Tema 3: Ejemplo de protocolo p2p

Sistemas Distribuidos

2022-2023

DHT

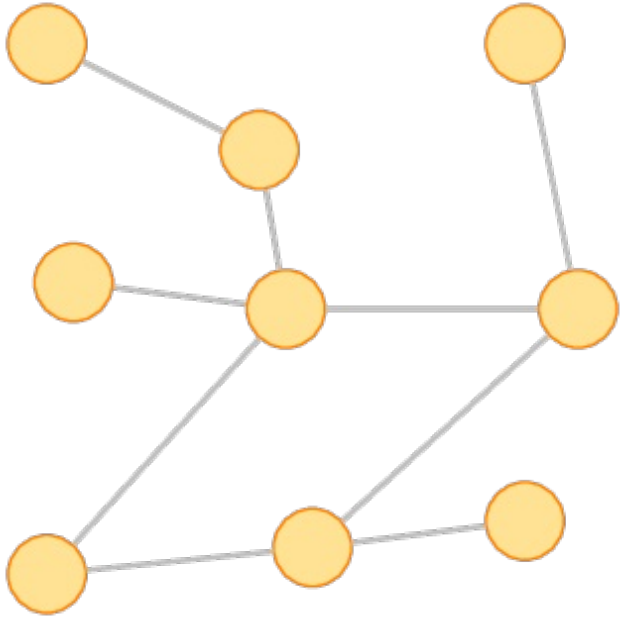
(*Distributed Hash Table*)

Problema que resuelve

> **Idea básica**: Almacén clave-valor distribuido

Creación de una red "overlay" de *peers*. Distribución de claves y valores entre los *peers*. Los *peers* van y vienen. Cada *peer* conoce sólo las IPs de algunos de los otros *peers*. Cada *peer* conoce la localización sólo de algunas claves Un *peer* debe poder encontrar qué IP tiene cualquier otro *peer*. Un *peer* debe poder encontrar qué *peer* almacena cualquier clave. Todo ello **sin un servidor** central.

