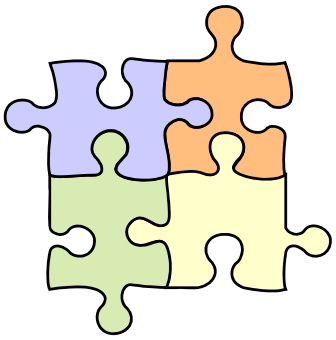
Luca Cabibbo



Architetture Software

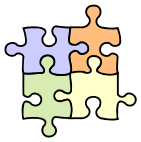
Tattiche architetturali
(seconda parte)

Dispensa ASW 311

ottobre 2014

Il pezzo più affidabile di un aereo
è quello che non c'è –
perché non serve.

(DC9 Chief Engineer)



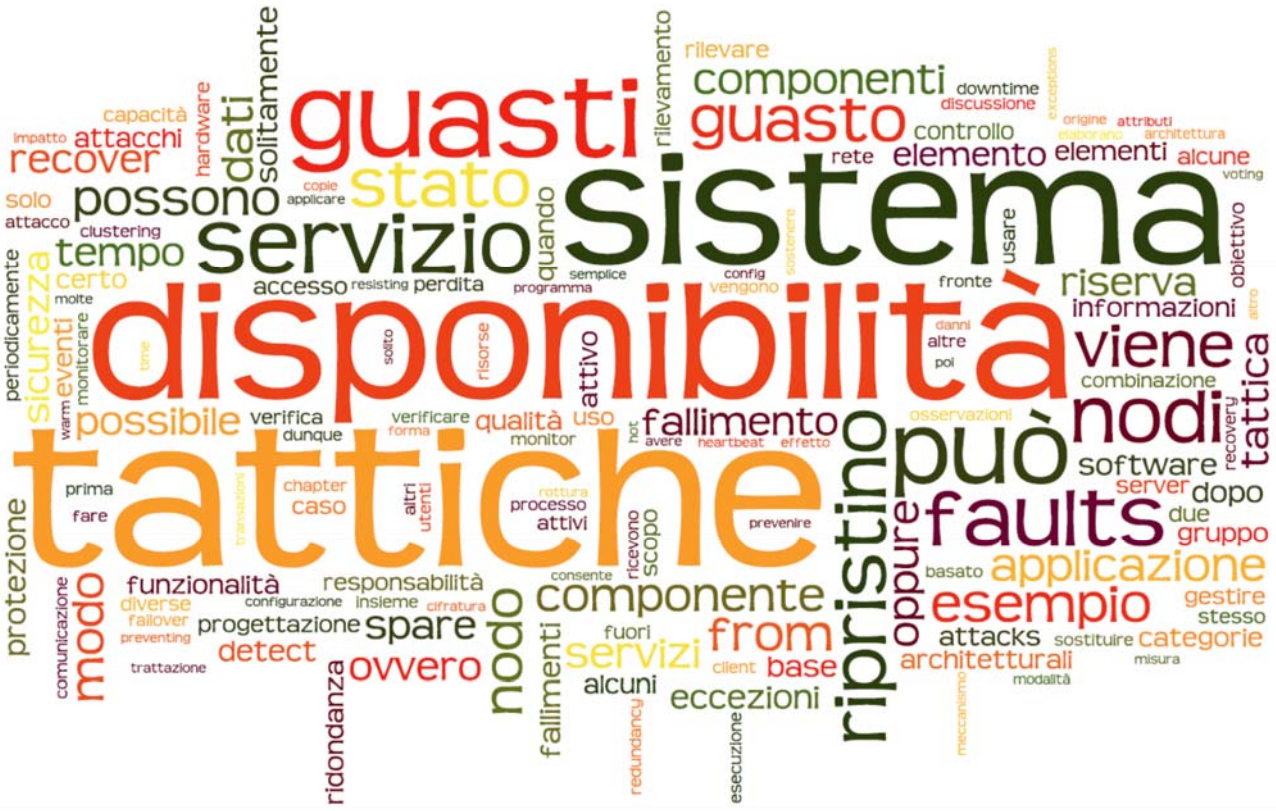
- Fonti

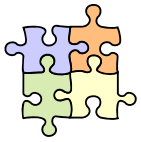
- [SAP] Part 2, Quality Attributes
  - Chapter 4, Understanding quality attributes
  - Chapter 5, Availability
  - Chapter 7, Modifiability
  - Chapter 8, Performance
  - Chapter 9, Security
- [Parnas] On the Criteria To Be Used in Decomposing Systems into Modules, Communications of the ACM, 15, 1972
- [Bachmann, Bass, Nord] Modifiability Tactics, Technical report CMU/SEI-2007-TR-002, 2007
- [Scott, Kazman] Realizing and Refining Architectural Tactics: Availability, Technical report CMU/SEI-2009-TR-006, 2009
- [Kim, Kim, Lu, Park] Quality-driven architecture development using architectural tactics, Journal of Systems and Software, 82, 2009

- Obiettivi e argomenti

- Obiettivi
  - introdurre le tattiche architetturali come guida per il raggiungimento di attributi di qualità
  - illustrare alcune tattiche architetturali
- Argomenti
  - *tattiche architetturali*
  - *tattiche per le prestazioni*
  - *tattiche per la modificabilità*
  - tattiche per la disponibilità
  - tattiche per la sicurezza
  - discussione

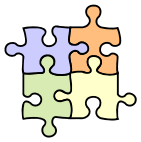
- Wordle





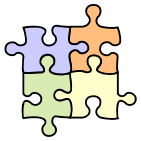
* Tattiche per la disponibilità

- La *disponibilità* (*availability*) si riferisce alla proprietà di un sistema software di essere lì e pronto a svolgere i compiti che gli sono richiesti
  - questa proprietà riguarda sia l'*affidabilità* (*reliability*) – ovvero la qualità del sistema di essere coerente con la sua specifica – che il *ripristino* (*recovery*) – ovvero, la capacità del sistema di "ripararsi" a fronte di guasti



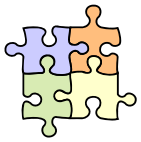
Disponibilità

- La *disponibilità* è interessata ai guasti e ai fallimenti di un sistema – nonché al ripristino da guasti e fallimenti
  - un *fallimento* si verifica quando il sistema non eroga più un servizio in modo coerente con la sua specifica
    - i fallimenti sono osservabili esternamente dagli utenti del sistema
  - un *guasto* è una problematica interna al sistema – ad es., la rottura di un disco o un nodo, o un errore di programmazione
  - un fallimento è causato da un guasto (o una loro combinazione)
  - anche se ogni guasto ha la potenzialità di causare un fallimento, non sempre un guasto produce un fallimento
    - un sistema può essere *tollerante* a un certo guasto (o una loro combinazione) se il verificarsi di quel guasto non provoca un fallimento del sistema
  - un altro aspetto rilevante è il *ripristino* da fallimenti



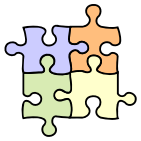
Tattiche per la disponibilità

- In questa trattazione consideriamo *tattiche per la disponibilità* che hanno l'obiettivo di
  - impedire ai guasti di provocare fallimenti
  - o, comunque, di limitare l'effetto del fallimento e rendere possibile il ripristino



