Fault Tolerance nei sistemi informatici

Alcuni concetti sulla probabilità

- **Sia P(A) la probabilità che un evento A accada in un dato periodo di tempo**
- **P(A) è tra 0 e 1 e 1- P(A) è la probabilità che l'evento A non accada nel periodo**
- **Due eventi A e B sono indipendenti se l'occorrenza di uno non influenza l'occorrenza dell'altro, allora:**

P(A and B)= P(A) x P(B) ***(entrambi accadono)***

P(A or B)= P(A)+ (1- P(A)) x P(B) ***(uno solo accade)***

= P(A) + P(B) – P(A)xP(B) ≈ P(A) + P(B)

Alcuni concetti sulla probabilità

- **Se, ad esempio, la P di un sistema di guastarsi in un giorno è 0.01, la P della rete è 0.02 e la P del terminale è 0.03 allora che un evento dei tre accada in un giorno è 0.06, mentre che accadano tutti nello stesso giorno è $6x10^{-6}$.**
- **I guasti sono memoryless quando la P non è influenzata da cosa è accaduto prima, questo è approssimativamente vero per componenti in operazione lontani da quello che è considerato il tempo di vita (dopo un certo tempo il componente si usura (wear out) e la P aumenta)**
- **Non sempre i guasti sono indipendenti: la rete non determina un guasto sulla CPU mentre la ventola si.**

Alcuni concetti sulla probabilità

- **Poiché le P sono troppo piccole si usa al loro posto il MT(A) cioè il tempo medio di occorrenza dell'evento *(mean time to event)* : MT(A) =1/P(A)**

  Nell'es. precedente MT(sistema) è 100 giorni, MT(rete) è 50 e MT(terminale) è 33 allora MT(almeno uno) è 17, mentre MT(tutti) è 170000.

- **Attenzione, se i tre componenti lavorano insieme MT(almeno uno) è il MT globale ! Quindi per alzare il MT globale bisogna alzare la qualità di tutti.**

  MT(G) = 1/(P(A)+P(B)+P(C)) =

  1/(1/MT(A)+1/MT(B)+1/MT(C))

  se gli MT di N componenti sono uguali allora:

  MT(G)= MT(A)/N

Alcune definizioni

Un sistema può essere visto come un singolo modulo o composto da moduli più piccoli che a loro volta sono composti da moduli. Queste definizioni si applicano ricorsivamente…

Ogni modulo ha un *specified behavior* e un *observed behavior*. Un *failure* (fallimento) avviene quando l'observed behavior si scosta dallo specified.

Un failure avviene a causa di un *error* o un difetto di un modulo. La causa dell'error è un *fault*. Ad es. un errore di programmazione è un fault (BUG) che può rimanere *latente* ma quando le istruzioni errate sono eseguite con certi dati l'errore diventa effettivo e si genera il failure.

BUG: parola inventata da: Grace Hopper (1906-1992)

Ufficiale della US Navy (Ammiraglio)

Importantissimi contributi nello sviluppo dei Compilatori e del COBOL.

E' chiamata "Admiral of the Cyber Sea"

La US Navy le ha dedicato una nave

Alcune definizioni

Il comportamento di un modulo si può alternare tra lo stato di *service accomplishment* quando si comporta come specificato e *service interruption* quando devia dal comportamento specificato.

***Timing failure* si verifica quando il sistema risponde correttamente ma con ritardo superiore a quello specificato.**

Il tempo nello stato di service interruption si compone di:

failure detection latency, report, correct or repair,

per poi ritornare nello stato di service accomplishment.

Alcune definizioni

Reliability di un modulo: misura il tempo tra un istante iniziale fino al prossimo failure.

Si quantifica statisticamente come MTTF *(mean time to failure)* se il modulo (componente) termina la sua vita e non viene riparato. L'istante iniziale corrisponde al tempo di installazione.

Si quantifica statisticamente come MTBF *(mean time between failures)* se il modulo viene riparato. L'istante iniziale corrisponde al tempo di ripresa del servizio.

Service interruption si quantifica come MTTR *(mean time to repair)* .

Availability (disponibilità) di un modulo: misura il rapporto tra il tempo di service accomplishment rispetto al tempo totale osservato e si quantifica come: MTBF/(MTTF+MTBR).

System availability: frazione del carico di lavoro processata con tempo di risposta accettabile.

Alcune definizioni

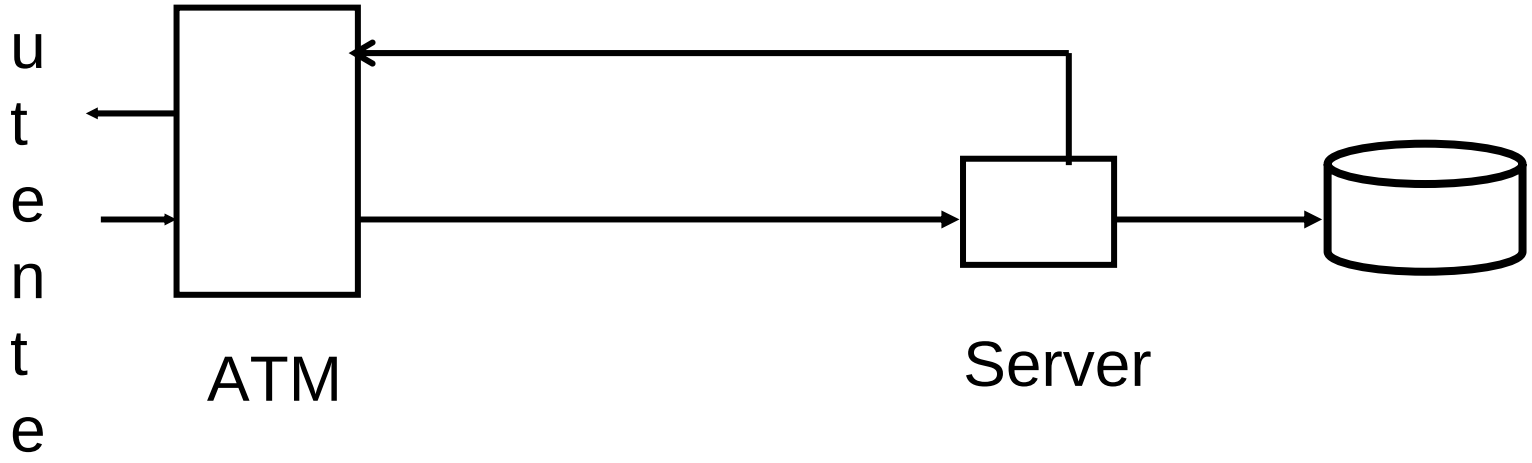
La classe corrisponde a: $\log_{10}(1/(1\text{-Availability})$

Tipo di sistema	Unavailability Minuti/anno	Availability	Classe
Unmanaged	52560	90%	1
Managed	5256	99%	2
Well managed	526	99.9%	3
Fault tolerant	53	99.99%	4
Haigh avail.	5	99.999%	5
Very high avail.	.5	99.9999%	6
Ultra avail.	.05	99.99999%	7

Banca fault tolerant

In un sistema Bancomat il modulo più debole è l'ATM

(automatic teller machine: lettore di card- distributore di denaro). Se l'ATM ha una disponibilità di circa il 99% (oltre ai tempi morti di rifornimento) sarà in servizio per 99 giorni e per uno sarà in riparazione (classe 2) . Se un utente lo usa due volte a settimana (circa 100 volte all'anno) avrà un denial of service una volta all'anno.

