# Arquitectura de los Sistemas Distribuidos

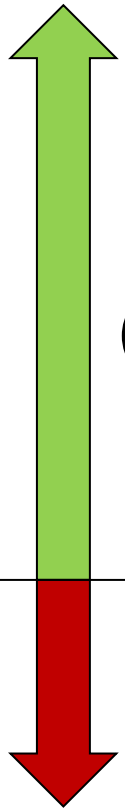
*(2ª parte)*

# Índice

- Modelos de interacción
- Arquitectura cliente-servidor
  - Variaciones del modelo
  - Aspectos de diseño del modelo cliente/servidor
- Arquitectura editor-subscriptor
- Arquitectura productor-consumidor

---

- Arquitectura *peer-to-peer*
- Arquitecturas para computación distribuida



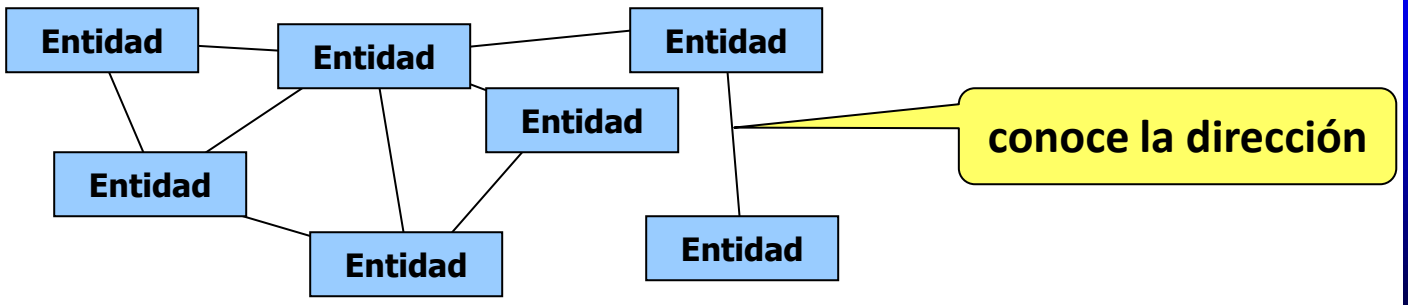
1ª parte (incluida en examen)

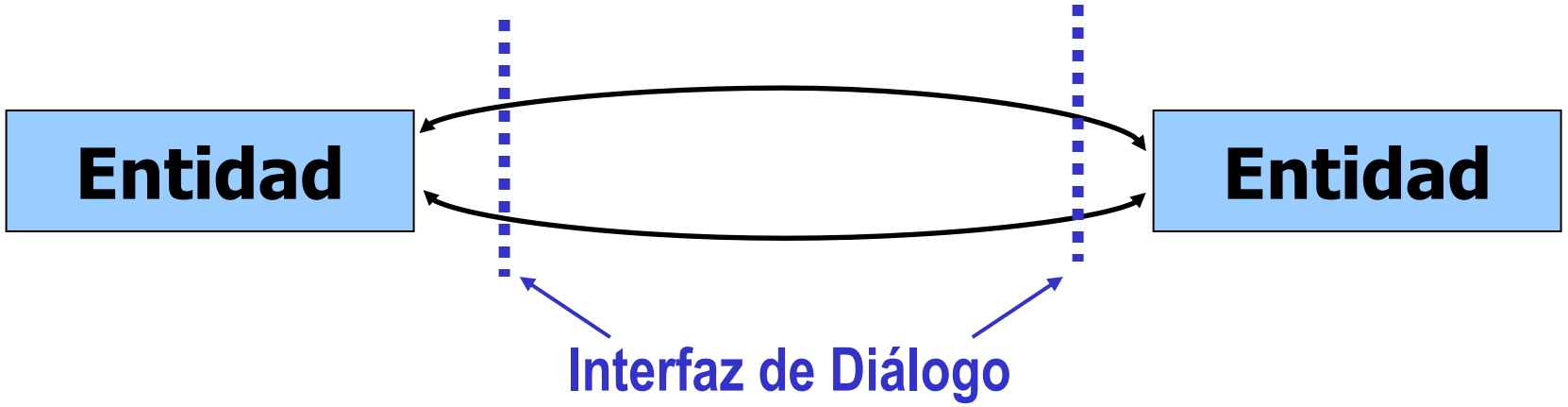
2ª parte (no incluida en examen)

# Modelo *Peer-to-Peer (P2P)*

- Todos los nodos tienen mismo rol y funcionalidad
  - No hay cuellos de botella ni puntos críticos de fallo
  - Se aprovechan recursos de todas las máquinas
- Se suelen caracterizar además por:
  - Volatilidad: nodos entran y salen del sistema
  - Capacidad de autogestión sin una autoridad global centralizada
- Uso de una *overlay network*:
  - Red lógica sobre la física
  - Nodos conocen las direcciones de otros nodos del sistema P2P
  - Nodos gestionan el encaminamiento
- Desventajas de arquitectura P2P
  - Difícil administración conjunta y mayores problemas de seguridad

# Esquema *Peer-to-Peer (P2P)*



# Aplicaciones de *P2P*

- Compartir contenidos (*content-sharing*)
  - Nodo exporta contenidos locales y puede importar remotos
  - *Napster*, *Gnutella*, *Kazaa*…
- Distribución de contenidos
  - Descarga contenido de gran volumen por muchos clientes (*BitTorrent*)
- Almacenamiento distribuido
  - Sistemas de ficheros P2P, Cachés P2P
  - Soporte para criptomonedas (*Bitcoin/Blockchain*)
- Comunicación
  - Mensajería instantánea, VoIP, Videoconferencia (*Skype*)
- *Streaming*: P2PTV
- Computación distribuida

# *P2P* para compartir contenidos

- Primera aplicación relevante de P2P (desde finales siglo XX)
  - Gran impacto social con muchas repercusiones jurídicas y económicas
  - Pero nos centramos en aspectos tecnológicos
- Muy frecuente: la usaremos como ejemplo para estudiar P2P
- Identificamos operaciones genéricas para facilitar el estudio:
  - *Alta*: nodo se incorpora a la red P2P
  - *Publica*: pone a disposición de otros su contenido local
  - *Búsqueda:* solicita un contenido
  - *Descarga*: obtiene de un nodo o de múltiples el contenido localizado
  - *Elimina*: deja de exportar contenido local
  - *Baja*: nodo deja la red P2P

