Massimo Benerecetti

Tabelle Hash: gestione delle collisioni

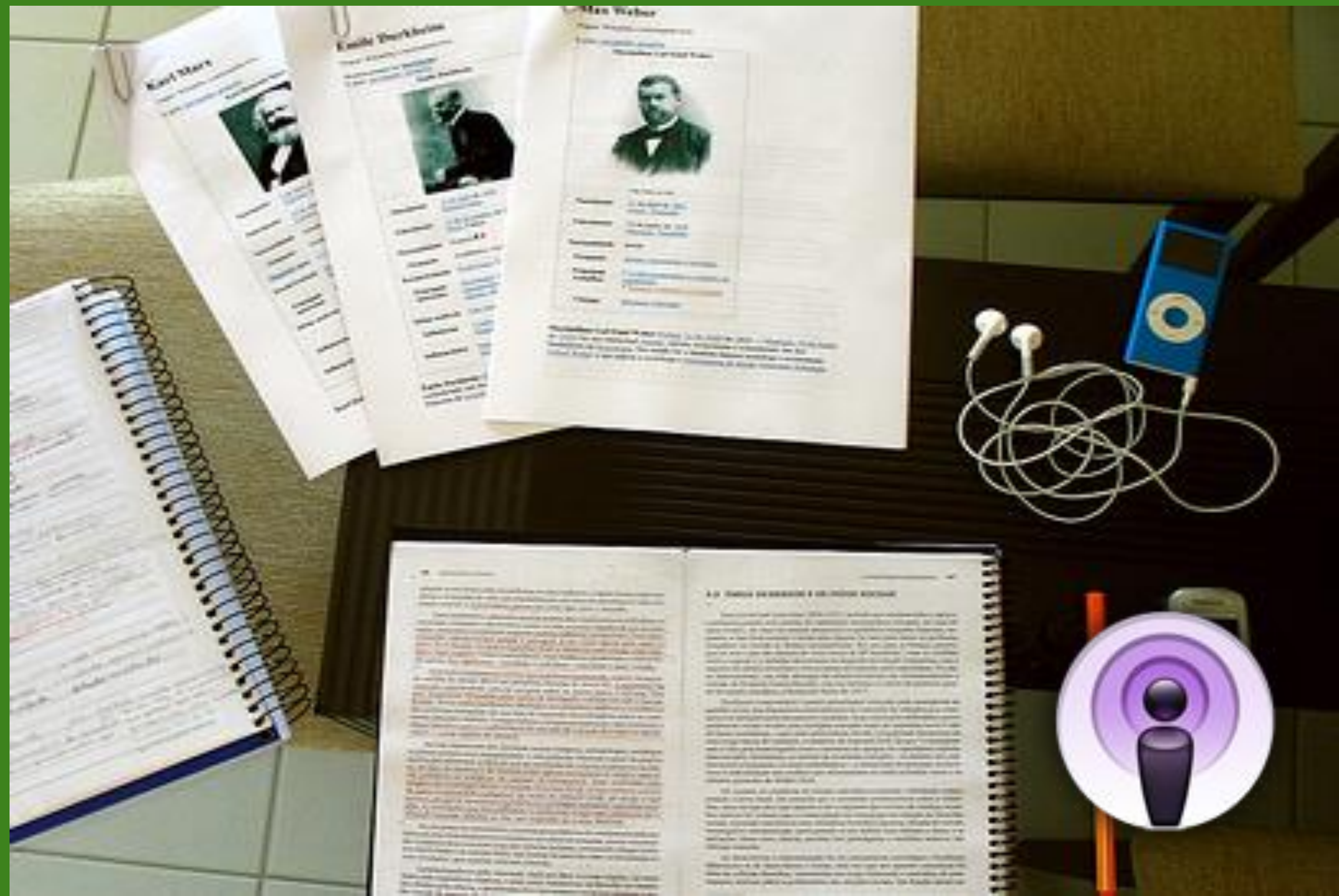
Lezione n.#
Parole chiave:

Corso di Laurea:
Informatica

Insegnamento:
Algoritmi e Strutture Dati I

Email Docente:
bene@na.infn.it

A.A. 2009-2010



Nessuno schema di hashing può garantire, in generale, l'assenza di collisioni. Si pone quindi il problema di ***come risolvere le collisioni?***

Esistono due tecniche principali:

- ***Indirizzamento aperto***
  - la tabella è una array che contiene al massimo un oggetto per indice
  - spazio di memeoria contiguo
- ***Indirizzamento chiuso (concatenamento)***
  - la tabella è un array di catene (ad esempio liste concatenate). Tutti gli elementi su una catena hanno lo stesso indice
  - spazio di memoria dinamico

Indirizzamento aperto

$$h(k) = k \bmod 10$$

- Tutti i dati sono contenuti all'interno della tabella.
- Non è necessaria memoria aggiuntiva oltre alla tabella.
- In caso di collisioni, vengono ***tentate posizioni alternative*** finché non se ne trova una vuota.
- Per avere buone prestazione è tipicamente necessaria una tabella assai più grande del numero di elementi da memorizzare.
- Il ***tasso di riempimento*** (**m/n**, numero di chiavi in tabella diviso dimensione della tabella) dovrebbe essere mantenuto al di sotto del **50%**.
- La figura a destra riporta il risultato dell'inserimento, nell'ordine, delle chiavi **0, 1, 4, 9, 16, 25, 36, 49**.
- La chiave **36** collide in posizione **6** con la chiave **16**. **36** viene così inserita nella prima cella libera in posizione **7**. Analogamente accade per la chiave **49** (che collide con **9**). La prima cella libera è, in questo caso, in posizione **2**.

0	0
1	1
2	49
3	
4	4
5	25
6	16
7	36
8	
9	9

```
Hash-Search(T,k)
  i = 0
  repeat
       j = h(k,i)
       if (T[j] = k) then
            return j
       else
            i = i + 1
  until (T[j] = NIL or i = n)
  return NIL
```

```
Hash-Insert(T,k)
  i = 0
  repeat
        j = h(k,i)
        if (T[j] = NIL) then
            T[j] = k
            return j
        else
            i = i + 1
  until i = n
  error: "Table Overflow"
```

Data una funzione di hash ***h*(*k*)** (detta ***hash primario***) definiamo una nuova ***funzione di hash h*(*k,i*)**, con parametri la chiave ***k*** e l'indice ***i*** del sondaggio tale che ***h*(*k,0*)= *h*(*k*)**.

- Tentare altre celle
  - Le celle ***h(x,0)***, ***h(x,1)***, ***h(x,2)***, **...** vengono sondate in successione finché non se ne trova una libera.
  - La ***funzione di Hash h(x,i)*** ideale soddisfa la condizione di ***Hashing Uniforme***:

    «ogni chiave k ha la stessa probabilità di determinare, come sequenza di sondaggi, una qualsiasi delle n! permutazioni della sequenza 0,1,...,n-1»
  - In pratica si riescono a realizzare funzioni di Hash che ***approssimano*** più o meno bene l'***Hashing Uniforme***.

Funzione di hash con sondaggi

- Tentare altre celle
  - Le celle ***h(x,0)***, ***h(x,1)***, ***h(x,2)***, **...** vengono sondate in successione.
  - $h(x,i) = (h(x) + f(i)) \bmod TSIZE$
  - $f(0) = 0$
  - ***f*** definisce la ***strategia di risoluzione delle collisioni***.
- ***Sondaggio (probing) lineare***
  - $f(i) = i$
- ***Sondaggio (probing) quadratico***
  - $f(i) = i^2$

Sondaggio lineare

A titolo di esempio, definiamo il seguente schema di hashing con sondaggio lineare:

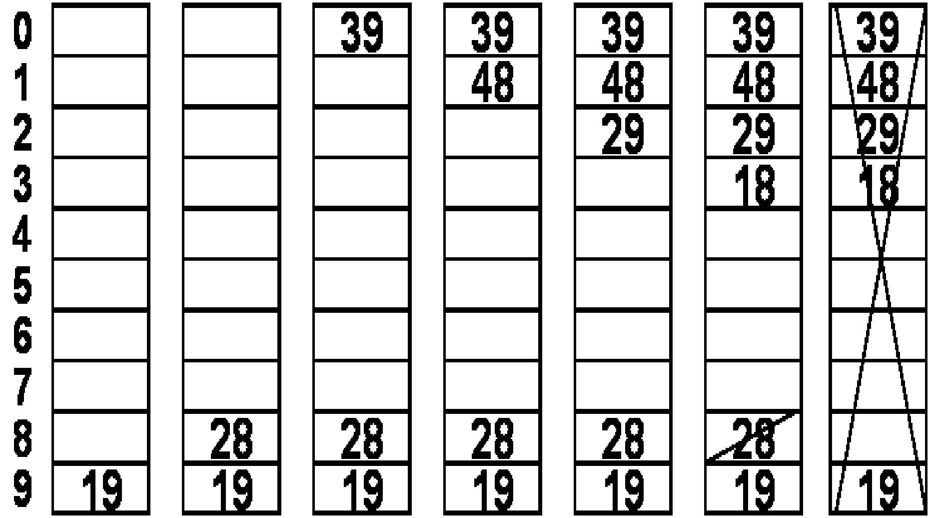
- $h(x) = x \bmod \text{TZISE}$ **e** $f(i) = i$
- $h(x,i) = (\ h\ (x) + f(i)\ ) \bmod \text{TSIZE}$
  $= (\ x + i\ ) \bmod \text{TSIZE}$

Nella figura a fianco è riportata la simulazione dello schema a fronte delle seguenti operazioni:

– **Inserimento** di 19, 28, 39, 48, 29, 18
– **Cancellazione** di 28
– **Ricerca** di 18

Si noti che l'eventuale cancellazione fisica di **28** dalla tabella impedirebbe una ricerca con successo di **18**.

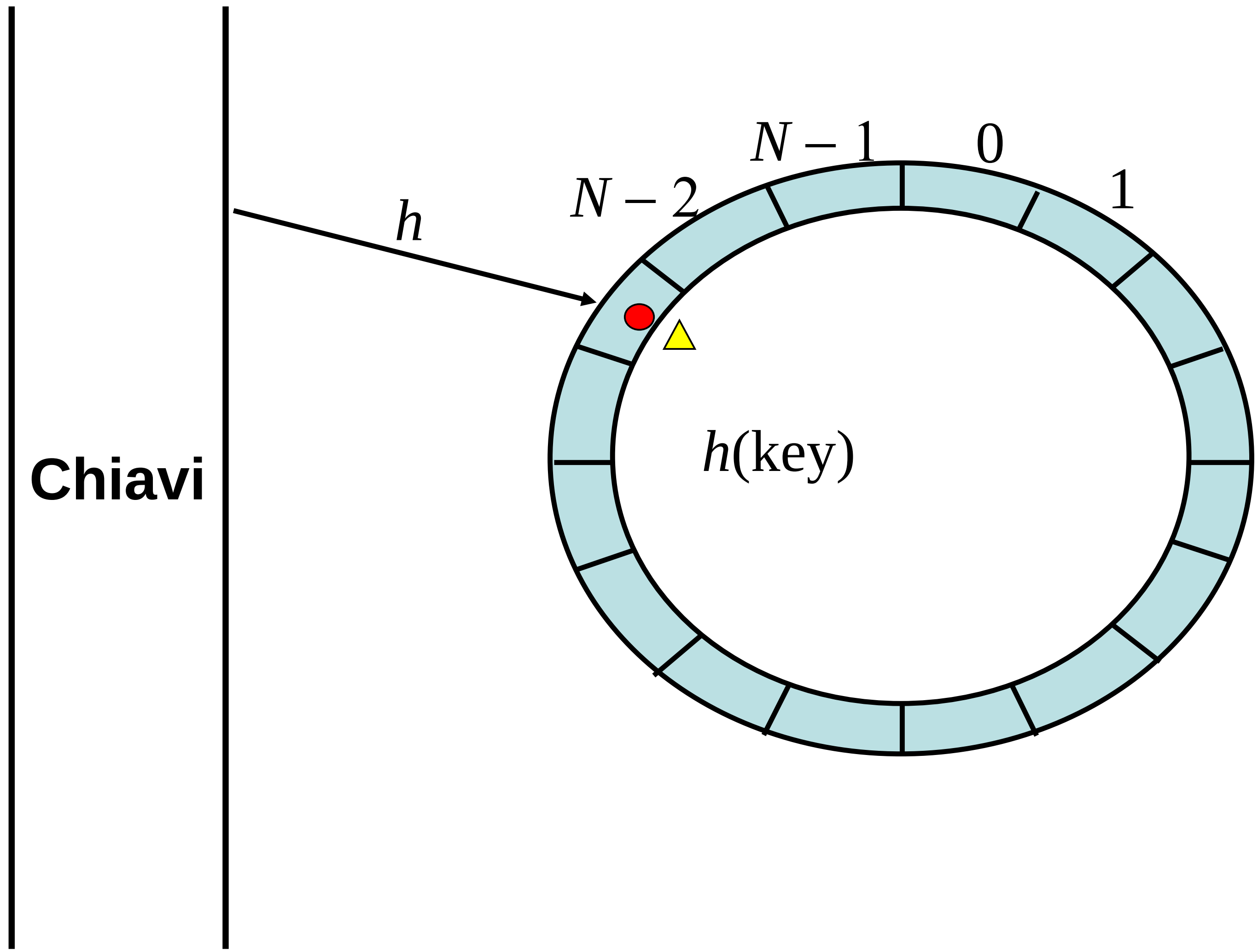
Infatti, **la cancellazione di 28 interromperebbe la catena dei sondaggi necessari a trovare 18**.



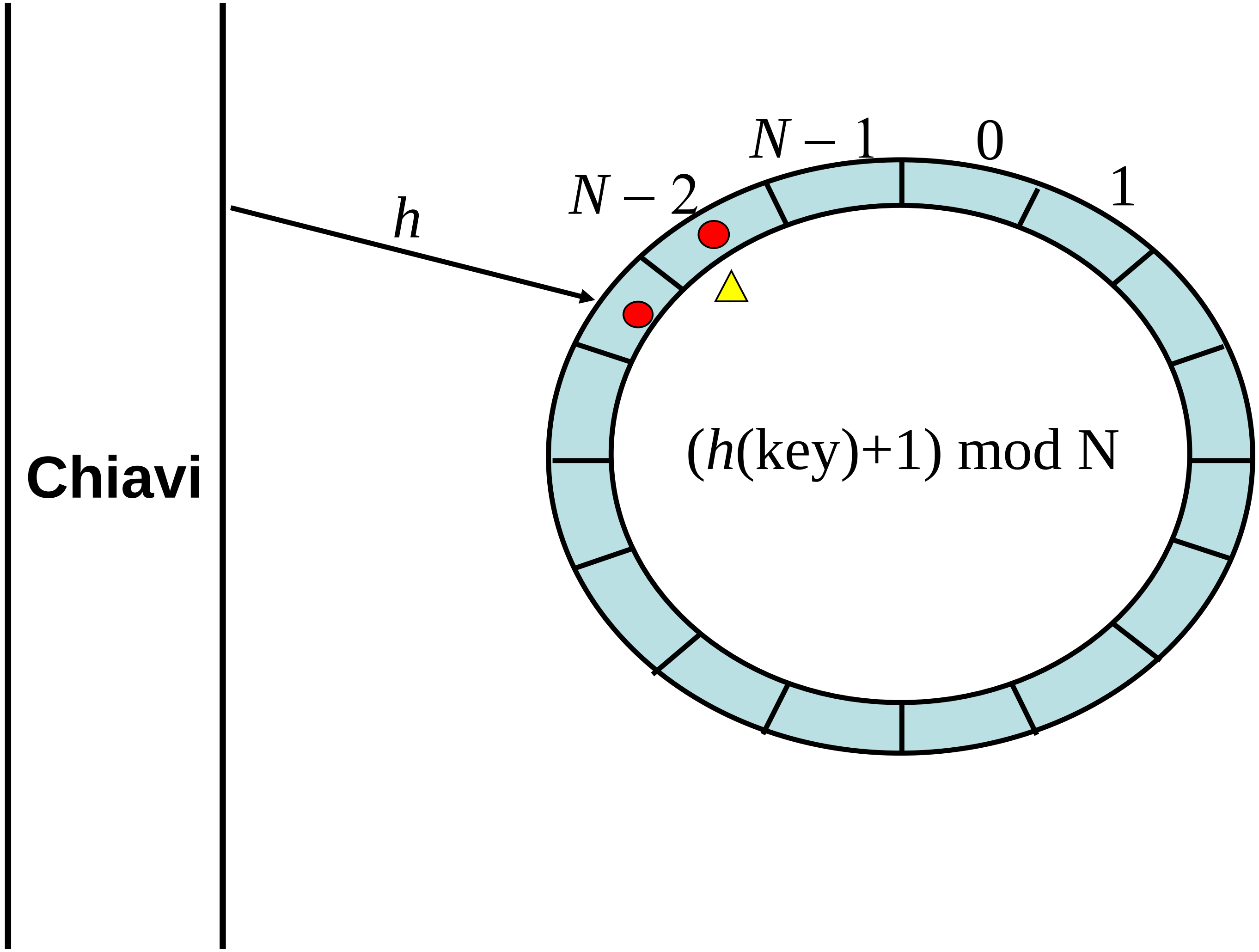
Esempio di operazioni su tabella hash con sondaggio lineare

