Universidad de O'Higgins

ICI INSTITUTO DE CIENCIAS DE LA INGENIERÍA

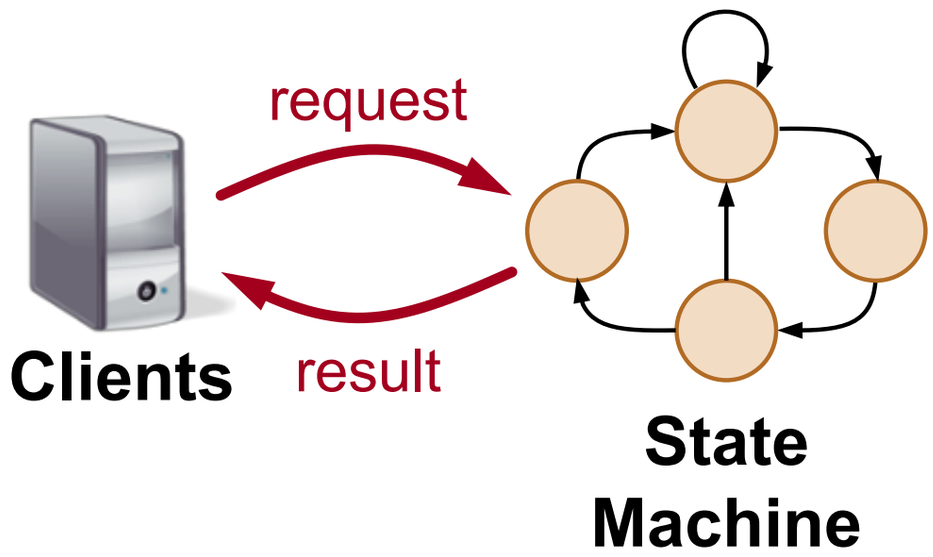
Bases de datos distribuidas II

Alex Di Genova

20/11/2024

Algoritmo de consenso
RAFT

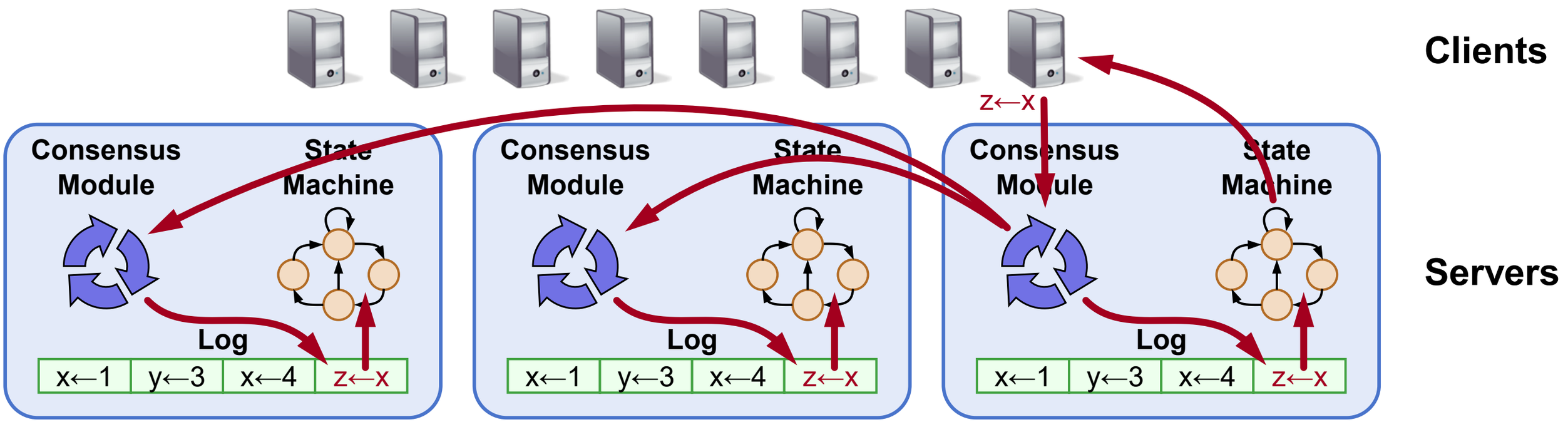
- Consenso
  - Permite a una colección de máquinas trabajar como un grupo coherente.
  - Servicio es robusto, incluso si algunas máquinas fallan.
- Máquinas de estado
  - Responden a estimulos externos.
  - HDFS (name node)



Maquinas de estado replicadas

RAFT

- El “log” replicado garantiza que las máquinas de estado ejecuten los mismos comandos en el mismo orden.
- El módulo de consenso asegura una replicación adecuada del log.
- El sistema funciona siempre que la mayoría de los servidores estén operativos.
- Modelo de error: mensajes demorados/perdidos, parar por falla.



