

# Computação Distribuída

DHT Chord

Vladimir Rocha (Vladi)

**CMCC - Universidade Federal do ABC**

# Disclaimer

- Estes slides foram baseados no tutorial do professor Amir Paybereah  
<https://payberah.github.io/files/download/p2p/chord.pdf>

# Tabela de Hash Distribuída

- Denominada DHT (Distributed Hash Table)
- Sistema P2P estruturado
- Criada em 2001 para encontrar um recurso de forma eficiente:  $O(\log N)$
- Provê escalabilidade de tamanho e particionamento (sharding).
- Evita o ponto único de falha (SPoF)
- Usada em:



NETFLIX



tinder



# Conceito

- Considere uma tabela de hash normal

Key	Value
Fatemeh	Stockholm
Sarunas	Lausanne
Tallat	Islamabad
Cosmin	Bucharest
Seif	Stockholm
Amir	Tehran

# Conceito

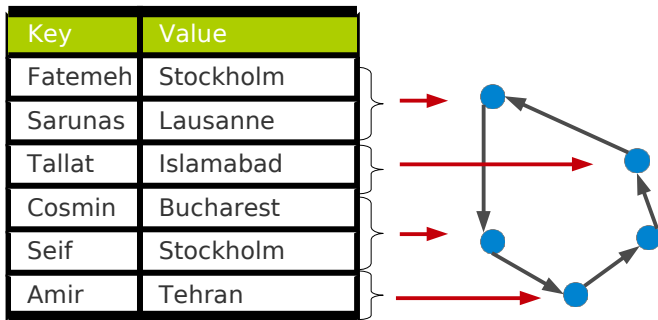
- Considere uma tabela de hash normal. Se ela estiver só em um servidor, esse se transforma no chamado “ponto único de falha” (SPOF – Single Point of Failure) caso morra.

Key	Value
Fatemeh	Stockholm
Sarunas	Lausanne
Tallat	Islamabad
Cosmin	Bucharest
Seif	Stockholm
Amir	Tehran



# Conceito

- Então, que tal distribuir a tabela de hash em vários nós?  
Ou seja, teremos uma tabela de hash **distribuída**



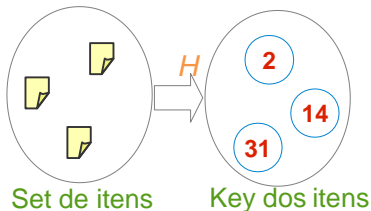
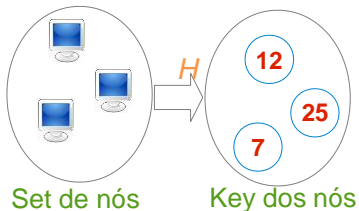
# Chord (DHT estruturado em anel)

- Visão geral da construção e utilização

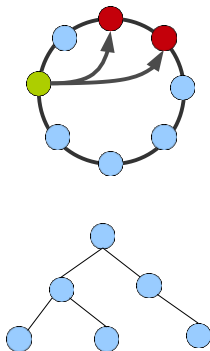
# Chord (DHT estruturado em anel)

## Visão geral da construção e utilização

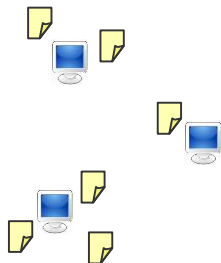
- 1 Definir o espaço comum de valores para os nós e itens



- 2 Conectar os nós usando um número pequeno de links (minimizar hops)



- 3 Definir a estratégia para designar itens aos nós

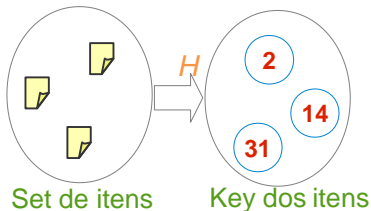
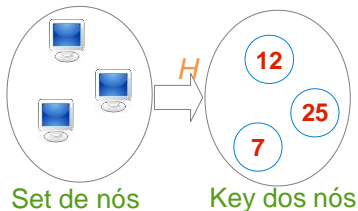