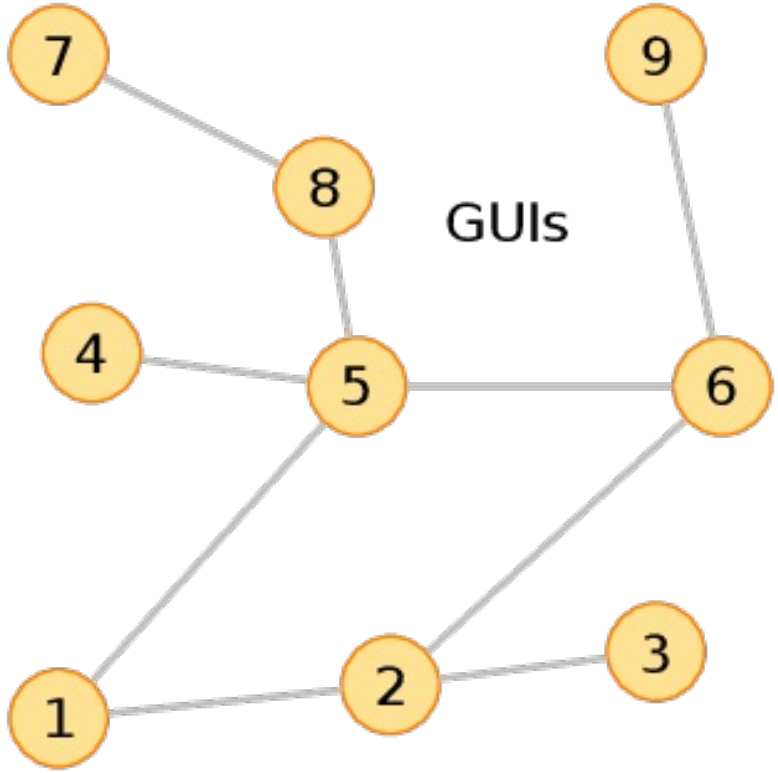
Ejemplo



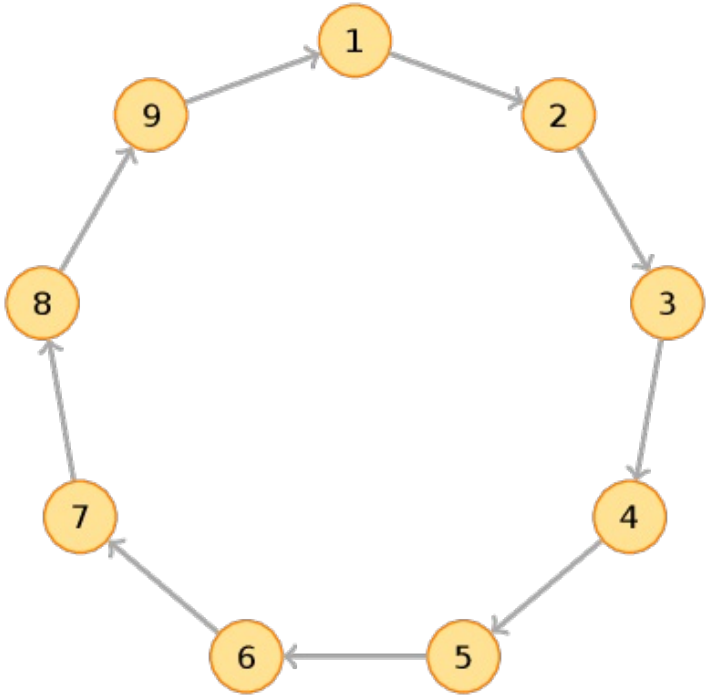
- Red de nodos, sin una red *overlay*
- Las líneas representan enlaces físicos
- ¿Cómo llega un mensaje de un nodo a otro cuya IP es desconocida?

Primer paso: GUIs

- Cada nodo recibe un GUI
- Es un identificador único (ej: un número *m* de 160 bits)



- Los GUIs crean un "anillo virtual".
- Las conexiones reales pierden importancia.
- Se crea una ordenación de los nodos, cada uno conoce la IP del siguiente



Problema 1: asignación de contenidos a nodos

- Cada contenido recibe un *id*, llamado **key**
- La **key** es también un número, en el mismo rango que los GUI de los nodos (por ejemplo, un hash de 160 bits)
- El contenido con la clave=**key** se asocia al nodo con GUI=**key**

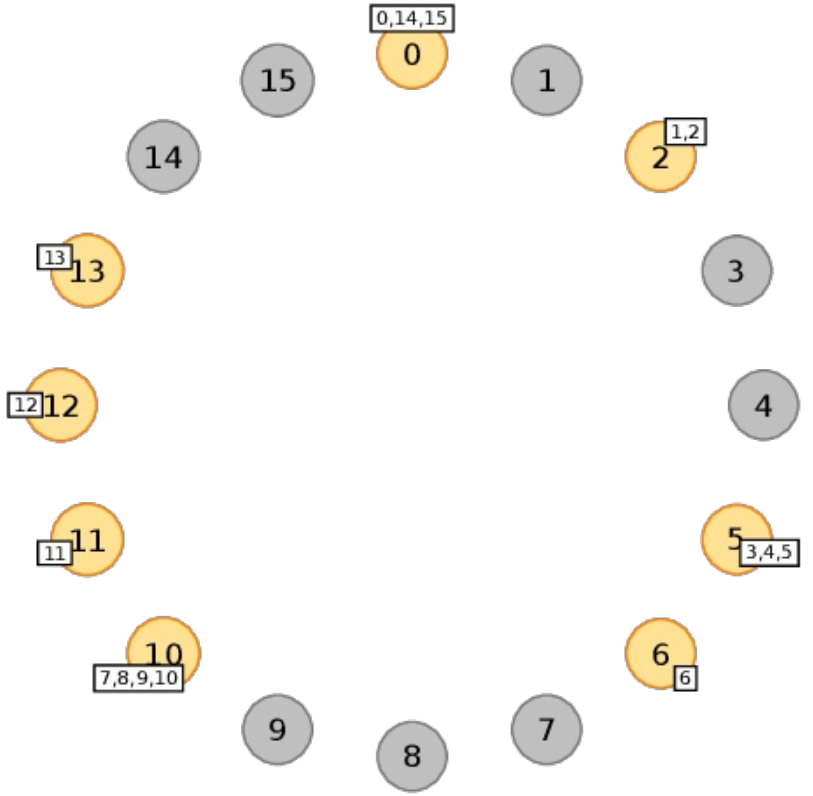
El anillo puede tener "huecos"

En ese caso se asigna al nodo con GUI = `siguiente(key)`

Siendo `siguiente()` la función que nos da el GUI siguiente a uno dado. Esta función se calcula de forma distribuida.

