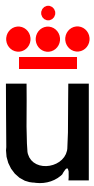
Confiabilidad

Sistemas Distribuidos

Gorka Guardiola

LS, GSYC

24 de noviembre de 2016



Universidad
Rey Juan Carlos

¿Qué es?

- Birman, Kenneth P. Reliable distributed systems: technologies, web services, and applications. Springer, 2005.

¿Qué es?

- Confianza que se puede poner en el servicio que da el sistema
- disponibilidad
- seguridad
- tolerancia a fallos

Disponibilidad

- Medida de cuanto tiempo ininterrumpido se da el servicio
- Tiempo medio entre fallos (MTBF - Mean time between failures)
- Tiempo medio de reparación (MTTR - Mean time to repair)
- Tiempo medio hasta fallo (MTTF - Mean time to failure), si no se puede reparar

Tiempo medio de reparación

- Tiempo de detección
- Tiempo de respuesta - entre detección y diagnóstico
- Tiempo de recuperación

Seguridad

- Importante para la confiabilidad
- Uso no autorizado, acceso no autorizado, datos falsos, etc.
- No se ve en esta asignatura, pero no se puede olvidar

Tolerancia a fallos

- Evitar fallos
- Recuperarse de fallos
- Esconder fallos

Tipos de fallos

- Fallo parada (se para, puede ser silencioso o no)
- Fallo omisión (no responde)
- Fallo temporización (o rendimiento)
- Fallo respuesta (responde mal)
- Fallo bizantino

Recuperarse de fallos/Evitar fallos

- Los sistemas fallan (en especial si la escala es grande)
- Estar preparado
- Reiniciar servidores, mejor que se caigan a que se corrompan
- Introducir errores, incidentes, adrede
- Probar de verdad los backups/mecanismos de recuperación

Introducir errores

- Una idea importante
- No sólo para probar el backup (también), sino todo el rato
- Ejemplo: Chaos monkey de Netflix para AWS

Análisis básico de probabilidad de fallo

- Tenemos N componentes independientes
- Si falla cualquiera, falla el sistema (Luz, red ...)

$$P(nofallo) = \prod_i P(nofallo_i) = \prod_i (1 - P(fallo_i)),$$

OJO, sólo si las $P(fallo_i)$ son muy pequeñas
$P(fallo_i)P(fallo_j) \ldots << P(fallo_i)$,

$$\prod_i (1 - P(fallo_i)) = 1 - P(fallo_1) - P(fallo_2) \cdots + P(fallo_1)P(fallo_2)$$
$$+ P(fallo_1)P(fallo_3) \ldots \approx 1 - \sum_i P(fallo_i).$$

Si la probabilidad de fallo es muy pequeña, es aproximadamente aditiva.

Análisis básico de probabilidad de fallo, ejemplo

Tenemos 2 componentes independientes,

$$
\begin{aligned}
P(fallo_1) &= 0{,}001, \\
P(fallo_2) &= 0{,}001, \\
P(nofallo) &= \prod_1^2 (1 - P(fallo_i)) \\
&= 1 - P(fallo_1) - P(fallo_2) + P(fallo_1)P(fallo_2).
\end{aligned}
$$

Pero tenemos que $P(fallo_1)P(fallo_2) = 1E-6 << P(fallo_1) = 1E-3$,

$$
\prod_1^2 (1 - P(fallo_i)) \approx 1 - 2E - 3 = 0{,}998.
$$

Análisis básico de probabilidad de fallo

- Lo anterior no es cierto si
- Hay muchos componentes
- La probabilidad no es muy pequeña
- ¿Qué sucede en ese caso?

Redundancia Modular Triple (TMR) O N-modular

- Replicación de los sistemas (ya hemos visto maestro-esclavo)
- Cuando hay pocos fallos, maestro-esclavo
- Muchos fallos, replicación, TMR
- Para esconder fallos
- Se tienen N módulos que votan el resultado
- Al menos tres
- Ojo, sistema de votado también replicado o protegido
- También se puede votar con un algoritmo distribuido de consenso (Raft, Paxos)

Redundancia Modular Triple (TMR)

- Tenemos N servidores independientes
- Probabilidad de fallo independiente (¿es esto cierto, diferente HW, OS?)
- $P(fallo) = \prod_i P(fallo_i)$
- Cada servidor que añado la hace multiplicativamente más pequeña
- . . . y cuesta dinero y recursos

Redundancia Modular Triple (TMR)

- Ojo, sistema de votado también replicado o protegido
- También se puede votar con un algoritmo distribuido de consenso (Paxos, Raft)

Complejidad

- Algorítmica (local)
- Máxima (peor caso), medio
- Número de rondas (latencia, $Algorit \times N$)
- Número de mensajes (throughput, ancho de banda)

Paxos

- Consenso distribuido
- N servidores deciden un valor
- Lamport, Leslie. “Paxos made simple” ACM SIGACT News 32.4 (2001): 18-25.
- Robbert van Renesse, “Paxos Made Moderately Complex”, Cornell University, (2011).

Paxos

- Paxos es complicado, poco detalle
- Sólo aprenden un valor, quiero un log

Paxos: single-decree, multi-decree

- El protocolo tradicional, single-decree (o synod) decid un valor
- Si quiero un log distribuido
- Multi-decree más complicado
- Hay N versiones.

Raft

- Similar a Paxos (multi-decree)
- Diseñado alrededor de un log distribuido
- Combina el log distribuido con un sistema de elección del líder (similar al algoritmo del dictador)
- Mucho más sencillo, bien especificado
- Basado en timeouts y aleatorización (no totalmente asíncrono)
- Creado por Diego Ongaro y John Ousterhout (es la tesis doctoral del primero)

Raft

- Similar a Paxos
- Procesadores, velocidad arbitraria acotada, timeout, parcialmente síncrono con detección de fallos
- Errores de fallo-recuperación
- Red, mensajes a cualquier procesador
- Pérdidas, reordenación, duplicación
- No hay corrupción (no hay fallos bizantinos en general)

Propiedades de Raft

- No trivialidad: al log sólo van valores propuestos
- Consistencia: Sólo va un valor para cada índice
- Viveza: Si se propone un valor, tarde o temprano va al log de una mayoría de miembros

