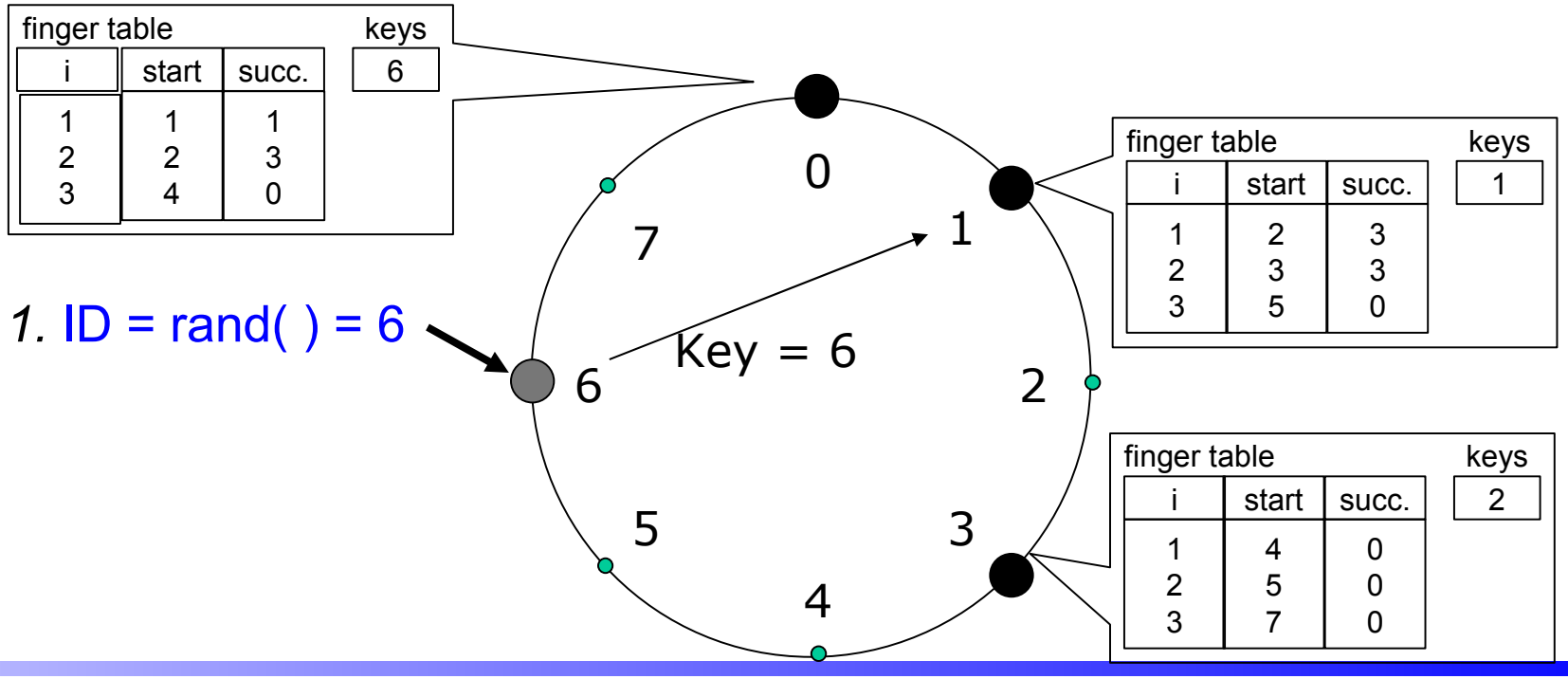
CHORD: INSERIMENTO DI NODI

- il nuovo nodo n sceglie un identificatore ID (nell'esempio 6)
- n contatta un nodo esistente sulla rete (entry point, nell'esempio 1), scelto in modo arbitrario
- individua il suo successore sull'anello e vi si aggancia (subito)
- costruisce la propria finger table (può essere lazy)



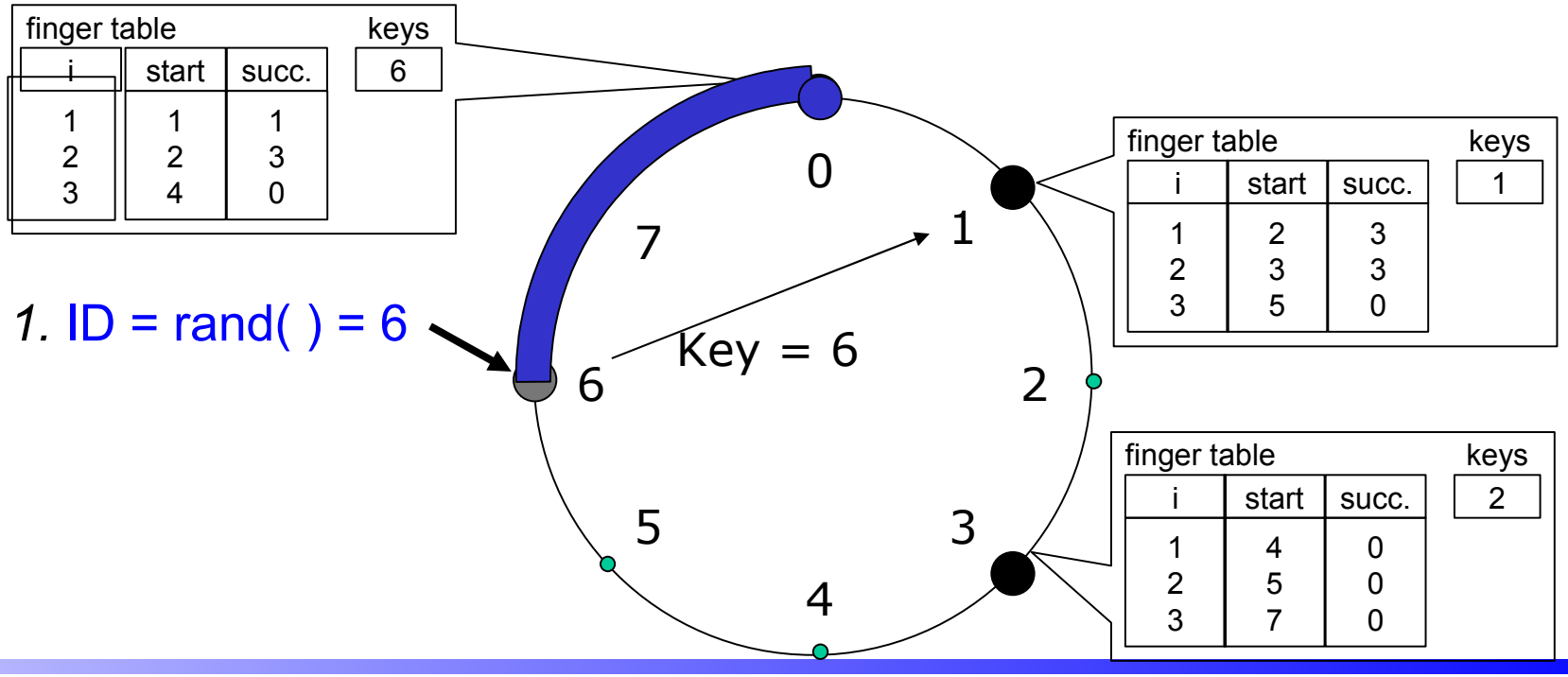
CHORD: INSERIMENTO DI NODI

- il nuovo nodo n sceglie un identificatore ID
- n contatta un nodo esistente sulla rete (entry point), scelto in modo arbitrario
- individua il suo successore sull'anello e vi si aggancia (subito)
- costruisce la propria finger table (può essere lazy)



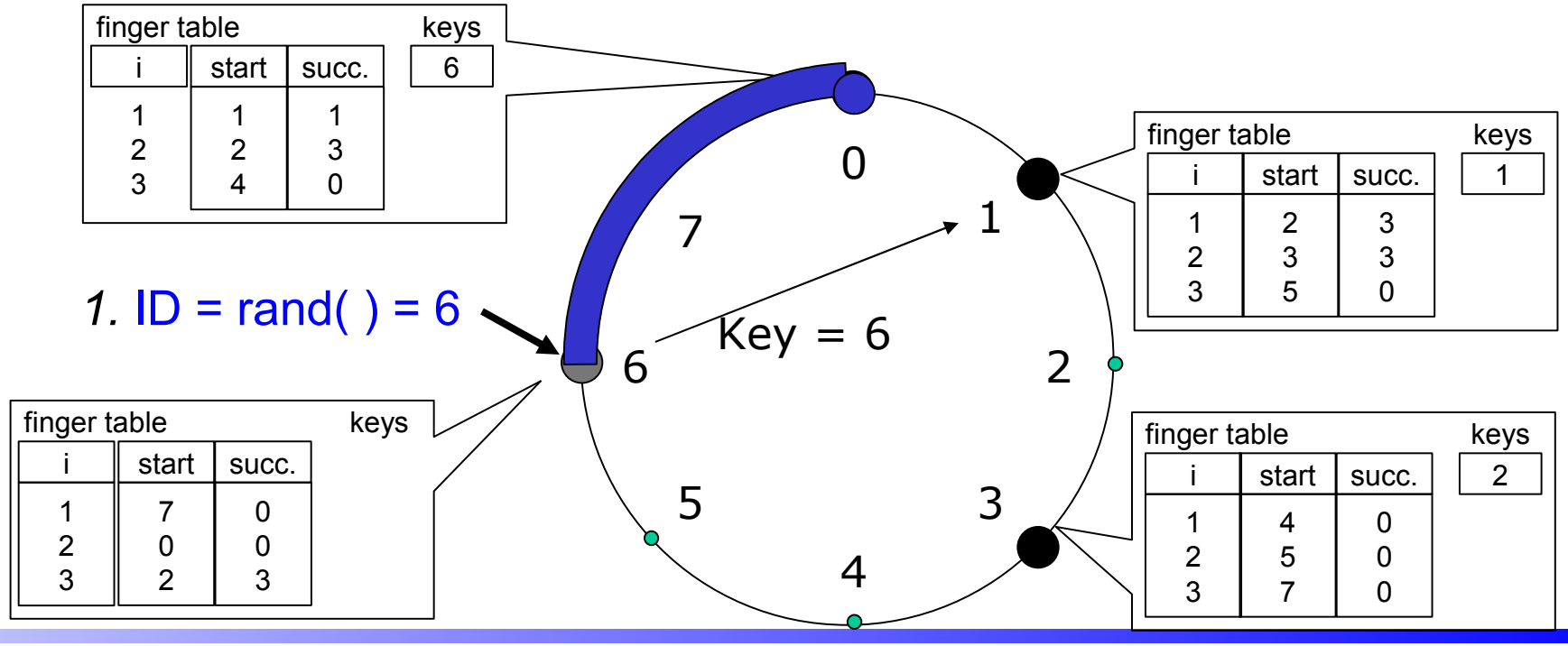
CHORD: INSERIMENTO DI NODI

- il nuovo nodo n sceglie un identificatore ID
- n contatta un nodo esistente sulla rete (entry point), scelto in modo arbitrario
- individua il suo successore sull'anello e vi si aggancia (subito)
- costruisce la propria finger table (può essere lazy)