Sondaggio (probing) lineare

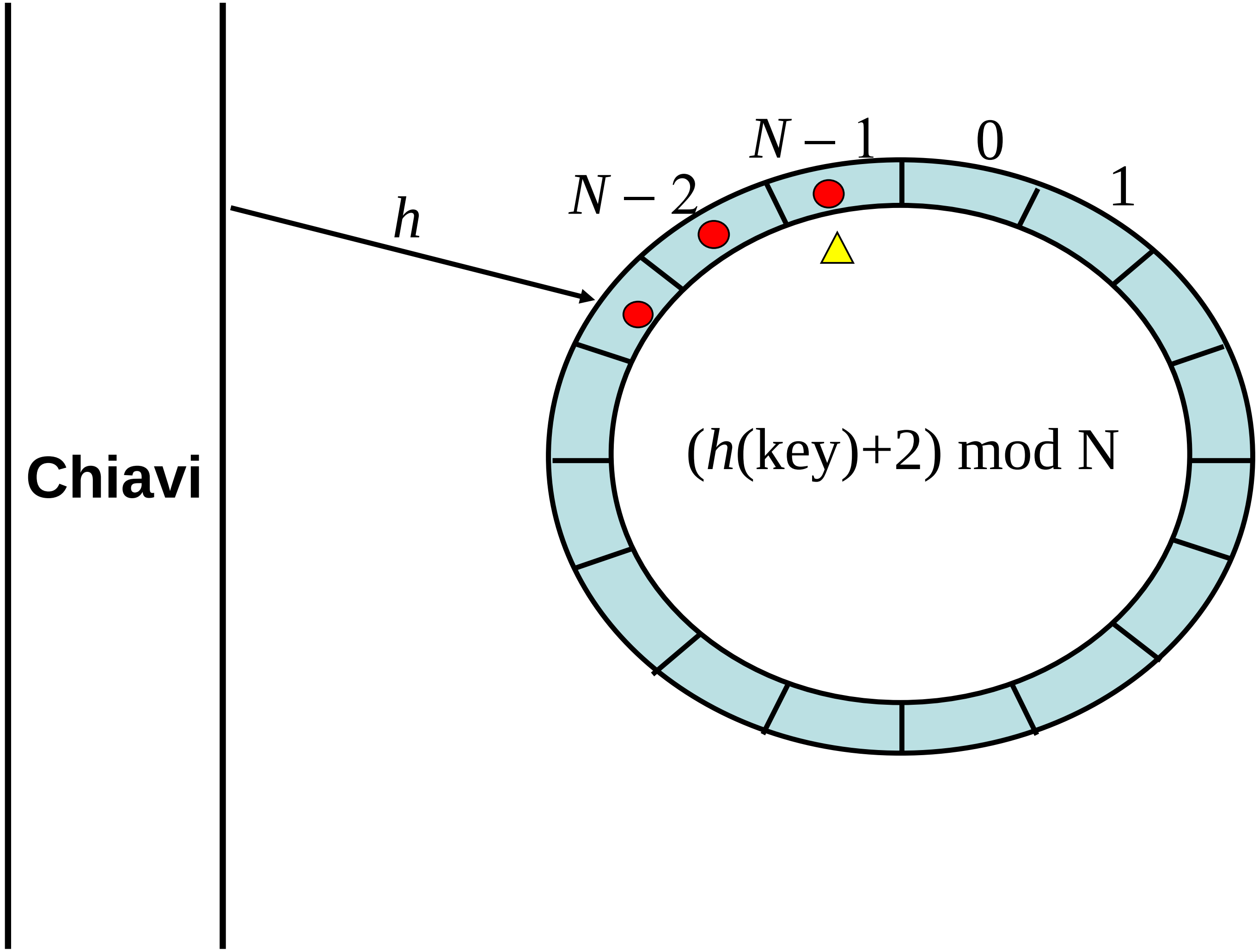
- È il *metodo* di risoluzione delle collisioni *più semplice*
- Dato un *indice hash*, esegue una ricerca lineare della chiave desiderata una locazione vuota (per l'inserimento)
- La tabella è vista come un ***array circolare***, al raggiungimento dell'ultimo indice, la ricerca riparte dal primo indice (realizzato tramite operazione di modulo)
- Se la tabella non è piena, è sempre possibile trovare una posizione vuota.



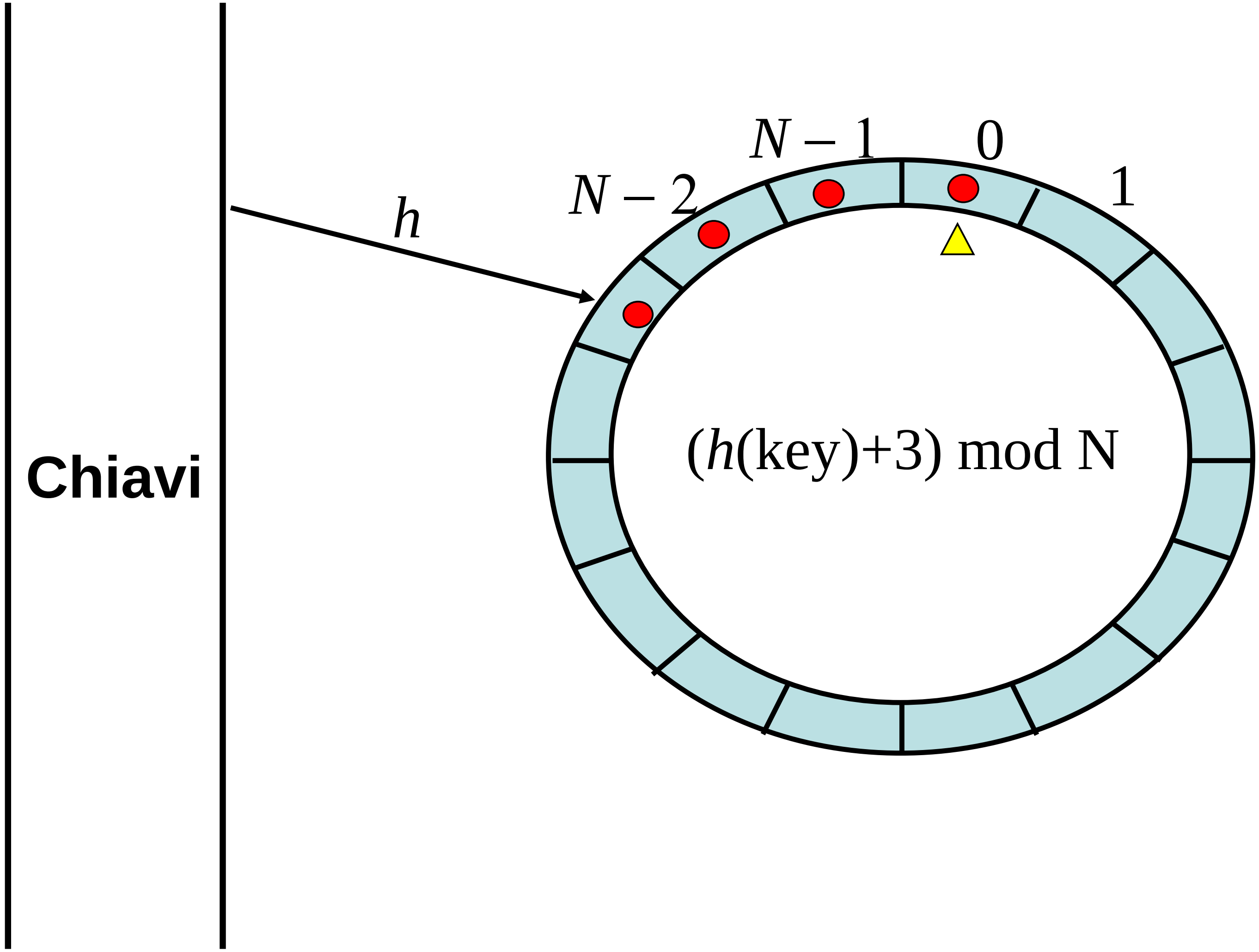
Simulazione del sondaggio lineare.



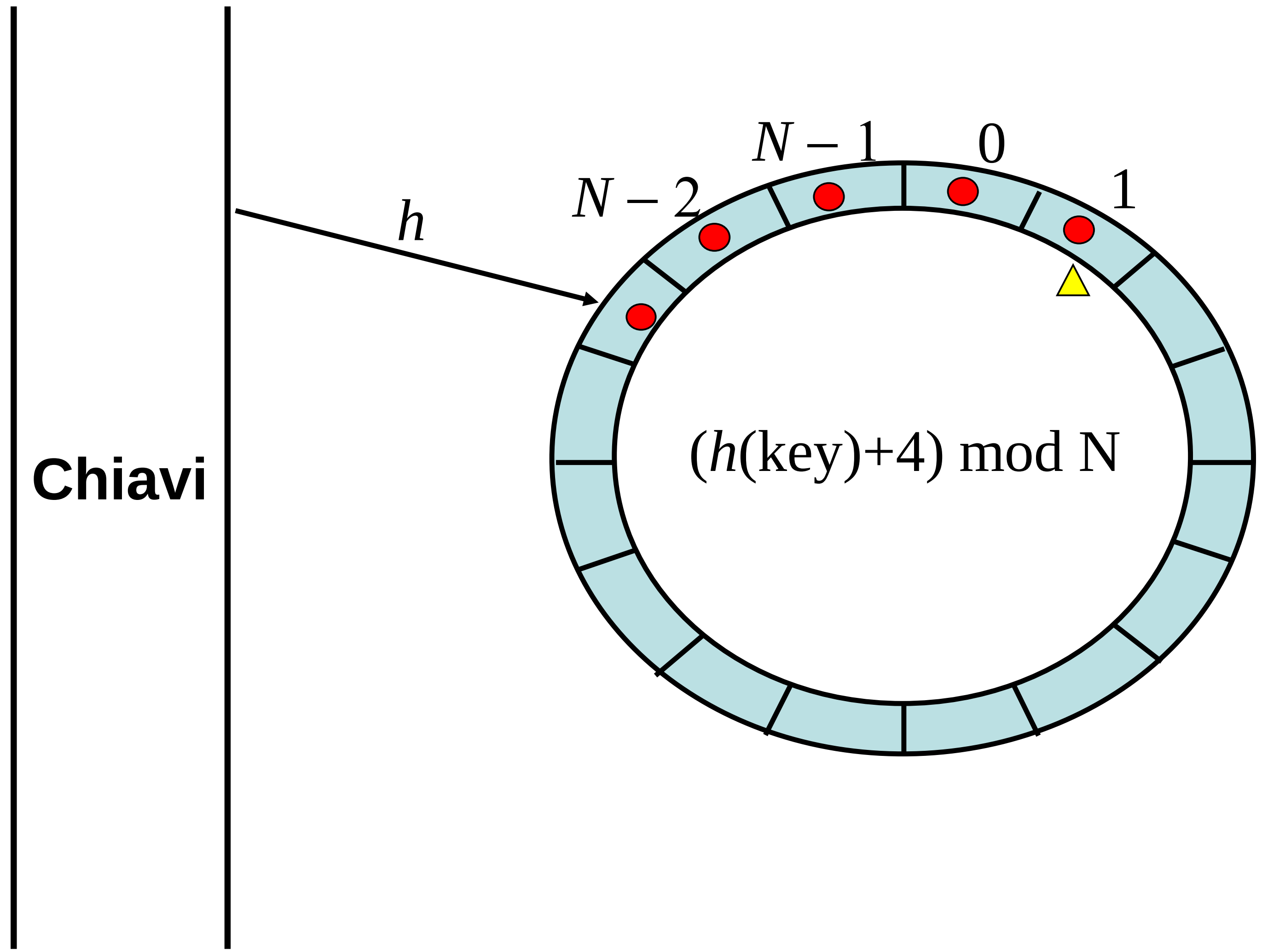
Simulazione del sondaggio lineare.



Simulazione del sondaggio lineare.



Simulazione del sondaggio lineare.



Simulazione del sondaggio lineare.

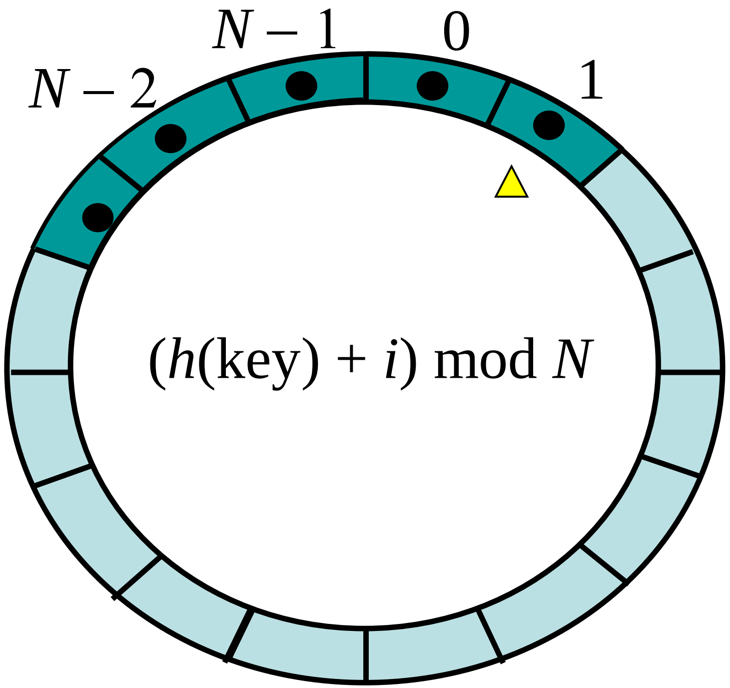
Clustering primario

- Lo *svantaggio* maggiore del ***sondaggio lineare*** è che quando la tabella diventa piena quasi per metà, si verifica una tendenza al "**Clustering**" (raggruppamento) ***primario*** (sul valore primario dell'hash)

- Il "**clustering primario**" si verifica quando gli elementi iniziano a delinearsi come lunghe file di posizioni adiacenti occupate. Tutte le chiavi il cui hash è nel cluster, aumentano la dimensione del cluster.
- Anche con chiavi che hanno valori di ***hash*** diversi dalla chiave che ha dato origine al cluster.

- La ricerca lineare di una locazione libera (vuota) diviene sempre più lunga

- Il "**clustering primario**" si verifica quando gli *elementi* iniziano a delinearsi come ***lunghe file di posizioni adiacenti occupate***.
- Tutte le chiavi il cui ***hash ricade nel cluster*** aumentano la dimensione del cluster, anche chiavi che hanno valori di ***hash*** primari diversi quello dalla chiave che ha dato origine al cluster.
- La ***ricerca*** di una ***locazione libera*** (vuota) diviene ***sempre più lunga***.