# Chord (DHT estruturado em anel)

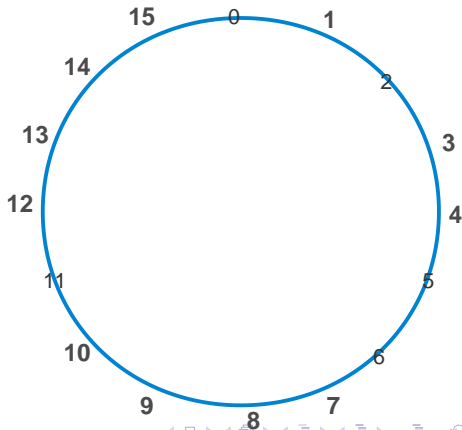
- Visão geral da construção e utilização

- 1 Definir o espaço comum de valores para os nós e itens



# Construção da DHT Chord

- Usar um espaço de nomes (**espaço de identificadores**) de  $\{0, 1, 2 \dots N-1\}$ 
  - Esse espaço de nomes formará um **anel lógico** módulo  $N$



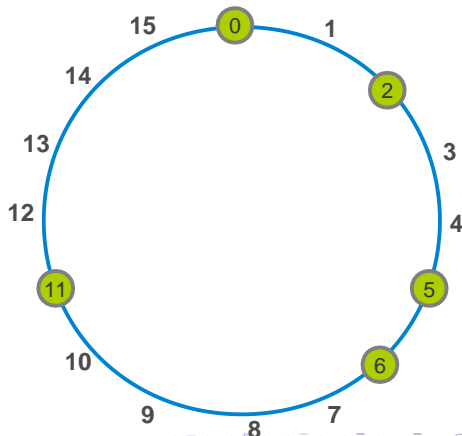
## Construção da DHT Chord

- Usar um espaço de nomes (espaço de identificadores) de  $\{0, 1, 2 \dots N-1\}$ 
  - Esse espaço de nomes formará um anel lógico módulo  $N$
- O nó terá um nº aleatório obtido de uma função de **hash H**

Exemplo:

Espaço  $N=16 \{0, \dots, 15\}$

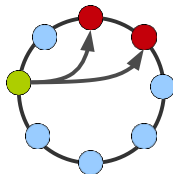
Cinco nós a, b, c, d, e

$$H(a) = 6$$
$$H(b) = 5$$
$$H(c) = 0$$
$$H(d) = 11$$
$$H(e) = 2$$




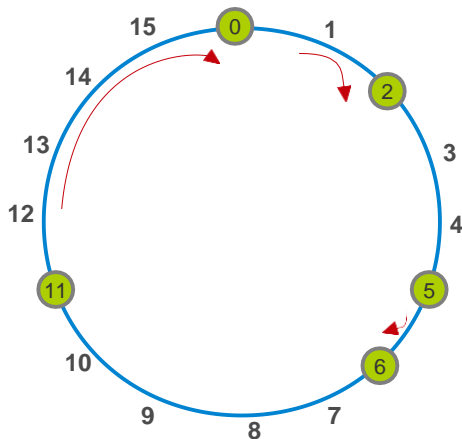
## Chord (DHT estruturado em anel)

- 2 Conectar os nós usando um número pequeno de links (minimizar hops)




## Construção da DHT Chord

- O **sucessor** de um identificador é o primeiro nó indo na **direção do relógio** começando no identificador



## Construção da DHT Chord

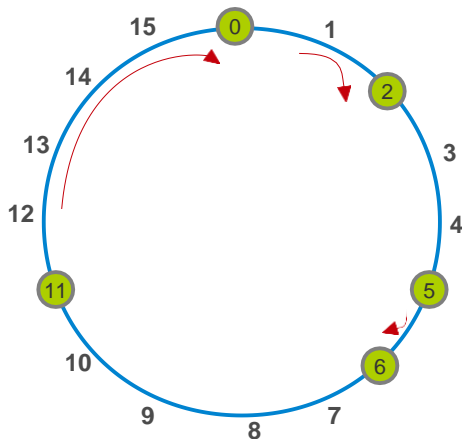
- O sucessor de um identificador é o primeiro nó indo na direção do relógio começando no identificador
  - $\text{succ}(x)$  é o primeiro nó no anel com id **maior ou igual** a  $x$
- 