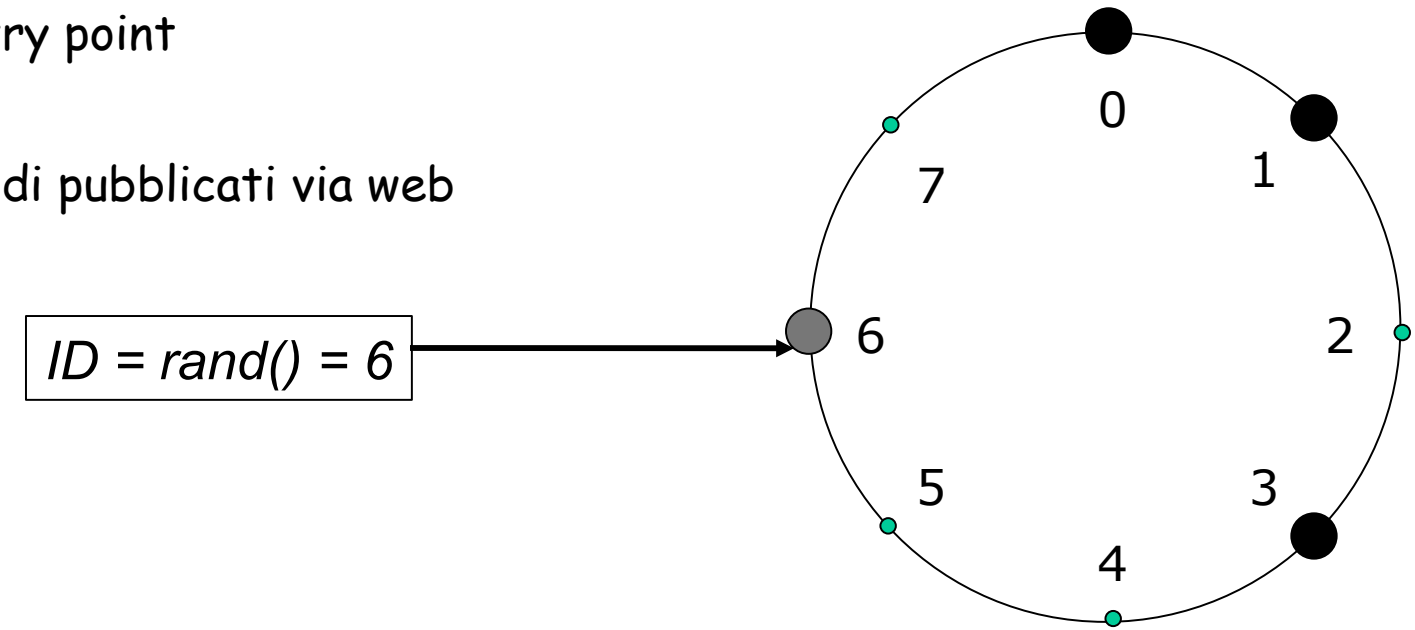
CHORD: INSERIMENTO DI NODI

- il nuovo nodo n sceglie un identificatore ID
- n contatta un nodo esistente sulla rete (entry point), scelto in modo arbitrario
- individua il suo successore sull'anello e vi si aggancia (subito)
- costruisce la propria finger table (può essere lazy)



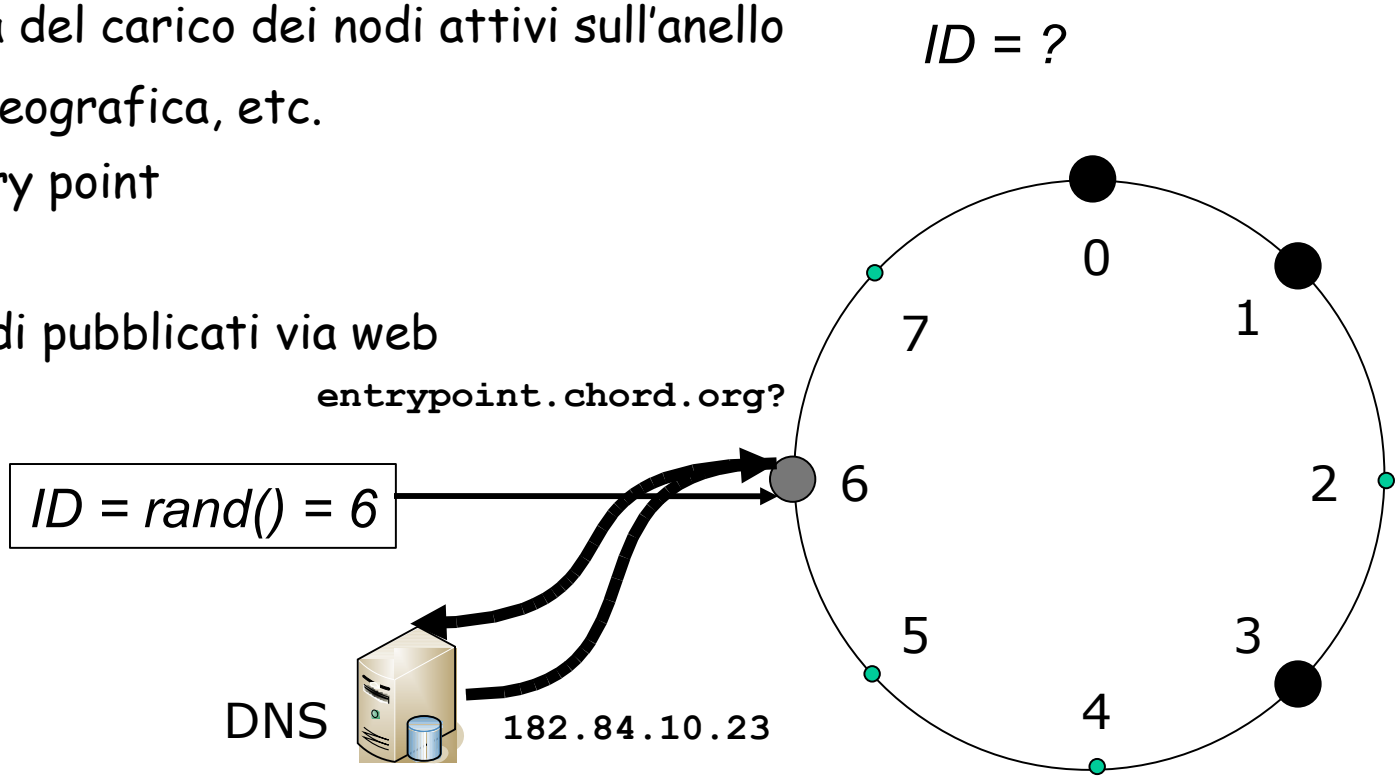
CHORD: INSERIMENTO DI NODI

1. Scelta dell'identificatore da associare al nuovo nodo
   - ID random: valido se la distribuzione sia uniforme
   - calcolo dell'hash dell'indirizzo IP + porta
   - Algoritmi di generazione degli ID basati su:
     - conoscenza del carico dei nodi attivi sull'anello
     - locazione geografica, etc.
2. Ricerca dell'entry point
   - DNS aliases
   - Indirizzi di nodi pubblicati via web
   - etc.



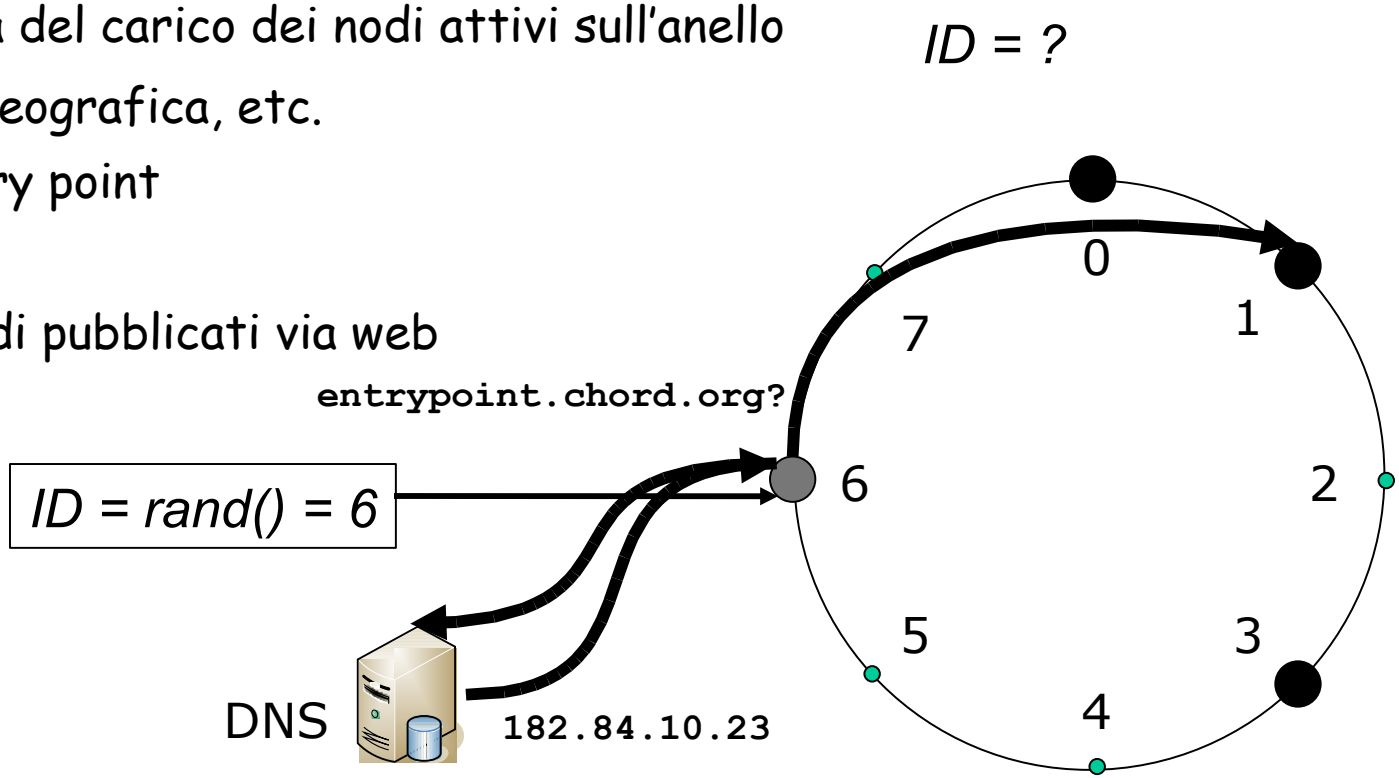
CHORD: INSERIMENTO DI NODI

1. Scelta dell'identificatore da associare al nuovo nodo
   - ID random: valido se la distribuzione sia uniforme
   - calcolo dell'hash dell'indirizzo IP + porta
   - Algoritmi di generazione degli ID basati su:
     - conoscenza del carico dei nodi attivi sull'anello
     - locazione geografica, etc.
2. Ricerca dell'entry point
   - DNS aliases
   - Indirizzi di nodi pubblicati via web
   - etc.