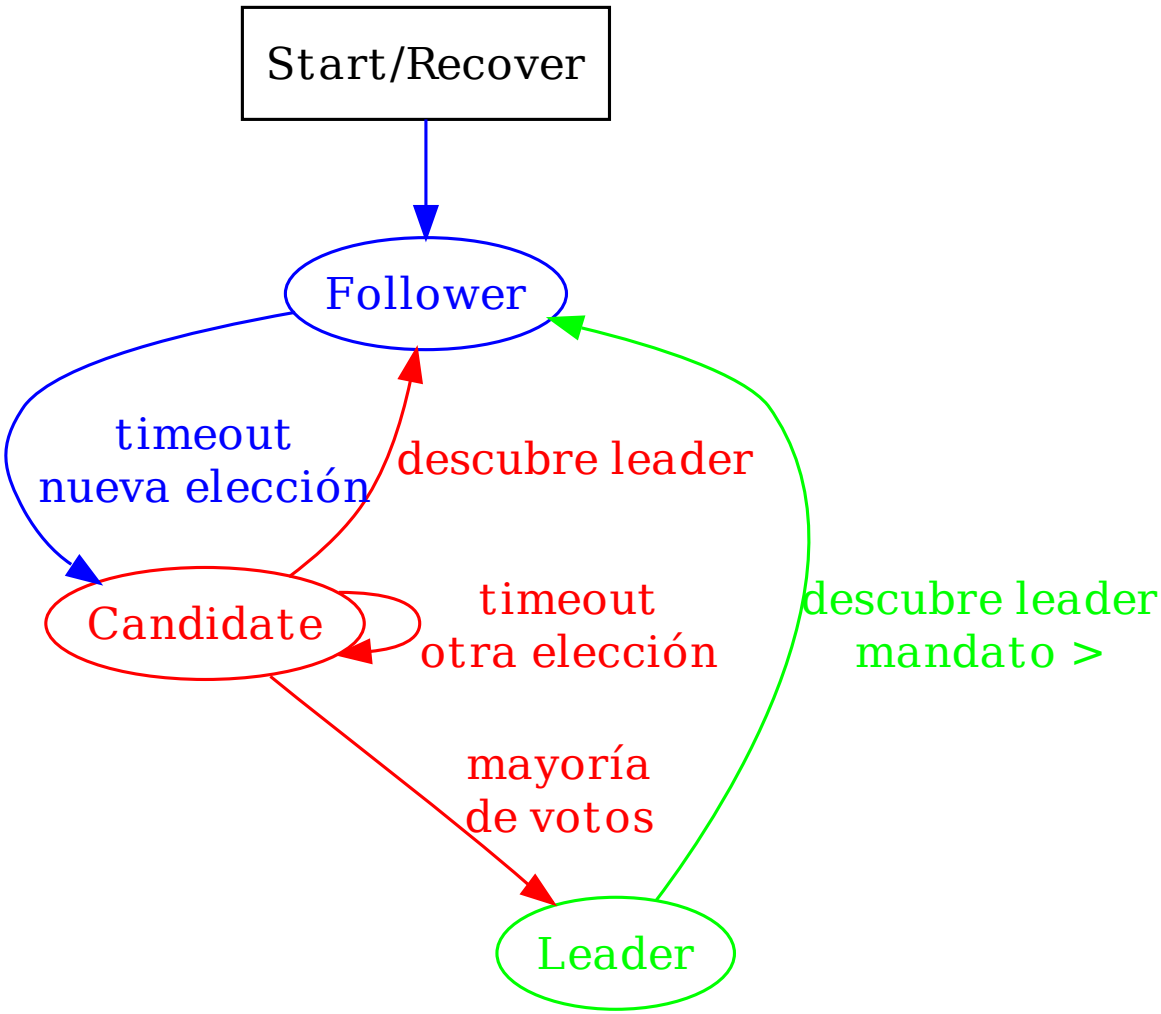
Raft

- http://thesecretlivesofdata.com/raft

Roles en Raft

- `Follower`
- `Candidate`
- `Leader` (tiene un número de mandato, Term)
- Caso normal, $N - 1$ followers y un leader
- Con timeout aleatorio si no hay leader se presentan

Roles en Raft



Fase inicial de Raft (i)

- Empieza con todos como `Followers`
- Uno da timeout (no hay lider, no le llegan mensajes)
- Se pone como candidato, `Candidate` (incrementa el mandato, Term)

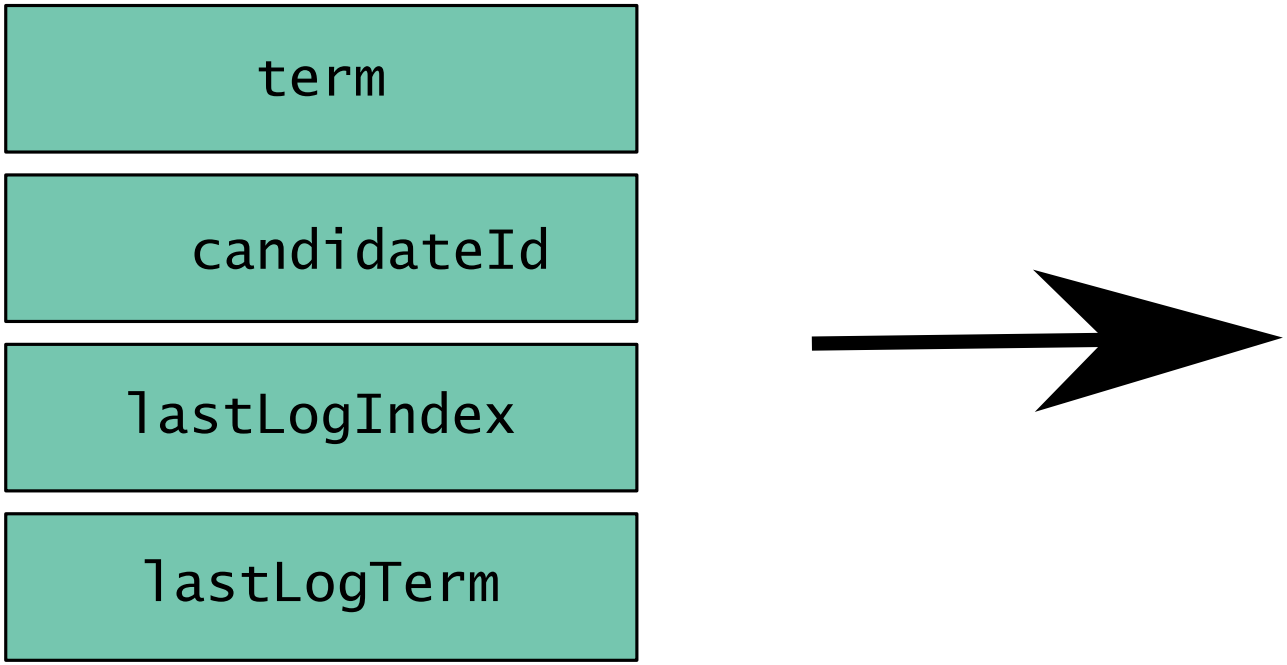
Mensajes de Raft

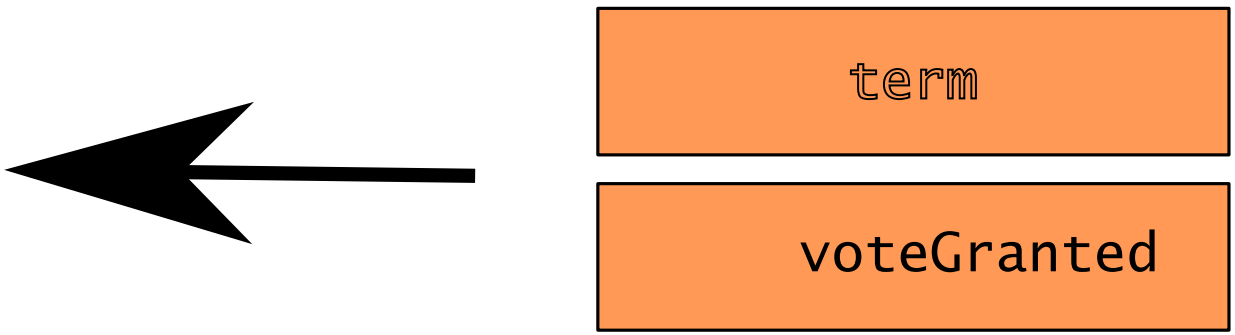
- Son RPCS, cuatro mensajes (dos peticiones y sus respuestas)
- `RequestVote`: para pedir el voto (respuesta, sí o no)
- `AppendEntries`: dos propósitos, mandar un valor a log, heartbeat

Fase inicial de Raft (ii)

- El candidato pide votos (vota por sí mismo), `RequestVote` si saca mayoría absoluta, gana, es el líder
- El nuevo líder comienza un nuevo mandato `Term`
- Cada `Follower` puede votar sólo a uno
- Si hay otro Candidate a la vez (da timeout), no sacará mayoría absoluta, verá un mensaje con un Term mayor (`AppendEntries` o un nuevo `RequestVote`)

Fase inicial de Raft: requestVotes





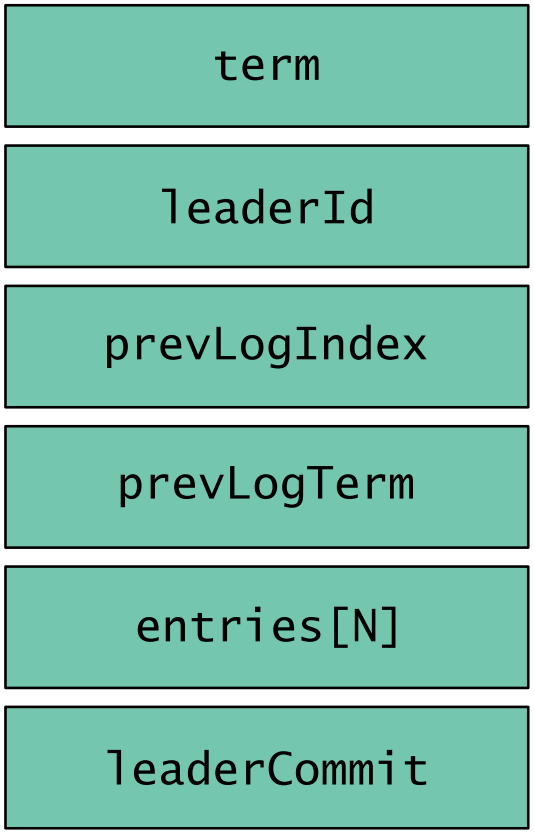
Fase normal de Raft

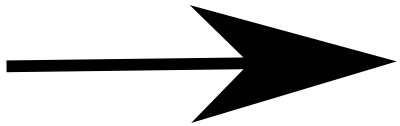
- Tengo un Leader
- Periódicamente manda un `AppendEntries` para avisar de que está vivo
- El `AppendEntries` puede tener valores para el log de los Followers