Exemplo:

$$\text{succ}(12) = \theta$$

$$\text{succ}(1) = 2$$

$$\text{succ}(6) = 6$$



## Construção da DHT Chord

- Apontar cada nó  $p$  ao seu sucessor  $\text{succ}(p+1)$ .

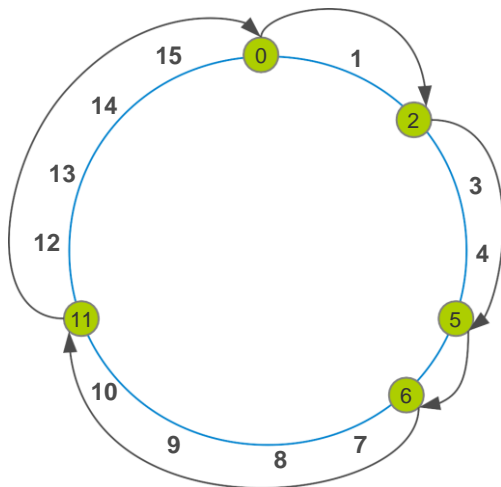
Sucessor do nó 0 é  $\text{succ}(1) = 2$

Sucessor do nó 2 é  $\text{succ}(3) = 5$

Sucessor do nó 5 é  $\text{succ}(6) = 6$

Sucessor do nó 6 é  $\text{succ}(7) = 11$

Sucessor do nó 11 é  $\text{succ}(12) = \emptyset$

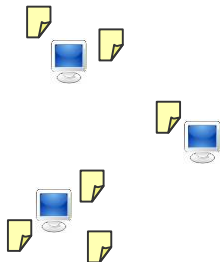


**Invariante:** um nó sempre deverá ter a conexão com seu sucessor!



# Chord (DHT estruturado em anel)

- 3 Definir a estratégia para designar itens aos nós



# Estratégia para armazenar itens

- Onde armazenar os dados?
  - Utilizando uma função de hash  $H$
  - A mesma da construção
- Cada item  $\langle \text{key}, \text{value} \rangle$  obterá o identificador  $H(\text{key}) = k$

$H(\text{'Fateme h'}) = 12$

$H(\text{'Cosmin'}) = 2$

$H(\text{'Seif'}) = 9$

$H(\text{'Sarunas'}) = 14$

$H(\text{'Tallat'}) = 4$



# Estratégia para armazenar itens

- Onde armazenar os dados?
  - Utilizando uma função de hash  $H$
  - A mesma da construção
- Cada item  $\langle \text{key}, \text{value} \rangle$  obterá o identificador  $H(\text{key}) = k$

$H(\text{'Fateme h'}) = 12$

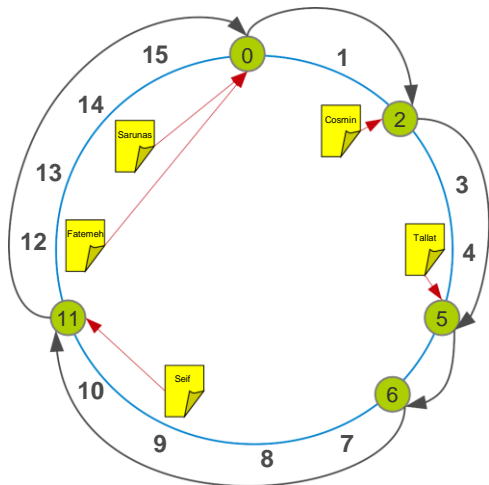
$H(\text{'Cosmin'}) = 2$

$H(\text{'Seif'}) = 9$

$H(\text{'Sarunas'}) = 14$

$H(\text{'Tallat'}) = 4$

- Armazene cada item no **sucessor**





## Estratégia para armazenar itens: *consistent hashing*

- Chord utiliza *consistent hashing* para designar chaves aos nós
- **Consistent Hashing**: técnica de hashing que permite que somente  $O(m/N)$  pares de chave-valor sejam transferidas na adição ou remoção de nós
  - $m$  é o número de chaves armazenadas.
  - $N$  é o número de nós.