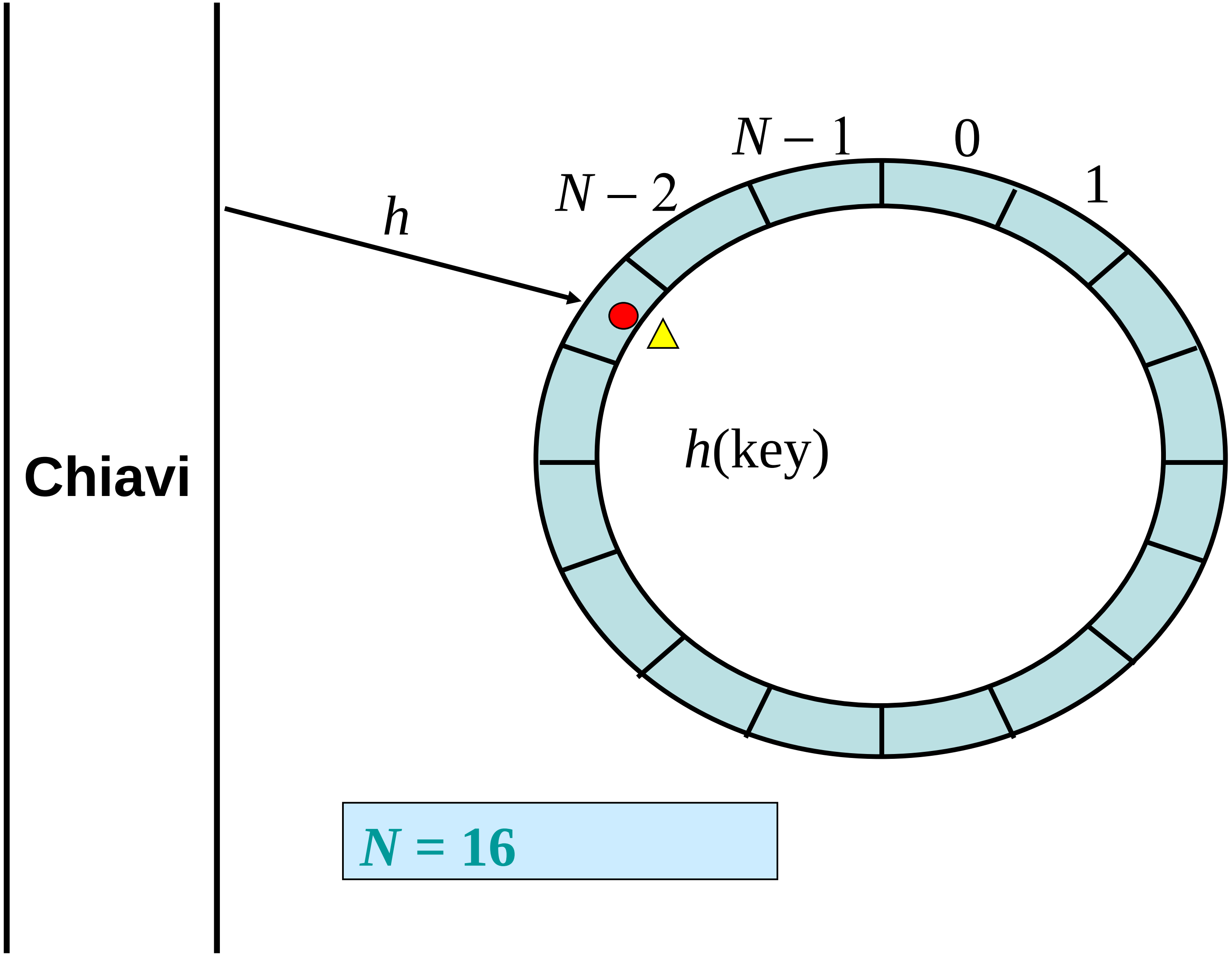


Sondaggio (probing) quadratico

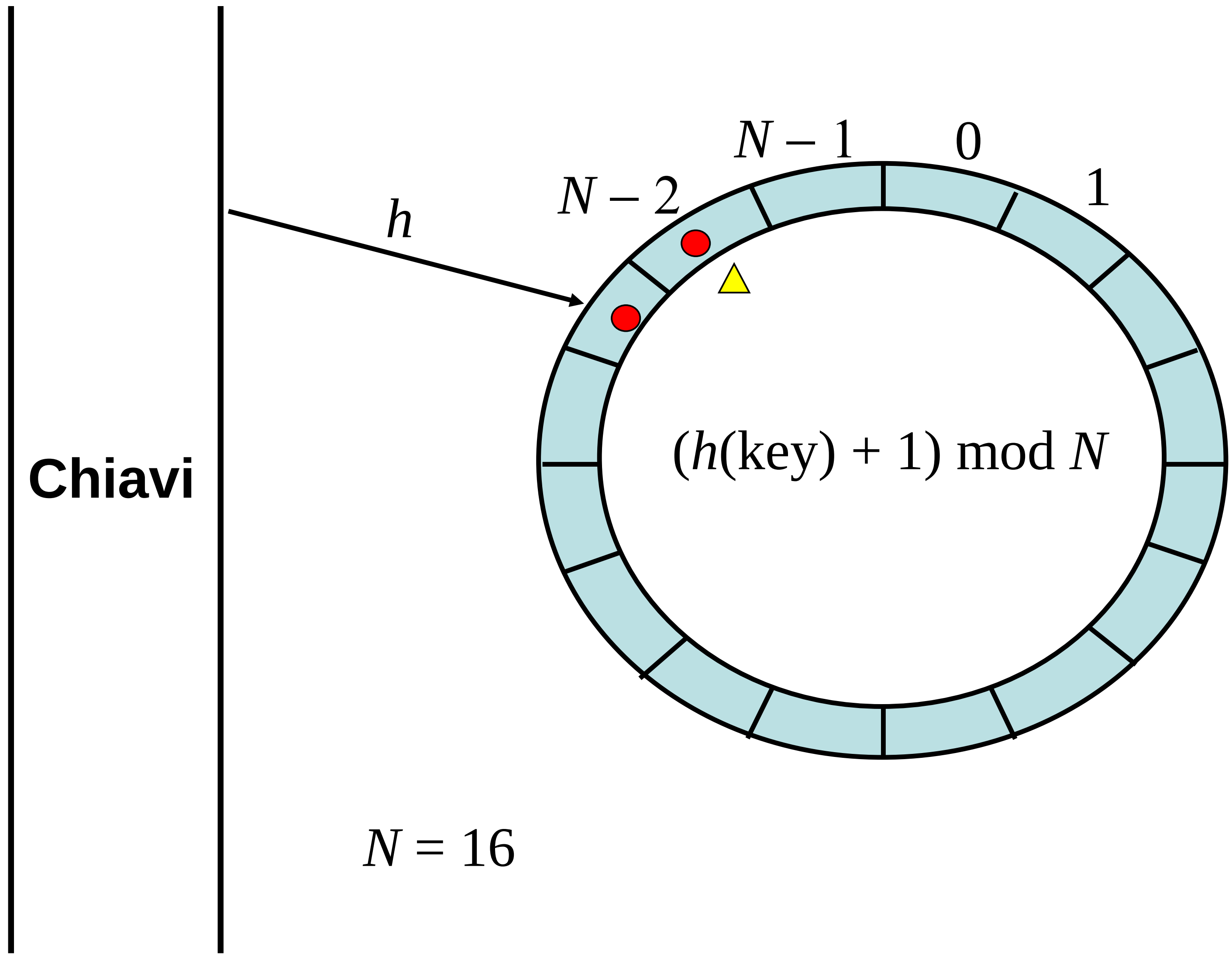
- Se ***hash(key) = h***, tentare le locazioni $h+1$, $h+4$, $h+9$, $h+16$, ecc. Cioè tentare le locazioni

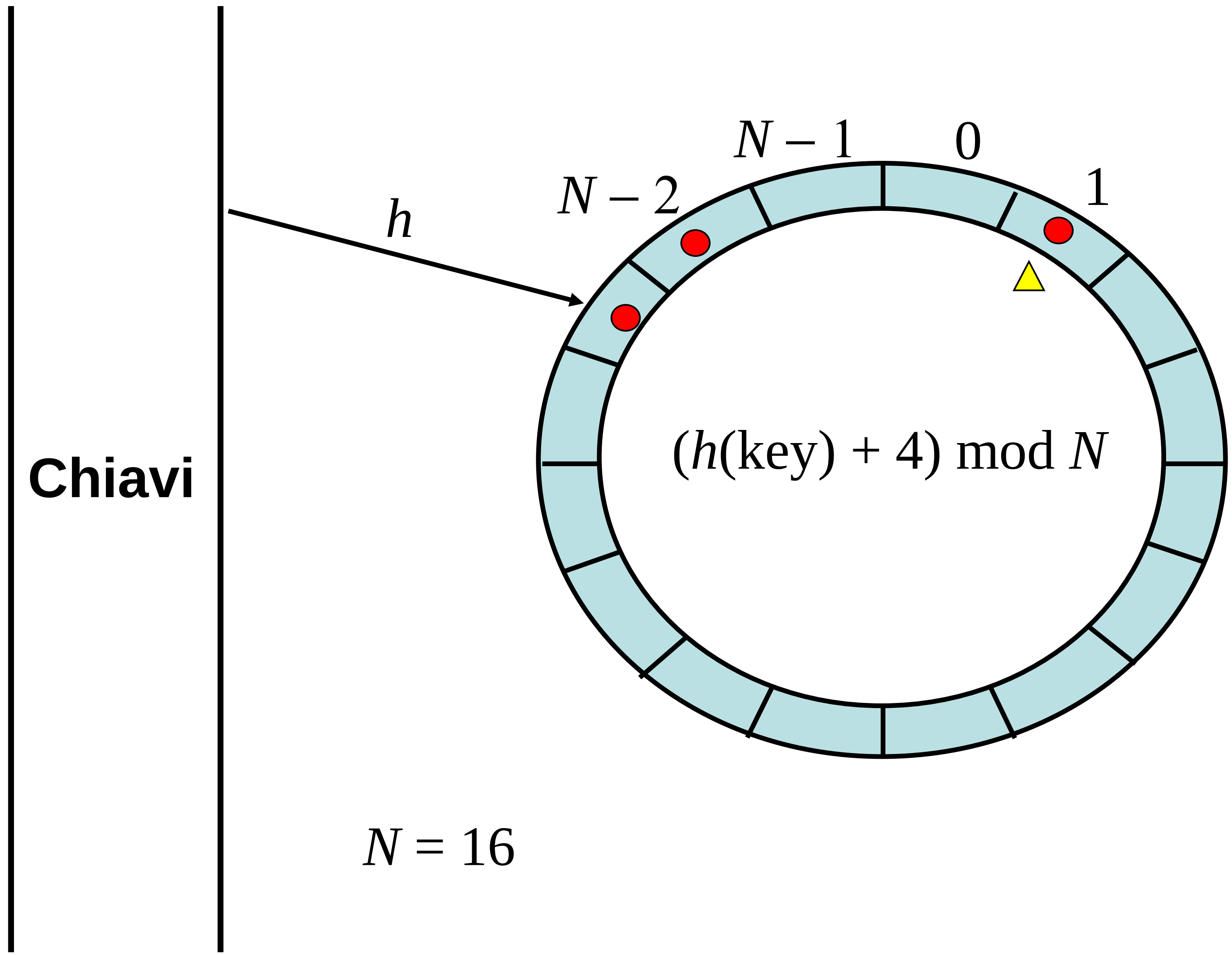
$$h + i^2 \qquad \text{per } i = 1,2,3,4,\ldots$$

- Elimina il problema del ***clustering primario*** (i sondaggi non sono contigui)
- Ma anche se la tabella non è piena, ***non*** è garantito che si riesca a trovare una cella vuota.
- Gli elementi che hanno lo ***stesso valore di hash primario***, indirizzeranno lo stesso insieme di celle alternative:
  - ***clustering secondario***
  - in pratica non costituisce un grosso problema