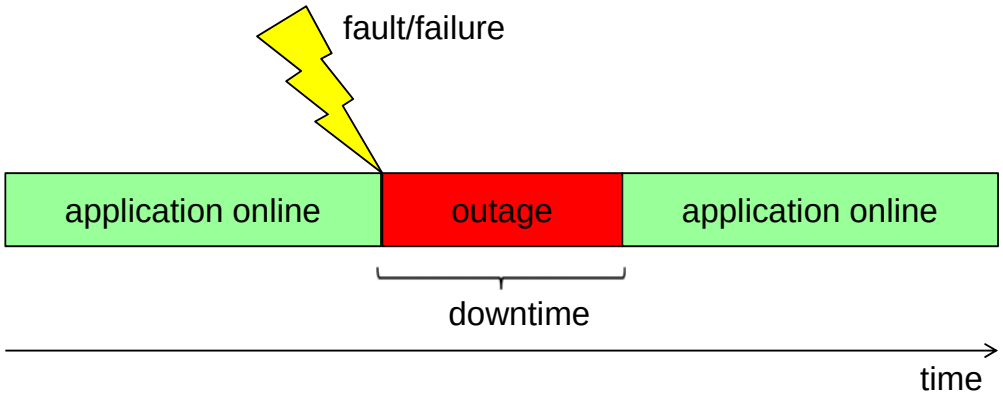
- Che cosa è la disponibilità

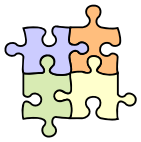
- La *disponibilità* può essere definita [SSA] come
  - la capacità di un sistema di essere completamente o parzialmente funzionante, come e quando richiesto
  - anche a fronte di guasti di componenti del sistema
    - in modo che eventuali guasti non comportino un fallimento totale dell'intero sistema
    - oppure che comportino un fallimento dal quale è possibile un recupero – preferibilmente entro tempi concordati
    - in alcuni casi, può essere accettabile (per un certo periodo di tempo) un funzionamento "parziale" del sistema



Fallimenti

- Un *fallimento* si verifica quando il sistema non eroga più un servizio in modo consistente con la sua specifica
  - si dice che c'è un'*interruzione* (*outage*) di servizio
  - il periodo di tempo in cui il sistema è fuori servizio per un outage è chiamato un *downtime*



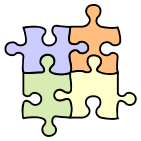


Misura della disponibilità

- La misura della *disponibilità* è definita come un rapporto percentuale

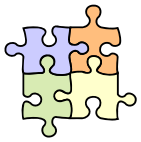
$$\text{disponibilità} = \frac{\text{periodo di tempo in cui il sistema è operativo}}{\text{periodo di riferimento}}$$

Uptime (%)	Downtime (%)	Downtime per year	Downtime per week
98%	2%	7.3 days	3:22 hours
99%	1%	3.65 days	1:41 hours
99.8%	0.2%	17:30 hours	20:10 minutes
99.9%	0.1%	8:45 hours	10:05 minutes
99.99%	0.01%	52.5 minutes	1 minute
99.999%	0.001%	5.25 minutes	6 seconds
99.9999%	0.0001%	31.5 seconds	0.6 seconds



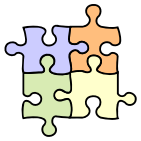
Che cosa è la disponibilità

- Un'altra caratterizzazione della *disponibilità* [SAP]
  - la capacità di un sistema di mascherare o riparare guasti
  - in modo tale che la durata complessiva delle interruzioni di servizio non ecceda un valore richiesto nell'ambito di un certo intervallo di tempo



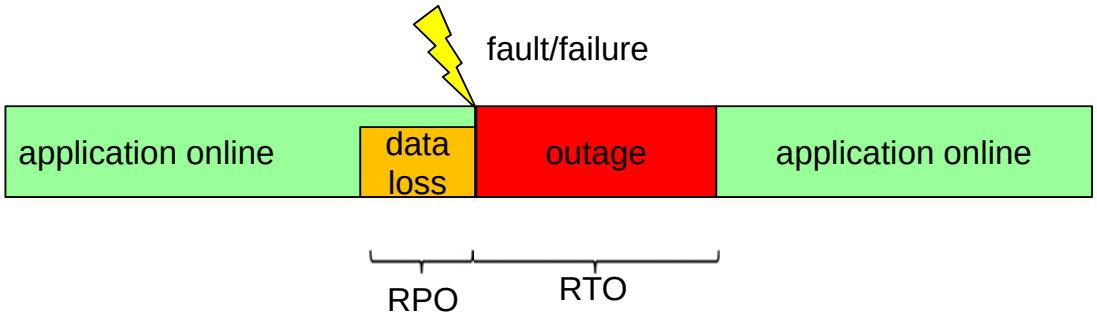
Osservazioni

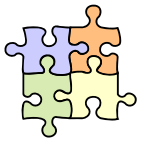
- In pratica, la caratterizzazione della disponibilità di un sistema software è molto più complessa – ad esempio, infatti
  - il sistema può offrire molti servizi, e questi servizi possono fallire in modo indipendente tra loro
  - servizi diversi possono avere requisiti di disponibilità differenti
  - ogni singolo servizio può avere requisiti di disponibilità diversi in momenti differenti (ad es., periodi della giornata o dell'anno)
  - alcuni servizi possono fallire in modo parziale – ovvero, in caso di guasti, può essere accettabile erogare il servizio in modalità "ridotta"
  - normalmente il sistema deve anche prevedere dei periodi di fuori servizio per consentire la manutenzione ordinaria del sistema – chiamati *downtime pianificati*
  - ...



Osservazioni

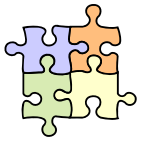
- Un fallimento può comportare, oltre alla perdita temporanea di funzionalità, anche la perdita di dati "all'indietro nel tempo"
  - ad es., tutte le modifiche fatte dopo l'ultimo backup dei dati
  - gli obiettivi di disponibilità per un sistema possono comprendere allora anche
    - un tempo massimo (in avanti) per il ripristino delle funzionalità del sistema – *Recovery Time Objective* (*RTO*)
    - un tempo massimo (all'indietro) di perdita di dati – *Recovery Point Objective* (*RPO*)





Osservazioni

- Per *business continuity* (*BC*) si intende
  - la capacità di un'organizzazione di continuare a esercitare il proprio business a fronte di guasti o eventi catastrofici
  - la *disponibilità tecnologica* è solo una componente della BC
- La quantità di disponibilità richiesta per un sistema può essere determinata sulla base di un'opportuna *valutazione economica*
  - valuta il costo delle interruzioni di servizio
    - a secondo del sistema, un'interruzione di servizio può avere sia conseguenze dirette – ad es., perdita di funzionalità di business, dati, tempo, danni a cose o persone, ... – che indirette – ad es., perdita di clienti, perdita di reputazione, multe, ...
  - sulla base di questa informazione, determina quanto sei disposto a spendere per proteggere il sistema da queste interruzioni di servizio

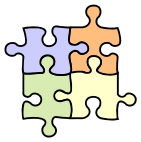


Misura della disponibilità

- Con riferimento all'hardware, la *disponibilità* di un componente può essere calcolata come

$$\text{disponibilità} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

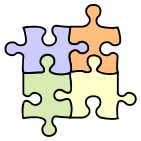
  - *MTBF* = *Mean Time Between Failures* = inverso del numero di guasti per unità di tempo
    - una proprietà del componente
  - *MTTR* = *Mean/Maximum Time To Repair or Resolve*
    - tempo medio/massimo impiegato dal fornitore per riparare un guasto al componente dal momento in cui gli viene notificato



Misura della disponibilità

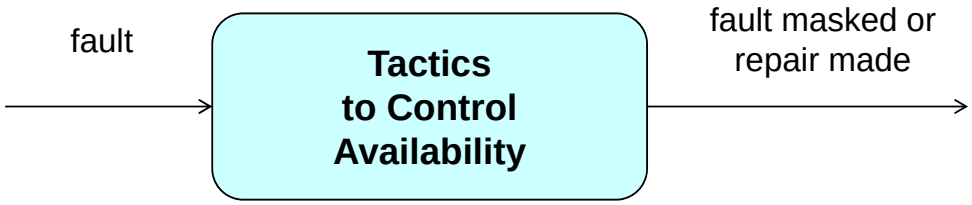
$$\text{disponibilità} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

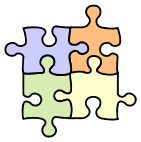
- Questa caratterizzazione della disponibilità ci offre un'intuizione importante su come sia possibile agire sulla disponibilità
  - da una parte, è possibile intervenire sull'MTBF – ovvero, utilizzare componenti più affidabili
  - dall'altra, è possibile intervenire sull'MTTR – ovvero, applicare meccanismi di ripristino più efficienti



- Tattiche per la disponibilità

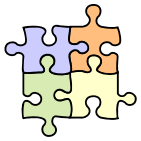
- In questa trattazione consideriamo tattiche per la disponibilità per impedire ai guasti di provocare fallimenti, o per limitare l'effetto dei guasti e rendere possibile il ripristino