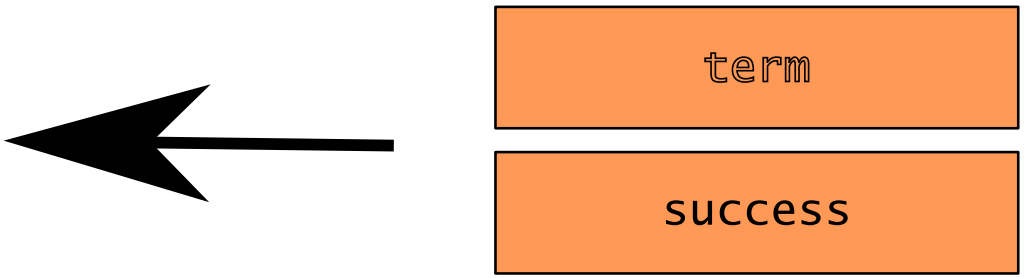
Fase normal de Raft

- De un valor no se hace commit en el log
  - Del servidor hasta que no han contestado una mayoría absoluta de Followers
  - Del cliente hasta que no avanza el índice del qué ha hecho commit el servidor (un campo de `AppendEntries`)
- Cuando se hace commit, ese valor ya no debería cambiar, se avanza la máquina de estados

