

Chord:

A scalable peer-to-peer lookup protocol for internet applications.

Stoica – Morris – Liben-Nowell – Karger –
Kaashoek – Dabek - Balakrishnan

Presentazione a cura di
Angelo Spognardi
Dipartimento di Informatica
Università “La Sapienza” - Roma
aspora@libero.it



Il peer-to-peer

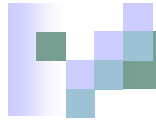
- n La comunicazione fra diversi host può avvenire secondo diversi modelli:
 - Client – Server
 - Gerarchico
 - Peer – to – Peer (Gnutella, FreeNet...)

Nel peer-to-peer ogni nodo ha le stesse funzioni e capacità degli altri.



Esempi di applicazioni

- n Assicurare ridondanza delle informazioni
- n Garantire disponibilità delle informazioni
- n Fornire possibilità di scelta dell'origine delle informazioni
- n Consentire l'anonimità degli host

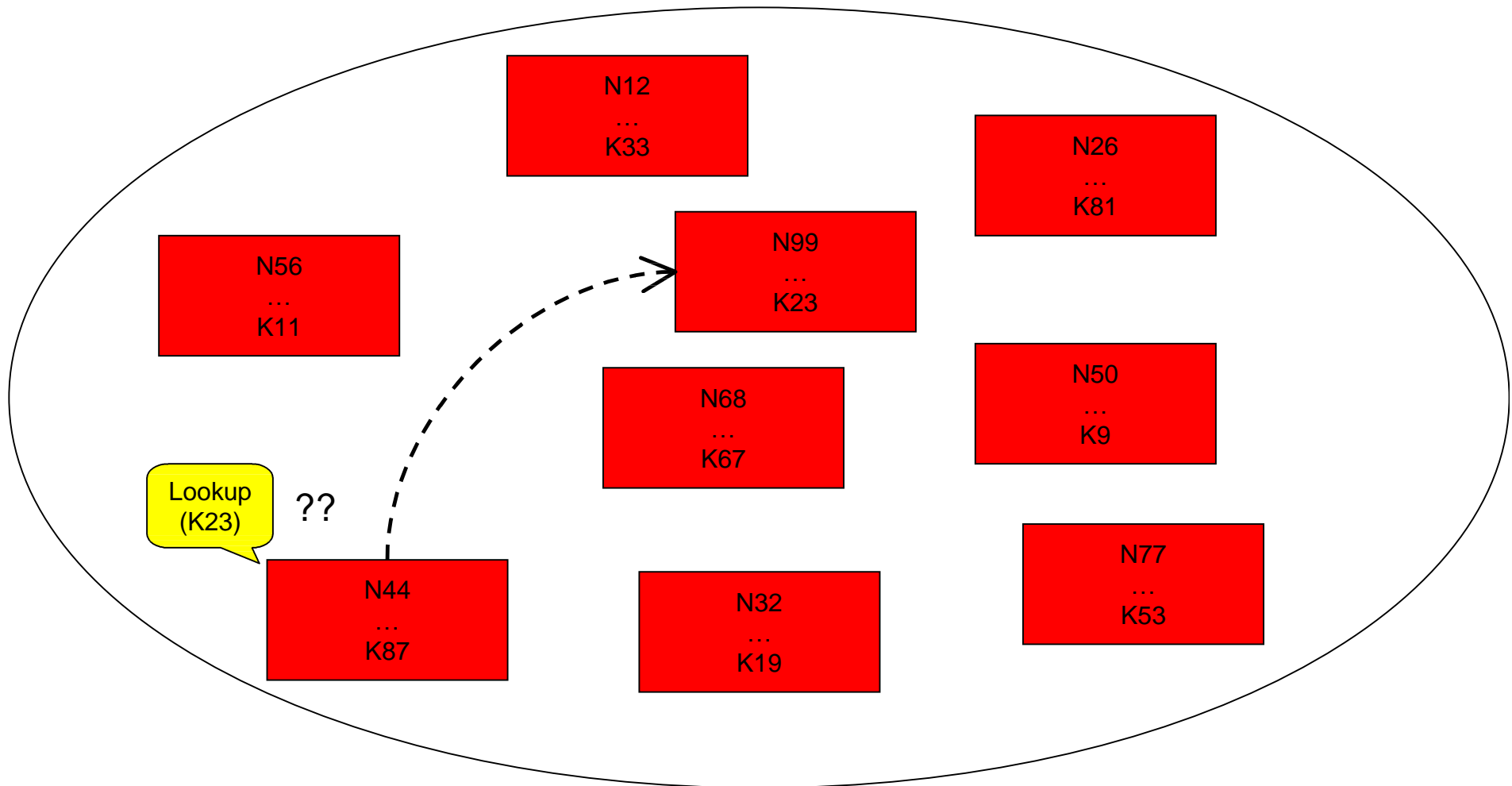


Problematiche

- n Rischio di sbilanciamento della distribuzione delle informazioni
- n Incompletezza delle ricerche
- n Inefficienza delle ricerche
- n Garantire scalabilità

