UNIVERSITÀ
DEGLI STUDI
DI UDINE

IoT e Sistemi Distribuiti

Alcuni limiti e problemi dei sistemi distribuiti

Coordinamento e consenso nei sistemi distribuiti

nozione condivisa del tempo / consenso

- Dato un insieme di processi distribuiti:
  - come **coordinare** le loro azioni?
  - come raggiungere un **accordo su un valore**?
- Ad esempio: dato un **sistema di controllo automatico** della velocità, implementato da diversi processi di controllo ridondanti: come decidere se **il veicolo sta funzionando correttamente**?
- Non abbiamo relazioni master/slave, al fine di evitare i cosiddetti "**single point of failure**".
- Nei sistemi distribuiti, sia i **nodi** che i **canali di comunicazione** possono essere **soggetti a guasti/problemi**.
- La ricerca in questo campo ha dimostrato dei **risultati negativi sorprendenti** anche in presenza di guasti non particolarmente gravi.

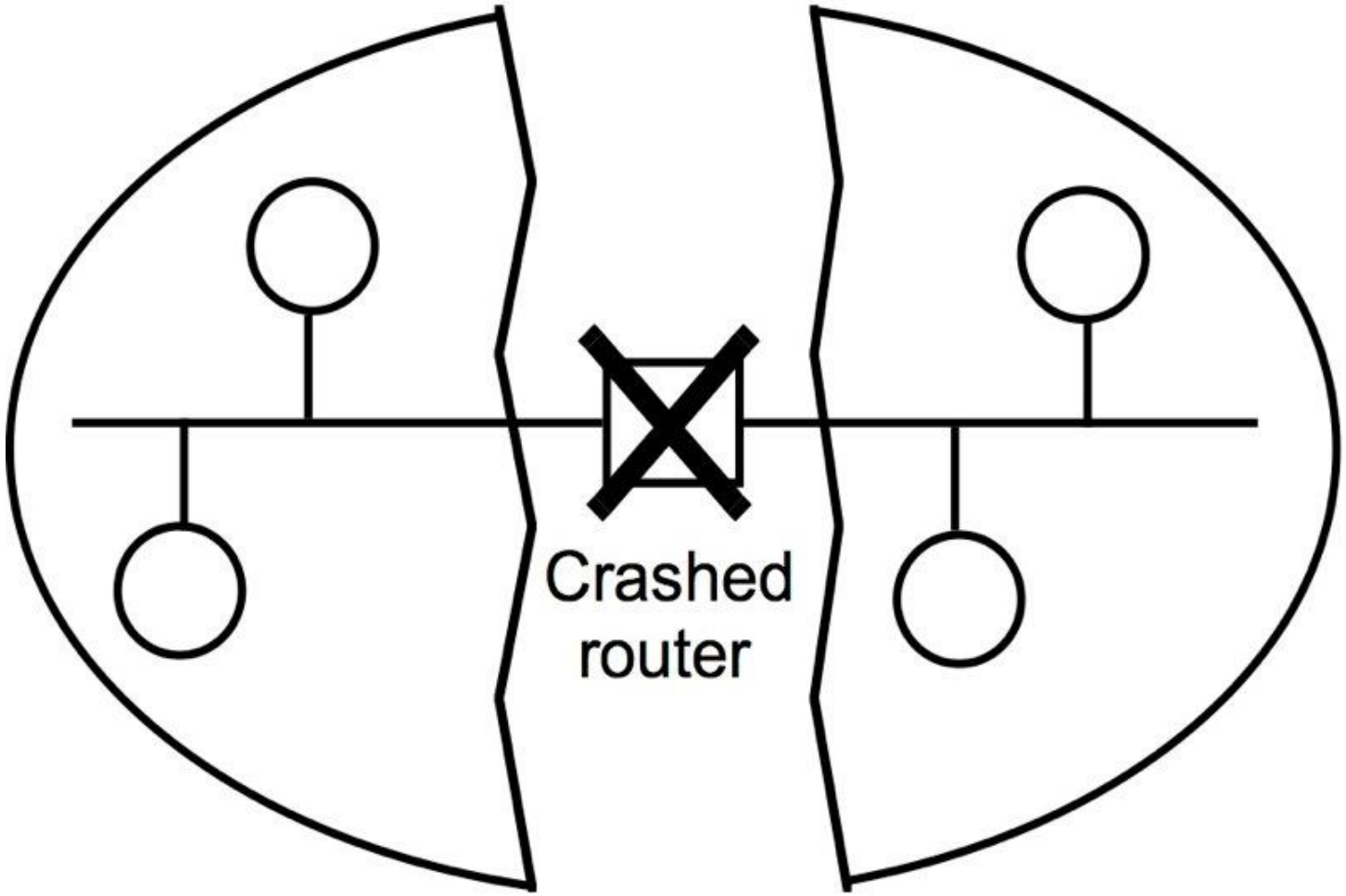
per raggiungere il consenso

Assunzioni sulle condizioni di fallimento

- Supponiamo che i **canali di comunicazione**, a differenza dei nodi, siano **affidabili**.
  - Su ogni canale i messaggi inviati **alla fine** giungono al **buffer di input** del destinatario.
  - Nei sistemi **asincroni** non ci sono vincoli temporali.
- Questo copre anche il caso in cui un collegamento venga interrotto, ma poi venga ripristinato (ad esempio un router si arresta in modo anomalo) - i **messaggi** vengono **mantenuti in coda**.
- Durante il guasto, la **rete** può essere **partizionata**.
  - La comunicazione all'interno della stessa partizione rimane funzionante.
  - La comunicazione tra le partizioni è (molto) ritardata.
  - I processi potrebbero non essere in grado di comunicare contemporaneamente.
- Nelle reti reali, la connettività può essere **asimmetrica** e **non transitiva**.

Partizionamento di una rete

- In seguito ad un guasto:



Il problema del consenso in sistemi con errori

- Finora, abbiamo ipotizzato che i processi cooperino per produrre un risultato finale corretto.
- In generale, l'accordo/consenso è fondamentale in molti casi:
  - elezione di un coordinatore,
  - decisione se effettuare o meno una transazione,
  - divisione dei compiti tra i processi cooperanti,
  - sincronizzazione,
  - swarming di droni,
  - …
- Quando i processi sono "**perfetti**", raggiungere tale accordo è semplice.
- Altrimenti, possono verificarsi dei problemi ...

Fallimenti dei processi

- Salvo diversa indicazione, un processo fallisce a causa di un crash.
- Situazioni più complesse sono problemi arbitrari ("bizantini") (ad es., ancora in esecuzione, ma malfunzionanti, con stato interno corrotto, ecc.).
- Un **processo corretto** è quello che **non presenta errori** in nessun momento dell'esecuzione.
  - un processo che subisce un crash, prima di tale evento, è classificabile come "non fallito", ma non come "corretto".
- Problema: **come decidere** se un processo ha avuto esito negativo (ad esempio, si è bloccato)?
  - I messaggi di "**ping**" potrebbero essere **ritardati**.

Modelli di fallimento dei processi

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send*, but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

fail-silent

cause 'guota

Rilevamento dei crash

- Un **rilevatore di errori/fallimenti** è un **servizio** software che risponde alle richieste sullo **stato di funzionamento** di un determinato processo.
- Spesso viene implementato come un **oggetto locale** del processo monitorato, eseguendo un algoritmo di rilevamento guasti (watchdog, rilevatore di guasti locale).
- I software di rilevamento dei guasti sono generalmente **poco affidabili**.