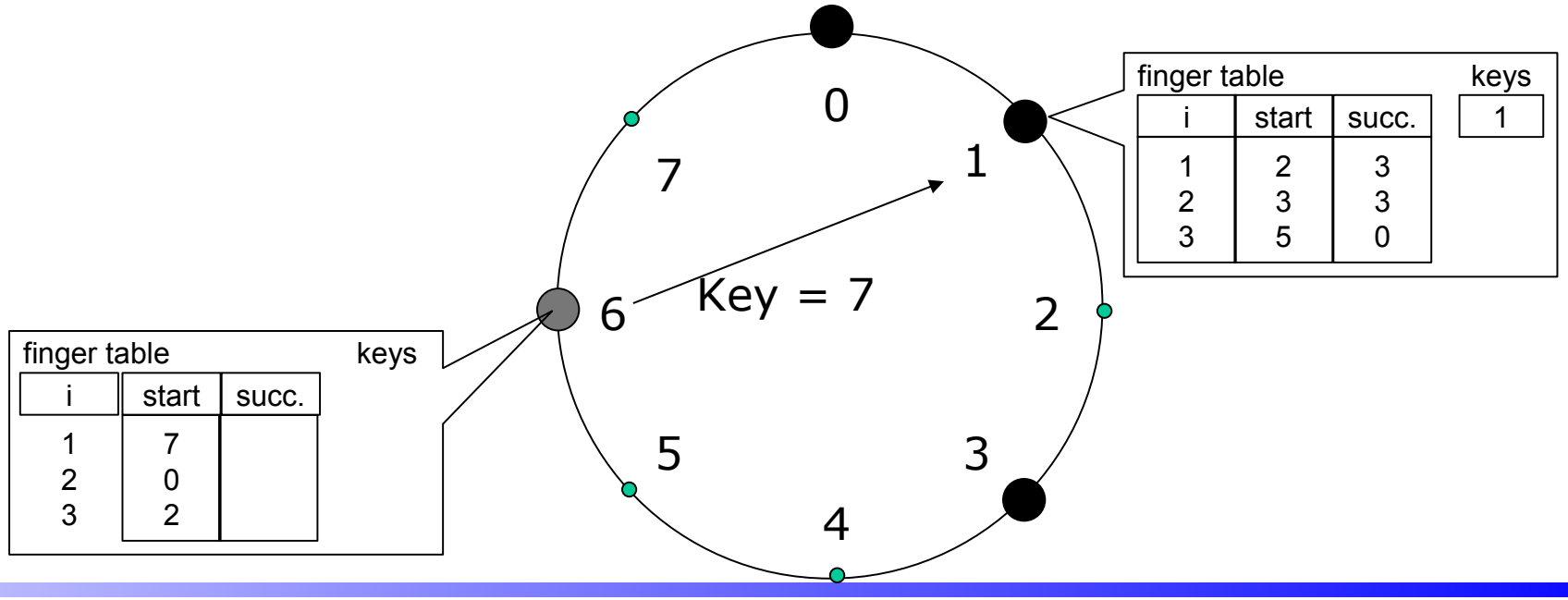
CHORD: INSERIMENTO DI NUOVI NODI

1. Scelta dell'identificatore da associare al nuovo nodo
   - ID random: valido se la distribuzione sia uniforme
   - calcolo dell'hash dell'indirizzo IP + porta
   - Algoritmi di generazione degli ID basati su:
     - conoscenza del carico dei nodi attivi sull'anello
     - locazione geografica, etc.
2. Ricerca dell'entry point
   - DNS aliases
   - Indirizzi di nodi pubblicati via web
   - etc.



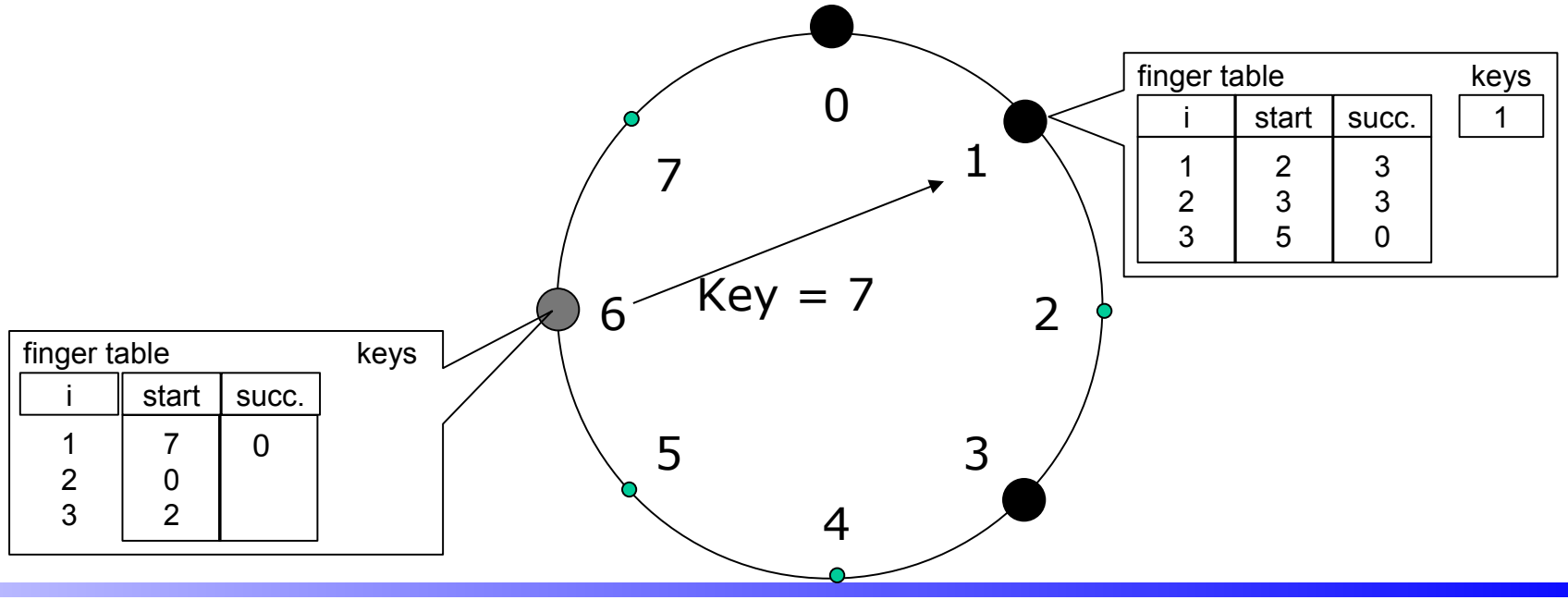
CHORD: INSERIMENTO DI NODI

3. Individuazione del nodo successore sull'anello
4. Costruzione della finger table
   - definire una struttura di m entrate
   - iterare sulle righe della finger table
   - per ogni riga: inviare una query all' entry point
   - entry point: utilizza l'algoritmo di routing standard di Chord



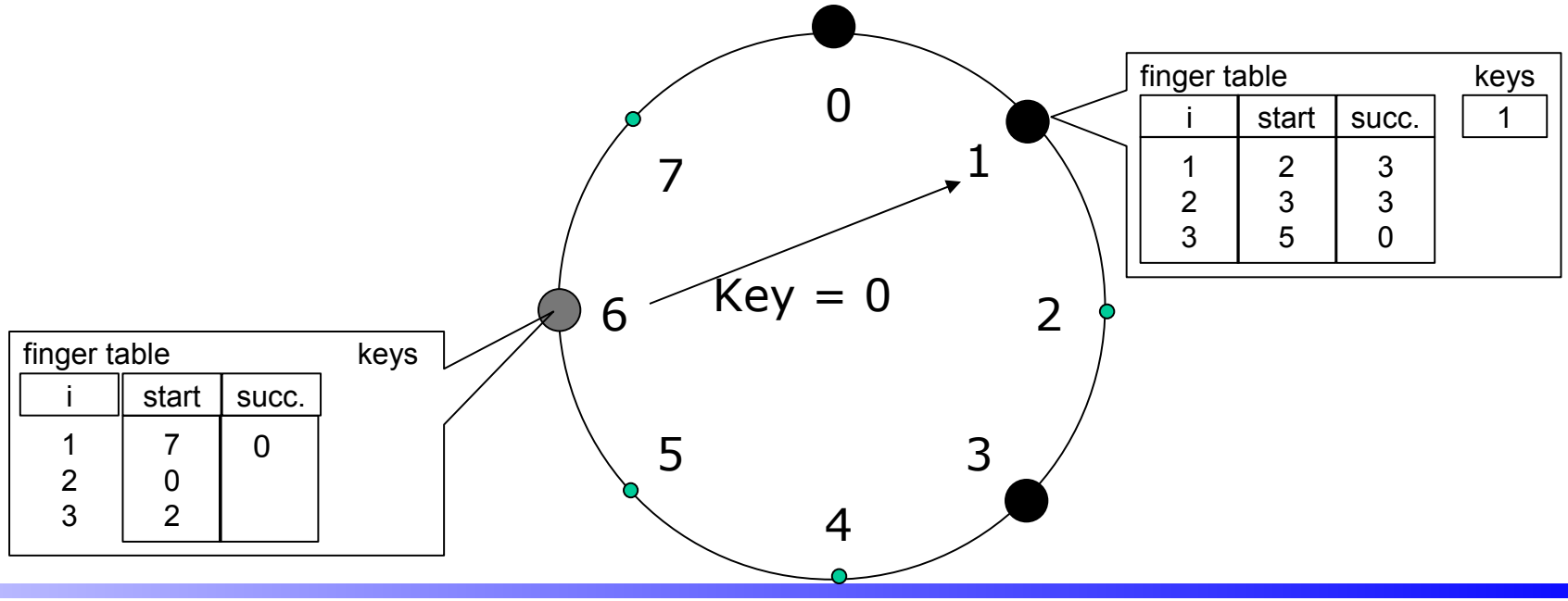
CHORD: INSERIMENTO DI NODI

3. Individuazione del nodo successore sull'anello
4. Costruzione della finger table
   - definire una struttura di log(n) entrate
   - iterare sulle righe della finger table
   - per ogni riga: inviare una query all' entry point
   - entry point: utilizza l'algoritmo di routing standard di Chord