Asignación de contenidos

- Cada nodo es "responsable" de sus claves-valores.
- Los contenidos han de moverse si nodos van o vienen



Problema 2: ¿Qué nodos se conocen?

- Cada nodo mantiene la dirección IP y puerto de otros nodos
- Es su *lista de fingers*

Los nodos a mantener se eligen con una regla fija:

- El que está a distancia 1.
  - El que está a distancia 2.
  - El que está a distancia 4.
  - ...
- El que está a distancia 2^{n-1}. En una red formada por 2^n nodos, debe mantener por tanto n direcciones.

¿Qué es la distancia entre nodos?

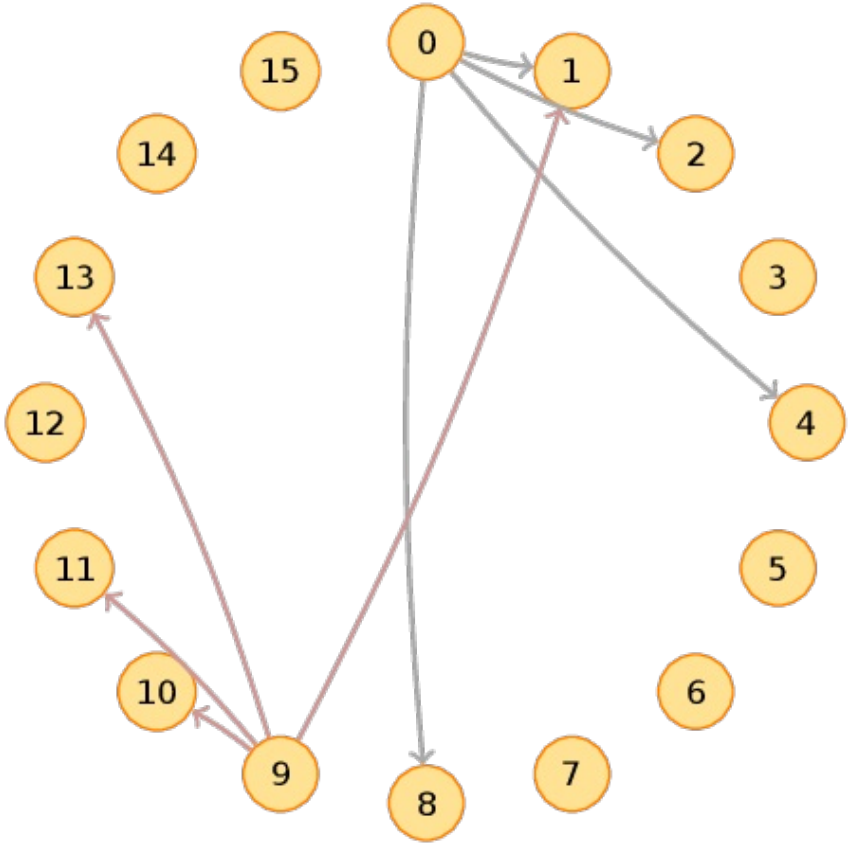
- Depende del algoritmo P2P (Chord, Kademlia, etc.)
- Se calcula en base a sus GUI.

Entre dos nodos con GUI respectivos: A y B

- **Chord**
  distancia(A,B) = mod(B-A+2^n, 2^n)