Fase normal de Raft, AppendEntries



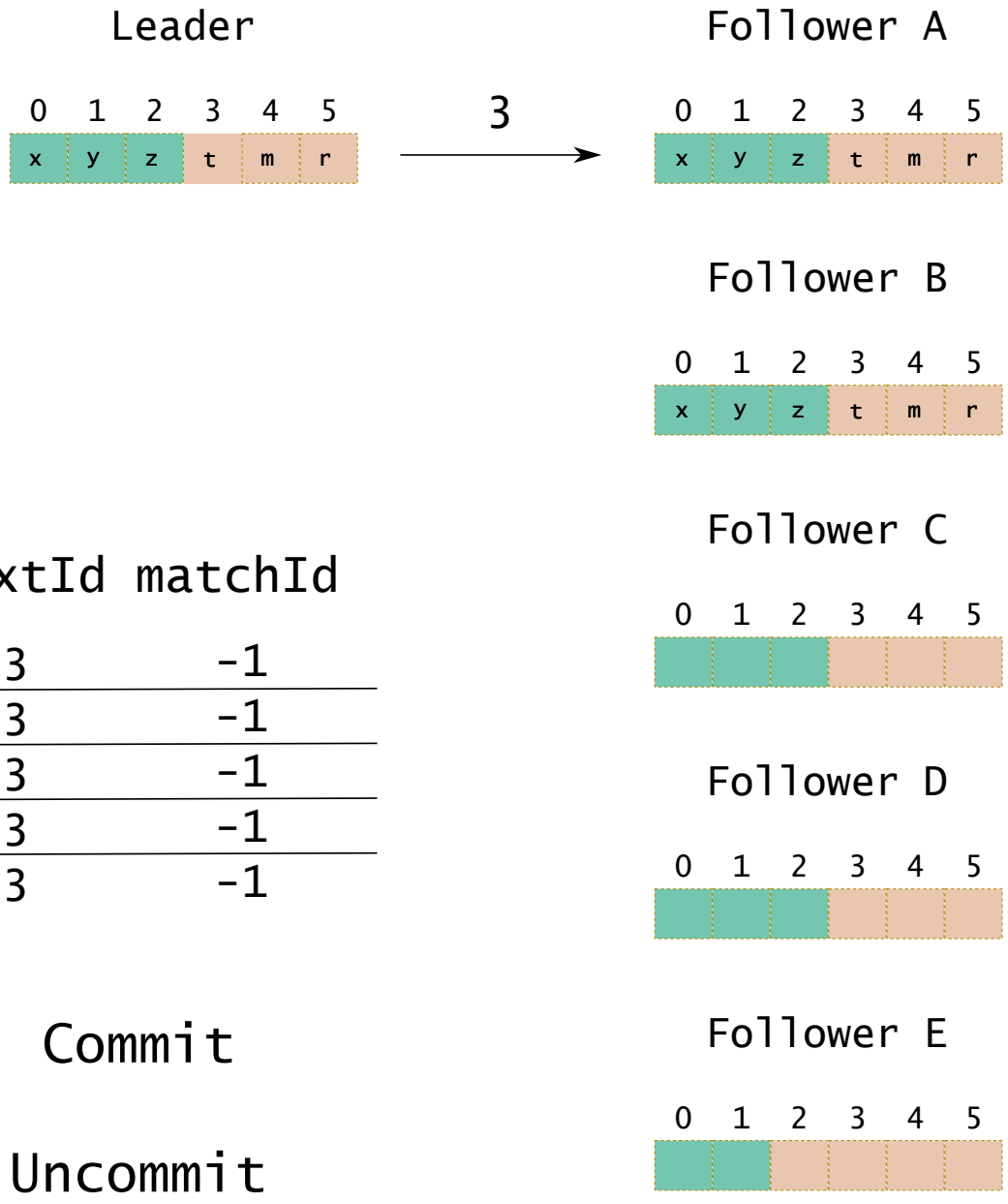




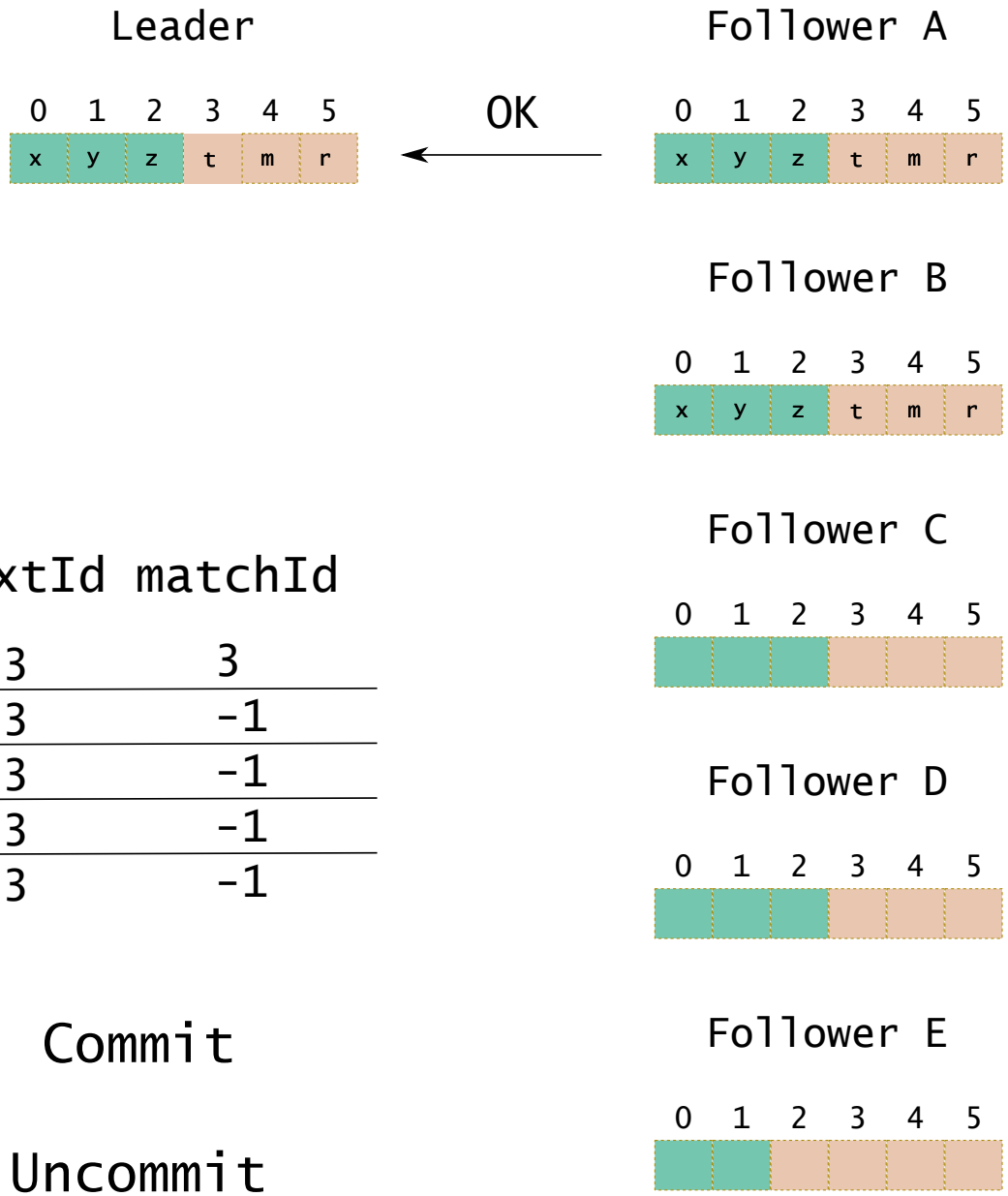
Fase normal de Raft

- Ojo, en lo que sigue está simplificado
- El mensaje de `AppendEntries` y el heartbeat
- Son lo mismo y llevan todo
- `AppendEntries` se manda periódicamente, puede que sin entries (heartbeat)
- Dibujamos los mensajes a los `Follower` secuencialmente, en realidad van en paralelo