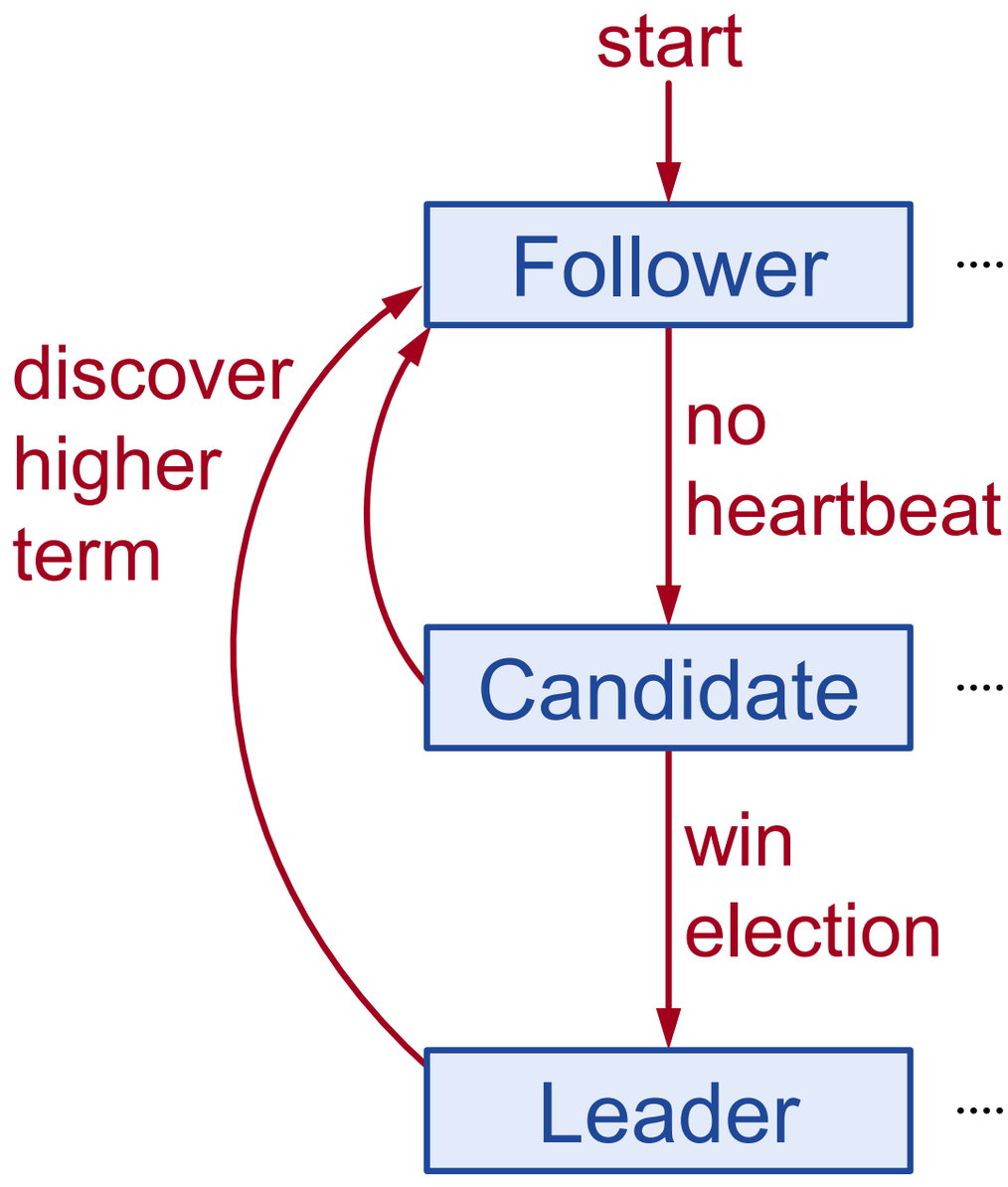
RAFT

Funciones centrales

- **Elección del líder:**
  - Seleccionar un servidor para actuar como líder.
  - Detectar fallas y elegir un nuevo líder.
- **Replicación de log (operación normal):**
  - El líder acepta comandos de los clientes y los añade a su log.
  - El líder replica su log en otros servidores (sobrescribe inconsistencias)
- **Seguridad:**
  - Mantener los logs consistentes.
  - Solo los servidores con registros actualizados pueden convertirse en líder.

Estados de los servidores

RAFT



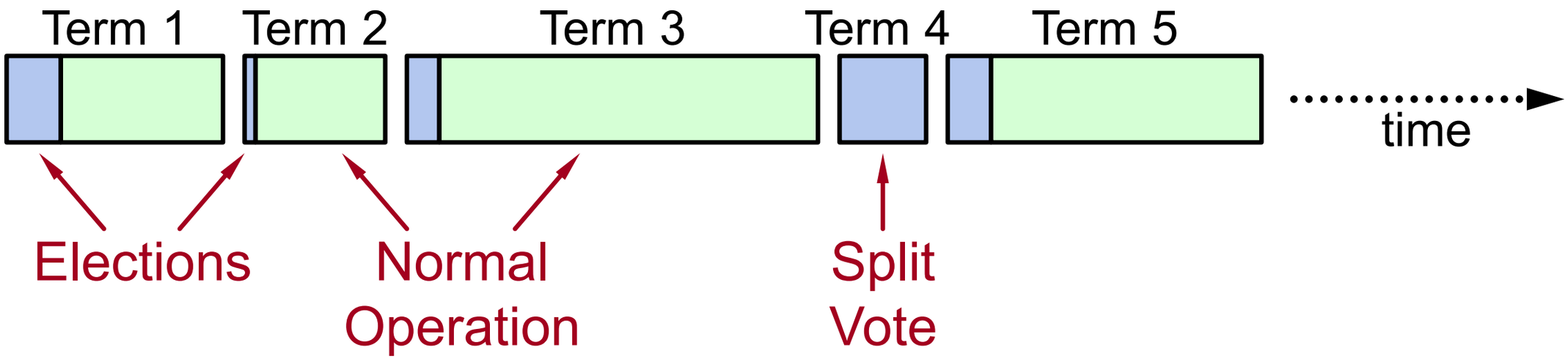
- Pasivo (pero espera mensajes regulares).

- Emite solicitudes de RPC (Remote Procedure Call) **RequestVote** para ser elegido como líder.

- Emite solicitudes de RPC (Remote Procedure Call) **AppendEntries** para:
- Replicar su registro.
- Enviar mensajes para mantener el liderazgo.

Periodos

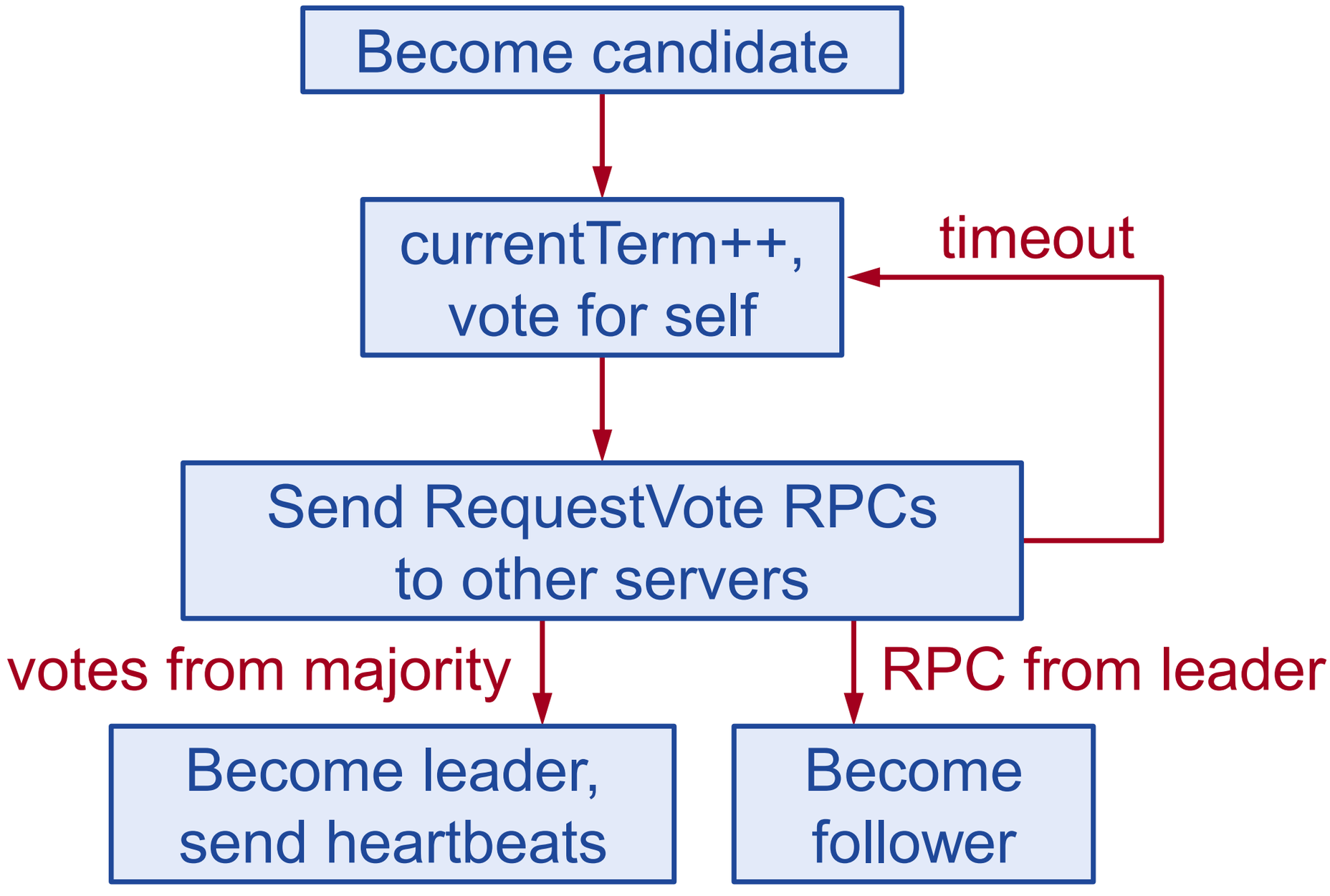
RAFT



- Máximo un líder por periodo.
- Algunos periodos no tienen líder (elección fallida).
- Cada servidor mantiene el valor actual del periodo.
  - Se intercambia en cada RPC.
  - ¿El par/follower tiene un periodo más antiguo? Actualizar periodo, transformarlo en seguidor.
  - ¿El RPC entrante tiene un periodo obsoleto? Responder con un error.
- **Los periodos identifican información obsoleta.**

Elección del líder

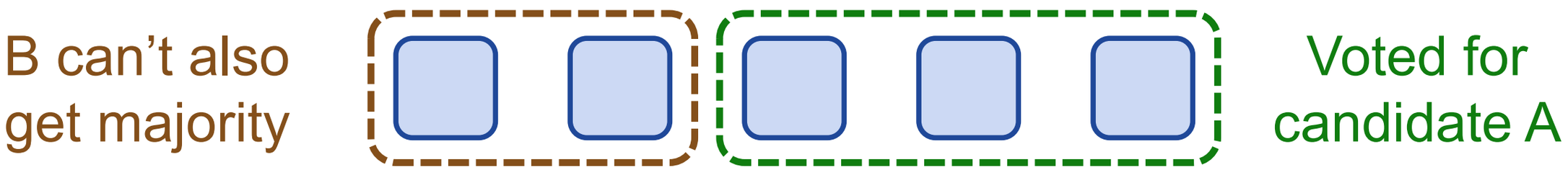
RAFT



Correctitud de la elección

RAFT

- **Seguridad**: permitir como máximo un ganador por periodo.
  - Cada servidor otorga solo un voto por periodo/término (se guarda en disco).
  - Se requiere mayoría para ganar la elección.