Rilevamento dei crash

- Un rilevatore di crash inaffidabile può portare alle seguenti valutazioni:
  - **sospetto fallimento**: qualcosa (apparentemente) sta andando storto (ma forse non è così);
  - **errore inatteso**: apparentemente funziona ancora (ad es.: è stato ricevuto un messaggio di recente... ma forse in seguito si è verificato un problema).
- Un rilevatore di guasti affidabile invece porta a stabilire quanto segue:
  - **processo fallito**: il rilevatore è sicuro che il processo si sia bloccato;
  - **errore inatteso**: apparentemente funziona ancora (è stato ricevuto un messaggio di recente... ma forse in seguito si è verificato un problema).

Processo Compagno

Rilevamento di crash

- Il rilevamento degli errori rappresenta una **conoscenza locale** di un processo:
  - quindi processi diversi possono "vedere" diversi esiti di fallimento.
- Nei **sistemi asincroni**, può essere implementato da messaggi di tipo "**still-alive**" ("ping"), inviati ogni T secondi:
  - i rilevatori possono inferire un "guasto sospetto" se non ricevono un messaggio dopo T + D (D = ritardo di rete);
  - scegliere T è difficile: se troppo piccolo, comporta troppi falsi fallimenti; se troppo grande, può comportare ritardi nella scoperta degli errori.
- I **rilevatori** di guasti **affidabili** possono essere implementati solo in **sistemi sincroni** (dove D è limitato):
  - canali speciali (ad es. collegamenti di rete dedicati, porte seriali).

Il problema del consenso

- Tutti i dispositivi corretti che controllano un **veicolo autonomo** dovrebbero decidere di procedere a compiere la manovra necessaria, oppure tutti dovrebbero decidere di interrompere le azioni in corso (dopo che ciascuno ha proposto un‘opzione o l'altra).
- In una **transazione di trasferimento di denaro elettronico**, tutti i processi coinvolti devono concordare coerentemente sull'esecuzione della transazione (debito/credito) o meno.
- In un **regime di mutua esclusione**, i processi devono concordare su quale processo debba entrare nella sezione critica.
- Nelle **procedura di elezione**, i processi devono concordare il processo eletto.
- In un **multicast totalmente ordinato**, i processi devono concordare un ordine di consegna dei messaggi che sia coerente.

blockchain

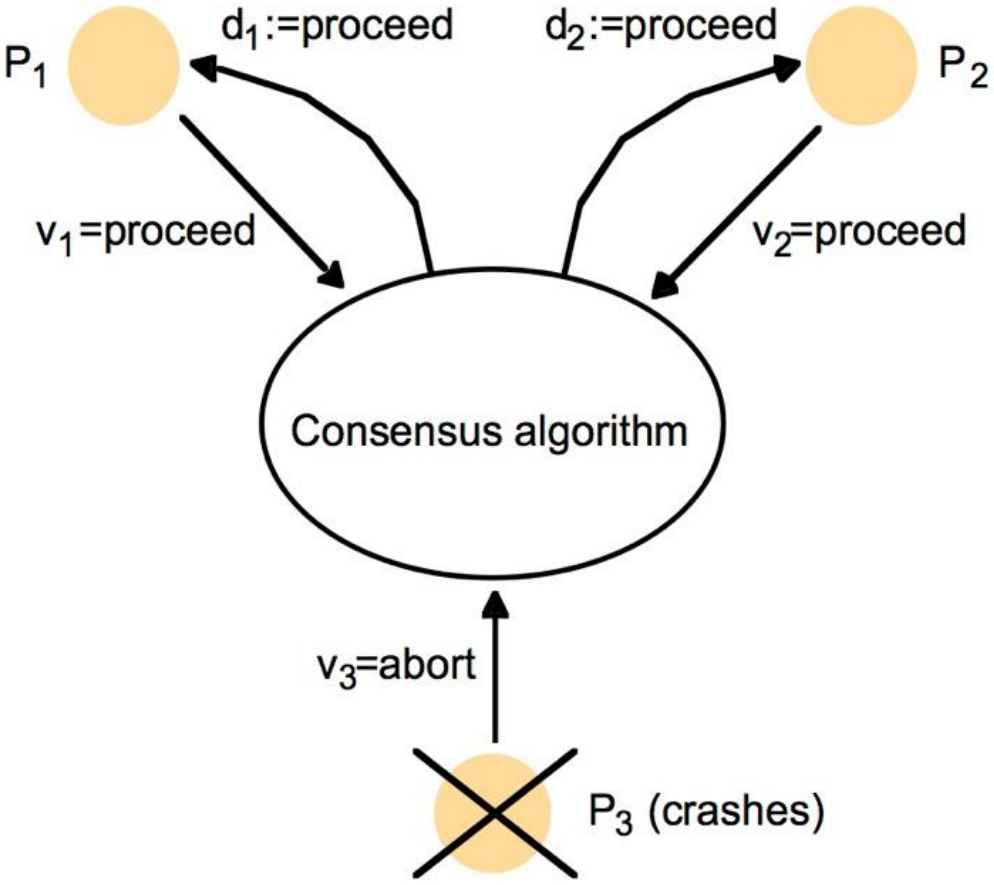
come raggiungere il consenso?

Il problema del consenso

- Fattori che influiscono sul raggiungimento del consenso:
  - fallimenti (crash),
    - guasti di canali di comunicazione o errori di processo,
    - crash (fail-silent) o errori bizantini (arbitrari);
  - caratteristiche della rete (sincrone o asincrone);
  - rilevatori di guasti (affidabili o inaffidabili);
  - autenticazione dei messaggi (con firma digitale) o no:
    - un processo può "mentire" sul contenuto del messaggio ricevuto da un processo corretto?
    - l'avversario può inviare un messaggio con un ID di un mittente falso?
  - modello:
    - processi comunicanti mediante **scambio di messaggi**;
    - obiettivo: raggiungere il consenso anche in presenza di malfunzionamenti:
      - presupposto: la **comunicazione** è **affidabile**, ma i processi possono fallire.

Il problema del consenso

- Consenso sul **valore di una variabile di decisione** tra **tutti** i processi corretti:
  - p_i è in uno stato di indecisione e propone un singolo valore v_i;
  - successivamente, i processi comunicano tra loro per scambiarsi i valori;
  - nel fare ciò, p_i imposta la variabile di decisione d_i ed entra nello stato di avvenuta decisione dopo il quale il valore di d_i rimane invariato.



Proprietà di un algoritmo del consenso

- **Terminazione**: alla fine, ogni processo corretto imposta la sua variabile di decisione.
- **Accordo**: per tutti i p_i e p_k corretti tali che stato(p_i) = stato(p_k) = deciso → $d_i = d_k$
- **Integrità**: se tutti i processi corretti hanno proposto lo stesso valore, tutti i processi corretti hanno scelto quel valore nello stato di avvenuta decisione.
  - variazione: ... un numero di processi corretti oltre una certa soglia prefissata ha scelto quel valore una volta raggiunto lo stato di avvenuta decisione.