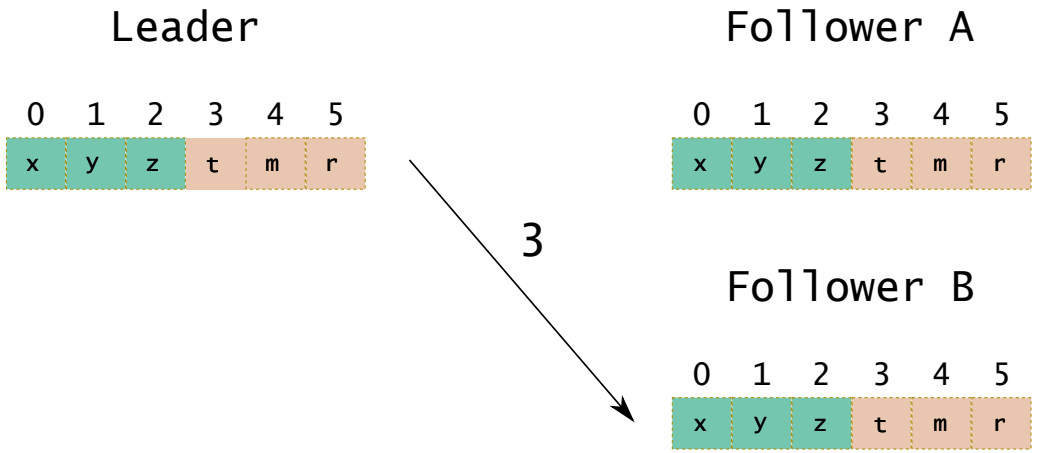
Fase normal de Raft, log



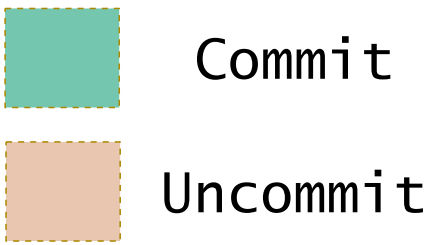
Fase normal de Raft, log



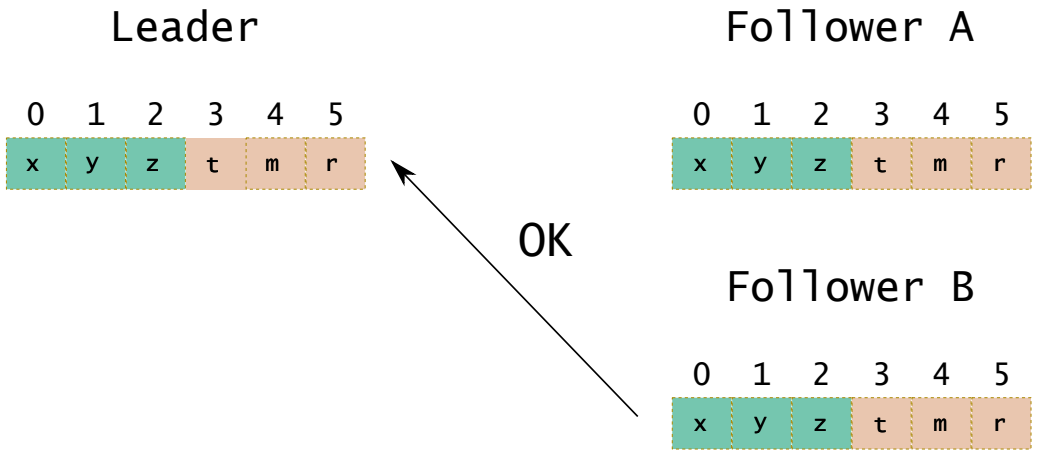
Fase normal de Raft, log



	nextId	matchId
A	4	3
B	3	-1
C	3	-1
D	3	-1
E	3	-1



Fase normal de Raft, log