### Para pensar ...

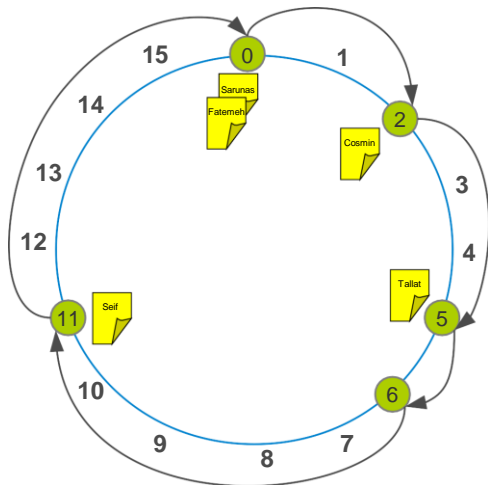
Em vez de armazenar o item no sucessor (como mostrado no slide anterior) qual seria o problema de designar a chave ao nó utilizando a função módulo e não usando o consistent hashing?

Ou seja, armazene o item no nó:  $H(\text{key}) \% N$

# Como realizar uma busca?

# Como realizar uma busca?

- Para buscar uma chave  $k$ 
  - Pergunte para **qualquer** nó do anel
  - Calcule  $H(k)$
  - Siga os apontadores dos succ até  $k$  ser encontrado





# Como realizar uma busca?

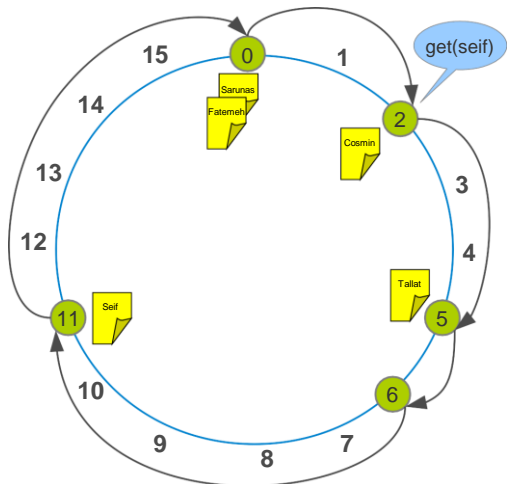
- Para buscar uma chave  $k$ 
  - Pergunte para **qualquer** nó do anel
  - Calcule  $H(k)$
  - Siga os apontadores dos succ até  $k$  ser encontrado

Exemplo, procurando "Seif" em nó 2

$H('Seif') = 9$

Percorrer nós 2, 5, 6, 11

Devolver "Stockholm" ao iniciador



Key	Value
Seif	Stockholm

# Algoritmos

```
// Estrutura do nó n
```

```
No {
```

```
    int meuld = n
```

```
    No predecessor = null
```

```
    No sucessor = null
```

```
}
```

# Algoritmos

- $(a, b]$  é o segmento do anel que não inclui  $a$  mas inclui  $b$
- $n.foo(\dots)$  é uma chamada RPC ao nó  $n$  pelo método  $foo$
- $n.bar$  é uma chamada RPC ao nó  $n$  obtendo a variável  $bar$

// pergunta ao nó  $n$  encontrar o successor do  $id$  procurado

```
procedure  $n.findSuccessor(id, No\ ini)$  {  
  if ( $predecessor \neq nil$  and  $id \in (predecessor, n]$ ) then  $ini.response(id, n)$   
  else if ( $id \in (n, successor]$ ) then  $ini.response(id, successor)$   
  else // encaminha a pergunta pelo anel  
     $successor.findSucessor(id, ini)$   
}
```