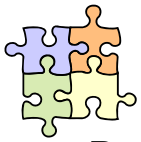
Osservazioni

- Molte tattiche per la disponibilità hanno l'obiettivo di eliminare *punti di fallimento singoli*
  - per questo, di solito comprendono qualche forma di *ridondanza* – ovvero, la presenza di più "copie" di uno stesso componente
  - il componente in questione può essere un elemento hardware – ad es., un nodo di calcolo, un tratto di rete, una scheda di rete, un elemento di storage o un alimentatore – oppure un componente software (funzionalità oppure dati)
  - è anche possibile "ridondare" più componenti
  - in caso di componenti software, le diverse "copie" possono essere diversificate – ad es., funzionalità implementate con algoritmi diversi, oppure dati rappresentati in forme differenti
  - inoltre, di solito è necessario aggiungere ulteriori elementi o responsabilità per gestire queste "copie" multiple – ad es., un meccanismo di replicazione per mantenere allineate le diverse copie di stessi dati



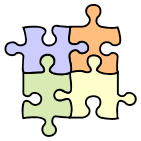
Osservazioni

- Molte delle tattiche per la disponibilità che saranno discusse nel seguito sono presenti in ambienti di esecuzione standard – come sistemi operativi, infrastrutture software e middleware come application server e database server
  - è importante comprendere tali tattiche per considerare esplicitamente (e talvolta individualmente) il loro effetto
  - il lavoro dell'architetto sarà poi spesso quello di scegliere e valutare (piuttosto che implementare) le tattiche per la disponibilità da applicare nella progettazione di un sistema
- Infine, è bene osservare che molte delle tattiche che saranno discusse nel seguito fanno riferimento principalmente alla disponibilità di nodi (hardware) che implementano servizi
  - talvolta – ma non sempre – possono essere applicate direttamente anche per aumentare la disponibilità dei dati, delle reti, dello storage



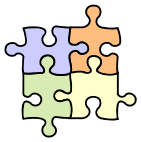
Gruppo di protezione

- Preliminarmente, definiamo un *gruppo di protezione* per un servizio S come un insieme di nodi di elaborazione (server) dedicati all'erogazione del servizio S
  - in un gruppo di protezione, in un dato momento
    - uno o più nodi sono *attivi* – ovvero, sono utilizzati per erogare effettivamente il servizio ai client del servizio
    - possono essere anche presenti altri nodi, che sono delle *riserve* ridondanti
  - il caso più semplice di gruppo di protezione è la ridondanza 1+1
    - un nodo attivo e un nodo di riserva
  - attenzione, una riserva per un servizio S potrebbe essere
    - completamente inattiva, oppure
    - impegnata nei confronti del servizio S – ma non utilizzata direttamente dai client del servizio – oppure
    - impegnata nell'erogazione di altri servizi S', S'', ...



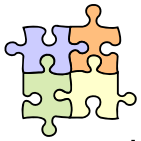
Ragionamento

- Progettazione per la disponibilità
  - viene identificato un albero dei guasti che si possono verificare
  - viene poi considerato ciascun ramo dell'albero (un guasto o combinazione di guasti) e, anche sulla base della gravità e dell'impatto sui servizi, viene identificata un'opportuna strategia per la gestione di tale guasto
  - attenzione, bisogna fare anche delle considerazioni complessive sulla disponibilità
- Gli approcci per la disponibilità sono solitamente rivolti a
  - rilevare guasti
  - gestire il ripristino da guasti
  - prevenire guasti



Categorie di tattiche per la disponibilità

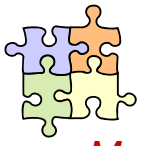
- Categorie principali di tattiche per la disponibilità
  - *detect fault*
    - tattiche che hanno lo scopo di rilevare guasti – per stabilire se e quando debba aver luogo un ripristino (magari automatico)
      - ad es., heartbeat
  - *recover from faults*
    - tattiche che hanno a che fare con il ripristino del sistema a fronte di guasti
      - ad es., clustering e failover
  - *prevent faults*
    - tattiche che hanno lo scopo di prevenire guasti
      - ad es., riavviare periodicamente un servizio



- Detect faults (1)

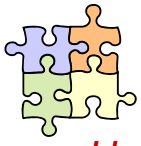
- Prima che un sistema possa intraprendere qualunque azione che riguardi un guasto (come un ripristino automatico), la presenza del guasto deve essere rilevata

- *Ping/echo*
  - una coppia di elementi (nodi/componenti/processi) si scambia in modo asincrono delle coppie di messaggi ping/echo – per determinare la raggiungibilità e il tempo di roundtrip del canale di comunicazione tra i due elementi
  - l'assenza di una risposta indica un problema nell'altro elemento – oppure nella rete di comunicazione
  - questa tattica può essere usata dagli elementi che appartengono ad un gruppo di protezione, che sono dunque mutuamente responsabili per un certo compito o servizio
    - ad es., per capire se e quando un nodo di riserva debba sostituire un nodo attivo



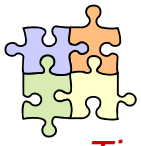
Detect faults (2)

- *Monitor*
  - un *monitor* è un elemento utilizzato per monitorare lo stato di salute del sistema – come la presenza di processi bloccati o fuori controllo, la congestione della rete o un attacco DoS in corso – sulla base di test automatici o l'applicazione di altre tattiche



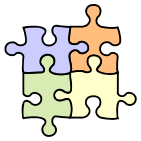
Detect faults (3)

- *Heartbeat*
  - un meccanismo di rilevamento dei guasti basato su uno scambio periodico di messaggi tra un monitor del sistema e un elemento (processo o componente) da monitorare
  - ad es., l'elemento da monitorare emette periodicamente un messaggio *heartbeat* – mentre il monitor lo ascolta
    - l'assenza di un heartbeat indica un problema nel primo elemento (o nella rete di comunicazione)
  - ad es., un *watchdog* è basato sull'uso di un contatore o un timer (hardware) che dovrebbe essere periodicamente resettato dall'elemento da monitorare
    - un valore "troppo alto" per questo contatore indica un problema in quell'elemento
  - la differenza con *ping/echo* è relativa a chi ha la responsabilità di iniziare le azioni per il controllo dello stato di salute



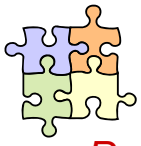
Detect faults (4)

- *Time stamp*
  - a ciascun evento/messaggio scambiato nel sistema viene assegnato un timestamp – per consentire di rilevare sequenze di eventi non corrette
- *Sanity checking*
  - verificare la validità o la ragionevolezza delle richieste o delle risposte relative a specifiche operazioni
  - viene applicata all'interfaccia tra elementi, per esaminare uno specifico flusso di informazioni – quando si ha conoscenza dello scambio atteso di informazioni tra elementi
- *Condition monitoring*
  - viene effettuato il monitoraggio di alcune condizioni, e convalidate delle ipotesi fatte durante la progettazione – ad esempio, il calcolo di checksum



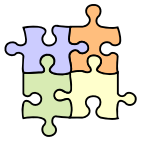
Detect faults (5)

- *Voting*
  - ci sono più componenti che ricevono gli stessi eventi e inviano le loro risposte a un componente che gestisce la votazione – questo componente è in grado di rilevare inconsistenze, che indicano guasti
  - criteri per lo scrutinio possono essere, ad es., “maggioranza” o “componente preferito”
  - poiché tutti i componenti ricevono e elaborano gli stessi eventi, lo stato di questi componenti è continuamente sincronizzato
  - nell’hardware, una realizzazione comune di questa tattica è la *triple modular redundancy* (TMR)
    - ad esempio, se un componente ha un error rate (inverso dell’MTBF) pari a 10^{-3}, allora l’applicazione di TMR porta a un error rate pari a circa $2 \cdot 10^{-6}$, ovvero circa 3 ordini di grandezza migliore



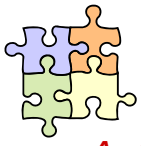
Detect faults (6)