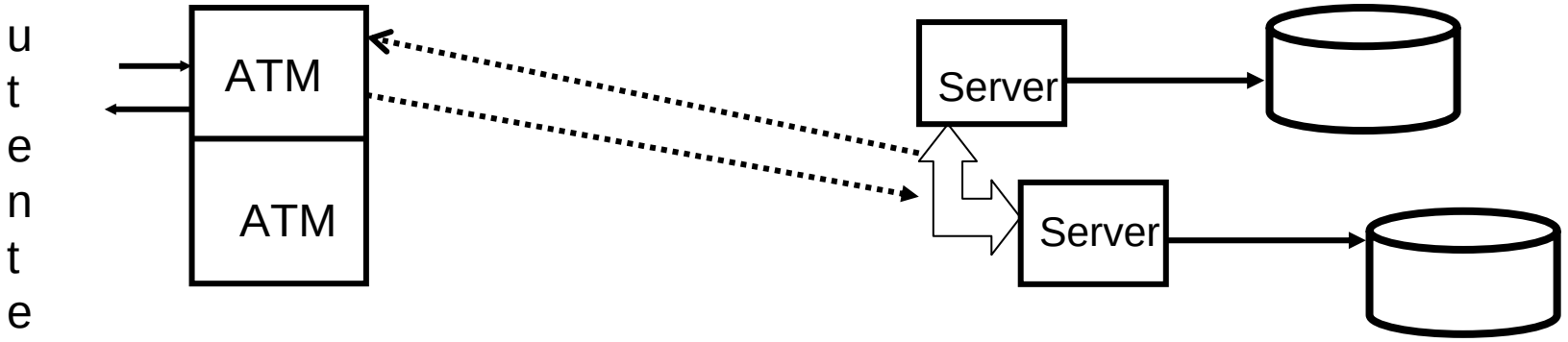


Banca fault tolerant

Con due ATM in parallelo si avrà almeno un Atm in servizio per 9999 giorni (classe 4) e l'utente avrà il denial ogni 100 anni.

Il sistema nel suo complesso è costoso più del doppio e con 10000 ATM ce ne sarà sempre uno guasto ogni giorno.

A questo punto la availability del doppio ATM è superiore a quella della rete e del server. Si può pensare di raddoppiare la rete e duplicare il server.

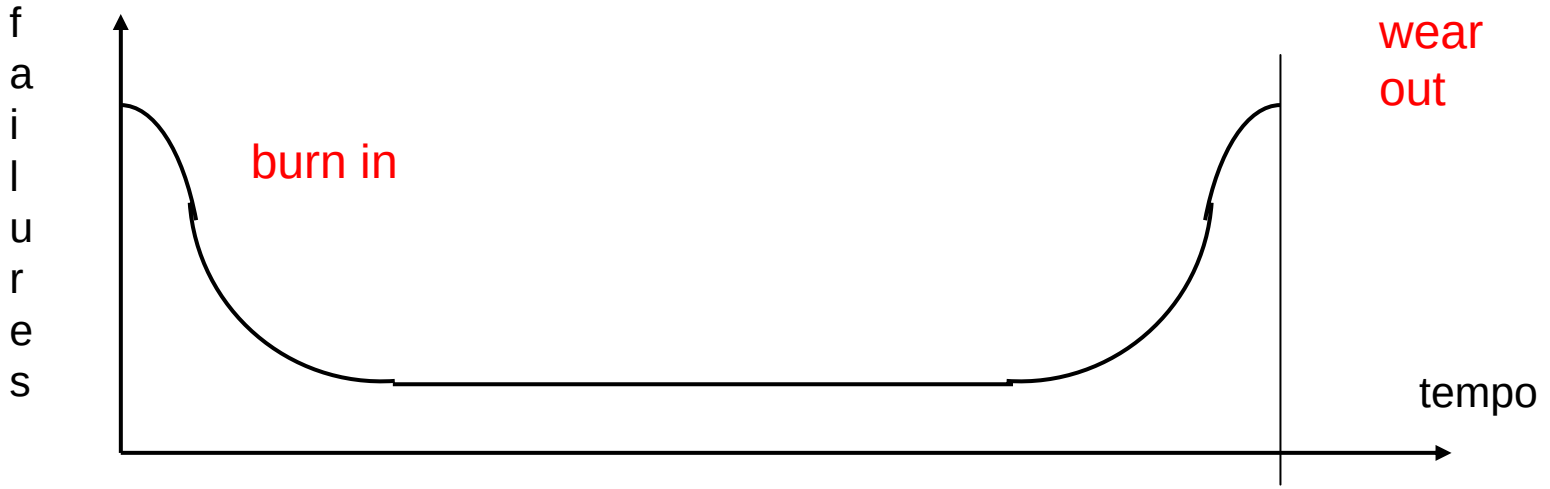


Fault avoidance

La reliability si accresce migliorando progetto e costruzione, adoperando componenti di buona qualità. Ma non basta a eliminare la probabilità di guasto.

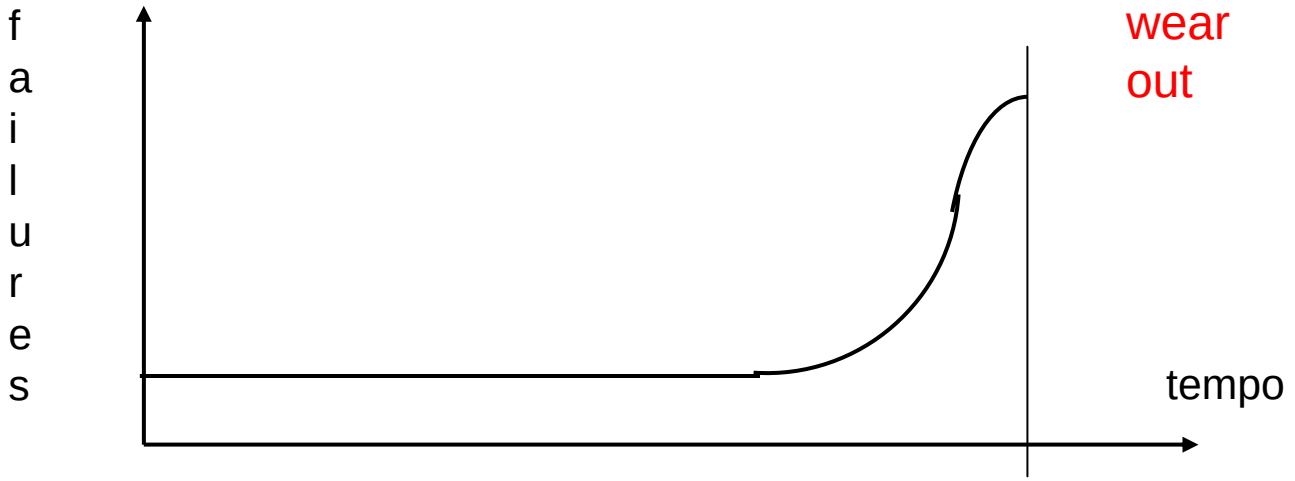
Prima di far entrare in servizio il modulo si effettua la validazione con prove in *positivo* e in *negativo* per un certo tempo eliminando buona parte dei problemi.

Ma i guasti si verificano poi in servizio fino a che il componente non si usura. Anche il SW si usura dopo un elevato numero di correzioni e manutenzioni.

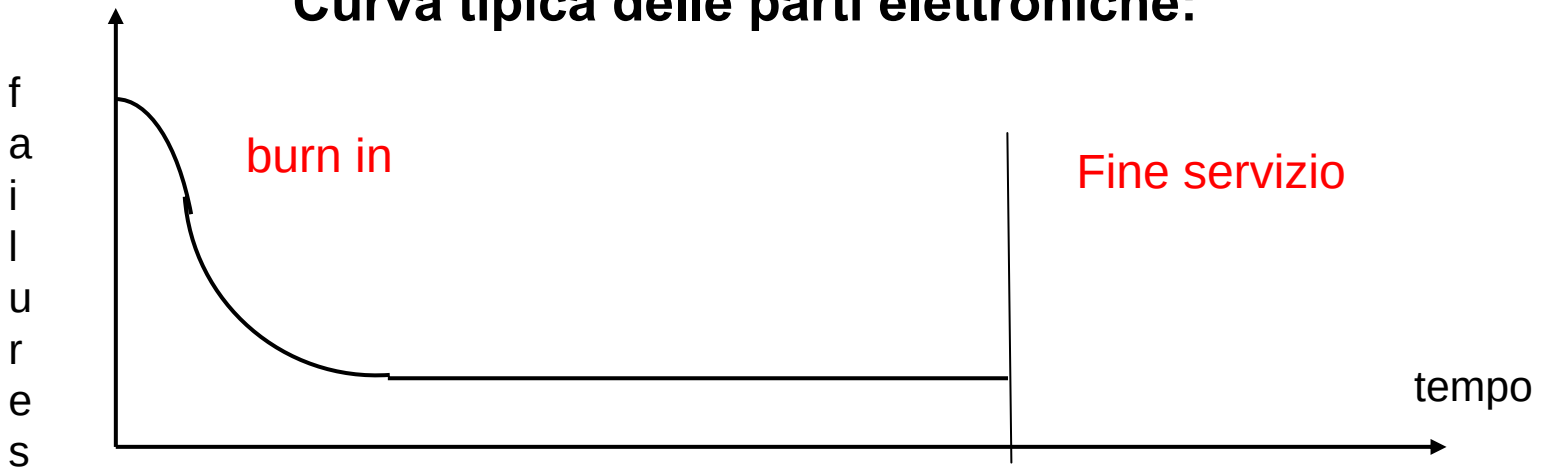


Fault avoidance

Curva tipica delle parti meccaniche:



Curva tipica delle parti elettroniche:



Fault avoidance

il Software:

Ad ogni nuova installazione vengono utilizzate parti nuove del software

manutenzione

La manutenzione è di due tipi:

Manutenzione correttiva per correggere il funzionamento insoddisfacente di un sistema. In pratica si tratta di riparare un malfunzionamento mediante sostituzione di componenti o lavorazione su componenti.

Manutenzione preventiva che tende a prevenire il funzionamento insoddisfacente di un sistema. Intervenendo quando ancora il sistema fornisce prestazioni soddisfacenti a scadenze di tempo o di uso.

ammodernamento

Ammodernamento: modifica o sostituzione per avvenuta obsolescenza di componenti, modifica di requisiti, cambio di normative, cambio di vincoli di legge.

L' Obsolescenza tecnica è causata dalle difficoltà e dai costi per trovare componenti di vecchia tecnologia, rispetto dell'ambiente, oppure personale in grado di effettuare manutenzione e correzione .

correzione

La correzione è di due tipi:

Latent error processing: tenta di individuare errori latenti prima che si verifichino con la manutenzione preventiva.

Effective error processing: corregge gli errori dopo che si sono manifestati. Può essere ottenuta con:

Error masking: usa informazioni ridondanti per ricostruire lo stato corretto (Error correcting codes usati in trasmissione e memorizzazione), usa componenti ridondanti mettendo fuori servizio i componenti guasti e continuando con quelli efficienti (dischi mirrored, RAID, moduli in duplex, sistemi resilienti).

correzione

Il Fuori servizio(Outage o denial of service) può essere causato da fattori esterni o interni:

HW e SW: faults.

Environment: problemi nella machine room, impianto di raffreddamento, interruzione delle linee di comunicazione, interruzione di potenza elettrica, fulmini, fuoco, terremoto, allagamenti, sabotaggi, atti di guerra.

Operazioni: amministrazione di sistema, riconfigurazioni, attacchi, spamming con conseguente timing fault.

Manutenzione: manutenzione riparazione, installazione di componenti hw e sw.

Processo: scioperi, decisioni del management, errori del personale, sabotaggio interno.

MTTF/MTBF tipici

Connettori e cavi: MTTF di circa 1000 anni. I fault si manifestano subito nelle fase di burn in. L'MTTF si riduce moltissimo a causa dell'environment: corrosione, piegature, fenomeni di fatica, riscaldamento, agenti atmosferici.

Circuiti, piastre, cpu: MTTF da 5 a 20 anni.

Dischi: MTTF da 5 a 20 anni a seconda della qualità. Si fa largo uso di ECC per mascherare zone "macchiate" o sciupate del disco. I RAID superano i 50 anni.

Dischi: MTBF tipici

Tipo di error	MTBF	Recovery	Conseguenze
Soft data read error	>1 ora	Retry o ECC	no
Recoverable seek error	>6 ore	Retry	no
Maskable hard data read error	6 giorni	ECC	Rimappa su un nuovo settore, riscrive i dati buoni
Unrecoverable data read error	1 anno	no	Rimappa su un nuovo settore, i dati sono persi
Riparazione	5 anni	riparazione	Dati non disponibili, qualcosa si salva
Miscorrected read data error	10^7 anni	no	Legge dati sbagliati

MTTF/MTBF tipici

Workstations: varia enormemente a seconda della qualità dei componenti. Il mercato è estremamente competitivo e molti costruttori economizzano. Le maggiori debolezze si hanno sul software (MTBF di pochi mesi) mentre per display, hardware e dischi si ha un MTTF 5-10 anni.

Software: anche un buon software non è esente da faults : circa 3 design faults per 1000 linee di codice. La maggior parte dei questi provocano soft bugs che possono essere mascherati da restart e retry. Il rapporto tra soft e hard bugs è tipicamente dell'ordine del 100:1.

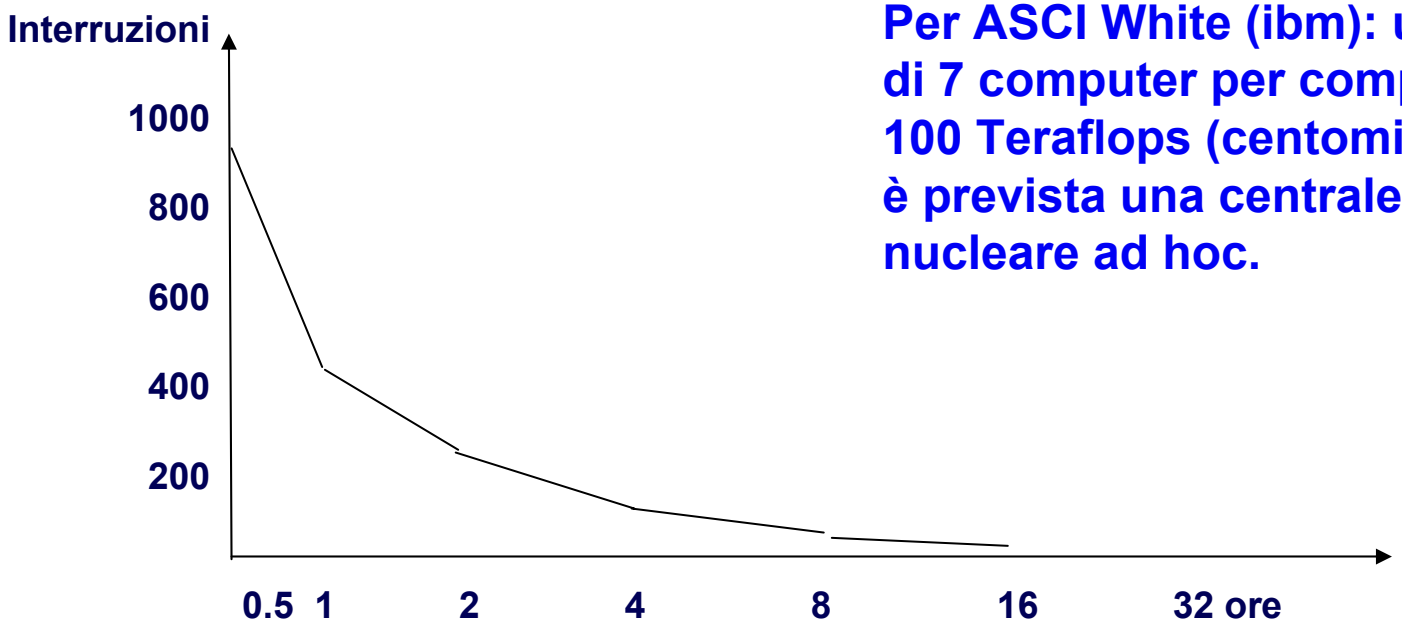
MTTF/MTBF tipici

LAN: cavi e fibre presenteno un BER (bit error rate) di 10^{-9} e 10^{-6} messaggi errati per ora. Gran parte dei problemi sono dovuti a sovraccarico ed errori di protocollo. IL MTBF è di circa 3-4 settimane.