# Algoritmos

```
procedure n.put(id, value) {
```

```
}
```

```
procedure n.get(id) {
```

```
}
```

# Algoritmos

```
procedure n.put(id, value) {  
    op = put  
    findSuccessor(id)  
}
```

```
procedure n.get(id) {  
    op = get  
    findSuccessor(id)  
}
```

```
procedure n.response(id, node) {  
    if (op == put) node.store(id,value)  
    else if (op == get) node.retrieve(id)  
}
```

- Ambos métodos put e get utilizam a chamada a findSucessor

# Algoritmos

// Estrutura do nó n (inicial)

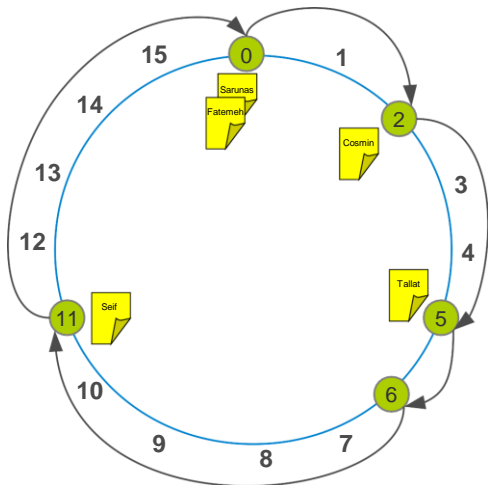
```
No {  
    int meuld = n  
    No predecessor = null  
    No sucessor = null  
}
```

// Modificando a estrutura do nó n (agora considerando store/retrieve)

```
No {  
    int meuld = n  
    No predecessor = null  
    No sucessor = null  
    KV chaves-valor = null  
}
```

# Buscas mais rápidas

- Se somente o apontador ao sucessor for usado:  $O(N)$  para  $N$  nós
- Melhor usar "atalhos" (*shortcuts*)



# Buscas mais rápidas

- Fingertable

Aponta para  $\text{succ}(p+1)$

Aponta para  $\text{succ}(p+2)$

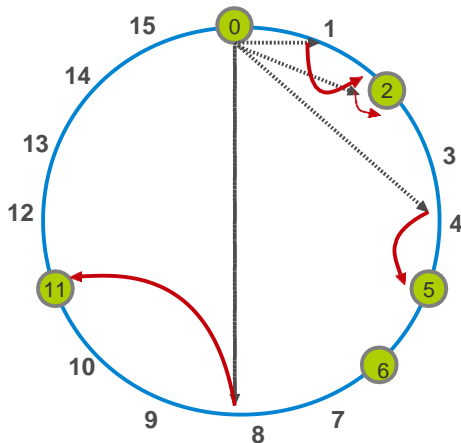
Aponta para  $\text{succ}(p+4)$

Aponta para  $\text{succ}(p+8)$

...

Aponta para  $\text{succ}(p+2^{M-1})$

FingerTable nó 0	
i	Succ ( $n + 2^{i-1}$ )
1	$0 + 2^0 = 1 \rightarrow 2$
2	$0 + 2^1 = 2 \rightarrow 2$
3	$0 + 2^2 = 4 \rightarrow 5$
4	$0 + 2^3 = 8 \rightarrow 11$







# Buscas mais rápidas


// pergunta ao nó n encontrar o successor do id

```
procedure n.findSuccessor(id, No ini) {  
  if (predecessor  $\neq$  nil and id  $\in$  (predecessor, n]) then ini.response(id, n)  
  else if (id  $\in$  (n, successor]) then ini.response(id, sucessor)  
  else // encaminha a pergunta pelo anel  
    sucessor.findSucessor(id,ini)  
}
```

# Buscas mais rápidas

// pergunta ao nó n encontrar o successor do id

```
procedure n.findSuccessor(id, No ini) {  
  if (predecessor  $\neq$  nil and id  $\in$  (predecessor, n]) then ini.response(id, n)  
  else if (id  $\in$  (n, successor]) then ini.response(id, sucessor)  
  else // encaminha a pergunta pelo anel  
    sucessor.findSucessor(id,ini)  
}
```



closestPrecedingNode(id)