- **Vitalidad**: algún candidato debe ganar eventualmente.
  - Elegir tiempos de espera para elecciones de manera aleatoria en [T, 2T] (por ejemplo, 150-300 ms).
  - Normalmente, un servidor agota el tiempo y gana la elección antes de que otros.
  - Funciona bien si T >> tiempo de difusión.

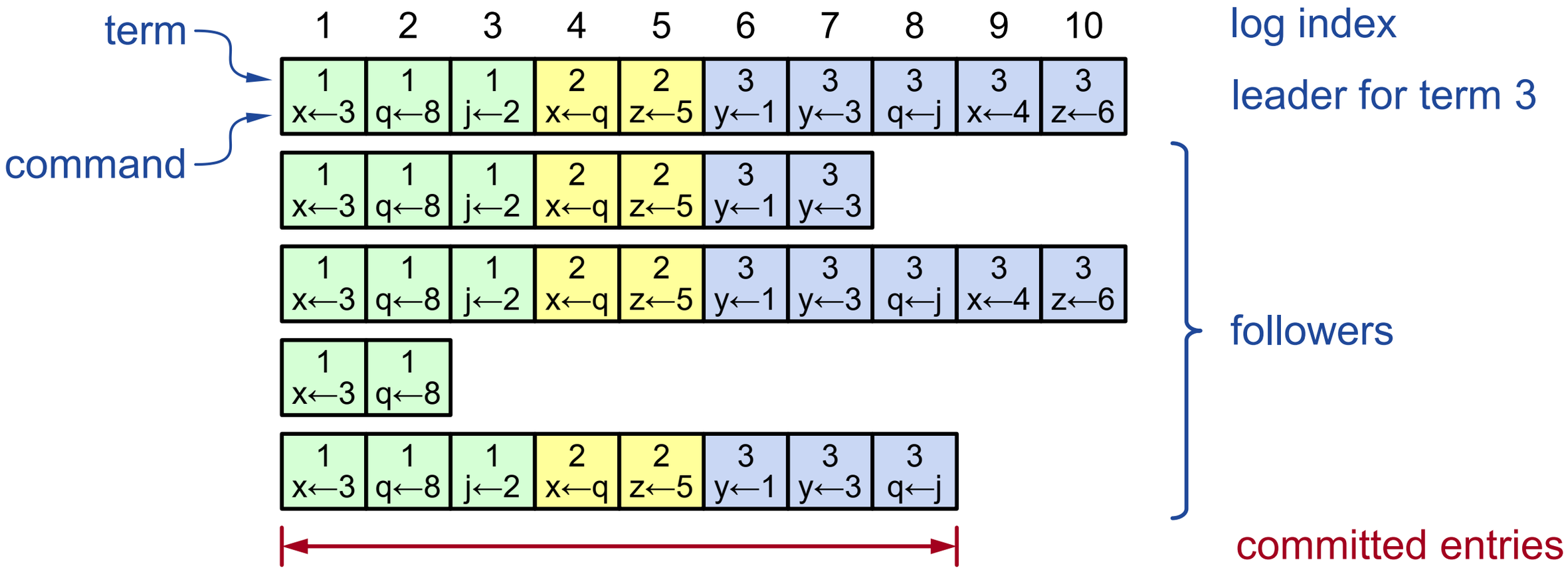
Operación Normal

RAFT

- **El cliente envía un comando al líder.**
- **El líder añade el comando a su registro/log.**
- **El líder envía solicitudes de RPC (AppendEntries) a todos los seguidores.**
- **Una vez que la nueva entrada se ha comprometido:**
    - El líder ejecuta el comando en su máquina de estado y devuelve el resultado al cliente.
    - El líder notifica a los seguidores las entradas comprometidas en RPCs de AppendEntries subsiguientes.
    - Los seguidores ejecutan los comandos comprometidos en sus máquinas de estado.
- **¿Seguidores bloqueados/lentos?**
    - El líder vuelve a intentar solicitudes de RPC (AppendEntries) hasta que tengan éxito.
- **Rendimiento óptimo en el caso común:**
    - Un RPC exitoso a cualquier mayoría de servidores.

Estructura del registro/log

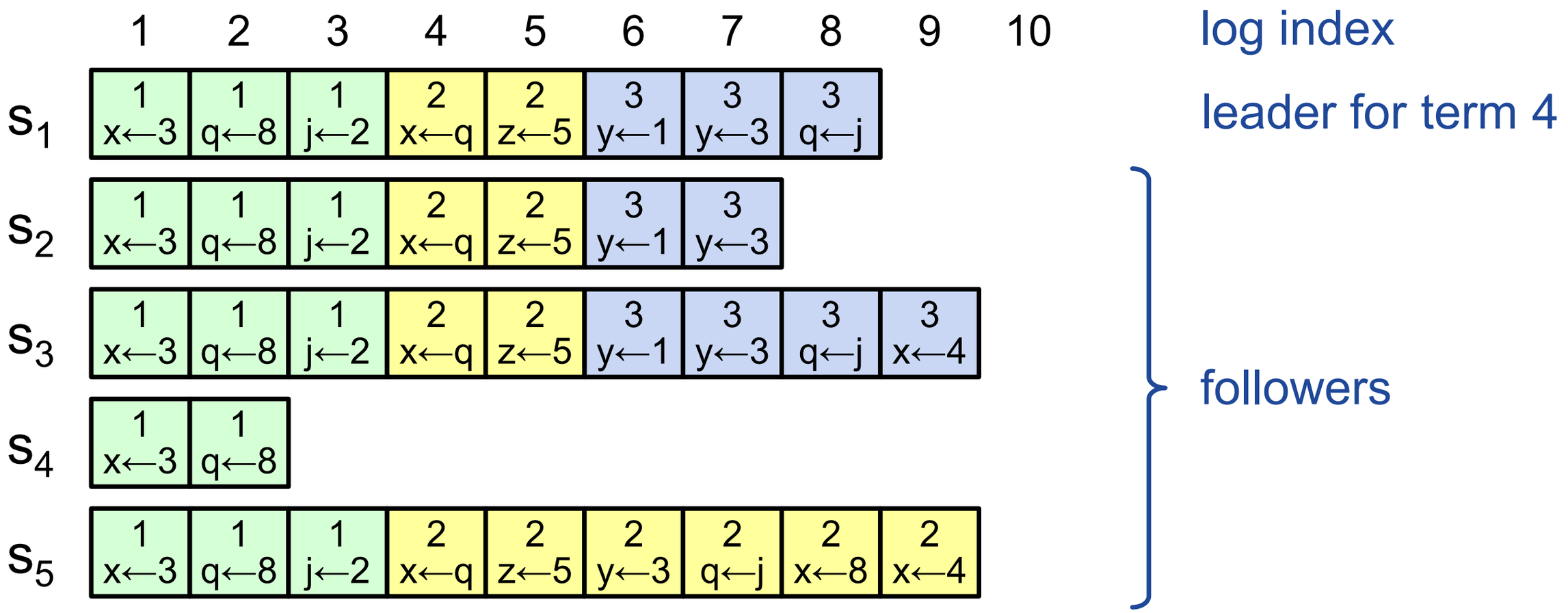
RAFT



- Debe sobrevivir a errores (almacenar en disco).
- Una entrada se considera comprometida si es seguro ejecutarla en las máquinas de estado.
  - Replicada en la mayoría de los servidores por el líder de su periodo.