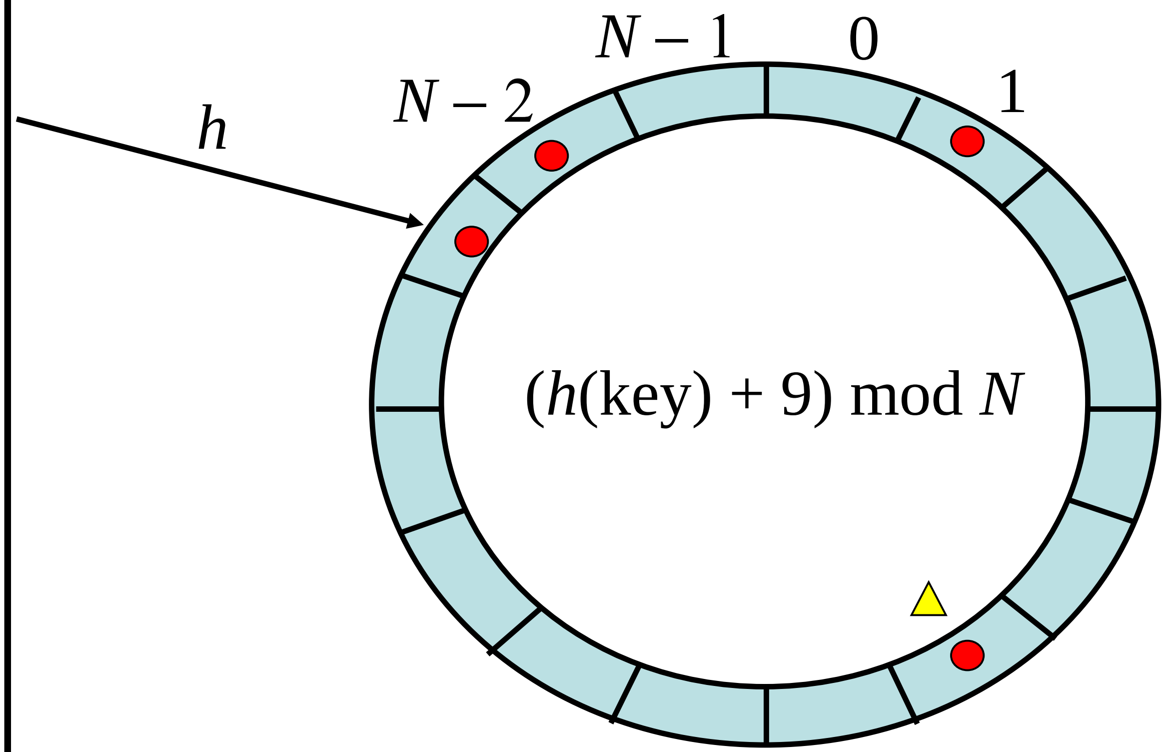


$N = 16$

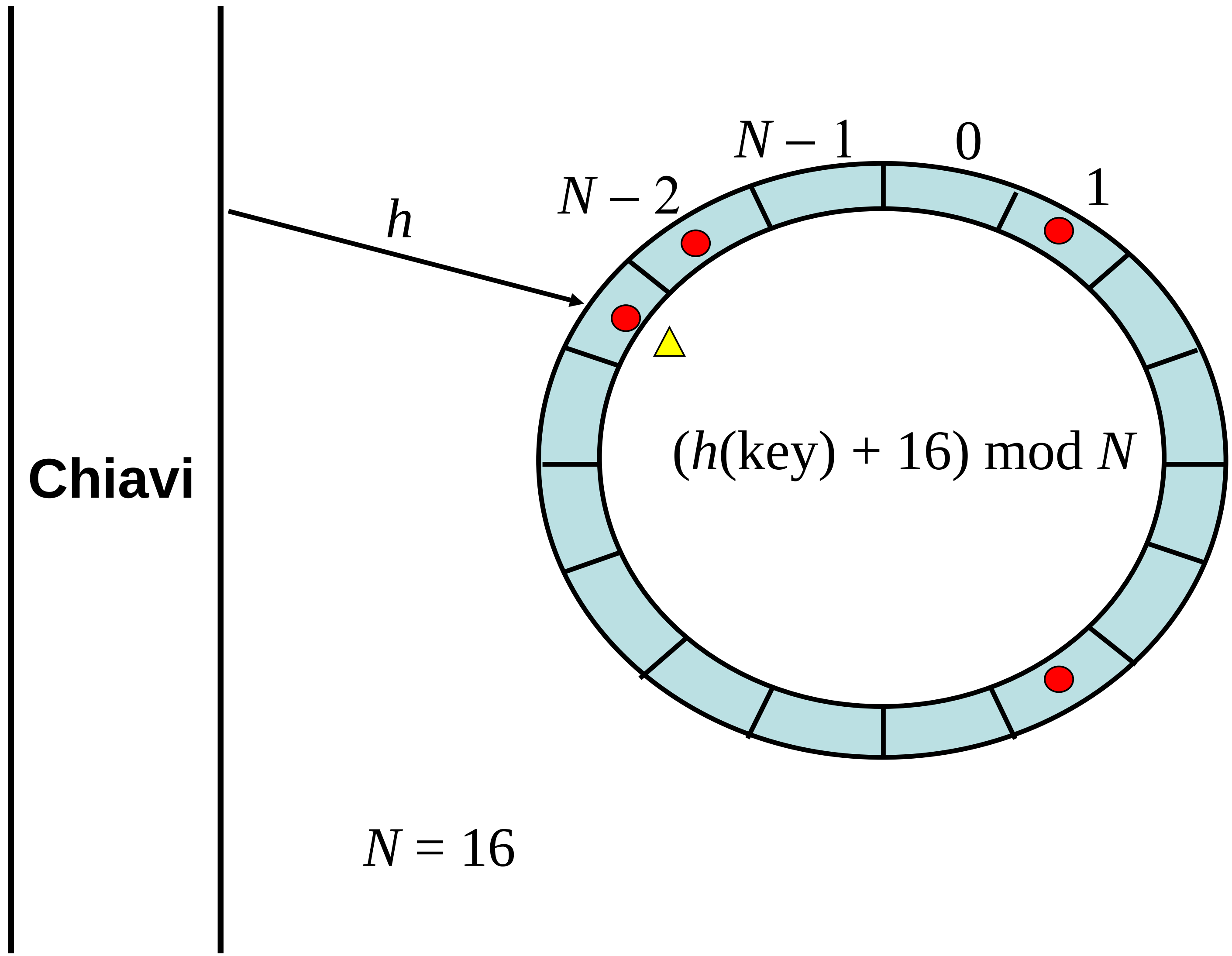

Chiavi
h
N – 2
N – 1
0
1
(h(key) + 1) mod N
N = 16

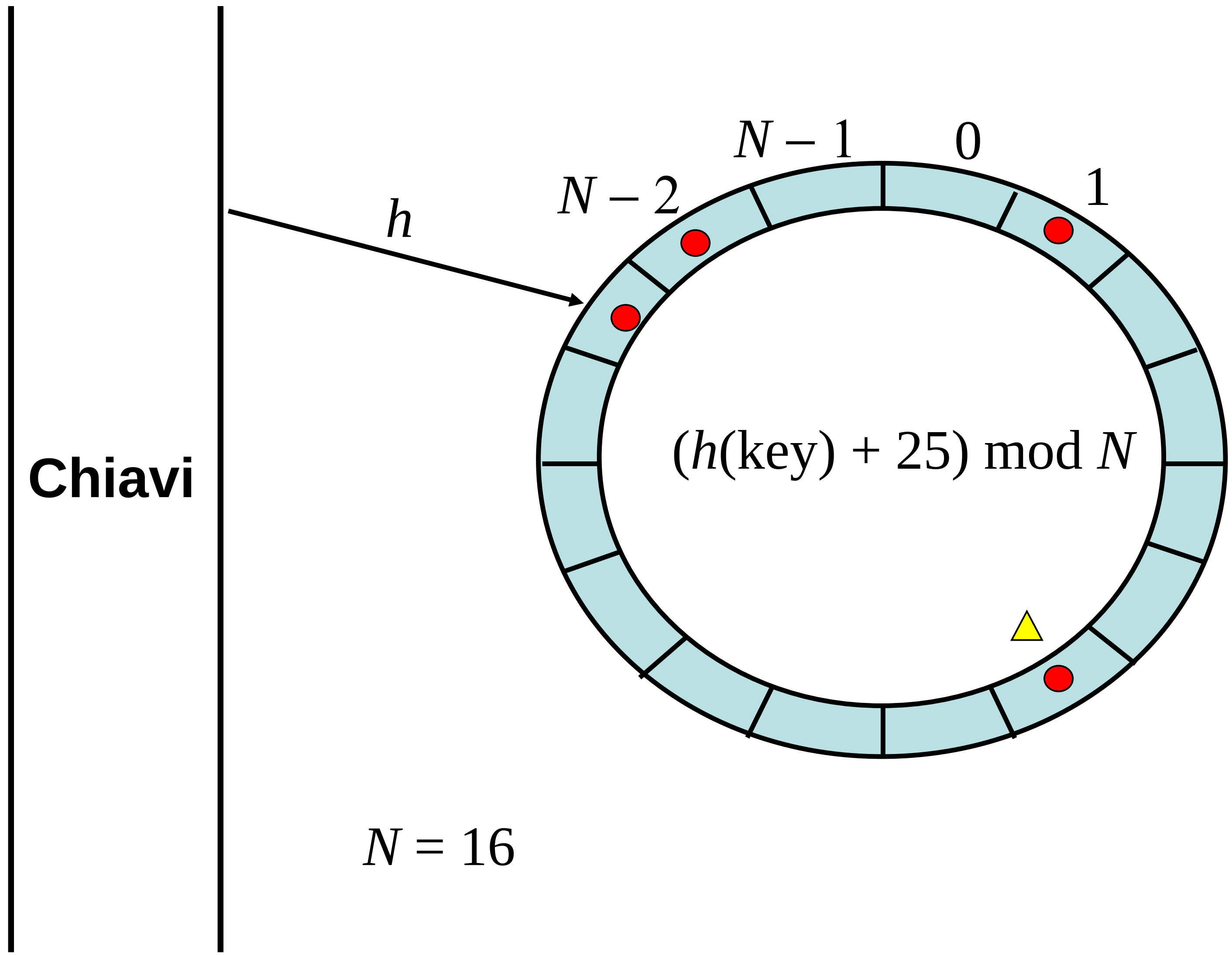

Chiavi
h
N – 2
N – 1
0
1
(h(key) + 4) mod N
N = 16

Chiavi

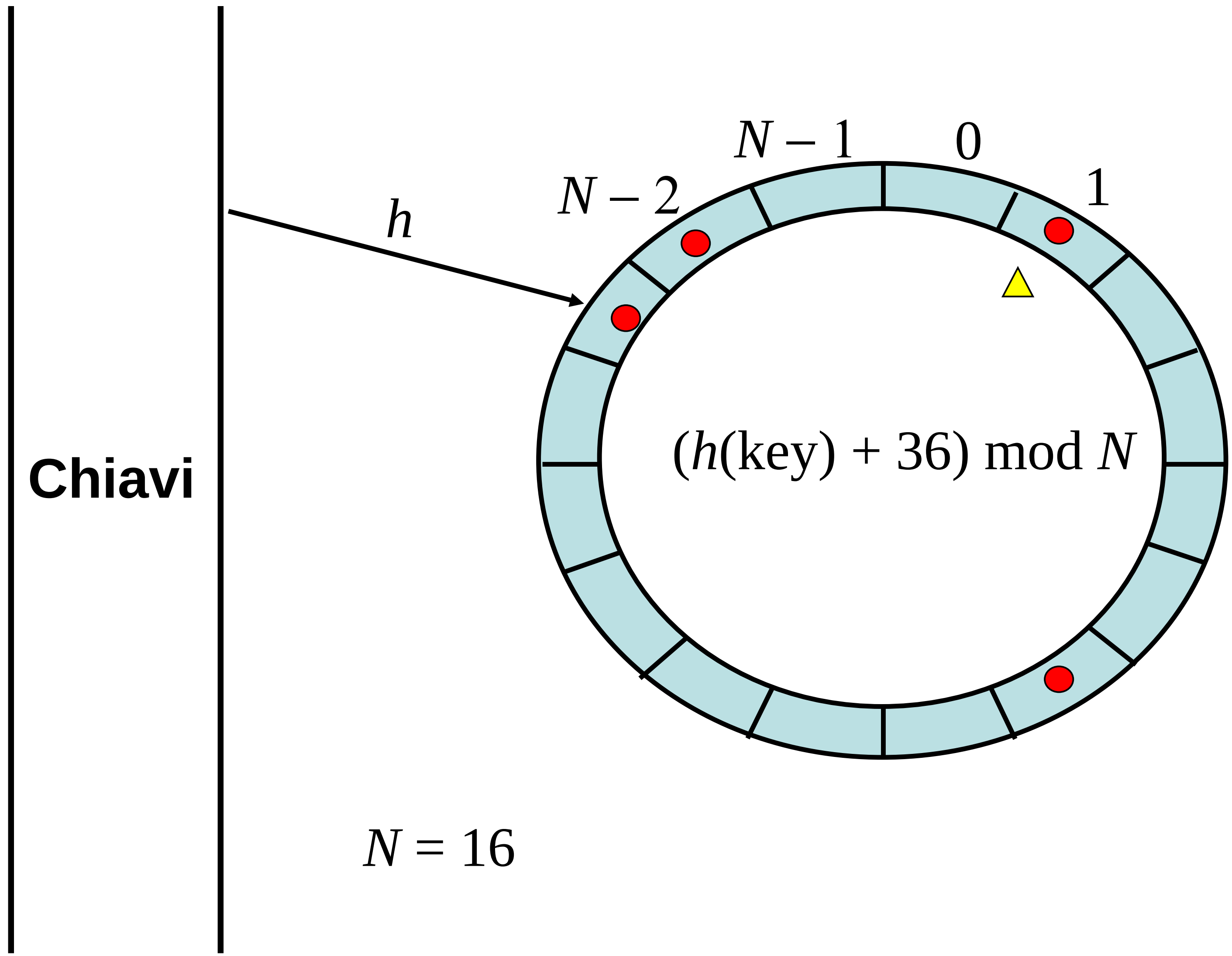


$N = 16$

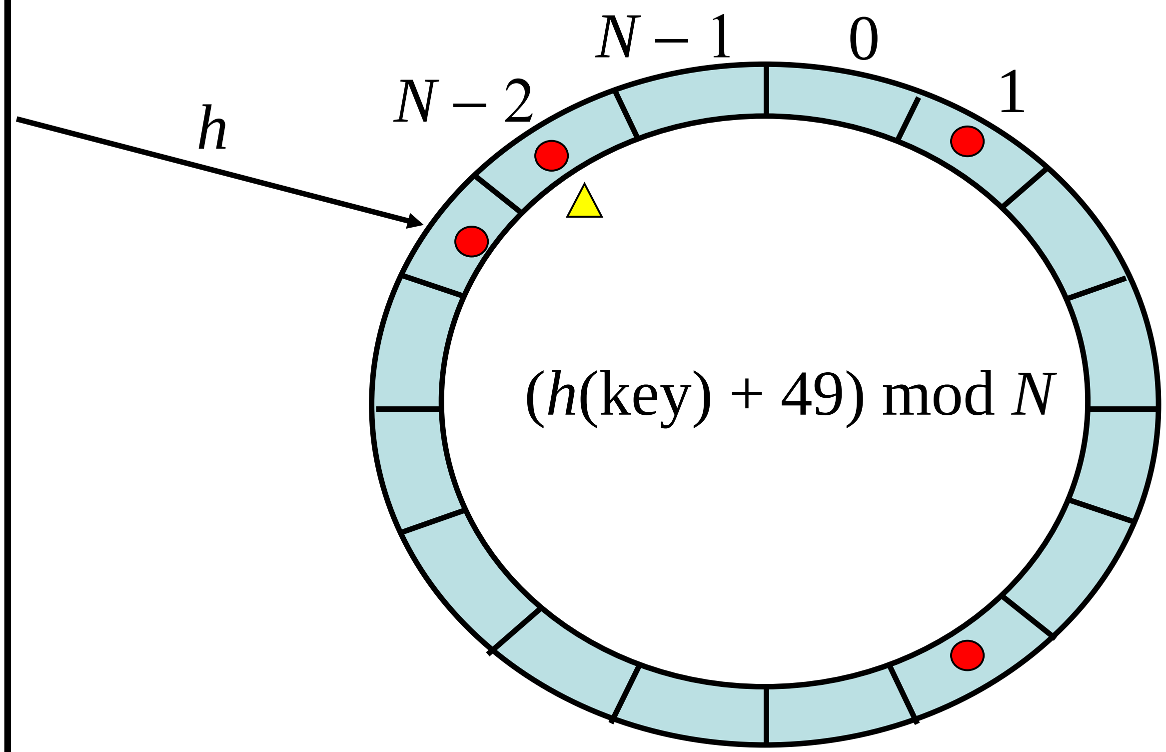

Chiavi
h
N – 2
N – 1
0
1
(h(key) + 16) mod N
N = 16



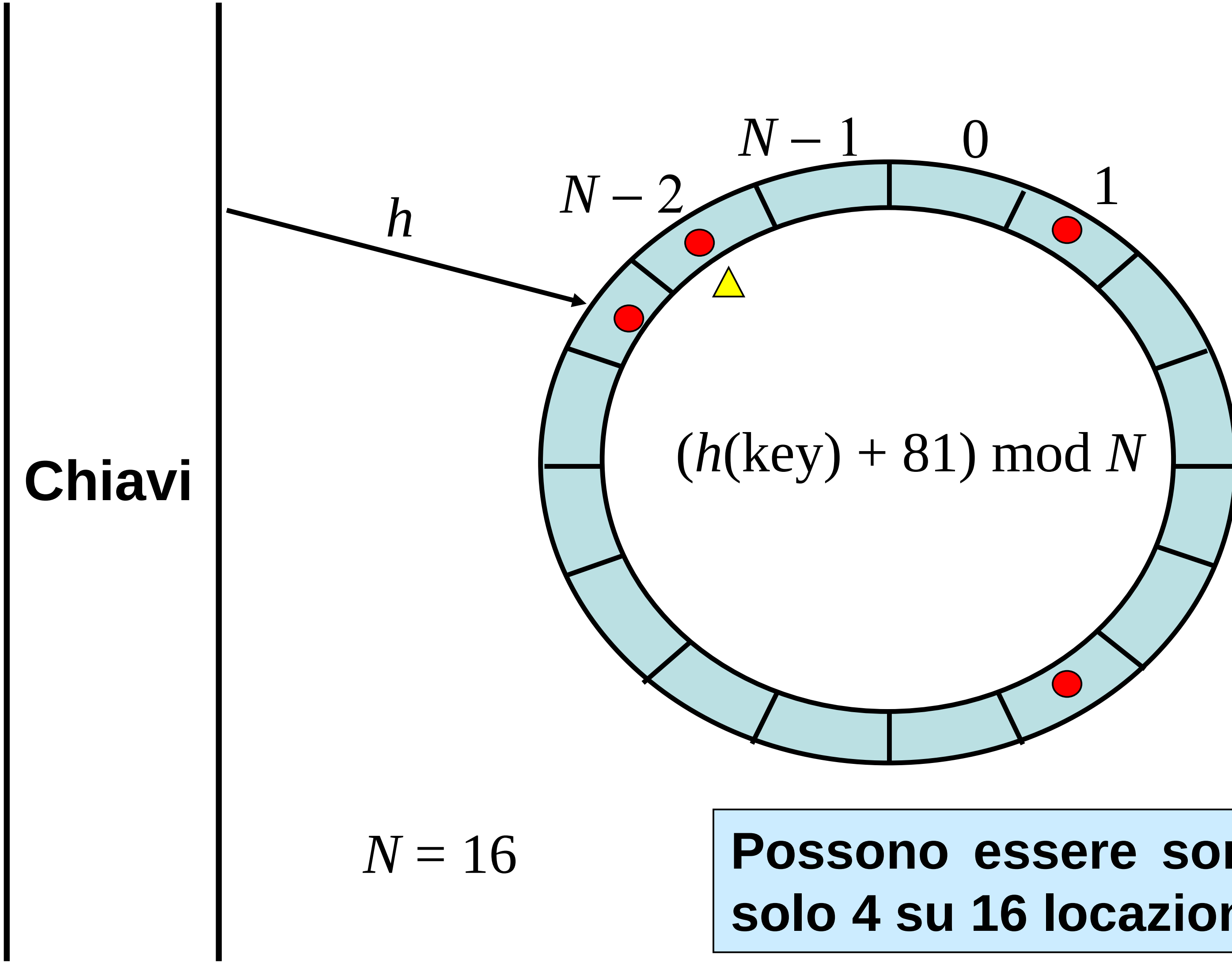
Chiavi
h
N – 2
N – 1
0
1
(h(key) + 25) mod N
N = 16


Chiavi
h
N – 2
N – 1
0
1
(h(key) + 36) mod N
N = 16

Chiavi

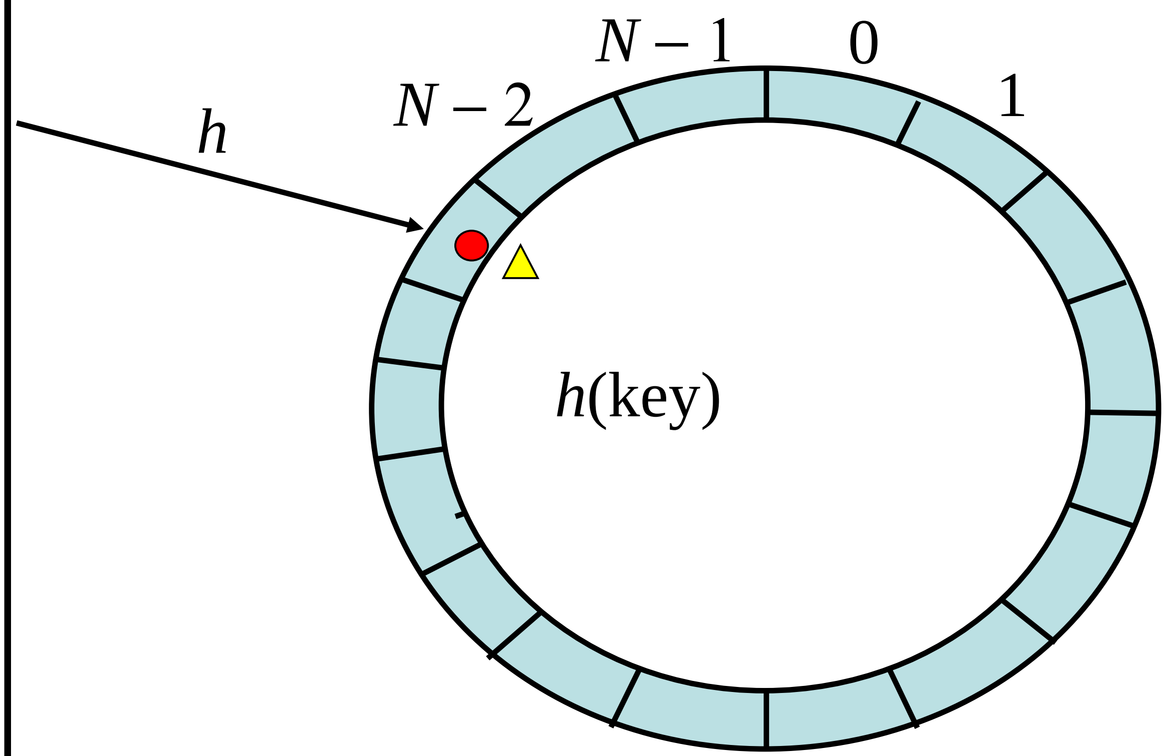


$N = 16$



Possono essere sondate solo 4 su 16 locazioni!