Sistema di comunicazione (CE e USA): il BER con le fibre è del tipo di quello delle LAN aggravato da moduli intermedi che lo portano a 10^{-5} - 10^{-6} . Se un messaggio è lungo un kbit si ha un error rate da 1:100 a 1:1000, generalmente questi sono soft e si correggono. Gli hard error si verificano generalmente a gruppi in periodi delimitati a temporizzazione saltuaria. Una buona availability si ottiene con il duplexing delle linee su percorsi distinti.

MTTF/MTBF tipici

Energia elettrica: l'interruzione dell'energia è un problema serio che deve essere combattuti adeguatamente con batterie tampone nei piccoli sistemi e con gruppi di continuità nei sistemi più grandi. Esistono statistiche regionali sulle quali basarsi:



Per ASCI White (ibm): un sistema di 7 computer per complessivi 100 Teraflops (centomila miliardi!) è prevista una centrale nucleare ad hoc.

failfast

I faults possono essere Hard o Soft.

Un hard fault provoca il malfunzionamento del modulo che dopo un certo tempo viene individuato provocando il failure (fault latency). Il modulo ritorna a funzionare correttamente dopo la riparazione (o sostiuzione).

Un soft fault viene riparato o mascherato durante il funzionamento del modulo.

Un modulo è Failfast (failstop) se si ferma immediatamente dopo il fault.

Il comportamento failfast è importante perché:

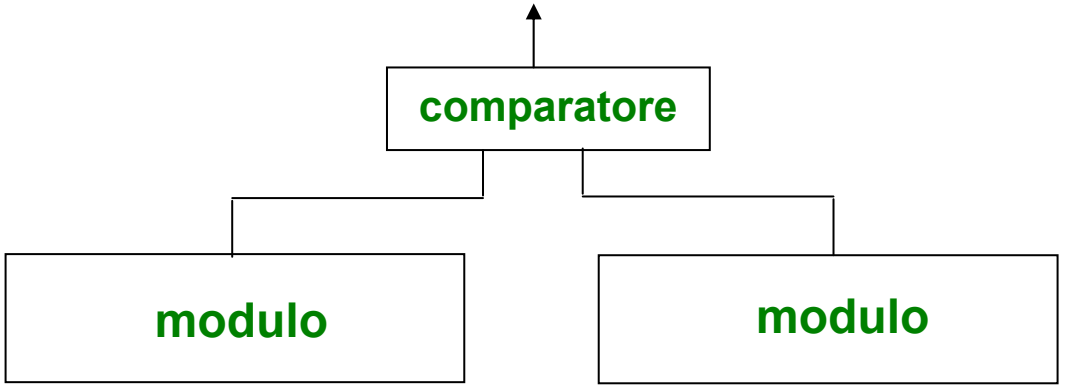
Permette l'individuazione immediata del fault con techiche più semplici ed efficaci di riparazione o di mascheramento.

Impedisce altri fault a cascata nel sistema.

Come costruire moduli failfast?: l'idea N-plex.

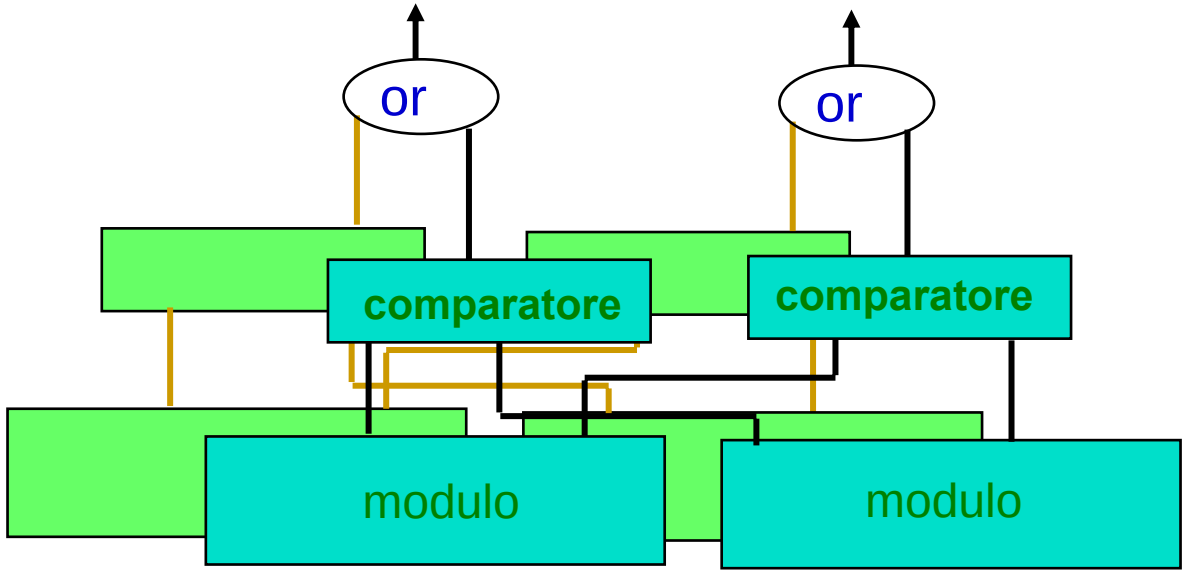
Il progetto più semplice è mettere in Duplex due moduli uguali (duplexing, pairing). L'uscita dei due viene mandata ad un Comparatore che se le uscite sono diverse interrompe il funzionamento del duplex.

Anche se un duplex si ferma (fails) due volte più spesso non c'è latenza e il SW per la fault tolerance è più semplice perche l'unica classe di failure è STOP.



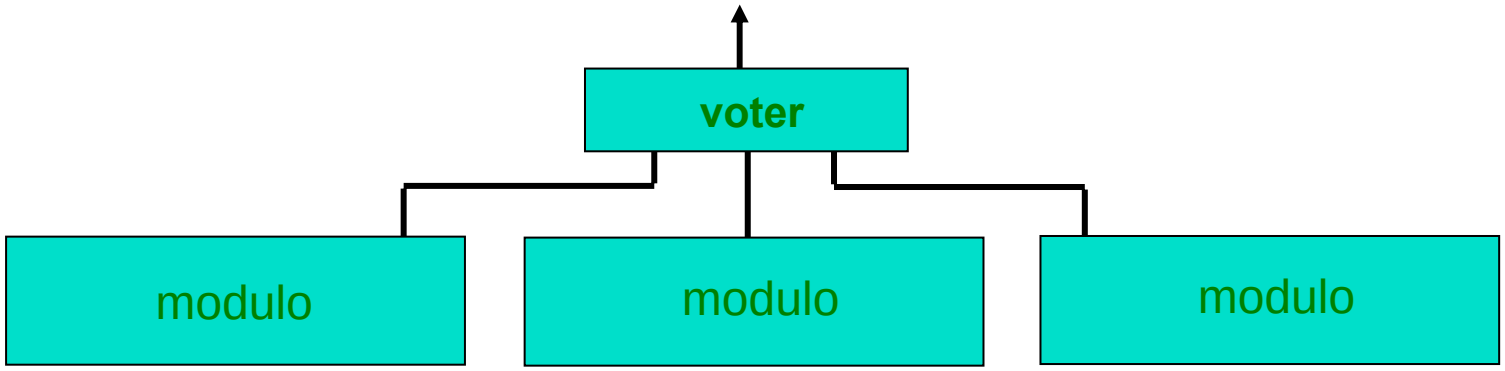
2-plex

Il duplex suggerisce un progetto fault tolerant che consente elevata availability. Si duplica anche il comparatore che potrebbe guastarsi e si usano 4 moduli



3-plex.

Un altro progetto semplice consiste nell'usare 3 moduli uguali, mandando le 3 uscite ad un Voter che tollera un fault andando in failfast se le tre uscite sono diverse.



Se moduli sono più numerosi il voter si affiderà alla maggioranza. N=10 è il caso limite per non eccedere in eccessiva complicazione, e, ovviamente, costo e numero di faults.