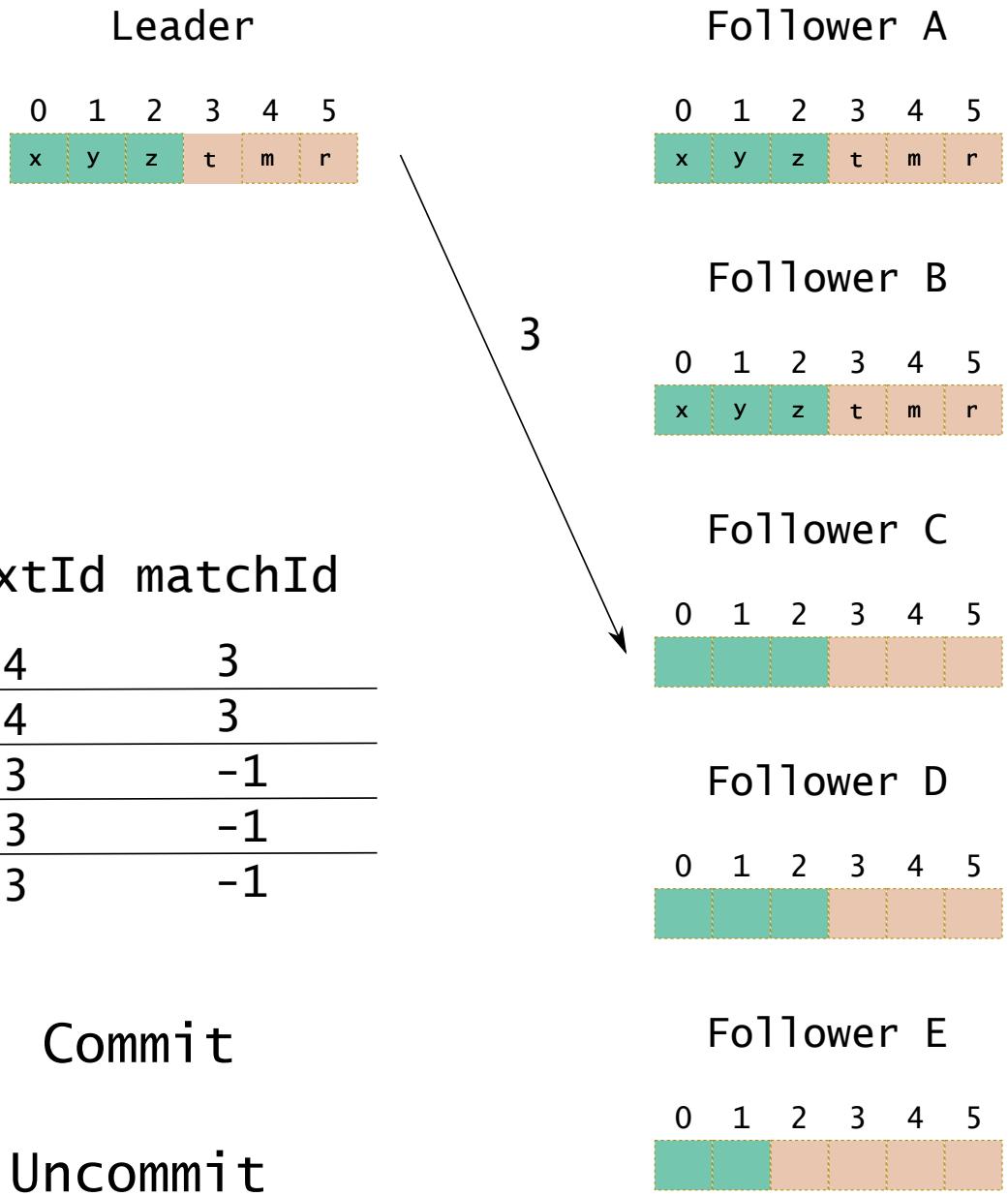


	nextId	matchId
A	4	3
B	3	3
C	3	-1
D	3	-1
E	3	-1

Commit

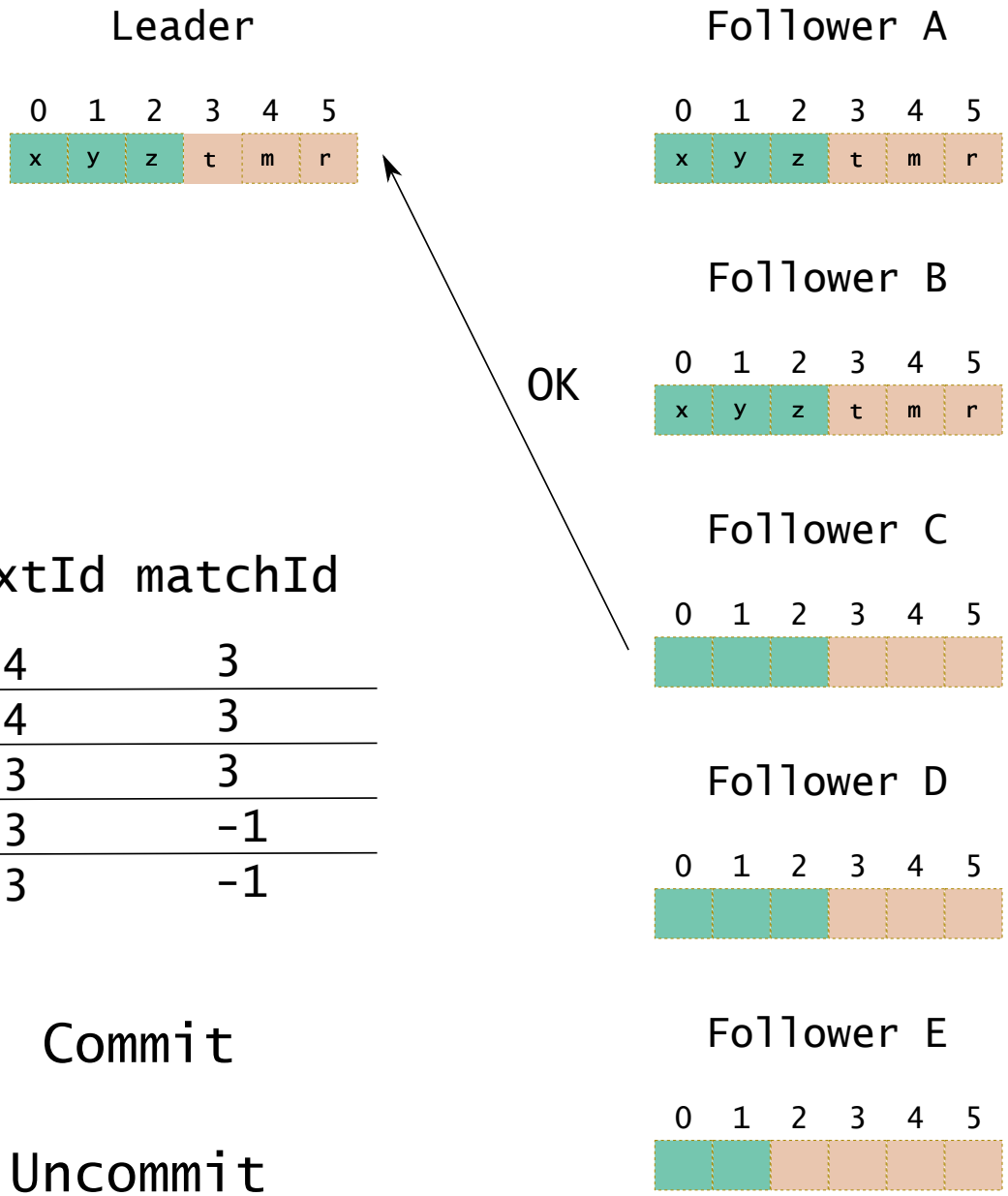
Uncommit

Fase normal de Raft, log



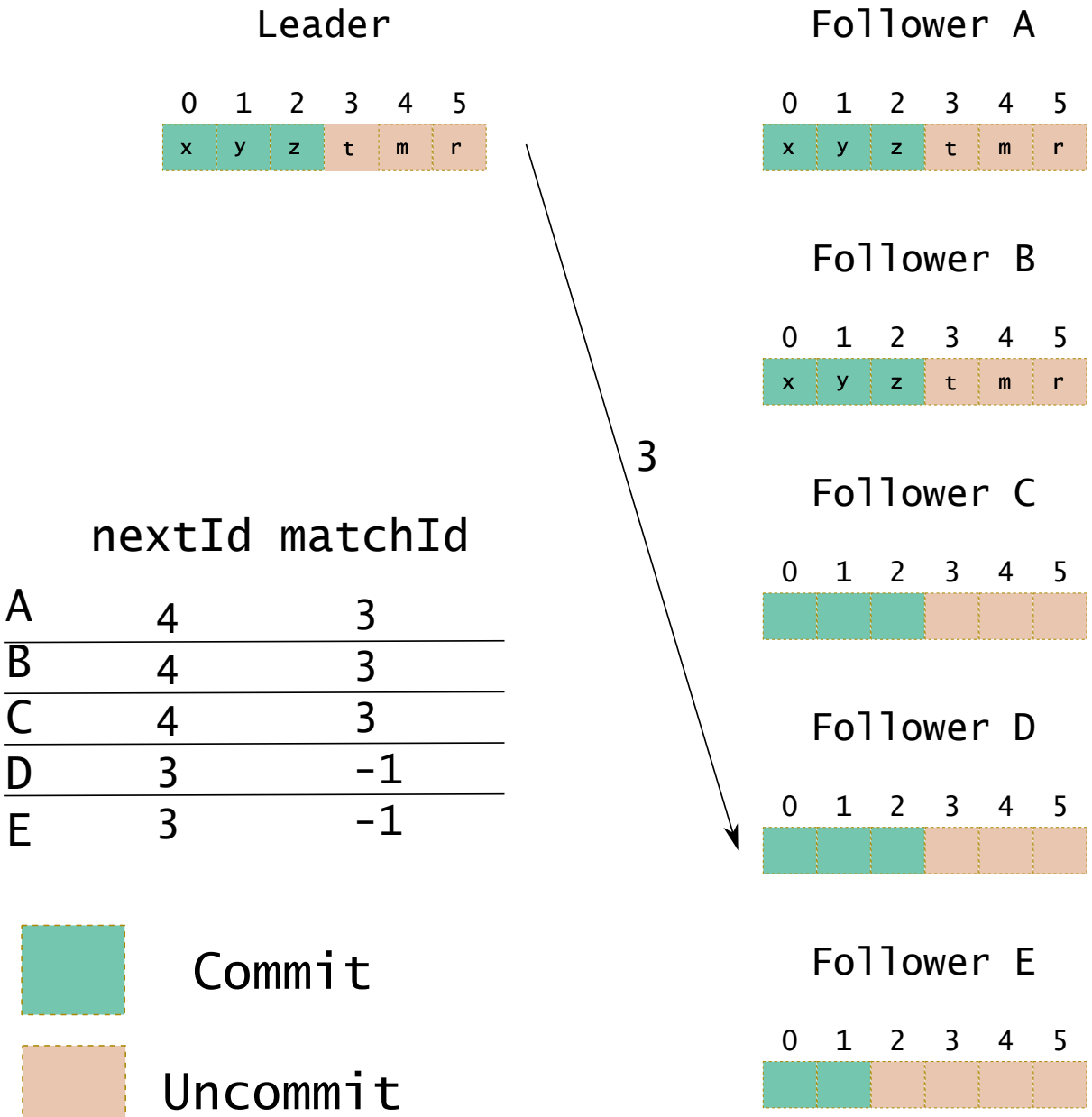
	nextId	matchId
A	4	3
B	4	3
C	3	-1
D	3	-1
E	3	-1