Inconsistencias de registro/log

RAFT

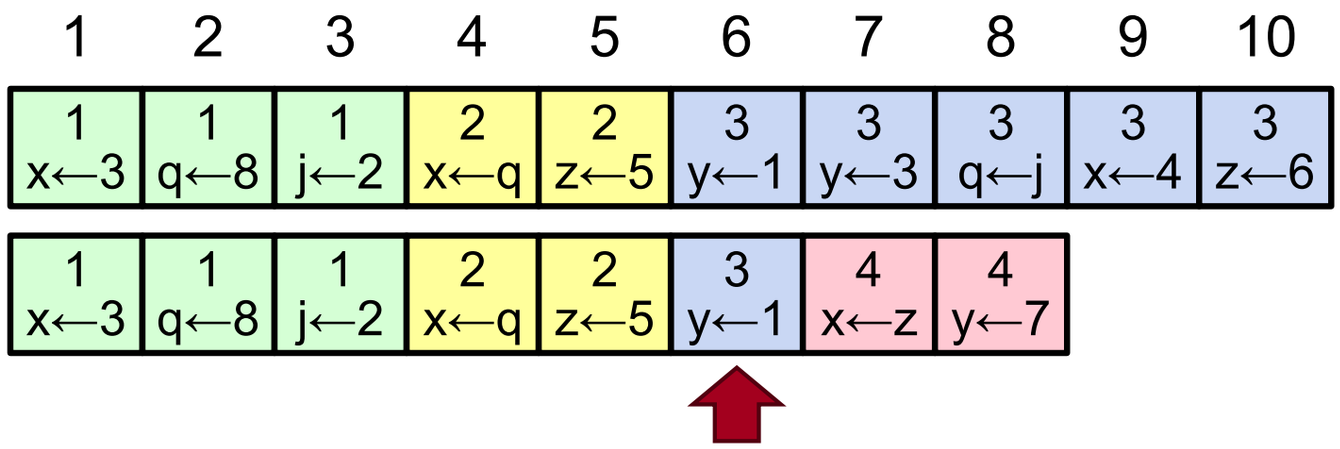


- Raft repara las inconsistencias:
  - El líder asume que su registro es correcto.
  - La operación normal reparará todas las inconsistencias.

Propiedad de alineamineto de Logs

RAFT

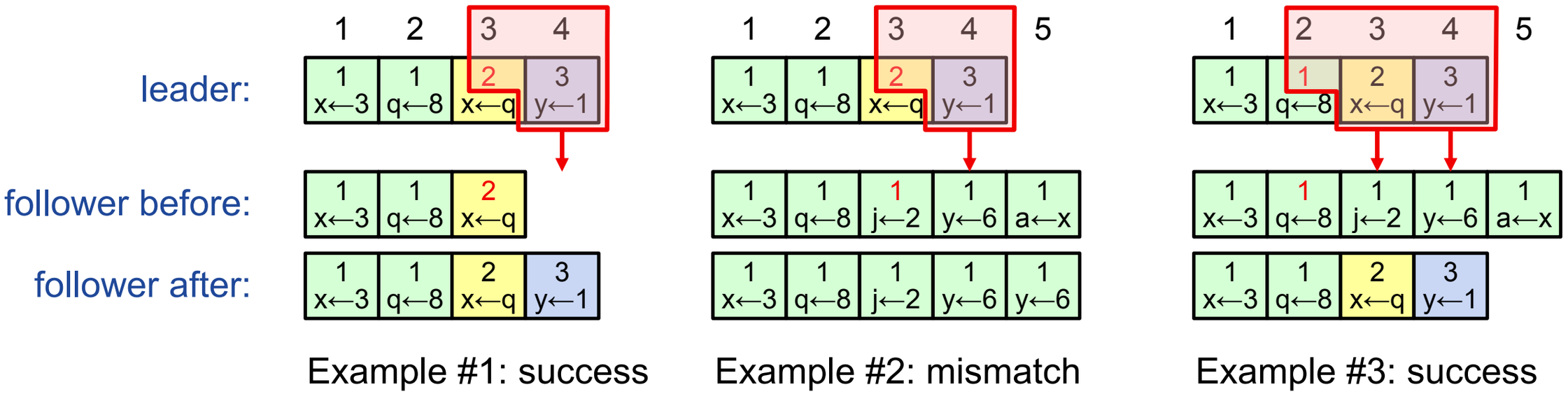
- Objetivo : Consistencia de alto nivel entre logs/registros.
- Si las entradas del log en diferentes servidores tienen el mismo índice y periodo:
  - Almacenan los mismos comandos.
  - Los logs son idénticos en todos las entradas previas.



  - Si una entrada es enviada, todas las entradas anteriores también.

Prueba de consistencia (AppendEntries)

RAFT



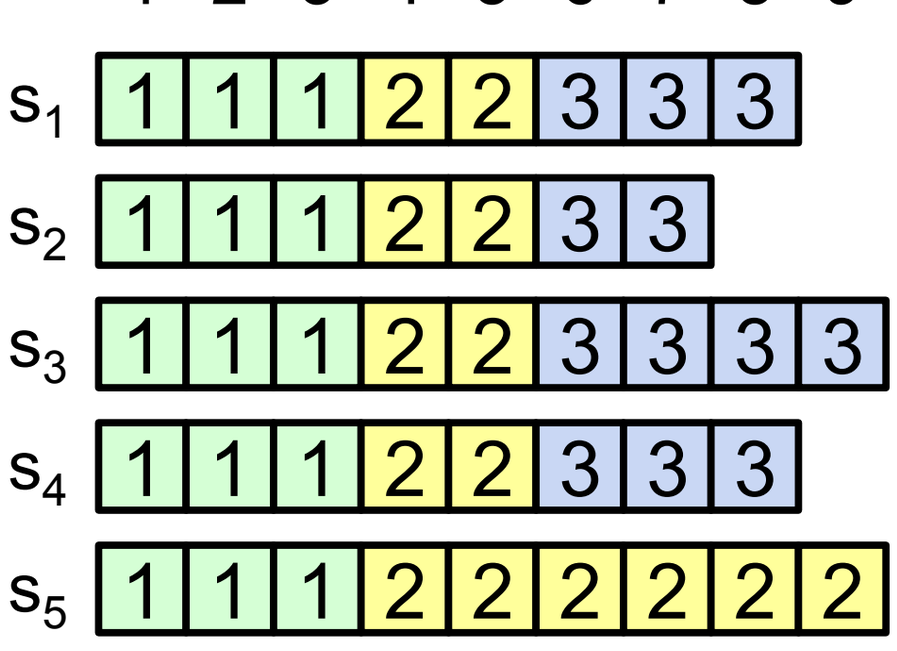
- Las solicitudes de RPC (AppendEntries) incluyen <índice, periodo> de la entrada que precede a la(s) nueva(s).
- El follower debe contener una entrada coincidente; de lo contrario, rechaza la solicitud.
  - El líder reintentará con un índice de registro más bajo.
- Implementa un paso de inducción, asegura la propiedad de coincidencia de registros (Log Matching Property).