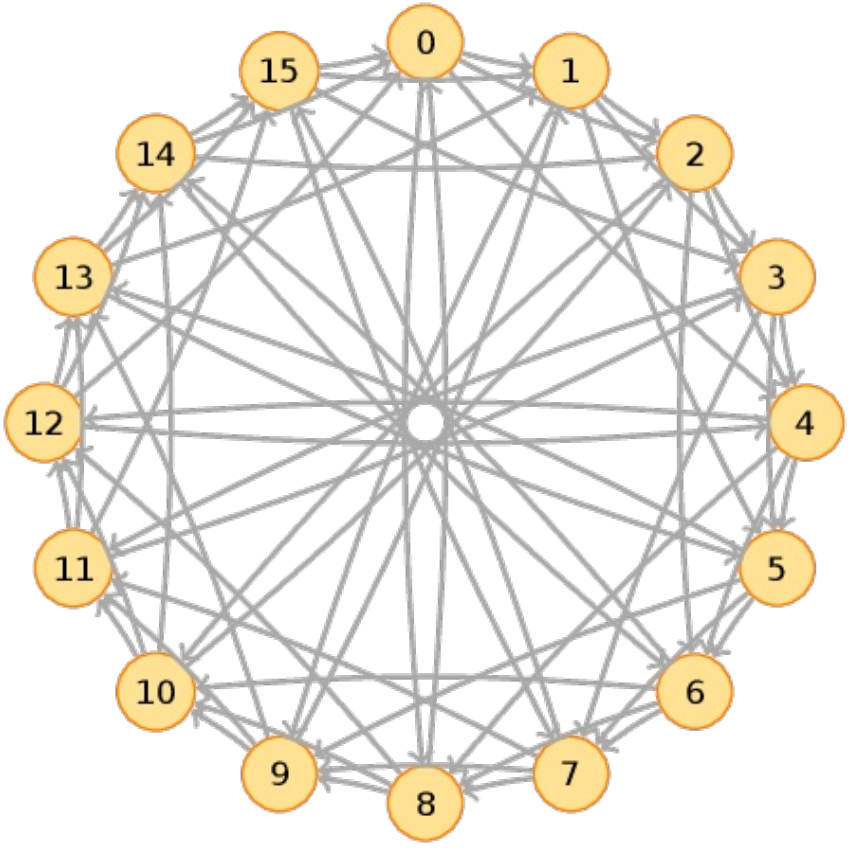
- **Kademlia**
  distancia(A,B) = A xor B
  Cumple que: distancia(A,B) == distancia(B,A)