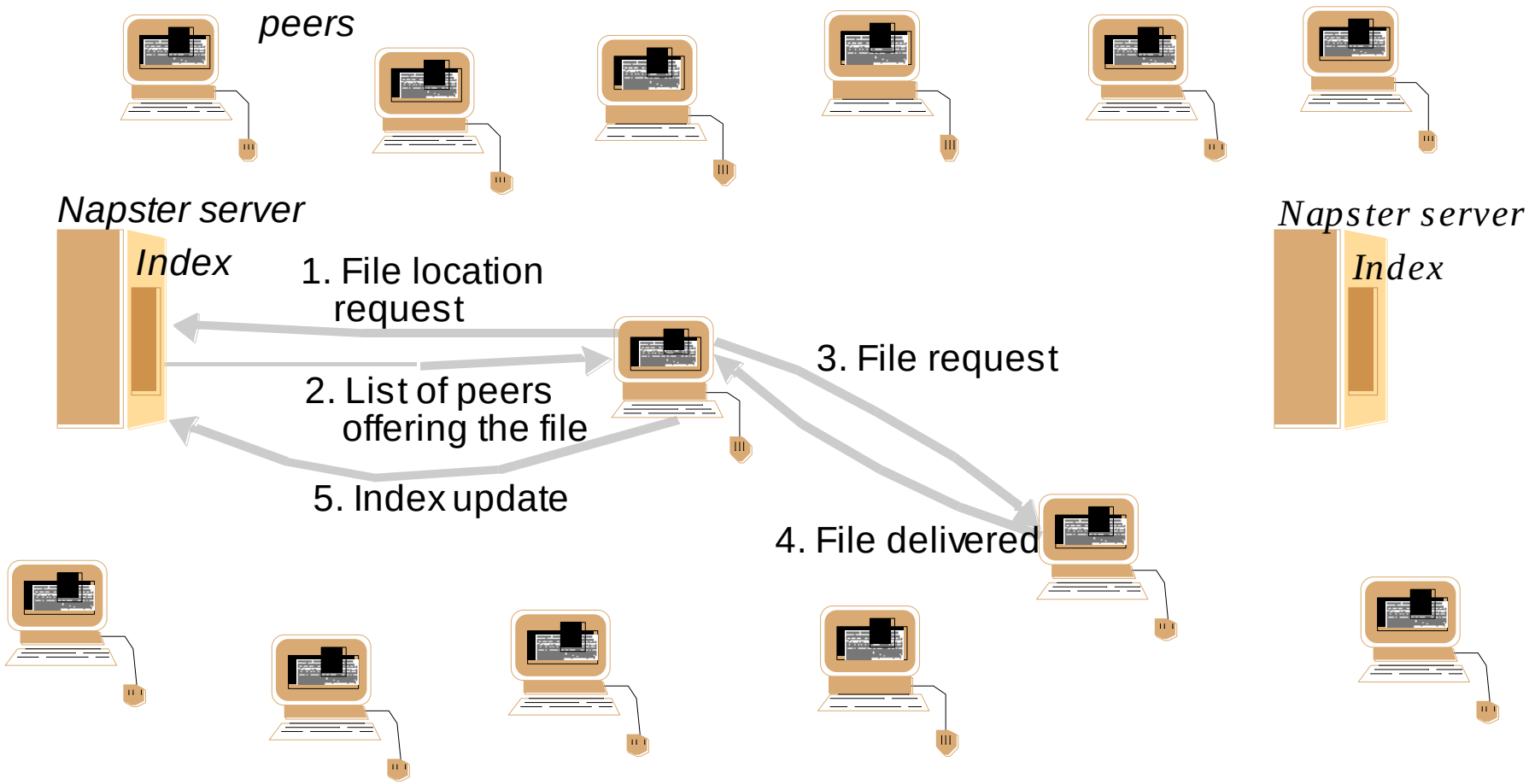
# Tipos de sistemas P2P

- Híbridos: Cliente/servidor + P2P (Naspter, BitTorrent)
- Desestructurados: P2P puros (Gnutella, Blockchain)
  - Topología de conexión lógica (*overlay network*) arbitraria
- Jerárquicos (Kazaa, Skype):
  - Nivel superior: P2P desestructurados
  - Nivel inferior: P2P híbridos
- Estructurados (Chord, Pastry)
  - Topología de conexión prefijada (p.e. anillo)
- Orden en esa clasificación refleja la cronología
- En casos de estudio: revisaremos las propuesta originales
  - No mejoras y evoluciones posteriores

# Sistemas P2P híbridos

- Cliente/servidor + P2P: existe un nodo que actúa de servidor
  - Cuello de botella y punto crítico de fallo
  - Servidor conoce dirección de nodos y localización de recursos
- P2P híbrido para compartir contenidos: Napster (1999)
  - Nuevo nodo conoce dirección de servidor
  - Se conecta con él (*alta*) y le informa de qué ficheros exporta (*publica*)
  - Solicita al servidor un recurso (*búsqueda*) **1**
  - Obtiene del servidor la lista de nodos que contienen ese recurso **2**
  - Solicita la *descarga* del nodo con mejor tiempo de respuesta **3**
  - Completada descarga **4**, informa a servidor de que posee copia **5**
    - Se lo podrán pedir otros nodos a partir de ahora
  - Se desconecta del servidor (*elimina* y *baja*)
- eDonkey: similar pero descarga de varios nodos por trozos

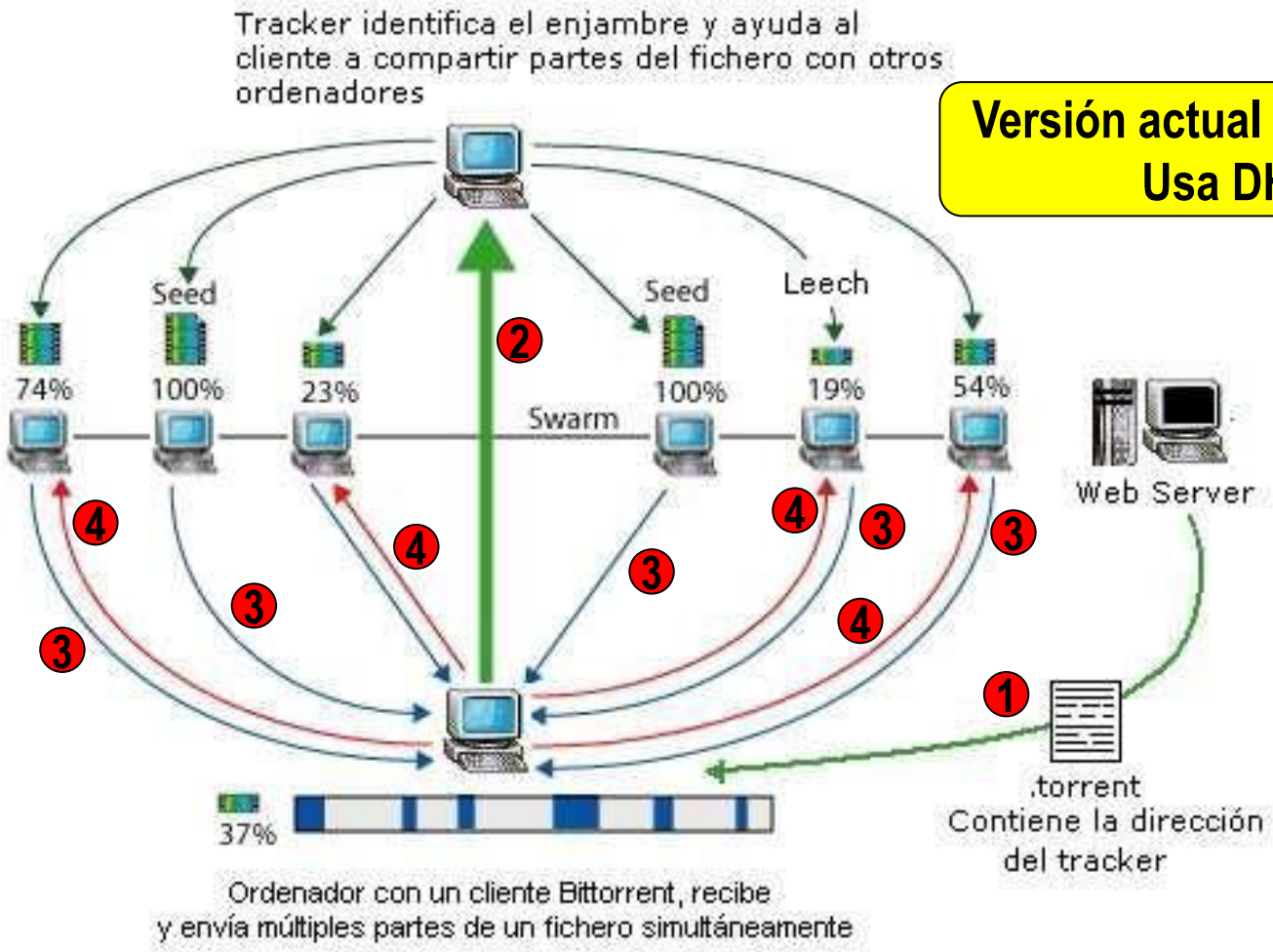
# Napster (Libro de Couloris et al.)



# Protocolo *BitTorrent* (2001)

- Distribución de contenidos mediante P2P híbrido
- Fichero muy grande descargado por muchos clientes simultáneos
  - Dueño de fichero requeriría sistema de altas prestaciones
  - Alternativa: cliente puede descargar trozos de otro cliente en descarga
  - Servidor (*tracker*) conoce qué clientes están descargando el fichero
- Modo de operación:
  - Dueño almacena fichero en su nodo (se le considerará un cliente más)
  - Crea fichero *.torrent*: metadatos del fichero + dirección del *tracker*
  - Cliente encuentra de alguna forma *.torrent* del recurso **1**
  - Contacta con *tracker* ⟶ obtiene lista de algunos clientes en descarga **2**
  - Se conecta con esos clientes que le dicen qué trozos tienen
  - Descarga trozos de clientes en paralelo **3**
  - Primero los menos frecuentes: para aumentar la replicación
  - Cuando completa descarga de un trozo, lo notifica a esos clientes
  - Otros clientes le comenzarán a solicitar trozos **4**