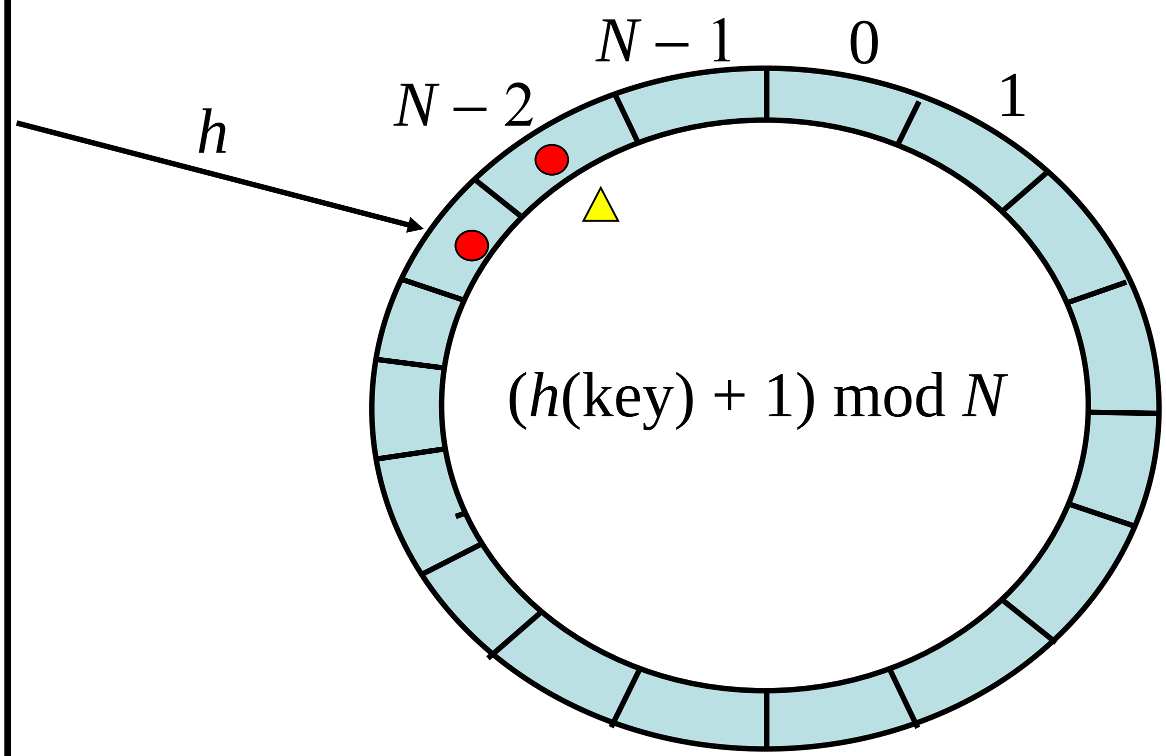
Chiavi



$N = 17$ **(primo)**

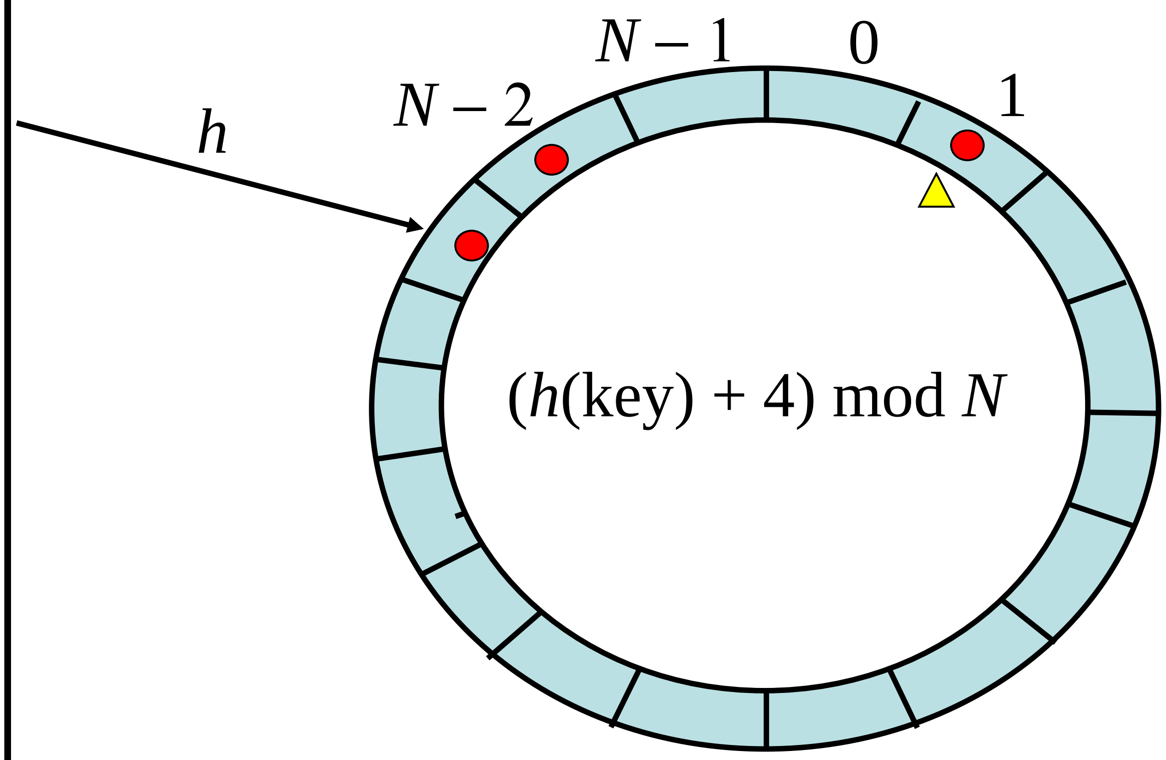
Proviamo a cambiare la dimensione della tabella

Chiavi



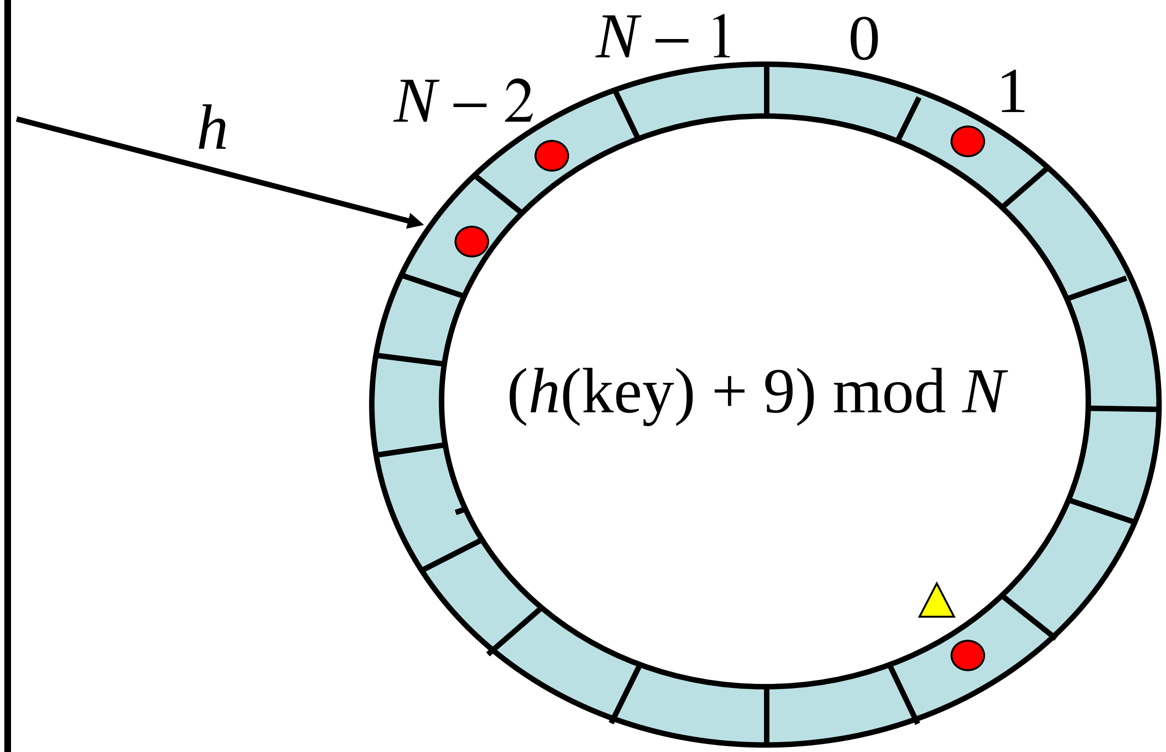
N **= 17 (primo)**

Chiavi



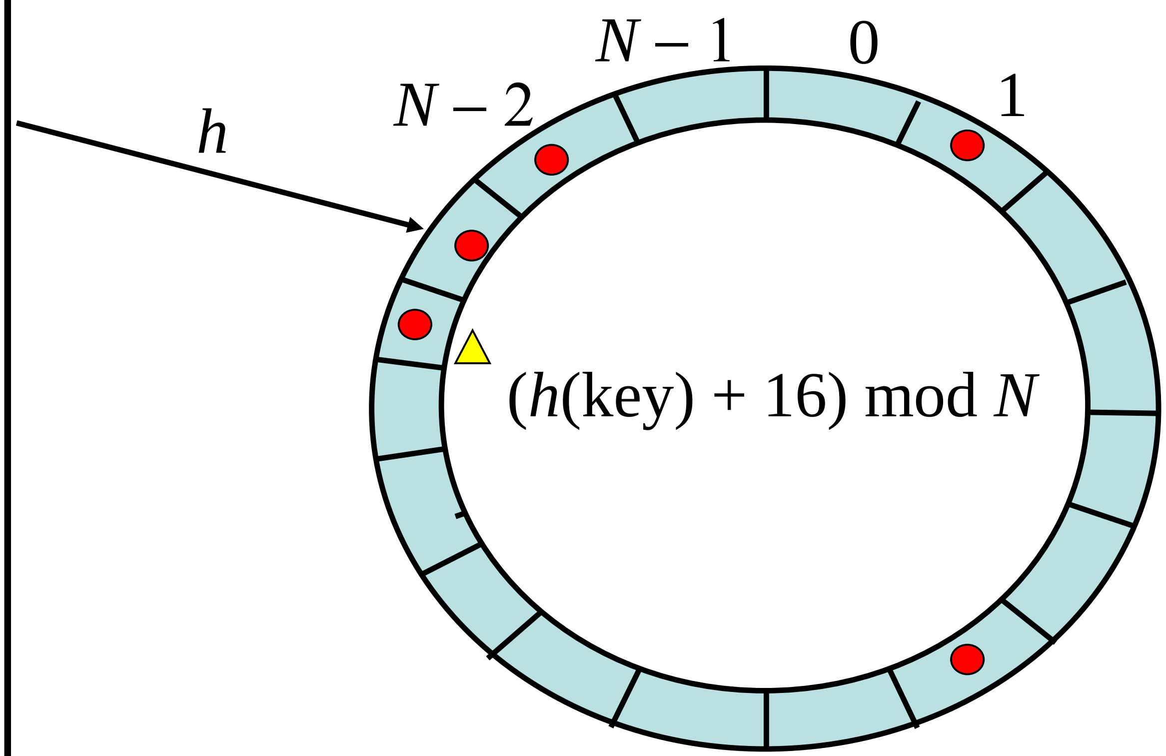
N = 17 (primo)

Chiavi

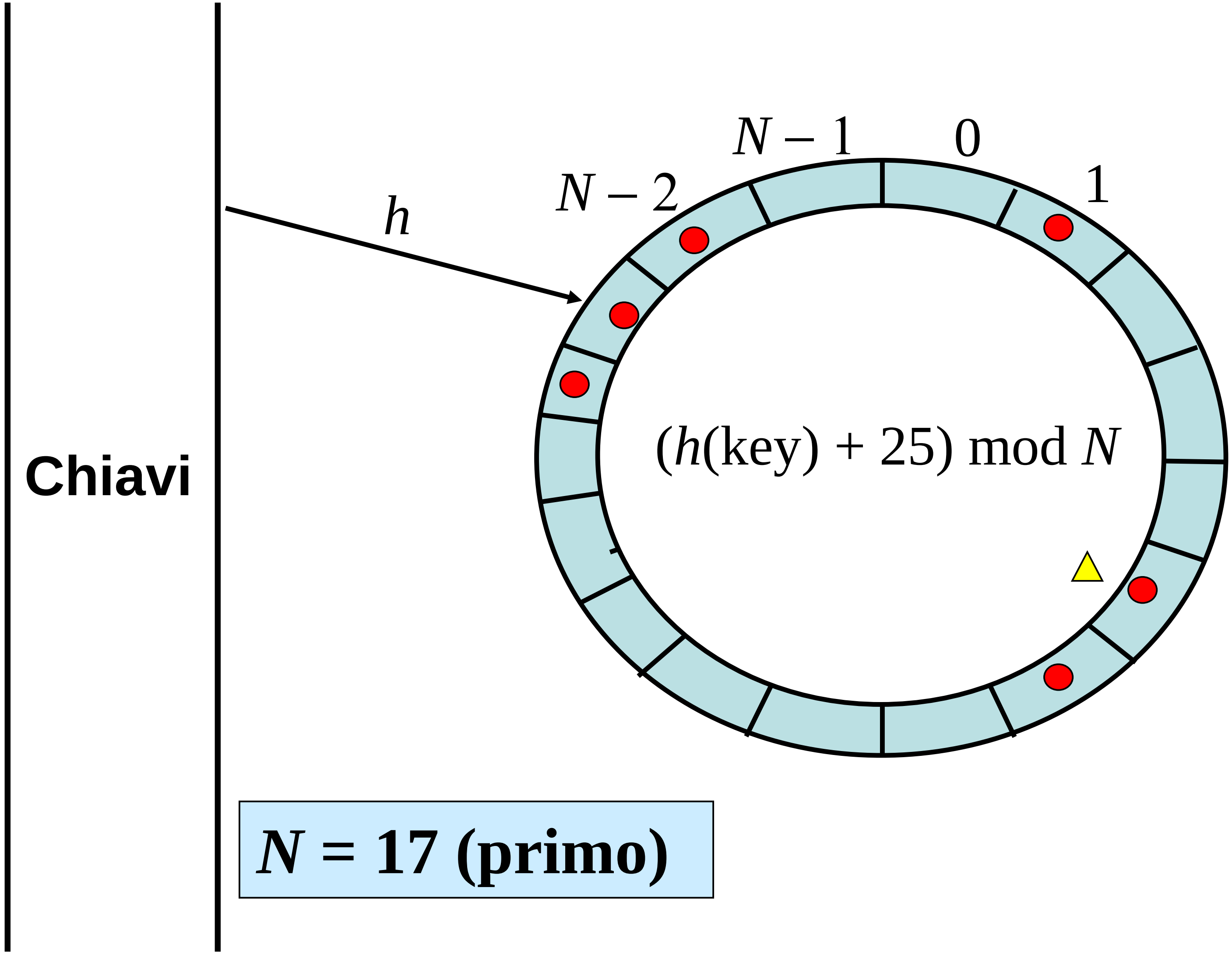


N = 17 (primo)