Listas de *fingers* en Chord



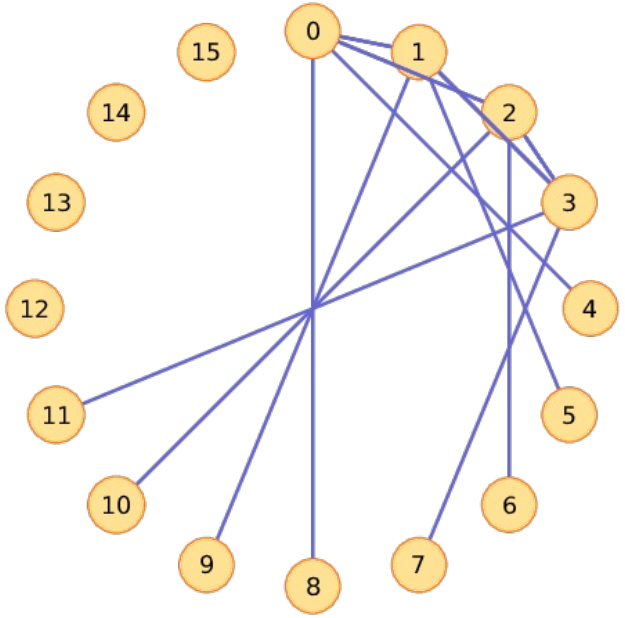
Fingers de nodos 0 y 9

Listas de *fingers* en Chord



Fingers de todos los nodos

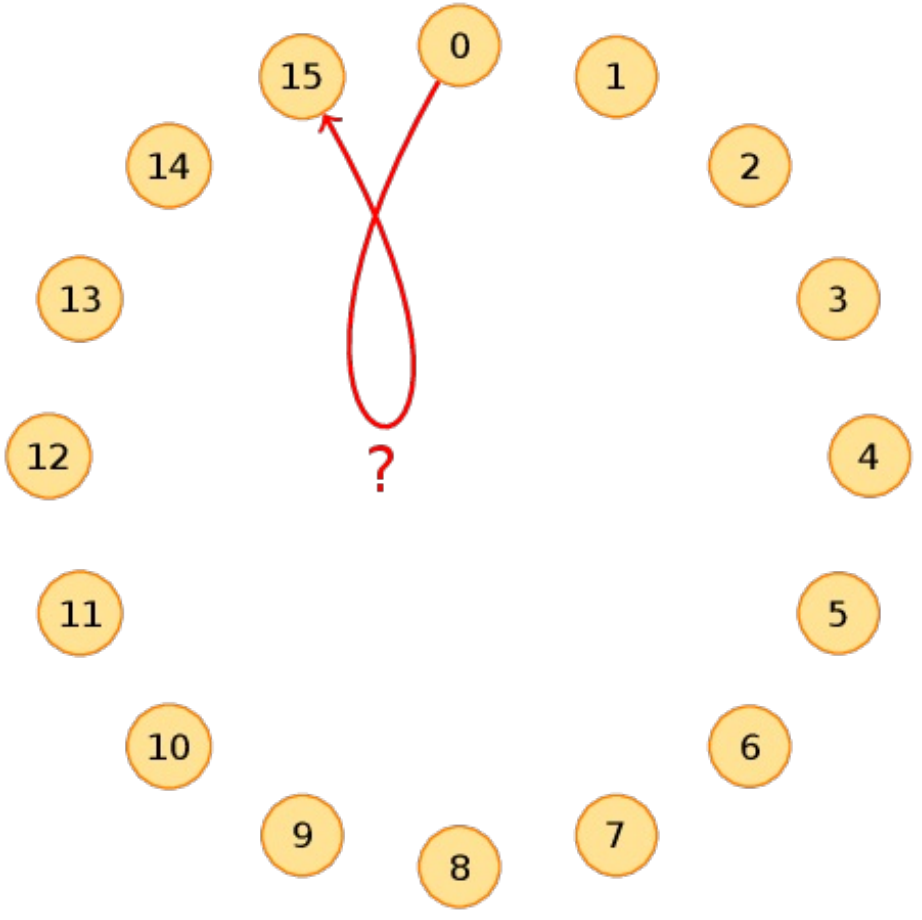
Listas de *fingers* en Kademlia



- `0` Tiene por *fingers* `[1,2,4,8]`
- `1` Tiene por *fingers* `[0,3,5,9]`
- `2` Tiene por *fingers* `[3,0,6,10]`
- `3` Tiene por *fingers* `[2,1,7,11]`

Problema 3. Rutas entre nodos

El nodo 0 quiere comunicarse con el nodo 15.
¿Cómo obtiene su IP y puerto?