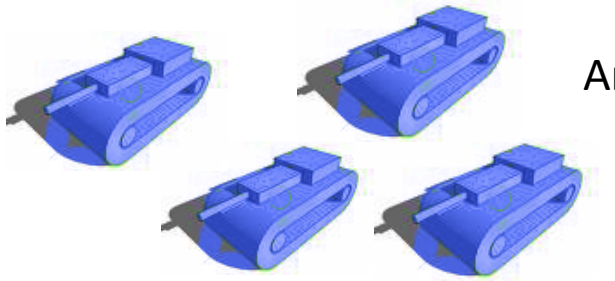
Il problema del consenso in un ambiente privo di errori

- Ogni processo **invia in multicast** (affidabile) i valori proposti.
- Dopo aver ricevuto i messaggi, **viene calcolato il valore proposto** dalla maggioranza oppure il valore "non definito", se non esiste una maggioranza. [Nota: è possibile scegliere altri criteri.]
- Proprietà:
  - **la terminazione è garantita** dall'affidabilità del multicast;
  - accordo, integrità: definizione della maggioranza e integrità del multicast affidabile (**tutti i processi calcolano la stessa funzione sugli stessi dati**).

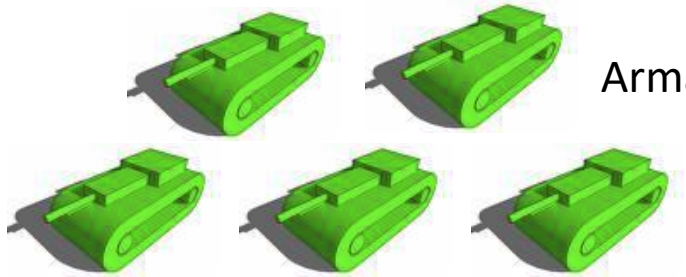
Il problema del consenso in un ambiente soggetto ad errori

Il problema dei Generali Bizantini

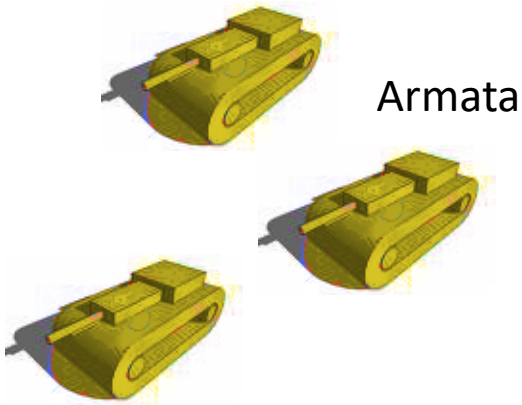
(Lamport et al. 1982)



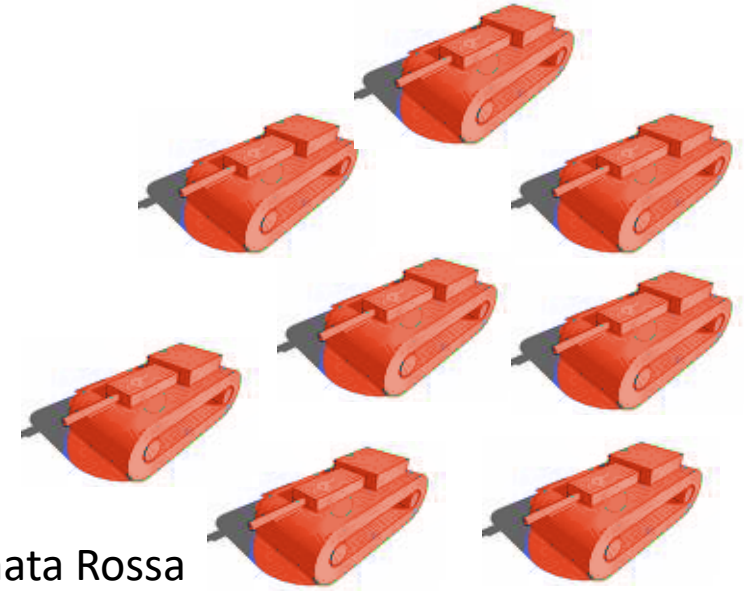
Armata Blu



Armata Verde



Armata Gialla



Armata Rossa

Se le armate blu, gialle e verdi decidono di attaccare insieme l'armata rossa, avranno una possibilità di vittoria, altrimenti perderanno sicuramente.

Il problema del consenso in un ambiente soggetto ad errori

Il problema dei Generali Bizantini

(Lamport et al. 1982)

- Tre (o più) generali devono concordare un attacco o una ritirata.
- Il comandante supremo emette un ordine:
  - gli altri devono decidere di attaccare o ritirarsi in base all'ordine;
  - uno dei generali può essere infido.
- Se il comandante è infido, può proporre di attaccare ad un generale e di ritirarsi all'altro.
- Se gli altri sono infidi, possono dire ad uno dei loro pari che il comandante ha ordinato di attaccare, e agli altri che il comandante ha ordinato di ritirarsi.

Il problema del consenso in un ambiente soggetto ad errori

Il problema dei Generali Bizantini

(Lamport et al. 1982)

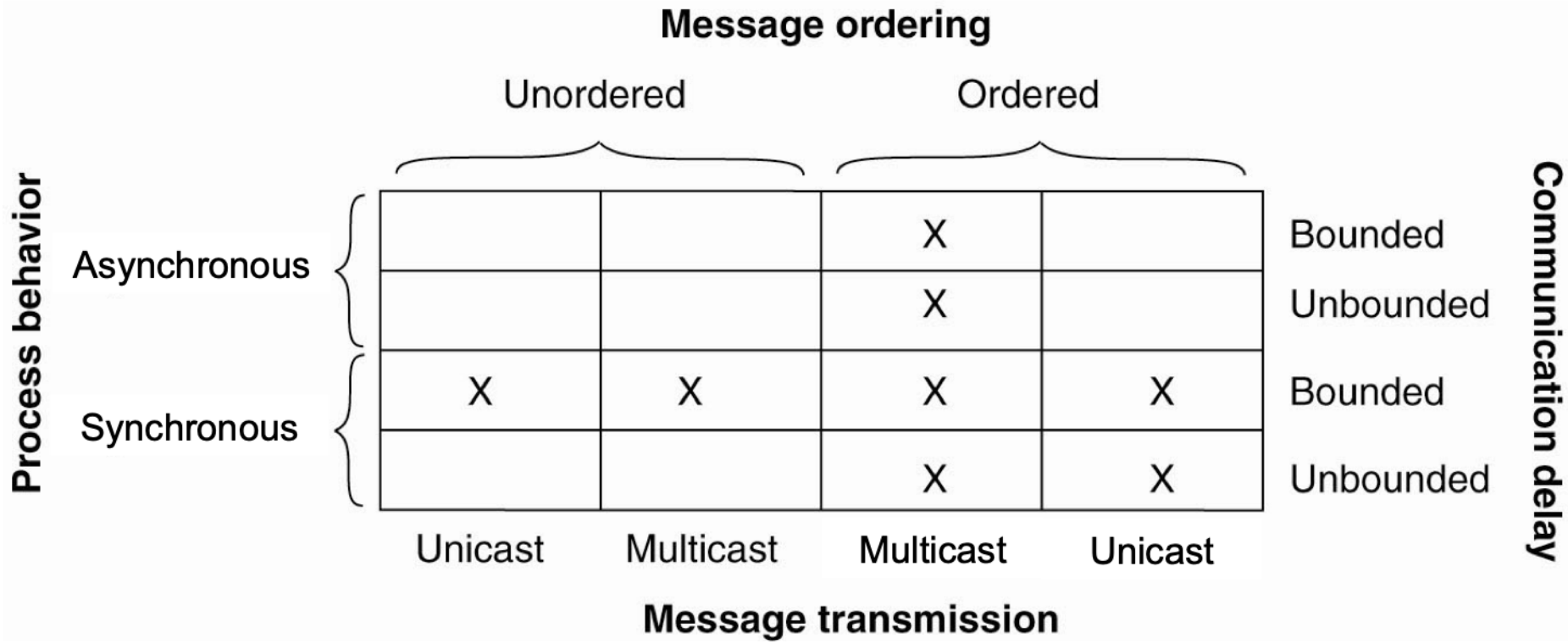
- Differenze rispetto al problema del consenso:
  - un processo fornisce un valore su cui gli altri devono concordare.
  - i due tipi di problemi sono riducibili fra loro.
- Proprietà:
  - **terminazione**: alla fine ogni processo corretto imposta la propria variabile di decisione;
  - **accordo**: il valore decisionale di tutti i processi corretti è lo stesso;
  - **integrità**: se il comandante è corretto, tutti i processi decidono in base al valore proposto dal comandante.
    - Nota: implica accordo solo se il comandante è corretto, ma il comandante non deve essere corretto (vedi sopra).