# Protocolo *BitTorrent (wikipedia)*




**Versión actual sin *tracker***
**Usa DHT**

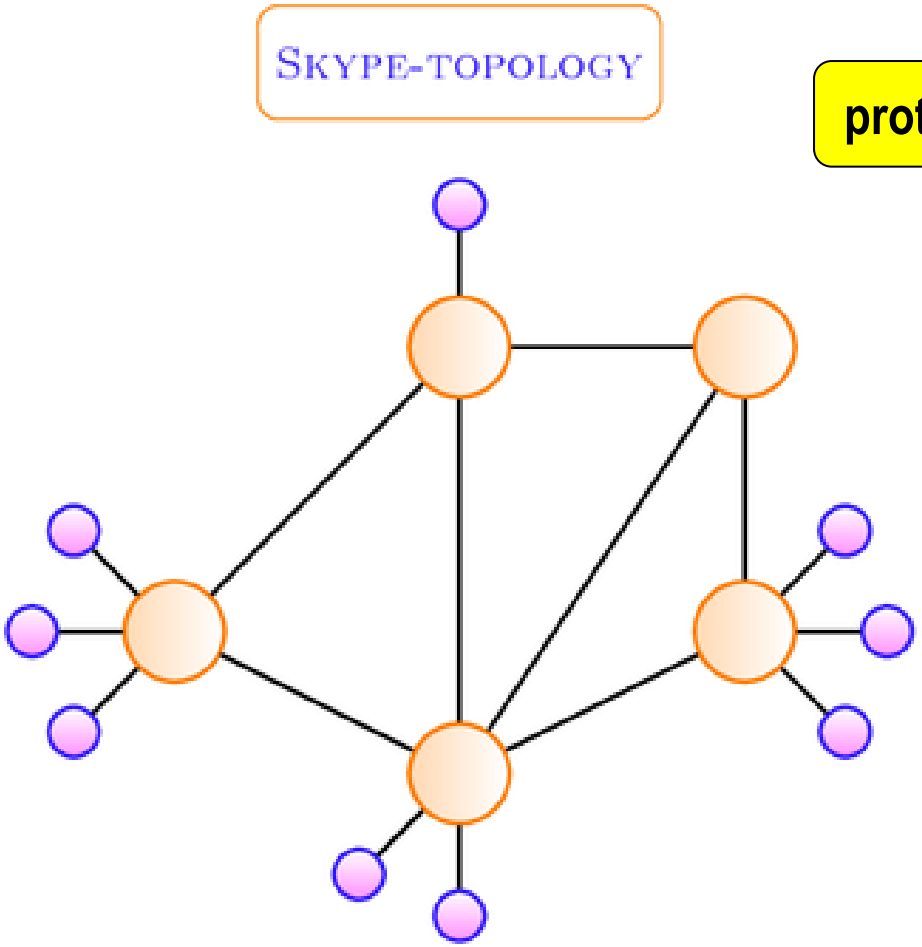
# Sistemas P2P desestructurados

- Todos los nodos con la misma funcionalidad
  - No hay cuellos de botella ni puntos críticos de fallo
  - Nodos conectados intercambian direcciones de sus "vecinos"
    - Refuerza conectividad del sistema
- P2P desestructurado para compartir contenidos: Gnutella (2000)
  - Nuevo nodo descubre de alguna forma dirección de nodo en el sistema
  - Contacta con ese nodo y se integra en el sistema (*alta*)
  - Localización de recursos (*búsqueda*) por inundación (***flooding***)
    - Se propaga petición a vecinos y estos a los suyos…
    - Búsqueda afecta a todos los nodos del sistema.
    - Se puede usar TTL para limitar la propagación
    - Alternativa: usar una propagación aleatoria
  - *Descarga* contenido de nodo(s) encontrado(s)
- Gnutella evolucionó al P2P jerárquico: O(log n) en vez O(n)

# Sistemas P2P jerárquicos

- Sistema jerárquico: nodos ordinarios ON y supernodos SN
  - Nodo ordinario debe estar asociado a un supernodo
  - Supernodo tiene información de su grupo de nodos asociados
  - SN dinámicamente elegido entre ONs por potencia, conectividad…
- Mezcla de desestructurado e híbrido
  - Red desestructurada de supernodos (como Gnutella)
  - Cada SN servidor de ONs asociados al mismo (como Napster)
- P2P jerárquico para compartir contenidos: Kazaa (2001)
  - Nodo obtiene dirección de algún supernodo SN
    - En algún momento este nodo puede llegar a ser elegido como SN
  - Se conecta con él (*alta*) y le informa de qué ficheros exporta (*publica*)
  - Solicita a SN un recurso (*búsqueda*):
    - SN conoce cuáles de sus nodos dependientes tienen ese recurso
    - Además, propaga petición por red de SNs
  - *Descarga* contenido de nodo(s) encontrado(s)

# P2P jerárquico para comunicación: Skype




**protocolo Skype no es público**

https://arstechnica.com/information-technology/2012/05/skype-replaces-p2p-supernodes-with-linux-boxes-hosted-by-microsoft/

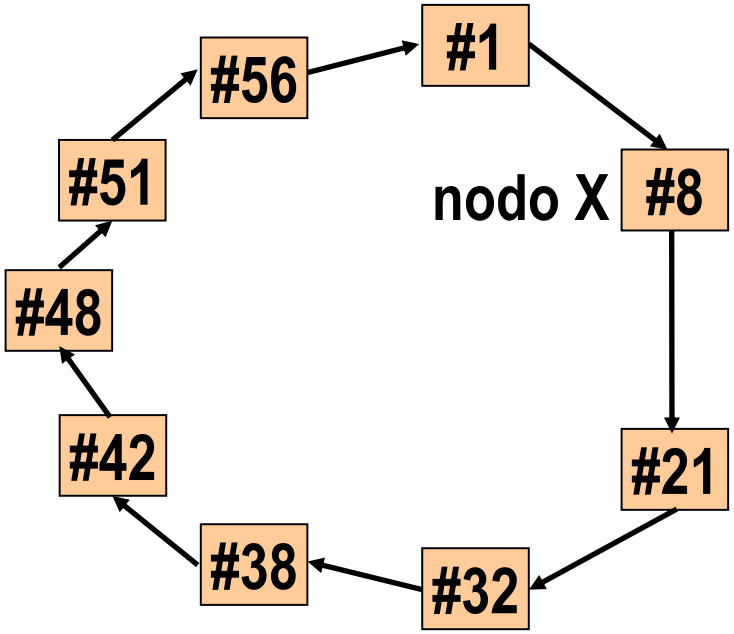
# Sistemas P2P estructurados

- Topología de conexión prefijada
  - hipercubo en CAN, árbol en Tapestry, anillo en Chord…
- ¿Cómo conoce nodo su ubicación en la topología?
  - Aplica función *hash* a su ID (nombre, IP, MAC…):
    - Valor obtenido identifica su ubicación
  - *Distributed Hash Table* (DHT)
- Localización de recursos O(log n): misma estrategia
  - Aplica misma función *hash* a ID de recurso
    - Valor obtenido identifica en qué máquina está almacenado
- Uso para compartir contenidos:
  - *Alta*: nodo obtiene dirección de algún nodo y se incorpora a red
    - En la posición lógica que determina la función *hash*
  - *Publica*: nodo aplica la función *hash* para ubicar el recurso
  - *Búsqueda*: nodo aplica la función *hash* para encontrarlo

# Protocolo *Chord* (2001)

- Uso de función *hash* que genera valores de *m* bits
  - Función *hash* se aplica a ID de nodos y recursos
  - Valor *m* tal que muy baja probabilidad de que ≠ IDs den mismo *hash*
- Nodos se ubican en anillo por *hash* creciente (módulo $2^m$)
  - Nodo guarda IP y *hash* de sucesor y de predecesor
- Ubicación en anillo de nuevo recurso
  - Se calcula su *hash* y se ubica en el primer nodo con un *hash* ≥
- Alta de un nuevo nodo
  - Se calcula su *hash* y se ubica entre los dos nodos correspondientes
  - Se ajusta información para reflejar nuevos sucesores y predecesores
  - Se le asignan los recursos de su sucesor que le correspondan
- Baja de un nodo
  - Se ajusta información para reflejar nuevos sucesores y predecesores
  - Se transfieren sus recursos al sucesor