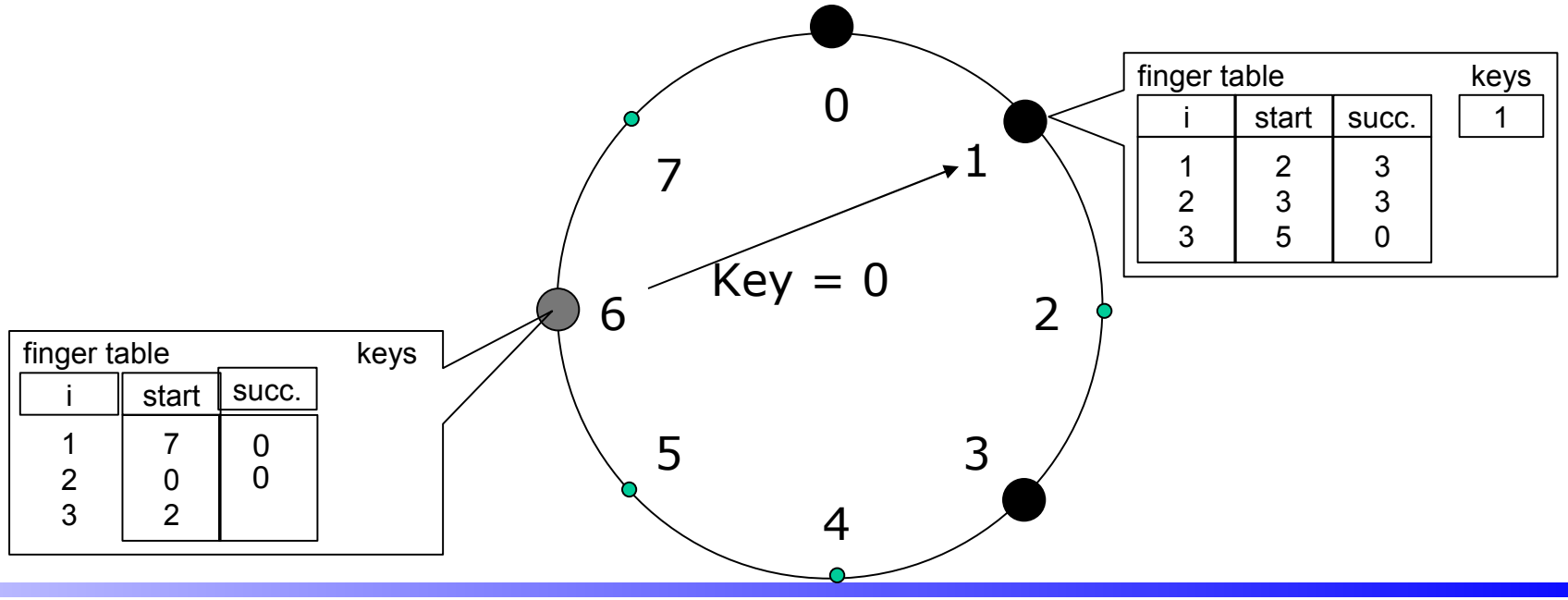
CHORD: INSERIMENTO DI NODI

3. Individuazione del nodo successore sull'anello
4. Costruzione della finger table
   - definire una struttura di log(n) entrate
   - iterare sulle righe della finger table
   - per ogni riga: inviare una query all' entry point
   - entry point: utilizza l'algoritmo di routing standard di Chord



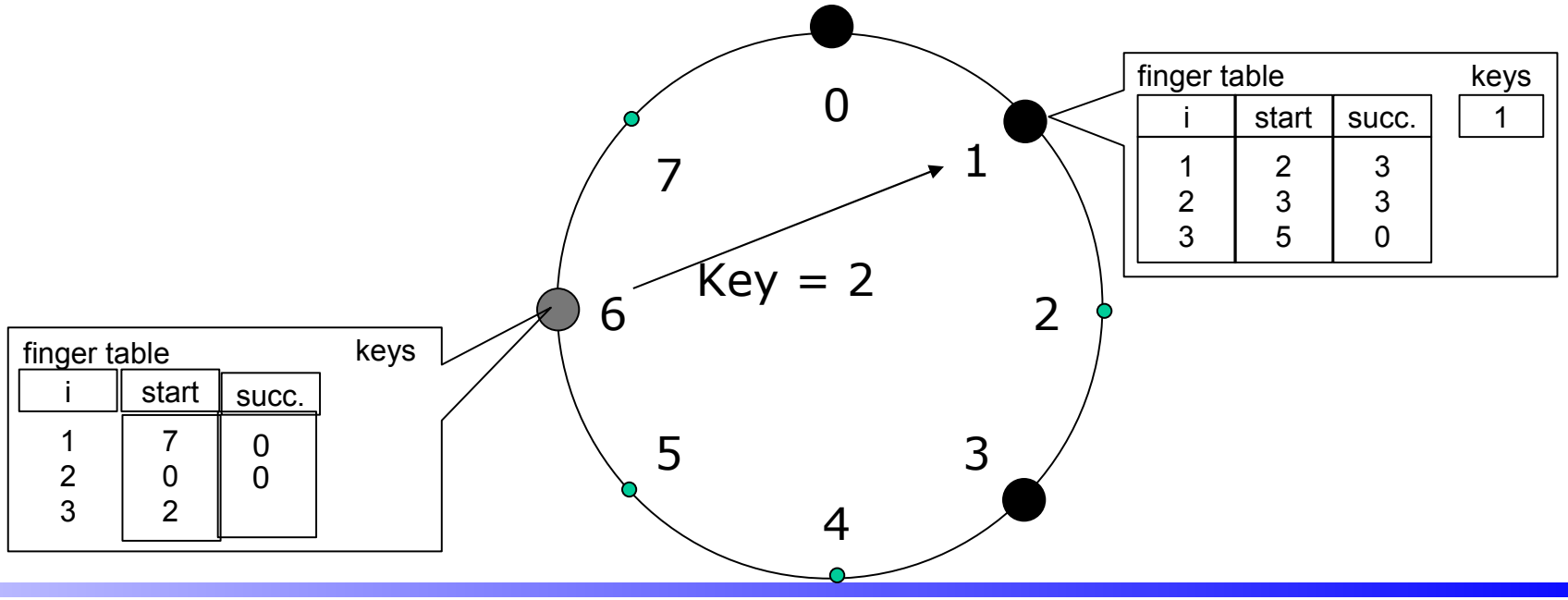
CHORD: INSERIMENTO DI NODI

3. Individuazione del nodo successore sull'anello
4. Costruzione della finger table
   - definire una struttura di log(n) entrate
   - iterare sulle righe della finger table
   - per ogni riga: inviare una query all' entry point
   - entry point: utilizza l'algoritmo di routing standard di Chord



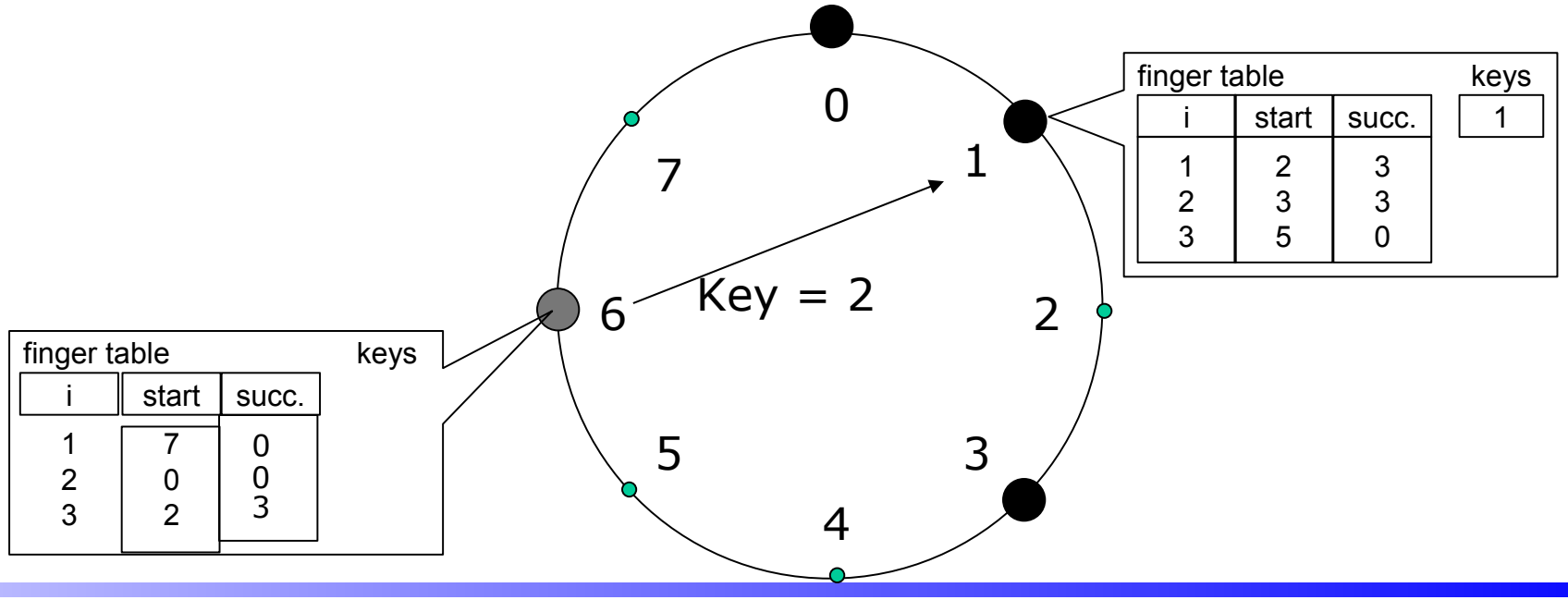
CHORD: INSERIMENTO DI NODI

3. Individuazione del nodo successore sull'anello
4. Costruzione della finger table
   - definire una struttura di log(n) entrate
   - iterare sulle righe della finger table
   - per ogni riga: inviare una query all' entry point
   - entry point: utilizza l'algoritmo di routing standard di Chord



CHORD: INSERIMENTO DI NODI

3. Individuazione del nodo successore sull'anello
4. Costruzione della finger table
   - definire una struttura di log(n) entrate
   - iterare sulle righe della finger table
   - per ogni riga: inviare una query all' entry point
   - entry point: utilizza l'algoritmo di routing standard di Chord



CHORD: INSERIMENTO DI NODI

- Creazione di un nuovo anello Chord

  n.create( )

  predecessor = nil;

  successor =n;

- Inserzione del nodo n nell'anello Chord contenente il nodo n'

  n.join(n')

  predecessor = nil:

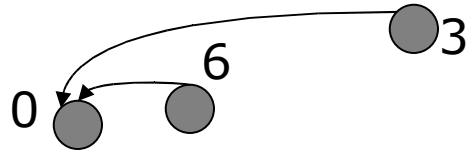
  successor = n'. find_successor (n);

- Il successore di n aggiorna l'informazione sul proprio predecessore

- Fino a che la rete non si stabilizza, solo il successore di n è consapevole della presenza di n sulla rete