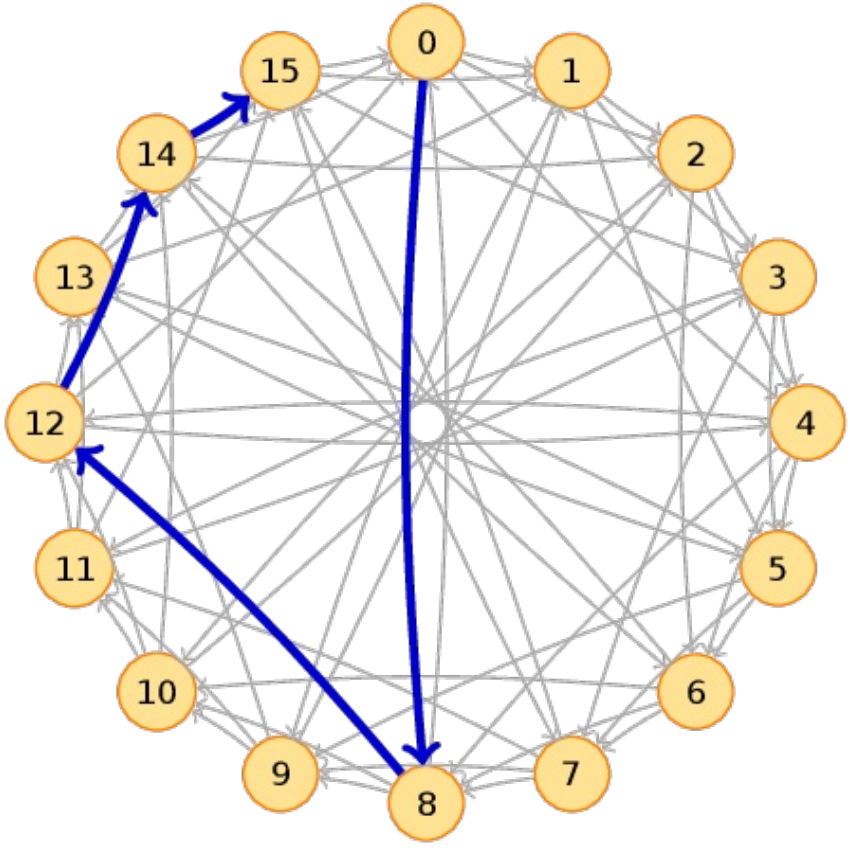


Algoritmo de búsqueda

- Si el nodo con que quiere contactar está en su lista de *fingers* → Resuelto
- Si no, preguntarle al *finger* más alto, anterior al destino.
- Si éste sabe la respuesta, la envía.
  Si no, aplica el mismo algoritmo
- El primero que sepa la respuesta la envía al origen.

*Ejemplo**
. El nodo 0 quiere obtener IP y puerto del nodo 15. . El *finger* más alto anterior a 15 es el 8. . Le pide al 8 que reenvíe la pregunta.

Solución del ejemplo

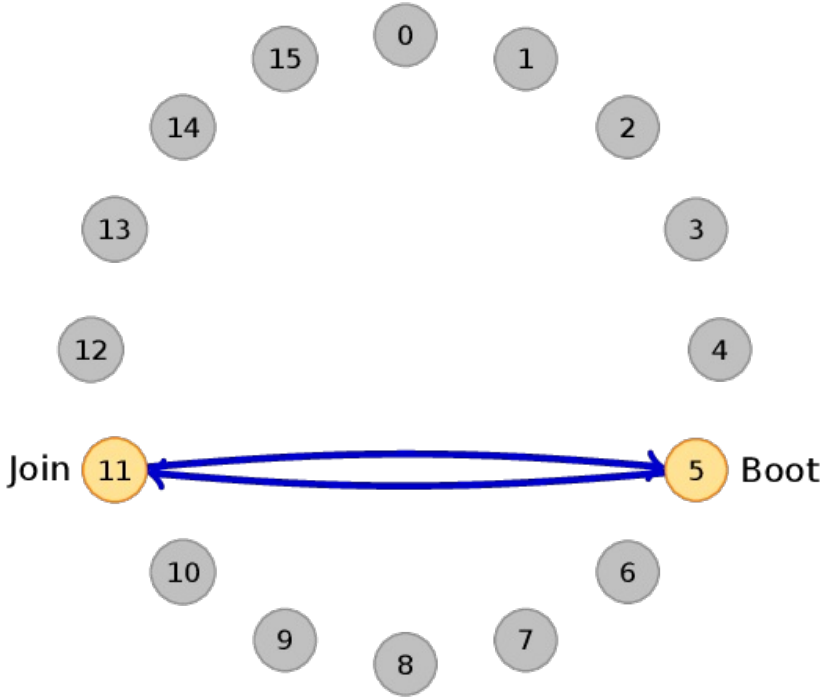


La búsqueda finaliza en 4 pasos como máximo (en n en una red de 2^n nodos)

Problema 4: el arranque de la red

- La red debe comenzar por un nodo (*Boot*).
  - Otros nodos se añaden posteriormente a la red.
    - Para añadirse deben conocer (IP, puerto) del nodo *Boot*.
    - O de cualquiera de los otros nodos ya añadidos.
- Cuando un nodo se añade, debe anunciarlo.
  - Los restantes nodos deben actualizar sus listas de *fingers*.
    - Y las claves-valores asignados.