Fase normal de Raft, log



	nextId	matchId
A	4	3
B	4	3
C	3	3
D	3	-1
E	3	-1

Fase normal de Raft, log

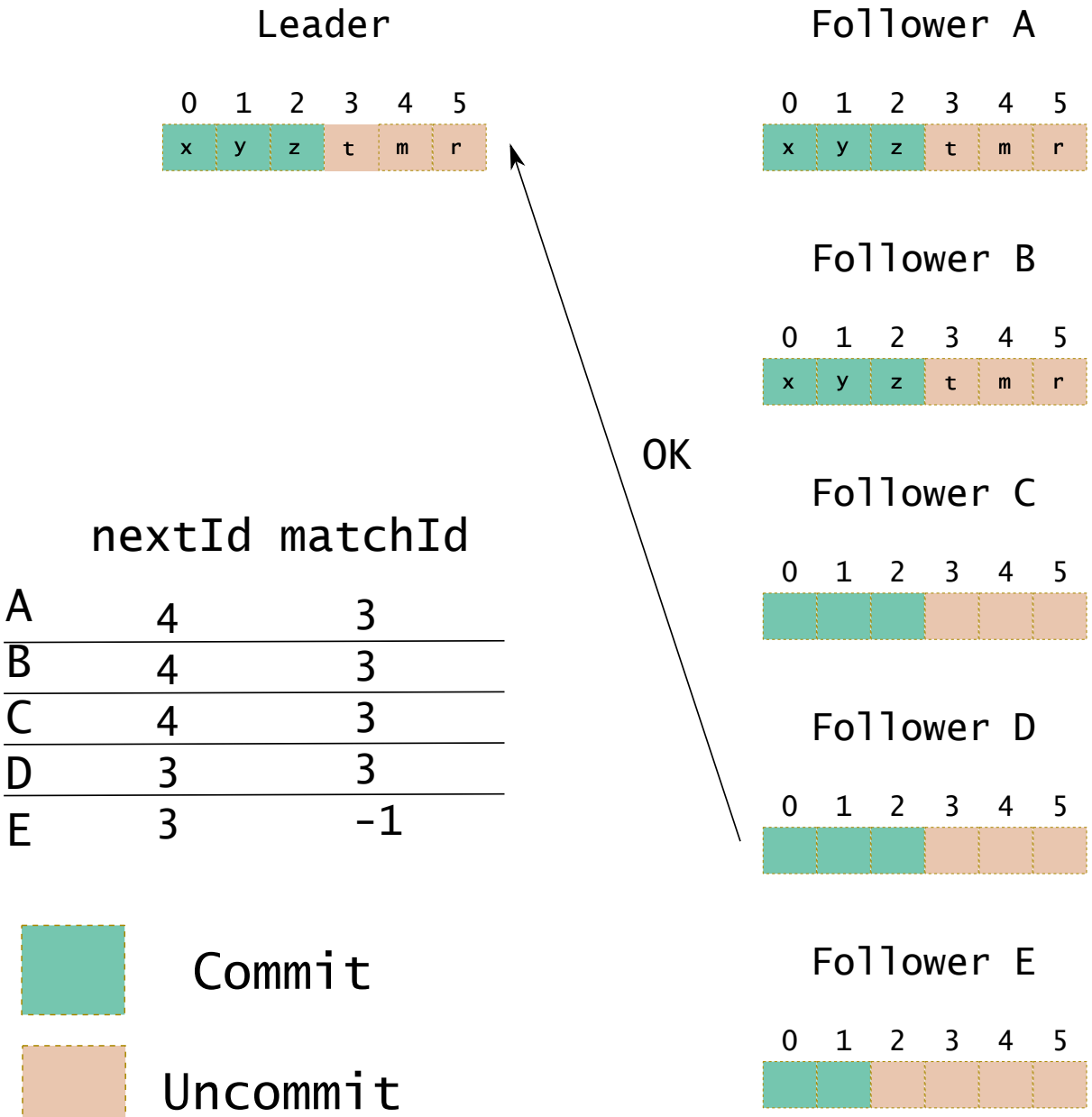


	nextId	matchId
A	4	3
B	4	3
C	4	3
D	3	-1
E	3	-1

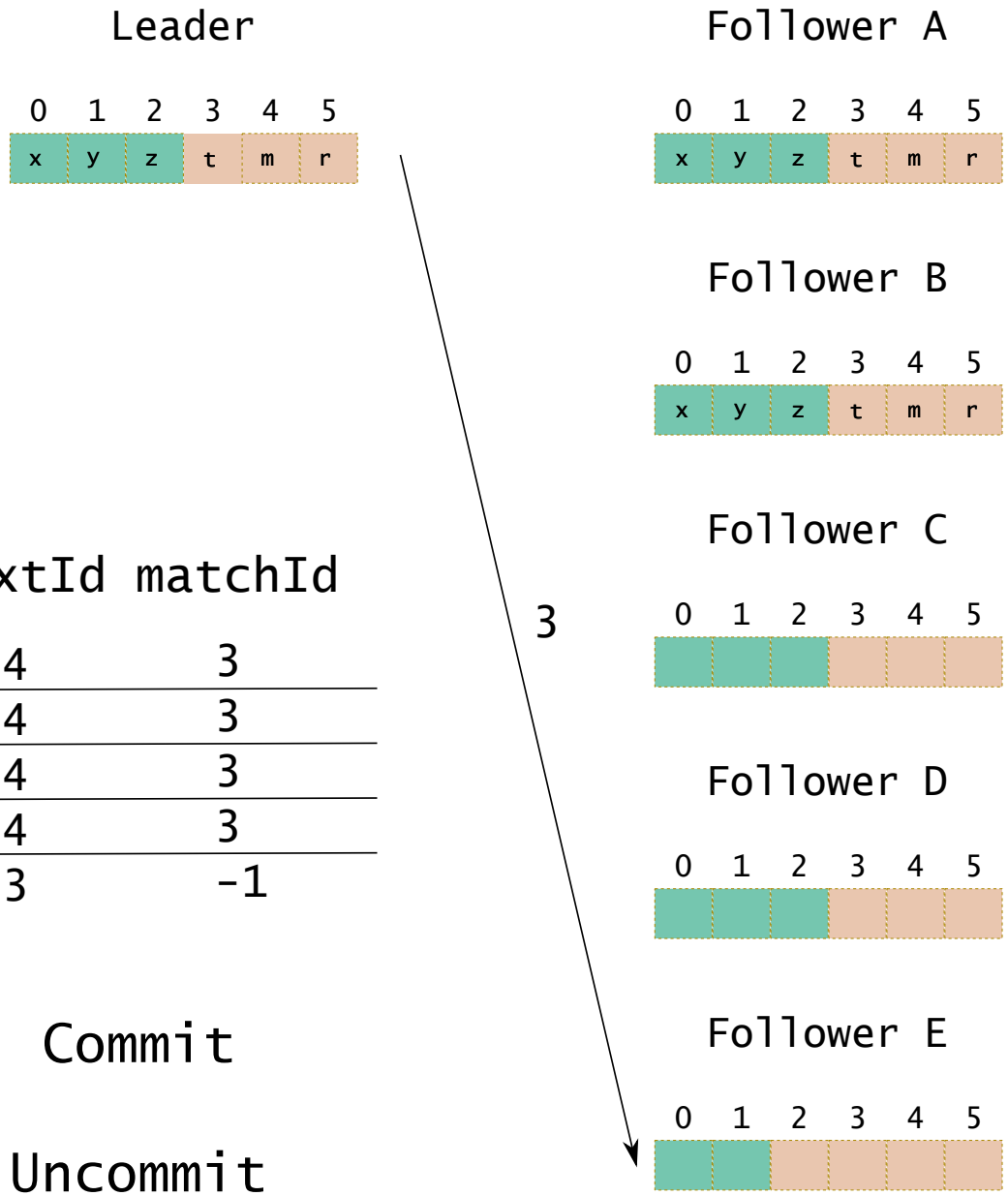
Commit

Uncommit

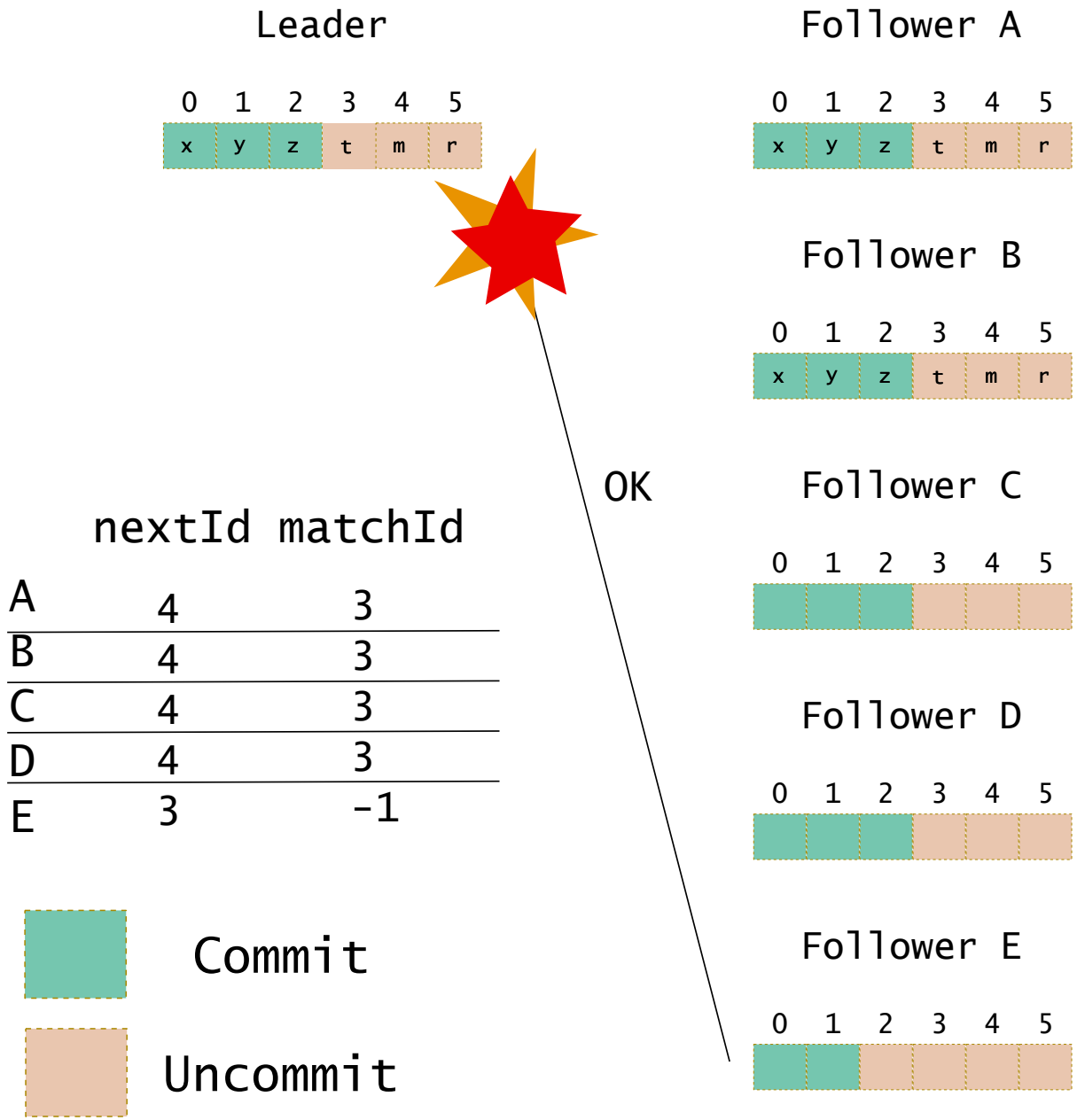
Fase normal de Raft, log



Fase normal de Raft, log



Fase normal de Raft, log

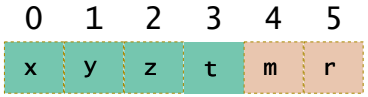


Fase normal de Raft, log

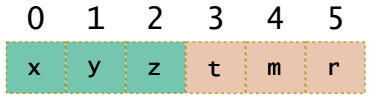
- Aunque se ha perdido uno, hay mayoría, aumento mi commit
- De ese no me contestan así que mando con un índice menos (si no me contestan o me contestan error)
- Si me contestan error es porque el prefijo del log no coincide