CHORD:INSERIMENTO DI NODI

- Dopo i passi illustrati nei lucidi precedenti
  - il nuovo nodo n conosce il suo successore sull'anello e possiede una finger table valida, ma gli altri nodi dell'anello non sono consapevoli della presenza di n



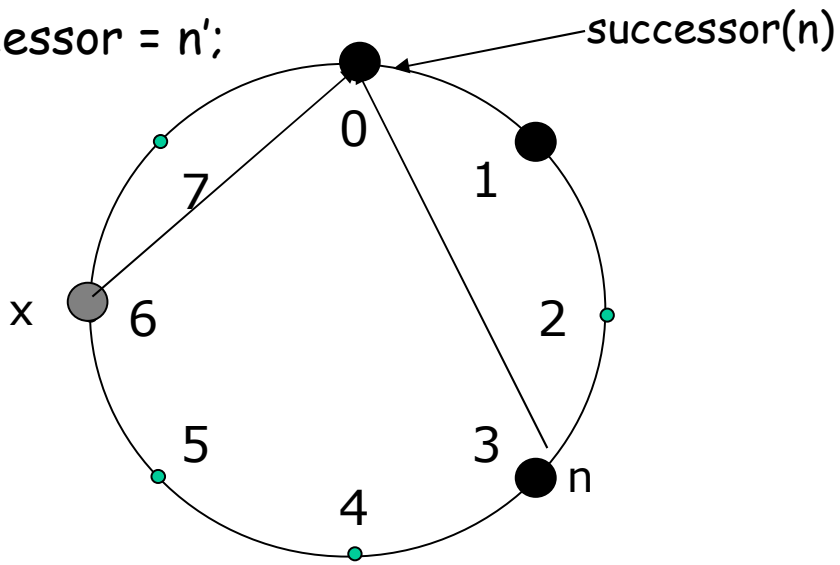
- I nodi eseguono periodicamente alcune procedure per acquisire conoscenza sui nuovi nodi che si sono inseriti nella rete

- stabilize ( ) aggiorna il puntatore al nodo successivo

- fix fingers ( ) aggiorna la finger table di ogni nodo, inserendovi eventuali riferimenti a nuovi nodi

CHORD:INSERIMENTO DI NODI

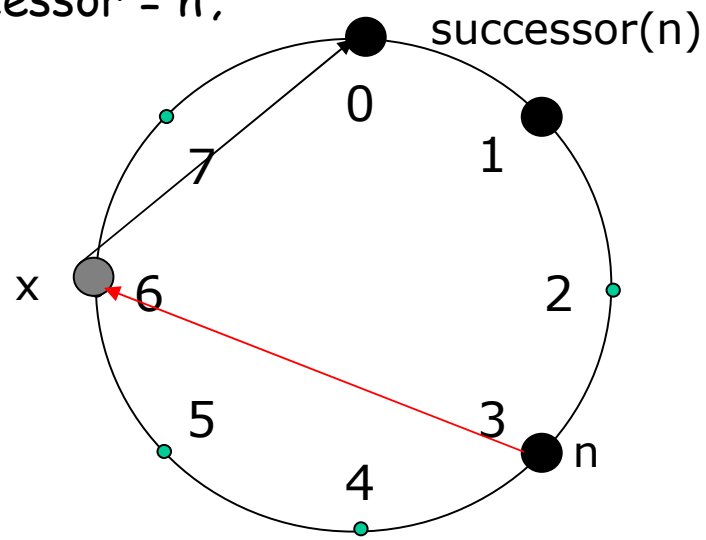
```
n.stabilize ( )
        x = successor.predecessor;
        if (x∈(n,successor))
                successor = x;
        successor.notify (n)
n.notify (n')
        if (predecessor è nil o n' ∈(predecessor, n))
                predecessor = n';
```



x è il nuovo nodo che si è inserito

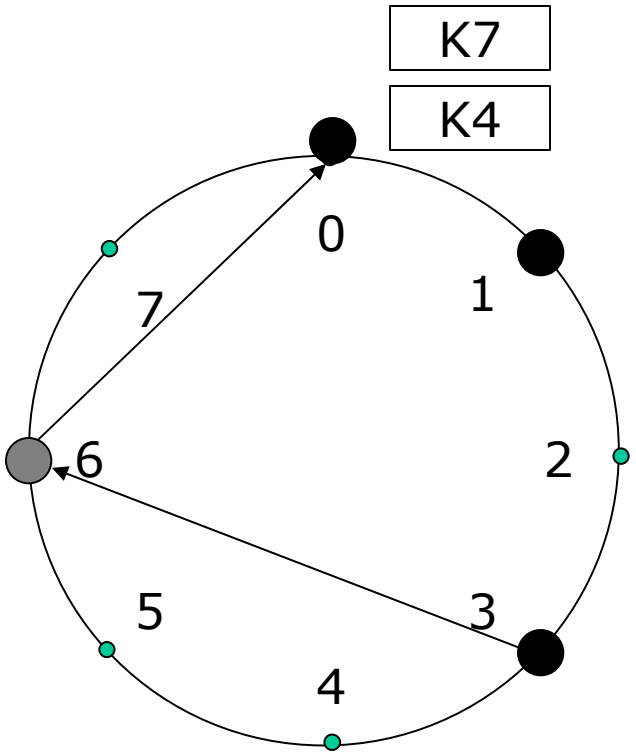
CHORD: INSERIMENTO DI NODI

```
n.stabilize ( )
        x = successor.predecessor;
        if (x∈(n,successor))
                successor = x;
        successor.notify (n)
n.notify (n')
        if (predecessor è nil o n' ∈(predecessor, n))
                predecessor = n';
```



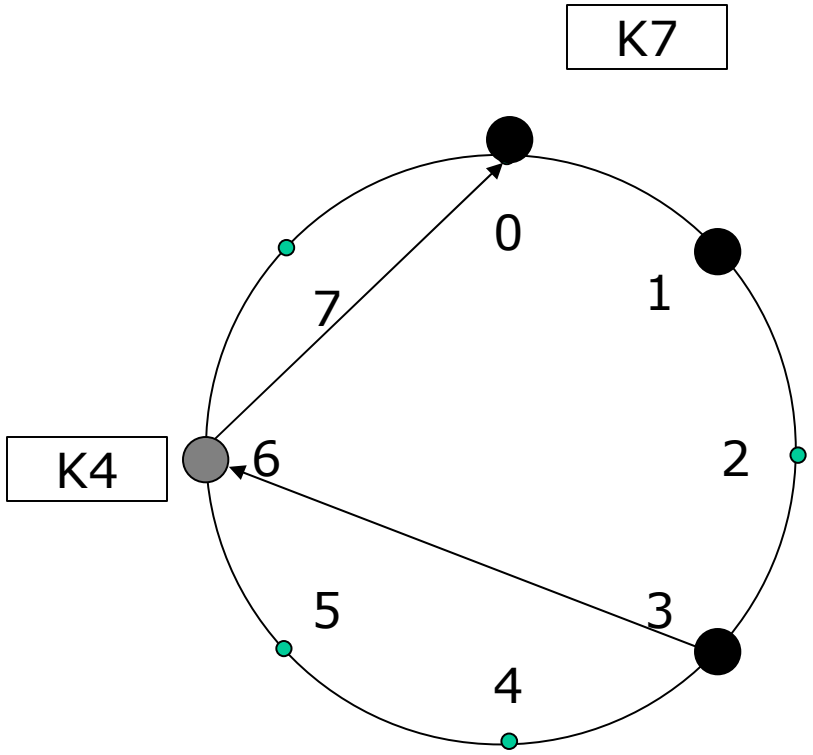
CHORD: INSERIMENTO DI NODI

• Per rispettare la strategia di allocazione chiavi-nodi,il nodo 6 deve copiare tutte le chiavi minori o uguali a 6 dal nodo 0. Questo deve essere effettuato dalla applicazione



CHORD: INSERIMENTO DI NODI

- Per rispettare la strategia di allocazione chiavi-nodi,il nodo 6 deve copiare tutte le chiavi minori o uguali a 6 dal nodo 0. Questo deve essere effettuato dalla applicazione



CHORD: INSERIMENTO DI NODI

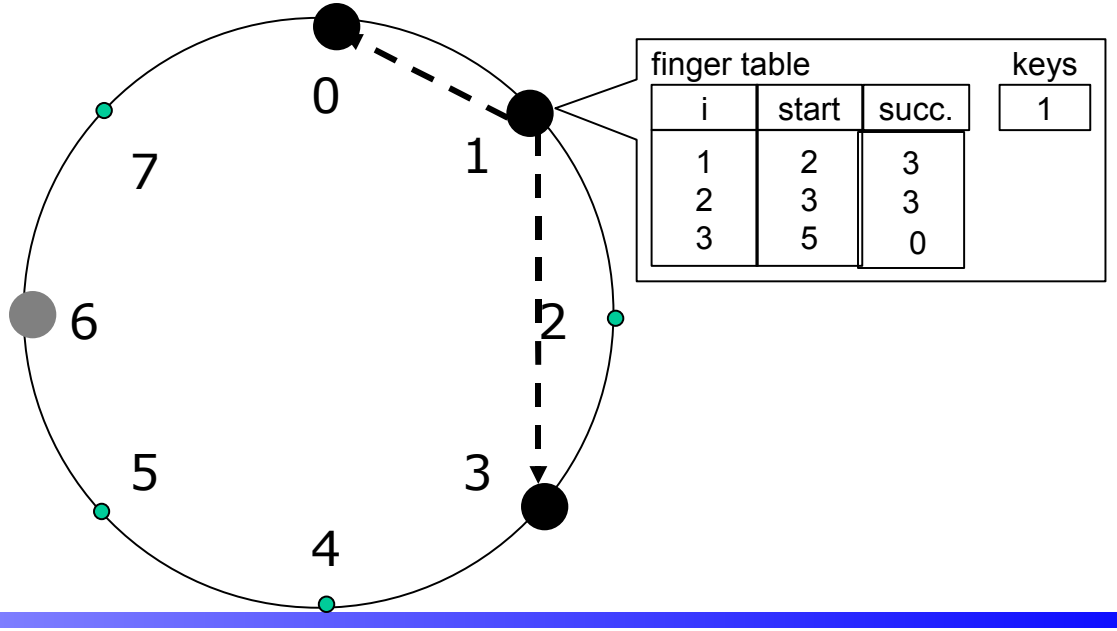
Quando tutti i puntatori ai nodi successivi sono aggiornati,

- la find_successor( ) la può essere utilizzata per aggiornare le finger_tables
- viene invocata periodicamente su un finger scelto a caso

n.fix_fingers ( )

next = scegliere una entrata nella finger table in modo casuale

finger[next] = find_successor(n+2 $^{next-1}$);