# Anillo $2^6$



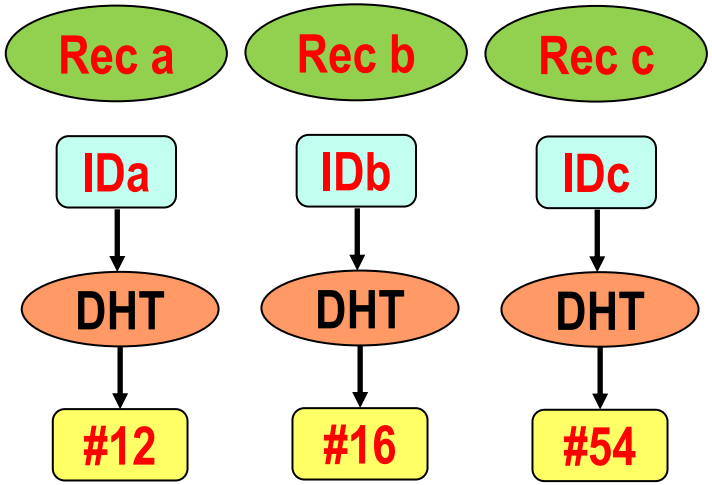

IP y *hash* propios
IP y *hash* predecesor
IP y *hash* sucesor

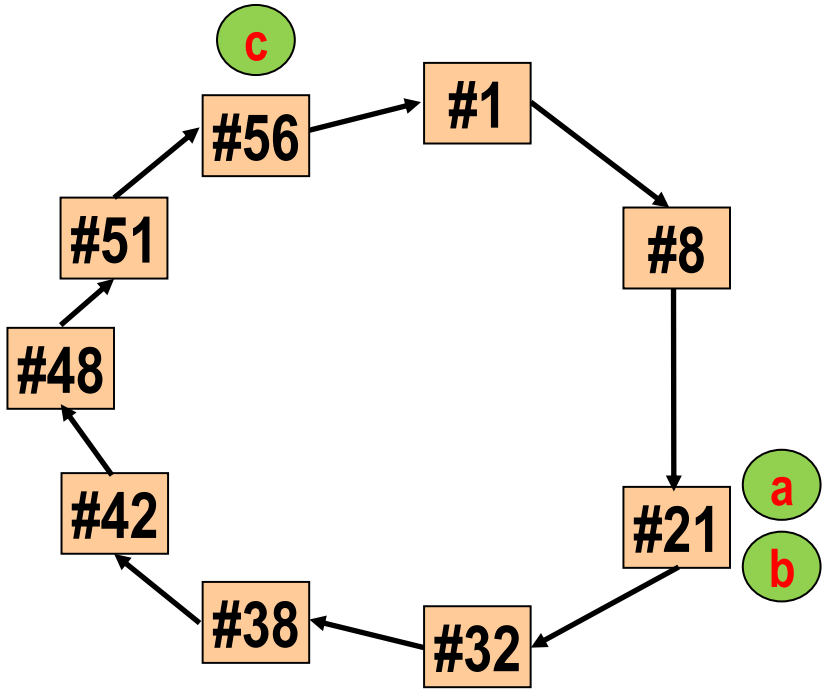
Info en un nodo

IPX | #8
IP predecesor | #1
IP sucesor | #21

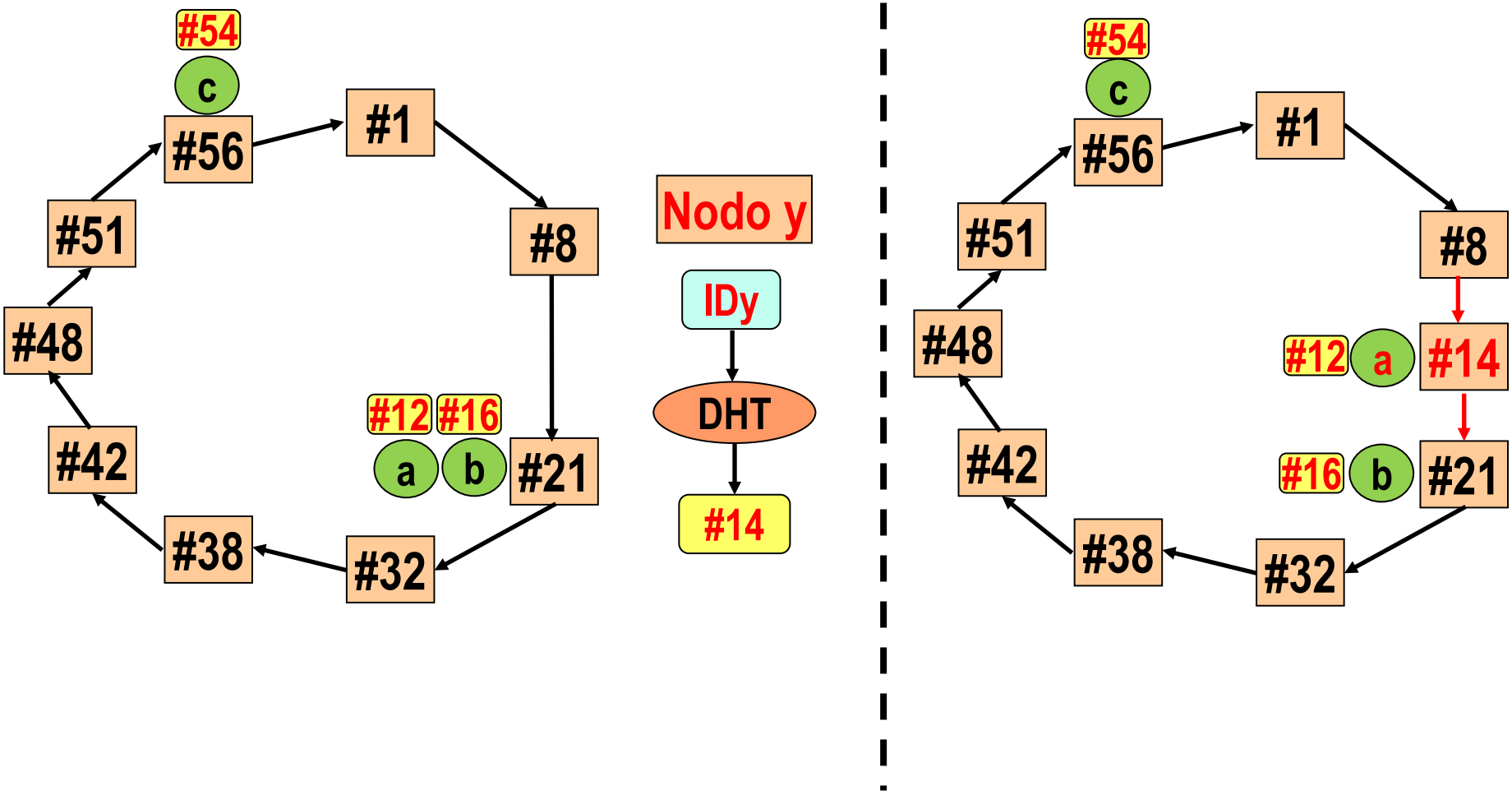
Info en nodo X

# Alta de recursos en anillo

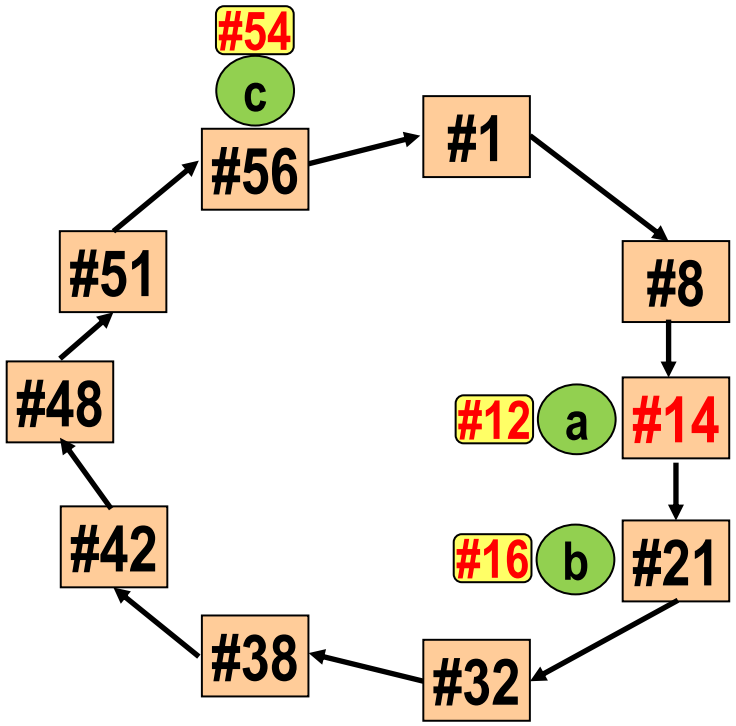


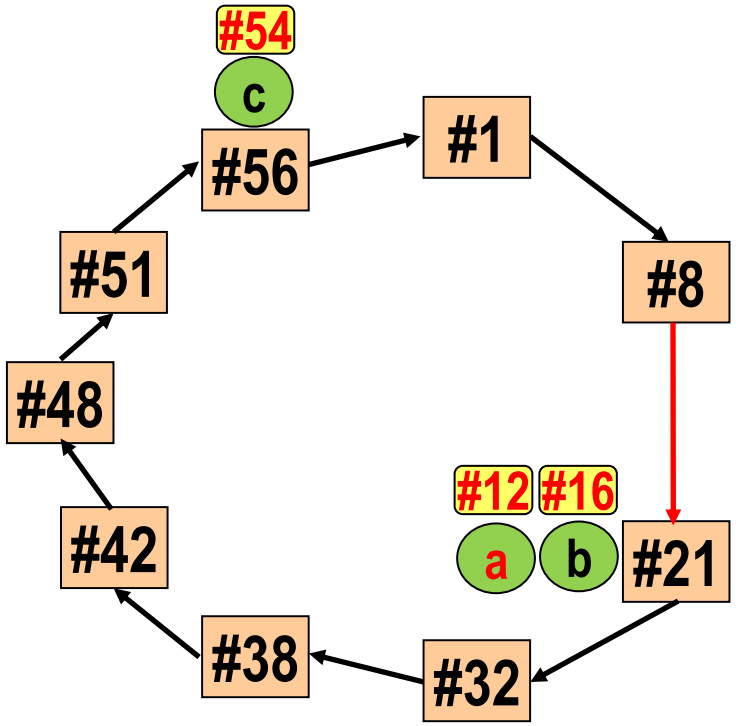



# Alta de nodo en anillo



# Baja de nodo en anillo

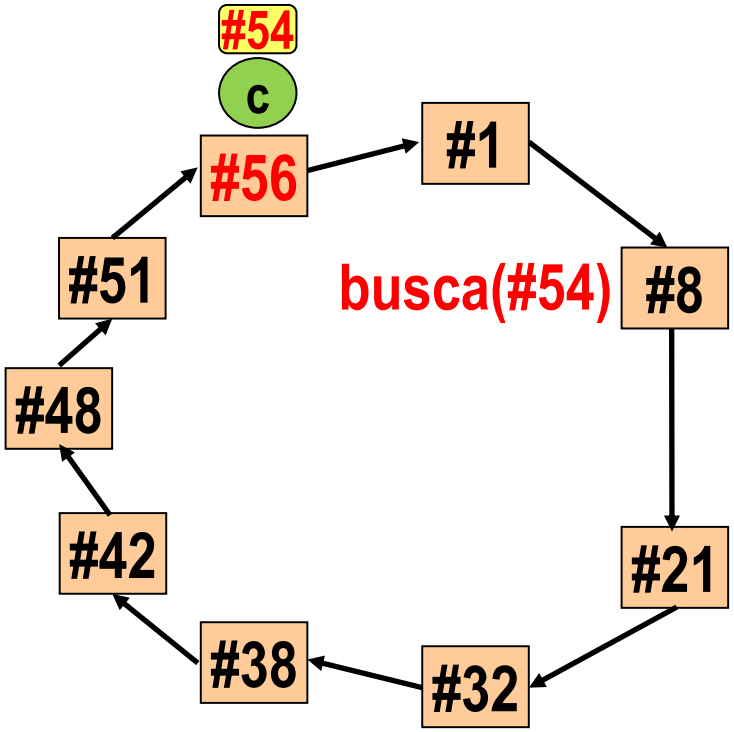


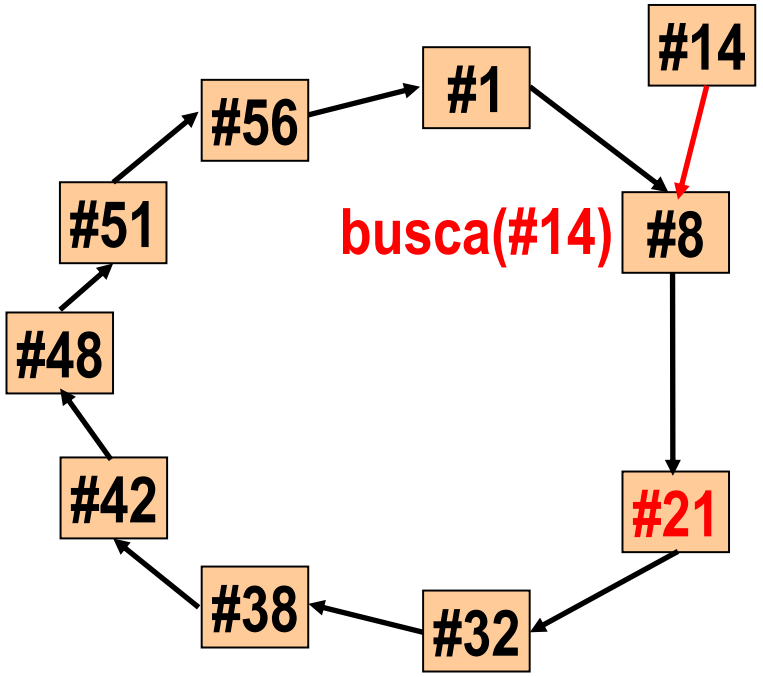
# Búsqueda de valores *hash*

- Nodo necesita encontrar primer nodo con *hash* ≥ que uno dado
  - Para publicar o encontrar un recurso
  - Porque un nuevo nodo le ha pedido que busque su ubicación
- Solución ineficiente: búsqueda lineal *O(N)* en el anillo
  - Propaga búsqueda a siguiente hasta que encuentre el nodo buscado
- *Fingers*: nodo guarda información de *m-1* sucesores
  - Primer nodo cuyo *hash* está a una distancia $\geq 2^1$ del hash de este nodo