El arranque, gráficamente. *Boot* y primer *join*

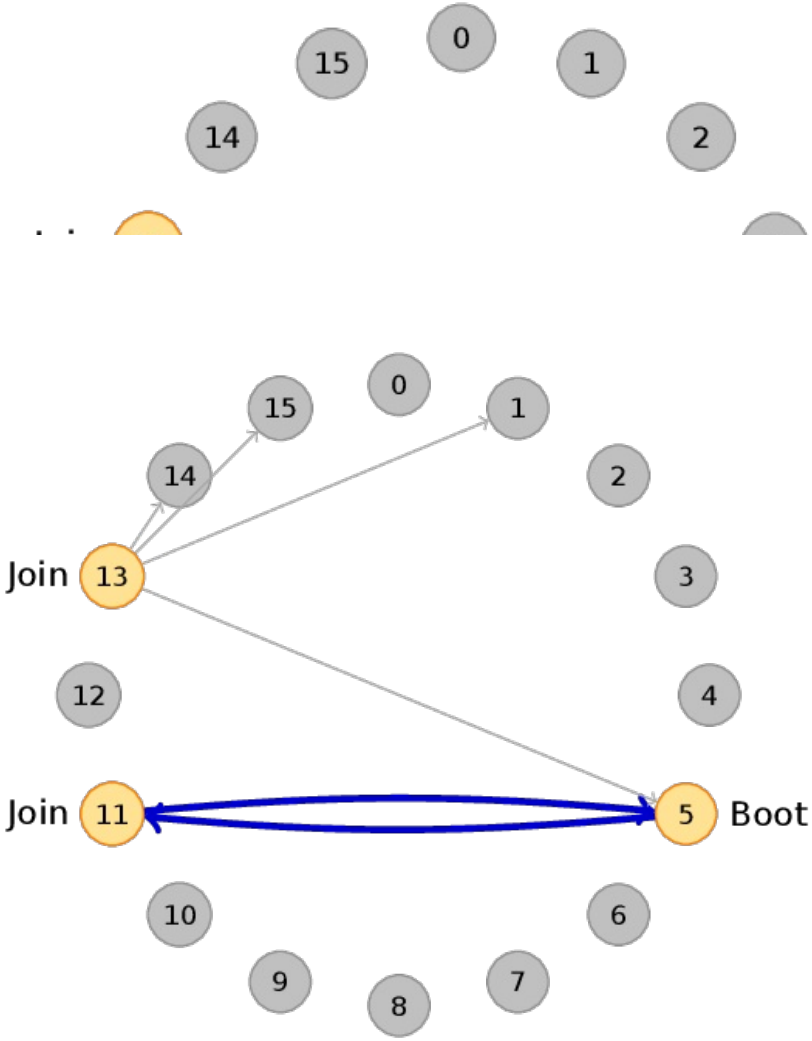


Los 3 primeros *finger* del nodo 5 apuntan al 11, y el cuarto a sí mismo.

Los 4 *finger* del nodo 11 apuntan al 5

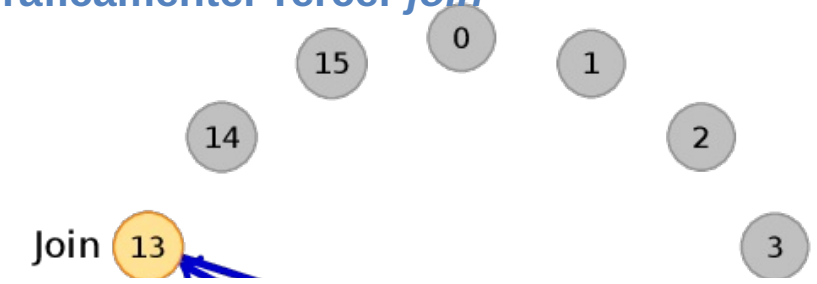
paso[

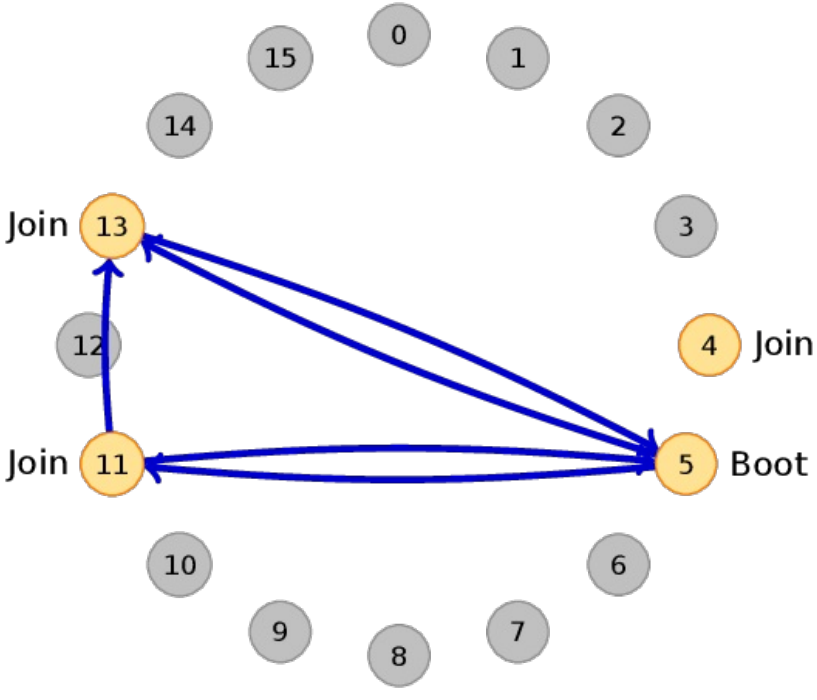
El arranque, gráficamente. Segundo *join*