CHORD: INSERIMENTO DI NODI

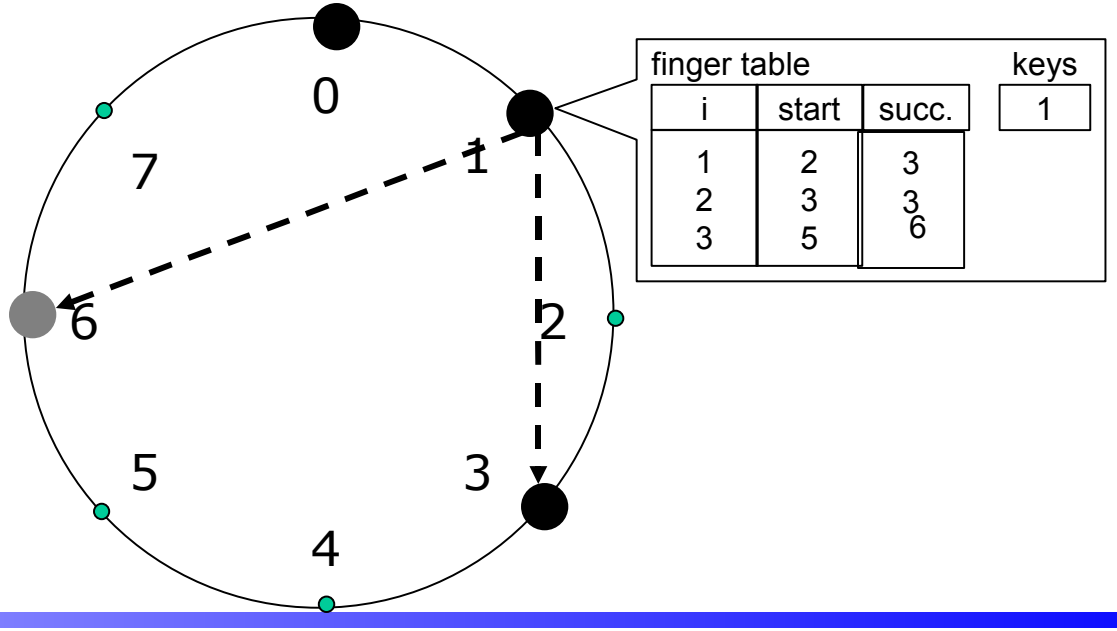
Quando tutti i puntatori ai nodi successivi sono aggiornati,

- la find_successor( ) la può essere utilizzata per aggiornare le finger_tables
- viene invocata periodicamente su un finger scelto a caso

n.fix_fingers ( )

next = scegliere una entrata nella finger table in modo casuale

finger[next] = find_successor(n+2 $^{next-1}$);



CHORD: INSERIMENTO DI NODI

- Lazy Join:
  - Quando un nodo si inserisce sull'anello può inizializzare solo il suo successore
  - Periodicamente rinfresca il contenuto della propria finger table (fix.finger)
- La correttezza della ricerca dipende dalla corretta impostazione del nodo successore di ogni nodo appartenente all'anello Chord
- La migrazione delle risorse al nuovo nodo non è gestita da Chord, deve essere eseguita dalla applicazione
- Tutti i link di un nodo sono correttamente aggiornati solo quando
  - è stata eseguita la stabilize( ) (che aggiorna il successore di un nodo considerando eventuali nodi inseriti)
  - È stata eseguita la fixfingers per tutti i finger della finger table sia del nodo che ha effettuat la join() sia degli altri nodi

CHORD: INSERIMENTO DI NODI

Esito di una ricerca eseguita prima che il sistema sia ritornato in uno stato stabile, dopo un inserimento:

- non tutti i puntatori ai nodi successivi sono corretti oppure le chiavi non sono state completamente trasferite. La ricerca può fallire ed occorre ritentare in seguito

- Ogni nodo ha aggirnato il suo successore reale sull'anello e le chiavi sono state correttamente trasferite tra nodi, le fingers table possono contenere informazione non completamente aggiornata. La ricerca ha esito positivo , ma può essere rallentata

- Tutte le finger tables sono in uno stato "raginevolmente aggiornato", allora il routing viene garantito in O(log N) passi

CHORD: SCELTE DI PROGETTO

Approccio a livelli

- Chord e' responsabile del routing
- La gestione dei dati è demandata alle applicazioni
  - persistenza
  - consistenza
  - fairness

Approccio soft

- i nodi cancellano le coppie (key, value) dopo che è trascoso un intervallo di tempo (periodo di refresh) dall'ultimo inserimento
- le applicazioni effettuano il refresh periodico delle coppie (key, value)
- in questo modo si attribuiscono le informazioni ai nuovi nodi arrivati
- se un nodo fallisce occorre aspettare il periodo di refresh per avere le informazioni nuovamente disponibili

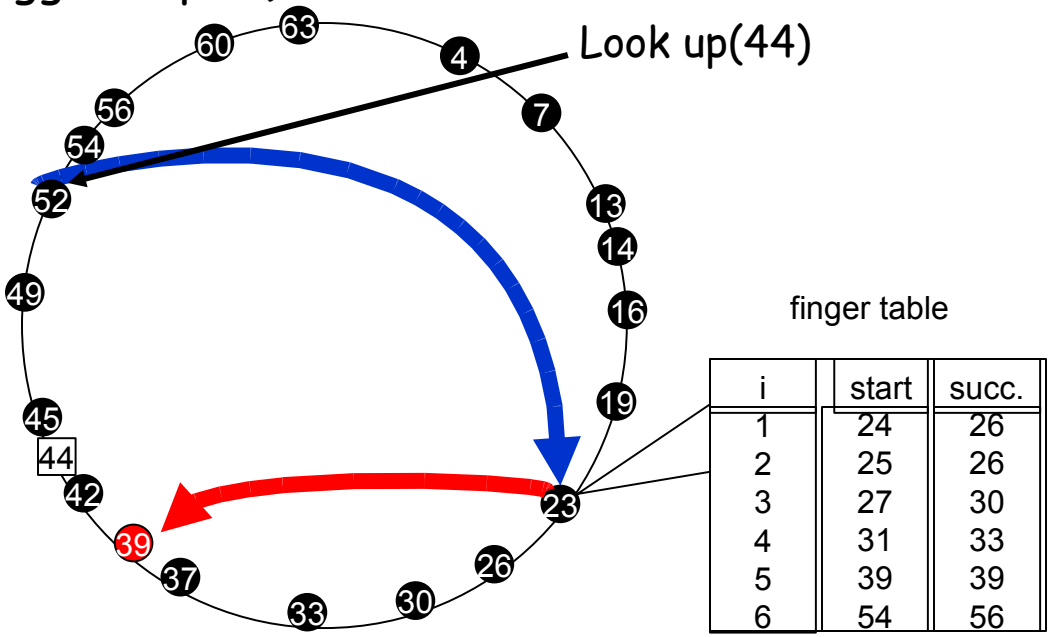
INSERIMENTO/CADUTA DINAMICA DI NODI

- Tutte le operazioni relative all'inserimento/caduta di un nodo sono corrette se avvengono quando l'anello si trova in uno stato stabile
- L'anello deve avere il tempo di stabilizzarsi tra due operazioni successive
- In pratica l'anello potrebbe non essersi stabilizzato prima di un nuovo inserimento/cancellazione
- Risultato generale: se i protocolli di stabilizzazione dell'anello vengono eseguiti con una frequenza opportuna, dipendente dalla frequenza degli aggiornamenti (inserimenti/fallimenti) allora l'anello rimane costantemente in uno stato stabile, in cui il routing rimane corretto e veloce (si mantiene il limite di complessità log(N))

CHORD: SITUAZIONE CONSITENTE

durante il routing, ogni comunicazione con i fingers è controllata mediante l'attivazione di time outs. Se il time-out scade :

- la query viene inviata al finger precedente, per evitare di oltrepassare il nodo destinazione.
- il finger caduto viene rimpiazzato con il suo successore nella finger table (trigger repair)



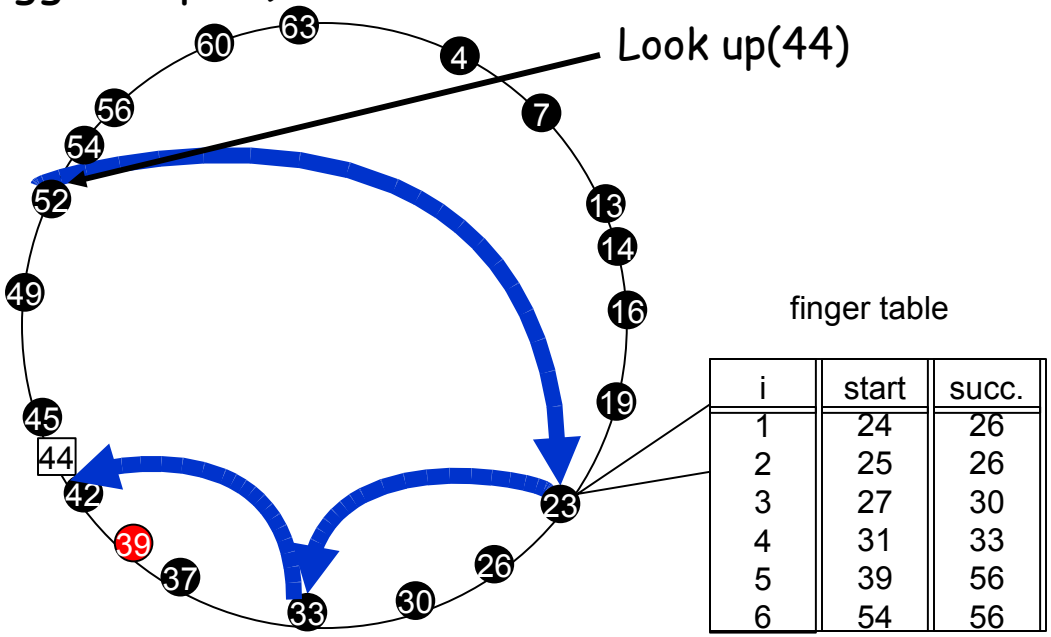
finger table

i	start	succ.
1	24	26
2	25	26
3	27	30
4	31	33
5	39	39
6	54	56

CHORD: SITUAZIONE CONSISTENTE

durante il routing, ogni comunicazione con i fingers è controllata mediante l'attivazione di time outs. Se il time-out scade :

- la query viene inviata al finger precedente, per evitare di oltrepassare il nodo destinazione.
- il finger caduto viene rimpiazzato con il suo successore nella finger table (trigger repair)