- *Replication*
  - la forma più semplice di voting, in cui i componenti sono ciascuno cloni esatti degli altri
  - questa tattica è utile per rilevare guasti dell’hardware – ma non protegge da errori di progettazione o implementazione del sw
- *Functional redundancy*
  - una forma di voting, in cui i componenti sono progettati e implementati separatamente
  - il controllo della disponibilità del software richiede spesso di utilizzare meccanismi di diversificazione
- *Analytic redundancy*
  - viene effettuato una diversificazione anche degli input e degli output dei componenti che partecipano al voting – per tollerare errori relativi alla specifica dei componenti



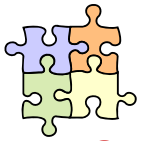
- Recover from faults

- Due categorie di tattiche per il ripristino da guasti
  - tattiche che preparano ed eseguono il ripristino
  - tattiche relative alla reintroduzione del componente che si è guastato – ed è stato "riabilitato"
  - la riparazione può essere automatica o richiedere un intervento manuale
    - la riparazione automatica richiede evidentemente anche l'applicazione di tattiche per il rilevamento di guasti



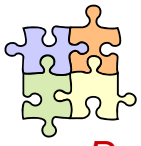
Recover from faults (1)

- *Active redundancy (hot spare)*
  - si riferisce a una configurazione in cui tutti i nodi (attivi o riserve) in un gruppo di protezione ricevono ed elaborano in parallelo gli stessi identici eventi – pertanto sono normalmente tutti nello stesso stato (ovvero, sono sincronizzati)
  - se si verifica un guasto in un nodo attivo, uno dei nodi di riserva lo può sostituire immediatamente, *poiché è già sincronizzato*
    - *il downtime del sistema può essere nell'ordine dei millisecondi*
  - il caso più semplice è la ridondanza 1+1
  - la sincronizzazione tra i nodi è automatica, poiché tutti i nodi ricevono ed elaborano gli stessi eventi – è però opportuno usare dei protocolli di comunicazione affidabili tra di essi
  - la ridondanza può essere anche nel canale di comunicazione (rete) e nelle unità disco (condivise)



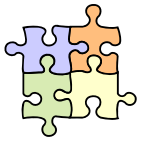
Recover from faults (2)

- *Spare (cold spare)*
  - si riferisce a una configurazione in cui i nodi di riserva di un gruppo di protezione sono tenuti fuori servizio fino a quando non è necessaria la sostituzione di un nodo attivo
    - i nodi di riserva sono configurati per poter sostituire altri nodi che potrebbero fallire
  - quando si verifica un guasto in un nodo attivo, il nodo di riserva viene avviato, eventualmente configurato, e il suo stato aggiornato/sincronizzato (vedi ad esempio *Rollback*) – solo allora può sostituire il nodo che è fallito
    - *il downtime del sistema può essere nell'ordine dei minuti*

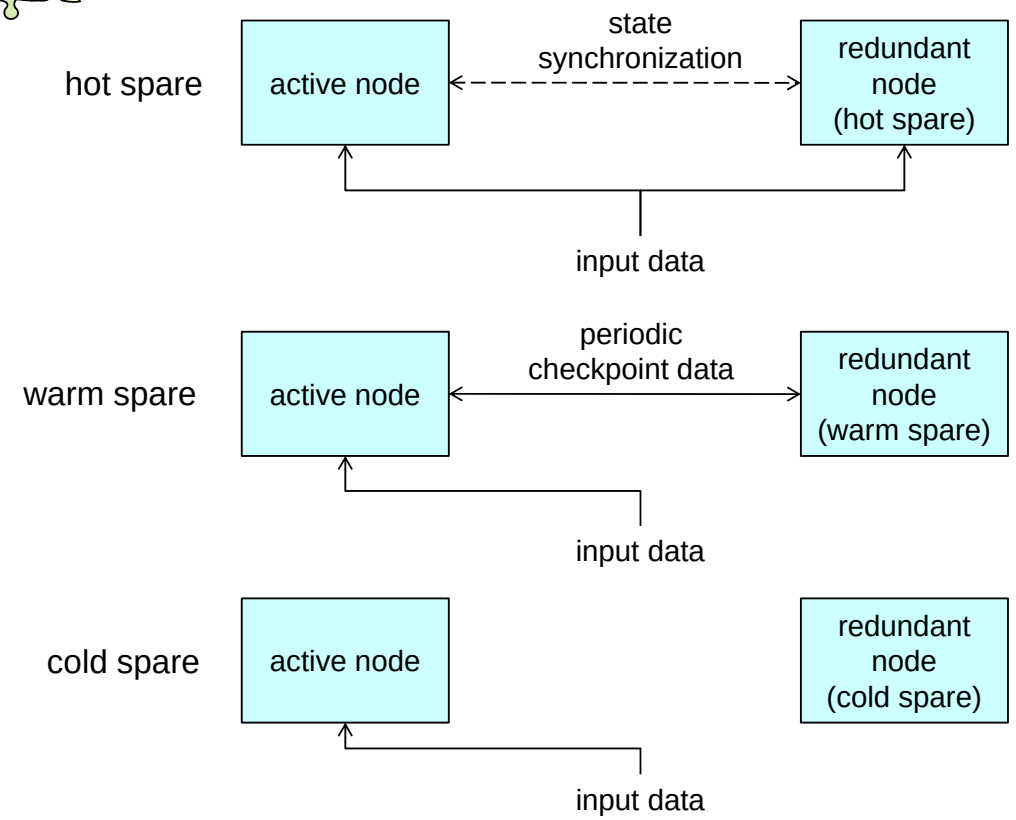


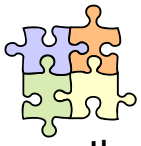
Recover from faults (3)

- *Passive redundancy (warm spare)*
  - solo i nodi attivi del gruppo di protezione ricevono ed elaborano gli eventi di input – ai nodi attivi viene data anche la responsabilità di notificare periodicamente i nodi di riserva con i cambiamenti di stato che si sono verificati
  - se si verifica un guasto in un nodo attivo, uno dei nodi di riserva lo può sostituire – ma solo dopo che il suo stato è stato opportunamente aggiornato/sincronizzato
  - la disponibilità può essere aumentata forzando periodicamente lo scambio attivo-riserva – ad esempio, giornalmente, dopo ogni ora, oppure anche dopo ciascuna transazione
  - i nodi attivi hanno dunque la responsabilità aggiuntiva di controllare la sincronizzazione – anche sulla base di un meccanismo di broadcast affidabile
  - è un compromesso (affidabilità/complessità) tra le due tattiche precedenti – *il downtime può essere nell'ordine dei secondi*



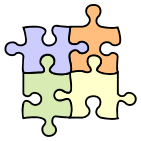
Hot spare, warm spare, cold spare





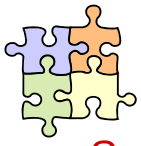
Clustering

- Il *clustering* è una configurazione per la disponibilità di uso comune, e può assumere anche forme diverse da quelle mostrate
  - le tecniche di clustering prevedono di usare più nodi per erogare attivamente un certo servizio
    - è presente anche un elemento di bilanciamento del carico, che ha la responsabilità di assegnare le diverse richieste al servizio (da parte di molteplici client) ai nodi del cluster – anche con lo scopo di sostenere la scalabilità
    - per sostenere la disponibilità, vengono di solito utilizzati anche meccanismi per sincronizzare lo stato tra i nodi del cluster
  - l'obiettivo di queste tecniche è dunque sostenere sia disponibilità che scalabilità
  - considereremo il clustering nel contesto degli stili architetturali per sistemi distribuiti



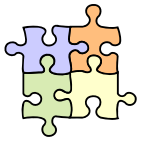
Recover from faults (4)

- *Rollback*
  - un *checkpoint* è la registrazione di uno stato consistente che avviene periodicamente oppure in risposta ad eventi specifici
  - un *log* (delle transazioni) è la registrazione di tutti i cambiamenti di stato avvenuti dopo l'ultimo checkpoint
  - questa tattica ha lo scopo di sostenere il ripristino dello stato del sistema, ad es., sulla base di checkpoint e log delle transazioni – e viene utilizzata in combinazione con le tattiche per la ridondanza



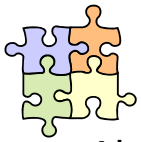
Recover from faults (5)