paso[

El arranque, gráficamente. Tercer *join*

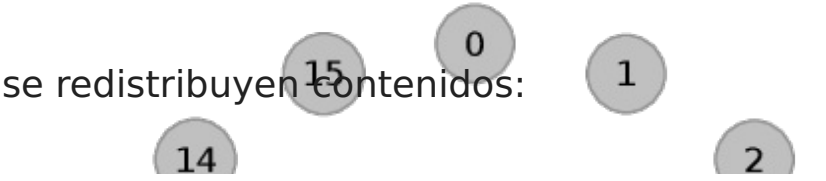


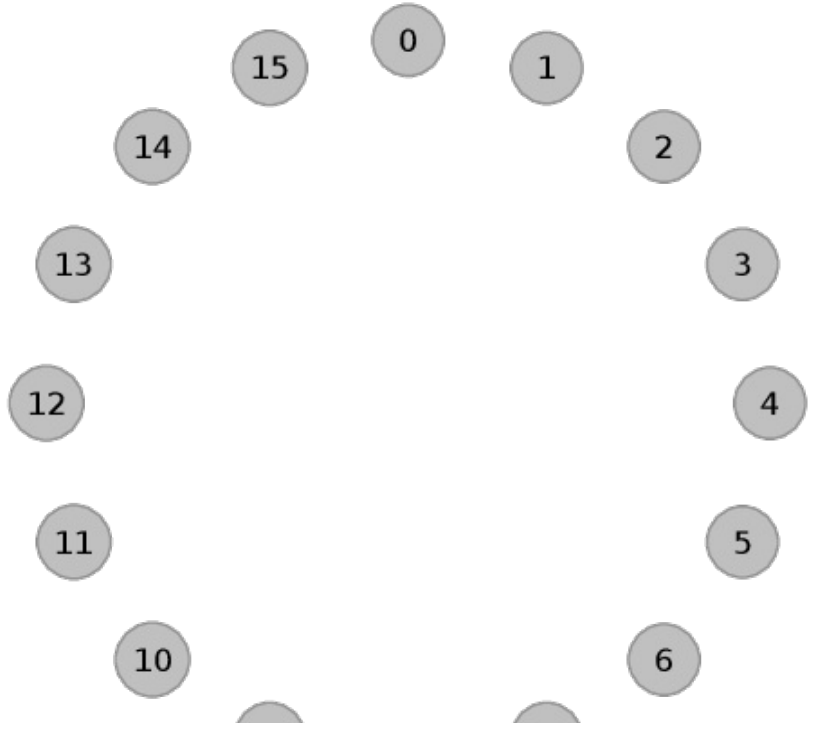


paso[

Etc...

Y en cada *join* se redistribuyen contenidos:





Problema 5: nodos que se van

- Cuando un nodo se va, debe notificarlo para que:
  - Los demás actualicen sus *fingers*
  - Quien corresponda se haga cargo de sus contenidos
- Pero a veces un nodo desconecta "sin avisar"
- Los demás nodos periódicamente deben
  - Comprobar que sus *finger* siguen vivos
  - Actualizar la red si no

Conclusión

- Este tipo de algoritmos **evitan servidores** que podrían ser cuellos de botella.
- La solución es **altamente escalable**, pues a medida que se añaden nodos el trabajo se reparte más.
- Una búsqueda en la red *overlay* no pasa por todos los nodos, se completa en un tiempo O(log(n)) y siempre encuentra lo que busca (si existe).
- Existe mucha investigación sobre **algoritmos** para mapear contenidos a nodos, calcular distancias entre nodos, enrutar las búsquedas, etc.
- La implementación no es trivial.