Seguridad: Completitud del líder

RAFT

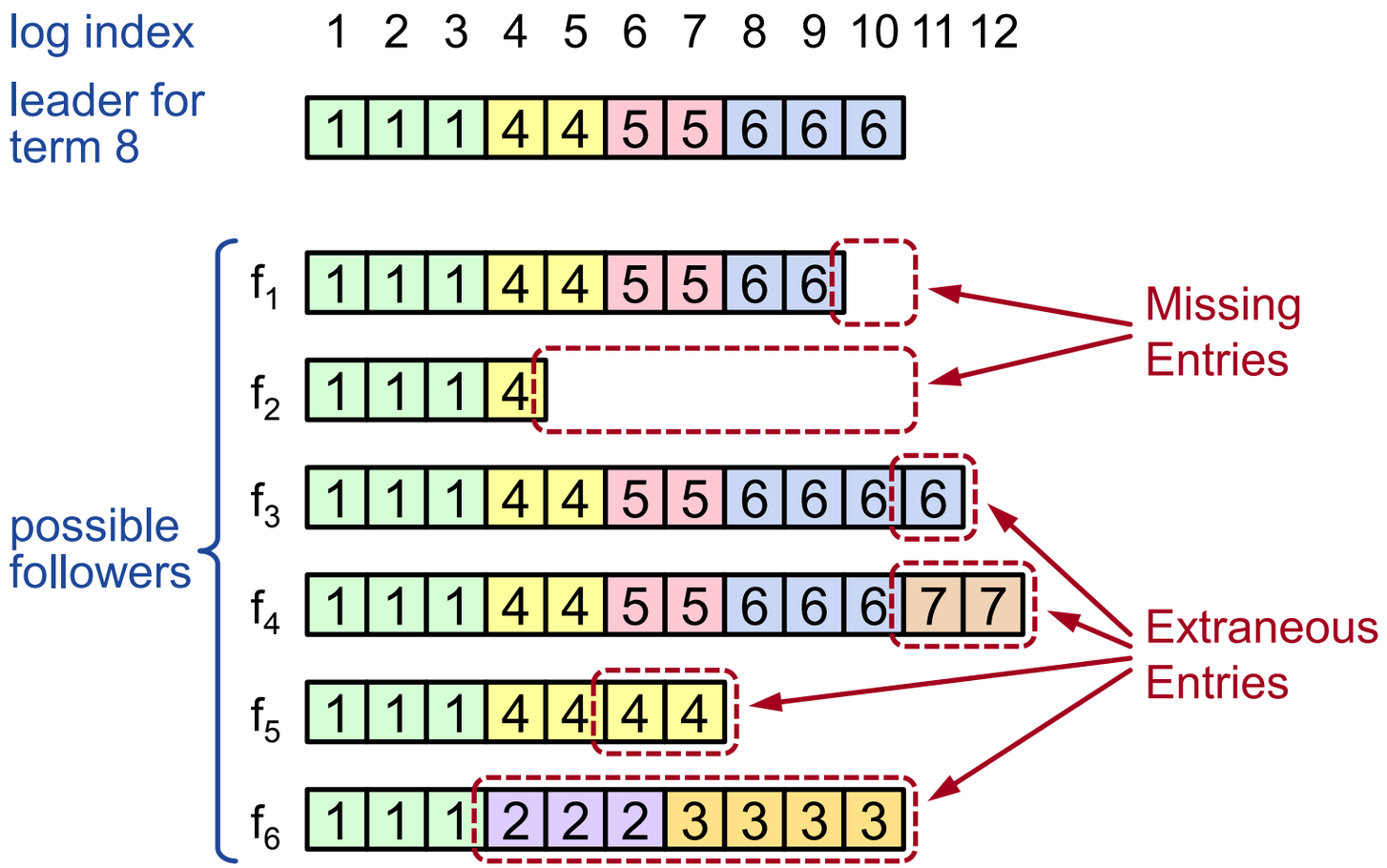
- Una vez que una entrada de registro se ha enviado/comprometido, todos los futuros líderes deben almacenar esa entrada.
- Los servidores con registros incompletos no deben ser elegidos:
  - Los candidatos incluyen el índice y el periodo de la última entrada de registro en las solicitudes de RPC de RequestVote.
  - El servidor que emite el voto lo deniega si su registro está más actualizado.
  - Los registros se ordenan por <lastTerm, lastIndex>.

Leader election for term 4:



Cambios de líder

RAFT



- Los registros pueden ser incosistentes después del cambio de líder.
- Cuando hay un nuevo líder en Raft, no se necesitan pasos especiales:
  - Inicia la operación normal.
  - Con el tiempo, los registros de los seguidores se igualarán al del líder.
- **El registro del líder es considerado como "la verdad".**

Propiedades

RAFT

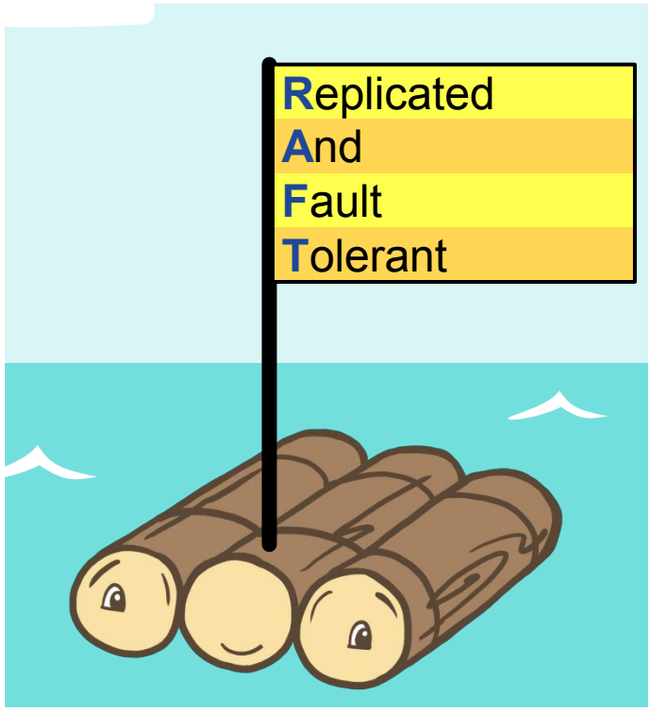
- **Elección segura:** En un periodo dado, se puede elegir como máximo un líder. Esto evita problemas como la elección de múltiples líderes en un mismo período.

- **Líder agrega solamente**: Un líder nunca modifica ni elimina entradas en su registro. Solo agrega nuevas entradas al final.

- **Alineamiento de registros (Log Matching)**: Si dos registros contienen una entrada con el mismo índice y periodo, entonces ambos registros son idénticos en todas las entradas hasta el índice dado.

- **Completitud del líder**: Si una entrada de registro fue enviada/ comprometida, entonces esa entrada estará presente en los registros de todos los líderes futuros. Esto garantiza la coherencia en el largo plazo.

- **Seguridad del estado de máquina**: Si un servidor ha aplicado una entrada de registro en un índice dado a su máquina de estado, ningún otro servidor aplicará nunca una entrada de registro diferente para el mismo índice. Esto asegura la consistencia en las aplicaciones de las entradas del registro.



Resumen

RAFT

- Raft es un algoritmo de consenso utilizado en sistemas distribuidos para garantizar que todos los servidores en un grupo estén de acuerdo sobre el mismo estado.

- Cuando un cliente quiere realizar una operación, envía la solicitud al líder. El líder agrega la operación a su registro y luego replica ese registro en los seguidores para mantenerlos actualizados.

- Raft utiliza un sistema de votación para elegir un líder. Los servidores votan por un líder en intervalos de tiempo aleatorios. El candidato con la mayoría de los votos se convierte en el líder. Para evitar problemas, como la elección de múltiples líderes, Raft garantiza que solo un líder sea elegido en un término dado.

- La seguridad y la consistencia se mantienen mediante el seguimiento de índices y periodos/terminos de registros/logs. Los líderes envían solicitudes de RPC para replicar su registro, y los seguidores verifican si tienen la información más reciente antes de aceptarla. Además, Raft asegura que un líder haya replicado una entrada de registro en la mayoría de los servidores antes de considerarla comprometida/enviada.