  - Primer nodo cuyo *hash* está a una distancia $\geq 2^2$ del hash de este nodo
  - Primer nodo cuyo *hash* está a una distancia $\geq 2^{m-1}$ de hash de este nodo
- Búsqueda con *fingers O(m)*:
  - Si *hash* buscado no es el mío o el de mi sucesor
  - Propago búsqueda a *finger* con *hash* + cercano al buscado sin pasarse
  - Ese nodo repite el mismo proceso

# Necesidad de búsqueda de *hash*

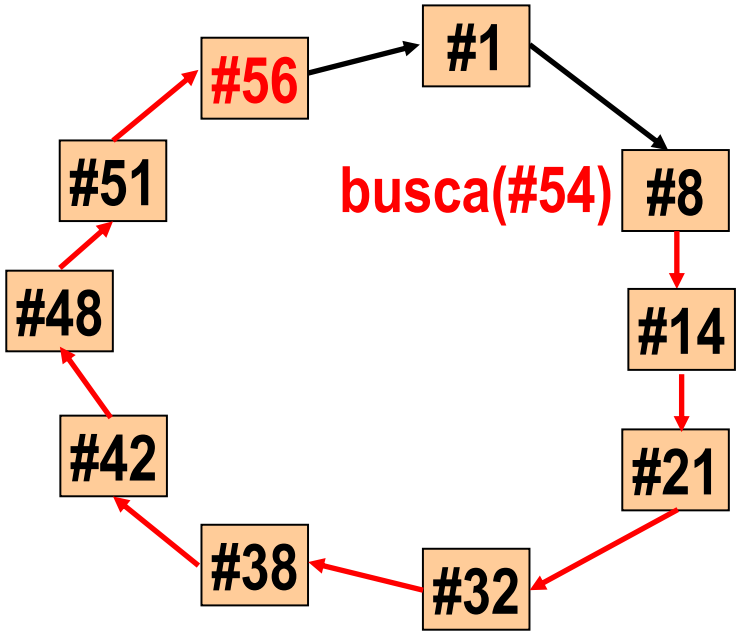



nodo #8 quiere publicar o encontrar recurso #54

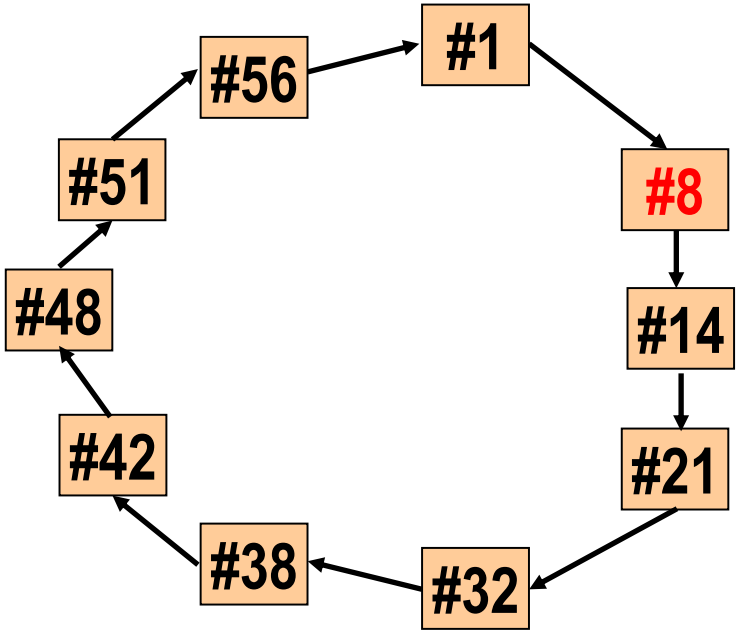



nodo nuevo #14 pide a nodo #8 que le busque su ubicación

# Búsqueda lineal



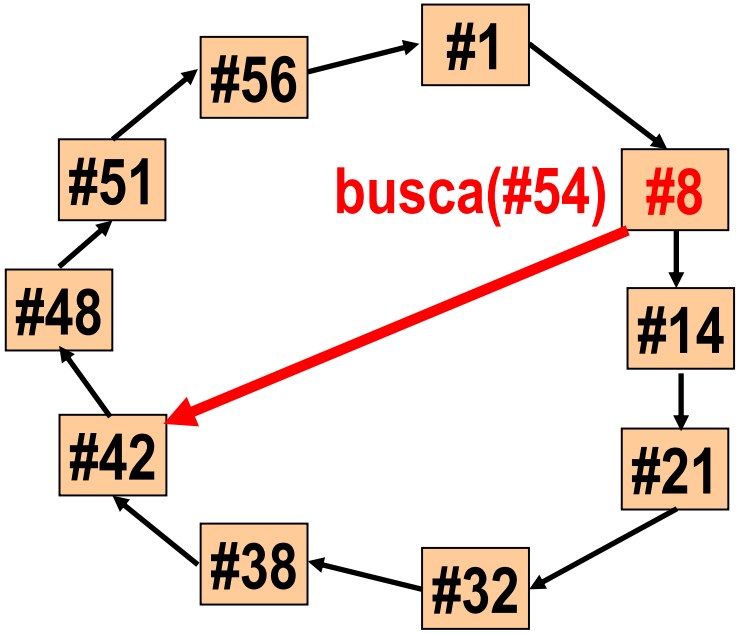
# *Fingers* de un nodo




**Info en nodo**

| |
|---|
| IP nodo \| #8 |
| IP predecesor \| #1 |
| IP sucesor \| #14 |
| IP *finger* dist 2 \| #14 |
| IP *finger* dist 4 \| #14 |
| IP *finger* dist 8 \| #21 |
| IP *finger* dist 16 \| #32 |
| IP *finger* dist 32 \| #42 |