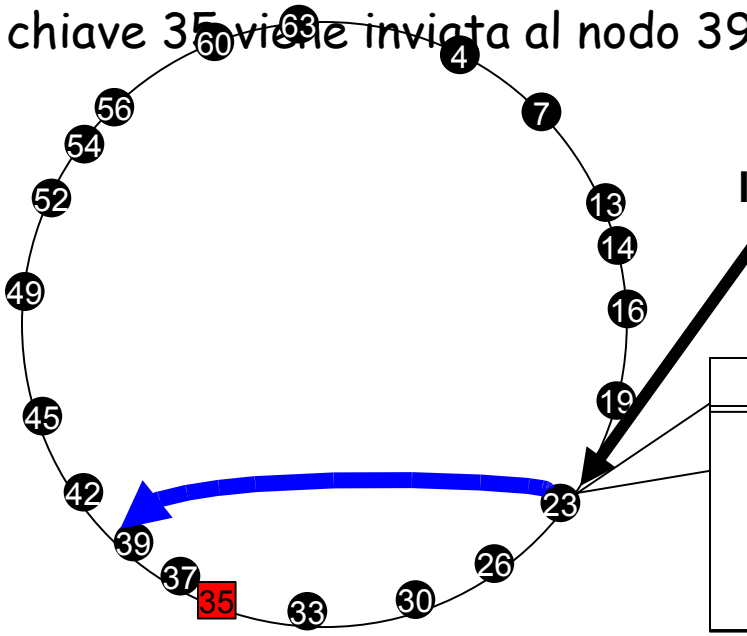
finger table

i	start	succ.
1	24	26
2	25	26
3	27	30
4	31	33
5	39	56
6	54	56

CHORD: INCOSISTENZA

Situazione inconsistente: un nodo perde il riferimento al suo vero successore sull'anello

- i primi tre successori del nodo 23 (26,30,33) falliscono.
- il successore sull'anello del nodo 23 diventa il nodo 37
- poichè il nodo 23 non possiede un riferimento al nodo 37 nella finger table, 23 considera 39 come suo nuovo successore.
- La chiave 35 viene inviata al nodo 39, mentre essa è gestita dal nodo 37.



lookup (35)

finger table

i	start	succ.
1	24	26
2	25	26
3	27	30
4	31	33
5	39	39
6	54	56

finger table

i	start	succ.
1	24	39
2	25	39
3	27	39
4	31	39
5	39	39
6	54	56

CHORD: CADUTA DI NODI

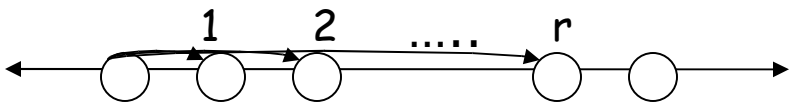
- la correttezza dell'algoritmo di routing è garantita se ogni nodo mantiene aggiornato il riferimento al nodo successivo anche in caso di fallimenti multipli

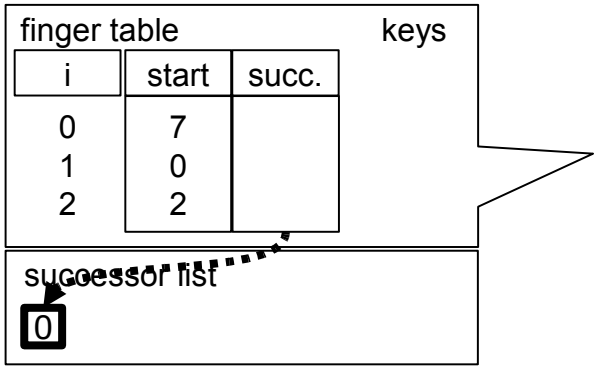
- ma… in caso di fallimenti simultanei questo invariante non può essere garantito

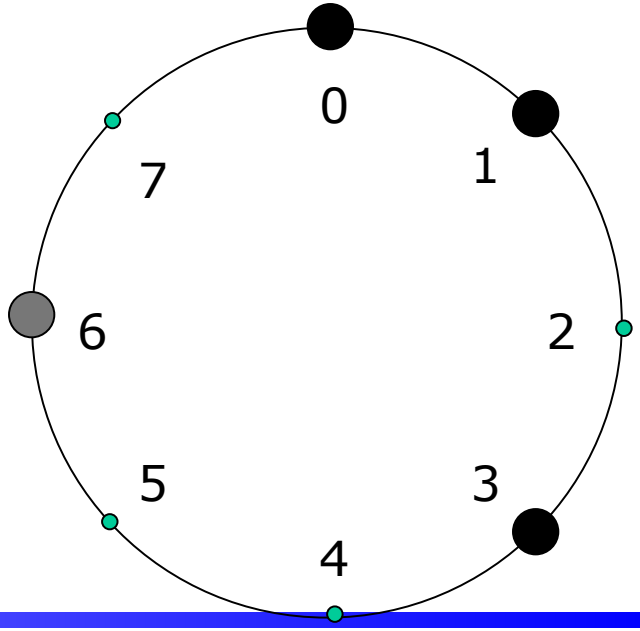
- Per migliorare la robustezza del sistema
  - ogni nodo n mantiene una lista dei suoi r immediati successori sull'anello (r=log N)
  - se n rileva la caduta del suo successore, il successore viene sostituito con il secondo elemento della lista e così via
  - La lista viene mantenuta consistente mediante la procedura di stabilizzazione
  - valori crescenti di r rendono il sistema maggiormente robusto

CHORD: LISTA DEI SUCCESSORI

- lista dei successori di n = include i primi r successori di n sull'anello, in senso orario
- il nodo n richiede la lista dei successori al suo immediato successore s, rimuove l'ultimo elemento ed inserisce s in testa
- Se il successore cade, viene sostituito con l'elemento successivo della lista
- Look up: ricerca nella finger table + lista successori

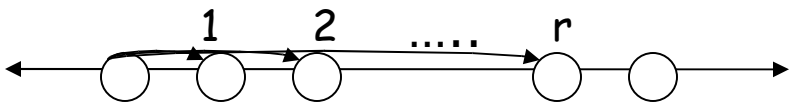


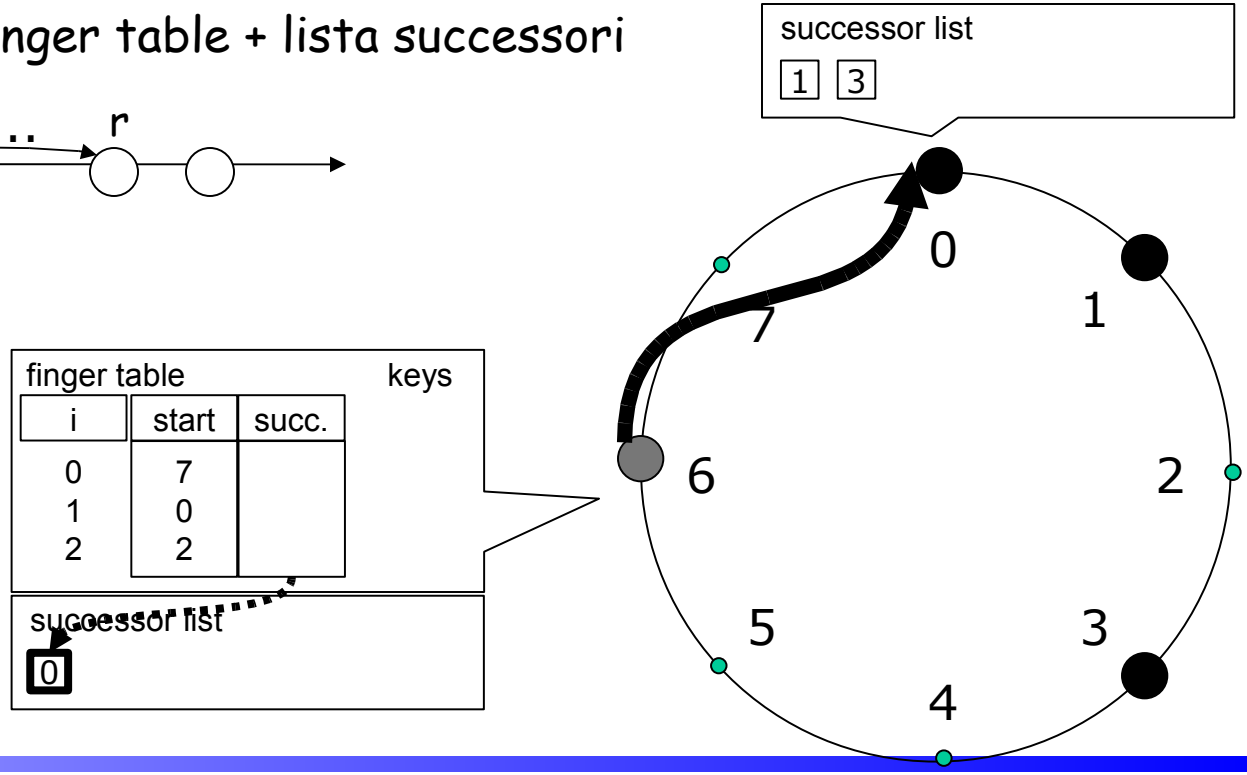




CHORD: LISTA DEI SUCCESSORI

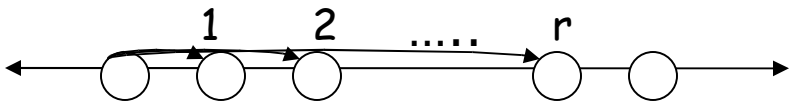
- lista dei successori di n = include i primi r successori di n sull'anello, in senso orario
- il nodo n richiede la lista dei successori al suo immediato successore s, rimuove l'ultimo elemento ed inserisce s in testa
- Se il successore cade, viene sostituito con l'elemento successivo della lista
- Look up: ricerca nella finger table + lista successori