- Raft simplifica el consenso en sistemas distribuidos al tener un líder único, un proceso de elección claro y mecanismos para garantizar la consistencia y la seguridad en la replicación del registro.

- **http://thesecretlivesofdata.com/raft/**

RQlite

- rqlite es una base de datos relacional distribuida, ligera y fácil de usar, que utiliza SQLite como motor de almacenamiento.
- rqlite es simple de implementar, operar y sus capacidades de distribución brindan tolerancia a fallas y alta disponibilidad.
- https://github.com/rqlite/rqlite
  - La versión actual es v7.11.0

```
$ curl -XPOST 'localhost:4001/db/execute?pretty&timings' -H "Content-Type: application/json" -d '[
    ["INSERT INTO foo(name, age) VALUES(?, ?)", "fiona", 20]
]'

{
    "results": [
        {
            "last_insert_id": 1,
            "rows_affected": 1,
            "time": 0.00886
        }
    ],
    "time": 0.0152
}

$ curl -G 'localhost:4001/db/query?pretty&timings' --data-urlencode 'q=SELECT * FROM foo'
```

SQLite

- Base de datos relacional sólida, dentro de un solo archivo fuente C
- Con la replicación obtienes confiabilidad.
- Instalación e implementación súper fáciles gracias a Go
- Operación ligera.
- Un sistema distribuido proporciona confiabilidad
  - Los datos se encuentran en varios lugares
  - El cálculo está disponible desde varios lugares.
- Un sistema distribuido proporciona escalabilidad.
  - Los sistemas distribuidos pueden ser más potentes.

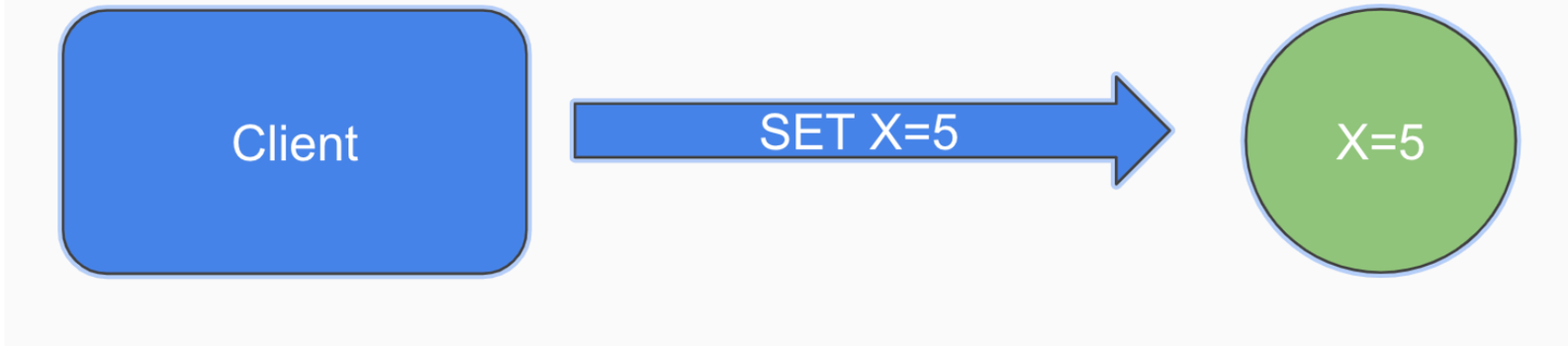
SQLite

eSoftner

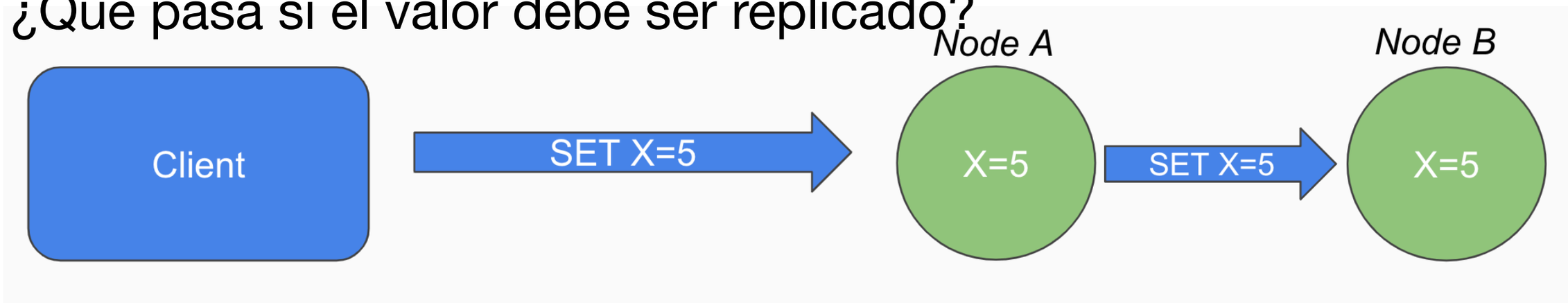
BDD

Replicación

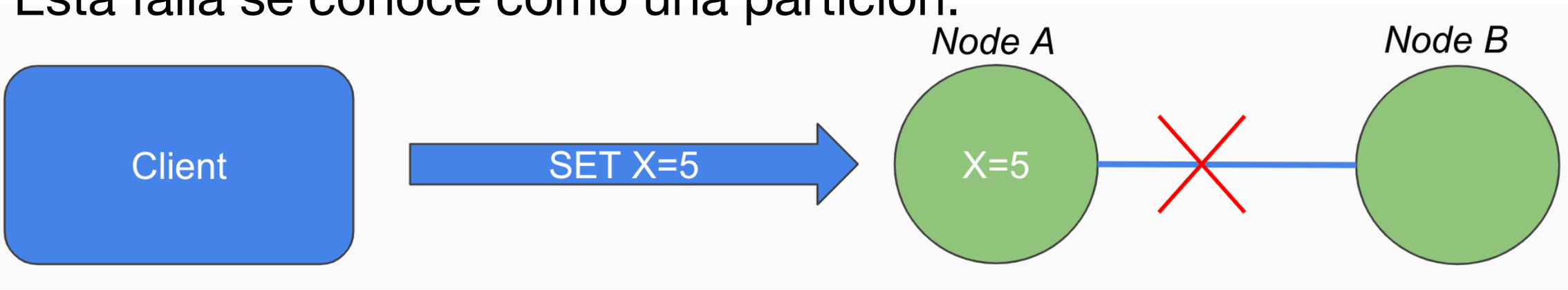
Es fácil establecer el valor de un solo nodo



¿Qué pasa si el valor debe ser replicado?