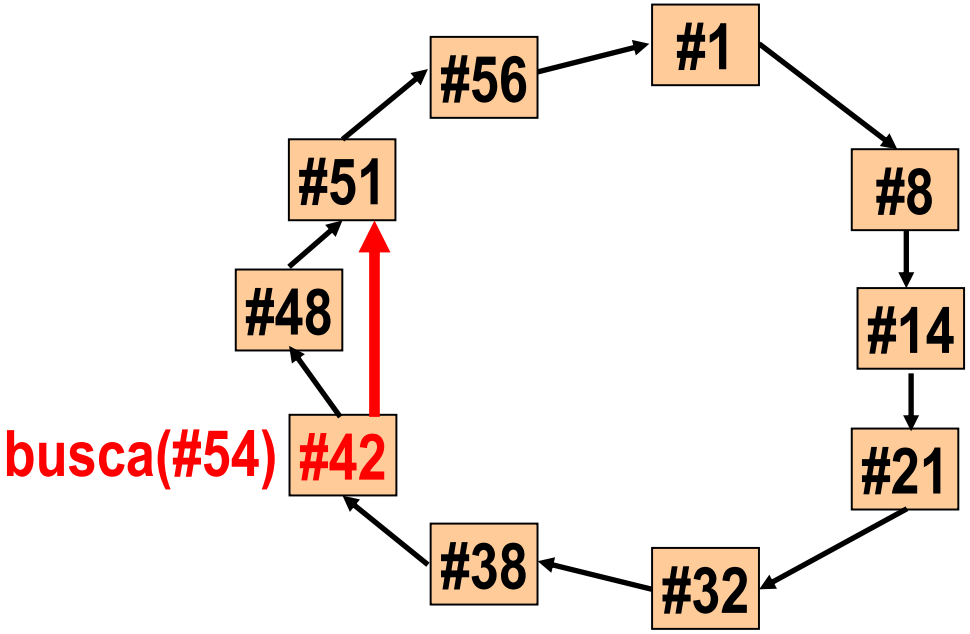
# Búsqueda con *fingers*




Info en nodo

| |
|---|
| IP nodo \| #8 |
| IP predecesor \| #1 |
| IP sucesor \| #14 |
| IP *finger* dist 2 \| #14 |
| IP *finger* dist 4 \| #14 |
| IP *finger* dist 8 \| #21 |
| IP *finger* dist 16 \| #32 |
| IP *finger* dist 32 \| #42 |

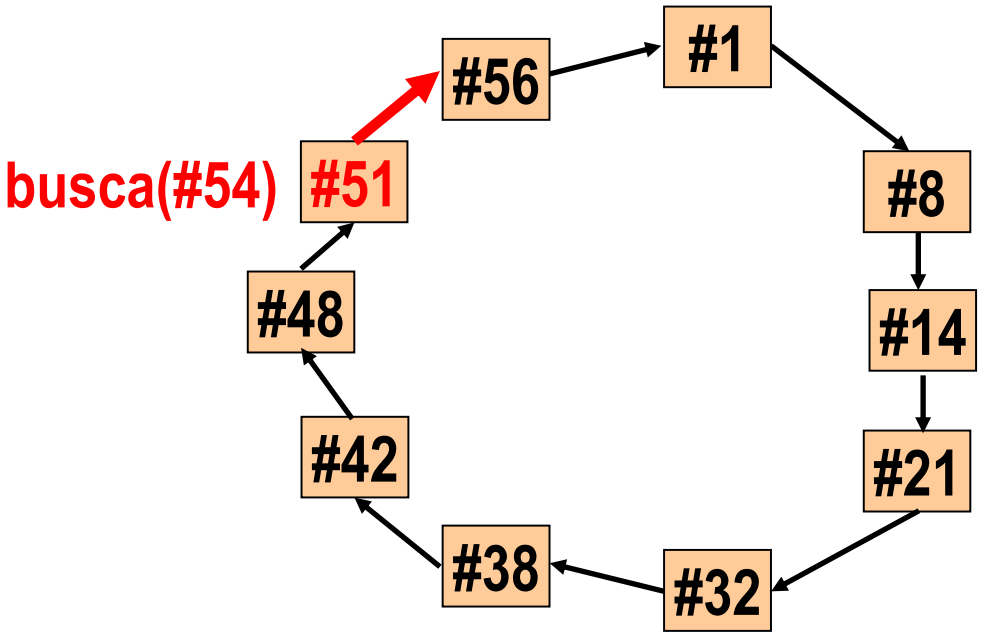
# Búsqueda con *fingers* continúa




Info en nodo

| |
|---|
| IP nodo \| #42 |
| IP predecesor \| #38 |
| IP sucesor \| #48 |
| IP *finger* dist 2 \| #48 |
| IP *finger* dist 4 \| #48 |
| IP *finger* dist 8 \| #51 |
| IP *finger* dist 16 \| #1 |
| IP *finger* dist 32 \| #14 |

# Búsqueda con *fingers* se completa




Info en nodo

| |
|---|
| IP nodo \| #51 |
| IP predecesor \| #48 |
| IP sucesor \| #56 |
| IP *finger* dist 2 \| #56 |
| IP *finger* dist 4 \| #56 |
| IP *finger* dist 8 \| #1 |
| IP *finger* dist 16 \| #8 |
| IP *finger* dist 32 \| #21 |