N-plex.

Failvote Voter: se un modulo non risponde come la maggioranza il voter lo ignora fino a che non viene riparato e si affida alla maggioranza di quelli che fino a quel momento si sono comportati come la maggioranza.

Con il criterio della maggioranza un sistema 10-plex si ferma quando 5 moduli sono considerati guasti. Con il Failvote voter il sistema può arrivare fino a 3 funzionanti.

Failfast Voter : consiste nell'usare moduli uguali di tipo failfast mandando le uscite ad un voter che verifica soltanto se i moduli rispondono o no.

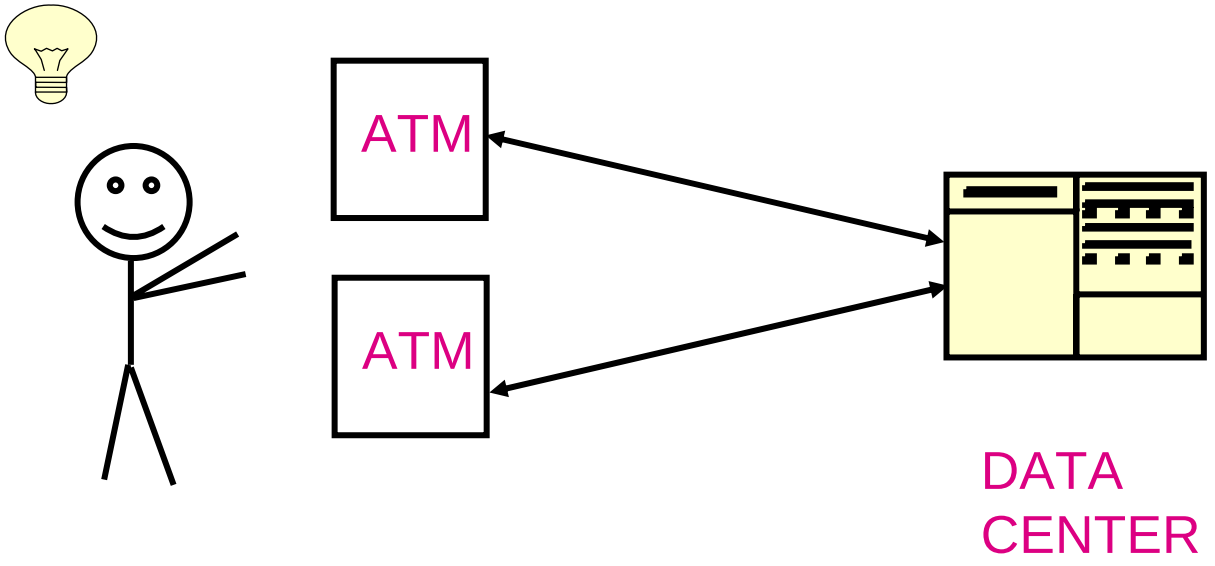
E' il caso della coppia dei dischi "mirrored"(due dischi con dati esattamente replicati) e in questo caso è possibile fornire servizio anche con un disco guasto o in manutenzione.

Poiché i moduli sono failfast un sistema10-plex failfast voter funziona fino a che almeno 1 modulo funziona.

failfast

Anche il Voter si può guastare. Si può quindi fare l'n-plex dei voter, però chi garantisce l'ultimo voter?

Nel caso degli ATM l'ultimo voter è l'utente che, se gli ATM sono failfast allora si rivolge direttamente all'ATM funzionante:



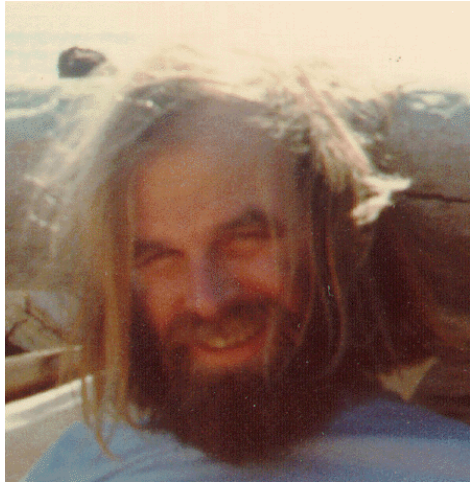
Sistemi transazionali e non

Le Transazioni di DB ACID (atomicity, consistency, isolation, durabilty) sono un'ottima garanzia ma in caso di problema la computazione non procede oltre e fa un recovery lento.

N-version programming, usato nel SW strategico: vengono scritte più versioni dello stesso programma e vengono usate come in un sistema N-plex. E' ovviamente un sw molto costoso.

Le Transazioni ACID: Jim Gray

James Nicholas Gray (1944- 2007)



IBM, Microsoft, Google (Google Earth)

Byzantine fault tolerance

Un fault Bizantino è un fault arbitrario (cioè non facilmente riproducibile e quindi non corretto dalla manutenzione) che capita durante l'esecuzione di un programma in un sistema distribuito. Generalmente è dovuto a step di algoritmo non eseguito, step eseguito in modo scorretto, esecuzione di uno step non previsto.

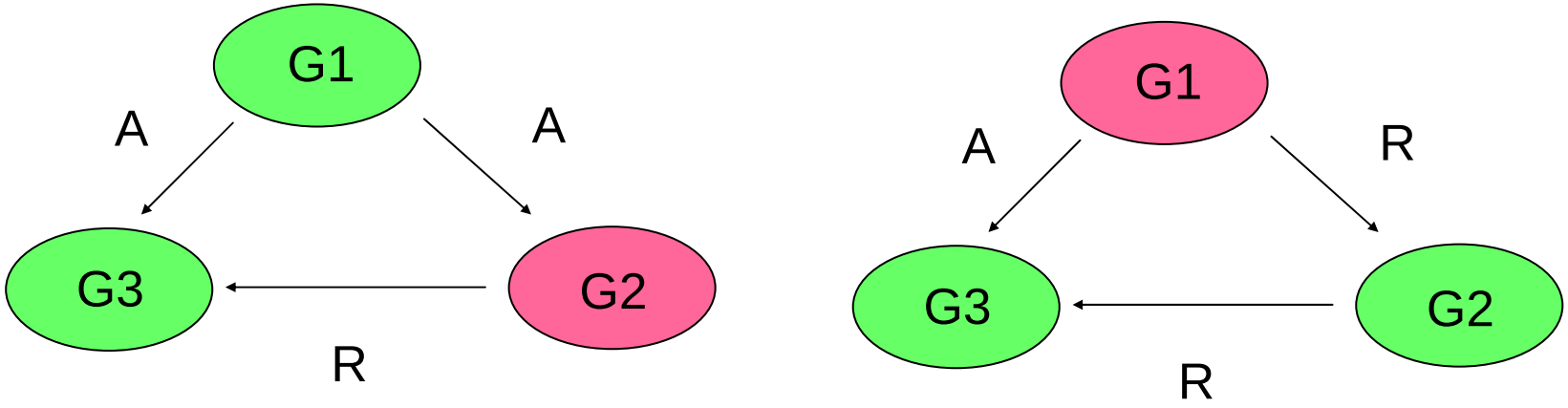
Le cause possono essere dovute a: errori del sistema operativo, errori del compilatore, scorretta inizializzazione di variabili, guasti introdotti da manutenzione scorretta, attacchi al sistema, uso malizioso del sistema.

Byzantine fault tolerance

In sistemi “non transazionali” i componenti del sistema si devono confrontare per assumere decisioni comuni o non contraddittorie su come procedere (controllo di impianti , sistemi di sicurezza, controllo di auto, aerei etc..)