- *Software upgrade*
  - questa tattica ha l'obiettivo di effettuare aggiornamenti al codice eseguibile – se possibile, senza nessuna interruzione di servizio
  - ad esempio, l'aggiornamento viene effettuato dapprima su un nodo di riserva, che poi (dopo una risincronizzazione dello stato) sostituisce il corrispondente nodo attivo
  - alcuni ambienti di esecuzione offrono delle modalità di aggiornamento online più sofisticate



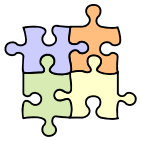
Recover from faults (6)

- *Retry*
  - si assume che il guasto sia transitorio e che riprovare ad eseguire l'operazione possa avere un esito positivo
- *Ignore faulty behavior*
  - si assume che la segnalazione del guasto sia spuria – ad esempio, per evitare che un attacco DoS abbia successo
- *Degradation*
  - mantiene la disponibilità delle funzioni del sistema più importanti e critiche – ma non le funzioni meno importanti
  - possibile solo se i guasti che si sono verificati hanno l'impatto di ridurre le funzionalità del sistema in modo graduale
- *Reconfiguration*
  - il ripristino cerca di riassegnare responsabilità agli elementi sopravvissuti ai guasti



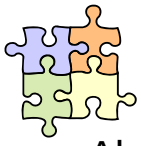
Recover from faults (7)

- Alcune tattiche di ripristino dai guasti sono relativi alla reintroduzione di componenti che si sono guastati – che vengono reintrodotti dopo essere stati "corretti"
- *State resynchronization*
  - le tattiche di ridondanza attiva e passiva richiedono che i vari nodi attivi e di riserva siano tutti continuamente nello stesso stato
  - pertanto, prima di reintrodurre in servizio un nodo, è necessario verificare la correttezza del suo stato oppure ripristinarlo
- *Shadow*
  - un nodo in precedenza fallito (ed è stato reintrodotto) viene temporaneamente eseguito in "modalità ombra" (come riserva) per poter verificare la correttezza del suo comportamento, prima di essere reintrodotto come attivo



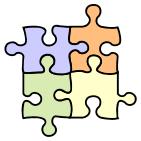
Recover from faults (8)

- *Escalating restart*
  - una tattica di reintroduzione che consente di eseguire in modo granulare il riavvio dei sistemi ripristinati da guasti – per minimizzare il livello dei servizi su cui il guasto ha impatto
  - ad esempio, ad un livello vengono riavviati solo alcuni processi, ma ad un altro livello viene riavviato anche il sistema operativo
- *Non-stop forwarding*
  - le funzionalità sono divise in più parti – in caso di degradazione del servizio, alcune funzionalità vengono lasciate sempre attive – mentre altre vengono interrotte, e quindi vanno riavviate al momento del ripristino
  - concetto che ha origine nella progettazione dei router – l'aggiornamento delle rotte può essere interrotto e riavviato, mentre viene continuato l'instradamento di pacchetti su rotte note



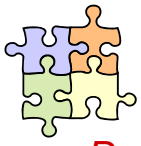
- Preventing faults (1)

- Alcune tattiche per la prevenzione dei guasti – per evitare che si verifichino e di doverli rilevare e gestire
- *Removal from service*
  - questa tattica rimuove temporaneamente un componente del sistema dalle operazioni per fargli svolgere attività che possono mitigare possibili fallimenti di sistema
  - ad esempio, effettuare un reboot per fare in modo che sfridi o perdite nell'allocazione di memoria causino dei fallimenti



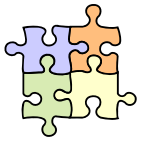
Preventing faults (2)

- *Transactions*
  - i servizi ad alta disponibilità sono solitamente basati su una semantica transazionale – una transazione è una sequenza di passi elementari che viene complessivamente considerata un'operazione atomica – da svolgere oppure annullare completamente
  - consente di prevenire inconsistenze nello stato del sistema, nel caso di fallimento di uno dei passi elementari
  - ad esempio, 2PC nel caso di transazioni distribuite



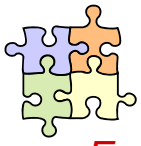
Preventing faults (3)

- *Predictive model*
  - un modello predittivo, usato in combinazione con un monitor, viene utilizzato per monitorare lo stato di salute di un processo – il processo può essere cancellato e poi ricreato se viene rilevata un'anomalia nel comportamento del processo (ad es., aumento del tempo di risposta), che indica la probabilità di un futuro guasto



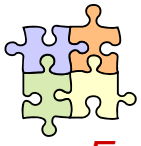
- Exceptions

- Le *eccezioni* sono un meccanismo dei linguaggi di programmazione per gestire situazioni “eccezionali” rispetto al flusso di esecuzione normale del programma – dunque particolarmente adatto alla gestione di guasti
  - è dunque in genere opportuno applicare le eccezioni nello sviluppo di software con requisiti di disponibilità
  - ci sono diverse tattiche per l’applicazione delle eccezioni
    - che contestualizzano l’applicazione delle eccezioni alle tre categorie di tattiche introdotte in precedenza



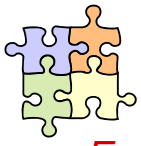
Exceptions (1)

- *Exception detection* – per detect faults
  - il rilevamento di eccezioni si riferisce al rilevamento di condizioni del sistema che, appunto, devono alterare il normale flusso di esecuzione
  - ad esempio, eccezioni di sistema (es., divisione per zero, errore nell’accesso alla memoria), oppure eccezioni legate alla verifica di codici di controllo nei dati scambiati tra componenti, della correttezza delle risposte calcolate da un algoritmo, a possibili timeout



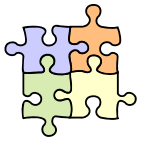
Exceptions (2)

- *Exception handling* – per recover from faults
  - la gestione di eccezioni può essere usata per il ripristino da guasti – notificati appunto come eccezioni
  - il tipo, l'origine e la causa dell'eccezione possono essere usate per selezionare un'opportuna modalità di ripristino
- *Increase competence set* – per detect faults & recover from faults
  - la competenza di un programma è l'insieme di stati in cui il programma ha competenza per operare – ad es., la divisione per zero è, di solito, al di fuori della competenza dei programmi
  - in alcuni casi è possibile estendere l'insieme di competenza, facendo gestire al programma più casi in modo normale

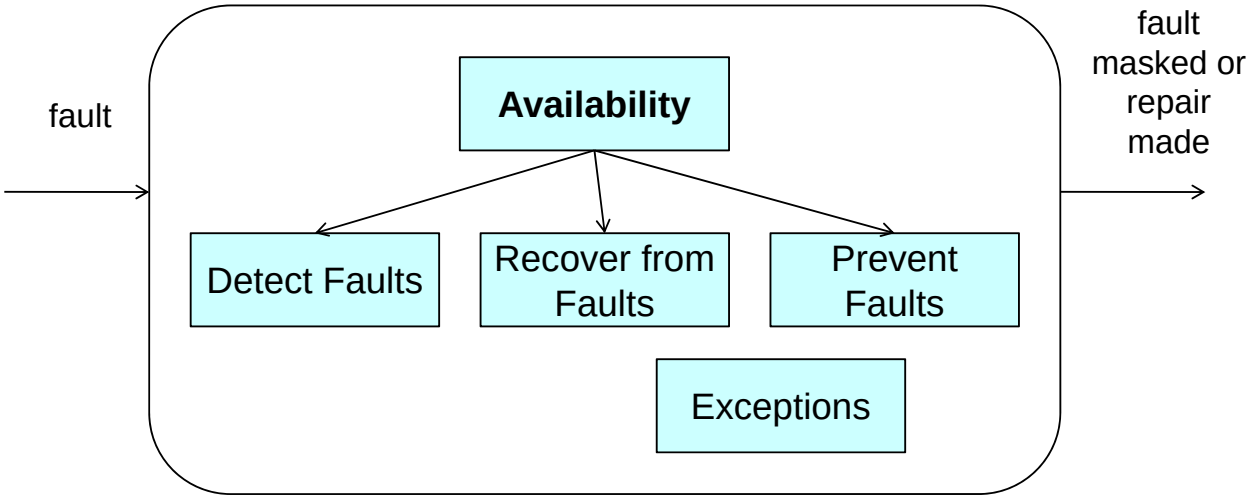


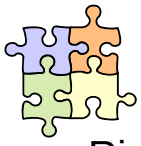
Exceptions (3)