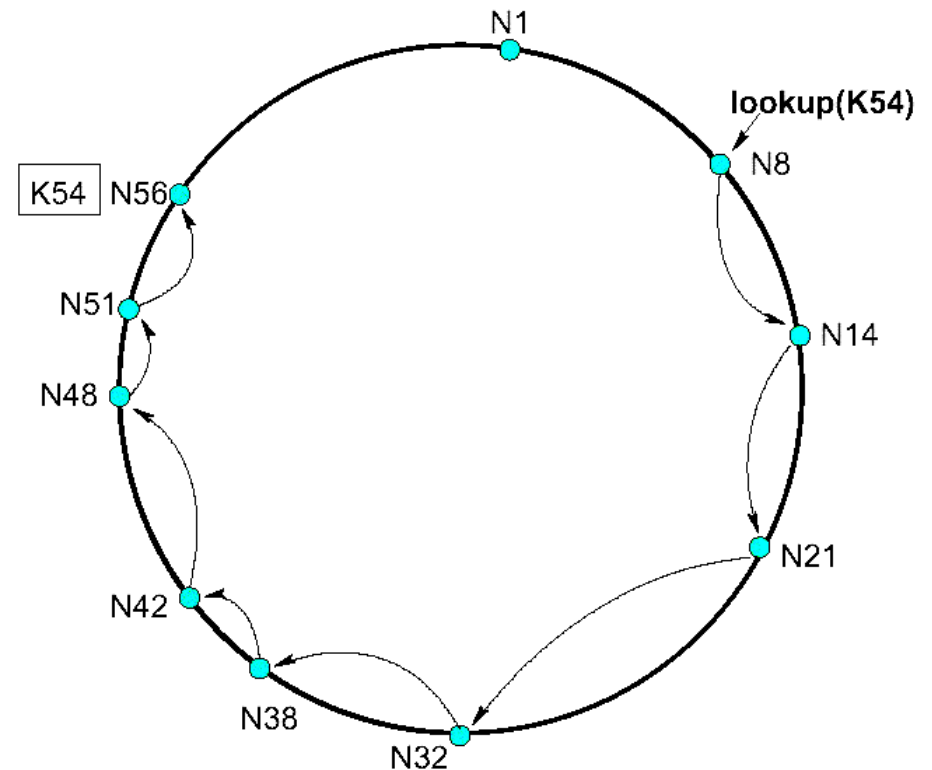
Esempio di lookup

N_i: id nodo
K_j: id chiave



Lookup inefficiente

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





Lookup Inefficiente

- n Dimensione delle tabelle di routing costante (1: un solo successore)
- n Numero medio di messaggi lineare rispetto al numero di nodi



Protocollo CHORD

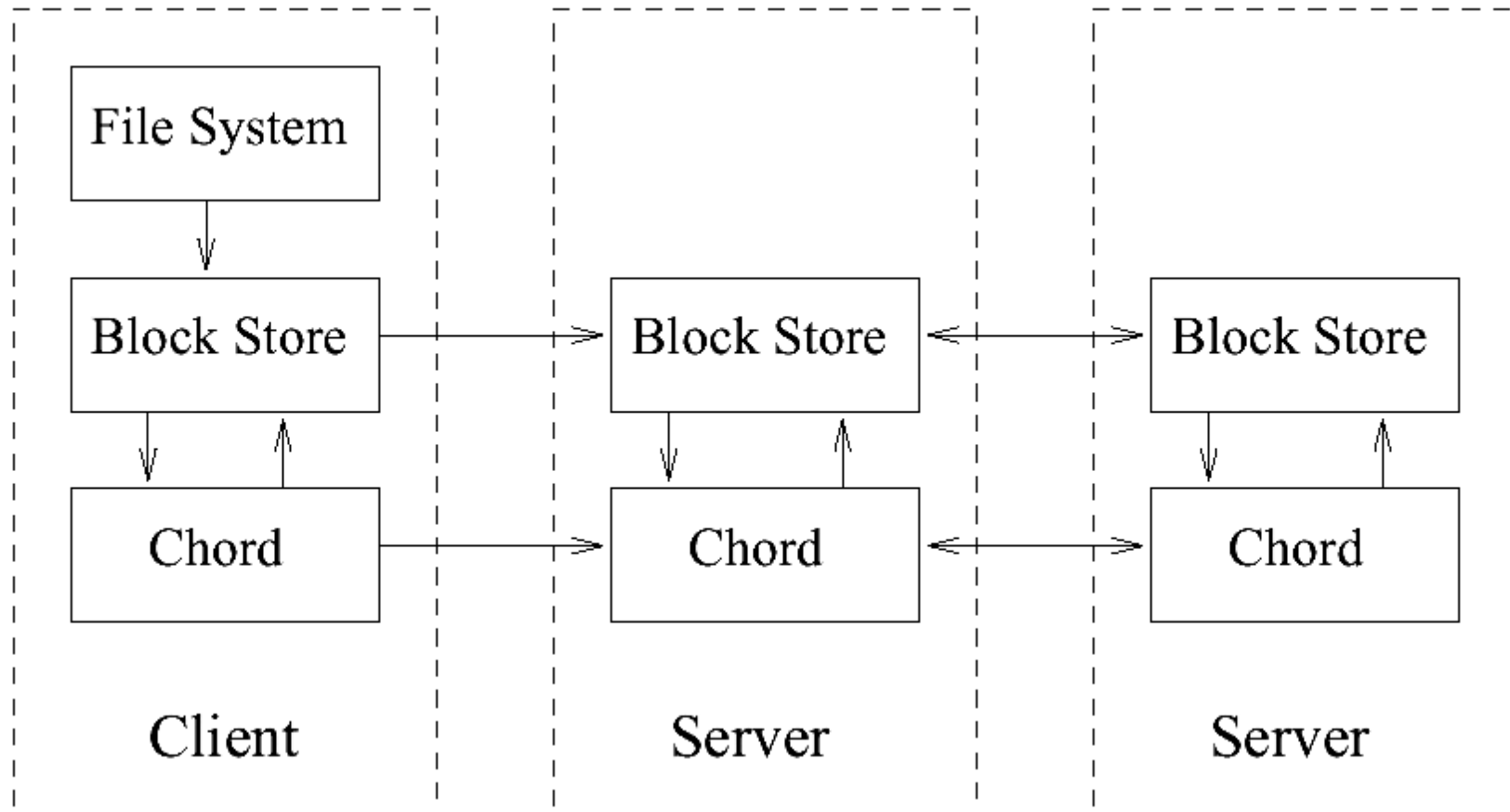
n Specifica:

- come trovare le locazioni delle chiavi di ricerca,
- in che modo i nodi devono unirsi al sistema e
- come comportarsi in caso di fallimento

n Caratteristiche del protocollo:

- Ricerche in $O(\log N)$
- Tavole di routing nell'ordine $\log N$
- Bilancia il carico dei dati
- È totalmente distribuito
- È scalabile
- Si adatta ai cambiamenti della rete
- Non pone vincoli sulla struttura delle chiavi di ricerca

Uso di Chord





Uso di Chord cont.

- n Il File System è l'applicazione di livello superiore che permette alle applicazioni di fruire del sistema
- n Il Block Store realizza le funzioni di gestione dei dati locali
- n Il livello Chord realizza la ricerca nella rete, lavorando sulle tabelle di routing e interagendo con gli altri peer



Prerequisito

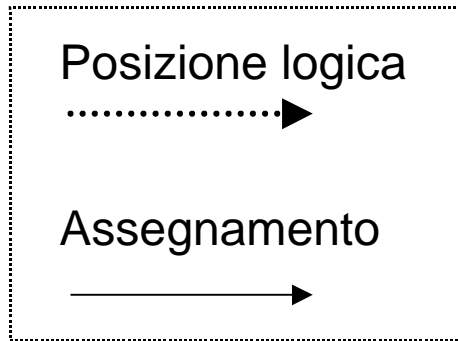
- n Chord assegna ad ogni nodo (n) e chiave di item (k) un identificatore di m -bit impiegando una funzione hash “coerente”
- n Gli identificatori (*keys*) sono ordinati in un anello di identificatori modulo 2^m (*Chord ring*)