# Buscas mais rápidas

// pergunta ao nó n encontrar o successor do id

```
procedure n.findSuccessor(id, No ini) {  
  if (predecessor  $\neq$  nil and id  $\in$  (predecessor, n]) then ini.response(id, n)  
  else if (id  $\in$  (n, successor]) then ini.response(id, sucessor)  
  else // encaminha a pergunta pelo anel  
    m = closestPrecedingNode(id)  
    m.findSucessor(id,ini)  
}
```

// procura localmente pelo nó próximo ao id

```
procedure closestPrecedingNode(id) {  
  for (i = M to 1 do) {  
    if (FT[i]  $\in$  (n, id)) then  
      return FT[i]  
  }  
  return n // meu id  
}
```

# Algoritmos

// Estrutura do nó n (considerando store)

```
No {  
    int meuld = n  
    No predecessor = null  
    No sucessor = null  
    KV chaves-valor = null  
}
```

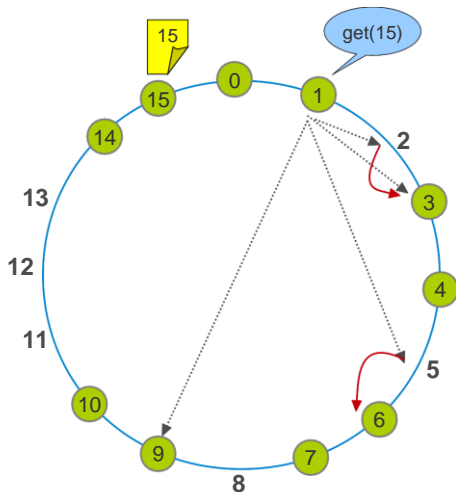
// Modificando a estrutura do nó n (agora considerando FT)

```
No {  
    int meuld = n  
    No predecessor = null  
    No sucessor = null  
    KV chaves-valor = null  
    No[] FT = null  
}
```

# Buscas mais rápidas

```
procedure n.findSuccessor(id, No ini) {  
  if (predecessor ≠ nil and id ∈ (predecessor, n]) then ini.response(id, n)  
  else if (id ∈ (n, successor]) then ini.response(id, successor)  
  else // encaminha a pergunta pelo anel  
    m = closestPrecedingNode(id)  
    m.findSucessor(id, ini)  
}
```

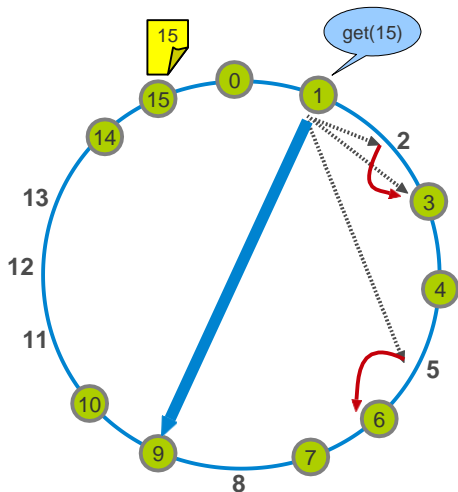
FingerTable nó 1	
i	Succ ( $n + 2^{i-1}$ )
1	$1 + 2^0 = 2 \rightarrow 3$
2	$1 + 2^1 = 3 \rightarrow 3$
3	$1 + 2^2 = 5 \rightarrow 6$
4	$1 + 2^3 = 9 \rightarrow 9$



# Buscas mais rápidas

```
procedure n.findSuccessor(id, No ini) {  
  if (predecessor ≠ nil and id ∈ (predecessor, n]) then ini.response(id, n)  
  else if (id ∈ (n, successor]) then ini.response(id, successor)  
  else // encaminha a pergunta pelo anel  
    m = closestPrecedingNode(id)  
    m.findSucessor(id, ini)  
}
```