Il problema del consenso in un ambiente soggetto ad errori

Casi possibili:

- Sistemi sincroni vs. sistemi asincroni:
  - i processi sono sincroni se esiste una costante **c ≥ 1** in modo tale che ogni qualvolta un processo ha eseguito **c + 1** passi, tutti gli altri processi hanno eseguito almeno un passo.
- Il ritardo di comunicazione è limitato o meno:
  - il ritardo è limitato se tutti i messaggi inviati da un processo arrivano entro **r** passaggi in tempo reale, per un valore **r** predeterminato.
- La consegna dei messaggi è ordinata o meno:
  - la consegna dei messaggi è ordinata se i messaggi dei diversi mittenti vengono recapitati nell'ordine in cui sono stati inviati (in accordo al tempo reale globale).
- La trasmissione dei messaggi avviene tramite unicast o multicast.

Il problema del consenso in un ambiente soggetto ad errori

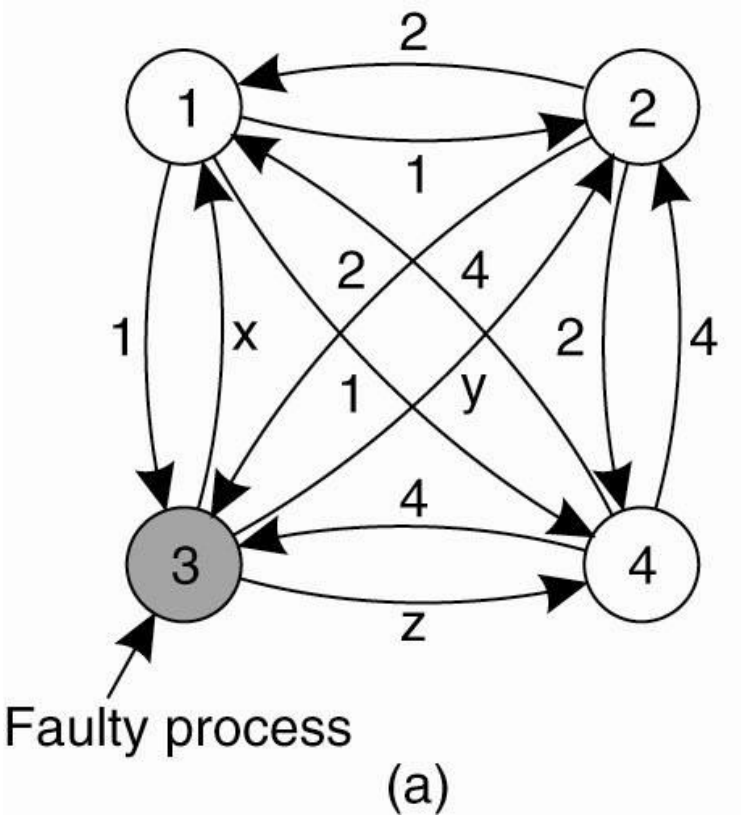


Casi in cui è possibile raggiungere il consenso in un sistema distribuito.

Il problema del consenso in un ambiente soggetto ad errori

Partiamo dal presupposto che:

- i processi siano sincroni;
- i messaggi vengano inviati in unicast, preservando l'ordinamento;
- il ritardo nella comunicazione sia limitato.



Il problema dei generali bizantini con un processo "difettoso".
(a) Ogni processo invia il proprio valore (decisione) agli altri.

Il problema del consenso in un ambiente soggetto ad errori

1	Got(1, 2, x, 4)
2	Got(1, 2, y, 4)
3	Got(1, 2, 3, 4)
4	Got(1, 2, z, 4)

(b)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

Il problema dei generali bizantini con un processo "difettoso".

(b) I vettori che ogni processo assembla in base alla fase (a).

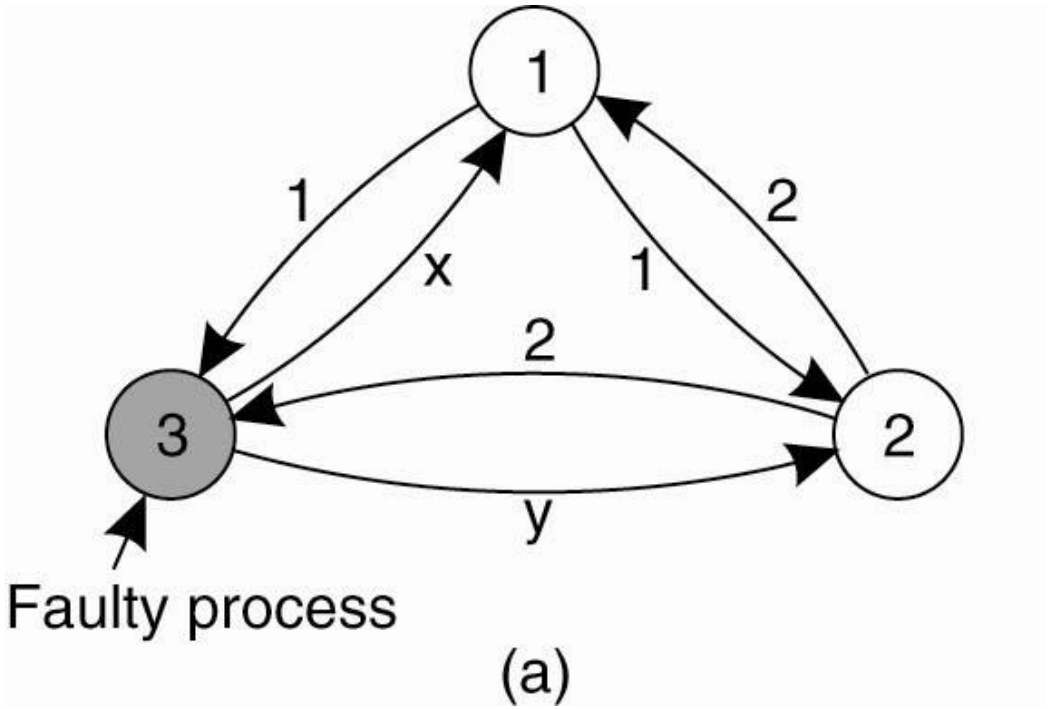
(c) I vettori che ciascun processo riceve dagli altri processi.