La regola di Chord

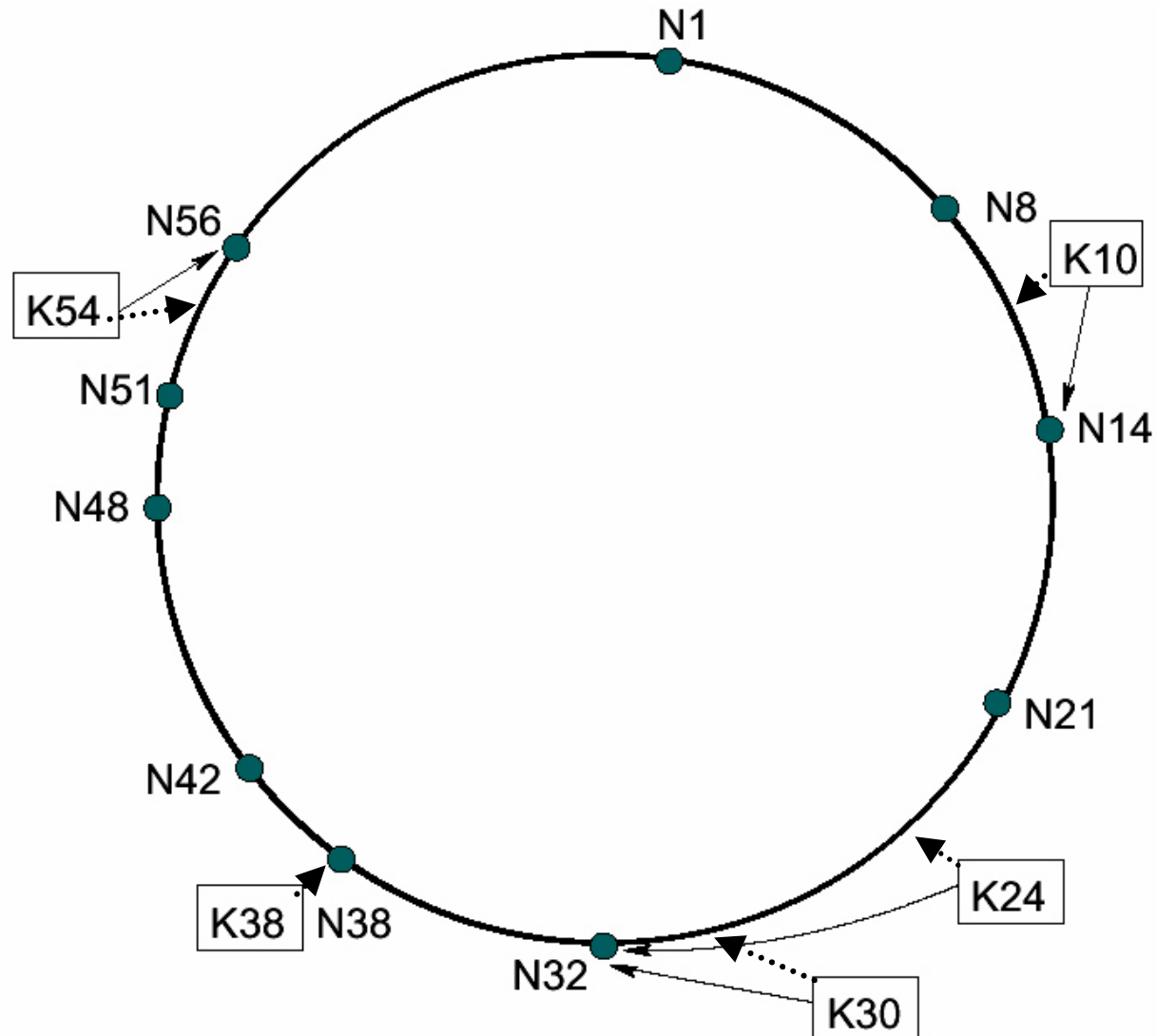
- n Una chiave k è assegnata al primo nodo n il cui identificatore è uguale o segue k nello spazio degli identificatori
- n Il nodo n è detto il nodo successore della chiave k , denotato con $successor(k)$
- n All'interno del *Chord ring*, $successor(k)$ è il primo nodo in senso orario, dopo k