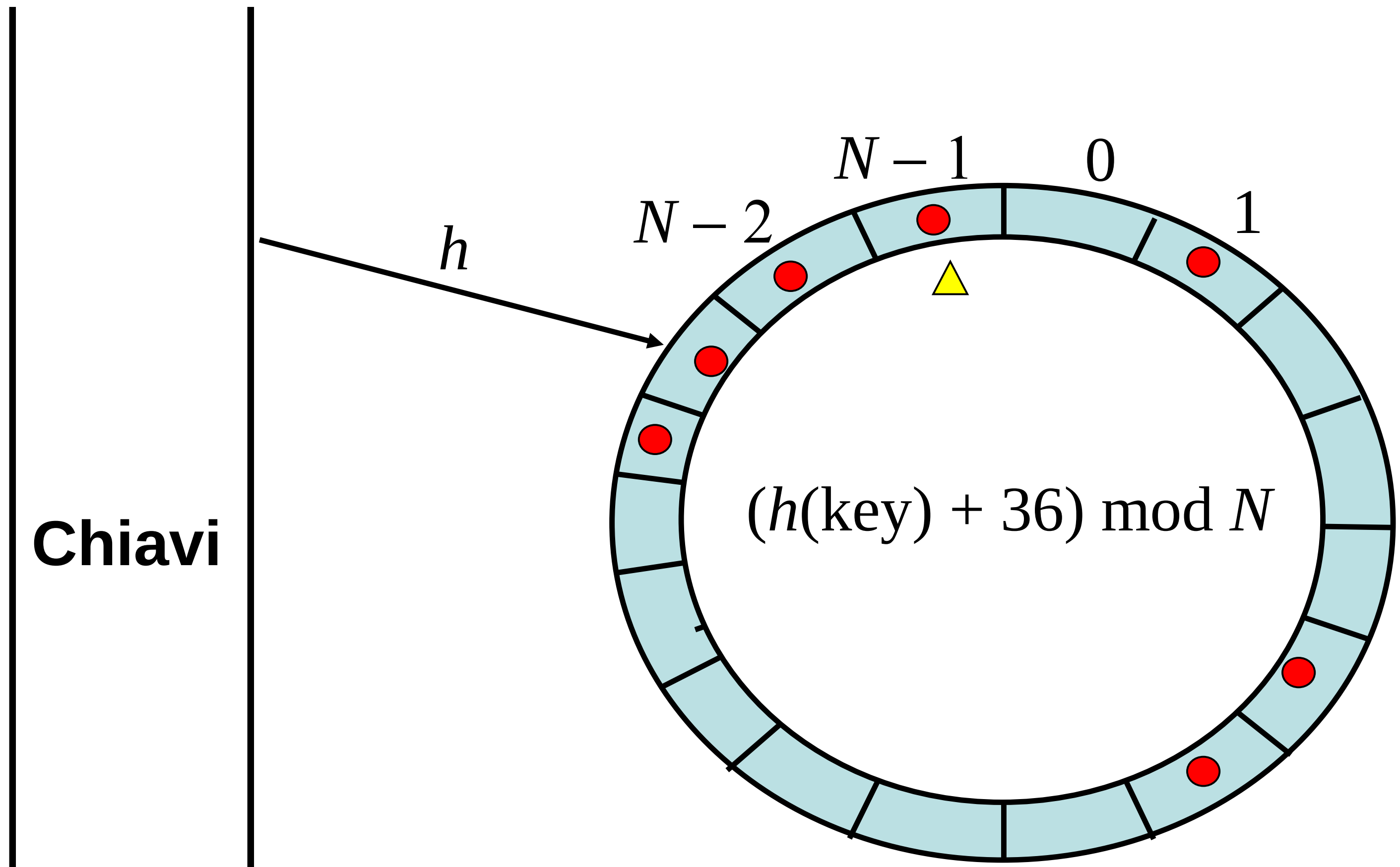
Chiavi



N **= 17 (primo)**

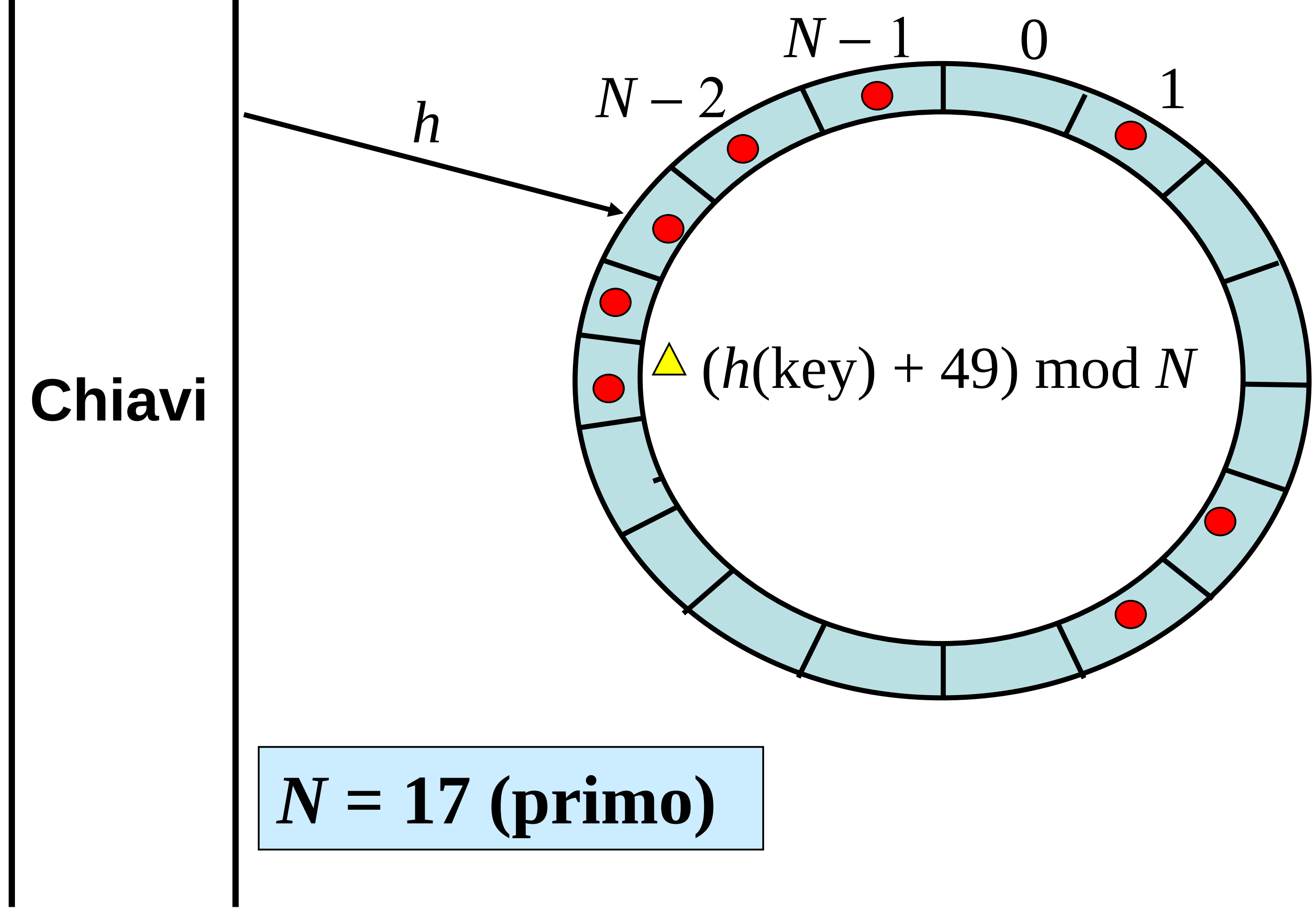


N = 17 (primo)

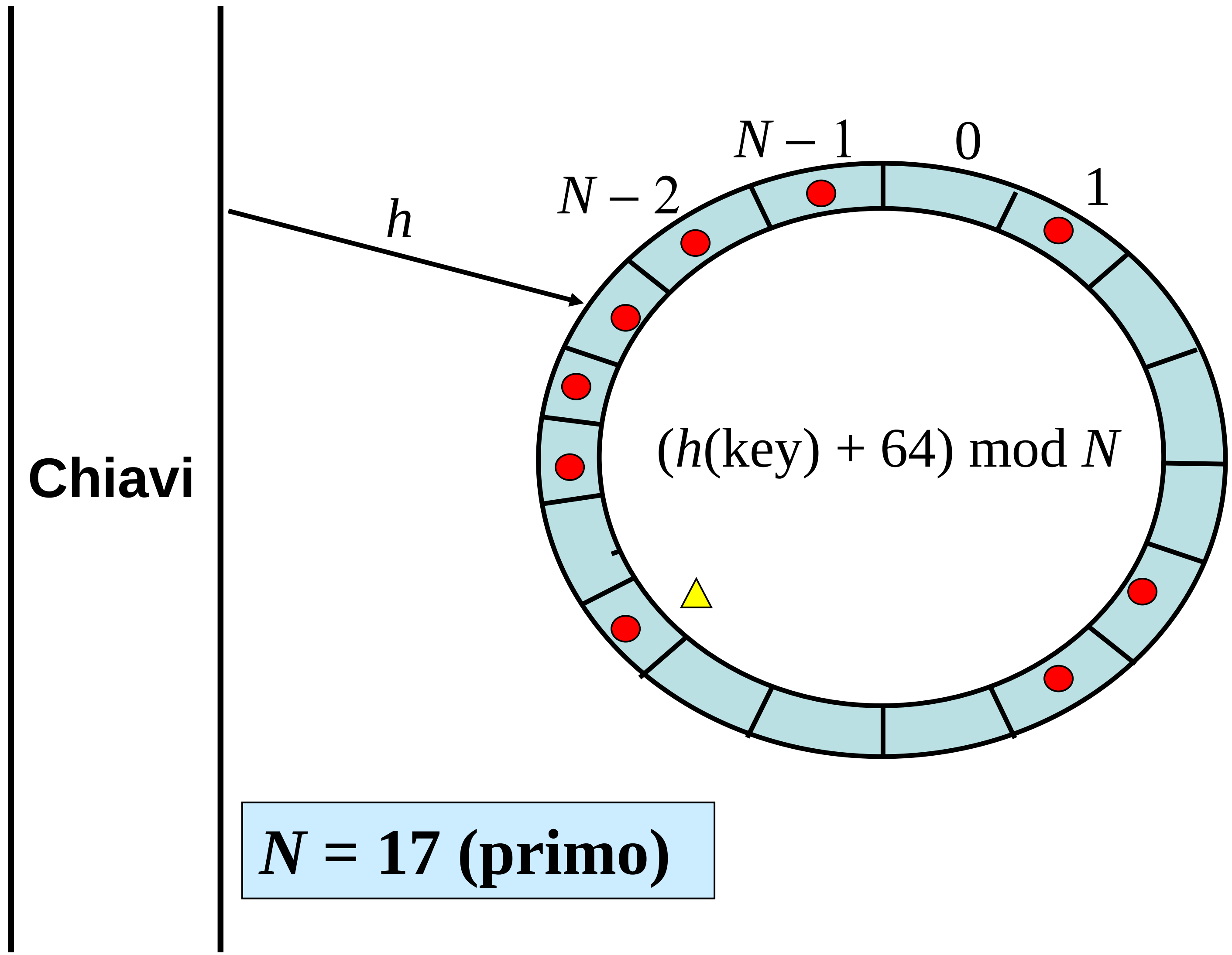


N = 17 (primo)