Fase normal de Raft, log

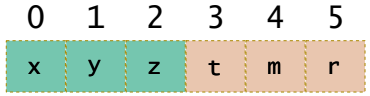
Leader



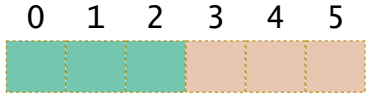
Follower A



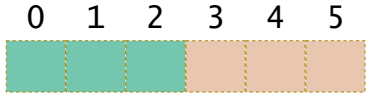
Follower B



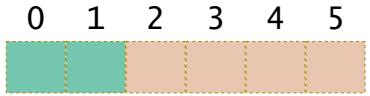
Follower C



Follower D



Follower E

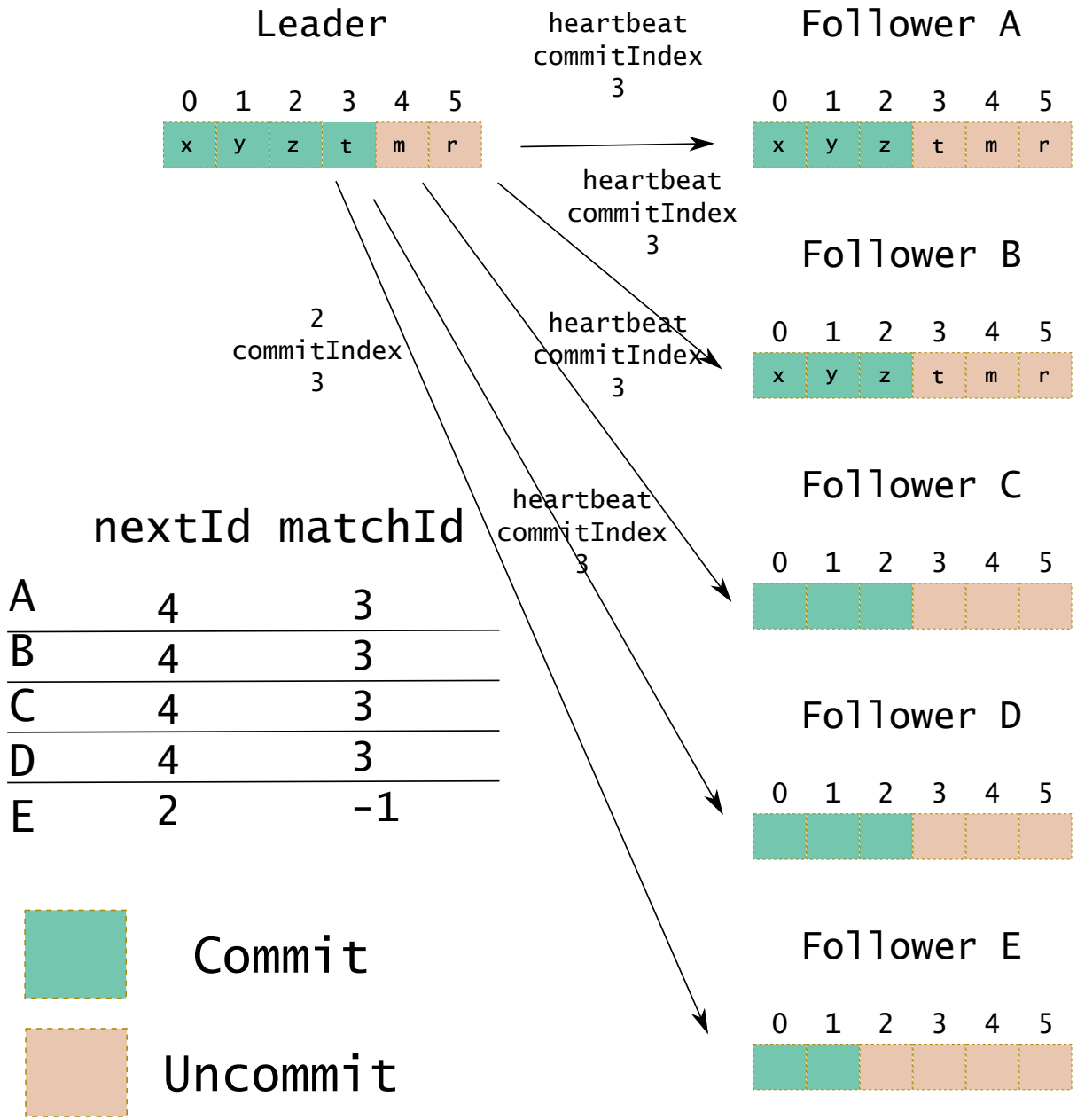


	nextId	matchId
A	4	3
B	4	3
C	4	3
D	4	3
E	2	-1

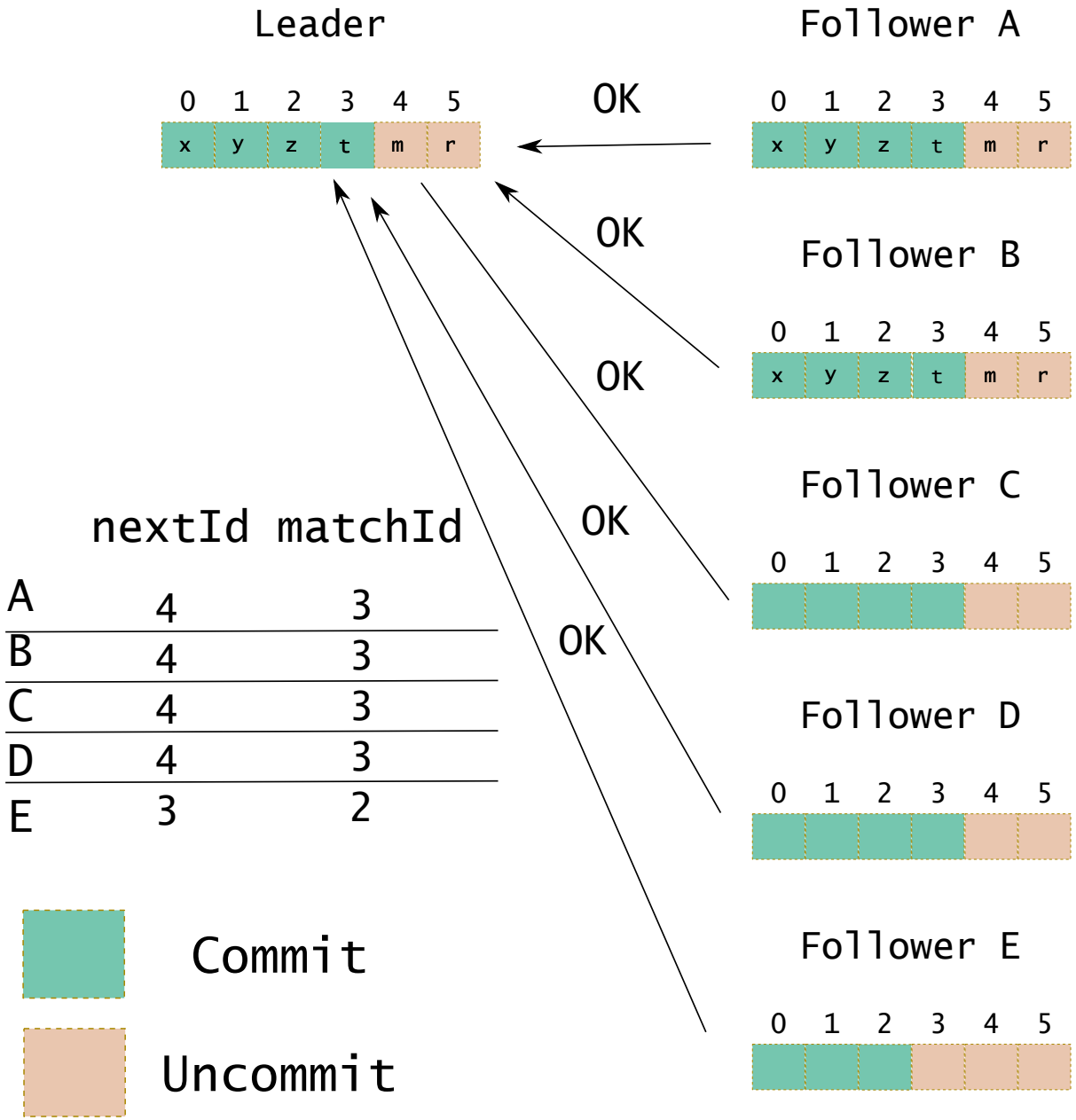
Commit

Uncommit

Fase normal de Raft, log



Fase normal de Raft, log



Log

- El log es lo más delicado de Raft
- En teoría cada entrada es un cambio de estado en una máquina de estados
- Hacen falta ciertas garantías para que funcione.

Log

- Cuando se manda `AppendEntries` a un Follower se indica la entrada anterior
- Se intenta garantizar que hay un prefijo común entre el Leader y los Followers
- El Leader lleva cuenta de qué le ha mandado a cada uno `nextId` y `matchId`
- Empieza desde lo que se hizo commit en él, si hay un error, retrocede

Raft: cuando falla

- Varios Candidate entran en un duelo
- Piden votos a diferentes grupos
- Entran en un bucle en el que vas teniendo muchas elecciones
- La aleatorización hace que (finalmente) converja a un sólo líder
- Hay que elegir bien los timeouts y depende de la red

Raft: cuando falla

- No hay suficientes réplicas para mayoría (error duro)
- En este caso, no avanza nunca el líder

Flp: imposibilidad de consenso con fallos bizantinos

- En un sistema completamente asíncrono
- Hay un teorema que dice que es imposible
- Lo que dice es que ningún algoritmo garantiza viveza
- Es imposible distinguir un fallo de un tarda mucho en contestar
- En la práctica, no es tan importante (timeouts, como en Raft)
- Ojo, los timeouts dependen del tipo de red

Raft en la industria, Apache Kudu

- `https://kudu.apache.org/overview.html`
- Capa de analítica de datos de Hadoop
- Haddop es un framework de almacenamiento distribuido

Raft en la industria, CockroachDB

- `https://www.cockorachlabs.com`
- Base de datos relacional distribuida escalable SQL

Raft en la industria, Consul

- `https://www.consul.io`
- Para orquestación de contenedores/máquinas/infrasestructura/servicios

Raft en la industria, LogCabin

- `https://github.com/logcabin`
- Almacentamiento distribuido, del creador de Raft

Paxos en la industria, Chubby, Zookeeper

- Chubby, sistema de locking de grano gordo con algo de almacenamiento, lo usa GFS y Bigtable (el fs y la base de datos de Google)
- Zookeeper, sistema de coordinación de servidores distribuidos, no implementa exactamente Paxos, sino Zab, muy similar (lo usa Hadoop, implementación abierta de algo parecido a GFS).

Paxos en la industria, Chubby

- Multi-decree Paxos
- Chubby, sistema de locking de grano gordo
- almacenamiento pequeño, seguro, lento, para cosas muy importantes (metadatos de GFS, Bigtable)

Paxos en la industria, Zookeeper

- Parte de Hadoop
- Implementa Zab, algoritmo de consenso similar a Paxos (no exactamente igual)
- Similar a un Sistema de Ficheros (jerárquico, znodes, ficheros y directorios)
- Elección de líder
- Propagación de estado (hay un orden total establecido)
- Eventos de cambios de estado (znodes)
- Lento

Paxos en la industria, Apache Cassandra

- Base de datos híbrida key/value y tabular
- En Facebook originalmente, muy escalable
- Usa un lenguaje de acceso propio CQL