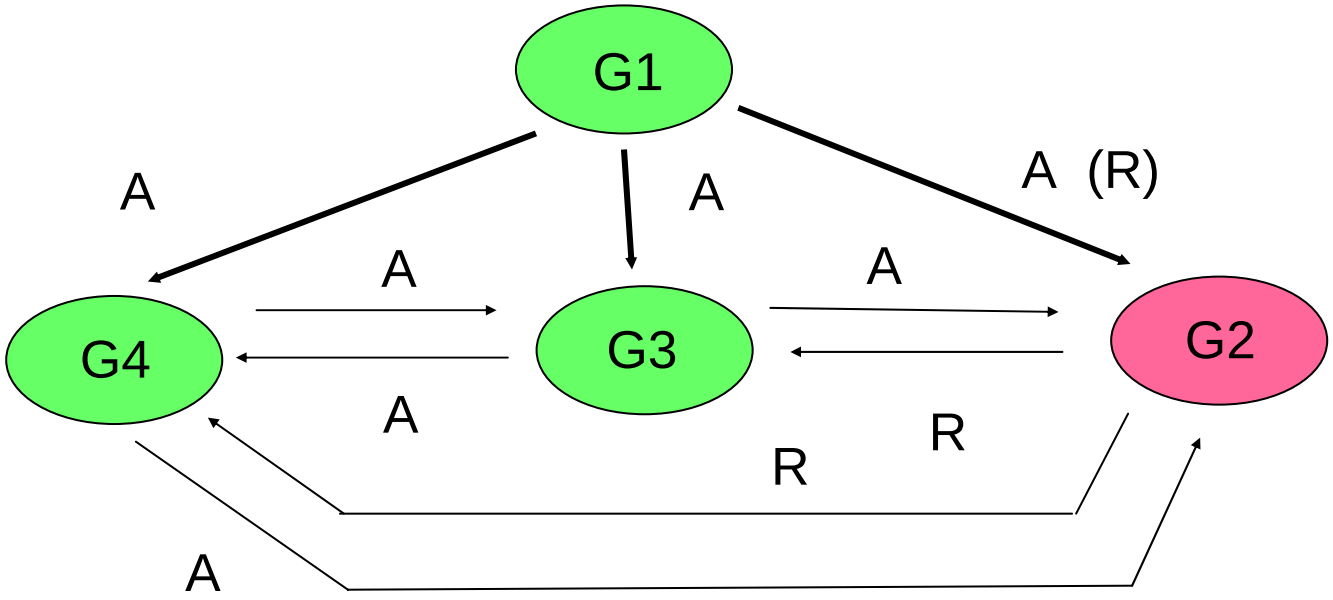
I generali di Bisanzio scoprirono e dimostrarono che sono necessari 3m+1 generali per difendersi da m generali traditori!

Byzantine fault tolerance



G3 si accorge del tradimento ma non sa cosa fare.
G3 non sa se il traditore è G1.
Si può dimostrare che il sistema a tre non ha soluzione per scoprire il traditore.

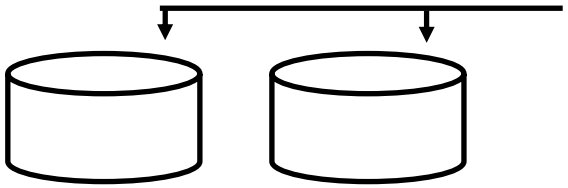
Byzantine fault tolerance



G3 e G4 si accorgono del tradimento di G2
e agiscono come ordinato da G1.
Se il messaggio A da G1 a G2 viene cambiato in R
G2 è comunque considerato traditore.

DISCHI: parallelismo e sicurezza

Disk Mirroring : dischi con dati replicati



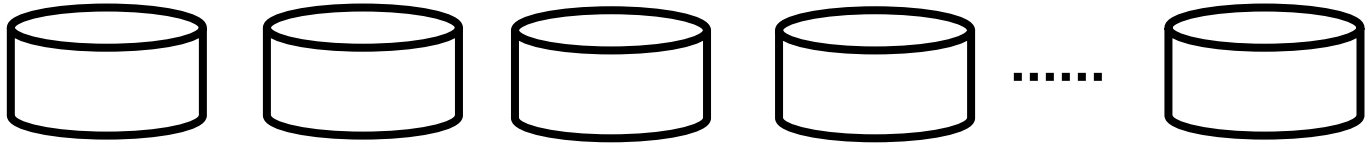
letture indipendenti
scritture su entrambi

- **aumentando il numero di dischi aumenta la probabilità di averne uno guasto (diminuisce il MTBF)**
- **diminuisce la probabilità di perdita dei dati dovuta al guasto contemporaneo (aumenta il MTTF)**

la ridondanza è utile!

dischi RAID

Redundant Array of Inexpensive Disks



- **architettura che migliora le prestazioni e l'affidabilità del sistema di mem. permanente**
- **l'uso di N dischi consente di suddividere i dati in piccoli blocchi da scrivere e leggere in parallelo**
- **informazioni ridondanti consentono la correzione di errori dovuti a guasti**

correzione degli errori

CODICI A RILEVAZIONE E CORREZIONE DI ERRORE

Data una parola binaria $C = (c_1, c_2, \ldots, c_k)$ (dove ogni c_i è un bit) è possibile effettuare un controllo di presenza di errore aggiungendo un bit c_{k+1} in modo tale che:

$$c_1 \oplus c_2 \ldots \oplus c_k \oplus c_{k+1} = 0.$$

dove $\oplus$ indica la somma modulo 2 .
c_{k+1} si chiama *bit di parità* poiché se i bit a 1 in C sono in numero pari c_{k+1} è = 0 , altrimenti è =1.

Se la parità non torna vuol dire che almeno un bit è errato.

correzione degli errori

Questo sistema intercetta, quindi, tutti le C con un bit errato senza però dire quale è.
Gli errori su più bit possono compensarsi.

Esistono sistemi migliori che però usano più bit di ridondanza per il controllo degli errori.

Molto usato è il codice Reed Solomon sia in trasmissione dati che in memorizzazione.

Gestione delle copie

La replicazione dei dati è un ingrediente fondamentale dei sistemi informativi

motivazioni:

- **efficienza**
- **affidabilità**
- **autonomia**
- **controllo**

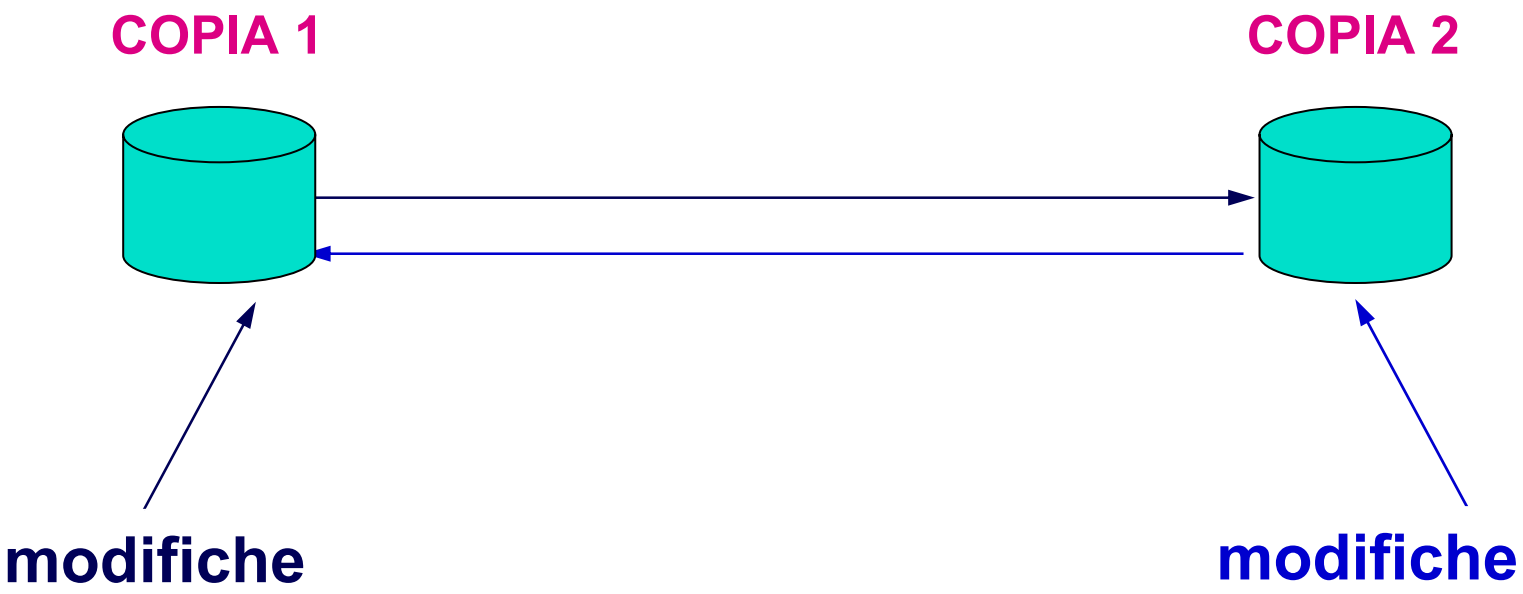
Modalità di replicazione

- **asimmetrica**

modifiche

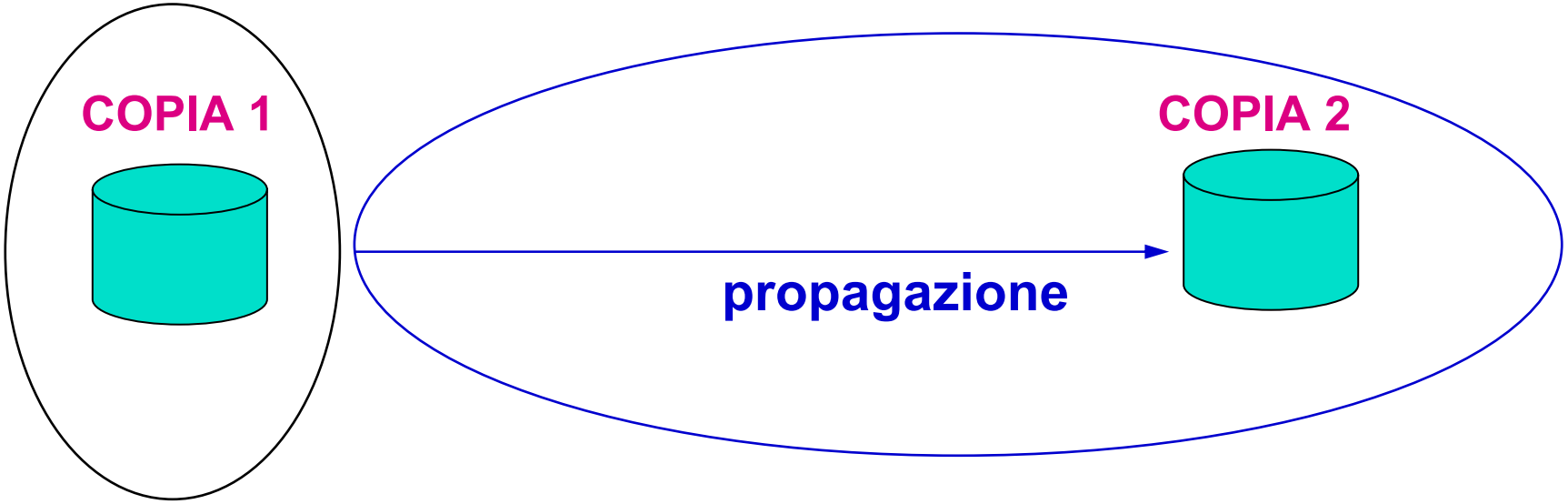
Modalità di replicazione

simmetrica



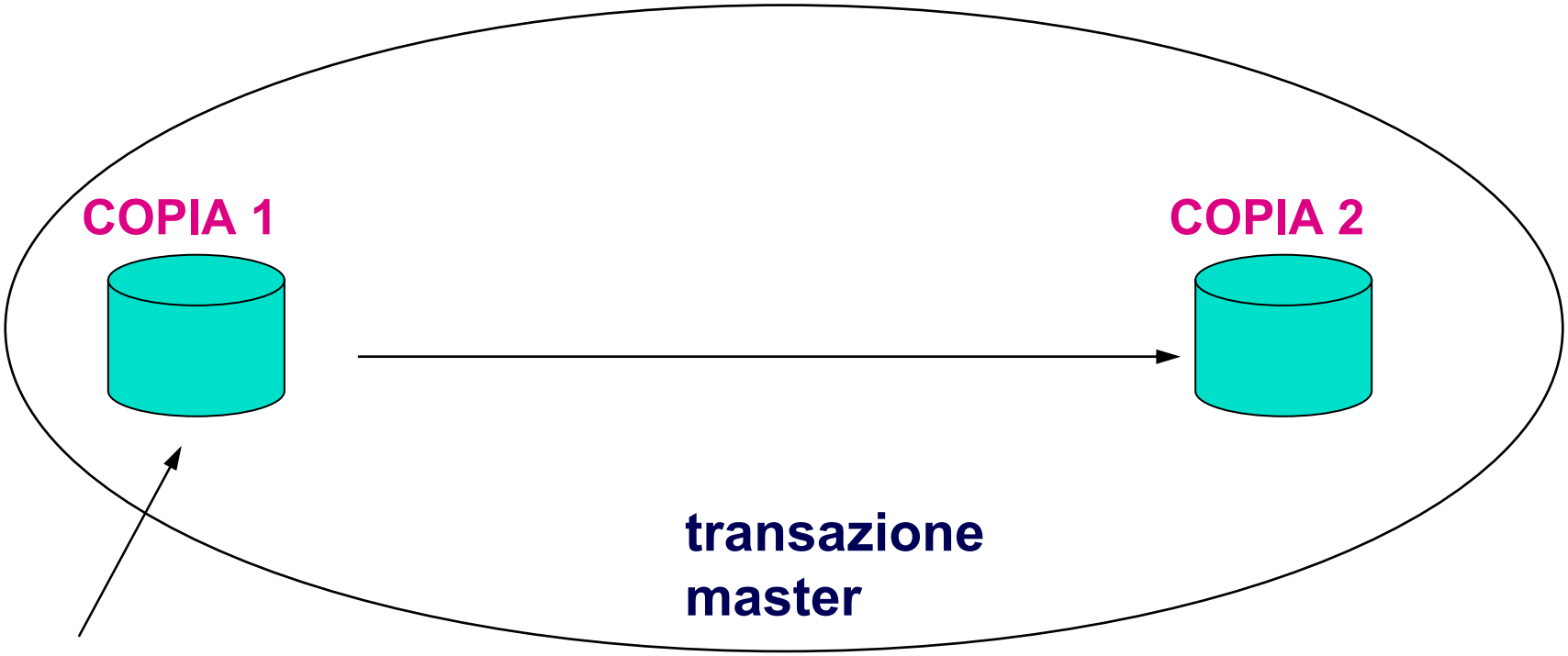
Modalità di trasmissione delle variazioni

- **trasmissione asincrona**



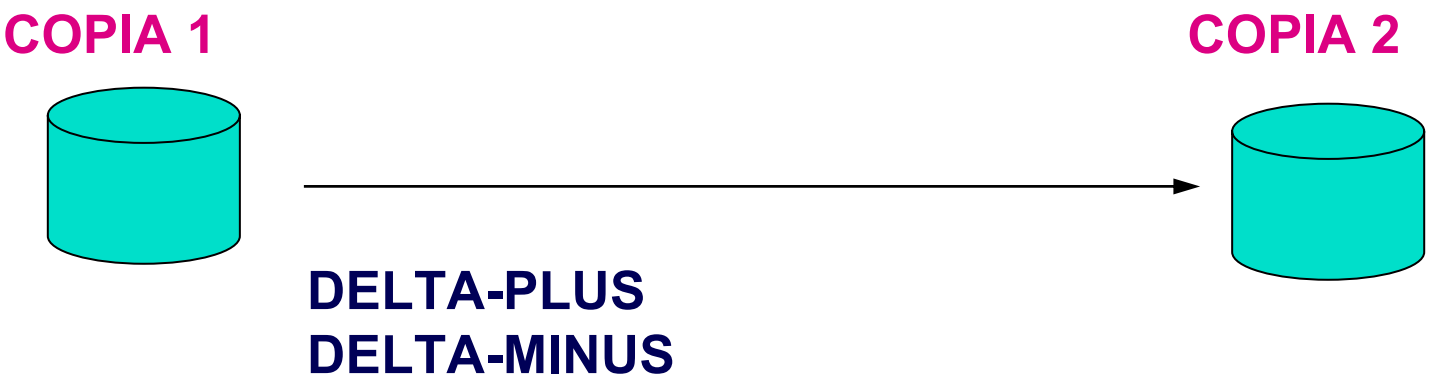
Modalità di trasmissione delle variazioni