Chiavi

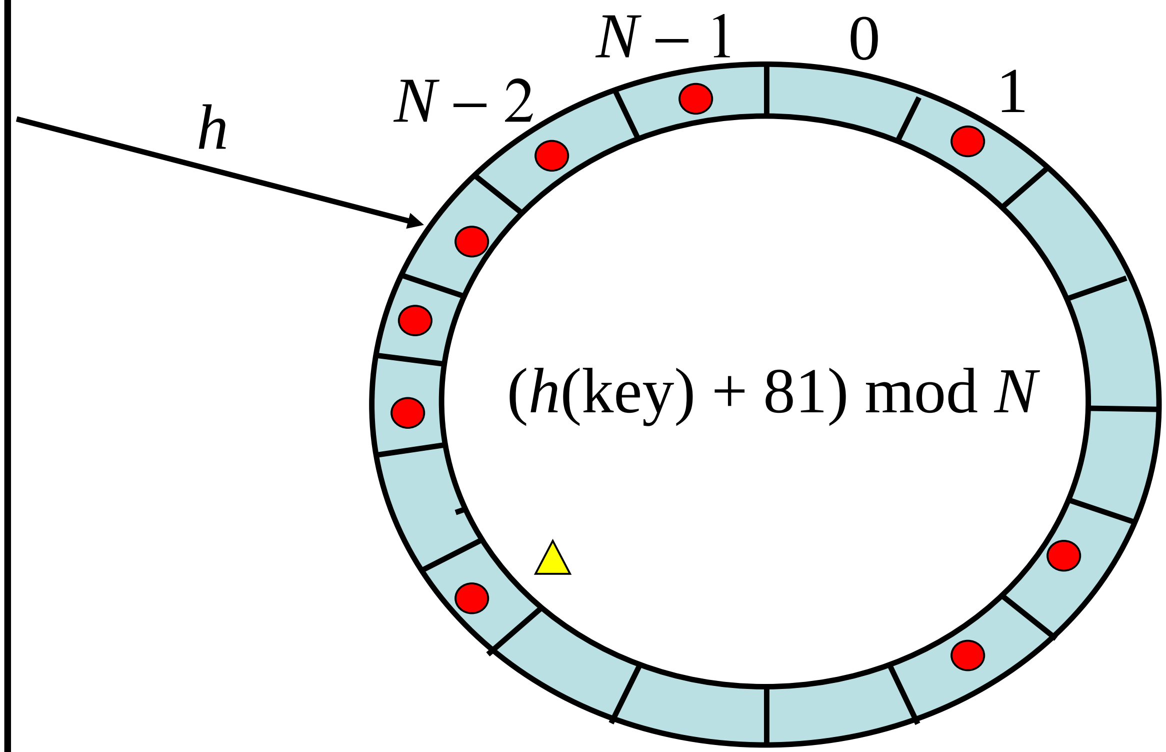


N **= 17 (primo)**



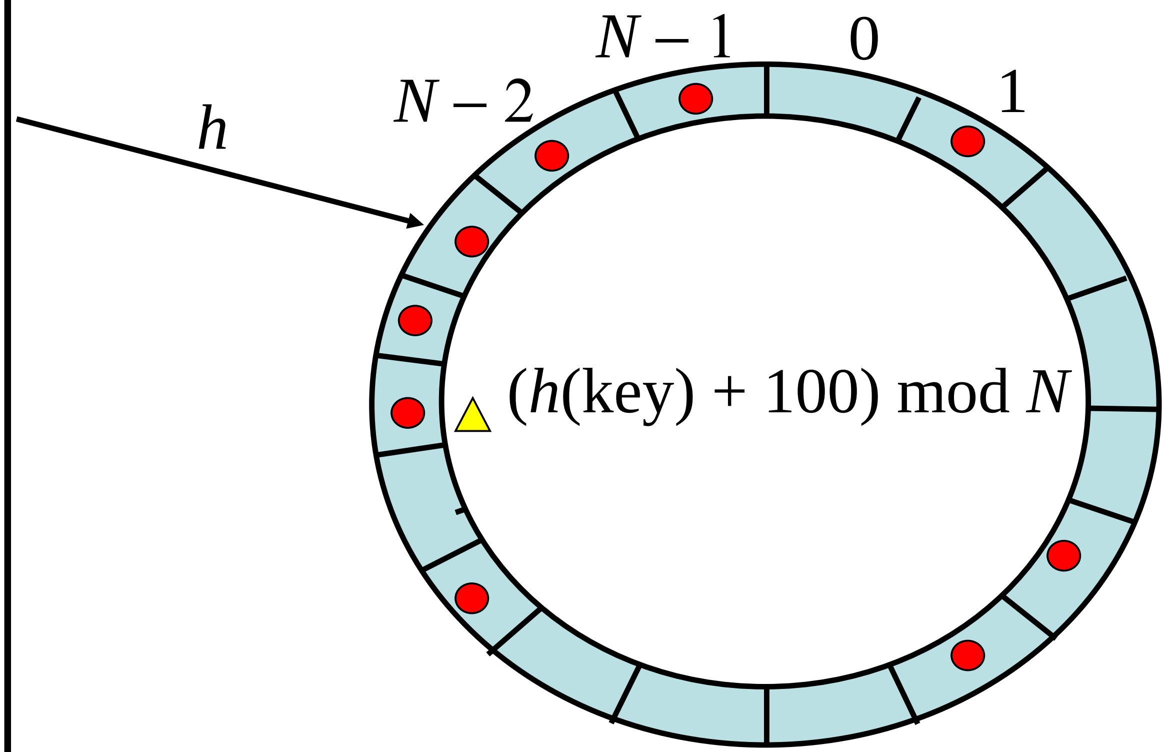
N = **17 (primo)**

Chiavi



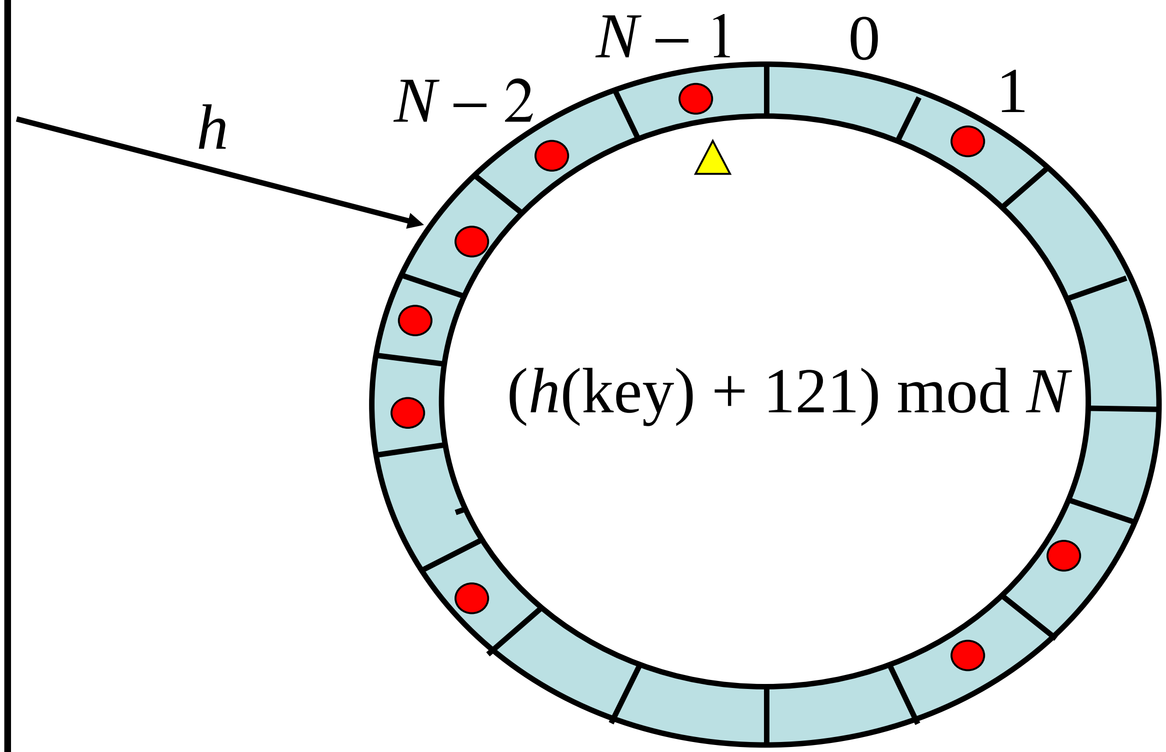
N = 17 (primo)

Chiavi



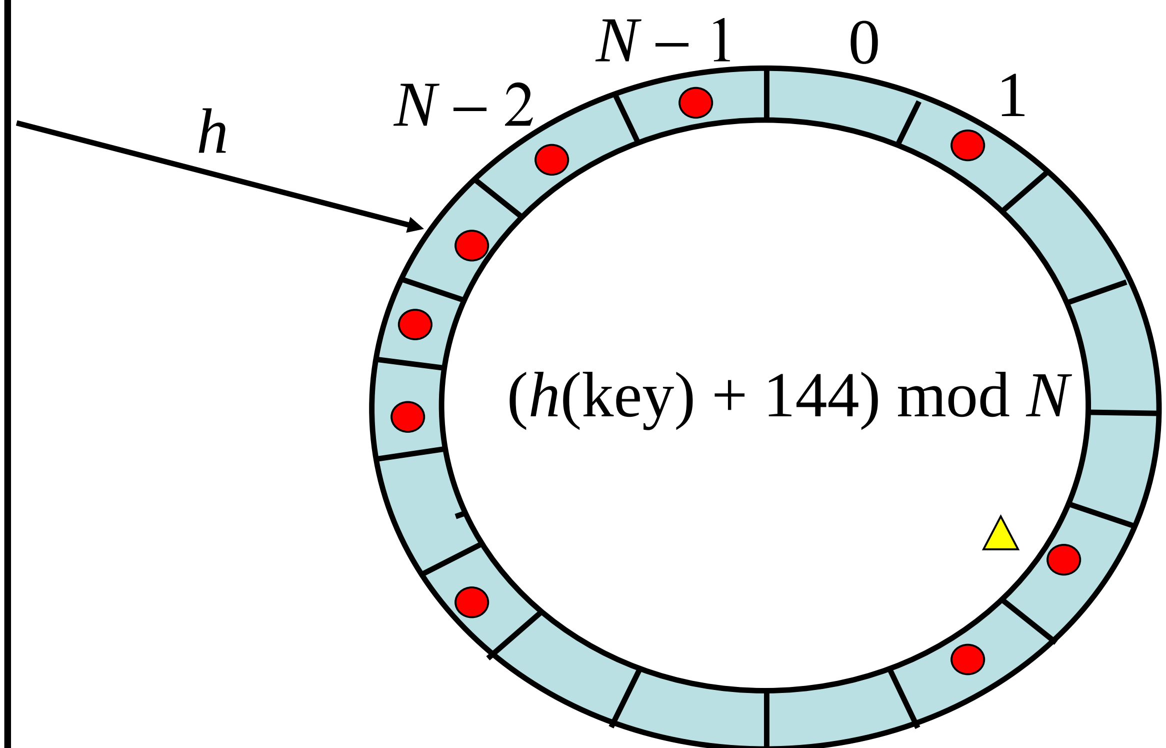
$N = 17$ (primo)

Chiavi



N **= 17 (primo)**

Chiavi



***N* = 17 (primo)**

Ora si possono sondare 9 delle 17 locazioni!

Teorema: Nel sondaggio quadratico con dimensione della tabella numero primo, un nuovo elemento trova sempre spazio se la tabella è vuota almeno per metà.

Dimostrazione:

- Sia la dimensione della tablla, **TSIZE**, un numero primo **> 3**, mostreremo che le prime $\lfloor \mathbf{TSIZE/2} \rfloor$ locazioni alternative (sondaggi) sono tutte distinte.
- Sia $\mathbf{0} \leq \mathbf{i} < \mathbf{j} \leq \lfloor \mathbf{TSIZE/2} \rfloor$ due diversi indici di sondaggio.
- Dimostriamo che

$$(\mathrm{h}(x) + i^2) \bmod \mathrm{TSIZE} \neq (\mathrm{h}(x) + j^2) \bmod \mathrm{TSIZE}$$

- Assumiamo, al contrario, che

$$(\mathrm{h}(x) + i^2) \bmod \mathrm{TSIZE} = (\mathrm{h}(x) + j^2) \bmod \mathrm{TSIZE}$$

- Quindi, deve essere $i^2 \bmod \mathrm{TSIZE} = j^2 \bmod \mathrm{TSIZE}$, cioè
  - $(j^2 - i^2) \bmod \mathrm{TSIZE} = 0$
  - $(j-i)(j+i) \bmod \mathrm{TSIZE} = 0$
- …

Teorema: Se si usa il sondaggio quadratico, e la dimensione della tabella è un numero primo, allora: un elemento nuovo può sempre essere inserito se la tabella è vuota almeno per metà.

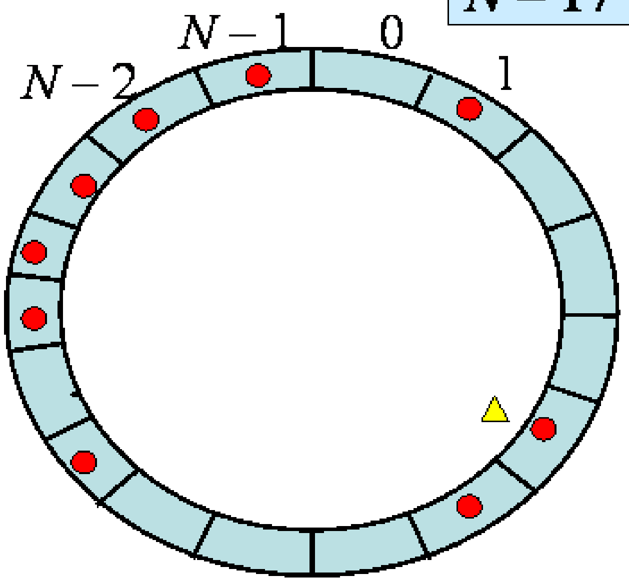
Dimostrazione:

- Quindi, deve essere i^2 mod TSIZE **=** j^2 mod TSIZE, cioè
  - $(j^2 - i^2)$ mod TSIZE = 0
  - $\boldsymbol{(j-i)(j+i)}$ **mod TSIZE = 0**
- Ma $\mathbf{0} \le \boldsymbol{i} < \boldsymbol{j} \le \lfloor \mathbf{TSIZE/2} \rfloor$, quindi $\boldsymbol{j + i <}$ **TSIZE** (**TSIZE** è dispari)
  - $\boldsymbol{(j-i)(j+i)}$ **mod TSIZE = 0** implica che **TSIZE** divide $\boldsymbol{(j-i)(j+i)}$
- Per ***lemma di Euclide*** se un primo $\boldsymbol{p}$ divide $\boldsymbol{a \cdot b}$ allora $\boldsymbol{p}$ divede $\boldsymbol{a}$ o $\boldsymbol{b}$.
- Poiché **TSIZE** è primo, deve valere che **TSIZE** divide $\boldsymbol{(j-i)}$ o $\boldsymbol{(j+i)}$
- Ma entrambi i casi sono ***impossibili***, poiché $\boldsymbol{0 < (j-i)}$ **,** $\boldsymbol{(j+i)} <$ **TSIZE**, quindi **TSIZE** non può dividere nessuno dei due numeri.
- Concludiamo, quindi, che $\boldsymbol{i^2}$ **mod TSIZE** $\boldsymbol{\neq}$ $\boldsymbol{j^2}$ **mod TSIZE**.

Teorema: Se si usa il sondaggio quadratico, e la dimensione della tabella è un numero primo, allora un elemento nuovo può sempre essere inserito se la tabella è vuota almeno per metà.

Applicazione: Il sondaggio visita solo 9 delle 17 posizioni, ma se la tabella è metà vuota, non tutte le 9 posizioni possono essere occupate, quindi possiamo inserire un nuovo elemento in una di esse.

$N = 17$



Esempio di sondaggio quadratico

Clustering secondario: **gli elementi con lo *stesso valore primario di hash*, genereranno la stessa sequenza di sondaggi:**

- **in pratica non costituisce un grosso problema**

Sia data una ***tabella hash*** con ***fattore di carico*** pari a $\lambda = m/n < 1$ (m numero di chiavi inserite, n dimensione della tabella), allora:

- Il ***numero medio di sondaggi*** per una ricerca **senza successo** è al massimo:

$$\frac{1}{1-\lambda}$$

Il ***numero medio di sondaggi*** per una **ricerca con successo** è al massimo:

$$\frac{1}{\lambda} \cdot \ln\left(\frac{1}{1-\lambda}\right)$$

Sondaggio casuale (random)

- usare un ***generatore di numeri pseudo-casuali*** per ottenere una sequenza, ripetibile nella stessa sessione di esecuzione, per identificare lo ***i***-esimo sondaggio.

$$h(k,i) = (h(k) + r_i) \bmod N$$

- dove r_i è l'***i***-esimo valore in una ***permutazione random*** dei numeri **1…(N-1)**.
- più semplice da implementare del sondaggio quadratico ma soffre degli stessi problemi: ***clustering secondario***.
- infatti la sequenza di sondaggio ***dipende solo dalla*** dal valore di hash primario **h(k)**.

Tutte le soluzioni viste permettono di generare **N** differenti sequenze di sondaggio (una per ogni locazione della tabella).

Doppio Hashing

- usare una ***seconda funzione hash*** per trovare una locazione alternativa

$$h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod N$$

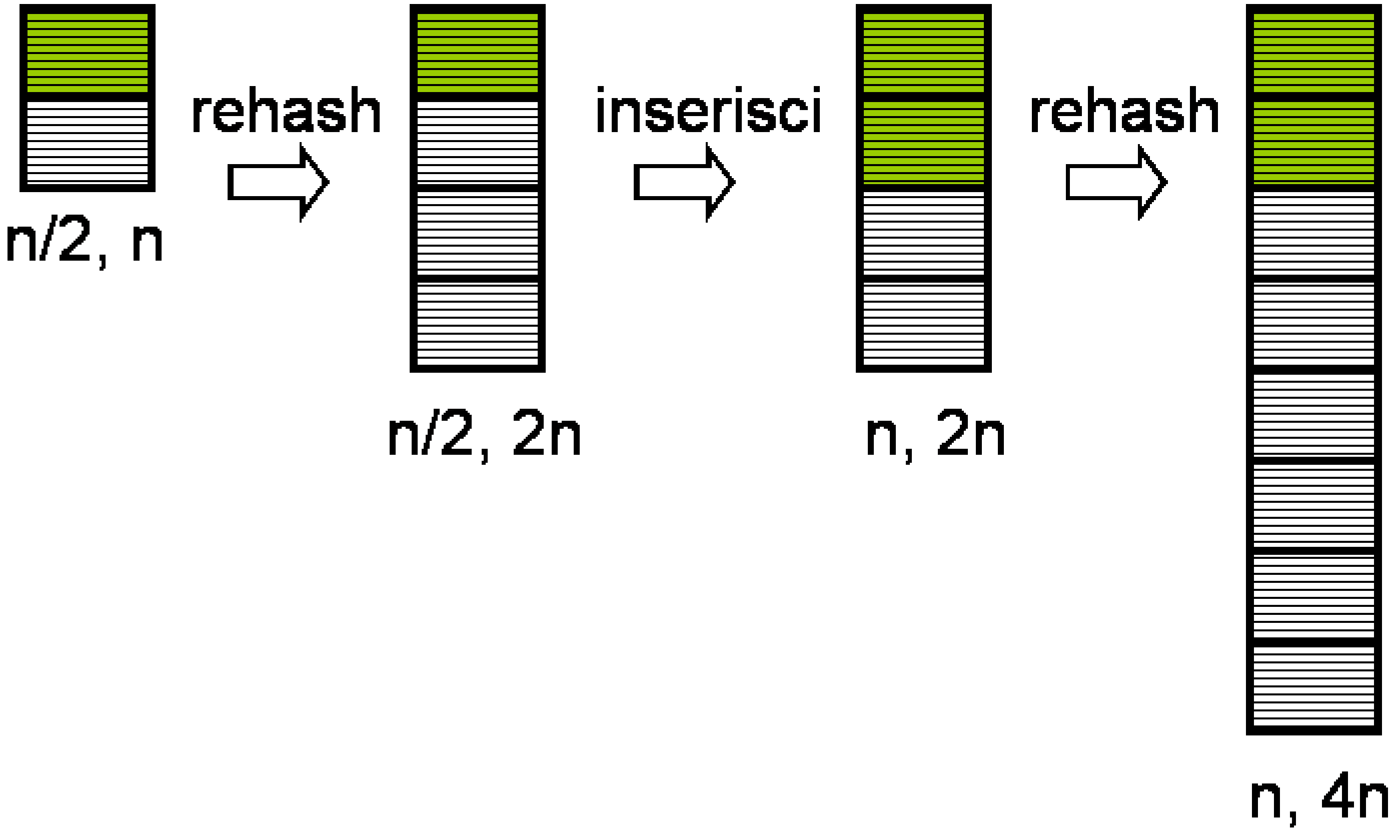
- è la soluzione migliore in quanto permette di generare **N^2** ***differenti sequenze di sondaggio*** (la sequenza, infatti, dipende in *due modi differenti*, e tra loro indipendenti, dalla chiave ***k***).
- ***$h_2(k)$*** deve essere ***relativamente primo*** con **N**, altrimenti non tutta la tabella potrebbe venire ispezionata.
- es. 1: **N** primo, $h_1(k) = k \bmod N$ e $h_2(k) = 1 + (k \bmod N-1)$.
- es. 2: **$N = 2^p$** e $h_2(k)$ produce sempre **numeri dispari**.