- *Exception prevention* – per preventing faults
  - questa tattica ha lo scopo di prevenire il verificarsi di eccezioni
    - ad esempio, usare un linguaggio che prevede un insieme di controlli statici forti – oppure, l'uso di tipi che controllano tutti gli accessi tramite puntatori, per evitare riferimenti appesi



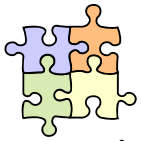
- Tattiche per la disponibilità – sintesi





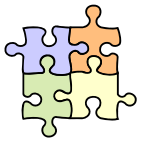
Tattiche per la disponibilità – discussione

- Riguardo alle tattiche per la disponibilità
  - l'applicazione di una tattica può richiedere anche l'applicazione di altre tattiche – in altre categorie
    - ad esempio, le tattiche di ripristino da guasti (hot spare, warm spare e cold spare) vanno sicuramente applicate insieme a tattiche di rilevamento di guasti – per consentire un ripristino automatico
  - l'applicazione di una tattica non impedisce l'applicazione di tattiche – anche nella stessa categoria
    - ad esempio, la realizzazione di un servizio altamente disponibile può richiedere sia una ridondanza hot spare (nello stesso sito) che una ridondanza warm spare (in un sito remoto di disaster recovery)

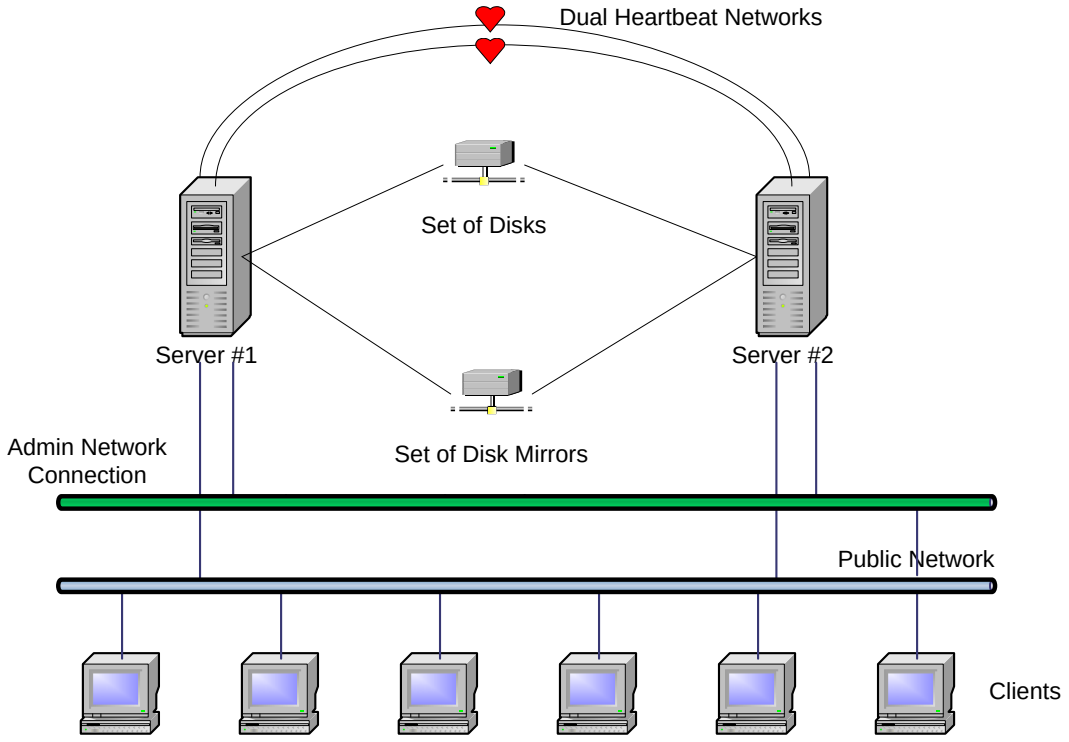


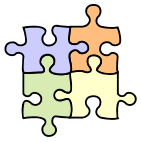
Tattiche per la disponibilità – discussione

- Riguardo alle tattiche per la disponibilità
  - l'applicazione di una tattica per la disponibilità può essere efficace a fronte di un certo tipo di guasto – ma non lo è necessariamente con tutti i possibili guasti (allo stesso modo)
    - ad es., la replicazione di un server in un rack consente di gestire la rottura del server – ma non scenari come un incendio, un allagamento o un'interruzione di energia elettrica
    - ad es., una certa configurazione potrebbe essere in grado di gestire la rottura di un server in pochi millisecondi – ma la rottura di un'unità di storage in diverse ore
  - molte configurazioni richiedono considerazioni speciali
    - ad es., se un processo è replicato ed è in esecuzione su due nodi "virtuali" – allora è bene che i due nodi "virtuali" siano allocati su due nodi "fisici" distinti

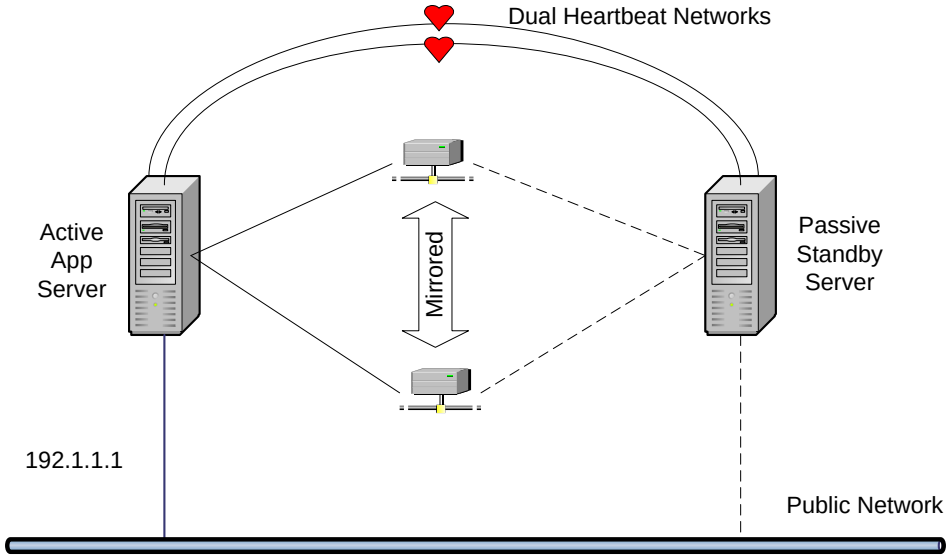


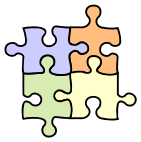
Esempio - un semplice cluster



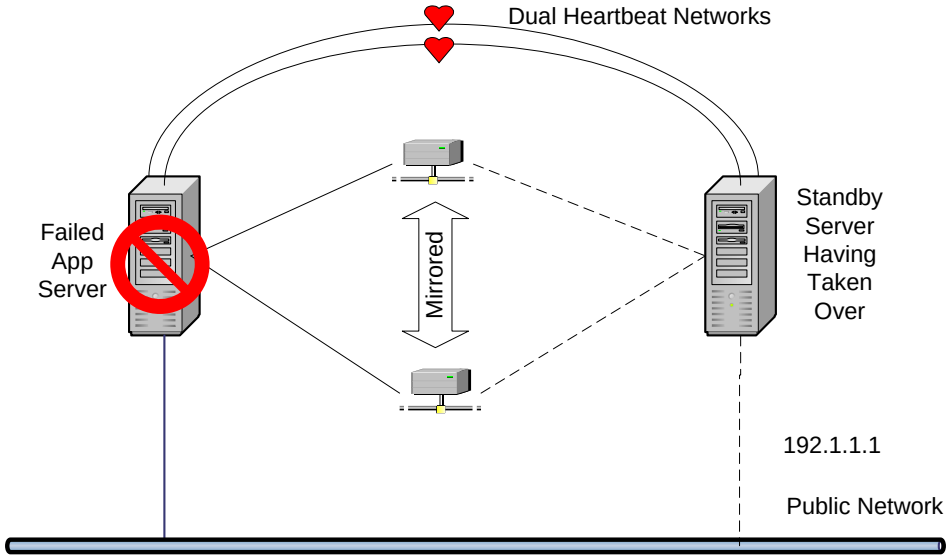


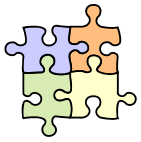
Config. active-passive – prima di un failover