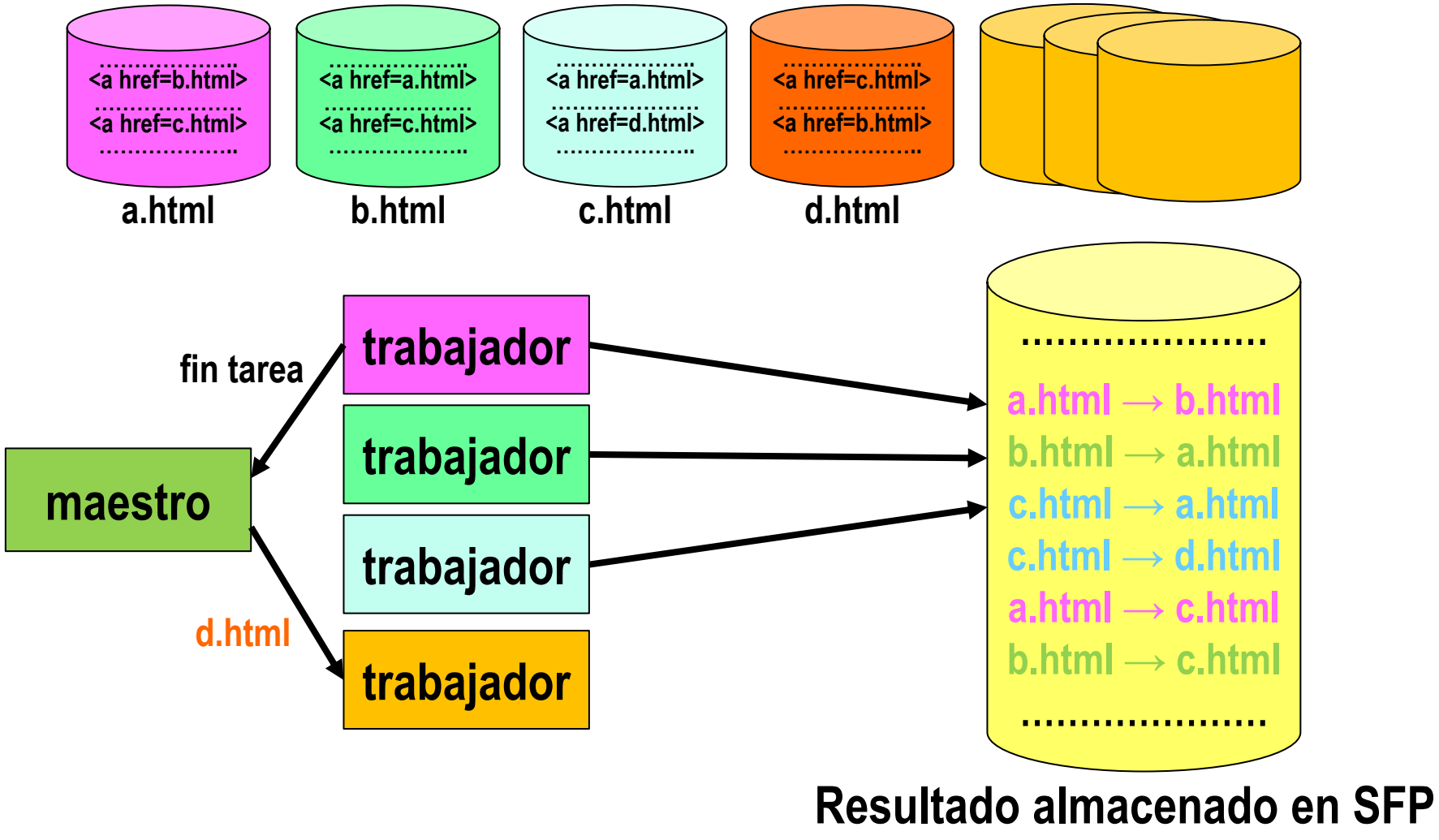
# Arquitecturas para comp. distribuida

- Arquitecturas revisadas hasta ahora: SD de “propósito general”
- Computación distribuida: SD ejecuta una aplicación paralela
- Generalmente, procesando un gran volumen de datos
  - Necesidad de plataforma de almacenamiento paralelo
  - Por ejemplo, *Google File System* (tema de sistema de ficheros)
- Normalmente, aplicación tiene larga duración
  - Probabilidad no despreciable de que nodo se caiga durante la misma
  - Necesidad de técnicas de tolerancia a fallos
- Puede ejecutar en sistema dedicado
- O convivir con otras aplicaciones paralelas
  - Necesidad de un gestor que planifique los recursos
- Modelo más habitual: maestro/trabajador

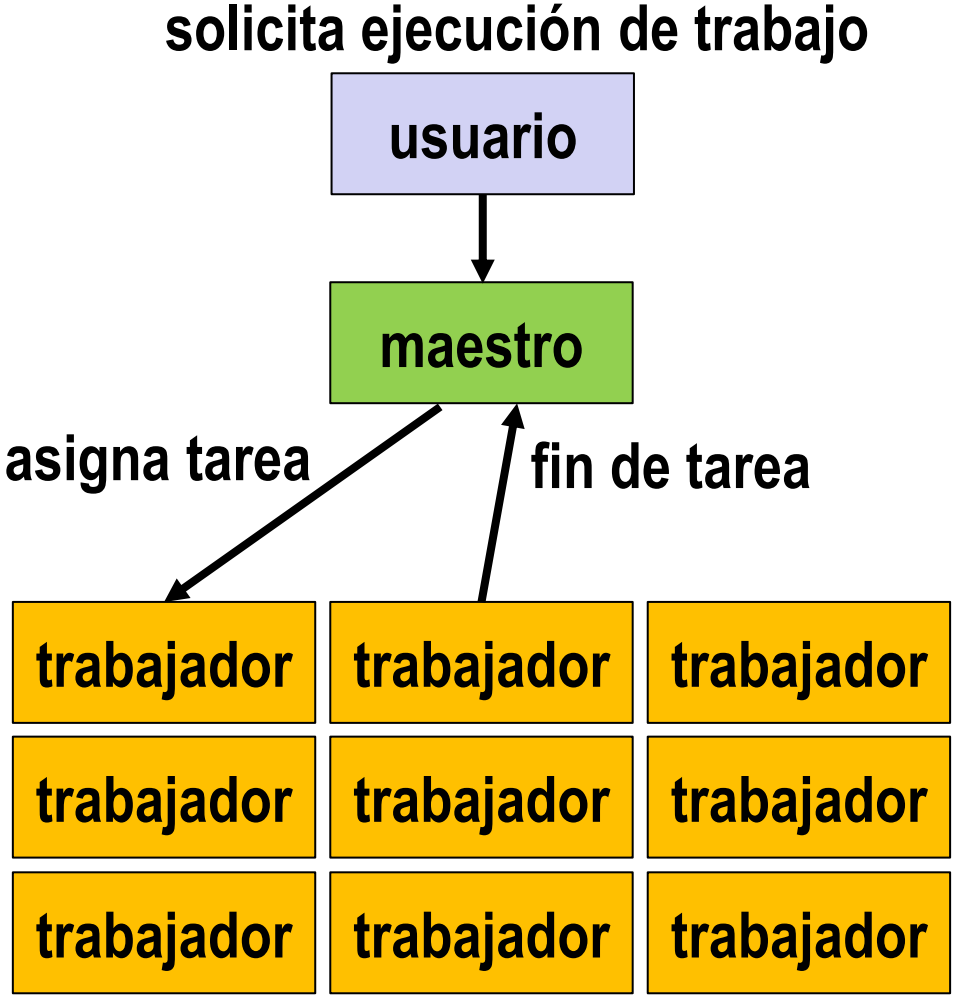
# Arquitectura Maestro/Trabajador

- Trabajo: múltiples tareas que pueden ser de distinto tamaño
- Maestro *M* reparte tareas entre nodos trabajadores *T*
  - Cuando *T* termina tarea, lo notifica a *M*, que le asigna otra
- Si nº tareas >> nº trabajadores → reparto automático de carga
  - Reparto equilibrado ante tareas de distintos tamaños
  - Reparto equilibrado ante nodos de distintas prestaciones
- Tolerancia a fallos:
  - Caída de *T* (frecuente si SD grande y trabajo largo):
    - *M* reasigna a otro trabajador la tarea no completada
    - Pero esta tarea ya se ha ejecutado parcialmente
    - Cuidado con los efectos colaterales (*idempotencia*: 1ª parte del tema)
  - Caída de *M* (muy poco frecuente): Punto crítico de fallo
    - Uso de múltiples maestros: replicación activa o pasiva (1ª parte del tema)