Cancellazioni

- le *cancellazioni* con l'*indirizzamento aperto* sono *difficili*
- la ***cencellazone "lazy"*** è il metodo preferibile
  - si marcano gli elementi cancellati come non allocati, in modo che possano essere riutilizzati
  - quando il numero di record cancellati raggiunge qualche livello di soglia (predeterminato), si può effettuare un'operazione di "***garbage collection***" per riottenere le locazioni cancellate (***perché è utile?***)

Nel caso di **indirizzamento aperto**

- Tabella troppo piena
  - Tempo di esecuzione troppo lungo
  - Inserimento può fallire (quando?)
- La dimensione **DEVE** essere scelta in anticipo
  - Ma non sappiamo il numero di elementi
- ***Rehashing***
  - Costruire una nuova tabella di dimensione circa il doppio
  - Rimappare (via Hash) tutti gli elementi nella nuova tabella (*preché?*)



Reashing: per **n/2** elementi, la dimensione della tabella è **n**. Il costo addizionale per ogni inserimento è **O(n)/(n/2) = O(1)**, dove **O(n)** è il costo per rimappare nella nuova tabella tutte le chiavi presenti in tabella.

- Quando applicare l'***estensione*** della ***tabella hash***?
  - La tabella è piena per metà
  - L'inserimento fallisce
  - λ raggiunge una soglia ***t*** predeterminata
- Rimensione della nuova tabella
  - circa 2 volte più grande o, più in generale,
  - circa ***x*** volte più grande: con ***x* =** 1.5, 2, 2.5, 3

Rehashing: esempio

Si supponga uno schiema di hashing in cui

– Valore iniziale di ***n* = TSIZE = 7**

– Funzoine di hashing: ***h*(*x*)** = *x* mod **TSIZE**

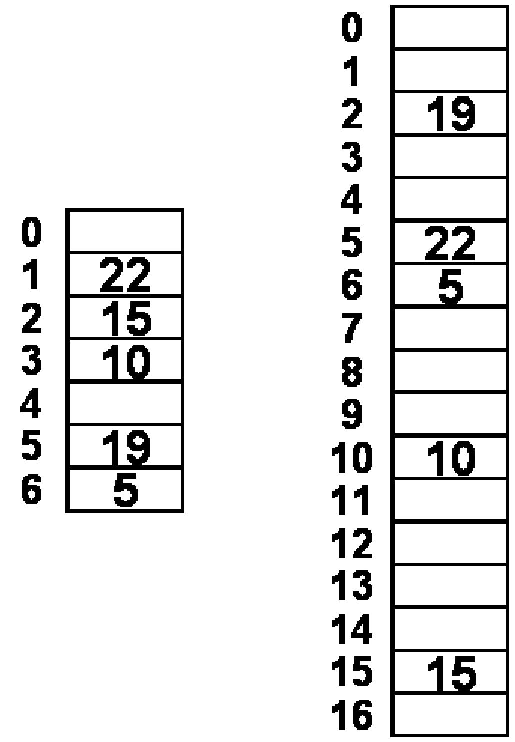
– Risoluzione delle collisioni con sondaggio lineare

– Soglia di rehashing **m/n** = 66% (con **m** numero di chiavi presenti)

– Inserimento delle chiavi: 19, 10, 5, 22, 15

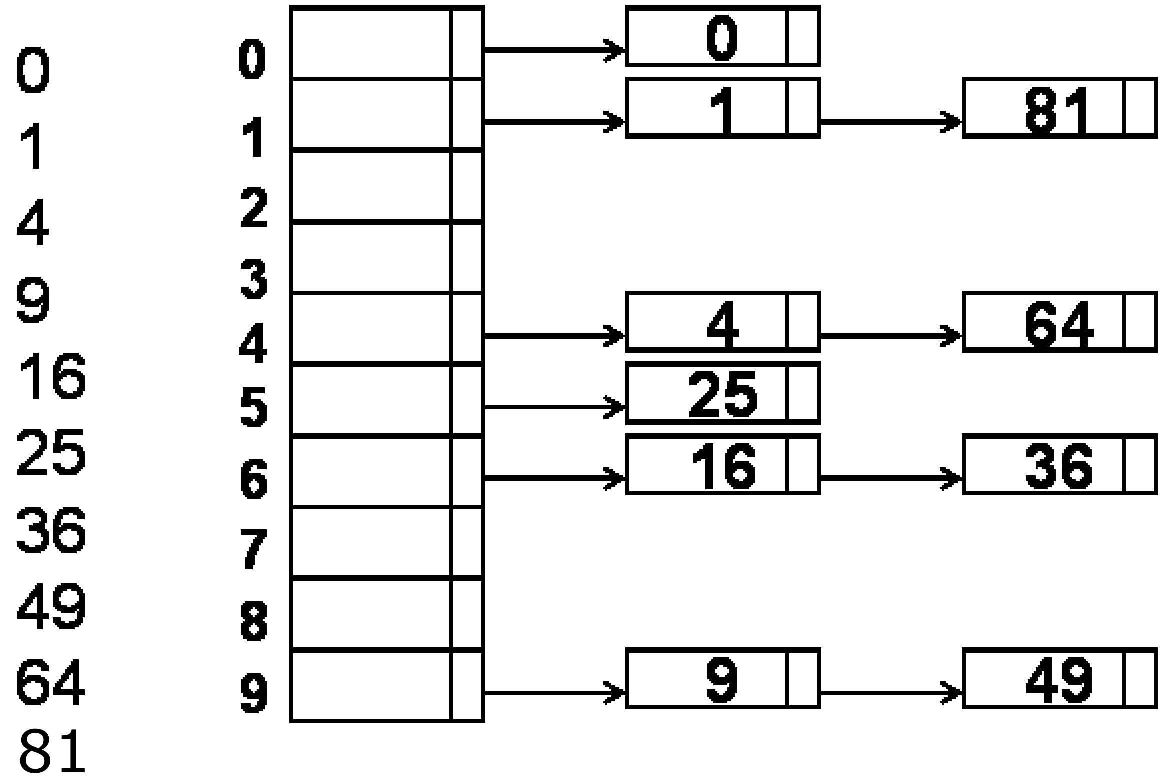
– Rehashing con nuovo valore **TSIZE = 17**

La figura a destra illustra il risultato della operazione di **rehashing**.



Esempio di Rehashing.

- É necessaria una ***Struttura Dati Secondaria***
  - Lista
  - Albero
  - Una seconda Tabella Hash
- Se ci aspettiamo poche collisioni, possiamo utilizzare un lista
  - Semplice
  - Poco lavoro aggiuntivo necessario
  - Inserimenti nelle catene in tempo costante
- **Vantaggi**: permette ***maggiore utilizzo*** della tabella e non necessita di rehashing.

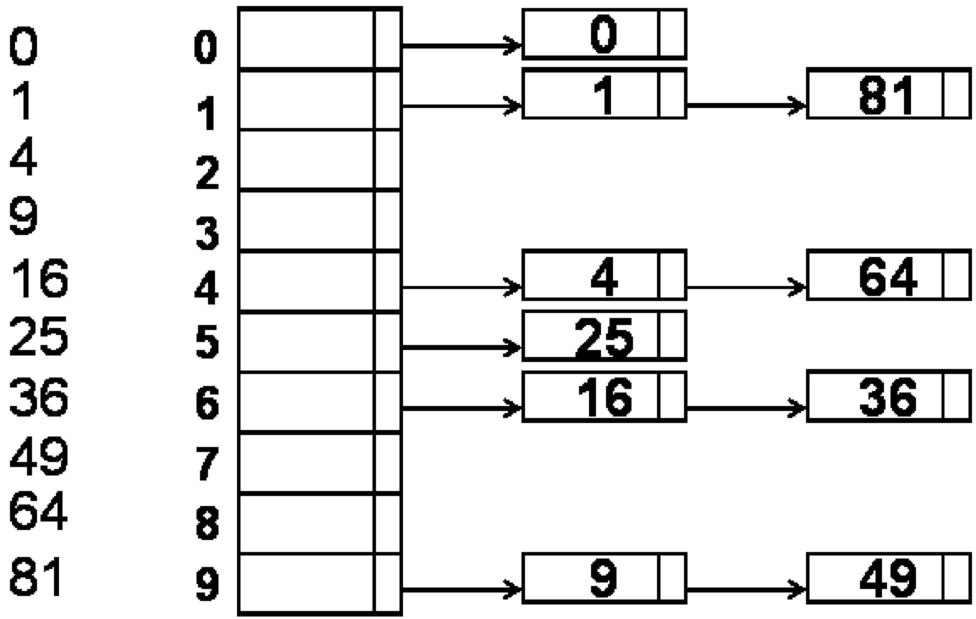


Risultato dell'inserimento delle chiavi a sinistra con funzione di hashing ***hash(x) = x mod 10***.

Lunghezza media lista di concatenamento

l_i = lungezza lista *i*-esima

$$\frac{\sum_{i=0}^{TSIZE-1} l_i}{TSIZE} = \frac{4*2+2*1+4*0}{10} = 1$$



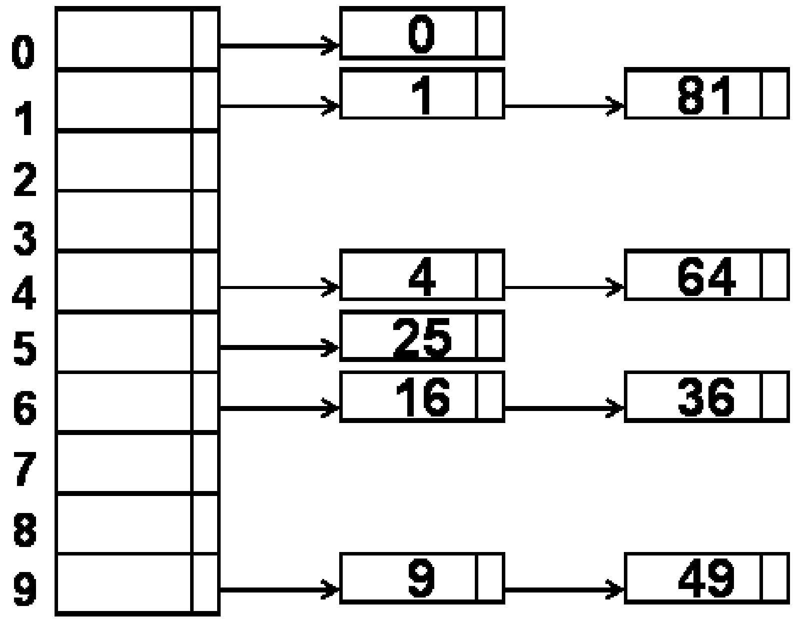
Calcolo della lunghezza media di una lista di collisione.

l_i = lungezza lista *i*-esima

$$\frac{\sum_{i=0}^{TSIZE-1} l_i}{TSIZE} = \frac{4 * 0 + 2 * 1 + 4 * 2}{10} = 1$$

numero di elementi inseriti

0
1
4
9
16
25
36
49
64
81



Calcolo della lunghezza media di una lista di collisione.

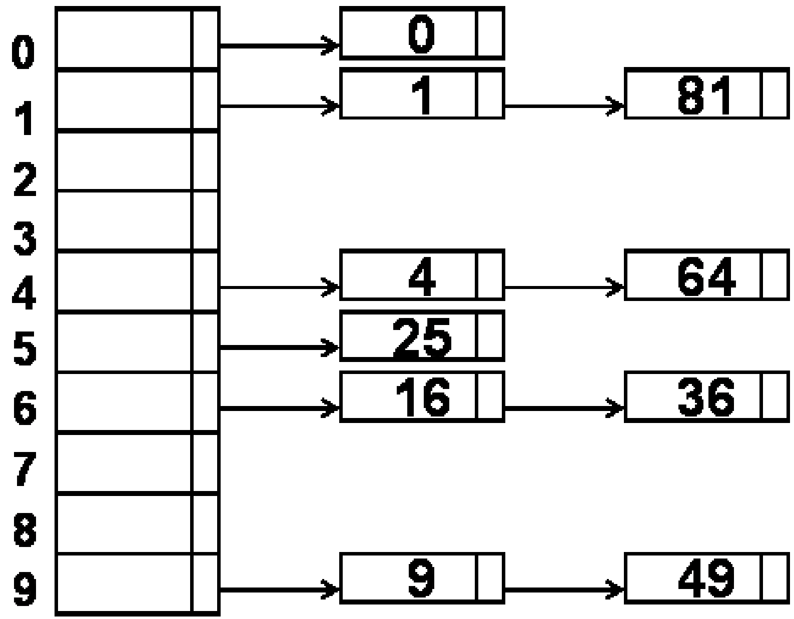
Lunghezza media lista di concatenamento

l_i = lungezza lista *i*-esima

$$\frac{\sum_{i=0}^{TSIZE-1} l_i}{TSIZE} = \frac{4*0+2*1+4*2}{10} = 1$$

Fattore di carico λ

0
1
4
9
16
25
36
49
64
81

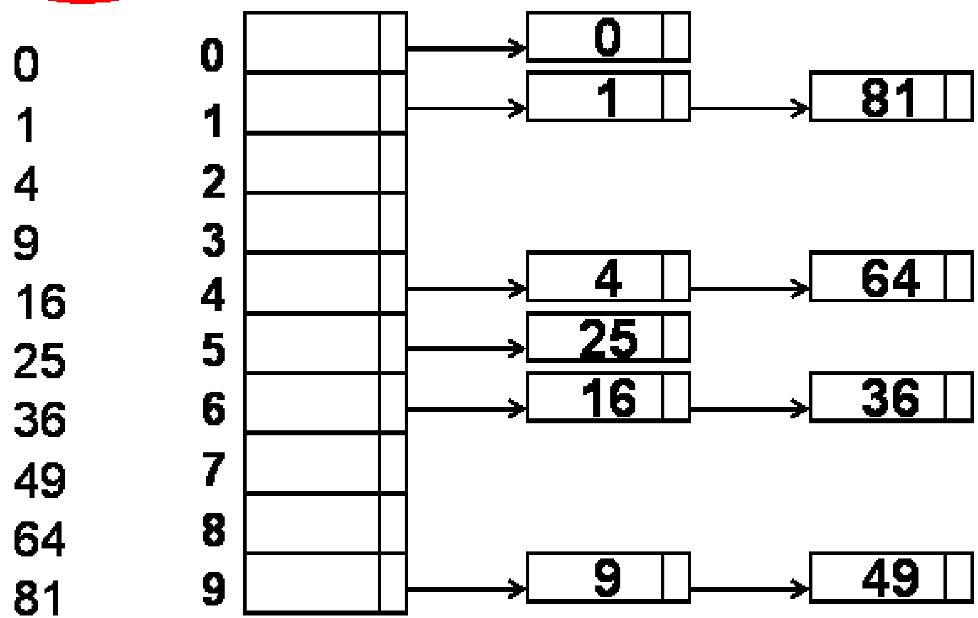


Calcolo della lunghezza media di una lista di collisione.

Il *fattore di carico* λ è la lunghezza media delle liste

$$\frac{\sum_{i=0}^{TSIZE-1} l_i}{TSIZE} = \frac{4*0+2*1+4*2}{10} = 1$$

Fattore di carico λ



La ***lunghezza media di una lista di collisione*** equivale al ***fattore di carico***

- λ **= m/n** : numero di *elementi inseriti* (**m**) *diviso* per la *dimensione della tabella* (**n**).
- λ : rappresenta la *lunghezza media di una lista*
- Tempo medio di esecuzione della ricerca
  - hashing (**Θ(1)**) **+** attraversamento della lista
  - elemento non trovato: λ
  - elemento trovato: **1 + λ/2**
    - Almeno 1 link viene ispezionato
    - La lunghezza media di una lista è λ
    - Ci si aspetta *in media* di trovare un elemento presente dopo aver attraversato *metà della lista*.

- Regola euristica: mantenere $\lambda \approx \mathbf{1}$.

Data una ***tabella hash*** con ***fattore di carico*** $\lambda = m/n$ (***m*** numero di chiavi inserite, ***n*** dimensione della tabella), allora:

- il tempo medio impiegato per una ***ricerca senza successo*** è al massimo:

$$\Theta(1 + \lambda)$$

- il tempo medio impiegato per una ***ricerca con successo*** è al massimo:

$$\Theta(1 + \lambda)$$

- Vantaggi del concatenamento a liste
  - con una buona funziona hash le *liste puntate risultano corte*
  - il *clustering* non è un problema — gli elementi con chiavi differenti stanno su catene differenti
  - la *dimensione della tabella è meno rilevante*
  - le *cancellazioni* sono *facili* ed *efficienti*
  - le *catene* potrebbero anche essere implementate come *alberi binari di ricerca* o loro varianti più efficienti

- *Dimensione della tabella*: *Numero Primo*
- *Funzione Hash*: basata su *operazioni di modulo*
- *Gestione delle Collisioni*
  - Indirizzamento chiuso (concatenamento)
  - Indirizzamento aperto (sondaggi)
    - Sondaggio lineare, quadratico, doppio
- λ: *fattore di carico* (numero di elementi diviso per la dimensione della tabella).
  - Indirizzamento chiuso: $\lambda \approx \mathbf{1}$
  - Indirizzamento aperto: $\lambda \approx \mathbf{0.5}$
- *Rehashing*