Esempio di Chord Ring



K54: risultato dell'hashing di un item

N51: risultato dell'hashing dell'IP di un nodo



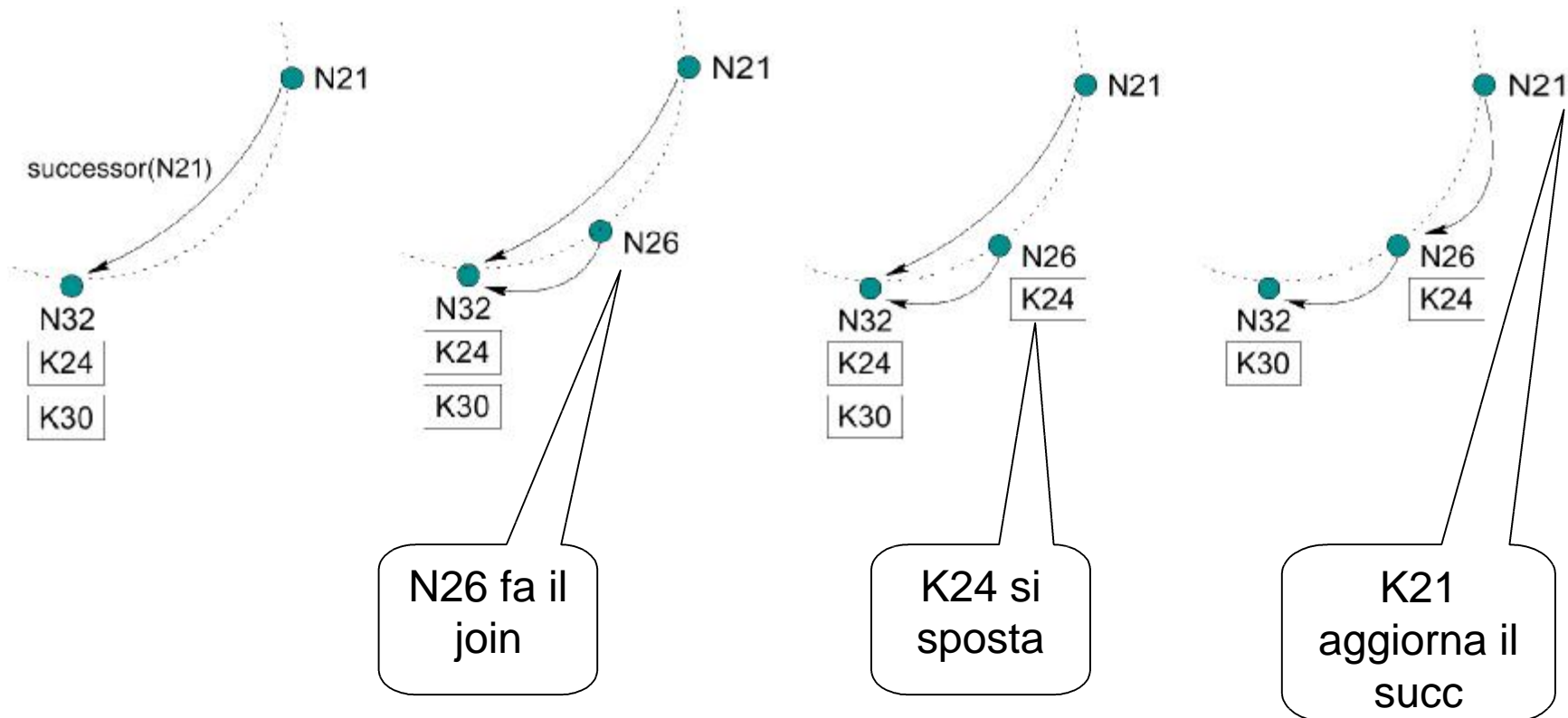


Join e leave

- n Grazie alla regola di Chord, i join e i leave dei nodi avvengono con spostamenti minimi delle chiavi fra i nodi
- n Si può dimostrare che con alta probabilità:
 - Ogni nodo è responsabile di al più $(1+\varepsilon)K/N$ chiavi
 - Quando un $n+1$ esimo nodo si unisce o abbandona il sistema, soltanto $O(K/N)$ chiavi cambiano di gestione

Join (e all'inverso, leave)