Il problema del consenso in un ambiente soggetto ad errori

Algoritmo:

- P_i manda in unicast v(i) a tutti gli altri processi.
- P_i assembla il vettore dei valori ricevuti [v (1), ..., v (n)].
- P_i manda in unicast [P_i, [v (1), ..., v (n)]] a tutti gli altri processi.
- P_i assembla il risultato finale nel modo seguente:
  - per ogni j, assegna al j° elemento del vettore risultato il valore presente nella maggior parte dei vettori ricevuti nella fase precedente in posizione j.

Il problema del consenso in un ambiente soggetto ad errori



(a)

1 Got(1, 2, x)
2 Got(1, 2, y)
3 Got(1, 2, 3)

(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

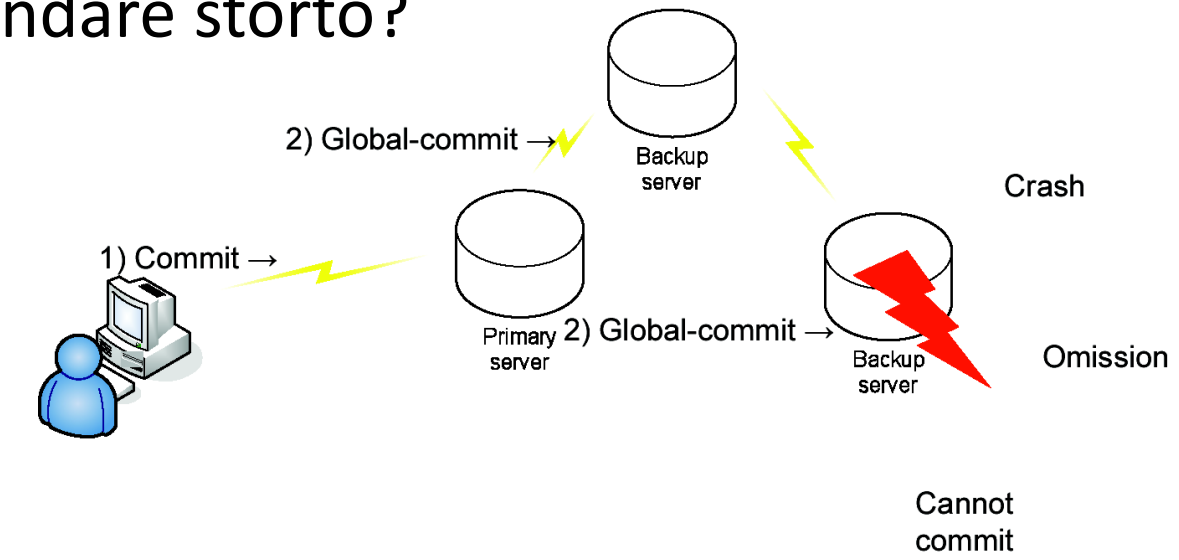
Situazione analoga a quella precedente, ma con due processi corretti e un processo "difettoso".

Il problema del consenso in un ambiente soggetto ad errori

- Finora, abbiamo ipotizzato quanto segue:
  - i processi sono **sincroni**,
  - i messaggi sono inviati in **unicast** e preservano l'**ordine**,
  - il ritardo nella comunicazione è **limitato**.
- Lamport ha dimostrato che in un tale sistema con k processi difettosi è possibile raggiungere un accordo solo se sono presenti 2k + 1 processi correttamente funzionanti (per un totale di 3k + 1).
- Fisher ha dimostrato che se il ritardo di comunicazione è illimitato, non è possibile alcun accordo, se anche un solo processo è difettoso (anche se quel processo fallisce in modalità silente).

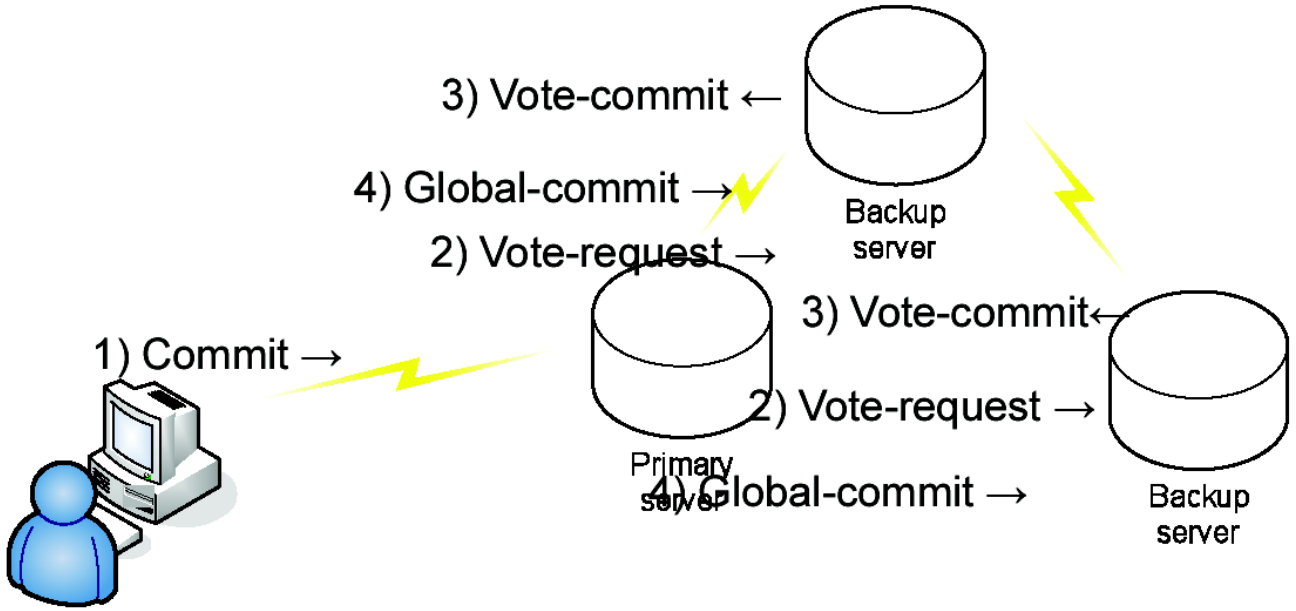
Esempio: commit in un database distribuito

- Dato un gruppo di processi ed un'operazione su un database:
  - o tutti effettuano il **commit** o tutti effettuano un **abort** (coerenza, validità, terminazione).
- Cosa può andare storto?