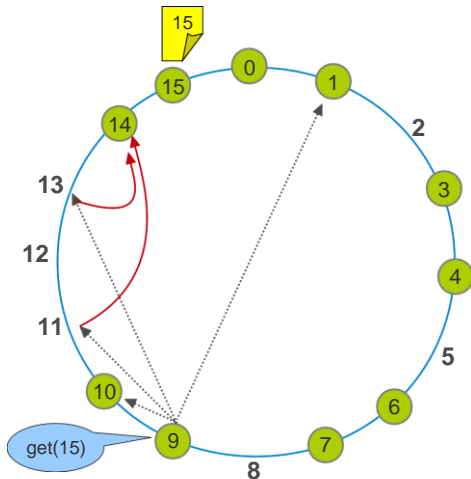
FingerTable nó 1	
i	Succ ( $n + 2^{i-1}$ )
1	$1 + 2^0 = 2 \rightarrow 3$
2	$1 + 2^1 = 3 \rightarrow 3$
3	$1 + 2^2 = 5 \rightarrow 6$
4	$1 + 2^3 = 9 \rightarrow 9$



# Buscas mais rápidas

```
procedure n.findSuccessor(id, No ini) {  
  if (predecessor ≠ nil and id ∈ (predecessor, n]) then ini.response(id, n)  
  else if (id ∈ (n, successor]) then ini.response(id, successor)  
  else // encaminha a pergunta pelo anel  
    m = closestPrecedingNode(id)  
    m.findSucessor(id, ini)  
}
```

FingerTable nó 9	
i	Succ ( $n + 2^{i-1}$ )
1	$9 + 2^0 = 10 \rightarrow 10$
2	$9 + 2^1 = 11 \rightarrow 14$
3	$9 + 2^2 = 13 \rightarrow 14$
4	$9 + 2^3 = 17 \% 16 \rightarrow 1$

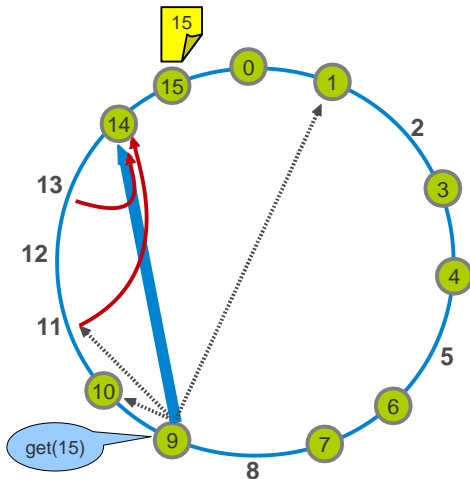




# Buscas mais rápidas

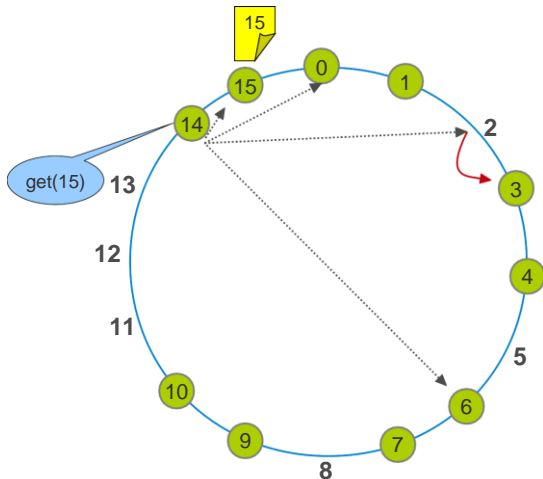
```
procedure n.findSuccessor(id, No ini) {  
  if (predecessor ≠ nil and id ∈ (predecessor, n]) then ini.response(id, n)  
  else if (id ∈ (n, successor]) then ini.response(id, successor)  
  else // encaminha a pergunta pelo anel  
    m = closestPrecedingNode(id)  
    m.findSucessor(id, ini)  
}
```