successor(N21)=N32

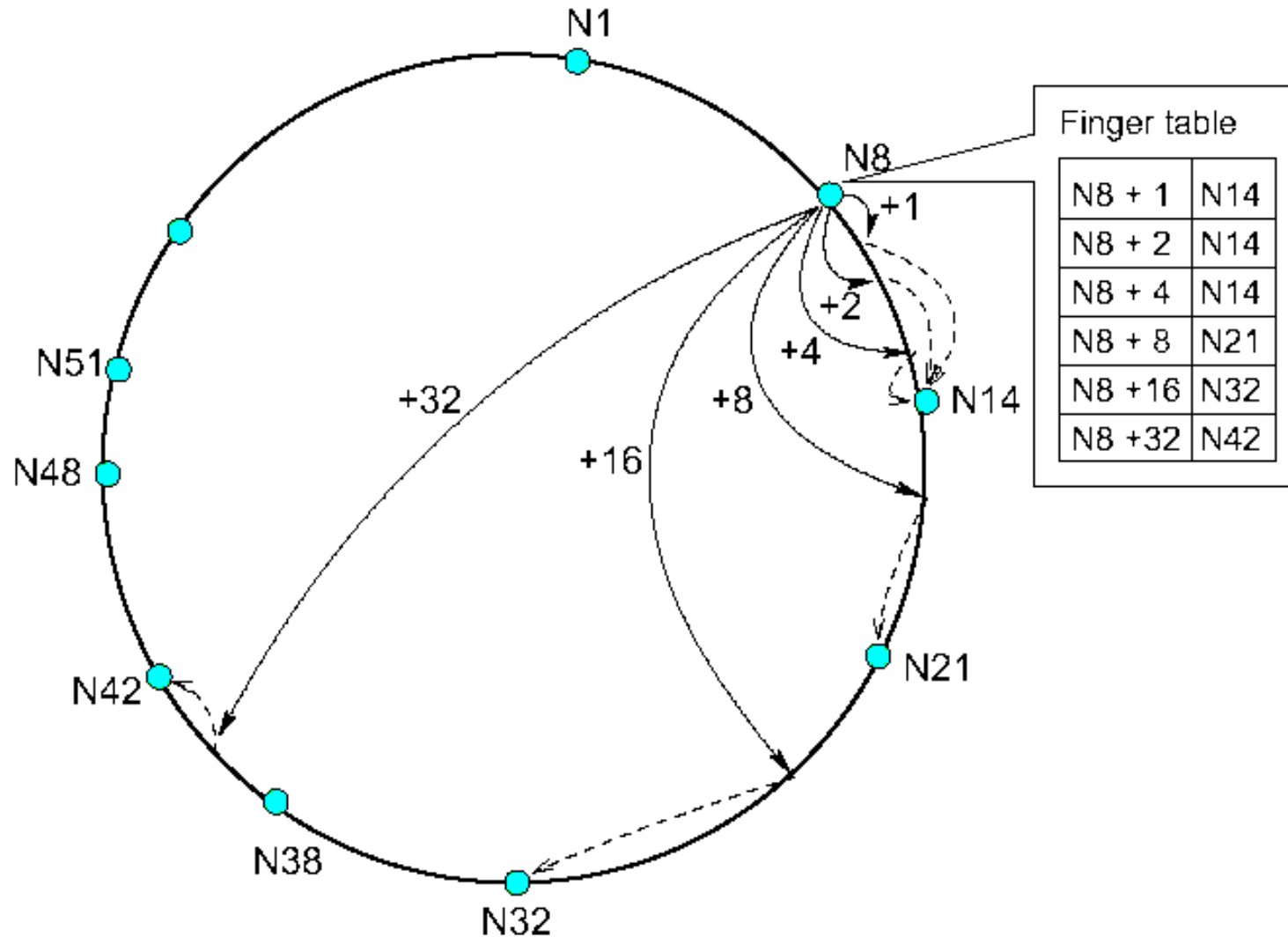




Regola dei *finger*

- n Ogni nodo mantiene una tabella di routing (*finger table*) di dimensione m :
- n L' i -esimo elemento di un nodo n è il primo nodo s che segue n di almeno 2^{i-1} posizioni nel *ring*, ovvero $s = \text{successor}(n + 2^{i-1})$, nell'aritmetica modulo 2^m
- n s è detto l' i -esimo *finger* ed è indicato con $n.\text{finger}[i]$
- n $n.\text{finger}[1]$ è l'immediato successore di n

La *finger table* di N8





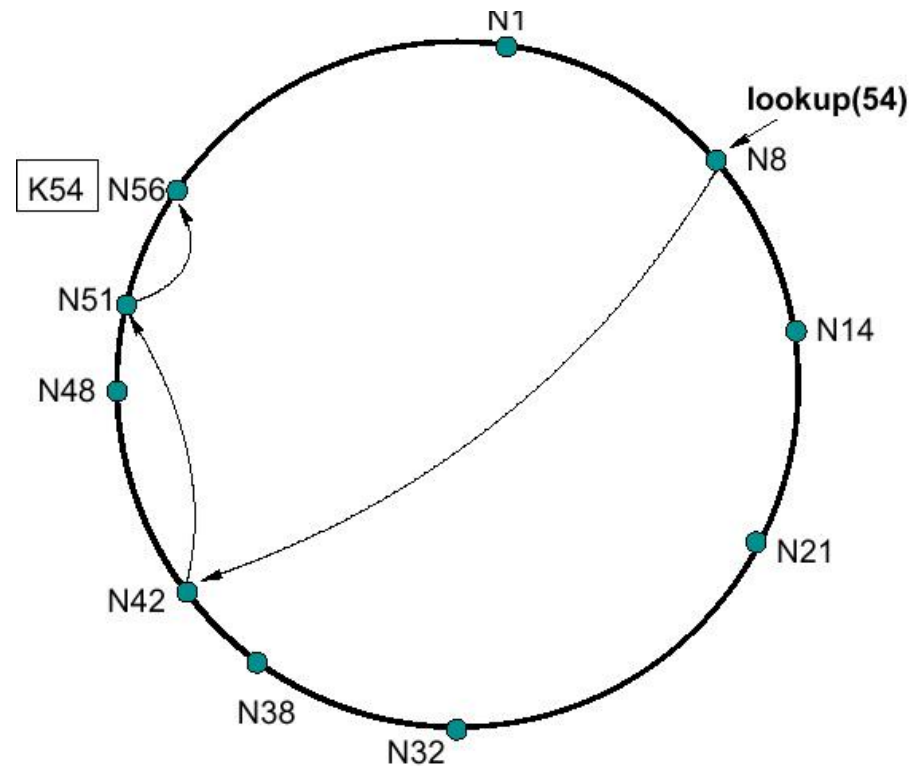
La finger table

- n Ha tre caratteristiche:
 - Contiene pochi valori ($\log n$) rispetto al numero totale di nodi
 - Fornisce una migliore conoscenza nelle immediate vicinanze di un nodo
 - In genere non contiene abbastanza informazioni per raggiungere una chiave direttamente

Lookup di Chord

```
// ask node n to find the successor of id
n.find_successor(id)
  if (id ∈ (n, successor])
    return successor;
  else
    n' = closest_preceding_node(id);
    return n'.find_successor(id);

// search the local table for the highest predecessor of id
n.closest_preceding_node(id)
  for i = m downto 1
    if (finger[i] ∈ (n, id))
      return finger[i];
  return n;
```





Prestazioni

- n Con alta probabilità il numero di messaggi è dell'ordine $O(\log N)$, dove N è il numero di nodi nel sistema
- n Il numero medio di messaggi è $\frac{1}{2} \log N$



Come mantenere informazioni coerenti

- n Per aggiornare le tabelle di routing, ogni nodo esegue periodicamente queste procedure:
 - *Stabilize*, per verificare il proprio immediato successore
 - *Fix_fingers*, per aggiornare le tabelle
 - *Check_predecessor*, per verificare il proprio antecedente nel *ring*



Stabilize

- n Chiamato periodicamente da un nodo n
- n Consente ad n di verificare se è ancora il predecessor del proprio successor p
- n Se non lo è, n aggiorna il proprio *successor* con il nodo appena conosciuto q e notifica a tale nodo la propria esistenza, eseguendo una procedura *notify*
- n Con *notify*, q ha a possibilità di aggiornare il proprio predecessor, settandolo a n