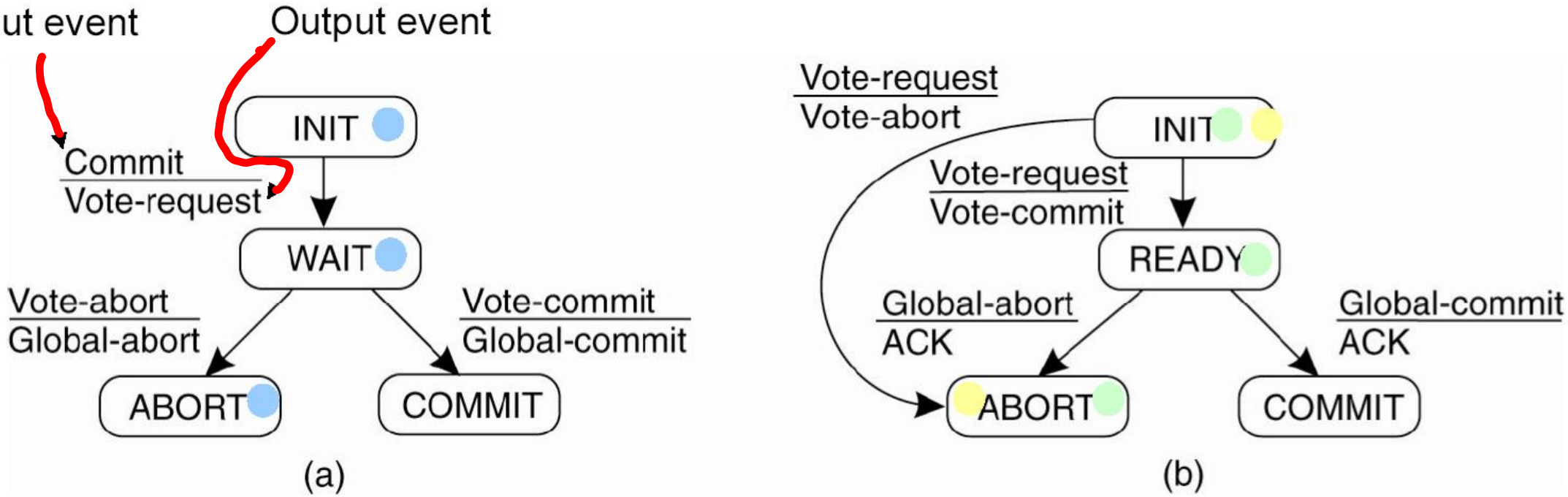
PERCORSO PERICOLOSO

2PC (Two-Phase Commit) Protocol



- Ipotesi di lavoro:
  - Errori e guasti vengono rilevati per mezzo di **timeout**.
  - I processi coinvolti possono essere **ripristinati**:
    - necessità di un meccanismo di **logging** su file system.

2PC

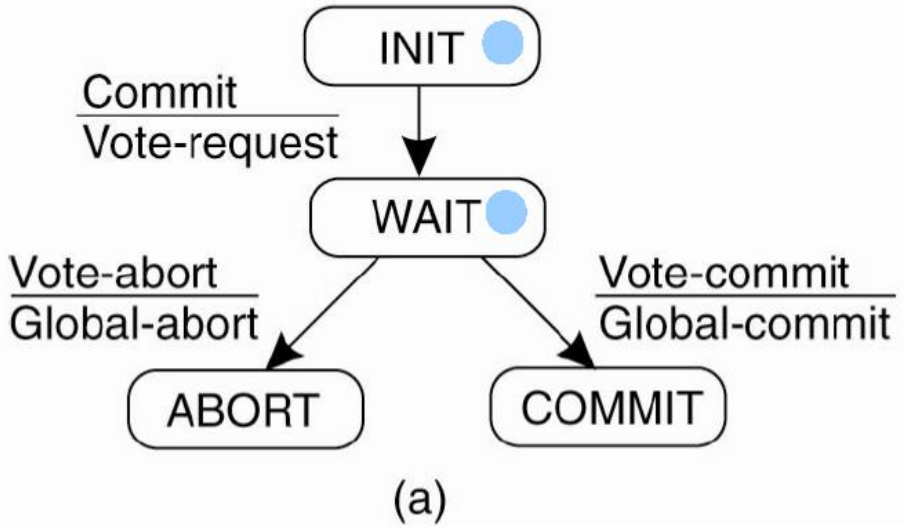


(a) Macchina a stati finiti del coordinatore.
(b) Macchina a stati finiti dei partecipanti.

2PC (il coordinatore - 1)

Actions by coordinator:

```
write START_2PC to local log;
multicast VOTE_REQUEST to all participants;
while not all votes have been collected {
    wait for any incoming vote;
    if timeout {
        write GLOBAL_ABORT to local log;
        multicast GLOBAL_ABORT to all participants;
        exit;
    }
    record vote;
}
```

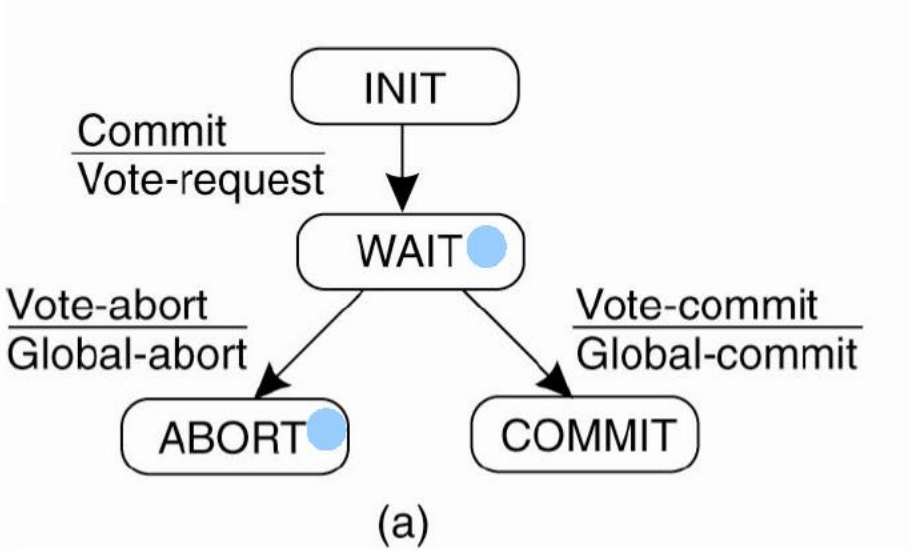


(a)

2PC (il coordinatore – 2)

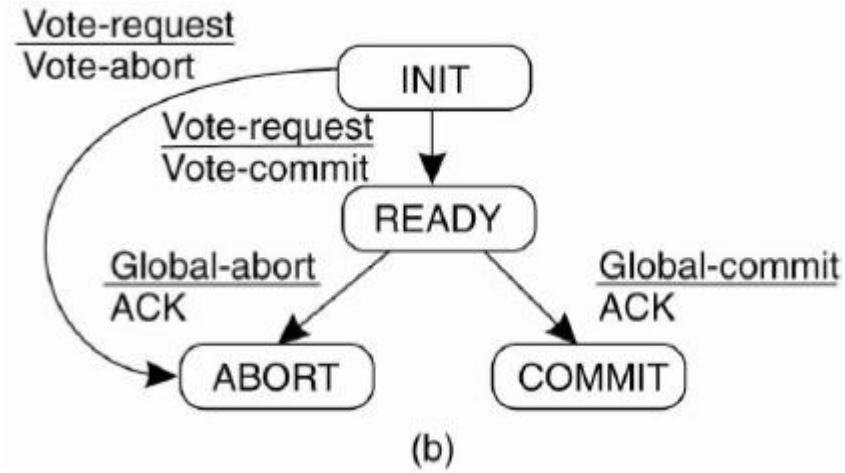
```
. . .

if all participants sent VOTE_COMMIT and coordinator votes COMMIT {
        write GLOBAL_COMMIT to local log;
        multicast GLOBAL_COMMIT to all participants;
} else {
        write GLOBAL_ABORT to local log;
        multicast GLOBAL_ABORT to all participants;
}
```



2PC (i partecipanti - 1)

- Blocco in INIT: vote-abort ed ABORT (dopo timeout).
- Blocco in READY:
  - il coordinatore potrebbe essere in crash.
  - Cosa fare?
    - attendere…
    - forse un altro il partecipante sa cosa fare…?