FingerTable nó 9	
i	Succ ( $n + 2^{i-1}$ )
1	$9 + 2^0 = 10 \rightarrow 10$
2	$9 + 2^1 = 11 \rightarrow 14$
3	$9 + 2^2 = 13 \rightarrow 14$
4	$9 + 2^3 = 17 \% 16 \rightarrow 1$



# Buscas mais rápidas

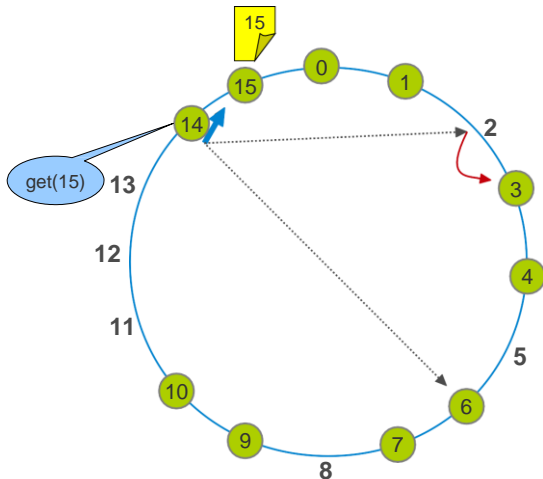
```
procedure n.findSuccessor(id, No ini) {  
  if (predecessor ≠ nil and id ∈ (predecessor, n]) then ini.response(id, n)  
  else if (id ∈ (n, successor]) then ini.response(id, successor)  
  else // encaminha a pergunta pelo anel  
    m = closestPrecedingNode(id)  
    m.findSucessor(id, ini)  
}
```



FingerTable nó 14	
I	Succ ( $n + 2^{i-1}$ )
1	$14 + 2^0 = 15 \% 16 \rightarrow 15$
2	$14 + 2^1 = 16 \% 16 \rightarrow 0$
3	$14 + 2^2 = 18 \% 16 \rightarrow 2$
4	$14 + 2^3 = 22 \% 16 \rightarrow 6$

# Buscas mais rápidas

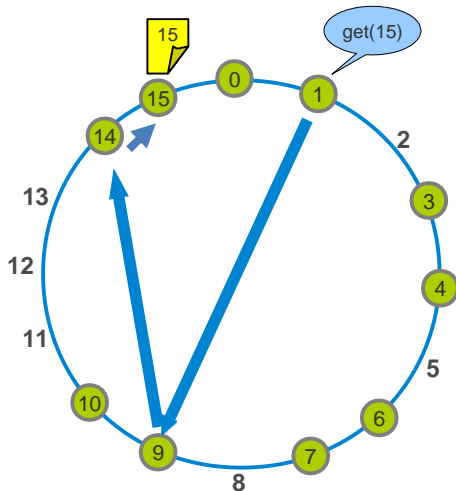
```
procedure n.findSuccessor(id, No ini) {  
  if (predecessor ≠ nil and id ∈ (predecessor, n]) then ini.response(id, n)  
  else if (id ∈ (n, successor]) then ini.response(id, successor)  
  else // encaminha a pergunta pelo anel  
    m = closestPrecedingNode(id)  
    m.findSucessor(id, ini)  
}
```



FingerTable nó 14	
I	Succ ( $n + 2^{i-1}$ )
1	$14 + 2^0 = 15 \% 16 \rightarrow 15$
2	$14 + 2^1 = 16 \% 16 \rightarrow 0$
3	$14 + 2^2 = 18 \% 16 \rightarrow 2$
4	$14 + 2^3 = 22 \% 16 \rightarrow 6$

# Buscas mais rápidas

```
procedure n.findSuccessor(id, No ini) {  
  if (predecessor ≠ nil and id ∈ (predecessor, n]) then ini.response(id, n)  
  else if (id ∈ (n, successor]) then ini.response(id, successor)  
  else // encaminha a pergunta pelo anel  
    m = closestPrecedingNode(id)  
    m.findSucessor(id, ini)  
}
```



# Conceitos adquiridos

- Busca distribuída em  $O(\log N)$
- Consistent Hashing.
- SPOF – Single Point of Failure.