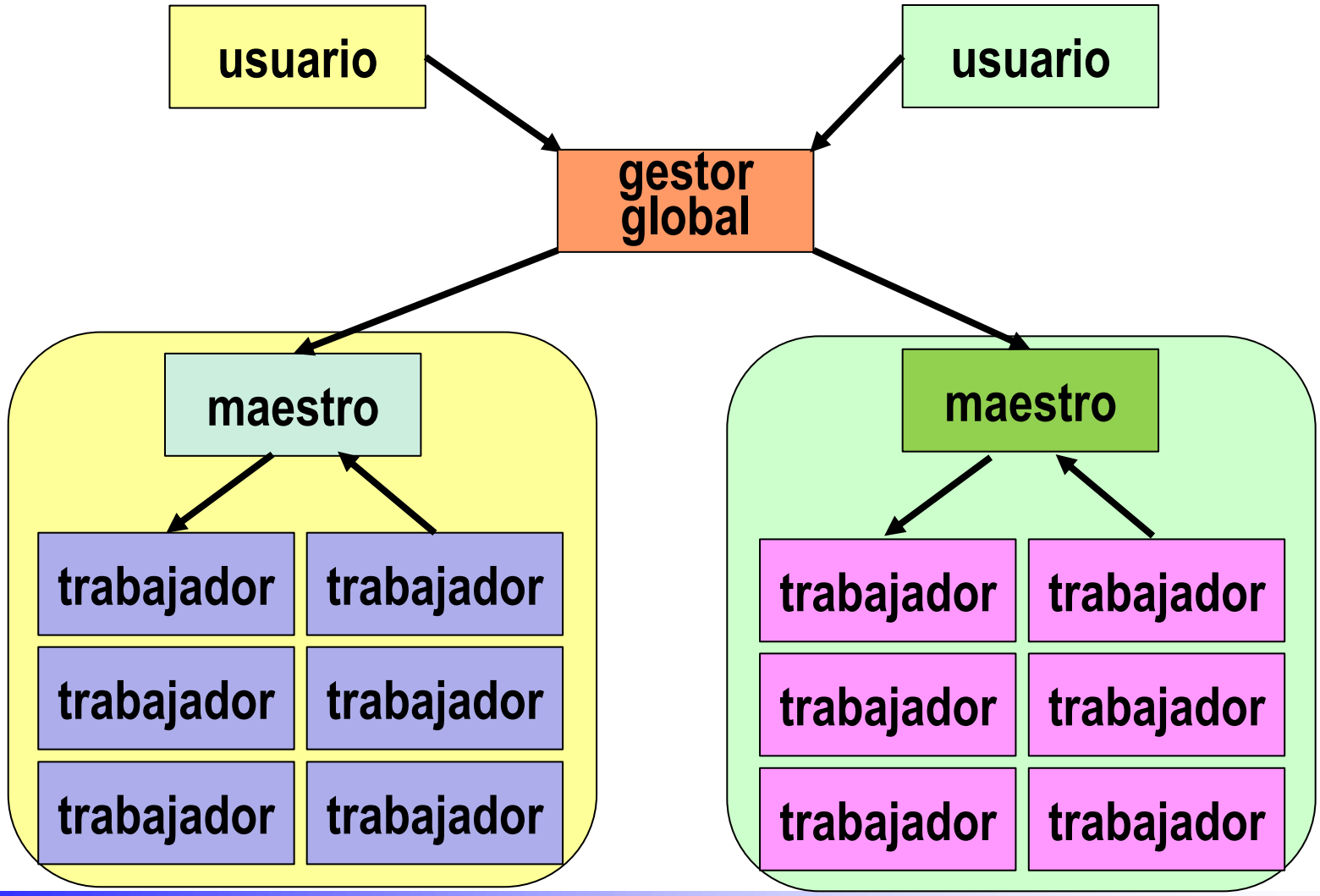
# Ejemplo M/T: grafo de páginas web



# Ejecución M/T en sistema uso dedicado



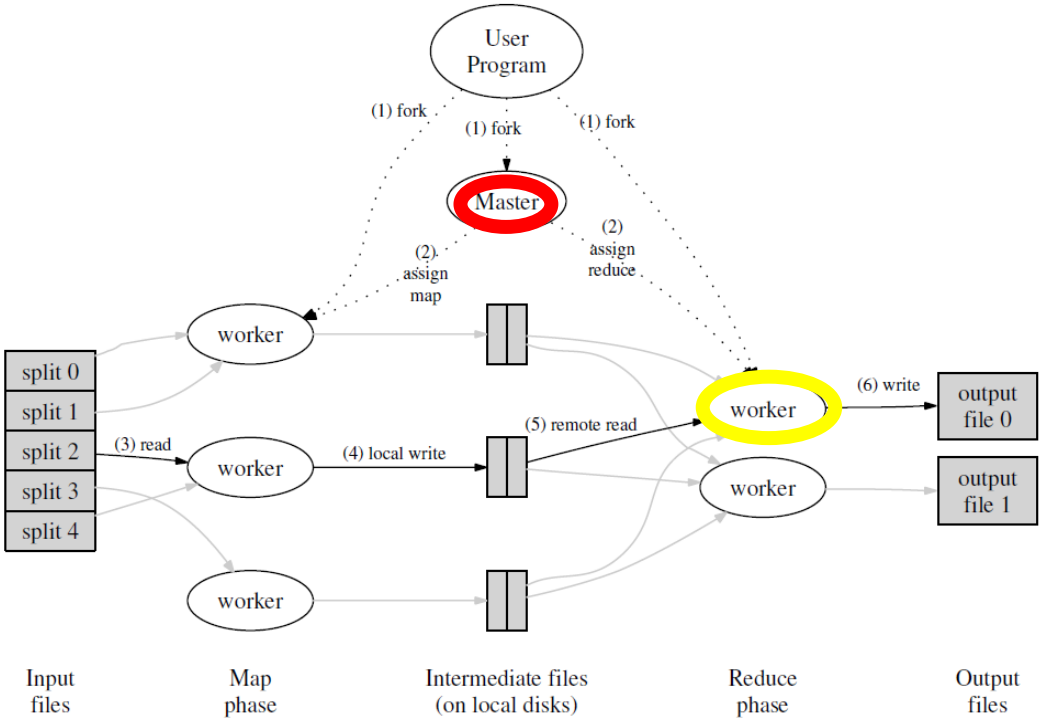
# Ejecución M/T en sistema uso compartido



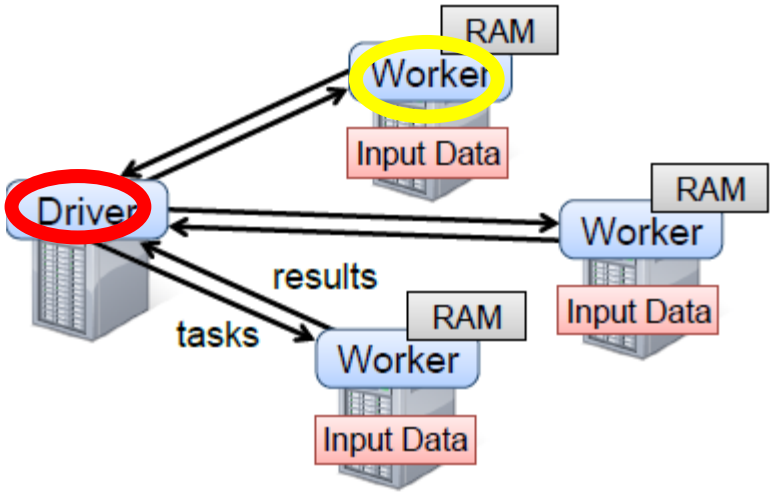
# Algunas herramientas basadas en M/T

- *MapReduce* (Google; versión libre: *Hadoop MapReduce*)
  - Procesamiento *batch* de datos masivos
- *Apache Spark*
  - Mejora *MapReduce* almacenando parte de los datos en memoria
- *Pregel* (Google; versión libre: *Apache Giraph*)
  - Especializado en procesamiento de grafos (p.e. *PageRank*)
- *Tensorflow* (Google)
  - Herramienta para el aprendizaje automático
- Todos ellos basados en la arquitectura maestro/trabajador
- No los estudiamos en esta asignatura:
  - Pero mostramos su esquema **solo** para refrendar su uso del M/T

# M/T en *MapReduce* y en *Spark*

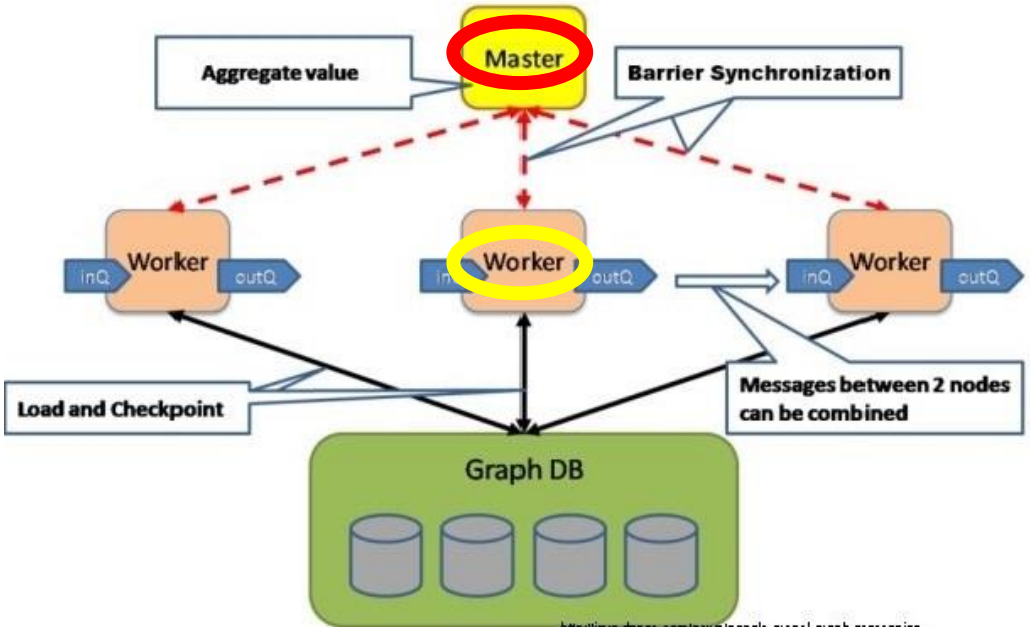



*MapReduce: Simplified Data Processing on Large Clusters*

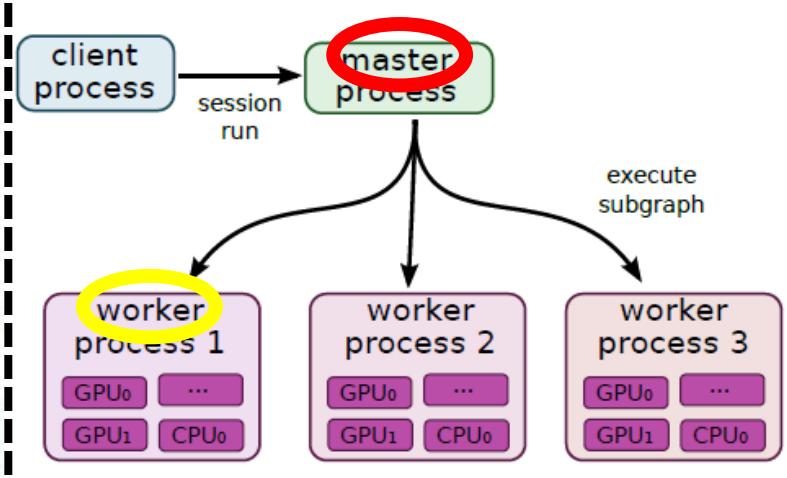



*Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*

# M/T en *Pregel* y en *TensorFlow*




*Pregel: A System for Large-Scale Graph Processing*




*TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*