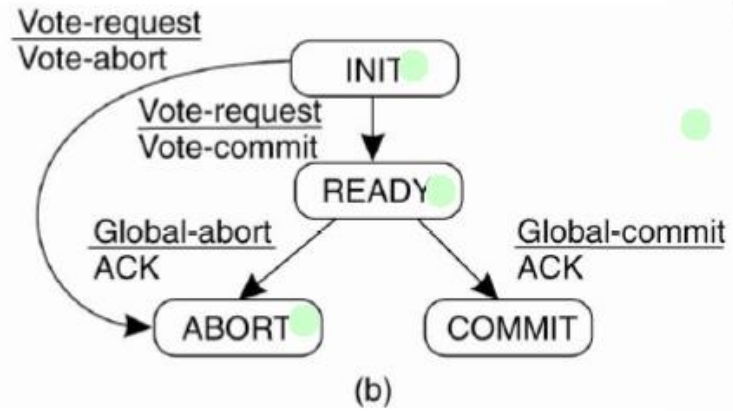
2PC (i partecipanti - 2)

P (bloccato in READY) chiede ad un altro partecipante Q…

State of Q	Action by P
COMMIT	Make transition to COMMIT
ABORT	Make transition to ABORT
INIT	Make transition to ABORT
READY	Contact another participant

se nessuno sa cosa fare ??

2PC (i partecipanti - 3)



(b)

actions by participant:

```
write INIT to local log;
wait for VOTE_REQUEST from coordinator;
if timeout {
    write VOTE_ABORT to local log;
    exit;
}
if participant votes COMMIT {
    write VOTE_COMMIT to local log;
    send VOTE_COMMIT to coordinator;
    wait for DECISION from coordinator;
    if timeout {
        multicast DECISION_REQUEST to other participants;
        wait until DECISION is received; /* remain blocked */
        write DECISION to local log;
    }
    if DECISION == GLOBAL_COMMIT
        write GLOBAL_COMMIT to local log;
    else if DECISION == GLOBAL_ABORT
        write GLOBAL_ABORT to local log;
} else {
    write VOTE_ABORT to local log;
    send VOTE_ABORT to coordinator;
}
```

(a)

wait + blocco

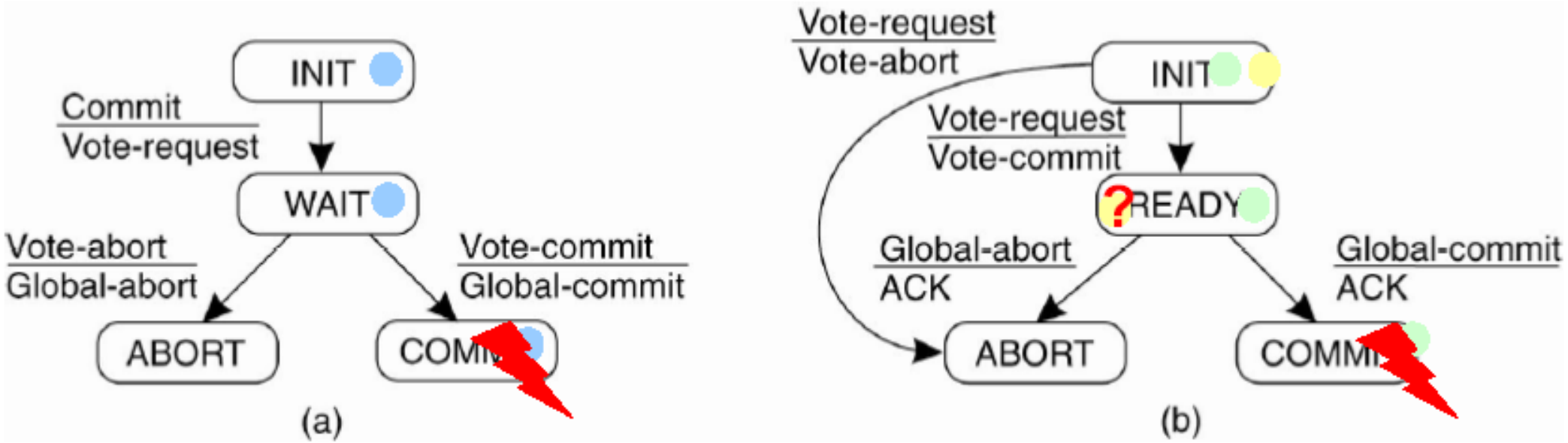
2PC (i partecipanti - 4)

Actions for handling decision requests: /* executed by separate thread */

```
while true {
    wait until any incoming DECISION_REQUEST is received; /* remain blocked */
    read most recently recorded STATE from the local log;
    if STATE == GLOBAL_COMMIT
        send GLOBAL_COMMIT to requesting participant;
    else if STATE == INIT or STATE == GLOBAL_ABORT
        send GLOBAL_ABORT to requesting participant;
    else
        skip; /* participant remains blocked */
}
```

(b)

Punto debole del 2PC



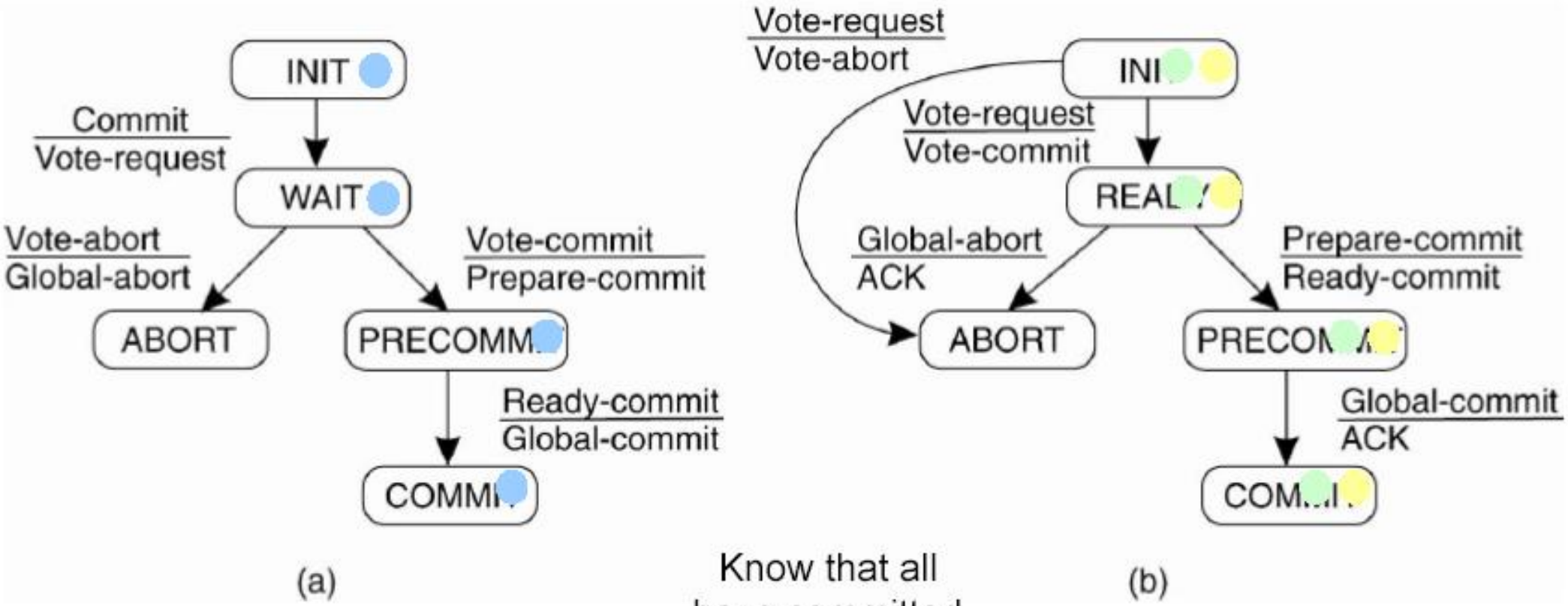
Il crash del coordinatore blocca tutto il sistema.