Fix_fingers

- n Chiamata periodicamente da un nodo n
- n Consente ad n di verificare le entry della tabella dei *fingers*
- n Consiste nel richiamare per ogni valore da 1 a m la procedura di ricerca sul valore $n+2^m$
- n Viene richiamata su un valore alla volta



Check_predecessor

- n* Chiamata periodicamente da un nodo *n*
- n* Consente ad *n* di resettare il proprio predecessor, nel caso che questo sia fallito
- n* In caso di fallimento, il predecessor viene settato quando *n* riceve una *notify* (durante una procedura *stabilize*)



Costi delle procedure

- n Stabilize: **costante** {uno o due messaggi}
- n Fix_finger: **$O(\log n)$** {una ricerca}
- n Check_predecessor: **costante** {un messaggio}
- n Fix_finger viene richiamata periodicamente su un *finger* per volta (non tutti di seguito)

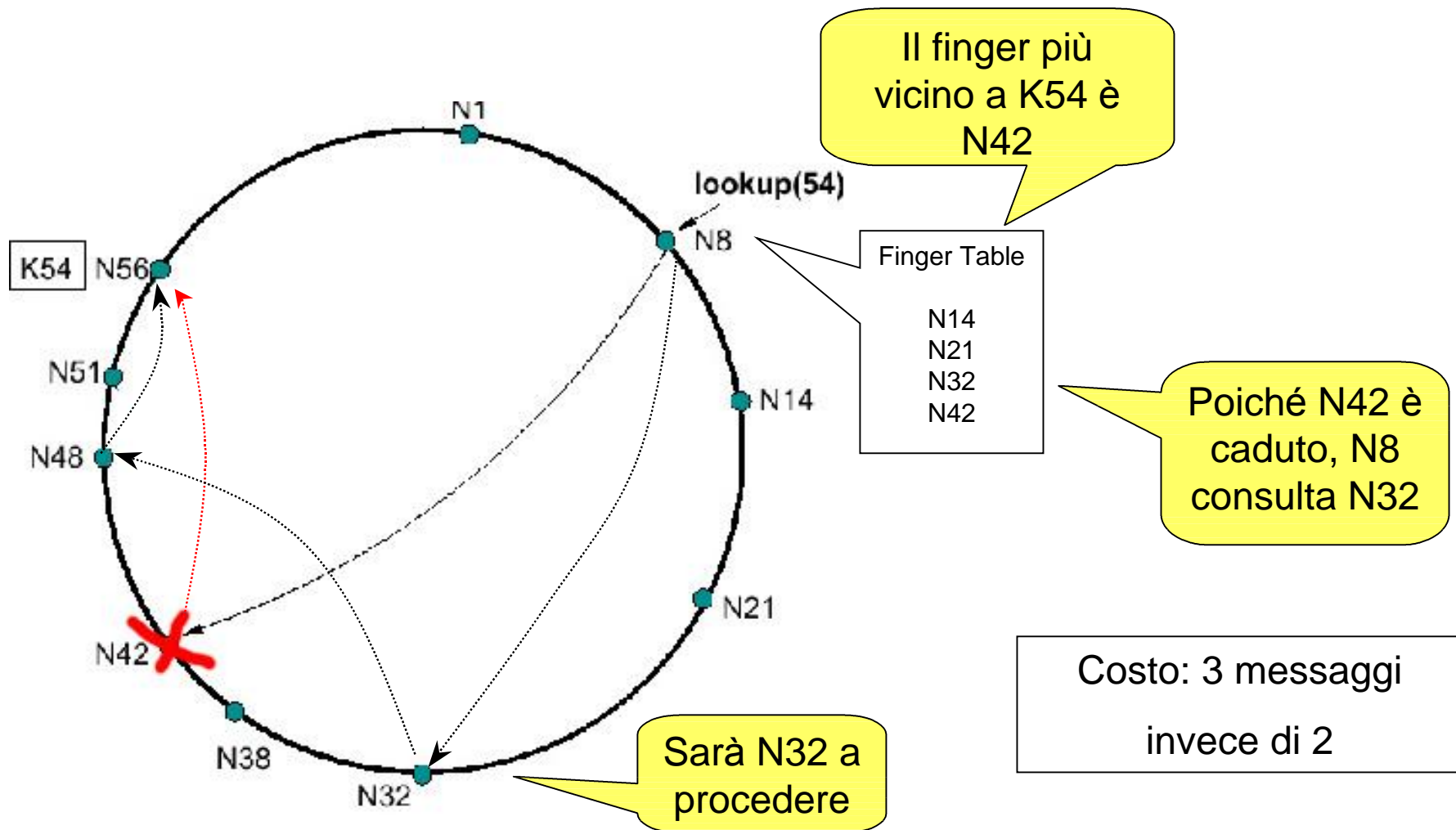


Lookup durante failure

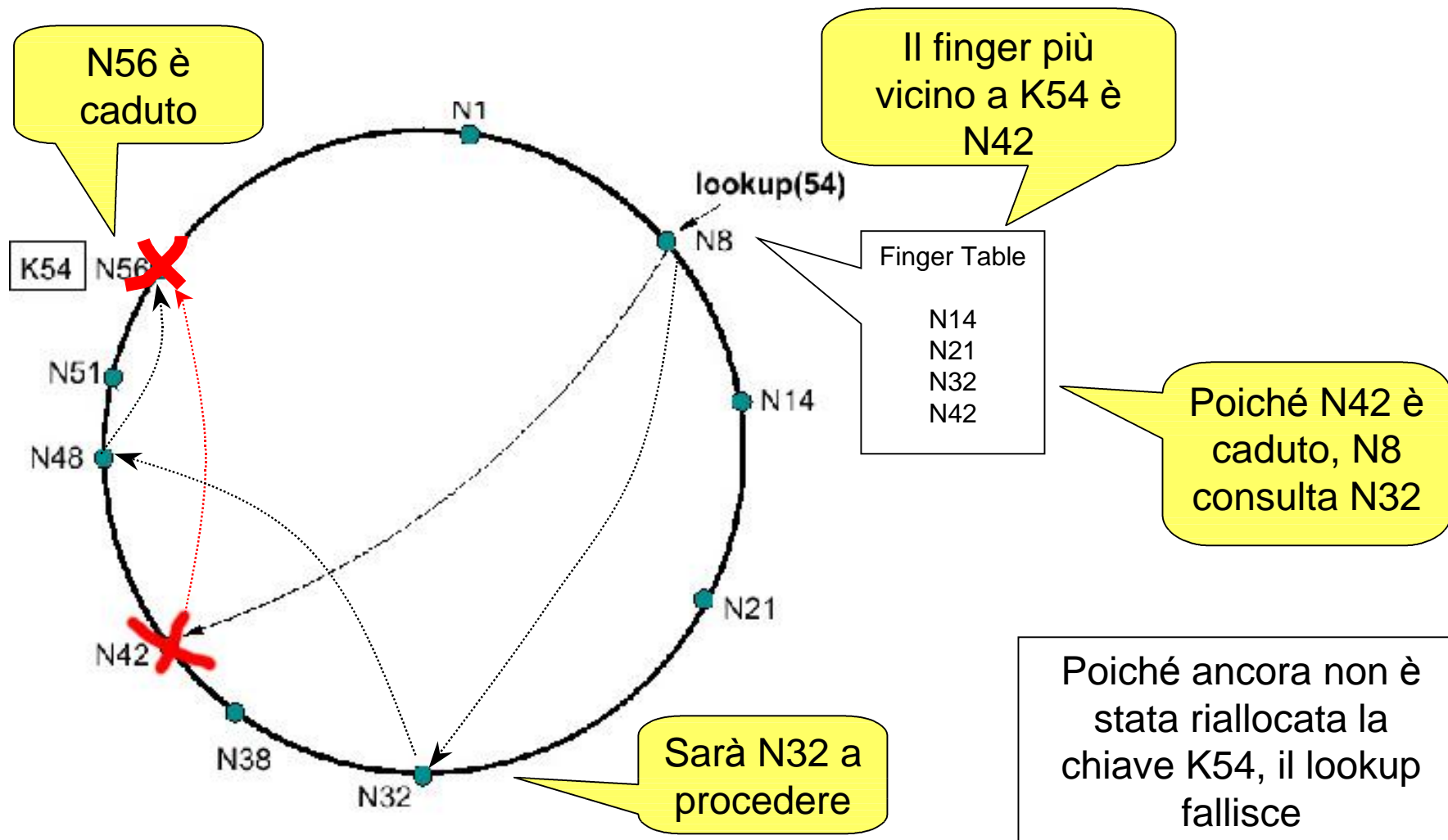
n Tre casi possibili

- Nessun nodo fallito interviene nel lookup
- Nodi falliti lungo il percorso del lookup, ma catena di successori valida: esito corretto ma più lento
- Nodi falliti sulla destinazione del lookup e informazioni non ancora recuperate: fallimento della ricerca. A livello application si deve rieseguire il lookup, dopo una pausa

Failure su qualche nodo



Fallimento del lookup





Aumentare la robustezza

- n Ogni nodo mantiene una lista di r successori, piuttosto che uno solo
- n Se $r = W(\log n)$ e la probabilità che un nodo fallisca è $\frac{1}{2}$, con alta probabilità l'esito di $lookup(k)$ è il nodo ancora attivo più vicino a k
- n Grazie alla randomicità dell'hashing, con alta probabilità gli attacchi localizzati si riflettono su diverse zone del ring



Conclusioni

- n Chord permette ricerche logaritmiche rispetto alla dimensione della rete
- n Si adatta velocemente alla dinamica dei join e dei leave
- n Robusto rispetto ai fallimenti dei nodi



Ring stabile

- n Se ogni nodo ha la corretta conoscenza del proprio successore (*ring stabile*), la probabilità che un lookup sia $O(\log n)$ è molto alta
- n In pratica un ring non sarà mai stabile: a causa della dinamicità sarà in uno stato di “quasi stabile”
- n Le analisi devono essere eseguite tenendo presente questa osservazione