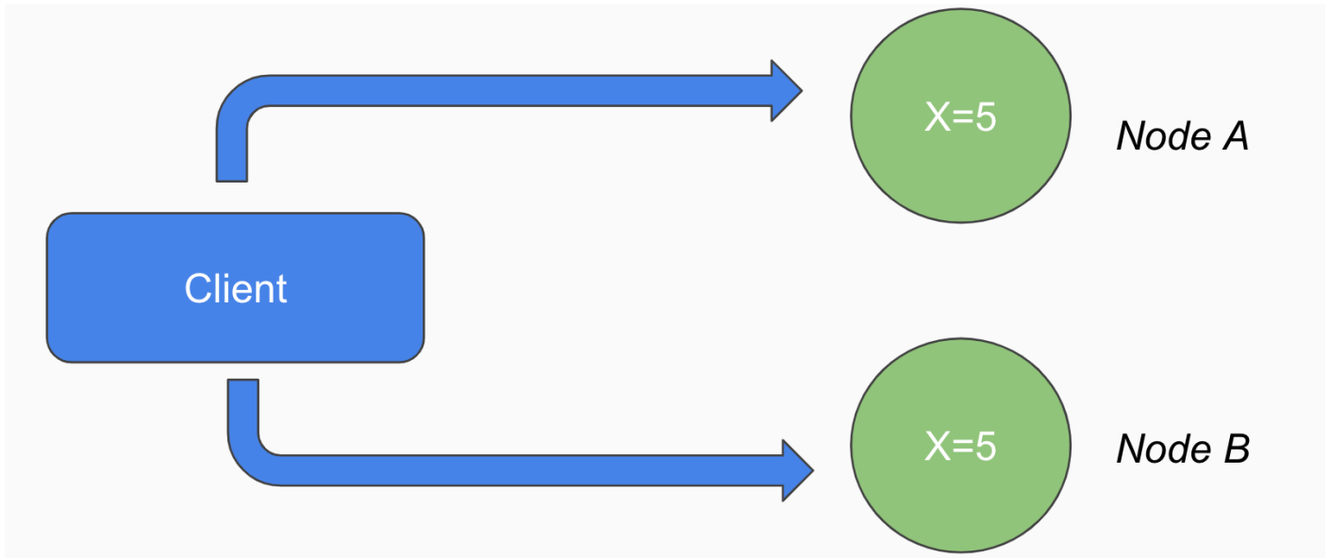
Esta falla se conoce como una partición.



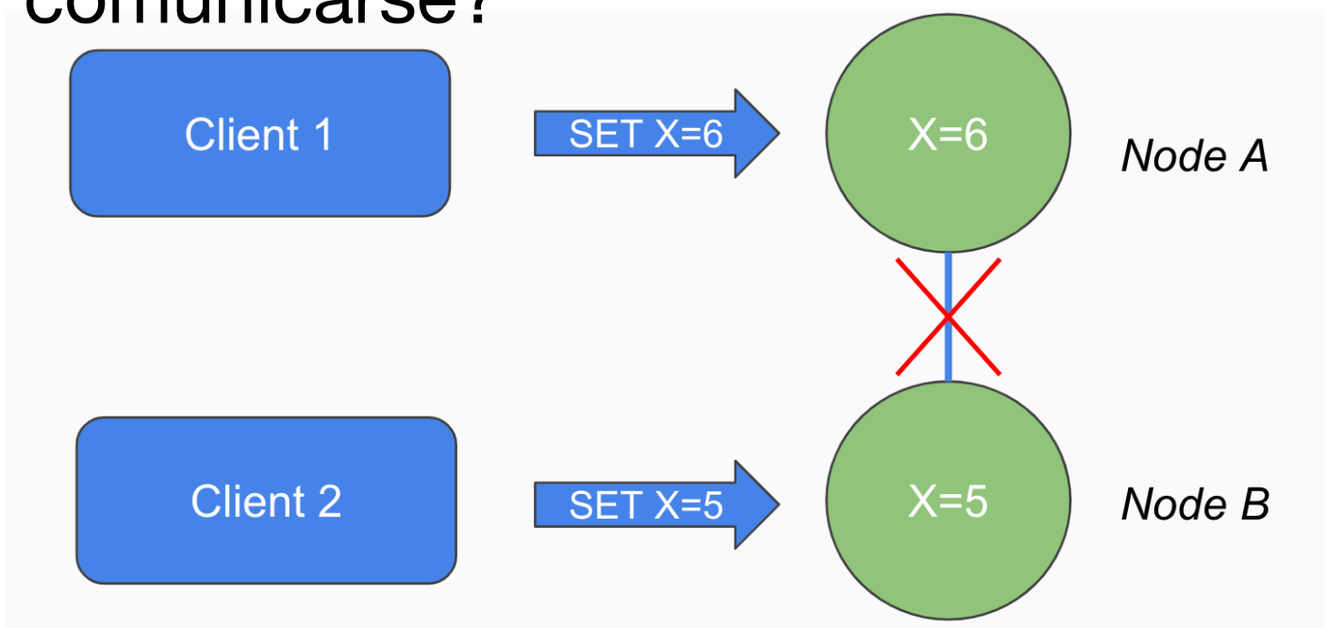
BDD

Replicación

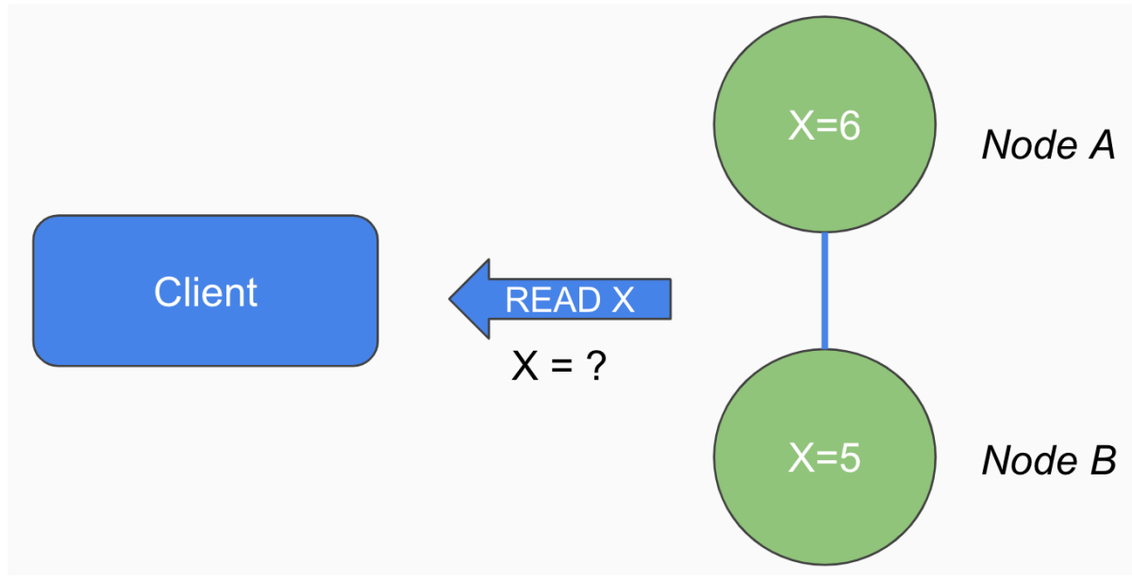
- Replicación de clientes: cada nodo debe admitir cambios de estado



¿Qué pasa si los nodos no pueden comunicarse?



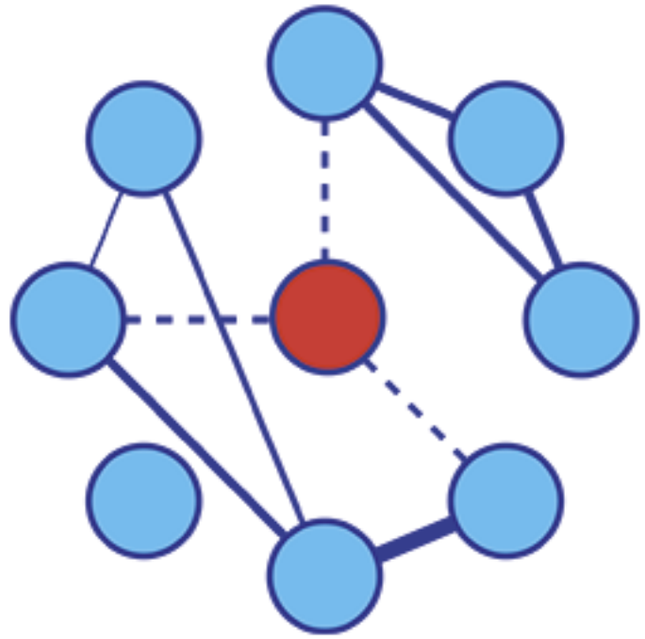
¿Qué valor se debe leer el cliente?



Este problema se conoce como Consenso Distribuido.