Soluzione: 3PC (Three-Phase Commit) Protocol

Gli stati del coordinatore e di tutti i partecipanti soddisfano le seguenti due condizioni:

1. Non esiste nessuno stato da cui sia possibile effettuare una transizione diretta ad uno stato COMMIT e ABORT.
2. Non esiste uno stato in cui non sia possibile prendere una decisione finale e dal quale sia possibile effettuare una transizione allo stato COMMIT.

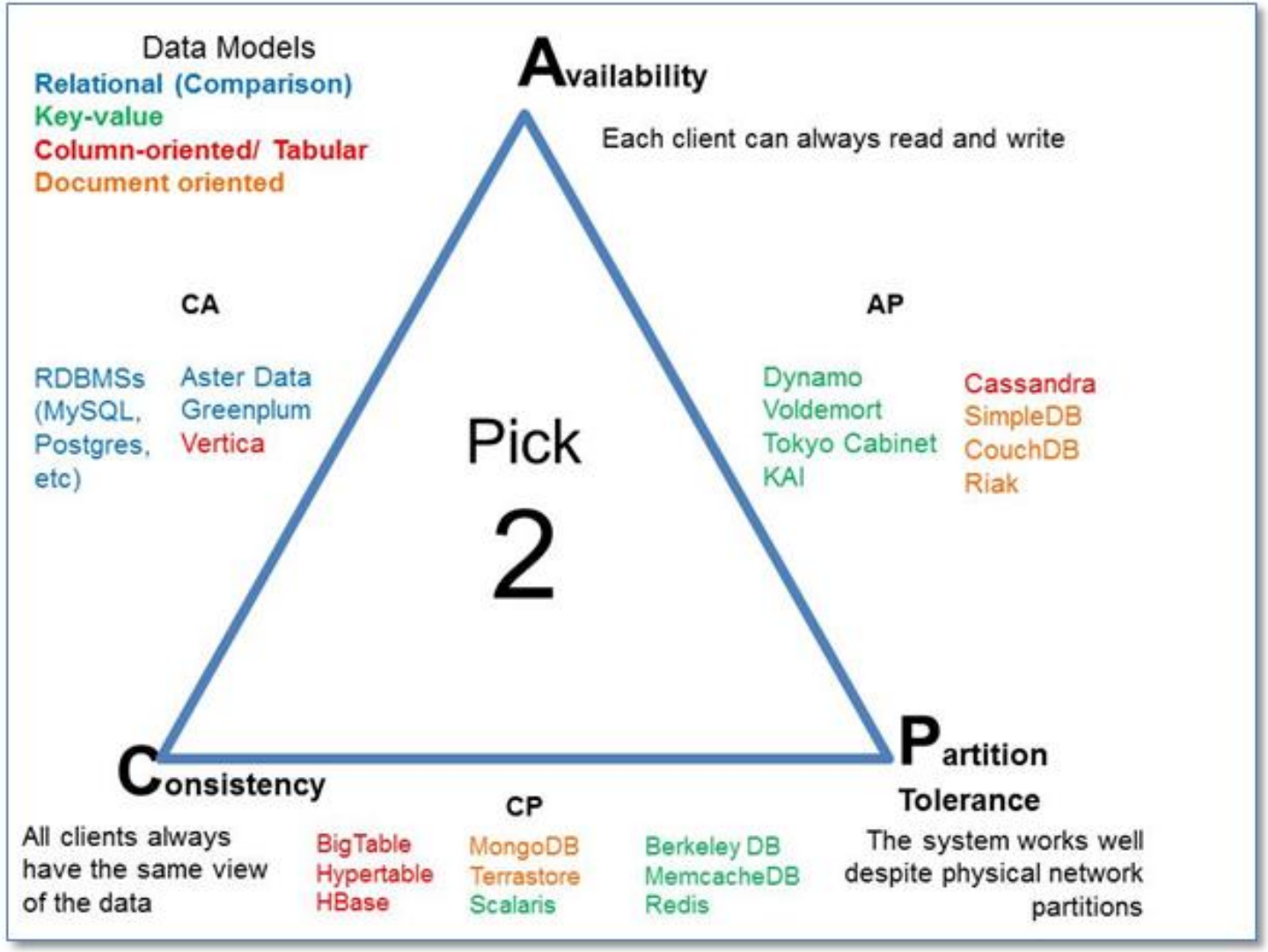
3PC


INIT
Commit
Vote-request
WAIT
Vote-abort
Global-abort
ABORT
Vote-commit
Prepare-commit
PRECOMM
Ready-commit
Global-commit
COMM
(a)
Vote-request
Vote-abort
INI
Vote-request
Vote-commit
REAL
Global-abort
ACK
ABORT
Prepare-commit
Ready-commit
PRECOM
Global-commit
ACK
COM
(b)
Know that all
have committed

Il Teorema del CAP sui sistemi distribuiti

- Nella seconda metà degli anni ‘90, Eric Brewer formulò il cosiddetto Teorema del CAP:
  - è impossibile per un archivio di dati distribuiti fornire simultaneamente più di due delle seguenti tre garanzie:
    - coerenza (consistency): ogni nodo ad ogni istante vede gli stessi dati;
    - disponibilità (availability): ogni richiesta riceve una risposta;
    - tolleranza alle partizioni (partition tolerance): il sistema continua a funzionare nonostante un numero arbitrario di messaggi venga perso (o ritardato);
  - Le alternative di fronte ad un errore (con conseguente partizionamento della rete) sono:
    - annullare l'operazione e quindi ridurre la disponibilità, ma garantire la coerenza;
    - procedere con l'operazione e quindi fornire disponibilità, ma rischi di incoerenza.
  - In altre parole, in presenza di una partizione di rete, si deve scegliere tra coerenza e disponibilità.

Il Teorema del CAP



Il Teorema del CAP