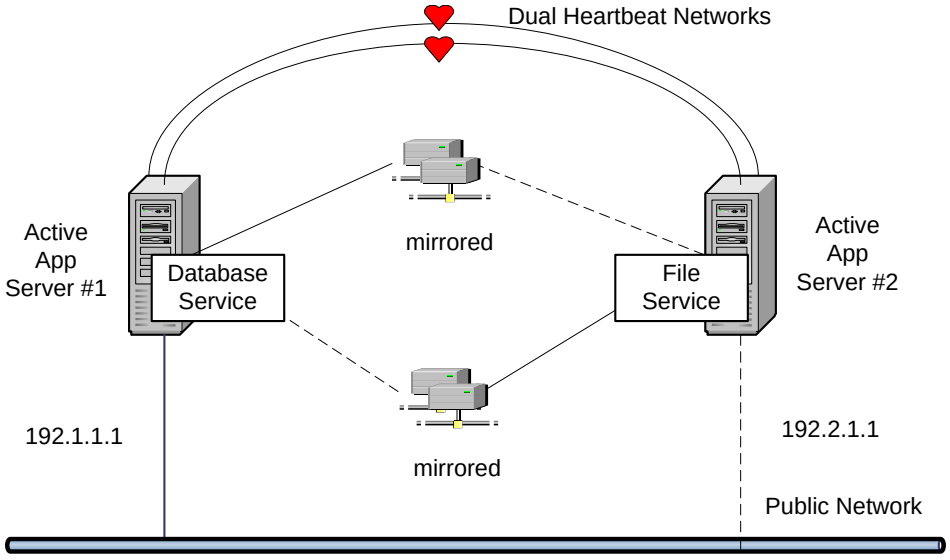


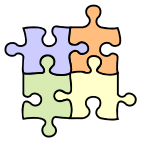
Config. active-passive – dopo un failover



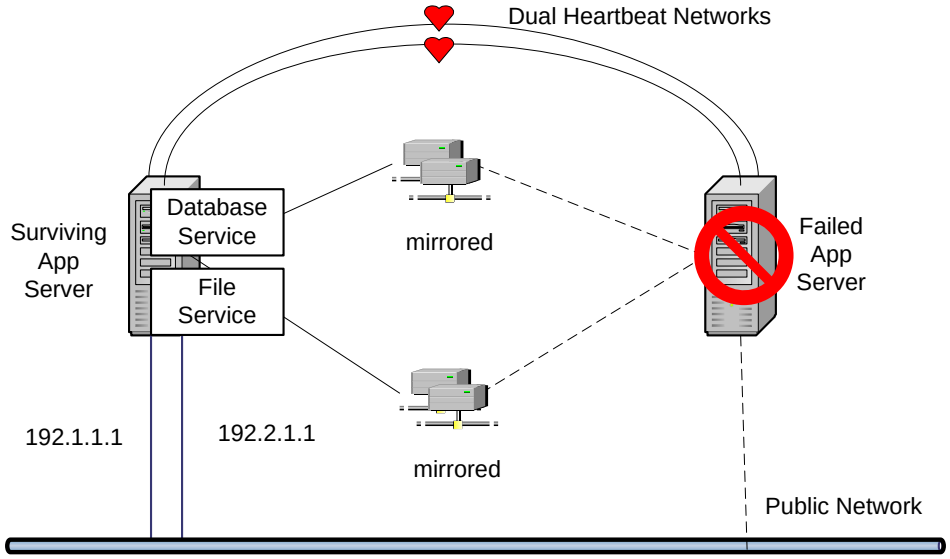


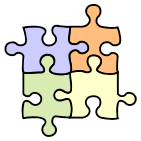
Config. active-active – prima di un failover





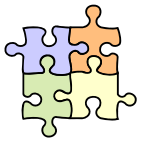
Config. active-active – dopo un failover





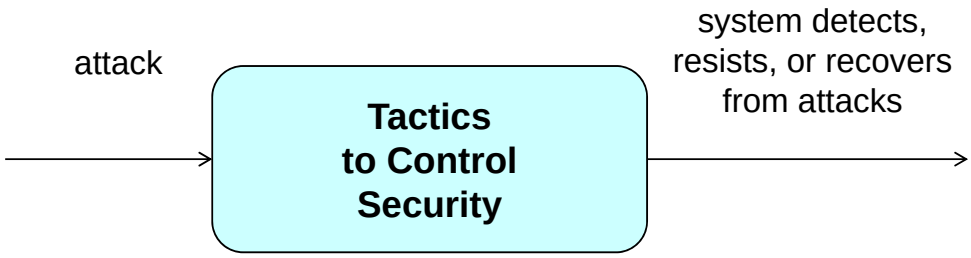
* Tattiche per la sicurezza

- La *sicurezza* è relativa alla capacità del sistema di resistere ad usi non autorizzati – e allo stesso tempo fornire servizi ai suoi utenti legittimi
- In questa trattazione – basata su [SAP/2e] – consideriamo alcune *tattiche per la sicurezza* per controllare alcuni dei seguenti aspetti
  - *confidenzialità/segretezza* – occultamento di informazioni o risorse sensibili (delicate)
  - *integrità* – fidatezza nelle informazioni o nelle risorse
    - integrità dei dati e dei programmi
    - integrità dell'origine – ovvero, dell'identità partecipanti alle transazioni
  - *disponibilità* – la capacità di usare le informazioni o le risorse desiderate
  - *accountability* – sapere chi ha avuto accesso ad informazioni o risorse

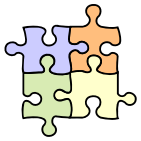


Tattiche per la sicurezza

- Obiettivo delle tattiche per la sicurezza che sono descritte in questa trattazione

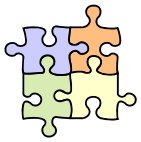


- un *attacco* è un tentativo di rompere la sicurezza di un sistema



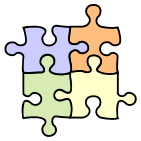
Categorie di tattiche per la sicurezza

- Tre categorie principali di tattiche per la sicurezza – solitamente da usare in combinazione
  - resistere agli attacchi – *resisting attacks*
    - simile all'uso di una serratura – obiettivo è tenere fuori i "cattivi"
  - rilevare attacchi – *detecting attacks*
    - simile all'uso di un allarme – i "cattivi" possono entrare, ma non possono fare danni
  - ripristino da attacchi – *recovery from attacks*
    - simile all'uso di un'assicurazione – i "cattivi" possono entrare e fare danni, ma i danni possono essere riparati



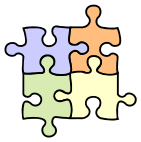
- Resisting attacks (1)

- Tattiche per resistere agli attacchi
- *Authenticate users*
  - l'autenticazione è assicurarsi che l'utente o sistema remoto è chi dichiara di essere
  - ad es., password, certificati, ...
- *Authorize users*
  - le autorizzazioni sono usate per verificare che un utente autenticato abbia degli opportuni diritti di accesso nei confronti delle risorse (dati o servizi) che intende utilizzare
  - ad es., acl (access control lists), autorizzazioni per utenti, gruppi di utenti o ruolo, ...



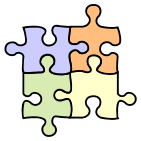
Resisting attacks (2)

- *Maintain data confidentiality*
  - protezione dei dati da un accesso non autorizzato – solitamente basata su qualche forma di cifratura
  - ad es., cifratura simmetrica o asimmetrica, ...
  - attenzione al costo temporale della cifratura
    - è in contrapposizione con *Reduce computational overhead*
- *Maintain integrity*
  - i dati devono essere consegnati come sono
  - solitamente basato sull'uso di informazioni di controllo ridondanti, certificati, cifratura, ...