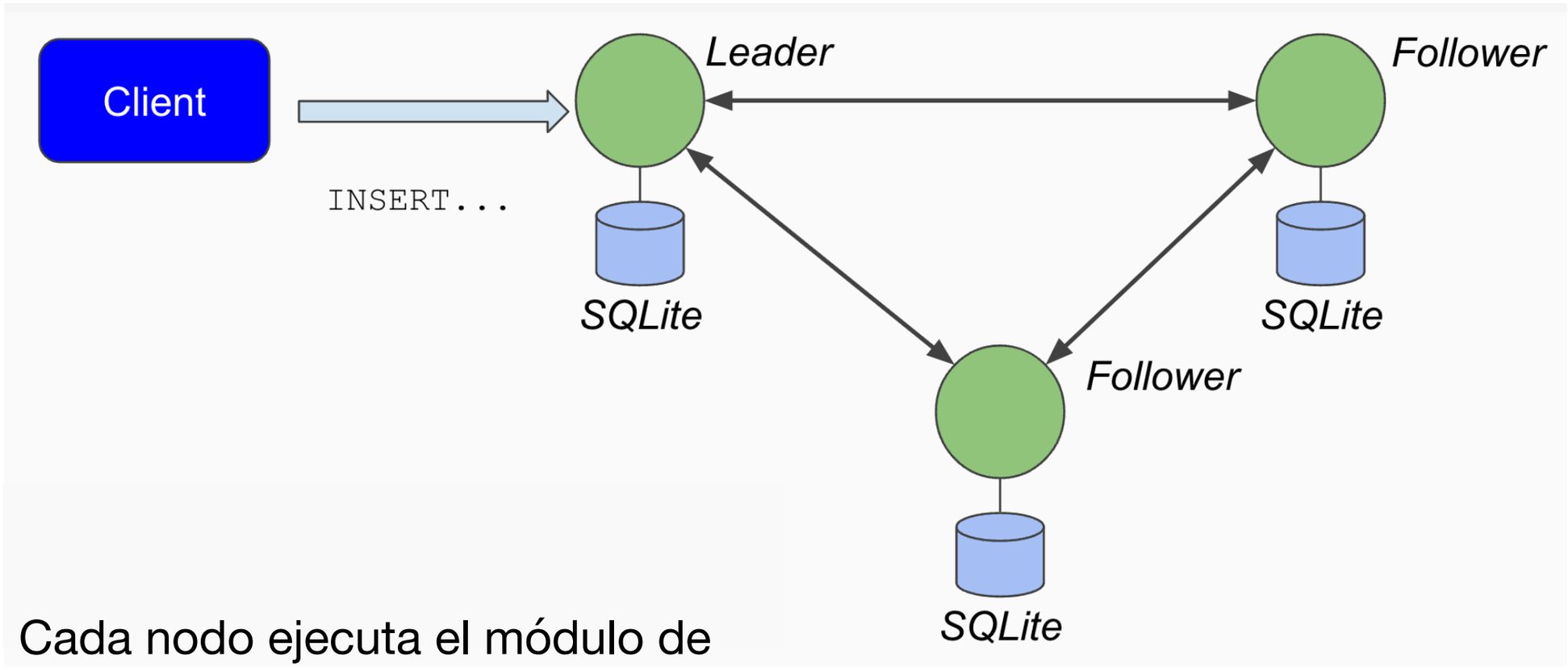
Que es Raft?

- Raft es un protocolo de consenso distribuido.
- Dichos protocolos se utilizan para garantizar que varios nodos diferentes (servidores) siempre estén de acuerdo en un conjunto de valores determinado.
- Nos permite construir un clúster de servidores, de modo que para un quórum de servidores dentro del clúster, cada uno de esos servidores tiene el mismo estado.
- Dentro de rqlite ese estado es una base de datos SQLite.
- http://thesecretlivesofdata.com/raft/

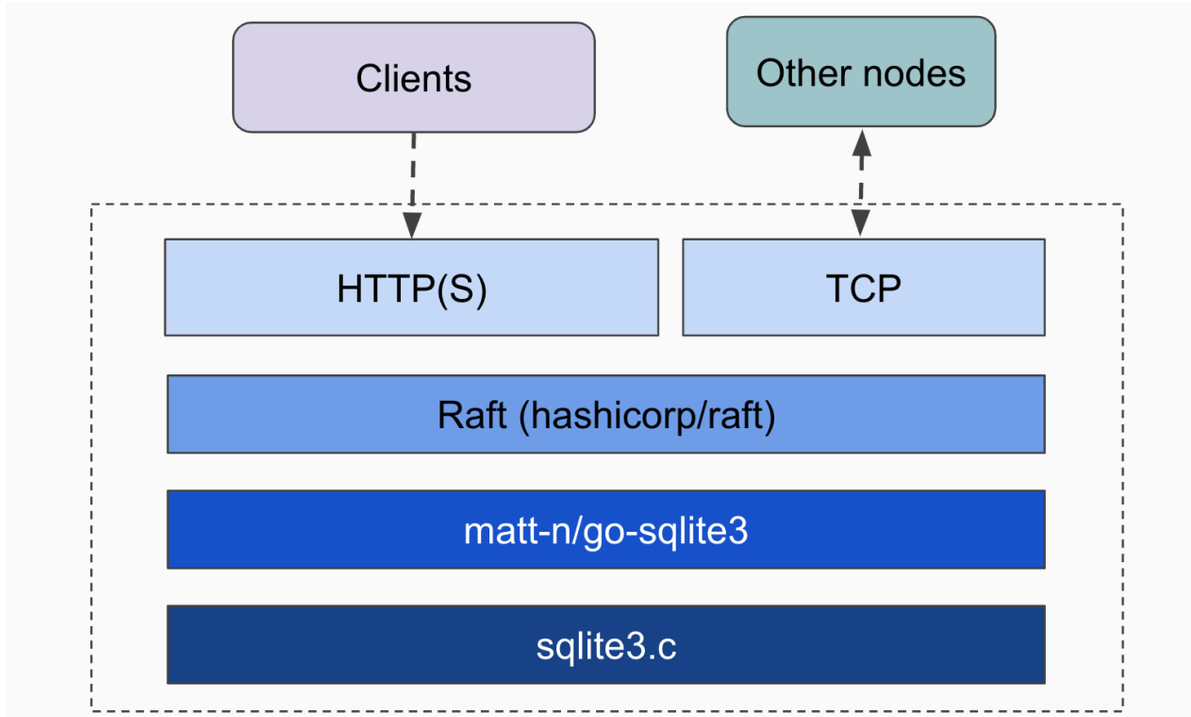


Rqlite

Arquitectura



- Cada nodo ejecuta el módulo de consenso Raft
- Una vez confirmados en los registros de Raft, cada nodo aplica los cambios a su base de datos **SQLite local**.
- Arquitectura a nivel de nodo.



Rqlite

Integración con raft

- La integración con el módulo de consenso de Raft implica implementar cinco funciones clave.

```
Apply(l *raft.Log) interface{} // Aplicar una entrada confirmada a la máquina de
    estado

Snapshot() (raft.FSMSnapshot, error) // Devuelve una snapshot de la máquina de estado.

Restore(rc io.ReadCloser) error // Crear máquina de estado a partir de un snapshot

Persist(sink raft.SnapshotSink) // Escribir el snapshot en almacenamiento persistente

Release()                       //libera el snapshot
```

https://github.com/otoolep/hraftd

Que puede hacer rqlite

- Rqlite es un sistema distribuido liviano y confiable para datos relacionales.
- Podemos usar Rqlite como parte de un sistema más grande, como un almacén central para algunos datos relacionales críticos, sin tener que ejecutar una solución más pesada como MySQL.
- Rqlite también podría ser una forma efectiva de proporcionar una pequeña cantidad de réplicas de lectura de SQLite.

Consultas?

Consultas o comentarios?

Muchas gracias