- **trasmissione sincrona**



Modalità di allineamento

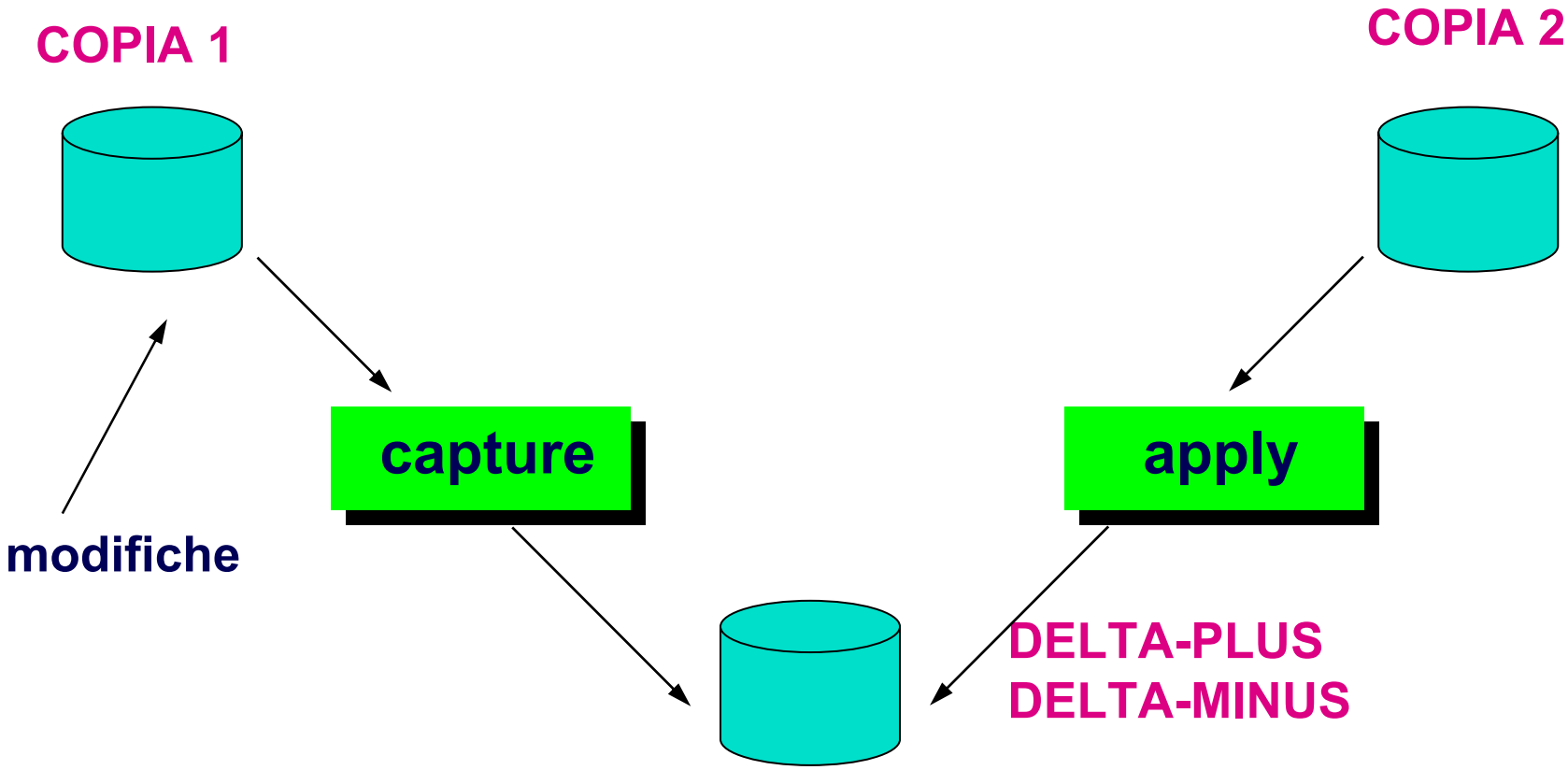
- **refresh incrementale**



Meccanismi per la replica

asimmetrica, asincrona e incrementale

due moduli : CAPTURE, APPLY



Applicazione

a comando

- **periodica**
- **guidata dalle modifiche**

Un caso particolare: replica in computer mobili

- **computer mobili : saltuariamente collegati ad una rete**
- **copie disconnesse per ore o giorni intere, poi riconnesse (riconciliazione)**
- **applicazione : agenti di vendita mobili**

Il problema più grosso è il SW

E' possibile costruire SW error free?

Forse si: è questione di tempo e denaro?

Business SW costa da 10 a 100 $ per linea di codice,

System SW costa circa da 100 a 1000$ per linea

SW per avionica , militare e Shuttle è arrivato a 5000$ per linea, a causa di un design più attento ed una attività di test molto accurata.

Chi ha più denaro?: US government

ma ha abbastanza tempo? …No ..quindi il SW perfetto non esiste.

Il problema più grosso è il SW

Perfino istruzioni semplici possono rivelarsi non corrette

Questa funzione è corretta?

Long add(long a, long b) {return (a+b;)}

No! : fa la somma modulo long size, (oppure va in overlow..)

Attenzione anche alle :

allocazioni dinamiche,

le ricorsioni(!)

le variabili non definite o non ben definite

gli array

Tipi di errori software

Secondo la terminologia definita dalla IEEE esistono tre tipi di errori software:

Error: errore umano nel processo di interpretazione delle specifiche, nel risolvere un problema o nell'usare un metodo;

Fault: difetto del codice sorgente (detto anche BUG);

Failure: comportamento del software non previsto dalle specifiche.

Famosi errori software

Sonda Mariner 1: distruzione in volo (1962)

Causa: un punto invece di una virgola in una istruzione Fortran…

Missili patriot: sbagli su missili iracheni (1991)

Causa: frequenti moltiplicazioni per 1/10 con variabili in virgola fissa di 24 bit e quindi accumulo di errori di troncamento.

Ariane 5: esplosione in volo (1996)

Causa: riutilizzo di software ben testato per Ariane 4 , alcune variabili di 16 bit non erano state riallocate in 64 bit.

Tecniche di Testing

Unit testing: il test unitario è la verifica di una singola funzionalità del sistema in esame. Lo sviluppatore stesso scrive questo tipo di test per verificare la correttezza del codice prima di rilasciarlo agli altri componenti del team di sviluppo. Normalmente questo tipo di test è automatizzato ogni volta che si modifica una singola funzionalità del sistema

Tecniche di Testing

Integration testing: il test di integrazione verifica che tutte le parti di un sistema funzionino correttamente. Questo tipo di test viene effettuato per verificare che il codice nuovo non interferisca con le funzionalità già presenti e funzionanti di un sistema . Normalmente questo tipo di test è automatizzato e viene eseguito periodicamente(ad ogni integrazione, ogni notte ogni settimana, a seconda dei casi).

Tecniche di Testing

System testing: il test di sistema consiste nel verificare l'intero prodotto per il rilascio di una versione al cliente. Il test comprende la verifica di tutte le componenti del prodotto che può comprendere più sistemi.

Tecniche di Testing

Valori di test:

Un ponte può essere testato facendo passare un camion del peso uguale alla portata massima ….

Il SW NO!

Se volessimo testare in modo esaustivo a=b+c a 16 bit

dovremmo generare 2^{32} coppie b c.

Quindi non si fa, si studiano i valori di test che possono validare il SW

Tecniche di Testing

Black box testing con generazione di molti input.

White box testing se si può osservare l'implementazione. Molto complesso riservato a specialisti

FINE