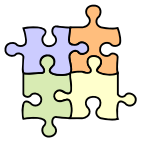
Resisting attacks (3)

- *Limit exposure*
  - i servizi sono essere distribuiti su più host, in modo tale che un attacco possa colpire i servizi solo in modo parziale o in numero limitato
  - infatti gli attacchi sono solitamente rivolti verso singole debolezze del sistema
- *Limit access*
  - un firewall può restringere l'accesso sulla base della sorgente del messaggio o la porta di destinazione
    - consente di limitare l'accesso nei confronti di client non conosciuti
  - per limitare l'accesso da client conosciuti (ad es., dal proprio web server pubblico) è possibile usare configurazioni come, ad es., una zona demilitarizzata (DMZ)



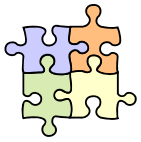
- Detecting attacks

- Il rilevamento di attacchi viene solitamente fatto mediante un sistema di rilevamento delle intrusioni
  - ad esempio, che confronta il traffico dei messaggi con pattern di dati storici di attacchi conosciuti

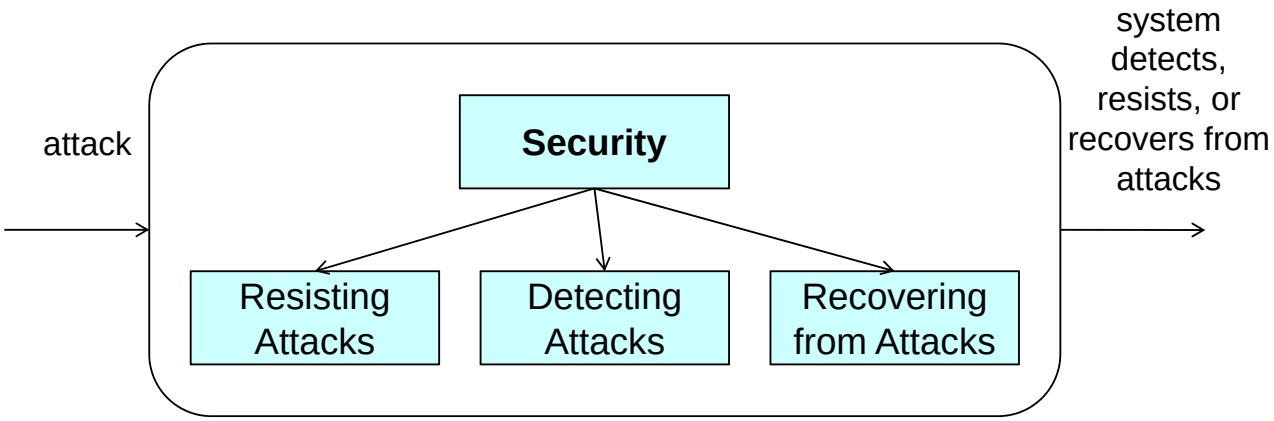


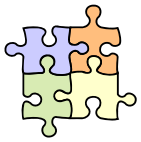
- Recovering from attacks

- Due categorie di tattiche per il ripristino da attacchi
  - tattiche che hanno lo scopo di ripristinare lo stato del sistema
    - vedi le tattiche corrispondenti per la disponibilità
  - tattiche che hanno lo scopo di identificare l'origine dell'attacco – a fini punitivi
    - solitamente sulla base di informazioni di auditing, ovvero la registrazione degli accessi al sistema e degli utenti che li hanno effettuati

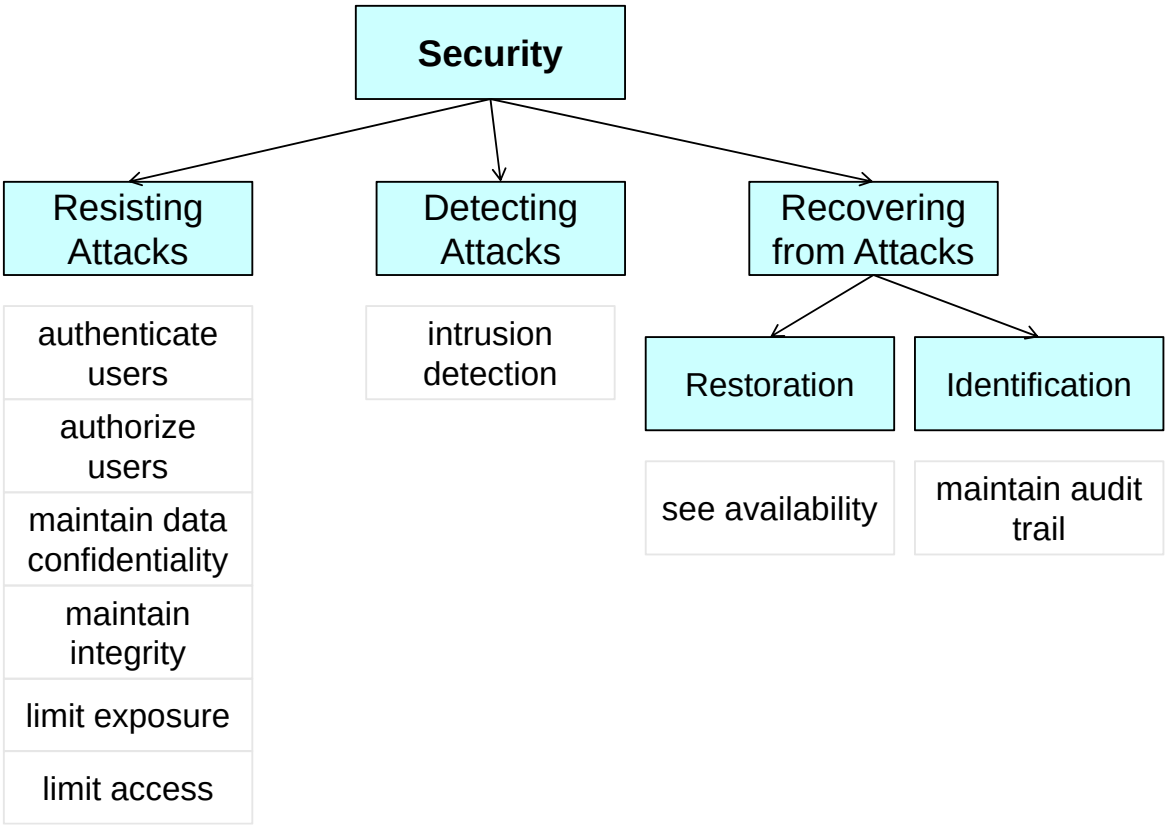


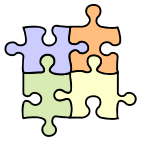
- Tattiche per la sicurezza – sintesi





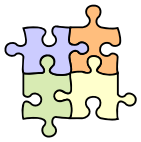
Tattiche per la sicurezza – sintesi





* Discussione

- Sono state presentate alcune tattiche architetturali – nel contesto del controllo di alcuni attributi di qualità
  - applicare una tattica vuol dire prendere una decisione di progetto per controllare un certo attributo di qualità
    - questo ha impatto sull'architettura – ovvero, sulla scelta degli elementi, delle loro responsabilità, o di come sono messi in relazione – e su come l'architettura sostiene le qualità
  - la presentazione è stata qualitativa e informale
    - sono talvolta disponibili modelli che descrivono l'effetto quantitativo dell'applicazione delle tattiche
    - è inoltre possibile modellare queste decisioni di progetto – anche sulla base dell'applicazione di scenari
  - sono stati mostrati alcuni compromessi che occorre valutare nel controllo congiunto di più attributi di qualità
  - esistono altri attributi di qualità e altre tattiche



Discussione

- La progettazione di un'architettura richiede l'applicazione di una collezione di tattiche, per realizzare una *strategia architetturale*
  - l'approccio (solitamente di compromesso) adottato al fine di raggiungere gli obiettivi complessivi di qualità del sistema
- Un altro approccio fondamentale per la progettazione dell'architettura è basato sugli stili architetturali
  - uno stile architetturale può essere basato sull'applicazione di un certo numero di tattiche
  - nel seguito del corso questa affermazione sarà esemplificata nel contesto dello studio degli stili architetturali