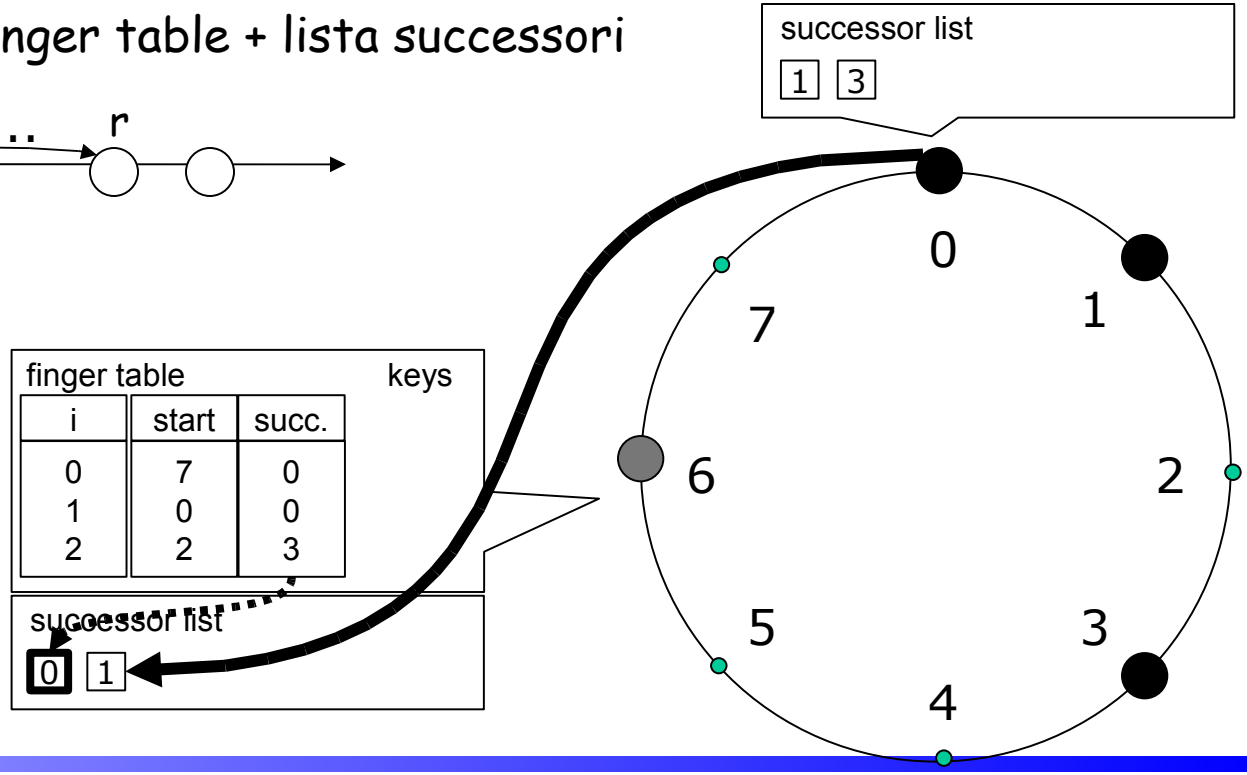




CHORD:LISTA DEI SUCCESSORI

- lista dei successori di n = include i primi r successori di n sull'anello, in senso orario
- il nodo n richiede la lista dei successori al suo immediato successore s, rimuove l'ultimo elemento ed inserisce s in testa
- Se il successore cade, viene sostituito con l'elemento successivo della lista
- Look up: ricerca nella finger table + lista successori





CHORD: CADUTA DEI NODI

- La correttezza della finger table viene verificata periodicamente
  - controllo periodico della caduta dei fingers
  - sostituzione con nodi attivi
  - trade-off: traffico aggiuntivo per la gestione dei fallimenti vs. correttezza e velocità nel reperimento delle informazioni

CHORD: REPLICAZIONE DEI DATI

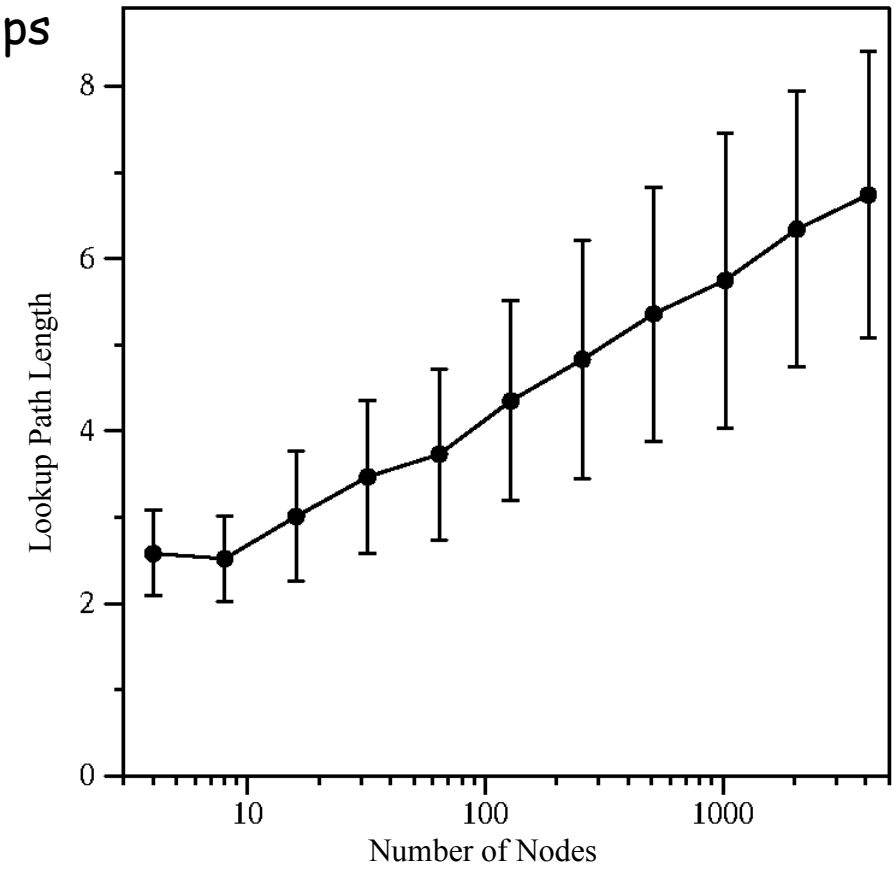
- Chord non garantisce l'affidabilità dei dati in presenza di fallimenti
- Ma….Chord fornisce dei meccanismi per garantirne l'affidabilità
- Ogni applicazione che utilizza il livello Chord, può utilizzare la lista dei successori di un nodo per garantire che un dato venga replicato negli r successori di un nodo
- Nel momento che un nodo fallisce, il sistema può utilizzare le repliche per individuare i dati in modo corretto

CHORD: PARTENZA VOLONTARIA DEI NODI

- Partenza volontaria di nodi
  - shutdown volontario di un nodo vs. caduta improvvisa del nodo
- Per semplicità: può essere trattata come un fault
- Ottimizzazioni: il nodo n che intende abbandonare l'anello
  - notifica la sua intenzione al successore, al predecessore ed ai fingers.
    - il predecessore può rimuovere n dalla sua lista di successori
    - il predecessore può aggiungere alla sua lista di successori il primo nodo della lista di successori di n
  - Trasferisce le chiavi di cui è responsabile al suo successore

CHORD: SIMULAZIONE

- Rete = 2^k nodi, 100 x 2^k chiavi, k variabile tra 3 e 14.
- Per ogni valore di k, si considera un insieme casuale di chiavi e si effettua un esperimento separato
- Per ogni chive si valuta il numero di hops per la ricerca
- Lookup Path Length ~ 1/2 $\log_2(N)$
- Conferma dei risultati teorici
- Il grafico mostra il 1 il 99 percentile e la media

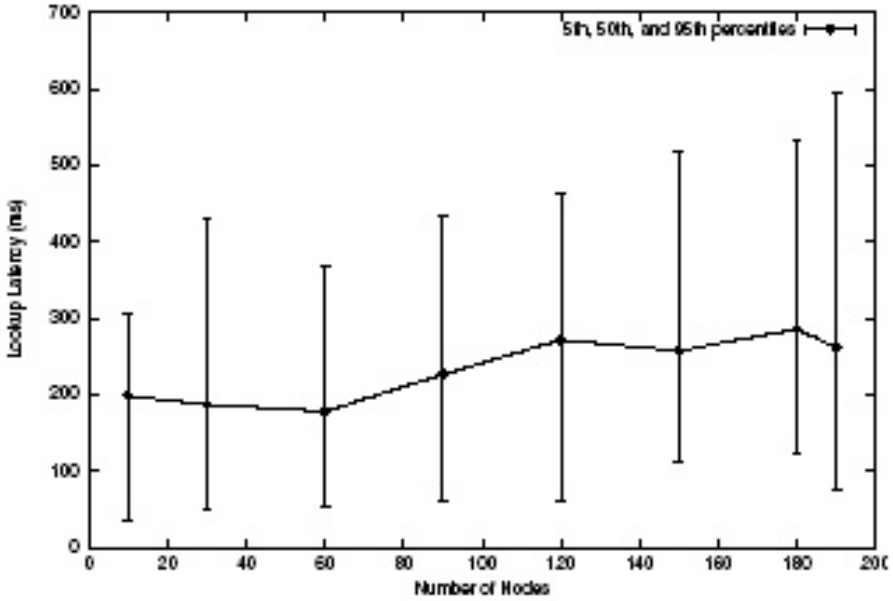


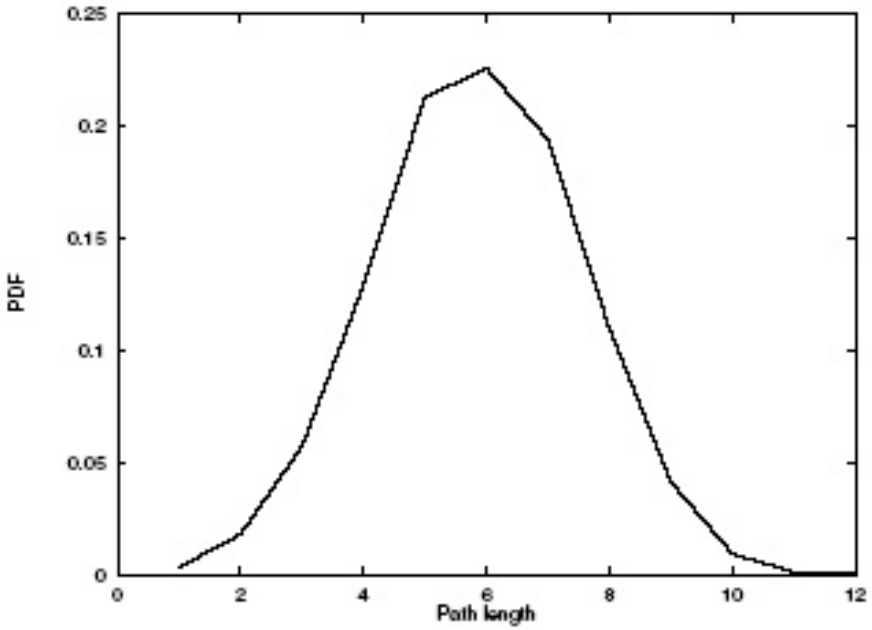
CHORD: IL PROTOTIPO

- Sviluppo di un prototipo Chord sviluppato su nodi Internet
- Nodi Chord dislocati in 10 siti (situati in diversi stati USA)
- Studio al variare del numero di nodi: per ogni numero di nodi sono effettuate 16 queries per chiavi scelte in modo casuale
- Latenza media varia da 180 ms. a 300 ms, dipende dal numero di nodi

CHORD: PERFORMANCE

Modesto impatto del numero di nodi sulla latenza





Lunhezza dei cammini:
PDF (Probability Density Function)
per una rete di 2^{12} nodi

CHORD: CONCLUSIONI

- Complessità
  - Messaggi di lookup: $O(\log N)$ hops
  - Memoria per node: $O(\log N)$
  - Messaggi per auto-organizzazione (join/leave/fail): $O(\log^2 N)$
- Vantaggi
  - Modelli teorici e prove di complessità
  - Semplice e flessibile
- Svantaggi
  - Manca una nozione di prossimità fisica
  - Casi reali: possibili situazioni limite
- Ottimizzazioni proposte
  - e.g. prossimità, links bi-direzionali, load balancing, etc.